Distributed computing is a field of computer science that studies distributed systems

Computer Science is no more about computers than astronomy is about telescopes

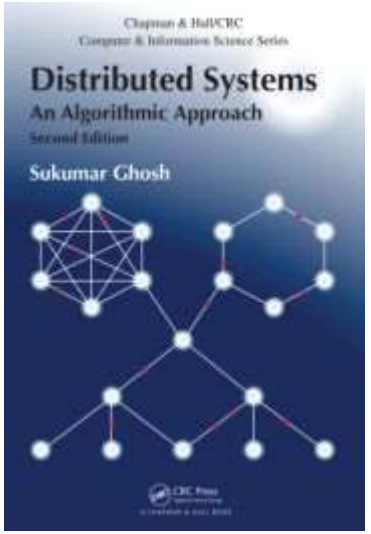
EWD

Distributed Computing

1. Introduction

2. Distributed algorithms

3. Distributed data



Distributed system

Collection of autonomous processors which can communicate with each other and which cooperate on a common goal

Some characteristics

- Autonomy and heterogeneity
- Geographical separation
- No shared memory
- No common physical clock

Why?

Distributed system

The solution to availability, performance and scalability is to decentralize and replicate functions and data

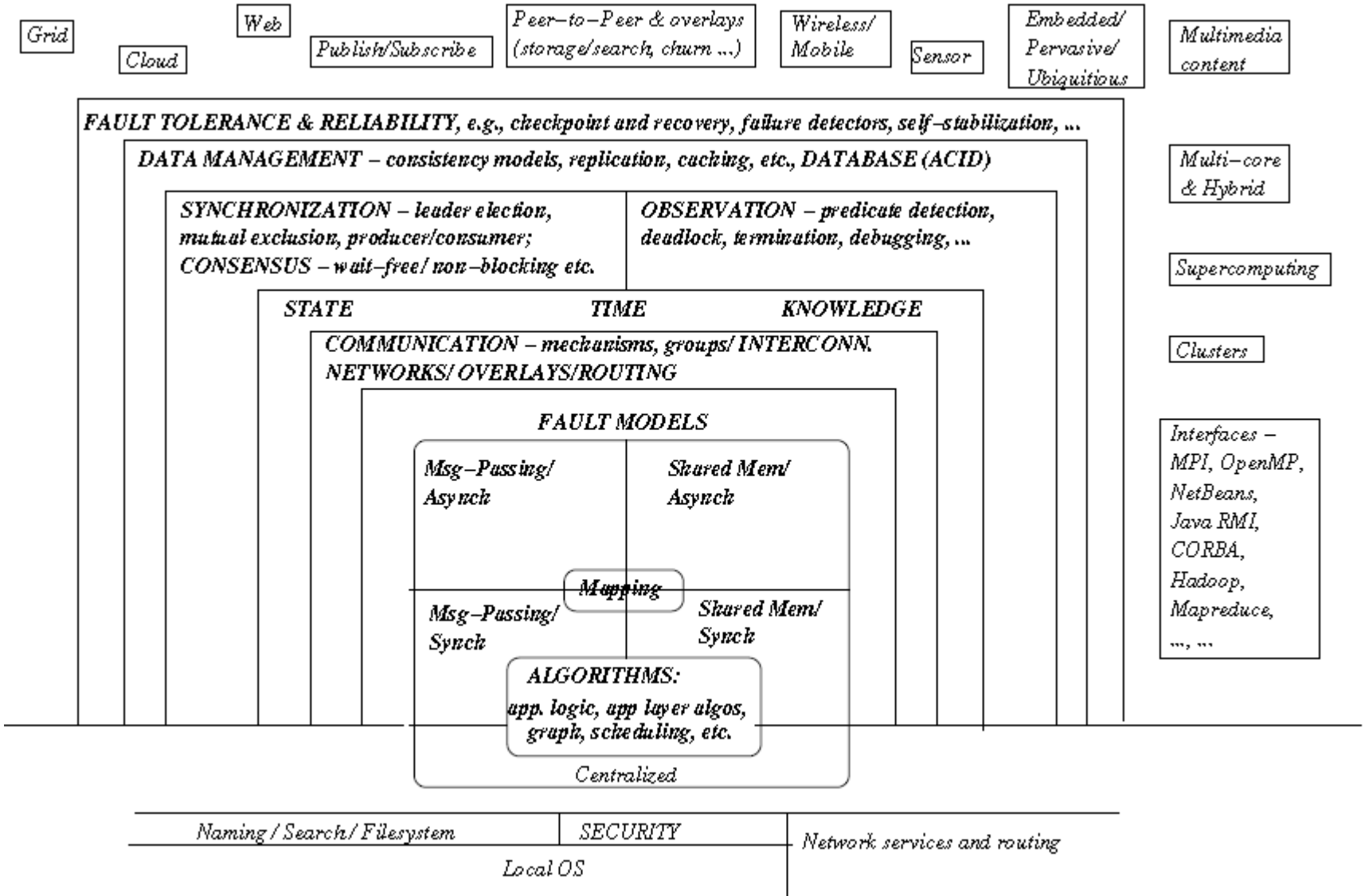
How do we coordinate the processors?

Distributed system

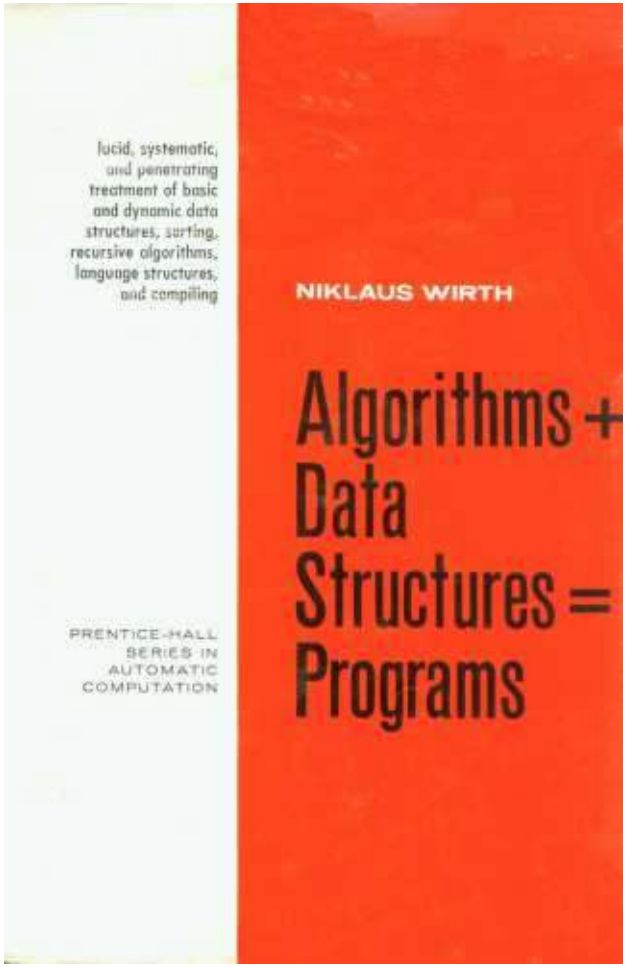
Can be characterized by the term uncertainty

This uncertainty is created by asynchrony, multiplicity of control flows, absence of shared memory and global time, failures, dynamicity, mobility

Distributed computing



Distributed program



Algorithms +

Data Structures +

Messages =

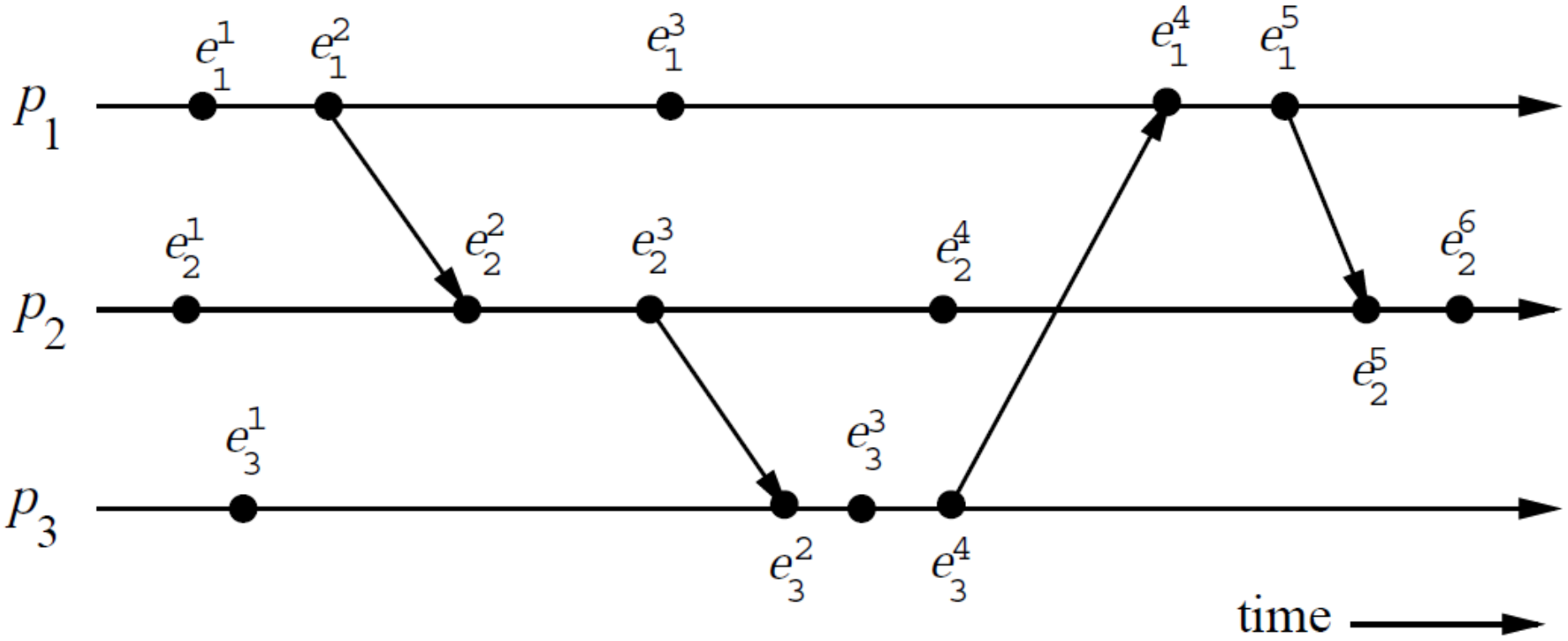
Distributed Programs

Distributed program

A distributed program is composed of a set of n processes: p_1, p_2, ..., p_i , ..., p_n

- The processes do not share a global clock that is instantaneously accessible to them
- The processes do not share a global memory and communicate solely by passing messages
- Process execution and message transfer are asynchronous
- Without loss of generality, we assume that each process is running on a different processor

Distributed execution



The (partial ordering) "happened before" relation →

Control algorithm

A distributed application execution represents the logic of the application (e.g. update of a distributed database)

In most cases, a distributed *control algorithm* also needs to be executed in order to perform various auxiliary functions, such as creating a superimposed spanning tree, detecting deadlocks or achieving consensus among the processes

Research papers

Gray, J., and Metz, S. (1983). **Solving the problems of distributed databases**. *Data Communications*, October, pp. 183-192.

1998 Turing award - For seminal contributions to database and transaction processing research and technical leadership in system implementation.

http://amturing.acm.org/award_winners/gray_3649936.cfm

https://sigmodrecord.org/publications/sigmodRecord/0806/p33.stonebraker.pdf

ACM Turing award winners

1. Dijkstra (1972)
2. Rabin (1976)
3. Hoare (1980)
4. Lampson (1992)
5. Gray (1998)
6. Rivest, Shamir and Adleman (2002)
7. Liskov (2008)
8. Lamport (2013)
9. Diffie and Hellman (2015)

EWD prize in distributed computing

1. Time, clocks, and the ordering of events in a distributed system
2. Distributed snapshots: Determining global states of distributed systems
3. Self-stabilizing systems in spite of distributed control
4. Reaching agreement in the presence of faults
5. Impossibility of distributed consensus with one faulty process

ACM Symposium on Principles of Distributed Computing (PODC)
EATCS Symposium on Distributed Computing (DISC)

EWD prize in distributed computing

6. Another advantage of free choice: Completely asynchronous agreement protocols
7. Randomized byzantine generals
8. Consensus in the presence of partial synchrony
9. Unreliable failure detectors for reliable distributed systems
10. Defining liveness

ACM SIGOPS hall of fame award

1. Chord: A scalable peer-to-peer lookup service for Internet applications
2. Time, clocks, and the ordering of events in a distributed system
3. Distributed snapshots: Determining global states of a distributed system
4. Implementing fault-tolerant services using the state machine approach: a tutorial

ACM SIGOPS hall of fame award

5. The part time parliament
6. Viewstamped replication: A new primary copy method to support highly-available distributed systems
7. Managing update conflicts in Bayou, a weakly connected replicated storage system
8. Dynamo: Amazon’s highly available key-value store

Real world issues

IEEE local area networks

Google infrastructure

P2P networks

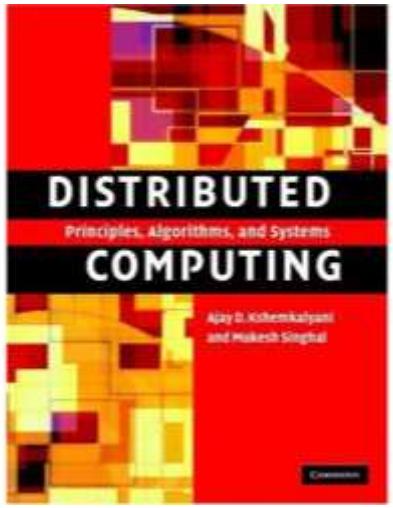
Distributed storage systems

Distributed Computing

1. Introduction

2. Distributed algorithms

3. Distributed data



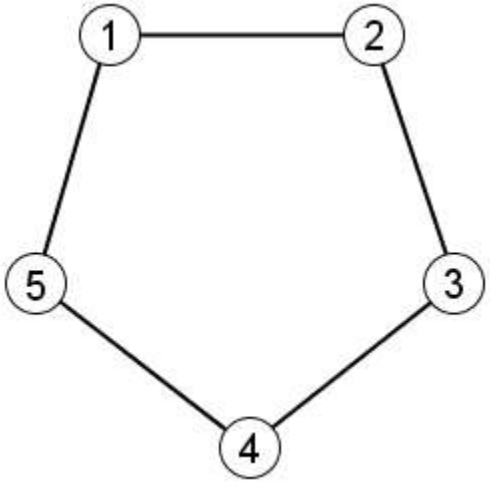
Characteristics of algorithms

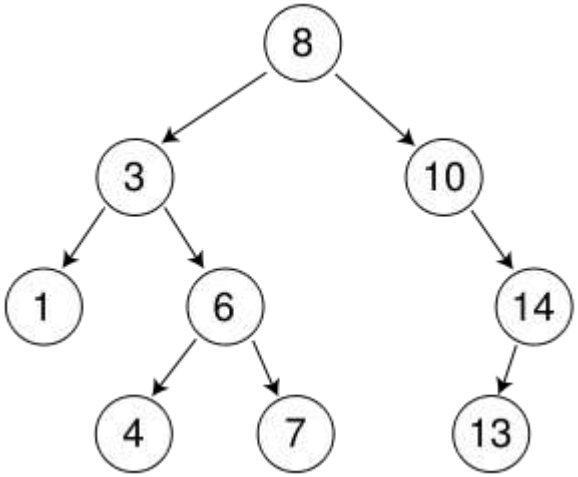
Hypothesis

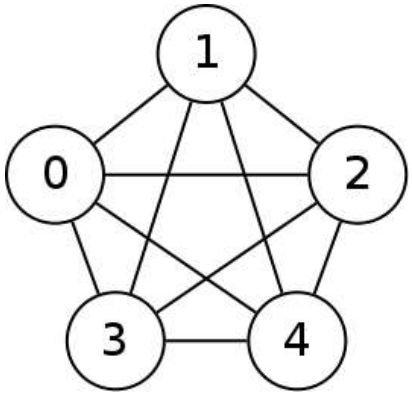
- Network topology
- Communication channels service model
- Failure model
- Synchrony model

Network topology

A graph is a representation of a set of processes where some pairs of processes are connected by communication channels







Communication channels

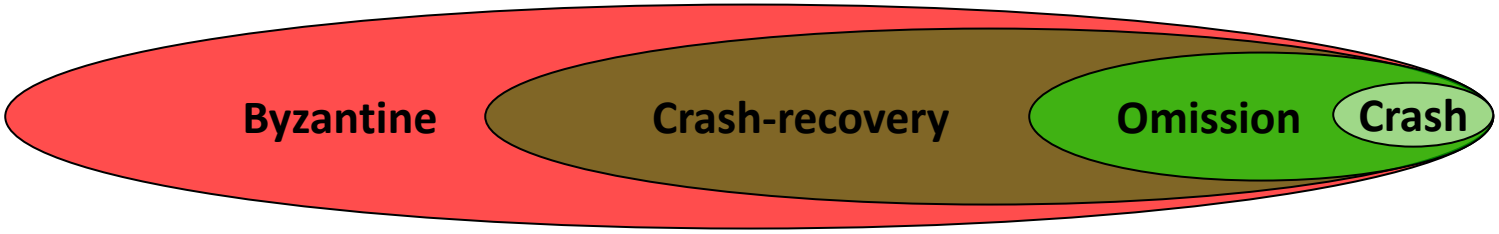
There are several models of the service provided by (reliable) communication channels

- Non-FIFO (set)
- FIFO (queue)
- Causal ordering

Message ordering paradigms

Failure model

A failure model specifies the manner in which the components (processes and channels) of the system may fail



Everything fails all the time

Werner Vogels - Amazon VP and CTO

Communications of the ACM, February 2020, Vol. 63 No. 2, pp. 95-96

A distributed system is one in which the failure of a computer you didn’t even know existed can render your own computer unusable

L. Lamport, email Message-Id: <8705281923.AA09105@jumbo.dec.com>

Synchrony model

Synchronous systems

- upper bound on message delay
- known bounded drift rate of clock
- known upper bound for process to execute a logical step

Asynchronous systems

- above criteria not satisfied

Physical clocks

UTC is broadcast

- Land-based short wave radio accuracy of +/- 1 ms
- GPS accuracy of +/- 0.1 µs

What problems does this solve/not solve?

- too coarse for synchronization of many kinds of events (e.g. 1 000 000 ops/s)

Causality

The concept of causality between events is fundamental to the design and analysis of distributed computing

- Usually causality is tracked using physical time, but in distributed systems it is not possible to have a global physical time
- As asynchronous distributed computations make progress in spurts, the logical time is sufficient to capture the fundamental monotonicity property associated with causality in distributed systems

Characteristics of algorithms

Message complexity O(*f(n)*)

- Number of messages exchanged
- Size of messages

Space complexity

Time complexity

- Number of steps or rounds needed

Characteristics of algorithms

Symmetry (distribution degree)

- Text asymmetry
- Text symmetry
- Strong symmetry (anonymous, uniform)
  - Ethernet

Mutual exclusion

Concurrent access of processes to a shared resource or data (critical section) is executed in mutually exclusive manner

Two basic approaches for distributed mutual exclusion:

- Token based
- Permission based

Requeriments of algorithms

- Safety: something bad **never** happens

  (two processes are not in CS at the same time)
- Liveness: something good **eventually** happens (every request to enter in CS will be granted)

- Fairness

Leader election

All processes must agree on a common distinguished process (leader)

- The leader is often referred to as the “master” or the “primary”

Distributed algorithms are typically not completely symmetrical

- There does not exist a deterministic leader election algorithm for anonymous rings, however the algorithm can be uniform

Termination detection

A distributed computation is globally terminated if every process is locally terminated and there is no message in transit between any processes

- Local condition: each process is in the passive state in which it has finished its computation and will not restart any action unless it receives a message
- Global condition: there is no message in transit between any pair of processes

Deadlock detection

A deadlock represents a system state where a subset of the processes are blocked on one another, waiting for a reply from other processes in the subset

- Local condition: each deadlocked process is locally blocked
- Global condition: the deadlocked process will not receive a reply from some process(es) in the subset

Requeriments of algorithms

Stable property sp (termination or deadlock)

$sp \rightarrow \Box\, sp$

- Safety (No false detection)

  $\Box\,(detect \rightarrow sp)$

- Liveness (No undetected sp)

  $\Box\,(sp \rightarrow \Diamond\, detect)$

Formal proofs

So-called natural language is wonderful for the purposes it was created for, such as to be rude in, to tell jokes in, to cheat or to make love in (and Theorists of Literary Criticism can even be content-free in it), but it is hopelessly inadequate when we have to deal unambiguously with situations of great intricacy, situations which unavoidably arise in such activities as legislation, arbitration, mathematics or programming.

EWD

Global state

A collection of the local states of the processes and the communication channels

- The state of a process is defined by the contents of processor registers, stacks, local memory, etc., and depends on the local context of the distributed application
- The state of a channel is given by the set of messages in transit in the channel

Snapshot recording

Recording the global state of a distributed system on-the-fly is an important paradigm

- In detection of stable properties (e.g. termination and deadlock), a global state is examined for certain properties
- For failure recovery, a global state (checkpoint) is periodically saved, and recovery from a failure is done by restoring the system to the last saved global state

Fault tolerance

A mechanism that masks or restores the expected behavior of a system following the occurrence of faults

The goal is to avoid service failures in the presence of faults

Self-stabilization

Regardless of its current state, the system is guaranteed to converge to a legitimate state in a bounded amount of time by itself without any outside intervention

Consensus

Many forms of coordination require the processes to exchange information to negotiate with one another and eventually reach a consensus, before taking application-specific actions, despite failures

- Commit decision in a database system
- Atomic broadcast

Consensus

- **Agreement** - All non-faulty processes must agree on the same value
  - *Validity*: If all the non-faulty processes have the same initial value, then the agreed upon value by all the non-faulty processes must be that same value
  - *Integrity*: if a process decides some value then that value must have been proposed by some process
- **Termination** - Each non-faulty process must eventually decide on a value

Consensus

There is a t-resilient synchronous protocol which solves the Byzantine Generals' problem in t rounds if n>3t+1 in the *oral-messages* model

In the *written-messages* model there are protocols that can tolerate n=t+1

t is the number of failures and n is the number of processes

Consensus

The problem is not solvable in **asynchronous** systems even if one process *can* fail by crashing

FLP

A distributed system is asynchronous if:

- there is no bound on the transmission delay of messages
- there is no bound on the processing time of a piece of code

Consensus

The FLP result states that under the model's assumptions, no algorithm can always reach consensus in bounded time

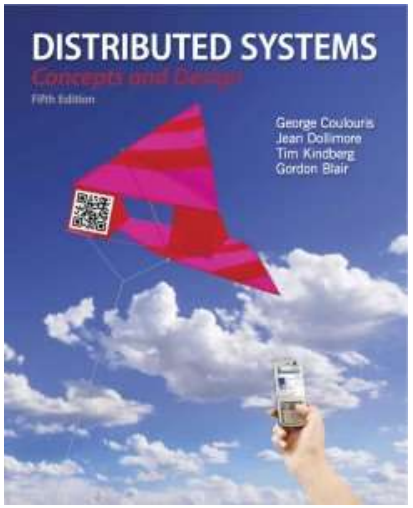
- Randomized algorithms
- Partially synchronous systems
- Unreliable failure detectors

Distributed Computing

1. Introduction

2. Distributed algorithms

3. Distributed data



Distributed data

- Replicated data
- Eventual consistency
- Blockchain

Paxos

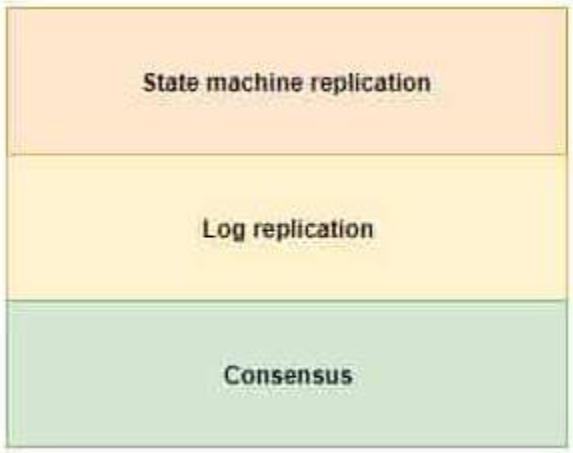
The Paxos algorithm is designed for implementing a fault-tolerant distributed system

At its heart is the “synod” consensus algorithm

https://seastheday.gr/greek-islands/paxos-the-ionian-paradise/

Paxos

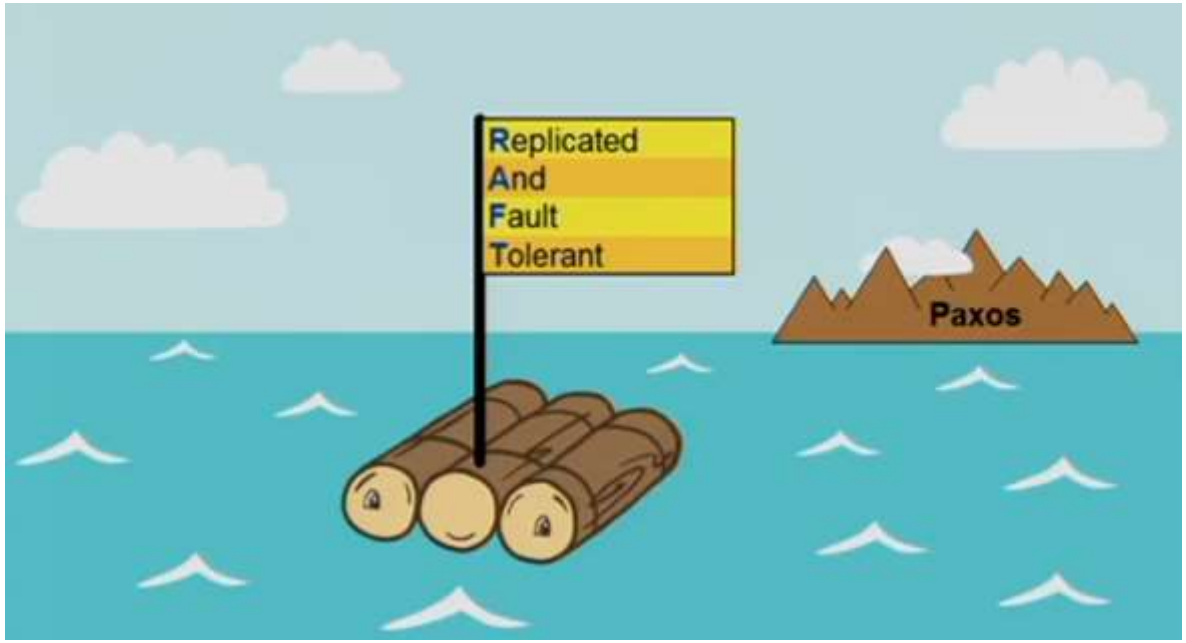
The complete Paxos algorithm is obtained by the straightforward application of consensus to the state machine (replication) approach for building a distributed system



Guarantees safety

Consensus

- Raft

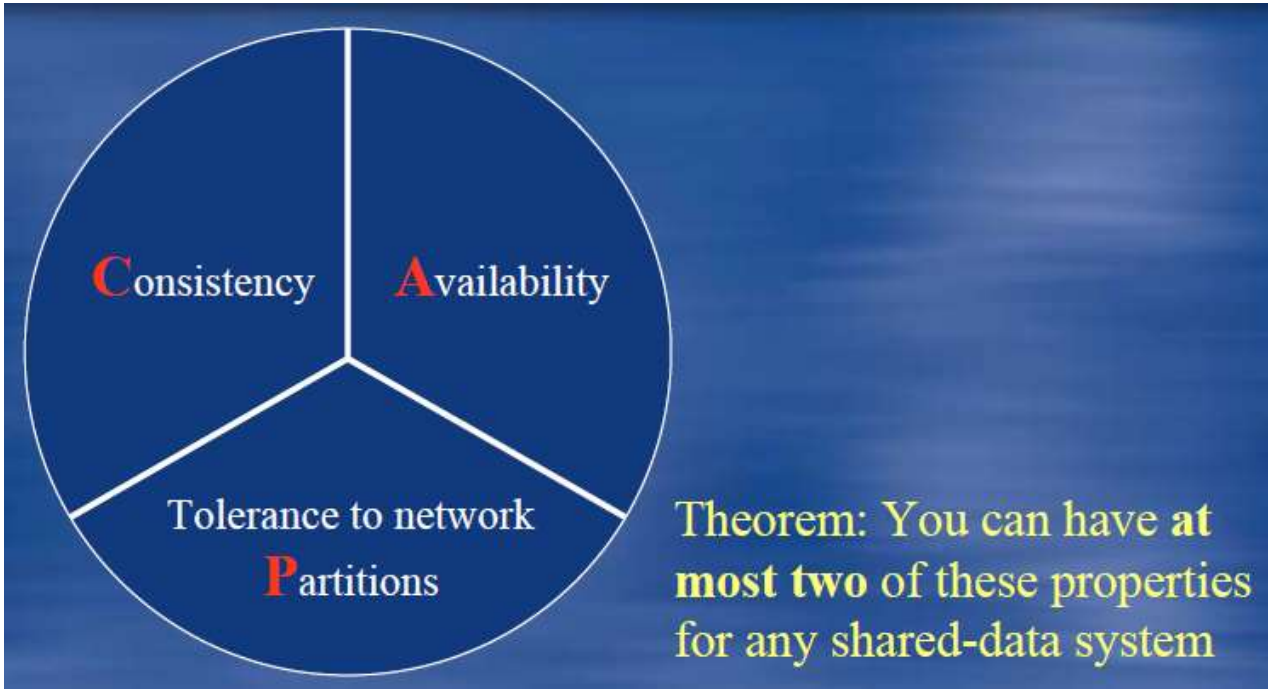


- VR and PBFT

Consensus

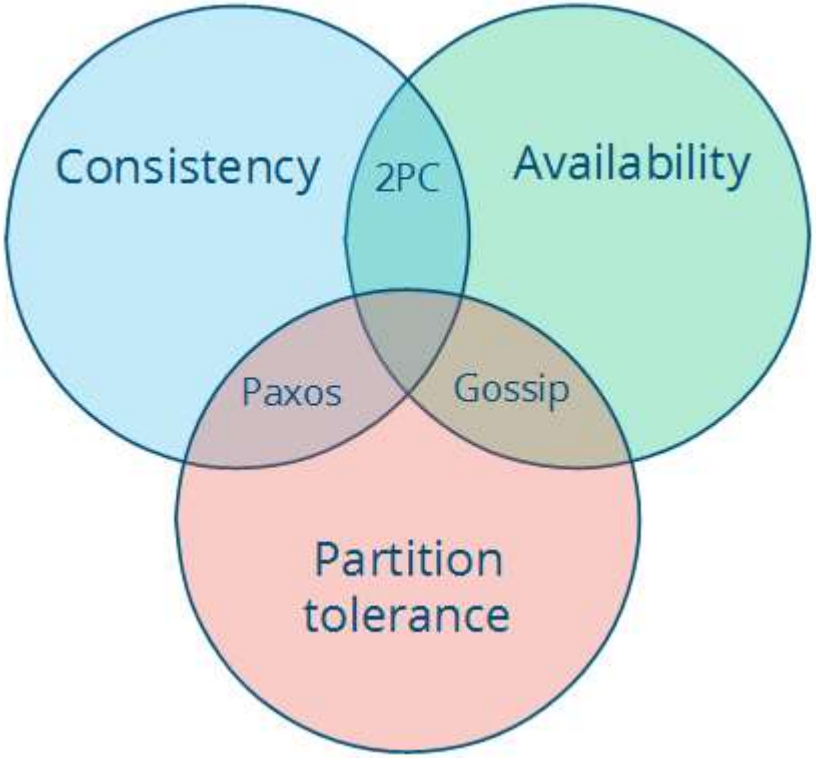
- Bitcoin Proof-Of-Work
- Hashgraph

CAP theorem



$P \Rightarrow \sim(C \,\&\, A)$

Replicated data



Consistency

ACID

Atomicity - Consistency – Isolation – Durability

Global (mutual) consistency must be ensured across a distributed database with replication

Availability

Applications enabling users to share documents over the Internet

Optimistic + conflict resolution

Consistency vs. latency

BASE

Basically Available - Soft state - Eventually consistent

Six different consistency guarantees

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Bounded Staleness	See all “old” writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader

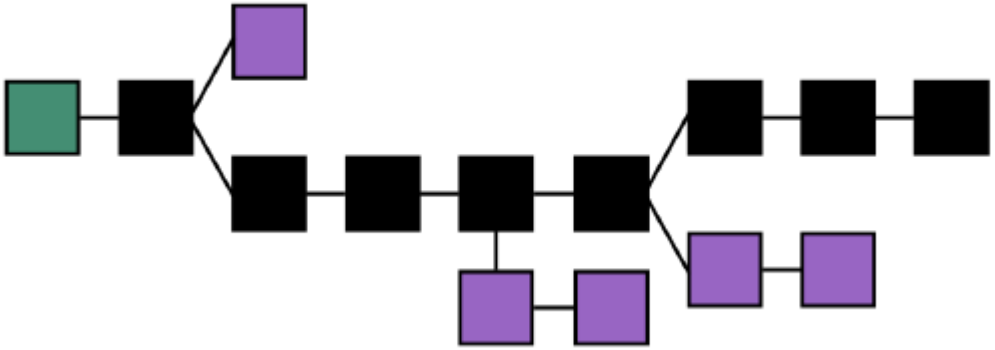
Eventual consistency

Dynamo

Blockchain

A replicated tamper-proof database of blocks that keeps public records in an append-only fashion

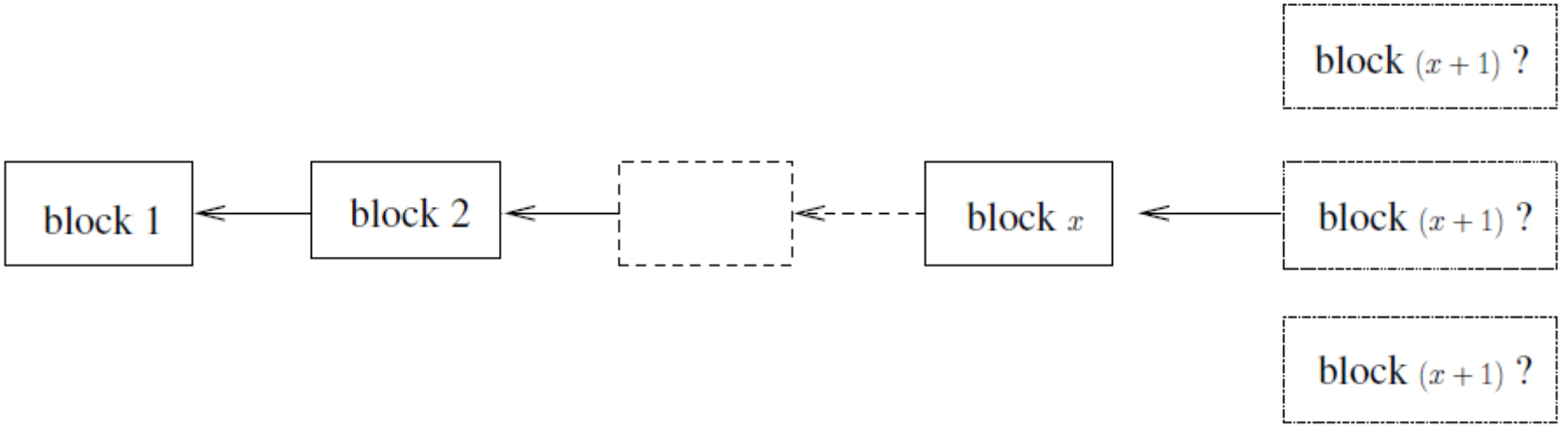
- Persistant
- Auditable



Chris Hammerschmidt , Consensus in Blockchain Systems. In Short. 2017.

Blockchain

A crucial issue for the processes is to agree on the next block to add



Rajsbaum & Raynal, 60 years of mastering concurrent computing through sequential thinking. 2020.

Research papers

Herlihy, M. (2019) **Blockchains from a Distributed Computing Perspective**, *Communications of the ACM*, Vol. 62, No. 2, pp. 78-85.

References

Acknowledgement of the sources will be provided where possible/practical

The slides for this lecture are based on material from: