分布式系统模型

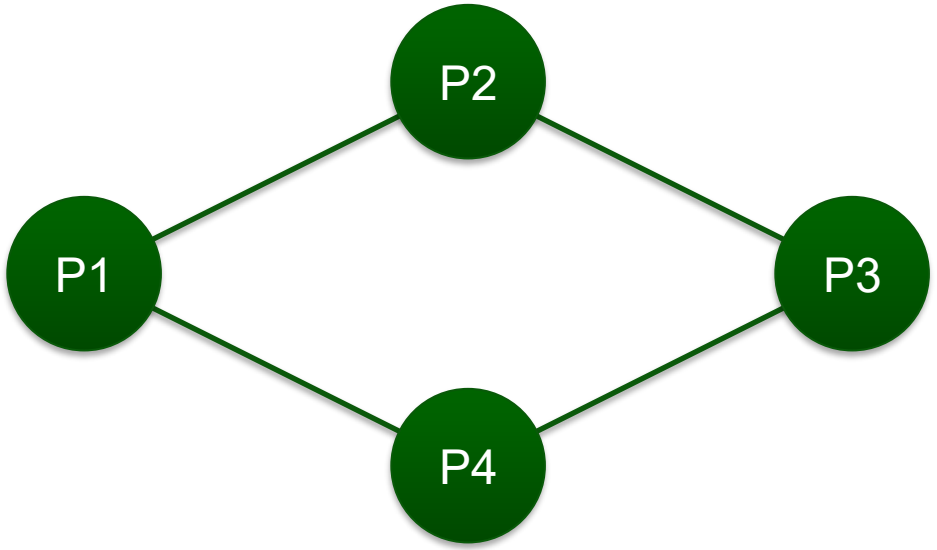
本节内容

- **介绍分布式系统的模型**
  - 消息
  - 同步
  - 异步
    - I/O自动机模型
  - 系统性质
  - 不可能性

分布式系统图模型

- **分布式系统可抽象为有向图 G = (V, E)**
  - V：节点集合，每个节点表示一个顺序执行的进程
  - E：通信通道集合，表示进程之间的通信路径
- **节点之间通过消息或共享内存进行通信**

分布式系统拓扑示意



通信模型

- **两种典型的通信模型**
  - 消息传递模型
  - 共享内存模型

基于消息的模型

- 消息沿着称为通道的有向边传播
- 通信被假定为点对点的
- 通道可以是可靠的或不可靠的
- 在可靠通道中，消息的丢失或损坏被排除在外

同步/异步模型

同步模型形式化定义

- **分布式系统表示为有向图 *G = (V, E)*，**
  - ***V***为节点，*E*表示节点的通信通道
  - **G** 中的每个节点 i，使用符号 out-nbrs(i)来表示 i 的“出邻居”
    - 即在有向图 ***G*** 中从 *i* 出发有边指向的那些节点；
  - **G** 中的每个节点 i，使用符号 in-nbrs(i)来表示 i 的“入邻居”
    - 即在有向图 ***G*** 中有边指向 *i* 的那些节点。
  - *distance(i, j)*，表示从 *i* 到 *j* 的最短有向路径的长度（如果存在的话）
  - *distance(i, j)* = ∞，表示没有从 *i* 到 *j* 的通路
  - 消息集***M***，***null***表示没有消息

同步模型形式化定义

- **与每个节点 $i \in V$ 相关联的是一个进程，由以下组件组成：**
  - $states_i$：一个（不一定有限的）状态集合
  - $start_i$： $states_i$的一个非空子集，称为起始状态或初始状态
  - $msgs_i$：一个消息生成函数，将 $states_i \times$out-nbrs(i)映射到 M∪{null}中的元素
  - $trans_i$：一个状态转换函数，将 $states_i$和由 in-nbrs(i) 索引的 M∪{null} 元素向量映射到 $states_i$
- **每个进程都有一组状态**
  - 其中有一个起始状态子集。
- **状态集合不一定是有限的**

同步模型形式化定义

- **整个系统的执行，从所有进程处于任意起始状态且所有通道为空开始**
- **进程以同步的方式重复执行以下两个步骤：**
  1. 将消息生成函数应用于当前状态，生成要发送给所有出邻居的消息，并将这些消息放入相应的通道中。
  2. 将状态转换函数应用于当前状态和接收到的消息，以获取新状态。然后从通道中移除所有消息。

     为什么要“从通道中移除所有消息”?
- **这两个步骤的组合，称为一轮**
- **模型是确定性的**
  - 即消息生成函数和状态转换函数都是单值函数

同步系统：Round 执行示意

更直观的理解

- **如果你没有理解**
  - 阅读参考书的相关章节
- **如果你不想花费时间，理解下面直观的性质**
  - 有上界

同步系统的直观理解

- 同步模型假设存在全局时间上界
- 消息传输延迟 ≤ Δ
- 进程执行速度存在上界
- 系统可以以 round（轮）为单位推进

异步模型

- **异步模型的形式化表示**
- **输入/输出自动机模型（ input/output (I/O) automaton model）**
  - 简单的**状态机**模型
  - 状态转换与动作相关联
    - 动作被分类为**输入(in)、输出(out)**或**内部(internal)**动作
    - 输入和输出用于与自动机的环境进行通信
    - 内部动作仅对自动机本身可见
    - 输入动作假设不受自动机控制，它们只是从外部到达
    - 自动机本身指定应执行哪些输出和内部动作

I/O自动机形式化定义

- **一个I/O自动机A，简称为自动机，由五个部分组成：**
  - ***sig(A)***，一个签名
  - ***states(A)***，一个（不一定有限的）状态集合
  - ***start(A)***，***states(A)***的一个非空子集，称为起始状态或初始状态
  - ***trans(A)***，一个状态转换关系，
    - 其中$trans(A) \subseteq states(A) \times acts(sig(A)) \times states(A)$；
    - 这必须满足一个性质，即对于每个状态s和每个输入操作π，都存在一个转换$(s, \pi, s') \in trans(A)$
  - ***tasks(A)***，一个任务划分，
    - 它是***local(sig(A))***上的等价关系，具有至多可数个等价类
- ***acts(A)*作为*acts(sig(A))*的简写，同样地，*in(A)*，*out(A)*等也是如此**

状态转换

- **转换或步骤**
  - 将***trans(A)***中的一个元素***(s, π, s')***称为*A*的一个**转换**或**步骤**。
  - 根据动作π是输入动作、输出动作等，转换***(s, π, s')***被称为**输入转换**、**输出转换**等
- **如果对于某个特定状态*s*和动作*π*，**
- ***A*存在一个形式为*(s, π, s')*的转换，那么*π*在*s*中是可执行的。**

任务划分

- **自动机A的任务划分*task(A)***
  - 自动机内部“任务” 的抽象描述
  - 对于一个分布式系统， ***task(A)***可以用来描述节点

I/O自动机执行

- **自动机的执行**
  - A的一个执行片段可以是，
    - 一个有限序列，如$s_0, \pi_1, s_1, \pi_2, \ldots, \pi_n, s_n$
    - 也可以是一个无限序列，如$s_0, \pi_1, s_1, \pi_2, \ldots, \pi_n, s_n, \ldots$
      - 其中交替出现*A*的状态和动作，且对于每一个$k \geq 0$，$(s_k, \pi_{k+1}, s_{k+1})$是*A*的一个转移
  - 用*execs(A)*表示*A*的所有执行的集合
- **自动机的trace**
  - I/O自动机的外部行为
  - *A*的一个执行*α*的迹，记作***trace(α)***，是*α*中所有外部动作(*in, out*)组成的子序列
  - *β*是*A*的某个执行的迹，则我们说*β*是*A*的一个**迹**
  - 用***traces(A)***表示A的所有迹的集合。

异步系统的直观理解

- 消息传输延迟没有上界
- 无法区分网络延迟与节点崩溃
- 系统执行顺序完全由调度决定
- 很多真实分布式系统更接近异步模型

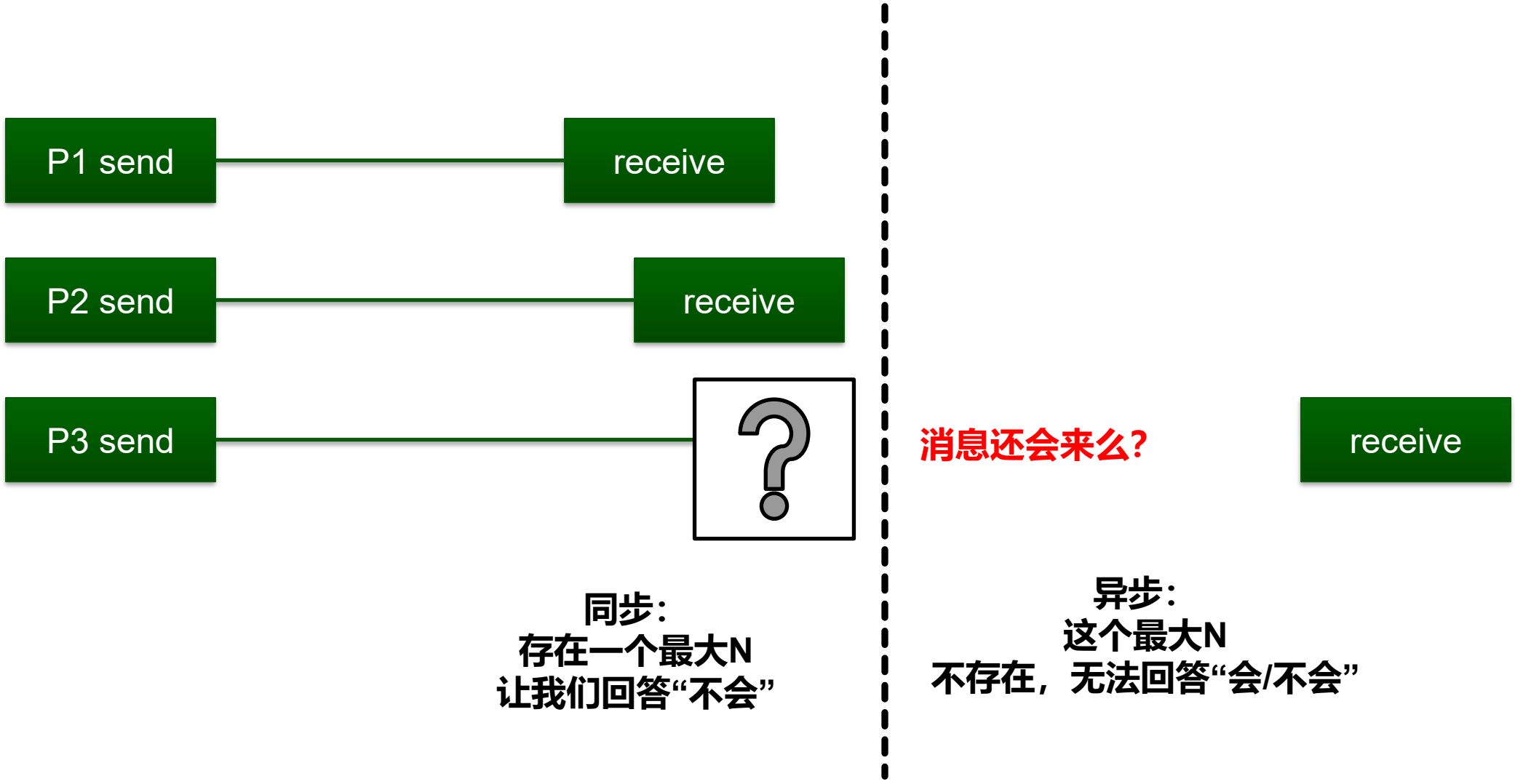
异步模型直观理解

- **异步模型直观的性质**
  - 消息/通信延迟没有时间上界
- **无法区分，消息/通信延迟与丢失**

对比和同步模型的不同

- **异步（I/O自动机）模型**
  - 抽象层次更高
    - 可以把同步模型为化归为一种特殊的I/O自动机模型
    - 可以描述消息模型/共享内存模型
  - $Output(m)_{i,j}$，$Input(m)_{i,j}$之间可以**N个step**，**N无上界**
- **而，对应到同步模型，**
  - $Output(m)_{i,j}$，$Input(m)_{i,j}$在**1 round**内

异步系统：消息延迟示意


P1 send
receive
P2 send
receive
P3 send
?
消息还会来么?
receive
同步：
存在一个最大N
让我们回答“不会”
异步：
这个最大N
不存在，无法回答“会/不会”

例：一种典型的分布式算法

- **共识**

什么是共识问题

- 多个节点需要对某个值达成一致
- 节点之间通过消息通信
- 可能存在网络延迟或节点崩溃
- 目标：所有正确节点最终决定同一个值

为什么需要共识

- 复制状态机（State Machine Replication）
- 分布式数据库一致性
- 分布式锁
- 领导者选举

共识需要满足的性质

- Agreement（一致性）：所有节点决定相同值
- Validity（有效性）：决定值必须来自某个节点提议
- Termination（终止性）：所有正确节点最终决定
- Integrity（完整性）：节点只决定一次

共识（Raft、Paxos）工作在哪种模型?

- **异步（Asynchronous）**
- **消息传递（Message Passing）**
- **失败停止 + 崩溃恢复（Fail Stop + Crash Recovery ）**
  - 非拜占庭式的

同步模型 vs 异步模型

- 同步：消息延迟有上界，可以定义 round
- 异步：消息延迟无上界，没有统一时钟
- 同步系统更容易设计协议
- 真实系统通常介于两者之间（部分同步）

Lamport Happens-Before 关系

- 用于描述分布式系统中的因果关系
- 同一进程内：事件按程序顺序发生
- 消息发送 → 消息接收
- 通过传递闭包定义因果关系
- 是逻辑时钟与向量时钟的基础

共识算法工作的系统模型

- 经典共识算法：Paxos、Raft
- 运行在消息传递模型
- 节点可能发生 Fail-stop 故障
- 通常假设部分同步系统

为什么需要系统模型?

- 分布式系统存在不确定性：延迟、并发、故障
- 模型帮助我们定义系统假设（Assumption）
- 模型帮助我们理解系统性质（System Properties，Safety/Liveness）
- 不同模型影响算法可行性（Impossibility）
- 例如：同步 / 异步 / 部分同步

安全性（Safety）/活性（Liveness）

- **安全性（Safety）**
  - 系统不会发生错误的事情
    - Bad thing never happen
  - 例：
    - 不会出现两个 Leader
    - 不会提交不一致的数据
- **活性（Liveness）**
  - 系统最终会取得进展
    - Good thing eventually happen
  - 例
    - 请求最终得到响应
    - 共识最终达成

Impossibility

A Hundred Impossibility Proofs for Distributed Computing

Nancy A. Lynch *
Lab for Computer Science
MIT, Cambridge, MA 02139
lynch@tds.lcs.mit.edu

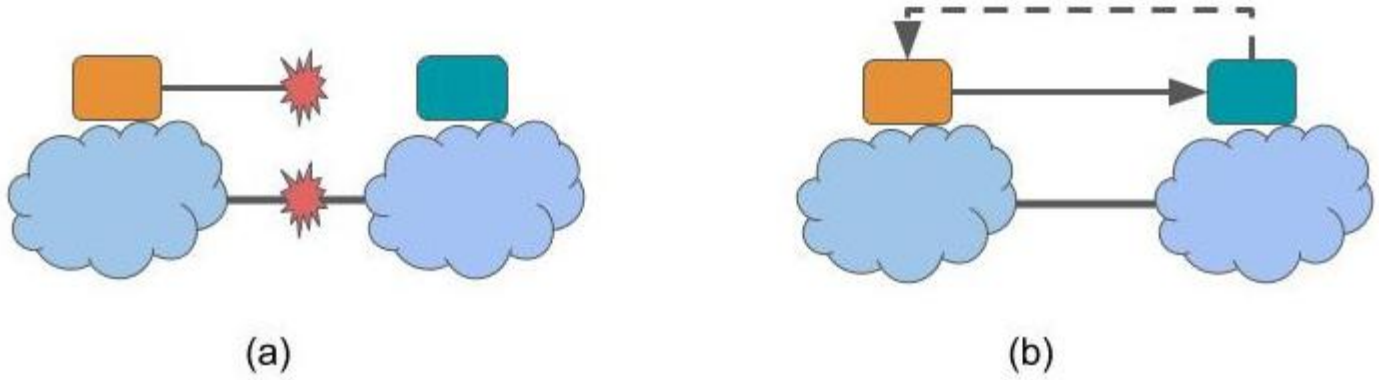
1 Introduction

This talk is about impossibility results in the area of distributed computing. In this category, I include not just results that say that a particular task cannot be accomplished, but also lower bound results, which say that a task cannot be accomplished within a certain bound on cost.

I started out with a simple plan for preparing this talk: I would spend a couple of weeks reading all the impossibility proofs in our field, and would categorize a tour of the impossibility results that I was able to collect. I apologize for not being comprehensive, and in particular for placing perhaps undue emphasis on results I have been involved in (but those are the ones I know best!). I will describe the techniques used, as well as giving some historical perspective. I'll intersperse this with my opinions and observations, and I'll try to collect what I consider to be the most important of these at the end. Then I'll make some suggestions for future work.

CAP Theorem

- **Consistency: Read must see the latest value of write.**
- **Availability: The system should be able to serve requests.**
- **Partition-tolerance: The system must handle network partitions (regardless of whether there actually is a partition or not).**

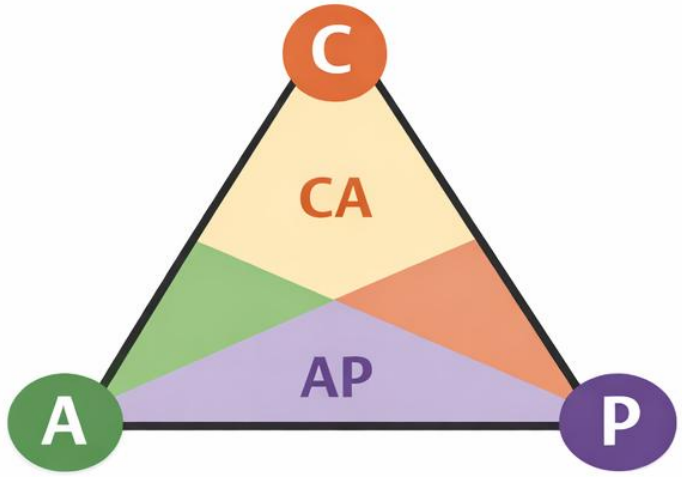


CAP Theorem

The CAP Theorem

Consistency (C)

All nodes have the same up-to-date data.



Availability (A)

Every request gets a response.

Partition Tolerance (P)

System works despite network partitions.

In a Partition, you can choose only 2 out of 3:

- ✓ Consistency + Availability (CA)
- ✓ Consistency + Partition Tolerance (CP)

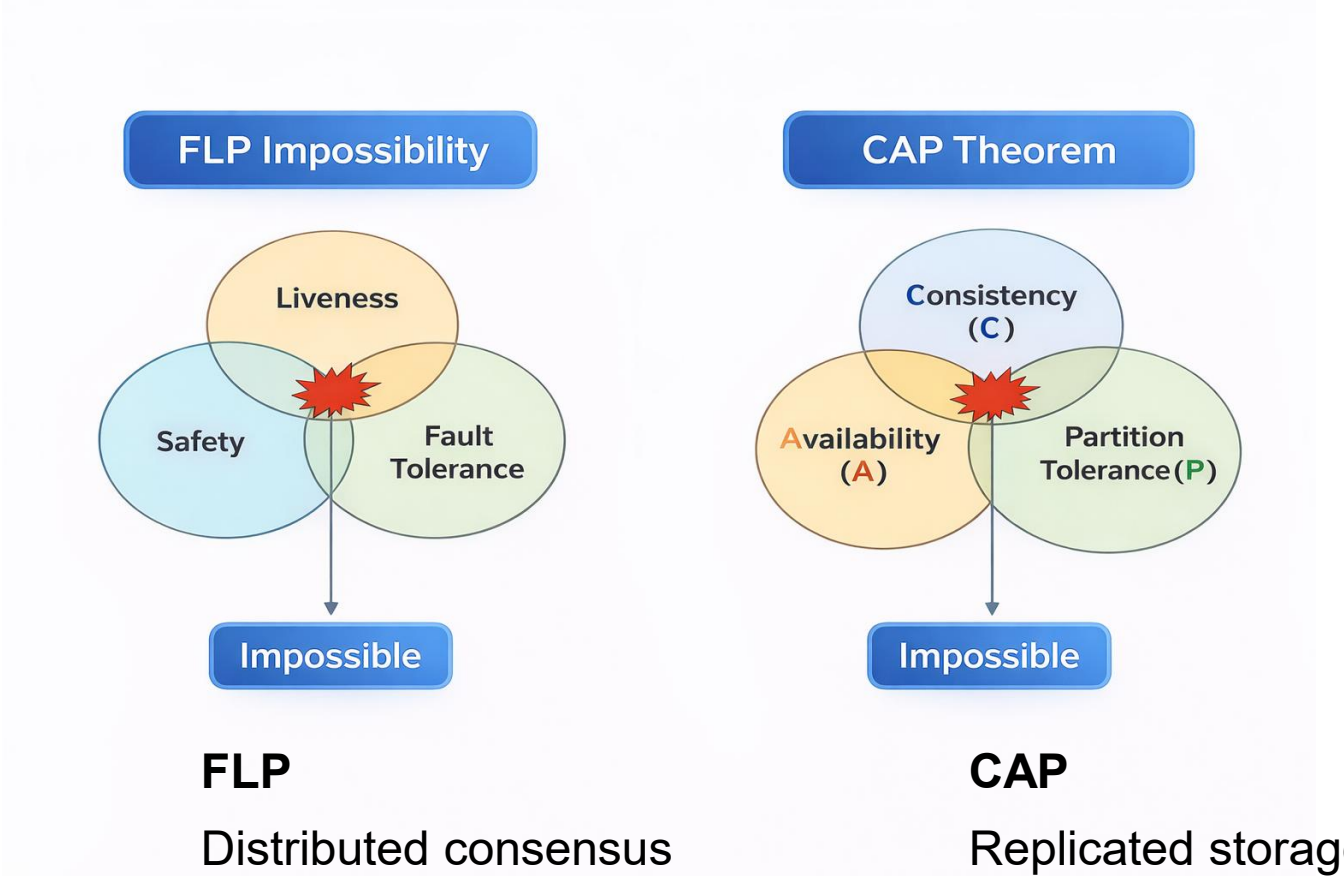
FLP Theorem （直观理解）

- 在完全异步系统中
- 只要允许一个进程崩溃
- 就不存在确定性算法可以保证共识一定终止
- 实际系统通过随机化或部分同步假设解决

FLP Theorem

- **FLP Theorem: In an asynchronous network, it is not possible to achieve safety and liveness when there may be one crash failure.**

FLP / CAP



Properties/Results	FLP	CAP
Problem scope	Distributed consensus	Replicated storage
Failure	Node crash fails	Network fails
Formalization	Rigorous	Gilbert & Lynch approximation
Solutions	Synchronous network model	No solutions exists

阅读

- **阅读列表**
  - A Note on Distributed Computing
  - An Introduction to Input/Output Automata
  - Time, Clocks, and the Ordering of Events in a Distributed System
  - Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial
  - Harvest, Yield, and Scalable Tolerant Systems