Making small modifications to our simple discrete-event simulations is non-trivial

- Add feedback to ssq2
- Add delivery lag to sis2

Next-event simulation is a more general approach to discrete-event simulation

- System state
- Events
- Simulation clocks
- Event scheduling
- Event list
The *state* of a system is a complete characterization of the system at an instance in time.

- Conceptual model - abstract collection of variables and how they evolve over time
- Specification model - collection of *mathematical variables* together with logic and equations
- Computational model - collection of *program variables* systematically updated

**Example 5.1.1** State of ssq is number of jobs in the node

**Example 5.1.2** State of sis is current inventory level
An *event* is an occurrence that *may change* the state of the system.

**Example 5.1.3** For ssq, events are arrivals or completion of a job.
- With feedback, the state *may change*.

**Example 5.1.4** For sis with delivery lag, events are demand instances, inventory reviews, and arrival of orders.

We can define artificial events:
- Statistically sample the state of the system
- Schedule an event at a prescribed time
The *simulation clock* represents the current value of simulated time.

Discrete-event simulations lack definitive simulated time. As a result, it is difficult to generalize or embellish models.

**Example 5.1.5** It is hard to reason about `ssq2` because there are effectively two simulation clocks.

- Arrival times and completion times are not synchronized.

**Example 5.1.6** In `sis2`, the only event is inventory review.

- The simulation clock is integer-valued and we aggregate all demand.
Definitions and Terminology - Event Scheduling & Event List

- It is necessary to use a time-advance mechanism to guarantee that events occur in the correct order.
- Next-event time advance is typically used in discrete-event simulation.
- To build a next-event simulation:
  - construct a set of state variables
  - identify the event types
  - construct a set of algorithms that define state changes for each event type
- The event list is the data structure containing the time of next occurrence for each event type.
Algorithm 5.1.1

1. **Initialize** - set simulation clock and first time of occurrence for each event type
2. **Process current event** - scan event list to determine most imminent event; advance simulation clock; update state
3. **Schedule new events** - new events (if any) are placed in the event list
4. **Terminate** - Continue advancing the clock and handling events until termination condition is satisfied

- The simulation clock runs asynchronously; inactive periods are ignored
- Clear computational advantage over *fixed-increment* time-advance mechanism
The state variable \( l(t) \) provides a complete characterization of the state of a ssq.

\[
\begin{align*}
l(t) &= 0 & \iff & \quad q(t) = 0 \quad \text{and} \quad x(t) = 0 \\
l(t) > 0 & \iff \quad q(t) = l(t) - 1 \quad \text{and} \quad x(t) = 1
\end{align*}
\]

Two events cause this variable to change:

1. An arrival causes \( l(t) \) to increase by 1
2. A completion of service causes \( l(t) \) to decrease by 1
The initial state \( l(0) \) can have any non-negative value, typically 0.

The terminal state can be any non-negative value.

- Assume at time \( \tau \) arrival process stopped. Remaining jobs processed before termination.

Some mechanism must be used to denote an event impossible.

- Only store possible events in event list.
- Denote impossible events with event time of \( \infty \).
The simulation clock (current time) is $t$
The terminal ("close the door") time is $\tau$
The next scheduled arrival time is $t_a$
The next scheduled service completion time is $t_c$
The number in the node (state variable) is $l$
Algorithm 5.1.2

\( l = 0; \ t = 0.0; \)
\( t_c = \infty; \ t_a = \text{GetArrival}(); /* initialize the event list */ \)
while ((\( t_a < \tau \)) or (\( l > 0 \))) {
    \( t = \min(t_a, t_c); /* scan the event list */ \)
    if (\( t == t_a \)) { /* process an arrival */
        \( l++; \)
        \( t_a = \text{GetArrival}(); \)
        if (\( t_a > \tau \))
            \( t_a = \infty; \)
        if (\( l == 1 \))
            \( t_c = t + \text{GetService}(); \)
    } else { /* process a completion */
        \( l--; \)
        if (\( l > 0 \))
            \( t_c = t + \text{GetService}(); \)
        else
            \( t_c = \infty; \)
    }
}

Discrete-Event Simulation: A First Course  Section 5.1: Next–Event Simulation
In ssq3, number represents \( l(t) \) and structure t represents time
- the event list \( t.\text{arrival} \) and \( t.\text{completion} \) (\( t_a \) and \( t_c \) from Algorithm 5.1.2);
- the simulation clock \( t.\text{current} \) (\( t \) from Algorithm 5.1.2)
- the next event time \( t.\text{next} \) (min(\( t_a \), \( t_c \)) from Algorithm 5.1.2)
- the last arrival time \( t.\text{last} \)

Time-averaged statistics are gathered with the structure area
- \( \int_{0}^{t} l(s) \, ds \) evaluated as \( \text{area.node} \)
- \( \int_{0}^{t} q(s) \, ds \) evaluated as \( \text{area.queue} \)
- \( \int_{0}^{t} x(s) \, ds \) evaluated as \( \text{area.service} \)
Programs ssq2 and ssq3 simulate exactly the same system
The two have different *world views*
- ssq2 naturally produces job-averaged statistics
- ssq3 naturally produces time-averaged statistics

The programs should produce *exactly* the same statistics
- To do so requires *rns*

**ssq2**: based on ``process interaction''
**ssq3**: based on ``event-scheduling''
else { /* process a completion of service */
    if (GetFeedback() == 0) { /* this statement is new */
        index++;
        number--;  
    }
}

Alternate Queue Disciplines

![Diagram of queue disciplines](image)
 Finite Service Node Capacity

if (t.current == t.arrival) {
    if (number < CAPACITY) {
        number++;
        if (number == 1)
            t.completion = t.current + GetService();
    }
    else
        reject++;
    t.arrival = GetArrival();
    if (t.arrival > STOP) {
        t.last = t.current;
        t.arrival = INFINITY;
    }
}
Random Sampling

The structure of ssq3 facilitates adding sampling
- Add a sampling event to the event list
  - Sample deterministically, every $\delta$ time units
  - Sample Randomly, every $Exponential(\delta)$ time units