Industrial Automation

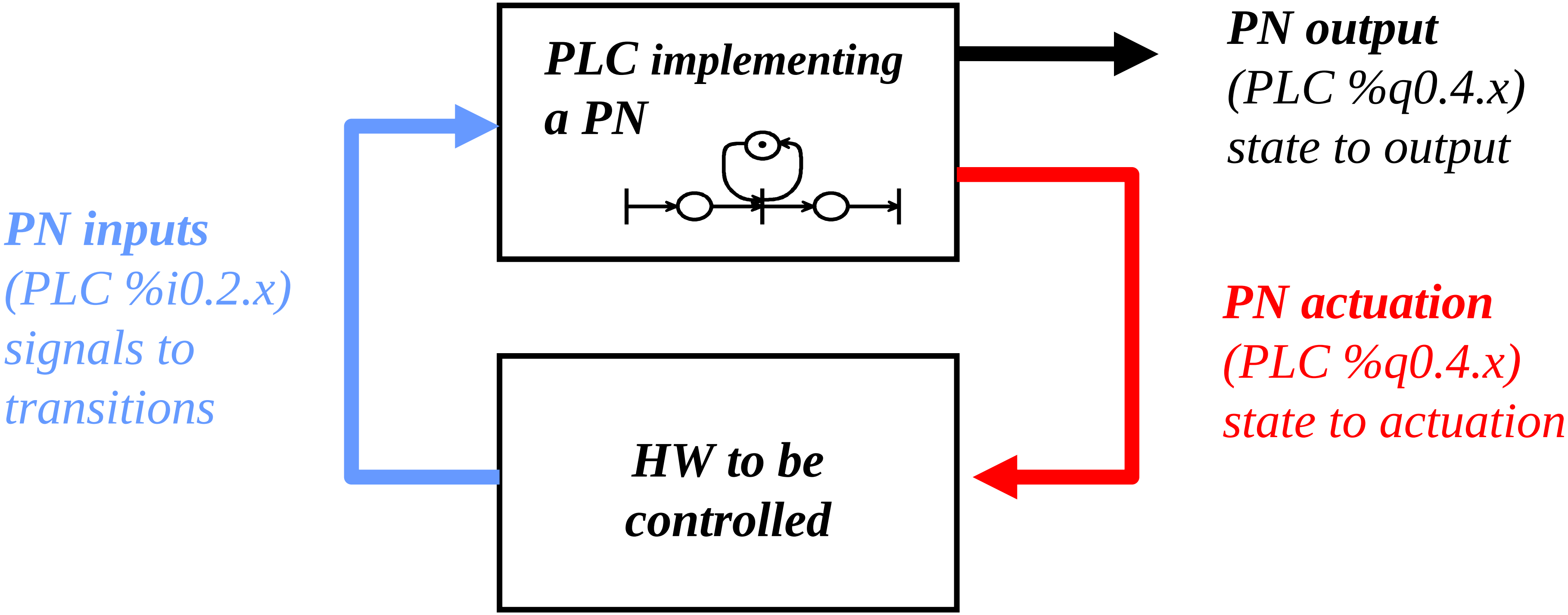
(Automação de Processos Industriais)

Analysis of Discrete Event Systems
Running a Petri net with I/O

http://users.isr.ist.utl.pt/~jag/courses/api1516/api1516.html

Slides 2012-2016 Prof. José Gaspar

Running a Petri net with HW inputs and outputs

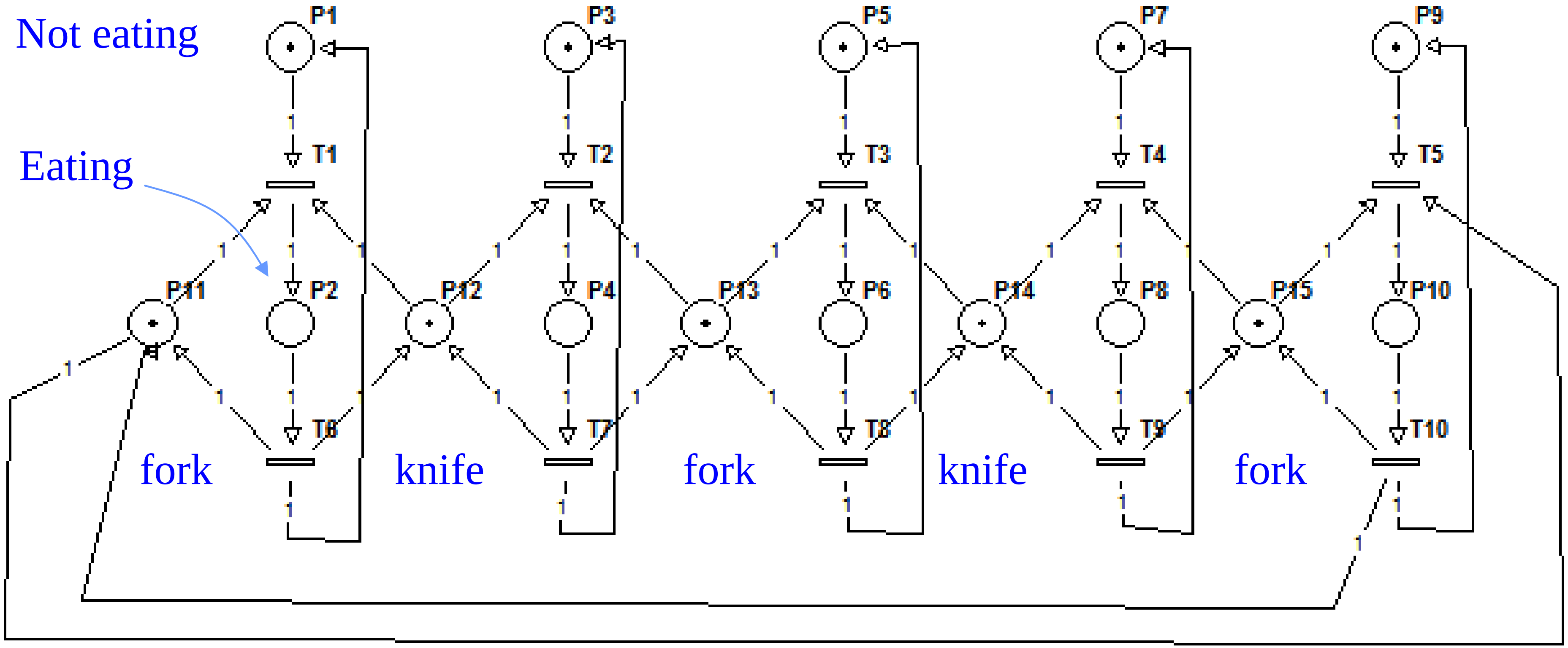


Summary of simulators: (a) simulation of the Petri net,
(b) simulation of the hardware to be controlled

Summary of functions: (1) state/places to actuation,
(2) signals to transitions,
(3) state/places to output

Running a Petri net with HW inputs and outputs

Example: Philosophers Dinner

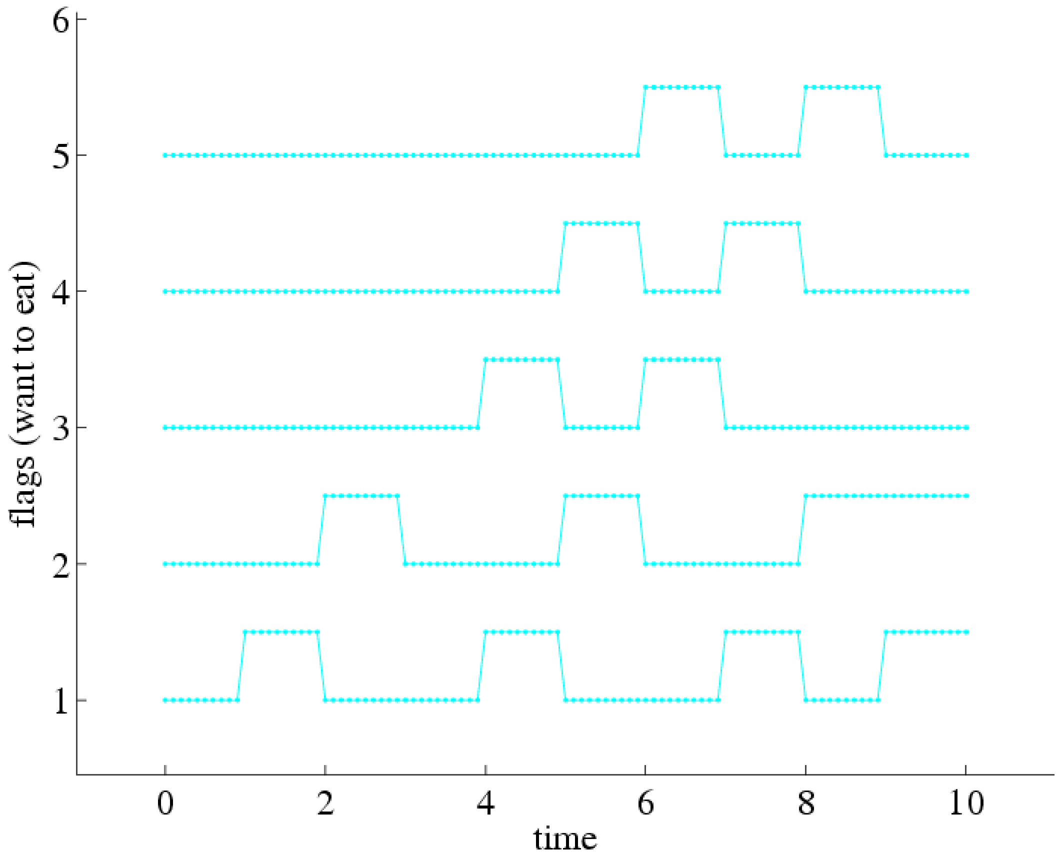


Philosopher1, Philosopher2, Philosopher3, Philosopher4, Philosopher5

Note: this PN has inputs "Philosopher i wants to eat" and has no outputs.

Running a Petri net with HW inputs and outputs

Example: Philosophers Dinner – input / events



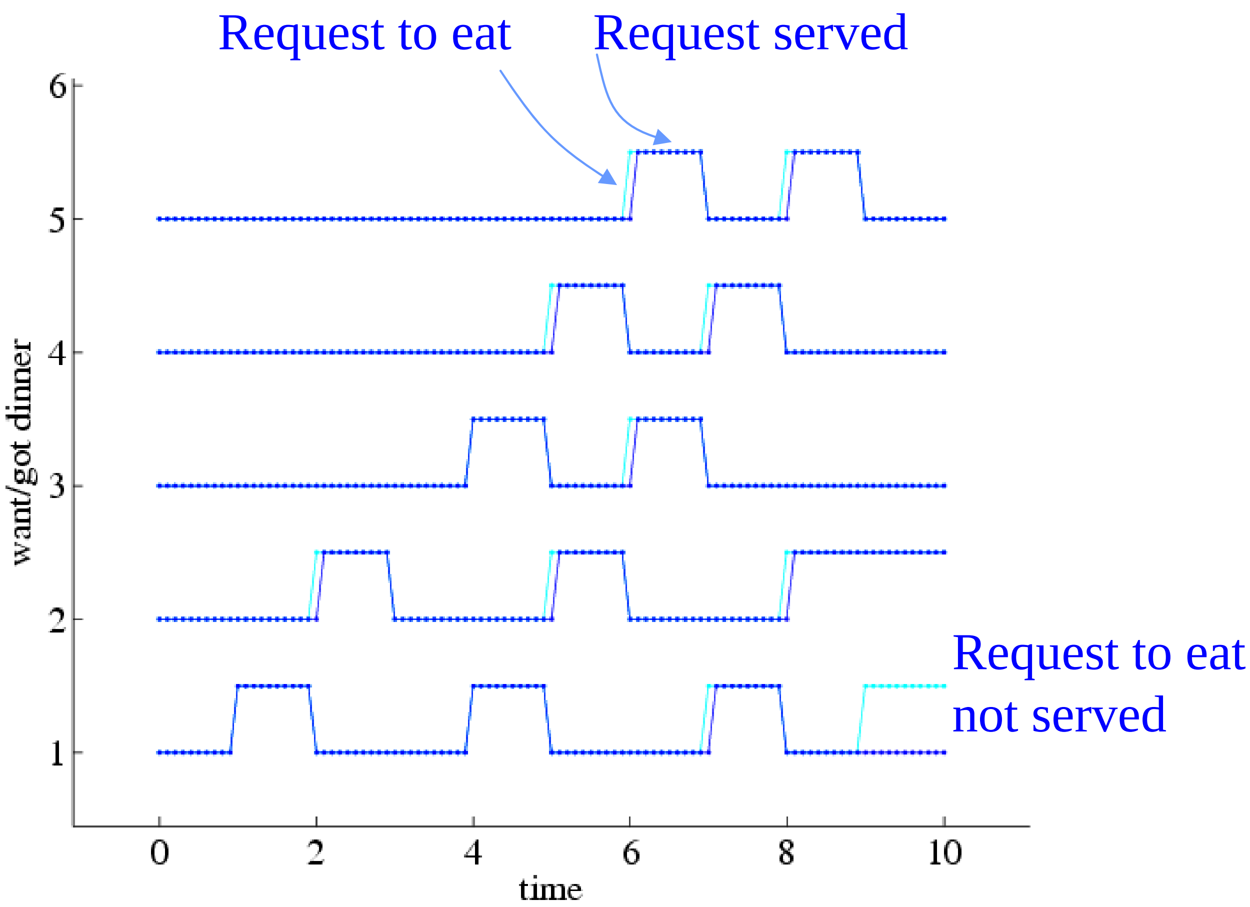

```
% first column = time in seconds
% next 5 columns = want to eat flags at time t
%

tu= [...
    0.0  want_to_eat( [] ) ; ...
    1.0  want_to_eat( 1  ) ; ...
    2.0  want_to_eat( 2  ) ; ...
    3.0  want_to_eat( []  ) ; ...
    4.0  want_to_eat( [1 3] ) ; ...
    5.0  want_to_eat( [2 4] ) ; ...
    6.0  want_to_eat( [3 5] ) ; ...
    7.0  want_to_eat( [4 1] ) ; ...
    8.0  want_to_eat( [5 2] ) ; ...
    9.0  want_to_eat( [1 2] ) ; ...
    ];


function y= want_to_eat(kid)
y= zeros(1,5);
for i=1:length(kid)
    y(kid(i))= 1;
end
```

Running a Petri net with HW inputs and outputs

Example: Philosophers Dinner – simulation



Note: See complete demo in the webpage of the course.

Note2: Modern operating systems must work better than failing early like in this PN simulation. E.g. two programs requiring simultaneously much CPU and memory, the O.S. has managers that hold the resources (CPU, memory, etc), queue the requests and in most cases even preempt the resources (CPU).

Running a generic Petri net

```
function [tSav, MPSav, youtSav]= PN_sim(Pre, Post, M0, ti_tf)
%
% Simulating a Petri net, using a SFC/Grafcet simulation methodology.
% See book "Automating Manufacturing Systems", by Hugh Jack, 2008
% (ch20. Sequential Function Charts)
%
% Petri net model:
%  M(k+1) = M(k) +(Post-Pre)*q(k)
%  Pre and Post are NxM matrices, meaning N places and M transitions

% 0. Start PN at state M0
%
MP=M0;
ti=ti_tf(1); tf=ti_tf(2); tSav= (ti:5e-3:tf)';
MPSav= zeros( length(tSav), length(MP) );
youtSav= zeros( length(tSav), length(PN_s2yout(MP)) );

for i= 1:length(tSav)

    % 1. Check transitions (update state)
    tm= tSav(i);
    qk= PN_tfire(MP, tm);
    qk2= filter_possible_firings(MP, Pre, qk(:));
    MP= MP +(Post-Pre)*qk2;

    % 2. Do place activities
    yout= PN_s2yout(MP);

    % Log all results
    MPSav(i,:)= MP';
    qkSav(i,:)= qk2';
    youtSav(i,:)= yout;

end
```

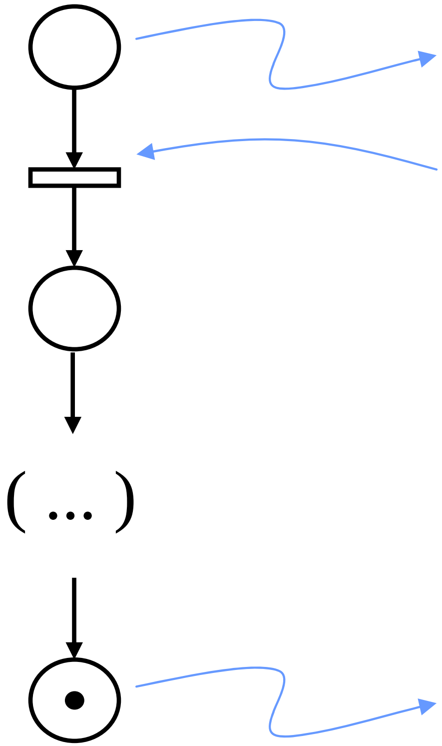
```
function qk2= filter_possible_firings(M0, Pre, qk)
% verify Pre*q <= M
% try to fire all qk entries

M= M0;
mask= zeros(size(qk));
for i=1:length(qk)
    % try accepting qk(i)
    mask(i)= 1;
    if any(Pre*(mask.*qk) > M)
        % exceeds available markings
        mask(i)= 0;
    end
end
qk2= mask.*qk;
```

Running a Petri net with HW inputs and outputs

Example: Keyboard reading

output = columns power,   *input* = lines read



1. state to actuation:
   power kb columns
2. signals to transitions:
   wait signal on kb lines
3. state to output:
   key X is pressed

See example in Matlab:

Summary of simulators

a) PN_sim.m
b) PN_device_kb_IO.m

Summary of functions

1) PN_s2act.m
2) PN_tfire.m
3) PN_s2yout.m

```
function lines= PN_device_kb_IO(act, t)

% Define 4x3-keyboard output line-values given actuation on the 3 columns
% and an (internal) time table of keys pressed
% Input:
%  act: 1x3 : column actuation values
%  t  : 1x1 : time
% Output:
%  lines: 1x4 : line outputs

global keys_pressed
if isempty(keys_pressed)
    % first column = time in seconds
    % next 12 columns = keys pressed at time t
    keys_pressed= [...
        0  mk_keys([]) ; 1  mk_keys(1)   ; ...
        2  mk_keys([]) ; 3  mk_keys(5)   ; ...
        4  mk_keys([]) ; 5  mk_keys(9)   ; ...
        6  mk_keys([]) ; 7  mk_keys([1 12]) ; ...
        8  mk_keys(12) ; 9  mk_keys([]) ; ...
        ];
end

% pressed keys yes/no
ind= find(t>=keys_pressed(:,1));
if isempty(ind)
    lines= [0 0 0 0]; % default lines output for t < 0
    return
end
keys_t= keys_pressed(ind(end), :);

% if actuated column and key pressed match, than activate line
lines= sum( repmat(act>0, 4,1) & reshape(keys_t(2:end), 3,4)', 2);
lines= (lines > 0)';
```

Keyboard simulator: generate line values given column values

```
function y= mk_keys(kid)
y= zeros(1,12);
for i=1:length(kid)
    y(kid(i))= 1;
end
```

Prototypes of the interfacing functions

```
function act= PN_s2act(MP)

% Create 4x3-keyboard column actuation
%
% MP:   1xN : marked places (integer values >= 0)
% act: 1x3 : column actuation values (0 or 1 per entry)
```

```
function qk= PN_tfire(MP, t)

% Possible-to-fire transitions given PN state (MP) and the time t
%
% MP: 1xN : marked places (integer values >= 0)
% t : 1x1 : time
% qk: 1xM : possible firing vector (to be filtered later with enabled
%            transitions)
```

```
function yout= PN_s2yout(MP)

% Show the detected/undetected key(s) given the Petri state
%
% MP: 1xN : marked places (integer values >= 0)
```