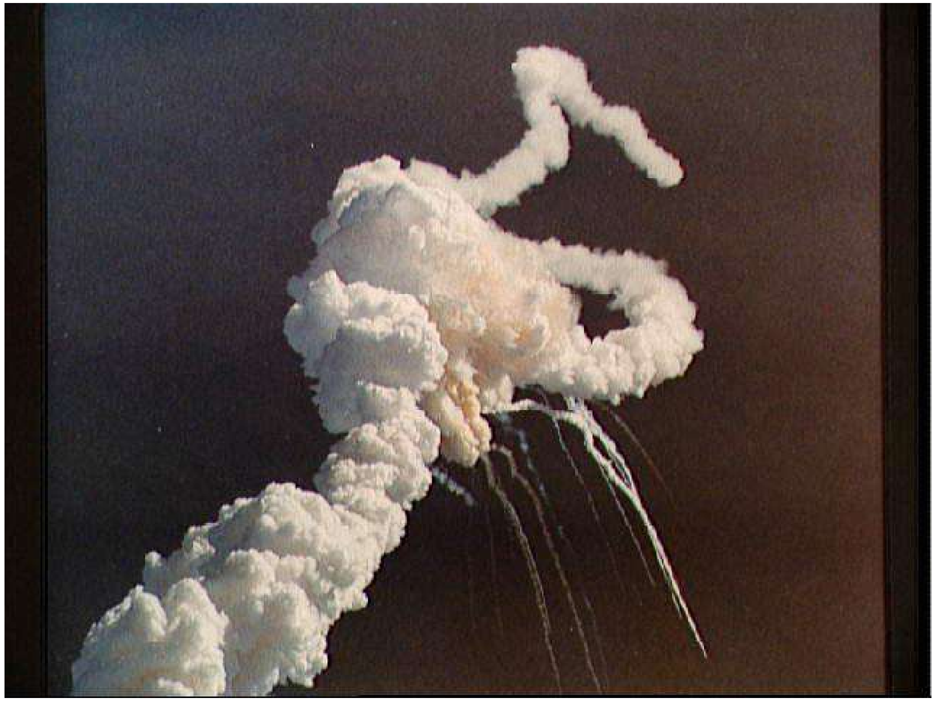


9.1 Dependability - Overview
Sûreté de fonctionnement - Vue d'ensemble
Verlässlichkeit - Übersicht

Prof. Dr. H. Kirrmann & Dr. B. Eschermann
ABB Research Center, Baden, Switzerland

Control Systems Dependability

9.1: Overview Dependable Systems
- Definitions: Reliability, Safety, Availability etc.,
- Failure modes in computers

9.2: Dependability Analysis
- Combinatorial analysis
- Markov models

9.3: Dependable Communication
- Error detection: Coding and Time Stamping
- Persistency

9.4: Dependable Architectures
- Fault detection
- Redundant Hardware, Recovery

9.5: Dependable Software
- Fault Detection,
- Recovery Blocks, Diversity

9.6: Safety analysis
- Qualitative Evaluation (FMEA, FTA)
- Examples

Motivation for Dependable Systems

Systems - if not working properly in a particular situation - may cause

- large losses of property or money
- injuries or deaths of people

To avoid such effects, these “mission-critical” systems must be designed specially so as to achieve a given behaviour in case of failure.

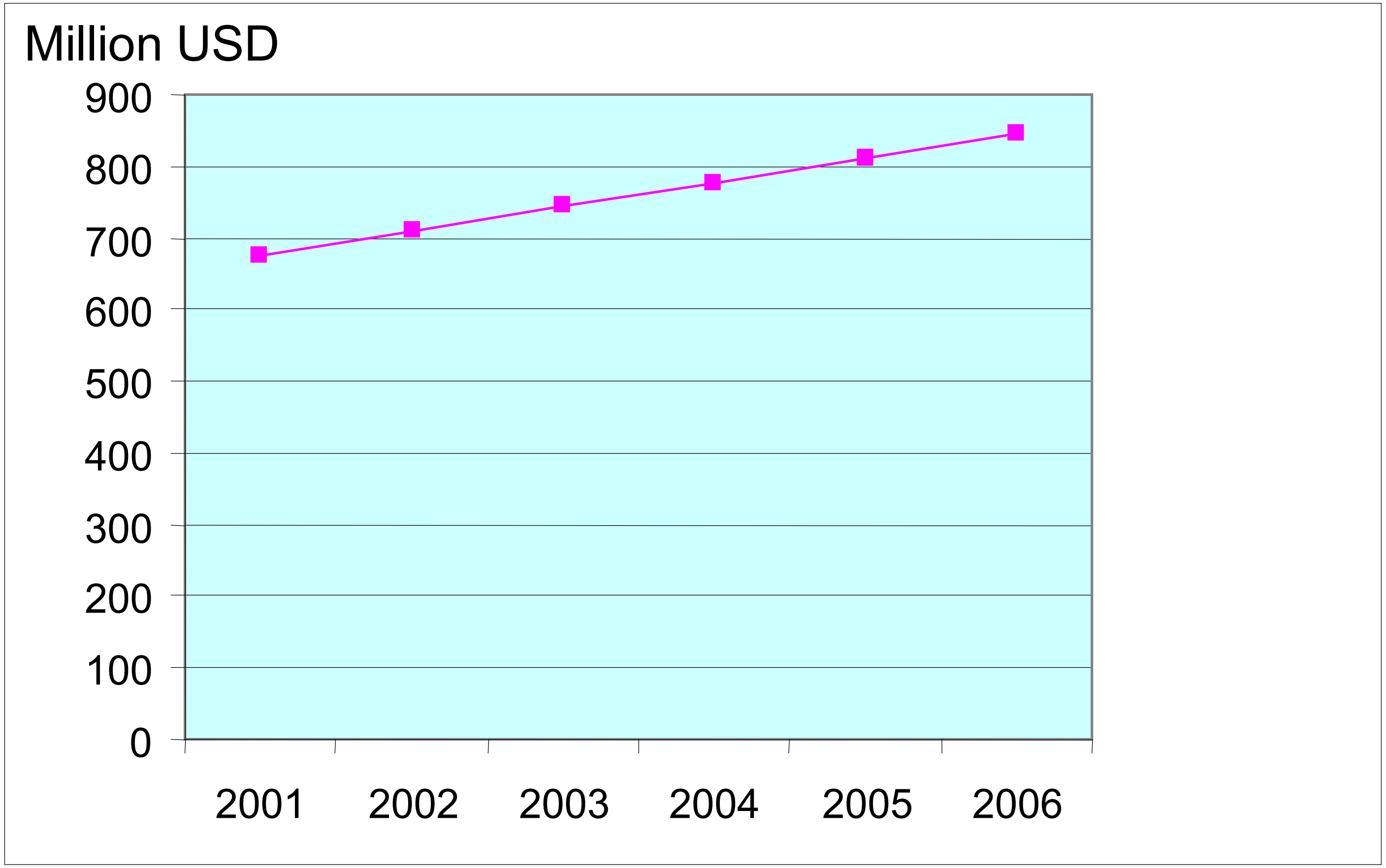
The necessary precautions depend on

- the probability that the system is not working properly
- the consequences of a system failure
- the risk of occurrence of a dangerous situation
- the negative impact of an accident (severity of damage, money lost)

Application areas for dependable systems

Space Applications	Launch rockets, Shuttle, Satellites, Space probes
Transportation	Airplanes (fly-by-wire), Railway signalling, Traffic control, Cars (ABS, ESP, brake-by-wire, steer-by-wire)
Nuclear Applications	Nuclear power plants, Nuclear weapons, Atomic-powered ships and submarines
Networks	Telecommunication networks, Power transmission networks, Pipelines
Business	Electronic stock exchange, Electronic banking, Data stores for Indispensable business data
Medicine	Irradiation equipment, Life support equipment
Industrial Processes	Critical chemical reactions, Drugs, Food

Market for safety- and critical control systems



increases more rapidly than the rest of the automation market

source: ARC Advisory group, 2002, Asish Ghosh

Definitions: Failure, Fault

A *mission* is the intended (specified) function of a device.

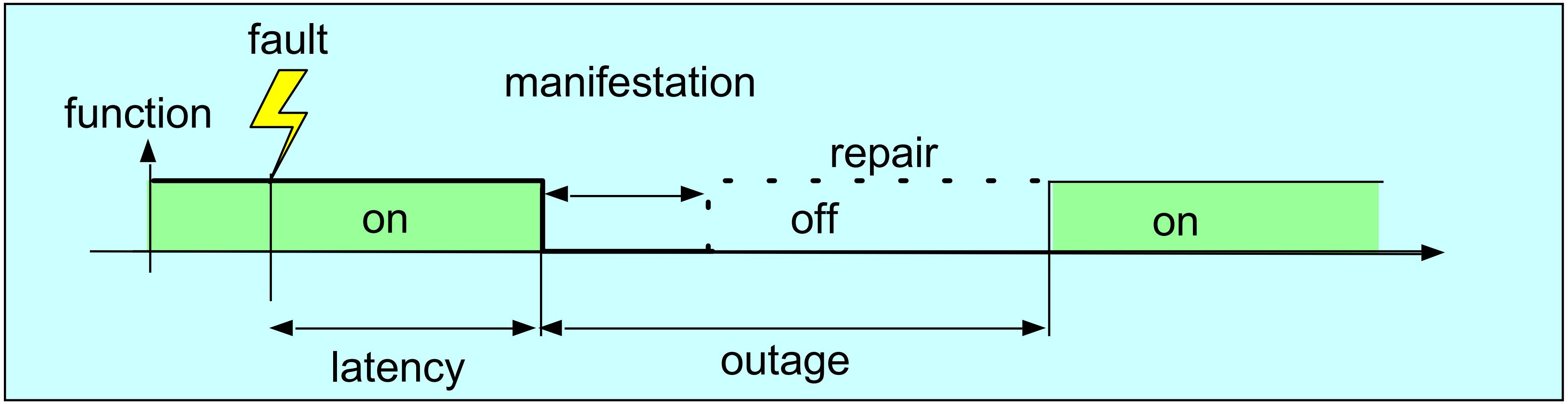
A *failure* (Ausfall, défaillance) is the non-fulfilment of this mission.

("termination of the ability of an item to perform its required function").

failures may be:

- momentary = outage (*Aussetzen,* raté)
- temporary = need repair = breakdown (*Panne*, panne) - for repairable systems only -
- definitive = (*Misserfolg*, échec)

A *fault* (Fehler, défaut) is the cause of a failure, it may occur long before the failure.
These terms can be applied to the whole system, or to elements thereof.



Fault, Error, Failure

Fault: missing or wrong functionality

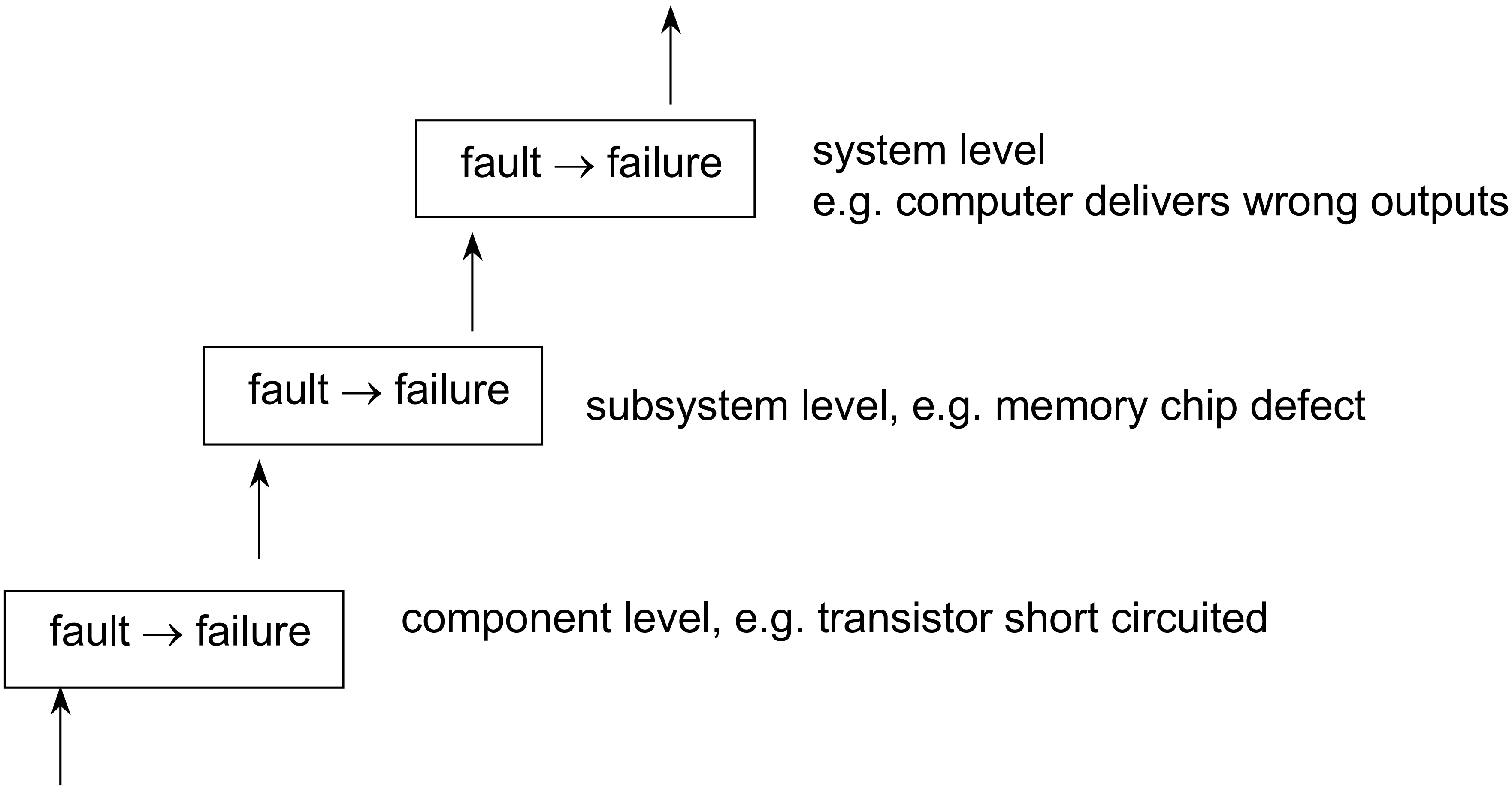
- permanent: due to irreversible change, consistent wrong functionality
  (e.g. short circuit between 2 lines)
- intermittent: sometimes wrong functionality, recurring
  (e.g. loose contact)
- transient: due to environment, reversible if environment changes
  (e.g. electromagnetic interference)

Error: logical manifestation of a fault in an application
(e.g. short circuit leads to computation error if 2 lines carry different signals)

Failure: to perform a prescribed function
(e.g. if different signals on both lines lead to wrong output of chip)

$$\text{fault} \xrightarrow[\text{cause}]{\text{may}} \text{error} \xrightarrow[\text{cause}]{\text{may}} \text{failure}$$

Hierarchy of Faults/Failures



Types of Faults

Computers can be affected by two kinds of faults:

physical faults	design faults
(e.g. hardware faults)	(e.g. software faults)

"a corrected physical fault can occur again with the same probability."

< definition ! >

"a corrected design error does not occur anymore"

Faults are originated by other faults (causality chain).

Physical faults can originate in design faults (e.g. missing cooling fan)

Most work in fault-tolerant computing addresses the physical faults, because it is easy to provide redundancy for the hardware elements.

Redundancy of the design means that several designs are available.

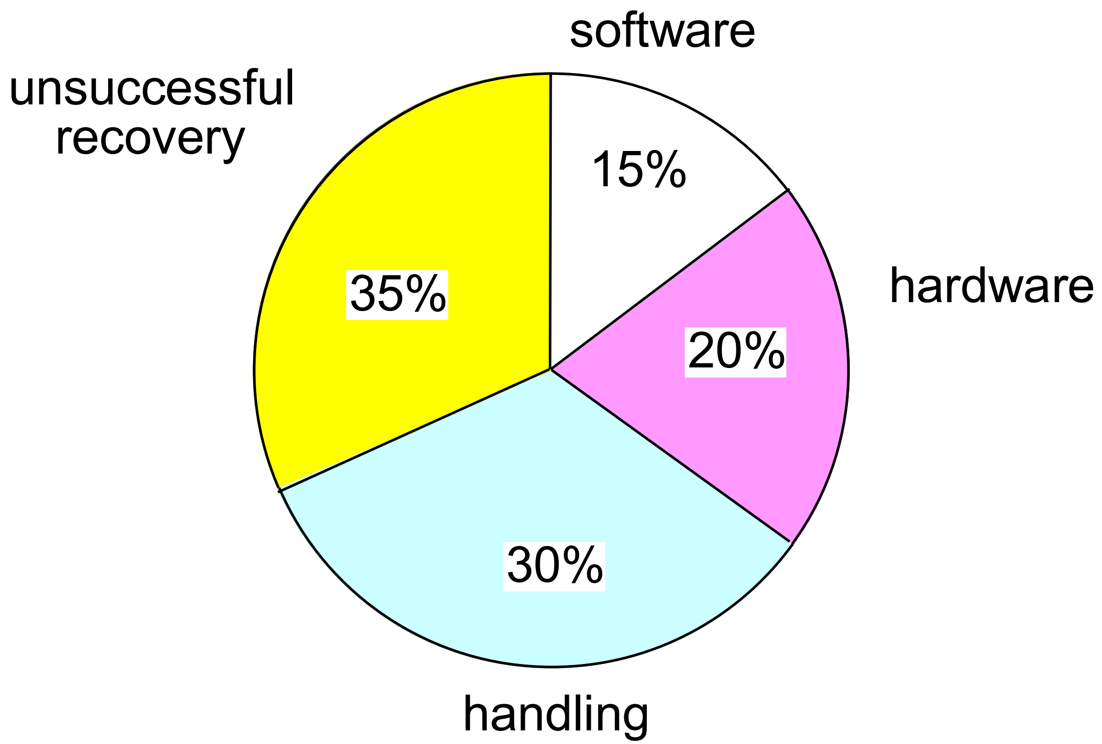
Random and Systematic Errors

Systematic errors are reproducible under given input conditions
Random Error appear with no visible pattern.

Although random errors are often associated with hardware errors and systematic errors with software errors, this needs not be the case

Transient errors , firm errors, soft errors,.... do not use these terms

Example: Sources of Failures in a telephone exchange

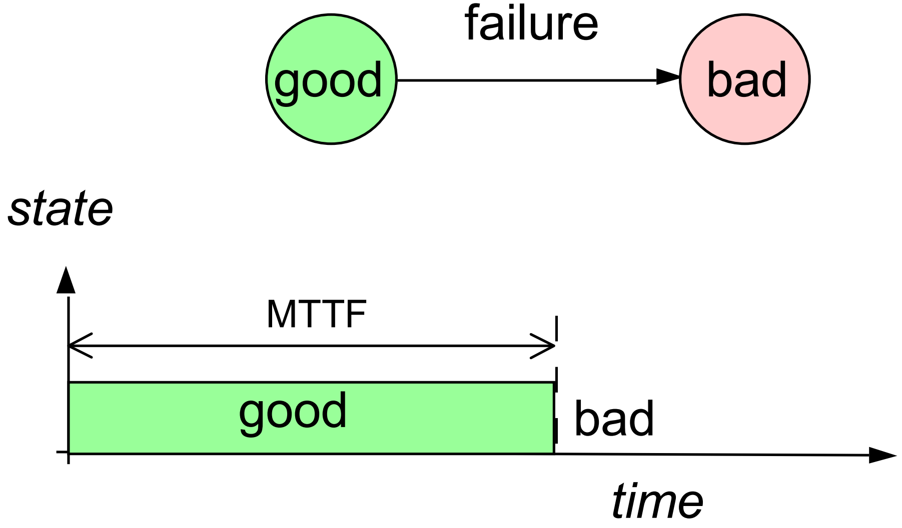


source: Troy, ESS1 (Bell USA)

Basic concepts

Reliability and Availability

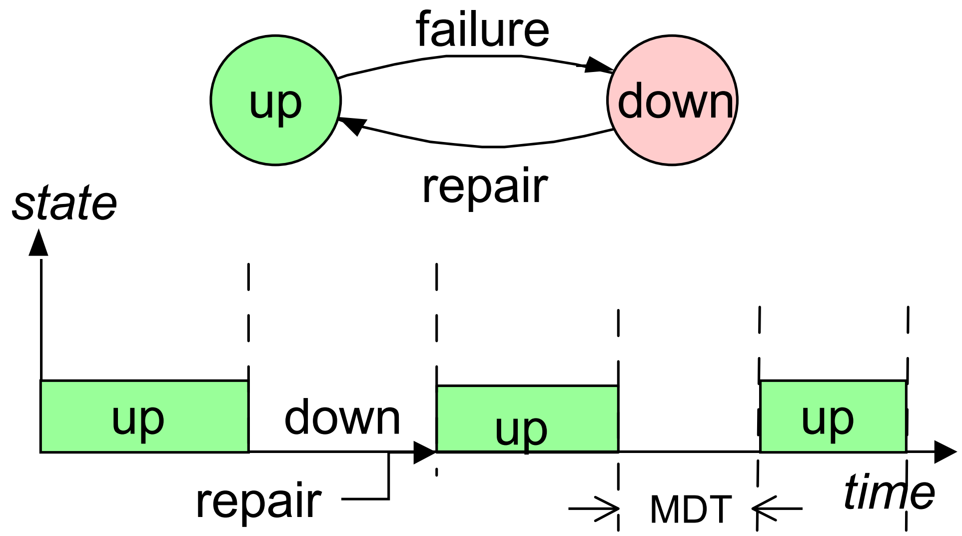
Reliability



definition: "probability that an item will perform its required function in the specified manner and under specified or assumed conditions *over a given time period*"

expressed shortly by its
MTTF: Mean Time To Fail

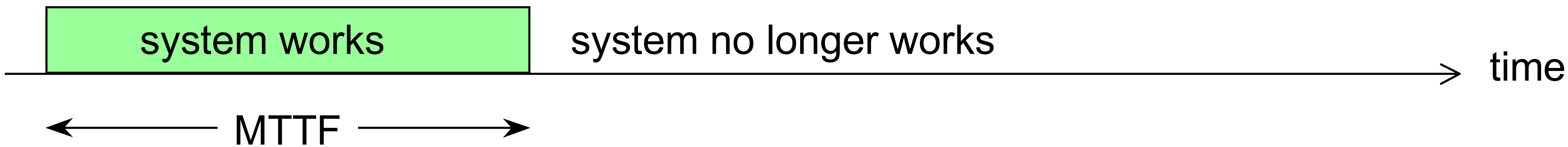
Availability



definition: "probability that an item will perform its required function in the specified manner and under specified or assumed conditions *at a given time* "

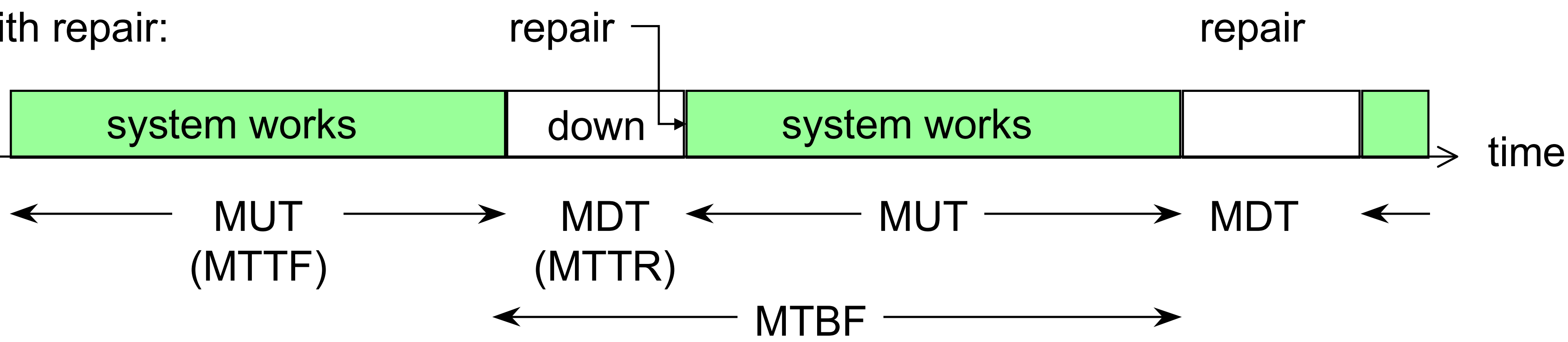
Failure/Repair Cycle

Without repair:



MTTF: mean time to fail

With repair:



MTTR: mean time to repair ~ MDT (mean down time)

MTBF: mean time between failures, (*n'est pas "moyenne des temps de bon fonctionnement«)

MTBF = MTTF + MTTR

if MTTR « MTTF: MTBF ≈ MTTF

Redundancy

Increasing safety or availability requires the introduction of redundancy (resources which are not needed if there were no failures).

Faults are detected by introducing a **check redundancy.**

Operation is continued thanks to **operational redundancy** (can do the same task)

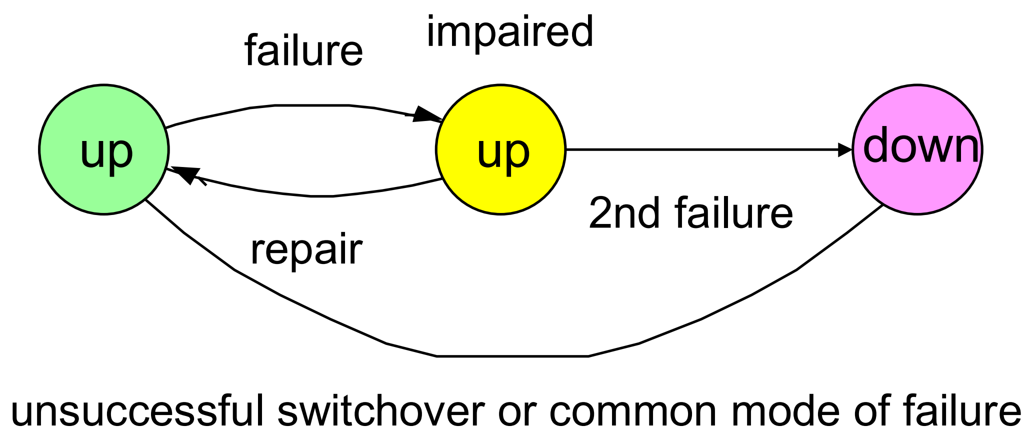
Increasing reliability and maintenance quality increases both safety and availability

detected fault (don´t know about failure)

switch to red:
no accident risk (safe)
decreased traffic performance

switch to green:
accident risk
traffic continues (available)

Availability and Repair in redundant systems



When redundancy is available, the system does not fail until redundancy is exhausted (or redundancy switchover is unsuccessful)

Maintenance

"The combination of all technical and administrative actions, including supervision actions intended to retain a component in, or restore it to, a state in which it can perform its required function"

Maintenance takes the form of

- **corrective maintenance**: executed when a part actually fails (repair)
  "go to the garage when the motor fails"

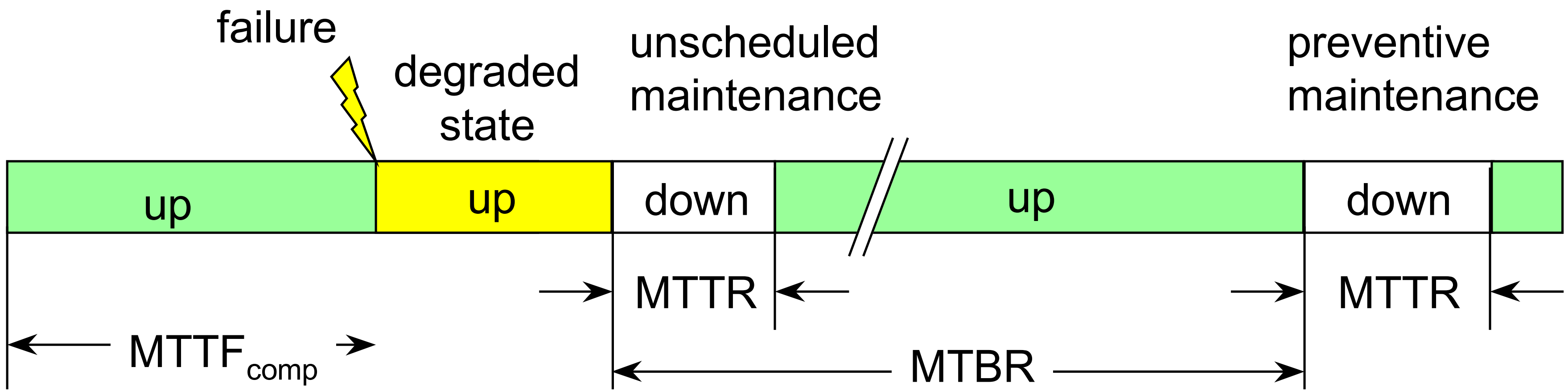
- **preventive maintenance**: restoring redundancy
  and in particular restore degraded parts to error-free state

  "go to the garage to change oil and pump up the reserve tyre"

  - **scheduled maintenance** (time-based maintenance)
    "go to the garage every year"
  - **predictive maintenance** (condition-based maintenance)
    "go to the garage at the next opportunity since motor heats up"

preventive maintenance does not necessarily stop production if redundancy is available
"differed maintenance" is performed in a non-productive time.

Differed maintenance



Redundancy does not replace maintenance:
it allows to differ maintenance to a convenient moment
(e.g. between 02h00 and 04h00 in the morning).

The system may remain on-line or be taken shortly out of operation.

The mean time between repairs (MTBR) expressed how often any component fails

The mean time between failure concerns the whole system.

Differed maintenance is only interesting for plants that are not fully operational 24/24.

Preventive maintenance

In principle, preventive maintenance restores the initially good state at regular intervals.

This assumes that the coverage of the tests is 100% and that no uncorrected aging takes place.

Safety

we distinguish:

•hazards caused by the presence of control system itself:
explosion-proof design of measurement and control equipment
(e.g. Ex-proof devices, see "Instrumentation")

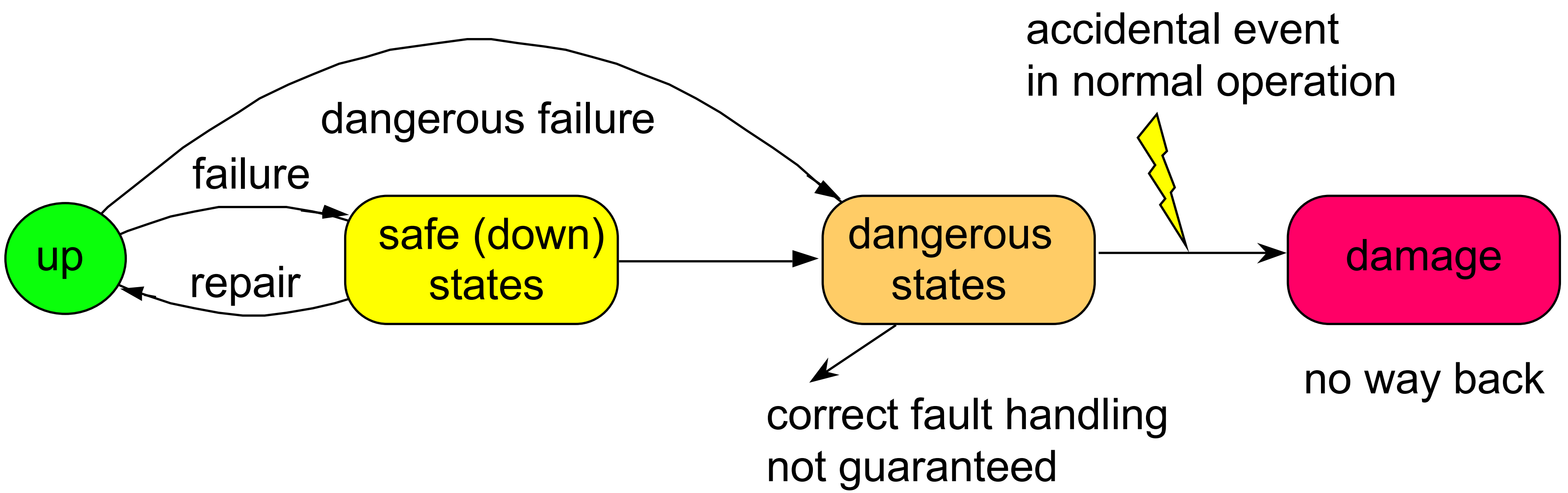
•implementation of safety regulation (protection) by control systems
"safety"- PLC, "safety" switches
(requires tamper-proof design)
protection systems in the large
(e.g. Stamping Press Control (*Pressesteuerungen*),
Burner Control (*Feuerungssteuerungen*)

•hazard directly caused by malfunction of the control system
(e.g. flight control)

Safety

The probability that the system does not behave in a way considered as dangerous.

Expressed by the probability that the system does not enter a state defined as dangerous



difficulty of defining which states are dangerous -

level of damage ? acceptable risk ?

Safe States

Safe state

- exists: sensitive system
- does not exist: critical system

Sensitive systems

- railway: train stops, all signals red (**but**: fire in tunnel?)
- nuclear power station: switch off chain reaction by removing moderator (may depend on how reactor is constructed)

Critical systems

- military airplanes: only possible to fly with computer control system (plane inherently instable)

Types of Redundancy

Structural redundancy (hardware):

Extend system with additional components that are not necessary to achieve the required functionality (e.g. overdimension wire gauge, use 2-out-of-3 computers)

Functional redundancy (software):

Extend the system with unnecessary functions

- –additional functions (e.g. for error detection or to switch to standby unit)
- –diversity (additional different implementation of the required functions)

Information redundancy:

Encode data with more bits than necessary
(e.g. parity bit, CRC, 1-out-of-n-code)

Time redundancy:

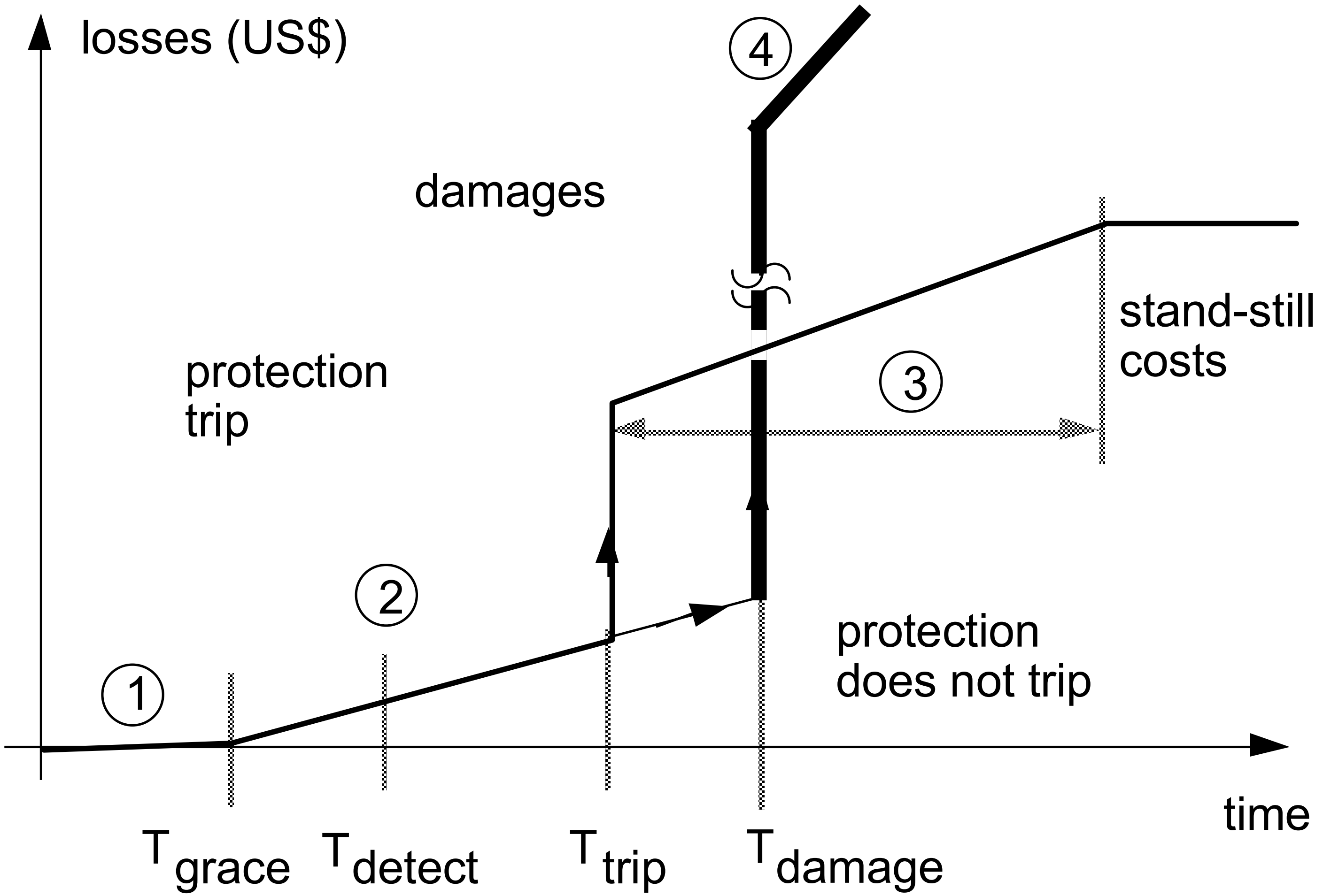
Use additional time, e.g. to do checks or to repeat computation

Availability and Safety (1)

Availability	Safety
availability is an economical objective.	safety is a regulatory objective
high availability increases production time and yield (e.g. airplanes are aloft)	high safety reduces the risk to the process and its environment
The gain can be measured in additional up-time	The gain can be measured in lower insurance rates
availability depends on a functional redundancy (which can take over the function) and on the quality of maintenance	safety depends on the introduction of check redundancy (fail-stop systems) and/or functional redundancy (fail-operate systems)

Safety and Availability are often contradictory (completely safe systems are unavailable) since they share redundancy.

Cost of failure in function of duration



Safety and Security

Safety (Sécurité, *Sicherheit*):

Avoid dangerous situations due to unintentional failures

–failures due to random/physical faults

–failures due to systematic/design faults

e.g. railway accident due to burnt out red signal lamp

e.g. rocket explosion due to untested software (→ Ariane 5)

Security (Sécurité informatique, IT-*Sicherheit*):

Avoid dangerous situations due to malicious threats

–authenticity / integrity (intégrité): protection against tampering and forging

–privacy / secrecy (confidentialité, Vertraulichkeit): protection against eavesdropping

e.g. robbing of money tellers by using weakness in software

e.g. competitors reading production data

The boundary is fuzzy since some unintentional faults can behave maliciously.

(*Sûreté: terme général: aussi probabilité de bon fonctionnement, Verlässlichkeit*)

How to Increase Dependability?

Fault tolerance: Overcome faults without human intervention.

Requires **redundancy**: Resources normally not needed to perform the required function.
Check Redundancy (that can detect incorrect work)
Functional Redundancy (that can do the work)

Contradiction: Fault-tolerance increases complexity and failure rate of the system.

Fault-tolerance is no panacea: Improvements in dependability are in the range of 10..100.

Fault-tolerance is costly:
x 3 for a safe system,
x 4 times for an available 1oo2 system (1-out-of-2),
x 6 times for a 2oo3 (2-out-of-3) voting system

Fault-tolerance is no substitute for quality

Dependability

(*Sûreté de fonctionnement, Verlässlichkeit*)

goals
- reliability
- availability
- maintainability
- safety
- security

achieved by
- fault avoidance
- fault detection/diagnosis
- fault tolerance
  (= error avoidance)

guaranteed by
- quantitative analysis
- qualitative analysis

by error passivation
- fault isolation
- reconfiguration
  (on-line repair)

by error recovery
- forward recovery
- backward recovery

by error compensation
- fault masking
- error correction

Failure modes in computers

9.1: Overview Dependable Systems

- Definitions: Reliability, Safety, Availability etc.,
- Failure modes in computers

9.2: Dependability Analysis

- Combinatorial analysis
- Markov models

9.3: Dependable Communication

- Error detection: Coding and Time Stamping
- Persistency

9.4: Dependable Architectures

- Fault detection
- Redundant Hardware, Recovery

9.5: Dependable Software

- Fault Detection,
- Recovery Blocks, Diversity

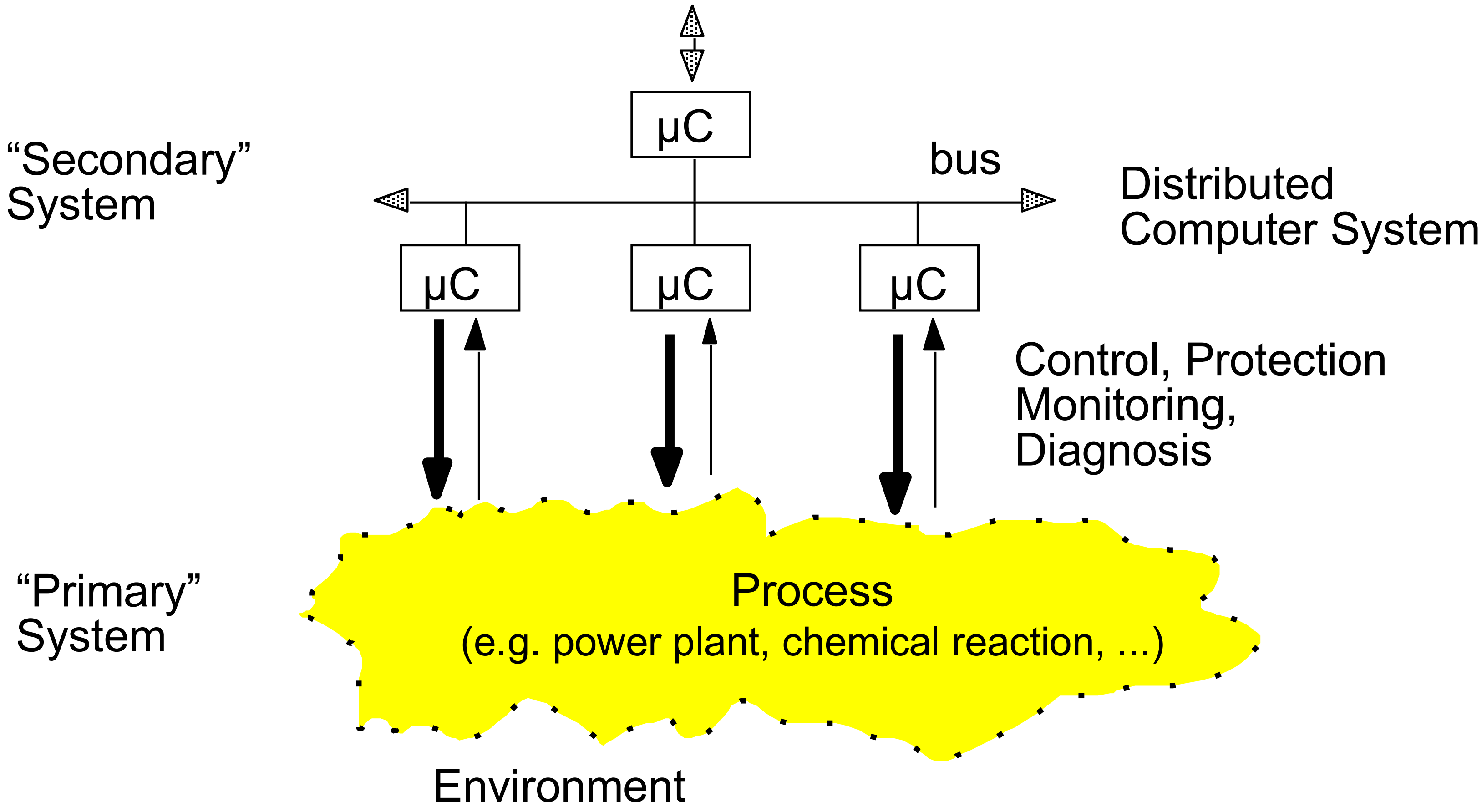
9.6: Safety analysis

- Qualitative Evaluation (FMEA, FTA)
- Examples

Failure modes in computers

Safety or availability can only be evaluated considering the total system controller + plant.

Computers and Processes



Availability/safety depends on **output** of computer system and process/environment.

Types of Computer Failures

Computers can fail in a number of ways

Breach of the specifications = does not behave as intended

reduced to two cases

output of wrong data or of correct data,but at undue time → integrity breach	missing output of correct data → persistency breach

Fault-tolerant computers allow to overcome these situations.

The architecture of the fault-tolerant computer depends on the encompassed dependability goals

Safety Threats

depending on the controlled process,

safety can be threatened by failures of the control system:

integrity breach

not recognized, wrong data, or correct data, but at the wrong time

if the process is irreversible

(e.g. closing a high power breaker, banking transaction)

Requirement:
fail-silent (fail-safe, fail-stop) computer
"rather stop than fail"

persistency breach

no usable data, loss of control

if the process has no safe side

(e.g. landing aircraft)

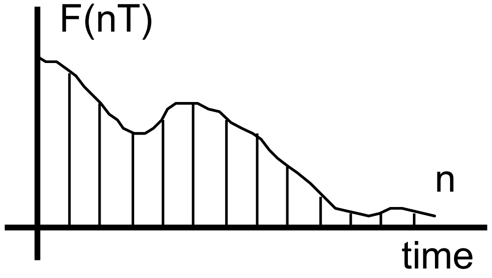
Requirement:
fail-operate computer
"rather some wrong data than none"

Safety depends on the tolerance of the process against failure of the control system

Plant type and dependability

continuous systems

modelled by differential equations, and in the linear case, by Laplace or z-transform (sampled)



continuous systems are generally reversible.

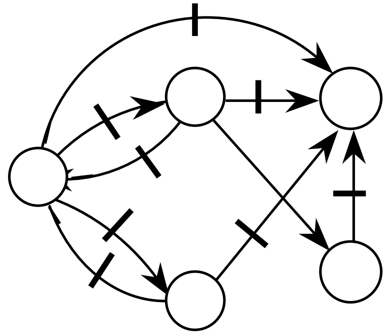
tolerates sporadic, wrong inputs during a limited time (similar: noise)

tolerate loss of control only during a short time.

require persistent control

discrete systems

modelled by state machines, Petri nets, Grafcet,....



transitions between states are normally irreversible.

do not tolerate wrong input.
difficult recovery procedure

tolerate loss of control during a relatively long time (remaining in the same state is in general safe).

require integer control

Persistency/Integrity by Application Examples

secondary system \ primary system	availability	safety
integrity	substation protection	railway signalling
persistency	airplane control	

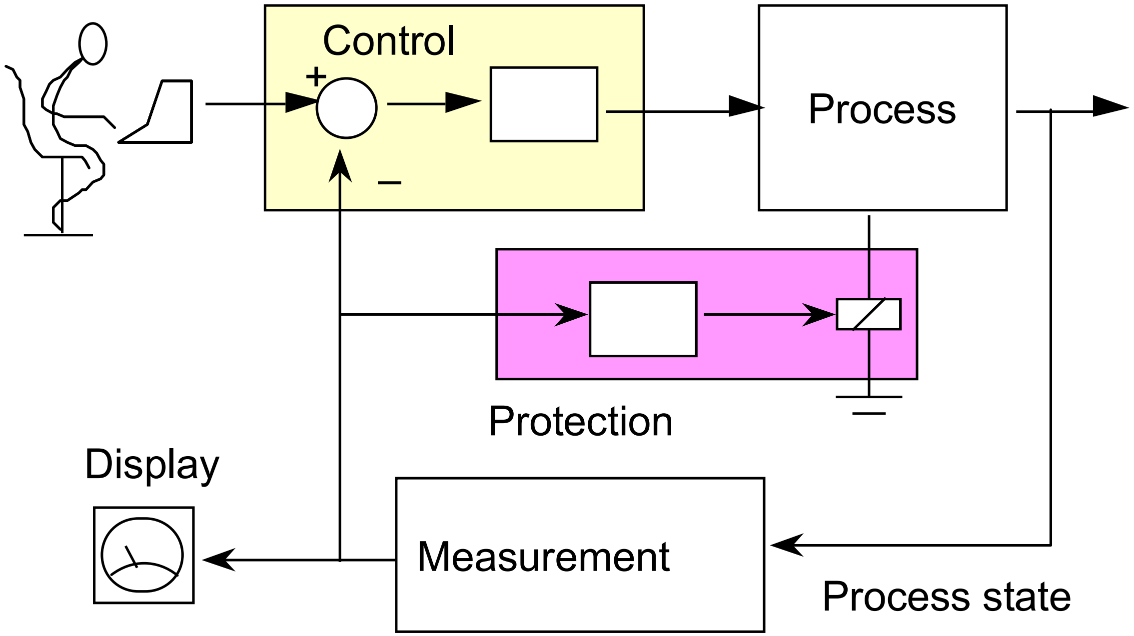
Protection and Control Systems

Control system:

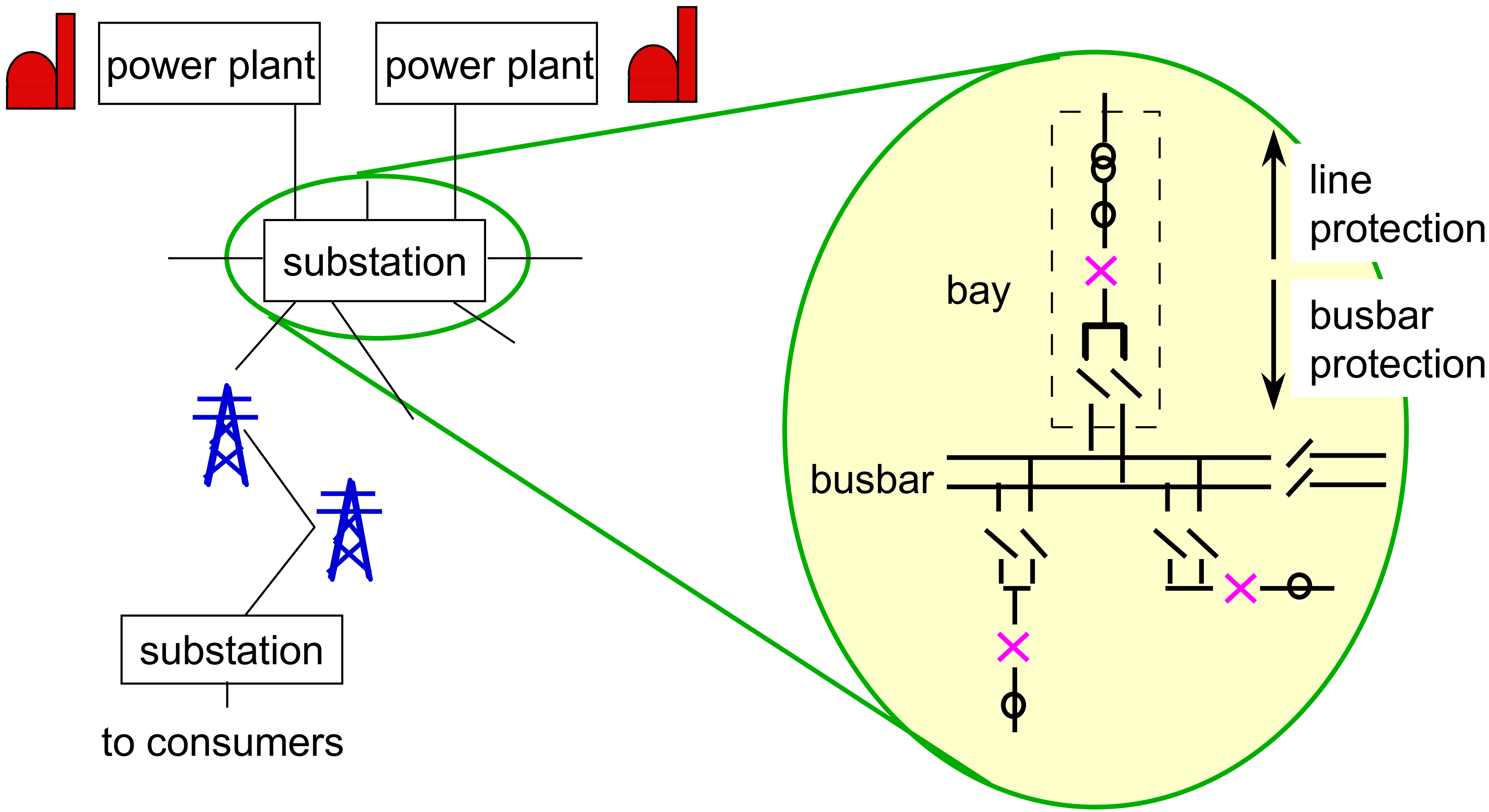
Continuous non-stop operation
(open or closed loop control)
Maximal failure rate given in failures per hour.

Protection system:

Not acting normally,
forces safe state (trip) if necessary
Maximal failure rate given in failures per demand.



Example Protection Systems: High-Voltage Transmission



Two kinds of malfunctions:
An underfunction (not working when it should) of a protection system is a safety threat
An overfunction (working when it should not) of a protection system is an availability threat

Findings

Reliability and fault tolerance must be considered early in the development process, they can hardly be increased afterwards.

Reliability is closely related to the concept of quality, its root are laid in the design process, starting with the requirement specs, and accompanying through all its lifetime.

References

H. Nussbaumer: Informatique industrielle IV; PPUR.

J.-C. Laprie (ed.): Dependable computing and fault tolerant systems; Springer.

J.-C. Laprie (ed.): Guide de la sûreté de fonctionnement; Cépaduès.

D. Siewiorek, R. Swarz: The theory and practice of reliable system design; Digital Press.

T. Anderson, P. Lee: Fault tolerance - Principles and practice; Prentice-Hall.

A. Birolini: Quality and reliability of technical systems; Springer.

M. Lyu (ed.): Software fault tolerance: Wiley.

Journals: IEEE Transactions on Reliability, IEEE Transactions on Computers

Conferences: International Conference on Dependable Systems and Networks, European Dependable Computing Conference

Assessment

which kinds of fault exist and how are they distinguished

explain the difference between reliability, availability, safety in terms of a state diagram

explain the trade-off between availability and safety

what is the difference between safety and security

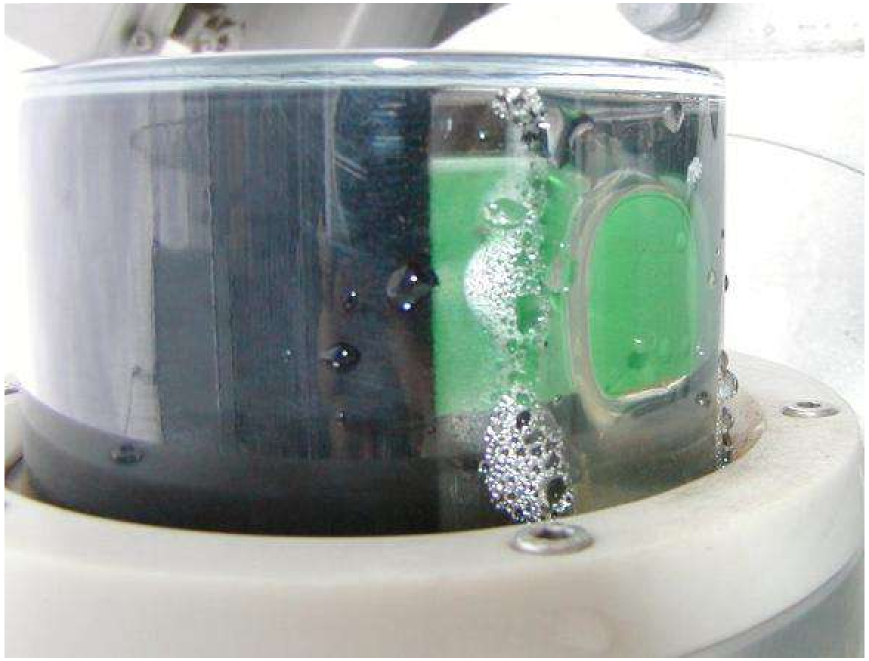
explain the terms MTTF, MTTR, MTBF, MTBR

how does a protection system differ from a control system when considering failures ?

which forms of redundancy exist for computers ?

how does the type of plant influence its behaviour towards faults ?

Industrial Automation
Automation Industrielle
Industrielle Automation



9.2 Dependability - Evaluation

Estimation de la fiabilité

Verlässlichkeitsabschätzung

Prof. Dr. H. Kirrmann
ABB Research Center, Baden, Switzerland

2006-06-14, HK

Dependability Evaluation

This part of the course applies to any system that may fail.

Dependability evaluation (*fiabilité prévisionnelle*, Verlässlichkeitsabschätzung) determines:

- the expected reliability,
- the requirements on component reliability,
- the repair and maintenance intervals and
- the amount of necessary redundancy.

Dependability analysis is the base on which risks are taken and contracts established

Dependability evaluation must be part of the design process, it is quite useless once a system has been put into service.

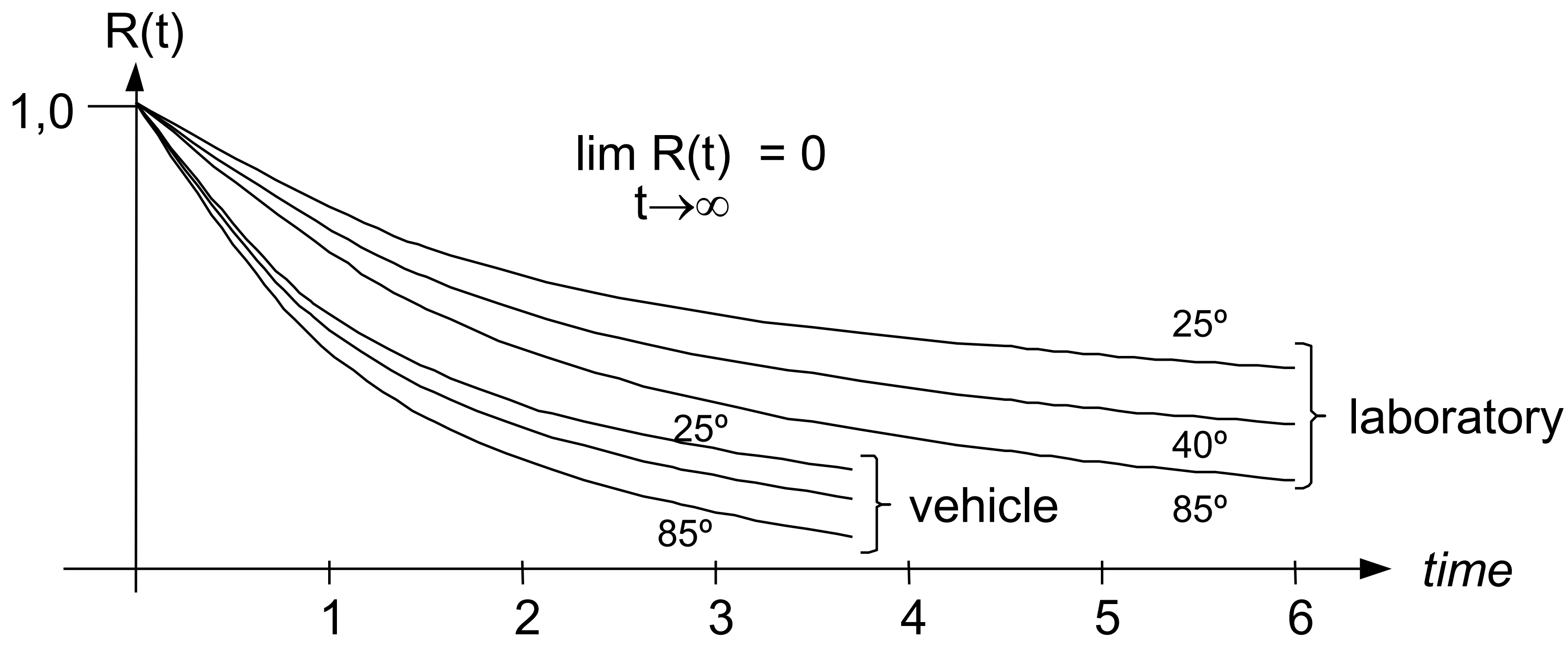
9.2.1 Reliability definitions

Reliability

Reliability = probability that a mission is executed successfully
(definition of success? : a question of satisfaction…)

Reliability depends on:

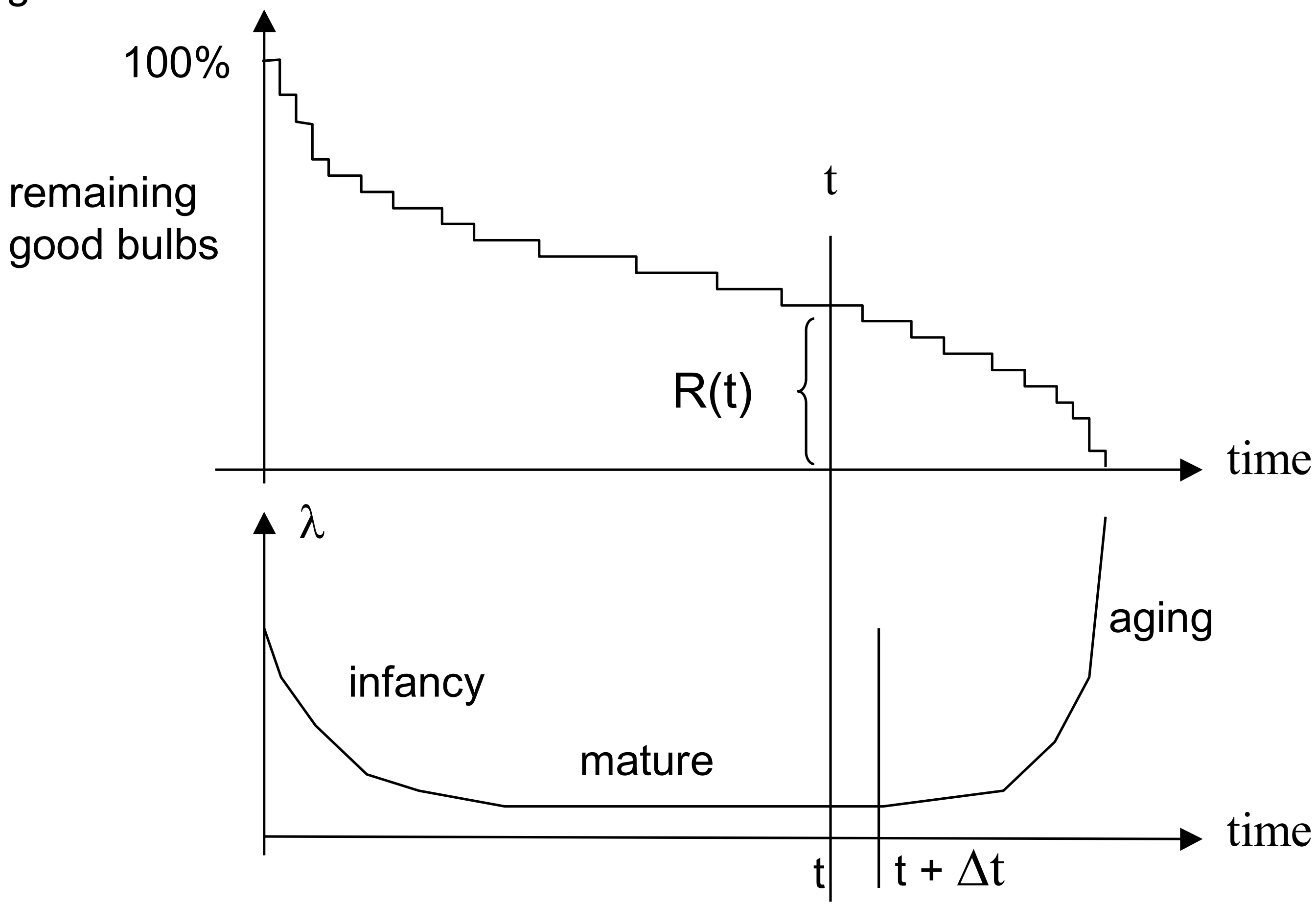
- duration (“*tant va la cruche à l’eau….*”, "der Krug geht zum Brunnen bis er bricht))
- environment: temperature, vibrations, radiations, etc...



Such graphics are obtained by observing a large number of systems,
or calculated for a system knowing the expected behaviour of the elements.

Reliability and failure rate - Experimental view

Experiment: large quantity of light bulbs



Reliability **R(t):** number of good bulbs remaining at time t divided by initial number of bulbs

Failure rate **λ(t)**: number of bulbs that failed in interval t, t+Δt, divided by number of remaining bulbs

Reliability R(t) definition

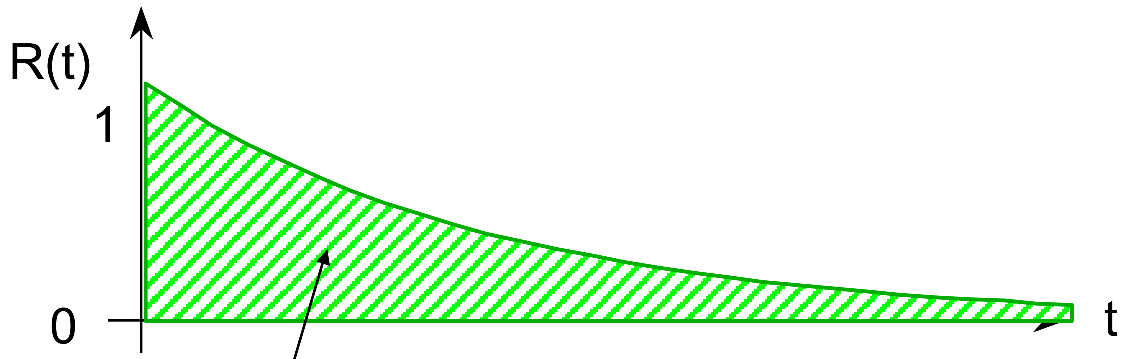
Reliability R(t): probability that a system does not enter a terminal state until time t,
while it was initially in a good state at time t=0"

$$R(0) = 1; \qquad \lim_{t \to \infty} R(t) = 0$$

Failure rate **λ(t)** = probability that a (still good) element fails during the next time unit dt.

$$\lambda(t) = -\frac{dR(t)/dt}{R(t)}$$

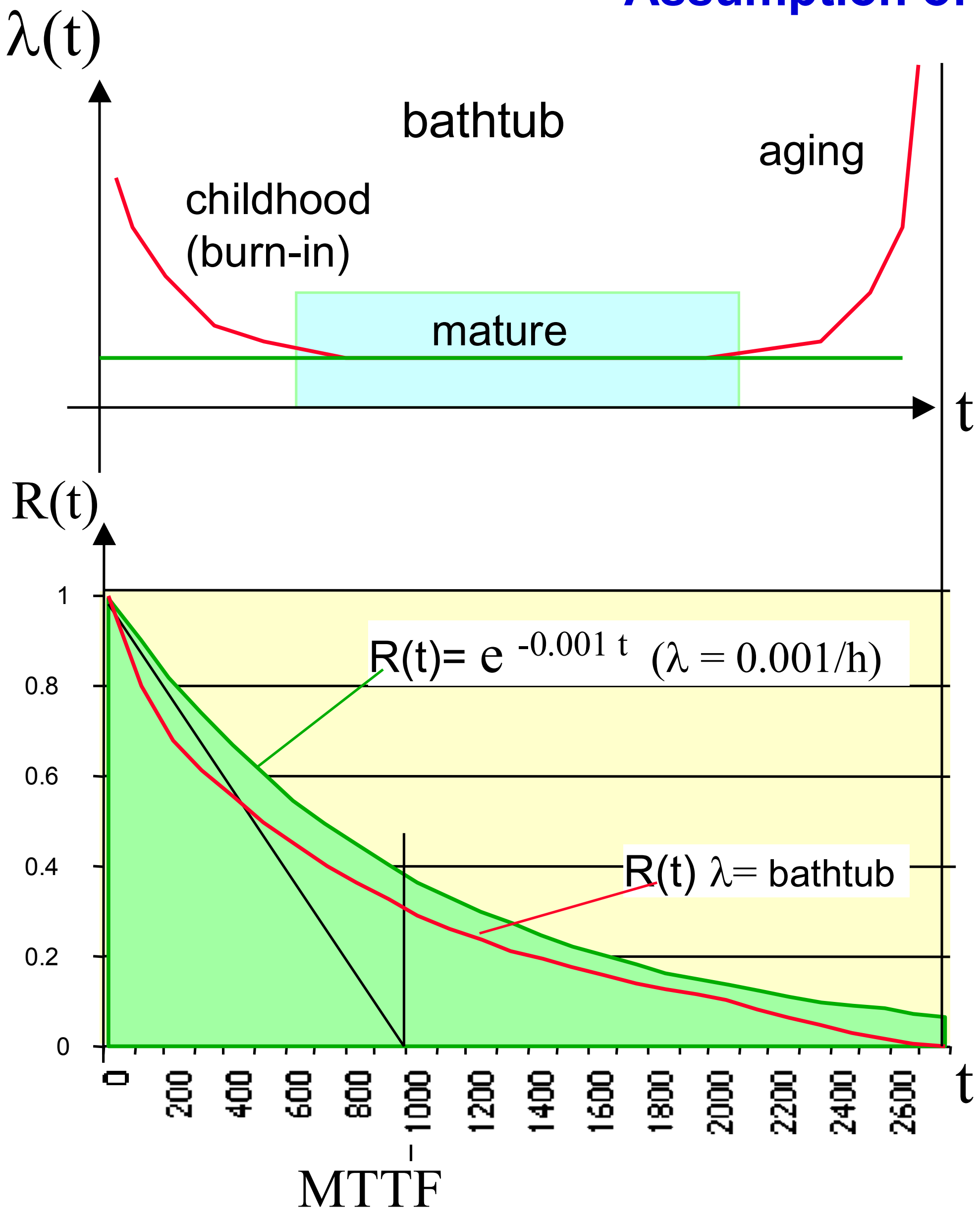
and: $R(t) = e^{-\int_0^t \lambda(x)\,dx}$



MTTF = mean time to fail = surface below R(t)

$$MTTF = \int_0^\infty R(t)\,dt$$

Assumption of constant failure rate



Reliability = probability of not having failed until time t expressed:

by discrete expression

$$R\ (t+\Delta t) = R\ (t) - R\ (t)\ \lambda(t)^*\Delta t$$

by continuous expression simplified when λ = constant

$$R\ (t) = e^{-\lambda t}$$

assumption of λ = constant is justified by experience, simplifies computations significantly

MTTF = mean time to fail = surface below R(t)

$$MTTF = \int_0^\infty e^{-\lambda t}\, dt = \frac{1}{\lambda}$$

Examples of failure rates

To avoid the negative exponentials, λ values are often given in FIT (Failures in Time),

$$1 \text{ fit} = 10^{-9} \text{ /h} = \frac{1}{114'000 \quad \text{years}}$$

Element	Rating	failure rate
resistor	0.25 W	0.1 fit
capacitor	(dry) 100 nF	0.5 fit
capacitor	(elect.) 100 μF	10 fit
processor	486	500 fit
RAM	4MB	1 fit
Flash	4MB	12 fit
FPGA	5000 gates	80 fit
PLC	compact	6500 fit
digital I/O	32 points	2000 fit
analog I/O	8 points	1000 fit
battery	per element	400 fit
VLSI	per package	100 fit
soldering	per point	0.01 fit

These figures can be obtained from catalogues such as MIL Standard 217F or from the manufacturer’s data sheets.

Warning: Design failures outweigh hardware failures for small series

MIL HDBK 217 (1)

MIL Handbook 217B lists failure rates of common elements.

Failure rates depend strongly on the environment:
temperature, vibration, humidity, and especially the location:

- Ground benign, fixed, mobile
- Naval sheltered, unsheltered
- Airborne, Inhabited, Uninhabited, cargo, fighter
- Airborne, Rotary, Helicopter
- Space, Flight

Usually the application of MIL HDBK 217 results in pessimistic results in terms of the overall system reliability (computed reliability is lower than actual reliability).

To obtain more realistic estimations it is necessary to collect failure data based on the actual application instead of using the generic values from MIL HDBK 217.

Failure rate catalogue MIL HDBK 217 (2)

Stress is expressed by lambda factors

Basic models:

- discrete components (e.g. resistor, transistor etc.)
  $\lambda = \lambda_b \, p_E \, p_Q \, p_A$
- integrated components (ICs, e.g. microprocessors etc.)
  $\lambda = p_Q \, p_L \, (C_1 \, p_T \, p_V + C_2 \, p_E)$

MIL handbook gives curves/rules for different element types to compute factors,

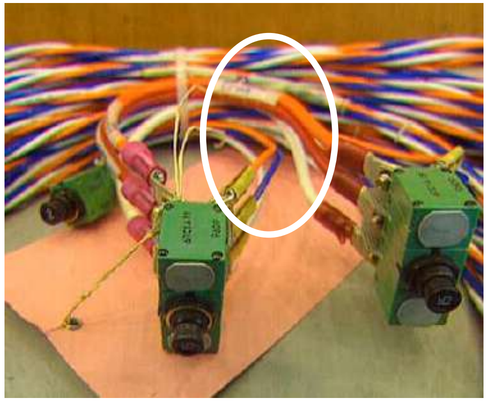
- λ_b based on ambient temperature Q_A and electrical stress S
- p_E based on environmental conditions
- p_Q based on production quality and burn-in period
- p_A based on component characteristics and usage in application
- C_1 based on the complexity
- C_2 based on the number of pins and the type of packaging
- p_T based on chip temperature Q_J and technology
- p_V based on voltage stress

Example: λ_b usually grows exponentially with temperature Θ_A (Arrhenius law)

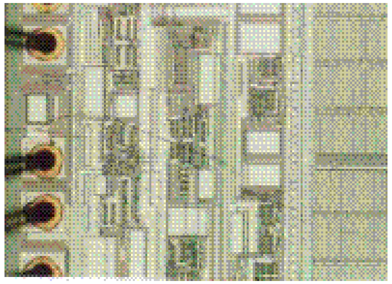
What can go wrong…

poor soldering (manufacturing)…

broken wire… (vibrations)



broken isolation (assembly…)



chip cracking (thermal stress…)

Failures that affect logic circuits

Thermal stress (different dilatation coefficients, contact creeping)
Electrical stress (electromagnetic fields)
Radiation stress (high-energy particles, cosmic rays in the high atmosphere)

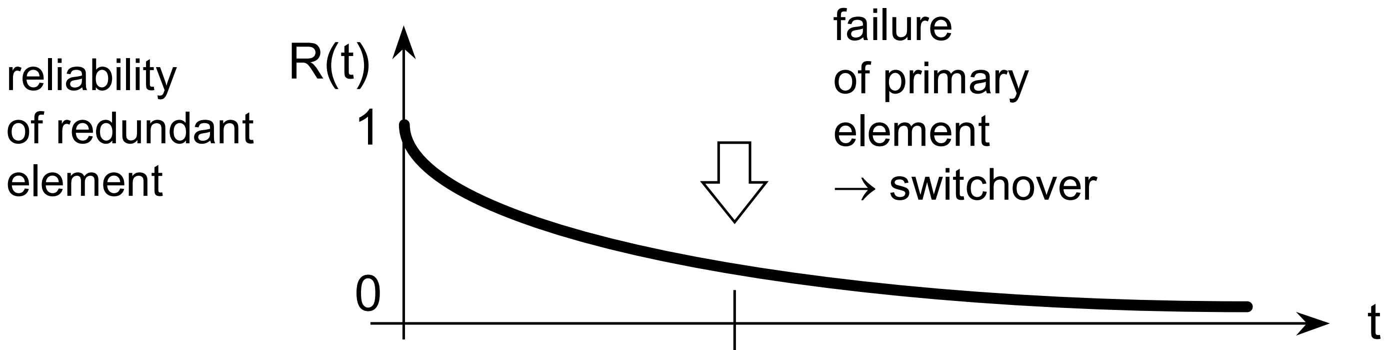
Errors that are transient in nature (called soft-errors) can be latched in memory systems and become firm errors. Solid errors will not disappear at restart.

E.g. FPGA with 3 M gates, exposed to 9.3 108 neutrons/cm^2 exhibited
320 FIT at sea level and 150000 FIT at 20 km altitude
(see: http:\\www.actel.com/products/rescenter/ser/index.html)

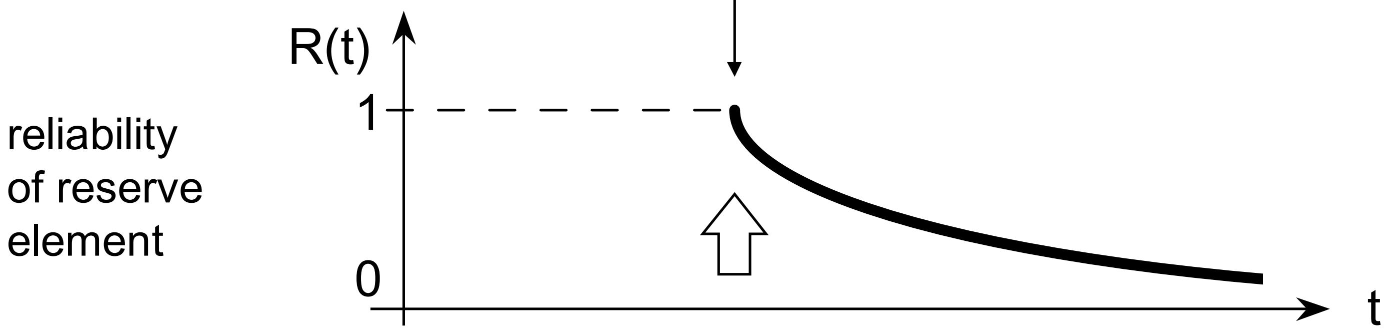
Cold, Warm and Hot redundancy

Hot redundancy: the reserve element is fully operational and under stress, it has the same failure rate as the operating element.

Warm redundancy: the reserve element can take over in a short time, it is not operational and has a smaller failure rate.



Cold redundancy (cold standby): the reserve is switched off and has zero failure rate



9.2.2 Reliability of series and parallel systems (combinatorial)

9.2.1 Reliability definitions

9.2.2 Reliability of series and parallel systems

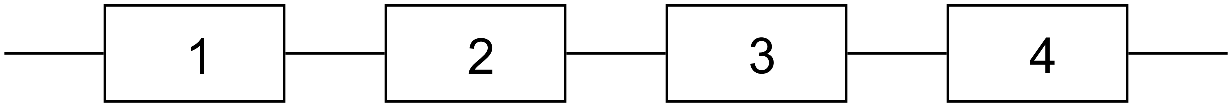
9.2.3 Considering repair

9.2.4 Markov models

9.2.5 Availability evaluation

9.2.6 Examples

Reliability of a system of unreliable elements



The reliability of a system consisting of n elements, each of which is necessary for the function of the system, whereby the elements fail independently is:

$$R_{total} = R_1 * R_2 * .. * R_n = \prod_{I=1}^{n} (R_i)$$

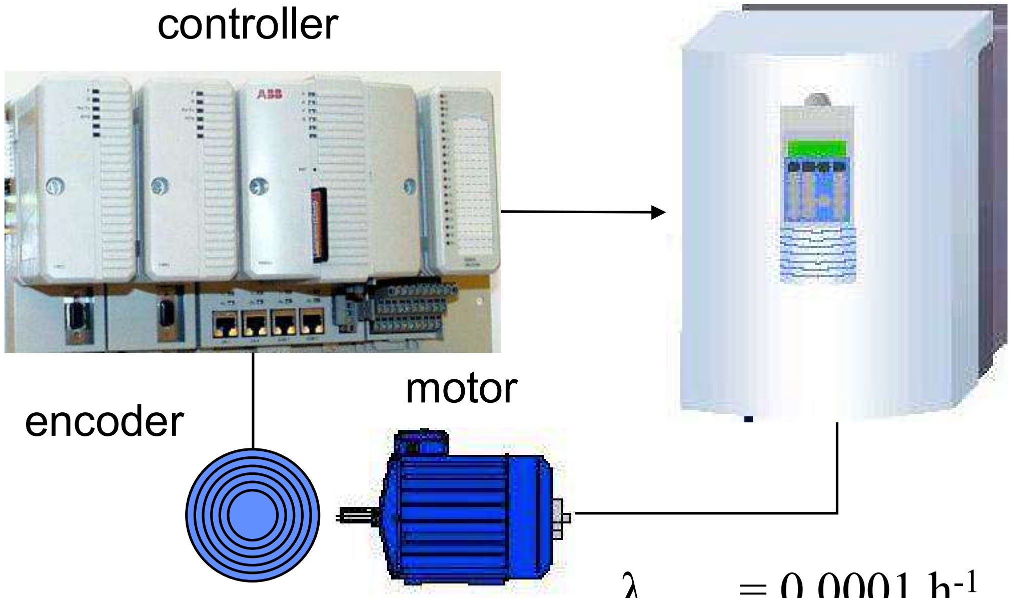
Assuming a constant failure rate λ allows to calculate easily the failure rate of a system by summing the failure rates of the individual components.

$$R_{NooN} = e^{-\Sigma\lambda_i t}$$

This is the base for the calculation of the failure rate of systems (MIL-STD-217F)

Example: series system, combinatorial solution

$\lambda_{control} = 0.00005\ h^{-1}$



$\lambda_{supply} = 0.001\ h^{-1}$

$\lambda_{motor} = 0.0001\ h^{-1}$

power supply → motor+encoder → controller

$$R_{tot} = R_{supply} * R_{motor} * R_{control}$$

$$= e^{-\lambda_{supply} t} * e^{-\lambda_{motor} t} * e^{-\lambda_{control} t} = e^{-(\lambda_{supply} + \lambda_{motor} + \lambda_{control}) t}$$

$$\lambda_{total} = \lambda_{supply} + \lambda_{motor} + \lambda_{control} = 0.00115\ h^{-1}$$

This does not apply any more for redundant system !

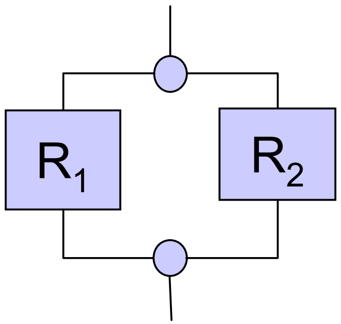
Exercise: MTTF calculation

An embedded controller consists of:
- one microprocessor 486
- 2 x 4 MB RAM
- 1 x Flash EPROM
- 50 dry capacitors
- 5 electrolytic capacitors
- 200 resistors
- 1000 soldering points
- 1 battery for the real-time-clock

what is the MTTF of the controller and what is its weakest point ?

Parallel system 1 out of 2 with no repair - combinatorial solution

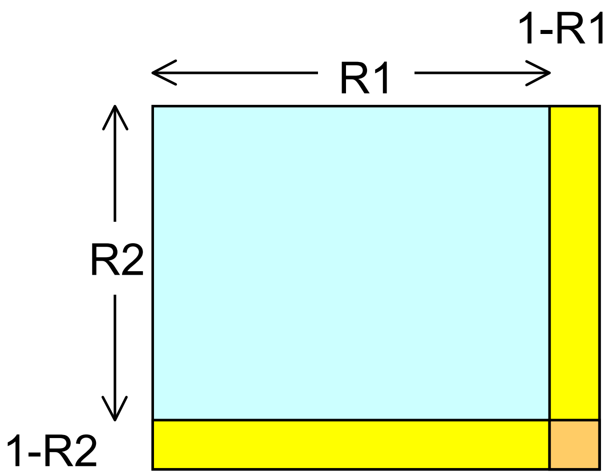
simple redundant system:
the system is good if any (or both) are good



R_1	ok		ok	
R_2	ok	ok		

R_1 good, R_2 good — R_1 good, R_2 down — R_1 down, R_2 good

$$R_{1oo2} = \underbrace{R_1 R_2}_{R_1 \text{ good},\ R_2 \text{ good}} + \underbrace{R_1 (1-R_2)}_{R_1 \text{ good},\ R_2 \text{ down}} + \underbrace{(1-R_1) R_2}_{R_1 \text{ down},\ R_2 \text{ good}}$$



$$R_{1oo2} = 1 - (1-R_2)(1-R_1)$$

with $R_1 = R_2 = R$:

$$R_{1oo2} = 2R - R^2$$

with $R = e^{-\lambda t}$

$$R_{1oo2} = 2e^{-\lambda t} - e^{-2\lambda t}$$

Combinatorial: R1oo2, no repair

Example R_{1oo2}: airplane with two motors

MTTF of one motor = **1000 hours** (this value is rather pessimistic)

Flight duration, t = **2 hours**

- what is the probability that any motor fails ?
- what is the probability that both motors did not fail until time t (landing)?

apply: $R_{1oo1} = e^{-\lambda t}$ single motor doesn't fail: 0.998 (0.2 % chance it fails)

$R_{2oo2} = e^{-2\lambda t}$ no motor failure: 0.996 (0.4 % chance it fails)

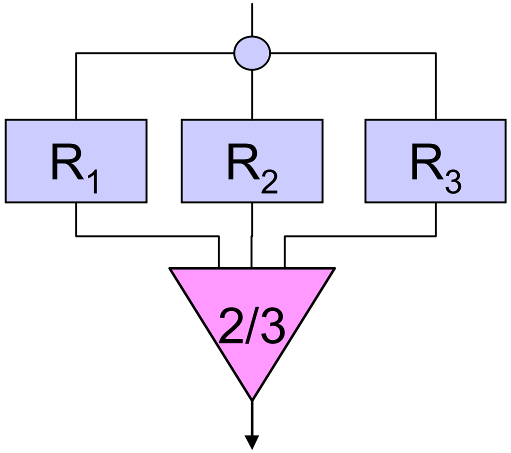
$R_{1oo2} = 2e^{-\lambda t} - e^{-2\lambda t}$ both motors fail: 0.0004 % chance

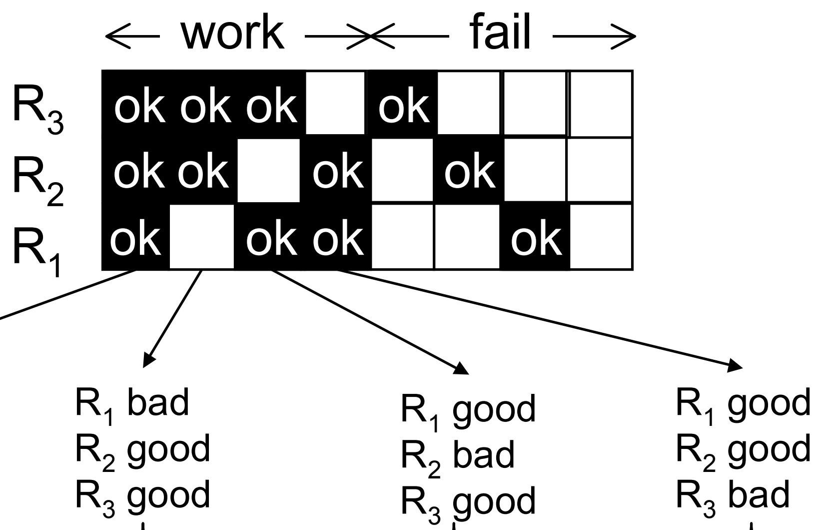
assuming there is no common mode of failure (bad fuel or oil, hail, birds,…)

Combinatorial: 2 out of three system

E.g. three computers,

majority voting





$$R_{2oo3} = R_1R_2R_3 + (1-R_1)R_2R_3 + R_1(1-R_2)R_3 + R_1R_2(1-R_3)$$

with identical elements: $R_1=R_2=R_3= R$

$$R_{2oo3} = 3R^2-2R^3$$

with $R = e^{-\lambda t}$

$$R_{2oo3} = 3\, e^{-2\lambda t} - 2\, e^{-3\lambda t}$$

General case: k out of N Redundancy (1)

K-out-of-N computer (KooN)

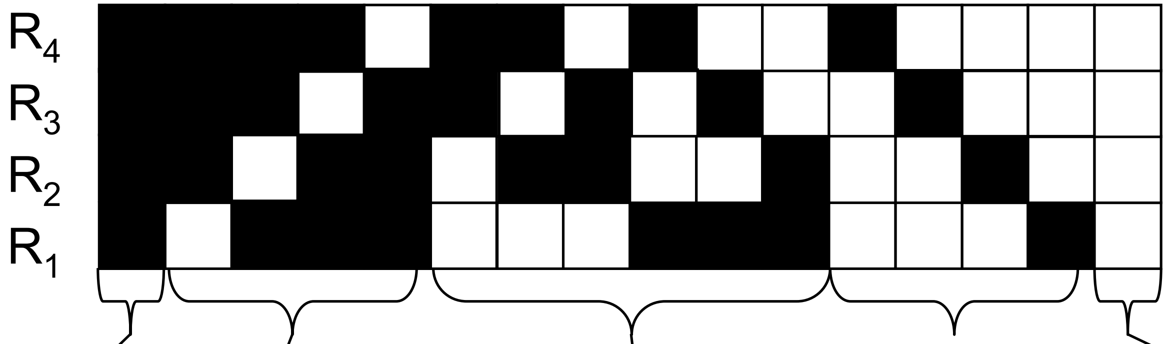
- N units perform the function in parallel
- K fault-free units are necessary to achieve a correct result
- N – K units are “reserve” units, but can also participate in the function

E.g.:

- aircraft with 8 engines: 6 are needed to accomplish the mission.
- voting in computers: If the output is obtained by voting among all N units
  $N \leq 2K - 1$ worst-case assumption: all faulty units fail in same way

General case: k out of N redundancy (2)

Example with
N = 4



$$R_{KooN} = \overbrace{R^N}^{\text{no fail}} + \overbrace{\binom{N}{1} (1-R)\, R^{N-1}}^{\text{one of N fail}} + \overbrace{\binom{N}{2} (1-R)^2 R^{N-2}}^{\text{two of N fail}} + \ldots + \overbrace{\binom{N}{K} (1-R)^K R^{N-K}}^{\text{K of N fail}} + \ldots + \overbrace{(1-R)^N}^{\text{all fail}} = 1$$

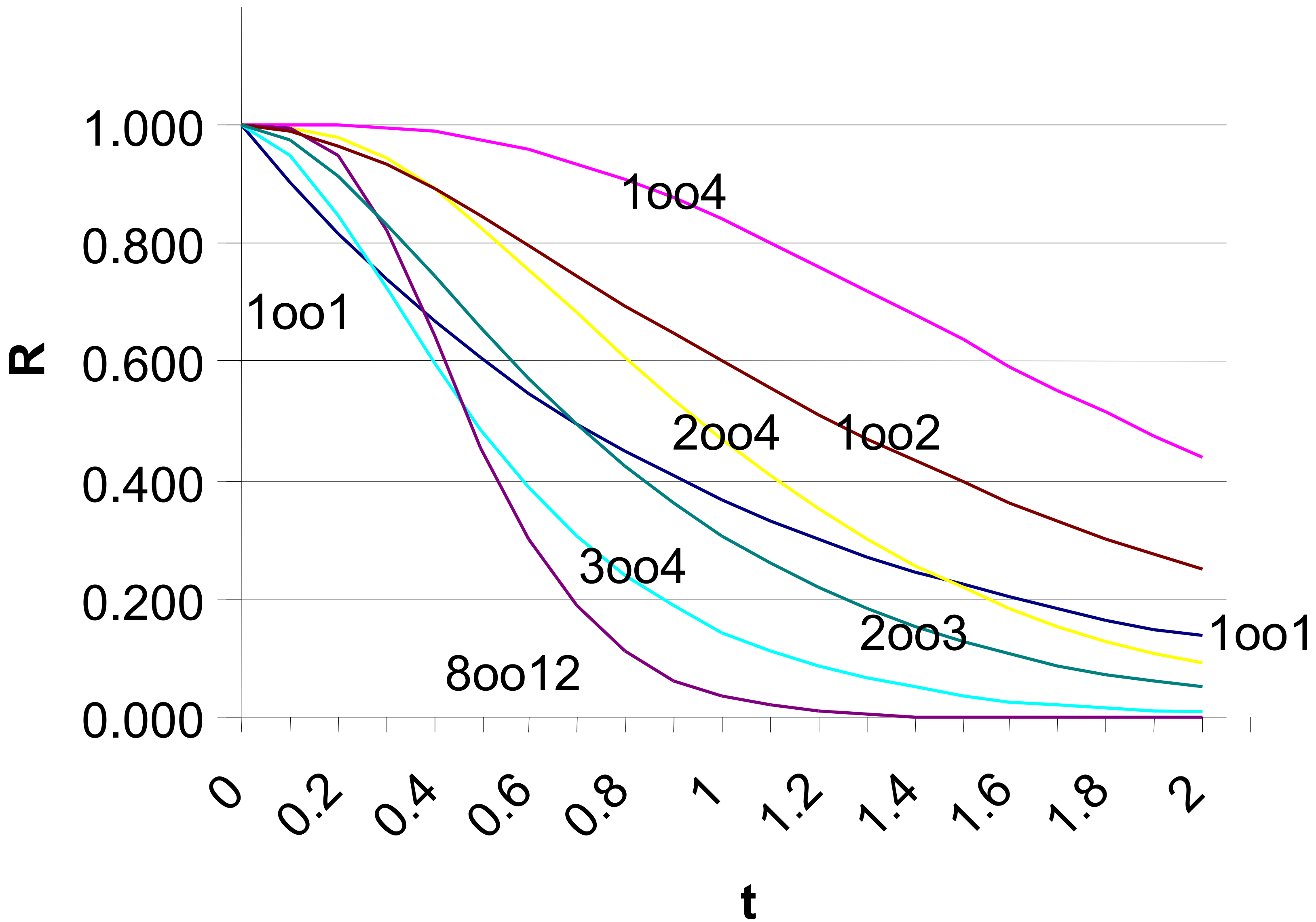
N of N

N + (N-1) of N

N + (N-1) + (N-2) of N

$$R_{KooN} = \sum_{i=0}^{K} \binom{N}{i} R^i (1-R)^{N-i}$$

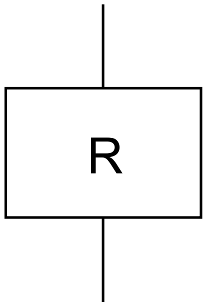
Comparison chart



Summary

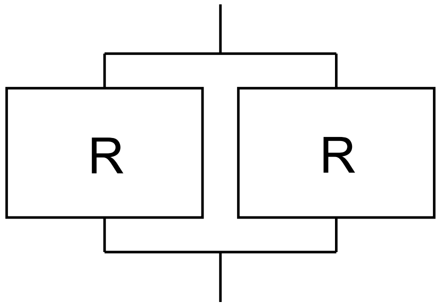
Assumes: all units have identical failure rates and comparison/voting hardware does not fail.

1oo1 (nonredundant)



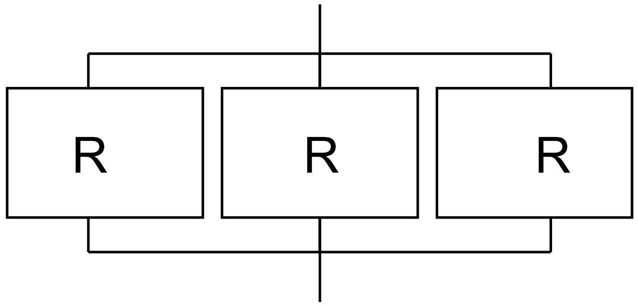
$R_{1oo1} = R$

1oo2 (duplication and error detection)



$R_{1oo2} = 2R – R^2$

2oo3 (triplication and voting)



$R_{2oo3} = 3R^2 – 2R^3$

kooN (k out of N must work)

$$R_{KooN} = \sum_{i=0}^{K} \binom{N}{i} R^i (1 – R)^{N-i}$$

What is better ?

NASA Dryden flight Research Center Photo Collection
http://www.dfrc.nasa.gov/gallery/photo/index.html
NASA Photo: ED03-0152-1 Date: June 7, 2003 Photo By: Carla Thomas

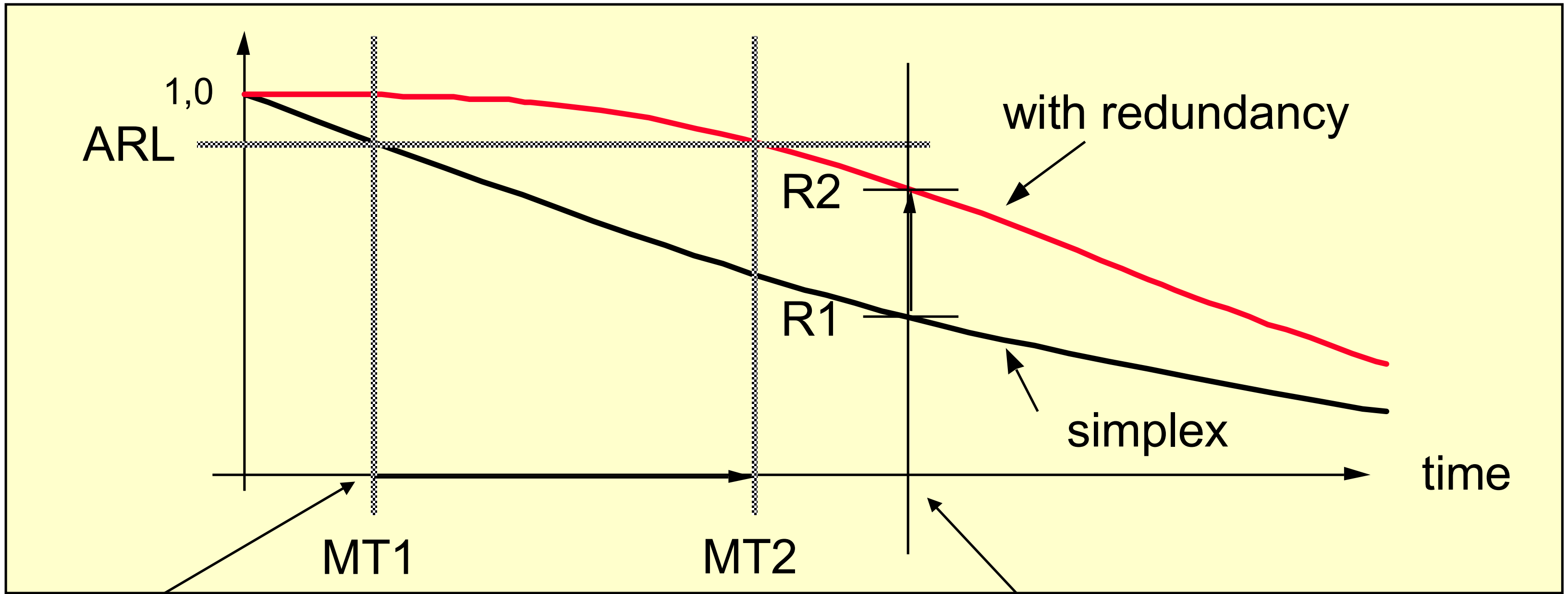
Equipped with an experimental fuel cell system to power the aircraft at night, the solar-electric Helios Prototype is shown during a checkout flight prior to its long-endurance flight demonstration in the summer of 2003.

12 motors, 8 of which are sufficient
to accomplish the mission
(fly 21 days, MTTF = 5'000 h per motor)

2 motors, one of which is sufficient
to accomplish the mission
(fly 21 days, MTTF = 10'000 h per motor)

MIF, ARL, reliability of redundant structures

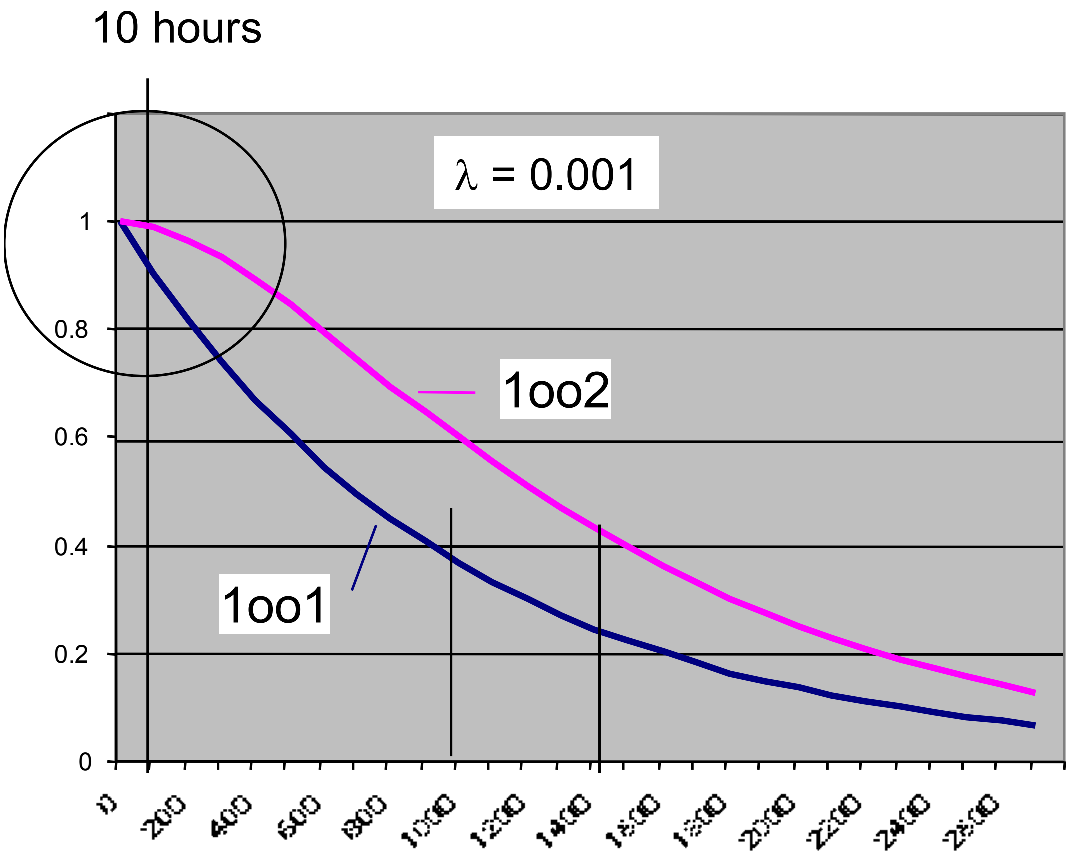
ARL: Acceptable Reliability Level



MIF: Mission Time Improvement Factor (for given ARL)
MIF = MT2/MT1

RIF: Reliability Improvement Factor (at given Mission Time)
RIF = $(1-R_{with}) / (1-R_{without})$ = quotient of unreliability

R1oo2 Reliability Improvement Factor



Reliability improvement factor (RIF)
$= (1-R_{with}) / (1-R_{without})$

but:

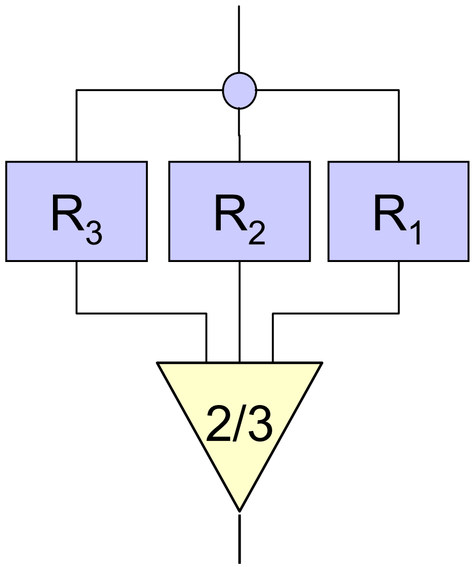
$$MTTF_{1oo2} = \int_0^{\infty} (2 e^{-\lambda t} - e^{-2\lambda t})\, dt = \frac{3}{2\lambda}$$

no spectacular increase in MTTF !

1oo2 only suited when mission time << $1/\lambda$

RIF for 10 hours mission:
R1oo1 = 0.990
R1oo2 = 0.999901
RIF = 100

2 out of 3 without repair - combinatorial solution



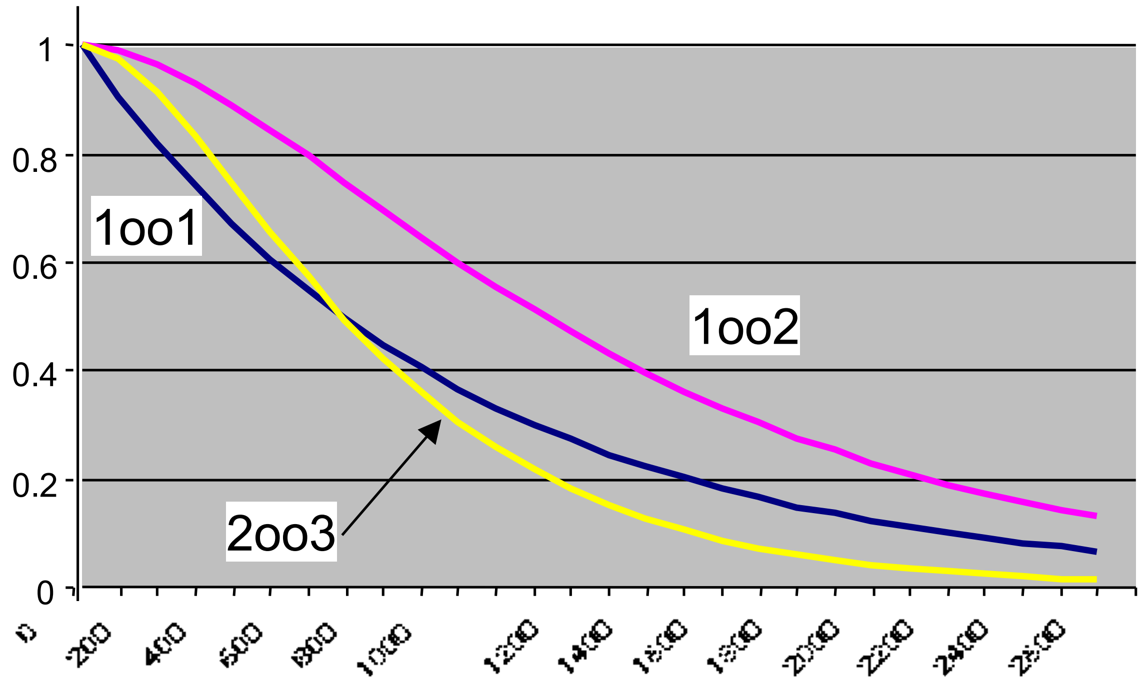
$$R_{2oo3} = 3R^2 - 2R^3 = 3e^{-2\lambda t} - 2e^{-3\lambda t}$$

$$MTTF_{2oo3} = \int_0^{\infty} (3e^{-2\lambda t} - 2\,e^{-3\lambda t})\,dt = \frac{5}{6\lambda}$$

RIF < 1 when t > 0.7 MTTF !

2003 without repair
is not interesting for long mission

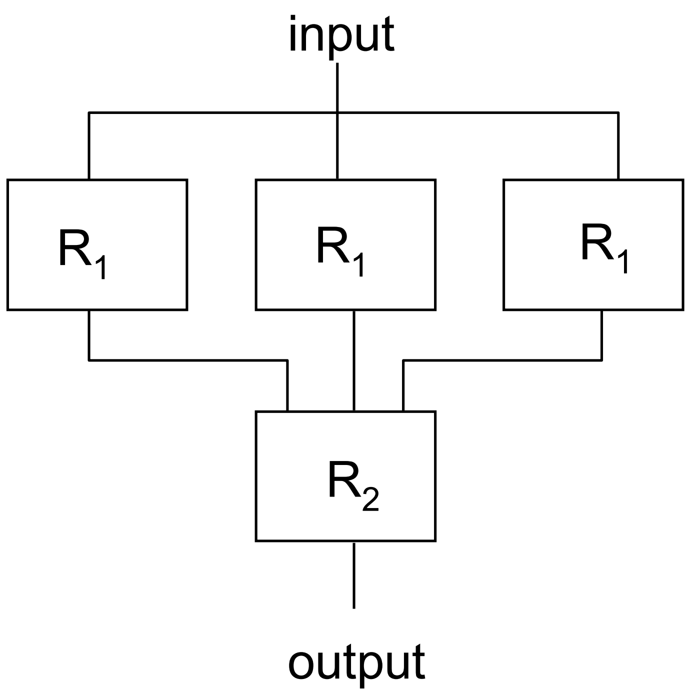
repair is awkward to consider in
combinatorial analysis, another
method - Markov - will be used.



Exercise: 2oo3 considering voter unreliability

Compute the MTTF of the following 2-out-of-3 system with the component failure rates:

- redundant units $\lambda_1 = 0.01\ h^{-1}$
- voter unit $\lambda_2 = 0.001\ h^{-1}$



9.2.3 Considering repair

Repair

Fault-tolerance does not improve reliability under all circumstances.
It is a solution fro short mission duration

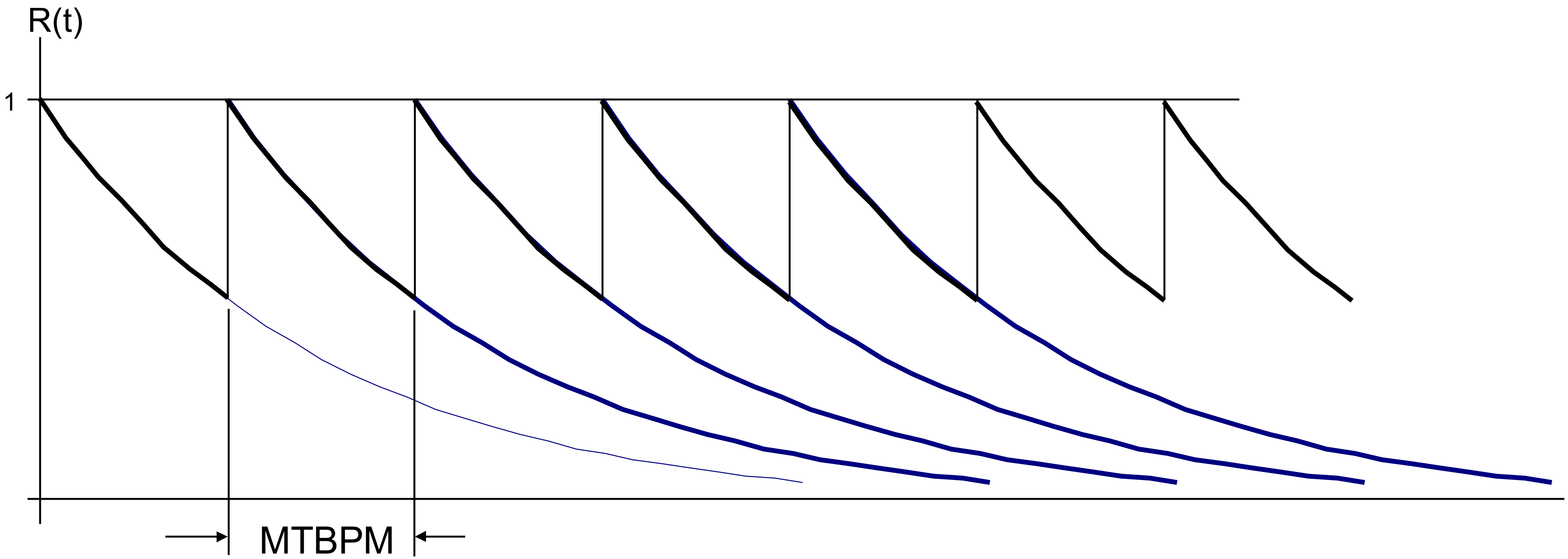
Solution: repair (preventive maintenance, off-line repair, on-line repair)

Example: short Mission time, high MTTF: pilot, co-pilot

long Mission time, low MTTF: how to reach the stars ?
(hibernation, reproduction in space)

Problem: exchange of faulty parts during operation (safety !)
reintegration of new parts,
teaching and synchronization

Preventive maintenance



Mean Time between preventive maintenance

Preventive maintenance reduces the probability of failure, but does not prevent it.
in systems with wear, preventive maintenance prevents aging (e.g. replace oil, filters)
Preventive maintenance is a regenerative process (maintained parts as good as new)

Considering Repair

beyond combinatorial reliability, more powerful tools are required.

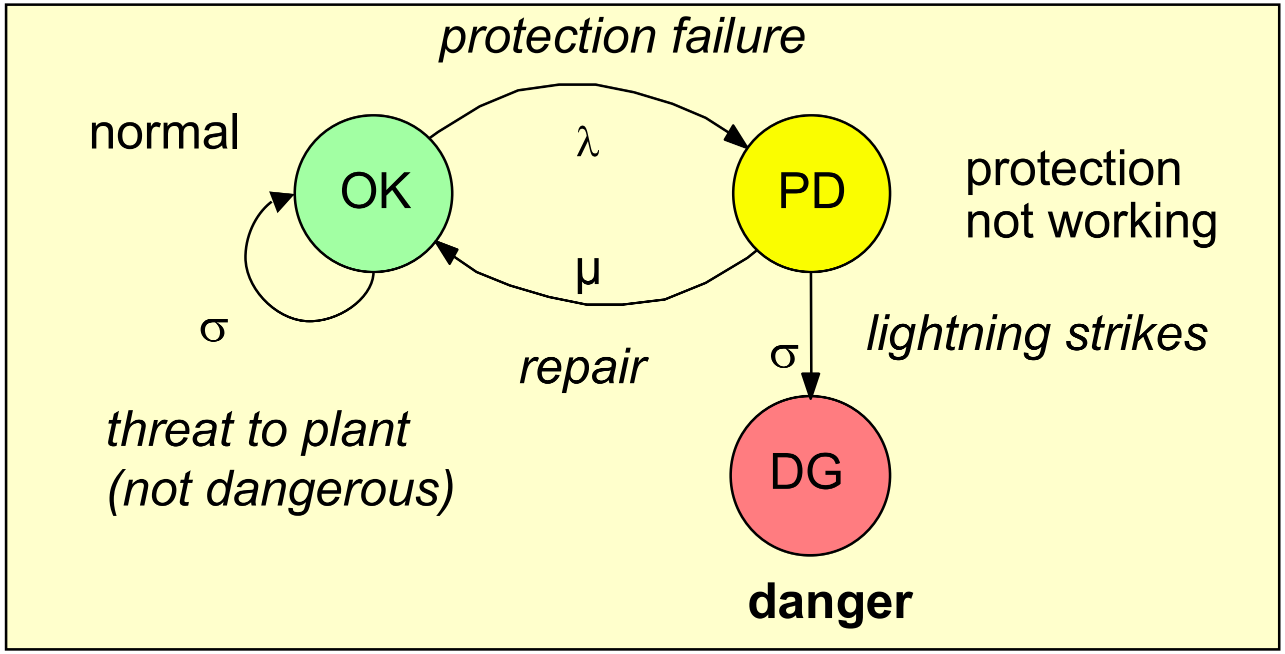
the basic tool is the Markov Chain (or Markov Process)

9.2.4 Markov models

Markov

Define distinct states of the system depending on fault-relevant events

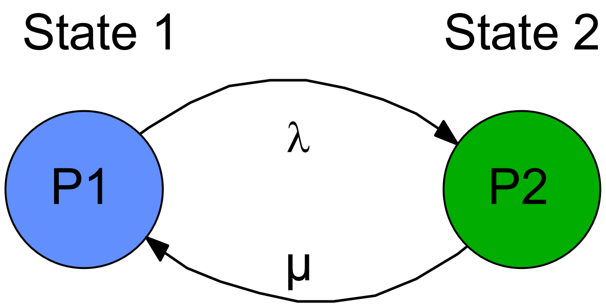
States must be – mutually exclusive
– collectively exhaustive



Let p_i (t) = Probability of being in state S_i at time t -> $\sum_{\text{all states}} p_i(t) = 1$

probability of leaving that state depends only on current state
(is independent of how much time was spent in state, how state was reached)

Continuous Markov Chains



Time is considered continuous.

Instead of transition probabilities, the temporal behavior is given by transition rates (i.e. transition probabilities per infinitesimal time step).

A system will remain in the same state unless going to a different state.

Relationship between state probabilities are modeled by differential equations,

e.g. $dP1/dt = \mu\, P2 - \lambda\, P1$,

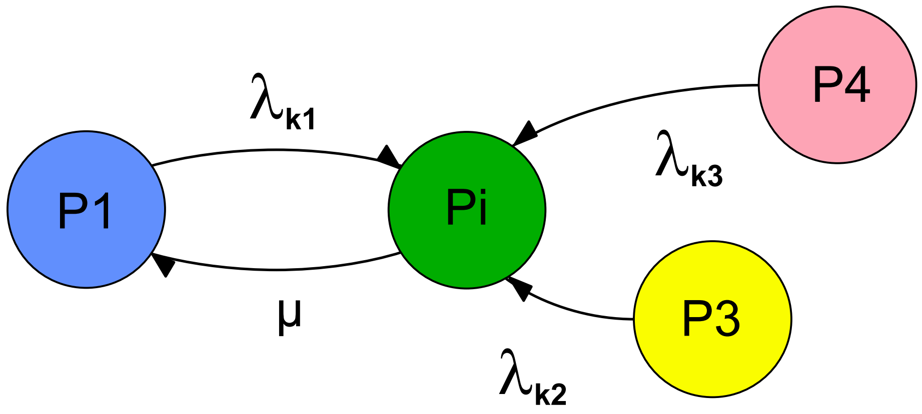
$dP2/dt = \lambda\, P1 - \mu\, P2$

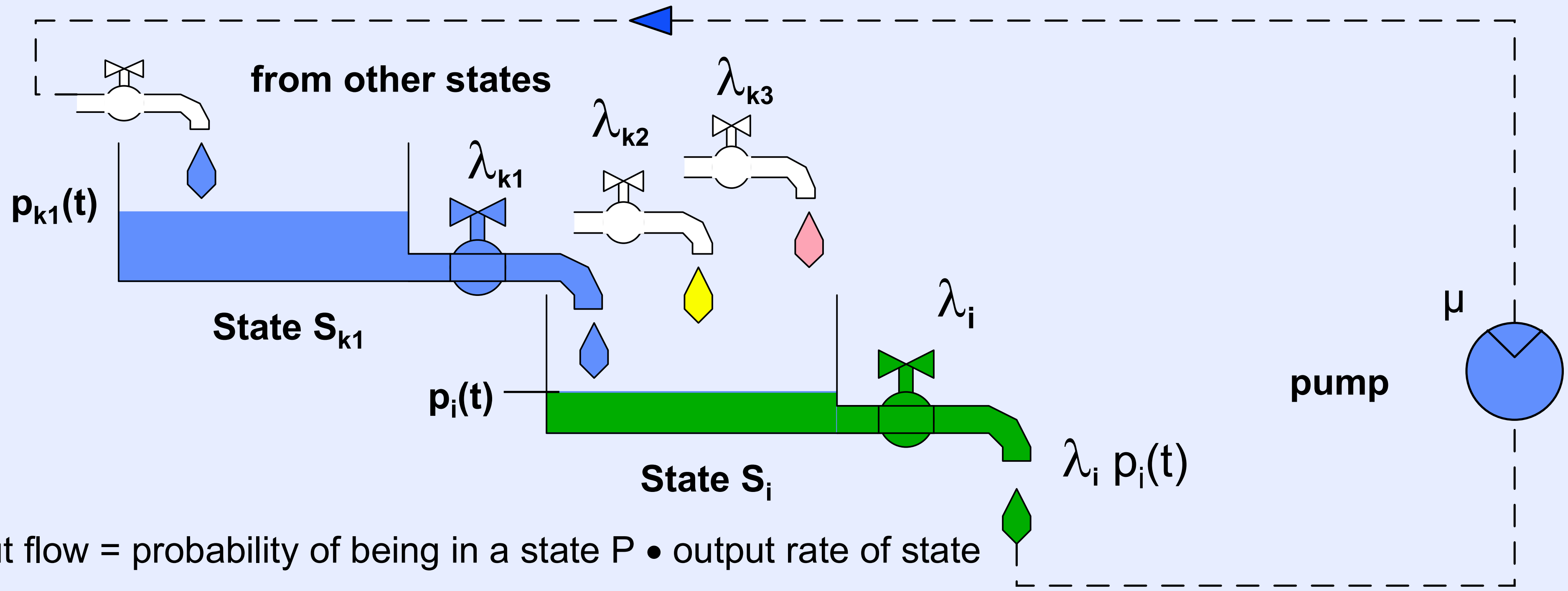
Note: there also exist discrete Markov Chains, in which the time takes discrete steps t = 0, 1, 2, etc., with similar definition

Markov - hydraulic analogy

inflow **outflow**

$$\frac{dp_i(t)}{dt} = \sum \lambda_k \, p_k(t) \; - \; \sum \lambda_i \, p_i(t)$$



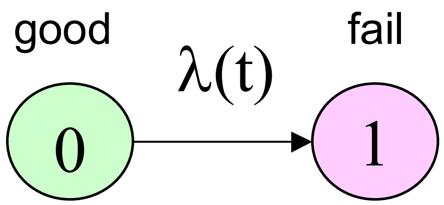


Output flow = probability of being in a state P • output rate of state

Simplification: output rate λ_j = constant (not a critical simplification)

Reliability expressed as state transition

one element:

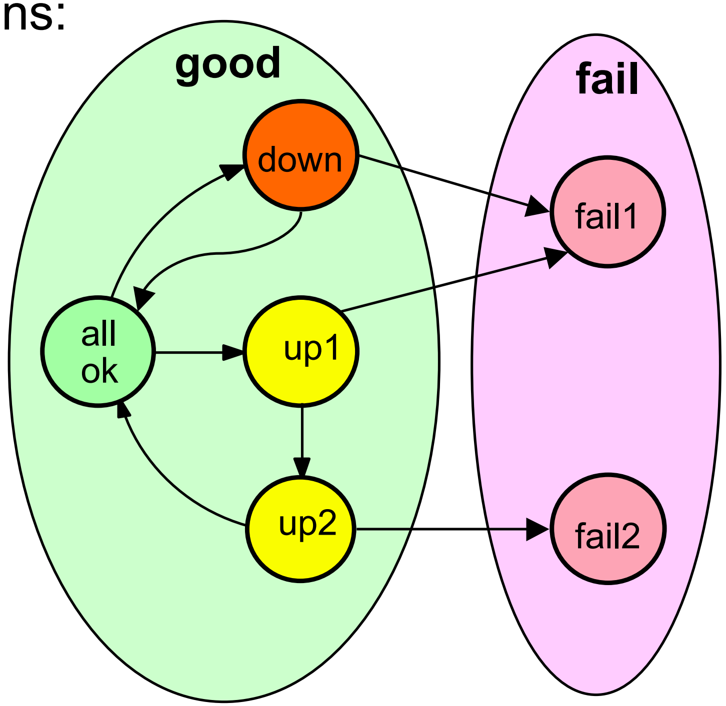


$$\frac{dp_0}{dt} = - \lambda \, p_0$$

$$\frac{dp_1}{dt} = + \lambda \, p_0$$

$$R(t) = p_0 = e^{-\lambda t}$$

arbitrary transitions:

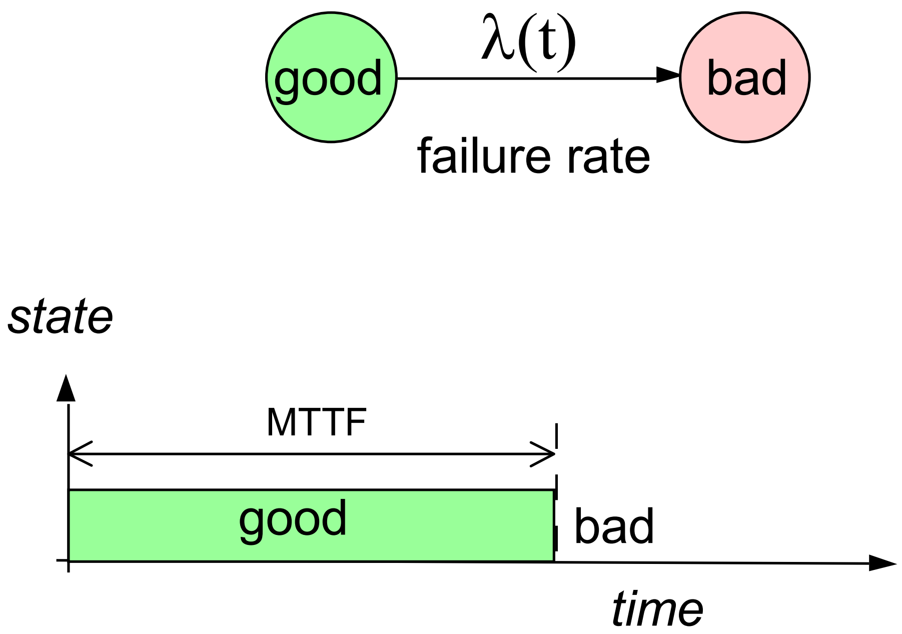


non-terminal states   terminal states

$$R(t) = 1 - (p_{fail1} + p_{fail2})$$

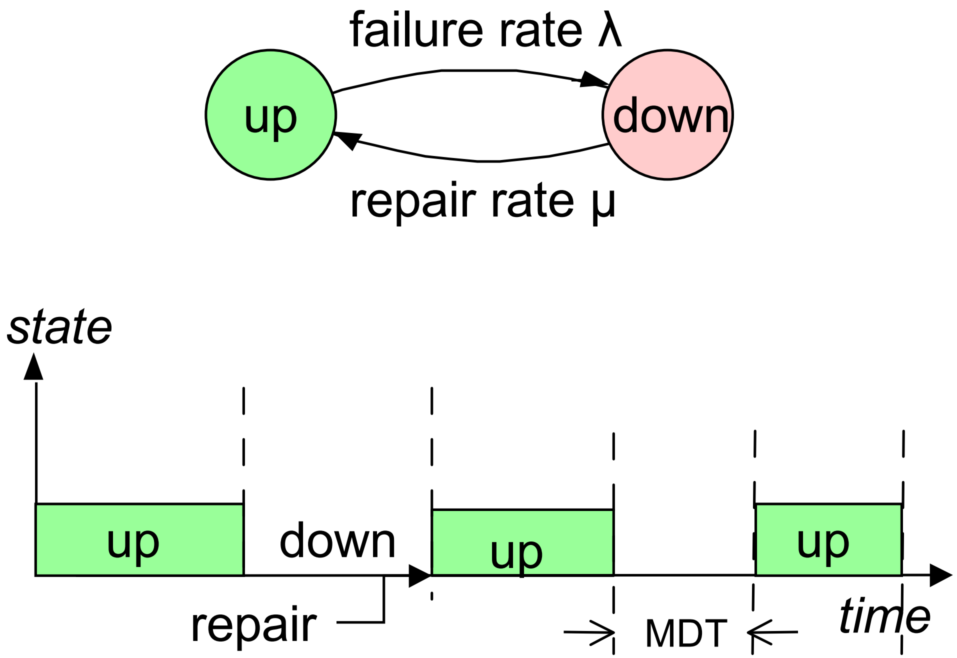
Reliability and Availability expressed in Markov

Reliability



definition: "probability that an item will perform its required function in the specified manner and under specified or assumed conditions *over a given time period*"

Availability

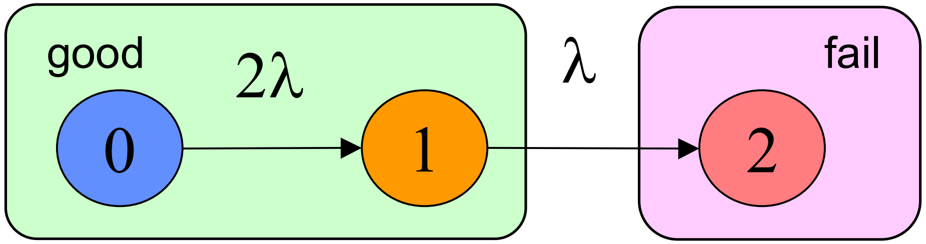


definition: "probability that an item will perform its required function in the specified manner and under specified or assumed conditions *at a given time* "

reliable systems have absorbing states,
they may include repair, but eventually, they will fail

Redundancy calculation with Markov: 1 out of 2 (no repair)

Markov:



λ = constant

What is the probability that system be in state S_0 or S_1 until time t ?

Linear Differential Equation

$$\frac{dp_0}{dt} = -2\lambda\, p_0$$

$$\frac{dp_1}{dt} = +2\lambda\, p_0 - \lambda p_1$$

$$\frac{dp_2}{dt} = \qquad\qquad + \lambda p_1$$

initial conditions:

$p_0(0) = 1$ (initially good)

$p_1(0) = 0$

$p_2(0) = 0$

Solution:

$$p_0(t) = e^{-2\lambda t}$$

$$p_1(t) = 2\, e^{-\lambda t} - 2\, e^{-2\lambda t}$$

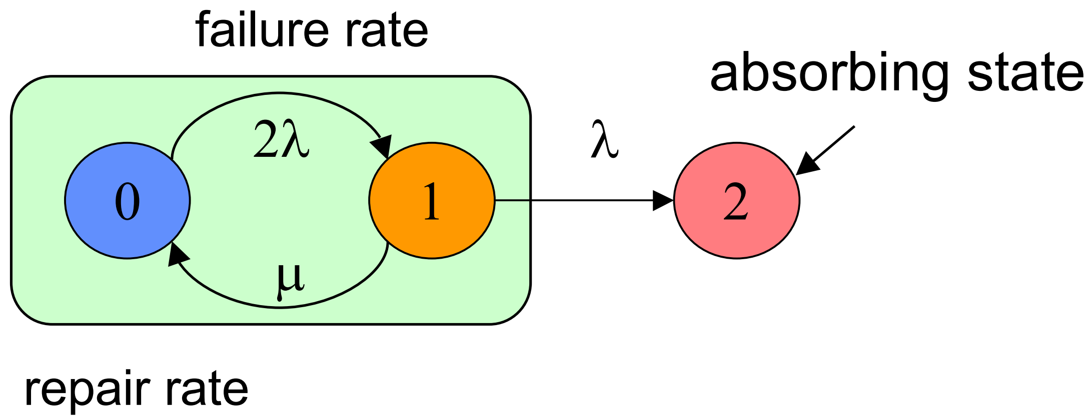
$$R(t) = p_0(t) + p_1(t) = 2\, e^{-\lambda t} - e^{-2\lambda t}$$

(same result as combinatorial - QED)

1out-of-2 with repair (1oo2)

What is the probability that a system fails while one failed element awaits repair ?

Markov:



Linear Differential Equations:

$$\frac{dp_0}{dt} = -2\lambda p_0 + \mu p_1$$

$$\frac{dp_1}{dt} = +2\lambda p_0 - (\lambda+\mu) p_1$$

$$\frac{dp_2}{dt} = + \lambda p_1$$

initial conditions:

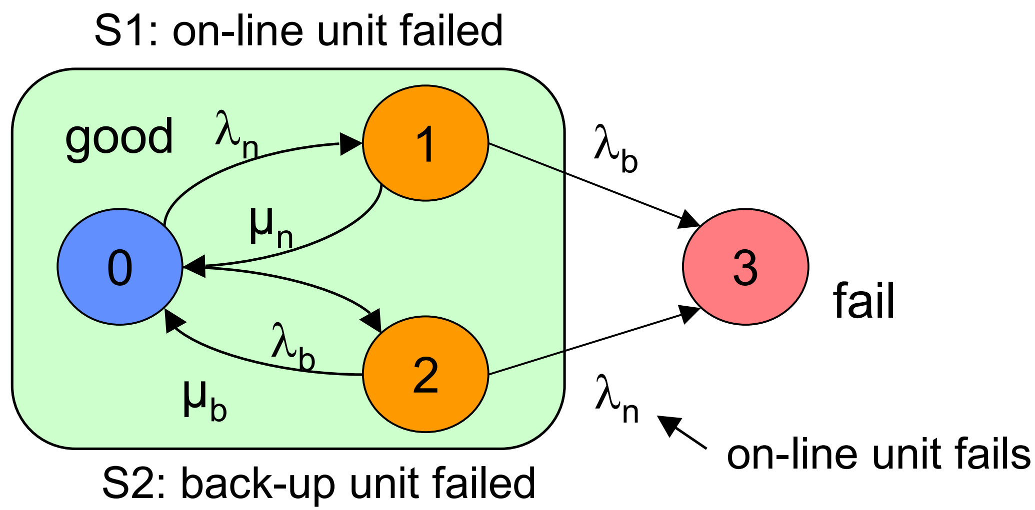
$p_0(0) = 1$ (initially good)

$p_1(0) = 0$

$p_2(0) = 0$

Ultimately , the absorbing states will be “filled”, the non-absorbing will be “empty”.

One or two repair teams...



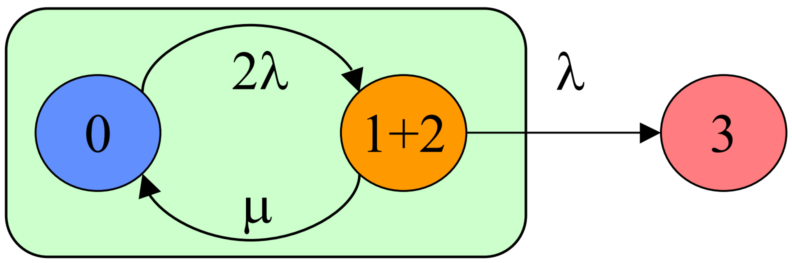
$$\frac{dp_0}{dt} = -2\lambda\, p_0 + \mu\, p_1 + \mu\, p_2$$

$$\frac{dp_1}{dt} = +\lambda\, p_0 - (\lambda+\mu)\, p_1$$

$$\frac{dp_2}{dt} = +\lambda\, p_0 - (\lambda+\mu)\, p_2$$

$$\frac{dp_3}{dt} = +\lambda\, p_1 + \lambda\, p_2$$

is equivalent to:



with $\mu_n = \mu_b$; $\lambda_n = \lambda_b$

$$\frac{dp_0}{dt} = -2\lambda\, p_0 + \mu\, p_1 + \mu\, p_2$$

$$\frac{dp_{1+2}}{dt} = +2\lambda\, p_0 - (\lambda+\mu)\, p_{1+2}$$

$$\frac{dp_3}{dt} = +\lambda\, (p_1+p_2)$$

it is easier to model with a repair team for each failed unit (no serialisation of repair)

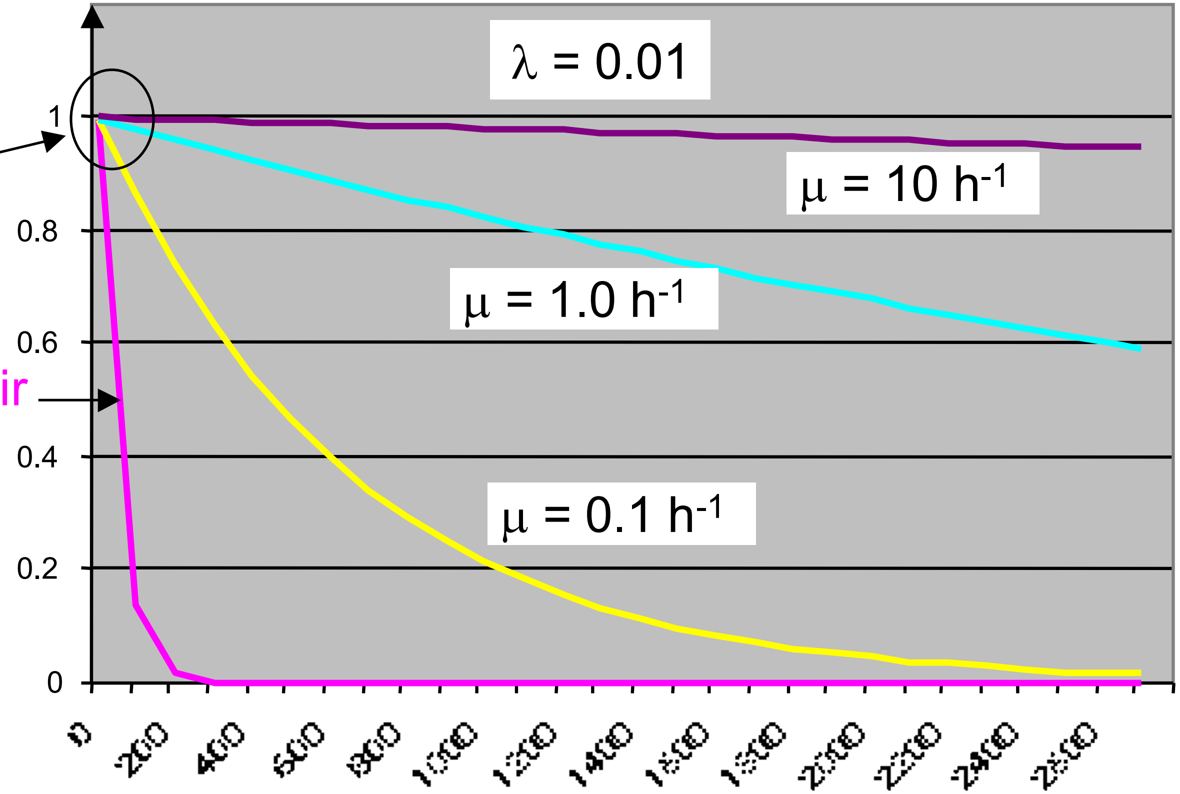
Results: reliability R(t) of 1oo2 with repair rate μ

$$R(t) = P_0 + P_1 = \frac{(3\lambda+\mu)+W}{2W}\, e^{-(3\lambda+\mu-W)\,t} - \frac{(3\lambda+\mu)-W}{2W}\, e^{-(3\lambda+\mu+W)\,t}$$

with:

$$W = \sqrt{\lambda^2 + 6\lambda\mu + \mu^2}$$

we do not consider short mission time



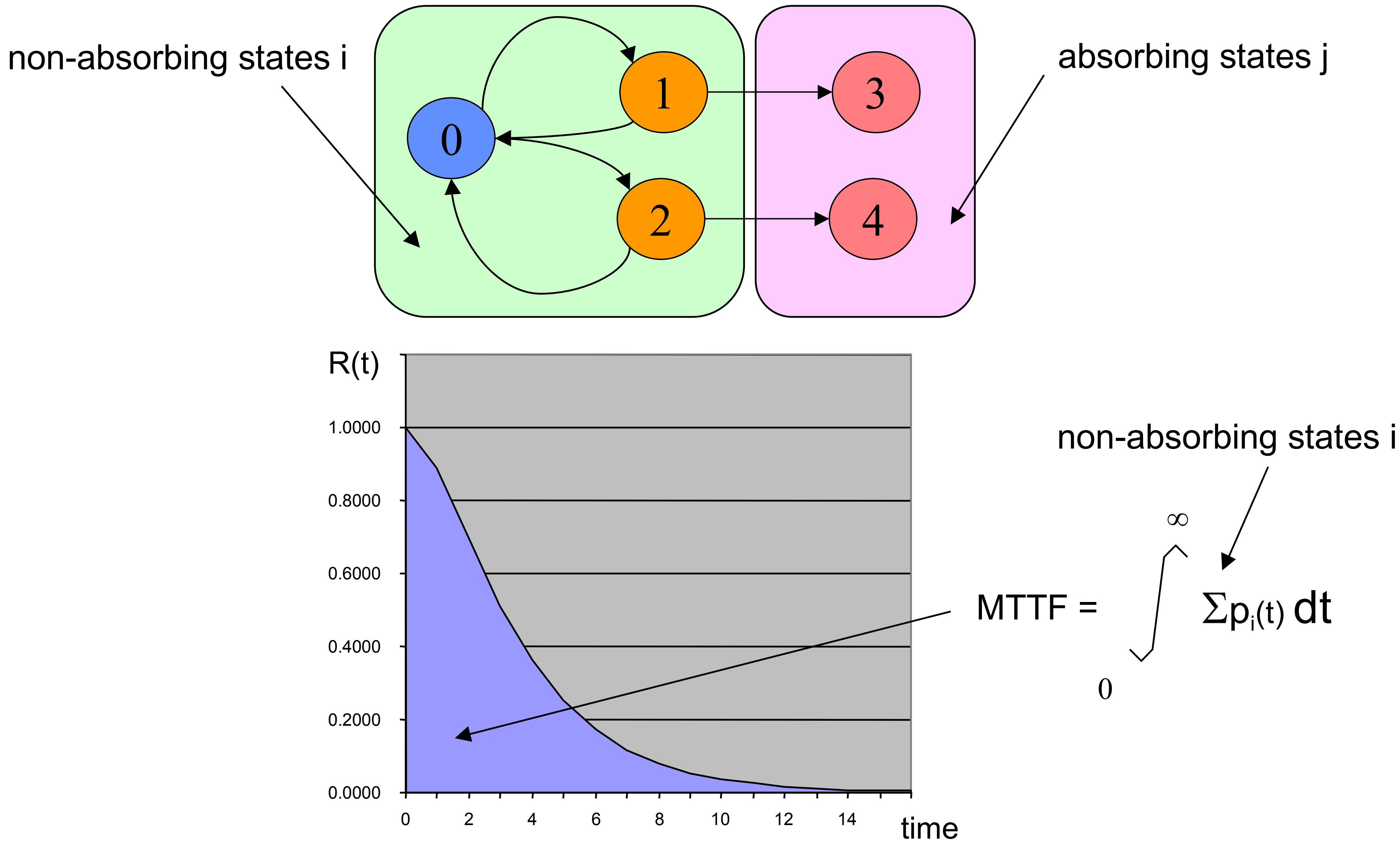
1oo2 no repair

repair does not interrupt mission

Time in hours

R(t) accurate, but not very helpful - MTTF is a better index for long mission time

Mean Time To Fail (MTTF)



$$\text{MTTF} = \int_0^{\infty} \Sigma p_i(t)\, dt$$

MTTF calculation in Laplace (example 1oo2)

Laplace transform
initial conditions:
p_0 (t=0) = 1 (initially good)

$$sP_0(s) - p_0(t=0) = -2\lambda P_0(s) \quad + \mu P_1(s)$$

$$sP_1(s) - 0 = +2\lambda P_0(s) - (\lambda+\mu) P_1(s)$$

$$sP_2(s) - 0 = \quad + \lambda P_1(s)$$

apply boundary theorem

$$\lim_{t \to \infty} \int_0^{\infty} p(t)\,dt = \lim_{s \to 0} s\,P(s)$$

only include non-absorbing states
(number of equations =
number of non-absorbing states)

$$-1 = -2\lambda P_0 \quad + \mu P_1$$

$$0 = +2\lambda P_0 \quad - (\lambda+\mu) P_1$$

solution of linear equation system:

$$\text{MTTF} = P_0 + P_1 = \frac{(\mu+\lambda)}{2\lambda^2} + \frac{1}{\lambda} = \frac{\mu/\lambda + 3}{2\lambda}$$

General equation for calculating MTTF

1) Set up differential equations

2) Identify terminal states (absorbing)

3) Set up Laplace transform for the non-absorbing states

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ .. \end{bmatrix} = M \, \overline{P}_{na}$$

the degree of the equation is equal to the number of non-absorbing states

4) Solve the linear equation system

5) The MTTF of the system is equal to the sum of the non-absorbing state integrals.

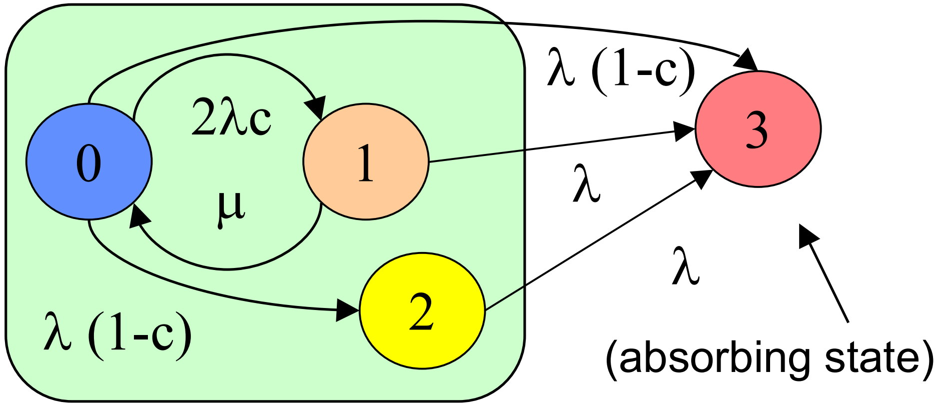
6) To compute the probability of not entering a certain state, assign a dummy (very low) repair rate to all other absorbing states and recalculate the matrix

Correct diagram for 1oo2

Consider that the failure rate of a device in a 1oo2 system is divided into two failure rates:

1) a benign failure, immediately discovered with probability c
   - if device is on-line, switchover to the stand-by device is successful and repair called
   - if device is on stand-by, repair is called

2) a malicious failure, which is not discovered, with probability (1-c)
   - if device is on-line, switchover to the standby device fails, the system fails
   - if device is on stand-by, switchover will be unsuccessful should the online device fail



1: on-line fails, fault detected (successful switchover and repair) or standby fails, fault detected, successful repair

2: standby fails, fault not detected

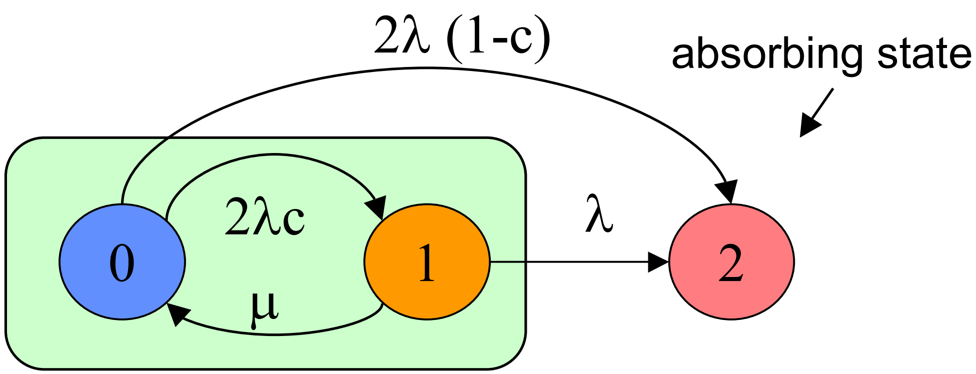
3: both fail, system down

$$
\begin{aligned}
1 &= -2\lambda P_0 + \mu P_1 \\
0 &= +2\lambda c\, P_0 - (\lambda+\mu) P_1 \\
0 &= +\lambda(1-c)\, P_0 - \lambda P_2
\end{aligned}
$$

$$
\text{MTTF} = \frac{(2+c) + \mu/\lambda\,(2-c)}{2\,(\lambda + \mu\,(1-c))}
$$

Approximation found in the literature

This simplified diagram considers that the undetected failure of the spare causes immediately a system failure



$$-1 = -2\lambda P_0 + \mu P_1$$

$$0 = +2\lambda c P_0 - (\lambda+\mu) P_1$$

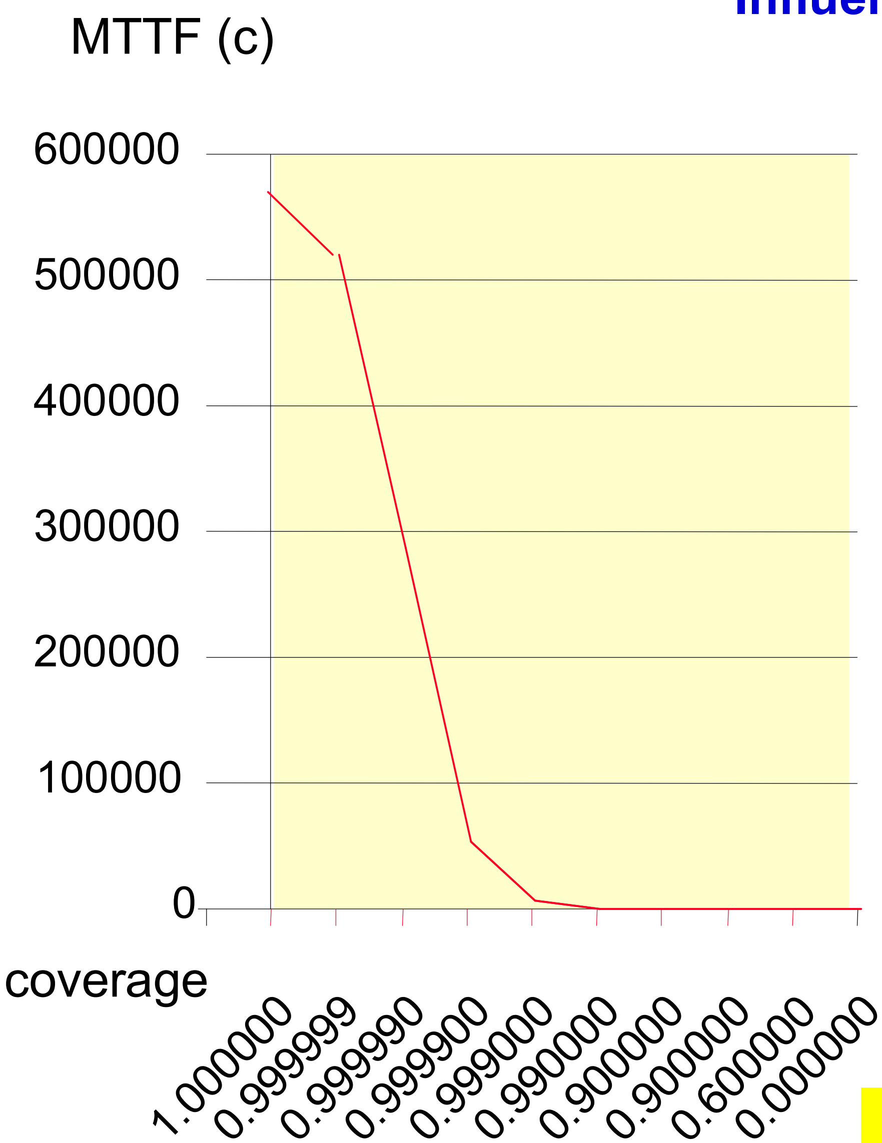
$$\cancel{0 = +2\lambda(1-c) P_0 + \lambda P_1}$$

applying Markov:

$$\text{MTTF} = \frac{(1+2c) + \mu/\lambda}{2(\lambda + \mu(1-c))}$$

The results are nearly the same as with the previous four-state model…

Influence of coverage (2)



Example:

$\lambda = 10^{-5}\ h^{-1}$ (MTTF = 11.4 year),
$\mu = 1\ hour^{-1}$
MTTF with perfect coverage = 570468 years

When coverage falls below 60%, the redundant (1oo2) system performs no better than a simplex one !

Therefore, coverage is a critical success factor for redundant systems !

In particular, redundancy is useless if failure of the spare remains undetected (lurking error).

$$\lim_{\mu \to 0} MTTF = \frac{1}{\lambda} \left(\frac{3}{2} + \frac{\mu}{2\lambda} \right)$$

$$\lim_{\lambda/\mu \to 0} MTTF = \frac{1}{\lambda\ (1-c)}$$

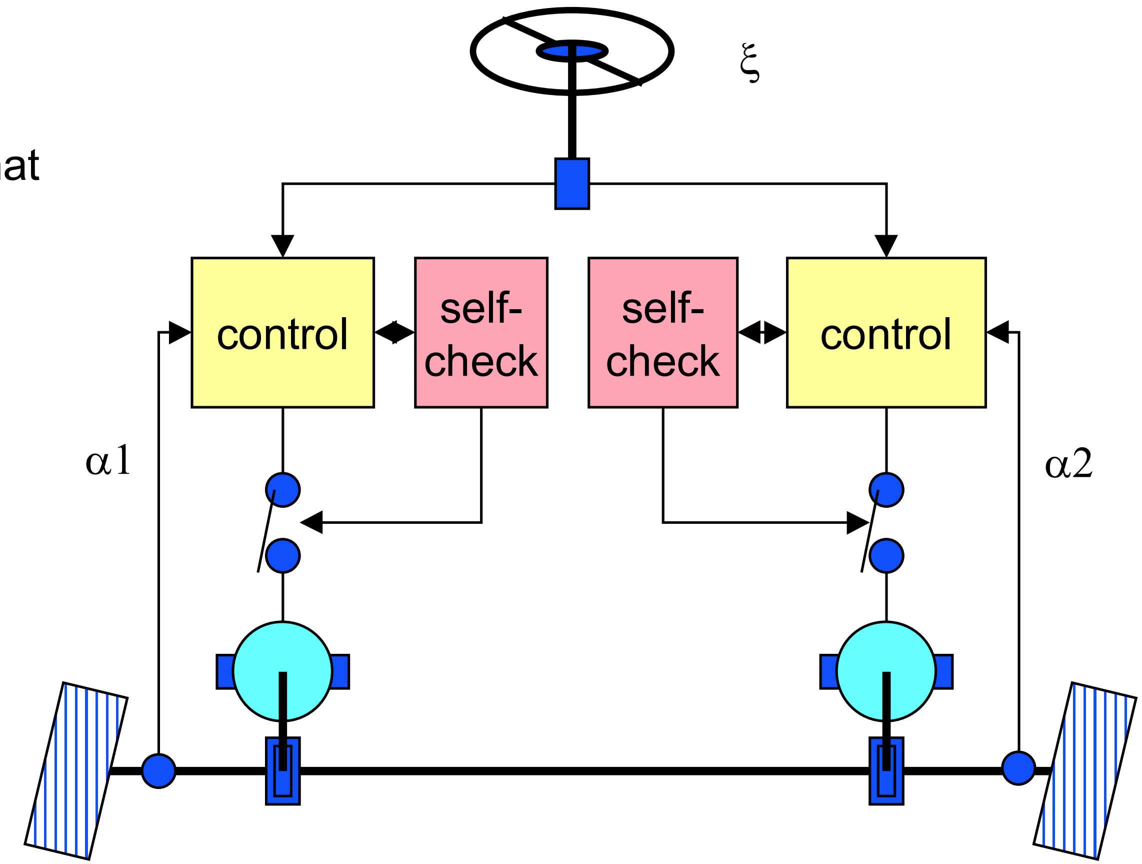
Application: 1oo2 for drive-by-wire

coverage is assumed to be the probability that self-check detects an error in the controller.

when self-check detects an error, it passivates the controller (output is disconnected) and the other controller takes control.

one assumes that an accident occurs if both controllers act differently, i.e. if a computer does not fail to silent behaviour.

Self-check is not instantaneous, and there is a probability that the self-check logic is not operational, and fails in underfunction (overfunction is an availability issue)



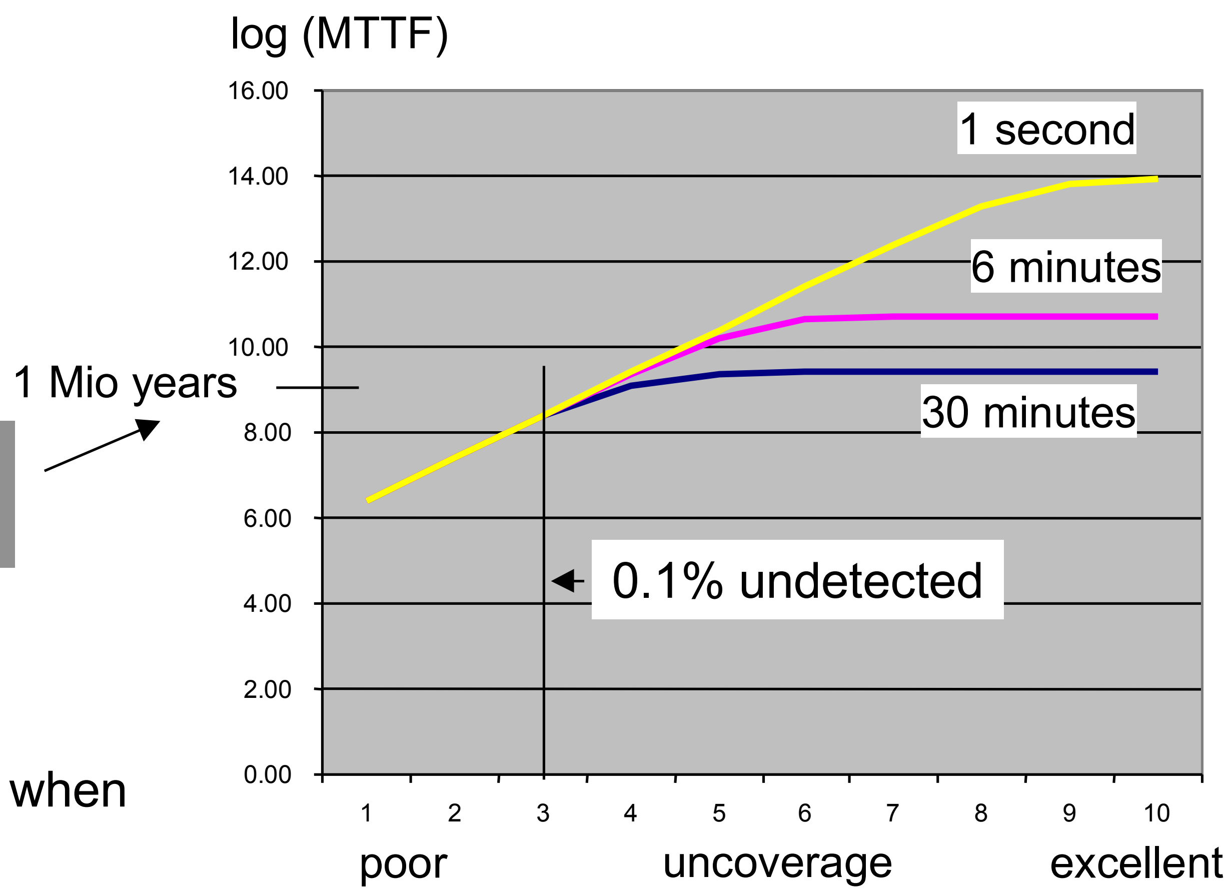
Results 1oo2c, applied to drive-by-wire

λ = reliability of one chain (sensor to brake) = 10^{-5} h^{-1} (MTTF = 10 years)

c = coverage: variable (expressed as uncoverage: 3nines = 99.9 % detected)

μ = repair rate = parameter
- 1 Second: reboot and restart
- 6 Minutes: go to side and stop
- 30 Minutes: go to next garage

or once per year on a million vehicles

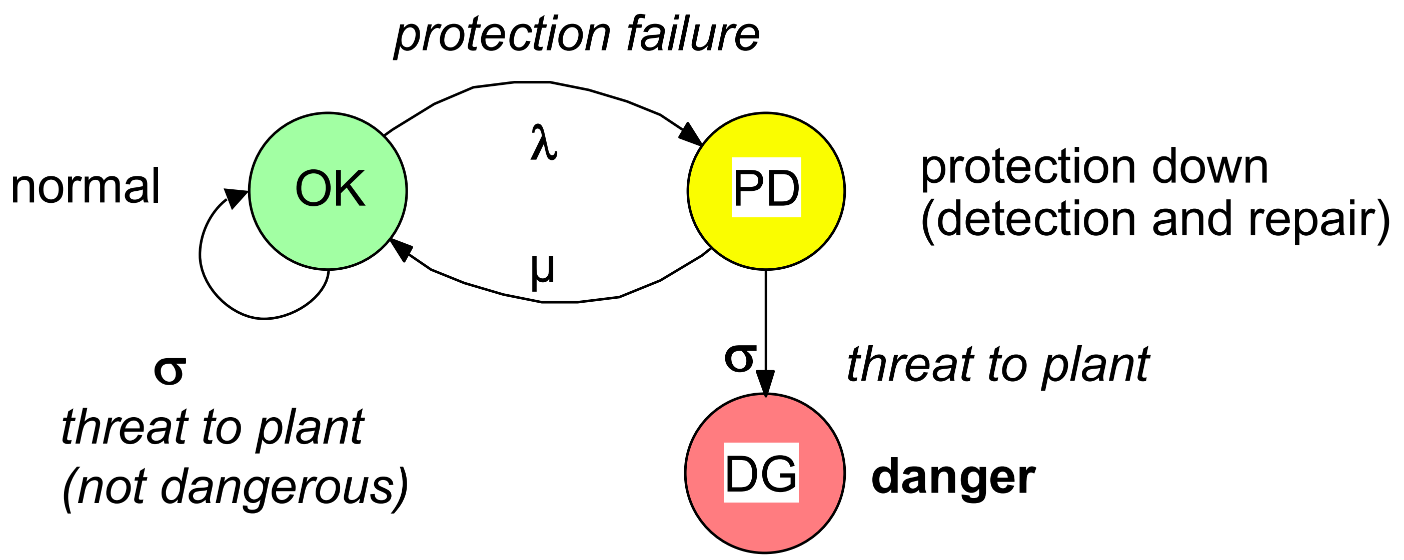


conclusion:
the repair interval does not matter when coverage is poor

Protection system (general)

In protection systems, the dangerous situation occurs when the plant is threatened (e.g. short circuit) and the protection device is unable to respond.

The threat is a stochastic event, therefore it can be treated as a failure event.



The repair rate μ includes the detection time t !
This impacts directly the maintenance rate.
What is an acceptable repair interval ?

Note: another way to express the reliability of a protection system will be shown under “availability”

Protection system: how to compute test intervals

$\lambda 1$ = overfunction of protection

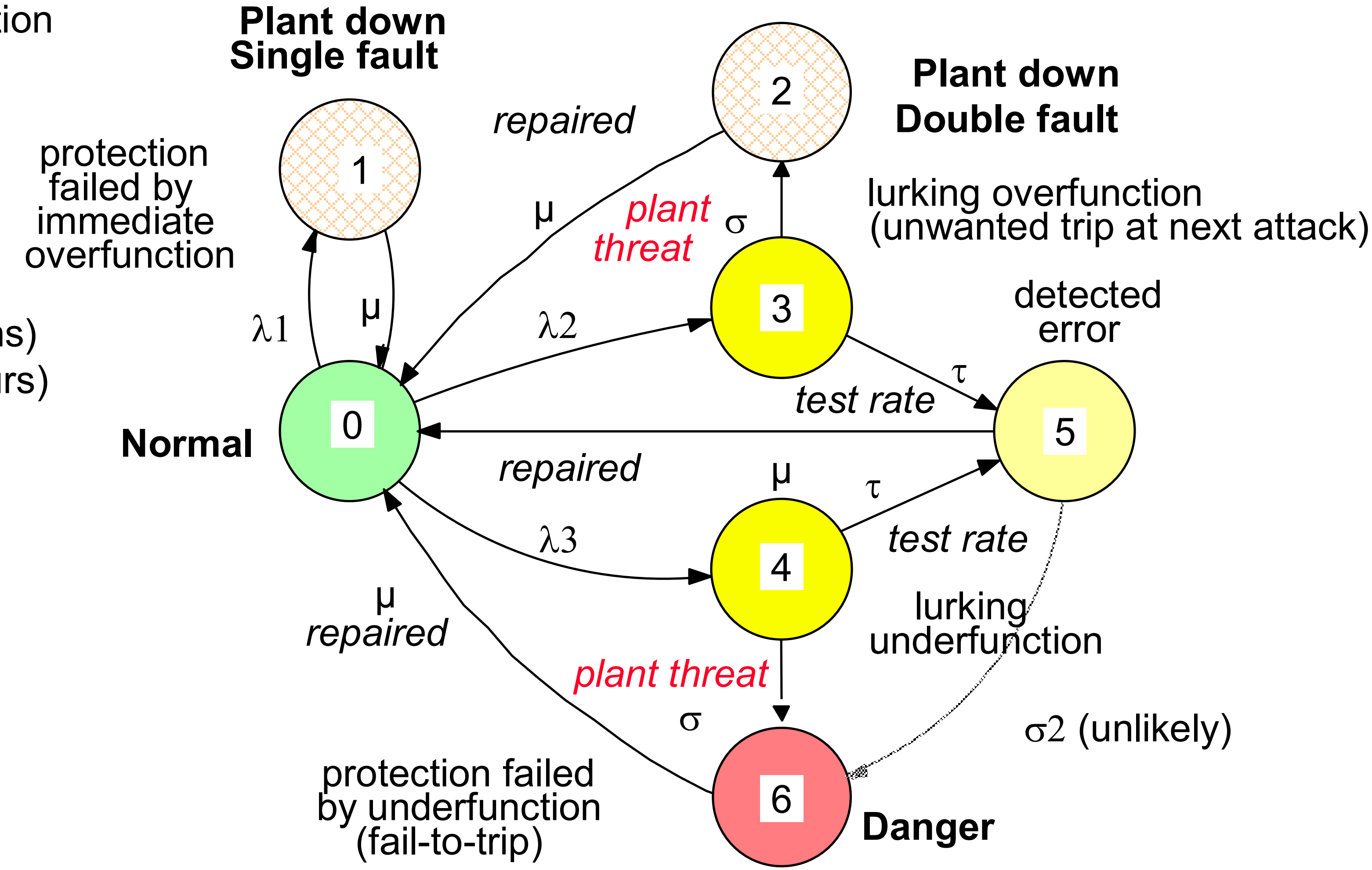
$\lambda 2$ = lurking overfunction

$\lambda 3$ = lurking underfunction

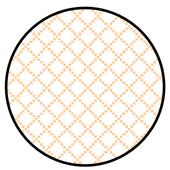
σ = plant suffers attack

τ = test rate (e.g. 1/6 months)

μ = repair rate (e.g. 1/8 hours)



unavailable
states



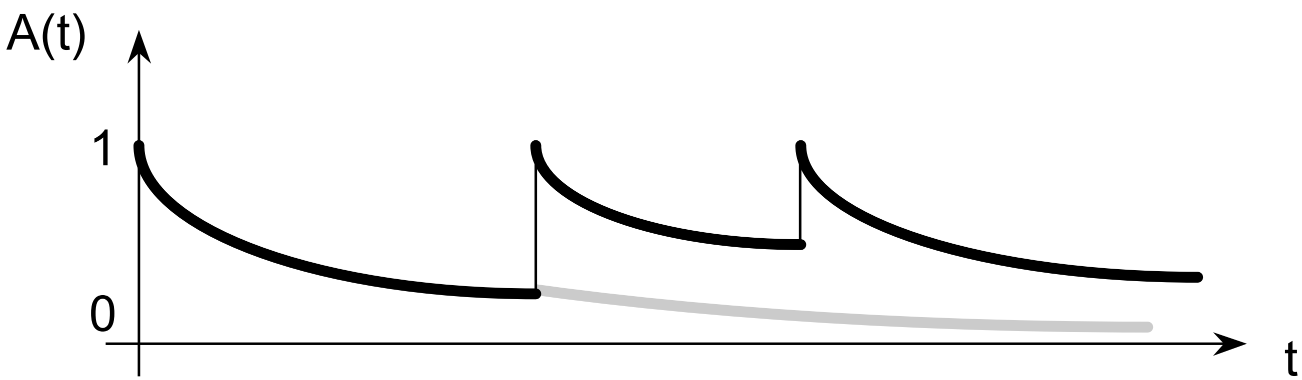
since there exist back-up protection systems, utilities are more concerned by non-productive states

9.2.5 Availability evaluation

Punctual and Stationary Availability

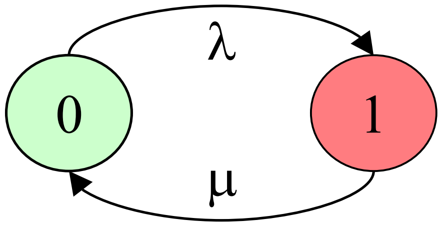
Punctual availability: Probability that a system works at a time t (with repair):

$R(t) \leq A(t)$ due to repair or preventive maintenance
(exchange parts that did not yet fail)



$$\text{Stationary availability } A = \frac{MTTF}{MTTF + MTTR} \quad \text{over the lifetime}$$

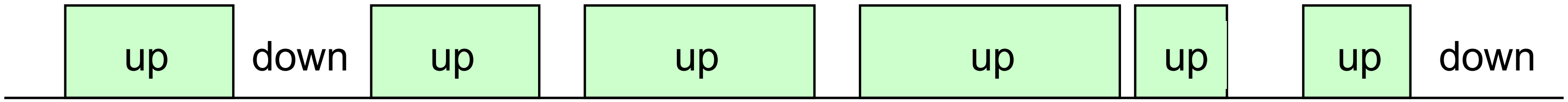
Availability



Availability expresses how often a piece of (repairable) equipment is functioning
The answer depends on failure rate λ and repair rate μ.

Punctual availability (is the system working at time t) is not relevant for most processes.

Stationary availability (duty cycle) impacts financial results



$$A_\infty = \text{availability} = \lim_{t \to \infty} \frac{?\ \text{up times}}{?\ (\text{up times + down times})}$$

Availability is often expressed by its complement, U = unavailability
(e.g. 5 minutes downtime per year = availability is 0.999%)

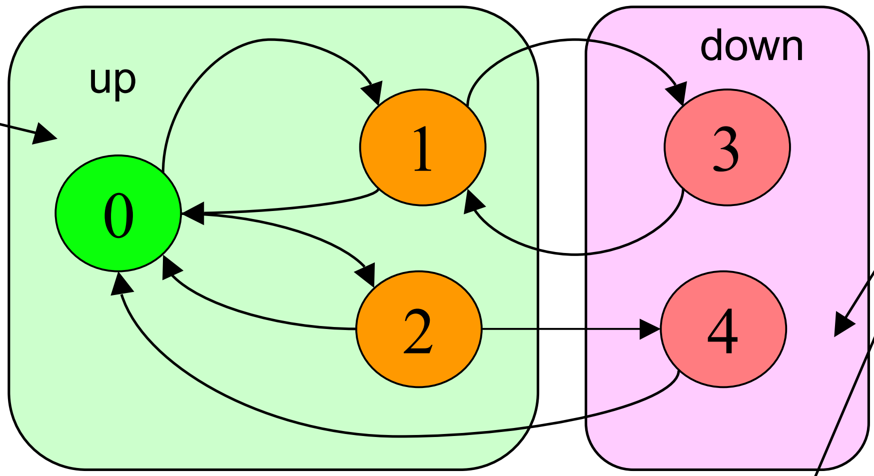
Examples of availability requirements

substation automation	> 99,95%	4 hours per year
telecom power supply	$5 * 10^{-7}$	15 seconds per year

Availability expressed in Markov states

up states i

down states j (non-absorbing)



Availability = $\Sigma p_i(t = \infty)$

Unavailability = $\Sigma p_j\ (t = oo)$

Computation of Availabilities

Divide states into two sets: UP states (system works) and DOWN states (system doesn't work).

The stationary availability is given by the formula A = MTTF / (MTTF + MTTR).

The MTTR is given by the inverse of the repair rate, MTTR = $1/\mu$, to get the system back from the set of **down** states to the set of **up** states.

The MTTF is given by the following set of equations:

$$MTTF(i) = 1/\rho i + \sum (\rho ij / \rho i)\ MTTF(j)$$

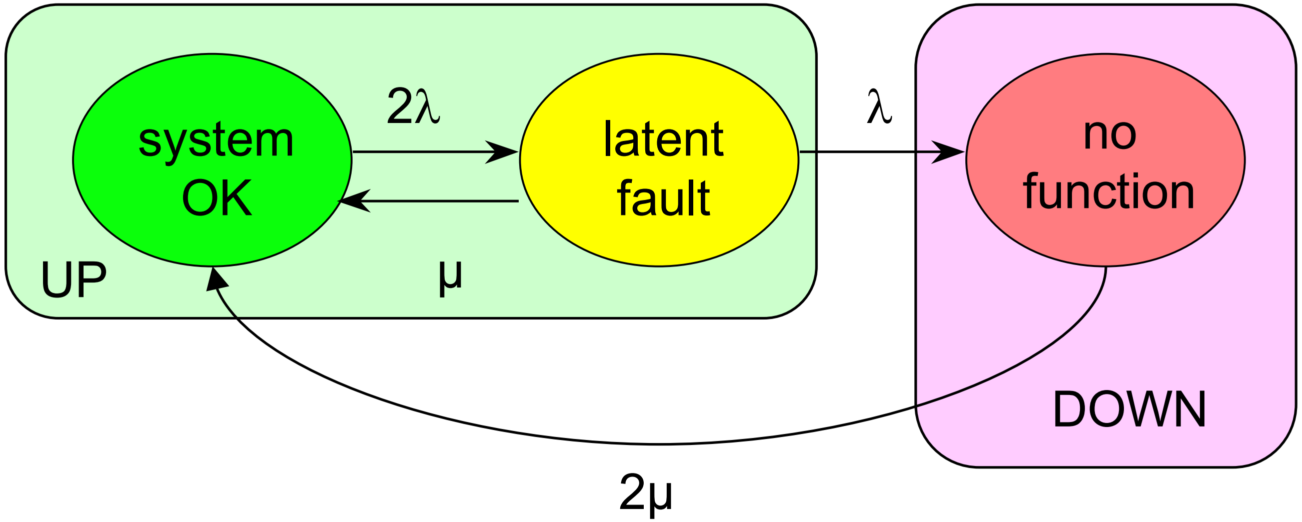
where i, j denote states, ρij is the transition rate from state i to state j, $\rho ii = 0$, the sum is taken over all states j which belong to the set of UP states and

$\rho i = \sum \rho ij$

and MTTF = MTTF(i) if i is the initial state in which the system starts

Example: 1 out of 2 System (1oo2)

The system works if one out of two components (each with failure rate λ) works
each of the components is repaired with a repair rate μ.



Assuming the system is originally in the "OK" state:

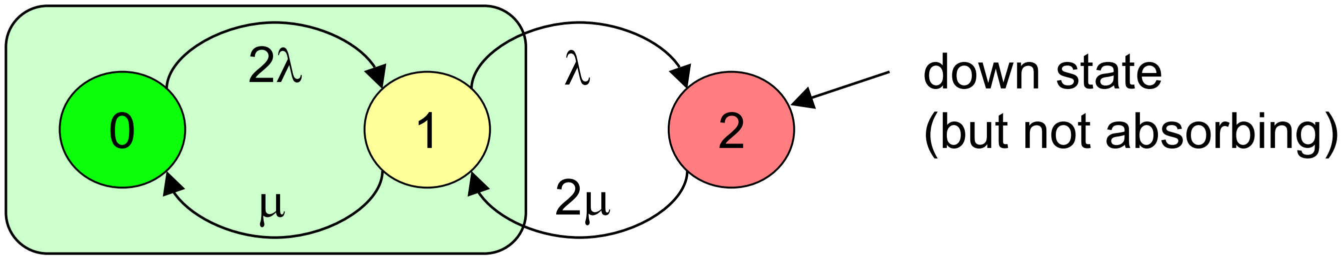
a) Compute the MTTF of the system

b) Compute the availability of the system.

c) Compute the MTBR (mean time between repairs) of the system

Available 1oo2 (1 out-of-2)

Markov states:



assumption: devices can be repaired independently (little impact when $\lambda << \mu$)

$$\frac{dp_0}{dt} = -2\lambda\, p_0 + \mu p_1$$

$$\frac{dp_1}{dt} = +2\lambda\, p_0 - (\lambda+\mu)\, p_1 + 2\mu\, p_2$$

$$\frac{dp_2}{dt} = + \lambda p_1 - 2\mu\, p_2$$

stationary state: $\lim_{t \to \infty} \frac{dp_0}{dt} = \frac{dp_1}{dt} = \frac{dp_2}{dt} = 0$

due to linear dependency add condition: $p_0 + p_1 + p_2 = 1$

$$A = \frac{1}{1 + \dfrac{2\lambda^2}{\mu^2 + 2\lambda\mu}}$$

unavailability $U = (1 - A) = \lim_{U<<1} \dfrac{2\lambda^2}{\mu^2 + 2\lambda\mu}$

Availability calculation

1) Set up differential equations for all states

2) Identify up and down states (no absorbing states allowed !)

3) Remove one state equation save one (arbitrary, for numerical reasons take unlikely state)

4) Add as first equation the precondition: 1 = ? p (all states)

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ .. \end{bmatrix} = M \, \overline{P}_{all}$$

5) The degree of the equation is equal to the number of states

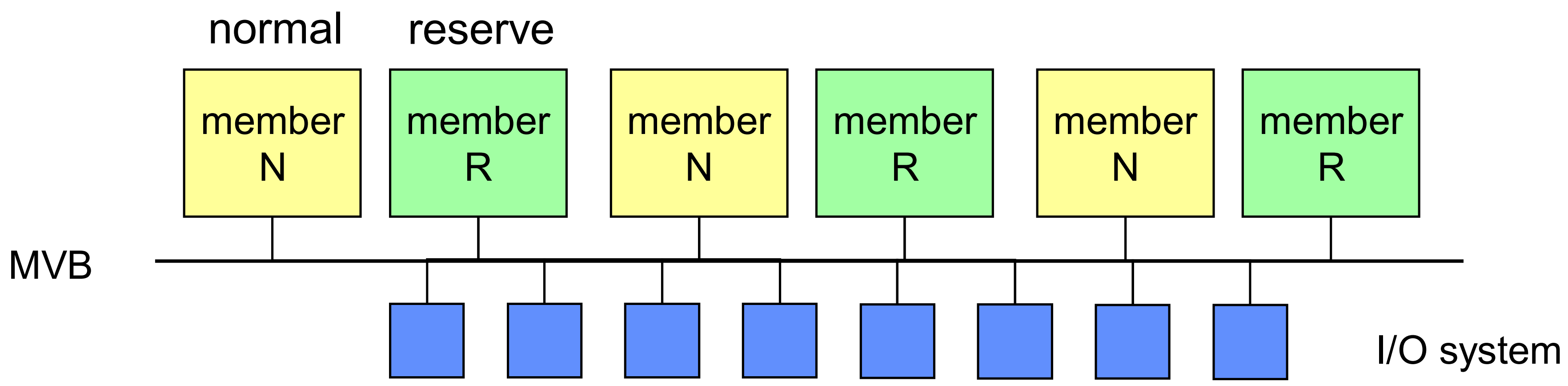
6) Solve the linear equation system, yielding the % of time each state is visited

7) The unavailability is equal to the sum of the down states

We do not use Laplace for calculating the availability !

9.2.6 Examples

Case study: Swiss Locomotive 460 control system availability



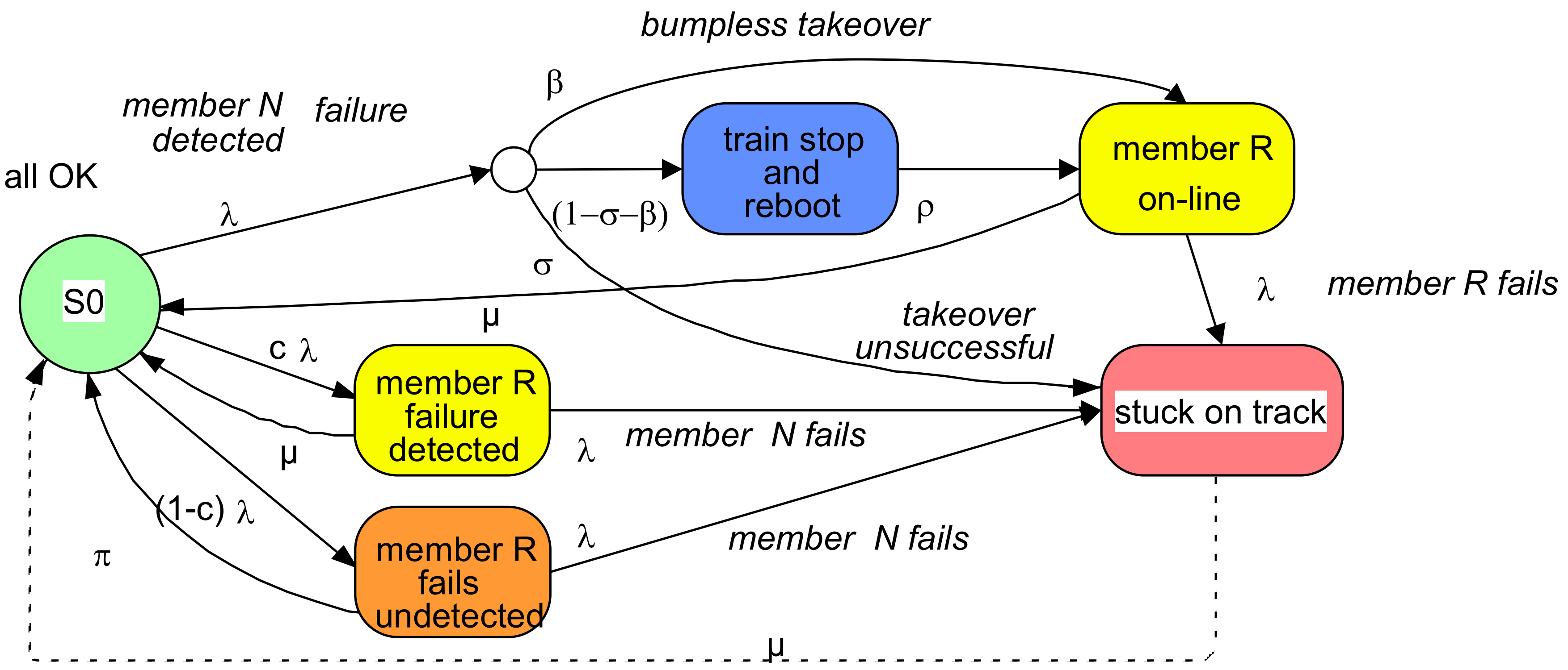
Assumption: each unit has a back-up unit which is switched on when the on-line unit fails

The error detection coverage c of each unit is imperfect

The switchover is not always bumpless - when the back-up unit is not correctly actualized, the main switch trips and the locomotive is stuck on the track

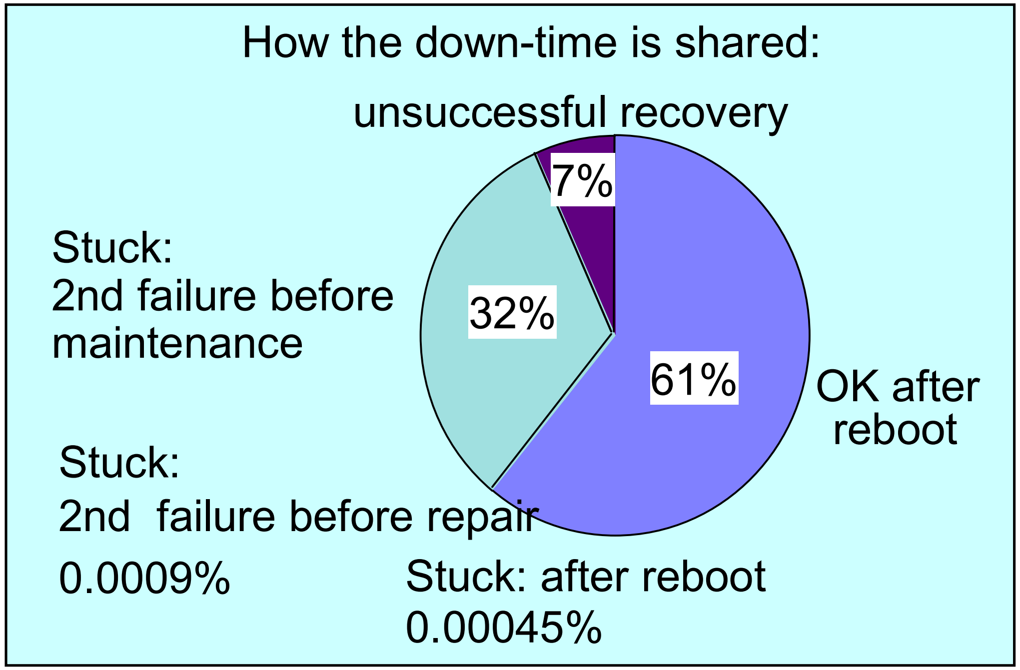
What is the probability of the locomotive to be stuck on track ?

Markov model: SBB Locomotive 460 availability



λ	probability that member N or member R fails	$\lambda = 10^{-4}$	(MTTF is 10000 hours or 1,2 years)
μ	mean time to repair for member N or member P	$\mu = 0.1$	(repair takes 10 hours, including travel to the works)
c	probability of detected failure (coverage factor)	$c = 0.9$	(probability is 9 out of 10 errors are detected)
β	probability of bumpless recovery (train continues)	$\beta = 0.9$	(probability is that 9 out of 10 take-over is successful)
σ	probability of unsuccessful recovery (train stuck)	$\sigma = 0.01$	(probability is 1 failure in 100 cannot be recovered)
ρ	time to reboot and restart train	$\rho = 10$	(mean time to reboot and restart train is 6 minutes)
π	periodic maintenance check	$\pi = 1/8765$	(mean time to periodic maintenance is one year).

SBB Locomotive 460 results

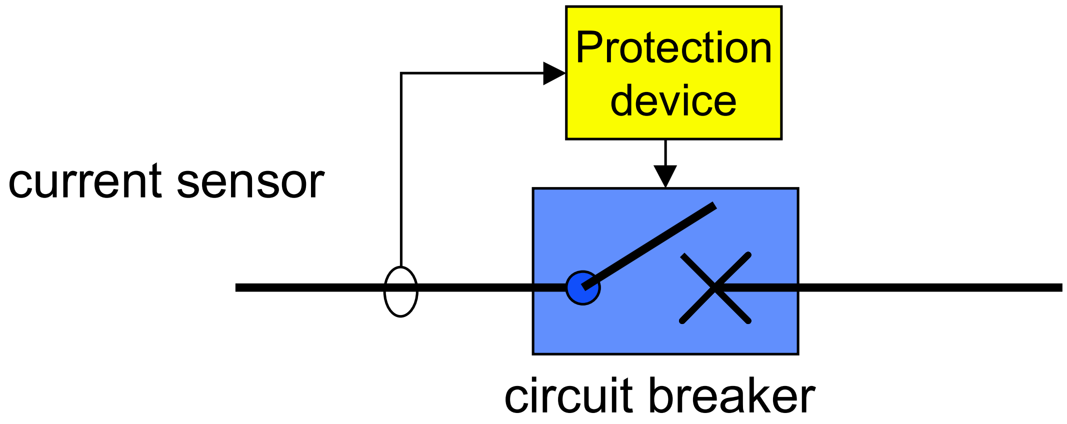


Under these conditions:

unavailability will be **0.5 hours a year**.
stuck on track is once every **20 years**.
recovery will be successful **97%** of the time.

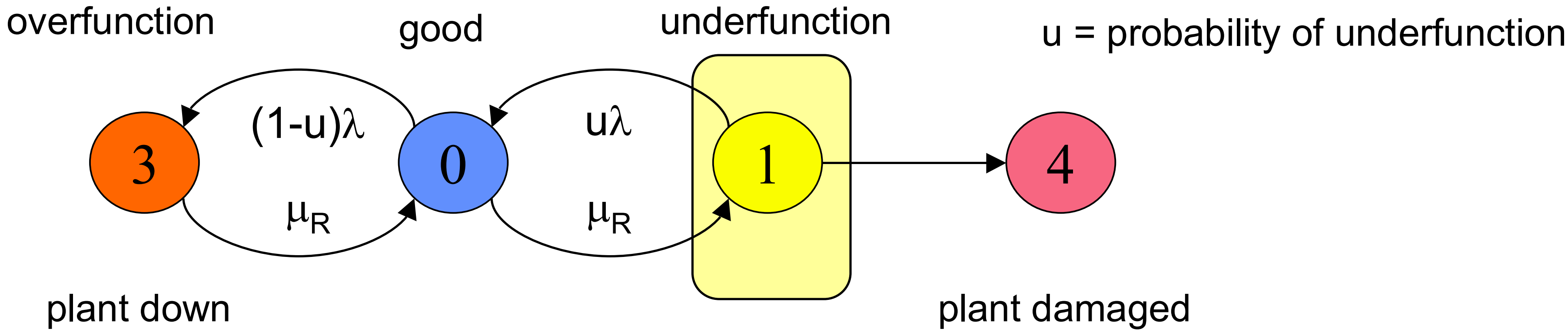
recommendation: increase coverage by using alternatively members N and R
(at least every start-up)

Probability to Fail on Demand for safety (protection) system

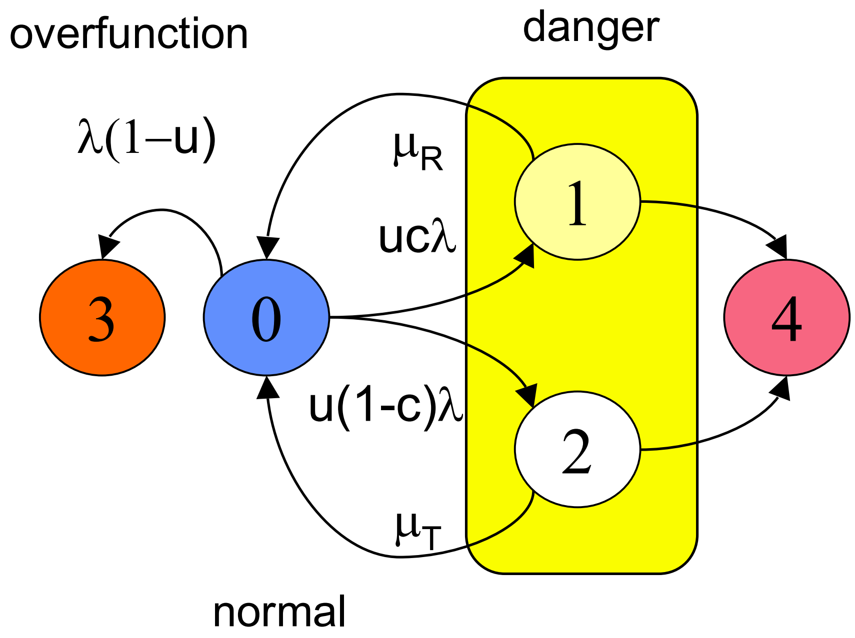


IEC 61508 characterizes a protection device by its Probability to Fail on Demand (PFD):

PFD = (1 - availability of the non-faulty system) (State 0)



Protection system with error detection (self-test) 1oo1



λ: protection failure

u: probability of underfunction [IEC 61508: 50%]

C: coverage, probability of failure detection by self-check

S1: protection failed in underfunction, failure detected by self-check (instantaneous), repaired with rate $\mu_R = 1/MRT$

S2: protection failed in underfunction, failure detected by periodic check with rate $\mu_T = 2/TestPeriod$

S3: protection failed in overfunction, plant down

S4: system threatened, protection inactive, danger

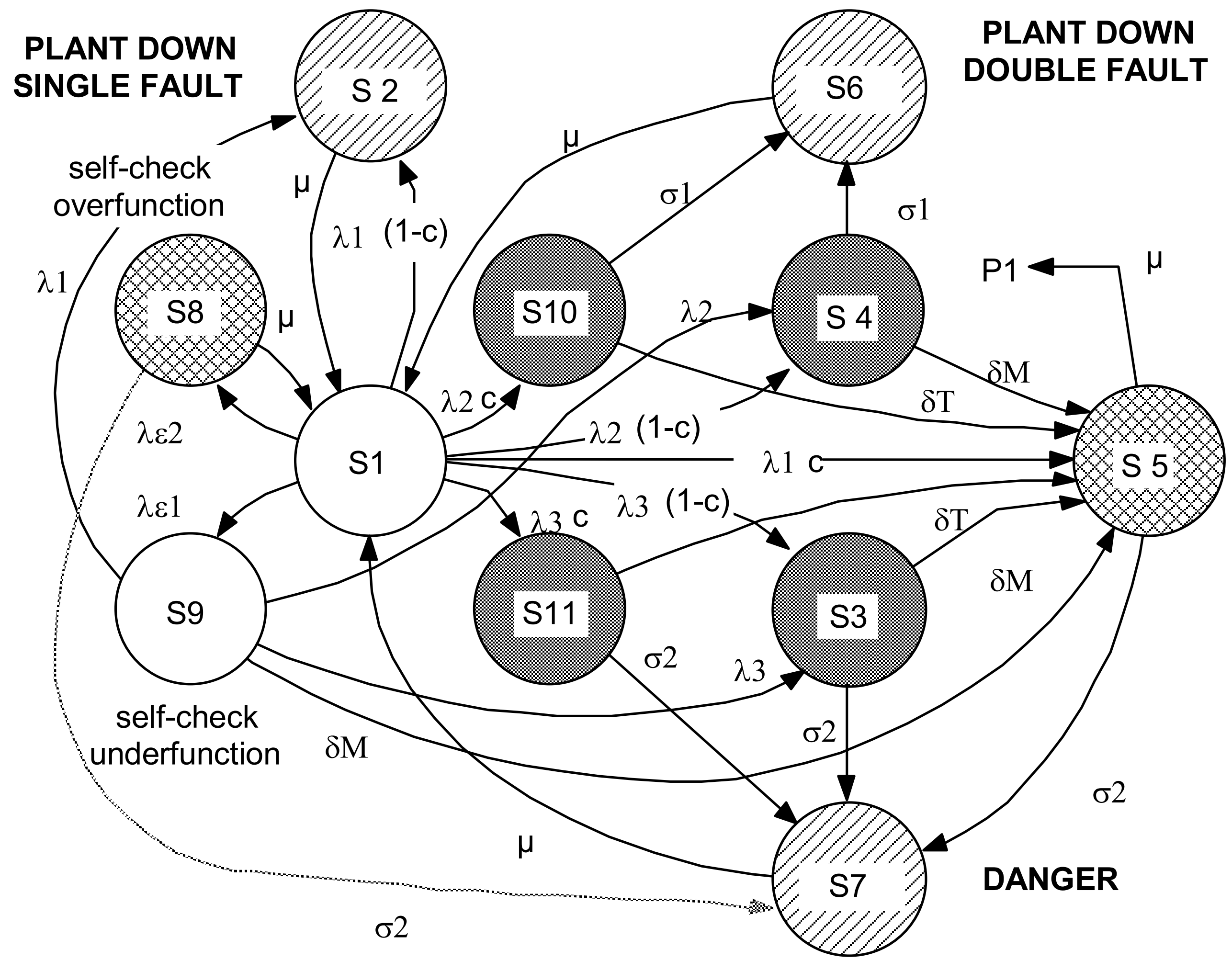
$$PFD = 1 - P_0 = 1 - \frac{1}{1 + \frac{\lambda u (1-c)}{\mu_T} + \frac{\lambda u c}{\mu_R}} \sim \lambda u \left(\frac{(1-c)}{\mu_T} + \frac{c}{\mu_R} \right)$$

with:
$\lambda = 10^{-7}\ h^{-1}$
MTTR = 8 hours -> $\mu_R = 0.125\ h^{-1}$
Test Period = 3 months -> $\mu_T = 2/4380$
coverage = 90%

$\longrightarrow$ PFD = $1.1\ 10^{-5}$

for S1 and S2 to have same probability: c = 99.8% !

Example: CIGRE model of protection device with self-check



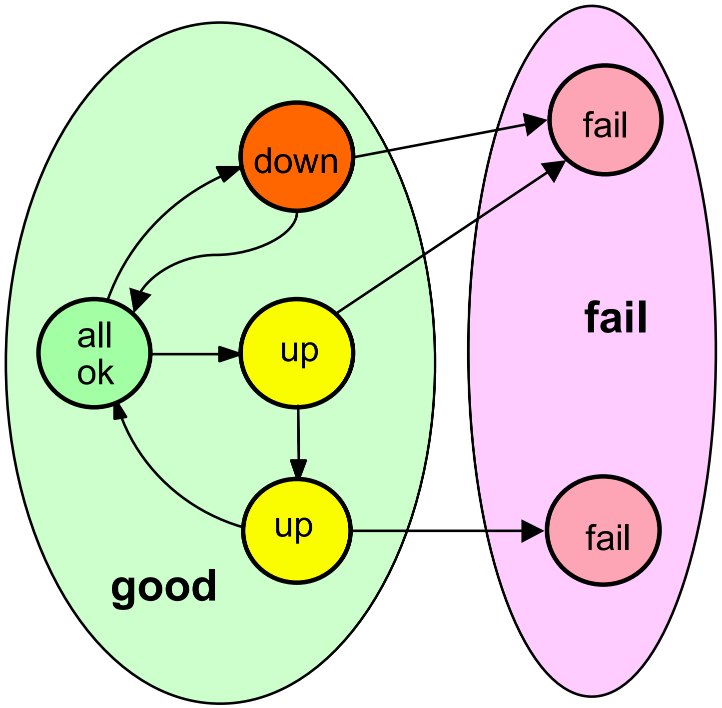
P8, P9: error detection failed

P10, P11: failure detectable by self-check

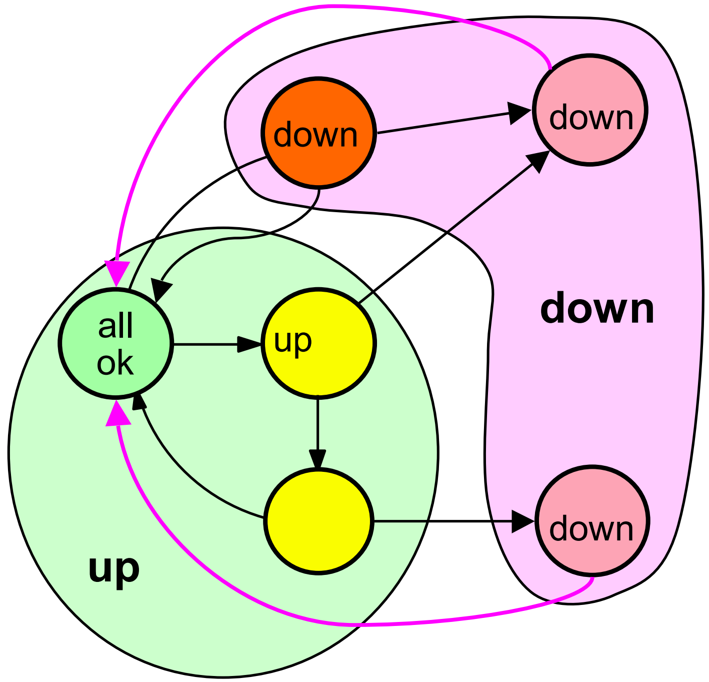
P4, P3: failure detectable by inspection

Summary: difference reliability - availability

Reliability



Availability



look for: MeanTime To Fail
(integral over time of all non-absorbing states)
set up linear equation with s = 0,
initial conditions S0 =1
solve linear equation

look for: availability
(duty cycle in UP states)
set up differential equation (no absorbing states!)
initial condition is irrelevant
solve stationary case with ? p = 1

Case Study: the Eurocab railways signaling

Cas d'etude: signalisation ferroviaire Eurocab (ETCS)

Studienfall: die Eurocab-Signalisierung

Dr. Eschermann
ABB Research Center, Baden, Switzerland

Overview Dependable Communication

9.3.1 Cyclic and Event-Driven Communication (Revisited)

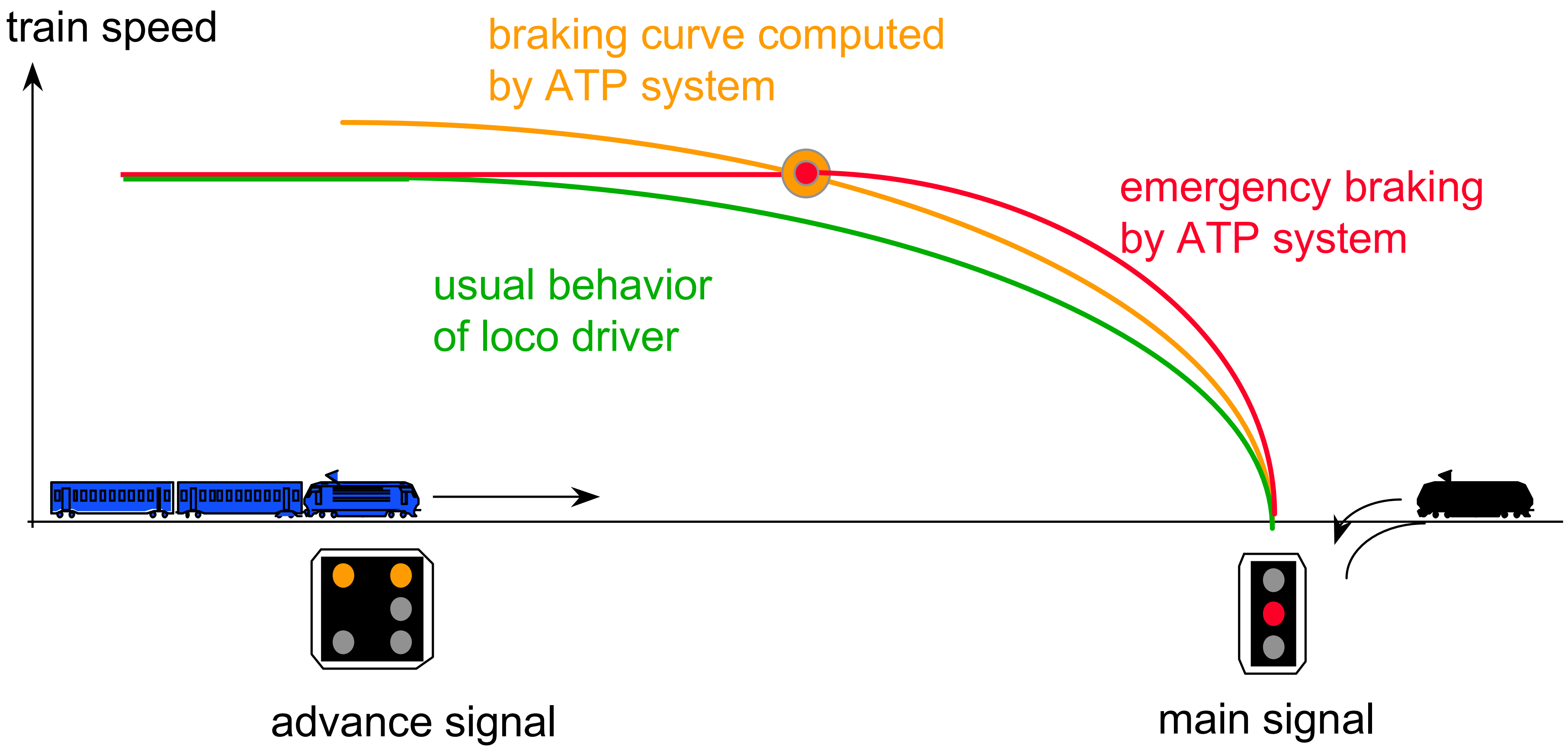
9.3.2 Communication Availability and Safety (Persistency and Integrity)

- Communication Hazards
- Transmission Redundancy
- Error-Detecting and Correcting Codes
- Time Stamps, Sequence Numbers and Timeouts
- Source and Sink Identification

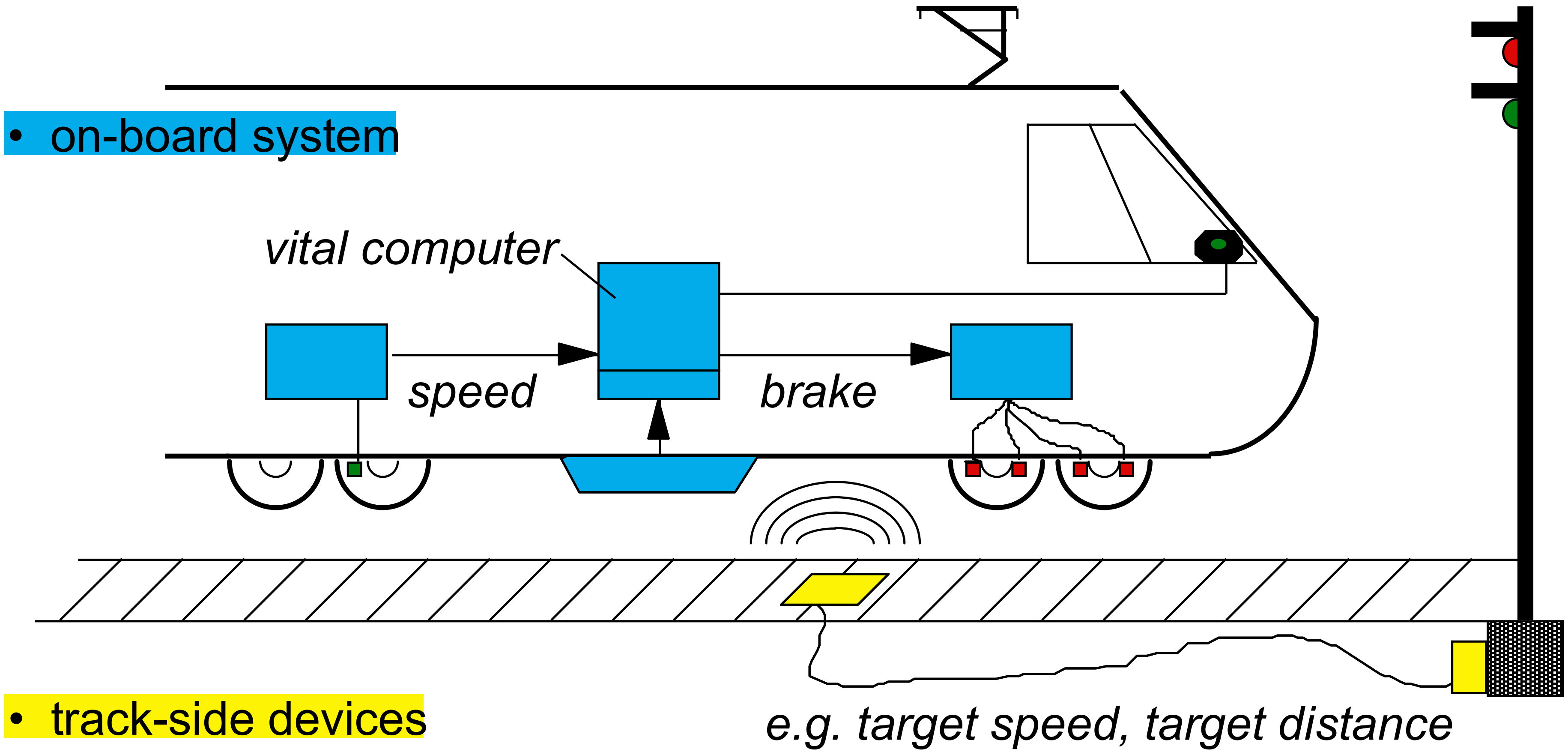
9.3.3 Example: Eurocab Safety Protocol

Example: Automatic Train Protection (ATP)

TASK: Train speed ≤ maximal allowed speed.



Simplified Structure of an ATP System



Eurocab: Motivation

o TODAY

- 13 different ATP systems in Western Europe
- either change locomotive at border or carry several ATP systems

ABB, ACEC, Alcatel SEL, Ansaldo,
CSEE, GEC Alsthom, SASIB,
Siemens, Westinghouse

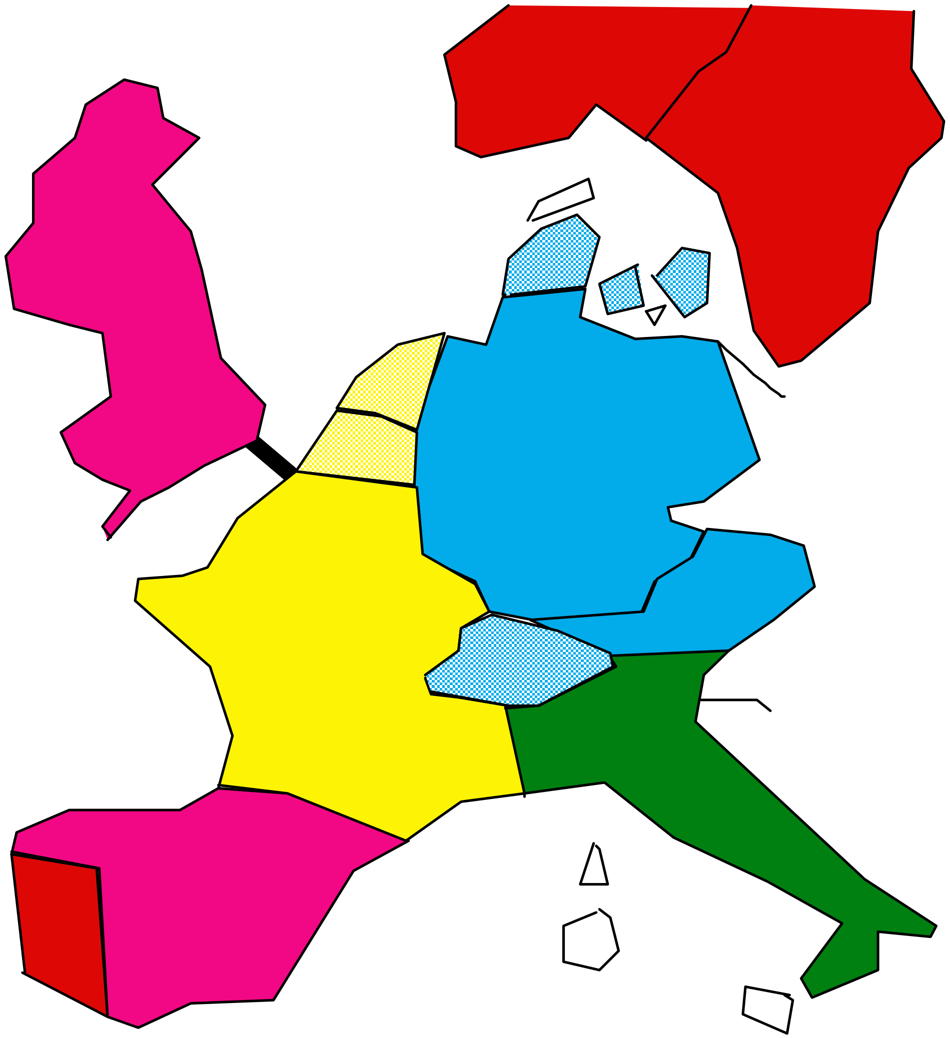
EU: part of funding

railways: requirements

o TOMORROW

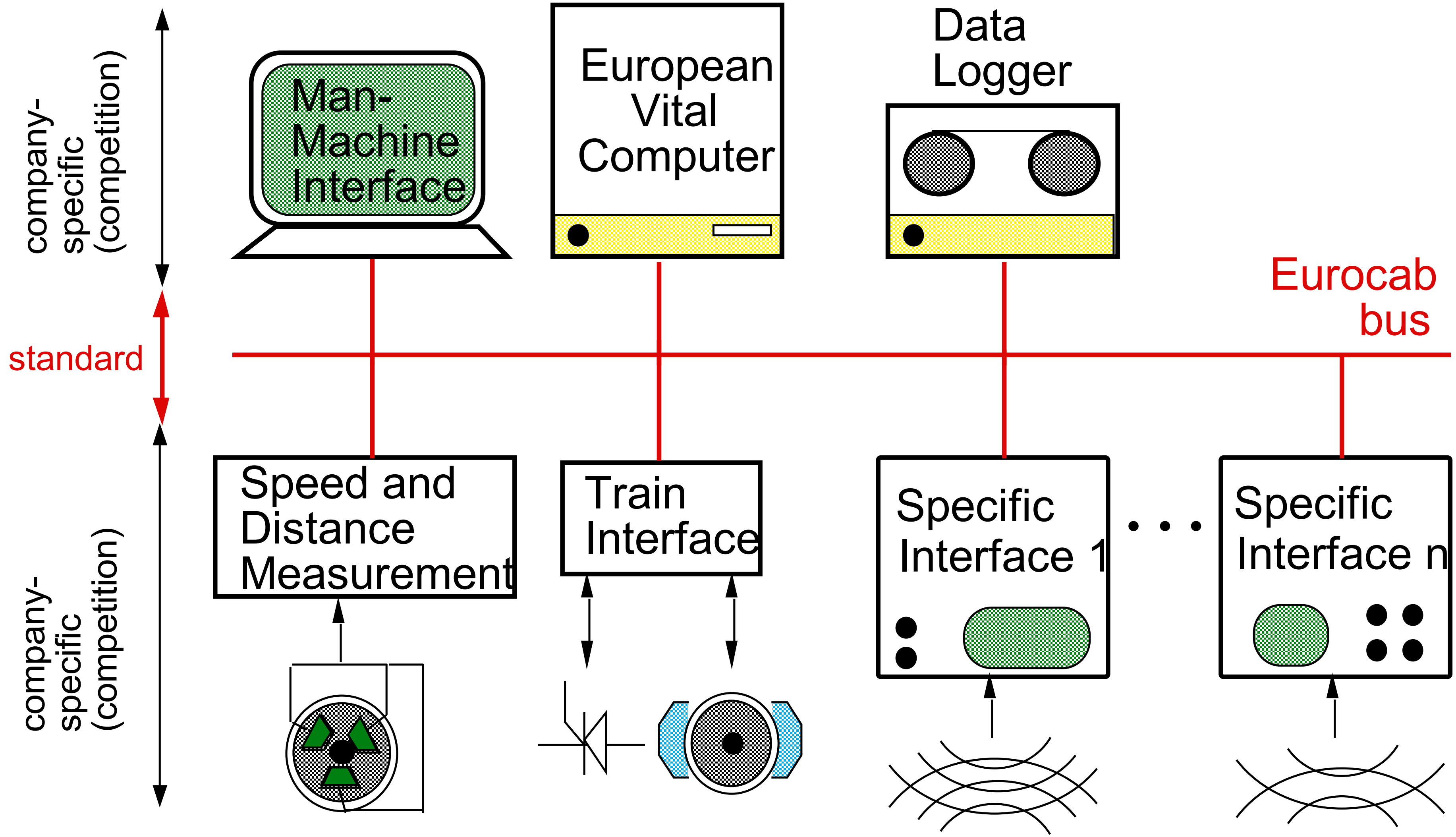
- Eurocab on-board system for all of Europe
- Eurobalise/Euroradio track-side devices complement existing track-side devices

ATP Systems in Western Europe

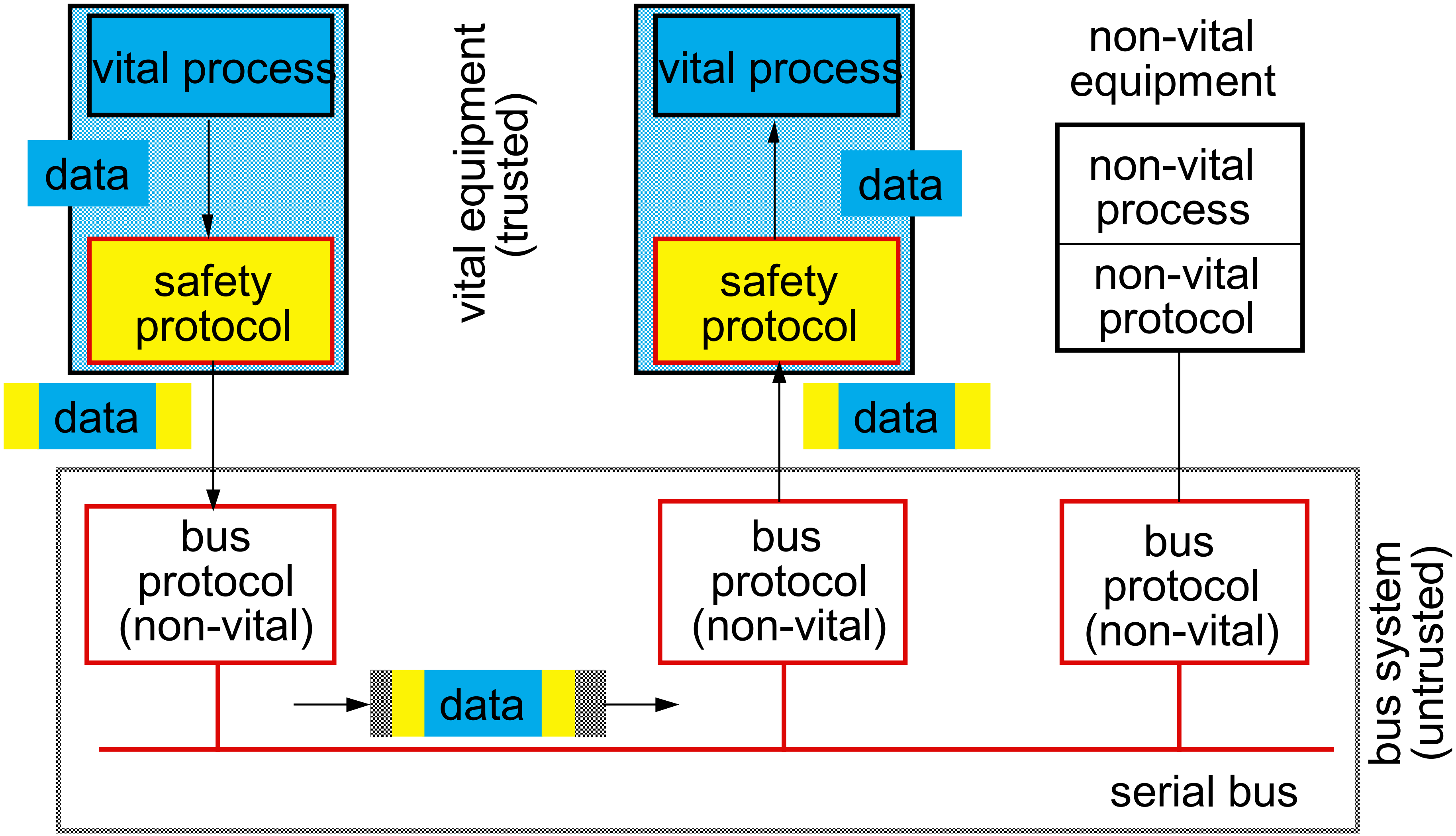


- ASFA: Dimetronic
- ATB: ACEC Transport
- AWS: Westinghouse
- BACC: Ansaldo, SASIB
- KVB: GEC Alsthom
- EBICAB: ABB Signal
- Indusi, LZB: Siemens, SEL
- SELCAB: Alcatel SEL
- TBL: ACEC Transport
- TVM: CS Transport
- ZUB: Siemens

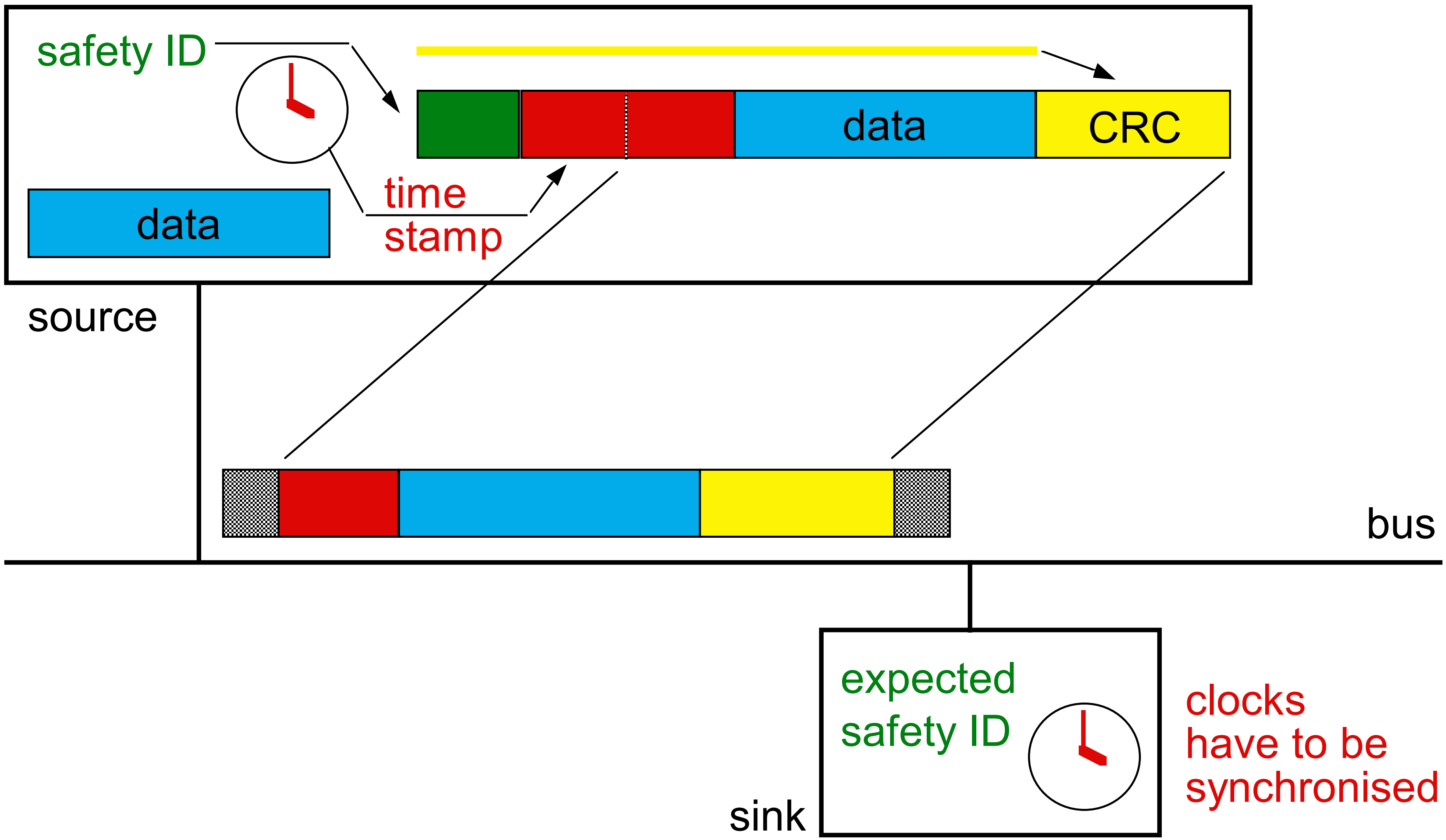
Eurocab: Bus-Based Structure



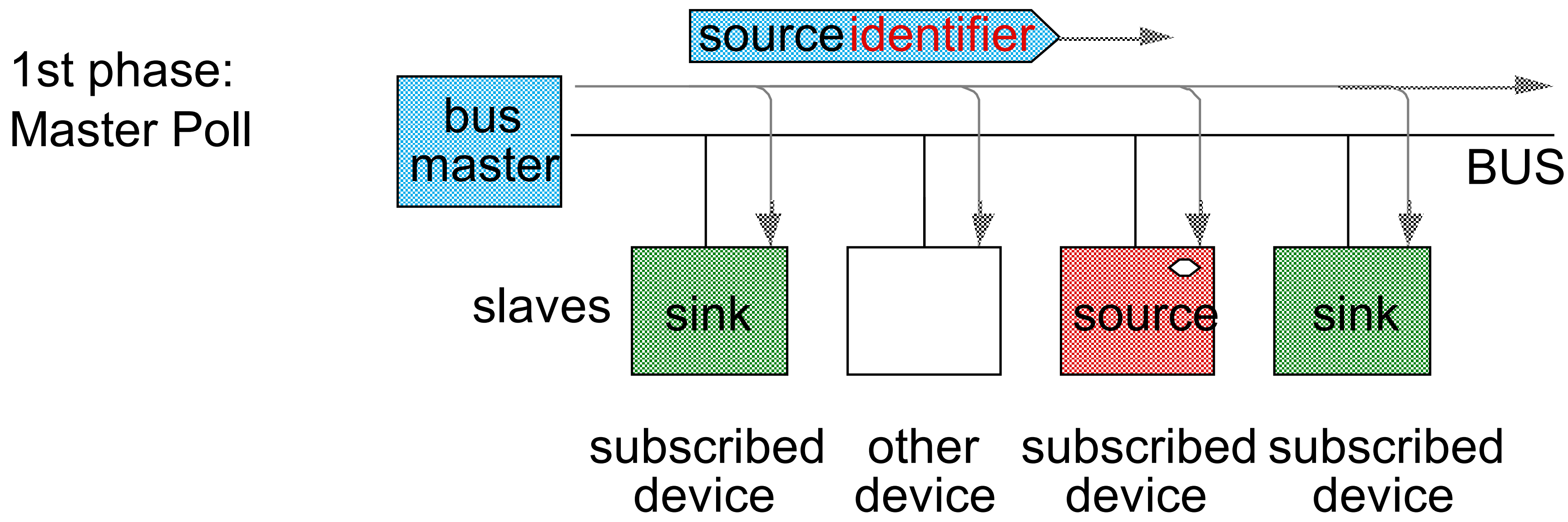
Role of the "Safety" Protocol

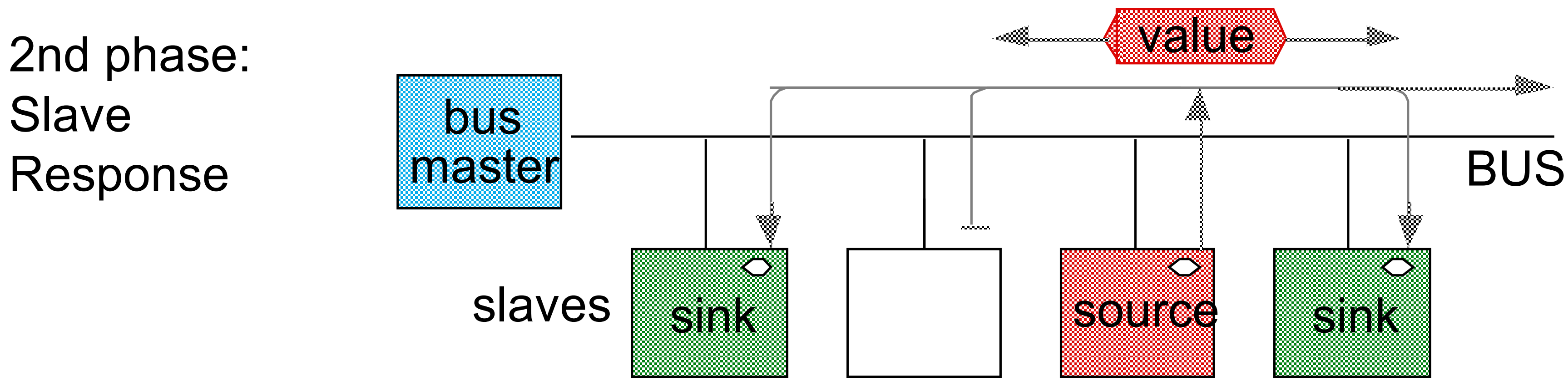


Protection of Vital Periodic Data



Addressing on Bus: Source-Addressed Broadcast

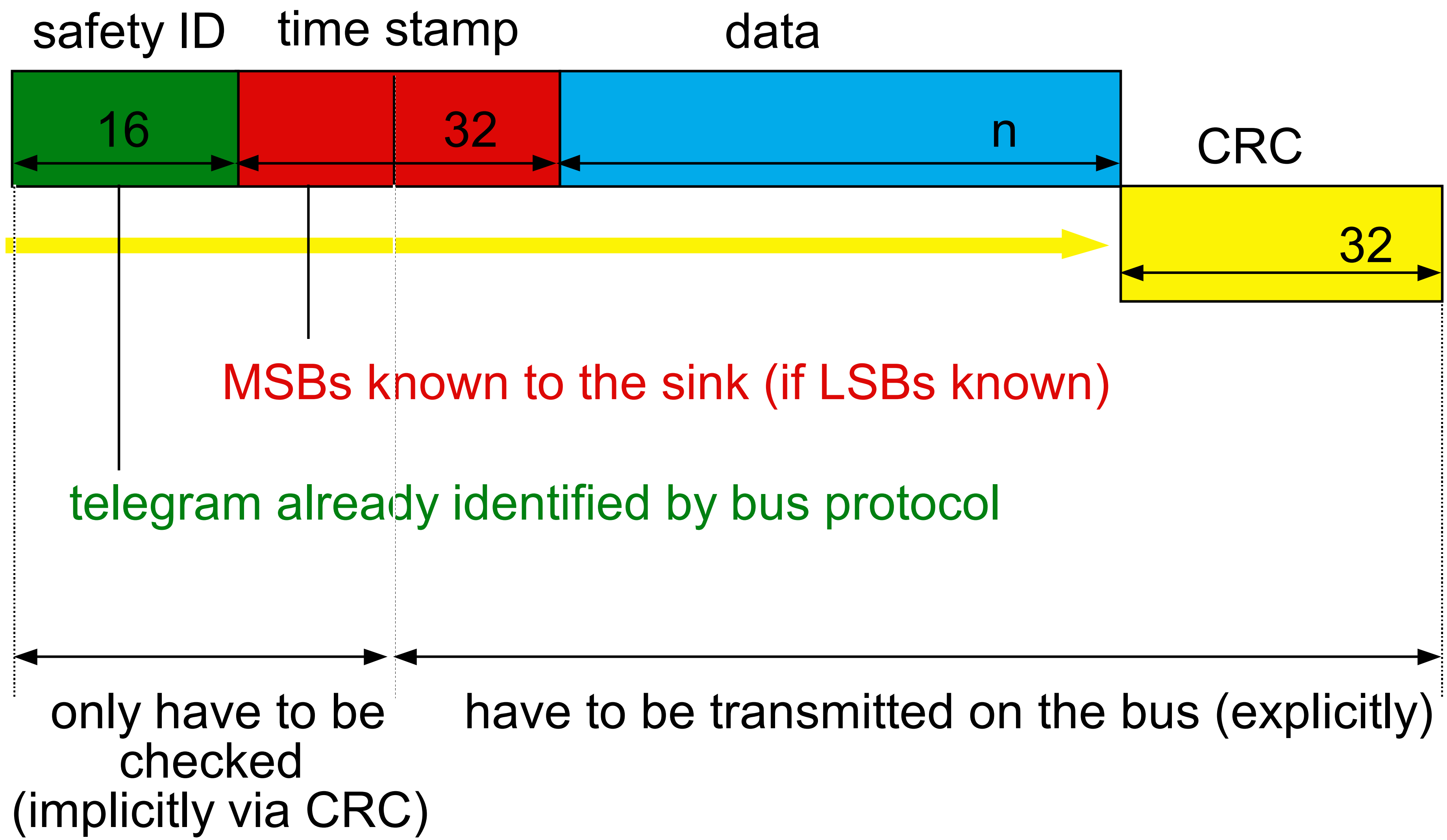




Safety ID for Vital Data

item	example value	comment
safety ID	0F11	unique value for telegrams with given characteristics
name of telegram	measured_speed	for identification
length	256 bits	data + explicit safety fields
periodic/sporadic	periodic	
broadcast/point-to-point	broadcast	
source function	SDM	producer of the data
sink function	any	since data are broadcast
grace period	3	number of telegrams that may be lost before safety reaction has to be initiated
time stamp interval	- 1 ms, + 257 ms	receiver check accuracy for time stamp
etc.	...	

Implicit and Explicit Data



Time Stamp Characteristics

Creation

Resolution (≠ accuracy !): 1 ms
Range (32 bits implicit+explicit): about 50 days

Resolution gives upper bound on accuracy,
but maximal accuracy does not have to be utilized today and by all units

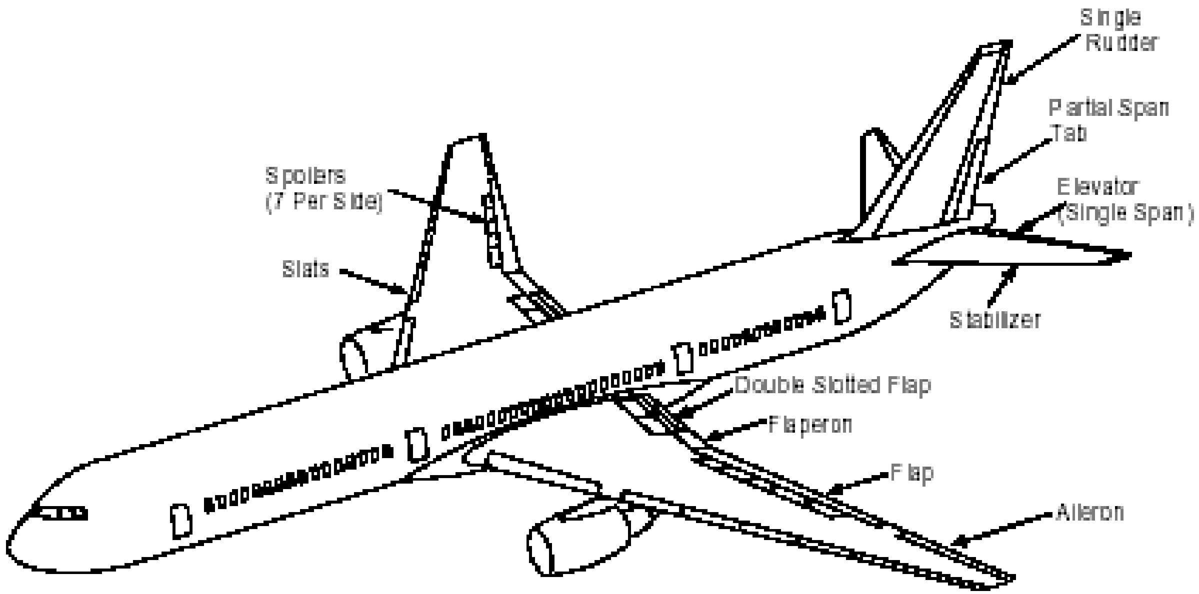
Checking

Sequence check by comparison $TS(i) \geq TS(i-1)$
Age check by comparison $LBTS(i) \leq TS(i) \leq UBTS(i)$

Acceptable window [LBTS(i)-TS(i), UBTS(i)-TS(i)] defines accuracy of age check.
Window accounts for unknown effects of clock inaccuracy, clock drifts, transmission delays, etc.
Can be tuned to exact telegram requirements (specified in Description Table for each Safety ID).

Summary: Eurocab Safety Protocol

error in ...	Protection of periodic data	Protection of sporadic data
... content	Safety CRC	Safety CRC
... address	Implicit Safety ID	Safety ID
... time	Explicit Time Stamp (LSBs) Implicit Time Stamp (MSBs) Receiver Time-Out	Sequence/Retry Nr. Sender Time-Out
... sequence	Explicit Time Stamp (LSBs) Implicit Time Stamp (MSBs)	Sequence/Retry Nr.



Dependable Architectures

9.4 *Architectures sûres de fonctionnement*

Verlässliche Architekturen

Prof. Dr. H. Kirrmann
ABB Research Center, Baden, Switzerland

Overview Dependable Architectures

9.4.1 Error detection and fail-silent computers
- check redundancy
- duplication and comparison

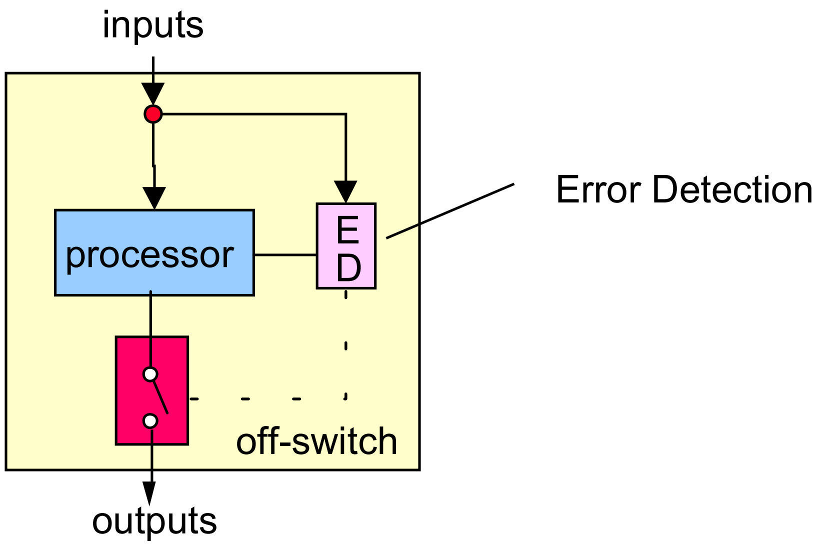
9.4.2 Fault-Tolerant Structures

9.4.3 Issues in Workby operation
- Input Processing
- Synchronization
- Output Processing

9.4.4 Standby Redundancy Structures
- Checkpointing
- Recovery

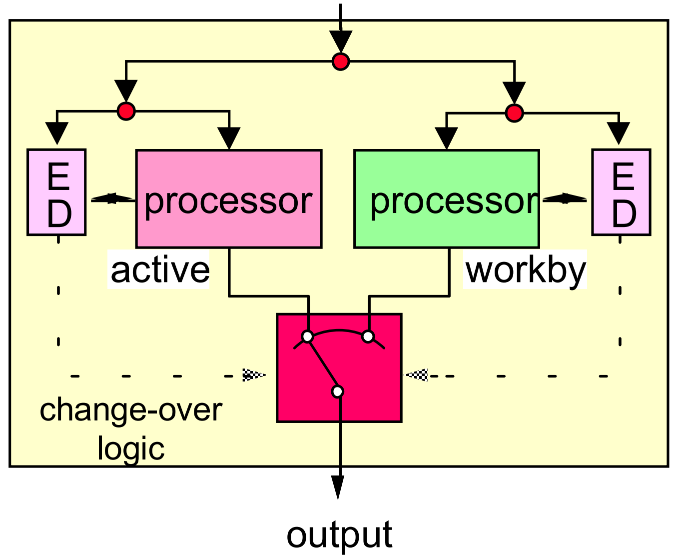
9.4.5 Examples of Dependable Architectures
- ABB dual controller
- Boeing 777 Primary Flight Control
- Space Shuttle PASS Computer

Dependable Computer Architectures



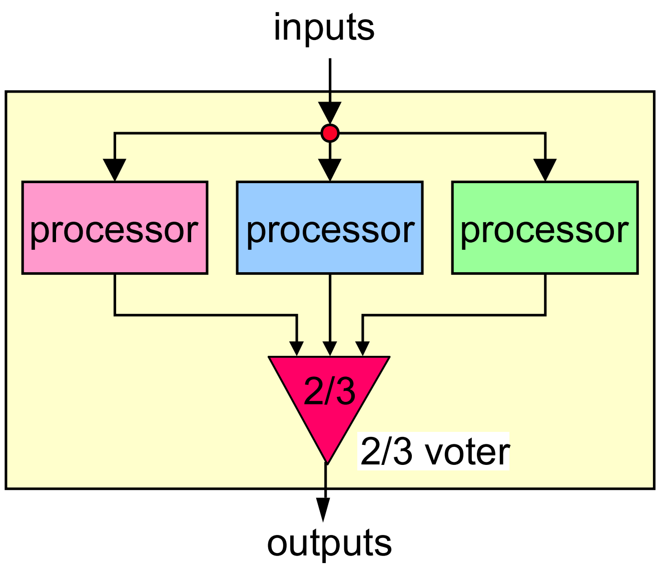
a) Integer

" rather nothing than wrong "
(fail-silent, fail-stop, "fail-safe")



b) Persistent

" rather wrong than nothing "
"fail-operate"



c) Integer & persistent
error masking

9.4.1 Error Detection and Fail-Silent

Error Detection

Error detection is the base of safe computing (fail-silent)

-> disable outputs if error detected

Error detection is the base of fault-tolerant computing (redundancy)

-> switchover if error detected

Key factors:

hamming distance:

how many simultaneous errors can be detected

coverage (*recouvrement*, Deckungsgrad)

probability that an error is discovered within useful time
(definition of "useful time": before any damages occur, before automatic shutdown,…)

latency (*latence*, Latenz)
time between occurrence and detection of an error

Detection of Errors Caused by Physical Faults

Error detection depends on the type of component, its error rate and its complexity.

Data transmission lines
medium to high error rate, memoryless

parity, CRC, watchdog

Regular memory elements
medium error rate, large storage

parity, Hamming codes, CRC on disk.

Processors and controllers
low error rate, high complexity

duplication and comparison, coded logic

Supporting elements
high error rate, high diversity

mechanical integrity, power supply supervision, watchdogs,...

Error Detection: Classification

Errors can be detected, with increasing latency:

–on-line (while the specified function is performed)

→ continuous monitoring/supervision

–off-line (in a time period when the unit is not used for its specified function)

→ periodic testing

–during periodic maintenance (when the unit is tested and calibrated)

The correctness of a result can be checked with

–**relative tests** (comparison tests):

by comparing several results of redundant units or computations

→ pessimistic, i.e. differences due to (allowed) indeterminism count as errors

high coverage, high cost

–**absolute tests** (acceptance tests):

by checking the result against an a priori consistency condition (plausibility check)

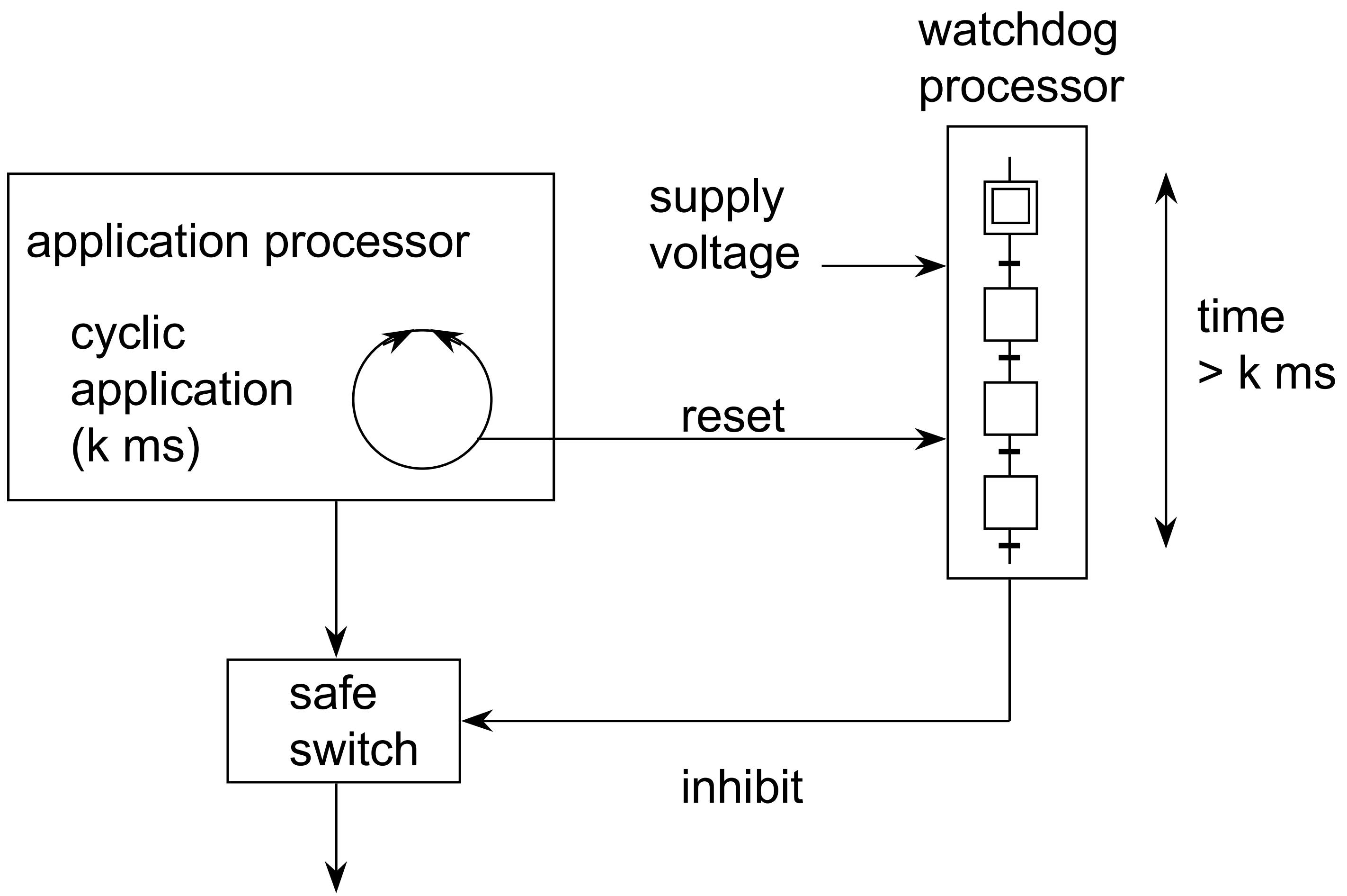
→ optimistic, i.e. even if result is consistent it may not be correct

(but can catch some design errors)

Error Detection: Possibilities

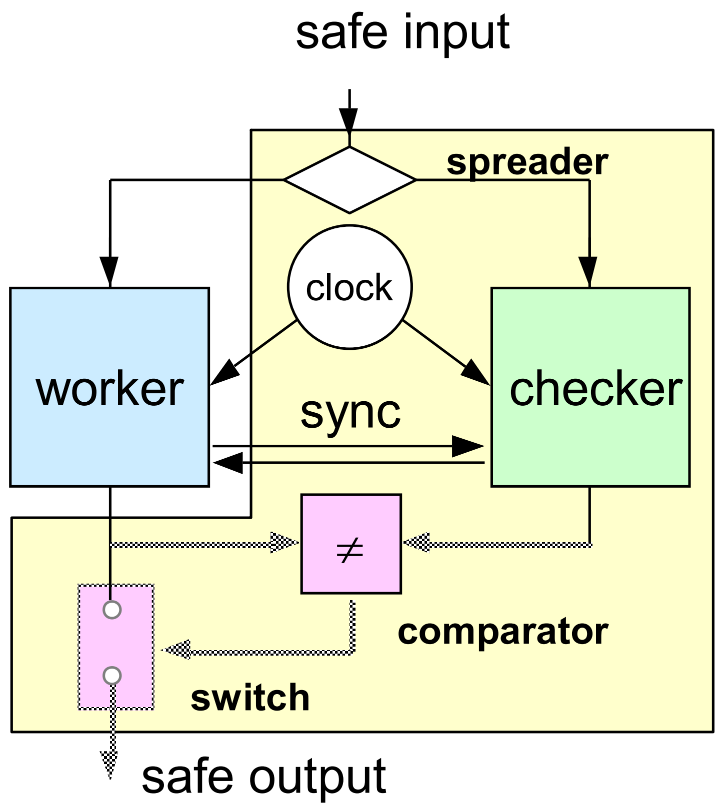
	relative test	absolute test
on-line	duplication and comparison (either hardware duplication or time redundancy) triplication and voting	watchdog (time-out) control flow checking error-detecting code (CRC, etc.) illegal address checking
off-line	comparison with precomputed test result (fixed inputs) e.g. memory test	check of program version check of watchdog function check code for program code

Error Detection: Watchdog Processor



The application processor periodically resets the watchdog timer. If it fails to do it, the watchdog processor will shut down and restart the processor processor.

Error Detection: Duplication and Comparison



Advantage: high coverage, short latency

Problems: non-determinism: digital computers are made of analog elements: (variable delays, levels, asynchronous clocks...)

The safety-relevant parts are useless if not regularly checked.

Conditions: worker and checker are identical and deterministic.
inputs are (made) identical and synchronized (interrupts !)
output must be synchronized to allow comparison.

Variant: the checker only checks the plausibility of the results
(requires definition of what is forbidden)

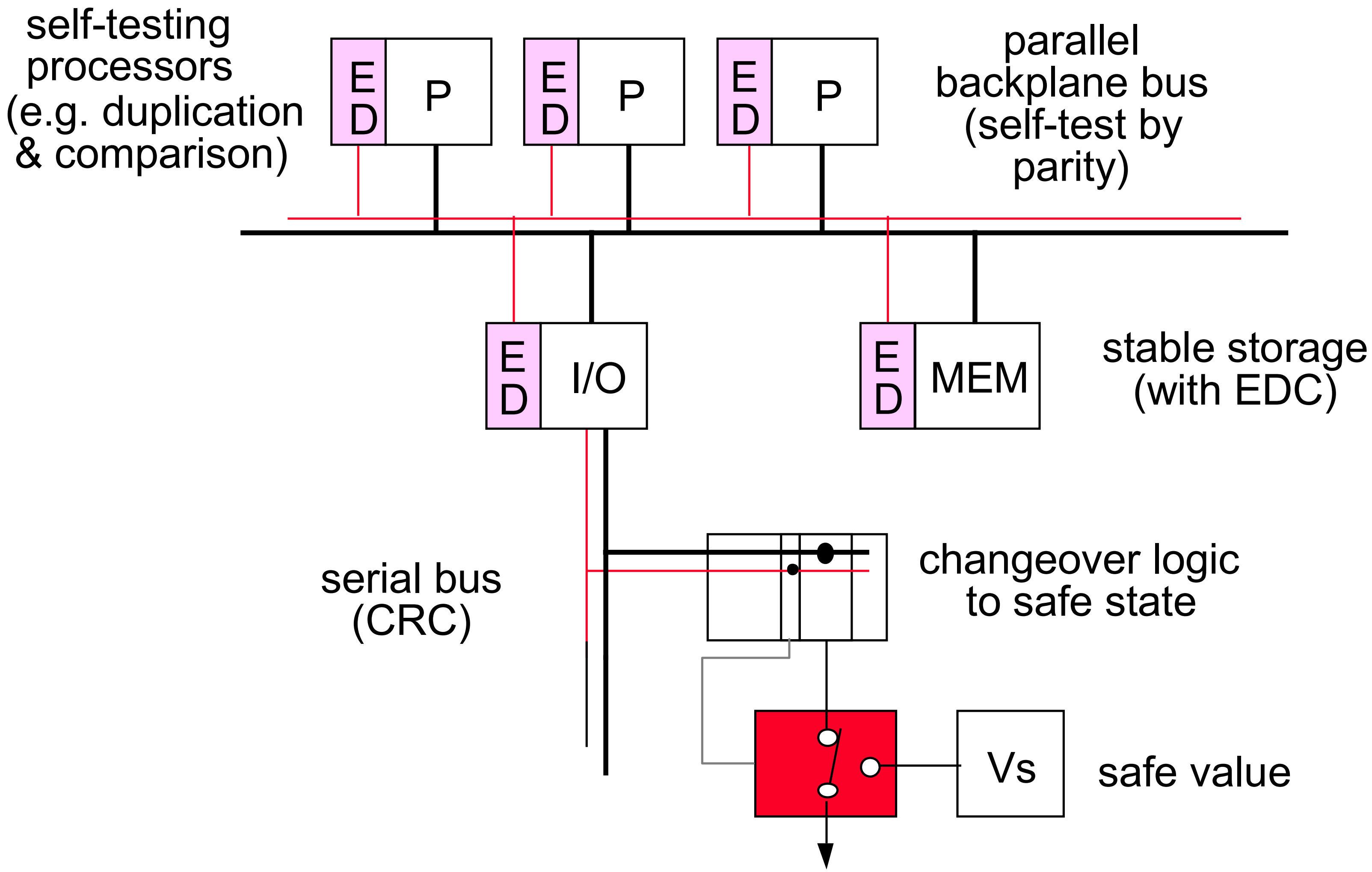
Integer processors

Integer processors are often called “fail-safe” processors, but they are only safe when used in plants where a safe state can be reached by passive means.

This requires a high coverage, that is usually achieved by duplication and comparison.

For operation, both computers must be operational, this is a 2oo2 structure (2 out of 2).

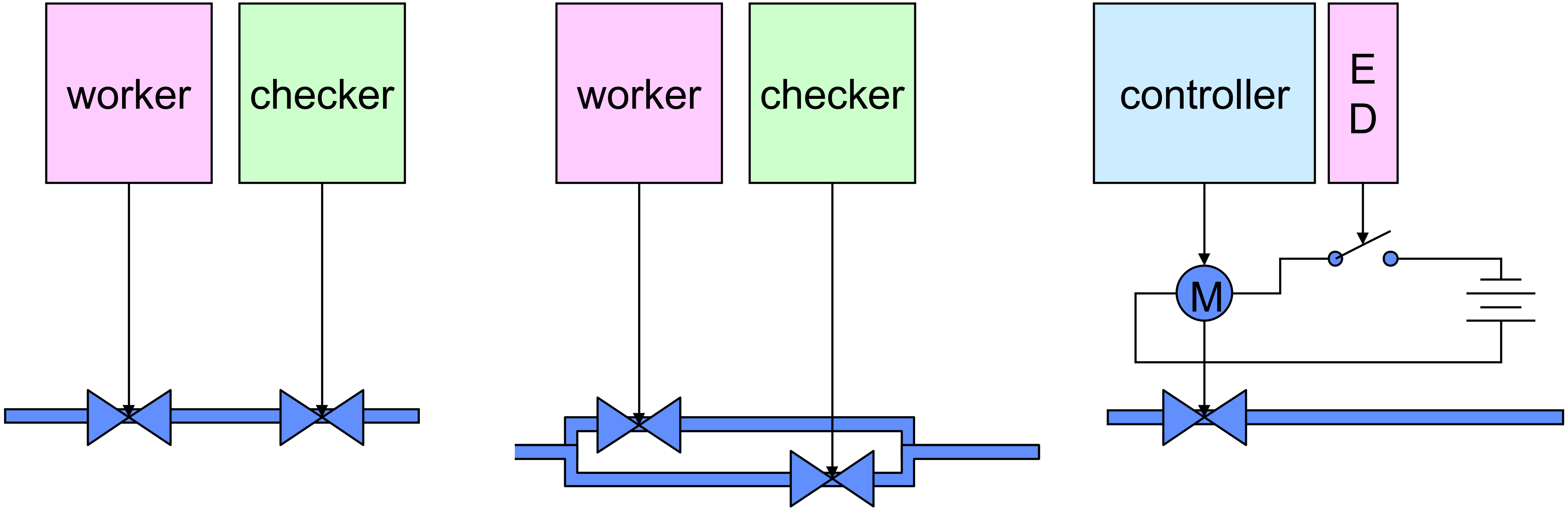
Integer Computers: Self-Testing System



Computers include increasingly means to detect their own errors.

Integer outputs: selection by the plant

The dual channel should be extended as far as possible into the plant



act if both agree
(workby)

act if any does
(workby)

act if error detection agrees

9.4.2 Fault-tolerant structures

Fault tolerant structures

Fault tolerance allows to continue operation in spite of a limited number of independent failures.

Fault tolerance relies on operational redundancy.

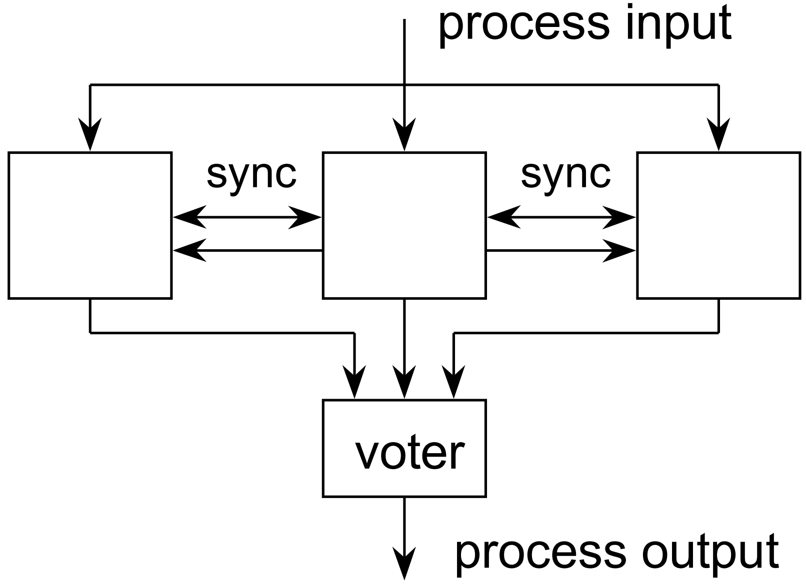
Static redundancy: 2 out of 3 (2oo3) Computer

Workby of 3 synchronised and identical units.

- All 3 units OK: Correct output.
- 2 units OK: Majority output correct.
- 2 or 3 units with same failure behaviour: Incorrect output.
- Otherwise: Error detection output.

also known as:

TMR (triple module redundancy)



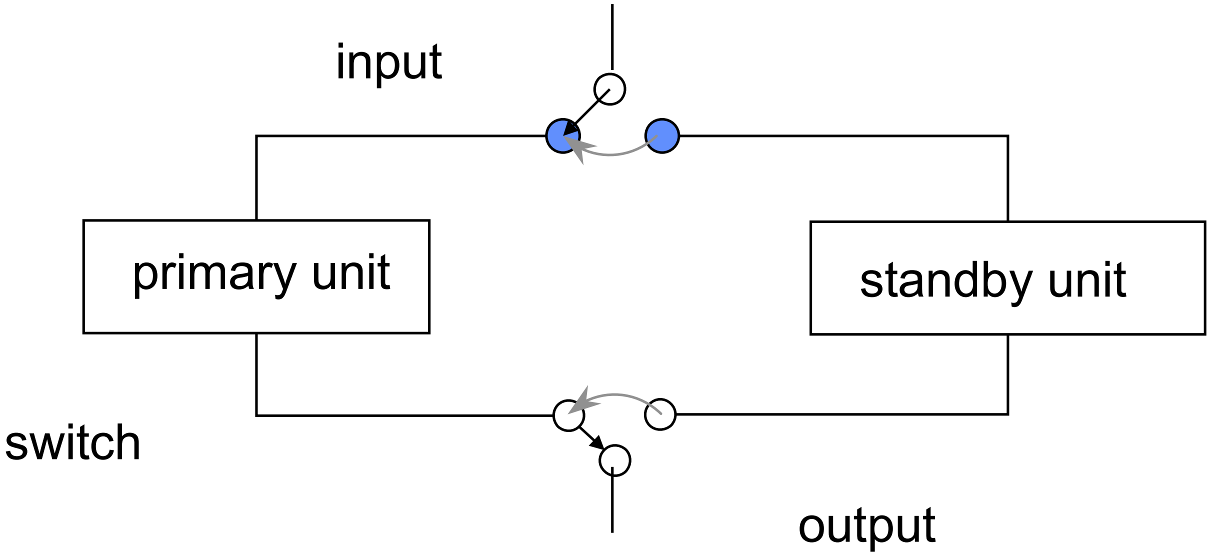
provides Safety (fail-silent) **and** availability (fail-operate) !

Dynamic Redundancy (vs. static redundancy like 2/3)

Redundancy only activated after an error is detected.

–primary components (non-redundant)

–reserve components (redundancy), standby (cold/hot standby)

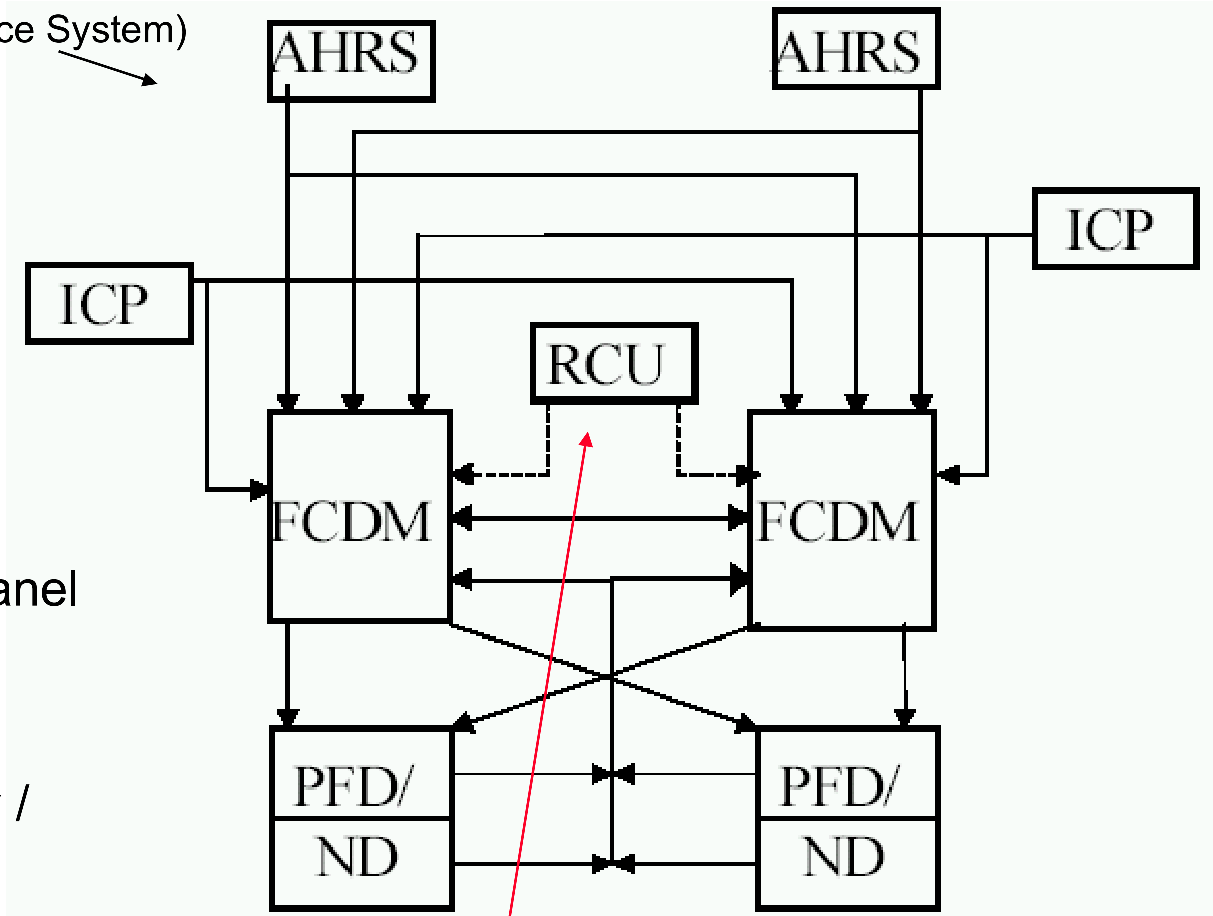


What are standby units used for?

– only as redundancy
– for other functions (can get lower priority in case of primary unit failure)
– better performance (“graceful degradation” in case of failure)

Example: Flight Control Display Module for helicopters

sensors
(Attitude Heading Reference System)



instrument control panel

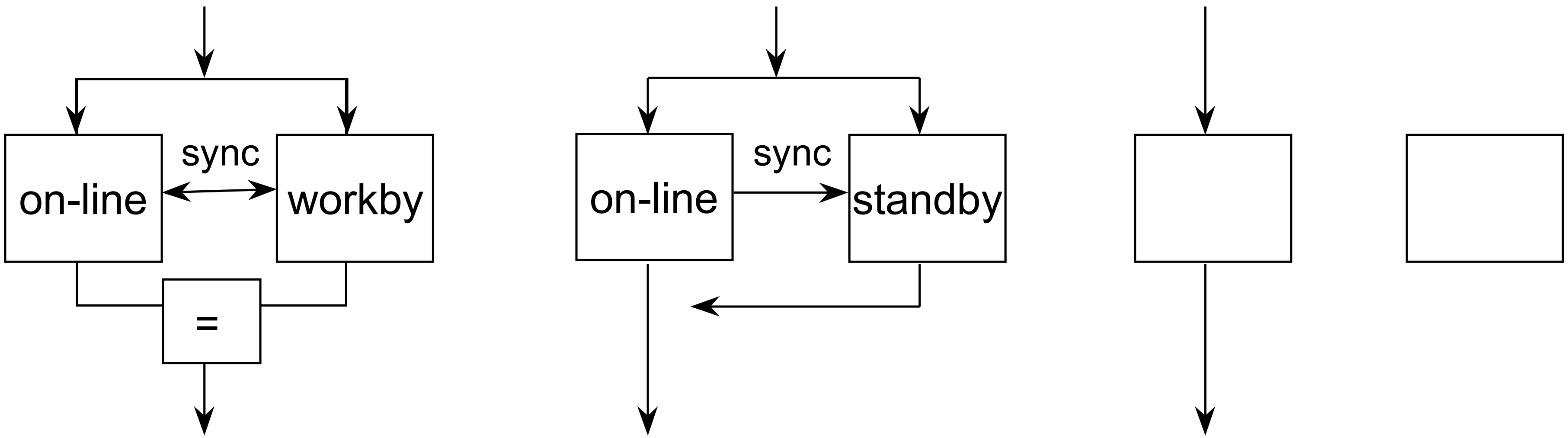
primary flight display /
navigation display

reconfiguration unit:
the pilot judges which
FCDM to trust in case of
discrepancy

source: National Aerospace Laboratory, NLR

Workby and Standby

Workby | Hot standby | Cold standby



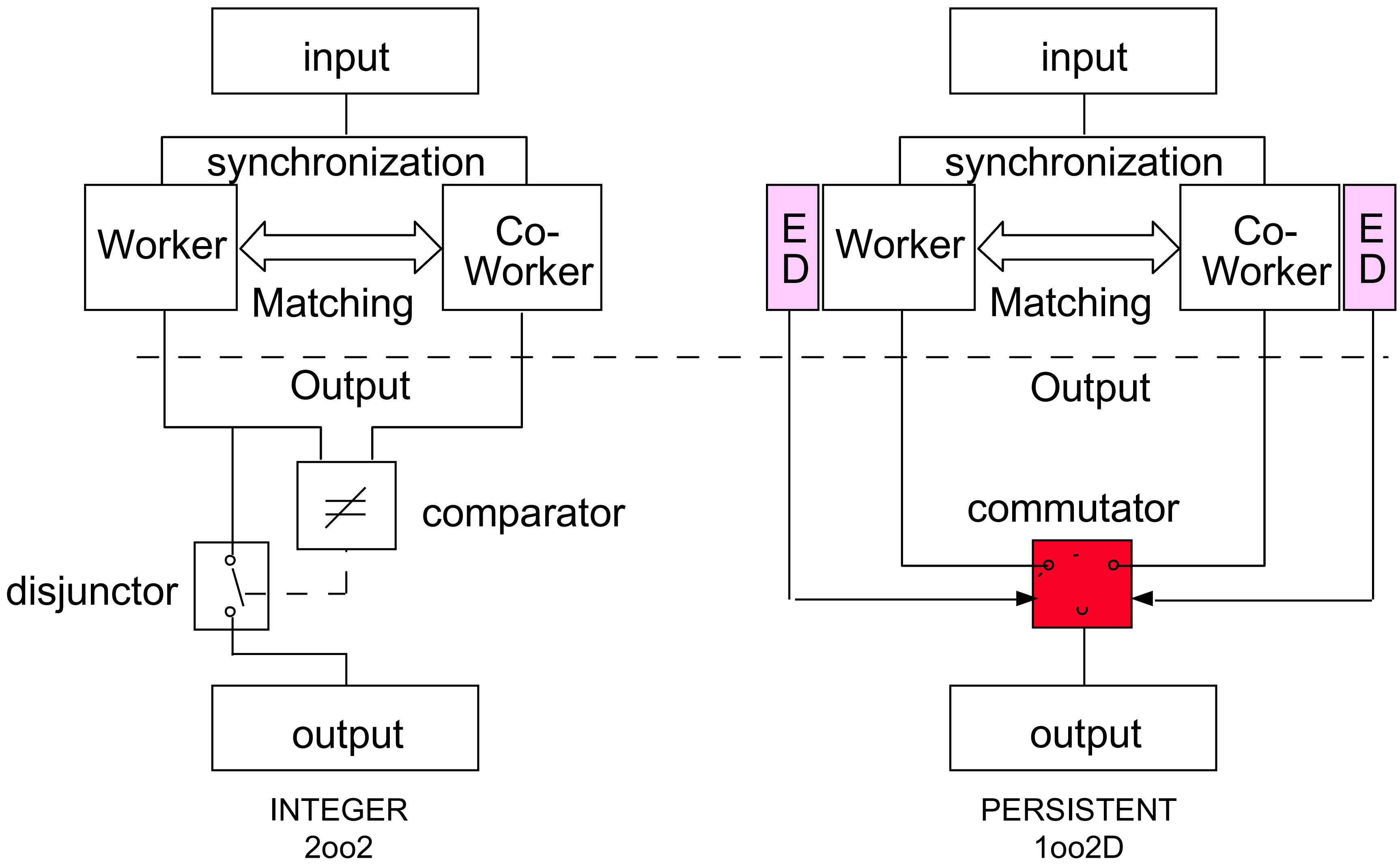
Both computers are doing the same calculations at the same time
Comparison for easy error detection.
Comparator needed.
Non-redundant continuation in case of failure?

Standby is not computing
Error detection needed.
Easy switchover in case of failure.
Easy repair of reserve unit.

Standby is no operational
Error detection needed.
Long switchover period with loss of state info.
No aging of reserve unit.

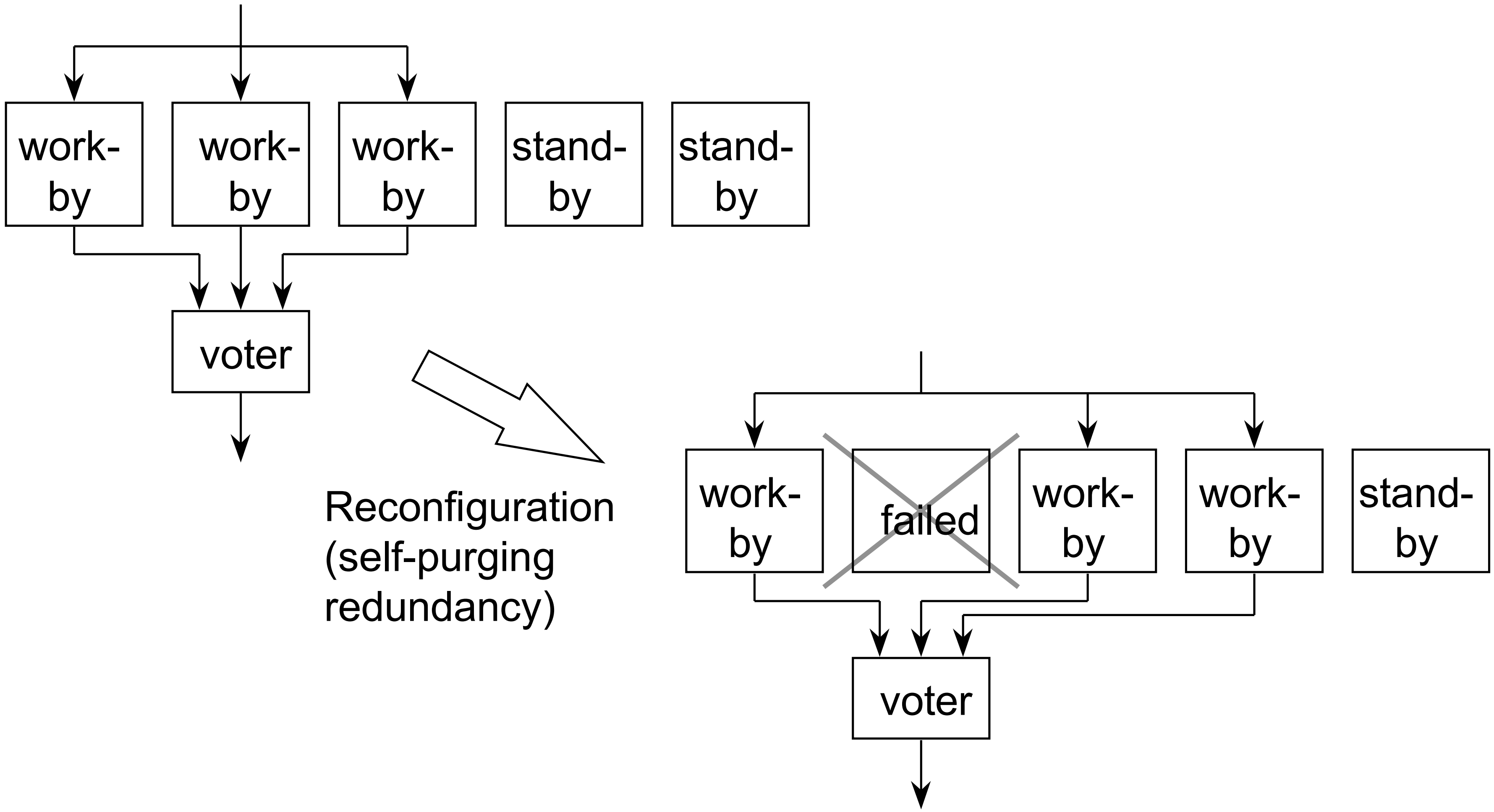
Workby: Fault-Tolerance for both Integrity and Persistency

réserve synchrone, synchroner Ersatz



Hybrid Redundancy

Mixture of workby (static redundancy) and standby (dynamic redundancy).



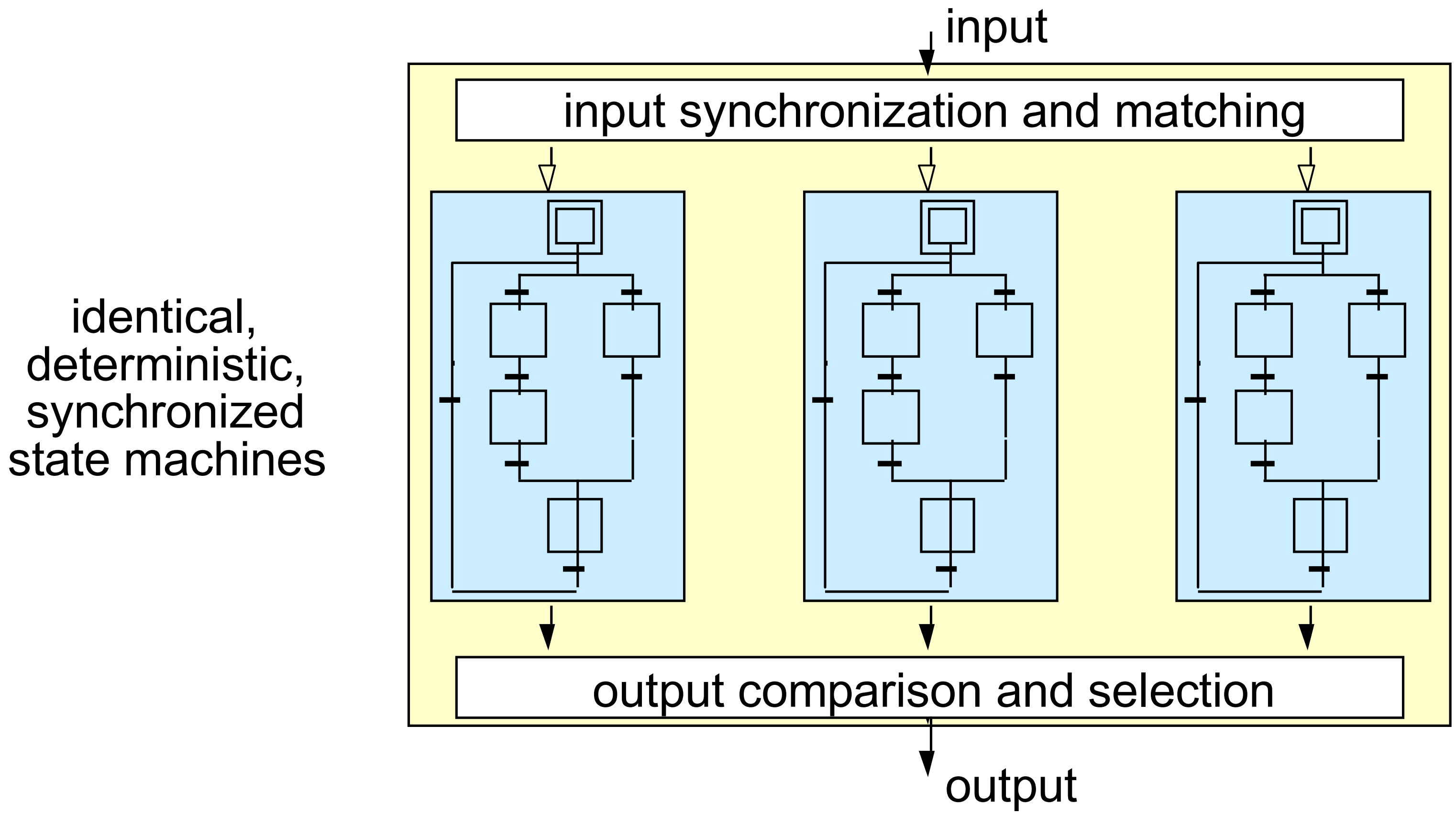
General designation

NooK: N out-of K

1oo1:	simplex system
1oo2:	duplicated system, one unit is sufficient to perform the function
2oo2:	duplicated system, both units must be operational (fail-safe)
1oo2D:	duplicated system with self-check error detection (fail-operational)
2oo3:	triple modular redundancy: 2 out of three must be operational (masking)
2oo4:	masking architecture

9.4.3 Workby

Workby: Input and Output Handling

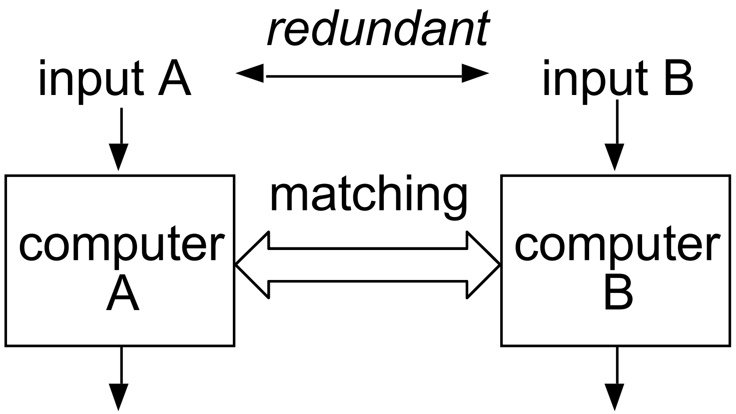


Replicated units must receive exactly the same input at the same time.

Delay (skew, jitter) between outputs must be below a certain value to allow comparison and smooth switchover.

Workby can be used to provide integrity (safety) or persistency (availability) and massive redundancy (masking)

Workby: Matching two inputs



Redundant inputs may differ in:

- value (different sensors, sampling)
- timing (even when coming from the same sensor, different delays)

Matching: reaching a consensus value used by all replicas

Binary inputs: matching within a time window, biased decision,...

Analog inputs: matching on median value, time-averaged value, exclusion of untrusted values,...

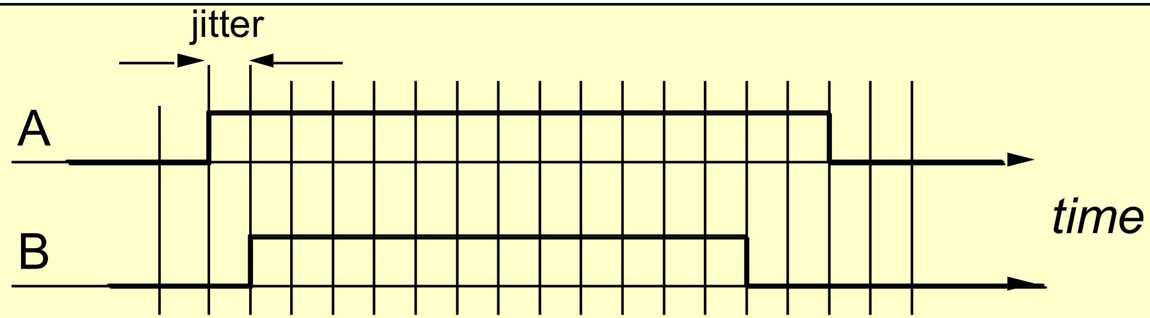
To reach a consensus, each computer must know the input value received by the other computer.

Matching requires application knowledge of the physical quantities involved.

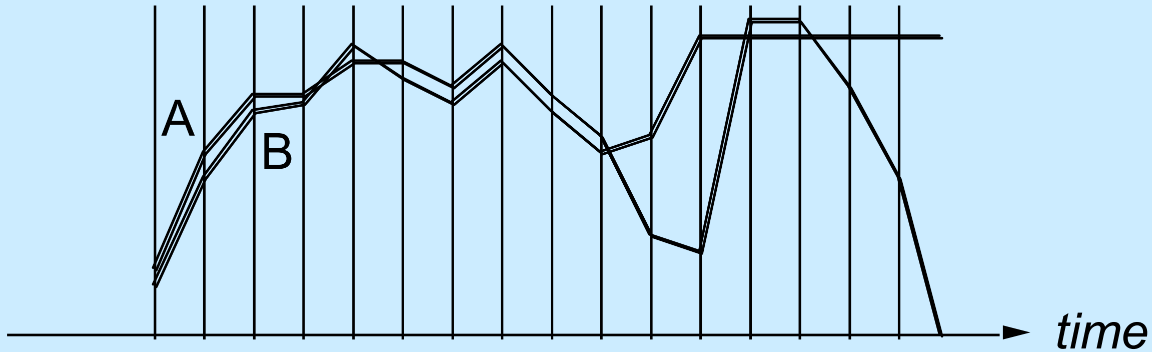
Matching

The matched value depends on the semantics of the variables.
Matching needs knowledge of the dynamic and physical behaviour.
Matching stretches over several consecutive values of the variables.

Binary variables:

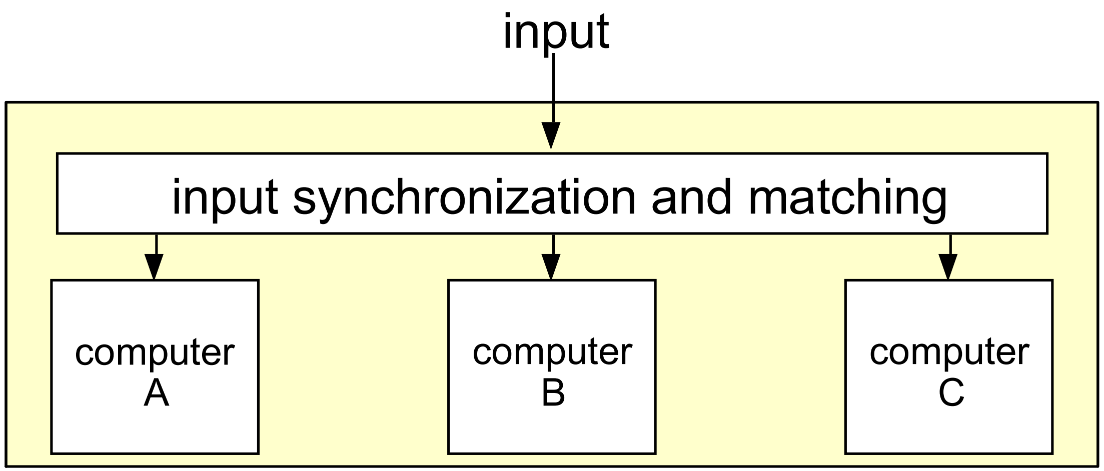


Analog variables:



Therefore, matching must be done by an application-dependent process.

Input synchronisation and matching in massive redundancy



Correct input synchronisation require input synchronization and matching (building a consensus value used by all the replicas)

Redundant sensors or same sensor value distributed to all replicas: needs application knowledge

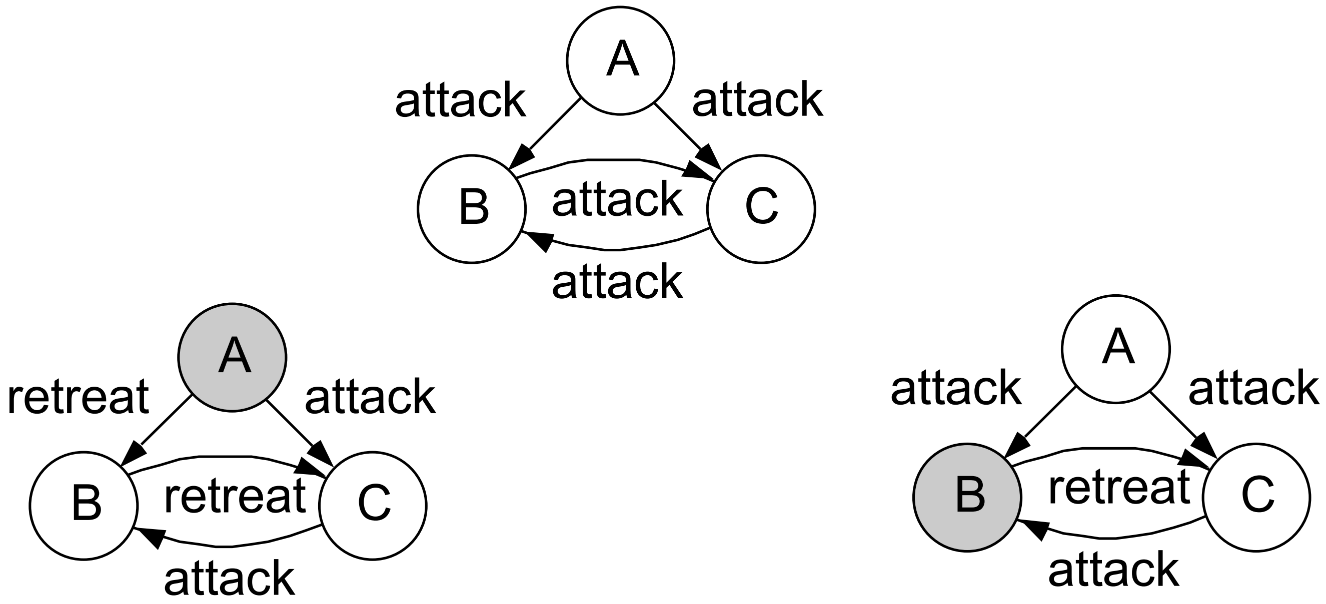
Every replica builds a vector of the value it received directly and the value received by the other units and applies the matching algorithm to it.

It is mandatory that all units can compare the same vector

-> reliable broadcast, Byzantine problems.

The Byzantine Generals´ Problem

For success, all generals must take the same decision, in spite of 't' traitors.



C cannot distinguish who is the traitor, A or B

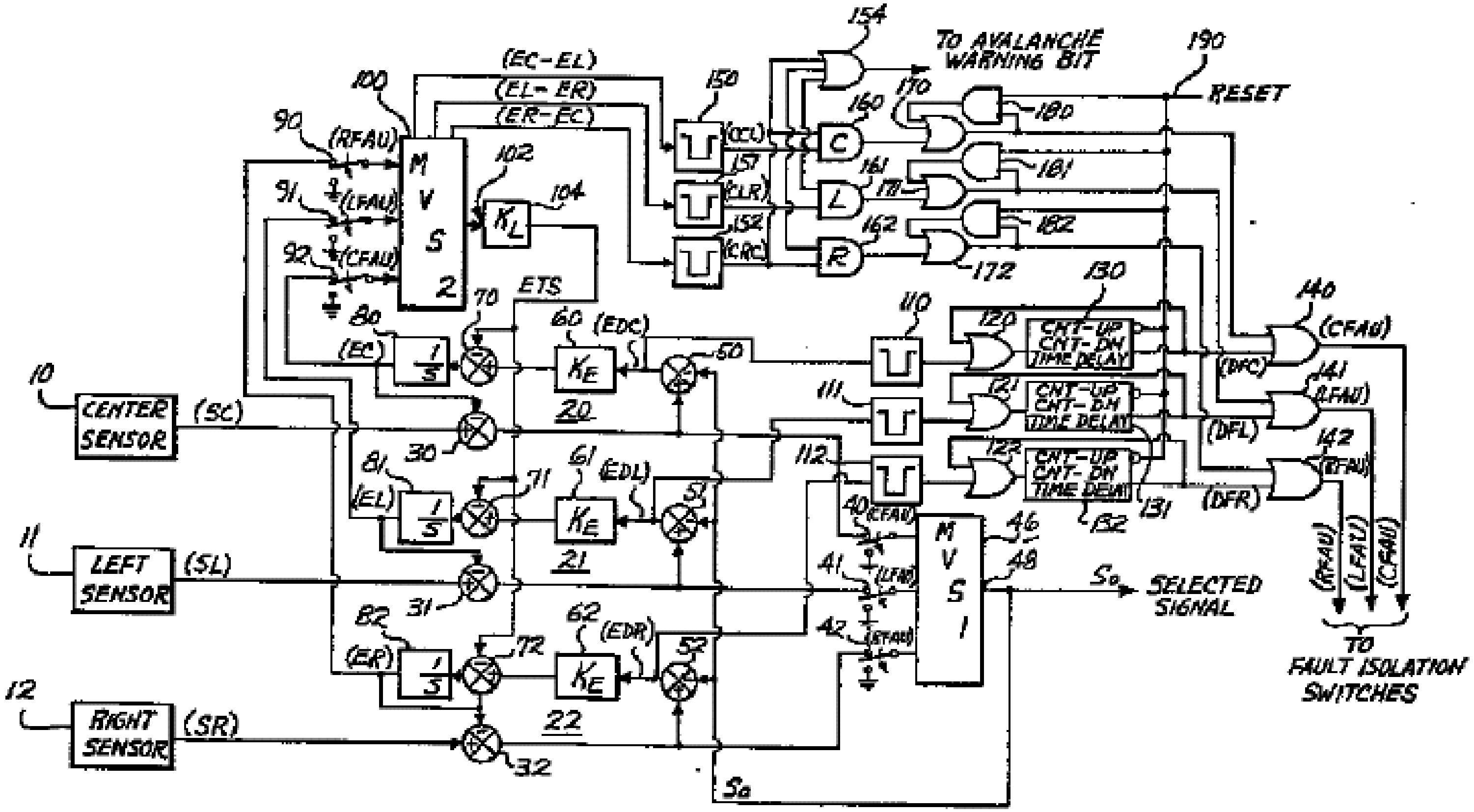
Solutions: No solution for ≤3t parties in presence of t faults.
Encryption (source authentification)
Reliable broadcast

Sources: Lamport, Shostak, Pease, "Reaching Agreement", J Asso. Com. Mach, 1980, , 27, pp 228-234.

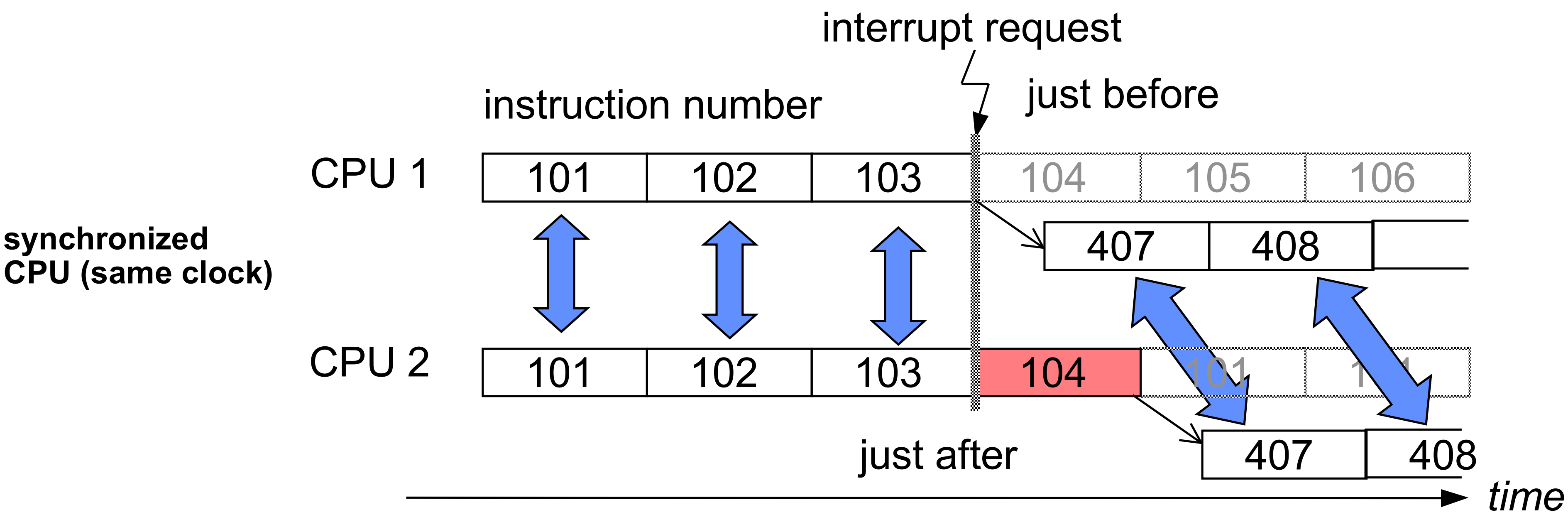
This is a general problem also affecting replicated databases

Matching - not so easy (a Boeing Patent)

(54) Title: SIGNAL SELECTION FROM REDUNDANT CHANNELS



Workby: Interrupt Synchronisation



Instructions may affect the control flow

Interrupts must be matched, like any other input data

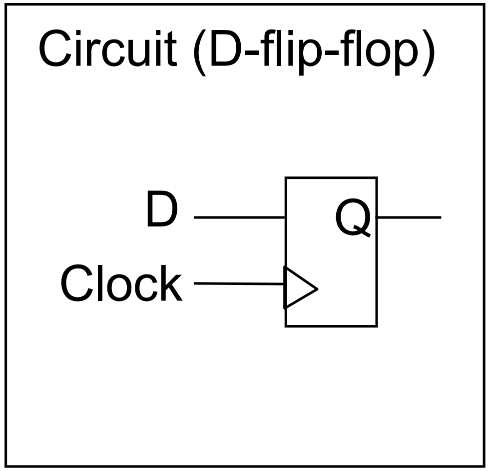
All decisions which affect the control flow (task switch) require previous matching.

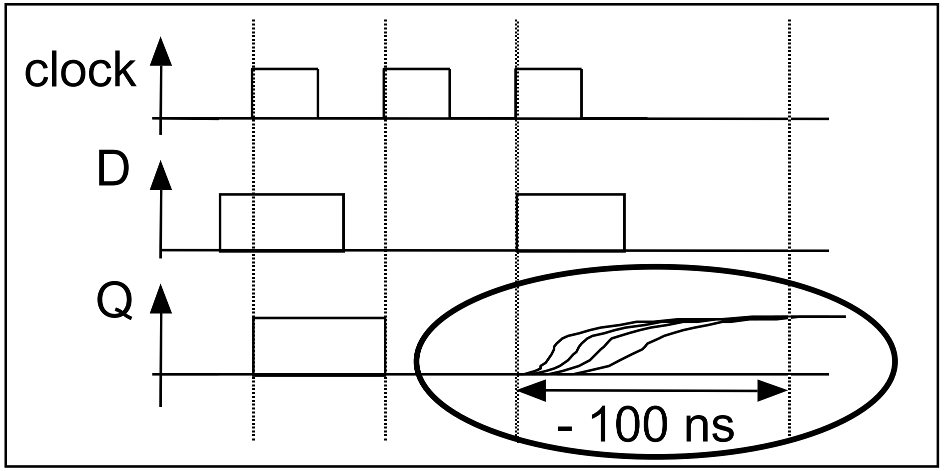
The execution paths diverge, if any action performed is non-identical

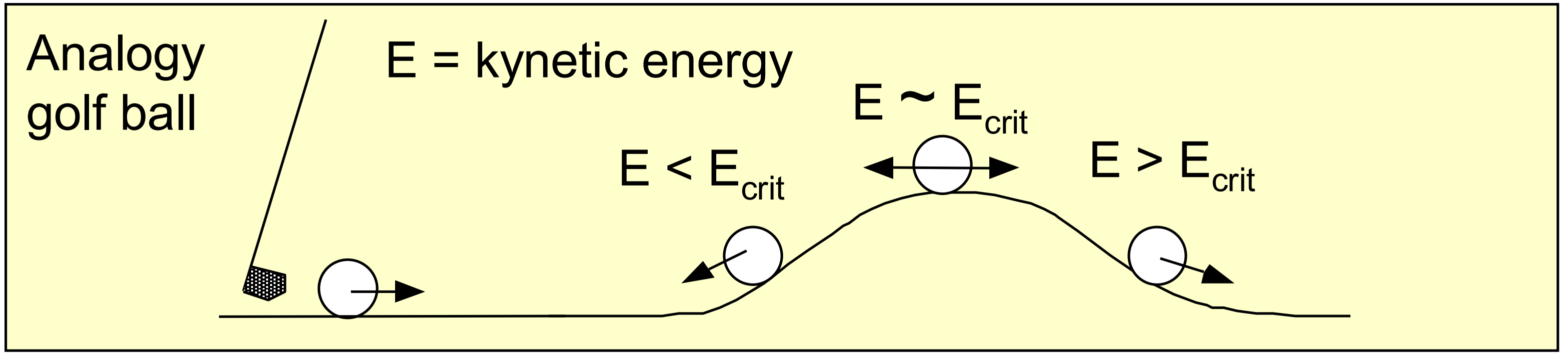
Solution: do not use interrupt, poll the interrupt vector after a certain number of instructions

Workby synchronisation: Metastability issue

The synchronization of asynchronous inputs by hardware means is only possible with a certain probability



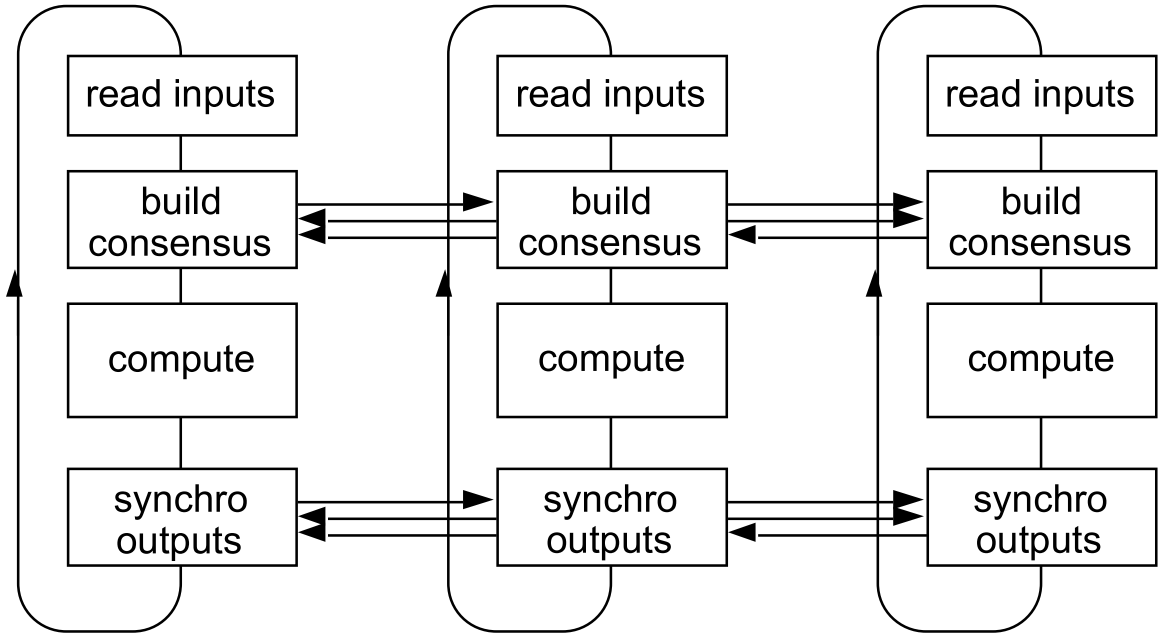




matching must rely on the exchange of defined signals, common signals are no suitable mean for reaching a consensus.

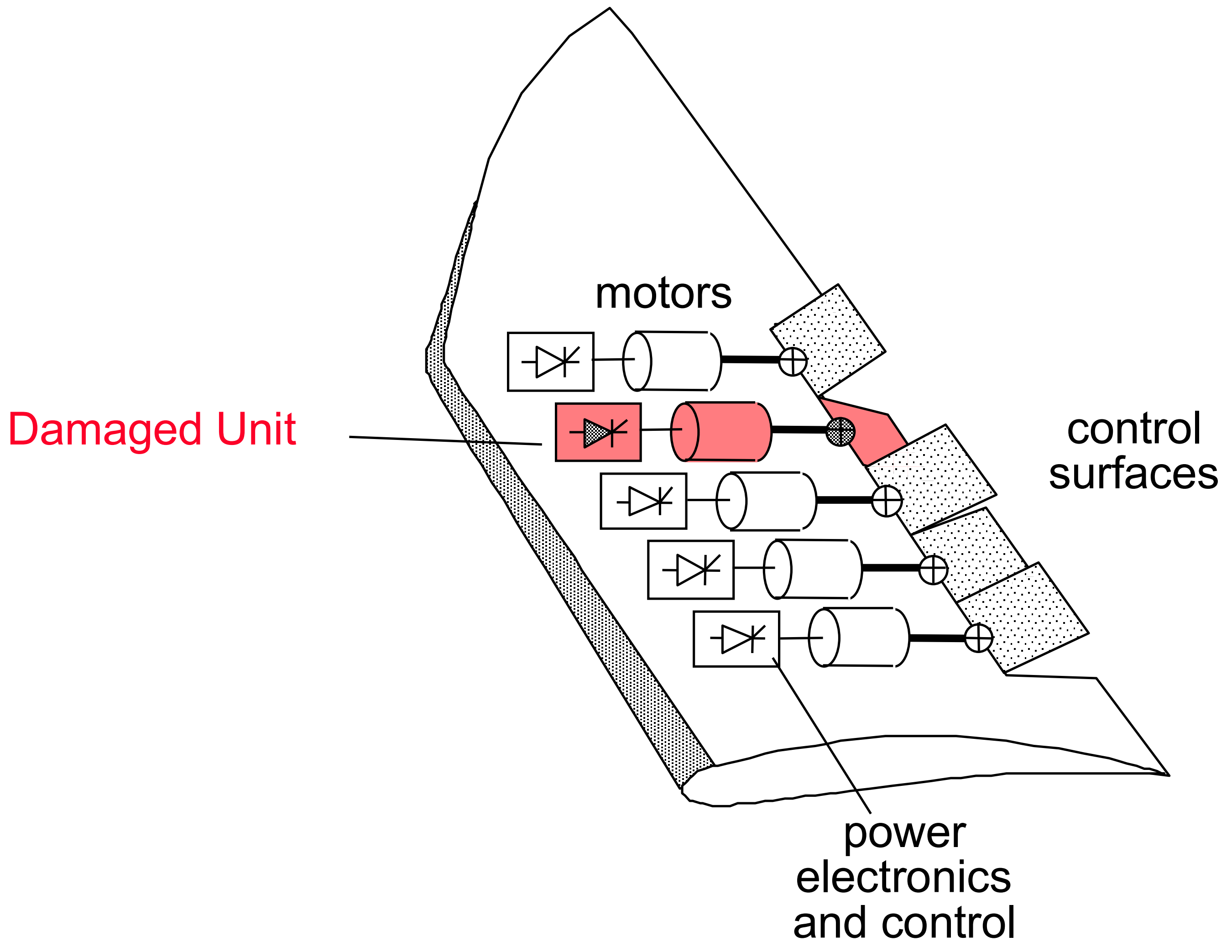
Workby: Output Comparison and Voting

The synchronized computers operate preferably in a cyclic way so as to guarantee determinism and easy comparison.



The last decision on the correct value must be made in the process itself.

Workby: Voting done by the controlled process



9.4.4 Standby

réserve asynchrone, *unbeteiligter Ersatz*

Dynamic Redundancy (e.g. with cold standby)

Standby consists in restarting a failed computation.

At the simplest, restart can be done on the same machine
(to cope with manipulation errors or transient faults) -> automatic restart.
this needs a recovery state stored on the same machine.

The basic techniques for state saving are the same as for the back-up in a personal computer or on mainframe computers.
Restart after repair requires a more elaborate state saving.

Standby relies on the existence of a stable storage in which the state of the computation is guarded, either in a non-volatile memory (Non-Volatile RAM, disk) or in a fail-independent memory (which can be the workspace of the spare machine).

Standby requires a periodic checkpointing to keep the stable storage up-to-date.
There is always a lag between the state of computations and the state of stable storage, because of the checkpointing interval or because of aynchronous input/outputs.

Recovery

It is not sufficient that a back-up unit exists, it must be loaded with the same data and be in a state as near possible to the state of the on-line unit.

The actualisation of the back-up assumes that computers are deterministic and identical machines.

"Given two identical machines, initially in the same state, the states of these machines will follow each other provided they always act on the same inputs, received in the same sequence."

workby

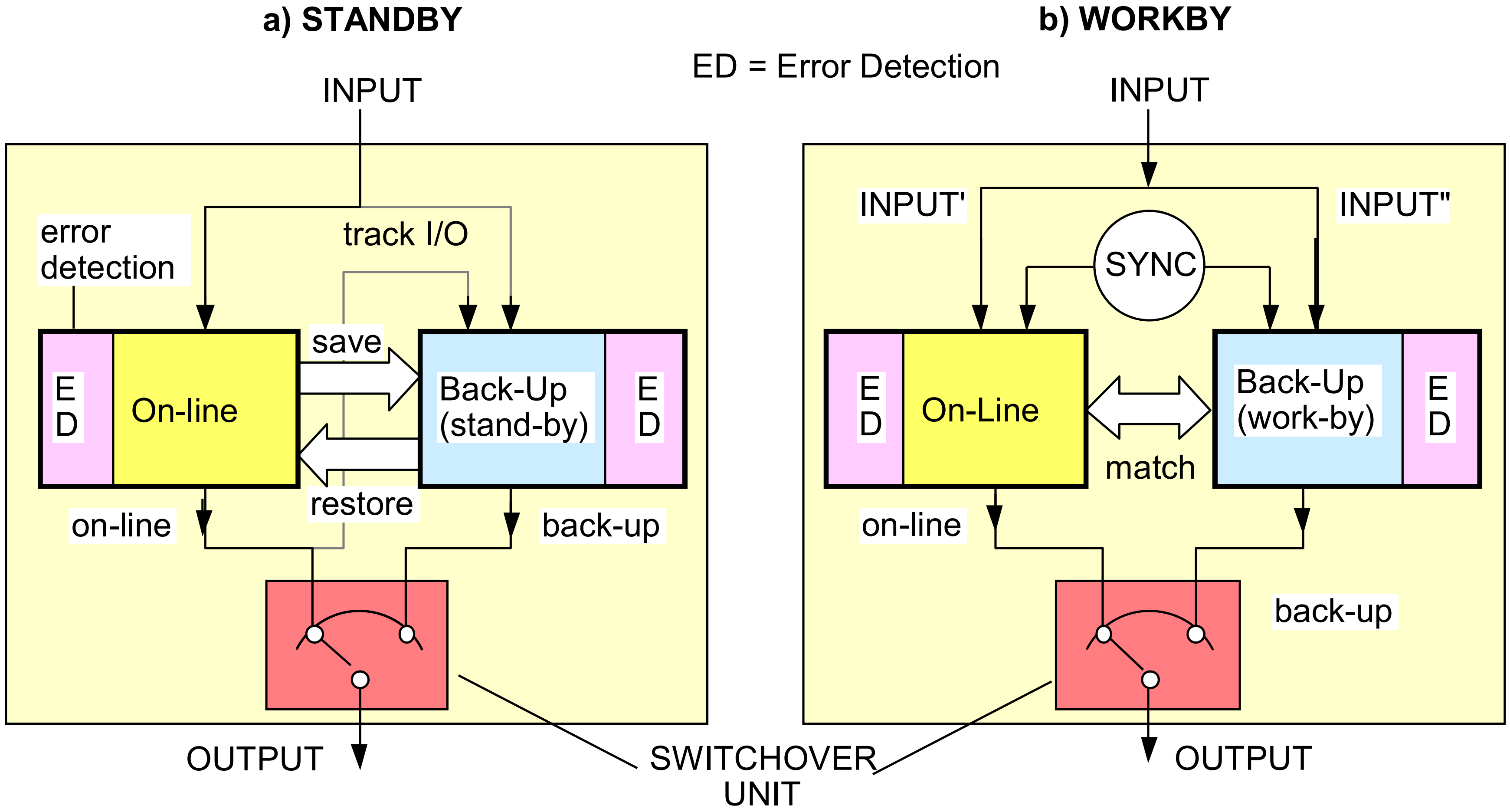
both machines are fed with the same, synchronized inputs and modify their states based on these inputs only in the same manner

standby

the on-line unit regularly copies its state and its inputs to the back-up.

OFF-LINE ACTUALIZATION (cold standby): irrelevant for process control, except for the reintegration of repaired units.

Comparison: Standby and Workby Computers



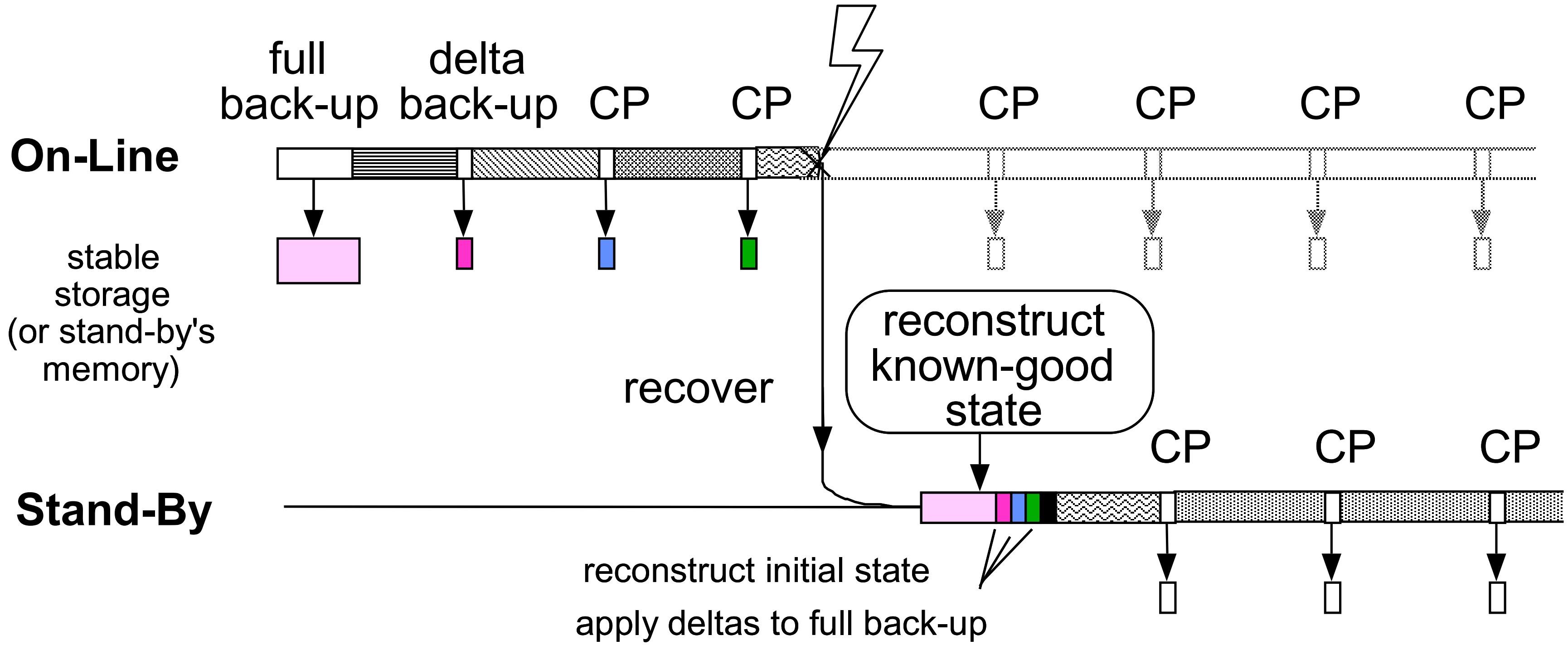
The on-line unit regularly actualises the state of the stand-by unit, which otherwise remains passive.

On-line unit and Back-up execute the same programs at (about) the same time. They are tightly synchronized.

Checkpointing

Saving enough information to reconstruct a previous, known-good state.

To limit the data to save (checkpoint duration, distance between checkpoints), only the parts of the state modified since last checkpoint are saved.

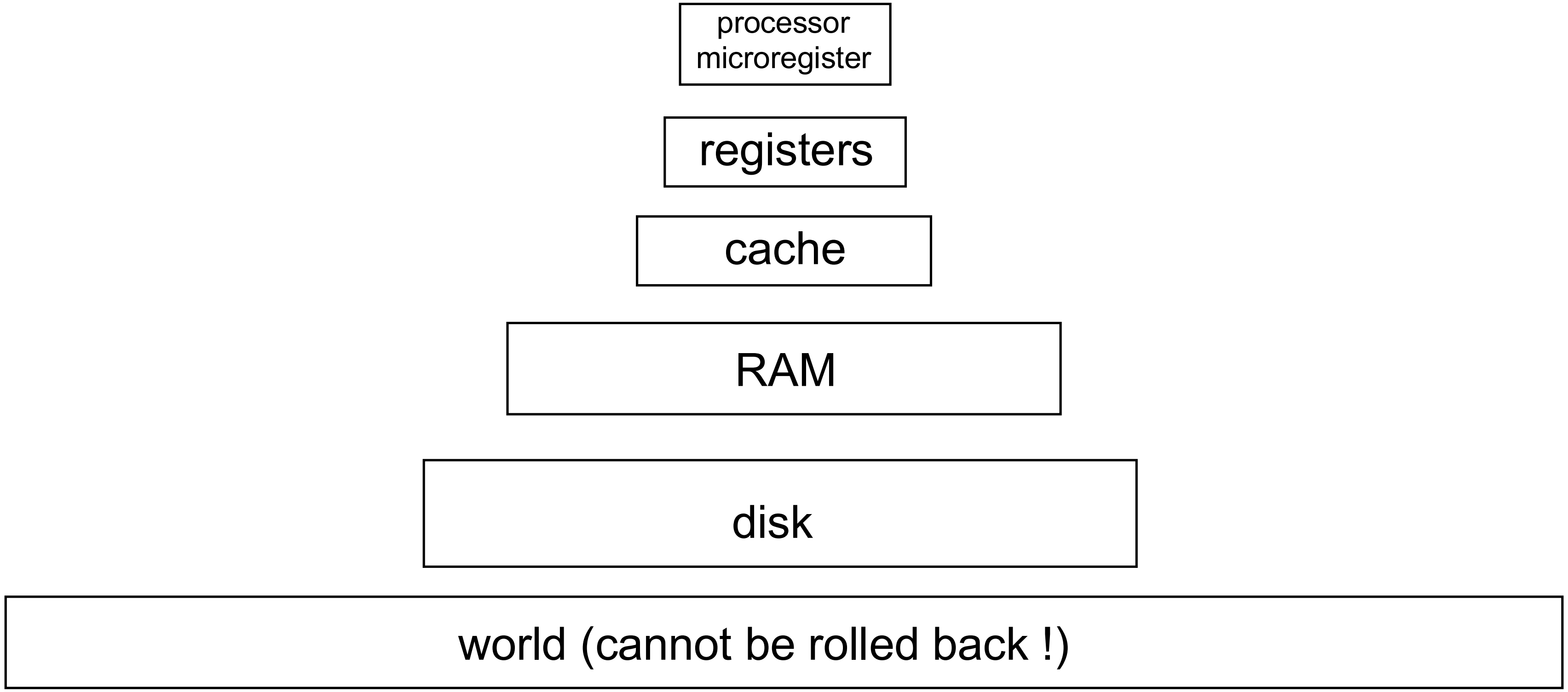


Checkpointing requires identification of the parts of the context modified since last checkpoint - application dependency !

To speed up recovery, the stand-by can apply the deltas to its state continuously.

Checkpointing

The amount of data to save to reconstruct a previous known-good state depend on the instant the checkpoint is taken.



Recovery depends on which parts of the state are trusted after a crash = stable storage , and which are not (volatile storage)
and on which parts are relevant.

Checkpointing Strategy

Checkpointing is difficult to provide automatically, unless every change to the trusted storage is monitored. This requires additional hardware (e.g. bus spy). Many times, the changes cannot be controlled since they take place in cache.

The amount of relevent information depends on the checkpoint location:

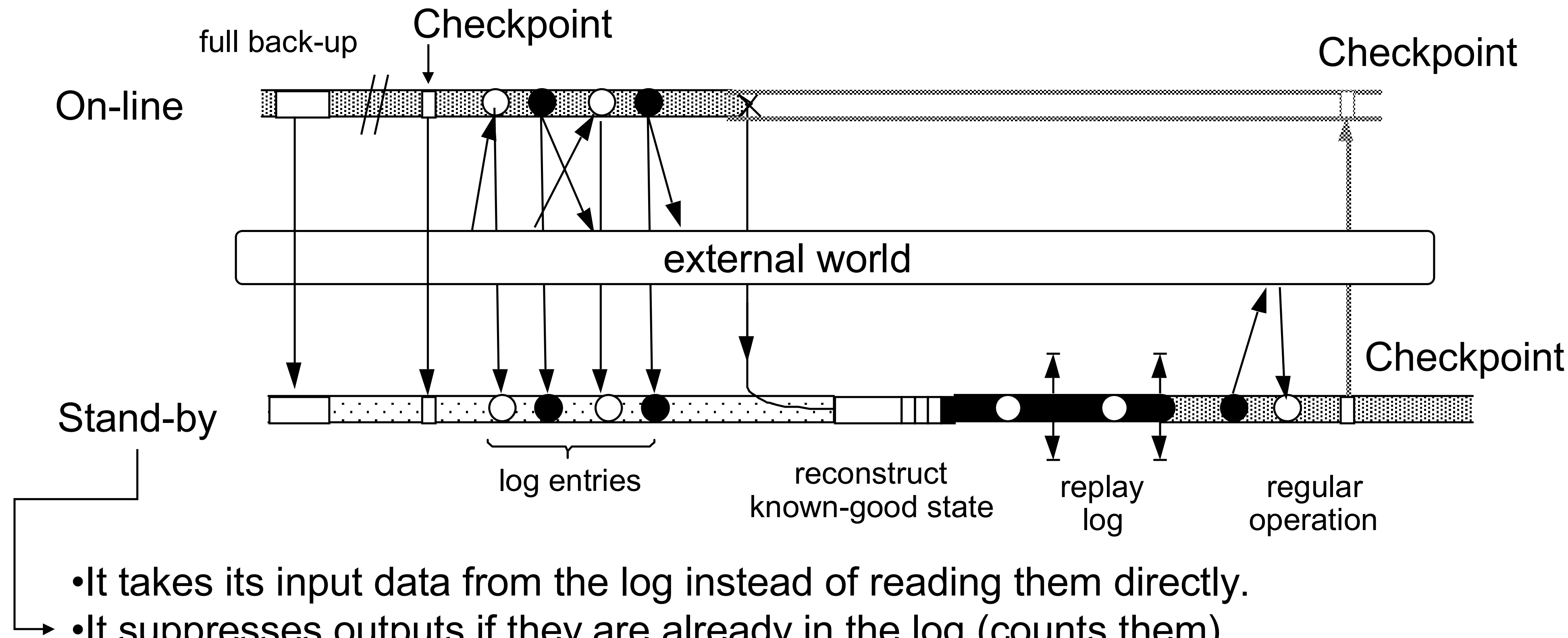
- after the execution of a task, its workspace is not anymore relevant.
- after the execution of a procedure, its stack is not anymore relevant
- after the execution of an instruction, microregisters are no more relevant.

Therefore, an efficient checkpointing requires that the application tags the data to save and decide on the checkpoint location.

Problem: how to keep control on the interval between checkpoints if the execution time of the programs is unknown ?

Logging

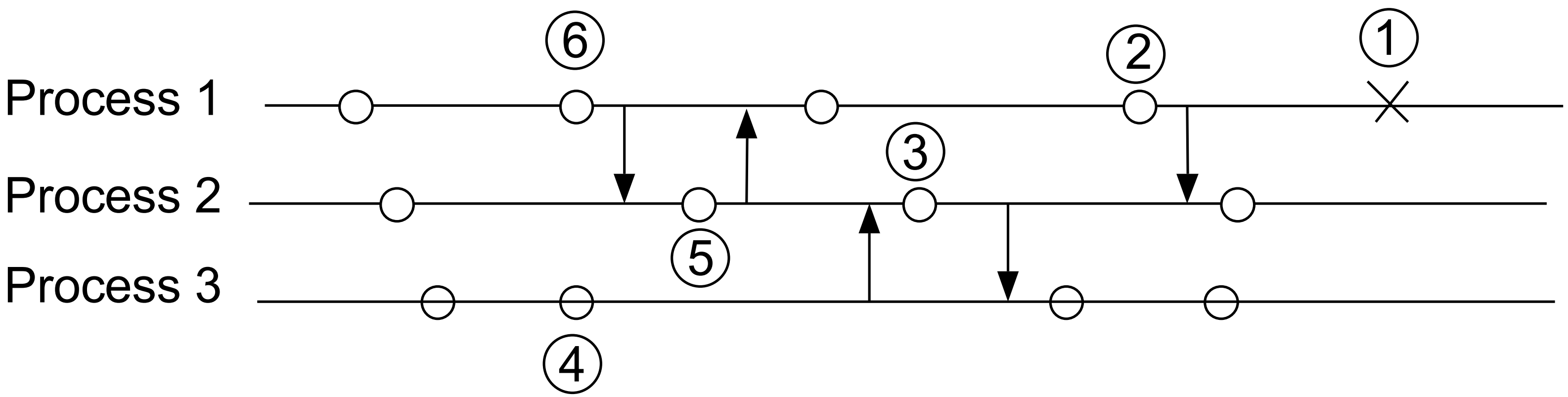
For faster recovery and closer checkpointing, the stand-by monitors the input-output interactions of the on-line unit in a log (fifo).
After reconstructing a know-good state, the stand-by resumes computation and applies the log of interactions to it:



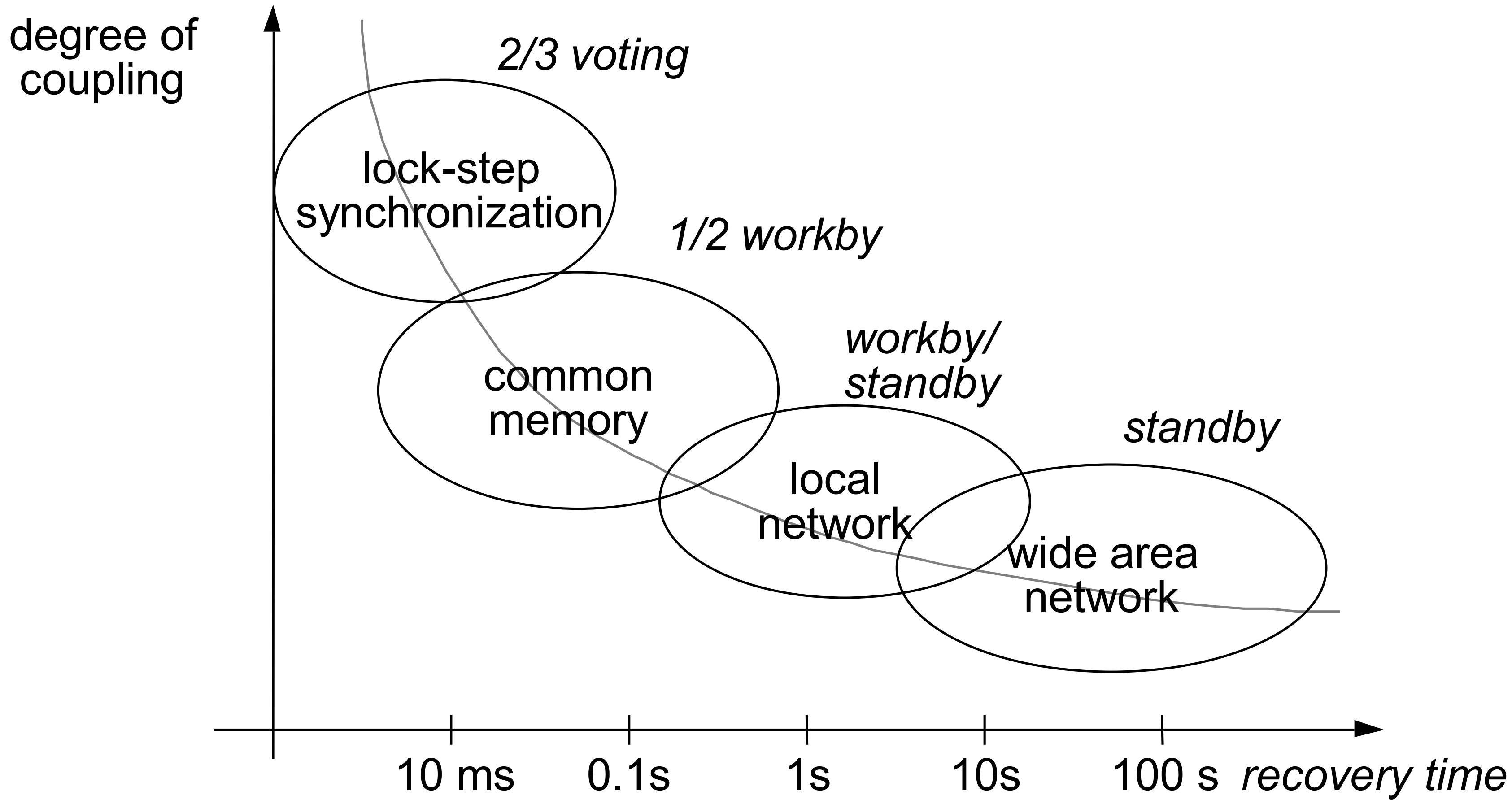
- It takes its input data from the log instead of reading them directly.
- It suppresses outputs if they are already in the log (counts them)
- It resumes normal computations when the log is void.

Domino Effect

As long as a failed unit does not communicate with the outer world, there is no harm.
The failure of a unit can oblige to roll back another unit which did not fail,because it acted on incorrect data.
This roll-back can propagate under evil circumstances ad infinitum (Domino-effect)
This effect can be easily prevented by placing the checkpoints in function of communication - each communication point should be a checkpoint.



Recovery Times for Various Architectures



The time available for recovery depends on the tolerance of the plant against outages.

When this time is long enough, stand-by operation becomes possible

9.4.5 Example Architectures

9.4.1 Error detection and fail-silent computers
- check redundancy
- duplication and comparison

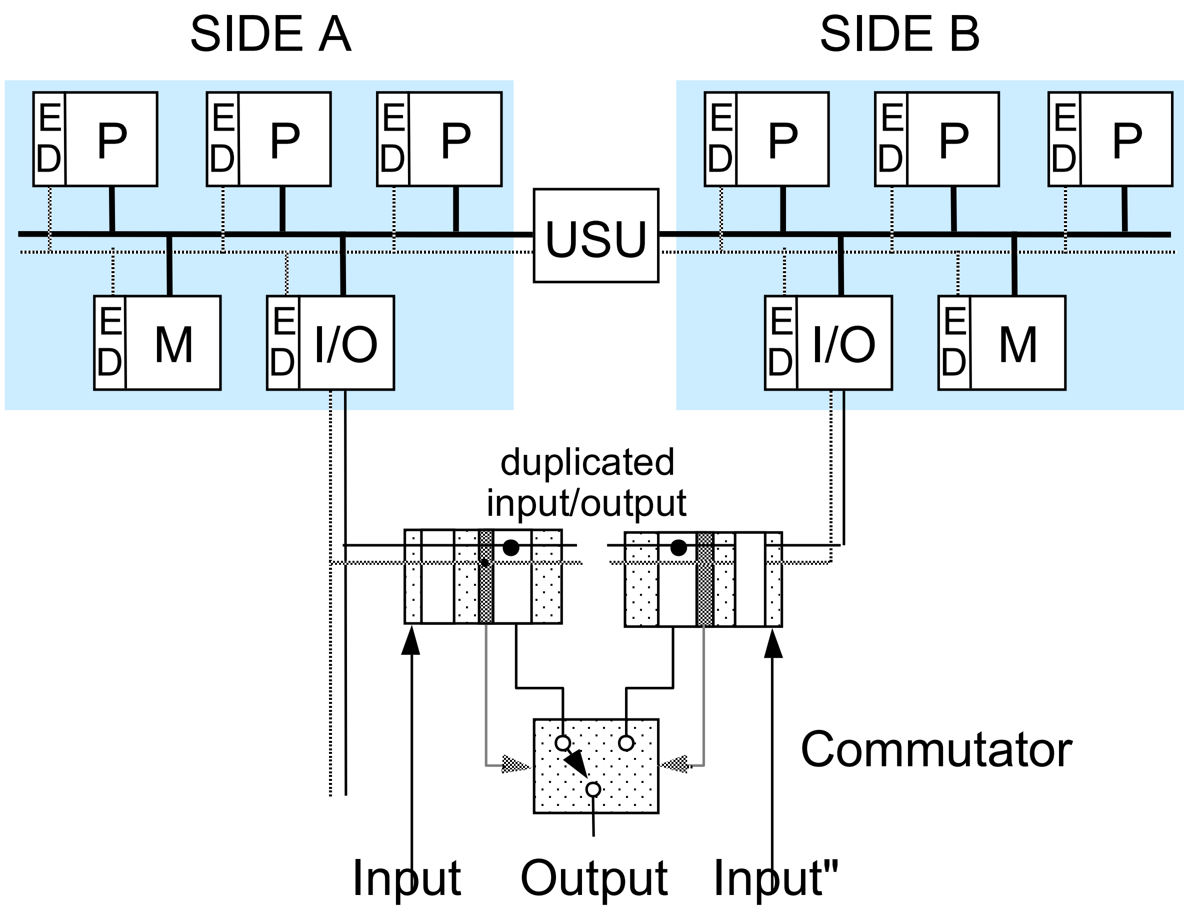
9.4.2 Fault-Tolerant Structures

9.4.3 Issues in Workby operation
- Input Processing
- Synchronization
- Output Processing

9.4.4 Standby Redundancy Structures
- Checkpointing
- Recovery

9.4.5 Examples of Dependable Architectures
- ABB dual controller
- Boeing 777 Primary Flight Control
- Space Shuttle PASS Computer

ABB 1/2 Multiprocessor



Synchronizing multiprocessors means: synchronize processors with the peer processor, and pairs with other pairs.
The multiprocessor bus must support a deterministic arbitration.
The Update and Synchronization Unit USU enforces synchronous operation.

Redundant control system

Central repository
- Redundant 2oo3

Duplication of connectivity severs
- each maintains its own A&E and history log

Network
- Dual lines, dual interfaces,
  dual ports on controller CPU

Controller CPU
- Hot standby, 1oo2

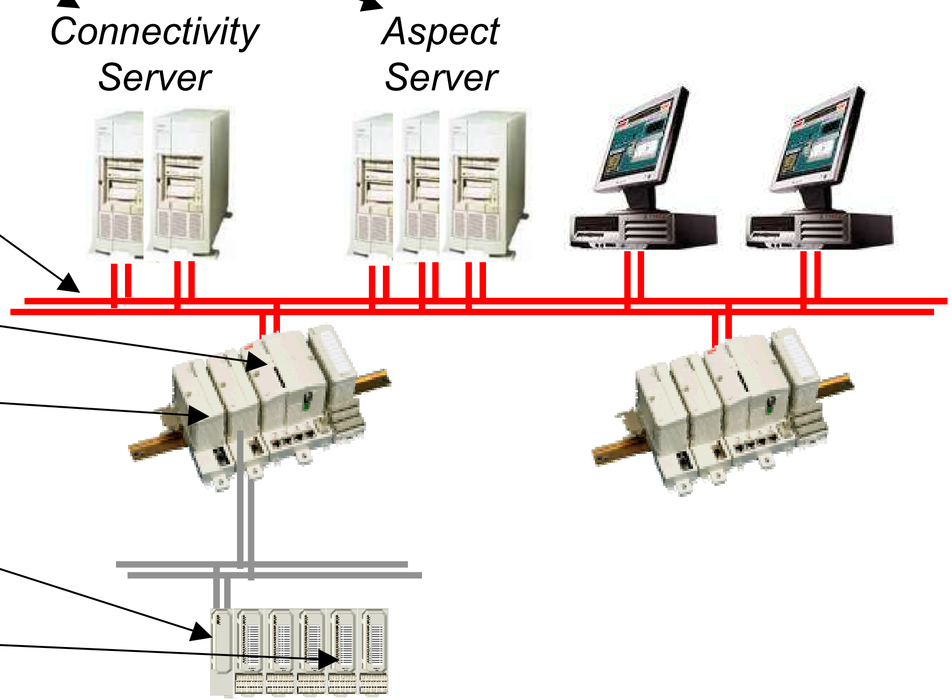
PROFIBUS DP/V1 line redundancy
- Single bus interface, dual lines

PROFIBUS DP/V1 slave redundancy
- S800, S900, dual bus interfaces

Redundant I/O, remote

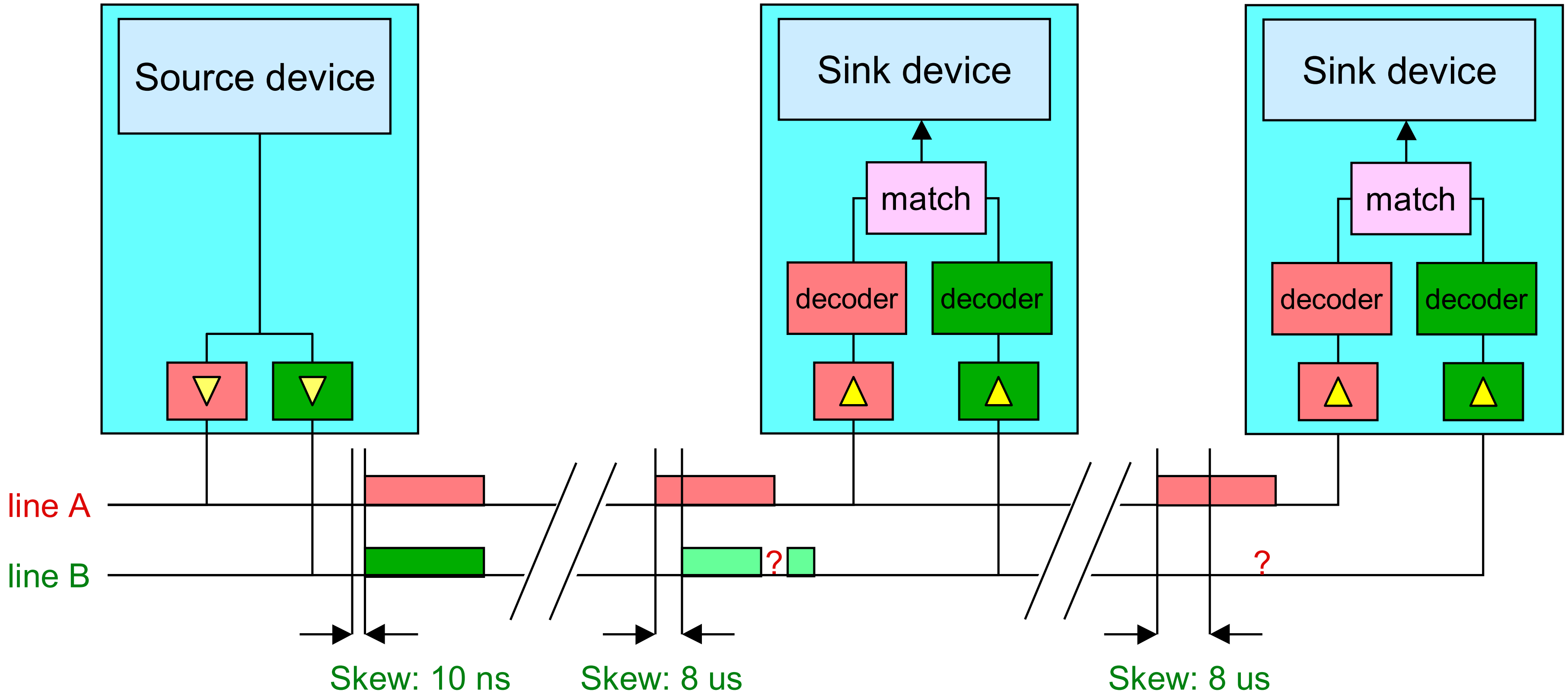
Dual power supplies
- Supervision of A and B power lines in
  AC 800M, S800 I/O, S900 I/O

Power back-up for workplaces and servers
- UPS (Uninterruptible Power Supply) technology



Bus line redundancy principle

- Principle: send on both, listen on both, take from one
- Skew between lines (repeaters,…) allowed
- Sequence number allows to track and ignore duplicates (not necessary for cyclic data)
- Duplicated complete decoder avoids systematic rejection of good frames
- Line redundancy is periodically checked
- Continuous transmitter fault limited to one repeater area



B777: airplane

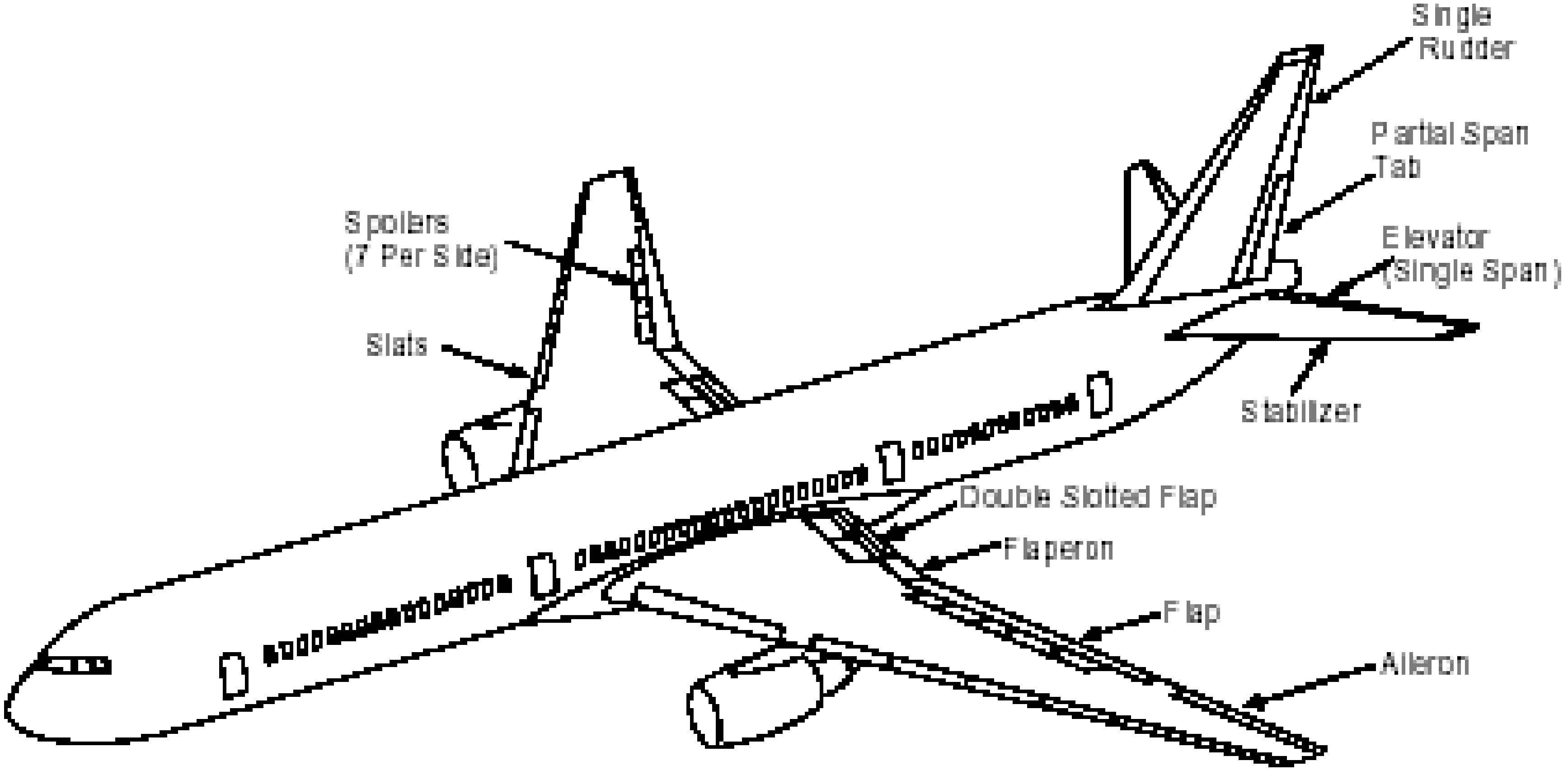
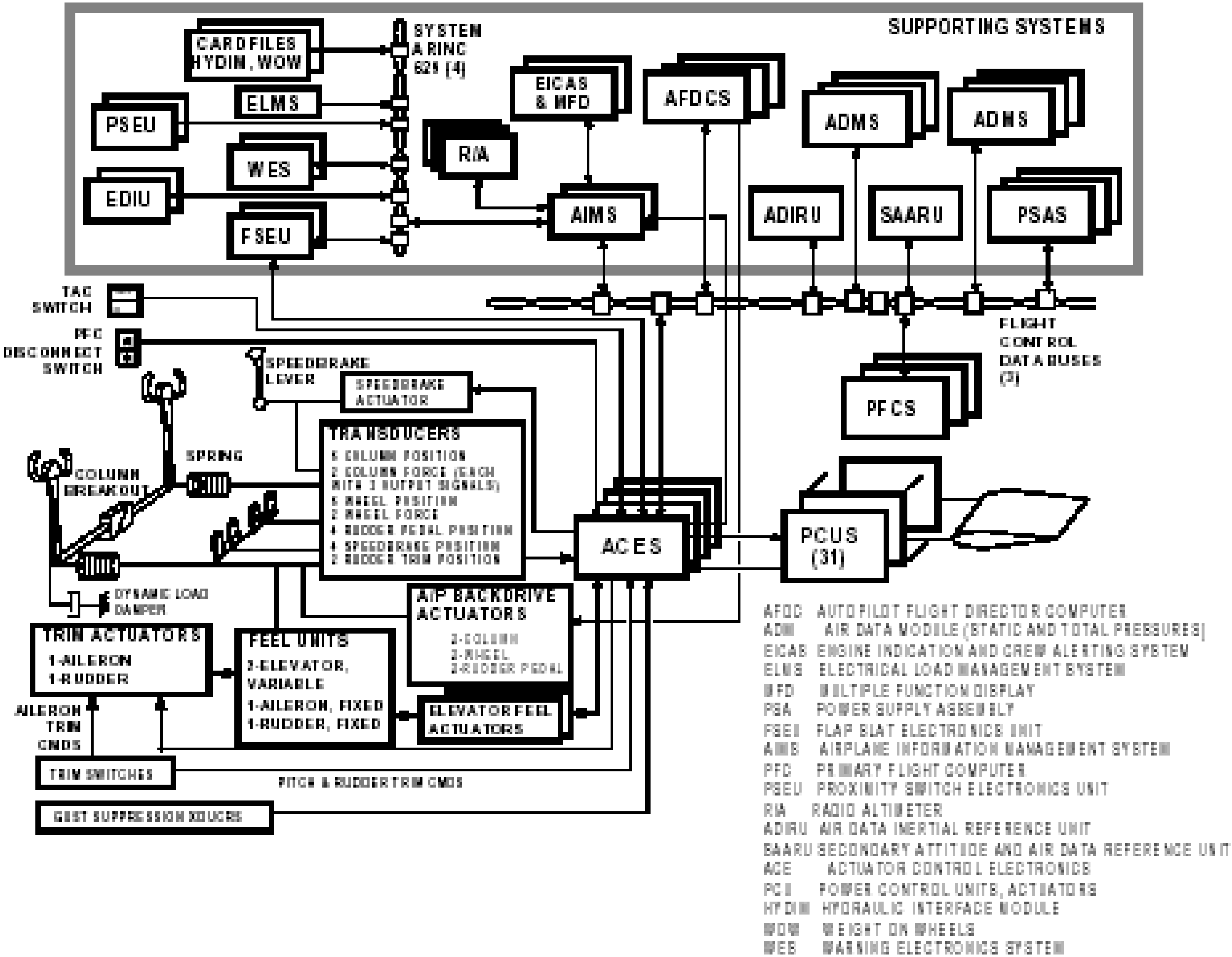
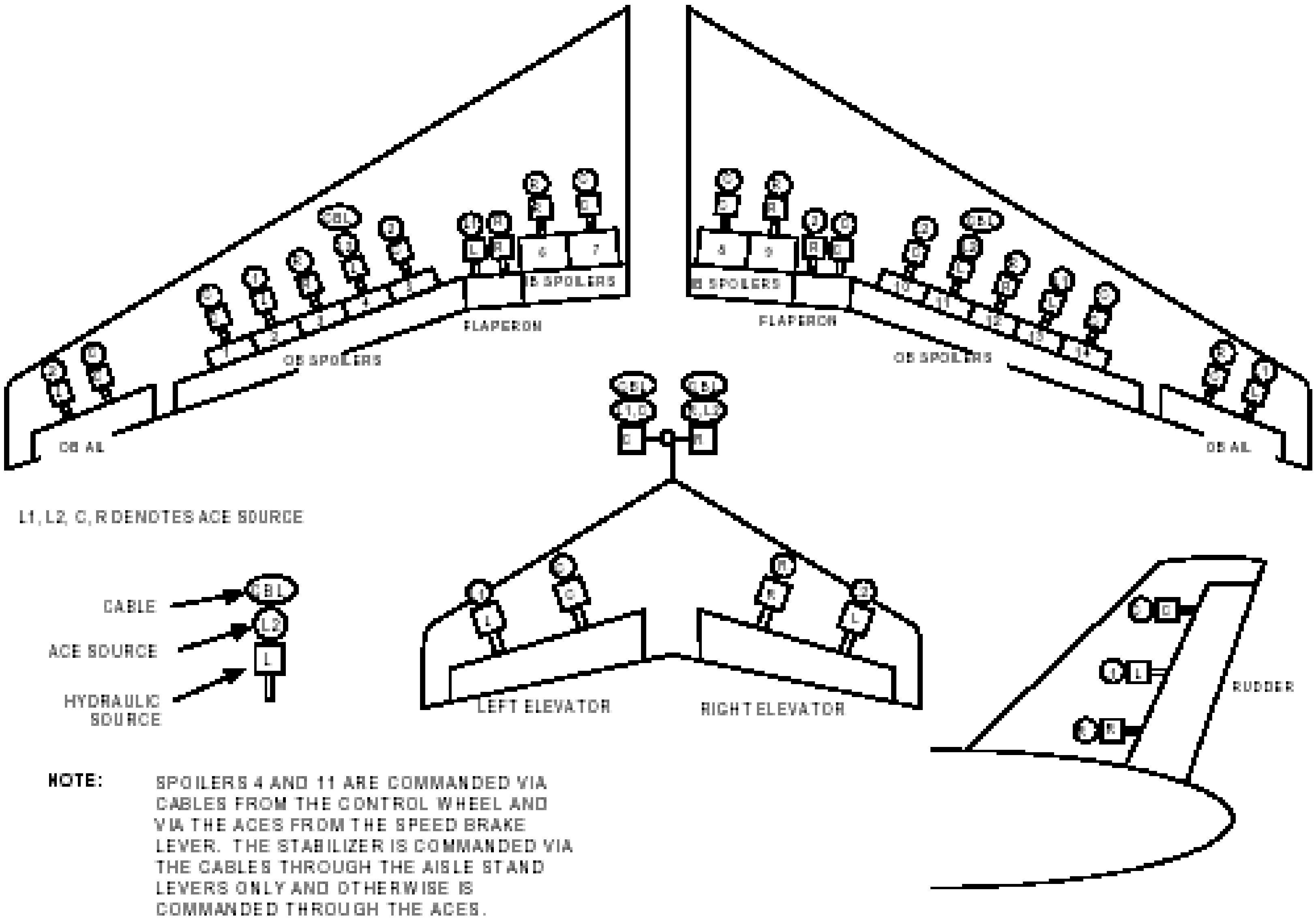


FIGURE 1 777 FLIGHT CONTROL SURFACES

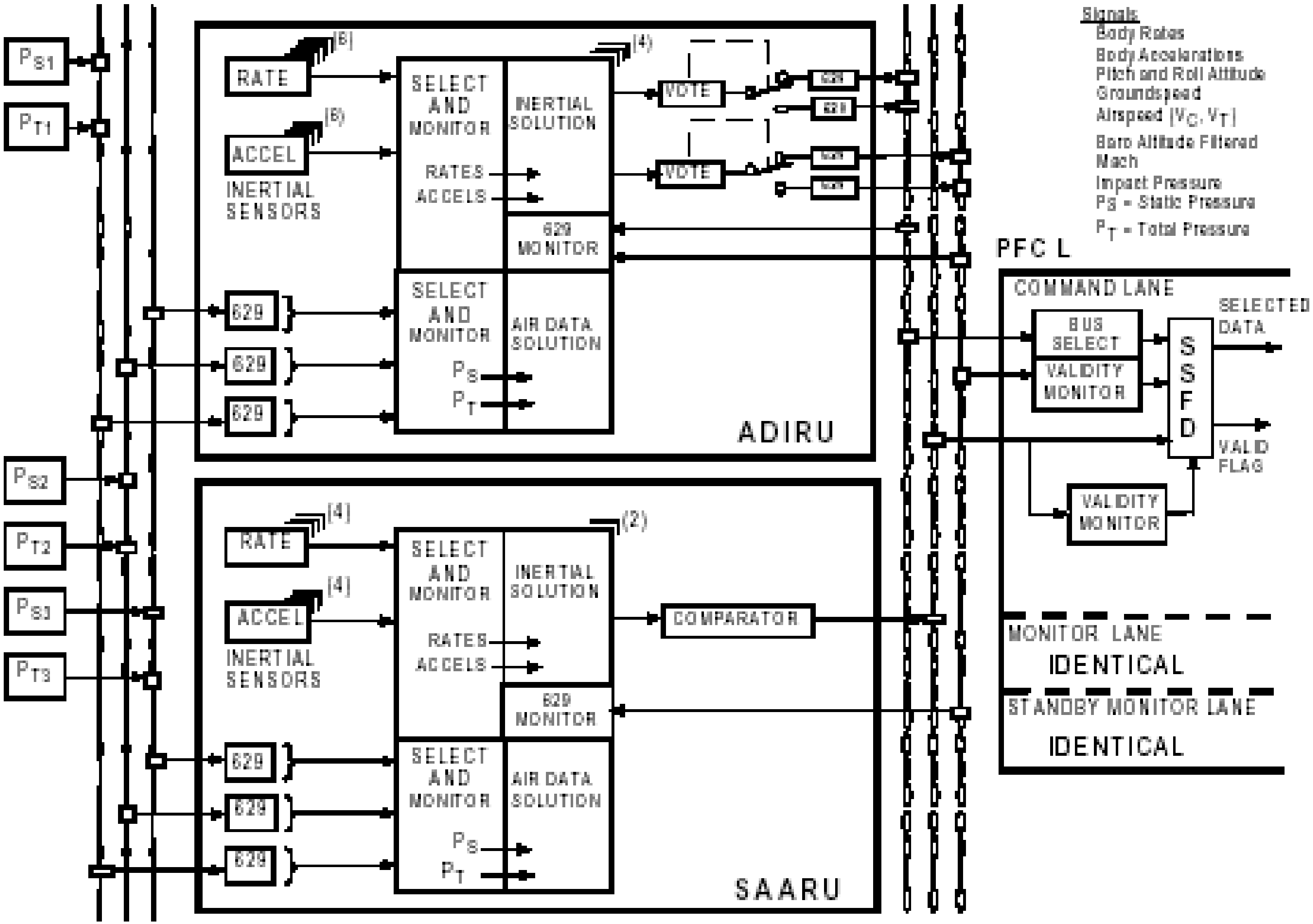
B777 control architecture



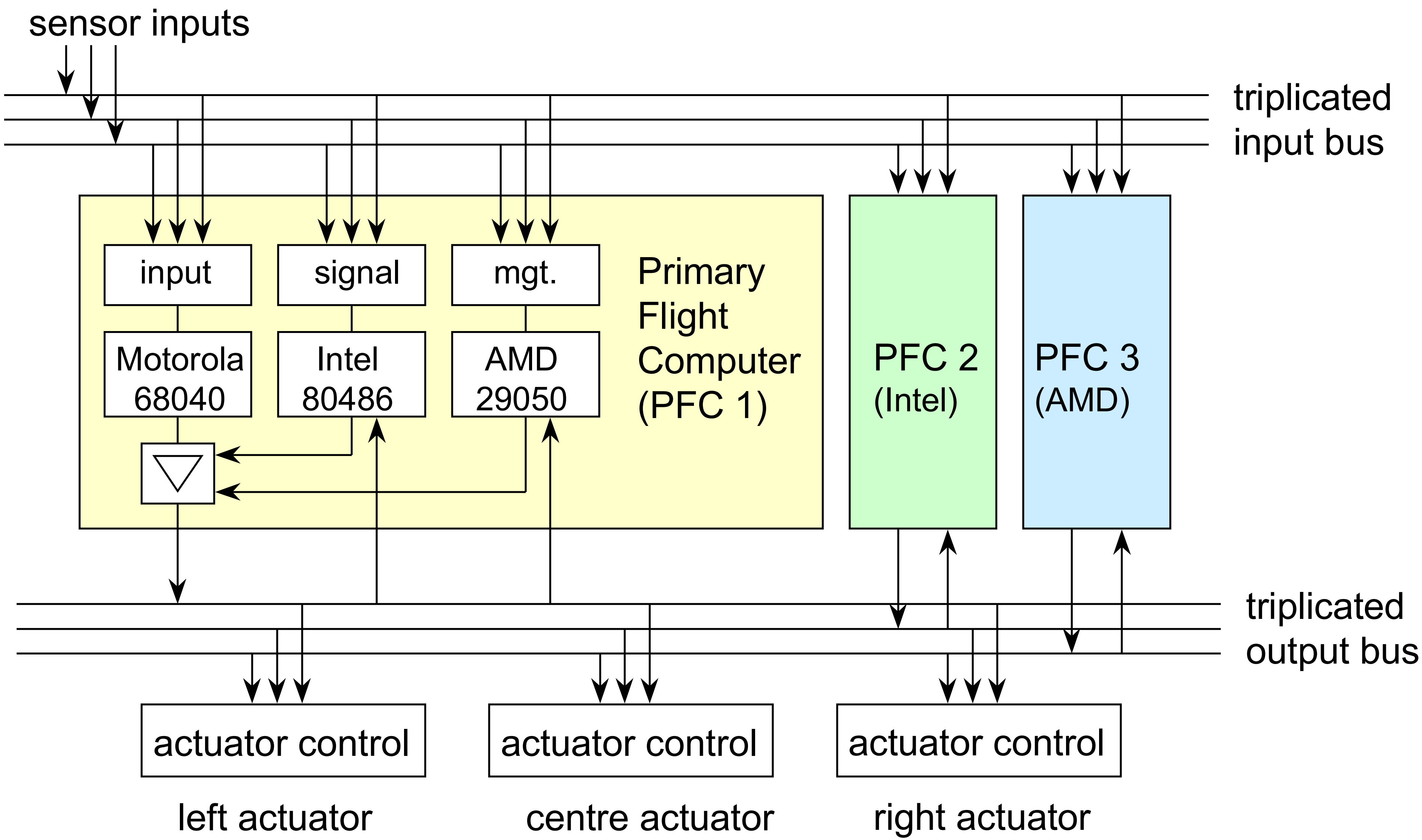
B777 control surfaces



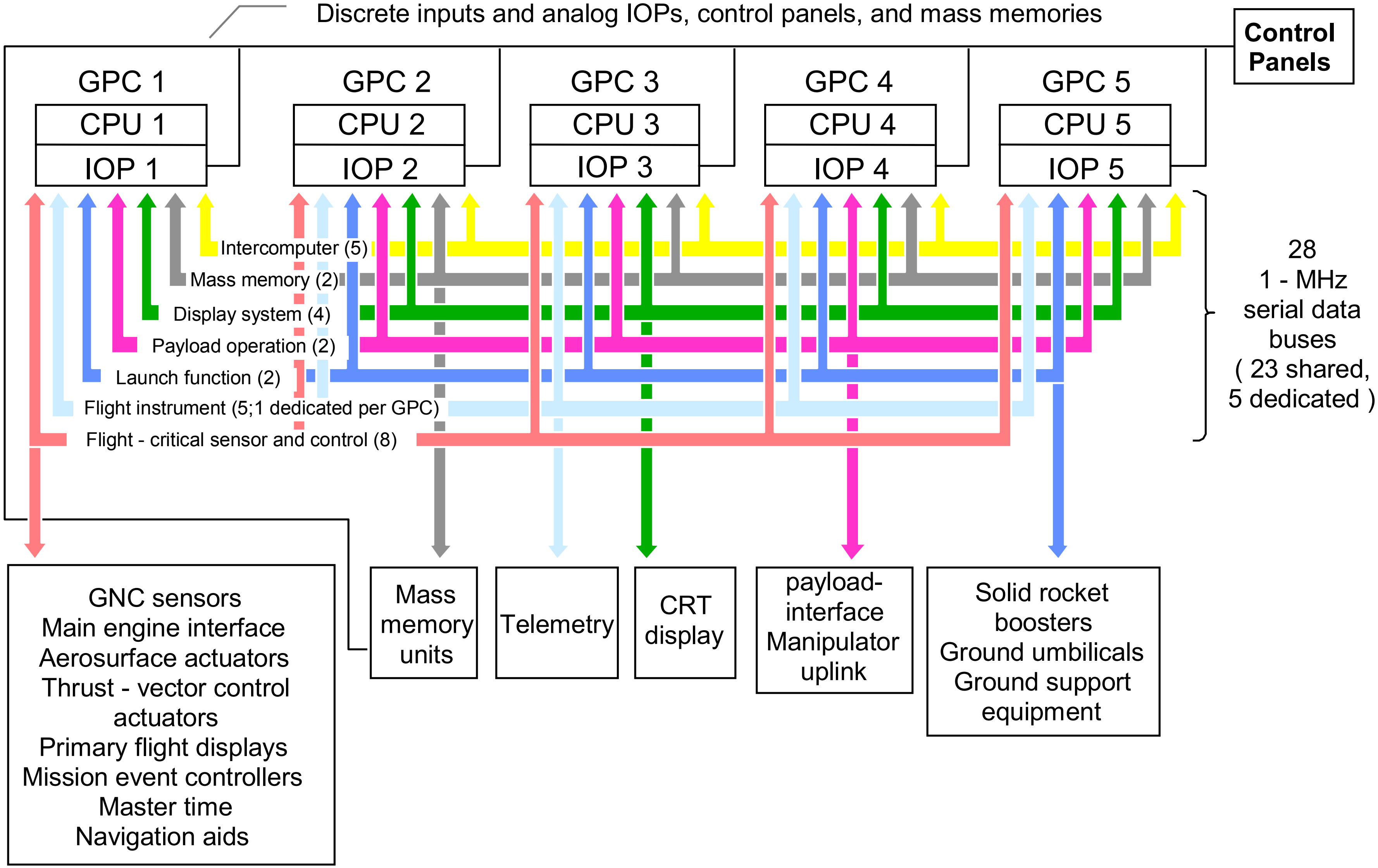
B777 Modules



B777 Primary Flight Control



Space Shuttle PASS Computer



Wrap-up

Fault-tolerant computers offer a finite increase in availability (safety)

All fault-tolerant architectures suffer from the following weaknesses:

- assumption of no common mode of error
  hardware: mechanical, power supply, environment,
  software: no design errors

- assumption of near-perfect coverage to avoid lurking errors and ensure fail-silence.

-assumption of short repair and maintenance time

-increased complexity with respect to the 1oo1 solution

ultimately, the question is that of which risk is society willing to accept.

9.5 Dependable Software
Logiciel fiable
Verlässliche Software

Prof. Dr. H. Kirrmann & Dr. B. Eschermann
ABB Research Center, Baden, Switzerland

2005-06-14, HK

Overview Dependable Software

9.5.1 Requirements on Software Dependability

- –Failure Rates
- –Physical vs. Design Faults

9.5.2 Software Dependability Techniques

- –Fault Avoidance and Fault Removal
- –On-line Fault Detection and Tolerance
  - –On-line Fault Detection Techniques
  - –Recovery Blocks
  - –N-version Programming
  - –Redundant Data

9.5.3 Examples

- –Automatic Train Protection
- –High-Voltage Substation Protection

Requirements for Safe Computer Systems

Required failure rates according to the standard IEC 61508:

safety integrity level	control systems [per hour]	protection systems [per operation]
4	$\geq 10^{-9}$ to $< 10^{-8}$	$\geq 10^{-5}$ to $< 10^{-4}$
3	$\geq 10^{-8}$ to $< 10^{-7}$	$\geq 10^{-4}$ to $< 10^{-3}$
2	$\geq 10^{-7}$ to $< 10^{-6}$	$\geq 10^{-3}$ to $< 10^{-2}$
1	$\geq 10^{-6}$ to $< 10^{-5}$	$\geq 10^{-2}$ to $< 10^{-1}$

most safety-critical systems
(e.g. railway signalling)

< 1 failure every 10 000 years

Software Problems

Did you ever see software that did not fail once in 10 000 years

(i.e. it never failed during your lifetime)?

- First space shuttle launch delayed due to software synchronisation problem, 1981 (IBM).
- Therac 25 (radiation therapy machine) killed 2 people due to software defect leading to massive overdoses in 1986 (AECL).
- Software defect in 4ESS telephone switching system in USA led to loss of $60 million due to outages in 1990 (AT&T).
- Software error in Patriot equipment: Missed Iraqi Scud missile in Kuwait war killed 28 American soldiers in Dhahran, 1991 (Raytheon).
- ... [add your favourite software bug].

The Patriot Missile Failure

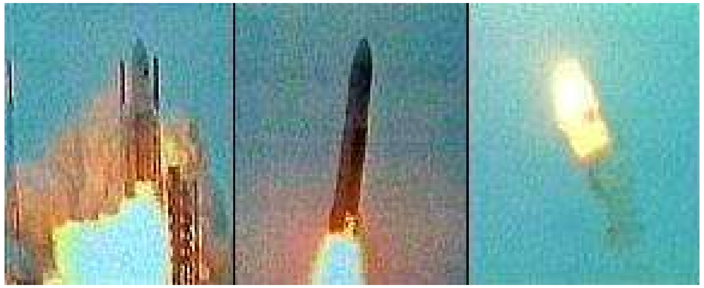
The Patriot Missile failure in Dharan, Saudi Arabia, on February 25, 1991 which resulted in 28 deaths, is ultimately attributable to poor handling of rounding errors.
On February 25, 1991, during the Gulf War, an American Patriot Missile battery in Dharan, Saudi Arabia, failed to track and intercept an incoming Iraqi Scud missile. The Scud struck an American Army barracks, killing 28 soldiers and injuring around 100 other people.

A report of the General Accounting office, GAO/IMTEC-92-26, entitled *Patriot Missile Defense: Software Problem Led to System Failure at Dhahran, Saudi Arabia* analyses the causes (excerpt):

"The range gate's prediction of where the Scud will next appear is a function of the Scud's known velocity and the time of the last radar detection.
Velocity is a real number that can be expressed as a whole number and a decimal (e.g., 3750.2563...miles per hour).
Time is kept continuously by the system's internal clock in tenths of seconds but is expressed as an integer or whole number (e.g., 32, 33, 34...).
The longer the system has been running, the larger the number representing time. To predict where the Scud will next appear, both time and velocity must be expressed as real numbers. Because of the way the Patriot computer performs its calculations and the fact that its registers are only 24 bits long, the conversion of time from an integer to a real number cannot be any more precise than 24 bits. This conversion results in a loss of precision causing a less accurate time calculation. The effect of this inaccuracy on the range gate's calculation is directly proportional to the target's velocity and the length of the system has been running. Consequently, performing the conversion after the Patriot has been running continuously for extended periods causes the range gate to shift away from the center of the target, making it less likely that the target, in this case a Scud, will be successfully intercepted."

Ariane 501 failure

On June 4, 1996 an unmanned Ariane 5 rocket launched by the European Space Agency exploded just forty seconds after its lift-off from Kourou, French Guiana. The rocket was on its first voyage, after a decade of development costing $7 billion. The destroyed rocket and its cargo were valued at $500 million. A board of inquiry investigated the causes of the explosion and in two weeks issued a report.



http://www.ima.umn.edu/~arnold/disasters/ariane5rep.html
(no more available at the original site)

"The failure of the Ariane 501 was caused by the complete loss of guidance and attitude information 37 seconds after start of the main engine ignition sequence (30 seconds after lift-off). This loss of information was due to specification and design errors in the software of the inertial reference system.
The internal SRI software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer. "*
*SRI stands for Système de Référence Inertielle or Inertial Reference System.

Code was reused from the Ariane 4 guidance system. The Ariane 4 has different flight characteristics in the first 30 s of flight and exception conditions were generated on both inertial guidance system (IGS) channels of the Ariane 5. There are some instances in other domains where what worked for the first implementation did not work for the second.

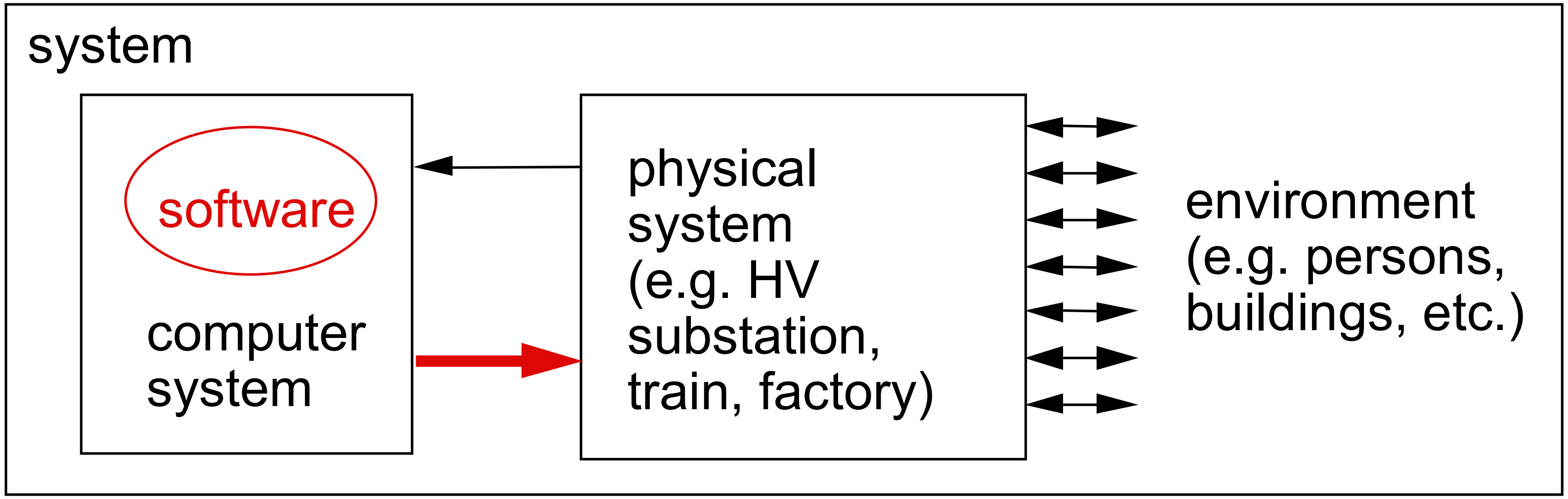
"Reuse without a contract is folly"
90% of safety-critical failures are requirement errors (a JPL study)

It begins with the specifications ....

A 1988 survey conducted by the United Kingdom's Health & Safety Executive (Bootle, U.K.) of 34 "reportable" accidents in the chemical process industry revealed that inadequate specifications could be linked to 20% (the #1 cause) of these accidents.

Software and the System

"Software by itself is never dangerous, safety is a system characteristic."



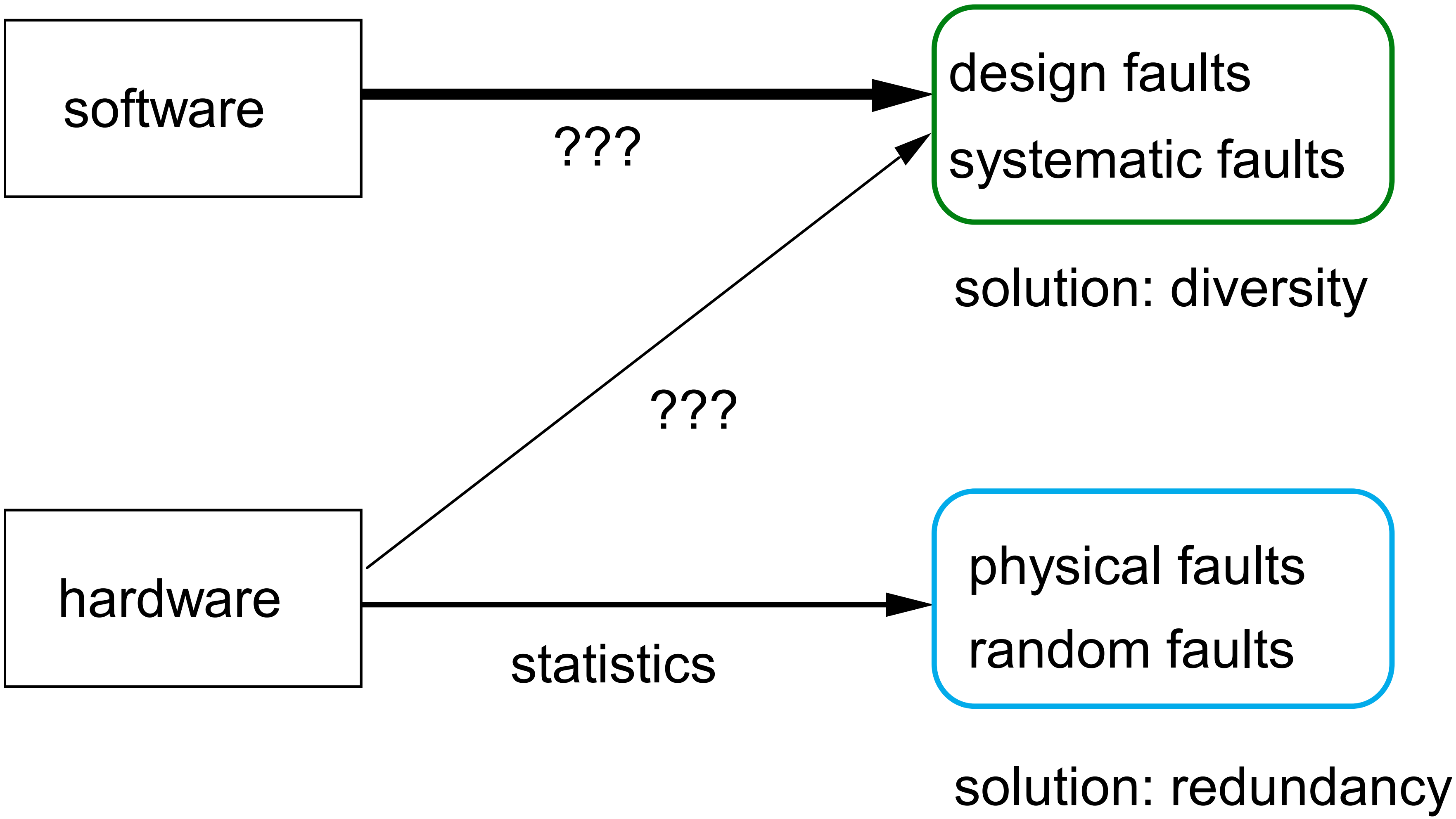
Fault detection: Safe state of physical system exists (fail-safe system).

Fault tolerance: No safe state exists.

Persistency: Computer always produces output (which may be wrong).

Integrity: Computer never produces wrong output (maybe no output at all).

Which Faults?



Fail-Safe Computer Systems

Approach 1: Layered

against
design faults

against
physical faults

- systematic
- flexible
- expensive

Approach 2: All in One

- less flexible
- less expensive
- clear safety responsibility

Software Dependability Techniques

1) Against design faults

–Fault avoidance → (formal) software development techniques

–Fault removal → verification and validation (e.g. test)

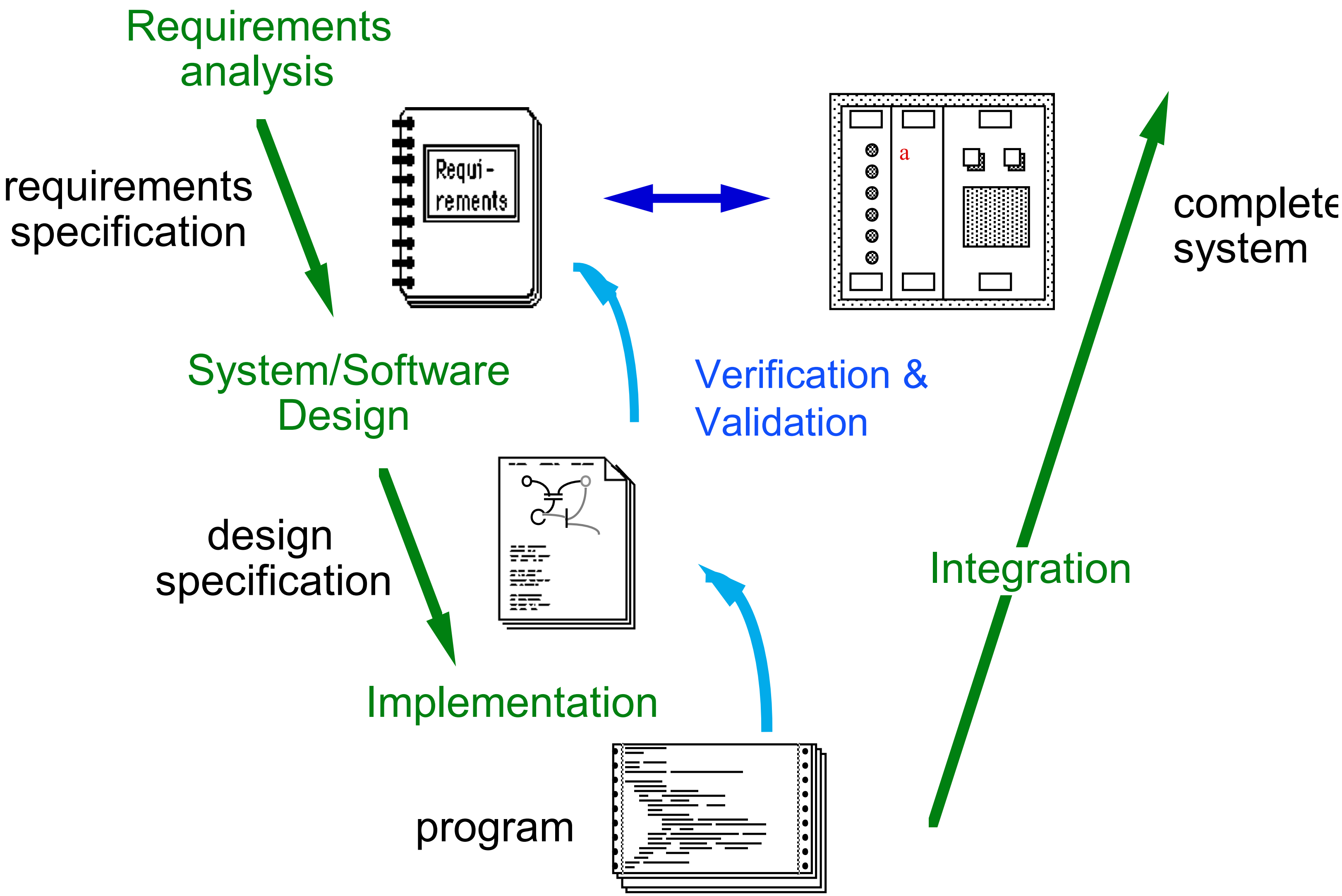
–On-line error detection and fault tolerance → design diversity

2) Against physical faults

–Fault detection and fault tolerance
(physical faults can not be detected and removed at design time)

- –Systematic software diversity (random faults definitely lead to different errors in both software variants)
- –Continuous supervision (e.g. coding techniques, control flow checking, etc.)
- –Periodic testing

Fault Avoidance and Fault Removal



Validation and Verification (V&V)

Validation: Do I develop the right solution?

Verification: Do I develop the solution right?

dynamic techniques

- test
- simulation

static techniques

- review
- proof

Test: Enough for Proving Safety?

How many (successful !) tests to show failure rate < limit ?

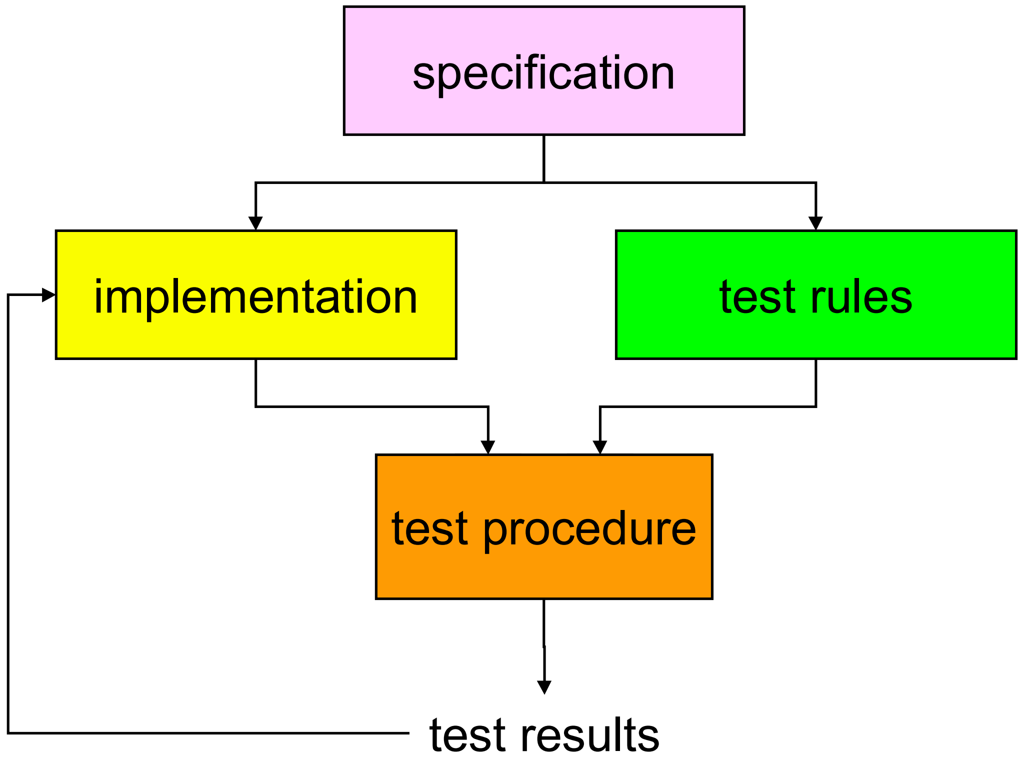
→ Depends on required confidence.

confidence level	minimal test length
95 %	3.00 / limit
99 %	4.61 / limit
99.9 %	6.91 / limit
99.99 %	9.21 / limit
99.999 %	11.51 / limit

Example: c = 99.99 % , failure rate 10^{-9}/h → test length > 1 million years

Testing

Testing requires a test specification, test rules (suite) and test protocol



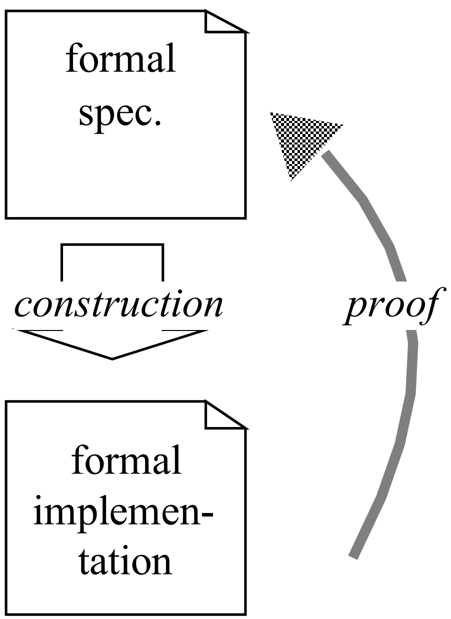
Testing can only reveal errors, not demonstrate their absence ! (Dikstra)

Simulation: Tools and Languages

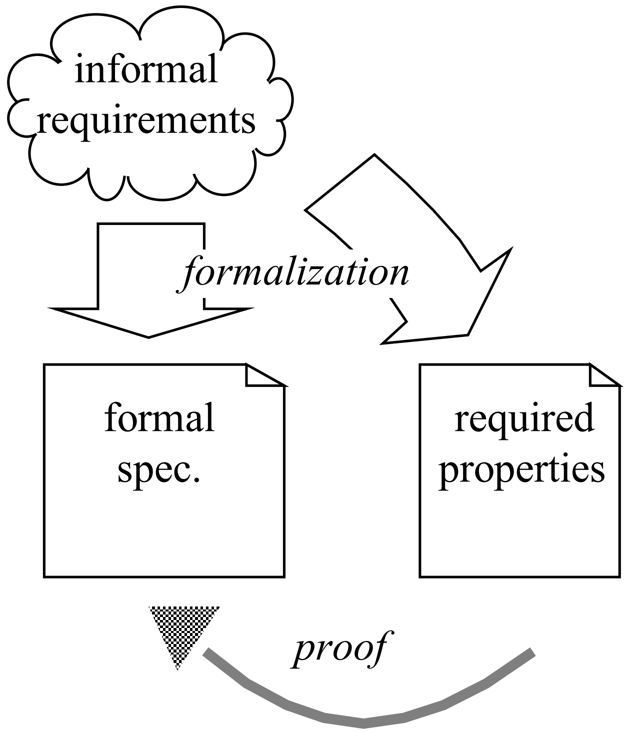
	SDL	LOTOS	Esterel	Statecharts
graphical syntax	3	3	–	3
syntax analysis, static checks	3	3	3	3
interactive simulation	3	3	3	3
deterministic simulation	3	3	?	3
stochastic simulation	–	?	–	3
code generation	C	C	C	C, Ada

Formal Proofs

Implementation Proofs



Property Proofs



Formal Languages and Tools

	mathematical foundation	example tools
VDM	dynamic logic (pre- and postconditions)	• Mural from University of Manchester • SpecBox from Adelard
Z	predicate logic, set theory	• ProofPower from ICL Secure Systems • DST-fuzz from Deutsche System Technik
SDL	finite-state machines	• SDT from Telelogic • Geode from Verilog
LOTOS	process algebra	• The LOTOS Toolbox from Information Technology Architecture B.V.
NP	propositional logic	• NP-Tools from Logikkonsult NP

Dilemma:

Either the language is not very powerful,

or the proof process cannot be easily automated.

On-line Error Detection by N-Version programming

N-Version programming is the software equivalent of massive redundancy (workby)

"detection of design errors on-line by diversified software, independently programmed in different languages by independent teams, running on different computers, possibly of different type and operating system".

Difficult to ensure that the teams end up with comparable results, as most computations yield similar, but not identical results:

- rounding errors in floating-point arithmetic
  (use of identical algorithms)

- different branches taken at random
  (IF (T >100.0) THEN ...)

- equivalent representation (data formats)
  If (success = 0)….
  If success = TRUE
  If (success)…

Difficult to ensure that the teams do not make the same errors
(common school, and interpret the specifications in the same wrong way)

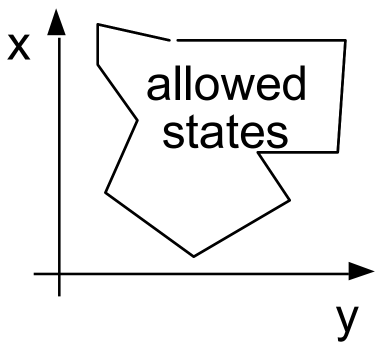
Acceptance Tests

Acceptance Test are invariants calculated at run-time

• definition of invariants in the behaviour of the software

• set-up of a "don't do" specification

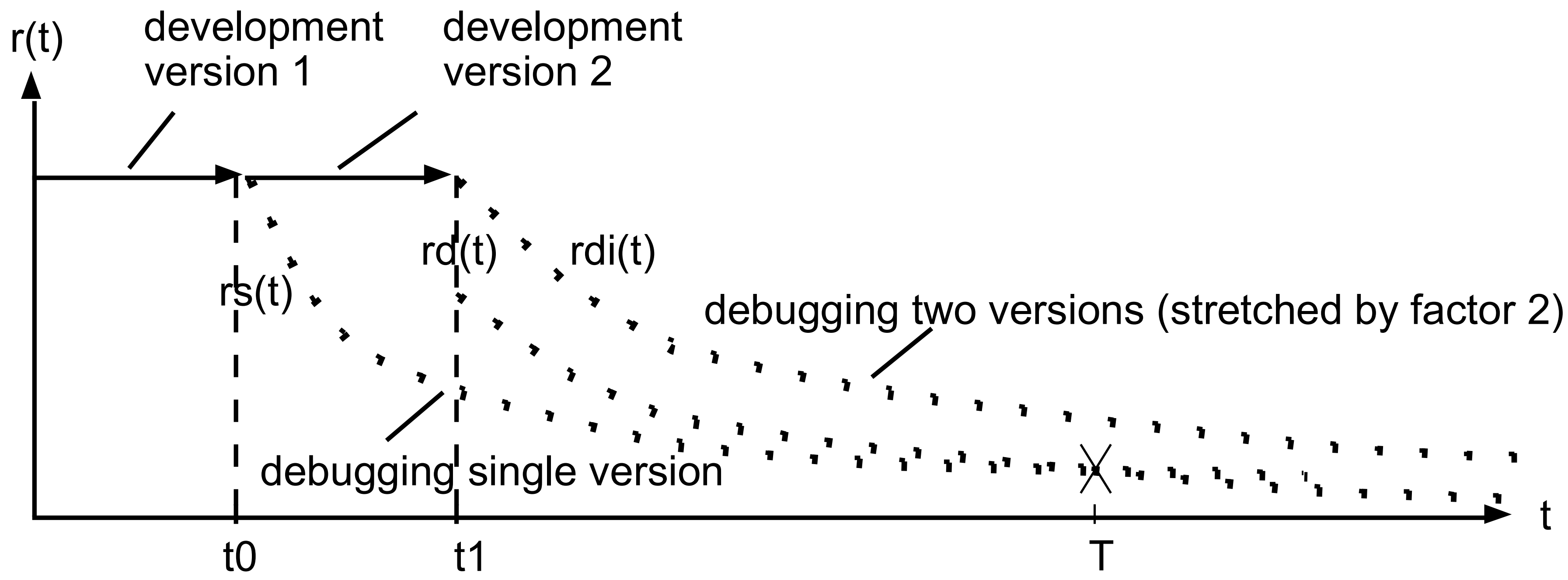
• plausibility checks included by the programmer of the task (efficient but cannot cope with surprise errors).



Cost Efficiency of Fault Removal vs. On-line Error Detection

Design errors are difficult to detect and even more difficult to correct on-line.
The cost of diverse software can often be invested more efficiently in off-line testing and validation instead.

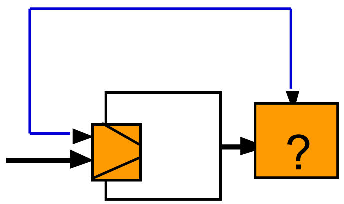
Rate of safety-critical failures (assuming independence between versions):



On-line Error Detection

- periodical tests

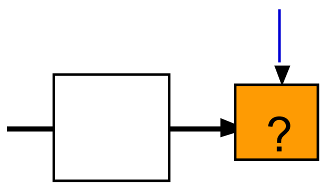
example test



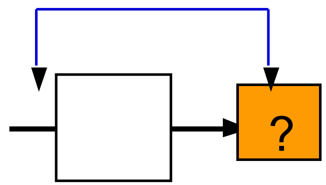
overhead

- continuous supervision

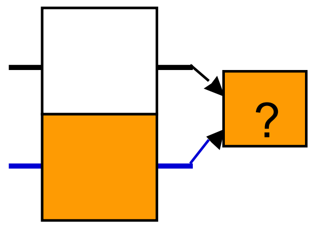
plausibility check



acceptance test



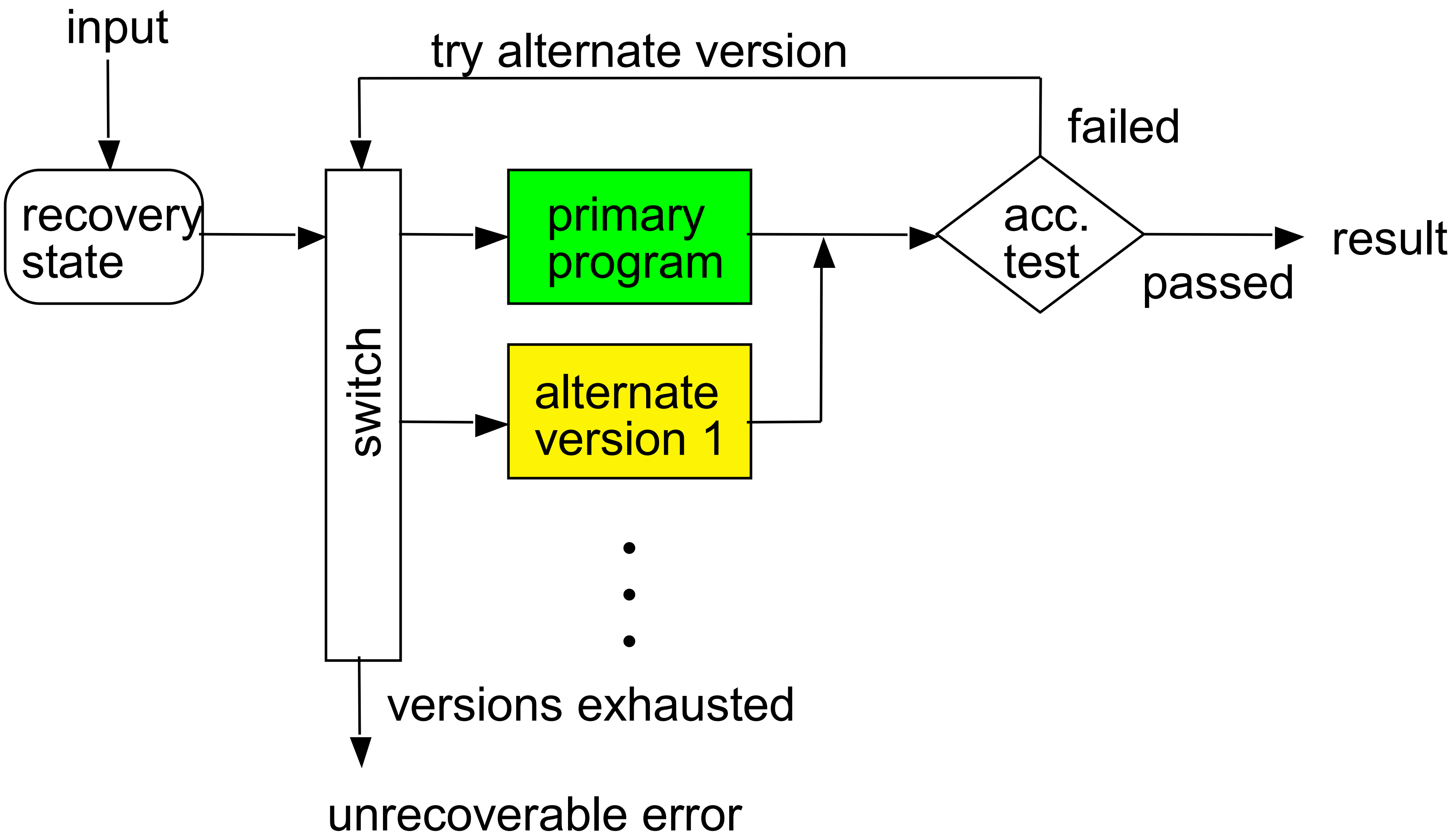
redundancy/diversity
hardware/software/time



Plausibility Checks / Acceptance Tests

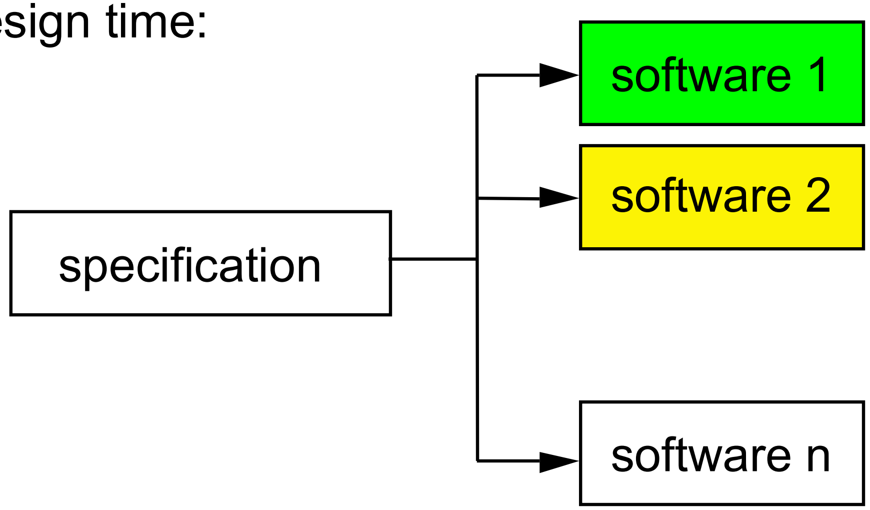
• range checks	`0 ≤ train speed ≤ 500`
	`safety assertions`
• structural checks	`given list length / last pointer NIL`
• control flow checks	`set flag; go to procedure; check flag`
	`hardware signature monitors`
• timing checks	`checking of time-stamps/toggle bits`
	`hardware watchdogs`
• coding checks	`parity bit, CRC`
• reversal checks	compute $y = \sqrt{x}$; check $x = y^2$

Recovery Blocks



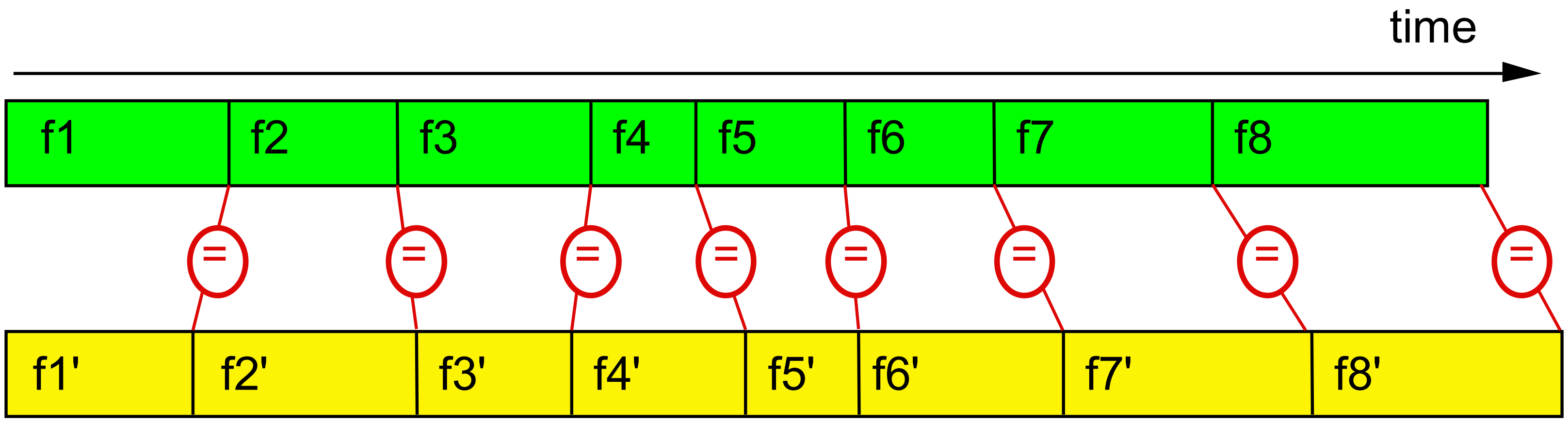
N-Version Programming (Design Diversity)

design time:



different teams
different languages
different data structures
different operating system
different tools (e.g. compilers)
different sites (countries)
different specification languages
• • •

run time:



Issues in N-Version Programming

- number of software versions (fault detection ↔ fault tolerance)
- hardware redundancy ↔ time redundancy (real-time !)
- random diversity ↔ systematic diversity
- determination of cross-check (voting) points
- format of cross-check values
- cross-check decision algorithm (consistent comparison problem !)
- recovery/rollback procedure (domino effect !)
- common specification errors (and support environment !)
- cost for software development
- diverse maintenance of diverse software ?

Consistent Comparison Problem

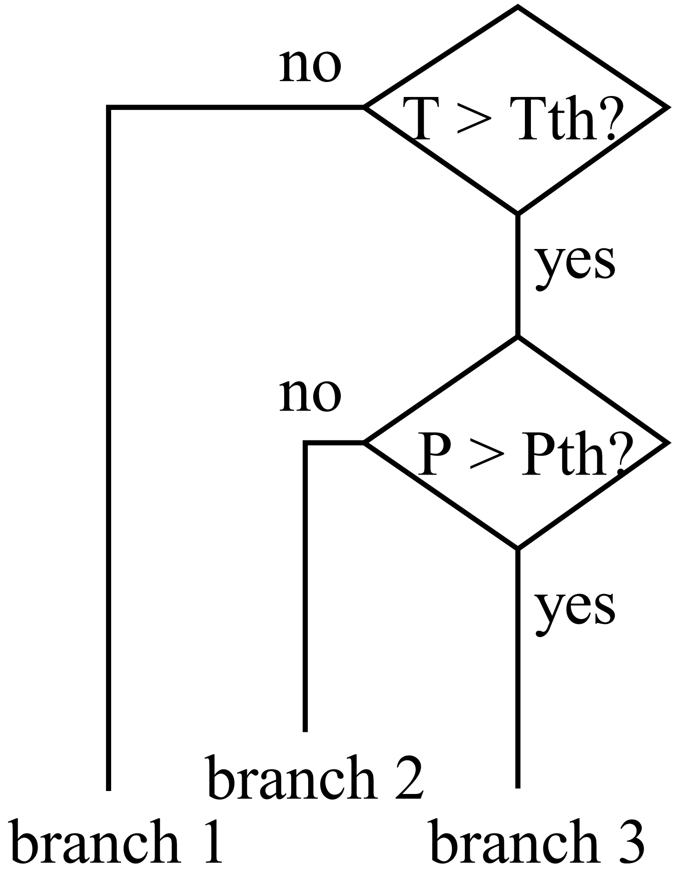
Problem occurs if floating point numbers are used.

Finite precision of hardware arithmetic
$\rightarrow$ result depends on sequence of computation steps.

Thus: Different versions may result in slightly different results
$\rightarrow$ result comparator needs to do "inexact comparisons"

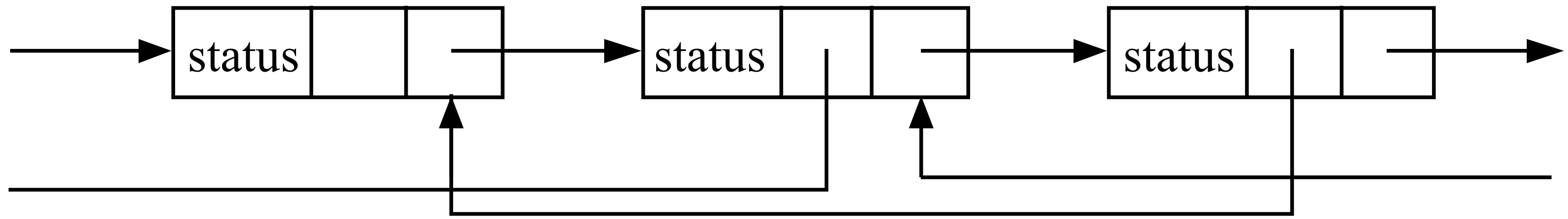
Even worse: Results used internally in subsequent computations with comparisons.

Example: Computation of pressure value P and temperature value T with floating point arithmetic and usage as in program shown:

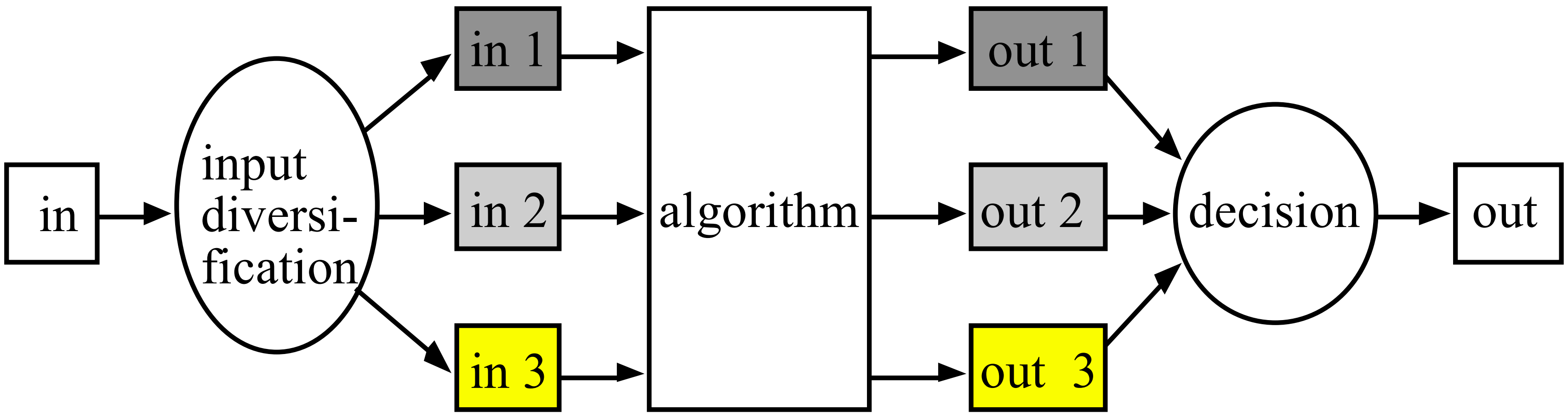


Redundant Data

Redundantly linked list



Data diversity



Examples

Use of formal methods

- Formal specification with Z
  Tektronix: Specification of reusable oscilloscope architecture
- Formal specification with SDL
  ABB Signal: Specification of automatic train protection systems
- Formal software verification with Statecharts
  GEC Alsthom: SACEM - speed control of RER line A trains in Paris

Use of design diversity

- 2x2-version programming
  Aerospatiale: Fly-by wire system of Airbus A310
- 2-version programming
  US Space Shuttle: PASS (IBM) and BFS (Rockwell)
- 2-version programming
  ABB Signal: Error detection in automatic train protection system EBICAB 900

Example: 2-Version Programming (EBICAB 900)

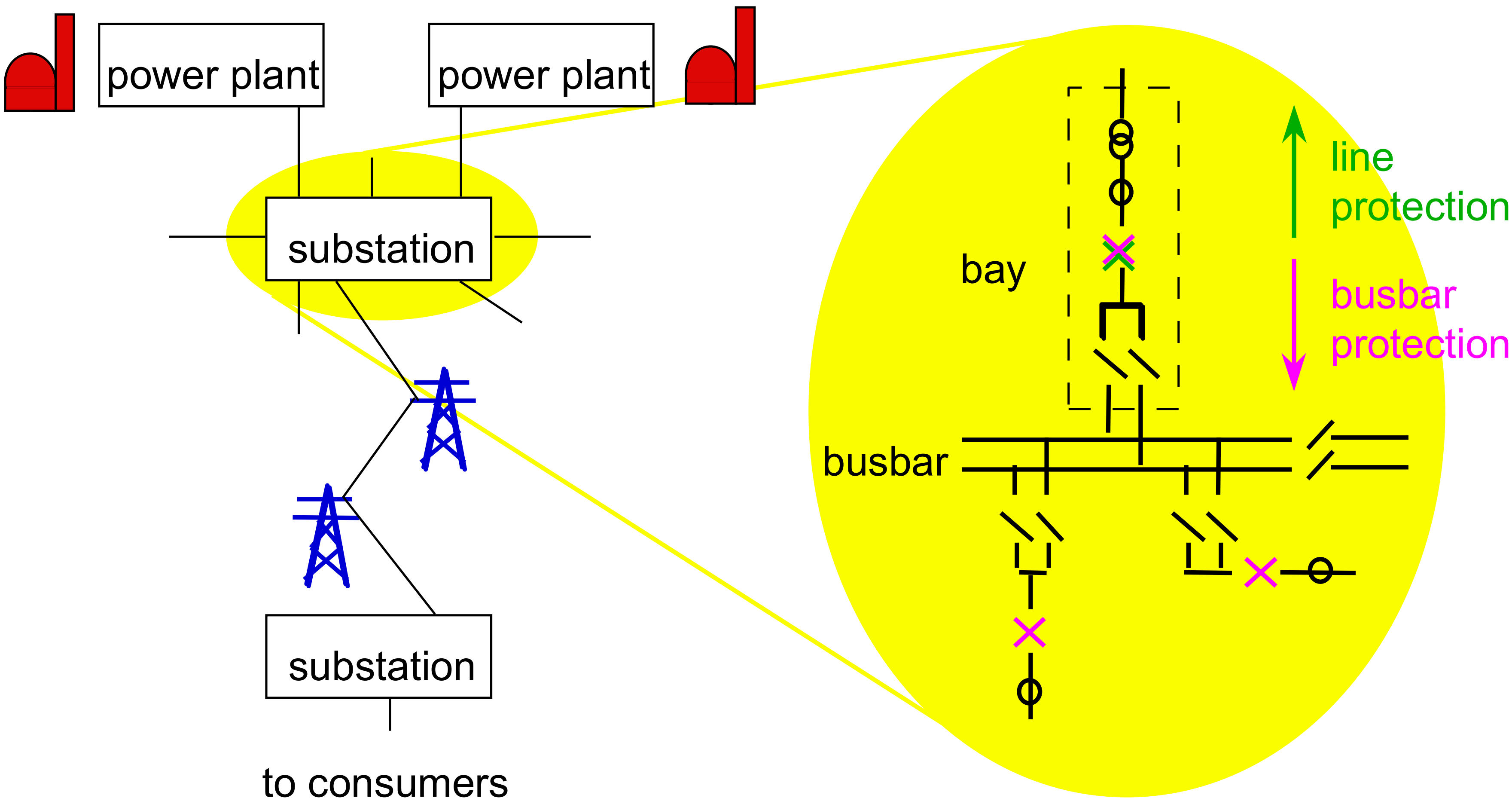
Both for physical faults and design faults (single processor → time redundancy).

data input	algorithm A	algorithm B	A = B?	data output	• • •

→ time

- 2 separate teams for algorithms A and B
  3rd team for A and B specs and synchronisation
- B data is inverted, single bytes mirrored compared with A data
- A data stored in increasing order, B data in decreasing order
- Comparison between A and B data at checkpoints
- Single points of failure (e.g. data input) with special protection (e.g. serial input with CRC)

Example: On-line physical fault detection



Functionality of Busbar Protection (Simplified)

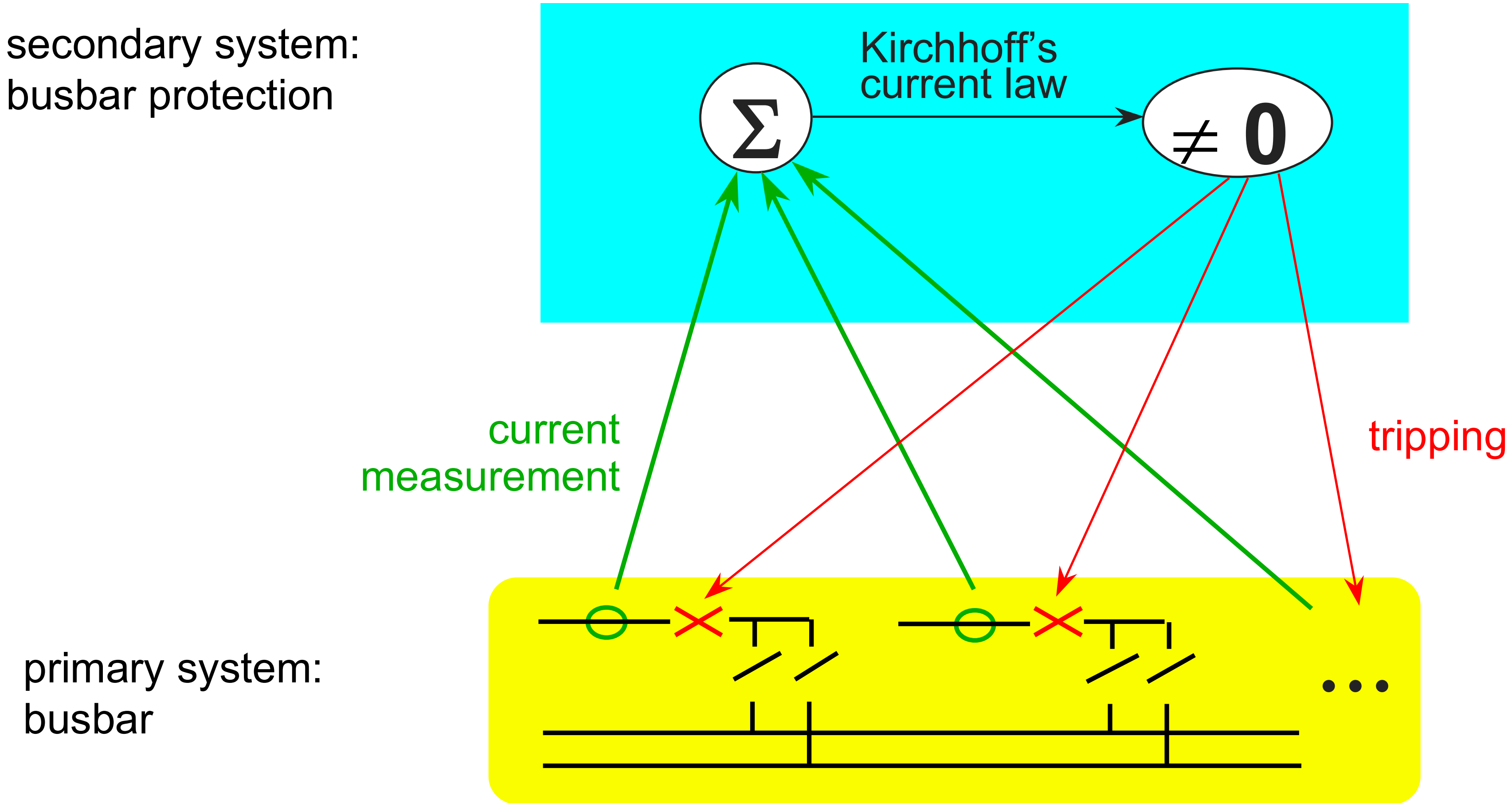
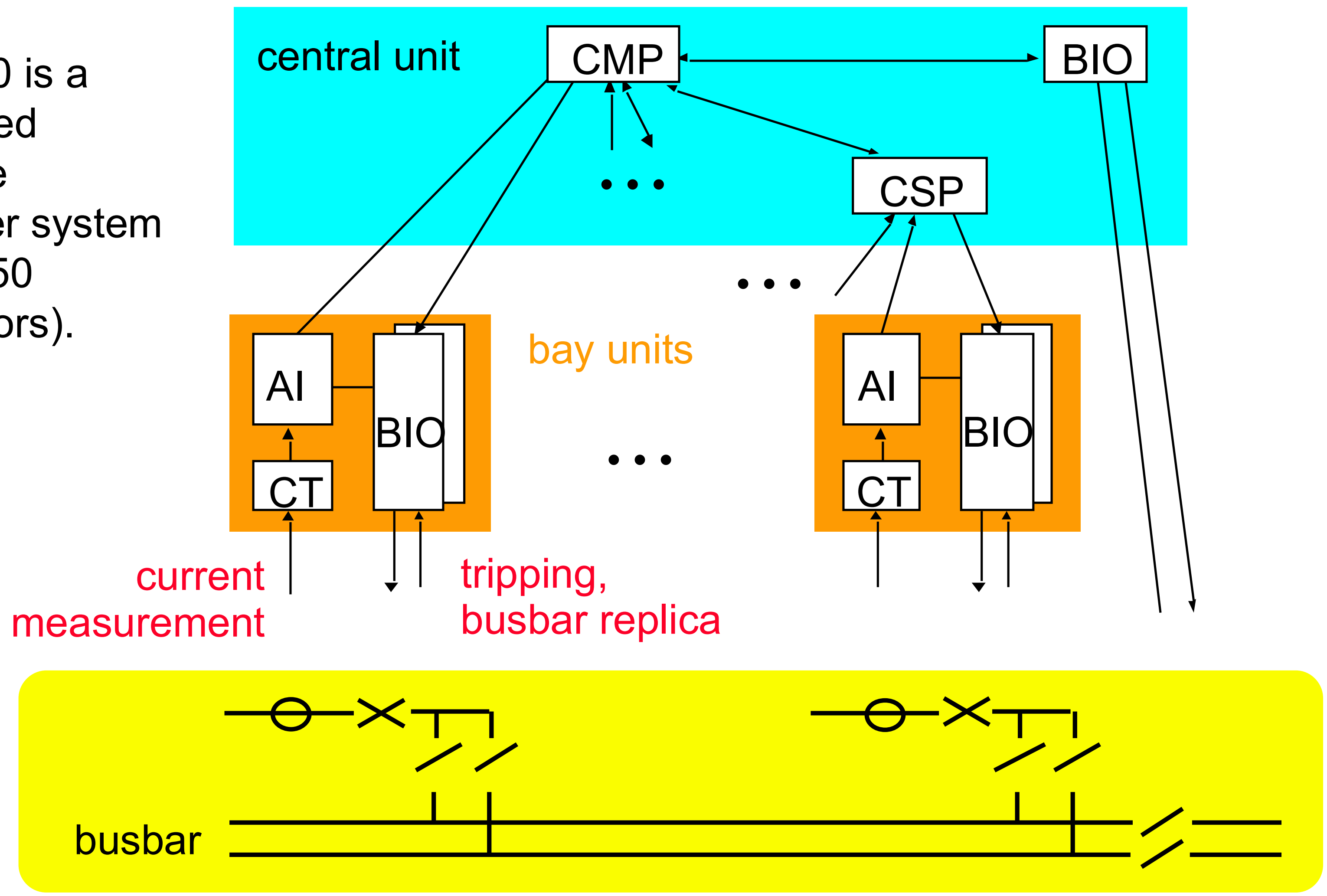


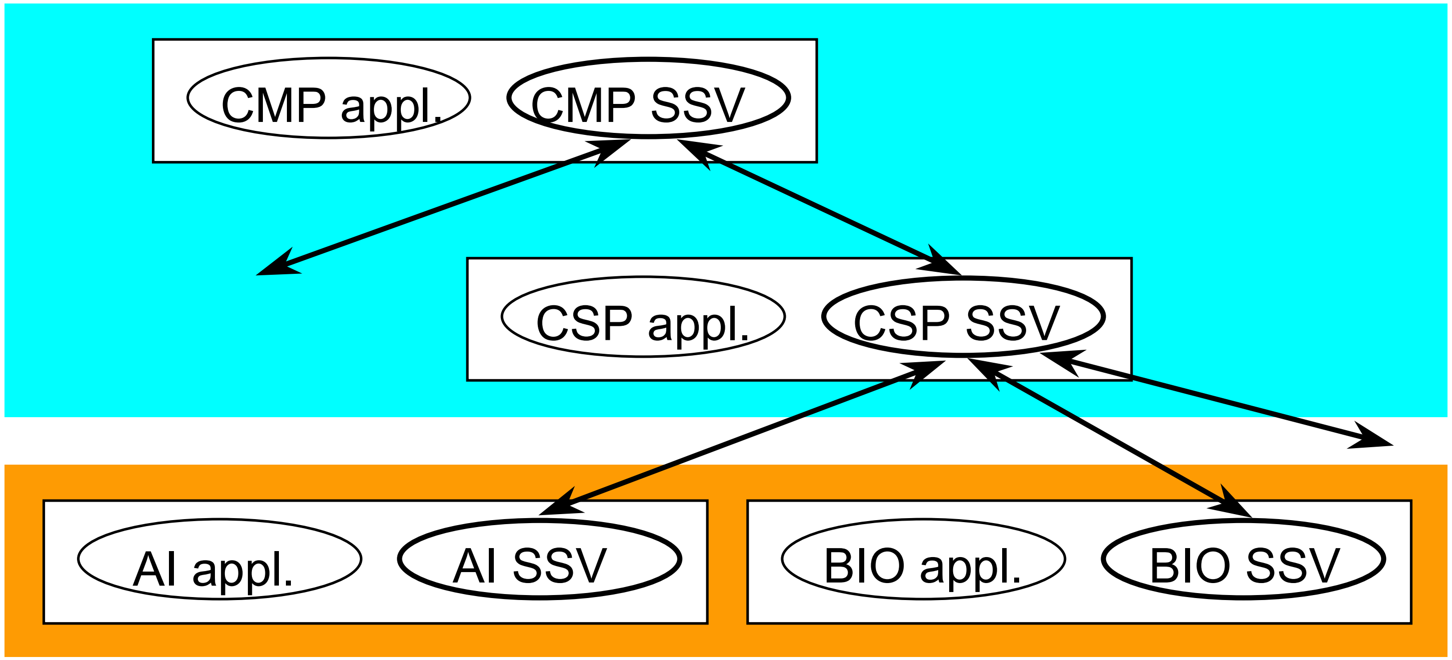
ABB REB 500 Hardware Structure

REB 500 is a distributed real-time computer system (up to 250 processors).



Software Self-Supervision

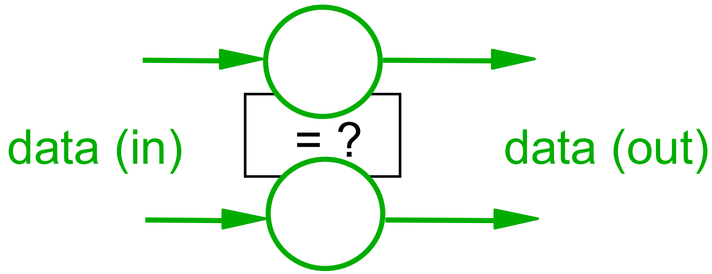
Each processor in the system runs application objects and self-supervision tasks.



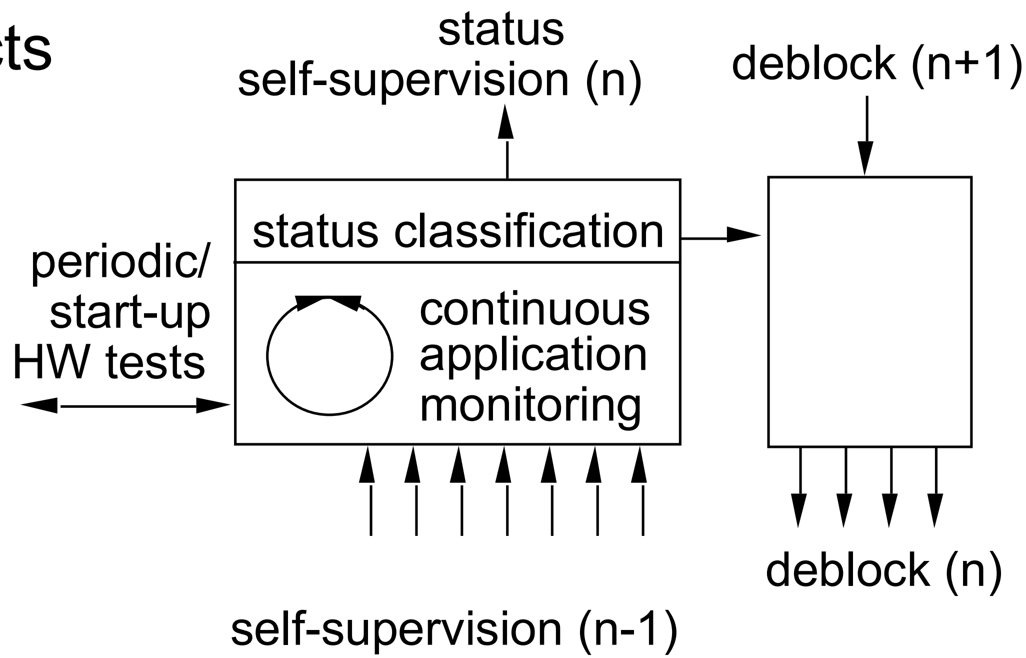
Only communication between self-supervision tasks is shown.

Elements of the Self-Supervision Hierarchy

Application Objects



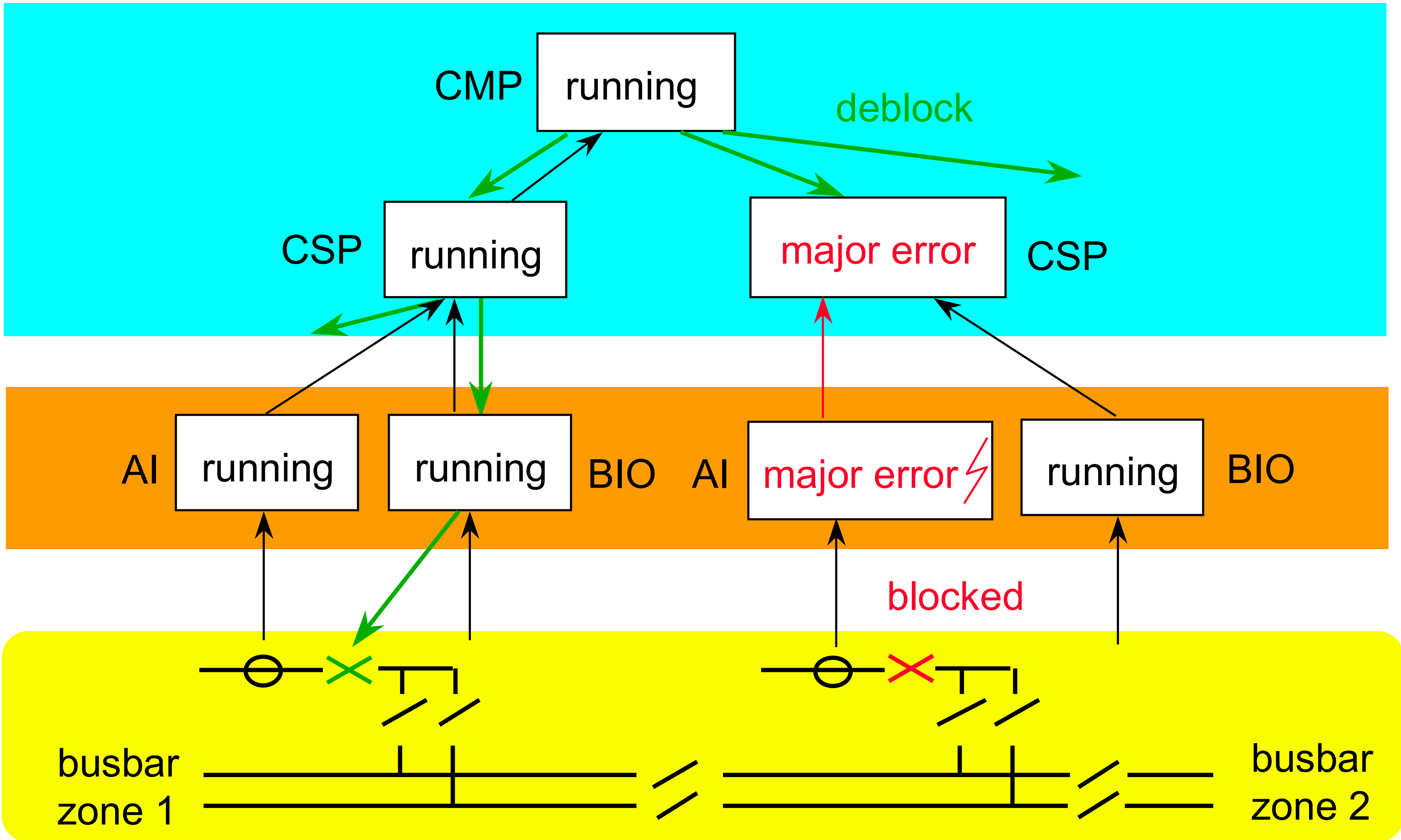
Self-Supervision Objects



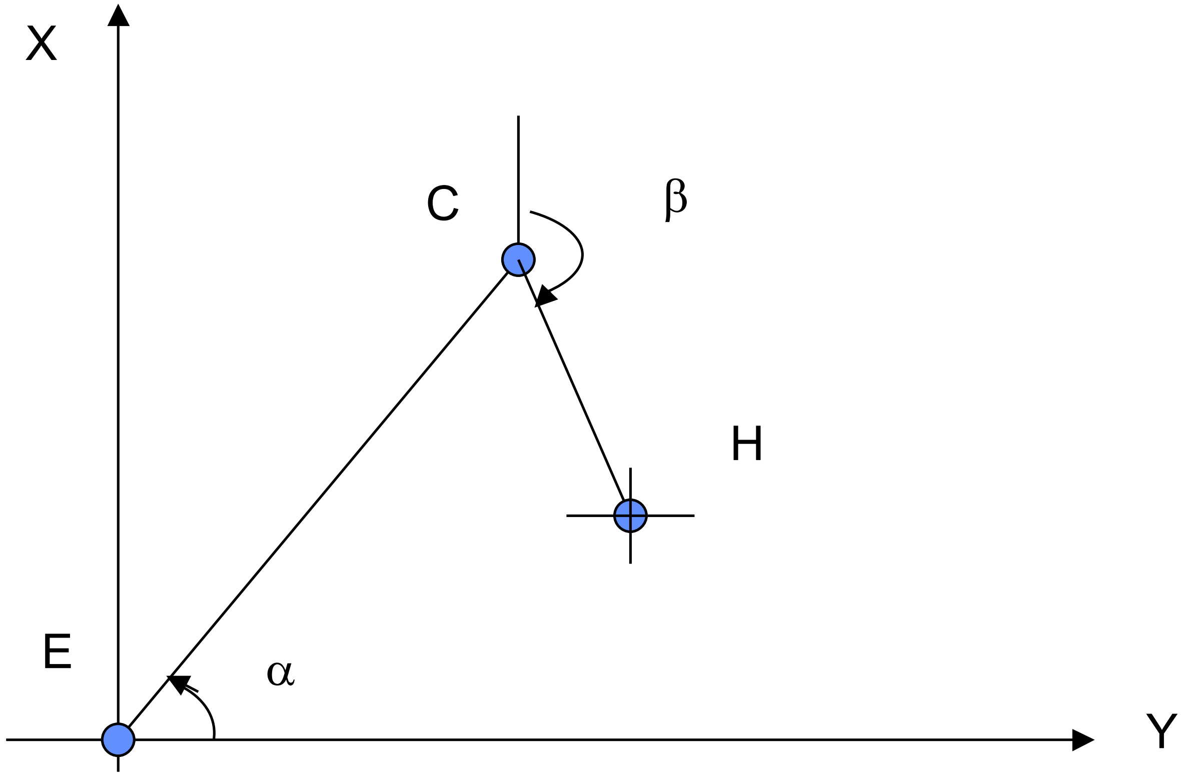
Example Self-Supervision Mechanisms

- Binary Input Encoding: 1-out-of-3 code for normal positions (open, closed, moving)
- Data Transmission: Safety CRC
  Implicit safety ID (source/sink)
  Time-stamp
  Receiver time-out
- Input Consistency: Matching time-stamps and data sources
- Safe Storage: Duplicate data
  Check cyclic production/consumption with toggle bit
- Diverse tripping: Two independent trip decision algorithms (differential with restraint current, comparison of current phases)

Example Handling of Protection System Faults

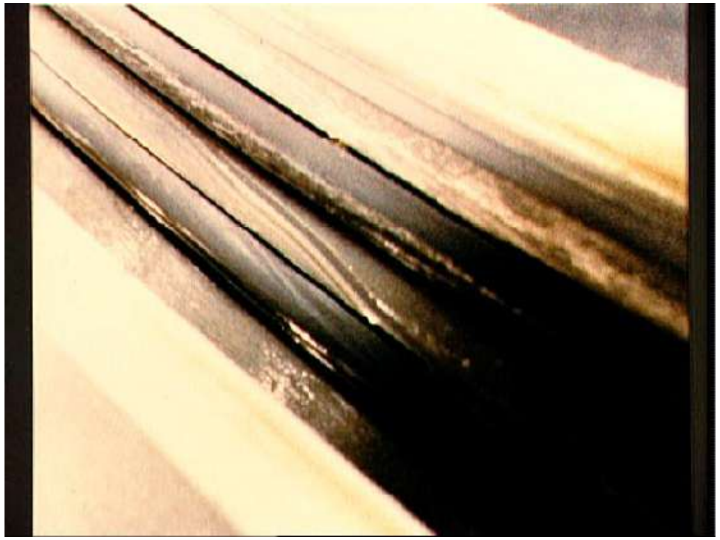


Exercise: Robot arm



write a program to determine the x,y coordinates of the robot head H, given that EC and CH are known.
The (absolute) angles are given by a resolver with 16 bits (0..65535), at joints E and C

Industrial Automation
Automation Industrielle
Industrielle Automation



9.6 Safety analysis and standards

Analyse de sécurité et normes

Sicherheitsanalyse und Normen

Dr. B. Eschermann

ABB Research Center, Baden, Switzerland

Overview Dependability Analysis

9.6.1 Qualitative Evaluation

- Failure Mode and Effects Analysis (FMEA)
- Fault Tree Analysis (FTA)
- Example: Differential pressure transmitter

9.6.2 Quantitative Evaluation

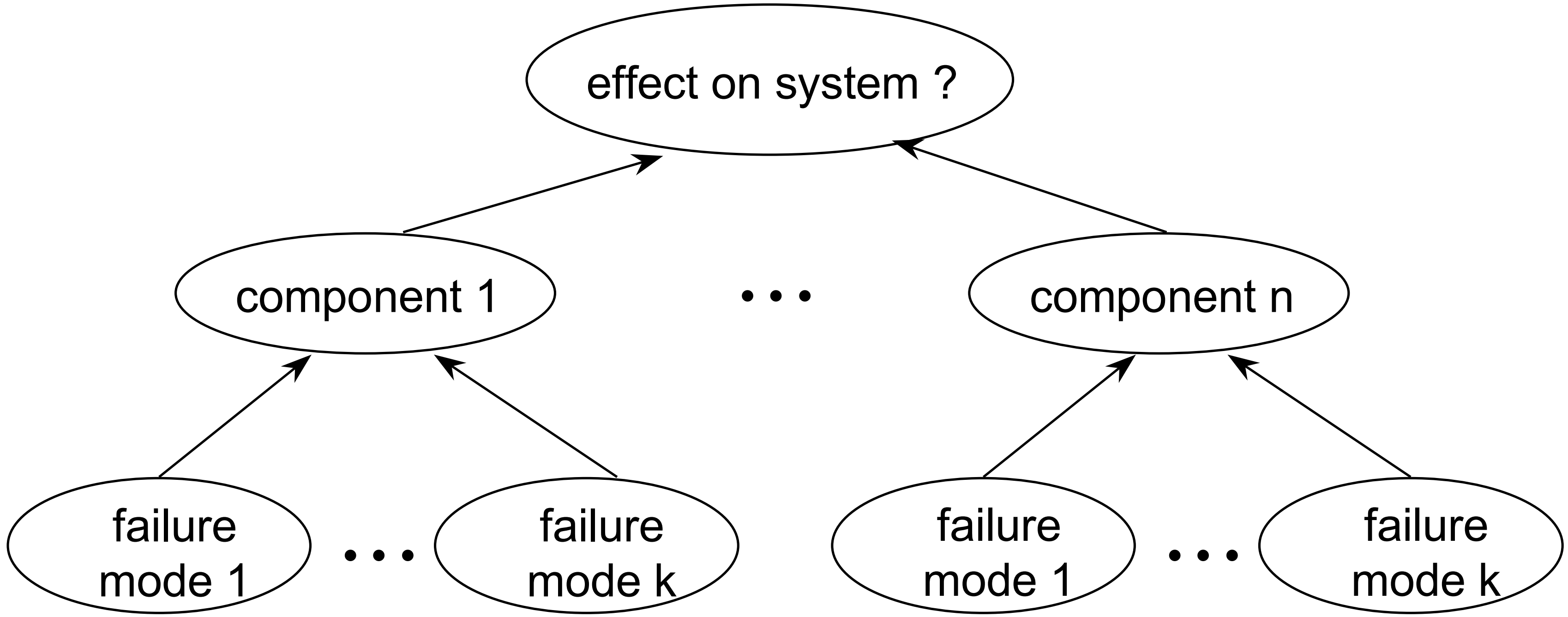
- Combinational Evaluation
- Markov Chains
- Example: Bus-bar Protection

9.6.3 Dependability Standards and Certification

- Standardization Agencies
- Standards

Failure Mode and Effects Analysis (FMEA)

Analysis method to identify component failures which have significant consequences affecting the system operation in the application considered.
$\rightarrow$ identify faults (component failures) that lead to system failures.



FMEA is inductive (bottom-up).

FMEA: Coffee machine example

component	failure mode	effect on system
water tank	empty	no coffee produced
	too full	electronics damaged
coffee bean container	empty	no coffee produced
	too full	coffee mill gets stuck
coffee grounds container	too full	coffee grounds spilled
………		

FMEA: Purpose (overall)

There are different reasons why an FMEA can be performed:

- Evaluation of effects and sequences of events caused by each identified item failure mode
  (→ get to know the system better)
- Determination of the significance or criticality of each failure mode as to the system's correct function or performance and the impact on the availability and/or safety of the related process
  (® identify weak spots)
- Classification of identified failure modes according to their detectability, diagnosability, testability, item replaceability and operating provisions (tests, repair, maintenance, logistics etc.)
  (® take the necessary precautions)
- Estimation of measures of the significance and probability of failure
  (® demonstrate level of availability/safety to user or certification agency)

FMEA: Critical decisions

Depending on the exact purpose of the analysis, several decisions have to be made:

- For what purpose is it performed (find weak spots « demonstrate safety to certification agency, demonstrate safety « compute availability)
- When is the analysis performed (e.g. before « after detailed design)?
- What is the system (highest level considered), where are the boundaries to the external world (that is assumed fault-free)?
- Which components are analyzed (lowest level considered)?
- Which failure modes are considered (electrical, mechanical, hydraulic, design faults, human/operation errors)?
- Are secondary and higher-order effects considered (i.e. one fault causing a second fault which then causes a system failure etc.)?
- By whom is the analysis performed (designer, who knows system best « third party, which is unbiased and brings in an independent view)?

FMEA and FMECA

FMEA only provides qualitative analysis (cause effect chain).

FMECA (failure mode, effects and criticality analysis) also provides (limited) quantitative information.

- each basic failure mode is assigned a failure probability and a failure criticality
- if based on the result of the FMECA the system is to be improved (to make it more dependable) the failure modes with the highest probability leading to failures with the highest criticality are considered first.

Coffee machine example:

- If the coffee machine is damaged, this is more critical than if the coffee machine is OK and no coffee can be produced temporarily
- If the water has to be refilled every 20 cups and the coffee has to be refilled every 2 cups, the failure mode “coffee bean container too full” is more probable than “water tank too full”.

Criticality Grid

Criticality levels

I				
II				
III				
IV				
	very low	low	medium	high

Probability of failure

Failure Criticalities

IV: Any event which could potentially cause the loss of primary system function(s) resulting in significant damage to the system or its environment and causes the loss of life

III: Any event which could potentially cause the loss of primary system function(s) resulting in significant damage to the system or its environment and negligible hazards to life

II: Any event which degrades system performance function(s) without appreciable damage to either system, environment or lives

I: Any event which could cause degradation of system performance function(s) resulting in negligible damage to either system or environment and no damage to life

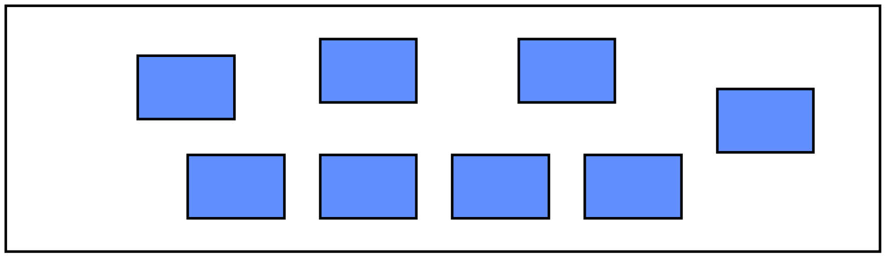
FMEA/FMECA: Result

Depending on the result of the FMEA/FMECA, it may be necessary to:

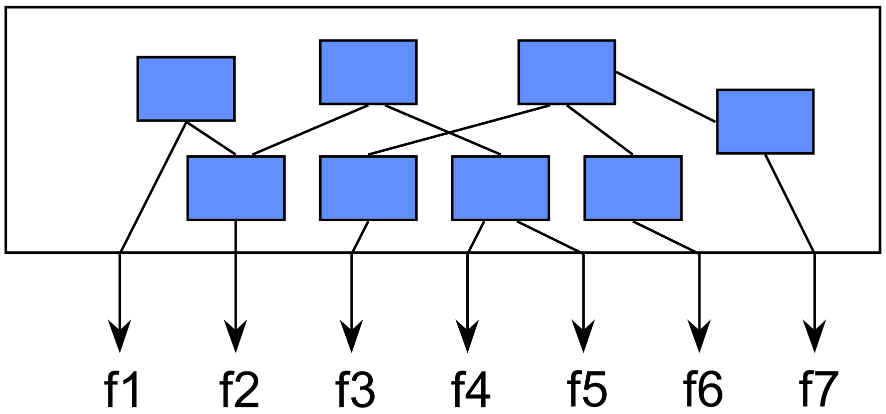
- change design, introduce redundancy, reconfiguration, recovery etc.
- introduce tests, diagnoses, preventive maintenance
- focus quality assurance, inspections etc. on key areas
- select alternative materials, components
- change operating conditions (e.g. duty cycles to anticipate/avoid wear-out failures)
- adapt operating procedures (allowed temperature range etc.)
- perform design reviews
- monitor problem areas during testing, check-out and use
- exclude liability for identified problem areas

FMEA: Steps (1)

1) Break down the system into components.



2) Identify the functional structure of the system and how the components contribute to functions.



FMEA: Steps (2)

3) Define failure modes of each component
- new components: refer to similar already used components
- commonly used components: base on experience and measurements
- complex components: break down in subcomponents and derive failure mode of component by FMEA on known subcomponents
- other: use common sense, deduce possible failures from functions and physical parameters typical of the component operation

4) Perform analysis for each failure mode of each component and record results in table:

component name/ID	function	failure mode	failure cause	failure effect		failure detection	other provision	remark
				local	global			

Example (Generic) Failure Modes

- fails to remain (in position)
- fails to open
- fails to close
- fails if open
- fails if closed
- restricted flow
- fails out of tolerance (high)
- fails out of tolerance (low)
- inadvertent operation
- intermittent operation
- premature operation
- delayed operation
- false actuation
- fails to stop
- fails to start
- fails to switch
- erroneous input (increased)
- erroneous input (decreased)
- erroneous output (increased)
- erroneous output (decreased)
- loss of input
- loss of output
- erroneous indication
- leakage

Other FMEA Table Entries

Failure cause: Why is it that the component fails in this specific way?
To identify failure causes is important to
- estimate probability of occurrence
- uncover secondary effects
- devise corrective actions

Local failure effect: Effect on the system element under consideration (e.g. on the output of the analyzed component). In certain instances there may not be a local effect beyond the failure mode itself.

Global failure effect: Effect on the highest considered system level. The end effect might be the result of multiple failures occurring as a consequence of each other.

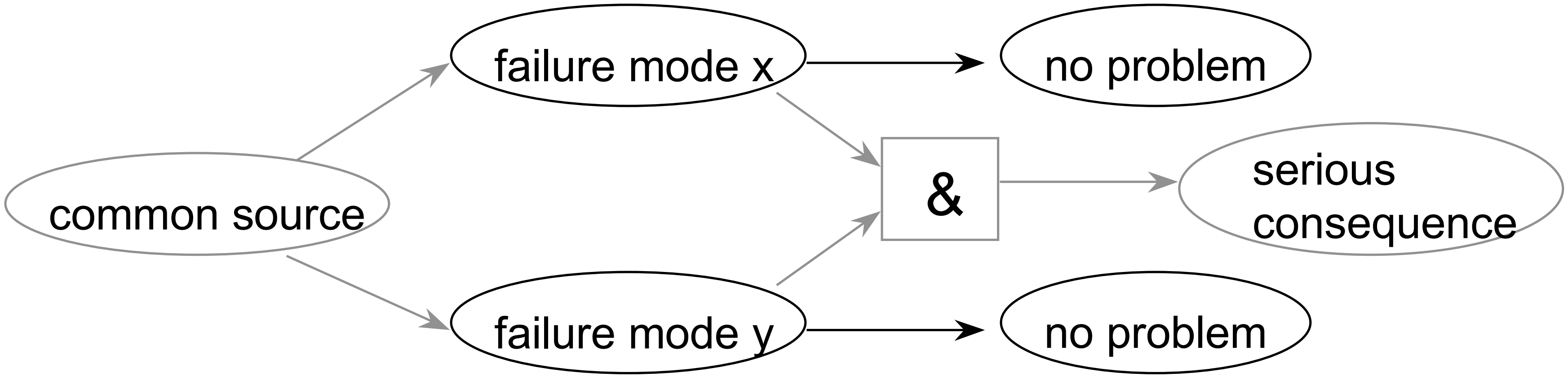
Failure detection: Methods to detect the component failure that should be used.

Other provisions: Design features might be introduced that prevent or reduce the effect of the failure mode (e.g. redundancy, alarm devices, operating restrictions).

Common Mode Failures (CMF)

In FMEA all failures are analyzed independent of each other.

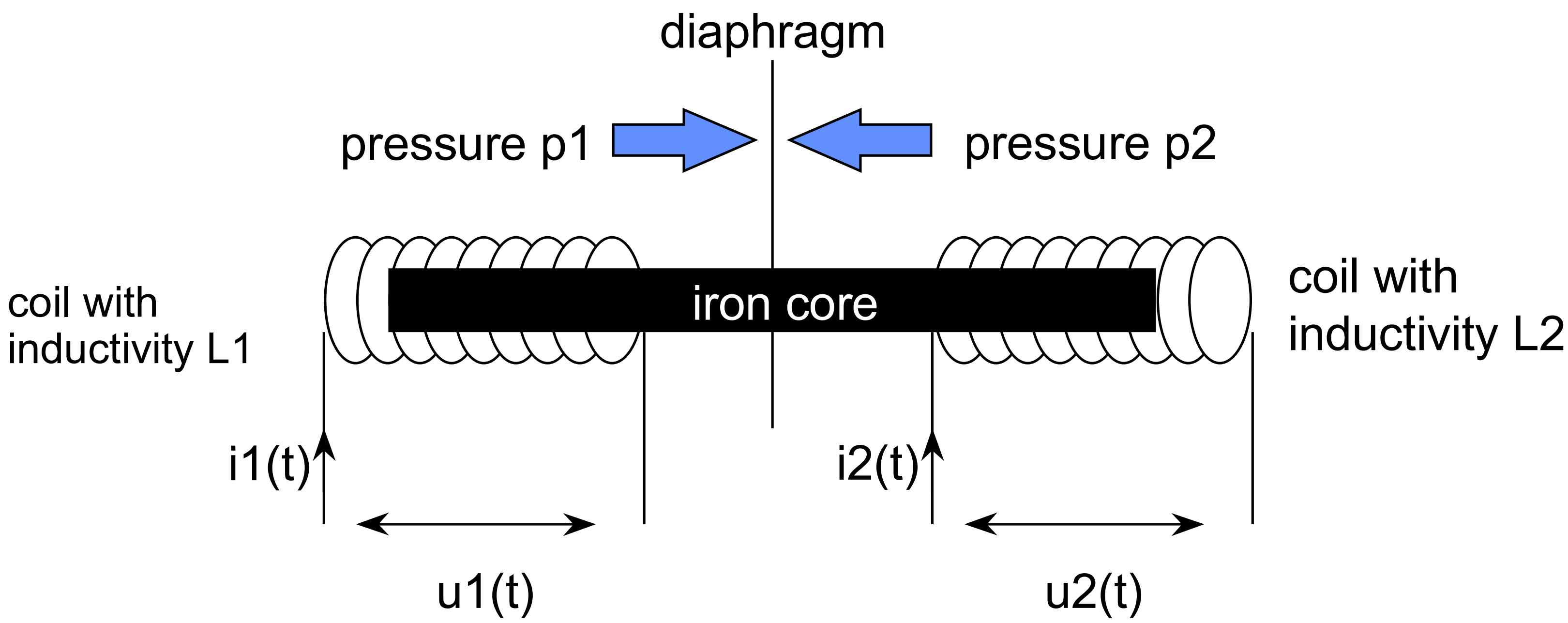
Common mode failures are related failures that can occur due to a single source such as design error, wrong operation conditions, human error etc.



Example: Failure of power supply common to redundant units causes both redundant units to fail at the same time.

Example: Differential Pressure Transmitter (1)

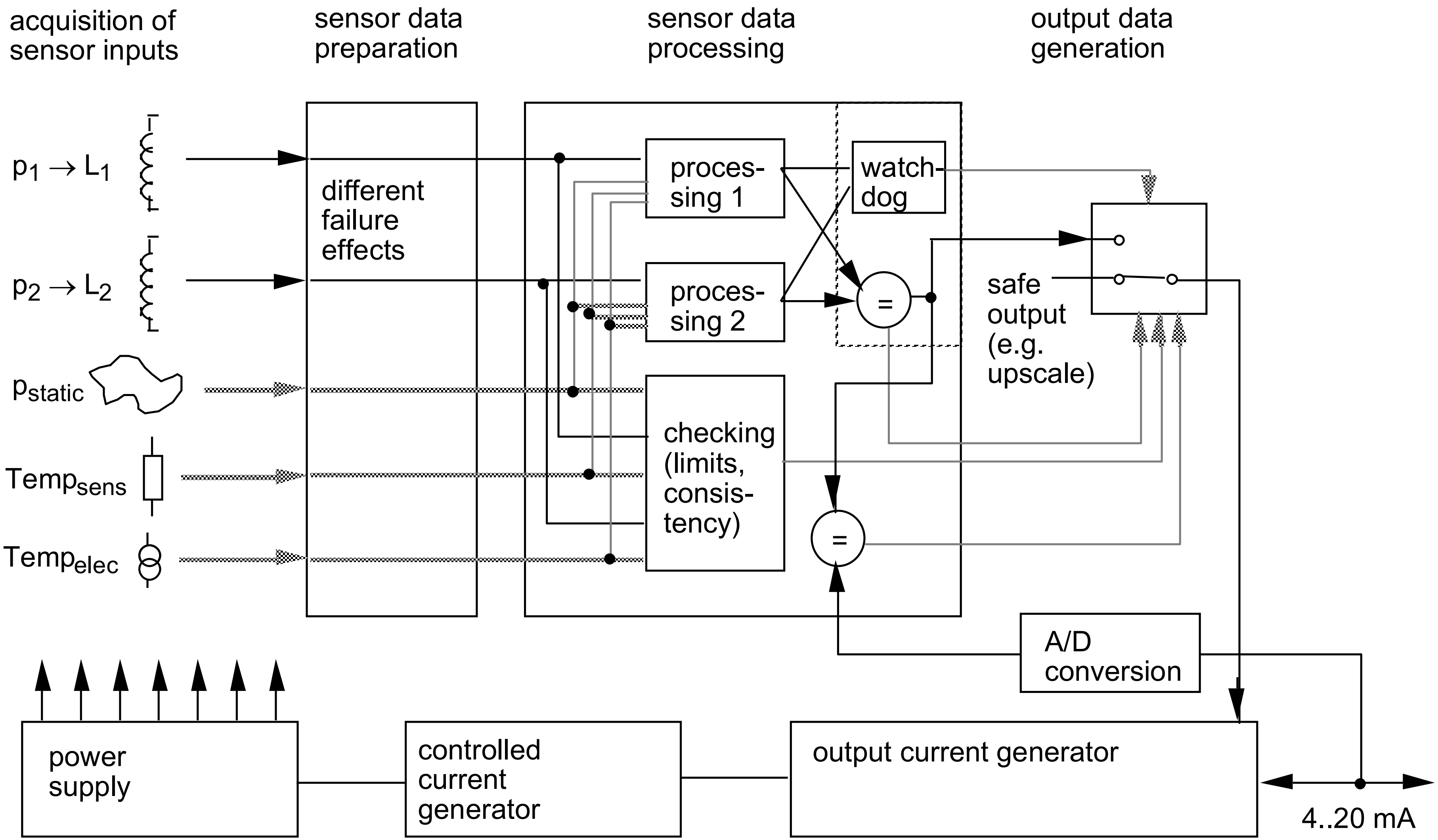
Functionality: Measure difference in pressures p1 – p2.



p1 – p2 = f1 (inductivity L1, temperature T, static pressure p)

p1 – p2 = f2 (inductivity L2, temperature T, static pressure p)

Example: Differential Pressure Transmitter (2)



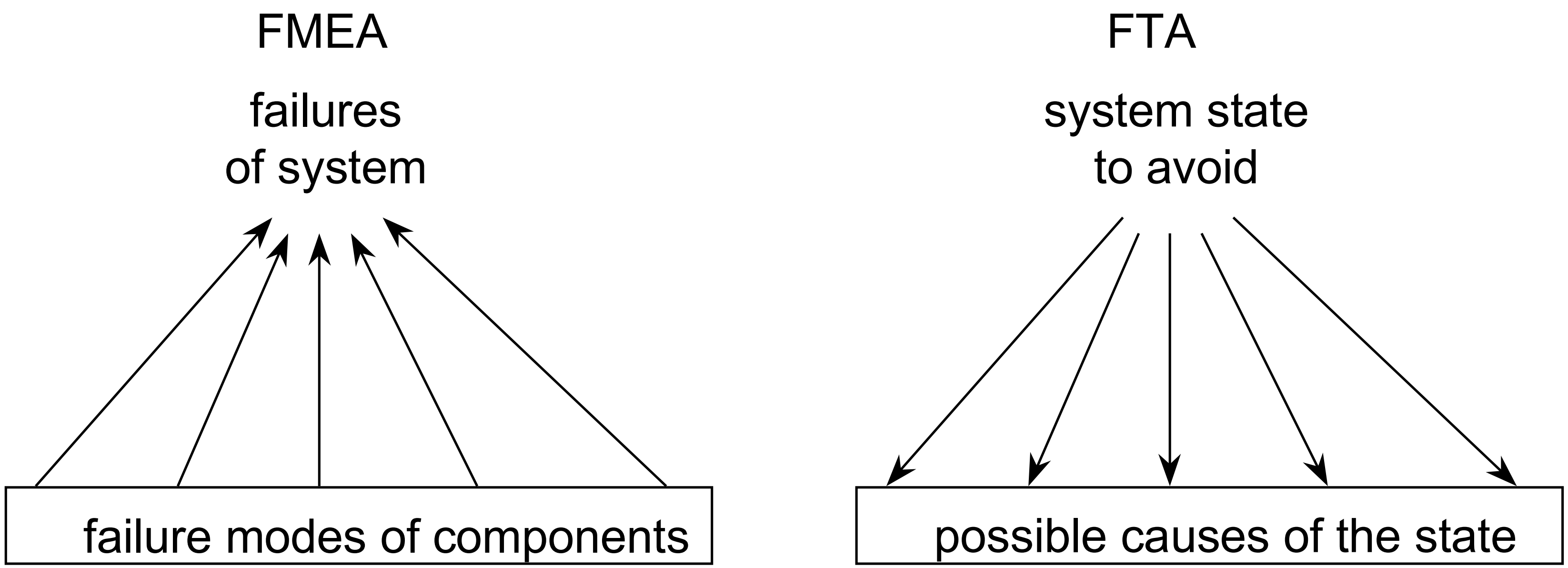
FMEA for Pressure Transmitter

ID-Nr	Function	Failure Mode	Local Effect	Detection Mechanism	Failure Handling	Global Effect	Comments
1.1.1	p1 measure-ment	out of fail-safe accuracy range	pressure input via L1 wrong	limit check and consistency check (comparison with p2) in software of sensor data processing	go to safe state	output driven to up/downscale	diaphragm failure (both p1 and p2 wrong) detected by comparison with pstatic, requires that separate sensor is used for pstatic
1.1.2		wrong but within fail-safe accuracy range	pressure input via L1 slightly wrong	consistency check (comp. with p2), detection of small failures not guaranteed (allowed difference p1-p2)	not applicable (n/a)	output value slightly wrong, but within fail-safe accuracy range	
1.2.1	p2 measure-ment	out of fail-safe accuracy range	pressure input via L2 wrong	limit check and consistency check (comparison with p1) in software of sensor data processing	go to safe state	output driven to up/downscale	
1.2.2		wrong but within fail-safe accuracy range	pressure input via L2 slightly wrong	consistency check (comp. with p1), detection of small failures not guaranteed (allowed difference p1-p2)	n/a	output value slightly wrong, but within fail-safe accuracy range	

continue on your own ...

Fault Tree Analysis (FTA)

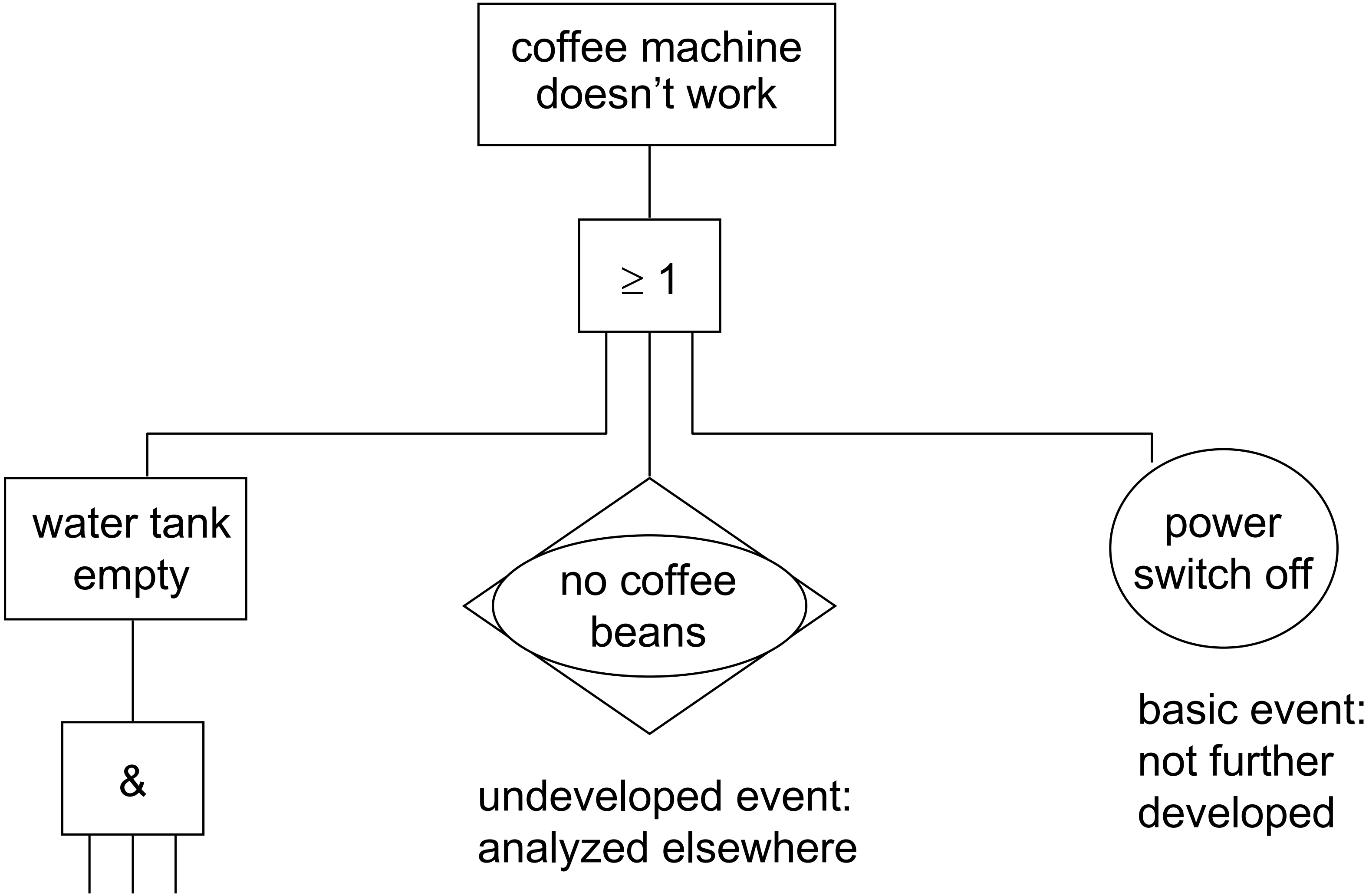
In contrast to FMEA (which is inductive, bottom-up), FTA is deductive (top-down).



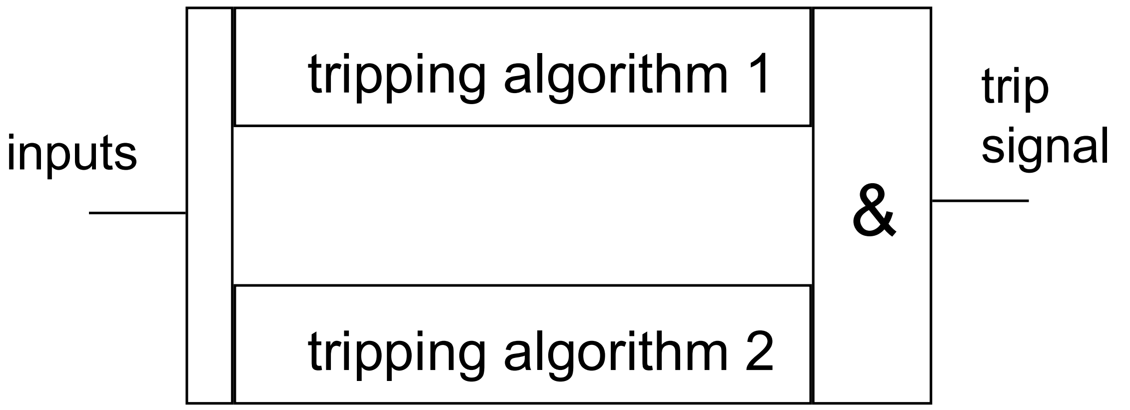
The main problem with both FMEA and FTA is to not forget anything important.

Doing both FMEA and FTA may help to become more complete (2 different views).

Example Fault Tree Analysis



Example: Protection System

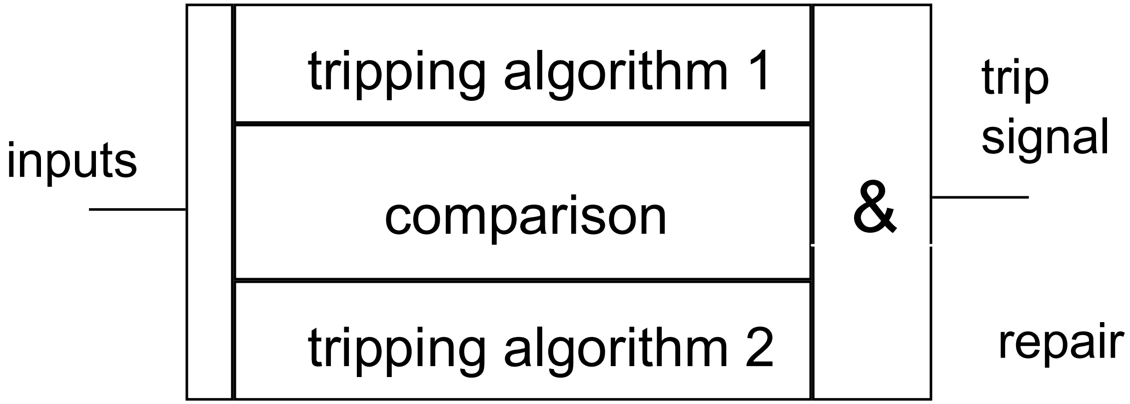


overfunctions reduced

$$P_{otot} = P_o^2$$

underfunctions increased

$$P_{utot} = 2P_u - P_u^2$$

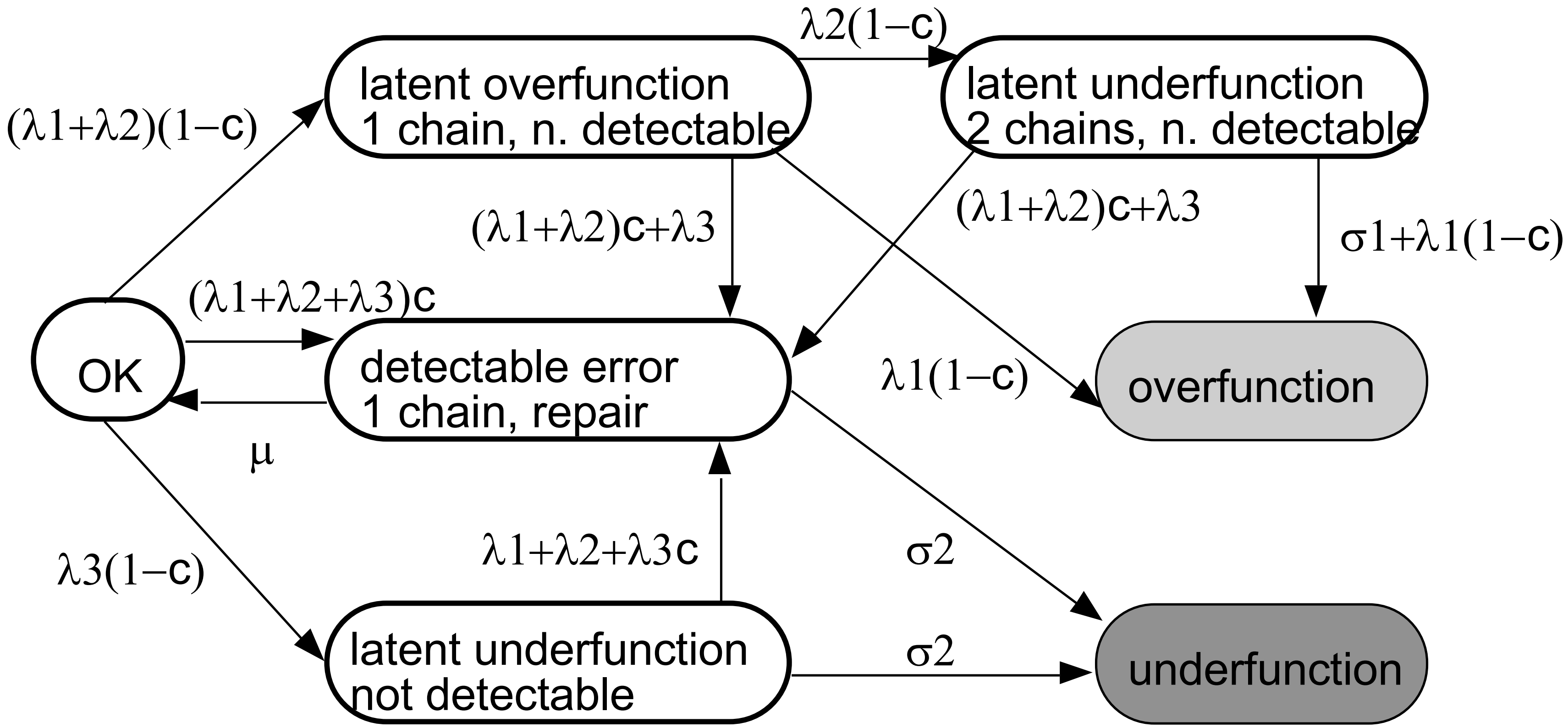


dynamic
modeling
necessary

FTA: IEC Standard

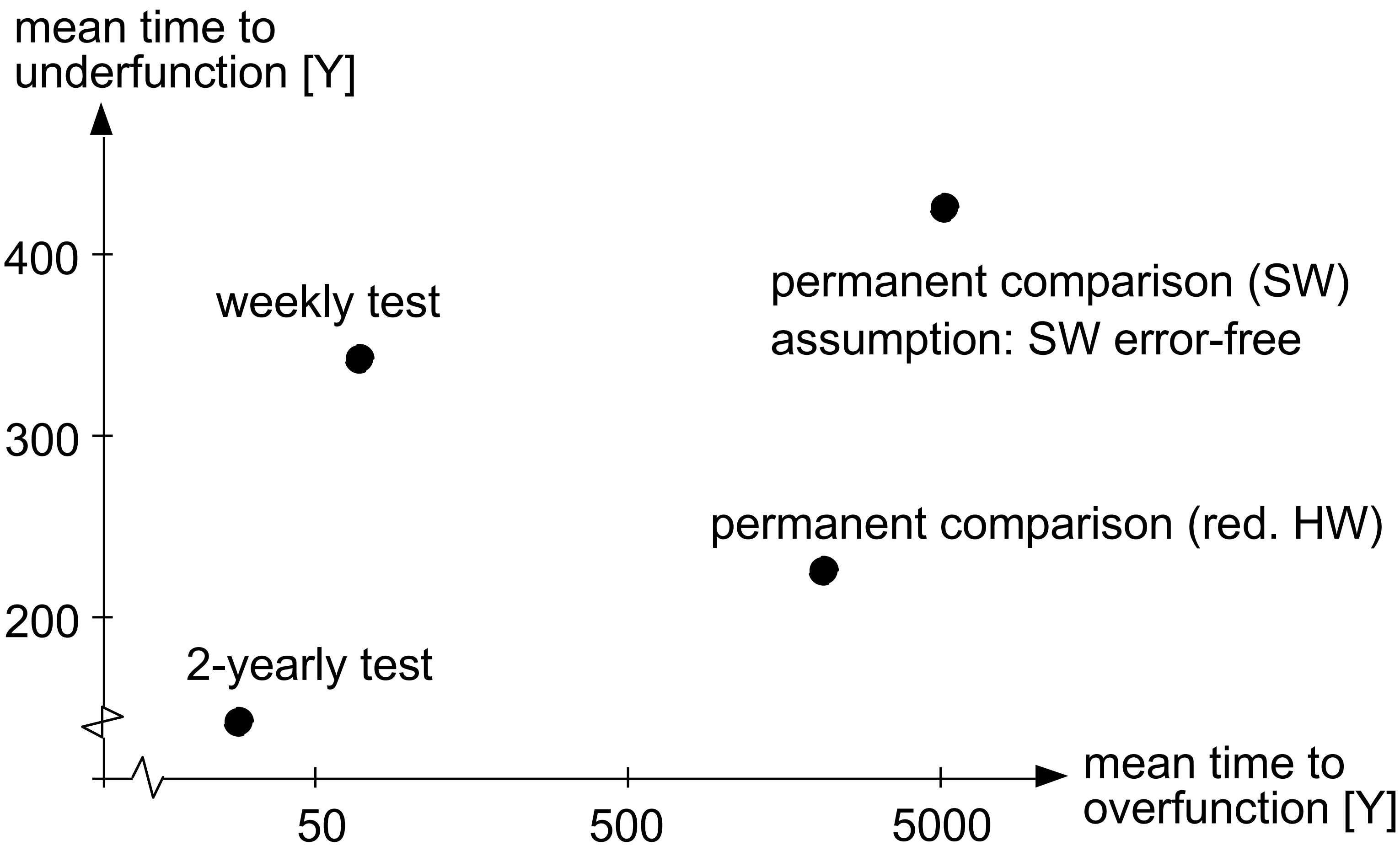
defines basic principles of FTA
provides required steps for analysis
identifies appropriate assumptions, events and failure modes
provides identification rules and symbols

Markov Model



$\lambda 1=0.01$, $\lambda 2=\lambda 3=0.025$, $\sigma 1=5$, $\sigma 2=1$, $\mu=365$, $c=0.9$ [1/Y]

Analysis Results



Example: IEC 61508

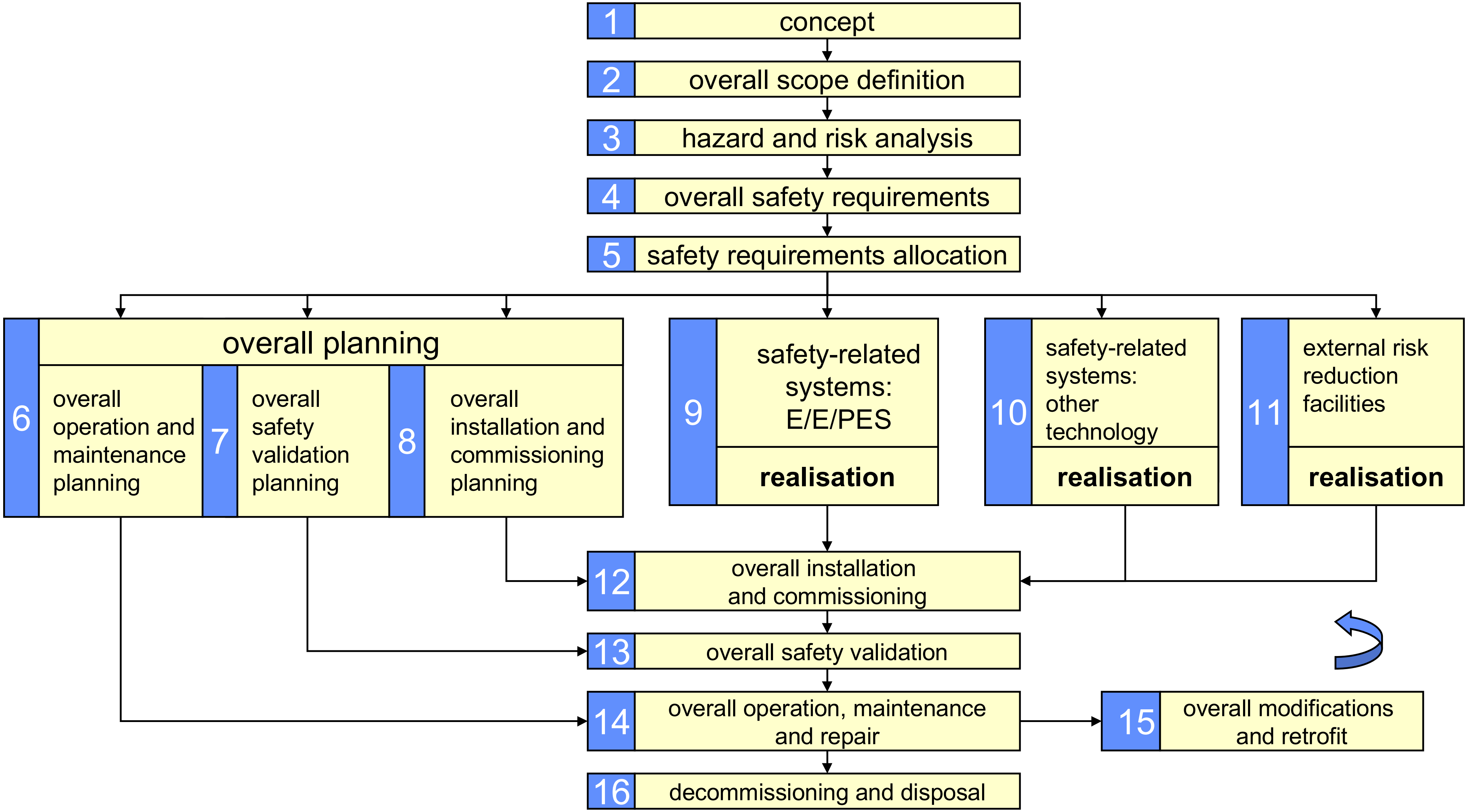
Generic standard for safety-related systems.

Specifies 4 safety integrity levels, or SILs (with specified max. failure rates):

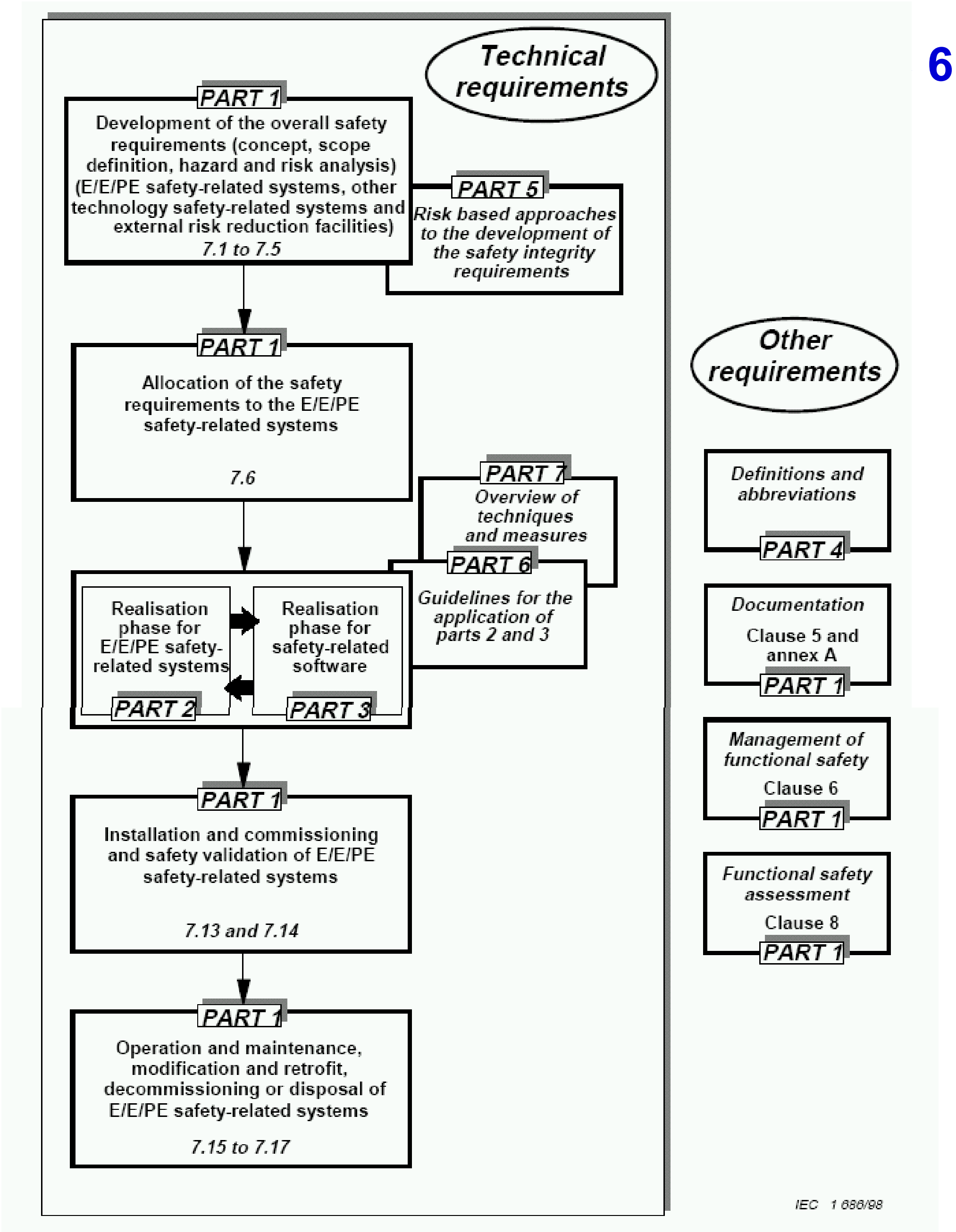
safety integrity level	control systems [per hour]	protection systems [per operation]
4	$\geq 10^{-9}$ to $< 10^{-8}$	$\geq 10^{-5}$ to $< 10^{-4}$
3	$\geq 10^{-8}$ to $< 10^{-7}$	$\geq 10^{-4}$ to $< 10^{-3}$
2	$\geq 10^{-7}$ to $< 10^{-6}$	$\geq 10^{-3}$ to $< 10^{-2}$
1	$\geq 10^{-6}$ to $< 10^{-5}$	$\geq 10^{-2}$ to $< 10^{-1}$

For each of the safety integrity levels it specifies requirements
(see copy out of standard).

Cradle-to-grave reliability (IEC 61508)



61580



Technical requirements
PART 1
Development of the overall safety requirements (concept, scope definition, hazard and risk analysis) (E/E/PE safety-related systems, other technology safety-related systems and external risk reduction facilities)
7.1 to 7.5
PART 5
Risk based approaches to the development of the safety integrity requirements
PART 1
Allocation of the safety requirements to the E/E/PE safety-related systems
7.6
PART 7
Overview of techniques and measures
PART 6
Guidelines for the application of parts 2 and 3
Realisation phase for E/E/PE safety-related systems
PART 2
Realisation phase for safety-related software
PART 3
PART 1
Installation and commissioning and safety validation of E/E/PE safety-related systems
7.13 and 7.14
PART 1
Operation and maintenance, modification and retrofit, decommissioning or disposal of E/E/PE safety-related systems
7.15 to 7.17
Other requirements
Definitions and abbreviations
PART 4
Documentation
Clause 5 and annex A
PART 1
Management of functional safety
Clause 6
PART 1
Functional safety assessment
Clause 8
PART 1
IEC 1 686/98

Software safety integrity and the development lifecycle (V-model)