Real-time for real machines

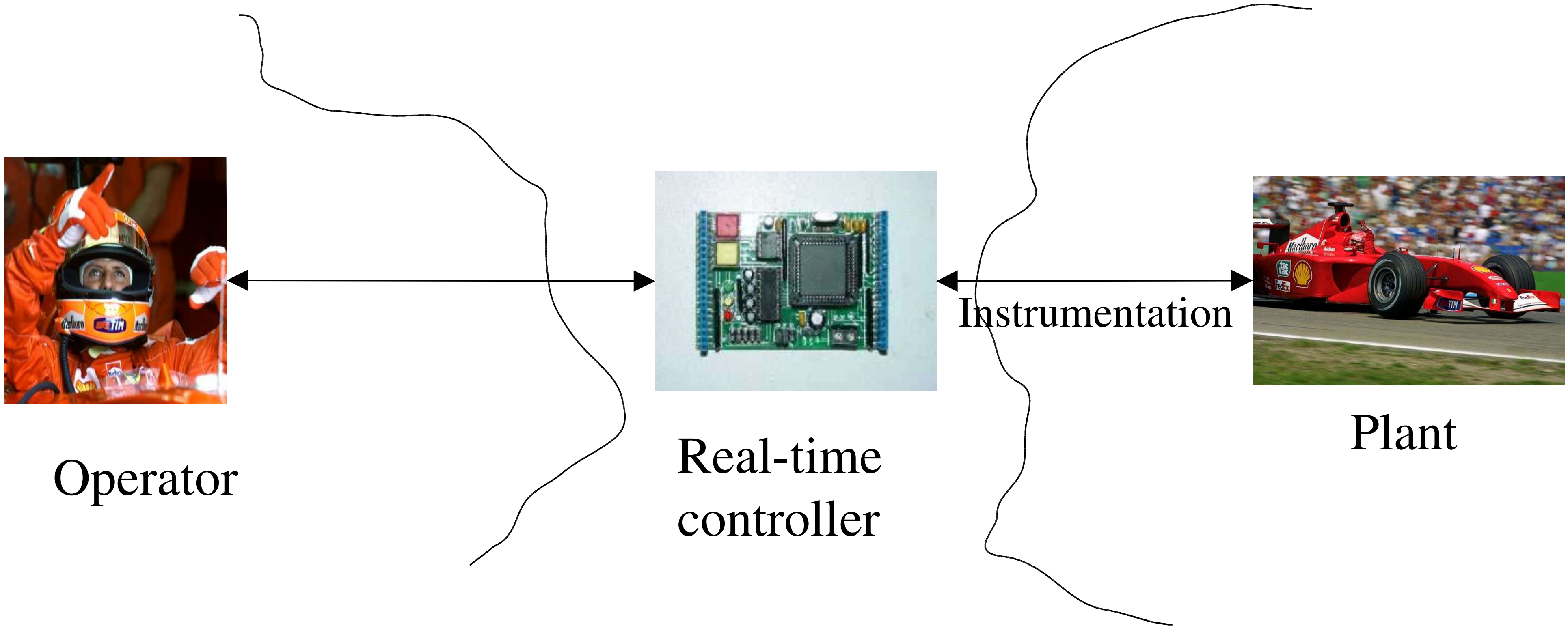
Luigi Palopoli

Bertinoro – 19/July/2003

Outline

- Basic definitions on real-time systems
- Development cycle
- Managing real-time concurrency
- Control/Scheduling co-design

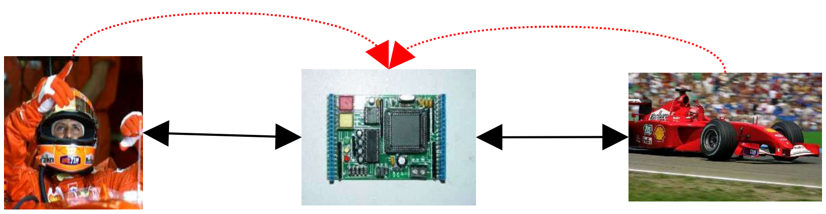
A real-time system



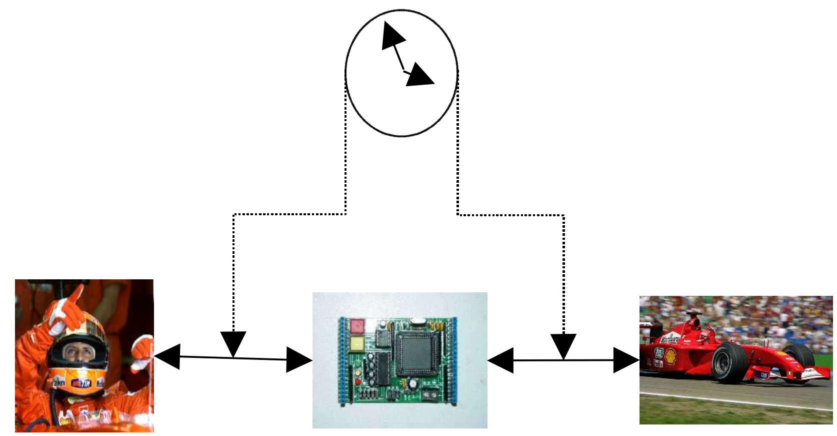
A real-time controller is a computer based system, which produces results to inputs complying with some temporal constraints

Some useful definitions...

- *Event triggered vs Time-triggered*



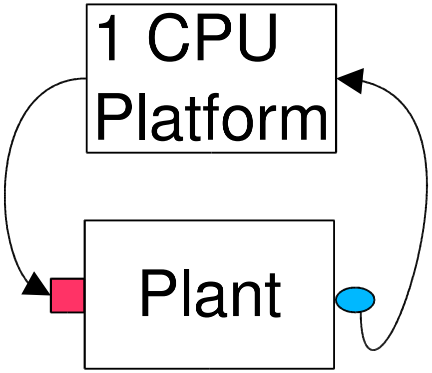
Event-triggered: system's reactions are elicited by the occurrence of certain events in the environment

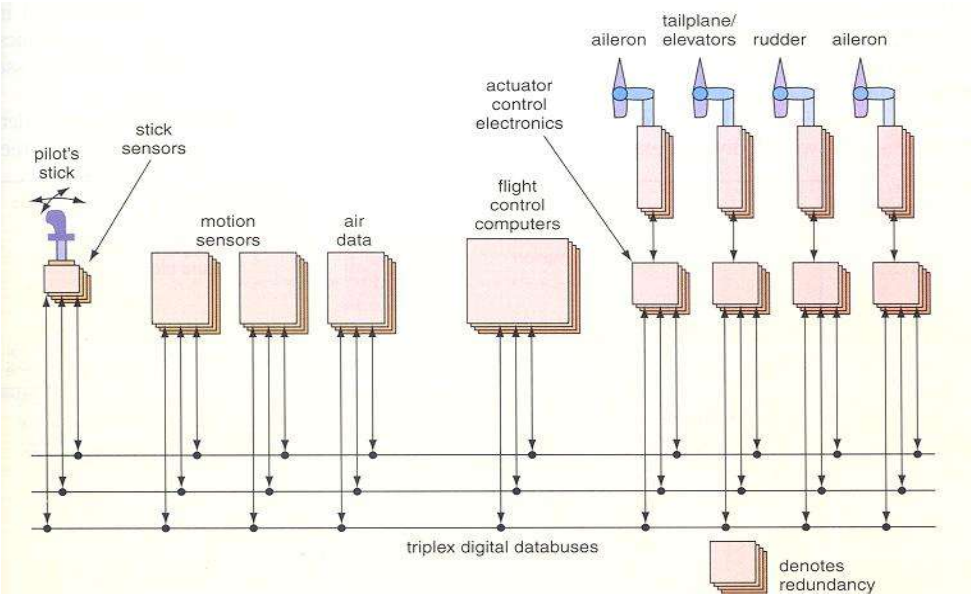


Time-triggered: interactions with the environment take place upon well defined instants

Some useful definitions I

- *Single node vs distributed*



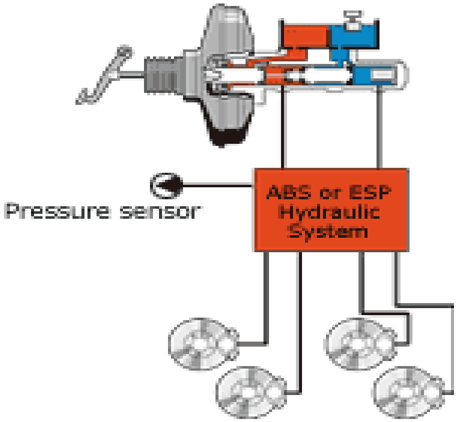


Single node: computation is concentrated in one node, which has direct access to sensors and actuators

Distributed: computation is distributed across different nodes which communicate by means of a bus

Some useful definitions ... II

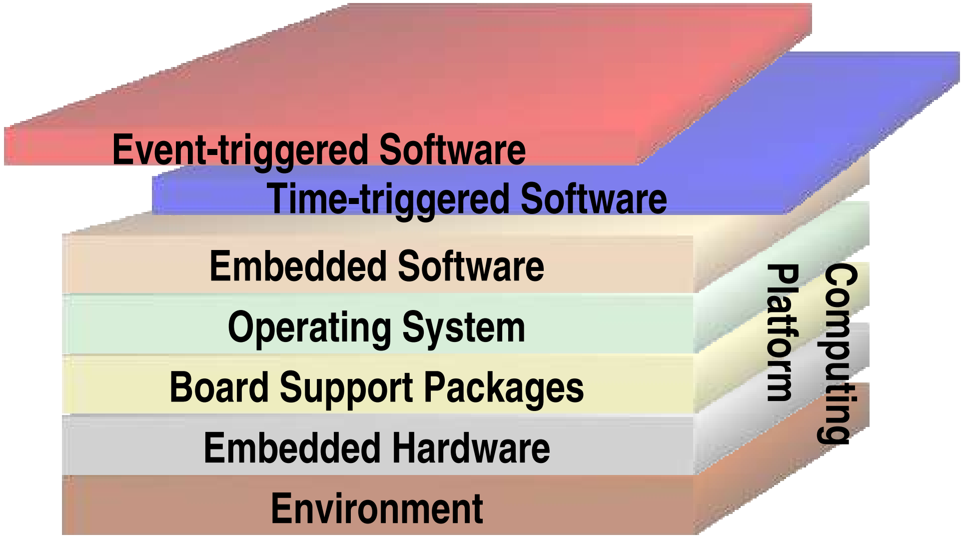
- *Hard real-time vs Soft real-time*



Hard real-time: computation must terminate within certain deadlines

Soft real-time: deadlines can occasionally be missed but the anomaly has to be kept in check, lest the Quality of Service be severely degraded

Real-time systems: what's in a name?



- A an embedded controller is a complex ensemble of software and hardware components:
  - Hardware devices
  - Software support components
  - Software applications
- Event-triggered and Time-triggered semantics are often intertwined

Do they work?

- Sometimes they don't!!!

Mars pathfinder: contact lost for 1.5 days due to a failure in real-time software

Ariane V explosion: 800 M€ lost due to a bug in software

Why should a control engineer care about real-time software?

- The pure “springs” of control engineer:
  - instantaneous computations and communication
  - infinite bandwidth links and nodes
  - ideal sampling
  - infinite precision

- The polluted “delta” of system engineer
  - communication and computation take a (random) time
  - links and nodes have finite bandwidth
  - there is sampling and actuation jitter
  - information is quantised
- Performance can be severely degraded

Outline

- Basic definitions on real-time systems
- Development cycle
- Managing real-time concurrency
- Control/Scheduling co-design

From design to deployment: the long run....

- The starting point is the outcome of classical digital control; something like:

$$u(k)=u(k-1)+K\left[\left(1+\frac{T}{T_I}+2\frac{T_D}{T}\right)e(k)-\left(1+2\frac{T_D}{T}\right)e(k-1)+\frac{T_D}{T}e(k-2)\right]$$

- Underlying assumptions:
  - samples are collected with period T
  - the new u(k) is emitted and applied to the plant as soon as the new e(k) arrives

From design to deployment: the long run ... I

- First step: generation of source code (typically in C)

```
typedef ... REAL
struct PID_DATA {
      struct {...} PARAMS;
      struct {...} STATE;
      struct  { REAL uc;     /*set point */
                REAL y;        /* measured variable */
                REAL u;       /* control variable */
      } SIGNALS;
};
void pid_init(struct PID_DATA * v) {...};
REAL pid_update(struct PID_DATA * v) {...};
```

What REALs do we work with?
- *Fixed point (fast computation, handle one order of magnitude)*
- *Floating point (slower computation, handle different orders of magnitude)*

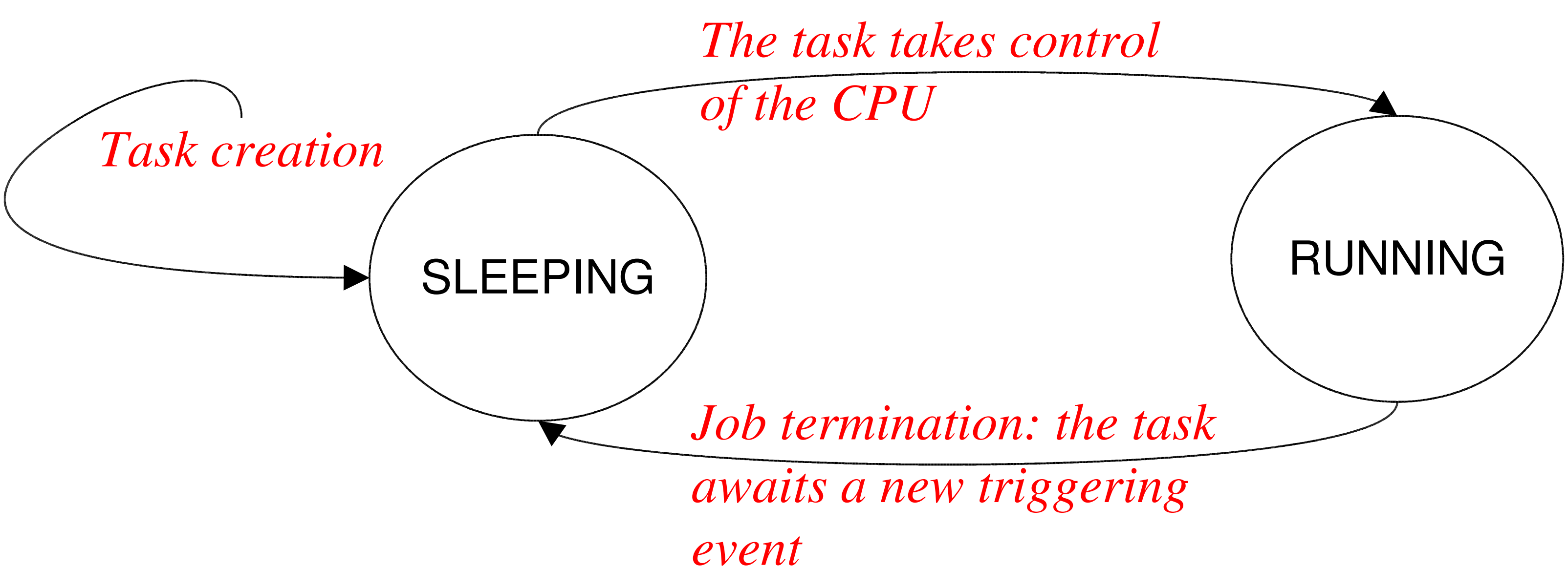
From design to deployment: the long run ... II

- II step: embedding of the function into a thread (or Task)

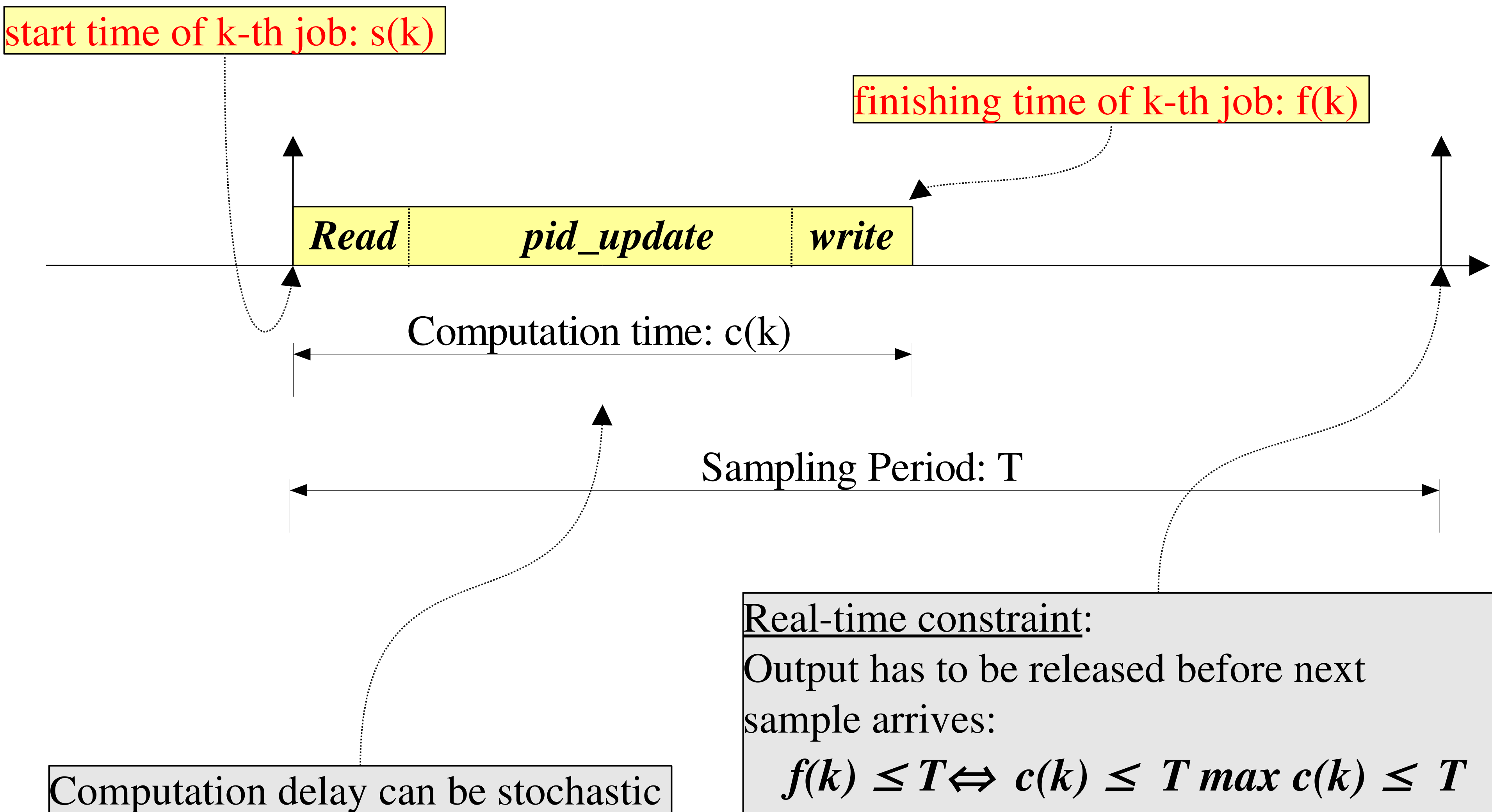
```
PID_DATA d;
THREAD PIDtask() {
      <fill in gains in d>
      pid_init(&d);
      while (1) {
            <wait for an event>
            readPort(a, &((d.SIGNALS).y));
            pid_update(&d);
            writePort(b,(d.SIGNALS).u);
      }
};
```

What is a task?

- *A task is a piece of code that, when triggered, executes a job on a processor*
- The event triggering a task's execution (*job*) can either be an alarm expiration (time-triggered paradigm) or an interrupt triggered by the arrival of new data (even-triggered paradigm)



Timing behaviour



Real-time constraint:
Output has to be released before next sample arrives:

$$f(k) \leq T \Leftrightarrow c(k) \leq T \max c(k) \leq T$$

Possible improvement

```
PID_DATA d;
THREAD PIDtask() {
     <fill in gains in d>
     pid_init(&d);
     while (1) {
          <wait for an event>
          readPort(a, &((d.SIGNALS).y));
          pid_output(&d);
          writePort(b, (d.SIGNALS).u);
          pid_update_state(&d);
     }
};
```

Emit new data as soon as possible and do the internal updates afterward

Breaking Pandora's Box!

- While a task is waiting for an event it does not need the processor's control: it is possible to execute other tasks!
- This is called *Concurrency*

> ***[The interrupt] was a great invention, but also a Pandora's Box. Essentially, for the sake of efficiency, concurrency [became] visible and then, all hell broke loose***
>
> ***[E.W.Dijkstra]***

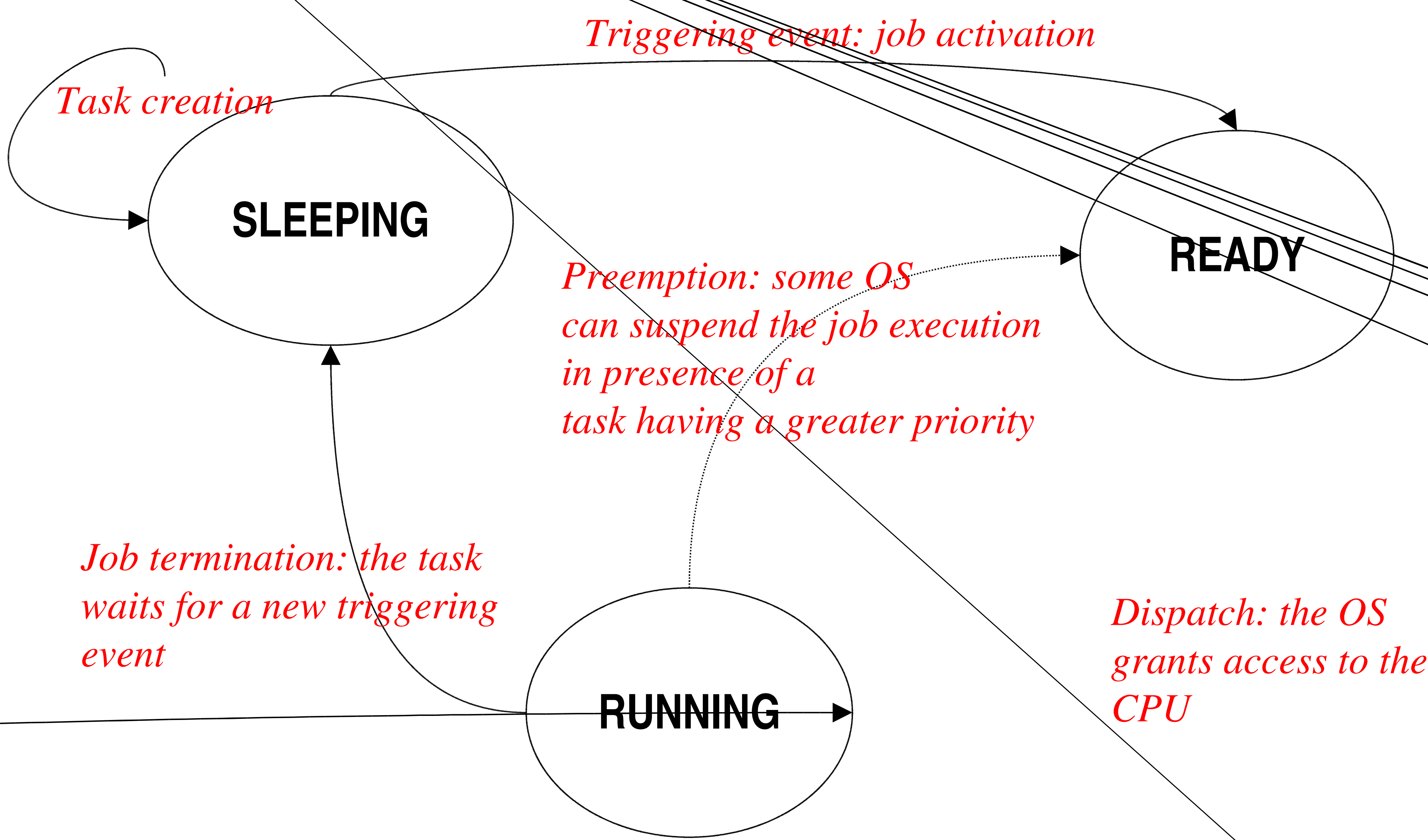
Outline

- Basic definitions on real-time systems
- Development cycle
- Managing real-time concurrency
- Control/Scheduling co-design

Managing concurrency

- The problem arises when multiple tasks are ready to execute at the same time
- A component of the OS (scheduler) is needed to grant the access to the CPU (or more generally to shared resources)
- A task can be in three states:
  - SLEEPING: it awaits the triggering event
  - READY: it requires the access to the CPU
  - RUNNING: it handles the CPU

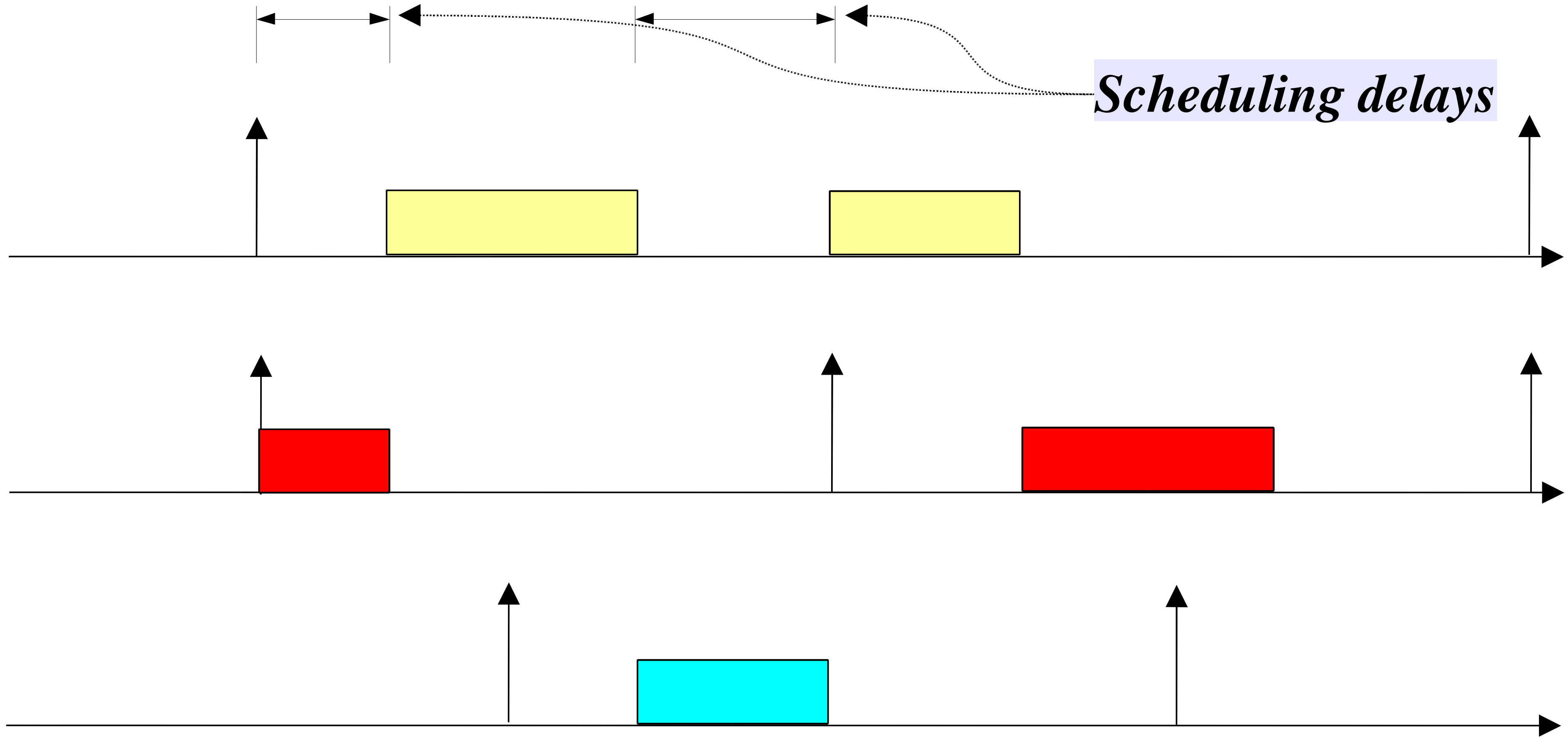
Task States


Triggering event: job activation
Task creation
SLEEPING
READY
Preemption: some OS
can suspend the job execution
in presence of a
task having a greater priority
Job termination: the task
waits for a new triggering
event
RUNNING
Dispatch: the OS
grants access to the
CPU

What's wrong with concurrency?

- From computer engineering point of view, it becomes much more difficult to write and debug programs (especially if tasks interact)
  - *deadlock*
  - *livelock*
  - *starvation*
- From control engineering point of view, it becomes more difficult
  - to enforce timing constraints
  - to ensure regularly spaced sampling
  - to ensure regularly spaced command release

Timing Behaviour



Problems with concurrency

- Delays introduced by scheduling
  - Sampling intervals irregular
  - Output release intervals irregular
- Real-time constraints
  - How is it possible to ensure that every job finishes in due time?

Is that all?

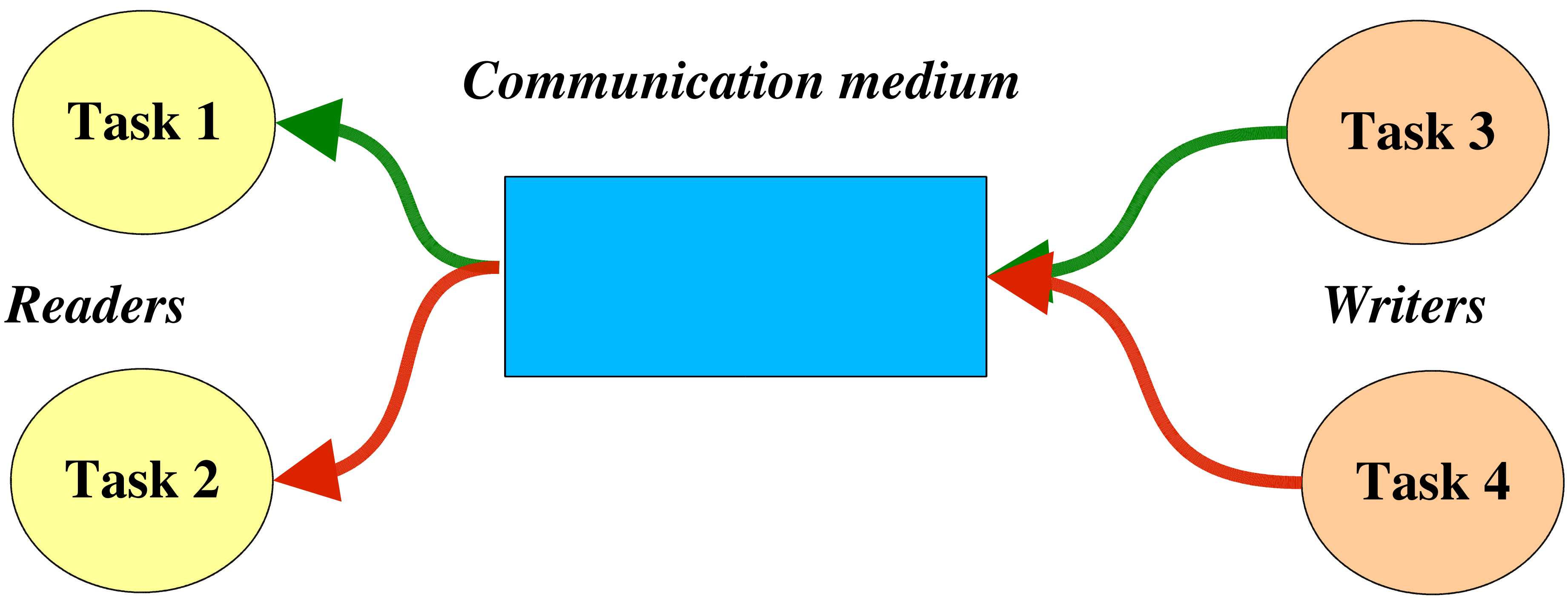
- Well it may not be!!

```
PID_DATA d;
THREAD PIDtask() {
     <fill in gains in d>
     pid_init(&d);
     while (1) {
          <wait for an event>
          readPort(a, &((d.SIGNALS).y));
          pid_update(&d);
          writePort(b,(d.SIGNALS).u);
     }
};
```

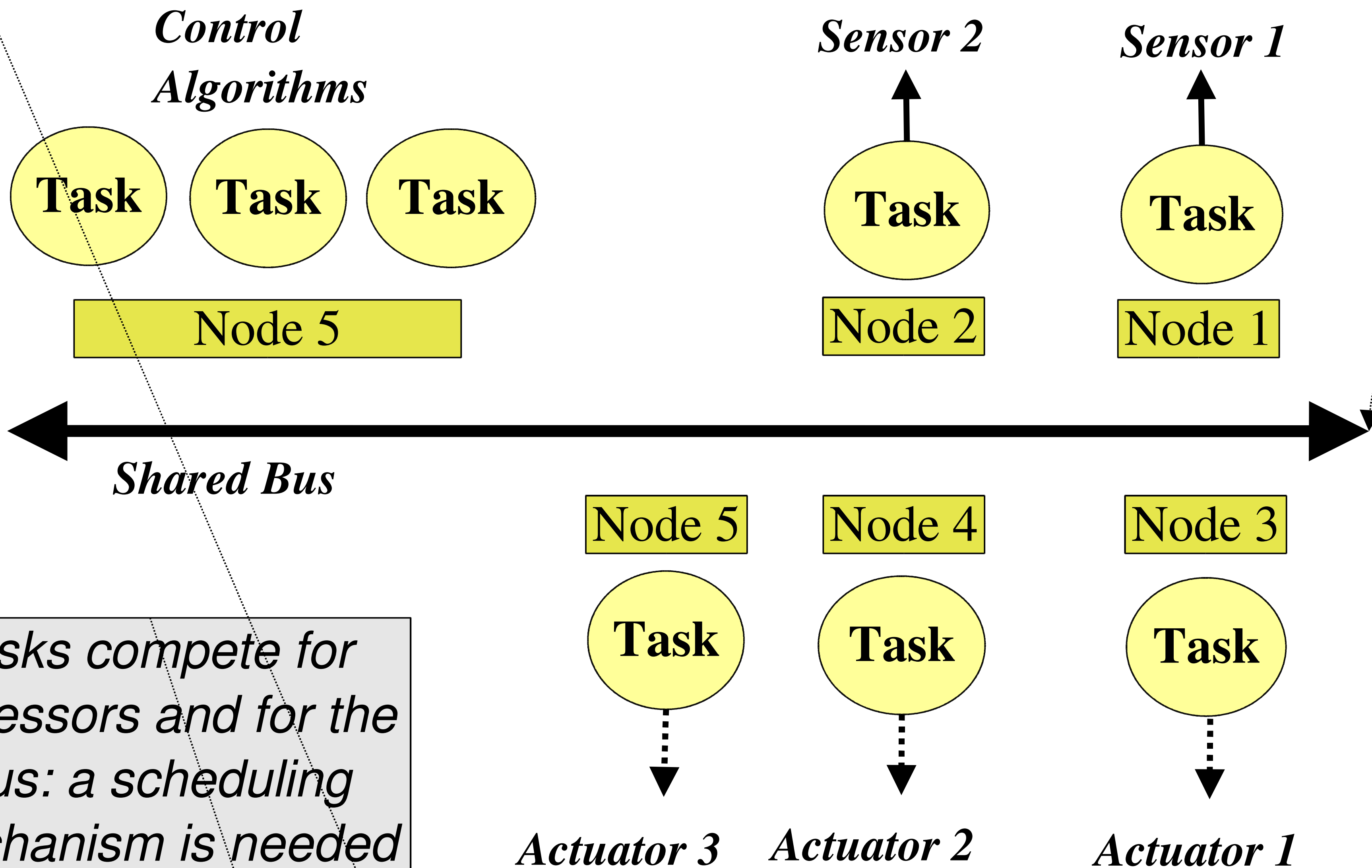
Tasks communicate with other tasks and with the environment!

Communication

- Communication takes time and it entails resource sharing



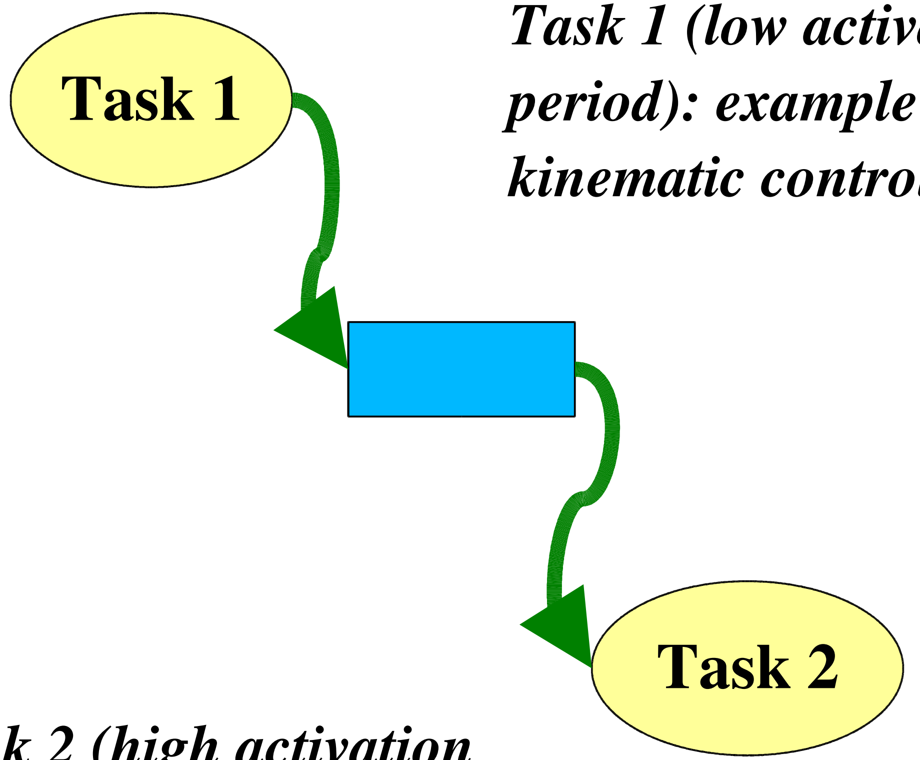
Example 1: distributed control



Tasks compete for processors and for the Bus: a scheduling mechanism is needed

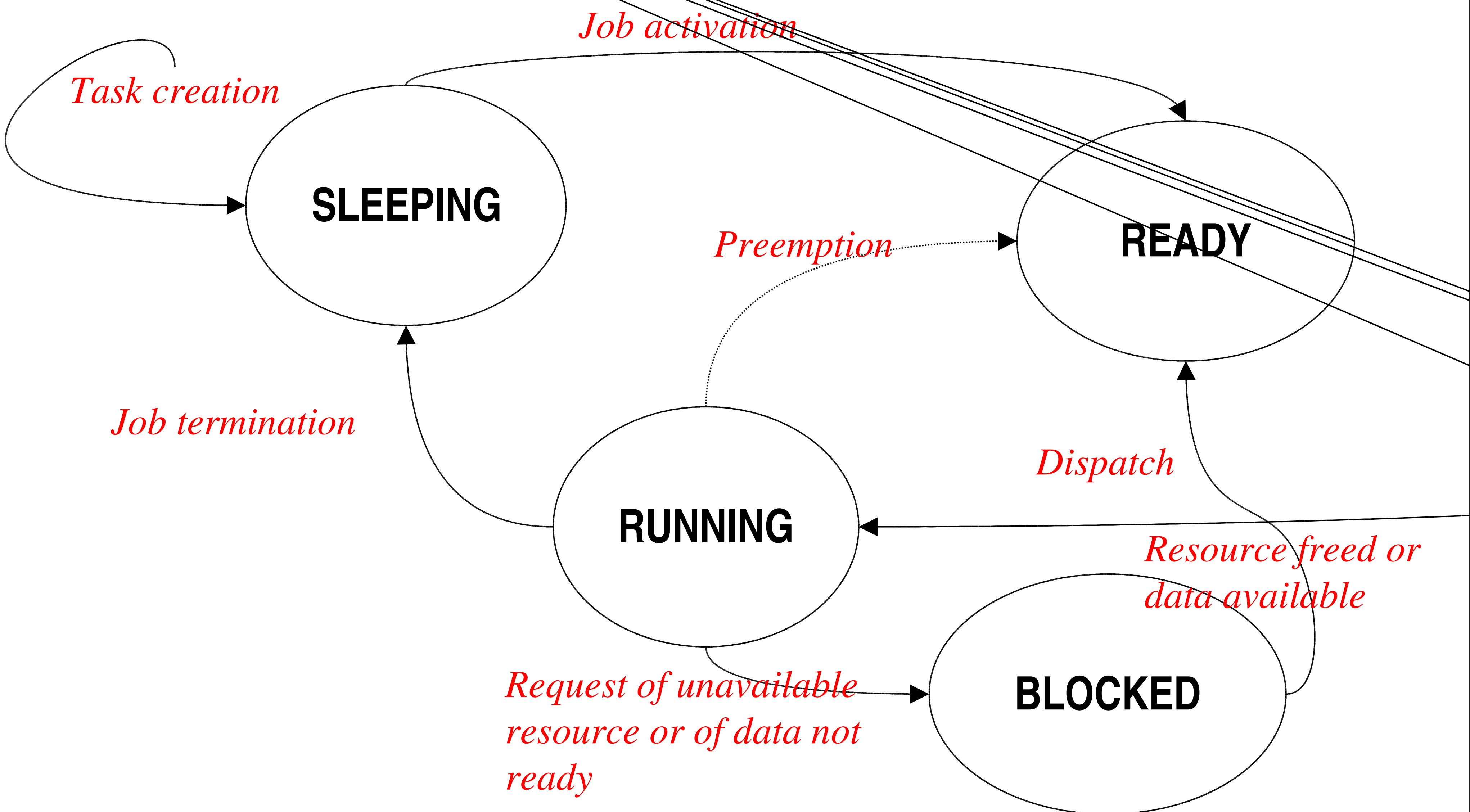
Example 2: Multilevel control

Task 1 (low activation period): example kinematic control



Task 2 (high activation period): example motor PID

A new state for the task



Real-time scheduling

Given a set of task, endowed with execution timing constraint, and a set of shared resources, decide an allocation of resources to tasks (for each time instant) such that timing constraints are met