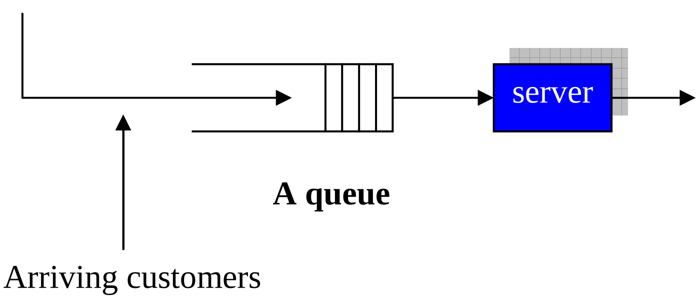
GPSS Models.

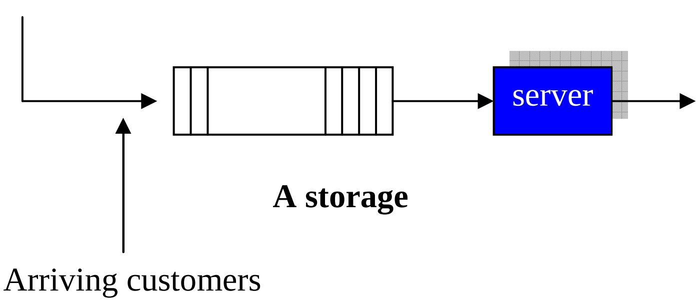
GPSS ≡ General Purpose Simulation System.
Developed by Jeff Gordon at IBM around 1961, 1962.

GPSS is not a programming system like Simscript; one doesn't "write" program here, but design a network of Blocks through which percolating simulation objects give rise a sense of process.

A discrete event simulator that basically sees system dynamics in terms of queues, storage, etc.



Another possiblity

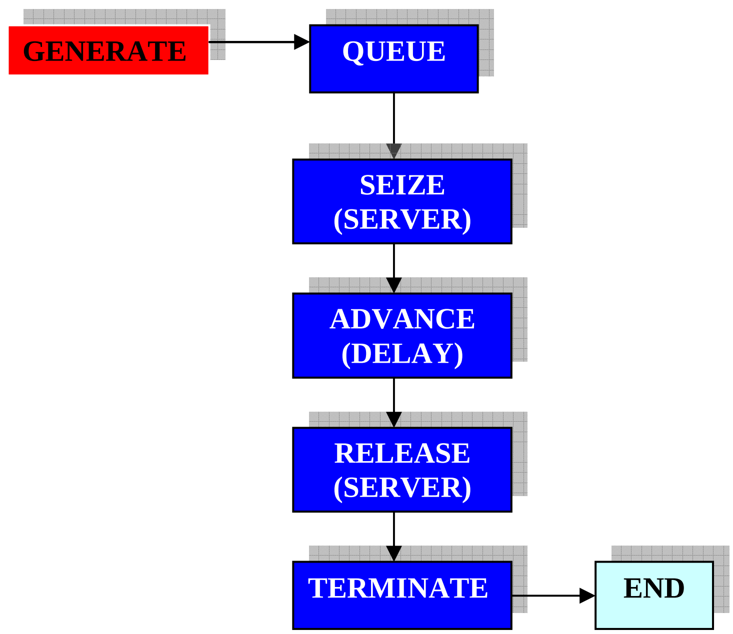


GPSS is a process-oriented simulation.

A process begins when we start generating transactions. As transactions percolate through system blocks, the process advances.

Process orientation is communicated in a FORTRAN type simulation platform.

We would use GPSS/PC package by Minuteman Software.



Basic structure:

● A transaction is a GPSS object with a number of attributes. A transaction is like a customer entering into the process for service. A single transaction may represent several individual entities.

● Each transaction has to be generated either one at a time or in batches. Once they appear into the system, they must be contained exactly in one action Block. However, a Block may contain many transactions.

Some typical Blocks.

■ A GENERATE Block generates a stream of transactions with a specific set of behavior. No transaction may again enter this block. Behavior could be deterministic, stochastic, functional, etc.

■ A Transaction leaving a GENERATE Block descends into the next available Block it finds. The entering Block shouldn't deny entries to transactions. Otherwise, system backups may result.

■ A QUEUE Block never refuses any transaction. If a transaction cannot enter into the next Block, it stays at the current Block. Therefore, a QUEUE simulates an infinitely long buffer.

■ A transaction attempts to SEIZE a facility (server, router, CPU) for service. If it succeeds, it would leave the current Block and start using the facility. If not, it stays where it is until the next time. As long as a facility is occupied, it cannot allow another transaction to SEIZE it.

■ An ADVANCE Block captures the transaction and imposes a delay on it wherever it is. The delay could be deterministic, probabilistic, etc.

■ A RELEASE Block forces a transaction to release its facility. For every successful SEIZE, there must be a RELEASE.

■ A TERMINATE Block kills the entering transaction here.

Example.

```
10    GENERATE     28, 6      ;customers for gas
15    QUEUE        PUMPQ      ;join here for service
17    SEIZE        PUMP1      ;try to get the pump if available
25    DEPART       PUMPQ      ;get out of PUMPQ
30    ADVANCE      15,8       ;spend sometime at the facility
35    RELEASE      PUMP1      ;release facility grabbed earlier
40    TERMINATE    1          ;kill this transaction
START    250
STOP
50    END
```

This is entered as a line by line interpreted program under GPSSPC. When simulation is completed, an alarm will sound.

To see the result, run GPSSREPT. exe

A typical block appears as

Line_number Label BLOCKTYPE A,B,C,… ;comment

e.g.

32 DURN ADVANCE FN$DELAY ; some comments here

A comment card begins with an asterisk after the line number.

Some more basic models.

The buffer is finite sized. Once the buffer is full, the incoming packets at a router are dropped. The size of the router buffer is initialized before transactions are generated.

```
10 BUFFER       STORAGE        15     ; storage size of buffer is 15
15 *********************************************************
16 *                                                       *
17 *                  ROUTER SIMULATION                    *
18 *                                                       *
19 *********************************************************
20          GENERATE      25,15,3
25          TRANSFER      BOTH,,DROP
30          ENTER         BUFFER,1        ;enter into buffer
35          SEIZE         ROUTER
40          LEAVE         BUFFER,1        ;give up buffer
45          ADVANCE       12,5
50          RELEASE       ROUTER
60 DROP     TERMINATE     1
START       80
STOP
70          END
```

Some explantions.

1. A transaction ENTERs a storage, and must LEAVE a storage.
2. TRANSFER works in BOTH mode as follows:

```
TRANSFER     BOTH, FIRST, SECOND
```

Try to get into the FIRST Block. If denied for some reason, go to the Block specified as SECOND. If SECOND is not ready to take, stay at the current Block.

IF FIRST is missing (a comma is there, instead), the transaction would first try to go the next block immediately following the TRANSFER.

An unconditional transfer (a goto statement) appears as

```
TRANSFER     ,THERE
```

As soon as a transaction hits this block it is sent to Block marked THERE.

Generating parallel streams of independent processes.

```
TERM1    GENERATE    18,6,,250,1
         TRANSFER    ,THERE
TERM2    GENERATE    21,5,,,2
         TRANSFER    ,THERE
TERM3    GENERATE    15,5,,200,3
THERE    QUEUE       MEMQ
         …
         …
```

3. GENERATE creates transactions for future entry into the simulation.

```
         GENERATE    A,B,C,D,E
```

A: Mean interarrival time

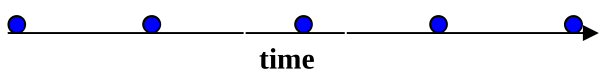
B: Interarrival time modifier. Optional
C: Start delay time. Time increment for the 1st transaction. Optional.
D: Creation limit. Optional
E: Priority. Zero is the default. Optional

All transactions are generated with interarrival time uniformly distributed between **A-B** and **A+B**, **A** must be greater than **B**.

Examples:

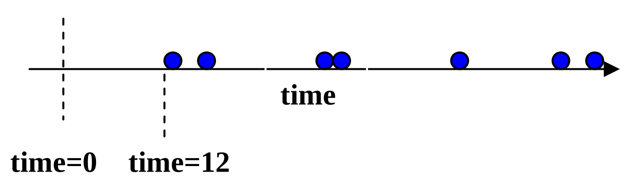
GENERATE 60

Generate a priority 0 transaction every 60 units of time.



GENERATE 40,8,12,250,5

First transaction will be ready at time 12. After that the interarrival time between transactions will be a uniformly distributed between (40-8) and (40+8) units of time. This block will create at most 250 transactions, all with priority 5 value.



GENERATE ,,,50

A bunch of 50 transactions will be generated by this block and all will arrive together at time = 1 (the default simulation time). Notice its first three operands are missing.

Example.

A service center that opens up at time 1 and stays open till time 480 when an alarm clock signaling the end of the day is received.

Service center works from 9:00 AM to 5:00 PM = 8 hrs.
= 480 mins.

```
10    GENERATE     28, 6     ;customers for gas
15    QUEUE        PUMPQ     ;join here for service
17    SEIZE        PUMP1     ;try to get the pump if available
25    DEPART       PUMPQ     ;get out of PUMPQ
30    ADVANCE      15,8      ;spend sometime at the facility
35    RELEASE      PUMP1     ;release facility grabbed earlier
40    TERMINATE              ;kill this transaction
45
50    GENERATE     480       ;send a bell at 480 mins
60    TERMINATE    1
62
START    1
50    END
```

We would have two processes: One comprises transactions coming in and getting service before they depart, the second a clock process which sends its alarm at the conclusion of the day.

Note that the counter begins with a value 1. And it wouldn't be reduced until the alarm is received. Meanwhile, the regular transactions would receive service and exit via the first TERMINATE block.

Some points to remember.

■ No transaction may enter a **GENERATE** Block.
■ A **QUEUE** block cannot refuse an entering transaction since it is infinitely long.
■ **START n** sets transaction accounting counter to **n.** When this counter becomes 0, simulation stops.
■ Every time a transaction hits a **TERMINATE k** block transaction accounting counter would be changed from counter(now) $\rightarrow$ counter(now) – k.
■ If a transaction enters a **QUEUE**, it should be potentially capable to **DEPART** it. **QUEUE** $\leftrightarrow$ **DEPART** goes together.
■ If a transaction enters a facility via **SEIZE**, it should get the opportunity to **RELEASE** it.
RESET card.
To remove the initial bias, we may use RESET block to wipe out all statistics except the entry-counts on the blocks. The relative clock will be set to zero, but the absolute clock will continue as before.

```
10     GENERATE      28, 6       ;customers for gas
15     QUEUE         PUMPQ       ;join here for service
17     SEIZE         PUMP1       ;try to get the pump if available
25     DEPART        PUMPQ       ;get out of PUMPQ
30     ADVANCE       15,8        ;spend sometime at the facility
35     RELEASE       PUMP1       ;release facility grabbed earlier
40     TERMINATE     1           ;kill this transaction
START    250
RESET
START    100
STOP
50  END
```

If we want to suppress output from the first set of run, we need to change it in the following way:

```
START    250, NP       ; NP stands for no printing
RESET
START    100
STOP
50  END
```

To repeat simulation with a different set of customers we use a **CLEAR** card followed be a different **START** card.

```
START    250
CLEAR
START    100
STOP
50  END
```

Non-empty queue at start of the simulation.

When a server starts, we want it to find customers already in its queue. This could be done in the following way:

```
10          GENERATE     ,,,5  ; 5 customers arrive at time 1
20          TRANSFER     , HERE
25          GENERATE     20, 8
27 HERE     QUEUE        LINE
30          SEIZE        PUMP
35          DEPART       LINE
39          ADVANCE      16,4
43          RELEASE      PUMP
47          TERMINATE    1
START 40
STOP
END
```

Mean interarrival time depends on the time of day

```
10  AGAIN          GENERATE     18,6
15                 QUEUE        LINE
20                 SEIZE        PUMP
25                 DEPART       LINE
30                 ADVANCE      16,4
35                 RELEASE      PUMP
40                 TERMINATE
45                 GENERATE     60 ; Timer arrives every hour
50                 TERMINATE    1
START       3    ; Simulate three hours
55  AGAIN          GENERATE     9,6
START       1    ; Simulate one hour
60  AGAIN          GENERATE     12,6
START       6    ; Simulate six hours
STOP
END
```