**Event Scheduling Algorithm**

- Simulation tables constructed in spreadsheet are ad-hoc for individual systems.
  - For larger scale applications, the ad-hoc approach can become unbearably complex.
- A more general, common framework for modeling complex systems by discrete event simulation is needed.
  - Off the shelf software can be developed.
  - Once a system is specified, the simulation table is automatically generated.
- How to implement the framework in Java is studied.
- Reference: 3.1, 4.4

---

**Discrete-Event Simulation**

- A **discrete-event simulation** models a system whose state changes at discrete points in time.
  - Produce a sequence of system snapshots that collectively represent system evolution over time.
- Content of a snapshot
  - Simulated time \( t \) and system state at time \( t \)
  - Activities current in progress and when each will end
  - Current values of cumulative statistics and counters

<table>
<thead>
<tr>
<th>Clock</th>
<th>System State</th>
<th>Future Event List</th>
<th>Cumulative Statistics and Counters</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )</td>
<td>( (x, y, z, \ldots) )</td>
<td>( (e_1, t_1) )</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( (x_2, x_3, \ldots) )</td>
<td>( (e_2, t_2) )</td>
<td>( \ldots )</td>
<td></td>
</tr>
<tr>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td></td>
</tr>
</tbody>
</table>

---

**Event Notice and Future Event List**

- **Event notice**
  - A record of an event to occur at the current time or some future time
  - Include at least event type and event time
- **A future event list (FEL)** is a list of event notices for future events, ordered by event time.
- **A delay** is a duration of time of unspecified length.
  - The length is unknown until the duration ends.
  - Activity is a duration of specified length.

---

**Event Scheduling Based on FEL**

- At any Clock time \( t \), FEL contains each scheduled future event \( (at t_i) \), ordered by event times: \( t < t_1 < t_2 < \ldots < t_n \).
- After system snapshot at Clock=\( t \) has been updated, Clock is advanced to the time of **imminent event** \( t_j \).
- Imminent event notice is removed from FEL and is executed.
  - Snapshot for time \( t_j \) is created.
  - New future events are generated and notices placed in FEL.
  - Generally, snapshot for time \( t \) will not be kept.
- Repeat the above until the simulation is over.
Examples of Future Event Generation

- **Bootstrapping** on external arrival
  - At time 0, generate 1st arrival event and place a notice on FEL.
  - When Clock advances to the event time, generate the 2nd arrival.
- **Service completion**
  - When a unit exists, the next service completion event is scheduled.
  - **Primary event** versus **conditional event**
  - Only primary events are maintained in FEL.
- **Runtime-downtime alternation**
  - At time 0, an end-of-runtime event is scheduled.
  - When it 'occurs', an end-of-downtime event is scheduled.

Stopping Event

- Define how long the simulation should run.
  1) **Specifying stopping time**
     - At time 0, set stopping event at future time \( T_E \).
     - Ex A bank that closes at 5:30 pm
  2) **Specifying stopping condition**
     - \( T_E \) is the time of occurrence of some event \( E \).
     - \( T_E \) is unknown at time 0, and is determined by simulation.
     - Ex Simulate 20 bearing changes

Single-Server Queue Simulation in Java

- Ex A grocery checkout counter, where customer interarrival time is **exponentially distributed** with a mean of 4.5 min, and service times are **normally distributed** with a mean of 3.2 min and a standard deviation of 0.6 min.
- Interarrival and service times are modeled as continuous variables (their distributions are covered later).
  - Illustration: normal distribution with mean = 0.5 and stdev = 0.1.
- Treat the Java methods as primitives for now.
  - `exponential(rng, mean)`
  - `normal(rng, mean, sigma)`

Java Classes

- **Sim.java**
  - The main simulation class and launch pad
- **Event.java**
  - Representing an event
- **EventList.java**
  - Representing FEL
- **Queue.java**
  - Representing a FIFO waiting line
### Event.java
- Representing an event
- Main instance variables: event type, event time
- Constructor: Create an event given its type and time.
- Methods
  - `get_type()`: Return event type.
  - `get_time()`: Return event time.
  - `compareTo(another_event)`
    - Return -1 if this event time is earlier than another_event time.
    - Return 0 or 1 accordingly, depending on event time comparison.

### EventList.java
- Representing FEL
- Methods
  - `isEmpty()`: Check if the list is empty.
  - `enqueue(event)`: Insert an event to list.
  - `getMin()`: Return the imminent event in the list.
  - `dequeue()`: Remove most recently accessed event from list.

### Queue.java
- Representing a FIFO waiting line
- Constructor: Create an empty queue.
- Main methods
  - `empty()`: Check if the queue is empty.
  - `enqueue(unit)`: Add a unit to the end of line.
  - `dequeue()`: Return head unit and remove it from line.
  - `peekFront()`: Return head unit without removing it from line.
  - `numElements()`: Return the number of units in line.

### Sim.java
- Method main(): The main simulation method
- Representing system snapshot by a set of variables.
  1) `Initialization()`: Initialize simulation condition
  2) `ProcessArrival()`: Execute an arrival event
  3) `ScheduleDeparture()`: Create a departure event
  4) `ProcessDeparture()`: Execute a departure event
  5) `ReportGeneration()`: Report model response
Variables in the Java Model (1)

- **System state**
  - QueueLength: # waiting customers at the current time
  - NumberInService: Server status

- **Entity set**
  - Customers: Customers waiting in FIFO queue

- **Future event list**
  - FutureEventList: FEL

- **Input parameters**
  - MeanInterArrivalTime: Interarrival time parameter
  - MeanServiceTime, SIGMA: Service time parameters
  - TotalCustomers: # customer to be simulated - stopping criterion

Variables in the Java Model (2)

- **Simulation variable**
  - Clock: Current time of simulation
  - LastEventTime: Time of the previous event

- **Statistical accumulators**
  1) TotalBusy: Total busy time of the server so far
  2) MaxQueueLength: Max length of waiting line so far
  3) SumResponseTime: Sum of system times for all customers departed so far
  4) NumberOfDepartures: # departing customers so far
  5) LongService: # customers whose response time > 4 min

Variables in the Java Model (3)

- **Summary statistics**
  - RHO: Proportion of time when server is busy
  - AVGR: Average customer response time
  - PCA: Proportion of customers with response time > 4 min

```java
main(argv[]) {
    MeanInterArrivalTime = 4.5; TotalCustomers = 1000;
    MeanServiceTime = 3.2; SIGMA = 0.6;
    long seed = Long.parseLong(argv[0]);
    stream = new Random(seed);
    FutureEventList = new EventList();
    Customers = new Queue();
    Initialization();
    while (NumberOfDepartures < TotalCustomers) {
        Event evt = (Event) FutureEventList.getMin();
        FutureEventList.dequeue();
        Clock = evt.get_time();
        if (evt.get_type() == arrival) ProcessArrival(evt);
        else ProcessDeparture();
    }
    ReportGeneration();
}
```
Initialization() {
    Clock = 0.0; QueueLength = 0;
    NumberInService = 0; LastEventTime = 0.0;
    TotalBusy = 0.; MaxQueueLength = 0;
    SumResponseTime = 0;
    NumberOfDepartures = 0;
    LongService = 0;
    Event evt = new Event(arrival,
        exponential(stream, MeanInterArrivalTime));
    FutureEventList.enqueue( evt );
}

ProcessArrival(Event evt) {
    Customers.enqueue(evt);
    QueueLength++;
    if (NumberInService == 0) ScheduleDeparture();
    else TotalBusy += (Clock - LastEventTime);
    if (MaxQueueLength < QueueLength)
        MaxQueueLength = QueueLength;
    Event next_arrival = new Event(arrival, 
        Clock + exponential(stream, MeanInterArrivalTime));
    FutureEventList.enqueue( next_arrival );
    LastEventTime = Clock;
}

ScheduleDeparture() {
    while ((ServiceTime = normal(stream,
        MeanServiceTime, SIGMA)) < 0 );
    Event depart =
        new Event(departure, Clock + ServiceTime);
    FutureEventList.enqueue( depart );
    NumberInService = 1;
    QueueLength--;
}

ProcessDeparture() {
    finished = Customers.dequeue();
    if ( QueueLength > 0 ) ScheduleDeparture();
    else NumberInService = 0;
    response = Clock - finished.get_time();
    SumResponseTime += response;
    if ( response > 4.0 ) LongService++;
    TotalBusy += Clock - LastEventTime;
    NumberOfDepartures++;
    LastEventTime = Clock;
}
Simulation Report

> java Sim 12345
SINGLE SERVER QUEUE SIMULATION - GROCERY STORE CHECKOUT COUNTER
MEAN INTERARRIVAL TIME 4.5
MEAN SERVICE TIME 3.2
STANDARD DEVIATION OF SERVICE TIMES 0.6
NUMBER OF CUSTOMERS SERVED 1000

SERVER UTILIZATION 0.7136864459484095
MAXIMUM LINE LENGTH 10.0
AVERAGE RESPONSE TIME 7.0600148213418015 MINUTES
PROPORTION WHO SPEND FOUR MINUTES OR MORE IN SYSTEM 0.669
SIMULATION RUNLENGTH 4479.704874601935 MINUTES
NUMBER OF DEPARTURES 1000