Fault Tolerance

Yao Liu

Failures

Class of failure	*Affects*	*Description*
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a *send*, but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behavior: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.

Timing failures

Class of Failure	*Affects*	*Description*
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message's transmission takes longer than the stated bound.

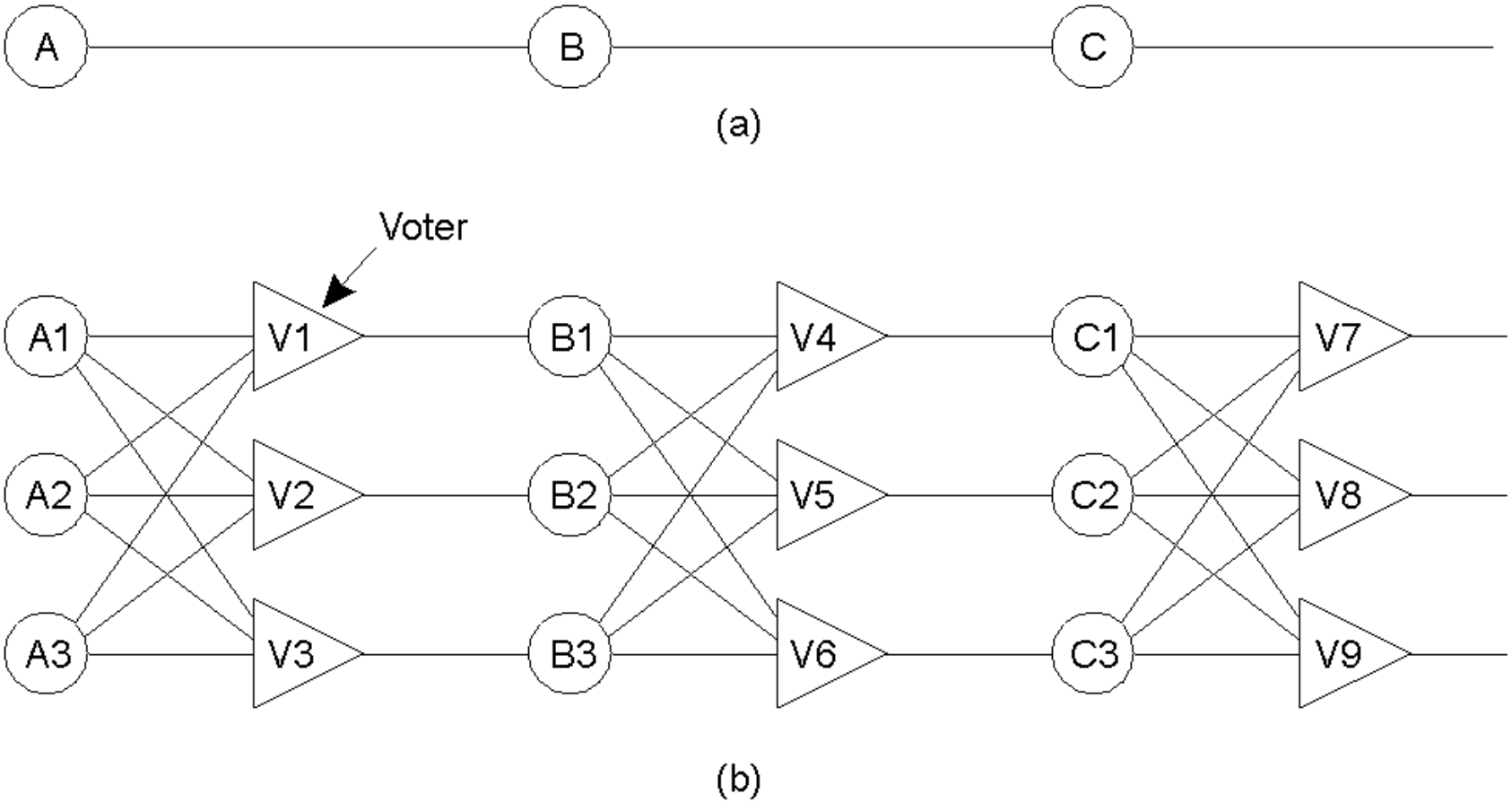
How to mask failures

Mask failures by redundancy

- Information redundancy
  - e.g., add extra bits to detect and recover data transmission errors
- Time redundancy
  - e.g., when a transaction aborts re-execute it without adverse effects
- Physical redundancy
  - Hardware redundancy
    - Take a distributed system with 4 file servers, each with a 0.95 chance of being up at any instant
    - The probability of all 4 being down simultaneously is $0.05^4 = 0.000006$
    - So the probability of at least one being available (i.e., the reliability of the full system) is 0.999994, far better than 0.95
    - If there are 2 servers, then the reliability of the system is $(1-0.05^2)= 0.9975$
  - Software redundancy
    - Process redundancy with similar considerations

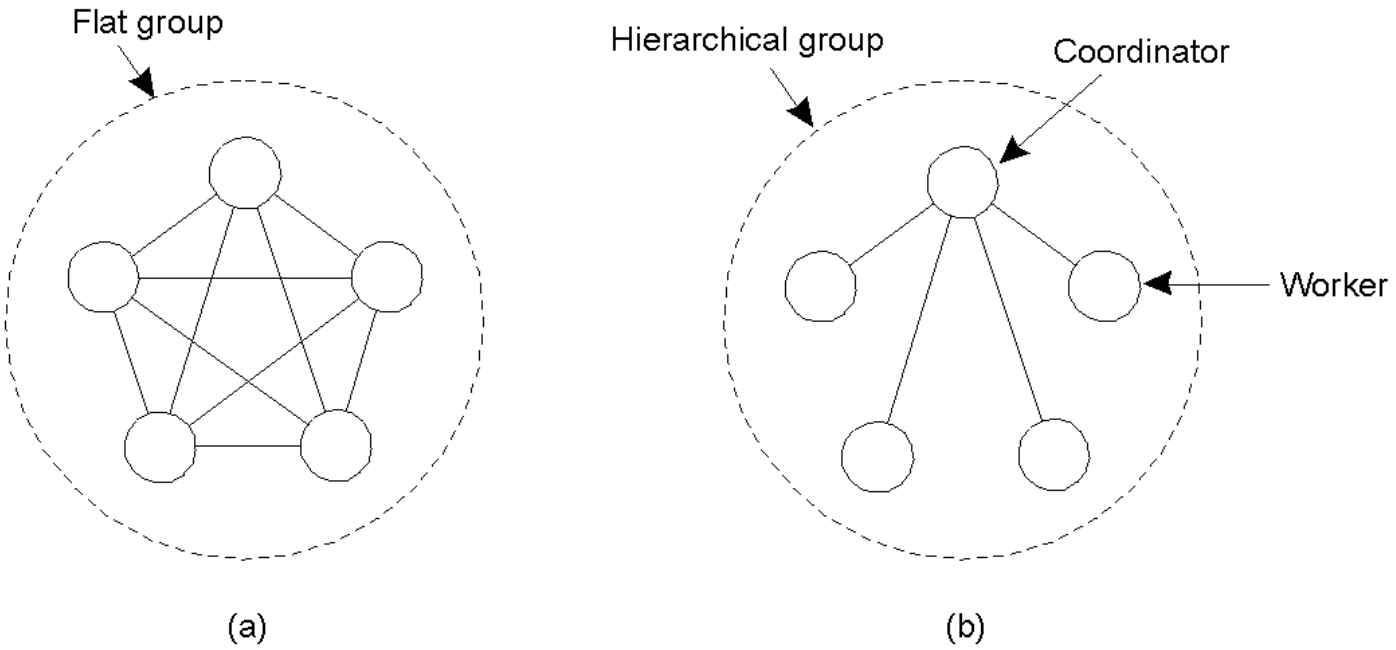
Failure masking by redundancy

- Triple modular redundancy.



Process redundancy

- Process group
  - All members of a group receive a message sent to the group
  - If one process fails, others can take over.
  - Can be dynamic; processes can have multiple memberships
  - Flat vs. hierarchical groups:



Management of replicated processes

- Primary copy
  - Primary-backup setup
  - Coordinator is the primary that coordinates all updates
  - If coordinator fails, one backup takes over (usually through an election procedure)
  - Processes are organized hierarchically
- Replicated-writes
  - Active replication and quorum-based protocols
  - Flat group organization
  - No single points of failure

Fault tolerance of process groups

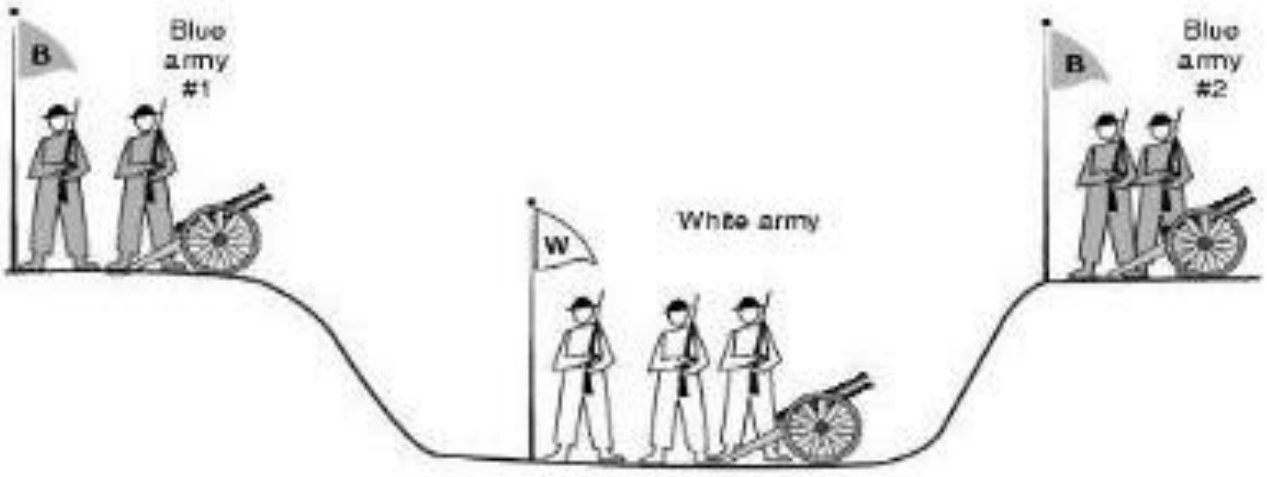
- A system is ***k*** fault tolerant if it can survive faults in ***k*** components and still meets its specification.
- If failures are safe (silent), then ***k*+1** processes are sufficient to get ***k*** fault tolerance.
- In case of arbitrary failures, **2*k*+1** processes are required (since *k* failing processes can all generate the same result by chance)

Agreement in faulty systems

- Many things can go wrong…
- Communication
  - Message transmission can be unreliable
  - Time taken to deliver a message is unbounded
  - Adversary can intercept messages
- Processes
  - Can fail or team up to produce wrong results
- Agreement very hard, sometimes impossible, to achieve!

Two-army problem

- "Two blue armies need to simultaneously attack the white army to win; otherwise they will be defeated. The blue army can communicate only across the area controlled by the white army which can intercept the messengers."



Two-army problem

- Good processors
- Faulty, unreliable communication channel
- Coordinated attack
- Provably impossible
- Multiple acknowledgement problem
  - Every message must be acknowledged, but every acknowledgement must be acknowledged, etc.…

Byzantine agreement problem

- Byzantine Agreement [Lamport, Shostak, Pease, 1982]
- Goal:
  - Each process learn the true values sent by correct processes
- Assumptions:
  - Every message that is sent is delivered correctly
  - The receiver knows who sent the message
  - Message delivery time is bounded

Byzantine agreement problem

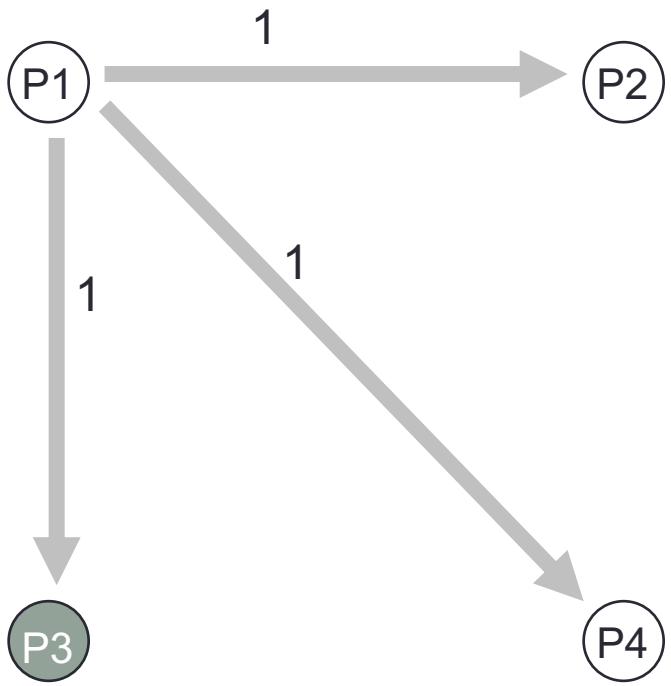
- Reliable communication channel
- Faulty processors
- *n* generals head different divisions
- *m* generals are traitors and are trying to prevent others from reaching agreement
  - 4 generals agree to attack
  - 4 generals agree to retreat
  - 1 traitor tells the 1st group that he'll attack and tells the 2nd group that he'll retreat
- Can the loyal generals reach agreement?

Byzantine agreement result

- In a system with ***m*** faulty processes, agreement can be achieved only if there are ***2m+1*** functioning correctly
- ***3m+1*** participants for ***m*** traitors
- Agreement is possible only if **more than two thirds** of the processes are working properly

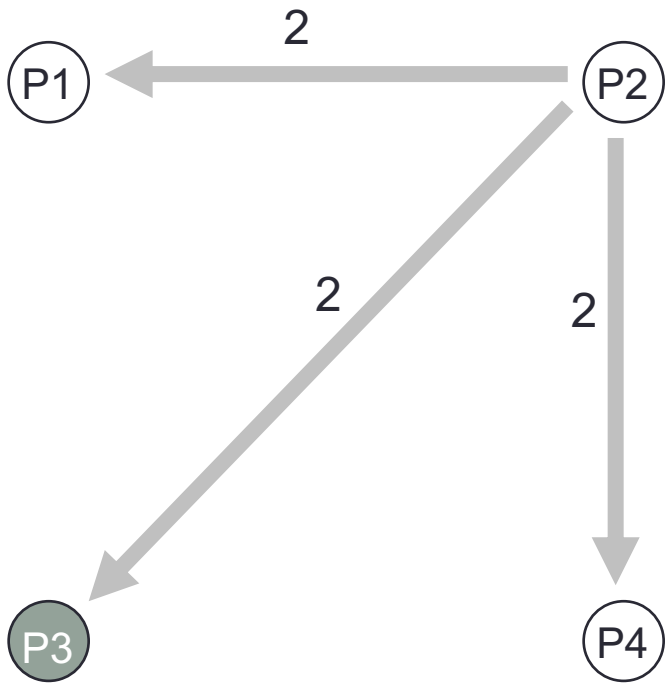
Byzantine agreement: example

- Phase 1: Generals announce their troop strengths to each other



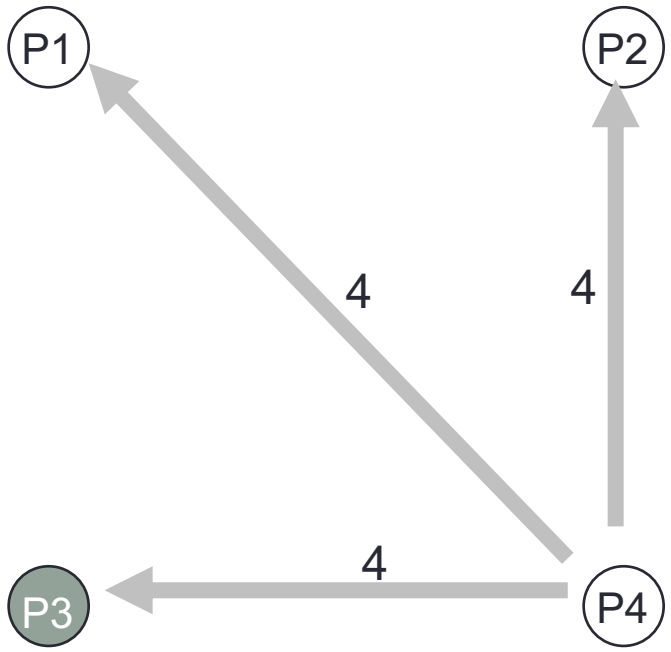
Byzantine agreement: example

- Phase 1: Generals announce their troop strengths to each other



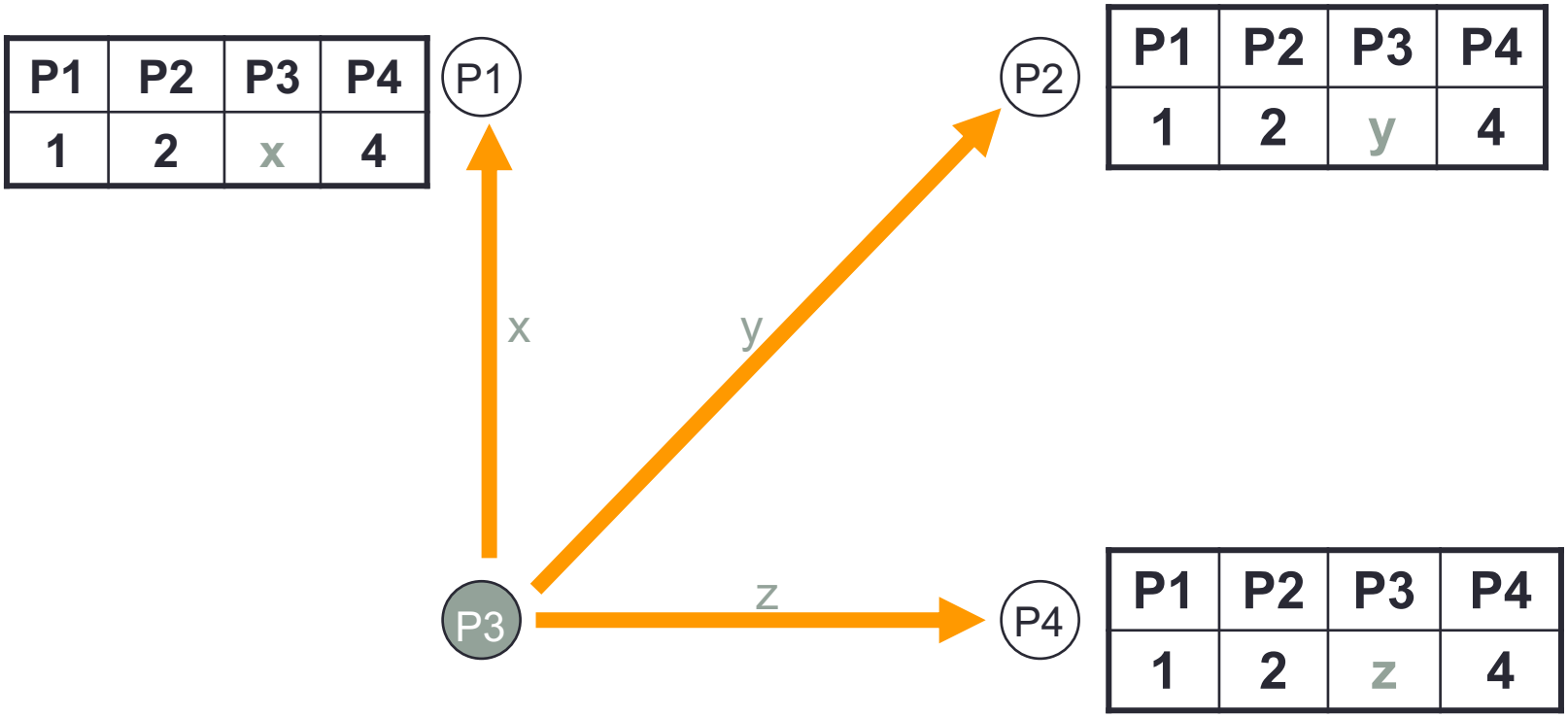
Byzantine agreement: example

- Phase 1: Generals announce their troop strengths to each other



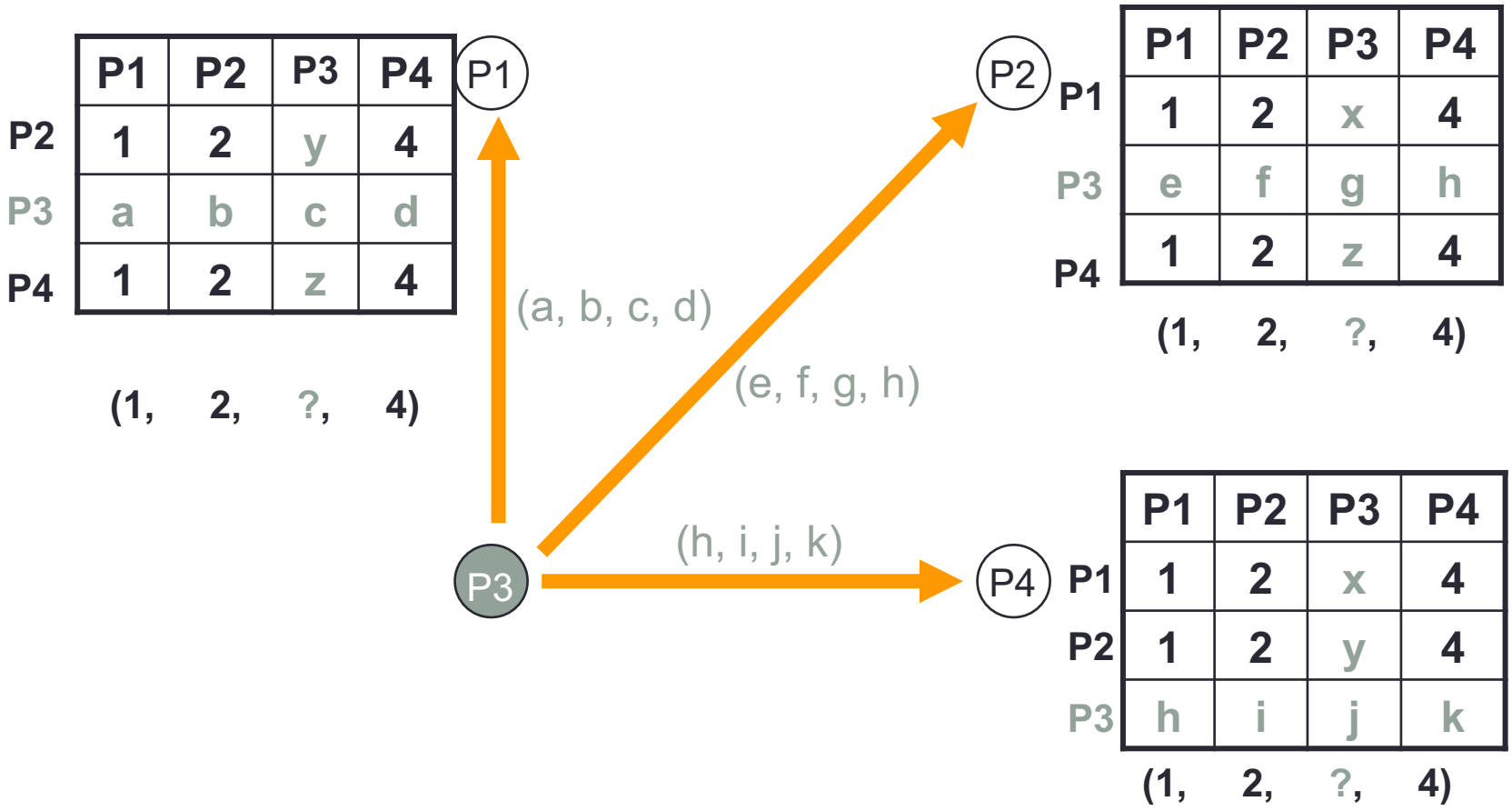
Byzantine agreement: example

- Phase 2: Each general construct a vector with all troops

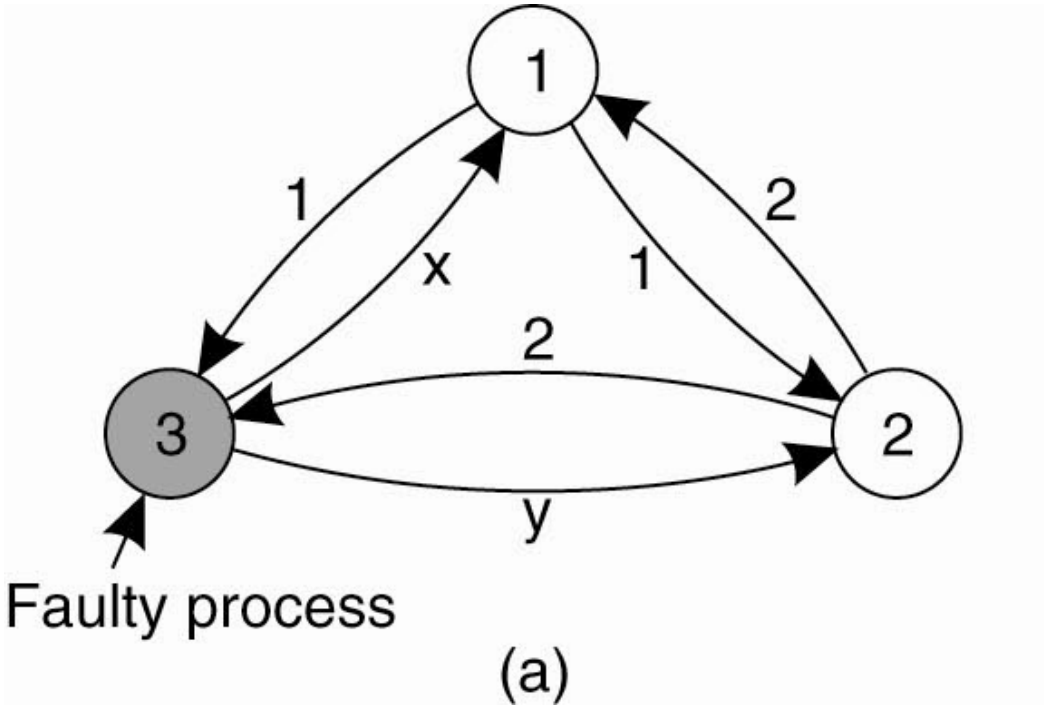


Byzantine agreement: example

- Phase 3: Generals send their vectors to each other and compute majority voting



Agreement in faulty systems



(a)

1 Got(1, 2, x)
2 Got(1, 2, y)
3 Got(1, 2, 3)

(b)

1 Got	2 Got
(1, 2, y)	(1, 2, x)
(a, b, c)	(d, e, f)

(c)

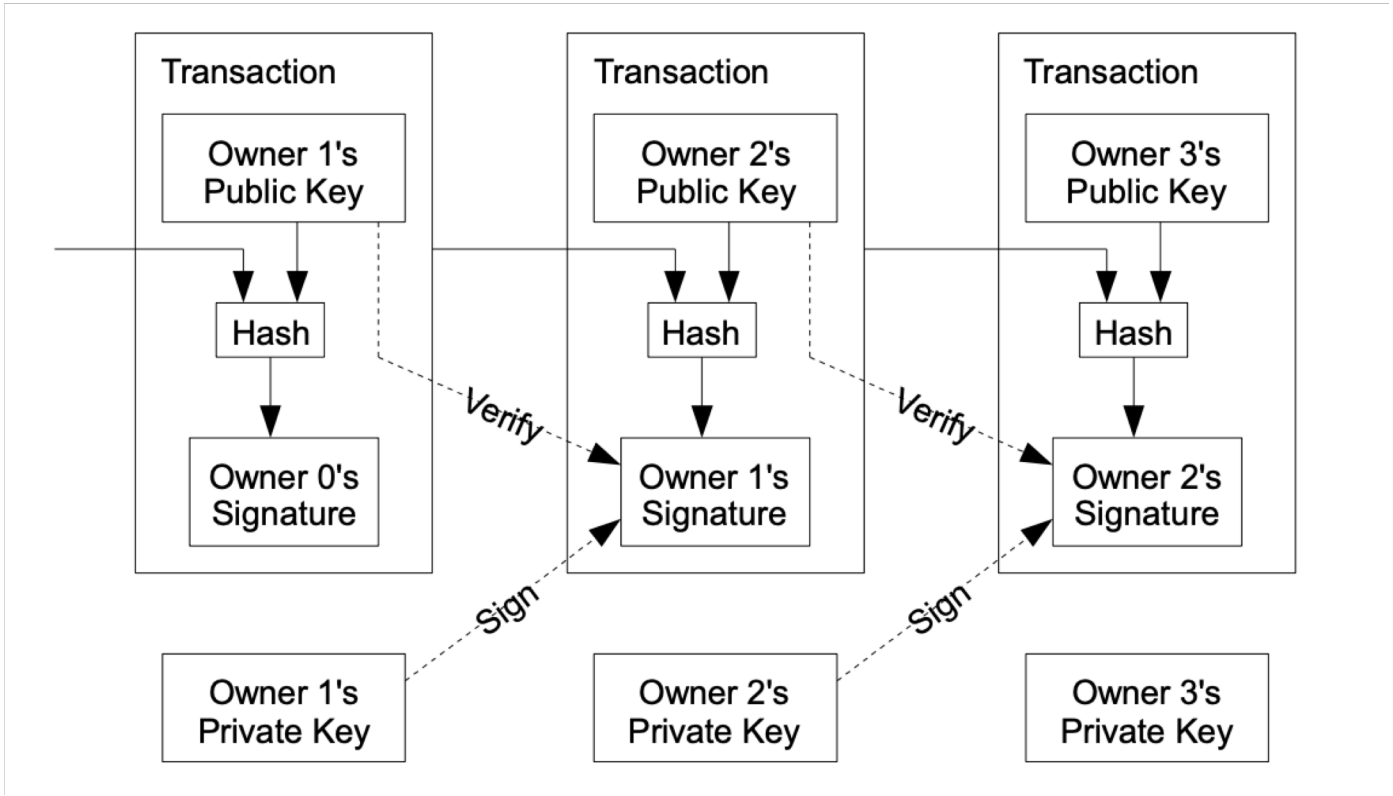
Unable to reach agreement

Bitcoin

- Stores decentralized ledgers in blockchains
- No centralized authority
- Attackers have economic incentives to cause faults
  - e.g., transmit false transactions
- Byzantine fault tolerance (BFT) for bitcoin is needed

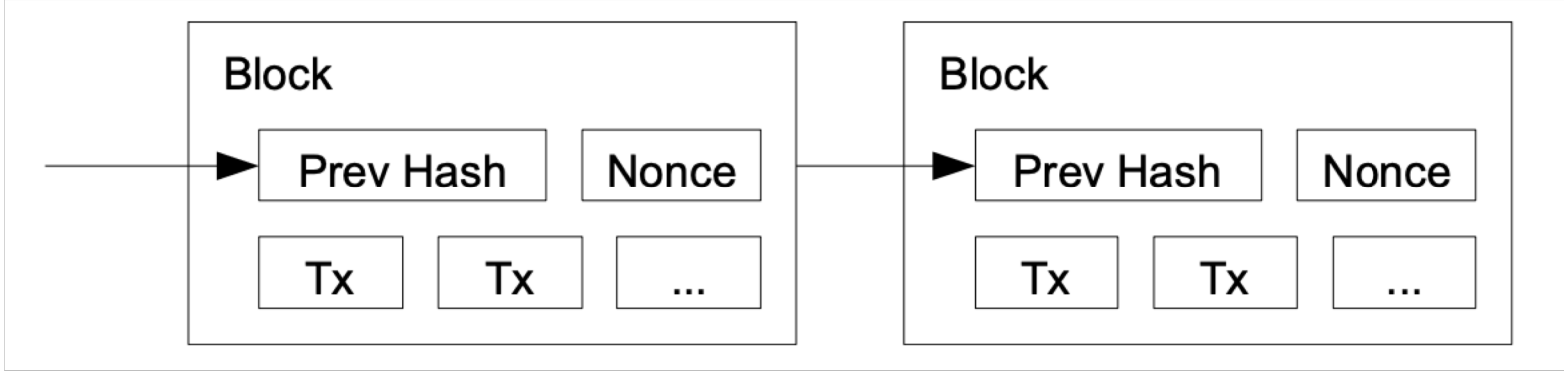
Bitcoin

- A transaction is of the form “send these Bitcoins from address Y to address Z”



Bitcoin blocks

- A transaction is recorded in a block that contains:
  - hash of previous block
  - nonce
  - new transactions to include in the blockchain
  - creation of reward bitcoins (e.g., 12.5 new BTC now)



Bitcoin blockchain

- Every ten minutes, one lucky Bitcoin miner earns a reward for extending the block chain by one block.
- The reward is currently 12.5 BTC

- Mining is the only mechanism for creating new bitcoins.
- The total number of Bitcoins will never exceed 21M.

Proof-of-work (PoW)

- A probabilistic solution to the Byzantine Generals Problem.
- A block contains transactions to be validated and previous hash value.
- Pick a nonce such that:
  - **H(prev hash, nonce, Tx, Tx, …) < E**
  - E is a variable that the system specifies. Basically, this amounts to finding a hash value whose leading bits are zero.
  - The work required is exponential in the number of zero bits required.

Proof-of-work (PoW)

- There is no easy way to find the nonce that can hash to enough leading zero bits, other than:
- Brute-force

- The difficulty, i.e., the number of leading zeros bits required, is chosen so that the time until the first miner wins is about ten minutes, on average.

Overall workflow

- New transactions are broadcast to all nodes.
- Each node collects new transactions into a block.
- Each node works on finding a difficult proof-of-work for its block.
- When a node finds a proof-of-work, it broadcasts the block to all nodes.
- Nodes accept the block only if all transactions in it are valid and not already spent.
- Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

Transaction confirmation

- A transaction is said to have received ***k*** confirmations if:
  - it has been published in a block that has been added to the blockchain, and
  - ***k-1*** more blocks have also been added.
- Currently, k is chosen to be 6

What if an attacker wants to modify an old transaction?

- ***p*** = probability an honest node finds the next block
- ***q*** = probability the attacker finds the next block
- q_z = probability the attacker will ever catch up from z blocks behind

$$q_z = \begin{cases} 1 & \text{if } p \leq q \\ (q/p)^z & \text{if } p > q \end{cases}$$

The attackers are unable to catch up

- As long as $p > q$, i.e., the honest nodes control **more than half** the computation power,
- the probability the attacker can catch up from behand drops exponentially as the number of blocks the attacker has to catch up with increases.

Example results

```
q=0.1
z=0     P=1.0000000
z=1     P=0.2045873
z=2     P=0.0509779
z=3     P=0.0131722
z=4     P=0.0034552
z=5     P=0.0009137
z=6     P=0.0002428
z=7     P=0.0000647
z=8     P=0.0000173
z=9     P=0.0000046
z=10    P=0.0000012
```

```
q=0.3
z=0     P=1.0000000
z=5     P=0.1773523
z=10    P=0.0416605
z=15    P=0.0101008
z=20    P=0.0024804
z=25    P=0.0006132
z=30    P=0.0001522
z=35    P=0.0000379
z=40    P=0.0000095
z=45    P=0.0000024
z=50    P=0.0000006
```

```
P < 0.001
q=0.10    z=5
q=0.15    z=8
q=0.20    z=11
q=0.25    z=15
q=0.30    z=24
q=0.35    z=41
q=0.40    z=89
q=0.45    z=340
```

Reading

- Chapter 8 of Tbook
- Bitcoin whitepaper: https://bitcoin.org/bitcoin.pdf