Chapter 7: Simulation Techniques and Simulators

Before You Start

What you are assumed to know. You write software for a living, you read a function signature and know what its arguments and return value commit it to, and you remember basic probability from a first course. You have read the previous six chapters. From Chapter 3 you have three words: a **model** is a description of how something behaves, and it does not run; a **simulation** is one execution of a model over time from a starting state; a **simulator** is the software that performs simulations. From Chapter 4 you have the seven blocks, in particular the **simulators** block – the block that executes models – and the **models** block, which that chapter reduced to one interface: given a state and a horizon, return a predicted state. From Chapter 5 you have the tuple, the sampling interval T_s, the event time, the hot, warm and cold paths, and the age of twin. From Chapter 6 you have the five **model types** – lumped first-principles, discrete-state and discrete-event, high-fidelity field, reduced-order and surrogate, and probabilistic with Bayesian updating – each with its step function, the four criteria a model is judged on, and the choosing table. The offshore wind turbine of that chapter is this chapter's running example too, with the same numbers. From Chapter 2 you have the plant rig, which reappears once, in the one section where it is allowed.

What you are not assumed to know. Numerical analysis, queueing theory, statistics beyond a first course, and any simulation tool. Every mathematical result this chapter leans on is recalled in one clause and pointed at the section of the mathematics appendix that builds it: *From an Equation to a Loop* for what a solver is and what Euler's step does; *Better Solvers* for Heun's step, the implicit step and adaptive stepping; *Stiff Equations for Digital Twins* for why a step size is forced; *Many Quantities at Once* for the state-space form, dimension and modes; *Probability and Distributions* and *State Machines with Probabilities* for the Gaussian, Bayes' rule and the belief that leaks away as a chain is predicted forward; and *What a Solver and a Simulator Do* for the event queue, the cost of a run and the real-time factor. If a paragraph below feels like it is skipping a step, the appendix section named beside it is where the step is.

What this chapter covers. Chapter 6 said what goes in Chapter 4's model slot. This chapter says what executes it. It first takes a simulator apart into four pieces – an engine, a step, a clock and an input-output contract – and gives four criteria a simulator is judged on, parallel to the four that chapter gave a model. It then walks the five model types again, in the same order, as five **execution problems**: running rate rules, running an event queue, running both at once, running one model many times, and running a field model that cannot be run at all inside a twin – followed by the section on how the field model's answer is bought back cheaply. Three sections then make the chapter usable: time, where the real-time factor stops being a definition and becomes a placement decision; the concrete interface a twin calls a simulator through, with five run modes; and a choosing table with four questions and the numerical checks that make a run trustworthy. The rig, three faded examples, a summary, exercises and further reading close the chapter.

What this chapter deliberately does not cover.

- **Co-simulation, its standard interfaces, and distributed or federated simulation.** Running two or more simulators together, exchanging values at agreed instants, is a large and well-standardised subject, and it belongs with the standards it rests

on: the Functional Mock-up Interface and the High Level Architecture live in the platforms and standards chapter of Part III, and co-simulation is taught there. This chapter is about one simulator, and says nothing further about joining two.
- **The mathematics of any solver.** Recalled in one clause each, with a pointer to the appendix section that builds it.
- **State estimation, the Kalman filter and the correction step.** The state and estimator chapter's. This chapter produces the prediction that an estimator corrects, and stops at the handover.
- **Whether a run deserves to be believed.** The credibility chapter's. This chapter covers numerical verification only – did the arithmetic converge, is the answer the model's and not the step size's – and hands validation on in one sentence.
- **Model updating and parameter re-estimation.** Chapter 6 kept them.
- **Neural, hybrid physics-plus-network and generative surrogates.** Parked for the AI chapter; named here only as parked.
- **Any simulation tool's syntax.** Named where a published system used one, taught nowhere.

Learning objectives. By the end of this chapter you will be able to:

1. **Name** the parts of a simulator and **state** what a twin asks of one.
2. **Choose** a solver and a step size for a rate-rule model and **defend** the choice against stiffness and a time budget.
3. **Decide** whether a model needs a solver, a scheduler or both, and **say** what a mode switch costs.
4. **Design** an ensemble run – how many replications, which seeds, and what the output is.
5. **Compute** a run's real-time factor and **place** the simulator as embedded, edge or batch.
6. **State** what a run's provenance must record and **name** the numerical checks that make a run trustworthy without claiming it valid.

Every section that follows serves at least one of those six, and the exercises at the end say which.

What a Simulator Is, Architecturally

Chapter 4 gave the models block one interface – give me a state and a horizon, return a predicted state – and said nothing about what sits behind it. Behind it is a simulator, and a simulator is four things. Figure 1 draws them.

The engine owns the loop. It holds the current simulated time and the current state, decides when to stop, decides when to record, and calls the step. Nothing else in the simulator knows what time it is.

The step advances the model once. It comes in exactly two kinds, and the whole of this chapter is a consequence of that fact. A **solver** takes a state, asks the model for its rate, and returns the state an interval h later; the appendix's *From an Equation to a Loop* is a solver with h chosen once, up front. An **event scheduler** takes a priority queue of pending events, pops the earliest, advances the clock to its timestamp, runs its handler, and pushes whatever the handler schedules; the appendix's *What a Solver and a Simulator Do* writes the loop out

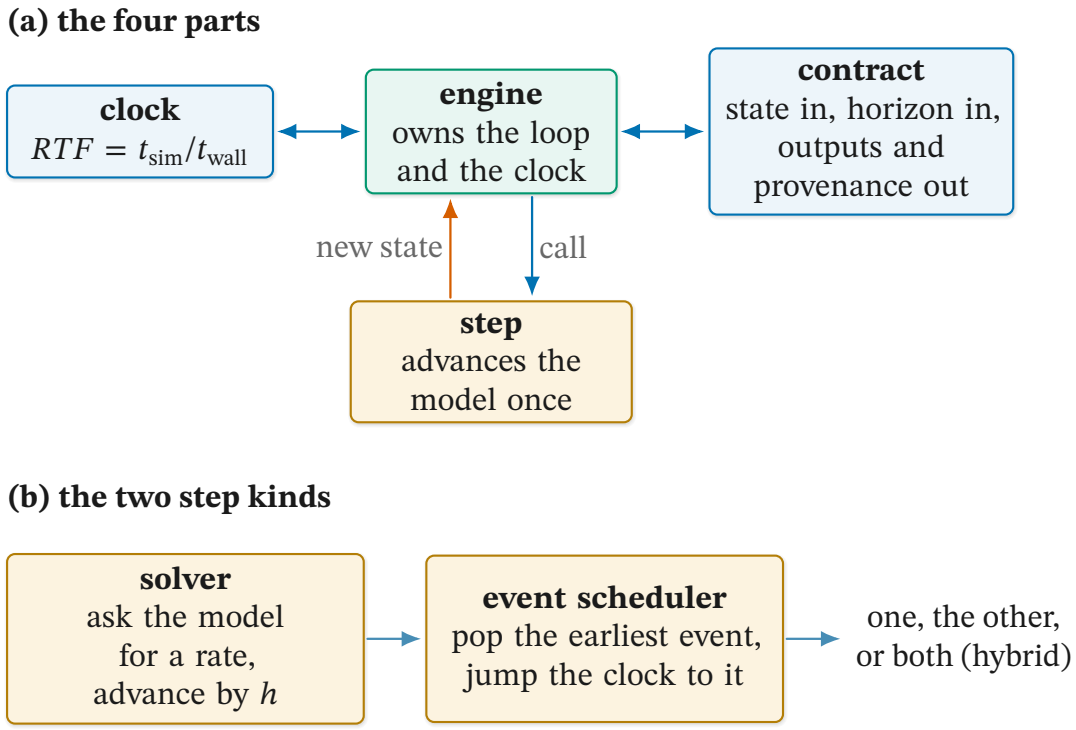


Figure 1: The four parts of a simulator, and the two interchangeable step kinds. Panel (a): the engine owns the loop and the clock, calls the step, and writes outputs through the contract. Panel (b): the step is either a solver, which asks the model for a rate and advances by an interval, or an event scheduler, which pops the earliest pending event and advances the clock to it.

in seven lines. A simulator has one, the other, or – the Hybrid Simulation section below – both, with one driving the other.

The clock decides what one second of simulated time costs in wall-clock time. It is the part a reader new to simulation does not expect to exist, because in an ordinary program there is only one clock. In a simulator there are two, they run at different rates, and the ratio between them is the **real-time factor** *RTF* the appendix defines: simulated time covered per unit of wall-clock time. The Time section below is entirely about that ratio.

The input-output contract is Chapter 4's interface, written out: how the initial state gets in, where the inputs over the horizon come from, what the outputs look like on the way out, and what the run records about itself. The Driving a Simulator from a Twin section below gives it as a function signature.

The four criteria

Chapter 6 judged a model on four criteria. A simulator is judged on four of its own, and they are about execution rather than about representation.

1. **It finishes inside the decision's time budget.** Chapter 1 gave every value pattern a latency; a simulator that answers after the decision was due has not answered. This is the *RTF* question, and it is the one criterion that can be computed in advance rather than argued about.
2. **It is restartable from a state the twin supplies.** A simulator written for design

runs starts from a designed initial condition and runs for hours. A simulator inside a twin is handed the state block's current belief every cycle and asked for the next few minutes. If setting the state costs more than the run, the simulator is the wrong one, however fast its inner loop is.

3. **It is deterministic enough to be compared with itself.** Two runs from the same state with the same inputs and the same seed must give the same answer, or the difference between yesterday's prediction and today's says nothing about the asset. This is weaker than bit-reproducibility and stronger than "close enough": what it rules out is a simulator whose answer depends on thread scheduling, on an unseeded random source, or on wall-clock time leaking into the model.
4. **It says what it ran.** The model and its version, the solver and its settings, the initial state, the inputs, the seed, and what the run cost. A number a twin cannot reproduce is a number a twin cannot defend, and the Driving a Simulator from a Twin section makes this a record the run returns rather than a logging convention.

Criteria 3 and 4 are software-engineering criteria, not numerical ones, and they are the two most often missing from a simulator that was written for a design office and then pointed at an asset.

The simulator is usually something you are handed

Chapter 4 observed that a digital twin framework names the blocks it implements and implements the loop rather than the solver. That is worth restating here, because it decides what a software engineer's job on this chapter actually is. You will rarely write a solver. You will choose one, configure it, work out whether it fits the budget, wire the twin's state into it, and be responsible for the answer. The executable-twin literature names the artefact this produces: a self-contained executable digital behaviour of an asset, to be used by anyone at any point in the lifecycle, which requires the model to be fast enough – often real-time – and in most cases requires the model order reduction the Making an Expensive Simulator Cheap section below describes, with the packaged model carrying its execution engine with it and an explicit link back to the parent twin that keeps the two consistent [1]. A simulator, in other words, is increasingly shipped rather than written; the four criteria above are what you check it against on arrival.

Continuous-Time Simulation: Running the Rate Rules

The execution problem of Chapter 6's **lumped first-principles** model type. The model is a handful of rate rules; the simulator is a solver; the only real decision is the step, and it is worth more thought than it usually gets.

Four steps, priced

The appendix's *Better Solvers* builds these; this section prices them. The price is in **rate evaluations per step**, because for a small model almost all the work is evaluating the right-hand side.

Table 1: The four step kinds a twin's solver will offer, priced in rate evaluations. The error column is how the error at the end of a run shrinks when the step is halved; the stability column is what forces a short step whether or not the accuracy needs one.

Step	What it does	Rate evaluations per step	Error falls as	Stable for
explicit Euler	uses the rate at the start of the interval	1	h	h below about twice the fastest time constant
Heun	averages the rate at both ends, guessing the end with an Euler step	2	h^2	roughly the same bound
classical Runge-Kutta	samples the rate four times across the interval	4	h^4	a somewhat larger bound
implicit Euler	uses the rate at the end, so each step solves a small equation	1, plus an equation solve	h	any h
adaptive	estimates its own error and changes h as it goes	varies	to a tolerance you set	it chooses

Read Table 1 as an engineering menu. Going from Euler to Runge-Kutta costs four times the work per step and buys a step that can be roughly ten times longer for the same accuracy – a net win, and the reason nobody uses fixed-step Euler in production. Going from explicit to implicit costs an equation solve per step and buys the removal of the stability column, which is the only thing that helps when the model is stiff. Going adaptive replaces "choose h" with "choose a tolerance", which is a better question to ask a domain expert; solvers of this kind take long steps where the dynamics are slow and short ones where they are fast or where the instant something happens has to be located properly, and they are the family customarily used on stiff systems [2]. What a tolerance costs is measurable: one study sweeping a general-purpose solver's relative tolerance from 10^{-2} down to 10^{-12} on the same model reports the step count rather than the elapsed time, because on a small model the elapsed time is buried in operating-system noise – and notes that on a larger model the same effect does appear in the time [3]. That is the honest way to measure a solver setting, and it is worth copying.

Stiffness, on the turbine's own numbers

Chapter 6 wrote the turbine's rotor and gearbox oil as two rate rules:

$$J\,\frac{d\omega_{\mathrm{r}}}{dt} = Q_{\mathrm{a}} - Q_{\mathrm{g}} \tag{1}$$

$$C_{\mathrm{th}}\,\frac{dT_{\mathrm{oil}}}{dt} = P_{\mathrm{loss}} - k_{\mathrm{th}}\,(T_{\mathrm{oil}} - T_{\mathrm{amb}}) \tag{2}$$

with $J = 6.0 \times 10^{6}$ kilogram square metres, $C_{\mathrm{th}} = 2.0 \times 10^{6}$ joules per kelvin, $k_{\mathrm{th}} = 1.0 \times 10^{3}$ watts per kelvin and $T_{\mathrm{amb}} = 12$ degrees. Equation 1 has no damping term in it at all as Chapter 6 wrote it, and Equation 2 has one; that asymmetry is where this subsection starts. The oil's time constant is $C_{\mathrm{th}}/k_{\mathrm{th}} = 2000$ seconds: left alone it closes a third of its gap to the steady value every half hour. That is the slow process.

Chapter 6 took the generator torque Q_{g} as an input, which is right for a model. A *running* turbine sets it: below rated wind the controller holds the tip-speed ratio by commanding torque proportional to the square of the rotor speed, $Q_{\mathrm{g}} = K\,\omega_{\mathrm{r}}^{2}$. Chapter 6's operating point – $Q_{\mathrm{g}} = 1.44 \times 10^{6}$ newton metres at $\omega_{\mathrm{r}} = 1.70$ radians per second – fixes $K = 4.98 \times 10^{5}$, and closing that loop is what makes the pair stiff. The rotor now settles rather than drifting: it runs to the speed at which aerodynamic torque equals generator torque, $\omega_{\mathrm{r}} = \sqrt{Q_{\mathrm{a}}/K} = 1.736$ radians per second for $Q_{\mathrm{a}} = 1.50 \times 10^{6}$, and it gets there with a time constant of

$$\tau_{\mathrm{r}} = J/(2K\omega_{\mathrm{r}}) = 6.0 \times 10^{6}/(2 \times 4.98 \times 10^{5} \times 1.736) = 3.5\ \mathrm{s} \tag{3}$$

Equation 3 is three and a half seconds against the oil's two thousand. That ratio, near six hundred to one, is exactly the appendix's definition of a **stiff** system: a fast process and a slow one in the same model, the fast one over almost immediately, so that stability rather than accuracy sets the step. An explicit solver is stable only for a step below about twice the fastest time constant – here about seven seconds – and the slow state is the one the twin actually cares about, arriving once every ten minutes with the SCADA reading – and it arrives at that rate on purpose, because the diagnostic twin of a real floating turbine chose the gearbox temperatures as its modelled variables precisely on the grounds that temperatures change slowly and need only low measurement rates [4]. So an explicit solver steps the oil rule six hundred times per reading to move it a fraction of a degree, not because the oil needs it but because the rotor would explode if it did not. The state the twin cares about is the slow one; the step is set by the fast one; and nothing about the decision justifies the cost.

The same ten minutes, three ways

Take one SCADA interval, $h = 600$ seconds of simulated time, from Chapter 6's state $\omega_{\mathrm{r}} = 1.70$, $T_{\mathrm{oil}} = 48.0$. Table 2 runs it three ways.

Table 2: One SCADA interval of the turbine's two-state model, stepped three ways. The middle row is what stiffness looks like when it is ignored: a step chosen for the quantity you care about, destroyed by the one you do not.

Way	ω_r after 600 s	T_{oil} after 600 s	Rate evaluations	Verdict
explicit, $h = 1$	1.736	49.04	1,200	correct, and cheap enough
explicit, $h = 600$	7.70, then diverges	49.20	2	unstable; the oil is fine and the rotor is nonsense
implicit, $h = 600$	1.735	48.92	2, plus two small equation solves	stable and usable; the rotor lands on its steady value

The middle row is worth doing by hand, because it is the failure this section exists to prevent. At $\omega_r = 1.70$ the torque imbalance is $Q_a - Q_g = 6.0 \times 10^4$ newton metres, so the rotor's rate is 0.010 radians per second per second and a six-hundred-second explicit step lands on $\omega_r = 7.70$ – a turbine spinning four and a half times too fast. Evaluate the controller there and $Q_g = K \times 7.70^2 = 2.95 \times 10^7$ newton metres, so the next rate is $(1.50 \times 10^6 - 2.95 \times 10^7)/(6.0 \times 10^6) = -4.67$, and the step after that puts the rotor at minus two thousand eight hundred radians per second. The simulation has not become inaccurate; it has become meaningless, in two steps, while returning numbers the whole way.

The implicit row solves, per step, the small equation $\omega_r = 1.70 + 10^{-4}(1.50 \times 10^6 - 4.98 \times 10^5\,\omega_r^2)$, whose positive root is 1.735 – the steady value, reached in one step, from a solver that never became unstable. The oil is even easier: the implicit step rearranges to $T_{oil} = (48 + 0.3 \times 12 + 12)/1.3 = 48.92$, against a true value of 49.04 and an explicit $h = 600$ answer of 49.20. Both are within a fifth of a degree; the choice between them was never about the oil.

Algebraic constraints, and the loops they make

Not every line of a model has a rate in it. Chapter 6's turbine had two that did not: the gear ratio, $\omega_g = n_{gear}\,\omega_r$, and the referral of the fast-shaft torque back to the rotor. An **algebraic constraint** is an equation that must hold at every instant, and a solver handles it by solving it after each step rather than integrating it.

That is straightforward while the constraints form a chain. It stops being straightforward when they form a cycle – when an output feeds back to its own input within one step, with no rate in between. That is an **algebraic loop**, and a solver cannot evaluate it in any order: there is no first equation. The multi-paradigm modelling literature is precise about where these come from. Equations written without saying which variables are known and which unknown – the equation-based style – have their **computational causality** assigned afterwards, and where causality cannot be assigned without conflict, the model needs a

special implicit integration routine whose equations are solved by an iteration scheme inside the numerical integration method; loops of this kind are named there as zero-order causal paths [5]. In practice: your tool will either report the loop and refuse, or break it by iterating, and the second costs an inner solve per step that nobody budgeted for. If a model that should run in microseconds runs in milliseconds, an algebraic loop is the second thing to suspect, after stiffness.

Acausal tools, and what their translation step buys

There are two ways to write the six lines of Chapter 6's turbine. The **causal** way is the one in that chapter: a `step` function, assignments in an order you chose, the outputs computed from the inputs. The **acausal** or **equation-based** way writes the same physics as equations without choosing which side is the unknown, and lets a compiler work out the order – so a resistor is written $v = iR$ once and used in both directions, and a model is assembled by connecting components rather than by wiring signals. Modelica is the language this style is usually met in, and it is described exactly that way in the modelling literature: differential-algebraic equations written without specifying the knowns, with causality assigned afterwards to produce something executable [5].

What the translation step buys is composition: a heat exchanger written once works in any circuit. What it costs is a compilation stage between your edit and your run, and a debugging experience conducted in transformed equations rather than in the ones you wrote. In a twin the compiled artefact is the thing that ships: a model-based engineering pipeline of this kind brings Modelica and other formalisms in as compiled units with generated interface code, one pipeline producing the simulation and another the deployed system [6]. The tools also mix. A production building tool chain couples an equation-based Modelica model of the heating, ventilation and control side to a conventional envelope simulator through the slowly-changing surface temperatures rather than the fast room air, precisely so that the interval between the two sides can be long – and long intervals reduce simulation time, particularly for the adaptive solvers that stiff systems demand [2]. Which is this section's argument in one sentence: where you put the boundary decides what step you can take, and the step decides what the run costs.

Consistent initialisation, every cycle

A solver cannot start from an arbitrary state. The algebraic constraints have to hold at the first instant as well as at every later one, so before the first step the simulator solves them for whatever the given state left undetermined. That is **consistent initialisation**, and in a design run it happens once and nobody notices.

In a twin it happens every cycle. The twin hands the simulator the state block's current belief – Chapter 4's `state` block, as corrected by the estimator – and asks for the next ten minutes; ten minutes later it hands over a new belief and asks again. Initialisation is therefore a **per-cycle cost**, not a start-up cost, and a simulator whose initialisation takes longer than its run is one you cannot use, however good its inner loop is. This is the second of the four criteria with a number attached to it. For the turbine's two states it is free: the gear ratio is one multiplication. For a model with a hundred algebraic constraints and a nonlinear solve among them, it is the whole budget, and the usual remedy is to keep the simulator warm between cycles and re-initialise incrementally rather than from cold.

Discrete-Event Simulation: Running the Event Queue

The execution problem of Chapter 6's **discrete-state and discrete-event** model type. The model is a set of modes and a set of things that happen; the simulator is an event scheduler; and the decision that matters is not the step size but *how many times you run it*.

Two ways to advance a clock

The appendix's *What a Solver and a Simulator Do* gives the loop: a **future-event list**, which is a priority queue of pending events ordered by time, and a loop that pops the earliest, jumps the clock to it, runs the handler, and pushes whatever the handler schedules. That is **next-event time advance**. The alternative is **fixed-increment time advance**: step the clock by a constant interval and process whatever fell inside it, which is what a solver does.

Figure 2 draws the difference on one timeline, and why the first exists is arithmetic. Chapter 6's turbine changes mode a few thousand times a year – cut-in, cut-out, curtailments, alarms, repairs. A next-event loop over a simulated year therefore executes a few thousand iterations. A fixed-increment loop at one-second resolution executes $365 \times 24 \times 3600 = 31.5$ million, of which all but a few thousand find nothing to do. The ratio is four orders of magnitude, and it is why a year of a turbine's operational life simulates in milliseconds while a year of its rotor dynamics does not simulate at all.

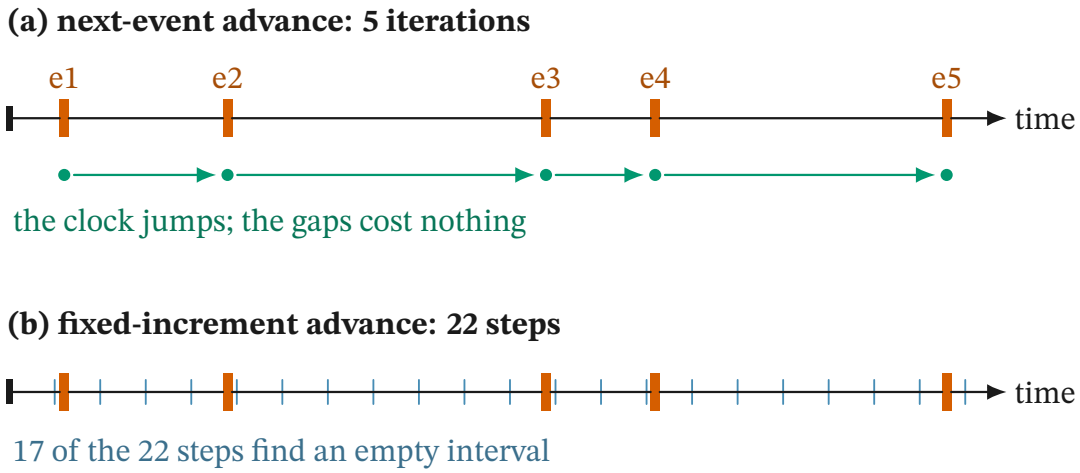


Figure 2: The same five events under the two ways of advancing a clock. Panel (a), next-event advance: the clock jumps to each event in turn, and the gaps cost nothing. Panel (b), fixed-increment advance: the clock steps by a constant interval and most steps find an empty interval.

Fixed-increment advance is not always wrong. It is the right choice when events are dense relative to the interval, when the model is already being stepped by a solver for other reasons – the Hybrid Simulation section below – or when the output has to be sampled on a regular grid anyway. But the default for a model whose state is a mode is the queue.

The event formalism you will meet by its acronym

You will meet the acronym, so here is what it is. The **Discrete Event System Specification** (DEVS) is a formalism that says exactly what an atomic model owes a scheduler, and the modelling literature states it as four things: an **internal transition function** from state to

state, fired when the model's own time runs out; a **time advance function** that says how long the model will sit in its current state, and whose access to the state is read-only; an **external transition function** that fires on an incoming event and sees both that event and the elapsed time since the last transition; and an initial total state [5]. Read as a software engineer: DEVS is an interface with four methods, and any model implementing it can be run by any DEVS scheduler and composed with any other. That is its whole value, and it is why a design method for twinning systems lists choosing between differential equations, DEVS and Petri nets as an explicit design stage rather than an accident of tooling [7]. This book does not teach DEVS; it names it so that the acronym on a tool's front page means something.

One run of a stochastic model answers nothing

This is the central lesson of the section, and it is the one most often skipped by engineers arriving from deterministic simulation.

Chapter 6's vessel-dispatch model has randomness in it: the weather windows open and close at times drawn from a winter distribution. Run its queue once and you get the five-event trace that chapter tabulated, ending in twenty hours and thirty minutes of lost production. Run it again with different weather and you get a different number. The twenty and a half hours is not the answer. It is *one sample* from the answer, and reporting it as the answer is the most common way a simulation study misleads the person who commissioned it. Computer simulation models are widely used to replicate and analyse stochastic systems, and discrete-event simulation is the most common technique for industrial settings [8] – which makes this a very widely available mistake.

Four pieces of machinery follow from it.

Random variate generation, seeds and streams. The simulator's randomness comes from a pseudo-random generator: a deterministic sequence, fixed by a **seed** s, from which variates of the distributions you need are derived. Two properties matter in a twin. First, the same seed gives the same run, which is what makes criterion 3 of the four achievable. Second, a generator can be split into independent **streams**, so that the weather uses one and the repair durations another; without that, adding a vessel changes the weather, because it changes how many numbers were drawn before the weather asked for one, and your comparison is meaningless.

Replication. A **replication** is one complete run with its own seed. The output of a stochastic model is the sample of N replications, and the answer is a summary of that sample – a mean, a spread, a percentile – with the uncertainty of the summary reported beside it.

Warm-up and run length. A model started in an empty state spends its first stretch in a condition the real system is never in: no faults pending, no vessel at sea, no backlog. Statistics collected during that stretch are biased, so the stretch is discarded as **warm-up**. The **run length** is how much simulated time each replication then covers, and it is chosen so that the quantity of interest has been exercised enough times to matter – for the turbine, a whole winter, because the whole question is about winter.

Confidence intervals. Report the answer as an interval. For a mean over N independent replications with sample standard deviation s_N, the usual ninety-five per cent interval is

$$\text{mean} \pm 1.96\, s_N/\sqrt{N} \tag{4}$$

and the half-width in Equation 4 is the number to quote, not the mean alone.

The vessel run, a thousand times

Chapter 6's five-event trace, run a thousand times with the weather windows drawn from their winter distribution rather than fixed. Suppose the sample comes out with a mean lost production of 20.4 hours per fault and a sample standard deviation of 6.0 hours – a spread that wide because the waiting-for-a-window part, nine and a half hours in that chapter's single trace, is sometimes forty minutes and sometimes three days.

The ninety-five per cent interval on the mean is then $1.96 \times 6.0/\sqrt{1000} = 0.37$ hours either side: 20.4 hours, plus or minus about twenty-two minutes. If the planner wants the mean pinned to half an hour – a quarter of an hour either side – rearrange Equation 4:

$$N = (1.96\, s_N/\text{half width})^2 = (1.96 \times 6.0/0.25)^2 = 2213 \tag{5}$$

Two thousand two hundred and thirteen replications, which at a few milliseconds each is a second of compute. Equation 5 is the arithmetic behind every "how many runs?" question in this chapter, and its shape is the shape of the whole subject: the half-width falls as one over the square root of N, so **ten times the precision costs a hundred times the runs**.

Two warnings the arithmetic does not carry. First, it is the interval on the *mean*. If the decision is about the bad winters – the ninety-fifth percentile of lost production, which is what a spares policy is sized on – the same precision needs substantially more replications, because a percentile out in the tail is estimated from far fewer effective observations than a mean. Second, the interval says how well you know *this model's* mean. It says nothing about whether the model's weather distribution resembles the North Sea. That is the credibility chapter's question, and no number of replications answers it.

Trace-driven replay

The other way to run the queue is not to generate the events at all. **Trace-driven replay** feeds the scheduler the control room's own event log – last winter's actual alarms, dispatches and windows – and runs the model against them. It answers a different question: not "what will happen", but "what would this policy have done to what did happen". It needs no distributions, which removes the largest modelling assumption, and it is limited to the history you have, which is usually one or two winters and no rare events at all. Use it to check that a model reproduces a known year before trusting its generated ones; that is the cheapest verification in this chapter.

Replay is also what makes the event queue a component of a what-if loop rather than a study. In a published production twin the discrete-event model is called repeatedly by an optimiser: candidate schedules are generated, each evaluated by the discrete-event model of the production system, the best by makespan are kept, a new population is generated, and the loop runs until a stopping criterion is met, after which the winning sequence is fed back to the operator or to the manufacturing execution system [9]. The simulator there is

not answering a question; it is being called a few thousand times as a subroutine, which is a workload, and the Driving a Simulator from a Twin section below names it as a run mode.

Hybrid Simulation: Where the Two Meet

Chapter 6 named the **hybrid automaton** – a state machine whose modes each carry their own rate rules – and stopped, handing the execution question here. It is a real question, and it has three mechanisms and three costs.

A turbine in start-up obeys one set of rate rules, a turbine producing under torque control another, and a turbine in fault with the brake on a third. The mode machine chooses which rate rules the solver runs; the solver's states decide when a guard fires. Two loops, each driving the other. The manufacturing-twin literature's own table lists the hybrid automaton among the model types used for machine and process states, with a generic hybrid automaton written for the continuous and discrete-event dynamics of additive processes [10]; this section is what running one costs.

Zero-crossing detection

A guard is a condition – the rotor speed reaching rated, the oil passing an alarm temperature. Between two solver steps, a condition is either true or false; the switching **instant** is somewhere inside a step and the solver walked straight past it.

The remedy is to write each guard as a function that changes **sign** at the switching point – $\omega_r - \omega_{rated}$, rather than $\omega_r \geq \omega_{rated}$ – and have the solver watch it. At the end of each step the solver checks whether any guard function changed sign; if one did, it does not accept the step. It searches inside the interval for the instant of the crossing, usually by bisection or interpolation, truncates the step there, applies the transition, and starts again.

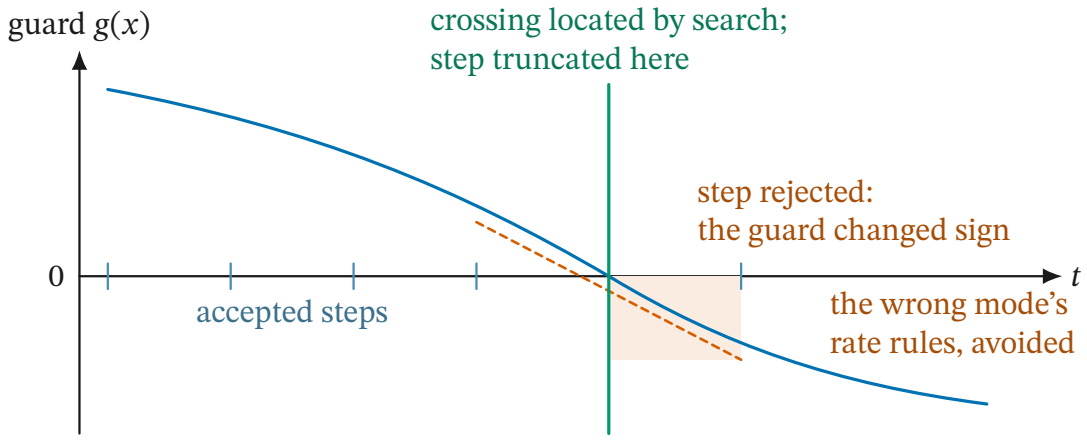


Figure 3: Locating a switching instant inside a step. The solver takes a step, notices that the guard function changed sign, searches inside the interval for the crossing, truncates the step at it, applies the transition, and restarts from there. Without the search, the model runs the wrong mode's rate rules for the shaded fraction of the step.

The cost, in Figure 3, is the search – a handful of extra rate evaluations per switch – and the benefit is that the switch happens at its time. Skipping it is tempting and occasionally defensible: if your step is a millisecond and your guard fires once a minute, the error from

switching at the end of the step is a millisecond of the wrong physics. If your step is ten minutes, it is ten minutes of the wrong physics, and the alarm the model should have raised happens in the wrong state. This is one reason a twin that has to detect mode changes cannot take the biggest step its accuracy allows, and it is precisely the case adaptive solvers advertise: short steps where the dynamics are fast **and where the instant at which something happens must be found properly** [2].

State reset, and the restart it forces

A transition may change the continuous state as well as the mode. A fault trip applies the brake and sets the generator torque to zero; the rotor's speed does not jump, but the torque driving it does, so the *rate* is discontinuous. A contact model, or a tank that is suddenly isolated, can make the state itself jump.

Either way the solver's accumulated history is now wrong. An adaptive solver has spent the last few hundred steps learning how smooth this trajectory is and how long a step it can afford; a multi-step method has several past points it is extrapolating from. At a switch, all of it is discarded: the solver **restarts** from the new state with no history and a conservative first step, and climbs back to its cruising step size over the next several steps.

That is the real cost of a hybrid model, and it is counted in switches, not in steps. The declarative-modelling literature states it plainly: with a variable-step solver, events influence the solver's behaviour because integration must stop every time an event is triggered, and a high number of events slows the simulation down [3]. A model with a thousand switches an hour is not a model with a thousand extra evaluations an hour; it is a model whose solver never gets to take a long step.

Chattering, and the hysteresis band

The pathological case is a guard that fires, is undone by its own transition, and fires again. A thermostat whose switching threshold is exactly its set point switches every step. A curtailment rule that compares an instantaneous power to a limit switches on every gust. The solver detects a crossing, truncates, transitions, and immediately detects the reverse crossing: the simulated clock advances by microseconds per iteration and the run never finishes. This is **chattering**, and its arrival is unmistakable – wall-clock time passing with simulated time standing still.

The ordinary remedy is a **hysteresis band**: switch up at one threshold and down at a lower one, so that a return trip requires a real change rather than a wobble. Mode changes then cost a minimum amount of movement, which is also true of the physical asset, so the band is usually a better model as well as a runnable one. The alternatives – a minimum dwell time in each mode, or filtering the guard's input – do the same job in the time domain.

The turbine through a curtailment and a trip

Chapter 6's six-mode machine driving the two-state rotor-and-oil model, with an adaptive solver at a cruising step of five seconds and a restart costing the solver about ten steps to climb back.

A curtailment. The grid operator caps output at noon and lifts the cap at 16:00. Two guards fire, both on an input rather than on a state, so no search is needed; the mode changes, the

commanded torque changes, neither state jumps. Two restarts, twenty steps of climb-back, and nothing else. Over four hours at a five-second step that is 2,880 steps, of which 20 were slow: under one per cent overhead.

A fault trip. A gearbox alarm at 02:10 trips the turbine. The guard is on a state – an oil temperature crossing a limit – so the solver searches inside a step for the crossing instant. The transition applies the brake, which sets Q_{g} to the brake torque and Q_{a} to approximately zero: the rate is discontinuous, the solver restarts, and the rotor's dynamics are now much faster than they were, so the solver's first attempt at its old step is rejected and it takes several tries to settle. Call it one search and one restart costing thirty steps. Then the turbine sits in fault for twelve hours with the rotor stopped and only the oil cooling – during which the adaptive solver stretches its step to minutes, because nothing is happening, which is the same arithmetic as the event queue arriving at the solver.

The lesson to keep is the arithmetic of the two together: a day with two curtailments and one trip costs three searches and four restarts, perhaps seventy steps of overhead against seventeen thousand steps of work. A day of a chattering guard costs everything. The number of switches, not the number of steps, is the quantity to watch.

The mode belongs in the state record

Chapter 6 asked for this and this chapter can now say why. The mode is part of the state the simulator is restarted from, so it must be part of the state the twin hands it. A simulator restarted with the right temperatures and the wrong mode will run the wrong rate rules from the first step, and produce a smooth, plausible, completely wrong trajectory – the failure mode hardest to notice, because nothing crashes. Put the mode in the state block beside the speed and the temperature, and put its `as_of` beside it too.

Stochastic Simulation: Many Runs Instead of One

The execution problem of Chapter 6's **probabilistic and Bayesian** model type. It comes before the field-solver section, out of the models chapter's order, for one reason: the field section's verdict is that you cannot afford to run the field model, and the argument for what to do instead needs the ensemble already on the table.

The previous section met randomness inside one model and answered it with replications. This section is about the case where the *uncertainty itself is the output*: the twin is asked how long the bearing has, and the honest answer is a distribution.

The ensemble

Chapter 6 modelled the bearing's wear as a Wiener process with drift,

$$W(t+h) = W(t) + \mu\, h + \sigma\sqrt{h}\,\xi, \qquad \xi \sim \mathcal{N}(0,1) \tag{6}$$

and Equation 6 is the step this section runs many times. That chapter then used one month's vibration reading to update a belief over the drift μ across three candidates – 0.5, 1.0 and 2.0 millimetres per second per month – arriving at a posterior of 0.36, 0.53 and 0.11. That posterior is where this section starts. The execution recipe is four lines:

```
for i in 1 .. N:
    mu_i = draw from the posterior over mu        # parameter uncertainty
    W = W_now                                     # the state block's
 ↪  belief
    t = 0
    while W < W_f:
        W = W + mu_i * h + sigma * sqrt(h) * normal(0, 1, stream=i)
        t = t + h
    record t                                  # this member's crossing time
```

Each pass is one **ensemble member**; the recorded crossing times are the sample; their histogram *is* the remaining-life distribution. Figure 4 draws it.

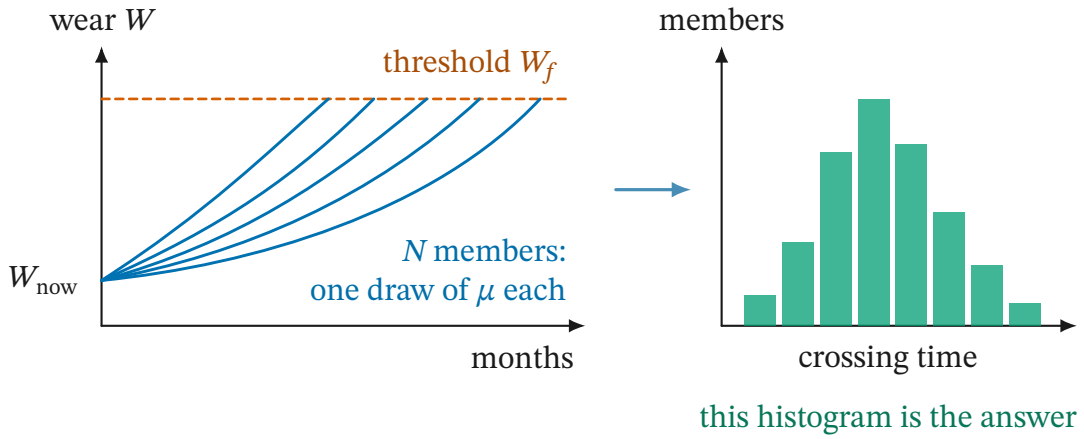


Figure 4: An ensemble as a workload. Left: N trajectories started from the same state, each with its own draw from the parameter posterior and its own random path, run until they cross the failure threshold. Right: the crossing times collected into the histogram that is the answer. One trajectory is not a small version of this picture; it is a single bar.

Two things to notice about the loop. The draw of `mu_i` happens **once per member**, outside the time loop: it is a parameter, not a noise term, and re-drawing it each step would model a bearing whose wear rate changes every month rather than a bearing whose wear rate is unknown. And the two sources of spread are different in kind – the draw of `mu_i` is what the twin does not know about this bearing, the ξ term is the wandering the bearing will really do – which is why the ensemble has to carry both and why a twin that reports only one of them is reporting the wrong confidence.

Chapter 6's three-centre mixture is this ensemble with $N = 3$ and the random term switched off: three candidate drifts, three crossing times, weighted by the posterior. That is a perfectly good answer when the parameter has three candidates. It stops being one the moment μ is continuous, the model has three uncertain parameters instead of one, or the threshold depends on the operating condition – at which point the only tractable method is to sample.

How big is N

The same arithmetic as the last section, with the same shape. If the ensemble's crossing times have a spread s_N and the answer wanted is the mean, the ninety-five per cent half-width is $1.96\, s_N/\sqrt{N}$, so ten times the precision costs a hundred times the runs.

Work one. Suppose the bearing's ensemble gives a mean remaining life of 14.2 months with a spread of 4.8 months. At $N = 100$ the half-width on the mean is $1.96 \times 4.8/10 = 0.94$ months – the twin knows its own average to within four weeks, which is useless when the maintenance window is monthly. At $N = 2000$ it is 0.21 months, about six days, which is inside the window. Two thousand members, each a few thousand steps of four arithmetic operations, is a few tens of millions of operations: milliseconds. For a bearing, N is free and there is no excuse for a small ensemble.

N is free because the step is four operations. The moment the step is a field-solver run, the same N is a supercomputer-month, and that is the trade the next two sections are about.

One caution, the same one the vessel run carried. The decision is rarely about the mean. "Schedule the vessel so that there is a five per cent chance of failing first" asks for the fifth percentile of the crossing times, and a tail percentile is estimated from the handful of members that landed out there, so it needs far more members than the mean does for the same precision. Decide which statistic the decision needs *before* sizing the ensemble.

Variance reduction and common random numbers

Most ensembles in a twin exist to compare two options, not to measure one. Does a second vessel pay for itself? Does replacing the bearing now beat replacing it in six months? The quantity wanted is a **difference**, and a difference between two noisy estimates is noisier than either.

Common random numbers is the cheapest fix in simulation, and it is one line of code: run both options with the *same* seeds, so that member i of option A and member i of option B see the same weather, the same arrival times, the same wear kicks. The randomness then cancels out of the difference, and what is left is the effect of the decision. Run them with different seeds and you may need ten times the members to see the same effect. The practical requirement is the independent streams of the previous section: option B must not consume a different number of random draws before reaching the weather, or the runs are no longer paired.

Other variance-reduction techniques exist – antithetic variates, control variates, importance sampling for rare events – and all of them are worth reaching for before reaching for more compute. Importance sampling in particular is how a question like "what is the chance of three simultaneous faults in a winter" is answered at all, because a plain ensemble would need millions of members to see the event once.

Seeds, determinism, and a run you can repeat

Criterion 3 of the four said a simulator must be deterministic enough to be compared with itself, and an ensemble is where that criterion is either met or quietly lost. The requirement is precise: given the same model version, the same initial state, the same inputs and the same seed, the run returns the same numbers.

Three things break it in practice, and all three are software problems rather than numerical ones. An unseeded generator – seeded from the clock, or from the process id – makes yesterday's prediction irreproducible for no gain. Parallel execution that assigns members to workers dynamically makes the *set* of members reproducible but not their order, which is harmless for a histogram and fatal for a paired comparison; give each member its own stream

indexed by i rather than drawing from a shared generator. And floating-point summation whose order depends on thread scheduling makes bitwise comparison impossible; usually tolerable, occasionally not.

Beyond the seed sits the environment. Computational reproducibility is defined as obtaining consistent and verifiable results from a computational experiment when the same input data, code and software environment are used, and containers are the standard way to pin the third of those – packaging a library with its dependencies so that another party can run it without a dependency conflict and reproduce the result [11]. The twin literature makes the same point about itself: reproducibility means a different team obtaining the results with the same experimental setup, replicability a different team with a different setup, and the reason so few twin results are either is that reproducing the physical side of the experiment is prohibitively expensive – which is why affordable, fully published exemplars are built on purpose [12]. For a twin's simulator the practical reading is short: the seed goes in the provenance record, and so does the container digest.

When the posterior is expensive

Chapter 6 flagged where this section ends up. Its three-candidate table was a teaching simplification; a real twin carries μ as a continuous distribution, and updating a continuous posterior is done by sampling – the twin review lists Markov chain Monte Carlo, Gibbs sampling and slice sampling as the methods [13]. Sampling means evaluating the model once per proposal, and a chain is thousands to millions of proposals long.

That arithmetic is what puts a surrogate inside the loop, and the structural-health literature shows the assembled result: a multi-fidelity neural surrogate maps damage and operational parameters onto sensor recordings and is called by a Markov chain Monte Carlo procedure to update the distribution of the structural state from noisy observations, standing in for a finite-element discretisation of several thousand degrees of freedom that the chain could not have afforded to call [14]. The same shape appears in clinical twins, where the output for a patient is a family of posterior trajectories rather than one forecast, and the quantified uncertainty is what makes the downstream decision risk-aware rather than merely optimal [15].

Hold that thought for two sections. First, the model the surrogate is standing in for.

Field Solvers, and Why They Live Outside the Loop

The execution problem of Chapter 6's **high-fidelity field** model type. The answer, arrived at honestly, is that there is no execution: this model type cannot run inside a twin's decision loop for any purpose, and the value of the section is in seeing exactly why and what the field model is therefore *for*.

To add physical realism to a twin the governing equations of the physics have to be solved, and for most real geometries no analytical solution exists [16]. Solving them numerically is a four-stage pipeline, and each stage has a cost a software engineer should be able to recognise.

Meshing, and the mesh convergence study

The body is divided into elements. The mesh is not an input to the answer; it is part of the model, and a coarse mesh gives a different answer from a fine one. Which raises the question the whole discipline turns on: is this answer the model's, or the mesh's?

The **mesh convergence study** is how that is settled, and it is the one piece of field-solver practice that belongs in a twin engineer's vocabulary. Solve on a mesh. Refine it – halve the element size, or double the element count – and solve again. Repeat until the quantity you care about stops moving by more than your tolerance. The mesh at which it stopped moving is the mesh you use; the ones before it produced answers about discretisation. The study is a fixed cost paid once per model, before any twin runs, and it is a cost that has to be repaid every time the geometry changes.

It is also the cleanest example in this chapter of a **verification** check as against a validation one. A converged mesh says the answer is the model's. It says nothing whatever about whether the model resembles the turbine.

Assembly and the sparse solve

The elements are assembled into a system of equations – one unknown per degree of freedom, a million of them for a tower and blades – and the system is solved. The appendix's *Dimension, and cost growing with it* costs a dense step of a million-state model at around 10^{12} multiplications and notes that a real solver exploits the zeros: the matrix of a meshed body is **sparse**, because an element's equation involves only its neighbours, and everything in the field is built on that.

Two families do the solve. A **direct** solver factorises the matrix once and then solves cheaply for many right-hand sides; it is predictable, and its memory use is the thing that stops it. An **iterative** solver improves a guess until the residual is small enough; it uses far less memory, and its convergence depends on the problem – which is what **preconditioning**, transforming the system into one that converges faster, exists to fix. Choosing between them and tuning a preconditioner is specialist work. Knowing that the choice exists, and that it is where a run's time actually goes, is not.

Eigenvalue extraction is a different, cheaper run

This is the detail Chapter 6 used without naming. That chapter compared the tower's *computed* first sway frequency against the one measured by operational modal analysis. Computing a natural frequency is not a time-domain simulation at all. It is an **eigenvalue problem** – find the shapes in which the structure prefers to oscillate and the frequencies at which it does so, which the appendix's *Modes* section describes – and extracting the first few modes is far cheaper than integrating the structure's motion through time, because it asks a question about the structure rather than about a particular loading.

That is why structural twins are built on it. Measured natural frequencies from operational modal analysis serve as the target values for a parameter optimisation of a detailed finite-element model's design variables, which is finite-element model updating [17]; the parameters such an update adjusts are the mass, centre of gravity, element stiffness, local joint stiffness, soil stiffness and damping, driven by a sensitivity matrix until the analytical

modal parameters match the experimental ones [18]. And once updated, the same eigenvalue run yields what no sensor can: an aircraft twin's wing model, calibrated on the first two bending modes extracted from strain data, then computes higher bending, torsional and skin-buckling modes that are hard to characterise experimentally [19].

So the field model's most useful run, inside a twin, is not a simulation. It is an eigen-analysis on a schedule.

Where it actually runs

Batch. On a cluster or a large machine, on a schedule – quarterly, after a storm, after an inspection – or on a trigger, off the critical path entirely, with results delivered to the twin as a finished artefact rather than as a service call. A run takes hours, sometimes days: the offshore structural-health review says exactly that, adding that understanding a structure's behaviour requires *repeated* simulations with varying input parameters, which multiplies the cost again [20].

Two operational consequences follow from "hours to days" that a software engineer owns rather than the analyst. **Checkpoint and restart**: a twelve-hour job that cannot resume from an interruption will, on a shared cluster with a wall-clock limit, never finish. And **job scheduling**: the twin's request for a field run is a submission to a queue, so the twin's interface to this model type is asynchronous by construction – submit, poll, collect – and pretending otherwise is how a twin's architecture acquires a synchronous call that blocks for six hours.

The verdict

The appendix computes the number. A million-state field model whose step is 10^6 operations, stepped at the resolution the physics requires, comes out at a real-time factor of about one hundredth: a hundred days of compute for a simulated day.

A factor of one hundred is not closed by buying a faster computer, and it is not closed by waiting. It rules the field model out of the loop for every value pattern in Chapter 1 – not only control, but monitoring and diagnosis too, because even the ten-minute SCADA cycle is a thousand times faster than this model can answer. The unit-level manufacturing review states the general case in one sentence: high-fidelity numerical models are not always appropriate for twin applications, because fidelity comes at the price of computational cost or simulation time, and reduced-order modelling or simplification is what is used instead [10]. Reduced-order modelling is named as a key enabler for compressing high-fidelity models into much lower dimensions precisely to relieve that demand [16].

Which is the next section. The field model stays – it is what the reduced model is made from, and what the eigenvalue run updates – but it stays outside, in batch, and something else goes in the loop.

Making an Expensive Simulator Cheap

The execution side of Chapter 6's **reduced-order and surrogate** model type. That chapter said what the cheap model *is*. This section says what it costs to get one, and the argument is a single sentence: **the cheap model is bought with an expensive simulation campaign.**

Figure 5 puts the two halves of that bargain side by side: an offline half that runs once and costs weeks, and an online half that runs every cycle and costs microseconds.

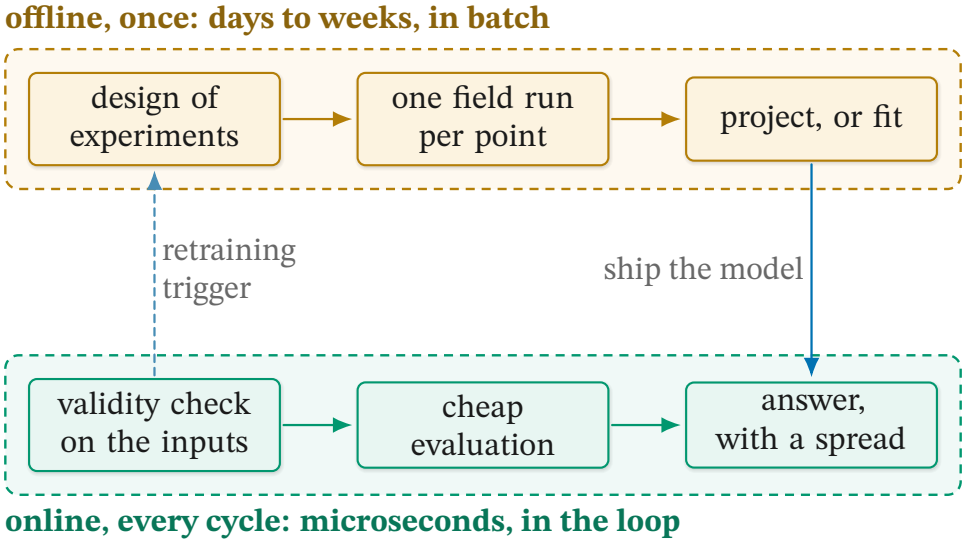


Figure 5: The offline campaign and the online model. Offline, once: a design of experiments over the input ranges the asset will meet, one field-solver run per point, the runs collected and either projected onto a basis or fitted to a function. Online, every cycle: the cheap model, guarded by a validity check that refuses inputs outside the region the campaign covered, with a retraining trigger back to the offline side.

The campaign, priced

The surrogate is a function fitted to the field model's own output, so before it can be fitted, the field model must be run – once per training point. Choosing those points is the **design of experiments**, and it is where the money goes.

Work the arithmetic. Suppose the tower's fatigue surrogate of Chapter 6 takes five inputs: wind speed, turbulence intensity, wave height, wave period and yaw misalignment. Sample each at five levels on a full factorial grid and the campaign is

$$5^5 = 3125 \text{ runs} \tag{7}$$

At four hours a run that is 12,500 core-hours, or about 521 days on one core; on sixty-four cores, eight days of wall-clock time and a noticeable cloud invoice. And five levels per input is coarse: it resolves the response into four intervals per axis, which for a nonlinear quantity like fatigue damage is not much.

Equation 7 is why a grid is usually the wrong design. The count grows as levels to the power of inputs, so the sixth input multiplies the campaign by five. The usual answers are to sample the space rather than grid it – a Latin hypercube spreads a fixed budget of points evenly over every axis at once – or to sample adaptively, fitting after an initial batch and spending the next runs where the fit is worst. Both turn "how many levels" into "how many runs can I afford", which is the question with a budget attached to it.

This is the step the literature names as a first-order obstacle rather than a detail: the prohibitive cost of generating sufficient data for surrogate model training is listed among the

foundational challenges for twins [21], and the structural-health review, having noted that one run takes hours or days, identifies the availability or generation of training data sets as the main problem with substituting a surrogate at all [20].

Projection: the basis you already paid for

For the **reduced** model rather than the fitted one, there is a shortcut, and Chapter 6's tower is the example. The reduction projects the million-state model onto a handful of shapes and keeps only the coefficients of those shapes. Where do the shapes come from? From the modes – and the modes are what the eigenvalue run of the previous section already computed, for the model-updating loop.

So the reduced tower is very nearly free once the field model exists and has been updated: the expensive part, the eigen-analysis, was already on the schedule for a different reason. That is the cheapest path to a fast model in this chapter, and it is available whenever the quantity of interest is dominated by a few modes – structures, acoustics, thermal diffusion. It is not available when the behaviour is not modal, which is when you are back to fitting.

The literature's mature form of this is a **library**: physics-based reduced-order models built component by component from high-fidelity finite-element simulations, which scale to a large asset and, shared across assets, to a fleet of twins, with a classifier inferring from sensor data which library models best describe the asset now – the motivation being stated exactly as this section's: large-scale physics-based models are intractable in a real-time or many-query context [22]. Reduced models built to be interpretable and reusable across processes are argued for on the same grounds in manufacturing [23], and the executable-twin literature reports the finished article: a detailed finite-element model of a turbine blade compressed into a reduced-order model which, with a state estimator, shows the full strain field in real time from a handful of strain gauges [1].

Fitting, and the multi-fidelity trick

For the **surrogate**, the campaign's outputs are training data and the surrogate is whatever function you fit to them – a response surface, a Gaussian process, a kriging model. Chapter 6 used a Gaussian process for the fatigue surrogate because it returns a spread alongside its prediction, which is the property that makes the validity check below possible. Surrogates are defined in the twin literature as exactly this: cheaper approximations that mimic computationally expensive physics models, built from training data taken from the original simulation, not a twin in themselves but usable inside one when the original model is too slow for the decision interval [24].

The **multi-fidelity** trick halves the campaign. Run a few points on the expensive model and many on a cheap one – a coarser mesh, a smaller domain, a simplified physics – and fit a surrogate that learns the *correction* from cheap to expensive rather than the expensive function from scratch. The structural-health surrogate cited in the Stochastic Simulation section is built this way, and its purpose is precisely to survive being called thousands of times by a sampler [14]. When the cheap model is ten times faster and the correction is smooth, the campaign cost falls by most of an order of magnitude.

The validity check, as code

Chapter 6 gave the warning: a surrogate is valid only where it was trained. This section turns it into something a simulator executes, because a warning in a book does not stop a twin extrapolating on a February morning.

```
def predict(surrogate, u, run_log):
    """Evaluate the surrogate, or refuse. Never extrapolate silently."""
    if not surrogate.domain.contains(u):
        run_log.record("surrogate: input outside training domain", u)
        return Refused(reason="outside training domain",
                       fallback="field model, batch")
    mean, spread = surrogate.evaluate(u)   # the process returns both
    if spread > surrogate.spread_limit:
        run_log.record("surrogate: spread above limit", u, spread)
        return Uncertain(mean, spread)     # usable, but flagged
    return Ok(mean, spread)
```

Three properties of that listing are the point of it. The domain check is on the **inputs**, before any evaluation, and is a box or a hull recorded by the campaign – not a guess. The spread check is the surrogate's own admission of ignorance, which is why the fitted model was chosen to produce one. And the refusal is a **return value**, not a log line: the caller has to decide what to do without an answer, which is the only way the twin's behaviour outside the training region is designed rather than discovered.

The retraining trigger, and the offline/online split

Two rates, and they are the reason this section is structured the way it is.

Offline, rarely: the field model, the mesh convergence study, the campaign, the fit. Days to weeks, in batch, off the critical path, and repeated only when something makes the campaign stale.

Online, every cycle: the surrogate evaluation and the validity check. Microseconds, in the loop.

The **retraining trigger** is what joins them, and there are three honest ones. The validity check refuses, or flags high spread, on a growing fraction of cycles – the asset has moved into a corner the campaign did not cover. The field model has been updated, so the surrogate now mimics a model that no longer exists. Or the asset itself has changed – a blade replaced, a foundation grouted – which is a configuration change and the configuration and manager chapter's business, but it invalidates the campaign all the same. The National Academies' report names the obligation created by all three: surrogates that make dynamic adaptation feasible must themselves be updated and validated as the representation evolves, and it lists this among the open research gaps rather than among the solved problems [21]. A twin that ships a surrogate without a retraining trigger has bought a fast model and a slow-growing lie.

Time: Simulated Clock, Wall Clock and the Twin's Clock

Every section so far asked whether a model *can* be run. This one asks whether it can be run **here**, and it is the pivot of the chapter.

The appendix defines the **real-time factor** as simulated time covered per unit of wall-clock time,

$$RTF = t_{\text{sim}}/t_{\text{wall}} \tag{8}$$

and that is the last time this chapter defines it. Equation 8 is a pure number, and the three regimes it falls into, drawn on one axis in Figure 6, are three different jobs.

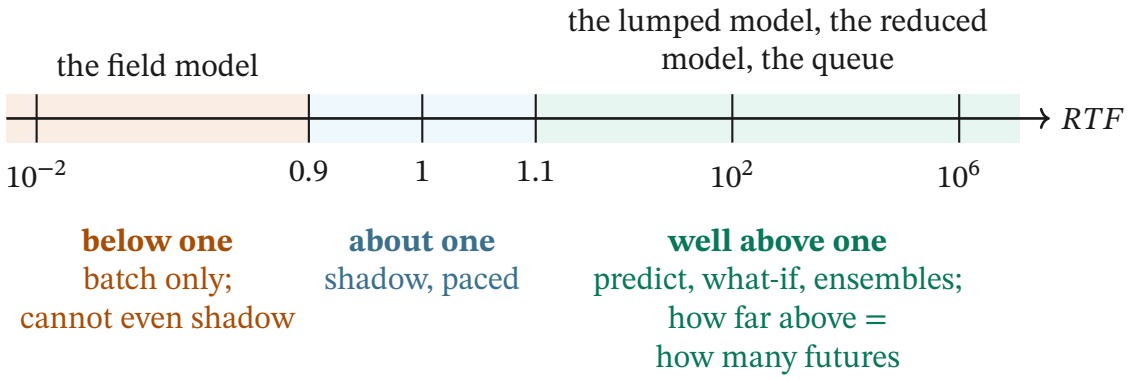


Figure 6: The three real-time regimes on one axis. Below one, the simulator falls behind the world and can only be run in batch, after the fact. Around one, the simulator is paced to consume readings as they arrive and shadows the asset. Above one, the simulator outruns the world, and how far above decides how many futures fit inside a decision's latency.

Three regimes

RTF **below one: batch only.** The simulator cannot keep up with the world, so it can only ever be run after the fact, on a horizon that has already happened or on a schedule. The field model of the previous section, at $RTF = 0.01$, is here, and nothing about the deployment changes that. Note what it rules out: not merely control, but *shadowing*. A model that cannot cover a minute of asset behaviour in a minute cannot be kept in step with the asset at all.

RTF **about one: shadow, and pace.** The simulator runs alongside the asset, consuming readings as they arrive and producing its expectation as the asset produces its reality. Nothing in this regime happens by accident: a simulator whose natural *RTF* is 50 must be **paced**, or it will run fifty minutes of simulated time in the first minute and then sit idle with nothing to consume. Pacing is the engine deliberately sleeping – compute the next step, note the wall-clock time it should be delivered at, wait. The twin of a floating turbine reported in the literature is built for exactly this regime, intended to run in real time on a cloud platform beside its asset [25].

RTF **well above one: predict and explore.** This is what makes a twin more than a shadow. At $RTF = 10$ the simulator covers ten minutes of future in a minute, or ten one-minute futures; at $RTF = 1000$, a thousand. The figure that matters is not *RTF* alone but *RTF* against

the decision's latency: a decision with a five-minute budget and a simulator at $RTF = 600$ can afford $600 \times 5/60 = 50$ hours of simulated future, which can be one fifty-hour forecast or fifty one-hour what-ifs. That division is the single most useful calculation in this chapter, and it is why faster-than-real-time simulation is reported as an enabling property rather than a nicety – the modelling survey cites work using a faster-than-real-time twin to predict the dynamic behaviour of smart grid systems [16].

Pacing, and what to do when it fails

Pacing costs nothing while there is slack. The interesting case is when there is not: the simulator is asked for a ten-minute horizon every ten minutes, and this cycle takes eleven. A twin has four options and they should be chosen in advance rather than discovered.

- **Shorten the horizon.** Predict five minutes instead of ten. Cheapest, and it degrades the answer in a way the consumer can be told about.
- **Coarsen the model.** Switch to the surrogate, or a larger tolerance. Degrades accuracy rather than reach.
- **Drop the cycle.** Skip this prediction and re-initialise from the next state. Acceptable for a prediction, never for an accumulator.
- **Fall behind.** The worst option, and the default if the other three were not implemented: the twin's answers arrive later and later, describing a world that is further and further in the past.

Time discrepancy, and getting back in step

Chapter 5 defined the time discrepancy $D = t_i - t_e$ between the instant a twin holds and the instant its asset is at. This chapter reads the same quantity as how far a *running simulation* has fallen behind its asset, and the published measurements are worth quoting in numbers because they are more sobering than the definition.

In a study that set out to measure and address exactly this, the maximum observed discrepancy during a deliberately degraded period was around five seconds; when normal conditions returned, the twin caught up with wall-clock time in 60.8 time steps of 100 milliseconds, averaged over five runs – about six seconds of catching up for five seconds of falling behind, and possible only because the twin could run faster than real time [26]. Two findings from the same work matter more than the headline. First, **the maximum step a model accepts bounds how fast it can catch up**: a component with a stricter limit on its step size slows the pace at which the whole twin gets back in sync, so a model chosen for accuracy can prevent resynchronisation. Second, **the threshold was exceeded on occasion even during a normal run**, which the authors read as a need to estimate the threshold from the variability of normal operation rather than from an ideal. A twin whose "behind schedule" alarm was set from a quiet afternoon will cry wolf.

Chapter 5's twinning rate is the other side of this: the rate at which the twin is synchronised with its asset, which the manufacturing review is careful to distinguish from the rate at which the data is sampled [10]. A simulator sets an upper bound on the twinning rate that no amount of data can raise.

How far ahead is honest

RTF says how far ahead you *can* run. It does not say how far ahead you *should*, and the two are not related.

The appendix's *State Machines with Probabilities* has the argument in its cleanest form. Predict a Markov chain forward without looking at the world and the belief spreads at every step, drifting towards the chain's long-run distribution – at which point the prediction carries no information about *this* asset at all, only about assets of this kind. Certainty leaks away with every step predicted without a correction, and it leaks at a rate set by the system, not by the simulator. **A prediction horizon is therefore set by how fast the belief decays, not by how fast the simulator runs.** A simulator at $RTF = 10^6$ can produce a thirty-year forecast of the turbine's rotor speed in a second; nobody should read it.

The practical form of the rule: run ahead to the horizon at which the predicted spread reaches the width at which the decision flips, and stop there. For the bearing's remaining life that horizon is months, because the wear model's spread grows slowly. For the rotor under a gusting wind it is seconds.

Table 3 ties the two clocks to Chapter 5's age of twin.

Table 3: What each of the turbine's models can sustain. Read the last column as the smallest age of twin the simulator does not prevent – the data usually decides, and the field model is the one row where the simulator decides instead.

Model, on the turbine	Achievable *RTF*	What that allows	Age of twin it can sustain
two-state lumped, implicit, $h = 600$	about 10^7	prediction, what-if, ensembles, anything	bounded by the data, not the simulator
mode machine and vessel queue	about 10^8 for a simulated year	a thousand-member ensemble in a second	bounded by the event log's latency
bearing Wiener ensemble, $N = 2000$	about 10^6	a fresh remaining-life distribution every cycle	monthly readings; the simulator is never the constraint
reduced tower, three modes	about 10^3	load estimates every second, with headroom	one second
fatigue surrogate	about 10^9 per evaluation	ten-minute fatigue totals for microseconds	ten minutes
field model of tower and blades	about 10^{-2}	a quarterly batch run	not applicable; it is not in the loop

The shape of Table 3 is the shape of the whole chapter: five of the six rows are limited by the data arriving, not by the arithmetic, and the sixth is limited so badly that it is not in the

loop at all. That is a comfortable place to be, and it is a consequence of Chapter 6 having chosen small models for the decisions that needed speed.

Driving a Simulator from a Twin

Everything above is about a simulator. This section is about the twin calling one, and it is the section to copy from.

The call

One function, four things in, two things out.

```
def run(model_ref, initial_state, inputs, horizon, *, mode, seed,
 ↪  solver):
    """Advance a model from a state the twin supplies, and say what was
 ↪  run.

    model_ref     which model, and which version of it
    initial_state the state block's current belief, with its as_of
    inputs        the input series over the horizon: measured,
                  forecast or assumed
    horizon       how far ahead, in simulated time
    mode          one of the five run modes below; fixes where
                  the inputs come from
    seed          fixes the run's randomness; recorded, never
                  defaulted to a clock
    solver        the step kind and its settings

    returns (outputs, provenance)
    """
```

Four things about that signature are deliberate and each corresponds to a criterion from the What a Simulator Is section.

`initial_state` **is an argument, not a constructor parameter.** The simulator is restarted from the twin's belief every cycle – criterion 2 – so setting the state is a normal call, not a set-up phase. If your simulator can only be initialised at construction, you will pay to rebuild it every cycle, and the Consistent Initialisation subsection above is where that cost lives.

`inputs` **is a series over the horizon, not a value.** The twin must supply the wind, the load, the grid command for the whole interval being simulated. Where that series comes from is what distinguishes the run modes below, and getting it wrong – feeding a forecast where a measurement was assumed – is a category of bug that produces perfectly smooth wrong answers.

`seed` **is explicit and never defaults to the clock** – criterion 3.

`provenance` **is returned, not logged** – criterion 4. A caller that stores the outputs and discards the provenance has stored numbers it cannot defend.

The output is Chapter 5's tuple

The run does not return an array. It returns a stream of Chapter 5's tuples, in the same shape as a reading, with the fields read as Table 4 gives them.

Table 4: Chapter 5's tuple, read for a simulated value. Simulated data lands in the same store, in the same shape, as measured data; the binding is what tells them apart, and nothing downstream has to be rewritten to accept it.

Field	For a sensor reading	For a simulated value
value	what the sensor reported	what the model computed
unit	the sensor's unit	the model's unit, which must be the same one
`as_of`	when the world was in this condition	the **simulated** instant, not the wall-clock instant of the run
binding	which sensor on which asset	which **model version** on which asset, plus the run id
quality	the sensor's own health flags	whether the step converged, whether a surrogate refused, whether the value is inside the model's validated range
provenance	the acquisition path	the run that produced it

Two of those rows are where mistakes get made. The `as_of` of a simulated value is the instant in the *simulated* world, which for a prediction is in the future – and a store that silently stamps it with the wall-clock time of the run has turned a forecast into a measurement. And `quality` for a simulated value is not a sensor concept: it is the solver's convergence flag, the surrogate's refusal, the ensemble's spread. Give it a vocabulary in the twin's own terms rather than reusing the sensor's.

Five run modes

The same simulator, the same model, and five different jobs. What distinguishes them is where the initial state comes from, where the inputs come from, and what is done with the output – Figure 7 wires all five up at once, and Table 5 is the table to keep.

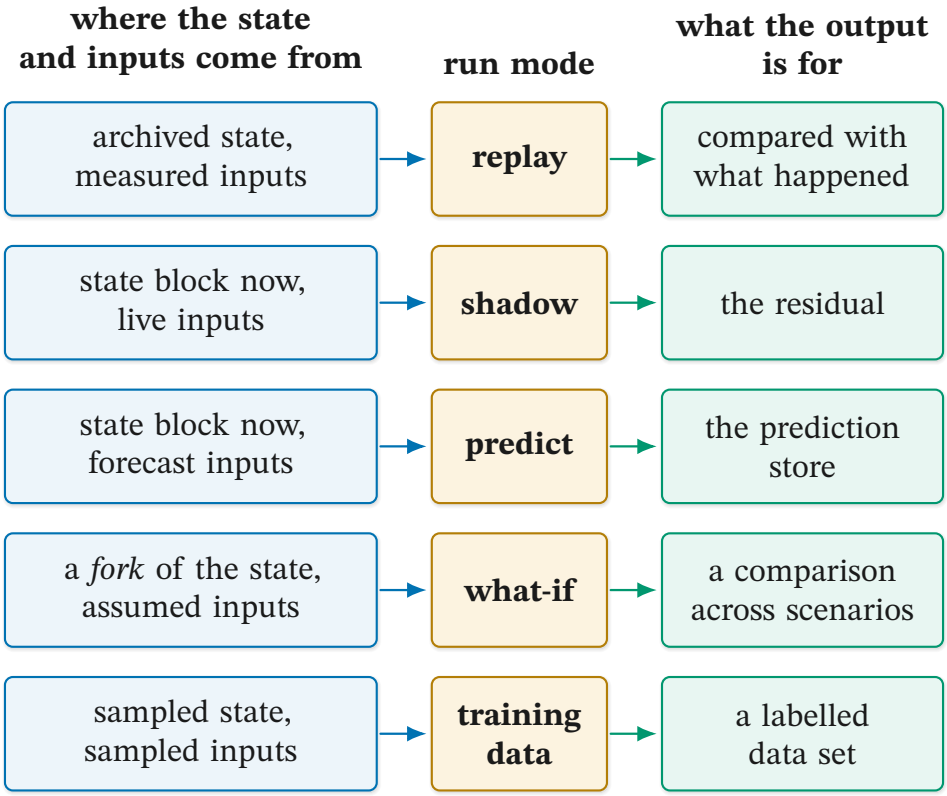


Figure 7: The five run modes. Each is the same call with a different source for the initial state and the inputs, and a different destination for the output: replay from the archive, shadow from the live state and live inputs, predict from the live state and a forecast, what-if from a forked state and an assumed input, and training-data generation from sampled states and sampled inputs.

Table 5: The five run modes. The mode is an argument to the call because it fixes three things the simulator itself cannot know.

Run mode	Initial state from	Inputs from	Output goes to	Typical *RTF* needed
replay	an archived state at a past instant	the archive, measured	comparison against what happened	above 1, and the higher the better
shadow	the state block, now	the live data stream	the residual, against the live reading	about 1, paced
predict	the state block, now	a forecast, or a persistence assumption	the prediction store, with a horizon	well above 1
what-if	a **fork** of the state block	assumed, per scenario	a comparison across scenarios	above 1 times the number of scenarios

Run mode	Initial state from	Inputs from	Output goes to	Typical *RTF* needed
generate training data	sampled across the operating envelope	sampled across the operating envelope	a labelled data set	irrelevant; it is a batch job

Replay re-runs a past interval from an archived state against the inputs that really occurred, and compares the model's trajectory with the asset's. It is the cheapest verification a twin has and the first thing to build.

Shadow is the mode Chapter 3's shadow twin runs continuously: the model steps alongside the asset from the same state, and the difference between its output and the incoming reading is the residual Chapter 2 already watches.

Predict runs from now into the future, with the inputs necessarily assumed – a wind forecast, a persistence assumption, a planned schedule. The honesty of a prediction is bounded by the honesty of its input series, and the horizon is bounded by the belief-decay argument of the Time section rather than by the budget.

What-if is predict, forked. The state block's belief is copied, several copies are run under different assumed inputs or different decisions, and the *differences* between them are the answer. The literature's definition is compact: a what-if simulation inspects the behaviour of the physical twin under given hypotheses called scenarios [27]. Two engineering points follow. The fork must be a real copy, or scenario B starts from the state scenario A left. And the comparison must use common random numbers, from the Stochastic Simulation section, or the difference between scenarios drowns in the difference between their weather. A published production twin runs this mode as an inner loop, generating candidate schedules, evaluating each with the discrete-event model and feeding the winner back to the operator or the execution system [9].

Generating training data is the mode that is not about this asset at all. Run the model across the operating envelope, record inputs and outputs, and use the result to train something – a surrogate, as the previous section described, or a controller, or a perception model. The survey of AI simulation by twins describes precisely this: virtual training environments in which agents are developed safely and efficiently on simulated, synthetic data, offered as the response to insufficient data volume and quality, with dataset generation the offline end of the practice and training inside the environment the online end [28]. For this book, the point is that it is a **batch** job with no real-time requirement and a very large total compute, which is a different operational shape from the other four and should not share their infrastructure.

Where the simulator runs

Chapter 5 already decided this, and this section reads it off rather than re-deciding. That chapter's hot, warm and cold paths map onto the four placements of Table 6.

Table 6: Where a simulator runs, read off Chapter 5's path table. The row is chosen by the *RTF* the decision needs and the data the model must reach, in that order.

Placement	Chapter 5's path	Suits	Cost shape
embedded, on the controller	hot	a step function of a few lines, at the control rate	fixed, tiny, and capped by the hardware you already bought
edge, beside the asset	hot and warm	the lumped model, the mode machine, the reduced model	fixed hardware, no per-run charge, limited headroom for an ensemble
container, in a cluster	warm	ensembles, what-if fans, replay	per-run, elastic, the natural home of a bursty workload
cloud batch	cold	the field model, the surrogate campaign, training-data generation	per-core-hour, scheduled, and the one placement where a job queue is part of the design

Packaging is what makes a row portable. The **executable digital twin** is the literature's name for the artefact: a self-contained executable behaviour of an asset, usable by anyone at any point in the lifecycle, fast enough for its purpose – usually through model order reduction – with the packaged model carrying its own execution engine, and an explicit link back to the parent twin that keeps the two consistent and traceable [1]. Platforms take the same idea further down the stack, using containerisation, loose coupling and micro-services to compose twins dynamically and naming reproducibility as one of the properties that buys [29]. For a software engineer the reading is familiar: the simulator is an artefact with a version, a digest and a manifest, and it is deployed, not installed.

Run provenance

The last of the four criteria, written out. A run records:

- **the model**, by identifier and version, and the digest of the model file if it has one;
- **the solver**, by name, with its step size or tolerance and every option that was not the default;
- **the initial state**, or its digest, with the `as_of` it carried;
- **the inputs**, by reference to the series and its version, with a note of whether they were measured, forecast or assumed;
- **the seed**, and the stream assignment if the run was parallel;
- **the environment**, as a container digest or the equivalent [11];
- **the cost**: wall-clock time, steps taken, rate evaluations, and the achieved *RTF*.

That is six identifiers and a cost, and it is small enough to attach to every tuple the run

produces. The test of whether it is complete is simple and worth running once: hand the provenance record to a colleague and see whether they can reproduce the numbers. If they cannot, the missing field is the one to add.

Choosing a Simulation Technique

Chapter 6 closed with a table of five model types against four criteria and four questions to walk it with. This section is the same shape for execution, and the two are meant to be read as a pair: that table chooses what to write, this one chooses what will run it and where.

The table

Table 7: The five model types against the technique each forces, what it can achieve, where it goes and who is needed. The third row's *RTF* is the reason the fourth row exists.

Model type	Technique it forces	Achievable *RTF*	Where it runs	Expertise needed
lumped first-principles	a solver; implicit or adaptive if stiff	10^5 to 10^7	embedded or edge	the domain's physics, plus a step size defended
discrete-state and discrete-event	an event scheduler, and replications	10^6 to 10^8 per run, times N	edge or container	the operation's rules, plus output analysis
high-fidelity field	a batch field solver; meshing, sparse solve, eigen-analysis	about 10^{-2}	cloud batch, off the critical path	a specialist package and a specialist
reduced-order and surrogate	a cheap evaluation online, an expensive campaign offline	10^3 to 10^9 online	edge online, cloud batch offline	the field model's, plus projection or fitting
probabilistic, Bayesian	an ensemble: N runs of whatever the wrapped type uses	the wrapped type's, divided by N	container, or edge if N and the step are small	probability, plus seed and stream discipline

Two entries in Table 7 are worth reading twice. The probabilistic row's *RTF* is the wrapped type's *divided by N*, which is the whole reason an ensemble over a lumped model is free and an ensemble over a field model is impossible. And the reduced-order row has two placements and two rates, because it is the one type whose cost is paid somewhere other than where its benefit is taken. Choosing the formalism, and therefore the simulator, is

a deliberate stage in a twinning system's design rather than something inherited from whatever tool was already installed [7].

Four questions

Walk the table with these, in order. Each removes rows.

1. **Rates, events, or both?** Quantities that change continuously want a solver; modes, arrivals and waiting want a scheduler. If the answer is both – and for any asset with operating modes it is – it is a hybrid model, and the extra question is how many switches an hour, because switches, not steps, set the cost.
2. **A number, or a spread – and can the step afford the runs?** A monitoring threshold takes a number. Anything with "when", "how long" or "how likely" in it takes a spread, which means N runs, which means the step function's cost is multiplied by N before it is compared with the budget. This is the question that most often sends a design back to a cheaper model type.
3. **Does the achievable *RTF* clear the decision's latency?** Take the latency from Chapter 1, the *RTF* from the table, and divide. If it clears, stop. If it does not, there are exactly four remedies and they are worth naming so that the choice is made rather than defaulted: shorten the horizon, coarsen the model, buy a surrogate with an offline campaign, or move the decision off the critical path.
4. **Where must it run, and does that placement allow the technique?** An embedded target may have no floating point worth the name, no room for an implicit solver's inner iteration and a hard real-time deadline that an adaptive solver's variable cost violates. An edge box can run the ensemble but not the campaign. Placement constrains technique, not only the other way round, and it is the question most often asked last and answered first.

The turbine's answers, returning to Chapter 6's five decisions. Power and thermal expectation: rates, a number, ten-minute latency, edge – an implicit or adaptive solver on two states, *RTF* around 10^7, and the only real decision was the step. Availability and repair planning: events, a spread, hours of latency, container – an event scheduler with a couple of thousand replications, and the arithmetic of the Discrete- Event section sizes them. Structural integrity: rates over a body, a number, seasonal latency, cloud batch – a field solver, and an eigen-analysis rather than a time-domain run. Load monitoring every second: the reduced tower at the edge, its basis taken from the eigen-analysis already paid for. Bearing replacement: a spread on a monthly latency, two thousand ensemble members over a four-operation step, anywhere at all. Five decisions, five techniques, four placements, one turbine.

Trusting a run, as arithmetic

Four checks make a run trustworthy *as arithmetic*. Each is cheap, each is mechanical, and a twin should run them on a schedule rather than when something looks wrong.

1. **Refine the step.** Halve h, or tighten the tolerance by an order of magnitude, and rerun. If the answer moves by more than your tolerance, the answer you had was the step size's. This is the mesh convergence study of the Field Solvers section, applied in time instead of in space, and it is the single most informative check in the list. Measure it the way the tolerance sweep of the Continuous- Time section was measured –

reporting step counts, not only elapsed time, on a model small enough for the elapsed time to be noise [3].

2. **Swap the solver.** Run the same model with a different step kind – explicit against implicit, one adaptive family against another. Two independent methods agreeing is evidence about the arithmetic that neither gives alone; two disagreeing means one of them is wrong and you do not yet know which.
3. **Check a conserved quantity.** Energy, mass, charge, the number of parts in the system, the sum of the probabilities. Whatever the physics conserves, compute it at the start and the end and compare. This catches sign errors, unit errors and dropped terms that no amount of step refinement will, because a wrong model converges beautifully to the wrong answer.
4. **Record the seed and the versions.** Which is the provenance record of the previous section, and it is a check because it is what makes the other three repeatable next quarter.

And then stop. All four are **verification**: they establish that the equations were solved correctly. None of them establishes that the equations describe the turbine. That is **validation**, it is the credibility chapter's subject, and the distinction is load-bearing enough that standards bodies make it explicit – model credibility assessment, including verification, validation and uncertainty quantification, is to be applied throughout a twin's life cycle [30], and because a twin is dynamic, verification and validation are a continuous process performed periodically or on demand rather than a step completed once [31]. It is also a gap in practice rather than a solved problem: a recent survey observes both that ISO 23247 does not address verification, validation and uncertainty quantification at all, and that only about sixty per cent of the twin-driven AI simulation techniques it reviewed do anything to maintain the simulator's own fidelity over time [28].

For readers who know control theory

One analogy, gathered here rather than scattered through the chapter, for the readers who will find it useful; skip the paragraph if the words are unfamiliar, because nothing later depends on it.

A twin's simulator is doing the job the internal model does in a model-predictive controller: it is the plant model rolled forward over a horizon to produce a predicted trajectory that something else then acts on. The correspondences carry further than they usually do in analogies. The prediction horizon is bounded by model quality in both cases, which is the belief-decay argument of the Time section. The what-if fan is a candidate-input sweep. The real-time factor against the decision's latency is the constraint that decides how large a candidate set the controller can evaluate per interval, which is the classic implementation limit of the method. And the residual of Chapter 2 is the innovation. What is *not* shared is the closing of the loop: the controller commits its first move and re-plans, whereas a twin at the monitor, diagnose or predict patterns stops at the prediction and hands it to a person. The control chapter of a later part is where the difference is taken seriously; here it only explains why so much of this chapter's vocabulary will sound familiar to a control engineer, and why the state and estimator chapter, next, is the other half of the picture.

Simulating the Plant Demonstrator

The one section where the rig of Chapter 2 appears, with that chapter's symbols unchanged. It is here for a negative result: the rig is a perfectly good twin and it teaches almost nothing about choosing a simulator, and knowing *why* is worth a section.

The simulator is twelve lines

Chapter 3 named the rig's simulator without writing it. Chapter 6 showed that the rig's model is already a lumped first-principles model in sampled time,

$$m_{k+1} = m_k + g\,V_k - d\,T_c \tag{9}$$

Equation 9 is the whole model; here is the whole simulator around it.

```
def simulate(m0, doses, n_checks, theta):
    """Run the pot forward n_checks checks. theta = (g, d, T_c)."""
    g, d, T_c = theta
    m = m0
    trace = [(0, m)]
    for k in range(n_checks):
        V = doses[k]                  # millilitres at this check
        m = m + g * V - d * T_c           # the step: one line, no solver
        trace.append(((k + 1) * T_c, m))
    return trace
```

Twelve lines, and there is no solver in them. That is the section's first point and it is a general one: **the rig's step is already a difference equation in sampled time, so there is no solver choice to make.** The step size is not a numerical parameter at all – it is T_c, the check interval, fixed by the decision to look at the plant every six hours. Chapter 6 observed that this is Euler's step with $h = T_c$; the consequence for this chapter is that every question the Continuous-Time Simulation section spent pages on – explicit or implicit, what tolerance, what does stability cost – has already been answered by somebody choosing a check interval for reasons that had nothing to do with numerics.

Why no implicit solver

One state and one time constant. The moisture falls at d per hour and rises with a dose; there is no fast process hiding behind the slow one, so the system is not stiff, so there is nothing for an implicit step to buy. An implicit step here would solve a linear equation in one unknown to get the answer the explicit line already gave.

Worth being precise about the accuracy, because "it is not stiff" is not the same as "it is exact". Drying at d per hour over a six-hour interval is treated as $d\,T_c$ of loss, which assumes the rate is constant across the interval. If the drying rate in fact depends on how wet the soil is – it does, a little – then the six-hour explicit step is inaccurate in a way a shorter step would fix. The rig does not fix it, and it is right not to: the probe's own resolution is coarser than the error, so refining the step would improve a number nobody can measure. That is the honest form of "accurate enough", and it is a decision, not an oversight.

The watering schedule as an event queue

The rig has a second simulator, and Chapter 2 wrote it without naming it. The controller's day is a sequence of timed events: check at 06:00, check at 12:00, check at 18:00, and a pump that runs for a computed number of seconds when a check decides to dose. As an event queue that is three scheduled events a day, plus at most three pump-off events.

Six events a day, fifty steps of arithmetic between them. A whole growing season – one hundred and eighty days – is about a thousand events, which runs in a millisecond. The same season in a fixed-increment loop at one-second resolution would be 15.5 million steps of nothing. The Discrete-Event section's arithmetic, on a rig.

The rig's real-time factor

One step of the rig's model is a multiplication, a multiplication, an addition and a subtraction: call it four operations, plus the loop overhead, and call the whole step a microsecond on the rig's controller to be generous. One step covers $T_c = 6$ hours of simulated time, which is 21,600 seconds. So

$$RTF = 21600/10^{-6} = 2.2 \times 10^{10} \tag{10}$$

Twenty-two billion. Equation 10 means the rig's controller can simulate a full growing season in about a millisecond, which is why nothing in Chapter 2 ever discussed performance and why this chapter's central tension – can it run here, in time – does not arise at all. Every placement in the placements table is available. Every run mode is affordable. The choosing table's four questions all answer "anything".

Why no ensemble yet, and what would change that

The rig's twin reports a number: the predicted moisture, and the residual against the probe. It could report a spread instead – Chapter 6 gave it a posterior over the dose response g – and the ensemble to turn that posterior into a distribution of predicted moisture would cost a thousand runs of a four-operation step, which is nothing.

So why not? Because nothing downstream would use it. The decision is dose or do not dose, once every six hours, against a threshold; a spread around the predicted moisture changes that decision only when the threshold falls inside the spread, and a technician with a watering can is not going to act differently on "probably 22, could be 19 to 25" than on "22". Add the ensemble when the decision changes – when the rig grows a scarce water budget to allocate across pots, or a plant whose cost of over-watering is not symmetric with the cost of under-watering. Then the spread is what the allocation is computed from, and the ensemble earns its place.

Which is the honest conclusion of the section. The rig is the right scale for learning what a twin *is*; the turbine is the right scale for learning what a simulator *costs*. This chapter needed both.

Faded Examples

Three examples on Chapter 6's other systems, decreasing in how much is worked. None uses the rig.

Example 1, worked in full: the battery cell, stepped and then run many times

Continuous-time, then stochastic. Chapter 6's lithium-ion cell in a bus: an equivalent circuit with two states, the state of charge *SoC* and the polarisation voltage V_1, with $R_1 = 0.015$ ohms, $C_1 = 2000$ farads, a capacity $Q = 10800$ coulombs and a discharge current $I = 3.0$ amperes. Model-based battery prognostics work on exactly these equivalent-circuit models of cell voltage, or on physics-based degradation models [32].

Step 1: is it stiff? Two time constants. The polarisation pair settles with $R_1 C_1 = 30$ seconds. The state of charge falls at $I/Q = 2.78 \times 10^{-4}$ per second, so from full it takes $Q/I = 3600$ seconds to empty. The ratio is 120 to 1 – **mildly stiff**, which is the ordinary case and exactly the situation the appendix describes: a fast process finished almost immediately, sitting inside a slow one that is the reason the model exists.

Step 2: choose the step. Explicit stability needs h below about twice the fastest time constant, so $h < 60$ seconds. Accuracy on the polarisation transient needs rather better – a tenth of its time constant, $h = 3$ seconds, is the usual rule of thumb, and gives a voltage sag curve accurate to well under a millivolt. At $h = 3$ seconds an hour of discharge is 1,200 steps of about ten operations: twelve thousand operations, microseconds. So the answer is $h = 3$ seconds explicit, and the stiffness cost nothing at all.

Notice what made that painless: the *fast* state is the one the output depends on, so accuracy and stability pull the same way. Compare the turbine, where the fast state was the one nobody was asking about and the step it forced was pure overhead. That contrast is the whole of stiffness as an engineering matter.

Step 3: the ensemble, for capacity fade. The decision is not about this hour but about this cell's life: when will its capacity fall to eighty per cent of new, at which point the pack is retired? Chapter 6 gave the empirical fade law

$$Q(n) = Q_0 \left(1 - a\, n^b\right) \tag{11}$$

with n the cycle count. Take $b = 0.5$ – fade proportional to the square root of cycles, the common empirical form – and a fade coefficient a that is *not* known for this cell: from the fleet, it is normally distributed with mean 0.004 and standard deviation 0.0005. End of life is $Q/Q_0 = 0.80$, which by Equation 11 happens at

$$n_{\text{eol}} = (0.2/a)^2 \tag{12}$$

At the mean a, Equation 12 gives $(0.2/0.004)^2 = 2500$ cycles. But the answer wanted is not that number. Draw a from its distribution N times, compute n_{eol} for each, and the histogram is the answer: at $a = 0.0035$ the cell lasts 3,265 cycles, at $a = 0.0045$ it lasts 1,975, so a one-standard-deviation swing in a parameter nobody measured moves the retirement date by thirteen hundred cycles. Reporting "2,500 cycles" would have thrown that away. Algebraic reduced-order capacity models of exactly this form, paired with sampling methods for the

parameter uncertainty, are reported as capturing model-form and parameter uncertainty in a way a plain neural-network ensemble does not [33].

Step 4: how many members? The spread of n_{eol} across the draws comes out at about 550 cycles. For the mean to be known to plus or minus 25 cycles at ninety-five per cent, $N = (1.96 \times 550/25)^2 = 1860$, so 1,900 members. Each member is one evaluation of Equation 12 – not a simulation at all – so the ensemble costs microseconds. If instead the decision needs the fifth percentile – "retire early enough that only five per cent of cells fail first" – size the ensemble for the percentile, which will need several times as many members, and say in the twin's output which statistic was asked for.

Step 5: the retraining trigger. The fade coefficient is a parameter with a posterior, and every month's capacity measurement updates it by Chapter 6's Bayes' rule. That is the twin working as designed. The trigger for something larger is a *systematic* drift – the posterior walking steadily in one direction over many months, which says the fade *law* is wrong for this duty cycle and not merely its coefficient unknown. A published framework does the same job the other way round, fitting an empirical capacity fade model and training a machine-learning model to predict its parameters from early-life features, reporting under two per cent error in distribution and under four per cent out of it [34] – and that out-of-distribution number is the validity check of the Making an Expensive Simulator Cheap section, in a battery's clothes.

Example 2, one step left to you: the 3D printer as a hybrid simulation

Hybrid. Chapter 6's desktop 3D printer, whose modes have been published as a state machine: StandBy, Connected on a device connecting, Operational on receiving a print file with preheating before printing, pausing and cancelling while printing, back to StandBy when the job finishes, Error reachable from anywhere and a reset back to StandBy [35]. That machine is the discrete half. The continuous half is one rate rule per mode: the hot end's temperature, rising towards its set point under the heater's power and falling by loss to the chamber, with the extruder's filament consumption running alongside it while printing.

The hybrid structure is the interesting part. **Preheating** ends at a guard on a continuous state – the hot-end temperature reaching the material's print temperature – which is a zero-crossing to be located, not a flag to be polled. **Printing to paused** and **paused to printing** are guards on inputs, so no search is needed, but each one changes the heater's duty and therefore the rate: a solver restart each time. **Error** can fire from any mode and applies a state reset – the heater off, the extruder retracted – which is the expensive kind.

Your step. A four-hour print job on a printer that is paused twice by the operator, hits one filament-jam error and is resumed, and whose chamber-temperature controller cycles a cooling fan on and off with a guard on the chamber temperature every ninety seconds:

1. Count the solver restarts over the job, and separate them into those that need a zero-crossing search and those that do not.
2. The fan guard is the one to worry about. If the fan's threshold has no hysteresis and the chamber temperature is noisy around it, describe what the simulator does. Then say what a two-degree hysteresis band changes, in restarts per hour.
3. At an adaptive solver cruising at a one-second step and costing fifteen steps to climb back after a restart, what fraction of the four-hour job is restart overhead, with and without the band?

Hint. Work in restarts per hour before working in steps. The two pauses and the error are three restarts in four hours and are not the problem; the fan at ninety seconds is forty an hour, and a chattering fan is unbounded.

Example 3, posed as a problem: the industrial oven with a conveyor

Field, then reduced. Chapter 6's industrial oven with a conveyor, whose published relative is the induction furnace twin: steel bars passing through a furnace divided into five zones of induction heaters of four coils each, with pyrometers between the coils, entering a holding mode with reduced coil power and the bars oscillating slowly when something downstream stops [36]. A twin of it must answer, for each bar, whether its core reached temperature – a question about the inside of a body, which the pyrometer cannot see and only a field model can answer. Two small published cases, a line-following robot and an incubator, are offered in the literature as the tractable stand-ins for an automated guided vehicle and an industrial convection oven respectively, which is the right scale to prototype this at [37].

The problem. The oven runs a bar through in eight minutes. The field model of one bar is a three-dimensional transient thermal model with about 200,000 elements, and a time-domain run of one bar's passage takes forty minutes on sixteen cores.

1. **Size the field run.** Compute the real-time factor of the field model as stated, and say which of the placements table's four rows it belongs in. Then say what would have to change for it to move up a row, and whether any of those changes is available.
2. **Cost a mesh convergence study.** The quantity of interest is the core temperature at the exit. Design a study over three mesh refinements, say what you would compare and what tolerance would let you stop, and give the total core-hours.
3. **Design the campaign.** A reduced or surrogate model has to run at belt speed – one bar every eight minutes, with an answer before the bar leaves the oven. The inputs are the bar's diameter, its entry temperature, the five zones' power settings and the belt speed: eight inputs. Say why a full factorial is not available here, choose a design, and give its cost in core-hours against the forty-minute run time.
4. **Choose the run mode.** Which of the five run modes does the twin use this surrogate in during normal production, and which does it use during the holding mode the source describes? They are not the same one, and the difference is where the inputs come from.
5. **Name what is still missing.** The surrogate answers for a bar whose inputs lie inside the campaign. A bar of a new alloy arrives. Say what the validity check does, what the twin reports, and what the retraining trigger costs in days.

Summary

Six items, returning to the six objectives.

1. **A simulator is an engine, a step, a clock and a contract** (objective 1). The engine owns the loop and the clock; the step is either a solver over rate rules or a scheduler over an event queue, and hybrid models need both; the clock relates simulated time to wall-clock time and their ratio is the real-time factor; the contract is Chapter 4's interface written out. Four criteria judge a simulator: inside the decision's time budget, restartable from a state the twin supplies, deterministic enough to compare with itself,

and able to say what it ran. The last two are software criteria and are the two most often missing.

2. **The step size is an engineering choice, and stiffness is what forces it** (objective 2). Explicit, Heun, Runge-Kutta, implicit and adaptive, priced in rate evaluations; closing the turbine's torque loop puts a 3.5-second time constant in the same model as a 2000-second one, and an explicit solver must then step at about one second – six hundred steps per SCADA reading – or diverge in two steps. An implicit step at the full six hundred seconds costs one small equation solve and lands on the answer. Algebraic constraints are solved after each step; algebraic loops need an inner iteration nobody budgeted for; and a twin re-initialises every cycle, so initialisation is a per-cycle cost.
3. **A discrete model needs a queue, and one run of a stochastic model answers nothing** (objective 3, objective 4). Next-event advance turns 31.5 million empty seconds into a few thousand events. Then the real work: seeds, independent streams, replication, warm-up, run length, and a confidence interval reported rather than a mean. The vessel's twenty and a half hours is one sample; two thousand two hundred replications pin the mean to half an hour, and ten times the precision costs a hundred times the runs. Hybrid models cost switches, not steps: a zero-crossing search to locate the instant, a solver restart to pay for the reset, and a hysteresis band to stop chattering.
4. **An ensemble is a workload, and its size is arithmetic** (objective 4). Draw the parameter from its posterior once per member, step to the threshold, and the histogram of crossing times is the remaining-life distribution – Chapter 6's three-centre mixture with $N = 3$ and the noise switched off. Common random numbers are what make a comparison affordable; the seed and the container digest are what make a run repeatable; and a tail percentile needs many more members than a mean.
5. **The real-time factor decides where a simulator lives, and the belief decides how far ahead it runs** (objective 5). Below one, batch only – which rules the field model out of the loop entirely, at $RTF = 0.01$, and sends its answer to a reduced model bought with an offline campaign whose grid of five inputs at five levels is 3,125 runs. Around one, pace it. Well above one, divide by the decision's latency to see how many futures fit. And stop predicting at the horizon where the belief has leaked away, which is a property of the system and not of the simulator.
6. **A run is a call that returns numbers and a provenance record** (objective 6). Initial state in, input series in, horizon in, mode in, seed in; simulated-data tuples out, with the binding naming a model version and the `as_of` naming the simulated instant. Five run modes – replay, shadow, predict, what-if, generate training data – distinguished by where the state and inputs come from. Four numerical checks make a run trustworthy as arithmetic: refine the step, swap the solver, check a conserved quantity, record the seed and the versions. None of the four says the answer is right about the asset.

The sentence to carry forward: **this chapter produces the twin's expectation; the estimator, next, is how the twin corrects that expectation with what the sensors actually said.**

Exercises

Solutions or hints follow the exercises; each says which. The objective each exercise serves and its difficulty are given in brackets. None of the exercises uses the plant demonstrator.

Exercise 1 (Objective 2, easy). A thermal model of a transformer has two states: the winding, with a time constant of 90 seconds, and the oil tank, with a time constant of 4 hours. The twin needs the oil temperature once a minute.

(a) Is the model stiff? Give the ratio. (b) What is the largest step an explicit solver can take? (c) How many steps per minute of simulated time is that, and how many of them are there for the oil's sake?
(b) Name the two remedies and say what each costs.

Exercise 2 (Objective 2, medium). You are handed a solver with a relative tolerance setting and a model you did not write. Design the experiment that tells you what tolerance to use, and say what you will report. You have one afternoon.

Exercise 3 (Objective 4, easy). A discrete-event model of a spare parts depot gives, over 400 replications, a mean stockout duration of 11.2 hours with a sample standard deviation of 9.0 hours.

(a) Give the ninety-five per cent interval on the mean. (b) How many replications are needed to pin it to plus or minus one hour? (c) The planner actually wants the ninetieth percentile. Say why (b) is the wrong calculation and what you would do instead.

Exercise 4 (Objective 4, medium). Two maintenance policies are compared by simulation: 500 replications each, seeds drawn independently. Policy B looks 0.4 hours better, with a standard error on the difference of 0.5 hours – not significant. Your colleague proposes running 5,000 replications of each. Propose something cheaper that would probably settle it, say precisely what it requires of the simulator, and say what would make it fail.

Exercise 5 (Objective 3, medium). A pump model has a guard $p \geq 4.0$ bar that switches it from idle to running, and $p < 4.0$ that switches it back. The pressure signal has noise of about 0.1 bar.

(a) Describe what the simulator does when the pressure sits at 4.0.
(b) Give a hysteresis band that fixes it, and say how you chose the two thresholds. (c) Name one alternative remedy and one case where the band is the wrong answer.

Exercise 6 (Objective 2, objective 3, medium). An adaptive solver runs a hybrid model at a cruising step of 0.5 seconds and needs 12 steps to return to cruising after a restart. The model switches mode 20 times an hour, and each switch needs a zero-crossing search costing 6 extra rate evaluations.

(a) What fraction of an hour's rate evaluations is switch overhead?
(b) The model is changed so that it switches 2,000 times an hour. Recompute. (c) At what switch rate does the solver spend half its effort on switching, and what would you do about it?

Exercise 7 (Objective 5, medium). A reduced model covers 1 second of simulated time in 1.2 milliseconds of wall-clock time on the edge box it is deployed to.

(a) Give its real-time factor. (b) The decision it serves has a 30-second latency budget. How many 10-minute what-if scenarios fit?

(b) The model is moved to a container in a cluster where it runs three times faster but the round trip to fetch the state costs 250 milliseconds. Recompute (b) and say which placement you would choose.

Exercise 8 (Objective 6, open-ended). You inherit a twin whose simulator returns arrays of numbers and writes a line to a log file per run. Write the provenance record you would add, field by field, and for each field say what question it answers and what goes wrong without it. Then say how you would test that the record is complete.

Solutions and hints

Exercise 1. (a) 4 hours is 14,400 seconds, so the ratio is $14400/90 = 160$: stiff. (b) About twice the fastest time constant, so roughly 180 seconds – but accuracy on the winding wants perhaps 9 seconds, a tenth of its constant, and whichever is smaller governs, so 9 seconds. (c) Between 7 steps a minute (at the stability bound) and about 7 steps per minute at the accuracy bound – at $h = 9$ seconds, 6.7 steps a minute, essentially all of them there for the winding, since the oil moves by about a thousandth of its range in a minute. (d) An implicit step, which costs an equation solve per step and removes the stability bound but not the accuracy one; or an adaptive solver, which costs a variable per-step budget – awkward on a hard real-time target – and spends its effort where the model needs it.

Exercise 2. Hint. Sweep the tolerance over several orders of magnitude, as the tolerance-sweep study in the Continuous-Time section did, and at each setting record both the quantity of interest and the **number of steps**. Plot the quantity against the tolerance: it will move and then stop moving, and you want the loosest tolerance in the flat region, with a safety margin. Report the step count rather than the wall-clock time if the model is small enough for the timing to be noise, and report both if it is not. Then repeat once with a different solver, because a tolerance that is fine for one is not automatically fine for another.

Exercise 3. (a) $1.96 \times 9.0/\sqrt{400} = 0.88$ hours: 11.2 plus or minus 0.88, so about 10.3 to 12.1. (b) $N = (1.96 \times 9.0/1.0)^2 = 311$ – fewer than you already have, so the interval is already inside one hour. (c) The formula is the interval on the **mean**. A ninetieth percentile is estimated from the upper tail of the sample, so its precision depends on how many replications landed out there – roughly forty of the four hundred – and it needs several times as many replications for comparable precision. Estimate the percentile directly from the order statistics, put a bootstrap interval on it, and size N from that interval rather than from the mean's.

Exercise 4. Re-run both policies with **common random numbers**: the same seeds, paired member for member, so the weather, the arrivals and the failures are identical under both policies and the difference is the policy's. The requirement is independent random streams – policy B must draw its weather from the same stream at the same points as policy A, which means the streams must be indexed by what they are for and not consumed in call order. It fails when the two policies do not consume the same random draws in the same order – for instance if B dispatches a second vessel and thereby asks the weather model a different number of questions – in which case the pairing breaks and you are back to independent runs, or to restructuring the model so that every random draw is requested from a named stream at a fixed point.

Exercise 5. (a) It chatters: the guard fires, the transition changes the state, the reverse guard immediately fires, and the solver truncates step after step while simulated time barely advances – wall-clock time passing with the simulated clock stationary. (b) Switch on at 4.1 bar and off at 3.9; the band must be comfortably wider than the noise, so about two standard deviations either side, and it should also be narrower than any pressure change the twin must detect. (c) A minimum dwell time in each mode, or filtering the guard's input. The band is the wrong answer when the real asset genuinely does switch at a single threshold and the rapid switching is a real behaviour the twin exists to report – a relay welding shut, for instance – in which case the fix is to model the switching, not to suppress it.

Exercise 6. (a) Cruising at 0.5 seconds is 7,200 steps an hour. 20 switches cost $20 \times 6 = 120$ extra rate evaluations for the searches and $20 \times 12 = 240$ steps of climb-back: about 360 against 7,200, or five per cent. (b) 2,000 switches cost 12,000 search evaluations and 24,000 climb-back steps against 7,200 cruising steps – the solver now does five times as much switching work as useful work, and in practice never reaches its cruising step at all. (c) Setting climb-back plus search equal to the cruising steps gives roughly $18\, n_{\text{switch}} = 7200$, so about 400 switches an hour, one every nine seconds. Above that, either the mode structure is too fine-grained for the question and should be coarsened, or the switching is chattering and needs a band, or the model should be reformulated so that the fast switching becomes a continuous approximation.

Exercise 7. (a) $RTF = 1/0.0012 = 833$. (b) 30 seconds of budget covers $833 \times 30 = 25{,}000$ seconds of simulated time, which is about seven hours, or 41 ten-minute scenarios. (c) Three times faster is $RTF = 2500$, but 250 milliseconds of the 30-second budget is gone to the round trip, leaving 29.75 seconds: $2500 \times 29.75 = 74{,}375$ seconds, about 124 scenarios. The cluster wins on throughput by a factor of three. Choose it if the what-if fan is the workload and the round trip is paid once per fan; choose the edge if the same simulator also has to run the shadow mode against a live stream, because there the 250 milliseconds is paid every cycle and the availability of the link becomes a dependency the asset does not otherwise have.

Exercise 8. Hint. The seven fields of the Run Provenance subsection are the answer's skeleton: model and version, solver and settings, initial state and its `as_of`, inputs and whether they were measured, forecast or assumed, seed and stream assignment, environment digest, and cost. For each, the question it answers is "what would change the numbers if it changed" – which is also the test for whether a field is missing. The completeness test is the one in that subsection: hand the record to a colleague with access to the same systems and ask them to reproduce the run. Whatever they have to come back and ask you for is the field you left out. Do it once per twin, not once per run.

Further Reading

Grouped by technique, one line per entry saying what you would go to it for, and saying where a worked example is the book's own construction rather than a source's.

- *What a simulator is:* [1] for the executable digital twin – a self-contained, deployable, fast-enough model carrying its own execution engine, which is the clearest published statement of what a simulator becomes inside a twin; [16] for high-fidelity simulators and reduced-order modelling named as twin enablers, with the challenge-by-technology table worth a read on its own. The four parts and the four criteria are

the book's own framing.

- *Continuous-time execution:* [5] at book length for equation-based modelling, causality assignment, the implicit routine that algebraic loops force, and the Runge-Kutta family; [3] for a measured tolerance sweep and the plain statement that events stop a variable-step solver; [2] for adaptive solvers as the customary answer to stiffness and for what moving a coupling boundary onto the slow variable buys; [6] for equation-based models reaching a running system as compiled artefacts. The turbine's three-ways table and its closed-loop time constant are the book's own.
- *Discrete-event execution:* [5] again for DEVS stated as a formalism, with worked code; [8] for what it takes to turn an existing discrete-event model into a twin, as a manufacturing case study; [9] for a published twin that calls its discrete-event model in an optimisation loop; [7] for formalism choice as a design stage. The vessel ensemble and its replication arithmetic are the book's own.
- *Hybrid execution:* [10] for hybrid automata of machine and process states in manufacturing twins, and for the plain finding that fidelity is paid for in simulation time; [3] and [2] again, for the two halves of what a switch costs – the stopped integration and the search for the instant.
- *Stochastic execution and reproducibility:* [13] for the sampling methods a continuous posterior needs and why they are expensive; [14] for a surrogate built specifically to survive a sampler; [15] for a twin whose output is a family of trajectories and a risk-aware decision taken from them; [11] for computational reproducibility and what a container actually pins; [12] for reproducibility and replicability defined for twin research, and why so little of it is either; [29] for a platform that treats reproducibility as an architectural property.
- *Field solvers:* [20] for the honest statement that a run takes hours or days and that a study needs many of them; [17] and [18] for finite-element model updating from measured natural frequencies, which is the eigenvalue run this chapter says is the field model's most useful one; [19] for a calibrated model computing modes no experiment could reach.
- *Buying the time back:* [22] for a component-based library of reduced-order models and the many-query argument that motivates it; [24] for the definition of a surrogate as a mimic of a model rather than of an asset; [14] for the multi-fidelity trick; [23] for reduced models argued for as reusable assets; [21] for fit-for-purpose as a principle and for the cost of training data named as an open gap. The campaign arithmetic and the validity-check listing are the book's own.
- *Time:* [26] for measured time discrepancy, measured catch-up, and the finding that a threshold set from ideal conditions is exceeded in normal ones – the most useful single paper for this chapter's Time section; [10] for the twinning rate as distinct from the sampling rate; [25] for a turbine twin built to run in real time beside its asset.
- *Driving a simulator, and run modes:* [27] for a compact definition of what-if simulation; [28] for the training-data run mode surveyed in full, and for the observation that most surveyed techniques do nothing to maintain the simulator's own fidelity; [9] again for what-if as an inner loop.
- *Trusting a run:* [30] for where a standard puts model credibility assessment; [31] for verification and validation as continuous rather than as a step; [21] for verification and validation across the whole twin ecosystem. The four numerical checks are the book's own list.
- *The faded examples:* [32] and [33] for equivalent-circuit and algebraic capacity models

of a cell; [34] for a fade model whose parameters are predicted from early life; [35] for the printer's published state machine; [36] for the induction furnace and its holding mode; [37] for two small, fully worked cases at a scale you can rebuild.

- *Deliberately absent:* **co-simulation and distributed simulation**. Running two or more simulators together is a large literature with its own standards, and both the literature and the standards belong to the platforms and standards chapter of Part III rather than here. If you have come to this chapter looking for the Functional Mock-up Interface or the High Level Architecture, that is where they are.

References

[1] D. Hartmann and H. Van der Auweraer, *The Executable Digital Twin: Merging the digital and the physics worlds*. 2022.

[2] M. Wetter, K. Benne, H. Tummescheit, and C. Winther, "Spawn: Coupling Modelica Buildings Library and EnergyPlus to enable new energy system and control applications," *Journal of Building Performance Simulation*, vol. 17, no. 2, pp. 274–292, Mar. 2024, doi: 10.1080/19401493.2023.2266414.

[3] C. Cimino, F. Terraneo, G. Ferretti, and A. Leva, "Efficient Control Representation in Digital Twins: An Imperative Challenge for Declarative Languages," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 11, pp. 11080–11090, Nov. 2023, doi: 10.1109/TII.2023.3242806.

[4] F. Stadtmann and A. Rasheed, "Diagnostic Digital Twin for Anomaly Detection in Floating Offshore Wind Energy." arXiv, Jun. 2024. doi: 10.48550/arXiv.2406.02775.

[5] P. Carreira, V. Amaral, and H. Vangheluwe, Eds., *Foundations of Multi-Paradigm Modelling for Cyber-Physical Systems*. Springer Nature, 2020. doi: 10.1007/978-3-030-43946-0.

[6] J. Hugues, J. Yankel, J. Hudak, and A. Hristozov, "Twinops: Digital twins meets devops," *CARNEGIE-MELLON UNIV PITTSBURGH PA, Tech. Rep.*, 2022, Accessed: Jan. 08, 2025. [Online]. Available: https://apps.dtic.mil/sti/trecms/pdf/AD1168471.pdf

[7] G. Lugaresi and H. Vangheluwe, *From Digital Twins to Twinning Systems*. 2025.

[8] A. Wooley, J. Bitencourt, and D. Silva, "Bridging the gap between discrete event simulation and digital twin: A manufacturing case study☆☆," *Manufacturing Letters*, vol. 44, pp. 1274–1284, Aug. 2025, doi: 10.1016/j.mfglet.2025.06.147.

[9] A. Villalonga *et al.*, "A decision-making framework for dynamic scheduling of cyber-physical production systems based on digital twins," *Annual Reviews in Control*, vol. 51, pp. 357–373, Jan. 2021, doi: 10.1016/j.arcontrol.2021.04.008.

[10] T. Böttjer *et al.*, "A review of unit level digital twin applications in the manufacturing industry," *CIRP Journal of Manufacturing Science and Technology*, vol. 45, pp. 162–189, Oct. 2023, doi: 10.1016/j.cirpj.2023.06.011.

[11] D. Moreau, K. Wiebels, and C. Boettiger, "Containers for computational reproducibility," *Nature Reviews Methods Primers*, vol. 3, no. 1, p. 50, 2023, Accessed: Oct. 25, 2024. [Online]. Available: https://www.nature.com/articles/s43586-023-00236-9

[12] A. Barbie and W. Hasselbring, "Toward Reproducibility of Digital Twin Research: Exemplified with the PiCar-X." arXiv, Aug. 2024. doi: 10.48550/arXiv.2408.13866.

[13] A. Thelen *et al.*, "A comprehensive review of digital twin—part 2: Roles of uncertainty quantification and optimization, a battery digital twin, and perspectives," *Structural and Multidisciplinary Optimization*, vol. 66, no. 1, p. 1, Dec. 2022, doi: 10.1007/s00158-022-03410-x.

[14] M. Torzoni, A. Manzoni, and S. Mariani, "A Deep Neural Network, Multi-fidelity Surrogate Model Approach for Bayesian Model Updating in SHM," in *European Workshop on Structural Health Monitoring*, P. Rizzo and A. Milazzo, Eds., Cham: Springer International Publishing, 2023, pp. 1076–1086. doi: 10.1007/978-3-031-07258-1_108.

[15] A. Chaudhuri *et al.*, "Predictive digital twin for optimizing patient-specific radiotherapy regimens under uncertainty in high-grade gliomas," *Frontiers in Artificial Intelligence*, vol. 6, Oct. 2023, doi: 10.3389/frai.2023.1222612.

[16] A. Rasheed, O. San, and T. Kvamsdal, "Digital Twin: Values, Challenges and Enablers From a Modeling Perspective," *IEEE Access*, vol. 8, pp. 21980–22012, 2020, doi: 10.1109/ACCESS.2020.2970143.

[17] N. Wagner and D. Tcherniak, *Data-Driven Updating of Digital Twins through Experimental Measurements and Parametric Finite Element Model Optimization*. 2025.

[18] U. T. Tygesen, K. Worden, T. Rogers, G. Manson, and E. J. Cross, "State-of-the-Art and Future Directions for Predictive Modelling of Offshore Structure Dynamics Using Machine Learning," in *Dynamics of Civil Structures, Volume 2*, S. Pakzad, Ed., Cham: Springer International Publishing, 2019, pp. 223–233. doi: 10.1007/978-3-319-74421-6_30.

[19] M. G. Kapteyn, J. V. R. Pretorius, and K. E. Willcox, "A probabilistic graphical model foundation for enabling predictive digital twins at scale," *Nature Computational Science*, vol. 1, no. 5, pp. 337–347, May 2021, doi: 10.1038/s43588-021-00069-0.

[20] H. Pezeshki, H. Adeli, D. Pavlou, and S. C. Siriwardane, "State of the art in structural health monitoring of offshore and marine structures," *Proceedings of the Institution of Civil Engineers - Maritime Engineering*, vol. 176, no. 2, pp. 89–108, Apr. 2023, doi: 10.1680/jmaen.2022.027.

[21] *Foundational Research Gaps and Future Directions for Digital Twins*. Washington, D.C.: National Academies Press, 2024. doi: 10.17226/26894.

[22] M. G. Kapteyn, D. J. Knezevic, and K. Willcox, "Toward predictive digital twins via component-based reduced-order models and interpretable machine learning," in *AIAA Scitech 2020 Forum*, in AIAA SciTech Forum., American Institute of Aeronautics; Astronautics, 2020. doi: 10.2514/6.2020-0418.

[23] V. Zambrano *et al.*, *Interpretable and Reusable Reduced Order Models for Digital Twins in Manufactory as a Service*. 2023. doi: 10.23967/admos.2023.070.

[24] E. VanDerHorn and S. Mahadevan, "Digital Twin: Generalization, characterization and implementation," *Decision Support Systems*, vol. 145, p. 113524, Jun. 2021, doi: 10.1016/j.dss.2021.113524.

[25] E. Branlard, J. Jonkman, C. Brown, and J. Zhang, "A digital twin solution for floating offshore wind turbines validated using a full-scale prototype," *Wind Energy Science*, vol. 9, no. 1, pp. 1–24, Jan. 2024, doi: 10.5194/wes-9-1-2024.

[26] M. Frasheri *et al.*, "Addressing time discrepancy between digital and physical twins," *Robotics and Autonomous Systems*, vol. 161, p. 104347, Mar. 2023, doi: 10.1016/j.robot.2022.104347.

[27] L. Esterle, C. Gomes, M. Frasheri, H. Ejersbo, S. Tomforde, and P. G. Larsen, "Digital twins for collaboration and self-integration," in *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*, IEEE, 2021, pp. 172–177. Accessed: May 25, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9599295/

[28] X. Liu and I. David, "AI simulation by digital twins: Systematic survey, reference framework, and mapping to a standardized architecture," *Software and Systems Modeling*, Aug. 2025, doi: 10.1007/s10270-025-01306-0.

[29] J. Grübel *et al.*, "Outlining the Open Digital Twin Platform," in *2023 IEEE Smart World Congress (SWC)*, Aug. 2023, pp. 1–3. doi: 10.1109/SWC57546.2023.10448743.

[30] G. Shao, S. Frechette, and V. Srinivasan, "An Analysis of the New ISO 23247 Series of Standards on Digital Twin Framework for Manufacturing," American Society of Mechanical Engineers Digital Collection, Sep. 2023. doi: 10.1115/MSEC2023-101127.

[31] Z. Ali, R. Biglari, J. Denil, J. Mertens, M. Poursoltan, and M. K. Traoré, "From modeling and simulation to Digital Twin: Evolution or revolution?" *SIMULATION*, vol. 100, no. 7, pp. 751–769, Jul. 2024, doi: 10.1177/00375497241234680.

[32] A. Thelen, M. Li, C. Hu, E. Bekyarova, S. Kalinin, and M. Sanghadasa, "Augmented model-based framework for battery remaining useful life prediction," *Applied Energy*, vol. 324, p. 119624, Oct. 2022, doi: 10.1016/j.apenergy.2022.119624.

[33] A. Thelen, X. Huan, N. Paulson, S. Onori, Z. Hu, and C. Hu, "Probabilistic machine learning for battery health diagnostics and prognostics—review and perspectives," *npj Materials Sustainability*, vol. 2, no. 1, pp. 1–33, Jun. 2024, doi: 10.1038/s44296-024-00011-1.

[34] T. Li *et al.*, "Coupling a capacity fade model with machine learning for early prediction of the battery capacity trajectory," *Applied Energy*, vol. 389, p. 125703, Jul. 2025, doi: 10.1016/j.apenergy.2025.125703.

[35] D. Lehner, S. Sint, M. Eisenberg, and M. Wimmer, "A pattern catalog for augmenting Digital Twin models with behavior," *at - Automatisierungstechnik*, vol. 71, no. 6, pp. 423–443, Jun. 2023, doi: 10.1515/auto-2022-0144.

[36] Y. Ma *et al.*, "Automated and Systematic Digital Twins Testing for Industrial Processes." arXiv, Feb. 2023. doi: 10.48550/arXiv.2302.13198.

[37] R. Paredis, C. Gomes, and H. Vangheluwe, "A Family of Digital T Workflows and Architectures: Exploring Two Cases," in *Innovative Intelligent Industrial Production and Logistics*, A. Smirnov, H. Panetto, and K. Madani, Eds., Cham: Springer Nature Switzerland, 2023, pp. 93–109. doi: 10.1007/978-3-031-37228-5_6.