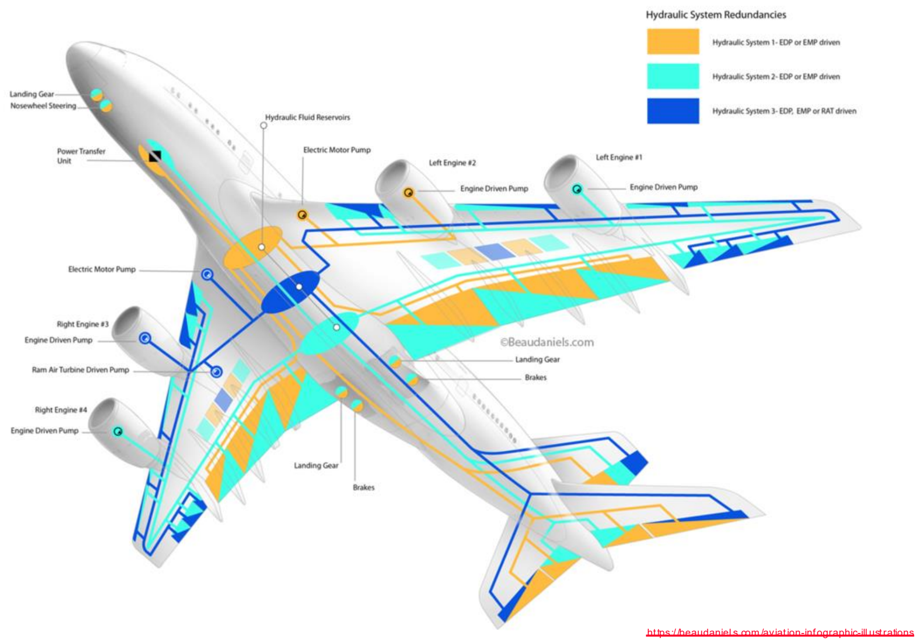
Industrial Automation
Automation Industrielle
Industrielle Automation



https://beaudaniels.com/aviation-infographic-illustrations

Dependable Architectures

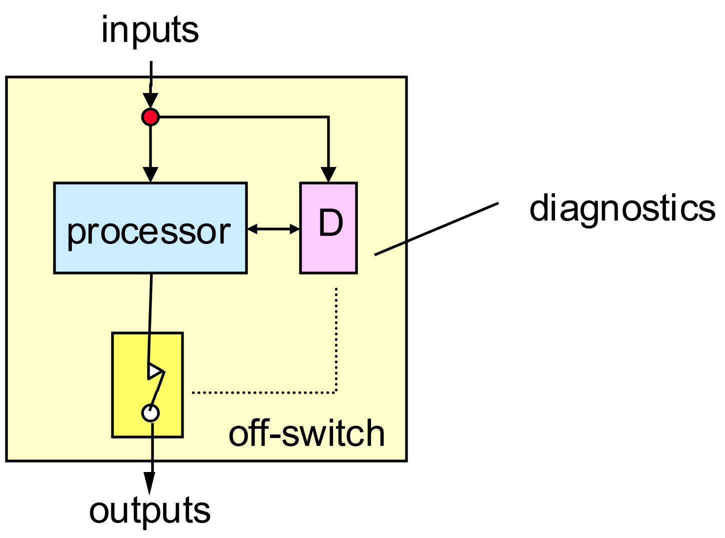
Architectures sûres de fonctionnement

Verlässliche Architekturen

Dr. Jean-Charles Tournier
CERN, Geneva, Switzerland

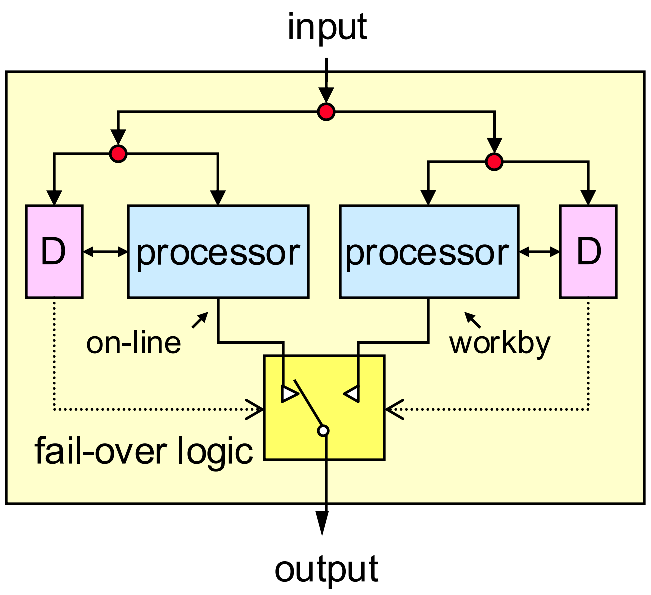
The material of this course has been initially created by Prof. Dr. H. Kirrmann and adapted by Dr. J-C. Tournier

The three main dependable computer architectures



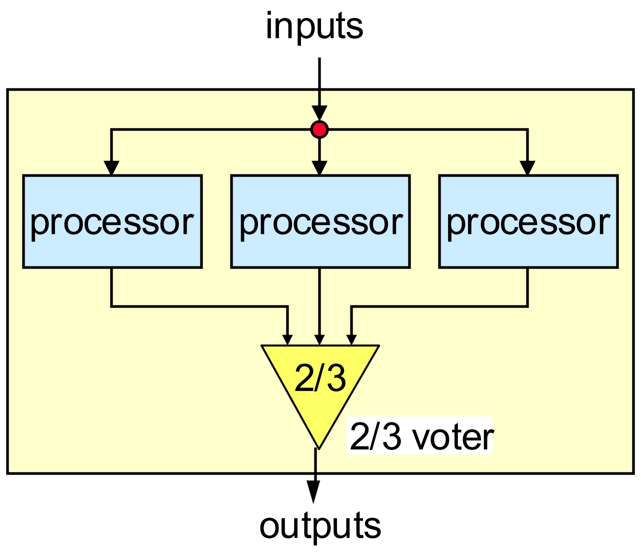
a) Integer

" rather nothing than wrong "
(fail-silent, fail-stop, "fail-safe")
1oo1d



b) Persistent

" rather wrong than nothing "
"fail-operate“
(1oo2D)



c) Integer & persistent

error masking, massive redundancy (2oo3)

Overview Dependable Architectures

1 Error Detection and Fail-Silent

Error Detection: Classification

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

-> disable outputs if error detected

Error detection is the base of fault-tolerant computing (“fail-operate”)

-> switchover if error detected, passivate faulty unit.

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

Error Detection: Classification

Errors can be detected, (in order of increasing latency):

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

→ by continuous monitoring/supervision

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

→ by periodic testing

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

→ by thorough testing, uncovering lurking errors

Error detection

The correctness of a result can be checked by:

relative tests (comparison tests):
by comparing several results of redundant units or computations (not necessary identical)

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

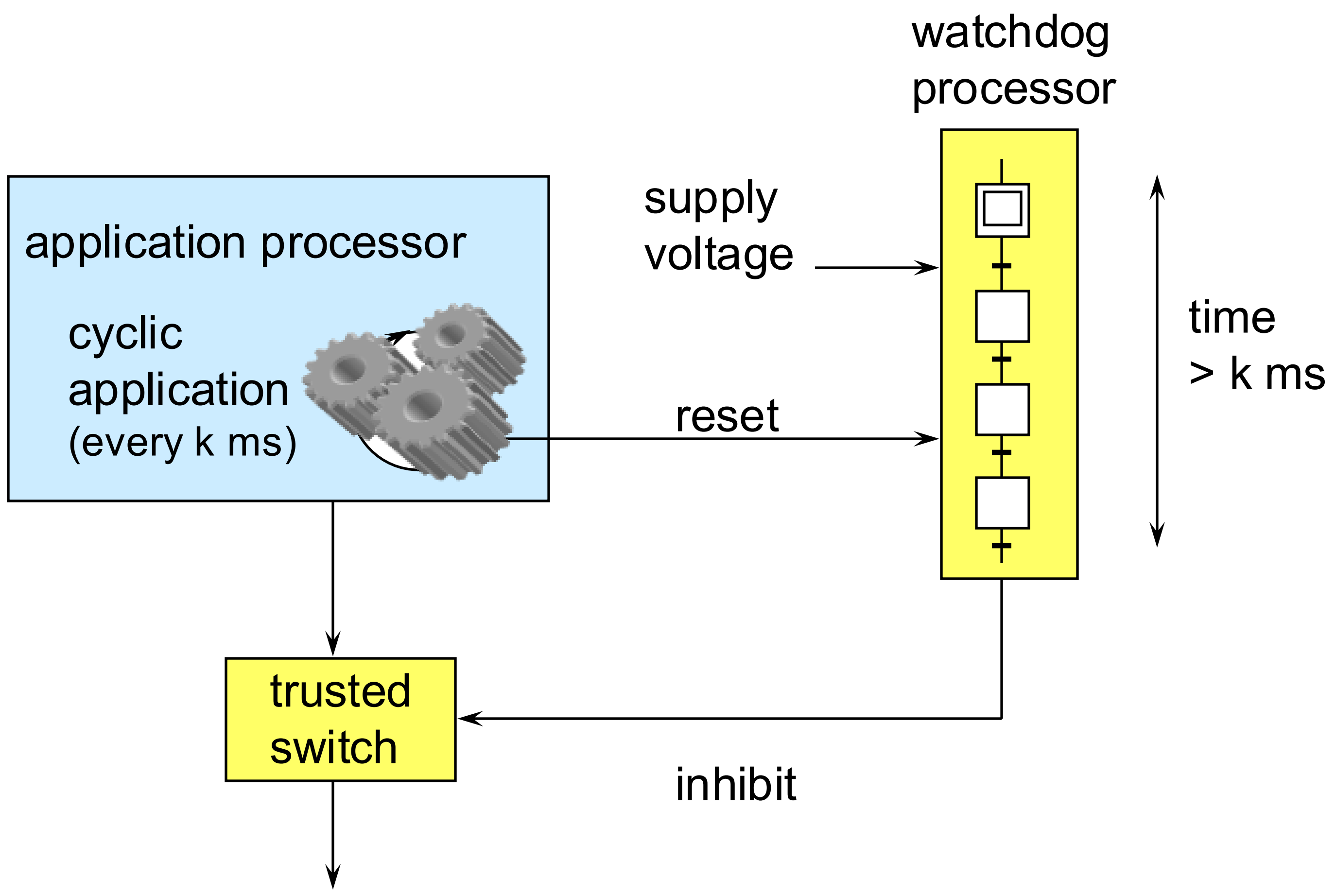
	relative test	absolute test
on-line	duplication and comparison (either hardware duplication or time redundancy) triplication and voting	watchdog (time-out) control flow checking error-detecting code (CRC, etc.) illegal address checking
off-line	comparison with precomputed test result (fixed inputs) e.g.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

Detection of Errors Caused by Physical Faults

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

Component	*Error characteristics*	Typical error detection
Data transmission lines	*medium to high error rate, memoryless*	parity, CRC, watchdog
Regular memory elements	*medium error rate, large storage*	parity, Hamming codes EDC CRC on disk.
Processors and controllers	*low error rate, high complexity*	duplication and comparison, coded logic, control flow, watchdog
Auxiliary elements (hard disk, ventilation)	*high error rate, high diversity*	mechanical integrity, voltage supervision, watchdogs,...

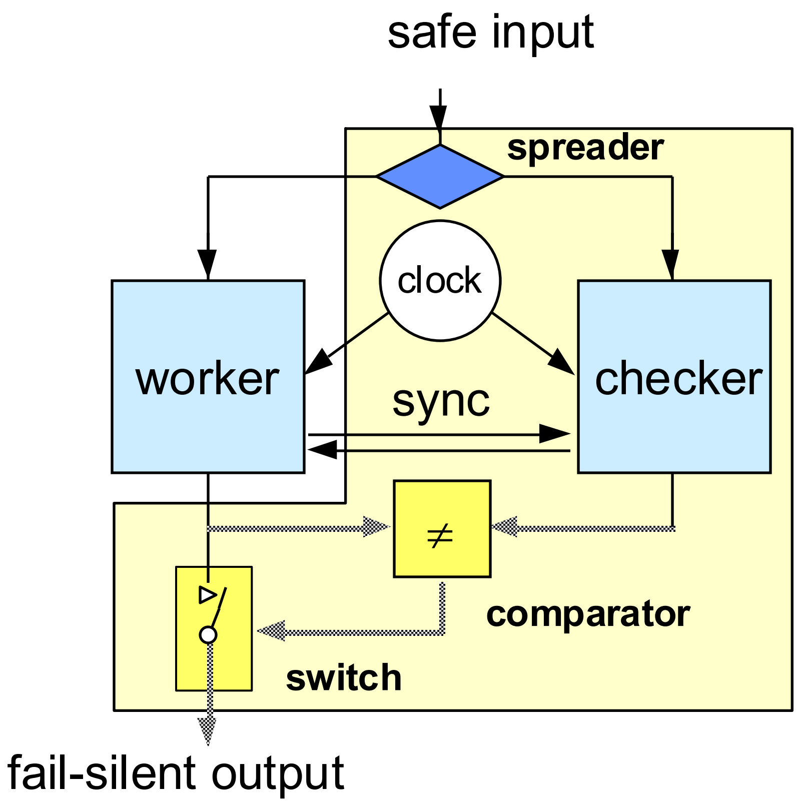
Watchdog Processor (absolute test)



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

Typically implemented in PLCs but also autonomous robots.

Duplication and Comparison (relative test)



Advantage: high coverage, short latency

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

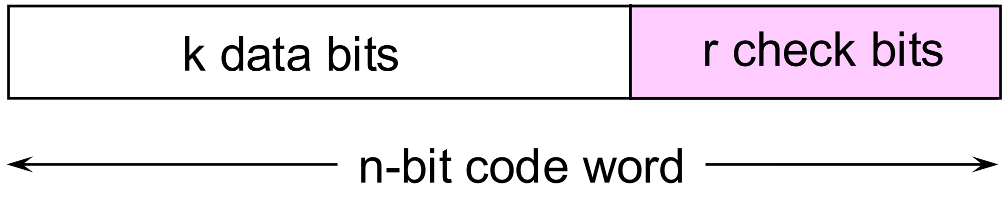
The safety-relevant parts (comparator and switch) are useless if not regularly checked.

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)

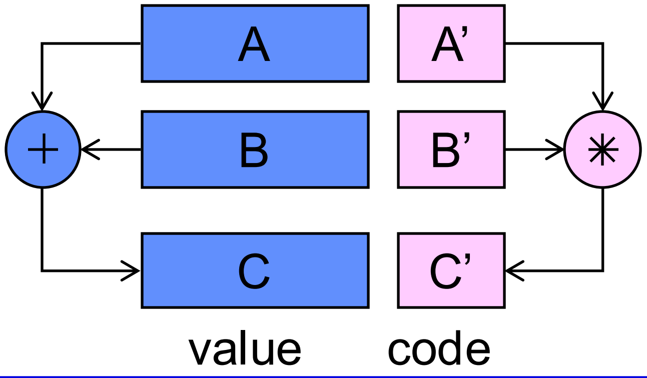
Error detection method by coding (absolute test)

This method is used in network and storage, where error patterns are simple.
It consists in adding a code (parity, checksum, cyclic redundancy check,…) to the useful data that guarantees its integrity.



Coding is more efficient than duplication and comparison.

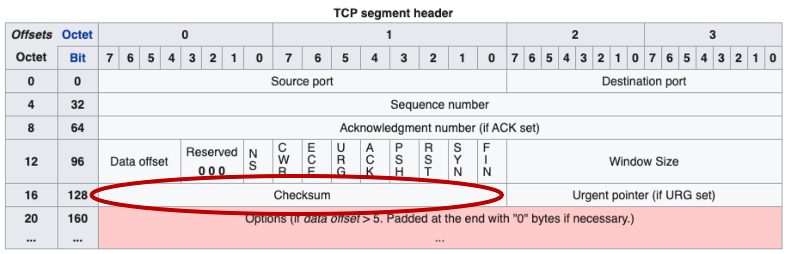
Coding has also been applied to processing elements, but the complexity is huge.
For each operation, a corresponding operation on the check bits has to be done.



Error detection method by coding (absolute test) in practice

TCP – 2 bytes

“The checksum field is the 16 bit one's complement of the one's complement sum of all 16-bit words in the header and text.” – RFC 793

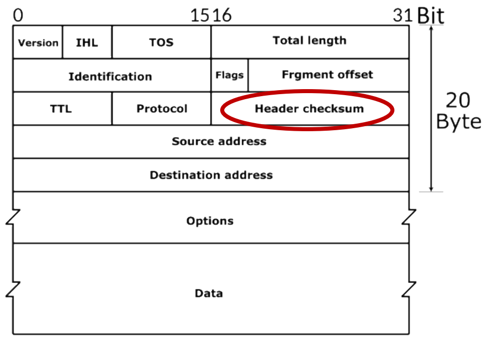


TCP segment header

Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0			NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size															
16	128	Checksum																Urgent pointer (if URG set)															
20	160	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.)																															
...	...	...																															

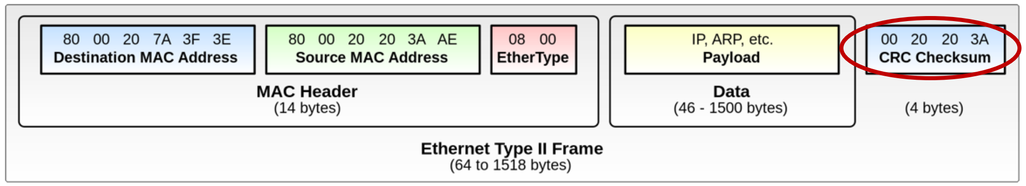
https://en.wikipedia.org/wiki/Transmission_Control_Protocol

IP – 2 bytes



https://en.wikipedia.org/wiki/IPv4

Ethernet – 4 bytes



https://en.wikipedia.org/wiki/Ethernet_frame

Error detection by predicates (absolute check)

The results of a computation are checked against predicates that must be fulfilled,

e.g. the sum of two positive integers is a positive integer

Plausibility checks require knowledge of the specification:

e.g. not all traffic lights may be green at the same time

Plausibility may involve different information sources:

e.g. compare wheel speed with GPS speed

Danger is

-detection of wrong errors

legal situations not foreseen by the application, e.g. flight altitude below sea level (-385m Bar Yehuda Aiport)

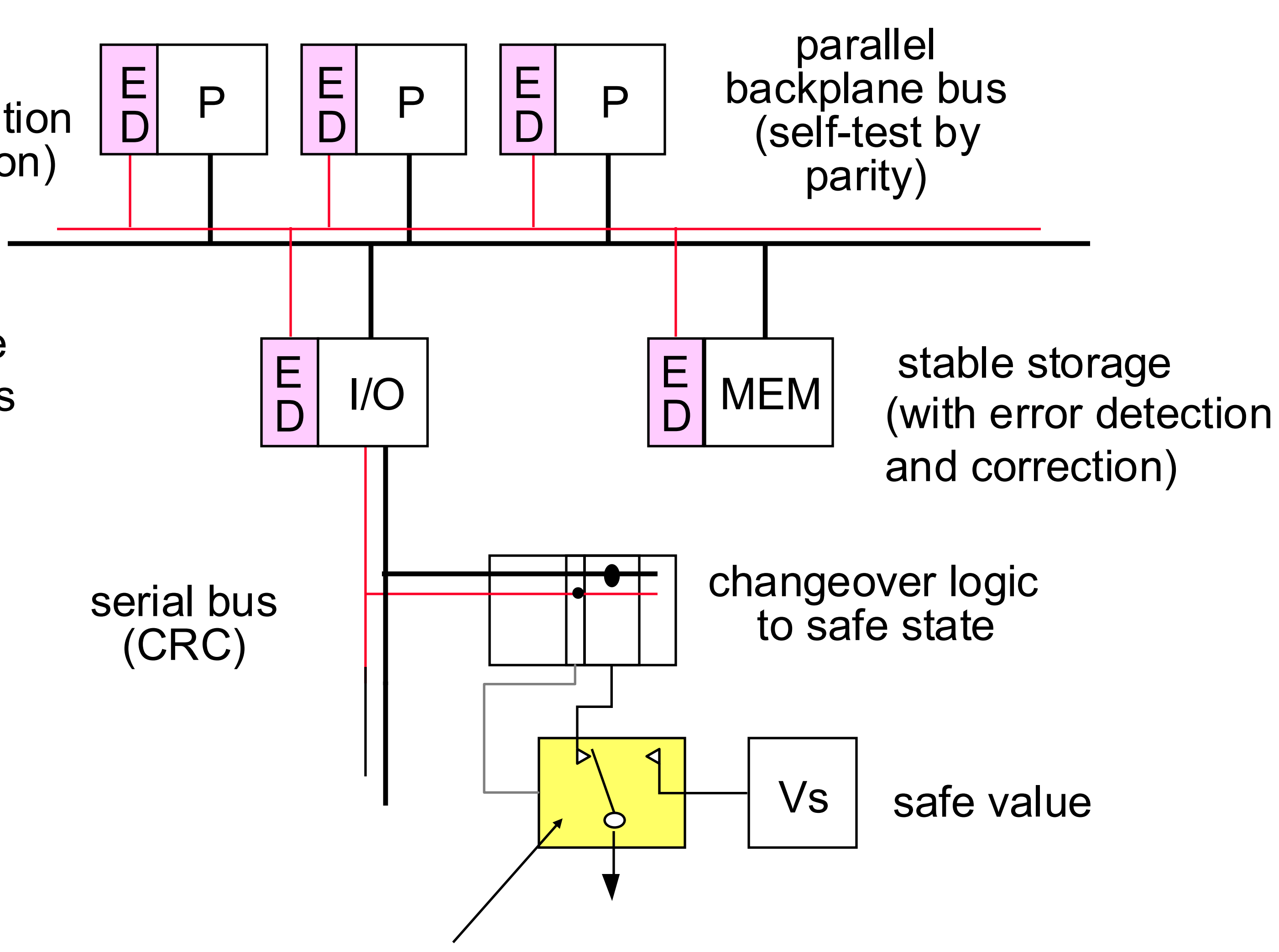
and

-not detection of real errors

the result is wrong, but plausible

Error coverage is not 100% !

Integer Computers: Self-Testing System



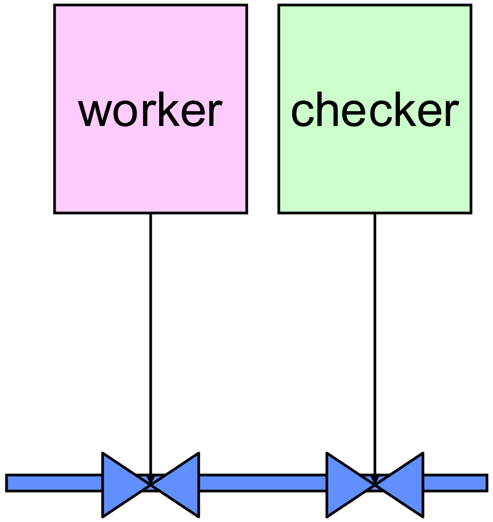
self-testing processors (e.g. duplication & comparison)
E D
P
parallel backplane bus (self-test by parity)
I/O
MEM
stable storage (with error detection and correction)
serial bus (CRC)
changeover logic to safe state
Vs
safe value

Computers include increasingly means to detect their own errors.

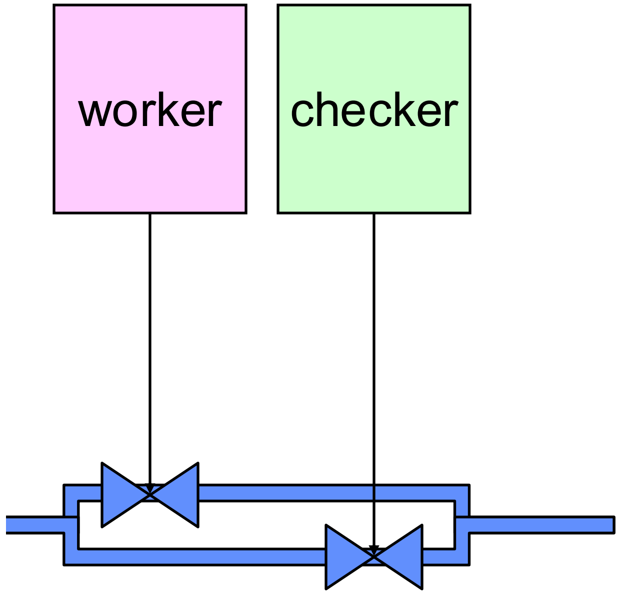
What happens if the safe switch fails ?

Integer outputs: selection by the plant

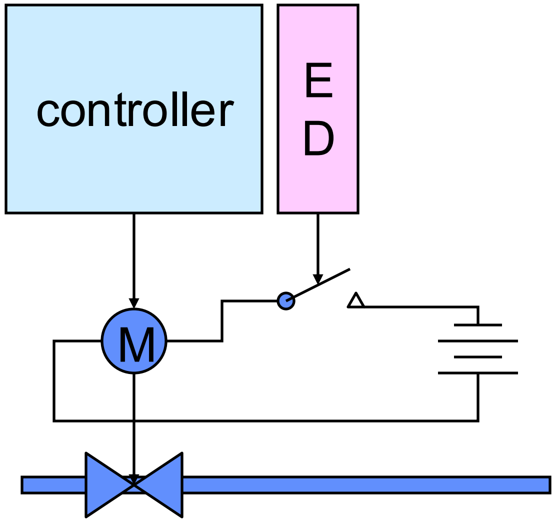
The dual channel should be extended as far as possible into the plant



act if both agree
(workby)



act if any does
(workby)



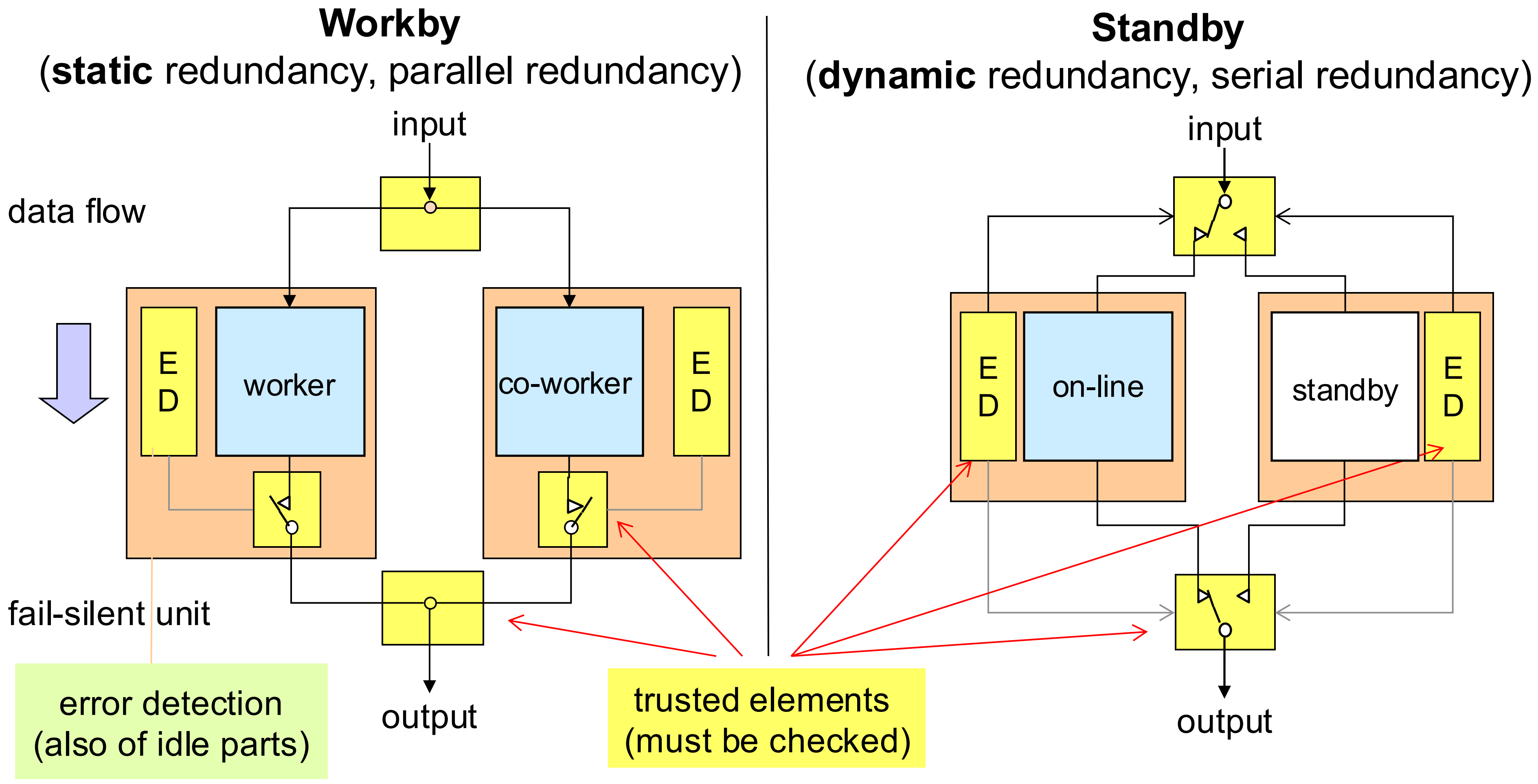
act if error detection agrees
(error detector controls power)

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.
- 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 in order to take over smoothly.
- 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."

Fault-tolerance: the two approaches


Workby
(**static** redundancy, parallel redundancy)
Standby
(**dynamic** redundancy, serial redundancy)
input
data flow
E D
worker
co-worker
E D
fail-silent unit
output
error detection (also of idle parts)
trusted elements (must be checked)
input
E D
on-line
standby
E D
output

both machines modify synchronously their states based on the same inputs in the same manner

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

Workby: 2 out of 3 (2oo3) Computer

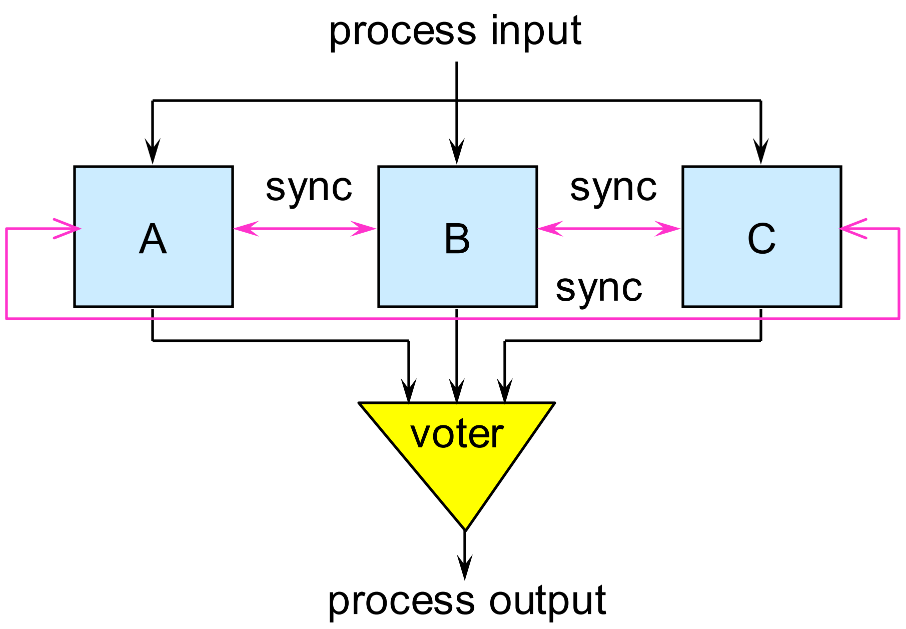
Workby of 3 synchronised and identical units.

– 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)

2oo3v (two out of three with voting)



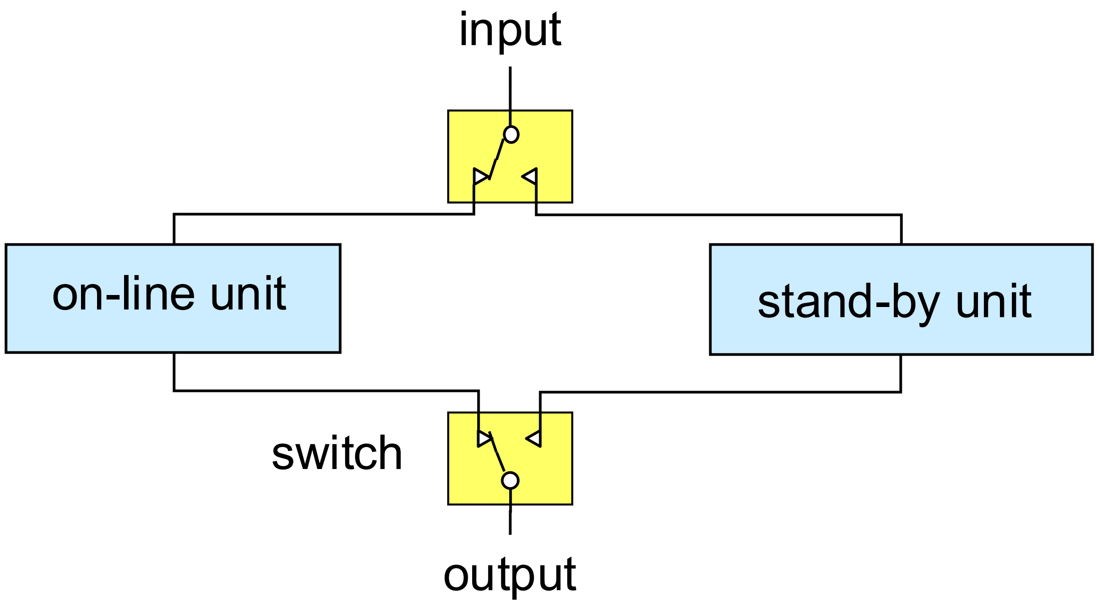
provides integrity (fail-silent) **and** persistency (fail-operate) !

Standby (Dynamic Redundancy)

Redundancy only activated and inserted after an error is detected.

– restart on the same hardware (non-redundant)

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

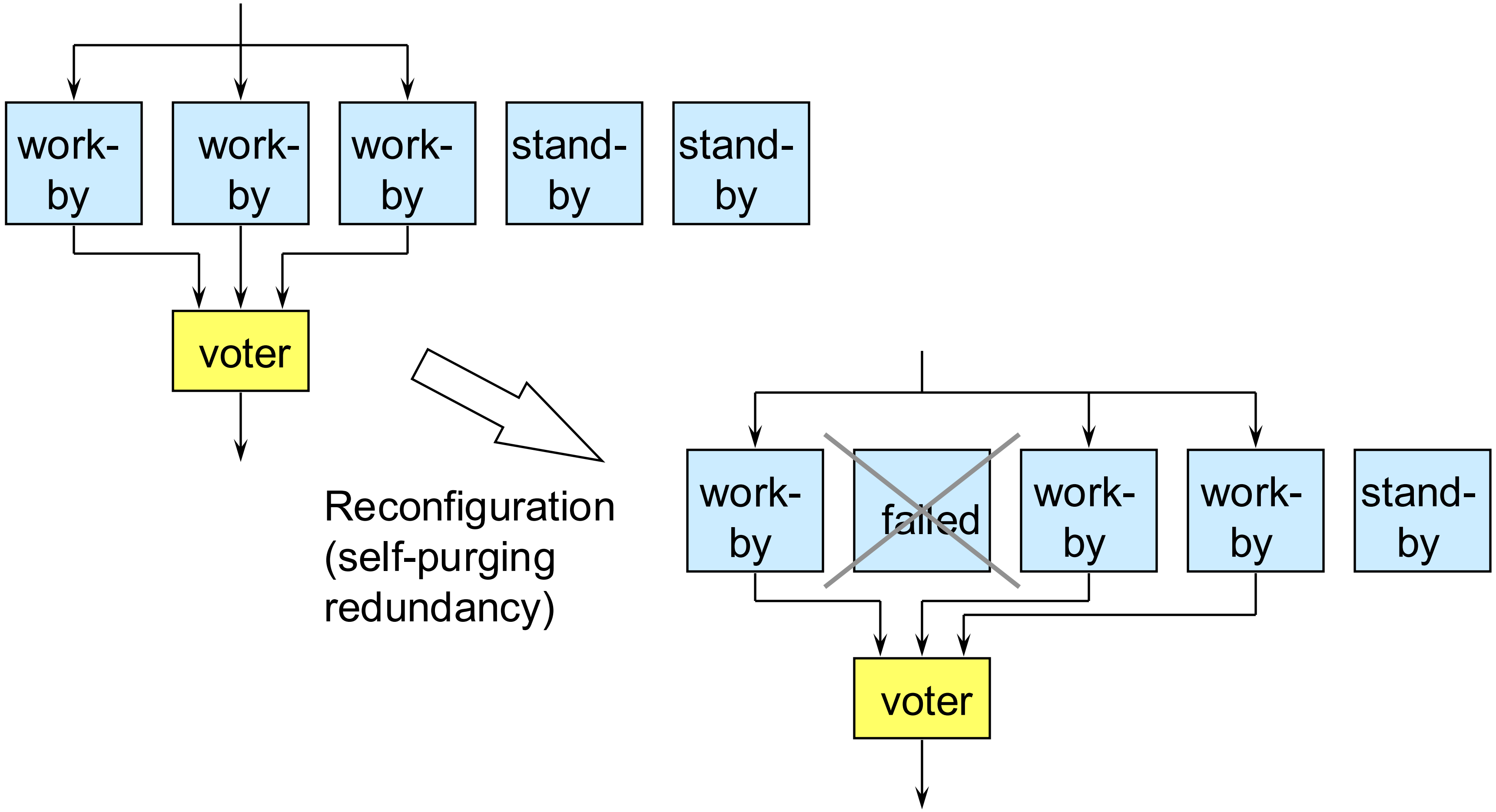


What are standby units used for?

– only as redundancy
– for other functions (that get lower priority in case of primary unit failure)
– better performance ("graceful degradation" in case of failure – wishful thinking)

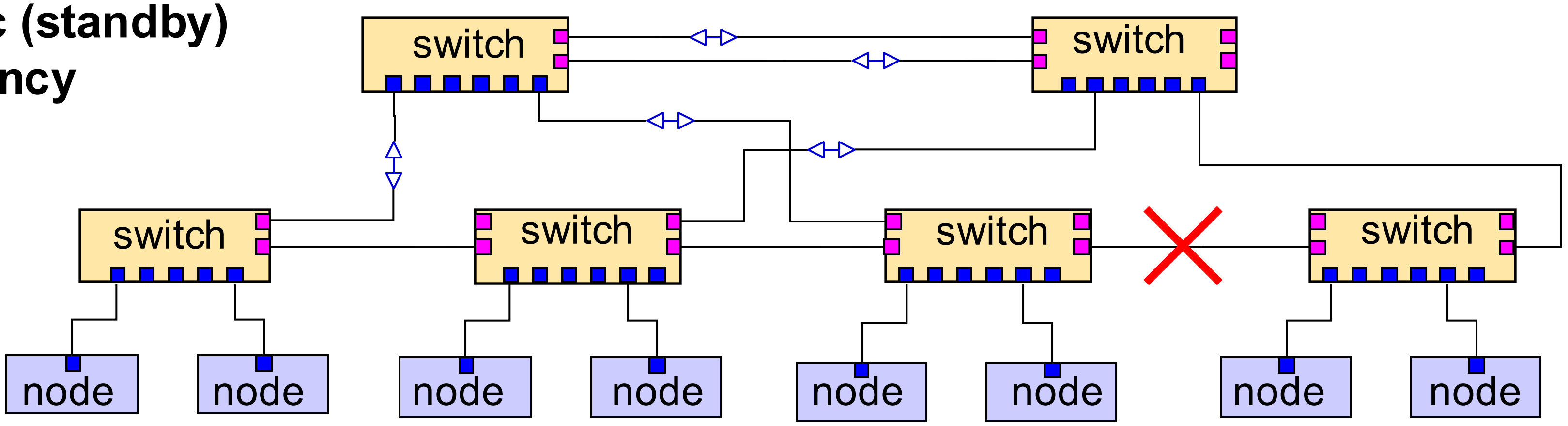
Hybrid Redundancy

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



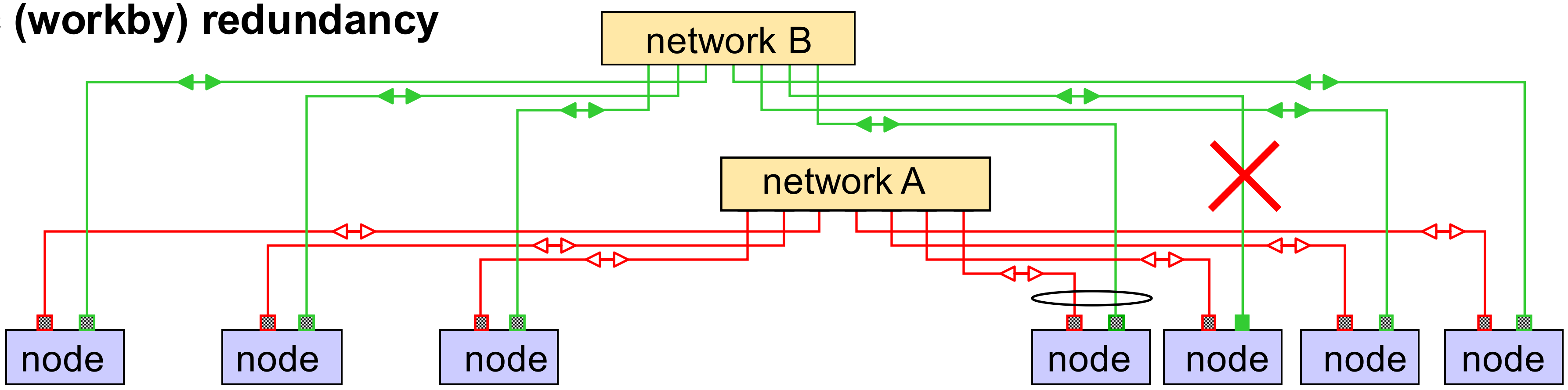
Workby vs. Standby: applies to redundant computer networks

Dynamic (standby) redundancy



nodes are singly attached in case of failure, the switches route the traffic over an other port
(partial redundancy: loss of switch = loss of attached nodes, loss of leaf link = loss of node)

Static (workby) redundancy



nodes send on both networks - in case of failure the nodes work with the remaining network
(partial redundancy: loss of node = loss of function)

General designation

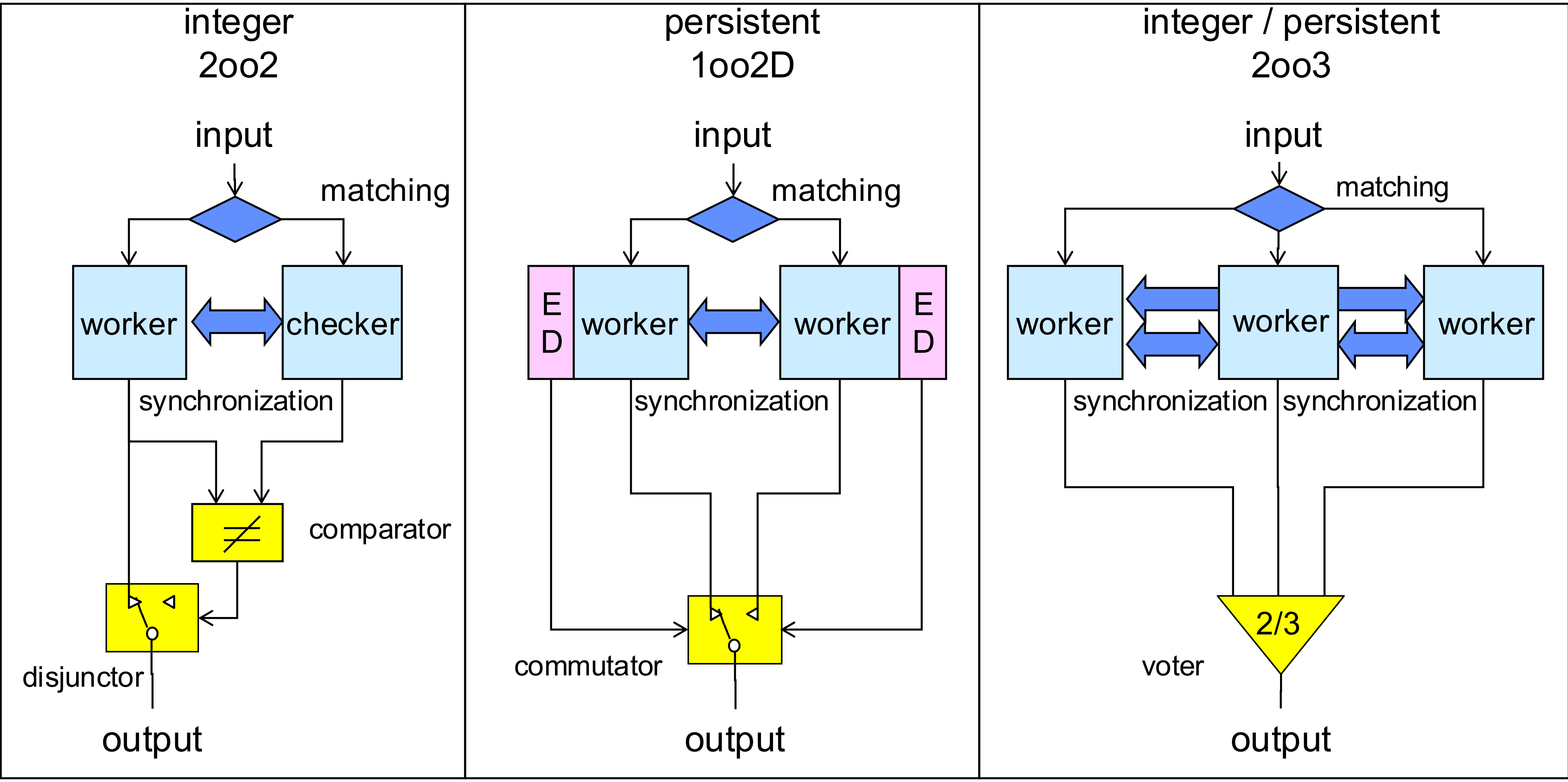
KooN: K out-of N

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 (massive redundancy) architecture

3 Workby

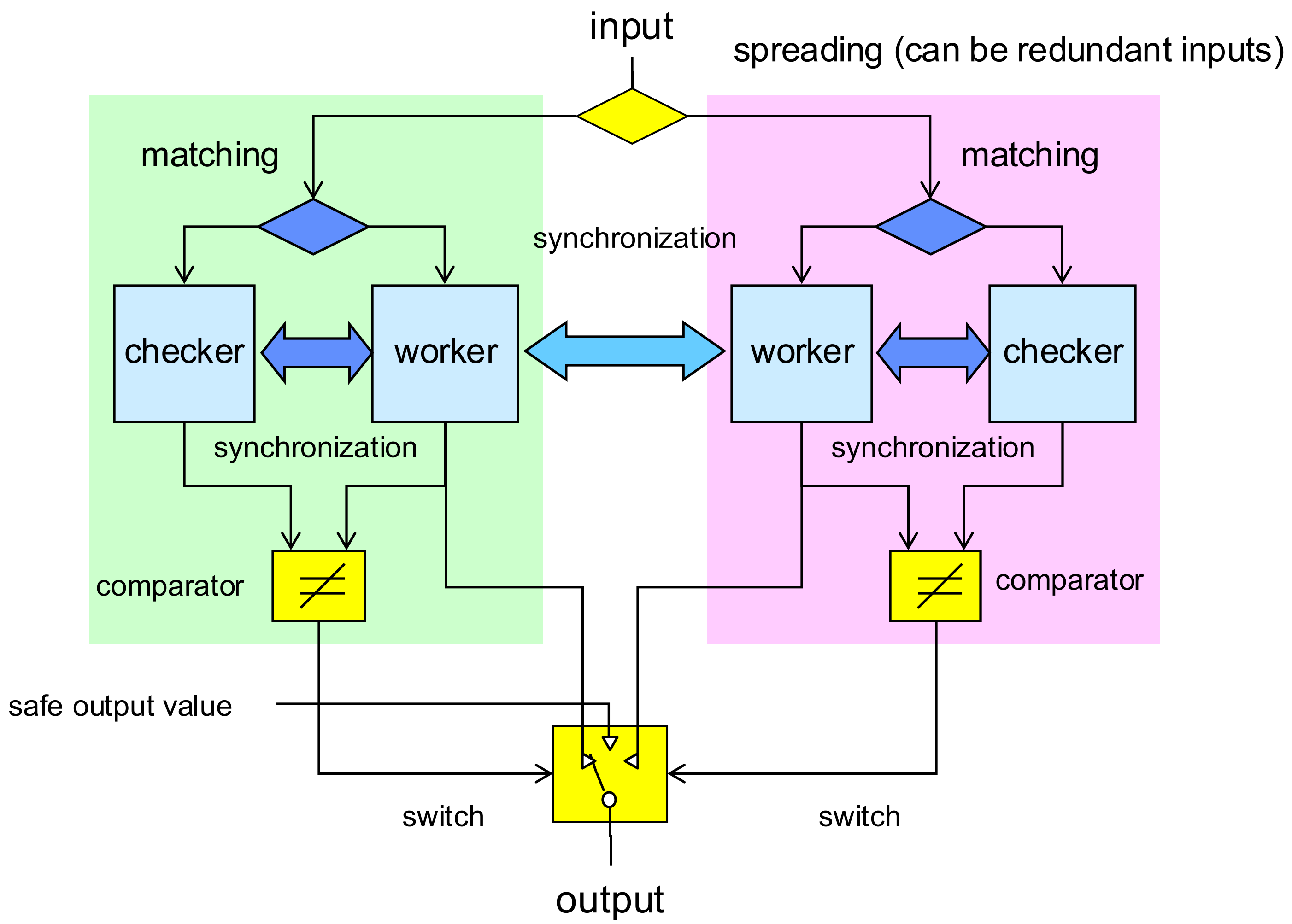
Workby: Fault-Tolerance for both Integrity and Persistency

réserve synchrone, synchrone Redundanz



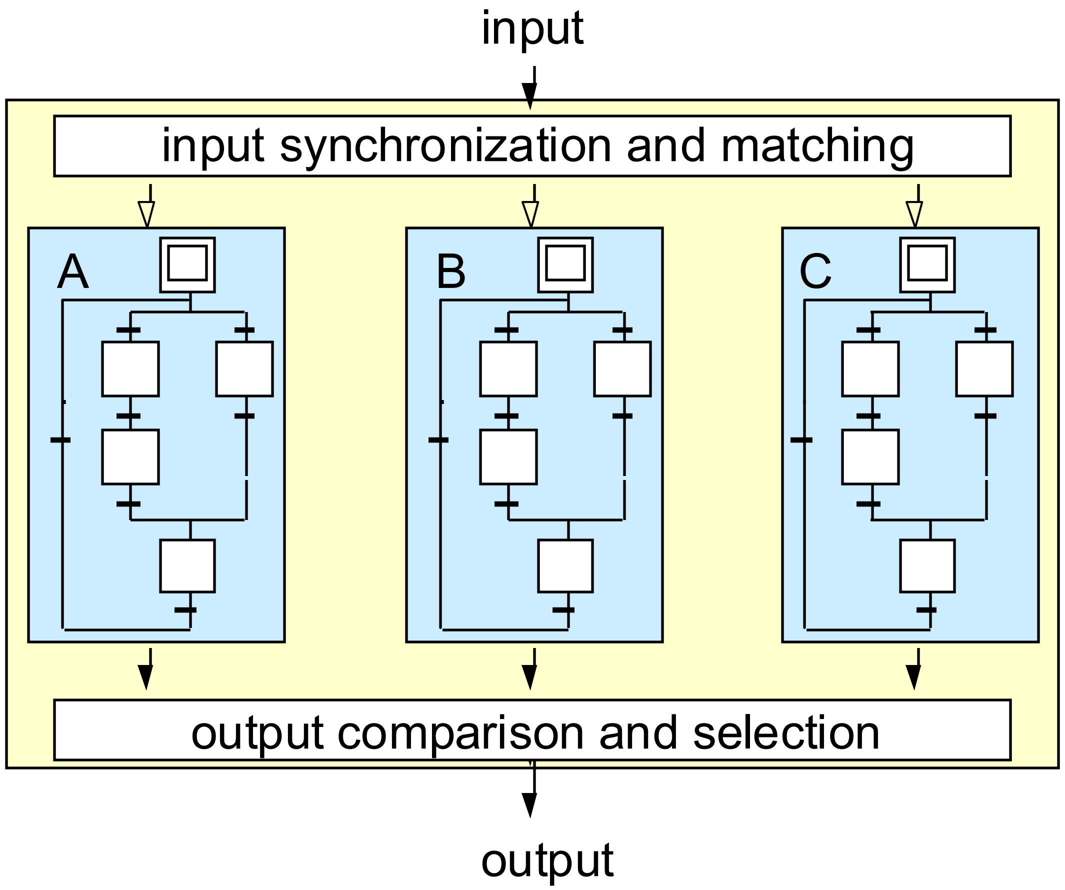
provides integrity (fail-safe) or persistency (fail-operate) and massive redundancy (masking)

“2oo4D” architecture



provides integrity in face of any two unit failures, but cannot provide operation in face of any two unit failure (but 2oo4 it is an accepted designation in safety automation systems)

Workby: Input and Output Handling

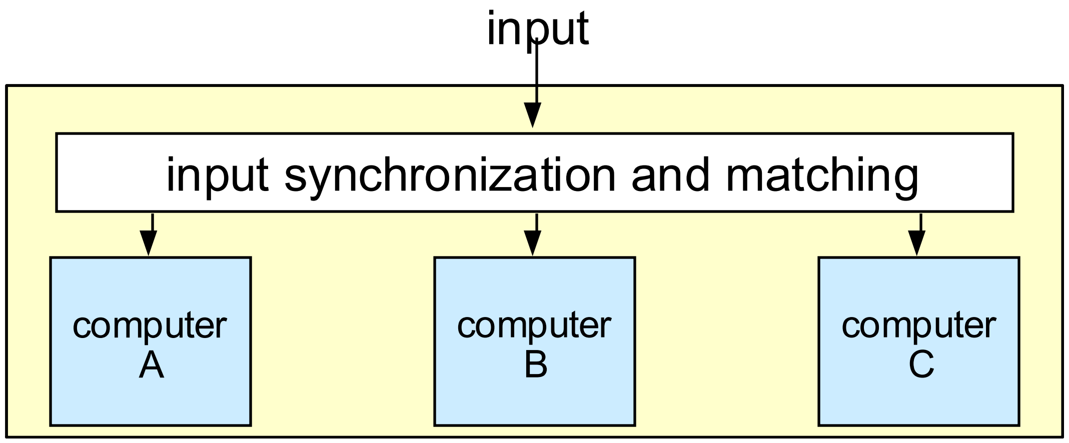

input
input synchronization and matching
A
B
C
output comparison and selection
output

three identical, deterministic, synchronized state machines

Replicated units must receive exactly the same input at the same time (execution step).

Delay (skew, jitter) between outputs must be small enough to allow comparison and smooth switchover.

Workby: Input synchronisation and matching



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

Common signals are not suitable for reaching a consensus

Input from same source: single point of failure, propagation delays causes differences.

Input from different sources: redundant sensors: needs application knowledge.

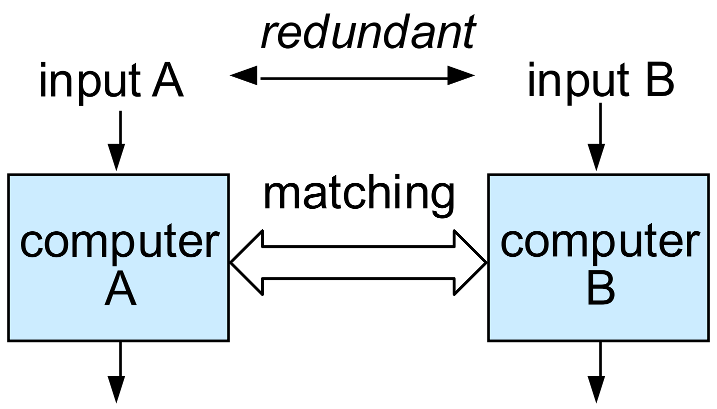
Every replica builds a vector of the value it received directly and the value received from the other units and applies a matching algorithm to it.

All units can then compare the same vector and act on it.

-> requires solving: matching, reliable broadcast, Byzantine problems

c.f. “Reliable and Secure Distributed Programming” from C. Cachin et al. for details on consensus algorithms

Workby: Matching redundant inputs



Redundant inputs may differ in:

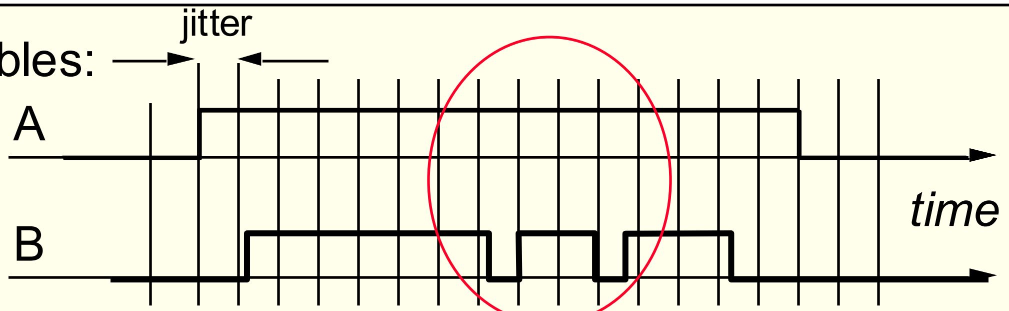
- value (different sensors, sampling)
- timing (even when coming from the same sensor, different delays)

Matching: reaching a consensus value used by all replicas
To reach a consensus, each computer must know the input value received by the other computer(s), through some (often dedicated) communication link.

Workby: Input matching

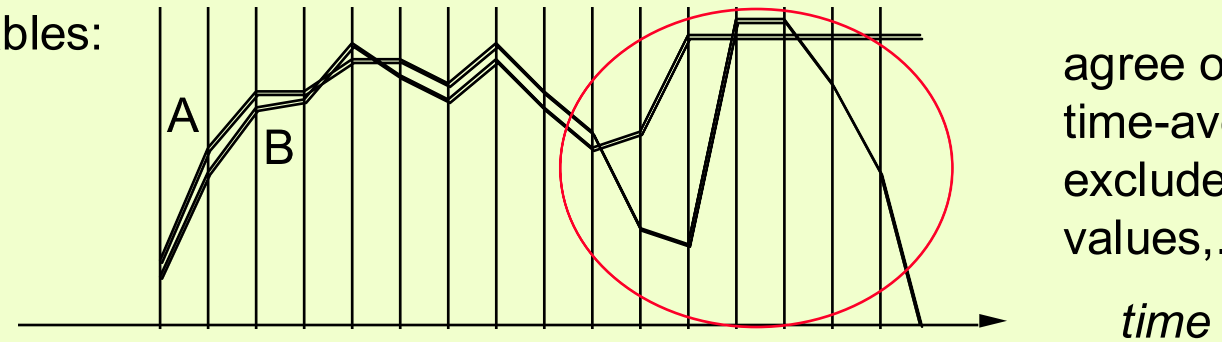
The matched value depends on the semantics of the variables.
Matching needs knowledge of the dynamic and physical behaviour.
Matching stretches over several consecutive values of the variables.

Binary variables:



agree on value stable during a time window, biased decision,...

Analog variables:



agree on median value, time-averaged value, exclude not plausible values,...

Therefore, matching is application-dependent !

Consensus Issue - Byzantine Faults and Failures

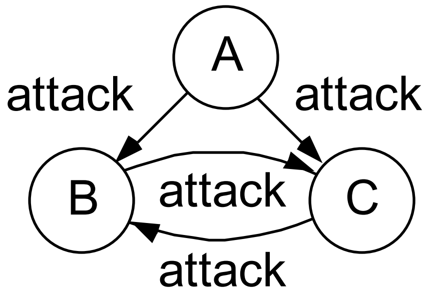
- **Byzantine fault**
  - Any fault presenting different symptoms to different observers

- **Byzantine failure**
  - The loss of a system service due to a Byzantine fault in systems that require consensus

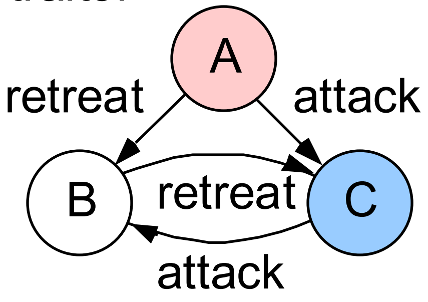
Worst-case scenario… but they may happen!

The Byzantine Generals´ Problem

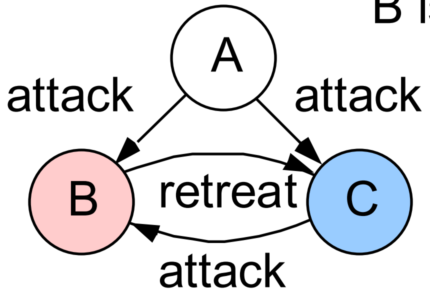
For success, all generals must take the same decision, in spite of 't' traitors.



A is a traitor



B is a traitor



C cannot distinguish who is the traitor, A or B

In the computer world, A can be a faulty processing
unit or the link to B and C can be not reliable.

Exercise: Byzantine Faults

Assume that a dependable computer system consists of four computers.

Each of the computers has a point-to-point data link to the other three computers.

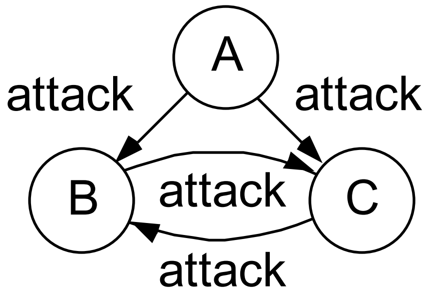
Each of these computers reads an input value from a sensor to which it is connected. However, the sensor reading is unreliable and thus the computer connected to it has to confirm the sensor reading by agreeing with the other computers.

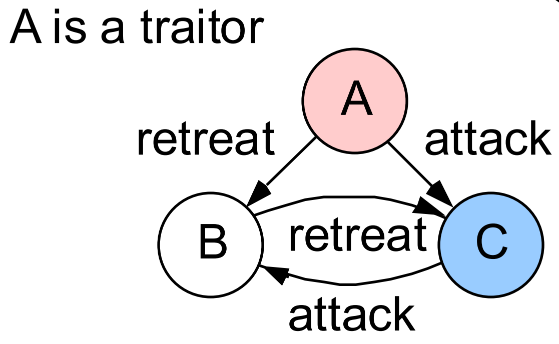
a) Assume that one of the computers fails in such a way that its outputs to different computers can be different. Can the remaining three fault-free computers agree on a common sensor value?

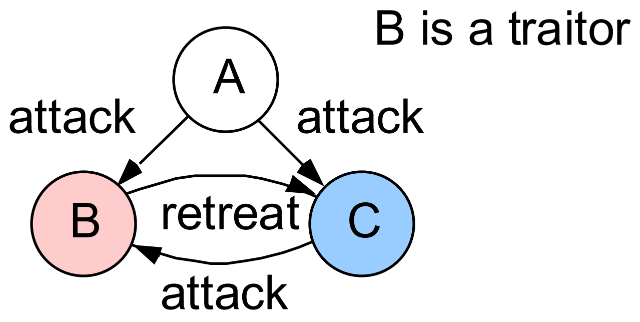
b) Assume that there are two “Byzantine” computers. Is the answer different?

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 authentication)
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 and blockchains

Matching - not so easy
Extract from a Boeing Patent :
Midvalue signal selection and fault detection apparatus and method

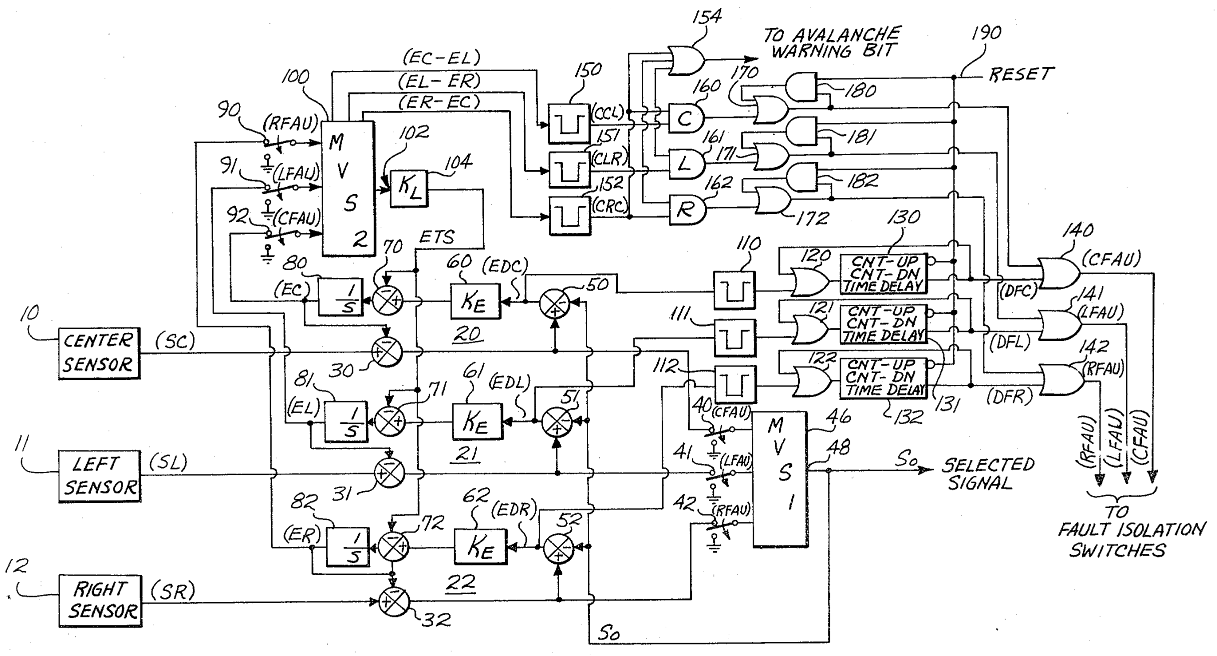
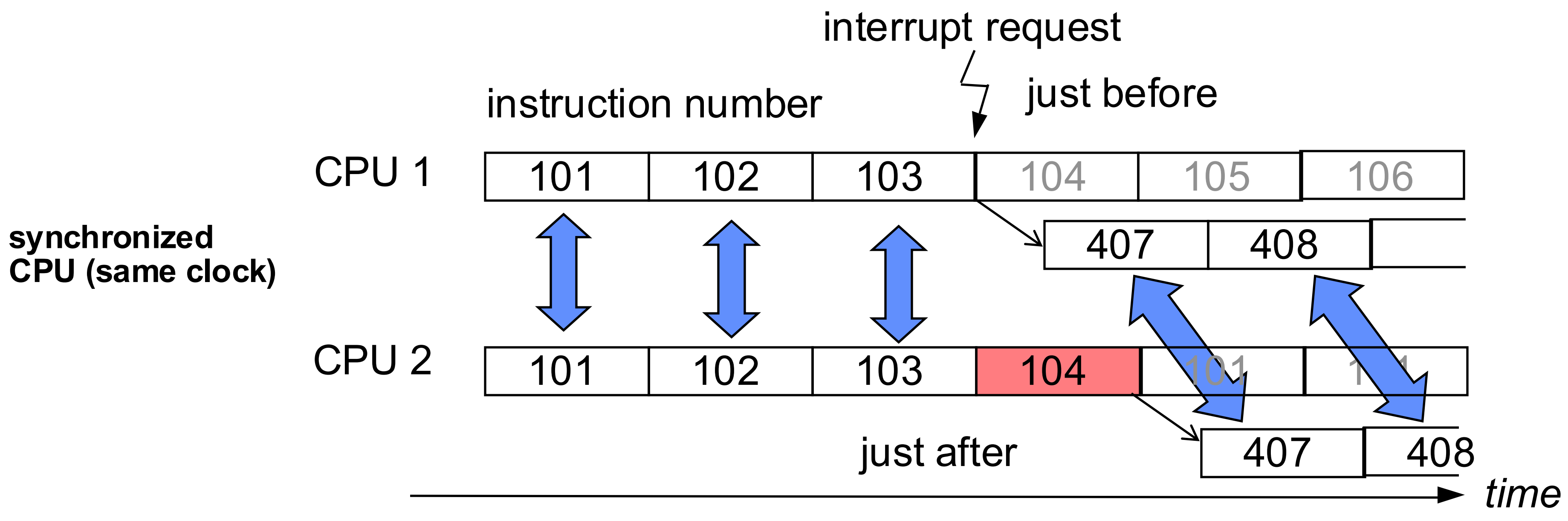


Fig. 1

U.S. Patent Jun. 30, 1981 Sheet 1 of 3 4,276,648

Workby: Interrupt Synchronisation



Instructions may affect the control flow

Interrupts must be matched, like any other input data

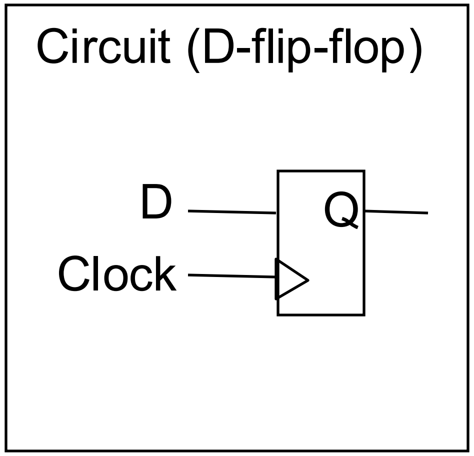
All decisions which affect the control flow (task switch) require previous matching.

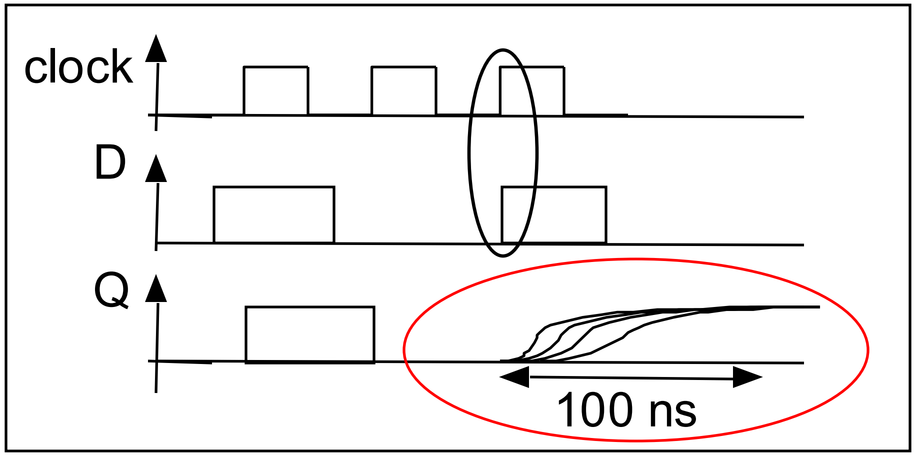
The execution paths diverge, if any action performed is non-identical

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

Workby synchronisation: fundamental metastability limit

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





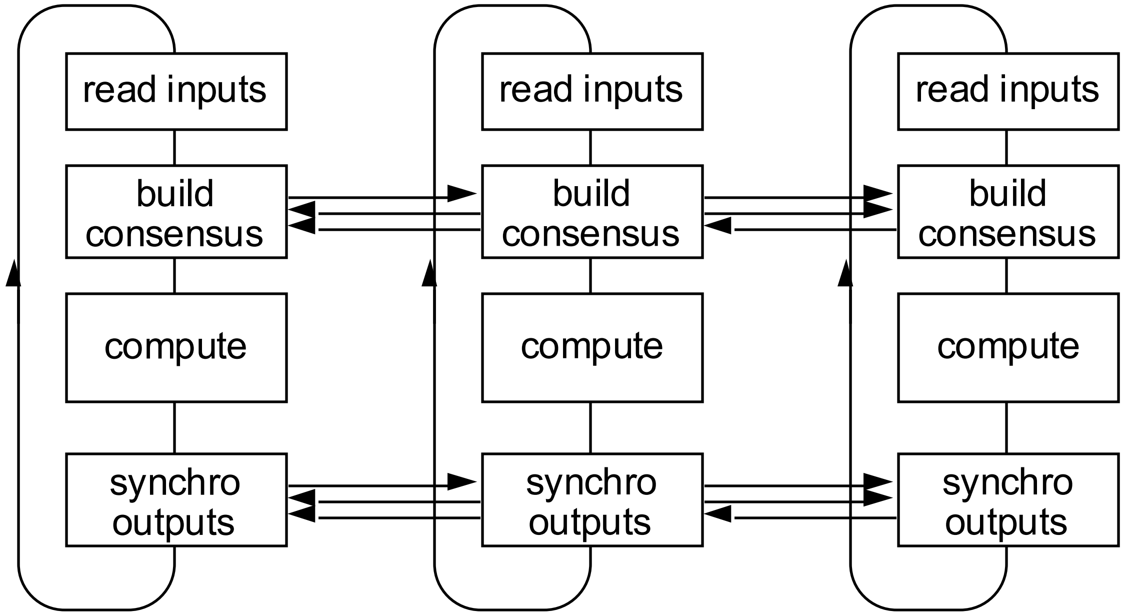
When input signal changes extremely close (simultaneously) to the clock edge

- Output signal may enter a metastable state (value which is neither 0 nor 1)
- Output signal will eventually settle randomly to either 0 or 1

Effect can be reduced (but not totally avoided) by cascading flip-flops (synchronizer)

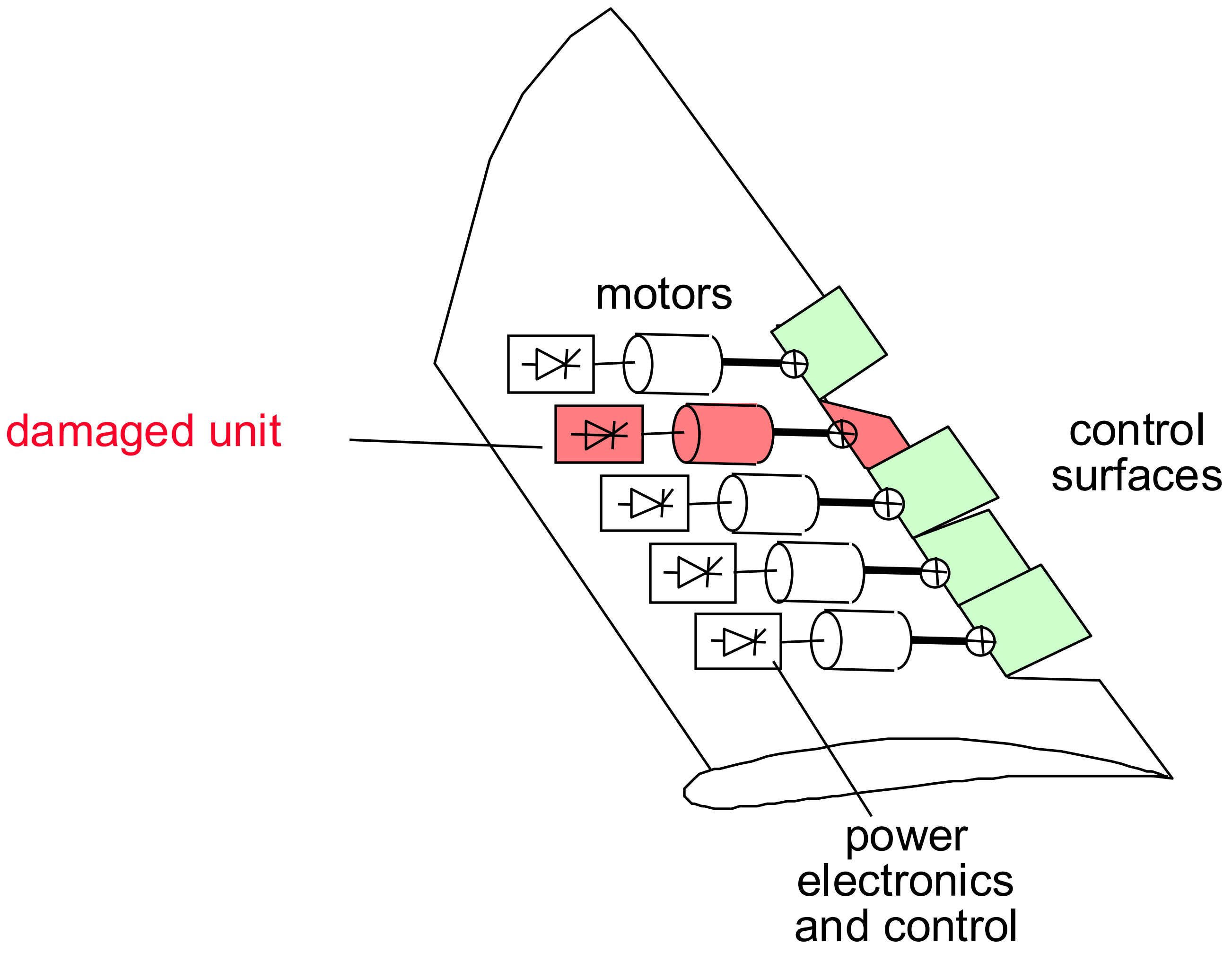
Workby: Output Comparison and Voting

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.

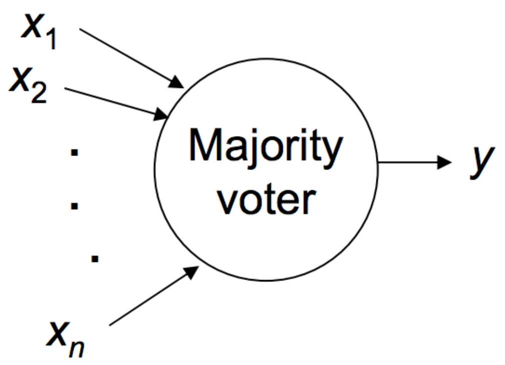
Workby with massive (static) redundancy: the plant votes

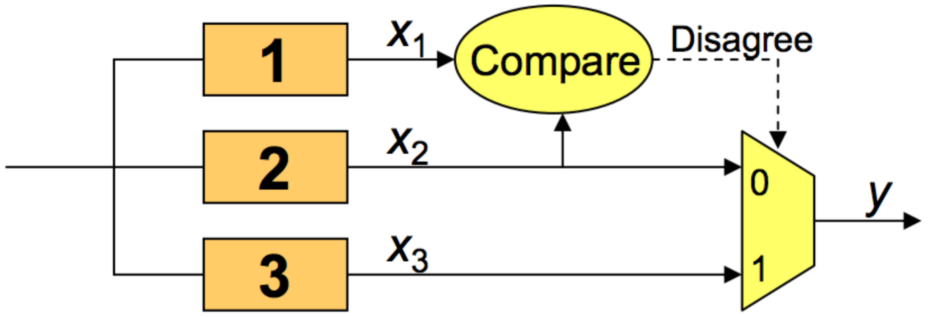


the damaged unit is outvoted by the working units. If the damaged unit can be passivated, (i.e. autodetects its faults and disengages), impact is reduced.

Voters – Not So Simple

- **Majority voting:**
  - Select the value that appears on at least $\lfloor n/2 \rfloor + 1$ of the n inputs
  - Number n of inputs is usually odd, but does not have to be
  - Example: vote(1, 2, 3, 2, 2) = 2
  - Sometimes we can not use strict equality
    - If $|x-y| < \Delta$, then $x = y$
  - Simple implementation with comparator and muxes
    - In case of 3-way disagreement, any value is chosen





Voters

- **Plurality voting**
  - Select the value that occurs the most or a number of time defined by the developper
  - Example: vote(0,1,3,2,3,5,4)=3
- **Median voting**
  - Select the median value of the set of inputs
  - Example: vote (1.00, 3.00, 0.99, 3.00, 1.01) = 1.01
  - It is another way of dealing with approximation
- **Threshold voting**
  - Output is 1 if at least k out of n inputs is 1
  - Majority voting is a special case of threshold voting
- **Weighted threshold voting**

Workby: teaching

When a workby unit is repaired and reintegrated, it is brought to the state of the running unit before it can serve as workby unit again.

To this effect, the state of the running unit is copied to the repaired unit while it is operating.

Since the state of the running unit is continuously changing, the copying must take place much faster than the changes to the state.

This is only possible if the state is handled at a high abstraction level (for speed reasons) and states are tagged (to retransmit them if they changed in between).

State restoration

State saving and restoring applies in a modified form to reintegration of repaired units.

This applies especially to standby computers, that must be reinitialized to the state of the running machine.

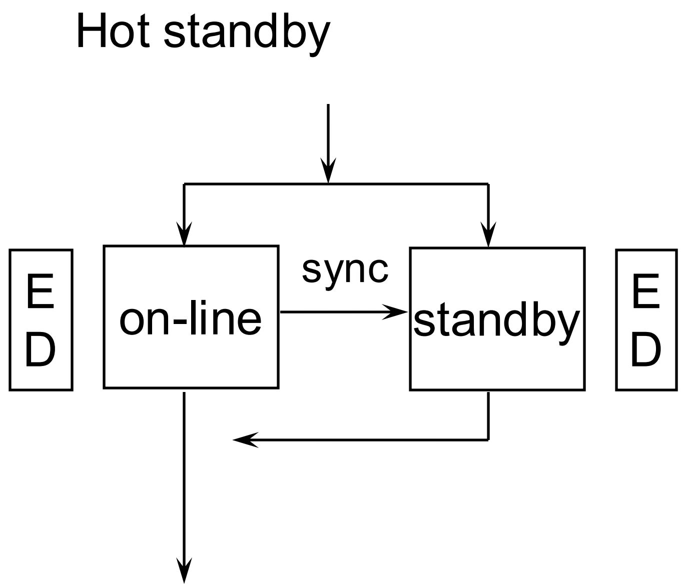
This requires the on-line unit to spare a portion of its computing power to restore the state of the reintegrated unit and bring it to synchronism.

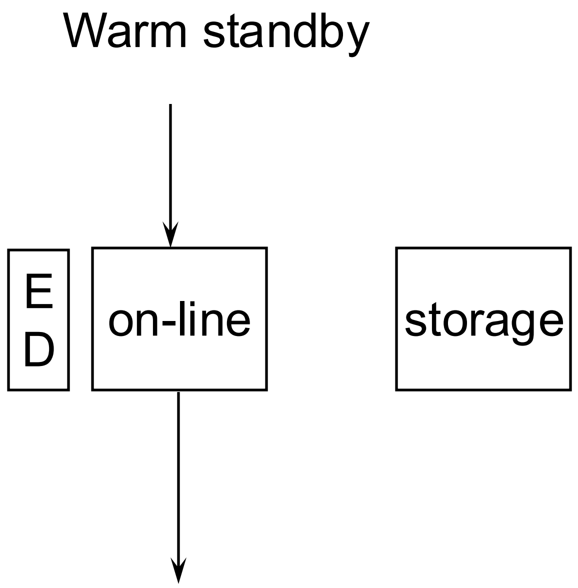
This is a more challenging task than just switching over in case of failure.

4 Standby

réserve asynchrone, *unbeteiligte Redundanz*

Standby





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

Standby is not operational
Error detection needed.
Long switchover period with loss of state info.
Smaller failure rate of storage unit

Standby: cold, warm hot

Standby consists in restarting a failed computation from a known-good state.

The basic techniques for state saving are the same as for the back-up in a personal computer or on mainframe computers.

At the simplest, restart can be done on the same machine when only transient faults are considered -> “automatic restart”, “warm start”.

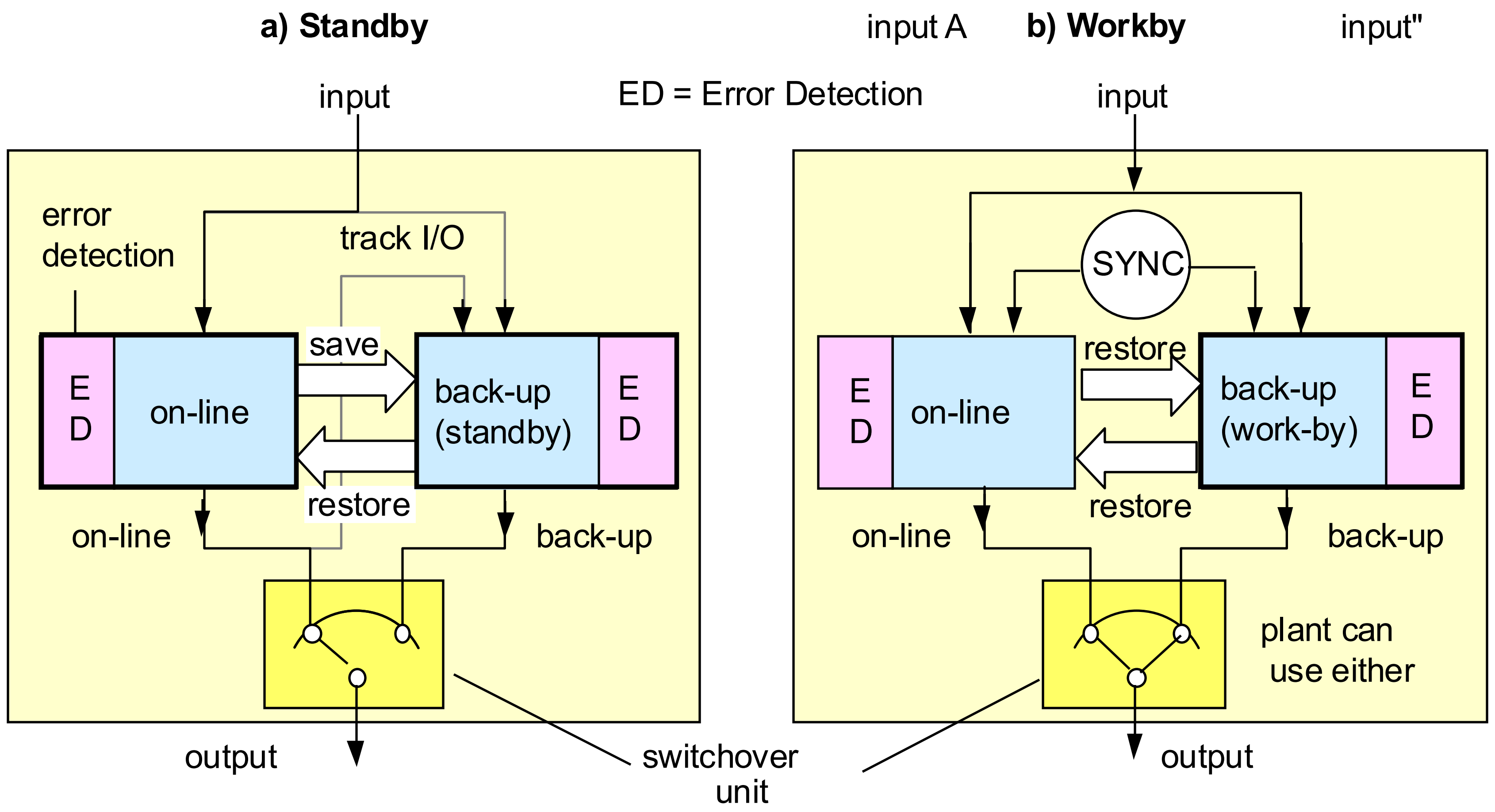
Restart after repair requires a more elaborate state saving.

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 asynchronous input/outputs.

Actualization of state in standby vs. workby

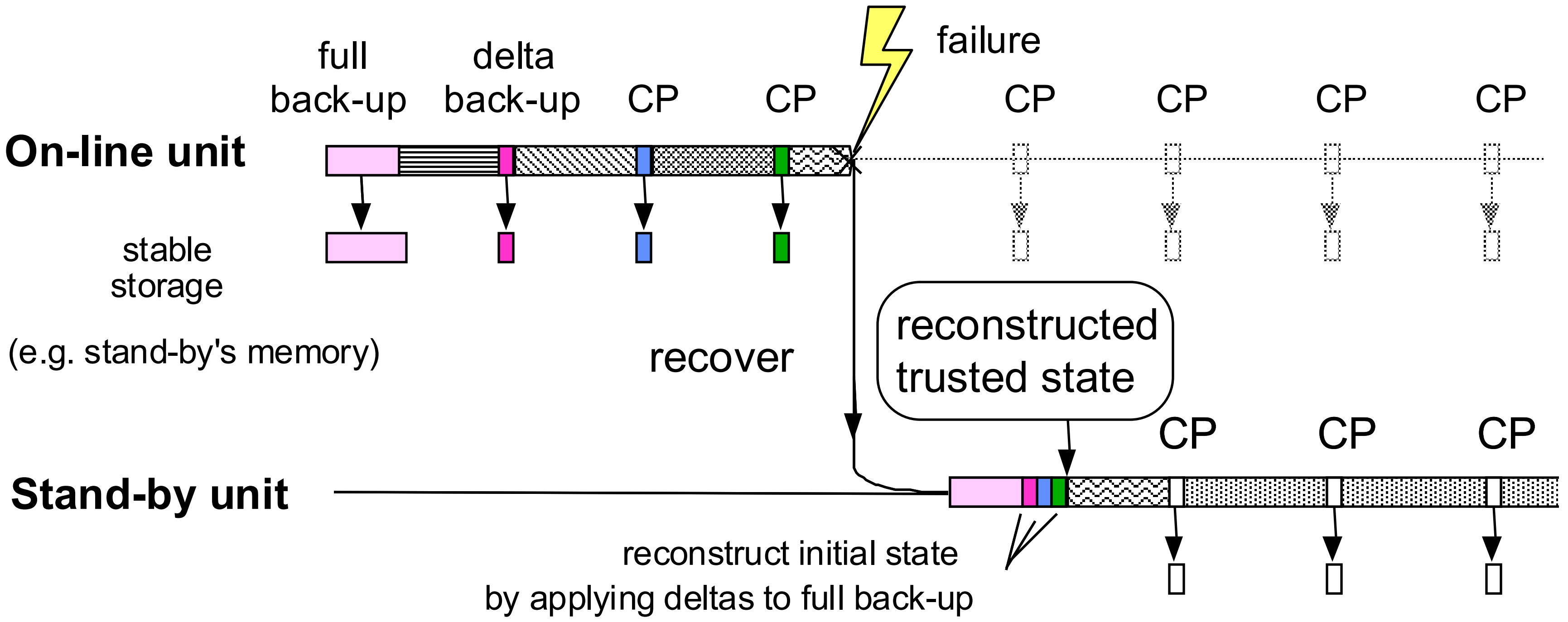


The on-line unit regularly actualises the state of the stand-by unit, which otherwise remains passive.

on-line and back-up are synchronized by parallel operation (synchronized inputs) restore for hot reintegration, no save.

Standby: Checkpointing for state transfer

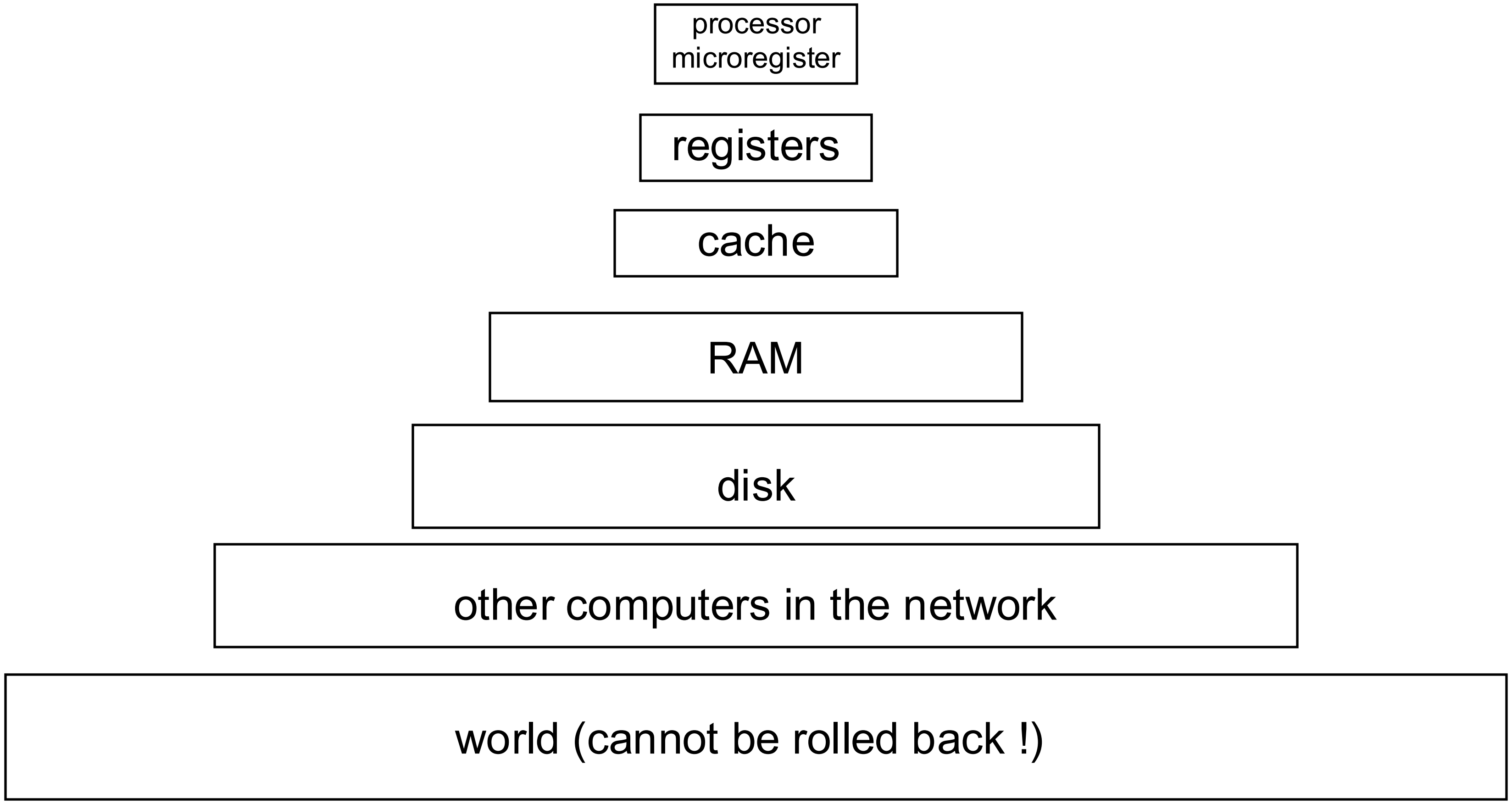
Checkpoints save enough information to reconstruct a previous, known-good state.
To limit the data to save (checkpoint duration, distance between checkpoints),
only the parts of the state modified since last checkpoint are saved.



Checkpointing requires identification of the parts of the context modified since last checkpoint – this is application dependent !
To speed up recovery, the stand-by can apply the deltas to its state continuously.

Standby: 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 (trusted storage), on which are not (volatile storage) and on which parts are relevant.


processor microregister
registers
cache
RAM
disk
other computers in the network
world (cannot be rolled back !)

Standby: Checkpointing Strategy

Checkpoints are difficult to insert 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 relevant 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

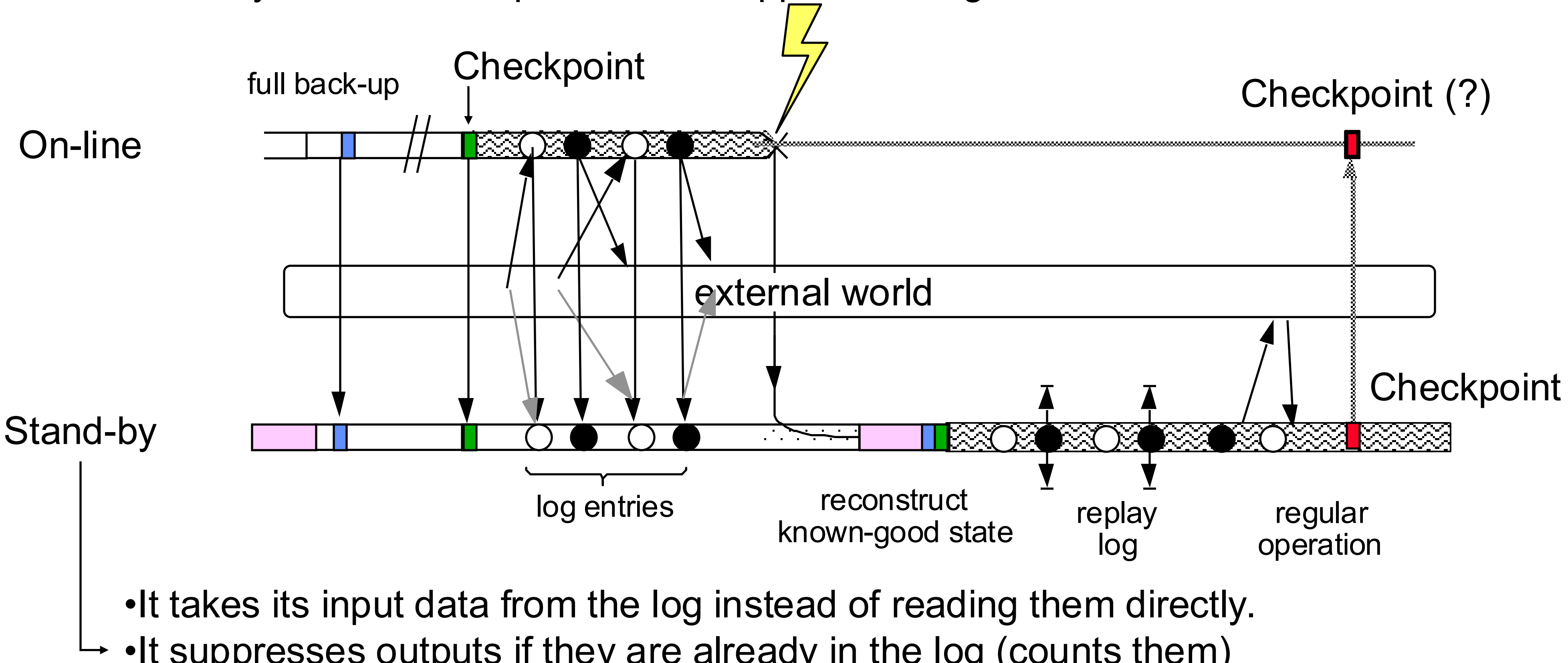
• after the execution of an instruction, microregisters are no more relevant.

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 ?

Standby: Logging

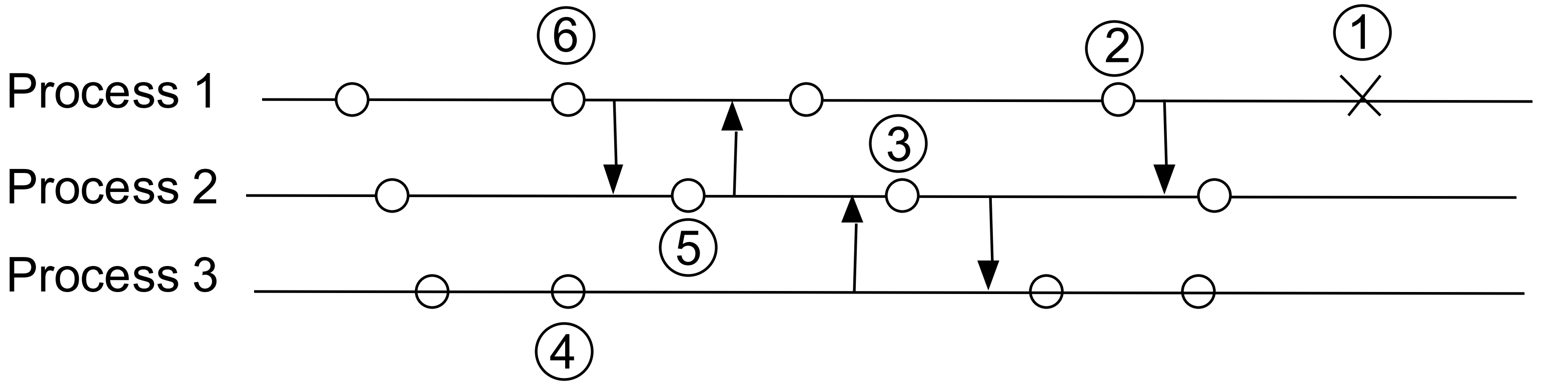
For faster recovery and closer checkpointing, the stand-by monitors the input-output interactions of the on-line unit in an interaction log.
After reconstructing a known-good state from the full copy and incremental back-ups, the stand-by resumes computation and applies the log of interactions to it:



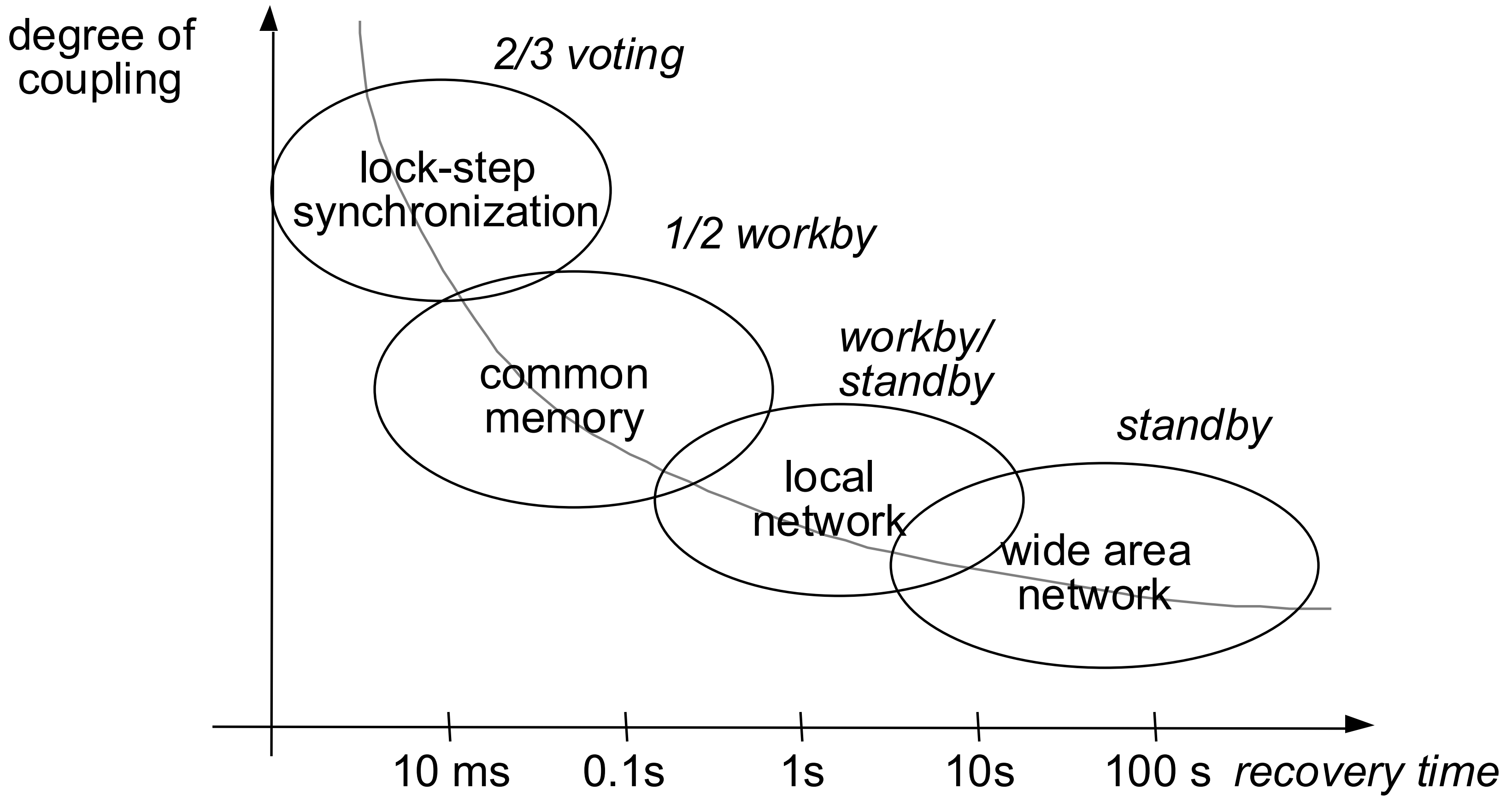
- •It takes its input data from the log instead of reading them directly.
- •It suppresses outputs if they are already in the log (counts them)
- •It resumes normal computations (and checkpointing) when the log is void.

Standby: 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 preceded by a checkpoint.



Recovery times for various architectures

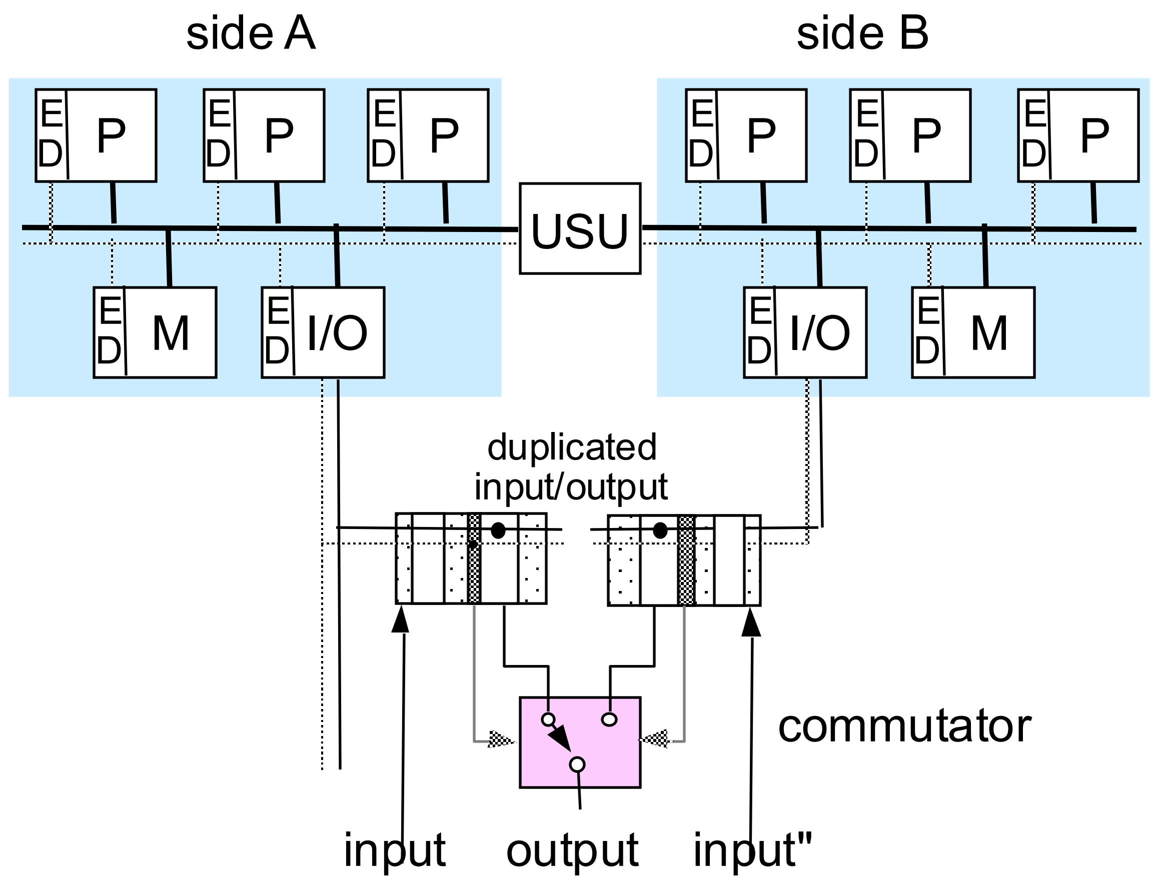


The time available for recovery depends on the tolerance of the plant against outages.

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

5 Example Architectures

ABB 1/2 Multiprocessor for HVDC substation



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

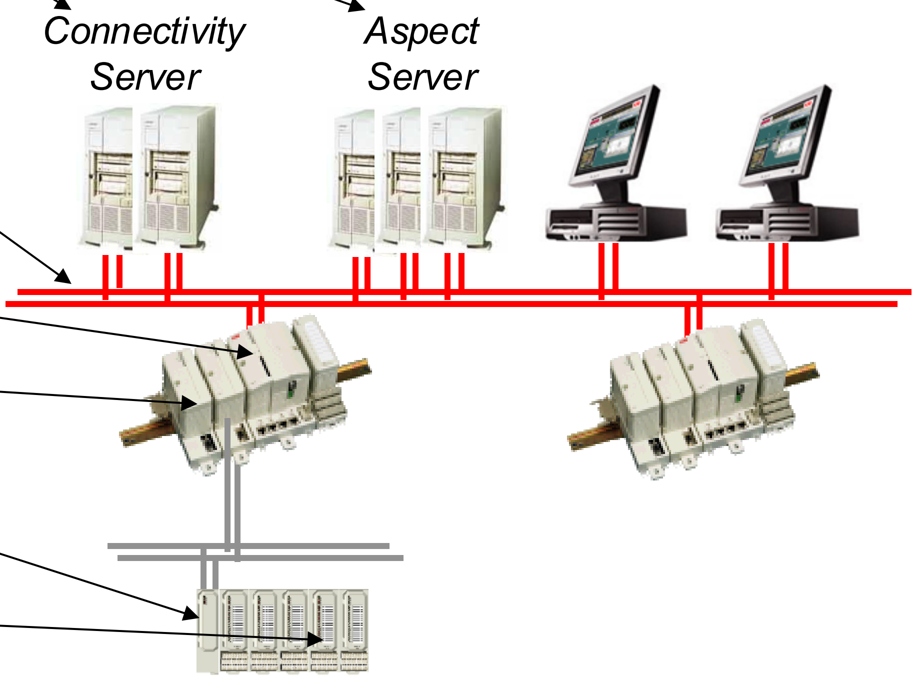
Fieldbus line redundancy
- Dual physical lines

Fieldbus device redundancy
- Duplicated bus interfaces

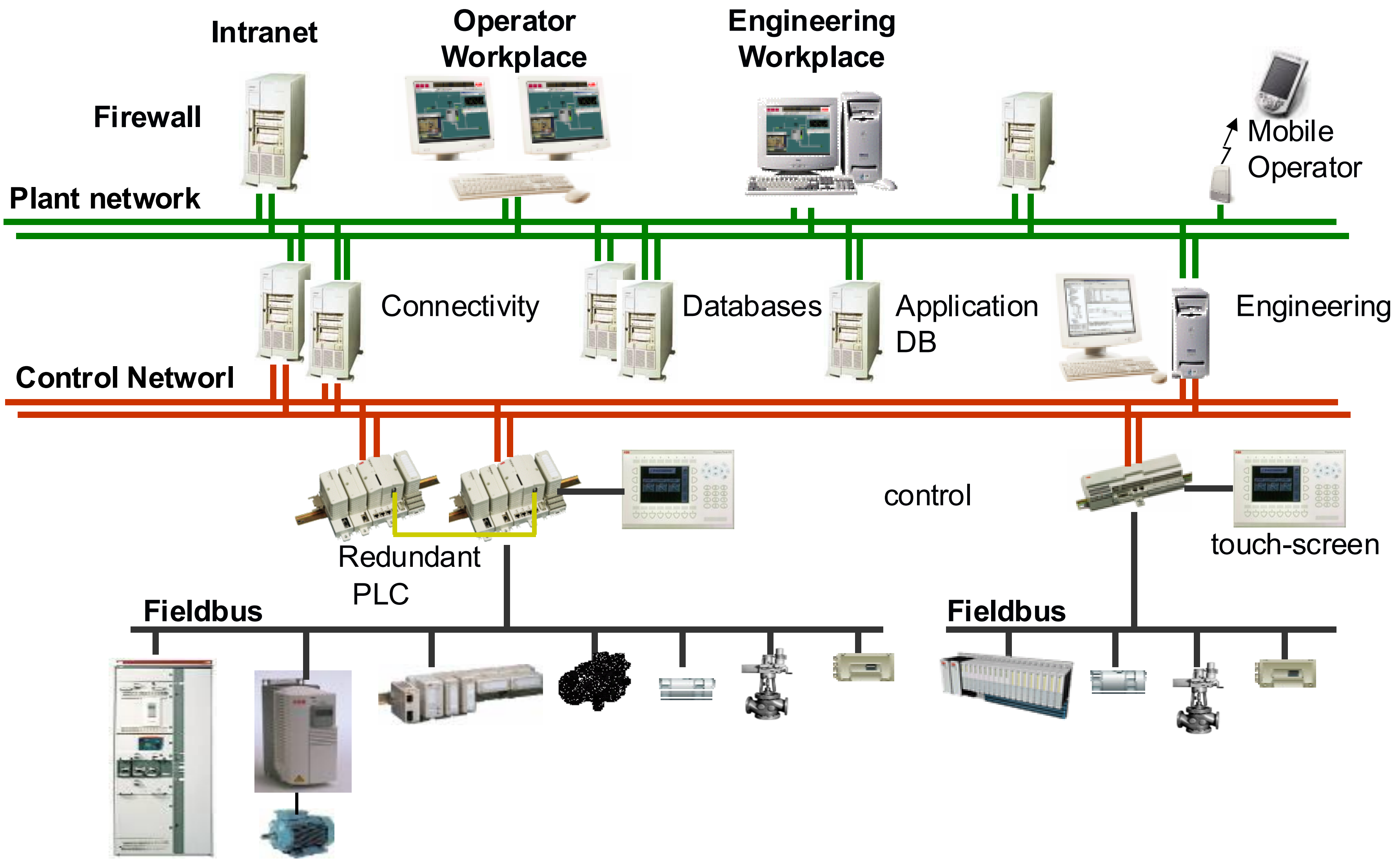
Redundant I/O, remote, 1oo2

Dual power supplies
- Supervision of A and B power lines

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

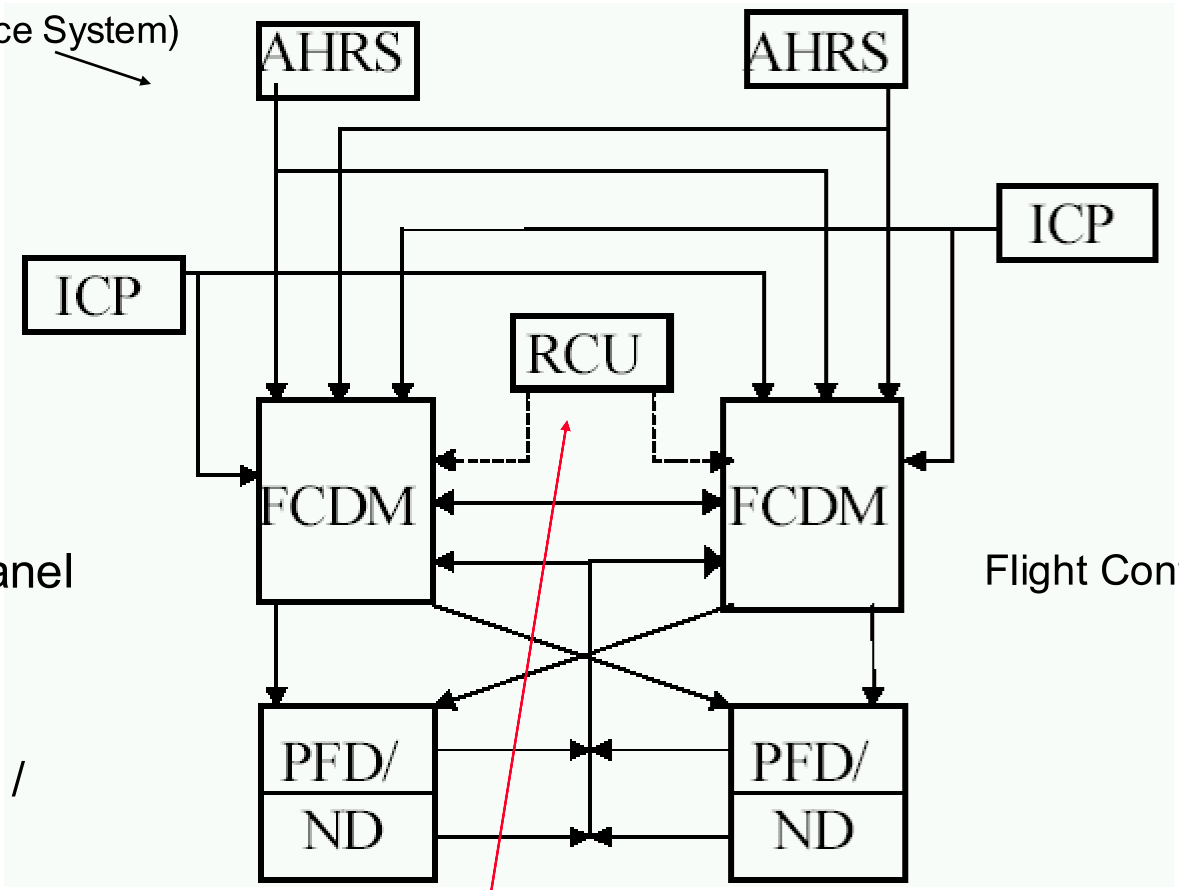


Full redundant system



Example: Flight Control Display Module for helicopters

sensors
(Attitude Heading Reference System)



instrument control panel

Flight Control Display Module

primary flight display /
navigation display

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

source: National Aerospace Laboratory, NLR

B777: airplane

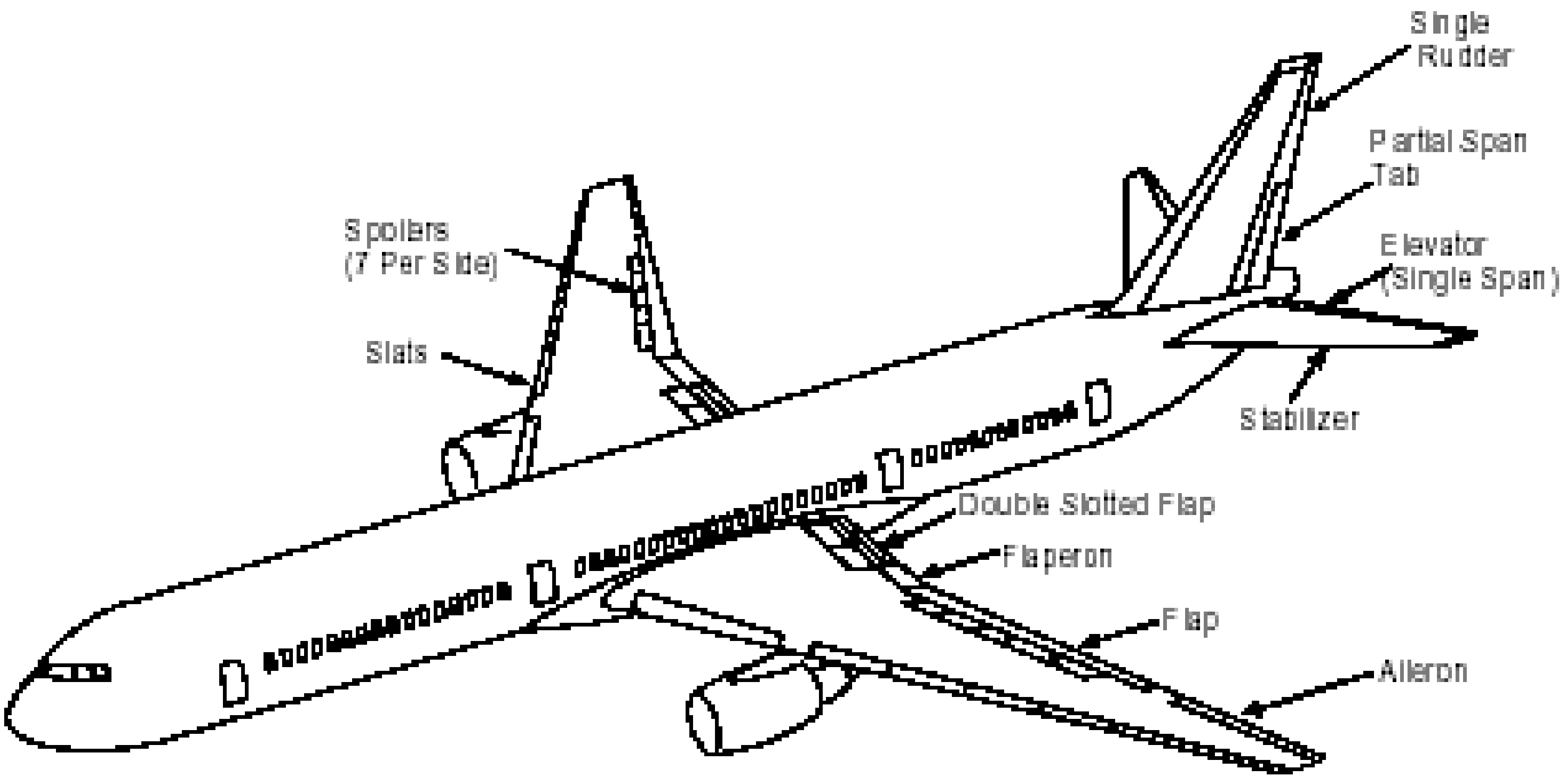


FIGURE 1 777 FLIGHT CONTROL SURFACES

Source: Boeing

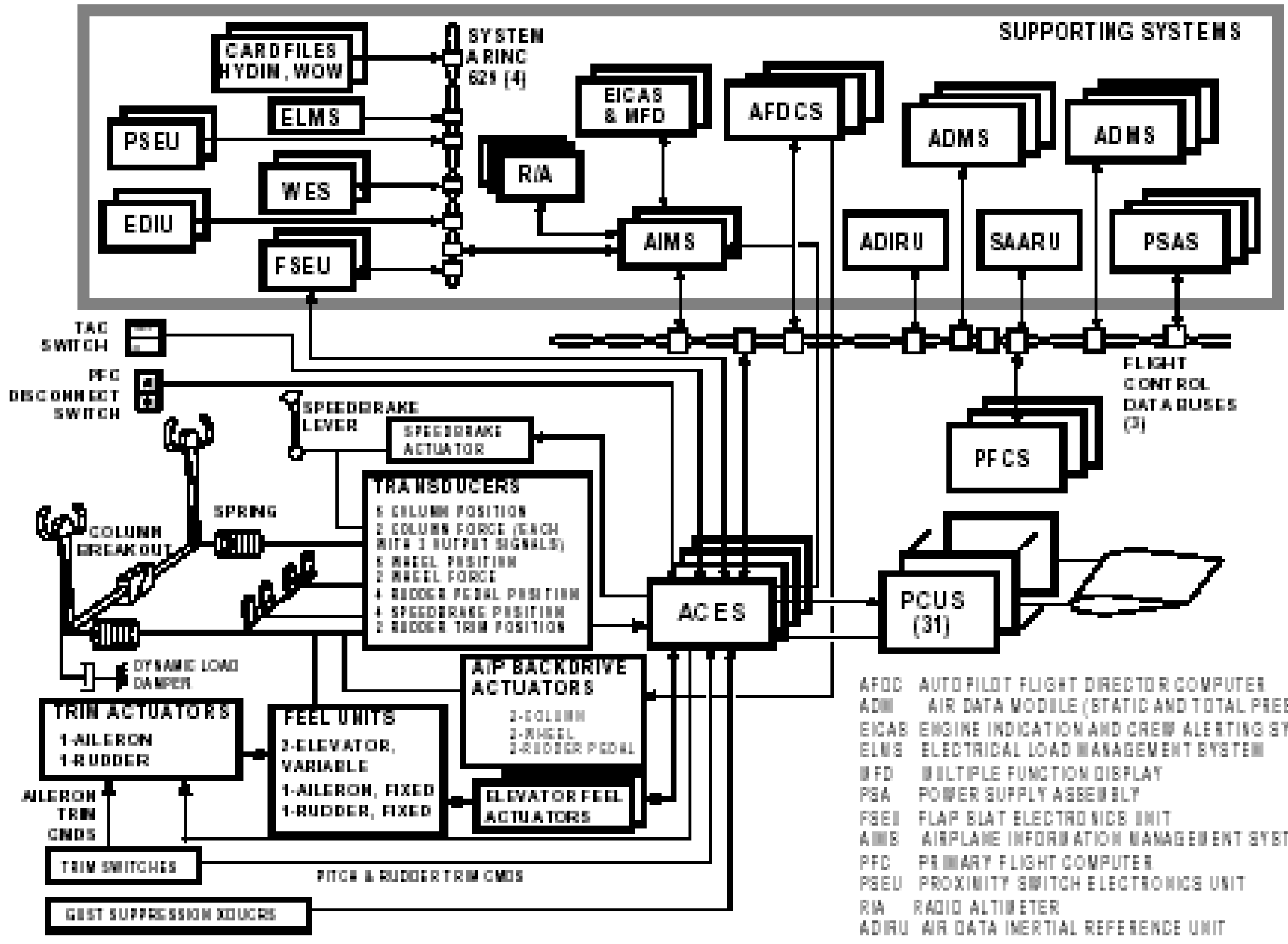
Triple-triple redundant 777 primary flight computer

Abstract:

"The flight control system for the Boeing 777 airplane is a Fly-By-Wire (FBW) system. The FBW system must meet extremely high levels of functional integrity and availability. The heart of the FBW concept is the **use of triple redundancy for all hardware resources: computing system, airplane electrical power, hydraulic power and communication path**. The Primary Flight Computer (PFC) is the central computation element of the FBW system. The triple modular redundancy (TMR) concept also applies to the PFC architectural design. Further, the N-version dissimilarity issue is integrated to the TMR concept. The PFCs consist of three similar channels (of the same part number), and each channel contains three dissimilar computation lanes. The 777 program design is to select the ARINC 629 bus as the communication media for the FBW."

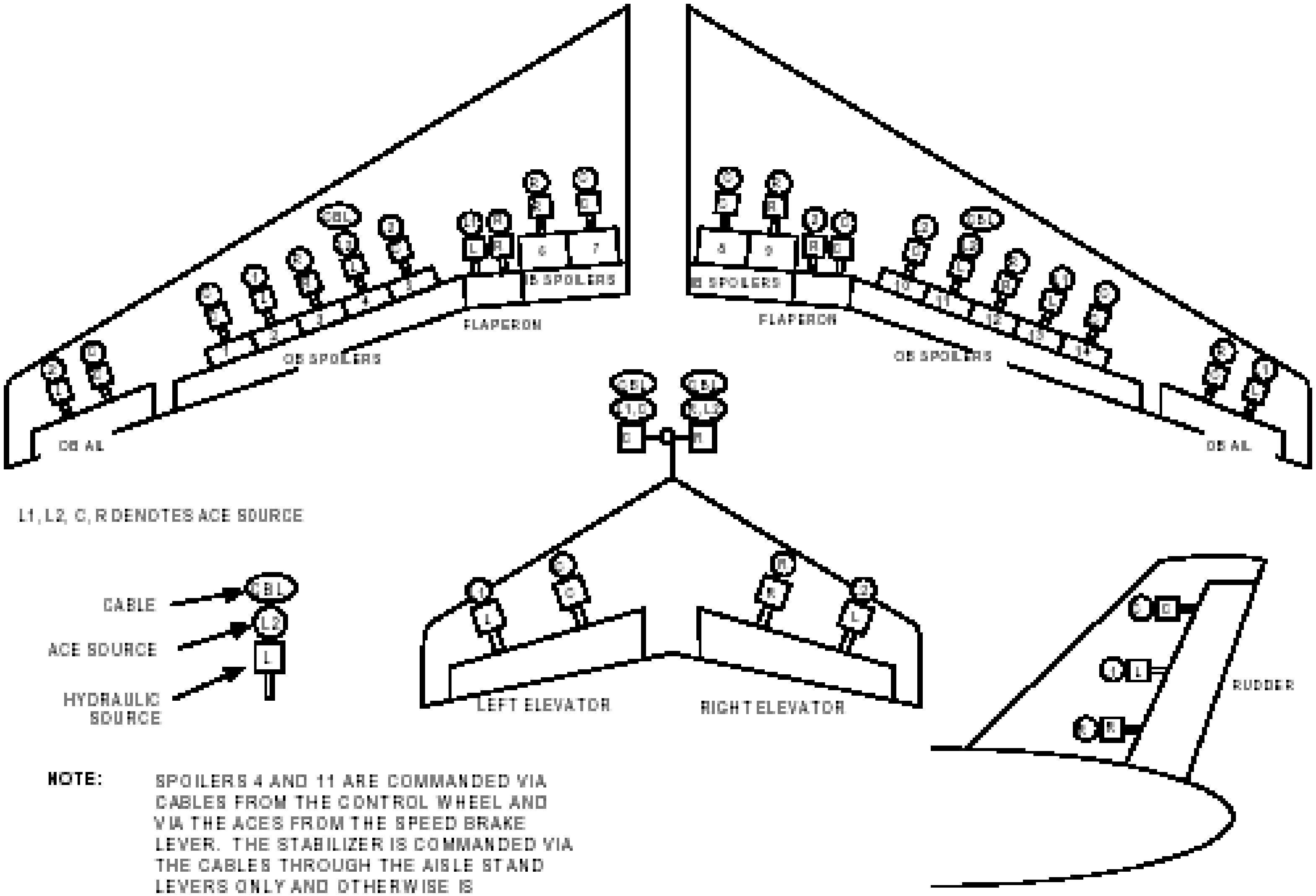
"each PFC (primary fight computer) channel contains three dissimilar processor lanes, and software from Ada source code using three different Ada compilers to provide triple dissimilarity"

B777 control architecture



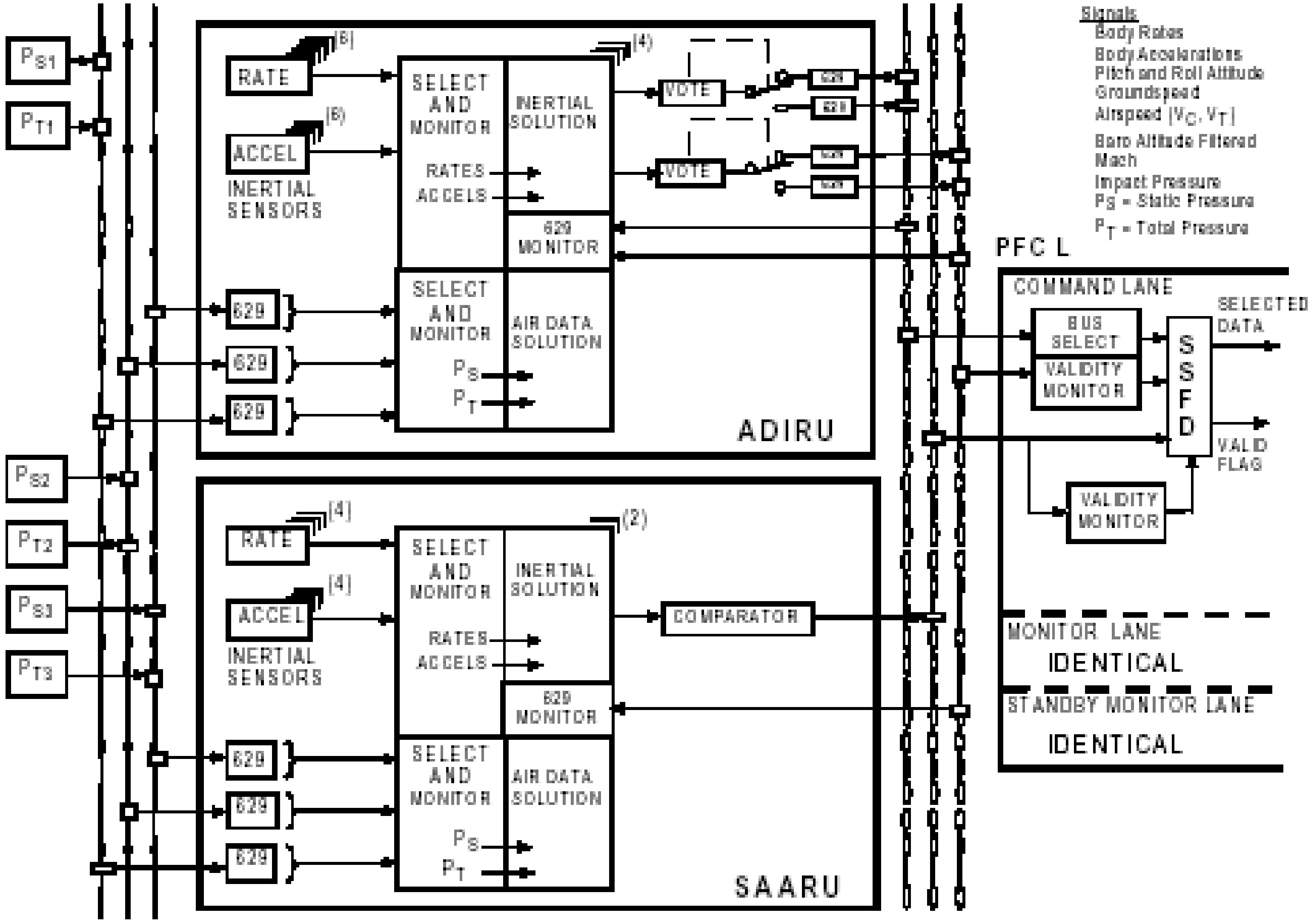
AFDC	AUTOPILOT FLIGHT DIRECTOR COMPUTER
ADM	AIR DATA MODULE (STATIC AND TOTAL PRESSURES)
EICAS	ENGINE INDICATION AND CREW ALERTING SYSTEM
ELMS	ELECTRICAL LOAD MANAGEMENT SYSTEM
MFD	MULTIPLE FUNCTION DISPLAY
PSA	POWER SUPPLY ASSEMBLY
FSEU	FLAP SLAT ELECTRONICS UNIT
AIMS	AIRPLANE INFORMATION MANAGEMENT SYSTEM
PFC	PRIMARY FLIGHT COMPUTER
PSEU	PROXIMITY SWITCH ELECTRONICS UNIT
R/A	RADIO ALTIMETER
ADIRU	AIR DATA INERTIAL REFERENCE UNIT
SAARU	SECONDARY ATTITUDE AND AIR DATA REFERENCE UNIT
ACE	ACTUATOR CONTROL ELECTRONICS
PCU	POWER CONTROL UNITS, ACTUATORS
HYDIM	HYDRAULIC INTERFACE MODULE
WOW	WEIGHT ON WHEELS
WES	WARNING ELECTRONICS SYSTEM

B777 control surfaces

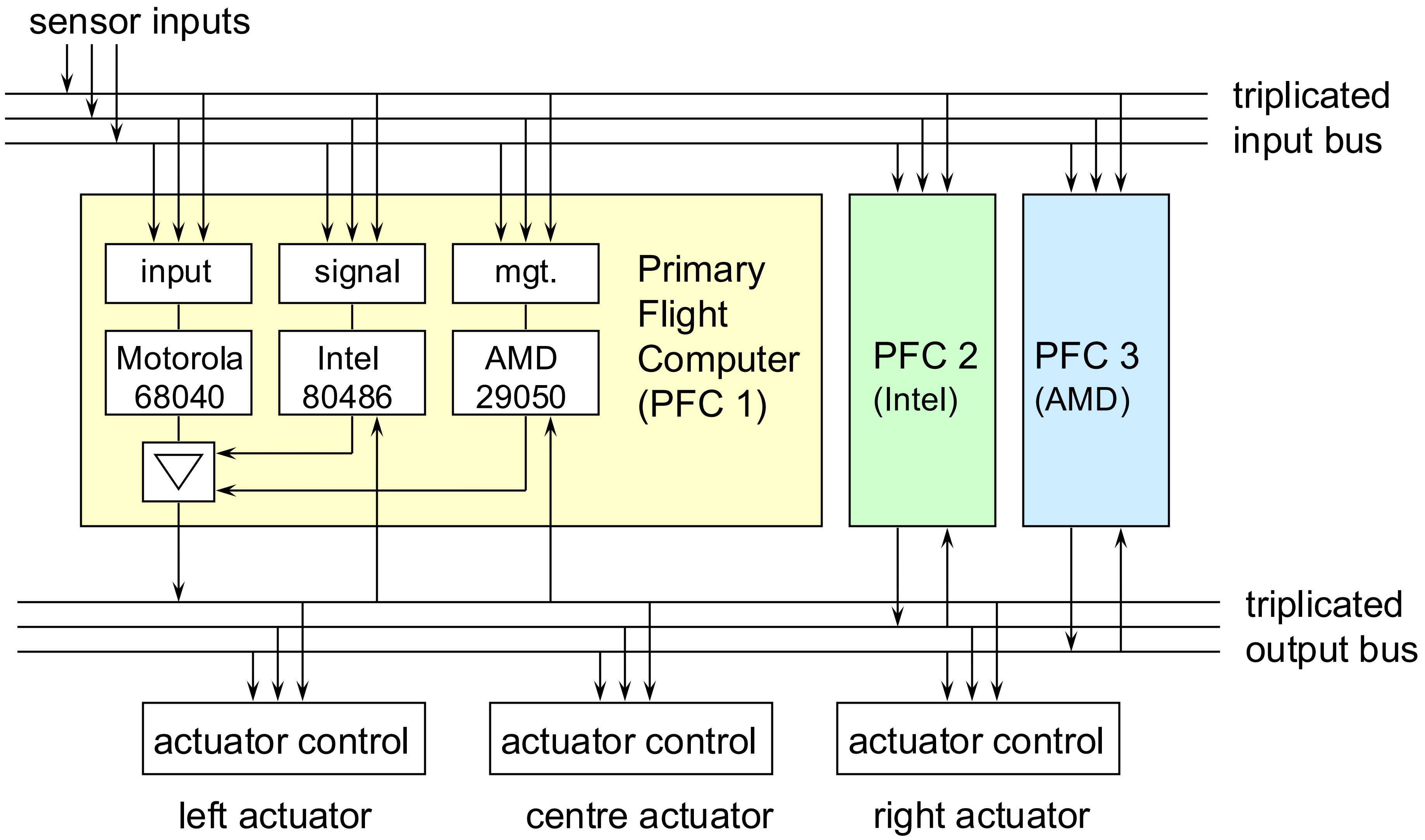


NOTE: SPOILERS 4 AND 11 ARE COMMANDED VIA CABLES FROM THE CONTROL WHEEL AND VIA THE ACES FROM THE SPEED BRAKE LEVER. THE STABILIZER IS COMMANDED VIA THE CABLES THROUGH THE AISLE STAND LEVERS ONLY AND OTHERWISE IS COMMANDED THROUGH THE ACES.

B777 Modules



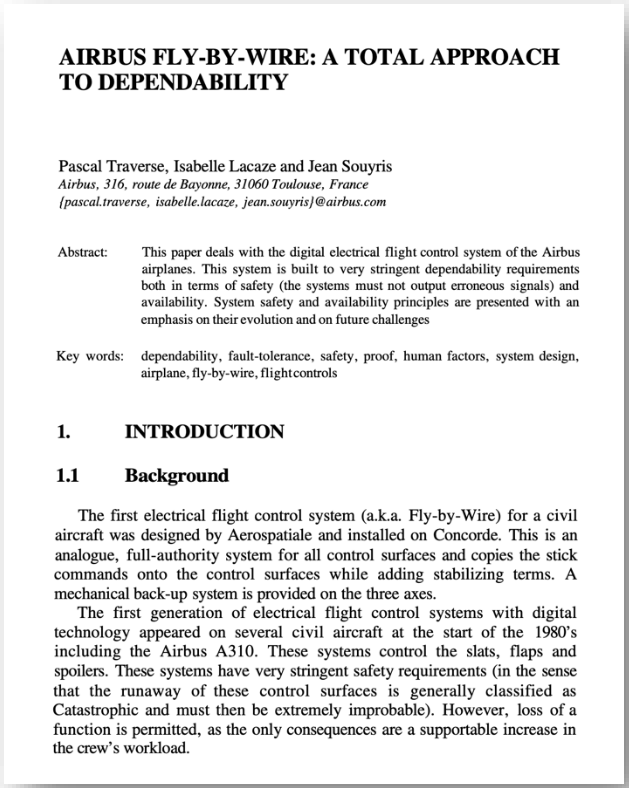
B777 Primary Flight Control: example of diverse programming



Airbus

- Airbus Fly-byWire: a total approach to dependability
  - https://link.springer.com/content/pdf/10.1007%252F978-1-4020-8157-6_18.pdf

- Airbus flight control system
  - https://ifs.host.cs.st-andrews.ac.uk/Resources/CaseStudies/Airbus/Airbus-fcs.pdf
  - 1oo5 flight control computers redundancy
  - Primary (3 units) and secondary (2 units) computers
    - Use different computers
    - Designed and supplied by different companies
    - Processor chips from different manufacturers



AIRBUS FLY-BY-WIRE: A TOTAL APPROACH TO DEPENDABILITY

Pascal Traverse, Isabelle Lacaze and Jean Souyris
Airbus, 316, route de Bayonne, 31060 Toulouse, France
{pascal.traverse, isabelle.lacaze, jean.souyris}@airbus.com

Abstract: This paper deals with the digital electrical flight control system of the Airbus airplanes. This system is built to very stringent dependability requirements both in terms of safety (the systems must not output erroneous signals) and availability. System safety and availability principles are presented with an emphasis on their evolution and on future challenges

Key words: dependability, fault-tolerance, safety, proof, human factors, system design, airplane, fly-by-wire, flight controls

1. INTRODUCTION

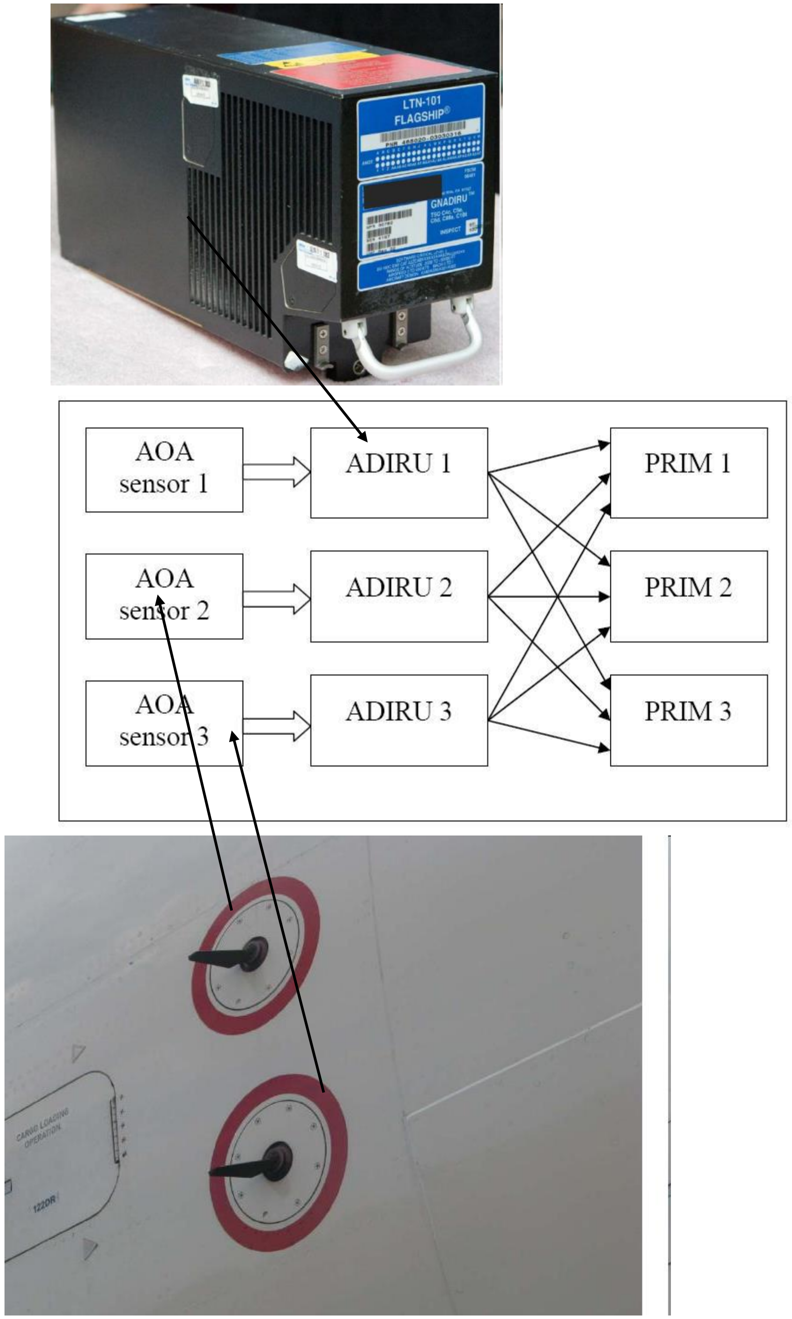
1.1 Background

The first electrical flight control system (a.k.a. Fly-by-Wire) for a civil aircraft was designed by Aerospatiale and installed on Concorde. This is an analogue, full-authority system for all control surfaces and copies the stick commands onto the control surfaces while adding stabilizing terms. A mechanical back-up system is provided on the three axes.

The first generation of electrical flight control systems with digital technology appeared on several civil aircraft at the start of the 1980's including the Airbus A310. These systems control the slats, flaps and spoilers. These systems have very stringent safety requirements (in the sense that the runaway of these control surfaces is generally classified as Catastrophic and must then be extremely improbable). However, loss of a function is permitted, as the only consequences are a supportable increase in the crew's workload.

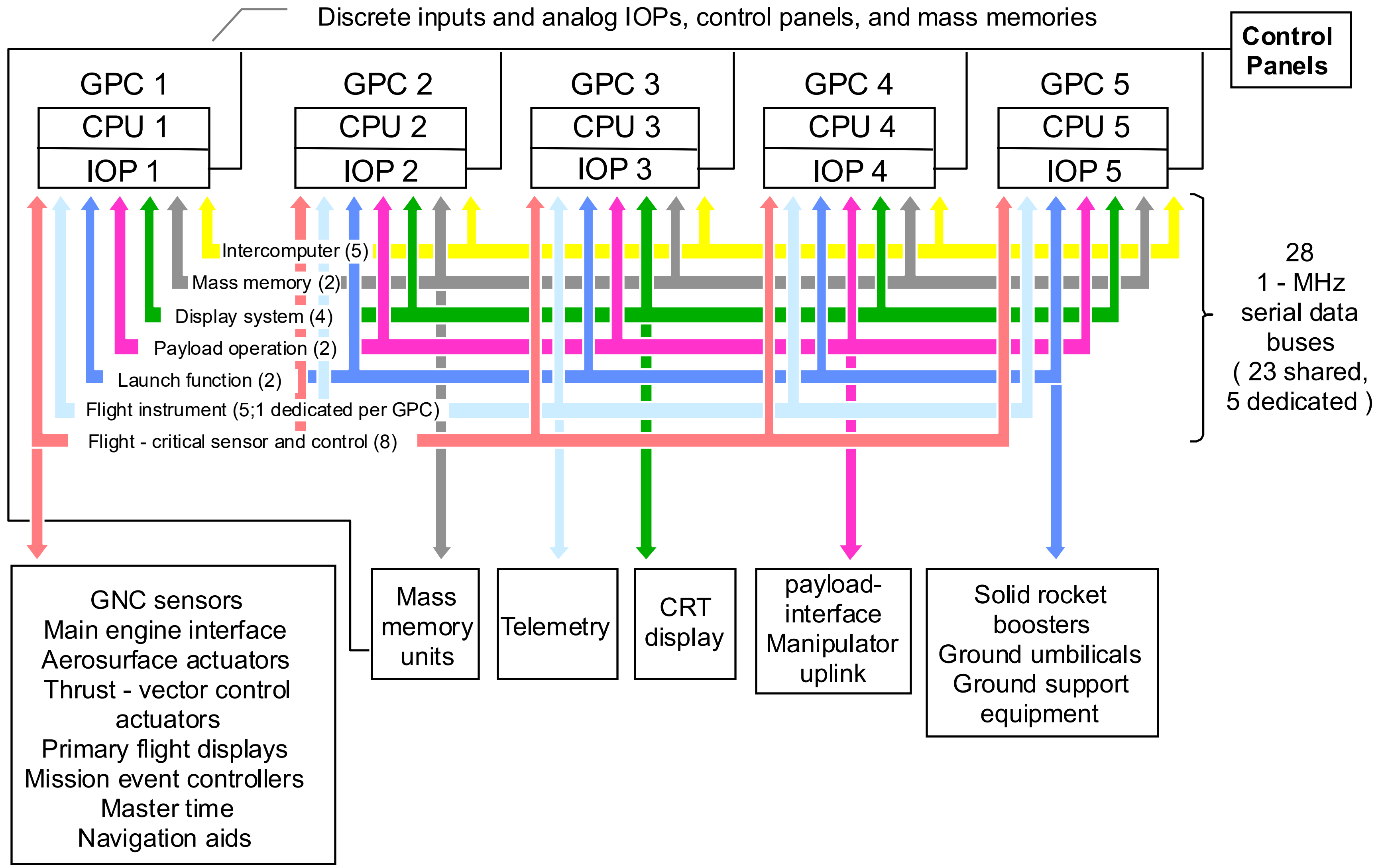
Airbus 330

1) A flight computer (ADIRU) that does not disengage in case of malfunction will poison the remaining good units ! ⇨ fail silent did not work
2) In case of sensor problems, no consensus can be built. all units could disengage !



Quantas airbus after ADIRU failure

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.