

Principles of Communications

E.G.Class 2 – 26/4/16

Jon Crowcroft,

<http://www.cl.cam.ac.uk/~jac22>

<http://www.cl.cam.ac.uk/teaching/1516/PrincComm/>

Generic Advice

- All lectured material is examinable this year.
Questions should contain any equations needed
But you may have to explain the terms
And know how to use them 😊
- Problem sheets 3 & 4 (but ignore question 1 on problem set 3, plus 2011 exam question 10 paper 8, as these are no longer relevant) are at:
<http://www.cl.cam.ac.uk/teaching/1314/PrincComm/problems/PoC3.pdf>
<http://www.cl.cam.ac.uk/teaching/1314/PrincComm/problems/PoC4.pdf>
- A problem (challenging) is at the end of these slides...

Flow Control & Scheduling Linked

- Admitting packets or flows of packets
 - Feedback at flow setup, or per packet
 - Controller needs to measure to make decisions
 - Drop packet, or block call....
 - Similar problem even inside switch fabric
- Complexity in scheduler all about
 - State
 - weights
 - Differentiating flow weights & policing

Conservation law [1]

- FCFS is work-conserving:
 - not idle if packets waiting
- Reduce delay of one flow, increase the delay of one or more others
- We can not give *all* flows a lower delay than they would get under FCFS

$$\sum_{n=1}^N \rho_n q_n = C$$

$$\rho_n = \lambda_n \mu_n$$

ρ_n : mean link utilisation

q_n : mean delay due to scheduler

C : constant [s]

λ_n : mean packet rate [p/s]

μ_n : mean per – packet service rate [s/p]

The max-min fair share criteria

- Flows are allocated resource in order of increasing demand
- Flows get no more than they need
- Flows which have not been allocated as they demand get an equal share of the available resource
- Weighted max-min fair share possible
- If max-min fair → provides protection

$$m_n = \min(x_n, M_n) \quad 1 \leq n \leq N$$

$$M_n = \frac{C - \sum_{i=1}^{n-1} m_i}{N - n + 1}$$

C : capacity of resource (maximum resource)

m_n : actual resource allocation to flow n

x_n : resource demand by flow n , $x_1 \leq x_2 \cdots \leq x_N$

M_n : resource available to flow n

- Example:
- $C = 10$, four flow with demands of 2, 2.6, 4, 5
- actual resource allocations are 2, 2.6, 2.7, 2.7

Simple priority queuing

- K queues:
 - $1 \leq k \leq K$
 - queue $k + 1$ has greater priority than queue k
 - higher priority queues serviced first
- ✓ Very simple to implement
- ✓ Low processing overhead
- Relative priority:
 - no deterministic performance bounds
- ✗ Fairness and protection:
 - not max-min fair: starvation of low

Generalised processor sharing (GPS)

- Work-conserving
- Provides max-min fair share
- Can provide weighted max-min fair share
- Not implementable:
 - used as a reference for comparing other schedulers
 - serves an infinitesimally small amount of data from flow i
- Visits flows round-robin

$$\phi(n) \quad 1 \leq n \leq N$$

$$S(i, \tau, t) \quad 1 \leq i \leq N$$

$$\frac{S(i, \tau, t)}{S(j, \tau, t)} \geq \frac{\phi(i)}{\phi(j)}$$

$\phi(n)$: weight given to flow n

$S(i, \tau, t)$: service to flow i in interval $[\tau, t]$

flow i has a non – empty queue

GPS – relative and absolute fairness

- Use fairness bound to evaluate GPS emulations (GPS-like schedulers)
- Relative fairness bound:
 - fairness of scheduler with respect to other flows it is servicing
- Absolute fairness bound:
 - fairness of scheduler compared to GPS for the same flow

$$RFB = \left| \frac{S(i, \tau, t)}{g(i)} - \frac{S(j, \tau, t)}{g(j)} \right|$$

$$AFB = \left| \frac{S(i, \tau, t)}{g(i)} - \frac{G(i, \tau, t)}{g(i)} \right|$$

$S(i, \tau, t)$: actual service for flow i in $[\tau, t]$

$G(i, \tau, t)$: GPS service for flow i in $[\tau, t]$

$$g(i) = \min \{g(i, 1), \dots, g(i, K)\}$$

$$g(i, k) = \frac{\phi(i, k)r(k)}{\sum_{j=1}^N \phi(j, k)}$$

$\phi(i, k)$: weight given to flow i at router k

$r(k)$: service rate of router k

$1 \leq i \leq N$ flow number

$1 \leq k \leq K$ router number

Weighted round-robin (WRR)

- Simplest attempt at GPS
- Queues visited round-robin in proportion to weights assigned
- Different mean packet sizes:
 - weight divided by mean packet size for each queue
- Mean packets size unpredictable:
 - may cause unfairness
- Service is fair over long timescales:
 - must have more than one visit to each flow/queue
 - short-lived flows?
 - small weights?
 - large number of flows?

Deficit round-robin (DRR)

- DRR does not need to know mean packet size
- Each queue has deficit counter (dc): initially zero
- Scheduler attempts to serve one quantum of data from a non-empty queue:
 - packet at head served if $\text{size} \leq \text{quantum} + \text{dc}$
 $\text{dc} \leftarrow \text{quantum} + \text{dc} - \text{size}$
 - else $\text{dc} += \text{quantum}$
- Queues not served during round build up “credits”:
 - only non-empty queues
- Quantum normally set to max expected packet size:
 - ensures one packet per round, per non-empty queue
- RFB: $3T/r$ ($T = \text{max pkt service time}$, $r = \text{link rate}$)
- Works best for:
 - small packet size
 - small number of flows

Weighted Fair Queuing (WFQ) [1]

- Based on GPS:
 - GPS emulation to produce **finish-numbers** for packets in queue
 - Simplification: GPS emulation serves packets bit-by-bit round-robin
- **Finish-number:**
 - the time packet would have completed service under (bit-by-bit) GPS
 - packets tagged with finish-number
 - smallest finish-number across queues served first
- **Round-number:**
 - execution of round by bit-by-bit round-robin server
 - finish-number calculated from round number
- If queue is empty:
 - finish-number is:
number of bits in packet + round-number
- If queue non-empty:
 - finish-number is:
highest current finish number for queue + number of bits in packet

Weighted Fair Queuing (WFQ) [2]

$$F(i, k, t) = \max\{F(i, k - 1, t), R(t)\} + P(i, k, t)$$

$F(i, k, t)$: finish - number for packet k
on flow i arriving at time t

$P(i, k, t)$: size of packet k on flow i
arriving at time t

$R(t)$: round - number at time t

$$F_{\phi}(i, k, t) = \max\{F_{\phi}(i, k - 1, t), R(t)\} + \frac{P(i, k, t)}{\phi(i)}$$

$\phi(i)$: weight given to flow i

- Rate of change of $R(t)$ depends on number of active flows (and their weights)
- As $R(t)$ changes, so packets will be served at different rates

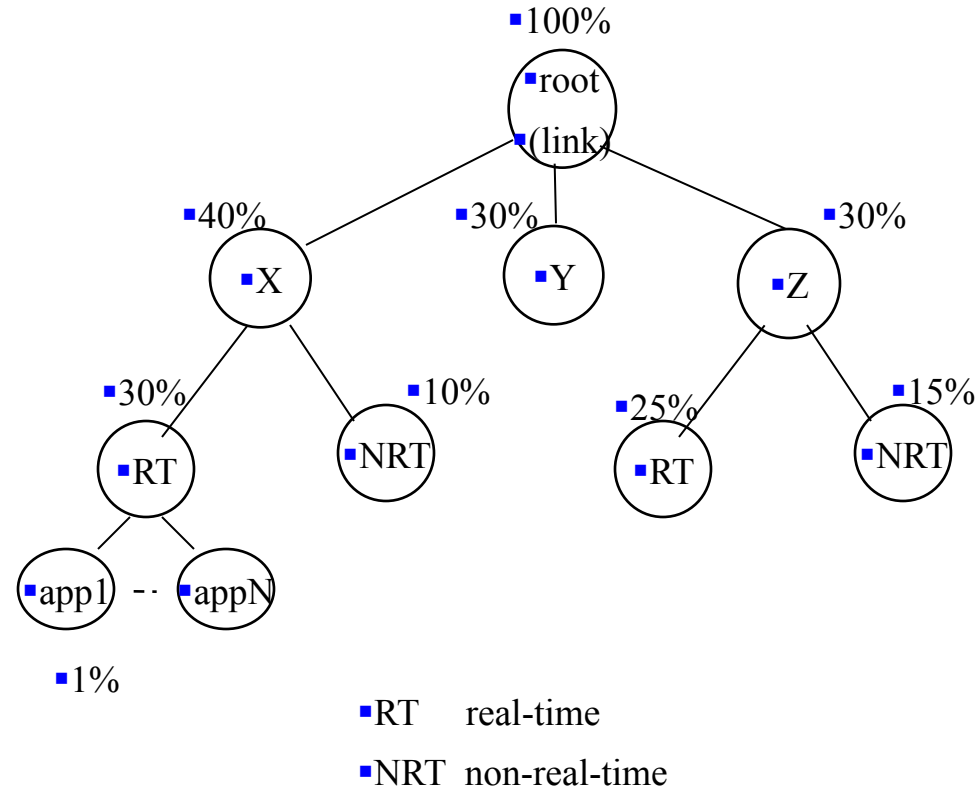
- Flow completes (empty queue):
 - one less flow in round, so
 - R increases more quickly
 - so, more flows complete
 - R increases more quickly
 - etc. ...
 - **iterated deletion** problem
- WFQ needs to evaluate R each time packet arrives or leaves:
 - processing overhead

Weighted Fair Queuing (WFQ) [3]

- Buffer drop policy:
 - packet arrives at full queue
 - drop packets already in queue, in order of decreasing finish-number
- Can be used for:
 - best-effort queuing
 - providing guaranteed data rate and deterministic end-to-end delay
- WFQ used in “real world”
- Alternatives also available:
 - self-clocked fair-queuing (SCFQ)
 - worst-case fair weighted fair queuing (WF²Q)

Class-Based Queuing

- Hierarchical link sharing:
 - link capacity is shared
 - class-based allocation
 - policy-based class selection
- Class hierarchy:
 - assign capacity/priority to each node
 - node can “borrow” any spare capacity from parent
 - fine-grained flows possible
- Note: this is a queuing mechanism: requires use of a scheduler



1. Deterministic latency bounding

- Learned what I was teaching wrong!
- I used to say:
 - Integrated Service too complex
 - Admission&scheduling hard
 - Priority Queue can't do it
 - PGPS computation for latency?
- I present Qjump scheme, which
 - Uses intserv (PGPS) style admission ctrl
 - Uses priority queues for service levels

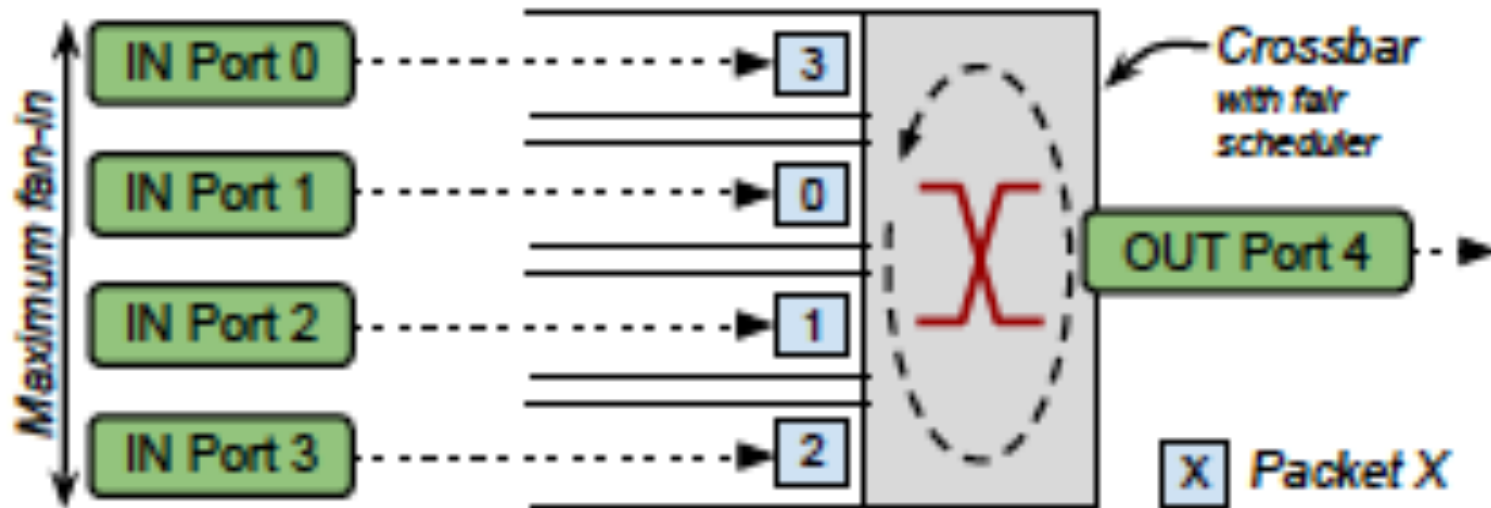
Data Center Latency Problem

- Tail of the distribution,
 - due to long/bursty flows interfering
- Need to separate classes of flow
 - Low latency are usually short flows (or RPCs)
 - Bulk transfers aren't so latency/jitter sensitive

Data Center Qjump Solution

- In Data Center, not general Internet!
 - can exploit topology &
 - traffic matrix &
 - source behaviour knowledge
- Regular, and simpler topology key
- But also largely “cooperative” world...

5 port virtual output queued

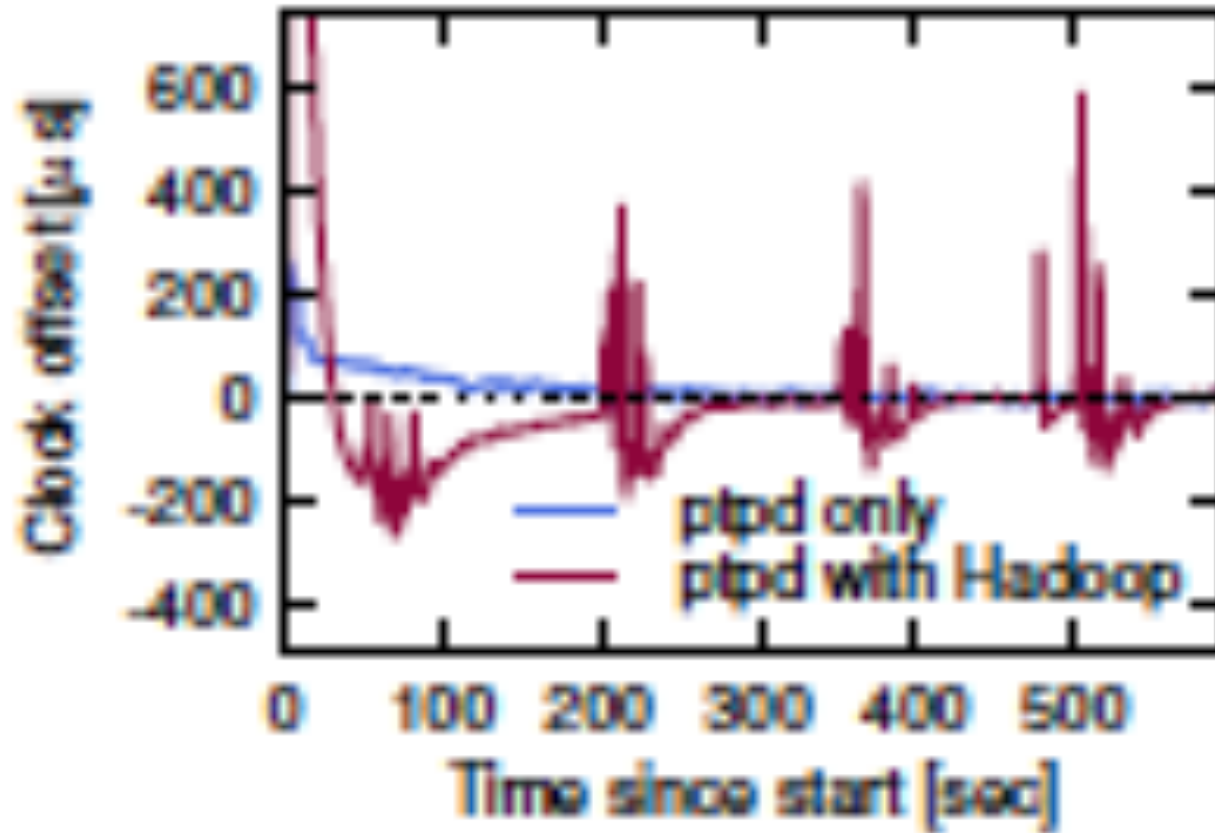


Latency of ping v. iperf

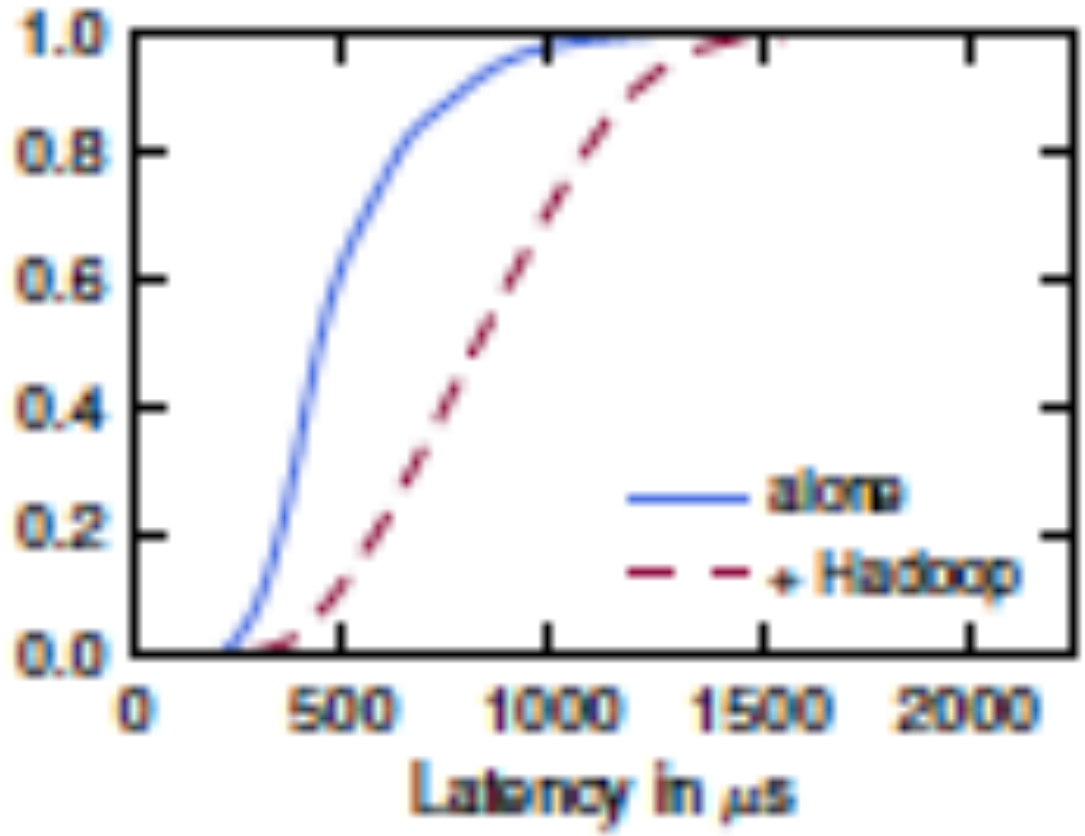
Setup	<i>10th%</i>	<i>50th%</i>	<i>90th%</i>	<i>99th%</i>
otherwise idle network	36	85	118	126
independent hosts	37	110	120	130
shared host egress	168	228	259	268
shared host ingress	55	125	249	278
shared ingress & egress	171	221	224	229
shared switch queue	1,790	1,920	2,010	2,100

Table 1: Latency of ping vs. iperf with various degrees of host resource sharing, all numbers in μs over 5,000 samples.

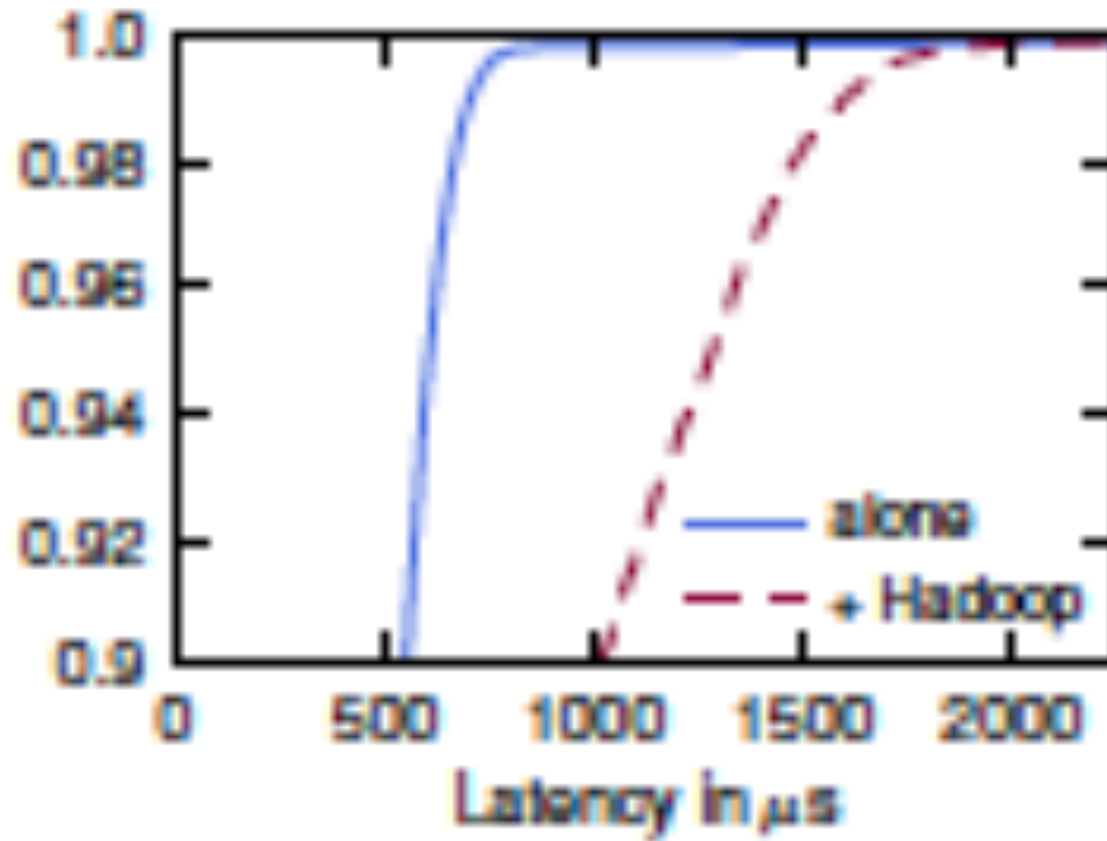
Hadoop perturbs time synchrony



Hadoop perturbs memcached



Hadoop perturbs Naiad



Qjump – two pieces

1. At network config time

- Compute a set of (8^*) rates based on
- Traffic matrix & hops \Rightarrow fan in (f)

2. At run time

- Flow assigns itself a priority/rate class
- subject it to (per hypervisor) rate limit

* 8 arbitrary – but often h/w supported 😊

Data Center wide bound

n hosts in network,

packet experience at most max network

fan-in - 1 = n - 2 \approx n pkts worth of delay.

Packet size P will take P/R seconds to transmit at link-rate R,

Bound maximum interference delay at:

worst case end-to-end delay $\leq n \times P/R + \epsilon$

Where...

n is number of hosts,

P the maximum packet size (in bits),

R is the rate of the slowest link in bits per second

ϵ is the cumulative processing delay introduced by switch hops

Qjump – self admit/policer (per hypervisor, at least)

Algorithm 1 QJUMP rate-limiter pseudocode

```
1: epoch_cycles  $\leftarrow$  to_cycles(network epoch)
2: function QJUMPRATELIMITER(buffer)
3:   cyclesnow  $\leftarrow$  rdtsc /*read timestamp counter*/
4:   level  $\leftarrow$  buffer.socket.priority
5:   if cyclesnow > timeout then
6:     timeout  $\leftarrow$  timeout + epoch_cycles
7:     bucket[level]  $\leftarrow$  tokens[level]
8:   if buffer.size > bucket[level] then
9:     return DROP
10:  bucket[level]  $\leftarrow$  bucket[level] - buffer.size
11:  return SENDTOHWQUEUE[level]
```

Why use Transforms?

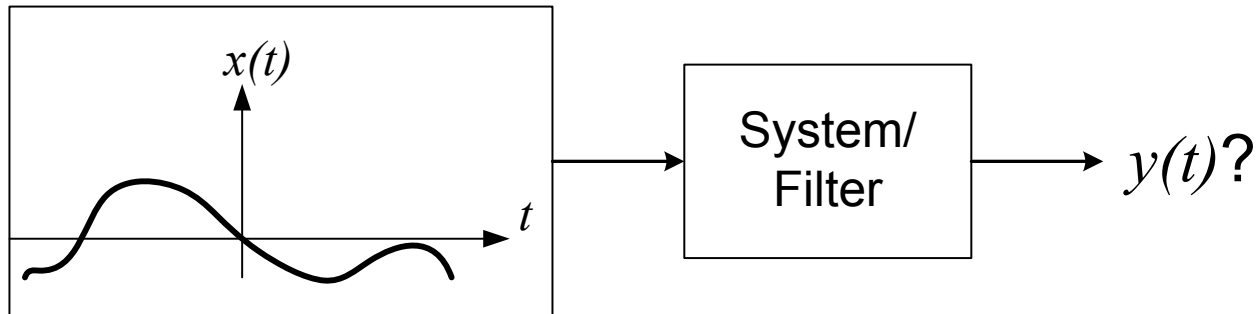
- Transforms are not simply math curiosity sketched at the corner of a woodstove by ol' Frenchmen.
- Way to reframe a problem in a way that makes it easier to understand, analyze and solve.

Which Transform to Use?

<i>Application</i>	<i>Continuous Domain</i>	<i>Discrete Domain</i>
Signal Processing	Fourier T.	Discrete F.T. (DFT/FFT)
Control Theory	Laplace T.	z-Transform

Typical Problem

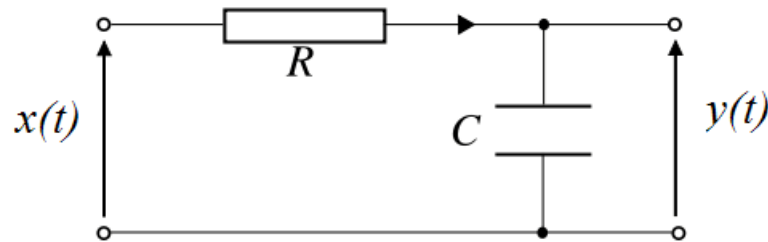
- Given an input signal $x(t)$, what is the output signal $y(t)$ after going through the system?



- To solve it in the time domain (t) is cumbersome!

Integrating Differential Equation?

- Let's have a simple first order low-pass filter with resistor R and capacitor C :



- The system is described by diff. eq.:

$$RCy'(t) + y(t) = x(t)$$

- To find a solution, we can integrate. *Ugh!*

Laplace Transform

- Formal definition:

$$\mathbf{L}[f(t)] = F(s) = \int_0^{\infty} f(t)e^{-st} dt$$

- Compare this to FT:

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i\omega t} dt$$

- Small differences:

- *Integral from 0 to ∞ to for Laplace*
 - *$f(t)$ for $t < 0$ is not taken into account*
- *$-s$ instead of $-i\omega$*

■ What is Laplace Transform?

- The Laplace transform of a function $f(t)$ is defined as:

$$F(s) = \ell[f(t)] = \int_0^{\infty} f(t)e^{-st} dt$$

- The inverse Laplace transform is defined as:

$$f(t) = \ell^{-1}[F(s)] = \frac{1}{2j\pi} \int_{\sigma-j\infty}^{\sigma+j\infty} F(s)e^{st} ds$$

where $j = \sqrt{-1}$ and the value of σ is determined by the singularities of $F(s)$.

$$s \equiv \frac{d}{dt}, \quad \frac{1}{s} \equiv \int_0^t dt$$

Why Is Laplace Transform Useful ?

- Model a linear time-invariant analog system as an algebraic system (transfer function).
- In control theory, Laplace transform converts linear differential equations into algebraic equations.
- This is much easier to manipulate and analyze.

An Example

- The Laplace transform of e^{-at} can be obtained by:

$$F(s) = \int_0^{\infty} e^{-at} e^{-st} dt = \int_0^{\infty} e^{-(a+s)t} dt = \frac{-e^{-(s+a)t}}{s+a} \Big|_0^{\infty} = \frac{1}{s+a}$$

■ Linearity property

- These are useful properties:

$$\ell[kf(t)] = k\ell[f(t)] = kF(s)$$

$$\ell[f_1(t) + f_2(t)] = \ell[f_1(t)] + \ell[f_2(t)] = F_1(s) + F_2(s)$$

Name	Time function $f(t)$	Laplace Transform
Unit Impulse	$\delta(t)$	1
Unit Step	$u(t)$	1/s
Unit ramp	t	1/s ²
nth-Order ramp	t^n	n!/s ⁿ⁺¹
Exponential	e^{-at}	1/(s+a)
nth-Order exponential	$t^n e^{-at}$	n!/(s+a) ⁿ⁺¹
Sine	$\sin(bt)$	b/(s ² +b ²)
Cosine	$\cos(bt)$	s/(s ² +b ²)
Damped sine	$e^{-at} \sin(bt)$	b/((s+a) ² +b ²)
Damped cosine	$e^{-at} \cos(bt)$	(s+a)/((s+a) ² +b ²)
Diverging sine	$t \sin(bt)$	2bs/(s ² +b ²) ²

TABLE 2.2 Laplace transform theorems

Item no.	Theorem	Name
1.	$\mathcal{L}[f(t)] = F(s) = \int_{0-}^{\infty} f(t)e^{-st} dt$	Definition
2.	$\mathcal{L}[kf(t)] = kF(s)$	Linearity theorem
3.	$\mathcal{L}[f_1(t) + f_2(t)] = F_1(s) + F_2(s)$	Linearity theorem
4.	$\mathcal{L}[e^{-at} f(t)] = F(s + a)$	Frequency shift theorem
5.	$\mathcal{L}[f(t - T)] = e^{-sT} F(s)$	Time shift theorem
6.	$\mathcal{L}[f(at)] = \frac{1}{a} F\left(\frac{s}{a}\right)$	Scaling theorem
7.	$\mathcal{L}\left[\frac{df}{dt}\right] = sF(s) - f(0-)$	Differentiation theorem
8.	$\mathcal{L}\left[\frac{d^2f}{dt^2}\right] = s^2F(s) - sf(0-) - f'(0-)$	Differentiation theorem
9.	$\mathcal{L}\left[\frac{d^nf}{dt^n}\right] = s^n F(s) - \sum_{k=1}^n s^{n-k} f^{(k-1)}(0-)$	Differentiation theorem
10.	$\mathcal{L}\left[\int_{0-}^1 f(\tau)d\tau\right] = \frac{F(s)}{s}$	Integration theorem
11.	$f(\infty) = \lim_{s \rightarrow 0} sF(s)$	Final value theorem ¹
12.	$f(0+) = \lim_{s \rightarrow \infty} sF(s)$	Initial value theorem ²

Find the Laplace transform of $f(t)=5u(t)+3e^{-2t}$.

• Solution:

$$\ell[5u(t)] = 5\ell[u(t)] = \frac{5}{s}$$



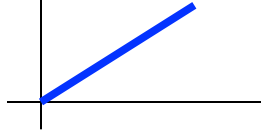
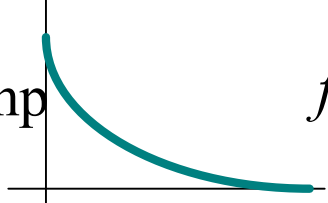
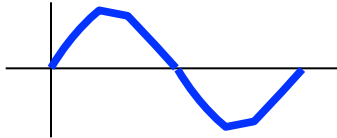
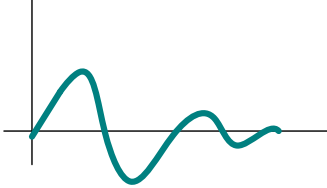
$$\ell[3e^{-2t}] = 3\ell[e^{-2t}] = \frac{3}{s+2}$$

$$F(s) = \frac{5}{s} + \frac{3}{s+2} = \frac{8s+10}{s(s+2)}$$

Another Example:

$$\frac{d^2 y}{dt^2} + 12 \frac{dy}{dt} + 32y = 32u(t)$$

Common Laplace Transform

Name	$f(t)$		$F(s)$
Impulse δ	$f(t) = \begin{cases} 1 & t = 0 \\ 0 & t > 0 \end{cases}$		1
Step	$f(t) = 1$		$\frac{1}{s}$
Ramp	$f(t) = t$		$\frac{1}{s^2}$
Exponential	$f(t) = e^{-at}$		$\frac{1}{s+a}$
Sine	$f(t) = \sin(\omega t)$		$\frac{\omega}{s^2 + \omega^2}$
Damped Sine	$f(t) = e^{-at} \sin(\omega t)$		$\frac{\omega}{(s+a)^2 + \omega^2}$

Laplace Transform Properties

- Similar to Fourier transform:
 - Addition/Scaling

$$L[af_1(t) \pm bf_2(t)] = aF_1(s) \pm bF_2(s)$$

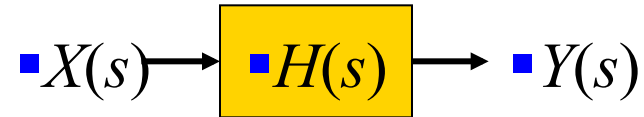
- Convolution

$$\int_0^t f_1(t - \tau)f_2(\tau)d\tau = F_1(s)F_2(s)$$

- Derivation $L\left[\frac{d}{dt}f(t)\right] = sF(s) - f(0\pm)$

Transfer Function $H(s)$

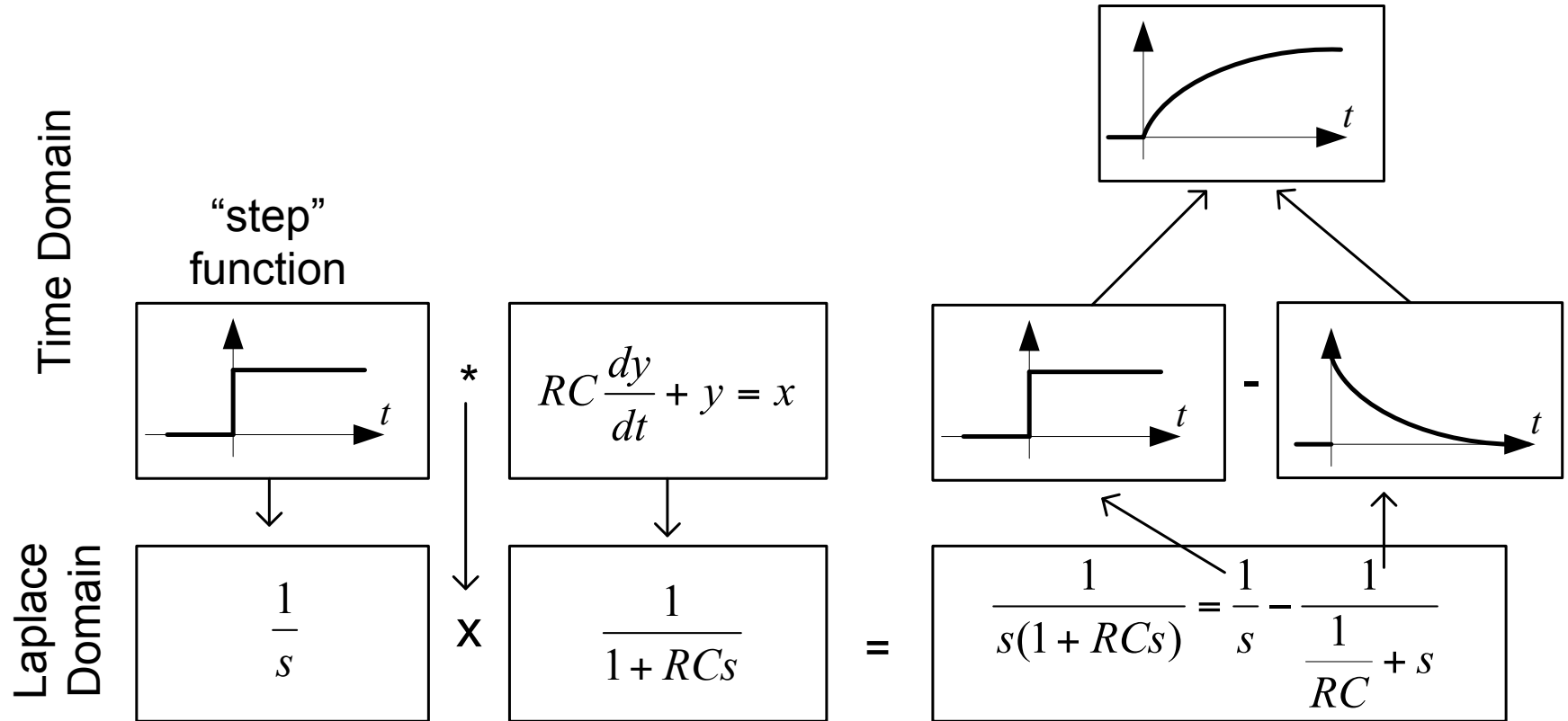
- Definition



- $H(s) = Y(s) / X(s)$

- Relates the output of a linear system (or component) to its input.
- Describes how a linear system responds to an impulse.
- All linear operations allowed
 - Scaling, addition, multiplication.

RC Circuit Revisited



Poles and Zeros

$$\text{Given } F(s) = \frac{A(s)}{B(s)}$$

$$A(s) = a_n s^n + \dots + a_1 s + a_0$$

$$B(s) = b_m s^m + \dots + b_1 s + b_0$$

Poles are the values of s for which $B(s) = 0$

Zeros are the values of s for which $A(s) = 0$

Poles and zeros are complex

Order of system = # poles = m

Poles and Zeros

For example, $L[e^{-at}] = \frac{1}{s + a}$

Pole is $s = -a$

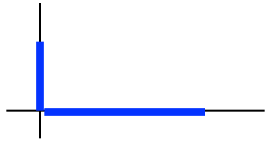

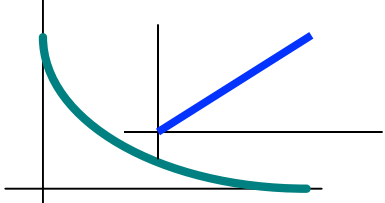
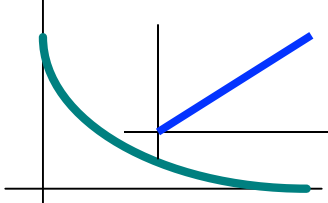
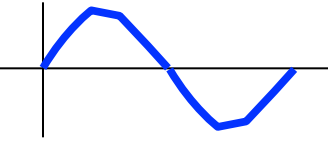
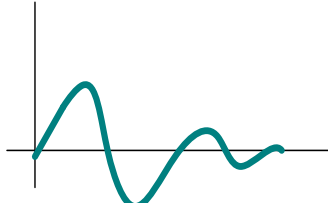
No Zeros

For sine : $L[\sin(\omega t)] = \frac{\omega}{s^2 + \omega^2}$

Poles are $s = -i\omega, i\omega$

No Zeros

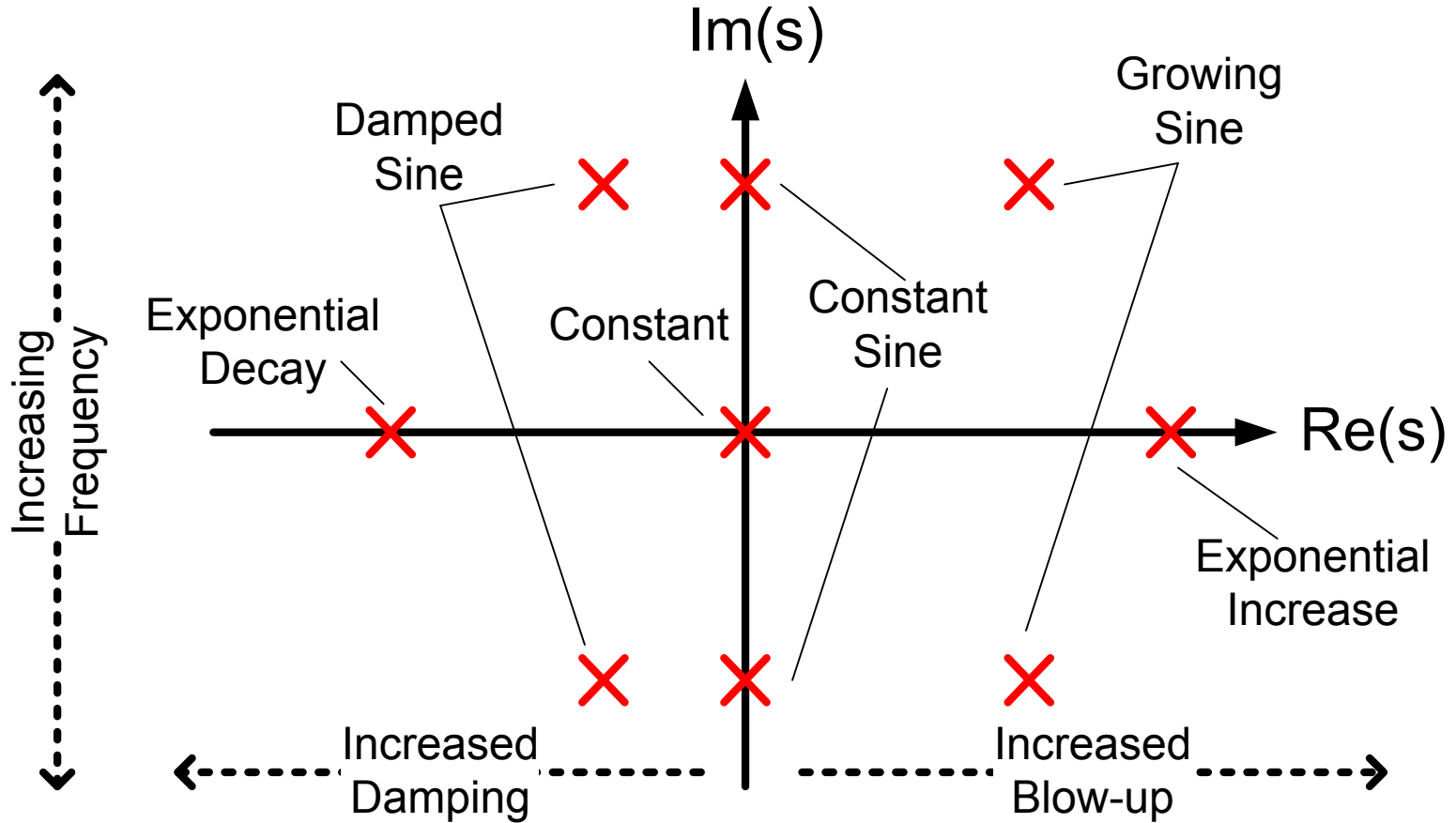
Poles and Zeros

Name	$f(t)$		$F(s)$	Poles
Impulse δ	$f(t) = \begin{cases} 1 & t = 0 \\ 0 & t > 0 \end{cases}$		1	n/a
Step	$f(t) = 1$		$\frac{1}{s}$	0
Ramp	$f(t) = t$		$\frac{1}{s^2}$	0 (double)
Exponential	$f(t) = e^{-at}$		$\frac{1}{s+a}$	-a
Sine	$f(t) = \sin(\omega t)$		$\frac{\omega}{s^2 + \omega^2}$	$-i\omega, i\omega$
Damped Sine	$f(t) = e^{-at} \sin(\omega t)$		$\frac{\omega}{(s+a)^2 + \omega^2}$	$-a-i\omega, -a+i\omega$

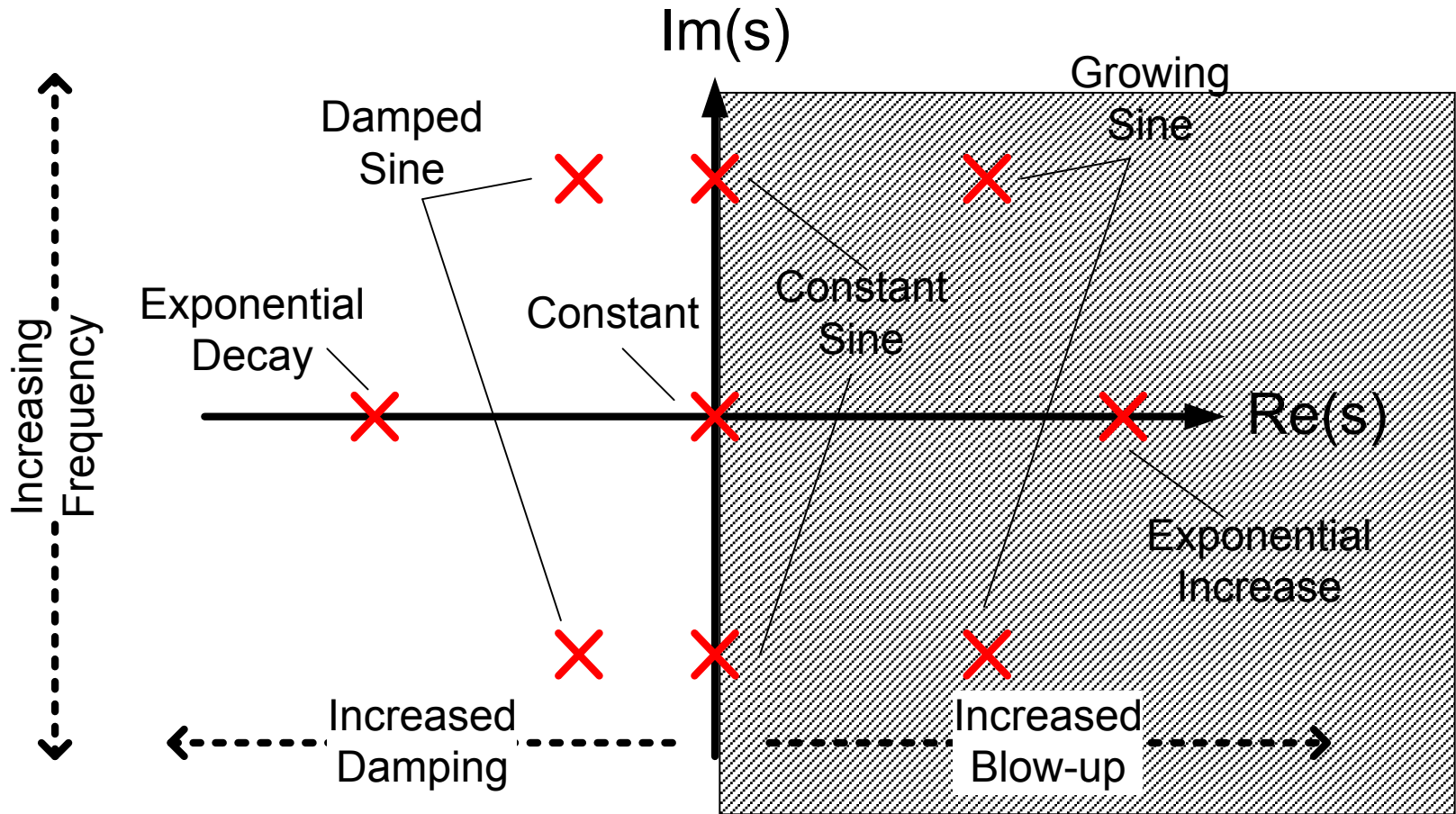
Poles and Zeros

- If pole has:
 - Real negative: exponential decay
 - Real positive: exponential growth
 - If imaginary $\neq 0$: oscillation of frequency ω

Effect of Poles Location

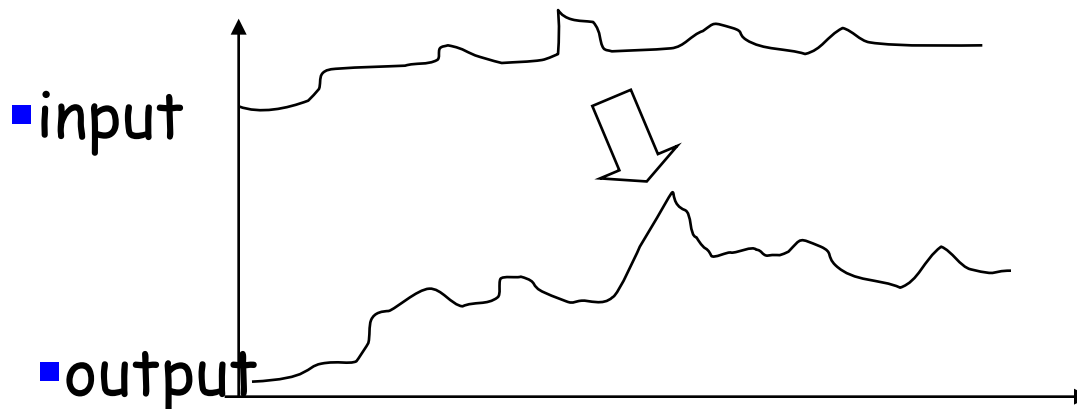
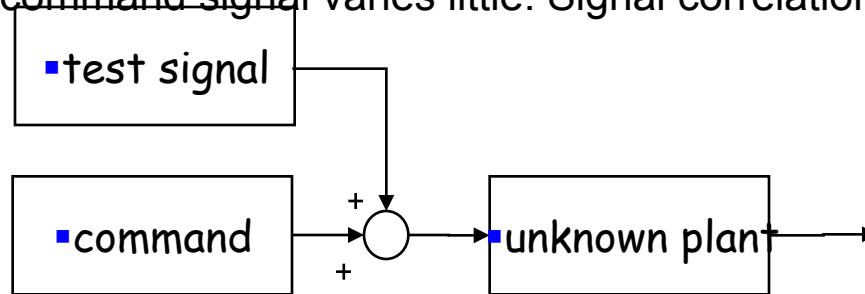


Where do you don't want to be?



Plant Identification

- Once the model is approximately known, the parameters must be determined by measurements.
- Classical methods are the response to a pulse at the input or to a calibrated noise at the input, in case the command signal varies little. Signal correlation then yields the parameters.

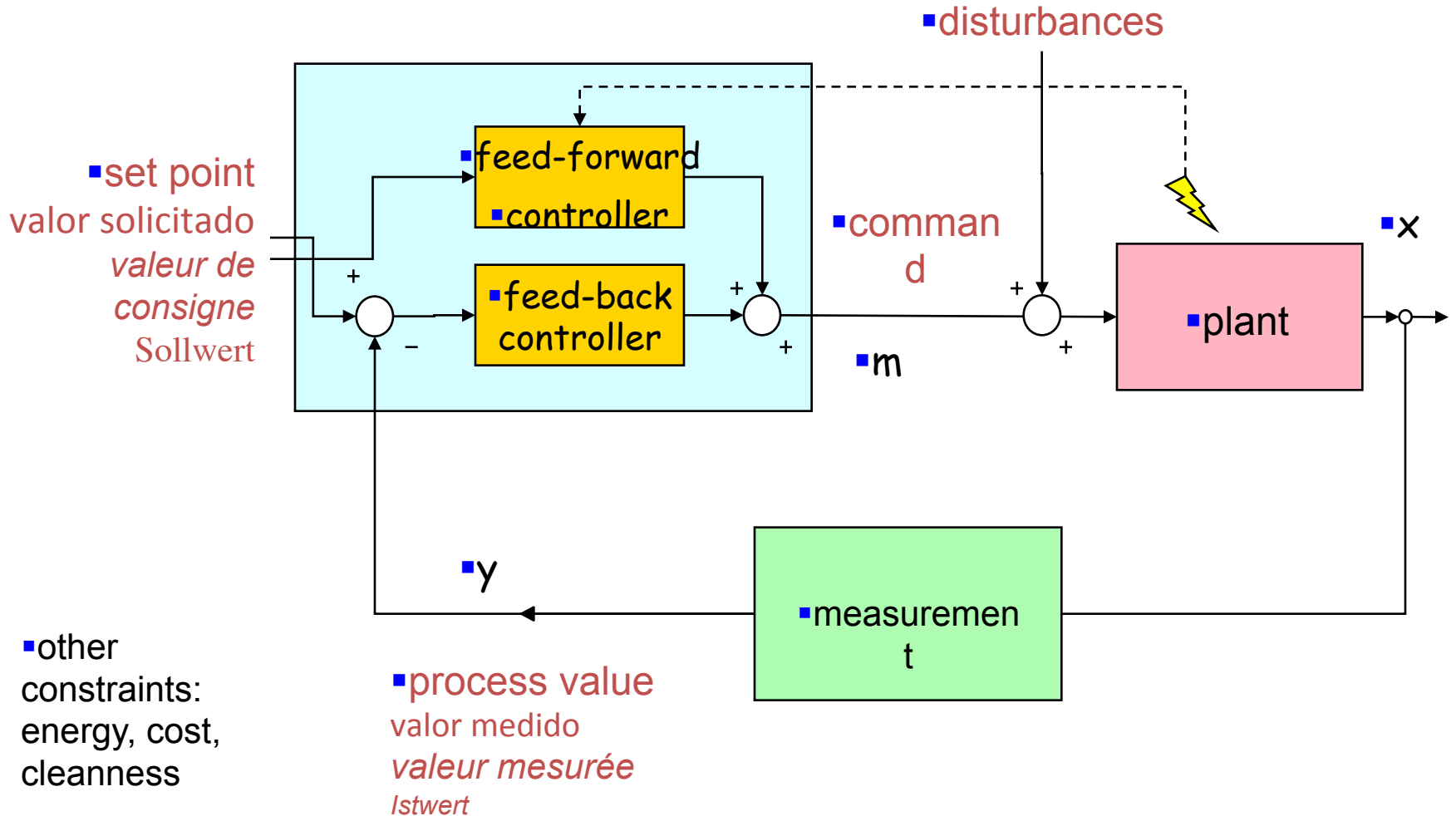


Controllers

- When a plant is known, a controller can be designed.
- In practice, the plant's parameters vary (e.g. number of passengers in a train), and the plant is subject to disturbances (wind, slope)
- The controller
 - needs to measure through sensors the state of the plant and if possible the disturbances.
 - follows certain quality laws to stabilize the output within useful time, not overshoot, minimize energy consumption, etc.....

Human body as a regulator example

- Consider a person taking a shower as a control system



Where is that controller located ?

- directly in the sensor
- or in the actuator
- (analog PIDs)



- high-end: in a set of possibly redundant controllers (here: turbine control)



- as a separate device (analog PIDs)
- (some times combined with a recorder)



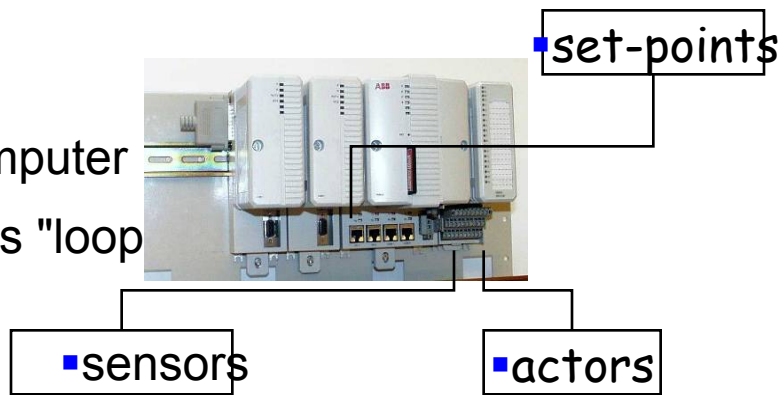
- as an algorithm in a computer
- (that can handle numerous "loop



set-points

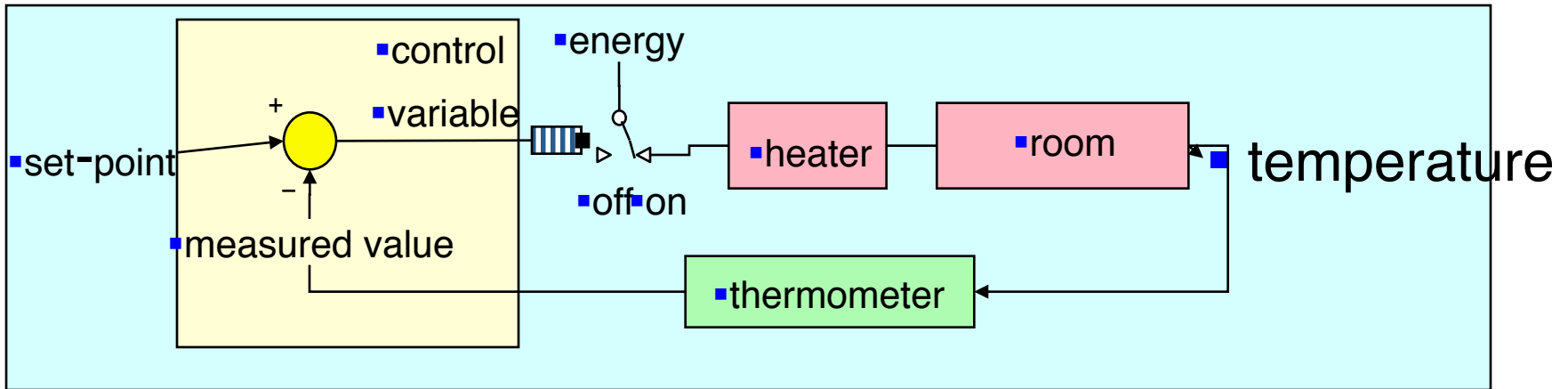
sensors

actors



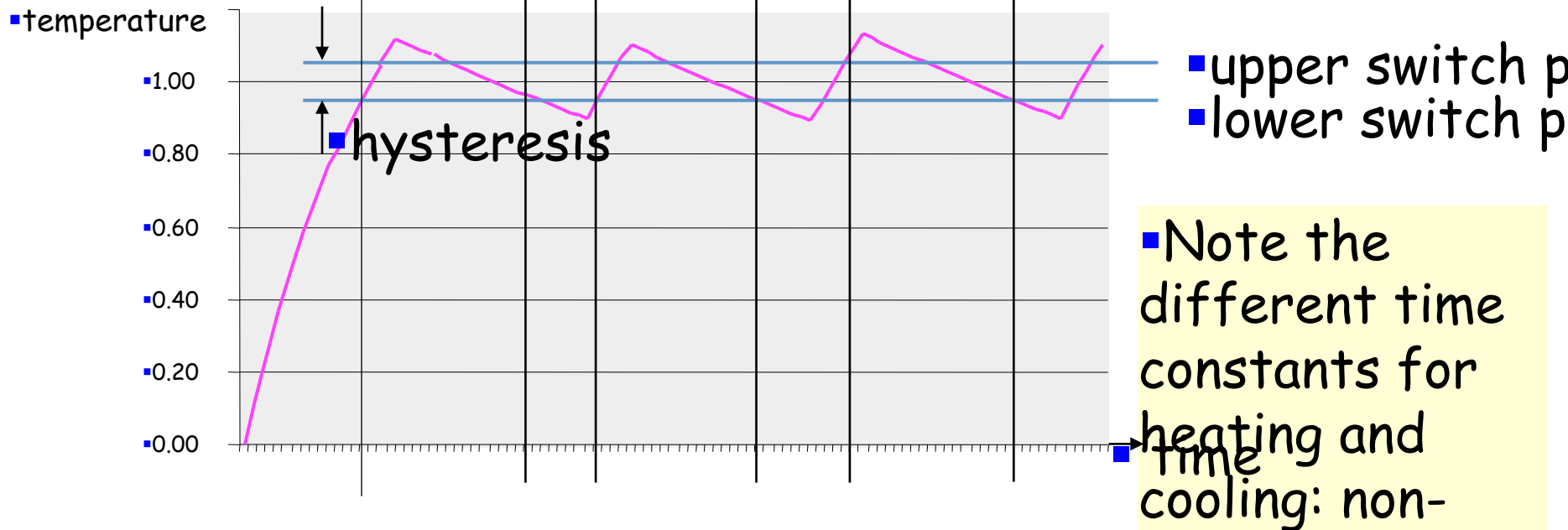
Two-point controller: principle

- The two-point controller (or bang-bang controller, regulator, has a binary output: on or off (example: air conditioning)



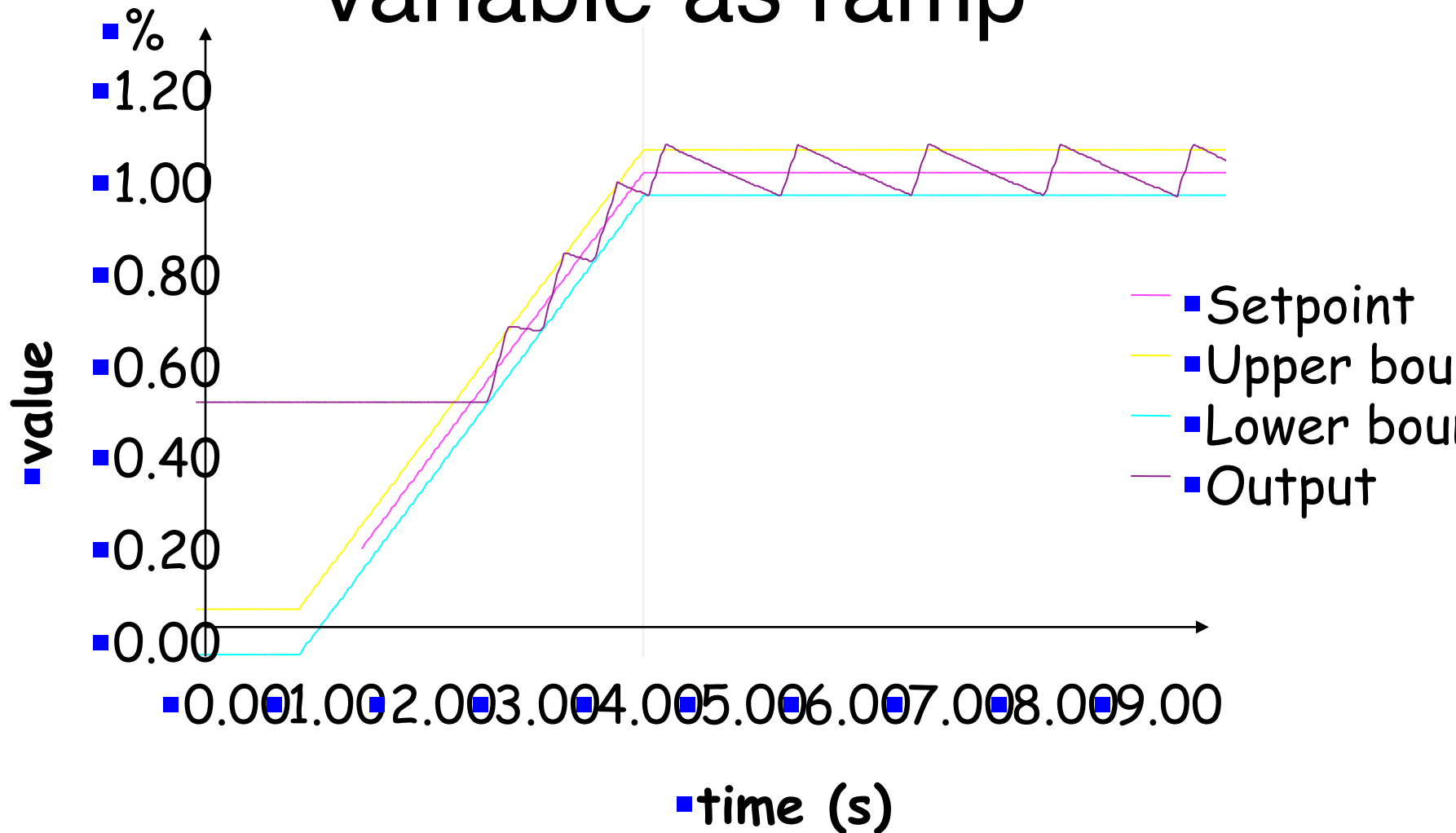
- commercial controller with integrated
- thermometer

Two-point controller: Hysteresis / Deadband



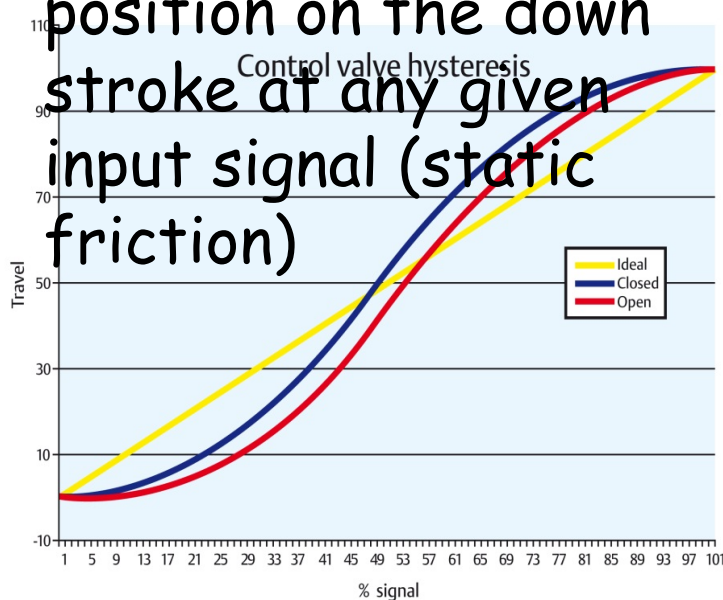
- If the process is not slow enough, hysteresis or switching period should be included
- to limit switching frequency and avoid wearing off the contactor.
- (thermal processes are normally so inertial that no hysteresis is needed)

Two-point controller: Input variable as ramp

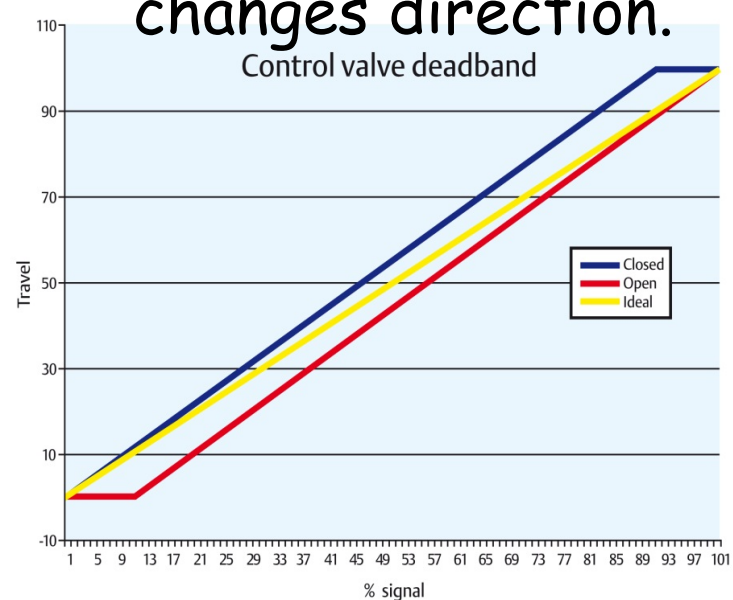


Hysteresis and Deadband of a Valve

■ Hysteresis: difference between the valve position on the upstroke and its position on the down stroke at any given input signal (static friction)

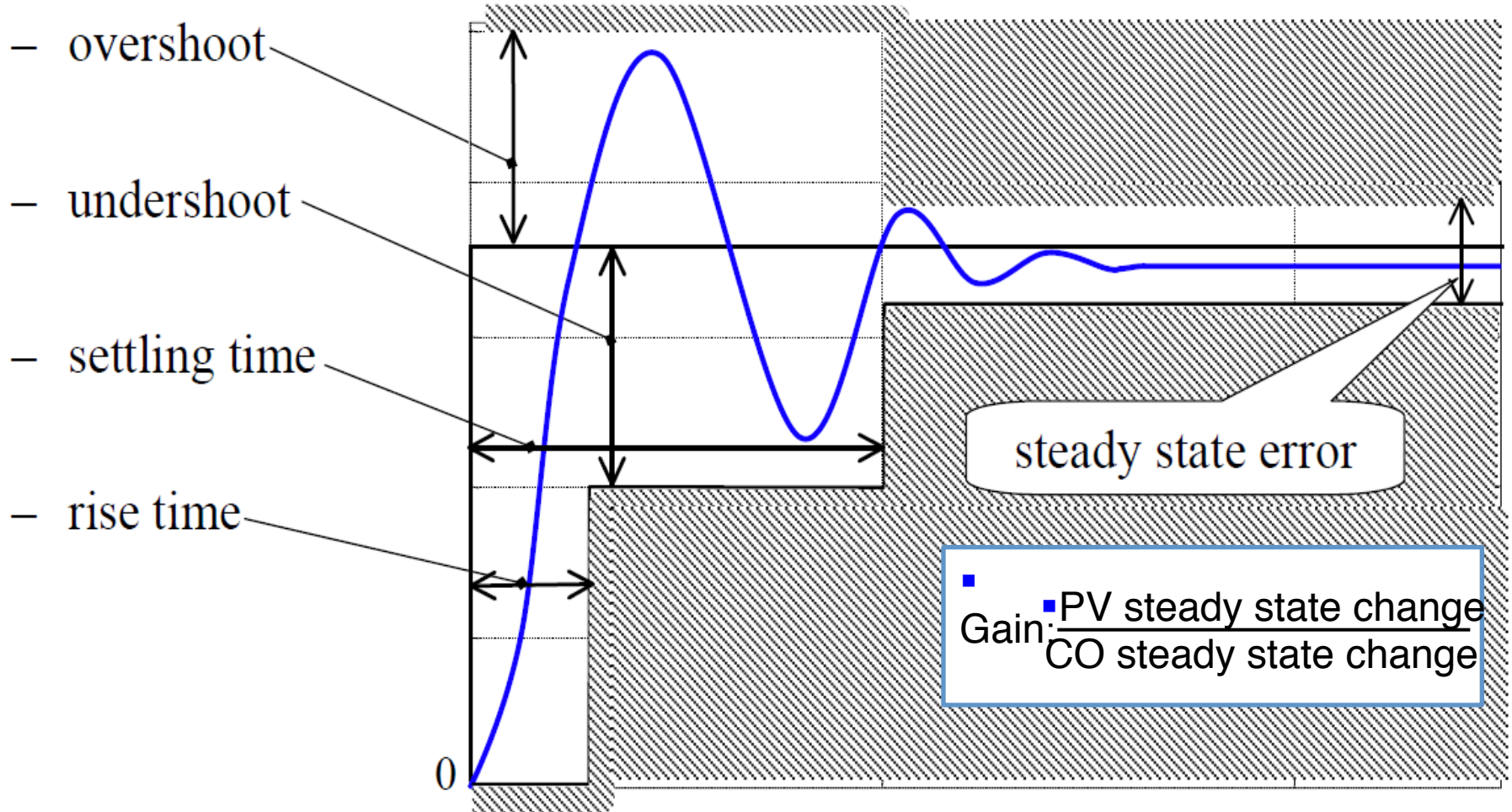


■ Deadband: no movement, generally occurs when the valve changes direction.

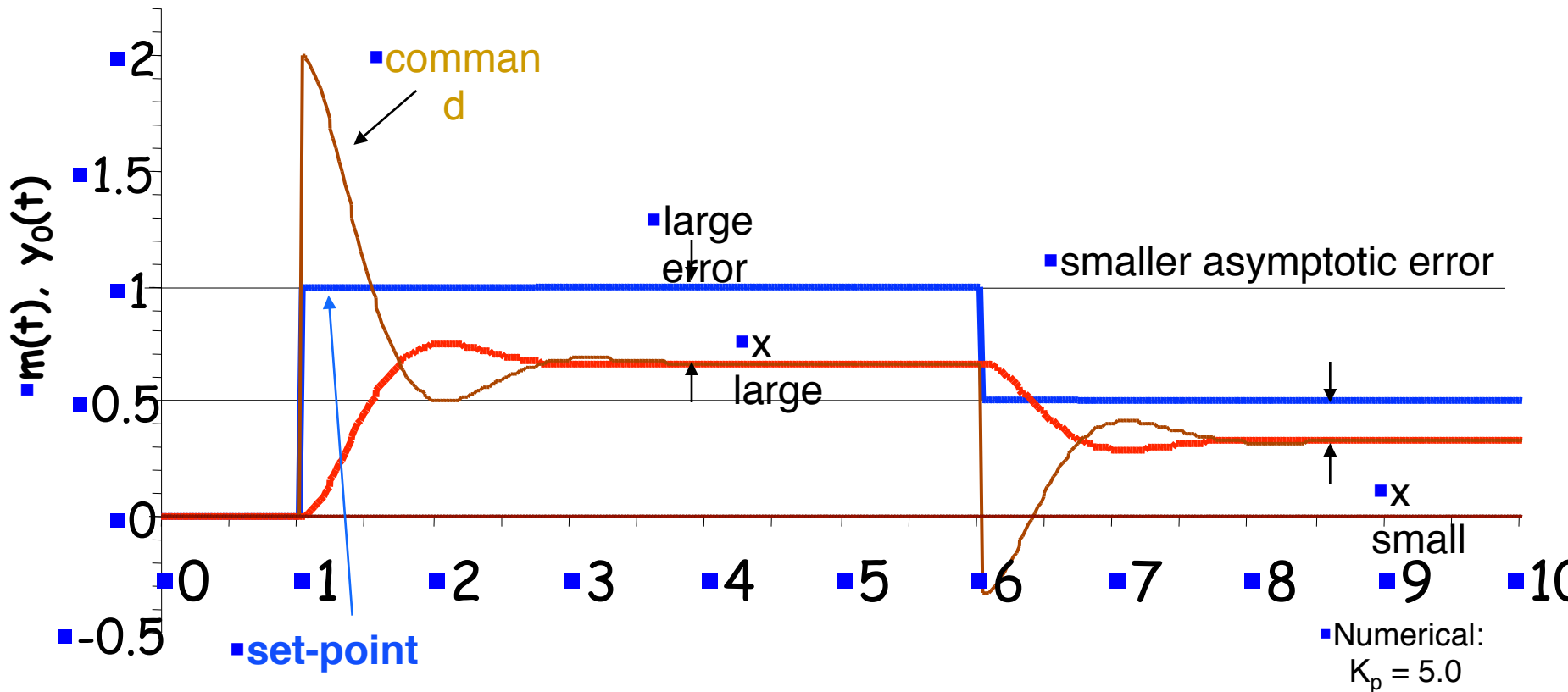


■ Source: <http://www.processindustryforum.com/solutions/valve-terminology-basic-understanding-of-key-concepts>

Step response



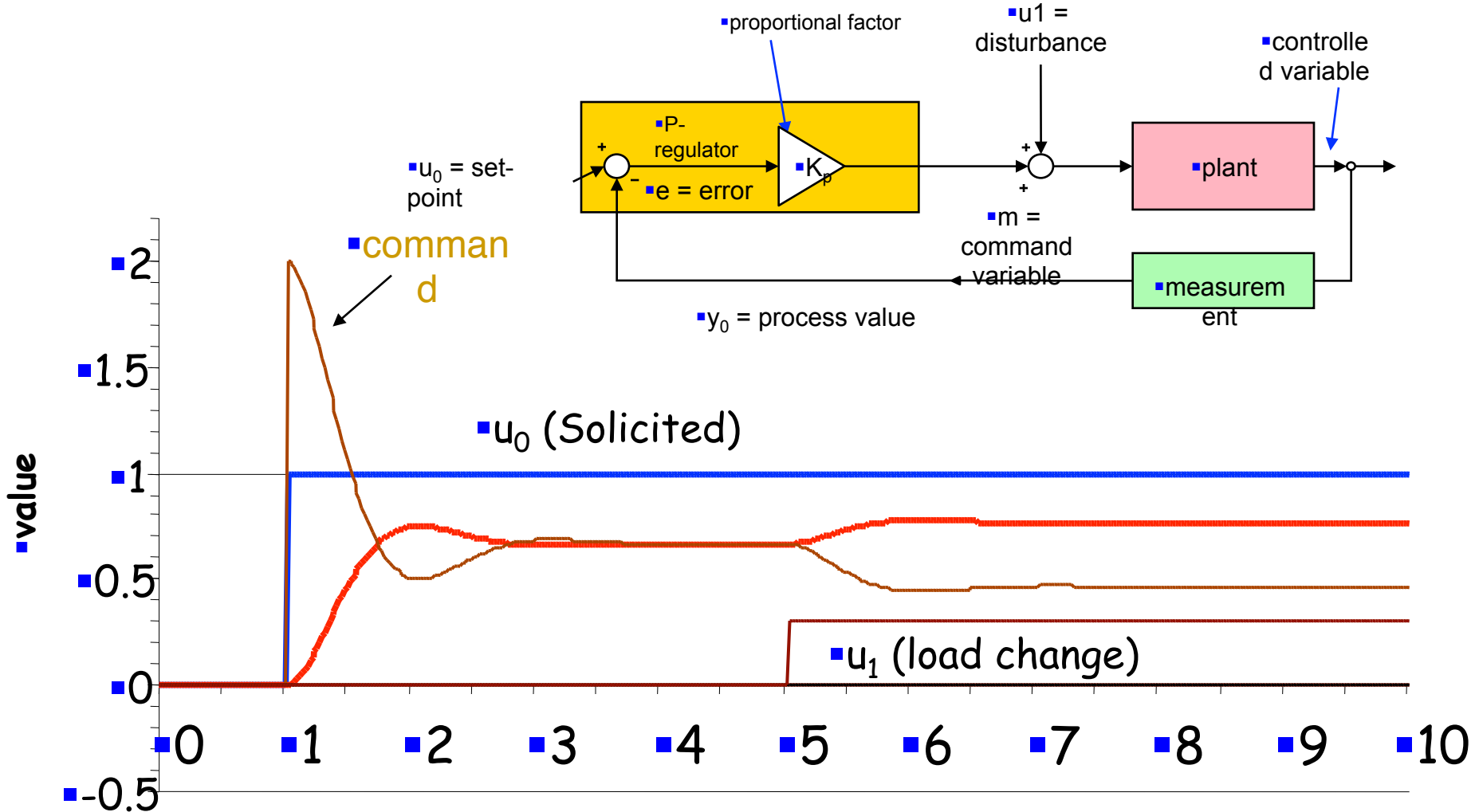
P-Controller: Step response



▪ The larger the set-point, the greater the error.

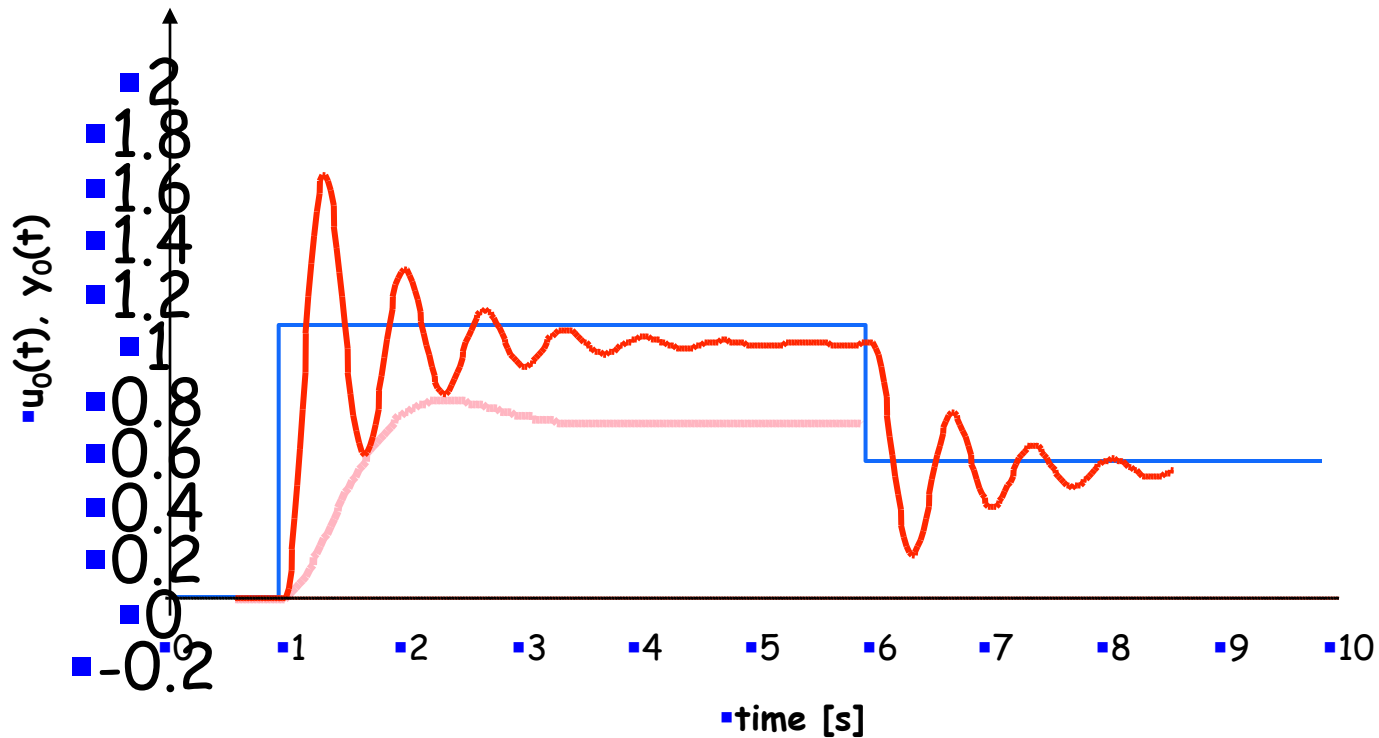
(The operator used to "reset" the control by increasing or decreasing the set-point)

P-Controller: Effect of Load change



- Not only a set-point change, but a load change causes the error to increase or decrease
- (A load change, modeled by disturbance u_1 , is equivalent to a set-point change)

P-Controller: Increasing the proportional factor



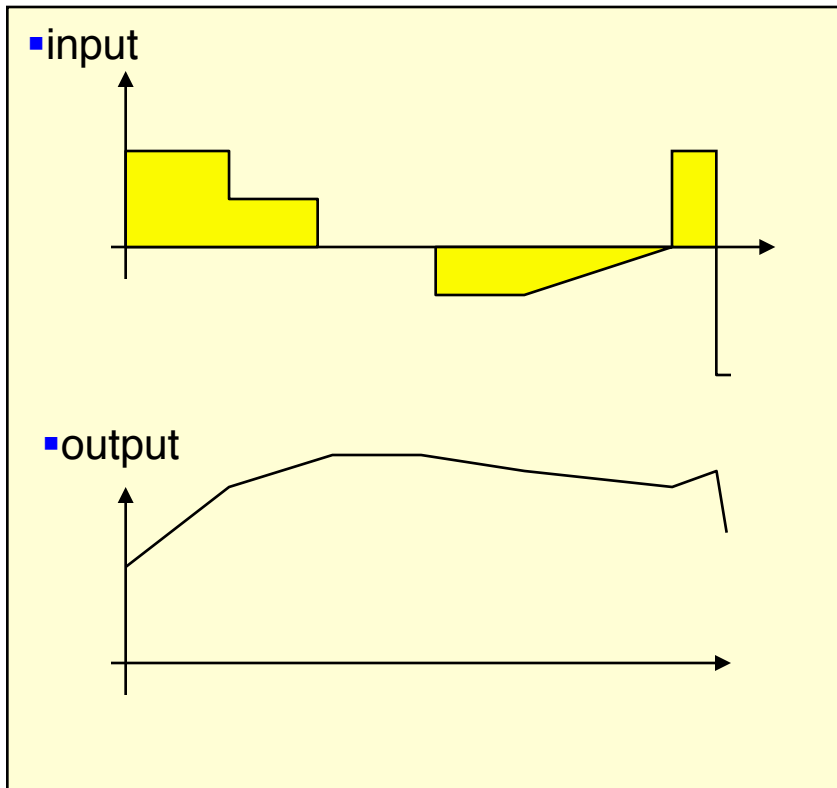
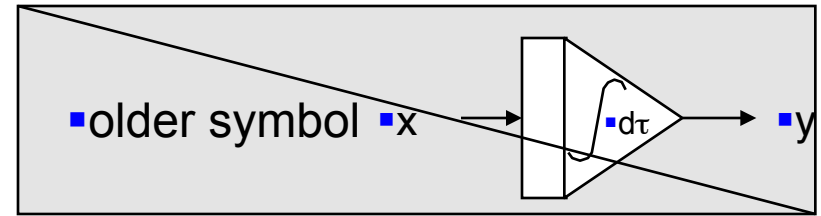
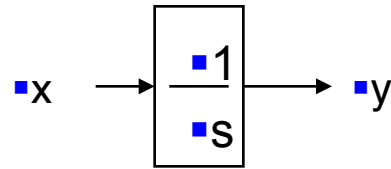
- increasing the proportional factor reduces the error, but the system tends to oscillate

PI-Controller (Proportional Integral): introducing the integrator

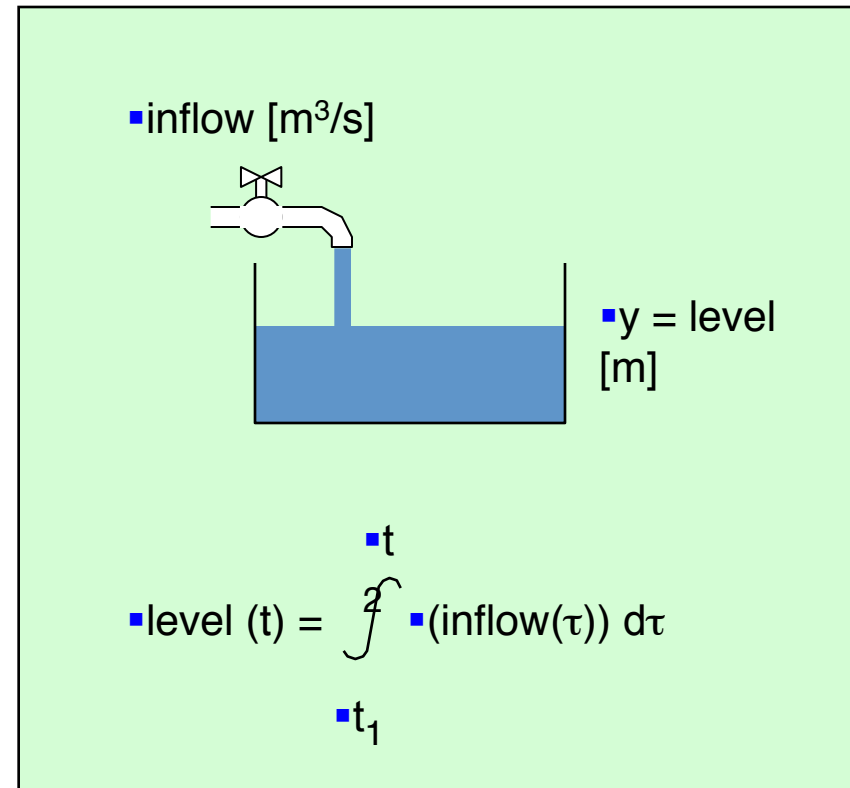
equation

$$y = \int_{t_0}^t x(\tau) d\tau$$

symbol



Time response of an integrator



Example of an integration process

PI – Regulator : Equations

- Time domain

$$m = K_p \left(e(t) + \frac{1}{T_i} \int_{t_0}^t e(\tau) d\tau \right)$$

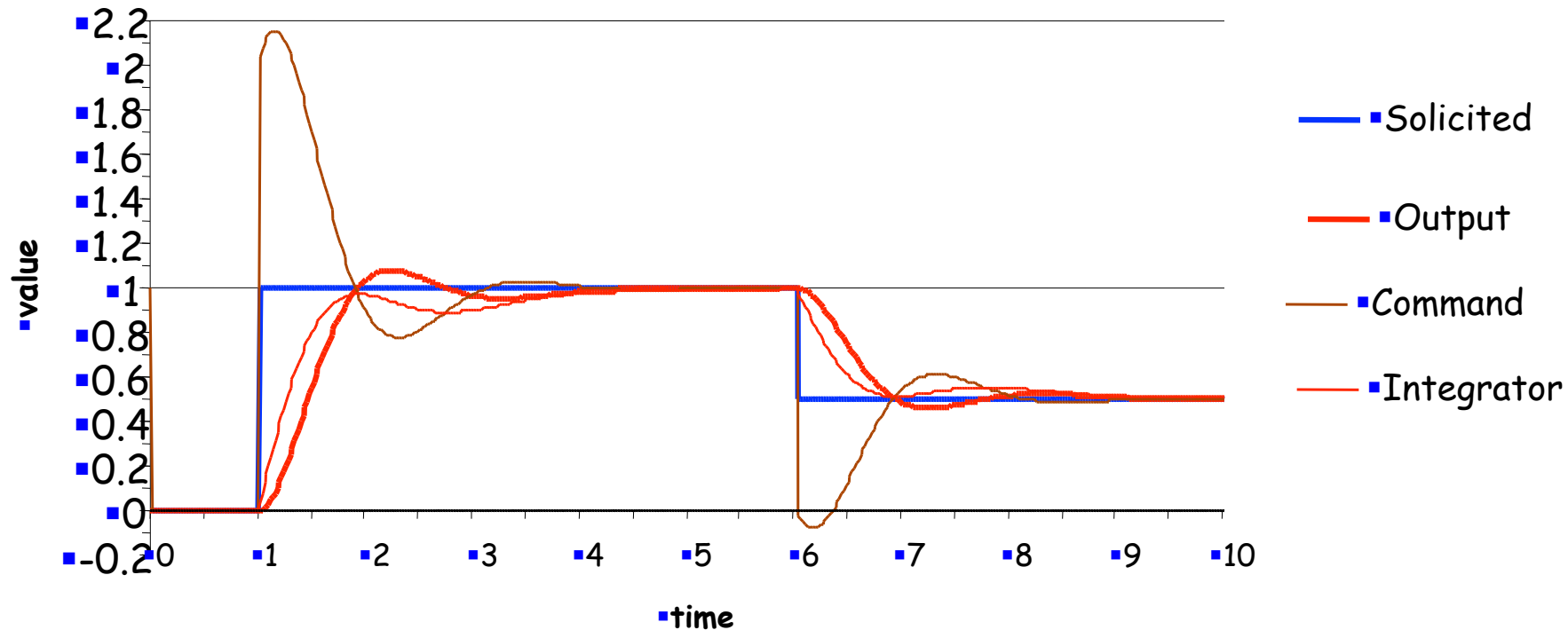
- Laplace domain

$$\tilde{m} = K_p \left(1 + \frac{1}{sT_i} \right) \tilde{e}$$

- T_i = reset time, tiempo de integración, *temps d'intégration*, Nachstellzeit

PI-Controller: response to set-point change

▪ $K_p = 2,0$, $T_i = 1,0$ s



- The integral factor reduced the asymptotical error to zero, but slows down the response
- (if K_p is increased to make it faster, the system becomes unstable)

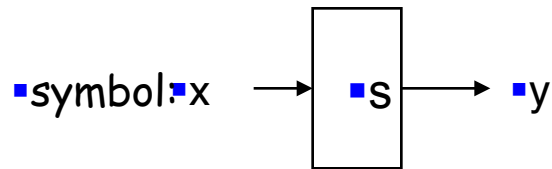
PD controller

- Basic idea of the PD regulator: take into account not only the value of the error, but the rate at which the error changes.

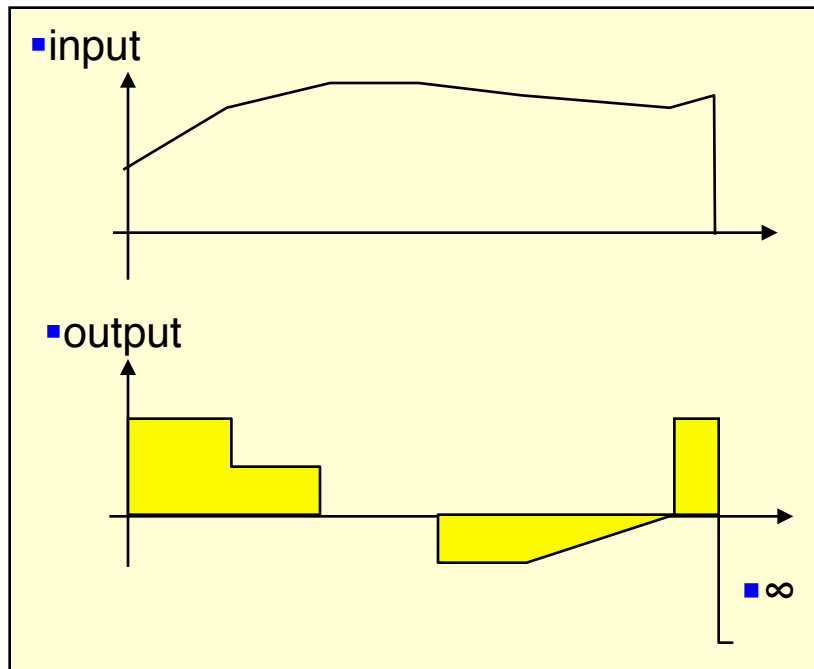
- Example: when parking a car in front of a wall, the driver not only looks at the distance to the wall, but also at the speed at which the car approaches the wall.

PD-Controller: Introducing the differentiator

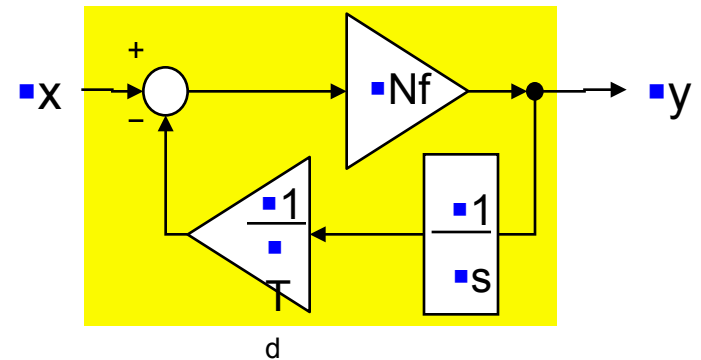
- equation: $\dot{y} = \frac{dx}{dt}$



- temporal response:

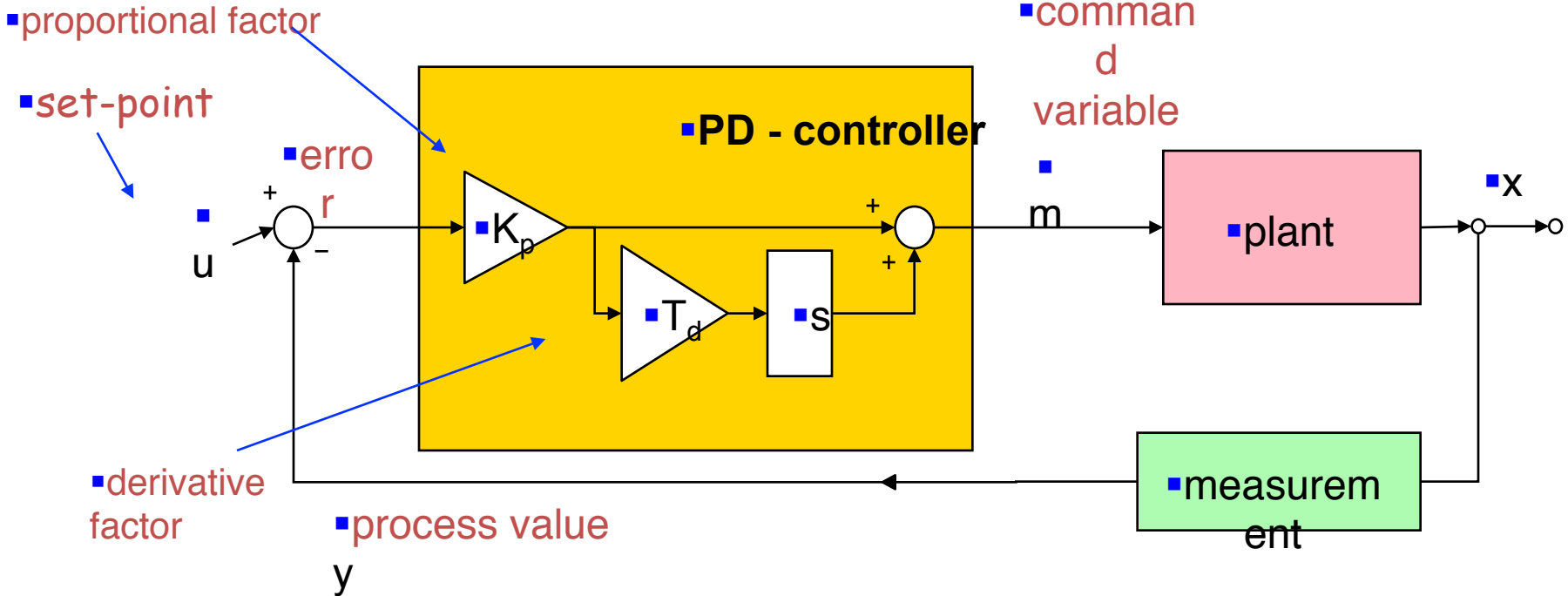


- A perfect differentiator does not exist.
- Differentiators increase noise.
- Differentiators are approximated by
- feed-back integrators (filtered differentiator):



- Instead of differentiating, one can use an already available variable:
- e.g. the speed for a position control

PD - controller



■ Adding the D-part allows to react vigorously to changes in set-point or perturbations.

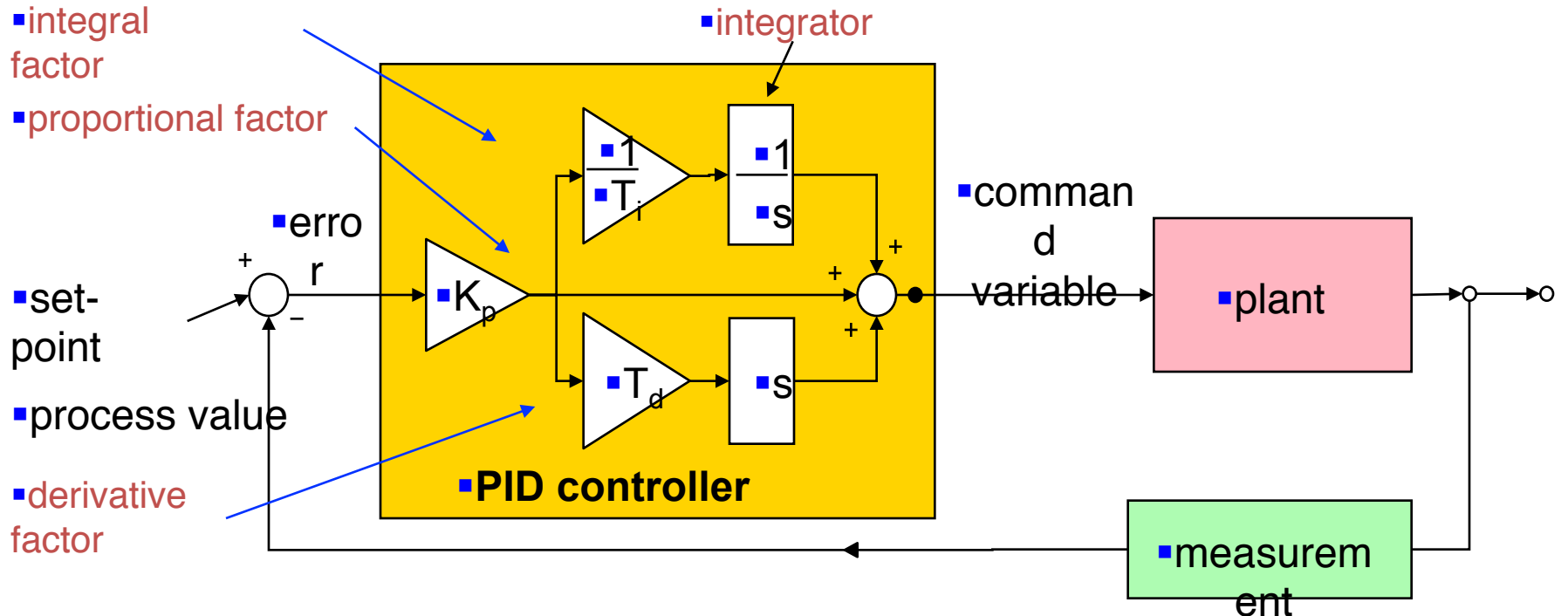
PD – Controller: Equations

- Time domain
$$m(t) = K_p \left(e(t) + T_d \frac{de(t)}{dt} \right)$$

- Laplace domain
$$\tilde{m}(s) = K_p (1 + T_d s) \tilde{e}(s)$$

- T_d = derivative time, *temps de dosage de dérivée*, Vorhalte

PID-Controller (Proportional-Integral-Differential)



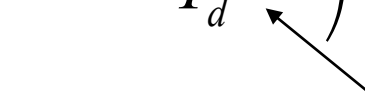
- The proportional factor K_p generates an output proportional to the error, it requires a non-zero error to produce the command variable.
- Increasing the amplification K_p decreases the error, but may lead to instability
- The integral time constant T_i produces a non-zero control variable even when the error is zero, but makes the system instable (or slower).
- The derivative time T_d speeds up response by reacting to an error change with a control variable proportional to the steepness of change.

PID controller: Equations

- time domain

$$m(t) = K_p \left(e(t) + \frac{1}{T_i} \int_{t_0}^t e(\tau) d\tau + T_d \frac{de(t)}{dt} \right)$$

- Laplace domain

$$\frac{\tilde{m}(s)}{\tilde{e}(s)} = K_p \left(1 + \frac{1}{sT_i} + \frac{T_d s}{\left(1 + \frac{N_f}{T_d} s\right)} \right)$$


- Real differentiators include this filtering

- K_p = proportional factor, gain, Reglerverstärkung,
- T_i = reset time, temps de dosage d'intégration (Nachhaltezeit, T_N)
- T_d = derivative time, *temps de dosage de dérivée*, Vorhaltezeit T_v

PID response summary

■ P_{large} ($K_p = 15$)
less error, but unstable

■ PI: no remaining error,
but sluggish response

■ (or instable, if K_p increased)

■ P_{small} ($K=5$) asymptotic error

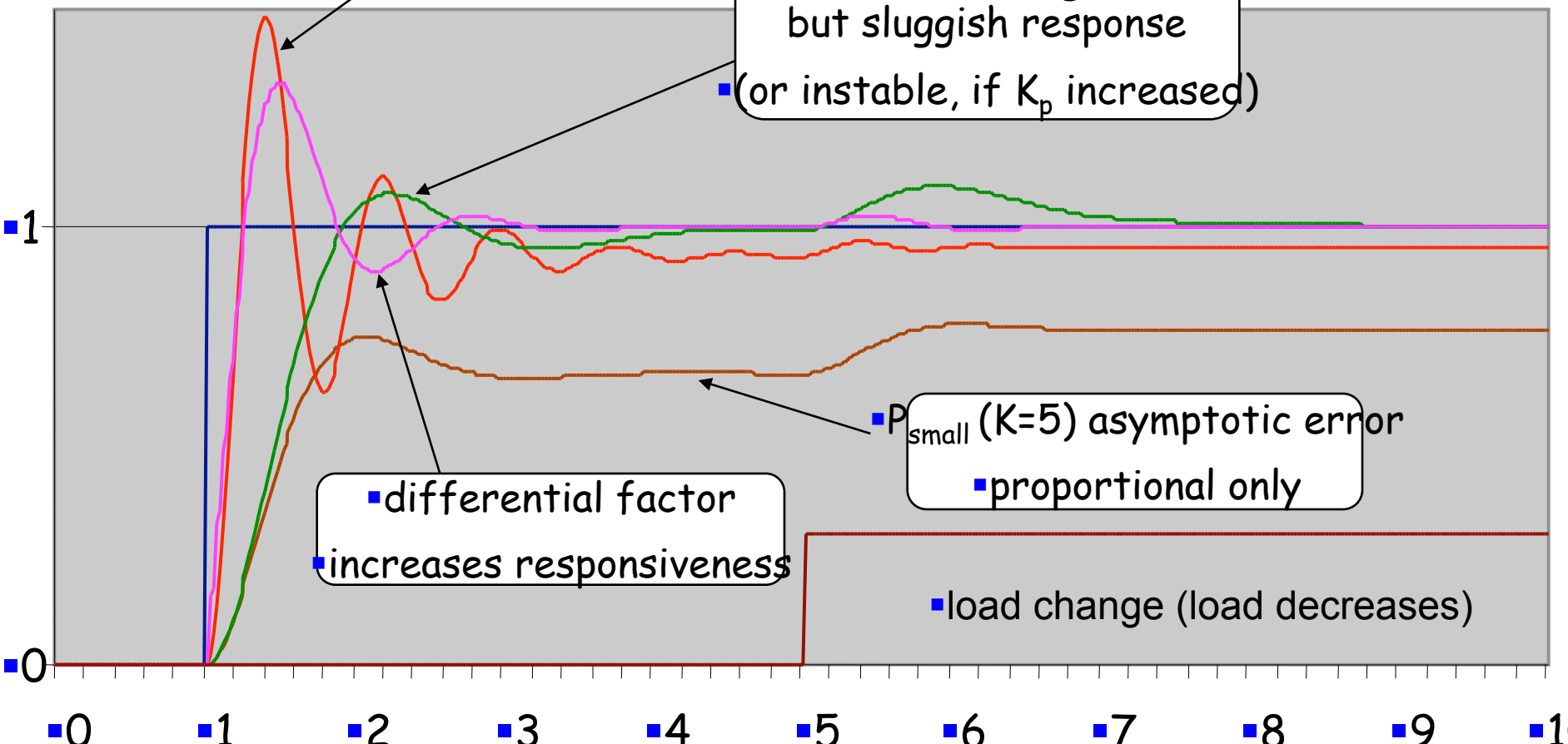
■ proportional only

■ differential factor

■ increases responsiveness

■ load change (load decreases)

■ Solicited ■ P_{small} ■ P_{large} ■ PI ■ PID ■ U_1



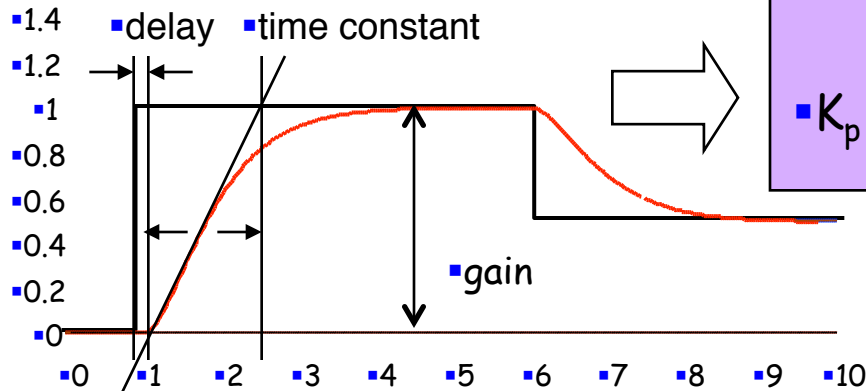
PID-Controller: empirical settings

Error	Rise time	Overshoot	Settling time	Steady-State
▪ increasing				
▪ K_p	Decrease	Increase	Small Change	Decrease
▪ T_i	Decrease	Increase	Increase	Eliminate
▪ T_d	Small Change	Decrease	Decrease	Small Change

▪ Empirical formula of Nichols (1942 !)

▪ See examples on http://en.wikipedia.org/wiki/PID_controller

▪ step response (open loop)



▪ $d \sim 0.2, T = 1.5s$

$$\begin{aligned}
 & \bullet K_p = \frac{1.2 T}{K_v} \quad \bullet T_i = 2.0 d \quad \bullet T_d = 0.5 d \quad \bullet (N_f = 10)
 \end{aligned}$$

Performance Specs

Steady-state error

- Steady state (tracking) error of a stable system

$$e_{ss} = \lim_{t \rightarrow \infty} e(t) = \lim_{t \rightarrow \infty} (r(t) - y(t))$$

- $r(t)$ is the reference input, $y(t)$ is the system output.
 - How accurately can a system achieve the desired state?
 - **Final value theorem**: if all poles of $sF(s)$ are in the open left-half of the s -plane, then

$$\lim_{t \rightarrow \infty} f(t) = \lim_{s \rightarrow 0} sF(s)$$

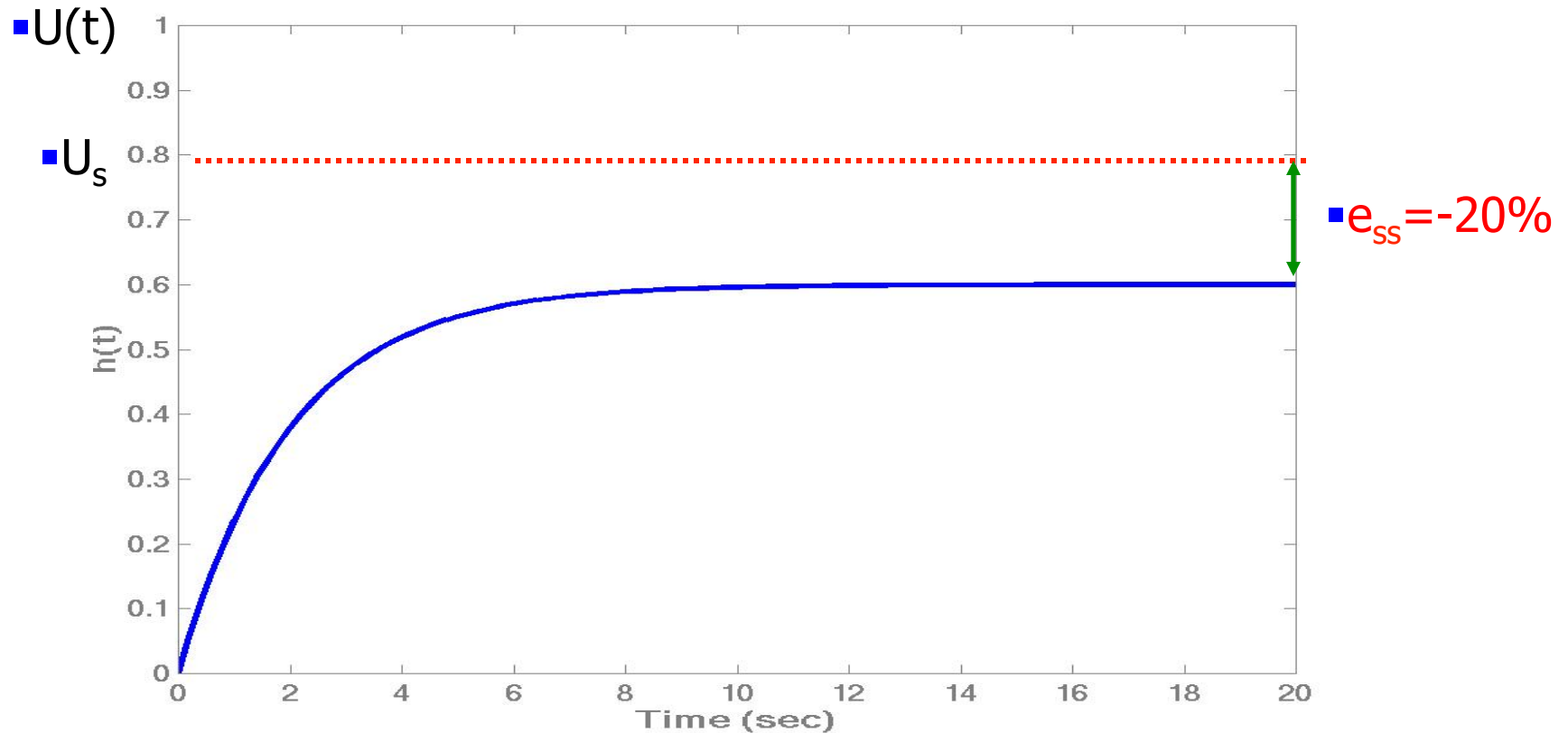
- Easy to evaluate system long term behavior without solving it

$$e_{ss} = \lim_{t \rightarrow \infty} e(t) = \lim_{s \rightarrow 0} sE(s)$$

Performance Specs

Steady-state error

- Steady state error of a CPU-utilization control system



Controller Design

PID control

- Proportional-Integral-Derivative (PID) Control

- Proportional Control $x(t) = Ke(t) \Leftrightarrow C(s) = K$

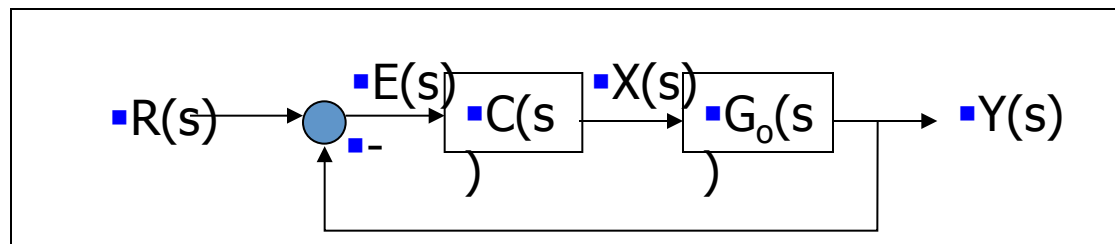
- Integral control $x(t) = KK_i \int_0^t e(\tau) d\tau \Leftrightarrow C(s) = \frac{KK_i}{s}$

- Integral control

- Derivative control $x(t) = KK_d \dot{e}(t) \Leftrightarrow C(s) = KK_d s$

- Derivative control

- Classical controllers with well-studied properties and tuning rules



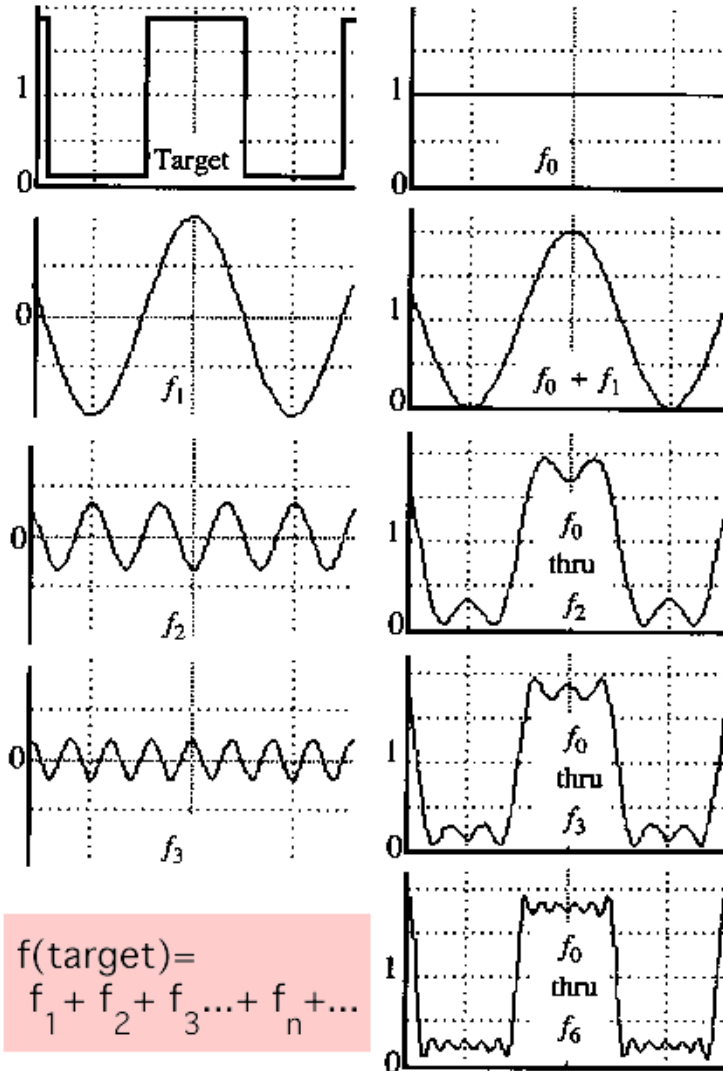
Fourier Transform

- Jean-Baptiste Fourier had crazy idea (1807):
 - *Any periodic function can be rewritten as a weighted sum of sines and cosines of different frequencies.*



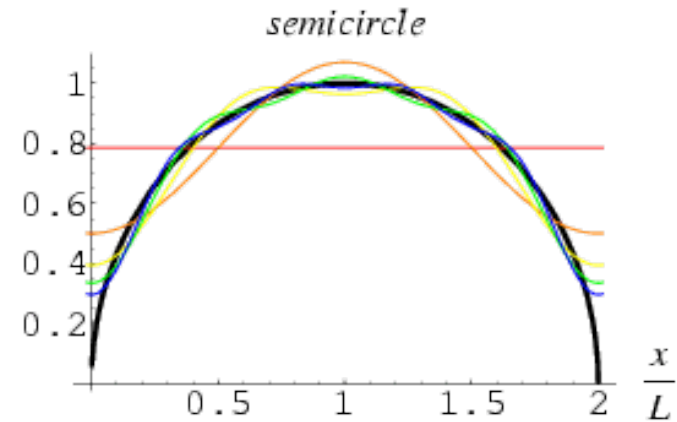
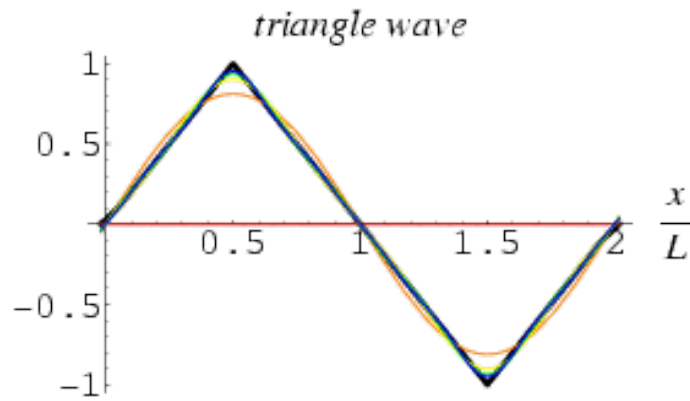
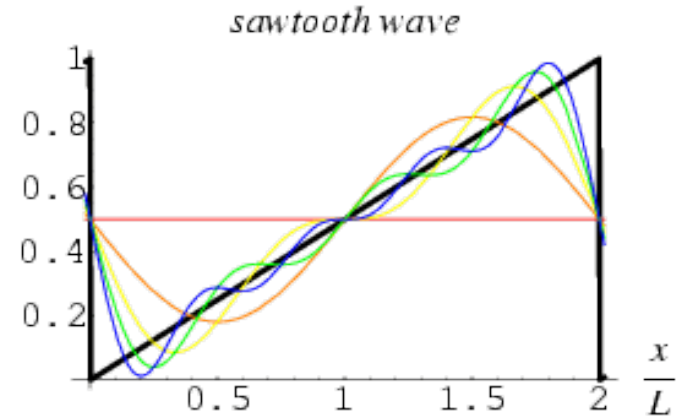
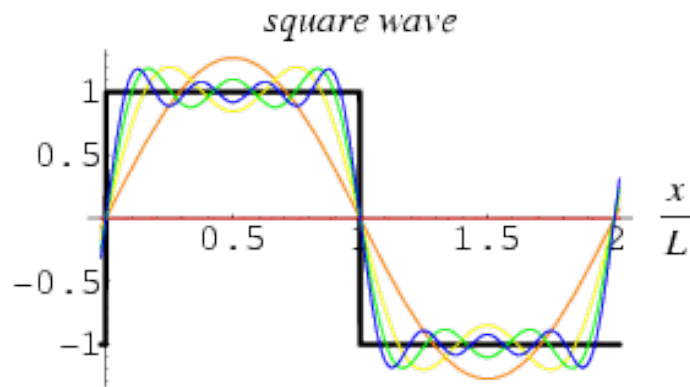
- Called Fourier Series

Square-Wave Deconstruction



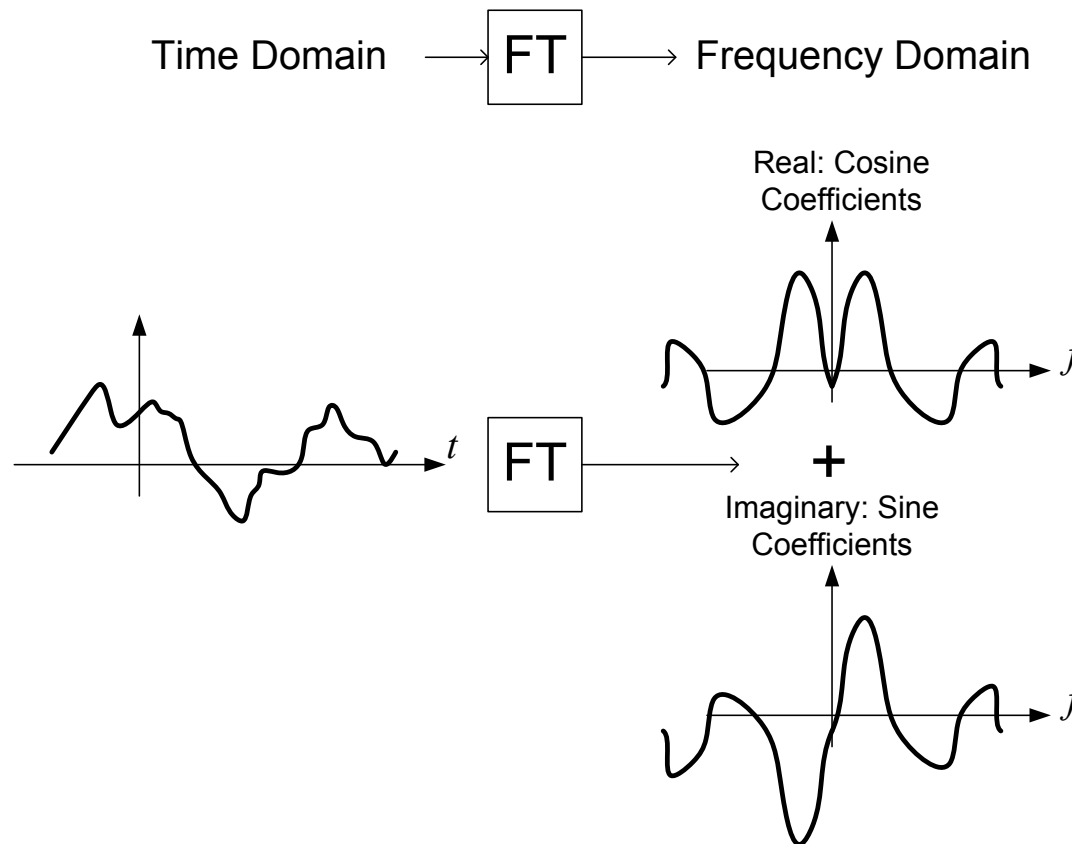
$$f(\text{target}) = f_0 + f_1 + f_2 + f_3 + \dots + f_n + \dots$$

Other examples



FT expands this idea

- Take **any** signal (periodic and non-periodic) in *time* domain and decompose it in sines + cosines to have a representation in the *frequency* domain.



FT: Formal Definition

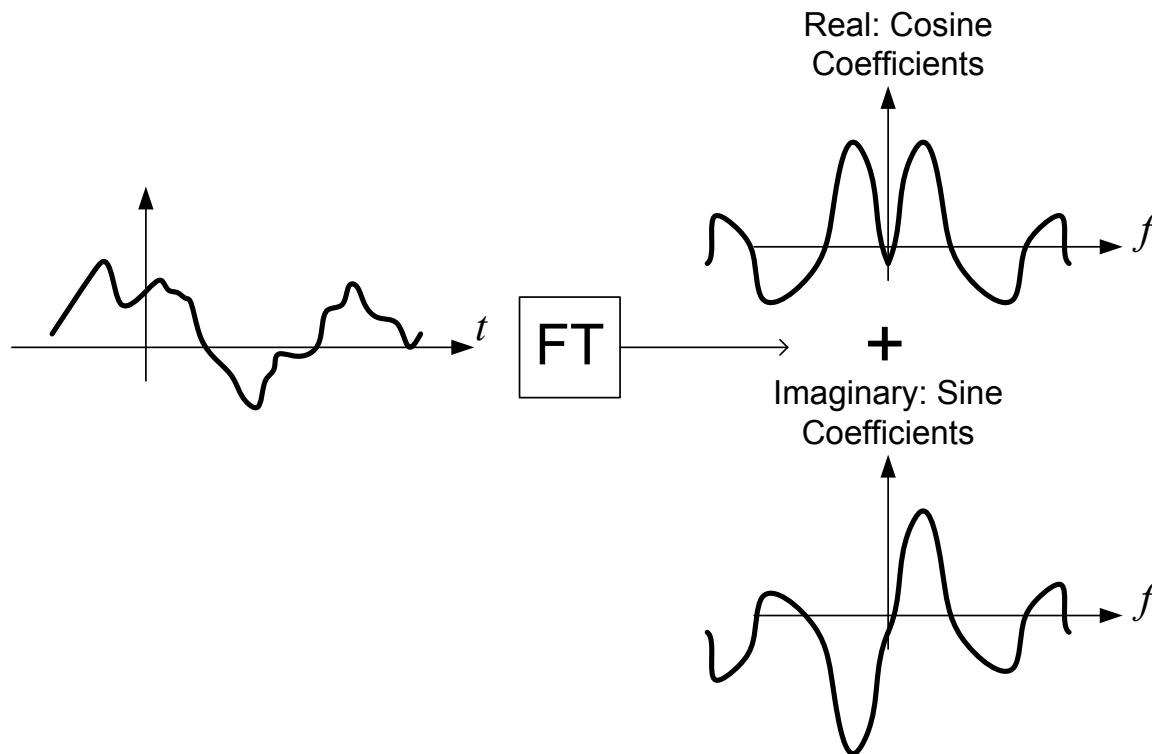
$$\text{Fourier Transform : } F(\omega) = \int_{-\infty}^{+\infty} f(x)e^{-i\omega x} dx$$

$$\text{Inverse Fourier Transform : } f(x) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} F(\omega)e^{i\omega x} d\omega$$

- Convention: Upper-case to describe transformed variables:
- Transform: $F\{x(t)\} = X(\omega)$ or $X(f)$ ($\omega=2\pi f$)
- Inverse: $F^{-1}\{Y(\omega)$ or $Y(f)\} = y(t)$

FT gives complex numbers

- You get complex numbers
 - Cosine coefficients are real
 - Sine coefficients are imaginary



Complex plane

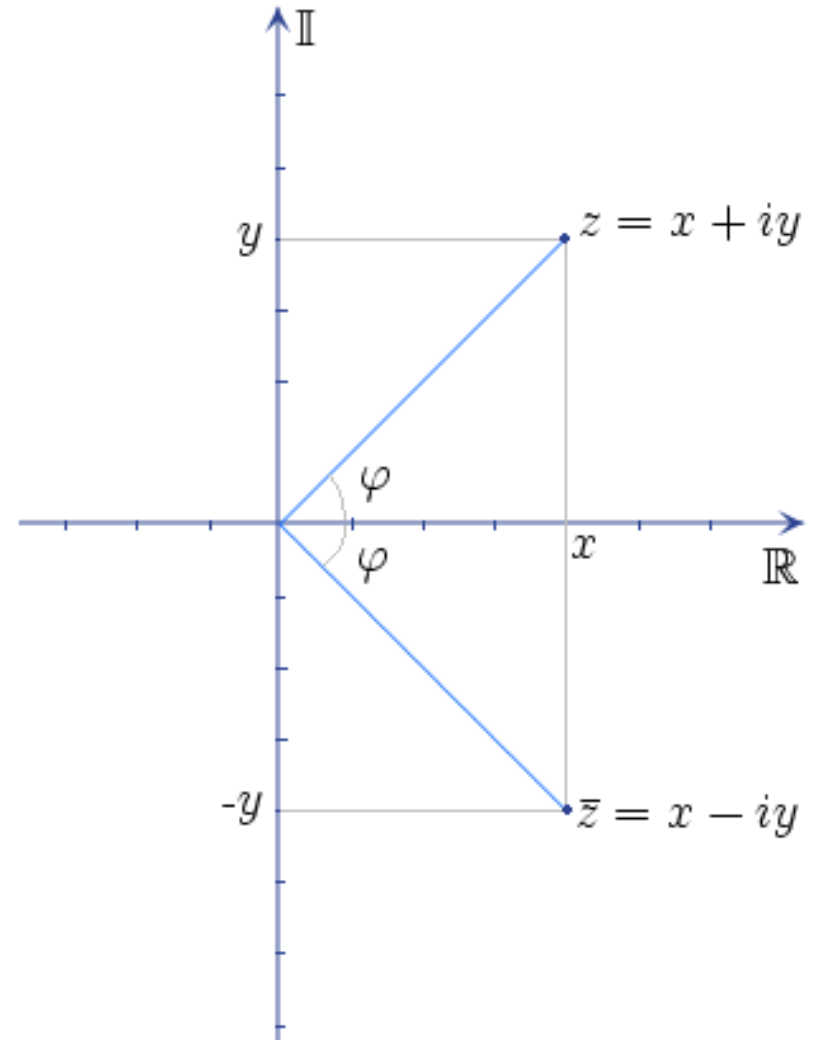
- Complex number can be represented:

- Combination of real + imaginary value:

$$x + iy$$

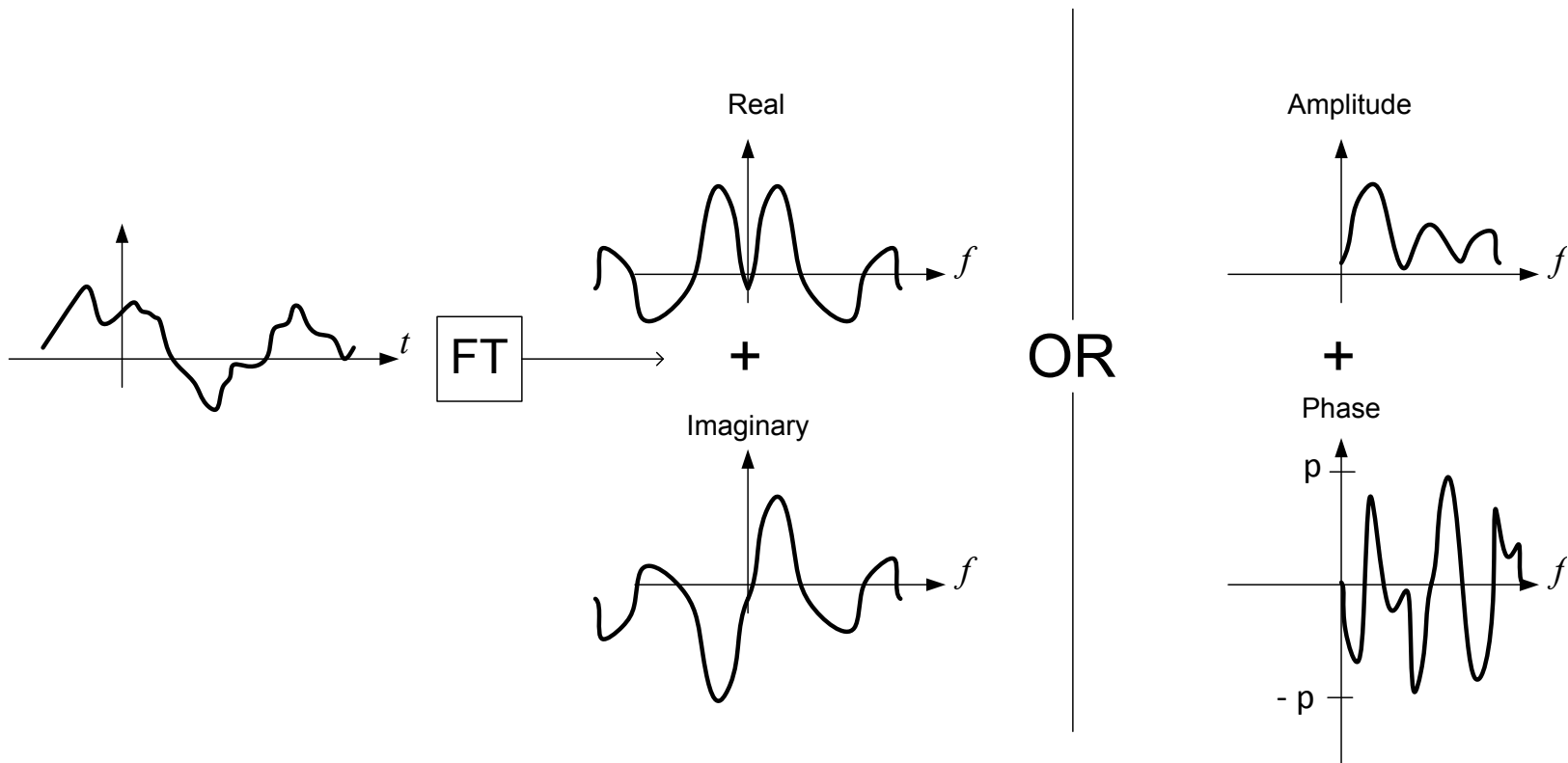
- Amplitude + Phase

$$A \text{ and } \varphi$$



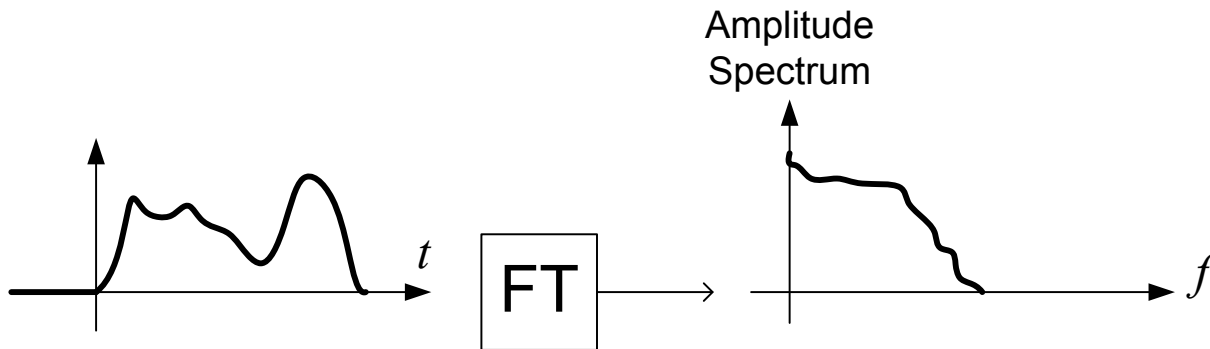
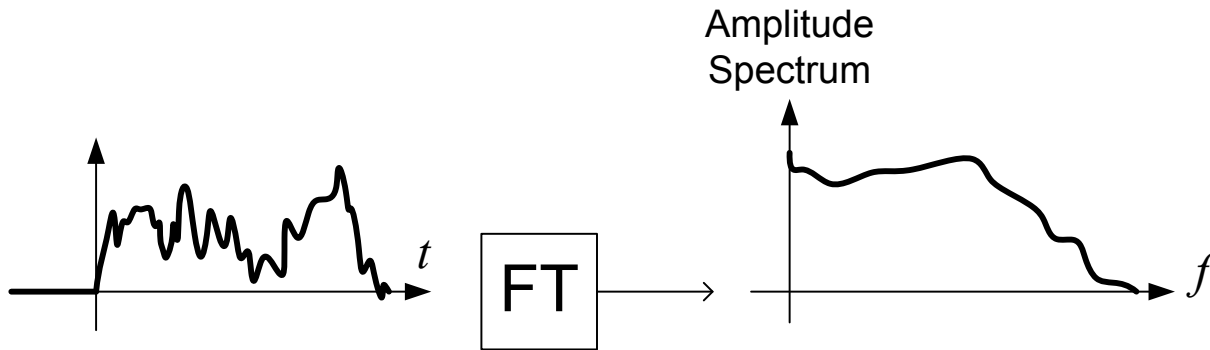
Alternative representation of FT

- Complex numbers can be represented also as **amplitude + phase**.



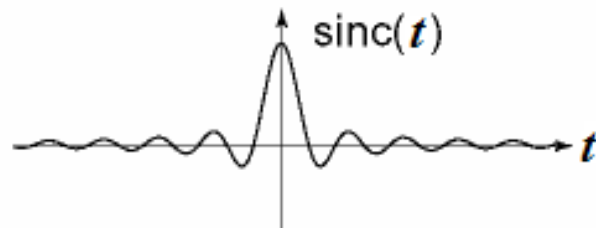
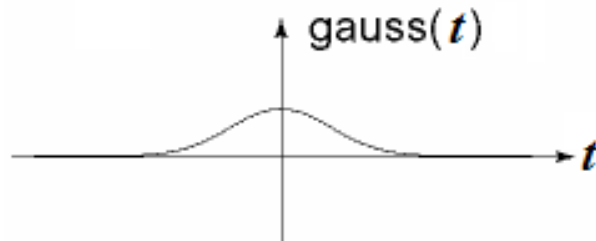
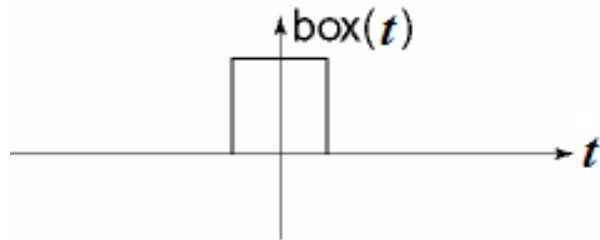
Example Fourier Transform

- Fast moving vs slow moving signals

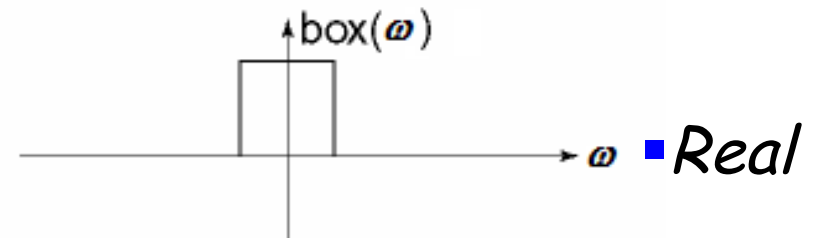
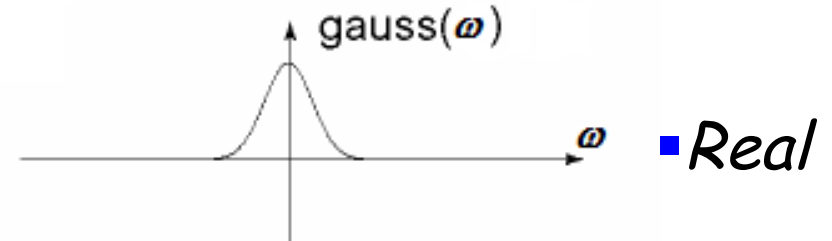
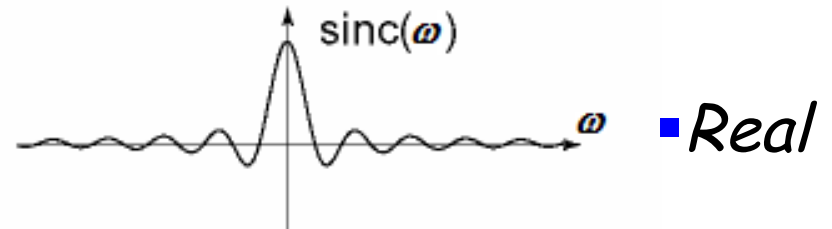


Example Fourier Transform

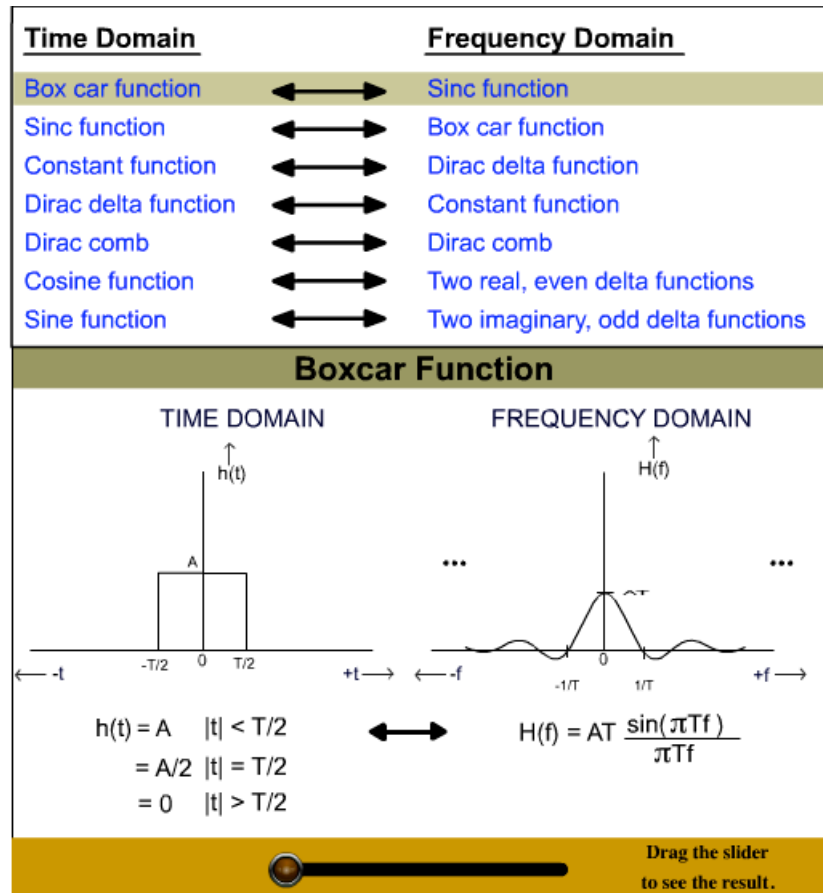
- Time Domain t



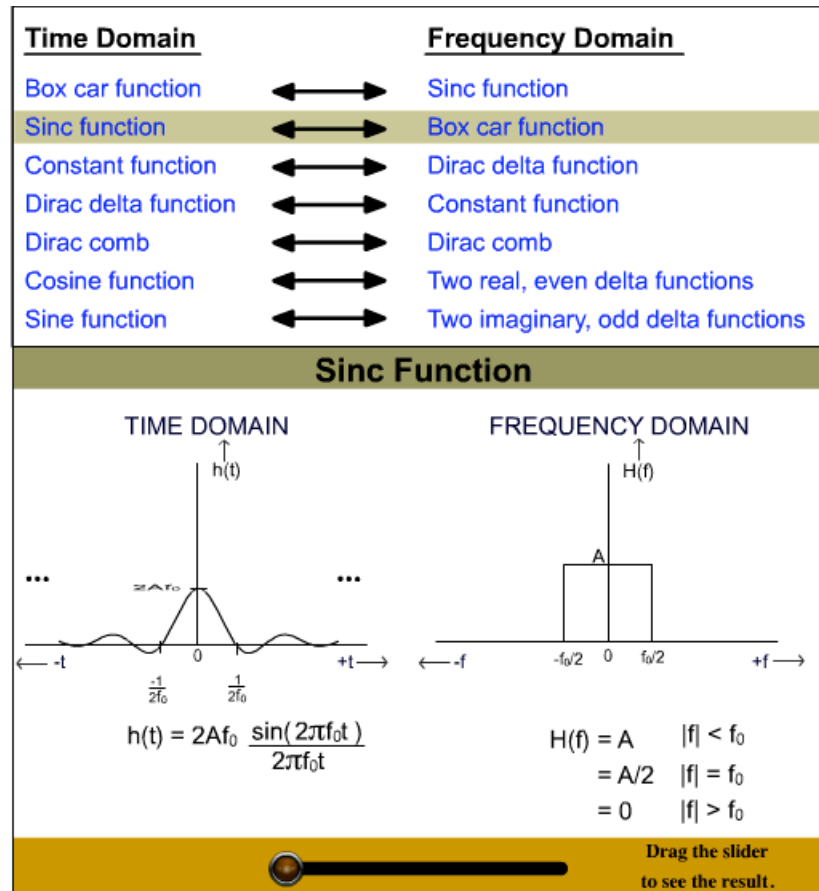
- Frequency Domain ω



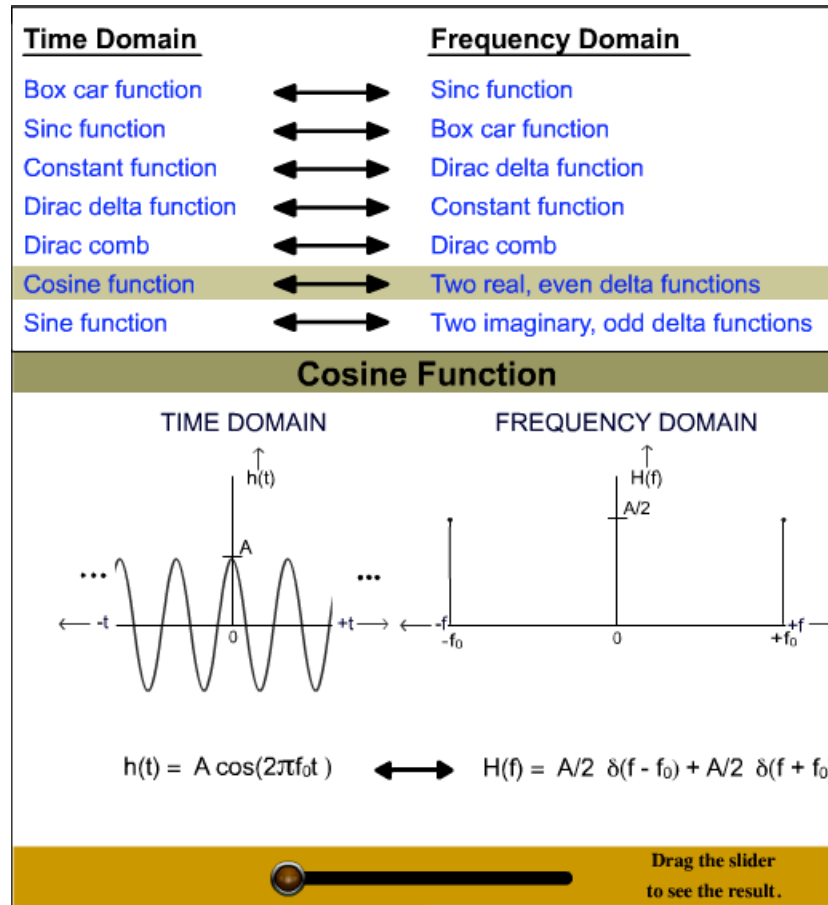
Example Fourier Transform



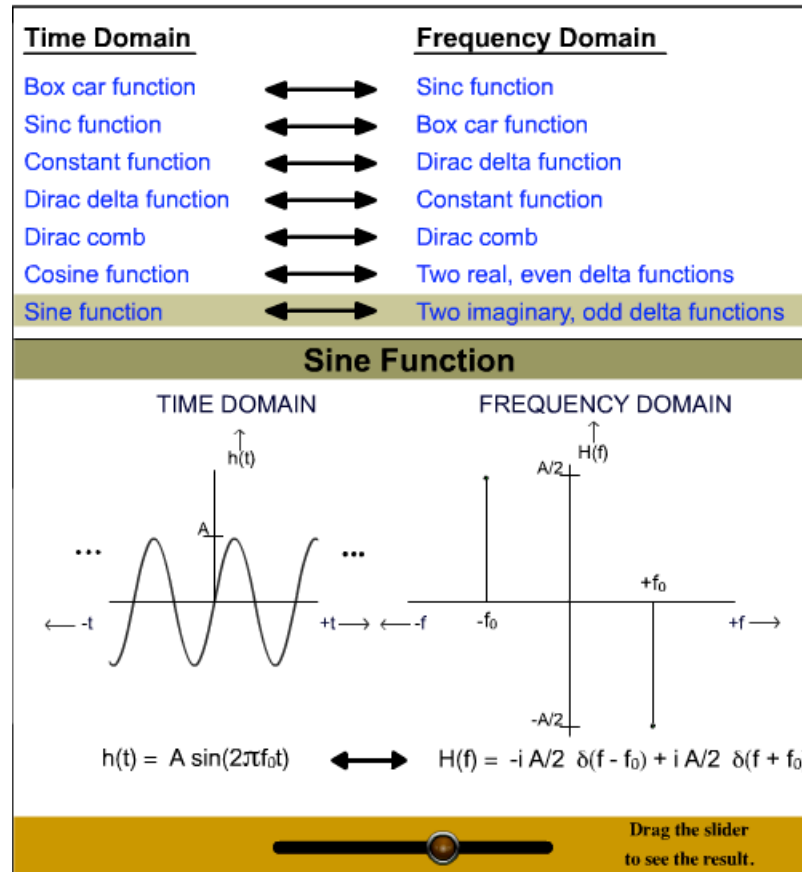
Example Fourier Transform



Example Fourier Transform



Example Fourier Transform



- Note: FT is imaginary for sine

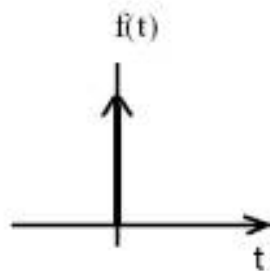
Example Fourier Transform

■ Time Domain t

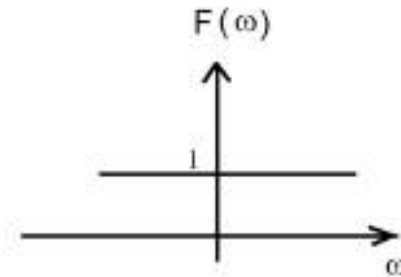
Frequency Domain

ω

6. $\delta(t)$

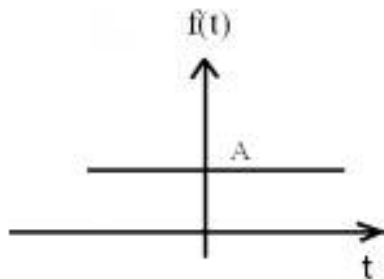


1

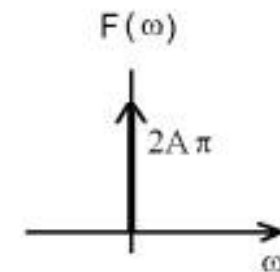


■ *Real*

7. A



$2A\pi\delta(\omega)$



■ *Real*

■ « DC component »

Problems

The M25 (London's "orbital carpark") has traffic problems – here are three things you might do to alleviate them

1. Design a controller that turns on-ramp traffic lights to red when the mean speed of cars (as measured by a sensor network) goes below 60kph, and green when it goes above – explain the behaviour of traffic that results (think admission control!).
2. Design a controller for each car that sets the speed the same as the car in front (as measured by radar).
3. Design a cruise control that keeps the speed of the car at a set point, despite encountering hills (up and down). How would this be modified to account for head or tail wind (if the impact of the wind is proportional to the speed of the car) [hardish]
4. Explain the "stop start" behaviour of traffic that you sometimes (or often on the M25) encounter without the schemes above being deployed – quite tricky.