Distributed Systems

Winter Term 2024/25

Roland Wismüller
Universität Siegen
roland.wismueller@uni-siegen.de
Tel.: 0271/740-4050, Büro: H-B 8404

Stand: January 9, 2025

Distributed Systems

Winter Term 2024/25

09.01.2025

Roland Wismüller
Universität Siegen
roland.wismueller@uni-siegen.de
Tel.: 0271/740-4050, Büro: H-B 8404

Stand: January 9, 2025

Distributed Systems

Winter Term 2024/25

11 Fault Tolerance

Contents

- Introduction
- Process elasticity
- Reliable communication
- Recovery

Literature

- Tanenbaum, van Steen: Ch. 7

Concepts

- **Failure**: external incorrect behavior (system no longer keeps its promises)
- **Error**: (unobserved) incorrect internal state
- **Fault**: physical defect (in HW or SW) causing the error
  - fault can be transient, periodic or permanent
- **Fault tolerance**: system does not fail despite a fault
- Requirement for reliable systems:
  - **availability**: p(system is working at time t)
  - **reliability**: p(system is working in time interval Δt)
  - **safety**: no major damage if system fails
  - **maintainability**: effort for “repair” after a failure

Failure models

Crash failure	Server halts
Omission failure	Server is not responding to requests
Receive omission	Server doesn't receive incoming requests
Send omission	Server doesn't send messages
Timing failure	Response time is outside the specification
Response failure	Server's response is incorrect
Value failure	Only the value of the answer is wrong
State transition f.	Incorrect control flow in server
Byzantine failure	Random answers at arbitrary time

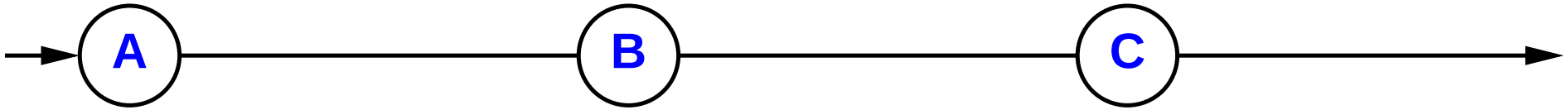
➥ Further distinction: can the client detect the failure or not?

Failure masking through redundancy

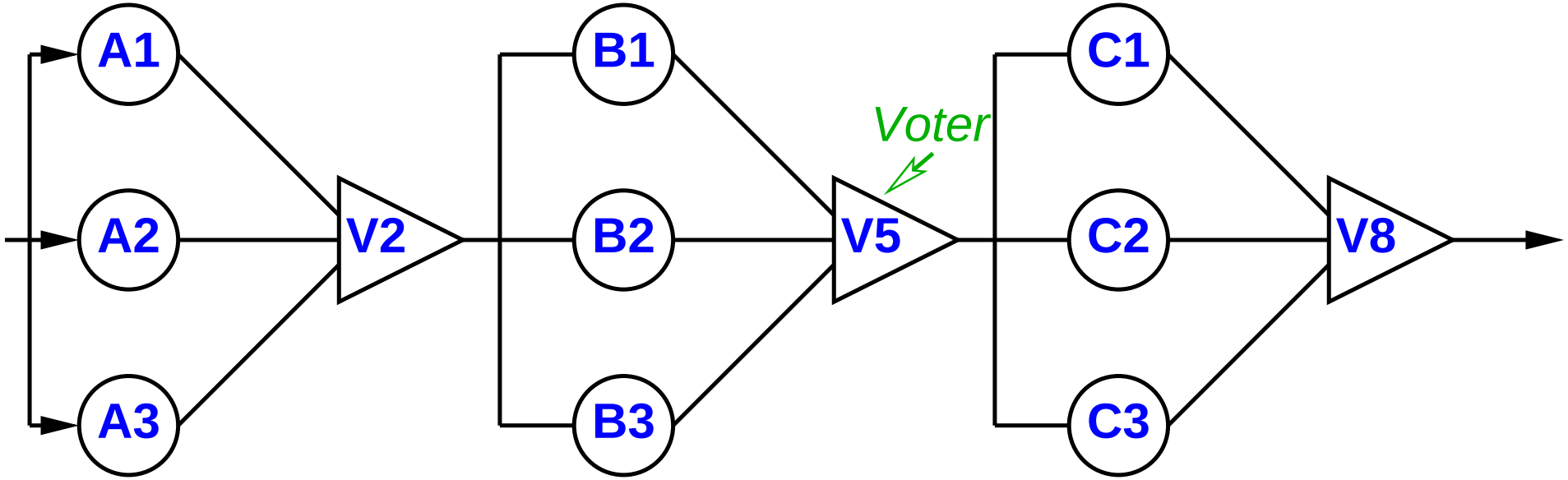
- Fault tolerant system must hide faults from other processes
- Most important technique: **redundancy**
  - **information redundancy**: additional “check bits” (e.g., CRC)
  - **time redundancy**: repetition of faulty actions
  - **physical redundancy**: important components are provided multiple times
- Example: **TMR, *triple modular redundancy***
  - components are replicated three times
  - majority decision for the results
  - protects against (Byzantine) failure of a *single* replicated component

Example for TMR

Without redundancy

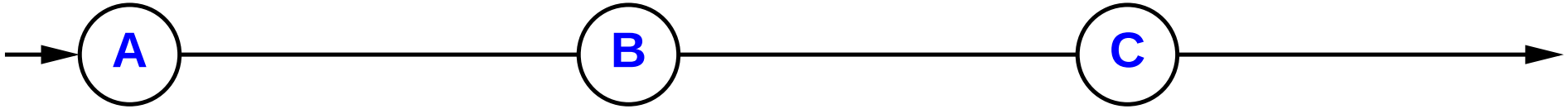


With TMR

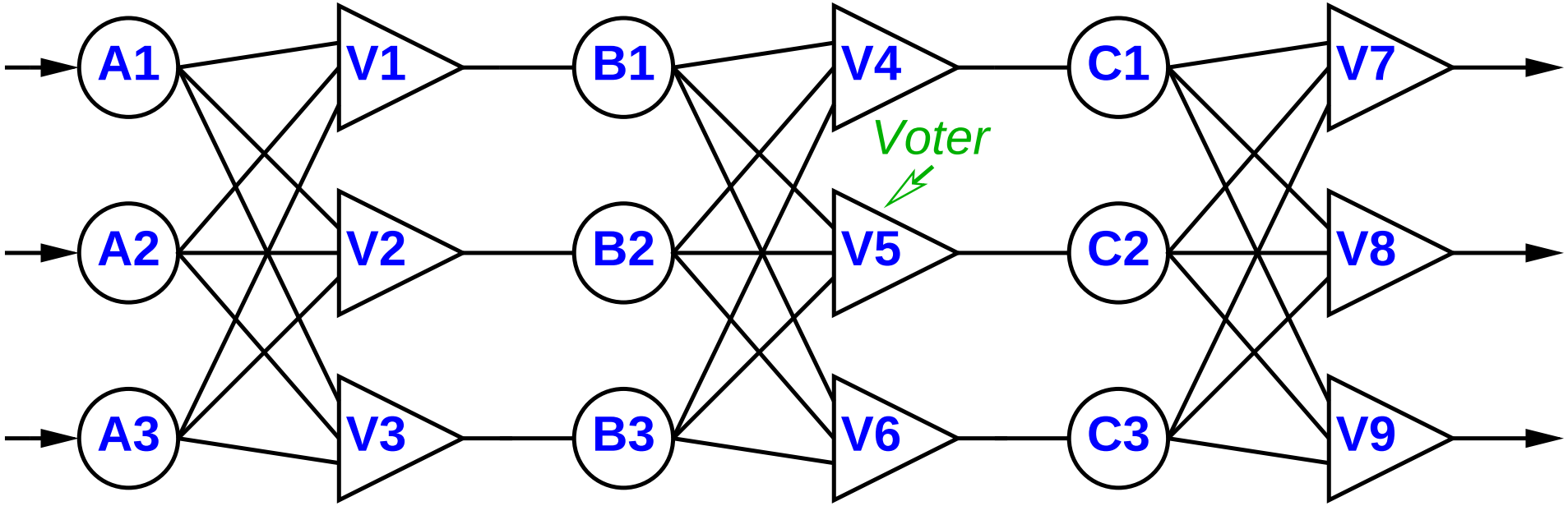


Example for TMR

Without redundancy



With TMR



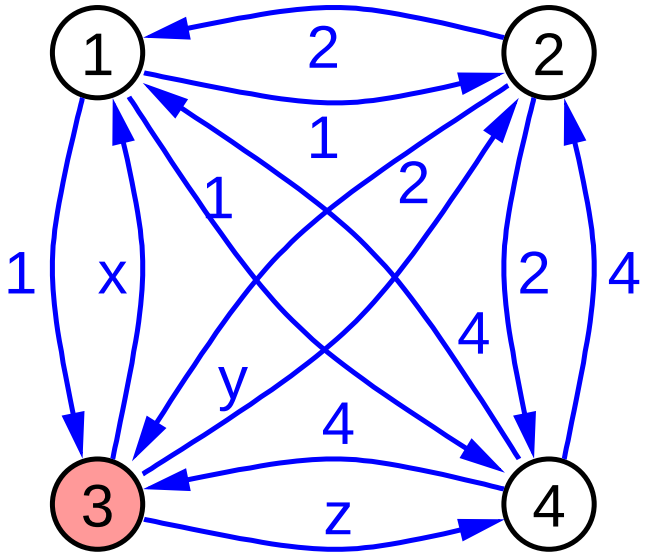
Objective: Protection Against Process Failure

- By replicating processes in groups
  - message to the group is received by all members
    - usually with totally ordered multicast
- Questions:
  - organization of the groups?
    - flat (symmetrical) vs. hierarchical (central coordinator)
    - group administration, synchronous join / exit
  - necessary number of replicas?
    - **k fault tolerant**: failure of k processes can be tolerated
    - for silent failures: $\geq k + 1$ Processes
    - for Byzantine failures: $\geq 2k + 1$ processes
  - agreement in faulty systems?

Agreement in faulty systems

- Agreement is impossible with unreliable communication
  - two army problem
- Agreement of faulty processes with reliable communication
  - Byzantine agreement problem (*byzantinische Generäle*)
  - agreement only possible if $> \frac{2}{3}$ of the processes work correctly

1. Send information



2. Received information

1: (1, 2, x, 4)
2: (1, 2, y, 4)
3: (t, u, v, w)
4: (1, 2, z, 4)

3. Send received information to all other processes

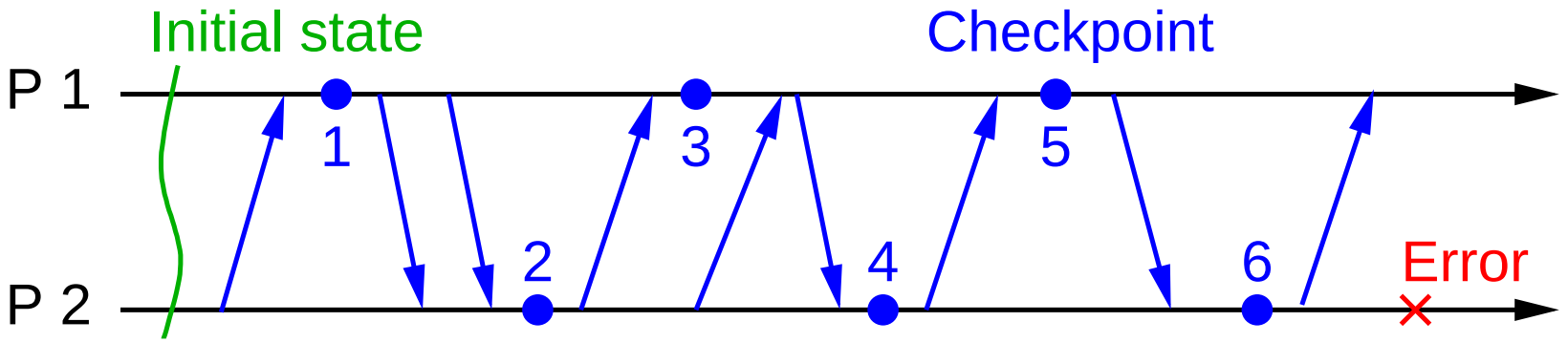
	1 gets:	2 gets:	4 gets:
from 1:		(1, 2, x ,4)	(1, 2, x ,4)
from 2:	(1, 2, y ,4)		(1, 2, y ,4)
from 3:	(a, b, c, d)	(e, f, g, h)	(i, j, k, l)
from 4:	(1, 2, z ,4)	(1, 2, z ,4)	

Objective: Protection Against Communication Failures

- Point-to-point communication (☞ **RN_I**)
  - TCP masks omission failures, but not crash failures
- Client/server communication (☞ **2.1**)
  - possible failures:
    - server not found
    - lost request
    - server crash while processing the request
    - lost reply
    - client crash after sending the request
- Group communication (☞ **7.3**)
- Distributed commit (☞ **7.4**)

Objective: System Recovery After an Error

- Forward error recovery: go to a correct new state
- Backward error recovery: go to a correct earlier state
  - i.e. reset to a consistent cut
  - regular backup to stable storage (*checkpointing*)
- Independent checkpointing
  - processes save their state independently of each other
  - problem: domino effect



11.4 Recovery ...

- Coordinated checkpoints
  - Chandy/Lamport algorithm (☞ **6.4**)
  - alternatively: blocking 2 phase protocol
  - problem: requires to reset all processes
- Local checkpoints with message logging
  - goal: restore the crashed process to a state consistent with the current state of the other processes
  - reset to last checkpoint and restore the received messages