Concepts and Mechanisms of Dependable Systems

Summer Term 2011

References and Readings:

Paulo Veríssimo, Luís Rodrigues:
Distributed Systems for System Architects
Kluwer Academic Publishers, Boston, January 2001

Eugen Schäfer: **"Zuverlässigkeit, Verfügbarkeit und Sicherheit in der Elektronik, Eine Brücke von der Zuverlässigkeittheorie zu den Aufgaben der Zuverlässigkeits-praxis"**, 1. Auflage, Vogel Verlag, 1979, ISBN 3-0823-0586-8,

Stefan Poledna: "**Lecture on Fault-Tolerant Systems**", Vorlesungsfolien, Institut für Technische Informatik, TU Wien, SoSe 1996

Dependability

The dependability of a system is its ability to deliver specified services to the end users so that they can justifiably rely on and trust the services provided by the system.

The function or service is the behaviour which can be observed at the interface to other systems which interact with the observed system. Quality referes to the conformance to the specifcations.

Algirdas Avižienis, Jean-Claude Laprie, Brian Randell

Fundamental Concepts of Dependability (2001)

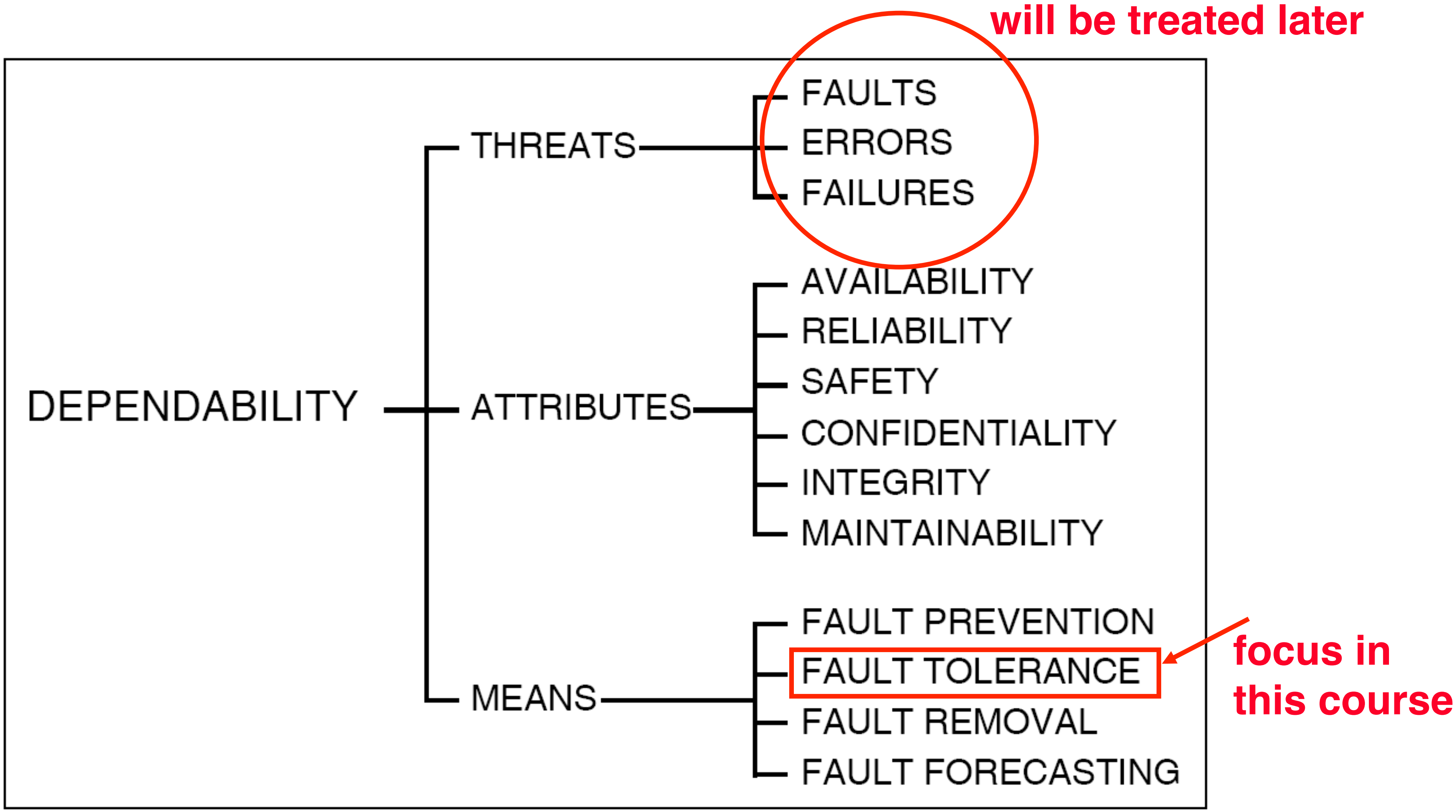
UCLA CSD Report no. 010028
LAAS Report no. 01-145
Newcastle University Report no. CS-TR-739

Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. 2004. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Trans. Dependable Secur. Comput.* 1, 1 (January 2004), 11-33.

Security

Security is defined as the absence of unauthorized access to, or handling of, system state. This includes multiple aspects as unauthorized disclosure of information (confidentiality), unauthorized change of information (integrity) and stopping or slowing down authorized access to information (availability). The fault model for security particularly copes with faults originating from intended malicious attacks to the system.

Dependability Tree



Attributes of Dependability

Dependability has several attributes, including reliability, availability, maintainability, security (with aspects like privacy, confidentiality and integrity) and safety.

Reliability: Reliability of a system for a period (0,t) is the probability that the system is continuously operational (i.e., does not fail) in time interval (0,t) given that it is operational at time 0.

Availability: Availability of a system for a period (0,t) is the probability that the system is available for use at any random time in (0,t).

Safety: Safety of a system for a period (0,t) is the probability that the system will not incur any catastrophic failures in time interval (0,t).

Maintainability: Maintainability of a system is a measure of the ability of the system to undergo maintenance or to return to normal operation after a failure.

Mechanisms of Fault-Tolerance

(Explizit) Fault-Treatment	Fault-Masking
Dynamic Redundancy	Static Redundancy

(Explizit) Fault-Treatment — Dynamic Redundancy

Error Detection
↓
Fault-Treatment: Damage Assessment and Confinement
↓
Fault-Treatment: Reconfiguration
↓
Recovery

Fault-Masking — Static Redundancy

e.g. Error Correcting Codes

n-out-of - m - Majority Voting

All Mechanisms of Fault-Tolerance are based on Redundancy

- Information Redundancy
- Component Redundancy
- Time Redundancy

Impairments:

Faults, errors, failures

The Cause-Effect-Chain: Classifying Impairments

inherently unavoidable

Fault

failure of a physical component or a faulty statement in a program.

Methods of *fault avoidance*

this must be treated! a faulty state must be recovered to a correct state.

a fault may probably cause a erroneous change of the system state.

Error

faulty state, e.g. memory or register contents

Methods of *fault-tolerance*

an error may cause a change of the system behaviour

this cannot be tolerated because it becomes visible at the system's interface and may be propagated to other systems.

Failure

deviation from the specified behaviour.

cannot be handled by the system.

action from outside needed. May lead to a disaster in safety-critical apps.

The Cause-Effect-Chain: Classifying Impairments

$$\cdots \longrightarrow \text{fault} \xrightarrow{\textit{activation}} \text{error} \xrightarrow{\textit{propagation}} \text{failure} \xrightarrow{\textit{causation}} \text{fault} \longrightarrow \cdots$$ *

transitions:

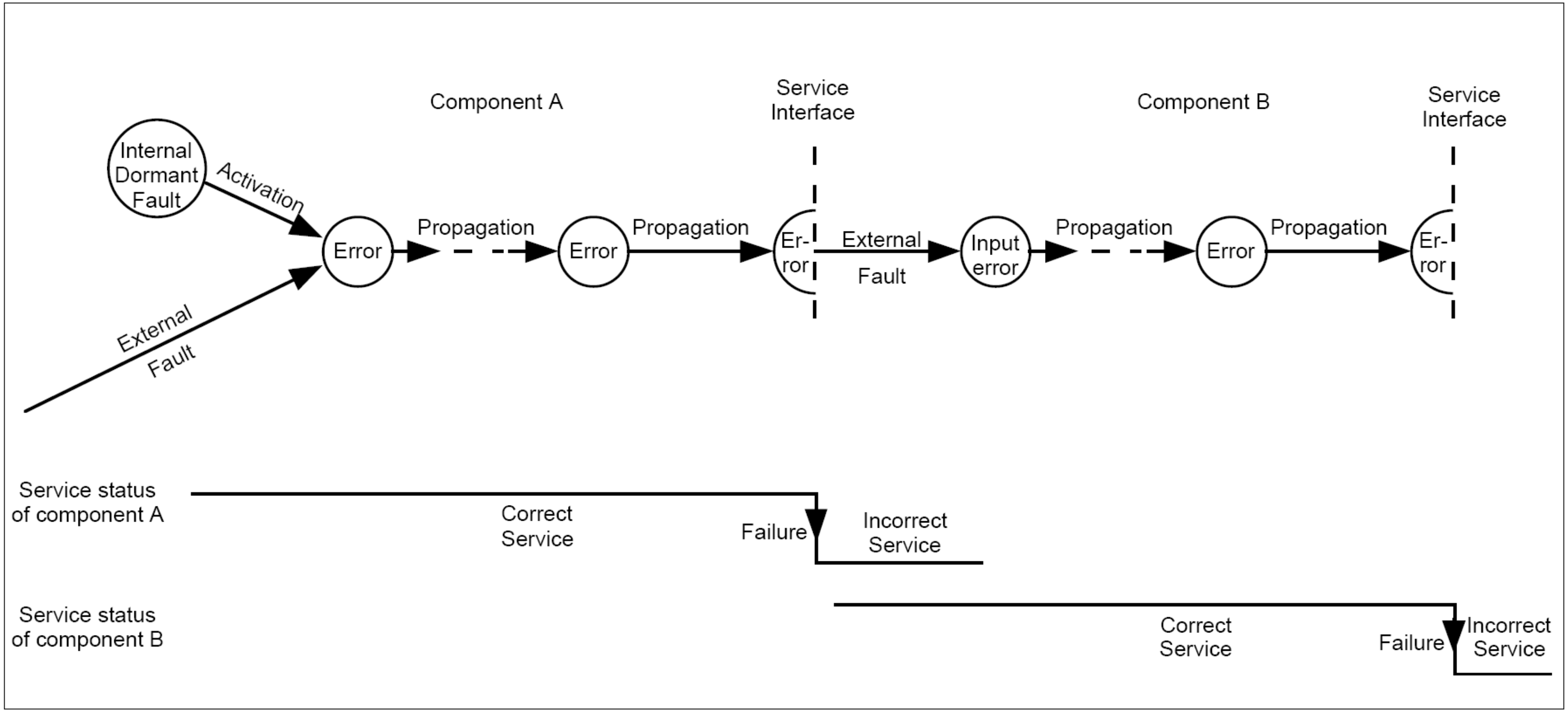
fault → error: **A fault which has not been activated by a computation is called *dormant*. A fault is *activated* if it causes an error.**

error→ failure: **An error is *latent* if it has not yet lead to a failure or has been detected by some error detection mechanism.**
An error is *effective* if it caused a failure.

failure→ fault: **A fault is caused if the error becomes effective and the specified service is affected. This failure can be propagated and appears as a fault on a higher system layer or in a connected component.**

* Algirdas Avižienis, Jean-Claude Laprie, Brian Randell: Fundamental Concepts of Dependability

The Cause-Effect-Chain: Classifying Impairments



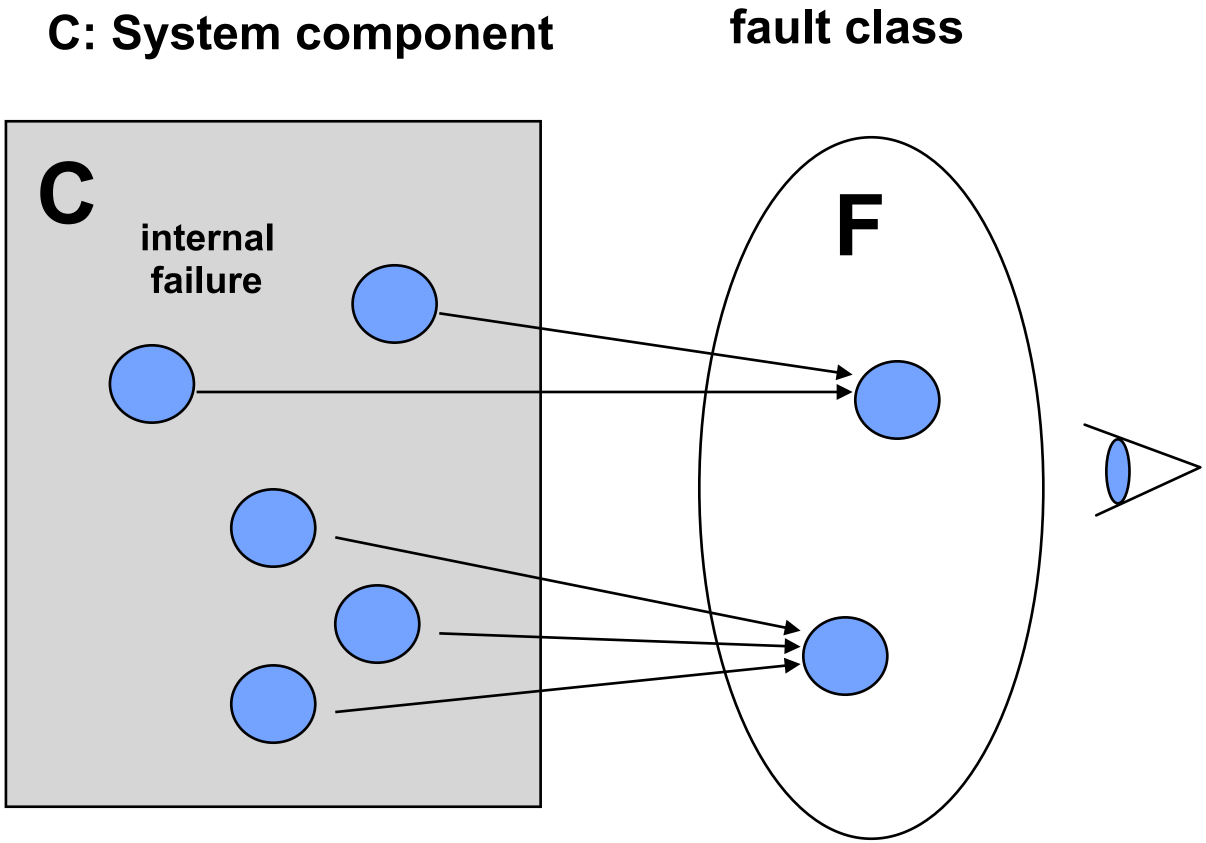
*

Error Propagation

* Algirdas Avižienis, Jean-Claude Laprie, Brian Randell: Fundamental Concepts of Dependability

Abstracting Failures: Failure Semantics

The fault semantics describes the assumptions about the effect of internal failures on the observable behaviour of a system component. It thus describes an abstraction of internal failures.



Problem:
The mechanisms to handle component failures are related to the assumed fault class.

It has to be guaranteed that the fault class F is enforced by the system, i.e. no failure inside the component may lead to a fault not covered by the failure semantics visible at the interface.

Examples:
Omission-Failure Semantics
Crash-Failure Semantics

S has the failure semantics of F

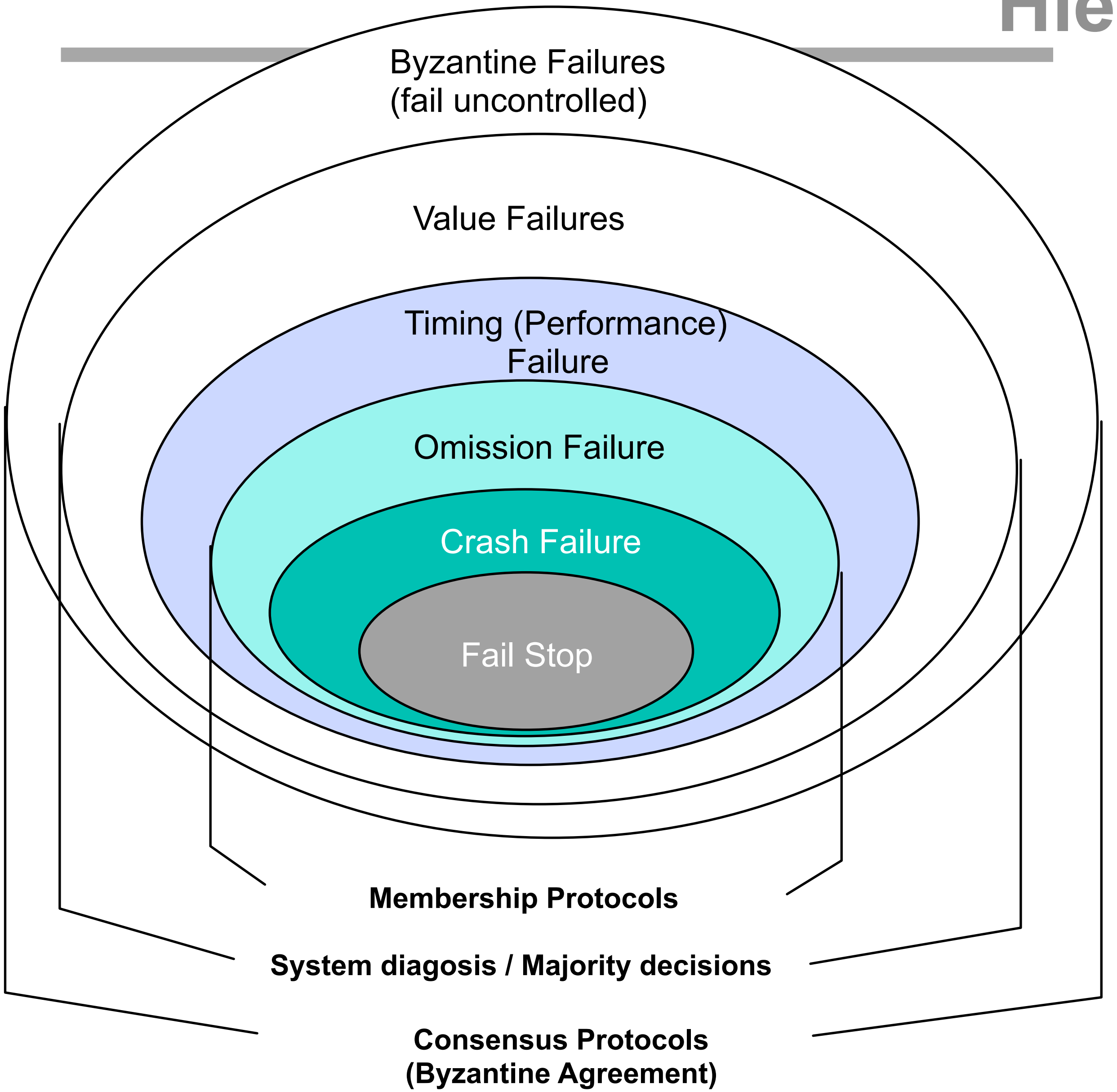
Hierarchy of failures in a networked system

System assumptions:
- many processes
- processes cooperate by sending and receiving messages

What may fail?
- processes may fail
- the network may fail

How it may fail?
- in the temporal domain
- in the value domain
- benign
- arbitrary (malicious)

Hierarchy of Failures



Byzantine Failure:
Arbitrary, uncontrolled.

Value Failures:
Corrupted value delivered to all nodes.

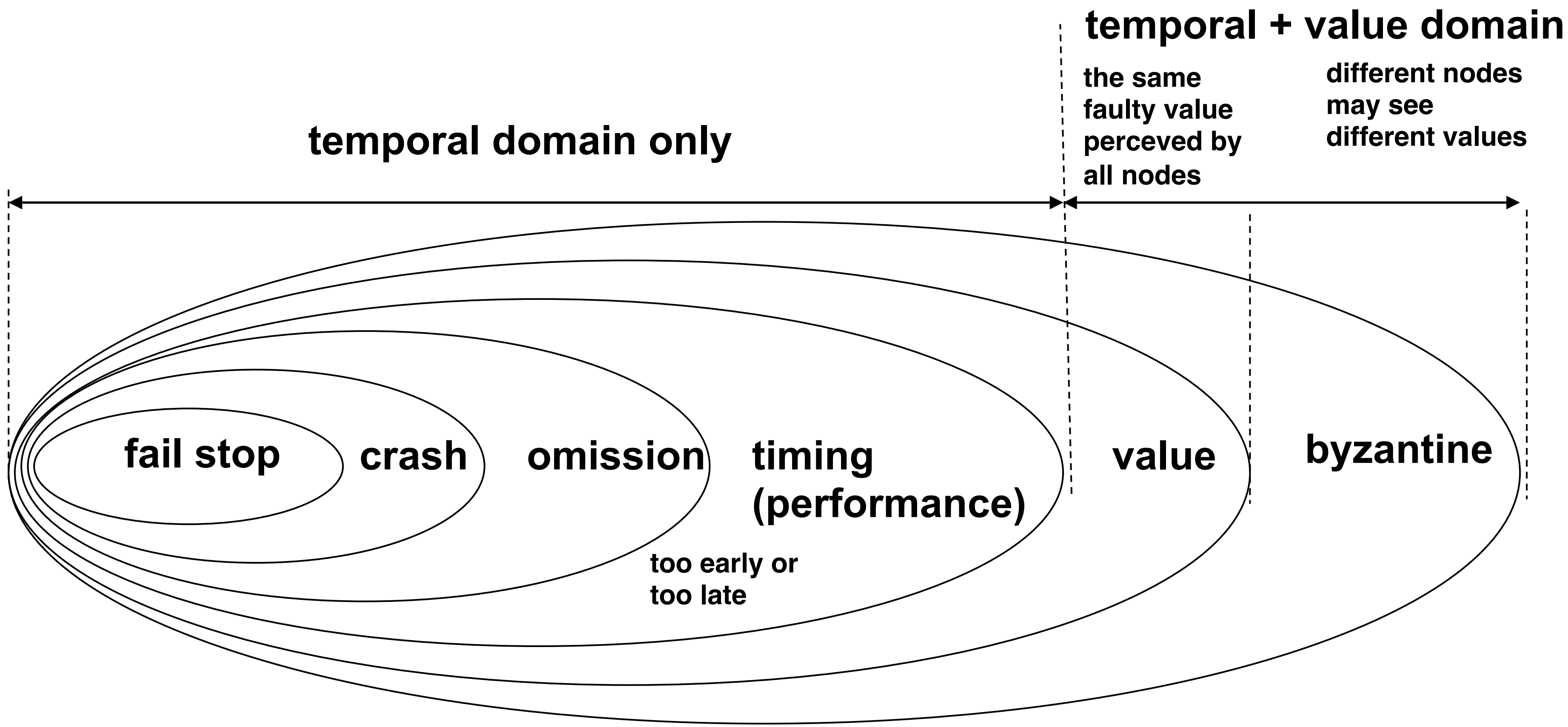
Timing (Prerformance) Failures:
Correct values but too early or too late.

Omission Failures:
Special class of timing failures. Correct values are available in time or not at all.

Crash Failures:
Component does not deliver any data.

Fail Stop:
Failed component stops to produce results.
Components are able to diagnose the Crash Failure correctly.

Fault Model and Failure Semantics



masking / mapping } **resend, time-out, duplicate msg. recognition and removal, check sum, replication, majority voting.**

Fault Model and Failure Semantics

Fault Class	affects:	description
fail stop	process	A process crashes and remains inactive. All all participants safely detect this state.
crash	process	A process crashes and remains inactive. Other processes may not detect this state.
omission - send om. - receive om.	channel channel channel	A message in the output message buffer of one process never reaches the input message buffer of the other process. A process completes the send but the respective message is never written in its send output buffer. A message is written in the input message buffer of a process but never processed.
byzantine	process	An arbitrary behaviour of a process.

Fault Model and Failure Semantics

Reliable 1-to-1 Communication:

Validity: every message which is sent (queued in the out-buffer of a correct process) will eventually be received (queued in the in-buffer of an correct process)

Integrity: the message received is identical with the message sent and no message is delivered more than once.

Validity and integrity are properties of a channel!

Fault Model and Failure Semantics

UDP provides a Channels with Omission Faults and doesn't guarantee any order.
TCP provides a Reliable FIFO-Ordered Point-to-Point Connection (Channel)

Mechanisms	Effect
sequence numbers assigned to packets	FIFO between sender and receiver. Allows to detect duplicates.
acknowledge of packets	Allows to detect missing packets on the sender side and initiates retransmission
Checksum for data segments	Allows detection of value failures.
Flow Control	Receiver sends expected "window size" characterizing the amount of data for future transmissions together with ack.

How to determine the reliability of a systems?

Structure-based modelling:

- **identifiable independent components**
- **every component has its individual fixed reliability**
- **the system is composed from multiple interconnected components**
- **the construction of the model is based on the connection structure**

Determining reliability quantitatively by reliability diagrams

Probability of a correctly working component:

For every part of the system we distinguish two states:

• <u>intact</u> (correctly working component)
• failed

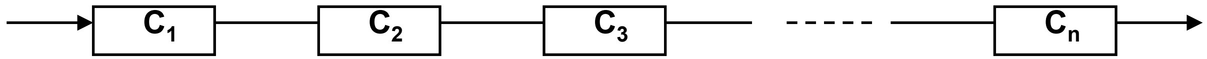
<u>C-Probability (probability of working correctly)</u> of a component is defined by:
Probability that the component exhibits the specified behaviour.

A system is <u>fault-tolerant</u>, if it is showing the overall specified behaviour while some components fail.

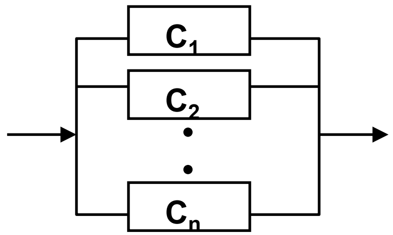
Reliability Diagrams (do not mix up with electrical schematics) :

Abstracting a system in components. Every component has a specified reliability.

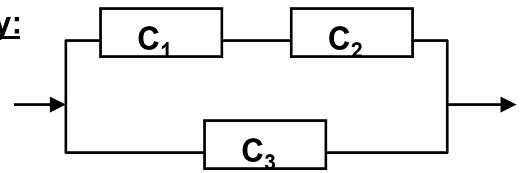
- **<u>serial dependency:</u>**



- **<u>parallel dependency:</u>**

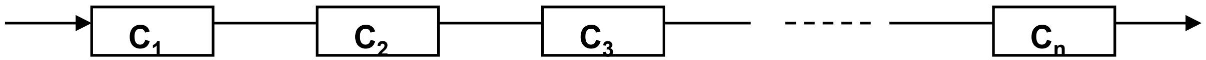


- **<u>serial/parallel dependency:</u>**



Probability for a correctly working system:

Serial dependencies



P_{series} = P (C_1 intact) and P(C_2 intact) and ....P(C_n intact)

Assumption: The properties (C_i intact) (i=1,..,n) are independent.

P_{series} = P (C_1 intact) • P(C_2 intact) • .... •P(C_n intact)

with p_i : probability of unfailed component (C-probability):

$P_{series} = p_1 \bullet p_2 \bullet \ ..... \ \bullet p_n$

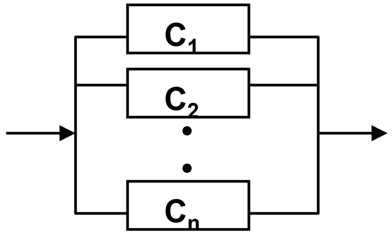
Examplel:

n identical Components:

P_{series} for p_i^n, n = 5, p_i = 0,99: P_{series} = $0{,}99^5$ = 0,95
P_{series} for p_i^n, n = 5, p_i = 0,70 : P_{series} = $0{,}70^5$ = 0,16

Probability for a correctly working system:

parallel dependencies



Probability of failure (F-probability) = 1 - C-probability

(correct and failed are complementary events).

$P_{parallel}$ = P (C_1 failed) and P(C_2 failed) and ....P(C_n failed)

Assumption: The properties (C_i failed) (i=1,..,n) are independent..

$P_{parallel}$ = P (C_1 failed) • P(C_2 failed) • .... •P(C_n failed)

p_i : F-probability of component i:

$P_{parallel} = 1 - (p_1 \bullet p_2 \bullet \ldots\ldots \bullet p_n)$

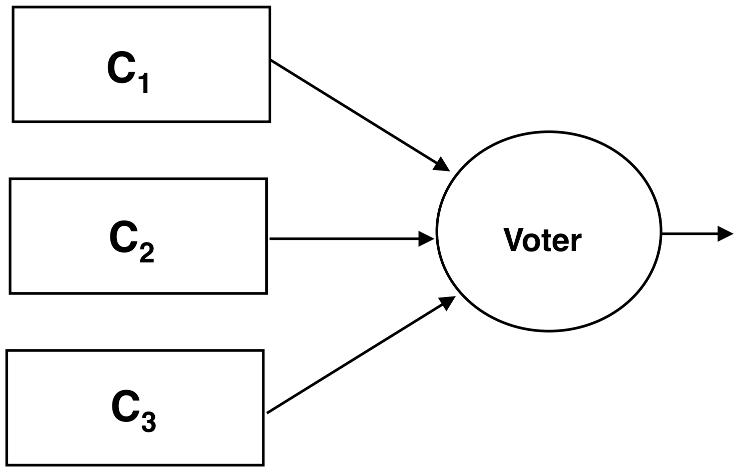
Example F-probability:

n identical Components:

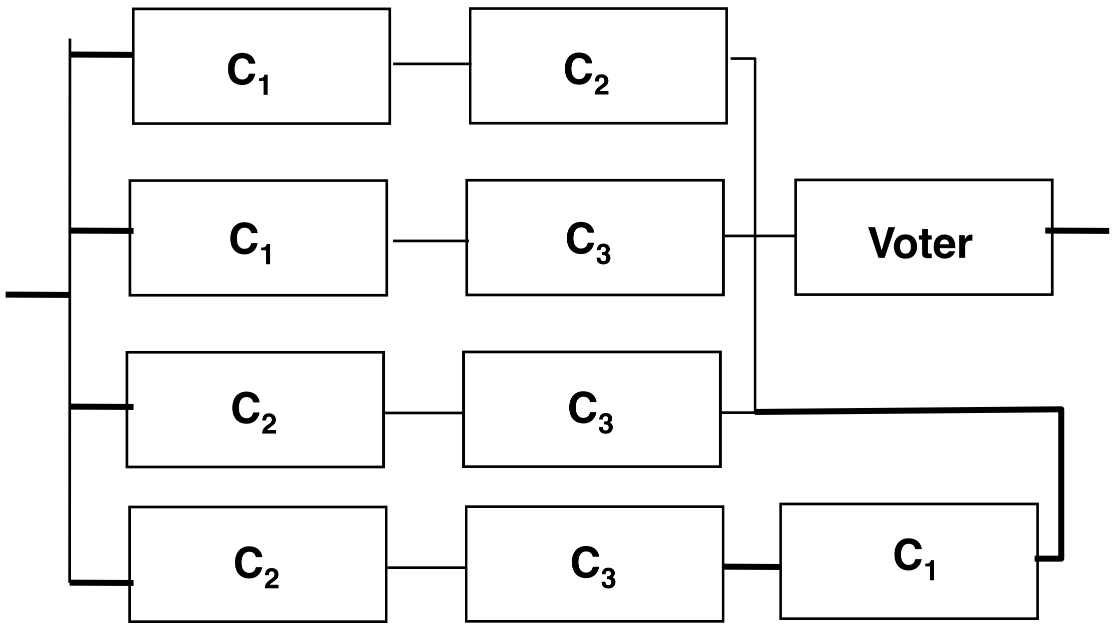
$P_{parallel}$ for p_i^n, n = 5, p_i = 1 - 0,99 : $P_{parallel} = 1 - 0{,}01^5$ = 1- 0,0000000001 = 0,9999999999
$P_{parallel}$ for p_i^n, n = 5, p_i = 1- 0,70 : $P_{parallel} = 1 - 0{,}30^5$ = 1 - 0,00243 = 0,99757

Example TMR (Triple Modular Redundancy: 2-out-of-3 system)

(electr.) block schematics



reliability diagram



$$P_{TMR} = (p^3 + 3\, p^2 \cdot (1 - p)\ ) \cdot p_{voter}$$

$p = 0{,}9$, $p_{voter} = 0{,}99$: $P_{TMR} = (0{,}9^3 + 3 \cdot 0{,}9^2 \cdot (1 - 0{,}9)) \cdot 0{,}99$

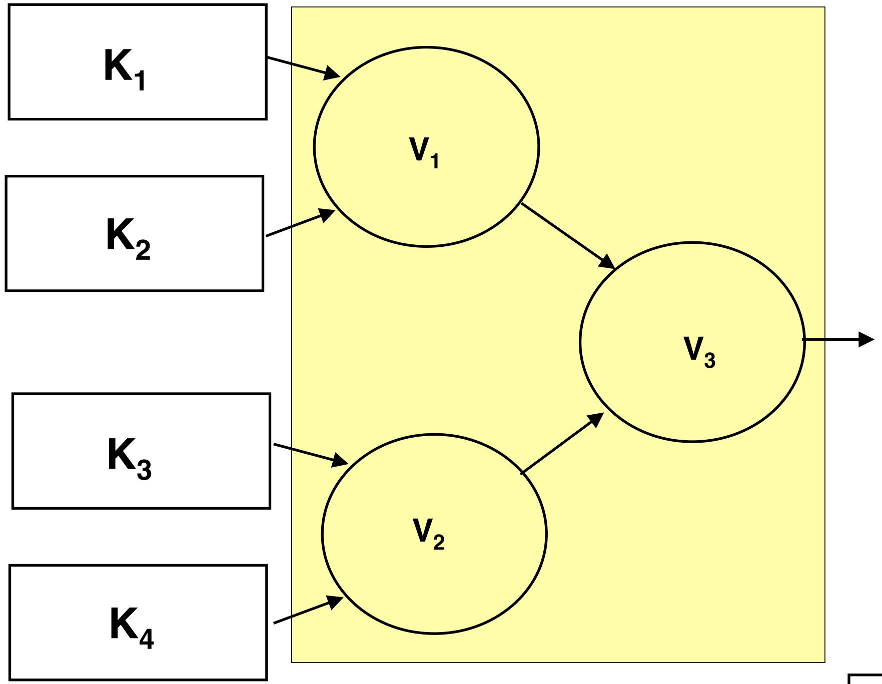
$$= (0{,}729 + 3 \cdot 0{,}81 \cdot (1 - 0{,}9)) \cdot 0{,}99$$

$$= (0{,}729 + 2{,}43 \cdot 0{,}1) \cdot 0{,}99 = 0{,}972 \cdot 0{,}99$$

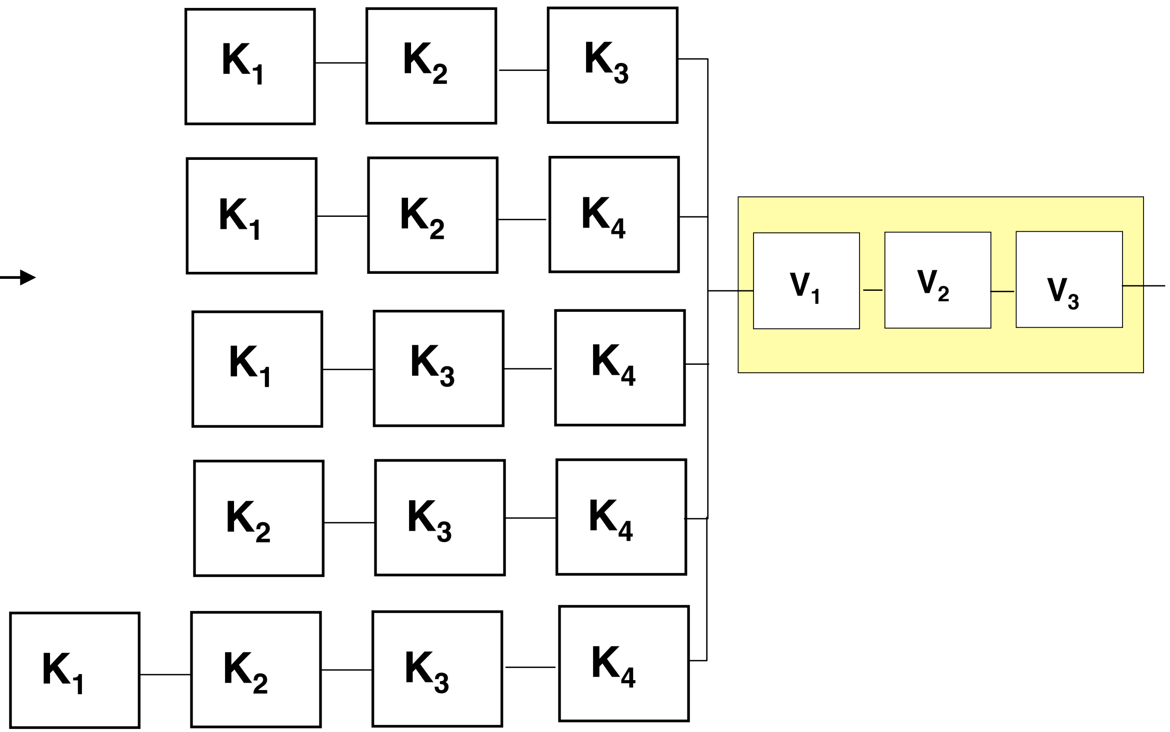
$$= 0{,}96228$$

Example Pair&Spare (3-out-of-4-System)

(electr.) block schematics



reliability diagram



Example Pair&Spare (3-out-of-4-System)

$P_{P\&S} = (p^4 + 4\ p^3 \cdot (1 - p)\ ) \cdot p_{voter}$

$p = 0{,}9$, $p_{voter} = 0{,}99$: $P_{P\&S} = (0{,}9^4 + 4 \cdot 0{,}9^3 \cdot (1 - 0{,}9)) \cdot 0{,}99$

$= (0{,}656 + 4 \cdot 0{,}73 \cdot (1 - 0{,}9)) \cdot 0{,}99$

$= (0{,}656 + 2{,}92 \cdot 0{,}1) \cdot 0{,}99 = 0{,}948 \cdot 0{,}99$

$= 0{,}9385$

$p = 0{,}9$, $p_{v1,2} = 0{,}99$, $p_{v3} = 0{,}999$:

$P_{P\&S} = (0{,}9^4 + 4 \cdot 0{,}9^3 \cdot (1 - 0{,}9)) \cdot 0{,}99^2 \cdot 0{,}999$

$= (0{,}656 + 4 \cdot 0{,}73 \cdot (1 - 0{,}9)) \cdot 0{,}979$

$= (0{,}656 + 2{,}92 \cdot 0{,}1) \cdot 0{,}99 = 0{,}948 \cdot 0{,}9879$

$= 0{,}928$

k-out-of-n - systems

Systems of n components in which at least k components are working correctly.

Probability that exactly k defined components are correct (components 1,..,k), while the other n-k components failed (componenten k+1,...,n) is given by:

$$P_{k\text{-aus-n}} = p_1 \cdot p_2 \cdot \ldots \cdot p_k \cdot (1 - p_{k+1}) \cdot (1 - p_{i+2}) \cdot \ldots \cdot (1 - p_n)$$

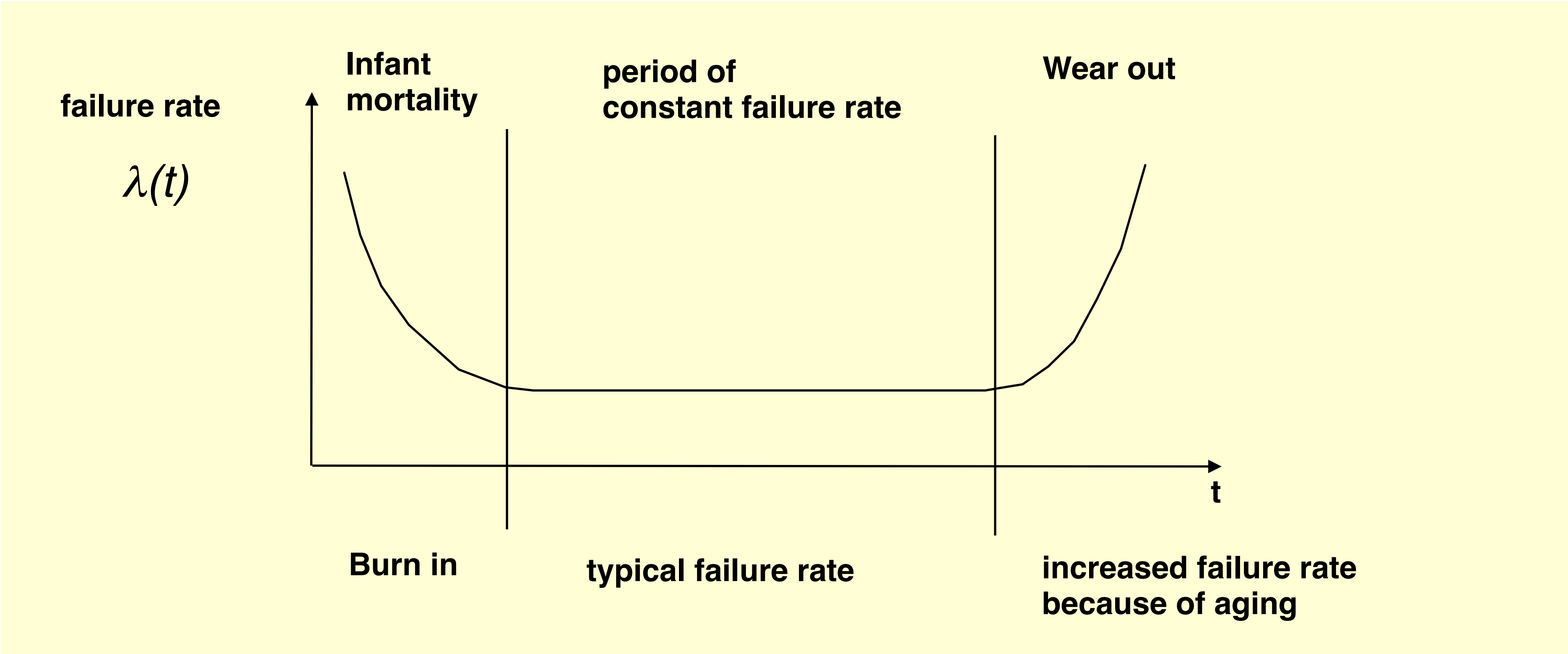
There are $\binom{n}{i}$ possibilities, to select i components out of n components:

$$P_{k\text{-out-of-n}} = \sum_{i=k}^{n} \binom{n}{i} p^i \cdot (1 - p)^{n-i}$$

Example: 2-out-of-3 System: $\binom{3}{2} p^2 \cdot (1 - p)^{3-2} + \binom{3}{3} p^3 \cdot (1 - p)^{3-3} = 3 \cdot p^2 \cdot (1 - p) + p^3 \cdot 1$

How to derive the probability of component failure ?

The "bath tub" curve



Typical failure rates:
VLSI-Chip: 10^{-8} failures/h = 1 failure during 115000 years

Note:

The failure rate is defined relative to the number of correct components. In a certain time interval, if always the same number of components fail, the failure rate increases relatitively to the number of correct components that becomes smaller by every failed component.

Dependability measures

Lifetime T
Time interval from the mission start to a non-repairable failure

Failure Rate λ ***(t)***
number of failures per time unit

Probability of failure F(t)
probability to fail in the interval [0,T], T < t_i .

Reliability R(t)
Probability that a component did not fail until time t_i.
F(t) is the complement to R(t).

$$R(t) = 1 - F(t)$$

for non repairable systems R(t) is a monotonely decreasing function. R(0) ≤ 1, R(∞) = 0

Probability density function f(t)
f(t) models how failures probabilities are distributed over time
f(t) • dt is the probability that a failure occurs in interval (t, t+dt))

$$f(t) = \frac{dF(t)}{dt} = -\frac{dR(t)}{dt}$$

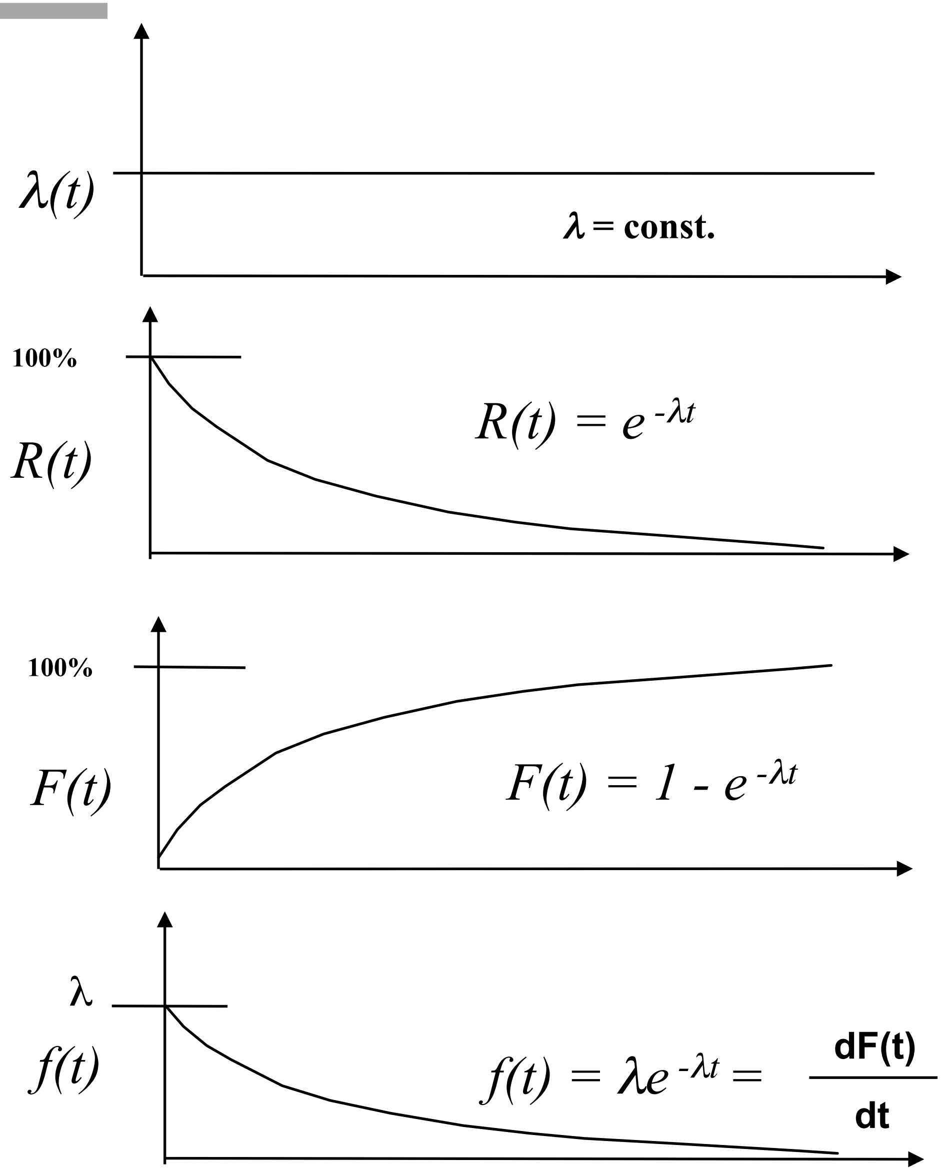
Dependability measures

failure rate λ (t)

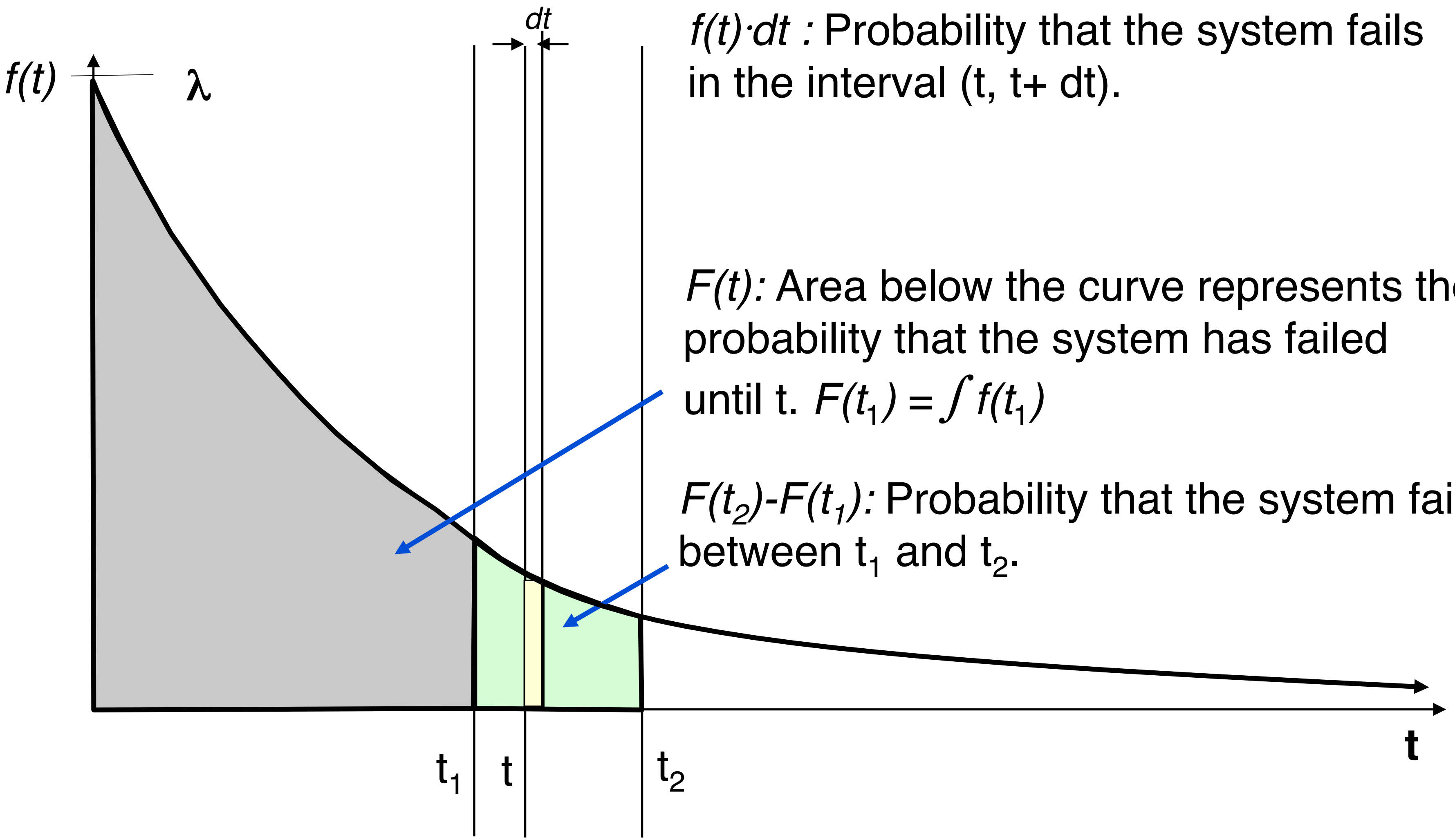
number of failures per hour

Remember: The failure rate is defined relativly to the number of correct components. In a certain time interval, if always the same number of components fail, the failure rate increases relatitively to the number of correct components that becomes smaller by every failed component.

If the failure rate remains constant wrt. the set of correct components, this results in an exponential distribution for the reliability *R(t)*.



Life time modelling



f(t)·dt : Probability that the system fails in the interval (t, t+ dt).

F(t): Area below the curve represents the probability that the system has failed until t. $F(t_1) = \int f(t_1)$

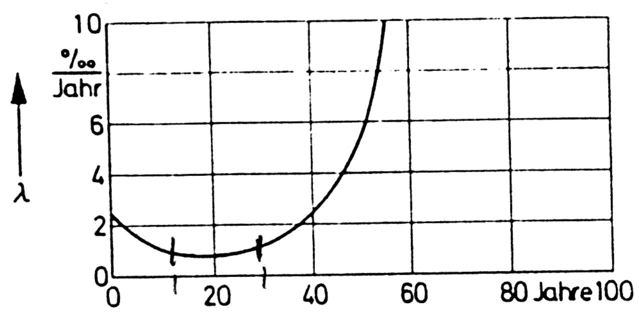
$F(t_2)$-$F(t_1)$: Probability that the system fails between t_1 and t_2.

f(t): PDF: Probability Density Function

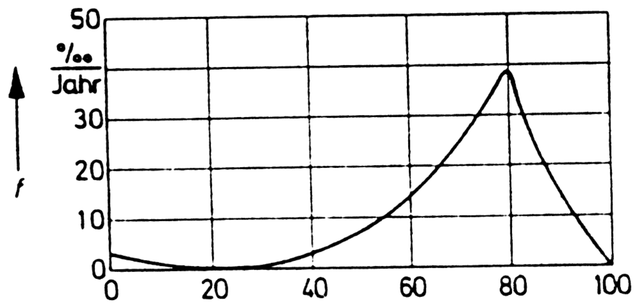
F(t): CDF: Cumulative Density Function. For $t_{\rightarrow\infty}$: $F(t) = \int f(t) = 1$

Probability distribution for human life

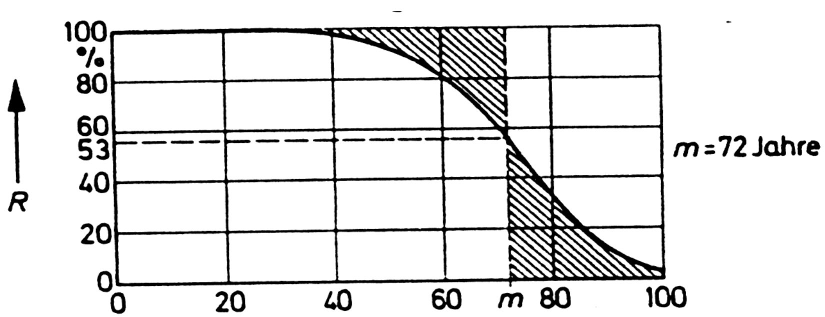
failure rate λ (t)



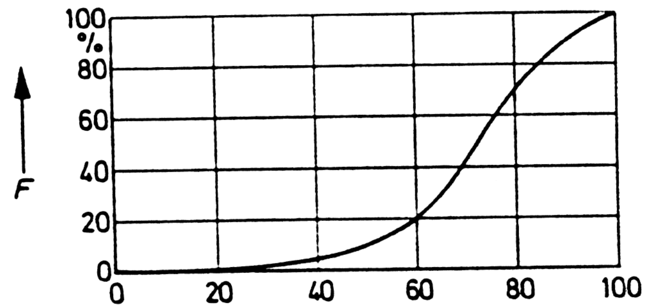
probability density f(t)



Reliability R(t)



failure probablity F(t)



Summary of Measures

Parameter	Symbol	Unit
life time	T	h
failure probability	F	%
reliability	R	%
probability density	f	%/h
failure rate	λ	1/h

Dependability measures

Assuming λ (t) = const. we have:

$$\frac{1}{\lambda} = \text{MTBF} = \text{MTTFF} = \text{MTTF}$$

MTBF : Mean Time Between Failures

MTTFF: Mean Time To First Failure

MTTF : Mean Time To Failure

Approximation of dependability measures

$$A = \frac{\text{U (Up time)}}{\text{M (Mission time)}}$$

M = U + TR (Repair time)

$$A = \frac{\text{MTBF}}{\text{MTBF + MTTR}}$$

Availability

time in which the system works correct related to the (down-) time when it is repaired.

$$R = 1 - F = 1 - \frac{\text{M (Mission time)}}{\text{MTBF (>> M)}} \sim e^{-\lambda t}$$

Reliability

Dependability measures

Availability Classes

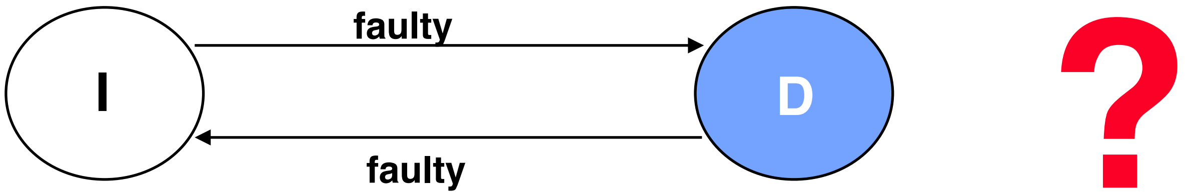
class: $\lfloor \log_{10} (1/(1-A)) \rfloor$

1 year = 525600 minutes = 8760 h

system type	non-availability minutes/year	availability %	class
non-adminitrated systems	50 000	~ 90	1
administrated systems	5 000	99	2
well admin. syst.	500	99,9	3
fault-tolerant syst.	50	99,99	4
high availability syst.	5	99,999	5
very high avail. syst.	0,5	99,9999	6
ultra-high avail. syst.	0,05	99,99999	7

Fault diagnosis in Distributed Systems

System diagnosis to detect and localize faults

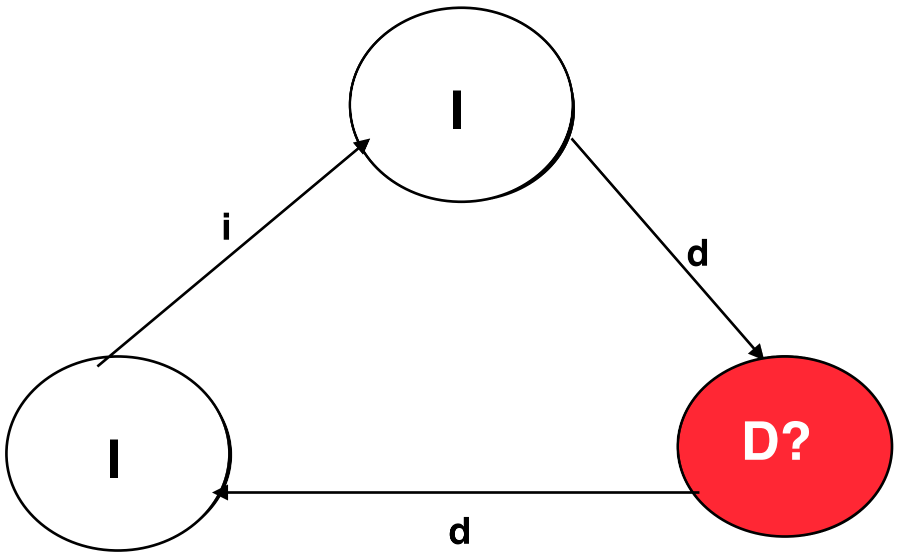


Assumptions:
- components are either faulty or correct.
- a test is complete and correct.
- a correct process wil deliver a correct result.
- a faulty process will deliver an arbitrary result.
- a central correct observer evaluates the result of the test.

F. P. Preparata, G. Metze, and R. T. Chien. On the connection assignment problem of diagnosable systems. IEEE Trans. Electron. Comput., EC--16:848--854, 1967

f – diagnosability

1-diagnosable system

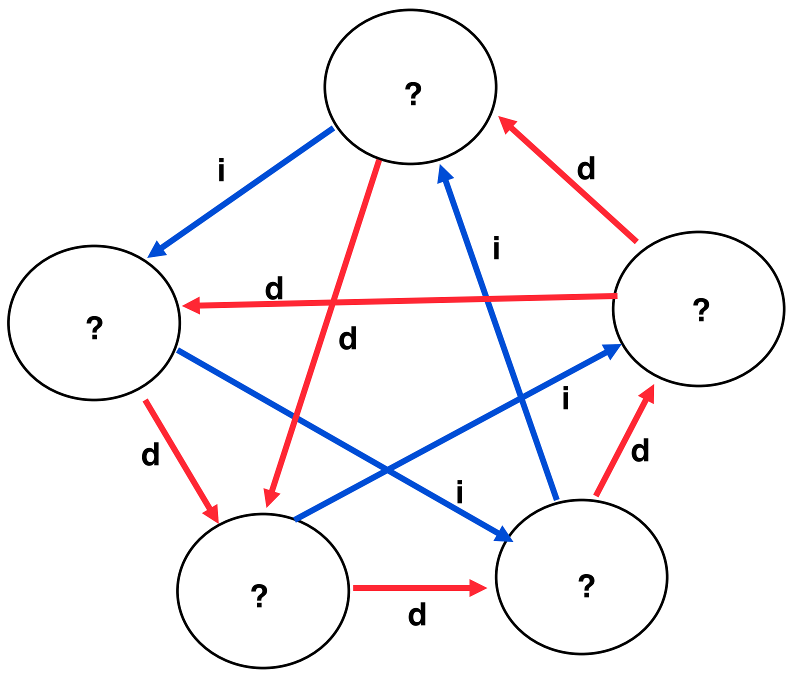


Assumptions:
- components are either faulty or correct.
- a test is complete and correct.
- a correct process wil deliver a correct result.
- a faulty process will deliver an arbitrary result.
- a node is marked as faulty if it has an incoming edge originating from a correct node, which has tested this node as faulty
- a central correct observer evaluates the result of the test.

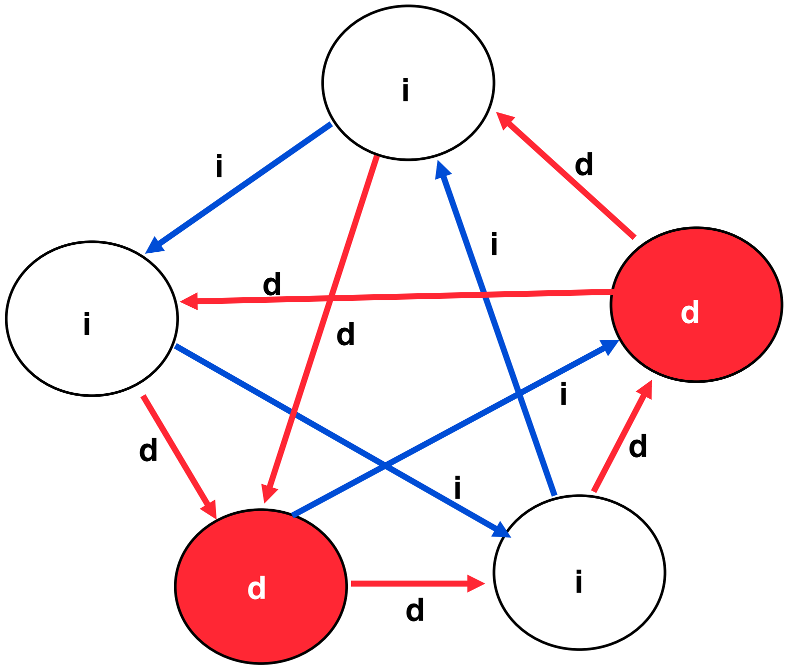
f – diagnosable :

A system with n components is f–diagnosable if
$n \geq 2f + 1$ and every component test at least f other components.
The components do not test each other.

Will diagnosis deliver an unambiguous result?



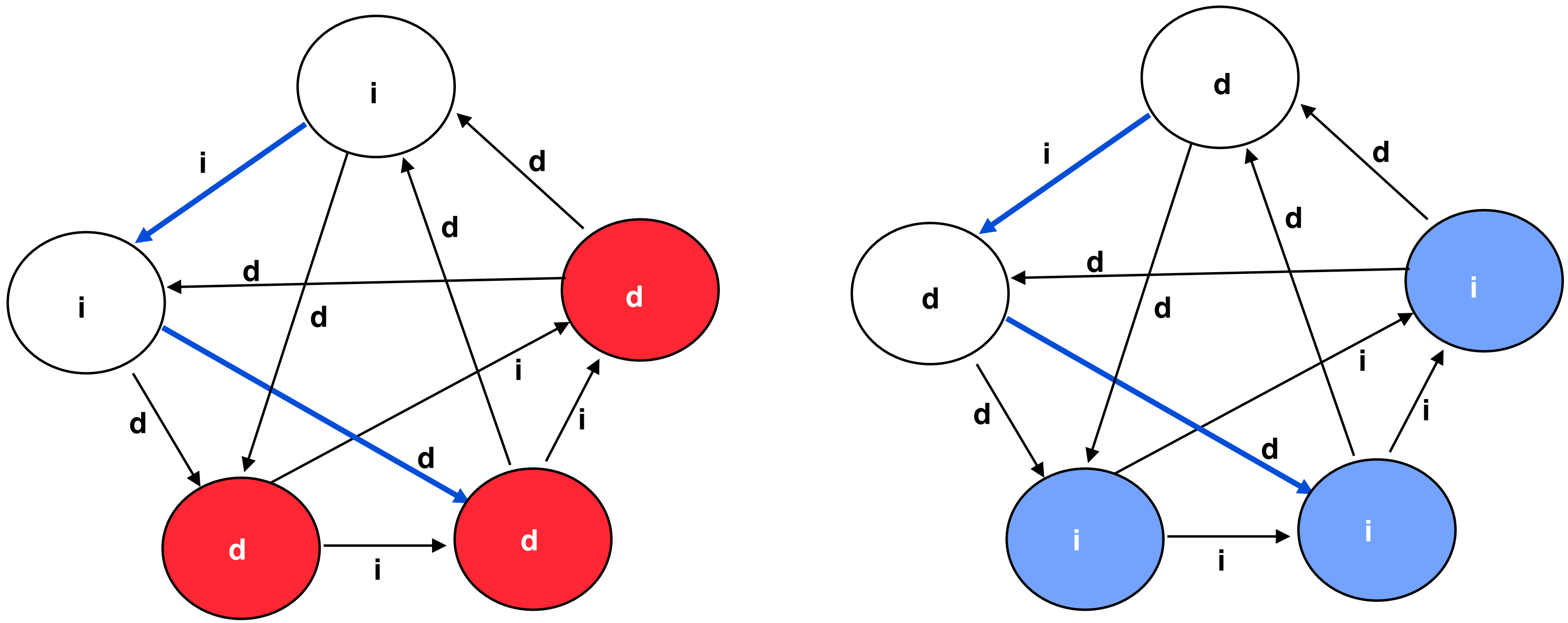
2-diagnosable system



Assumptions:

- components are either faulty or correct.
- a test is complete and correct.
- a correct process wil deliver a correct result.
- a faulty process will deliver an arbitrary result.
- a node is marked as faulty if it has an incoming edge originating from a correct node, which has tested this node as faulty
- a central correct observer evaluates the result of the test.

3 faulty nodes



fault cannot be detected (obviously) because the fault assumption (max. 2 faults) is violated.

Assumption:
Node is the unit of fault-containment and replacement!

Problems:

1. What kind of faults have to be considered?
➡ **Fault model.**

2. Can we replace the central evaluation component?
➡ **Distributed consensus.**

3. Can fault-detection always successfully be performed?
➡ **The problem of synchrony.**

The Network or the Node?

Fault-assumptions in Distributed Systems

Intuitive Consistency Criterion:

When a process fails, all correct processes are able to detect the failure and achieve consensus about the faulty process.

Formalisation by Chandra and Tueg (1996):

Strong Acuracy (SA): **No correct process ever is considered to be faulty. (safety criterion)**

Strong Completeness (SC): **A faulty process eventually will be detected by every correct process (liveness criterion).**

What are the conditions to achieve SA and SC?

Assumptions:

1. Transmission delays can be bounded.
2. Processes can generate and send a "heartbeat" message periodically in a bounded time interval.
3. We assume a crash failure model, i.e. the network is fault-free.

➡ **Heartbeat-mechanism is a perfect failure detector**

Assumptions:

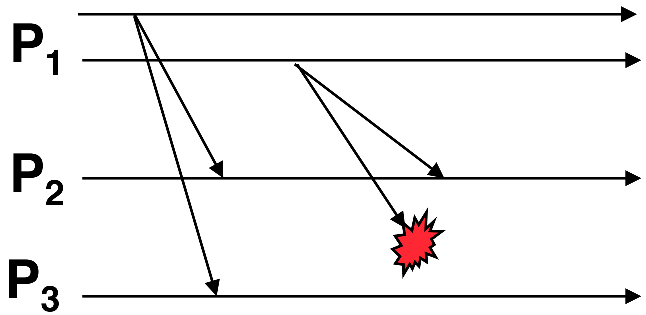
1. Transmission delays can be bounded.
2. Processes can generate and send a "heartbeat" message periodically in a bounded time interval.
3. We assume an omission failure model, however the omissions may be bounded.

➡ **Apply mechanisms to mask omissions.**

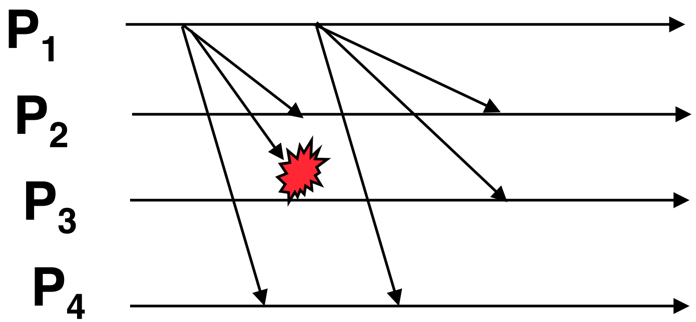
FT communication - Handling *message* failures

Static Redundancy: Masking Failures

component redundancy

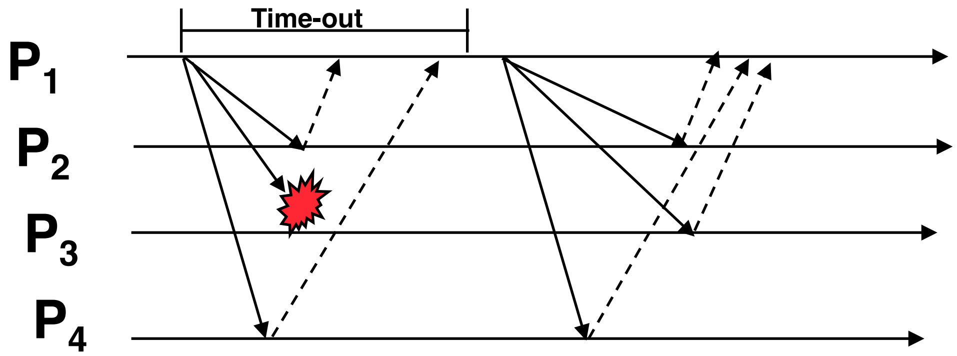


time redundancy



“forward” error recovery

Dynamic Redundancy: Detection + Recovery

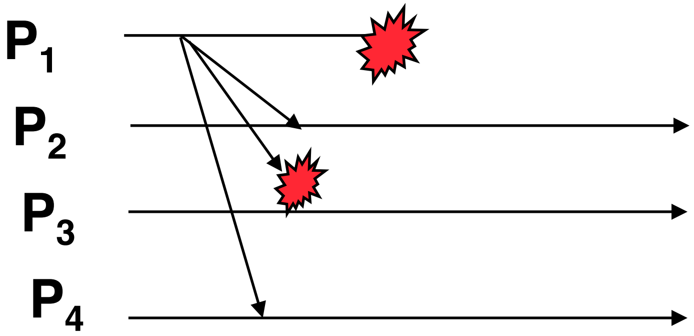


“backward” error recovery

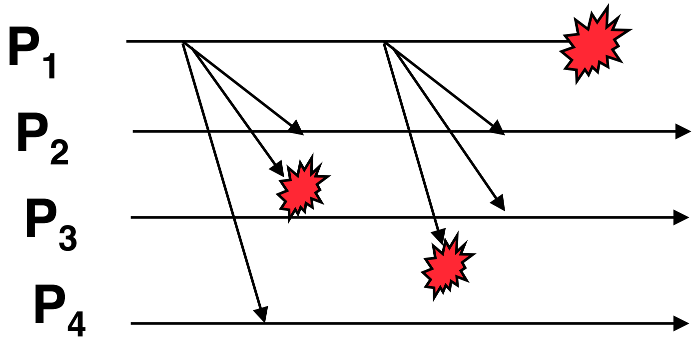
(requires add. ack!)

FT Communication - Handling *sender* failures

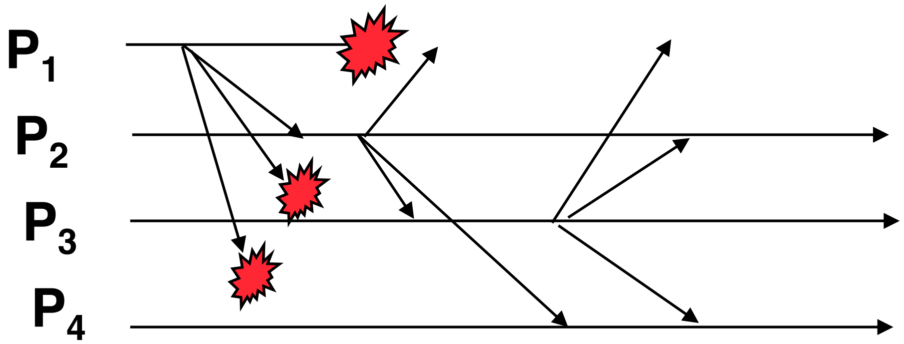
Unreliable Multicast



Best effort Multicast



Reliable Multicast



Imperfect failure detectors

Assumptions:

Temporal assumptions:

1. **the latency of messages cannot be bounded (asynchronous model),**
2. **processes cannot always produce a heartbeat in a bounded interval.**

Assmptions about the number of faults:

3. **The number of omissions cannot be bounded.**

➡ **No deterministic decision can be derived whether a process has failed or not.**

Consensus in Distributed Systems

Goal: A group of processes agree on a common value.
Every process proposes a value once.
Every process decides a value once.
Proposed and decided values are 0 or 1 (simplification).

The following conditions must be achieved:

Consistency: (Agreement) — All processes eventually agree on the same value and the decision is final.

Non Triviality: (Validity) — The decided value has been proposed by some process.

Termination: — Every correct process decides on the common value within a finite time interval.

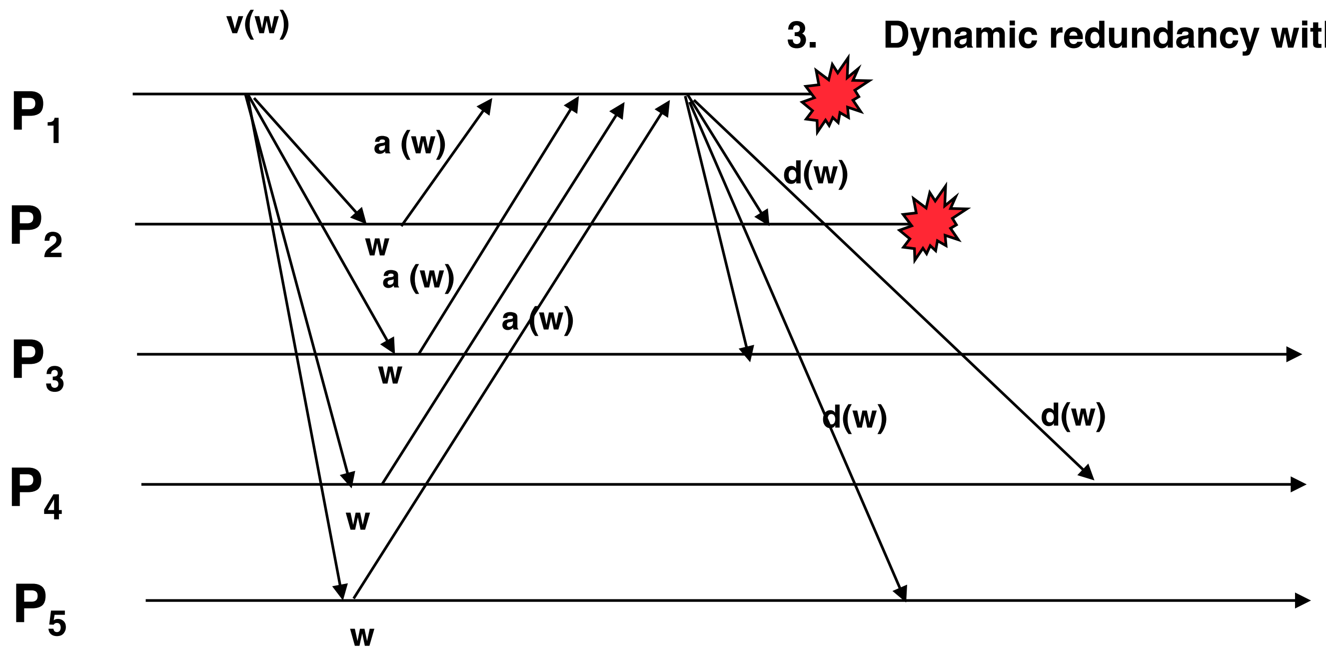
FLP Impossibility Result

Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. Journal of the ACM, 32(2):374{382, April 1985.

Fault-Tolerant Consensus

Assumptions:

1. The latency of messages is bounded.
2. Failure detection is reliable.
3. Dynamic redundancy with fault treatment.

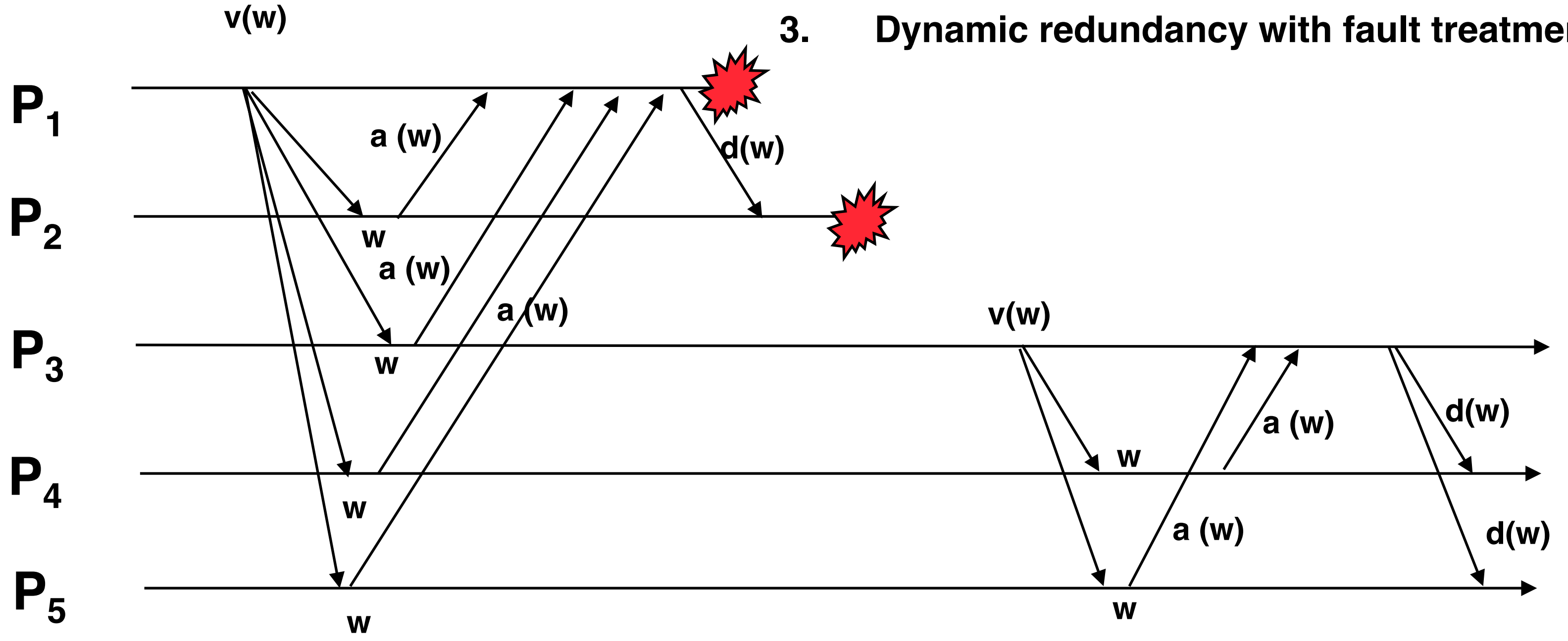


v(w): suggest(w)
a(w): accepted (w)
d(w): decided (w)

Fault-Tolerant Consensus

Assumptions:

1. The latency of messages is bounded.
2. Failure detection is reliable.
3. Dynamic redundancy with fault treatment.



v(w): suggest(w)
a(w): accepted (w)
d(w): decided (w)

Q

How much redundancy is needed to achieve consensus about the faulty nodes?

The results of Preparata, Metze & Chien say: 2f+1

- **But: Strong assumptions about testability**
- **Evaluation centralized! ➡ No consensus is needed.**

Is this majority also enough for distributed consensus?
Does the fault model influence the redundancy requirements?

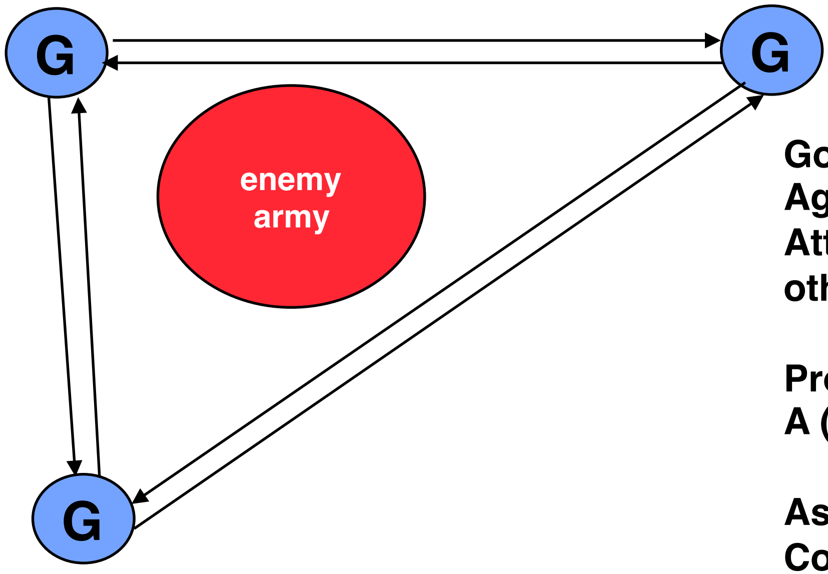
3 STEPS ➡

DETECTION
DISSEMINATION
EVALUATION

Byzantine Faults and Byzantine Agreement

L. Lamport, R. Shostak, M. Pease: „The byzantine generals' problem“, ACM TC on Progr. Languages and systems, 4(3), 1982

The Story:



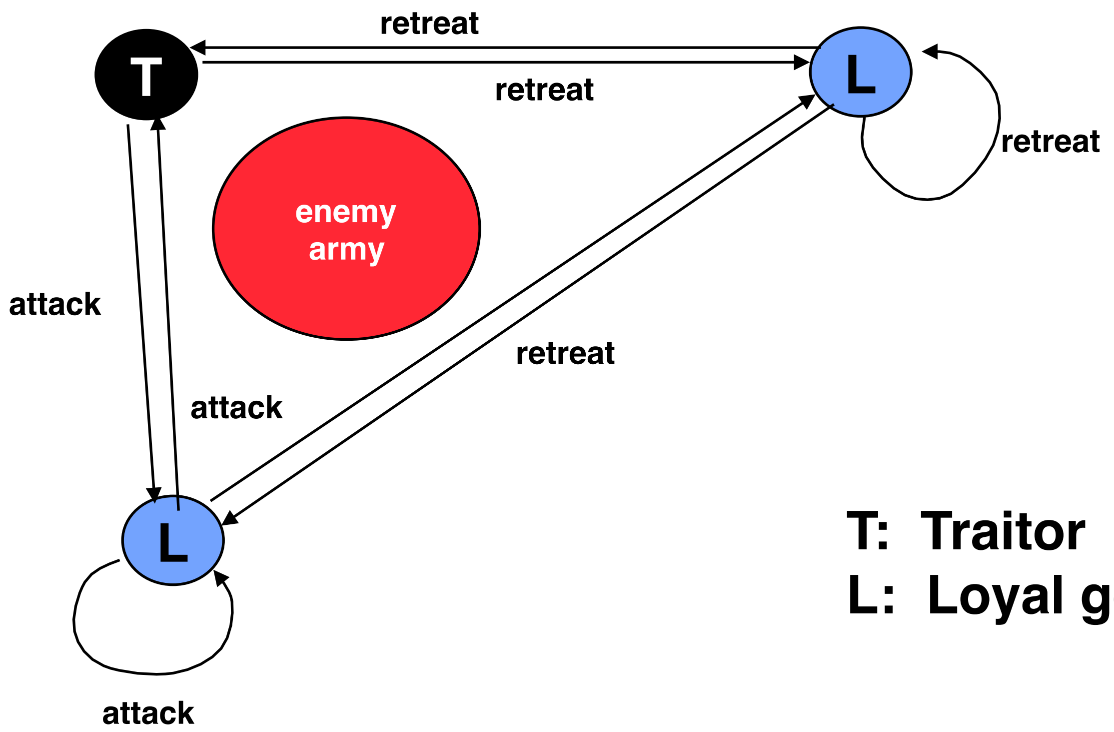
Goal:
Agreement about a common action.
Attack or retreat? Only a joint attack will be successful, otherwise the allies will be defeated.

Problem:
A (single) traitor

Assumptions:
Communication via a reliable point-to-point network.

Under which conditions and by which protocol is it possible to derive a correct majority vote?

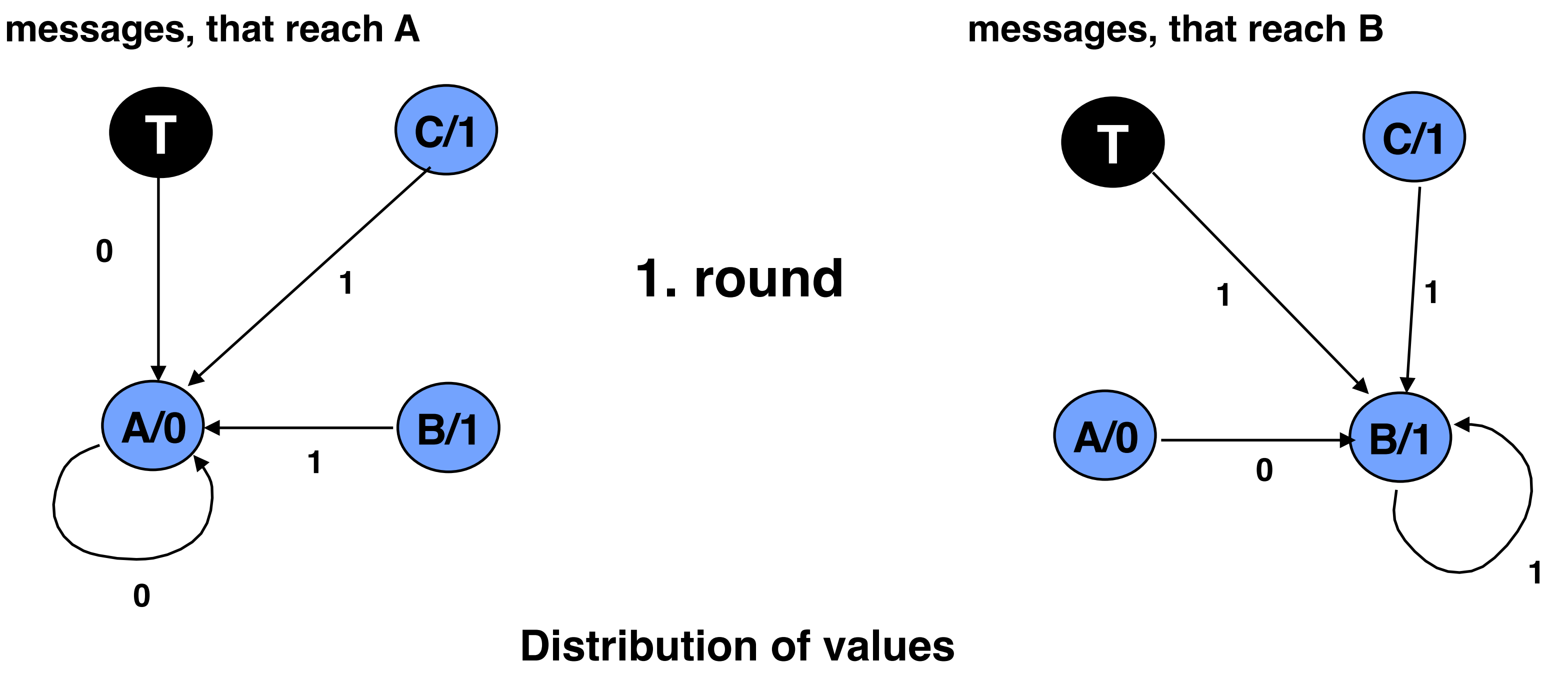
Byzantine Faults and Byzantine Agreement



T: Traitor
L: Loyal general

Even multiple rounds will not help to achieve agreement because a loyal general never knows who is the traitor.

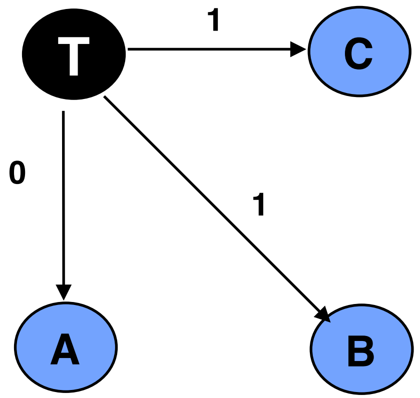
Agreement on a value in two rounds



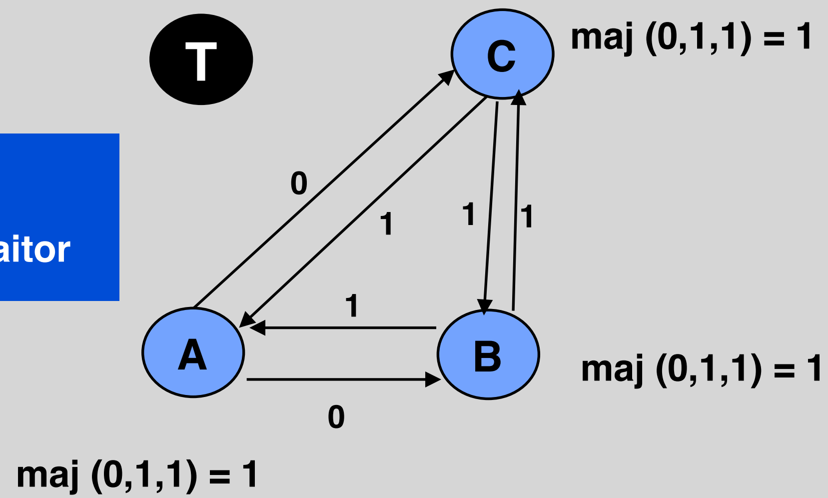
During the first round no unambiguous decision is possible because A and B don't agree.

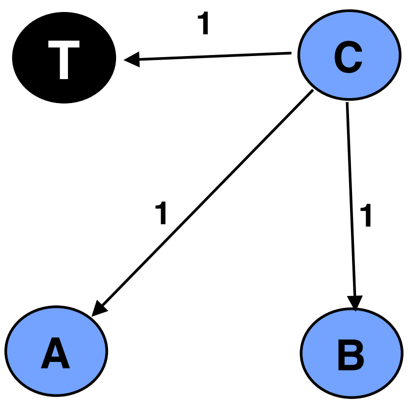
1. round
distribution of values from some participant

2. round
agreement on a value proposed by some participant.

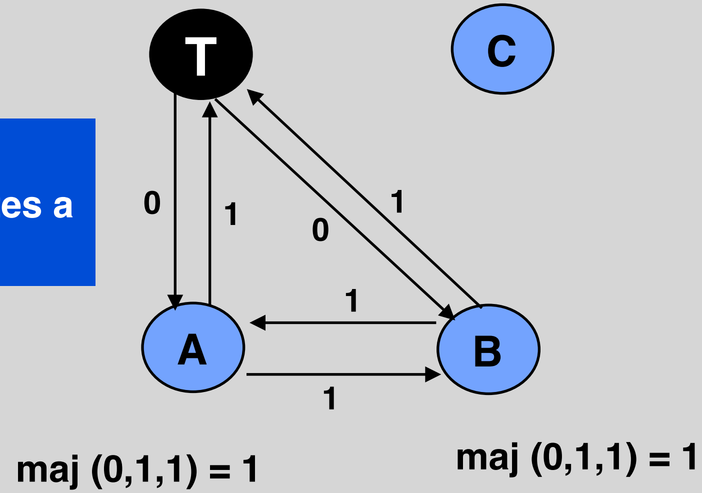


1. case
sender is the traitor





2. case
traitor disseminates a faulty value.



- Participants are processes.

- Evenry process locally desides by majority voting on the value that is decided by evera correct process.

- The value decided by the majority of processes is the corect value.

- To detect f byzantine faults,

(3f + 1) processes are needed.

Byzantine Faults and Byzantine Agreement

In a centralized evaluation, cheating is impossible, i.e. the central observer either receives a "good" or "faulty" result. Therefore, simple majority 2f+1 is sufficient.

In the distributed case, a faulty node may send different test outcomes to different nodes. Informally, the good nodes need to achieve a majority without the bad nodes. I.e. even if a good node has a wrong view on the state of some other node, it distributes this view consistently and no byzantine behaviour has to be considered in the subset of good nodes. Therefore in this subset, also simple majority is sufficient.

The equation 3f +1 can be written as: (2f + 1) + f

Failure Semantics and Coverage

David Powell, Failure Mode Assumptions and Assumption Coverage, Research Report 91462, March 1995

A supposedly fault-tolerant system may fail if any of the assumptions on which its design is based should prove to be false.

assumption	redundancy needed
k fail stop failures	k+1
k value failures	2k+1
k arbitrary failures	3k+1

failure assumptions have to be enforced in the system!

Failure Semantics and Coverage

The service delivered by a system can be defined in terms of a sequence of service items, s_i, i = 1,2,… each characterized by a tuple $\langle vs_i, ts_i \rangle$ where vs_i is the value or content of service item s_i and ts_i is the time of observation of service item s_i .

Def. 1. Service item s_i is correct iff: $(vs_i \in SV_i) \wedge (ts_i \in ST_i)$ where SV_i and ST_i are respectively the specified sets of values and times for service item . si .

$SV_i = \{sv_i\}$:	Set of correct values
$ST_i = [st_{min}(i), st_{max}(i)]$	Correct time interval

Formal Definition of failures

Arbitrary value error: s_i : $vs_i \notin Sv_i$

Noncode value error: s_i : $vs_i \notin CV$ where CV defines a code.

Arbitrary timing error: s_i : $ts_i \notin St_i$

Early timing error: s_i : $ts_i < \min(ST_i)$

Late timing error (or performance error): s_i : $ts_i > \max(ST_i)$

Infinitely late timing error or omission error: s_i : $ts_i = \infty$

Impromptu error: si : $(vs_i = \perp) \wedge (ts_i = \perp)$ since the admissible value and time sets are undefined s_i : $(vs_i \notin SV_i) \wedge (ts_i \notin ST_i)$

Formal definition of failures

V_{none}:= $\forall$ i, $vs_i \in Sv_i$: No value errors occur (every service item is of correct value).

V_N:= $\forall$ i, ($vs_i \in Sv_i$) , ($vs_i \notin CV$) : The only value errors that occur are non-code value errors (every service item value is either correct or noncode).

V_{arb}:= $\forall$ i, ($vs_i \in Sv_i$) $\vee$ ($vs_i \notin Sv_i$) $\equiv$ true : Arbitrary value errors can occur.

$$\boxed{V_{none}} \rightarrow \boxed{V_N} \rightarrow \boxed{V_{arb}}$$

Value error implication graph

Formal definition of failures

T_{none} : No timing errors occur (every service item is delivered on time).

T_O : The only timing errors that occur are omission errors. (every service item is deliverd on time or not at all)

T_L : The only timing errors that occur are late timing errors (every service item is delivered on time or too late)

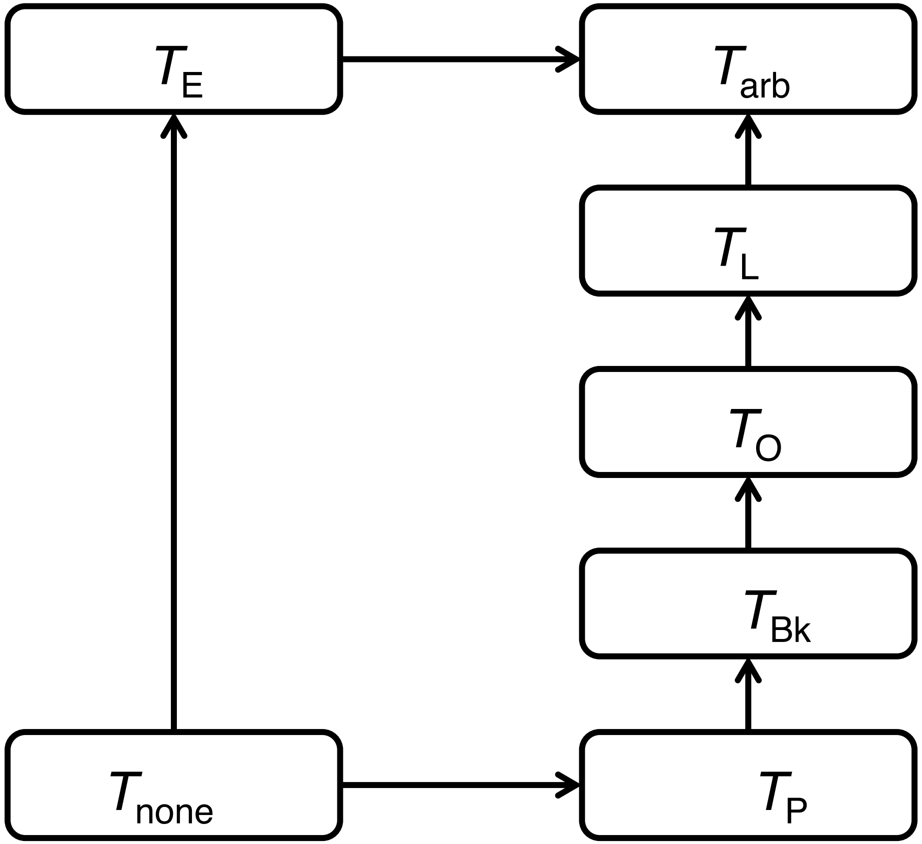
T_E : The only timing errors that occur are early timing errors (every service item is delivered on time or too early)

T_{arb} : Arbitrary timing errors can occur.

T_p : Permanent Timing failure: a component delivers correctly timed service items up to a particular item and then ceases (omits) to deliver service items.

T_{Bk} : Bounded ommission degree: a component omits to deliver some service items but, if more than k contiguous items are omitted then all further items are omitted.

Formal definition of failures



Timing error implication graph

Coverage

The failure mode assumption coverage **(**P_X **)** is defined as the probability that the assertion X defining the assumed behavior of a component proves to be true in practice conditioned on the fact that the component has failed:

$$P_X = \Pr \{X = \text{true} \mid \text{component failed}\}$$

Coverage:

$V_{arb} \wedge T_{arb}$ **= 1**

$V_{none} \wedge T_{none}$ **= 0**

All intermediate assumptions thus have a coverage $p \in]0,1[$.

Discussion

Does the highest coverage always lead to the most reliable system?

Summary and Points to Remember

- Strong failure semantics eases distributed system programming.

- Redundancy requirements:

    - In a centralized system and under a non-byzantine fault model, 2f+1 processes can achieve consistent system diagnosis.
    - Under a distributed system model and byzantine faults 3f+1 processes are needed.

- Synchrony requirements:

    - Synchronous systems and bounds on the communication delays allow deterministic consensus in a distributed system.
    - In an asynchronous system deterministic consensus is impossible if one process may be faulty.