The City College of New York

CSC 36000:
Modern Distributed Computing *NextGen with AI Agents*

By Saptarashmi Bandyopadhyay
Email: sbandyopadhyay@ccny.cuny.edu
Assistant Professor of Computer Science
City College of New York and Graduate Center at City University of New York

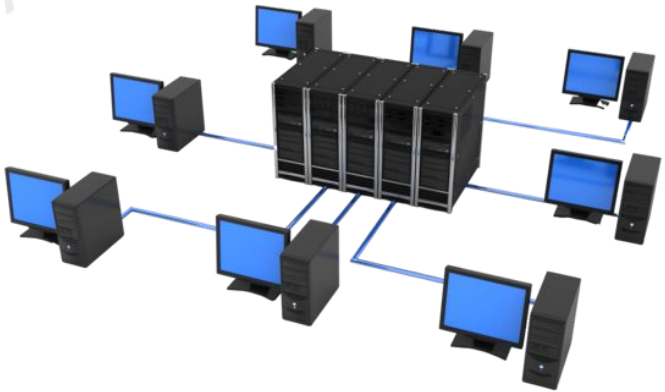
Class Website:

https://saptab.github.io/modern-distributed-computing-with-AI-Agents

Distributed Computing Systems Models

The Abstract World of Distributed Computation

Fundamentally, distributed systems are made up of many independent nodes (computers) connected together in a physical or digital network



These systems work together and communicate to accomplish a common goal

- Training an AI model
- Balancing an energy grid
- Coordinating self-driving cars

Properties of Distributed Systems

Not all distributed systems operate under the same rules!

They are often defined by three core properties of ***time***, ***failure***, and ***communication***.

- Synchronous vs. Asynchronous
- Benign Failures vs. Byzantine Failures
- Shared Memory vs. Message Passing

Depending on the model they operate under, we can prove different things about their ***correctness*** and ***reliability***.

Today's Lecture

Time and Message Delivery
- Synchronous
- Asynchronous

Failure Models
- Benign
- Byzantine

Communication
- Shared Memory
- Message Passing

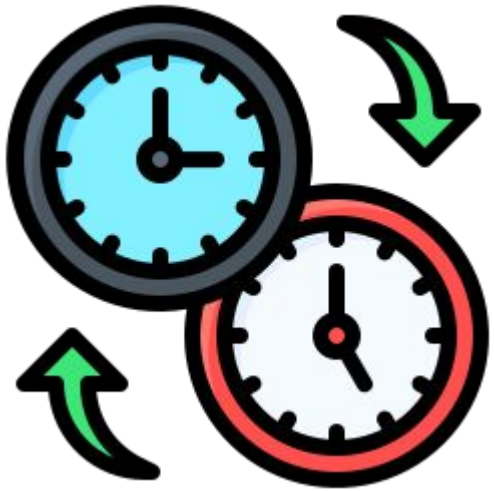
Time and Message Delivery

Synchronous Model

This model operates under strong and idealized assumptions that aren't feasible in the real world, but make algorithm analysis and design simpler.

A **Synchronous Model** assumes:

- **Shared Global Clock**: Every node agrees on a time step without needing to communicate it, allowing algorithms to proceed in lock-step
- **Known Maximum Message Delay**: There exists an upper bound Δ such that a message sent by one node to any other node is at time t is *guaranteed* to be received by time t + Δ.



Real-World Examples

Because of the strict rules Synchronous models assume, they are often found only in tightly controlled localized environments:

- Multi-core processors
- Industrial Controllers
- Automotive Computers
- Drone Controllers

Systems that benefit a lot from low-latency are likely to use a more synchronized model!

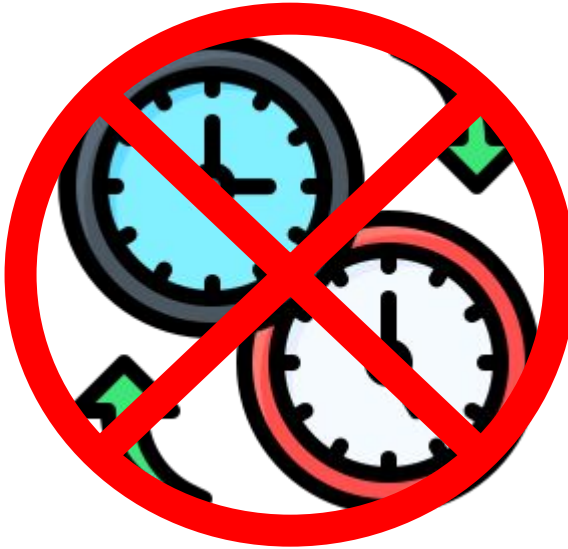
Question: Self-Driving Cars

Suppose you have three self-driving cars (C1, C2, and C3) communicating in a Synchronous model. Their maximum delay is $\Delta = 50$ milliseconds.

Each car can send a PREPARE message to warn the other cars of an upcoming hazard. When a car receives a PREPARE message, it will send a READY message back.

If C1 sends a PREPARE command to C2 and C3, by which time will C1 be guaranteed to know that C2 and C3 are listening properly?

Asynchronous Models



Usually we can't assume each node is timed in lock-step with *all* the other nodes.

Asynchronous Models have inherent uncertainty of timing between distributed nodes which may be in far-off locations. They assume:

- **No Global Clock**: Each node might have its own local clock, but there is no guarantee they run at the same rates or can be synchronized
- **Unbounded Message Delay**: The only guarantee is that a message between functioning nodes will be delivered. It could take any amount of time!

Real-World Examples

Asynchronous Models are helpful when delays can happen, but a message is still expected to reach its destination *eventually*.

- Independent AI Agents
- Email Servers
- Instant Messaging

Messages might be delayed or arrive out-of-order!

Consensus

The weakness of the assumptions in the Asynchronous model make it easy to create in the real world, but can also make them unreliable and undependable.

The FLP Impossibility Result, proved by Fischer, Lynch, and Paterson in 1985, shows it is *impossible* to guarantee that distributed systems can agree (reach “consensus”) on a single value if even *one* process is allowed to crash!

Failure Models

Benign Failures

Simple failures where nodes stop communicating or leave out (“omit”) information are **benign** failures.

They don’t actively try to deceive or send incorrect information.

Examples:

- Crash-Stop failures
- Omission failures
  - Send omission
  - Receive omission

Real-World Example

In a AI-managed battery-grid system, an AI Agent managing a battery might be able to *compute* that a battery is done charging, but due to the output buffer being full, it doesn't tell the other AI Agents that it doesn't need more power.

What kind of benign failure is this?

Byzantine Failures

These failures are more general and severe.

Not only can nodes not communicate, but they can now *lie* and *act maliciously*.

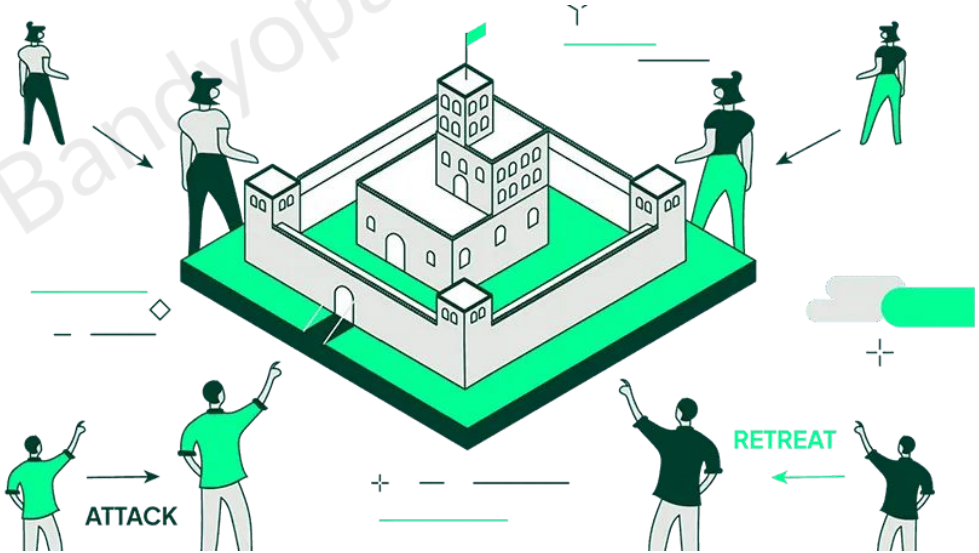
We go from simply trying to detect and recover from failures to resolving communication disputes within and between nodes.

In order to tolerate k crash failures, only k + 1 replicas are needed, but to tolerate k *Byzantine* failures, 2k +1 replicas are needed to reach a majority for the correct message, and often 3k + 1 replicas are needed in a system with asynchronous components.

Real-World Examples

Byzantine Failure Tolerance (BFT) is very computationally expensive, and is usually only present in high-stakes systems where it is absolutely needed, like:

- AI surgical assistants
- Aerospace Control Systems
- Military Communications



The name Byzantine comes from the Byzantine Generals Problem, where several generals need to agree on a common attack plan, but some generals could be traitors! How do you agree on a plan?

Questions?