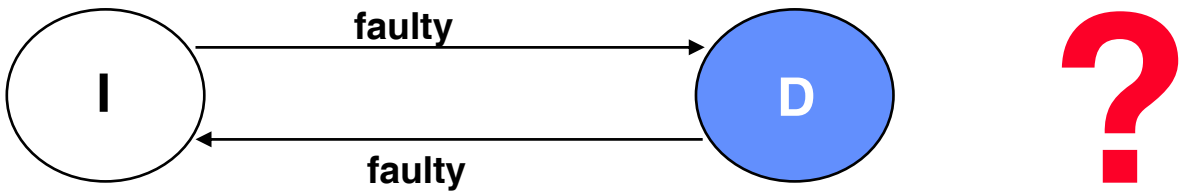
Concepts and Mechanisms of Dependable Systems

Failure detection in Distributed Systems

System diagnosis to detect and localize faults



?

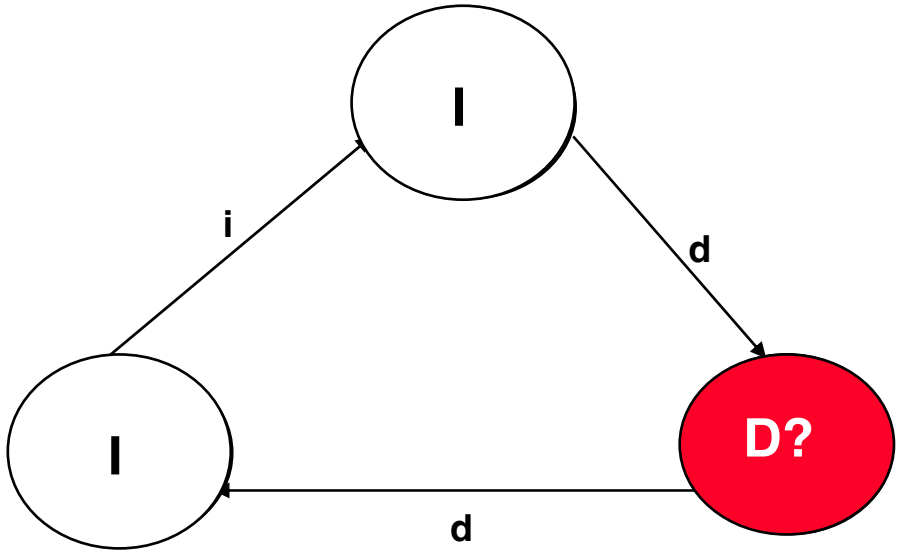
Assumptions:

- 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



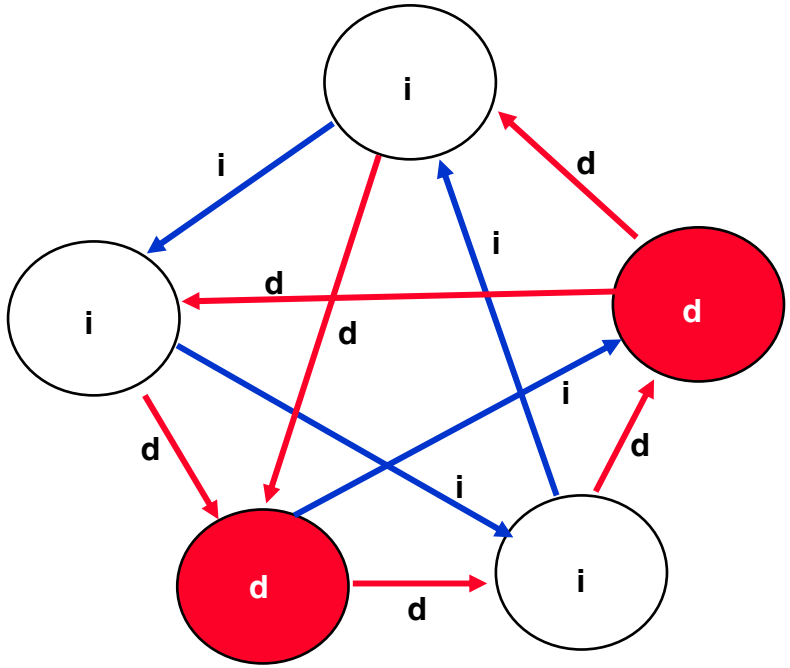
Assumptions:

- 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≥ 2f +1 and every component test at least f other components.
The components do not test each other.

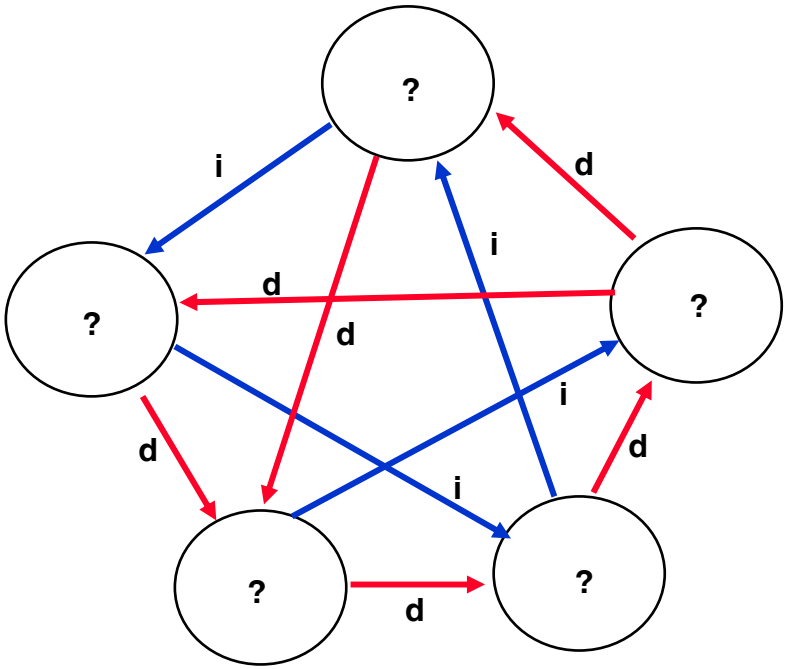
2-diagnosable system



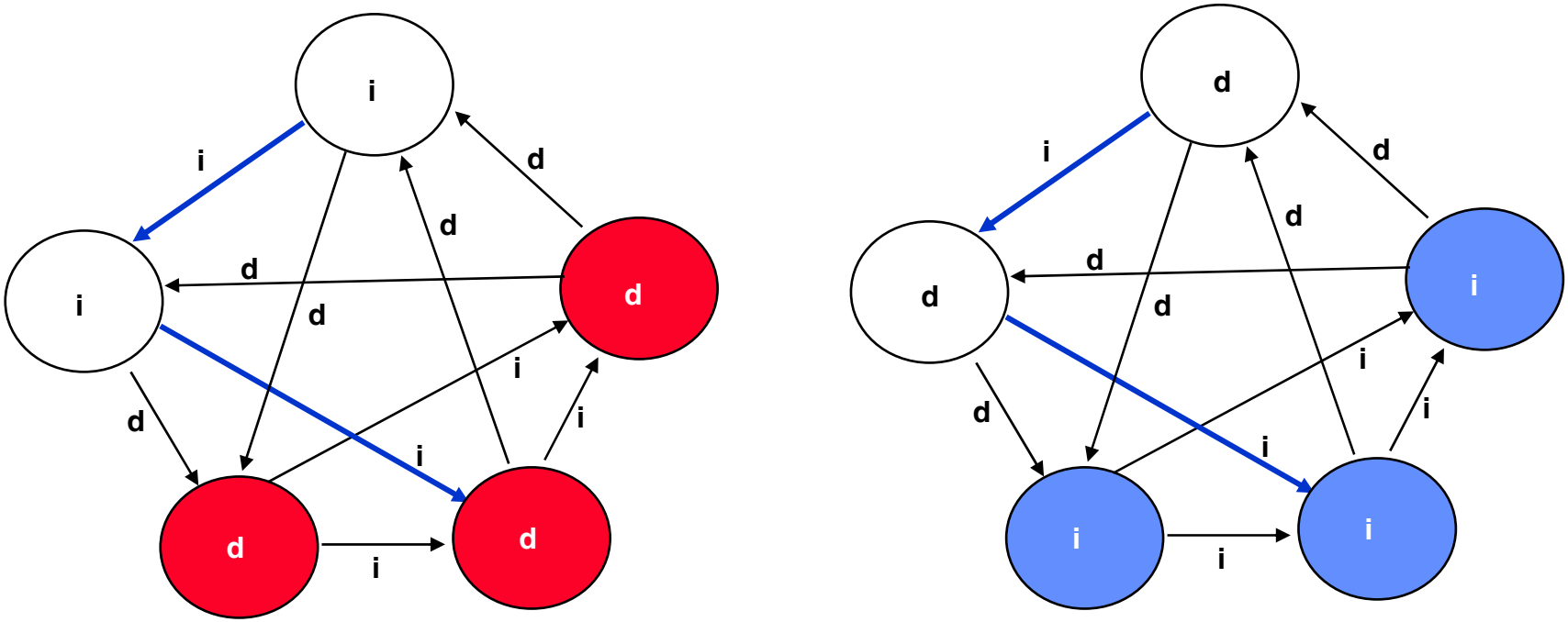
Assumptions:

- 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.

Can diagnosis deliver an unambiguous result?



3 faulty nodes



failure 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 failures have to be considered?
⇒ Fault model.

2. Can we replace the central evaluation component?
⇒ Distrbuted consensus.

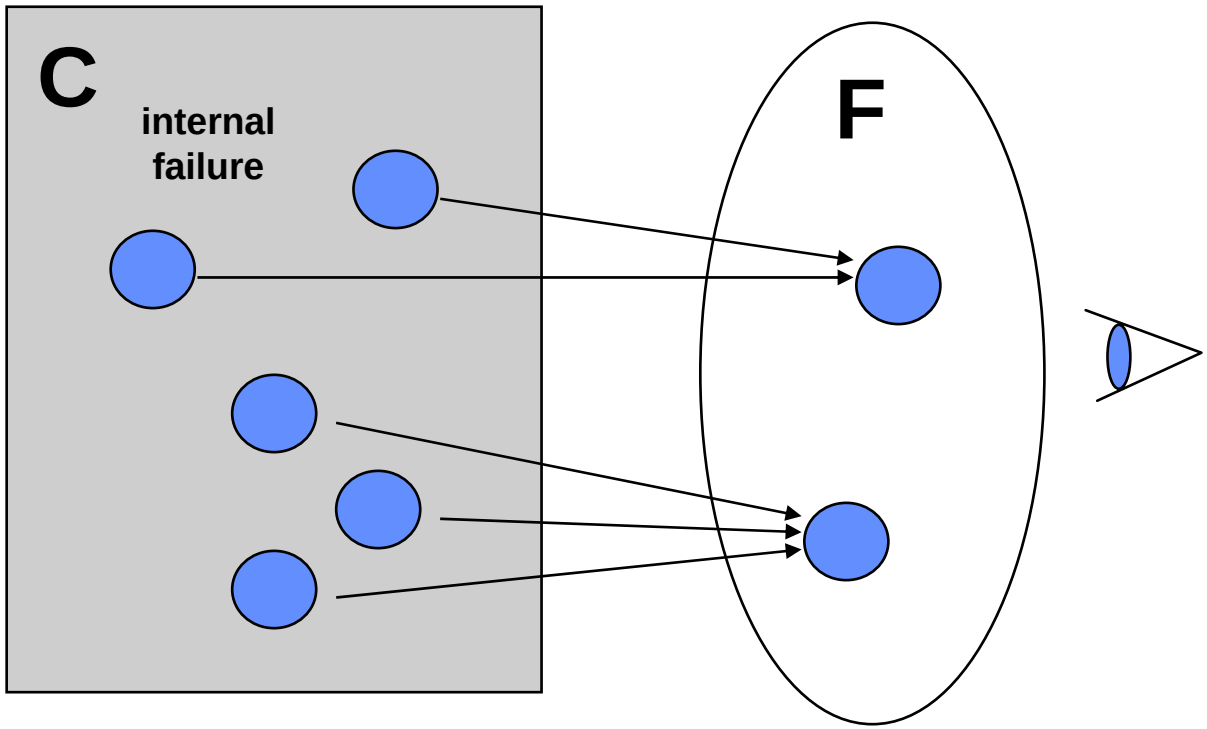
3. Can Fault-detection always successfully performed?
⇒ The problem of synchrony.

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.

C: System component

fault class



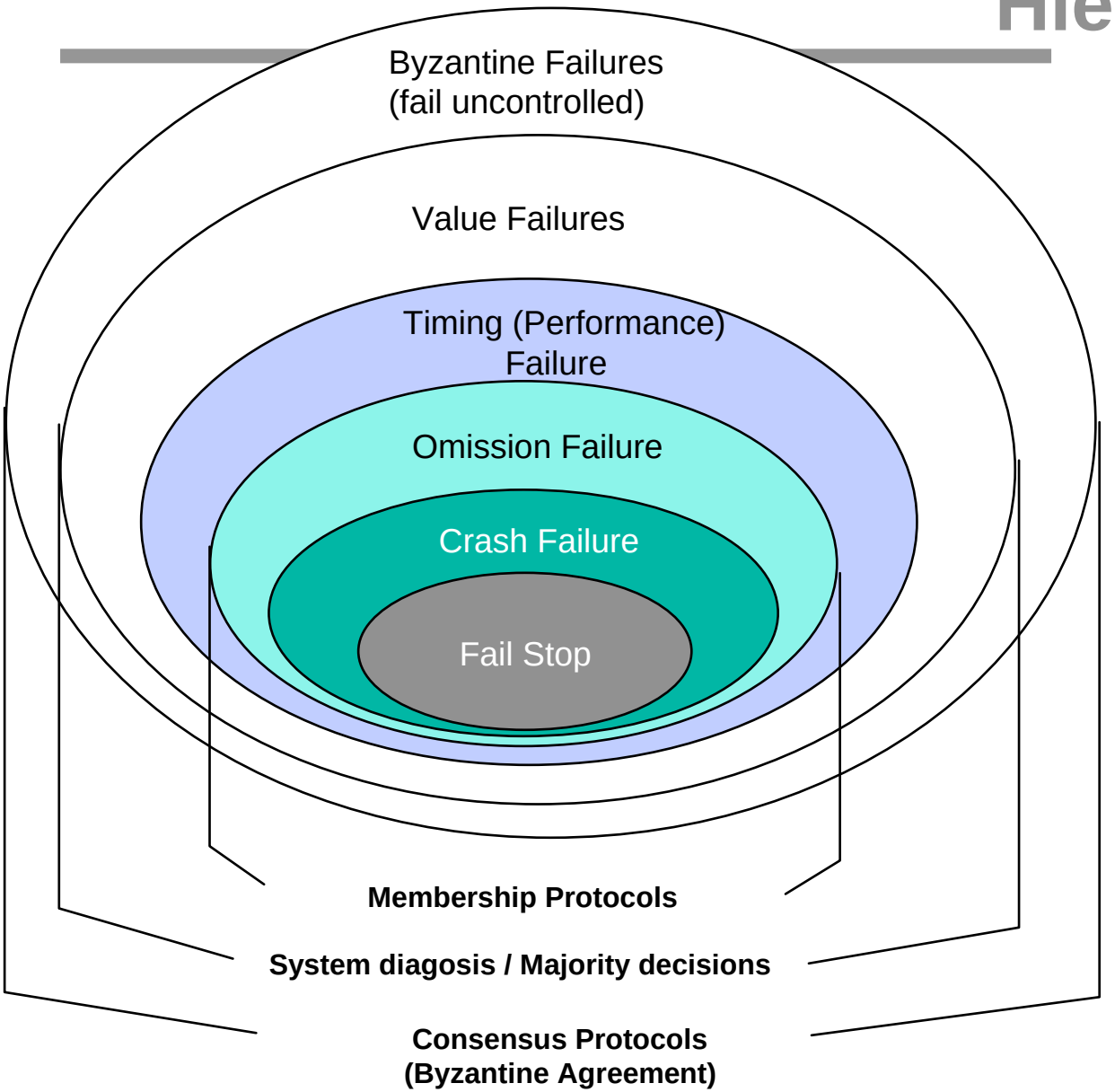
Problem:
The mechanisms to handle component failures are related to the assumed fault class.

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



Byzantine Failure:
Arbitrary, uncontrolled.

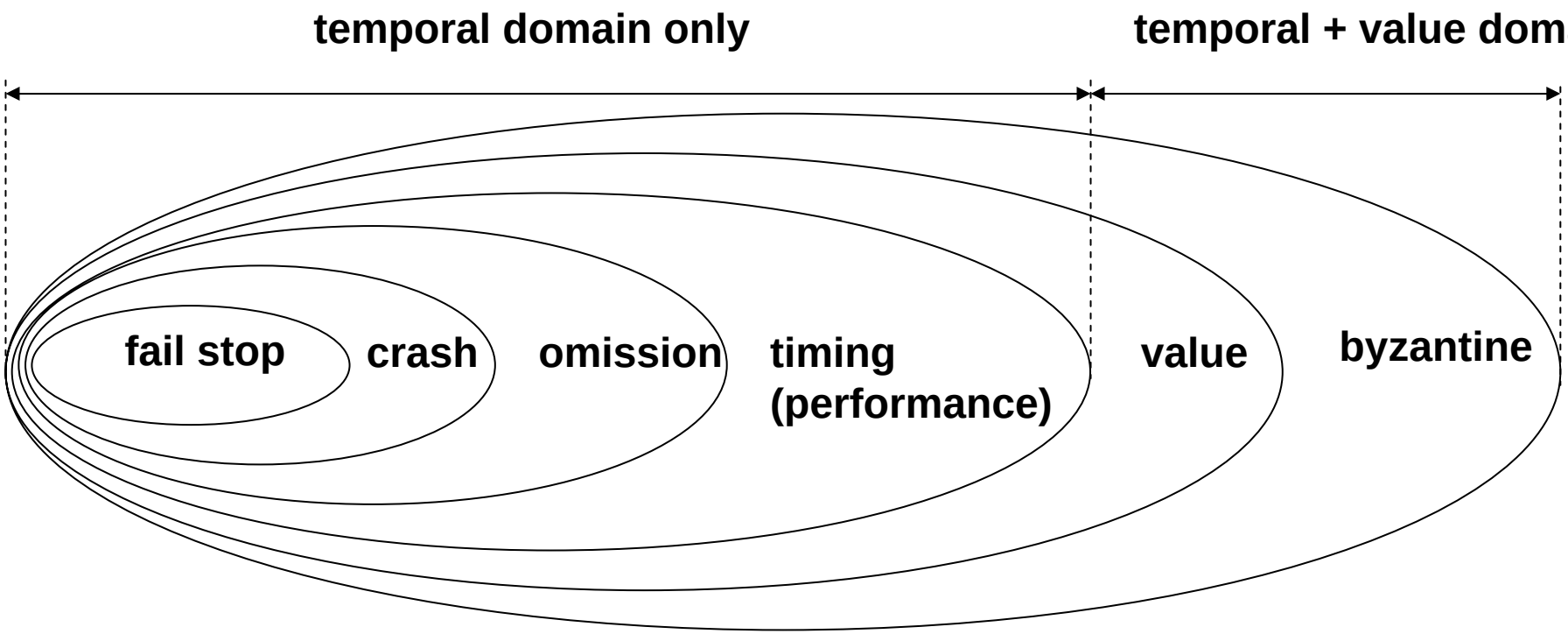
Timing (Prerformance) Failures
Correct values but too early or too late.

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 tomproduce 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.

The Network or the Node?

Fault-assumptions in Distributed Systems

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 amy 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 or channel	an arbitrary behaviour of process or channel.

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.

Intuitive Consistency Criterion:

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

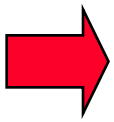
Formalisation by Chandra and Tueg (1996):

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).

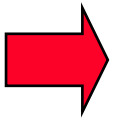
What are the conditions to achieve SA and SC?

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.**

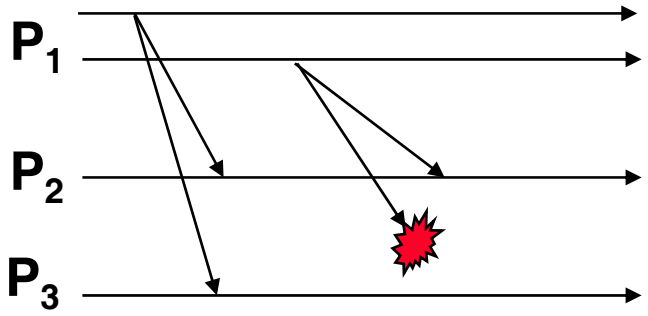


Apply mechanisms to mask omissions.

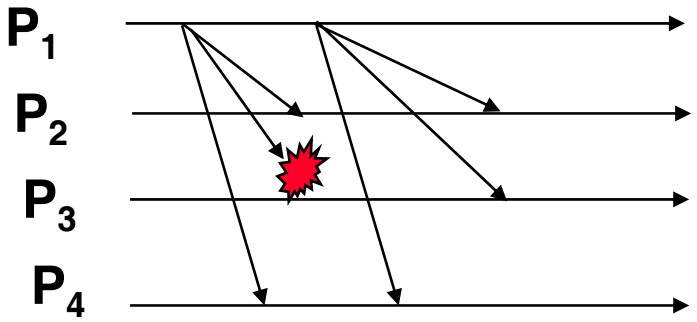
FT communication - Handling message failures

Static Redundancy: Masking Failures

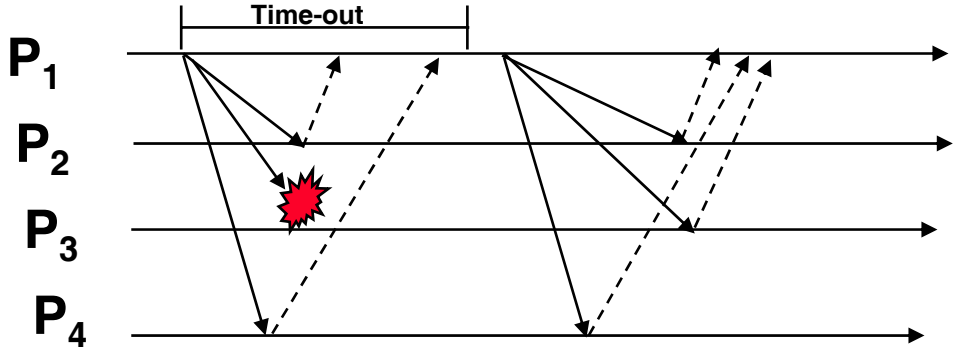
component redundancy



time redundancy

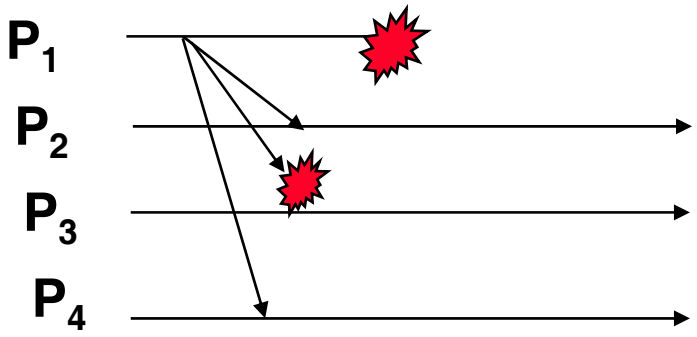


Dynamic Redundancy: Detection + Recovery

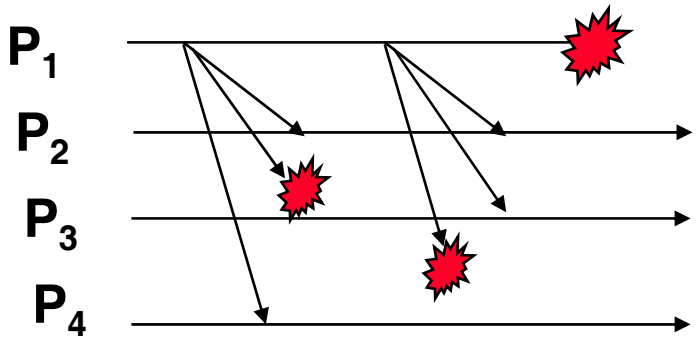


FT Communication - Handling sender failures

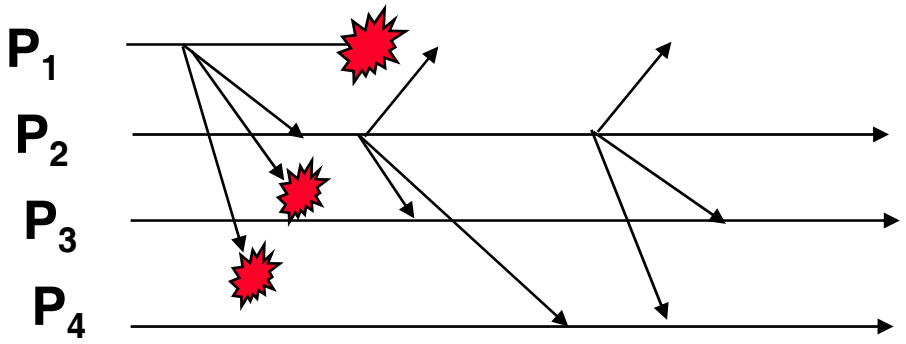
Unreliable Multicast



Best effort Multicast



Reliable Multicast

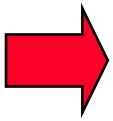


Imperfect failure detectors

Assumptions:

Temporal assumptions:
1. **the latency of messages cannot be bounded,**
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

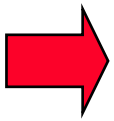
Goal: A group of processes agree on a common value.
Every process proposes a value once.
Every process decide a value once.
Proposed and decided values are 0 or 1 (simplification).

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.



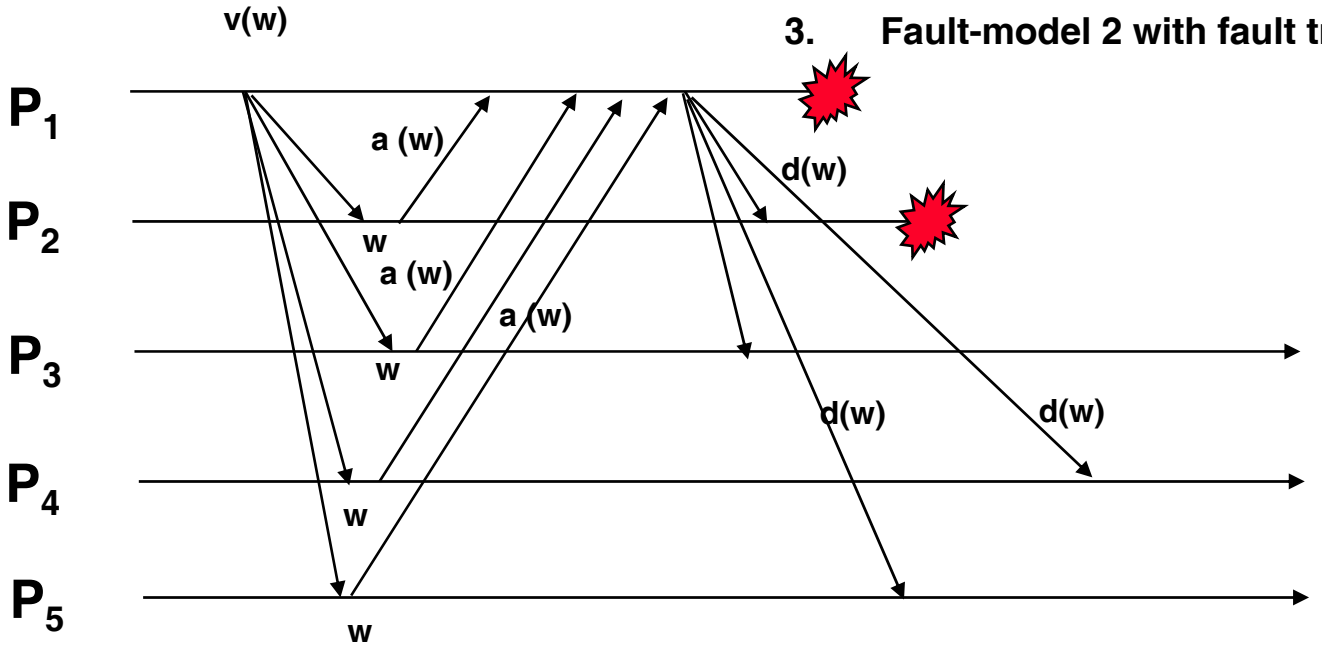
FLP Impossibility Result

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. **Fault-model 2 with fault treatment.**

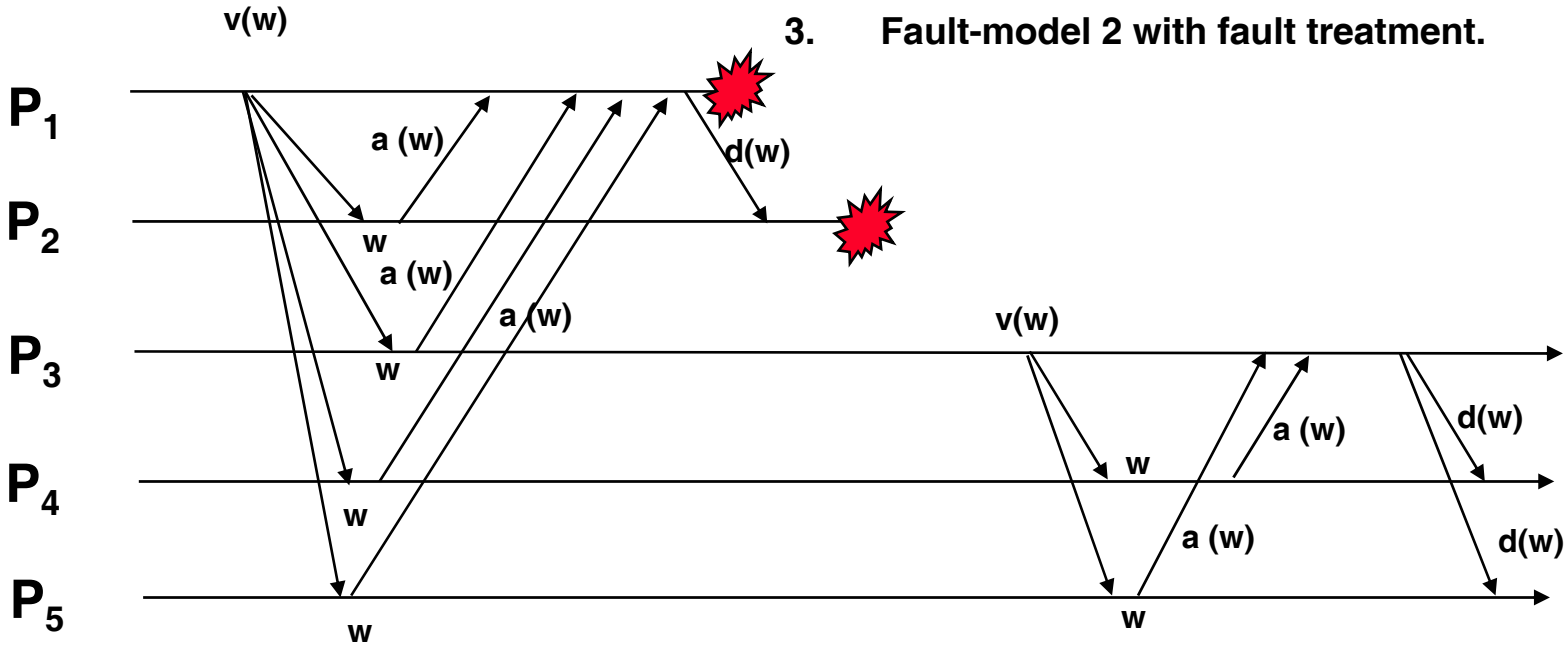


v(w): suggest(w)
a(w): accepted (w)
d(w): decided (w)

Fault-Tolerant Consensus

Assumptions:

1. **The latency of messages is bounded.**
2. **Failure detection is reliable.**
3. **Fault-model 2 with fault treatment.**

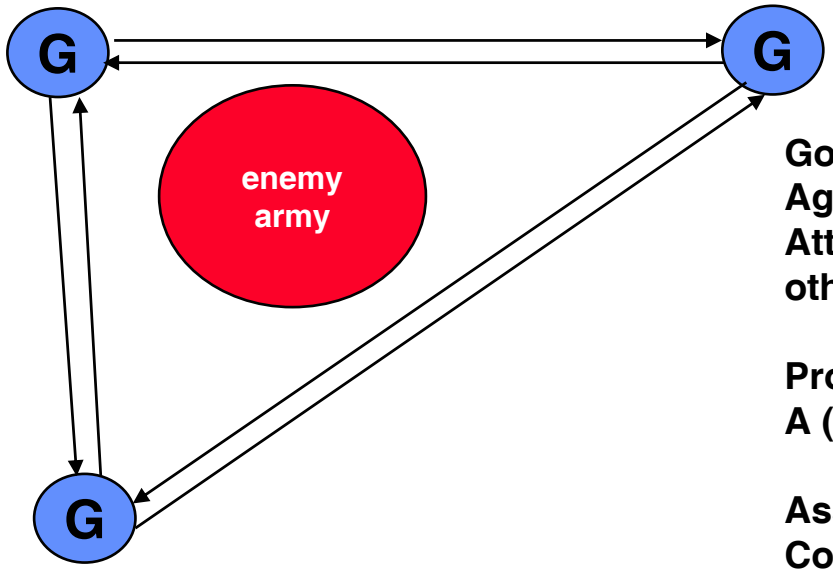


v(w): suggest(w)
a(w): accepted (w)
d(w): decided (w)

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:



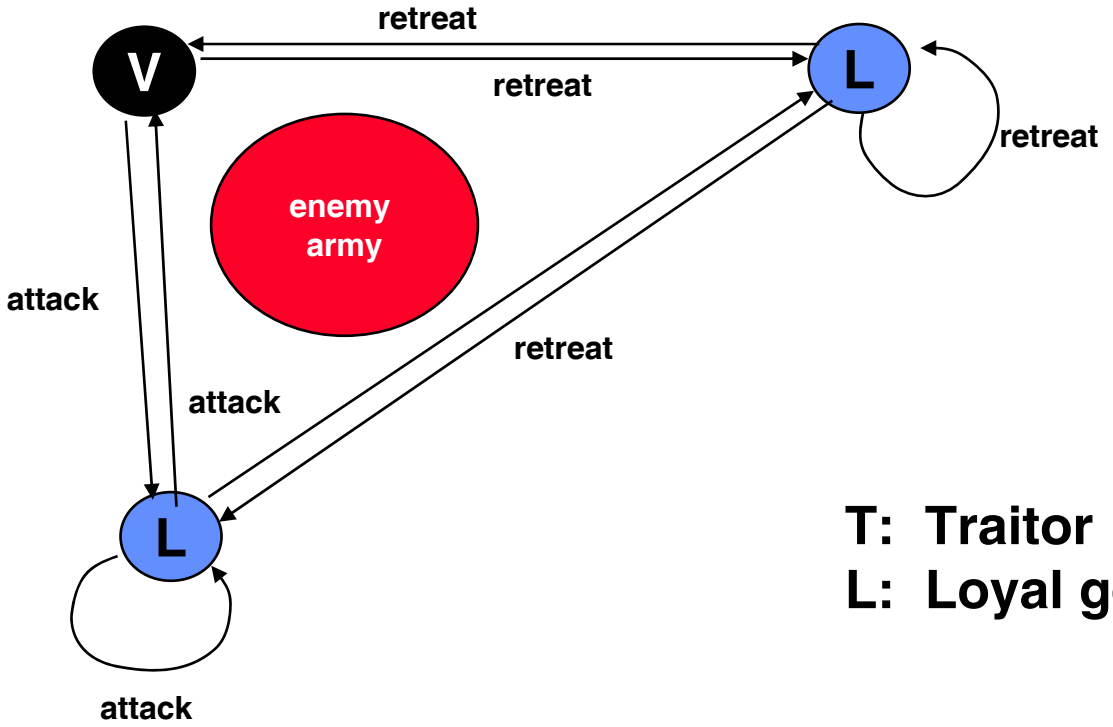
Goal:
Agreement about a common action.
Attack or retreat? Only a joint attack will be successful, otherwise the allies will be defeated.

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?

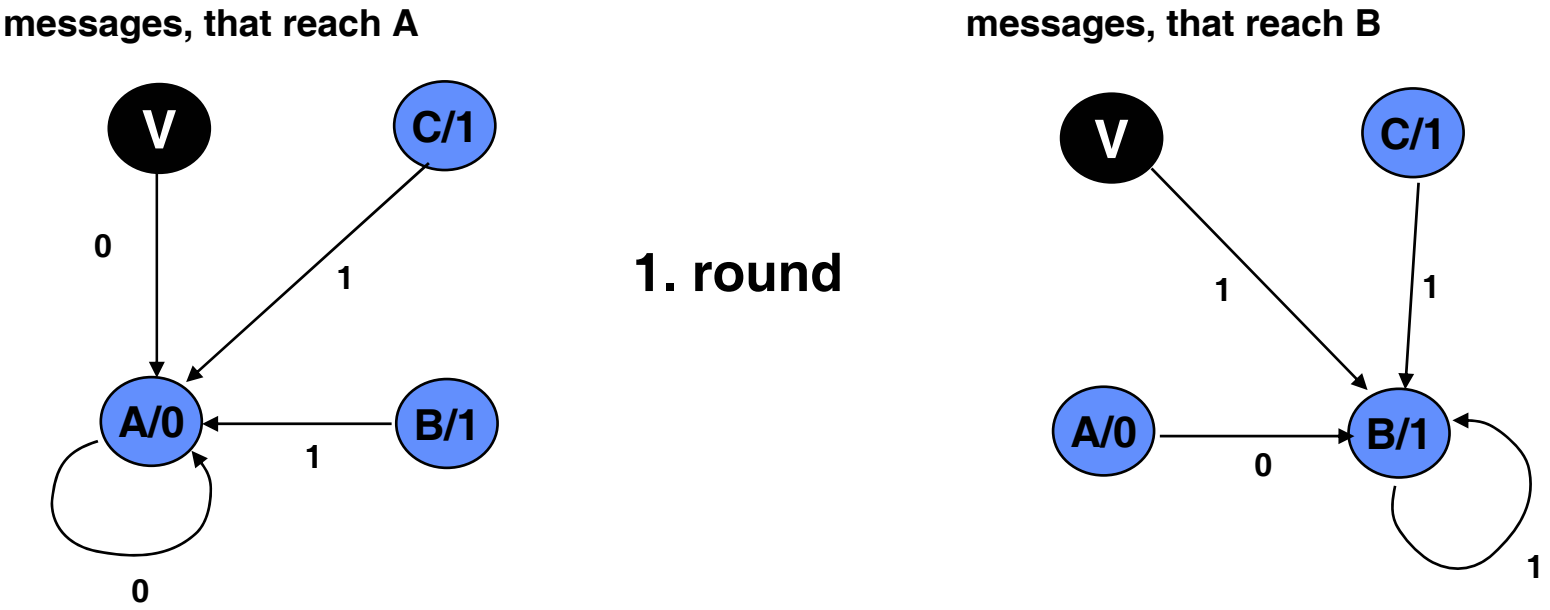
Byzantine Faults and Byzantine Agreement



T: Traitor
L: Loyal general

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

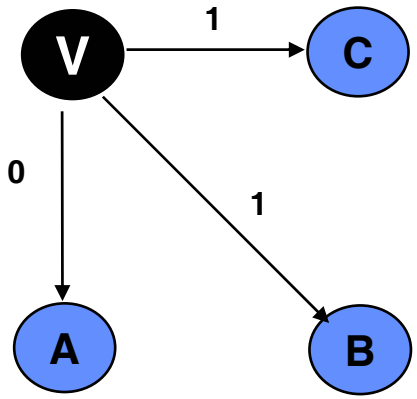
Agreement on a value in two rounds



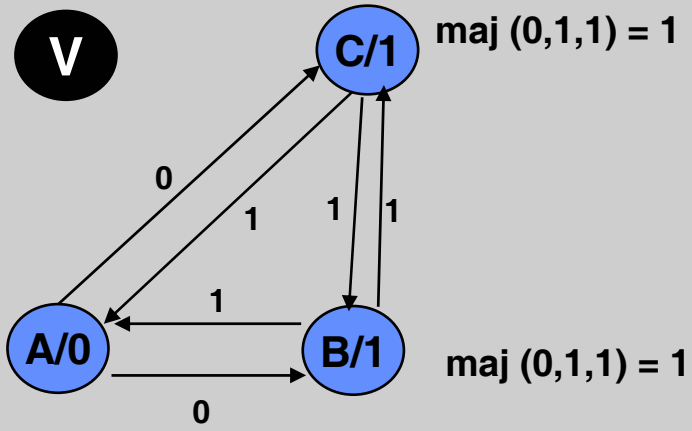
Distribution of values

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

1. round
distribution of values from some participant

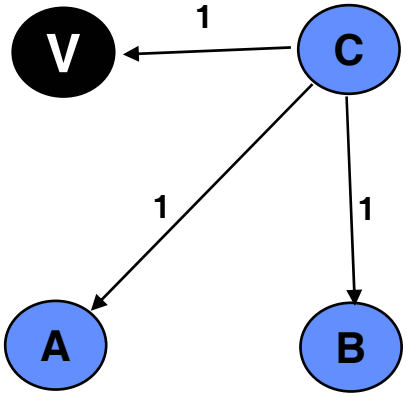


2. round
agreement on a value proposed by some participant.

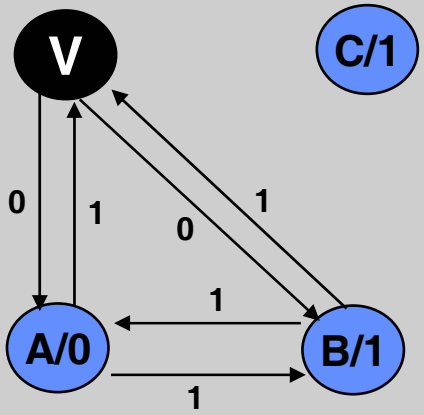


maj (0,1,1) = 1

1. case
sender is the traitor



2. case
traitor disseminates a faulty value.



maj (0,1,1) = 1

maj (0,1,1) = 1

- 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.