

分布式系统

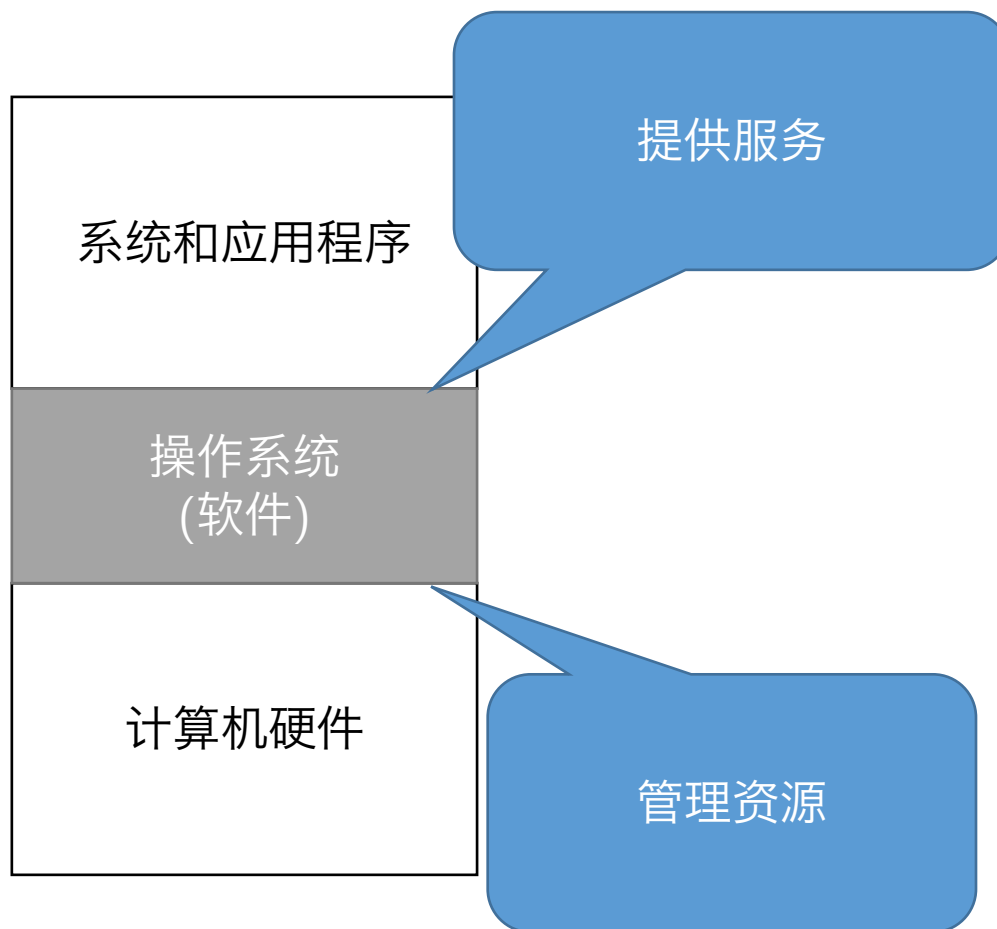
蒋炎岩

南京大学 | 计算机软件研究所 | 系统与软件分析研究组



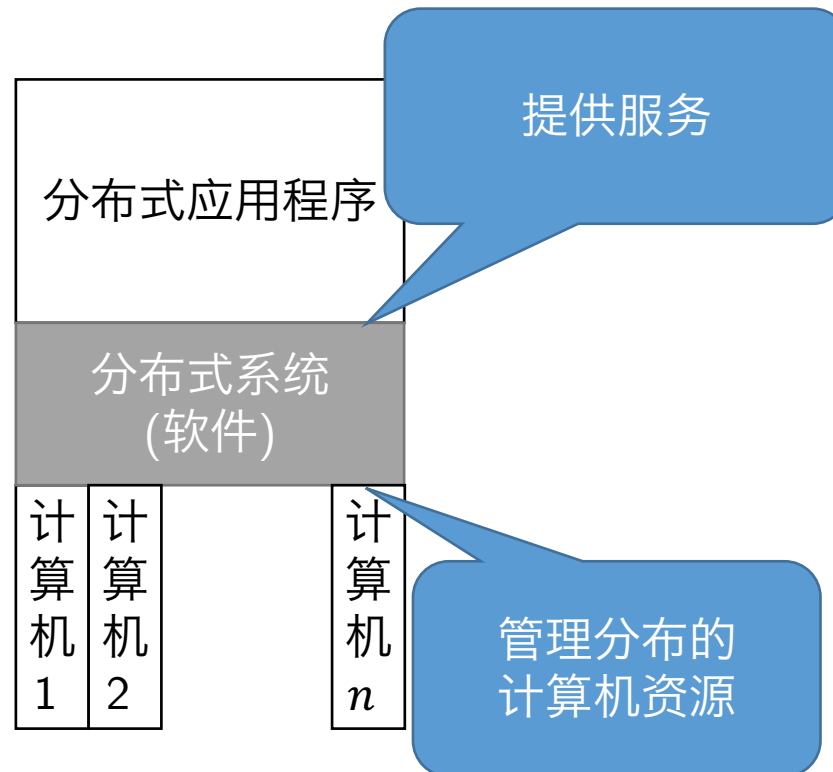


回顾：操作系统



分布式系统：自然的扩展

- 对物理上隔离的多台计算机作出的抽象
 - “A distributed system runs on a collection of computers that do not have shared memory, yet looks like a single computer to its users.”





应用场景

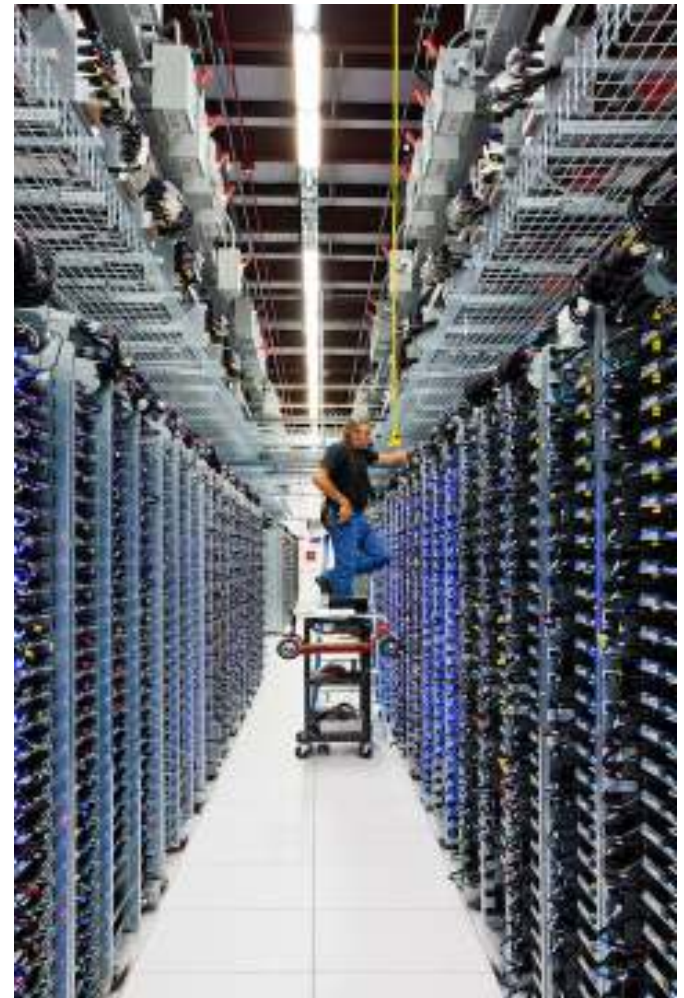
- 假如发生了灾难性的事件，**你的银行余额会清零吗？**



数据中心
(残骸)

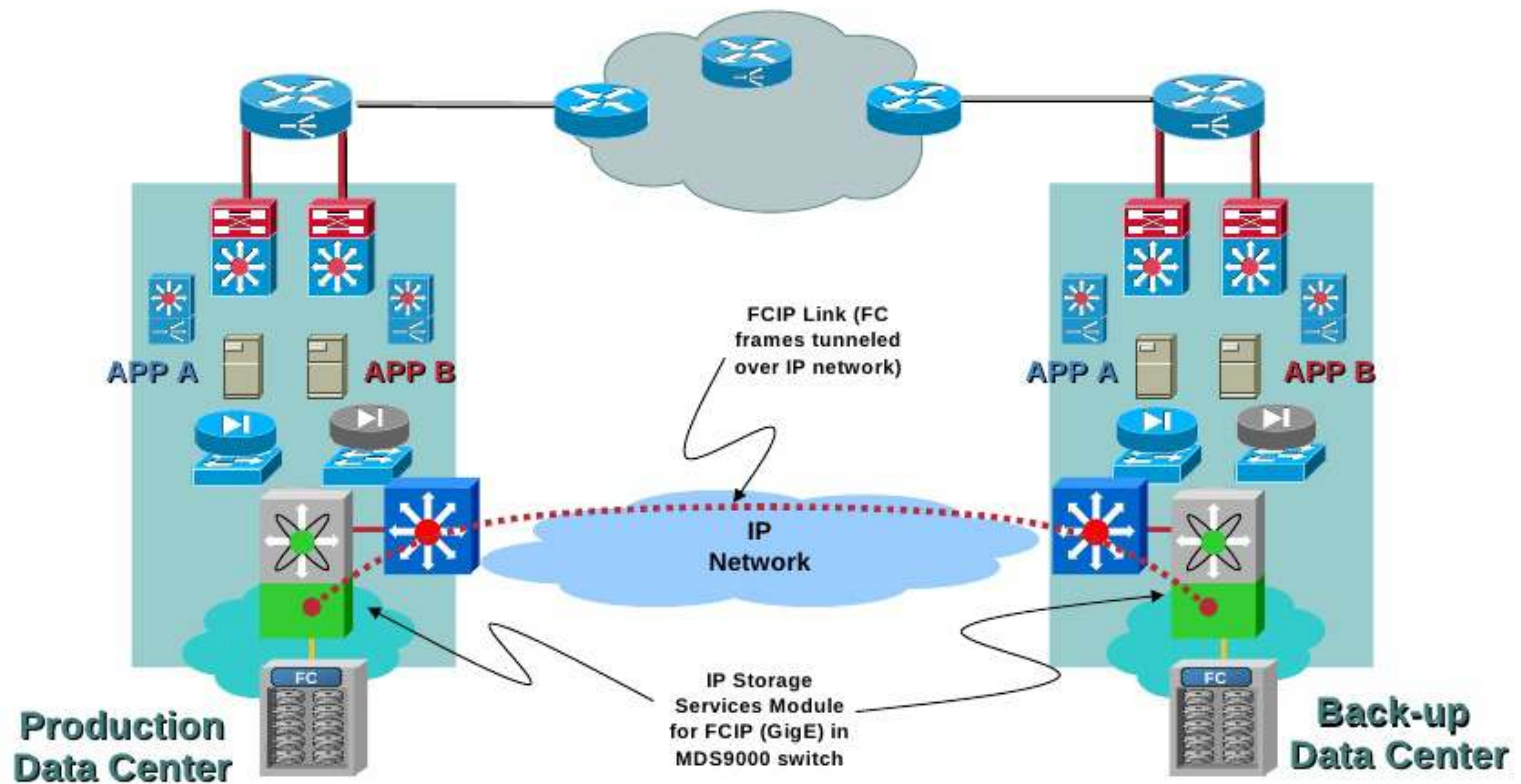
应用场景 (cont'd)

- Google拥有海量的服务器
- 构造超大型的应用
 - PB (1024TB)级的数据
 - 实时查询(邮件、账单、社交.....)
 - 后台计算服务(全互联网索引)
- 用网络/套接字实现?
 - 就像用汇编语言写操作系统



Picture from Google

分布式系统：支撑大数据的技术



并发程序 vs 分布式系统

并发程序：复习

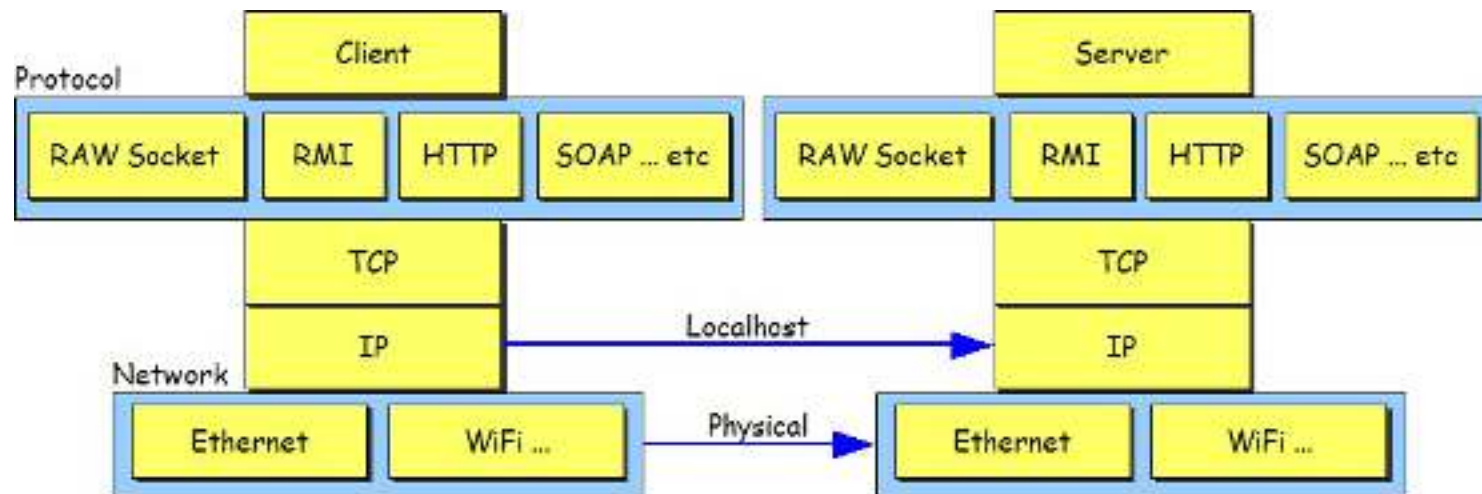
- 在一台计算机上运行多个线程
 - 线程可以并行(SMP, 对称多处理器)/并发执行(Lab1)
 - 线程共享同一地址空间(共享内存)
- 线程API
 - create/join – 创建/等待
 - lock/unlock – 互斥锁(互斥)
 - cond_wait/cond_post/sem_wait/sem_signal – 同步

数据传输：并发程序

- 存在物理共享的内存
 - 缓存一致性协议(例如x86-TSO)
- 访问共享内存只需一条/几条指令
 - `shared_var = 1; MFENCE;` 另一个线程就能看到
 - `xchg`可以实现共享内存里的原子操作
 - 延迟：几个/几十个时钟周期
 - 带宽：13.5GB/s (DDR4-2133 × 1)

数据传输：分布式系统

- 不存在物理共享的内存(通过网络传输数据)
 - (计算机1)打包数据 → 系统调用 → 协议栈 → 网络物理层 → (计算机2)网络物理层 → 协议栈 → 中断处理 → 系统调用返回
 - 延迟：从0.1ms到几百ms到几s (地理分布的计算机)
 - 带宽：1000Mbps (注意是bit)





分布式系统：数据传输带来的限制

- 不能像共享内存并发程序一样任性使用内存
 - 本身就不存在物理共享的内存
 - 高延迟：实现ad-hoc同步的代价过大
 - 无法用lock->flag实现自旋锁
 - 通常使用message passing

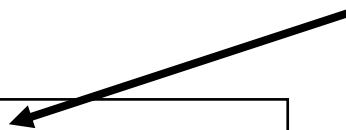
```
void send(machine_t *machine,  
          void *msg, size_t length);  
void receive(machine_t *machine,  
             void *msg, size_t length);
```

正确性：并发程序

- 一对lock-unlock构成了临界区
 - 对于任意一对 L_1, U_1 和 L_2, U_2 , $L_1 < U_2 \vee L_2 < U_1$

```
lock(&mutex);  
...  
unlock(&mutex);
```

```
lock(&mutex);  
...  
unlock(&mutex);
```

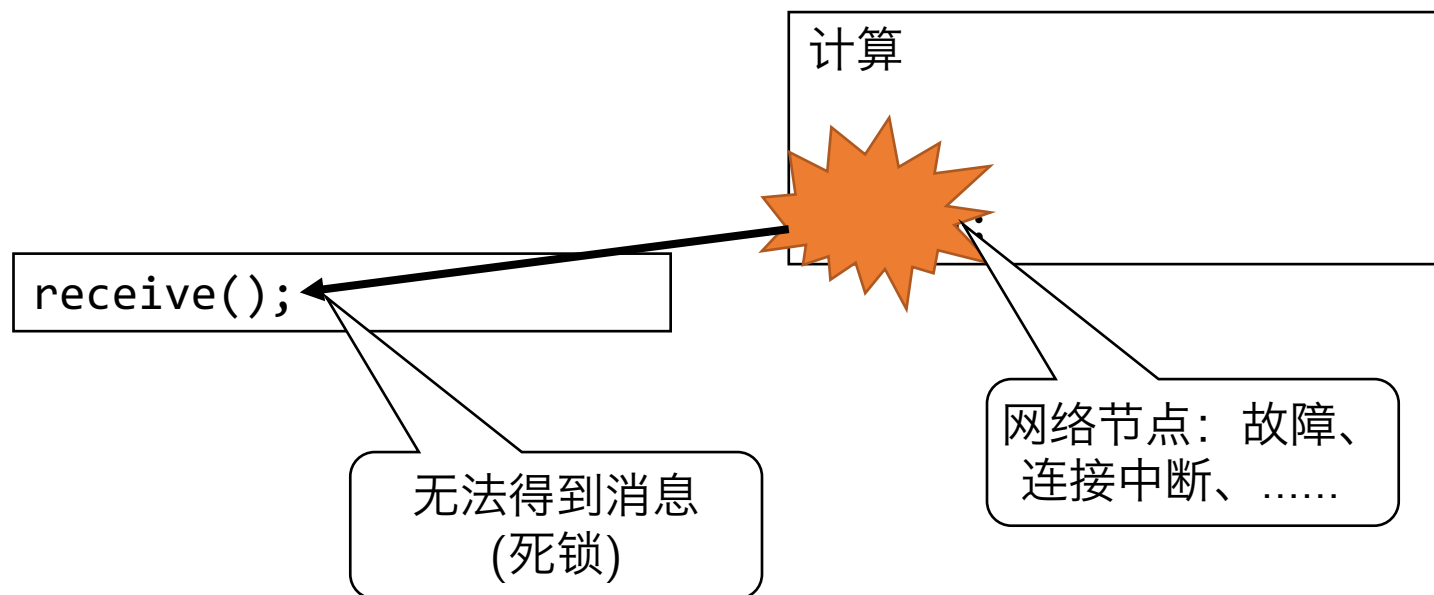


- 这里假设了liveness (每个线程eventually都会被执行)
 - 说人话：线程不会凭空“消失”



正确性：分布式系统

- 网络/其他计算机是不可靠的
 - 网络中的计算机可能凭空消失(相当于延迟无穷大)
 - 即便不消失，也可能带来很大延迟



分布式系统：机制

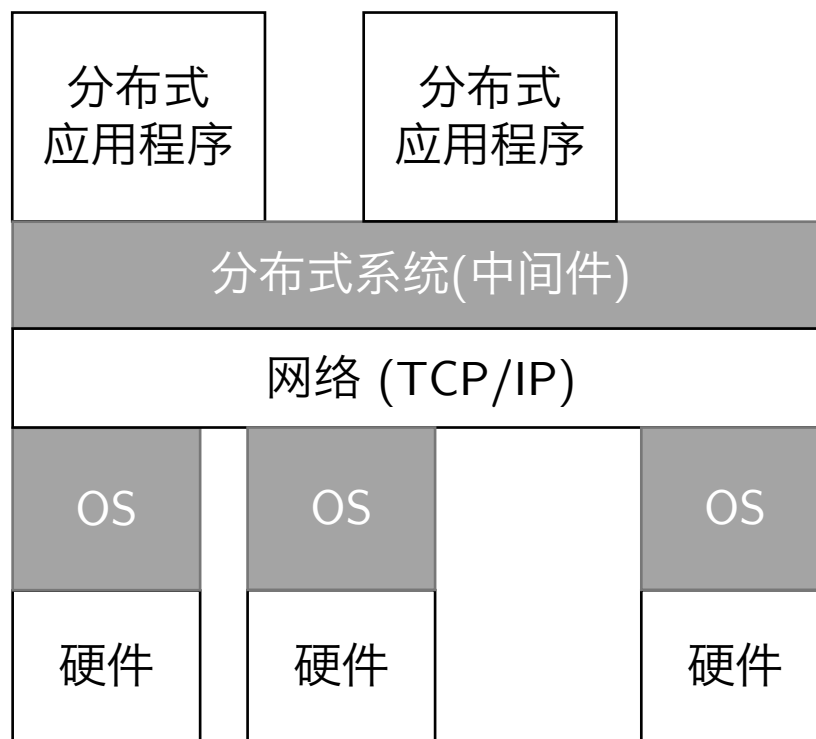
分布式系统设计

- Transparency
 - 应用不需要关注底层分布式硬件细节
- Flexibility
 - API能满足各种应用需求
- Reliability
 - 高可用、数据一致性、容错.....
- Performance
 - 能高效地提供服务
- Scalability
 - 随着系统中计算机的增加处理能力相应提升



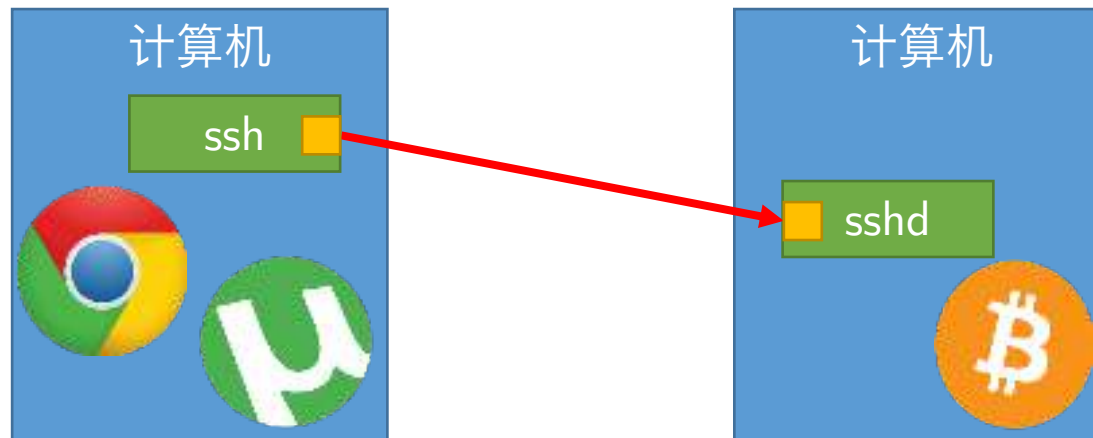
分布式系统：新的抽象层

- 现有操作系统在管理一台物理计算机方面已经做得很好了
 - 借助运行在操作系统上的中间件进行进一步的抽象和管理



复习：网络编程

- 套接字：网络连接的一个端口
 - 可以(本地/远程)连接应用-应用
 - 应用定义应用之间的协议



- 原则上可以使用套接字实现任何分布式系统
 - 就好像可以用汇编实现操作系统

基础设施

- Message Passing (消息传递)
 - send(), receive()
 - 在两个计算机之间传递消息
 - 相当于分布式系统的“汇编语言”
- RPC (Remote Procedure Call, 远程过程调用)
 - Java世界称为RMI (Remote Method Invocation)
 - 把对象打包；发送消息；调用完毕后发送结果消息
 - 看起来就像在本地调用了一个普通函数
 - 用起来自然多了
 - 能够实现跨平台(但也带来对象打包、解包的开销)



节点管理

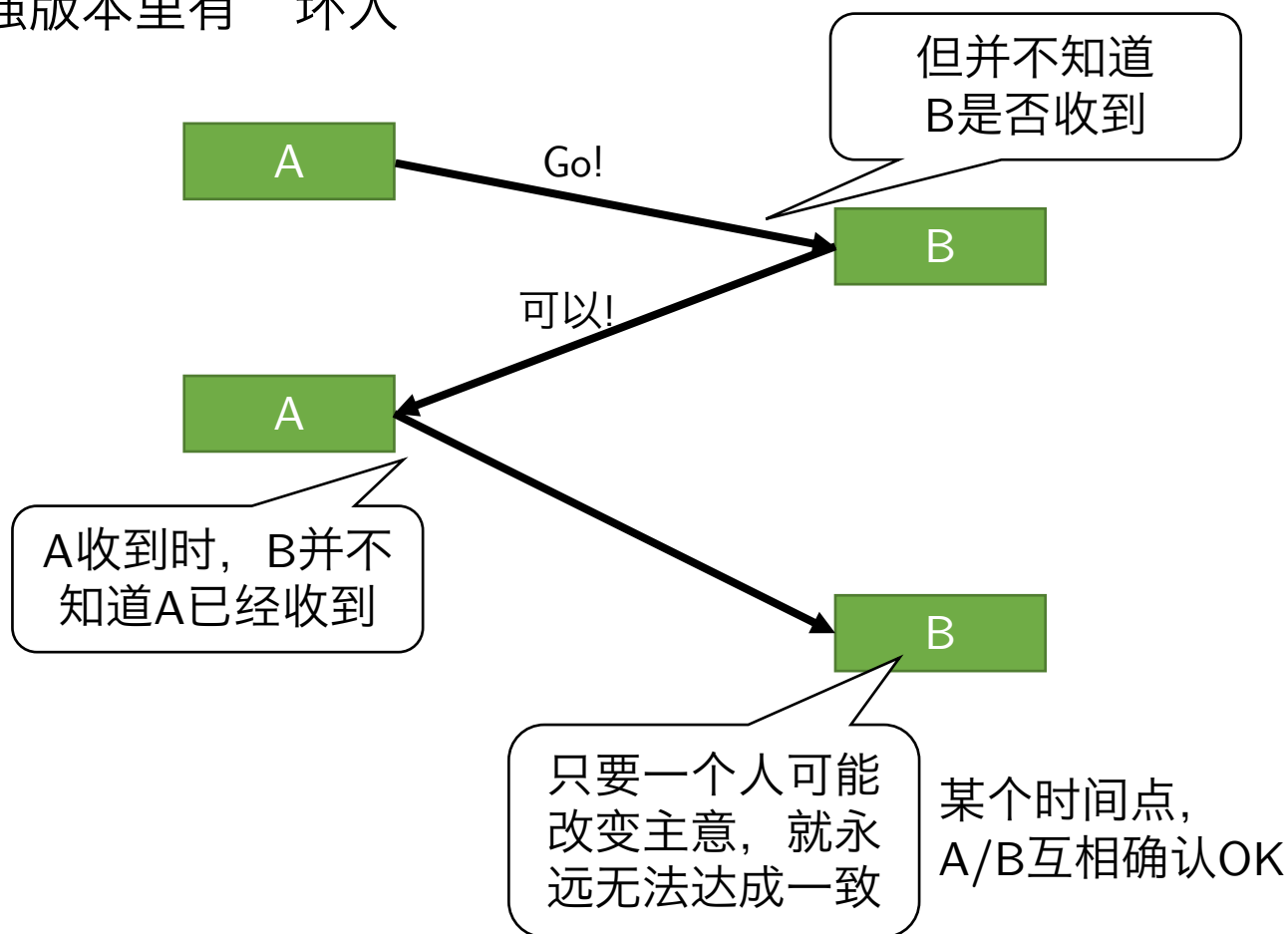
- 分布式系统中可能有很多机器
 - 组成机柜(rack)
 - 数据中心(datacenter)有多个机柜
 - 可能有多个地理分布的数据中心
- 通常总有一些节点充当比较特殊的作用
 - Client/Server; Master/Slave
 - 例如HBase中的RegionServer
 - 核心的节点可能有多个副本，并进行数据同步



分布式系统：困难

分布式系统不存在全局时钟同步

- 拜占庭将军问题(简化版本)
 - 加强版本里有“坏人”



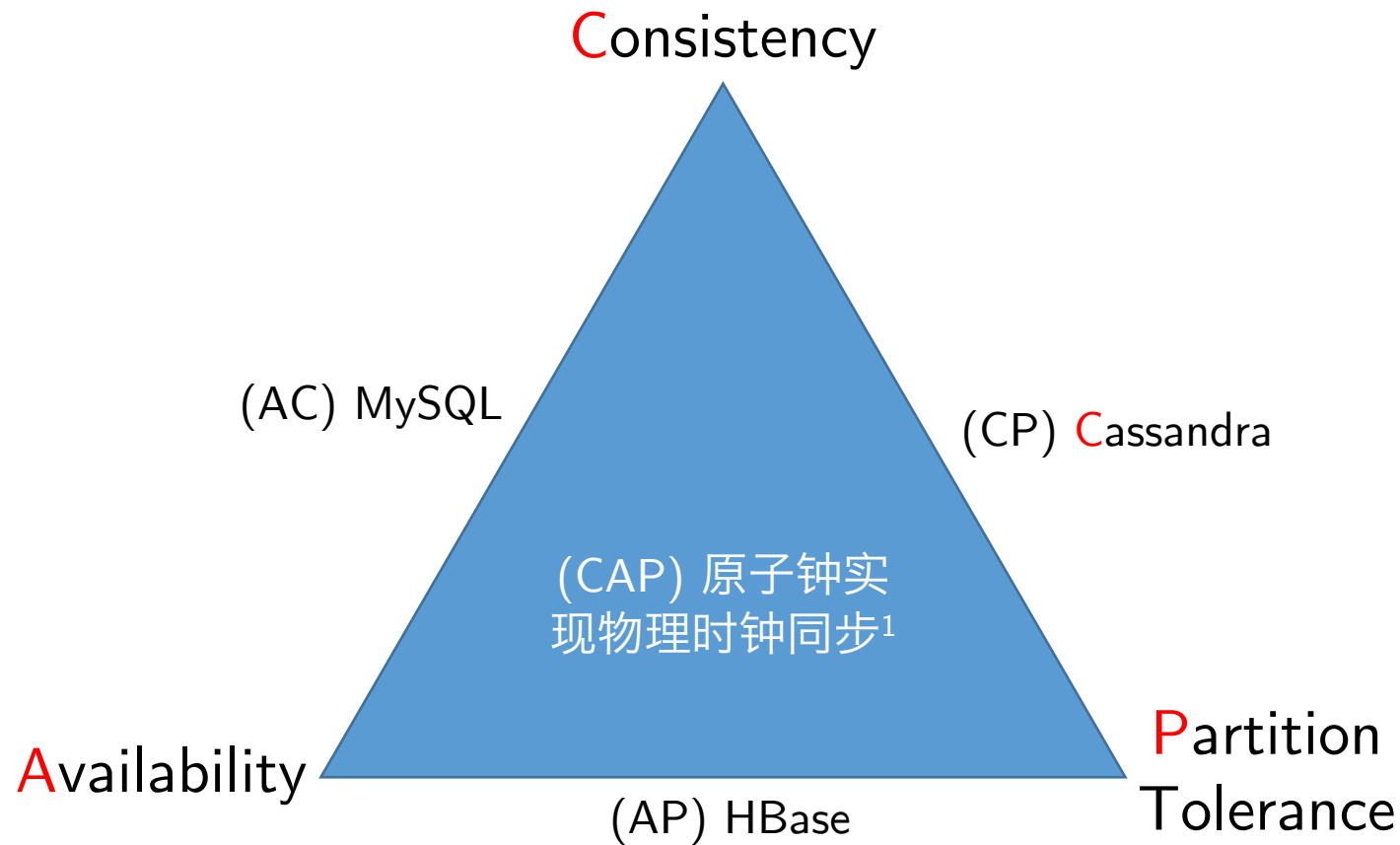
CAP定理：三者不能同时获得

- 假设消息传递分布式系统(无全局时钟)只有一个变量 x
- Consistency
 - 每一个读 $read(x)$ 都读到最近的写成功的数值 $write(x)$, 或出错
- Availability
 - 每个操作都能在有限时间内成功(不出错)
- Partition-tolerance
 - 系统能在任意数量的消息被丢弃(延迟)的前提下运行

CAP定理：解释

- $C + A =$ 顺序一致性
 - 如果没有P (partition), 相当于并发程序
 - 很容易实现保证
- 在有P的情况下
 - 如果要C \rightarrow 因为消息丢弃, 就没法保证成功(A)
 - 如果要A \rightarrow 来不及等消息, 就没法保证一致(C)

分布式系统：作出权衡



¹ James C. Corbett, Jeffrey Dean, Michael Epstein, etc. Spanner: Google's Globally-Distributed Database. In *Proc. of OSDI*, 2012.

分布式系统中的一致性



实现原子性：Two Phase Commit (2PC)

- 实现数据分布式副本上的原子操作

- read/write, 数据库事务.....
- 这是个非常重要的问题!

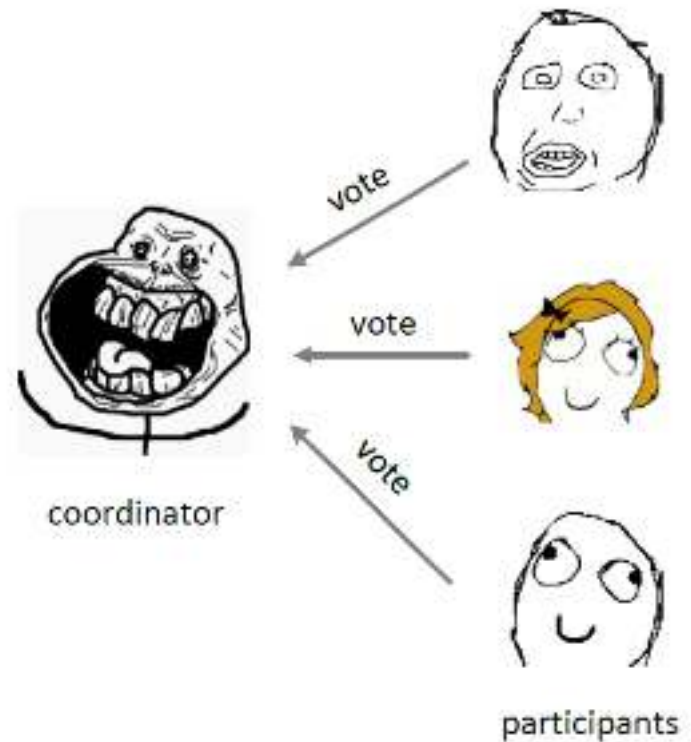
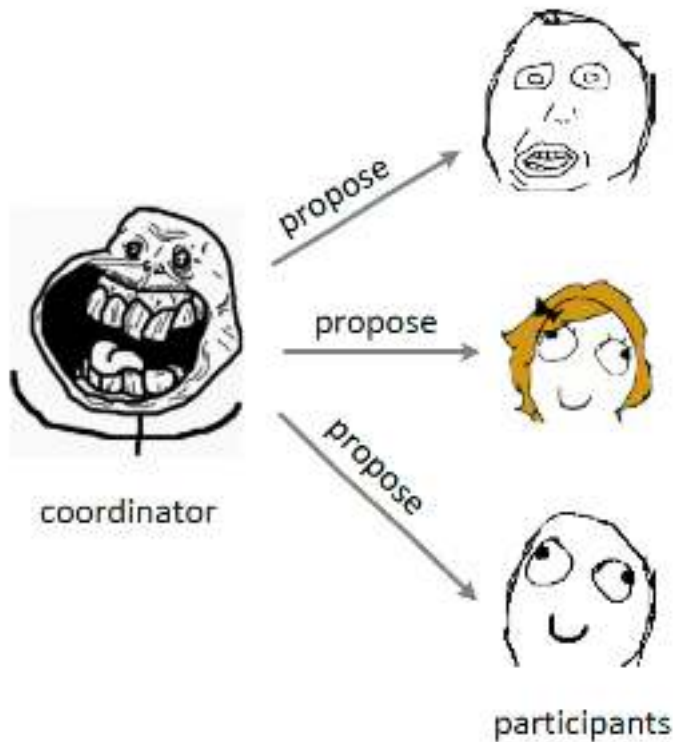
- Master/Slave结构

- 有一个主机管事
- 其他小弟负责同步

分布式系统必须解决：
出现物理破坏(数据中心地震啦)，银行/支付宝余额也还在

2PC: Prepare

- Coordinator向其他人propose, 其他人根据是否能提交进行vote

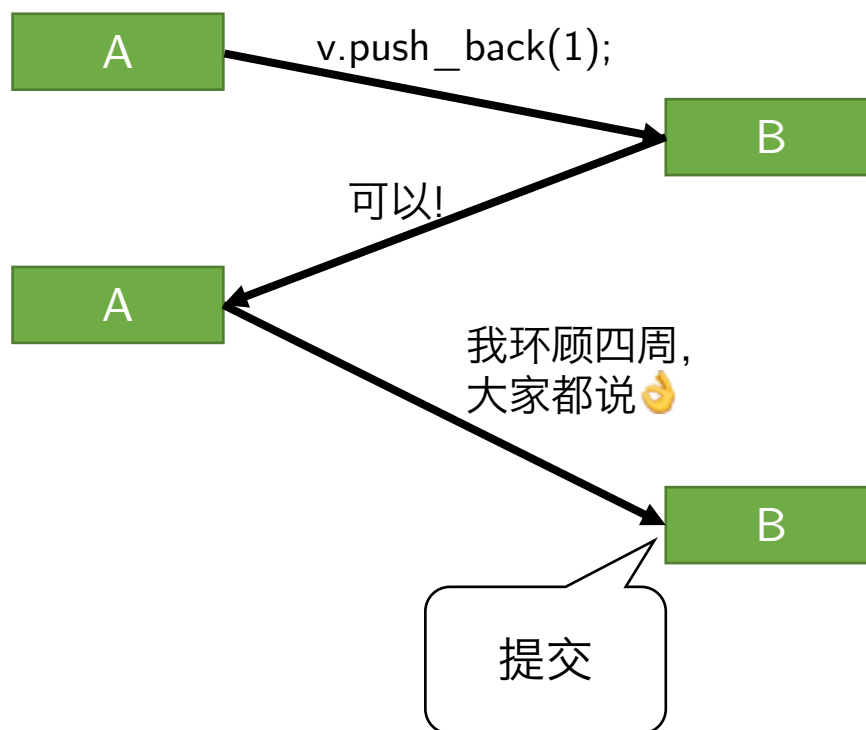


2PC: Commit

- participant block wait结果
 - 如果coordinator收到所有人的同意，则commit
 - 否则abort

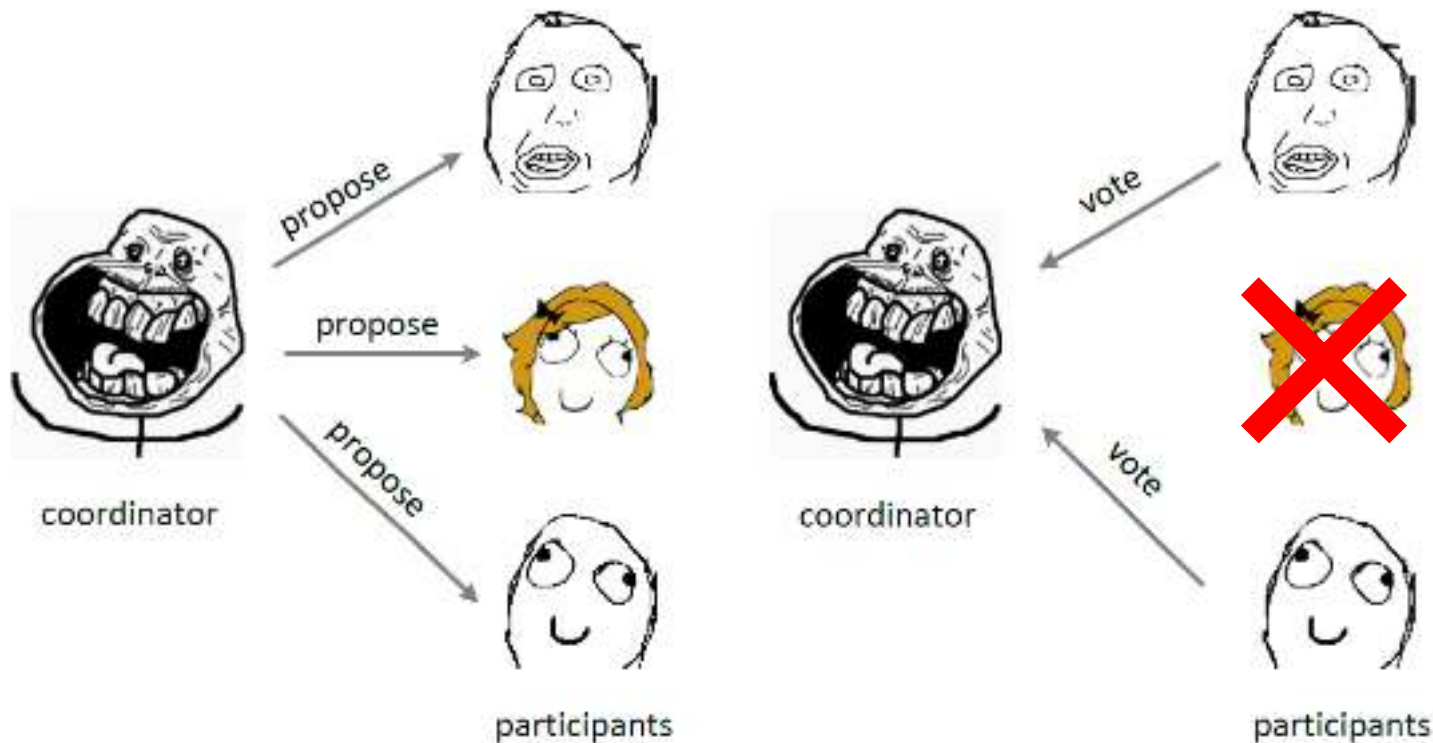


2PC: 时序上看



Participant Fail

- 会导致收到的vote变少，timeout后abort
 - 这是participant唯一发送消息的情况

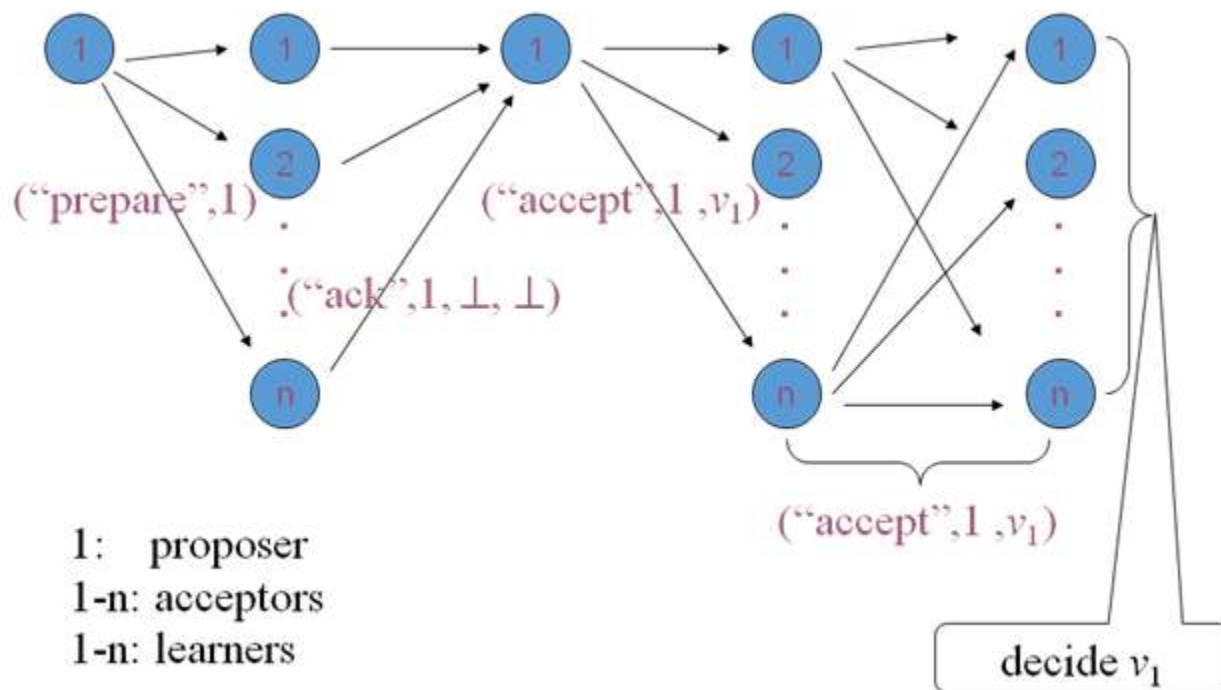


Coordinator Fail

- 这个情况就多了
 - propose消息没发送全就失败——participant timeout, 发送abort (丢失/之后被收到)
 - propose消息发送后失败——vote-commit丢失, 已经发送vote-commit的participant必须等待coordinator重启
 - 满足Availability, Consistency, 但是没有Partition-Tolerance
- 如何证明?
 - 和并发程序一样! 把程序看成状态机
 - $\langle s_1, s_2, s_3 \rangle$ 表示每个机器分别所在的状态

Paxos: 实现Consensus

- 让分布式系统中的所有 n 个节点都收敛到同一个值
 - 没有master/slave的结构，不会因为节点失效而延迟



这个算法似乎很难理解，但理解了以后又觉得特别自然。Leslie Lamport 特意为此写了 “Paxos Made Simple”，但很多人还是觉得很难理解。

Paxos: 应用

- 收敛到同一个值 = 提交
 - Propose: commit-#id
 - 最终所有人收敛——事务提交成功
- 分布式锁服务(Chubby)
 - 最终所有人收敛到锁的owner
- 数据副本