CS445: Basic Simulation Modeling - Part 2

Travis Desell

Averill M. Law, Simulation Modeling & Analysis, Chapter 1

Exponential Distribution Functions

M/M/1 queue

A simulation where there is a queue, and arrivals and departures generated using exponential random variables is extremely common in simulation, and is commonly called an *M/M/1* queue.

Exponential Distributions

For the time being (and for your next assignment) we'll use an exponential distribution to generate the inter arrival and service times:

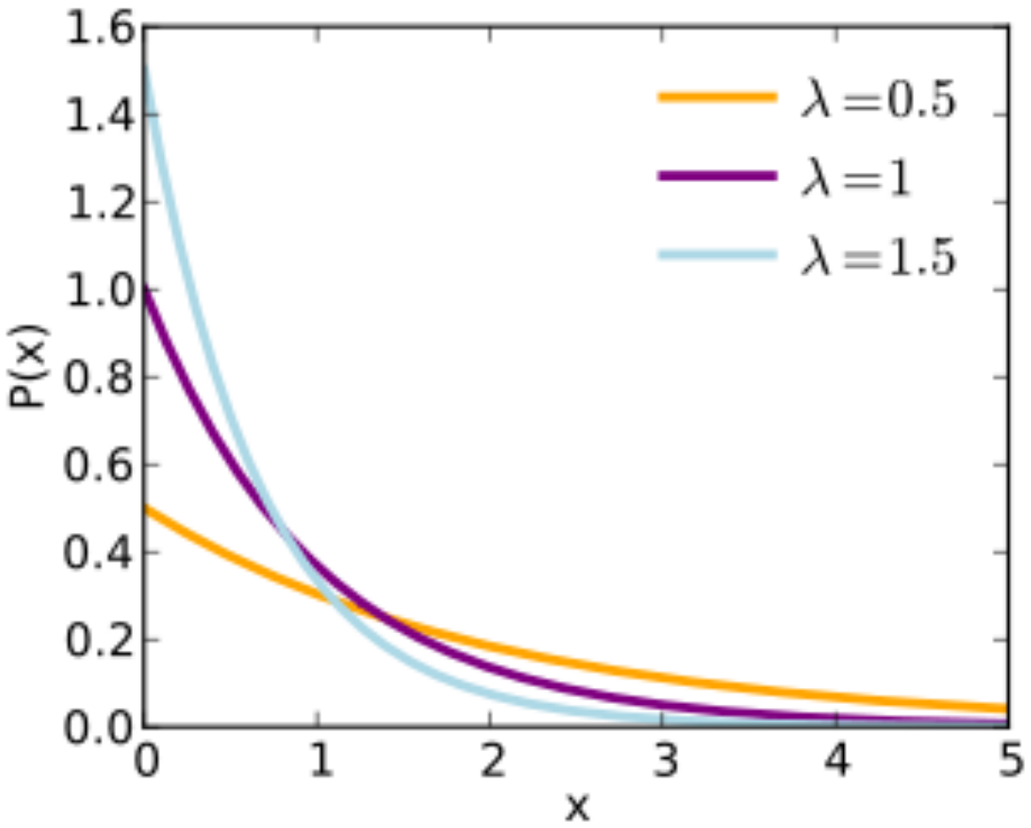
$$f(x) = \frac{1}{\beta} e^{\frac{-x}{\beta}}$$

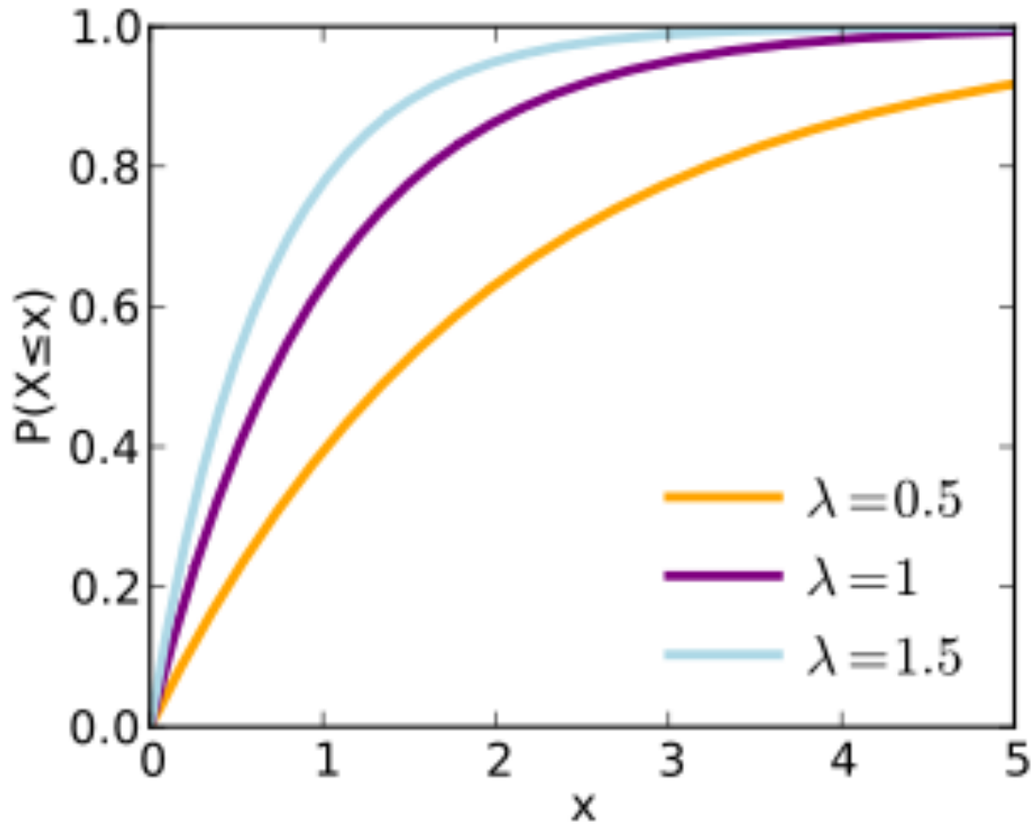
Where x >= 0.

It is fairly easy to generate random variables with an exponential distribution (especially using boost), and it can be realistic for inter arrival times (although not so much for service times).

The choice of beta (the B) is made arbitrarily (usually to fit historical data).

Exponential Distribution Function





The above are examples of the exponential distribution function. The left is the probability distribution function, and the right is the cumulative distribution function. More on these in chapter 6, but essentially they describe the chance of a random number being generated. The Gamma is 1/Beta.

See: http://en.wikipedia.org/wiki/Exponential_distribution

Generating Exponential Random Variables

If x is a variable generated from a uniform random distribution (i.e., from your standard random number generator, the following function will transform them into an exponential random distribution:

$$f(x) = \frac{1}{\beta} e^{\frac{-x}{\beta}}$$

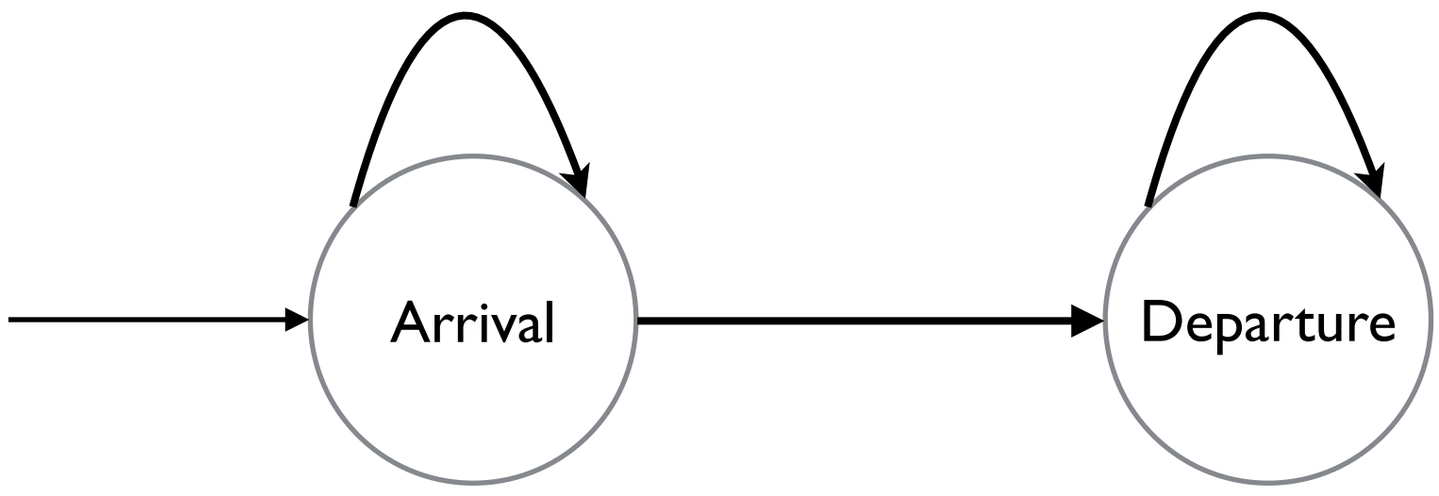
Where x >= 0. The mean of the variables generated will be Beta. There will be more on generating random variables of particular distributions in chapter 8.

Designing Simulations

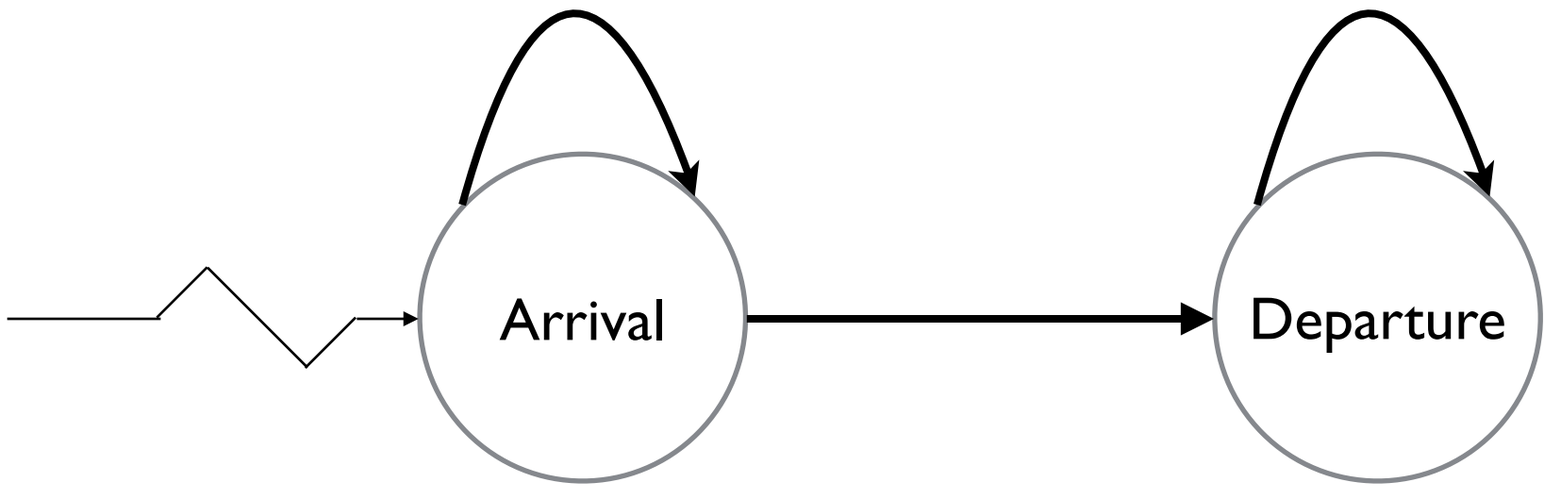
Designing Simulations

Schruben (1983b) presented an *event-graph* method that was later refined by Sargent (1988), and Som and Sargent (1989).

This method uses nodes and directed edges (like a graph) to depict how events are scheduled from other events. So in the case of our single server simulation:



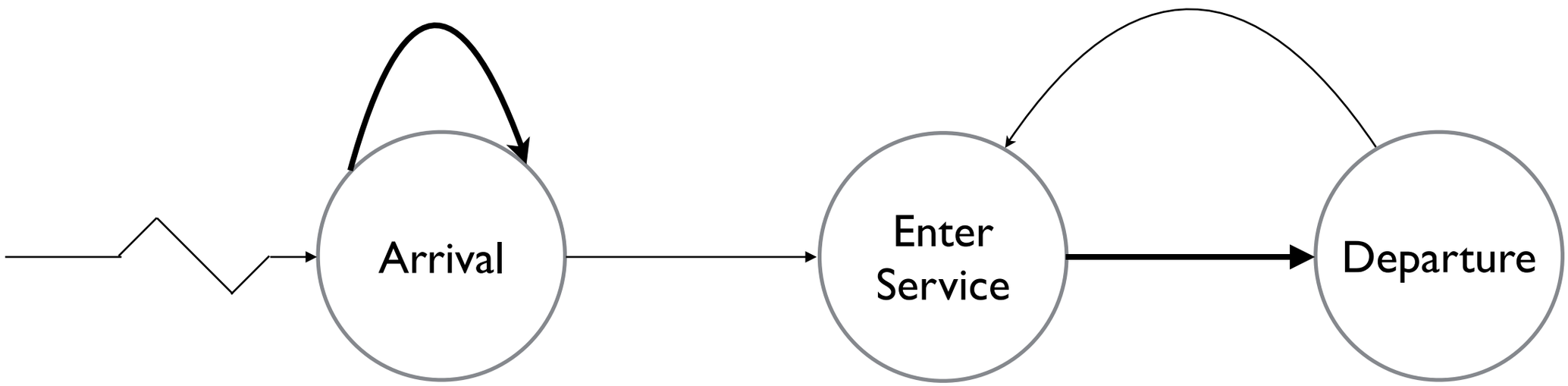
Designing Simulations



The small jagged line on the left indicates the beginning of the simulation. The thick lines indicate that the event *may* generate an event of the target type in a *nonzero* amount of time.

An arrival event generates another arrival event, as well as a departure event, and a departure event may generate another departure event (if there are people waiting in the queue).

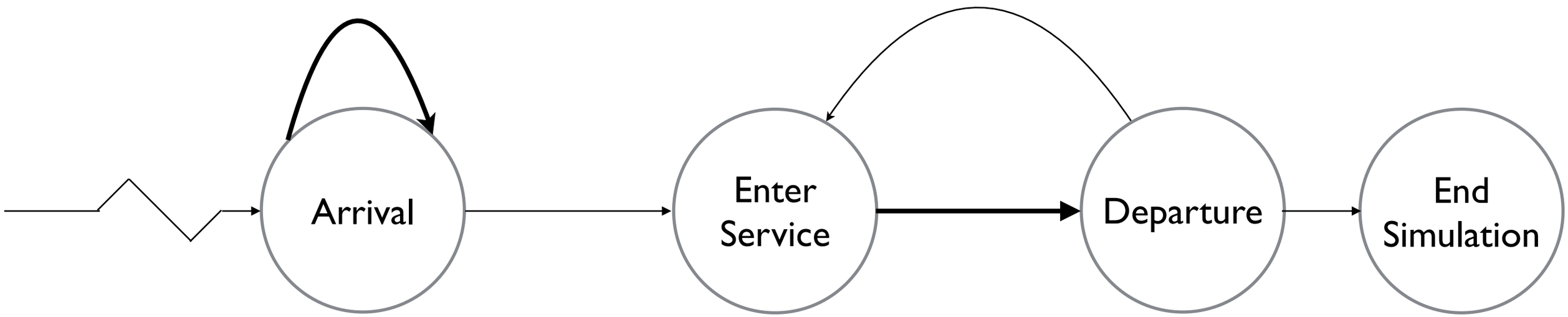
Designing Simulations



It could also be possible to specify our simulation as follows, where we have separate events for entering service and departure.

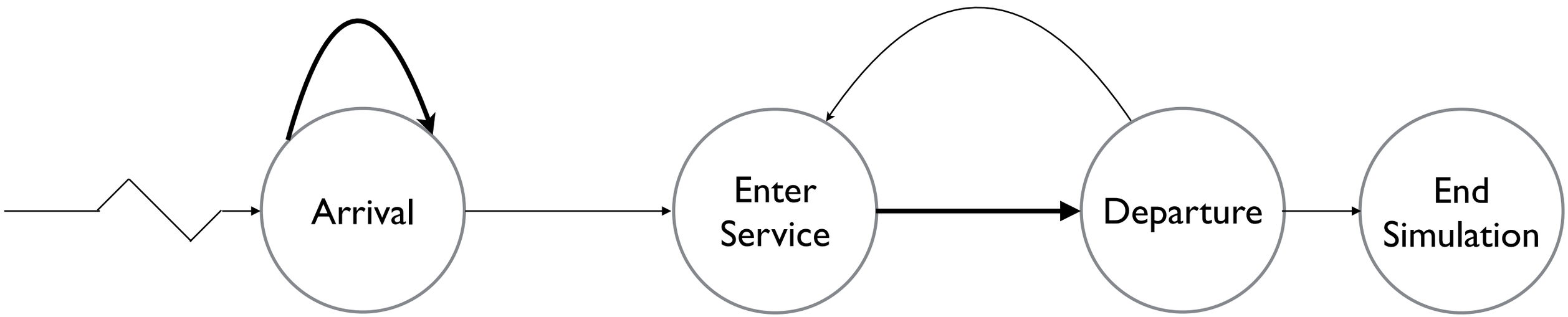
In this case however, the thin arrows indicate that the source of the arrow can schedule the target of the arrow to occur immediately (e.g., in the first arrival or when a departure occurs with more customers in line).

Designing Simulations



Likewise, if we are running the simulation for a fixed number of departures, the departure event can immediately schedule a end simulation event.

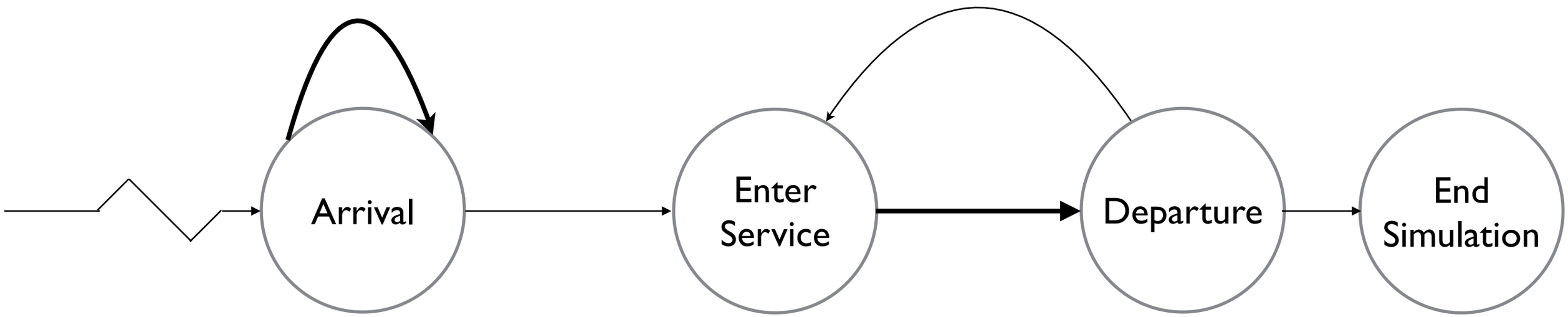
Designing Simulations



This increases the number of events by 2, and the complexity of our simulation accordingly. However, there are rules by which we can reduce the number of nodes in one of these event-graphs to simplify our implementation.

When designing a simulation, it may not be a bad idea to first generate an event-graph, and then use these rules to simplify it before implementing the simulation.

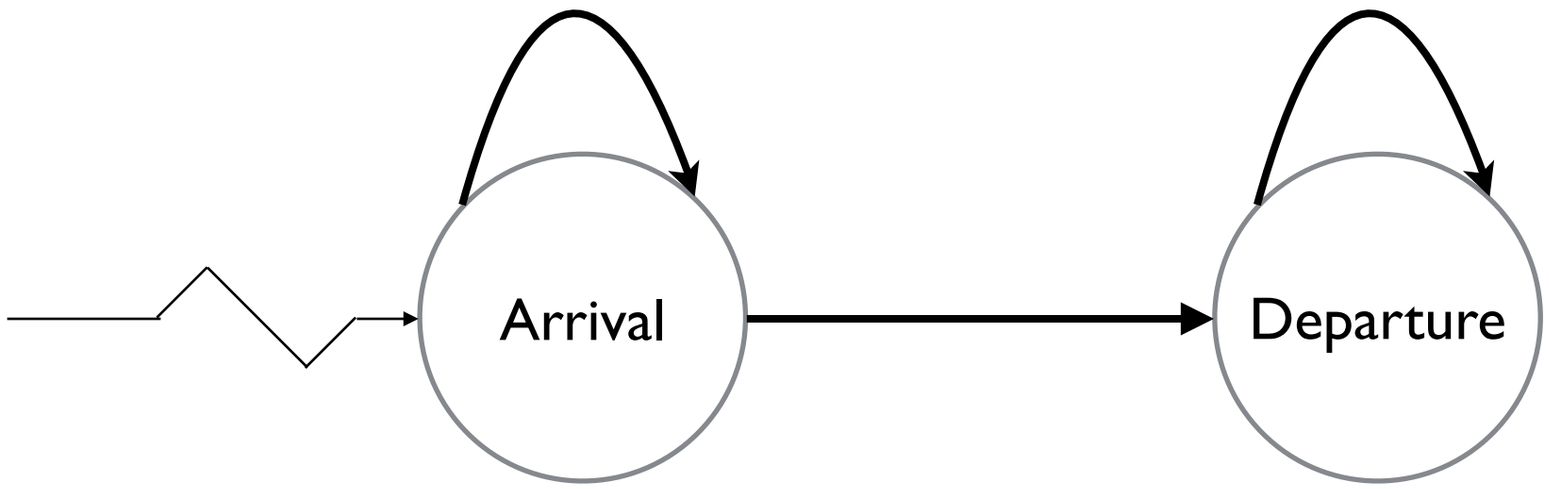
Designing Simulations



One rule for simplification is that if a node only has incoming arcs that are all thin and smooth (in the case of enter service and end simulation), it can be removed from the model and it's functionality added to the event(s) which schedule it in zero time.

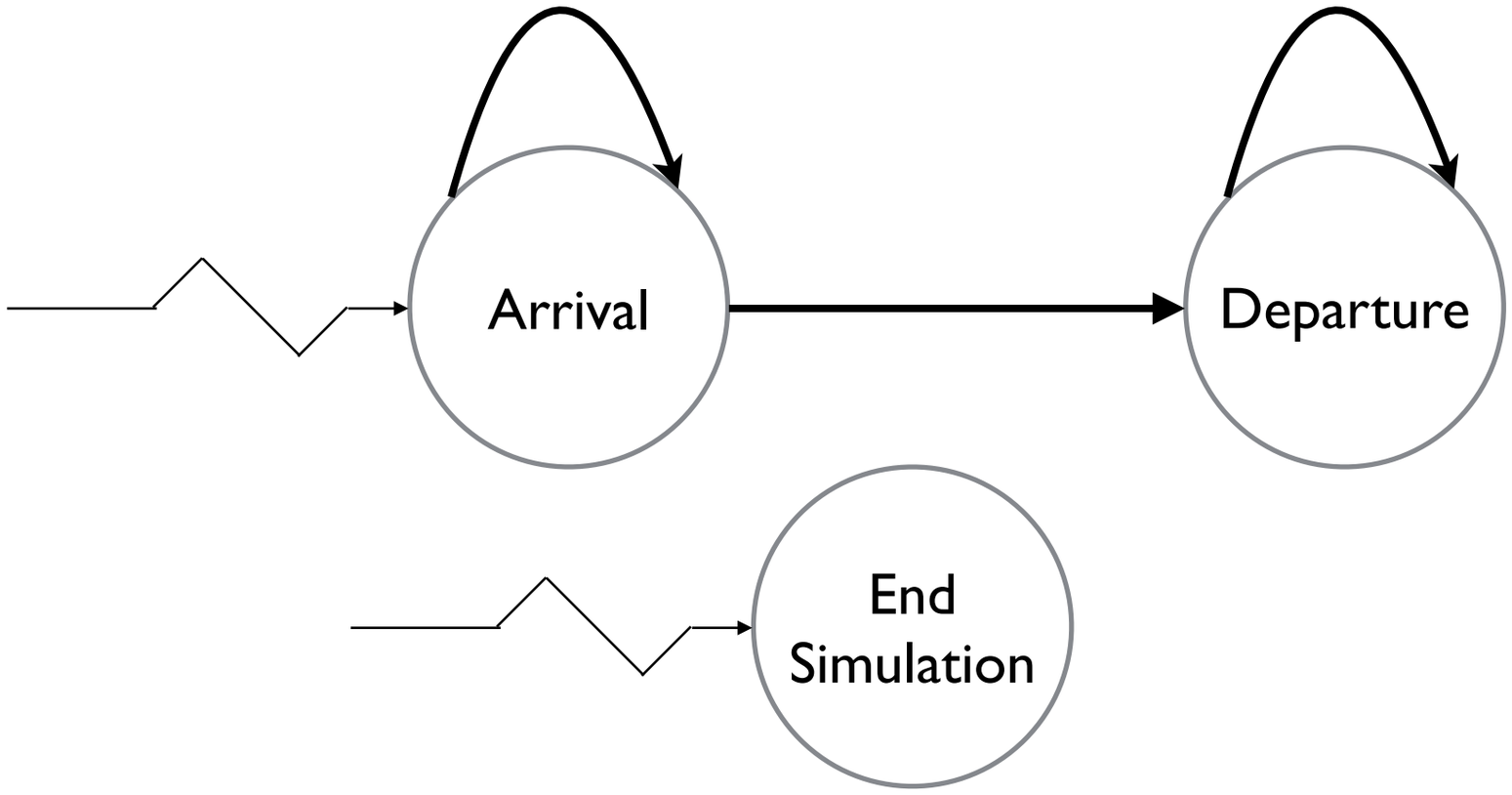
This makes sense, as when you think about our min-heap event queue, it doesn't make sense to schedule an event to happen now (which would involve inserting it and immediately removing it from the heap), when we could instead just execute the extra functionality.

Designing Simulations



Applying this rule brings our event graph to its original form.

Designing Simulations



Now say we want our simulation to end at a particular time, as opposed to in response to a departure event.

In this case we need to add another node, as in this case the end of the simulation is another event we generate at the beginning of the simulation.

Designing Simulations

This just scratches the surface of designing events via event-graphs. Some modern software even allows the development of simulations visually through interfaces which are similar to these event-graphs (see SIGMA).

Event-graphs can be used for more than just simulation design as well. It is also possible to analyze them for certain kinds of error detection, which is useful in highly complex simulation models.

Chapter 1.4.7 goes into further details and provides more references if you're interested.

Simulation of an Inventory System

Inventory System Simulation

A company that sells a single item (hey it could happen!) wants to decide how many items it should have in its inventory for the next *n* months.

The times between demands are IID (independent and identically distributed) exponential random variables with a mean of 0.1 month.

The size of the demands *D*, are IID random variables (also independent of when the demand occurs) with:

D = 1, 1/6th the time

D = 2, 1/3rd the time

D = 3, 1/3rd the time

D = 4, 1/6th the time

Inventory System Simulation

At the beginning of each month, the company reviews the inventory level and decides how many items to order from its supplier.

If the company orders Z items, it incurs a cost of $K + iZ$, where K = \$32 is the *setup cost*, and i = \$3 is the *incremental cost* per item ordered. If $Z = 0$, no cost is incurred.

When an order is placed, the time required for it to arrive (*delivery lag* or *lead time*) is a random variable (uniformly distributed) between 0.5 and 1 month.

Inventory System Simulation

The company uses a stationary (as in not changing, opposed to paper) policy (s, S) to decide how much to order:

Z = S - I, if I < s

Z = 0, otherwise

Essentially if the inventory is below some set quantity s, the company will order enough to bring it back to some maximum level, *S*. In this case, *I*, is the inventory at the beginning of the month (which is when the company places orders).

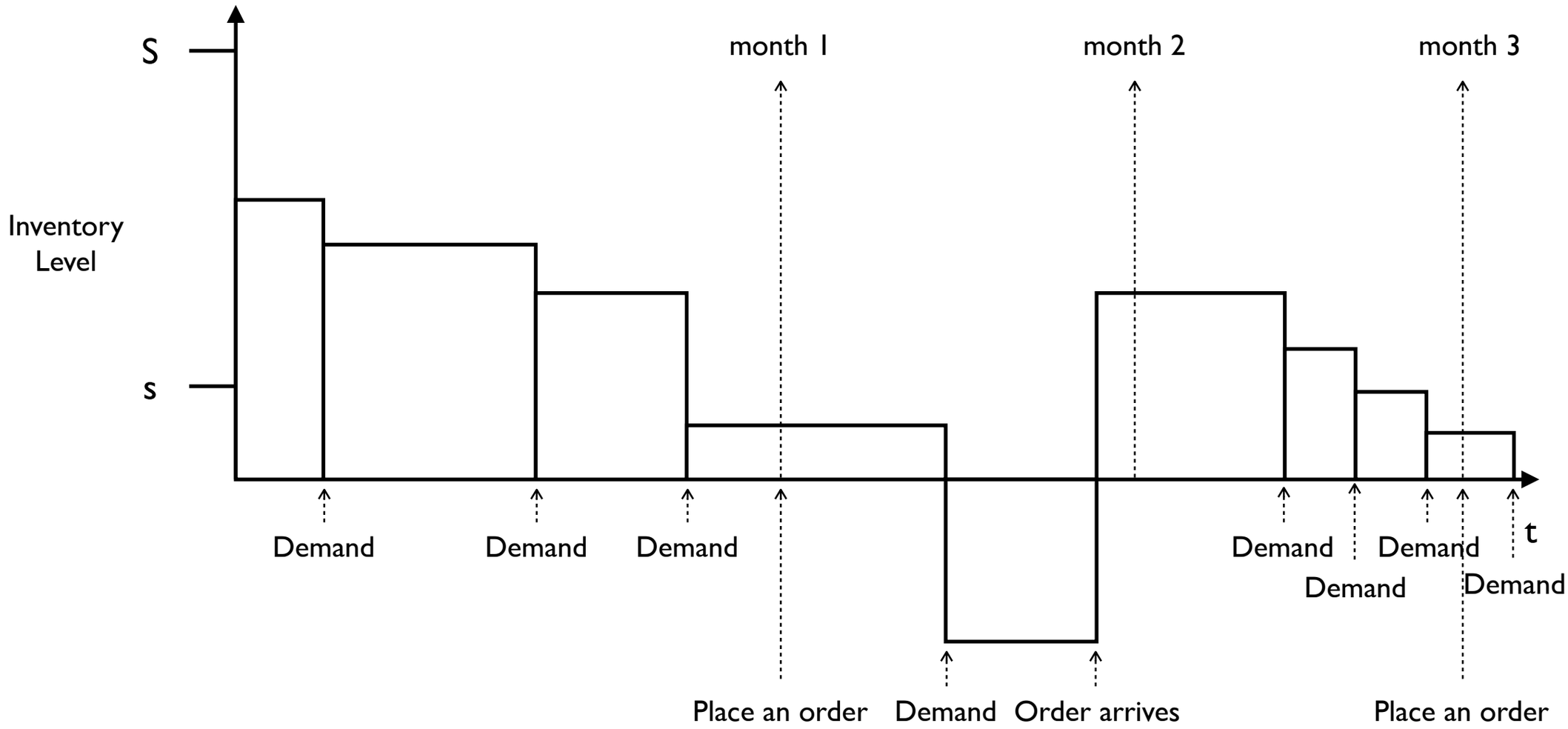
Inventory System Simulation

When a demand (a purchase) occurs, it is satisfied immediately if possible, if the inventory level is at least as much as the demand.

If the demand exceeds the inventory level, the excess of demand over supply is backlogged and satisfied for future deliveries. In this case, the new inventory level is equal to the old inventory level minus the demand size, resulting in a negative inventory level.

When an order arrives, it is first used to eliminate as much of the backlog as possible, and then the remainder is added to the inventory.

Inventory System Simulation



(Pardon the X and Y scales). The inventory level decreases when a demand is made. Orders are placed at the beginning of each month, and arrive sometime later. The inventory level can fall to negative if a demand comes in that is more than the inventory.

Inventory System Simulation

Most real inventory systems also have two additional costs: *holding* and *shortage* costs.

The company incurs a holding cost of h = \$1 per item per month held in (positive) inventory. This accounts for warehouse rental, taxes, maintenance, insurance, etc.

The company also incurs a shortage cost of of pi = \$5 per item per month in backlog — this accounts for the cost of extra record keeping when a backlog exists, as well as the loss of a customers' goodwill.

Note that these can both be calculated as continuous time statistics, like how we calculated time average number of customers in the queue with the other simulation.

Inventory System Simulation

Given:

$$\overline{I}^{+} = \frac{\int_0^n I^{+}(t)dt}{n} \qquad \overline{I}^{-} = \frac{\int_0^n I^{-}(t)dt}{n}$$

Where I+ is the positive inventory between times 0 and n, and I- is the negative inventory between times 0 and n.

The shortage cost for the simulation will be pi * I-, and the holding cost for the simulation will be h * I+.

Inventory System Simulation

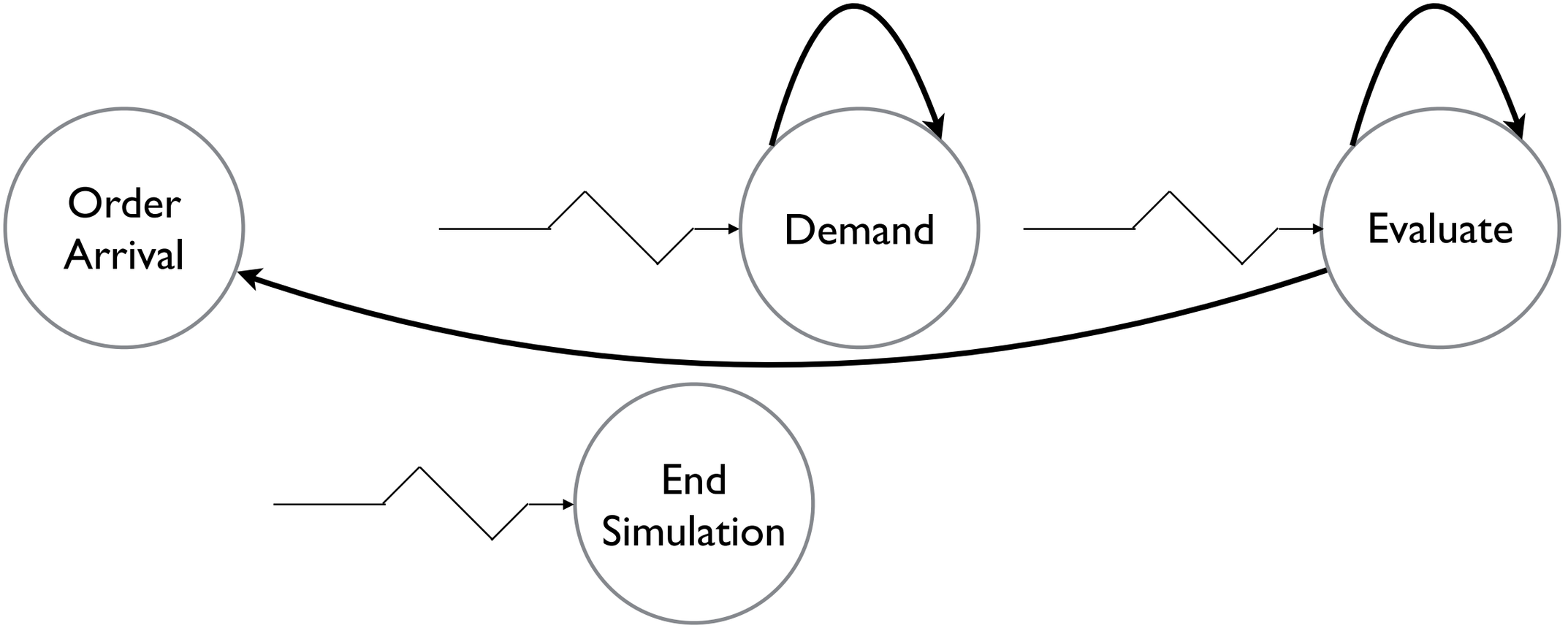
Assume the initial inventory level is I(0) = 60 and there are no outstanding orders.

We simulate the inventory system for n = 120 months and use the average total cost per month (which is the sum of the average ordering cost per month, the average holding cost per month, and the average shortage cost per month) to compare inventory policies, eg:

s =	20	20	20	20	40	40	40	60	60
S =	40	60	80	100	60	80	100	80	100

Note that this is a "brute force" search and not necessarily ideal for determining the optimal inventory policy.

Inventory System Simulation



Using the same event-graph description, we can create a design for the inventory simulation, as above.

Parallel/Distributed Simulation

Parallel/Distributed Simulation

Discrete event simulations basically operate in the same way. Events are generated, inserted into some data structure, and processed one after the other following the time of the simulation.

Given the fact we may wish to run very long (and complicated) simulations, we may need to find ways to speed up the time to run our simulations long enough in a short enough amount of time (we don't want to wait a year for simulation results).

Apart from precise code optimizations, the one thing that can aid with this is parallelization.

Another problem may arise if the simulation is so large it cannot fit in a single machines memory. In this case, it will need to be spread across multiple machines to even be able to run.

Parallel/Distributed Simulation

The simulation must then be split up into difference processes, each with their own event queue dealing with particular elements of the simulation.

The issue arises here that the events must still be processed in the order of their simulation time. So we can have the following situation:

1. process 1 processes an event at time 100, which generates an event in its queue which happens to be at time 200

2. at the same time, process 2 processes an event at time 90, which generates an event which would be processed by an element on process 1 at time 150.

3. process 1 processes its next event at time 200, because it hasn't received the event from process 2.

4. process 2 receives the event at time 150, and the distributed simulation is 'out of whack'.

Conservative Synchronization

One traditional approach to distributed simulation is *conservative synchronization.*

Suppose a process is at time 25, and its next event is scheduled be processed at time 30. This process won't proceed to time 30 until it receives a guarantee (probably from all the other processes, or from some centralized manager) that it will not receive an event with time < 30.

However, this has issues in that:

1. This type of system cannot fully exploit the parallelism available. For example, if event A can effect event B in any way, then A and B must be executed sequentially. If in the actual execution of the model, A rarely effects B, then they're still being processed sequentially most of the time.

2. They are not particularly robust, a small change in the model can result in a serious degradation of performance.

3. Calculating and performing the message sending to enforce the time guarantees can be quite expensive and limit performance and scalability.

Optimistic Synchronization

Another approach to distributed simulation is *optimistic synchronization*.

In this case, violations of causality (i.e., event B could be processed before event A even if it is scheduled later in time) are allowed to occur, but the synchronization mechanism detects this and recovers from these faults.

This allows each process to simulate their own elements within the simulation at their own speeds, without awaiting on guarantees or messages from other processors.

Optimistic Simulation

The best known optimistic approach is the *time-warp* mechanism.

When a process receives a message that should have been received in its past (which could have invalidated or changed what happened in future events that have already been processed), the simulation will *roll back* to the previous time, reverting its simulation clock and undoing any events that occurred in the meantime.

This is slightly problematic, as the events rolled back may have sent messages to other processes, which also need to be rolled back via an *anti-message*, which generate secondary rollbacks at their destination processes, which may in turn cause additional rollbacks at other processes and so forth.

Optimistic Simulation

Optimistic approaches exploit parallelism better (especially when there are few rollbacks), allowing greater scalability, however they do have disadvantages:

1. They incur the overhead computations associated with executing rollbacks (which can be quite expensive).
2. They require more computer memory as they need to save previous states so they can perform rollbacks to those states.

Advantages, Disadvantages and Pitfalls of Simulation

Advantages

- Most complex, real-world systems with stochastic elements cannot be accurately described by mathematical models that can be evaluated *analytically* — meaning simulation is the only type of investigation possible.
- Simulation allows one to estimate the performance of an existing system under some project set of operating conditions.
- Alternative proposed system designs (or alternative operating policies for a single system) can be compared via simulation to see which best meets a specified requirement or requirements.
- In a simulation we can maintain much better control over experimental conditions that would generally be possible when experimenting with the system itself (more on that in Chapter 11).
- Simulation allows us to study a system with a long time frame — e.g., an economic system — in compressed time, or alternatively to study the detailed workings of a system in expanded time.

Disadvantages

- Each run of a *stochastic* simulation model produces only e*stimate*s of a model's true characteristics for a particular set of input parameters. Because of this, several independent runs of a model will probably be required for each set of input parameters to be studied (more in Chapter 9).
- This stochastic nature creates challenges when optimizing simulation parameters for given output criteria (however, analytic models do not suffer from this), at least for traditional optimization methods.
- Simulation models are often expensive and time-consuming to develop and debug (as is all software!).
- The large volume of numbers produced by a simulation study or the persuasive impact of a realistic animation (see Sec 3.4.3) often creates a tendency to place greater confidence in a study's results than is justified. If the simulation is not a "valid" representation of the system under study, any results it produces are probably not useful (no matter how fancy they look).

Common Pitfalls 1

- Failure to have a well-defined set of objectives at the beginning of the simulation study
- Failure to have the entire project team involved at the beginning of the study
- Inappropriate level of model detail (either to much or too little)
- Failure to communicate with management throughout the course of the simulation study
- Misunderstanding of simulation by management
- Treating a simulation study as if it were primarily an exercise in computer programming
- Failure to have people with knowledge of simulation methodology (see Chapters 5, 6, 9, etc) and statistics on the modeling team
- Failure to collect good system data

Common Pitfalls 2

- Inappropriate simulation software
- Obliviously using simulation-software products whose complex macro statements may not be well documented and may not implement the desired logic
- Belief that easy-to-use simulation packages, which require little or no programming, require a significantly lower level of technical competence
- Misuse of animation
- Failure to account correctly for sources of randomness in the actual system
- Using arbitrary distributions (e.g., normal, uniform, or triangular) as input to the simulation
- Analyzing the output data from one simulation run (replication) using formulas that assume independence

Common Pitfalls 3

- Making a single replication of a particular system design and treating the output statistics as "true answers"
- Failure to have a warmup period, if the steady-state behavior of a system is of interest
- Comparing alternative system designs on the basis of one replication for each design
- Using the wrong performance measures