CS445: Basic Simulation Modeling

Travis Desell

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

What is Simulation?

A *simulation* uses a computer (or computers) to evaluate a model *numerically*, and data are gathered in order to *estimate* the desired true characteristics of the model.

Some Definitions

1. A *system* is a collection of entities (e.g., people or machines) that act and interact together toward the accomplishment of some logical end.
2. The *state* of a system is the collection of variables necessary to describe a system at a particular time, relative to the objectives of the study.

Discrete vs. Continuous

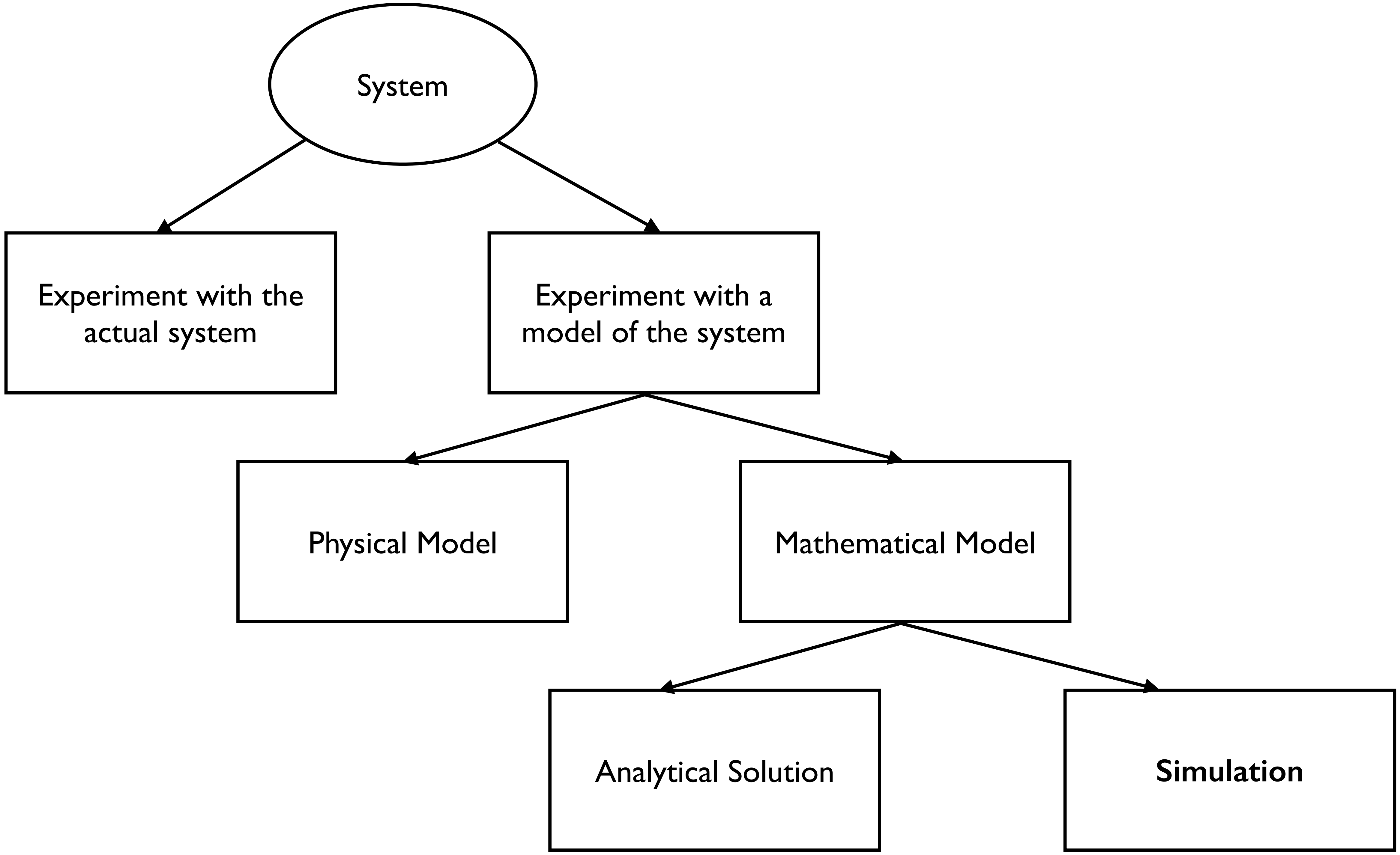
Systems can be *discrete*, where the state variables change "instantaneously" at separate points in time (think the number of customers in a bank), or *continuous*, where the state variables can change continuously in regards to time (think an airplane flying through the air, with velocity, altitude, etc variables).

Note that when it comes to a programmed simulation, nothing can ever be truly continuous or discrete.

Common Simulations

1. Designing and analyzing manufacturing systems.
2. Evaluating military weapons systems or their logistics requirements.
3. Determining hardware requirements or protocols for communication networks.
4. Determining hardware and software requirements for a computer system.
5. Designing and operating transportation systems such as airports, freeways, ports and subways.
6. Evaluating designs for service organizations such as contract centers, fast-food restaurants, hospitals and post offices.
7. Reengineering of business processes.
8. Analyzing supply chains.
9. Determining ordering policies for an inventory system.
10. Analyzing mining operations.

When to Simulate



Actual Systems vs. Models

In some cases, the actual system can be modified and examined, so this case (so long as the costs and doing so is feasible) is usually ideal, as the actual system can be measured so you know the results are valid.

However, these cases are rare, typically due to the costs involved or that it would be too disruptive to the system to actually perform the modifications. Sometimes the system might not even physically exist (or at least the parts we want to better understand). In these cases a *model* of the system needs to be build to do the analysis.

Physical vs. Mathematical Models

It is often possible to make miniature (or even to scale) physical models of a system, which is common in engineering fields. Think about putting airplane wings in a wind tunnel, etc. For some systems, there may not be accurate enough mathematical models so a physical simulation may be required.

However, most models are mathematical, as having an abstract model tends to be easier to modify and to more easily test different variations.

Analytical Solution vs. Simulation

Some mathematical models are simple enough to obtain exact answers. Suppose our *system* involves a satellite traveling through deep space (simplifying things). We can determine the distance of the satellite by the function:

$$d = rt$$

where d is the distance, r is the rate of travel (velocity), and t is the time spent traveling.

However, this is rarely the case, for example, what if the satellite is traveling within our solar system. It's velocity (and acceleration) will be altered by all other astronomical bodies due to their gravity, and all those other bodies are also moving. This makes it quite difficult to analytically solve the distance/where the satellite will be (although this is exactly what early "human" computers at NASA would do[1,2]).

1. "Human computers." http://crgis.ndc.nasa.gov/historic/Human_Computers

2. "She was a computer when computers wore skirts". http://www.nasa.gov/centers/langley/news/researchernews/rn_kjohnson.html

Simulation as a “last ditch” Effort

So historically, simulation has been used as a measure of last resort. Given the potential for more valid answers, and potentially easier development of the models. However, with advancements in computing, simulation has become more of a “go to” methodology.

Static vs. Dynamic Simulation Models

Static simulation models are representations of a system at a particular time, or representations where time plays no role (e.g., some types of Monte Carlo simulation).

Dynamic simulation models represent systems as they evolve or change over time (e.g., our previous satellite example, a conveyor system in a factory, a packet network simulation, etc.).

Deterministic vs. Stochastic Simulation Models

A *deterministic simulation model* does not contain any probabilistic (random) components. In this type of a simulation, if you start it with the same set of initial conditions and parameters, the end result will always be the same.

A *stochastic simulation model* (also called a *random simulation model* or *nondeterministic simulation model*) does contain probabilistic or random components. In these simulations, even if you start with the same set of initial conditions and parameters there is no guarantee that the end result will always be the same.

As the output of a stochastic simulation model is also random, it can only be treated as an estimate of the system; and in many cases this means that many simulations must be run to get an appropriate estimate of what can happen in the system (this also happens to be one of the major drawbacks in simulation).

Continuous vs. Discrete Simulation Models

Similar to systems, simulation models can be *continuous* or *discrete*.

It is possible to have some components in a simulation be continuous while others are discrete, so mixed simulation models are also possible.

For example with a traffic flow simulation, it is possible to model traffic as individual cars with their own characteristics and movement. It may also be possible (given the scale and amount of data) to model the cars in aggregate as flows, in a continuous manner.

In hydrodynamics simulations this is also common, it is possible to model these systems with individual particles (as in smoothed particle hydrodynamics and lattice Boltzmann simulations); or as flows (in computational fluid dynamics simulations which are based on calculating differential equations).

Discrete-Event Simulation

Discrete-Event Simulation

This course will mostly focus on discrete, dynamic, and stochastic simulation models, which are commonly called *discrete event simulations*.

Note that deterministic simulations are a special case or subset of stochastic simulations so limiting the material to stochastic simulations doesn’t functionally limit the material.

Discrete-Event Simulation

Again, *discrete event simulations* involve modeling of systems as they evolve over time, where the state variables change instantaneously at separate points in time (i.e., the system can change at only a *countable* number of points in time).

In these simulations, the *events* are defined as instantaneous occurrences that may change the state of the system (note they can also potentially create future events). Generally, these simulations generate and evaluate so many events that performing them by hand is not feasible.

Time Advancement and Event Queues

In these event based simulations, there is usually a *simulation clock* and an *event queue* to hold the events.

Events are processed in order, based on the time they are scheduled to be executed, so the simulation clock increments itself to the start time of the next event scheduled. In this case, this is called a *next-event time advance*. Discrete event simulations almost always use a *next-event time advance*.

Other simulations use a *fixed-increment time advance*. In this case, typically something happens ‘continuously’ throughout the simulation, so the state variables are updated at fixed time increments (this is common in n-body simulations, fluid dynamics, heat diffusion and geological simulations).

Example Discrete-Event Simulation

Cashier/Queue Example

Given a service facility with a single server — a one operator barbershop or store with one cashier, etc.

We want to estimate average (expected) delay in the line for arriving customers.

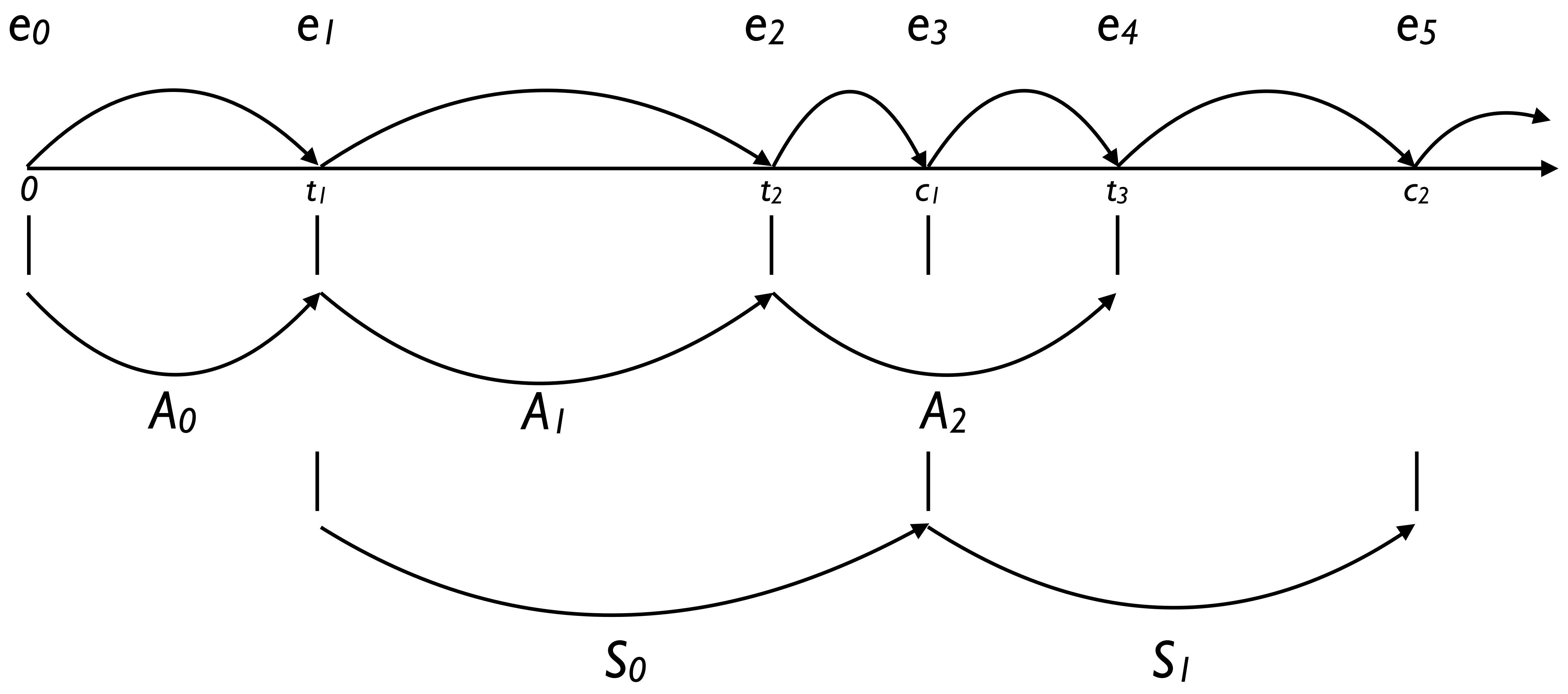
We can define the delay of a customer as the time it takes from when they arrive at the store to when they are served by the operator. A customer who is served immediately has a delay time of 0.

Cashier/Queue Example

What do we need to determine to make such a simulation?

1. A way to determine at what times the customers arrive. A customer arrival can be an event.
2. A way to determine what to do with a customer when it arrives. The arrival event can generate an event to put the customer in the line, or an event to process the customer and send them on their way.
3. A data structure to handle customers in line. Each customer can store the time they enter the line, and when a customer is done being processed an event can be generated to take the customer off of the front of the line and process them.

Cashier/Queue Example



Given:

t_i = time of arrival of the ith customer ($t_0 = 0$)

$A_i = t_i - t_{i-1}$ interarrival time between the (i - 1)st and ith arrivals

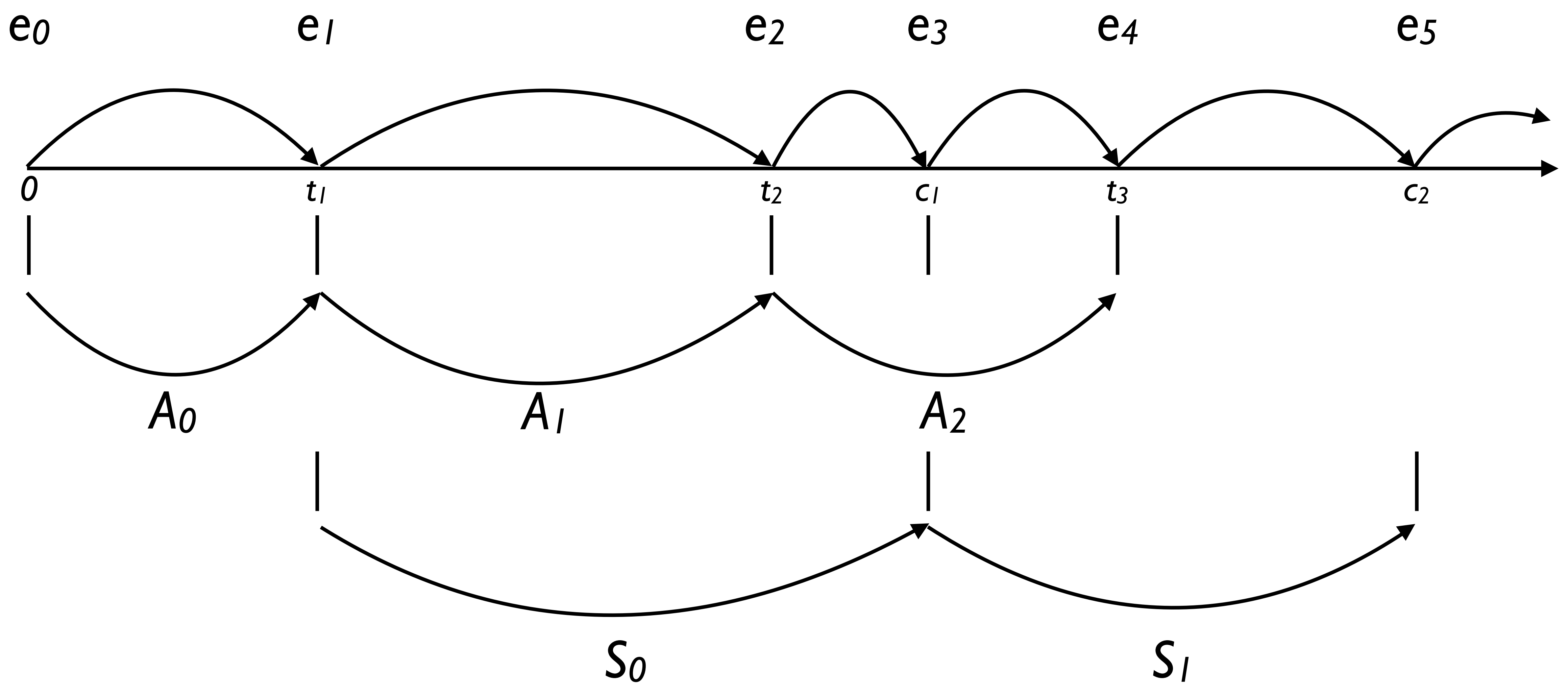
S_i = time the server spends serving the ith customer

D_i = delay in queue of the ith customer

$c_i = t_i + D_i + S_i$ = time the ith customer completes service and departs

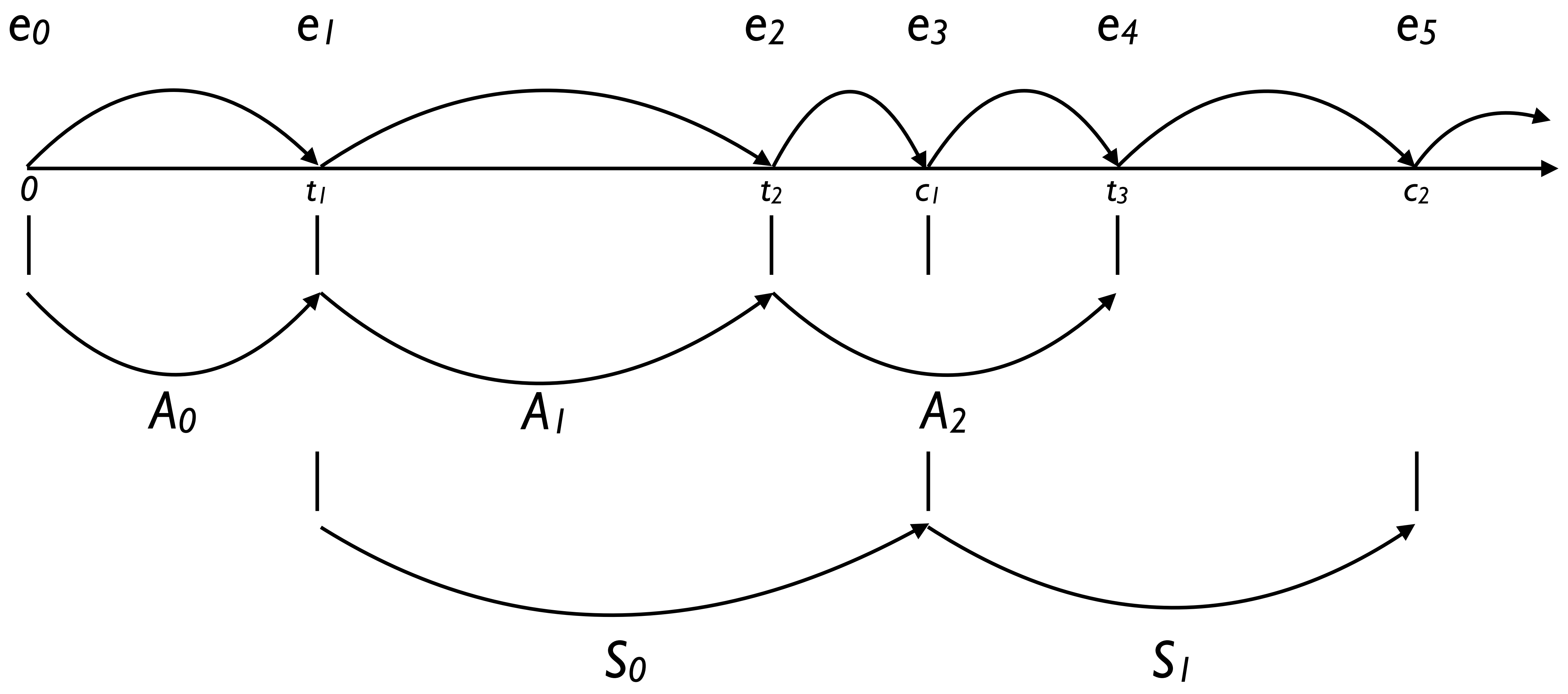
e_i = time of occurrence of ith event of any time (ith value the simulation clock takes on, excluding $e_0 = 0$)

Cashier/Queue Example



The inter arrival times ($A_0 \dots A_n$) and the service times ($S_0 \dots S_n$) are generated from cumulative distribution functions (we can call these F_A and F_S — distribution functions are described in detail in Chapters 4.2 and 6). That means these numbers take on random values generated from some kind of distribution function.

Cashier/Queue Example



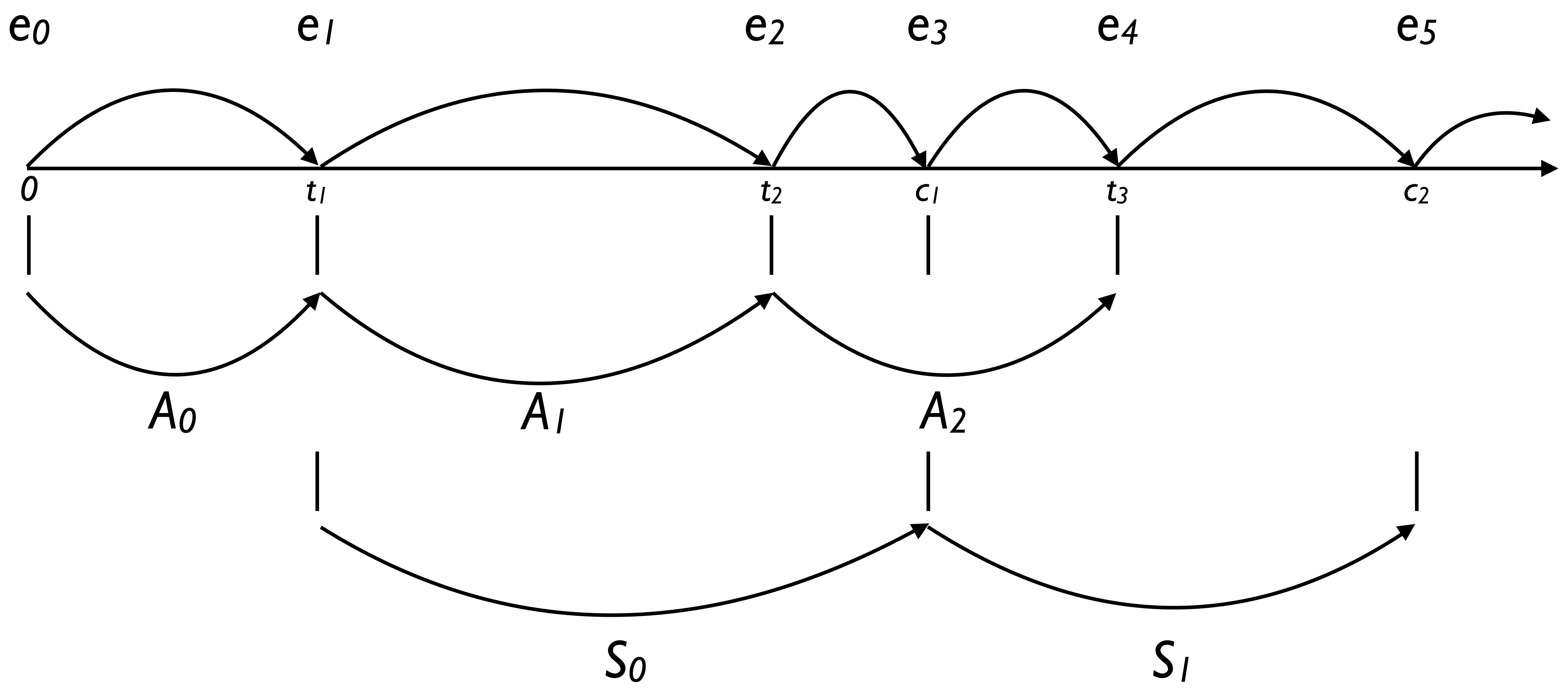
The simulation works as follows:

1. The simulation starts at time 0, with event e_0. Because there are no customers waiting in line, the arrival time for the next customer is generated as A_0, which causes event e_1 to be added to the queue with start time of $t_1 = A_0$.

2. The simulation clock then moves to time t1, and the simulation processes the next event, e_1.

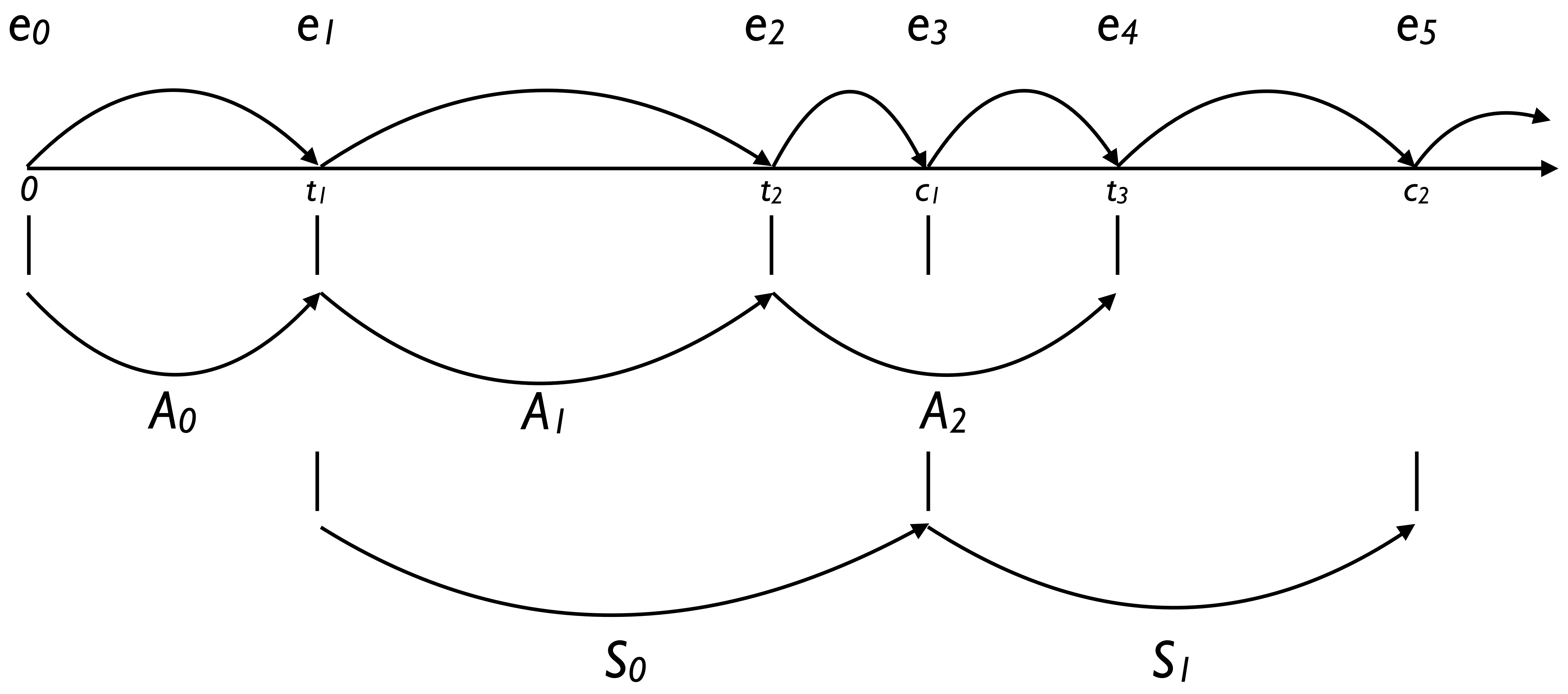
3. For event e_1 a customer arrives and there is no waiting line, it can be serviced. In response to this there are two events generated, e_2, when the next customer arrives, and e_3, where this customer finishes service.

Cashier/Queue Example



4. The simulation clock proceeds to time t_2, and processes event e_2.
5. Event e2 is the arrival of a new customer. As a customer is already being serviced, we place it in a waiting queue (and will also want to save the time it entered the waiting queue).
6. In response to this arrival, we need to generate the arrival time of the next customer, A2, and generate the event for it's arrival, e_4.
7. The simulation proceed to the time of the next event, c_1 and processes event e_3, when the first customer is done with service.

Cashier/Queue Example



8. When the first customer is done with service, (event e_3) we can save it's service time, and begin to service the next customer, removing it from the queue.
9. We generate another event (e_5) for the next customer finishing this service at the current time + S_1.
10. The simulation proceeds to time t3 to process event e_4. For event e_4, the next customer arrives. We could generate another time for the next customer arrive but this is a short example, so we'll stop the simulation from generating any more events.

Programming Discrete-Event Simulations

Cashier/Queue Rules

```
Time 0:
   1. Generate customer arrival event at time 0 + A0.

Customer Arrival Event (at time $t_n$):
    Generate next customer arrival event at time $t_n + A_n$.
   if (queue empty)
    1.Generate service finished event at time $t_n + S_n$.
  else
    1. Add customer to queue (store the start of it’s waiting time).

Service Finished Event (at time $t_n$):
   record wait time of customer
   if (queue not empty)
    1. Remove customer from front of queue.
    2. Generate service finished event at time $t_n + S_n$.
```

Cashier/Queue Requirements

Need to store:

- current simulation time
- customer waiting times (so we can calculate average wait time)
- customer service times (so we can calculate average service time)

Need a data structure (a class or struct) for each event, which stores (at least) the event type, and its start time.

Need a data structure to store events (so we can get the next event in time quickly — min heap).

Need a data structure to hold customers waiting in line (a linked list or queue will work great here, as we need something with first in, first out (FIFO) operation).

Need random number generators to generate events from given probability distributions.

Implementing the Simulation Clock and Event Queue

A common (perhaps the best) data structure for the event queue is a Min Heap. This allows fast insertion of events (with random times) - log(n) and fast removal of the event with the minimum time log(n). Further, a min heap can be easily implemented in an array, for fast access and low memory overhead.

General Discrete Event Simulation Requirements

System state: The collection of state variables necessary to describe the system at a particular time.

Simulation clock: A variable giving the current value of simulated time.

Event list: A list containing the times when each type of event will occur.

Statistical counters: Variables used for storing statistical information about system performance.

Initialization routine: A subprogram to initialize the simulation model at time 0 (and create the first event(s)).

Event routines: A subprogram that updates the system state when a particular type of event occurs (there is one event routine for each type).

Library routines: A set of subprograms used to generate random observations from probability distributions that were determined as part of the simulation model.

Report generator: A subprogram that computes estimates (from the statical counters) of the desired measures of performance and produces a report when the simulation ends.

Main program: A subprogram that initializes the simulation and processes all the events with the appropriate subroutines.

Simulation Pseudocode

```
class Event {
   int time;
   enum type = {ARRIVAL, LEAVE, …};
   …
}

int sim_time, end_time;
min_heap<Event> events();

int main() {
   //initialize simulation
   //generate first event
   events.add(new Event(…));
   while (sim_time < end_time) {
       Event e = events.remove();
       sim_time = e.time;
       switch (e.type) {
           case ARRIVAL:
               //process arrival
           case LEAVE:
               //process leave
           case …
               //process other events
       }
   }
   //simulation is over, print
   //simulation statistics.
}
```

Event Simulation vs. Process Simulation

This type of simulation implementation is quite common, and typically called an *event-scheduling approach*, as events are generated and scheduled to happen sometime in the simulated future.

There is another (less common) approach to simulation modeling called the *process approach*, where the simulation is viewed through the individual entities involved (e.g., cars in a traffic simulation) and the code instead describes the "experience" of a "typical" entity as it "flows" through the system. While this often requires custom code, behind the scenes is still typically an *event-scheduling approach*.

Generating Reports

Problem Statement

Consider the single server queuing system, where the inter arrival times A_0, …, A_n, are *independent and identically distributed (IID),* and the times customers spend being serviced S_0, …, S_n, also generated IID.

Identically distributed means that the inter arrival times use the same probability distribution, and independent means the A_1 does not depend on A_0, A_2 does not depend on A_0 or A_1, and so on.

We want to measure the performance of this system in three ways:

1. What is the expected time a customer will spend in the queue?
2. What is the average length of the queue?
3. What percentage of time is the server busy?

Average expected delay

We denote the expected average delay in a queue, given a simulation of *n* customers, as *d(n)*.

"Expected" here means that on any given run of the simulation, what do we expect the delay to be? Note that as the arrival and service times of each customer is randomly generated, the actual average time customers spend in any given simulation will change from simulation to simulation (we cannot just run it once to get this value).

Given this, we want to estimate *d(n)* given a very large number of customers (ideally infinite — but that's not possible). Given this large number of *n* customers, we can define an estimator for d(n), given a number of customer delays, $D_1, \ldots, D_n$:

$$\hat{d}(n) = \frac{\sum_{i=1}^{n} D_i}{n}$$

Average expected delay

$$\hat{d}(n) = \frac{\sum_{i=1}^{n} D_i}{n}$$

In this case, the hat over *d* means that *d* is an estimator. Also note that any given D_i could also be 0 (as there will be times in the simulation when the queue is empty).

Note that this is essentially just the average of D_i, all the times the customers spend in line.

Also note that this estimator is based on a sample size of 1 (one simulation run), which is not worth too much (but there is more on that later, in Chapters 9-12).

Average queue length

Calculating the average queue length is rather different than calculating the average wait time, namely because it is calculated over a continuous about of time, rather than a discrete (countable) number of customers.

We can define Q(t) as the number of customers in queue at time t, for any real number t >= 0. T(n) can be the time required to observe the *n* delays in the queue (assuming our simulation runs for *n* delays instead of a fixed amount of time).

Given this, we can define p_i to be the expected proportion (which will be between 0 and 1) of time that Q(t) is equal to i, which leads us to a definition for the average queue length q(t):

$$q(t) = \sum_{i=0}^{\infty} i p_i$$

Average queue length

$$q(t) = \sum_{i=0}^{\infty} i p_i$$

To estimate q(t) from a simulation, we can replace p_i with the expected proportion of time p-hat i:

$$\hat{q}(t) = \sum_{i=0}^{\infty} i \hat{p}_i$$

Where p-hat i is the observed proportion of time *during a simulation* where there were i customers in the queue.

This can be simplified for computation by letting T_i be the total time during the simulation where the queue is length i, so in this case:

$$T(n) = T_0 + T_1 + ...$$

$$\hat{p}_i = \frac{T_i}{T(n)}$$

Average queue length

$$T(n) = T_0 + T_1 + \ldots$$

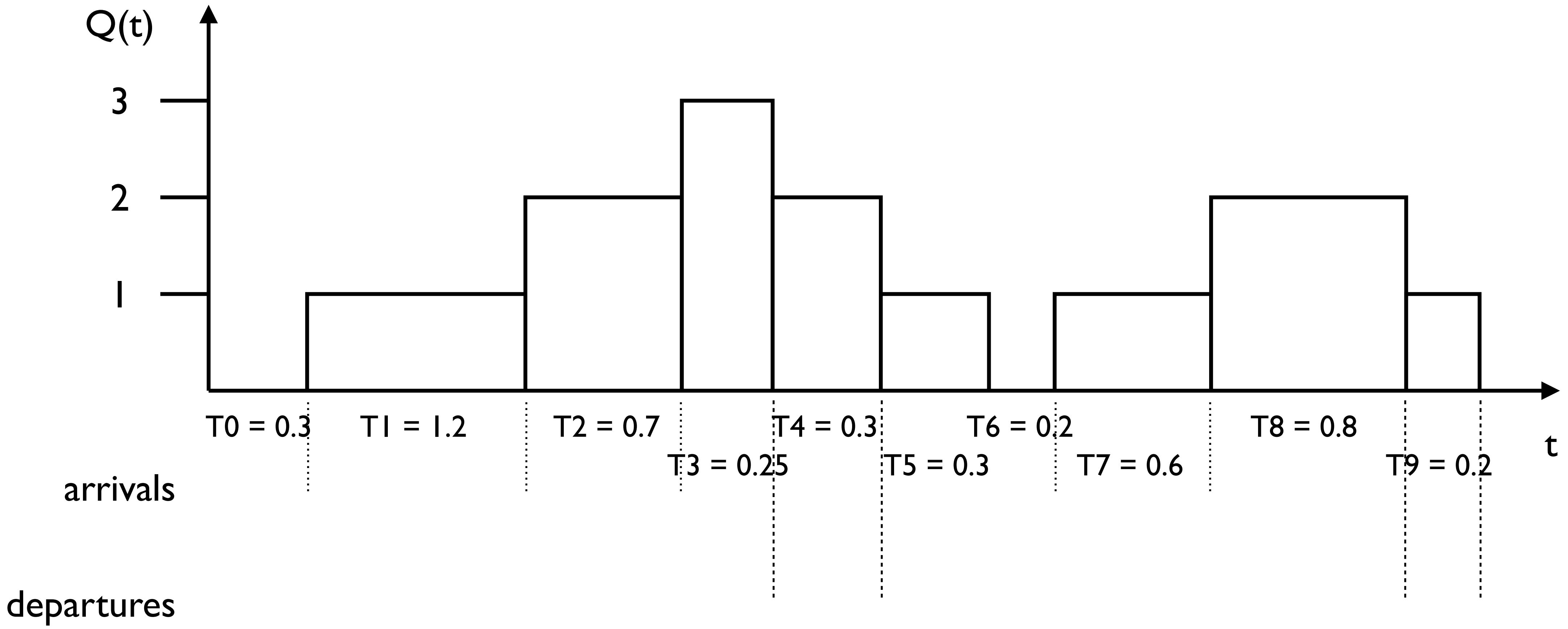
$$\hat{p}_i = \frac{T_i}{T(n)}$$

This gives us:

$$\hat{q}(n) = \frac{\sum_{i=0}^{\infty} iT_i}{T(n)}$$

This may seem a bit complicated so far, but lets graph this.

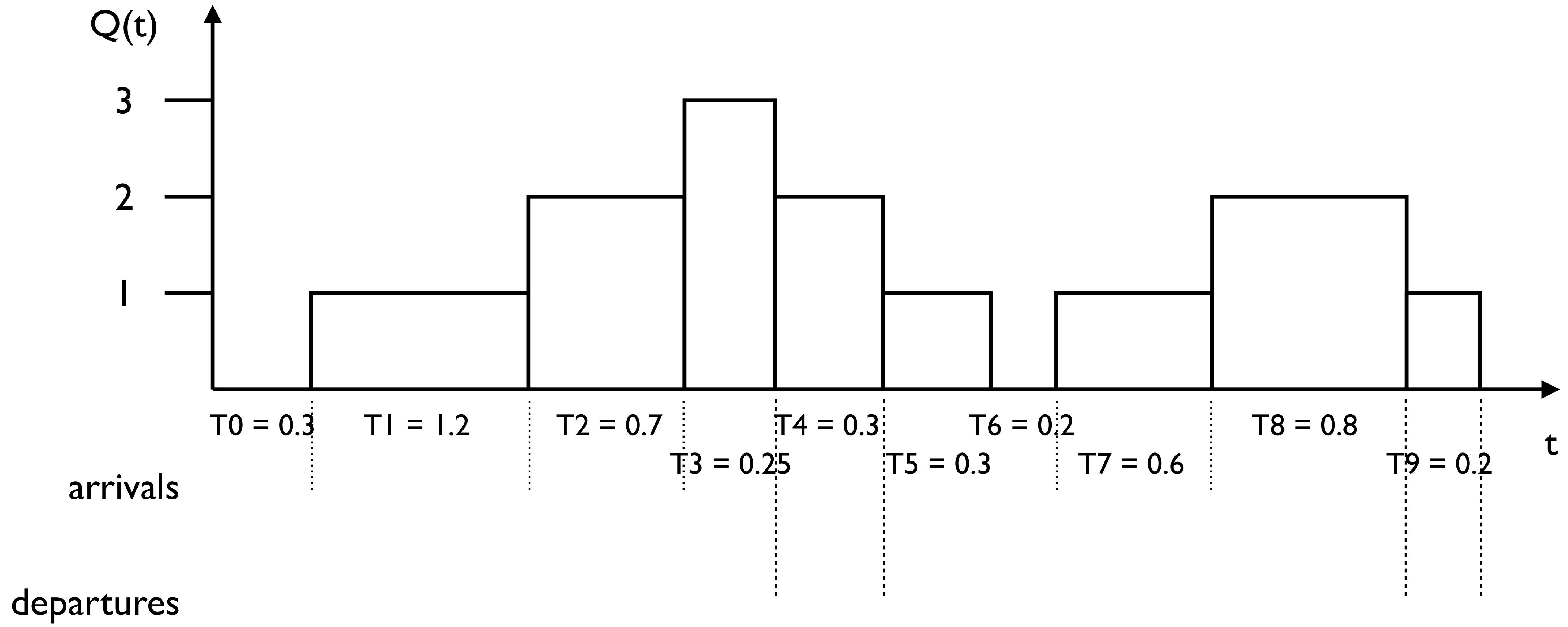
Average queue length



If we plot this, you can notice that q-hat(n) (the expected average queue length) is essentially the area under the non-continuous function Q(t), or the integral of Q(t):

$$\sum_{i=0}^{\infty} iT_i = \int_0^{T(n)} Q(t)\mathrm{d}t$$

Average queue length



Therefore we can express the expected average queue length as:

$$\hat{q}(n) = \frac{\int_0^{T(n)} Q(t)\mathrm{d}t}{T(n)}$$

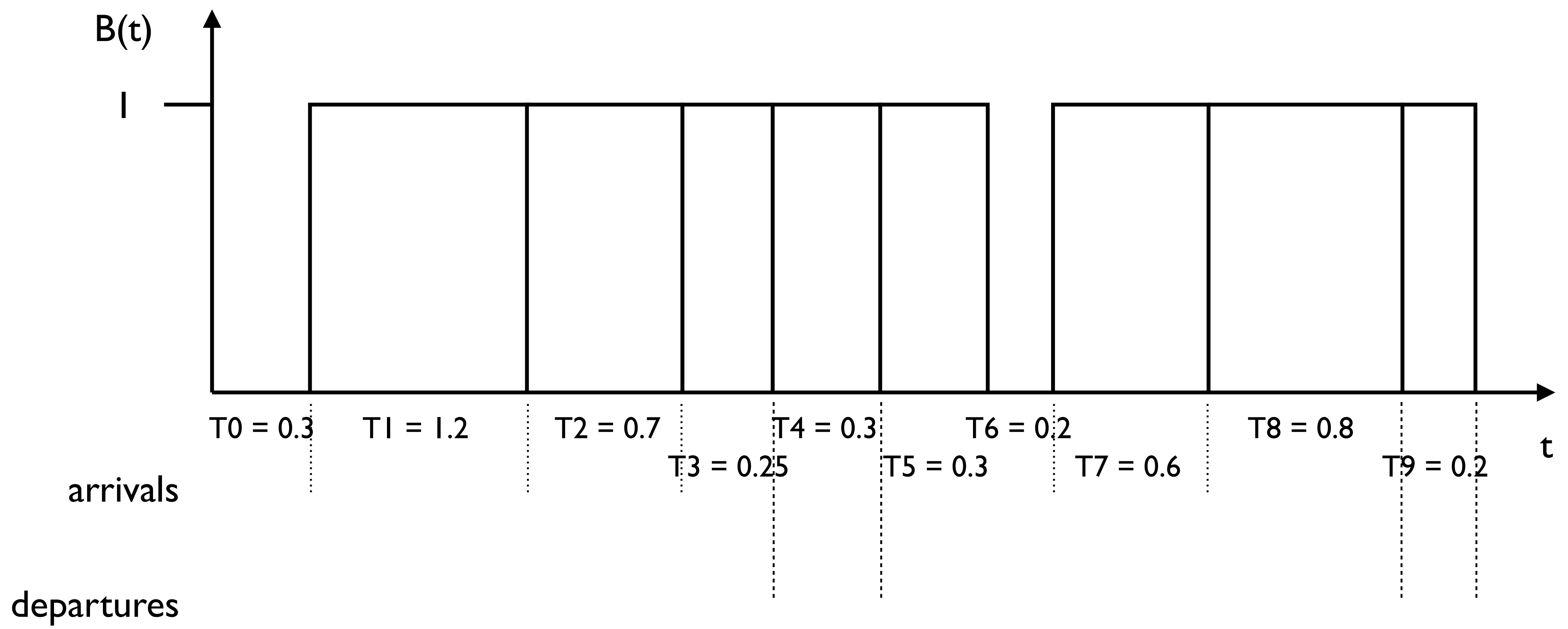
Average queue length

Therefore we can express the expected average queue length as:

$$\hat{q}(n) = \frac{\int_0^{T(n)} Q(t)\mathrm{d}t}{T(n)}$$

While this might look complicated mathematically, it is rather easy to calculate over the course of the simulation. Whenever a customer arrives or leaves, take the number of customers in the queue, and multiply it by the time since the last event. Add this to a summation variable, and increment a counter. At the end of the simulation divide the summation by the counter and you have the average queue length.

Server busy percentage



We can calculate the busy time of the server (or the server *utilization*) similar to the average queue length, except in this case it's a bit simpler as the server is either busy or not busy. Where B(t) = 1 if the server is busy at time t, and B(t) = 0 if the server is not busy at time t, we can calculate the expected utilization of the server, u-hat(n) as:

$$\hat{u}(n) = \frac{\int_0^{T(n)} B(t)\mathrm{d}t}{T(n)}$$

Discrete and Continuous Time Statistics

Given these definitions, we can call the average number of customers in the queue a *discrete time statistic*, as it is defined relative to the collection of random variables $\{D_i\}$ that have a discrete time index *i = 1, 2, ...* .

The average number of customers in the queue and the server utilization are *continuous time statistics* as they are defined based on the collection of random variables $\{Q(t)\}$ and $\{B(t)\}$, where t is a continuous time parameter between 0 and infinity.

These are the standard types of statistics from a simulation, for example if we are interested in the maximum of all delays in the queue observed, this would be a deterministic time statistic and could be calculated similar to the average queue length (actually its a lot easier than that).

If we are interested in the proportion of time that the queue contained at least five customers, this would be a continuous time statistic and we could calculate it similar to the server utilization time.