Chapter 6 – Simulators: How Each Kind of Model Gets Solved

6.0 Before you start

Where we are. Chapter 4 told you what kinds of model exist. Chapter 5 told you what happens when one is executed, using deliberately one stepping rule and calling it the simplest that exists. Between them, Part II has now promised you seven things and delivered none of them:

- the taxonomy of stepping rules (Chapter 5, Sec. 5.4.1);
- what *stiff* means and why it changes your compute bill (Chapter 5, Sec. 5.4.4);
- where the initial state comes from (Chapter 5, Sec. 5.5.1, and before that Chapter 3, Sec. 3.2.3);
- how many runs turn into one answer (Chapter 5, Sec. 5.6);
- what the Functional Mock-up Interface (FMI) actually is (Chapter 5, Sec. 5.7.1);
- what surrogates are for (Chapter 4, Sec. 4.5.3);
- and the machinery for models that are part continuous and part discrete (Chapter 4, Sec. 4.10).

This chapter pays all seven.

The register, stated plainly, because it is the whole design of the chapter. This chapter teaches what each kind of simulator is *for*, what it costs, and what to ask about it. **It teaches no method.** You will finish unable to implement a finite-element mesh or a particle filter, and that is deliberate: Part II's job is to make you a competent interlocutor, not a numerical analyst. What you will be able to do is look at a proposed twin, name what will have to solve each part of it, predict where the compute goes, and ask three useful questions per family.

What you are assumed to know. Everything so far. Especially, from Chapter 5: step size `h`, sampling interval `Ts`, macro step `H`, real-time factor, and the difference between numerical error and being wrong about the world.

The maths budget. Chapter 4 spent the book's one derivative. Chapter 5 spent none. This chapter spends none either: no derivatives, no integrals, **no matrices**, no probability notation. There are exactly two pieces of real arithmetic – a state estimate built as a weighted average of two numbers (Sec. 6.6.2), and a sample-count calculation with a square root (Sec. 6.7.3). Everything else is a table.

What this chapter deliberately does not cover. Any method's derivation or implementation. Calibration and validation – Chapter 7, and Sec. 6.7 is careful that Monte Carlo propagates *stated* uncertainty and cannot discover unstated uncertainty. Machine-learning specifics and their risks – Chapter 8. Tool and platform selection – Chapter 12. Standards in depth – Chapter 13. Optimisation – Chapter 11.

Learning objectives

By the end of this chapter, you will be able to:

1. **Distinguish** method, algorithm, solver and simulator, and **identify** which of the four a colleague means.

2. **Match** a model kind to the family of simulator that solves it, and **predict** which resource – compute, memory, expertise, or wall-clock latency – will dominate.
3. **Explain** what stiffness costs, and **recognise** the three symptoms that suggest a model is stiff before anyone uses the word.
4. **Compute** a corrected state estimate from a model prediction and a noisy measurement, and **state** when a Kalman filter is the wrong tool.
5. **Determine** how many Monte Carlo runs a stated precision requires, and **combine** that with Chapter 5's cost arithmetic to size a service.

6.1 What's in a name? Method, algorithm, solver, simulator

Four words, used interchangeably, meaning four things. The confusion costs real money in procurement conversations, because a vendor selling you a *simulator* and a modeller offering you a *method* are offering inventory that differs by two orders of magnitude in effort.

> **Method** – the mathematical idea. "Assume the rate of change is constant across the step and multiply." Chapter 5's rule was a method.
>
> **Algorithm** – the method written as a procedure, with its bookkeeping: how the step is chosen, how error is estimated, what happens at an event.
>
> **Solver** – an implementation of one or more algorithms, packaged to be called. It has a version, a licence, a performance profile and bugs.
>
> **Simulator** – a solver plus everything around it: loading the model, handling inputs, scheduling events, recording outputs, reporting failure.

Chapter 3 drew the *simulation runner* as one box. That box contains a simulator; the simulator contains a solver; the solver implements algorithms; the algorithms realise methods. Four layers, and you will inherit decisions at every one.

Why the distinction is practical, in three uses.

Estimating. "We need a new method" is a research project. "We need a different solver" is an afternoon, if the interfaces line up. "We need a different simulator" is a migration. When someone says a change is small, ask which layer.

Reproducibility. Chapter 5 required the solver's identity and version in the recorded inputs of a reproducible run. Now you can see why the *method* name is not enough: two solvers implementing the same method give different numbers, because the bookkeeping differs.

Reading the literature and the marketing. Solvers acquire names and reputations. You will hear one named in a design meeting as a general-purpose workhorse in dynamic simulation, chosen for comparisons precisely because it is not specialised to a particular problem [1]. You do not need to know how it works. You need to know that it is a solver, not a method, and to ask what happens when it is swapped.

6.2 The chooser

The rest of this chapter is a tour. Here is the map first, so the tour is navigable rather than merely sequential.

If the model is...	It is solved by...	Section	The cost that usually dominates	First question to ask
Quantities changing smoothly over time	Numerical integration, one step at a time	6.3.1	Compute, if steps are small or the model is big	"Is it stiff?"
The above, plus relationships that must hold exactly at every instant	A solver that handles algebraic constraints alongside rates	6.3.3	Expertise and tooling	"Does it need a Differential-Algebraic Equation (DAE) solver, and does our tool have one?"
Quantities varying across space as well as time	Finite elements, on a mesh	6.3.4	Compute and memory, by a wide margin	"Can we get away with not resolving space?"
Physical components connected together	Equation-based, acausal tooling	6.3.5	Licences and specialist skills	"Who else can maintain this model?"
Things happening at moments – arrivals, failures, batches	Discrete-event simulation	6.4.1	Rarely compute; usually modelling effort	"What is the event list, and what is random?"
A system in one of a few modes, with transitions	State machine execution	6.4.2	Negligible	"What happens on an unhandled transition?"
Concurrent activities contending for resources	Petri nets	6.4.3	Negligible to run, high to get right	"Can it deadlock, and would we see it?"
Many autonomous entities with local rules	Agent-based simulation	6.4.4	Compute, and validation difficulty	"What does one run of this actually tell us?"
An expensive model you must run many times	A surrogate or reduced-order model	6.5	Offline build cost, then almost nothing	"What was it fitted to, and where is it valid?"

If the model is...	It is solved by...	Section	The cost that usually dominates	First question to ask
Any of the above, plus live measurements	A state estimator on top	6.6	Design effort	"What is the gain doing, and does it ever ignore the sensor?"
Any of the above, with uncertain inputs	Monte Carlo over many runs	6.7	Compute, multiplied by the run count	"How many runs, and what precision does that buy?"
Several of the above at once	A co-simulation orchestrator	6.8	Integration effort and debugging	"Who owns each solver, and what is the macro step?"

Two observations before the tour.

Most twins need several rows. Our greenhouse needs continuous physics for the soil, discrete events for the pump, a state estimator to fuse the sensor, and eventually Monte Carlo to answer a what-if. That is four rows, and Sec. 6.8 is about making them run together.

The right-hand column is the deliverable. If you retain one thing from this chapter, retain that column.

6.3 Continuous physics

6.3.1 Stepping rules, and what "order" buys

Chapter 5 used one rule – assume the rate holds constant across the step, multiply, and move on – and demonstrated that halving the step roughly halves the error. That behaviour has a name: the rule is **first order**.

The families you will hear named differ in what they do inside a step, and the payoff is the exponent:

Family	What it does inside a step	Halving `h` divides the error by
Simplest (Chapter 5's)	One rate evaluation, assumed constant	2
Mid-range	Two to four rate evaluations, combined	4 to 16
High order	Many evaluations, combined cleverly	32 or more

How to read that table, which is the only thing you need from it. A higher-order rule costs more per step and buys much more accuracy per step, so it usually wins – it reaches a target accuracy in far fewer, larger steps. Chapter 5's rule-of-thumb consequence changes accordingly: with a first-order rule, ten times less error costs ten times more compute; with a fourth-order rule it costs about 1.8 times more. That difference is why nobody uses Chapter 5's rule in production, and why Chapter 5 said so.

The two knobs you actually set are the ones Chapter 5 introduced: a fixed step, or a tolerance for an adaptive solver. Which family is running underneath is usually the solver's business.

6.3.2 Stiffness, and why it changes the bill

Chapter 5 named *stiff* and promised an explanation. Here it is, in the greenhouse.

The soil dries over **days**. The pump fills the delivery tube in **seconds**. Both are in the same physical system. If you model both with one continuous model and one explicit stepping rule, the step must be small enough for the *fastest* behaviour – Chapter 5 showed what happens above the stability threshold – even though the answer you want concerns the slowest.

Put numbers on it. Suppose the tube fills with a time constant of 2 seconds, so stability demands a step of a second or so. Simulate seven days:

```
7 days = 604,800 seconds
at h = 1 s  ->  604,800 steps
```

Compare Chapter 5's Case B, which used `h` = 15 minutes and needed 672 steps for the same horizon. **Nine hundred times more work, to answer a question about soil.** Figure 6.1 is why.

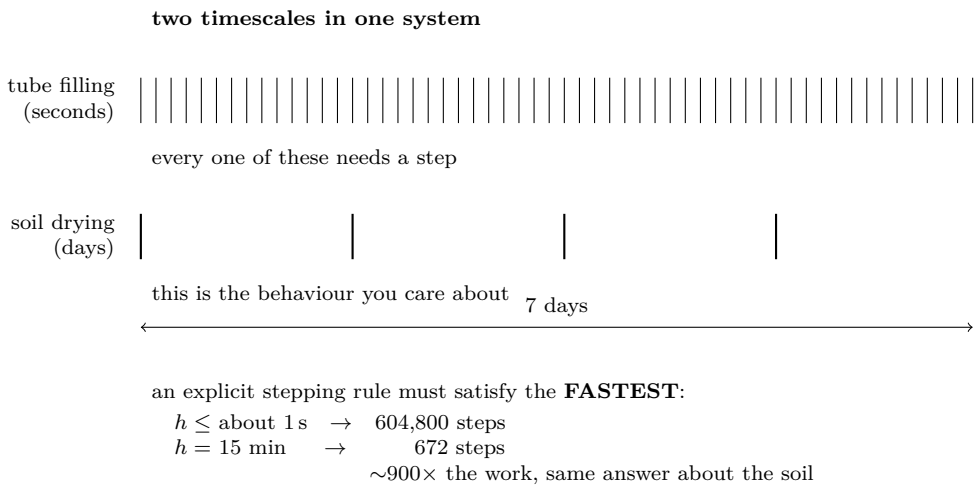

Figure 6.1 Stiffness. The step size is hostage to the fastest behaviour in the model, even when the question is about the slowest.

That is stiffness, and it is why the word is a budget warning. Note the second escape route in the figure: the cheapest fix for a stiff system is often to stop modelling the fast part, which is a modelling decision rather than a solver one.

The distinction that resolves it. Chapter 5's rule is **explicit**: it computes the next state directly from the current one. An **implicit** rule computes the next state by solving an equation that involves the next state – more work per step, and stable at vastly larger steps. Implicit solvers exist precisely for stiff models, and adaptive-step solvers are described as customary for stiff systems that mix continuous and discrete dynamics [2].

Three symptoms that suggest stiffness before anyone says the word:

1. The physical system has behaviour on timescales differing by three orders of magnitude or more – seconds and days, in our case.
2. The run is far slower than the model's size suggests it should be.
3. An adaptive solver reports taking enormous numbers of tiny steps in a period where, physically, nothing is happening.

And the cheapest fix, which is not a numerical one. Do not put the fast behaviour in the continuous model. The pump fills in seconds and the decision is about days; Chapter 4's assumption A3 already said the water mixes fast compared with the sampling interval, so the tube filling can be an *event* rather than a continuous process. Sec. 6.4 is about how events get executed, and Sec. 6.4.5 about mixing the two. **Reformulating the model beats buying a better solver, when you can do it** – and noticing that you can is a modelling conversation you are now equipped to have.

6.3.3 When rates are not enough: algebraic constraints

Some relationships are not about rates of change at all. They must hold exactly, at every instant. In the greenhouse: the water leaving the reservoir equals the sum of the water entering the pots, at all times. Nothing about that is a rate; it is a bookkeeping identity.

A model with both kinds of relationship – rates *and* instantaneous constraints – needs a solver that handles both together. You will hear the combination called a **differential-algebraic** system, and it is standard in engineering practice; lumped-parameter system models are routinely formulated either as pure-rate systems or as this combined kind [3], and equation-based tooling explicitly transforms the implicit combined system before handing it to a numerical solver [4].

What this means for you, in one sentence. Constraints are not free: a model with them needs a more capable solver, and "our tool cannot handle that" is a real answer you may get. Ask early, because discovering it late means changing tools.

6.3.4 Space as well as time

Everything so far treats a quantity as one number for the whole object. Our pot has "the moisture", as though the soil were uniformly wet. It is not: the top dries first, the region under the dripper is wetter, roots draw water unevenly.

Modelling that requires the quantity to vary across **space** as well as time, and the standard machinery is **finite elements**: chop the object into many small pieces, solve on each, and stitch them together. Physics-based simulations of this kind are described as front

and centre in domains such as aerospace research [5], and they are one of the standard families in the mechanical design space alongside multibody dynamics and computational fluid dynamics [3]. Treatments of modelling for twins present the progression explicitly: a simplified model based on rates over time, then a more advanced one that also resolves space [6].

The cost, and it is the biggest jump in this chapter. Chapter 5's Case C made the point with a factor of forty thousand. Resolving space multiplies the state from one number to thousands or millions, and the compute and memory follow. Structural-health work using this machinery routinely reports building fast approximations *specifically because* the full model is too slow to run repeatedly [7] – which is Sec. 6.5's subject and is a strong hint about how often the full model is affordable online.

The question to ask, and it is the first row of the chooser for a reason: *can we get away with not resolving space?* Chapter 1's principle answers it – fidelity is derived from the value metric. Our fault detector compares an expected step against an observed one at one sensor. One number is enough. A model that resolves the pot in three dimensions would be more faithful, cost four orders of magnitude more, and change no decision.

6.3.5 Equation-based, acausal tooling

One more thing you will meet by name. Ordinary programs are **causal**: you write what is computed from what. A resistor model in ordinary code has to decide whether it computes current from voltage or the reverse.

Acausal modelling languages let you write the *relationship* – the equation – and let the tool work out, per use, which variable to solve for. Components then connect like physical components rather than like function calls, which is why the approach dominates in engineering domains where systems are assembled from parts. The best-known such language is Modelica, an equation-based, object-oriented language for modelling physical systems [4], and the surrounding ecosystem is large enough to matter as an argument in its own right – an open standard with many tools, backends, domain libraries and related standards [2].

What to take from this as a software engineer. Two things, and one warning.

It is a genuinely different programming model, and if you review such a model expecting to read execution order, you will not find it. The tool derives it.

It composes physically. Connecting a pump model to a pipe model to a pot model works the way connecting the real components works, which is a real advantage over hand-wiring inputs and outputs.

And the warning: this is specialist tooling with specialist skills and often specialist licences. The chooser's question – *who else can maintain this model?* – is a staffing question, and it is the one that usually decides.

6.4 Discrete and computer-based models

Chapter 4 named these and deferred them. They matter more than their brief treatment in this book suggests, because a great deal of what a twin represents is not a smoothly varying quantity at all.

6.4.1 Discrete-event simulation

Chapter 5's loop advanced a clock in fixed steps and asked at each one whether anything had happened. For a system where things happen rarely, that is almost all wasted work.

Discrete-event simulation inverts it: keep a time-ordered queue of pending events, jump straight to the next one, process it, schedule whatever it causes, repeat. Simulated time leaps rather than steps.

A trace from the greenhouse, showing the queue rather than a clock:

```
queue: [09:16 water(plant1,100ml), 09:16 water(plant2,80ml), 17:05 water(plant1,120ml

pop 09:16 water(plant1,100ml)
    -> relay coil 0 closes; schedule 09:16:03.5 stop(plant1)
pop 09:16 water(plant2,80ml)
    -> relay coil 1 busy; enqueue behind; schedule retry 09:16:03.5
pop 09:16:03.5 stop(plant1)
    -> coil 0 opens; the queued plant2 request proceeds
...
```

Between 09:16:03.5 and 17:05 the simulation does nothing at all, because nothing happens. Seven hours pass in zero steps.

This is the natural formalism for factory flow, queues, batches and resource contention, and it is used that way in twin practice – reviews of smart-manufacturing twins report discrete-event simulation as the tool for process flow design and planning, including in non-automated processes [8], and roadmaps for twin construction list it among the formalisms to be chosen deliberately at design time alongside differential equations and Petri nets [9]. There is an established formalism behind it, used for queueing networks and performance models and described as something close to a simulation assembly language [4]; you do not need it, you need to recognise the name.

What to ask: *what is the event list, and what is random?* If arrival times or durations are drawn from distributions, one run is one sample – which is Sec. 6.7's problem, and it arrives here first.

6.4.2 State machines

The smallest formalism in the chapter, and the one you already use. A system in one of a few named modes, with transitions between them: the pump is `idle`, `pumping`, or `faulted`; the greenhouse is `day`, `night`, or `maintenance`.

State machines cost nothing to execute and are the right representation whenever the interesting behaviour is *which mode we are in* rather than *what value a quantity has*. Practical modelling stacks list them alongside continuous-time and discrete-event models as one of the techniques from which a twin's overall model set is assembled [10], and there

are catalogues of patterns for augmenting twin models with exactly this kind of behaviour [11].

What to ask: *what happens on an unhandled transition?* A state machine that silently ignores an impossible event is a state machine that will diverge from the physical twin and never say so – which, note, is Chapter 4's silent failure wearing different clothes.

6.4.3 Petri nets

When several activities run concurrently and contend for shared resources, plain state machines multiply out badly. **Petri nets** describe such systems with *places* holding *tokens*, and *transitions* that consume tokens from some places and produce them in others. Concurrency and contention are expressed directly rather than enumerated.

The greenhouse has a real instance. One relay module has four coils; six plants want watering; the twelve-volt supply serves all the pumps. A token in the "coil free" place is consumed when watering starts and returned when it stops. The interesting questions are then structural: can two plants water simultaneously? Can a request wait forever? Petri-net-based discrete-event methods are used to build twins of manufacturing systems for precisely this class of question [8], and they appear alongside differential equations and event formalisms in the list of languages a twin project chooses from [9].

What to ask: *can it deadlock, and would we notice?* The payoff of this formalism is that the question is answerable by analysis rather than by running the system and hoping – which is the same payoff you get from a type system, and will be a familiar trade.

6.4.4 Agent-based simulation

Many autonomous entities, each following local rules, with the system's behaviour emerging from their interaction. Traffic, crowds, markets, ecosystems, robot fleets.

It is a poor fit for one pot, and worth naming for two reasons. First, because you will meet it, and its analytical treatment is an active topic – data assimilation techniques for state estimation have been extended to discrete-event and agent-based models [12]. Second, because its characteristic difficulty is a good lesson in miniature: an agent-based model is easy to build, easy to make plausible, and hard to validate, because there is rarely a measurement of the emergent quantity it predicts.

What to ask: *what does one run of this tell us?* Usually the honest answer is "very little" – these models are almost always run many times (Sec. 6.7), and a proposal that shows you a single agent-based run as evidence is showing you an anecdote.

6.4.5 Hybrid: continuous plus discrete

Our pot has been hybrid all along. The soil dries continuously; the doses are events. Chapter 5 handled this informally by testing for a dose inside each step, and Chapter 5 also showed the price: the run behaved as though a 17:05 dose happened at 16:20.

The principled version interleaves the two properly. The execution loop alternates advancing continuous behaviour with checking for and processing discrete transitions, and terminates on reaching a time limit, a state configuration, or another criterion – with the system's mode and both its discrete and its continuous variables all liable to change at

each iteration [6]. The formal machinery for combining discrete and continuous behaviour in one model is a named subject in its own right [4], [6].

The engineering consequence, which is Chapter 5's Sec. 5.7.3 arriving from the other side. Somebody must decide what happens when an event lands in the middle of a continuous step. Chapter 5 showed that the naive answer silently moves the event by up to a step. A proper hybrid simulator will instead *stop the continuous step at the event*, handle it, and restart – which is more correct and, per Chapter 5's Sec. 5.4.3, exactly the pattern that degrades an adaptive solver's efficiency. There is no free choice here, only a stated one.

6.5 Data-driven: surrogates, reduced-order models, and physics-informed machine learning

Chapter 4 introduced surrogates and said they belong here, "where speed becomes a design constraint". Now it is.

6.5.1 What a surrogate is for

A **surrogate** is a fast approximation fitted to a slower model's outputs. Chapter 4's warning stands and is worth repeating because it is the most misunderstood point in the family: **a surrogate is fitted to a model, not to reality, so it inherits every one of that model's assumptions and adds approximation error of its own. It solves a speed problem, not an accuracy problem.**

The speed problem is the one Sec. 6.3.4 created. A **reduced-order model** is a surrogate built by discarding structure the answer does not depend on; such models sit at the intersection of high-fidelity physics simulation and data-driven modelling, and trade some accuracy for much greater computational speed [13]. The economics are explicitly offline-for-online: computational time is invested once, in advance, so that models otherwise intractable in a real-time or many-query context can be used [3], [14].

Where they show up in twins. Wherever the expensive model must run repeatedly: model updating driven by sampling procedures that need thousands of evaluations [7], evolutionary optimisation whose function-evaluation counts are otherwise prohibitive [15], and twins built from libraries of component-level reduced-order models so that a fleet can be served at scale [16].

Two questions to ask, and the second is the one people forget:

What was it fitted to, and over what range of conditions? Outside that range you are extrapolating, with Chapter 4's silent failure.

What happens when the underlying model changes? The surrogate is now stale, and refitting it is an offline job. A base model that is refitted weekly and a surrogate refitted quarterly is a twin that is quietly running last quarter's physics.

6.5.2 Physics-informed machine learning

Chapter 4 named this as the *constrained learning* form of hybrid modelling and deferred the mechanics. The mechanics, at the level this book needs: the fitting procedure is given an extra penalty for producing predictions that violate known physics, which pushes the fitted model toward physically compliant answers – reported as the commonest route to a hybrid physics-and-machine-learning model [14].

What it buys, and the honest limit. It buys better behaviour where data is thin, because physics fills the gap. It does not buy immunity from Chapter 4's extrapolation problem: reviews note that predictions from physics-informed models and from models with online updating can still fail unexpectedly on inputs unlike anything in training, and that such failures reduce trust and limit real-world adoption [15].

That limit is why Chapter 7 exists, and why Sec. 6.6's estimator – which keeps pulling the model back toward measurements – matters even when the model is good.

6.6 Keeping the twin in sync: state estimation

This is the third time the book has deferred this component. Chapter 3 drew it, Chapter 4 distinguished it from the model, Chapter 5 said a run's answer depends on its initial state as much as on its model. Here is the mechanism.

6.6.1 The problem

You have a model that predicts the pot's moisture. You have a sensor that reports it. They disagree. The model is approximate; the sensor is noisy. **Which do you believe?**

"The sensor, it is real" is wrong, and seeing why is the point. A single reading carries the sensor's noise. A model prediction carries the model's error but averages over the noise of every reading that fed it. The right answer is neither: it is a *combination*, weighted by how much you trust each.

That is what a state estimator does, and the twin literature treats it as core – monitoring the physical twin's states is described as the ability that delivers advanced visualisation, fault detection and reduced maintenance cost, with the estimator providing an estimate that is better than the one obtained from a single measurement alone [17].

Every estimator in this chapter runs the same two-step cycle:

> **Predict.** Advance the model from the last estimate to now. **Correct.** Pull the prediction toward the new measurement, by an amount reflecting how much each is trusted.

Figure 6.2 traces one turn of that cycle, and the thing to watch is where the estimate lands: never on the model's prediction, never on the reading, always somewhere between.

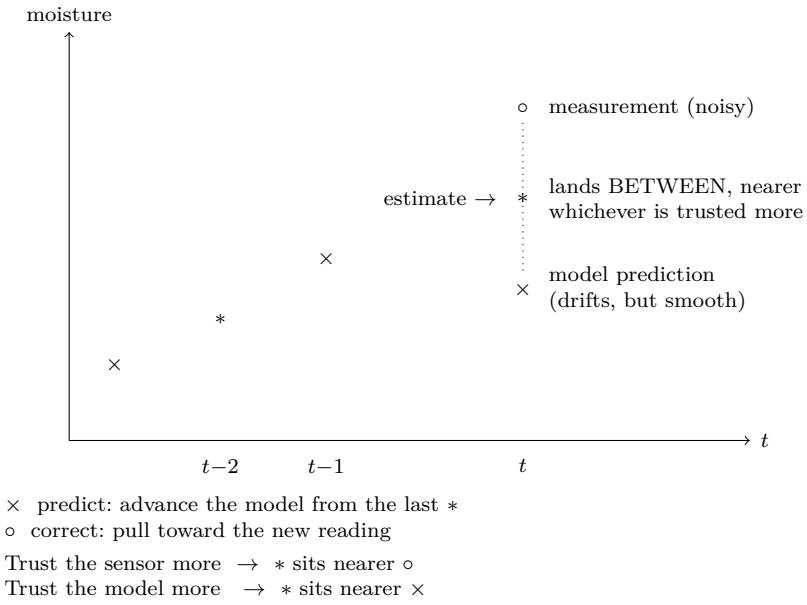

Figure 6.2 One predict-correct cycle. The estimator's output is a third quantity, and believing either input on its own is the mistake Sec. 6.6.1 opens with.

That third quantity is the answer to "which do you believe?", and it is why Chapter 3 drew the state estimator as its own component rather than folding it into the store: it produces something neither the connector nor the model has.

6.6.2 The correction, as arithmetic

Here is the entire idea, on our pot, with no matrices.

At 14:00 the last estimate and the model give a prediction. The sensor reports a reading. Each comes with a **spread** – a number saying how far off it is typically likely to be:

```
model prediction    m_pred = 636    spread 8
sensor measurement  m_meas = 630    spread 5
```

Weight each by one over its spread squared – more certain means more weight:

```
w_pred = 1 / 8^2 = 1/64 = 0.015625
w_meas = 1 / 5^2 = 1/25 = 0.04
```

Combine them as a weighted average:

```
estimate = (0.015625 * 636 + 0.04 * 630) / (0.015625 + 0.04)
         = (9.9375 + 25.2) / 0.055625
         = 631.7
```

And the spread of the combination:

```
spread = square root of ( 1 / (0.015625 + 0.04) ) = 4.24
```

Read those two results, because both are surprising the first time.

The estimate is 631.7, closer to the measurement than to the prediction. That is right: the measurement was the more certain of the two, so it pulled harder.

The spread is 4.24 – smaller than either input's. Combining two independent imperfect sources gives you something better than the best of them. This is the reason state estimation exists, and it is why a twin can report a state more accurate than its own sensor.

The gain. The same result, rewritten the way you will see it in a design document:

```
gain     = w_meas / (w_pred + w_meas) = 0.04 / 0.055625 = 0.72
estimate = prediction + gain * (measurement - prediction)
         = 636 + 0.72 * (630 - 636) = 631.7
```

The **gain** is how far toward the measurement the correction pulls: 0 means ignore the sensor, 1 means ignore the model. Here it is 0.72, meaning "trust the sensor about three quarters of the way".

That is the Kalman filter. The real one does this for many state variables at once, which is where the matrices come in, and it computes the spreads as it goes rather than being handed them. The idea is the arithmetic above. It is presented in twin-specific tutorials in exactly this predict-then-combine shape, on a running example of a small physical system [17], and it is not confined to teaching examples. One reported offshore deployment builds its twin on physical models, checks the result against measurements from a real turbine at full size, and uses a Kalman filter over a linearised model to recover structural quantities that are not directly instrumented [18].

Why you should care about the gain specifically. It is inspectable, and it is diagnostic. A gain that has drifted near 1 means the twin has stopped believing its model – so why keep it? A gain near 0 means the twin is ignoring its sensor, which is either a correct response to a broken sensor or a twin that has stopped tracking reality. Chapter 3's monitoring should carry it, and Chapter 7 will ask about it.

6.6.3 When the Kalman filter is the wrong tool

It is optimal under conditions, and the conditions are real. The most commonly-stated limit is bluntly practical: only linear systems can use a Kalman filter, and a non-linear system needs something else [19].

The alternative you will hear named is the **particle filter**: instead of one estimate with a spread, carry a cloud of sampled possible states, advance them all through the model, and weight them by how well each explains the measurement. It is chosen in twin work for two stated reasons: it makes fewer assumptions about non-linearity and about the shape of the noise than a Kalman filter does, and it composes well with surrogate models [14]. Reviews of predictive maintenance report both in use, as statistical approaches to noise reduction, with the particle filter reached for when the system is non-linear [19], and battery health work compares them directly [20].

The cost, and it is the one that lands on you. A particle filter runs the model once per particle per step. A thousand particles means a thousand times the simulation cost – which is Chapter 5's Sec. 5.3.4 arithmetic again, and it is why Sec. 6.5's surrogates and this section keep appearing in the same sentence in the literature.

6.6.4 What to ask

Three questions, in order of how often the answer is interesting:

1. **What is the gain doing over time?** A drifting gain is the single most informative number a twin can show you about its own health.
2. **What happens when a measurement does not arrive?** The estimator should predict without correcting, and its spread should *grow* – which is Chapter 2's age-of-twin acquiring a number, and Chapter 3's staleness guard acquiring a threshold it can actually test.
3. **What happens when a measurement is wildly wrong?** A sensor reporting nonsense should be rejected, not averaged in. Ask whether there is such a check, because the arithmetic of Sec. 6.6.2 has no opinion about it.

6.7 Uncertainty and what-if at scale: Monte Carlo

6.7.1 One run is not an answer

Chapter 5 said a seed makes a run repeatable, not correct, and handed the consequence here. The consequence is this: if anything feeding a simulation is uncertain – and something always is – then one run tells you what happens for *one* set of guesses.

Our pot: `k` was fitted from data and is not exactly 4. Call it somewhere between 3.5 and 4.5. The question "will moisture be below 500 in 24 hours if I water 100 ml now?" has no single answer, because the answer depends on which `k` is true.

Monte Carlo is the blunt, general, entirely reliable response: sample the uncertain inputs, run the simulation once per sample, and treat the spread of outcomes as the answer.

```
for run in 1..N:
    draw k from its range
    draw any other uncertain input
    simulate
    record the outcome
report: the fraction of runs below 500, and the spread
```

The output is no longer "moisture will be 512". It is "moisture will be below 500 in about 10% of plausible cases". For every decision in this book so far, the second statement is the useful one.

6.7.2 Why this is the workhorse

Because it needs nothing from the model. It does not care whether the model is continuous, discrete, agent-based, learned or a co-simulation of all four. It treats the simulator as a black box and samples around it. That generality is why it appears across every twin domain in the corpus: propagating uncertainty sources to the uncertainty of outputs [15], sampling procedures updating the probability distribution of a structural state from noisy observations [7], [21], posterior trajectories that let a clinical twin quantify risk for risk-aware decision-making [22], and probabilistic foundations that keep updating principled and able to scale to a fleet [16].

It is also **embarrassingly parallel**: every run is independent, so a thousand runs on a hundred cores take ten runs' worth of wall-clock time. That is a rare and welcome property, and it is yours to exploit rather than the modeller's.

6.7.3 How many runs? The cost law you must know

This is the section's practical payoff and the chapter's second piece of real arithmetic.

Monte Carlo error shrinks with the **square root** of the number of runs. Concretely, if the true answer is around 10% and you do 1,000 runs, the uncertainty in your estimate of that 10% is:

```
square root of ( 0.1 * 0.9 / 1000 ) = 0.0095
```

– about **one percentage point**. So 1,000 runs supports "about 10%, give or take 1". It does not support "10.3%".

Now the consequence that decides architectures:

```
to halve the error:        4 times the runs
to divide it by ten:     100 times the runs -> 100,000 runs
```

Combine that with Chapter 5's cost arithmetic and the design falls out. At Chapter 5's Case B cost – 672 steps per run, a microsecond a step – 100,000 runs is about 67 seconds. Fine. At Case C's cost of 40 milliseconds a step, the same 100,000 runs is 2.7 million seconds, or **31 days**. Not fine, and no amount of engineering closes a gap that size.

Which is why Sec. 6.5 exists. Surrogates and Monte Carlo are a matched pair: the sampling makes the many-run demand, the surrogate makes the runs affordable. When you see one in a proposal, look for the other.

6.7.4 The honest limit

Monte Carlo propagates the uncertainty you **state**. It cannot discover uncertainty you did not state.

If you sample `k` between 3.5 and 4.5 but the true `k` is 6 because the plant was repotted, every one of your hundred thousand runs is wrong, and the tight spread they produce will look like confidence. Reports on the foundations of the field make the related point about method limits directly – sampling becomes extremely inefficient for low-probability events, which is exactly where a risk question often lives [23].

Deciding what to sample, and whether the ranges are right, is calibration and validation. That is Chapter 7, and this is the last time this book will defer it.

6.8 Making models work together: orchestration and FMI

Chapter 5 described the co-simulation loop and its three decisions, and said you would hear FMI named. Here it is.

6.8.1 What the standard is for

Once a twin has models from several tools – an agronomy group's soil model, a supplier's pump model, your controller – something must let them be stepped together without each knowing the others' internals. The **Functional Mock-up Interface (FMI)** is the widely used standard for that. Models are packaged as **Functional Mock-up Unit (FMU)** components, which can then be handed between tools and stepped alongside one another [24]. The standard's contribution is a fixed set of calls: an FMU exposes them, and whatever environment is orchestrating the run drives the FMU through them – usually with several other models being driven the same way at the same time [25]. Its practical value is breadth: a broad span of modelling and simulation tools support it and can export models as FMUs [6], and it is what open twin frameworks build their co-simulation on [26], [27].

6.8.2 The one distinction to remember

FMI comes in two flavours, and the difference is entirely about **who owns the solver**:

Variant	The FMU contains	The orchestrator must
FMI for **Co-Simulation**	Its own solver	Ask it to advance by a macro step `H`
FMI for **Model Exchange**	Only the equations	Perform the numerical integration itself

Figure 6.3 draws the same split as a question about where the solver box sits, which is the form you can check against a supplier's deliverable.

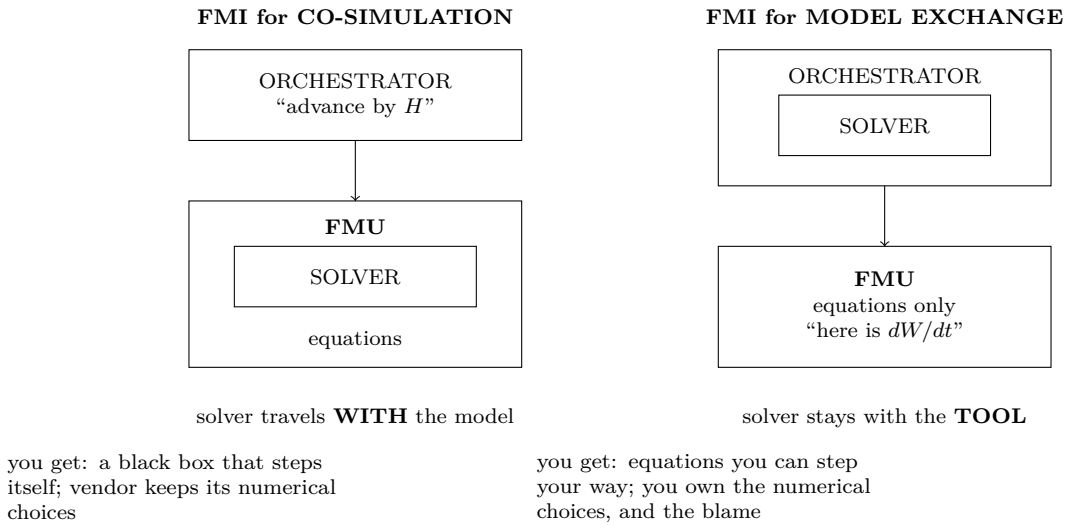

Figure 6.3 The one distinction in FMI. Ask a supplier which variant they ship and you have learned who owns the solver – and therefore who owns the step size.

That is the split, stated in the standard's own terms: an FMU either carries its own solver or leaves the numerical integration to the simulation environment [25]. The practical

consequence is Chapter 5's: whoever owns the solver owns the step size, and the step size is what Sec. 5.4.4 showed can turn a wrong answer into a meaningless one.

Why this is the distinction worth carrying. It determines who is responsible for everything in Sec. 6.3. With co-simulation FMUs, each supplier chose their own step size and stability regime and you cannot see inside; your job is the macro step and the ordering, and debugging a numerical problem means asking a supplier. With model-exchange FMUs, you own the integration, which is more work and far more control. When someone says "we will just use FMI", ask which one – the two produce different projects.

6.8.3 What the orchestrator does, and what it does not fix

Chapter 5 gave the loop: load, initialise and exchange initial outputs, set time and step, exchange values, advance, repeat [6]. Concrete implementations follow that shape, with a master algorithm managing the inter-operating units and a separate component handling data exchange [24]. A co-simulation framework may allow continuous-time and discrete-event models to be integrated together, over FMUs [26], which is Sec. 6.4.5's hybrid problem being solved at the level of whole models rather than inside one.

What a standard does not fix, and Chapter 5 already warned you: the macro step, the ordering, and the event placement remain your decisions, and practitioners report the first as genuinely difficult [28]. FMI makes the models *pluggable*. It does not make the coupling *correct*. There is research on monitoring at runtime whether master algorithms are being used correctly [29], which tells you something about how easy it is to get wrong.

6.9 Worked example: choosing simulators for the greenhouse twin

Everything above, applied. The greenhouse: six pots, one controller, one reservoir, a shared relay module, and Chapter 1's diagnose value case with Chapter 2's Stage 3 twin on the roadmap.

Go through it component by component, using the chooser of Sec. 6.2 and justifying each choice by the *decision*, never by the physics available.

6.9.1 The soil

What it is: one quantity changing smoothly over time. Chapter 4's two-parameter model.

What solves it: numerical integration, Sec. 6.3.1.

Which rule? The model is small and smooth and the horizon is short. Even Chapter 5's first-order rule is adequate at $\mathtt{h} = 15$ minutes; a mid-range rule costs nothing extra worth measuring and removes a class of question, so take one from the library rather than writing the loop.

Is it stiff? No – one timescale, days. Sec. 6.3.2's three symptoms are all absent. **Record that finding**, because it is the answer to a question someone will ask in six months.

Do we resolve space? No. Sec. 6.3.4's question, answered from the value metric: the decision compares an expected step at one sensor against an observed one. A three-dimensional soil model changes no decision and costs four orders of magnitude.

6.9.2 The watering events

What it is: things happening at moments – doses, pump runs, coil occupancy.

What solves it: discrete-event execution, Sec. 6.4.1.

The design decision this forces, and it is the one from Sec. 6.3.2: do **not** put the 3.5-second pump run into the continuous soil model. Doing so makes a non-stiff model stiff and multiplies the step count by about nine hundred, to answer a question about days. Model the pump run as an event that delivers a volume.

Is it random? In the open-loop twin, no: the schedule is fixed and known. Note this, because it means Sec. 6.7 is not needed for the first increment.

6.9.3 The relay contention

What it is: four coils, six plants, concurrent requests.

What solves it: a Petri net would express it exactly, Sec. 6.4.3 – and it is worth being honest that with four coils and six plants, a state machine or a queue in ordinary code is very likely enough.

The judgement, and it is the general one: **reach for the heavier formalism when the question you want to ask is structural.** "Can a watering request be starved indefinitely?" is a question a Petri net answers by analysis and code answers by testing and hoping. If nobody is asking that question, do not build it.

6.9.4 The estimator

What it is: one sensor, noisy, sampled every ten minutes, plus a model.

What solves it: the predict-correct cycle of Sec. 6.6, scalar, exactly the arithmetic of Sec. 6.6.2 – one state variable, so no matrices are involved even in the real implementation.

Kalman or particle? Kalman. The model is linear over the interval (Chapter 4's A1), which is the condition Sec. 6.6.3 named. A particle filter would cost a thousand times more and answer the same question.

What we monitor: the gain, per Sec. 6.6.4. It is the twin's own health indicator, and it is free.

6.9.5 The what-if service, when it arrives

What it is: "if I skip tomorrow's morning dose, what is the chance the pot drops below 500?"

What solves it: Monte Carlo over the uncertain `k` and `g`, Sec. 6.7.

How many runs? Work it, per Sec. 6.7.3. A percentage point of precision on a roughly-10% answer needs about 1,000 runs. At 672 steps and a microsecond a step, that is under a second. **No surrogate needed** – which is the useful negative result, and it holds only because Sec. 6.9.1 refused to resolve space.

6.9.6 Do we need co-simulation?

The honest answer for the first increment: no. The soil model and the event handling are both small enough to live in one service, written by one team, in one language. Chapter 5's three decisions do not arise if there is nothing to couple.

What would change it: a supplier-provided pump model as an FMU; an agronomy group's soil model in Modelica; a controller model that must be the *actual* controller firmware rather than a reimplementation. Any of the three brings Sec. 6.8, and with it the macro step, the ordering, and a debugging story that spans two organisations.

The general lesson from Sec. 6.9, and it is the point of the whole chapter. Every choice above was made by asking what the decision needed, and five of the six answers were the *cheaper* option. A chapter that catalogues simulators can read as an invitation to use them. It is the opposite: knowing what each family is for is mostly how you find out you do not need it.

6.10 Faded example: the offshore turbine, solved

The turbine service from Chapters 3, 4 and 5: streaming vibration and temperature, a hybrid model whose physics half accumulates fatigue and whose learned half maps vibration to condition, a work order to a planner.

Worked, for the first two components.

The fatigue half. Continuous, over years, driven by irregular storm loading. Chapter 5 identified this as an adaptive-stepping case. Is it stiff? Judge by Sec. 6.3.2's symptoms: the timescales present are storm gusts (seconds) and fatigue accumulation (years) – nine orders of magnitude, which is symptom 1 emphatically. The resolution is Sec. 6.9.2's: do not put the fast behaviour in the slow model. Summarise each storm into a loading contribution and step the fatigue model on the slow timescale.

The structural response. This is where finite elements appear (Sec. 6.3.4) and where the cost lands, which is why the literature on this exact application is full of surrogates and reduced-order models [7], [21], [30].

Now it is your turn.

- **(a)** The condition of interest – crack length, bearing wear – is not measured by any sensor, and vibration is. Name the component of Sec. 6.6 that bridges that gap, and say why a Kalman filter may not be the right choice here. Use Sec. 6.6.3's stated criterion, not your intuition.
- **(b)** The team proposes a particle filter with 2,000 particles, stepping every 10 minutes, where one model evaluation takes 30 milliseconds. Compute the wall-clock cost per estimation step, and say whether it fits. Then name the Sec. 6.5 device that would change the answer.
- **(c)** They want to answer "what is the probability the tower exceeds its fatigue limit within five years?" to within two percentage points. Using Sec. 6.7.3, estimate the number of runs, then use your answer to (b) to estimate the compute. State the one thing that would make this feasible.

- **(d)** The supplier ships the drivetrain model as an FMU. Ask the Sec. 6.8.2 question and describe how each of the two answers changes the project.

Hint for (b): it is one multiplication, and the answer is decisive rather than marginal.

6.11 Posed problem: the simulator selection memo

A water utility wants a twin of a pumping station: four pumps, a wet well whose level varies continuously, inflow that arrives as storm events, pumps that switch between `off`, `starting`, `running` and `faulted`, a shared electrical supply with a limit on simultaneous starts, and a regulator who requires the utility to demonstrate the probability of an overflow in a design storm.

Write the simulator selection memo. For each part of the system, state:

1. Which row of Sec. 6.2's chooser it falls under, and why.
2. What will solve it, and what you expect to dominate the cost.
3. Whether the parts should be one program or a co-simulation, with the Sec. 6.8.2 question answered for any external model.
4. For the regulator's probability question specifically: the number of runs the stated precision requires, the compute that implies, and what you would do if that number is unaffordable.

There is no single right memo. There is a wrong one, and it is the memo that picks the most capable option in every row.

6.12 Summary

Against the objectives.

1. **Four words, four layers** (Sec. 6.1). Method, algorithm, solver, simulator. Ask which layer a proposed change touches; the estimates differ by orders of magnitude, and the reproducibility record needs the solver, not the method.
2. **The chooser** (Sec. 6.2) maps model kinds to simulator families and to the question to ask about each. Continuous physics is integrated step by step (Sec. 6.3); events are executed from a queue (Sec. 6.4); expensive models get fast stand-ins (Sec. 6.5); live measurements need an estimator (Sec. 6.6); uncertain inputs need many runs (Sec. 6.7); and several at once need an orchestrator (Sec. 6.8).
3. **Stiffness is a budget warning** (Sec. 6.3.2). Behaviour on very different timescales forces an explicit rule to step at the fastest – nine hundred times the work, in the greenhouse. Three symptoms give it away, and the cheapest fix is to reformulate the model rather than to buy a better solver.
4. **State estimation is a weighted average** (Sec. 6.6.2). Weight the model prediction and the measurement by one over their spreads squared and combine: the result is closer to whichever is more certain, and *more certain than either*. The gain is the twin's own health indicator, and it is free to monitor. Kalman when the model is linear; a particle filter when it is not, at a thousand times the cost.

5. **Monte Carlo error falls as the square root of the run count** (Sec. 6.7.3). A percentage point on a 10% answer needs about a thousand runs; ten times better needs a hundred thousand. Multiply by Chapter 5's per-run cost and the architecture decides itself – and Monte Carlo propagates only the uncertainty you *stated*.

The sentence to carry forward: **knowing what each simulator family is for is mostly how you discover you do not need it** – and its companion, from Sec. 6.3.2, **reformulating the model beats buying a better solver.**

6.13 Exercises

Objectives in brackets. Solutions and hints follow.

6.13.1 (Objective 1, easy). Classify each as method, algorithm, solver or simulator: (a) "assume the rate is constant across the step"; (b) a commercial product that loads a model file, runs it and plots results; (c) a named library routine with a version number; (d) a procedure that estimates its own error each step and halves the step if it is too big.

6.13.2 (Objective 2, easy). For each, name the chooser row and the family that solves it: (a) the temperature profile through a furnace wall; (b) trolleys arriving at a supermarket checkout; (c) a lift that is `idle`, `moving` or `doors-open`; (d) the same lift plus three others contending for one shaft controller; (e) a wind-farm layout evaluated ten thousand times by an optimiser.

6.13.3 (Objective 3, easy). A colleague reports that their model takes four hours to simulate one day, and that the adaptive solver reports millions of steps overnight when the plant is switched off. Name the likely diagnosis and the two things you would ask next.

6.13.4 (Objective 4, medium). Redo Sec. 6.6.2's arithmetic with the sensor's spread changed from 5 to 12, the model's unchanged at 8. Compute the estimate, the combined spread and the gain. Then say in one sentence what changed about the twin's behaviour, and which of Sec. 6.6.4's three questions this makes concrete.

6.13.5 (Objective 5, medium). A what-if service must report the probability of a threshold breach to within half a percentage point, where the probability is around 5%. Compute the number of runs. At 2,000 steps per run and 50 microseconds per step, compute the wall-clock time on one core and on 64 cores.

6.13.6 (Objective 2, medium). Sec. 6.9.3 declined to build a Petri net for four coils and six plants. Construct a version of the greenhouse where that decision flips, and state the question that forces it.

6.13.7 (Objective 3, medium). Sec. 6.3.2 says reformulating beats buying a better solver "when you can do it". Give one example from any system where you *cannot* – where the fast behaviour genuinely must stay in the continuous model – and say what you would do instead.

6.13.8 (Objectives 4 and 5, hard). Complete Sec. 6.10, parts (a) through (d).

6.13.9 (Objective 4, hard). Sec. 6.6.4's second question asks what happens when a measurement does not arrive. Design that behaviour for the demonstrator: what the

estimator does, what happens to the spread, what Chapter 3's actuation guard should do as the spread grows, and what an operator sees. Then say how you would test it.

6.13.10 (Objectives 1 through 5, hard, open-ended). Do Sec. 6.11, the simulator selection memo, in full.

Solutions and hints

6.13.1. (a) method; (b) simulator; (c) solver; (d) algorithm – note it is the *bookkeeping around* a method, which is exactly Sec. 6.1's distinction, and note also that two solvers implementing this same algorithm will still differ in the last bits, which is why Chapter 5's reproducibility record names the solver.

6.13.2. (a) Space as well as time – finite elements, Sec. 6.3.4; though a good answer asks first whether one number per wall layer would do. (b) Discrete-event, Sec. 6.4.1, and it is random, so Sec. 6.7 applies. (c) State machine, Sec. 6.4.2. (d) Petri net, Sec. 6.4.3 – the shared controller is the contended resource, and "can a lift wait forever?" is the structural question that justifies it. (e) Surrogate, Sec. 6.5: ten thousand evaluations is the many-query case those exist for.

6.13.3. Stiffness. Symptoms 2 and 3 of Sec. 6.3.2 are both present, and symptom 3 is decisive – tiny steps while nothing physically happens is the signature. The two questions: *what is the fastest behaviour in the model, and does the decision depend on it?* and *is the solver explicit or implicit?* The first often produces a reformulation and the second a one-line configuration change.

6.13.4. Weights: `1/64 = 0.015625` for the model, `1/144 = 0.006944` for the sensor. Estimate `= (0.015625*636 + 0.006944*630) / 0.022569 = 634.2`. Combined spread `= square root of (1/0.022569) = 6.66`. Gain `= 0.006944/0.022569 = 0.31`. What changed: the sensor is now the *less* trusted source, so the estimate sits close to the model prediction and the twin has become model-dominated. This makes Sec. 6.6.4's first question concrete – a gain that has fallen to 0.31 is telling you something about the sensor, and if it got there gradually, that is a sensor degrading in plain sight.

6.13.5. Runs: `0.05 * 0.95 / (0.005)^2 = 0.0475 / 0.000025 = 1,900`. Steps: `1,900 * 2,000 = 3.8` million. At 50 microseconds: 190 seconds on one core; about 3 seconds on 64 cores, since Monte Carlo is embarrassingly parallel (Sec. 6.7.2). The interesting part is the ratio: this is a case where parallelism alone moves a service from "batch job" to "inside a request", with no modelling change at all.

6.13.6. *Hint:* scale it, or tighten the resource. Sixty plants and four coils, or a rule that no two pumps may draw from the twelve-volt supply simultaneously, plus a requirement that no plant may go more than 24 hours without water. The question that forces it is a *guarantee* – "prove no plant can be starved" – rather than a measurement. Sec. 6.4.3's payoff is answering by analysis rather than by running and hoping.

6.13.7. Open. Strong answers involve control loops where the fast dynamics are the thing being controlled (a switching power converter, an engine control loop) so summarising them away discards the phenomenon of interest. The fallback is then the numerical one: an implicit solver, and budgeting for the cost – plus the honest recognition that this is where Sec. 6.5's surrogates start to look attractive.

6.13.8. *Partial.* (a) The state estimator, Sec. 6.6 – the condition is inferred, not measured, which is the estimator's whole purpose. A Kalman filter may be wrong here by Sec. 6.6.3's stated criterion: crack growth and the vibration-to-condition map are not linear, and the criterion is linearity, not difficulty. (b) `2,000 particles * 30 ms = 60 seconds` per estimation step, against a step every 10 minutes. It fits, with a factor of ten to spare – decisive rather than marginal, as the hint said, and worth noticing that "it fits" was not obvious before the multiplication. The Sec. 6.5 device is a surrogate, which would move the factor from ten to thousands. (c) `0.5*0.5/(0.02)^2 = 625` runs, using the worst case p = 0.5 since the answer is unknown; five simulated years per run at whatever the fatigue model costs. The one thing that makes it feasible is a surrogate – the same answer as (b), which is the point. (d) is yours.

6.13.9. *Hint:* the estimator predicts without correcting, so its spread grows every step – which is the mechanism Chapter 2's age-of-twin needed and never had. Chapter 3's guard should refuse to command once the spread exceeds a stated bound, and the bound should be derived from the decision (Chapter 5's Sec. 5.7.4 reasoning). The operator sees the growing spread before the refusal, not at it. Testing: replay (Chapter 5, Sec. 5.5.2) with a gap cut out of the input stream, asserting that the spread grows monotonically and that the guard refuses at the right point.

6.13.10. *Hint:* no solution, but a test. Count how many rows of your memo chose the cheaper option. If it is none, re-read Sec. 6.9's closing paragraph. And check that your answer to item 4 states what you would do if the number is unaffordable – "run fewer" is a legitimate answer only if you also say what precision it buys.

6.14 Where to go next

In this book. Chapter 7 is the chapter this one has been deferring to throughout: whether any of this deserves to be believed – calibration, verification, validation, and the credibility argument Chapter 2 said arrives with the command path. Chapter 8 closes Part II by taking the data-driven family seriously as a risk. Then Part III builds: Chapter 9 the connector, Chapter 10 the store, Chapter 11 the services whose real-time factors Chapter 5 derived and whose optimisation runs Sec. 6.7 sized, Chapter 12 the platforms and tools this chapter named without recommending, Chapter 13 the standards including FMI.

In the literature.

- *The closest single reference to this chapter:* [6] covers physics-based models from rates to space, discrete formalisms, hybrid execution and FMI-based co-simulation, worked on an incubator; [5] is the containing book. [4] is the deeper treatment, including a Modelica chapter and a discrete-event-formalism chapter.
- *State estimation:* [17] is a tutorial derivation of the Kalman filter aimed specifically at twins; [18] is a validated production deployment on a full-scale floating wind turbine; [14] compares Kalman and particle filters and explains why particle filters pair with surrogates; [19] surveys both in predictive maintenance and states the linearity criterion; [20] compares them on battery life; [31] treats estimation and online calibration together.

- *Uncertainty and Monte Carlo:* [15] is the dedicated treatment for twins; [16] on scaling principled updating to a fleet; [22] for a clinical twin using posterior trajectories for risk-aware decisions; [21] and [7] for structural applications; [23] for what remains unsolved.
- *Surrogates and reduced-order models:* [13], [3], and [32] for the probabilistic-machine-learning side.
- *Discrete formalisms in twins:* [8] for discrete-event and Petri-net methods in manufacturing twins; [9] for formalism selection as an explicit design stage; [10] for the white-to-black scale of modelling techniques; [11] for behaviour patterns; [12] for data assimilation extended to discrete-event and agent-based models.
- *Co-simulation and FMI:* [25] for the model-exchange versus co-simulation split in the clearest terms; [24] for a concrete master implementation; [28] for what practitioners find hard; [26] and [27] for open frameworks built on FMI; [33] for a worked multi-domain case; [2] for coupling an equation-based tool to a legacy one; [29] for runtime monitoring of master algorithms.
- *Consulted, not drawn on above:* [1] on solver choice and control representation, [34] as a tutorial companion, [35] for time discrepancy (Chapter 5), [36] on models that learn their own structure, [37] on updating a finite-element twin from measurements, [38] on combining information sources across a wind farm, [39] on uncertainty in evolving twins, and [40] on testing twins systematically.

In the demonstrator. Take the loop you wrote after Chapter 5 and add the six lines of Sec. 6.6.2 to it: predict with the model, then combine with the next reading using the weighted average. Plot the raw readings, the model prediction and the combined estimate on the same axes. The estimate should sit between the other two and be visibly smoother than the readings. That picture is what every state estimator in this chapter is doing, and you will have built one without any matrices.

6.15 References

[1] 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.

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

[4] 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.

[5] J. Fitzgerald, C. Gomes, and P. G. Larsen, Eds., *The Engineering of Digital Twins*. Cham: Springer International Publishing, 2024. doi: 10.1007/978-3-031-66719-0.

[6] G. Abbiati, C. Gomes, M. Sandberg, Z. Kazemi, S. T. Hansen, and P. G. Larsen, "Modelling for Digital Twins," in *The Engineering of Digital Twins*, J. Fitzgerald, C. Gomes, and P. G. Larsen, Eds., Cham: Springer International Publishing, 2024, pp. 89–127. doi: 10.1007/978-3-031-66719-0_5.

[7] 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.

[8] J. Leng, D. Wang, W. Shen, X. Li, Q. Liu, and X. Chen, "Digital twins-based smart manufacturing system design in Industry 4.0: A review," *Journal of Manufacturing Systems*, vol. 60, pp. 119–137, Jul. 2021, doi: 10.1016/j.jmsy.2021.05.011.

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

[10] N. Anwer, R. Stark, F. Tao, and J. Erkoyuncu, "Developing and leveraging digital twins in engineering design," *CIRP Annals*, vol. 2025, Jun. 2025, doi: 10.1016/j.cirp.2025.05.002.

[11] 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.

[12] 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.

[13] 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.

[14] A. Thelen *et al.*, "A comprehensive review of digital twin — part 1: Modeling and twinning enabling technologies," *Structural and Multidisciplinary Optimization*, vol. 65, no. 12, p. 354, Nov. 2022, doi: 10.1007/s00158-022-03425-4.

[15] 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.

[16] 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.

[17] H. Feng, C. Gomes, and P. G. Larsen, "Model-Based Monitoring and State Estimation for Digital Twins: The Kalman Filter." arXiv, Apr. 2023. doi: 10.48550/arXiv.2305.00252.

[18] 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.

[19] R. van Dinter, B. Tekinerdogan, and C. Catal, "Predictive maintenance using digital twins: A systematic literature review," *Information and Software Technology*, vol. 151, p. 107008, Nov. 2022, doi: 10.1016/j.infsof.2022.107008.

[20] 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.

[21] M. Torzoni, M. Tezzele, S. Mariani, A. Manzoni, and K. E. Willcox, "A digital twin framework for civil engineering structures," *Computer Methods in Applied Mechanics and Engineering*, vol. 418, p. 116584, Jan. 2024, doi: 10.1016/j.cma.2023.116584.

[22] 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.

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

[24] C. Friedrich, A. Lombana, J. Fasquel, C. Schlick, N. Bennani, and M. Mendil, "CoFMPy: A Python Framework for Rapid Prototyping of FMI-based Digital Twins," in *The 2nd International Conference on Engineering Digital Twins*, 2025. Accessed: Nov. 10, 2025. [Online]. Available: https://hal.science/hal-05326255/

[25] 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

[26] S. Gil, P. H. Mikkelsen, C. Gomes, and P. G. Larsen, "Survey on open-source digital twin frameworks–A case study approach," *Software: Practice and Experience*, vol. 54, no. 6, pp. 929–960, Jun. 2024, doi: 10.1002/spe.3305.

[27] S. Infante *et al.*, "Integrating FMI and ML/AI models on the open-source digital twin framework OpenTwins," *Software Practice and Experience*, Mar. 2024, doi: 10.1002/spe.3322.

[28] G. Schweiger *et al.*, "An empirical survey on co-simulation: Promising standards, challenges and research needs," *Simulation Modelling Practice and Theory*, vol. 95, pp. 148–163, Sep. 2019, doi: 10.1016/j.simpat.2019.05.001.

[29] E. Kamburjan, A. Pferscher, R. Schlatte, R. Sieve, S. L. T. Tarifa, and E. B. Johnsen, "Semantic Reflection and Digital Twins: A Comprehensive Overview," in *The Combined Power of Research, Education, and Dissemination*, vol. 15240, M. Hinchey and B. Steffen, Eds., Cham: Springer Nature Switzerland, 2025, pp. 129–145. doi: 10.1007/978-3-031-73887-6_11.

[30] 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.

[31] N. Zhang, R. Bahsoon, N. Tziritas, and G. Theodoropoulos, "Knowledge Equivalence in Digital Twins of Intelligent Systems," *ACM Transactions on Modeling and Computer Simulation*, vol. 34, no. 1, pp. 1–37, Jan. 2024, doi: 10.1145/3635306.

[32] 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.

[33] Z. Liu, Y. Chu, G. Li, H. P. Hildre, and H. Zhang, "Shipboard crane digital twin: An empirical study on R/V Gunnerus," *Ocean Engineering*, vol. 302, p. 117675, Jun. 2024, doi: 10.1016/j.oceaneng.2024.117675.

[34] C. Gomes *et al.*, "Digital Twin Tutorial: The Incubator Case Study," in *Engineering Trustworthy Software Systems: 6th International School, SETSS 2024, Chongqing, China, April 14–21, 2024, Tutorial Lectures*, J. P. Bowen, C. Gomes, and Z. Liu, Eds., Singapore: Springer Nature, 2025, pp. 68–101. doi: 10.1007/978-981-96-4656-2_3.

[35] 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.

[36] F. Emmert-Streib, H. Cherifi, K. Kaski, S. Kauffman, and O. Yli-Harja, "Complexity data science: A spin-off from digital twins," *PNAS Nexus*, vol. 3, no. 11, p. pgae456, Nov. 2024, doi: 10.1093/pnasnexus/pgae456.

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

[38] L. A. Bull *et al.*, "Data-Centric Monitoring of Wind Farms: Combining Sources of Information," in *Data Driven Methods for Civil Structural Health Monitoring and Resilience*, CRC Press, 2023.

[39] Q. Xu, T. Yue, S. Ali, and M. Arratibel, "Pretrain, Prompt, and Transfer: Evolving Digital Twins for Time-to-Event Analysis in Cyber-physical Systems." arXiv, Apr. 2024. doi: 10.48550/arXiv.2310.00032.

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