Fault Tolerant Systems - 2

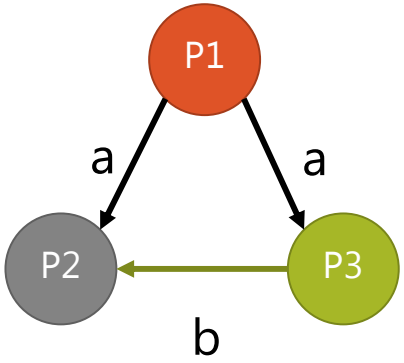
Slide set 7
Distributed Systems

Graduate Level

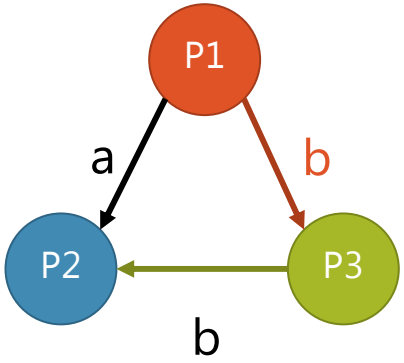
K. N. Toosi Institute of Technology
Dr. H. Khanmirza
h.khanmirza@kntu.ac.ir

Byzantine (Arbitrary) Fault Tolerance (BFT)

- System face with non-crash faults
- It works but may send faulty replies



P_2 reports wrong value



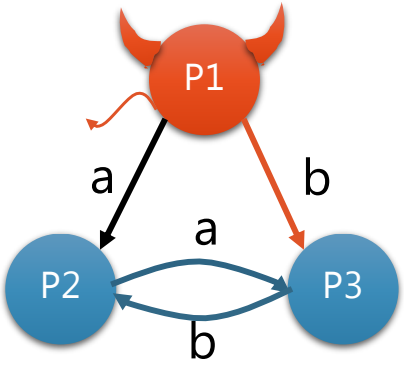
P_1 Sends different values to different processes

Real world examples

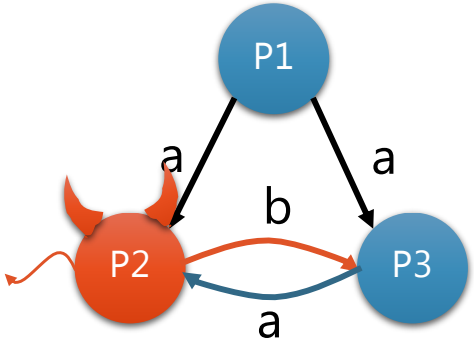
- Buggy servers, that compute incorrectly rather than stopping
- Servers that don't follow the protocol due to some hardware failure
- Servers that have been modified by an attacker

BFT

- K-faulty nodes can't be detected by 3k non-faulty processes
- It needs 2k+1 non-faulty processes; or, totally 3K+1 processes



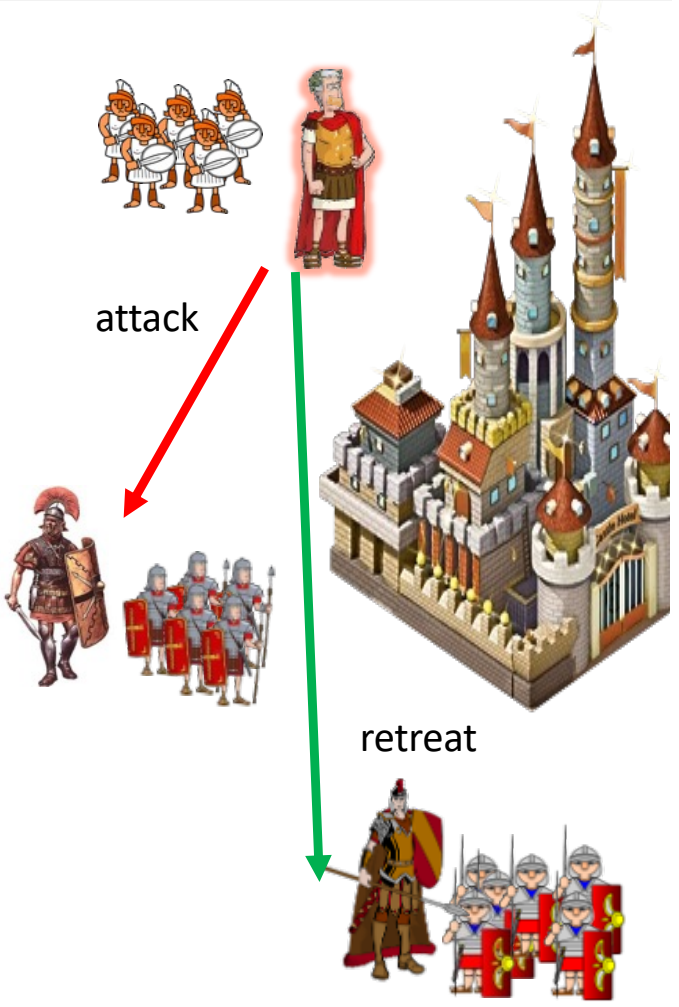
P2 and P3 can't draw a conclusion which value is correct



P1 and P3 can't draw a conclusion which value is correct

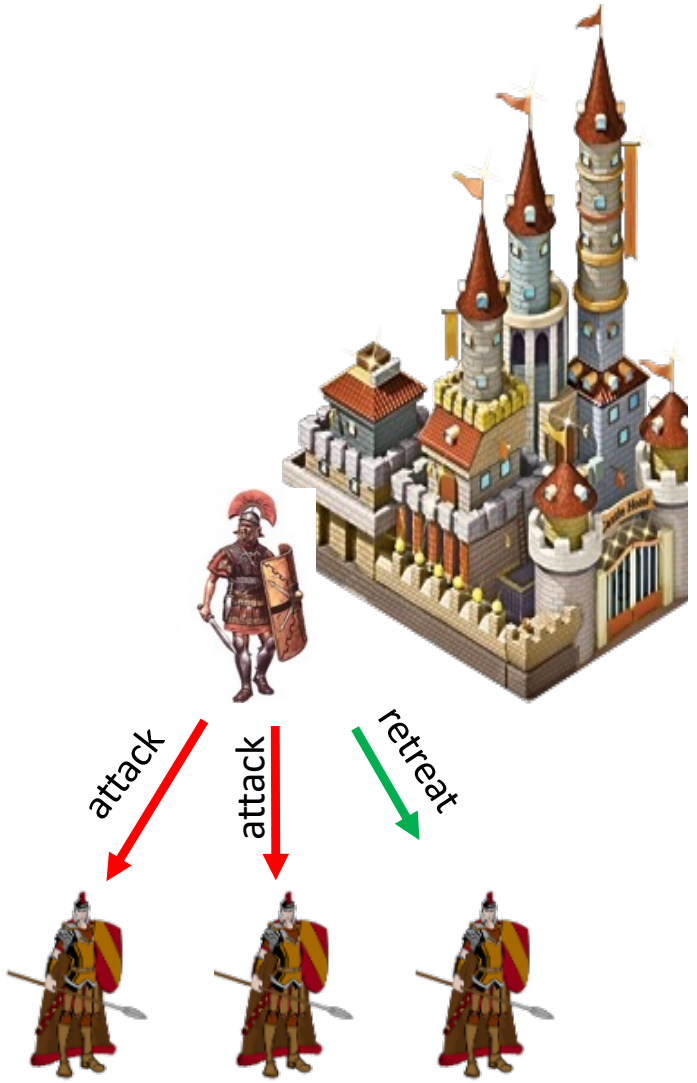
Byzantine Generals Problem V1

- First proposed by Lamport in 1982
- A group of Byzantine generals encircled a city
- They must agree to attack or retreat, otherwise their army will be slaughtered
- Some generals are traitorous
  - Traitorous generals say different opinions to break the union
- They can send their decision in messages carried by messengers



Byzantine Generals Problem V2

- A commanding general encircled a city
- He sends his order to his lieutenants
- Loyal lieutenants obey the order
- All lieutenants must agree to attack or retreat, otherwise their division will be slaughtered
- The general or several lieutenants can be traitorous.
- They communicate with messengers



Byzantine Generals Problem

- Two variations correspond to
  - V1: Flat group = Replicated-Write protocol
  - V2: Hierarchical group = Primary-Backup protocol
- From now we study algorithms for the second version of the problem
  - Solutions of both versions are convertible for each other

Byzantine Generals Problem

- Formal Definition
  - A process group has a primary member, **P** (=commander)
  - The remaining *n-1* members are backup $\{B_1, \ldots B_{n-1}\}$ (= lieutenants)
  - A client sends a value v $\in \{T, F\}$ to primary (={attack , retreat})
  - Messages may be lost, but this can be detected
  - Messages cannot be corrupted without being detected

Byzantine Generals Problem

- To achieve Byzantine Agreement two conditions must be satisfied:

- BA1: Every nonfaulty backup process stores the same value
  - Maybe a value different with what client had sent originally.

- BA2: If the primary is nonfaulty then every nonfaulty backup process stores exactly what the primary had sent.
  - If the primary is nonfaulty, satisfying BA2 implies BA1

Byzantine Generals Problem

- Synchronous Solutions
  - Oral Message Algorithm
  - Signed Message Algorithm
- What does it mean: Synchronous solution?

OM (Oral Message) Solution

- Assumptions:
  - Every message that is sent is delivered correctly
    - Messages are not lost
  - The receiver of a message knows who sent it
  - The absence of a message can be detected.
  - Original message may be forged 🤫.

  - We assume all group members can talk with each other
    - They are connected by a complete graph

- Goal
  - All loyal lieutenants agree on the same order
  - If the commander is loyal, then every loyal lieutenant obeys the order he sends

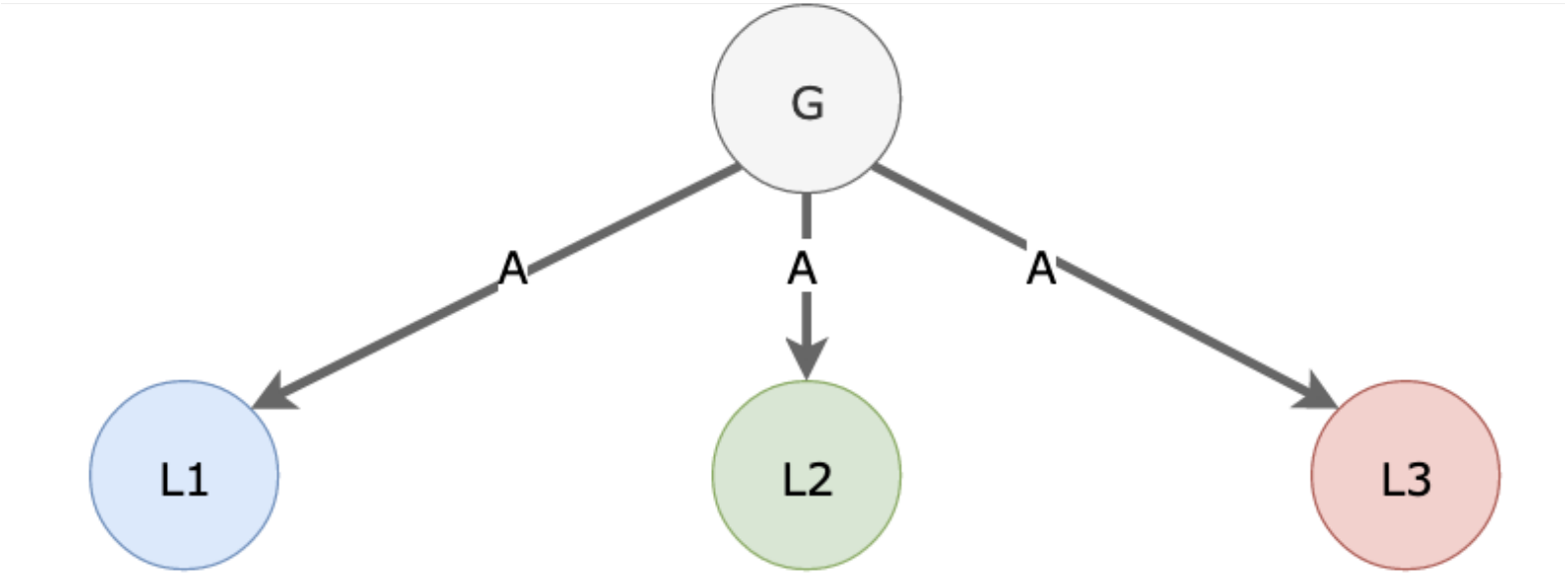
OM Solution

- OM(n, m)
  - n: 1 commanding general, n-1 lieutenants
    - i=0: commander node, i=[1,n-1]: lieutenant nodes
  - m traitors

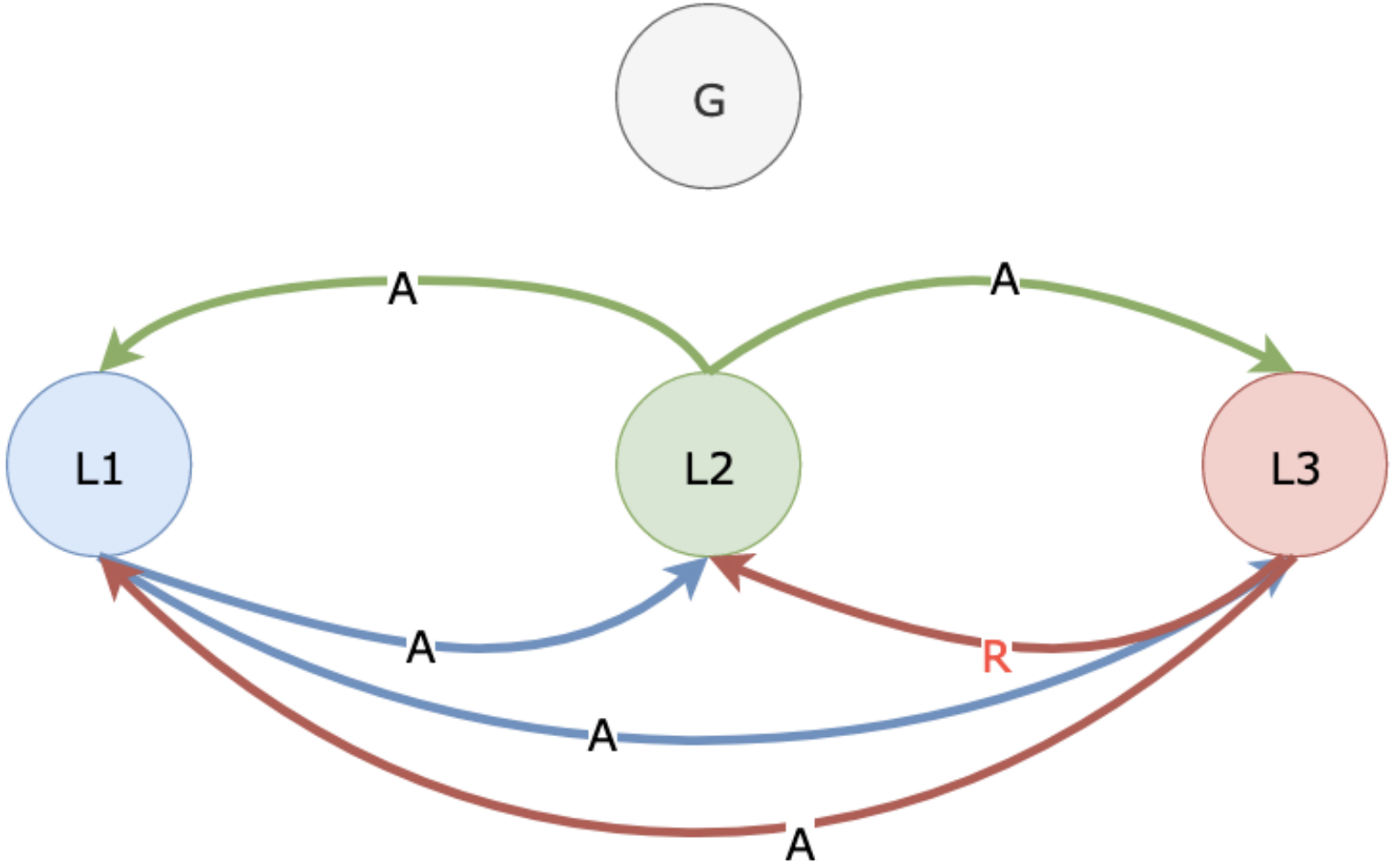
- Phase 0: The commanding general sends his order to all lieutenants

- Phase 1: Each lieutenant sends its value to n-2 lieutenants
  - Correct processes send the same (correct) value to all.
  - Faulty processes may send
    - Different values if desired
    - nothing

OM Solution



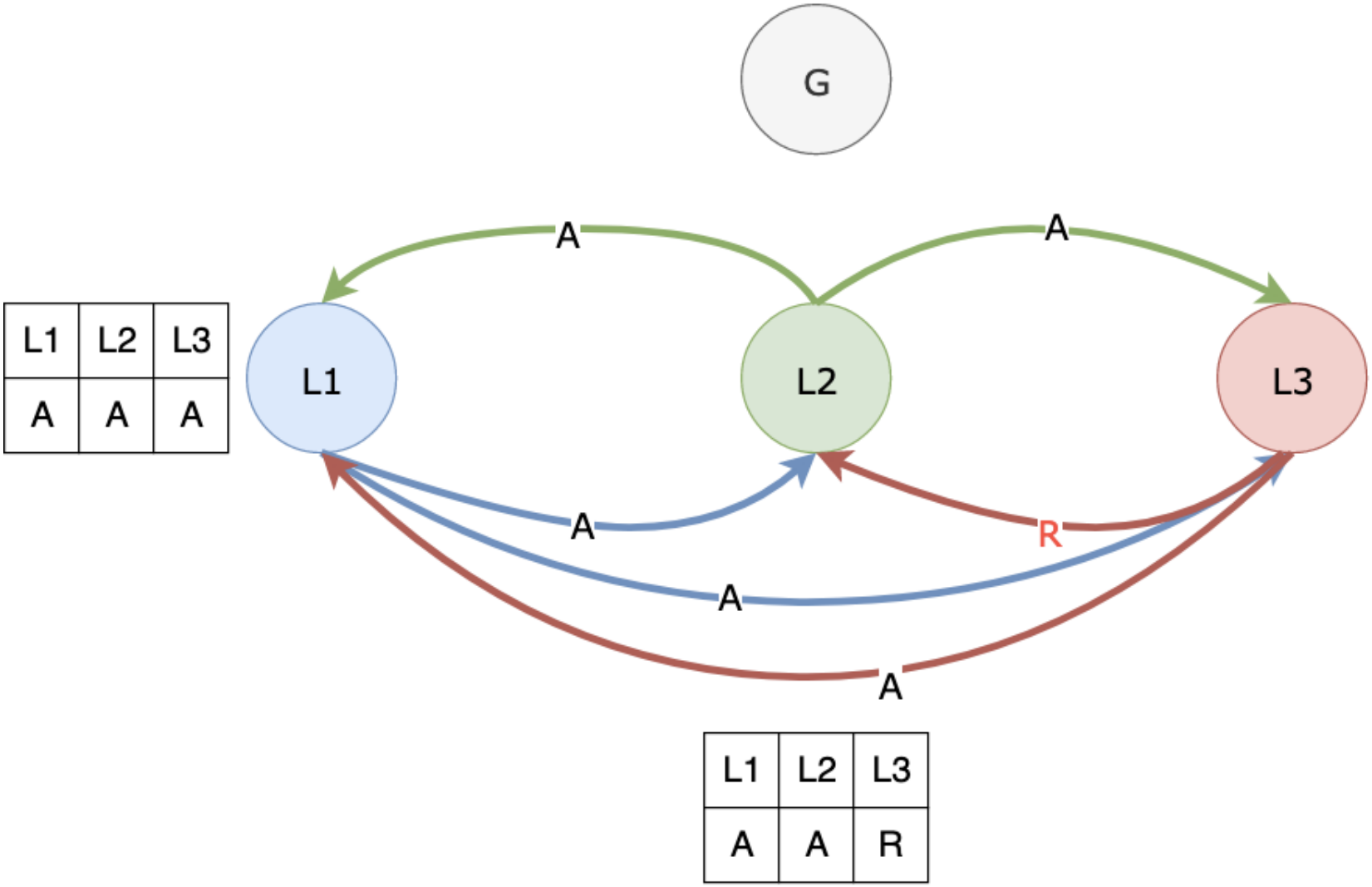
OM Solution



OM Solution

- Phase 2: Each lieutenant builds a vector V_i where
  - $\forall\, j \neq i, V_i[j] = \begin{cases} V_j[j] \\ v_{default} \end{cases}$
  - If the lieutenant$_i$ has received any message from lieutenant$_j$ it puts in the jth room of the vector
  - If not received anything, puts the default action in the jth room
  - If lieutenant received a new value broadcasts it to other (n-2) lieutenants

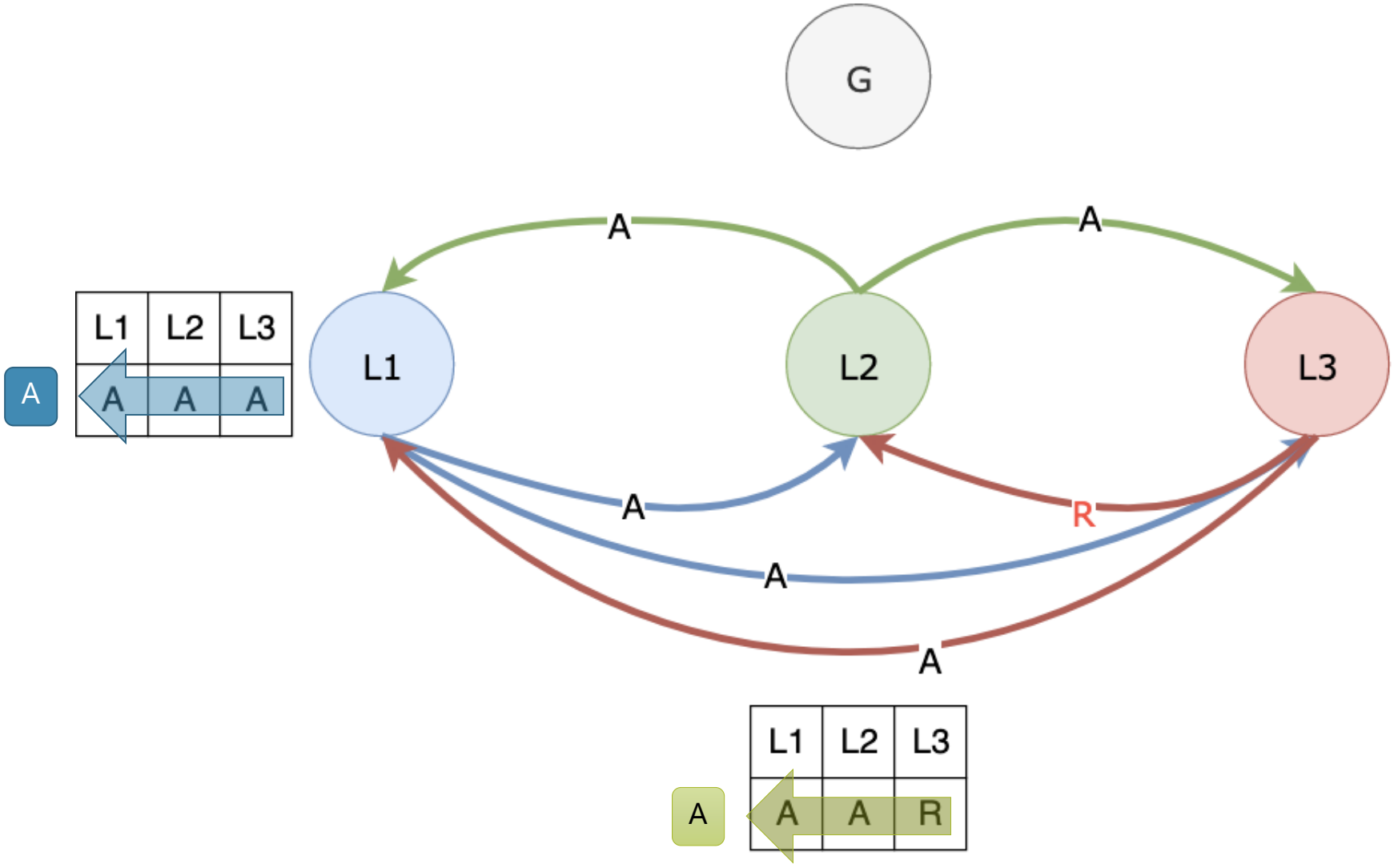
OM Solution



OM Solution

- Phase 3: choose the final action with $v_i = majority(V_i)$

OM Solution



OM Solution

- Have two variations
  - Consensus problem:
    - Every process have an initial value, all non-faulty processes must agree on a value (value have arbitrary type)
  - Interactive Consistency problem
    - Each process has an initial value, and all the correct processes must agree upon a set of values, with one value for each process

SM Solution

- Signed Message Algorithm (SM)
- Assumptions:
  1. Every message that is sent is delivered correctly
  2. The receiver of a message knows who sent it
  3. The absence of a message can be detected.
  4. A loyal general's signature cannot be forged
     - Any alteration of the contents of his signed message can be detected.
     - Anyone can verify the authenticity of a general's signature
- Goal
  - All loyal lieutenants obey the same order
  - If the commander is loyal, then every loyal lieutenant obeys the order he sends

SM Solution

- V_i is a set of orders
  - V_i can be {}, {attack}, {retreat}, {attack , retreat}
- $v: i$ means
  - ith Lieutenant has signed the message v
- $v: i: j$
  - $v \in V_i$
  - ith lieutenant has signed the message v
  - jth lieutenant has signed the message $v: i$

SM Solution

- Lieutenants decide based on choice function
- Choice function
  - $V_i = \{v\}$ ➔ $choice(V_i) = v$
  - $V_i = \{\}$ ➔ $choice(V_i) = v_{default}$
  - $V_i = \{v_1, v_2\}$ ➔ $choice(V_i) = v_{default}$

SM Solution

- SM (n, m):
  - n: one commanding general + n-1 lieutenants
    - i=0: commander node, i=[1,n-1]: lieutenant nodes
  - m traitors

- Phase 0: The commanding general signs and sends his order to all lieutenants $(v: 0)$

SM Solution

- Phase 1: Lieutenant$_i$ receives general order
  - If $(v: 0)$ is the first order
    - Set $V_i = \{v\}$
    - Sign & Send $(v: 0: i)$ to every other lieutenant
  - If a message in the form $(v: 0: j_1: \ldots : j_k$ & $v \notin V_i)$ is received
    - $V_i = V_i \cup v$
    - If $k < m$, send $(v: 0: j_1: \ldots : j_k: i)$ to other lieutenants not in $j_1, \ldots, j_k$.

SM Solution

- Phase 2:
  - Decide based on $choice(V_i)$
    - If $choice = \{attack, retreat\}$
      - Commander is traitor!
      - Do $v_{default}$
    - The traitor lieutenant can not change the message, he may send empty message or the original message

BFT

- Byzantine fault tolerance was for long more or less an exotic topic

- It turned out that combining safety, liveness, and practical performance was difficult to achieve

- Castro M. and Liskov B. Practical Byzantine Fault Tolerance, SOSP 1999
  - High-performance implementation
    - processing thousands of requests per second with sub-millisecond increases in latency
  - Partially-synchronous system

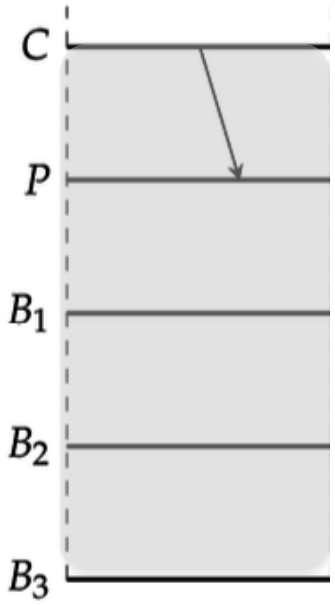
PBFT (Practical Byzantine Failure Tolerance)

- Assumptions
  - A faulty replica server may exhibit arbitrary behavior.
  - Messages may be lost, delayed, and received out of order.
  - However, a message's sender is assumed to be identifiable or messages are signed.
  - PBFT adopts a primary-backup model with a total of $3k + 1$ replica servers
  - The primary can still lie.
    - Send different sequence number for the same operation to different replicas
    - Use a duplicate sequence number for operation

PBFT

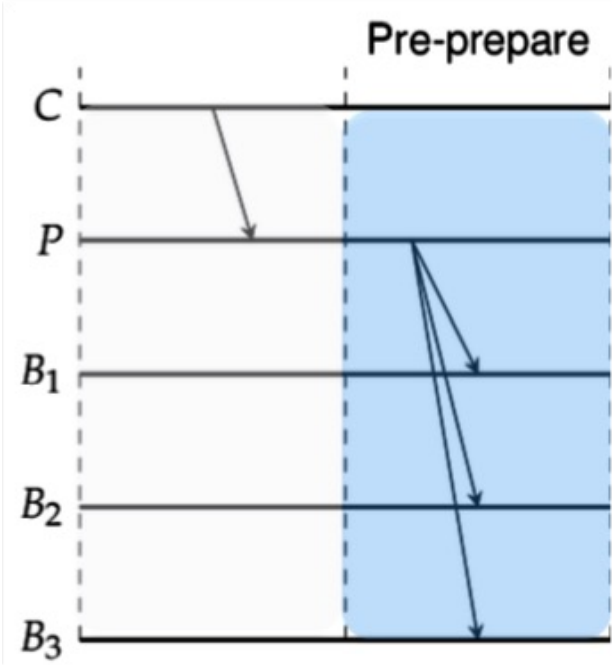
- Safety:
  - Client always receives a correct answer
- Liveness (Termination):
  - Every correct node eventually chooses a value.
  - If message delays and response times are bounded (synchronous system) PBFT provides liveness
  - PBFT assumes a partially synchronous model, in which unbounded delays are an exception, for example caused by an attack.

PBFT



Client sends a request to Primary

PBFT

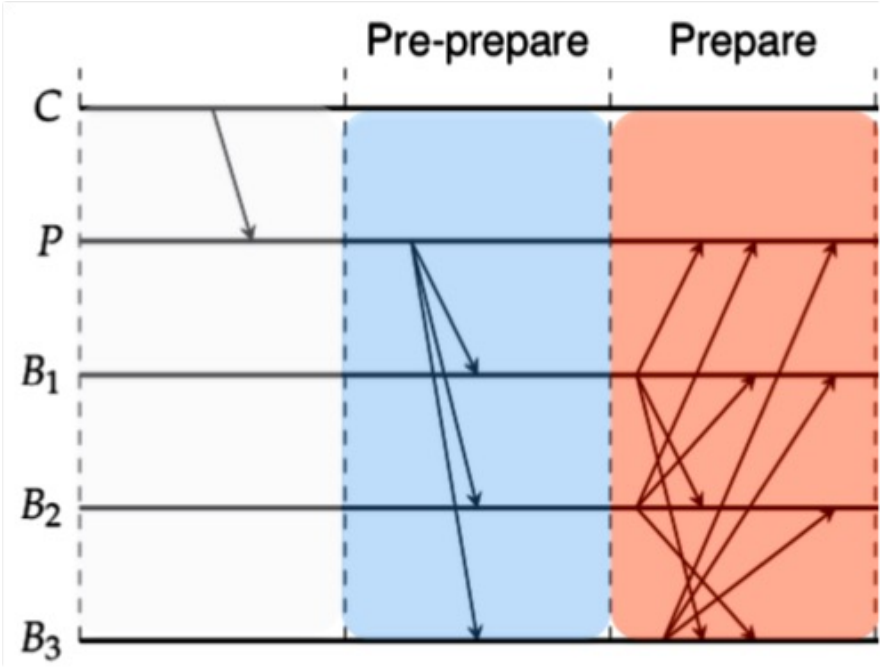


Primary sends **(t,v,o)** to backups:
t: timestamp
v: view is a number, simply the ID of the primary replica
o: operation

PBFT

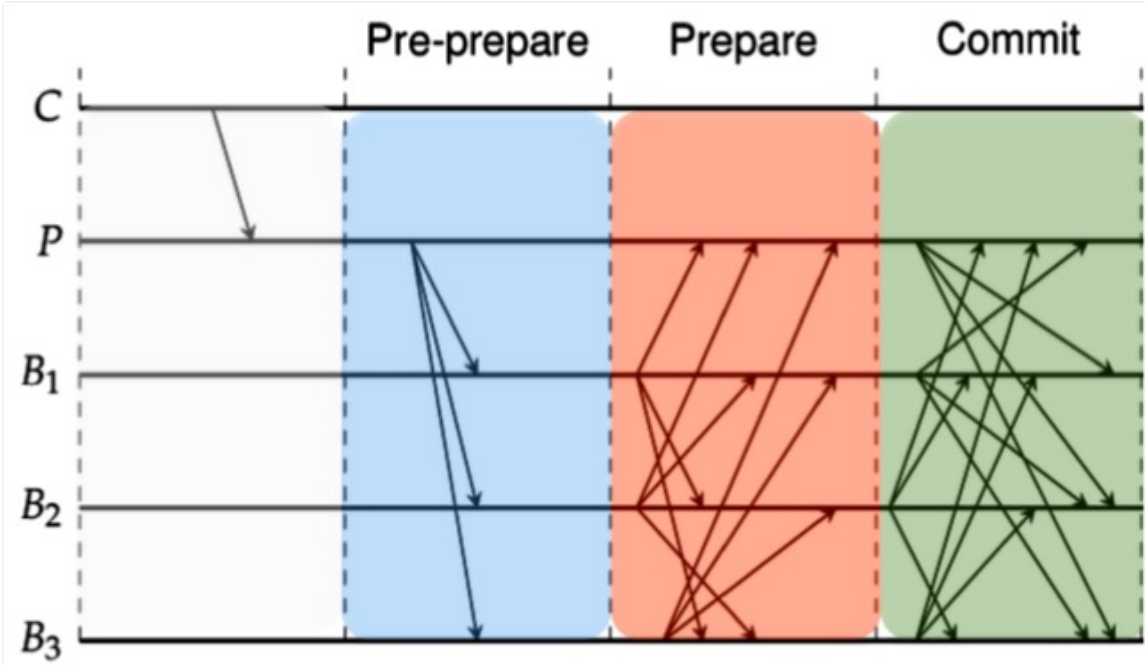
- A non-faulty backup accepts pre-prepare message if
  - It is in v
  - Has never accepted an operation with timestamp t in v before.
  - Sends prepare message (i,t,v,o) to all others (including primary)
    - i: id of the node

PBFT



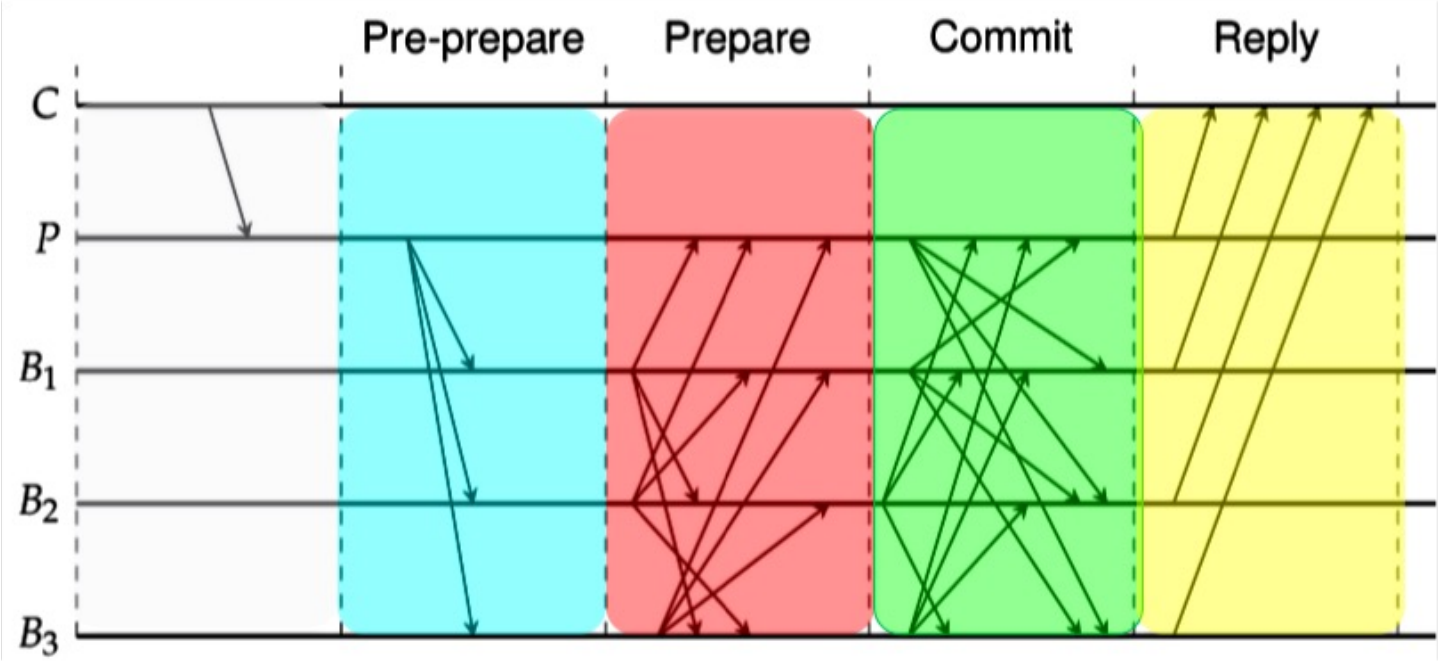
If a backup receives 2k prepare(i,t,v,o) messages matching its pre-prepare message, there is consensus among nonfaulty replicas → prepare certificate

PBFT



- Send commit(i,t,v,o) to all others
- Collecting 2k matching commits, leading to commit certificate causes replica to execute o

PBFT



- After executing o, send response to the client.
- Client mark a message as answer if receive k+1 similar messages.

PBFT

- What if the primary get failed?
- We must ensure requests being processed at failure time, to be executed exactly once by nonfaulty replicas.
- PBFT deterministically selects another node as primary.
  - Simply primary is the replica with ID (v+1 mod n)

PBFT

- If a backup detects (by timeout) primary failure
- Stops accepting messages (except view messages).
- Broadcasts view-change(v+1, **P**) message.
  - **P** contains valid prepare certificates, operations with a consensus

PBFT

- New Primary (view (v+1))
  - Waits for 2k+1 view-change messages
  - Broadcasts message new-view(v+1,**X**,**O**)
  - X contains view-change messages (view change certificate) - the digest of all messages proves that primary have all
  - O contains pre-prepare messages
    - Primary computes min and max timestamps of prepare messages in X
    - For each $t \in [t_{\min} \quad , t_{\max} \quad]$
      - If there is a message in **P** create message pre-prepare(t, v+1, o)
      - Otherwise create message pre-prepare(t, v+1, null)

PBFT

- In fact, new primary restarts all non-committed operations
- Backups switch to view v+1 after receiving this message

PBFT

- PRE-PREPARE picks order of requests
- PREPARE ensures order within views
- COMMIT ensures order across views

Discussion on Consensus Protocols

Consensus

- Reaching an agreement is inevitable in some scenarios:
  - Electing a coordinator
  - Deciding whether or not to commit a transaction
  - Dividing tasks among workers
  - Synchronization

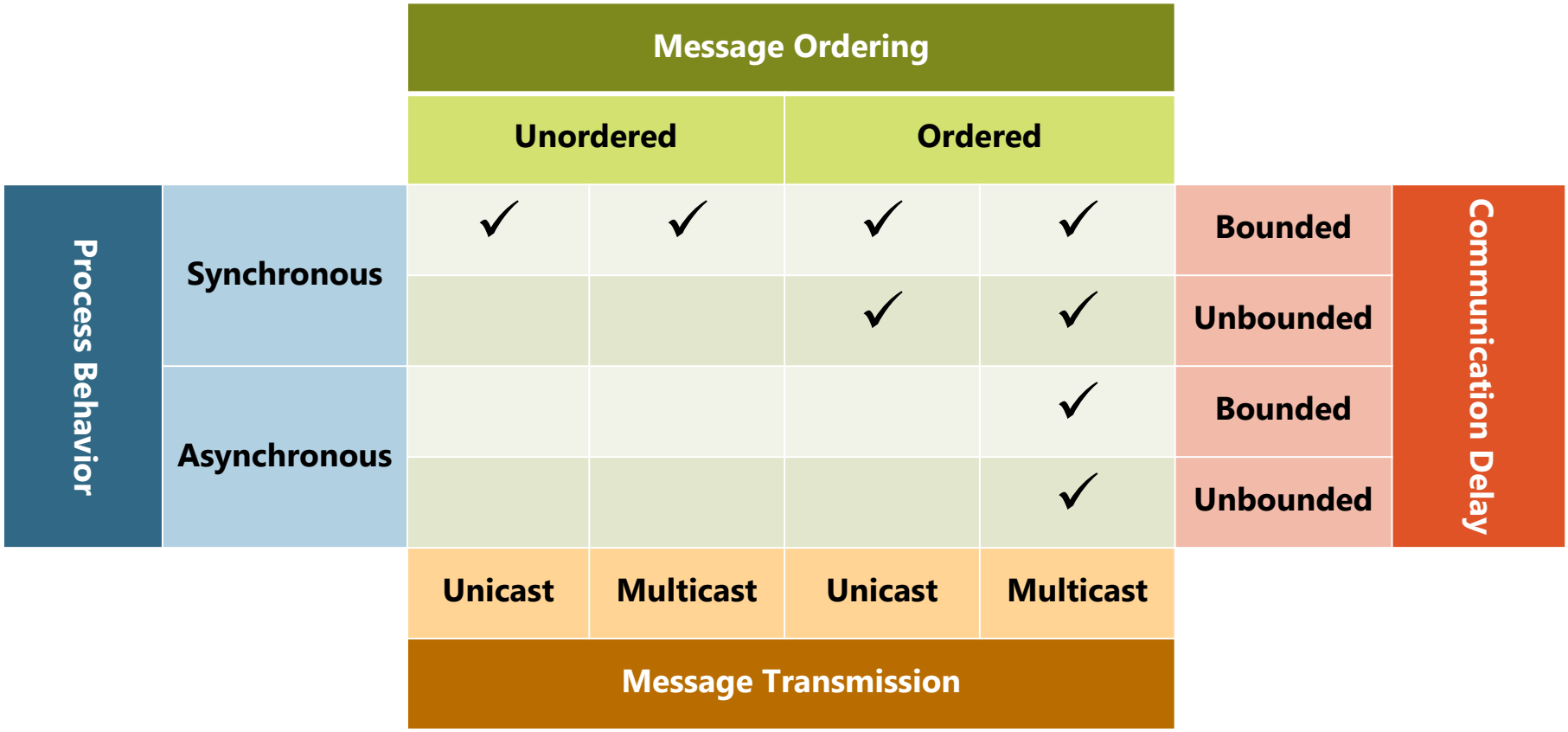
- If communication and processes are:
  - Perfect: Reaching an agreement is often straightforward
  - Not perfect: Reaching an agreement is typically not easy, have noticeable performance penalty

Consensus

- Goal of distributed consensus algorithms:
  - All non-faulty processes reach consensus on some operation
  - Reaching that consensus within a finite number of steps

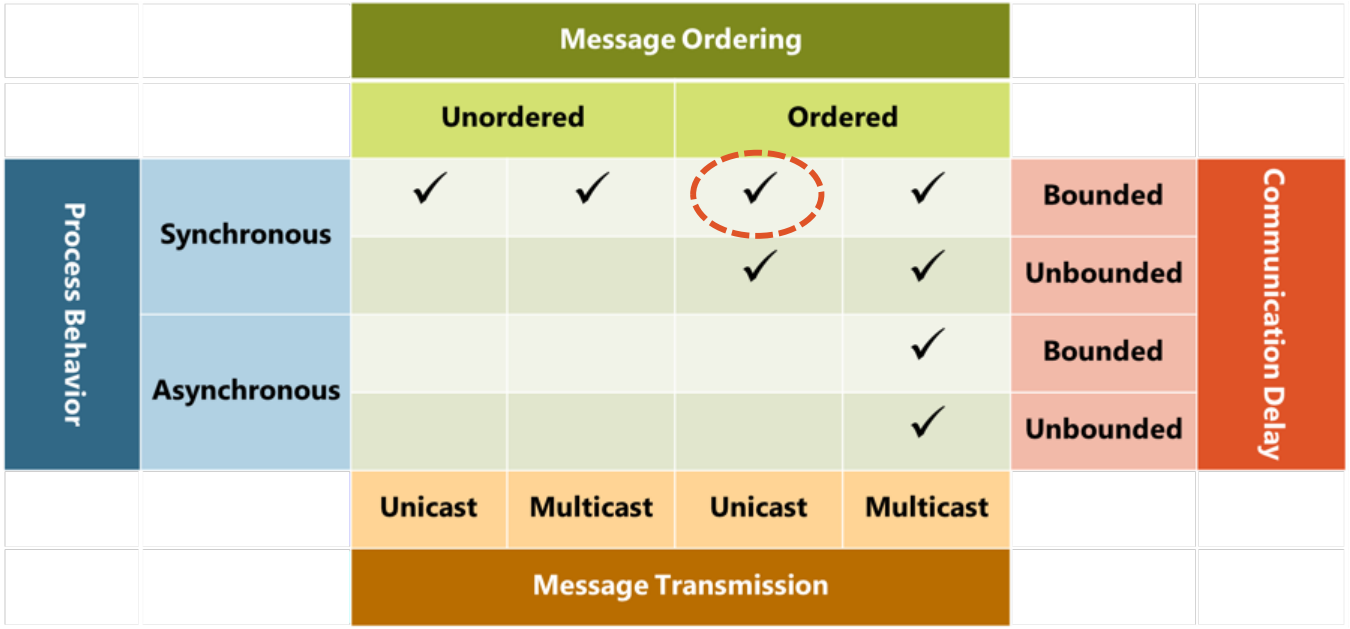
- Different assumptions about underlying system necessitate different solutions
  - Synchronous vs. asynchronous systems
  - Bounded vs. unbounded communication delays
  - Ordered vs. unordered message delivery (having global time)
  - Unicasting vs. multicasting message transmissions

Consensus


Message Ordering
Unordered
Ordered
Process Behavior
Synchronous
Asynchronous
✓
✓
✓
✓
✓
✓
✓
✓
Bounded
Unbounded
Bounded
Unbounded
Communication Delay
Unicast
Multicast
Unicast
Multicast
Message Transmission

Consensus

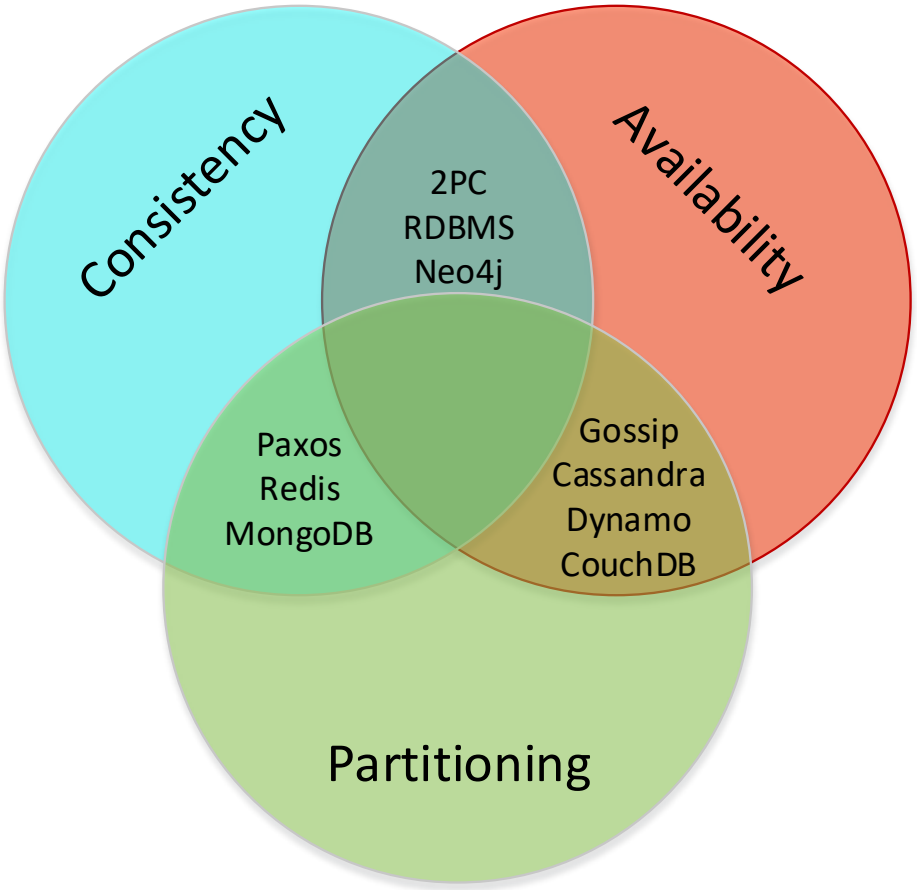
- Byzantine Agreement
  - Algorithm executes in rounds
  - Every message that is sent is delivered correctly
  - The receiver knows who sent the message
  - Message delivery time is bounded
  - Messages are unicast



Consensus

- **CAP** Theorem: Any networked system providing shared data can provide only two of the following three properties
  - **C**: consistency, by which a shared and replicated data item appears as a single, up-to-date copy
  - **A**: availability, by which updates will always be eventually executed
    - Eventually get a (correct) response to every request issued by a client
  - **P**: Tolerant to the partitioning of process group (e.g., because of a failing network).
- In a network subject to communication failures, it is impossible to realize an atomic read/write shared memory that guarantees a response to every request

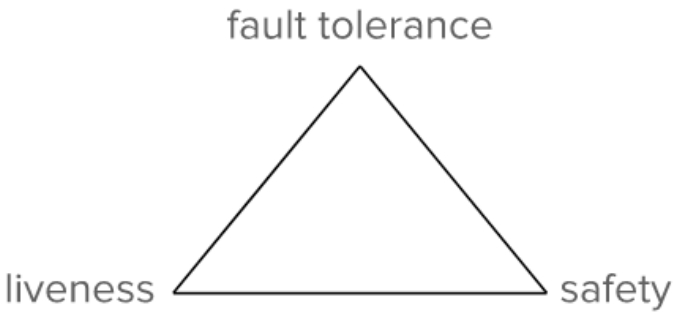
Consensus

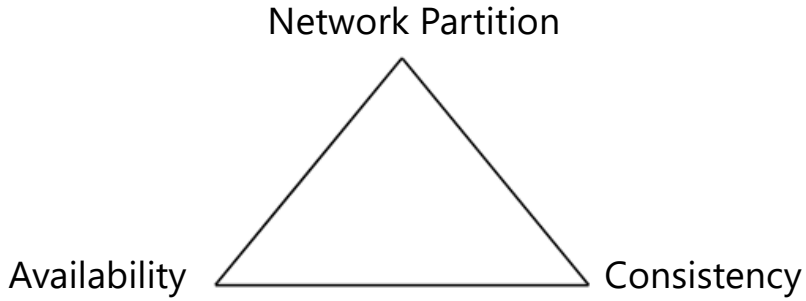


Prof. Eric Brewer

FLP vs CAP

- CAP has stronger conditions than FLP
  - In CAP, nodes are partitioned. A CAP solution requires that any live node be able to correctly serve requests, even if it has not received any messages.
  - A partitioned node in FLP does *not* have to achieve consensus, since it is considered failed, but the same node in CAP must keep up with the activity of the rest of the system.





FLP vs CAP

- The FLP states that in an asynchronous network where messages may be delayed but not lost, there is no consensus algorithm that is guaranteed to terminate in every execution for all starting conditions, if at least one node may fail-stop.

- The CAP states that in an asynchronous network where messages may be lost, it is impossible to implement a sequentially consistent atomic read / write register that responds eventually to every request under every pattern of message loss.

Fighting with Impossibility

- Exactly deciding on how to proceed is application dependent:
  - Having duplicate keys in a database can easily be fixed, implying that we should tolerate an inconsistency.
  - Duplicate transfers of large sums of money may not be easily fixed, that means we should decide to tolerate lower availability, but be consistent.

- One can argue that the CAP theorem essentially moves designers of distributed systems from theoretical solutions to engineering solutions.

The End!