BASIC EVENT SCHEDULING SIMULATION

Now we need certain definitions. We use the single-server queue as an example.

System State - number of customers in queue: $Q=\{0,1,2,3,\ \}$
number of customers being served: $B=\{0,1\}$
(This list grows with entities and system measures studied.)

Event Set - that which can change the system state: $E=\{a, d\}$, where
a - an arrival
d - a departure (finish service)

Feasible Event Set - those events that are permitted to happen at a given state:
$\Gamma(Q,B)$
$\Gamma(k,0) = \{a\}$; $\Gamma(k,1) = \{a,d\}$ where $k=\{0,1,2,\ \}$

Simulation Clock - time counter within the simulation: t

Event Lifetimes - when the event lifetime counts down to zero the event happens:
Inter-arrival times: $A_1, A_2, A_3,$
Service times: $S_1, S_2, S_3,$
Event Time - time of event i: t_i

Event List - times when each feasible event will occur (given a system state and those events scheduled, but have not yet occurred)
$\{e_i,t_i|\ e_i \in \Gamma(Q,B)\}$ where $e_i = \{a,d\}$

Triggering Event - the most imminent event in the event list: e'

$$e' = \arg \min_{e_i \in \Gamma(Q,B)} \{t_i\}$$

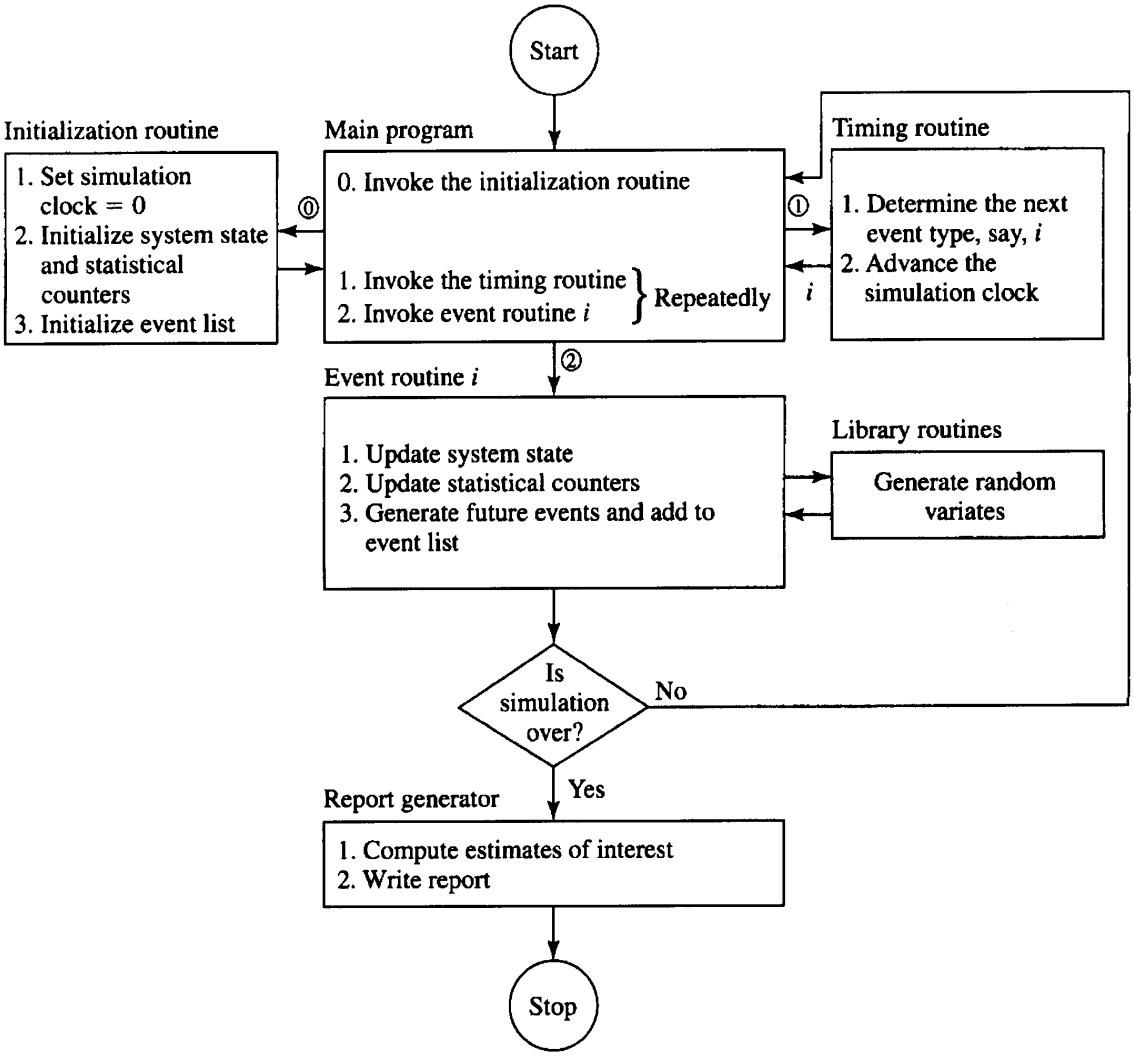
State Transition - allowed transitions between system states

(Q,B,e)	Q',B'	where e = a or d
(0,0,a)	0,1	customer goes right to service
(k,1,a)	k+1,1	where $k=\{0,1,2,3,\ \}$
(0,0,d)	impossible	
(0,1,d)	0,0	
(k,1,d)	k-1,1	where $k=\{1,2,3,\ \}$

Event Scheduling - The basic approach for most Discrete Event Simulations:
(Event scheduling (ES) is very effective in terms of computational effort.)

a) Start at t=0
b) Initialize system states
c) Find the most imminent event - e' and it's time t'
d) Advance the clock to t'
e) Gather statistics about system performance
f) Update the system states
g) Generate times of occurrences of future events
h) Go to 3 and continue.

Event Scheduling / Time-advance algorithm:

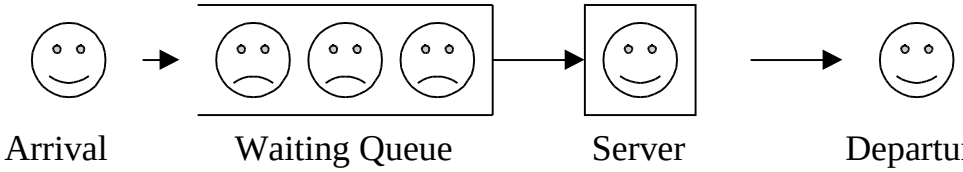


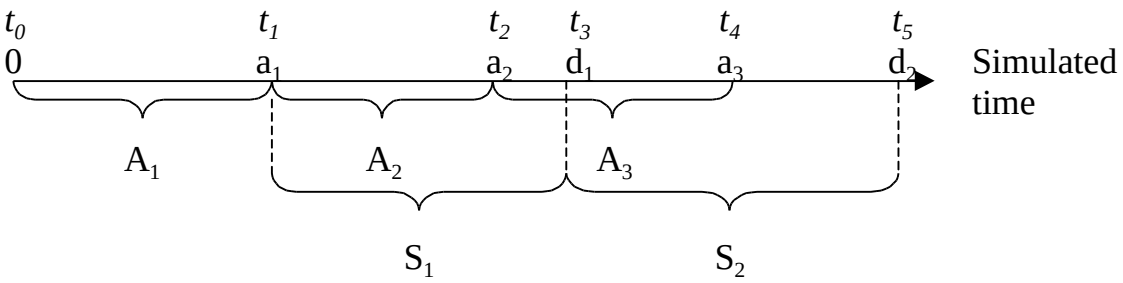
N.B. In the event routine, Update statistical counters should come first.

Alternatives to event scheduling:

a) **Process Interaction -** follows and entity throughout its life in the system. However, the underlying implementation is event scheduling.
b) **Activity Scanning -** uses fixed time increments. At each increment, all activities are scanned and if a scheduled event has occurred the state is adjusted accordingly.
c) **Three-phase -** combines event scheduling with activity scanning. Namely, event lists are kept according to activity rather than a master list.

Single-server queue system example (from Lecture 1):





Inter-arrival times (A): 0.4, 1.2, 0.5, 1.7, 0.2, 1.6, 0.2,⋯

Service times (S): 2.0, 0.7, 0.2, 1.1, 3.7

The simulation ends after the 5^{th} customer completes service.

If you recall, we did a hand simulation of this system using a 'process interaction' approach. We now apply a classical, event scheduling approach to the same simulation situation.

Simulation Outputs: Up till now we were not sure in what outputs we are interested. Knowing this is a necessity, however. For this example we decide that we are interested in the following three quantities: Expected average delay in queue, expected average number of customers in queue and observed proportion of time that the server is busy.

Expected average delay in queue: *d(n)* - This is a measure of the annoyance to customers of being kept waiting. n is the total number of customers that entered service in the simulation run. An obvious estimator of *d(n)* is

$$\hat{d}(n) = \frac{1}{n}\sum_{i=1}^{n} D_i \qquad \text{(The summation is Total delay)}$$

where D_i is the delay experienced by the *i*-th customer.

Thus, we need to keep track of arrival time of each customer, as well as the time that the customer begins service. The latter is either the departure time of the previous customer if the server is occupied or from the customer's arrival time if the server is empty.

Expected average number of customers in queue: *q(n)* - Again, *n* is the total number of customers that entered service during the simulation.

From probability theory,

$$q(n) = \sum_{i=0}^{\infty} i p_i$$

where p_i is the probability that there are exactly *i* customers in the queue. We can estimate p_i as follows:

$$\hat{p}_i = \frac{T_i}{T(n)}$$

where T_i is the time in the simulation that the queue is of length *i* and *T(n)* is the total time for the simulation. Thus a valid estimate for *q(n)* is:

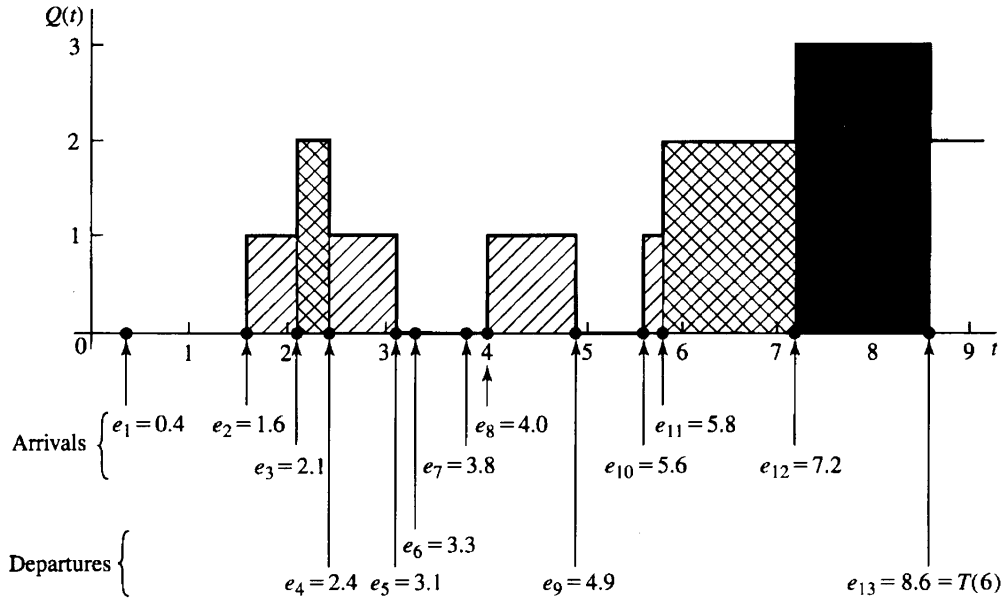
$$\hat{q}(n) = \sum_{i=0}^{\infty} \frac{i T_i}{T(n)}$$

or

$$\hat{q}(n) = \sum_{i=0}^{M} \frac{i T_i}{T(n)}$$

where *M* is larger than the maximum length of the queue during the simulation.

Lets look at the history of the queue length, *Q(t)* in terms of the events that occur during the simulation.



Here we see that:

$T_0 = (1.6 - 0.0) + (4.0 - 3.1) + (5.6 - 4.9) = 3.2$
$T_1 = (2.1 - 1.6) + (3.1 - 2.4) + (4.9 - 4.0) + (5.8 - 5.6) = 2.3$
$T_2 = (2.4 - 2.1) + (7.2 - 5.8) = 1.7$
$T_3 = (8.6 - 7.2) = 1.4$
$T_4 = T_4 = T_5 = \cdots = 0$

So that $T(n) = 3.2 + 2.3 + 1.7 + 1.4 = 8.6$

Thus, the estimate of $q(n)$ becomes

$$\hat{q}(n) = \frac{1}{T(n)} \sum_{i=0}^{M} iT_i$$

$$= \frac{1}{8.6}(0(3.2) + 1(2.3) + 2(1.7) + 3(1.4))$$

$$= \frac{9.9}{8.6} = 1.15$$

To implement this calculation requires some thought. Clearly, the $\{T_I\}$ are not known until after the simulation is completed. One way to handle this is to allocate M registers that would add up the $\{T_I\}$ contributions. But, this is foolish. One register suffices if at every t_I we add up the product $i \cdot (t_i - t_{i-1})$. At the end of the simulation, the contents of the register would indeed be

$\sum_{i=0}^{M} iT_i$ which is easily seen to be the area under $Q(t)$

It is necessary at the end of the simulation to divide by *T(n).*

Expected utilization of the server: *u(n)* - A similar method is suggested. Thus, we are interested in

$$\hat{u}(n) = \frac{1}{T(n)} \sum_{i=0}^{1} iT_{si}$$ (The summation is the area under *B(t).*)

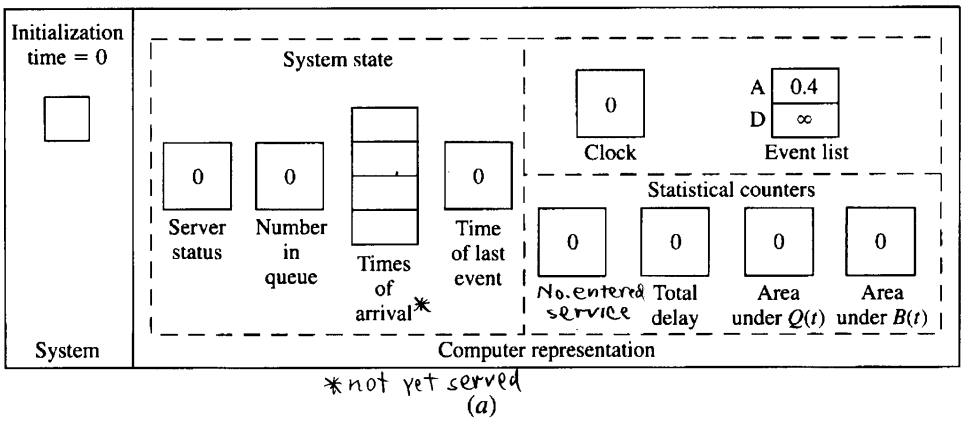
where T_{s1} is the total time that the server is busy.

This can also be accumulated in one register which adds up the product $i \cdot (t_i - t_{i-1})$, where $i = 1$ when the server has been busy and $i = 0$ when not.

Let's now go through the discrete event/time advance simulation of the single-server queue by hand.

Remember that the inter-arrival and service times are given by:
Inter-arrival times (A): 0.4, 1.2, 0.5, 1.7, 0.2, 1.6, 0.2, 1.4, ⋯
Service times (S): 2.0, 0.7, 0.2, 1.1, and 3.7
The simulation ends after the 5th customer completes service.

We first initialize the computer representation:



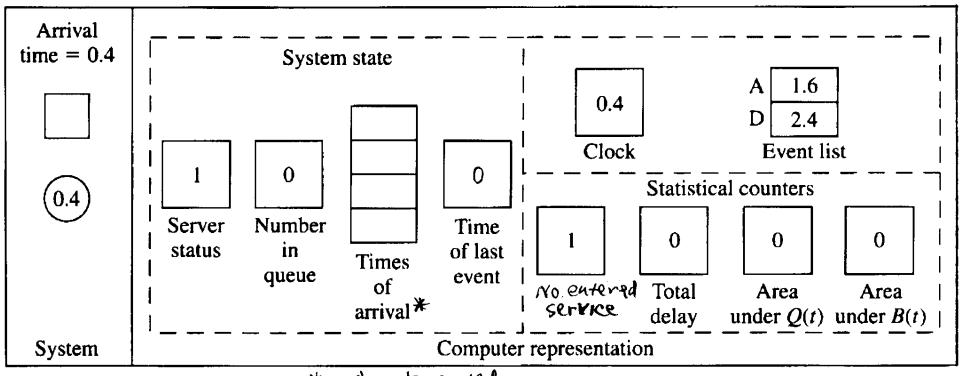
(*a*)

The hard part is the statistical counters. These must be updated prior to updating the System State.

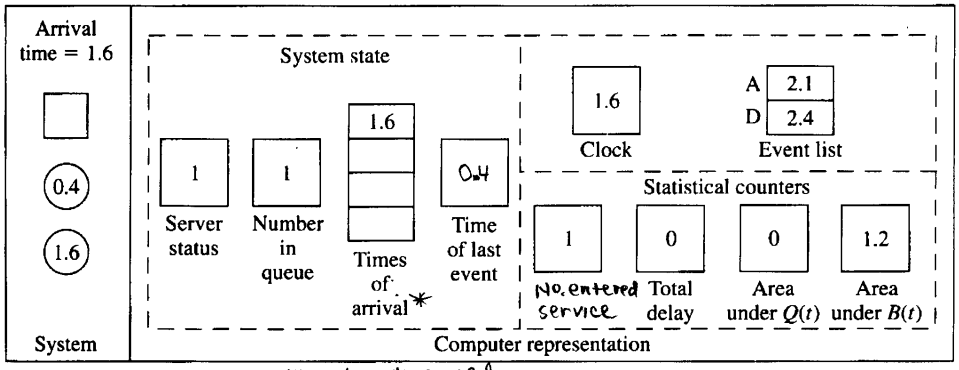
Number of customers entered service is used to determine average time in queue. To update this quantity we add the quantity one when there is either

a) an arrival and the <u>previous</u> server status is zero, or

b) a departure and the <u>previous</u> queue not empty.



(*b*)



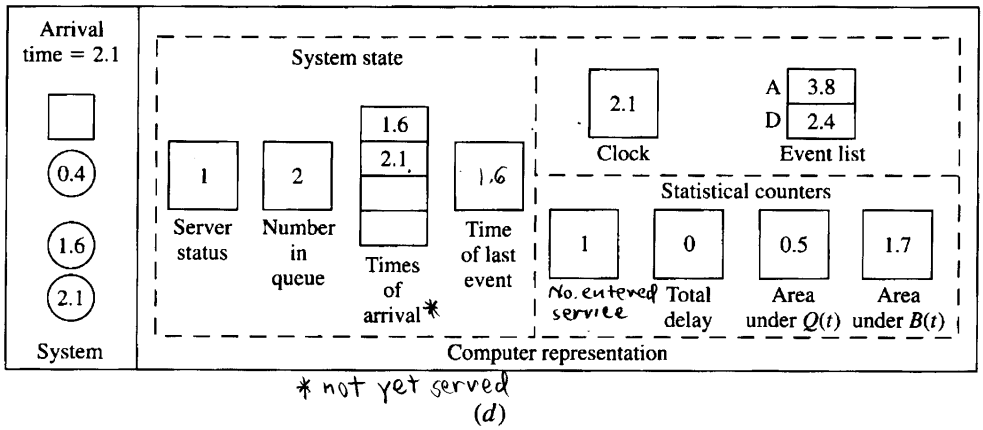
(*c*)

Area under *B(t)* - needed for the calculation of server utilization is updated by adding the following quantity at every event:

(Previous server status) x (Current clock - Previous clock)

Area under *Q(t)* - needed for the calculation of average queue length is updated by adding the following quantity at every event:

(Previous no in queue) x (Current clock - Previous clock)

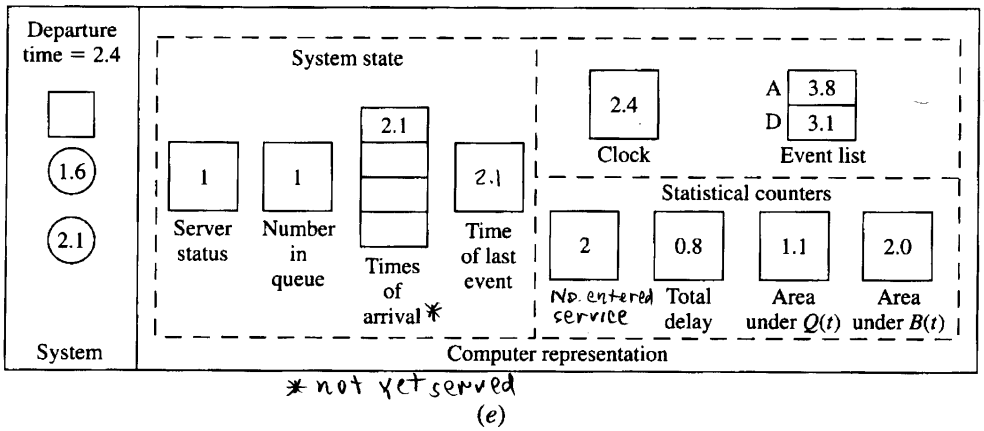


(*d*)

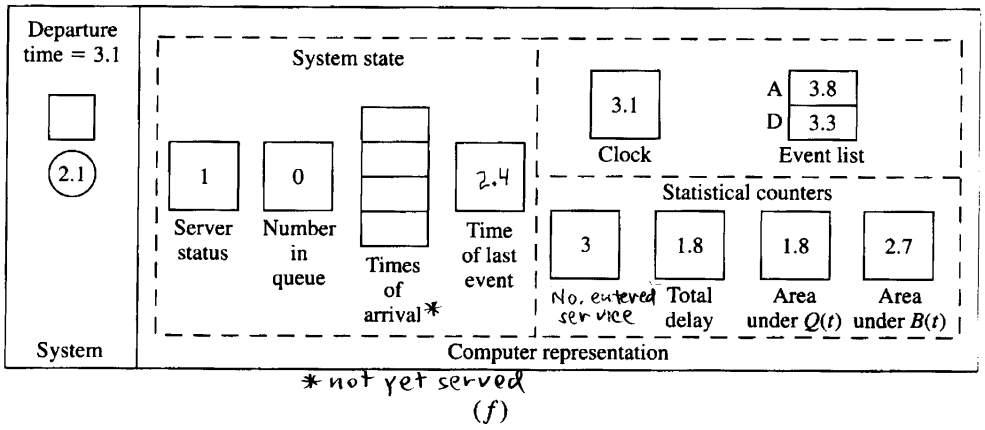
Total delay - is needed in the calculation of expected average delay in queue. It is updated only at a departure. If and only if the previous queue was not empty, we add the quantity

(Current clock - time of <u>previous</u> earliest arrival not yet served)

Why do we skip the situation where the previous queue was empty? In this case, the departure is not clearing the way for a customer to begin service, which is <u>really</u> the statistic we are collecting.



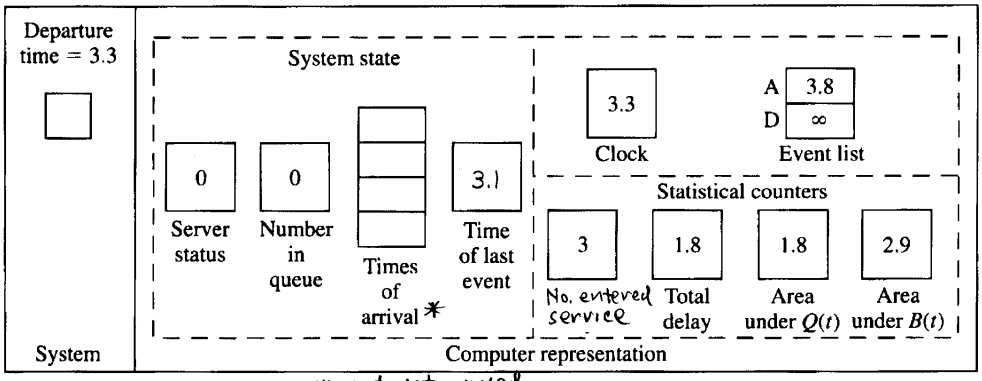
(e)



(f)

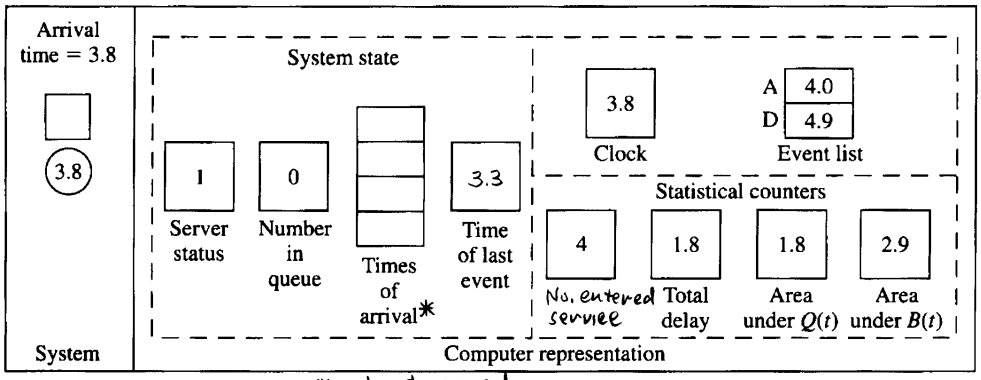
Why are we being so meticulous? We are being meticulous because we will have to be capable of programming a simulation and because we need to develop habits which will permit us to avoid two evils:

1. mathematical errors
2. unnecessary allocation of memory.



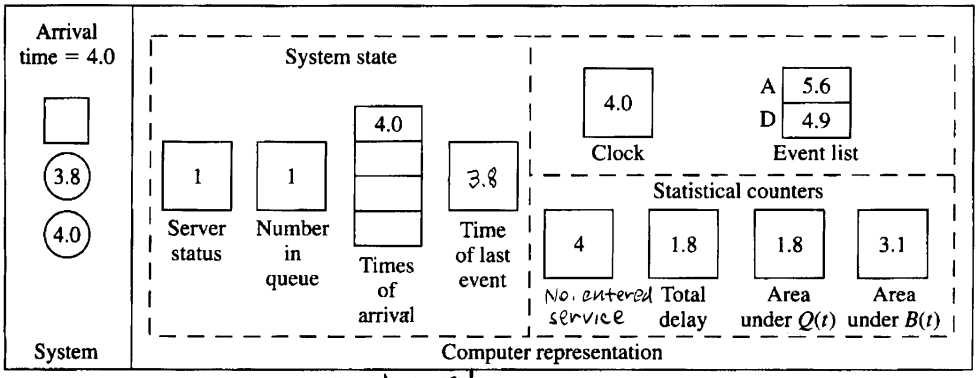
* not yet served

(g)



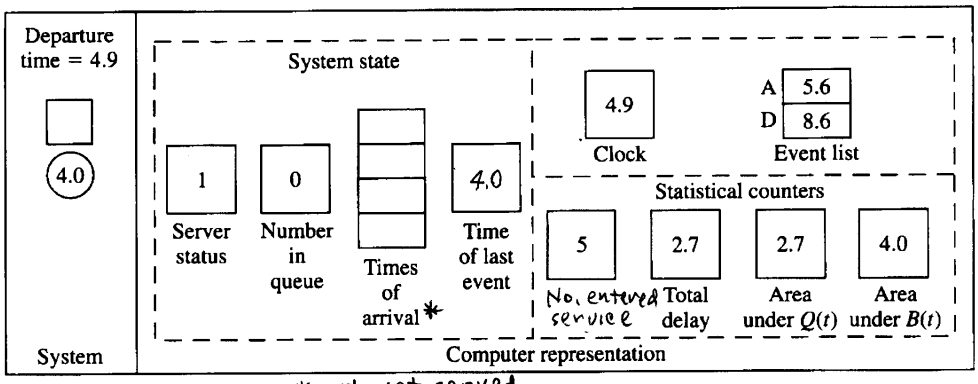
* not yet served

(h)

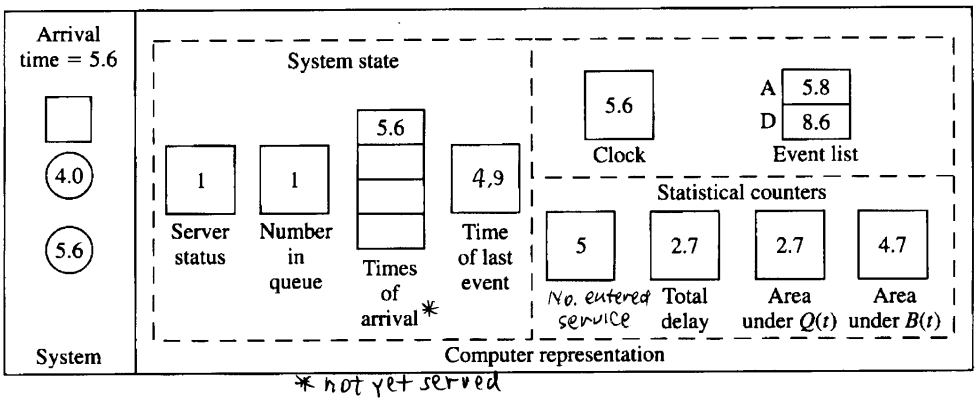


* not yet served

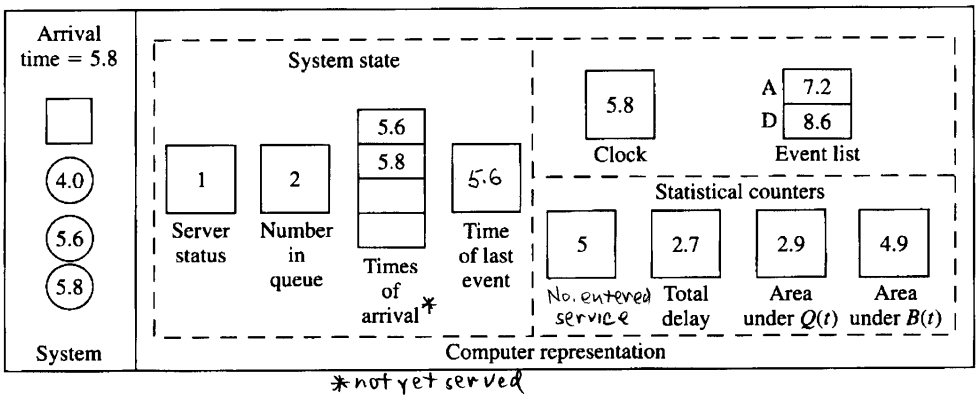
(i)


Departure
time = 4.9
4.0
System
System state
1
Server status
0
Number in queue
Times of arrival *
4.0
Time of last event
4.9
Clock
A 5.6
D 8.6
Event list
Statistical counters
5
No. entered service
2.7
Total delay
2.7
Area under Q(t)
4.0
Area under B(t)
Computer representation
* not yet served

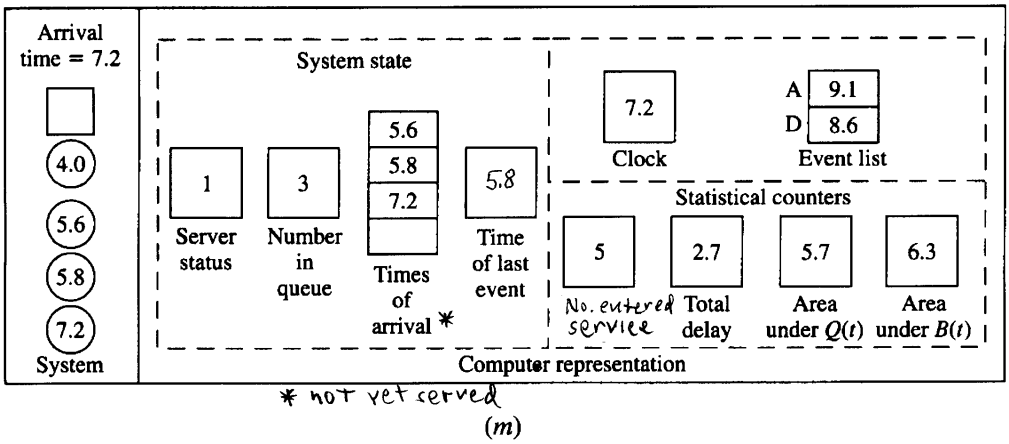
(j)


Arrival
time = 5.6
4.0
5.6
System
System state
1
Server status
1
Number in queue
5.6
Times of arrival *
4.9
Time of last event
5.6
Clock
A 5.8
D 8.6
Event list
Statistical counters
5
No. entered service
2.7
Total delay
2.7
Area under Q(t)
4.7
Area under B(t)
Computer representation
* not yet served

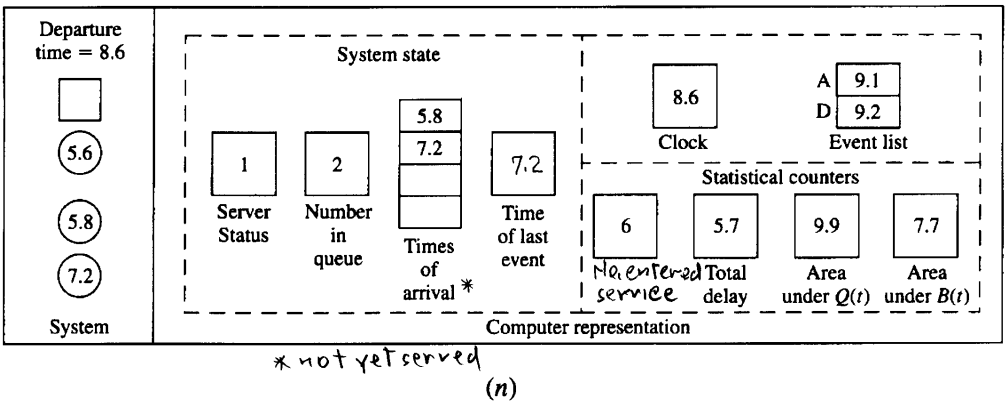
(k)


Arrival
time = 5.8
4.0
5.6
5.8
System
System state
1
Server status
2
Number in queue
5.6
5.8
Times of arrival *
5.6
Time of last event
5.8
Clock
A 7.2
D 8.6
Event list
Statistical counters
5
No. entered service
2.7
Total delay
2.9
Area under Q(t)
4.9
Area under B(t)
Computer representation
* not yet served

(l)



(*m*)



(*n*)

Final output measures:

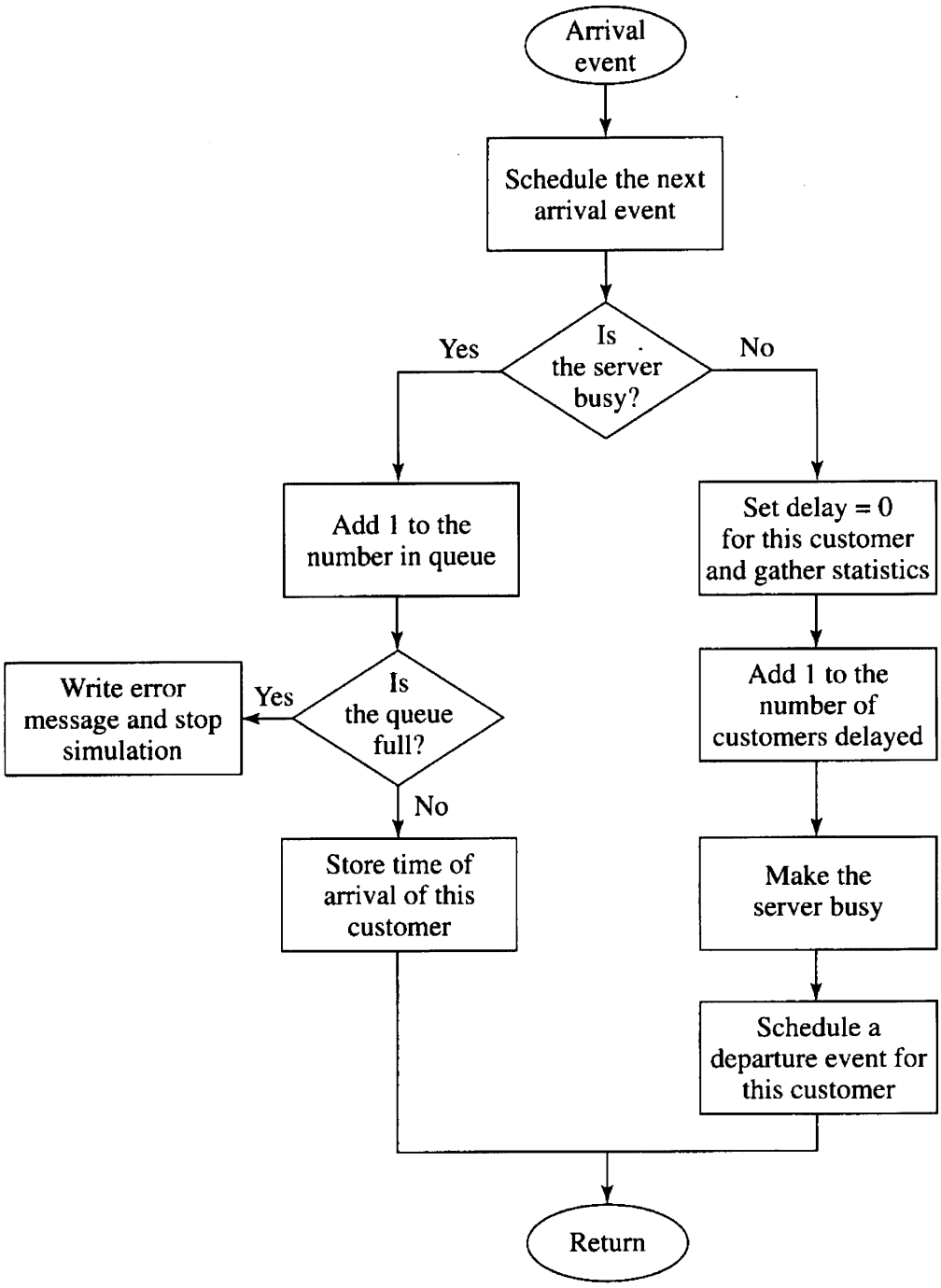
Average delay in queue = Total delay / Number of customers that entered service
= 5.7 / 6 = 0.95 (whatever units you used)

Expected average queue length = Area under Q(t) / Total simulation time
= 9.9 / 8.6 = 1.15

Proportion of time server is busy = Area under B(t) / Total simulation time
= 7.7 / 8.6 = 0.9

Programming a simulation: Recall the flow chart for the Discrete Event/Time Advance Simulation. The 'Event Routine' can in fact be multiple routines. For our example of a single-server queue system, there are two:

Flowchart for arrival event routine:

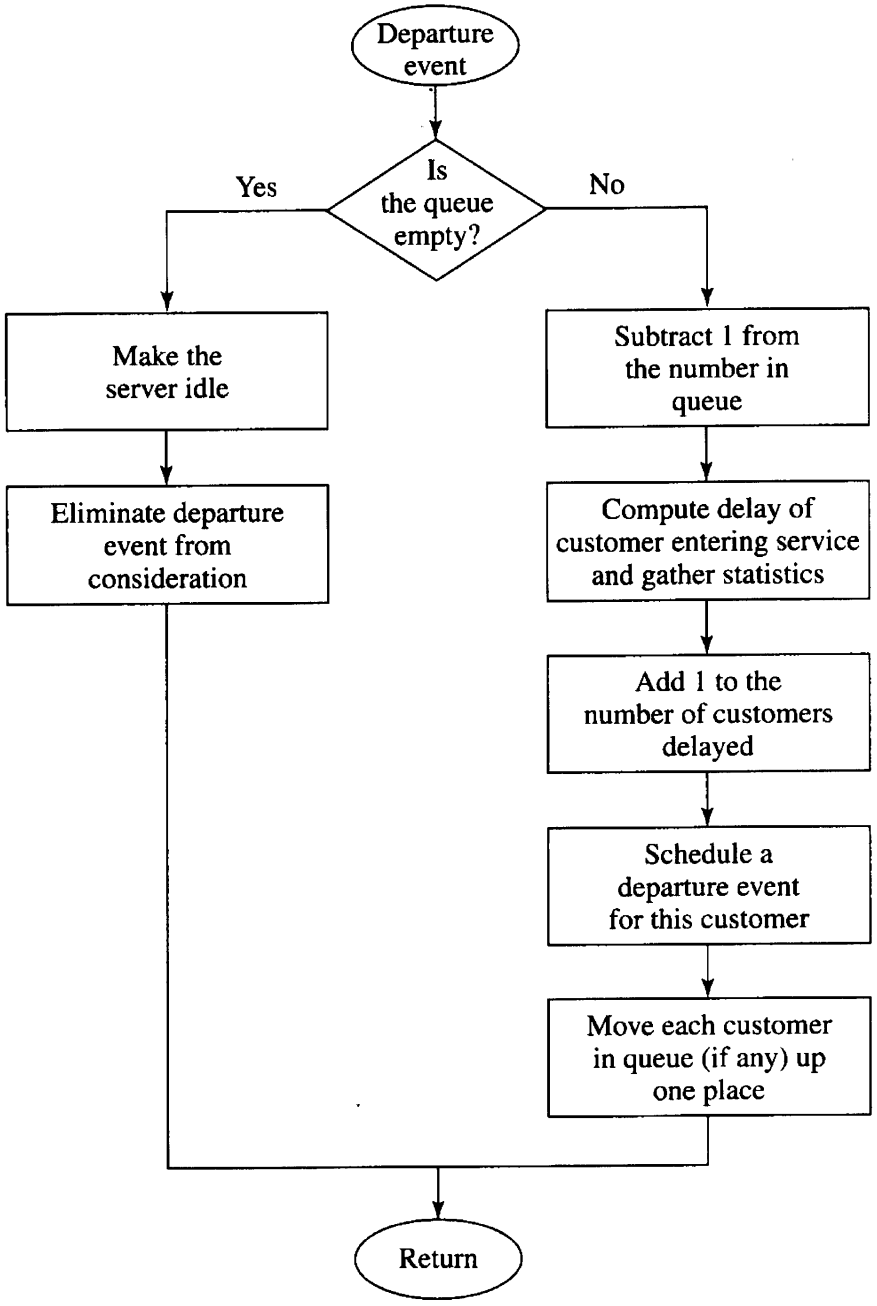


N.B. There are several misleading entries in the Law and Kelton charts:

a) IMPORTANT! Before you 'Add 1 to the number in queue' you must 'gather statistics'.

b) 'Number of customers delayed' means 'Number of customers that entered service'.

Flowchart for departure event routine:



N.B. There are several misleading entries in the Law and Kelton charts:

c) IMPORTANT! Even if the queue is not empty you must gather statistics.

d) 'Number of customers delayed' means 'Number of customers that entered service'.

e) Don't 'Subtract 1 from the number in queue' until after you 'Compute delay of customer entering service and gather statistics'.

f) Schedule a departure event for 'next' not for 'this' customer.

In the flowchart for the arrival event the gathering of statistics could have been done before the deciding whether or not the server is busy. A similar situation exists in the departure event flowchart. However, one must be careful that, if the queue is empty before the departure event, one does not update the total delay.

In the single-server queue simulation C code provided by Law and Kelton in their book, *Simulation Modeling and Analysis* statistics are gathered prior to entering the arrival or departure event routines. This code is available on their web site: http://www.mhhe.com/lawkelton

It is recommended that you get a copy of Law/Kelton single-server queue simulation C-code and bring it to the next lecture. Also, be sure to review carefully the single-server queue example of this lecture. The C-code differs from the lecture example, in that arrival and service times are randomly generated events rather than given as inputs. The method of generating such random events will be covered in later lectures.

HOMEWORK #2

1) For the single-server queuing system of this lecture, define $L(t)$ to be the *total* number customers in the system at time t. (including the queue and the customer in service at time t, if any).

a) Is it true that $L(t) = Q(t) + 1$? Why or why not?
b) For the same realization of the hand simulation in this lecture, make a plot of $L(t)$ vs t (similar to the plot of $Q(t)$ on page 5) between times 0 and $T(6)$.
c) From your plot in part b), compute $\hat{\ell}(6)$= the time-average of customers in the system during the time interval $[0,T(6)]$. What is $\hat{\ell}(6)$ estimating?
d) Augment the 14 figures of the computer representation of the hand simulation to indicate how $\hat{\ell}(6)$ is computed during the course of the simulation.

2) For the single-server queuing system of this lecture, suppose that we did not want to estimate the expected average delay in queue; the model's structure and parameters remaining the same. Does this change the state variables of the system? If so, how?