RELIABLE SOFTWARE TECHNOLOGIES
ADA-EUROPE 2010

Control Co-design: Algorithms and their Implementation

P. Albertos, A. Crespo, J. Simó
Dpt. Ing. de Sistemas y Automática
Instituto de Automática e Informática Industrial
Universidad Politécnica de Valencia
pedro@aii.upv.es

A. Fernández
Dpt. Automática y Computación
CUJAE, Habana. Cuba
adel@electrica.cujae.edu.cu

VALENCIA, SPAIN, 14-18 JUNE 2010

Thanks to:

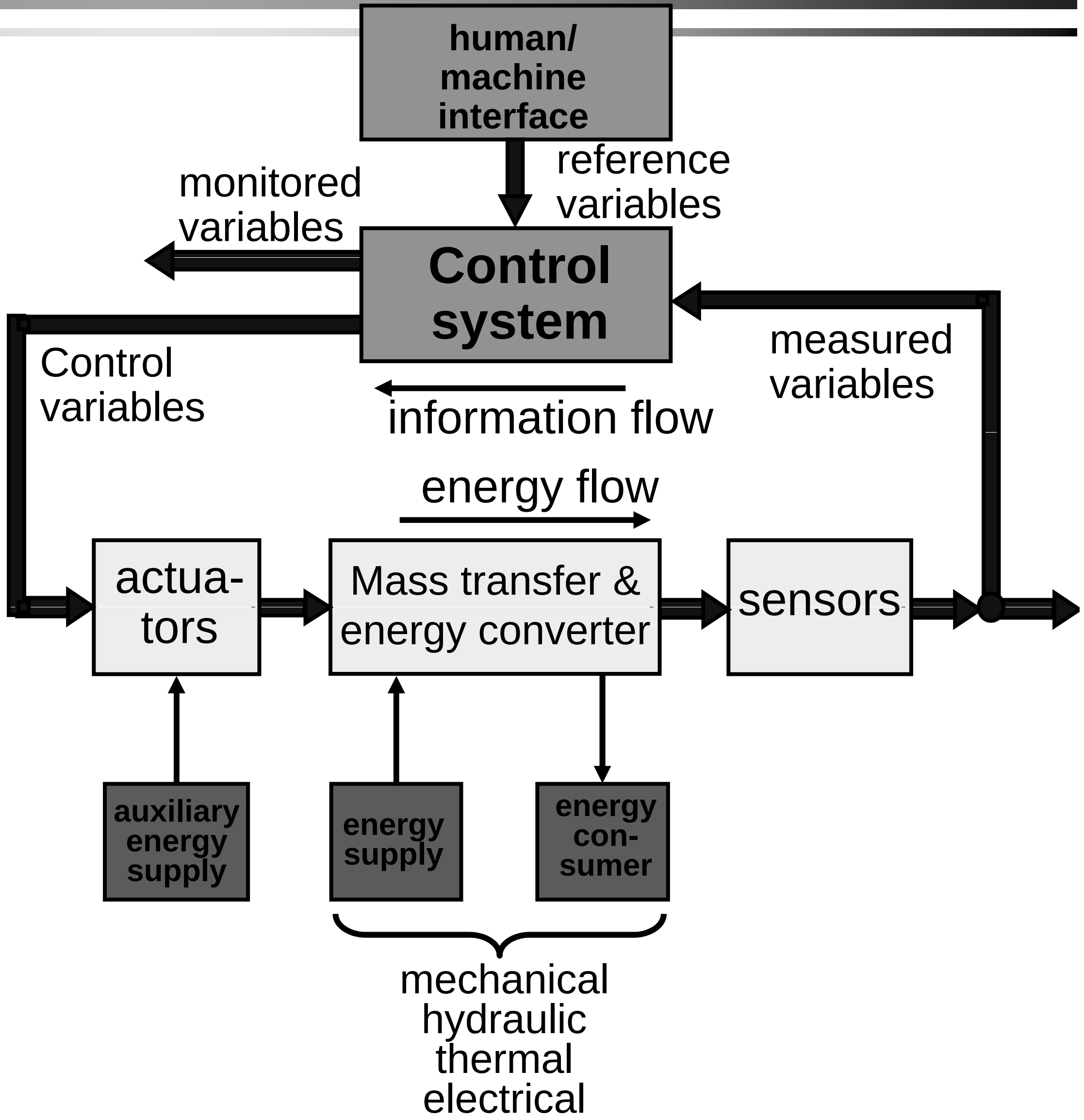
- My Co-authors:
  Alfons Crespo
  José Simó
  Adel Fernández

- My Co-workers :
  Karl-Erik Årzén
  Anton Cervin

  Marina Vallés
  Angel Valera
  Raúl Simarro
  Patricia Balbastre
  Pedro García

Motivation

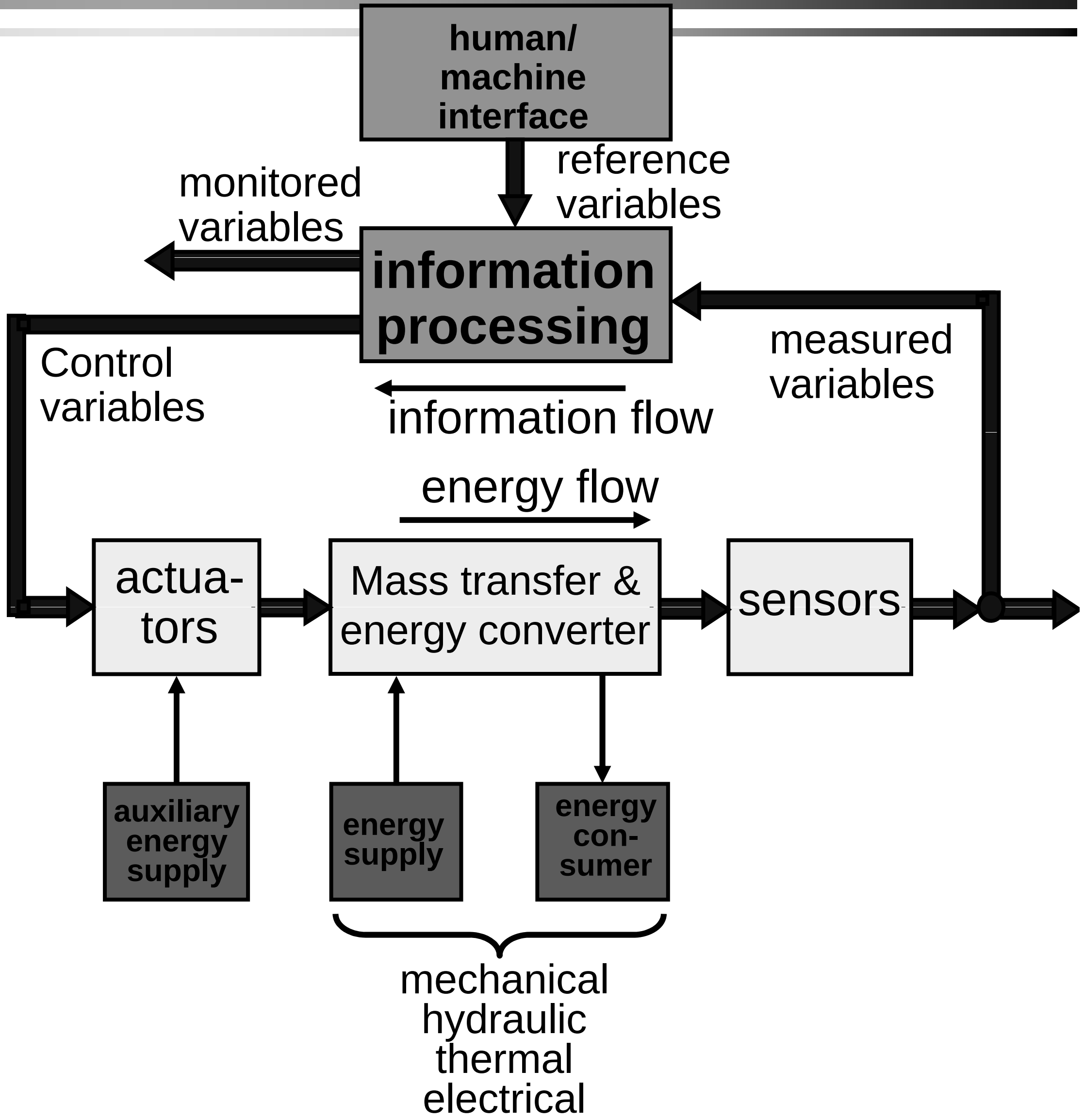
❖ Classical control loop



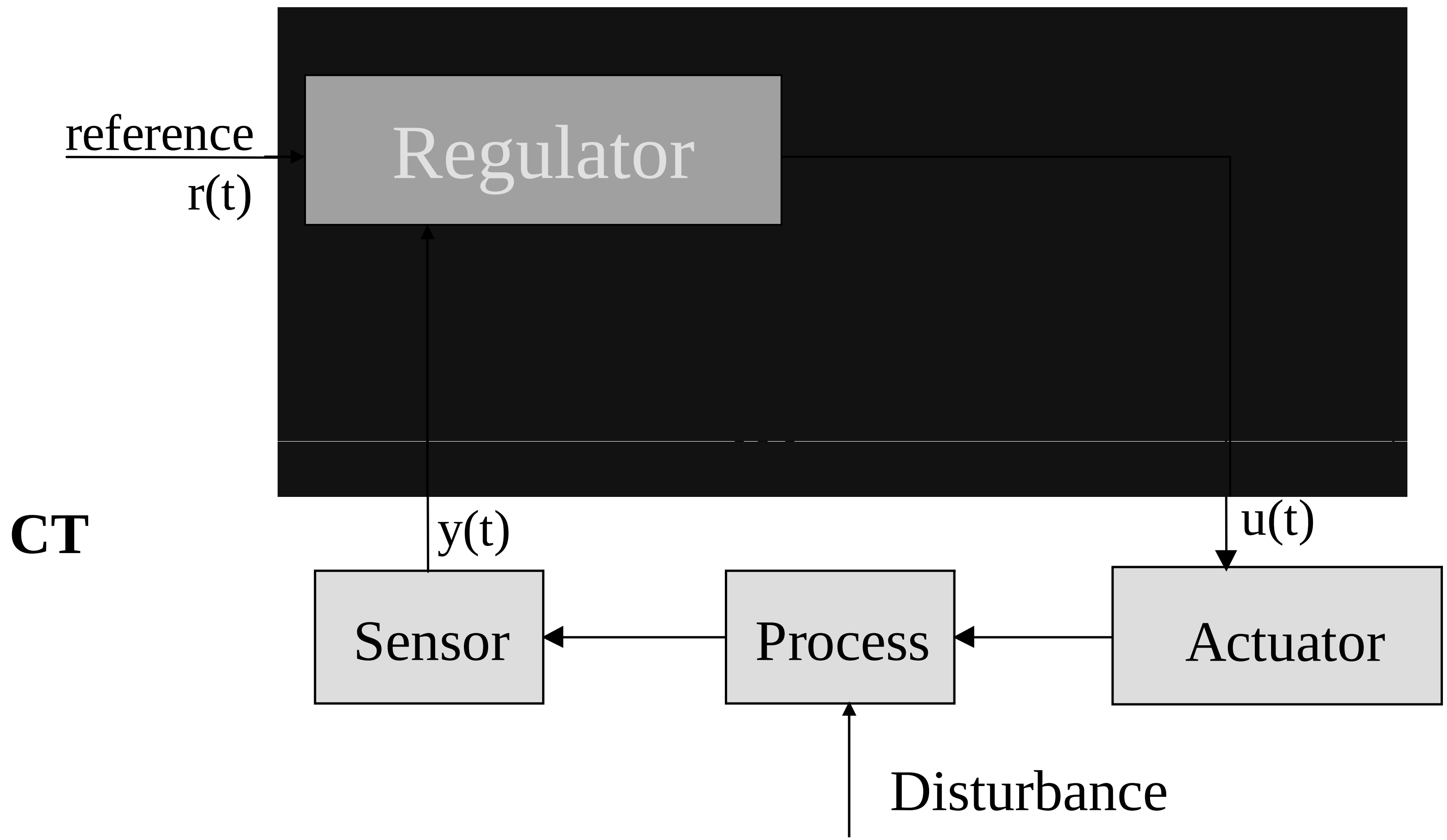
Motivation

❖ Classical control loop

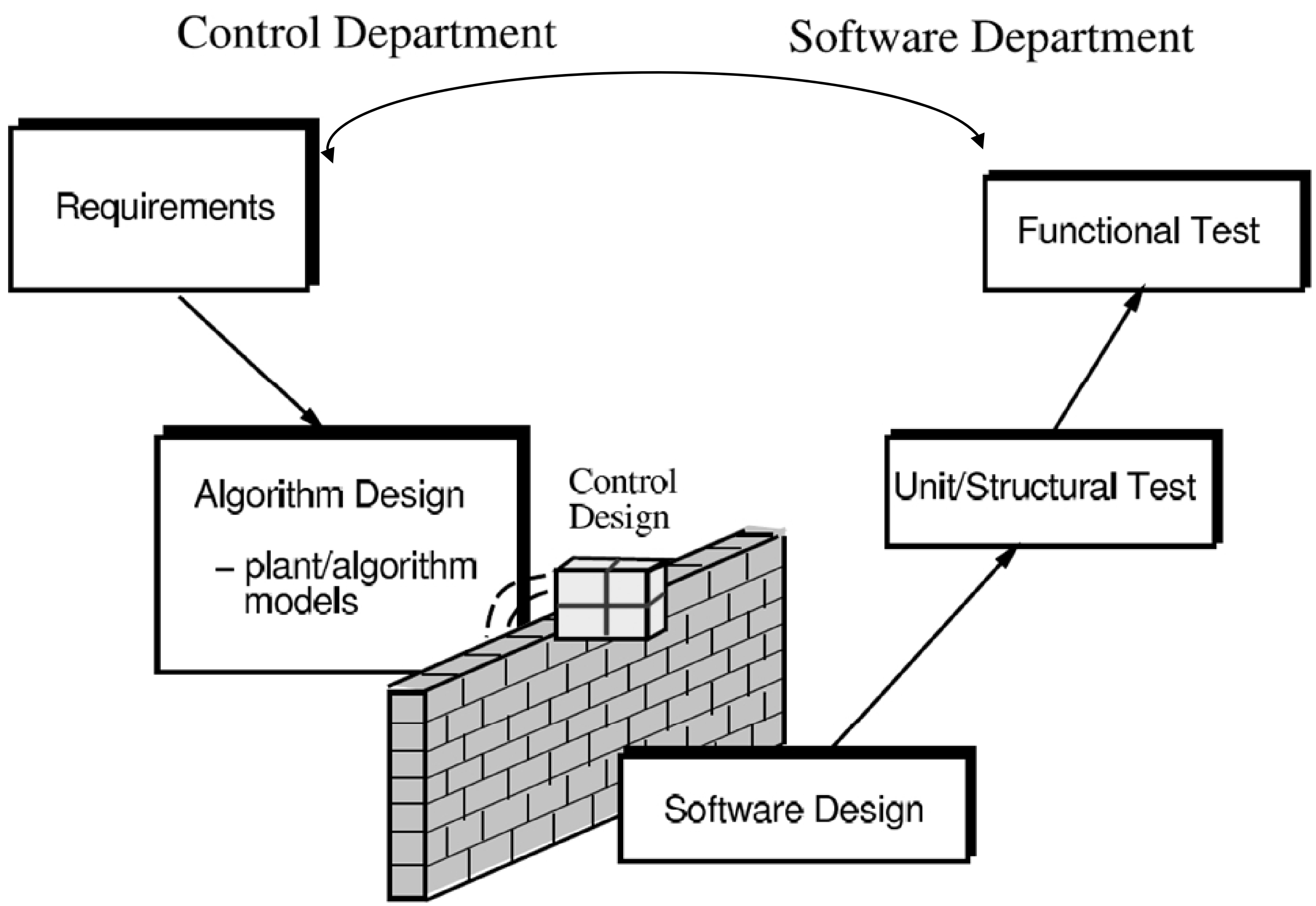
❖ Basic DT Controller



Basic Control Loop

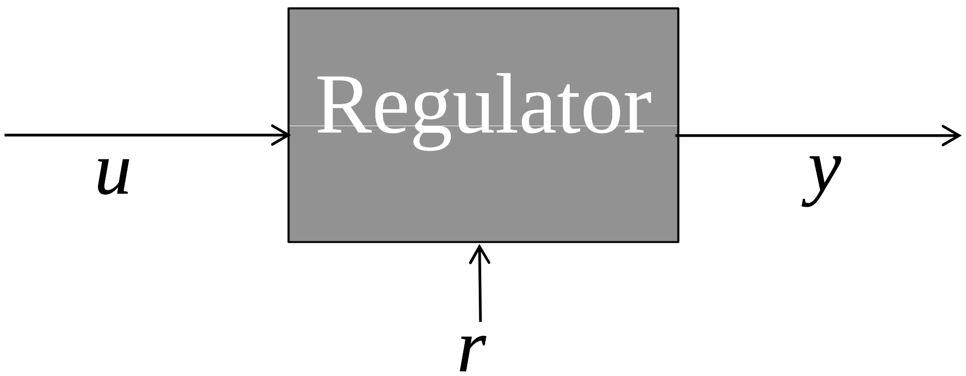


Control System Development Today

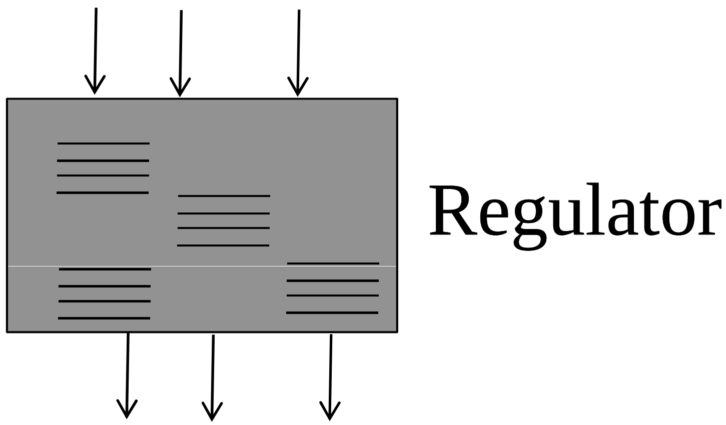


Regulator

- Control viewpoint:
  - Signal processor
  - Dynamic behavior
  - Process interaction



- Computer viewpoint:
  - Set of tasks
  - Resource allocation
  - RT constraints



Problems

- The control engineer does not care about implementations
  - **"trivial"**
  - **"buy a faster computer"**
- The software engineer does not understand controller timing
  - **"τ_i = (T_i, D_i, C_i)"**
  - **"hard deadlines"**
- Control theory and real-time scheduling theory have evolved as separate subjects for thirty years

Outline

- Introduction: Basic Digital Control
  - Main assumptions, Main concerns
- Computing requirements for control applications
- Control requirements for RT implementation
- New control scenarios
  - Embedded, networked, event-driven
- Algorithmic issues
  - Asynchronous sampling, Delays, Control effort
  - Hybrid systems
- Implementation issues
  - Control Kernel
- Conclusions

Introduction

- Digital control comes from SD Control
  - Discretization of CT controllers
- Control is applied to CT plants
  - Plants are discretized for control design purposes
- DC implies plant in open loop between samplings
- DC is implemented in Computers
  - Controller is no more one single device
  - Serial operation – Tasks conflicts
  - Sharing resources
- Control design ←→ implementation

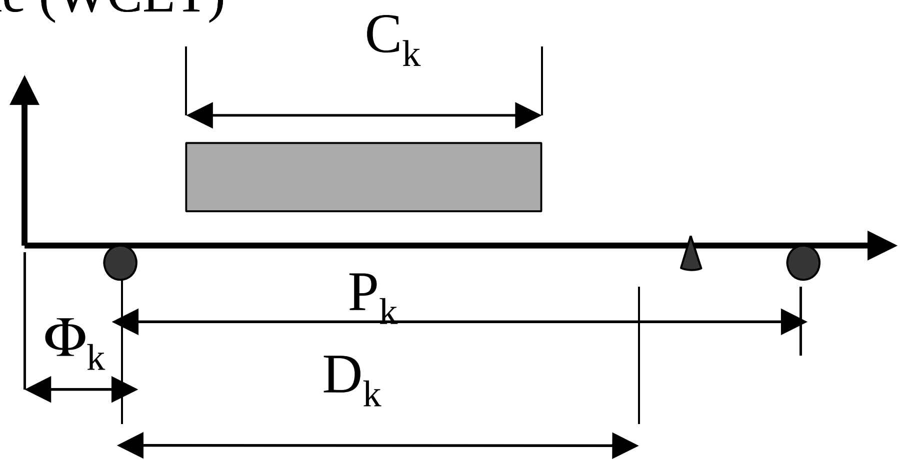
Basic Digital Control

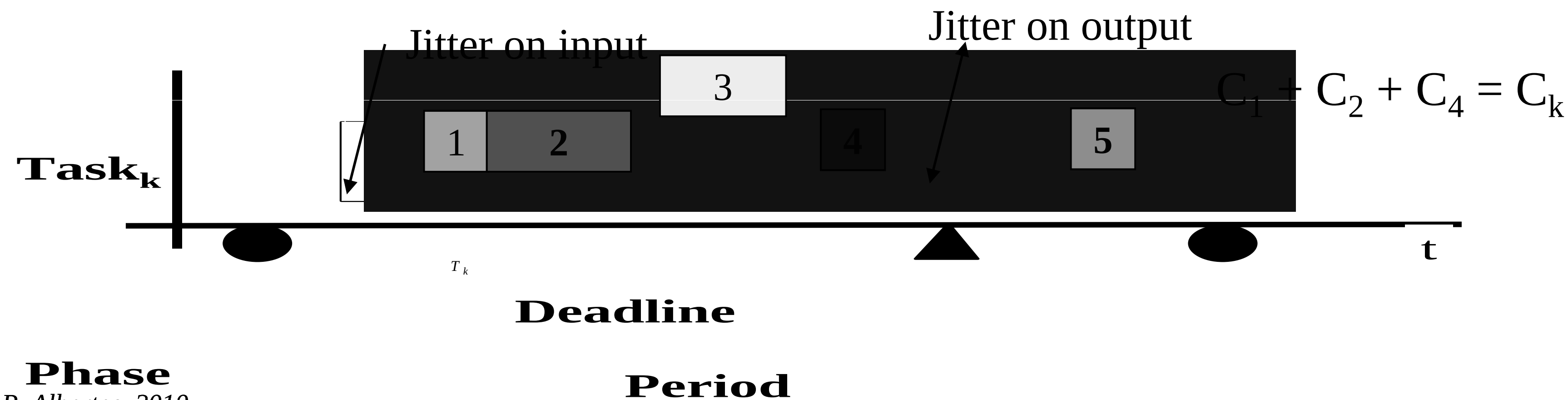
- ❖ The plant is without control between sampling/updating
- ❖ Sampling and updating should be as fast as possible
- ❖ Control is computed (updated) periodically
- ❖ Control task
  - ❖ parameters are stored in the memory
  - ❖ control law sequence is cyclic
  - ❖ control law is validated in CT operation
- ❖ Past data are accessible
- ❖ Communication channels are continuously operating

DT controller emulates CT controller

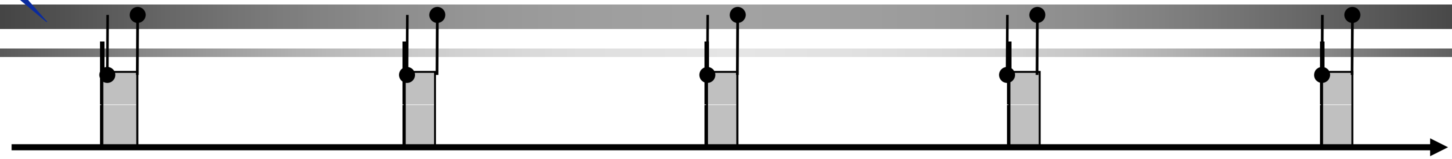
Real Time Task model

- A task (T_k) is defined by four parameters:
  - C_k: Worst Case Execution Time (WCET)
  - D_k: Deadline
  - P_k: Period
  - Φ_k: Phase





Control Loop Timing



- Classical control assumes deterministic sampling
  - in most cases periodic
- Classical control assumes negligible or constant input-output latency (from sampling to actuation)
  - if the latency is small compared to the sampling interval it can be ignored
  - if the latency is constant it can be included in the control design
  - too long latency or too much latency jitter give poor performance or instability
- Not always possible to achieve with limited computing resources that are shared with other applications

Sampling rate selection

Regular Sampling: Influences and is determined by desired closed loop performance

Fast sampling converges to CT but …

Computation load increases, Numerical errors

Same sampling rate for all processes?

Multirate controllers

Control computation is not required anytime:

Event-based control, Hybrid control

Classical control assumes negligible or constant input-output delays

small delays can be ignored, included in the control design

difficult to predict the required time (WCET)

WARNINGS

long sampling interval or delay may cause poor performance or instability …

Resources infra-use

Main Concerns in Computer Control

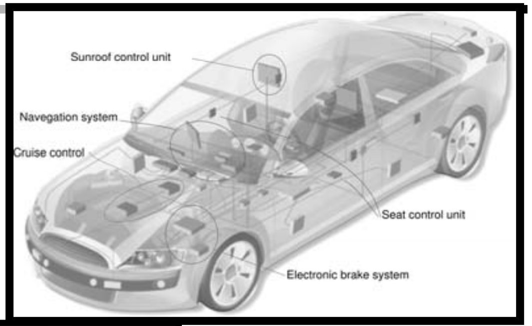
- Unavoidable delays between sampling & updating
- Sampling period may be changed
- Signal transmission may be delayed (or missing)
- Time sequencing may depend on other tasks
- Additional tasks may change the allocated resources: computation time, memory, data access

as a result →

- Non conventional sampling/updating pattern
- Delays and missing data
- Modes and sampling rate changes (alternatives)
- Event-based control

Applications

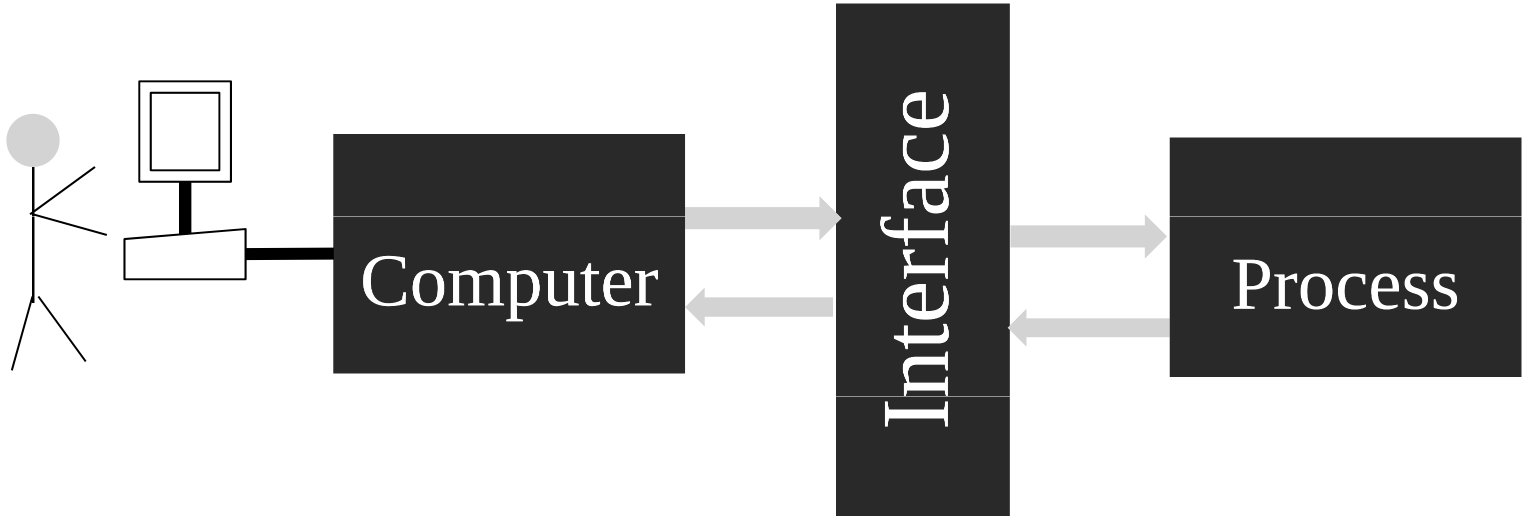
- Automotive Industry
  - Many, distributed



- Aerospace and Fly Control
  - Safety, reliability

- Industrial Processes
  - Many many, cost-effective

Computer-based control



Control loop implementation

Specification

```
task Level_Control (initial_time, period, phase: TIME);

task body Level_Control is
level : Sensor_Value;
action : Actuator_Value;
reg : Regulator;
next_period : TIME;       -- period  task attribute

begin
Define_Regulator(reg, par1, par2, ...);
```

Computation

```
....
delay until (initial_time + phase);
next_period := initial_time + phase;
loop
   level := get_level_sensor();
   Regulator_evaluate(reg, level, action);
   -- operations to improve the regulator results
   send_actuator(action);
   -- operations to evaluate the global state
   --operations to prepare the data for the next sampling
   -- operations of updating the global data base
   delay until next_period;
   next_period := next_period + period;
end loop;
...
end Level_Task;
```

Computer Control Task


```
Initialization
…
loop
    convert _sensor _analog_ digital (y); read reference (r);
    compute _control _action (u);
        compute _error (e)
        compute _control _action (u)  ←
    send _ converted _ control_ action (u);
    update_internal_variables (y,u, …);
    Next _Iteration:= Next  _Iteration + Period;
    delay until Next _Iteration;
end loop;
…
```

Computing Requirements for Control Applications

- To have a quick an secure dispatch of a control action
- To get a basic “picture” of the current situation
- To compute a simple and fast control action to be improved if resources are available
- To switch to the appropriate control mode, based on the resources availability
- To reconfigure under detected faults

Some RT Options

- Sub-task models
  - Sample & deliver, compute, refine
- Imprecise task model
  - mandatory + optional part
  - anytime algorithms or multiple versions
  - only applicable to control in special cases
- Alternatives for deadline overruns?
  - Continue with the computations of the controller job
  - Abort the job (lost sample → doubled samp. interval)
  - Postpone remaining computations until the beginning of the next sample (increased latency)

New Trends in Control Implementation

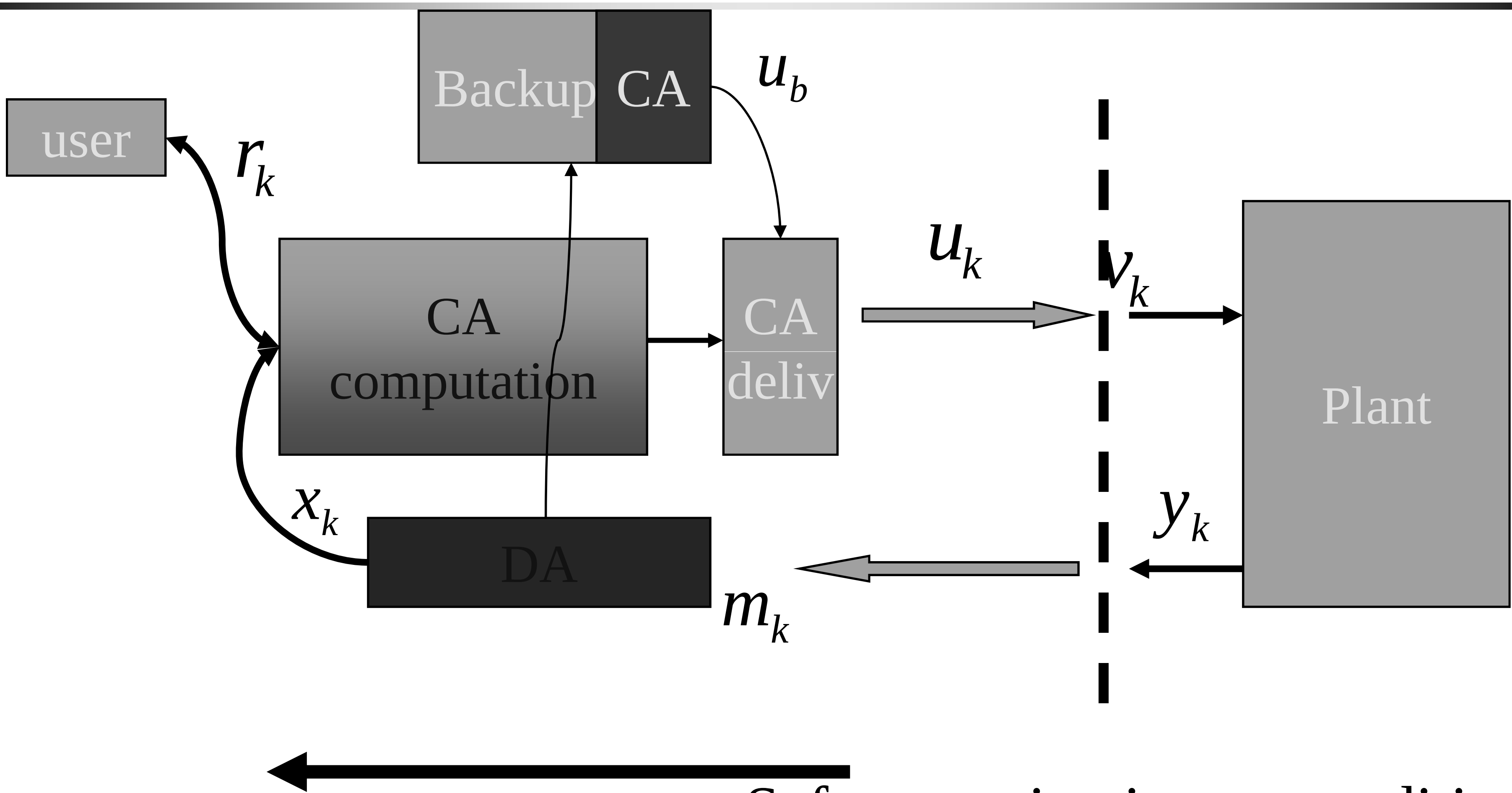
- RT Operating System → Secure Systems and isolation. Hypervisor System; → Dynamic load management
- RT Middleware
  - Communication Systems → QoS management according to the QoC observed (Quality of Control)
    - DDS
  - Virtual machines
  - Objects and Agents

  (Virtual machines, Objects and Agents) → Distributed execution environments. Physical Metaphor
- Control Middleware → Environment where control services are provided. Control Metaphor
  - Local, emergency, switching

Control requirements for RT implementations: Basic Assumptions

- The Data Acquisition system provides the required data
- The actuators' drivers deliver the control actions
- The CPU is fully available for the control task
- The CPU computes on-time (no errors) the control action
- The required data are stored in the memory
- The sampling pattern is regular (constant, synchronous and uniform for any control task)
- The control algorithm is well defined
- Alternative controllers are independent
- Power supply is guaranteed

Control activities' priority



Safe operation in any condition

Control activities' priority

- Ensure control action (CA) delivering
  - Safe (back-up) CA computation
  - Safe CA computation based on previous data
- Data acquisition of major signals
  - Safe CA computation based on current data
- Transfer to new control structure
  - Basic control structure parameters computation
  - CA computation

- Full DA
  - Control structures evaluation and selection
  - CA computation (different levels)
- Communication facilities
- Coordination facilities

Highest priority

- Control action delivering
- Detect missing data
- Evaluate control performance
- Evaluate alternative control options
- Determine the best CA
- Detect faulty conditions
- Change the operating mode
- Compute back-up signals

Influence on Control Performance

Scheduling & Network Parameters (T,D,Prio, Protocol,…) → Task Timing Parameters (latencies, jitter, …) → Control Performance (variance, rise time, overshoot, ….)

Complex relationship   Complex relationship

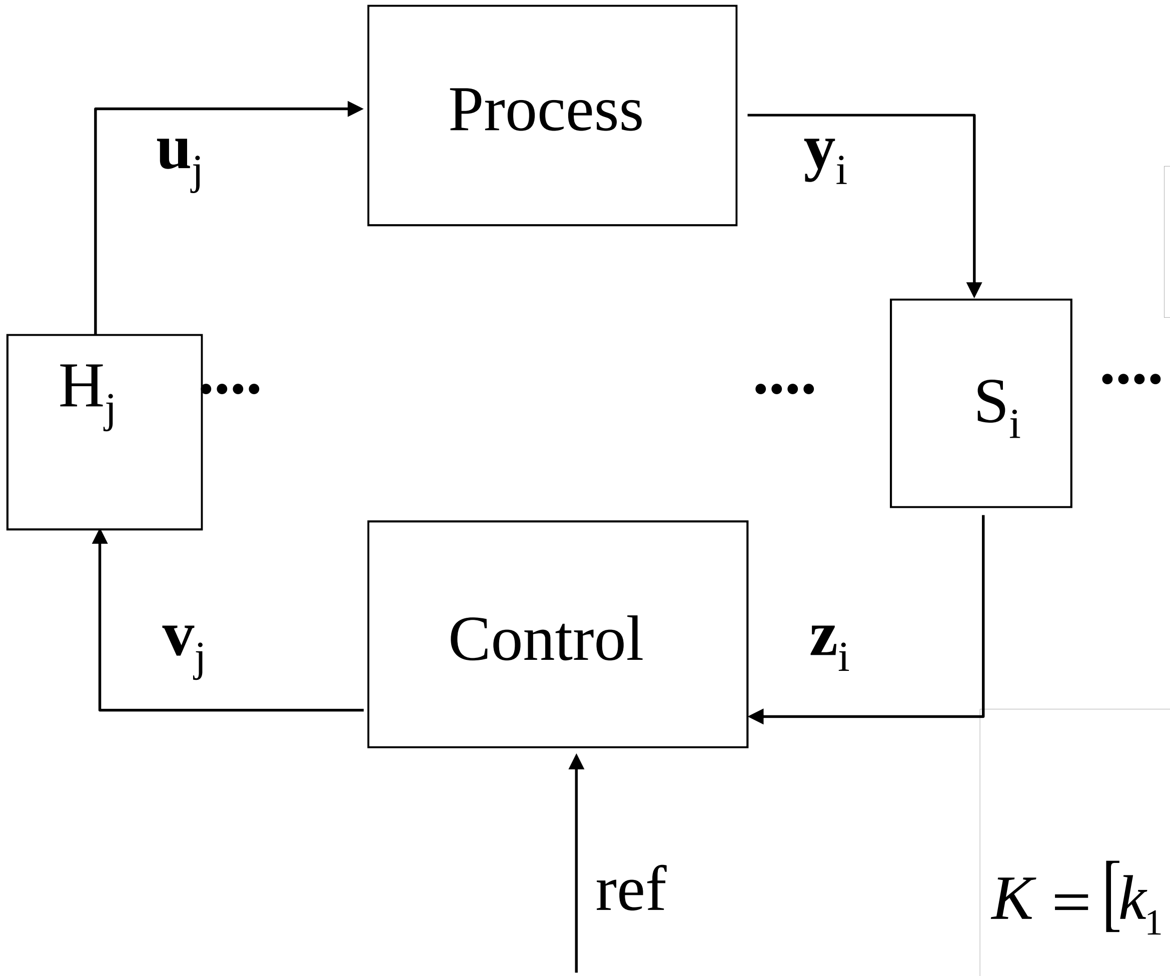
- In general:
  - sampling jitter has a negative effect on performance
  - a short latency is better than a long latency
  - latency jitter is bad, but a short jittery latency is in most cases better that a longer constant latency, also if the latter is compensated for
- However, anomalies exists
  - sampling jitter may improve performance
  - latency can sometimes have a stabilizing effect
  - a shorter varying latency can be worse than a longer constant latency

Control Effort Concept

Performance degrading

- The maximum allowable time delay is given by the phase margin, derived from the frequency analysis of the open loop output feedback controlled system
- The *Control Effort*, defined as the shift in damping from the open loop poles to the closed loop poles, provides a useful way to obtain the maximum allowable time delay, for both, continuous and discrete systems.
- The longer the sampling period T is, the more sensitive to the time delay the design is.

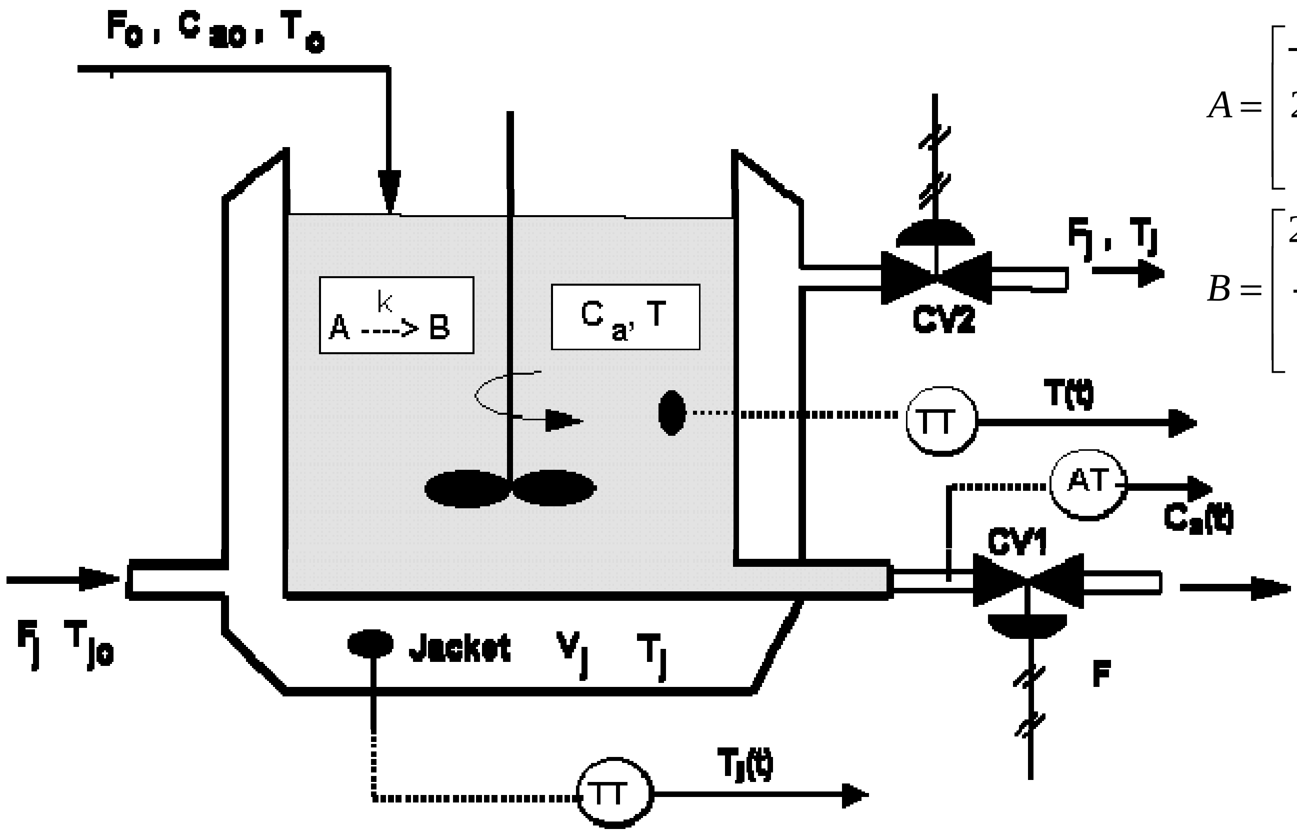
MIMO controlled plant:Signals relevance



$$u(t) = Kx(t)$$

$$K = \begin{bmatrix} k_1 & k_2 & \cdots & k_n \end{bmatrix} = \begin{bmatrix} {}_1k \\ {}_2k \\ \vdots \\ {}_mk \end{bmatrix}$$

Reactor: model

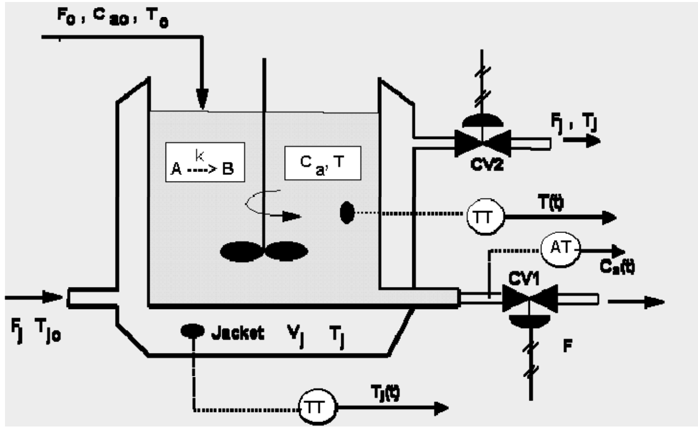


$$A = \begin{bmatrix} -1.705 & -0.2519 & 0 \\ 23.088 & -28.71 & 20.9 \\ 0 & 200.3 & -216.89 \end{bmatrix};$$

$$B = \begin{bmatrix} 2.918 & 0 \\ -28.6 & 0 \\ 0 & -415.29 \end{bmatrix}$$

$$x = \begin{bmatrix} C_a & T & T_j \end{bmatrix}^T; \quad u = \begin{bmatrix} F & F_j \end{bmatrix}^T$$

Reactor: control



$$\{a_i\} = \{eig(A)\} = \{-2.5878, -7.73, -236.987\} \quad A_n = -247.3$$

Control Goal: $p = \{-320, -340, -360\}$;

$P_n = -1020$

$$K = \begin{bmatrix} 858.5 & 68.5676 & 4.6683 \\ -40.463 & -4.2238 & -0.5505 \end{bmatrix}$$

Assume F active and F_j open

$$\{eig(A - b_{1\cdot 1} k)\} = \{-405.7, -336.1, -49.5\}; \quad \rightarrow S_2 = -791.4$$

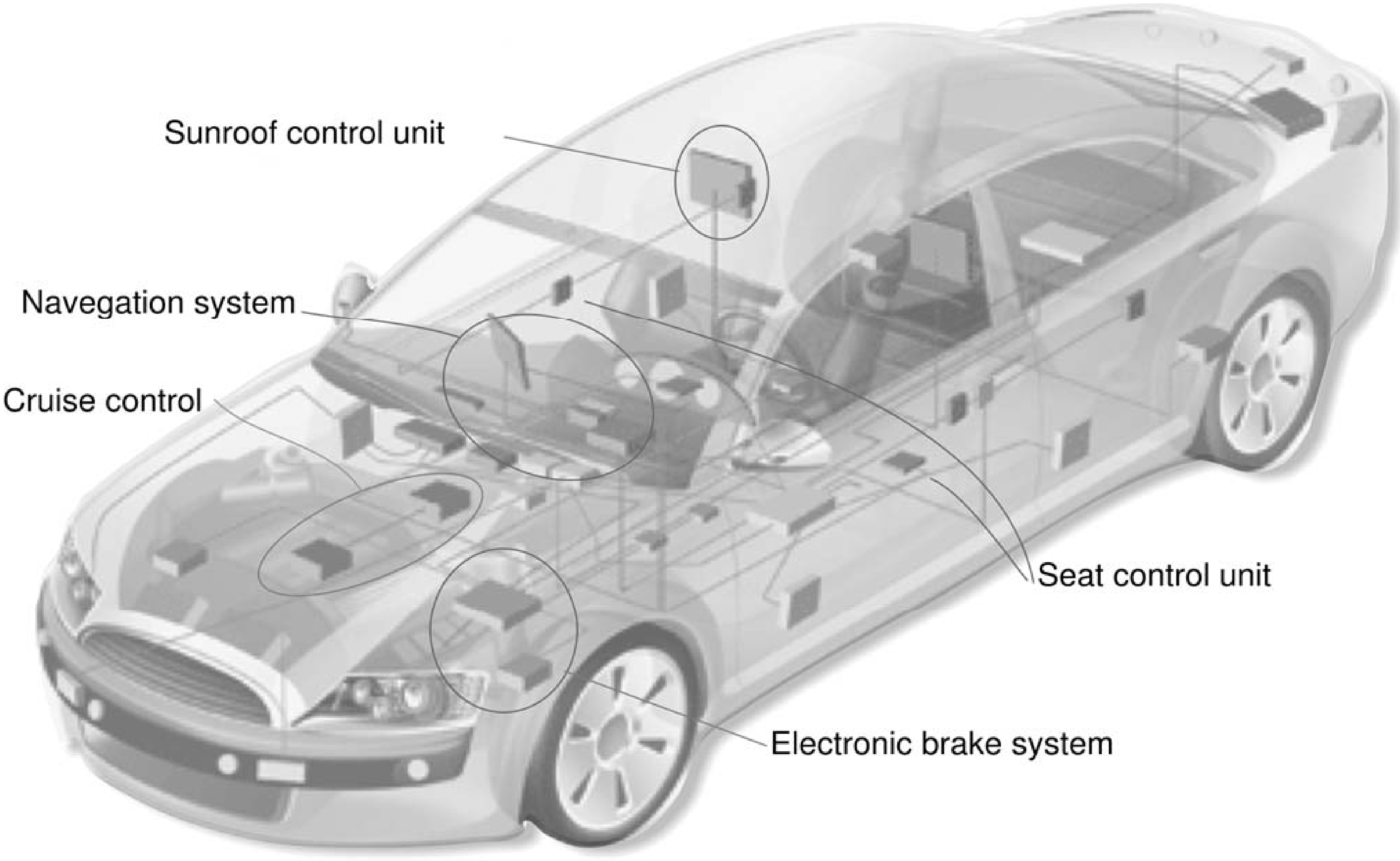
$$S_2 - P_n = 228.7$$

Give highest priority to most degrading signal failure

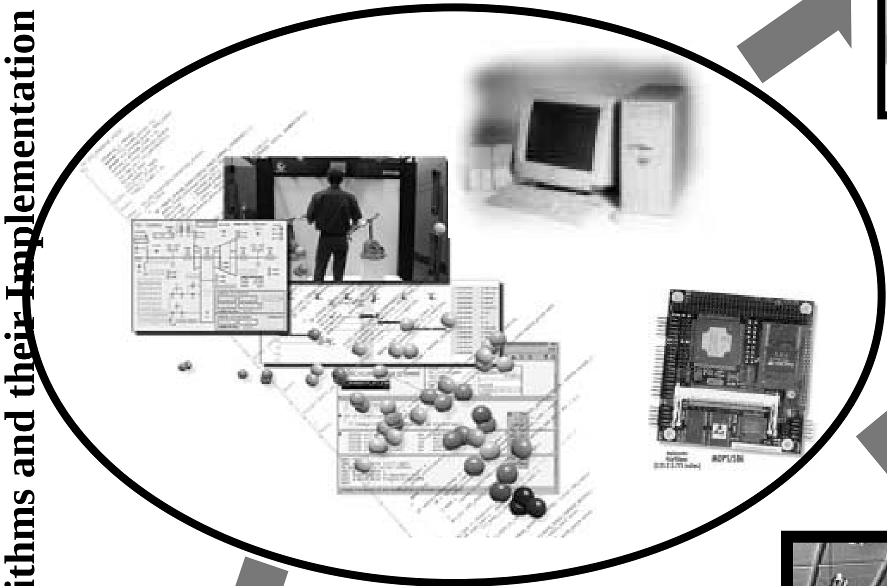
New Control Scenarios

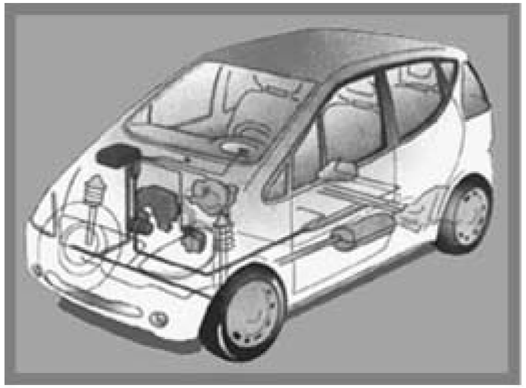
- Embedded Control Systems
- Networked Control Systems
- Non-regular sampling → Event-driven control

Embedded systems



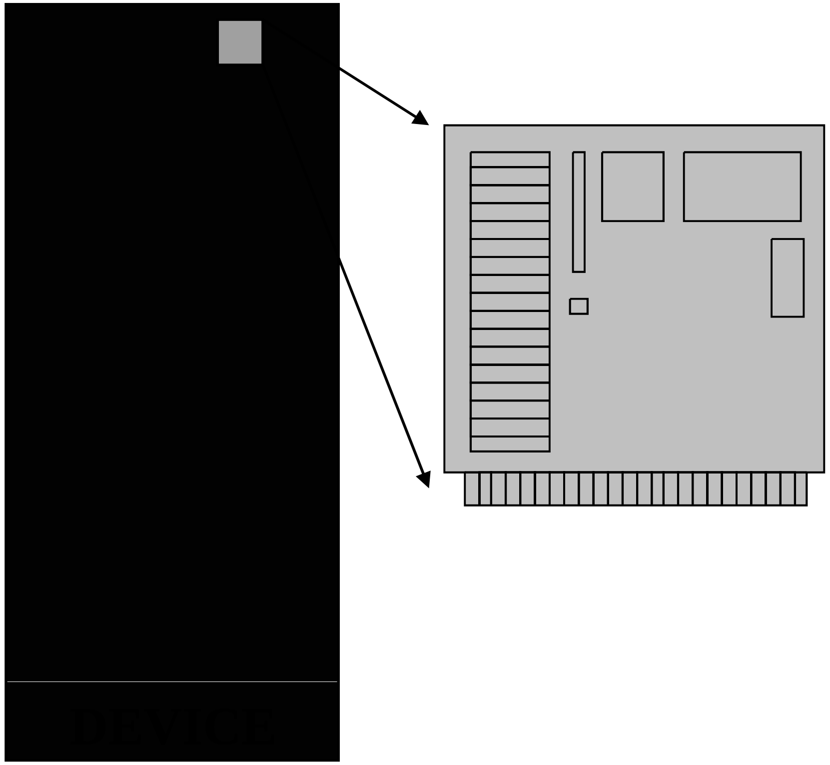
Embedded systems





Control is present in 99% of the embedded applications

Embedded systems



- Device:
  - Stand-alone
  - Networked
  - RT operation
- ES:
  - Compact and reduced size
  - Autonomy
  - Missing data operation
  - Fault-tolerant
  - Reconfigurability
  - Safety

Embedded control systems

Embedded systems

- Information processing systems embedded into a larger product
- Main purpose of the product is **not** information processing
- Must be
  - **Dependable**
  - **Efficient**
  - **Interactive with its environment**
  - **RT constrained**
  - **Application oriented**

ES Challenges

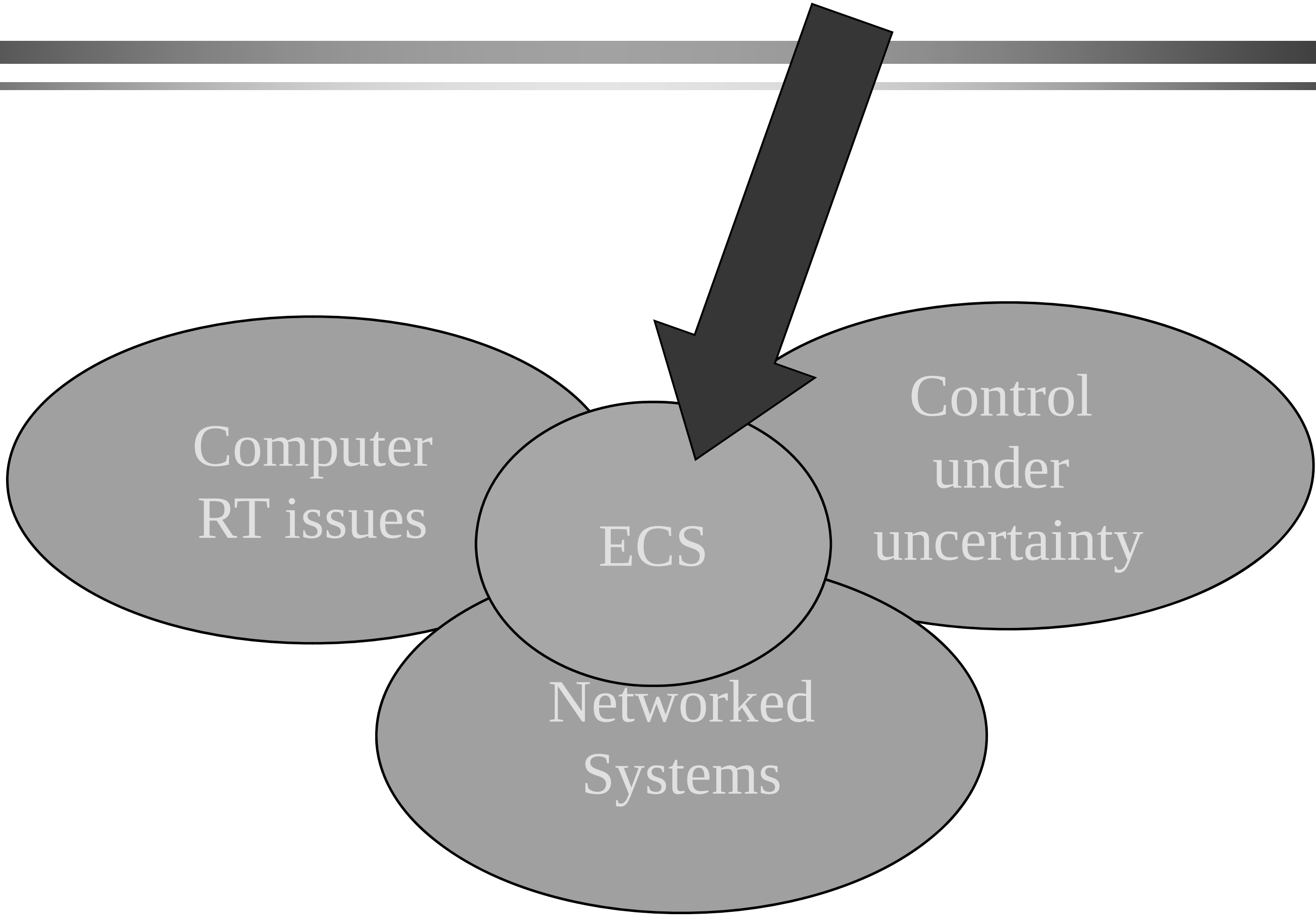
- Main issues refer to Embedded **Software**
  (not microelectronics / mechatronics)
- Most requirements / applications involve **control**
  - Reactive systems
  - RT constraints
  - Energy consumption
  - Environmental adaptation
  - ES control
  - **Pure control** applications

Strong interaction: control and its SW implementation

Embedded Control Systems

- Embedded systems with:
  - hard RT constraints
  - **guarantee of safe operation**
  - **best possible performances**
- Additional issues from viewpoint of:
  - implementation
  - computation
  - algorithmic

ECS

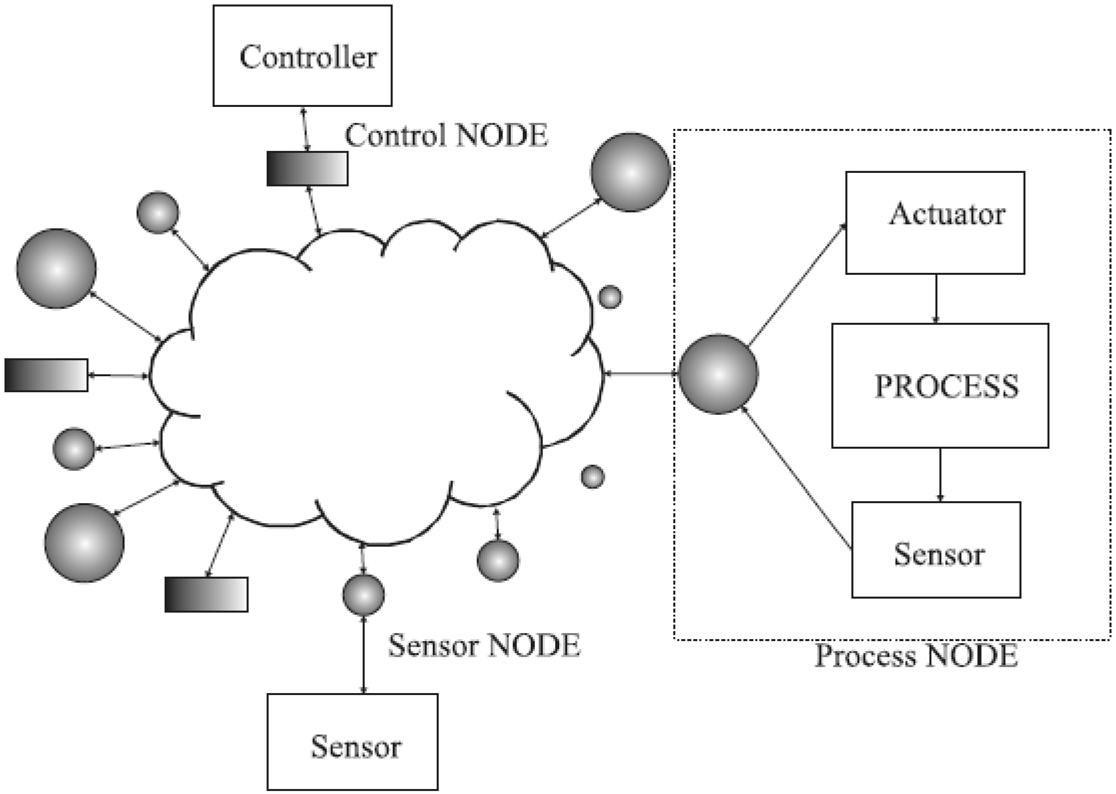

Computer
RT issues
ECS
Control
under
uncertainty
Networked
Systems

Embedded Control Characteristics

What distinguishes embedded control?

- **Limited computing and communication resources**
  - Often mass-market products, e.g, automobiles
  - CPU time, communication bandwidth, memory, energy, …
- **Autonomous operation**
  - No human ”operator”
  - Complex functionality
  - Often large amounts of software
  - Need for formal approaches
  - Need for design methodology

Networked Control Systems



❖ Networked Control Systems

Networked Control Systems

Basic Approach:

- 1. Identification of control tasks and mapping between tasks and processing nodes.
  - After this mapping, the worst case execution time (WCET) of each task can be computed as resulting from the target processing architecture.
- 2. Identification of shared information and bus scheduling.
  - Once the bus access protocols and scheduling have been determined the worst case for communication delays can be obtained.
- 3. Tasks scheduling for each node.
  - If some task cannot meet its deadline, return to Step 1 and reassig*n* tasks to nodes.

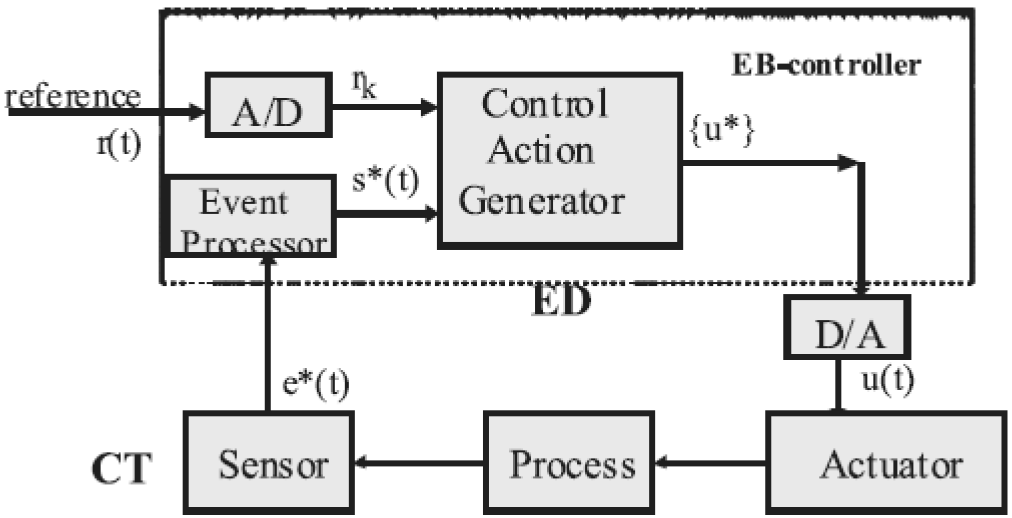
Networked Control Systems

Drawbacks

- 1. Off-line scheduling → too rigid and not optimal
  - Neighbor nodes, memory over-consumption.
  - Control problems in switching between nodes.
- 2. Resources infra-utilization
  - Everything should be scheduled for the worst case condition, with random and multiple switching
- 3. Control delays over estimation.
  - Control algorithms should be design for the longest communication delay, not being able to change under switching.

… and Improvements: under development!

Event-driven control

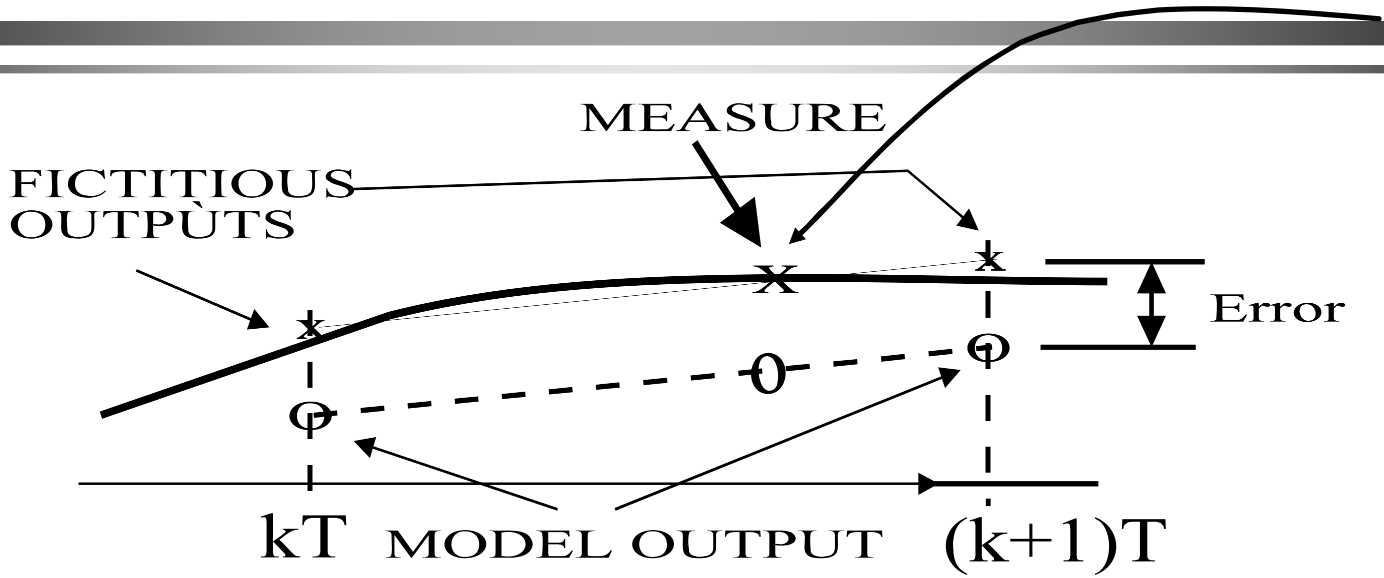


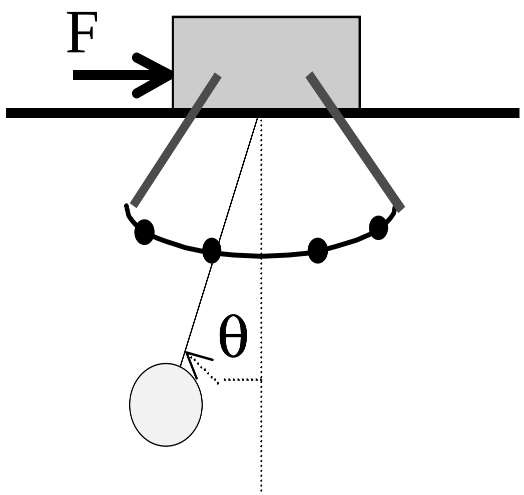
- Event-driven as opposite to time-driven:
  - Event generator (regular sampling data)
  - Event processor (data preprocessor)
  - Control action generator (controller)

Event-Driven Control

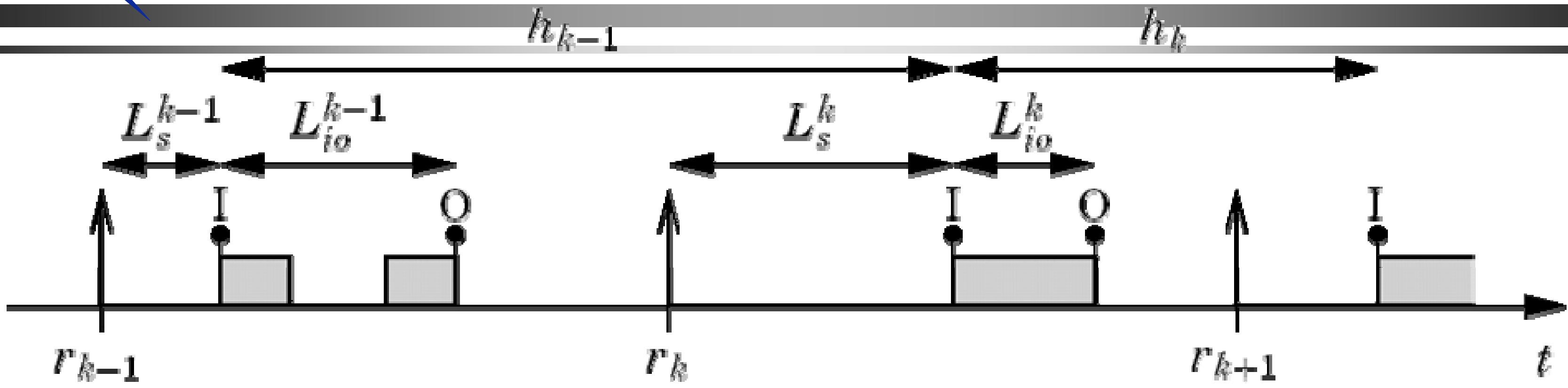
- Natural in networked embedded systems
  - describes the reality (e.g., networked control loops)
  - a possible modeling formalism for analyzing jitter
- Mode changes
- Discrete (binary sensors)
- Can it be combined with aperiodic schedulabiliy theory?

Event-driven control





Networked Embedded Control Timing

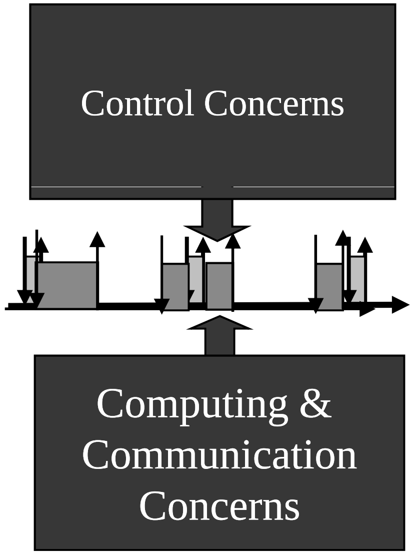


- Embedded control often implies temporal non-determinism
  - resource sharing
  - preemptions, blocking, varying computation times, non-deterministic kernel primitives, …
- Networked control often implies temporal non-determinism
  - network interface delay, queuing delay, transmission delay, propagation delay, link layer resending delay, transport layer ACK delay, ...
  - lost packets

How should we handle this?

Design Approaches

- Separation-of-concerns
  - Time-triggered approaches
  - Simple, deterministic, dependability, …
  - But, difficult to achieve in practice due to
    - Lack of resources
    - Incorrect assumptions
    - Technology incompatibility
- Integration
  - Optimize performance subject to limited resources
  - Codesign of control computing and communication
    - Temporal robustness analysis techniques
    - Implementation-aware control techniques
    - Control-aware computing and communication techniques
    - New analysis and design tools



ECS Design

Taking into account resources constraints

(delays, missing data, changes in the period …)

the goal of the temporal design methods is to

maintain/maximize the control performance :

- Maximize the time determinism
- Robust Design
- Active Robust design
- Prioritize control subtasks: Control Kernel

RT constraints

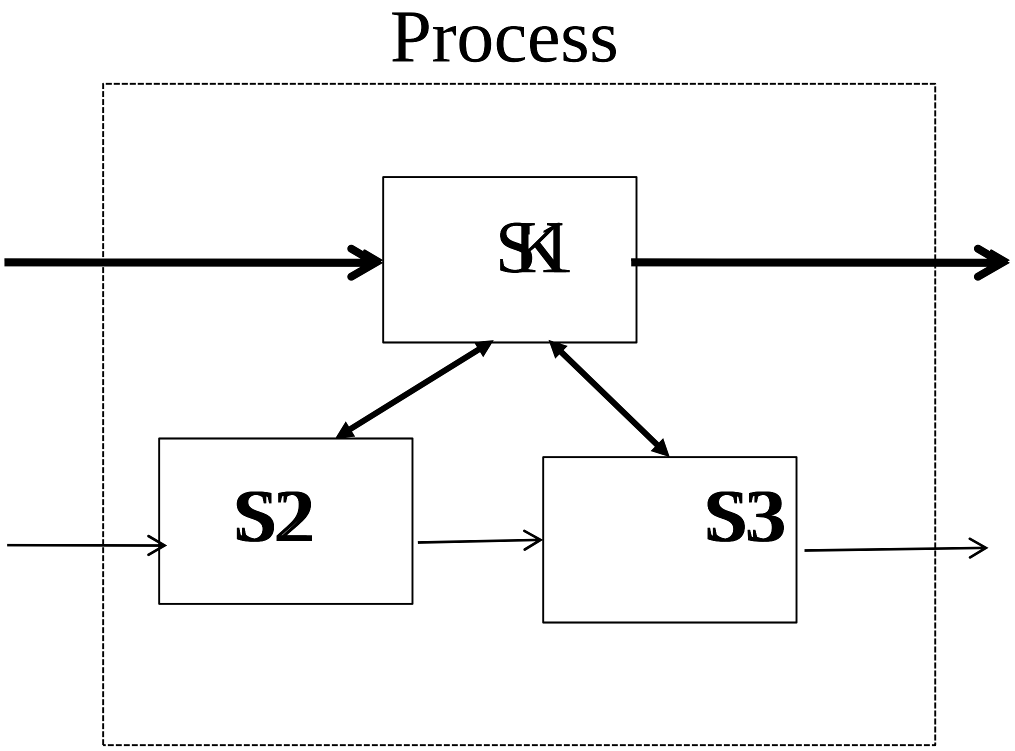
- Economic algorithms
- Optional tasks
- Hybrid systems
- CPU use measurement and optimisation
- On-line scheduling
- Memory saving
- Economic hardware redundancy
- Fault detection and isolation

→ Control algorithm design

Models, signals and controllers

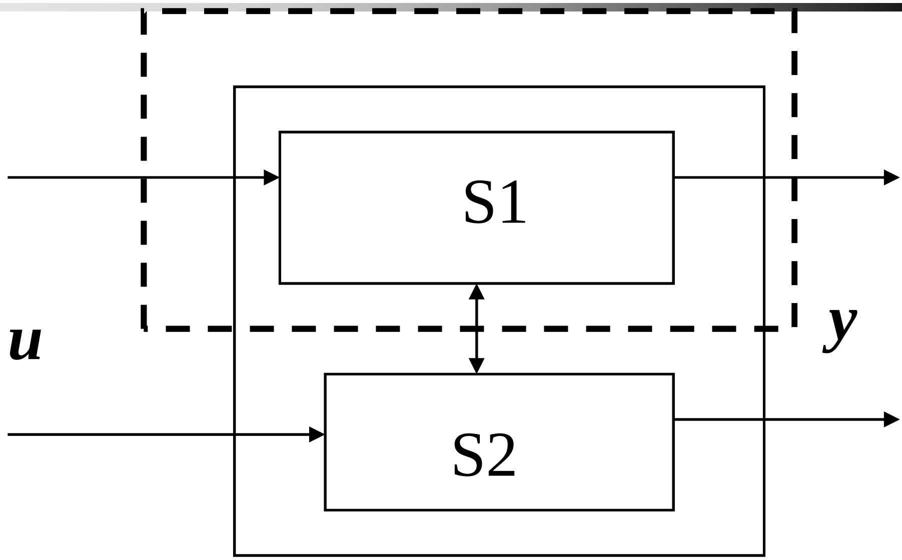
- Reduced order models
- Non-conventional sampling and updating patterns
  - Missing data control
  - Event-triggered control
- Decision and supervisory control
  - Hybrid control systems
  - Multimode control
  - Sampling rate changes
- Fault-tolerant control
- Degraded and back-up (safe) control strategies
- Battery monitoring and control

Reduced Order Model



Reduced Order Model

- Model reduction:
  - Partial control (parts of the plant)



- Partial phenomena (fast/slow dynamics)

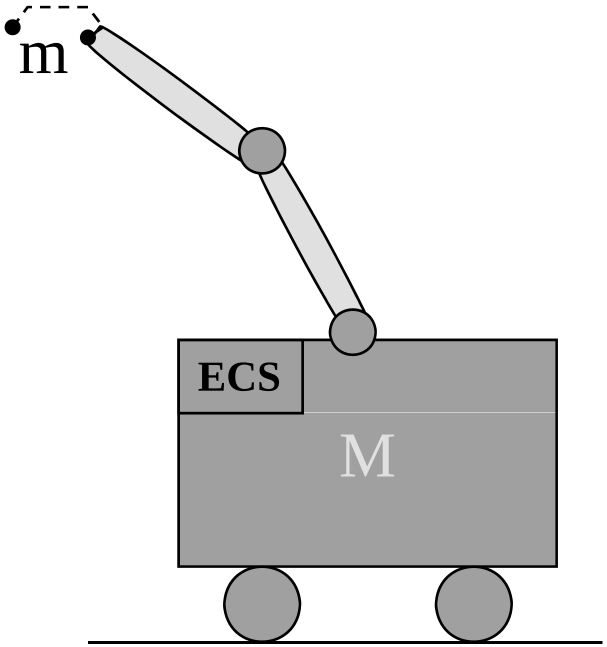
$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_{k+1} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_k + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} u_k; \quad y_k = \begin{bmatrix} C_1 & C_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_k$$

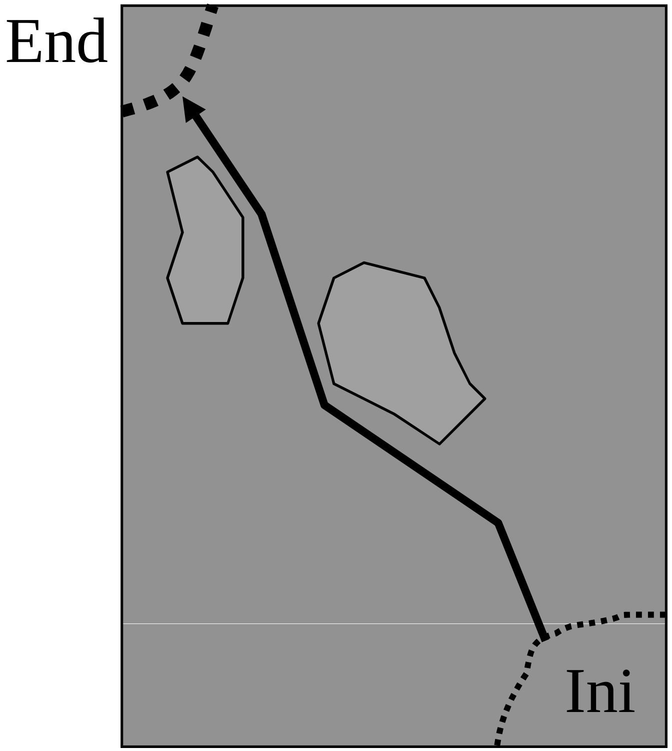
$$x_1 : \text{fast modes}; \quad x_2 : \text{slow modes}$$

- Flexible arm
- Navigation

Reduced Order Models: Example

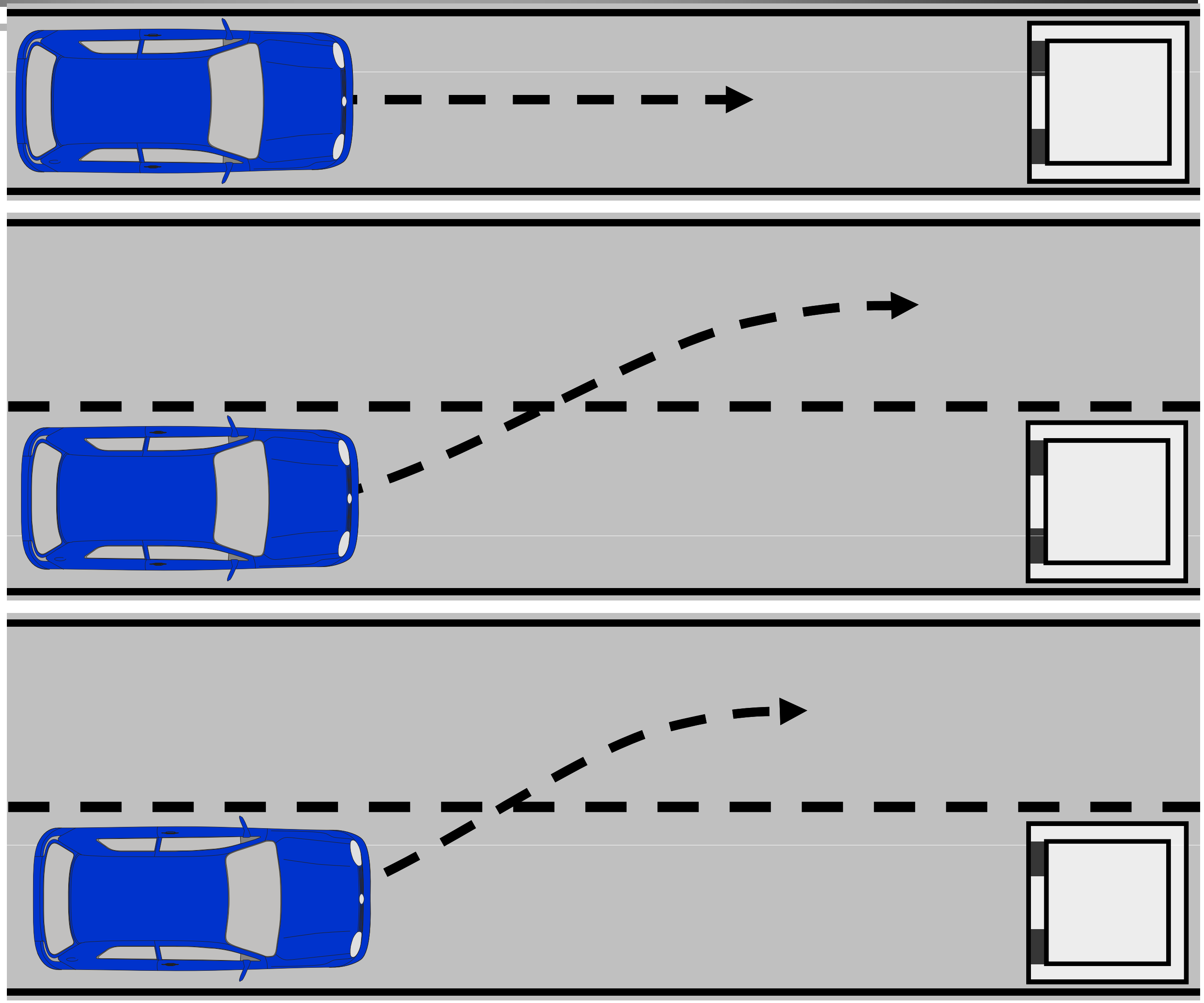
No obstacles: manipulator control
Obstacle detection. Navigation control
Final position: manipulator control





Interventions

- Braking
- Steering
- Combined steering and braking

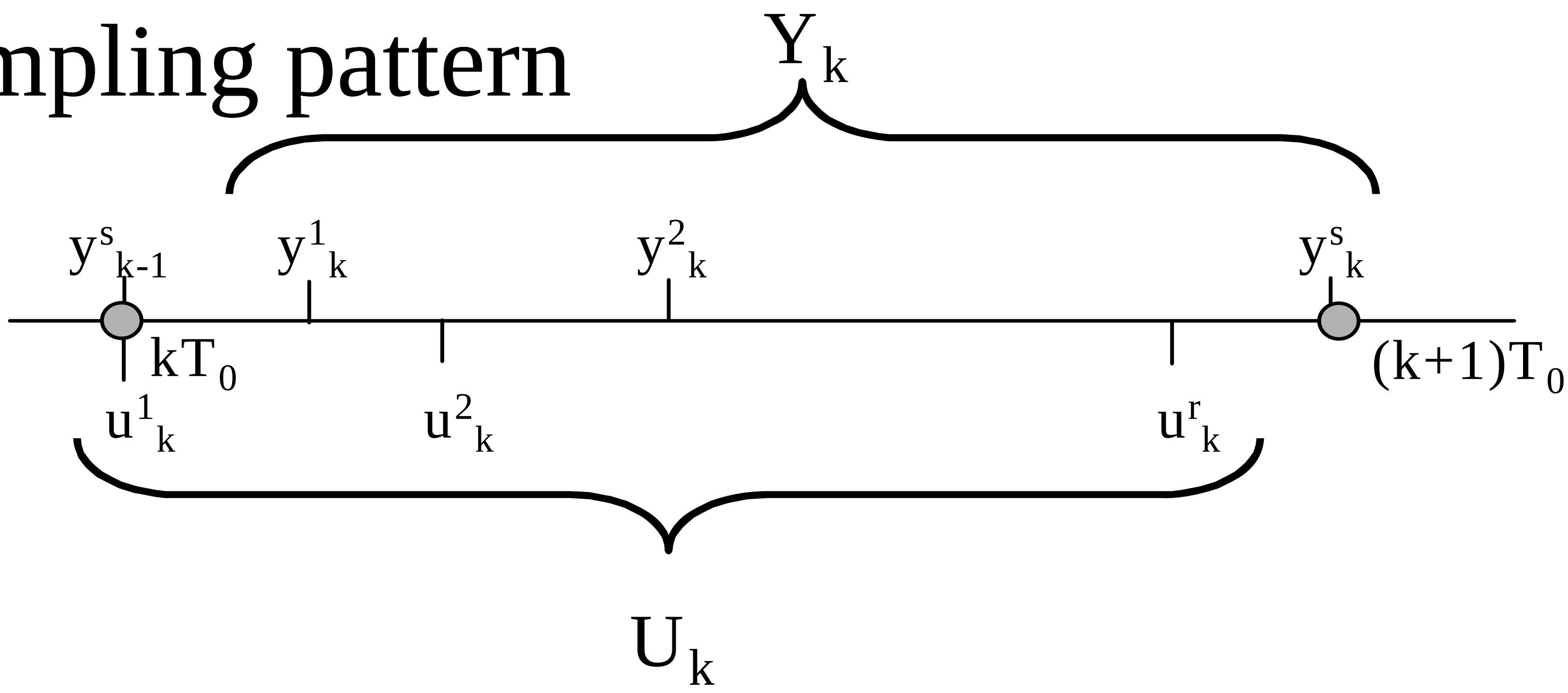


ECS: Control Algorithm viewpoint

- Reduced order models
- Non-conventional sampling and updating patterns
  - Missing data control
  - Event-triggered control
- Decision and supervisory control
  - Hybrid control systems
  - Multimode control
  - Sampling rate changes
- Fault-tolerant control
- Degraded and back-up (safe) control strategies
- Battery monitoring and control

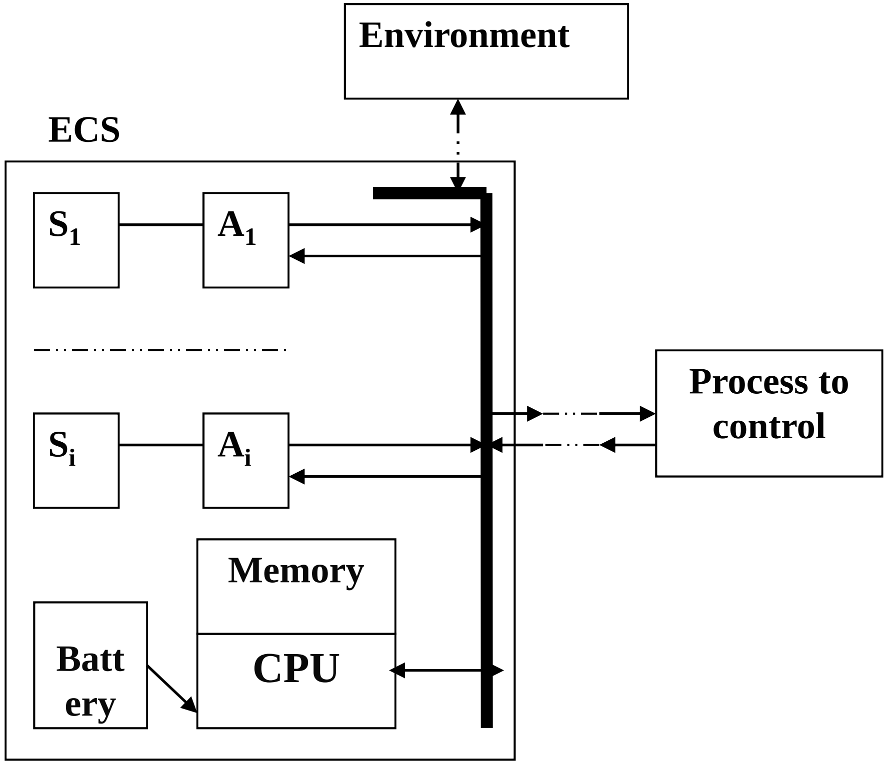
Non-uniform sampling

Sampling pattern



- Irregular sampling
- Time delays
- Relevance of variables

Non- Conventional Sampling & Updating



- Non synchronism
- Different timing
- CPU sharing &
- Communication channels

→

- Variable time-delays
- Delay counteraction
- Multirate control

Small changes in the code

Missing data & Event-triggered

- Sensors failure
- Com. Channels congestion
- Steady-state

→

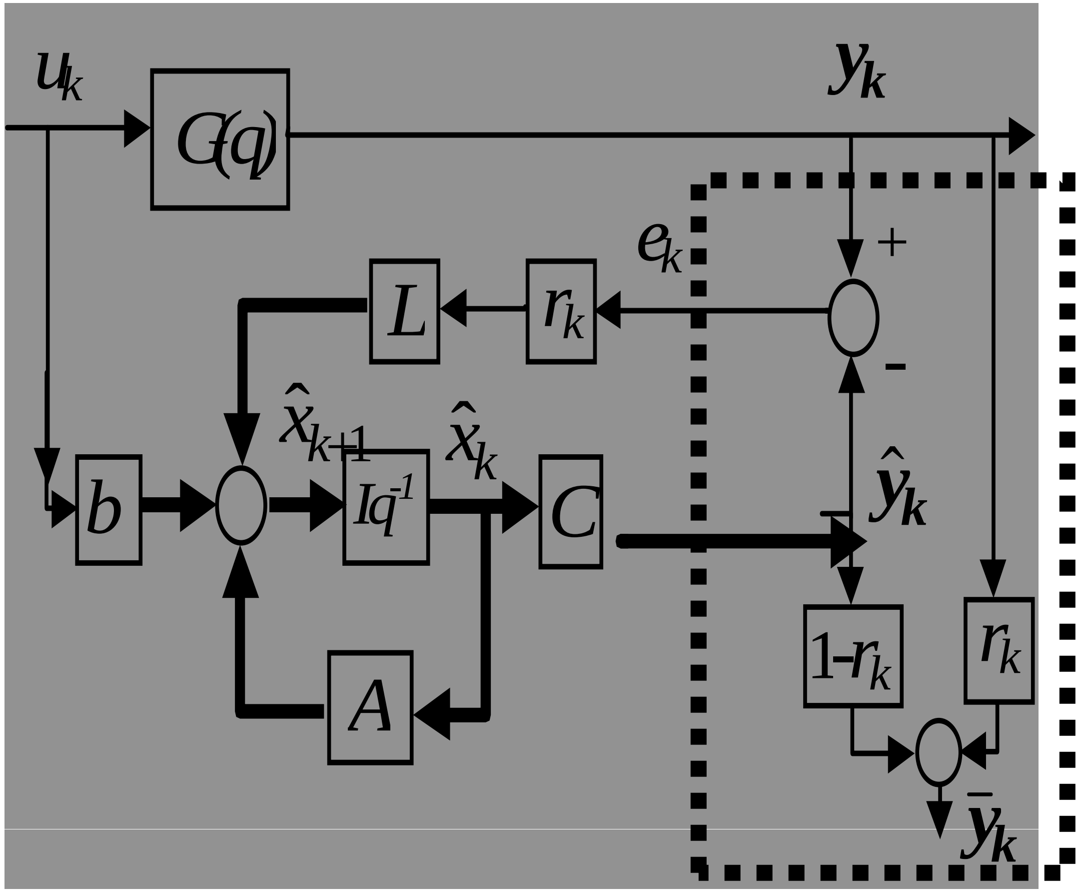
- Output prediction
- Parameter estimation

  "Virtual sensors"
- Convergence, stability
- Compute nothing

Missing Data

The output is only available at some time instants: $r_k = 1; \quad r_k = \{0,1\}$

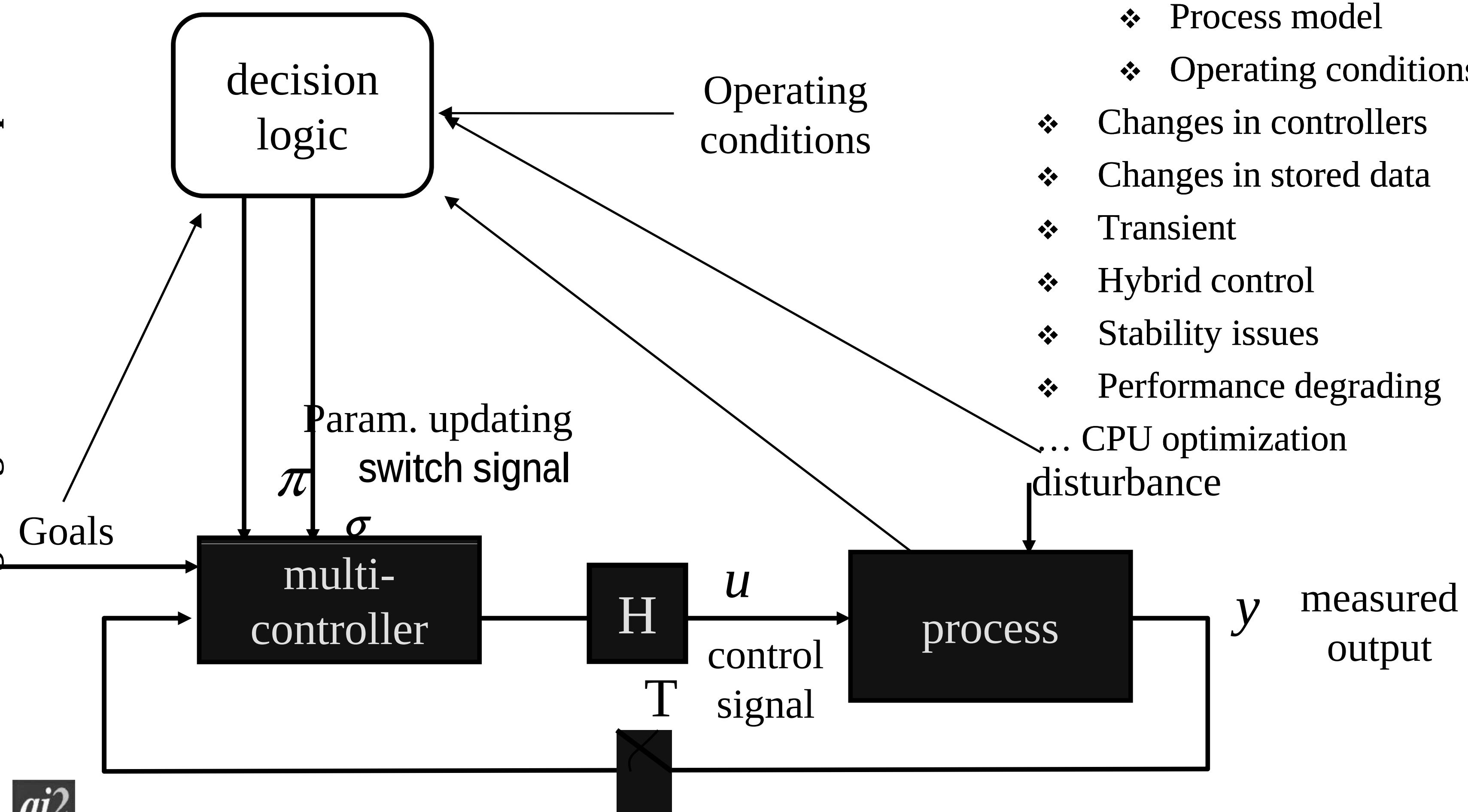
KALMAN Filter



ECS: Control Algorithm viewpoint

- Reduced order models
- Non-conventional sampling and updating patterns
  - Missing data control
  - Event-triggered control
- Decision and supervisory control
  - Hybrid control systems
  - Multimode control
  - Sampling rate changes
- Fault-tolerant control
- Degraded and back-up (safe) control strategies
- Battery monitoring and control

Decision & supervisory control



- Decision based on
  - Performances
  - Process model
  - Operating conditions
- Changes in controllers
- Changes in stored data
- Transient
- Hybrid control
- Stability issues
- Performance degrading

… CPU optimization

Controller commuting: Stability issues

Each controller may stabilize the plant under control,
But ... what under commuting?

- Common Lyapunov function
- Controller initialization
- Controller resetting

→ Not a problem if seldom changes

Supervision: Sampling rate

- CPU availability
- Battery level
- Mode changes

→

Changes in sampling rate

- Commuting problems
  - Stability
  - Transient

ECS: Control Algorithm viewpoint

- Reduced order models
- Non-conventional sampling and updating patterns
  - Missing data control
  - Event-triggered control
- Decision and supervisory control
  - Hybrid control systems
  - Multimode control
  - Sampling rate changes
- Fault-tolerant control
- Degraded and back-up (safe) control strategies
- Battery monitoring and control

Fault-tolerant control

- Use the control design freedom to
  - Ensure stability under sensors/actuators failure
  - Guarantee minimal performances
- Supervision based fault-tolerant control
  - FDI
  - Controller commuting
  - Safe (back-up) operation

Power awareness

- Power availability supervision
- Mode changes

RT control implementation

- The same resources must be **shared** between different tasks
- **Alternative** control **algorithms** should be ready to get the control of the process
- **Working conditions**, such as priority, allocated time and memory or signals availability may change
- **Variable delays** should be considered
- Priority to **safety** tasks
- **Validation** and certification

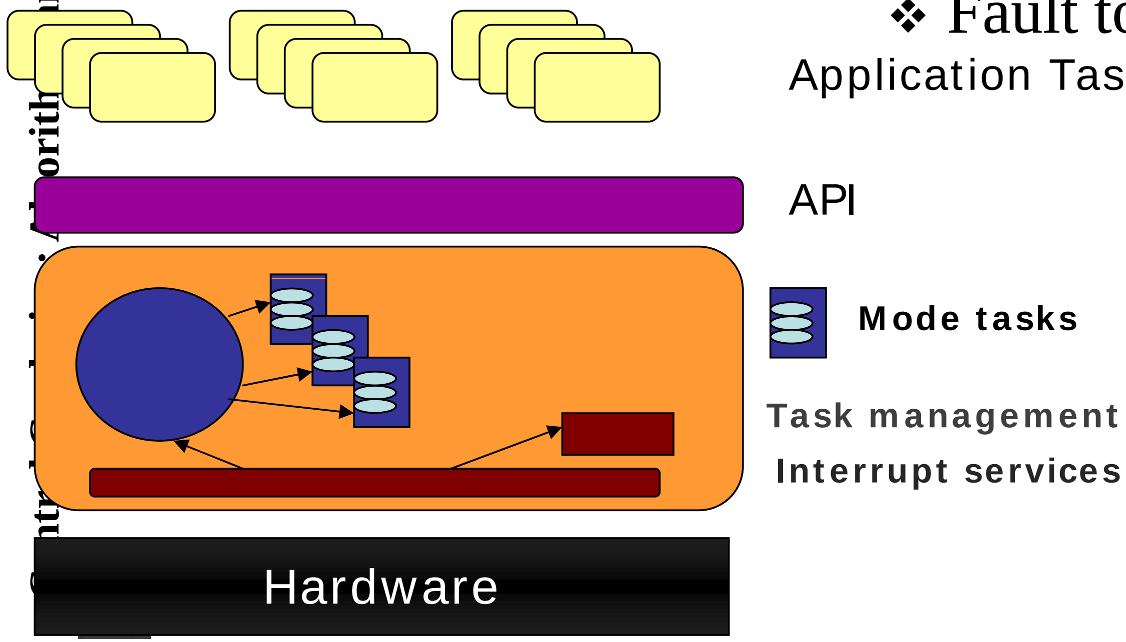
Control Kernel

Kernel Concept

OS kernel:

- Basic services:
  - Task and time management
  - Interrupt handling
  - Interface to the applications (API)
  - Mode changes
  - Fault tolerance

OS Kernel structure



Additional services

- File management
- Quality of service
- Tracing and debugging

Kernel Concept

OS kernel:

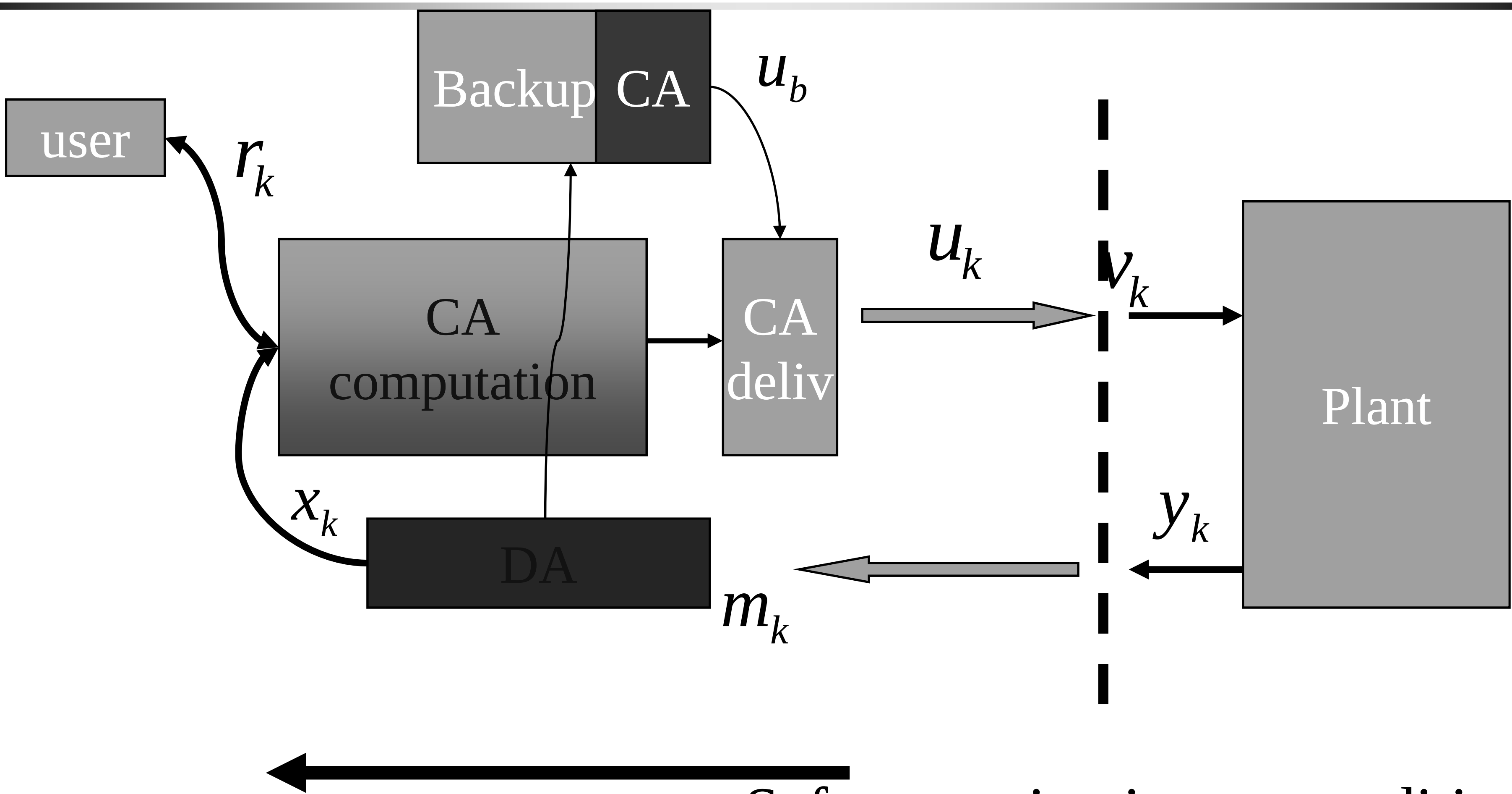
- Basic services:
  - Task and time management
  - Interrupt handling
  - Interface to the applications (API)
  - Mode changes
  - Fault tolerance
- Additional services
  - File management
  - Quality of service
  - Tracing and debugging

OS Kernel for control

The OS Kernel provides the minimal services that should be included in any embedded **control** system.

- **Fault tolerance**
  - Degrade task activity (when a task does not guarantee some timing constraints, the degraded behavior is executed)
  - Change mode events raised when some faults can not be managed.
- **Mode changes**
  - Mode definition (set of tasks associated to a mode)
  - Mode change events (event to change from one mode to another)
  - Mode change protocol

The control kernel concept



Safe operation in any condition

Control Kernel

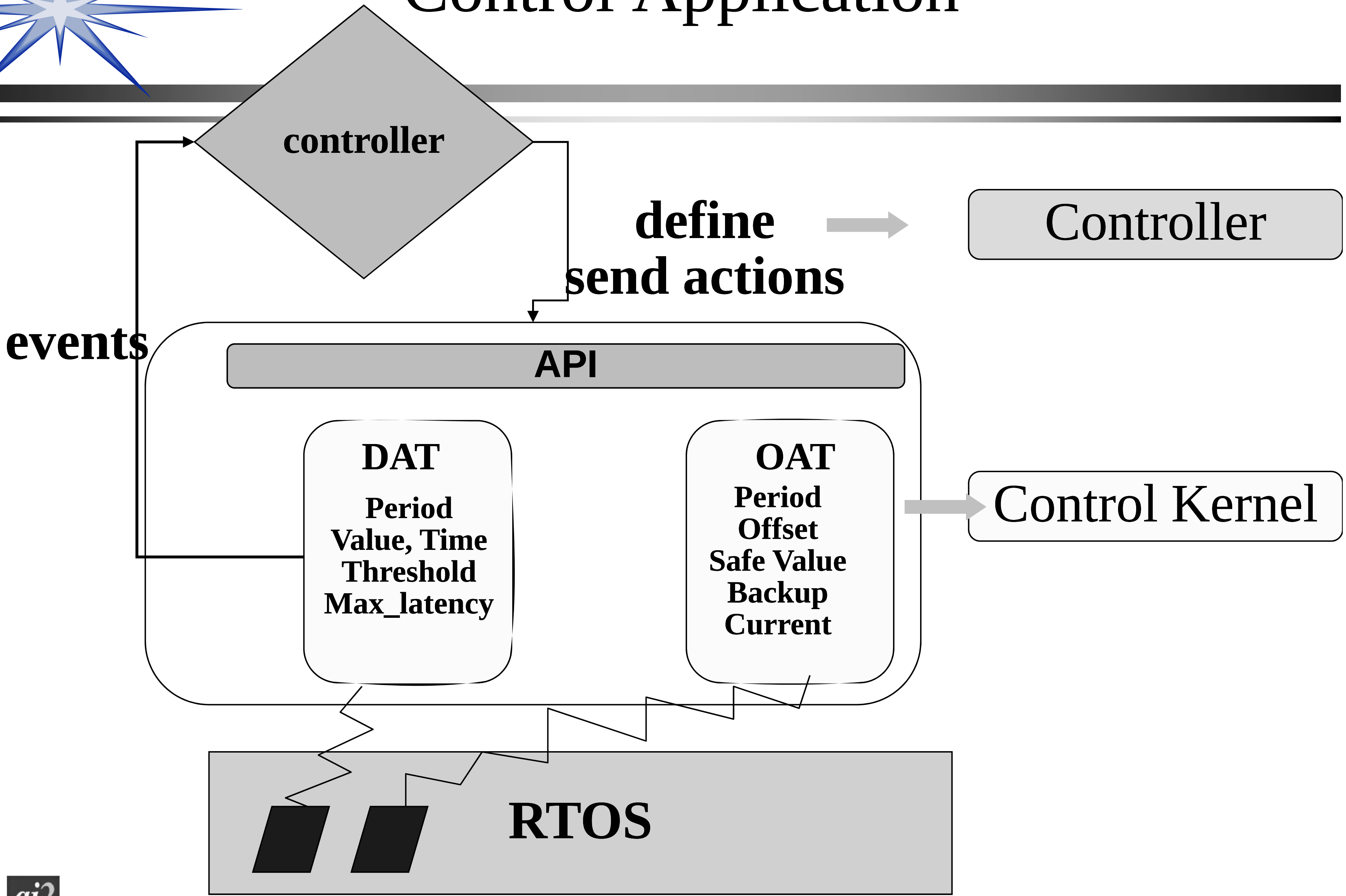
- Ensures control action (CA) delivering
  - Safe (back-up) CA computation
  - Safe CA computation based on previous data
- Data acquisition of major signals
  - Safe CA computation based on current data
- Transfer to new control structure
  - Basic control structure parameters computation
  - CA computation

Control Kernel

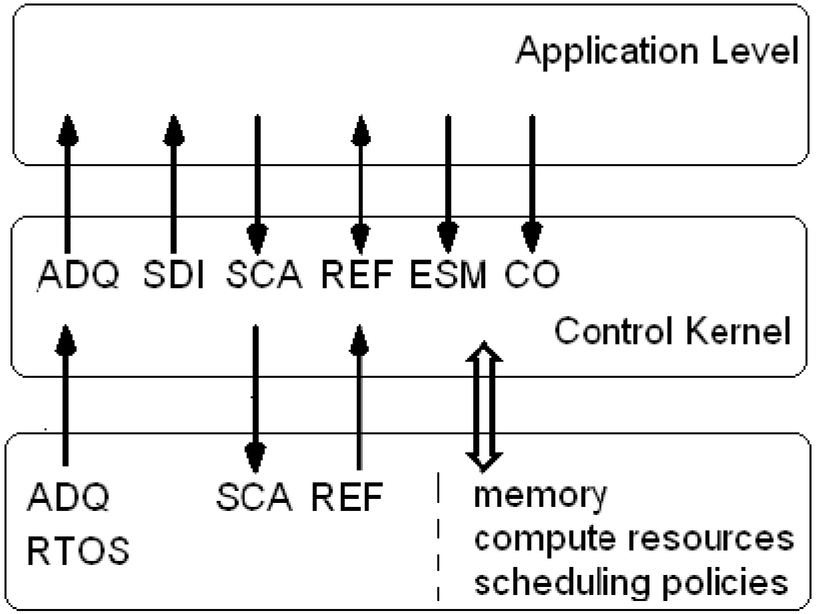
Controller

- Full DA
  - Control structures evaluation and selection
  - CA computation (different levels)
- Communication facilities
- Coordination facilities

Control Application

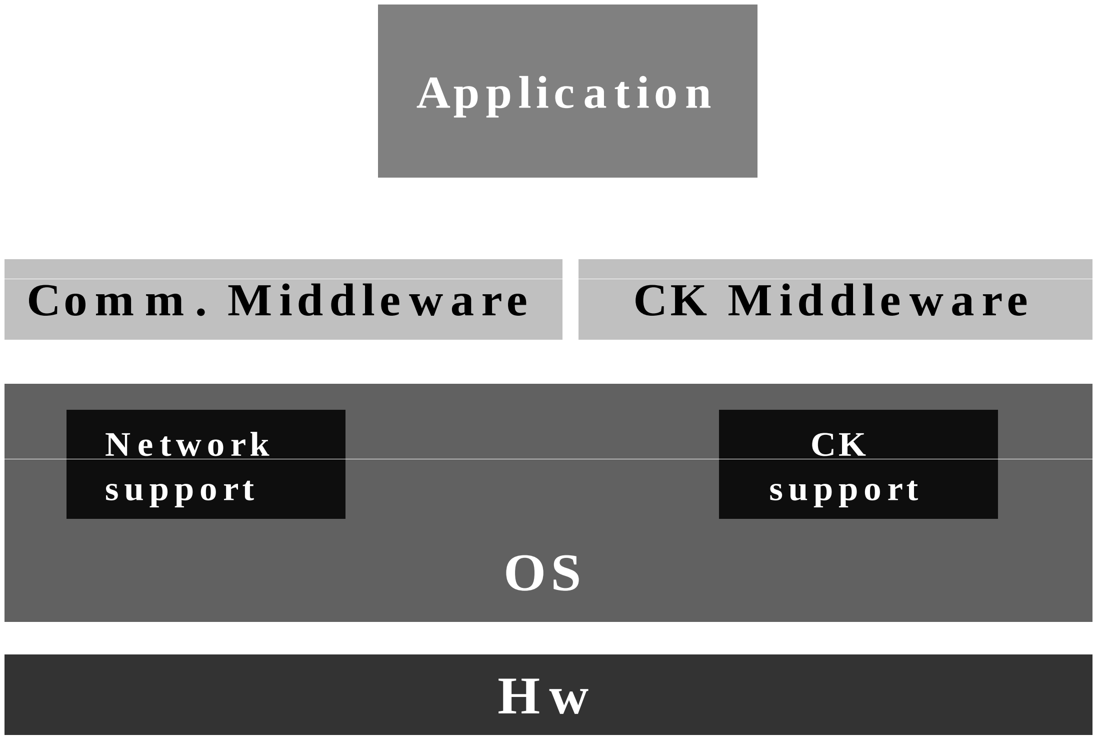


Layers and Interactions of the CK



- ADQ: Samples of variables.
- REF: Control references.
- ESM: Outputs, references and inputs estimation.
- CO: Control commands, including
  - Controller commuting
  - Change of controllers' parameters.
- SCA: Sending of control actions.
- SDI: State and diagnostic of inputs.

CK Middleware



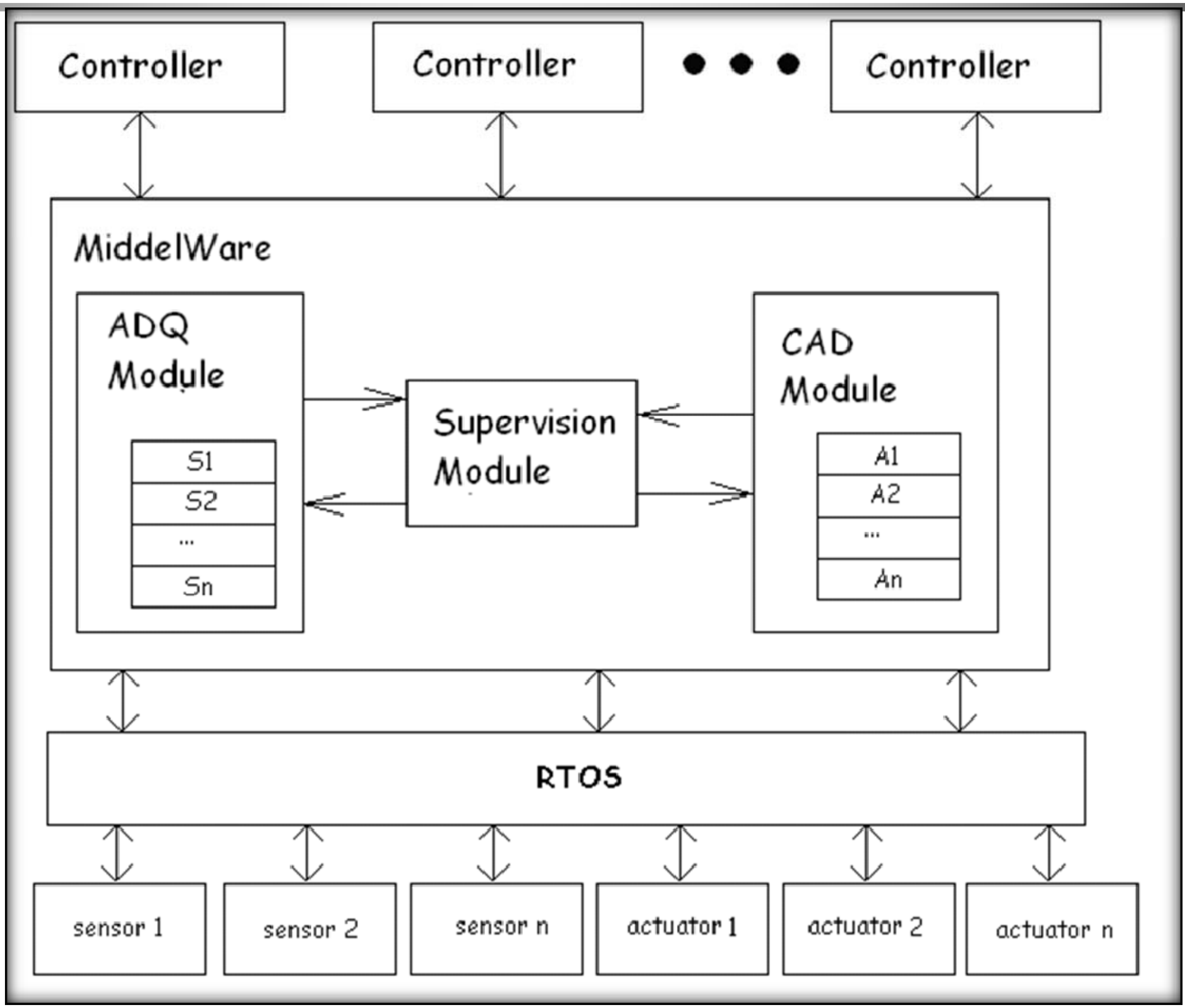
CK Middleware functionalities

- Provides object classes for sensors, actuators, controllers
- Remote communication through Comm. Middleware
- Pool of threads at different priority levels (acquisition, data acquisition, basic computation)
- Admission control (negotiation)
- Mode change (task + controllers commutation)

CK Middleware functionalities (II)

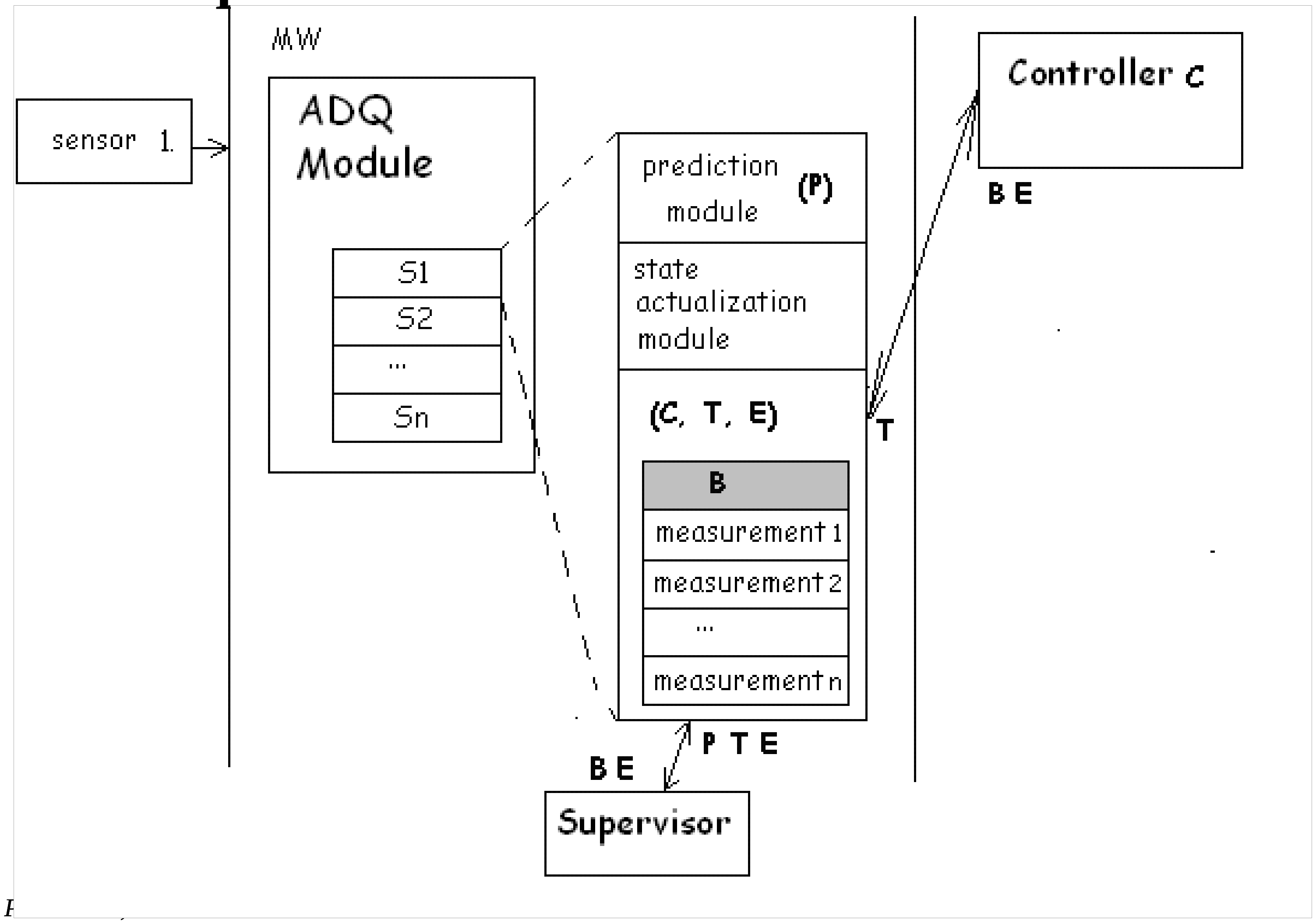
- Definition of controller parameters:
  - Reduced model controller
  - Backup actuation
  - Sensor characteristics (virtual/real, range, acquisition period, filter, threshold, ...
  - Actuator characteristics
  - Call-back function
- Compute RMController (locally)

CK Middleware structure



CK Middleware structure

Data acquisition

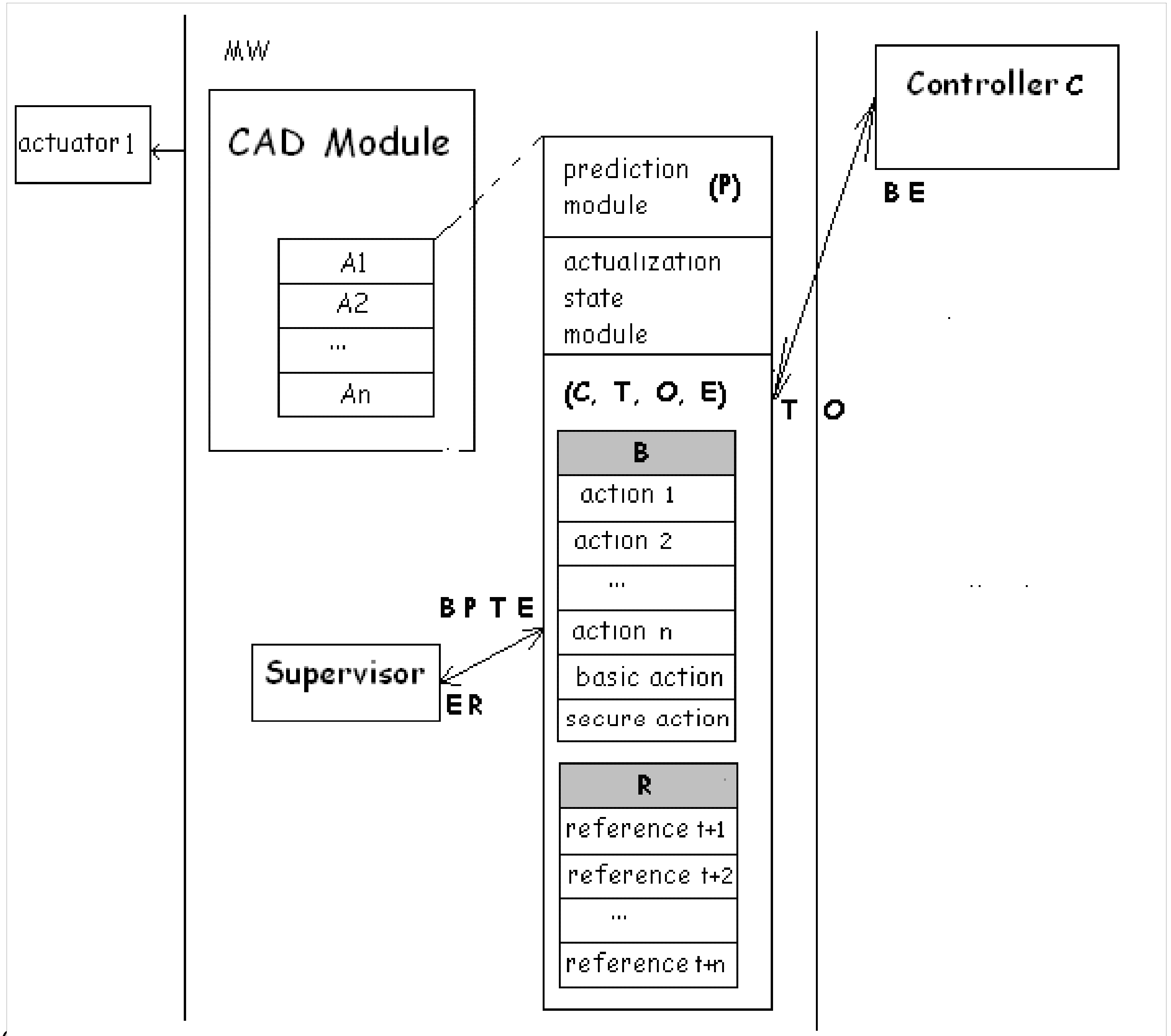


CK Middleware structure

- Each physical sensor S has:
- S = {T, B, C, E},
  - T: sampling period,
  - B: buffer n last values,
  - C: the controller function
  - E: the sensor state {fail, event, no_fail}
- Acquisition quality
  - Via data acquisition interval (DAI) concept

CK Middleware structure

Actuation

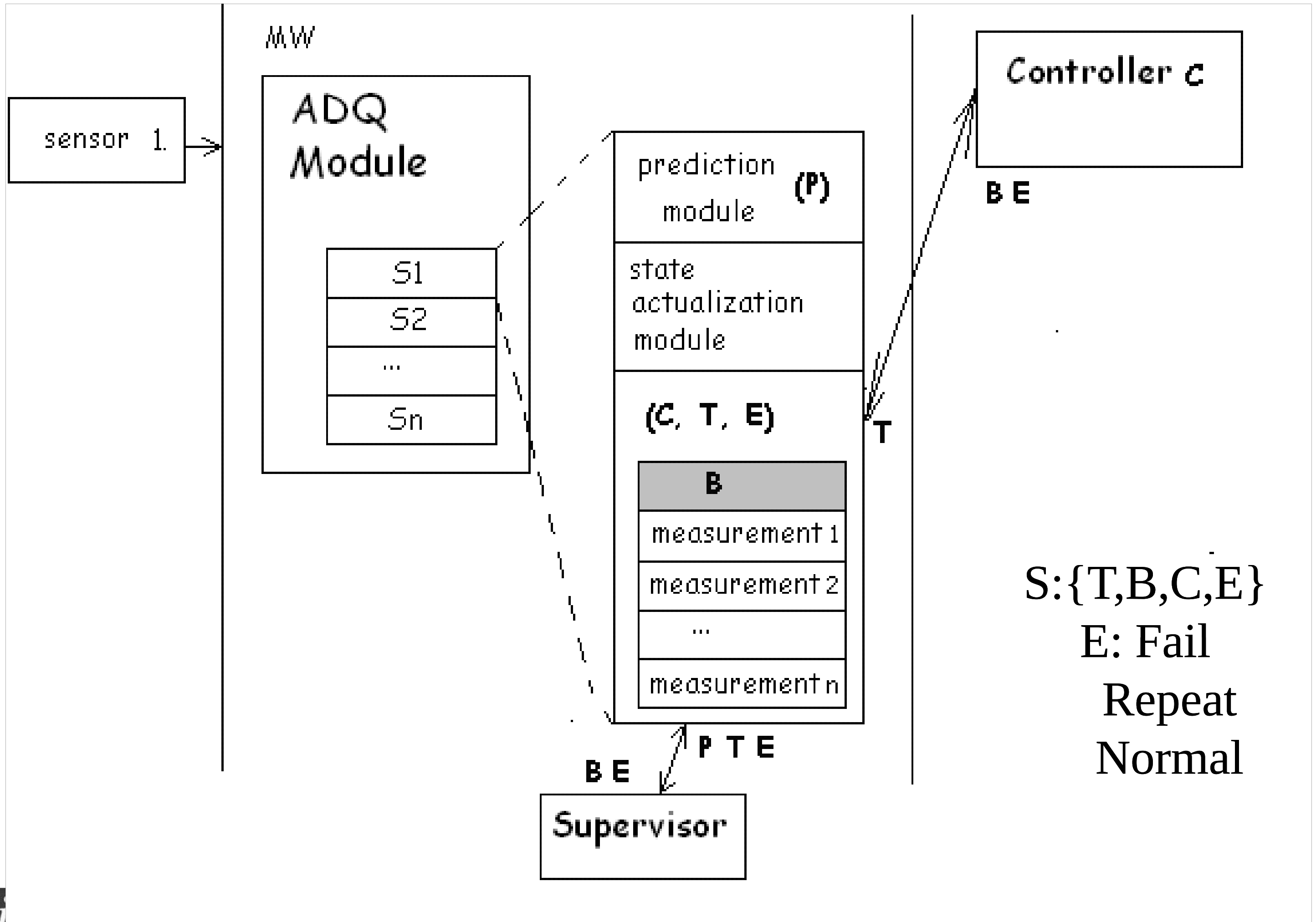


CK Middleware structure

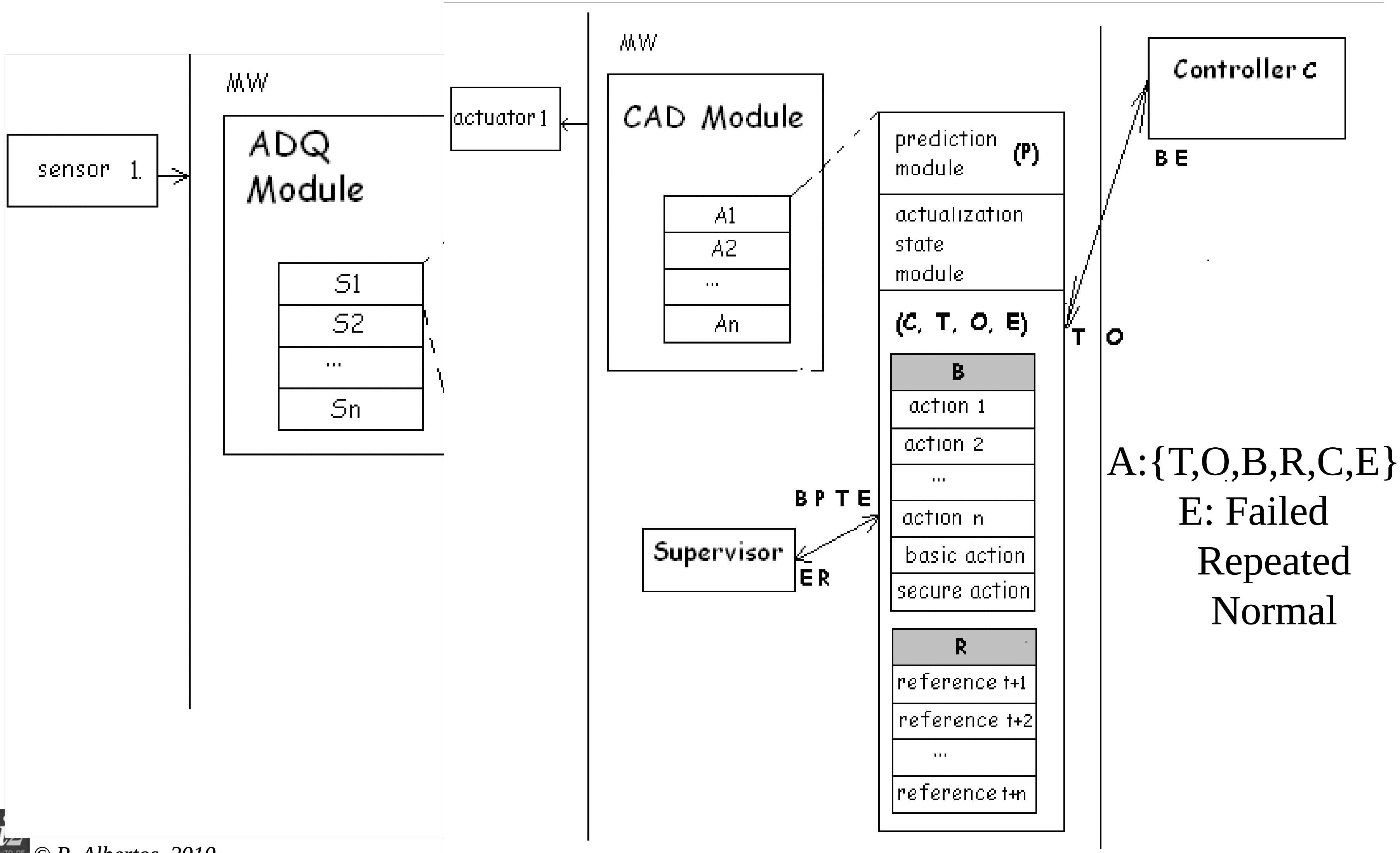
Actuation

- Each physical actuator A has:
- A= {T, O, B, R, C, E}
  - T: sampling period,
  - O: Offset between delivering of the action and acquisition of data.
  - B: buffer n last values.
  - R: To store the n future references values.
  - C: the controller function.
  - E: the sensor state {fail, event, no_fail}
- Delivering actions quality
  - Via data acquisition interval (CAI) concept

Control Kernel structure



Control Kernel structure



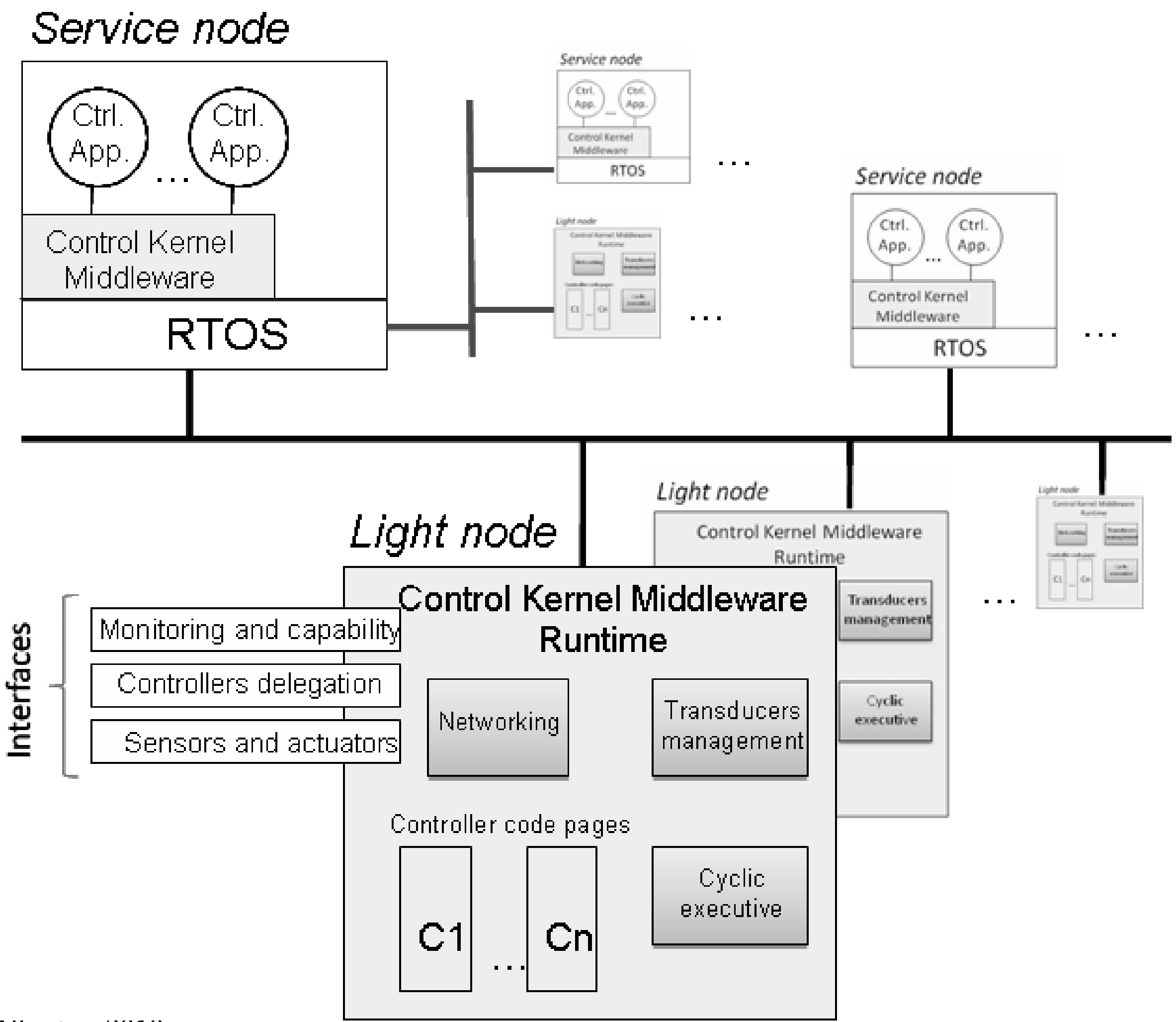
A:{T,O,B,R,C,E}

E: Failed

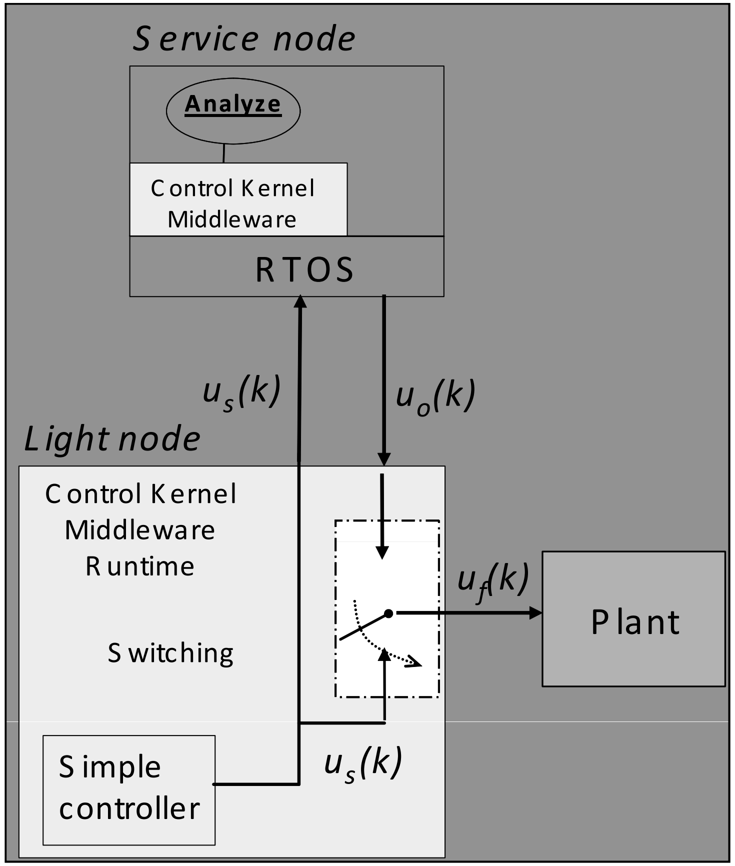
Repeated

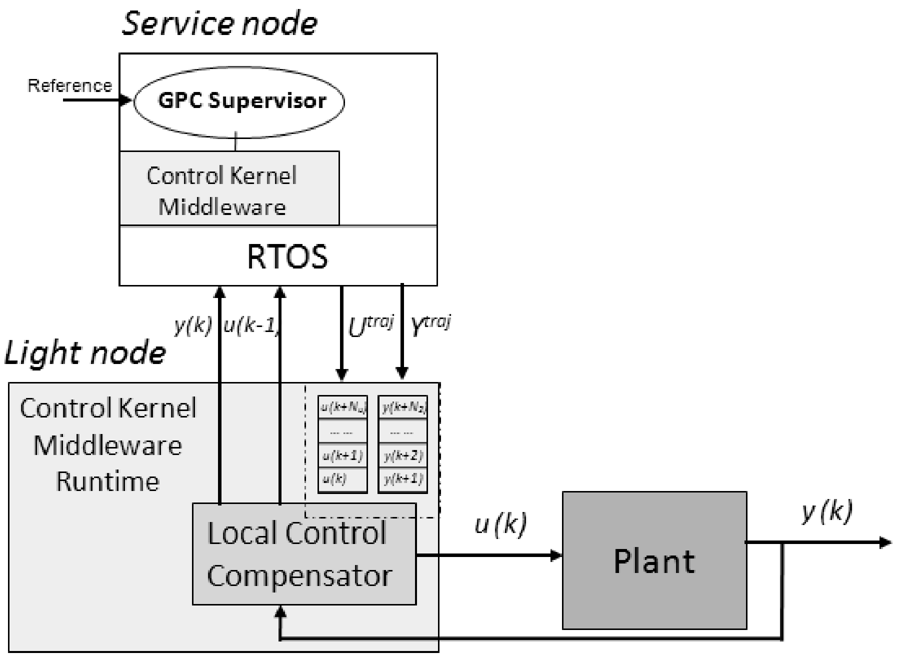
Normal

CK layout


Service node
Ctrl. App.
Ctrl. App.
Control Kernel Middleware
RTOS
Service node
Light node
Light node
Control Kernel Middleware Runtime
Interfaces
Monitoring and capability
Controllers delegation
Sensors and actuators
Networking
Transducers management
Controller code pages
C1
Cn
Cyclic executive

CK detail


Service node
Analyze
Control Kernel Middleware
RTOS
$u_s(k)$
$u_o(k)$
Light node
Control Kernel Middleware Runtime
Switching
$u_f(k)$
Plant
Simple controller
$u_s(k)$


Service node
Reference
GPC Supervisor
Control Kernel Middleware
RTOS
y(k)
u(k-1)
U^{traj}
Y^{traj}
Light node
Control Kernel Middleware Runtime
Local Control Compensator
u(k)
Plant
y(k)

Scheduling Policies

- Several scheduling policies can coexist depending on the thread level.
- Kernel threads (DAThread and OAThread) are executed as part of the RTOS. Both are periodic and serve acquisition and delivery actions
- Both have a **queue** were requests are served on deadline basis.
- Values are written/read to/from control kernel middleware.

Implementation

- Current version of the CK Middleware has been implemented in C
- The RTOS used is PartiKle and open-source rtos which is the new core of RTLinux_GPL
- It can be executed in x86 or ARM processors
- Different execution platforms

CONCLUSIONS

- Need of Codesign of Control algorithms and their implementation
- Flexibility in the control scenarios
  - Embedded, networked, event-driven
- Distribution of the computing resources
- Limitations in Communication and Computing
- Control safety
- Different treatment of:
  - Signals: Relevance and Control Effort
  - Tasks: Control Kernel
  - Models and goals: resource availability
- Integration of the control algorithm in the computing activities

→ Co-Design

THANK YOU

Control Co-design: Algorithms and their Implementation

P. Albertos, A. Crespo, J. Simó
Dpt. Ing. de Sistemas y Automática
Instituto de Automática e Informática Industrial
Universidad Politécnica de Valencia
pedro@aii.upv.es

A. Fernández
Dpt. Automática y Computación
CUJAE, Habana. Cuba
adel@electrica.cujae.edu.cu

VALENCIA, SPAIN, 14-18 JUNE 2010