Control of Computing Systems

Karl-Erik Årzén & Anton Cervin

Lund University

Outline

- **Overview**
- Feedback scheduling of control tasks
  - Heuristic
  - LQ-optimal
  - MPC
- Control of web servers

Control of computer systems

General idea: Apply control as a technique to manage uncertainty and achieve performance and robustness in computer and communication systems.

One of the strongest increasing areas in real-time computing (adaptive/flexible scheduling) and networking.

Applications in

- Internet protocols (TCP and its extensions)
- Internet servers (HTTP, Email)
- Cellular phone systems (power control, . . .)
- CPU scheduling (feedback scheduling)

Control of computer systems

New area

- however, feedback has been applied in ad hoc ways for long without always understanding that it is control

Textbook published a few years ago:

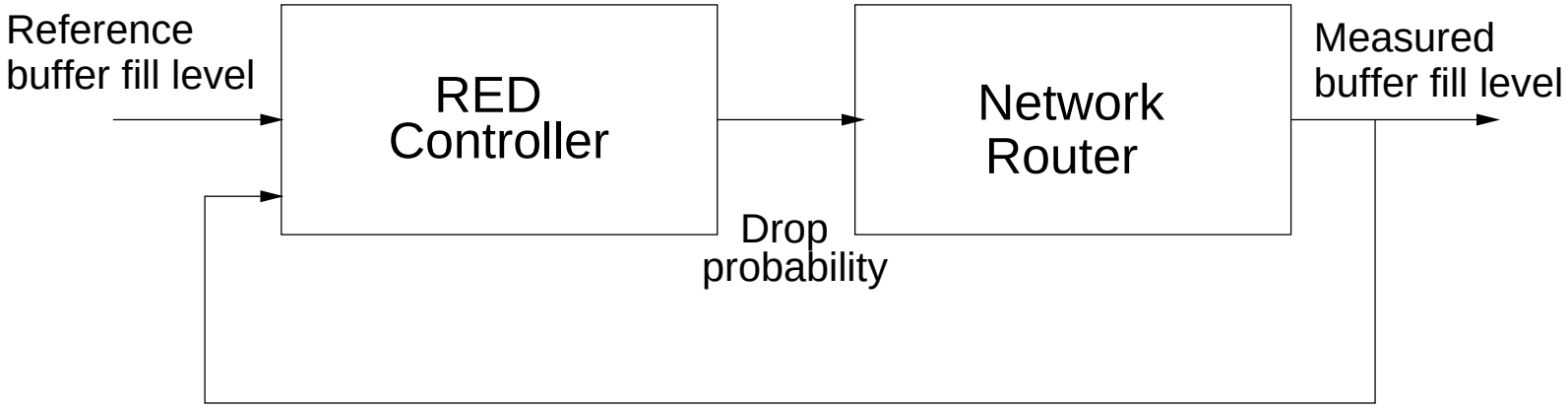
- “Feedback Control of Computer Systems”, Hellerstein, Diao, Parekh, Tilbury. IEEE Computer Society Press, 2004.

Example: Transport Control Protocol

The congestion control in TCP is one of the major reasons why Internet has been able to expand at the current high rate and still work properly.

- Congestion window cw decides how many un-ack'ed packets a host can have
- When cw below threshold it grows exponentially
- When cw above threshold it grows linearly
- Whenever there is a timeout the threshold is set to half the cw and cw is set to 1.
- Nonlinear behavior

Example: Internet protocols

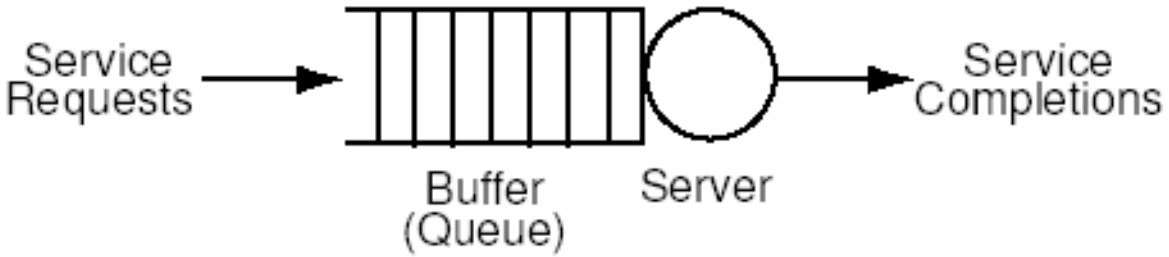


Random Early Detection (RED) of Router Overloads

- Prevent router buffers from overflowing
- Random drops of packets before the buffer is full

A lot of ongoing work on improvements of IP based on models and theory rather than on ad hoc fixes

Example: queuing systems



Work requests (customers) arrive and are buffered

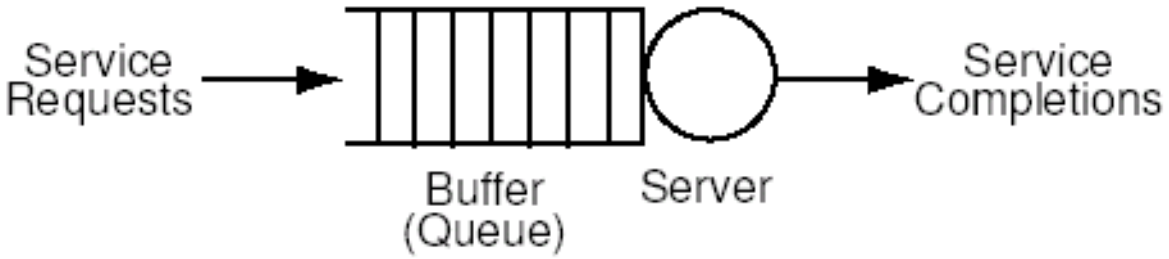
Service level objectives (e.g., response time for request belonging to class X should be less than Y time units)

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

Admission control

Example: queue length control

Assume an M/M/1 queuing system:

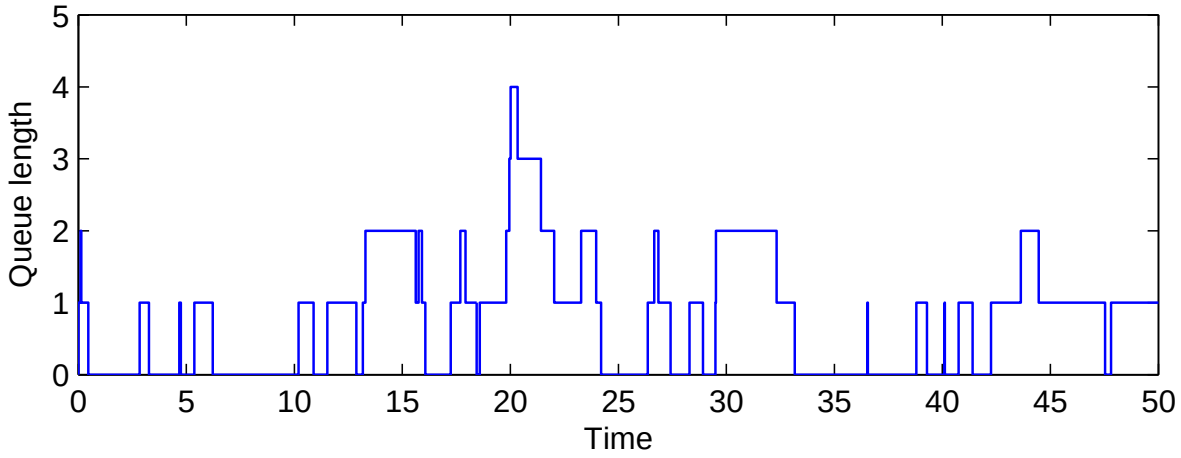


- Random arrivals (requests), Poisson distributed with average λ per second
- Random service times, exponentially distributed with average $1/\mu$
- Queue containing x requests

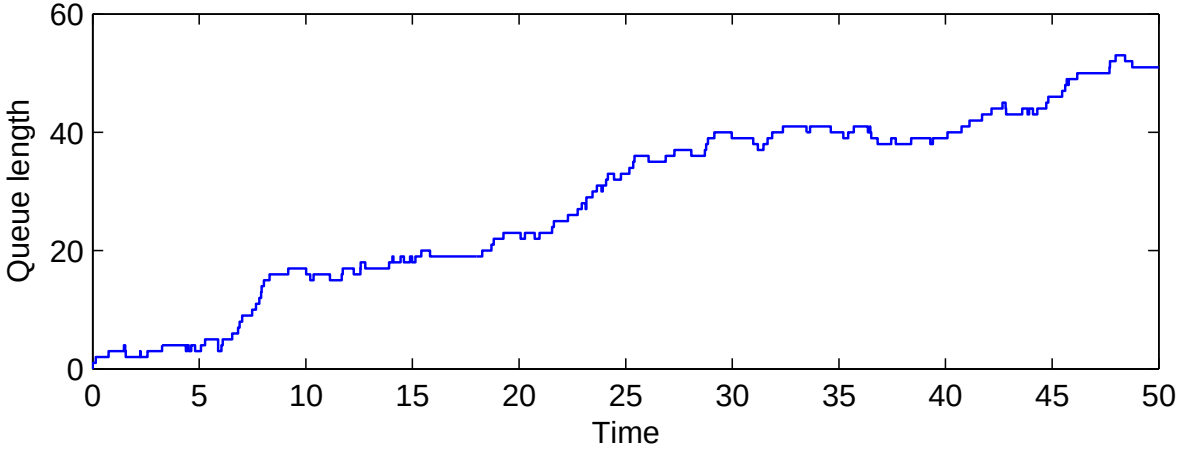
Intuition: $x \to \infty$ if $\lambda > \mu$

Queue length control: simulation

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



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



Queue length control: model

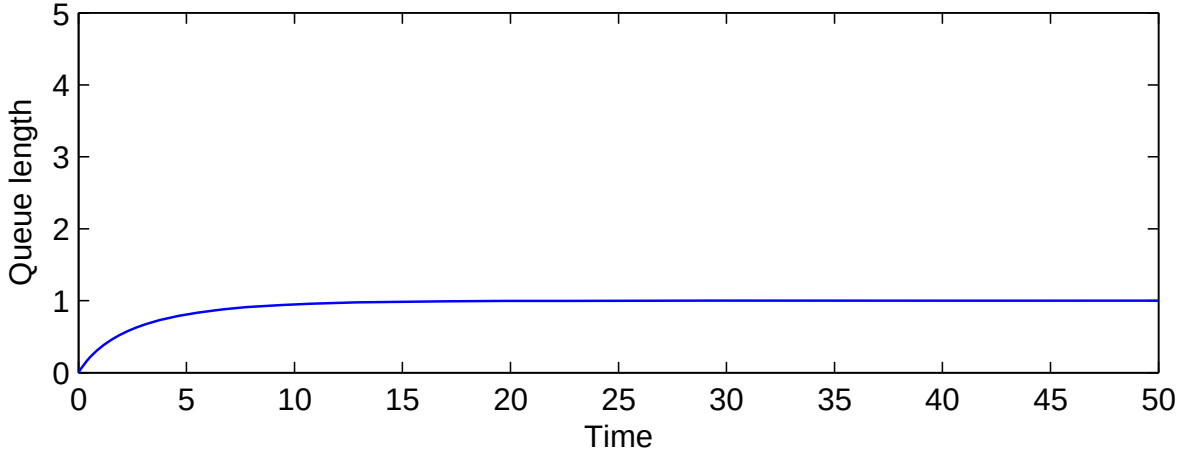
Approximate the system with a nonlinear flow model (Tipper's model from queuing theory)

The expectation of the future queue length x is given by

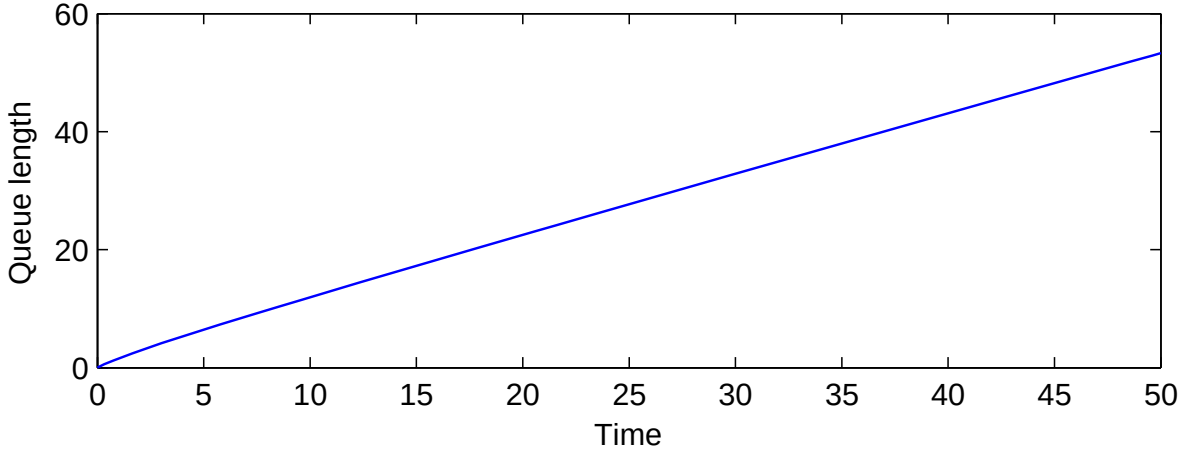
$$\dot{x} = \lambda - \mu \frac{x}{x+1}$$

Queue length control: model

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



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



Queue length control: model

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

Queue length control: linearization

Linearize around $x = x^\circ$

Let $y = x - x^\circ$

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

Queue length control: P-control

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

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

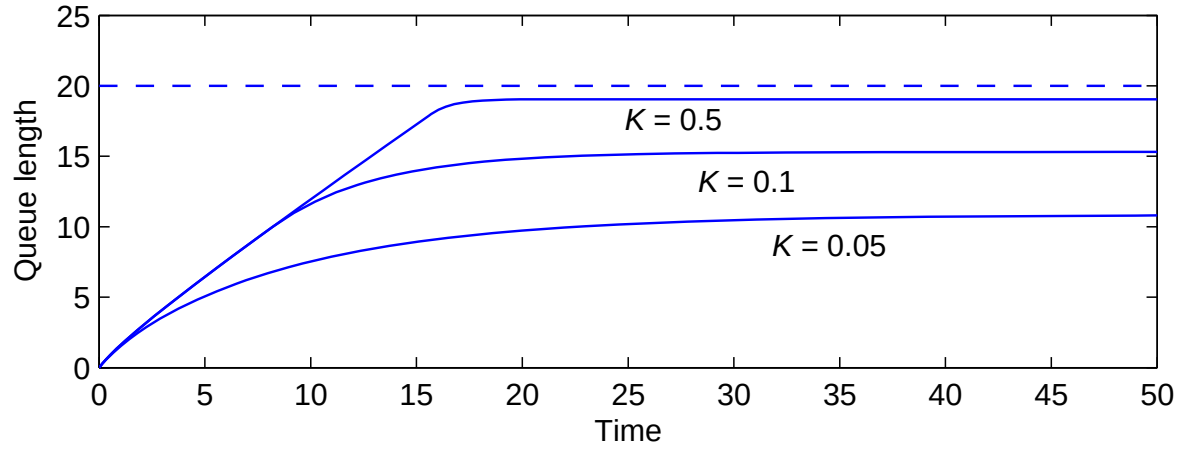
$$(s + \lambda K + \mu a)Y(s) = \lambda K R(s)$$

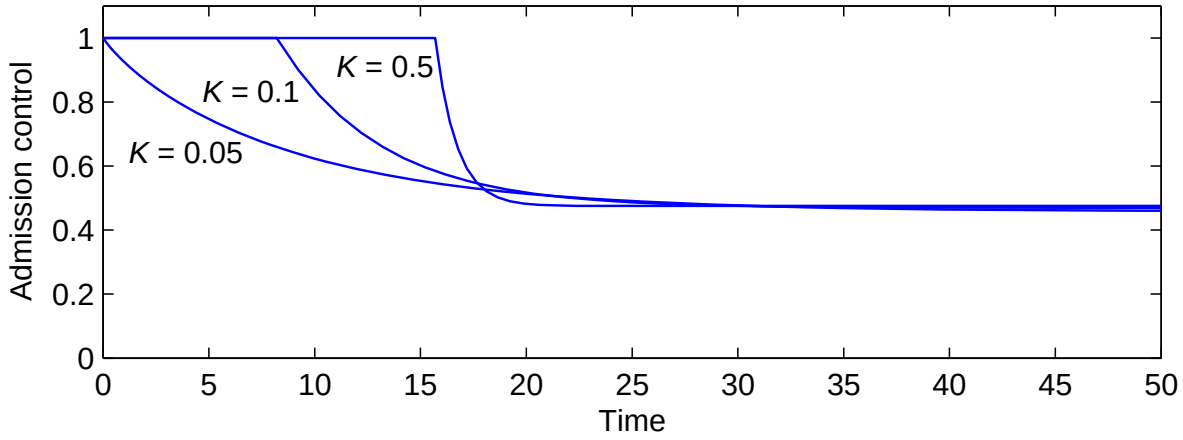
$$G_{cl}(s) = \frac{\lambda K}{s + \lambda K + \mu a}$$

With K the closed loop pole can be placed arbitrarily

Queue length control: P-control

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





Stationary error

ear response (control signal limitations)

Queue length control: PI-control

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

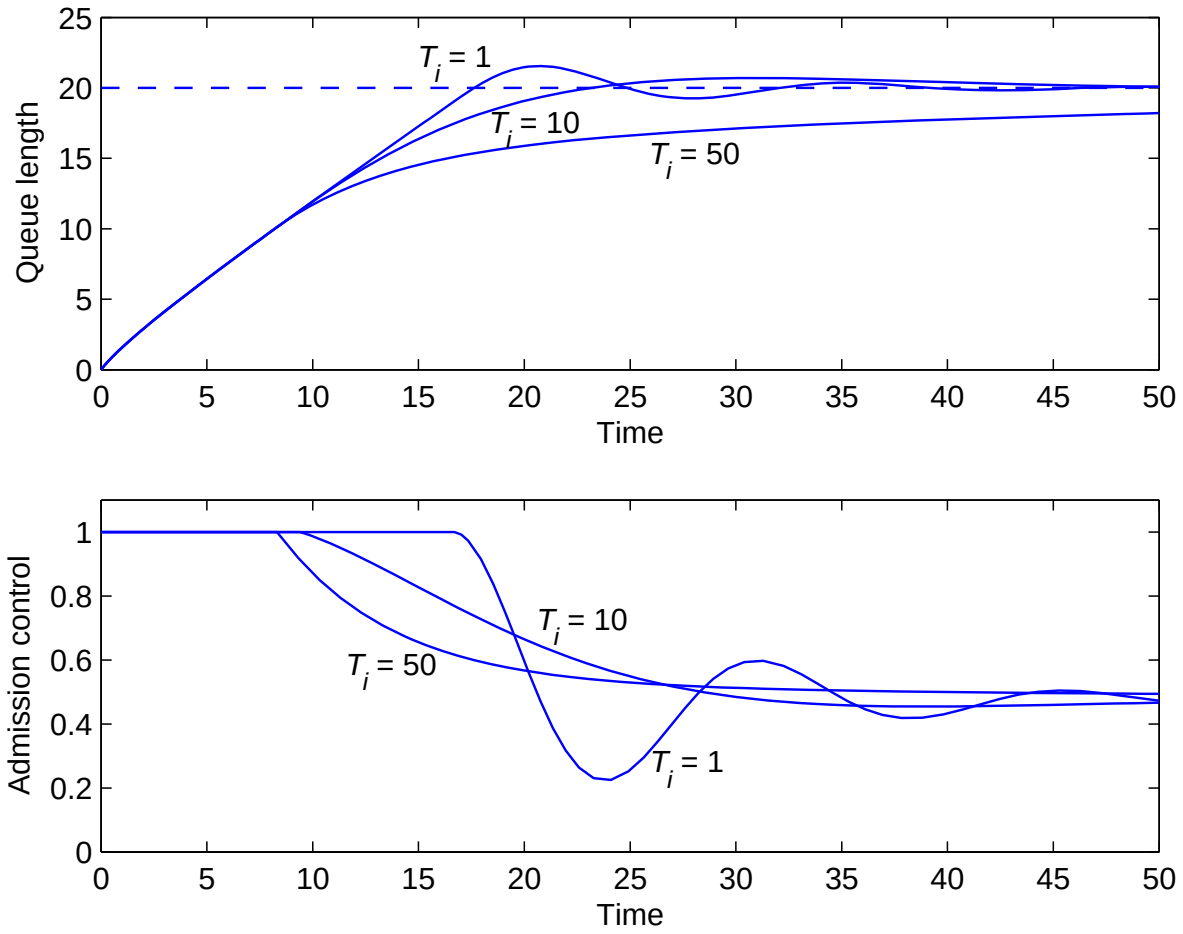
$$G_R(s) = K(1 + \frac{1}{sT_i})$$

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

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

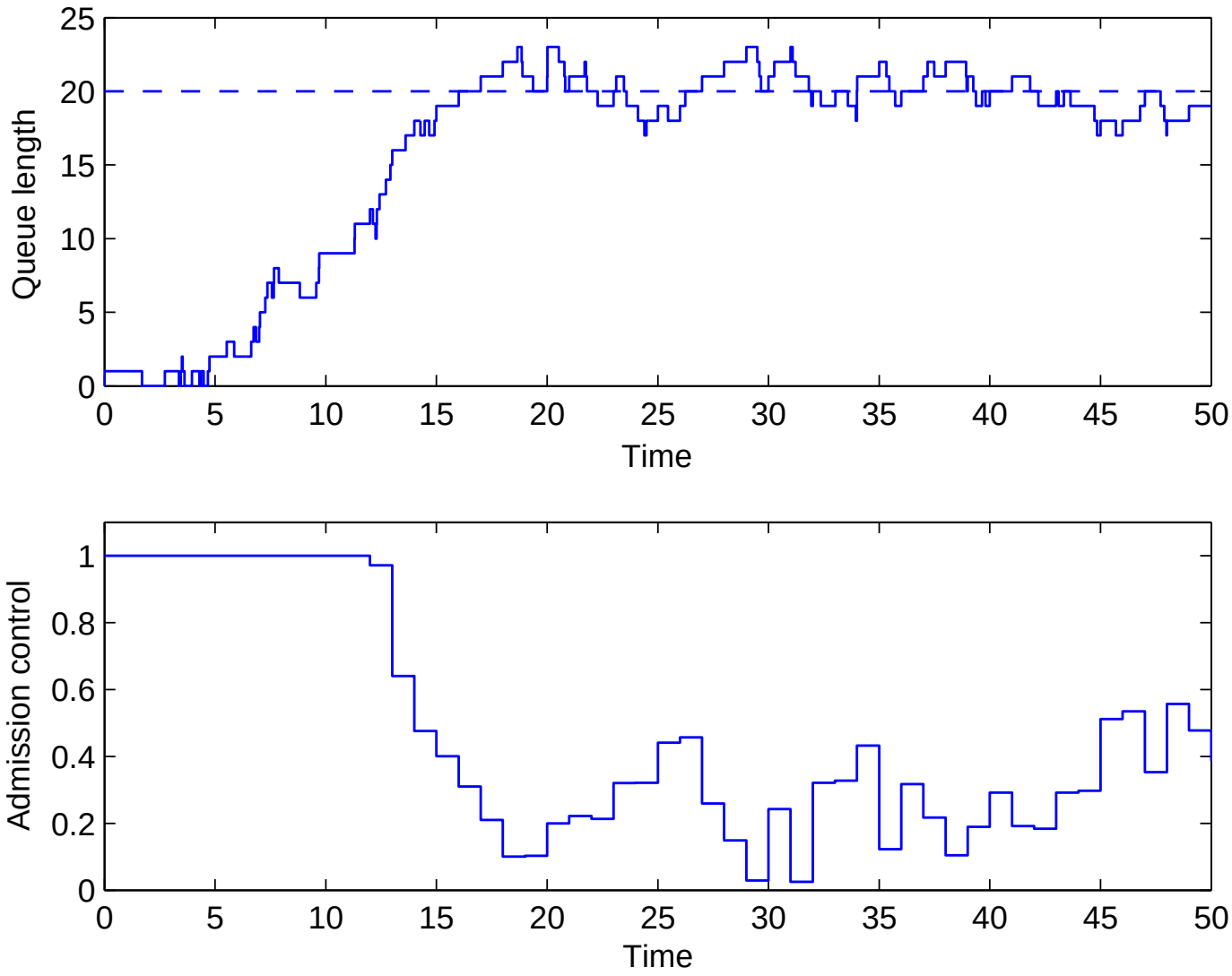
Queue length control: PI-control

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



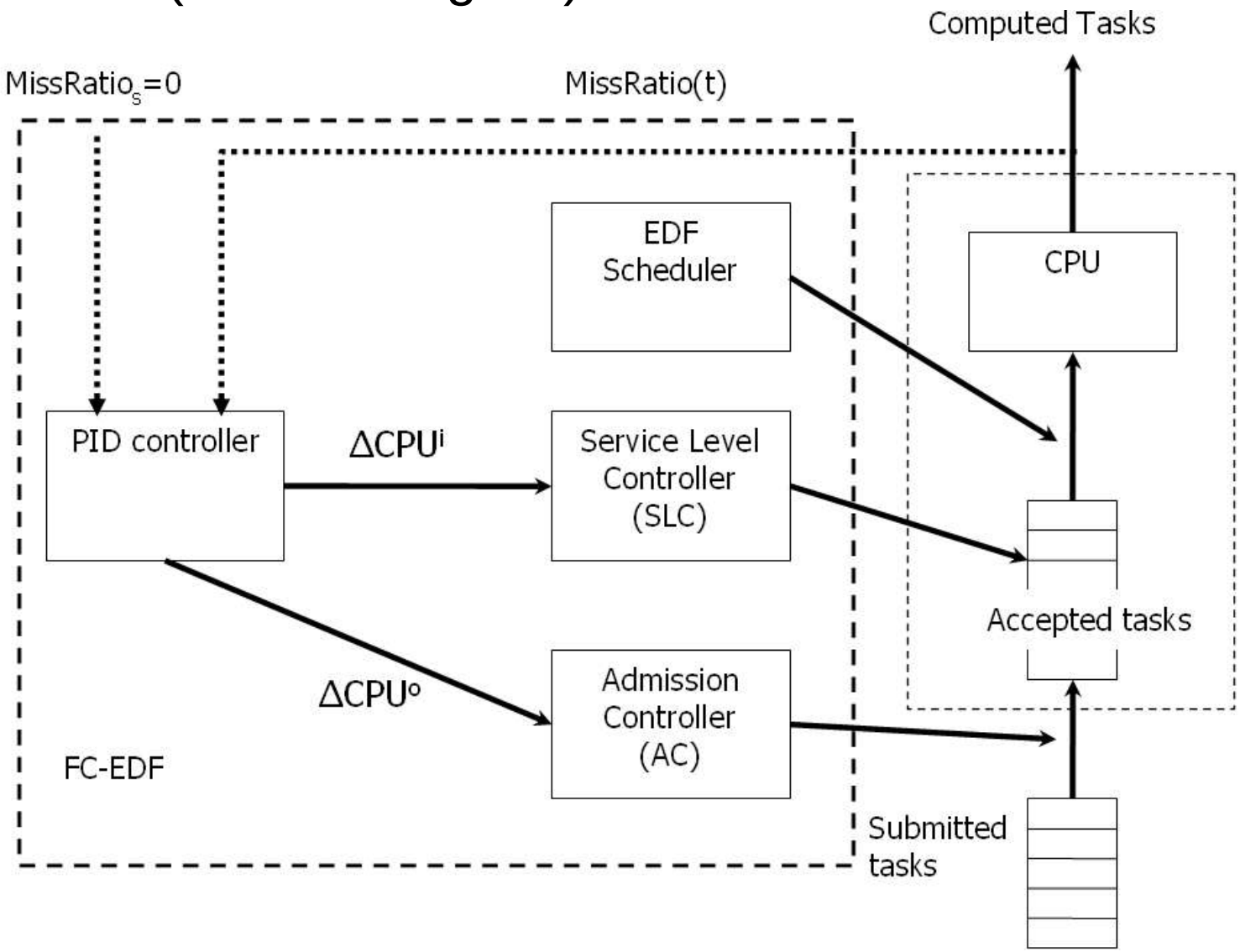
Stationary error removed

PI-control on event-based simulation model



Example: Feedback Control EDF Scheduling

Stankovic et al (Univ of Virginia)



Example: numerical integration

The automatic step-size adjustment in numerical integration routines for ODEs (ordinary differential equations) can be cast as a control problem

Ordinary PI/PID control works well

PhD thesis by Kjell Gustafsson, Dept of Automatic Control, Lund 1992

General observations

The plant under control rarely has any real dynamics or only very simple dynamics

- static nonlinearities + time delays (possibly time-varying)
- first or (maybe) second-order dynamics

Dynamics introduced through the sensors

- Time averages

Event-based control seems a more natural approach than time-based (though very have tried to apply it)

General observations

Seldom any measurement noise

- High-gain feedback (deadbeat control) a possibility

Simple controllers work well for the examples studied to far

- P, I, PI + feedforward, PD
- anti-windup to achieve good performance

Lack of first principles knowledge that can be used to derive models

- queuing systems an exception
  - however, the models here are long time averages
  - how use these for control?
- models often derived from input-output data

Outline

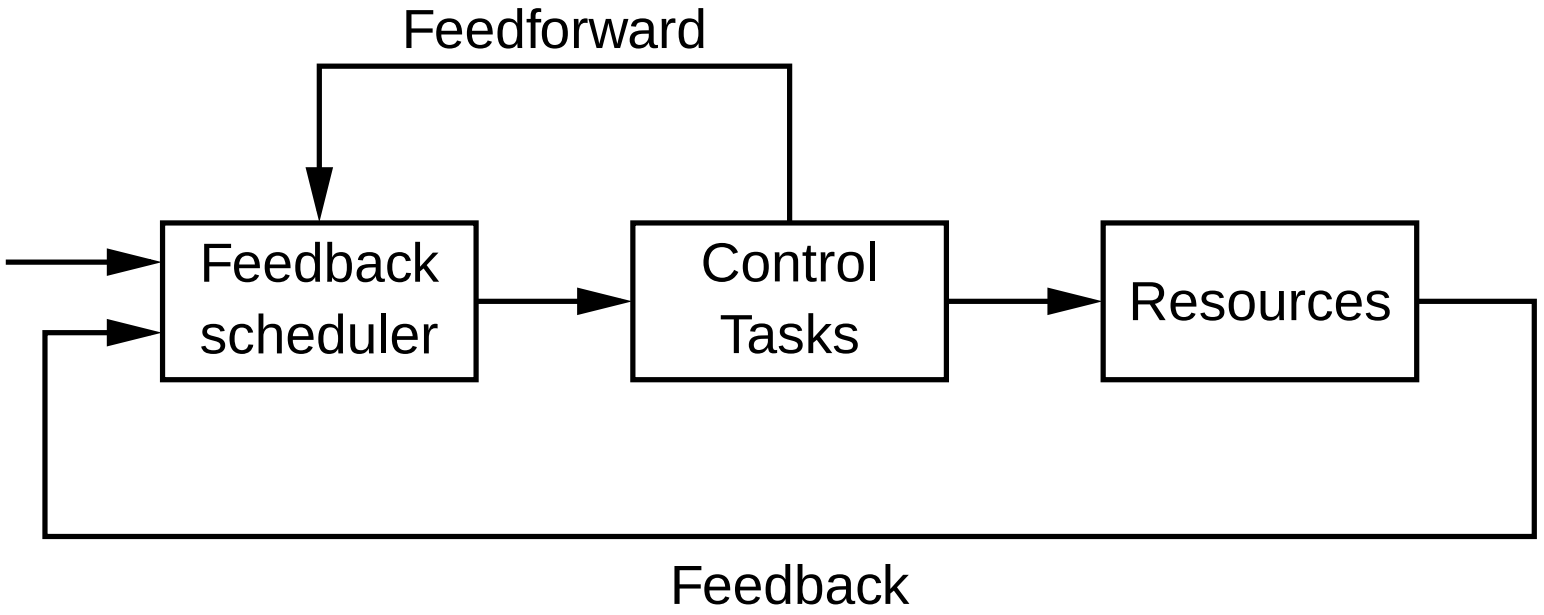
- Overview
- **Feedback scheduling of control tasks**
  - Heuristic
  - LQ-optimal
  - MPC
- Control of web servers

Feedback scheduling of control tasks

Dynamically handle control tasks with varying execution demands

Adjust sampling rates or execution times according to the current system state

A feedback scheduling structure



- Possible measurement signals: CPU utilization, execution times, control performance
- Possible control signals: sampling rates, execution budgets
- Feedforward: e.g. mode changes

Outline

- Overview
- Feedback control of Linux
- Feedback scheduling of control tasks
  - **Heuristic**
  - LQ-optimal
  - MPC
- Control of web servers

Heuristic feedback scheduling of control tasks

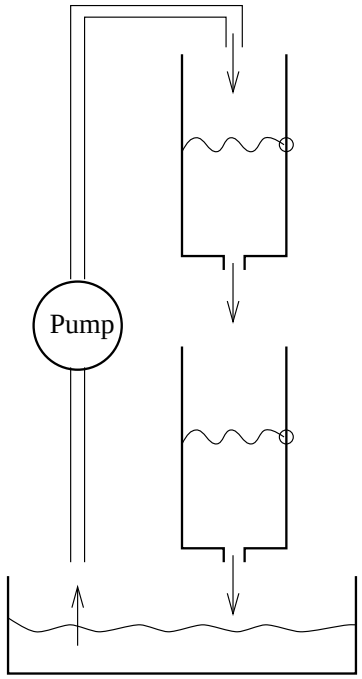
- Idea: On-line adjust sampling rates of a set of controllers to maximize CPU utilization and hence performance.
- Assume that nominal sampling periods h_{nom} are wisely chosen
- On-line estimate the total utilization U
- Periodically assign new sampling periods to meet the utilization setpoint U_{sp}:

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

- Possibly add feedforward to help with the estimation of U

Case study: set of hybrid controllers

The double-tank process: Use pump, $u(t)$, to control level of lower tank, $y(t)$



Hybrid control strategy:

- PID control in steady state
- Time-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

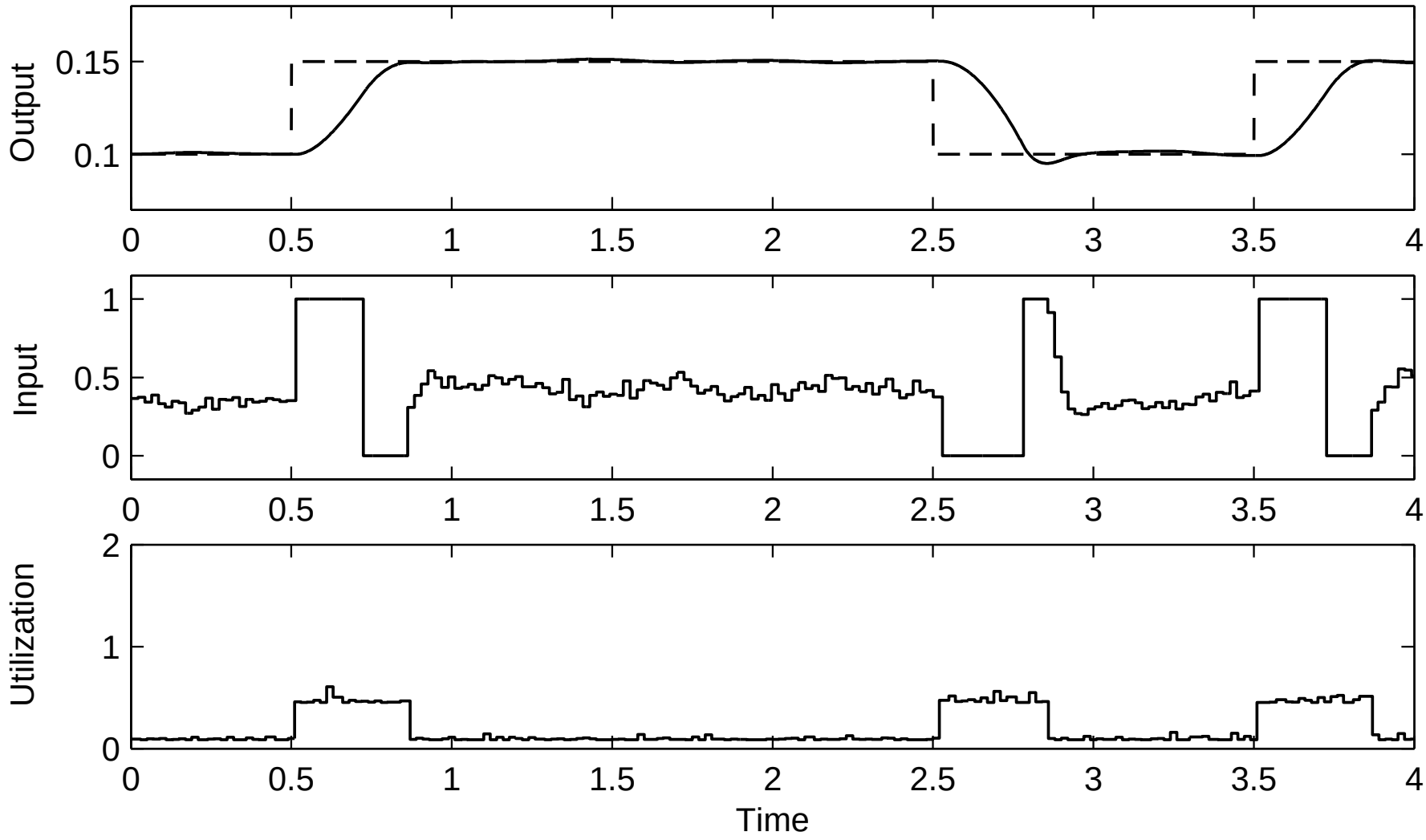
Computation of switching criterion:

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

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

Average execution time: $C = 10.0$ ms

Nominal behavior, $h = 21$ ms



Scheduling experiments

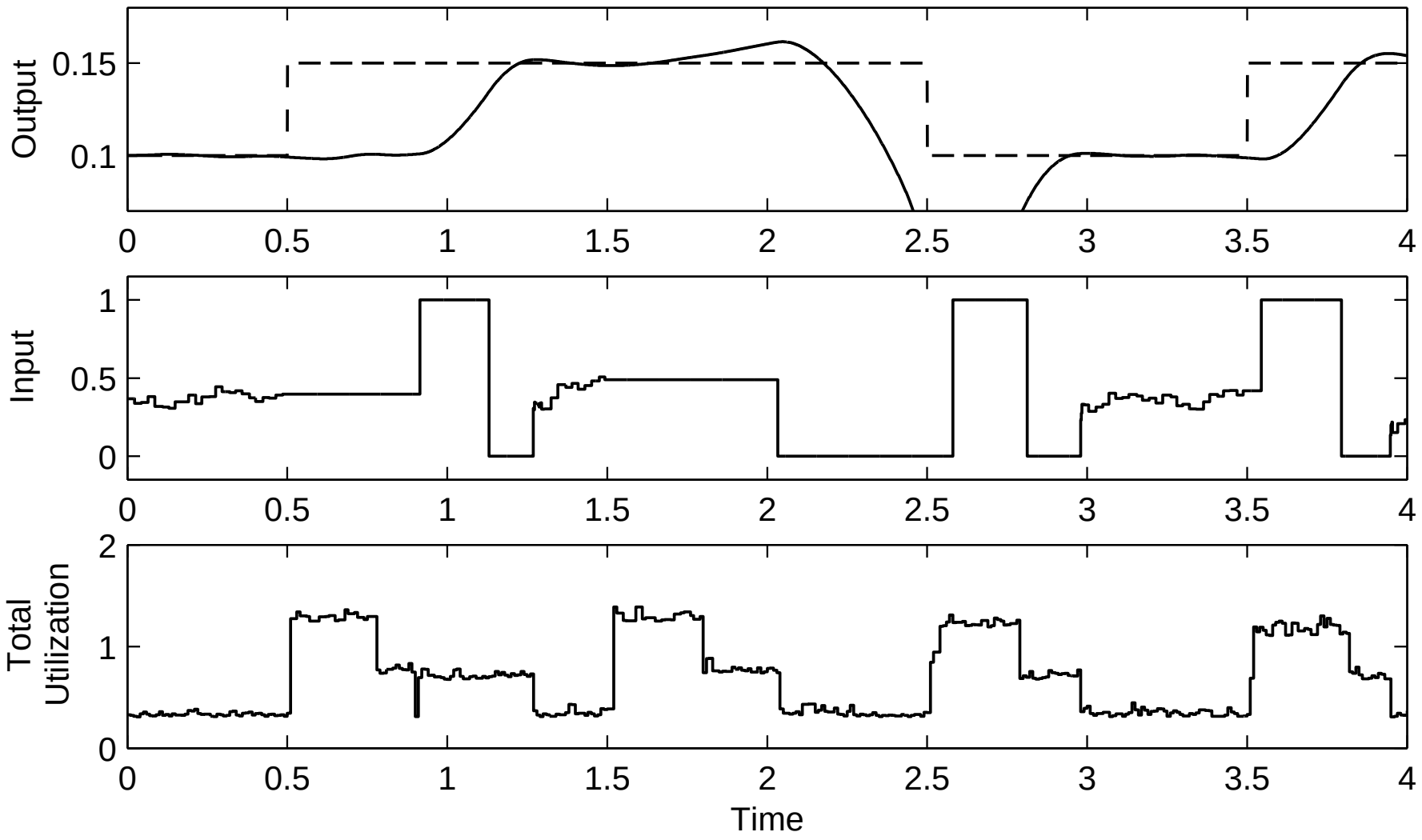
- Three hybrid controllers execute on one CPU
- Nominal sampling periods: $(h_1, h_2, h_3) = (21, 18, 15)$ ms
- Potential problem: All controllers in Optimal mode $\Rightarrow U = \sum \frac{C}{h} = 170\%$

Compare strategies:

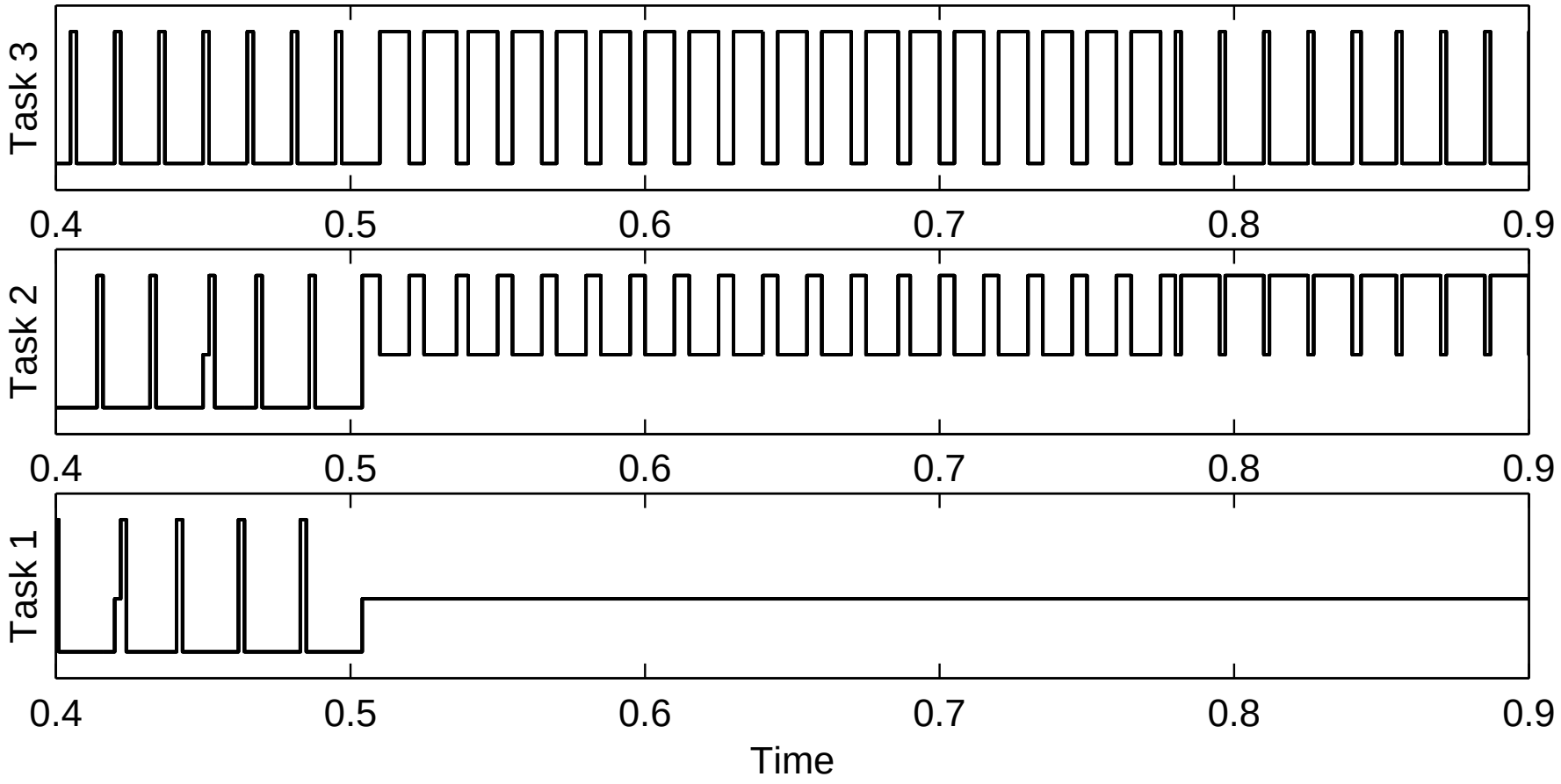
1. Open-loop scheduling
2. Feedback scheduling
3. Feedback + feedforward scheduling

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

Open-loop scheduling



Open-loop scheduling

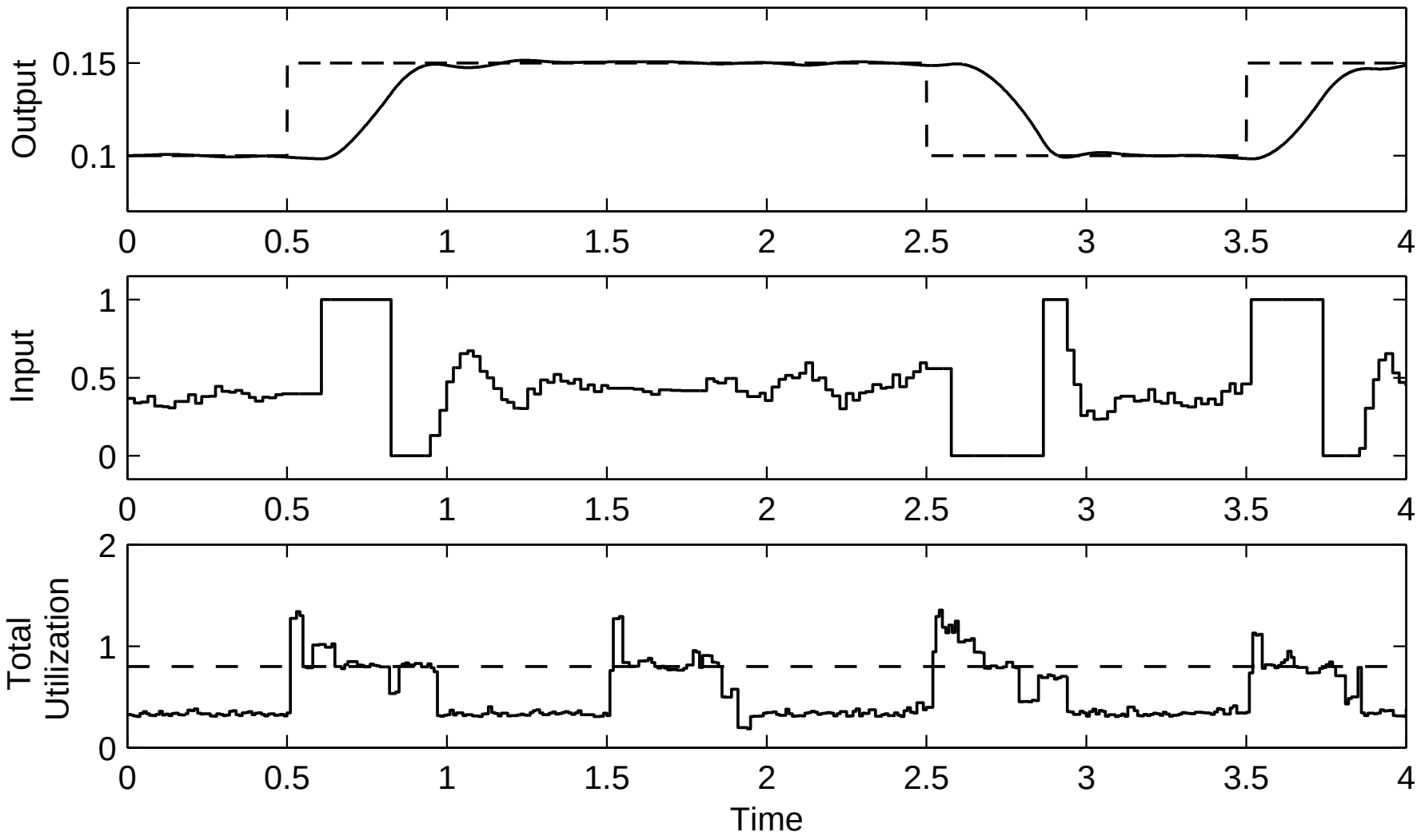


Feedback scheduler

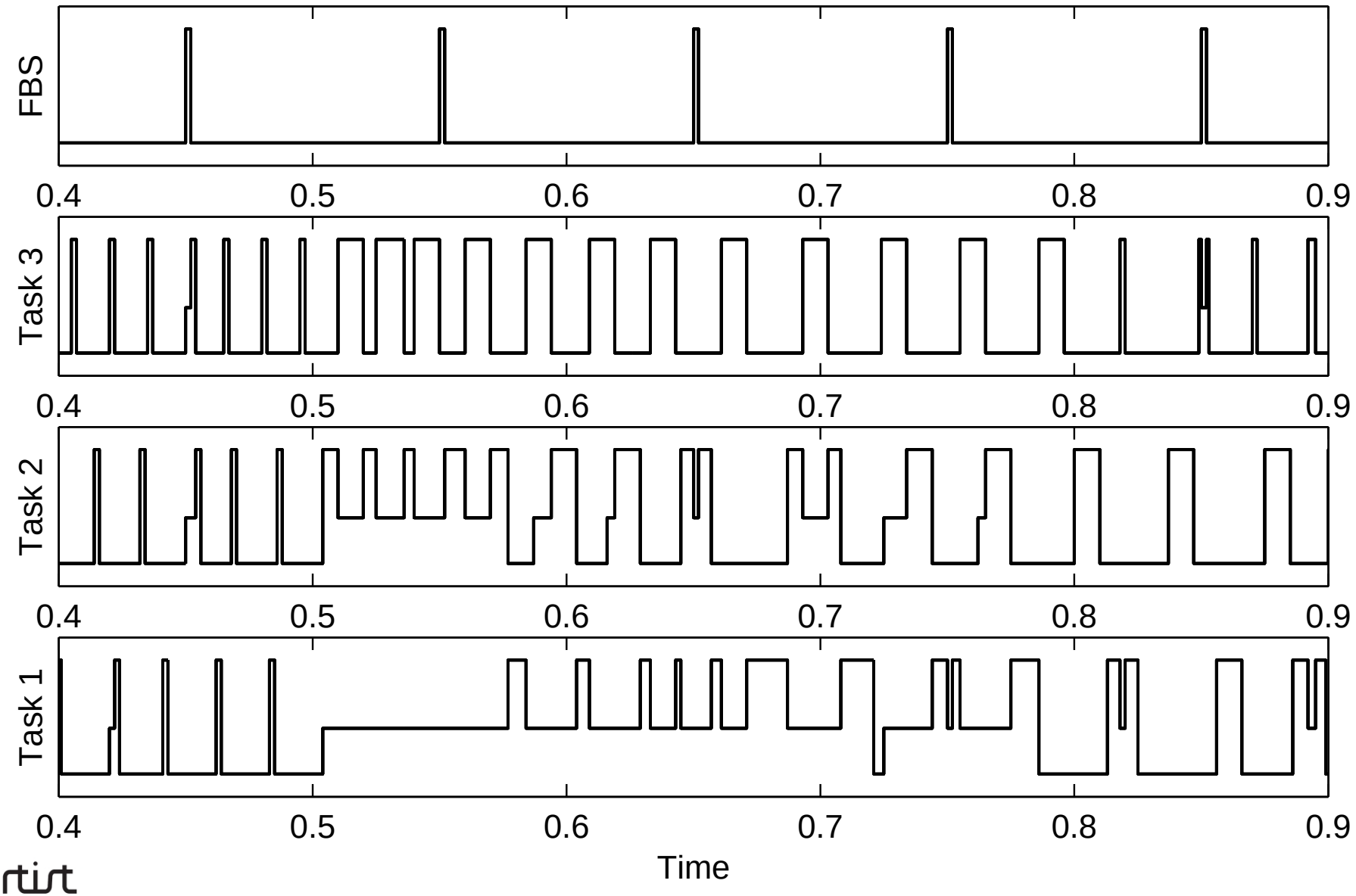
- A high-priority task, $T_{FBS} = 100$ ms, $C_{FBS} = 2$ ms
- Setpoint: $U_{sp} = 80\%$
- Estimate execution times using first-order filters
- Control U by adjusting the sampling periods:

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

Feedback scheduling



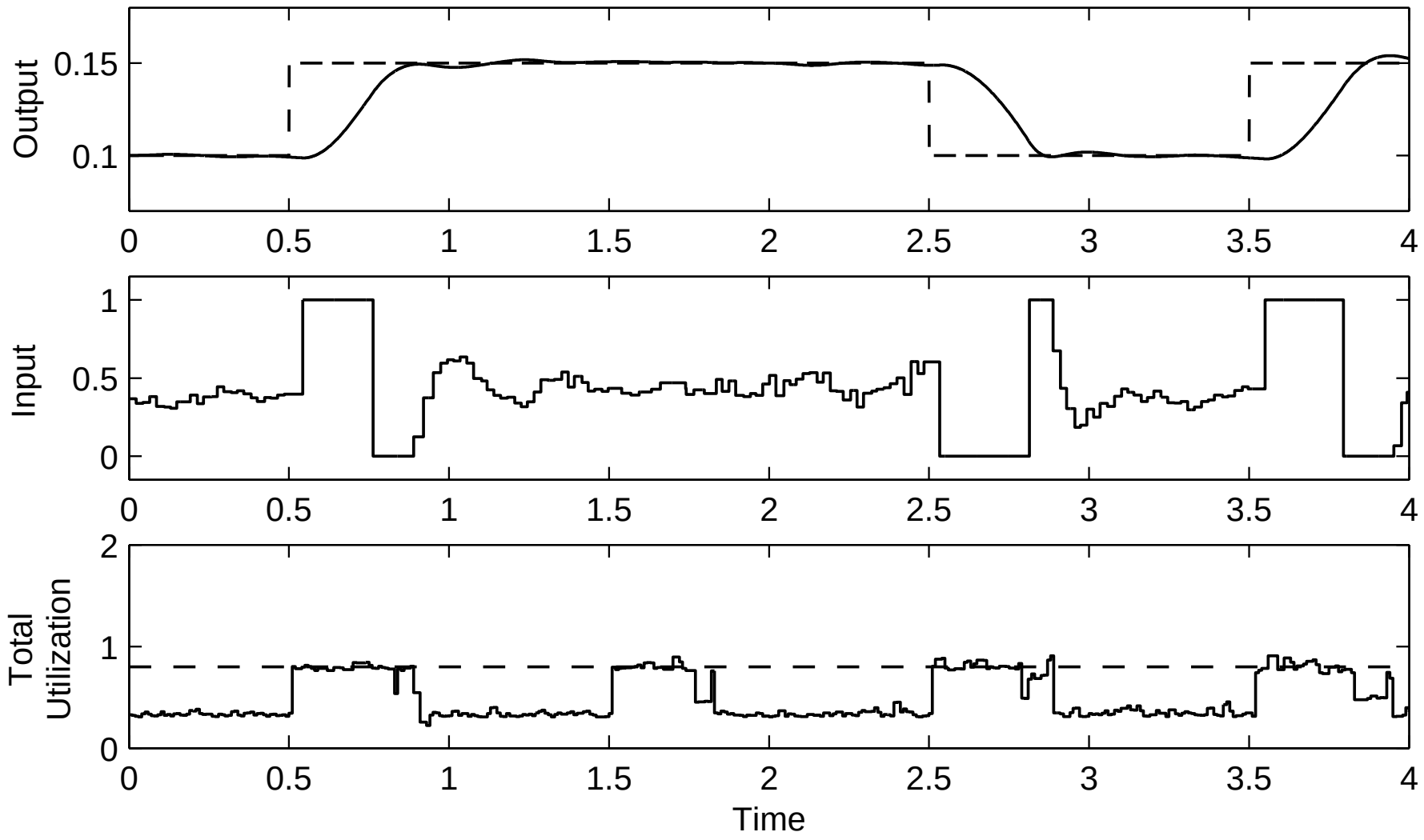
Feedback scheduling



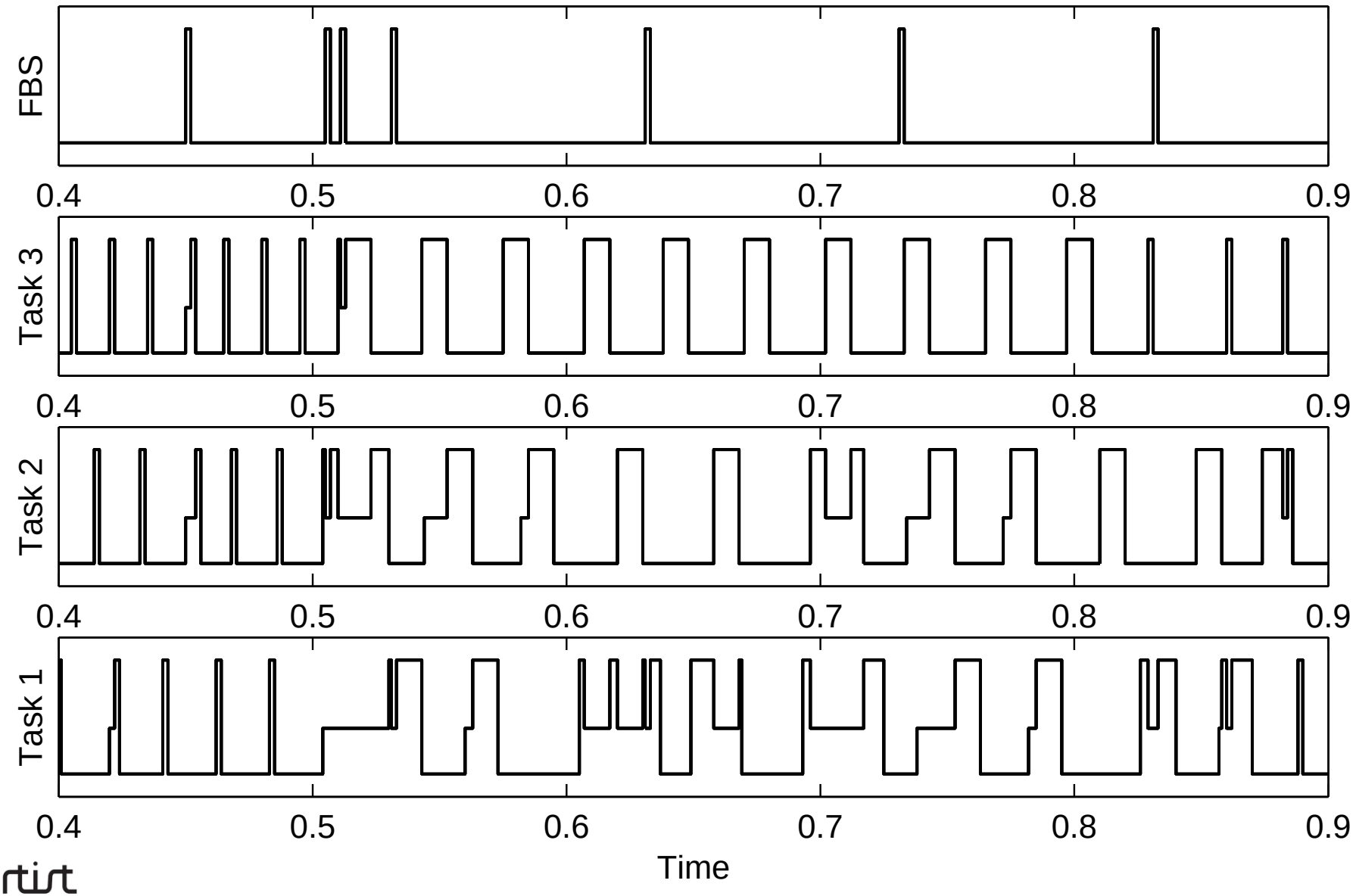
Feedforward

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

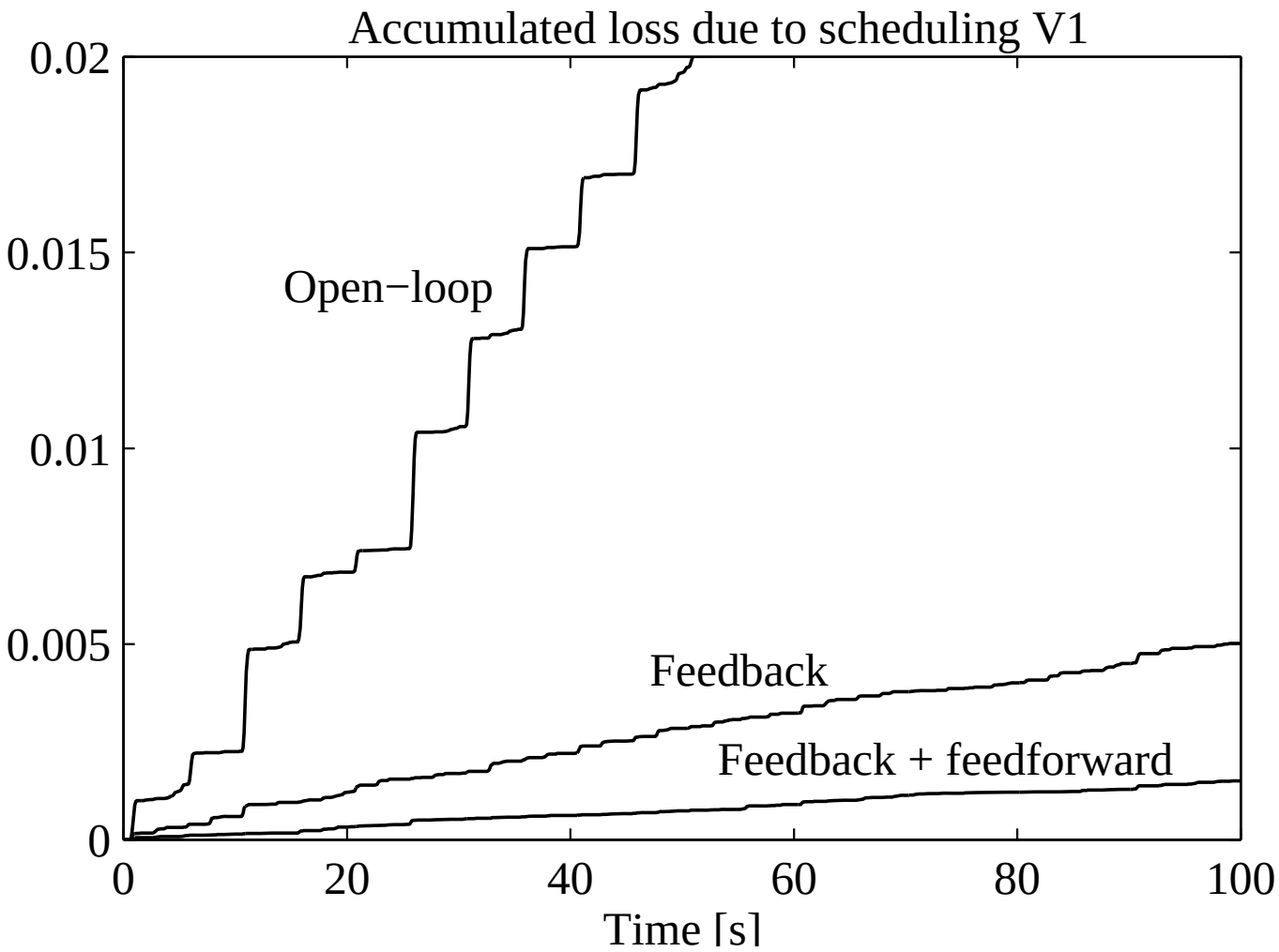
Feedback + feedforward scheduling



Feedback + feedforward scheduling



Control performance evaluation



Limitations

- Heuristic resource allocation
- Does not take the plant states into account
- The feedback scheduler period is not taken into account

Outline

- Overview
- Feedback control of Linux
- Feedback scheduling of control tasks
  - Heuristic
  - **LQ-optimal**
  - MPC
- Control of web servers

Feedback scheduling with LQ-optimal cost

Assume that the performance of each controller i can be described by a cost function $J_i(x_i, h_i, T_{FBS})$

- x_i – the current state of plant i
- h_i – the sampling period of controller i
- T_{FBS} – the optimization horizon of the feedback scheduler

The objective is to minimize the combined performance with respect to the utilization bound:

$$\min_{h_1 \ldots h_n} \sum_{i=1}^{n} J_i(x_i, h_i, T_{FBS})$$

$$\text{subj. to } \sum_{i=1}^{n} \frac{C_i}{h_i} \leq U_{sp}$$

Optimal period assignment

- The period assignment problem is convex if $J_i(x_i, 1/f_i, T_{FBS})$ are convex in f_i.
- Explicit solution if all cost functions have the same shape,

$$J_i = \alpha_i + \beta_i h_i^{\nu}$$

  - $\nu = 1$:

$$h_i = \left(\frac{C_i}{\beta_i}\right)^{1/2} \frac{\sum_{j=1}^{n} (C_j \beta_j)^{1/2}}{U_{sp}}$$

  - $\nu = 2$:

$$h_i = \left(\frac{C_i}{\beta_i}\right)^{1/3} \frac{\sum_{j=1}^{n} C_j^{2/3} \beta_j^{1/3}}{U_{sp}}$$

- Linear cost functions ($\nu = 1$) are often good approximations

Linear-Quadratic controllers

The cost function for an LQ controller is given by

$$J(x,h,T_{fbs}) = x^T S(h) x + \frac{T_{fbs}}{h}\left(\operatorname{tr} S(h) R_1(h) + J_v(h)\right)$$

- $S(h)$ – solution to the LQ Riccati equation
- $R_1(h)$ – sampled process noise variance
- $J_v(h)$ – inter-sample cost term

Example: integrator process

- Process: $dx = u\,dt + dv_c$
  - v_c – Wiener process with unit incremental variance
- Design cost function: $J = \int_0^{T_{fbs}} x^2(t)\,dt$
- Resulting cost:

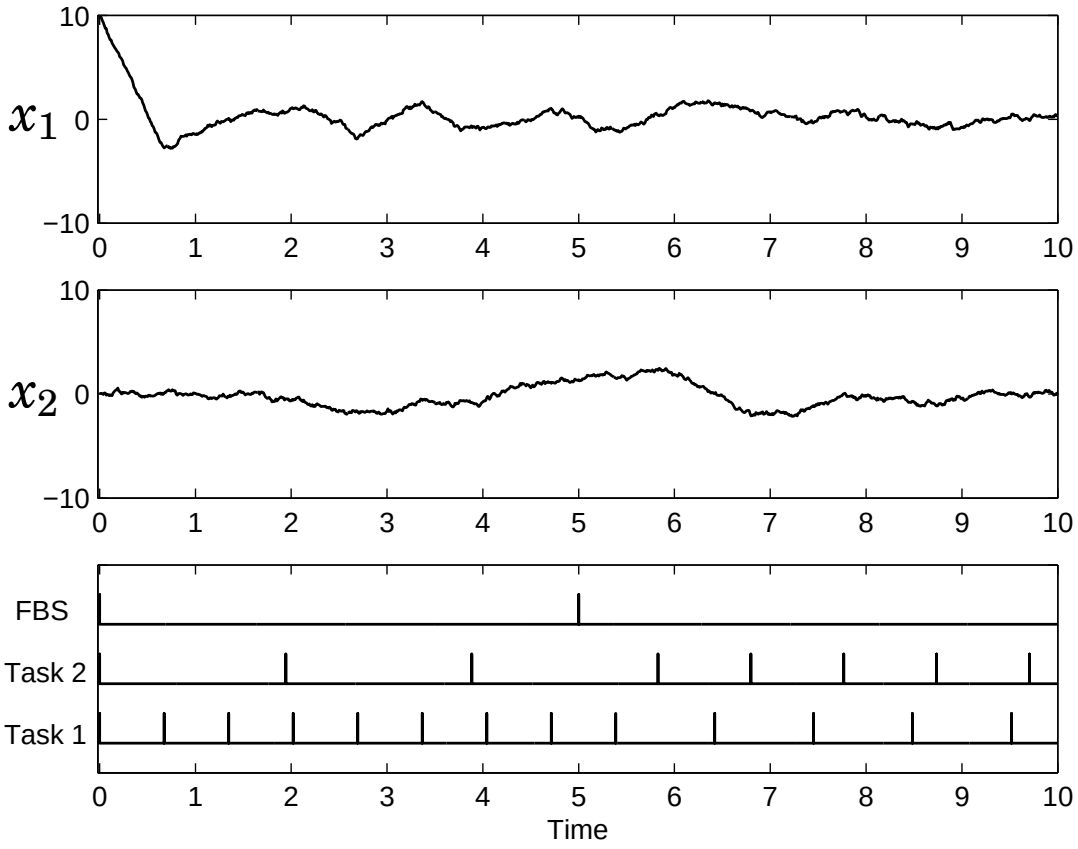
$$J(x, h, T_{fbs}) = \left(x^2 \frac{\sqrt{3}}{6} + T_{fbs} \frac{\sqrt{3}+3}{6} \right) h$$

  - Linear in h
  - Explicit solution for multiple controllers:

$$h_i \propto \sqrt{\frac{C_i}{x_i^2 + T_{fbs}(1+\sqrt{3})}}$$

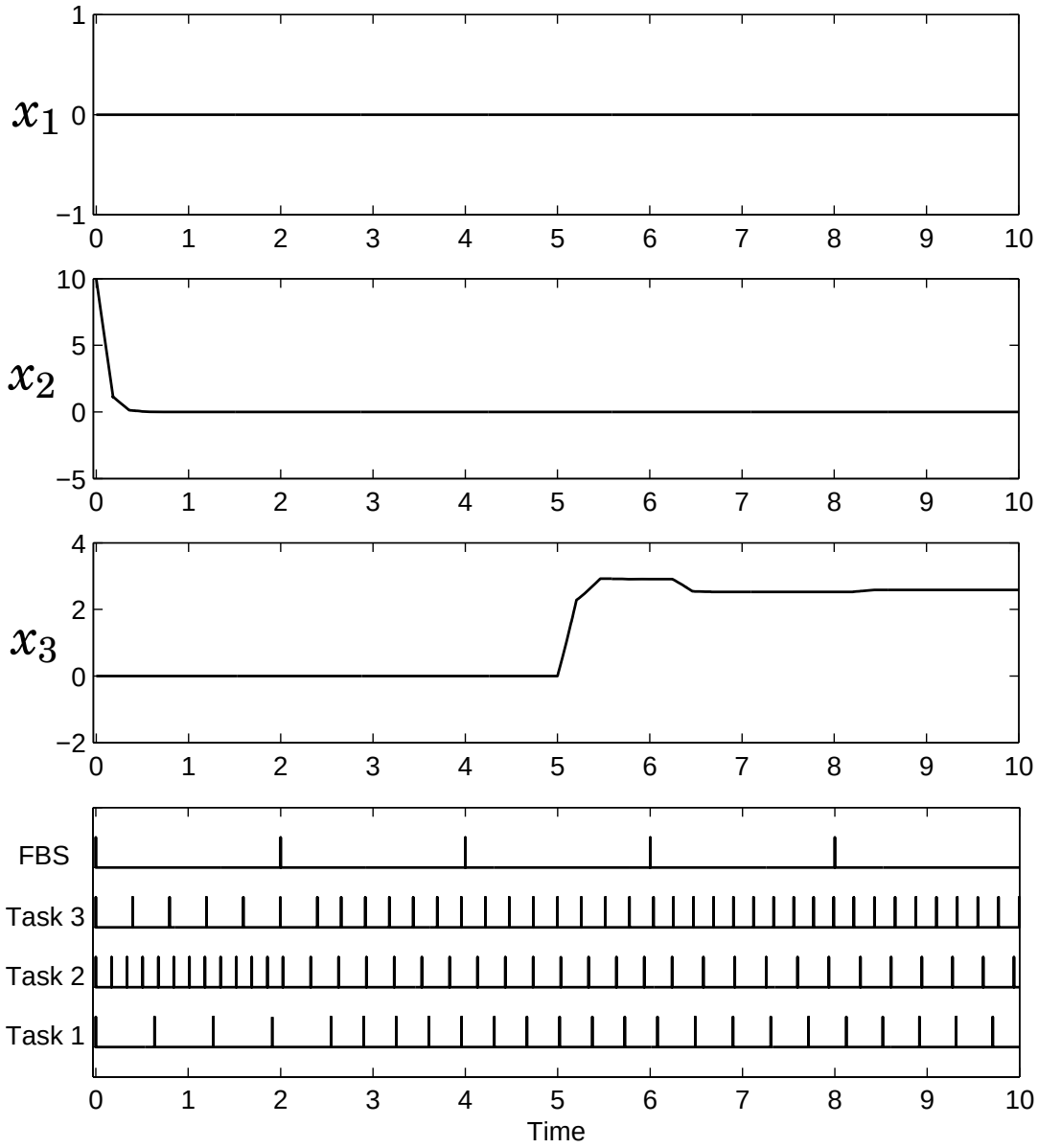
Simulation example 1

- Two integrator processes with different initial conditions
  - $x_1(0) = 10,\ x_2(0) = 0,\ C_1 = C_2 = 0.5,\ U_{sp} = 1,\ T_{fbs} = 5$



Simulation example 2

- Three first-order plants with $a = -1$, $a = 0$, and $a = 1$
- Load disturbance affecting plant 3 at time $t = 5$
- $x_1(0) = 0$, $x_2(0) = 10$, $x_3(0) = 0$, $C = 0.1$, $U_{sp} = 1$, $T_{fbs} = 2$


x_1
x_2
x_3
FBS
Task 3
Task 2
Task 1
Time

Outline

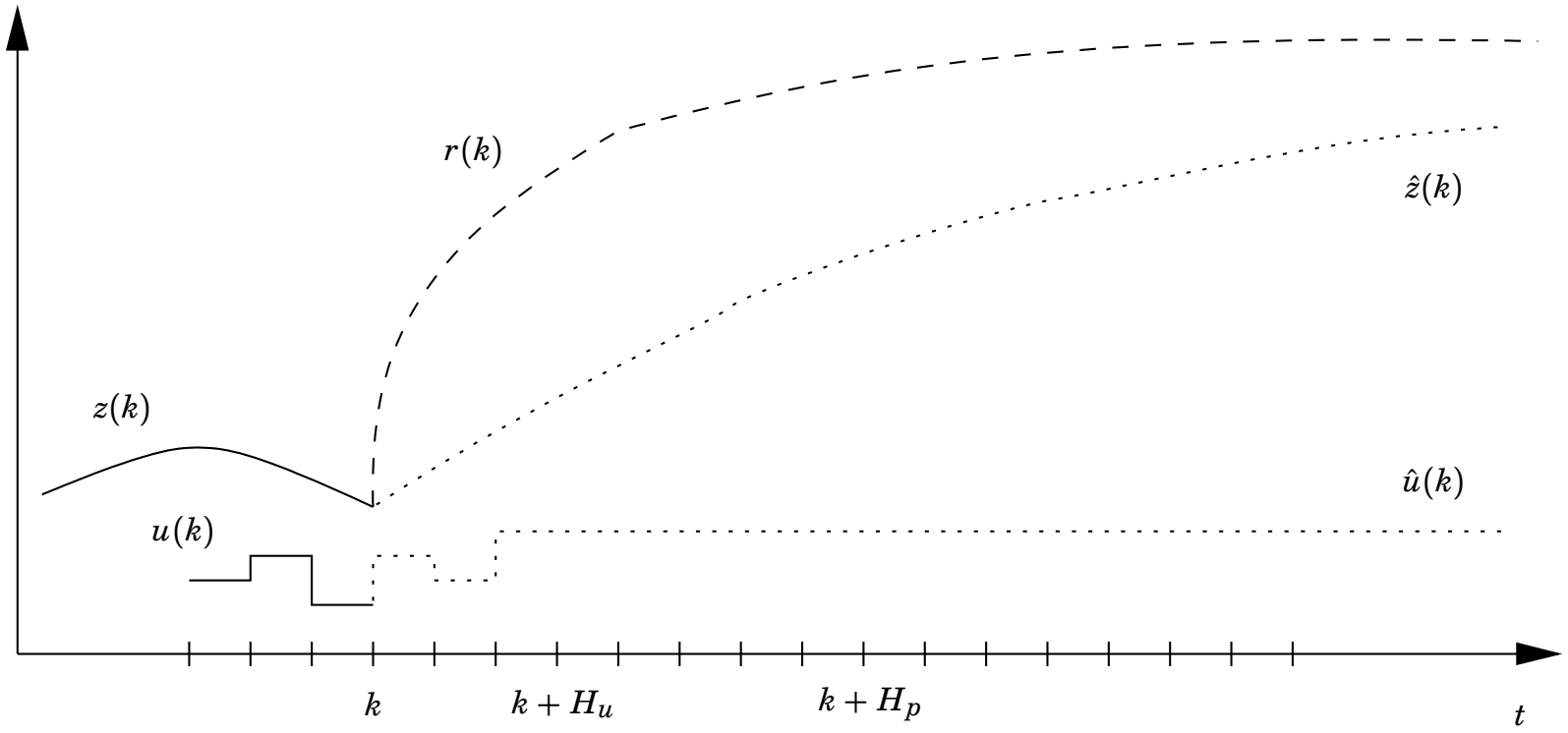
- Overview
- Feedback control of Linux
- Feedback scheduling of control tasks
  - Heuristic
  - LQ-optimal
  - **MPC**
- Control of web servers

Anytime controllers

Model-Predictive Control (MPC) is an example of an anytime controller

- On-line convex optimization problem solved each sample
- Highly varying execution times
- For fast processes the latency may effect the control performance considerably
- The control algorithm is based on a quality-of-service type cost measure, cp instantaneous cost
- As long as a feasible control signal has been found the iterative search can be aborted before it has reached completion
- Maps well to the imprecise task model
  - Mandatory part
  - Optional part

Model predictive control

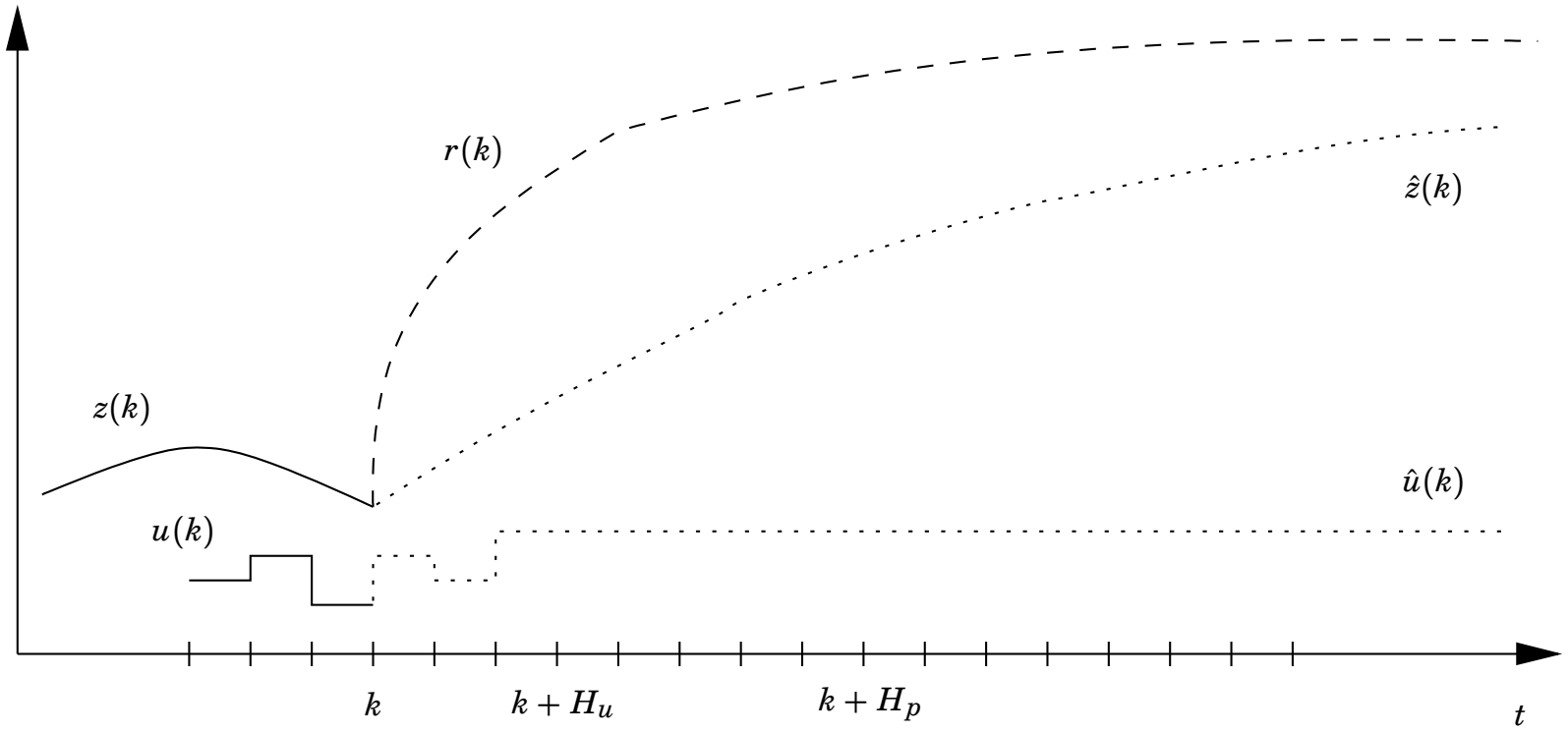


In *each sample*, find $\Delta\hat{u}(k) \ldots \Delta\hat{u}(k + H_u - 1)$ minimizing the cost

$$V(k) = \sum_{i=1}^{H_p} \|\hat{z}(k+i) - r(k+i)\|_Q^2 + \sum_{i=0}^{H_u-1} \|\Delta\hat{u}(k+i)\|_R^2$$

onstraints on control signals and controlled variables

Model predictive control



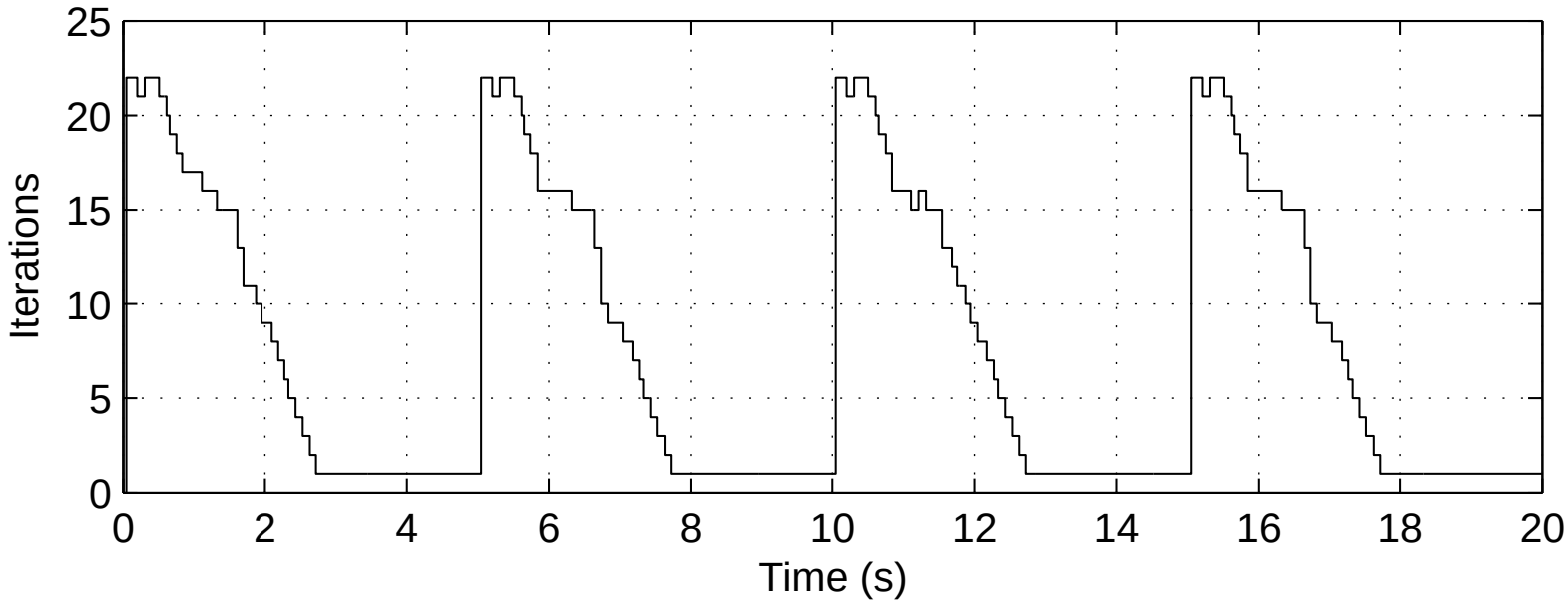
The formulation leads to a quadratic programming problem with linear inequality constraints

$$\begin{aligned} \text{minimize} \quad & \Delta\mathcal{U}^T(k)\mathcal{H}\,\Delta\mathcal{U}(k) - \Delta\mathcal{U}^T(k)\mathcal{G}(k) + C \\ \text{subject to} \quad & \Omega\,\Delta\mathcal{U}(k) \leq \omega(k) \end{aligned}$$

olved for $\Delta\mathcal{U}(k) = [\,\Delta\hat{u}(k) \ldots \Delta\hat{u}(k + H_u - 1)\,]$.

Execution-time properties

- Convex optimization problem solved each sample
- Highly varying execution times → worst-case pessimistic
- Execution time depends on external factors such as reference signals and disturbances



Early termination

- Optimization may be aborted any time after a feasible solution has been obtained
- Based on recent stability results [Scocaert et. al. 1999]
- A solution is feasible if it fulfills the constraints and obtains a lower cost than in the previous sample

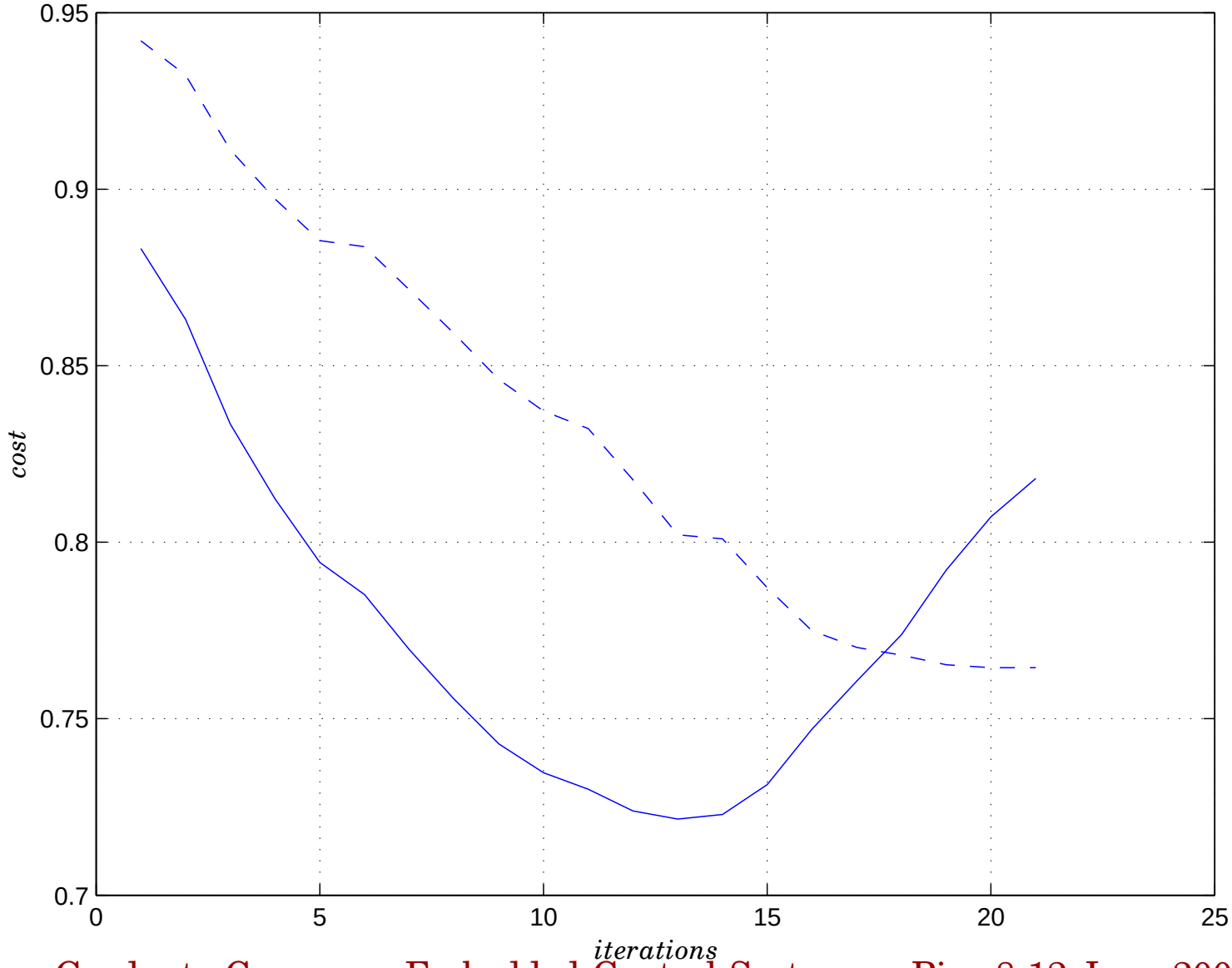
Trade-off cost vs delay

- Assuming a constant process delay, $\tau < h$, over the prediction horizon
- Leads to an augmented process model
- The matrices in the cost function are computed as functions of the delay, τ

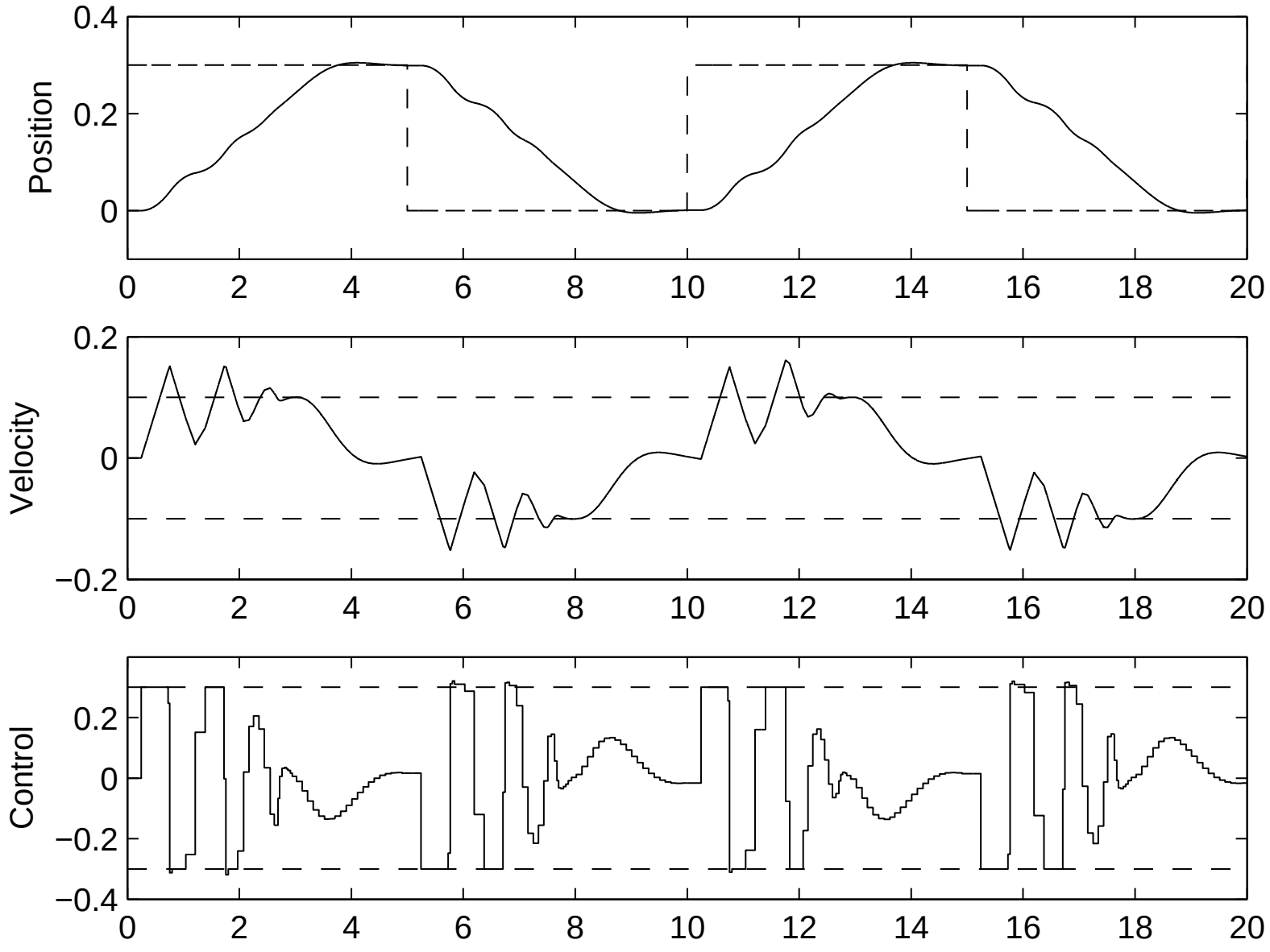
$$J_d(\Delta\mathcal{U}_i, \tau) = \Delta\mathcal{U}_i^T \mathcal{H}(\tau)\Delta\mathcal{U}_i - \Delta\mathcal{U}_i^T \mathcal{G}(\tau) + \mathcal{C}(\tau)$$

- The optimization algorithm is terminated based on this delay-dependent cost

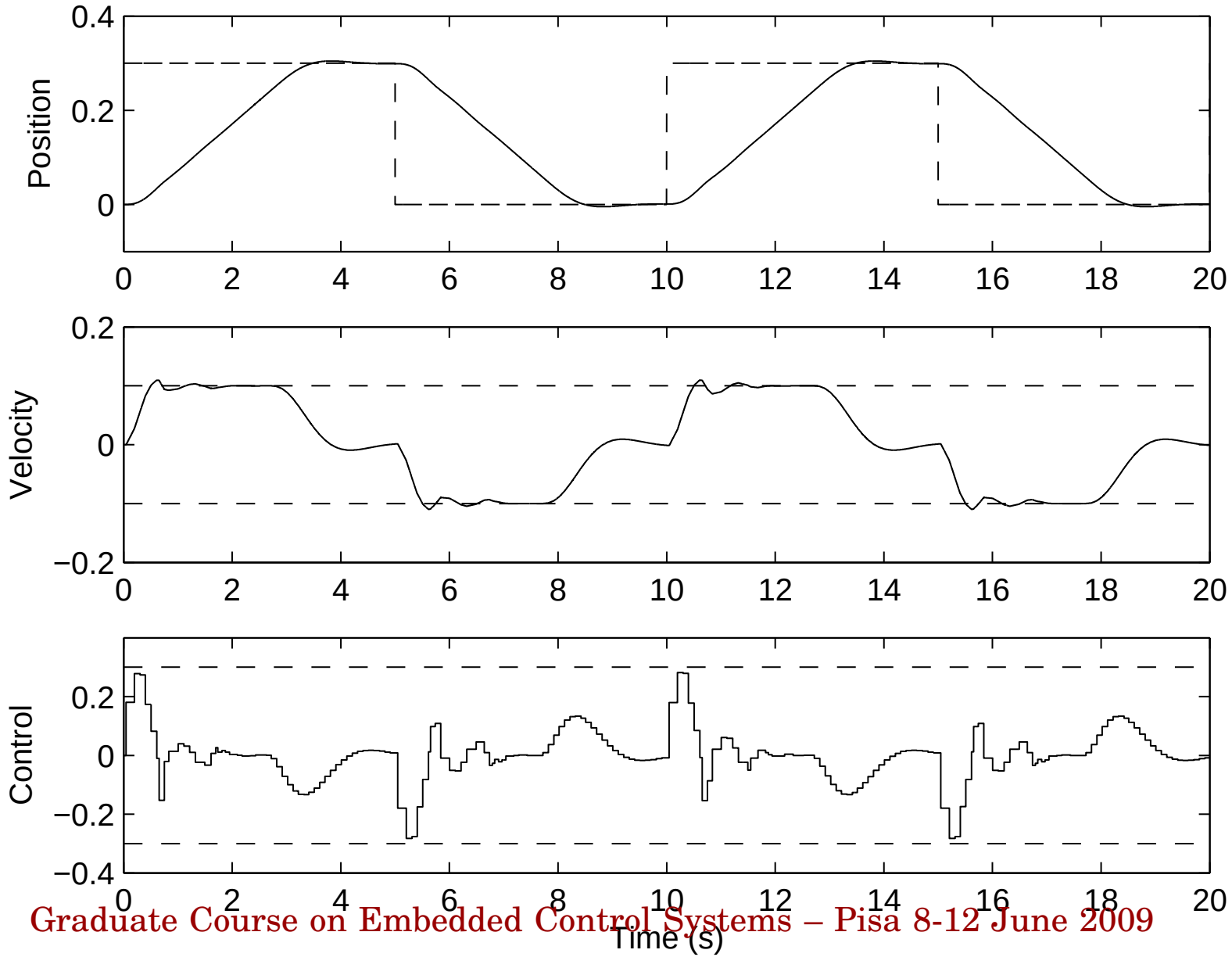
Trade-off cost vs delay



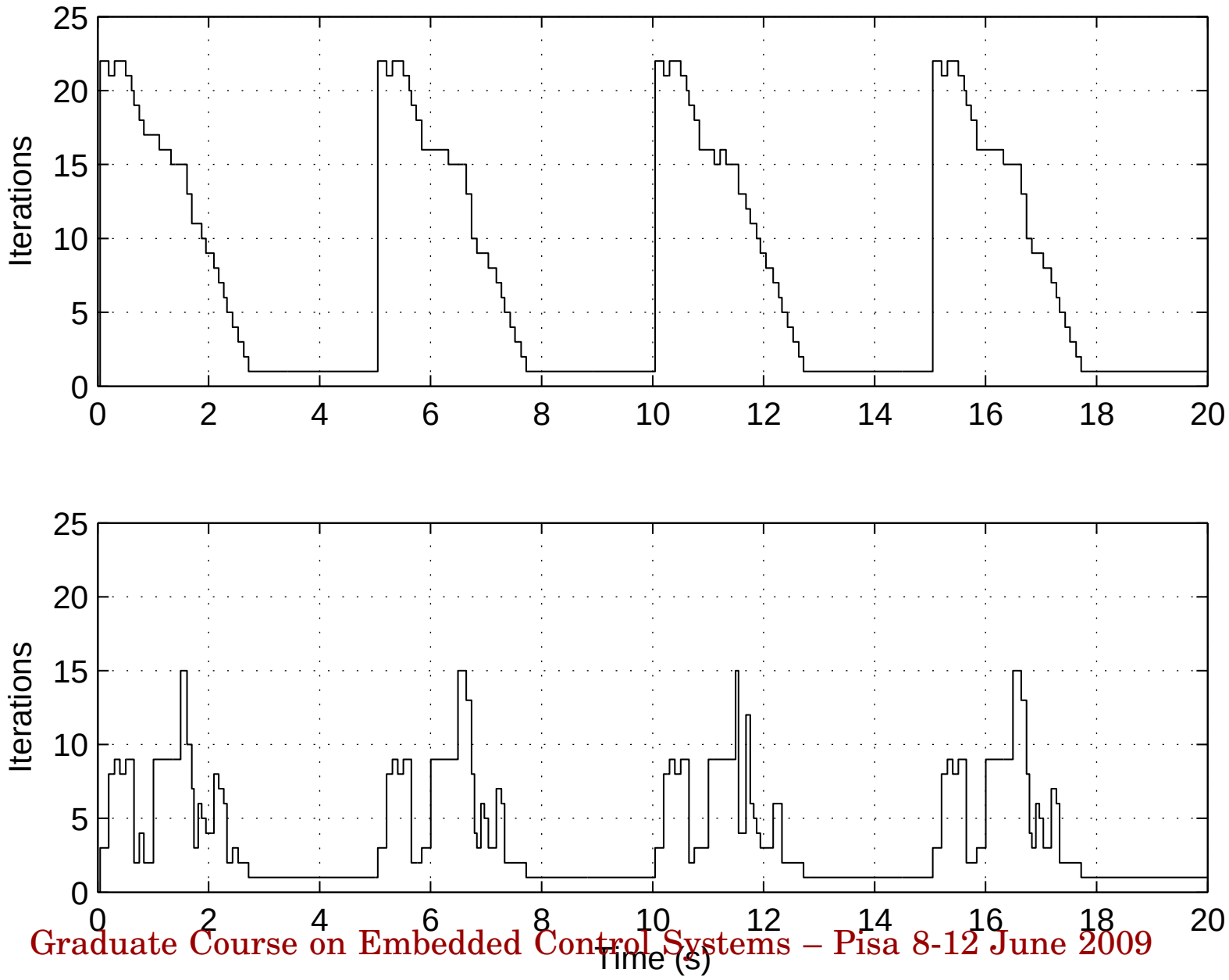
Full optimization



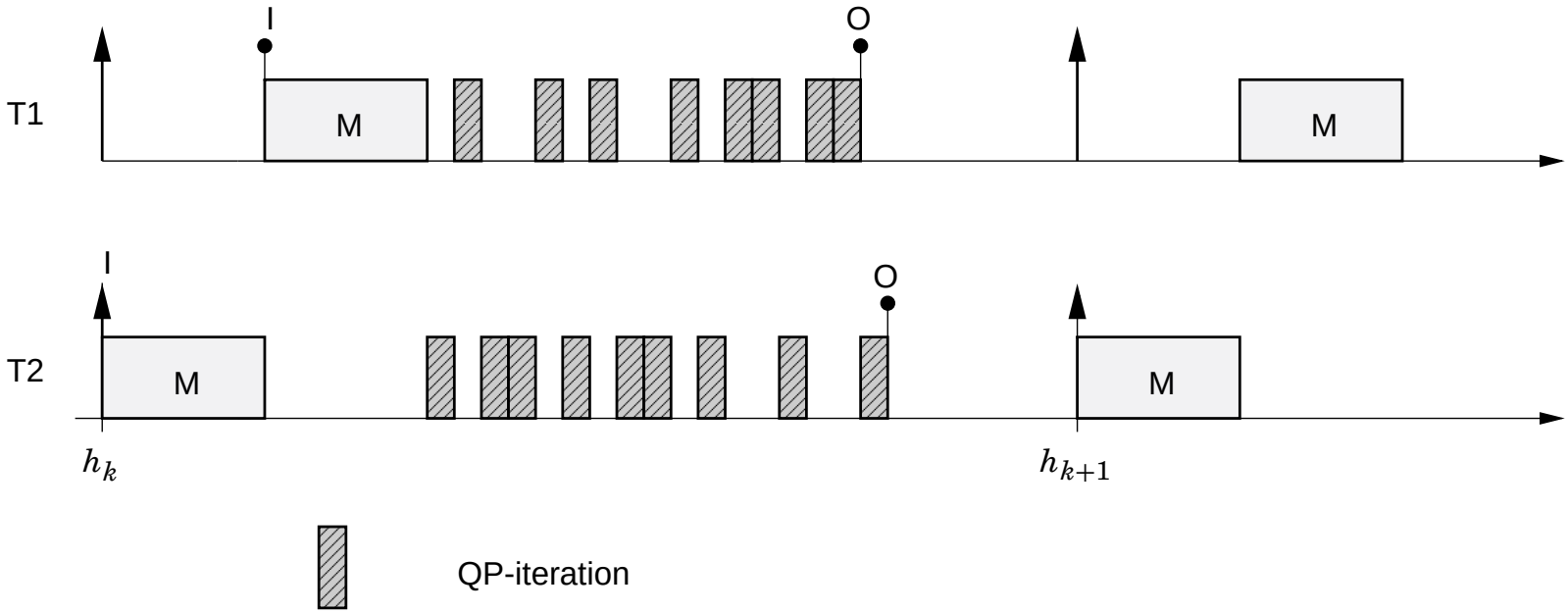
Exploiting the trade-off



Iteration comparison

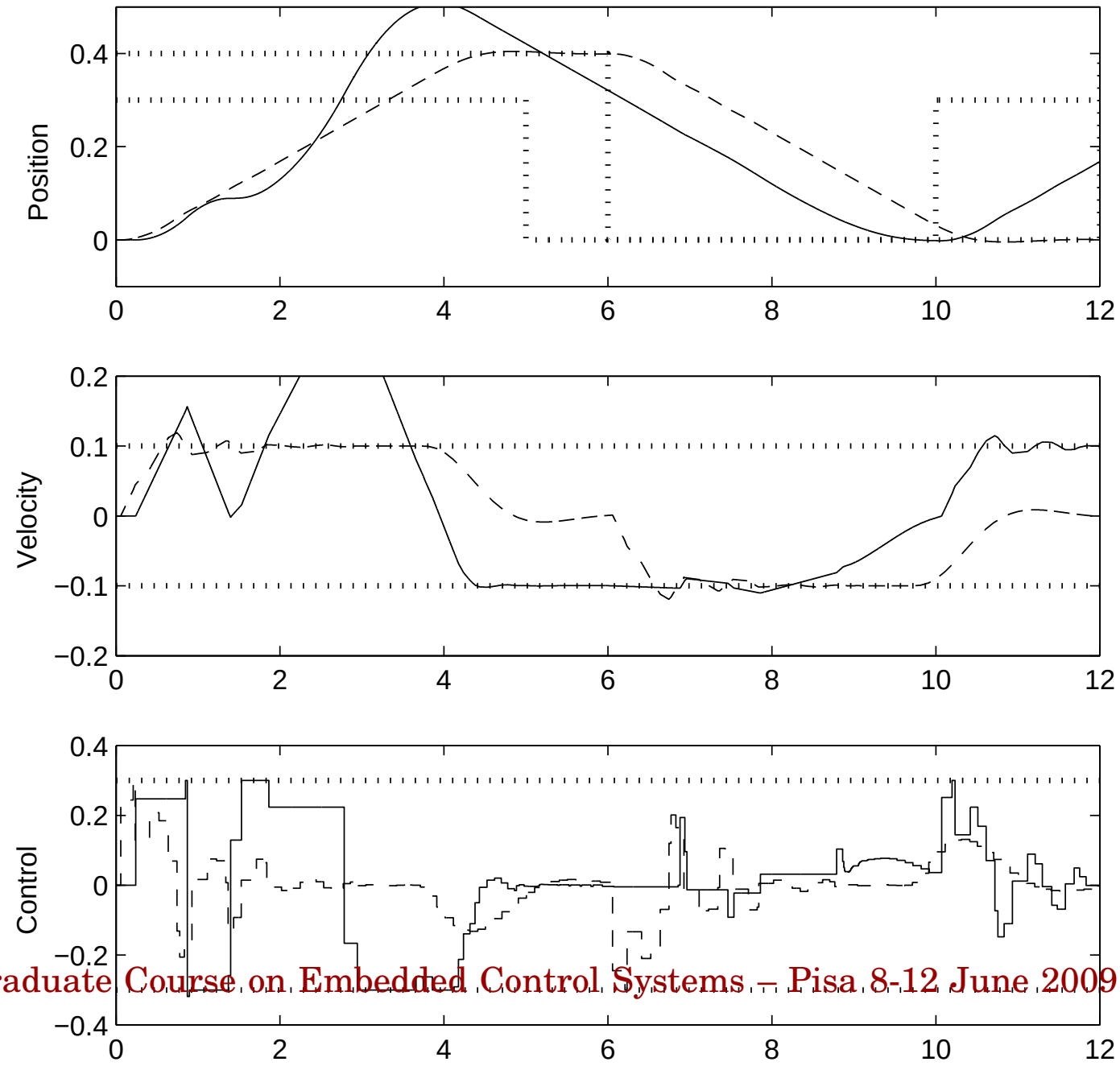


Scheduling of multiple MPCs

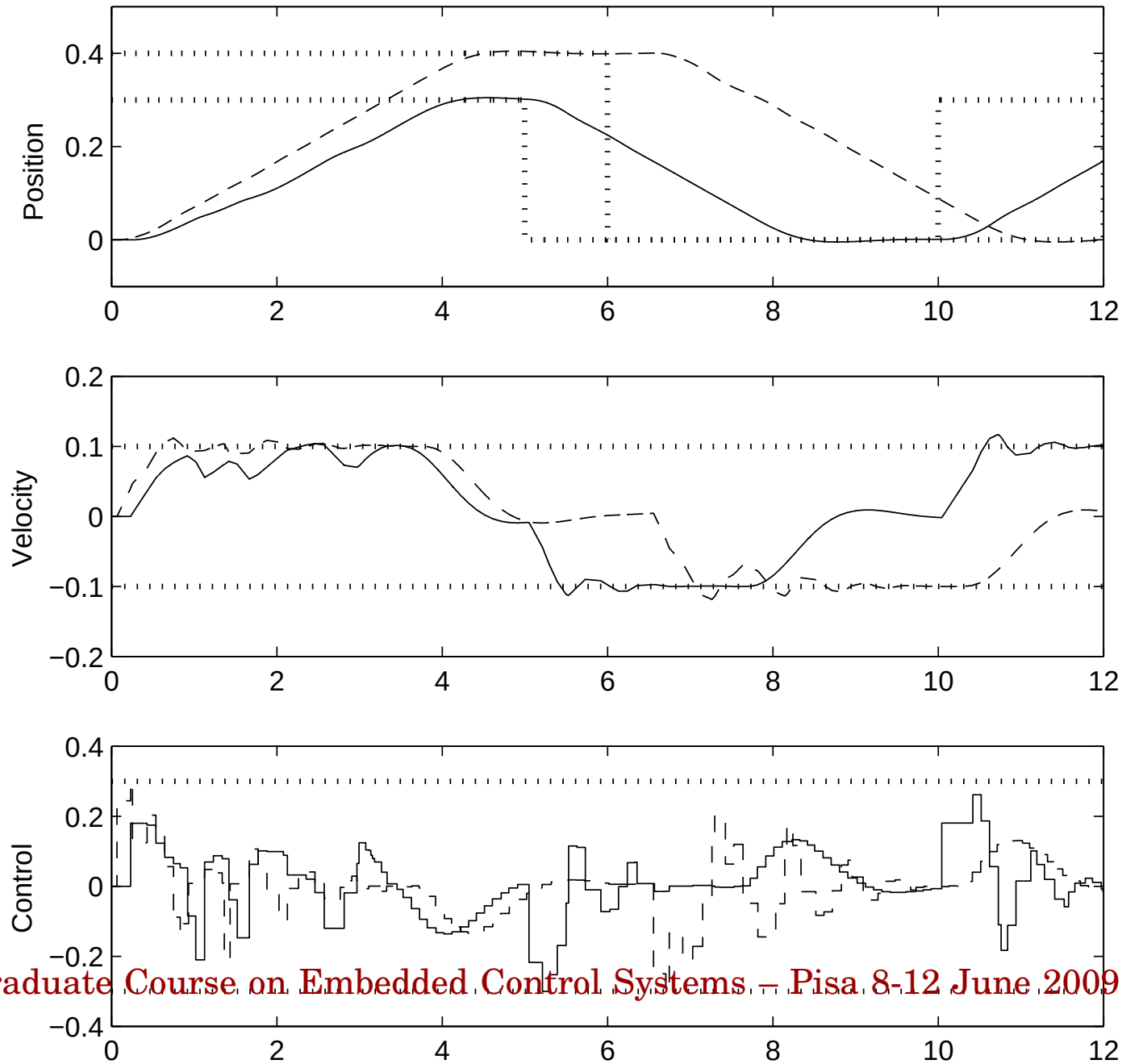


- Mandatory part consists of finding a feasible solution
- Remaining QP-iterations scheduled using the cost functions as dynamic priorities
- Reflects the relative importance of the tasks

Fixed-priority scheduling



Feedback scheduling

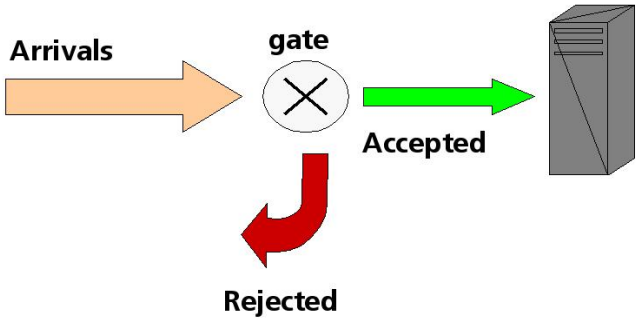


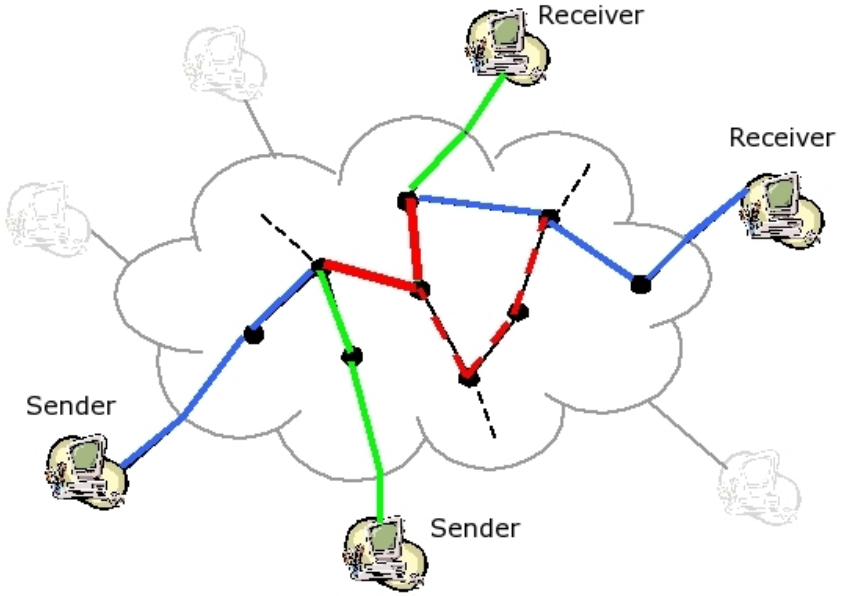
Outline

- Overview
- Feedback control of Linux
- Feedback scheduling of control tasks
  - Heuristic
  - LQ-optimal
  - MPC
- **Control of web servers**

What to control?

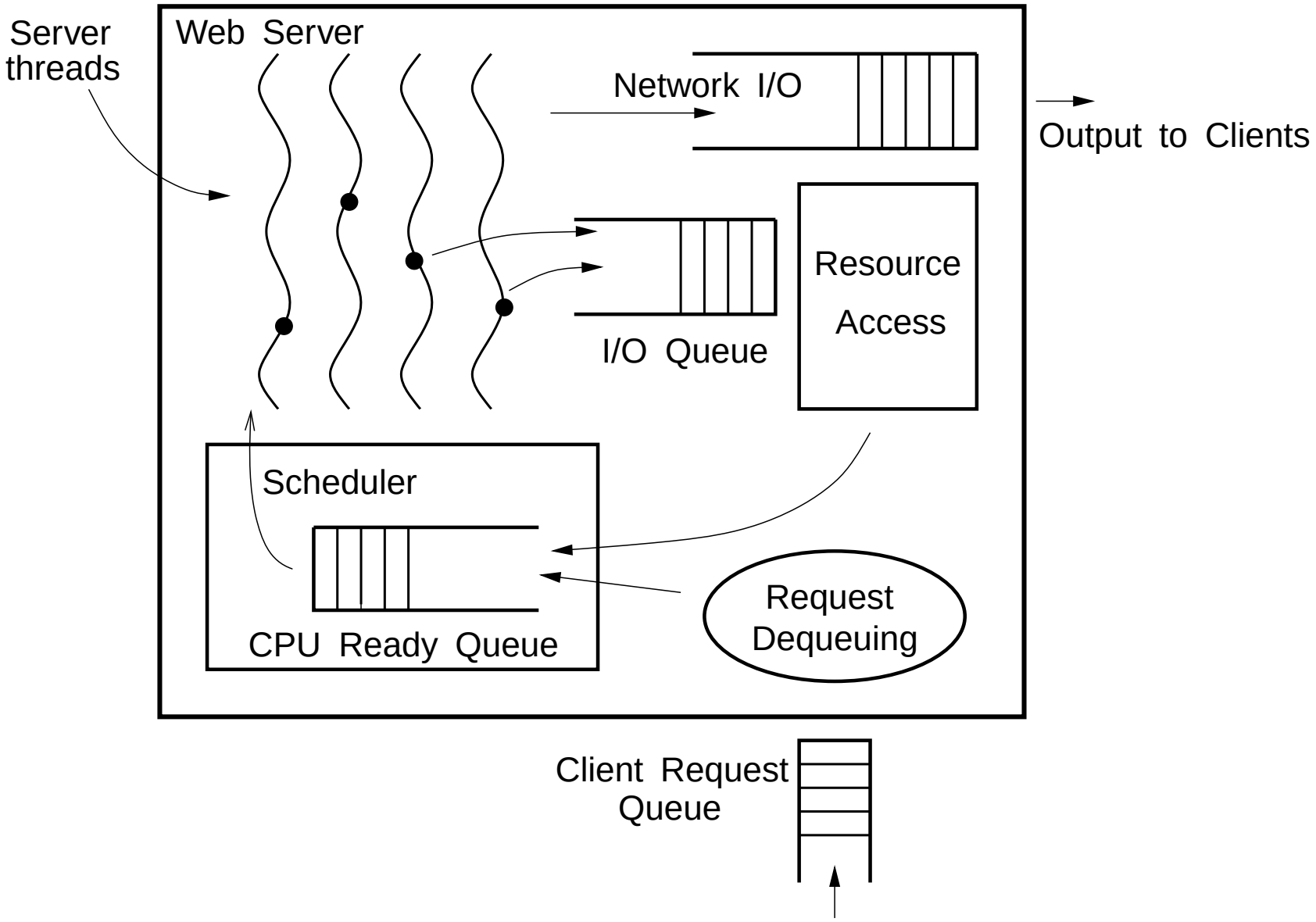
- Temporal
  - local (at server)
  - global (End-to-End/TCP)
- Spatial (routing)





We will focus on temporal control issues at the server.

Web service performance control

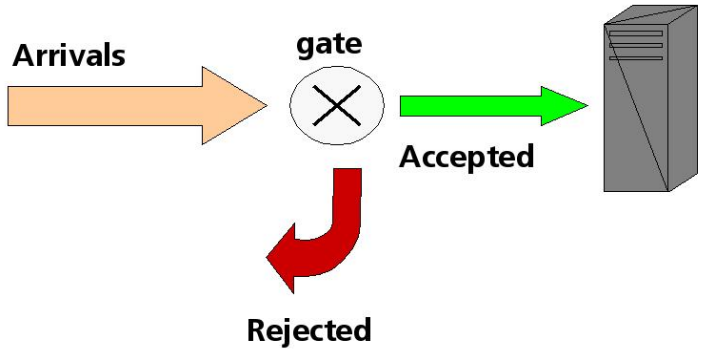


Control objective

- The main objective is to control the service delay of individual requests.
- Can be controlled directly or indirectly by manipulating the server queue lengths.
- The stochastic nature of the system requires averaging (inherent in the non-linear flow model).
- Want to be able to control both long-term averages and transient responses.

Actuator mechanisms

- The difference between the service rate, μ, and the arrival rate, λ, determines the delay experienced by the requests.
- Changing the arrival rate, admission control:



- Changing the service rate:
  - Number of server threads
  - Quality adaptation
  - Dynamic voltage scaling

Absolute delay control

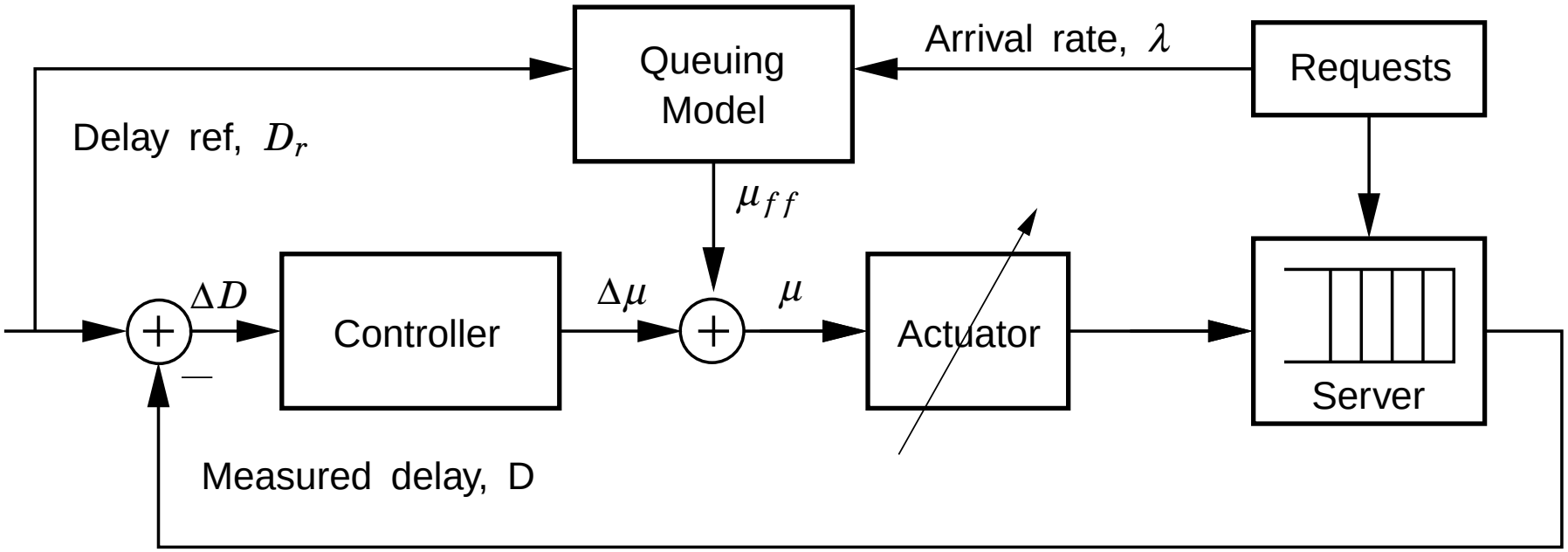
Queuing Model Based Absolute Delay Control

- L. Sha, X. Liu, UIUC and Y. Lu, T. Abdelzaher, UVa

Control objective

- Want to keep the average timing delay experienced by users close to a desired value, D_r.
- The delay specification, D_r, relates to the QoS agreement with the end user.
- Delays consistently longer than the specification are unacceptable to the users,
- and delays consistently shorter than the specification indicate over-provisioning of resources.

Absolute delay control



Key ideas

- Use queuing theory to model the non-linear behavior of the web server.
- Use the steady-state solution of the queuing model as feed-forward control to bring the system to an equilibrium point near the desired delay set-point.
- Example: M/M/1 queuing model where $\hat{D} = \frac{1}{\mu - \lambda}$. Use feed-forward control, $\mu_{ff} = \frac{1}{D_r} + \lambda$.
- Use linear feedback control to suppress approximation errors and transient errors around the operating point.

Problems

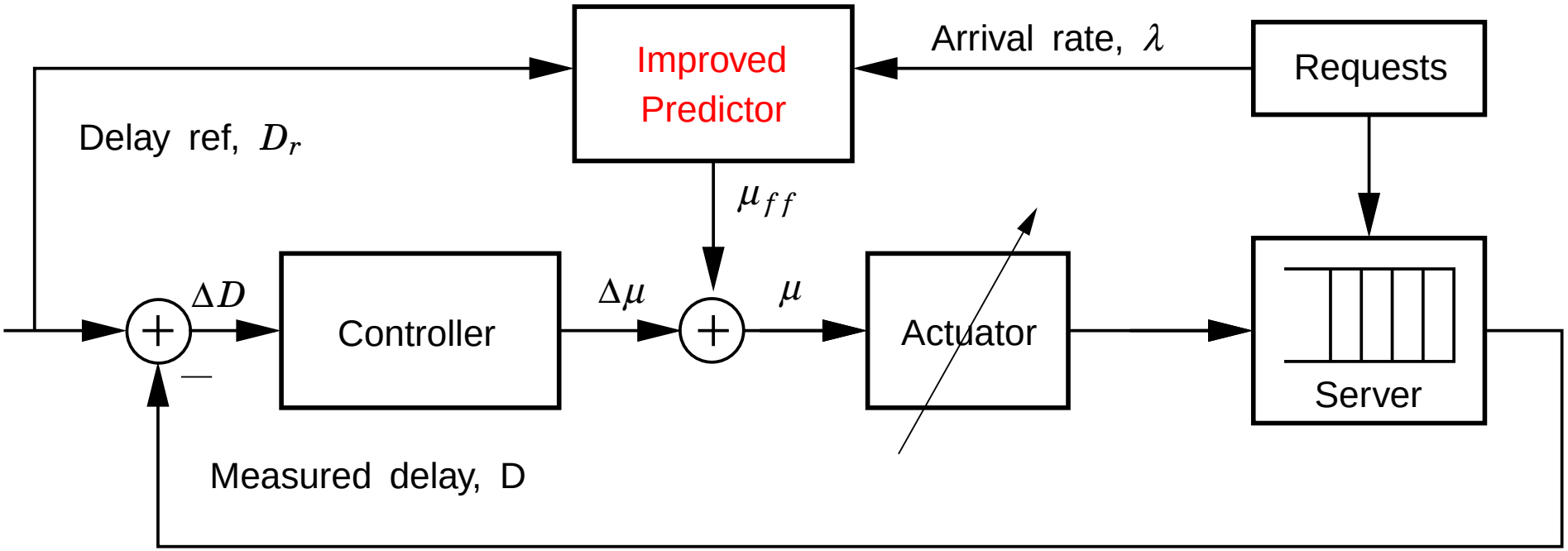
- Queuing theory predicts delay as a function of arrival and service rates.
- The prediction applies only to long-term averages.
- Insensitive to sudden load changes and does not handle transient responses very well.
- Internet load is very bursty and may change abruptly in a frequent manner.
- Inaccurate assumptions in the queuing model, e.g., Poisson distributed arrival and service processes.

Improved feed-forward prediction

Improved Feed-forward Prediction

- Y. Lu, T. Abdelzaher, UVa and D. Henriksson, LTH

Improved feed-forward predictor



- Based on instantaneous measurements instead of long-term averages.

Notation

$\hat{C}$ = average number of processor cycles required by a request

μ = server speed

N = number of waiting requests

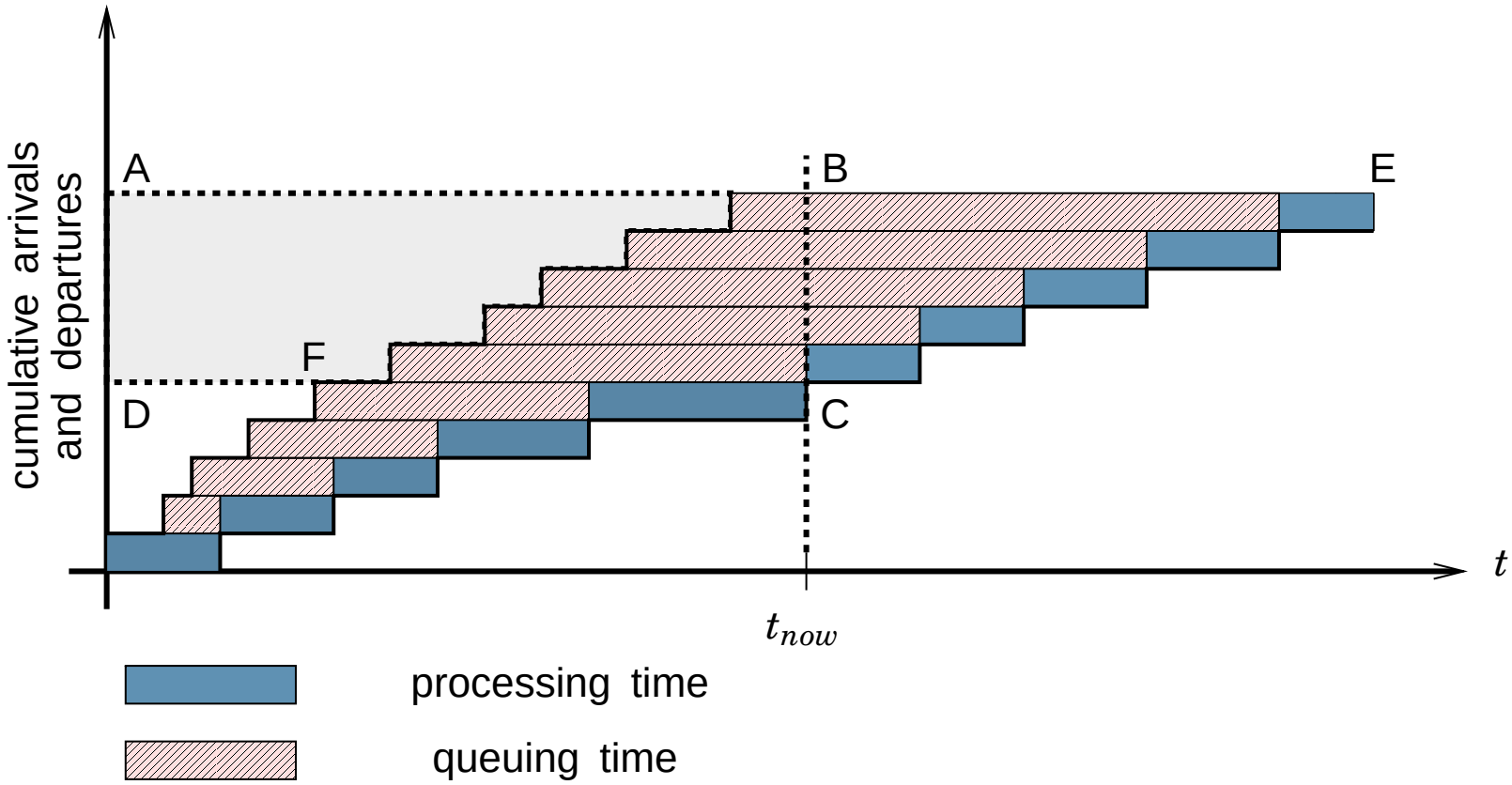
$\hat{D}$ = average delay experience by the N requests

$N\hat{D}$ = total delay experienced by the N requests

$\hat{A}_i = \frac{1}{N}\sum_{k=i}^{i+N-1} A_k$ = the average arrival time

$Q_i = t_{now} - \hat{A}_i$ = average queuing time for the requests being dequeued in the i'th sample

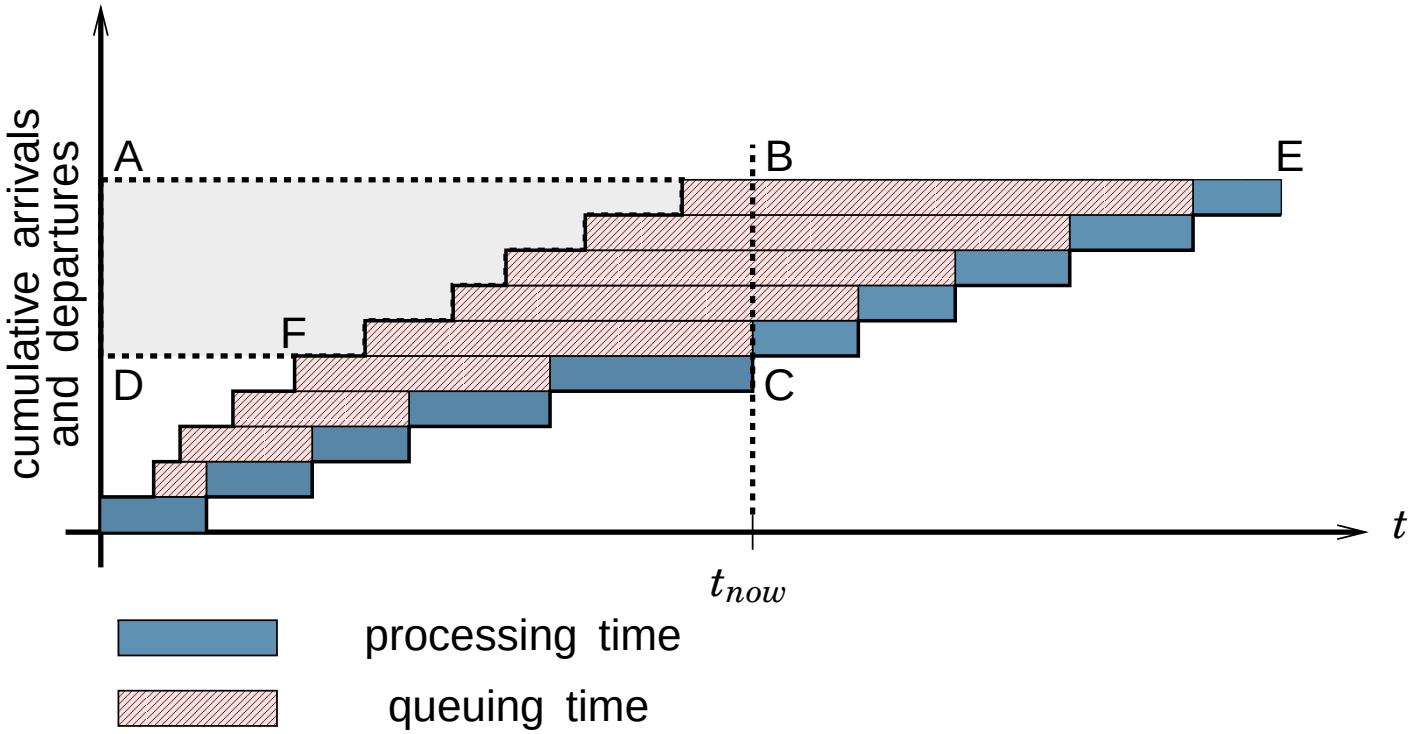
The predictor



$$BECF = ABCD + BEC - ABFD$$

$$N\hat{D} = N \cdot t_{now} + \frac{N \cdot (N\hat{C}/\mu)}{2} - \sum_{k=i}^{i+N-1} A_k$$

The predictor



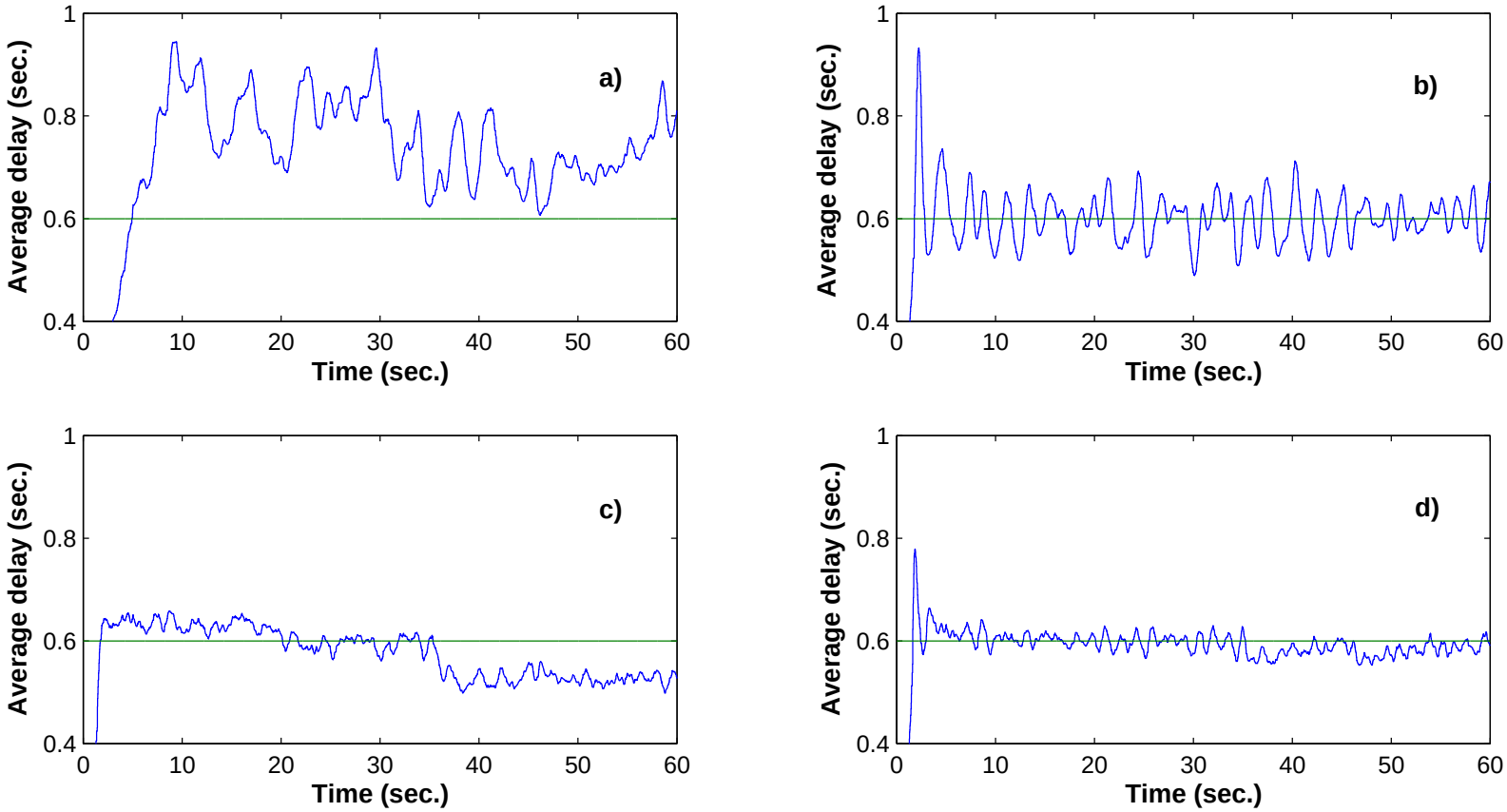
$$\hat{D} = t_{now} - \hat{A} + \frac{N\hat{C}}{2\mu} = \hat{Q} + \frac{N\hat{C}}{2\mu}$$

$$\mu_{ff} = \frac{N\hat{C}}{2(D_r - \hat{Q})}$$

The feedback controller

- Event-triggered PI-controller with sliding window action.
- Need a long observation window, N_{obs}, to accurately estimate the average values of arrival rates and processing times of requests.
- Long observation window does not imply slow control action. Control updated every $N < N_{obs}$ event (request departure).
- Quick update steps reduce the variance and control efforts in each sample.
- The PI-controller is implemented using gain-scheduling
  - tuned for different operating points (arrival rate and delay set-point, D_r).
- Anti-windup crucial.

Simulations



- a: M/M/1, b: M/M/1 + PI, c: Predictor d: Predictor + PI