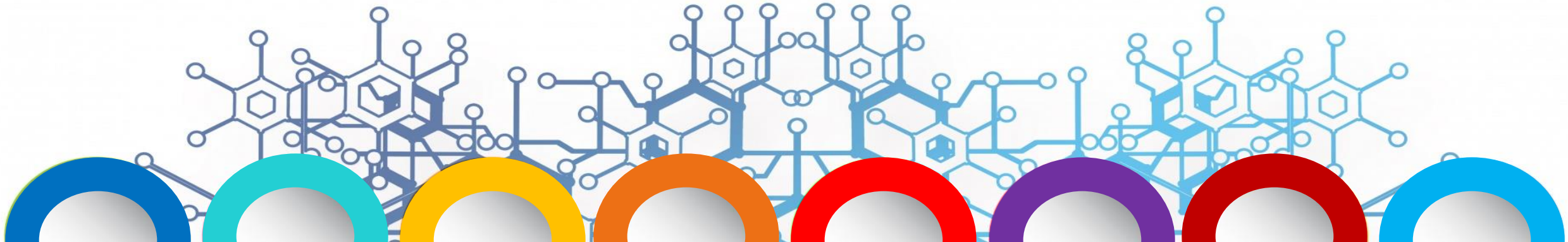
Fault-Tolerance



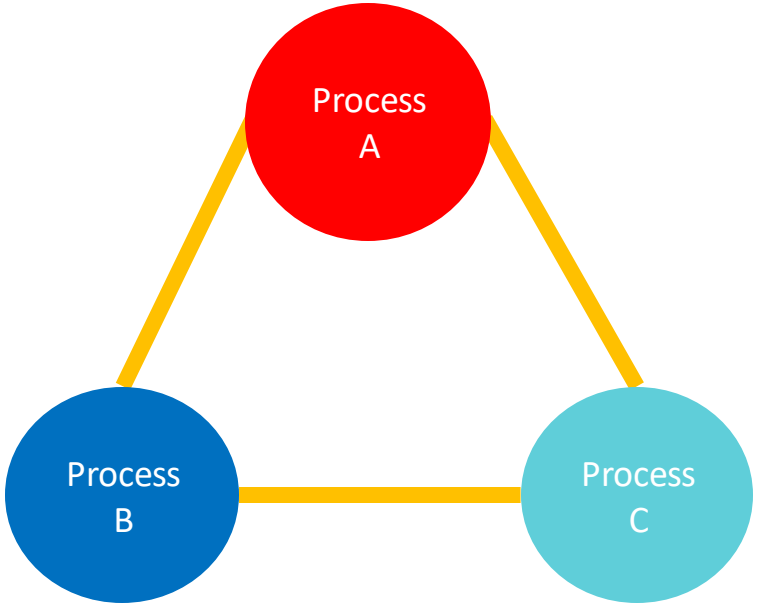
Fault Tolerance

- **Detecting** Failure
  - Every component (processors, storage, network, software) can and will fail
- **Preventing** Failure
  - Preventing element failure to cause system failure
- **Recovering** From Failure
  - High Availability
  - Logging
  - Checkpointing
  - Undo & Redo

Failure Models

- Process Failure Model
  - Fail-stop (stop execution - other processes learn the processes has failed)
  - Crash (stop execution - other processes do not learn the processes has failed)
  - Receive Omission (intermittent receiving of messages)
  - Send Omission (intermittent sending)
  - Byzantine or Malicious Failure (Arbitrary behavior)
    - With Authentication (received message claim check)
    - Without Authentication

- Communication Failure Models
  - Crash Failure (link stop)
  - Omission Failures (missing messages)
  - Byzantine Failures (link with arbitrary behavior)



***Reference**: Distributed Computing: Principles, Algorithms, and Systems, Kshemkalyani & Singhal (2008)*

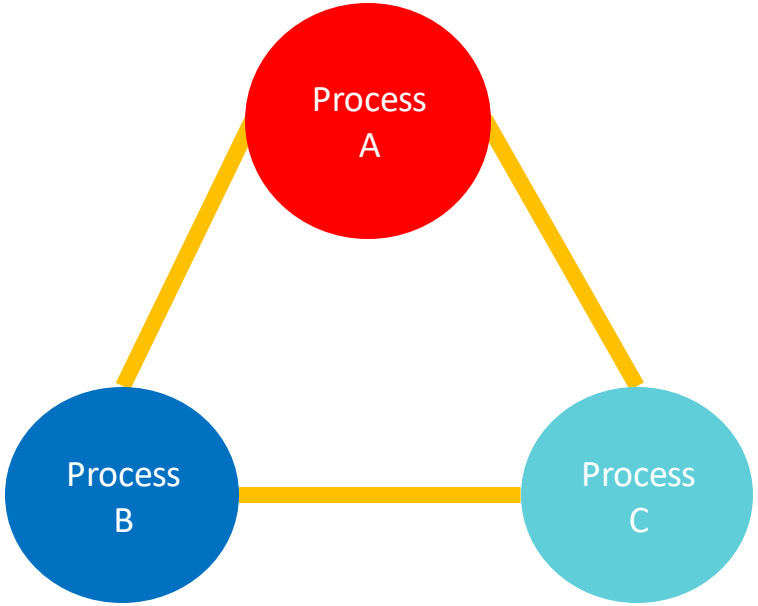
Failures

- Process Failure Model
  - Fail-stop
  - Crash
  - Receive Omission
  - Send Omission

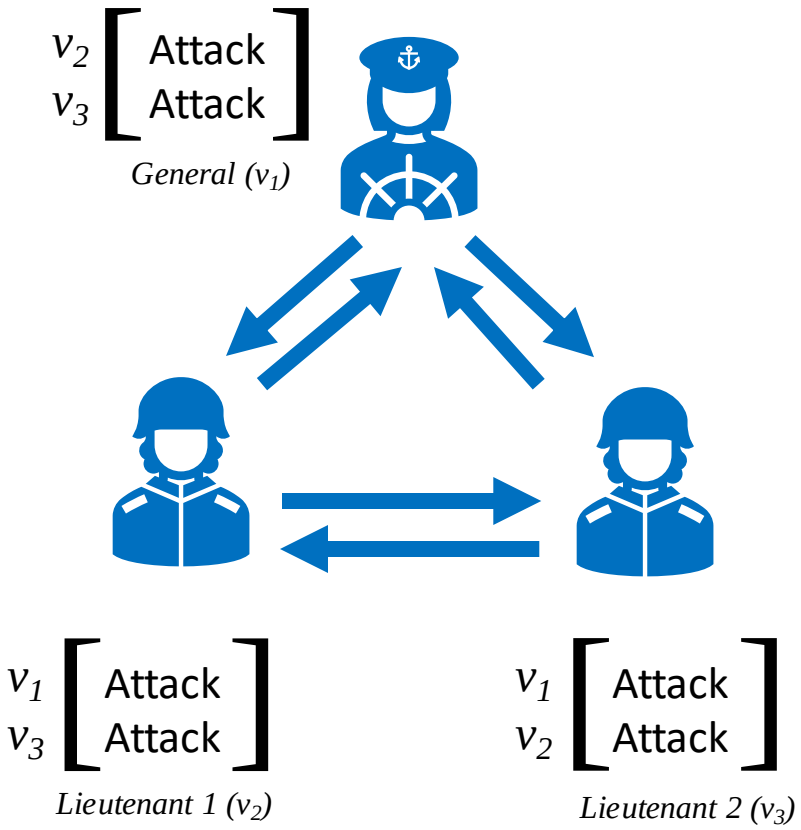
  } **Non-Byzantine**
  Paxos, Raft,…

  - **Byzantine or Malicious Failure**

    Paxos-PBT, PoW, PoS, PBFT,…



Byzantine Generals Problems



- Oral Messages:
  Contents in control of the sender
  - Every message that is sent is delivered correctly
  - The receiver of a message knows who sent it
  - The absence of a message can be detected

Condition One:

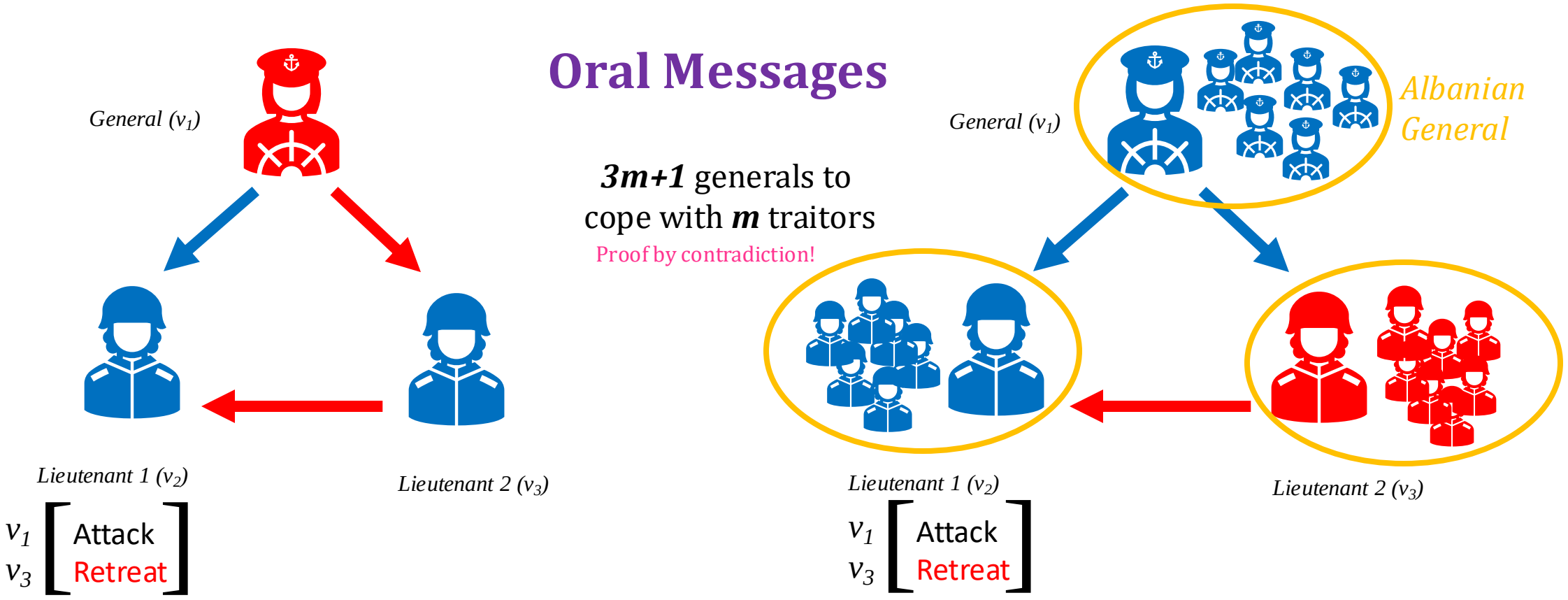
All loyal lieutenants obey the same order

Condition Two:

Commanding general is loyal → every loyal lieutenant obeys their order

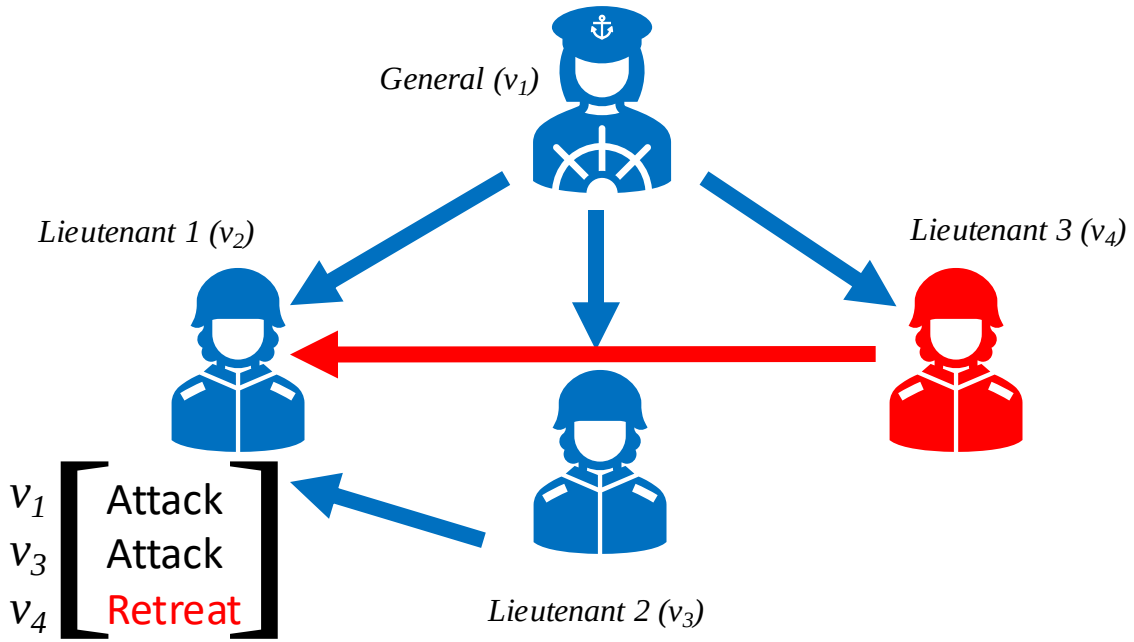
Reference: *Leslie Lamport, Robert Shostak, and Marshall Pease, The Byzantine Generals Problem, 1982*

Byzantine Generals Problems



Reference: *Leslie Lamport, Robert Shostak, and Marshall Pease, The Byzantine Generals Problem, 1982*

BFT: Byzantine Fault Tolerance



Algorithm OM(0):

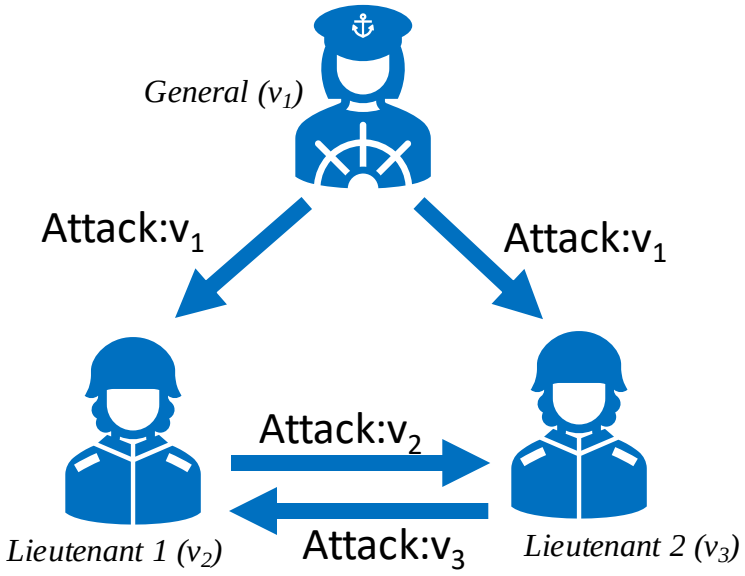
- *Commander Value → Every Lieutenant*
- *Lieutenant:*
  - *Receives: Value*
  - *No value: Retreat*

Algorithm OM(m), m>0:

- *Commander Value → Every Lieutenant*
- *Lieutenant:*
  - *Receives: Value*
  - *No value: Retreat*
  - ***OM(m-1) to each of n-2 lieutenants***
- ***Value Majority on n-1 values at every lieutenant***

Reference: *Leslie Lamport, Robert Shostak, and Marshall Pease, The Byzantine Generals Problem, 1982*

Byzantine Generals Problems



What about signed messages?

- Signed Messages:
  - Loyal general's signature cannot be forged
  - Anyone can verify authenticity of a signature

Where do we see this?

Reference: *Leslie Lamport, Robert Shostak, and Marshall Pease, The Byzantine Generals Problem, 1982*

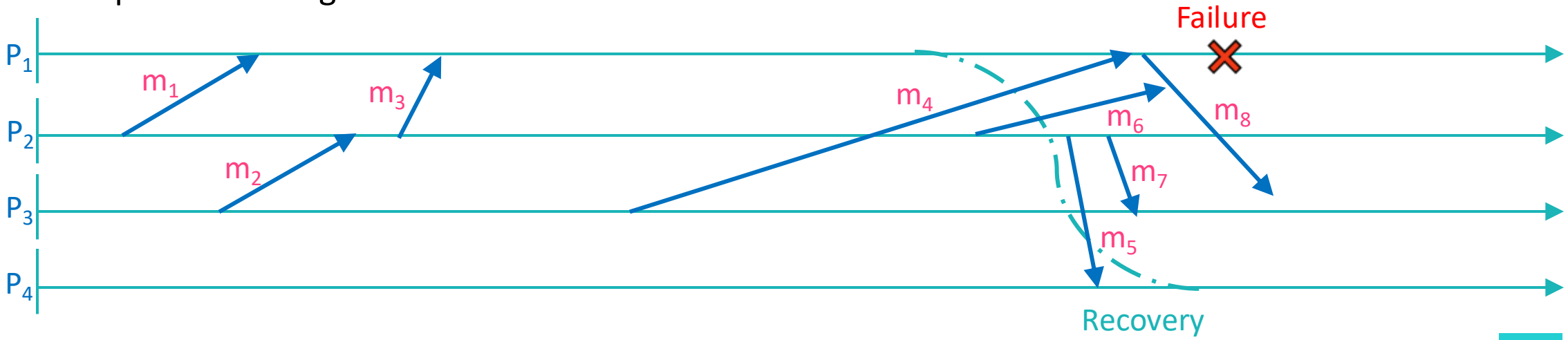
Failure Detection

- Detection: Accuracy, Completeness
  - Strong accuracy: **Correct** processes are never suspected by any correct process.
  - Weak accuracy: **Some** correct process is never suspected by any correct process.
    - [Eventual]: We require the accuracy property to be eventually satisfied (Not at all times)
  - Strong completeness: Eventually every process that crashes is permanently suspected by **every** correct process.
  - Weak completeness: Eventually every process that crashes is permanently suspected by **some** correct process.
- Detectors
  - Perfect Detector (strong accuracy, strong completeness)
  - Eventually Perfect Detector (eventual strong accuracy, strong completeness)
  - Strong Failure Detector (weak accuracy, strong completeness)
  - Eventually Strong Failure Detector (eventual weak accuracy, strong completeness)
  - Weak Failure Detector (weak accuracy, weak completeness)
  - Eventually Weak Failure Detector (eventually weak accuracy, weak completeness)

***Reference**: Distributed Computing: Principles, Algorithms, and Systems, Kshemkalyani & Singhal (2008)*

Failure & Messages

- In-Transit Messages
- Lost Messages (send is done but receive is undone)
- Delayed Messages (arrive after the expected time)
- Orphan Messages (receive recorded but send not recorded)
- Duplicate Messages



Reference: Distributed Computing: Principles, Algorithms, and Systems, Kshemkalyani & Singhal (2008)

Replication & Failover

- Replication
  - State Transfer (sending the state of the primary: contents of memory)
  - Replicated State Machine (do not send state, send external events: commands & operations)

- High-Availability Clusters
  - 1:1 (one active and one standby for failover)
  - 1:N (one passive failover node for N nodes)
  - 1+1 (active-active: both copies working in parallel)
  - 1+N (one additional active for N nodes)
  - N+M (M additional active for N nodes)

Resiliency

- The ability of the system to recover from failures and continue to function with minimal downtime and data loss before full recovery

- **MTBF** (Mean Time Between Failures)
  How long a component can reasonably expect to last between outages

- **MTTR** (Mean Time To Recovery)
  Average time it takes to restore a component after a failure

- **RPO** (Recovery Point Objective)
  Maximum acceptable time an application can be unavailable after an incident

***Reference**: https://docs.microsoft.com/en-us/azure/architecture/framework/*

Resiliency Patterns

Bulkhead	*Isolate elements of application into pools. If one fails, others continue to function*
Circuit Breaker	*Handle faults with variable amount of time to fix when connecting to a remote service or resource*
Compensating Transaction	*Undo the work performed by a series of steps of an eventually consistent operation*
Health Endpoint Monitoring	*Functional checks in an application that external tools can access through exposed endpoints*
Leader Election	*Select one instance as leader to coordinate the actions of a collection of collaborating task instances*
Queue-Based Load Leveling	*Use a queue that acts as a buffer between a task and a service that it invokes in order to smooth intermittent heavy loads*
Retry	*Enable an application to handle anticipated, temporary failures when it tries to connect to a service or network resource by transparently retrying an operation that's previously failed*
Scheduler Agent Supervisor	*Coordinate a set of actions across a distributed set of services and other remote resources*

***Reference**: https://docs.microsoft.com/en-us/azure/architecture/patterns/category/resiliency*

Logging

- Pessimistic Logging
  - A failure can happen after any non-deterministic event in the computation
  - Log before the event in stable storage
  - Synchronous
- Optimistic Logging
  - Assume logging will be complete before a failure occurs
  - Asynchronous
- Causal Logging
  - Optimistic
    - Does not require synchronous access to the stable storage except during output commit
  - Pessimistic
    - Allows each process to commit output independently and never creates orphans → Isolating processes from effects of failures at other processes

Reference: Distributed Computing: Principles, Algorithms, and Systems, Kshemkalyani & Singhal (2008)

Recovery

- Rollback to last checkpoint
  - Domino Effect: Cascaded Rollbacks (Rollback propagation)

- Log-based recovery
  - Undo & Redo

  Note: deterministic & non-deterministic operations

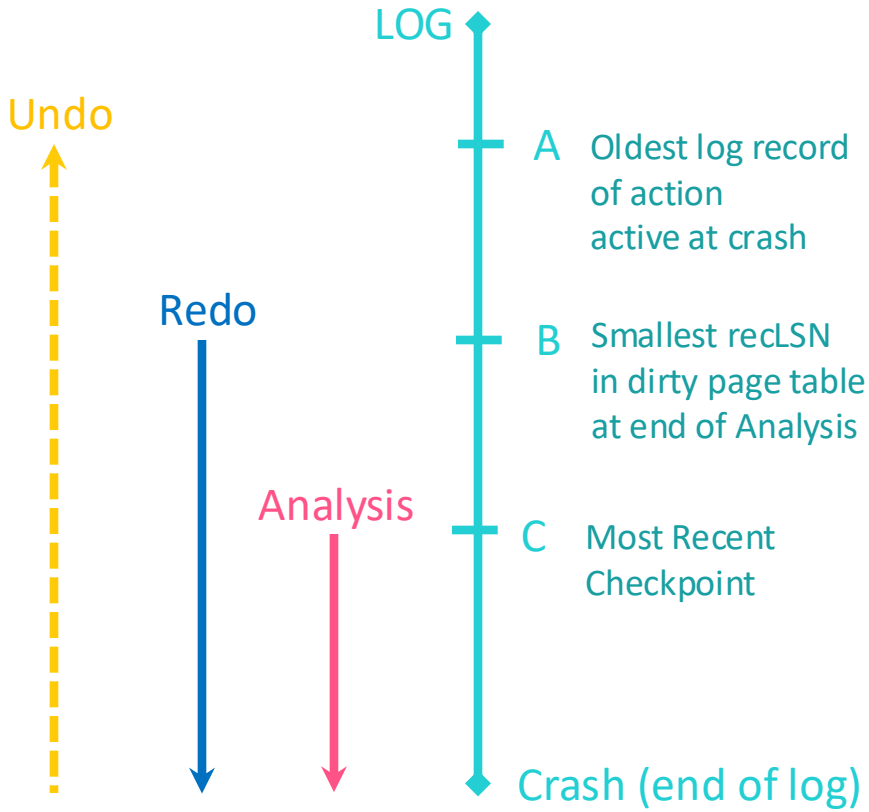
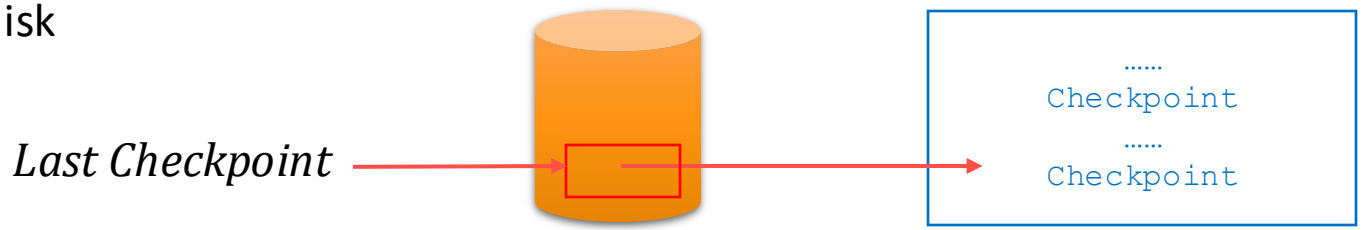


Figure (ARIES): Database Management Systems, Raghu Ramakrishnan, Johannes Gehrke (2003)

Checkpointing

- Local Checkpoints
  - No update to be performed while checkpoint operation in progress
    - If updates allowed: Fuzzy Checkpoint
  - Output all modified buffer pages to disk



- Distributed System
  - Uncoordinated Checkpointing
  - Coordinated Checkpointing (system-wide consistent state)
    - Blocking or Non-Blocking
  - Communication-Induced Checkpointing
    - Model-based
    - Index-based

***Reference**: Distributed Computing: Principles, Algorithms, and Systems, Kshemkalyani & Singhal (2008) Database System Concepts, Silbershatz, Korth, & Sudarshan (2019)*

Component Failures

HPC Cluster (1)	
Component	**Percentage**
Hard Drive	30.6
Memory	28.5
Misc	14.4
CPU	12.4
PCI Motherboard	4.9
Controller	2.9
QSW	1.7
Power Supply	1.6
MLB	1.0
SCSI BP	3.0

HPC Cluster (2)	
Component	**Percentage**
Power Supply	34.8
Memory	20.1
Hard Drive	18.1
Case	11.4
Fan	8.0
CPU	2.0
SCSI Board	0.6
NIC Card	1.2
LV Power Board	0.6
CPU Heatsink	0.6

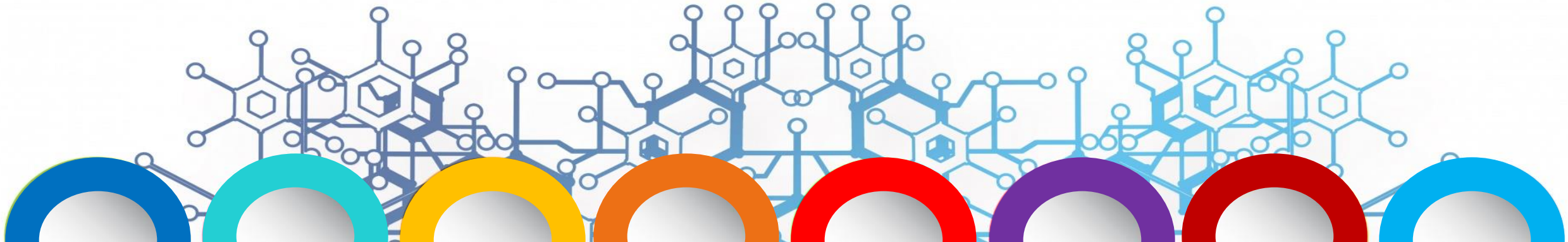
HPC Cluster (3)	
Component	**Percentage**
Hard Drive	49.1
Motherboard	23.4
Power Supply	10.1
RAID card	4.1
Memory	3.4
SCSI cable	2.2
Fan	2.2
CPU	2.2
CD-ROM	0.6
Raid Controller	0.6

__Reference__: Disk Failures in the Real World: What does an MTTF on 1,000,000 hours mean to you? (2007)

Summary

- **Preventing** Failure
  - Preventing element failure to cause system failure
- **Detecting** Failure
  - Every component (processors, storage, network, software) can and will fail
- **Recovering** From Failure
- Next: Coordination & Consensus

Coordination & Consensus



Context

- Replicated State Machines
  - State machines on a collection of servers compute
  - Identical copies of the same state
  - Can continue operating even if some of the servers are down

- Replicated State Machines Implemented using Replicated Log
  - Each server stores a log containing a series of commands
  - State machine executes those commands in order

Consensus Algorithms: Managing Replicated Log

***Reference**: In search of an understandable consensus algorithm (2014)*

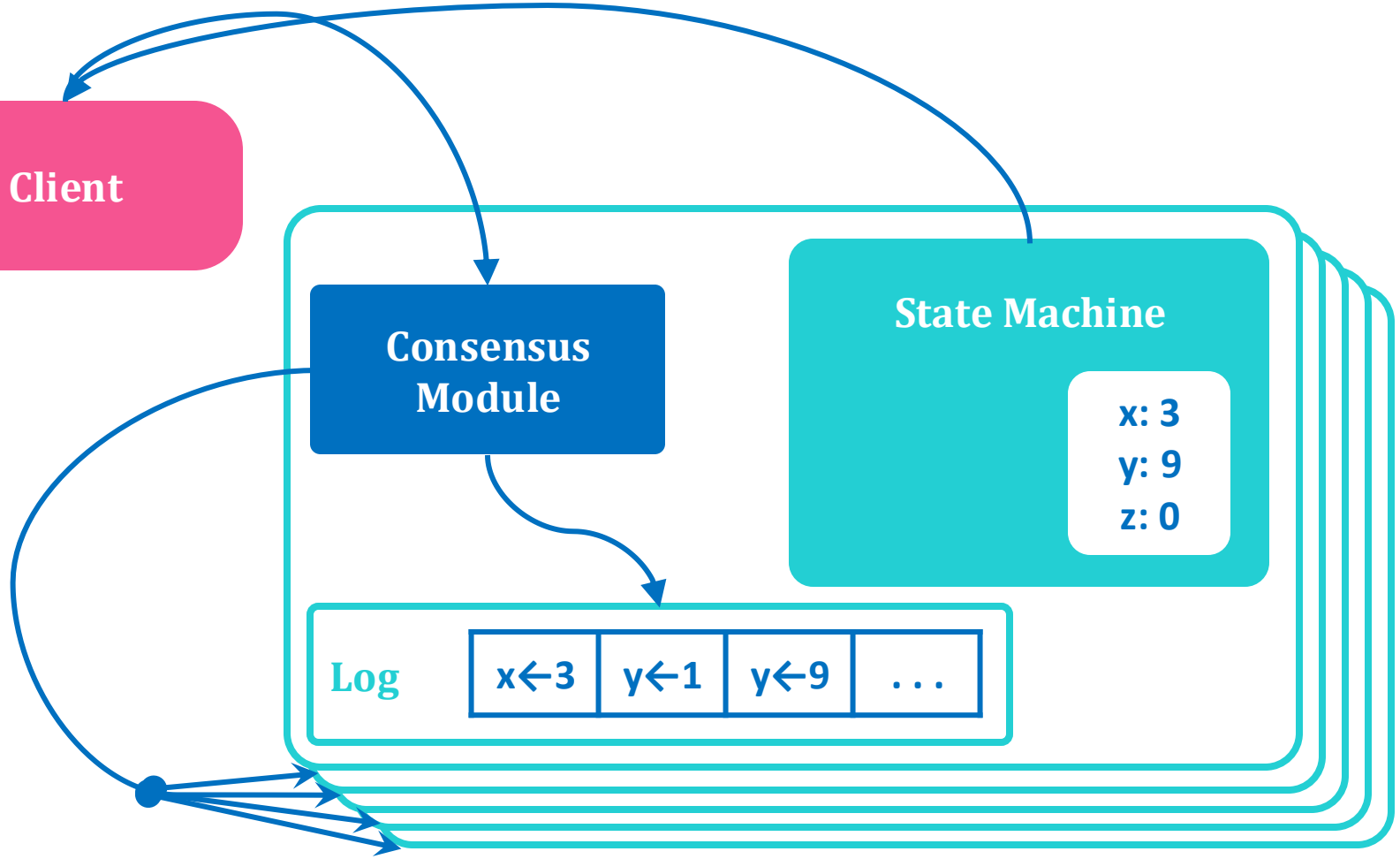
Consensus Algorithms

- Properties
  - Ensure ***safety***: Never return an incorrect result
  - ***Available*** as long as any ***majority of the servers*** are operational and can communicate with each other and with clients
  - ***Do not depend on timing*** to ensure the consistency of the logs
  - A command can ***complete as soon as a majority*** of the cluster has responded to a single round of remote procedure calls

Reference: *In search of an understandable consensus algorithm (2014)*

Replicated State Machine

Consensus Algorithm manages replicated log containing state machine commands from clients



Reference: In search of an understandable consensus algorithm (2014)

Consensus

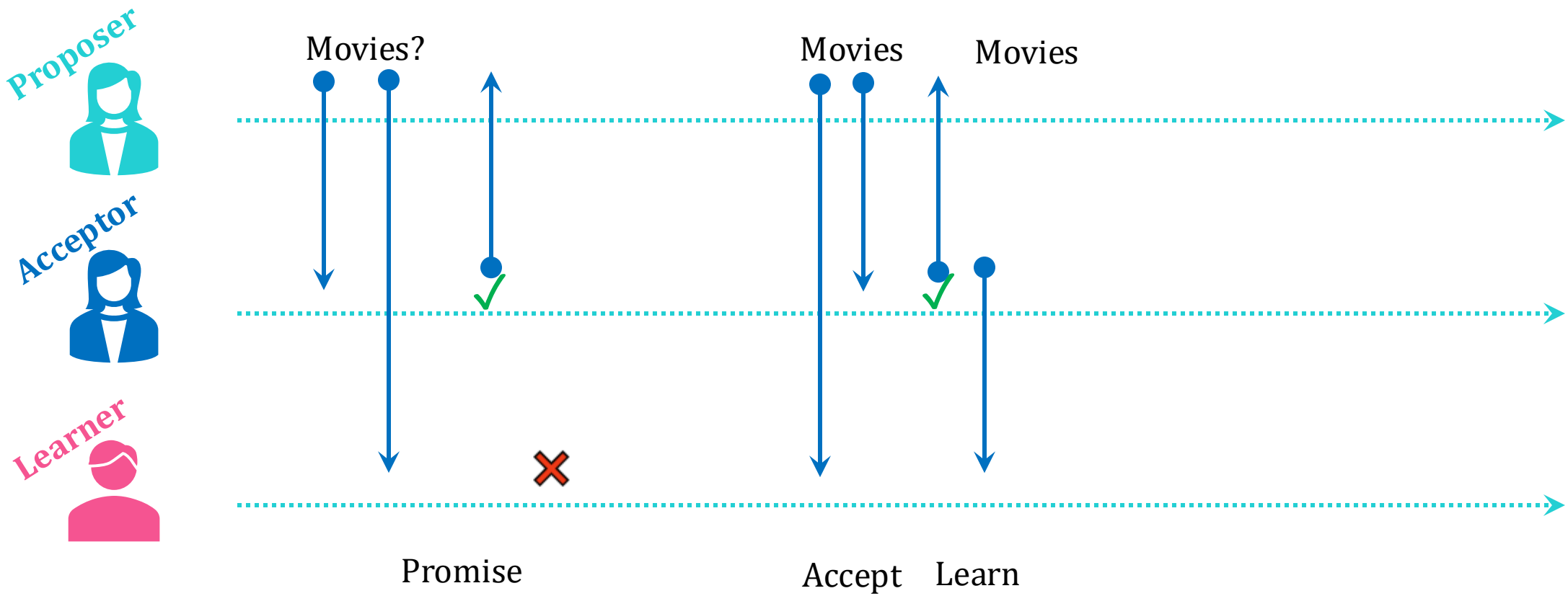
- Technical Context: Replicated State Machines

- Consensus Problem
  - **Agreement**: All non-faulty processes must agree on the same value
  - **Validity**: Agreed-upon value by all non-faulty processes must be the same value
  - **Termination**: Each non-faulty process must eventually decide on a value

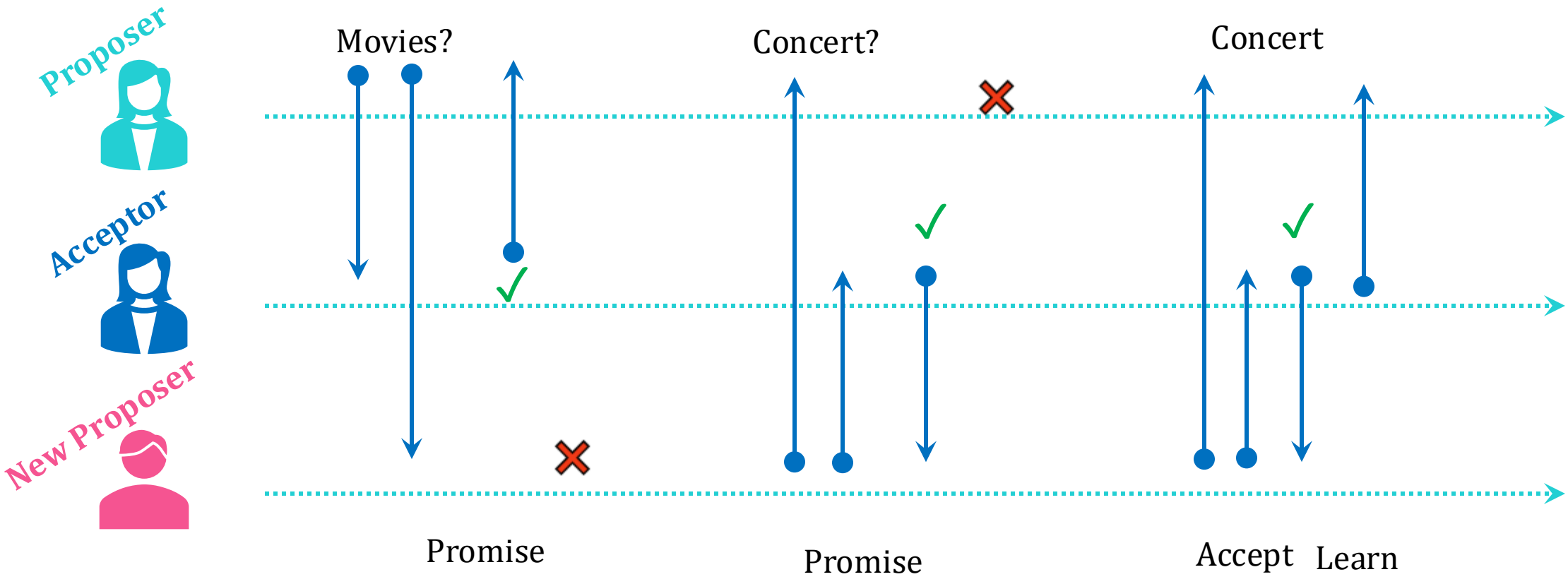
- **Example:** Group of friends deciding what to do?

Reference: Distributed Computing: Principles, Algorithms, and Systems, Kshemkalyani & Singhal (2008)

Movies?


Proposer
Acceptor
Learner
Movies?
Movies
Movies
Promise
Accept
Learn

How About a Concert?



Consensus

- Technical Context: Replicated State Machines

- Consensus Problem
  - **Agreement**: All non-faulty processes must agree on the same value
  - **Validity**: Agreed-upon value by all non-faulty processes must be the same value
  - **Termination**: Each non-faulty process must eventually decide on a value

- **Example:** Group of friends deciding what to do?

Reference: Distributed Computing: Principles, Algorithms, and Systems, Kshemkalyani & Singhal (2008)

Paxos

- Safety Requirements
  - Only a value that has been proposed may be chosen
  - Only a single value is chosen
  - A process never learns that a value has been chosen unless it actually has been

- Classes of Agents
  - Proposer
  - Acceptors
  - Learners

  In an implementation: a single process may act as more than one agent

Reference: Paxos Made Simple, Leslie Lamport (2001)

Non-Byzantine

- Agents
  - Operate at arbitrary speed
  - May fail by stopping
  - May restart

- Messages
  - Can take arbitrarily long to be delivered
  - Can be duplicated
  - Can be lost

→ No Corruption & No Collusion

***Reference**: Paxos Made Simple, Leslie Lamport (2001)*

Paxos

- *An **acceptor** can accept a proposal numbered **n iff** it has not responded to a **prepare** request having a number greater than **n***

- *If a proposal with value **v** is chosen*
  - ***Every higher-numbered proposal that is chosen has value v***
  - *Every higher-numbered proposal **accepted by any acceptor** has value **v***
  - *Every higher-numbered proposal **issued by any proposer** has value **v***
  - *For any v and n, if a proposal with value v and number **n** is issued*
    - *Then there is a set **S** consisting of a majority of acceptors such that either*

      *(a) no acceptor in **S** has accepted any proposal numbered less than **n**, or*
      *(b) **v** is the value of the highest-numbered proposal among all proposals numbered less than **n** accepted by the acceptors in **S***

***Reference**: Paxos Made Simple, Leslie Lamport (2001)*

Paxos

Phase One (Prepare)

a) *A **proposer** selects **n** and **sends** a **prepare** **request** to a **majority of** **acceptors***

b) *If an **acceptor** receives a **prepare** **request** with greatest **n** promised by far, it responds with a **promise** not to accept any more proposals numbered less than **n***

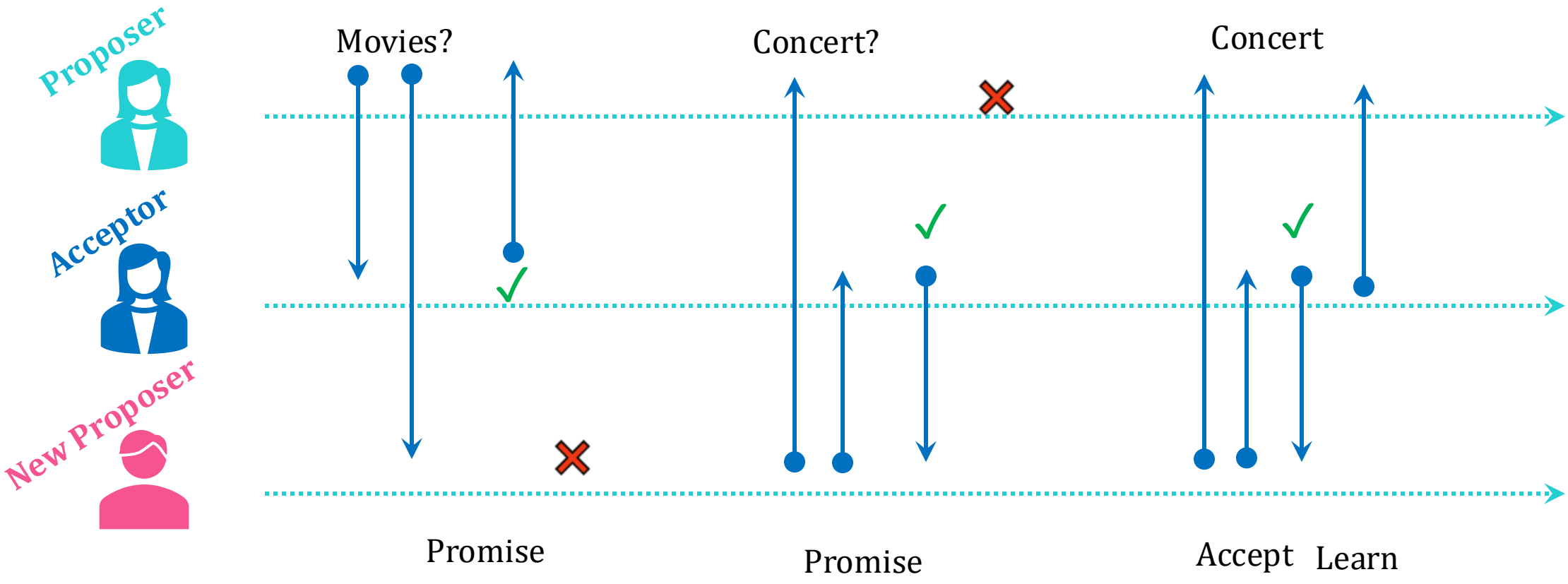
Phase Two (Accept)

a) *If the **proposer** receives a response to its **prepare** requests **n** from a **majority of** **acceptors**: it sends an **accept** **request** to **each of those** **acceptors***

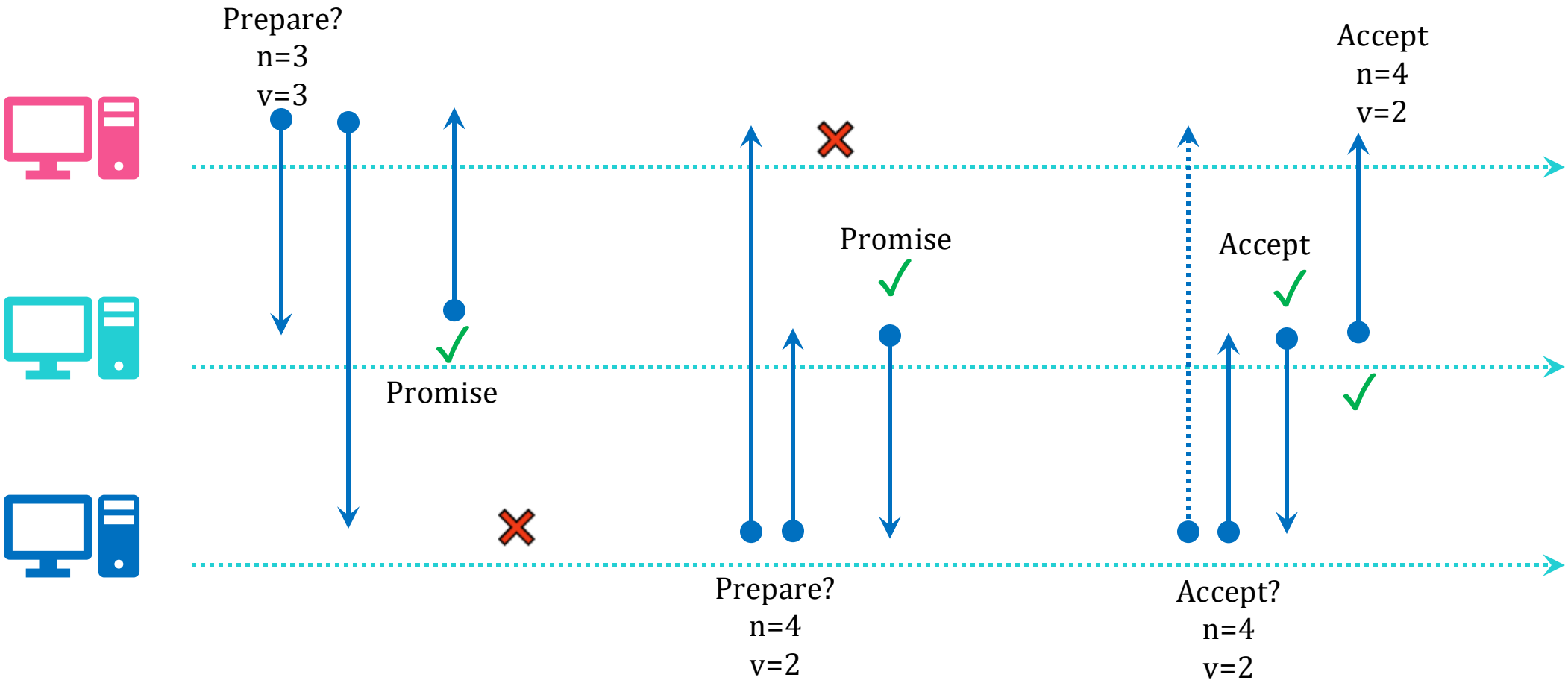
b) *If an **acceptor** receives an **accept** **request** for a proposal numbered **n**, it **accepts** the proposal **unless** it already responded to a **prepare** **request** greater than **n***

Reference: *Paxos Made Simple, Leslie Lamport (2001)*

How About a Concert?



Paxos



Assumptions about nodes and links?

Paxos

- Paxos: One Choice
  - More than one choice? Multi-Paxos (Repeat for each decision)

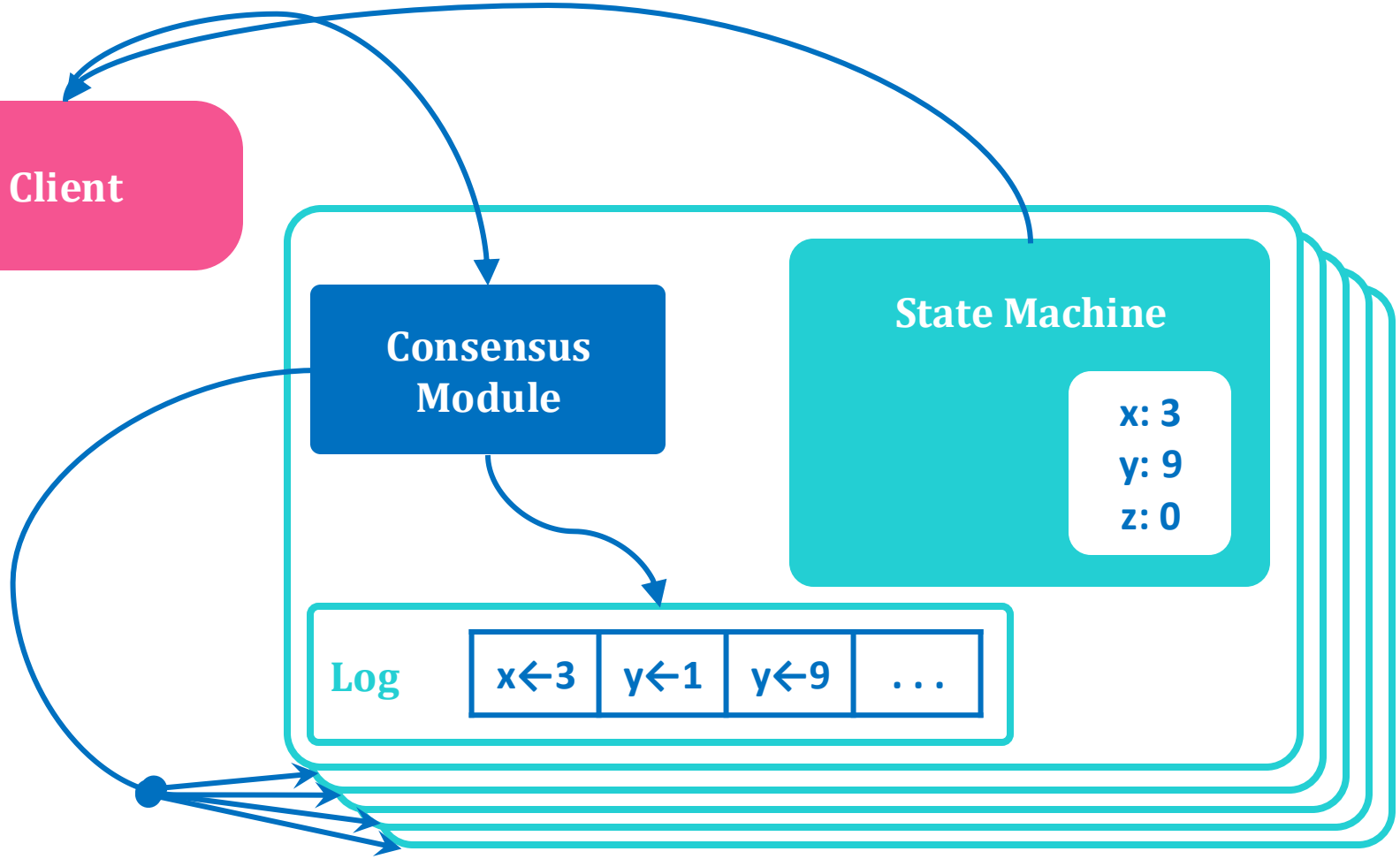
- Different real-world implementations
  - Multi-Paxos
  - Log of agreements
  - Node Churn
  - Byzantine Failures

	Paxos (or RAFT)
Fault Types	Fail-Stop
Failover	Instant
Servers	2f+1
Messages	Linear

Reference: Paxos in the Cloud: Paxos Systems Demystified (2016) and Chris Colohan presentation on Paxos (2017)

Replicated State Machine

Consensus Algorithm manages replicated log containing state machine commands from clients



Reference: In search of an understandable consensus algorithm (2014)

RAFT

- **RAFT**: Reliable & Fault Tolerant
- More understandable algorithm
  - Strong Leader
  - Majority vote: Leader election, commit
  - Heartbeat messages

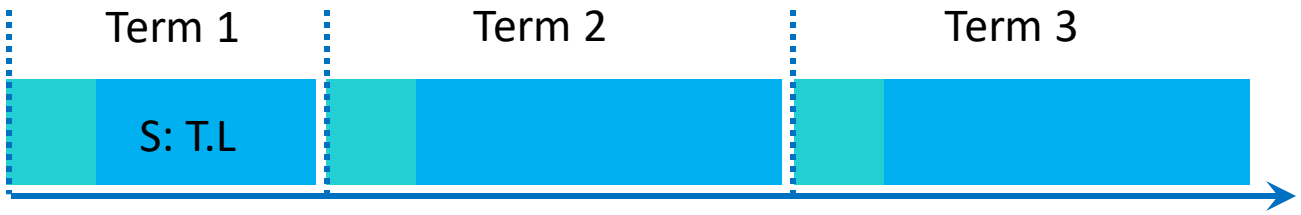
- Three Sub-problems
  - **Leader Election:** A new leader must be chosen when an existing leader fails
  - **Log Replication:** The leader must accept client log entries and replicate across the cluster (forcing other logs to agree with its own)
  - **Safety:** If any server has applied a particular log entry to its state machine, then no other server may apply a different command for the same log index

Reference: In search of an understandable consensus algorithm (2014)

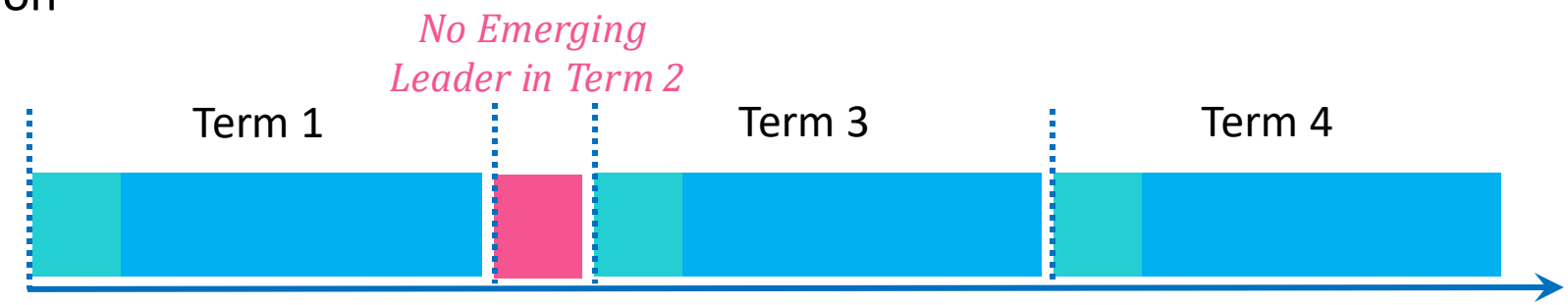
RAFT

- Server States: Leader, Follower, Candidate

- Time divided into terms
  - Each term begins with an election

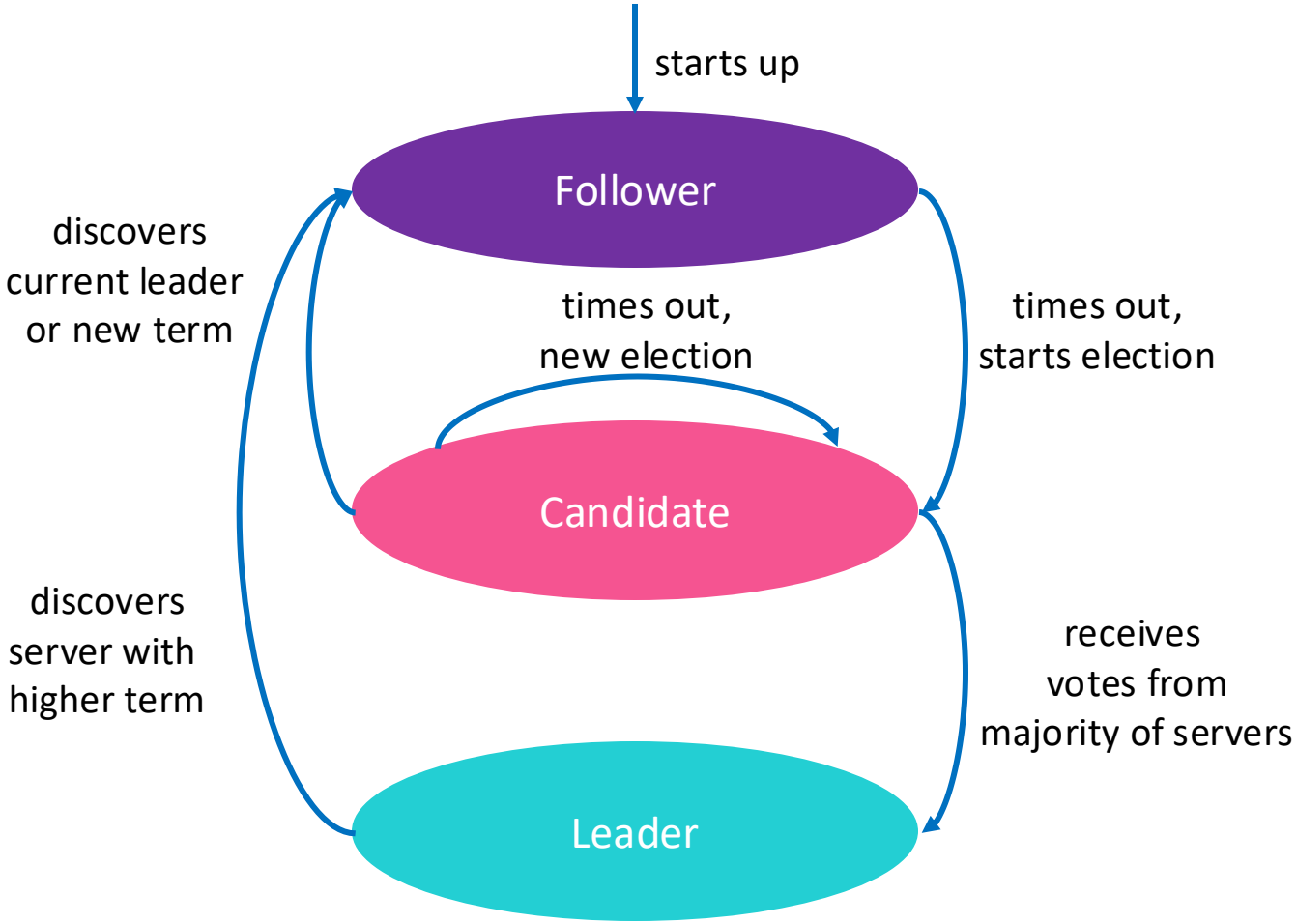


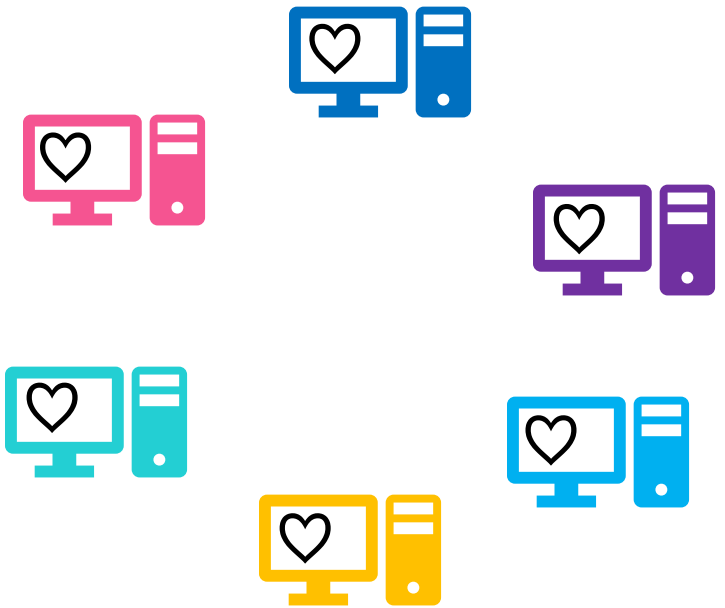
- Heartbeat to trigger election



***Reference**: In search of an understandable consensus algorithm (2014)*

RAFT





Reference: In search of an understandable consensus algorithm (2014)

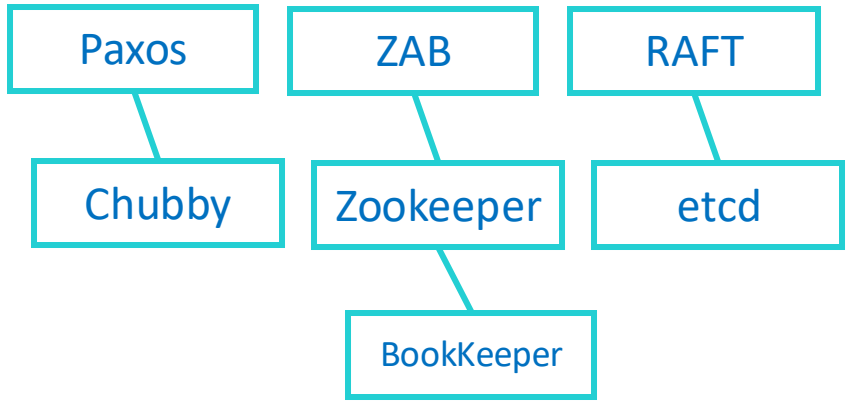
Consensus & The Cloud

- Server Replication (State Machine Replication)
- Log Replication (Data Integration Systems)
- Synchronization Service (Synchronization)
- Barrier Orchestration (Double barrier to orchestrate beginning and end of each synchronization)
- Configuration Management (Leader Election, Group Membership, Service Discovery, Metadata Management)
- Distributed Message Queues (Atomic Broadcast)

***Reference**: Paxos in the Cloud: Paxos Systems Demystified (2016)*

Paxos Protocols & Systems

- Paxos Protocols
  - Paxos: Consensus
  - Raft: Consensus (Explicit Leader Election)
  - ZAB: Consensus (Atomic Broadcast)



- Paxos Systems
  - Zookeeper (Zab for distributed consensus)
  - Chubby (Adaptation of Paxos for GFS lock service)
  - etcd (RAFT)

***Reference**: Paxos in the Cloud: Paxos Systems Demystified (2016)*

Summary & Conclusion

- Availability (Replication, Partitioning)
- Performance (Latency)
- Scalability (State, Sharing/Affinity/Coupling, Parallelism)
- Consistency (Locking)
- Fault Tolerance (Replication, High-Availability, Coordination & Consensus)

Acknowledgement

The list of resources used in preparation of this slide set are provided on:

https://canvas.sfu.ca/courses/88212/pages/references

Pictures and quoted resources are mentioned in each use.