Real-time Systems

9 September 2004

Dr. Sergio Ruocco

ERTOS Researcher

What are real-time systems ?

Real-time systems are computer-based systems that control a process whose correctness depends on TIME.

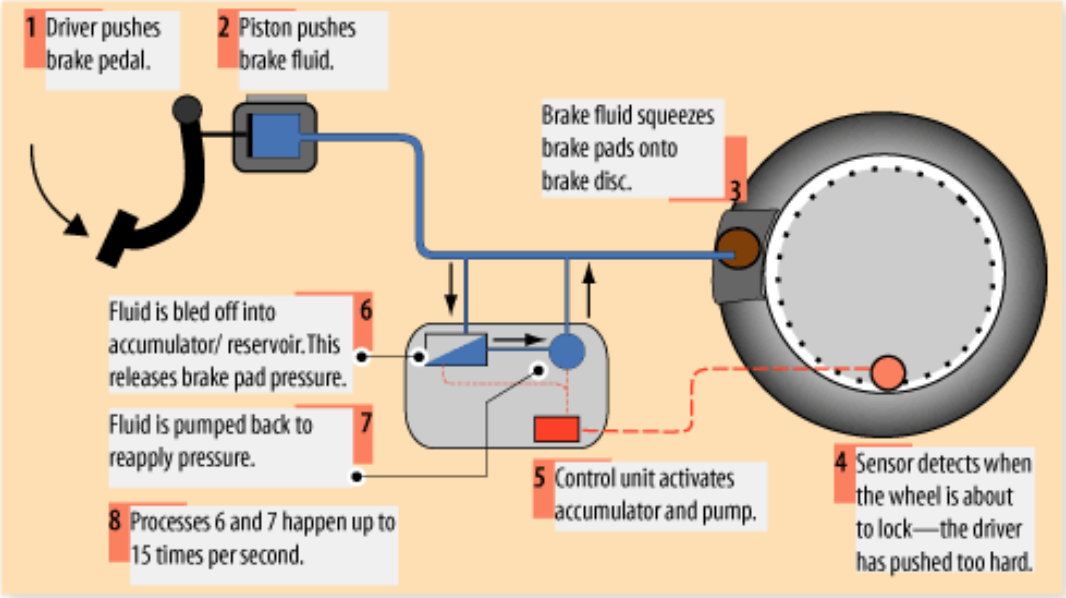
What are real-time systems ?

Real-time systems are computer-based systems that control a process whose correctness depends on TIME.

What are real-time systems ?

Real-time systems are computer-based systems that control a process whose correctness depends on TIME.

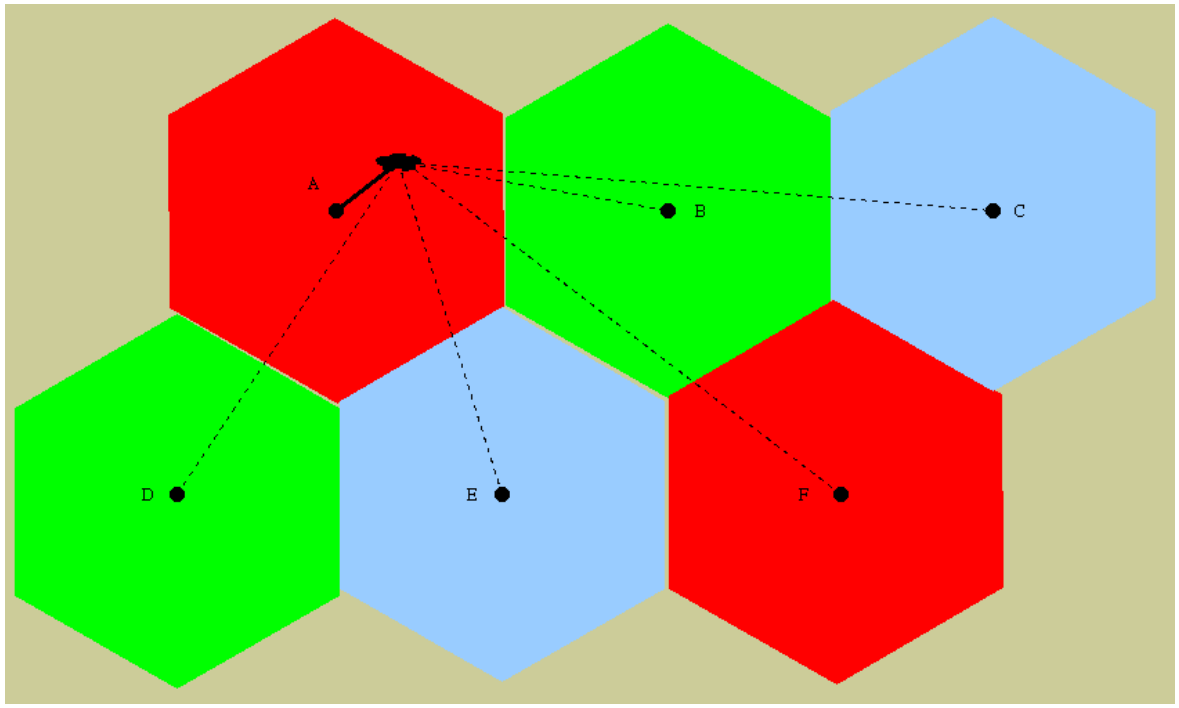
- Anti-lock Brake System



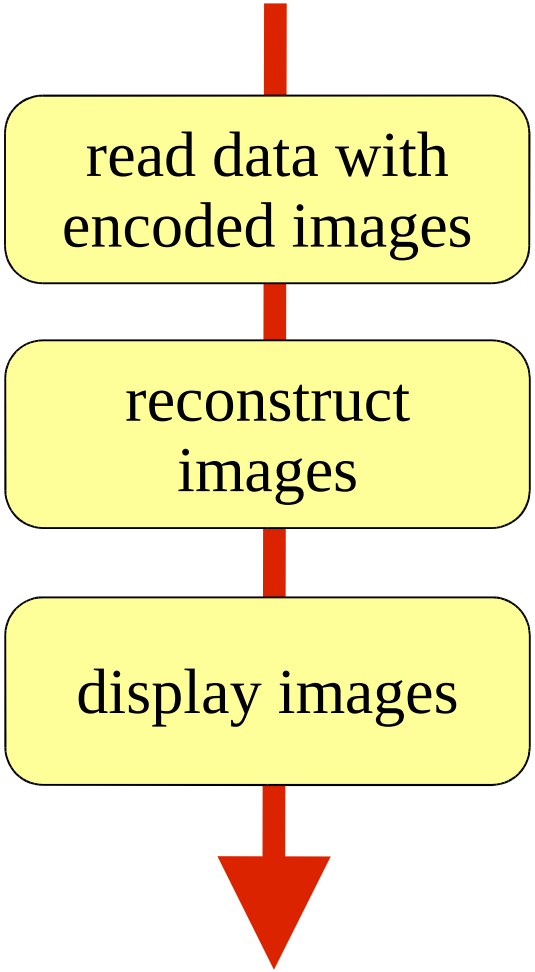
What are real-time systems ?

Real-time systems are computer-based systems that control a process whose correctness depends on TIME.

- Telecommunications



Example: an MPEG Video Player

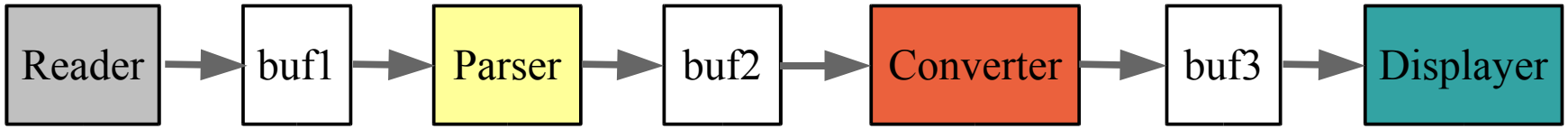


Functional requirements:

F1: read a bitstream with encoded data
F2: parse the data and reconstruct YUV images
F3: convert YUV frames in RGB
F4: display RGB frames in the proper sequence

MPEG Video Player: a straightforward solution

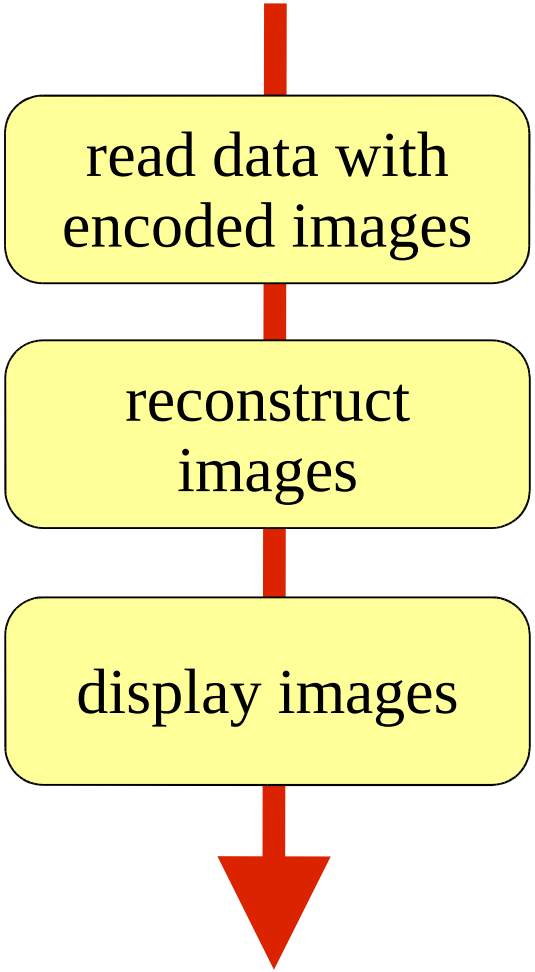
Arrange concurrent threads in a pipelined architecture:



Pros: simple to implement and understand.

Cons: ?

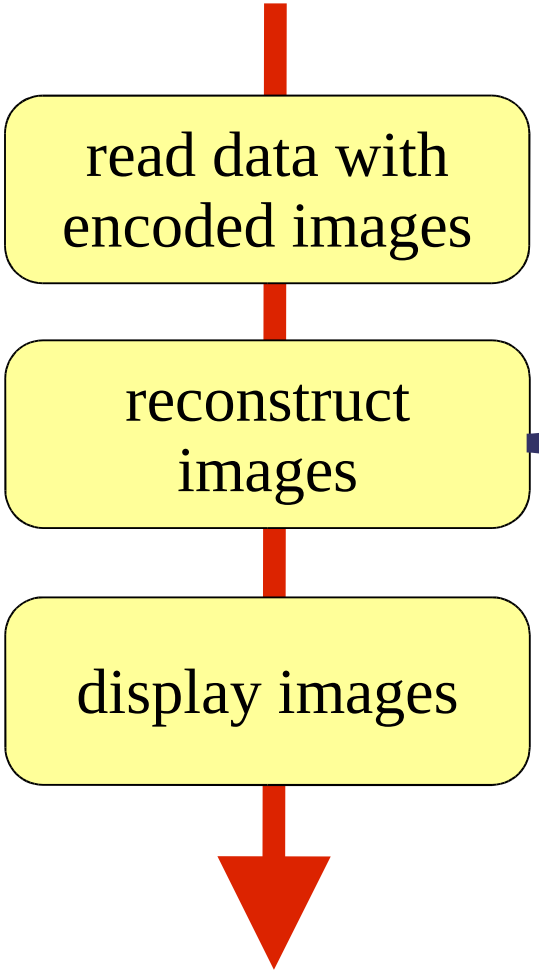
Are we missing something ?



Functional requirements:

F1: read a bitstream with encoded data
F2: parse the data and reconstruct YUV images
F3: convert YUV frames in RGB
F4: display RGB frames in the proper sequence

We forgot temporal requirements !



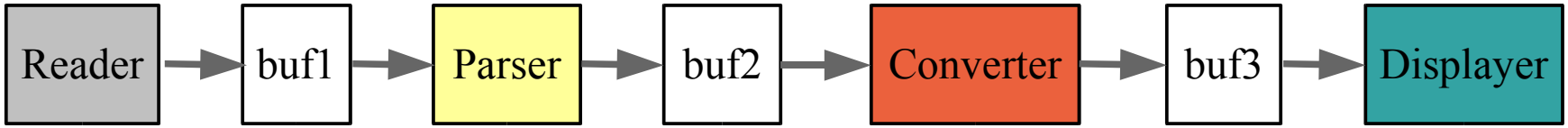
Functional requirements:

F1: read a byte stream with encoded data
F2: parse the data and reconstruct YUV images
F3: convert YUV frames in RGB
F4: display RGB frames in the proper sequence

Temporal requirements:

NF1: read promptly incoming data, or may be lost
NF2: adapt to fluctuating incoming data rate
NF3: absorb the variable duration of the algorithms
NF4: display RGB frames with the proper timing

MPEG Video Player: a straightforward solution



Temporal requirements:

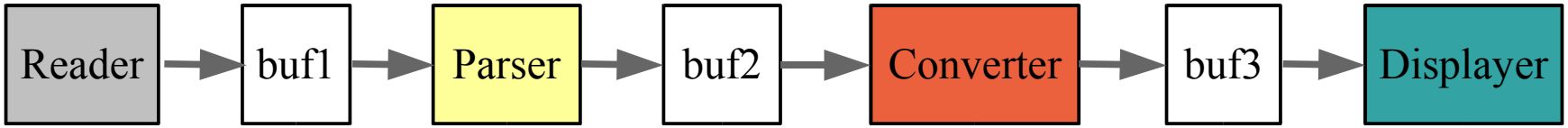
NF1: read promptly incoming data, or may be lost

NF2: adapt to fluctuating incoming data rate

NF3: absorb the variable duration of the algorithms

NF4: display RGB frames with the proper timing

MPEG Video Player: a straightforward solution



Pros: simple to implement and understand

Cons:

- No guarantee to meet *non-functional (temporal) requirements*
- May work in systems dimensioned for the worst case
- Fragile: correctness depends (among other things) on relative threads speed

Real-Time Systems: a definition

"In real-time computing the correctness of the system depends not only on the logical results of the computation but also on the time at which the results are produced."

J. Stankovic

Misconceptions About Real-Time Computing: A serious Problem for Next Generation Systems

IEEE Computer, October 1988

Misconceptions about real-time systems

- Real-time is just a matter of performance engineering
  but fast systems are not necessarily predictable
- Real-time systems operate in static environment
  but they interact and depend on the close interaction with the environment
- No science
  but synthesis of temporally-correct executable systems still elusive !
- ...and many others (see the paper !)

Time-Sensitive Systems

The correctness of *Time-Sensitive Systems* depends on:

- Timeliness (punctuality)
- Predictability
- Responsiveness
- Quality of Service (as perceived by the user)

} classic real-time

Time-Sensitive Systems

The correctness of *Time-Sensitive Systems* depends on:

- Timeliness (punctuality)
- Predictability
- Responsiveness
- Quality of Service (as perceived by the user)
- Performance
- Power consumption

classic real-time

additional time-related aspects

Time-Sensitive Systems

Their correctness depends on a number of *temporal aspects of computation:*

- Timeliness (punctuality)
- Predictability
- Responsiveness
- Quality of Service (as perceived by the user)
- Performance
- Power consumption

Time-sensitive systems

Open research issues:

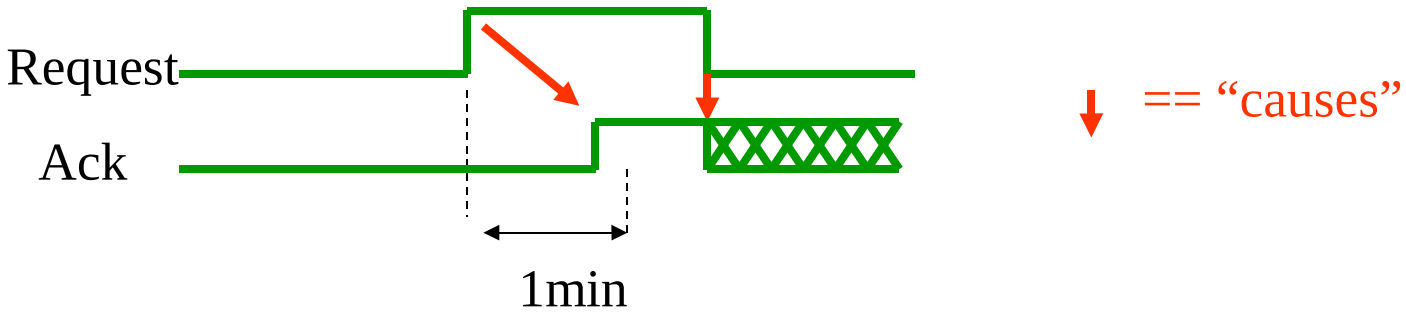
- How to deal with and combine req. on temporal aspects of computation ?
- How to design and implement time-sensitive systems ?

Current Approaches to Real-Time Systems Design

Theoretical approaches and formal specifications
- Temporal Logic
- Timed-CSP
- Timed Petri Nets
- ...

Real-time scheduling
- clock driven
- fixed priority – Rate Monotonic Analysis
- dynamic priority – Earliest Deadline First
- issues: priority inversion, implicit behavior...

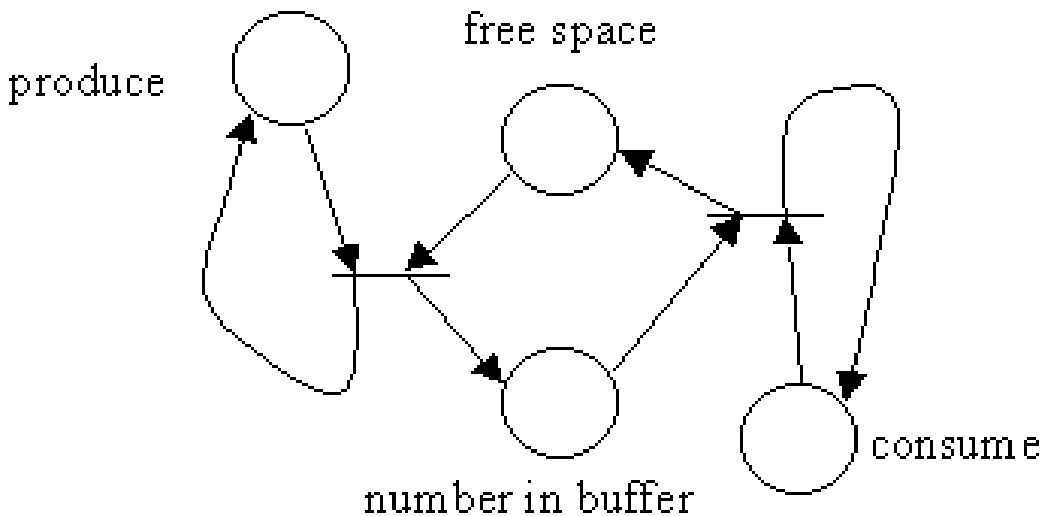
Formal specifications (sort of...)



Always (Request implies

Eventually $_{<=1\ minute}$ Ack *Until* Not Request)

Petri Nets
Example: Producer-Consumer

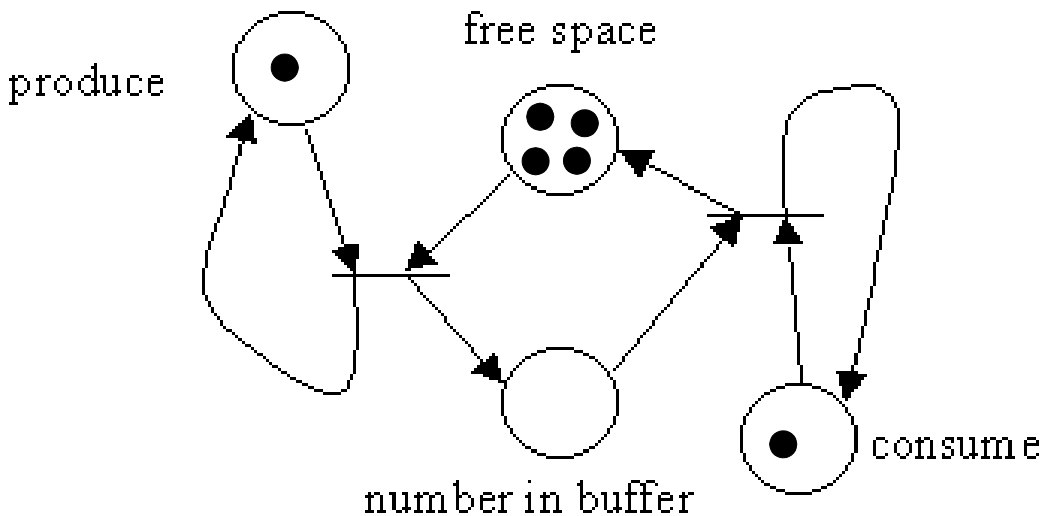


Places
Transitions
Tokens

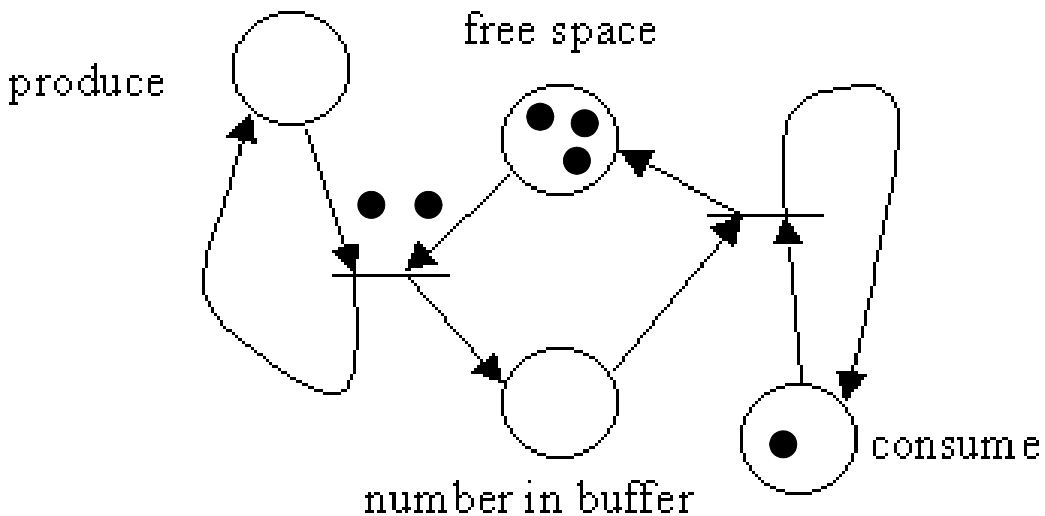
Model concurrent and distributed systems

Formalization allows proofs of properties (liveness/deadlocks, synchronization)

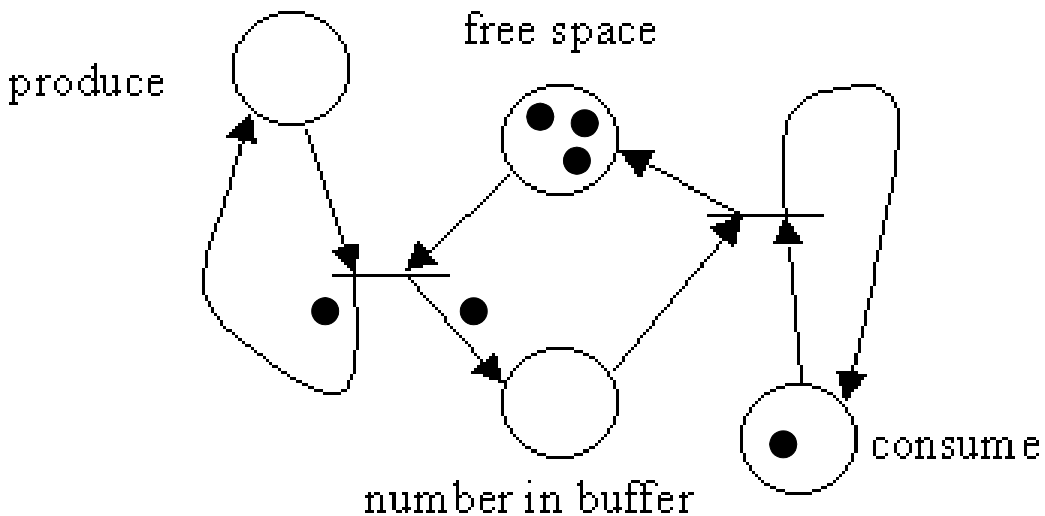
Petri Nets
Example: Producer-Consumer - 1



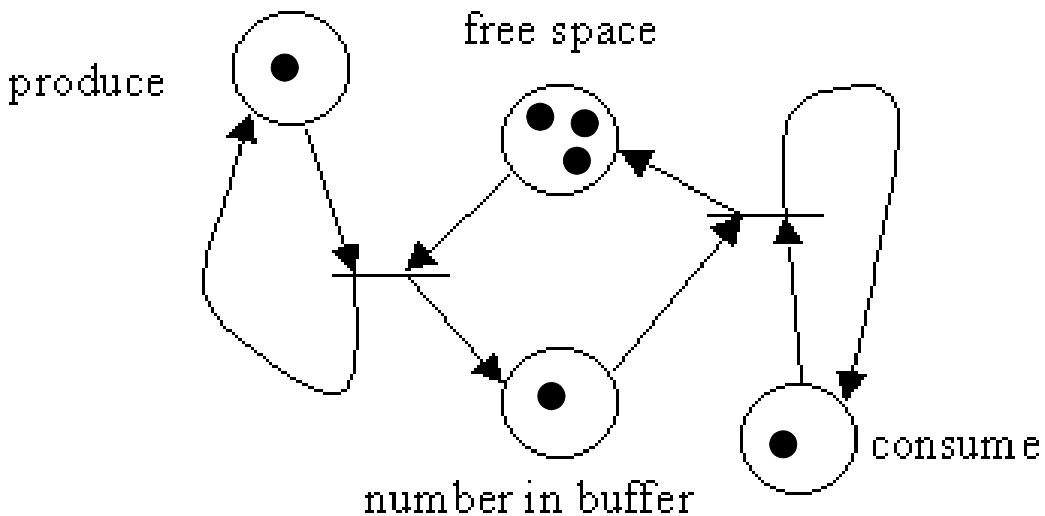
Petri Nets
Example: Producer-Consumer - 2



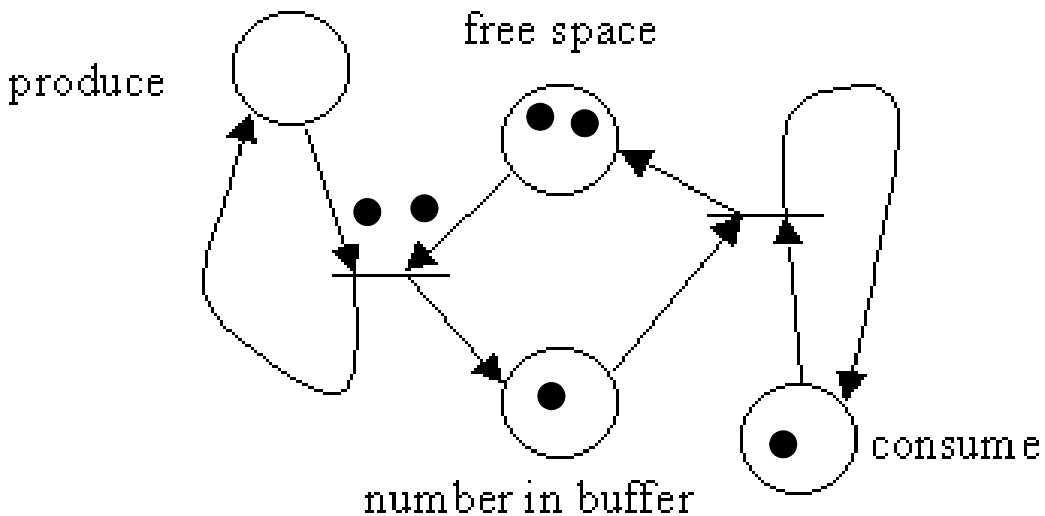
Petri Nets
Example: Producer-Consumer - 3



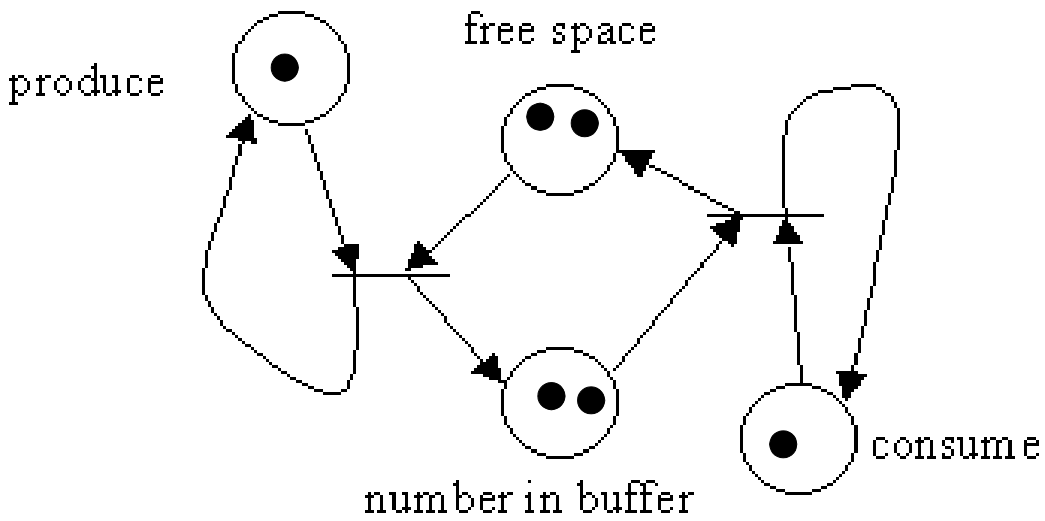
Petri Nets
Example: Producer-Consumer - 4



Petri Nets
Example: Producer-Consumer - 5

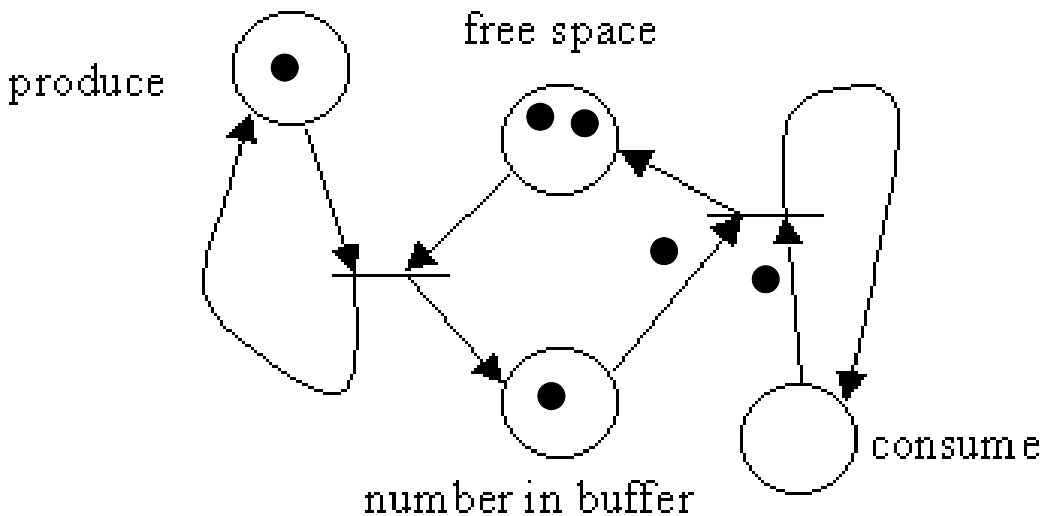


Petri Nets
Example: Producer-Consumer - 6

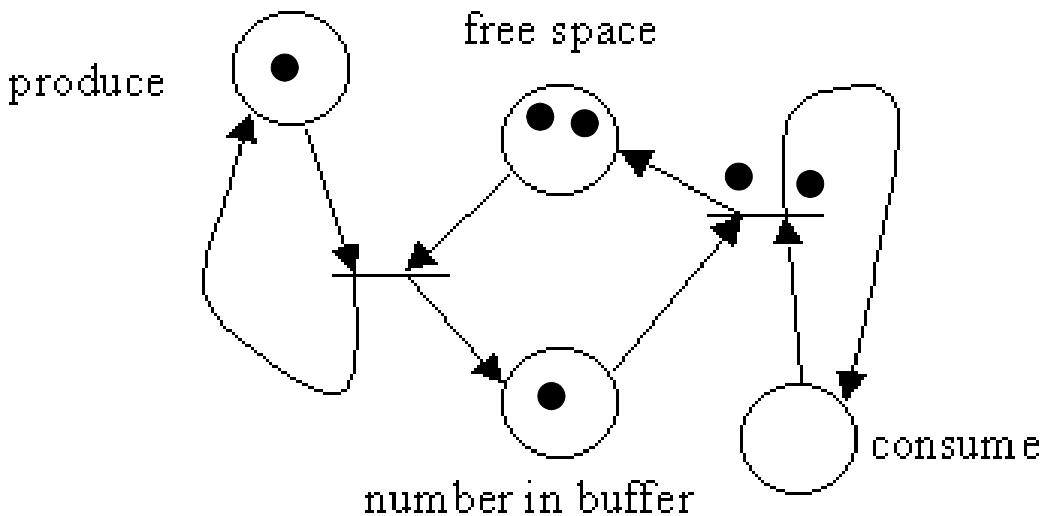


Petri nets do not necessarily evolve in a deterministic way

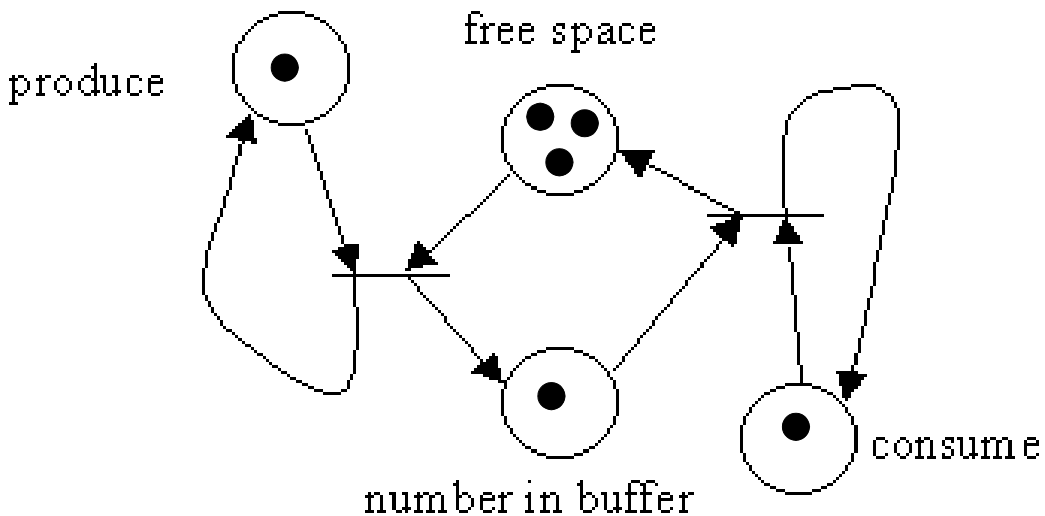
Petri Nets
Example: Producer-Consumer - 7a



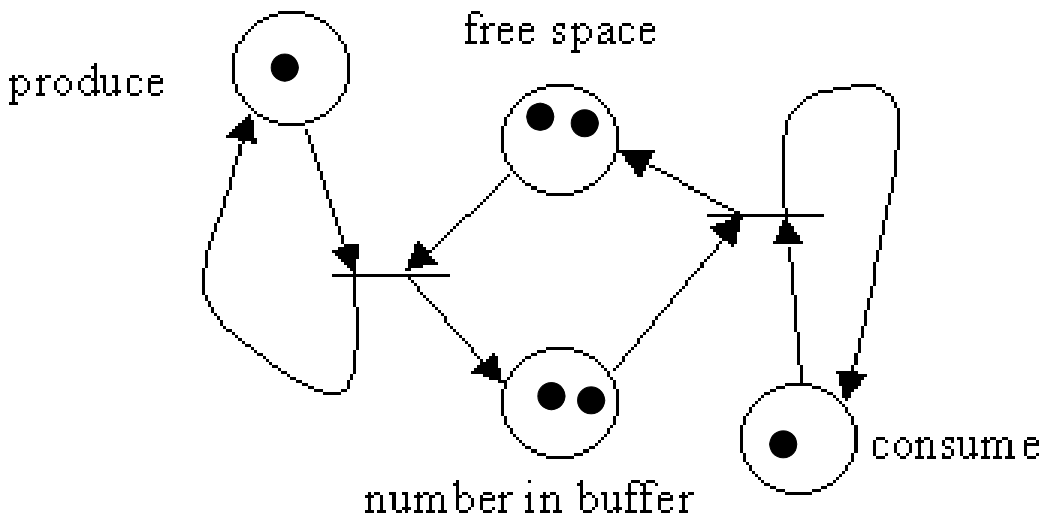
Petri Nets
Example: Producer-Consumer - 8a



Petri Nets
Example: Producer-Consumer - 9a

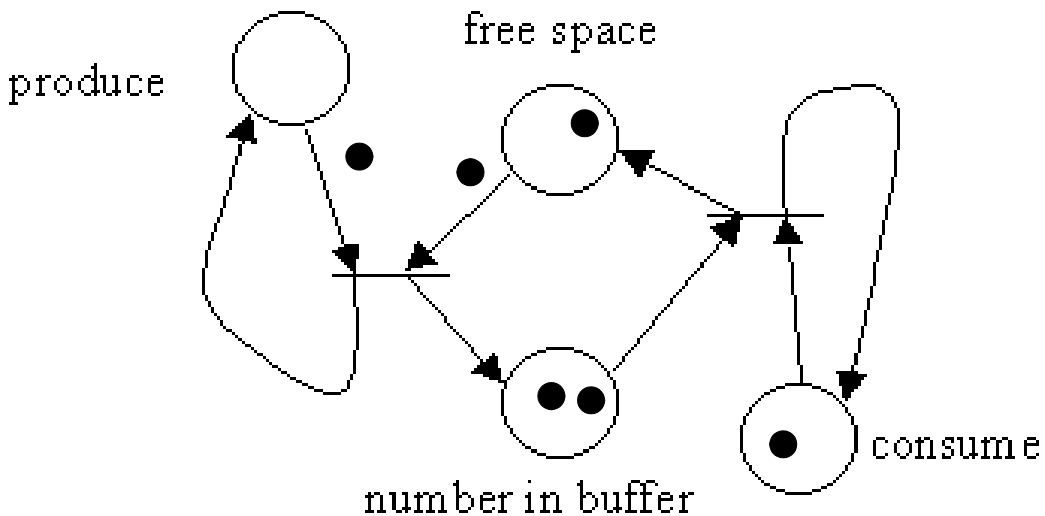


Petri Nets
Example: Producer-Consumer - 6

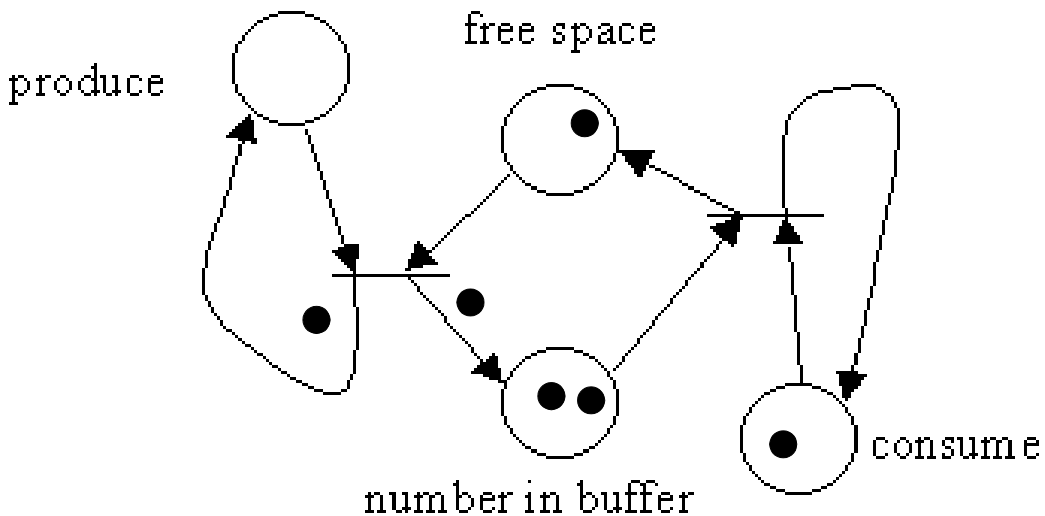


Petri nets do not necessarily evolve in a deterministic way

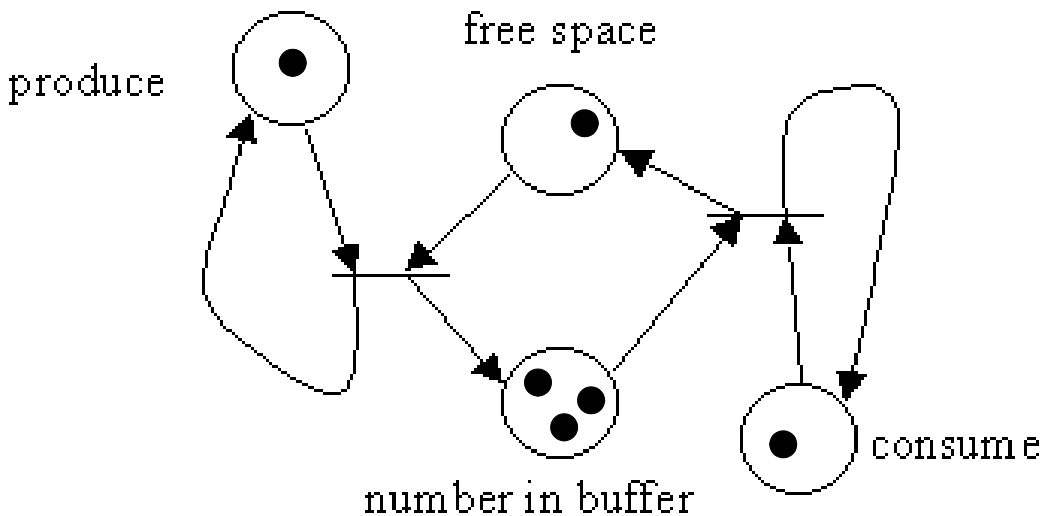
Petri Nets
Example: Producer-Consumer - 7b



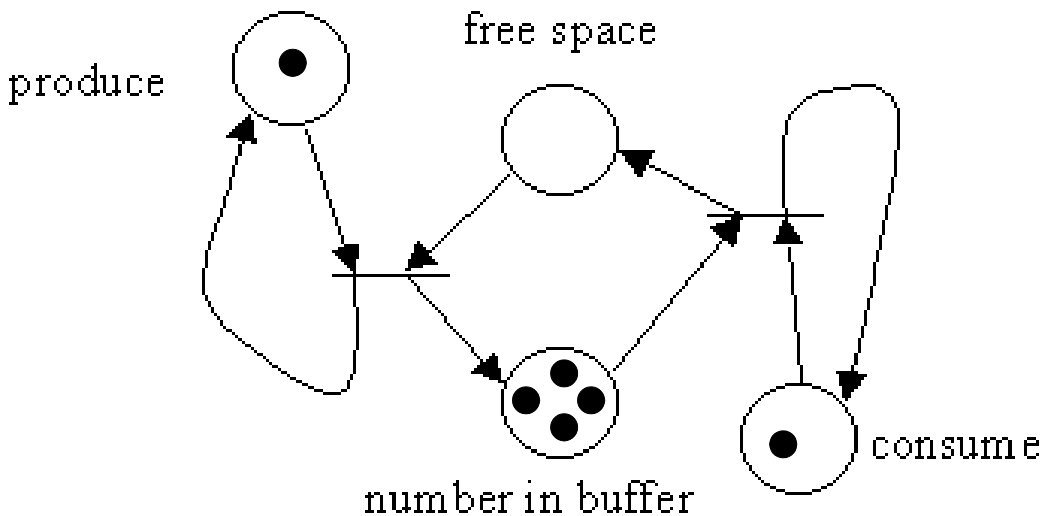
Petri Nets
Example: Producer-Consumer - 8b



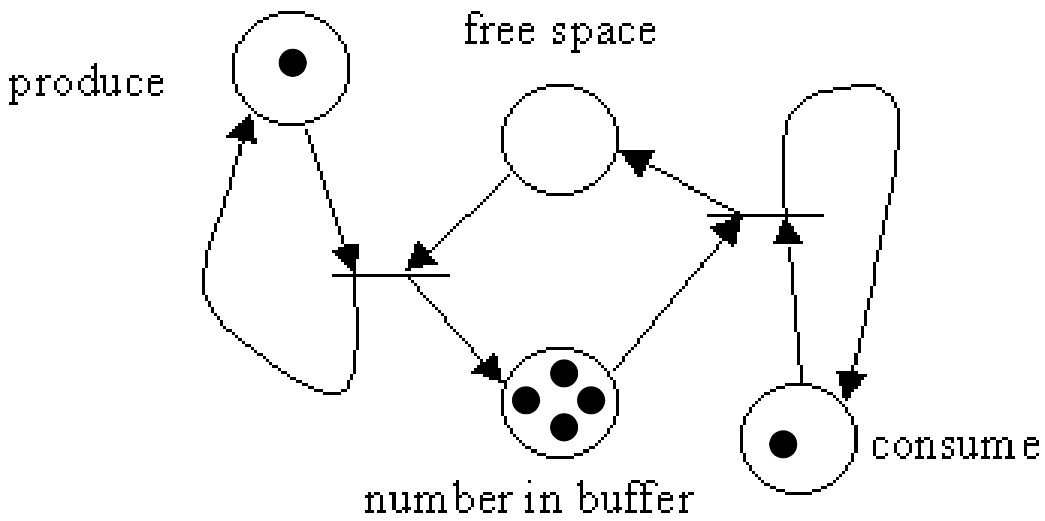
Petri Nets
Example: Producer-Consumer - 9b



Petri Nets
Example: Producer-Consumer - 10b

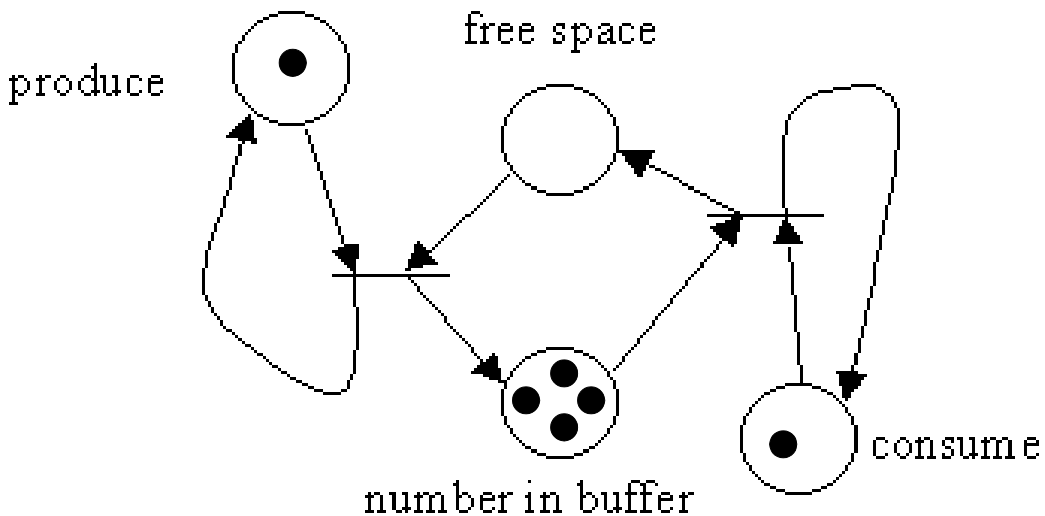


Petri Nets
Example: Producer-Consumer - 11b



(no transition)

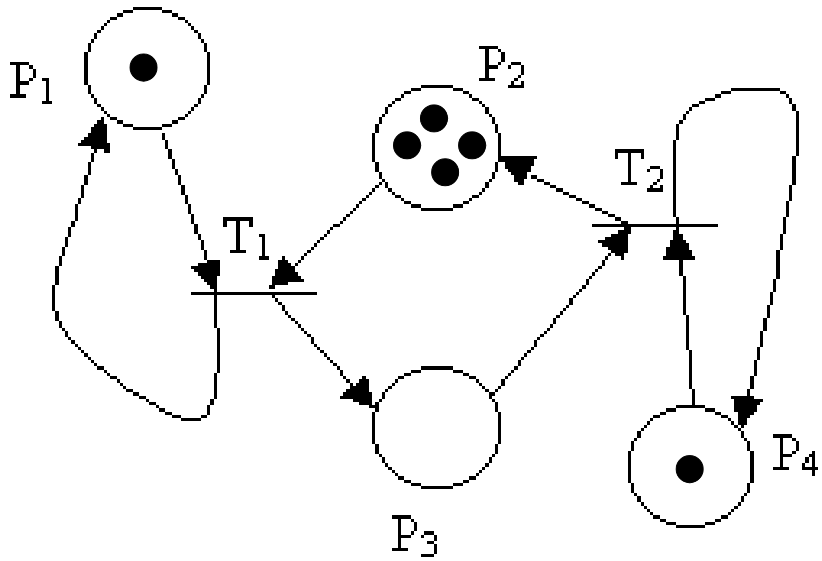
Petri Nets
Example: Producer-Consumer - 12b



(no transition)

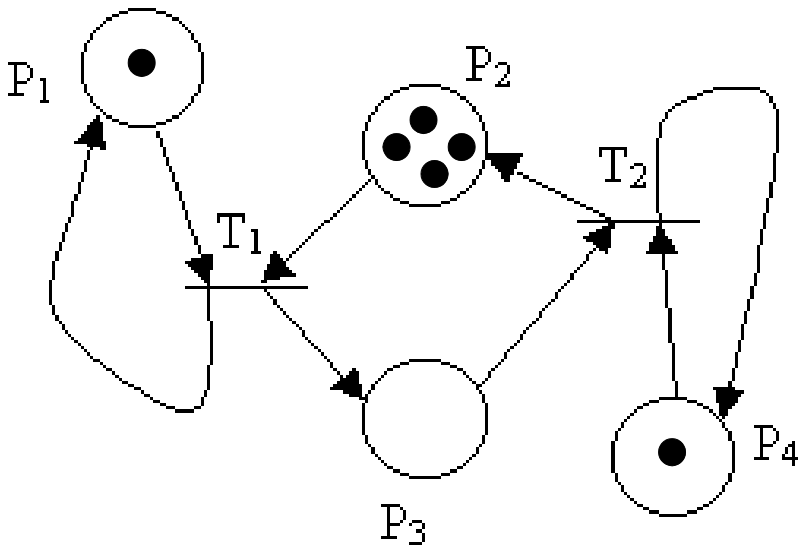
Standard Petri Nets do not include time

Timed Petri Nets Example:Producer-Consumer



Transition	Inputs	Min	Max
T1	P1,P2	0	P1+200
T2	P3,P4	P3+10	

Timed Petri Nets Example:Producer-Consumer



Problem: how to run the model on a computer ?

Transition	Inputs	Min	Max
T1	P1,P2	0	P1+200
T2	P3,P4	P3+10	

Issues of Formal Approaches to Real-Time Systems Design

- large semantic gap (specifications are not executable)
- rely on severely unrealistic assumptions
- do not address the implicit behavior in the system
- induce static design, result in rigid and brittle systems
- not integrated with QoS, power consumption, performance requirements

I can call spirits from the vasty deep.

Why, so can I, or so can any man; But will they come when you do call for them?

William Shakespeare
King Henry the fourth,
Part I (Hotspur at III, i)

Real-time Scheduling

Hard real-time:

service is guaranteed within a fixed time interval (deadline).

Soft real-time:

a statistical distribution of response times is acceptable.

If all request for all tasks at their critical instant are fulfilled,

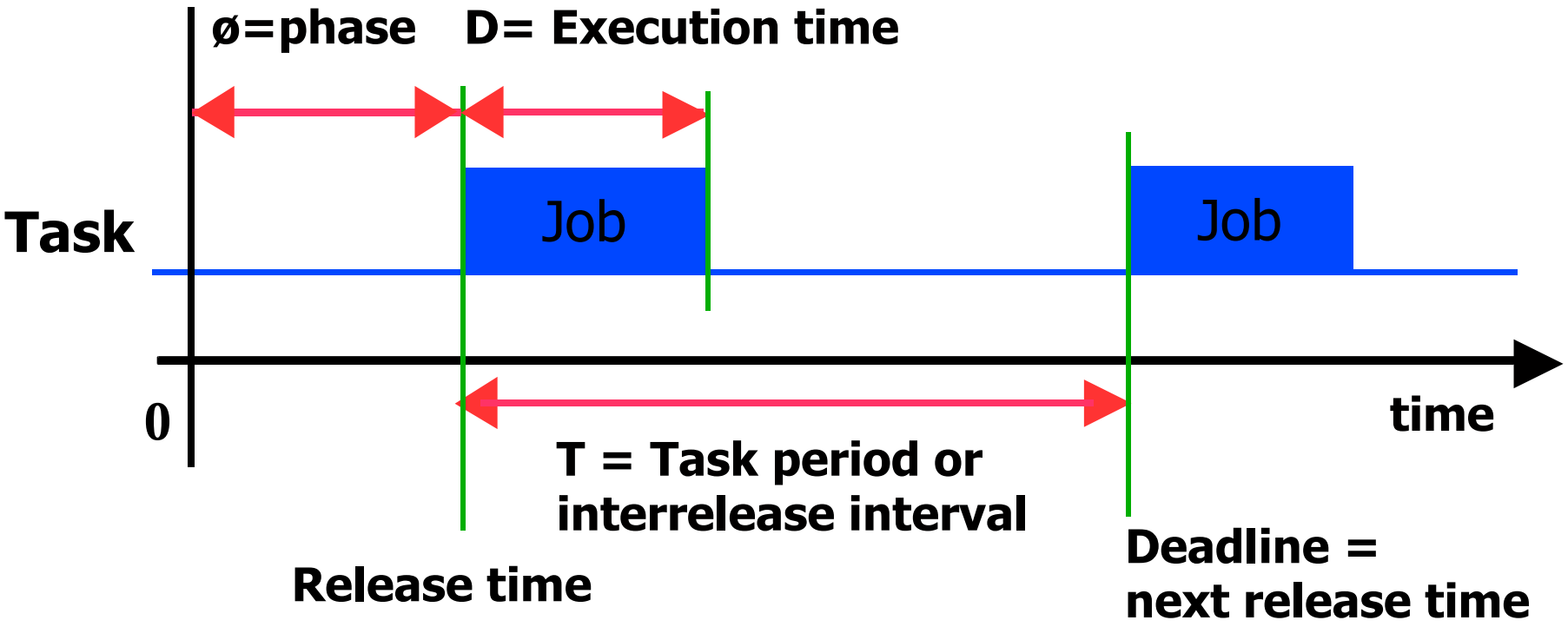
then the scheduling is *feasible*.

Pre-emptive: task can be suspended and resumed later

(N.B: tasks in critical sections are non-preemptable)

Periodic Task Model

Task = (T, D, ø)



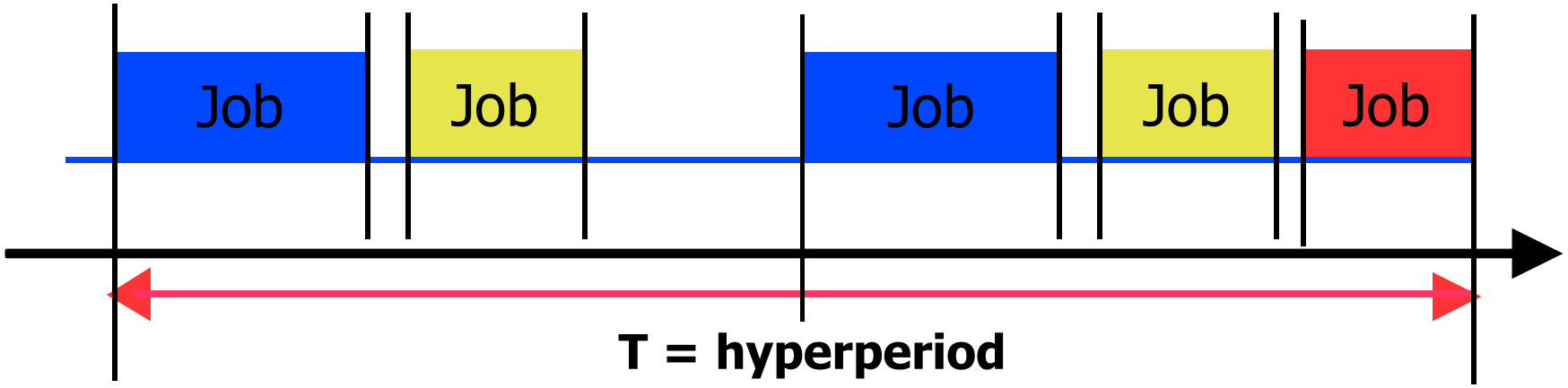
Time/Clock-Driven Scheduling

Which job execute at what times is decided at specific time instants

Instants chosen *a priori* before system begins execution.

All parameters of hard real-time jobs are fixed and known.

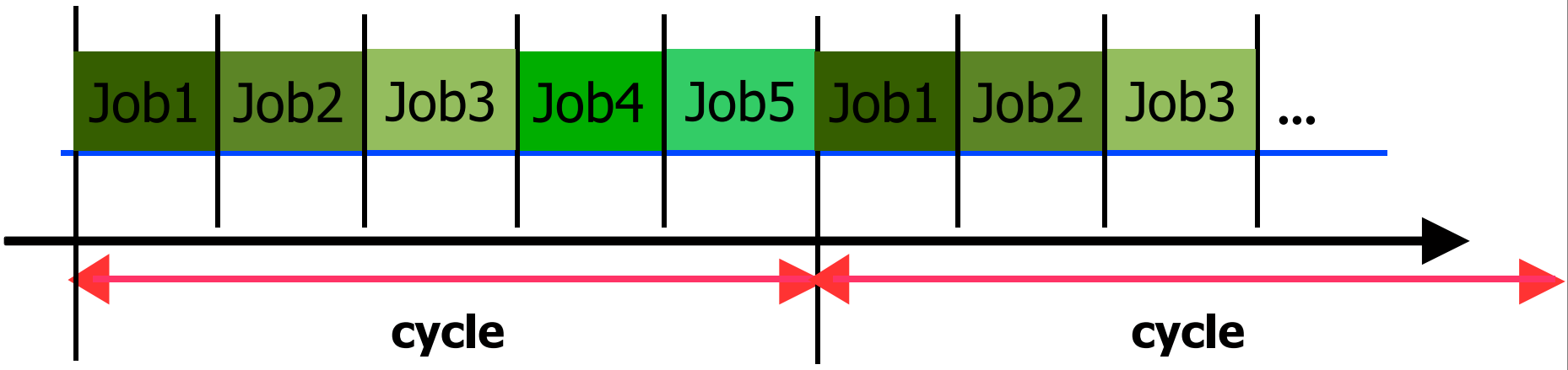
A schedule of the job is computed off-line and stored for use at run-time.



Round-Robin Approach

Round-robin used to schedule time-shared applications (general purpose OS).

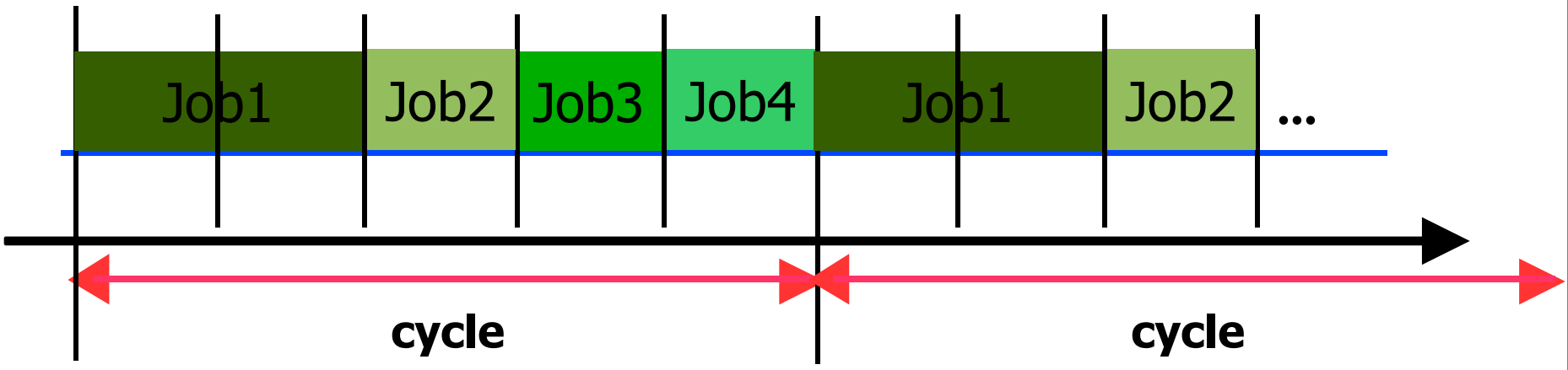
Each one of n tasks get 1/n of the processor.



Weighted Round-Robin Approach

Weighted RR schedules real-time traffic in high-speed switched networks.

A job with weight *w* gets *w/n* time slices of the processor.



Priority-Driven Scheduling

Scheduling decisions (wich job to run) are made when *events* like jobs releases and completitions occurs.

- Static (or Fixed) Priority Scheduling

  Algorithms assign priorities to task with deadlines at design time

  Ex: Rate Monotonic priority assignment

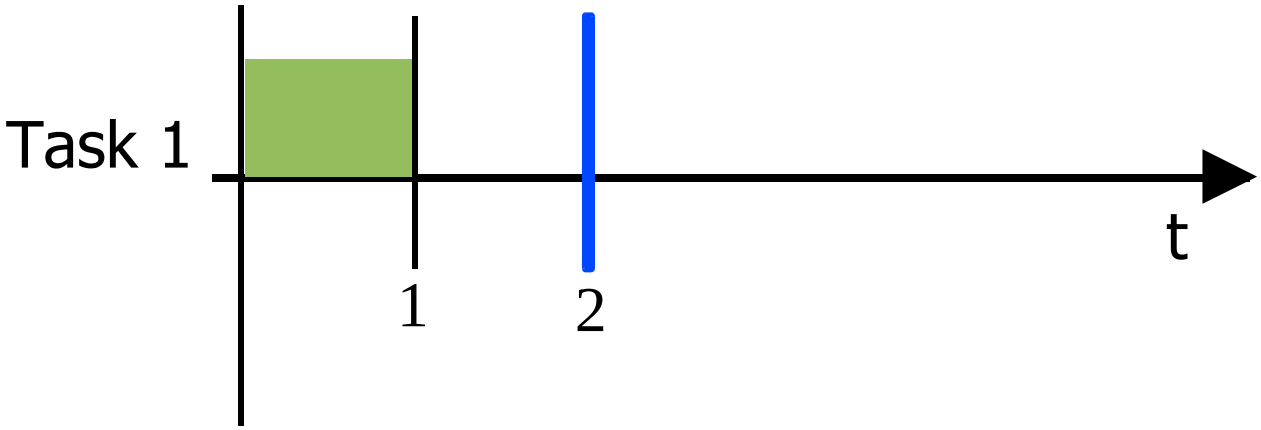
- Dynamic priority scheduling

  Priorities change during execution

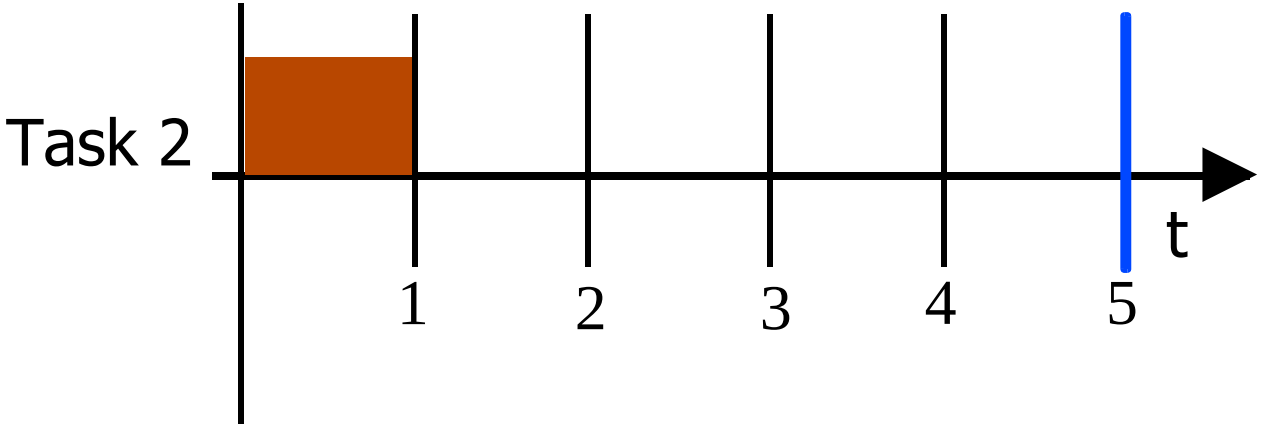
  Many existing operating systems do not support dynamic scheduling (pri.ch. costs)

  Ex: Earliest Deadline First priority assignment

Rate-Monotonic priority assign.

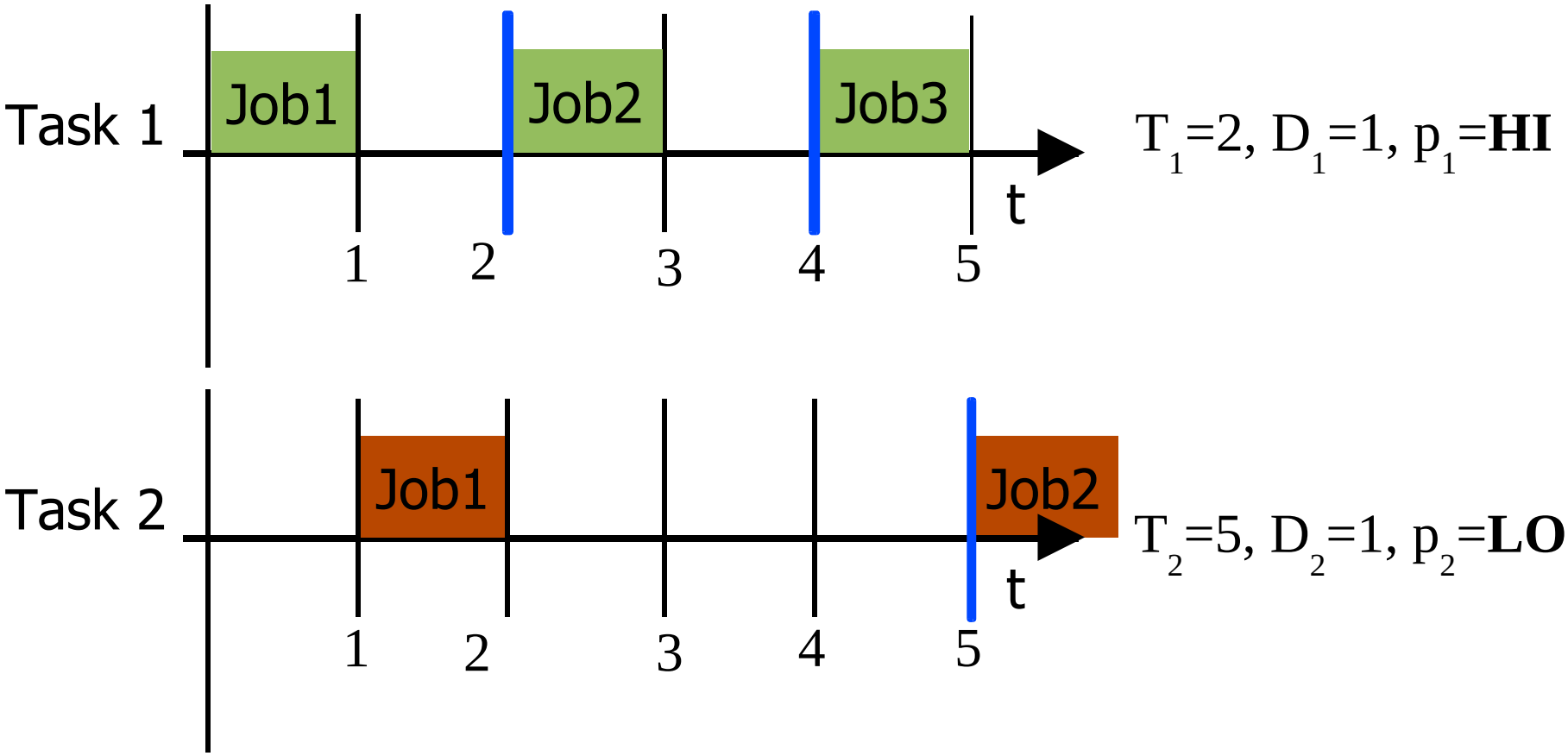


$T_1=2$, $D_1=1$, $p_1=$ **?**



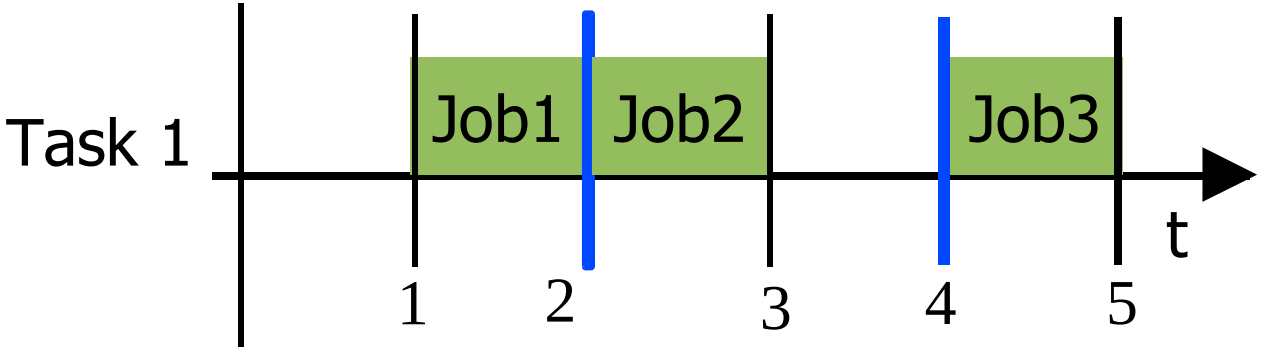
$T_2=5$, $D_2=1$, $p_2=$ **?**

Rate-Monotonic Scheduling

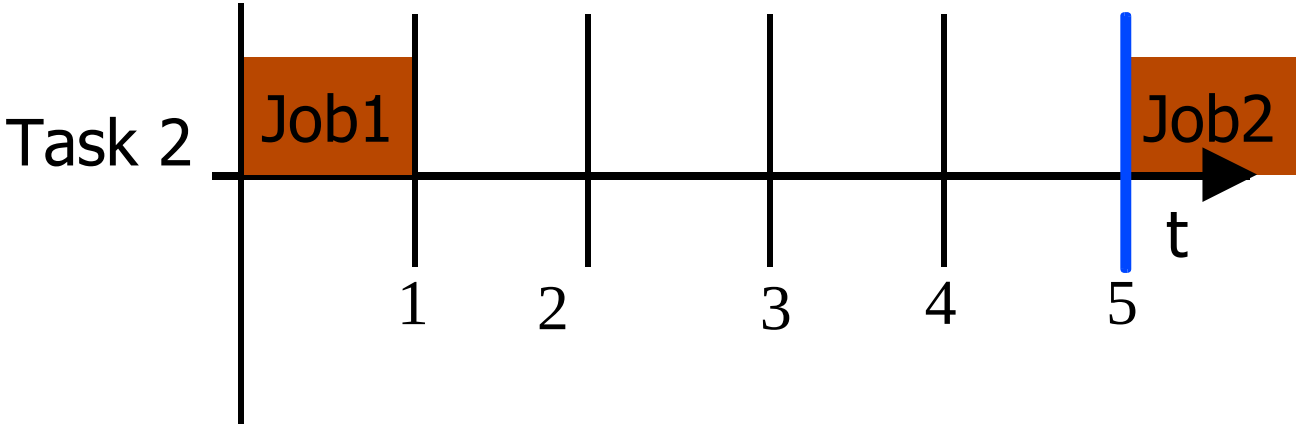


Feasible

Rate-Monotonic Scheduling



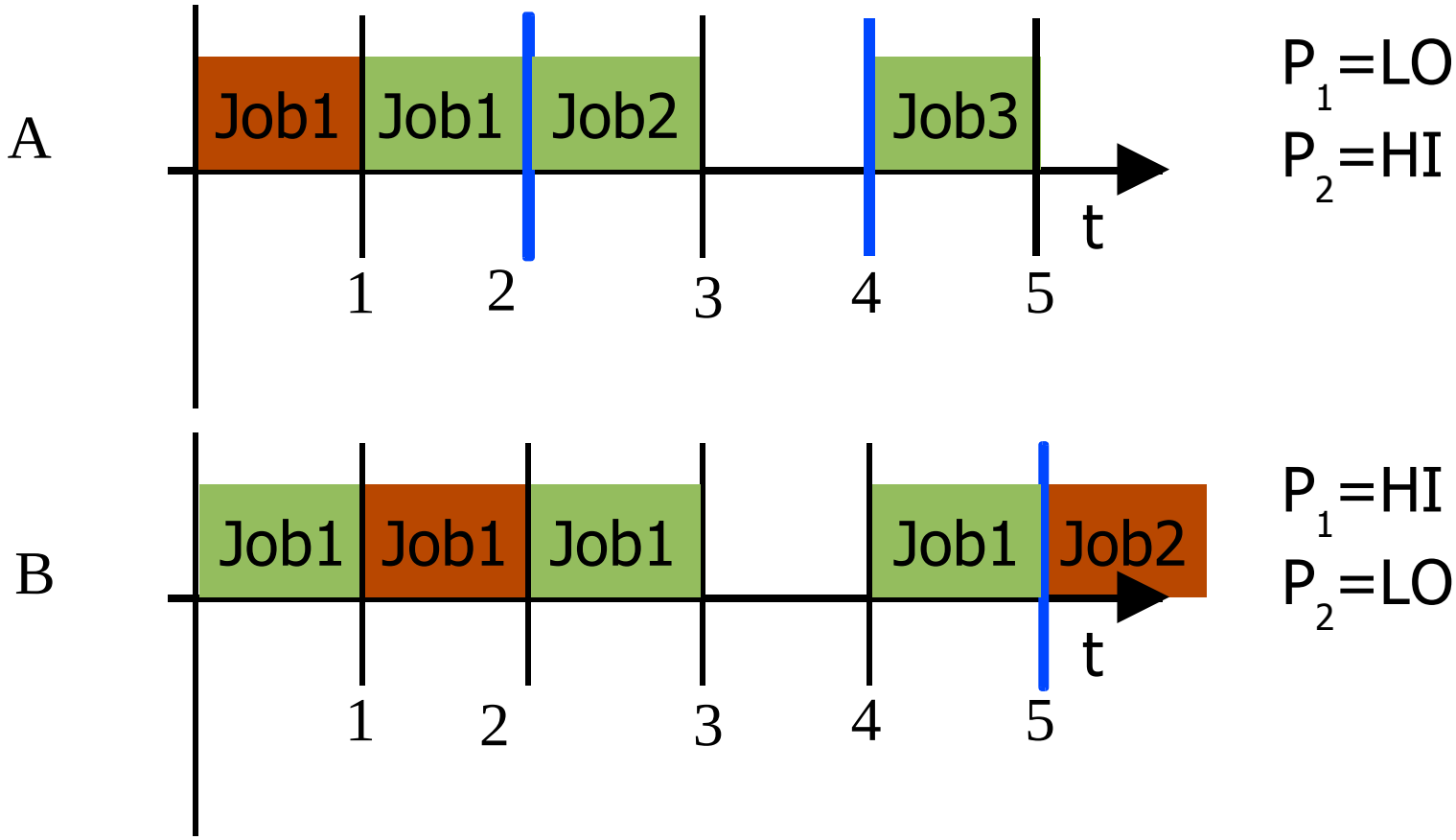
$T_1=2$, $D_1=1$, $p_1=$**LO**



$T_2=5$, $D_2=1$, $p_2=$**HI**

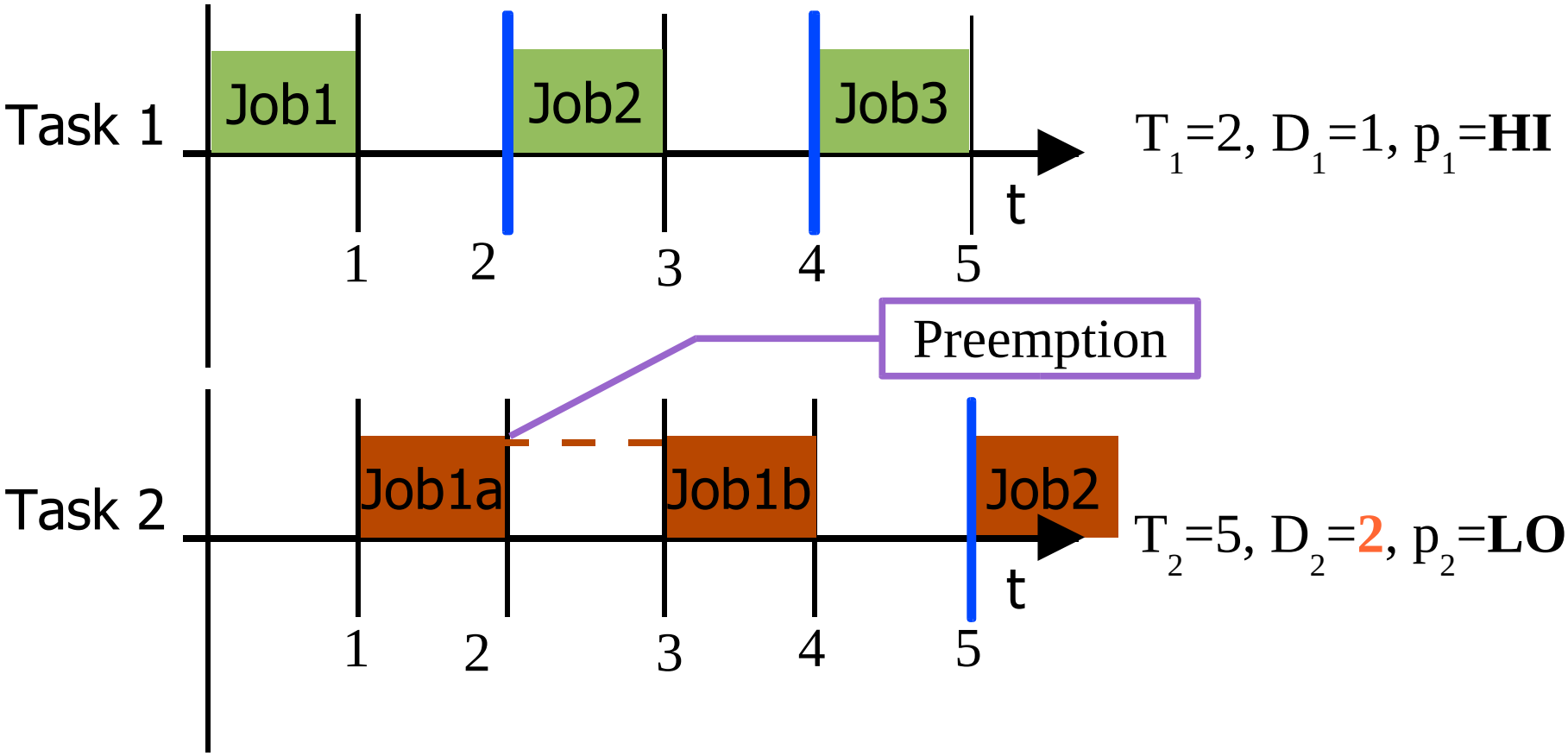
Feasible

Rate-Monotonic Scheduling



Both schedules are feasible, which one is the best ?

Rate-Monotonic Scheduling



With schedule B, D_2 can increase !

Rate-Monotonic Scheduling

Rate-monotonic Priority Assignment Rule:

Assign priorities according to tasks request rates (1/period).

Tasks with higher request rates will have higher priorities.

Theorem:

If a feasible priority assignment exists for some task set,
the rate-monotonic priority assignment is feasible for that task set.

[Liu, Layland 73]

Rate-Monotonic Scheduling

Definition:

The utilization factor is the fraction of processor time spent in the execution of the task set.

$$U = \sum_{i=1}^{m} \left(\frac{Di}{Ti}\right)$$

U is upper bounded by the requirement that all tasks meet deadlines.

For all task sets whose processor utilization factor is below the least upper bound *of the utilization factor, there exists a fixed priority assignment which is feasible.*

This condition is sufficient, but not necessary for feasibility.

Rate-Monotonic Scheduling

Theorem:

For a set of m tasks with fixed priority order, the least upper bound to processor utilization is

$$U = m(2^{1/m} - 1)$$

RMA produces a feasible scheduling if (but not only if):

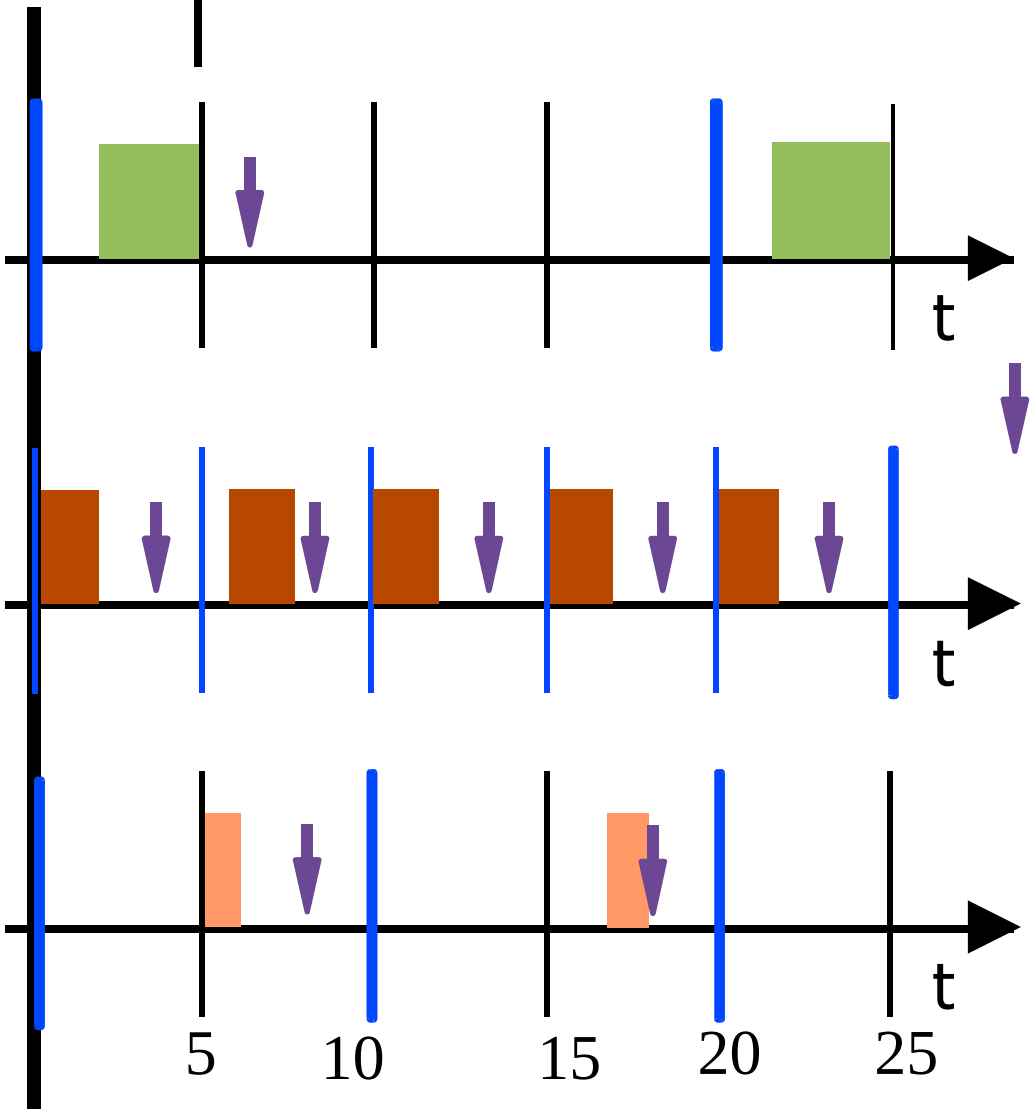
$$U = \sum_{i=1}^{m} \left(\frac{Di}{Ti}\right) < m(2^{1/m} - 1)$$

For large m,

$$U = \lim_{m \to \infty} m(2^{1/m} - 1) \approx \ln(2) = 0.693$$

[Liu, Layland 73]

Earliest Deadline First



$T_1=20,\ C_1=3,\ D_1=7,\ \acute{\imath}_1=0$

deadline

$T_2=20,\ C_2=2,\ D_2=4,\ \acute{\imath}_2=0$

$T_3=10,\ C_3=1,\ D_3=8,\ \acute{\imath}_3=0$

Earliest Deadline First

Periodic and independent tasks

Preemptive

D<=T, EDF feasible IFF $U = \sum_{i=1}^{m} (\frac{Ci}{Ti}) \leq 1$

Dynamic priorities

Earliest deadline highest

Compute priorities at release/arrival time

Assumptions on Tasks...

A1) All requests are periodic with constant interval

A2) Deadlines only in terms of the next request

A3) Execution-independent (no pipes, IPC, Prod/Cons)

A4) Run-time is constant, does not vary with time

A5) Any non-periodic tasks are special:
- when run displace periodic task
- do not have themselves hard, critical deadlines

On assumptions...

The IEEE International Symposium on

Object-oriented Real-time distributed Computing

(ISORC)

Papers that are based on severely unrealistic assumptions will not be accepted however mathematically or logically sophisticated the discussion may be

Additional requirements...

``Many issues are outside the scope of this paper including

distributed scheduling

integration of cpu scheduling with communication scheduling and I/O scheduling

groups of tasks with a single deadline

placement constraints

the impact of this placement on the run time scheduling

fault tolerance needs

other kinds of timing requirements besides simple deadlines and periods,

integration of critical and non-critical tasks

the interaction of scheduling algorithms with the system design and implementation including run time overhead.

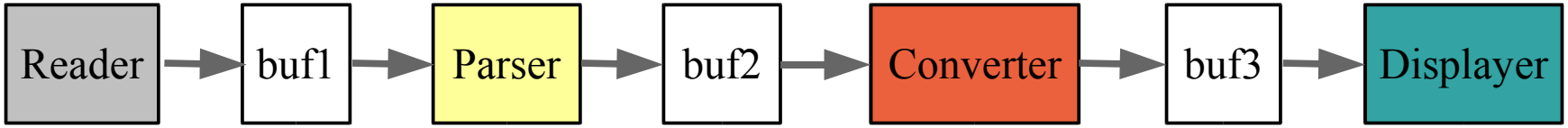
Most of these areas are wide open areas for research.".

Implications of Classical Scheduling Results For Real-Time Systems

John A. Stankovic, Marco Spuri, Marco Di Natale, Giorgio Buttazzo

IEEE Computer (1994)

MPEG Video Player: a straightforward solution



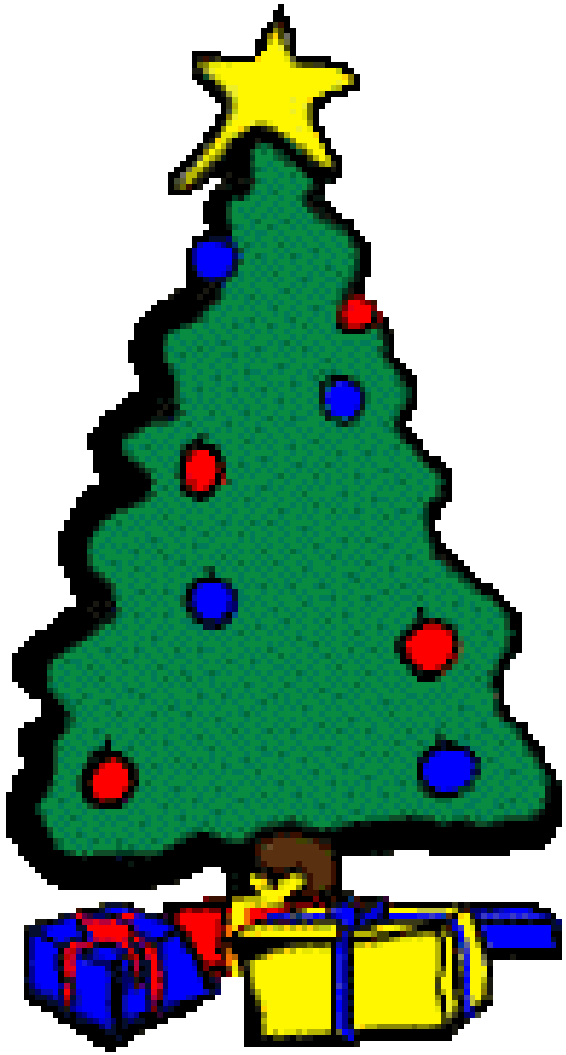
Pros: simple to implement and understand

Cons:

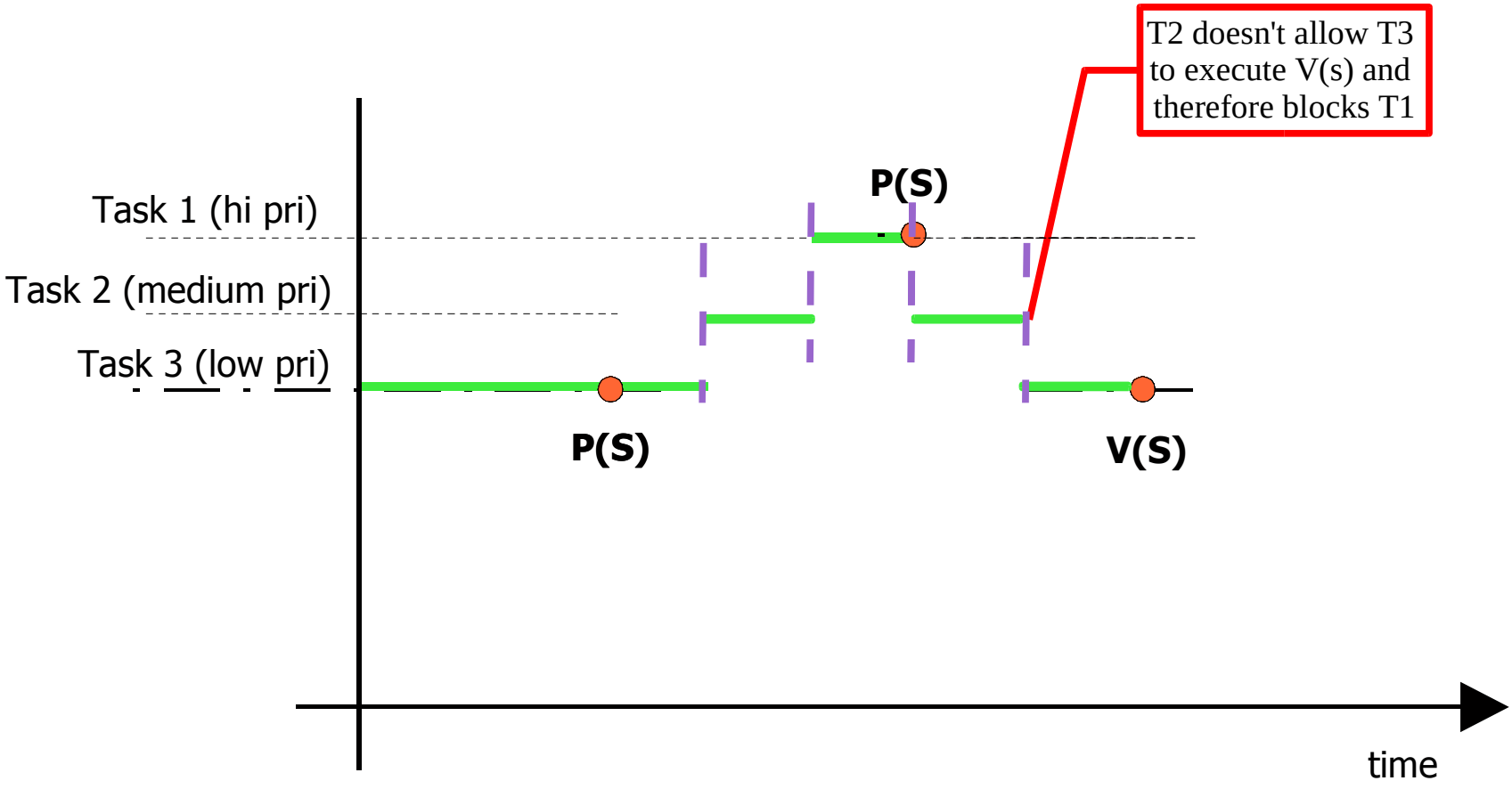
- No guarantee to meet *non-functional (temporal) requirements*
- ...
- Rate Monotonic Analysis is hard/impossibile to apply
  see "Multimedia Applications Require Adaptive CPU Scheduling", Baicenau *et al*., IEEE RTSS (1996)

So, what are scheduling algorithms good at ?

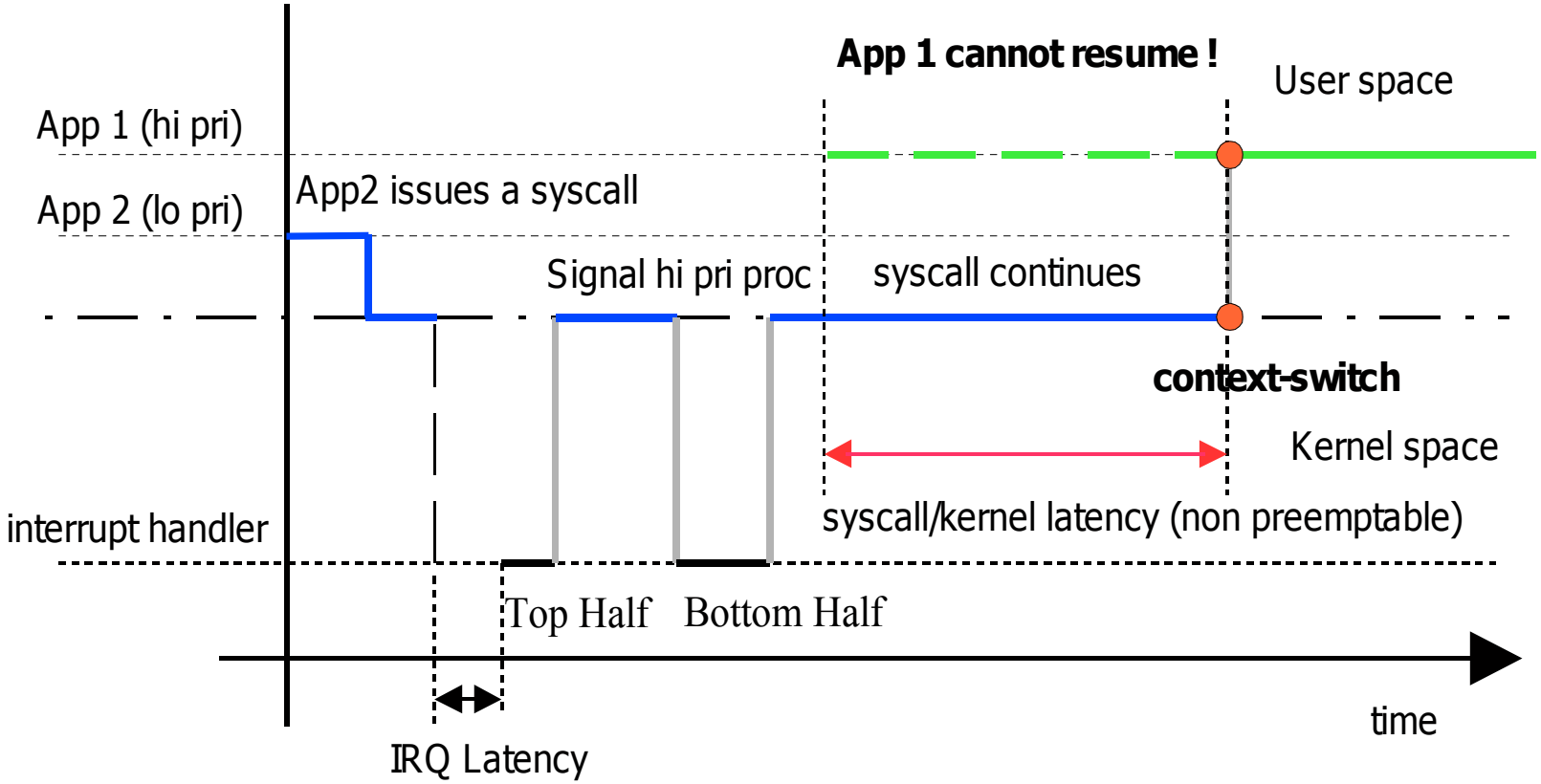
So, what are scheduling algorithms good at ?



Priority inversion: shared resource protected by semaphore ops P(s), V(s)



Kernel preemption and Priority Inversion



An example of implicit behaviour in real systems

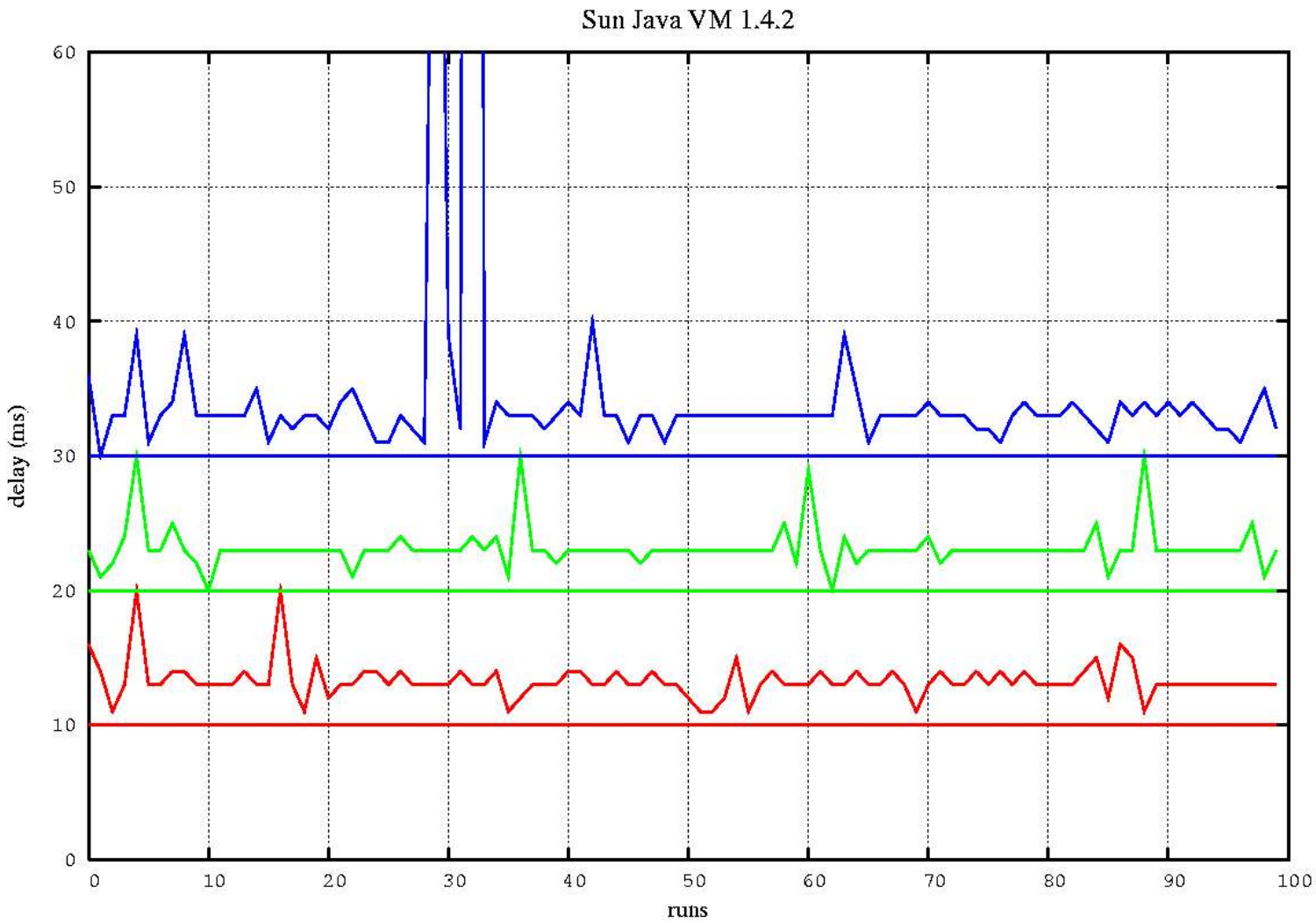
Main issue on a standard platforms:

the untimeliness of "timed" executions

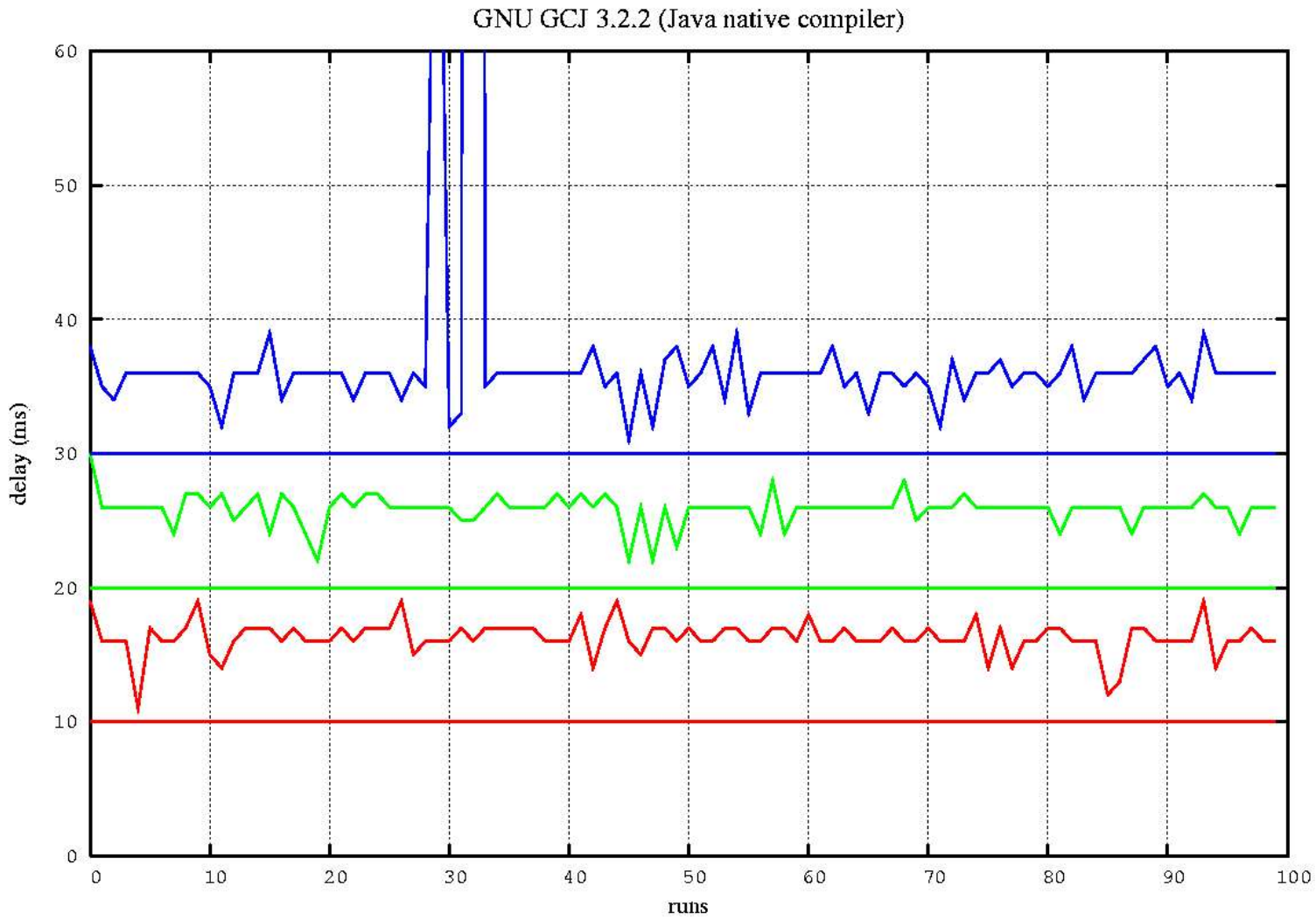
Likely sources of problems:

- language runtime (Java threads scheduling)
- garbage collector
- other internal VM activities (JIT, JNI etc.)
- underlying platform scheduling (Operating System + Middleware + ...)
- ...
- and their interaction !

Untimeliness of Timed Executions (Sun Java)

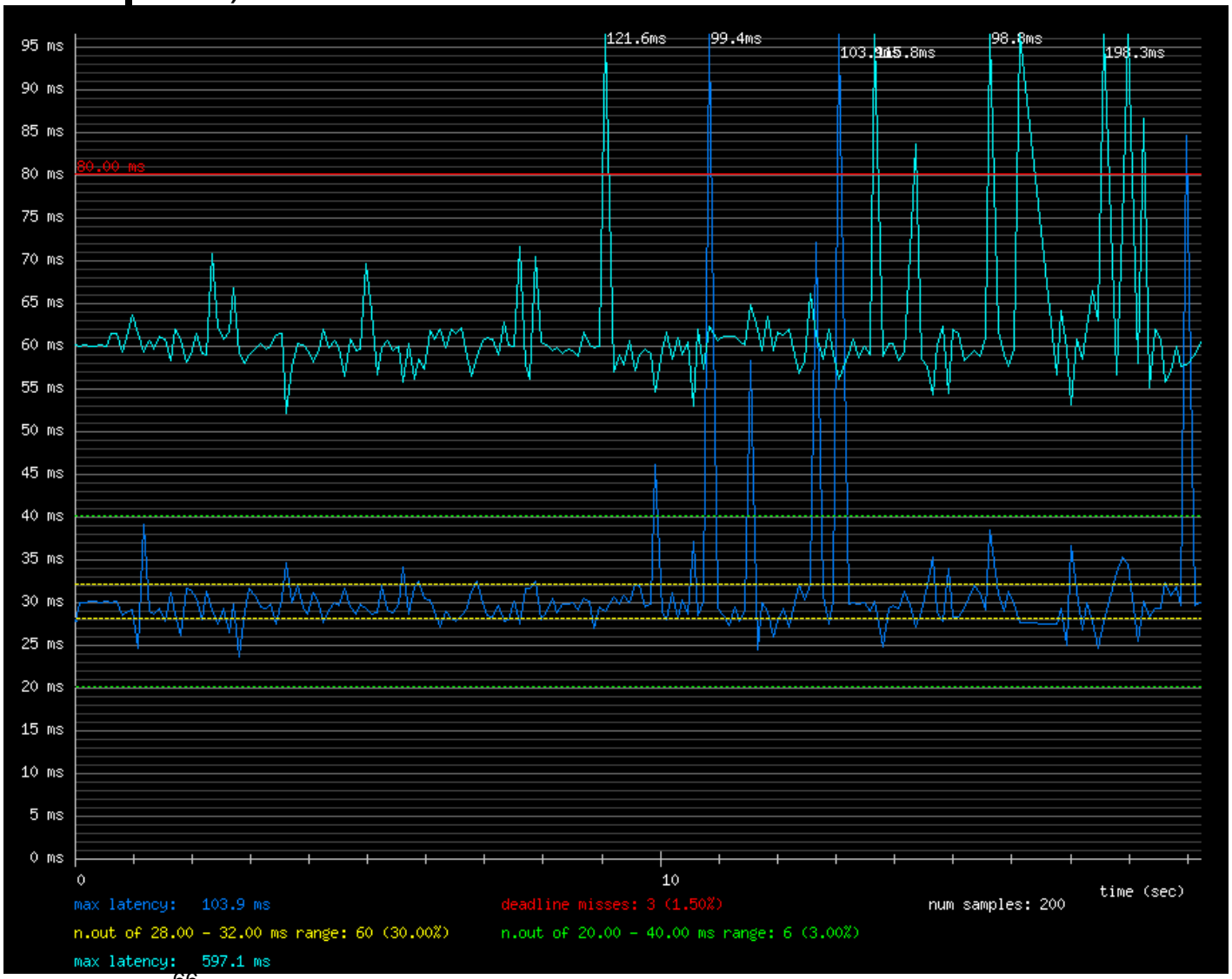


Untimeliness of Timed Executions (Gnu GCJ)

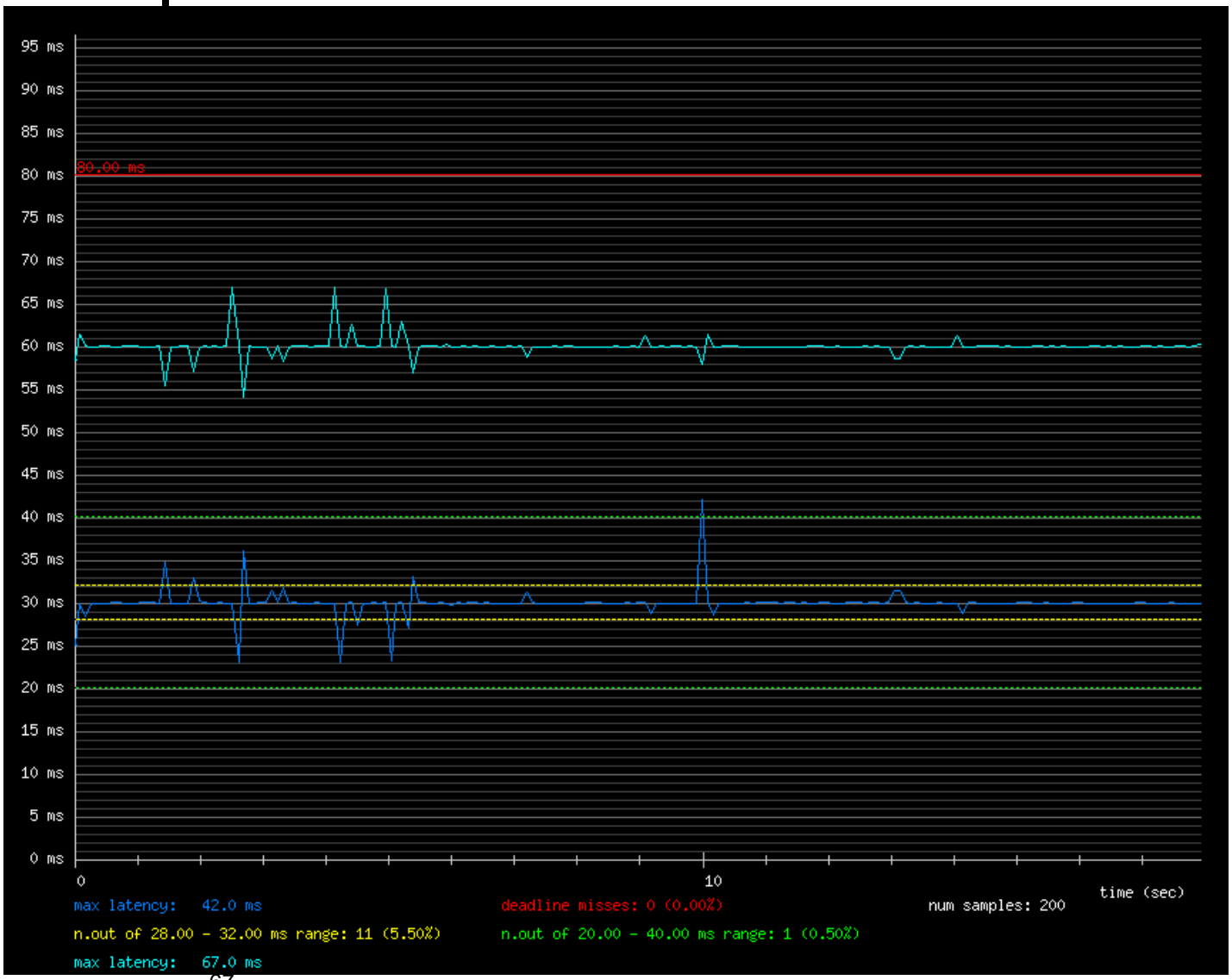


Low-latency patches by Benno Senoner

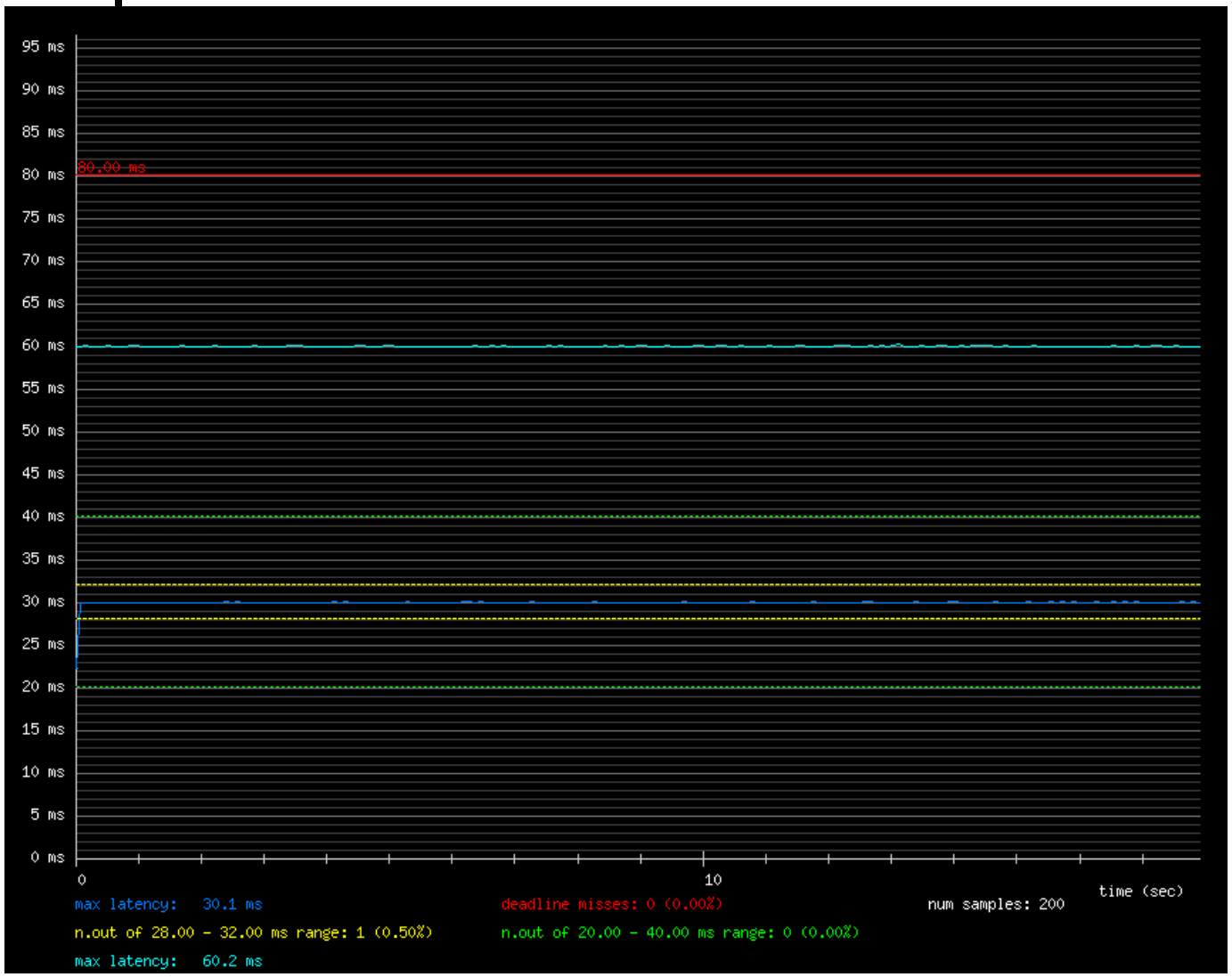
Latency test 30 e 60 ms on kernel 2.2.14 with hd
no DMA and high sistem load (disk, /proc stress, gfx output etc.).



Latency test 30 e 60 ms on kernel 2.2.14 with dma=on , 32bit access=on , unmask IRQ=on and high system load (disk, /proc stress, gfx output etc.)



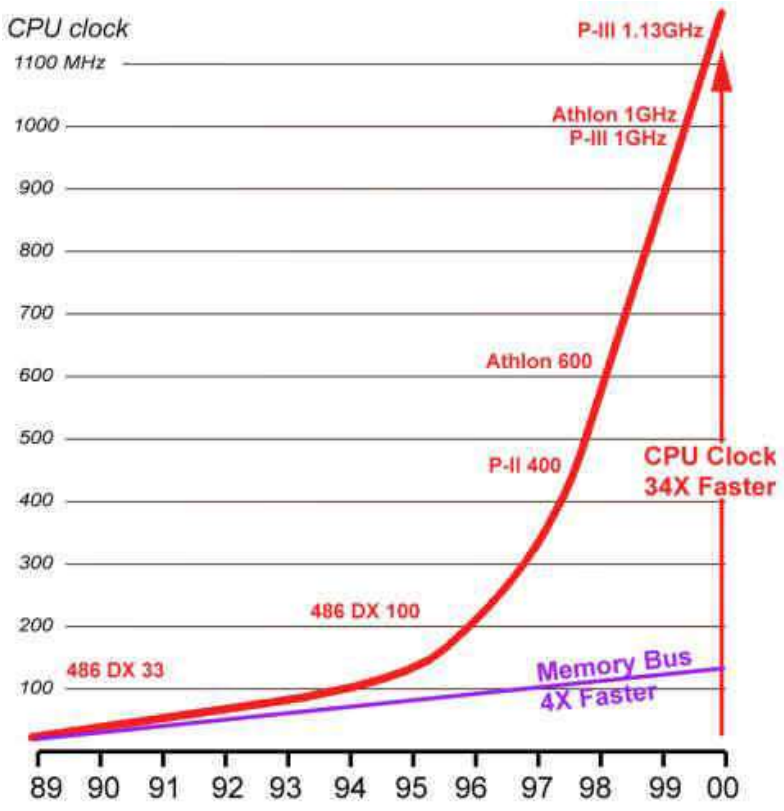
Latency test 30 e 60 ms on kernel 2.2.14 **with LowLatency Patch**
dma=on , 32bit access=on , unmask IRQ=on
and high system load (disk, /proc stress, gfx output ecc.).



Hardware & real-time

Timings, latencies, bandwidth of:

- CPU, MMU & caches
- RAM
- Chipset/bus controller
- I/O chips
- I/O bus(ses)
- Peripherals
- Influences of power availability
- etc.



Summary

Real-time is a whole-system issue

Little success in tackling it with traditional approaches

Basically an open problem since 30 years

Questions?