

Operations Research (Master's Degree Course)

8. Discrete Simulation

Silvano Martello

DEI "Guglielmo Marconi", Università di Bologna, Italy



This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License.

Based on a work at <http://www.editrice-esculapio.com>

Generalities

- Discrete Simulation is used when the system
 - is complex;
 - evolves over time;
 - involves queueing processes;
 - includes a large number of interacting entities.
- **Simulation** is a powerful tool, in many cases the only one available;
- it creates a numerical prototype of the model, used to predict its performance;
- some similarities with video games;
- it is useful both to design a new system, and to improve an existing one.
- We will consider **Discrete event simulation**, in which the system is represented as a sequence of (random) events that modify the status of the system.
- Discrete simulation has anticipated important computer science developments: the **Simula 67** language is regarded as the first **object oriented** language.
- A totally different simulation methodology is **Continuous simulation**, in which the system is modeled by a set of differential equations.

Generalities (cont'd)

- A discrete event simulation model
 - represents the system as sets of independent elements, called **entities**,
 - which have specific characteristics, called **attributes**,
 - and are interconnected by relationships, like belonging to **sets**

(STATIC structure)

 - The model describes the evolution of the system through actions, called **events**

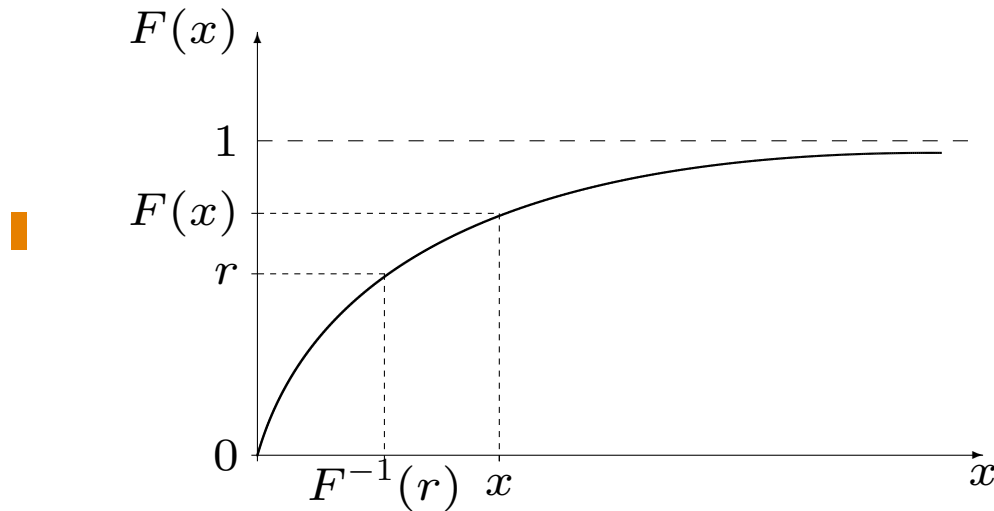
(DYNAMIC structure)
- This results in a **computer program**, written in a **simulation language**, whose runs
 - use **random numbers** to generate simulated events over time;
 - give on output **statistical information** on the system performance.
 - Through **iterated experiments (runs)** with different system configurations the best one (or a good one) is obtained.
 - Computers cannot generate truly random numbers, but any language has software functions to generate sequences of real numbers (usually with values **uniformly random in $[0, 1]$**), starting from a **seed** and recursively applying a **generating algorithm**.
- Such numbers are referred to as **pseudo random**: for all practical applications they can be used as genuine random numbers.

Generating pseudo random numbers from a probability distribution

- X = random variable;
- t = possible value (outcome) of X ;
- $f(x)$ = **probability density function** = $\lim_{\Delta x \rightarrow 0} \frac{\mathbb{P}(x \leq t \leq x + \Delta x)}{\Delta x}$;
- $f(x) \geq 0$;
- $f(x)dx = \mathbb{P}(x \leq t \leq x + dx)$;
- $\int_a^b f(x)dx = \mathbb{P}(a \leq t \leq b)$;
- $\int_{-\infty}^{+\infty} f(x)dx = 1$ (normalization condition).
- E = **expected value** = $\int_{-\infty}^{+\infty} f(x)x dx$;
- V = **variance** = $\int_{-\infty}^{+\infty} f(x)(x - E)^2 dx$.
- $F(x)$ = **cumulative distribution function** = $\int_{-\infty}^x f(\xi)d\xi = \mathbb{P}(t \leq x)$:
 - $0 \leq F(x) \leq 1$;
 - $F(x)$ monotone non-decreasing.

Generating pseudo random numbers from a probability distribution(cont'd)

Inverse transformation method: Given a probability density function $f(x)$, and values r uniformly random in $[0, 1]$, the values $F^{-1}(r)$ have the probability density function $f(x)$.



Proof For a random variable Y uniformly distributed in $[0,1]$: we have

$$f(y) = 1 \Rightarrow \mathbb{P}(r \leq q) = \int_0^q f(\xi) d\xi = q \quad (\forall q, 0 \leq q \leq 1).$$

Hence $\mathbb{P}(r \leq F(x)) = F(x)$.

From the transformation: $\mathbb{P}(r \leq F(x)) = \mathbb{P}(F^{-1}(r) \leq x)$.

$$\Rightarrow F(x) = \mathbb{P}(F^{-1}(r) \leq x)$$

i.e., $F^{-1}(r)$ has cumulative distribution function $F(x)$

i.e., $F^{-1}(r)$ has probability density function $f(x)$. \square

Generating pseudo random numbers from a probability distribution(cont'd)

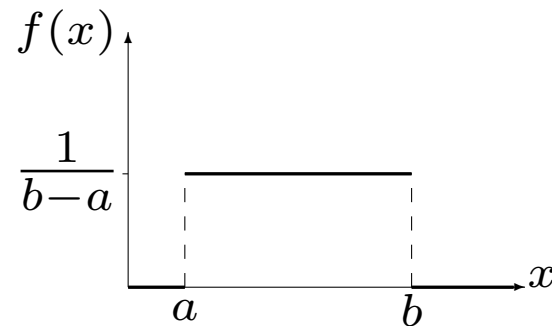
- **Example:** $f(x) = 8x$ for $0 \leq x \leq \frac{1}{2}$:

$$F(x) = \int_{-\infty}^x f(\xi) d\xi = \int_0^x 8\xi d\xi = 8 \left[\frac{\xi^2}{2} \right]_0^x = 4x^2;$$

$$r = 4x^2 \Rightarrow x = \frac{\sqrt{r}}{2}.$$

- **Uniform distribution in $[a, b]$:**

$$f(x) = \frac{1}{b-a} \quad (a \leq x \leq b);$$



$$F(x) = \int_a^x \frac{1}{b-a} d\xi = \frac{1}{b-a} [\xi]_a^x = \frac{x-a}{b-a};$$

$$r = \frac{x-a}{b-a} \Rightarrow x = a + r(b-a);$$

$$E = \int_a^b \frac{x}{b-a} dx = \frac{1}{b-a} \left[\frac{x^2}{2} \right]_a^b = \frac{1}{b-a} \cdot \frac{b^2 - a^2}{2} = \frac{a+b}{2}.$$

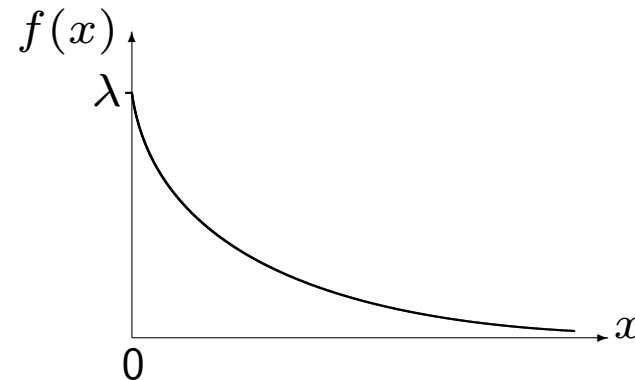
Generating pseudo random numbers from a probability distribution(cont'd)

- Exponential distribution:**

$$f(x) = \lambda e^{-\lambda x} \quad (\lambda > 0, x \geq 0);$$

$$E = \frac{1}{\lambda};$$

$$V = \frac{1}{\lambda^2};$$



$$F(x) = \int_0^x \lambda e^{-\lambda \xi} d\xi = \lambda \left[-\frac{e^{-\lambda \xi}}{\lambda} \right]_0^x = 1 - e^{-\lambda x};$$

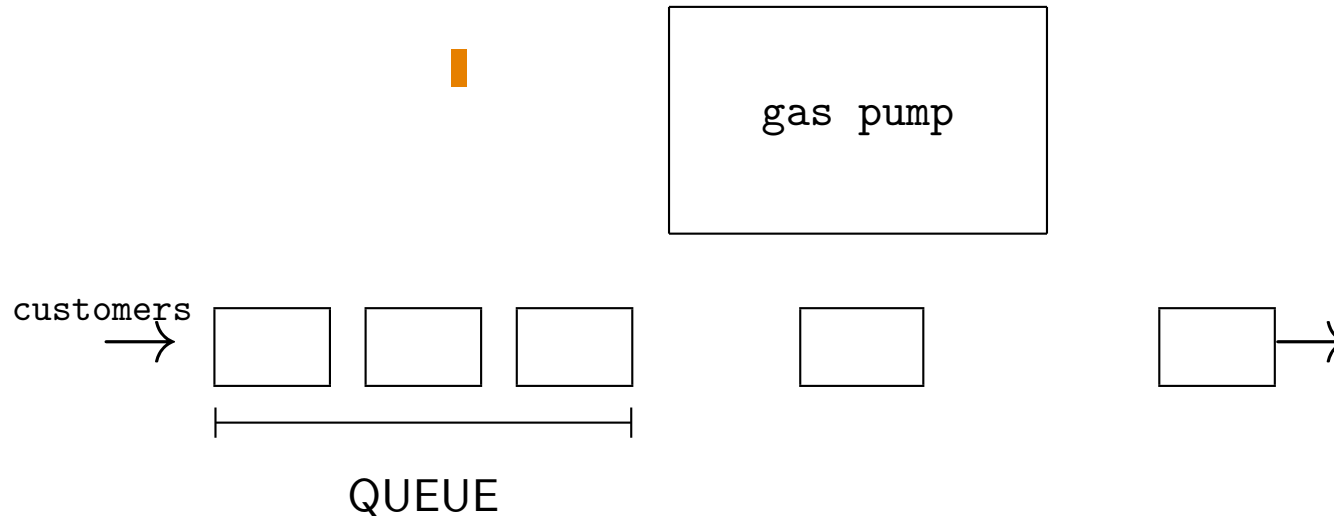
$$r = 1 - e^{-\lambda x} \Rightarrow 1 - r = r' = e^{-\lambda x} \Rightarrow \ln r' = -\lambda x$$

$$r' \text{ has a uniform distribution in } [0,1] \Rightarrow x = -\frac{1}{\lambda} \ln r'.$$

- In many cases, **Poisson processes** are good models for random arrivals in which λ = expected number of arrivals during a unit time interval.
- In a Poisson process of expected value λ the time between each pair of consecutive events has an exponential distribution of expected value $\frac{1}{\lambda}$.

Static and Dynamic description of a simulation model

We will use a first **Example:** gas station (with a single fuel dispenser)



- customers arrival: Poisson distribution of expected value λ ;
- refueling time: uniform distribution in $[T1, T2]$;
- if $NMAX$ customers are already queueing, the new customer gives up;
- end simulation after having **completely** simulated $NTOT$ customers;
- determine the **average time spent in queue**.

Static description

- System status \leftrightarrow data structures.
- Dynamic processes (events) modify the data \Rightarrow they modify the system status.
- **Main objects** in a simulation model:

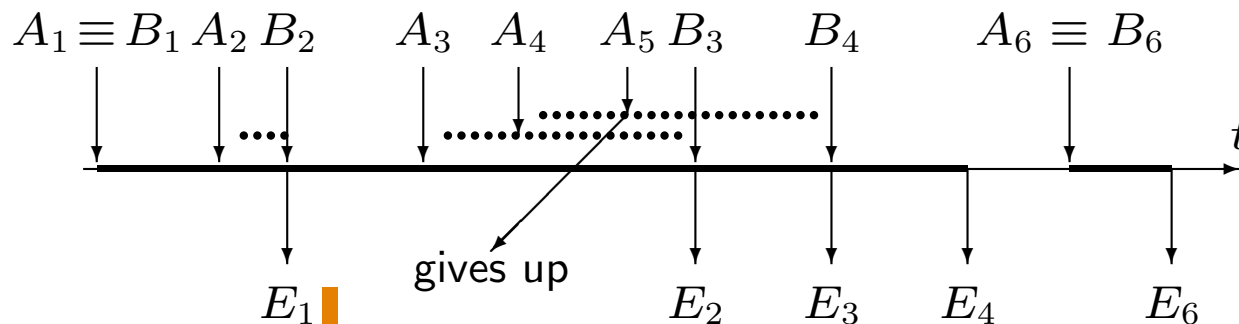
Terminology	Example	SIMSCRIPT
Entity	car	CAR
Attribute	entering time in queue TIC(CAR)	TIC(CAR)
Set	queue insert the car in the queue extract the first car from the queue	QUEUE FILE CAR IN QUEUE REMOVE FIRST CAR FROM QUEUE

- **Creation and destruction:**

Terminology	Example	SIMSCRIPT
Creation (entering the system)	create the car	CREATE CAR
Destruction (leaving the system)	destroy the car	DESTROY CAR

Dynamic description

- **Simulated clock (time):**
 - old languages: counter;
 - modern languages: only those time instants in which the system status varies are considered;
- **Example:** gas station with $NMAX = 2$ (*A*rrival, *B*egin service, *E*nd service):



- **Event** = subprogram containing the instructions to execute when the event occurs.
- Each event establishes which future events must occur, and “schedules” them (**Event-scheduling approach**):

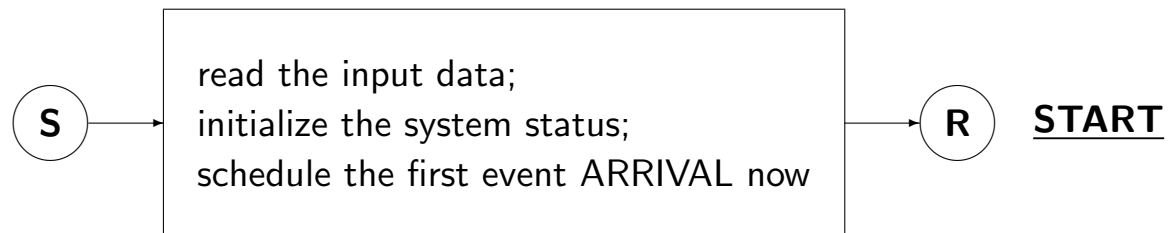
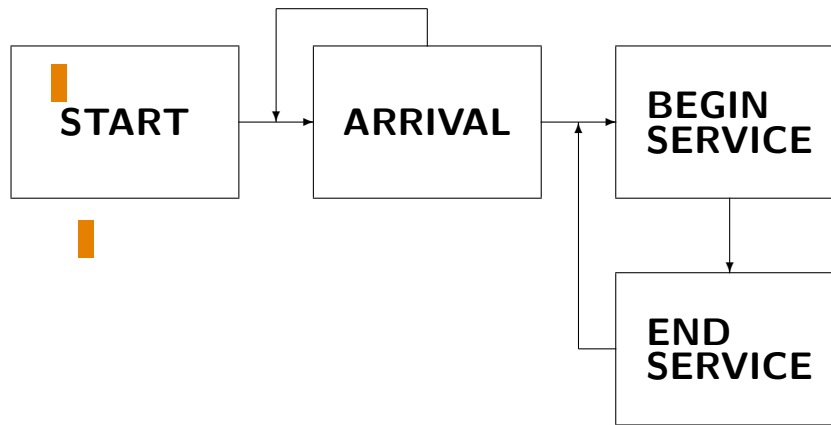
Example

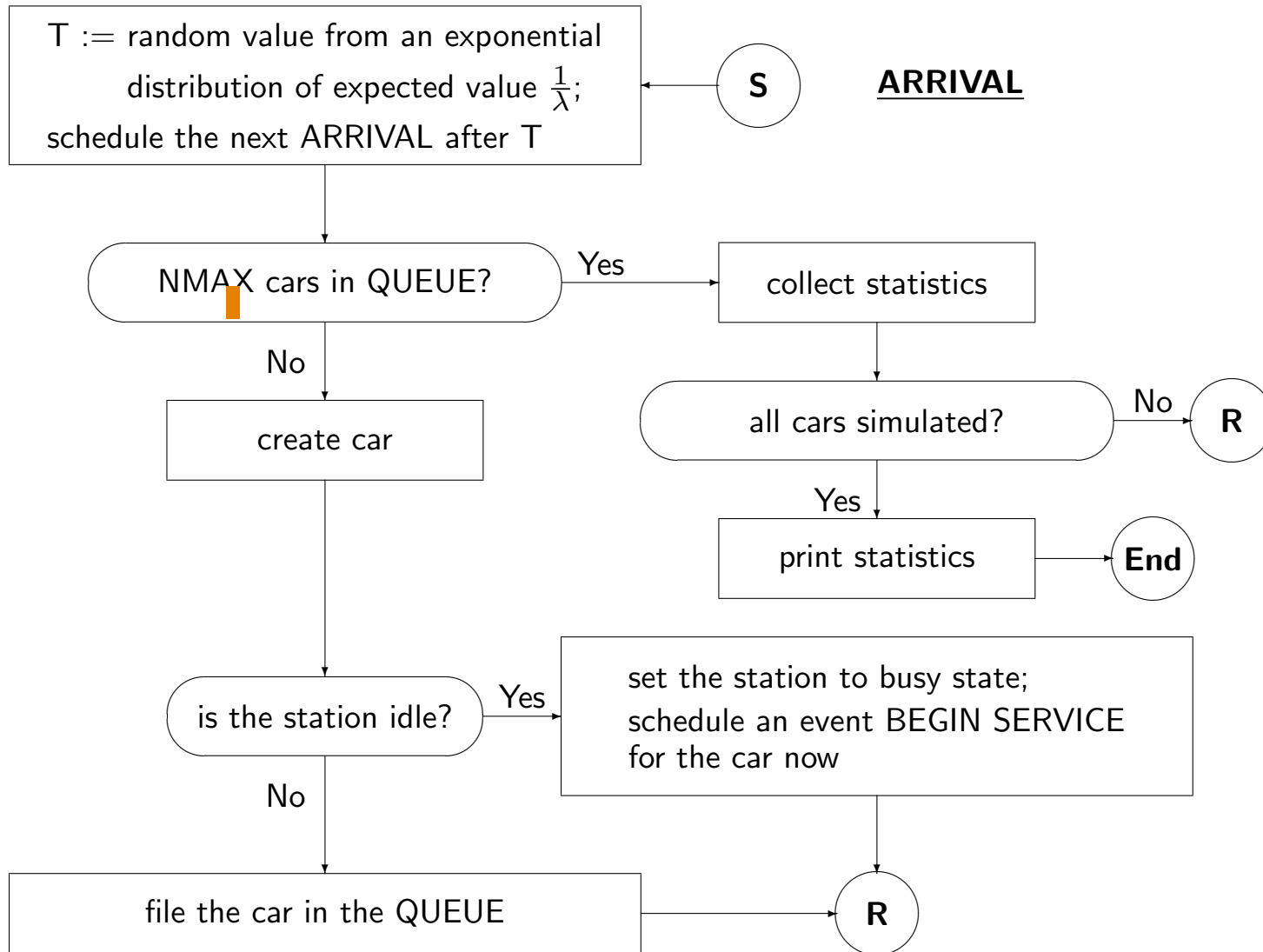
SIMSCRIPT

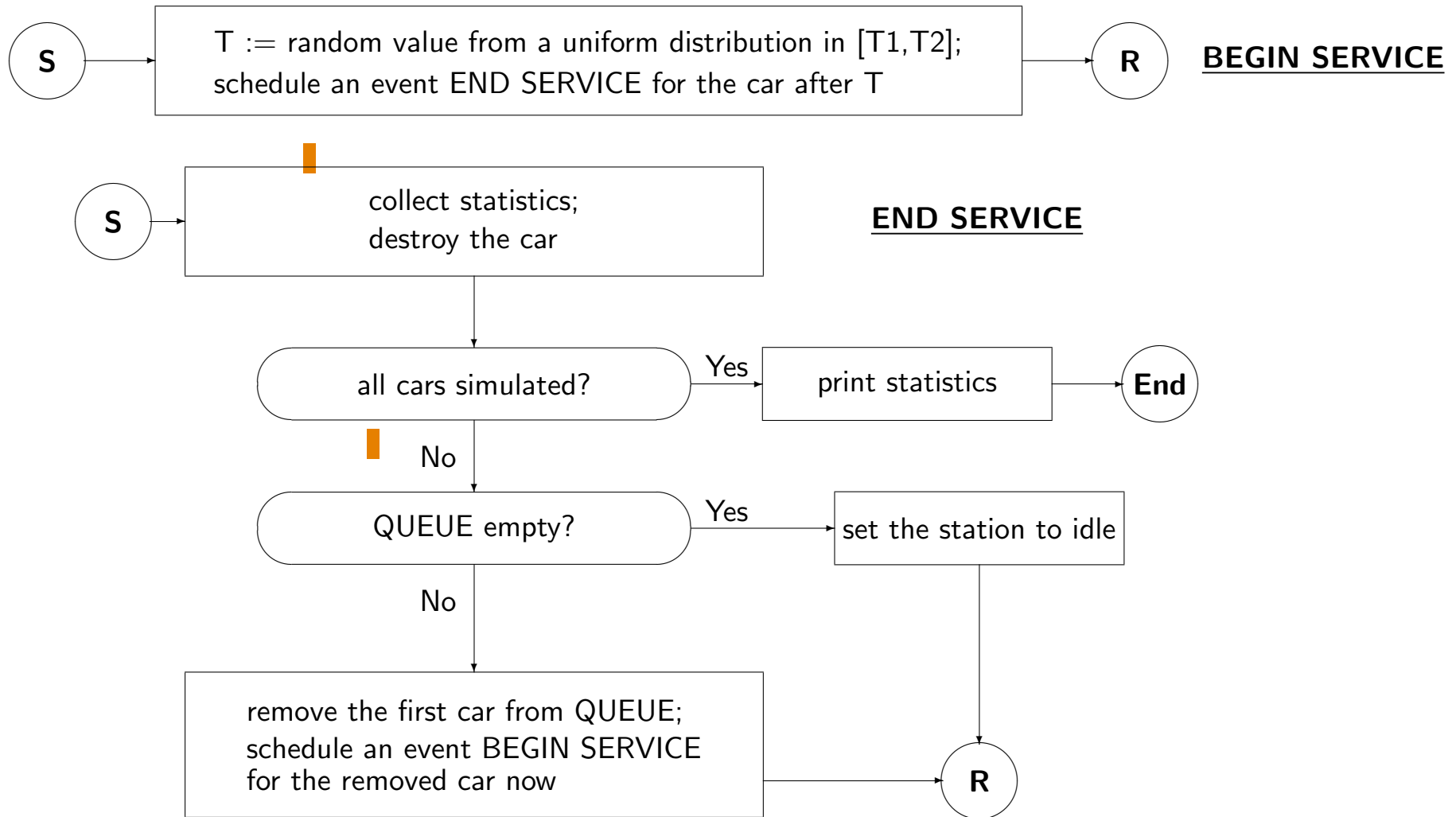
schedule an event END SERVICE after T (time units)

SCHEDULE AN END SERVICE AT TIME.V + T

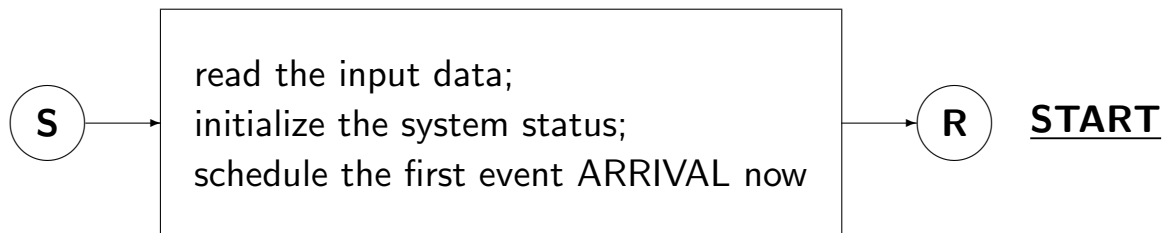
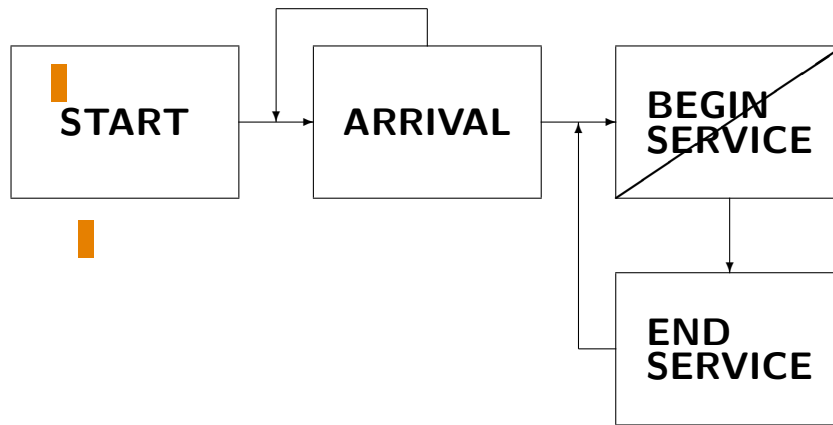
When the execution of an event terminates, the system determines the next scheduled event, and updates the simulated clock.

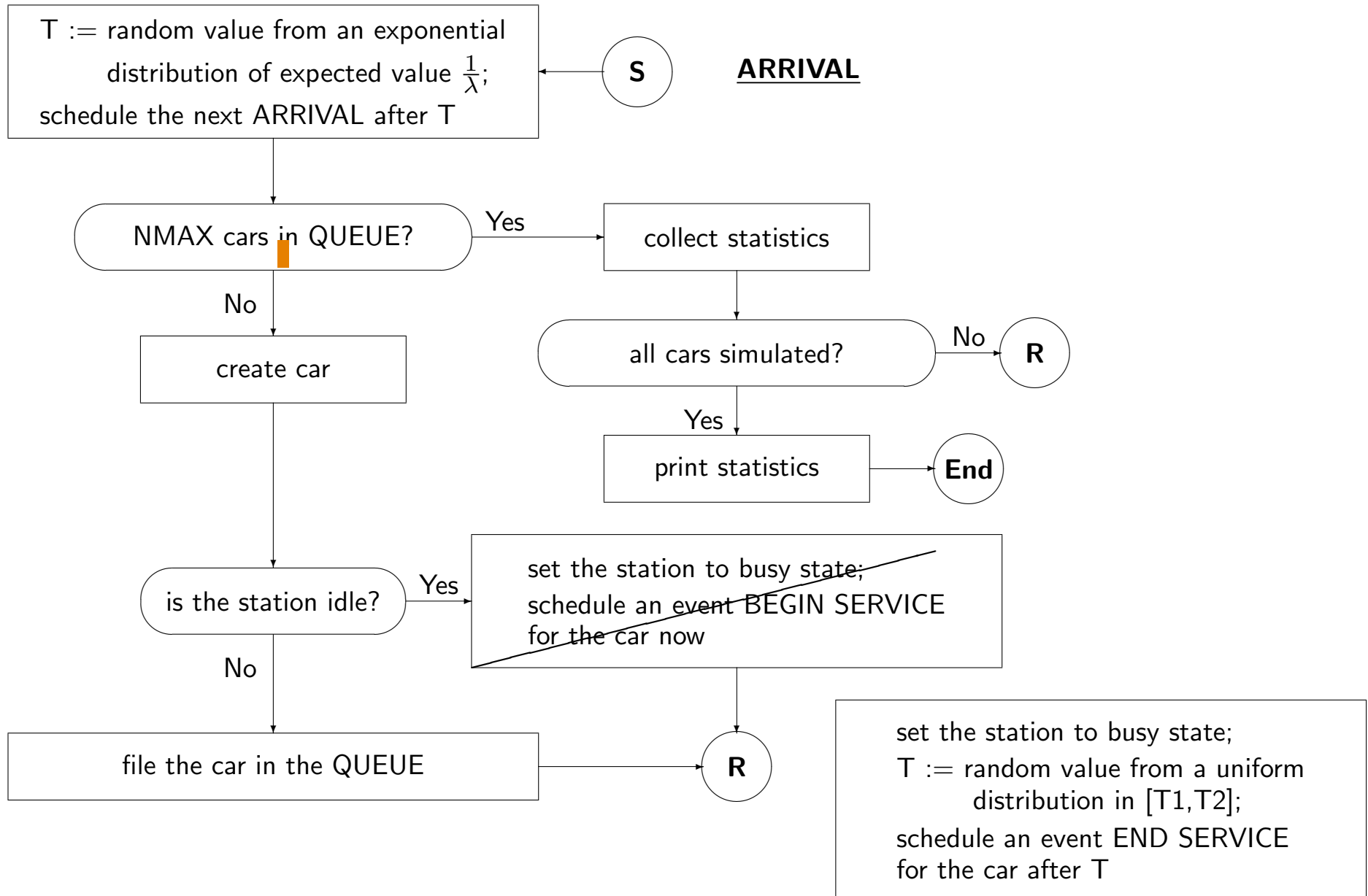


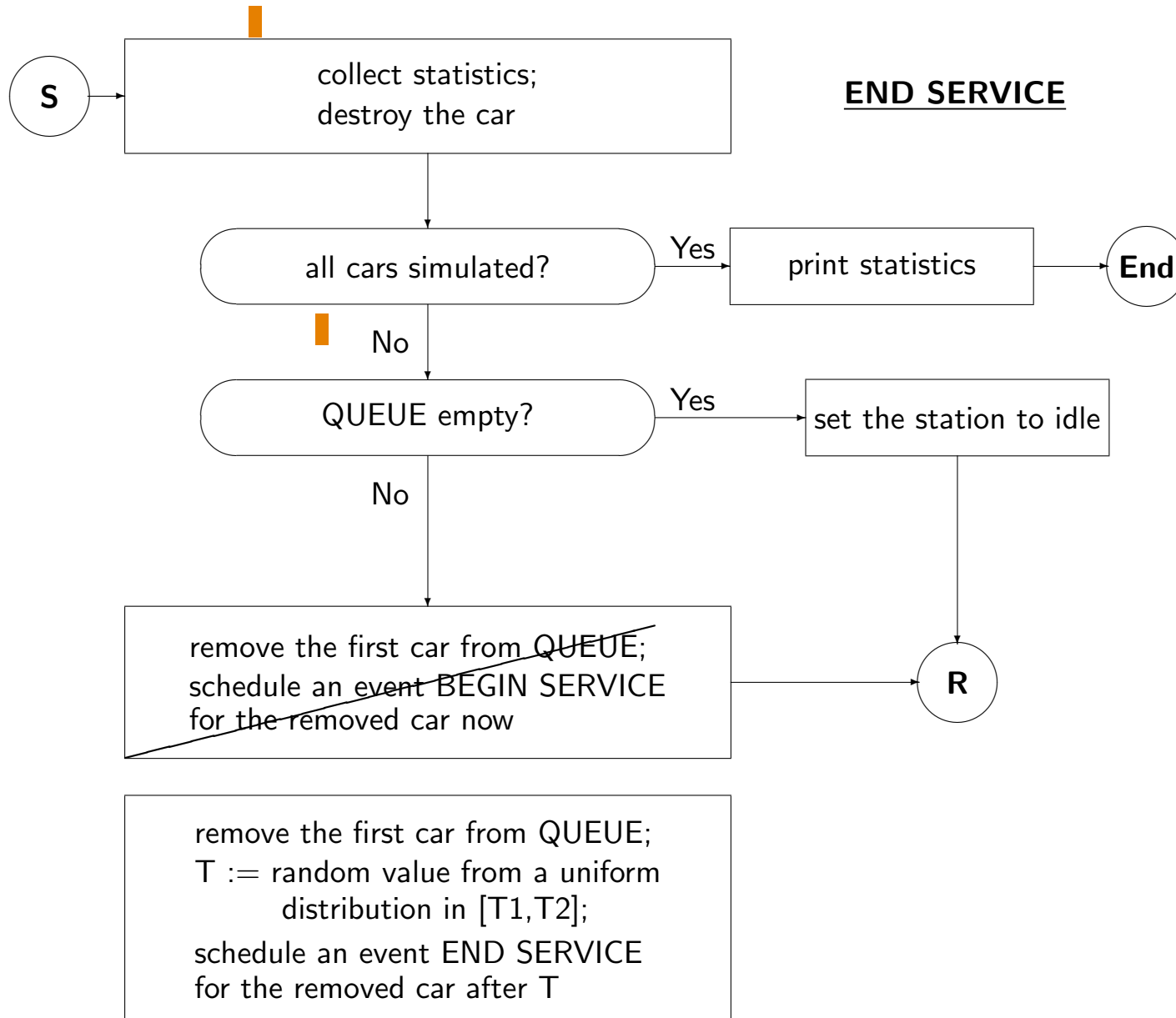
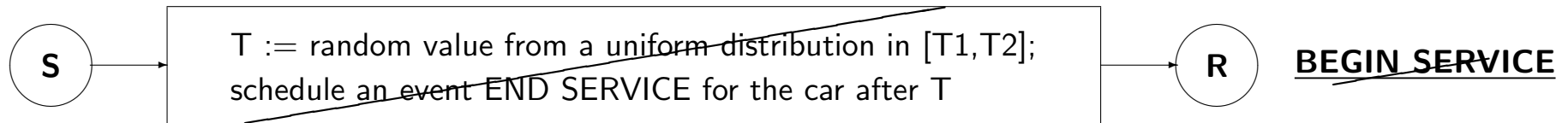




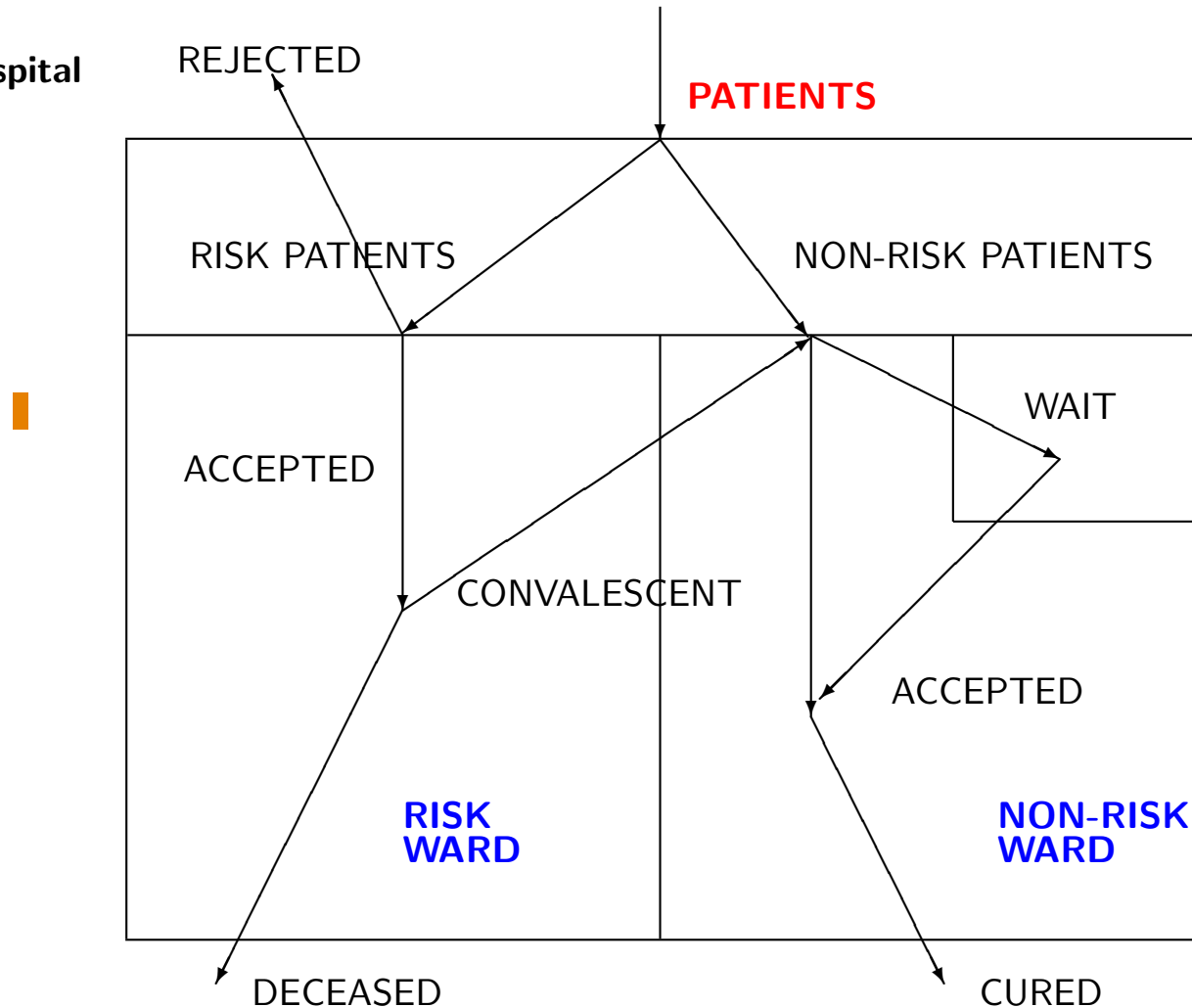
- An event **BEGIN SERVICE** always coincides with
 - an event **ARRIVAL**, **or** an event **END SERVICE**.
- Scheduling events requires computationally heavy inner procedures
- \Rightarrow better to eliminate useless events (further improvement: simpler and more readable model).







Example: small hospital



- Numbers of beds: **Risk** ward: NLG ; **Non-risk** ward: NLN ;
- Patients arrival: Poisson distribution with expected value λ : Risk with probability PG , non-risk with $1 - PG$;
- Risk patients rejected if there is no bed available in the risk ward;
- Stay durations: uniform distribution in $[DMIG, DMAG]$ (Risk ward), in $[DMIN, DMAN]$ (Non-risk ward);
- Outcome of Risk patients: successful with probability PS , unsuccessful with probability $1 - PS$.

$NLG = NLN = 1$,
system initially empty

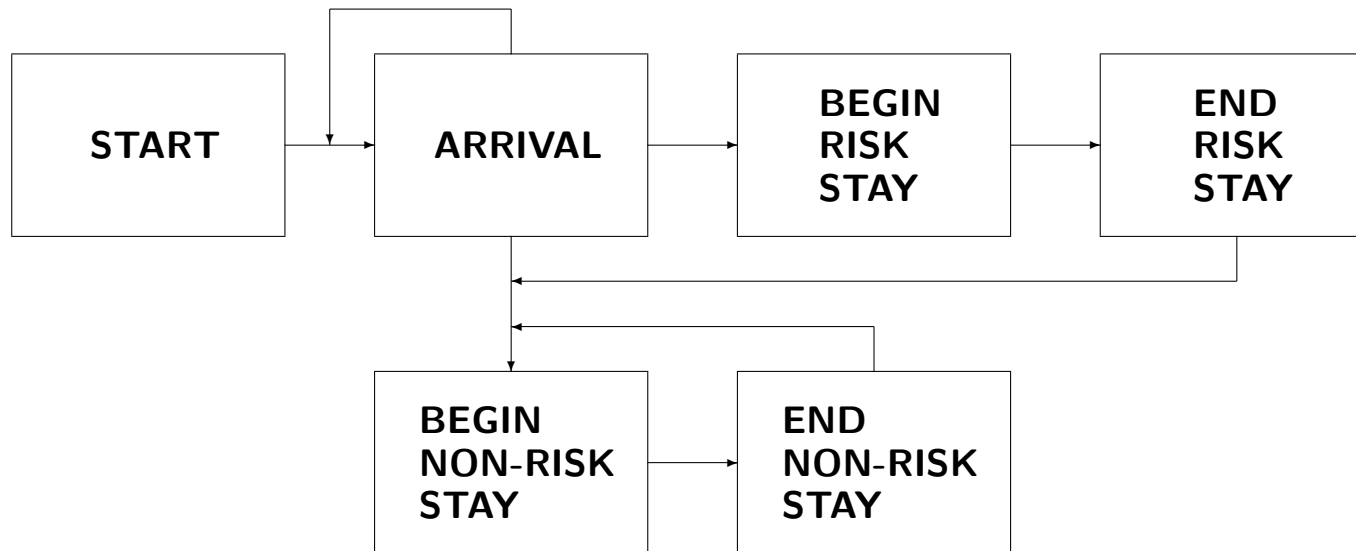
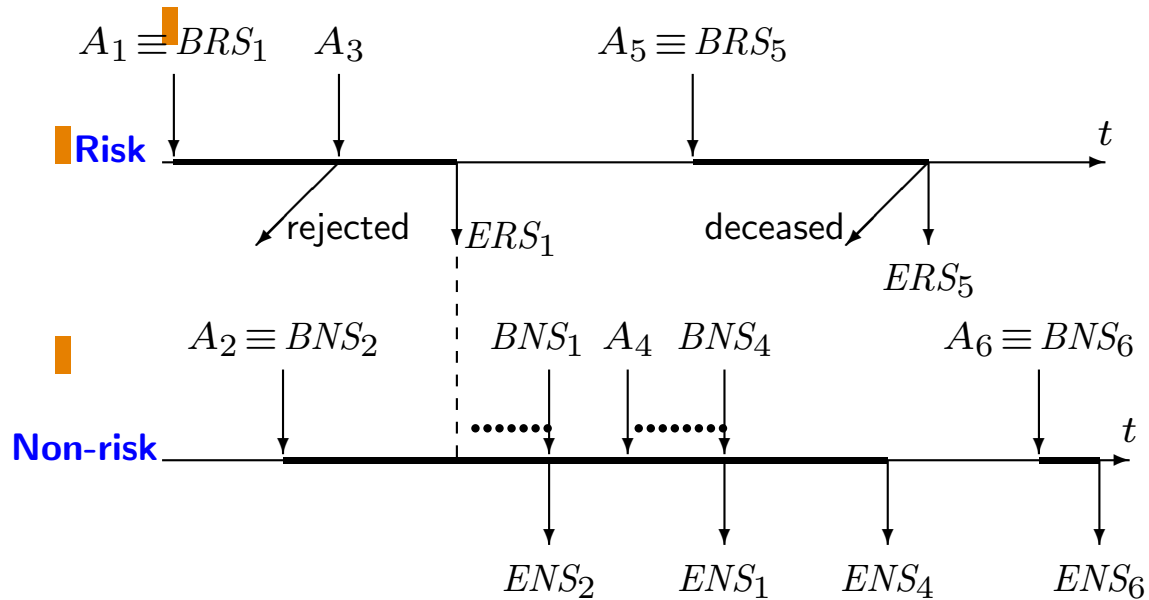
Arrival

Begin Risk Stay

Begin Non-risk Stay

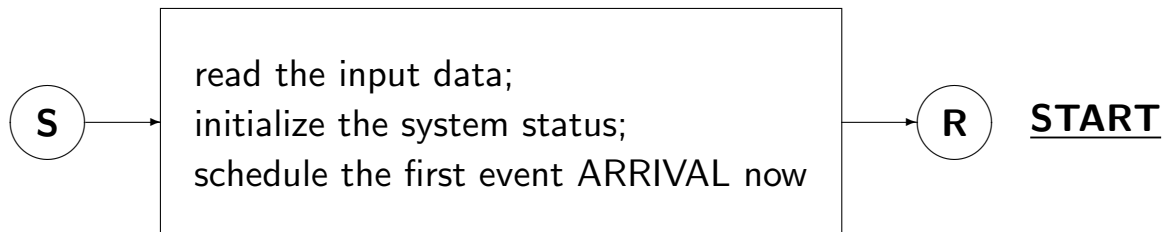
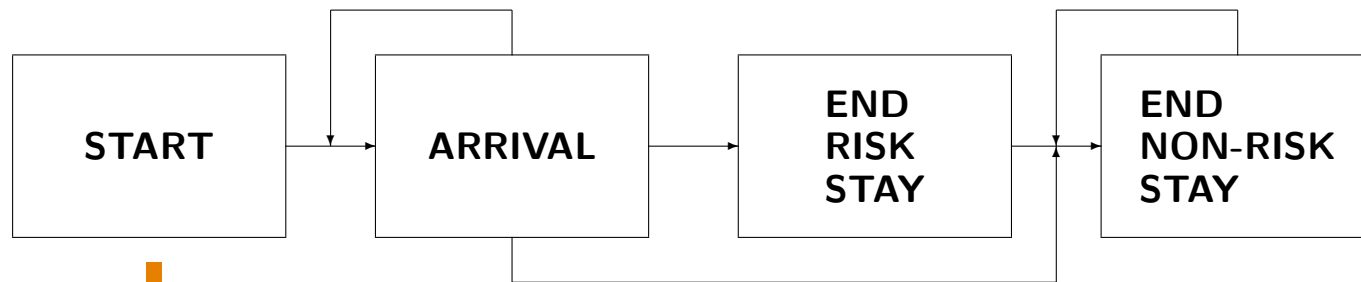
End Risk Stay

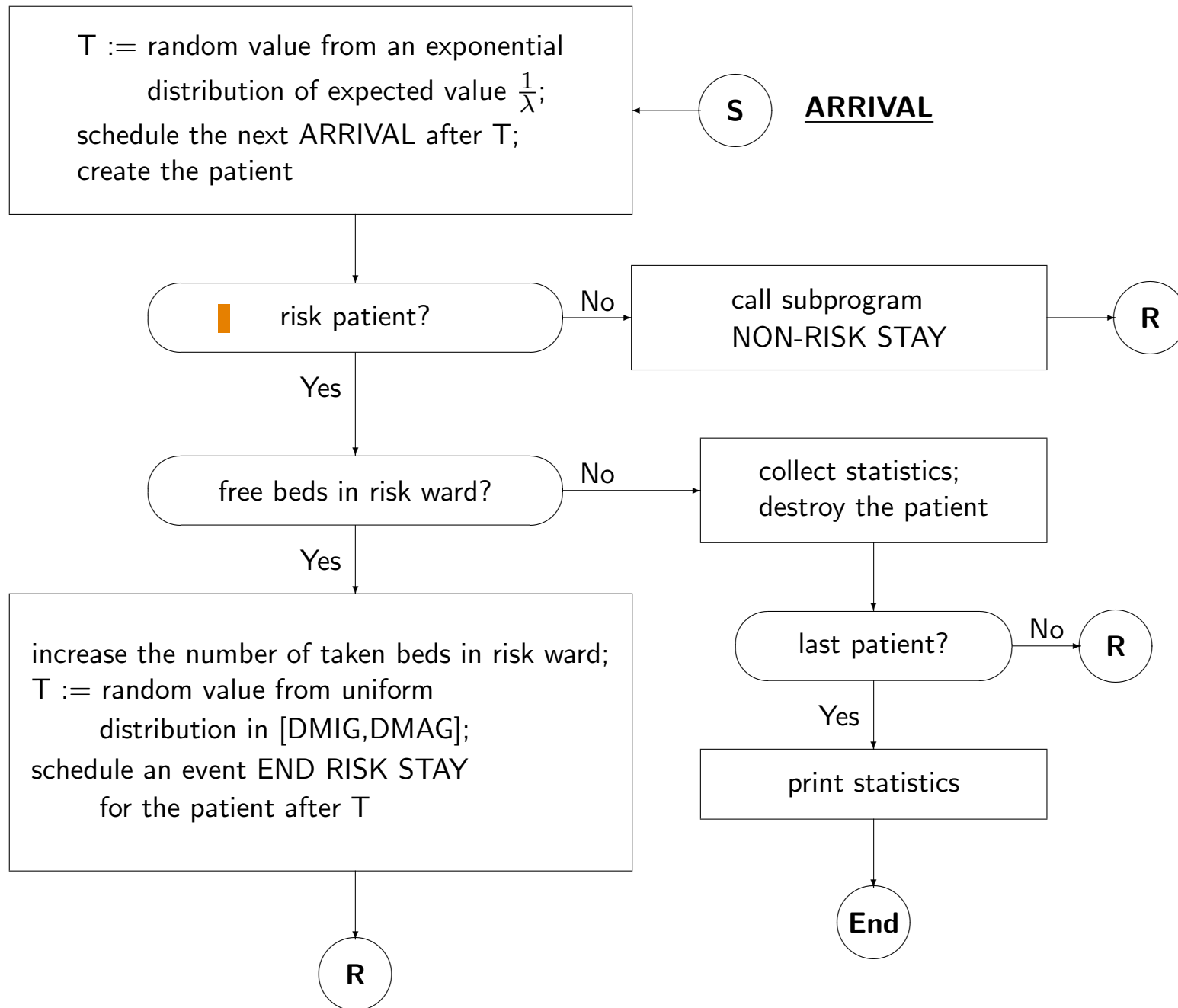
End Non-risk Stay



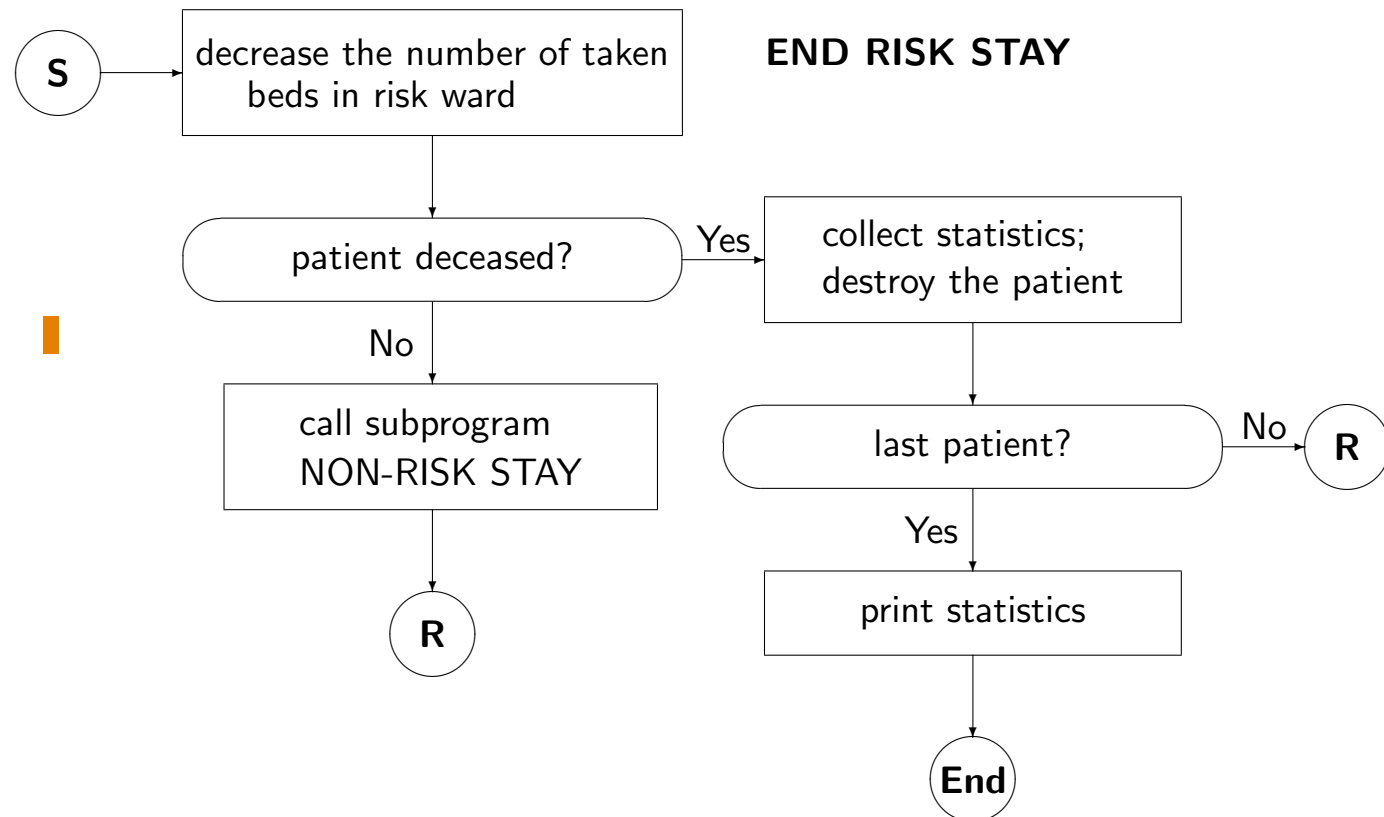
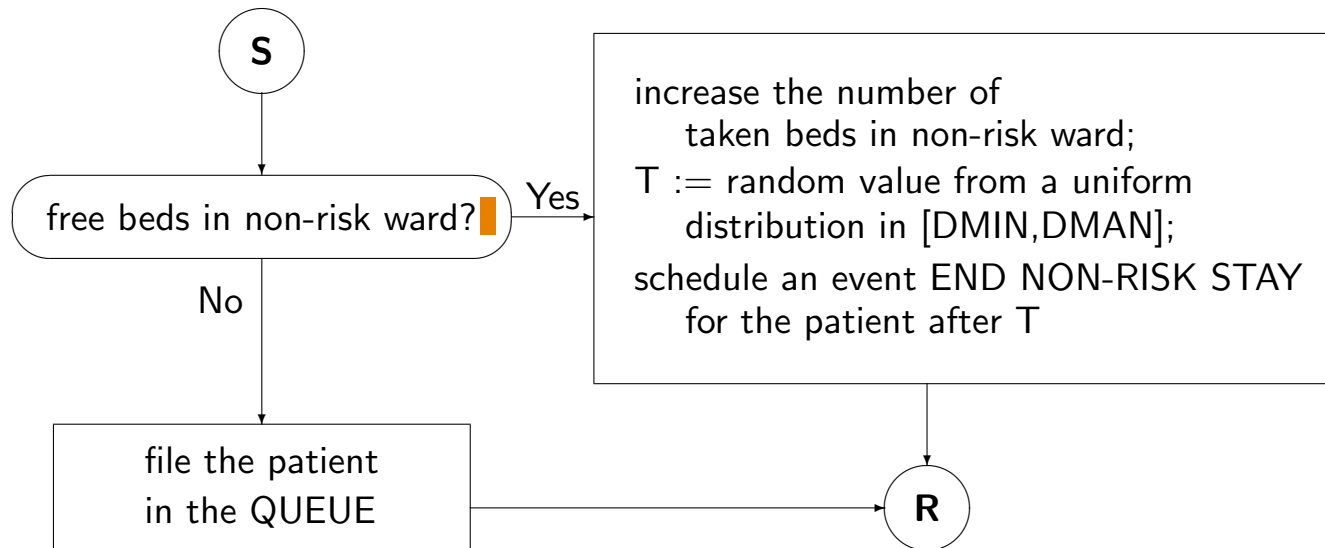
BRS always coincides with A ;

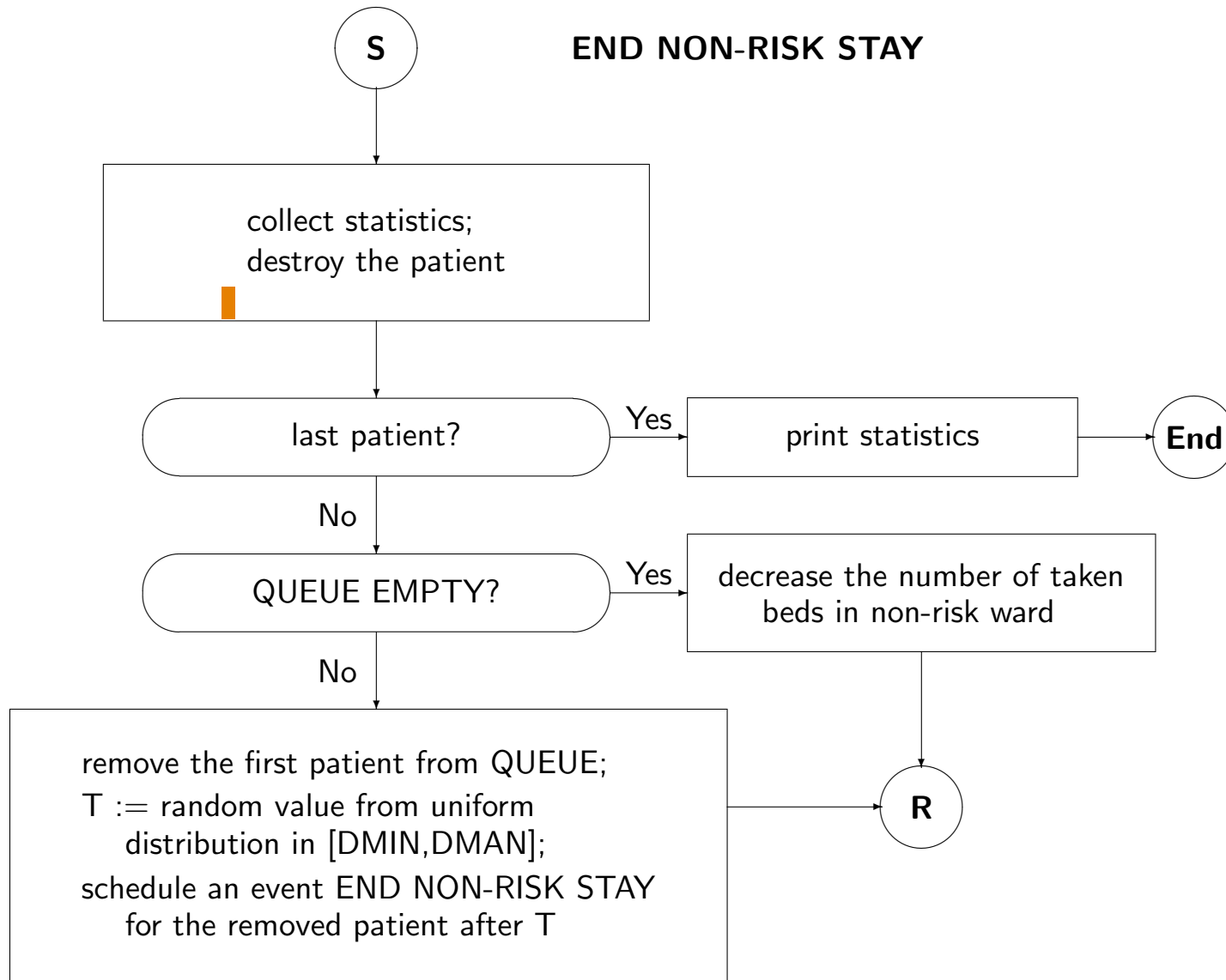
BNS always coincides with A , or with ERS , or with ENS .





Subprogram NON-RISK STAY





Simulation components

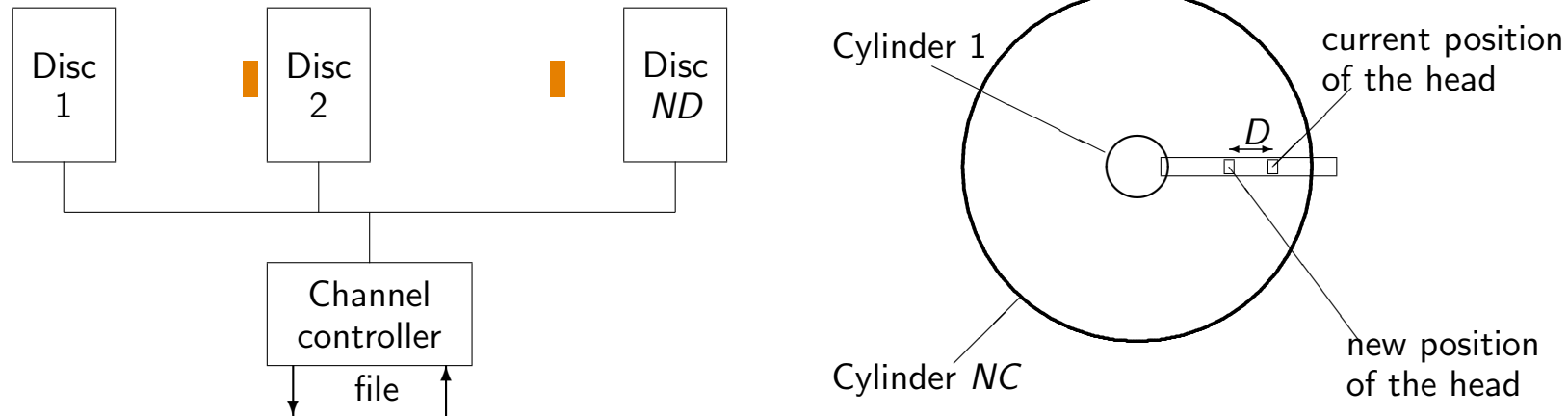
- **Entities:**

- *temporary entities* are created and destroyed (CAR, PATIENT);
 - * *temporary attributes* (kind of illness: risk, non-risk);
- *permanent entities*: always present during simulation
 - * fundamental permanent entity: the **System**;
 - *permanent attributes* (status (idle, busy), T1, T2, NMAX, ...; NLG, NLN, ...)
 - * other permanent entities are represented through **indices** (next example)

- **Events:**

- *endogenous events*: scheduled by other events (ARRIVAL, END SERVICE, END NON-RISK STAY, ...);
- *exogenous events*: scheduled from “outside” (START).

Example: simplified version of a multi-disc unit:



- Arrival of files (indifferently to read or write): Poisson distribution of expected value λ :
length: value L from uniform distribution in $[MIN, MAX]$;
requested disc / cylinder: uniform probability in $[1, ND]$ / in $[1, NC]$.
- If the disc is idle, the positioning of the head starts:
positioning time: function $f_1(D)$, with D = distance between current and new position.
- After positioning, if the channel is idle, waiting time for disc rotation (uniform distribution in $[0, TR]$), then transmission (time $f_2(L)$).
- **Queue at channel controller** (precedence to disks with lower number);
one queue per disk (precedence to files with lower length L).
- Terminate the simulation after TS time units.
 - Temporary entities **file**: Attributes: disc, cylinder, length.
 - Permanent entity **System**: Attributes: λ , ND , NC , TR , f_1 , f_2 , channel status.
 - Permanent entities **disks** (integers 1, 2, . . . , ND): Attributes: disc status, current head position.

$ND = 2$,
system initially empty

Arrival

Begin *B*usy *C*hannel

End *B*usy *C*hannel

Begin *P*ositioning

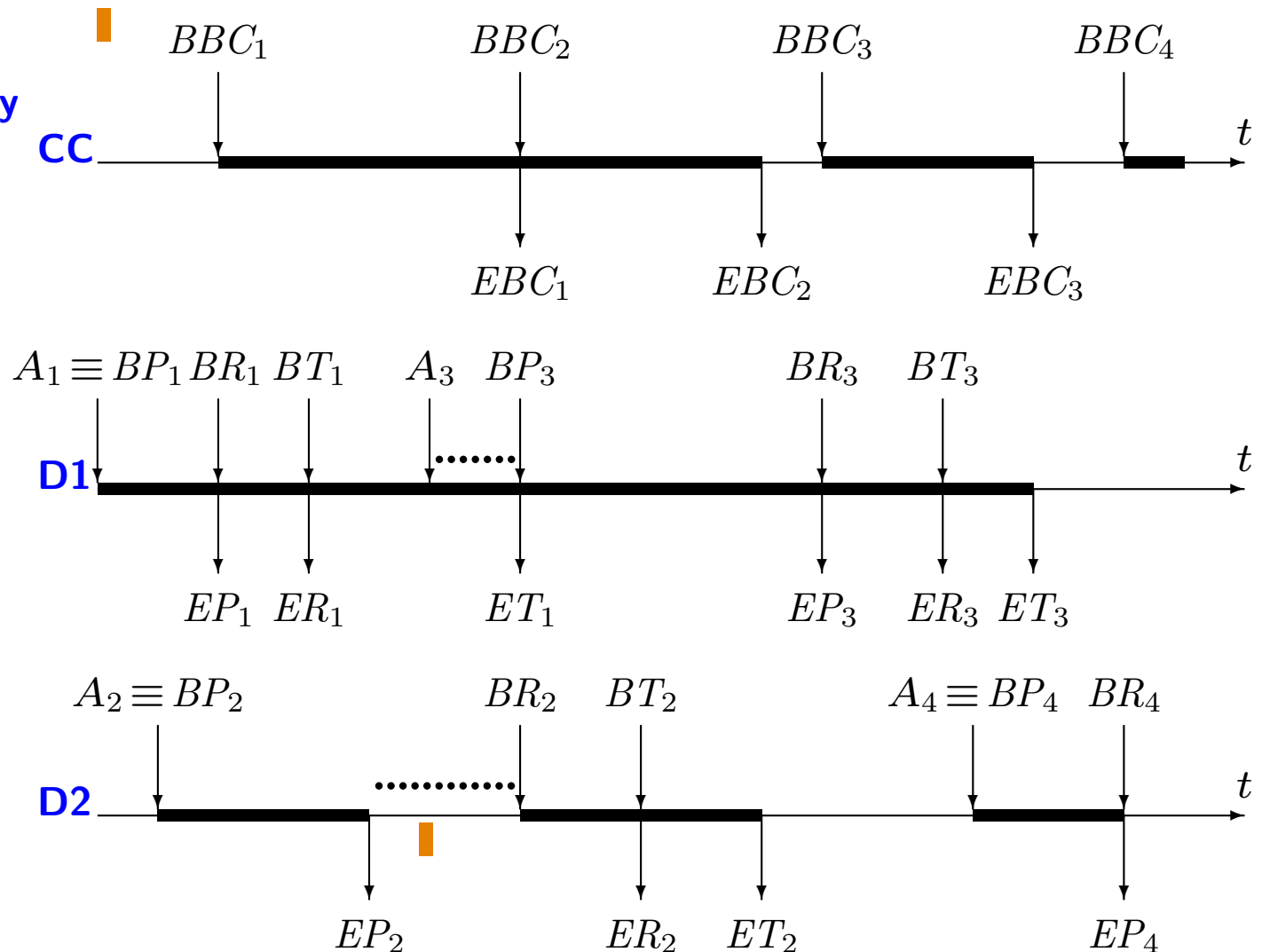
End *P*ositioning

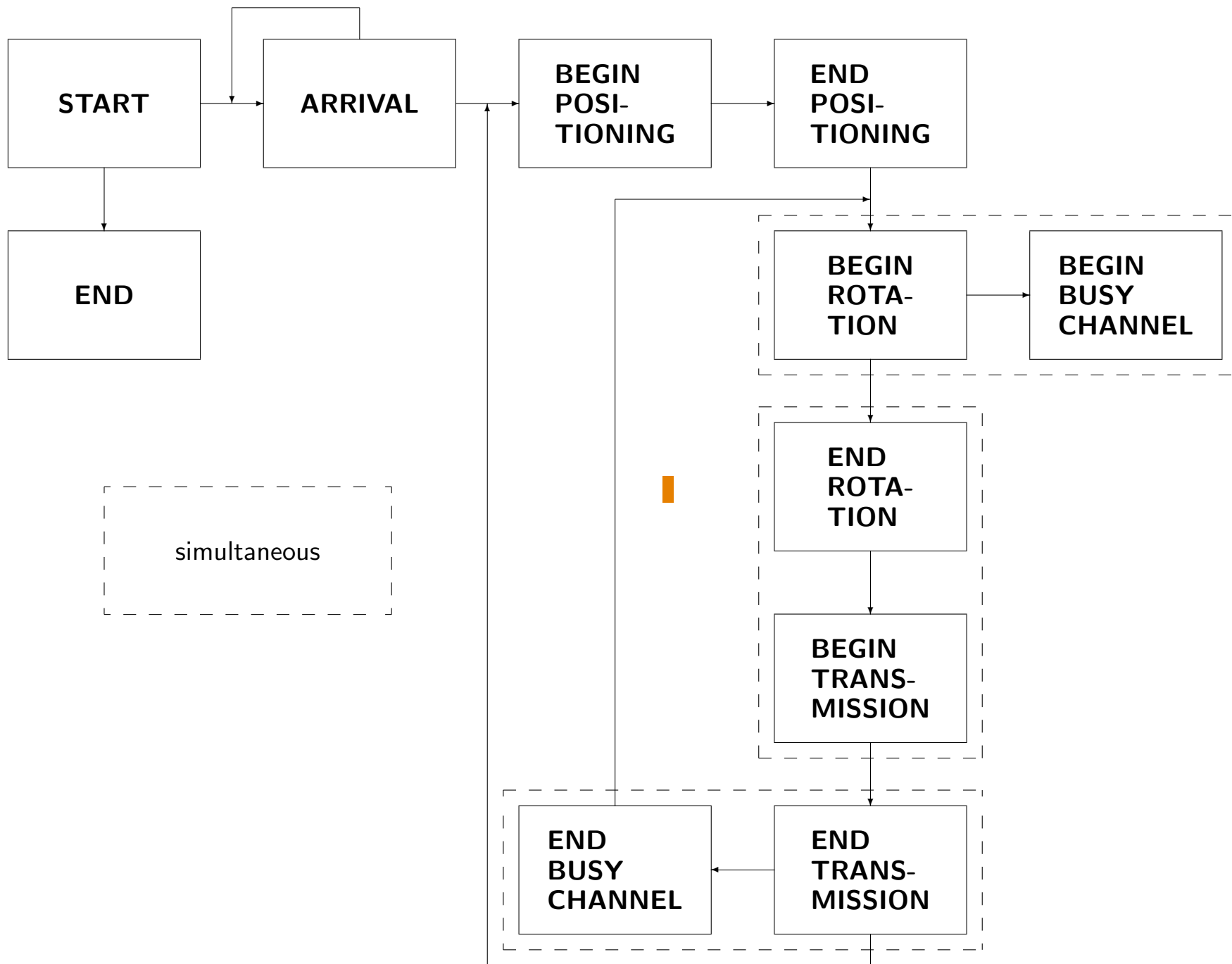
Begin *R*otation

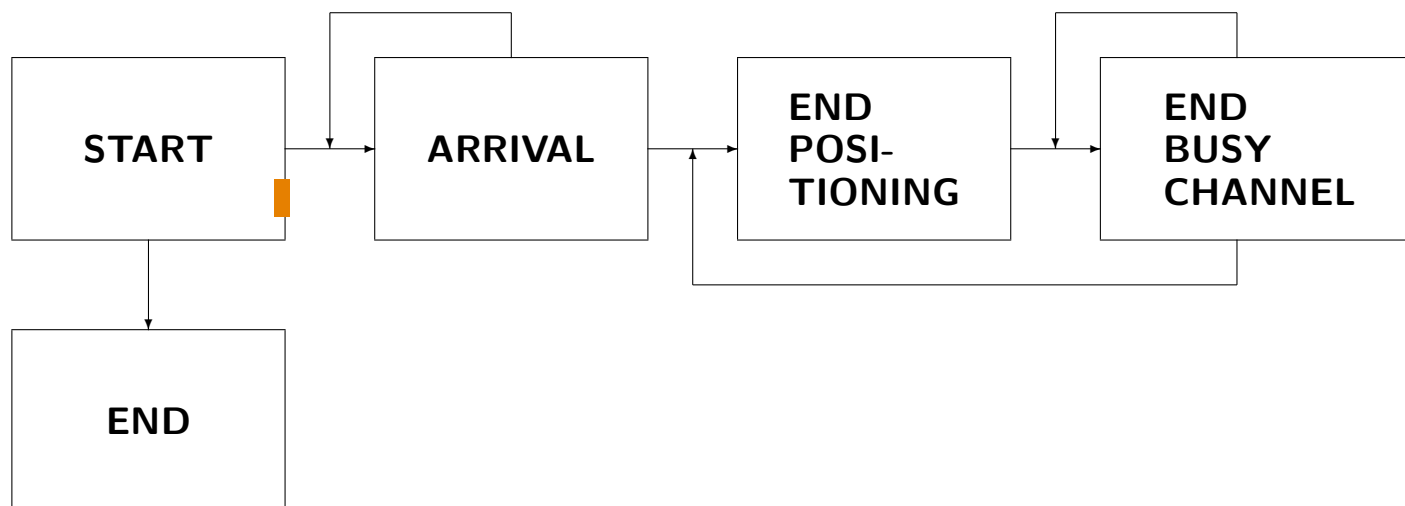
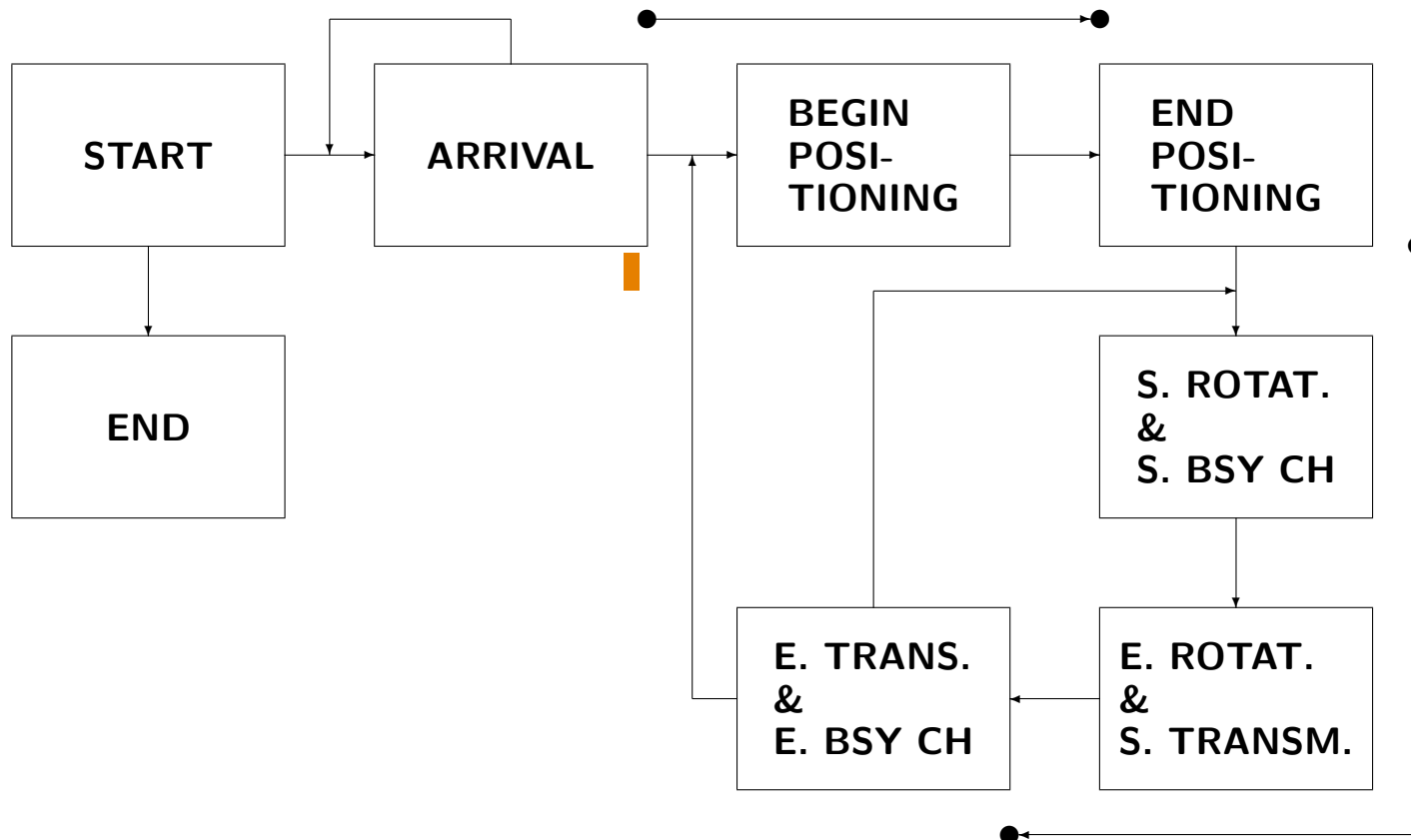
End *R*otation

Begin *T*ransmission

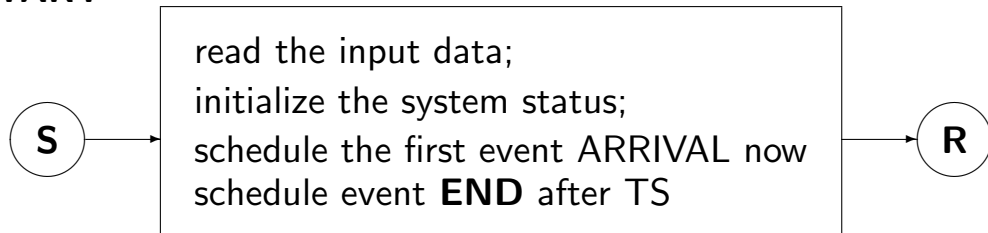
End *T*ransmission



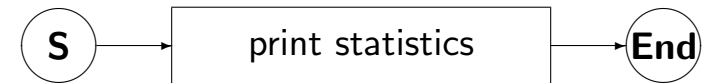




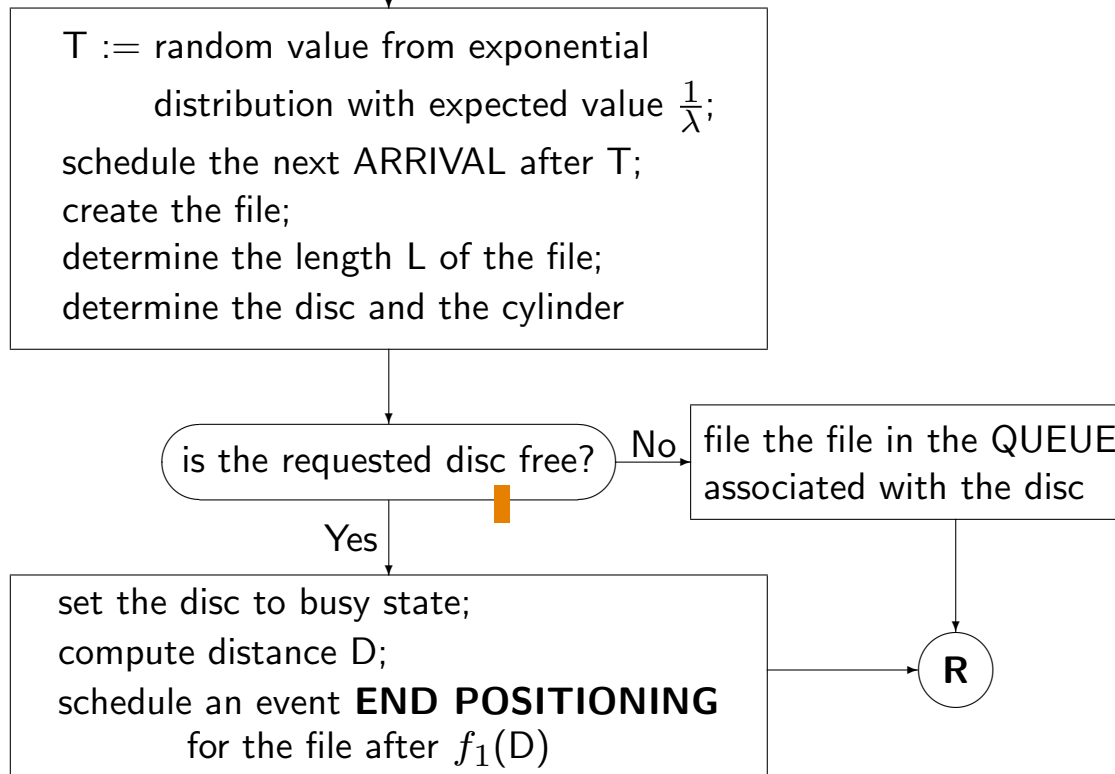
START



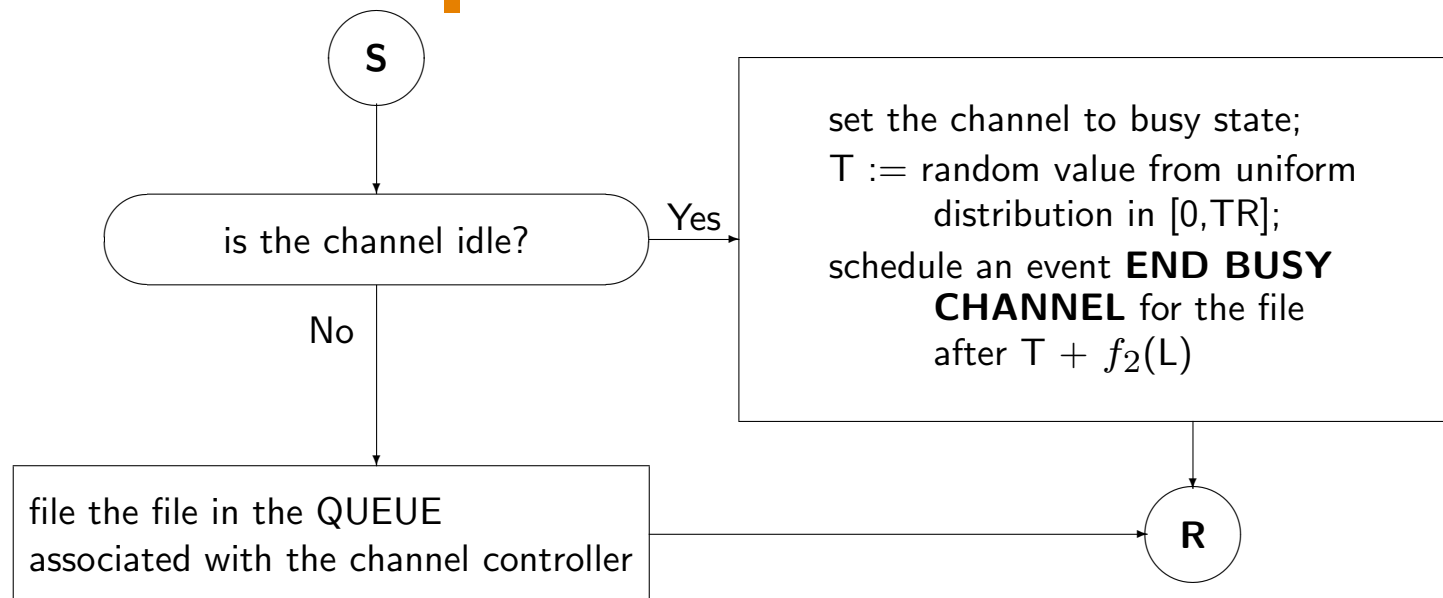
END

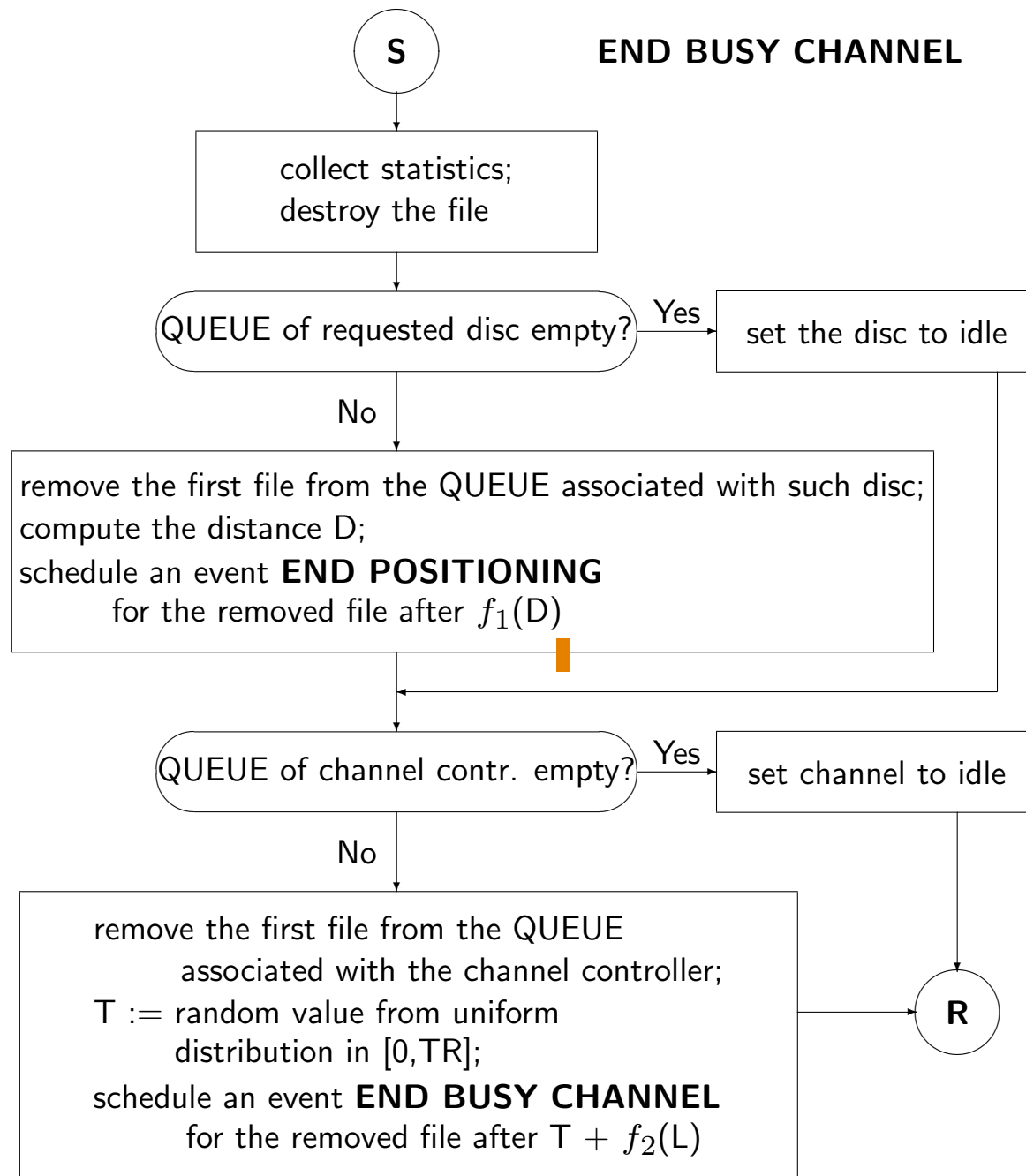


ARRIVAL



END POSITIONING





The main discrete simulation languages

Two logical approaches to discrete simulation:

- **Event scheduling**, preferable from a didactic point of view;
- **Process interaction**: Process = life of an entity = chronological sequence of events.

Imperative specialized languages:

- **Simula I, Simula 67** (process interaction, developed by **Dahl and Nygaard**):
 - * important for computer science: it introduced the concepts of **object** and **class**;
 - * it influenced the development of **C++**; rarely used today.
- **SIMSCRIPT I, SIMSCRIPT II.5** (event scheduling, developed by **Markowitz and Hausner**):
 - * the most powerful and widely used; it also allows process interaction.
 - * it influenced Simula, but it is easier to use and more efficient;

Interactive specialized languages (less versatile):

- **Arena** (process interaction):
 - * very simple to use; the model is implemented through **boxes** (\leftrightarrow flow chart);
- **SAS/OR** (event scheduling), integrated software for management.

General purpose languages (more difficult to use):

- Simulation can also be implemented with **general purpose languages** (e.g., **Java**).

Main simulation statements: Temporary entities

- We refer to SIMSCRIPT II.5 (Other languages have logically equivalent statements).
- Temporary entities are created when they enter the system, destroyed when they leave it.
- **CREATE** *[A, AN, THE]* *en* **[CALLED** *p*]
 - 1. reserves a new **block of consecutive words** for a new entity of class *en*;
 - 2. defines a **local variable** *p* containing the corresponding pointer;
if “CALLED *p*” is missing, the variable has name *en* (preferred for single entities).

Example: CREATE A CAR CALLED A1

CREATE A CAR CALLED A2

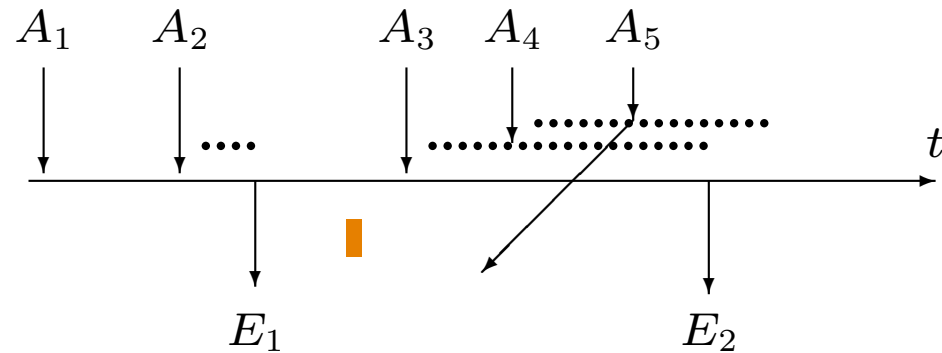
CREATE A CAR (equivalent to CREATE A CAR CALLED CAR)

- **DESTROY THE** *en* **[CALLED** *p*]
 - releases the reserved block of words pointed by *p* (by *en*, if “CALLED *p*” omitted)
- Temporary entities may have **temporary attributes**:
form: **attribute**(*p*)

Example: CREATE A CAR

LET TYPE(CAR) = X

Example: gas station, entities CAR having two attributes: TIS (entry instant time), TYPE (X or Y). ■



In A_1 :

CAR →	550
550	0.0
551	Y

After A_1 :

CAR →	
550	0.0
551	Y

In A_2 :

CAR →	575
550	0.0
551	Y
575	30.5
576	X

After E_1 :

CAR →	
550	
...	551
575	30.5
576	X

Main simulation statements: Permanent entities

- **System:**

- The System can have **permanent attributes:**

permanent attributes are *global variables*.

Example: NMAX, NLG, NLN.

- The System can own **sets with no index**

Example: QUEUE.

The system is automatically created, and is never destroyed.

- **“True” Permanent entities:**

- Permanent entities can have **permanent attributes** (global variables):

Form: like arrays (index = specific permanent entity).

Example: STATUS(J) = status (idle, busy) of disc J.

- Permanent entities are **created** by a single statement:

CREATE EVERY *en*;

must be preceded by the definition of $N.en$ (= number of entities of class *en*).

Example: READ N.DISC

CREATE EVERY DISC

Main simulation statements: Sets

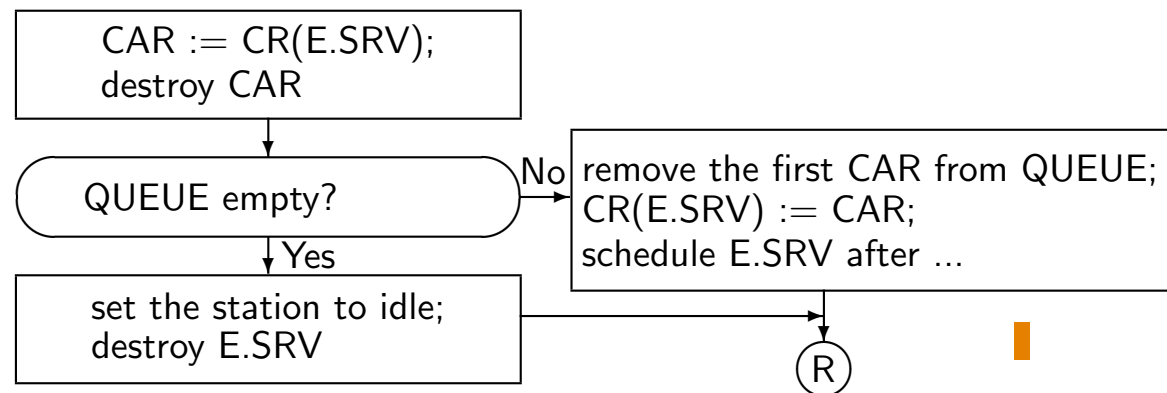
- **Sets** have members and owners: ■
- **members** are usually **temporary entities**; ■
- **owners** are usually **permanent entities**:
 - sets with no index are owned by **the system**; ■
 - sets with index are owned by **permanent entities**. ■
- **Sorting policies**:
 - **FIFO**;
 - **LIFO**; ■
 - **Ranked**: precedence given by the increasing or decreasing value of an attribute of the member entities. ■
- **FILE THE p IN THE s** inserts the entity pointed by p in set s ; ■
- **REMOVE THE FIRST q FROM THE s**
 1. removes the first entity from set s ; ■
 2. stores its pointer in a local variable named q . ■
- **REMOVE THE p FROM THE s** removes the entity pointed by p from set s . ■
- **IF THE s IS EMPTY I / IF THE s IS NOT EMPTY I** executes I if s is / is not empty. ■

Events and event notices

- **Exogenous events** are scheduled through input data (no need to reserve memory);
- each scheduled **Endogenous event** needs a block of words (time, entity pointer(s), . . .);
- each event has an associated special temporary entity having its name (**event notice**);
- event notices must be **created** before scheduling the event, and **destroyed** when the event occurs;
- event notices can have attributes (typically used to store pointers to the interested entities);
- when an event is executed, the system stores the pointer to the event notice in a local variable having the event name.
- **Example:** In Arrival:

```
set the station to busy state;  
T := random value from a uniform  
distribution in [T1,T2];  
create an E.SRV;  
CR(E.SRV) := CAR  
schedule E.SRV after T
```

In E.SRV:



Example: gas station

In A_1 :

CAR \rightarrow

550

E.SRV \rightarrow

800

550

0.0

551

Y

800

550

After A_1 :

CAR \rightarrow

--

E.SRV \rightarrow

--

550

0.0

551

Y

800

550

In E_1 :

CAR \rightarrow

--

E.SRV \rightarrow

800

550

0.0

551

Y

800

550

after CAR := CR(E.SRV)

CAR \rightarrow

550

E.SRV \rightarrow

800

550

0.0

551

Y

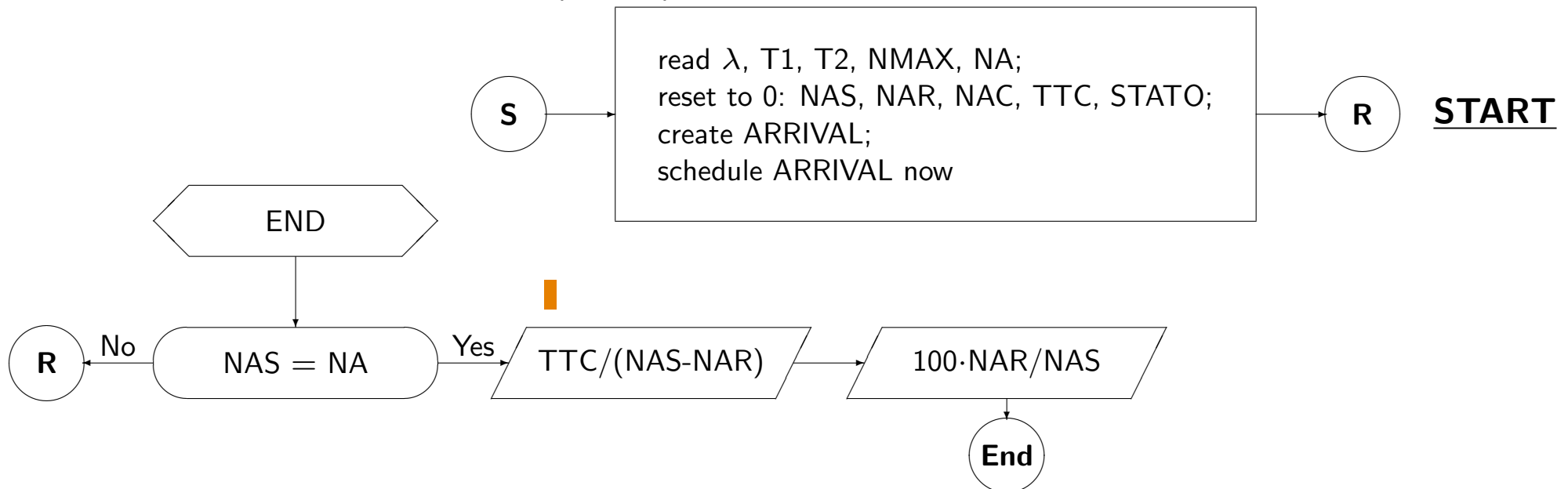
800

550

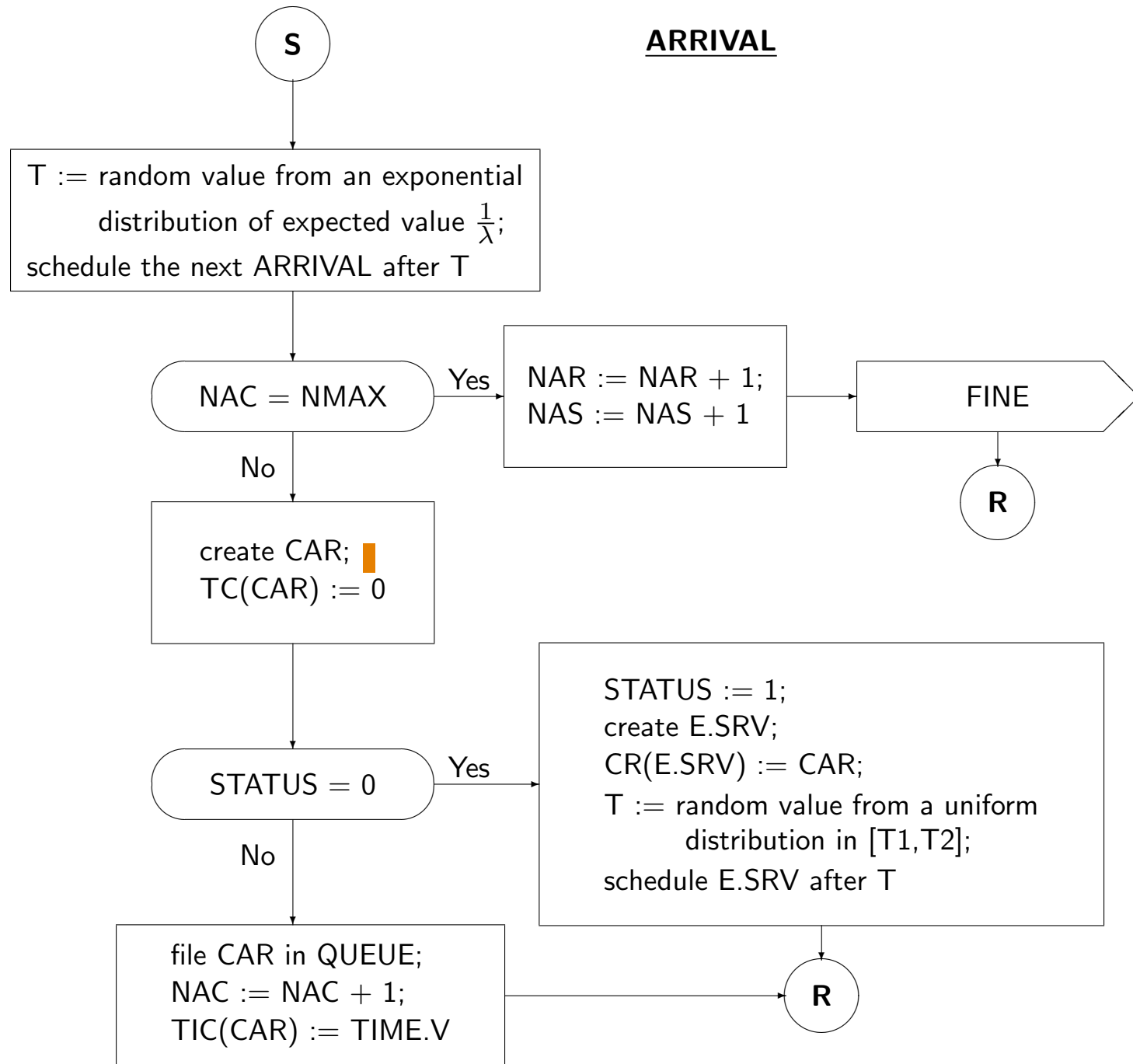
- **CREATE AN ev [CALLED p]**
 1. reserves a new **block of consecutive words** for a new event notice of class ev ;
 2. defines a **local variable** p containing the corresponding pointer;
if "CALLED p " is missing, the variable has name ev .
- **SCHEDULE THIS ev [CALLED p] AT t** schedules the event of pointer p (or ev) at t .
- **Note:** t = absolute time. Implemented as, e.g., SCHEDULE THIS E.SRV AT TIME.V + T
- **CANCEL THE ev [CALLED p]** cancels the event of class ev having pointer p (or ev), without destroying the event notice.

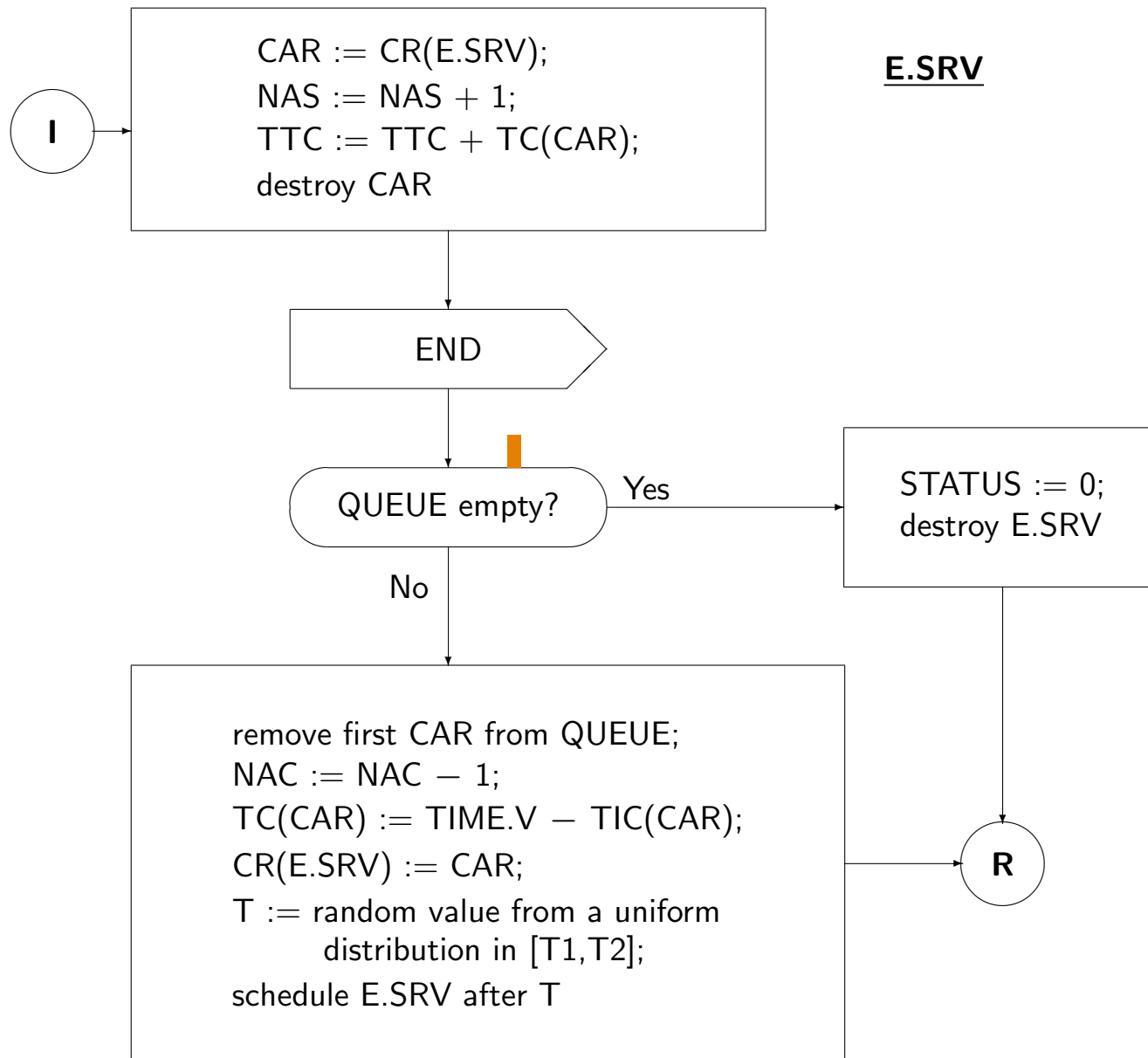
Example gas station revisited:

- Determine, over NA simulated cars, the percentage of rejected cars and the average queue time. ■
- CAR attributes: $TIC(CAR)$ = time instant the car enters the queue, $TC(CAR)$ (time in queue). ■
- System attributes: input data (λ , T1, T2, NMAX, NA); inner attributes:
 - NAS (number of simulated cars), NAR (number of rejected cars);
 - TTC (total time in queue), NAC (number of cars in queue);
 - STATUS (= 0 idle, = 1 busy); ■
- one FIFO QUEUE, owned by the System, with member entities CAR; ■
- events START and ARRIVAL have no attribute; ■
- event E.SRV has attribute $CR(E.SER)$ = pointer to the served car. ■



ARRIVAL





How do events exchange information?

There are **only three possibilities**:

1. **Permanent attributes**

Example: TTC is reset to 0 in START, updated in E.SRV, and used in END;

2. **Attributes of event notices**

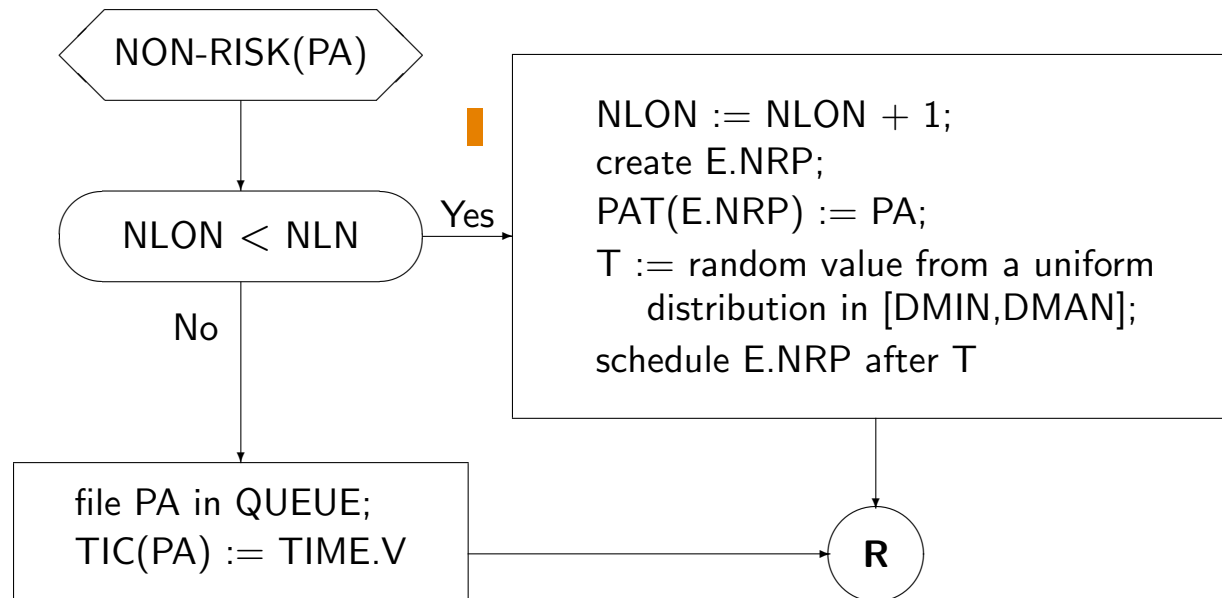
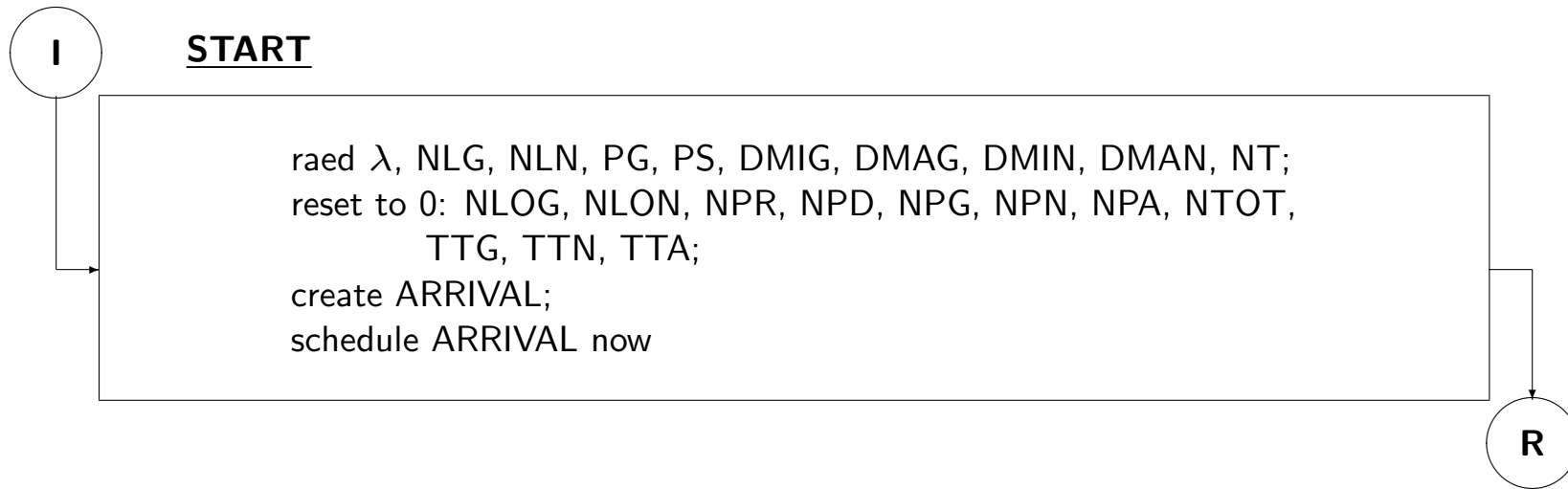
Example: CR(E.SRV) is defined in ARRIVAL and used in E.SRV;

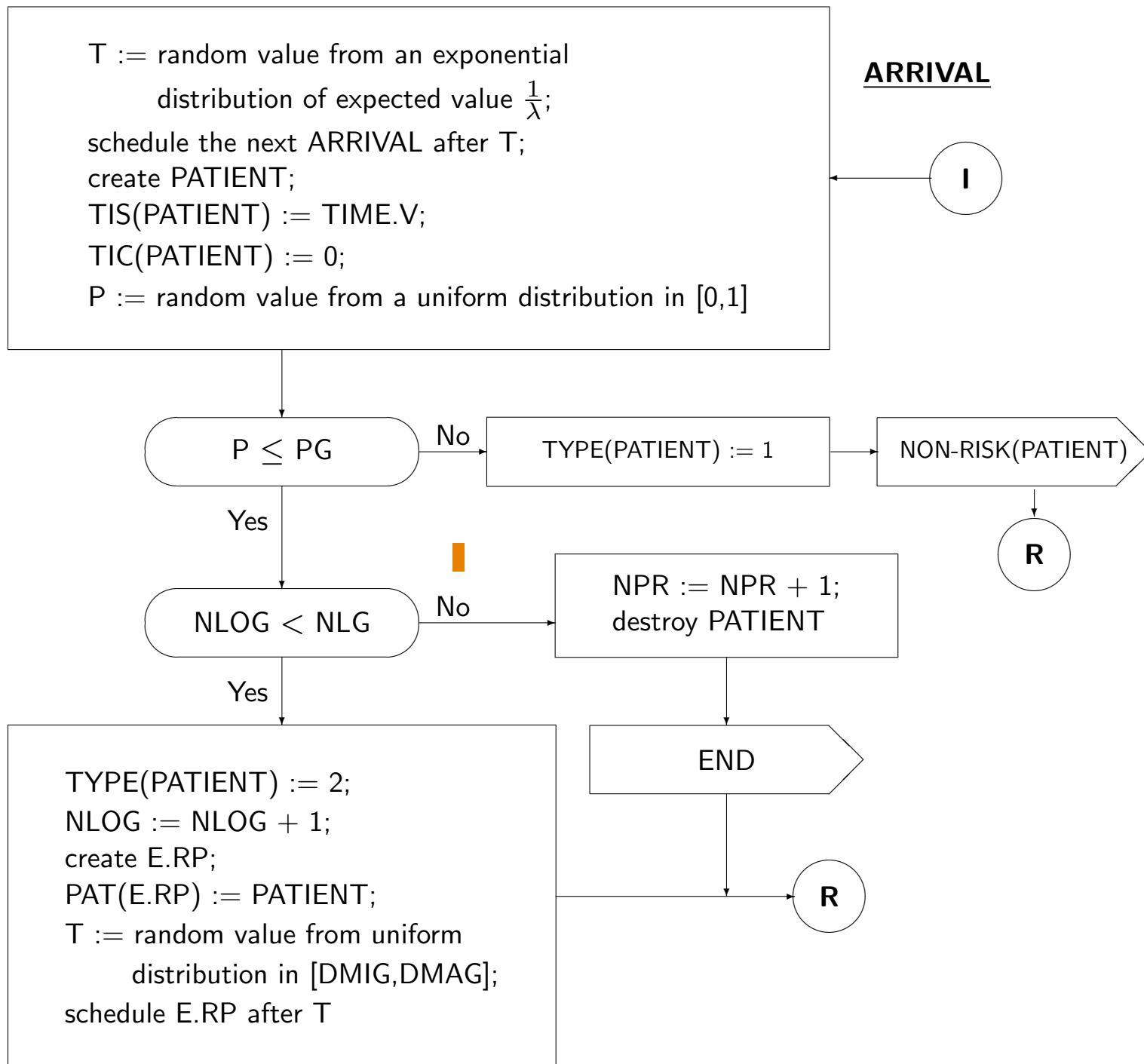
3. **Attributes of temporary entities** inserted and extracted from sets

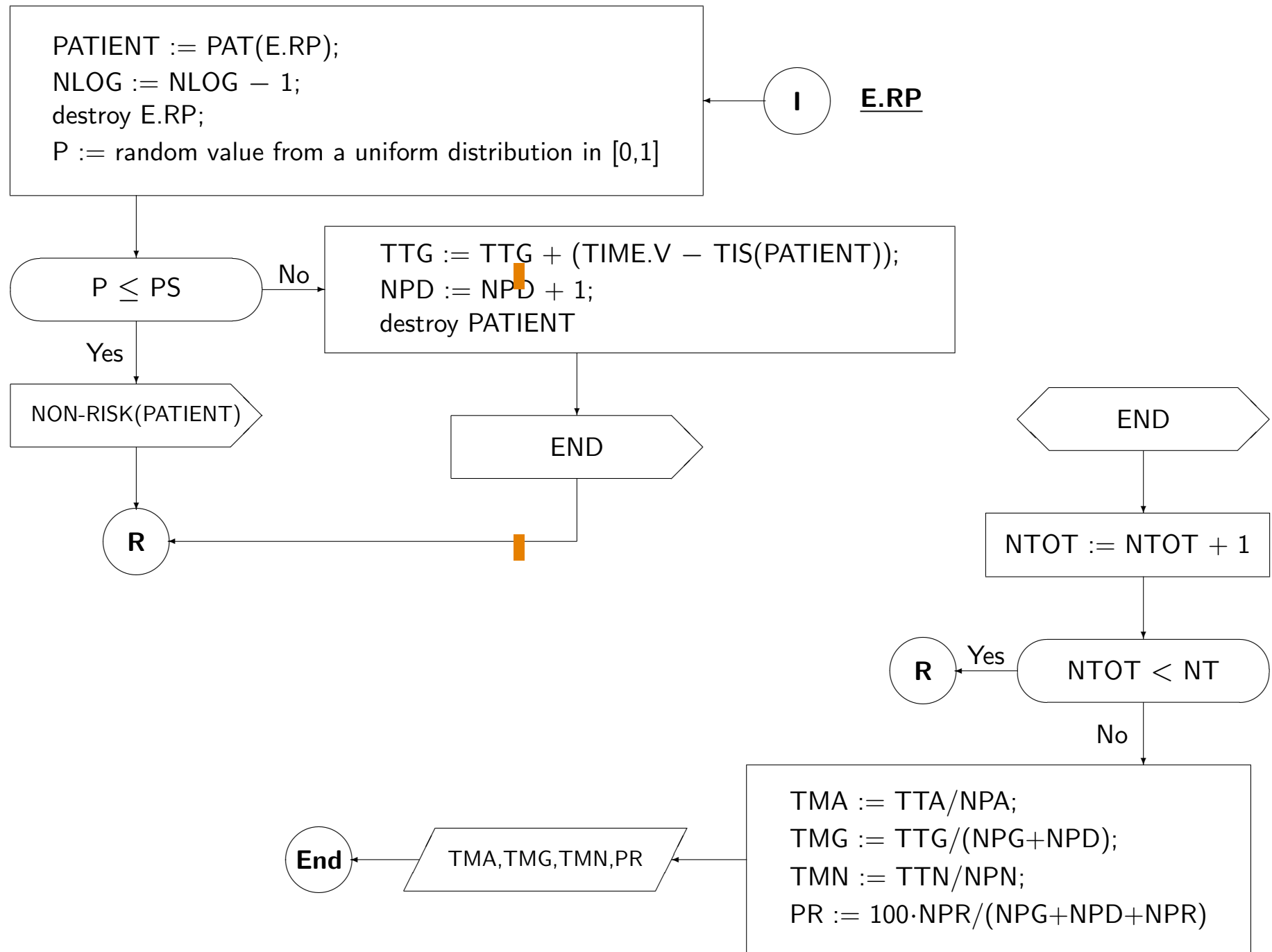
Example: CAR is inserted in QUEUE in ARRIVAL, and removed from it in E.SRV, where TIC(CAR) is used.

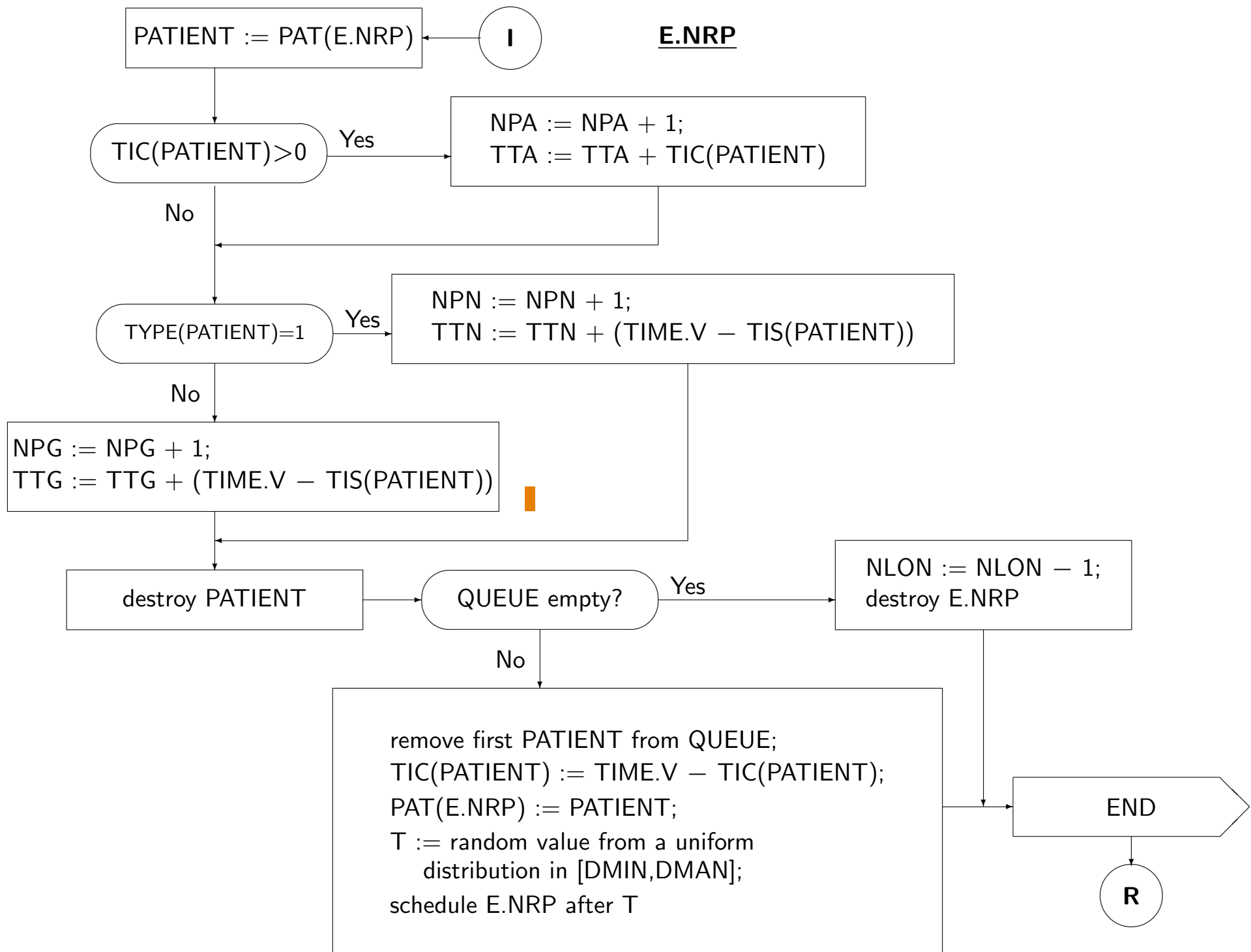
Example small hospital revisited:

- Determine, over NT simulated patients, ■
 - average queue time for the patients who queued;
 - average times in the system for risk and non-risk patients, respectively;
 - percentage of rejected risk patients. ■
- PATIENT attributes:
 - TIS(PATIENT) = time instant the patient enters the system,
 - TIC(PATIENT) = time instant the patient enters the queue,
 - TYPE(PATIENT) (= 1 if non-risk, = 2 if risk). ■
- System attributes:
input data (λ , NLG, NLN, PG, PS, DMIG, DMAG, DMIN, DMAN, NT); ■
inner attributes: ■
 - NPR, NPD and NPG (number of rejected, deceased and survived risk patients);
 - NPN (number of simulated non-risk patients),
 - NPA (number of patients who queued),
 - NTOT (number of simulated patients),
 - TTG and TTN (total time spent in the system by risk and non-risk patients),
 - TTA (total time spent in queue). ■
- one FIFO QUEUE, owned by the System, with member entities the PATIENTS; ■
- events START and ARRIVAL have no attribute; ■
- events E.RP and E.NRP have attribute PAT = pointer to patient ending a risk/non-risk stay. ■









Implementation and Experiments

- **Debugging:**

- not easy \Leftarrow difficult to spot logical errors \Leftarrow output not known a priori;
- use of prints in debugging to monitor the program execution: creation/destruction of entities, event scheduling, filing/removals from sets, . . .

- **Polarization:**

- if the system is initially in idle state, the first entities are advantaged: idle resources, (almost) empty queues, . . .
- \Rightarrow expected values depend on the simulation duration. Two methods to overcome this:
 1. create a reliable initial state: increased programming effort, experiments to test reliability; rarely used, unless an initial state is imposed;
 2. start in idle state but, after some time, reset the collected statistical data and perform the actual simulation.
experiments to decide the duration of the first phase (until final outcomes do not vary)

- **Experiments:**

- series of experiments with different input configurations to find the best one;
- preferable to change one (or few) input parameter(s) at a time;
- for each test, runs with different random sequences and mean of the expected values.

Pros and cons of simulation

Simulation is widely used as a decision support tool for systems whose behavior depends on random elements;

it is very general, in the sense that it is not based on any theoretical hypothesis, and hence it can be applied to very different contexts;

This methodology has many **Pros**. Indeed **Simulation**

- allows the impact of a decision on the system behavior to be evaluated without implementing it in practice;
- allows the analysis of the system evolution over time through time “compression”;
- is conceptually simple and flexible;
- can be used to analyze real-world systems that cannot be conveniently modeled through mathematical tools;
- allows the inclusion of every relevant element of the real system;
- allows quick answers to natural questions like “What happens if ...?”.

Pros and cons of simulation (cont'd)

The main **cons** of Simulation are:

- the development of good simulation models can be costly and time consuming;
- in order to provide useful evaluations, a good simulation model must include all the relevant system characteristics;
- simulation does not provide an optimal solution (like, e.g., LP or ILP). For a possible decision, it only provides the evaluation of the effect the decision has;
- in order to take an overall decision, managers must separately evaluate every possible choice, and then compare the results of all simulations to reach a final decision;
- a simulation model is generally very specific, and it is unlikely that it can be adapted to different situations.

Simulation it is one of the most widely used Operations Research tools. Together with

- **Linear Programming** and **Integer Linear Programming**, and
- methods based on **Graphs and Networks** (treated in the course *Network Optimization M*)

it is adopted by many industry managers as a decision support tool for many real world systems.

Good luck with your exam! Thank you for your attention