

Introduction to Modeling & Simulation

the nature of simulation and the M&S lifecycle

The nature of Simulation

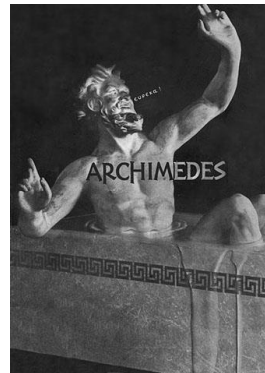
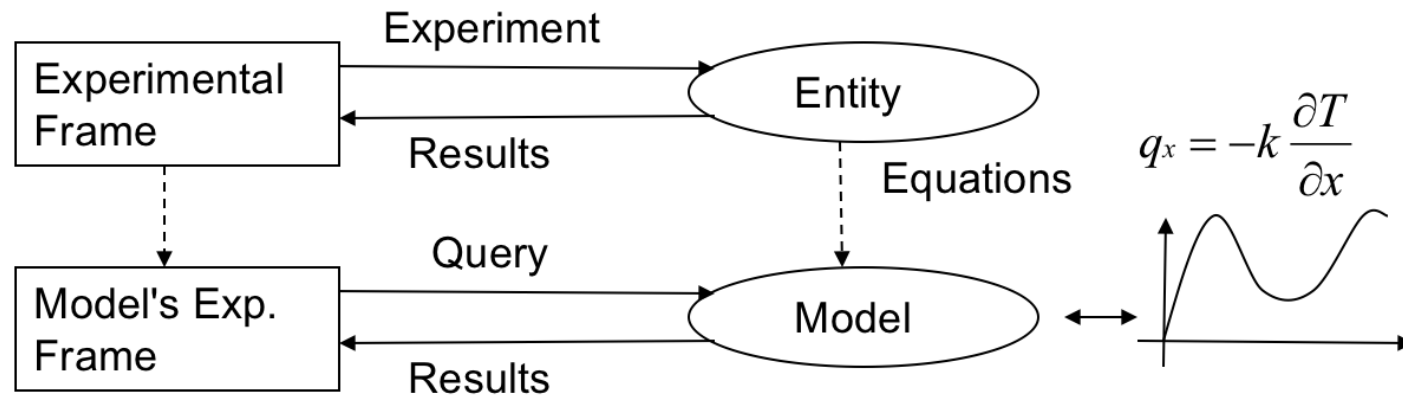
- To simulate, in the scientific connotation, is:
to reproduce the behavior of a given system, process or phenomenon by use of a model
- To simulate, in simple terms, implies to *imitate* or *emulate*
- Simulation is the act of executing, experimenting with or exercising a model for a specific objective (intended use) such as *problem solving, training, acquisition, entertainment, research and education*

The nature of Simulation

- What's being simulated is the **physical system** (or *system under investigation*), which can be:
 - an **actual** (i.e., already existing) system
 - a **hypothetical** (i.e., to be realized) system, which is being analyzed and designed
- To study system, often make assumptions/approximations, both logical and mathematical, about how it works (*abstraction*)
- These assumptions form a **model** of the system, i.e., an abstract representation of the system (*principle of substitutability*)

The nature of Simulation

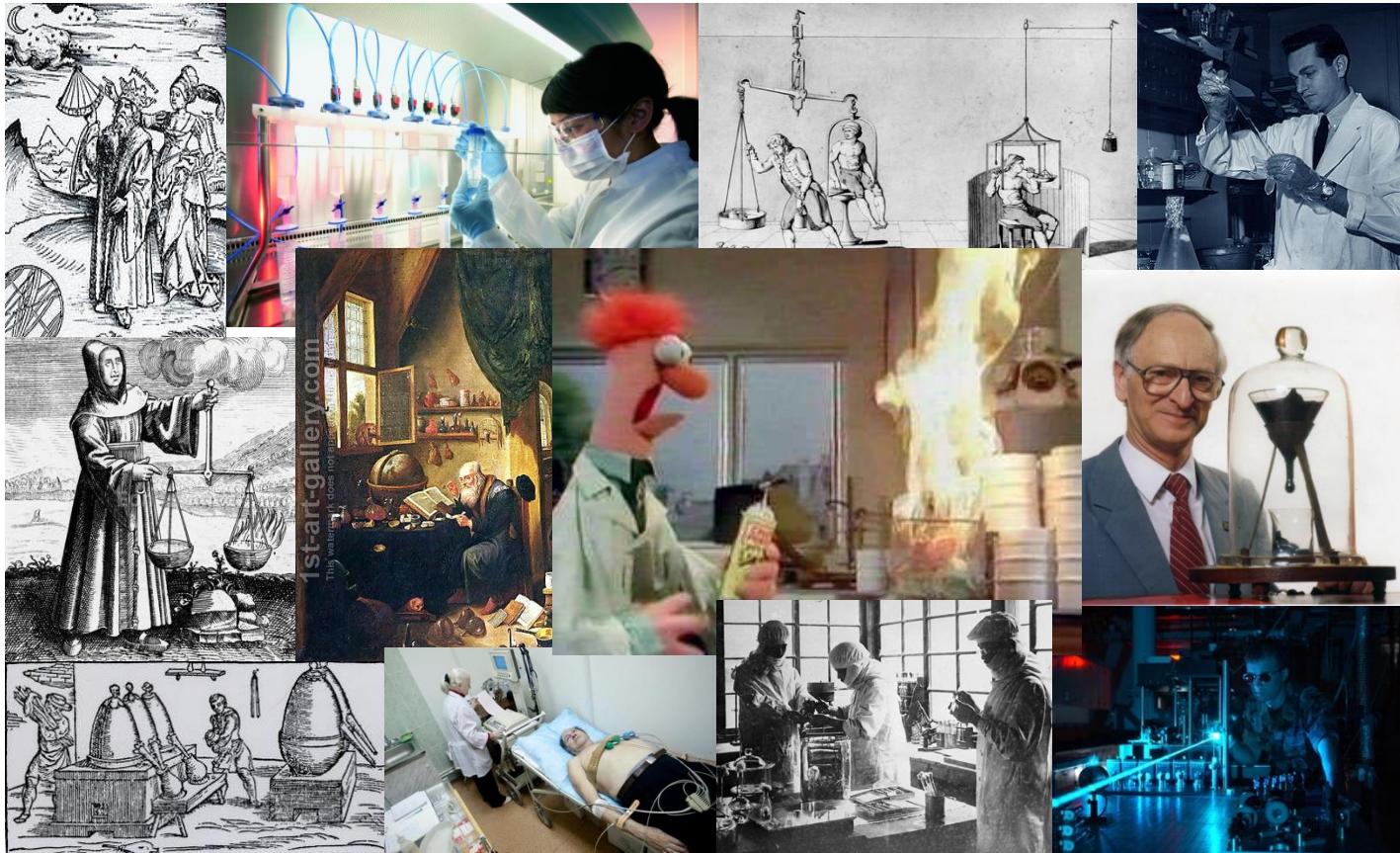
- If model structure is simple enough, could use mathematical methods to get exact information on questions of interest — *analytical solution*



$$\frac{\text{density of object}}{\text{density of fluid}} = \frac{\text{weight}}{\text{weight of displaced fluid}}$$

The nature of Simulation

- A possible alternative, in case the analytical solution is not feasible or too simplistic for the required complexity, is *experimentation*



The nature of Simulation

- *Simulation* is thus solicited when:
 - The mathematical model:
 - is hard or not possible to be defined and/or to be solved analytically
 - does not take into account the peculiar complexity of the problem under study
 - experimentation is:
 - not possible, because the system, process or phenomenon under study does not exist yet
 - not safe, because of the risk of social or economic losses
 - not convenient, because too expensive to be carried out
- The term in its scientific connotation dates back to 40's, when *Fermi, von Neumann e Ulam* introduced a numerical technique (later on coined as "Monte Carlo analysis") to solve nuclear problems that were too expensive to address by experimentation and too complex to address analytically

Simulation is just a part of the story..

- A **simulation** must always have a **model** and *modeling is an essential part of a simulation*
- To emphasize the modeling involved in a simulation, simulation is commonly referred to as **Modeling and Simulation (M&S)**
- The Society for Computer Simulation International (SCS) has changed its name in 2002 into *Society for Modeling & Simulation International* (www.scs.org)
- Modeling and simulation (M&S) constitutes the central part of our thinking process
 - we think by “constructing mental *models* and then *simulating* (i.e., executing) them in order to draw conclusions or make decisions”

Why Simulation?

- Modern systems are complex
 - Investment costs are high
 - Tolerance for error is low
- Simulation helps to
 - Hold costs down
 - Decrease errors (i.e., increase quality)
 - Save time and effort (i.e., avoid effort- and time-consuming rework activities)
- *Simulation* has today reached a level of predictive capability that it now firmly *complements the traditional pillars of science* (i.e., theory and experimentation/observation)

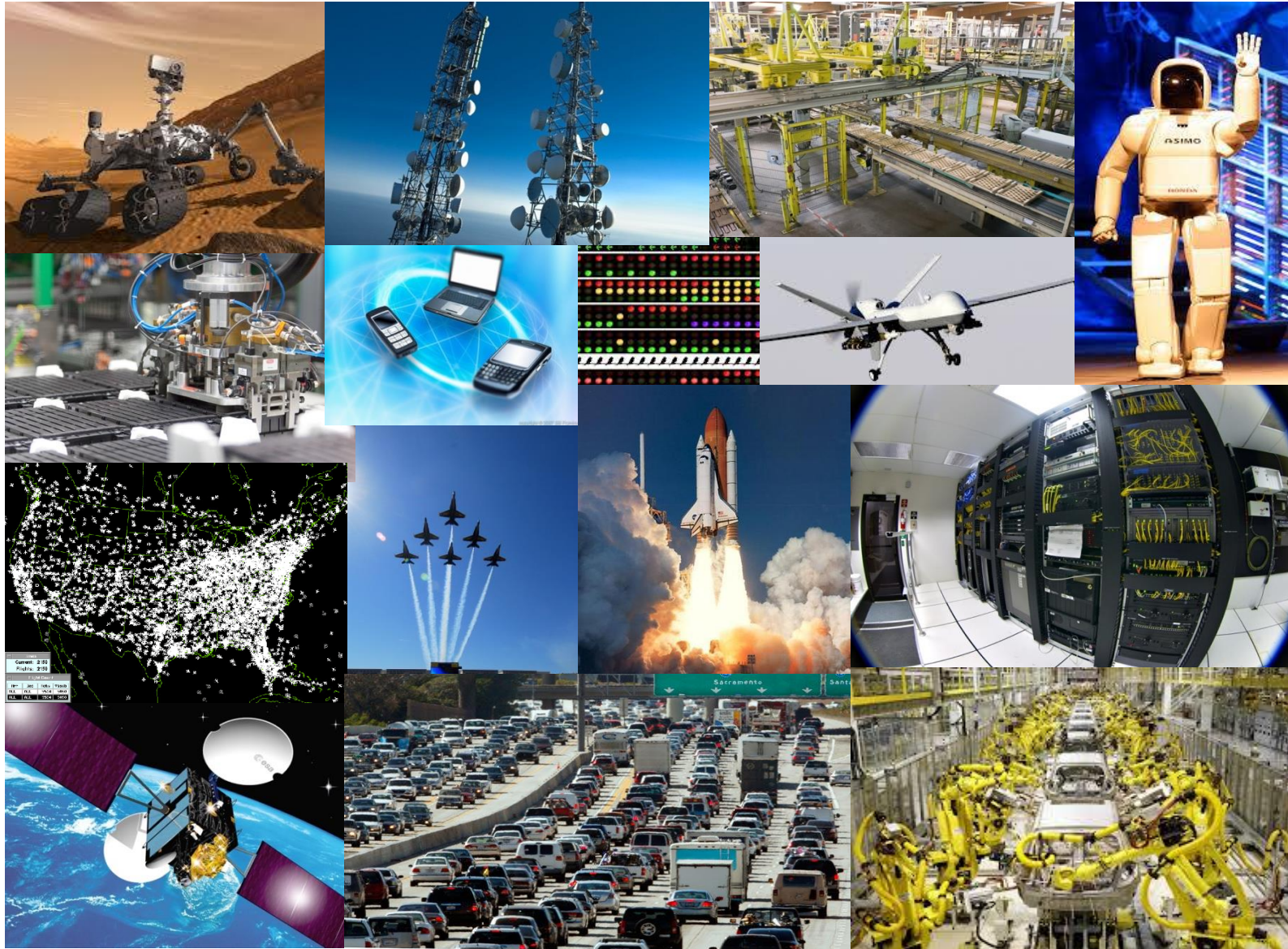
Historical overview

- 1940's
 - Computer simulation in Manhattan Project to model nuclear detonation
- 1950's
 - Analog computers commercially available and used for continuous simulations (solving differential equations)
- 1960's
 - The digital computer offers an alternative to the analog computer and provides a means for simulating discrete event systems incorporating stochastic phenomena
 - **SIMULA**: the first object-oriented programming language
- 1970's and 80's
 - Numerous modeling and simulation applications are developed
- 1980's and 90's
 - emergence of **parallel and distributed simulation** (e.g., SIMNET, simulation standards such as DIS, ALSP, HLA, etc.)

Computer simulation

- A simulation effort carried out by use of a *computer-based platform*
- Consists of:
 - **designing** a model of a system
 - **implementing** the model as an executable software
 - **executing** the software onto a computer-based platform
 - **analyzing** the execution output

Some (not all) application areas



Some (not all) application areas

- Designing and analyzing manufacturing systems
- Evaluating military weapons systems or their logistics requirements
- Determining hardware requirements or protocols for communications networks
- Determining hardware and software requirements for a computer system
- Designing and operating transportation systems such as airports, freeways, ports and subways
- Evaluating designs for service organizations such as call centers, fast-food restaurants, hospitals and post offices
- Analysis and reengineering of business processes
- Determining ordering policies for an inventory system
- Analyzing financial or economic systems

Intended use: problem solving

- **Evaluation**

- Evaluation of a proposed system design for the purpose of assessing its quality characteristics such as operational effectiveness, integrated system effectiveness, deployment readiness, performance, interoperability, and security

- **Comparison**

- Comparing competitive systems designed to carry out a specified function, or comparing several proposed operating policies or procedures

- **Prediction**

- Forecasting the behavior of a system under some projected set of conditions

- **Sensitivity Analysis**

- Determining which of many factors are the most significant in affecting overall system behavior

- **Optimization**

- Determining exactly which combination of factor levels will produce the optimal overall behavior of the system

- **Ranking and Selection**

- Ranking a number of alternatives (e.g., operating policies) and selecting the best one

LVC taxonomy

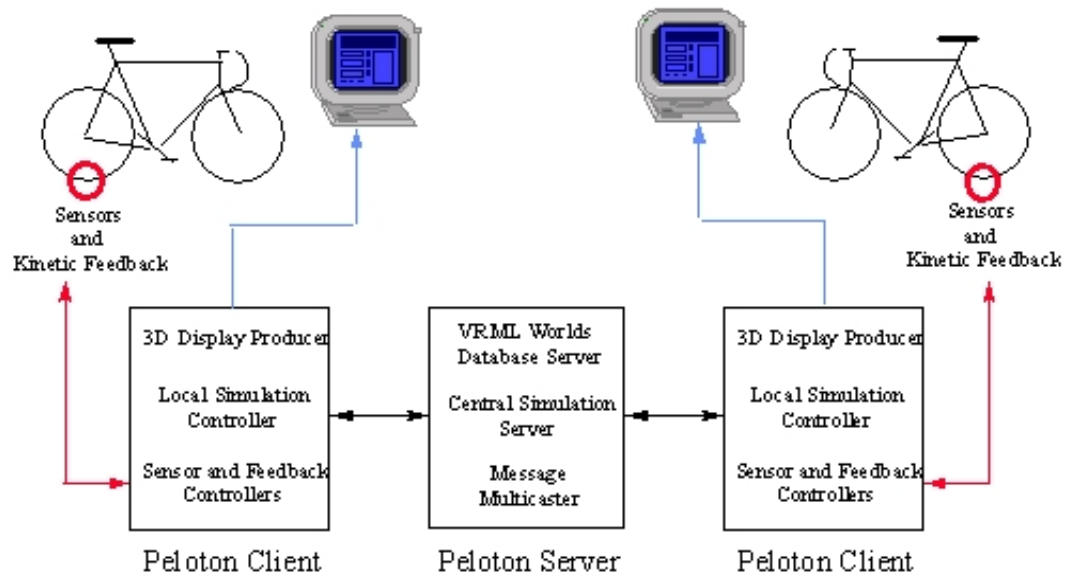
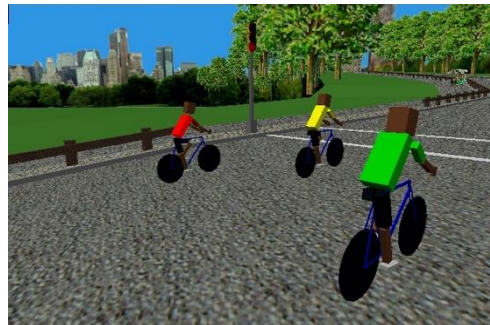
- Types of simulation
 - *Live*: real people operating real equipment
 - *Virtual*: real people operating simulated equipment (often referred to as a “simulator,” e.g., a flight simulator)
 - *Constructive*: simulated people operating simulated equipment
- Major application areas
 - Analysis: logistics, operations
 - Training: platform level, command level
 - Test and evaluation: hardware-in-the-loop

	Real machines	Simulated machines
Real people	Live	Virtual
Simulated people	? (hardware in the loop)	Constructive

Intended use: training



virtual simulation



Intended use: training

constructive simulation



Simulation vs. execution platform

- *Local (or Sequential, or Centralized) Simulation (LS)*
 - the simulation software is executed on top of a **single processor host** (i.e., a single processor computer)
- *Parallel Simulation (PS)*
 - the simulation software is executed on top of a **multi-processor host** (i.e., a computer with multiple processors connected through an “inter-processor” network)
- *Distributed simulation (DS)*
 - the simulation software is executed on top of **multiple hosts** (i.e., a set of computers connected through a network infrastructure of LAN, MAN or WAN type)
 - in the Internet-based DS case, the computer network makes use of Internet protocols

Example multi-processor host

Intel iPSC/2



Simulation speed record (Apr. 13)

Lawrence Livermore and Rensselaer Polytechnic Institute scientists set a new *simulation speed record* on the Sequoia supercomputer



Lawrence Livermore scientists, from left, David Jefferson and Peter Barnes.

Simulation speed record: 504 billion events per second!

M&S Approaches

Classification of M&S approaches:

- based on *model representation*
 - Discrete-event M&S
 - Continuous M&S
 - Monte Carlo M&S
 - Agent-based M&S
 - System Dynamics M&S
- based on *model execution*
 - Sequential simulation
 - Parallel simulation
 - Distributed/Web-based simulation
- based on *what-is-in-the-loop*
 - Hardware-in-the-loop
 - Software-in-the-loop
 - Human-in-the-loop



- M&S approaches differ from each other depending on how the model is *defined*, *implemented* and *executed*

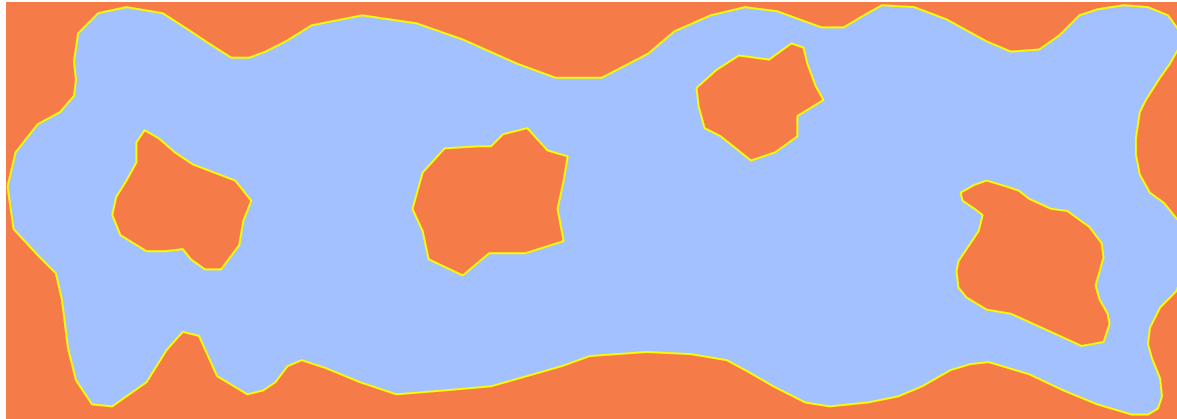
Intended use

- Problem Solving
- Training
- Acquisition
- Entertainment
- Research
- Education

- Use of single or combined M&S approaches *spans dozen of different disciplines and application domains*, for many different *intended uses*

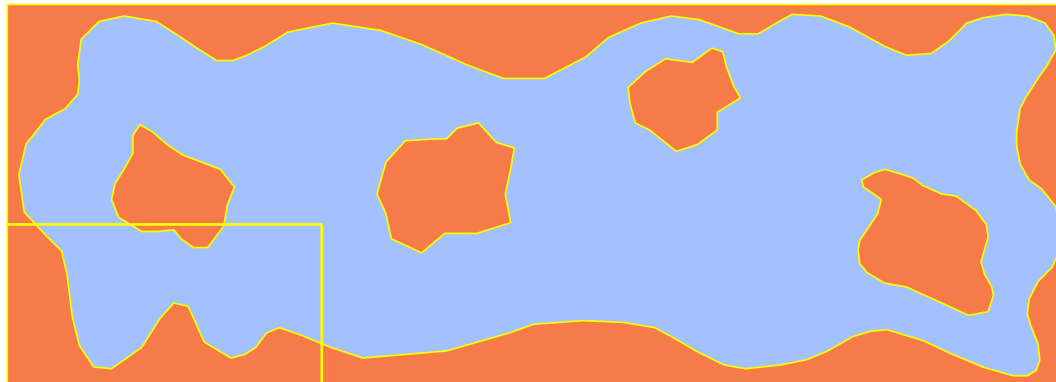
Monte Carlo M&S

- based on (pseudo-)casual number generation
- example: given a high-resolution computer image of a map of an irregularly shaped lake with several islands, determine the water surface area. Assume that the x-y coordinates of every point on the map can be measured.



Monte Carlo M&S

- *Step 1*: enclose the area of interest in the smallest rectangle of known dimensions X and Y
- *Step 2*: generate two uniformly distributed random numbers, RN_x in $[0..X]$ e RN_y in $[0..Y]$
- *Step 3*: If the point of intersection (RN_x, RN_y) falls on the water surface area, add 1 to S (being S the number of hits on the water surface, with $S=0$ at startup)
- *Step 4*: repeat Step 2 and Step 3 for a number N times
- *Step 5*: the estimate of the water surface area is $X \cdot Y \cdot \frac{S}{N}$



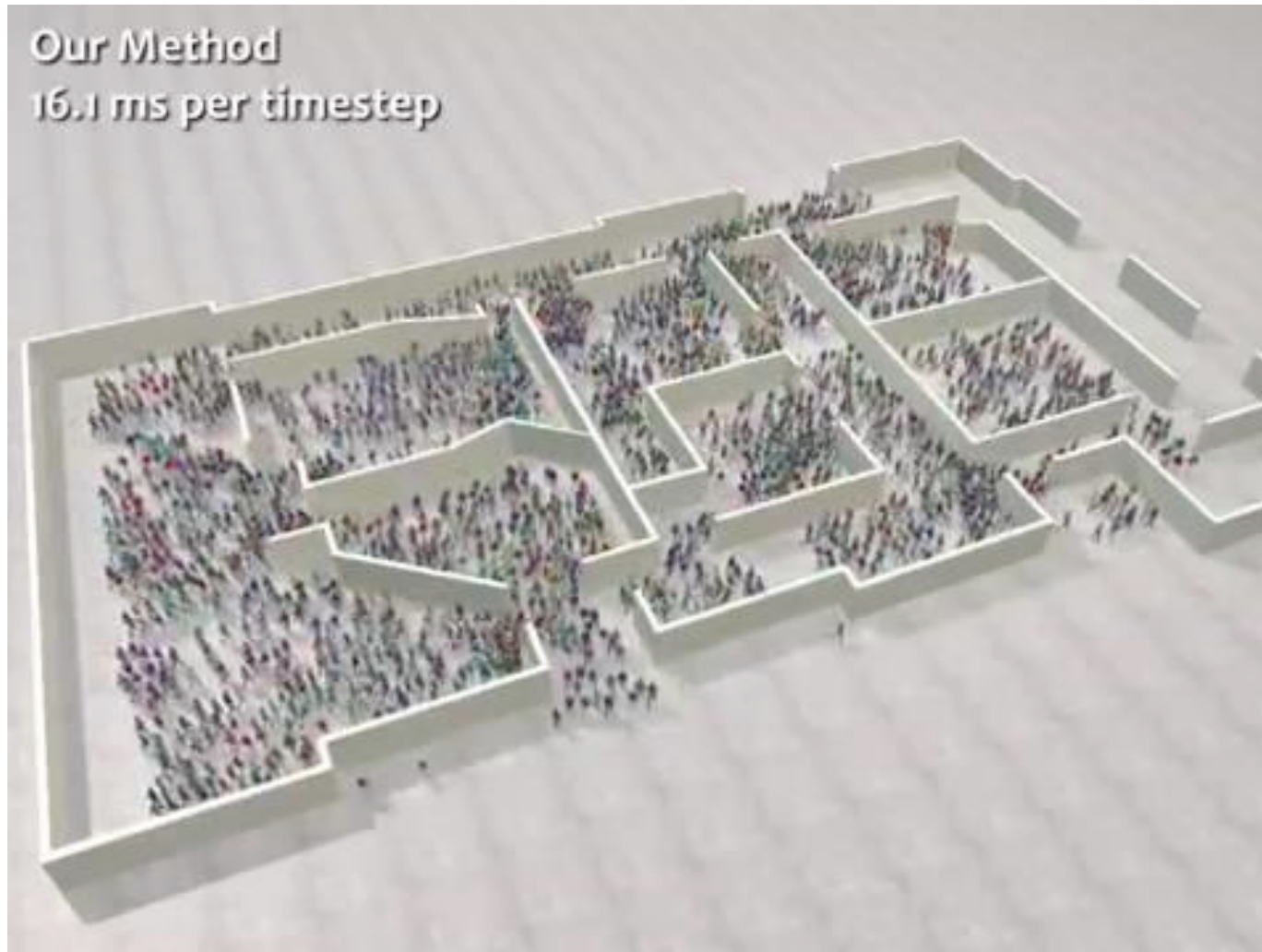
Agent-based M&S

- Agent-based M&S is the one which uses a model representing multiple autonomous entities and their interactions
 - actions are not predefined
 - *agents* representing entities react to the events occurring in their environment
- An agent is:
 - autonomous
 - adaptive
 - reactive
 - proactive
 - “social”

and has the ability to learn (change its behavior based on experience)

Agent-based M&S example

building evacuation



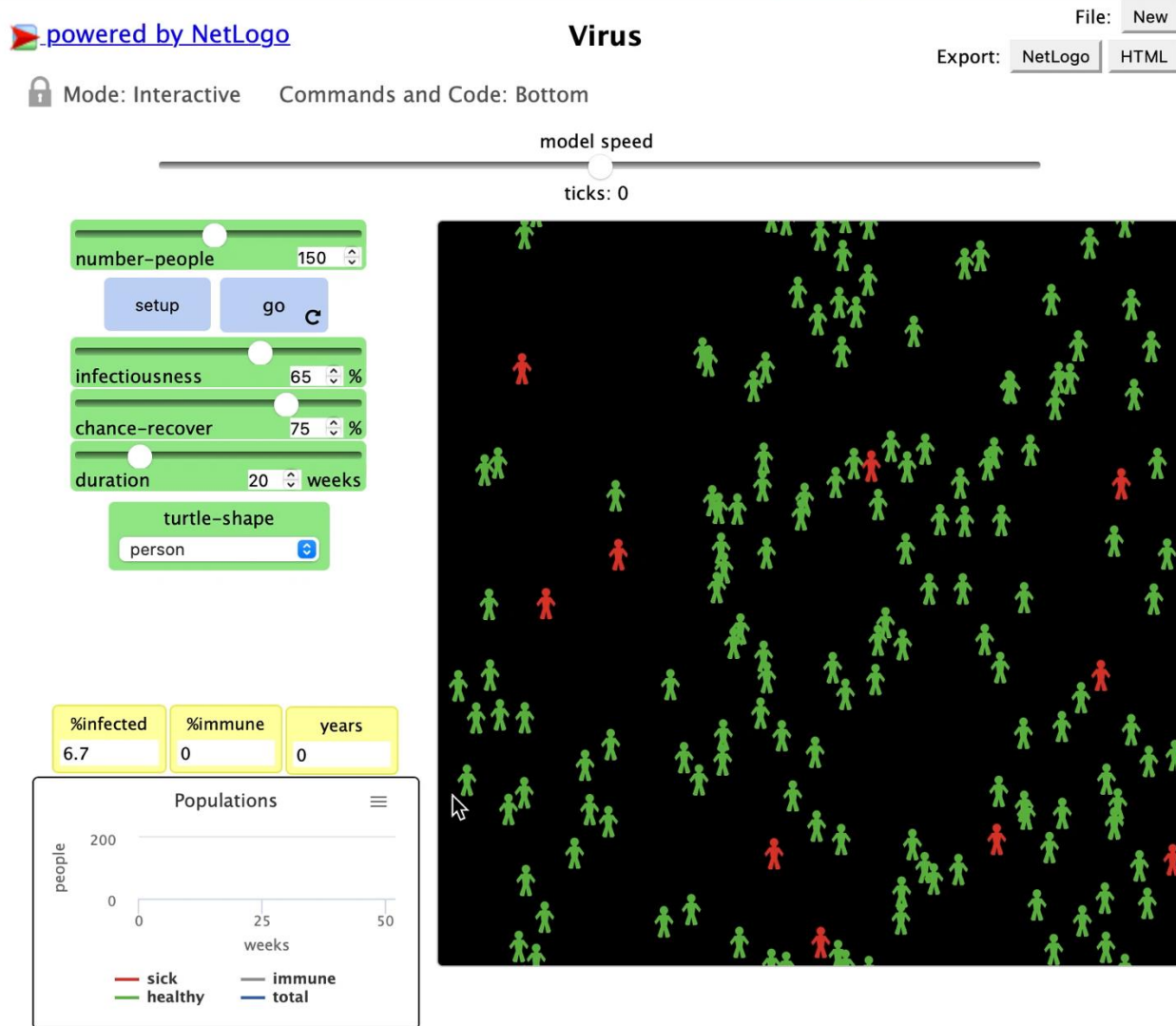
Agent-based M&S example

crowd simulation



Agent-based M&S example

virus spread simulation



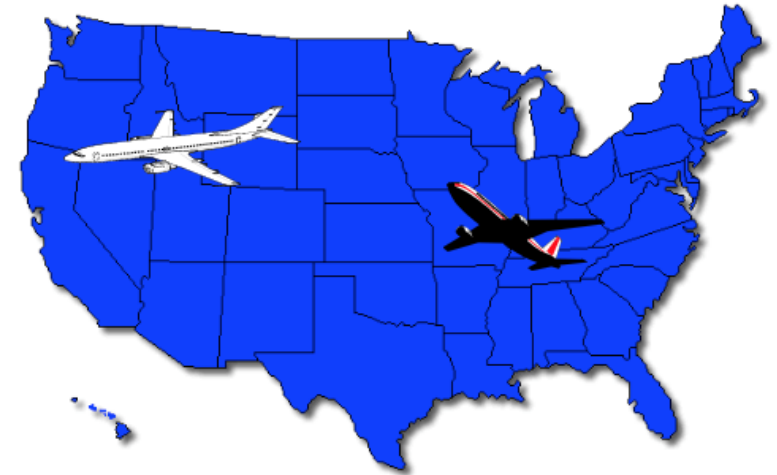
Discrete Event Simulation (DES)

- Modeling of a system as it evolves over time
- The simulation must provide:
 - a representation of the **state** of the physical system (*dependent variable*)
 - the state is defined as a collection of **state variables**, namely program variables usually specified in some high-level programming language such as C or Java that represents the state of the physical system
 - some means of changing this representation to model the evolution of the physical system
 - changes in the state of the physical system are realized by the simulation program writing new values into these state variables
 - a representation of **time** (*independent variable*)
 - the time is represented through an abstraction called **simulation time**
- More precisely, state can change at only a countable number of points in time
 - These points in time are when events occur
- **Event**: Instantaneous occurrence that *may* change the state of the system

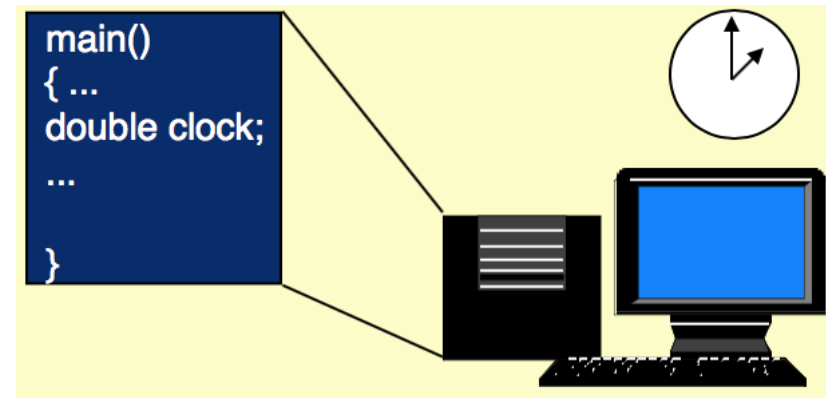
The time notion

There are several different notions of time that are important when discussing a simulation:

- **Physical time**
 - refers to time in the physical system
- **Simulation time**
 - is an abstraction used by the simulation to model physical time
- **Wallclock time**
 - refers to time during the execution of the simulation program
 - a simulation program can usually obtain the current value of wallclock time (accurate to some specifiable amount of error) by reading a hardware clock maintained by the operating system



physical system



The different time notions

To illustrate these different notions of time, consider a simulation of the transportation system in Atlanta during the 1996 summer Olympic games:

- *Physical time* for this simulation extends from July 19 to August 4, 1996
- *Simulation time* might be represented in the simulation program by a double precision floating point number with each unit corresponding to a single day. In this case, simulation time advances from 0.00 to 17.00 during each execution of the program
- If the simulation program ran for three hours on the afternoon of February 25, 1995, while planning for the Olympics, *wallclock time* might extend from 3:00 PM to 6:00 PM on that day

Time flow mechanisms

- A simulation execution can be classified in terms of temporal aspects (e.g., a *real-time* simulation)
- A second, independent, classification corresponds to the manner in which the state of the model changes as simulation time is advanced, sometime referred to as the *time flow mechanism*
- According to this second classification, simulation models may be broadly classified as *continuous* or *discrete*
- In a *discrete simulation*, the simulation model views the physical system as only changing state at discrete points in simulation time
 - conceptually the system is viewed as "*jumping*" from one state to the next, much like moving from one frame to another in a cartoon strip

Event-driven execution

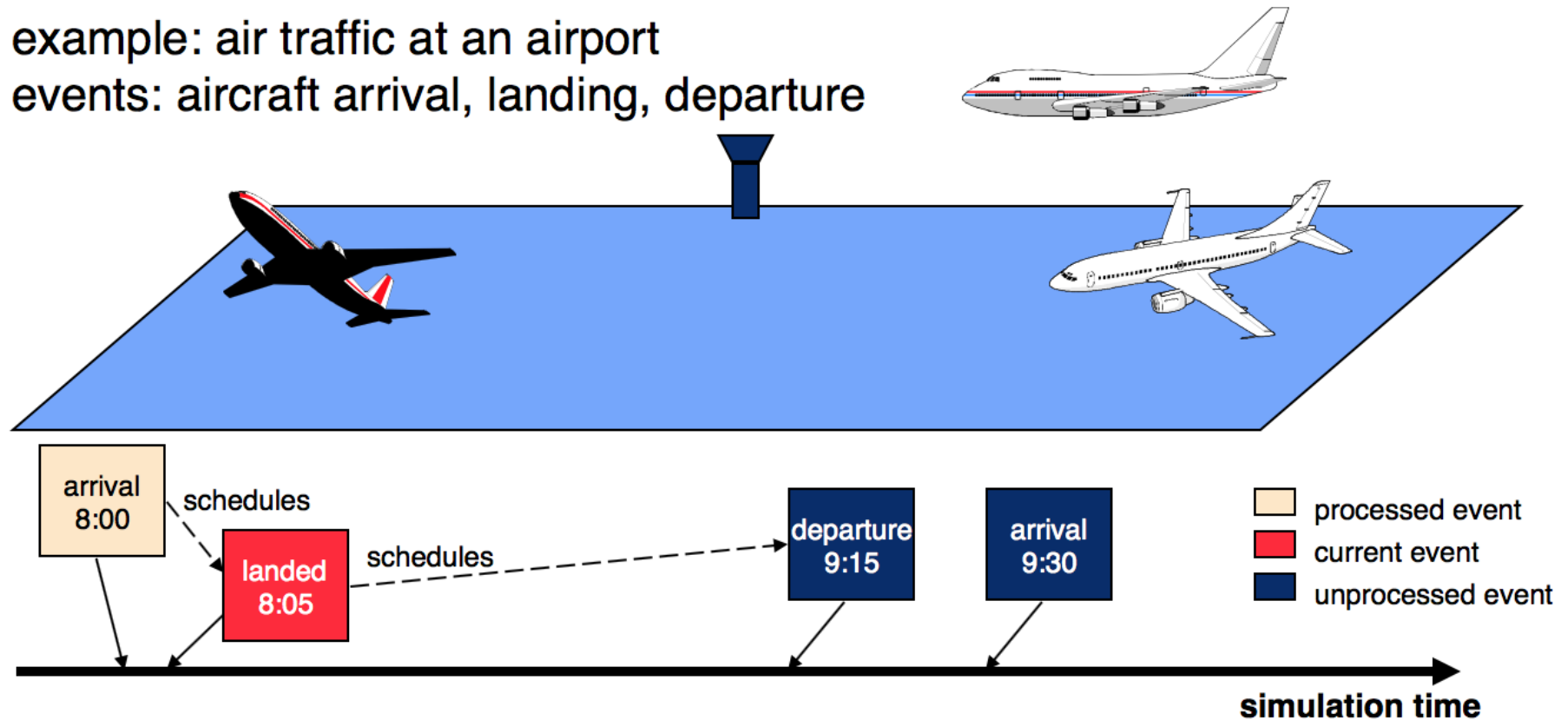
- Rather than compute a new value for state variables each time step, it may be more efficient to only update the variables when "*something interesting*" occurs
- The "something interesting" that occurs is referred to as an *event*
- This is the key idea behind *discrete event simulations (DES)*
- An event is an abstraction used in the simulation to model some instantaneous action in the physical system
- **Each event has a time stamp** associated with it that indicates the point in simulation time when the event occurs
- Each event usually results in some change in one or more state variables defined by the simulation

Example event-driven simulation

- Consider a simulation of an aircraft flying from New York to Los Angeles
- A time-stepped simulation with time step size of 10 minutes might compute the aircraft's new position every 10 minutes
- An event-driven approach is more efficient if intermediate positions are not needed
 - only update the aircraft's position variable when simulation time advances to the time the aircraft reaches Los Angeles
 - the aircraft arriving in Los Angeles is modeled by an event
 - changes in state variables only occur as the result of some event

Example event-driven simulation

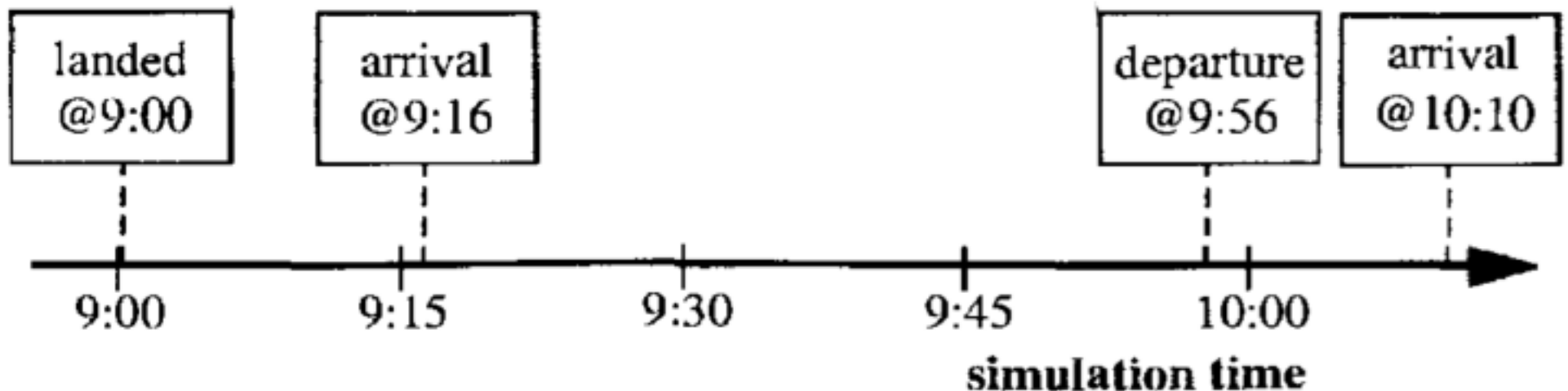
example: air traffic at an airport
events: aircraft arrival, landing, departure



- Unprocessed events are stored in a pending event list
- Events are processed in time stamp order

Example sequence of events

- Simulation time advances from the time stamp of one event to the next
- From a computational standpoint, the simulation can be viewed as a sequence of computations, one for each event, transforming the system across simulated time in a manner representing the behavior of the actual system



“Hand” DES Example

- *Single-server queue*

- **Objective**

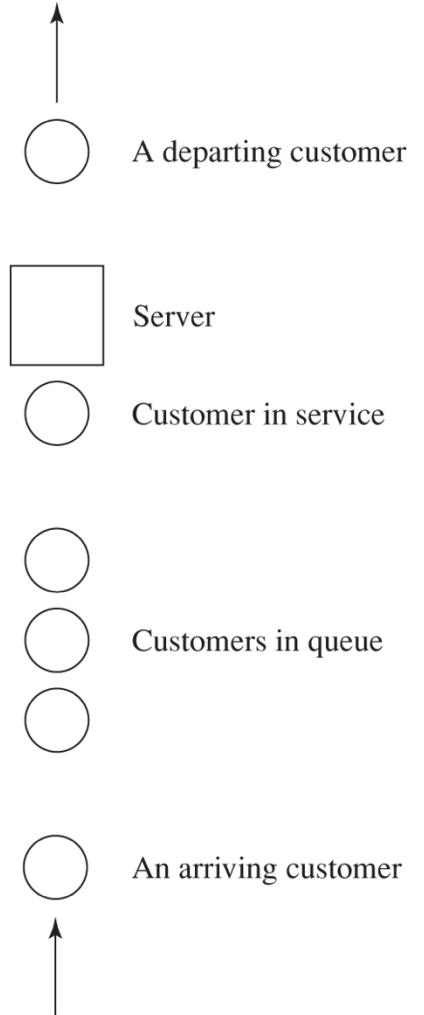
- Estimate expected average delay in queue (line, not service)

- **State variables**

- Status of server (idle, busy) – needed to decide what to do with an arrival
 - Current length of the queue – to know where to store an arrival that must wait in line
 - Time of arrival of each customer now in queue – needed to compute time in queue when service starts

- **Events**

- Arrival of a new customer
 - Service completion (and departure) of a customer
 - Maybe – end-simulation event (a “fake” event) – whether this is an event depends on how simulation terminates (a modeling decision)



“Hand” DES Example

- Next-event time advance for the single-server queue

t_i = time of arrival of i th customer ($t_0 = 0$)

$A_i = t_i - t_{i-1}$ = inter-arrival time between $(i-1)$ st and i th customers (usually assumed to be an IID random variable)

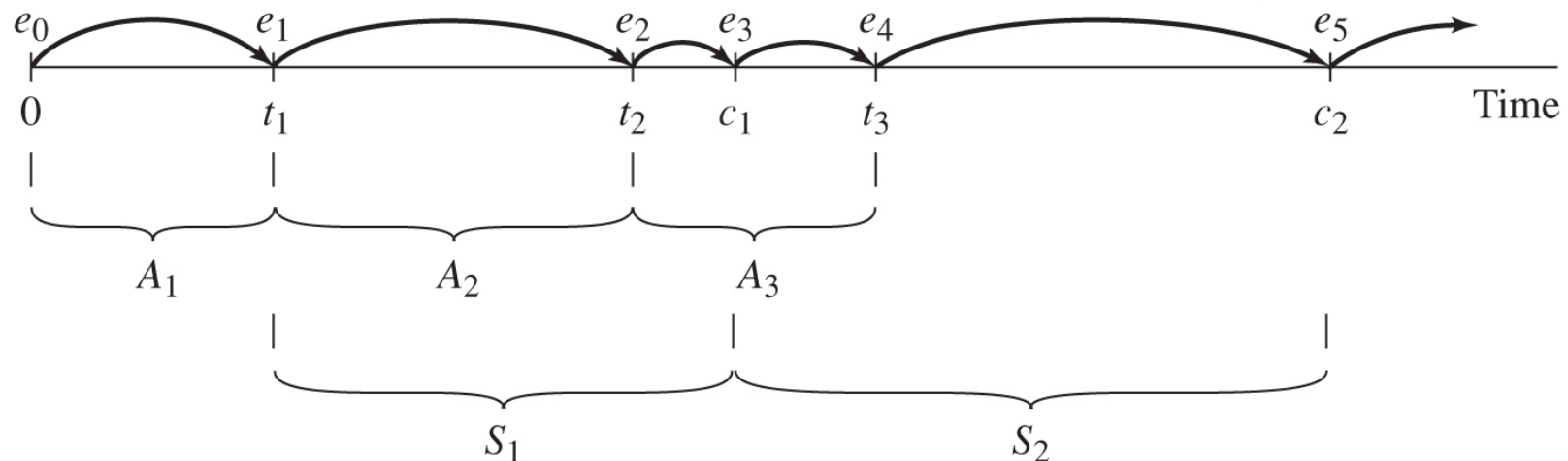
S_i = service-time requirement of i th customer (another IID random variable)

D_i = delay in queue of i th customer

$c_i = t_i + D_i + S_i$ = time i th customer completes service and departs

e_j = time of occurrence of the j th event (of any type), $j = 1, 2, 3, \dots$

- Possible trace of events



“Hand” DES Example

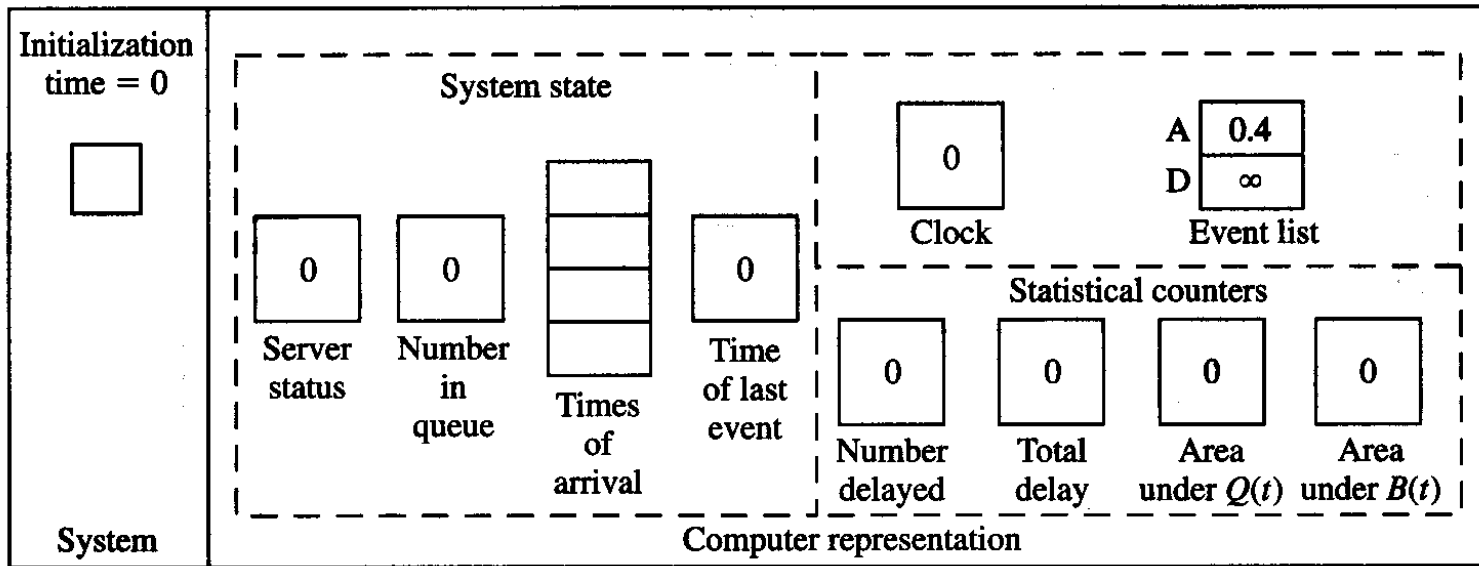
- Given (for now) inter-arrival times (all times are in minutes):
 - 0.4, 1.2, 0.5, 1.7, 0.2, 1.6, 0.2, 1.4, 1.9, ...
- Given service times:
 - 2.0, 0.7, 0.2, 1.1, 3.7, 0.6, ...
- $n = 6$ delays in queue desired
- “Hand” simulation:
 - Display system, state variables, clock, event list, statistical counters ... all after execution of each event
 - Use above lists of inter-arrival, service times to “drive” simulation
 - Stop when number of delays hits $n = 6$, compute output performance measures

“Hand” DES Example

- Quantities to be estimated
 - *Expected average delay in queue* (excluding service time) of the n customers completing their delays
 - Why “expected?”
 - *Expected average number of customers in queue* (excluding any in service)
 - A *continuous-time average*
 - Area under $Q(t)$ = queue length at time t , divided by $T(n)$ = time simulation ends
 - *Expected utilization (proportion of time busy) of the server*
 - Another *continuous-time average*
 - Area under $B(t)$ = server-busy function (1 if busy, 0 if idle at time t), divided by $T(n)$
 - Many others are possible (maxima, minima, time or number in system, proportions, quantiles, variances, etc.)
- Important: *Discrete-time vs. continuous-time* statistics

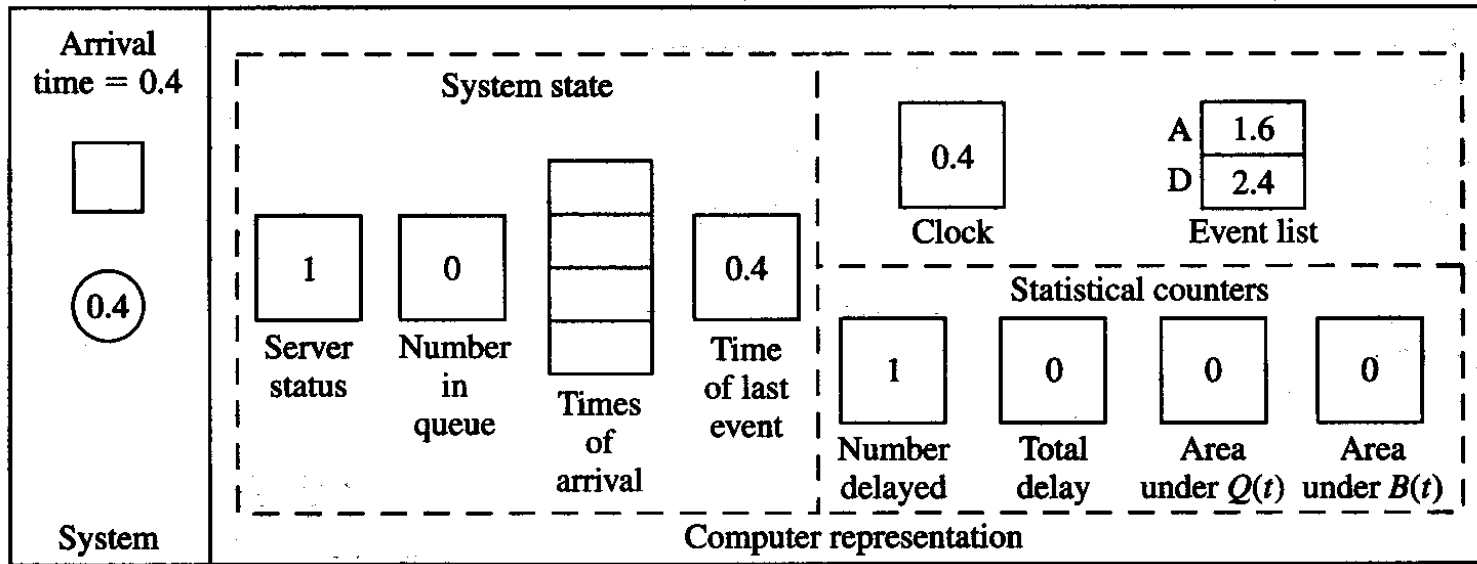
“Hand” DES Example

Status shown is *after* all changes have been made in each case ...



Interarrival times: ~~0.4~~, 1.2, 0.5, 1.7, 0.2, 1.6, 0.2, 1.4, 1.9, ...
 Service times: 2.0, 0.7, 0.2, 1.1, 3.7, 0.6, ...

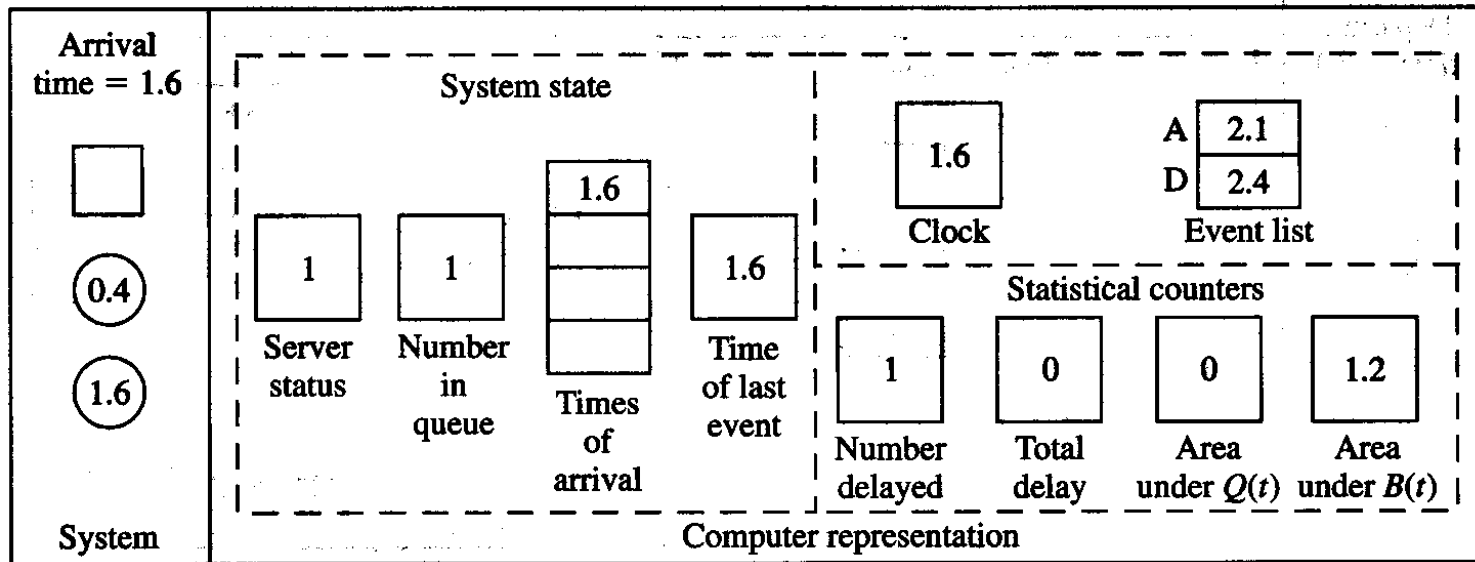
“Hand” DES Example



Interarrival times: ~~0.4~~, ~~1.2~~, 0.5, 1.7, 0.2, 1.6, 0.2, 1.4, 1.9, ...

Service times: ~~2.0~~, 0.7, 0.2, 1.1, 3.7, 0.6, ...

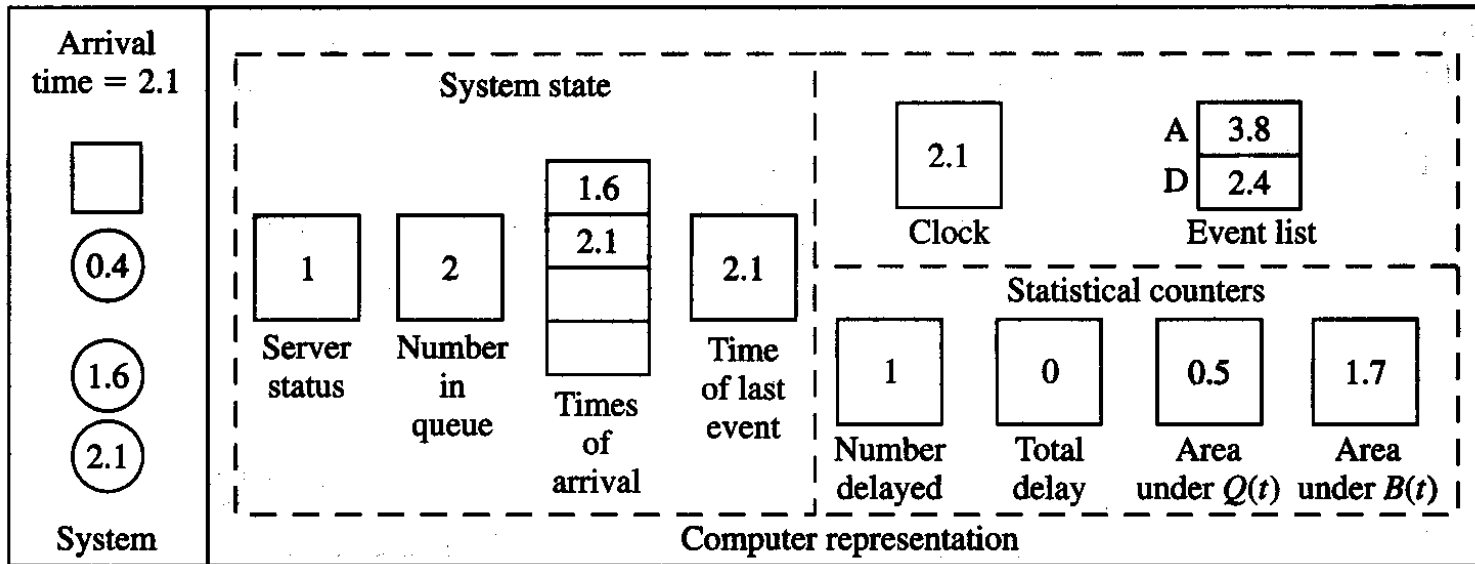
“Hand” DES Example



Interarrival times: ~~0.4~~, ~~1.2~~, ~~0.5~~, 1.7, 0.2, 1.6, 0.2, 1.4, 1.9, ...

Service times: ~~2.0~~, 0.7, 0.2, 1.1, 3.7, 0.6, ...

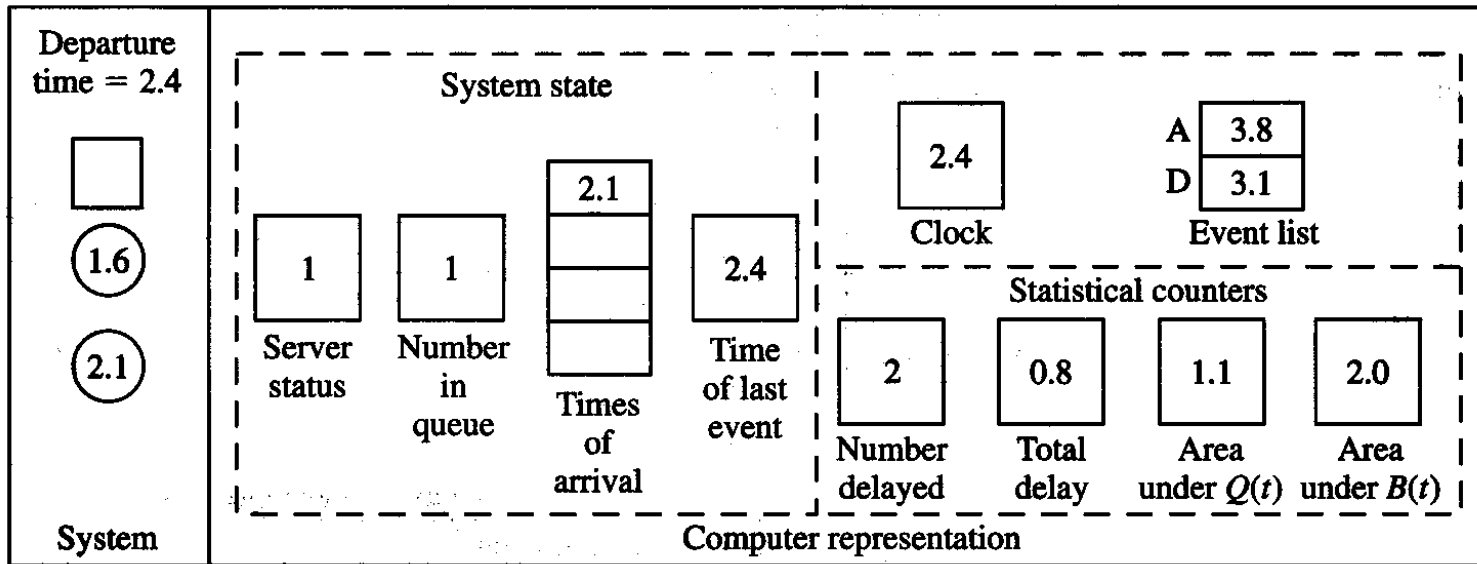
“Hand” DES Example



Interarrival times: ~~0.4~~, ~~1.2~~, ~~0.5~~, ~~1.7~~, 0.2, 1.6, 0.2, 1.4, 1.9, ...

Service times: ~~2.0~~, 0.7, 0.2, 1.1, 3.7, 0.6, ...

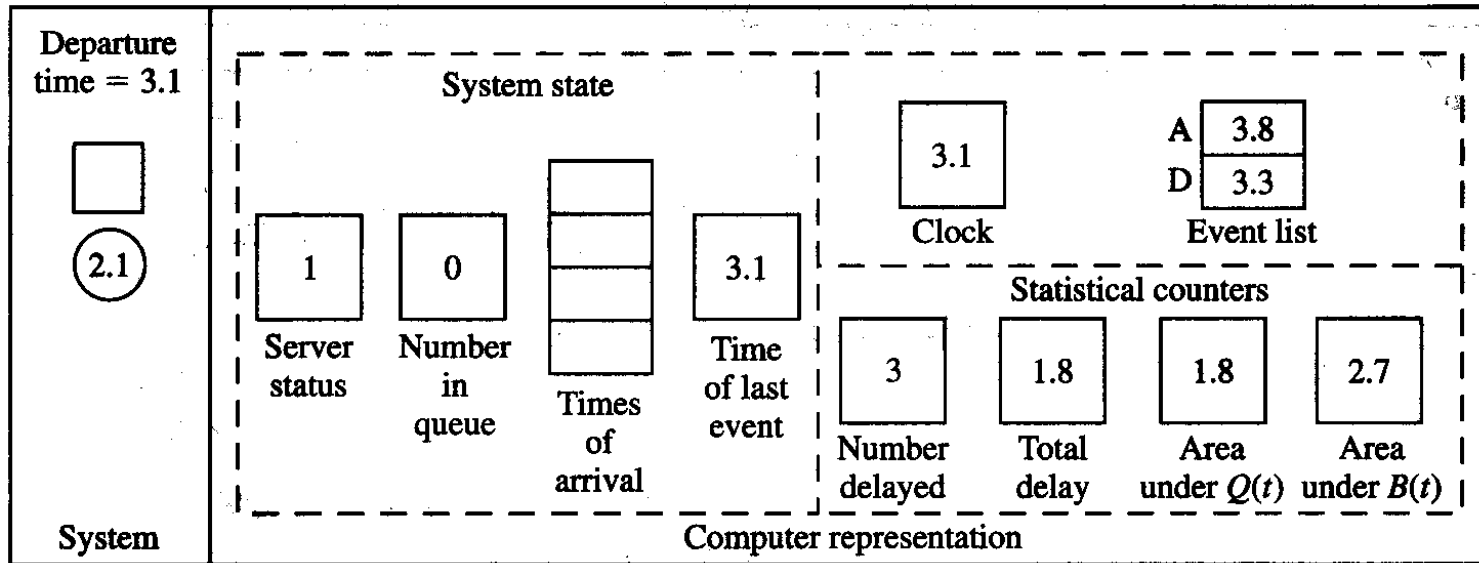
“Hand” DES Example



Interarrival times: ~~0.4~~, ~~1.2~~, ~~0.5~~, ~~1.7~~, 0.2, 1.6, 0.2, 1.4, 1.9, ...

Service times: ~~2.0~~, ~~0.7~~, 0.2, 1.1, 3.7, 0.6, ...

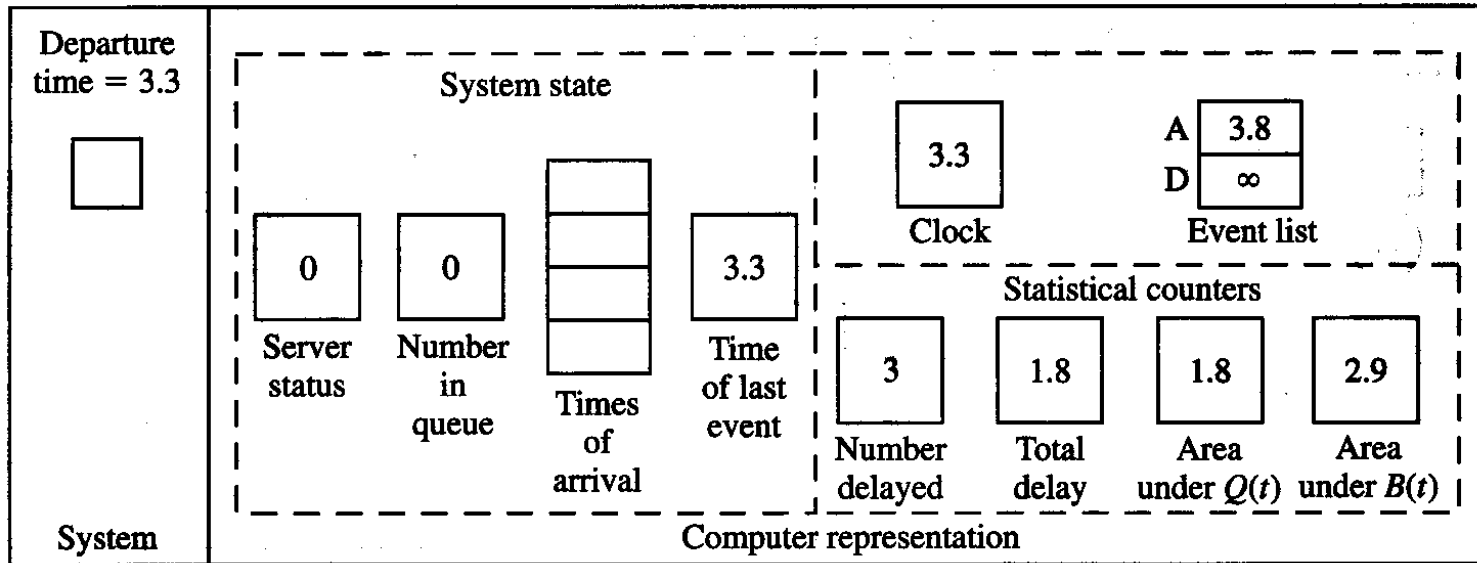
“Hand” DES Example



Interarrival times: ~~0.4~~, ~~1.2~~, ~~0.5~~, ~~1.7~~, 0.2, 1.6, 0.2, 1.4, 1.9, ...

Service times: ~~2.0~~, ~~0.7~~, ~~0.2~~, 1.1, 3.7, 0.6, ...

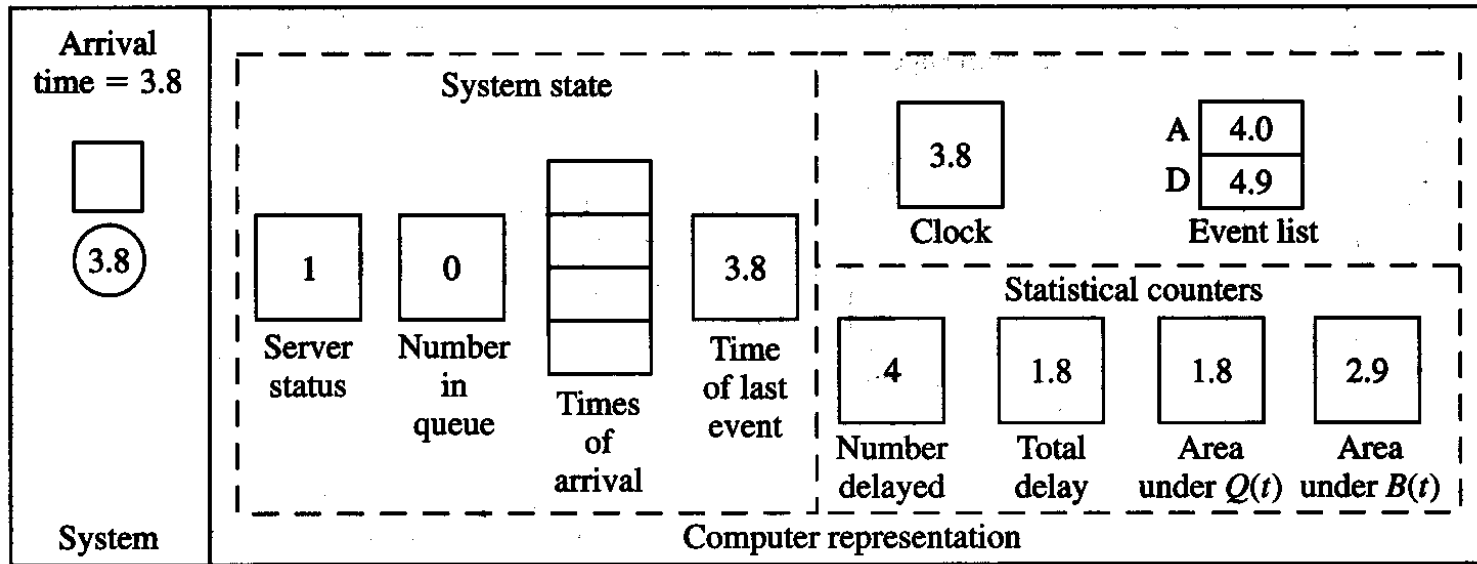
“Hand” DES Example



Interarrival times: ~~0.4~~, ~~1.2~~, ~~0.5~~, ~~1.7~~, 0.2, 1.6, 0.2, 1.4, 1.9, ...

Service times: ~~2.0~~, ~~0.7~~, ~~0.2~~, 1.1, 3.7, 0.6, ...

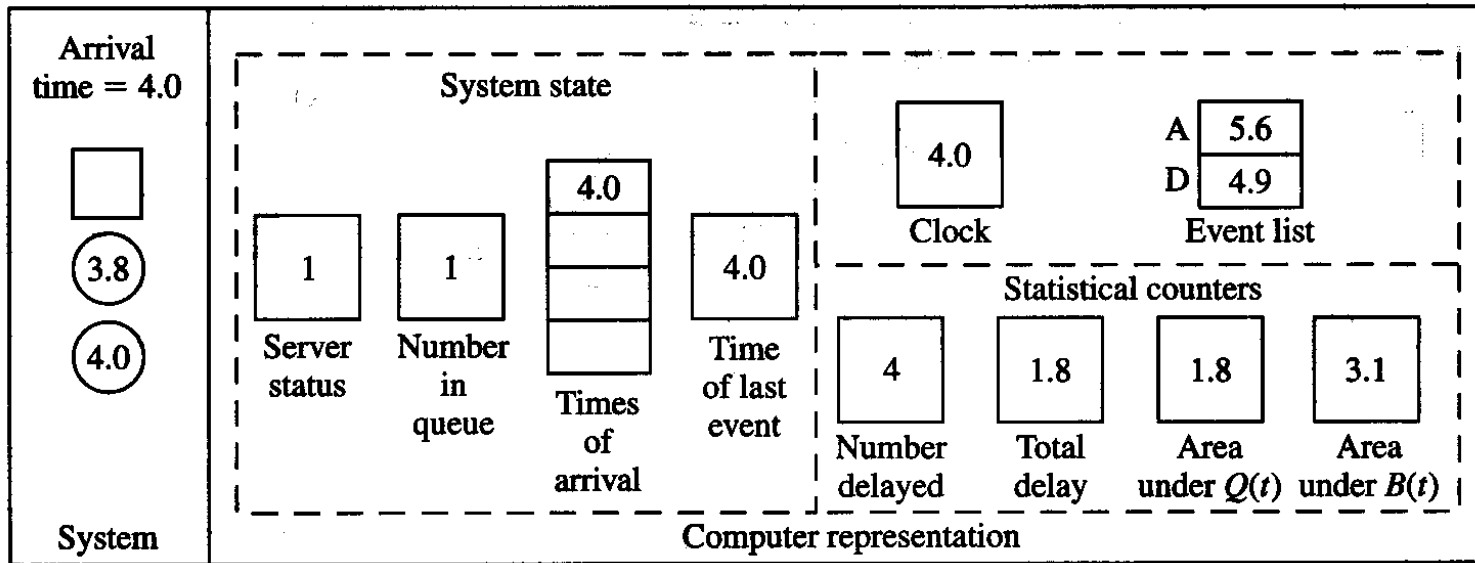
“Hand” DES Example



Interarrival times: ~~0.4~~, ~~1.2~~, ~~0.5~~, ~~1.7~~, ~~0.2~~, 1.6, 0.2, 1.4, 1.9, ...

Service times: ~~2.0~~, ~~0.7~~, ~~0.2~~, ~~1.1~~, 3.7, 0.6, ...

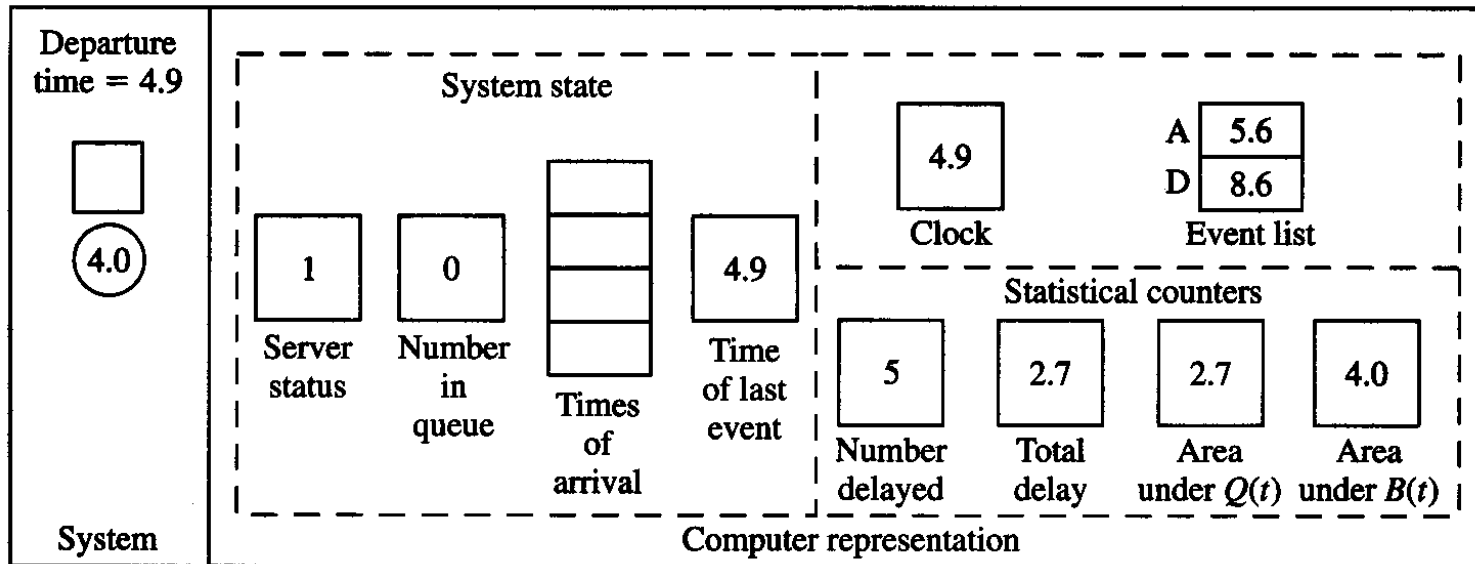
“Hand” DES Example



Interarrival times: ~~0.4~~, ~~1.2~~, ~~0.5~~, ~~1.7~~, ~~0.2~~, ~~1.6~~, 0.2, 1.4, 1.9, ...

Service times: ~~2.0~~, ~~0.7~~, ~~0.2~~, ~~1.1~~, 3.7, 0.6, ...

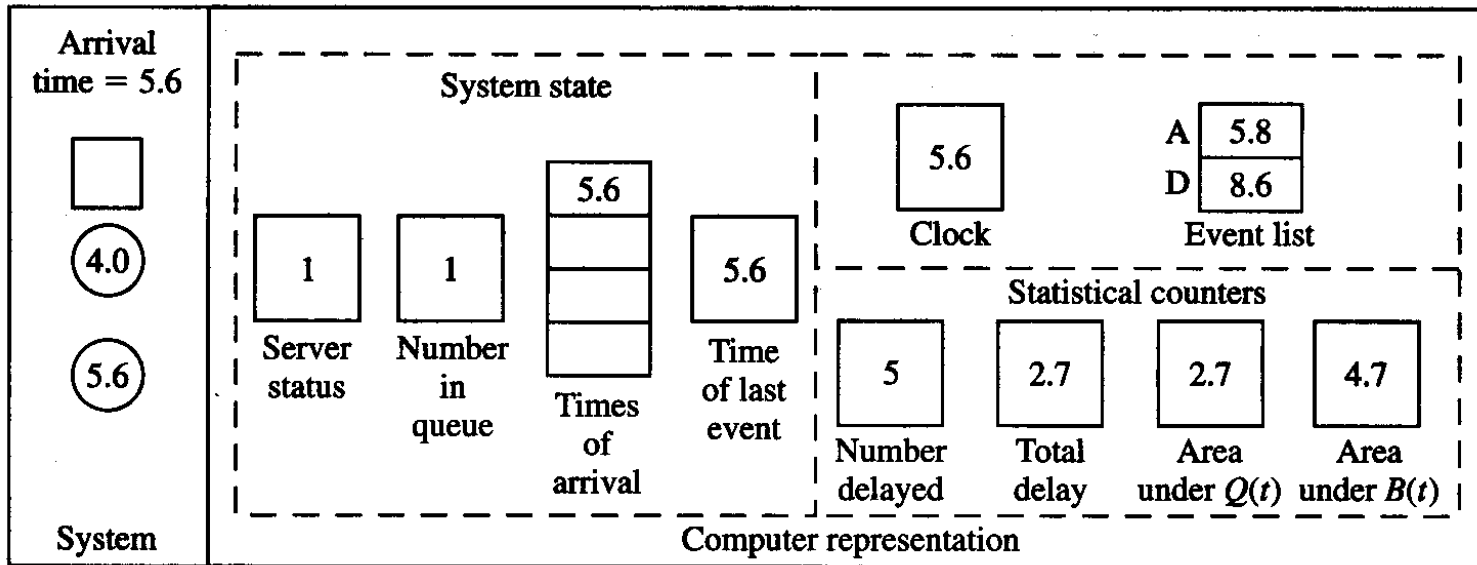
“Hand” DES Example



Interarrival times: ~~0.4~~, ~~1.2~~, ~~0.5~~, ~~1.7~~, ~~0.2~~, ~~1.6~~, 0.2, 1.4, 1.9, ...

Service times: ~~2.0~~, ~~0.7~~, ~~0.2~~, ~~1.1~~, ~~3.7~~, 0.6, ...

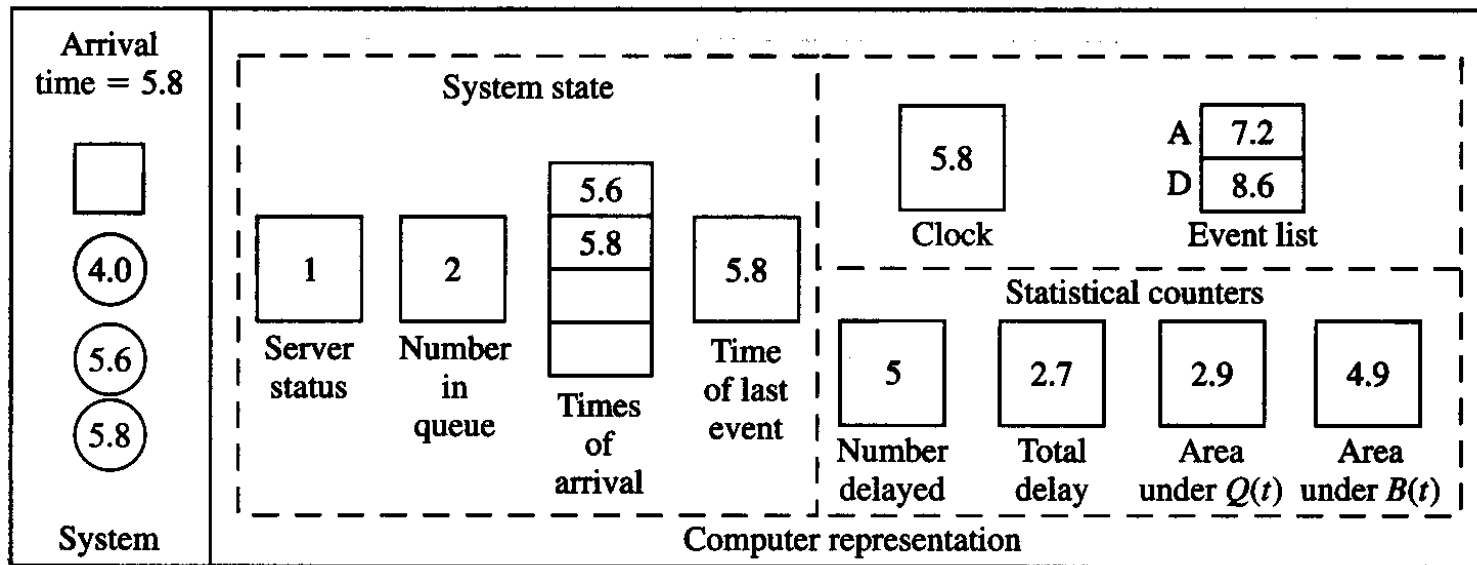
“Hand” DES Example



Interarrival times: ~~0.4~~, ~~1.2~~, ~~0.5~~, ~~1.7~~, ~~0.2~~, ~~1.6~~, ~~0.2~~, 1.4, 1.9, ...

Service times: ~~2.0~~, ~~0.7~~, ~~0.2~~, ~~1.1~~, ~~3.7~~, 0.6, ...

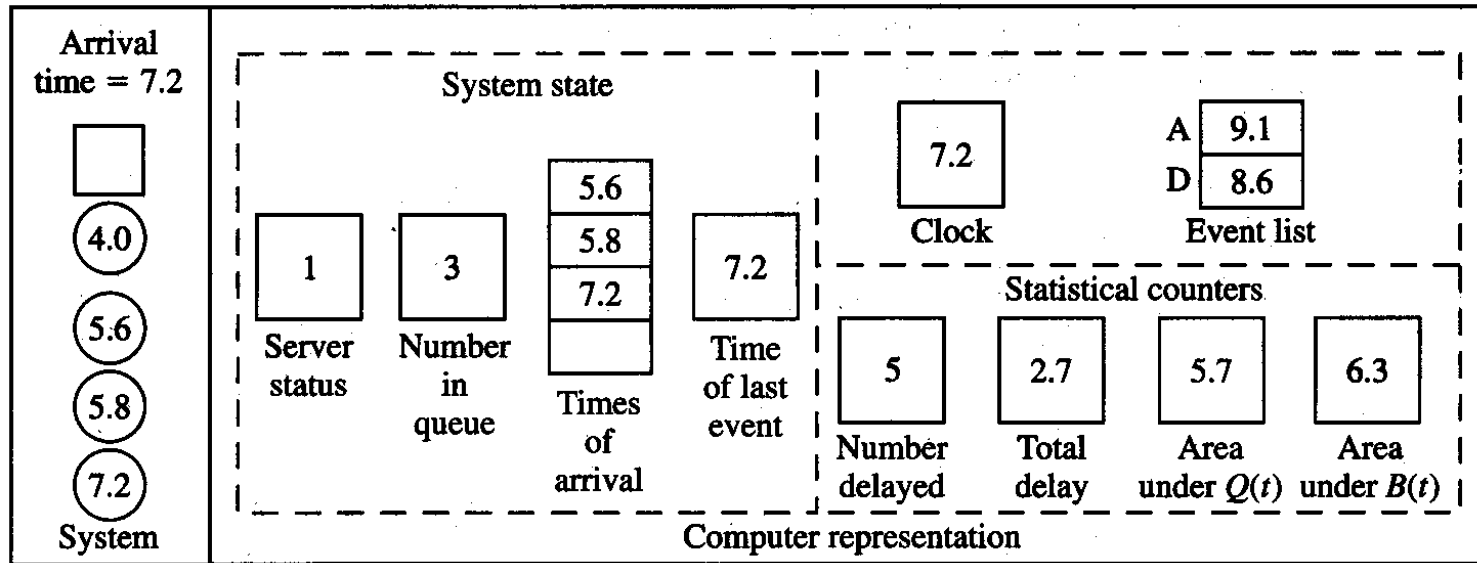
“Hand” DES Example



Interarrival times: ~~0.4~~, ~~1.2~~, ~~0.5~~, ~~1.7~~, ~~0.2~~, ~~1.6~~, ~~0.2~~, ~~1.4~~, 1.9, ...

Service times: ~~2.0~~, ~~0.7~~, ~~0.2~~, ~~1.1~~, ~~3.7~~, 0.6, ...

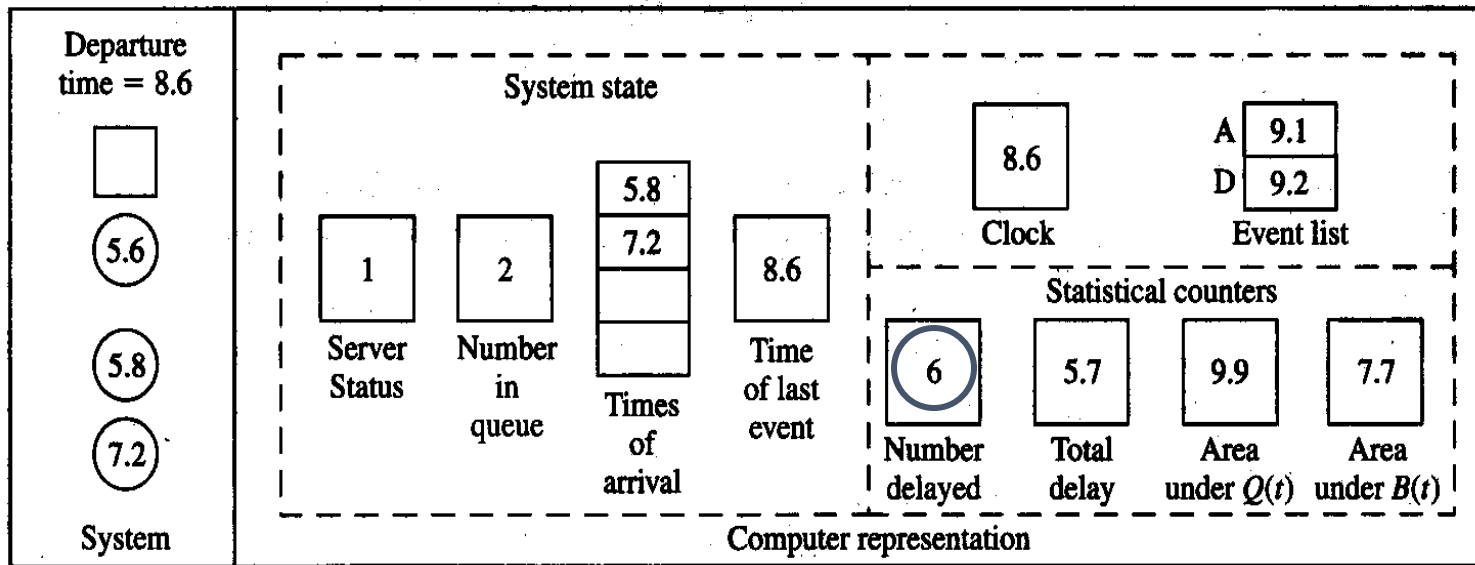
“Hand” DES Example



Interarrival times: ~~0.4~~, ~~1.2~~, ~~0.5~~, ~~1.7~~, ~~0.2~~, ~~1.6~~, ~~0.2~~, ~~1.4~~, ~~1.9~~, ...

Service times: ~~2.0~~, ~~0.7~~, ~~0.2~~, ~~1.1~~, ~~3.7~~, 0.6, ...

“Hand” DES Example



Interarrival times: ~~0.4~~, ~~1.2~~, ~~0.5~~, ~~1.7~~, ~~0.2~~, ~~1.6~~, ~~0.2~~, ~~1.4~~, ~~1.9~~, ...

Service times: ~~2.0~~, ~~0.7~~, ~~0.2~~, ~~1.1~~, ~~3.7~~, ~~0.6~~, ...

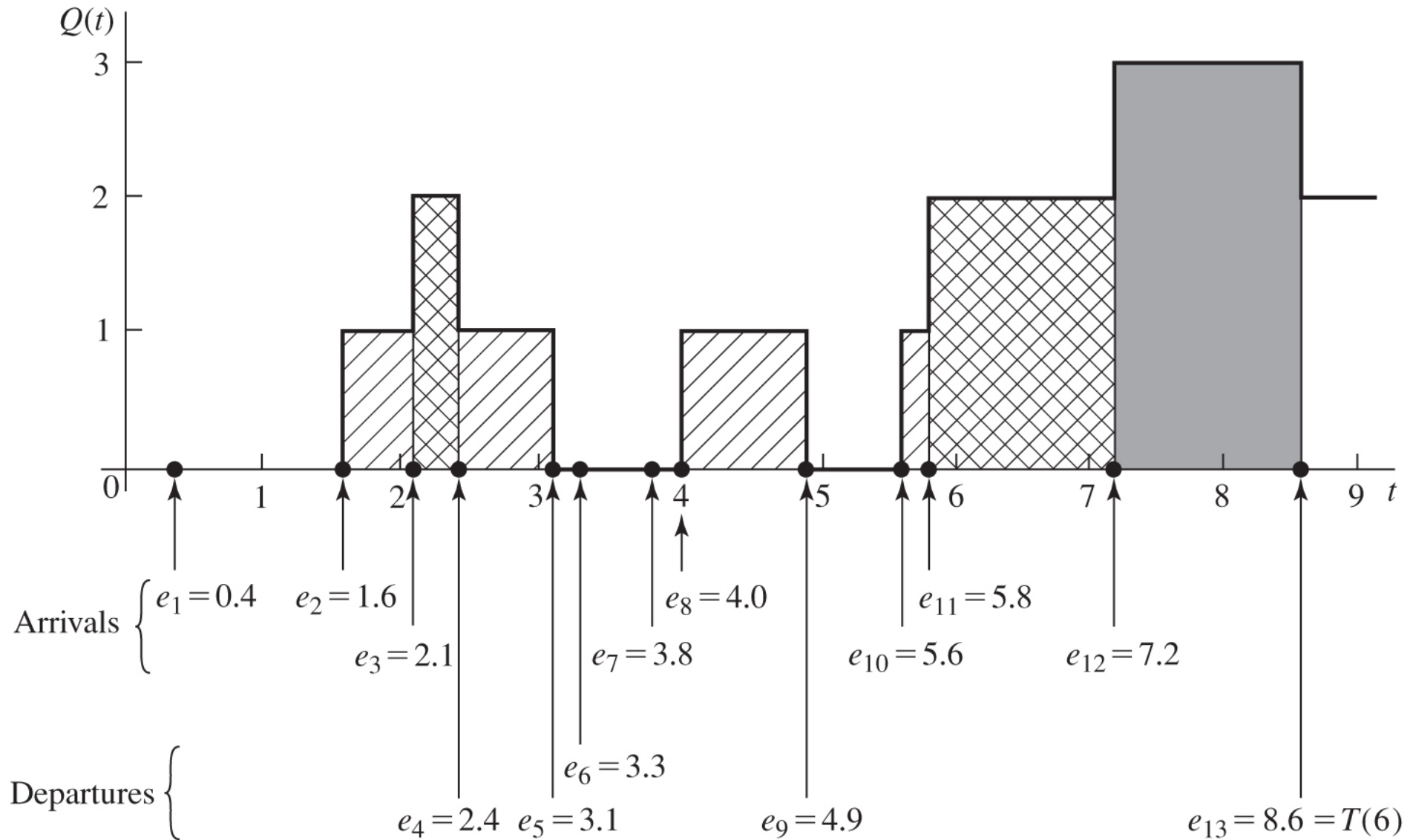
Final output performance measures:

Average delay in queue = $5.7/6 = 0.95$ min./cust.

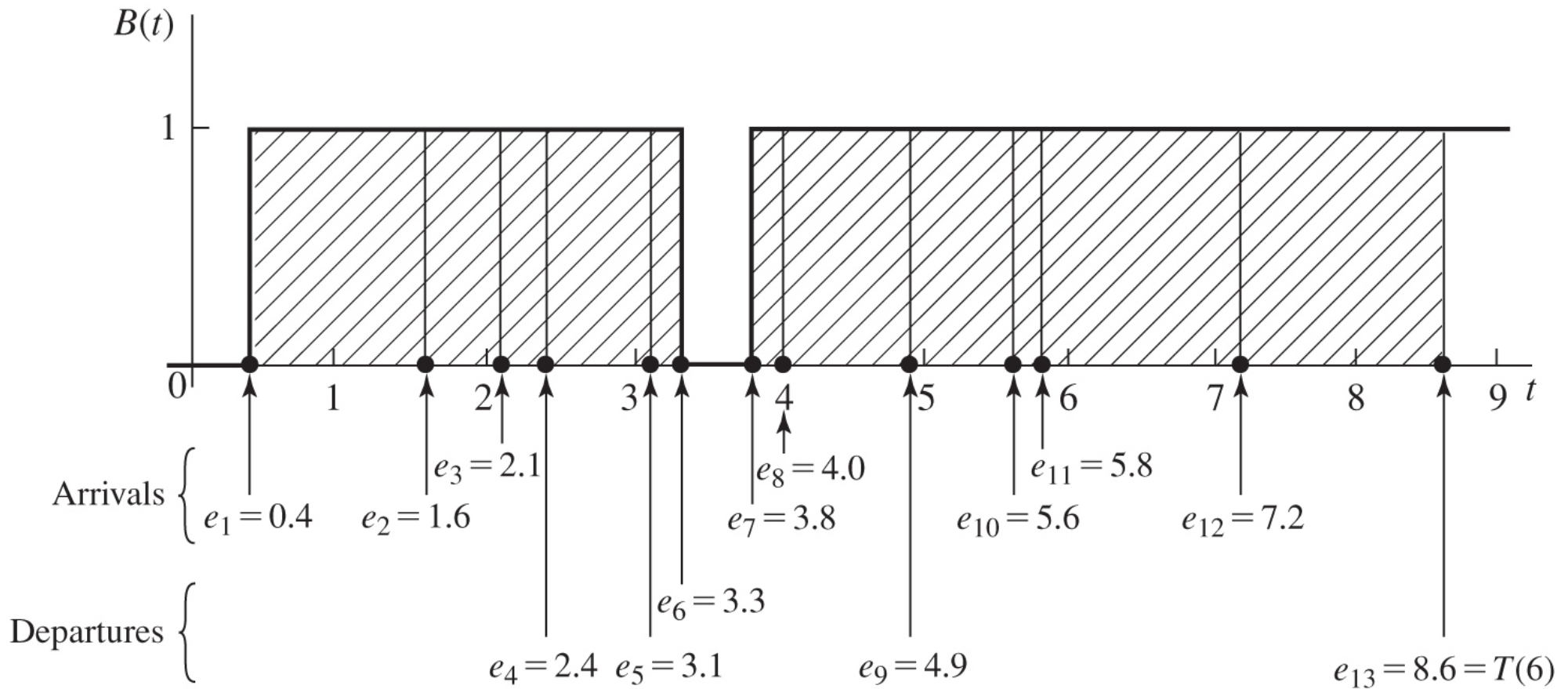
Time-average number in queue = $9.9/8.6 = 1.15$ custs.

Server utilization = $7.7/8.6 = 0.90$ (dimensionless)

Area under $Q(t)$

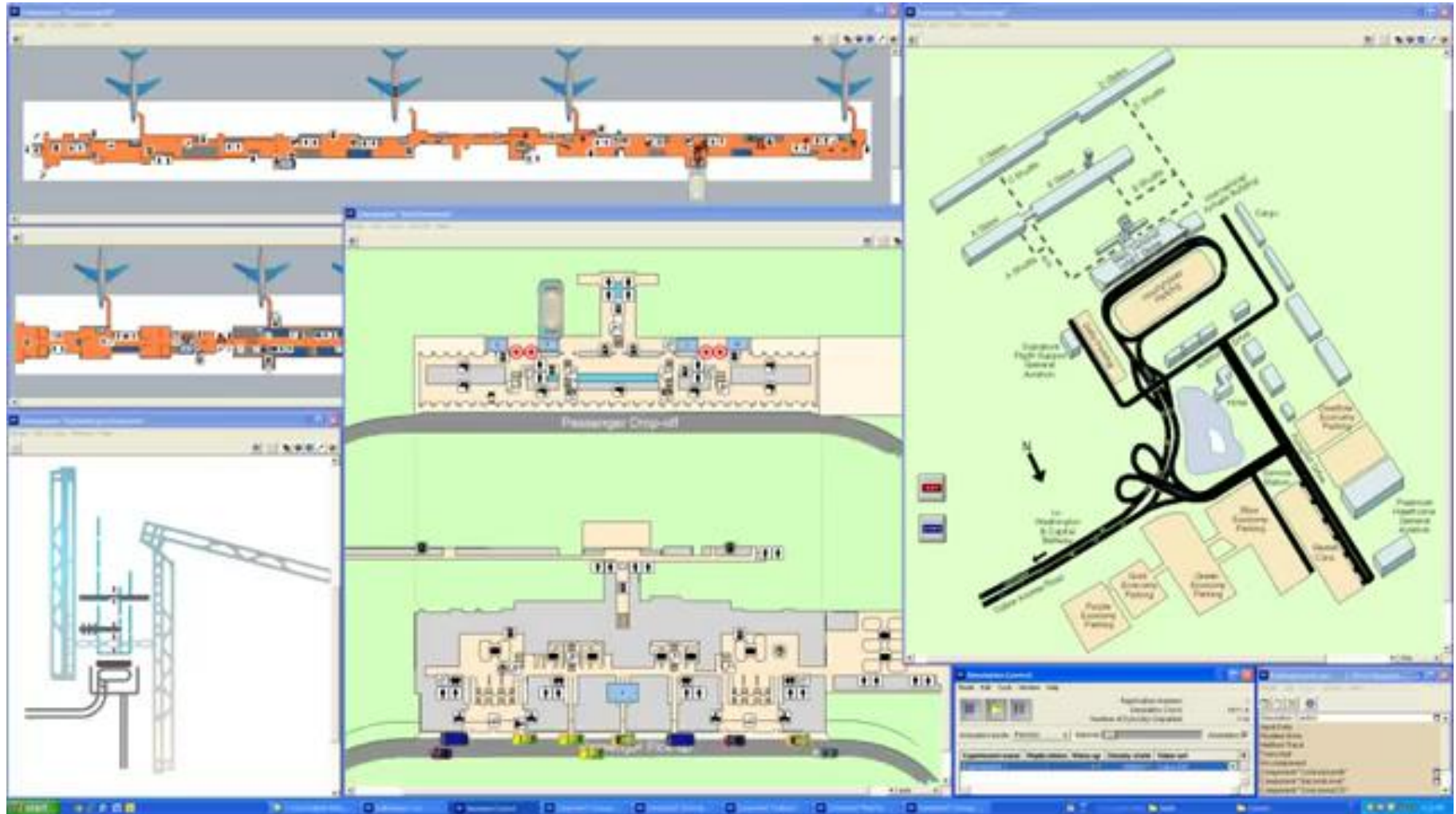


Area under $B(t)$



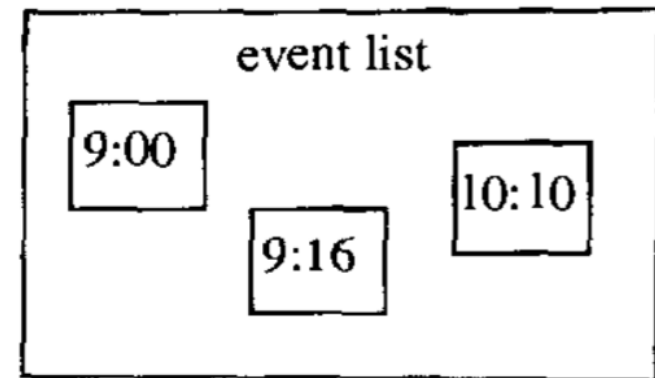
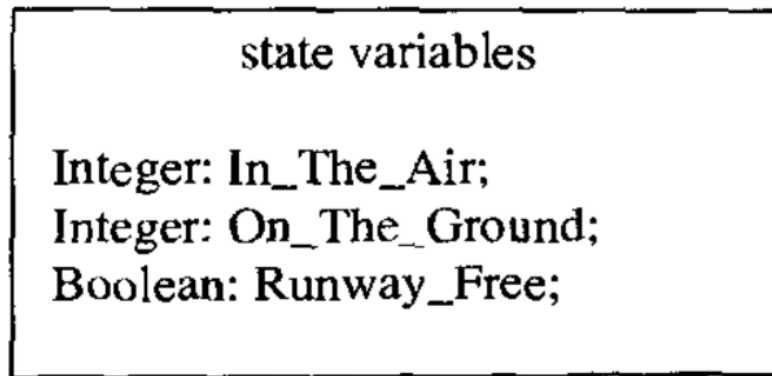
Simulation tool-based DES Example

Airport Simulation

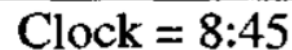


DES programs

- A *sequential* discrete event simulation program typically utilizes three data structures:
 - The *state variables* that describe the state of the system
 - An *event list* containing events that are to occur at some simulation time
 - A *global clock* variable to denote the instant on the simulation time axis at which the simulation now resides

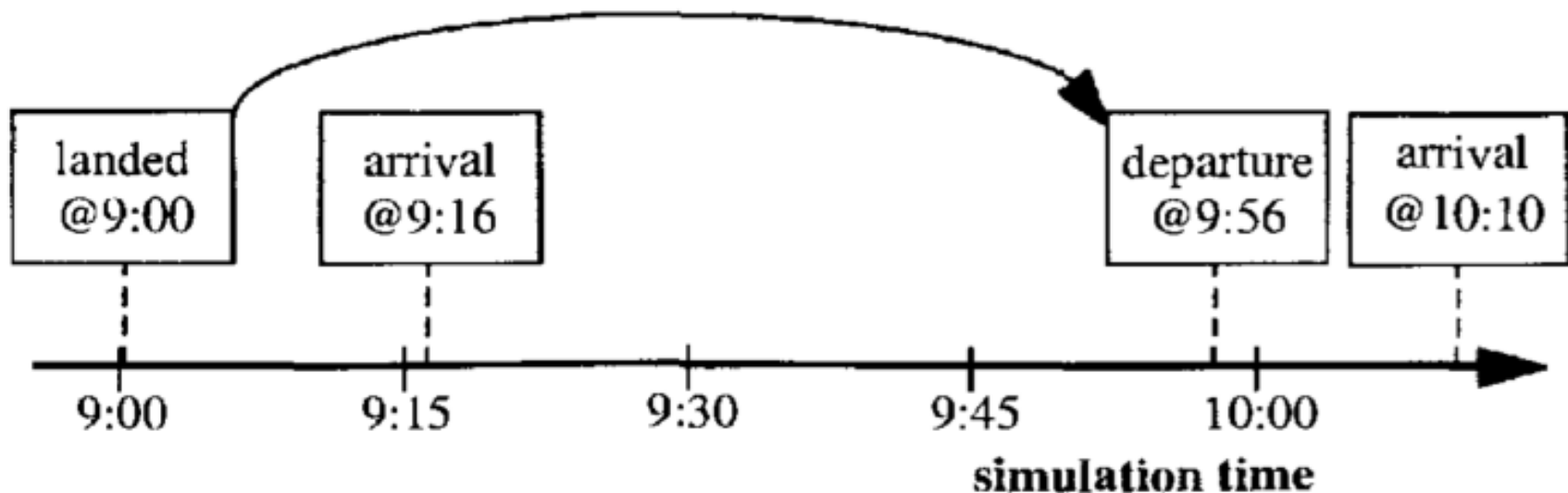


Clock = 8:45

A small rectangular box with a black border containing the text 'Clock = 8:45'.

Event scheduling

- In the physical system, "events" such as an aircraft arrival "just happen"
- In the simulated world, nothing happens unless the simulation computation makes it happen
- In other words, a mechanism is required to create new events
- The mechanism for creating a new event in the simulation is called *scheduling an event*



Event *scheduling*

- "*Scheduling an event*" in the simulator is implemented by:
 - allocating memory for a new event
 - filling in the fields for the time stamp, event type, and the associated parameters, and
 - adding the event to the event list data structure
- Event scheduling is one way that simulation programs can model causal relationships in the physical system

DES program

model of the
physical
system

Simulation Application

- state variables
- code modeling system behavior
- I/O and user interface software

calls to
schedule
events

calls to event
handlers

independent
of the
simulation
application

Simulation Executive

- event list management
- managing advances in simulation time

Event-oriented program

Event handler procedures

state variables

```
Integer: InTheAir;  
Integer: OnTheGround;  
Boolean: RunwayFree;
```

Arrival
Event

{

...

}

Landed
Event

{

...

}

Departure
Event

{

...

}

Simulation application

Simulation executive

Now = 8:45

Pending Event List (PEL)

9:00

9:16

10:10

Event processing loop

While (simulation not finished)

E = smallest time stamp event in PEL

Remove E from PEL

Now := time stamp of E

call event handler procedure

Simulation executive *main loop*

```
while (simulation is in progress)
    remove smallest time stamped event from
    event list
    set simulation time clock to time stamp of
    this event
    execute event handler in application to
    process event
```

- The event handler may do two things:
 1. modify state variables to model changes in the state of the physical system that result from this event
 2. schedule new events into the simulated future

Causality properties

- There are two important issues in a discrete event simulation that are worth highlighting, because they will become very important when we consider execution on parallel and distributed computers
 1. the **simulation application** can only schedule events into the simulated future, namely the time stamp of any new event must be at least as large as the current time of the simulation
 2. the **simulation executive** always processes the event containing the smallest time stamp next
- Both relate to ensuring that the simulation faithfully reproduces **causal relationships** in the physical system

DES program implementation

- A DES can in principle be executed by “hand”, but is usually implemented as a computer program
- Different types of DES implementations, with different characteristics in terms of flexibility and cost
 - use of a **general-purpose programming language (GPL)**, such as Java or C++ (highest flexibility, but requires a substantial technical know-how and a considerable extra development effort)
 - use of a **simulation package** (or **SPL – simulation programming language**) that provides the simulation executive with synchronization and communication mechanisms, such as Simjava or CSIM (loses some flexibility, while gaining a more cost-effective development)
 - use of a **high-level simulation tool**, such as Arena or SIMUL8 (no flexibility, best cost-effective compromise)