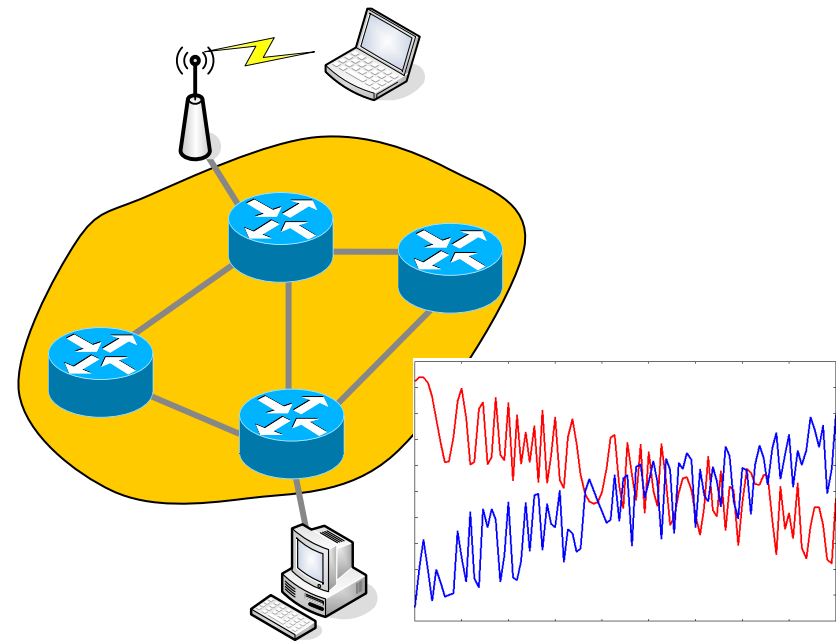


Chapter 3

General Principles



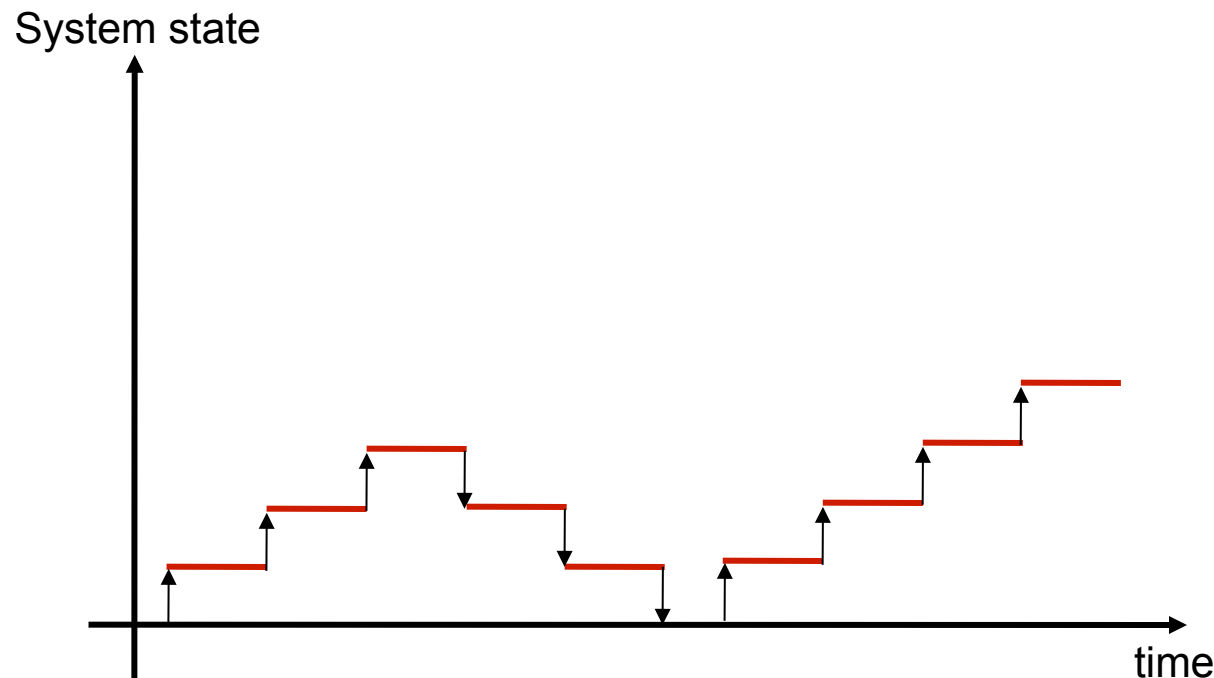
Contents

- Concepts of Discrete-Event Simulation
- The Event Scheduling / Time Advance Algorithm
- World Views
- Manual Simulation using Event Scheduling
- Simulation in Java
- Object-oriented Simulation Framework in Java
- Modeling of Discrete-Event Simulations

Concepts of Discrete-Event Simulation

General Principles: Introduction

- Framework for modeling systems by discrete-event simulation
 - A system is modeled in terms of its state at each point in time
 - This is appropriate for systems where changes occur only at discrete points in time



Concepts in Discrete-Event Simulation

- Concepts of dynamic, stochastic systems that change in a discrete manner

| | |
|--------------|--|
| System | A collection of entities that interact together over time to accomplish one or more goals, e.g., bank, production system, computer system, network. |
| Model | An abstract representation of a system, usually containing structural, logical, or mathematical relationships that describe the system. |
| System state | A collection of variables that contain all the information necessary to describe the system at any time. |
| Entity | An object in the system that requires explicit representation in the model, e.g., people, machines, nodes, packets, server, customer. |
| Attributes | The properties of a given entity , e.g., length of a packet, capacity of a machine. |
| List, Set | A collection of associated entities ordered in some logical fashion in a waiting line. <ul style="list-style-type: none">▪ Holds entities and event notices.▪ Entities on a list are always ordered by some rule, e.g., FIFO, LIFO, or ranked by some attribute, e.g., priority, due date. |
| Event | An instantaneous occurrence that changes the state of a system. |
| Event notice | A record of an event to occur at the current or some future time, along with any associated data necessary to execute the event. |

Concepts in Discrete-Event Simulation

| | |
|------------|--|
| Event list | <p>A list of event notices for future events, ordered by time of occurrence; known as the future event list (FEL) or future event set (FES).</p> <ul style="list-style-type: none">▪ Always ranked by the event time. |
| Activity | <p>A duration of time of specified length, which is known when it begins.</p> <ul style="list-style-type: none">▪ Represents a service time, interarrival time, or any other processing time whose duration has been characterized by the modeler. The duration of an activity can be specified as:<ul style="list-style-type: none">• Deterministic: Always 5 time units• Statistical: Random draw from {2, 5, 7}• A function: Depending on system variables and entities▪ The duration of an activity is computable when it begins▪ The duration is not affected by other events▪ To track activities, an event notice is created for the completion time, e.g., let clock=100 and service with duration 5 time units is starting<ul style="list-style-type: none">• Schedule an “end of service”-event for clock + 5 = 105 |
| Delay | <p>A duration of time of unspecified indefinite length, which is not known until it ends.</p> <ul style="list-style-type: none">▪ Customer’s delay in waiting line depends on the number and service times of other customers.▪ Typically a desired output of the simulation run. |
| Clock | <p>A variable representing the simulated time.</p> |

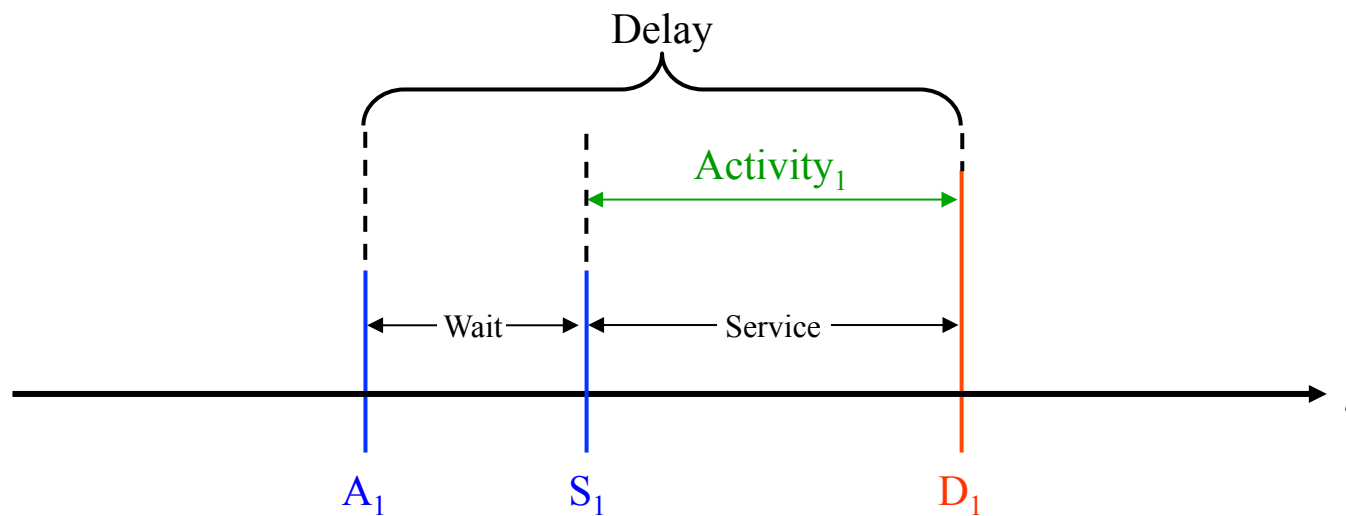
Concepts in Discrete-Event Simulation

Activity vs. Delay

- Activity
 - Activity is known as **unconditional wait**
 - End of an activity is an event, for this an event notice is placed in the future event list
 - This event is a **primary event**
- Delay
 - Delay is known as **conditional wait**
 - Delays are managed by placing the entity on another list, e.g., representing a waiting line
 - Completion of delay is a **secondary event**, but they are not placed in the future event list

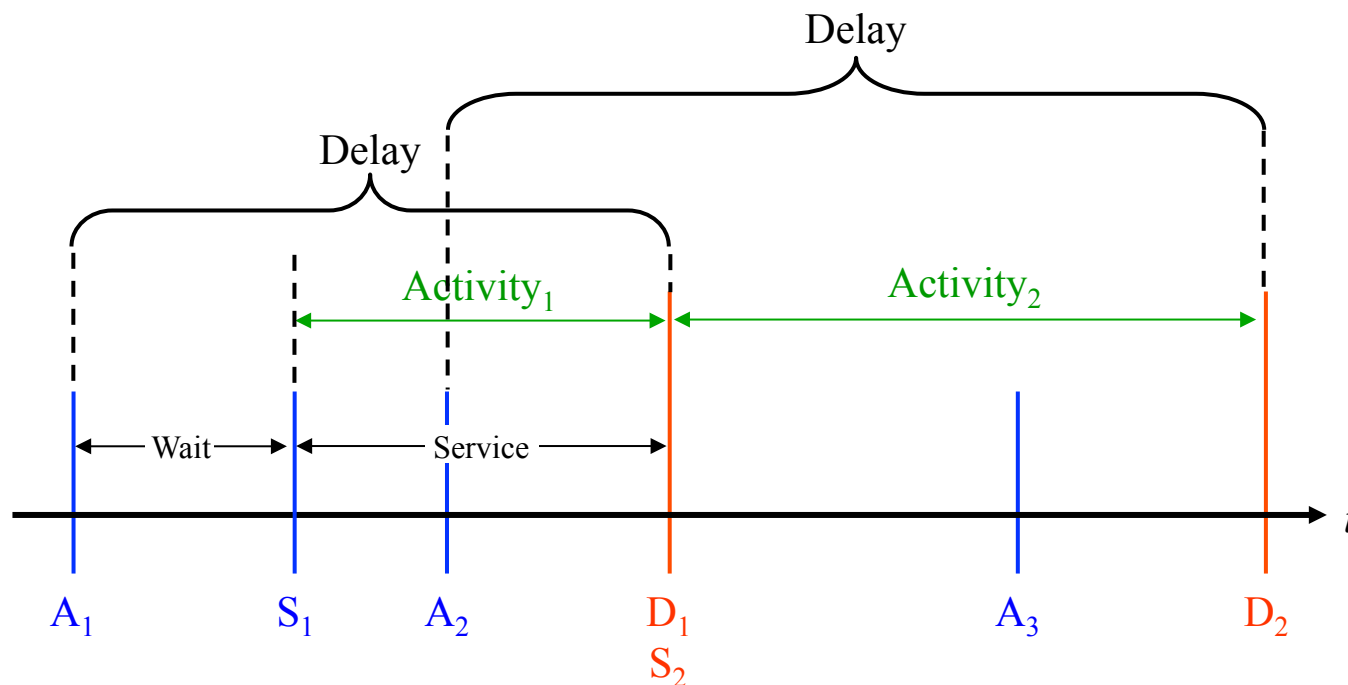
Concepts in Discrete-Event Simulation

- Activity vs. Delay



Concepts in Discrete-Event Simulation

- Activity vs. Delay



Concepts in Discrete-Event Simulation

- Activity vs. Delay



Concepts in Discrete-Event Simulation: Example

Consider Call Center Example from Chapter 2

- System state at time t is given by $[L_Q(t), L_A(t), L_B(t)]$
 - $L_Q(t)$: Number of callers waiting to be served at time t
 - $L_A(t)$: 0 or 1 to indicate Able as being idle or busy at time t
 - $L_B(t)$: 0 or 1 to indicate Baker as being idle or busy at time t
- Entities
 - Neither callers nor the servers are explicitly represented
- Events
 - Arrival event
 - Service completion by Able
 - Service completion by Baker
- Activities
 - Interarrival time of callers
 - Service time by Able
 - Service time by Baker
- Delay
 - A caller's waiting time in queue until Able or Baker becomes free

Concepts in Discrete-Event Simulation

- A model consists of
 - static description of the model and
 - the dynamic relationships and interactions between the components
- Some questions that need to be answered for the dynamic behavior
 - Events
 - How does each event affect system state, entity attributes, and set contents?
 - Activities
 - How are activities defined?
 - What event marks the beginning or end of each activity?
 - Can the activity begin regardless of system state, or is its beginning conditioned on the system being in a certain state?
 - Delays
 - Which events trigger the beginning (and end) of each delay?
 - Under what condition does a delay begin or end?
 - System state initialization
 - What is the system state at time 0?
 - What events should be generated at time 0 to “prime” the model – that is, to get the simulation started?

Concepts in Discrete-Event Simulation

- A discrete-event simulation proceeds by producing a sequence of **system snapshots** over time
- A snapshot of the system at a given time includes
 - System state
 - Status of all entities
 - Status of all sets
 - Sets are used to collect required information for calculating performance metrics
 - Future event list (FEL)
 - Statistics

| Clock | System state | Entities and attributes | Set 1 | Set 2 | ... | Future event list (FEL) | Statistics |
|-------|----------------|-------------------------|-------|-------|-----|---|------------|
| t | (x, y, z, ...) | | | | | (3,t ₁) – Type 3 event to occur at t ₁ | |
| ... | ... | ... | ... | ... | ... | ... | ... |

The Event Scheduling / Time Advance Algorithm

Event-scheduling/Time-advance algorithm

- Future event list (FEL)
 - All event notices are chronologically ordered in the FEL
 - At current time t , the FEL contains all scheduled events
 - The event times satisfy: $t < t_1 \leq t_2 \leq t_3 \leq \dots \leq t_n$
 - The event associated with t_1 is the **imminent event**, i.e., the next event to occur
- Scheduling of an event
 - At the beginning of an activity the duration is computed and an end-of-activity event is placed on the future event list
- The content of the FEL is changing during simulation run
 - Efficient management of the FEL has a major impact on the performance of a simulation run
 - Data structures and algorithms
 - Array, List, Tree, Heap etc.

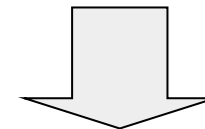
Event-scheduling/Time-advance algorithm

Event-scheduling/Time-advance algorithm

- Step 1: Remove the event notice for the imminent event from FEL
 - event $(3, t_1)$ in the example
- Step 2: Advance Clock to imminent event time
 - Set clock = t_1
- Step 3: Execute imminent event
 - update system state
 - change entity attributes
 - set membership as needed
- Step 4: Generate future events and place their event notices on FEL
 - Event $(4, t^*)$
- Step 5: Update statistics and counters

Old system snapshot at time t

| Clock | State | ... | Future event list |
|-------|-----------|-----|--|
| t | $(5,1,6)$ | | $(3,t_1)$ – Type 3 event to occur at t_1 |
| | | | $(1,t_2)$ – Type 1 event to occur at t_2 |
| | | | $(1,t_3)$ – Type 1 event to occur at t_3 |
| | | | ... |
| | | | $(2,t_n)$ – Type 2 event to occur at t_n |

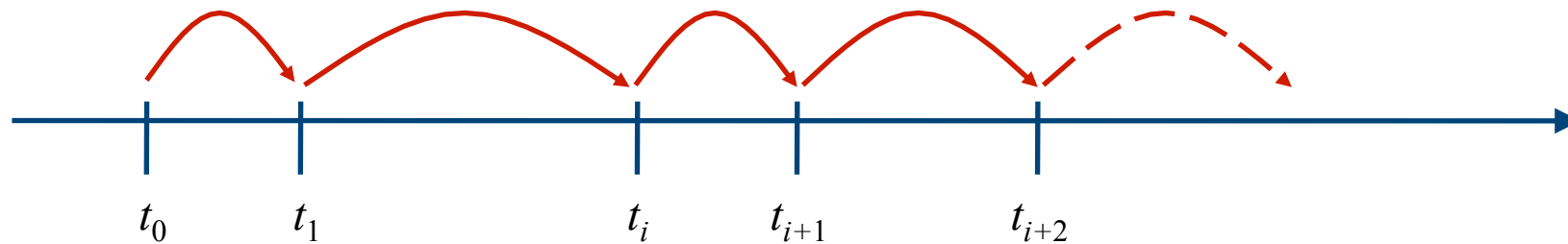


New system snapshot at time t_1

| Clock | State | ... | Future event list |
|-------|-----------|-----|--|
| t_1 | $(5,1,5)$ | | $(1,t_2)$ – Type 1 event to occur at t_2 |
| | | | $(4,t^*)$ – Type 4 event to occur at t^* |
| | | | $(1,t_3)$ – Type 1 event to occur at t_3 |
| | | | ... |
| | | | $(2,t_n)$ – Type 2 event to occur at t_n |

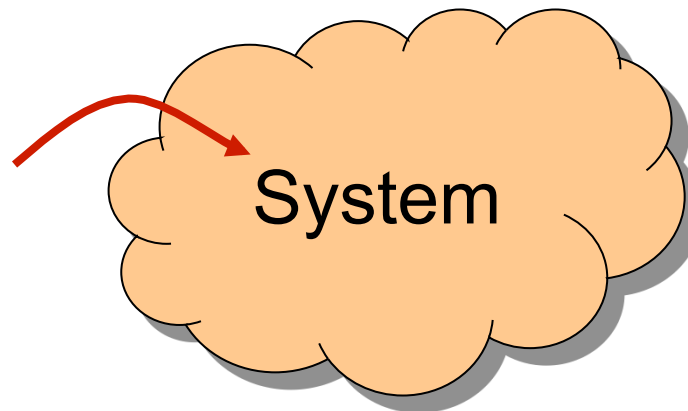
Event-scheduling/Time-advance algorithm

- Evolution of the system state (snapshots)



Event-scheduling/Time-advance algorithm

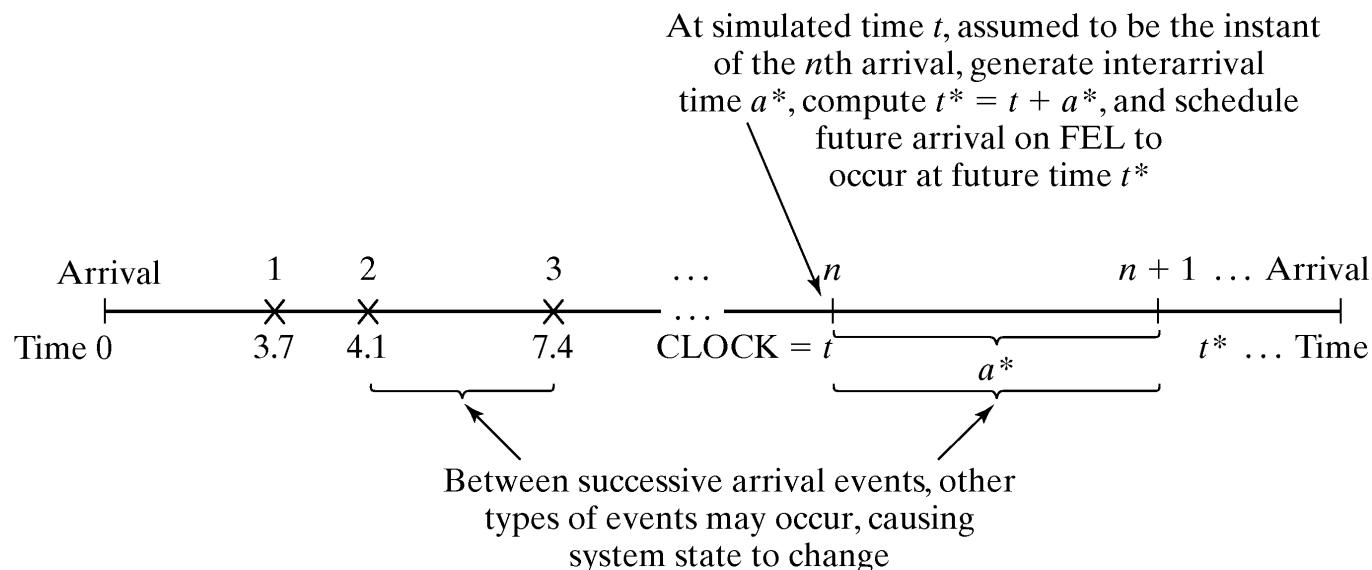
- System snapshot at time 0
 - Initial conditions
 - Generation of exogenous events
 - Exogenous event, is an event which happens outside the system, but impinges on the system, e.g., arrival of a customer.



Event-scheduling/Time-advance algorithm

Generation of events

- Arrival of a customer
 - At $t=0$ first arrival is generated and scheduled
 - When the clock is advanced to the time of the first arrival, a second arrival is generated
 - Generate an interarrival time a^*
 - Calculate $t^* = \text{clock} + a^*$
 - Place event notice at t^* on the FEL
- } Bootstrapping



Event-scheduling/Time-advance algorithm

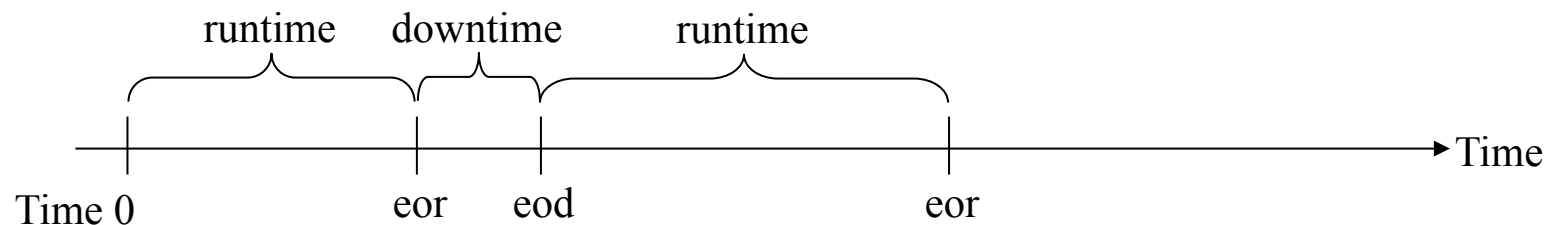
Generation of events

- Service completion of a customer
 - A customer completes service at t
 - If the next customer is present a new service time s^* is generated
 - Calculate $t^* = \text{clock} + s^*$
 - Schedule next service completion at t^*
 - Additionally: Service completion event will be scheduled at the arrival time, when there is an idle server
 - Service time is an activity
 - Beginning service is a conditional event
 - Conditions: Customer is present and server is idle
 - Service completion is a primary event

Event-scheduling/Time-advance algorithm

Generation of events

- Alternate generation of runtimes and downtimes
 - At time 0, the first runtime will be generated and an **end-of-runtime** event will be scheduled
 - Whenever an **end-of-runtime** (eor) event occurs, a downtime will be generated, and an **end-of-downtime** (eod) event will be scheduled
 - At the end-of-downtime event, a runtime is generated and an end-of-runtime event is scheduled
 - Runtimes and downtimes are activities
 - end-of-runtime and end-of-downtime are primary events



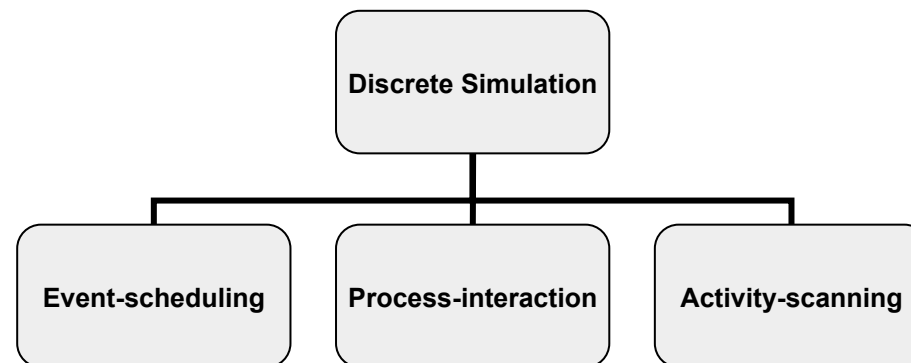
Event-scheduling/Time-advance algorithm

- Stopping a simulation
 1. At time 0, schedule a stop simulation event at a specified future time T_E ➡ Simulation will run over $[0, T_E]$
 2. Run length T_E is determined by the simulation itself.
 - T_E is not known ahead.
 - Example 1: T_E = When FEL is empty
 - Example 2: T_E = When k -th customer leaves the system

World Views

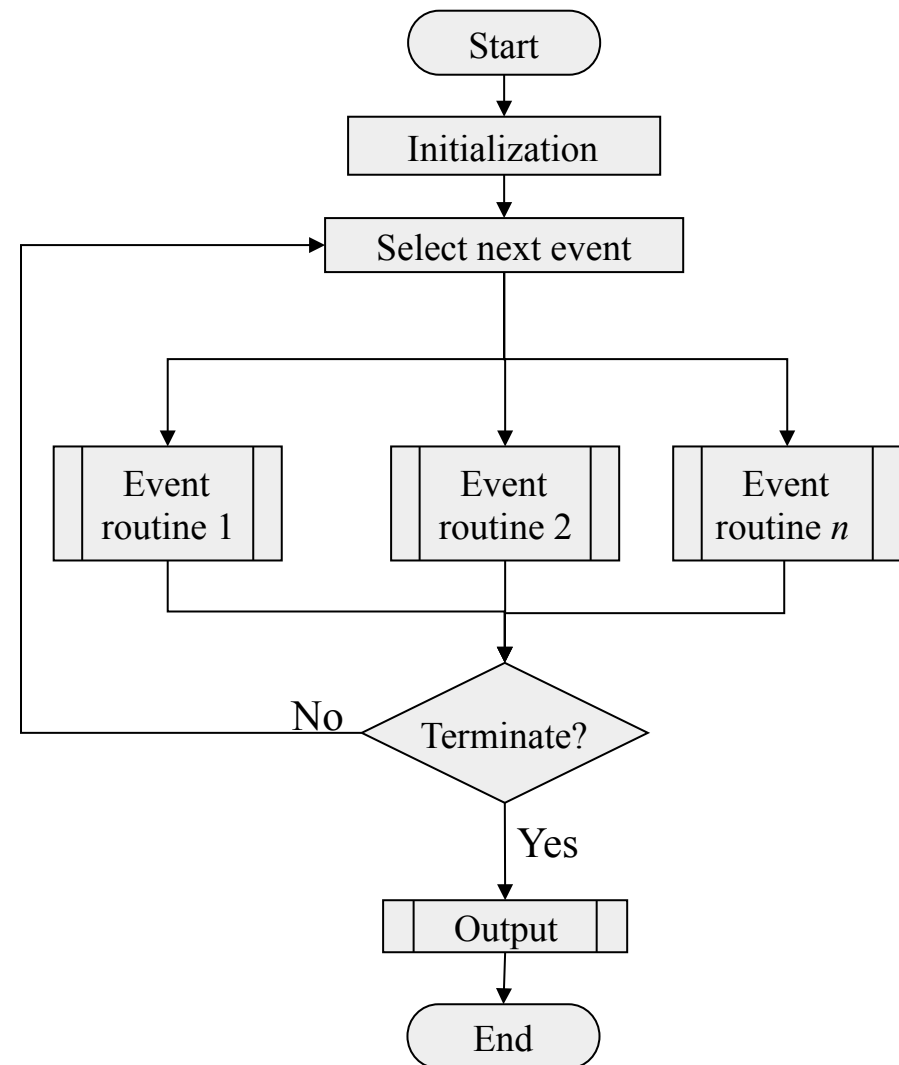
World Views

- World view
 - A world view is an orientation for the model developer
 - Simulation packages typically support some world views
 - Here, only world views for discrete simulations



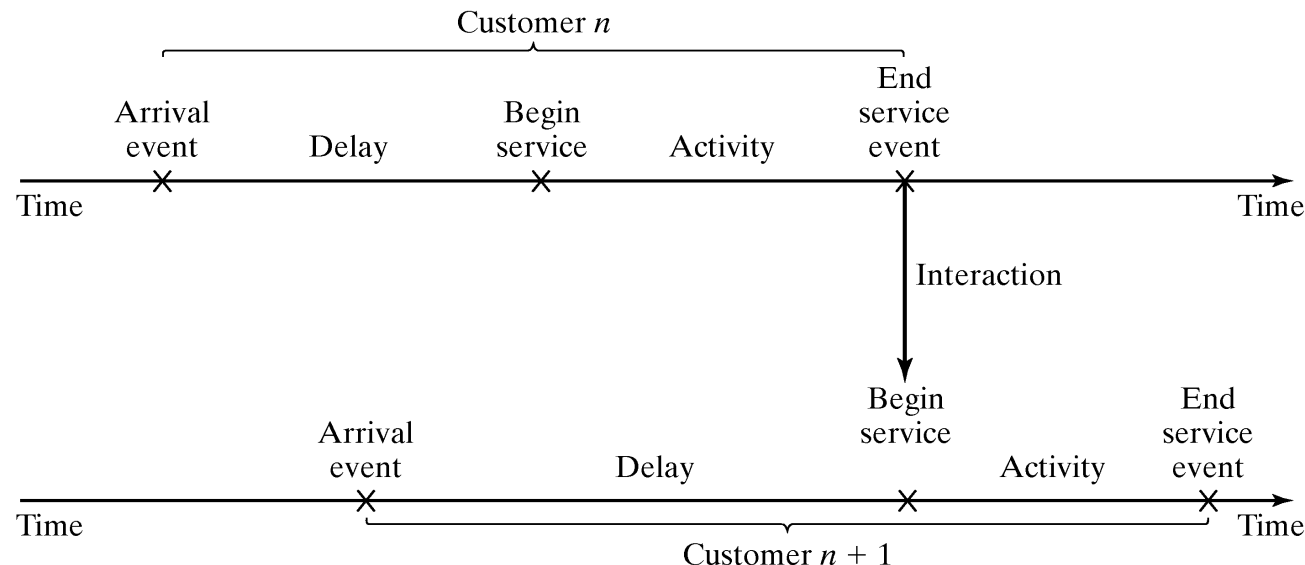
World Views

- Event-scheduling
 - Focus on events
 - Identify the entities and their attributes
 - Identify the attributes of the system
 - Define what causes a change in system state
 - Write a routine to execute for each event
 - Variable time advance



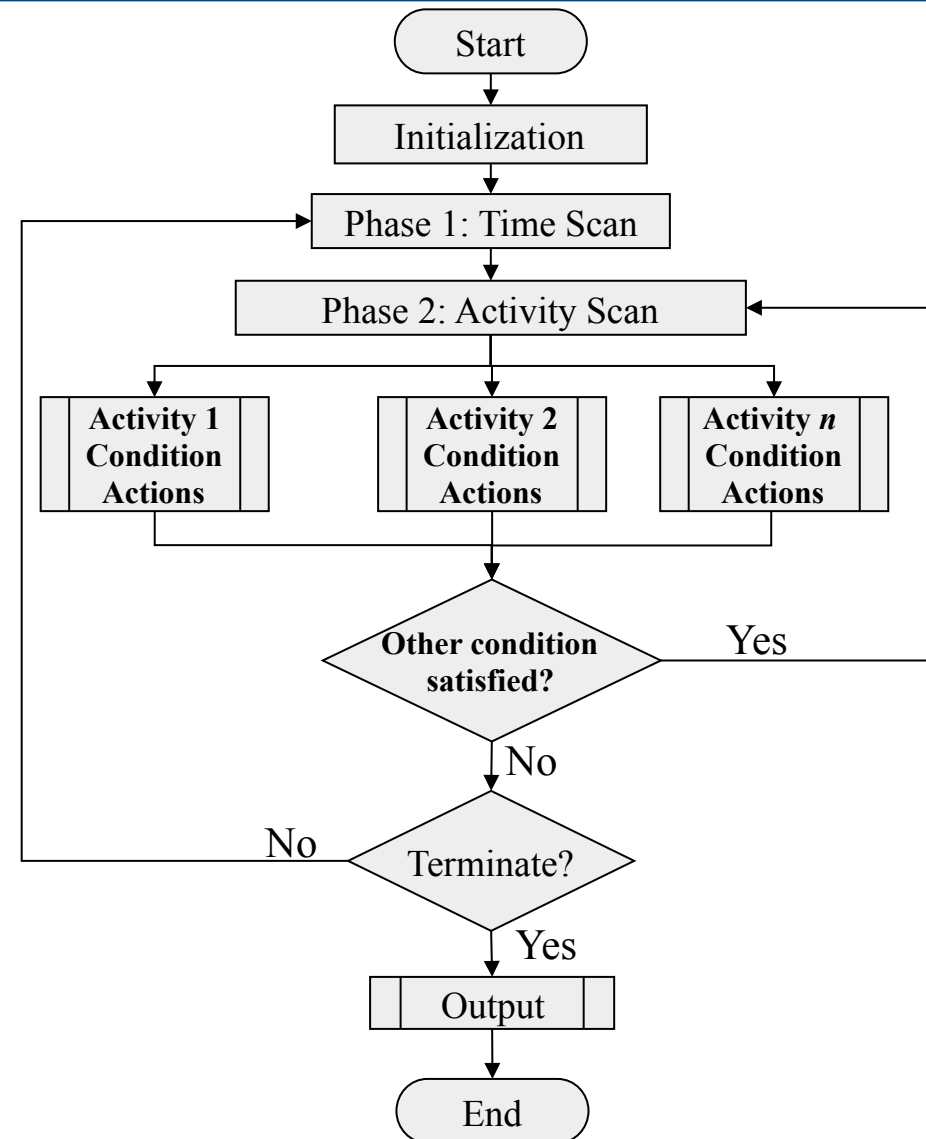
World Views

- Process-interaction
 - Modeler thinks in terms of processes
 - A process is the lifecycle of one entity, which consists of various events and activities
 - Simulation model is defined in terms of entities or objects and their life cycle as they flow through the system, demanding resources and queueing to wait for resources
 - Some activities might require the use of one or more resources whose capacities are limited
 - Processes interact, e.g., one process has to wait in a queue because the resource it needs is busy with another process
 - A process is a time-sequenced list of events, activities and delays, including demands for resources, that define the life cycle of one entity as it moves through a system
 - Variable time advance



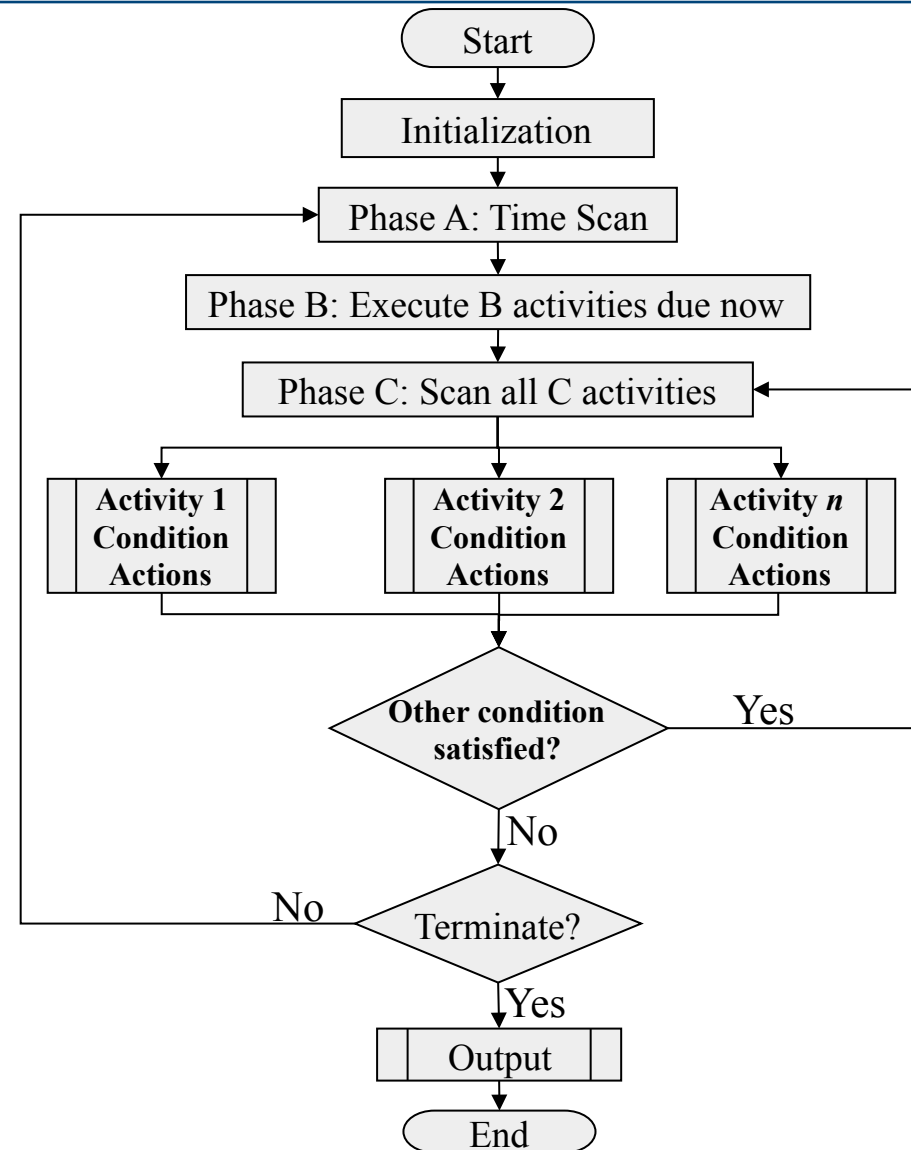
World Views

- Activity-scanning
 - Modeler concentrates on activities of a model and those conditions that allow an activity to begin
 - At each clock advance, the conditions for each activity are checked, and, if the conditions are true, then the corresponding activity begins
 - **Fix time advance**
 - Disadvantage: The repeated scanning to discover whether an activity can begin results in slow runtime
- ➡ Improvement: Three-phase approach
 - Combination of event scheduling with activity scanning

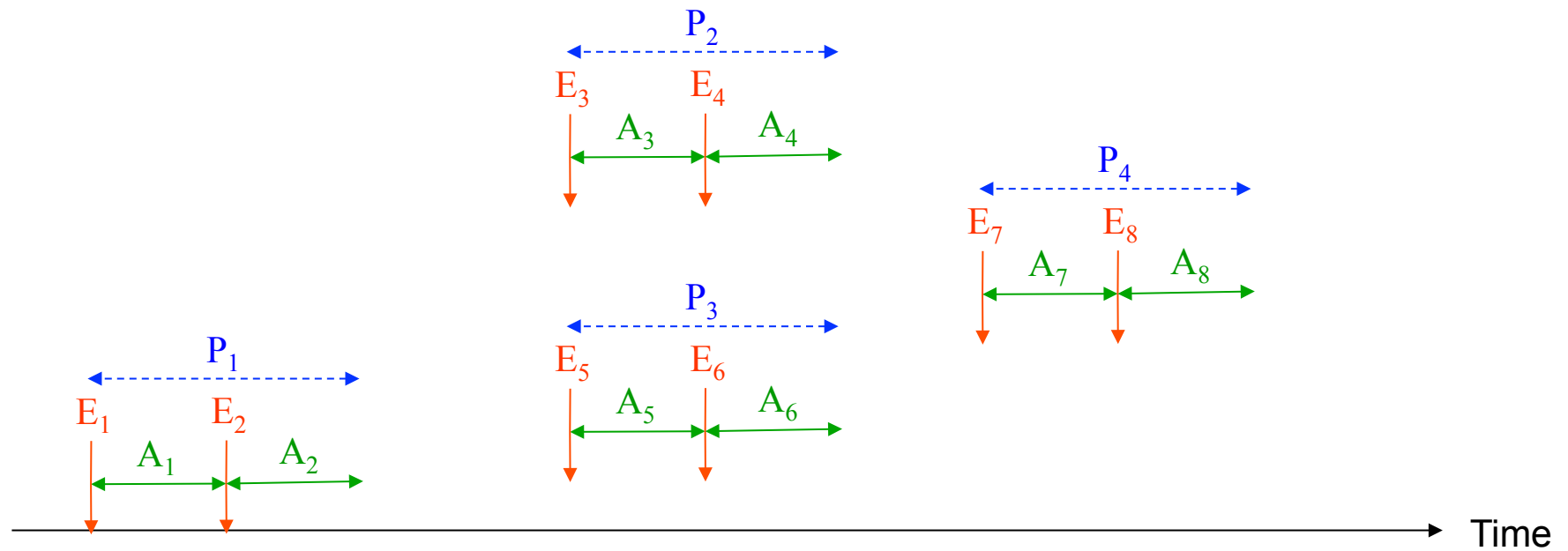


World Views

- Three-phase approach
 - Events are activities of duration zero time units
 - Two types of activities
 - **B activities:** activities bound to occur; all primary events and unconditional activities
 - **C activities:** activities or events that are conditional upon certain conditions being true
 - The B-type activities can be scheduled ahead of time, just as in the event-scheduling approach
 - Variable time advance
 - FEL contains only B-type events
 - Scanning to learn whether any C-type activities can begin or C-type events occur happen only at the end of each time advance, after all B-type events have completed



World Views

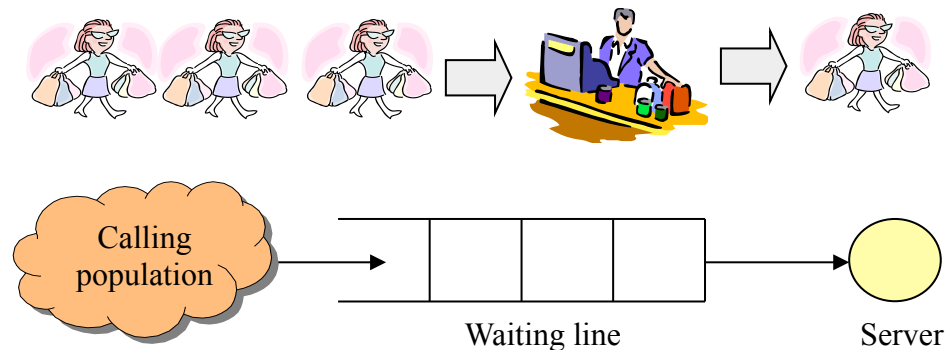


Examples

Example 1

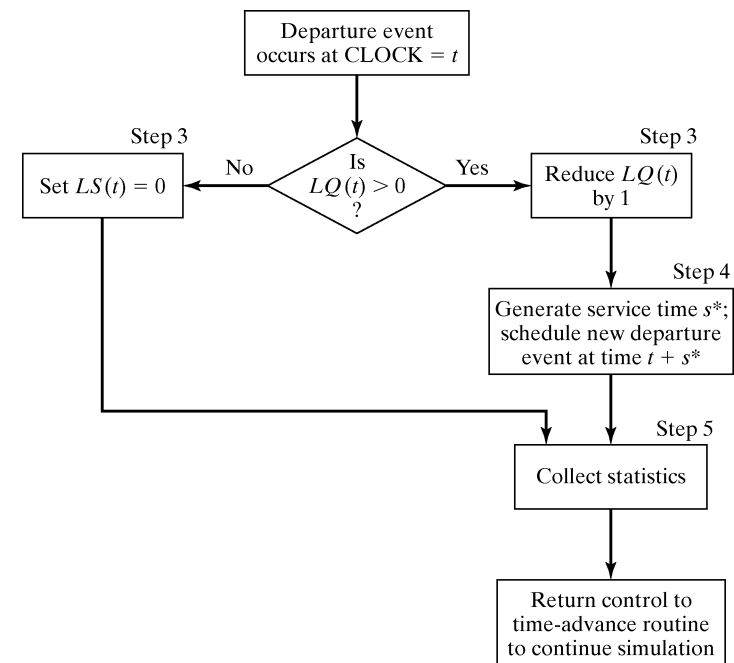
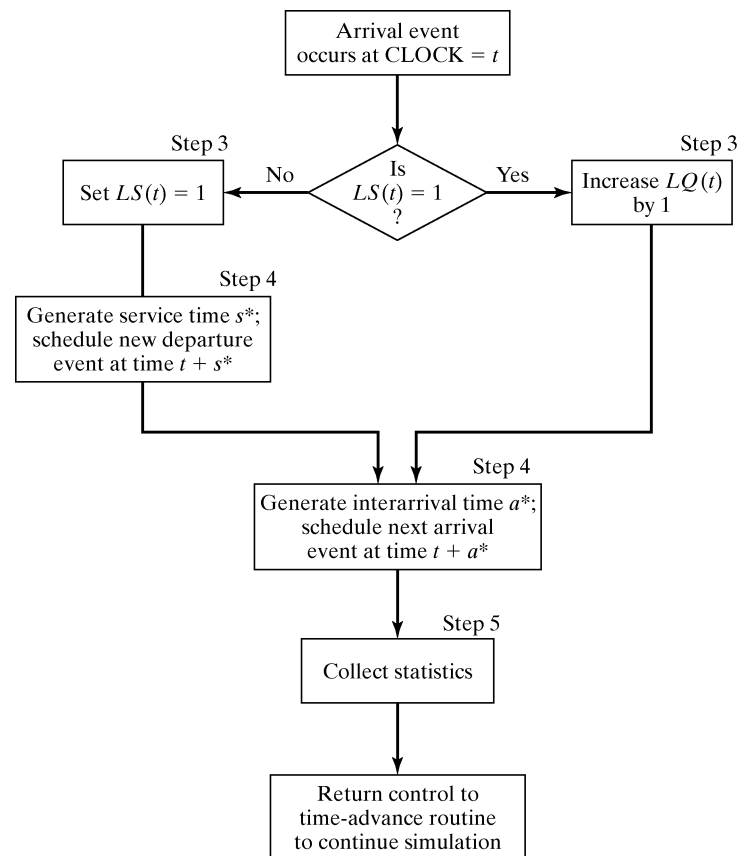
Manual Simulation Using Event Scheduling: Grocery

- Reconsider the grocery example from Chapter 2
 - In Chapter 2: We used an ad hoc method to simulate the grocery
- System state at time t is given by $[LQ(t), LS(t)]$
 - $LQ(t)$ = Number of customers in the waiting line at t
 - $LS(t)$ = Number of customers being served at t (0 or 1)
- Entities
 - Server and customers are not explicitly modeled
- Events
 - Arrival (A)
 - Departure (D)
 - Stopping event (E)
- Event notices
 - (A, t) arrival event at future time t
 - (D, t) departure event at future time t
 - (E, t) simulation stop at future time t
- Activities
 - Interarrival time
 - Service time
- Delay
 - Time customer spent in waiting line



Manual Simulation Using Event Scheduling: Grocery

- System state = [$LQ(t)$, $LS(t)$] is affected by the events
 - Arrival
 - Departure



Manual Simulation Using Event Scheduling: Grocery

| Clock | LQ(t) | LS(t) | FEL | Comment | B | MQ |
|-------|-------|-------|--------------------|--|----|----|
| 0 | 0 | 1 | (A,1)(D,4)(E,60) | First A occurs (a*=1) Schedule next A (s*=4) Schedule first D | 0 | 0 |
| 1 | 1 | 1 | (A,2)(D,4)(E,60) | Second A occurs: (A,1) (a*=1) Schedule next A (Customer delayed) | 1 | 1 |
| 2 | 2 | 1 | (D,4)(A,8)(E,60) | Third A occurs: (A,2) (a*=6) Schedule next A (Two customers delayed) | 2 | 2 |
| 4 | 1 | 1 | (D,6)(A,8)(E,60) | First D occurs: (D,4) (s* = 2) Schedule next D (Customer delayed) | 4 | 2 |
| 6 | 0 | 1 | (A,8)(D,11)(E,60) | Second D occurs: (D,6) (s* = 5) Schedule next D | 6 | 2 |
| 8 | 1 | 1 | (D,11)(A,11)(E,60) | Fourth A occurs: (A,8) (a* = 3) Schedule next A (Customer delay) | 8 | 2 |
| 11 | 1 | 1 | (D,15)(A,18)(E,60) | Fifth A occurs: (A,11) (a* = 7) Schedule next A Third D occurs: (D,11) (s* = 4) Schedule next D (Customer delay) | 11 | 2 |
| 15 | 0 | 1 | (D,16)(A,18)(E,60) | Fourth D occurs: (D,15) (s* = 1) Schedule next D | 15 | 2 |
| 16 | 0 | 0 | (A,18)(E,60) | Fifth D occurs: (D,16) | 16 | 2 |
| 18 | 0 | 1 | (D,23)(A,23)(E,60) | Sixth A occurs: (A,11) (a* = 7) Schedule next A (s* = 5) Schedule next D | 16 | 2 |
| 23 | 0 | 1 | (A,25)(D,27)(E,60) | Seventh A occurs: (A,23) (a* = 2) Schedule next A Sixth D occurs: (D,23) | 21 | 2 |

Server Busy time

Maximum Queue Length

Initial conditions

- First customer arrives at t=0 and gets service
- An arrival and a departure event is on FEL

Statistics

- Server was busy for 21 of 23 time units
- Maximum queue length was 2

Manual Simulation Using Event Scheduling: Grocery

- When event scheduling is implemented, consider
 - Only one snapshot is kept in the computer memory
 - A new snapshot can be derived **only** from the previous snapshot
 - Past snapshots are ignored for advancing the clock
 - The current snapshot must contain **all information necessary** to continue the simulation!
- In the example we do not have any information about particular customers
- If needed, the model has to be extended

Manual Simulation Using Event Scheduling: Grocery

- Analyst wants estimates per customer basis
 - Mean response time (system time)
 - Mean proportion of customers who spend more than 5 time units
- Extend the model to represent customers explicitly
 - **Entities:** Customer entities denoted as C_1, C_2, C_3, \dots
 - (C_i, t) customer C_i arrived at t
 - **Event notices**
 - (A, t, C_i) arrival of customer C_i at t
 - (D, t, C_j) departure of customer C_j at t
 - **Set**
 - “Checkout Line” set of customers currently at the checkout counter ordered by time of arrival
 - **Statistics** (updated when a departure event occurs)
 - S : sum of customer response times for all customers who have departed by the current time
 - F : total number of customers who spend ≥ 5 time units
 - N_D : number of departures up to the current simulation time

Manual Simulation Using Event Scheduling: Grocery

S = Sum response time
 N_D = # Customers left
 F = # Customer which spent ≥ 5 TU

Extended version of the simulation table from Slide 3.33

| Clock | System State | | Lists | | Statistics | | |
|-------|--------------|-------|--------------------|--------------------------|------------|-------|---|
| | LQ(t) | LS(t) | Checkout Line | Future Event List | S | N_D | F |
| 0 | 0 | 1 | (C1,0) | (A,1,C2) (D,4,C1)(E,60) | 0 | 0 | 0 |
| 1 | 1 | 1 | (C1,0)(C2,1) | (A,2,C3)(D,4,C1)(E,60) | 0 | 0 | 0 |
| 2 | 2 | 1 | (C1,0)(C2,1)(C3,2) | (D,4,C1)(A,8,C4)(E,60) | 0 | 0 | 0 |
| 4 | 1 | 1 | (C2,1)(C3,2) | (D,6,C2)(A,8,C4)(E,60) | 4 | 1 | 0 |
| 6 | 0 | 1 | (C3,2) | (A,8,C4)(D,11,C3)(E,60) | 9 | 2 | 1 |
| 8 | 1 | 1 | (C3,2)(C4,8) | (D,11,C3)(A,11,C5)(E,60) | 9 | 2 | 1 |
| 11 | 1 | 1 | (C4,8)(C5,11) | (D,15,C4)(A,18,C6)(E,60) | 18 | 3 | 2 |
| 15 | 0 | 1 | (C5,11) | (D,16,C4)(A,18,C6)(E,60) | 25 | 4 | 3 |
| 16 | 0 | 0 | | (A,18,C6)(E,60) | 30 | 5 | 4 |
| 18 | 0 | 1 | (C6,18) | (D,23,C6)(A,23,C7)(E,60) | 30 | 5 | 4 |
| 23 | 0 | 1 | (C7,23) | (A,25,C8)(D,27,C7)(E,60) | 35 | 6 | 5 |

$$\overline{\text{response time}} = \frac{S}{N_D} = \frac{35}{6} = 5.83$$

$$N_{\geq 5} = \frac{F}{N_D} = \frac{5}{6} = 0.83$$

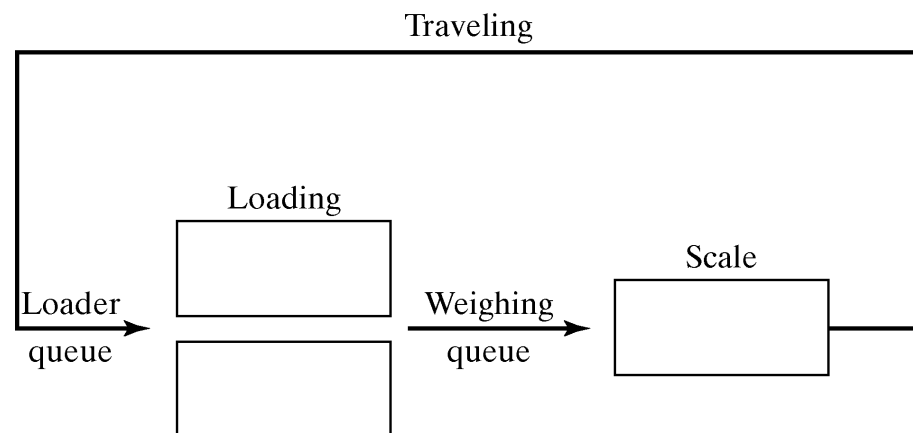
Examples

Example 2

Manual Simulation Using Event Scheduling: Dump Truck

- The DumpTruck Problem

- Six dump trucks are used to haul coal from the entrance of a small mine to the railroad
- Each truck is loaded by one of two loaders
- After loading, the truck immediately moves to the scale, to be weighed
- Loader and Scale have a first-come-first-serve (FCFS) queue
- The travel time from loader to scale is negligible
- After being weighed, a truck begins to travel, afterwards unloads the coal and returns to the loader queue
- Purpose of the study: Estimation of the loader and scale utilizations.



Manual Simulation Using Event Scheduling: Dump Truck

- System state [$LQ(t)$, $L(t)$, $WQ(t)$, $W(t)$]
 - $LQ(t)$ = number of trucks in the loader queue $\in \{0,1,2,\dots\}$
 - $L(t)$ = number of trucks being loaded $\in \{0,1,2\}$
 - $WQ(t)$ = number of trucks in weigh queue $\in \{0,1,2,\dots\}$
 - $W(t)$ = number of trucks being weighed $\in \{0,1\}$
- Event notices
 - (ALQ, t , DT i) truck i arrives at loader queue (ALQ) at time t
 - (EL, t , DT i) truck i ends loading (EL) at time t
 - (EW, t , DT i) truck i ends weighing (EW) at time t
- Entities
 - The six dump trucks DT1, DT2, ..., DT6
- Lists
 - Loader queue: Trucks waiting to begin loading, FCFS
 - Weigh queue: Trucks waiting to be weighed, FCFS
- Activities
 - Loading: Loading time
 - Weighing: Weighing time
 - Travel: Travel time
- Delays
 - Delay at loader queue
 - Delay at scale

Loading Time Distribution

| Loading Time | PDF | CDF |
|--------------|------|------|
| 5 | 0.30 | 0.30 |
| 10 | 0.50 | 0.80 |
| 15 | 0.20 | 1.00 |

Weighing Time Distribution

| Weighing Time | PDF | CDF |
|---------------|------|------|
| 12 | 0.70 | 0.70 |
| 16 | 0.30 | 1.00 |

Travel Time Distribution

| Travel Time | PDF | CDF |
|-------------|------|------|
| 40 | 0.40 | 0.40 |
| 60 | 0.30 | 0.70 |
| 80 | 0.20 | 0.90 |
| 100 | 0.10 | 1.00 |

Manual Simulation Using Event Scheduling: Dump Truck

- Initialization
 - It is assumed that five trucks are at the loader and one is at the scale at time 0
- Activity times
 - Loading time: 10, 5, 5, 10, 15, 10, 10
 - Weighing time: 12, 12, 12, 16, 12, 16
 - Travel time: 60, 100, 40, 40 80
- Statistics
 - BL: Total busy time of both loaders
 - BS: Total busy time of the scale

Both loaders
are busy!

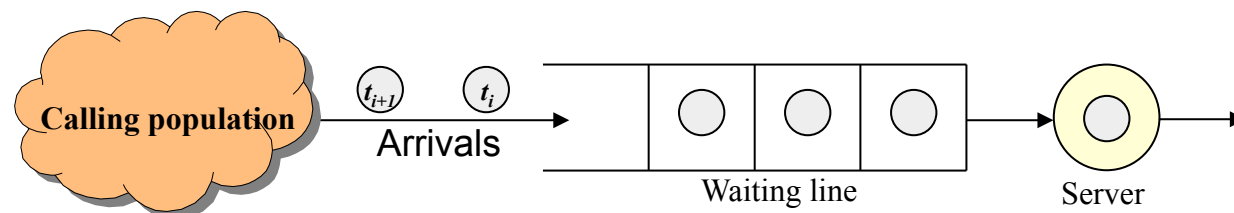
| Clock | System State | | | | Lists | | | Statistics | |
|-------|--------------|------|-------|------|---------------|---------------|---|------------|----|
| | LQ(t) | L(t) | WQ(t) | W(t) | Loader Queue | Weigh Queue | Future Event List | BL | BS |
| 0 | 3 | 2 | 0 | 1 | DT4, DT5, DT6 | | (EL,5,DT3) (EL,10,DT2) (EW,12,DT1) | 0 | 0 |
| 5 | 2 | 2 | 1 | 1 | DT5, DT6 | DT3 | (EL,10,DT2) (EL,5+5,DT4) (EW,12,DT1) | 10 | 5 |
| 10 | 1 | 2 | 2 | 1 | DT6 | DT3, DT2 | (EL,10,DT4) (EW,12,DT1) (EL, 10+10,DT5) | 20 | 10 |
| 10 | 0 | 2 | 3 | 1 | | DT3, DT2, DT4 | (EW,12,DT1) (EL,20,DT5) (EL, 10+15,DT6) | 20 | 10 |
| 12 | 0 | 2 | 2 | 1 | | DT2, DT4 | (EL,20,DT5) (EW,12+12,DT3) (EL, 25,DT6) (ALQ,12+60,DT1) | 24 | 12 |
| 20 | 0 | 1 | 3 | 1 | | DT2, DT4, DT5 | (EW,24,DT3) (EL,25,DT6) (ALQ,72,DT1) | 40 | 20 |
| 24 | 0 | 1 | 2 | 1 | | DT4, DT5 | (EL,25,DT6) (EW,24+12,DT2) (ALQ, 72,DT1) (ALQ,24+100,DT3) | 44 | 24 |

Implementation of Simulations

Simulation in Java

Simulation in Java

- Java is a general purpose programming language
 - Object-oriented
- First simple specific simulation implementation
- Later, object-oriented framework for discrete event simulation
- Again the grocery example
 - Single server queue
 - Run for 1000 customers
 - Interarrival times are exponentially distributed with mean 4.5
 - Service times are also exponentially distributed with mean 3.2
 - Known as: M/M/1 queueing system

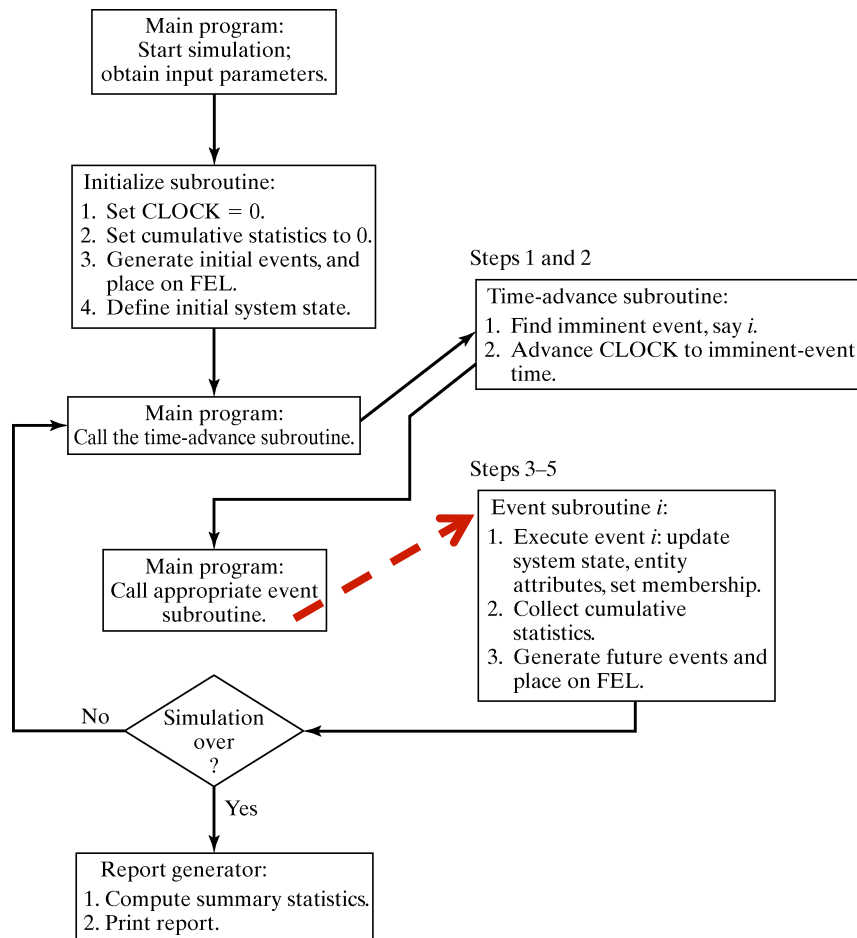


Simulation in Java

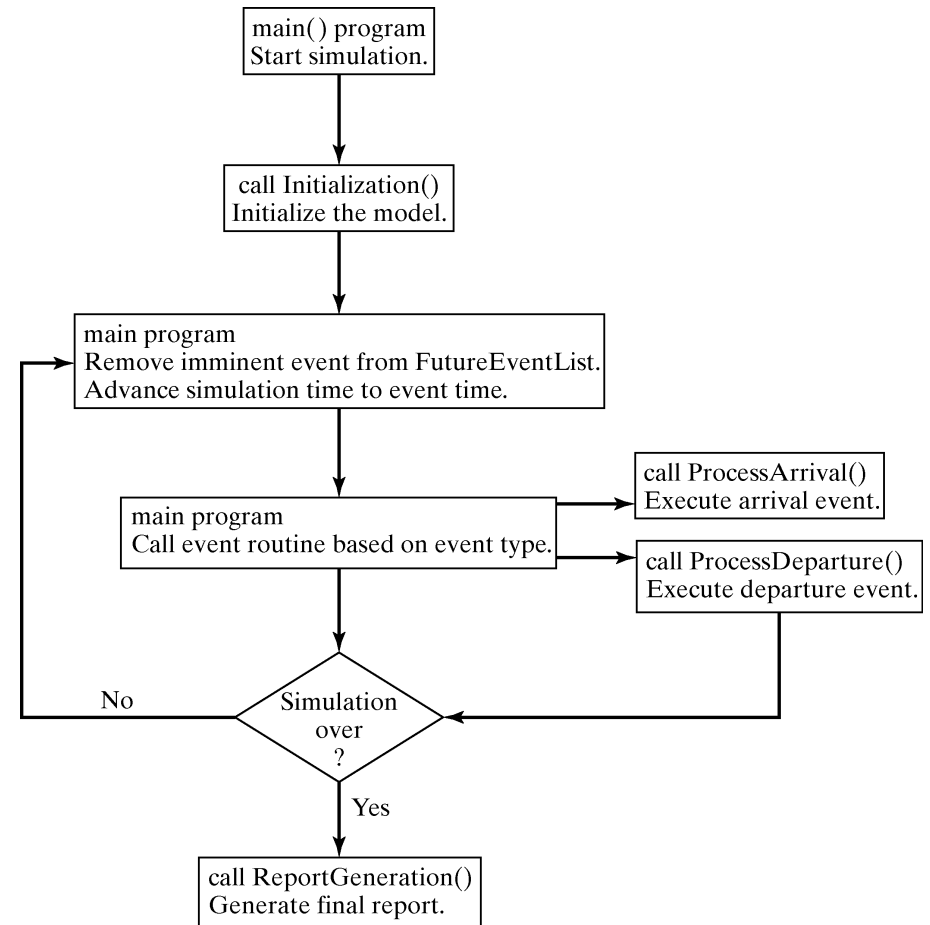
- System state
 - queueLength
 - numberInService
- Entity attributes
 - customers
- Future event list
 - futureEventList
- Activity durations
 - meanInterArrivalTime
 - meanServiceTime
- Input parameters
 - meanInterarrivalTime
 - meanServiceTime
 - totalCustomers
- Simulation variables
 - clock
 - lastEventTime
 - totalBusy
 - maxQueueLength
 - sumResponseTime
- Statistics
 - $\rho = \text{BusyTime/Clock}$
 - avgr = Average response time
 - pc4 = Number of customers who spent more than 4 minutes
- Help functions
 - exponential(mu)
- Methods
 - initialization
 - processArrival
 - processDeparture
 - reportGeneration

Simulation in Java

Overall structure of an event-scheduling simulation program



Overall structure of the Java program



Simulation in Java: Class Event

A very simple realization of an event

- Only two attributes: type and time

```
class Event {  
    public double time;  
    private int type;  
  
    public Event(int _type, double _time) {  
        type = _type;  
        time = _time;  
    }  
  
    public int getType() {  
        return type;  
    }  
  
    public double getTime() {  
        return time;  
    }  
}
```

Simulation in Java: Sim Class

```
class Sim {
    // Class Sim variables
    public static double clock,
                        meanInterArrivalTime,
                        meanServiceTime,
                        lastEventTime,
                        totalBusy,
                        maxQueueLength,
                        sumResponseTime;

    public static long  numberOfCustomers,
                        queueLength,
                        numberInService,
                        totalCustomers,
                        numberOfDepartures,
                        longService;

    public final static int arrival = 1;           // Event type for an arrival
    public final static int departure = 2;         // Event type for a departure

    public static EventList futureEventList;       // We use these data structures as
    public static Queue customers;                // given by the system.
    public static Random stream;                  // A random number generator
}
```

Simulation in Java: Main program

```
public static void main(String argv[]) {
    meanInterArrivalTime = 4.5;
    meanServiceTime      = 3.2;
    totalCustomers        = 1000;
    long seed              = Long.parseLong(argv[0]);

    stream = new Random(seed);           // Initialize RNG stream
    futureEventList = new EventList();   // Initialize the FEL
    customers = new Queue();             // Set which holds the customers

    initialization();                   // Initialize the simulation

    // Loop until first "totalCustomers" have departed
    while( numberOfDepartures < totalCustomers ) {
        Event event = (Event)futureEventList.getMin(); // Get imminent event
        futureEventList.dequeue();                     // Be rid of it
        clock = event.getTime();                       // Advance simulation time
        if( event.getType() == arrival ) {
            processArrival(event);                     // Call event function
        }
        else {
            processDeparture(event);                   // Call event function
        }
    }

    reportGeneration();                 // Print statistics
}
```

Simulation in Java: Initialization

```
// Seed the event list with TotalCustomers arrivals
public static void initialization() {
    clock = 0.0;
    queueLength = 0;
    numberInService = 0;
    lastEventTime = 0.0;
    totalBusy = 0 ;
    maxQueueLength = 0;
    sumResponseTime = 0;
    numberOfDepartures = 0;
    longService = 0;

    // Create first arrival event
    double eventTime = exponential(stream, meanInterArrivalTime);
    Event event = new Event(arrival, eventTime);
    futureEventList.enqueue(event);
}
```


Simulation in Java: Event Arrival

```
public static void processArrival(Event event) {
    customers.enqueue(event);
    queueLength++;
    // If the server is idle, fetch the event, do statistics and put into service
    if( numberInService == 0 ) {
        scheduleDeparture();
    }
    else {
        totalBusy += (clock - lastEventTime); // server is busy
    }

    // Adjust max queue length statistics
    if(maxQueueLength < queueLength) {
        maxQueueLength = queueLength;
    }

    // Schedule the next arrival
    Double eventTime = clock + exponential(stream, meanInterArrivalTime);
    Event nextArrival = new Event(arrival, eventTime);
    futureEventList.enqueue( nextArrival );
    lastEventTime = clock;
}
```

Simulation in Java: Event Departure

```
public static void scheduleDeparture() {
    double serviceTime = exponential(stream, meanServiceTime);
    Event depart = new Event(departure, clock + serviceTime);
    futureEventList.enqueue(depart);
    numberInService = 1;
    queueLength--;
}

public static void processDeparture(Event e) {
    // Get the customer description
    Event finished = (Event) customers.dequeue();
    // If there are customers in the queue then schedule the departure of the next one
    if( queueLength > 0 ) {
        scheduleDeparture();
    }
    else {
        numberInService = 0;
    }
    // Measure the response time and update cumulative statistics
    double response = clock - finished.getTime();
    sumResponseTime += response;
    if( response > 4.0 )
        longService++; // record long service
    totalBusy += (clock - lastEventTime);
    numberOfDepartures++;
    lastEventTime = clock;
}
```

Simulation in Java: Report Generator

```
public static void reportGeneration() {
    double rho    = totalBusy/clock;
    double avgr   = sumResponseTime/totalCustomers;
    double pc4    = ((double)longService)/totalCustomers;

    System.out.println("SINGLE SERVER QUEUE SIMULATION - GROCERY STORE CHECKOUT COUNTER ");
    System.out.println("\tMEAN INTERARRIVAL TIME                " + meanInterArrivalTime );
    System.out.println("\tMEAN SERVICE TIME                    " + meanServiceTime );
    System.out.println("\tNUMBER OF CUSTOMERS SERVED            " + totalCustomers );
    System.out.println();
    System.out.println("\tSERVER UTILIZATION                    " + rho );
    System.out.println("\tMAXIMUM LINE LENGTH                  " + maxQueueLength );
    System.out.println("\tAVERAGE RESPONSE TIME                " + avgr + " Time Units");
    System.out.println("\tPROPORTION WHO SPEND FOUR ");
    System.out.println("\tMINUTES OR MORE IN SYSTEM            " + pc4 );
    System.out.println("\tSIMULATION RUNLENGTH                  " + clock + " Time Units");
    System.out.println("\tNUMBER OF DEPARTURES                  " + totalCustomers );
}
```

Simulation in Java: Output

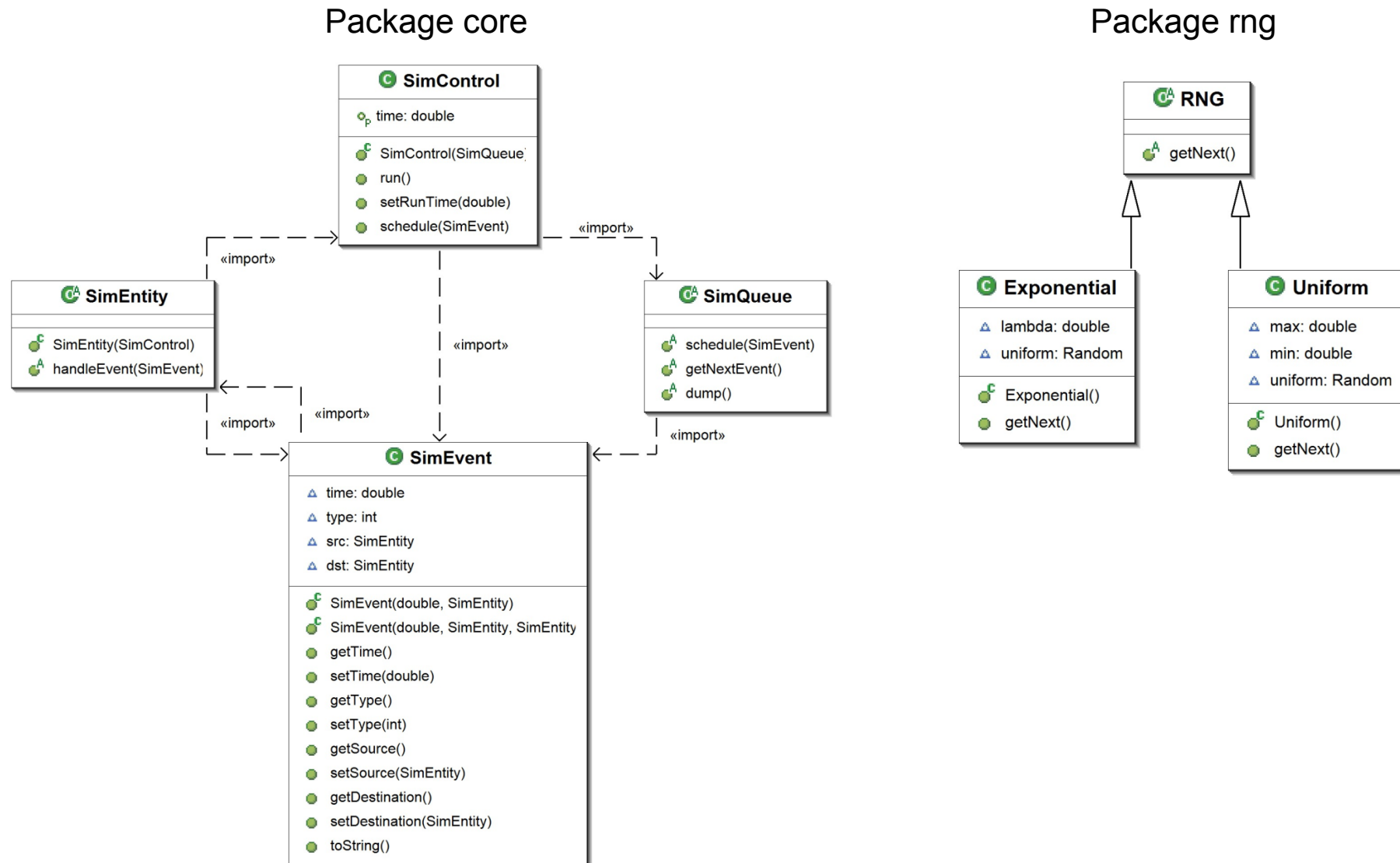
SINGLE SERVER QUEUE SIMULATION - GROCERY STORE CHECKOUT COUNTER

| | |
|--|----------|
| MEAN INTERARRIVAL TIME | 4.5 |
| MEAN SERVICE TIME | 3.2 |
| NUMBER OF CUSTOMERS SERVED | 1000.0 |
| SERVER UTILIZATION | 0.718 |
| MAXIMUM LINE LENGTH | 13.0 |
| AVERAGE RESPONSE TIME | 9.563 |
| PROPORTION WHO SPEND FOUR MINUTES OR MORE IN SYSTEM | 0.713 |
| SIMULATION RUNLENGTH | 4485.635 |
| NUMBER OF DEPARTURES | 1000.0 |

Implementation of Simulations

OO-Discrete-Event Simulation Framework

Object-Oriented Simulation Framework



Object-Oriented Simulation Framework

- OO Discrete-Event Simulation Framework consists of
 - Two packages
- Package core
 - SimEvent
 - SimEntity
 - SimQueue
 - SimControl
- Package rng
 - RNG

Object-Oriented Simulation Framework

```
public class SimEvent {
    double time;
    int type;
    SimEntity src;
    SimEntity dst;
    public long id;

    public SimEvent(SimEntity _dst) {
        type = 0;
        time = 0;
        src = null;
        dst = _dst;
    }

    public SimEvent(double _time, SimEntity _dst) {
        type = 0;
        time = _time;
        src = null;
        dst = _dst;
    }

    public SimEvent(double _time, SimEntity _src, SimEntity _dst) {
        type = 0;
        time = _time;
        src = _src;
        dst = _dst;
    }
}
```


Object-Oriented Simulation Framework

```
public abstract class SimEntity {
    protected SimControl simControl;

    /**
     * An entity has to know the current instance of the simulator.
     * @param _simControl
     * @see SimControl
     */
    public SimEntity(SimControl _simControl) {
        simControl = _simControl;
    }

    /**
     * This method handles the events destined to this entity.
     * @param event
     * @see SimEvent
     */
    abstract public void handleEvent(SimEvent event);
}
```

Object-Oriented Simulation Framework

```
public abstract class SimQueue {  
  
    /**  
     * Schedule the given event according to the event time.  
     * @param event  
     * @see SimEvent  
     */  
    abstract public void schedule(SimEvent event);  
  
    /**  
     * Return the next event in the queue.  
     * @return imminent event in queue.  
     * @see SimEvent  
     */  
    abstract public SimEvent getNextEvent();  
  
    /**  
     * This method dumps the content of the queue.  
     * It is for debugging purposes.  
     */  
    abstract public void dump();  
  
    abstract public boolean isEmpty();  
}
```

Object-Oriented Simulation Framework

```
public class SimControl {
    private SimQueue queue;
    private double time;
    private double endTime;

    public SimControl(SimQueue _queue) {
        queue = _queue;
    }

    public void run() {
        SimEvent event;
        while( queue.isEmpty() == false ) {
            // If there is an event in FEL and the sim-end is not reached ...
            event = queue.getNextEvent();
            time = event.getTime();
            if( event.getTime() <= endTime )
                dispatch(event); // ... call the destination object of this event
            else
                break;
        }
    }

    private void dispatch(SimEvent event) {
        event.getDestination().handleEvent(event);
    }
}
```

Object-Oriented Simulation Framework

```
... public class SimControl ...  
  
    public void setRunTime(double _runTime) {  
        endTime = _runTime;  
    }  
  
    public void schedule(SimEvent event) {  
        queue.schedule(event);  
    }  
  
    public void schedule(SimEvent event, double _delta) {  
        event.setTime(time + _delta);  
        schedule(event);  
    }  
}
```

Object-Oriented Simulation Framework

```
public abstract class RNG {
    abstract public double getNext();
}

public class Exponential extends RNG {
    double lambda;
    Random uniform;

    public Exponential(double _lambda) {
        lambda = _lambda;
        uniform = new Random(System.currentTimeMillis());
    }

    /*
     * @see rng.RNG#getNext()
     */
    public double getNext() {
        return -Math.log(uniform.nextDouble())/lambda;
    }
}
```

Object-Oriented Simulation Framework

- Again our Grocery example
 - Use of the object-oriented simulation framework
- MM1Generator
 - Generates new customer
- MM1Server
 - Serves customer

```
class MM1Generator extends SimEntity {
    MM1Server dst;
    RNG interarrivalTime;

    int noOfCustomer;
    public double sumInterArrivalTime = 0.0;

    /**
     * Constructor
     */
    public MM1Generator(SimControl _simControl, MM1Server dst, RNG _random) {
        super(_simControl);
        noOfCustomer = 0;
        dst = dst;
        interarrivalTime = _random;

        // Schedule the first arrival
        scheduleNextArrival();
    }

    public void handleEvent(SimEvent event) {
        dst.newCustomer(event);
        noOfCustomer++;
        scheduleNextArrival();
    }

    private void scheduleNextArrival() {
        double aInterArrivalTime = interarrivalTime.getNext();
        sumInterArrivalTime += aInterArrivalTime;

        SimEvent event = new SimEvent(this);
        event.id = noOfCustomer;
        simControl.schedule(event, aInterArrivalTime);
    }

    public int getNoOfCustomer() {
        return noOfCustomer;
    }

    public double getLambda () {
        return 1.0/getMeanArrivalTime();
    }

    public double getMeanArrivalTime() {
        return sumInterArrivalTime/(double)noOfCustomer;
    }
}
```

Object-Oriented Simulation Framework

```
class MM1Server extends SimEntity {
    boolean customerInService = false;
    int servicedCustomer = 0;
    double lastServiceStart = 0.0;

    double sumServiceTime = 0.0;
    double sumSystemTime = 0.0;
    double sumQueueTimes = 0.0;

    RNG serviceTime;
    LinkedList customerQueue;

    /**
     * @param _simControl
     */
    public MM1Server(SimControl _simControl, RNG _random) {
        super(_simControl);
        serviceTime = _random;
        customerQueue = new LinkedList();
    }

    public void newCustomer(SimEvent event) {
        customerQueue.add(event);
        if( customerInService == false ) {
            scheduleNextDeparture();
        }
    }

    /**
     * A departure event
     */
    public void handleEvent(SimEvent event) {
        SimEvent finished = (SimEvent)customerQueue.removeFirst();

        double sysTime = simControl.getTime() - finished.getTime();
        double queueTime = lastServiceStart - finished.getTime();

        customerInService = false;
        servicedCustomer++;
        sumSystemTime += sysTime;
        sumQueueTimes += queueTime;

        if( customerQueue.size() > 0 ) {
            scheduleNextDeparture();
        }
    }
}
```

```
private void scheduleNextDeparture() {
    // The next departure will be in aServiceTime time units from now
    // That means the service starts now
    double aServiceTime = serviceTime.getNext();
    sumServiceTime += aServiceTime;
    SimEvent event = new SimEvent(this);
    event.id = servicedCustomer;
    simControl.schedule(event, aServiceTime);
    customerInService = true;
    lastServiceStart = simControl.getTime();
}

public double getMu() {
    return 1.0/getMeanServiceTime();
}
```

Object-Oriented Simulation Framework

```
public class MM1Customer {
    public static void main(String[] args) {
        System.out.println("Start");

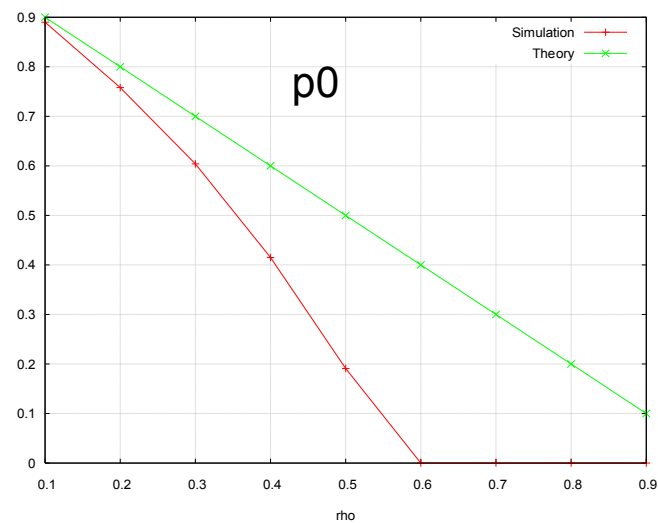
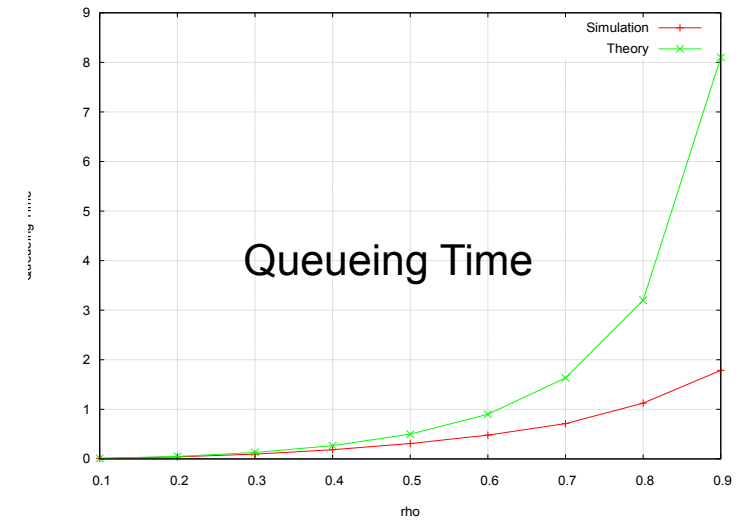
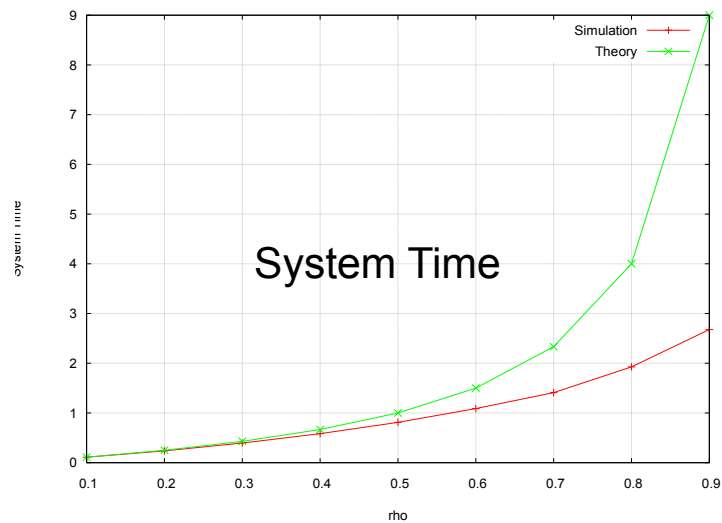
        SimControl simulator = new SimControl(new SimLinkedList());
        simulator.setLogEvents(true);
        simulator.setRunTime(1000);

        MM1Server dst = new MM1Server(new Exponential(0.3125));
        MM1Generator src = new MM1Generator(dst, new Exponential(0.238));

        simulator.run();

        System.out.println("End");
        double lambda = 1.0/src.getLambda();
        double mu = 1.0/dst.getMu();
        double p0 = (double)dst.getServicedCustomer() / simulator.getTime();
        double S = dst.getSystemTimes() / dst.servicedCustomer;
        System.out.println("Time" + simulator.getTime());
        System.out.println("Arrivals" + src.getNoOfCustomer());
        System.out.println("Serviced" + dst.getServicedCustomer());
        System.out.println("lambda" + lambda);
        System.out.println("mu" + mu);
        System.out.println("rho" + lambda/mu);
        System.out.println("p0" + p0);
        System.out.println("S" + S);
    }
}
```


Object-Oriented Simulation Framework



p_0 – Probability that a customer finds the system idle

How to model discrete-event simulations?

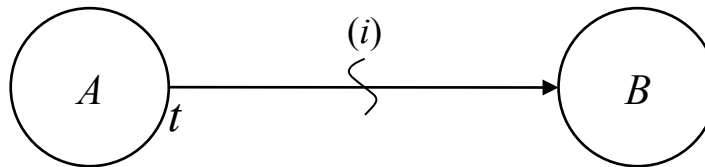
Simulation graphs

Simulation Graphs

- Modeling of Discrete-Event Simulations is difficult
 - There are no standardized modeling methods
 - Modeling is something like an art ;-(
- Simulation Graphs were introduced by Schruben (1983) as Event Graphs and later renamed
- Simulation Graphs are the only graphical paradigm that directly models the Future Event List logic for a discrete-event model

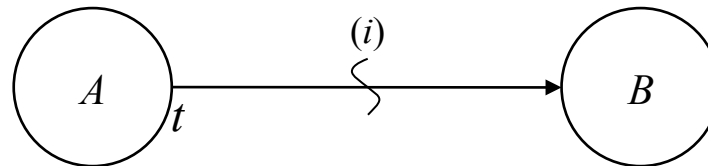
Simulation Graphs

- A Simulation Graph consists of nodes and edges
 - Nodes: Event or state transition
 - Edges: Scheduling of events
 - Edges can have
 - Boolean condition
 - Time delay

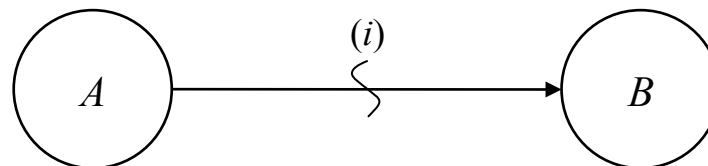


Simulation Graphs

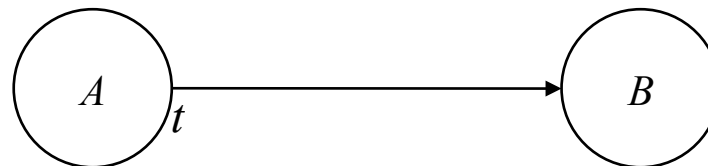
- Basic Simulation Graph
 - Event A causes event B to be scheduled after a time delay t , providing condition (i) is true



- If no time delay exists, t is omitted

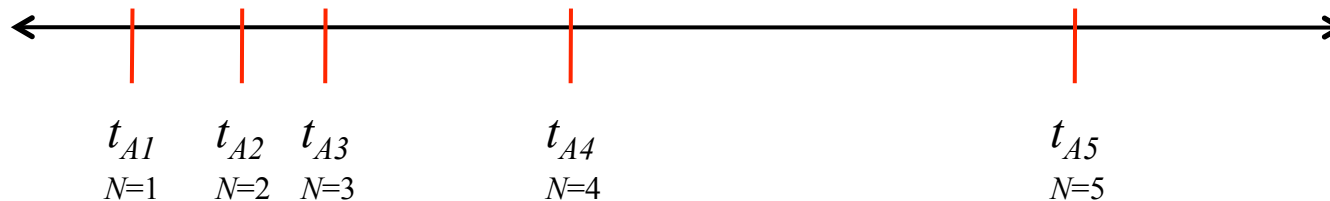
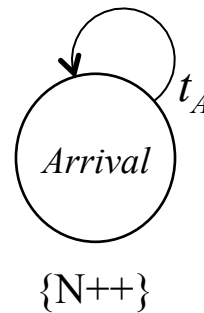


- If event B is always scheduled following the occurrence of event A , condition is omitted ➡ unconditional edge



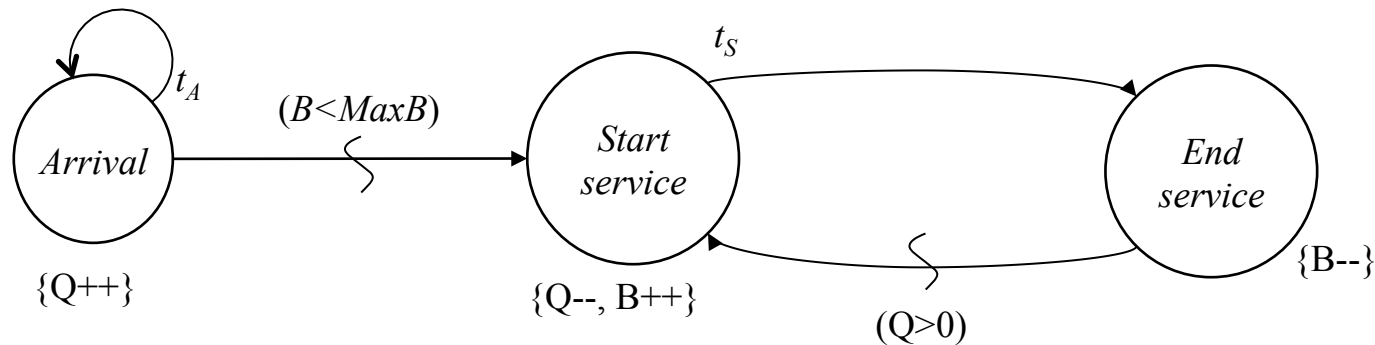
Simulation Graphs: Poisson Process

- A Poisson process counts the number of events in a time interval
 - N denotes the cumulative number of arrivals
 - t_A is exponentially distributed



Simulation Graphs: Simple Queueing

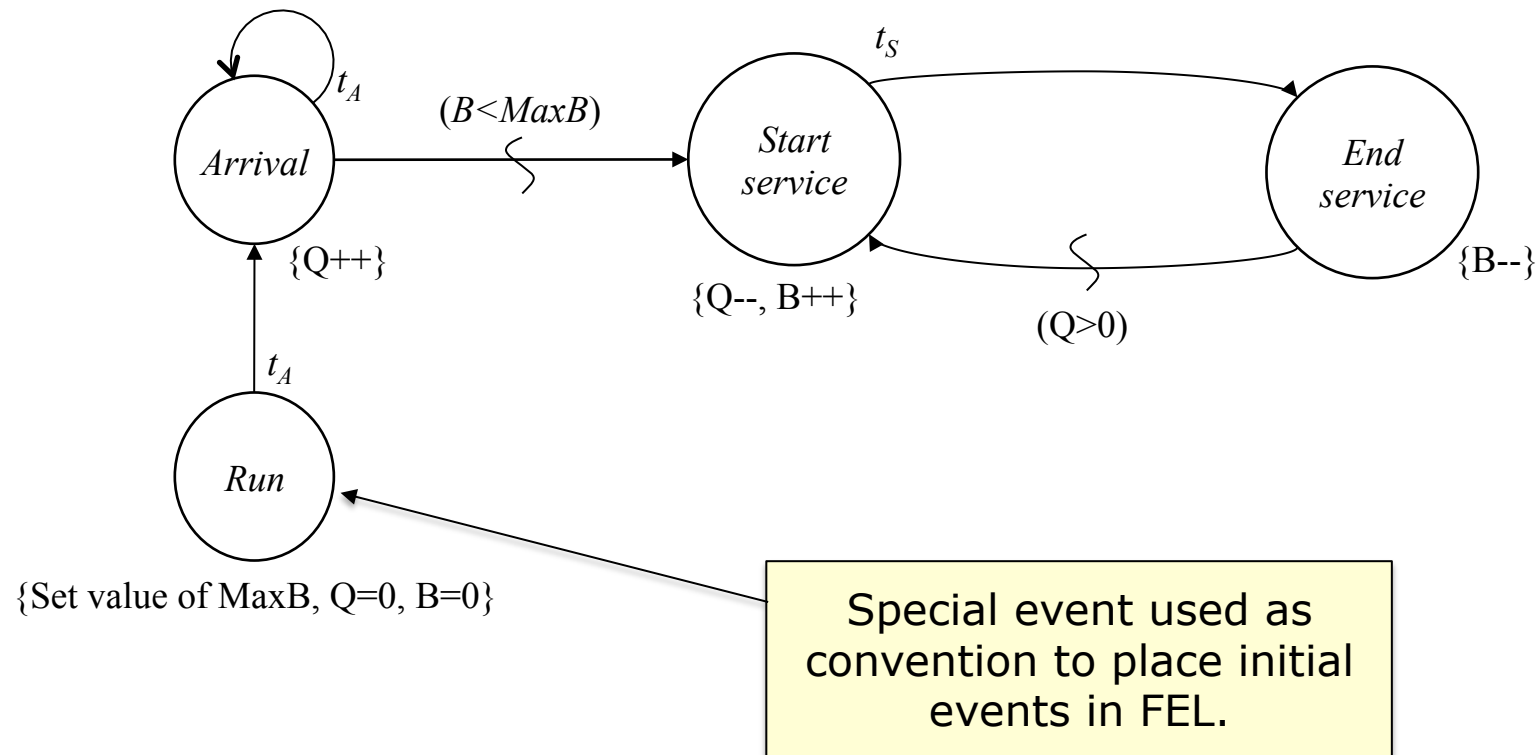
- Simple Queueing Model G/G/MaxB
 - t_A Interarrival time of customer
 - Q Number of customers in queue
 - B Number of busy servers
 - t_S Service time



Simulation Graphs: Simple Queueing

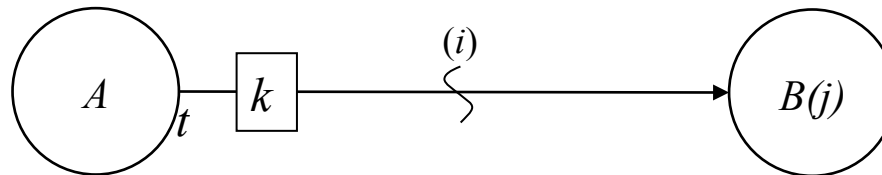
- Example yet not complete
- How to start simulation run? Initialization is open.
- Initialization consists of:
 - Setting of all parameters
 - Setting of initial values of state variables
 - Placing initial event notices
- How to determine initial events?
- With Simulation Graphs
 - Every event that has only incoming or self-scheduling edges must be scheduled at the beginning of the run.
 - As convention a special event is placed in the future event list:
Run

Simulation Graphs: Simple Queueing



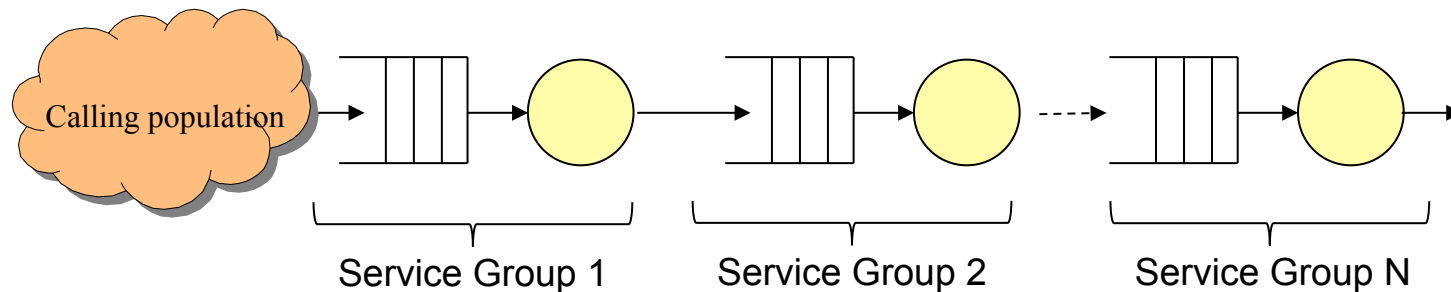
Simulation Graphs: Passing Attributes on Edges

- Enhancement to pass attributes on edges
 - Event A causes event B to be scheduled after a time delay t , providing condition (i) is true. The parameter j is set to k .
 - Passed parameter k can be a list of parameters.



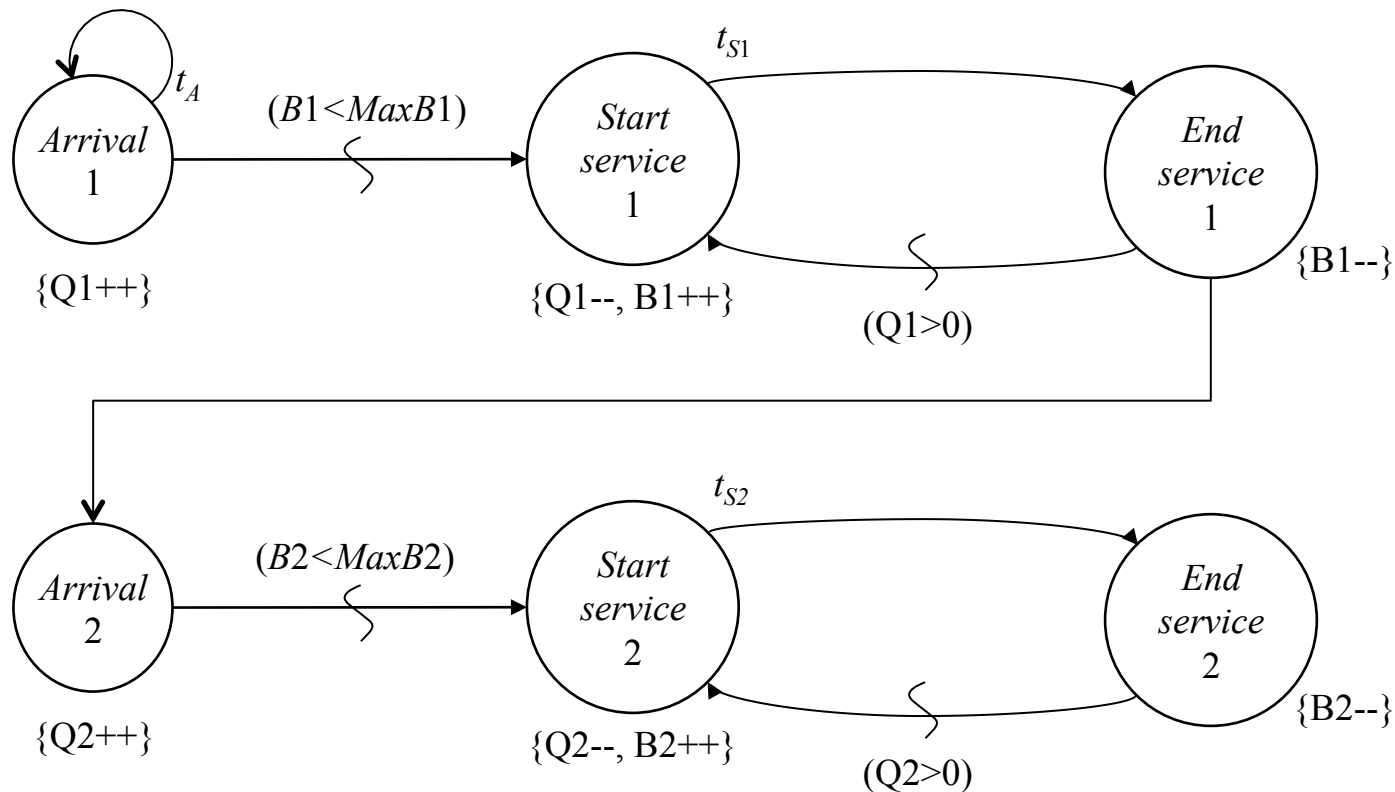
Simulation Graphs: Series of queues

- Serie of queues consisting of N separate queues
 - Production line
 - Chain of devices



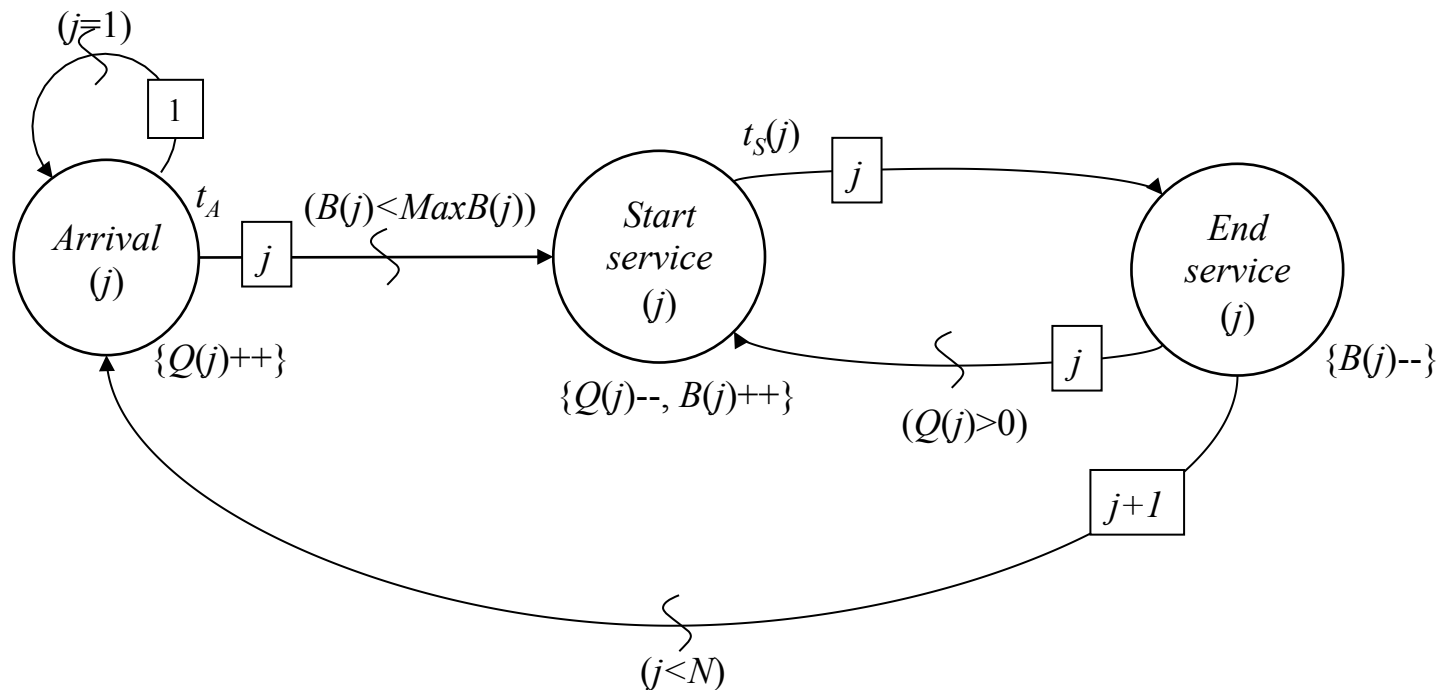
Simulation Graphs: Series of queues

- Approach 1
 - Chain several representations of a queue



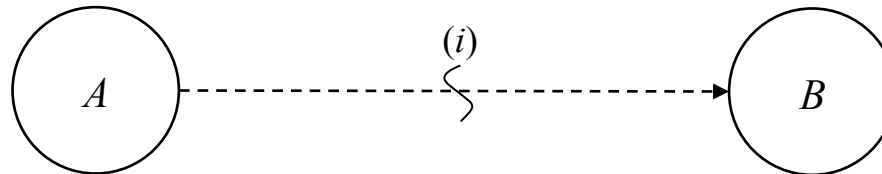
Simulation Graphs: Series of queues

- Approach 2
 - Use edge attributes
 - The simulation code of each service station is the same



Simulation Graphs: Canceling Edges

- Enhancement to cancel event notices from FEL
 - When event A occurs and condition (i) is true, the next occurrence of event B is removed from the event list
 - If no event notice for B is on the event list, nothing happens.



Simulation Graphs: Summary

- Simulation Graphs
 - Simple approach to model discrete-event system
 - Allows identification of required events of simulation system
 - Graphical representation of the relationship of events
 - Enhancements allow construction of more complex models from simple ones
 - Provides optimization of simulation graphs (not discussed)
- Disadvantage:
 - Focus is on events, modeler has to abstract from entities

Summary

- Introduced a general framework for discrete-event simulations
- Event-scheduling and time-advance algorithm
- Generation of events
- World views for discrete simulations
- Introduced manual discrete event simulation
- Introduced simulation in Java
- Object-oriented simulation framework in Java
- Introduced Simulation Graphs as a tool to model discrete-event simulations