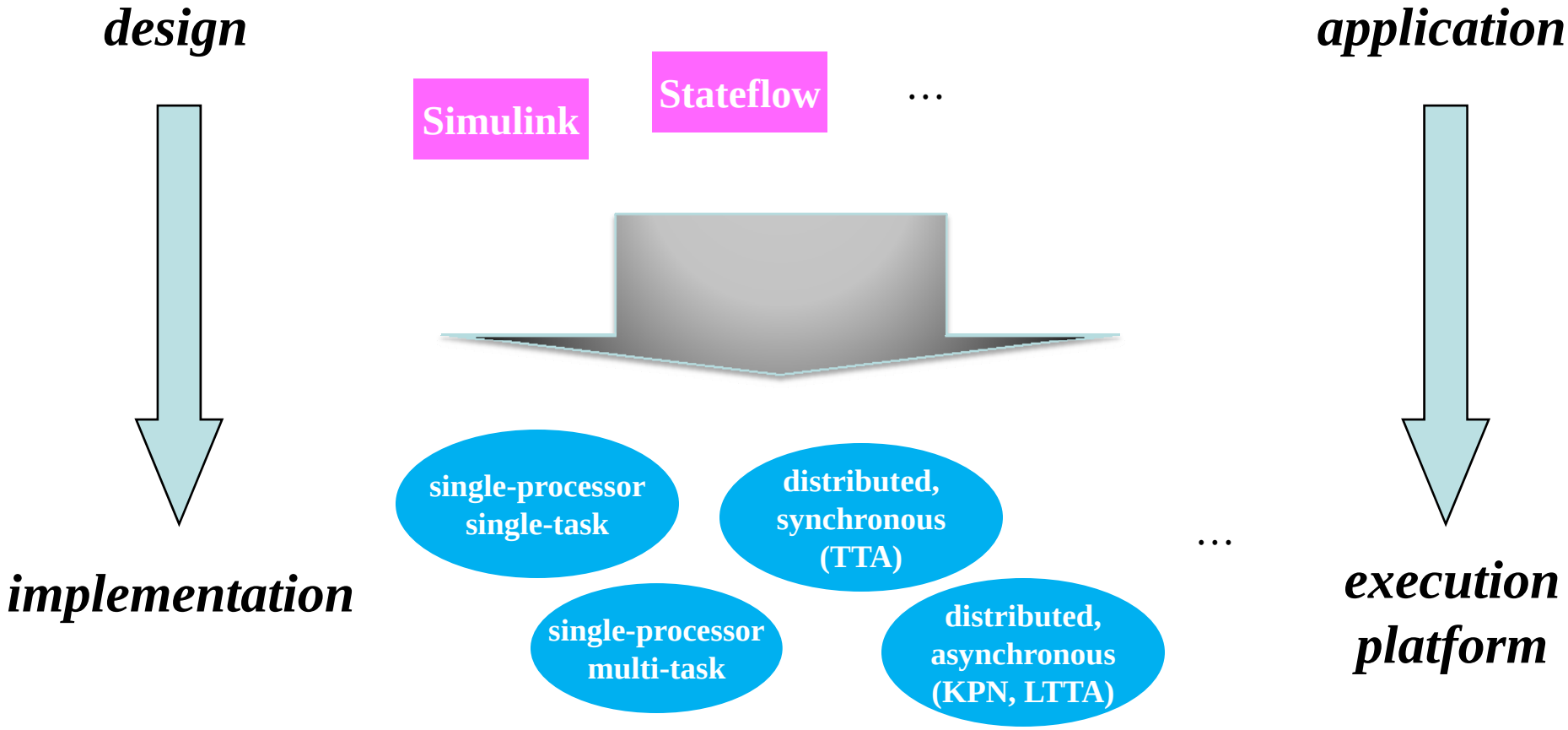
From synchronous models to distributed, asynchronous architectures

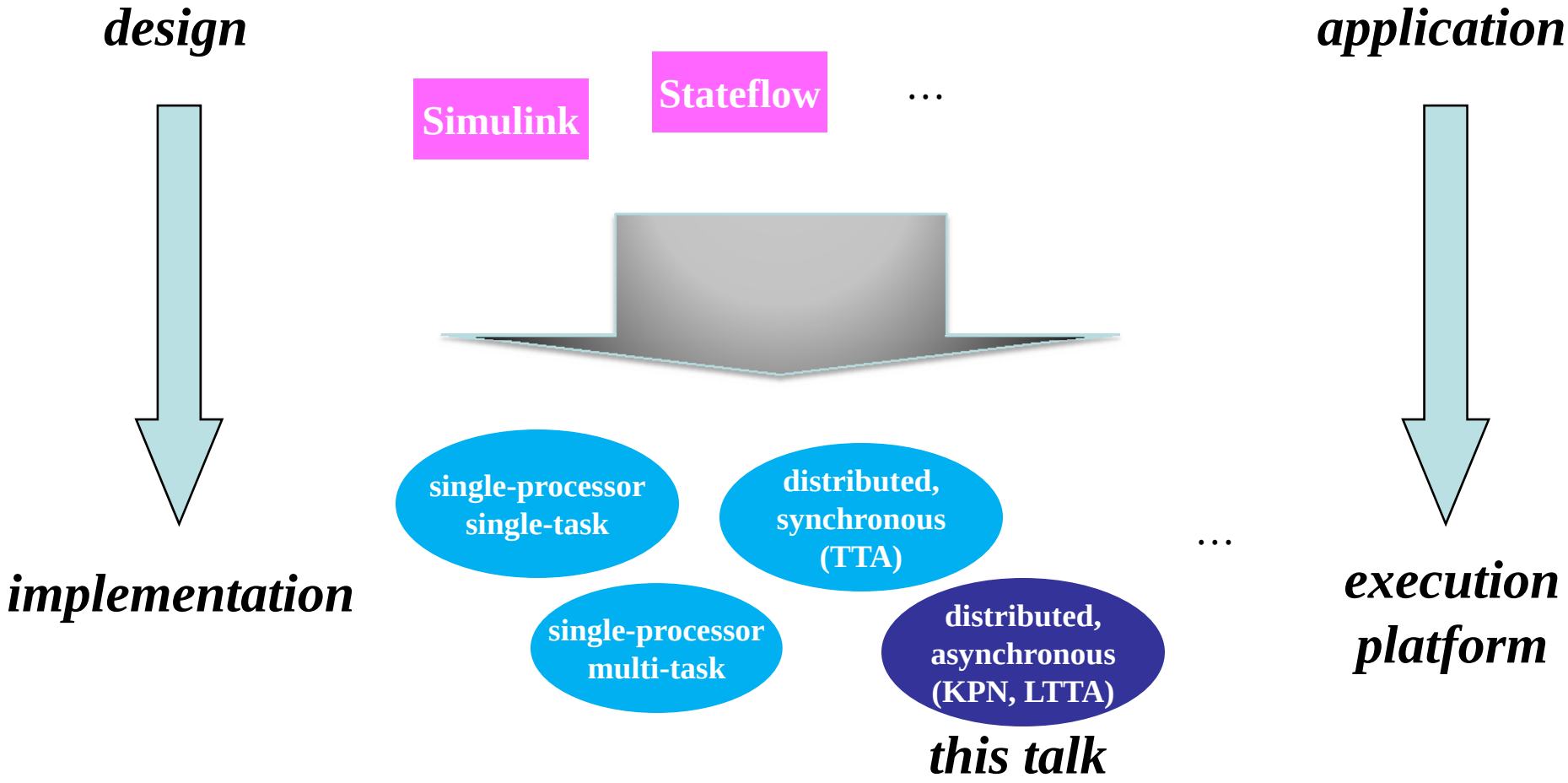
Stavros Tripakis

Joint work with
Claudio Pinello, Cadence
Alberto Sangiovanni-Vincentelli, UC Berkeley
Albert Benveniste, IRISA (France)
Paul Caspi, VERIMAG (France)
Marco di Natale, SSSA (Italy)

Model-based design:
semantics-preserving implementation of "high-level" models



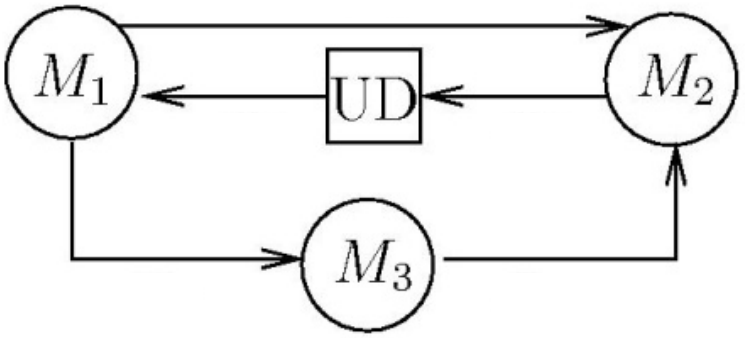
Model-based design:
semantics-preserving implementation of "high-level" models



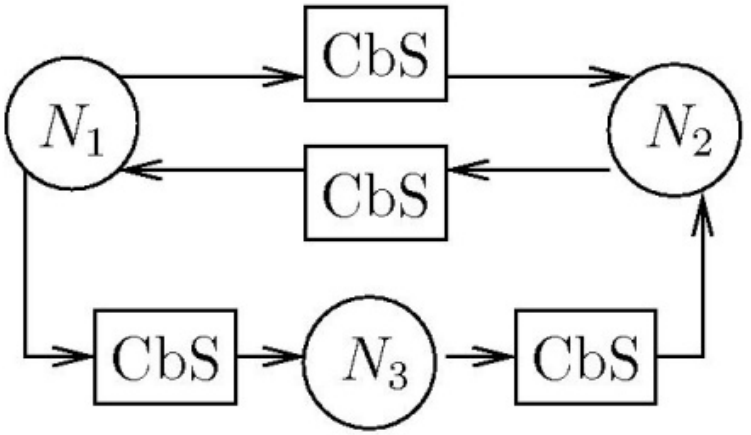
[IEEE Trans. Computers, Jan'08]

Goal: from Sync to LTTA

From a synchronous block diagram…

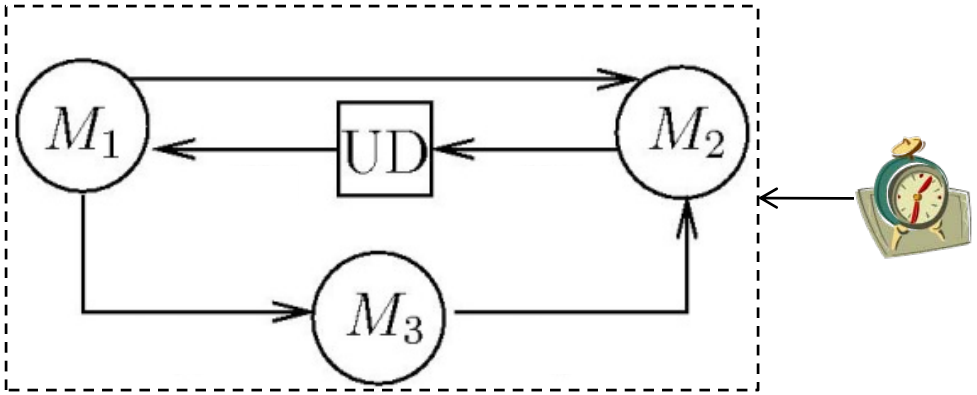


… to an implementation on LTTA

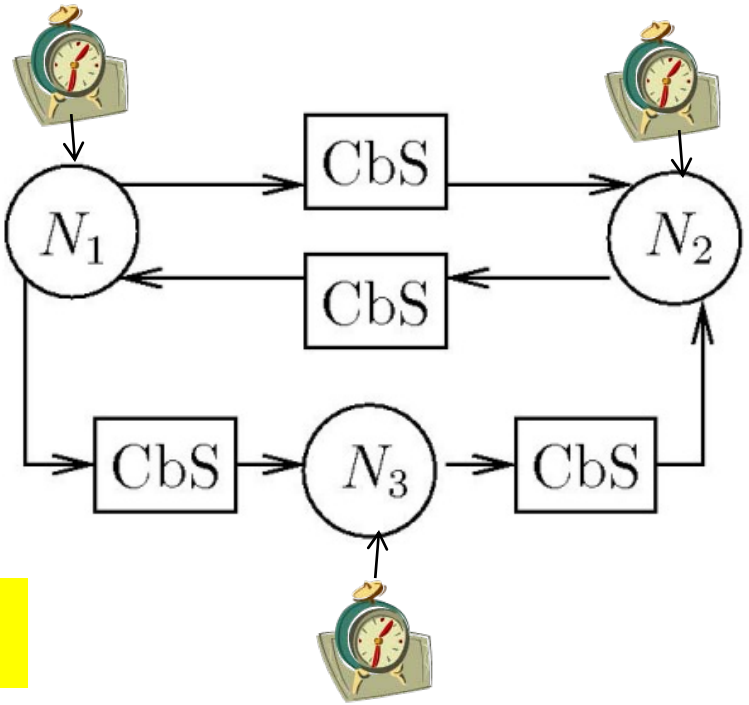


Goal: from Sync to LTTA

From a synchronous block diagram…

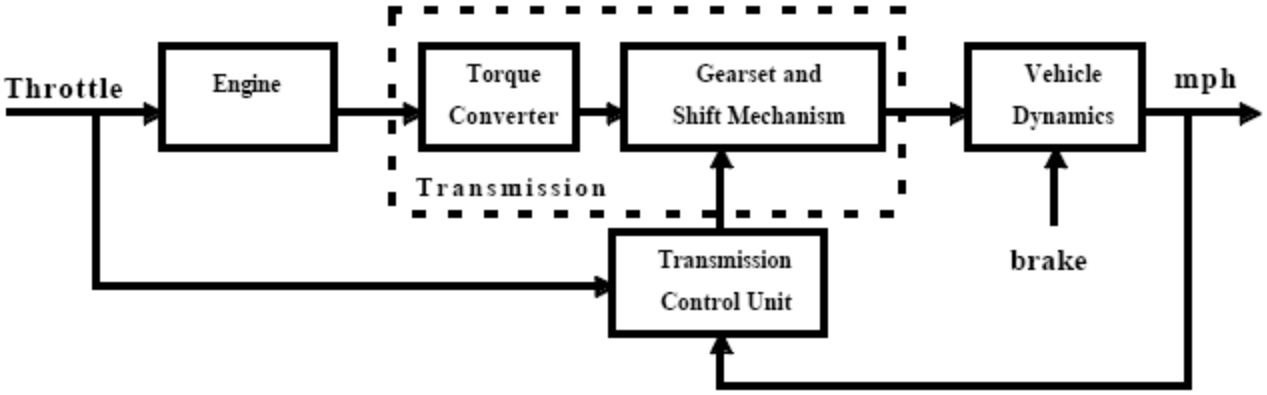


… to an implementation on LTTA: distributed, asynchronous

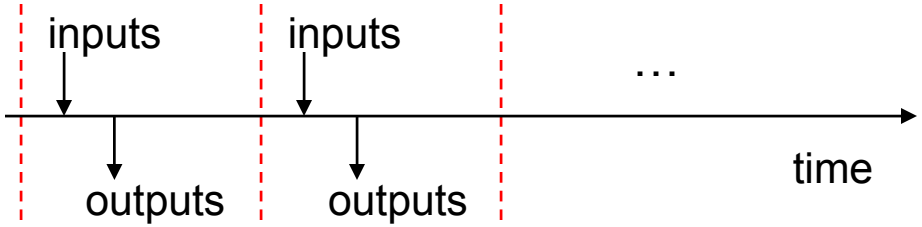


Preserve the semantics!

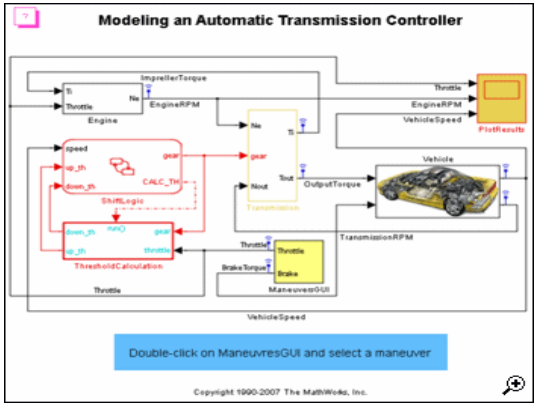
Synchronous block diagrams



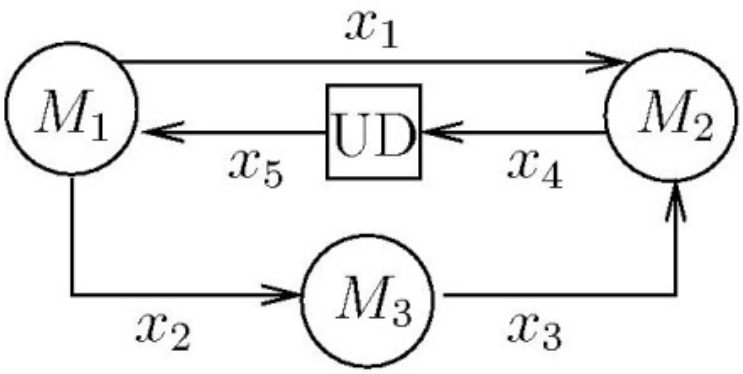
- Widely used in **embedded systems**
- Fundamental model behind (discrete-time) **Simulink**, SCADE, synchronous languages (Lustre, Esterel, …)
- **Synchronous**, deterministic semantics:



Copyright The Mathworks



Example



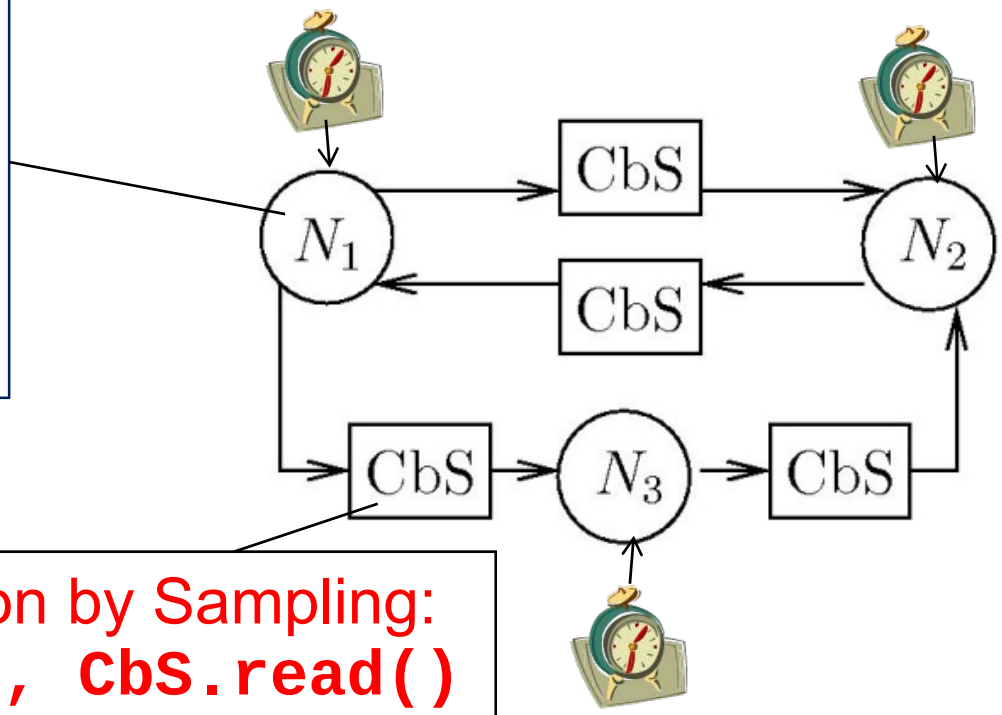
Static firing order in each synchronous cycle:
M1, M3, M2

LTTA

the Loosely Time-Triggered Architecture

Process structure:

```
initialize;
while (true) {
    await trigger;
    process body;
}
```



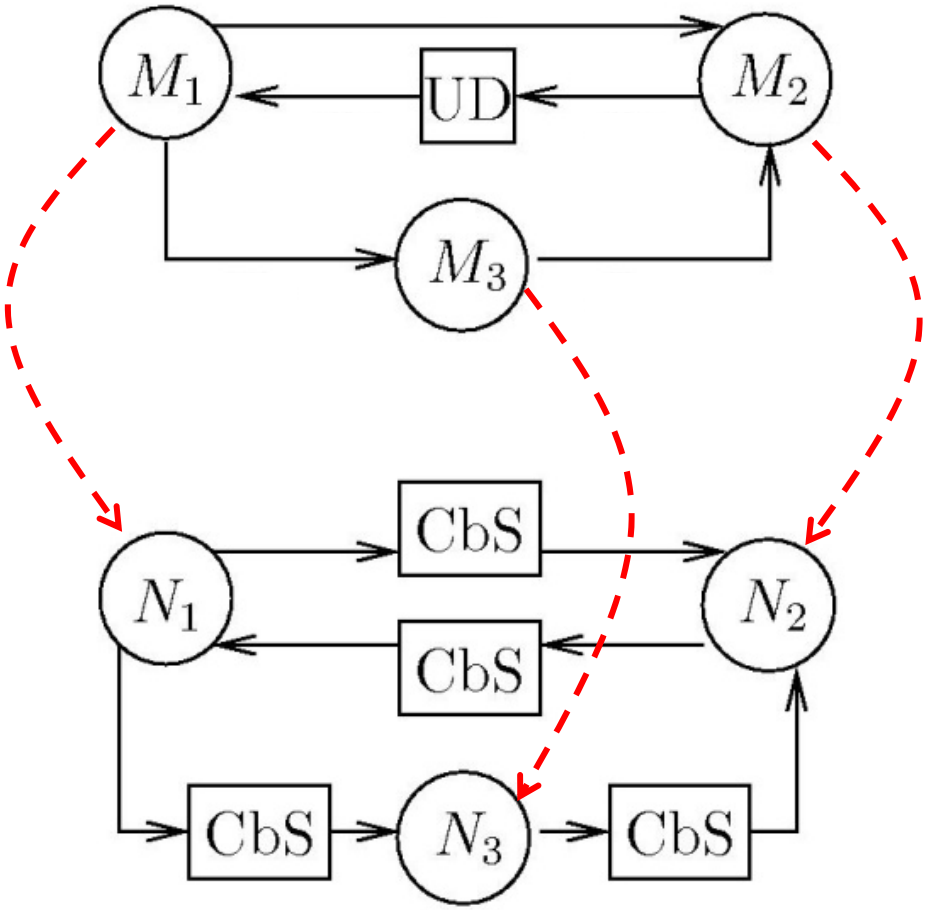
Communication by Sampling:
`CbS.write(), CbS.read()`

Assumptions: (1) atomic actions, (2) lossless network

More assumptions

From Sync…

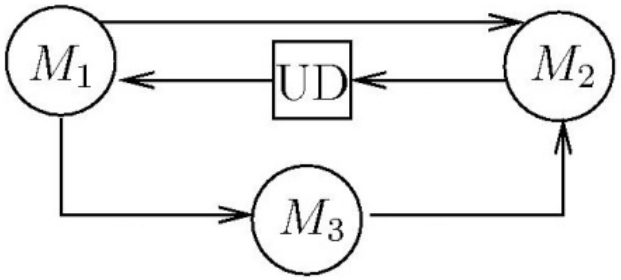
… to LTTA



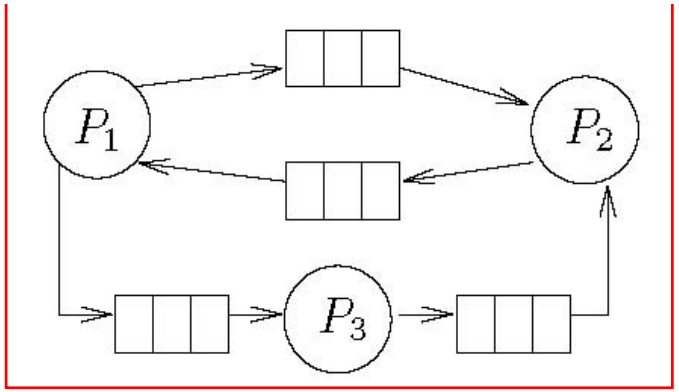
Assignment of blocks to nodes is 1-1

Intermediate layer: FFP

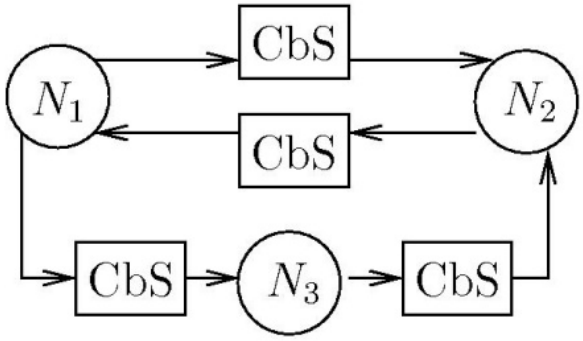
Sync



FFP : similar to a Kahn process network but FIFOs are finite

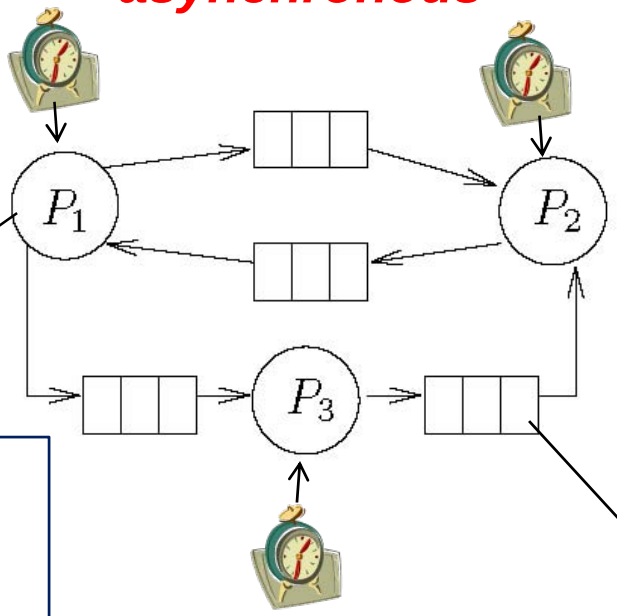


LTTA



FFP:
the Finite FIFO Platform

asynchronous



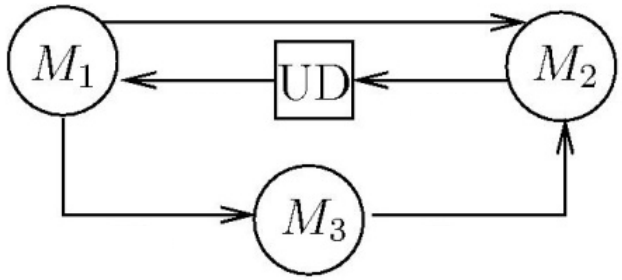
Process structure:

```
initialize;
while (true) {
    await trigger;
    process body;
}
```

FIFO queue:
FIFO.isFull(),
FIFO.isEmpty(),
FIFO.put(),
FIFO.get()

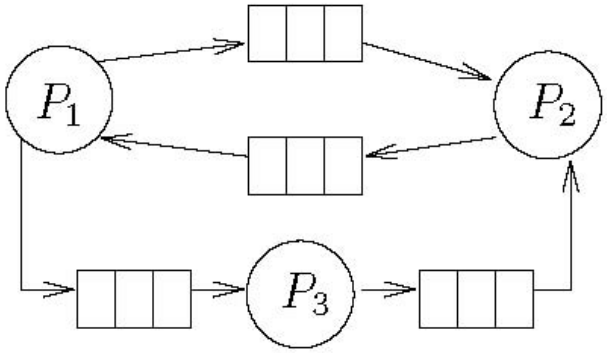
Intermediate layer: FFP

Sync



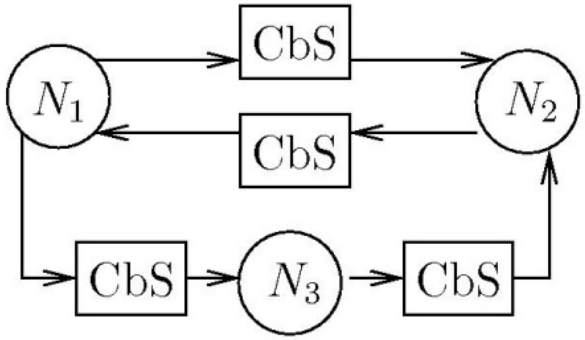
Problem 1:
from Sync to FFP

FFP



Problem 2:
from FFP to LTTA

LTTA



How to go from FFP to LTTA ?

- Suffices to implement FIFO queues on top of CbS buffers
  - Every FIFO of size k can be implemented with k+1 CbS buffers.
  - The +1 is for “back-pressure” (acknowledgments)

```
get() returns (msg) {
  ismsgNew := dataCbS[(rc mod k)].isNew();
  if (ismsgNew) {
    msg := dataCbS[(rc mod k)].read();
    rc := rc+1;
    backCbS.write(rc);
    oldmsg := msg;
  } else
    msg := oldmsg;
  return (msg);
}
```

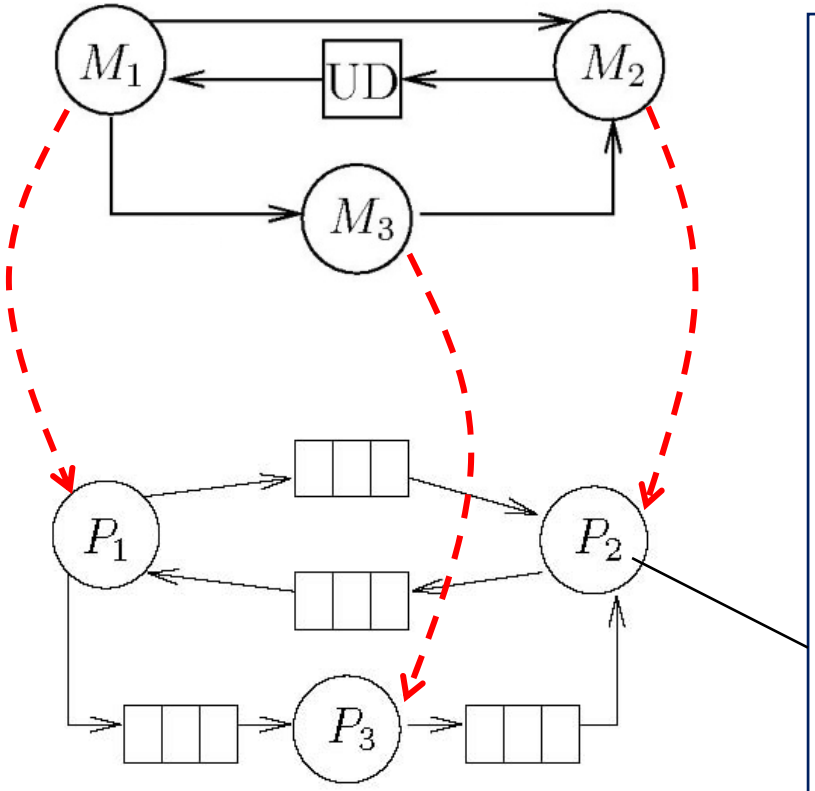
```
put(msg) {
  dataCbS[(wc mod k)].write(msg);
  wc := wc+1;
}
```

```
isFull() {
  lrc := backCbS.read();
  return (wc-lrc >= k);
}
```

```
isEmpty() {
  return not (dataCbS[(rc mod k)].isNew());
}
```

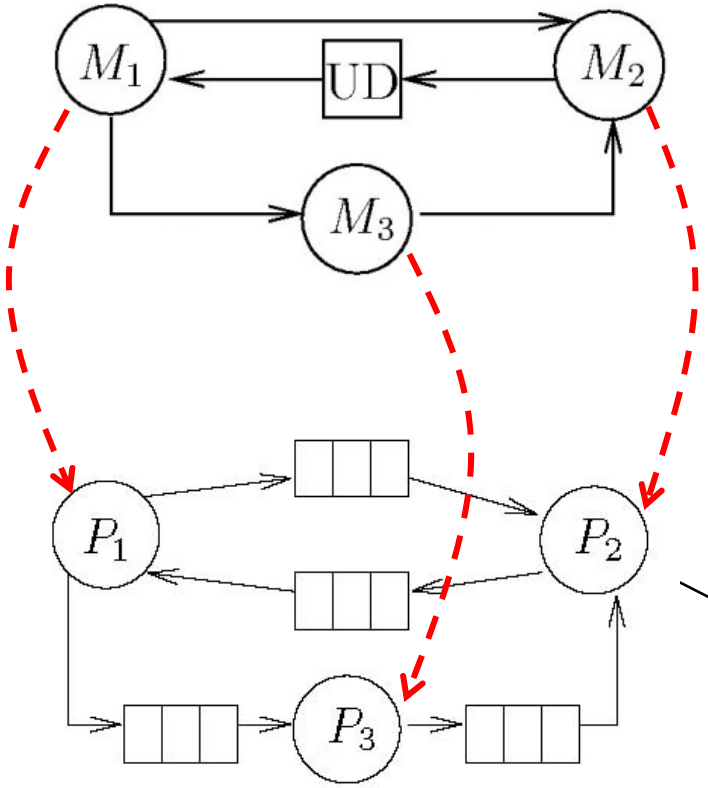
Details in paper

How to go from Sync to FFP ?



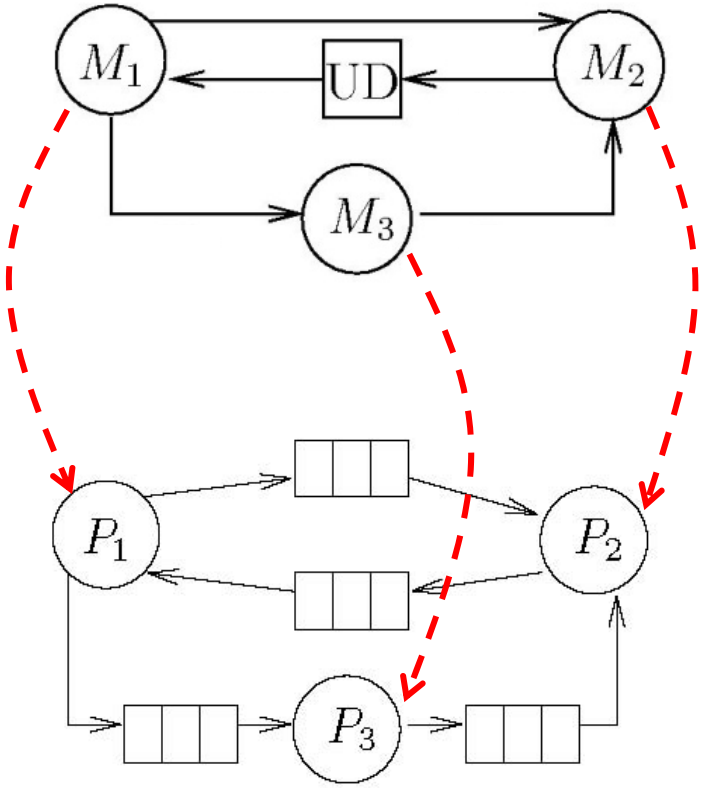

```
initialize state;
initialize output queues;
while (true) {
 await trigger;
 if (exists empty input queue
       OR full output queue) {
   /* skip */
 } else {
   /* fire */
   read from input queues;
   compute;
   write to output queues;
   update state;
 }
}
```

How to go from Sync to FFP ?



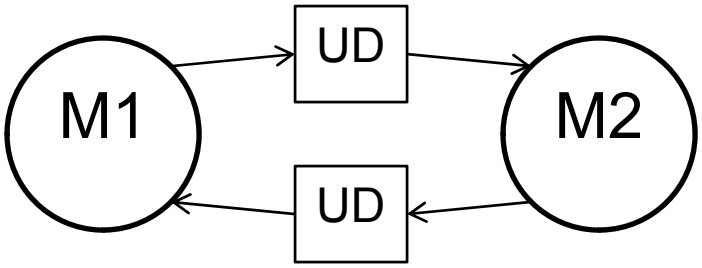

```
initialize state;
initialize output queues;
while (true) {
 await trigger;
 if (  inQ1.isEmpty() OR
       inQ2.isEmpty() OR
       outQ1.isFull() OR … ) {
   /* skip */
 } else {
   /* fire */
   read from input queues;
   compute;
   write to output queues;
   update state;
 }
}
```

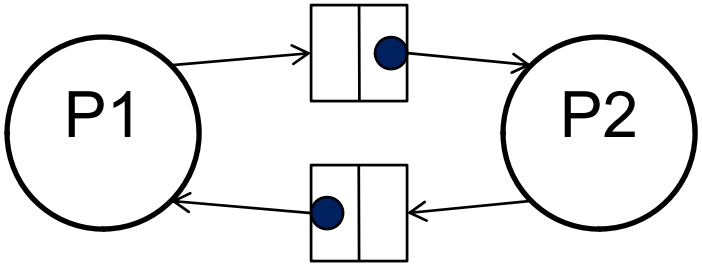
Semantical preservation



- **Theorem**: semantics is preserved provided queues have sufficient sizes (otherwise may deadlock)
  - Sufficient: 1-place FIFOs for non-unit-delay links, 2-place for UDs
  - Tight bound: at least m+1 places for every loop, where m is number of UDs in that loop
- Semantical preservation = **stream** preservation
  - The (infinite) sequence of values observed at a given link of Sync is equal to the sequence observed at the corresponding link in FFP

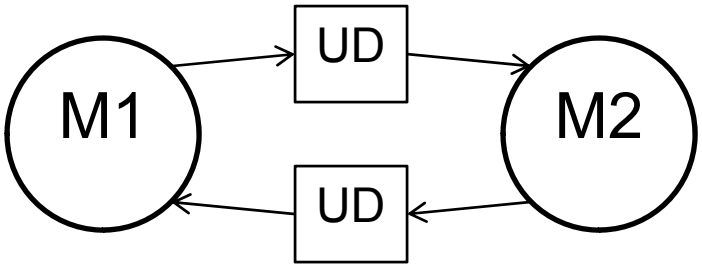
Why 2 place queues for unit-delays?

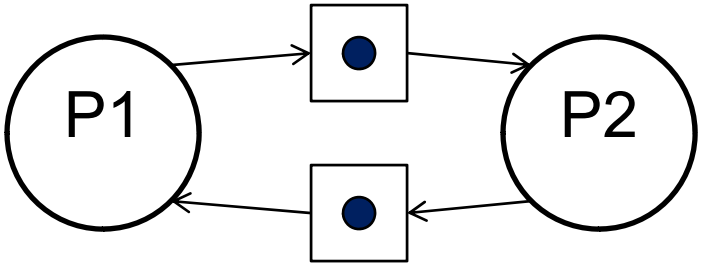




What would happen if both queues had size =1 ?

Why 2 place queues for unit-delays?





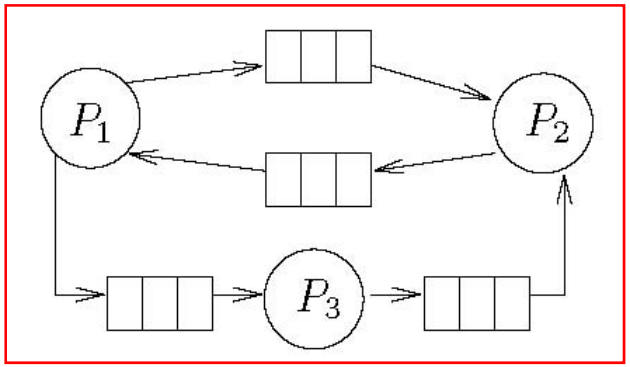
Deadlock !

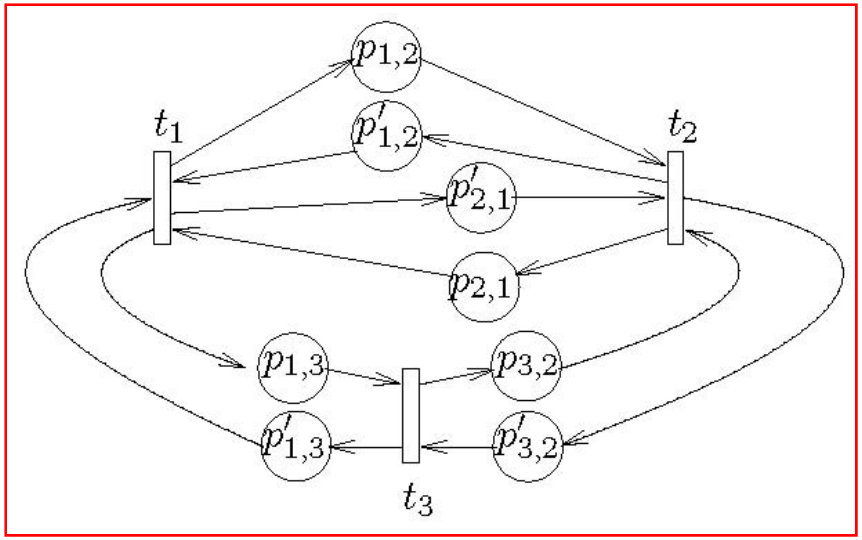
Proving preservation

- Used old theories [1970s]
- Marked graphs [Commoner et al ‘71]
  - Used to show that FFP is **live**: every process can **fire** infinitely often (does not deadlock)
  - Therefore all streams are infinite
- Kahn Process Networks [Kahn’74]
  - Every KPN is **determinate**: streams do not depend on process interleaving
  - One possible interleaving is the static one of the original synchronous diagram: this obviously yields the same streams as in the synchronous semantics
  - Thus every other interleaving will also yield equal streams

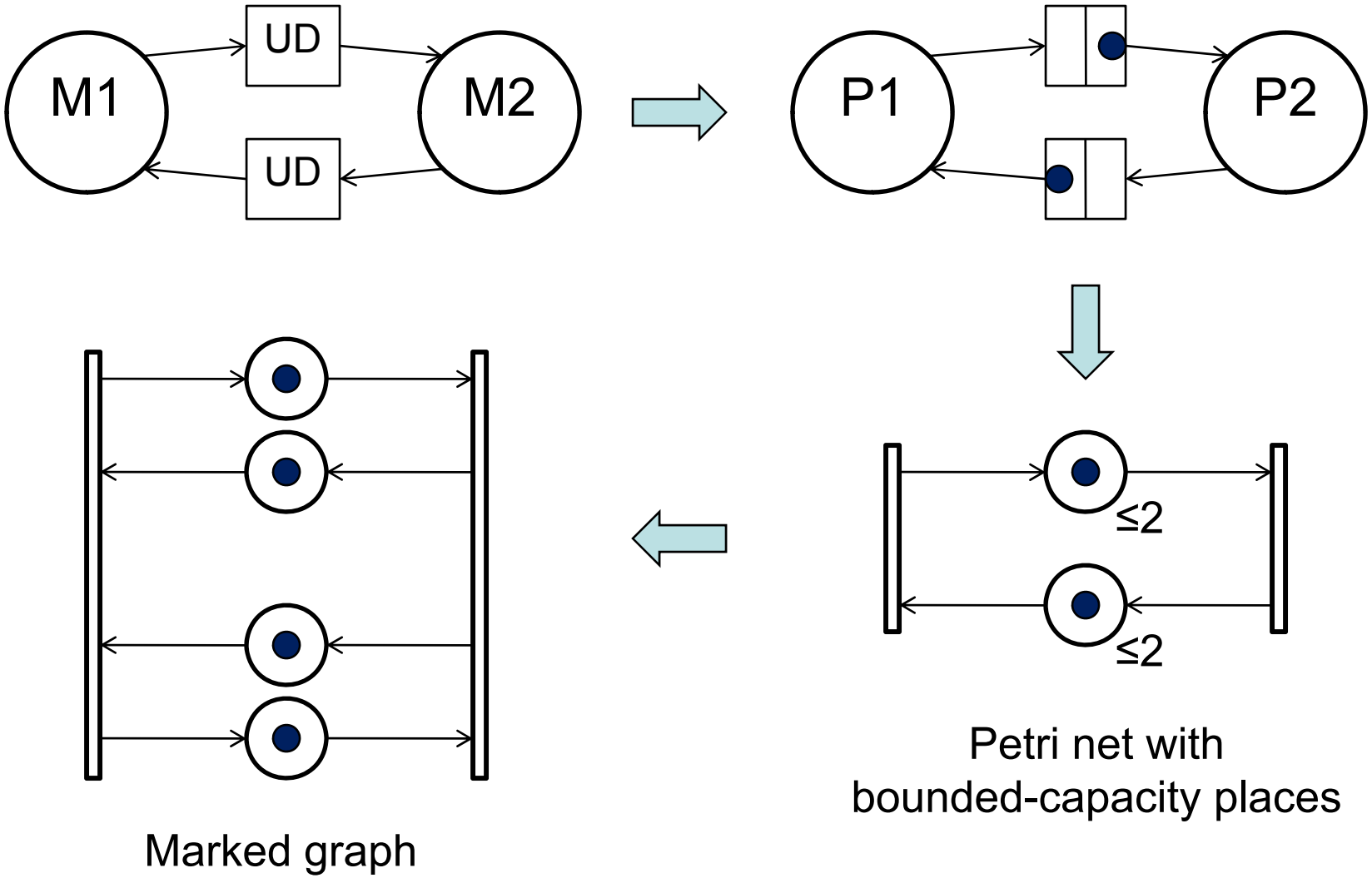
Proving liveness

- View FFP as a marked graph
- Marked graphs:
  - Subclass of Petri Nets
    - Every place has a unique input and a unique output transition
- Theorem [Commoner et al]:
  - A marked graph is live iff every loop has positive token count
  - (token count invariant in a loop)

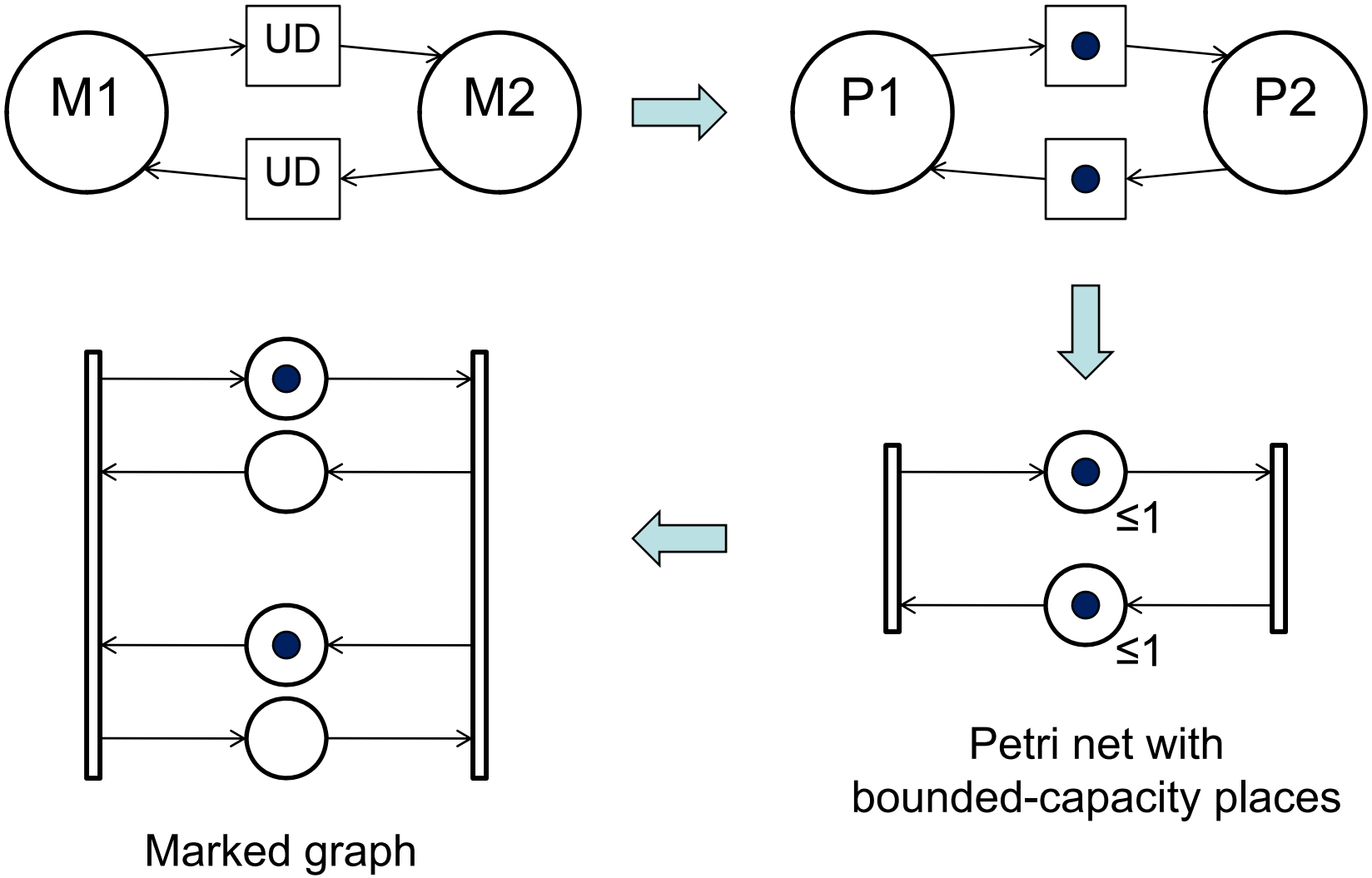




Example 1


M1
UD
UD
M2
P1
P2
≤2
≤2
Petri net with
bounded-capacity places
Marked graph

Example 2: deadlock



Performance analysis

- **Throughput**:
  - How often processes fire (vs. skip) – i.e., how often outputs are produced
- **Latency**:
  - What is the life-span of a token from production to consumption
- Metrics: **real-time** (RT) vs. **logical-time** (LT)
  - Real-time: metrics depend on real-time behavior of clocks (e.g., clock rates)
  - Logical-time: metrics depend only on topology, queue sizes and initial conditions !
- Results:
  - Algorithms to compute worst-case LT throughput/latency for arbitrary topologies
  - Theorems to compute them analytically (formulas) for special topologies
  - Theorems that relate RT metrics and LT metrics

Real-time and logical-time throughput: definitions

$$\lambda^{rt}(\mathcal{F}, P_i, c) = \lim_{t \to \infty} \frac{fireno_{\mathcal{F},c,P_i}(t)}{t}$$

$$\lambda^{lt}(\mathcal{F}, P_i, c, \chi) = \lim_{n \to \infty} \frac{fireno_{\mathcal{F},c,P_i}(\chi(n))}{n}$$

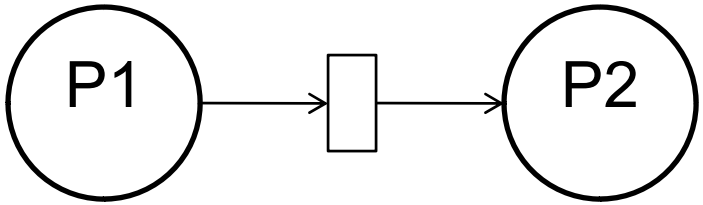
$$\lambda^{wclt}(\mathcal{F}, P_i, \chi) = \inf_{c \in C(\chi)} \lambda^{lt}(\mathcal{F}, P_i, c, \chi)$$

The “slow” triggering policy

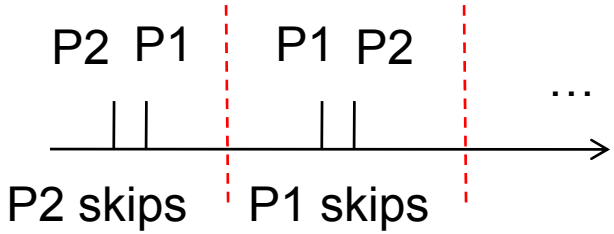
- At each synchronous cycle:
  - First trigger all disabled processes (they will skip)
  - Then trigger all enabled processes (they will fire)
  - Note:
    - Each queue has a unique reader and writer
    - Therefore firing one process cannot disable another
    - Thus the order in which the enabled processes are fired does not matter

- **Theorem**: worst-case logical-time throughput is achieved by real-time clocks that follow the slow triggering policy

Logical-time throughput: example



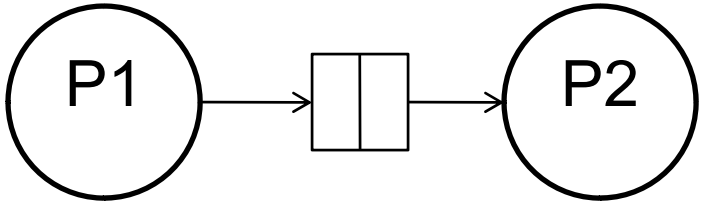
Worst-case scenario:

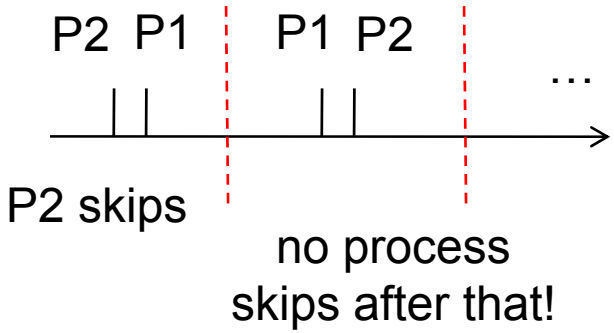


LT throughput:

½ times

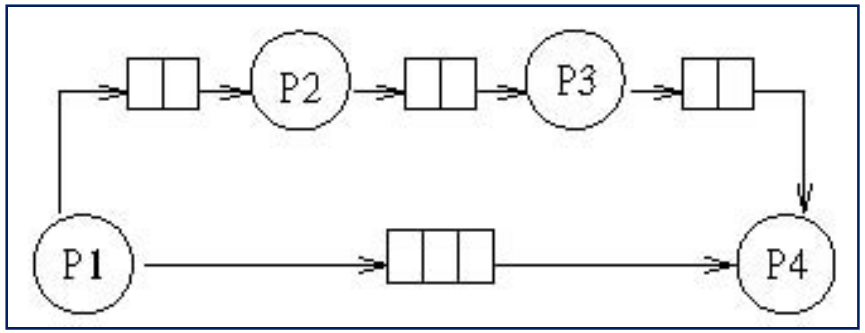
LT thput = 1/2



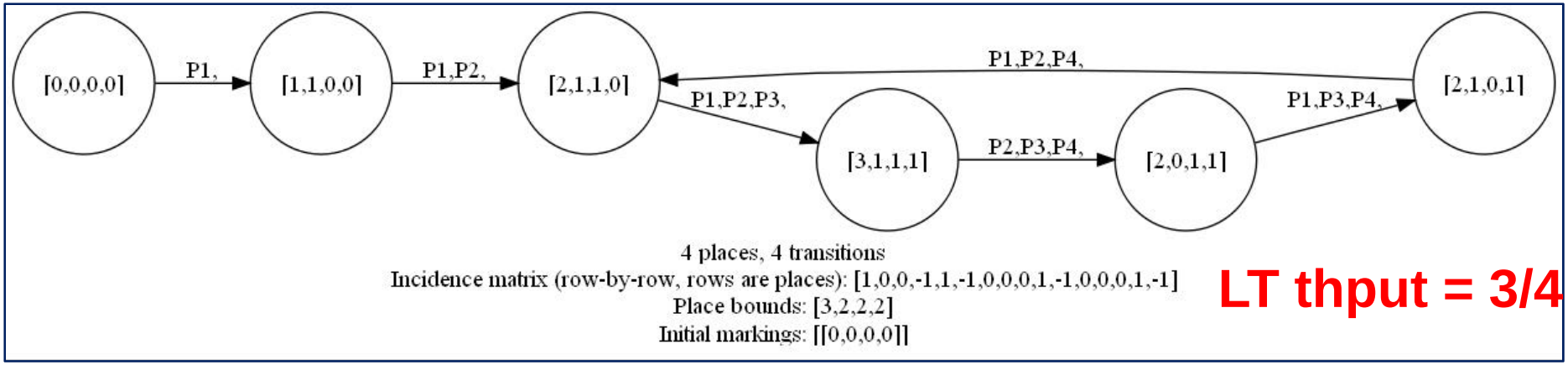


LT thput = 1

Computing the worst-case logical-time throughput



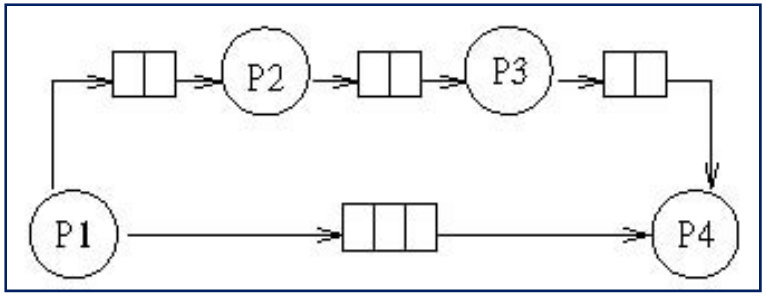
deterministic (slow) firing policy



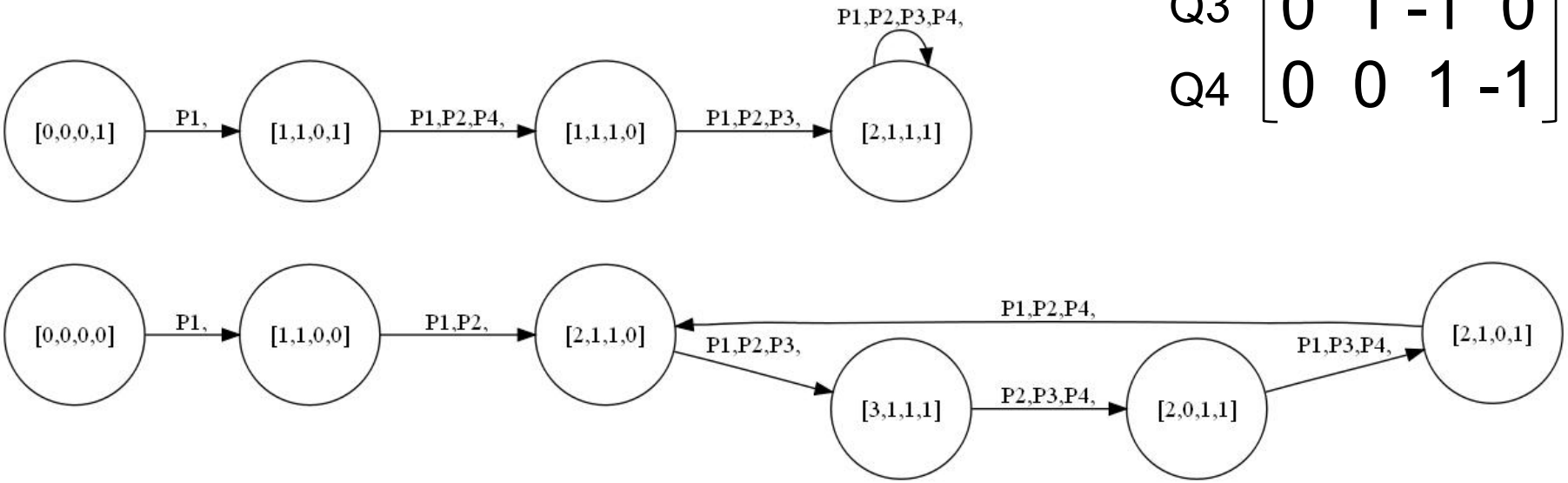
LT thput = 3/4

Reachable lasso of marked graph

Demo



	P1	P2	P3	P4
Q1	1	0	0	-1
Q2	1	-1	0	0
Q3	0	1	-1	0
Q4	0	0	1	-1



4 places, 4 transitions
Incidence matrix (row-by-row, rows are places): [1,0,0,-1,1,-1,0,0,0,1,-1,0,0,0,1,-1]
Place bounds: [3,2,2,2]
Initial markings: [[0,0,0,0],[0,0,0,1]]

Worst-case logical-time throughput: connected networks

Theorem 8 *Given a connected SFFP $\mathcal{F}$,*

$$\forall P_i, P_j \in \mathcal{F},\ \lambda^*(\mathcal{F}, P_i) = \lambda^*(\mathcal{F}, P_j).$$

i.e.: for connected networks, WCLT throughput is the same for all processes

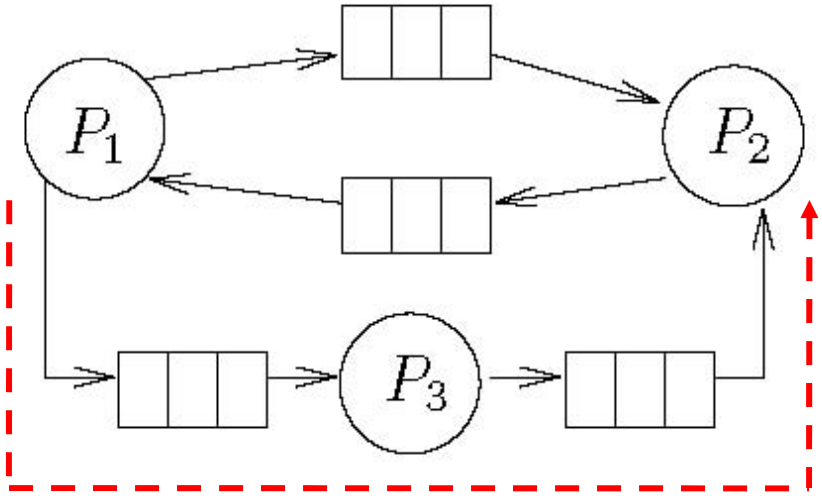
Intuition: total # tokens produced = total # tokens consumed

Relating real-time and worst-case logical-time throughput

Theorem 9: Let $\mathcal{F}$ be an SFFP. Let Δ be any positive real number. Let c be a vector of clocks such that $\forall i, \forall n, c_i(n+1) - c_i(n) \leq \Delta$. Then, for any process P_i of $\mathcal{F}$:

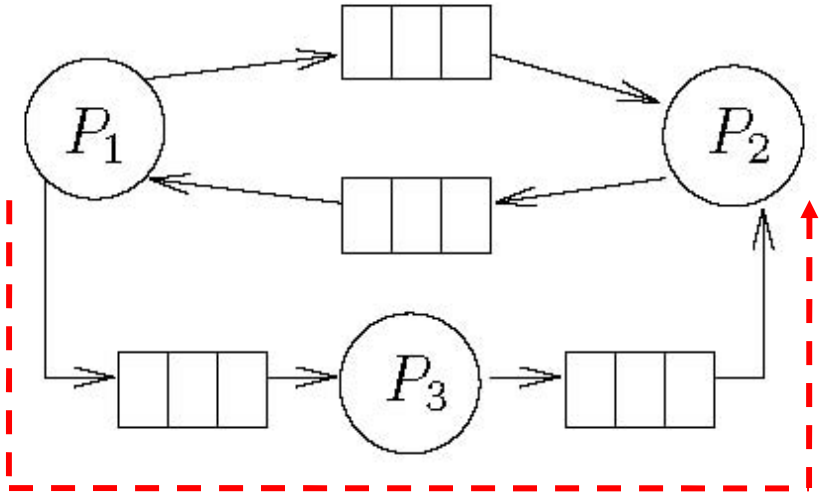
$$\lambda^{rt}(\mathcal{F}, P_i, c) \geq \frac{\lambda^*(\mathcal{F}, P_i)}{\Delta}$$

Latency: defined w.r.t. a path



$$\mu^{rt}(\mathcal{F}, \pi, c) = \sup_{z} travel_{\mathcal{F},c,\pi}(z)$$

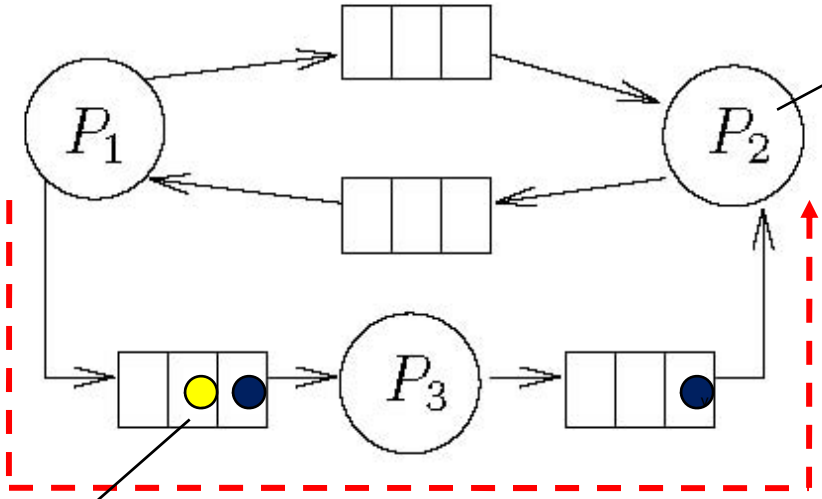
Latency: defined w.r.t. a path



$$\mu^{lt}(\mathcal{F}, \pi, c, \chi) = \sup_{z} travel^{\chi}_{\mathcal{F},c,\pi}(z)$$

$$\mu^{wclt}(\mathcal{F}, \pi, \chi) = \sup_{c \in \mathcal{C}(\chi)} \mu^{lt}(\mathcal{F}, \pi, c, \chi)$$

Worst-case logical-time latency: computation



3 **firings** of P2

How long until this token gets consumed?

Worst-case logical-time latency: computation

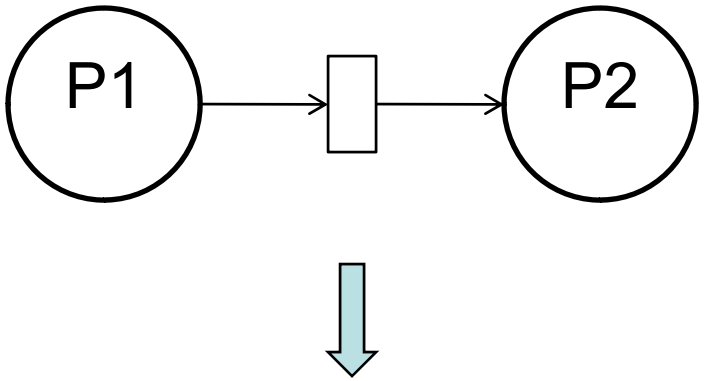
- Compute reachability graph

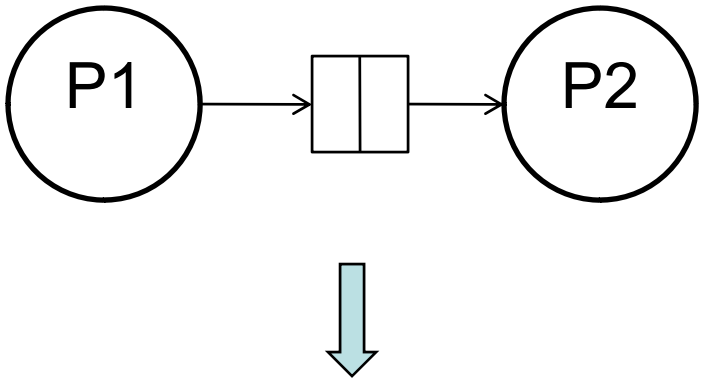
unfortunately not just a lasso …

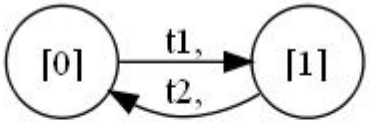
- For every reachable marking m s.t. the first queue in the path is non-empty:
  - Compute T(m) = sum of all tokens in the path
  - Compute L(m) = #steps it takes to fire the destination process T(m) times (w.r.t. the slow policy)

- WCLT latency = max L(m) over all such m

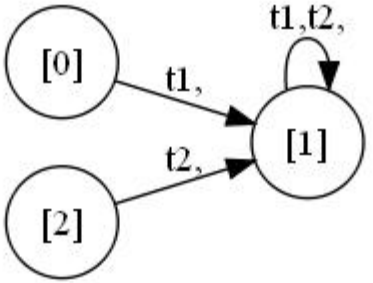
Demo







1 places, 2 transitions
Incidence matrix (row-by-row, rows are places): [1,-1]
Place bounds: [1]
Initial markings: [[0]]



1 places, 2 transitions
Incidence matrix (row-by-row, rows are places): [1,-1]
Place bounds: [2]
Initial markings: [[0]]

Relating real-time and worst-case logical-time latency

Theorem 17 *Let $\mathcal{F}$ be an SFFP. Let Δ be any positive real number. Let c be a vector of clocks such that $\forall i, \forall n, c_i(n+1) - c_i(n) \leq \Delta$. Then, for any path π of $\mathcal{F}$:*

$$\mu^{rt}(\mathcal{F}, \pi, c) < \Delta \cdot (\mu^*(\mathcal{F}, \pi) + 1)$$

Conclusions

- Semantics-preserving implementation of synchronous models on asynchronous execution platforms
- Layered approach: “platform-based design”
- Performance analysis
  - Logical time throughput and latency
  - Can be used for design-space exploration (e.g., queue sizing)

What next?

- Barely scratched the surface:
- More implementation options:
  - Lift 1-1 mapping assumption
  - Implement FFP on top of other platforms than LTTA
    - Bonus: semantical preservation of synchronous models!
  - …
- More performance analysis:
  - More efficient algorithms to compute LT throughput/latency
  - Done only for FFP, what about LTTA?
    - Lift negligible execution/communication delay assumptions
    - Lift lossless network assumption
  - What about average-case vs. worst-case?
  - What if other information on clock rates is available?
  - …
- Design-space exploration:
  - How to efficiently explore throughput-optimal queue sizes?
  - …

Thank you

Questions?

Reference: “Implementing Synchronous Models on Loosely Time Triggered Architectures”, IEEE Trans. Computers, 57(10), Oct 2008.
`http://www-verimag.imag.fr/~tripakis/papers/tc08-revised.pdf`