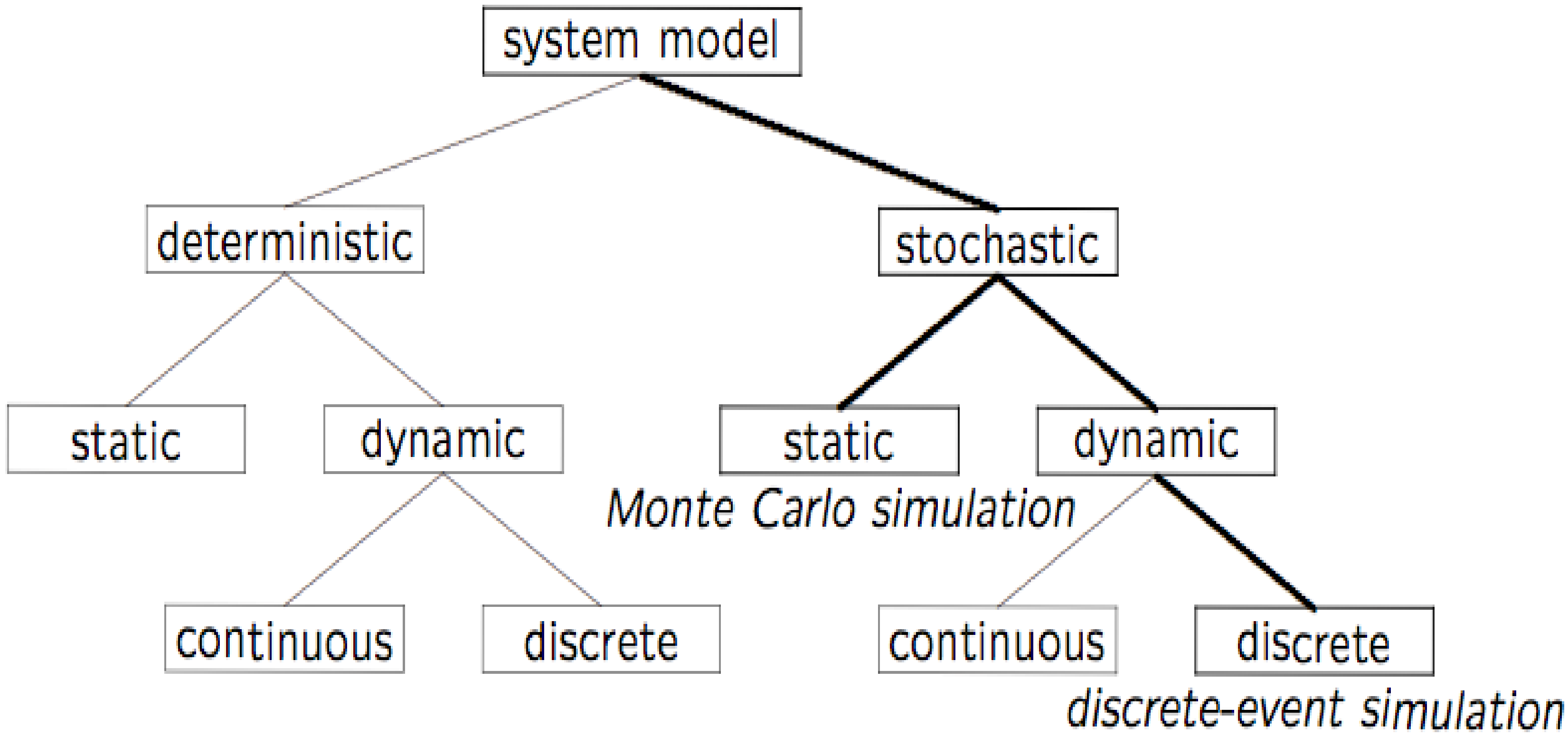
System Simulation
(CNG-476)

Discrete-Event Simulation

Instructor: **Asst. Prof. Muhammad Toaha Raza Khan**
Computer Engineering Department
Middle East Technical University,
Northern Cyprus Campus

Recap: Simulation Model Taxonomy



Recap: DES Model Development

- How to develop a simulation model:
    1. Determine the goals and objectives
    2. Build a ***conceptual*** model
    3. Convert into a ***specification*** model
    4. Convert into a ***computational*** model
    5. Verify the model
    6. Validate the model
- Typically an iterative process

Overview of DES Module

- Develops a common framework (and terminology) for the modeling of complex systems

- Covers the basic building blocks for all discrete-event simulation models

- Introduces and explains the fundamental concepts and methodologies underlying all discrete-event simulation packages:
  - These concepts and methodologies are not tied to any particular simulation package

Outline

- Concepts in discrete-event simulation
  - Terminology and concepts
  - Two pedagogical examples
- Components of discrete-event simulation
  - Time advance approaches
  - Event scheduling approach
- Manual simulation
  - Grocery store example
- Simulation program
  - Simulation of queuing systems
  - Infinite and finite population model
  - Tandem queue with blocking
- Verification and validation of simulation models

Outline

- Concepts in discrete-event simulation
  - Terminology and concepts
  - Two pedagogical examples
- Components of discrete-event simulation
  - Time advance approaches
  - Event scheduling approach
- Manual simulation
  - Grocery store example
- Simulation program
  - Simulation of queuing systems
  - Infinite and finite population model
  - Tandem queue with blocking
- Verification and validation of simulation models

Concepts in Discrete-Event Simulation (1 of 2)

- Model: an abstract representation of a (real) system
- System: a collection of entities that interact together over time (e.g., people, machines, CPU, Web server)
- System state: a collection of variables that contain all the information necessary to adequately describe the system at any time (e.g., occupancy)
- Entity: any object or component in the system (e.g., a server, a customer, a machine)
- Attributes: the properties of a given entity
- List: a collection of associated entities, ordered in some logical fashion (e.g., sets, queues)

Concepts in Discrete-Event Simulation (2 of 2)

- Event: an instantaneous occurrence that changes the state of a system (e.g., an arrival of a new customer)
- Event list: a list of event notices for future events, ordered by time of occurrence, also called the future event list (FEL)
- Activity (unconditional wait): a duration of time of specified length that is known when it begins (e.g., a service time)
- Delay (conditional wait): a duration of time of unspecified indefinite length, which is not known until it ends (e.g., customer delay while waiting in line)
- Clock: a variable representing simulated time, which can be either continuous or discrete

Note: different commercial simulation packages use different terminology for the same or similar concepts

Key Concepts in Discrete-Event Simulation

- An activity represents a service time, an inter-arrival time, or any processing time whose duration has been defined or characterized by the modeler:
  - An activity's duration may be specified as:
    - Deterministic or stochastic
    - A function depending on system variables and/or entity attributes
  - Duration is not affected by the occurrence of other events

- A delay's duration is determined by current system conditions (not specified by the modeller ahead of time):
  - For example, a customer's delay in a waiting line may be dependent on the number and duration of service of other customers ahead in line, and whether a server has a failure (and repair time) or not

Example 1: ABC Call Center

A computer technical support center with personnel taking calls and providing service:

- Three support staff: Alice, Bob, Chris (multiple support channel)
- A simplifying rule: alphabetical tie-breaker if > 1 staff are idle

- Goal: to find out how well the current arrangement works in terms of the response time of the system

- Random variables:
  - Arrival time between calls
  - Service time (different distributions for Alice, Bob, and Chris)

States in ABC Call Center Example

The ABC Call Center System is a discrete-event model with the following components:

- System state:
  - The number of callers waiting to be served at time t
  - Indicator that Alice is idle or busy at time t
  - Indicator that Bob is idle or busy at time t
  - Indicator that Chris is idle or busy at time t

- Entities: neither the caller nor the servers need to be explicitly represented, except in terms of the state variables, unless certain per-caller or per-server statistics are desired

Events in ABC Call Center Example

- Events:
  - Arrival of a call
  - Service completion by Alice
  - Service completion by Bob
  - Service completion by Chris
- Activities:
  - Inter-arrival time
  - Service time by Alice
  - Service time by Bob
  - Service time by Chris
- Delay: a caller’s wait in queue until Alice, Bob, or Chris becomes free

Example 2: Pancake Manor

A pancake restaurant in an old church in Brisbane, Australia:

- Host/hostess for seating of customers (possible waiting here)
- Waiter/waitress for ordering/bringing food and beverages
- Kitchen and cook(s) for preparing food (possible queueing too!)
- Cashier for payment and departure

- Goal: to find out how many staff (and tables) to have to keep the response time of the system reasonable

- Random variables:
  - Arrival times of customers
  - Sizes of groups
  - Time of day
  - Service times for ordering, eating, payment, etc.

Example 2: Pancake Manor

States in Pancake Manor Example

The Pancake Manor restaurant is a discrete-event model with the following components:

- System state:
  - The number of customers waiting to be seated at time t
  - The number of customers waiting to order at time t
  - The number of customers waiting for food at time t
  - The number of customers eating at time t
  - The number of customers waiting to pay at time t
  - The number of available/occupied tables at time t
- Entities: customers; host/hostess; waiter/waitress; cooks in kitchen; tables in restaurant; other?

Events in Pancake Manor Example

- Events:
  - Arrival of a customer (or group of customers)
  - Service completion by host/hostess
  - Service completion by waiter/waitress
  - Service completion by cook
  - Service completion by cashier

- Activities:
  - Inter-arrival time
  - Service time by host/hostess
  - Service time by waiter/waitress
  - Service time by cook
  - Service time by cashier

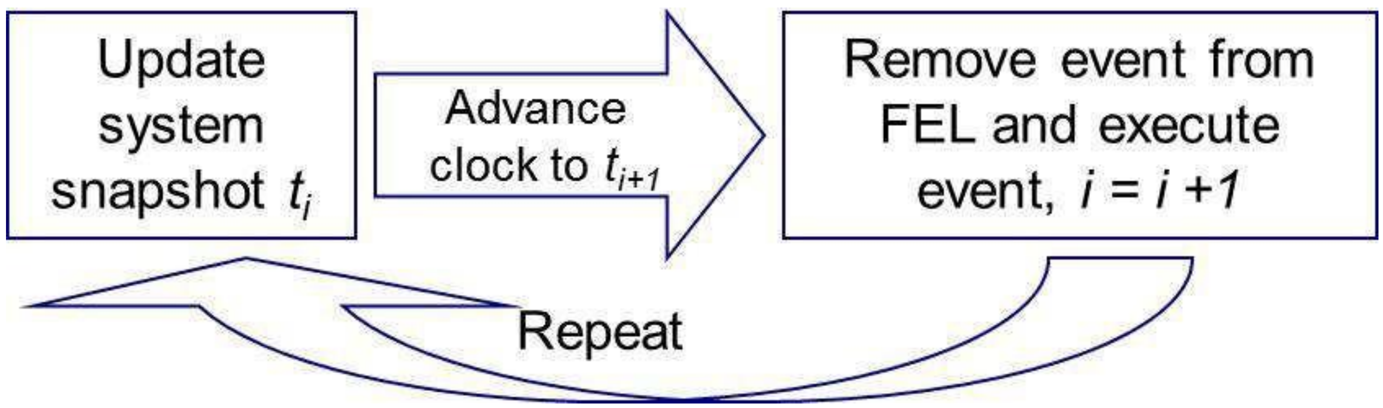
- Delay: a caller's wait for seating, ordering, eating, paying, etc.

Outline

- Concepts in discrete-event simulation
  - Terminology and concepts
  - Two pedagogical examples
- Components of discrete-event simulation
  - Time advance approaches
  - Event scheduling approach
- Manual simulation
  - Grocery store example
- Simulation program
  - Simulation of queuing systems
  - Infinite and finite population model
  - Tandem queue with blocking
- Verification and validation of simulation models

Components of a Simulation

- In DES simulation:
  - The simulation is driven by events
  - The simulation time advances based on sequence of events
  - System state changes with events
- Requirements:
  - Time advance algorithm
  - Event scheduling
  - Event processing



Time Advance Approaches

The mechanism for advancing simulation time and guaranteeing that all events occur in correct chronological order

- General approaches:

1. Time-stepping approach (fixed time increment):
   - Also known as the “activity scanning” approach
   - At each clock advance, the conditions for each activity are checked, and if the conditions are true, then the corresponding activities begin

2. Event-scheduling approach (variable time advance):
   - Concentrates on events and their effect on system state
   - The simulation clock is advanced to the time of the next imminent event on the FEL

Time-Stepping Approach

- At any given time t, the list of all pending future events is scanned to determine which ones are applicable
- FEL not strictly required, nor does it need to be ordered

- Main challenge is getting the time step appropriate
  - Too small: high overhead; lots of scanning; not much happens
  - Too large: too many events applicable at once

- Real systems often have highly-varying times between events
- Time-stepping approach is simple in concept, but often slow in execution (i.e., high overhead)
- Suitable only for simulating small systems with well-defined inherent time steps (e.g., mortgage.c, fluid flow)

Event-Scheduling Approach

- At any given time t, the future event list (FEL) contains all previously scheduled future events and their associated event times $(t_1, t_2, \ldots)$

- FEL is ordered by event time, and the event times satisfy:

  $t \leq t_1 \leq t_2 \leq \cdots \leq t_n$ where t is the value of the Clock.

Event-Scheduling Approach

Old system snapshot at time t

CLOCK	System 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 3 event to occur at t_n	

New system snapshot at time t_1

Step 1 –Remove the event notice for the imminent event (event 3, time t_1) from FEL.

Step 2 –Advance CLOCK to imminent event (i.e., advance CLOCK from t to t_1).

Step 3 –Execute imminent event: update system state, change entity attributes, and set membership as needed.

Step 4 –Generate future events (if necessary) and place their event notices on FEL, ranked by event time.
(Example: Event 4 to occur at time t*, where $t_2 < t^* < t_3$.)

Step 5 –Update cumulative statistics and counters.

New system snapshot at time t1

CLOCK	System 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 3 event to occur at t_n	

List Processing

- The management of a list
  - The major list processing operations performed on a FEL are:
    - Removal of the imminent event
    - Addition of a new event to the list
    - Occasionally removal of some event (cancellation of an event)
  - Efficiency of search within the list depends on the logical organization of the list and how the search is conducted

- Data structure for FEL? Choice depends on system size:
  - Variable(s)
  - Arrays
  - Files
  - Ordered linked list
  - Priority queue
  - Binary heap
  - Calendar queue

Future Events

- Arrival event:
  - For example, at time 0, the first arrival event is generated and is scheduled on the FEL. When the clock eventually is advanced to the time of this first arrival, a second arrival event is generated.
- Service completion event:
  - Triggered only on the condition that a customer is present and a server is free
- Stopping event, E:
  - At time 0: schedule a stop simulation event at a specified future time T_E
  - Run length T_E is determined by the simulation itself. Generally, T_E is the time of occurrence of some specified event E (e.g., completion of 1000^{th} customer) or condition (e.g., relative change in estimate of π, or standard deviation of queue size)

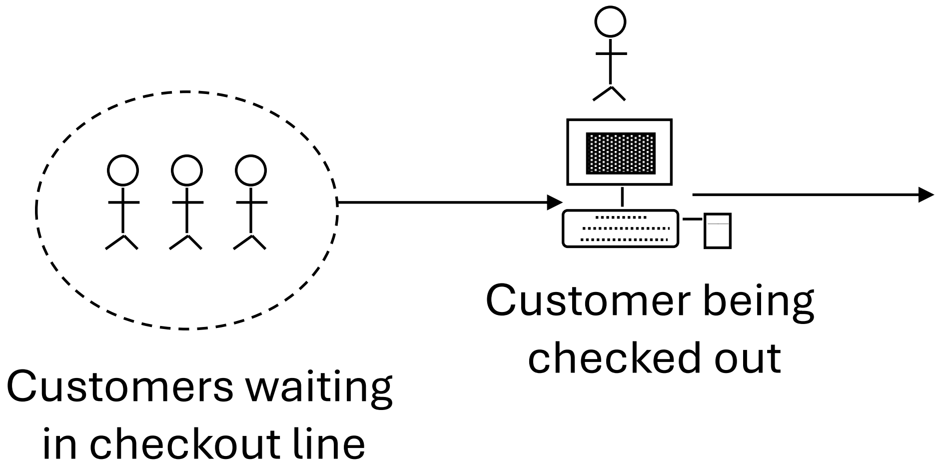
Outline

- Concepts in discrete-event simulation
  - Terminology and concepts
  - Two pedagogical examples
- Components of discrete-event simulation
  - Time advance approaches
  - Event scheduling approach
- Manual simulation
  - Grocery store example
- Simulation program
  - Simulation of queuing systems
  - Infinite and finite population model
  - Tandem queue with blocking
- Verification and validation of simulation models

Example 3: Grocery Store

Grocery Store with single checkout

- Single-channel queue:
  - The system consists of those customers waiting plus the one (if any) checking out
  - For this example, a stopping time of 60 minutes is set

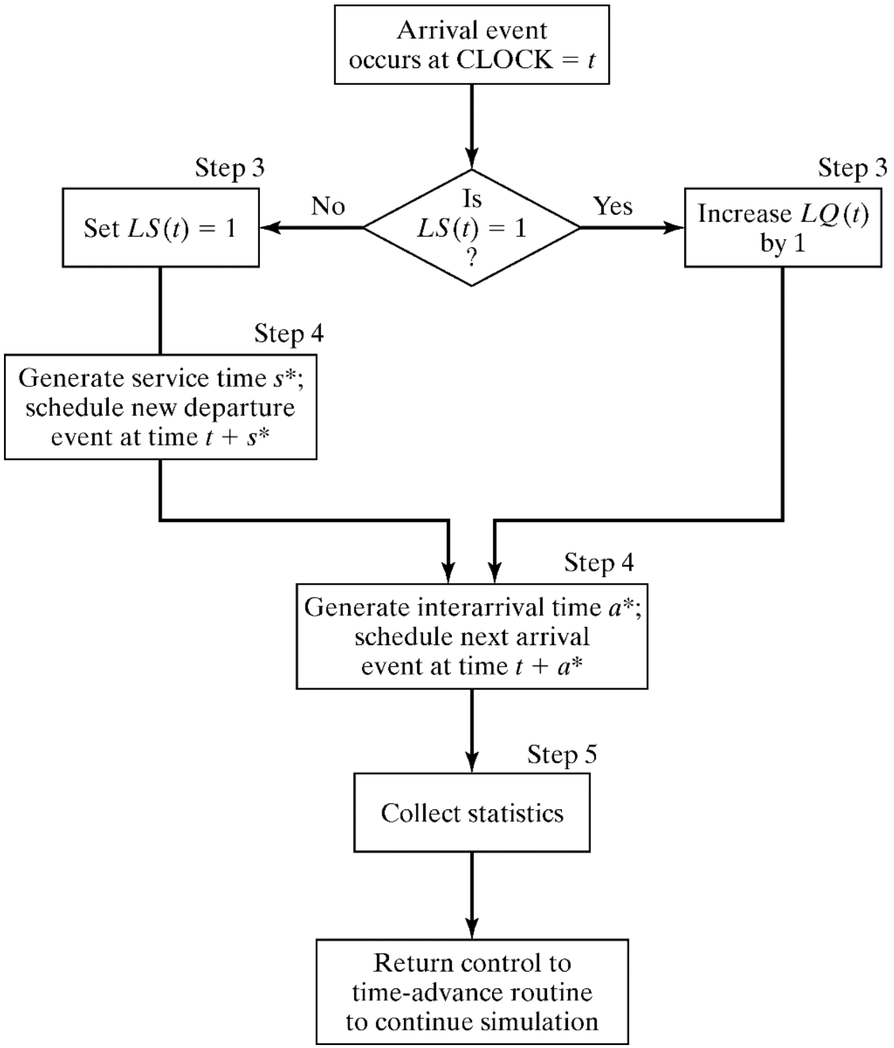


Components of Grocery Store Example

- Model components:
  - System state:
    - $LQ(t)$: # of customers waiting in line at time t (excluding the customer being checked out)
    - $LS(t)$: # of customer being checked out (1 or 0) at time t
  - Entities: the server and customers are not explicitly modeled, except in terms of the state variables
  - Events: arrival (A), departure (D), stopping event (E)
  - Event notices (event type, event time):
    - (A, t) representing an arrival event to occur at future time t
    - (D, t) representing a customer departure at future time t
    - (E, 60) representing the simulation stop event at future time 60
  - Activities: inter-arrival time and service time
  - Delay: customer time spent in waiting line

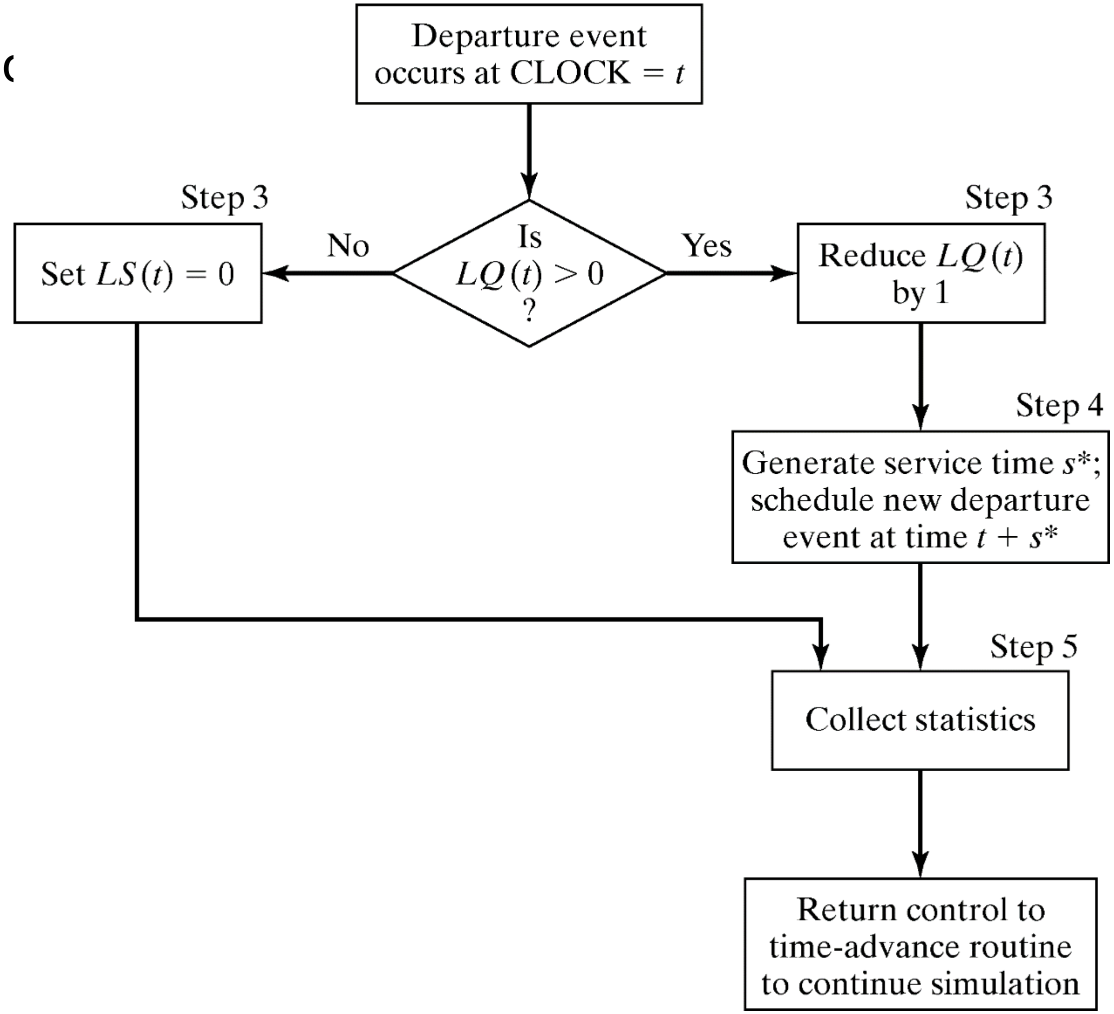
Arrivals in Grocery Store Example

- Event logic:



Departures in Grocery Store Example

- Event logic



Scenario 1 in Grocery Store Example

- Initial conditions are: the first customer arrives at time 0 and begins service
- Only two statistics:
  - B: total server busy time (server utilization $= B/T_E$)
  - $MaxQ$: maximum queue (checkout line) length observed
- Input parameters:

Interarrival Times	0	8	6	1	8
Service Times	4	1	4	3	5

Event Summary for Grocery Store Example

Clock	System State		Future Event List	Comment	Cumulative Statistics	
	LQ(t)	LS(t)			B	MQ
0	0	1	(D, 4), (A, 8), (E, 60)	First A occurs: (a* = 8), schedule next A; (s*=4) Schedule first D	0	0
4	0	0	(A, 8), (E, 60)	First D occurs: (D, 4)	4	0
8	0	1	(D, 9), (A, 14), (E, 60)	Second A occurs: (A, 8); (a* = 6) Schedule next A; (s* = 1) Schedule next D	4	0
9	0	0	(A, 14) (E, 60)	Second D occurs: (D, 9)	5	0
14	0	1	(A, 15) (D, 18) (E, 60)	Third A occurs: (A, 14); (s* = 4) Schdeule next D	5	0
15	1	1	(D, 18), (A, 23), (E, 60)	Fourth A occurs: (A, 15) (Customer delayed)	6	1
18	0	1	(D, 21) (A, 23) (E, 60)	Third D occurs: (D, 18); (s* = 3) schedule next D	9	1

Simulation Table

Manual vs. Computer Simulation

- When an event-scheduling algorithm is computerized, only one snapshot (the current one or partially updated one) is kept in computer memory
  - A new snapshot can be derived only from the previous snapshot, newly generated random variables, and the event logic
  - The current snapshot must contain all information necessary to continue the simulation

Scenario 2 in Grocery Store Example

- Suppose the simulation analyst desires to estimate
  - mean response time, and,
  - mean proportion of customers who spend 4 or more minutes in the system (i.e., waiting in line + checkout time)
- It is necessary to expand the previous model to represent the individual customers explicitly:
  - Customer entity with arrival time as an attribute will be added to the list of components
  - Customer entities will be stored in a list to be called 'Checkout Queue' as C1, C2, C3, ...

Statistics for Grocery Store Example

Collected Statistics:

- Three new cumulative statistics will be collected:
  - S: the sum of customer response times for all customers who have departed by the current time
  - F: the total number of customers who spend 4 or more minutes at the checkout counter
  - N_D: the total number of departures up to the current simulation time

Computing Statistics for Grocery Store Example

Updating Statistics:

- At time 18, when the departure event (D, 18, C3) is being executed, the response time for customer C3 is computed as:

  Response time = clock time - attribute ‘time of arrival’
  = 18 -14 = 4 minutes

- Then S is incremented by 4 minutes, and F and N_D by one customer

Summary Table for Grocery Store Example

- Input parameters:

Interarrival Times	0	8	6	1	8
Service Times	4	1	4	3	5

Clock	System State		Checkout Queue	Future Event List	Statistics		
	$LQ(t)$	$LS(t)$			S	N_D	F
0	0	1	(C1, 0)	(D, 4, C1), (A, 8, C2), (E, 60)	0	0	0
4	0	0		(A, 8, C2), (E, 60)	4	1	1
8	0	1	(C2, 8)	(D, 9, C2), (A, 14, C3), (E, 60)	4	1	1
9	0	0		(A, 14, C3), (E, 60)	5	2	1
14	0	1	(C3, 14)	(A, 15, C4), (D, 18, C3), (E, 60)	5	2	1
15	1	1	(C3, 14), (C4, 15)	(D, 18, C3), (A, 23, C5), (E, 60)	5	2	1
18	0	1	(C4, 15)	(D, 21, C4), (A, 23, C5), (E, 60)	9	3	2

Simulation Table

Outline

- Concepts in discrete-event simulation
  - Terminology and concepts
  - Two pedagogical examples
- Components of discrete-event simulation
  - Time advance approaches
  - Event scheduling approach
- Manual simulation
  - Grocery store example
- Simulation program
  - Simulation of queueing systems
  - Infinite and finite population model
  - Tandem queue with blocking
- Verification and validation of simulation models

Generic Simulation Program (1 of 4)

- Initialization
  - Initialize clock to zero
  - Initialize state variables and statistical counters
  - Initialize event list (with already known future events)

Generic Simulation Program (2 of 4)

- Main loop (repeat until the condition for terminating the simulation is met)
  - `Determine the most imminent event and remove it from the event list (suppose this event is of type i)`
  - `Advance clock to the time of this event`
  - `Invoke event routine for type i`

Generic Simulation Program (3 of 4)

- Event routine (a separate routine for each event type)
  - `Update state variables`
  - `Update statistical counters`
  - `When required, add future events to the event list`

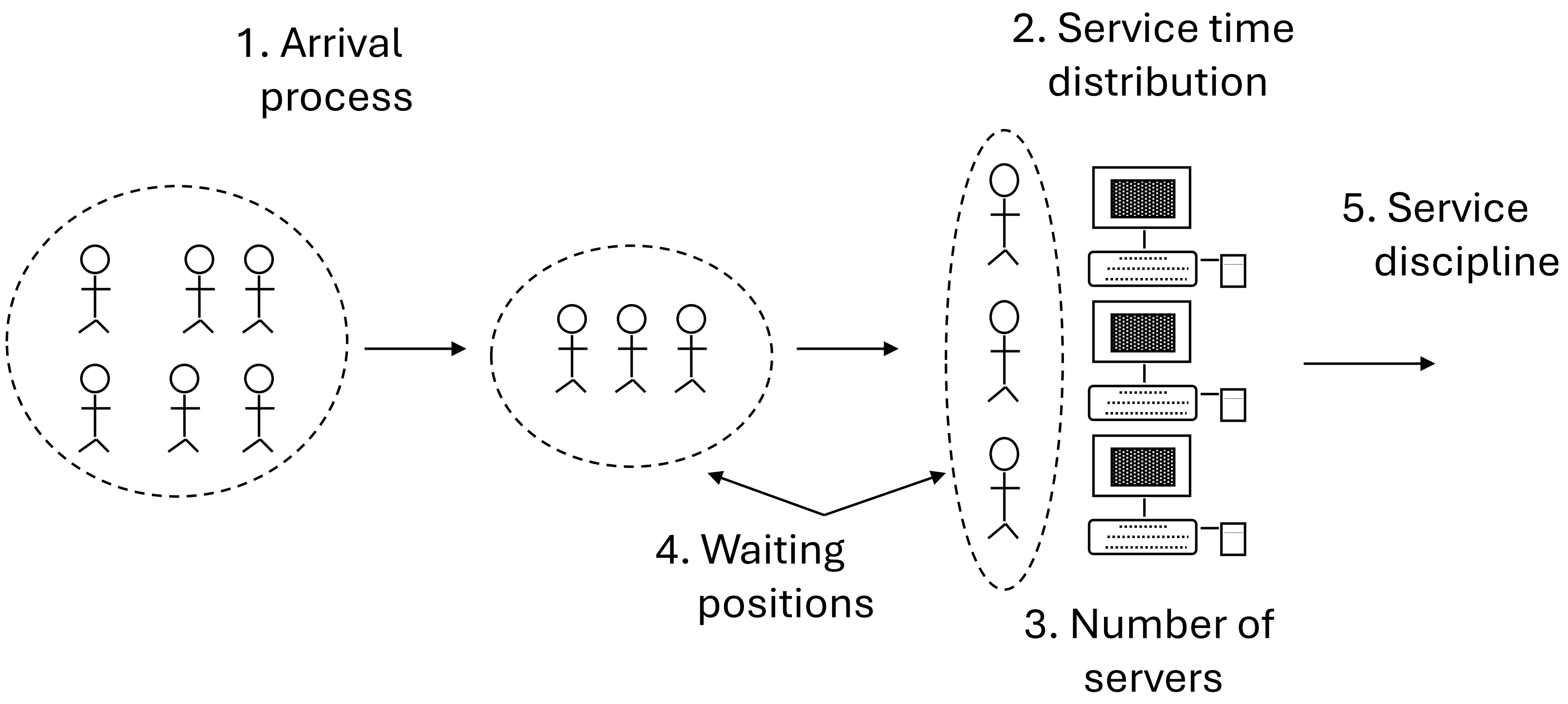
Generic Simulation Program (4 of 4)

- Report generator
  - `Invoked when simulation has terminated`
  - `Compute and output performance measures of interest`

Simulation of Queueing Systems

1. Single server infinite population
2. Single server finite population
3. Tandem queue
4. Tandem queue with blocking
5. Closed network model

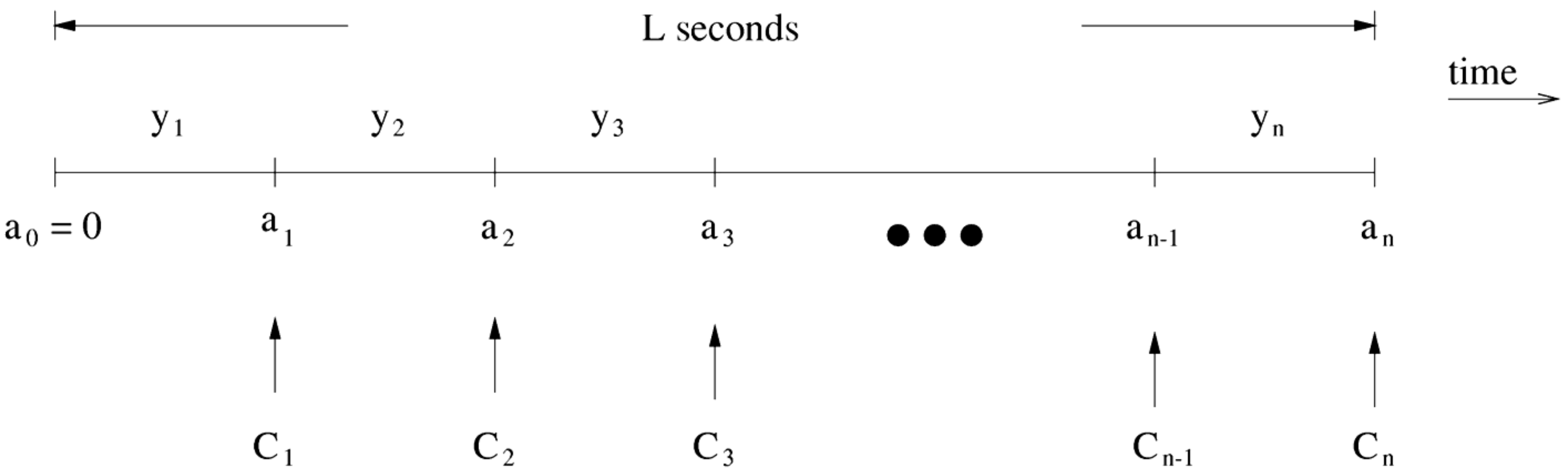
Components of a Queueing System



Arrival Definition

- Arrival time
  - Time at which a customer arrives at a service facility
- Inter-arrival time
  - Time between two successive arrivals to a service facility
- Arrival rate
  - Number of arrivals per unit of time

Timing Diagram



C_j - customer j

a_j - arrival time of C_j

y_j - interrarrival time between C_{j-1} and C_j

Relationship between Arrival Rate and Inter-arrival Time

- Define: $$L = \sum_{j=1}^{n} y_j$$

- Mean inter-arrival time $$= \frac{L}{n}$$

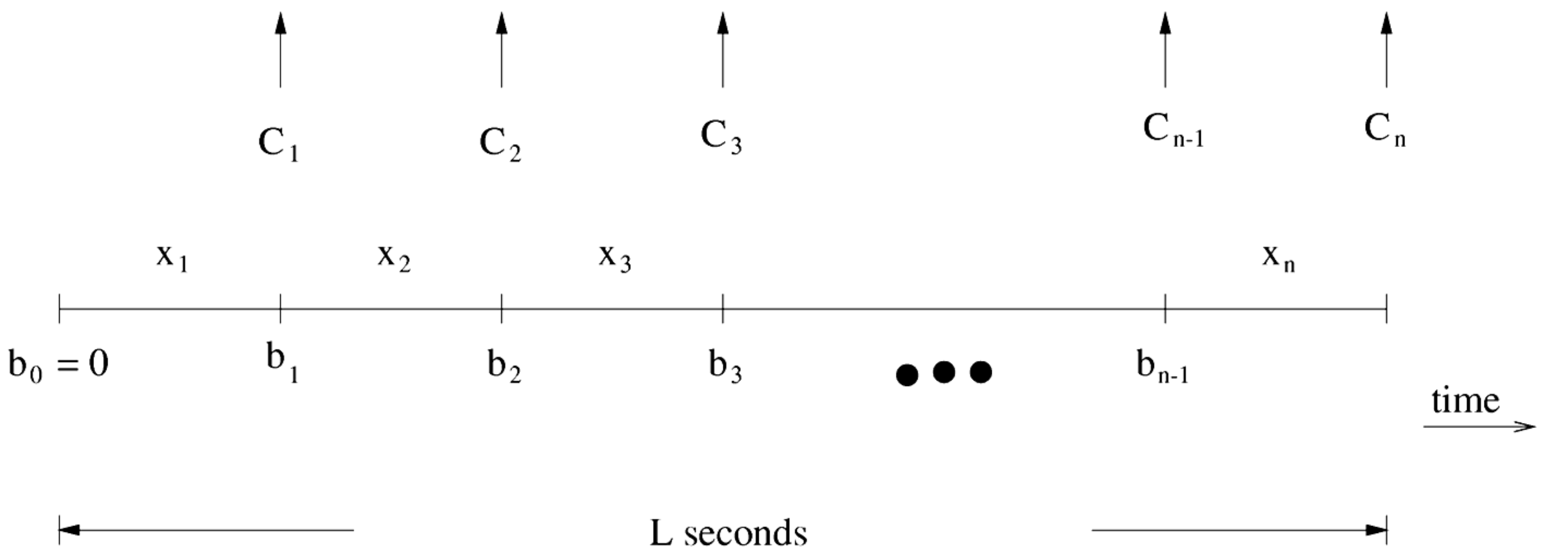
- Arrival rate

$$= \frac{n}{L} = \frac{1}{\text{mean interarrival time}}$$

Service Definition

- Service requirement - in units of work
  - Ex 1: CPU - unit of work is “instruction”
  - Ex 2: communication channel - unit of work is “bit”
- Server capacity - in units of work per second
  - Ex 1: CPU - server capacity is in “number of instructions executed per second”
  - Ex 2: communication channel - server capacity is in “number of bits transmitted per second”
- Service time = $\dfrac{\text{service requirement}}{\text{server capacity}}$
- Service rate
  - Number of customers served per second (assuming no idle time)

Timing Diagram



C_j - customer j

x_j - service time of C_j

b_{j-1} - time at which C_j starts service

Relationship between Service Rate and Service Time

- Define: $$L = \sum_{j=1}^{n} x_j$$

- Mean service time $$= \frac{L}{n}$$

- Service rate $$= \frac{n}{L} = \frac{1}{\text{mean service time}}$$

Service Disciplines

Examples:

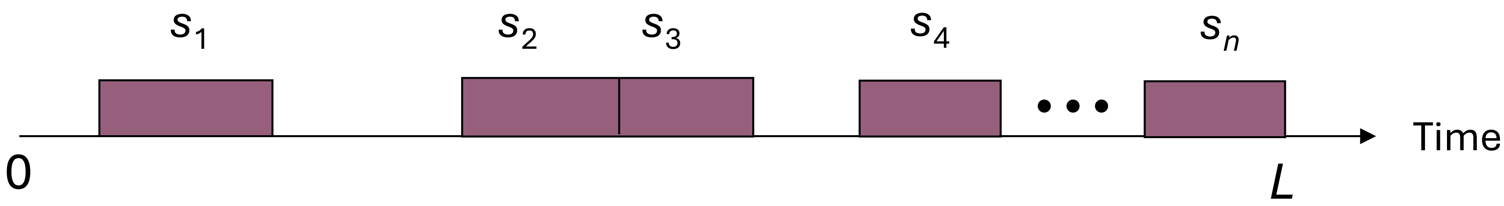
- First-Come-First-Serve (FCFS)   (a.k.a. FIFO)
- Last-Come-First-Serve (LCFS)
- Round-Robin (RR) with a fixed quantum
  Infinitesimal quantum $\Rightarrow$ Processor Sharing (PS)
- Shortest Job First (SJF)
- Shortest Remaining Processing Time (SRPT)
- And many more…

Typical Performance Measures

- Response time
  - Elapsed time from arrival to departure
- Waiting time
  - Time spent in queue
- Number of customers in system
- Number of customers in queue
- Server utilization
  - Proportion of time that the server is busy
- Throughput
  - Rate at which customers leave the service facility after completing service

Utilization

- Proportion of time that the server is busy



- Total busy time = $\sum_{j=1}^{n} s_j$
- U = proportion of time server is busy = $\frac{1}{L}\sum_{j=1}^{n} s_j$

Note: $U \leq 1$

Throughput

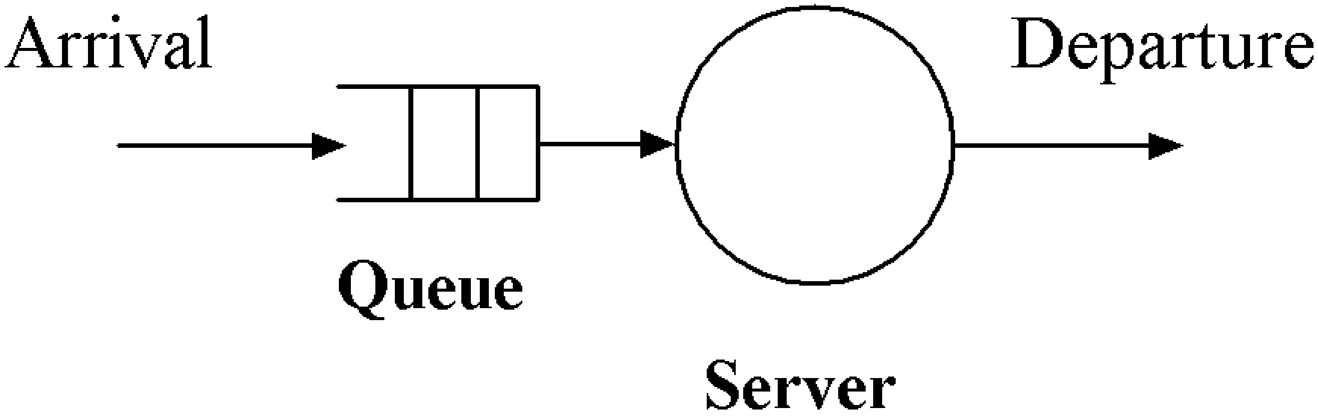
- Rate at which customers leave a service facility after completing service
  - Throughput:

$$\boldsymbol{R} = \frac{\boldsymbol{n}}{\boldsymbol{L}}$$

where $\boldsymbol{n}$ is the number of customers served in time $\boldsymbol{L}$

Single Server Queue Example (ssq3.c)

- Infinite population model



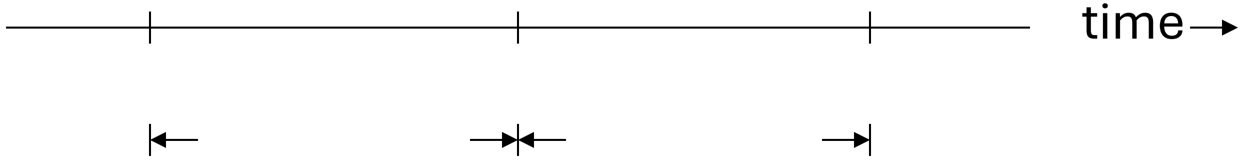
Customer Arrivals - Infinite Population Model

- Number of users of the service facility is large (potentially infinite)
- Pattern of customer arrivals is based on combined behavior of the customers, and is assumed to be independent of the state of the system

Single Server Queue Example

Events arrival start service departure

time →



Activities waiting in queue receiving service

Single Server Queue Model

- Assumptions
  - Inter-arrival times are independent of system state
  - Inter-arrival times are iid (independent and identically distributed)
  - Service times are independent of system state
  - Service times are iid
  - FCFS scheduling
  - System is empty at time zero
  - Arrival of first customer occurs after the first inter-arrival time
  - Simulation terminates when the m-th customer starts service

Single Server Queue Model

- Input parameters
  - Inter-arrival time distribution (e.g., exponential)
  - Service time distribution (e.g., uniform)

- Performance measures of interest
  - Mean waiting time in queue, $\overline{w}$
  - Mean number of customers in system, $\bar{n}$

Single Server Queue Model

- State variables
  - `status` = server status (busy or idle)
  - `n` = number of customers in system

- Statistical counters
  - `nw` = number of waiting times accumulated
  - `sw` = sum of accumulated waiting times
  - `sa` = sum of accumulated areas (for calculating $\bar{n}$)
  - `last_event` = time of last event when accumulating area

Single Server Queue Model

- Lists
  - event_list
  - queue

- Event types
  - type 1: arrival
  - type 2: start_service
  - type 3: departure

Single Server Queue Model (1 of 4)

- Initialization
  - `clock = 0`
  - `status = idle`
  - `n = 0`
  - `nw = sw = 0`
  - `last_event = 0`
  - `sa = 0`
  - Initialize `queue` to empty
  - Initialize `event_list` to empty
  - Determine `inter_t`, the first interarrival time
  - Schedule an arrival event to occur at `clock + inter_t`

Single Server Queue Model (2 of 4)

- Main loop (repeat until the condition for terminating the simulation is met)
  - Determine the most imminent event and remove it from the event list (suppose this event is of type `i` and occurs at time `t`)
  - `clock = t`
  - `sa = sa + (clock - last_event)· n`
  - `last_event = clock`
  - Invoke event routine for type `i`

Single Server Queue Model (3a of 4)

- arrival event – type 1
  - Determine `inter_t`, the interarrival time between the current and next arrivals
  - Schedule an arrival event to occur at `clock + inter_t`
  - `n = n + 1`
  - Enter arriving customer to end of queue, and save its time of arrival (given by clock)
  - If status is `idle`, invoke routine for `start_service` event

Single Server Queue Model (3b of 4)

- start_service event – type 2
  - Remove customer from front of queue, and retrieve time of arrival (`t_arrival`)
  - `nw = nw + 1`
  - `sw = sw + (clock - t_arrival)`
  - If `nw = m` (condition for terminating simulation), `exit` main loop
  - `status =  busy`
  - Determine `serv_t`, the customer service time
  - Schedule a departure event to occur at `clock + serv_t`

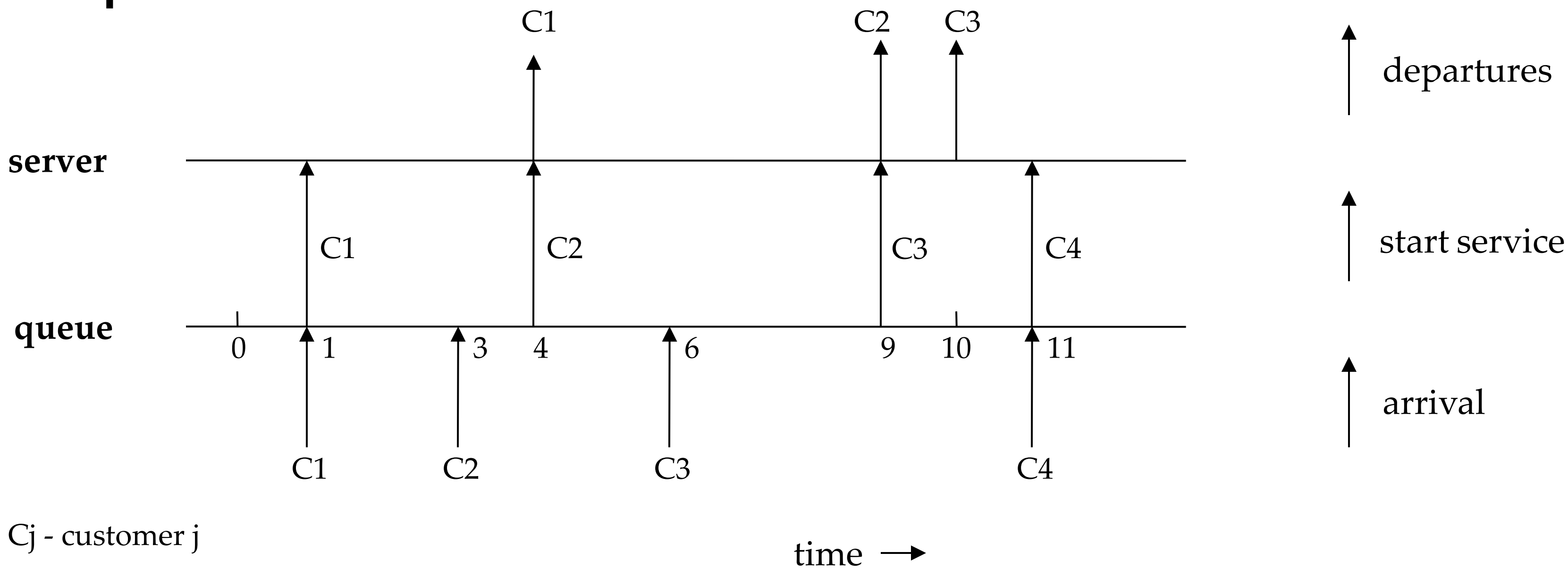
Single Server Queue Model (3c of 4)

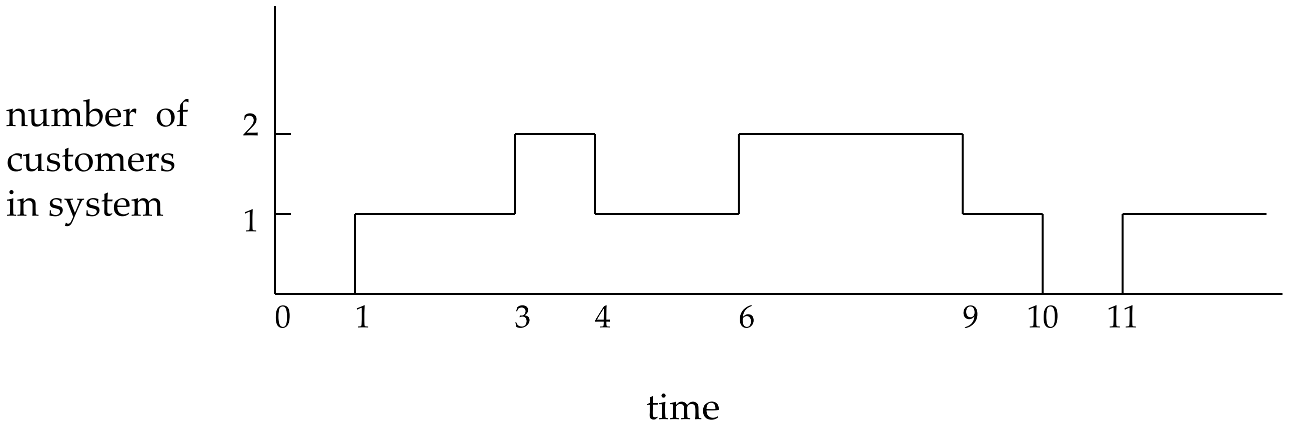
- departure event – type 3
  - `n = n - 1`
  - `status = idle`
  - If `n > 0`, invoke event routine for `start_service` event

Single Server Queue Model (4 of 4)

- Report generator
  - Mean waiting time: $\bar{w}$ `= sw/nw`
  - Mean no. of customers in system: $\bar{n}$ `= sa/clock`
  - Output results

Sequence of Events





Manual Trace of Single Server Queue Example

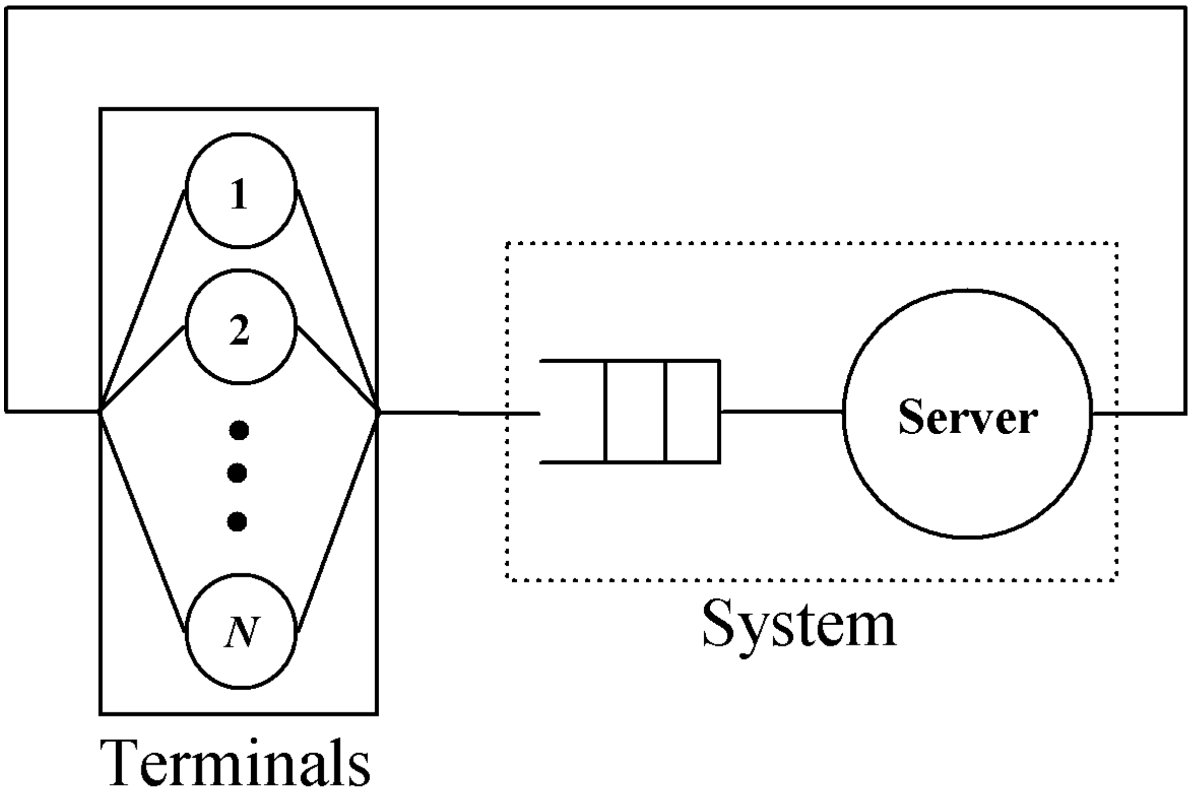
clock	event	status	n	event list	queue	nw	sw
0	--	idle	0	(A, 1)	empty	0	
1	A	busy	1	(A, 3), (D, 4)	empty	1	0
3	A	busy	2	(D, 4), (A, 6)	(C2, 3)	1	0
4	D	busy	1	(A, 6), (D, 9)	empty	2	1
6	A	busy	2	(D, 9), (A, 11)	(C3, 6)	2	1
9	D	busy	1	(D, 10), (A, 11)	empty	3	4
10	D	idle	0	(A, 11)	empty	3	
11	A	busy	1	(A, 15), (D, 17)	empty	4	4

mean waiting time = sw/nw = 1.0

Notation: A - arrival event, D - departure event
(Cj, x) - customer j in queue, time of arrival of this customer is x
n - number of customers in system

Single Server Queue Example

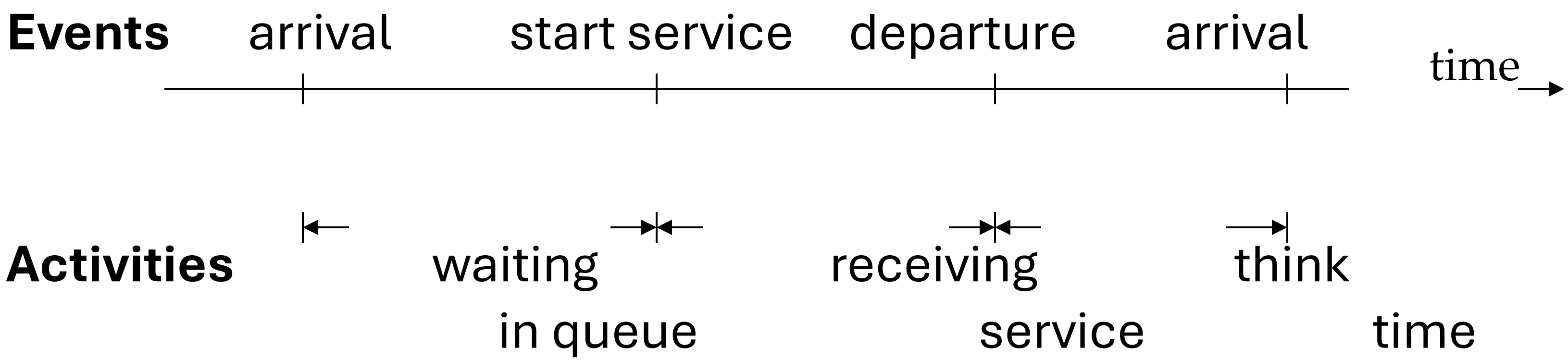
- Finite population model



Finite Population Model

- Number of users is not large
- The behavior of each user is modeled explicitly as far as arrival pattern is concerned
- Arrival rate is dependent on the state of the system
- Definition
  - Think time: elapsed time from completion of previous request to submission of next request

Finite Population Model



Finite Population Model

- Assumptions
  - Service times are iid and independent of system state
  - Think times are iid and independent of system state
  - FCFS scheduling
  - System is empty at time zero
  - For each of the `N` users, the first request is submitted after a think time
  - Subsequent arrivals depend upon prior service completions
  - Simulation terminates at time `term_sim`

Finite Population Model

- Initialization
  - `clock = 0`
  - `status = idle`
  - `n = 0`
  - Initialize `queue` to empty
  - Initialize `event_list` to empty
  - for user `j (j = 1 to N)`
    Determine `think_t`, a think time of user `j`,
    and schedule an arrival event at `clock + think_t`

    end for
  - Schedule an `end_simulation` event at `term_sim`

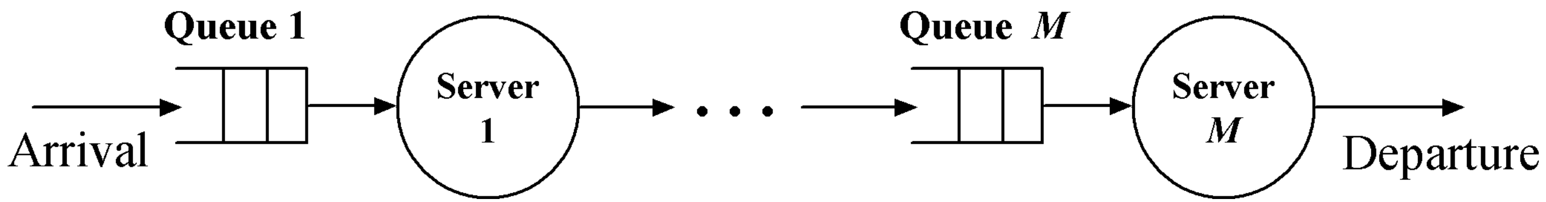
Finite Population Model

- arrival event
  - `n = n + 1`
  - Enter arriving customer to end of queue
  - If status is `idle`, invoke routine for `start_service` event
- start_service event
  - Remove customer from front of queue
  - `status = busy`
  - Determine `serv_t`, the service time of customer
  - Schedule a departure event to occur at `clock + serv_t`

Finite Population Model

- departure event
    - `n = n - 1`
    - `status = idle`
    - Determine `think_t`
    - Schedule an arrival event at `clock + think_t`
    - If `n > 0`, invoke event routine for `start_service`
- end-simulation event
    - `exit` main loop

Example: Tandem Queue – M Stages

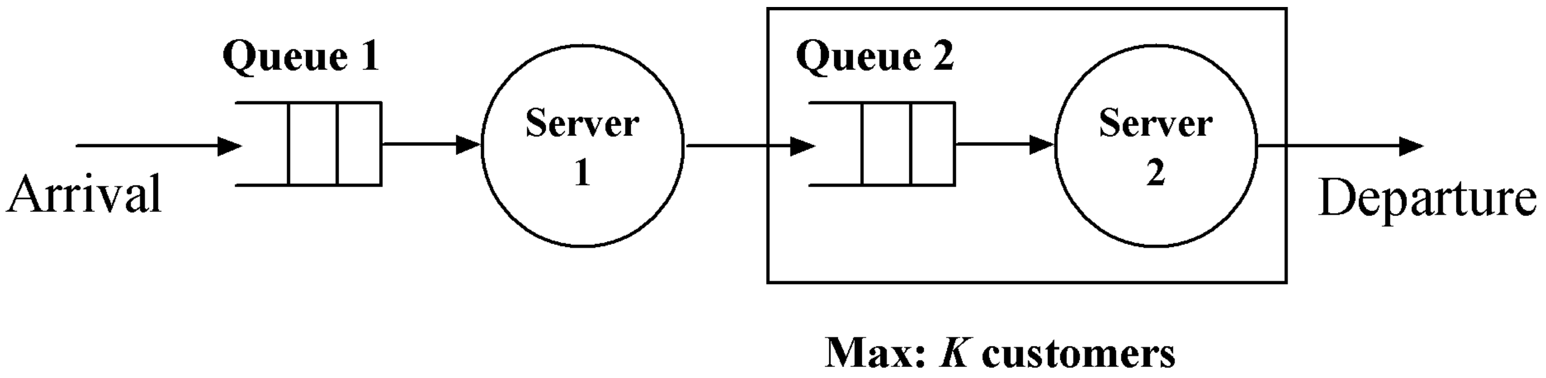


- Subsystems and interactions
  - `M` subsystems – one for each stage
  - A departure from stage `i` becomes an arrival to stage `i+1 (i = 1 to M-1)`

Simulation Program

- Use event routines for single server queue model for each of the `M` stages
- Modifications to implement tandem queue:
  - departure event: for stage `i (i = 1 to M-1)` – add the step
    - Invoke routine for arrival event at stage `i+1`
  - arrival event: for stage `i (i = 2 to M)` – do <u>not</u> schedule the next arrival event!

Example: Tandem Queue with Blocking



Tandem Queue with Blocking

- Two stages
- Finite waiting room at stage 2 (number of customers in system < K)
- Blocking
  - Server 1 is blocked if a customer completing service at stage 1 finds no queuing space at stage 2

Tandem Queue with Blocking

- Subsystems and interaction
  - 2 subsystems – one for each stage
  - Server 1 is blocked if a customer completing service at stage 1 finds no queuing space at stage 2
  - If server 1 is in the “blocked” state, it becomes “not blocked” when a departure occurs at stage 2
  - A departure from stage 1 becomes an arrival to stage 2

Simulation Program

- Use event routines for single server queue (infinite population model) for each of the 2 stages
- Modifications to implement tandem queue with blocking:
  - Add state variable b
    - `b = 1` if server 1 is blocked and 0 if server 1 is not blocked
  - Initialization : add the step
    - `b = 0`

Simulation Program

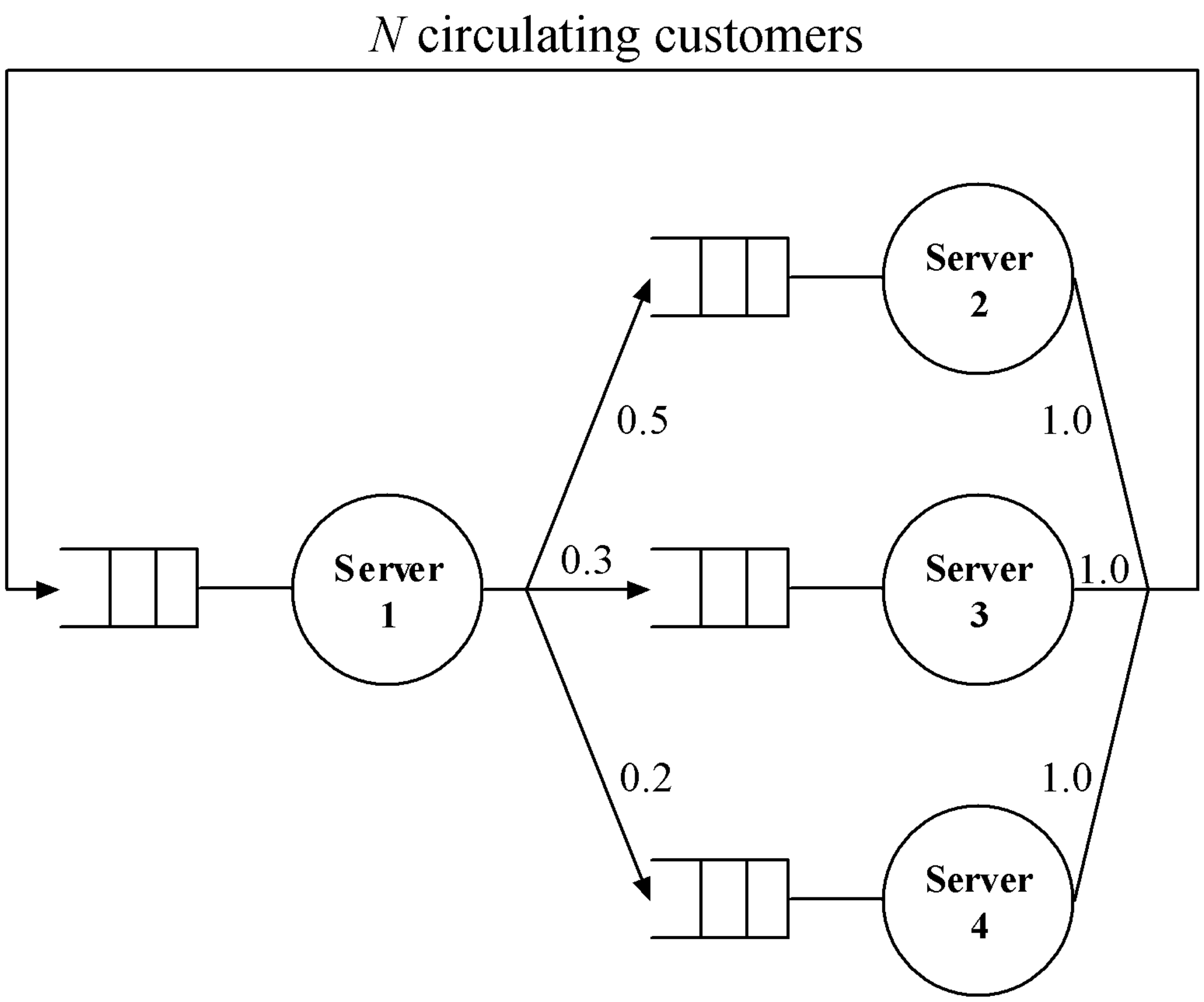
- Modifications (cont.):
  - start_service event at stage 1: add the step
    - `Schedule an end_service event at stage 1 instead of a departure event at stage 1`
  - end_service event at stage 1: add the step
    - `If number in system at stage 2 < K`

      `  invoke routine for departure event at stage 1`
    - `else b = 1`

Simulation Program

- Modifications (cont.):
  - departure event at stage 1: add the step
    - `Invoke routine for arrival event at stage 2`
  - arrival event at stage 2:
    - `Do not schedule the next arrival event`
  - departure event at stage 2: add the step
    - `If b = 1, then b = 0 and invoke routine for departure event at stage 1`

Example: Closed Network Model



Subsystems and Interaction

- Four subsystems, one for each server
- Interaction is defined by transition probabilities
  - A customer departing from server 1 has
    - 50% probability of arriving at server 2
    - 30% probability of arriving at server 3
    - 20% probability of arriving at server 4
  - A customer departing from server 2, 3 or 4 has 100% probability of arriving at server 1

Simulation Program

- Single server queue model for each subsystem with modifications to model the interaction
- Initialization
  - `clock = 0`
  - `for i = 1 to 4`
    - Initialize `queue(i)` to empty
    - `status(i) = idle`
    - `n(i) = 0`
  - Initialize `event_list` to empty
  - Enter `N` customers at end of `queue(1)`
  - `n(1) = N`
  - Invoke `start_service` event at server 1

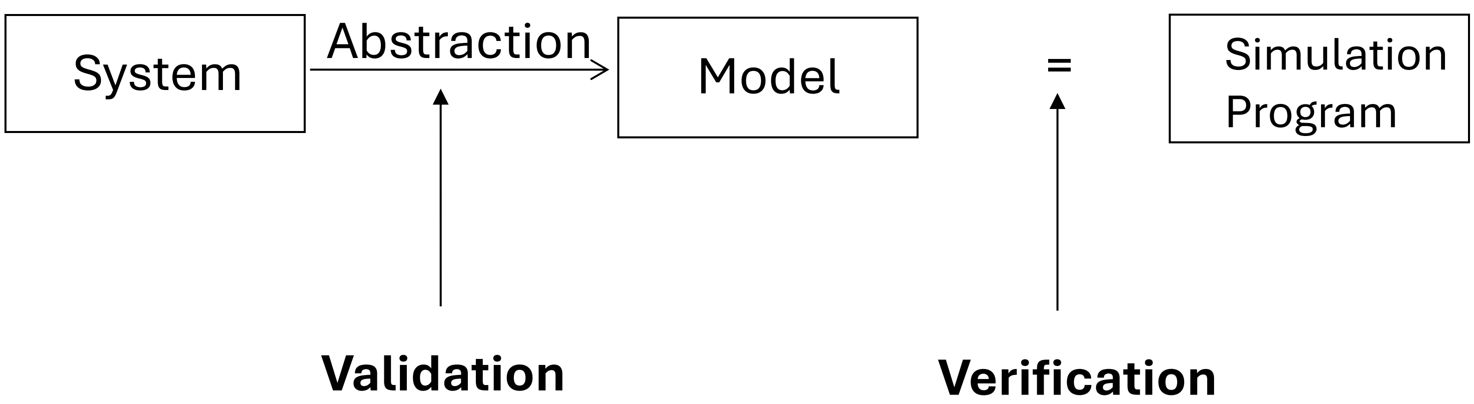
Simulation Program

- Departure event from server 1: add the steps
  - Determine k, the ID of the next server for the departing customer ($k = 2$: 50%, $k = 3$: 30%, $k = 4$: 20%)
  - Invoke arrival event to server k
- Departure event from server 2, 3, or 4: add the step
  - Invoke arrival event to server 1
- Arrival event at server 1, 2, 3, or 4
  - Do not schedule the next arrival event

Outline

- Concepts in discrete-event simulation
  - Terminology and concepts
  - Two pedagogical examples
- Components of discrete-event simulation
  - Time advance approaches
  - Event scheduling approach
- Manual simulation
  - Grocery store example
- Simulation program
  - Simulation of queuing systems
  - Infinite and finite population model
  - Tandem queue with blocking
- Verification and validation of simulation models

Verification and Validation



Verification

- Increase the level of confidence in the correctness of simulation program
- Approaches
  - Use a "trace" to debug simulation program
    - Trace is obtained by printing state variables, statistical counters, etc., after each event
  - Verify simulation output using analytic results

Fundamental Results

- Use fundamental results of queuing systems
- Examples
  - For any subsystem, mean arrival rate, mean number in system, and mean response time must be consistent with Little’s formula

Analytic Results

- Check results for cases where analytic results are known
- Examples
  - Simulation model: open networks with exponential interarrival time distribution and uniform service time distribution
  - Run simulation for the case of exponential service time distribution (analytic solution is available)
  - Verify if the simulation output is consistent with known analytic results

Validation

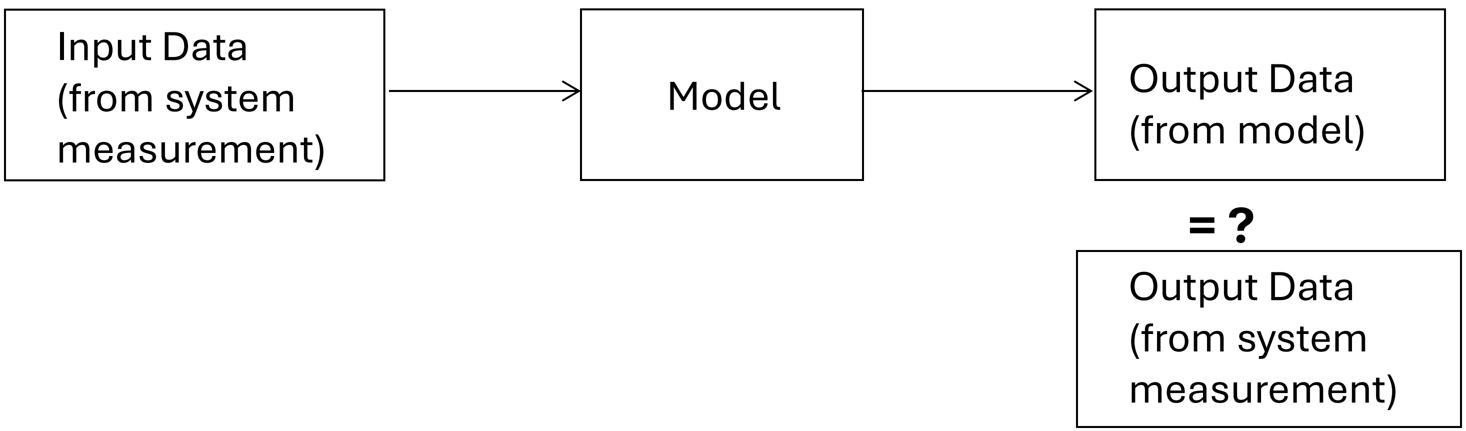
- Model should be "good enough" (subjective)
- Seek expert opinion on system components that need to be carefully modeled, e.g., bottleneck
- A model should be valid for the performance measures
- The most valid model may not be the most cost-effective model

Three Step Approach to Validation

1. Build a model with high face validity
    - Appears to be reasonable to people who are knowledgeable about the system being modeled
2. Validation of model assumptions
    - Structural assumptions: entities, attributes, sets, etc.
    - Data assumptions
        - Collect reliable data
        - Identify appropriate distribution
        - Validate the assumed distribution

Three Step Approach to Validation

3. Validation of input-output relationship
   - Model should be able to predict system behavior under existing conditions



Could be done using historical data collected for validation purposes.