



Discrete Event Simulation

IN2045

Chapter 1 – Simulation Types

Dr. Alexander Klein

Stephan Günther

Prof. Dr.-Ing. Georg Carle

Chair for Network Architectures and Services
Department of Computer Science
Technische Universität München
<http://www.net.in.tum.de>



Some of today's slides/figures are borrowed from:
Richard Fujimoto,
James Kurose,
Keith W. Ross,
Joachim Warschat
Oliver Rose, Averill
Law, David Kelton,
Manfred Jobmann



Simulation Fundamentals

A computer simulation is a computer program that models the behaviour of a **physical system** over time.

- Program variables (**state** variables) represent the current state of the physical system.
- Simulation program modifies state variables
 - ...to model the evolution of the physical system over time
 - ...and/or to incrementally enhance the level of detail of the physical system's state

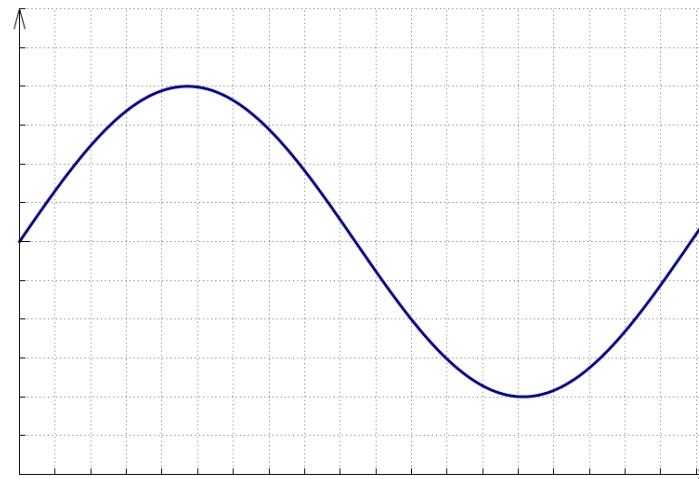


Model taxonomy

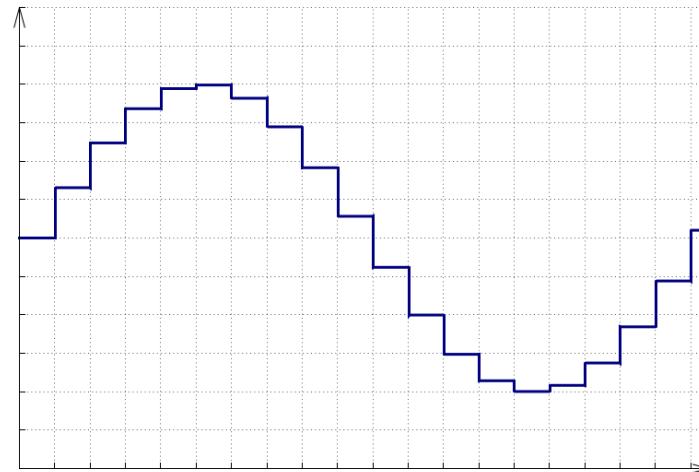
- Static vs. dynamic
 - Static: Simulate state at one point in time / without time
 - Dynamic: State changes over time (**focus of lecture!**)
- Deterministic vs. stochastic
 - Deterministic: The same input always effects the same output
 - Stochastic: Under same conditions, same input may yield different outputs
Usual reason: Environment modeled as pseudo-random input
- Continuous vs. discrete
 - cf. next slides ...



Continuous vs. discrete simulation



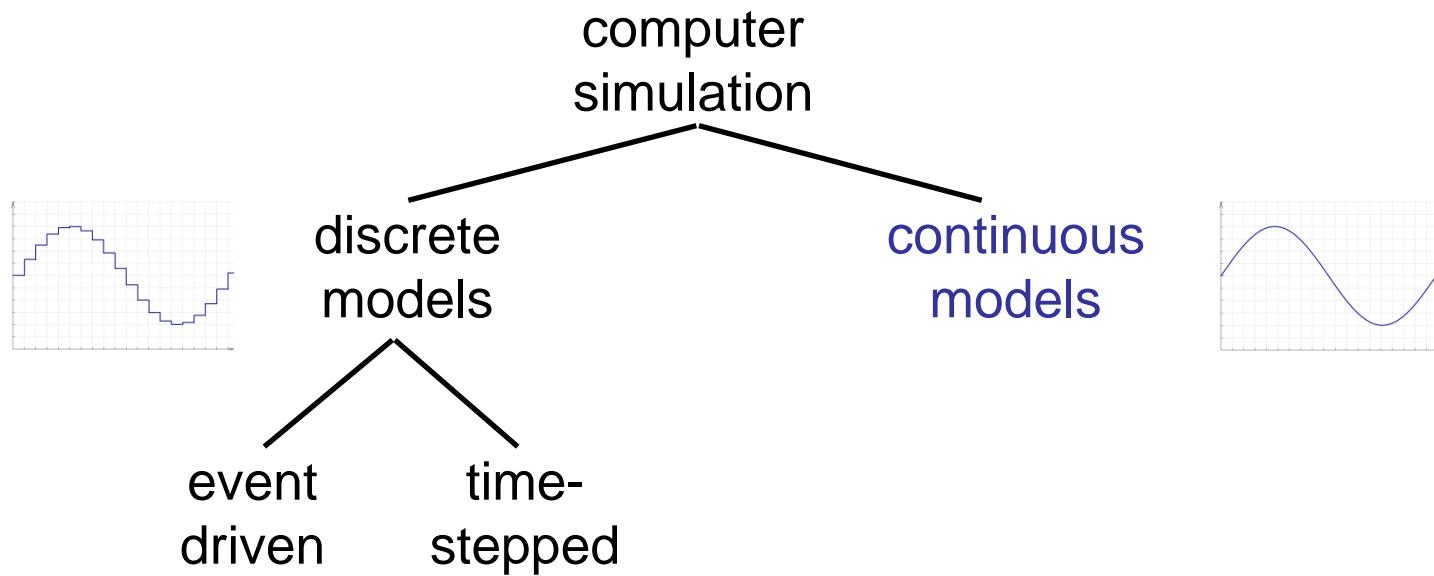
Continuous values



Discrete values



Simulation Taxonomy (1)

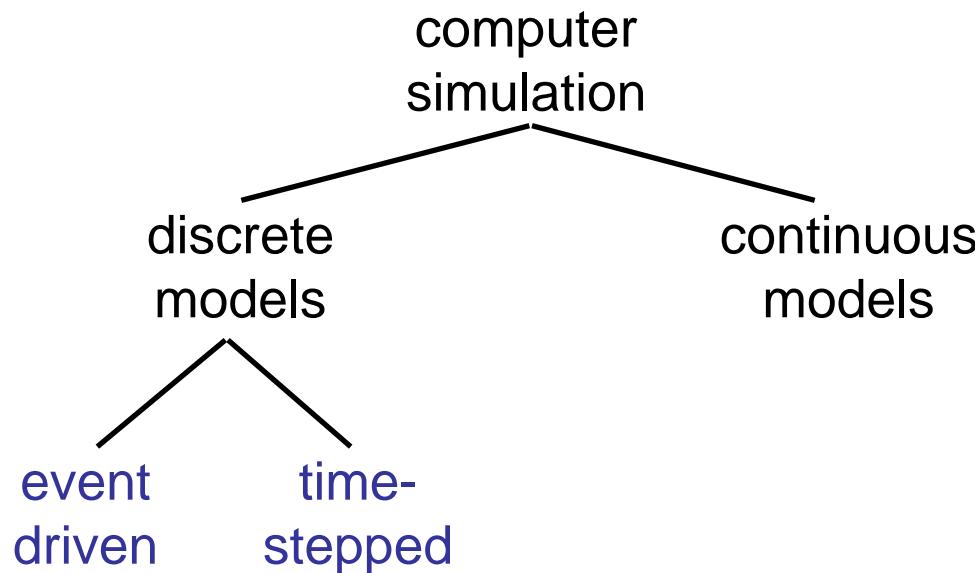


Continuous time simulation

- State changes occur continuously across time
- Typically, behavior described by differential equations
- Example: Flight simulator (time and space are not quantised – at least not at macroscopic dimensions...)



Simulation Taxonomy (2)



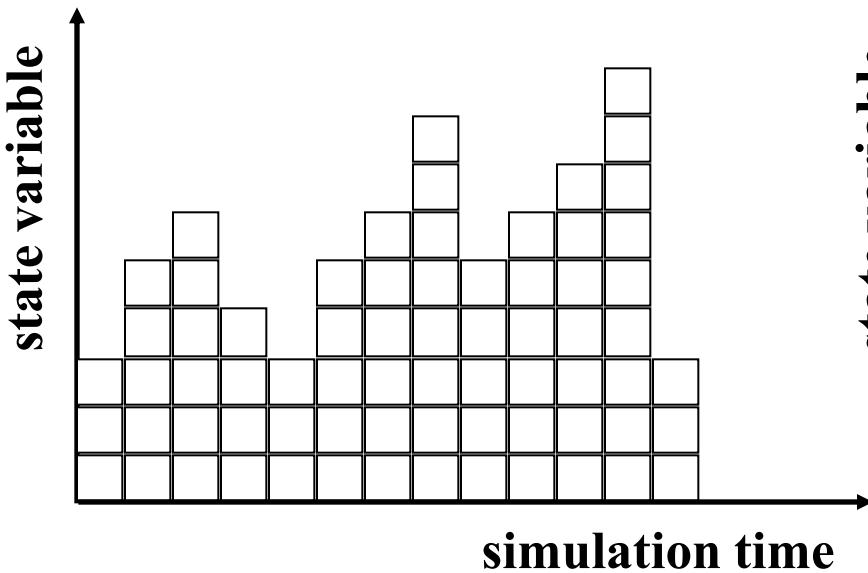
Discrete time simulation [zeitdiskrete Simulation]:

- State changes only occur at discrete time instants
 - Example: Simulating packets in a computer network
- **Time stepped:** time advances by fixed time increments
- **Event stepped:** time advances occur with irregular increments, i.e., to the next point “when something happens”
(cf. next slide)

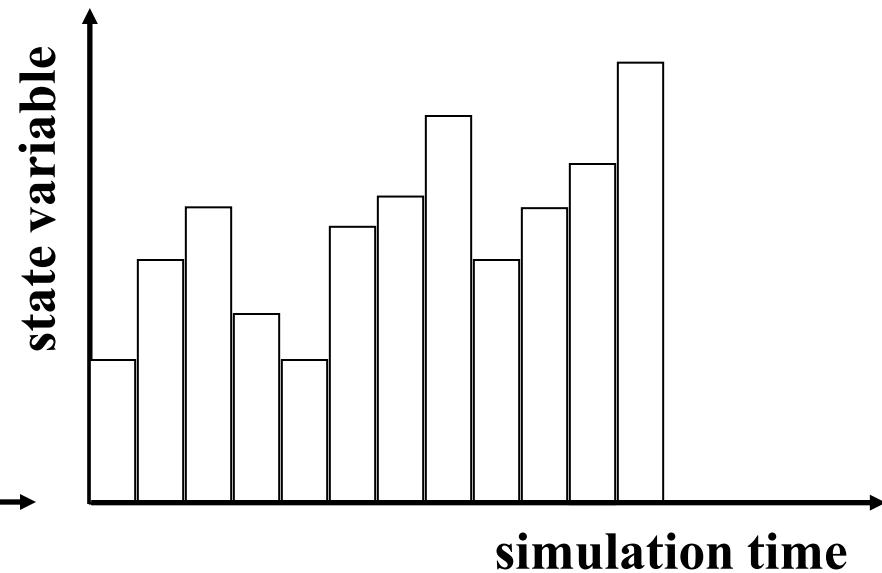


Discrete vs. continuous

Goal: compute state of system over simulation time



discrete

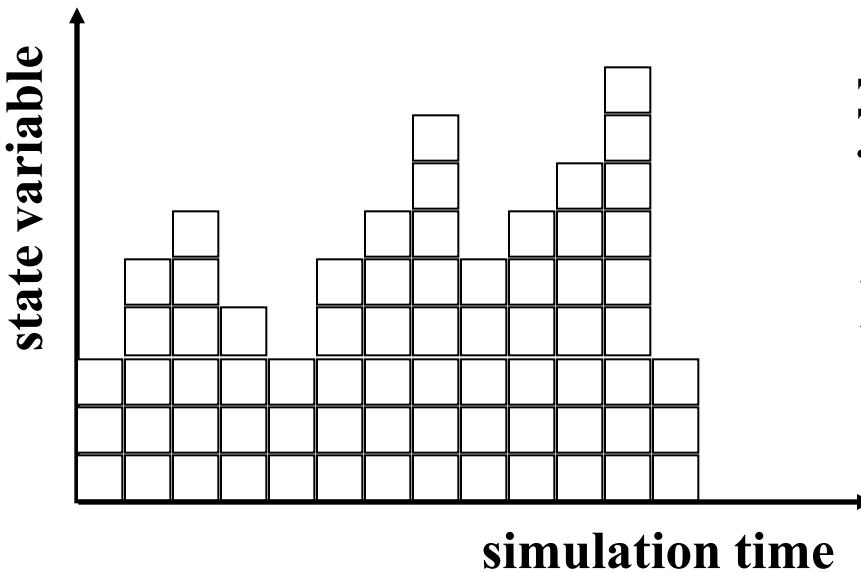


continuous

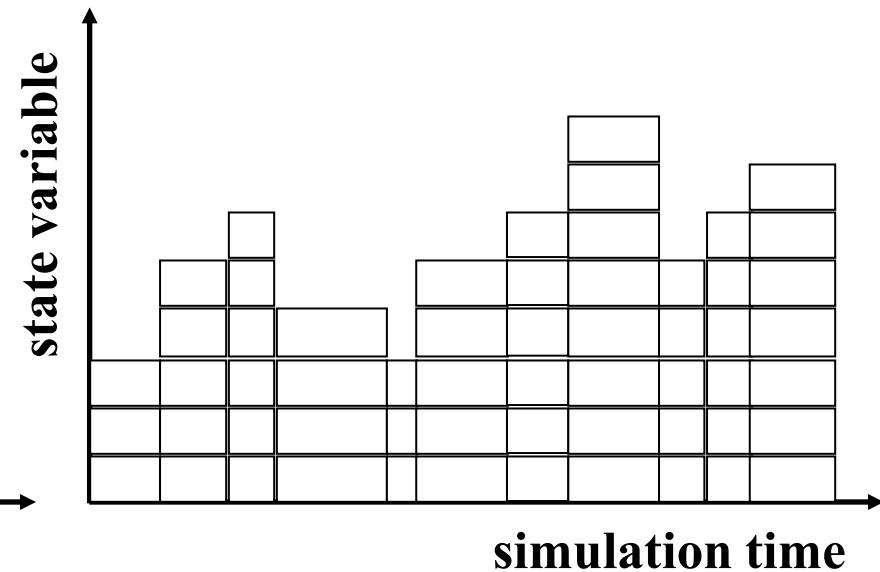


Time-stepped vs. event-stepped simulation

Goal: compute state of system over simulation time



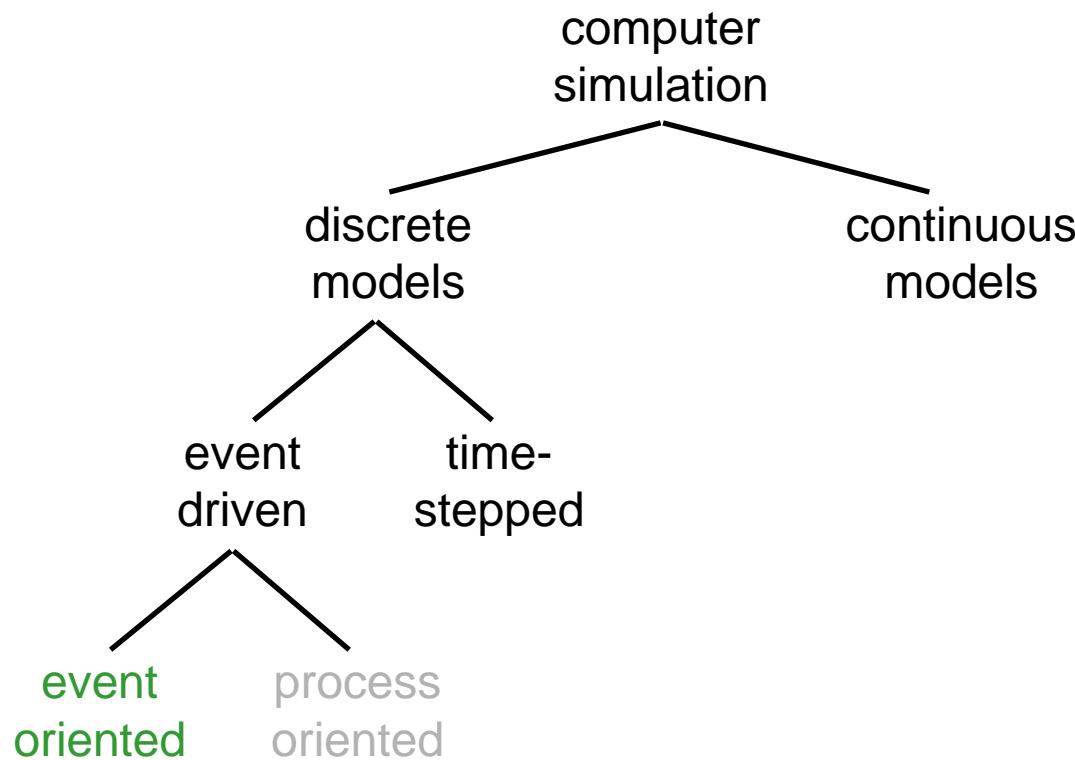
Time-stepped execution



Event-driven execution



Discrete Event Simulation



Event scheduling approach:

- Event-driven:
 - An event represents an action which might affect the system state.

State of the simulation can only change at the time an event is processed.



Discrete Event Simulation

- A Discrete Event Simulation (DES) is the reproduction of the behaviour of a system
 - over time
 - by means of a model where the state variables of the models change immediately at discrete points in time.
- These points in time are the ones at which an event occurs.
- Remark: There are (pseudo-)events that do not lead to changes in the state variables of the model, e.g.:
 - Data collection for statistics / writing to a log file
 - End of simulation
 - Manual garbage collection



Discrete Event Simulation – Waiting Systems

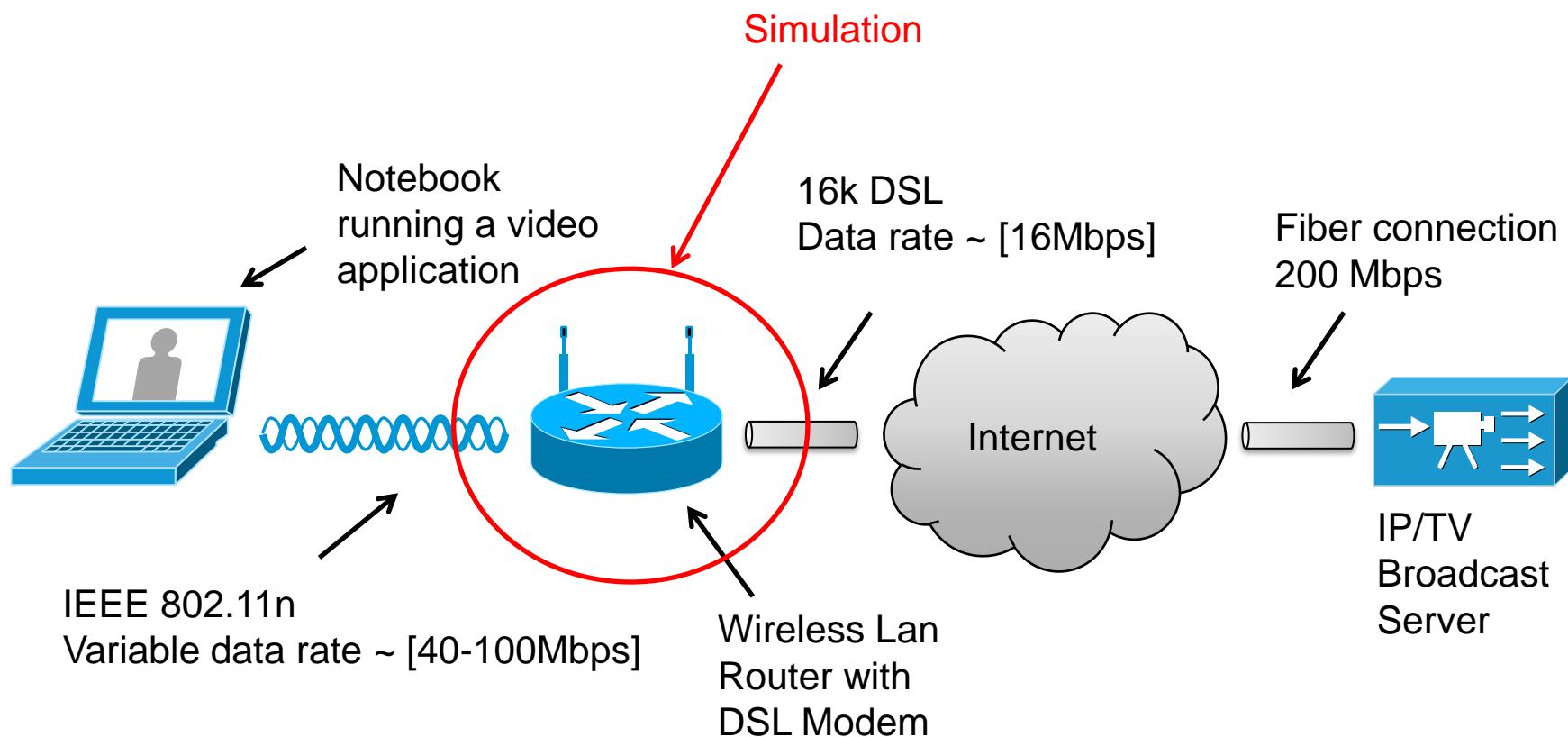
What are we talking about... and why?

- Simple queue model:
 - Customers arrive at random times
 - Execution unit serves customers (random duration)
 - Only one customer at a time; others need to queue
- Standard example
- Give deeper understanding of important aspects, e.g.
 - Random distributions (input)
 - Measurements, time series (output)
 - ...



Discrete Event Simulation

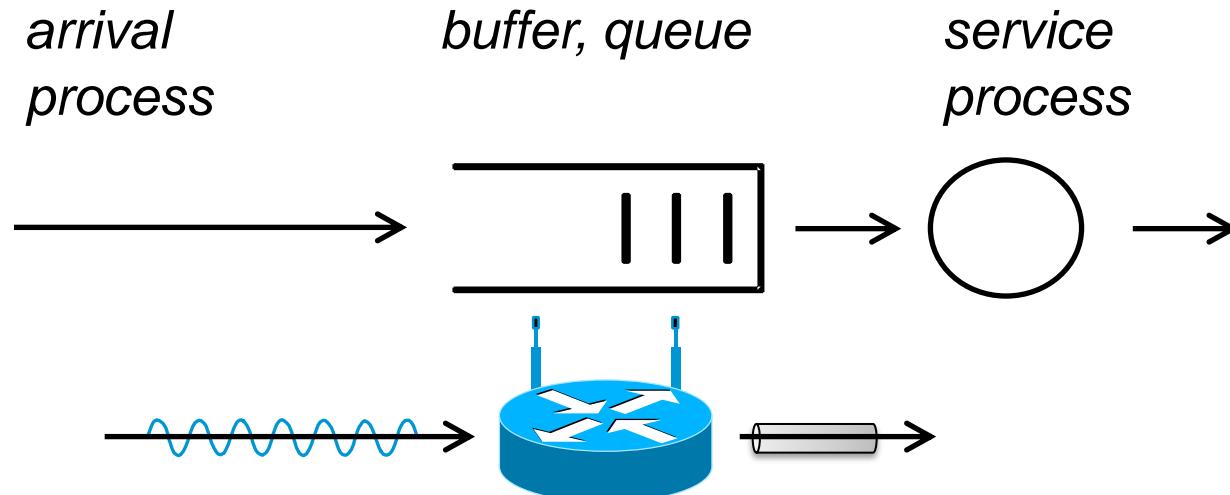
- Example:
Computer networks → Router





Discrete Event Simulation – Queuing Example

Waiting Queue Theory



- Example:

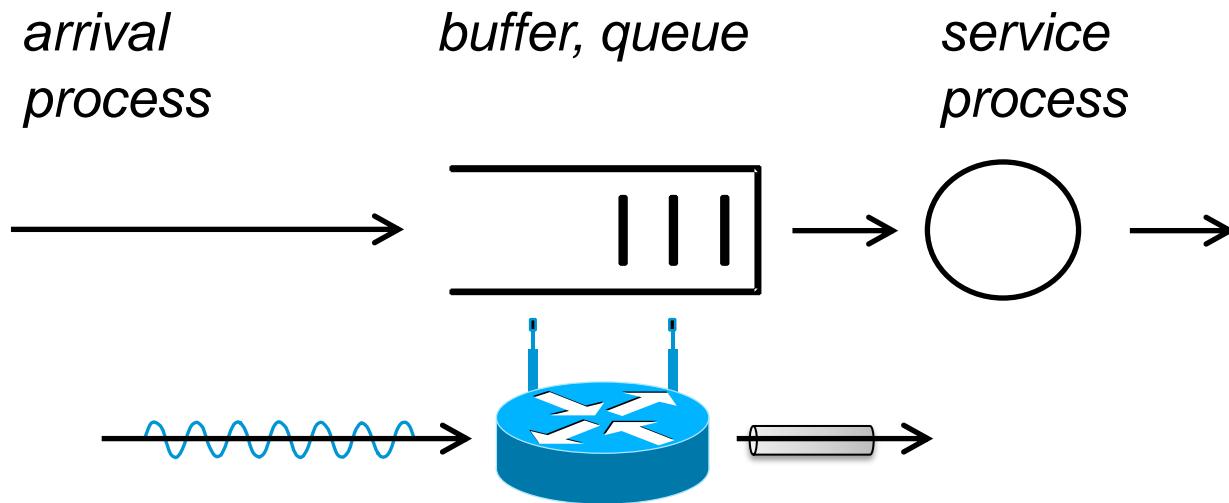
Router

- Data packets arrive at the router via its wireless interface
- A packet is forwarded immediately via the DSL interface if the buffer is empty and no packet is currently transmitted
- Otherwise the packet is stored in the buffer if the buffer is below its maximum capacity
- The service process simulates the time that is required by the router to write a packet on the DSL interface

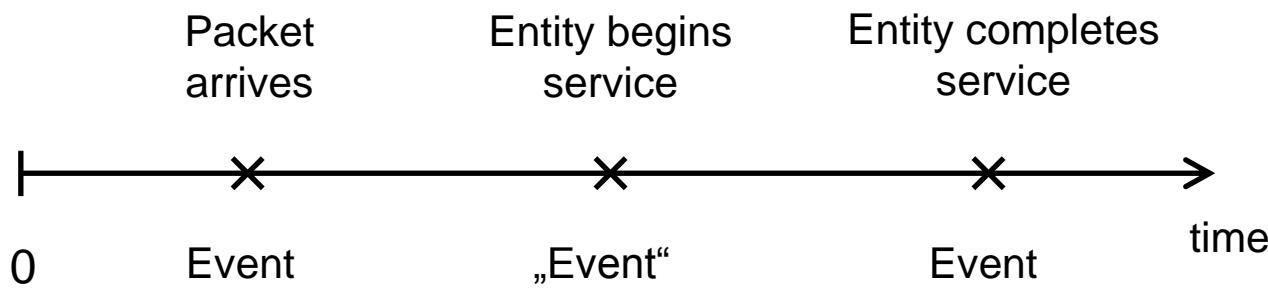


Discrete Event Simulation – Queuing Example

Waiting Queue Theory



- Entity/Packet flow:

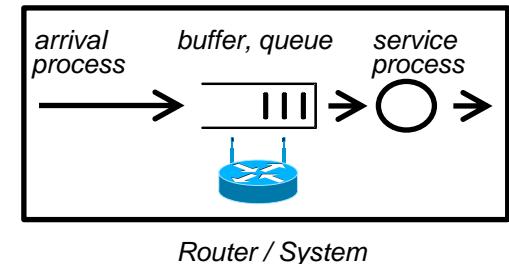




Discrete Event Simulation – Queuing Example

□ State variables:

- Fill state of the queue (discrete) between [0 ; S] with S being the maximum queue capacity
- State of the service process (discrete)
 - Idle (0)
 - Busy (1)



□ Events:

▪ Packet arrival:

A new packet arrives at the router

Process:

- Increase queue by one if service process is busy and queue size is below maximum capacity

▪ Service completion:

Service process has transmitted a packet



Discrete Event Simulation – Queuing Example

□ Events:

- **Packet arrival:**

A new packet arrives at the router

Process:

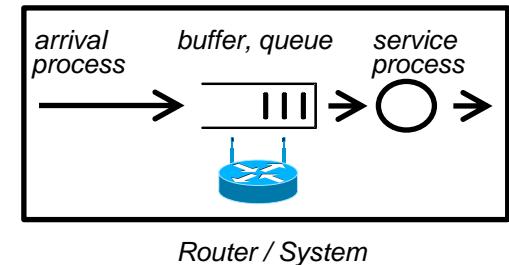
- Increase queue by one if service process is busy and queue size is below maximum capacity .
- If queue size is at its maximum capacity drop the packet.
- If service process is idle, set service process to busy state and schedule the next **service completion event**.

- **Service completion:**

Service process has transmitted a packet

Process:

- If queue size is equal to zero, set service process to idle.
- If queue size is greater than zero, reduce the queue size by one and schedule the next **service completion event**.

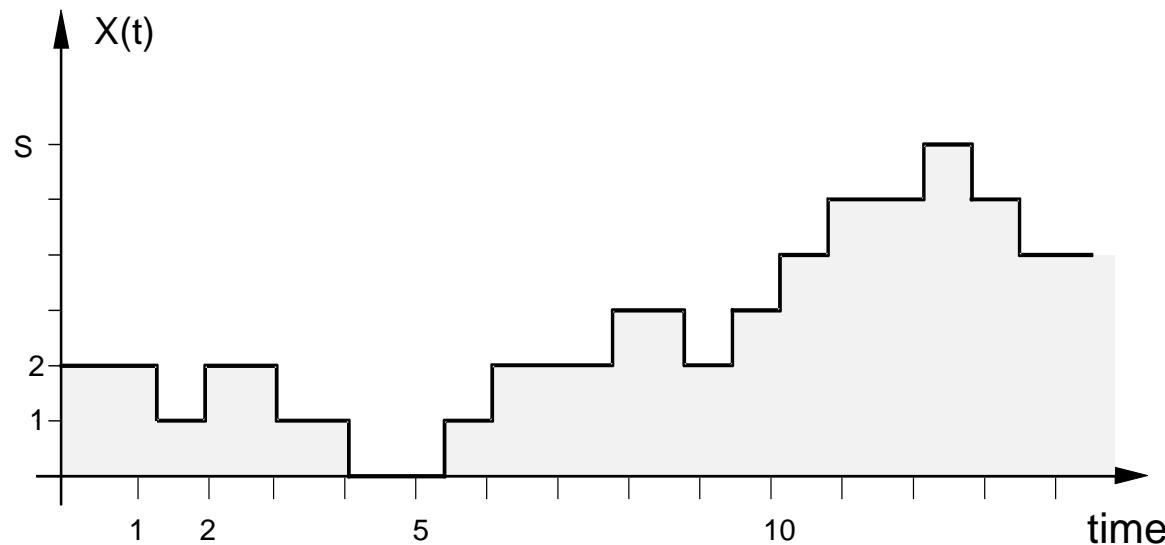




Discrete Event Simulation

□ System characteristic:

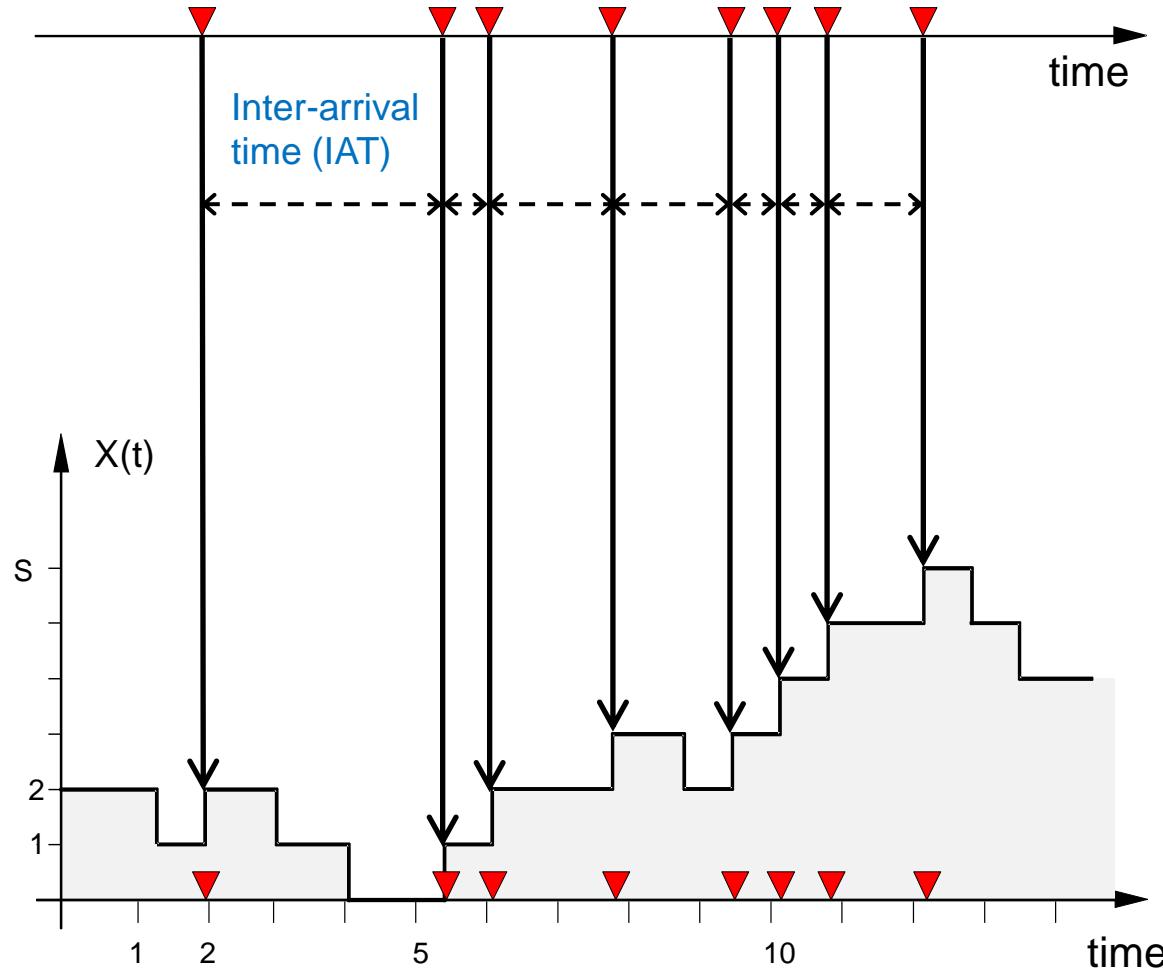
- Fill state of the queue at time is given by $X(t)$
- The fill state of the queue can only change when an event occurs





Discrete Event Simulation

□ Arrival Events:

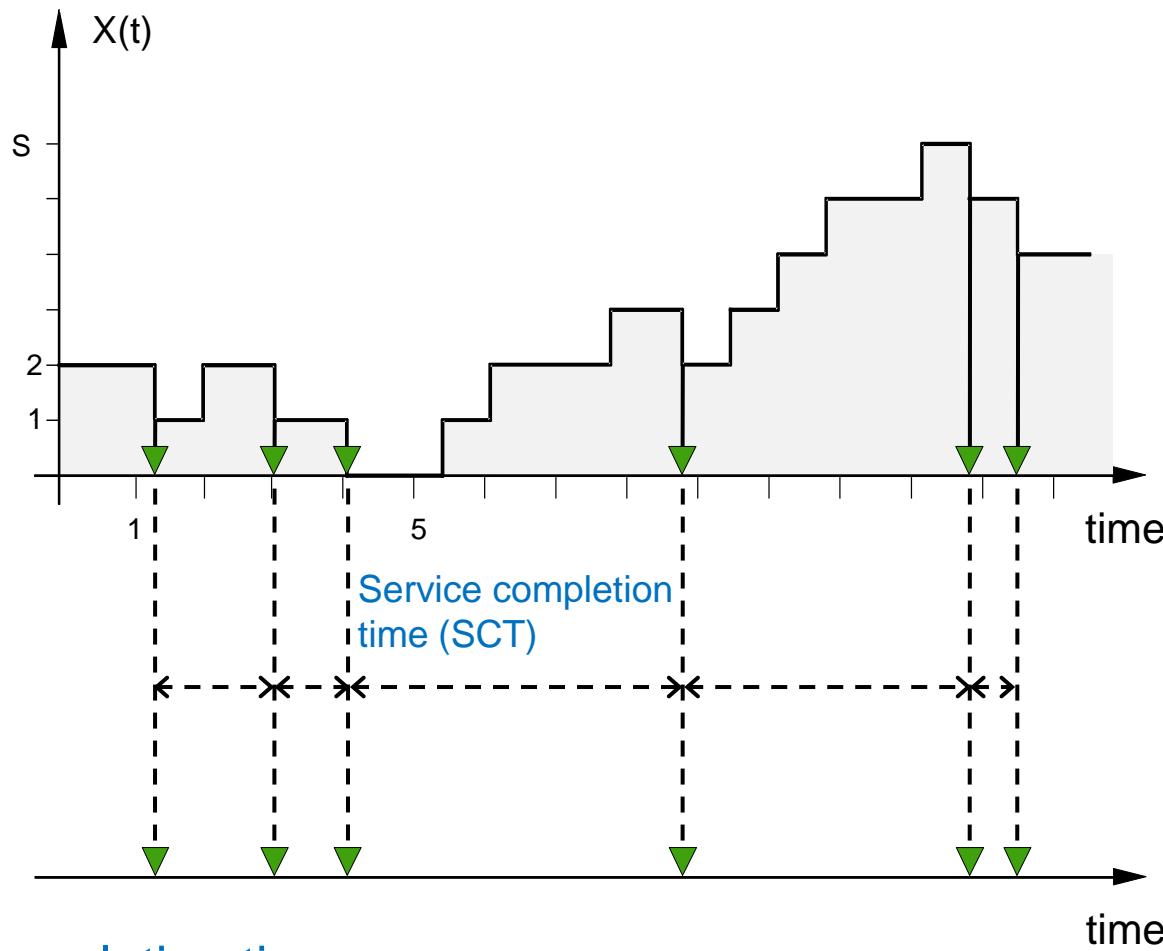


□ Inter-arrival time: Time between consecutive arrival events



Discrete Event Simulation

- Service completion events:



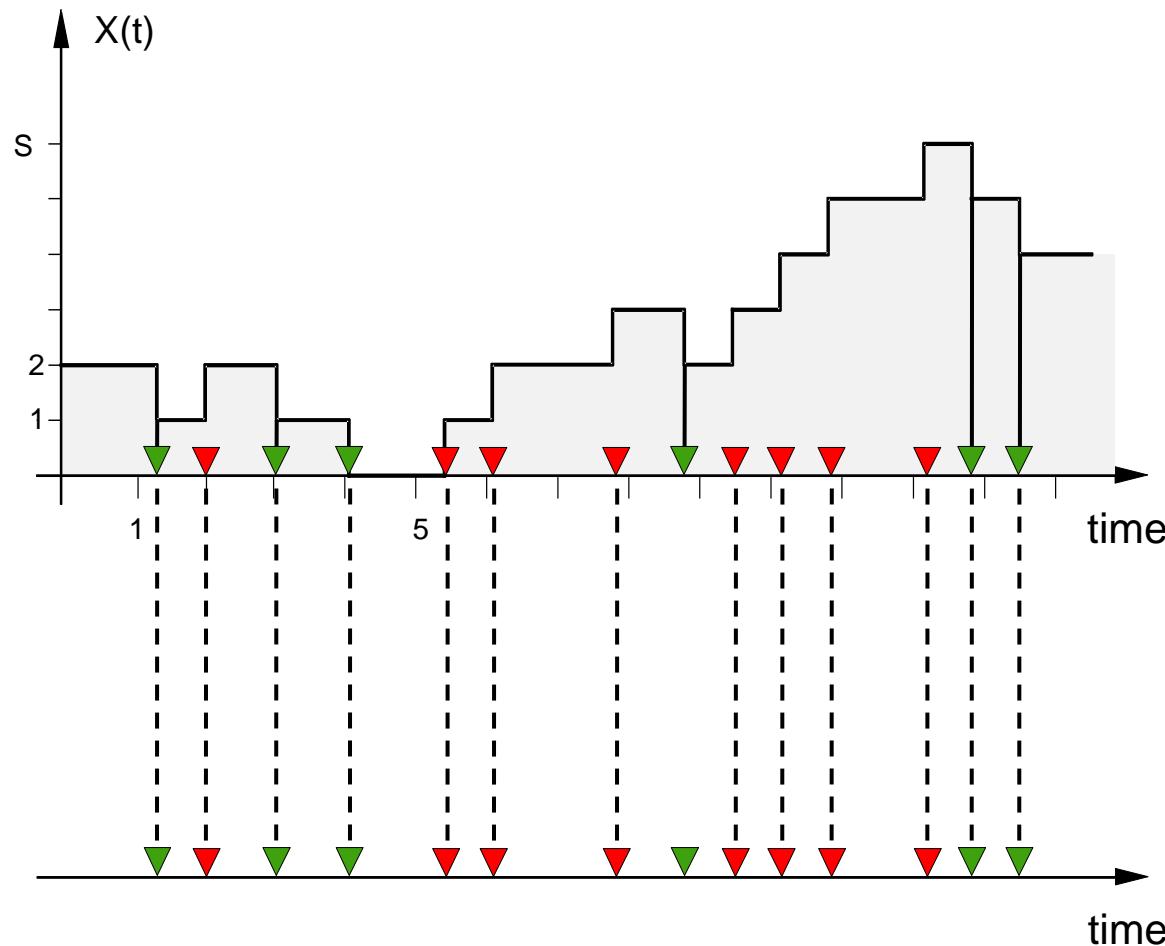
- Service completion time:

Time between consecutive service completion events.



Discrete Event Simulation

□ Event queue:

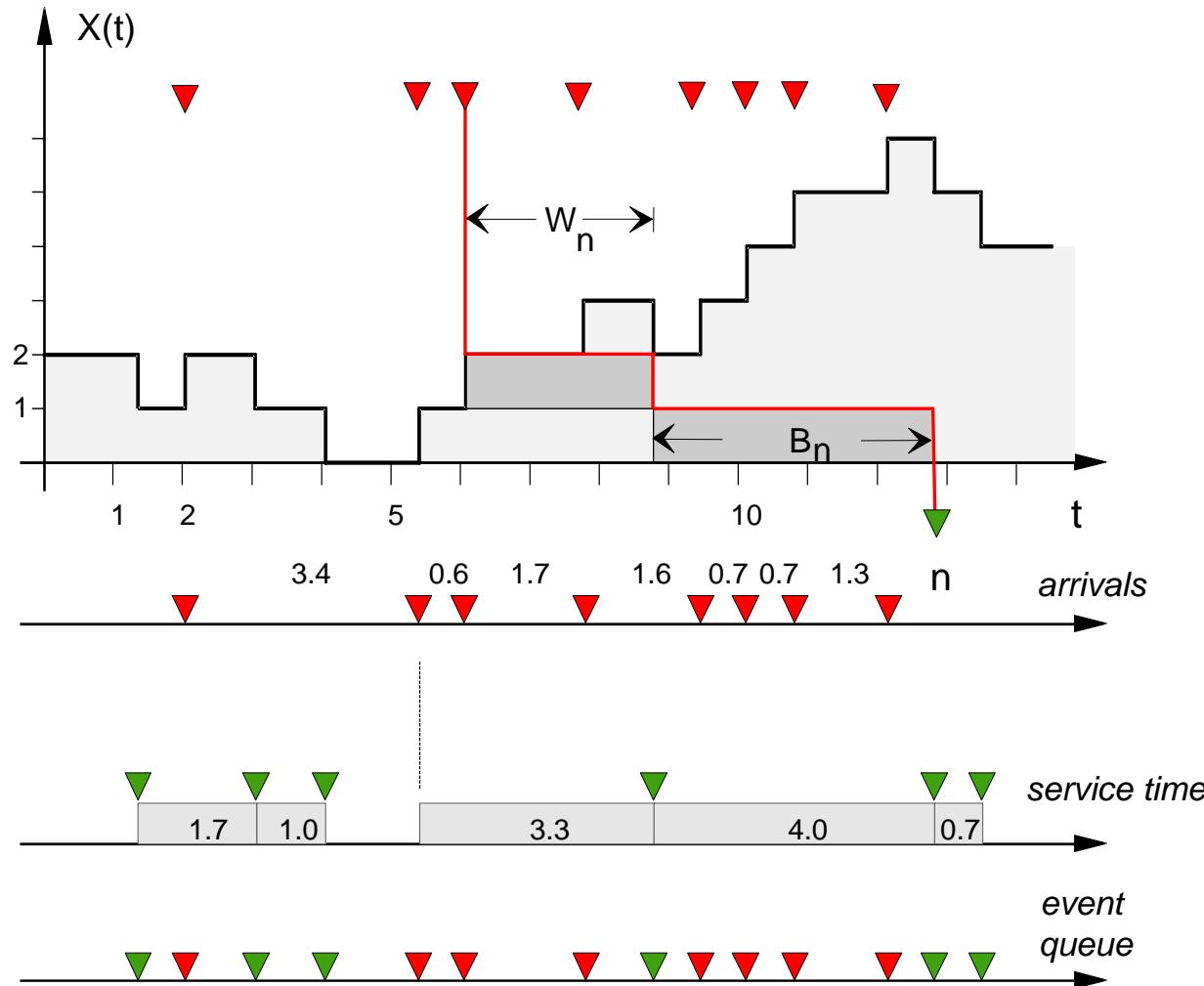


Event queue is a dynamic list of events which is executed in sequential order.



Discrete Event Simulation

□ Event queue:





What's inside a DES? (1/2: data)

- **Simulated time:** internal (to simulation program) variable that keeps track of simulated time
 - May progress in huge jumps (e.g., 1ms, then 20s, then 2ms,...)
 - Not related to real time or CPU time in any way!
- **System state:** variables maintained by simulation program define system state, e.g.: number of packets in queue, current routing table of a router, TCP timeout timers, ...
- **Events:** points in time when system may changes state
 - Each event has an associate **event time**
 - e.g., arrival of packet at a router, departure from the router
 - precisely at these points in time, the simulation must take action (i.e., change state and maybe come up with new future events)
 - Model for time between events (probabilistic) caused by external environment
- **Event queue:** dynamic list of events (→later slides)
- **Statistical counters:** used for observing the system



What's inside a DES? (2/2: program code)

- **Timing routine:**
 - determines the next event and
 - moves the simulation clock to the next event time
- **Event routine:** “process the event”, i.e., change the system state when an event happens
 - One subroutine per event type
- **[P]RNG library routines:** generate random numbers
- **Report generators:** compute performance parameters from statistical counters and generate a report. Runs at simulation end, at interesting events, and/or or at specific pseudo-events
- **Main program:**

```
while(simulation_time < end_time)
{
    next_event = getNextEvent();
    next_event.process();
}
```



Pure event-oriented simulation is challenging

- One event can be composed of a complicated sequence of many actions:
 - Web client sends HTTP request
 - HTTP request encapsulated into TCP frame
 - TCP frame encapsulated into IP frame
 - IP frame encapsulated into Ethernet frame
 - Put frame into **queue** of outgoing interface
- Even more complicated: Many complicated events (receiving request, sending back answer etc.) that are **correlated**

**one
single
event!**

(if we neglect
simulating
CPU time)



Pure event oriented simulation is challenging

- One event can be composed of a complicated sequence of many actions:
 - Web client sends HTTP request
 - HTTP request encapsulated into TCP frame
 - TCP frame encapsulated into IP frame
 - IP frame encapsulated into Ethernet frame
 - Put frame into queue of outgoing interface
 - Even more complicated: Many complicated events (receiving request, sending back answer etc.) that are correlated
 - Problem #1: Event-based programming doesn't look like normal programming at all!
 - Problem #2: Prone to create spaghetti code!?
- one
single
event!**
(if we neglect
simulating
CPU time)



Solution: Process-oriented simulation

- What is a process? (...in the context of simulation)
 - A body of code
 - Variables allocated to that code
 - Current point of execution in the code
 - ⇒ Not much different from a process in an OS
- How is it used?
 - A process groups sets of related events together
 - A process can execute and then be suspended.
Important use cases:
 - Simulation time elapses (e.g., simulate propagation delays)
 - Interactions with other processes that temporarily block (e.g., blocking system calls)
 - Internally, all this is translated into series of events without the programmer noticing it



Applying processes

Two alternative approaches:

One resource = one process

- ❑ Examples: One process for each simulated...:
 - CPU
 - Hard disk
 - Network interface
 - User
- ❑ Jobs using these services (e.g., simulated WWW client program)...
 - are data structures
 - are passed from process to process

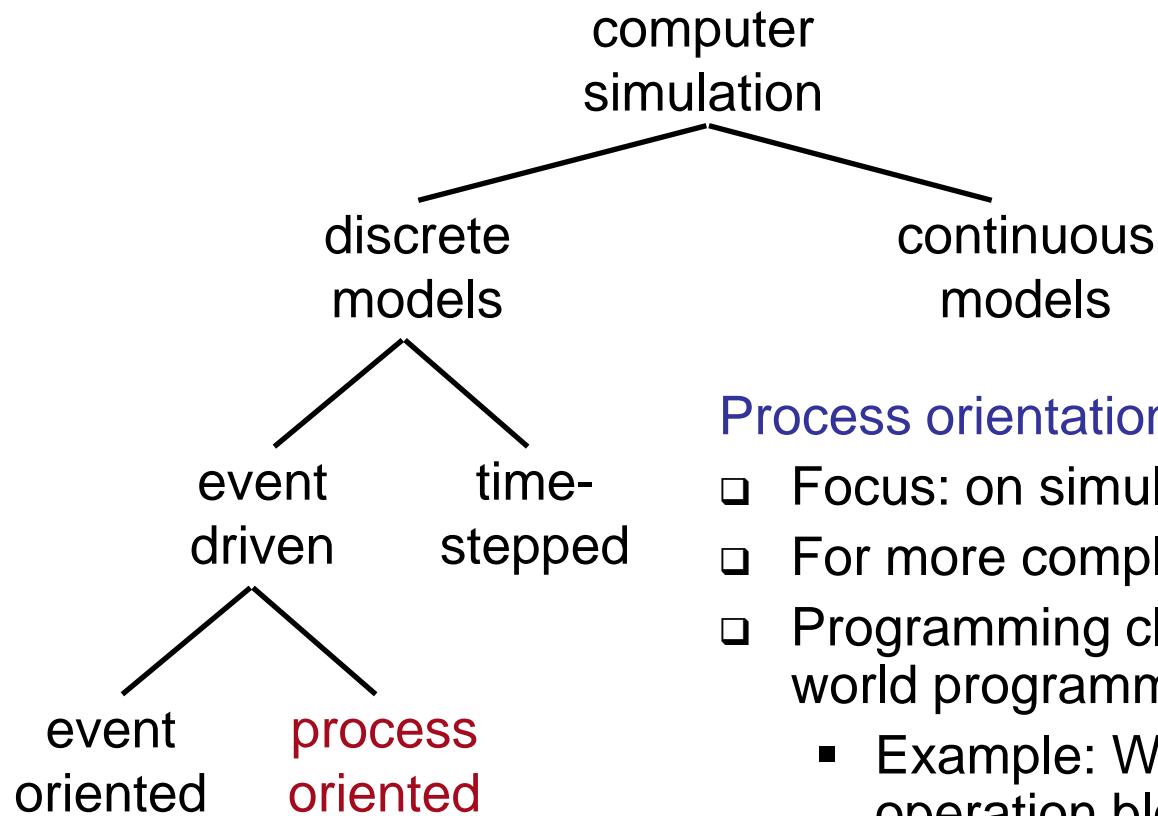
One job = one process

- ❑ Examples: One process for each simulated...:
 - WWW client program
 - WWW server program
 - Peer-to-peer client program
- ❑ Resources used by these jobs (e.g., simulated network interface)...
 - are global variables / data structures

Which approach is better? — It depends!



Simulation Taxonomy



Process orientation:

- Focus: on simulated objects
- For more complex systems
- Programming closer to real-world programming
 - Example: Write into socket; operation blocks
- Usually own simulation language/software (e.g. OPNET)
- Internally translated into sequence of events



Overview: Event orientation↔process orientation

□ Event-oriented simulation:

- Modeler considers one event after the other
- Simulation clock is stopped during event execution
- Rather straightforward to implement
- Often used in non-commercial simulators

□ Process-oriented simulation:

- A process is a ordered series of events related to a certain model object (e.g., customer, job, product)
- Simulation clock moves on during process execution
- Commercial simulators use this approach because of simplified model descriptions
- A process may have several entry points
- In the simulator kernel, the processes are split into events (may be tricky to implement)

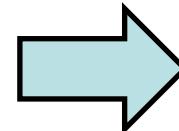


Event list for processes: Usually simpler

Event-oriented simulation

Time	Event
10.0	Take packet P1 out of sender queue and put on link
20.0	Take packet P2 out of sender queue and put on link
27.3	Deliver packet P1 from link to receiver
30.0	Take packet P3 out of sender queue and put on link
37.3	Deliver packet P2 from link to receiver

Process-oriented simulation
(still event-driven!)



Time	Event
10.0	Run queuing process
20.0	Run queuing process
27.3	Run receiver process
30.0	Run queuing process
37.3	Run receiver process



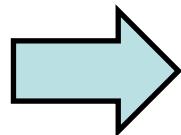
Problem: Simulating blocking behaviour

- Normal programming:

```
result = read(tcp_socket);  
// Blocked until tcp_socket has received some data.  
use(result);
```

- But how should we simulate the blocking character of `read()` in a process-oriented simulator!?

- Blocking call: consume simulation time
- Other events will take place during the time that `read()` is blocked
- In particular: The event that a new packet has arrived, which in turn triggers the return of the `read()` call!
- Obviously, these events must not be blocked



Resuming from returning `read()` is a new event



Other Solutions

- Solution #1: Use threads (??)
 - One thread for process that calls read()
 - One thread for process that moves packet in network
 - One thread for ...
 - Problem: Integration with event concept is difficult
 - Problem: Synchronisation of threads
 - All threads need to access the event list
 - Events must be ordered by time
 - Once some thread has processed an event at time t , then no other thread must generate any event at a time $t' < t$
 - [N.B.: Parallel simulation on multiple CPUs is a complex task.]
- Solution #2a: Using **continuations**
- Solution #2b: Using coroutines



Continuations

- Normal programming:

```
result = f(parameters);  
use(result);
```

- Continuation-passing style:

```
f(parameters, &callback);  
// callback means in C-like syntax:  
// pointer or reference to function callback  
do_other_stuff(...);  
  
callback(result) {  
    // This is just a normal function.  
    use(result);  
}
```



What is it good for?

- ❑ Normal programming:

```
result = f(parameters);  
// We're blocked until f() returns  
use(result);
```

- ❑ Continuation-passing style:

```
f(parameters, &callback); // & : pointer  
// Will return quickly without blocking.  
// Note that f() does not return any results.  
maybe_do_other_stuff(...);
```

```
f(p, cb) {  
    /* Do some calculations; set internal state flags so that  
    callback(..) is invoked as soon as the state of the  
    current process has been changed by one or more  
    events such that we simulate that "f() returns" */  
}  
  
callback(result) {  
    // A normal function; called to simulate that "f() returns"  
    use(result);  
}
```



Simulating blocking calls with continuations

- ❑ Normal programming:

```
result = read(tcp_socket);  
// Blocked until tcp_socket has received some data.  
answer = parse(result);  
write(tcp_socket, answer);
```

- ❑ Simulator:

```
simulate_read(tcp_socket, &cont);  
  
cont(result) {  
    answer = parse(result);  
    simulate_write(tcp_socket, answer, &cont2);  
}
```



What happens inside of simulate_read()?

- `simulate_read(tcp_socket, &cont);`
- Does `simulate_read()` schedule a new event for wakeup with a pointer to `Cont()`?
 - No, not quite!
 - When does the data arrive?—We don't know yet!
- Solution: During the processing of this event,...
 - `simulate_read()` passes control to other entities (processes); e.g., reader → IP stack → network card → physical link
 - Each of these entities sets state variables which indicate that new data arriving should wake them up.
 - At some point, the event ‘packet received’ is processed. The packet gets handled by the various entities (physical link → network card → IP stack → ...), and at some point, `cont` gets called, and “`read()` returns”



Problem: Source code difficult to read

Normal source code:

```
result = read(socket);
answer = parse(result);
success = write(socket, answer);
if (success == WRITE_OK) {
    blah;
} else if (success == WRITE_FAIL) {
    blubb;
}
```



Coroutines: Generalisation of subroutines

□ Subroutine

- Stateless: local variables always
- Execution always starts at beginning
- Execution always ends at last line or `return` statement
- Returning from subroutine = jump back to calling program context

□ Co-routine

- Can keep state
- Execution resumes from the place where you left
(or at the beginning when called for the 1st time)
- Execution is suspended at `yield` statement (...depends on programming language, of course!) and will resume thereafter
- Depending on definition/language, `yield` can specify a target to jump to (i.e., **not necessarily the caller of the coroutine!**)
- Multiple co-routines calling and yielding to each other
≈ cooperative multitasking



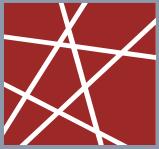
Simula—A forgotten curiosity

- Developed in the 1960s (standards: Simula-I, S.-67, S.-87) by Ole-Johan Dahl and Kristen Nygaard (both †2002)
- ≈ Superset of Algol-60
- Purpose: Process-oriented discrete-event simulation
- An underrated pioneering language:
 - The 1st language that introduced coroutines
 - The 1st language that introduced object-oriented programming!
 - Classes
 - Objects
 - Virtual method calls (dynamic binding)
 - Inheritance
 - Some Simula keywords still used today: `class`, `new`, `this`
 - Garbage collection (idea taken from Lisp, 1950s)



Summary of Introduction (Chapters 0 and 1)

- System, model, observer, simulation
- Why and why not to simulate
- Typical workflow / important aspects in a simulation study
 - Verify that your model makes sense
 - Verify the output of your simulation is not just random noise
 - Remember: trash in \Rightarrow trash out!
- Simulation taxonomy
 - Continuous \leftrightarrow discrete
 - Time-based \leftrightarrow event-based
 - Event-driven \leftrightarrow process-driven
- What is inside a discrete event simulator
 - Event list, sorted by time
 - Simulation time counter
 - State variables
 - Event processing: changes state, but consumes no time



Discrete Event Simulation

IN2045

Chapter 1.5 – How to build a DES

Dr. Alexander Klein

Stephan Günther

Prof. Dr.-Ing. Georg Carle

Chair for Network Architectures and Services
Department of Computer Science
Technische Universität München
<http://www.net.in.tum.de>

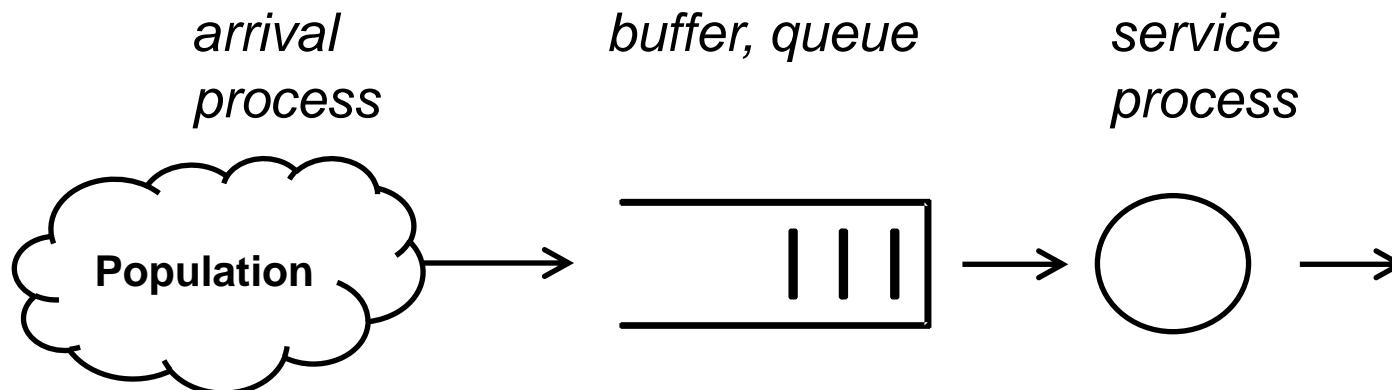
Some of today's
slides/figures are
adopted from
Law&Kelton





Discrete Event Simulation – Queuing Systems

□ System Characteristics:



- | | | |
|--|--|--|
| <ul style="list-style-type: none">• Finite / Infinite• Entity size• Deadline | <ul style="list-style-type: none">• Single queue / Multiple queues• Finite / Infinite capacity• Queuing strategy (FIFO, LIFO...) | <ul style="list-style-type: none">• Single processor / multiple processor• State-(in)dependent• Vacation |
|--|--|--|

□ Performance Characteristics:

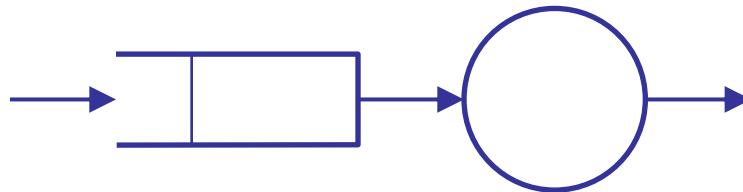
- Average/maximum customer waiting time
- Average/maximum processing time of a customer
- Average/maximum retention time of a customer
- Average/maximum number of customers in the queue
- Customer blocking probability
- Utilization of the system / individual processing units



Discrete Event Simulation – Waiting Queue Example

Queuing model: Input and output

- Input:
 - (Inter-)arrival times of customers (usually random)
 - Job durations (usually random)



- Direct output:
 - Departure times of customers
- Indirect output:
 - Inter-arrival times for departure times of customers
 - Queue length
 - Waiting time in the queue
 - Load of service unit (how often idle, how often working)



Discrete Event Simulation – Waiting Queue Systems

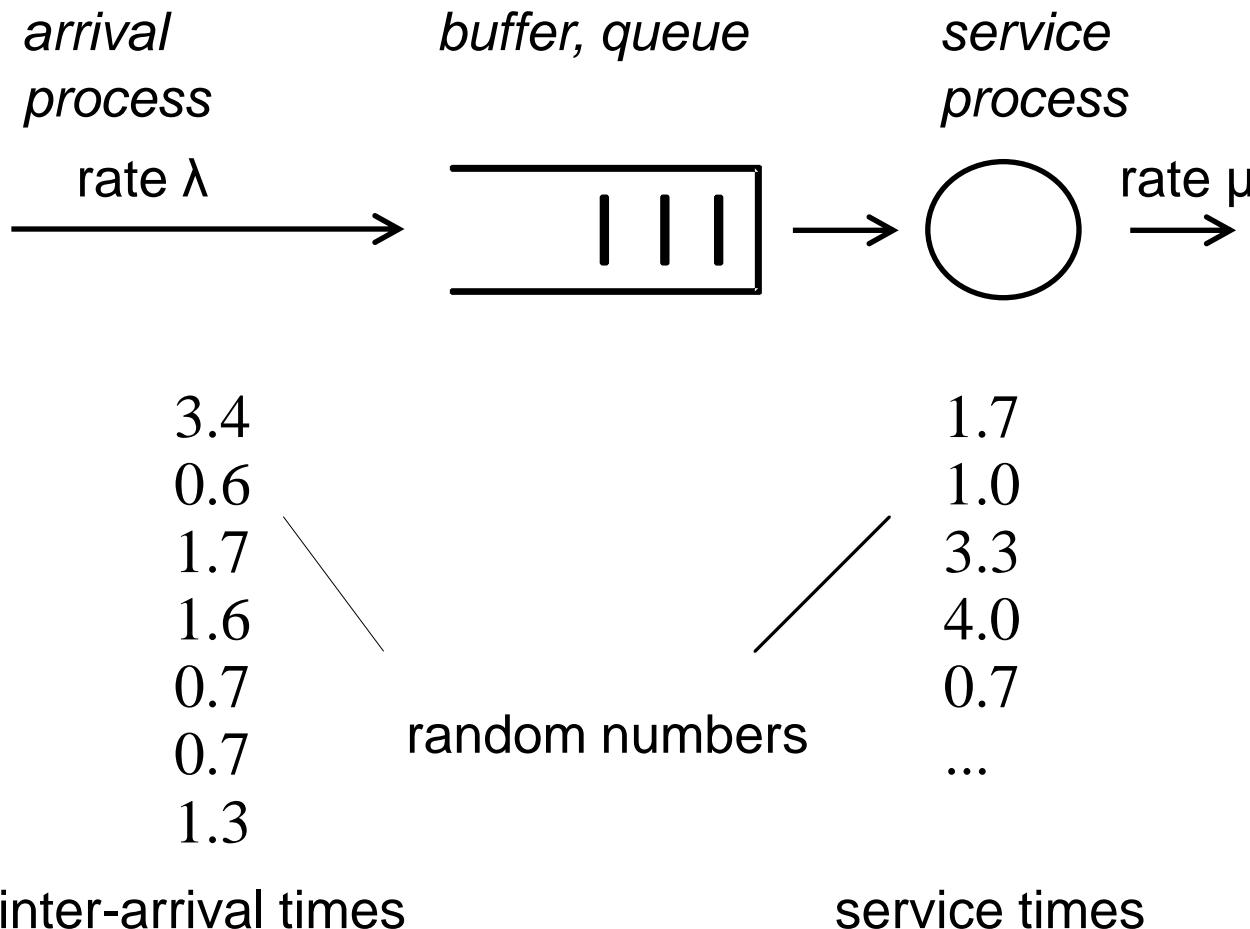
Applications:

System	Entity	Server
Store	Goods	(Lazy) cashier
Manufacturing	Customer order	Machine
Bank	Client	Clerk
Hospital	Patient	Doctor
Computer	Job	CPU
Computer network	Data packet	Radio channel
Cache	Content	Storage



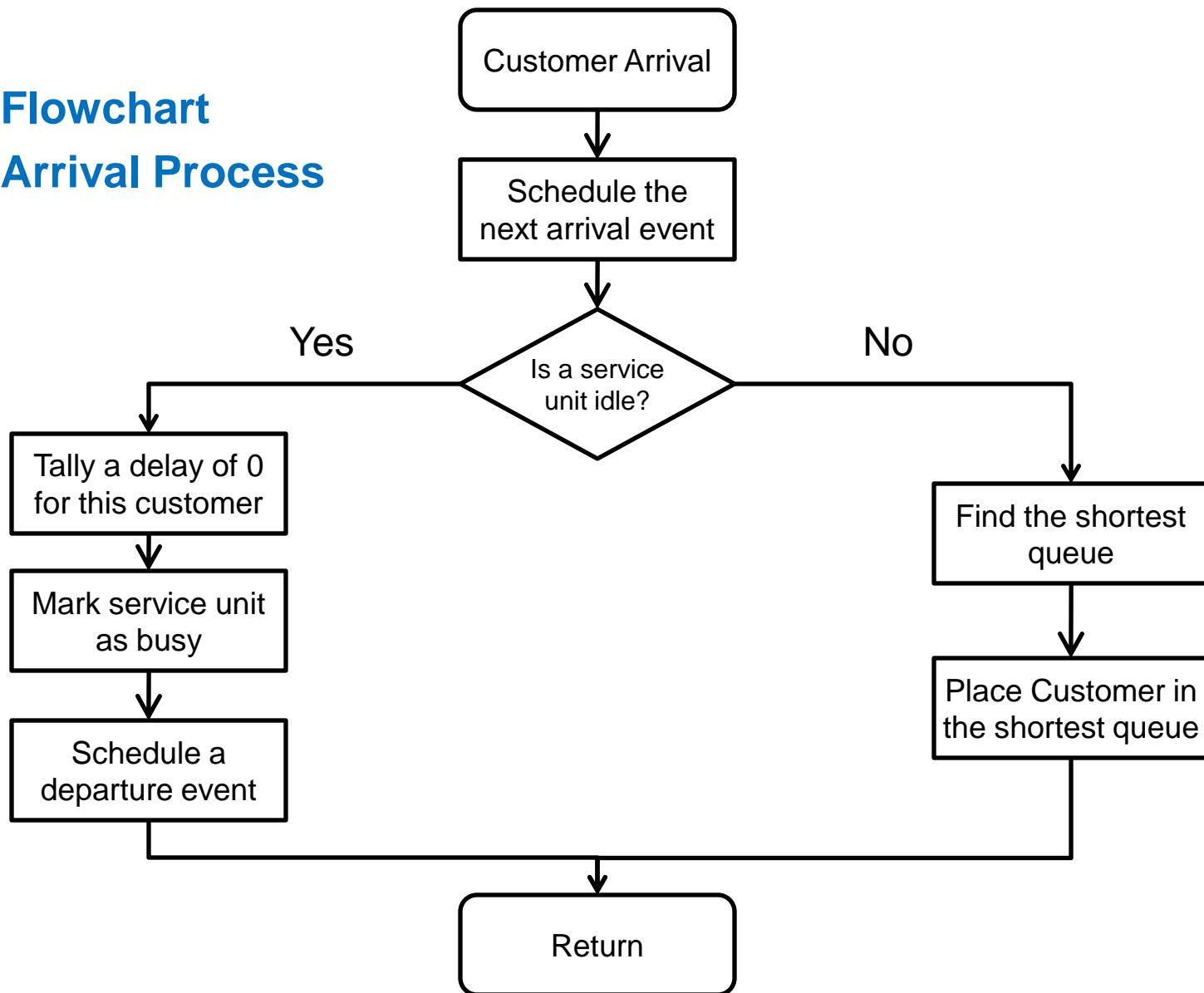
Discrete Event Simulation

Waiting Queue Theory



Discrete Event Simulation – Waiting Queue Example

Flowchart Arrival Process

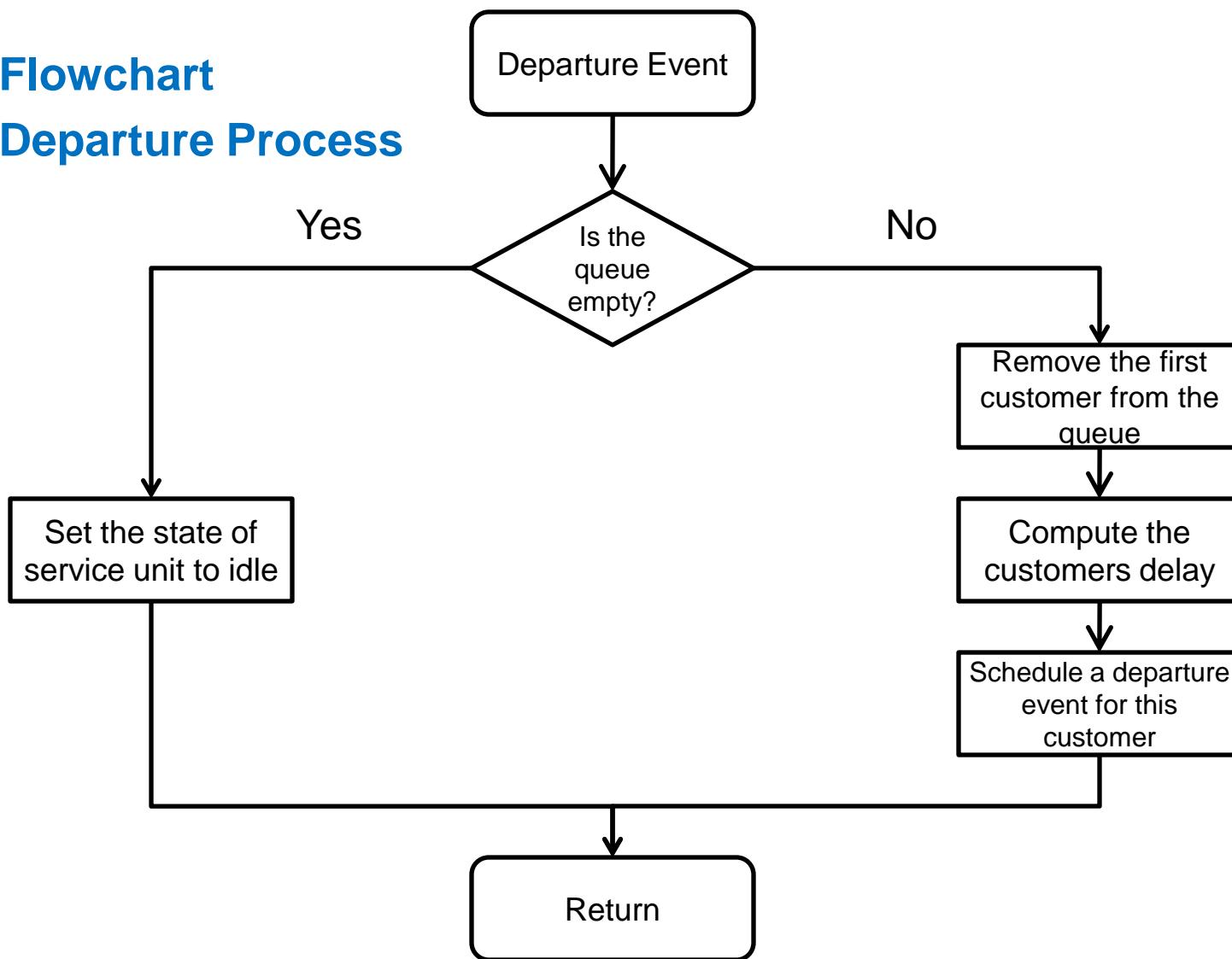




Discrete Event Simulation – Waiting Queue Example

Flowchart

Departure Process





Discussion

- How to build a Discrete Event Simulator?
- What are the necessary modules?
- What are the interesting performance parameters?

Step by Step



Waiting Queue Simulation – Step by Step

□ Events:

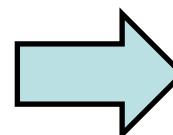
- Customer Arrival Events
- Service Completion Events
- Simulation Termination Event

□ System variables:

- $Q(t)$ – number of waiting customers at time t
- $B(t)$ – number of busy servers at time t

□ Performance parameters:

- Average customer waiting time
- Utilization of the system
- Customer blocking probability



We will need to collect some data to provide these statistics

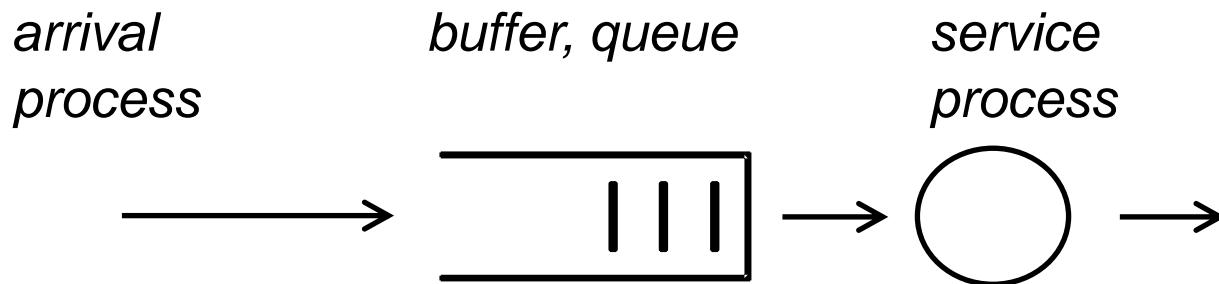


Waiting Queue Simulation – Step by Step

□ Waiting Queue Simulation:

- System with 2 serving units/processors
- Single queue with a capacity of 3 customers
- Simulation duration of 30 simulation ticks
- The first customer is inserted at time 0

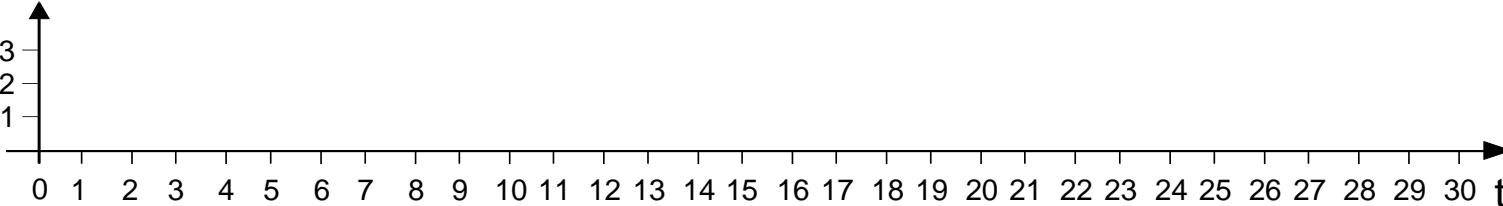
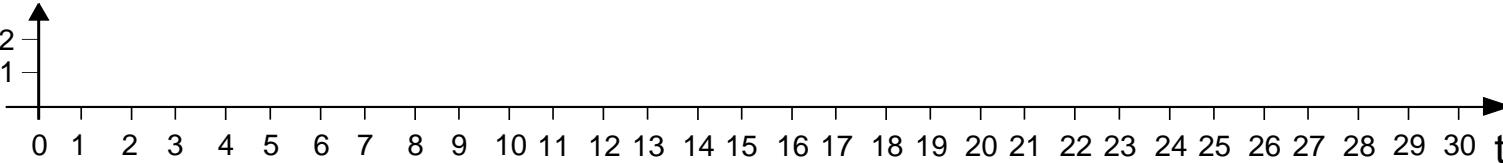
□ System Characteristics:



- | | | |
|---|--|--|
| • General independent inter-arrival times | • Finite capacity – 3 Customers
• Queuing strategy - FIFO | • Multi-processor – 2 processors
• General independent service completion times |
|---|--|--|



Waiting Queue Simulation – Step by Step

Simulation time	Current Event	Process Routine	Number of busy servers
Initialization	none	Insert first customer arrival; Insert simulation termination;	$B(t) = 0$
Event queue	[0;CA], [30;ST]	Time of arrival of customers in queue	Number of waiting customers $Q(t) = 0$
Number of arrived customers = 0 Number of served customers = 0		Number of blocked customers = 0	$\sum_t Q(t) = 0$ $\sum_t B(t) = 0$
Q(t)			t
B(t)			t
Inter-arrival time			
Service completion time			



Waiting Queue Simulation – Step by Step

Simulation time	Current Event	Process Routine	Number of busy servers
0	Customer Arrival [0;CA]	Schedule next arrival-> IAT 2.0 ; Increase $B(t)$; Schedule service completion-> SCT 5.0	$B(t) = 1$
Event queue	Time of arrival of customers in queue	Number of waiting customers	
[2;CA], [5;SC], [30;ST]		$Q(t) = 0$	
Number of arrived customers = 1 Number of served customers = 0	Number of blocked customers = 0	$\sum_t Q(t) = 0$	$\sum_t B(t) = 0$
Inter-arrival time	2.0		
Service completion time	5.0		

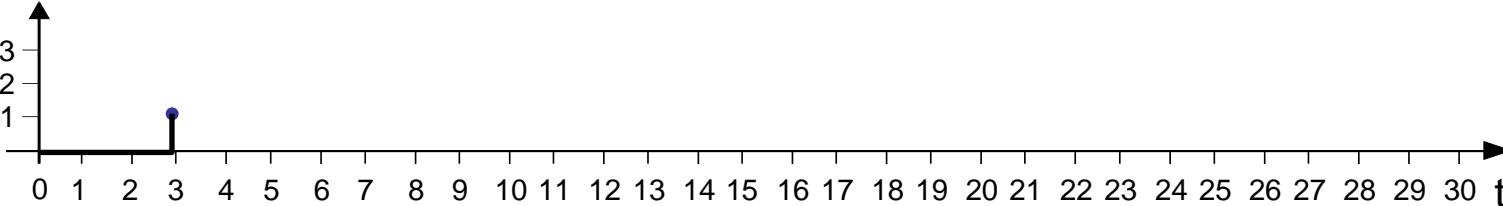
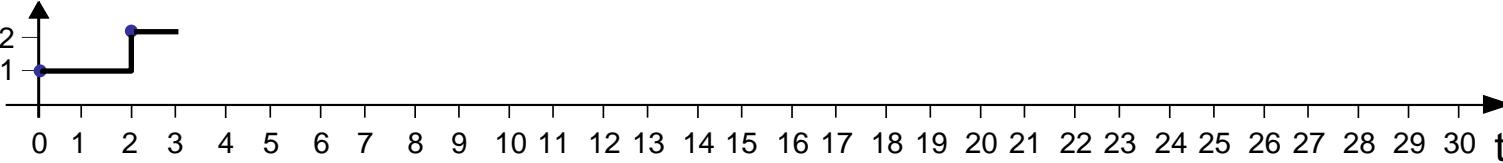


Waiting Queue Simulation – Step by Step

Simulation time	Current Event	Process Routine	Number of busy servers
2	Customer Arrival [2;CA]	Schedule next arrival-> IAT 1.0 ; Increase $B(t)$; Schedule service completion-> SCT 5.0	$B(t) = 2$
Event queue		Time of arrival of customers in queue	Number of waiting customers
	[3;CA], [5;SC], [7;SC], [30;ST]		$Q(t) = 0$
Number of arrived customers = 2 Number of served customers = 0		Number of blocked customers = 0	$\sum_t Q(t) = 0$ $\sum_t B(t) = 2$
<p>Q(t)</p> <p>t</p>			
<p>B(t)</p> <p>t</p>			
Inter-arrival time	2.0	1.0	
Service completion time	5.0	5.0	



Waiting Queue Simulation – Step by Step

Simulation time	Current Event	Process Routine	Number of busy servers
3	Customer Arrival [3;CA]	Schedule next arrival-> IAT 1.0 ; All servers busy->Insert customer in queue	$B(t) = 2$
Event queue [4;CA], [5;SC], [7;SC], [30;ST]		Time of arrival of customers in queue [3;CA]	Number of waiting customers $Q(t) = 1$
Number of arrived customers = 3 Number of served customers = 0		Number of blocked customers = 0	$\sum_t Q(t) = 0$ $\sum_t B(t) = 4$
Q(t)			t
B(t)			t
Inter-arrival time	2.0	1.0	1.0
Service completion time	5.0	5.0	

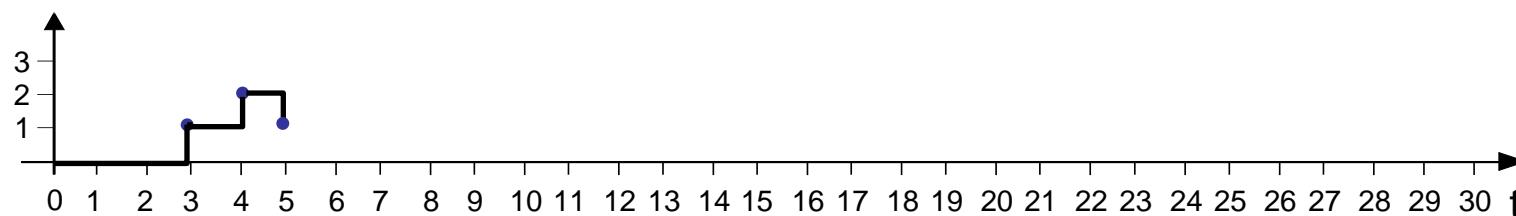
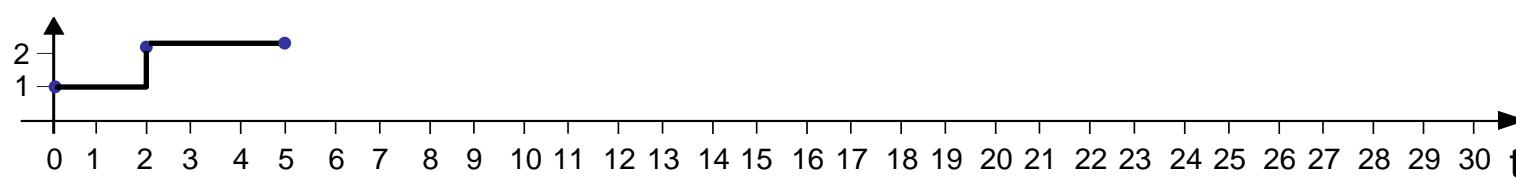


Waiting Queue Simulation – Step by Step

Simulation time	Current Event	Process Routine	Number of busy servers
4	Customer Arrival [4;CA]	Schedule next arrival-> IAT 2.0 ; All servers busy->Insert customer in queue	$B(t) = 2$
Event queue [5;SC], [6;CA] , [7;SC], [30;ST]		Time of arrival of customers in queue [3;CA], [4;CA]	Number of waiting customers
Number of arrived customers = 4 Number of served customers = 0		Number of blocked customers = 0	$\sum_t Q(t) = 1$ $\sum_t B(t) = 6$
 $Q(t)$			
 $B(t)$			
Inter-arrival time	2.0	1.0	1.0
Service completion time	5.0	5.0	



Waiting Queue Simulation – Step by Step

Simulation time	Current Event	Process Routine	Number of busy servers
5	Service Completion [5;SC]	Remove customer from queue; Schedule service completion -> SCT 6.0 ;	$B(t) = 2$
Event queue [6;CA], [7;SC], [11;SC] , [30;ST]	Time of arrival of customers in queue [4;CA]	Number of waiting customers	
Number of arrived customers = 4 Number of served customers = 1	Number of blocked customers = 0	$\sum_t Q(t) = 3$	$\sum_t B(t) = 8$
			
			
Inter-arrival time	2.0	1.0	1.0
Service completion time	5.0	5.0	6.0



Waiting Queue Simulation – Step by Step

Simulation time	Current Event	Process Routine	Number of busy servers
6	Customer Arrival [6;CA]	Schedule next arrival-> IAT 1.0 ; All servers busy->Insert customer in queue	$B(t) = 2$
Event queue		Time of arrival of customers in queue	Number of waiting customers
[7;SC], [7;CA], [11;SC], [30;ST]		[4;CA], [6;CA]	$Q(t) = 2$
Number of arrived customers = 5 Number of served customers = 1		Number of blocked customers = 0	$\sum_t Q(t) = 4$ $\sum_t B(t) = 10$
<p>Q(t)</p> <p>t</p>			
<p>B(t)</p> <p>t</p>			
Inter-arrival time	2.0	1.0	1.0
Service completion time	5.0	5.0	6.0



Waiting Queue Simulation – Step by Step

Simulation time	Current Event	Process Routine	Number of busy servers
7	Customer Arrival / Service Completion		$B(t) = 2$
Event queue		Time of arrival of customers in queue	Number of waiting customers
[7;SC], [7;CA], [11;SC], [30;ST]		[4;CA], [6;CA]	$Q(t) = 2$
Number of arrived customers = X Number of served customers = X		Number of blocked customers = 0	$\sum_t Q(t) = 4$ $\sum_t B(t) = 10$
Q(t)		Which event should be scheduled first? Customer Arrival or Service Completion?	
B(t)			
Inter-arrival time	2.0	1.0	1.0
Service completion time	5.0	5.0	6.0

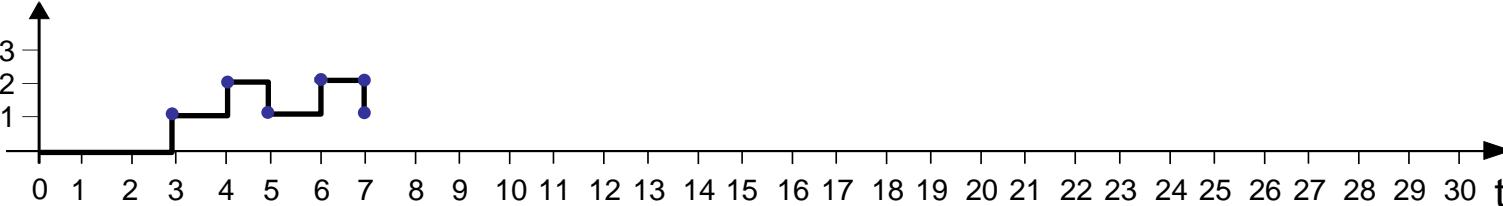
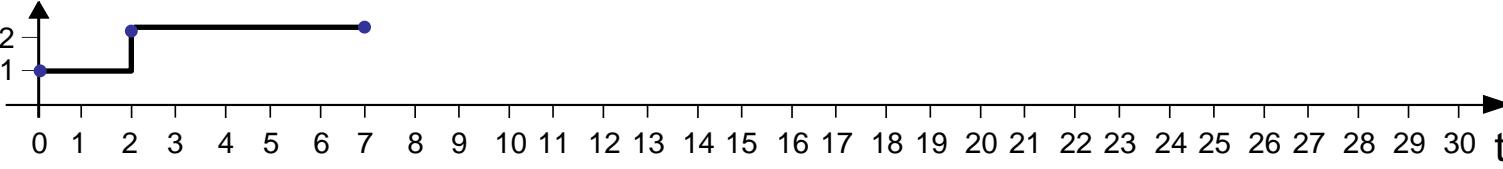


Waiting Queue Simulation – Step by Step

Simulation time	Current Event	Process Routine	Number of busy servers
7	Service Completion [7;SC]	Remove customer from queue; Schedule service completion -> SCT 4.0 ;	$B(t) = 2$
Event queue [7;CA], [11;SC], [11;SC], [30;ST]		Time of arrival of customers in queue [6;CA]	Number of waiting customers $Q(t) = 1$
Number of arrived customers = 5 Number of served customers = 2	Number of blocked customers = 0		$\sum_t Q(t) = 6$ $\sum_t B(t) = 12$
 We assume that the service is completed before the next customer arrives.			
Inter-arrival time	2.0	1.0	1.0
Service completion time	5.0	5.0	6.0

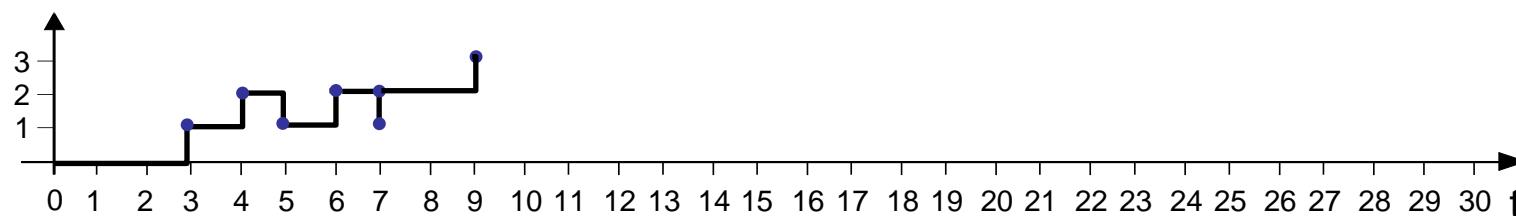
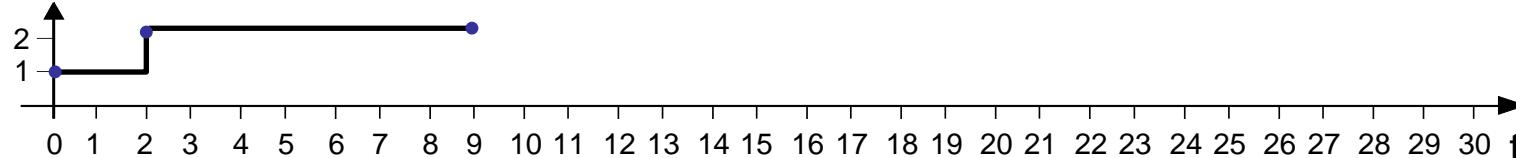


Waiting Queue Simulation – Step by Step

Simulation time	Current Event	Process Routine	Number of busy servers
7	Customer Arrival [7;CA]	Schedule next arrival-> IAT 2.0 ; All servers busy->Insert customer in queue	$B(t) = 2$
Event queue [9;CA], [11;SC], [11;SC], [30;ST]		Time of arrival of customers in queue [6;CA], [7;CA]	Number of waiting customers $Q(t) = 2$
Number of arrived customers = 6 Number of served customers = 2	Number of blocked customers = 0		$\sum_t Q(t) = 6$ $\sum_t B(t) = 12$
			
			
Inter-arrival time	2.0	1.0	1.0
Service completion time	5.0	5.0	6.0



Waiting Queue Simulation – Step by Step

Simulation time	Current Event	Process Routine	Number of busy servers
9	Customer Arrival [9;CA]	Schedule next arrival-> IAT 1.0 ; All servers busy->Insert customer in queue	$B(t) = 2$
Event queue [10;CA], [11;SC], [11;SC], [30;ST]		Time of arrival of customers in queue [6;CA], [7;CA], [9;CA]	Number of waiting customers $Q(t) = 3$
Number of arrived customers = 7 Number of served customers = 2	Number of blocked customers = 0		$\sum_t Q(t) = 10$ $\sum_t B(t) = 16$
 <p>Q(t)</p> <p>t</p>			
 <p>B(t)</p> <p>t</p>			
Inter-arrival time	2.0	1.0	1.0
Service completion time	5.0	5.0	6.0



Waiting Queue Simulation – Step by Step

Simulation time	Current Event	Process Routine	Number of busy servers
10	Customer Arrival [10;CA]	Schedule next arrival-> IAT 4.0 ; All servers busy->Queue full ->Block customer	$B(t) = 2$
Event queue [11;SC], [11;SC], [14;CA], [30;ST]		Time of arrival of customers in queue [6;CA], [7;CA], [9;CA]	Number of waiting customers $Q(t) = 3$
Number of arrived customers = 8 Number of served customers = 2		Number of blocked customers = 1	$\sum_t Q(t) = 13$ $\sum_t B(t) = 18$
<p>Graph of $Q(t)$ vs t. The queue length starts at 0, jumps to 1 at $t=3$, reaches 2 at $t=4$, stays at 2 until $t=9$, and then jumps to 3 at $t=10$.</p>			
<p>Graph of $B(t)$ vs t. The number of busy servers starts at 1, jumps to 2 at $t=2$, and remains at 2 until $t=10$.</p>			
Inter-arrival time	2.0	1.0	1.0
Service completion time	5.0	5.0	6.0



Waiting Queue Simulation – Step by Step

Simulation time	Current Event	Process Routine	Number of busy servers
11	Service Completion [11;SC]	Remove customer from queue; Schedule service completion -> SCT 4.0 ;	$B(t) = 2$
Event queue [15;SC], [14;CA], [30;ST]		Time of arrival of customers in queue [7;CA], [9;CA]	Number of waiting customers $Q(t) = 2$
Number of arrived customers = 8 Number of served customers = 3		Number of blocked customers = 1	$\sum_t Q(t) = 16$ $\sum_t B(t) = 20$
<p>Graph of $Q(t)$ vs t. The queue length increases at discrete time points: (0,1), (1,1), (2,1), (3,2), (4,2), (5,1), (6,2), (7,1), (9,3), (10,3), (11,2).</p>			
<p>Graph of $B(t)$ vs t. The number of busy servers increases at $t=2$ from 1 to 2 and remains constant until $t=11$.</p>			
Inter-arrival time	2.0	1.0	1.0
Service completion time	5.0	5.0	6.0



Waiting Queue Simulation – Step by Step

Simulation time	Current Event	Process Routine	Number of busy servers
11	Service Completion [11;SC]	Remove customer from queue; Schedule service completion -> SCT 6.0 ;	$B(t) = 2$
Event queue [14;CA], [15;SC], [17;SC] , [30;ST]		Time of arrival of customers in queue [9;CA]	Number of waiting customers $Q(t) = 1$
Number of arrived customers = 8 Number of served customers = 4		Number of blocked customers = 1	$\sum_t Q(t) = 16$ $\sum_t B(t) = 20$
<p>Graph of $Q(t)$ vs t. The queue length increases at discrete time points: (0,1), (1,1), (2,1), (3,2), (4,2), (5,1), (6,2), (7,1), (9,3), (10,3), (11,2).</p>			
<p>Graph of $B(t)$ vs t. The number of busy servers increases at $t=2$ from 1 to 2 and remains constant until $t=11$.</p>			
Inter-arrival time	2.0	1.0	1.0
Service completion time	5.0	5.0	6.0



Waiting Queue Simulation – Step by Step

Simulation time	Current Event	Process Routine	Number of busy servers
14	Customer Arrival [14;CA]	Schedule next arrival-> IAT 8.0 ; All servers busy->Insert customer in queue	$B(t) = 2$
Event queue [15;SC], [17;SC], [22;CA] , [30;ST]		Time of arrival of customers in queue [9;CA], [14;CA]	Number of waiting customers $Q(t) = 2$
Number of arrived customers = 9 Number of served customers = 4		Number of blocked customers = 1	$\sum_t Q(t) = 19$ $\sum_t B(t) = 26$
<p>Q(t)</p> <p>t</p>			
<p>B(t)</p> <p>t</p>			
Inter-arrival time	2.0	1.0	1.0
Service completion time	5.0	5.0	6.0



Waiting Queue Simulation – Step by Step

Simulation time	Current Event	Process Routine	Number of busy servers
15	Service Completion [15;SC]	Remove customer from queue; Schedule service completion -> SCT 3.0 ;	$B(t) = 2$
Event queue [17;SC], [18;SC] , [22;CA], [30;ST]		Time of arrival of customers in queue [14;CA]	Number of waiting customers $Q(t) = 1$
Number of arrived customers = 9 Number of served customers = 5		Number of blocked customers = 1	$\sum_t Q(t) = 21$ $\sum_t B(t) = 28$
<p>Graph of $Q(t)$ vs t. The queue length starts at 1, jumps to 2 at $t=3$, stays at 2 until $t=5$, jumps to 3 at $t=9$, stays at 3 until $t=10$, then drops to 2 at $t=11$, stays at 2 until $t=14$, then drops to 1 at $t=15$.</p>			
<p>Graph of $B(t)$ vs t. The number of busy servers starts at 1, jumps to 2 at $t=2$, stays at 2 until $t=15$.</p>			
Inter-arrival time	2.0	1.0	1.0
Service completion time	5.0	5.0	6.0



Waiting Queue Simulation – Step by Step

Simulation time	Current Event	Process Routine	Number of busy servers
17	Service Completion [17;SC]	Remove customer from queue; Schedule service completion -> SCT 3.0 ;	$B(t) = 2$
Event queue [18;SC], [20;SC], [22;CA], [30;ST]		Time of arrival of customers in queue	Number of waiting customers
Number of arrived customers = 9 Number of served customers = 6		Number of blocked customers = 1	$\sum_t Q(t) = 23$ $\sum_t B(t) = 32$
<p>Q(t)</p> <p>t</p>			
<p>B(t)</p> <p>t</p>			
Inter-arrival time	2.0	1.0	2.0
Service completion time	5.0	5.0	3.0



Waiting Queue Simulation – Step by Step

Simulation time	Current Event	Process Routine	Number of busy servers
18	Service Completion [18;SC]	Set one server to idle	$B(t) = 1$
Event queue [20;SC], [22;CA], [30;ST]		Time of arrival of customers in queue	Number of waiting customers
Number of arrived customers = 9 Number of served customers = 7		Number of blocked customers = 1	$\sum_t Q(t) = 23$ $\sum_t B(t) = 34$
<p>Q(t)</p> <p>t</p>			
<p>B(t)</p> <p>t</p>			
Inter-arrival time	2.0	1.0	1.0
Service completion time	5.0	5.0	6.0

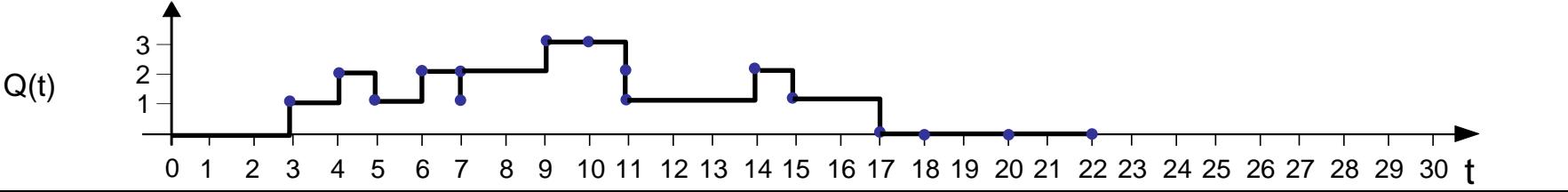
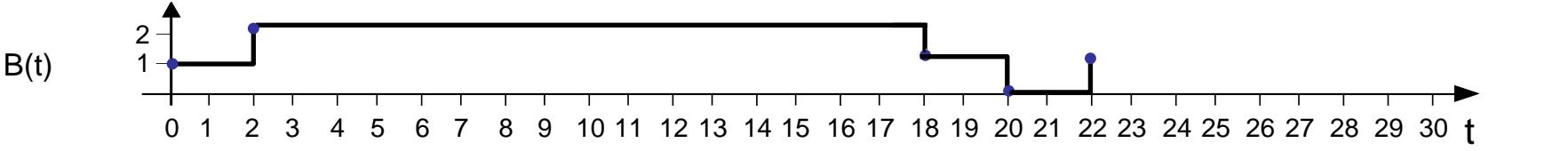


Waiting Queue Simulation – Step by Step

Simulation time	Current Event	Process Routine	Number of busy servers
20	Service Completion [20;SC]	Set server to idle	$B(t) = 0$
Event queue [22;CA], [30;ST]		Time of arrival of customers in queue	Number of waiting customers
Number of arrived customers = 9 Number of served customers = 8		Number of blocked customers = 1	$\sum_t Q(t) = 23$ $\sum_t B(t) = 36$
 $Q(t)$ vs t			
 $B(t)$ vs t			
Inter-arrival time	2.0	1.0	1.0
Service completion time	5.0	5.0	6.0

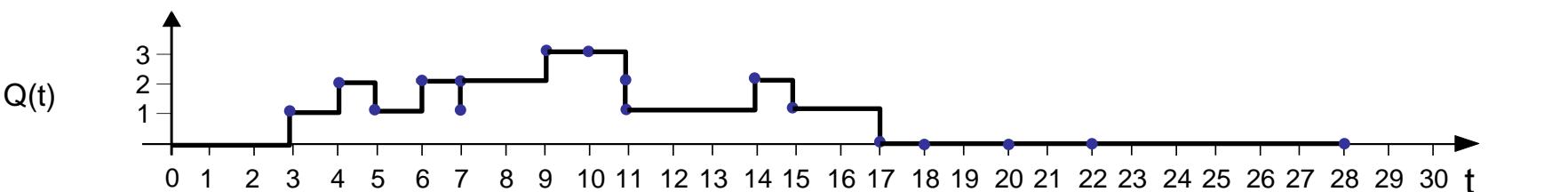
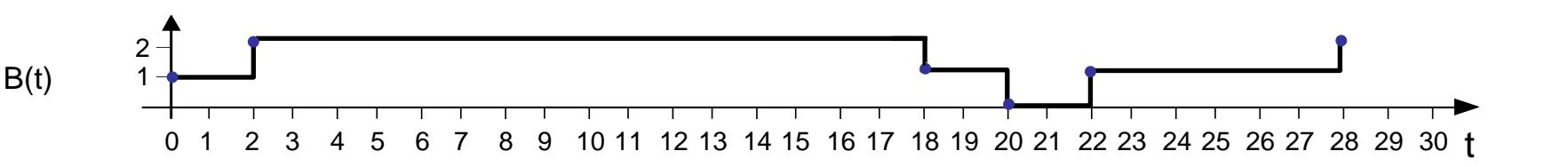


Waiting Queue Simulation – Step by Step

Simulation time	Current Event	Process Routine	Number of busy servers
22	Customer Arrival [22;CA]	Schedule next arrival-> IAT 6.0 ; Set server to busy; Schedule service completion-> SCT 9.0 ;	$B(t) = 1$
Event queue		Time of arrival of customers in queue	Number of waiting customers
	[28;CA], [30;ST], [31;SC]		$Q(t) = 0$
Number of arrived customers = 10 Number of served customers = 8		Number of blocked customers = 1	$\sum_t Q(t) = 23$ $\sum_t B(t) = 36$
			
			
Inter-arrival time	2.0	1.0	1.0
Service completion time	5.0	5.0	6.0
	4.0	6.0	3.0
	2.0	1.0	4.0
	8.0		
	6.0		



Waiting Queue Simulation – Step by Step

Simulation time	Current Event	Process Routine	Number of busy servers
28	Customer Arrival [28;CA]	Schedule next arrival-> IAT 3.0 ; Set server to busy; Schedule service completion-> SCT 4.0 ;	$B(t) = 2$
Event queue		Time of arrival of customers in queue	Number of waiting customers
	[30;ST], [31;SC] , [31;CA] , [32;SC]		$Q(t) = 0$
Number of arrived customers = 11 Number of served customers = 8		Number of blocked customers = 1	$\sum_t Q(t) = 23$ $\sum_t B(t) = 42$
			
			
Inter-arrival time	2.0	1.0	1.0
Service completion time	5.0	5.0	6.0



Waiting Queue Simulation – Step by Step

Simulation time	Current Event	Process Routine	Number of busy servers
30	Simulation Termination [30;ST]	End simulation->exit while loop	$B(t) = 2$
Event queue		Time of arrival of customers in queue	Number of waiting customers
[31;SC] , [31;CA], [32;SC]			$Q(t) = 0$
Number of arrived customers = 11 Number of served customers = 8		Number of blocked customers = 1	$\sum_t Q(t) = 23$ $\sum_t B(t) = 46$
Inter-arrival time	2.0	1.0	1.0
Service completion time	5.0	5.0	6.0



Waiting Queue Simulation – Step by Step

□ Statistics:

- Simulation duration: = 30 ticks
- Number of **arrived** customers = 11
- Number of **served** customers = 8
- Number of **blocked** customers = 1

- Sum of waiting time $\sum_t Q(t) = 23$
- Sum of server utilization $\sum_t B(t) = 46$

- Inter-arrival times 2.0;1.0;1.0;2.0;1.0;2.0;1.0;4.0;8.0;6.0

- Service-completion times 5.0;5.0;6.0;4.0;6.0;3.0;3.0;9.0;4.0



Only executed events are considered



Waiting Queue Simulation – Step by Step

□ Statistics:

- Average waiting time =
$$\frac{\text{Sum of waiting time}}{\text{Number of customer arrivals}} = \frac{23}{11}$$

- Average server utilization =
$$\frac{\text{Sum of server utilization}}{\text{Simulation duration} * \#Servers} = \frac{46}{2*30}$$

- Average Inter-arrival time =
$$\frac{\text{Sum of IAT}}{\text{Number of customer arrivals} - 1} = \frac{28}{10}$$

- Service-completion times =
$$\frac{\text{Sum of SCT}}{\text{Number of service completions}} = \frac{45}{9}$$



How to build a Discrete Event Simulator

Implementation



What are the necessary modules / classes?

- **DES** – Main class
- **EventChain** – Event queue
- **SortableQueue** – Part of the event chain
- **SortableQueueItem** – Represents a queued element (simulation time)
- **SimEvent** – Extends SortableQueueItem and provides a process routine
- **CustomerArrival / ServiceCompletion / SimulationTermination**
 - Extends SimEvent
- **SimState** – Holds information about the current system state



What are the necessary modules?

□ DES

- Main class
- Initializes the simulation
- Consists of a while loop in which events are processed

```
// Initialization Code  
...  
SimEvent = null;  
  
while ((e != TerminationEvent) && (stop != true))  
{  
    e = getNextEvent(e);  
    e.process();  
  
    // Collect Statistics  
    ...  
}
```

Pseudo Code - Example



What are the necessary modules?

- **SortableQueueItem**
 - Simple object which holds a parameter that represents the simulation time
- **SimEvent**
 - Extends SortableQueueItem
 - Contains a process routing which manipulates the system
(e.g. generates new events or increases / decreases the queue length)
 - **!!! Should contain a pointer to the current (static) SimState !!!**



What are the necessary modules?

□ CustomerArrival

- Extends SimEvent
- Process routine should change the system as follows:
 - In the case that all servers are busy:
 - Increase the waiting queue length
 - Create a new CustomerArrival event and insert it in the EventQueue
 - Otherwise:
 - Create a new CustomerArrival event and insert it in the EventQueue
 - Create a new ServiceCompletion event and insert it in the EventQueue
 - Set an idle server to busy state



What are the necessary modules?

□ ServiceCompletion

- Extends SimEvent
- Process routine should change the system as follows:
 - In the case that the queueSize is larger than 0:
 - Decrease the queueSize by one
 - Create a new ServiceCompletion event and insert it in the EventQueue
 - Otherwise:
 - Set the state of a busy server to idle



What are the necessary modules?

□ SimulationTermination

- Extends SimEvent
- Process routine should change the system as follows:
 - Set the simulation state to stop



What are the necessary modules?

□ SimState

- Holds all information of the current simulation state

Parameters:

- | | |
|---|---|
| ▪ <code>EventChain</code> ec | - Holds all pending SimEvents |
| ▪ <code>long</code> now | - The current simulation time |
| ▪ <code>boolean</code> stop | - Indicates the termination of the simulation |
| ▪ <code>long</code> queueSize | - Current size of the waiting queue |
| ▪ <code>boolean</code> serverBusy | - Represents the server state |
| ▪ <code>long</code> interArrivalTime | - Time between customer arrivals |
| ▪ <code>long</code> serviceCompletionTime | - Time service completions |

Statistics:

- | | |
|-------------------------|------------------------------|
| ▪ <code>long</code> min | - Minimum waiting queue size |
| ▪ <code>long</code> max | - Maximum waiting queue size |