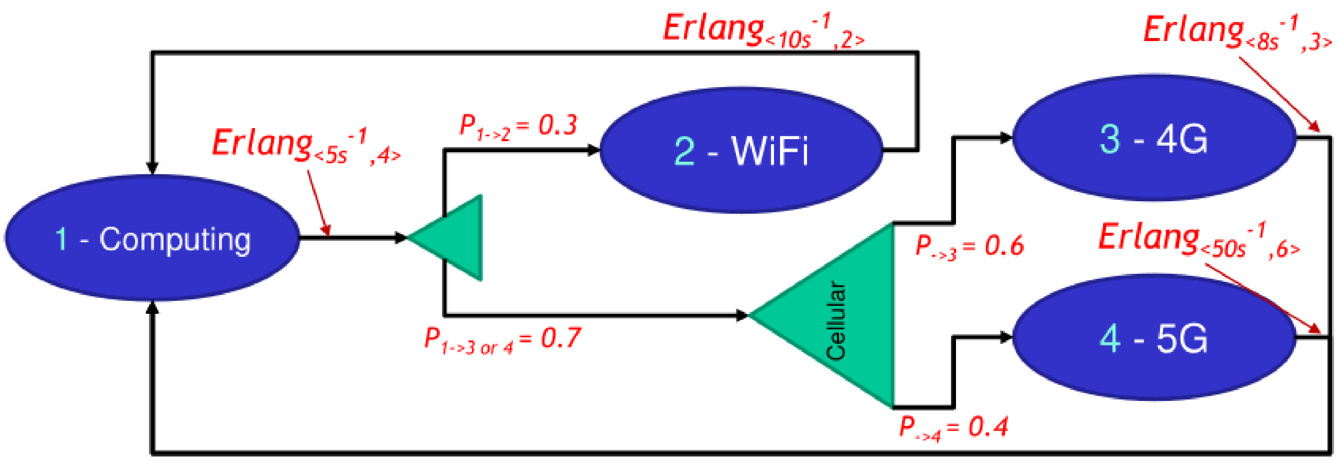
11.Race, Concurrency and the Exponential DIstribution

We now consider a mobile application. We need to consider different cellular applications.



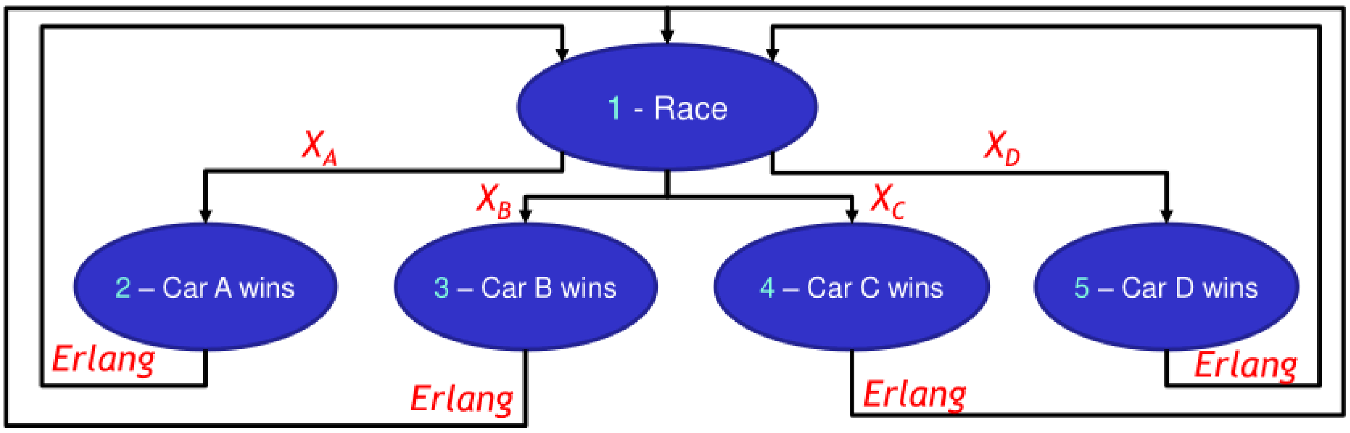
We would like to compute:

- State probabilities
- Throughput: trip that this application can do
  In this case, the choice between the transition from state 1 to either state 2, 3 or 4, is guided by a discrete random variable.
  $P_{1\to2}$ = 0.3
  $P_{1\to3}$ = 0.7 . 0.6 = 0.42
  $P_{1\to4}$ = 0.7 . 0.4 = 0.28
  The system is similar to the one we have done before:

```
Algorithm that handles the state machine
if s = 1 then
        ns = 2u = rand()
        if u < 0.3
                ns = 2;
        else if u < 0.72
                ns = 3;
        else
                ns = 4;
        end
dt = GenErlang(5,4);
end
        if s = 2 then
                ns = 1;
                dt = GenErlang(10,2);
        end
if s = 3 then
```

Race

Whenever we have different distribution that start at the same time.



The idea is that when the race start we will determine the random completion time of the racers and determine each time the winning racer.
We would like to compute

- Victory probability for each car
- Average duration of a match.

```
if s = 1 then
        tA = GenXA();
        …
        tD = GenXD();
        if (tA == min(tA, tB, tC, tD)) then
                ns = 2; dt = tA;
        end
        if (tB == min(tA, tB, tC, tD)) then
                ns = 3; dt = tB;
        …
end
if s = 2 then
        ns = 1;
        dt = GenErlang();
end
if s = 3 then
        ns = 1;
        …
        …
        if s = 1 then
```

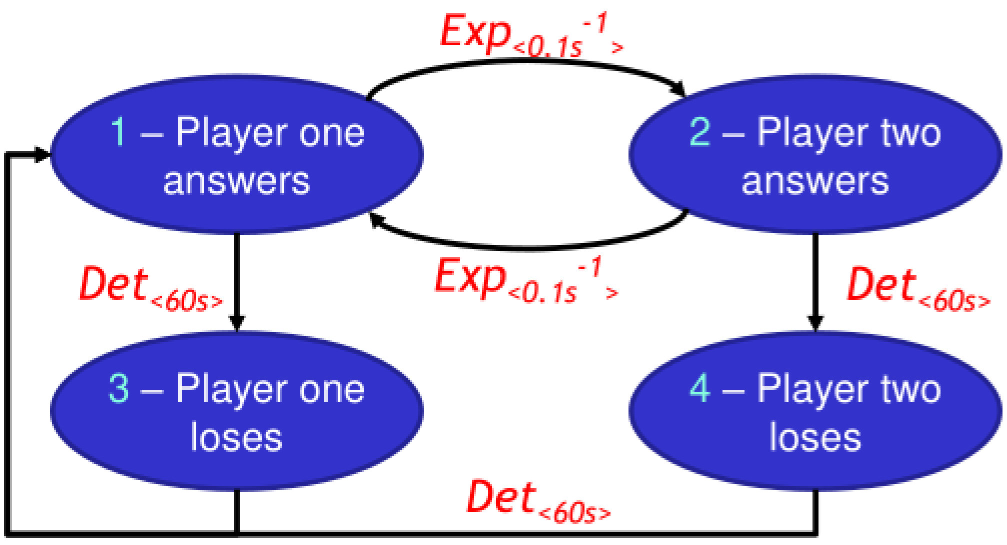
```
                if (tA == min(tA, tB, tC, tD)) then
                        ns = 2; dt = tA;
                        WinA ++;
                end
                …
                Races ++;
                RT(CurrRace,1) = dt;
                CurrRace ++;
        end
…
PwinA = WinA / Races;
PwinB = WinB / Races;
PwinC = WinC / Races;
PwinD = WinD / Races;
AvgRT = mean(RT);
```

In this case, the choice of the next state is determined by the so called Race policy:

- the next state is chosen according to the path that has extracted the minimum time
  This is possible only in very few cases, in which all the transitions starts at the same time (such in this example), or when we have some special firing time distribution for the event.

L'eredità exxample

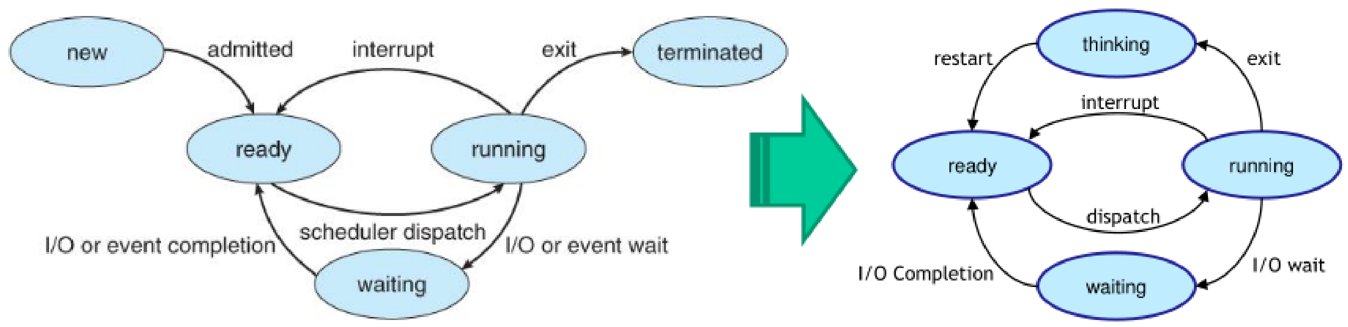
We want to simulate a two-contestant game: two people have 60 s. each, in which they have to answer a set of questions. Each time a player gives the correct answer, the turn passes to the other and hers timer stops. The player for which the timer expires first loses the game. Each player finds the correct answer in an exponentially distributed amount of time, with an average of 10 s. Player one starts, and when a contestant loses, the game restarts after a deterministic time of 1 min.



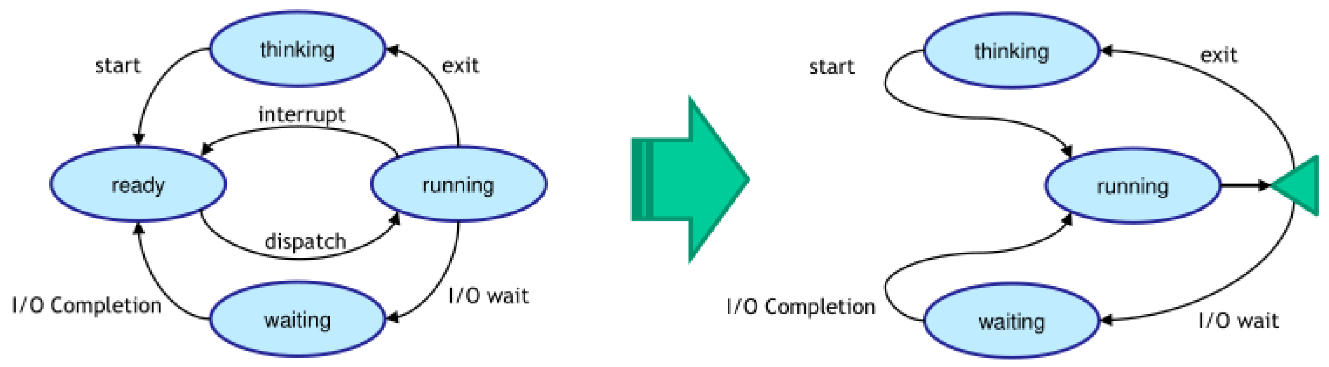
Concurrency

There are activities that run in parallel. When we have concurrency we need to decide which event happens first.

Concurrency can occur in many forms, and in general it cannot be handled with simple solutions like the ones previously seen. Consider for example the state machine for the operative system process that we saw last time, applied to a batch process (a closed model): whenever a job ends, it restarts after a “think time”.

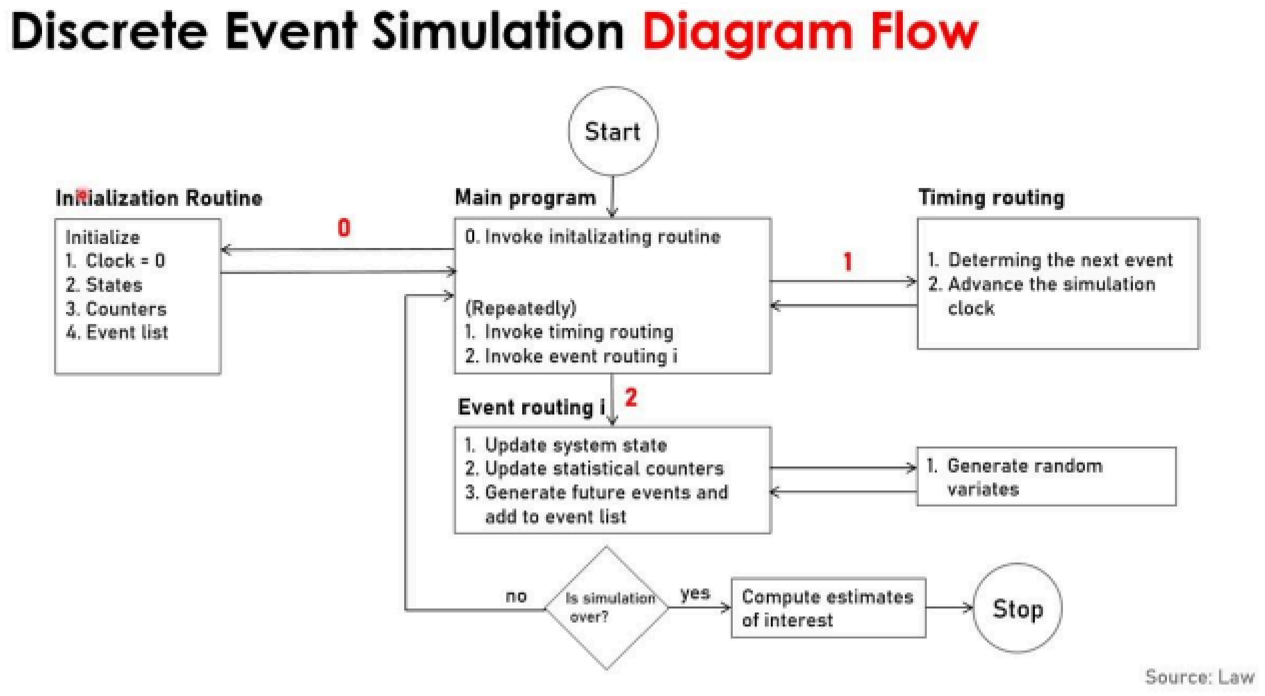


In case of a single batch process, jobs can be dispatched immediately, and the system would be in the “ready” state for a negligible amount of time. Such state could be removed. Selection between “I/O wait” and “exit” in the running state can be performed using choice: a distribution of running before an event, and couple of probabilities for either ending or doing I/O.



In case of more jobs, different types of concurrency occur in almost all the states, and choice cannot be used to solve the problem in all states.

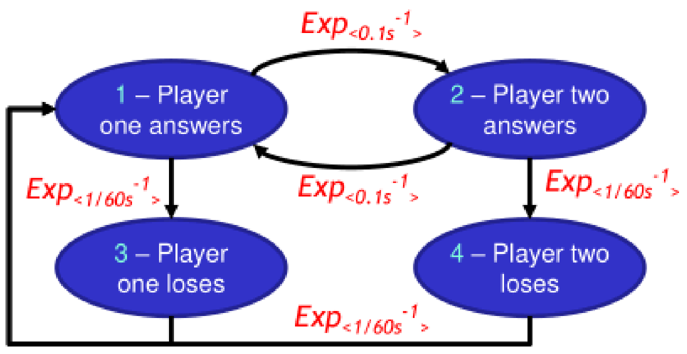
Concurrency in general can be handled using Discrete Event Simulation. A complete description on how to perform Discrete Event Simulation, is however outside the scope of this course.


Discrete Event Simulation Diagram Flow
Start
Initialization Routine
Initialize
1. Clock = 0
2. States
3. Counters
4. Event list
0
Main program
0. Invoke initalizating routine
(Repeatedly)
1. Invoke timing routing
2. Invoke event routing i
1
Timing routing
1. Determing the next event
2. Advance the simulation clock
Event routing i
2
1. Update system state
2. Update statistical counters
3. Generate future events and add to event list
1. Generate random variates
no
Is simulation over?
yes
Compute estimates of interest
Stop
Source: Law

Exponential distribution and concurrency

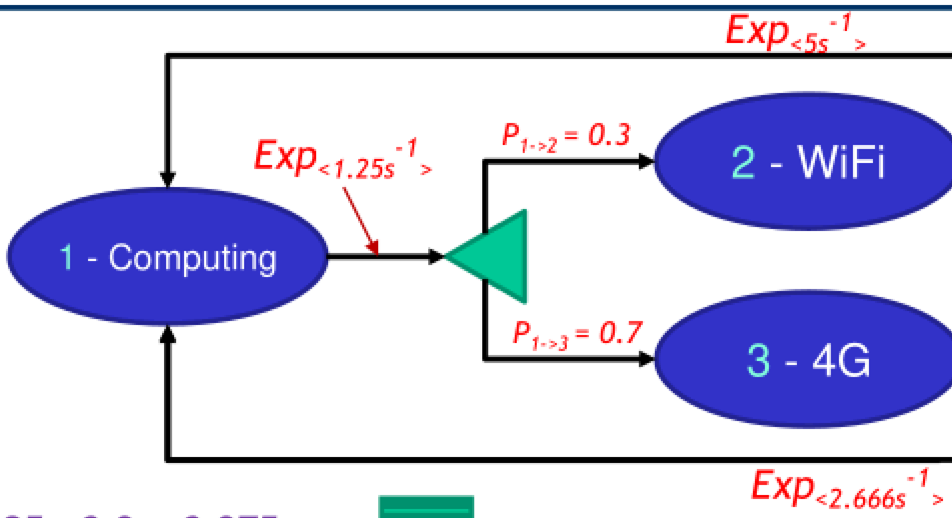
When ALL random timing follows an exponential distribution, analyzing systems become simpler thanks to its memory less property:

- Race policy becomes identical to concurrency
- Choice can be implemented with race policy, by embedding selection probability into the rates.
  Race policy becomes identical to concurrency: for example, if in the “l’Eredità” game the timer would be exponentially distributed with an average of one minute, it could be analyzed in the following way:

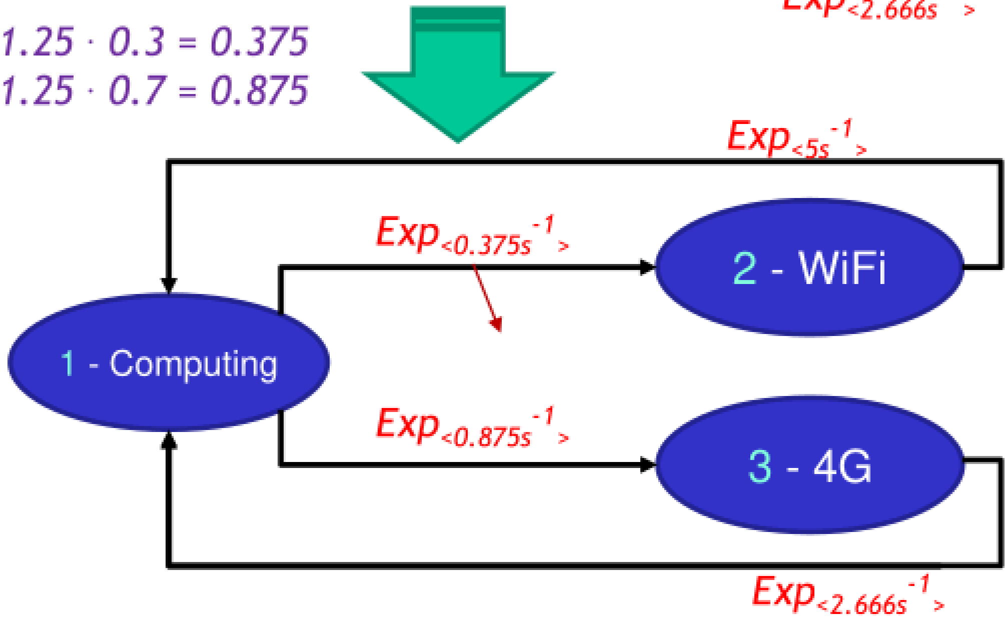


```
…
if s = 1 then
        tA = GenExp(0.1);
        timer = GenExp(1/60);
        if (tA == min(tA, timer)) then
                ns = 2; dt = tA;
        else
                ns = 3; dt = timer;
        end
        …
end
if s = 2 then
        …
end
if s = 3 then
        ns = 1; dt = GenExp(1/60);
```

Choice can be implemented with race policy, using the product of the original exponential rate parameter with the selection probability. For example, a 4G only version of the remote app model become:



$1.25 \cdot 0.3 = 0.375$
$1.25 \cdot 0.7 = 0.875$



$Exp_{<5s^{-1}>}$
$Exp_{<0.375s^{-1}>}$
2 - WiFi
1 - Computing
$Exp_{<0.875s^{-1}>}$
3 - 4G
$Exp_{<2.666s^{-1}>}$