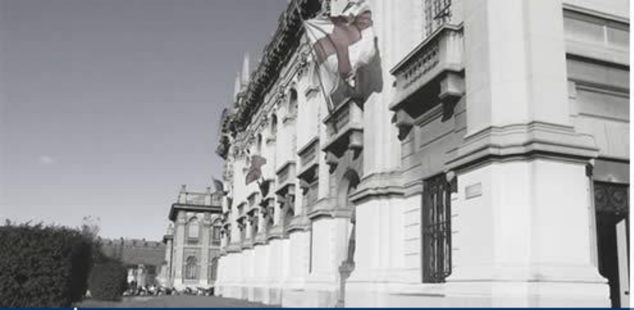


Performance Evaluation and Applications

POLITECNICO DI MILANO

State Machines in Performance Modelling

Motivating example

I have a simple system, like a small embedded device, or a mobile phone app.

I have collected traces, and found distributions for services and arrivals.

Even if very simple, my system is too complex to be described by just a couple of distributions.

How can I model a system whose components have been measured using distributions?

State machines in Performance modelling

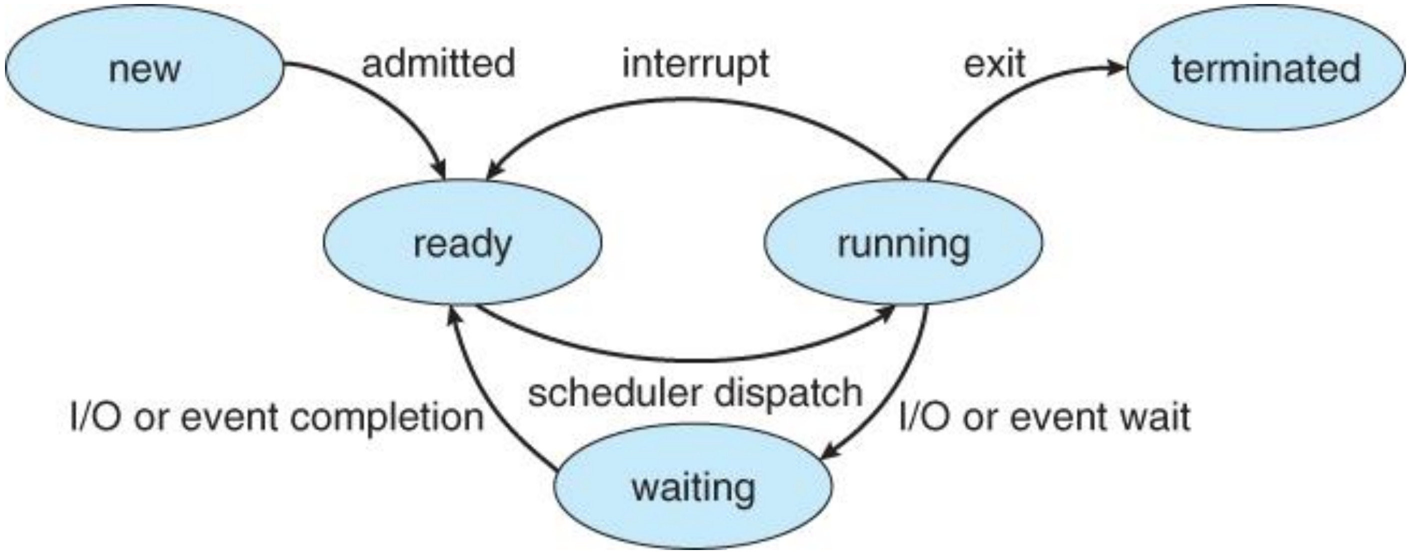
State machines are one of the most used tools in Engineering to formalize sequential processes.

The system is abstracted by *set of states* in which it can be.

Transitions determines the way in which the considered system changes state.

State machines in Performance modelling

Usually state machines are represented with *directed graphs*, where nodes describe states, and edges correspond to transitions.

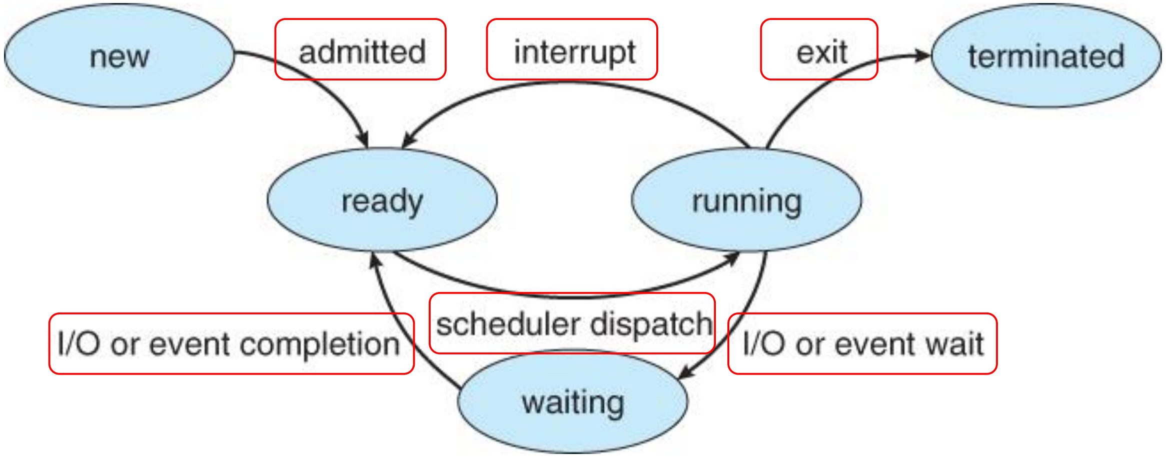


The state machine of a Unix process

State machines in Performance modelling

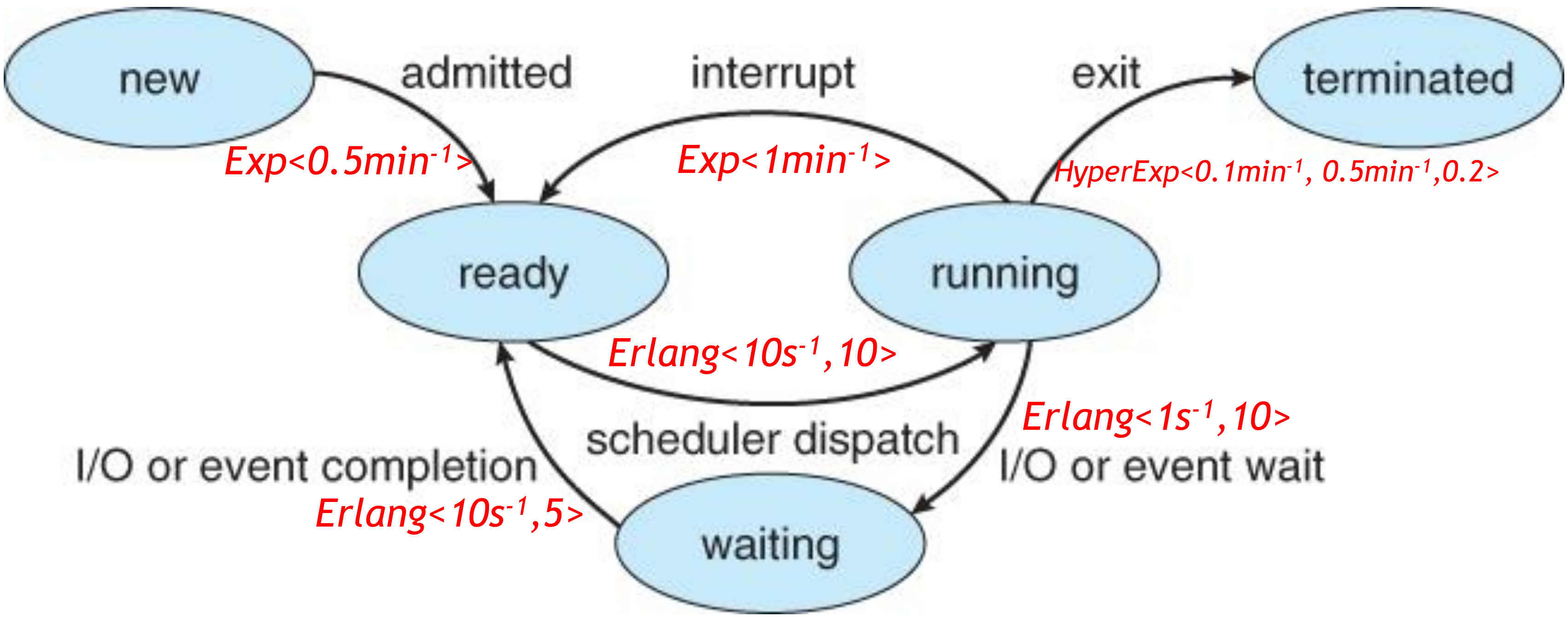
Transitions are usually triggered by *events*:

- In formal languages, by symbols read from the input
- In sequential circuits, by values in their input gates
- In flow control, by special conditions or signals received



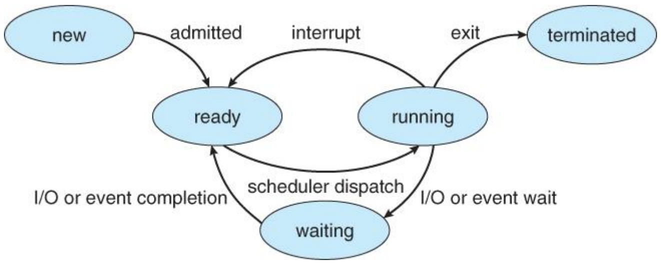
State machines in Performance modelling

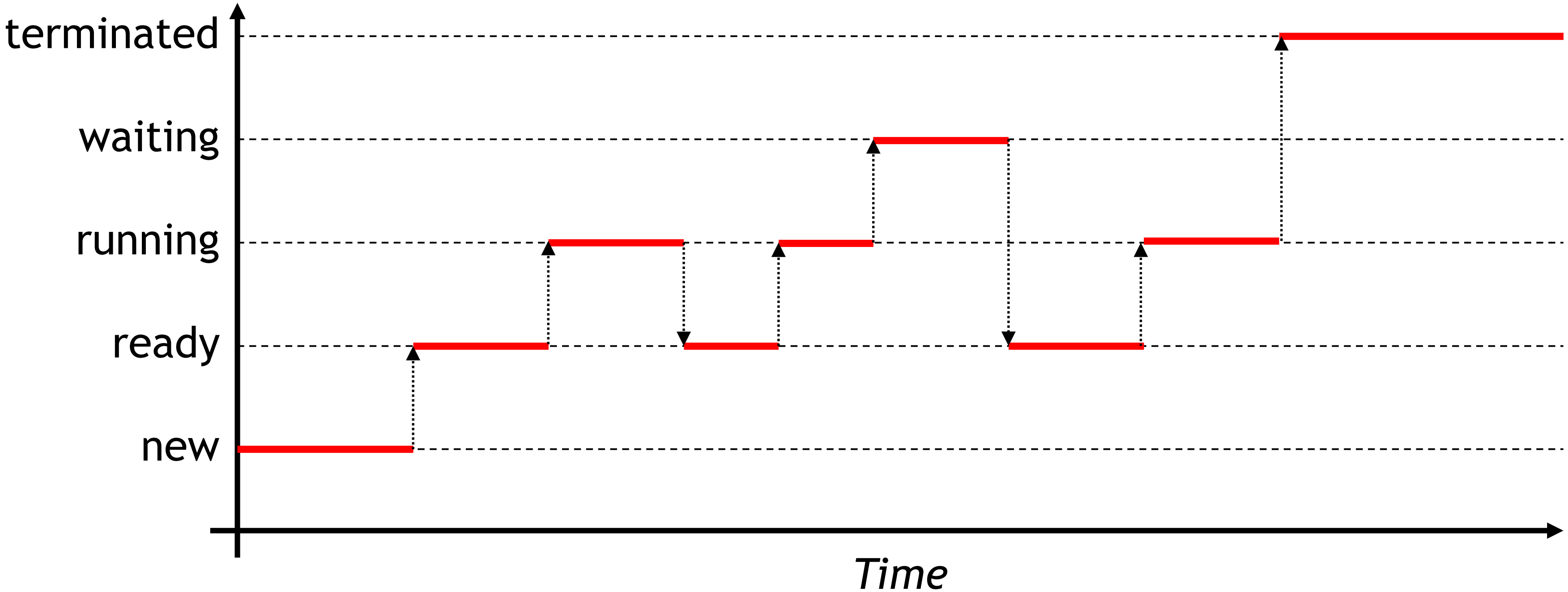
In performance modelling, transitions are generally associated with random variables: they represent the time after which the corresponding event will occur.



State machines in Performance modelling

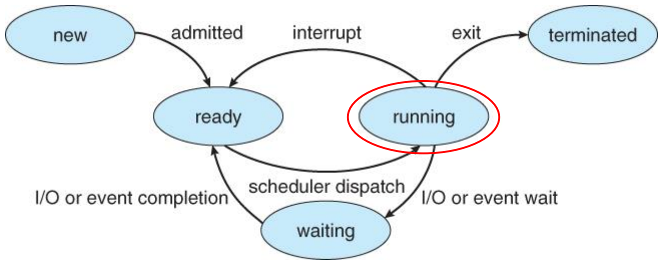
Using the given distributions and the corresponding state machine, a *synthetic trace* of the evolution of the system can be generated.

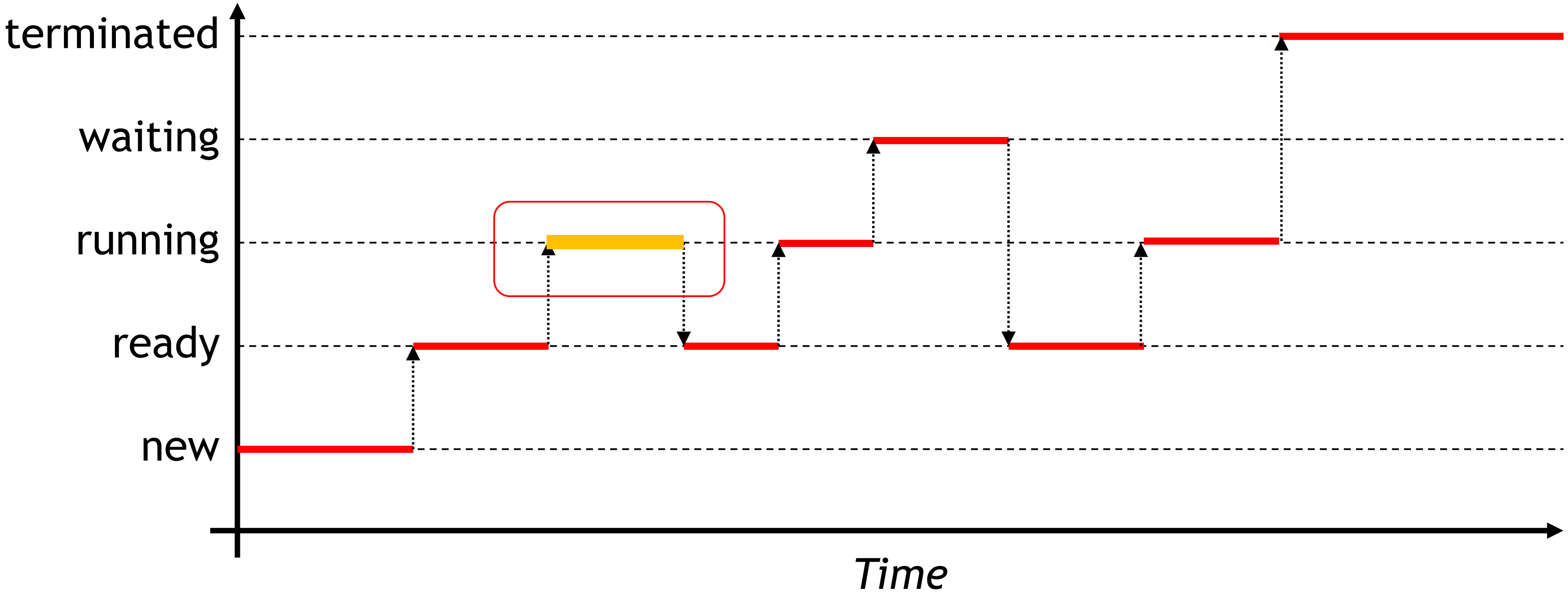




State machines in Performance modelling

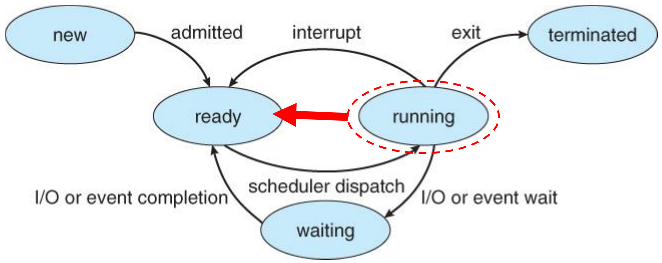
In particular, the system stays in a state for a given (random) amount of time.

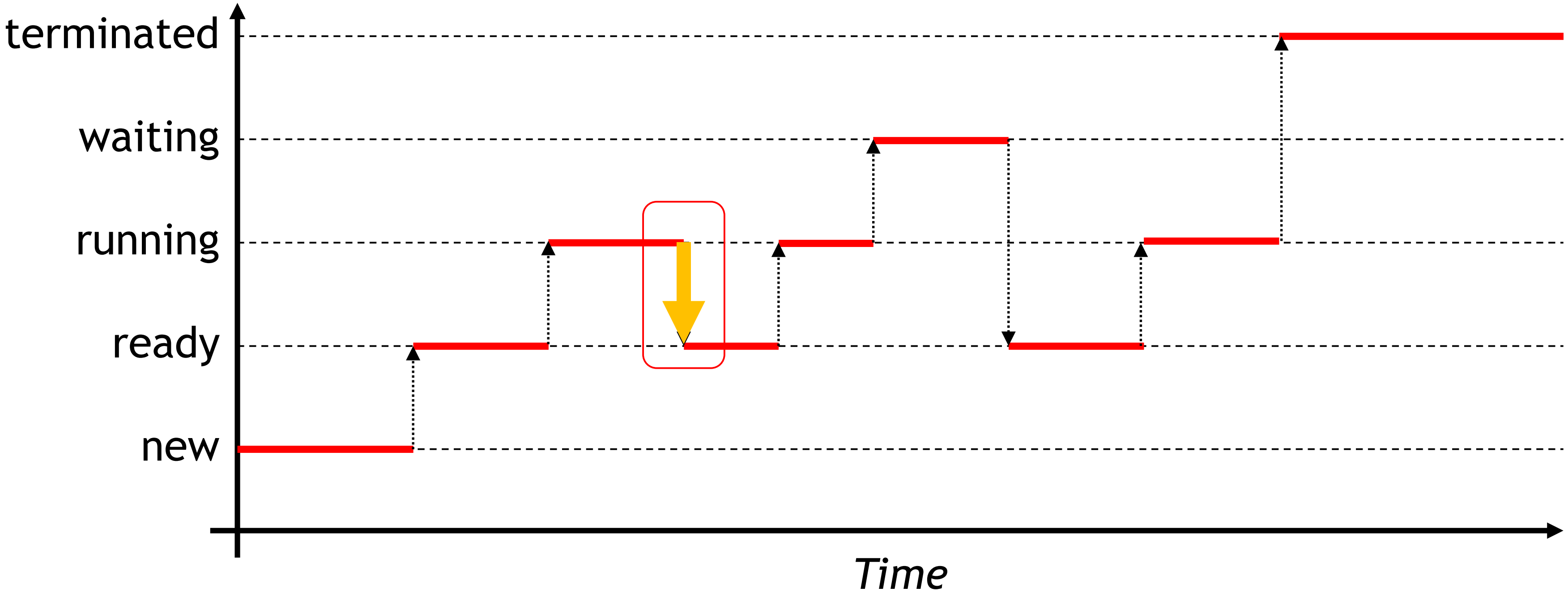




State machines in Performance modelling

Then it jumps to another state, where it stays for another (random) amount of time.





State machines: performance indices

From the states being visited, various performance metrics can be derived.

The main ones includes:

- Fraction of time spent by the system in a given state
- Frequency at which a given transition was triggered

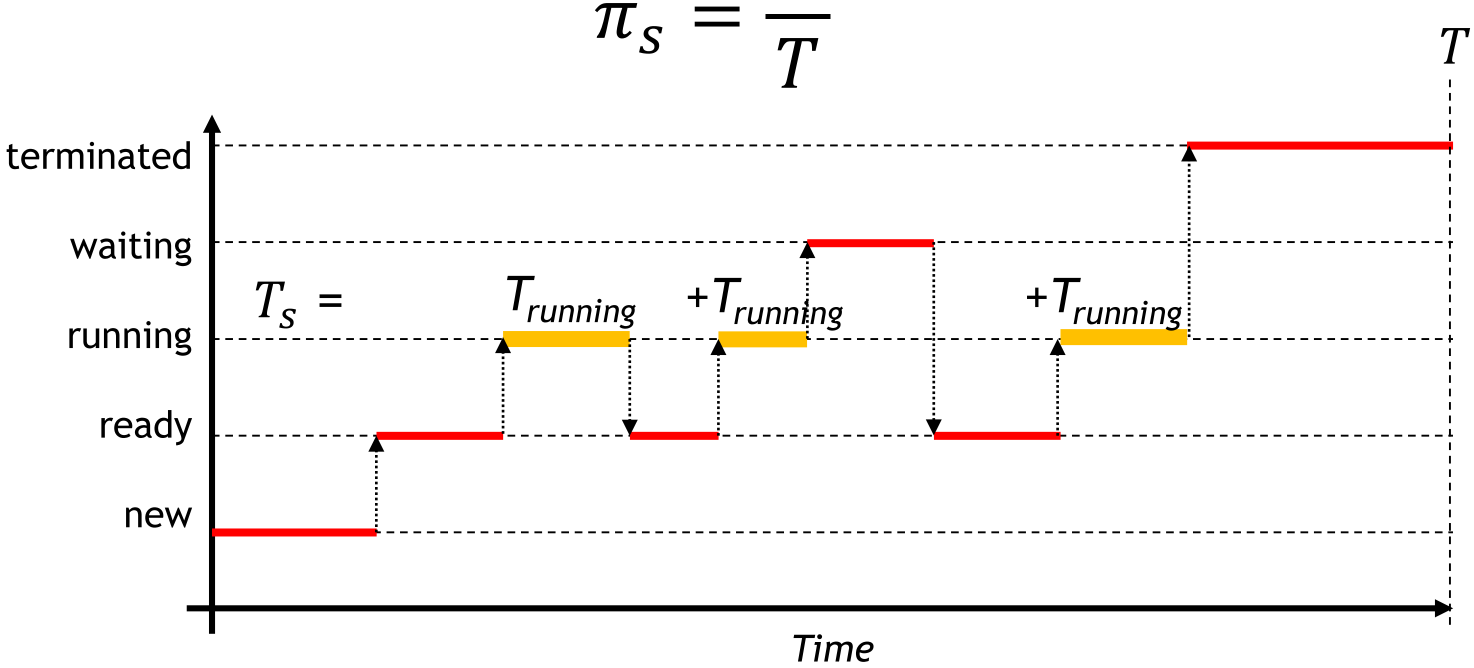
From those basic measurements, many other interesting performance metrics can be derived: we will return on them later.

State machines: performance indices

The fraction of time spent π_s in a given state *s* can be determined in a simple way.

Let us call T_s the time spent by the system in state s during the time of experiment *T*. Then:

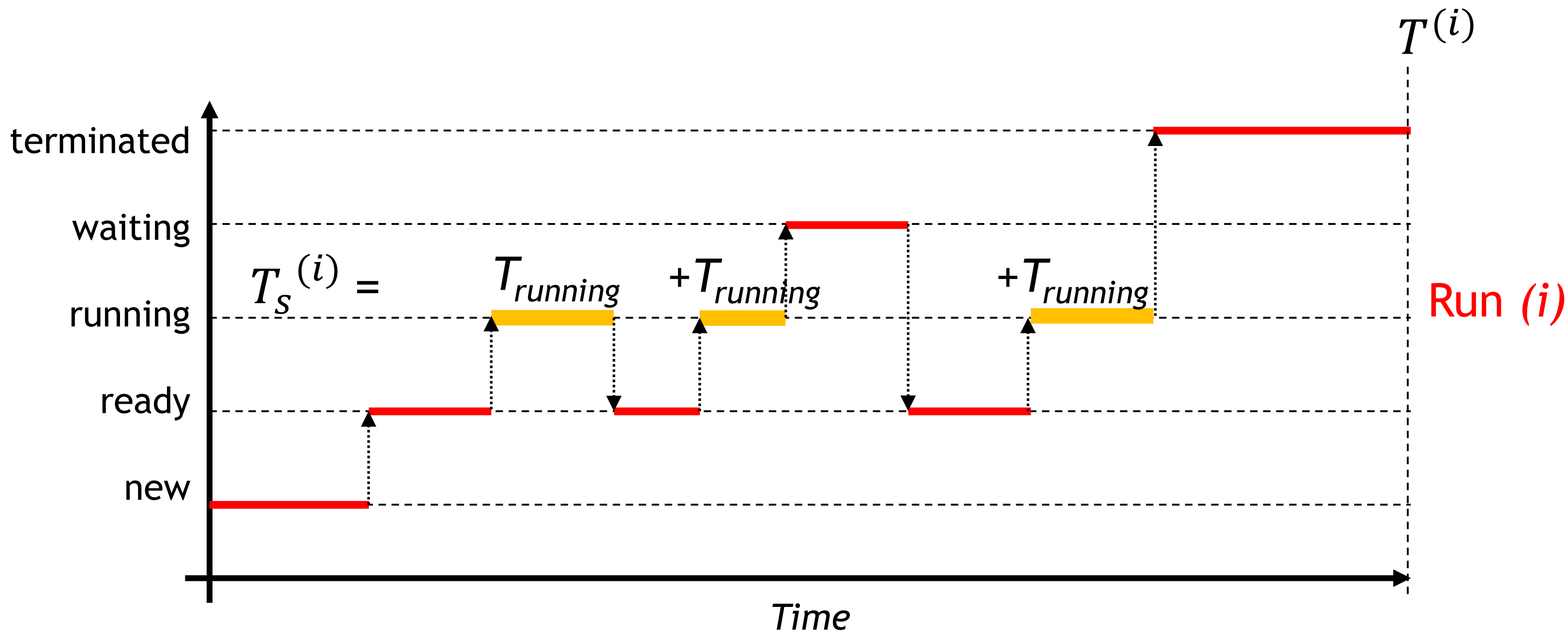
$$\pi_s = \frac{T_s}{T}$$



State machines: performance indices

Please note that, since random numbers are used, we should always compute confidence intervals, and never rely on simple averages.
However, to simplify the discussion, we will only focus on the average measures.

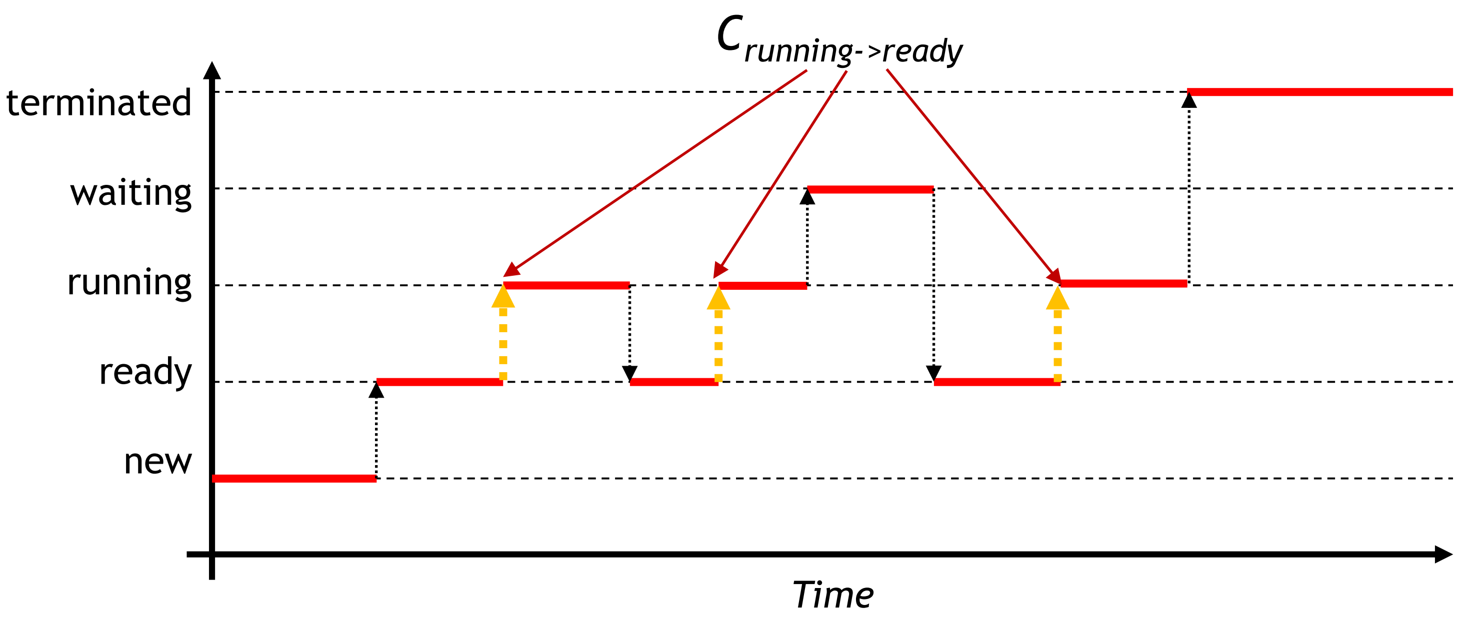
$$\pi_s{}^{(i)} = \frac{T_s{}^{(i)}}{T^{(i)}} \qquad \pi_s \in \left(\frac{\sum_{i=1}^{K} \pi_s{}^{(i)}}{K} \pm d_\gamma \sqrt{\frac{\sum_{i=1}^{K} (\pi_s{}^{(i)})^2 - K\left(\sum_{i=1}^{K} \pi_s{}^{(i)}\right)^2}{K(K-1)}}, \right)$$



State machines: performance indices

The frequency at which a transition *e* was triggered ϕ_e can be determined starting from C_e, the count of times the considered change of state occurred.

$$\phi_e = \frac{C_e}{T}$$



State machines: analysis

Although seeming simple, the generation and analysis of synthetic traces produced from state machines is not trivial.

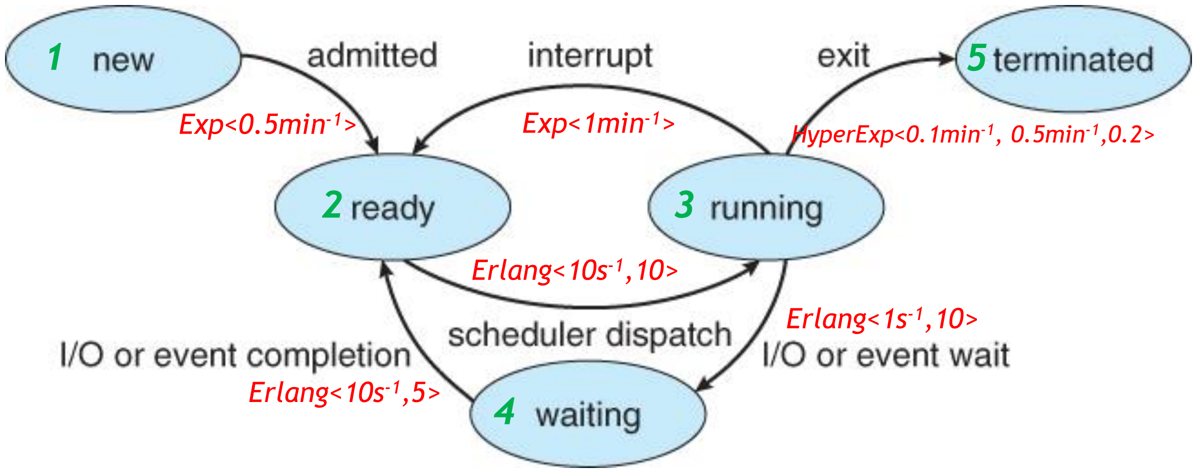
Several formal ways of presenting and considering state machines in Performance modelling have been developed in the literature.

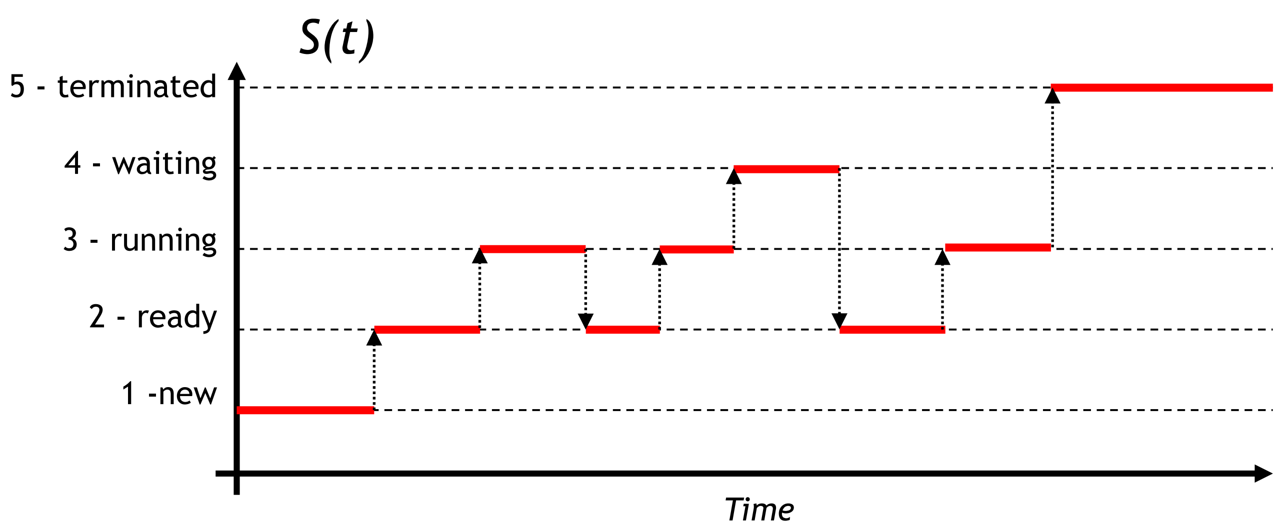
Here, we will give only an informal presentation, to outline the main difficulties and to introduce the techniques that will be considered later in the course.

State machines: analysis

In general, we start enumerating the states.

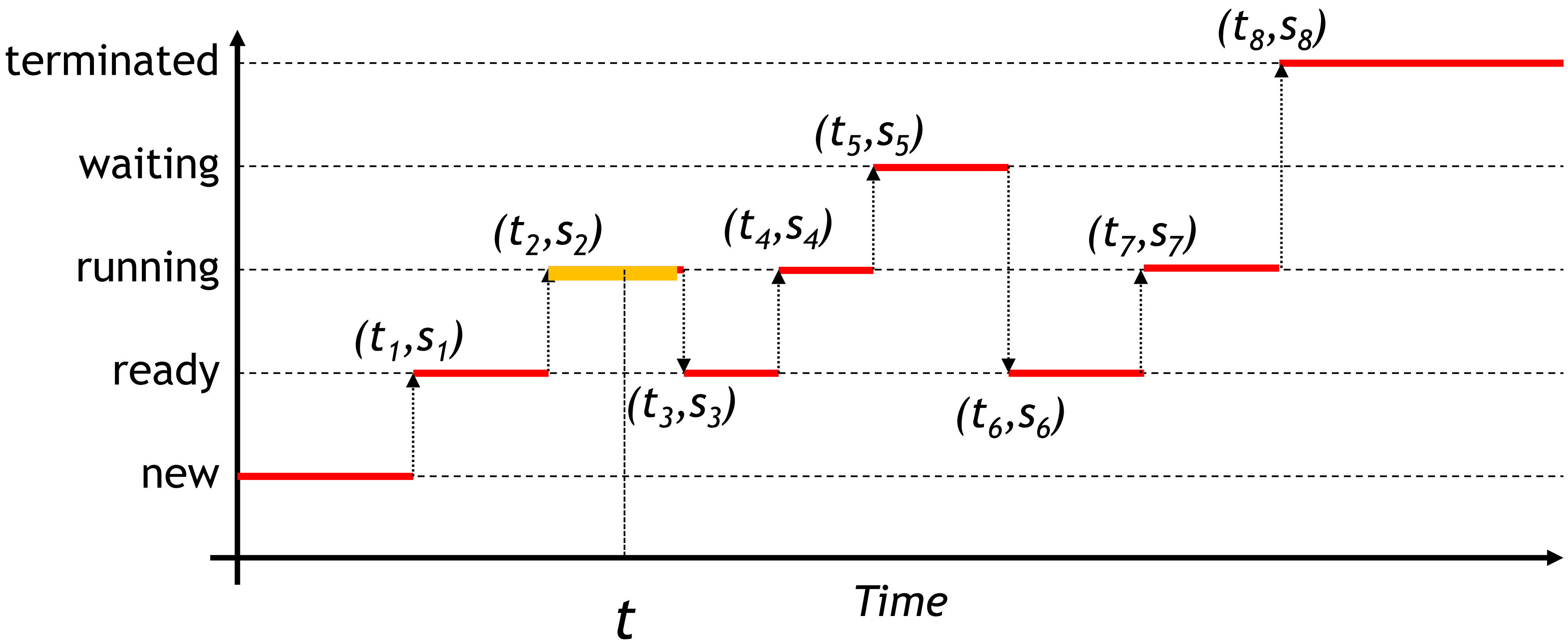
A *trace* can then be considered a function of time $S(t)$, that returns the number corresponding to the state in which the system is for each t .





State machines: analysis

The system remains in the same state for some time *T*, before jumping to the next one: the trace is thus a step function, which can be easily encoded by an array of couples (t_i, s_i) representing the time t_i at which we system enters state s_i due to a state change.



State machines: analysis

The general high-level algorithm to produce a trace can then be the following:

```
s = s0;                     % Variable s contains the current state
t = 0;                      % Variable t contains the current time
i = 0;                      % Variable i contains the index of the current output couple
while t < T do              % Until the end of trace time T
    if s = 1 then
            ns = newstateFromStateMachine…
            dt = durationFromDistribution…
    end
    if s = 2 then
    …
    i = i + 1;
    s = ns;                         % Move to next state
    t = t + dt;                     % Increase time
    Trace(i,:) = [t, s];            % Store in the trace
end
```

State machines: analysis

Two variables `s` and `t` are used to hold the current state, and the current time.

```
s = s0;                    % Variable s contains the current state
t = 0;                     % Variable t contains the current time
i = 0;                     % Variable i contains the index of the current output couple
while t < T do             % Until the end of trace time T
     if s = 1 then
              ns = newstateFromStateMachine…
              dt = durationFromDistribution…
     end
     if s = 2 then
     …
     i = i + 1;
     s = ns;                        % Move to next state
     t = t + dt;                    % Increase time
     Trace(i,:) = [t, s];           % Store in the trace
end
```

State machines: analysis

The main loop is repeated until the target time T is reached.

```
s = s0;                     % Variable s contains the current state
t = 0;                      % Variable t contains the current time
i = 0;                      % Variable i contains the index of the current output couple
while t < T do              % Until the end of trace time T
    if s = 1 then
            ns = newstateFromStateMachine…
            dt = durationFromDistribution…
    end
    if s = 2 then
    …
    i = i + 1;
    s = ns;                     % Move to next state
    t = t + dt;                 % Increase time
    Trace(i,:) = [t, s];        % Store in the trace
end
```

State machines: analysis

Depending on the current state, the next state n_s and the time d_t after which state change occurs, are determined:

```
s = s0;                          % Variable s contains the current state
t = 0;                           % Variable t contains the current time
i = 0;                           % Variable i contains the index of the current output couple
while t < T do                   % Until the end of trace time T
    if s = 1 then
            ns = newstateFromStateMachine…
            dt = durationFromDistribution…
    end
    if s = 2 then
    …
    i = i + 1;
    s = ns;                      % Move to next state
    t = t + dt;                  % Increase time
    Trace(i,:) = [t, s];         % Store in the trace
end
```

State machines: analysis

The actual way in which `ns` and `dt` are determined, depends on the particular state machine we are trying to reproduce:

```
s = s0;                    % Variable s contains the current state
t = 0;                     % Variable t contains the current time
i = 0;                     % Variable i contains the current output couple
while t < T do             % Until the end of trace time T
    if s = 1 then
            ns = newstateFromStateMachine…
            dt = durationFromDistribution…
    end
    if s = 2 then
    …
    i = i + 1;
    s = ns;                        % Move to next state
    t = t + dt;                    % Increase time
    Trace(i,:) = [t, s];           % Store in the trace
end
```

State machines: analysis

Finally results are stored, and both time and state are updated to advance the simulation.

```
s = s0;                      % Variable s contains the current state
t = 0;                       % Variable t contains the current time
i = 0;                       % Variable i contains the current output couple
while t < T do               % Until the end of trace time T
     if s = 1 then
               ns = newstateFromStateMachine…
               dt = durationFromDistribution…
     end
     if s = 2 then
     …
     i = i + 1;
     s = ns;                          % Move to next state
     t = t + dt;                      % Increase time
     Trace(i,:) = [t, s];             % Store in the trace
end
```

State machines: analysis

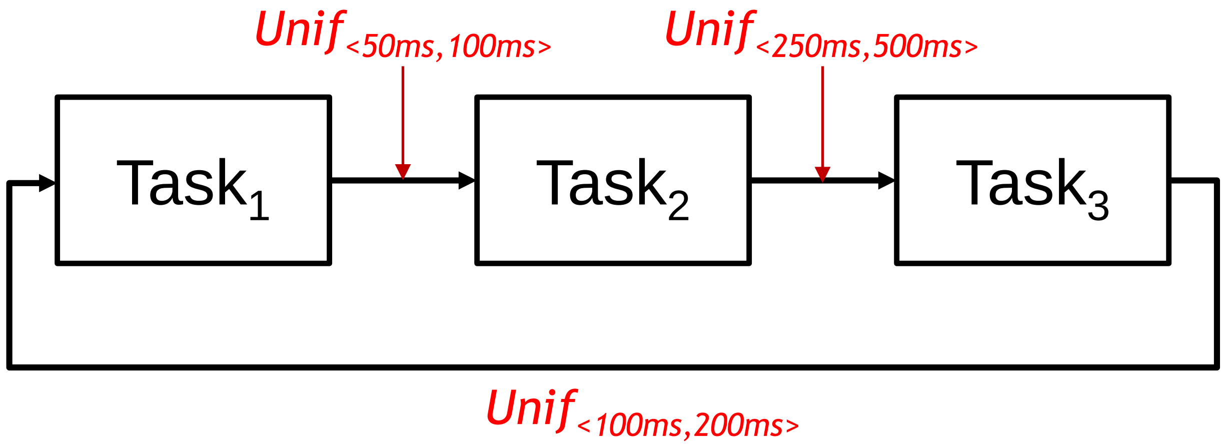
There are several alternatives possible to select the next state. Here we will present the main ones:

- Sequence
- Choice *(Next time)*
- Race *(Next time)*
- Concurrency *(Next time)*

Embedded system with 3 tasks *(Sequence)*

Consider and embedded system that shares the CPU running three tasks in a cyclic pattern. Each task has associated the time span where it can run: this is different for each task, depending on their function and priority.

In particular, we have measured that length of each task can be modeled with a uniform distribution, with different extents.



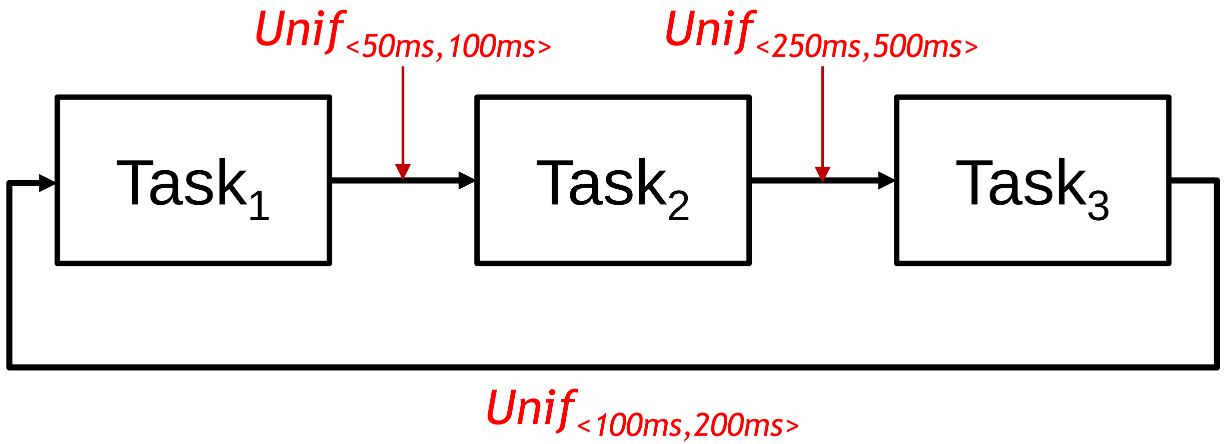
Embedded system with 3 tasks *(Sequence)*

We would like to compute:

- Probability that a given task is in execution
- Average time between two executions of the same task.

The task graph is already a state machine.

In this case, things are very simple since, in each state, there is a unique successor.



Embedded system with 3 tasks *(Sequence)*

```
…
    if s = 1 then
            ns = 2;
            dt = GenUnform(50,100);
    end
    if s = 2 then
            ns = 3;
            dt = GenUnform(250,500);
    end
    if s = 3 then
            ns = 1;
            dt = GenUnform(100,200);
    end
…
```

Embedded system with 3 tasks *(Sequence)*

```
CurrRountTime = 0; RoundTimeIndex = 1; RTT = [];
Ts1 = Ts2 = Ts3 = 0;
…
     if s = 1 then
               ns = 2;
               dt = GenUnform(50,100);
               Ts1 += dt;
               CurrRoundTime += dt;
     end
     if s = 2 then
               …
               Ts2 += dt;
               CurrRoundTime += dt;
     end
     if s = 3 then
               …
               Ts3 += dt;
               CurrRoundTime += dt;
               RRT(RoundTimeIndex) = CurrRoundTime;
               CurrRoundTime = 0;
               RoundTimeIndex ++;
     end
…
```

Performance counters are updated in the code that handles each state.

```
Ps1 = Ts1 / T;
Ps2 = Ts2 / T;
Ps3 = Ts3 / T;
AvgRT = mean(RTT);
```

Analysis of Motivating Example

We can create *State Machine based models* of the considered systems, generate the corresponding trace, and compute the required performance metrics.