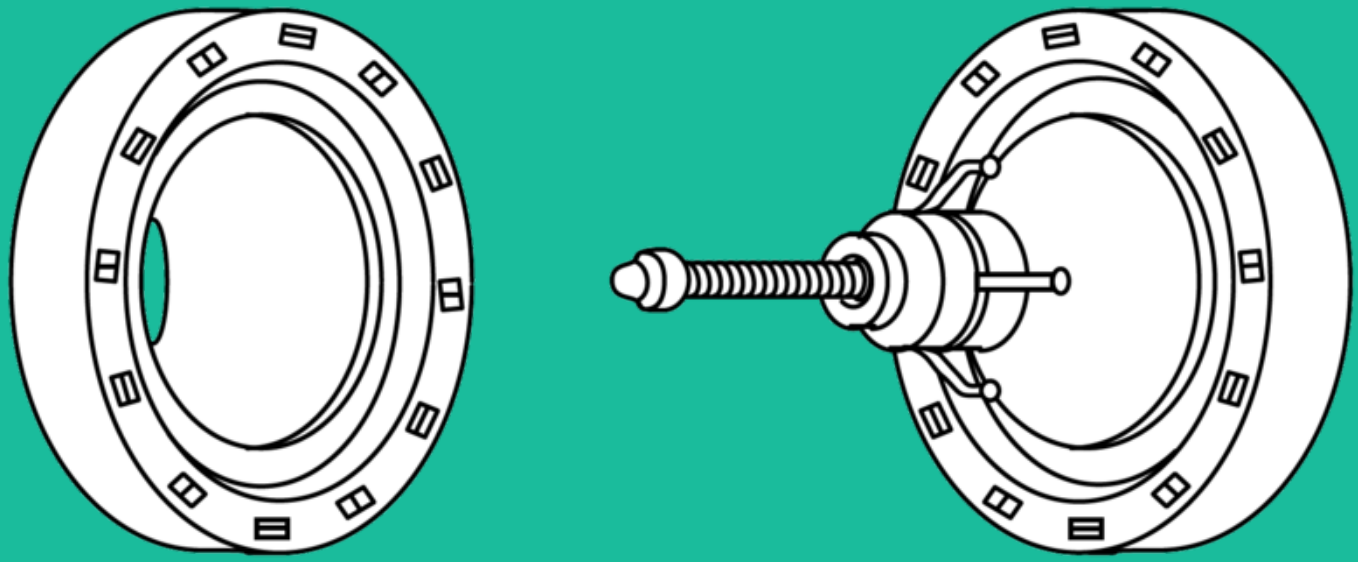


Chapter 8. Resource Management & Coordination

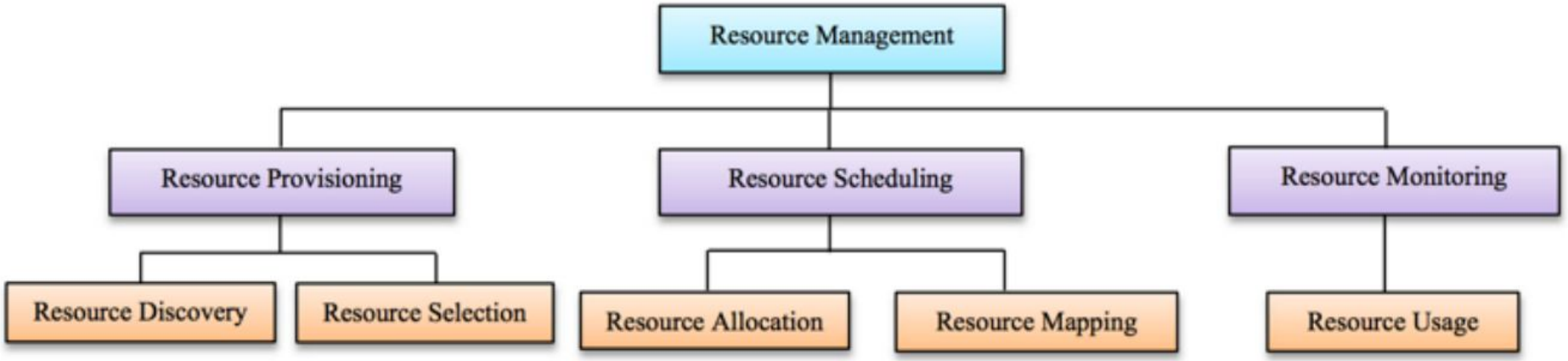
Bilkent University | CS443 | 2021, Spring | Dr. Orçun Dayıbaş

Cloud Resources

- **Computation**
  - Processor, Memory, Algorithms, APIs
- **Storage**
  - Hard/Flash drive, SW (Object store, DFS, etc.), DBs
- **Communication**
  - Physical (Routers, switches, cables, etc.)
  - Logical (Bandwidth, delay, protocols, etc.)
- **Power/Energy**
  - UPS, HVAC systems, etc.
- **Security**
  - Trust, Authentication, Integrity, Privacy

Cloud Resources

- **Resource management**
  - Provisioning
    - Simply provide it (conf. mng., deployment, etc.). Ex: Ansible
  - Scheduling
    - Utilize resource pool (hard to optimize). Ex: Titus, Kubernetes (Autoscaling)
  - Monitoring
    - Measure to manage. Ex: Prometheus



source: https://www.researchgate.net/publication/316494357

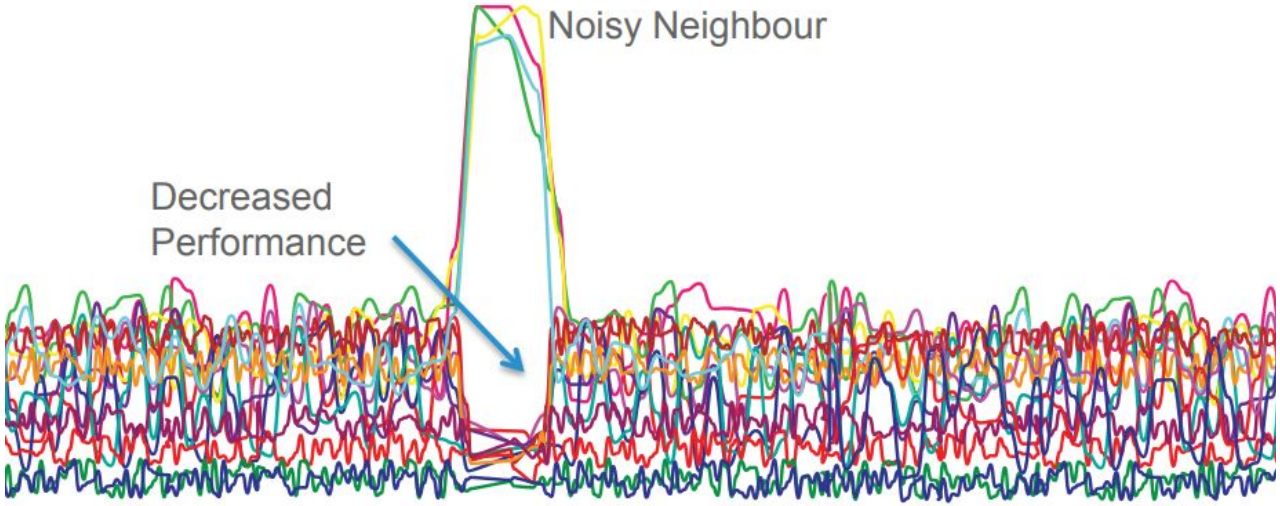
Cloud Resources

- **Everything is SW & every SW is containerized**
  - XaaS, Software-defined X
  - Elasticity → Automation of everything
  - Resource management ~= container orchestration
- **Container Orchestration Process**

Scheduling	Resource Management	Service Management
• Placement	• Memory	
• Replication/ Scaling	• CPU	• Labels
• Resurrection	• GPU	• Groups/ Namespaces
• Rescheduling	• Volumes	• Dependencies
• Rolling Deployment	• Ports	• Load Balancing
• Upgrades	• IPs	• Readiness Checking
• Downgrades		
• Collocation		

Cloud Resources

- **Noisy Neighbors**
  - Noisy neighbor is a phrase used to describe a cloud computing infrastructure co-tenant that monopolizes bandwidth, disk I/O, CPU and other resources, and can negatively affect other users' cloud performance.
  - The noisy neighbor effect causes other containers/applications that share the infrastructure to suffer from uneven cloud network performance.



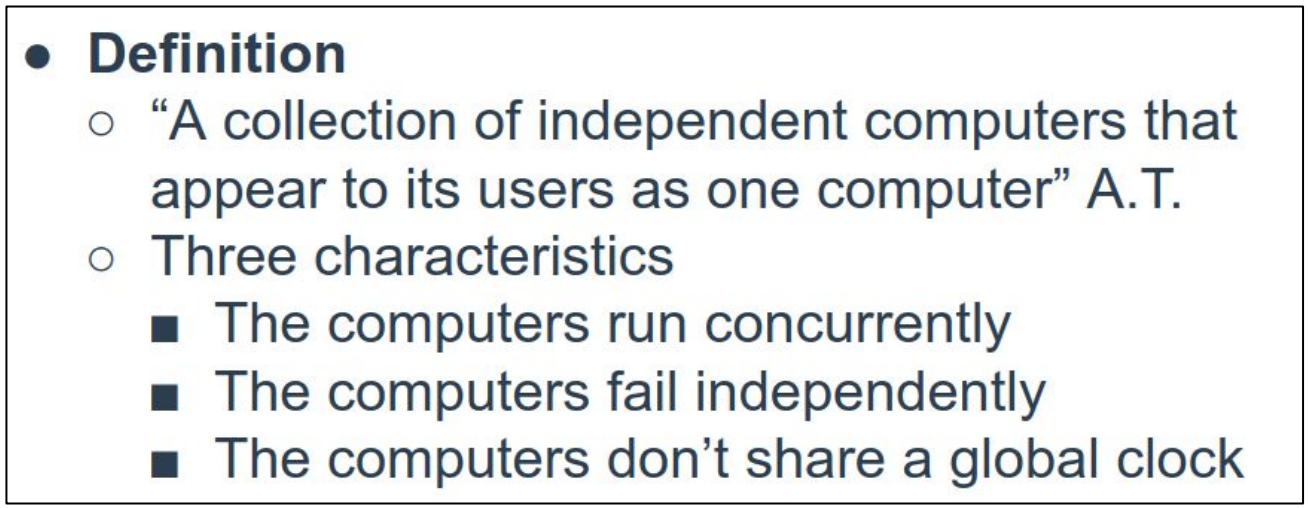
https://www.slideshare.net/Databarracks/controlling-performance-in-the-cloud-taking-charge-of-your-hosting-environment

Data Replication

- **Redundancy**
  - Cloud resources (individually or as a whole like hybrid cloud)
    - Ex: ANSI/TIA-942 standard tiers (for Datacenters)
  - It is perfectly OK for
    - Computation, Communication, Power or even storage
  - But redundant “meta”data → Inconsistent system
    - We need to solve that part
- **Data replication techniques**
  - Gossip/multicast protocols
    - Epidemic broadcast trees, bimodal multicast, SWIM, HyParView, etc.
    - Do not solve inconsistency problem
  - Consensus protocols
    - Paxos, Raft, Zab, etc.
    - Solves inconsistency problem

Distributed Consensus

- **Distributed systems**
  - Modern systems/solutions are distributed
  - Distributed systems are harder to implement
    - Lack of global knowledge: all you have exchanged messages, up-to-date?
    - Time: Clock skew, msg. order (delay/duplicate messages)
    - Consistency: Concurrent operations, conflict, consistent state
    - Failures: detecting and recovering
  - Remember chapter-2 (slide #22 to be exact)



Distributed Consensus

- **Definition**
  - A distributed consensus ensures a consensus of data among nodes in a distributed system or reaches an agreement on a proposal
  - The real world applications include clock synchronization, opinion formation, smart power grids, state estimation, control of UAVs (and multiple robots/agents in general), load balancing, blockchain and others
- **Consensus algorithms/protocols**
  - Paxos Family
    - Multi-paxos, EPaxos, WPaxos, Cheap/Fast Paxos, etc.
  - Raft, Zab
  - Bitcoin/cryptocurrency ecosystem???
    - Nakamoto Consensus
    - Proof-of-work

Distributed Consensus

- **Protocols & Implementations**

System	Protocol	Implementation	Usage
Google GFS	Multi-Paxos	Chubby	Lock Service
Google Spanner	Multi-Paxos	Chubby	
Google Borg	Multi-Paxos	Chubby	Configuration, Master election
Apache HDFS	Zab	ZooKeeper	Failure detection, Active NameNode election
Apache Giraph	Zab	ZooKeeper	Coordination, Configuration, Aggregators
Apache Hama	Zab	ZooKeeper	Coordination
CoreOS	Raft	etcd	Service Discovery
OpenStack	Zab	ZooKeeper	Service Discovery
Apache Kafka	Zab	ZooKeeper	Coordination, Configuration
Apache BookKeeper	Zab	ZooKeeper	Coordination, Configuration

source: https://muratbuffalo.blogspot.com/2015/10/consensus-in-wild.html

Paxos

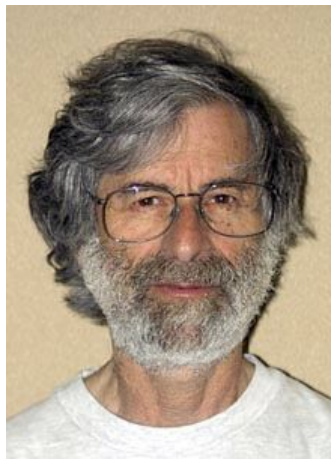
- **Definition**
  - Paxos is a family of protocols for solving consensus in a network of unreliable processors
- **Background**
  - ACM Transactions on Computer Systems
    - Submitted: 1990, Accepted: 1998

Γωνδα is the new cheese inspector

The Part-Time Parliament

LESLIE LAMPORT
Digital Equipment Corporation

Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned despite the peripatetic propensity of its part-time legislators. The legislators maintained consistent copies of the parliamentary record, despite their frequent forays from the chamber and the forgetfulness of their messengers. The Paxon parliament's protocol provides a new way of implementing the state machine approach to the design of distributed systems.



Paxos

- **Background**
  - 10 years later, Leslie Lamport revised the original paper

Paxos Made Simple

Leslie Lamport

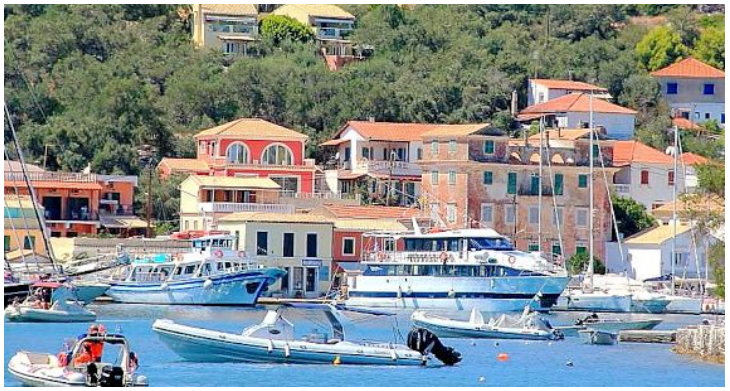
01 Nov 2001

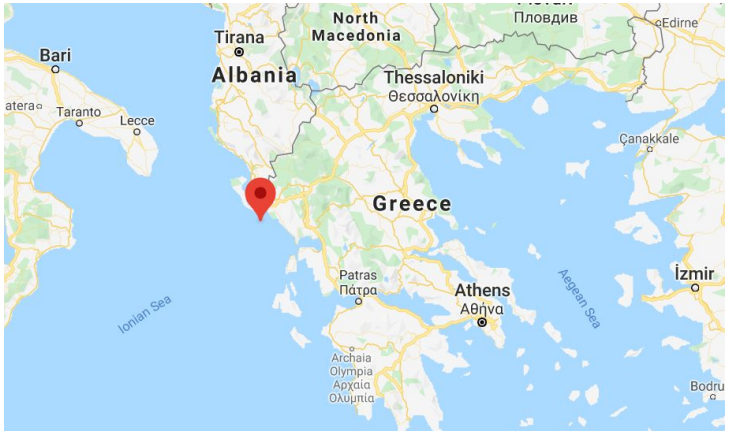
Abstract

The Paxos algorithm, when presented in plain English, is very simple.

1 Introduction

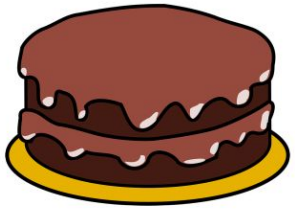
The Paxos algorithm for implementing a fault-tolerant distributed system has been regarded as difficult to understand, perhaps because the original presentation was Greek to many readers [5]. In fact, it is among the simplest and most obvious of distributed algorithms. At its heart is a consensus algorithm—the "synod" algorithm of [5]. The next section shows that this consensus algorithm follows almost unavoidably from the properties we want it to satisfy. The last section explains the complete Paxos algorithm, which is obtained by the straightforward application of consensus to the state machine approach for building a distributed system—an approach that should be well-known, since it is the subject of what is probably the most often-cited article on the theory of distributed systems [4].

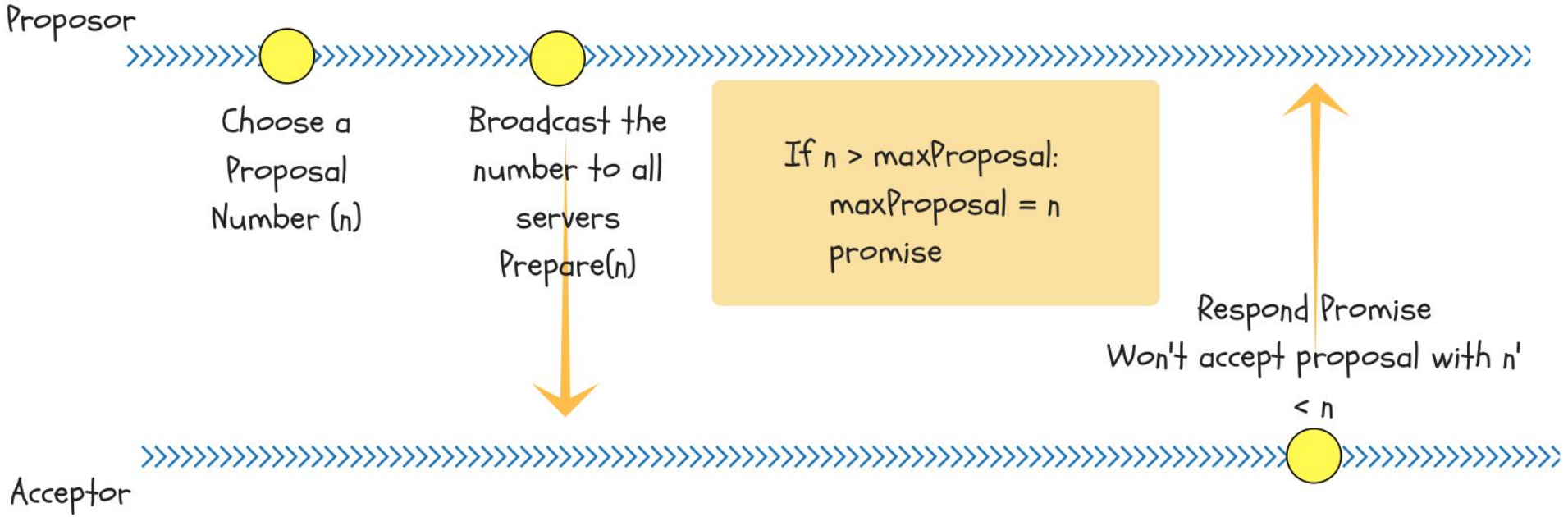




Paxos

- **Basic Paxos**
  - Assume group of people deciding on cake flavor to order
  - Prepare to propose & propose
  - Roles: Proposer, Acceptor

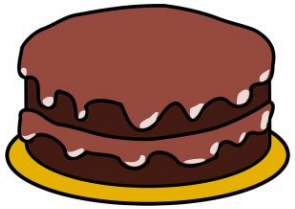


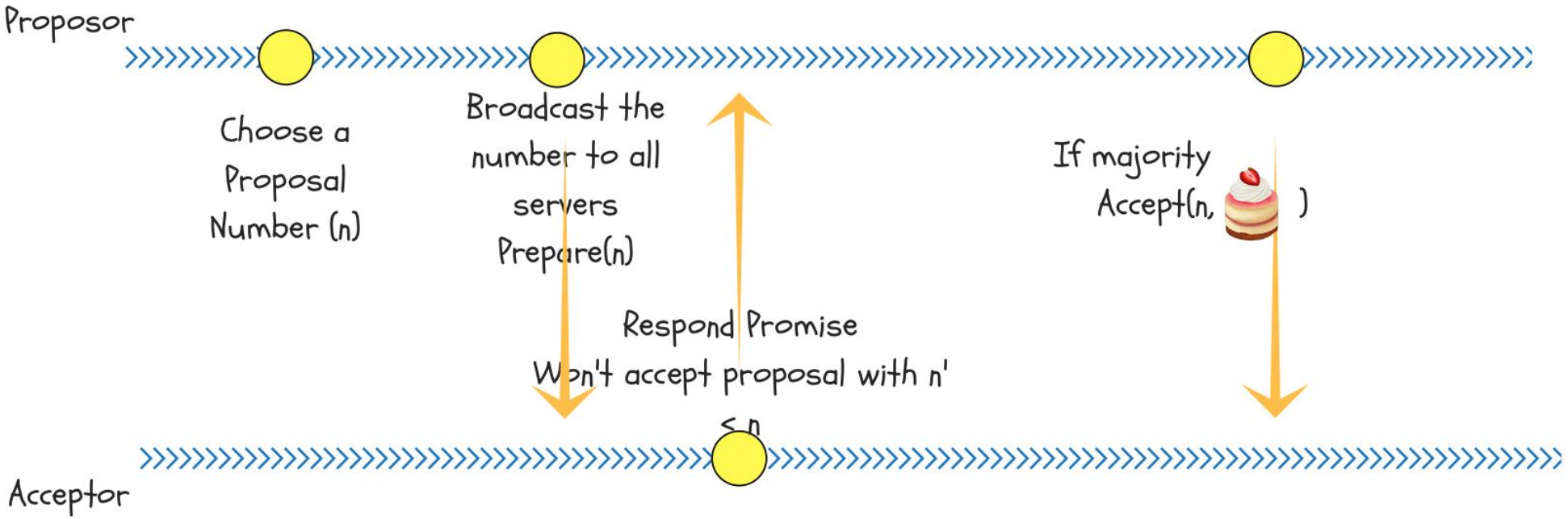


source: https://www.slideshare.net/YifanXing/consensus-algorithmspaperswelove2018sep

Paxos

- **Basic Paxos**
  - Assume group of people deciding on cake flavor to order
  - Prepare to propose & propose
  - Roles: Proposer, Acceptor

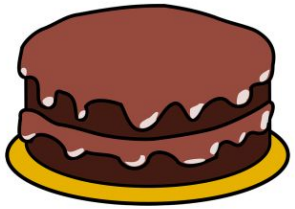


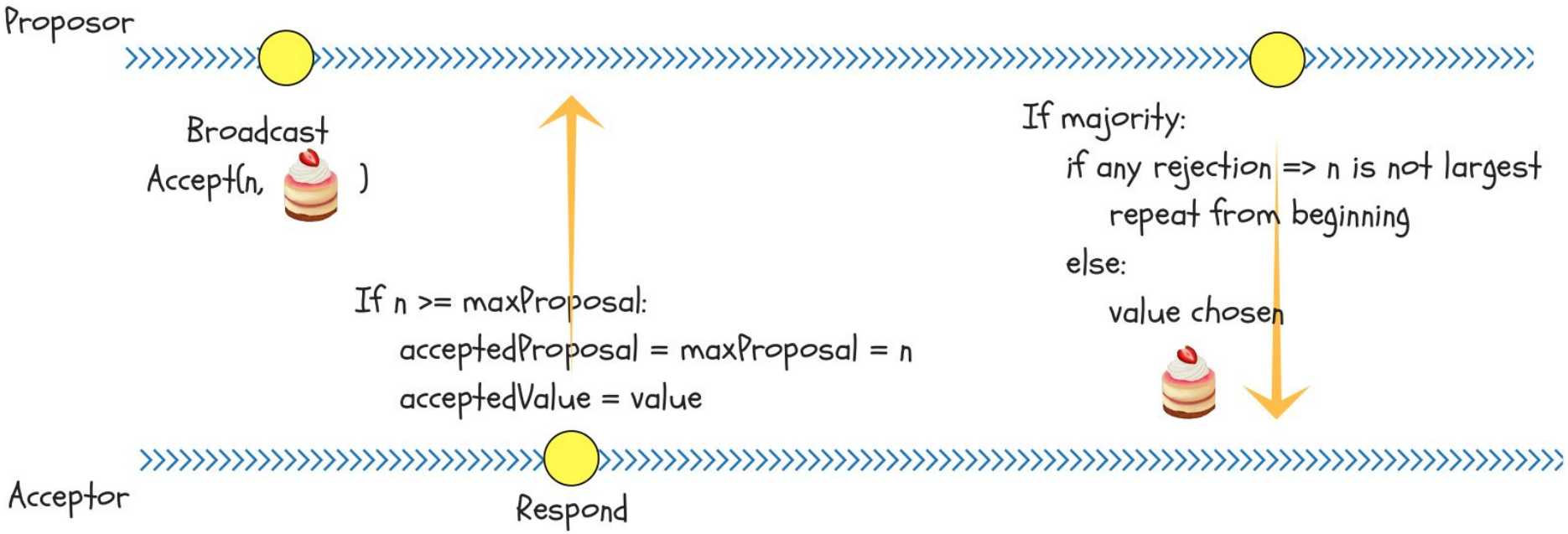


source: https://www.slideshare.net/YifanXing/consensus-algorithmspaperswelove2018sep

Paxos

- **Basic Paxos**
  - Assume group of people deciding on cake flavor to order
  - Prepare to propose & propose
  - Roles: Proposer, Acceptor





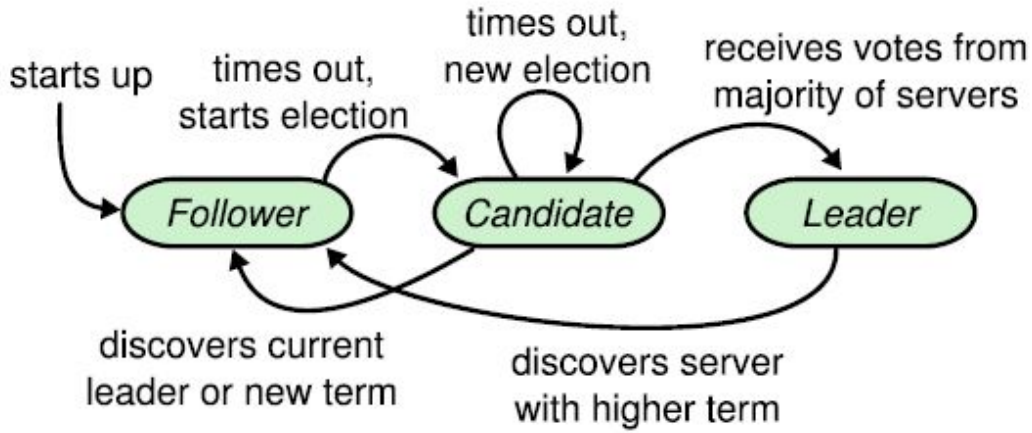
source: https://www.slideshare.net/YifanXing/consensus-algorithmspaperswelove2018sep

Raft

- **Definition**
  - **R**eliable, **R**eplicated, **R**edundant, **A**nd **F**ault-**T**olerant
  - A consensus algorithm designed as an alternative to Paxos. It was meant to be more **understandable** than Paxos by means of separation of logic, but it is also formally proven safe and offers some additional features
- **Goal**
  - Easier to be implemented (understandable)
  - The system is fully operational as long as **a majority of the servers are up**
- http://thesecretlivesofdata.com/raft/

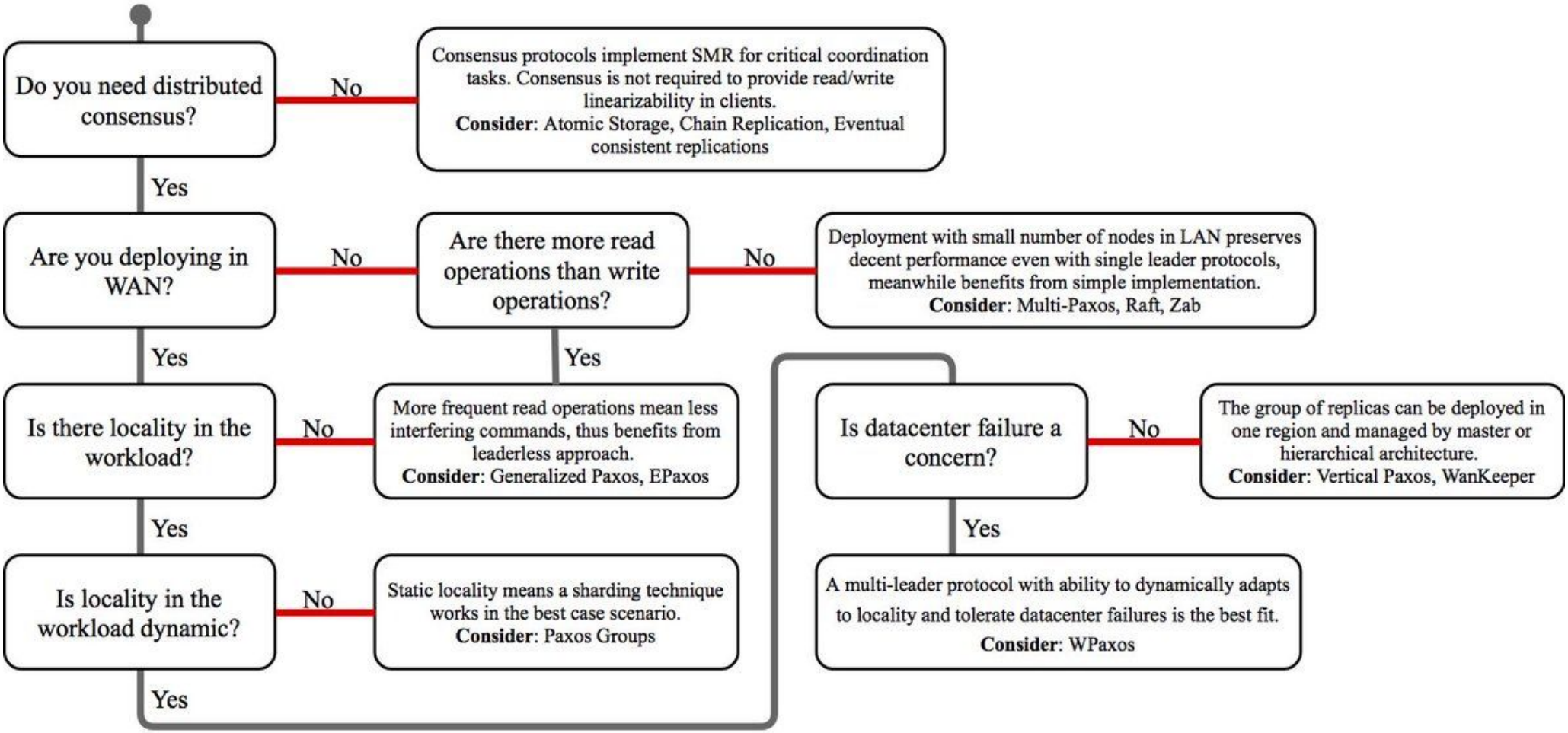
Raft

- **Raft decomposes consensus into 3 sub-problems**
  - **Leader Election:** A new leader needs to be elected in case of the failure of an existing one
  - **Log replication:** The leader needs to keep the logs of all servers in sync with its own through replication
  - **Safety:** If one of the servers has committed a log entry at a particular index, no other server can apply a different log entry for that index
- **State changes**



Choosing a Paxos Variant

http://paxos.systems/variants.html



Distributed Consensus

- **Recap**
  - Consensus on what?
    - Use coordination of servers on “meta”data of system
    - Do not abuse by using it in every occasion (yes, ZooKeeper is KV store but it is designed for a specific purpose): https://www.confluent.io/blog/distributed-consensus-reloaded-apache-zookeeper-and-replication-in-kafka/
  - Implementing consensus algorithms
    - Don’t do that
    - Resilient against issues?
    - Worth it to be resilient? (Scoping problem)
      - Byzantine Fault tolerant Paxos/Raft
  - Designing consensus algorithms
    - Consistency, Reliability, Complexity
    - Understandability

Time & Ordering

- **Time synchronization**
  - It is required for both correctness & fairness
    - Assume your watch is off by 15 mins. and you want to catch a bus: Late → missing the bus, Fast → unfairly waiting for the bus.
  - Distributed system → Server A, B and C
    - Server A receives a client request to purchase
    - Server A timestamps purchase using local clock 9h:15m:32.45s, and logs it. Replies ok to client.
    - That was the last seat.
    - Server A sends message to Server B saying “flight full.”
    - B enters “Flight ABC 123 full” + its own local clock value (which reads 9h:10m:10.11s) into its log.
    - Server C queries A’s and B’s logs. Is confused that a client purchased a ticket at A after the flight became full at B.
    - This may lead to further incorrect actions by C.
  - Each have their own clocks

source: Distributed Systems by Indranil Gupta

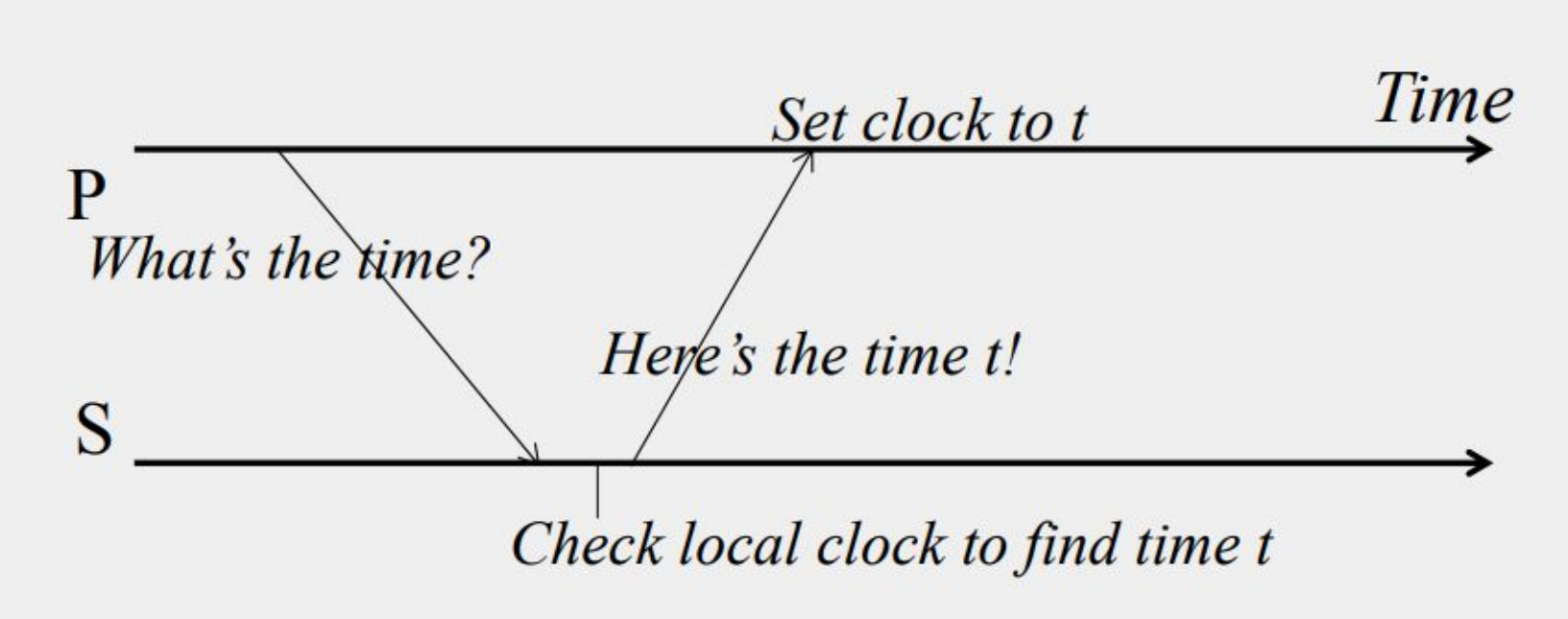
Time & Ordering

- **Time synchronization**
  - Cloud systems follow asynchronous system model
    - No bounds on message delays (network + processing)
    - Each sub-system consists of a number of **processes**
    - Each process has a **state** (values of variables)
    - Each process takes **actions** to change its states
    - An **event** is the occurrence of an action
  - But - in a distributed system, we also need to know the time order of events across different processes
- **Clock skew & drift**
  - Clock skew: relative diff. in clock values of two processes
  - Clock drift: relative diff. in clock frequencies of two processes

source: Distributed Systems by Indranil Gupta

Time & Ordering

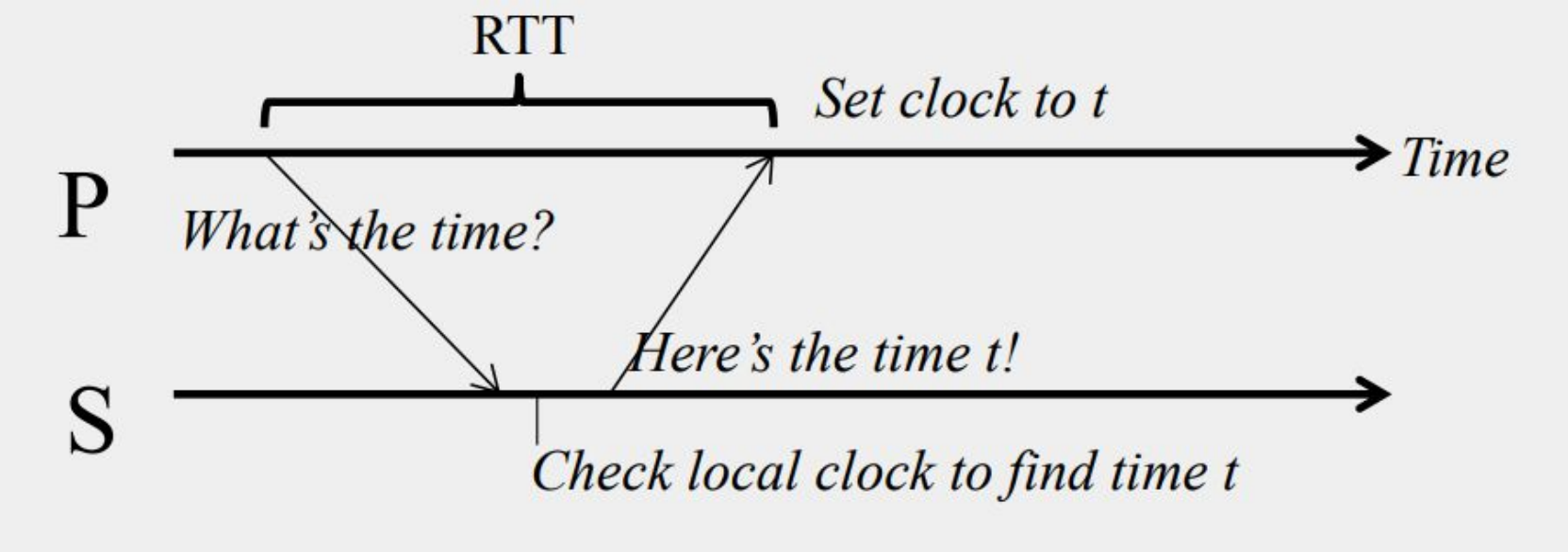
- **Naive method**
  - All processes P synchronize with a time server S



source: Distributed Systems by Indranil Gupta

Time & Ordering

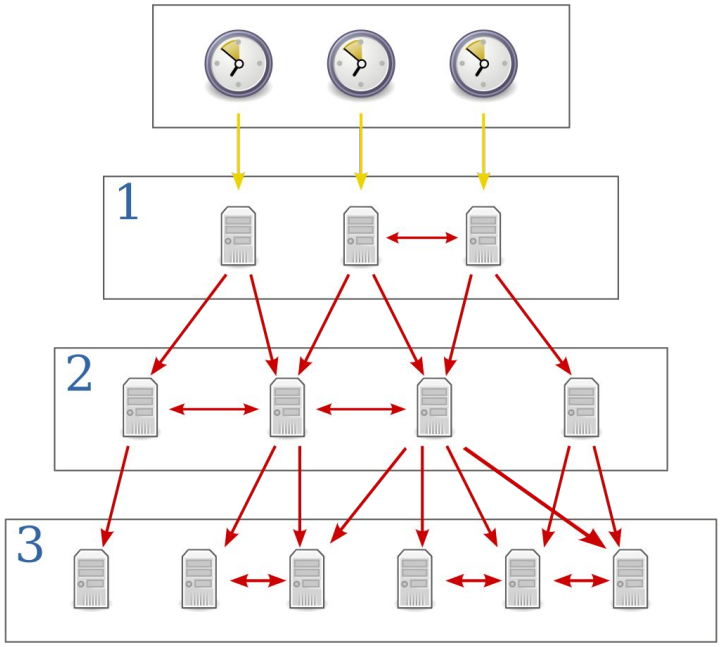
- **Cristian's algorithm**
  - P measures the round-trip-time RTT of msg. exchange
    - Suppose we know the min. latency for P→ S is min1, for S→ P is min2
    - The actual time at P when it receives resp. is between [t+min2, t+RTT-min1]
    - P sets its time to halfway through this interval: t+(RTT+min2-min1)/2
    - Error is at most (RTT-min2-min1)/2 → Bounded!
  - If error is too high, take multiple readings & average them

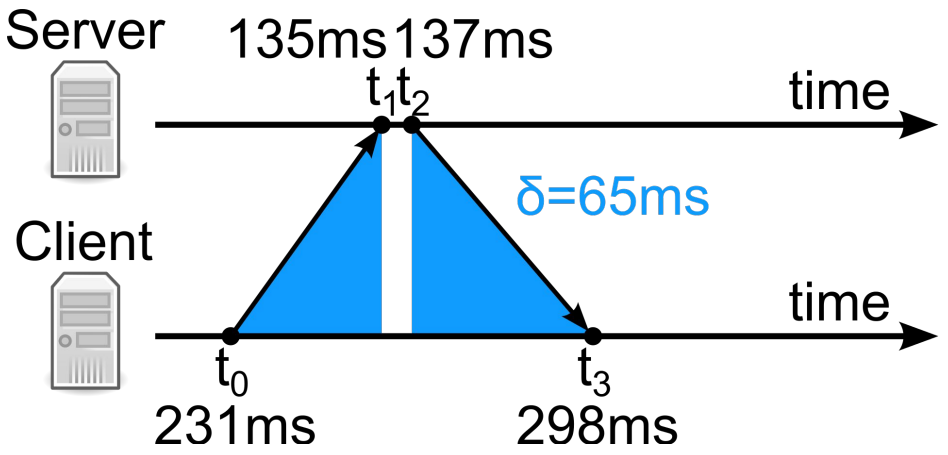


source: Distributed Systems by Indranil Gupta

Time & Ordering

- **NTP (Network Time Protocol)**
  - NTP servers are organized in a tree structure (Each level of this hierarchy is termed as **stratum**)
    - Stratum 0 consists of very precise atomic clocks
  - Each node synchronizes with its parent
  - Examples: Google NTP service, Facebook NTP service

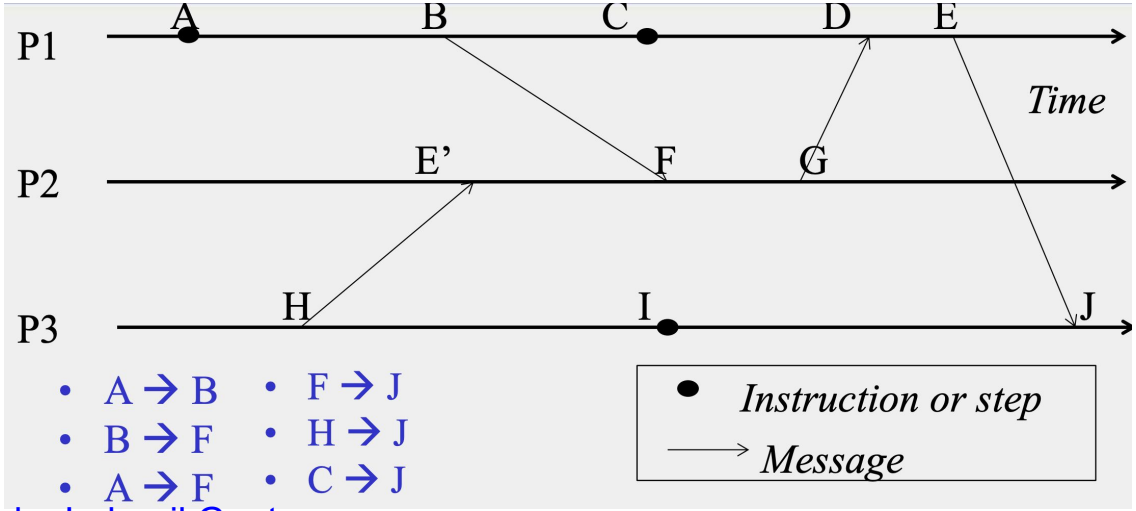




source: wikipedia

Logical/Lamport Timestamps

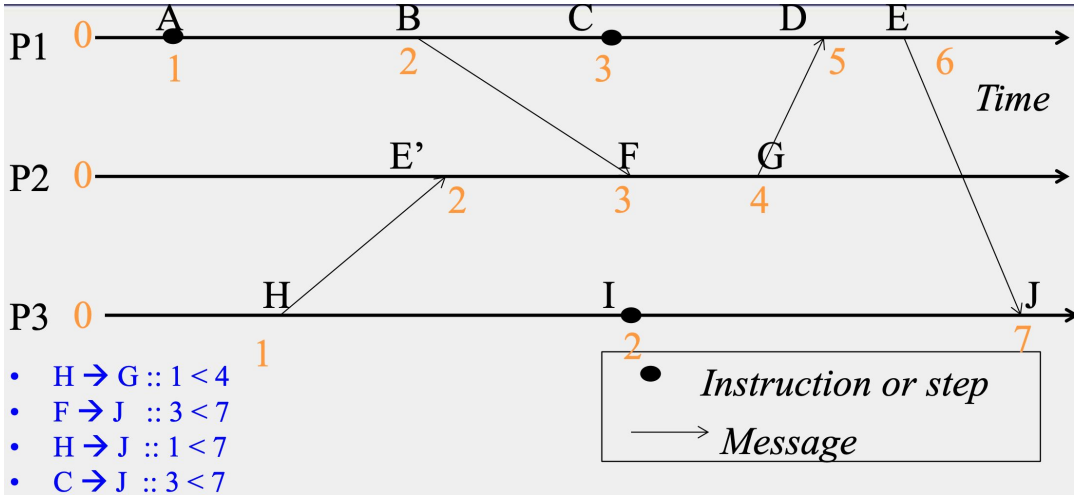
- **Instead of clock synchronization, timestamp events and order them**
- **Defines happens-before(→) relation with 3 rules:**
  - In a process (using local clock), A → B if time(A) < time(B)
  - If p1 sends m to p2, send(m) → receive(m)
  - If a → b and b → c then a → c (Transitivity)
- **Creates partial order among events**
  - Ex:



source: Distributed Systems by Indranil Gupta

Logical/Lamport Timestamps

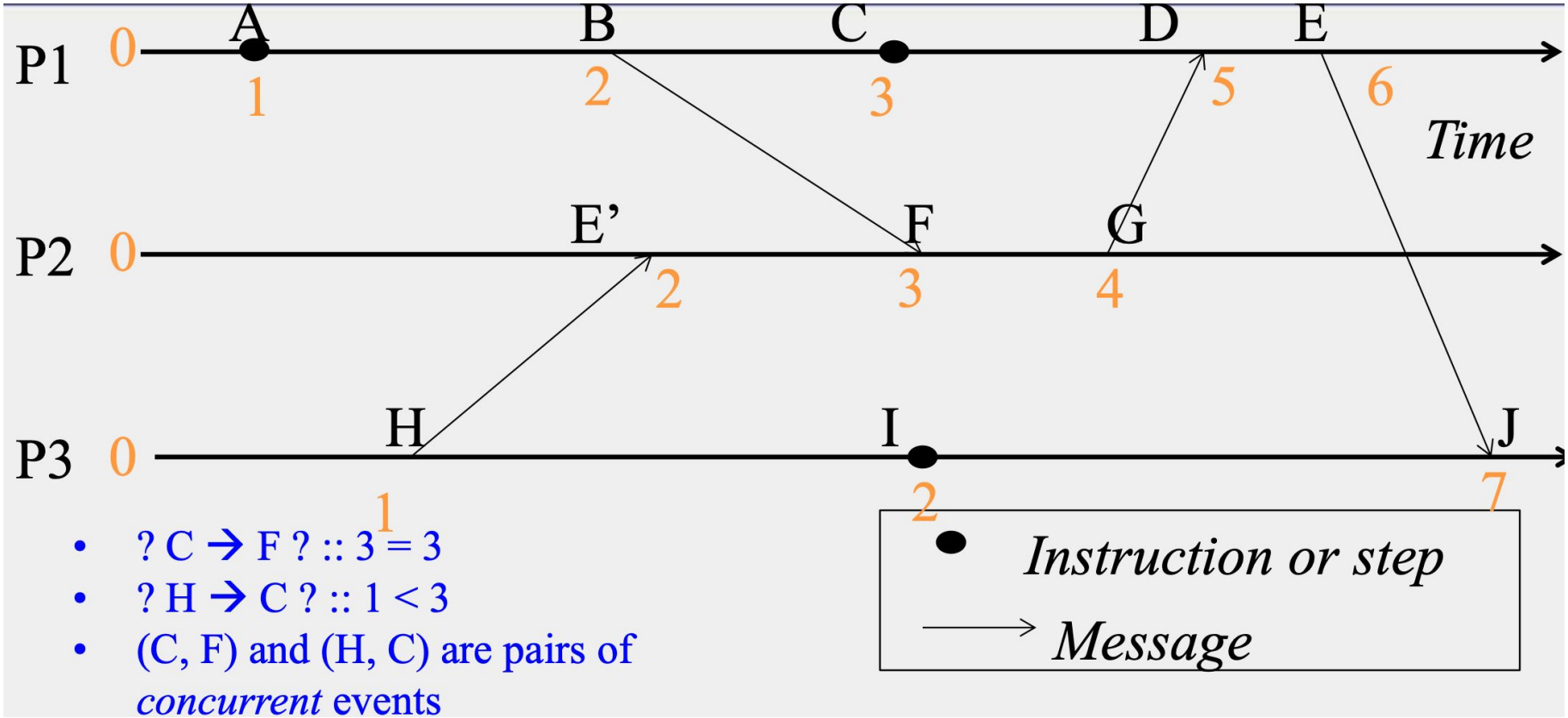
- **How to assign timestamps?**
  - Each process uses its own clock (counter)
    - Initial value is 0
    - Increases when a send msg. or an instruction happens
  - The counter is assigned to the event as its timestamp
  - A message (send event) carries its timestamp
  - For receiving side the counter is updated by;
    - MAX(local clock, message timestamp) + 1



source: Distributed Systems by Indranil Gupta

Logical/Lamport Timestamps

- **How to assign timestamps?**
  - Concurrent events



source: Distributed Systems by Indranil Gupta

Q/A