Adaptive Embedded Systems

Karl-Erik Årzén

Dept of Automatic Control

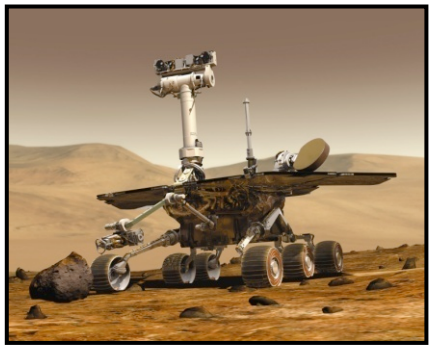
Lund University

Outline

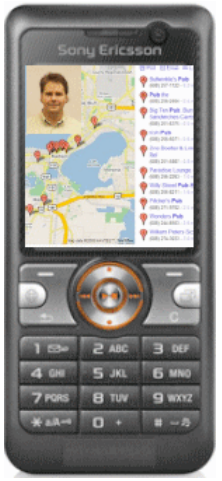
- Embedded System Trends
- Definitions
- Adaptivity and Control
- Reconfigurable hardware
- Embedded Adaptivity Issues
- Three examples:
  - Feedback-Based Queue Length Control
  - Feedback Scheduling of Control Tasks
  - Adaptive Resource Management in ACTORS
- Conclusions

Embedded System Trends

- Increasing functionality of embedded systems
  - From small microcontrollers to embedded laptops
    - Increased complexity
  - Higher requirements on autonomous behaviour
  - Mixed-criticality
    - Both hard and soft real-time constraints
    - Both safety-crictical parts and non-safety critical
  - Programmability
    - Software-based embedded systems
    - Programmable hardware



Embedded System Trends



- Applications increasingly adaptive
  - Single-application embedded systems
    - Example: A multimedia application that dynamically changes its resolution or frame rate to save battery life-time
  - Multiple-applications embedded systems
    - Embedded systems are increasingly open with support for (on-line) installation of third-party software
    - The number of applications executing and their run-time characteristics change dynamically
    - Increased uncertainty about use cases and workload scenarios → design based on worst-case prior information unfeasible
    - Adaptive resource management required

Embedded System Trends

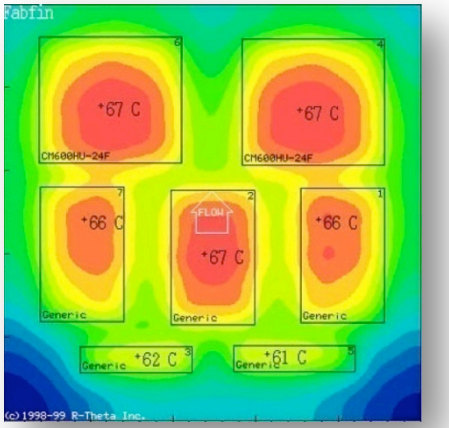
- Hardware increasingly adaptive
  - Run-time reconfigurability (FPGA, SoC, NoC,…)
  - Dynamic Voltage/Frequency Scaling (DVFS)
    - Dynamic adjustment of supply voltage and clock frequency to minimize power consumption
  - Dynamic Power Management (DPM)
    - Processors with power-down and power-off modes
    - Selective down-powering of MPSoCs

Embedded System Trends

- Hardware increasingly non-predictive
  - Pipelines, caches, multi-cores, etc make worst-case execution time (WCET) estimation difficult
    - Single core, single-thread with caches → can be handled
    - Single core, multiple threads, no caches → can be handled
    - Single core, multiple threads with caches → starting to be problematic
    - Multiple cores, with or without caches → very pessimistic
    - Increases the need for adaptive approaches
  - Variability in nanometer process technologies

Embedded System Trends

- Increased requirements on system reliability
  - Reactive:
    - Dynamic reallocation of application tasks from faulty architecture elements (e.g., cores), rather than, e.g. duplication and voting mechanisms
  - Proactive:
    - Dynamic reallocation to avoid hotspots and, hence, faults
      - Taking temperature gradients into account



- From static to dynamic mapping of applications

Example: Cellular Phones Today

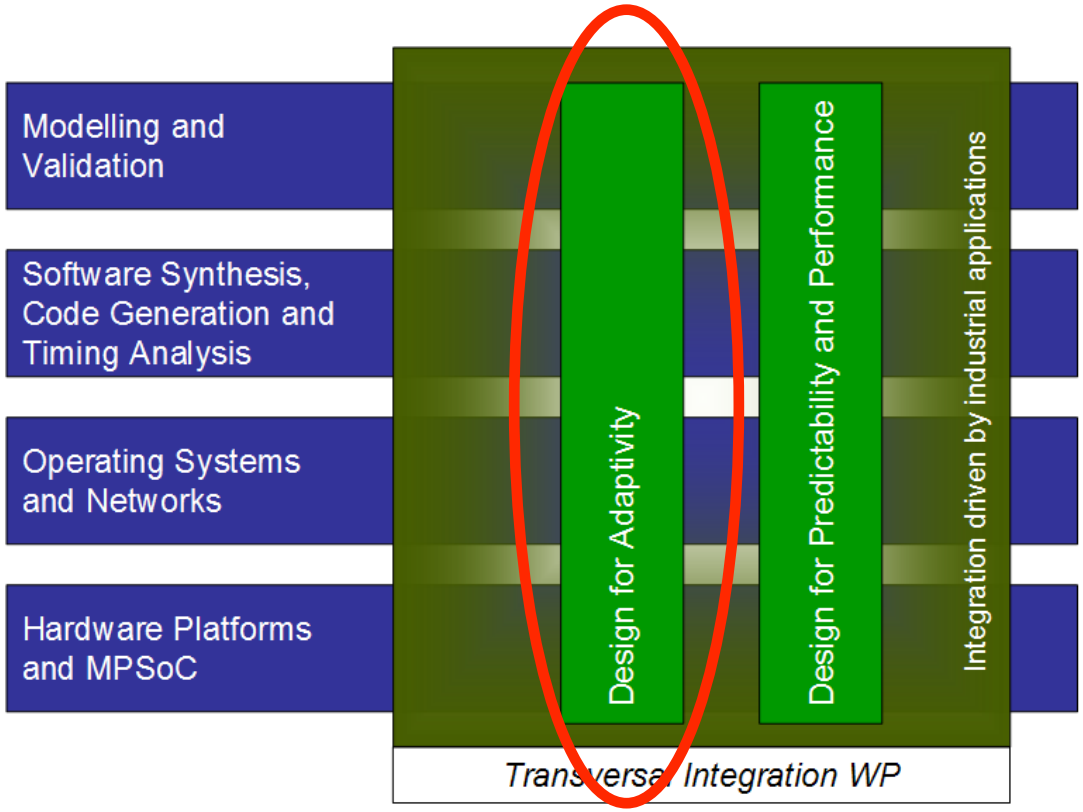
- Code Size
  - 15-20 Millions line of code
- 3-4 h build time
- Compiled into *one* program that runs from flash
- Around 100 threads with varying real-time criticality
- No static analysis
- Over-provisioning of resources to cater for worst-case not an option
- Many hundreds of parallel developers
- Certain time-critical parts hand-coded in machine language

Example: Cellular Phones Tomorrow

- Multimedia streaming and processing increasingly important
  - Multiple simultaneous streams
- Large dynamic variations in use cases and QoS demands
  - Dynamic adaptation necessary
  - Performance and power consumption reasons
- More advanced processors, e.g. ARM11 (12)
  - Multicore for performance and power
  - Powerful and complex instruction sets
  - Generation of efficient code an even higher challenge than today
- Heterogeneous
  - OS (RTOS – Linux & Windows)
  - Hardware (ASICs, multicore, hardware accelerators)

ArtistDesign

- European Network of Excellence on Embedded System Design



Outline

- Embedded System Trends
- **Definitions**
- Adaptivity and Control
- Reconfigurable hardware
- Embedded Adaptivity Issues
- Three examples:
  - Feedback-Based Queue Length Control
  - Feedback Scheduling of Control Tasks
  - Adaptive Resource Management in ACTORS
- Conclusions

Definitions

"An embedded system is adaptive if it is able to adjust its internal strategies to meet its objectives"

Comment:

- *The adjustment is made in response to a change in, or increased knowledge about, the environment or platform*
- *The objective for the change is to maintain the system performance or service at a desired level*
- *That fact that the adjustment is performed at run-time is implicit in the definition*

Definitions

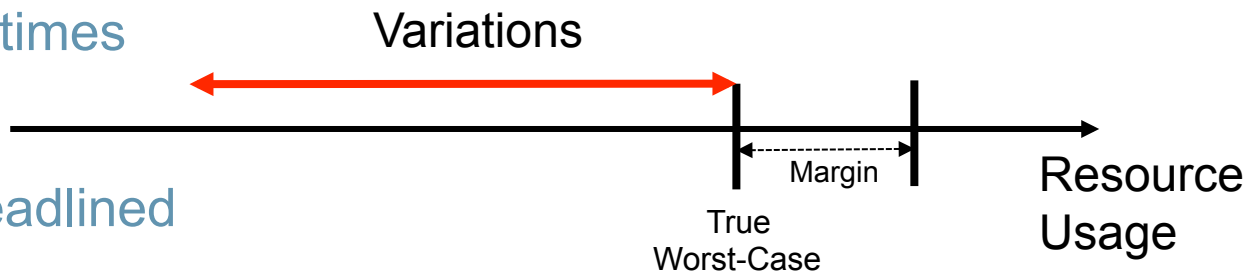
"An embedded system is **robust** if it meet its objectives under changing conditions without modifying its internal strategies"

"A **reconfiguration** is a change in the structure of the system"

- *Comment: A mechanism, among others, that could be used for achieving adaptivity*
- **"Flexibility** is a broader concept than adaptivity that, e.g., also covers off-line, design-time activities"

Sustainability

- The term ***sustainability*** was recently coined by Burns and Baruah to cover robustness in real-time scheduling towards ”benign” variations
  - Decreased execution time requirements
  - Later task arrival times
  - Smaller jitter
  - Larger relative deadlined



Outline

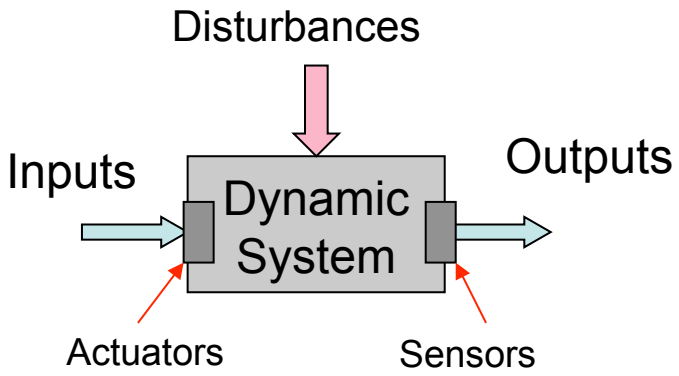
- Embedded System Trends
- Definitions
- **Adaptivity and Control**
- Reconfigurable hardware
- Embedded Adaptivity Issues
- Three examples:
  - Feedback-Based Queue Length Control
  - Feedback Scheduling of Control Tasks
  - Adaptive Resource Management in ACTORS
- Conclusions

Feedback and Adaptivity

- The need for adaptivity in embedded systems is often connected to the need to handle variability and uncertainties
- This is what feedback control is all about!!

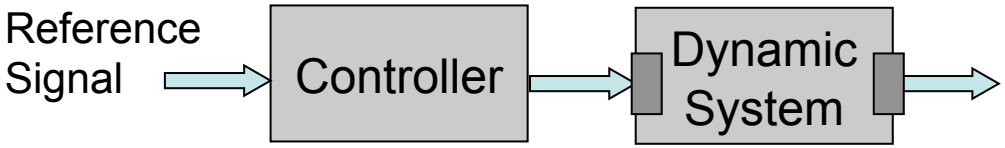
Adaptation in Control

- Feedback is one mechanism often proposed in the embedded system community to achieve adaptivity
- The control community has a somewhat different view on what adaptivity really means
- Some definitions
  - Dynamic system (process/plant)



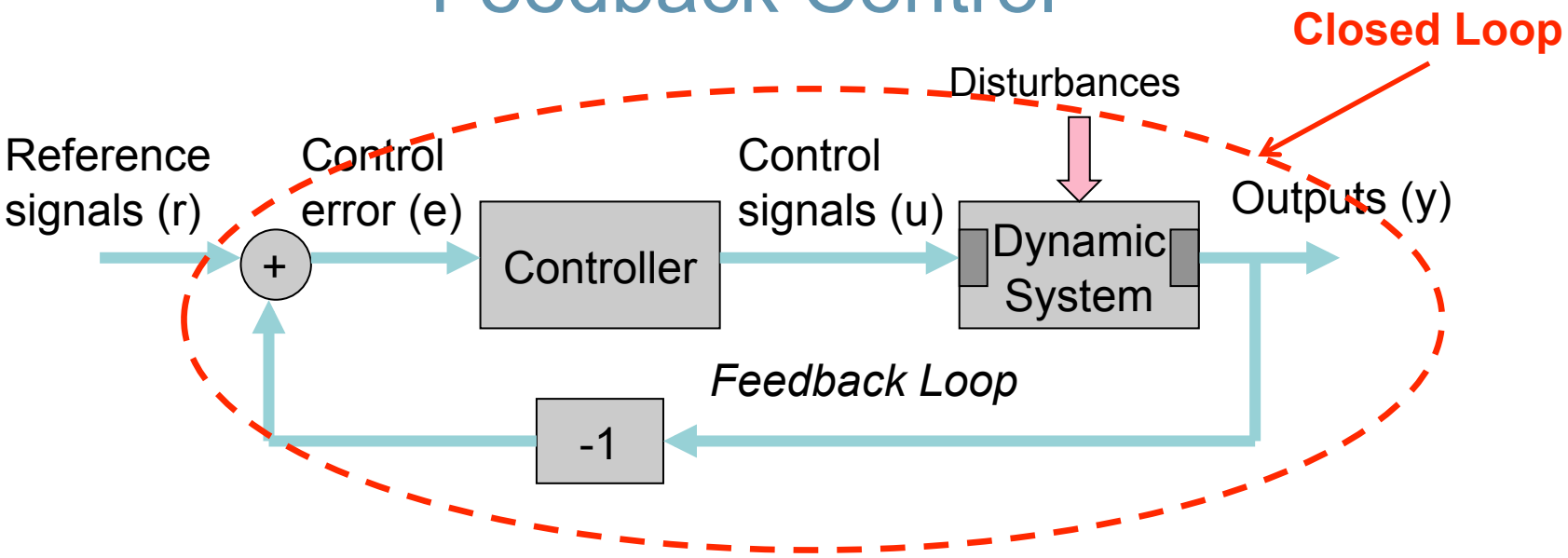
Feedforward Control

- Feedforward (open loop) control



  - Assumes perfect information (model) of the system
  - No disturbances (unless they are measured)

Feedback Control



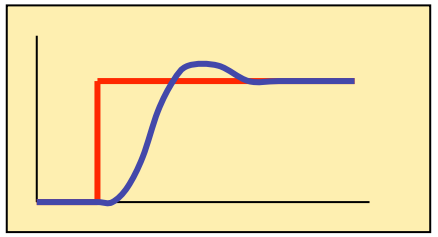
- Control algorithm, e.g.,
  - PID

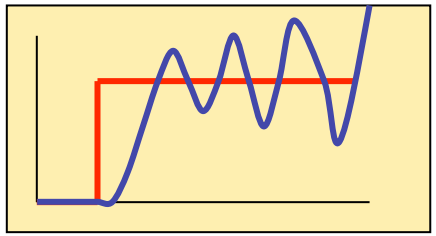
$$u(t) = K(e(t) + \frac{1}{T_I}\int e(s)ds + T_D \frac{de(t)}{dt})$$

  - **Fixed structure** and **constant parameters**

The Magic of Feedback

- Make a system behave as desired
- Maintain variables constant
- Stabilize an unstable system
- Reduce effects of disturbances and system variations

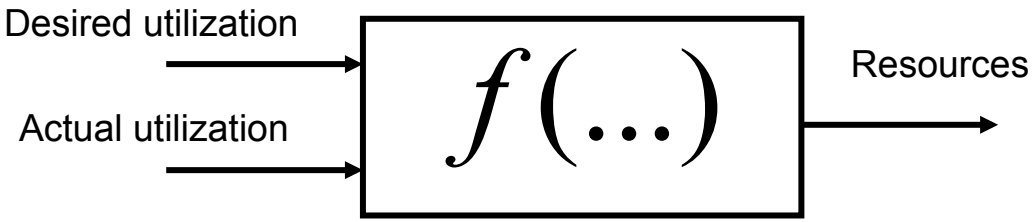




- Isn't this adaptivity?
  - Yes, in the general meaning of the word!
    - The closed loop system adapts to changing external conditions
  - Not in the control community!
    - The controller itself does not adapt.
    - Uses the same structure and parameters

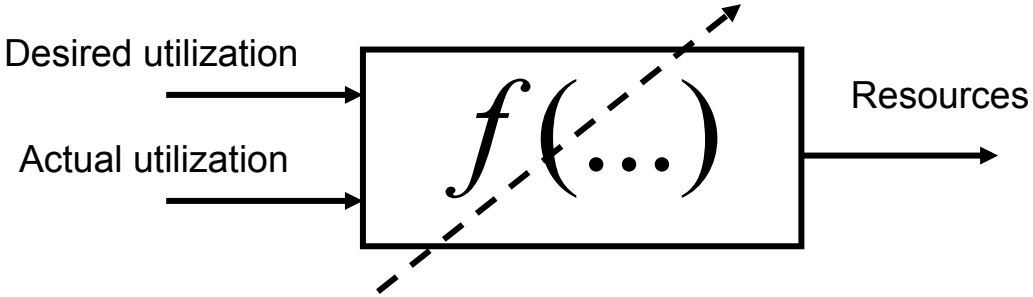
Adaptivity - Confusion

- Adaptivity in the CS/scheduling community



Adaptive Resource Management

- Adaptivity in the Control community

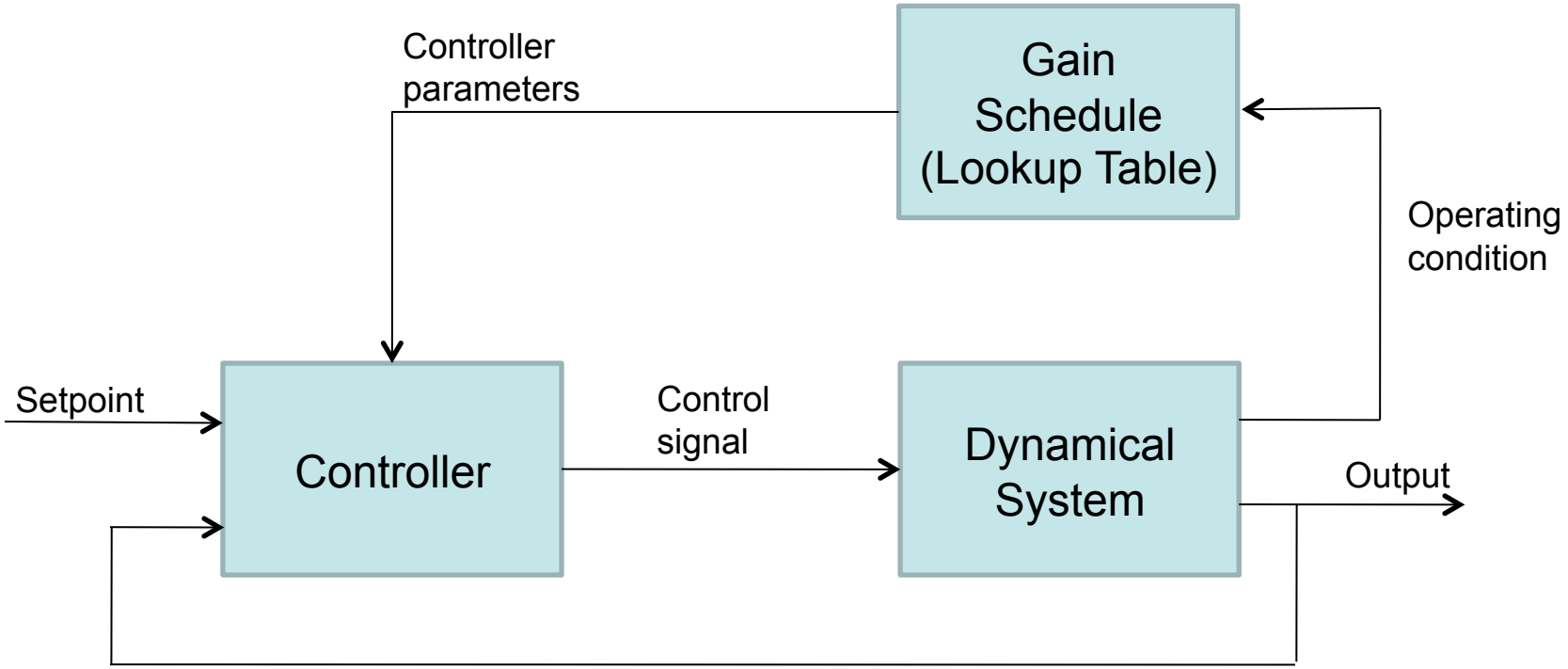


Adaptive Control

- In order for a controller to be adaptive the structure and/ or parameters should vary with the operating conditions
- In most cases only the parameters
  - Fixed structure controller with on-line adjustable parameters
- Adaptive control theory
  - Find parameter adjustment algorithms that offer global stability and convergence guarantees
- Main motivation:
  - Control of nonlinear and/or time-varying systems

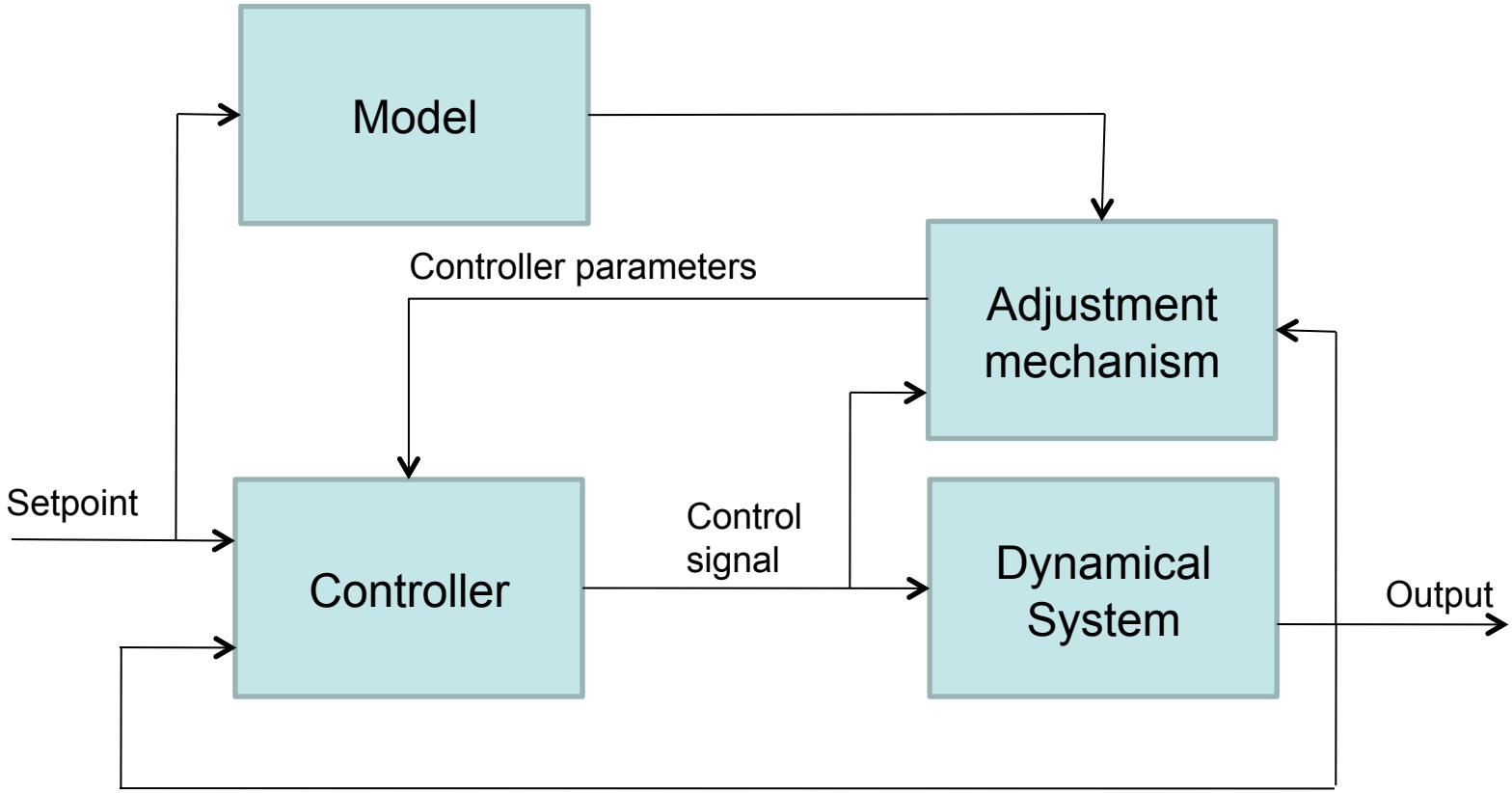
Adaptive Schemes

- Gain Scheduling:



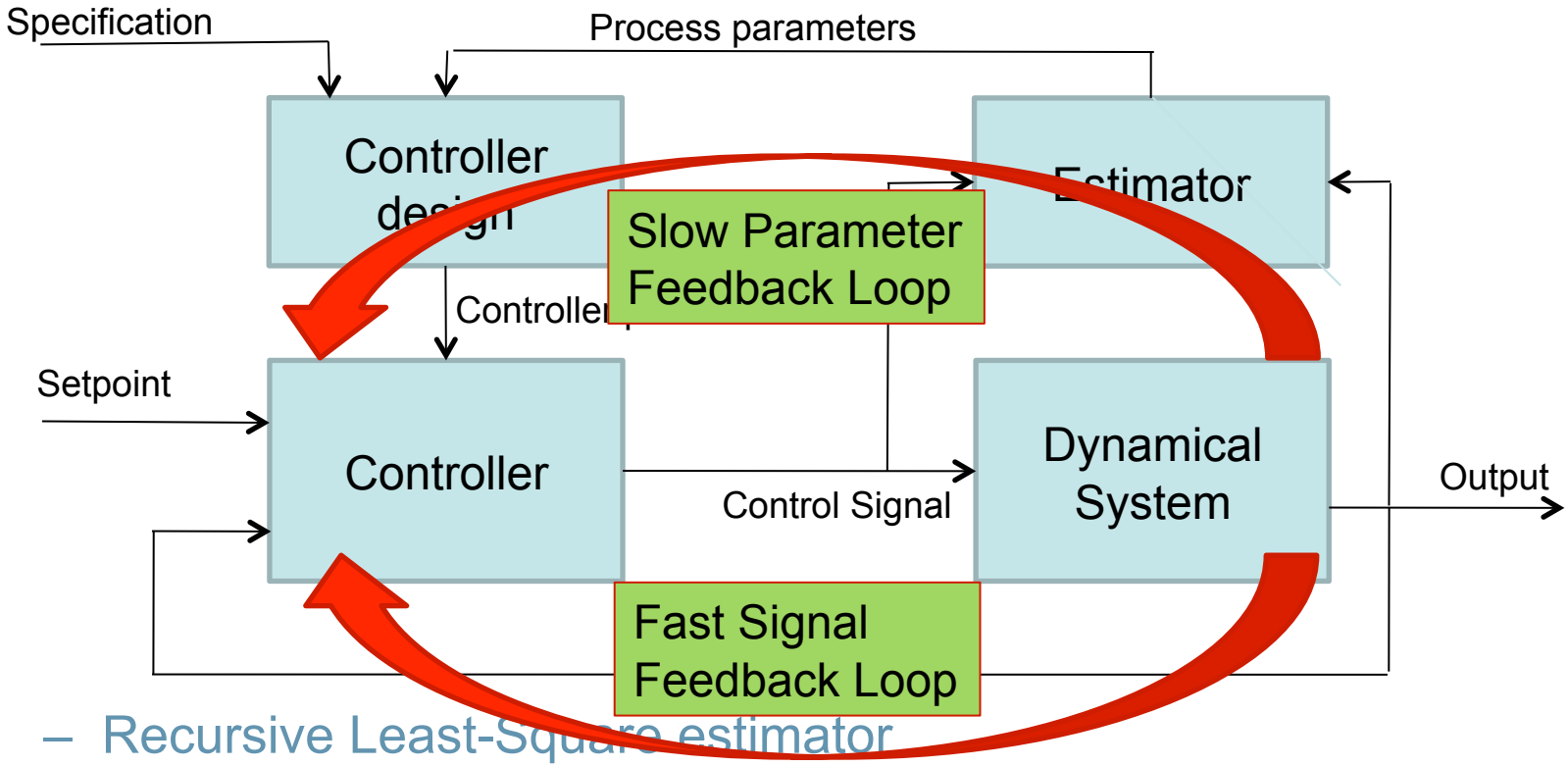
Adaptive Schemes

- Model Reference Adaptive System



Adaptive Schemes

- Self-Tuning Regulator



  - Recursive Least-Square estimator
  - Can be reparameterized to directly estimate the controller parameters

Non-Linear Adaptive Control

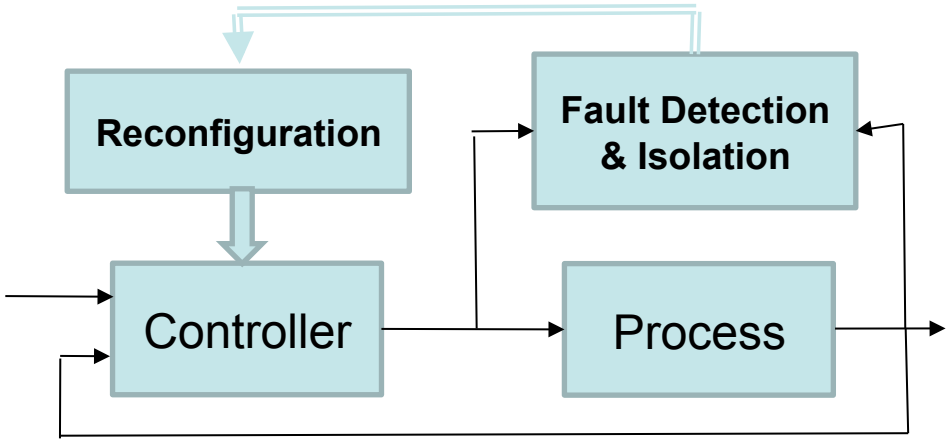
- Classical adaptive control assumes linear controllers with on-line adjustable parameters
- Main reason:
  - Linear control very powerful
- Nonlinear adaptive control
  - Neural networks
  - Radial basis functions
  - Fuzzy logic schemes
  - .....
- Structurally equivalent

Adaptive Control Confusion

- Also in control there is confusion about what adaptation is and is not
- A linear system with time-varying parameters can be viewed as nonlinear system with two types of states
  - Ordinary ”fast” states
  - Slow parameter states
- For example, when an Augmented Kalman filter is used to estimate both types of states simultaneously it is normally not considered as adaptive control
- **Therefore, in this context I will use the everyday meaning of adaptivity, i.e., include ordinary feedback!**

Reconfigurable Control

- A way of achieving fault-tolerant control
- Typically, actuator or sensor faults
- Reconfiguration by
  - Selecting new actuators and sensors
  - Changing the controller structure and/or parameters
- Motivation:
  - Flight control systems
    - Sensor and actuator redundancy

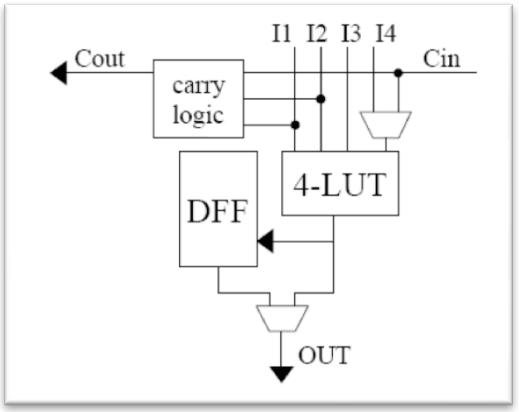


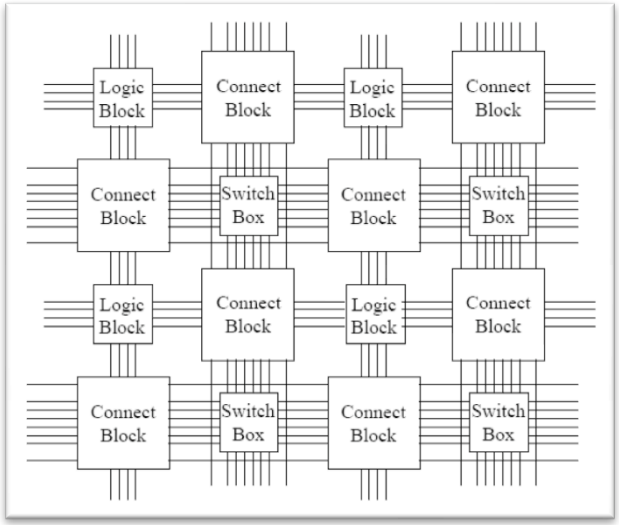
Outline

- Embedded System Trends
- Definitions
- Adaptivity and Control
- **Reconfigurable hardware**
- Embedded Adaptivity Issues
- Three examples:
  - Feedback-Based Queue Length Control
  - Feedback Scheduling of Control Tasks
  - Adaptive Resource Management in ACTORS
- Conclusions

Reconfigurable Computing

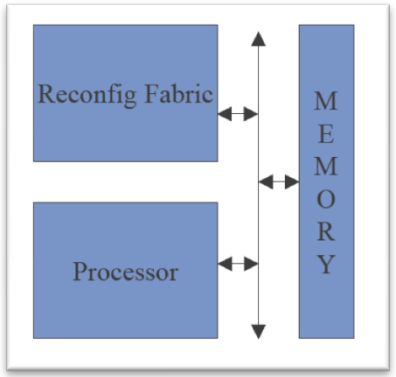
- Programmable Hardware
- FPGAs
- Programmable Logic Blocks
  - N-input Digital Lookup tables (LUT)
  - Programmable computing of any function of N inputs
- Programmable Interconnects
  - Routing between blocks
- Programmable IO



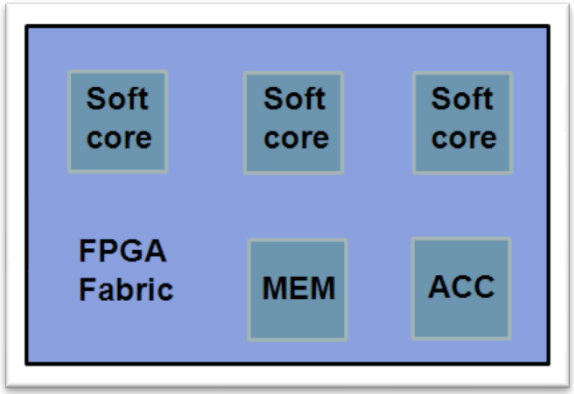


Computing Structures

- Standalone chip
  - Fine granularity
    - LUT blocks + regular interconnect structures
  - Coarse granularity
    - Path widths > 1 bit
    - More powerful blocks, e.g., ALUs, registers, small processors



- As a coprocessor to an ordinary processor
  - Reconfigurable hardware accelerator
- As a reconfigurable fabric containing
  - processor cores,
  - memory,
  - fine or coarse-grained FPGAs
  - ....



Soft Cores

- Hard core
  - Dedicated silicon on the FPGA
  - Similar speed to a discrete processor core
- Soft core
  - Implemented entirely in the logic primitives of the FPGA
  - Slower, but reconfigurable!
    - Peripherals (e.g., memory controllers, timers, counters, UARTs, bus interconnects, ...)
    - Core
      - Cache architecture
      - Pipeline stages
      - Instruction set (cp. Microcode)

Run-Time Reconfigurability

- Swap different hardware configurations in and out during execution
- ”Virtual hardware” customised for different stages of the application
- Allows a larger part of an application to be accelerated than what fits in a non run-time reconfigurable system
- Single context device
  - Requires a comple reconfiguration
  - Traditional FPGA
- Multi-context device
  - Fast context switches (nanoseconds)
- Partially reconfigurable device

Outline

- Embedded System Trends
- Definitions
- Adaptivity and Control
- Reconfigurable hardware
- **Embedded Adaptivity Issues**
- Three examples:
  - Feedback-Based Queue Length Control
  - Feedback Scheduling of Control Tasks
  - Adaptive Resource Management in ACTORS
- Conclusions

Adaptivity versus Predictability and Dependability

- The relation between adaptivity and dependability and predictability is interesting
- Ideally, all changes of a system due to adaptation should be predictable and shouldn't jeopardize dependability.
- However, in many cases adaptivity increases the risk of non-predictable behavior.
- On the other hand adaptivity can also be a prerequisite for dependability.
- Tradeoff between:
  - Dependability
  - Predictability
  - Adaptivity
  - Performance

Problems of Adaptivity

Adaptivity can introduce new problems:

- The adaptation mechanism itself consumes resources
- Harder to provide formal guarantees about the system
- Adds to the complexity
- May complicate the design process
  - Design space grows
- Requires tuning
- Bad tuning might lead to oscillations (stability problems)
- Sensors and actuators are necessary

Adaptivity Issues

- Adaptivity in system modelling – how is adaptivity modelled
- Efficient adaptation – how can adaptation mechanisms be made resource efficient
- Frameworks for adaptivity – unified frameworks for adaptivity (negotiation, contracts, QoS)
  - FRESCOR, ACTORS
- Predictable and dependable adaptivity – what types of formal guarantees concerning predictability and dependability can be stated for an adaptive system
- “Controlled adaptivity”
  - How do we ensure that a system only adapts within certain limits?
  - If everything is foreseen at design-time, could it still be considered as adaptivity?

Adaptivity Issues

- Verification and testing of adaptive system
- Adaptivity from an application's point of view – how should the adaptation mechanisms be exposed to the application developers (APIs etc)
- Interface between software and hardware
- Hardware based systems – How do model adaptivity?
- Run-Time reconfigurable hardware – How to use it to improve adaptivity

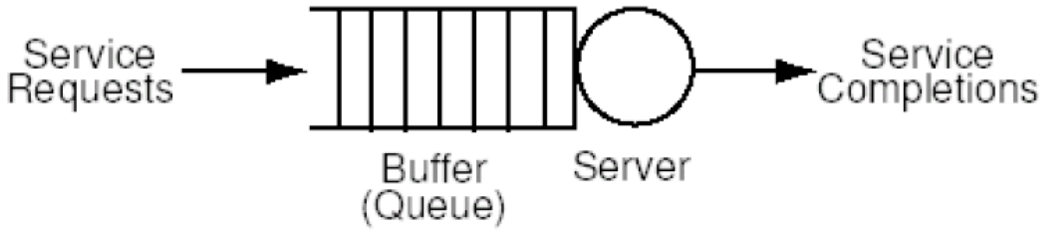
Outline

- Embedded System Trends
- Definitions
- Adaptivity and Control
- Reconfigurable hardware
- Embedded Adaptivity Issues
- Three examples:
  - **Feedback-Based Queue Length Control**
  - Feedback Scheduling of Control Tasks
  - Adaptive Resource Management in ACTORS
- Conclusions

Objective

- Show that classical linear continuous-time control methods are applicable also to embedded computing applications

Control of Queuing Systems



Work requests (customers) arrive and are buffered

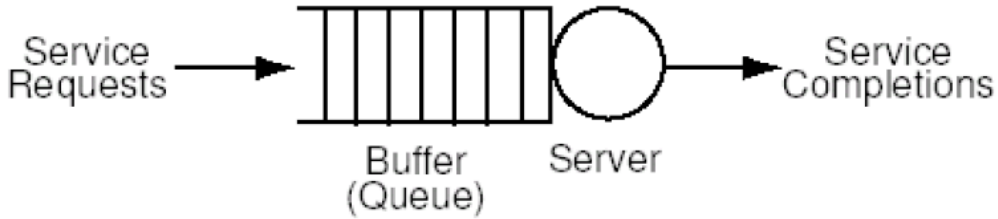
Service level objectives (e.g., response time for request belonging to class X should be less than Y time units)

Reduce the delay caused by other requests, i.e., adjust the buffer size and redirect or block other requests

Admission control

Queue Length Control

Assume an M/M/1 queuing system:

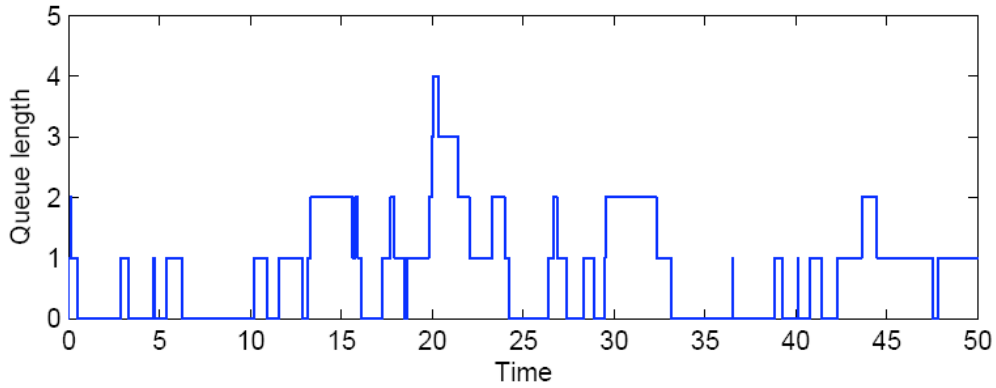


- Random arrivals (requests), Poisson distributed with average λ per second
- Random service times, exponentially distributed with average $1/\mu$
- Queue containing x requests

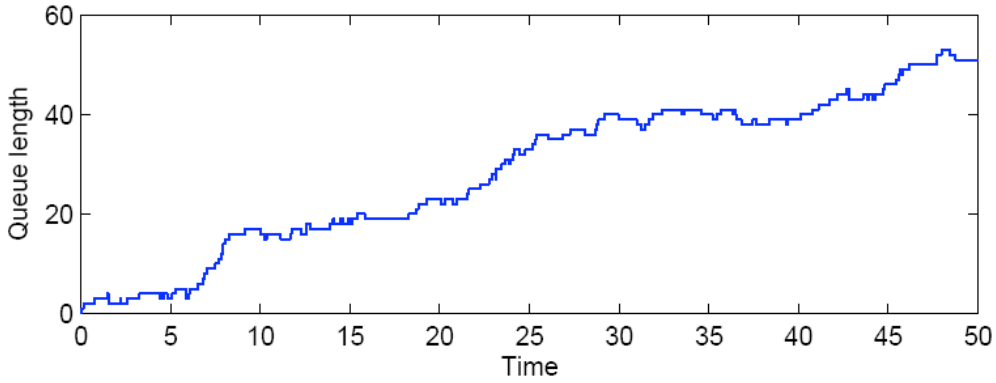
Intuition: $x \to \infty$ if $\lambda > \mu$

Simulation

$\lambda = 0.5,\ \mu = 1$:



$\lambda = 2.0,\ \mu = 1$:



Queue Length Control: Model

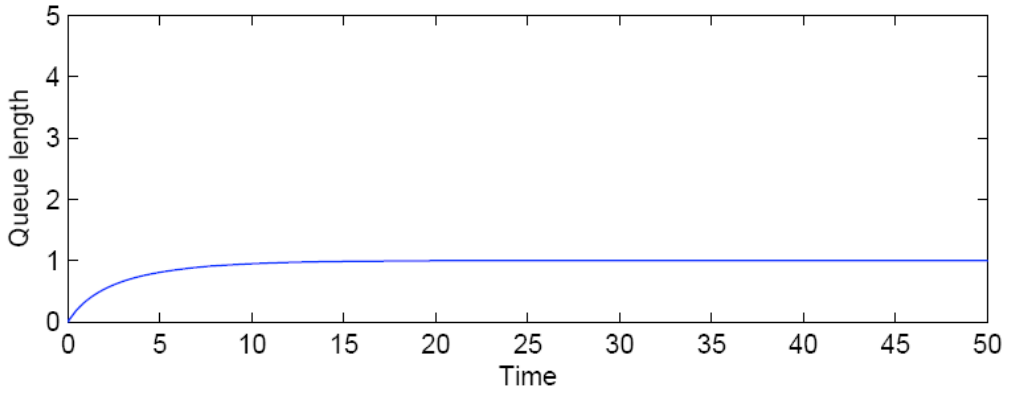
Approximate the system with a nonlinear flow model (Tipper's model from queuing theory)

The expectation of the future queue length x is given by

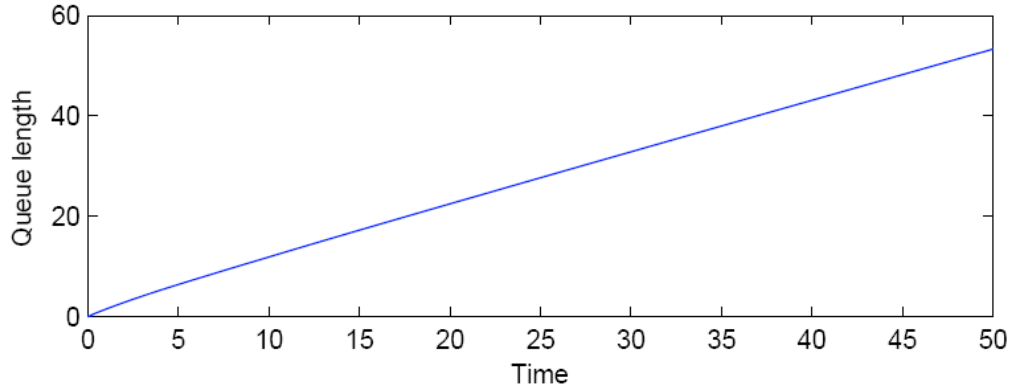
$$\dot{x} = \lambda - \mu \frac{x}{x+1}$$

Queue Length Control: Model

$\lambda = 0.5,\ \mu = 1$:



$\lambda = 2.0,\ \mu = 1$:



Queue Length Control: Control Signal

Control the queue length by only admitting a fraction u (between 0 and 1) of the requests

$$\dot{x} = \lambda u - \mu \frac{x}{x+1}$$

Admission control

Linearization

Linearize around $x = x^\circ$

Let $y = x - x^\circ$

$$\dot{y} = \lambda y - \mu \frac{1}{(x^\circ + 1)^2} y = \lambda u - \mu a y$$

Proportional Control

$$u = K(r - y)$$

$$\dot{y} = \lambda K(r - y) - \mu a y$$

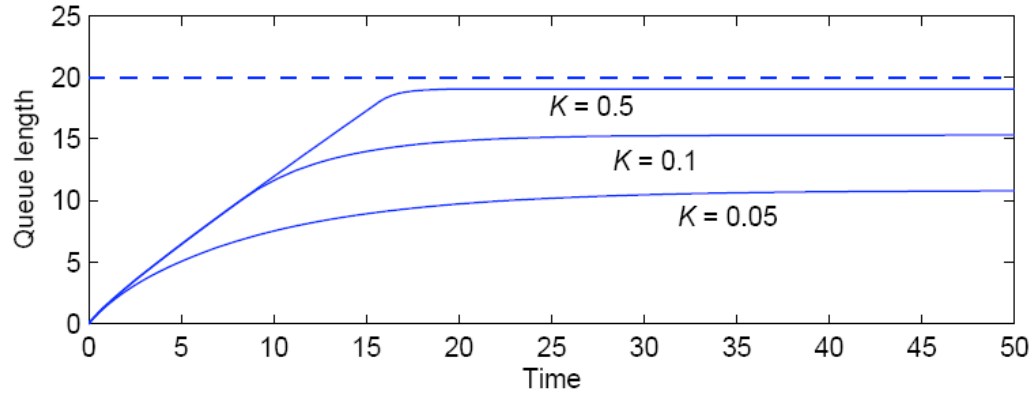
$$(s + \lambda K + \mu a)Y(s) = \lambda K R(s)$$

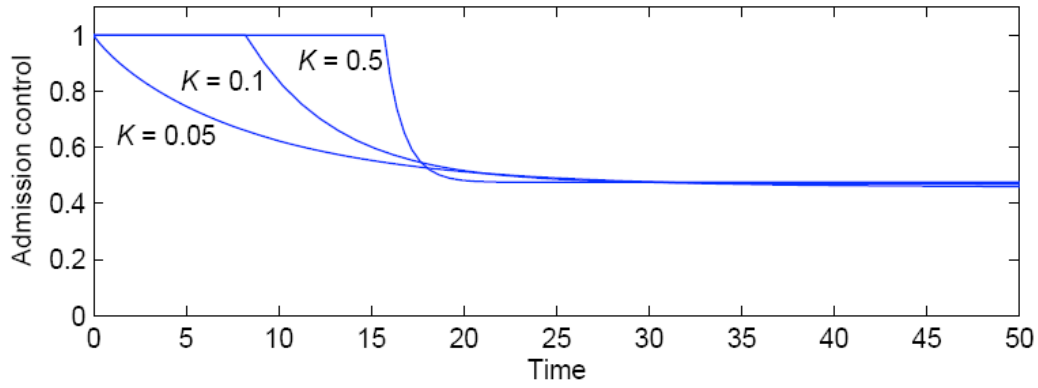
$$G_{cl}(s) = \frac{\lambda K}{s + \lambda K + \mu a}$$

With K the closed loop pole can be placed arbitrarily

Proportional Control

Simulations for $\lambda = 2, \mu = 1, x^\circ = 20$ and different values of K





Proportional + Integral (PI) Control

$$G_P(s) = \frac{\lambda}{s + \mu a}$$

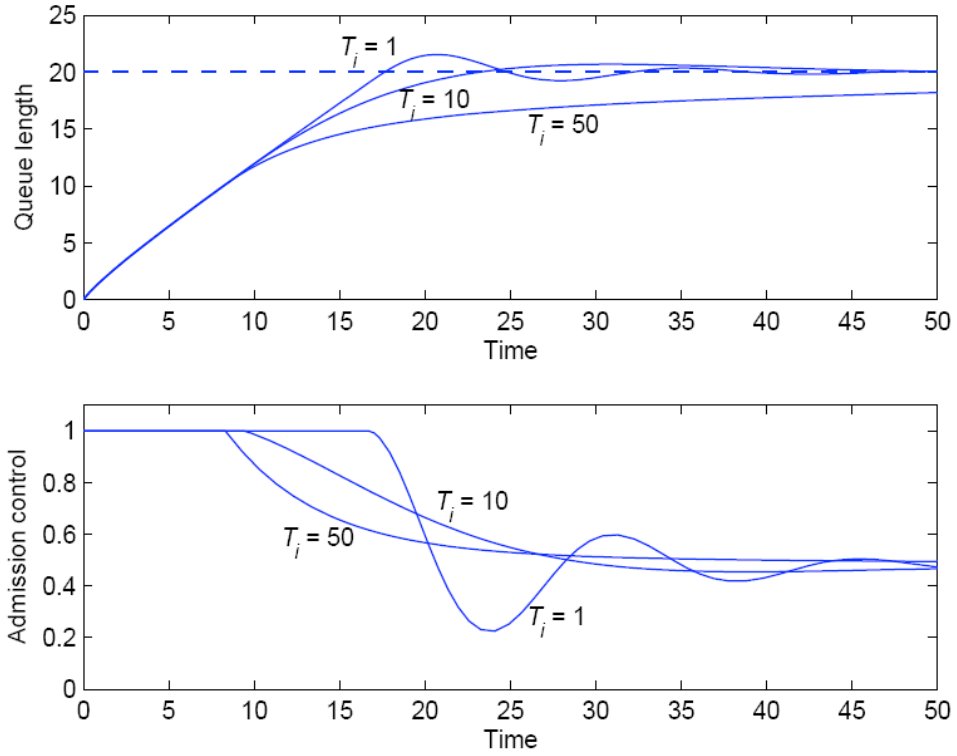
$$G_R(s) = K(1 + \frac{1}{sT_i})$$

$$G_{cl}(s) = \frac{G_P G_R}{1 + G_P G_R} = \frac{\lambda K(s + \frac{1}{T_i})}{s(s + \mu a) + \lambda K(s + \frac{1}{T_i})}$$

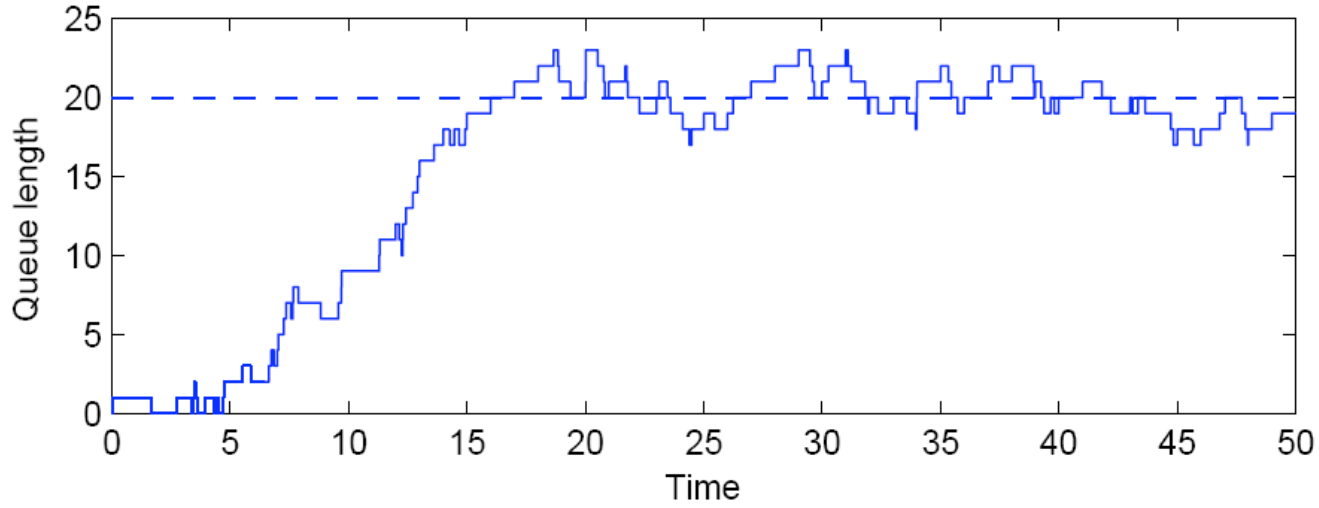
With K and T_i the closed loop poles can be placed arbitrarily

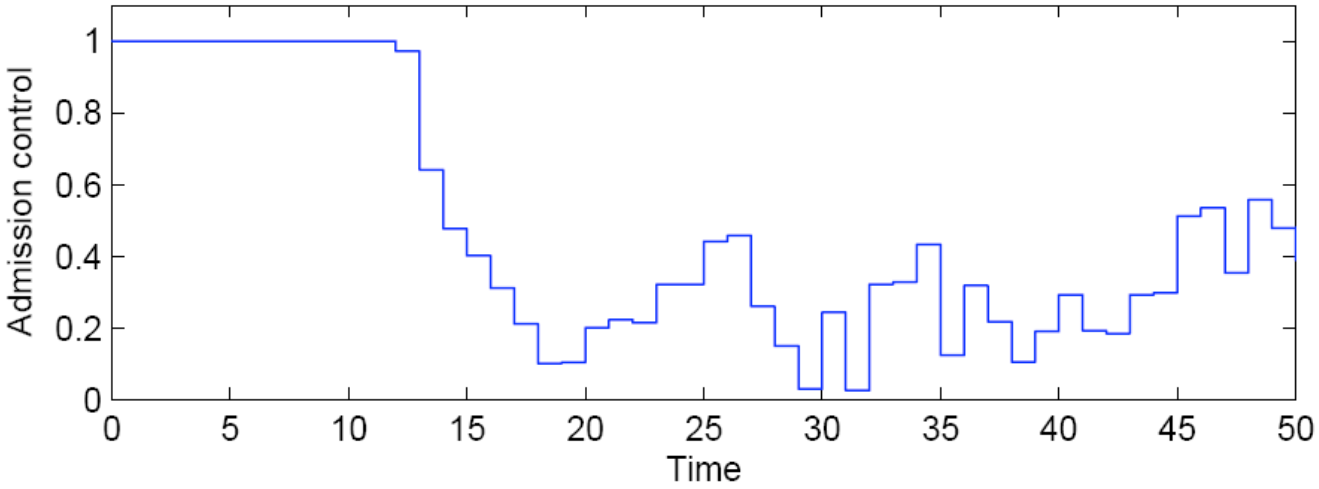
Proportional + Integral (PI) Control

Simulations for $\lambda = 2, \mu = 1, x^\circ = 20, K = 0.1$ and different values of T_i



PI Control of Queue Simulation





Conclusions

- Classical linear control techniques can be applied in certain cases
  - Time-triggered control
- However, in most cases event-based control is more natural
- It is only occasionally that continuous-time (flow) models are applicable
- Modeling of (embedded) computing systems is a general problem and challenge
  - No first-principles models
  - Discrete event-based models on the microscopic level
  - Transformed to continuous-time through averaging over moving time windows

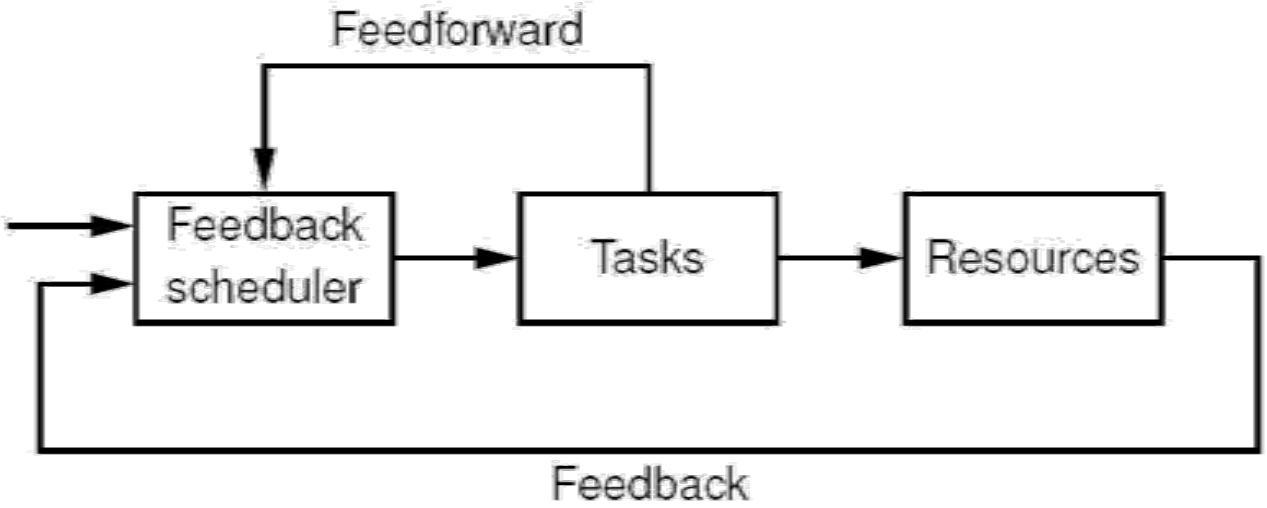
Outline

- Embedded System Trends
- Definitions
- Adaptivity and Control
- Reconfigurable hardware
- Embedded Adaptivity Issues
- Three examples:
  - Feedback-Based Queue Length Control
  - **Feedback Scheduling of Control Tasks**
  - Adaptive Resource Management in ACTORS
- Conclusions

Objective

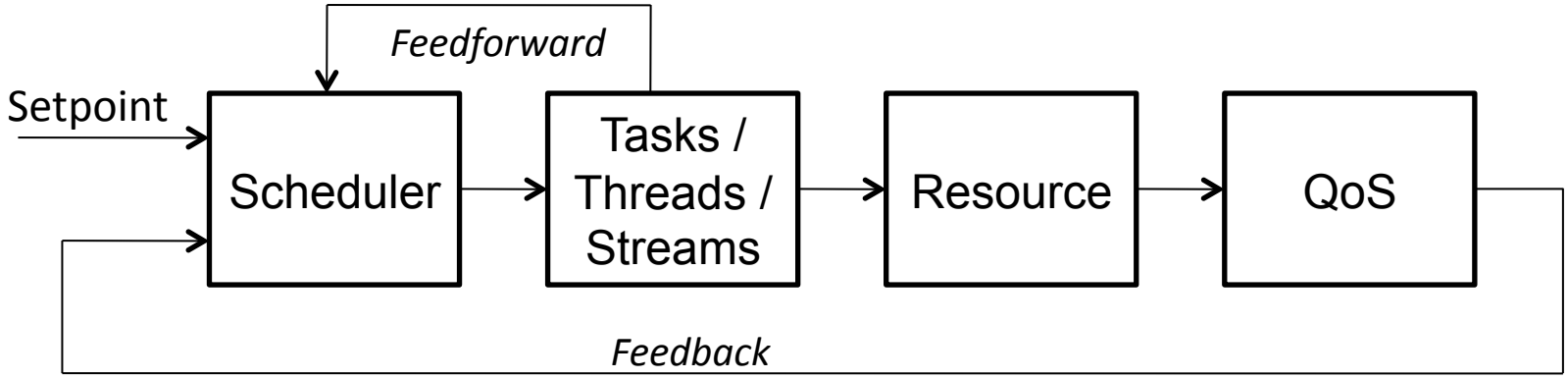
- Show how task sets with varying execution time demands can be handled by a combination of feedback and feedforward-based scheduling

Feedback Scheduling



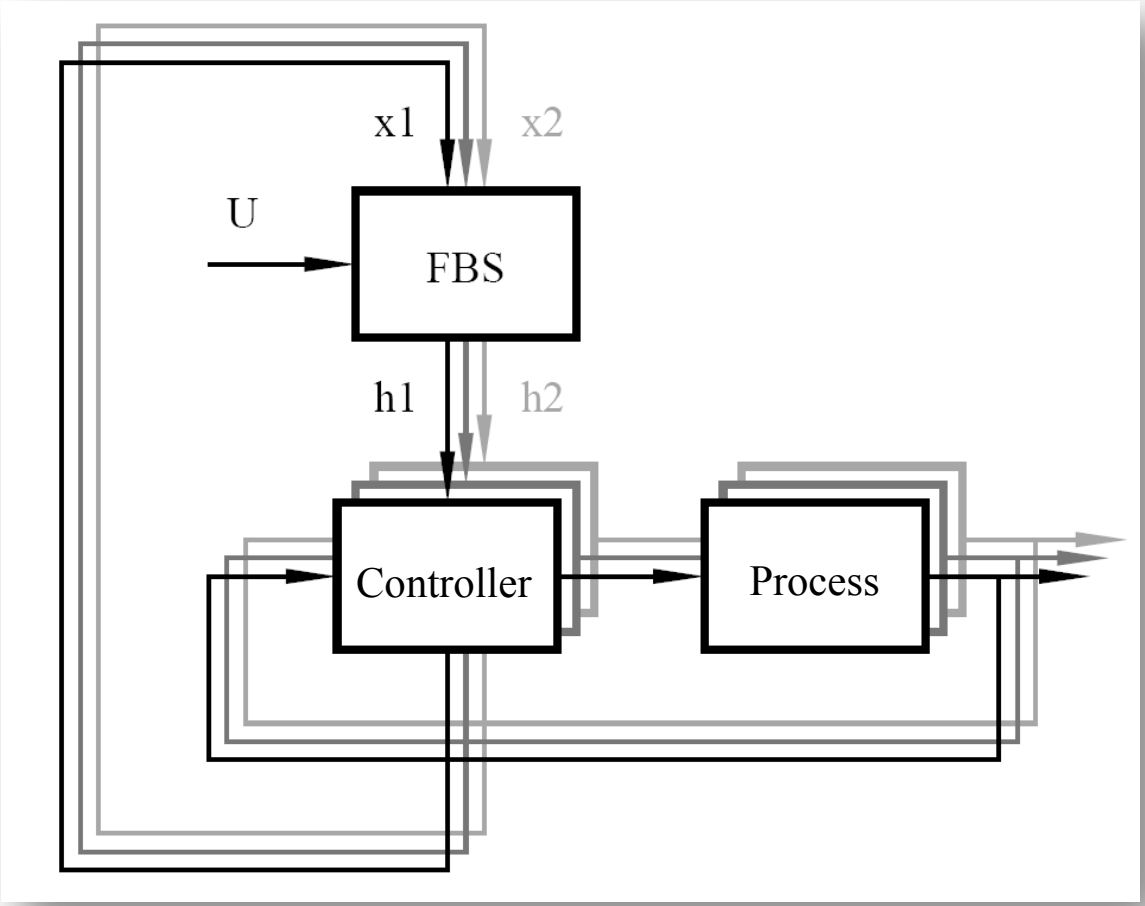
- Dynamic feedback-based resource allocation
- Adjust sampling periods and/or execution time demands
- Potential for higher resource utilization and increased control performance

Alternative Structure



- Feedback to handle uncertainties and disturbances
  - Unknown worst-case resource utilization
  - Load variations
- Feedforward to handle known changes in resource utilization

Feedback Scheduling of Feedback Controllers



On-Line Adjustment of Sampling Rates

- Idea: On-line adjust sampling rates of a set of controllers to maximize CPU utilization and hence performance.
- Assume that nominal sampling periods h_{nom} are wisely chosen
- On-line estimate the total utilization U
- Periodically assign new sampling periods to meet the utilization setpoint U_{sp}:

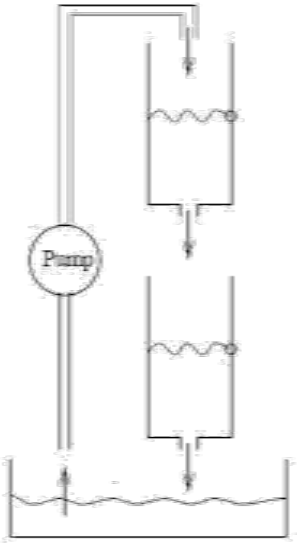
$$h_{\text{new}} = \frac{h_{\text{nom}} U}{U_{\text{sp}}}$$

- Possibly add feedforward to help with the estimation of U

Case Study:

The double-tank process:

Use pump, $u(t)$, to control level of lower tank, $y(t)$



Hybrid control strategy:

- PID control in steady state
- Optimal control for setpoint changes

PID Controller

$$
\begin{aligned}
P(t) &= K(y_{sp}(t) - y(t)) \\
I(t) &= I(t-h) + a_i(y_{sp}(t) - y(t)) \\
D(t) &= a_d D(t-h) + b_d(y(t-h) - y(t)) \\
u(t) &= P(t) + I(t) + D(t)
\end{aligned}
$$

Average execution time: $C = 2.0$ ms

Time-Optimal Controller

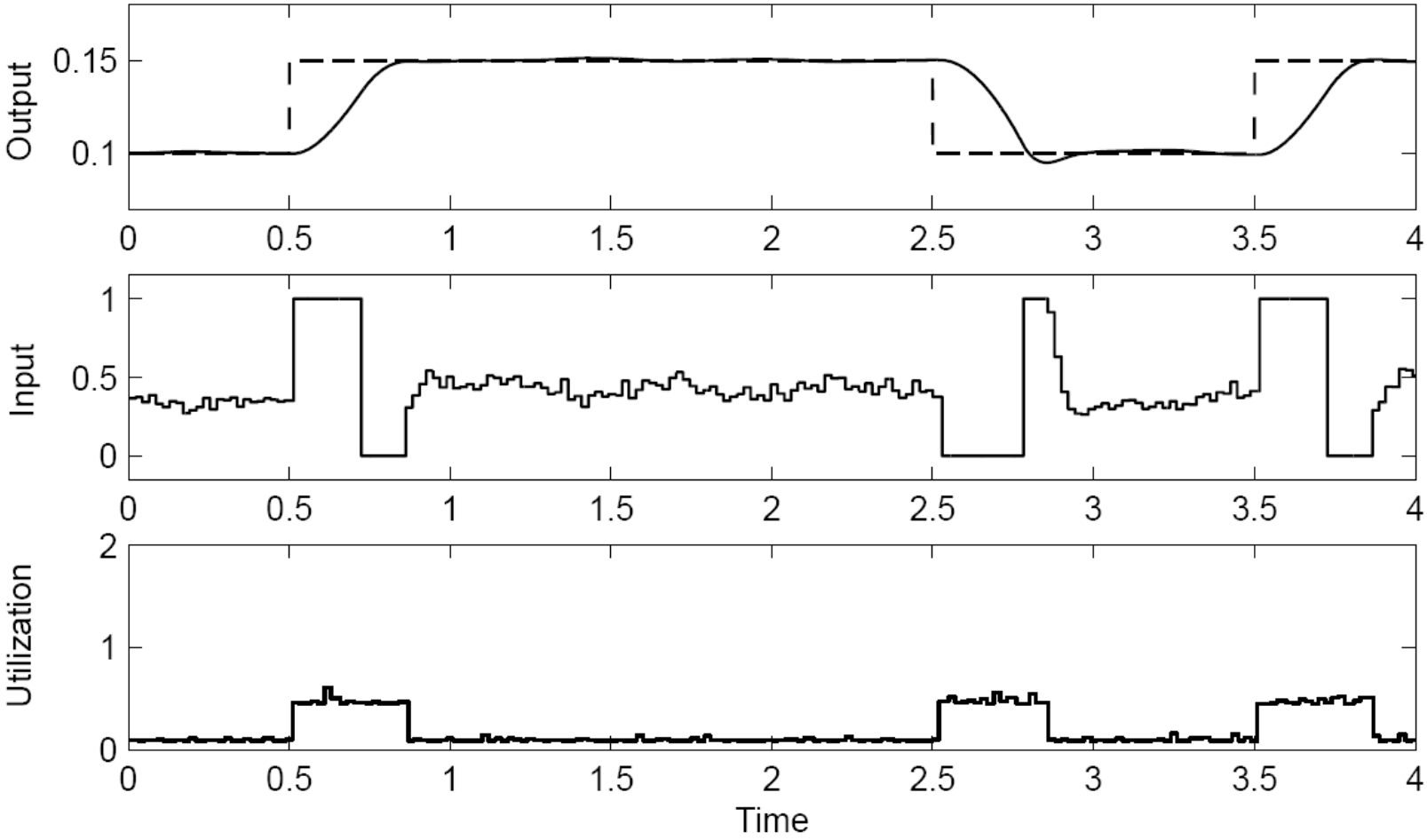
Computation of switching criterion:

$$x_2(x_1) = \frac{1}{a}((ax_1 - b\overline{u})(1 + \ln(\frac{ax_1^R - b\overline{u}}{ax_1 - b\overline{u}})) + b\overline{u})$$

$$V_{close} = \begin{bmatrix} x_1^R - x_1 \\ x_2^R - x_2 \end{bmatrix}^T P(\theta, \gamma) \begin{bmatrix} x_1^R - x_1 \\ x_2^R - x_2 \end{bmatrix}$$
$$+ \text{ more } \ldots$$

Average execution time: $C = 10.0$ ms

Nominal Performance, h = 21 ms



Experimental Setup

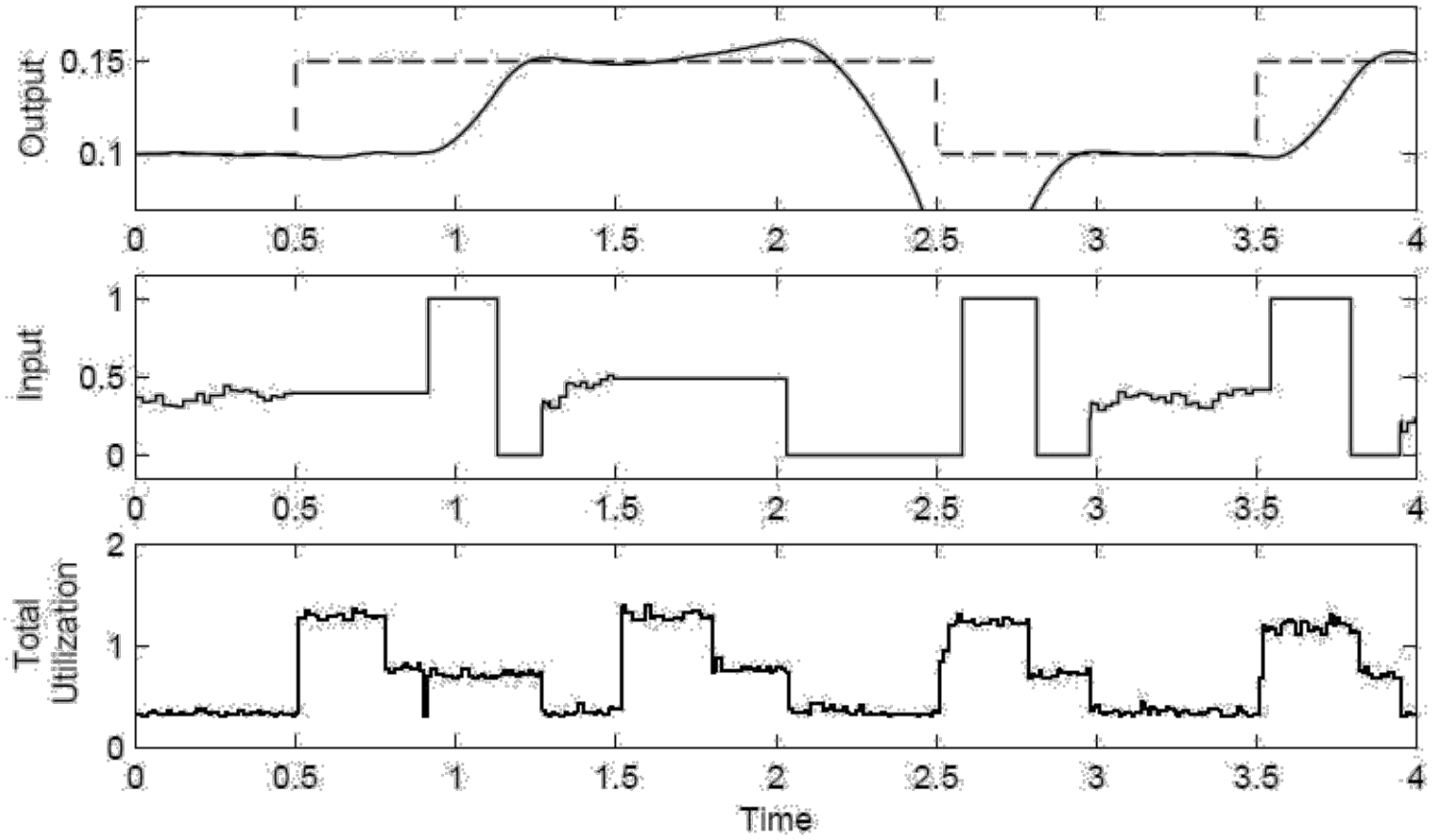
- Three hybrid controllers execute on one CPU
- Nominal sampling periods: $(h_1, h_2, h_3) = (21, 18, 15)$ ms
- Potential problem: All controllers in Optimal mode $\Rightarrow$ $U = \sum \frac{C}{h} = 170\%$

Compare strategies:

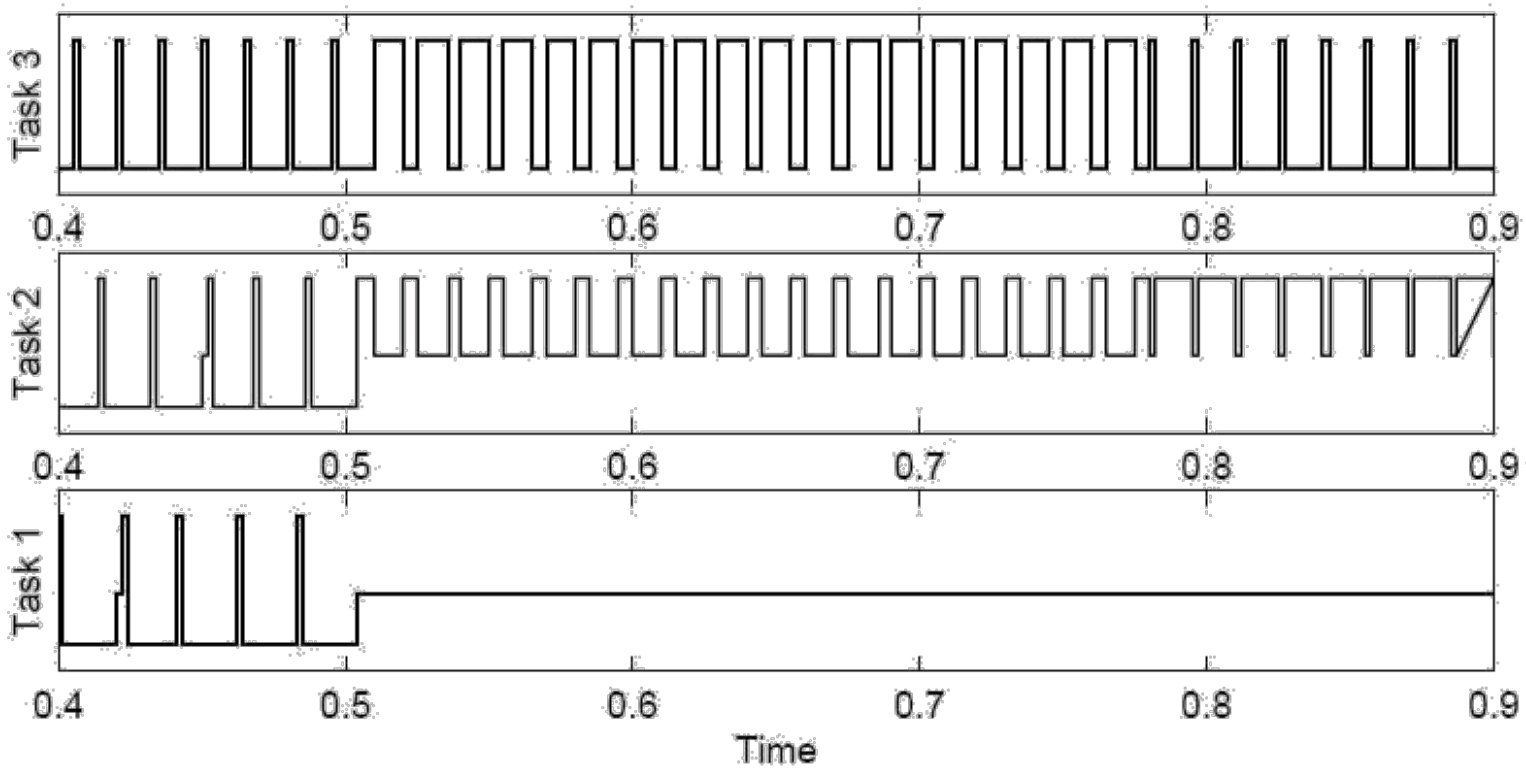
1. Open-loop scheduling
2. Feedback scheduling
3. Feedback + feedforward scheduling

- Co-simulation of scheduler, controllers, and double tanks
- Focus on the lowest-priority controller

Open Loop Scheduling



Open Loop Schedule

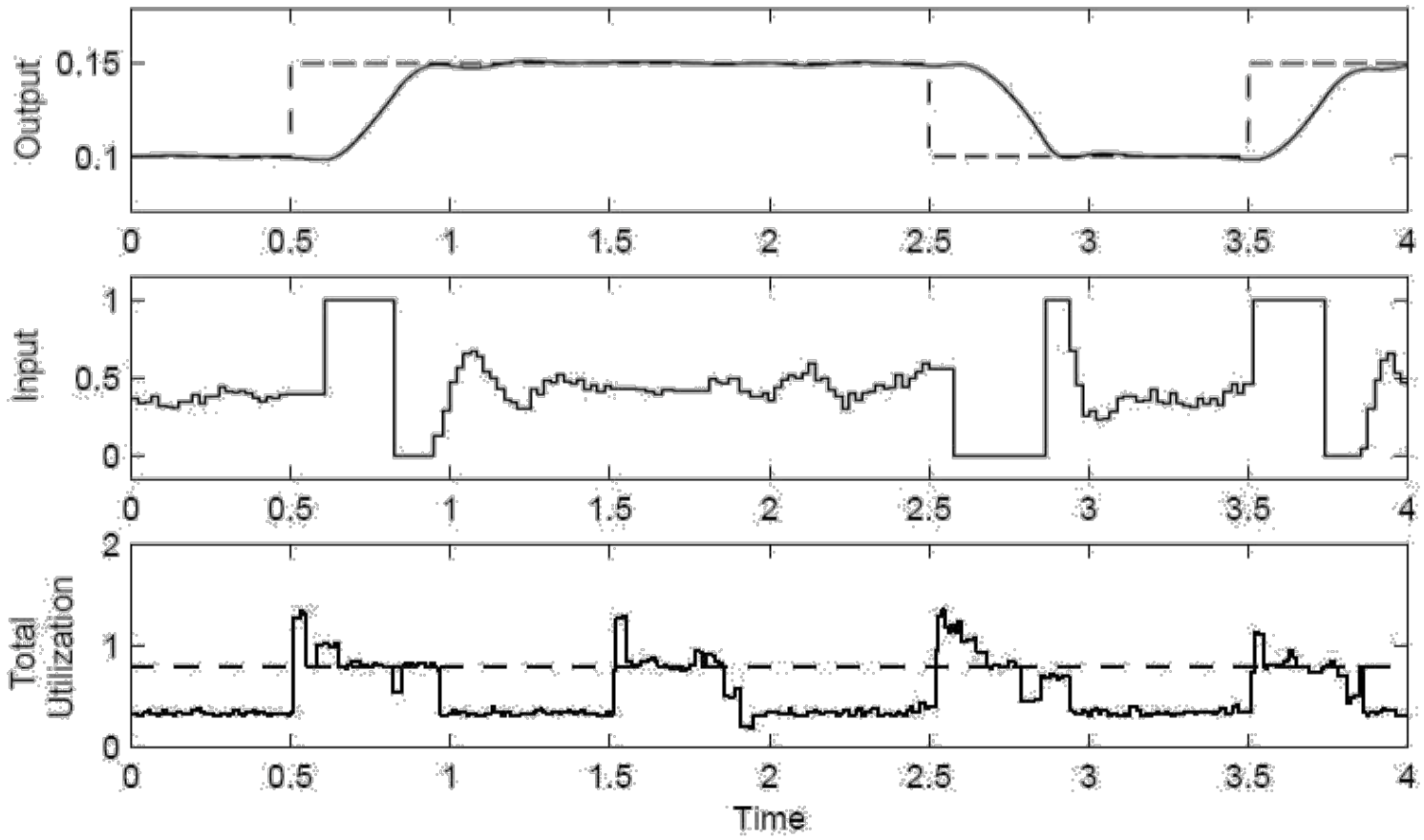


Feedback Scheduler

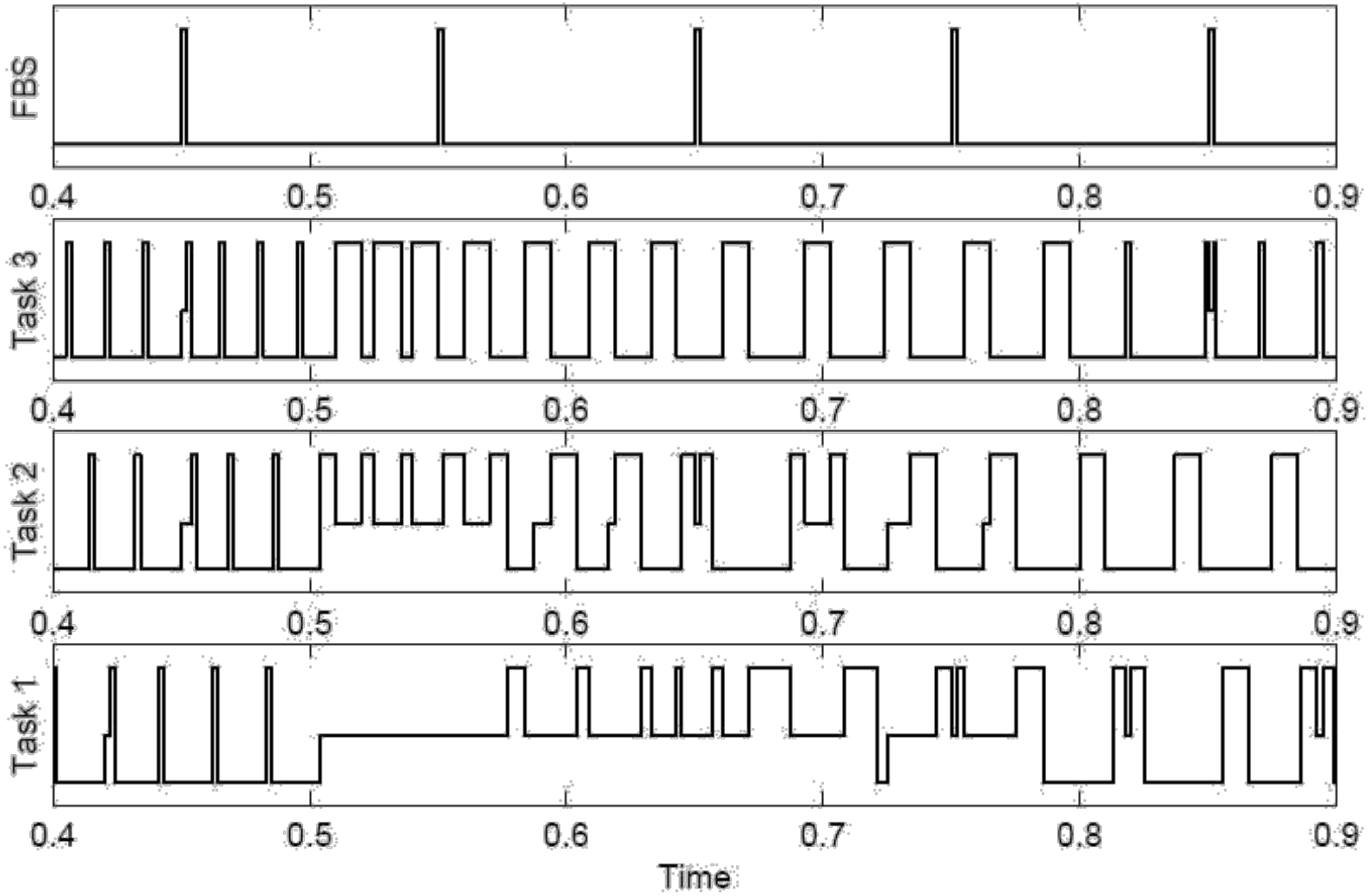
- A high-priority task, $T_{FBS} = 100$ ms, $C_{FBS} = 2$ ms
- Setpoint: $U_{sp} = 80\%$
- Estimate execution times using first-order filters
- Control U by adjusting the sampling periods:

$$h_{\text{new}} = \frac{h_{\text{nom}} U}{U_{\text{sp}}}$$

Feedback Scheduling



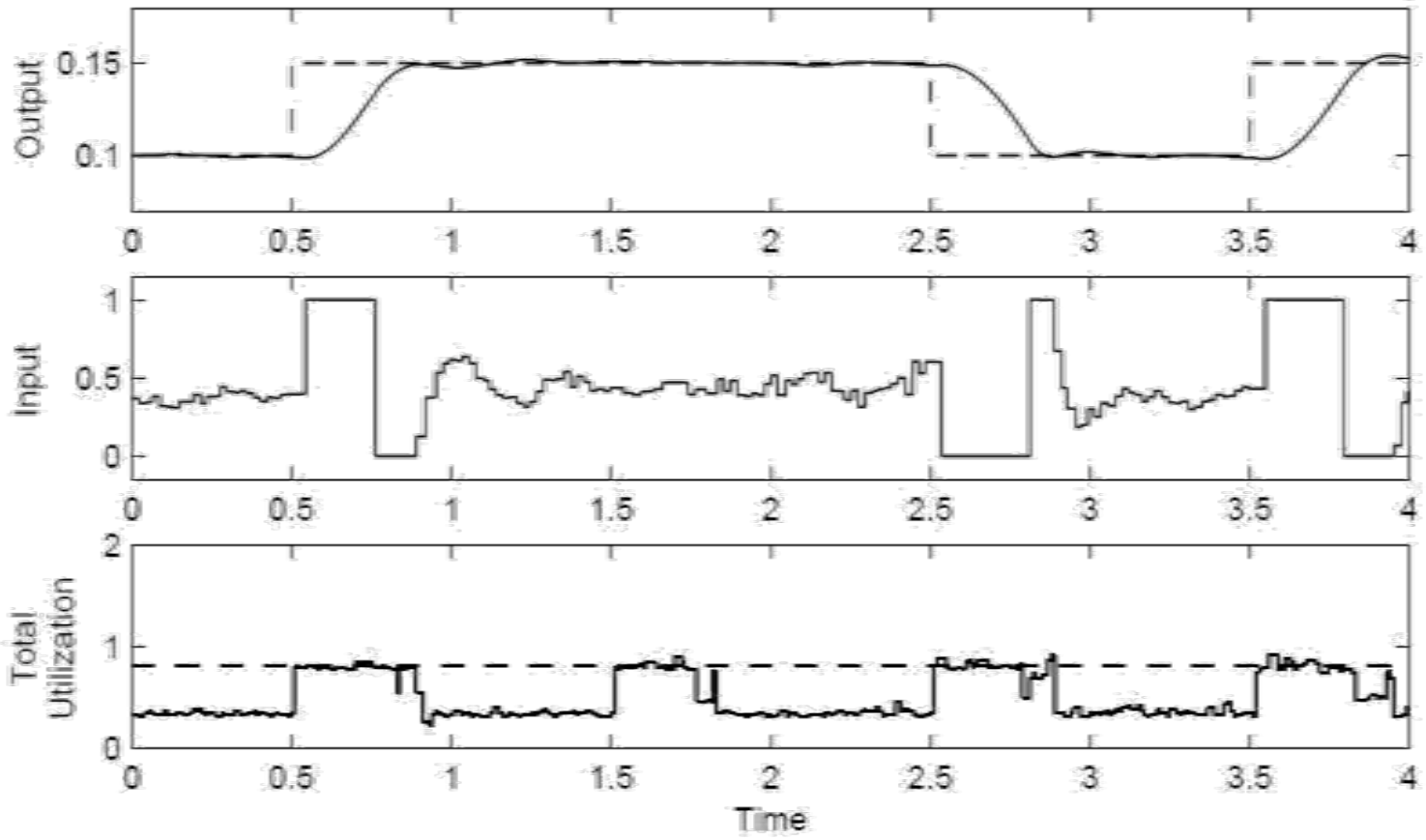
Feedback Schedule



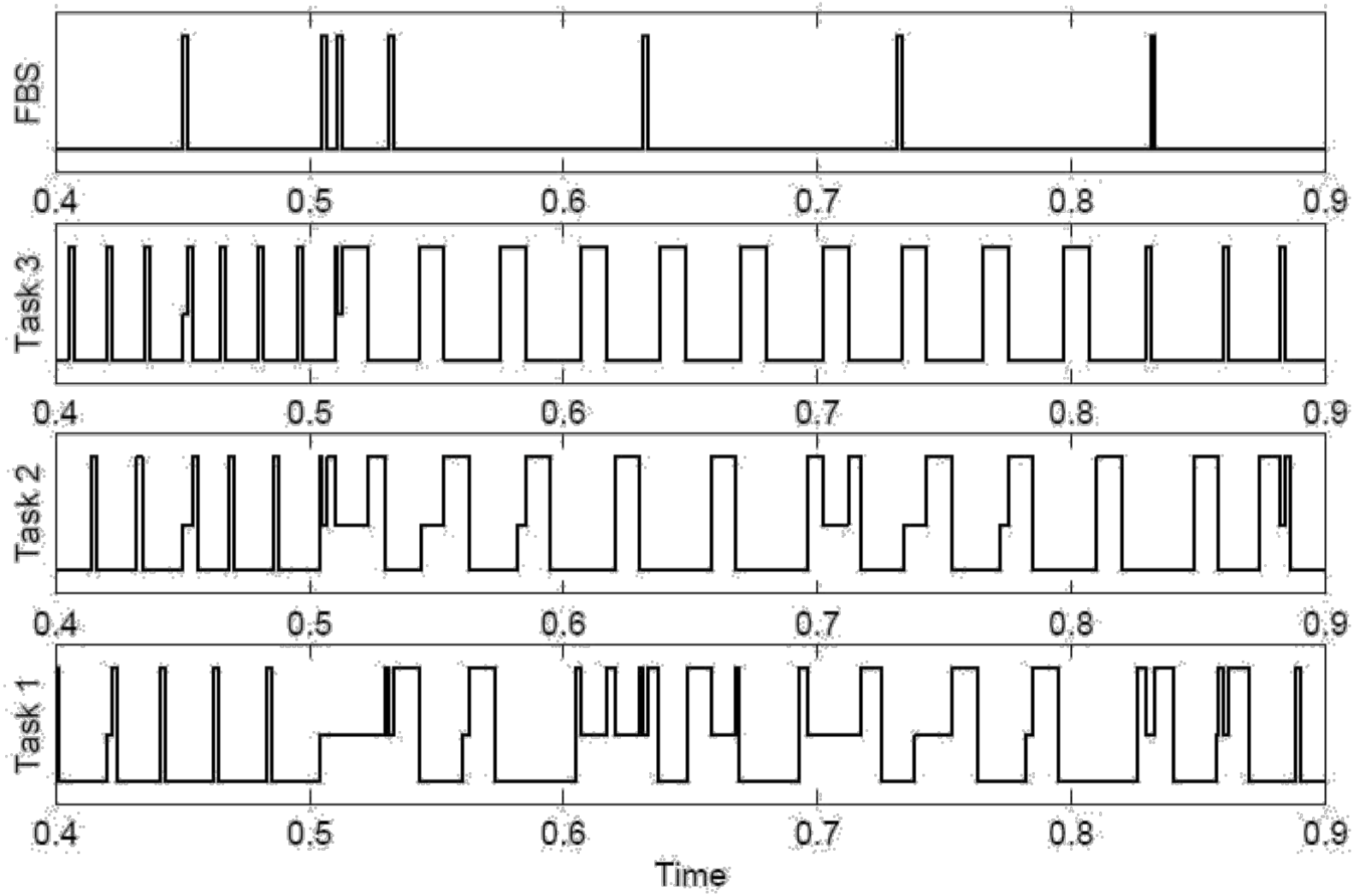
Feedforward

- Controller notifies feedback scheduler when switching from PID to Optimal mode
- Scheduler is released immediately
- Separate execution-time estimators in different modes

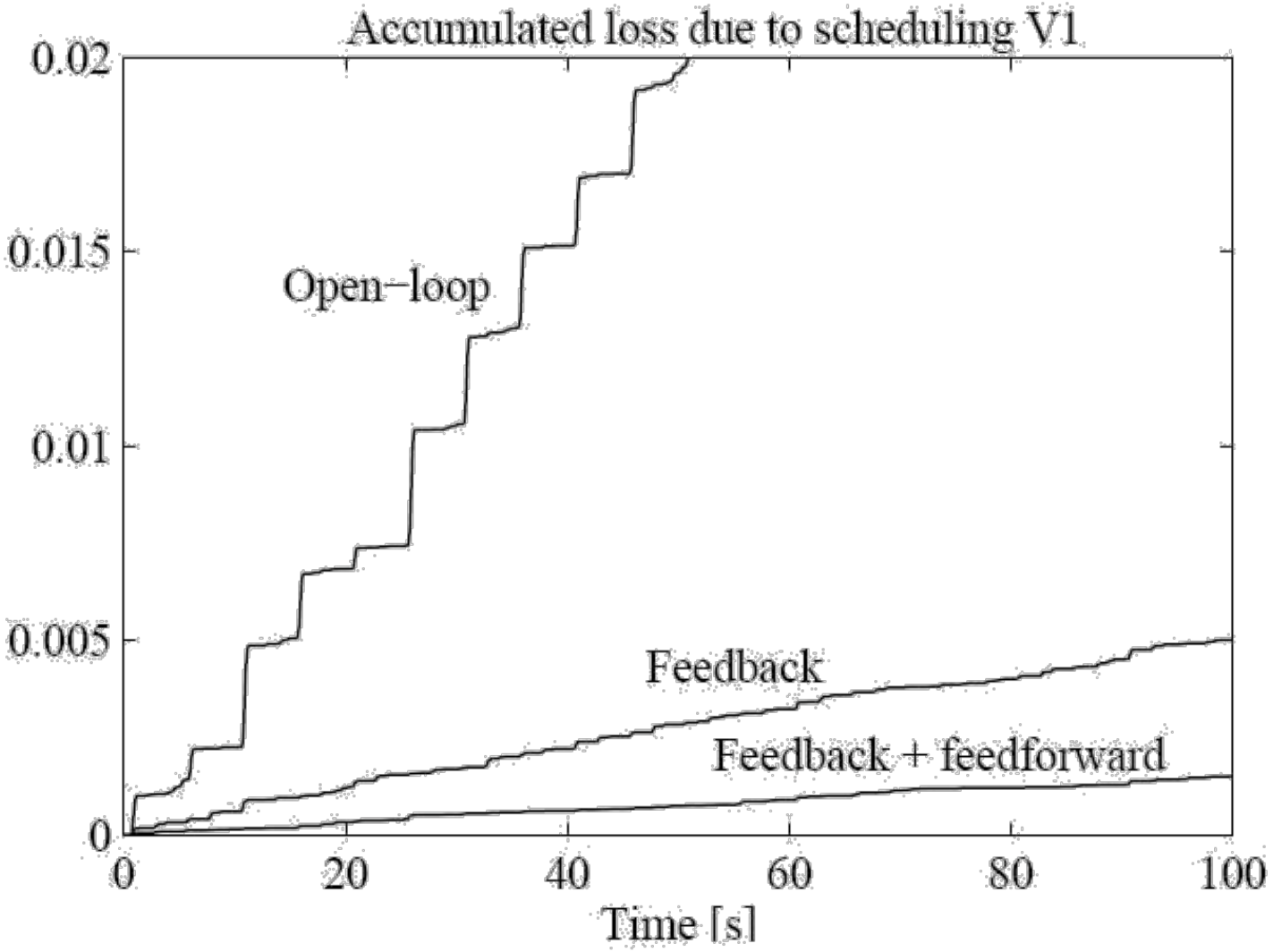
Feedback + Feedforward Scheduling



Feedback + Feedforward Schedule



Control Performance (QoS) Evaluation



Outline

- Embedded System Trends
- Definitions
- Adaptivity and Control
- Reconfigurable hardware
- Embedded Adaptivity Issues
- Three examples:
  - Feedback-Based Queue Length Control
  - Feedback Scheduling of Control Tasks
  - **Adaptive Resource Management in ACTORS**
- Conclusions

Objective

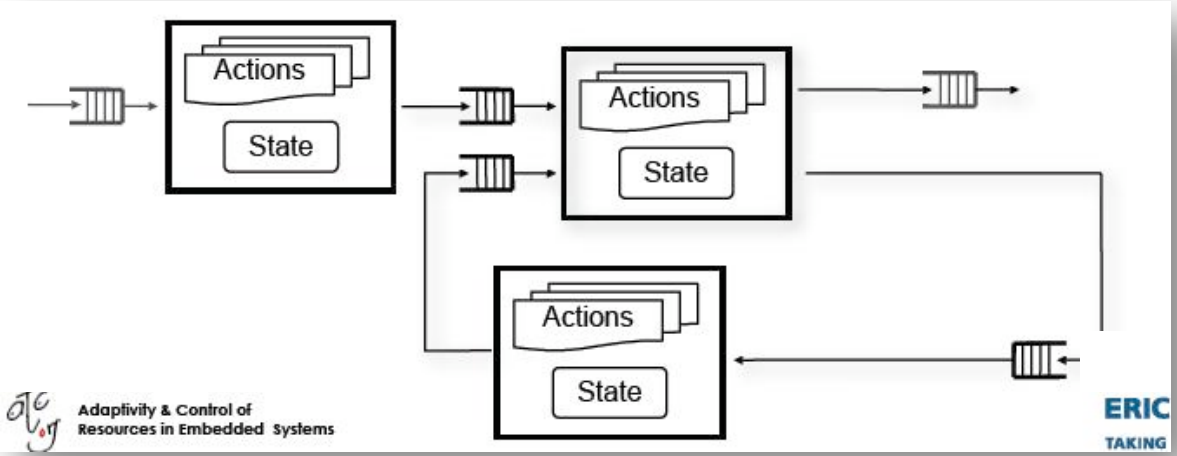
- Give an overview of the work on adaptive resource management in one of the current STREP projects

Feedback-Based Resource Management

- ACTORS – Adaptivity and Control of Resources in Embedded Systems
  - Ericsson (coord), SSSA, TUKL, Lund, EPFL, Akatech, Evidence
- Three main parts:
  - Dataflow Modeling for multimedia, control and signal processing
  - Reservation-based resource management (virtualization)
  - Feedback for providing adaptivity
- Demonstrators
  - Media streaming on cellular phones, control, high-performance video
- Platform: ARM 11 multicore with Linux 2.6.26

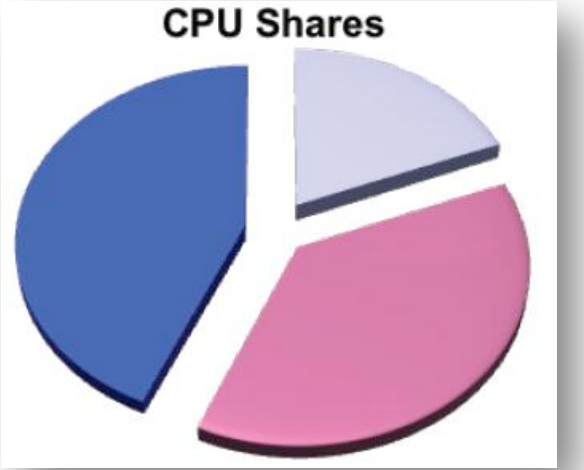
ACTORS: Dataflow Modeling

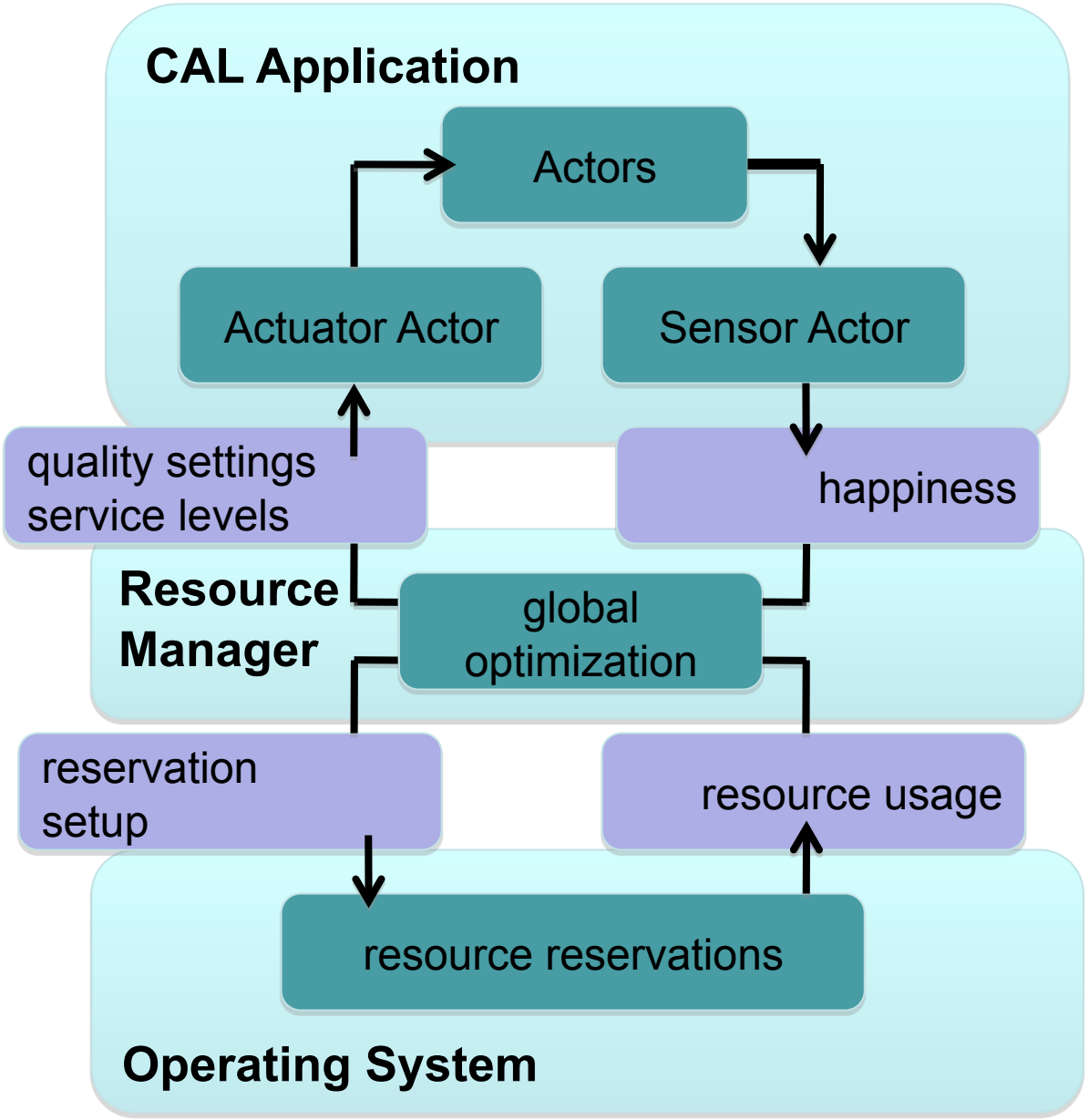
- Data flow programming with actors (Hewitt, Kahn, etc)
  - Associate resources with streams
  - Clean cut between execution specifics and algorithm design
  - Strict semantics with explicit parallelism provides foundation for analysis and model transformation
- CAL Actor Language (UC Berkeley, Xilinx) http://opendf.org
  - Part of MPEG/RVC



ACTORS: Resource Reservations

- Bandwidth servers for resource reservations
- Virtual processors
- Decouples the behavior of parallel activities (temporal isolation)




CAL Application
Actors
Actuator Actor
Sensor Actor
quality settings
service levels
happiness
Resource
Manager
global
optimization
reservation
setup
resource usage
resource reservations
Operating System

Outline

- Embedded System Trends
- Definitions
- Adaptivity and Control
- Reconfigurable hardware
- Embedded Adaptivity Issues
- Three examples:
  - Feedback-Based Queue Length Control
  - Feedback Scheduling of Control Tasks
  - Adaptive Resource Management in ACTORS
- **Conclusions**

Conclusions

- Adaptivity will without doubt be of increasing importance in future embedded systems
- The relations and tradeoffs between adaptivity, predictability, performance and dependability need investigations
- Parallel development taking place both in the hardware and the software community
  - Better connections and interfaces necessary
- Strong connections to control where adaptivity and reconfigurability have been studied since the 1960s.
  - Things to learn

Applications

- The dynamic nature of the approach makes it primarily applicable to applications with soft real-time constraints
  - Consumer electronics
  - Mobile telecommunications
  - Vehicular systems (informatics)
  - ….

What about Safety-Critical Systems?

- In many cases control systems
- Due to the feedback errors in the value domain are natural
- Control system designed using
  - Numerous approximations
    - Model reduction, linearization, …..
  - Verified through extensive simulations
  - Large safety margins when selecting, e.g., sampling periods
- Why is it then so unthinkable to use dynamic and adaptive approaches also at the implementation level?

Questions?

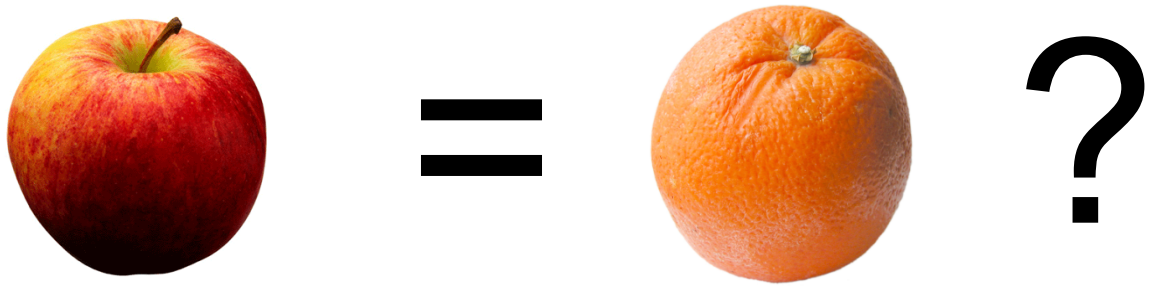
If there is time left….

Outline

- Embedded System Trends
- Definitions
- Adaptivity and Control
- Reconfigurable hardware
- Embedded Adaptivity Issues
- Three examples:
  - Feedback-Based Queue Length Control
  - Feedback Scheduling of Control Tasks
  - Adaptive Resource Management in ACTORS
- Conclusions
- **Quality-of-Service Adaptation (in case of time)**

Quality-of-Service

- Resource adaptation is ”easy” but QoS adaptation is ”difficult”
- QoS is difficult to define:
  - Application-dependent
  - User-dependent – Quality-of-Experience (QoE)
  - Context-dependent
- How to compare between different applications?



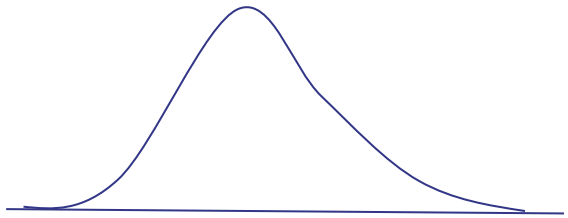
QoS = f(Resources)??

- The utility function - the relationship between the amount of resources given to an application and the QoS obtained - is not straightforward
- Often assumed to be monotonic, but not always the case
  - Internal dynamics: QoS = f(x, Resources)
    - X could be e.g., queue lengths
  - Linear or nonlinear
  - Time-varying
  - Multivariable resources

Specification of QoS

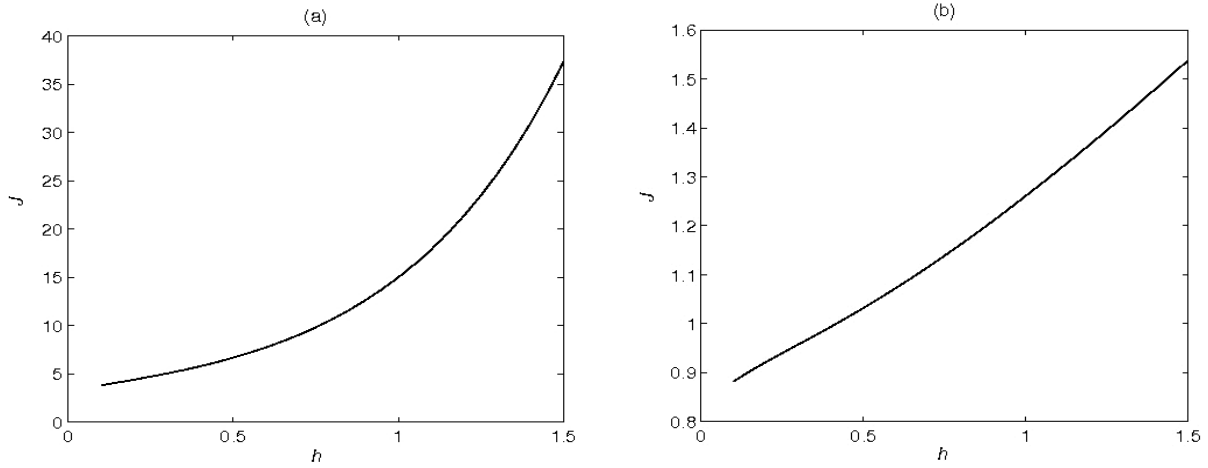
- How should we express the resource requirements or QoS requirements of a certain application?
  - Desired value + acceptable interval around this

  - ”Membership function”
  - Discrete levels
  - .....



Quality of Control

- QoS for control applications
- For linear control
  syste
  m
  s, it is possible to evaluate a quadratic cost function *J(h)*
- The shape of *J(h)* is often ”nice” (near-linear)

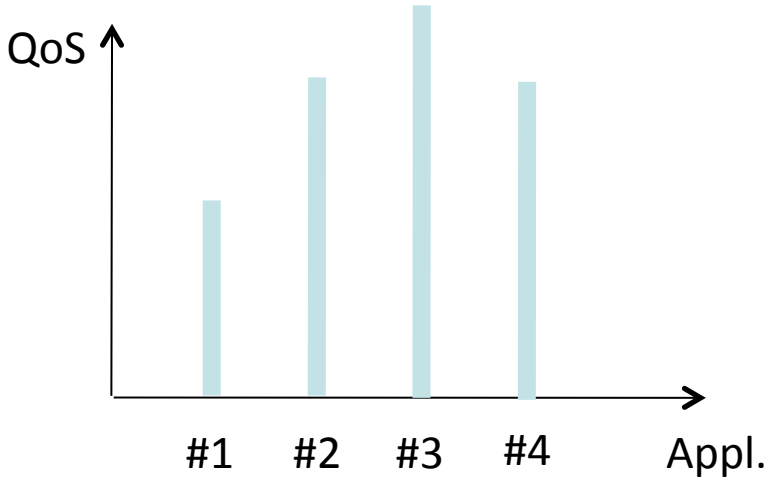


Specification of QoS

- How should one specify how the resources should be divided between different applications?
  - Statically
    - ”the MP3 player always gets (at least) 20%”
  - Dynamically/adaptively
    - Weights
    - Rules/policies
    - Wellness function
    - User preferences / profiles
    - .....

What is global QoS?

- How does QoS measures combine?
- Wellness functions:
  - Sum
  - Weighted sum
  - Max of minimum
  - ....



- Do they combine?
- Resource allocation frameworks such as, e.g., Q-RAM, give only a partial solution