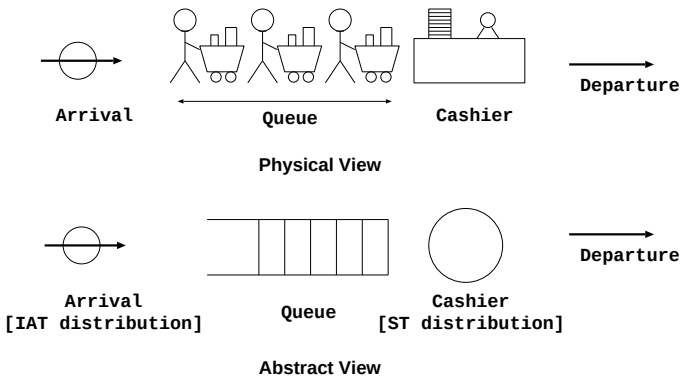
Discrete Event Modelling and Simulation

- Model : objects and relationships among objects
- Object : characterized by attributes to which values can be assigned
- Attributes:
  - indicative
  - relational
- Values: of a type

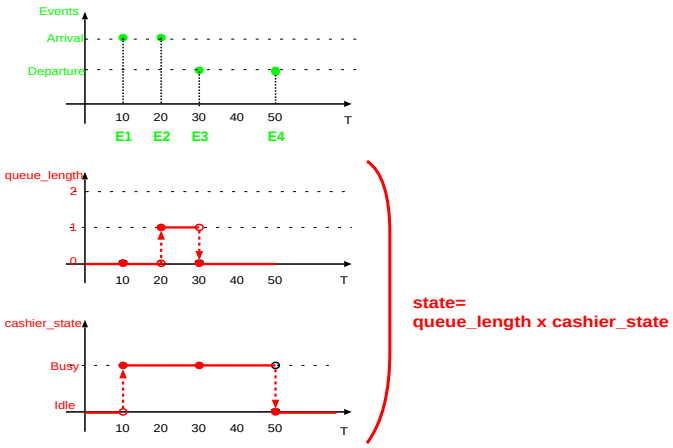
Time and State Relationships

- Indexing Attribute: enables state transitions
  Time is most common.
- Instant: value of System Time at which the value of at least one attribute of an object can be assigned.
- Interval: duration between two successive instants.
- Span: contiguous succession of one or more intervals.
- State of an object: enumeration of all attribute values at a particular instant.
- State of the system: all object states as a particular instant.

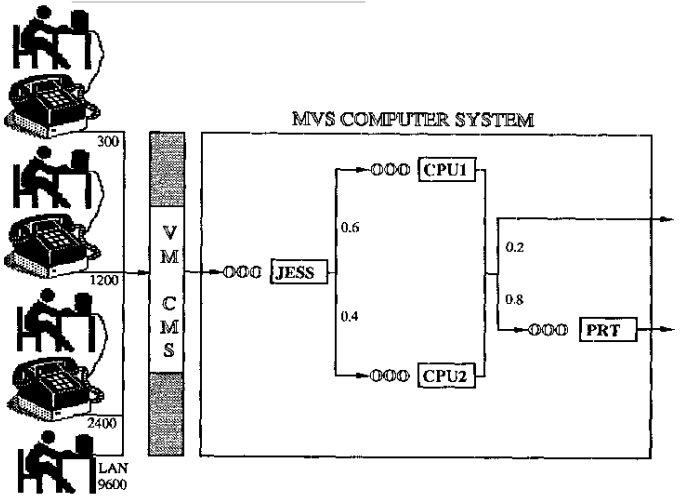
Single Server Queueing System



Queueing System State Trajectory



Example Problem



Example Parameters

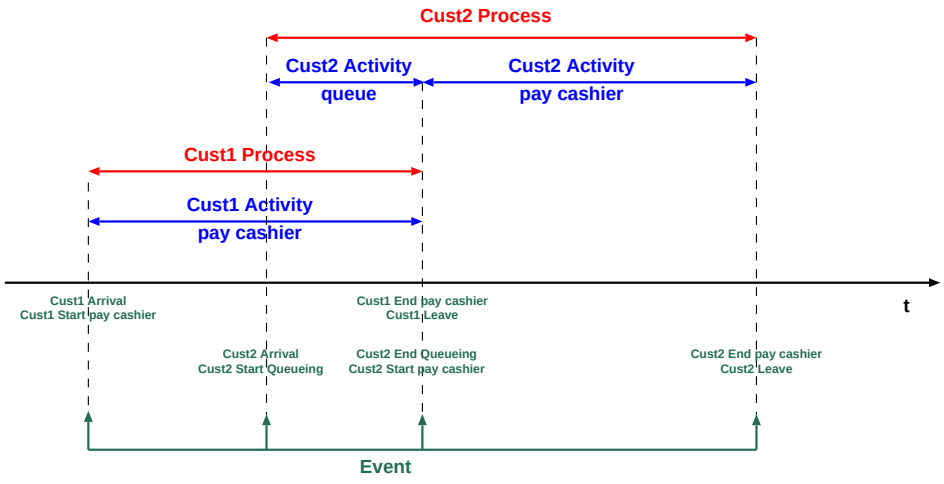
Table 1		
Type of User	*Interarrival Times*	*Mean*
Modem 300 User	Exponential	3200 Seconds
Modem 1200 User	Exponential	640 Seconds
Modem 2400 User	Exponential	1600 Seconds
LAN 9600 User	Exponential	266.67 Seconds

Table 2		
Facility	*Processing Times*	*Mean*
JES Scheduler	Exponential	112 Seconds
Processor 1 (CPU1)	Exponential	226.67 Seconds
Processor 2 (CPU2)	Exponential	300 Seconds
Printer (PRT)	Exponential	160 Seconds

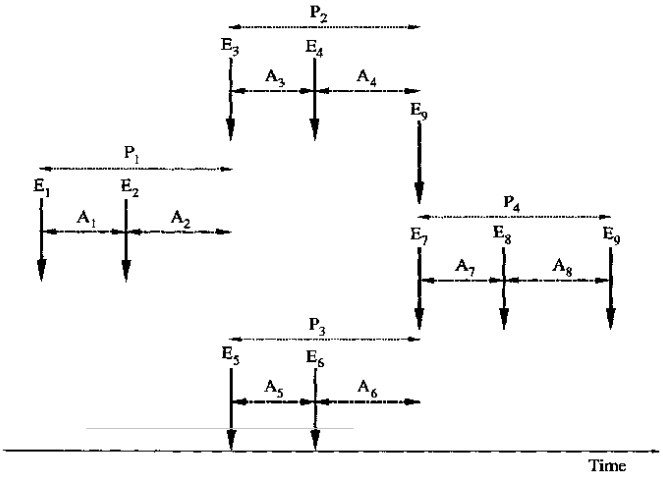
Time and State Relationships

- Activity: state of an object over an interval
- Event: change in object state, occurring at an instant. Initiates an activity.
  - Determined: occurrence based on time
  - Contingent: system conditions
- Object activity: state of object between two events for that object.
- Process: succession of states of object over a span

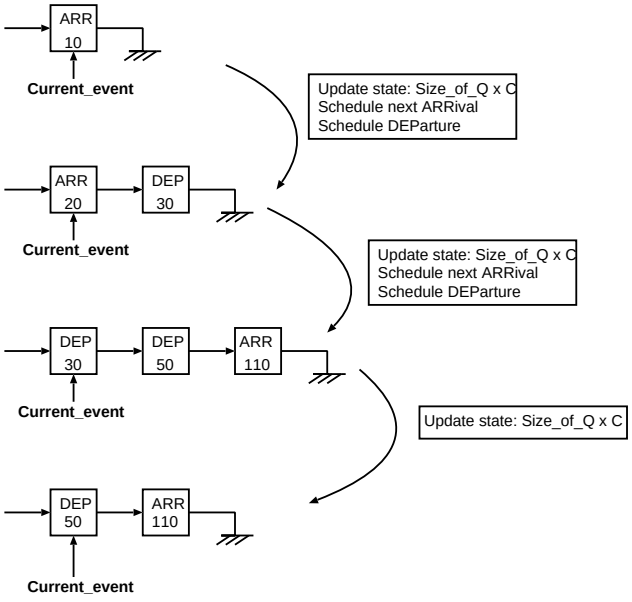
Event/Object Activity/Process



Event vs. Activity vs. Process



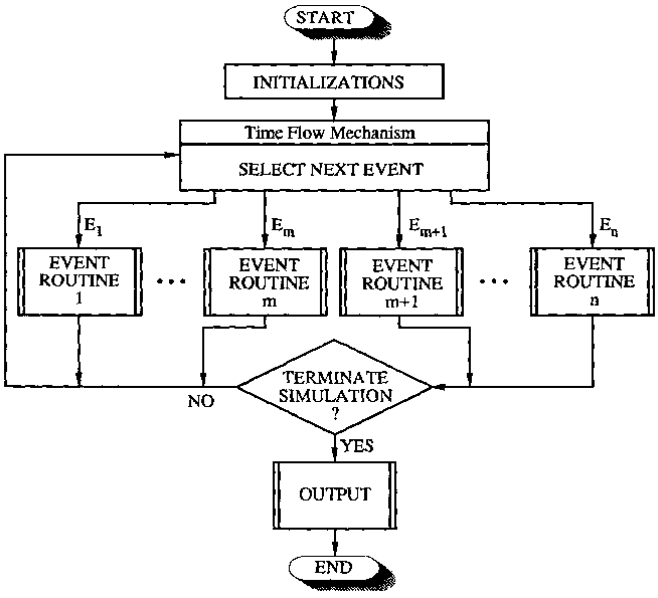
Cashier-queue Event List



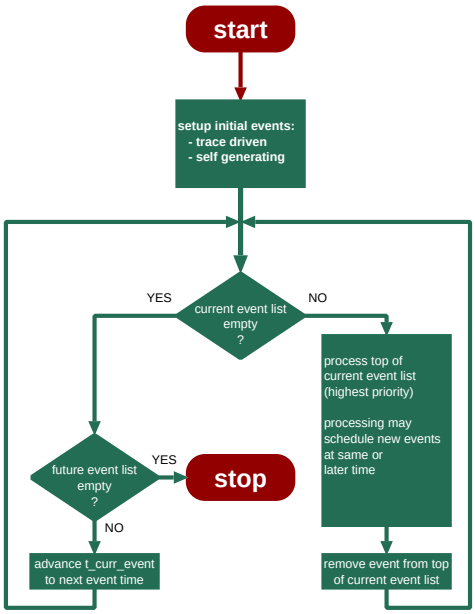
Event Scheduling

- Identify objects and attributes
- Identify attributes of the system
- Define what causes changes in attribute value as *event*
- Write event routine for each event
- Follow event scheduling logic

Event Scheduling Kernel (1)



Event Scheduling Kernel (2)



Input Generation

- Trace driven
- Auto generating (a model)

Cashier-queue Event Scheduling Model

```
declare (and initialise) variables:
 queue_length in PosInt = 0
 cashier_state in {Idle, Busy} = Idle

declare events:
 arrival
 departure

define events:

arrival event
 schedule arrival relative Random(mean, spread)
 if (queue_length == 0)
  if (cashier_state == Idle)
   cashier_state = Busy
   schedule departure relative Random(mean, spread)
  else
   queue_length++
 else /* queue_length != 0 */
  queue_length++

departure event
 if (queue_length == 0)
  cashier_state = Idle
 else /* queue_length != 0 */
  queue_length--
  schedule departure relative Random(mean, spread)
```

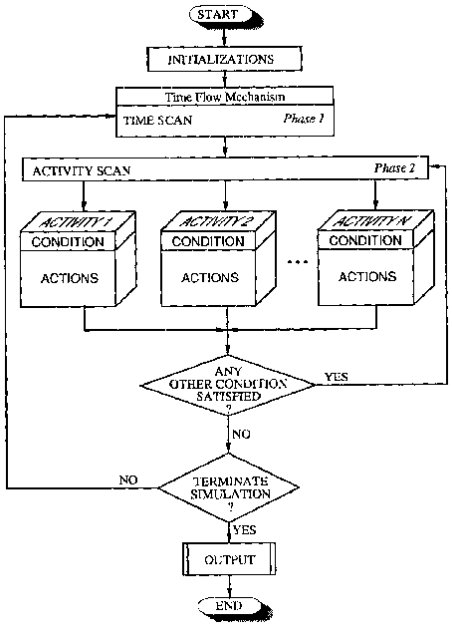

Termination Conditions

- Empty Event List
  Need to stop generating arrivals after t_{end} when auto-generating arrivals
- Schedule Termination Event (*caveat*)
  - process statistics
  - cleanup
  - stop
- Similarly: schedule initialisation/setup

Activity Scanning (rule-based)

- condition: must be satisfied for activity to take place.
  Becomes true *only* at event times.
- actions: operations performed when condition becomes true

Activity Scanning



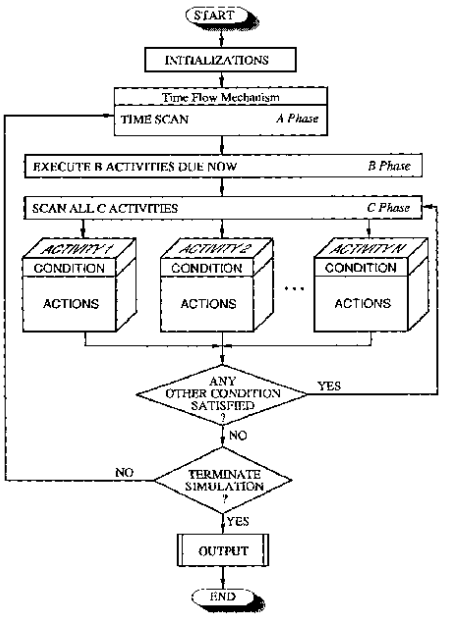
Cashier-queue Activity Scanning Model

```
declare (and initialise) variables:
 queue_length in PosInt = 0
 cashier_state in {Idle, Busy} = Idle
 t_arrival = 0, t_depart  = plusInf

define conditions:

arrival condition: t >= t_arrival
 if (queue_length == 0)
  if (cashier_state == Idle)
   keep queue_length == 0
   cashier_state = Busy
   t_depart = t + Random(mean, spread) /* service time */
  else
   queue_length++
 else /* queue_length != 0 */
  queue_length++, keep cashier_state == Busy
 t_arrival = t + Random(mean, spread)   /* inter arrival time */

departure condition: t >= t_departure
 if (queue_length == 0)
  cashier_state = Idle
 else /* queue_length != 0 */
  queue_length--, keep cashier_state == Busy
  t_depart = t + Random(mean, spread) /* service time */
```

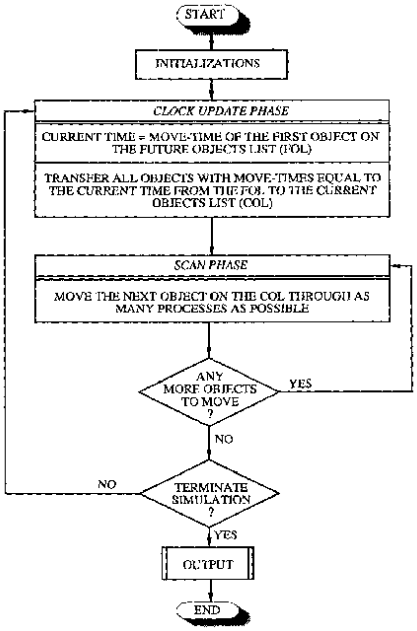

Three Phase Approach

- Bound to occur activities: unconditional state changes. Pre-scheduled.
- Conditional activites

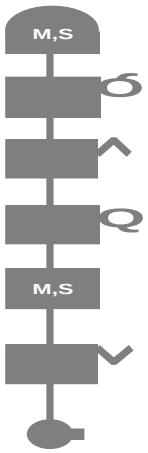
Three Phase Approach



Process Interaction

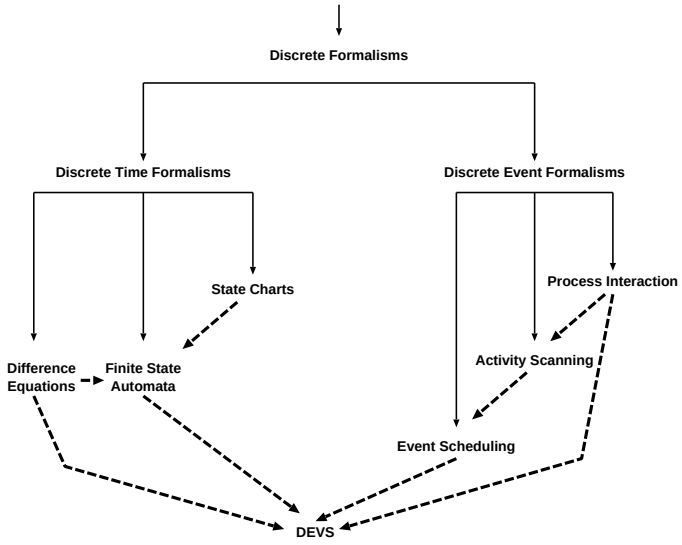


Cashier-Queue: GPSS Process Interaction View




```
GENERATE  10, 5
QUEUE     wait
SEIZE     cashier
DEPART    wait
ADVANCE   5, 3
RELEASE   cashier
TERMINATE 1
```


World Views: Classification



(Pseudo-) Random-number Generators

- SYS model is deterministic + random constructs
- randomness $\equiv$ not enough detail known or don't care
- randomness: characterized by distribution
- In SYS: *draw* from distribution and Monte-Carlo run multiple deterministic simulations.
- Alternative: Markov Chains (analytical).

Probability Distributions

- Continuous vs. discrete
- Probability Density Function ($f(x)$)
- Cumulative Probability Function ($F(X)$)
- see probability course: Poisson, Erlang, ...

Pseudo-random

- Sample from distribution ($U(0,1)$)
- Reproducability/comparison of experiments !
  - science needs reproducable results
  - makes debugging easier
  - *identical* random numbers to compare *different* systems
- Quality of generator:
  - appear uniformly distributed
  - non-correlated
  - fast and doesn't need much storage
  - long period, dense (full) coverage
  - provision for *streams* (subsegments)

Linear Congruential Generators

$$Z_i = (aZ_{i-1} + c) mod\ m$$

m is modulus
a is multiplier
c is increment
Z_0 is seed
$c = 0$ is called *multiplicative* LCG

Generators ctd.

- Composite Generators
- Tausworthe generators (operate on bits)
- L'Ecuyer, Devroye (non-uniform)
- Testing RNG: empirical vs. theoretical
- References: Knuth, Law & Kelton

Marse and Roberts' portable RNG

$$Z[i] = (630360016 * Z[i-1]) mod (2^{31} - 1)$$

- Prime modulus multiplicative linear congruential generator.
- Based on Fortran UNIRAN code.
- Multiple (100) streams are supported with seeds spaced 100,000 apart.
- Include file: rand.h
- C file: rand.c
- Example use: randtest.c

Non-uniform continuously distributed RNG

Inverse Transformation Method

Gathering Statistics (report generation)

1. *counters*
2. *summary measures*
3. *utilization*
4. *occupancy*
5. *distributions* and *transit times*

Counters

In all previous examples: keep/update counters !

- numbers of entities of different types in the system
- number of times a particular event occurred
- basis for statistics (performance metrics)

Summary Measures

- minima and maxima:
  compare new values to current *min* and *max*, update when necessary
- mean of a set of N observations $x_i, i = 1, 2, \ldots, N$

$$m = \frac{1}{N} \sum_{i=1}^{N} x_i$$

Summary Measures (ctd.)

- standard deviation (from mean)

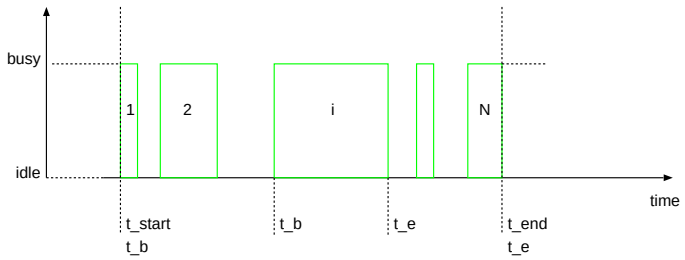
$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(m - x_i)^2}$$

  - need to calculate m first $\rightarrow$ need to keep all observations
  - sum of squares may grow *very* large (accuracy $\downarrow$)

$$\sum_{i=1}^{N}(m - x_i)^2 = \sum_{i=1}^{N} x_i^2 - Nm^2$$

Utilization

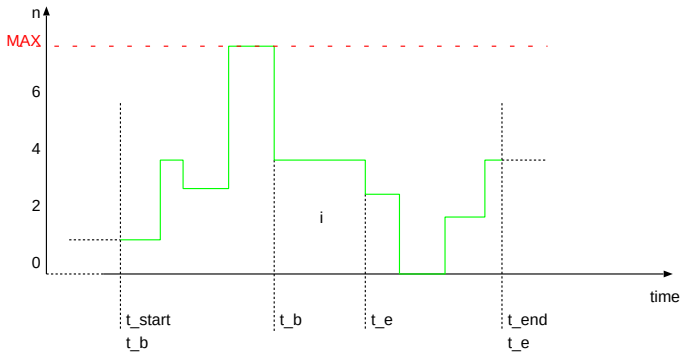
The fraction (or %) of time each *individual* entity is engaged



$$U = \frac{1}{t_{end} - t_{start}}\sum_{i=1}^{N}(t_e - t_b)_i$$

Average Use and Occupancy

for *groups* and *classes* of entities



Average Use and Occupancy (ctd.)

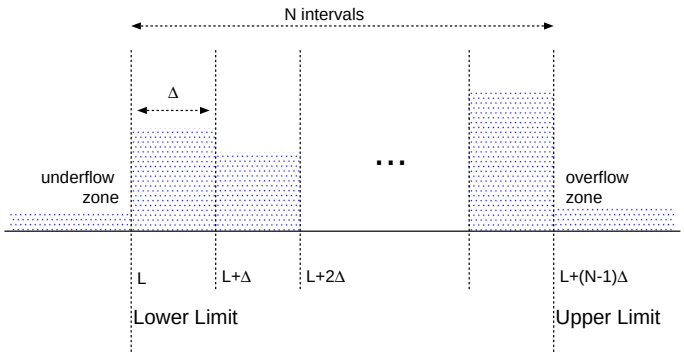
- Average use over time

$$A = \frac{1}{t_{end} - t_{start}} \sum_{i=1}^{N} n_i (t_e - t_b)_i$$

- Occupancy: average number in use with respect to MAX

$$O = \frac{1}{N \times MAX} \sum_{i=1}^{N} n_i (t_e - t_b)_i$$

Distributions and Transit Times



Number of intervals N, Uniform interval size Δ, Lower tabulation limit L.

Implementation: table of interval *counters*.

Global accumulation: number of entries, sum of entries, sum of squares.

Distributions and Transit Times (ctd.)

- Transit times: use clock as *time stamp*, enter in table at end of transit.
- Distribution of number of entitities: measure at uniform intervals of time.