
# 分布式系统
## Distributed Systems

陈鹏飞
数据科学与计算机学院
chchenpf7@mail.sysu.edu.cn
办公室：超算5楼529d
主页：http://sdcs.sysu.edu.cn/node/3747

# 第九讲 — 分布式系统共识

Slides come from drdr.xp@gmail.com, Amir H. Payberah, Jinyang Li

- 分布式系统协作

  多个部分协同工作，对外表现出一个整体；

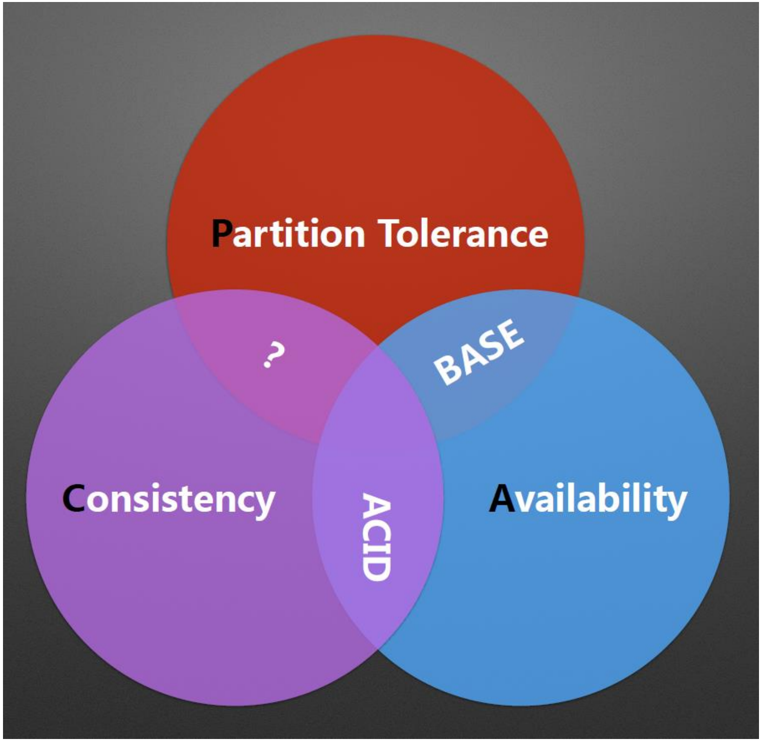
- 一致性

  数据对象在不同副本之间一致；

- 共识

  多个节点对于某件事情比如执行某个指令达成一致；

# 背景

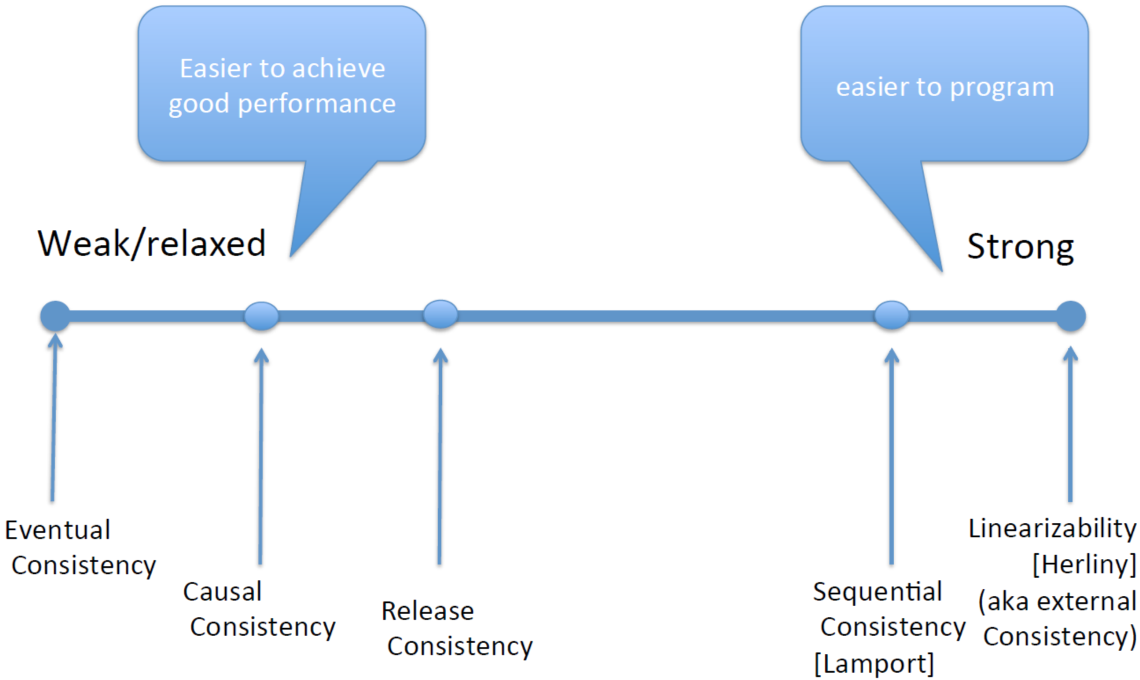


# 一致性

## 一致性级别

1. 强一致性（strong consistency）
2. 单调一致性（monotonic consistency）
3. 会话一致性（session consistency）
4. 最终一致性（eventual consistency）
5. 弱一致性（weak consistency）

# Spectrum of Consistency Models



# 主要的复制算法

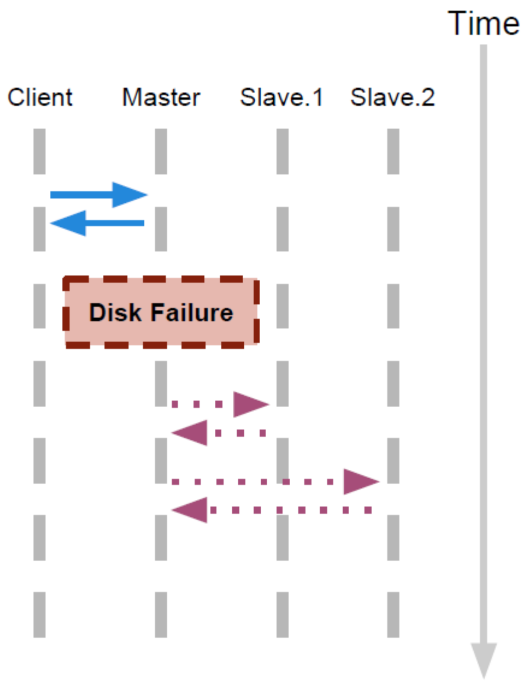
- 主从异步复制
- 主从同步复制
- 主从版同步复制
- 多数派写（读）

# 主从异步复制

如Mysql的binlog复制.

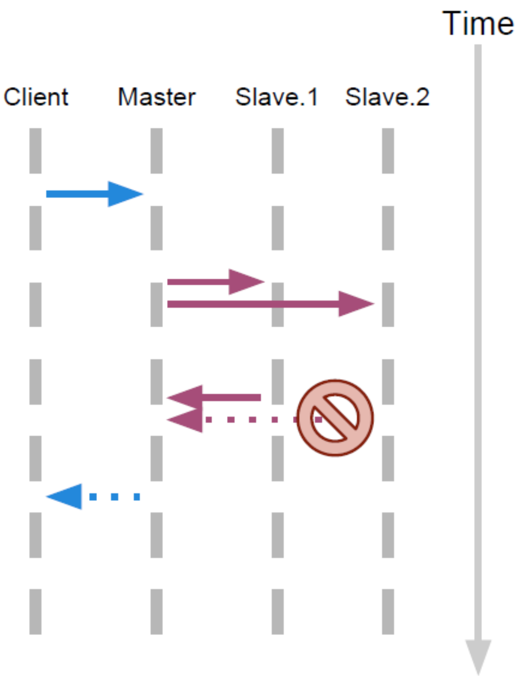
1. 主接到写请求.
2. 主写入本磁盘.
3. 主应答'OK'.
4. 主复制数据到从库.

如果磁盘在复制前损坏:
→ 数据丢失.



# 主从异步复制

1. 主接到写请求.
2. 主复制日志到从库.
3. 从库这时可能阻塞...
4. 客户端一直在等应答'OK', 直到所有从库返回.




一个失联节点造成整个系统不可用.

: 没有数据丢失.

: 可用性降低.

# 主从半同步复制

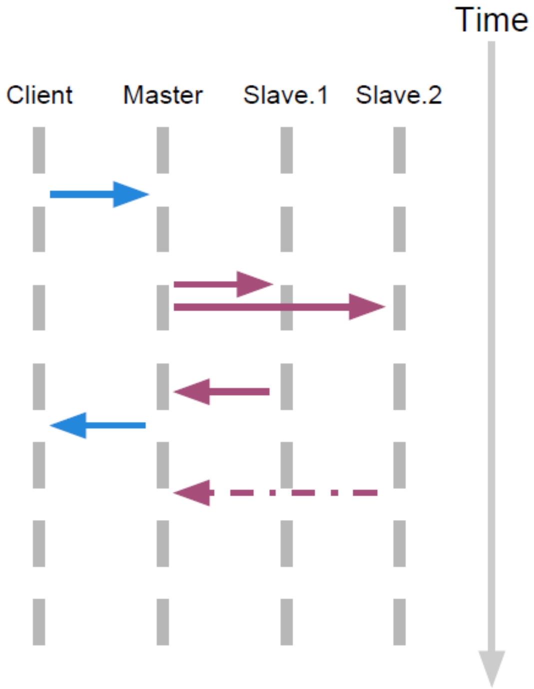
1. 主接到写请求.
2. 主复制日志到从库.
3. 从库这时可能阻塞...
4. 如果1<=x<=n个从库返回'OK',
   则返回客户端'OK'.

: 高可靠性.

: 高可用性.

: 可能任何从库都不完整

→ 我们需要 多数派写(读)



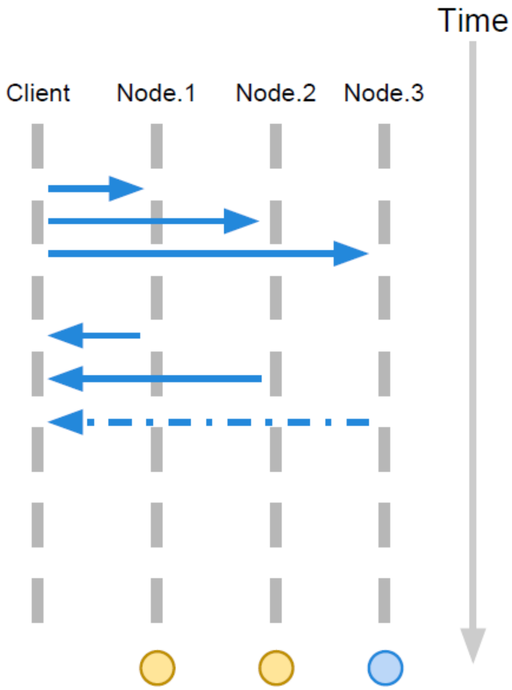
# 多数派写

## Dynamo / Cassandra

客户端写入W >=N/2+1个节点.
不需要主.

多数派读:
W + R > N; R >= N/2+1

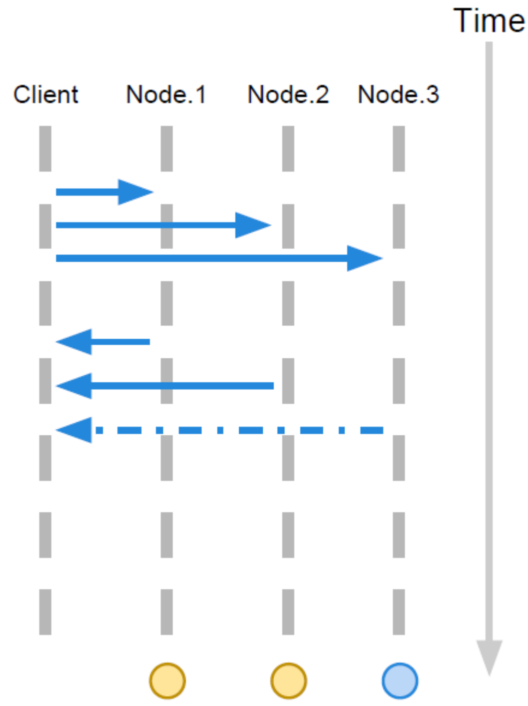
容忍最多(N-1)/2个节点损坏.



# 多数派写：后写入优胜

最后1次写入覆盖先前写入.

所有写入操作需要有1个全局顺序:时间戳



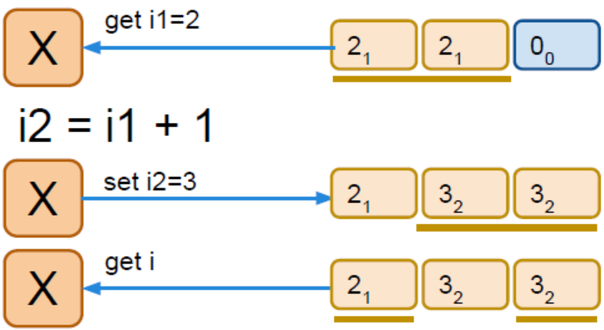
# 假设

命令实现:

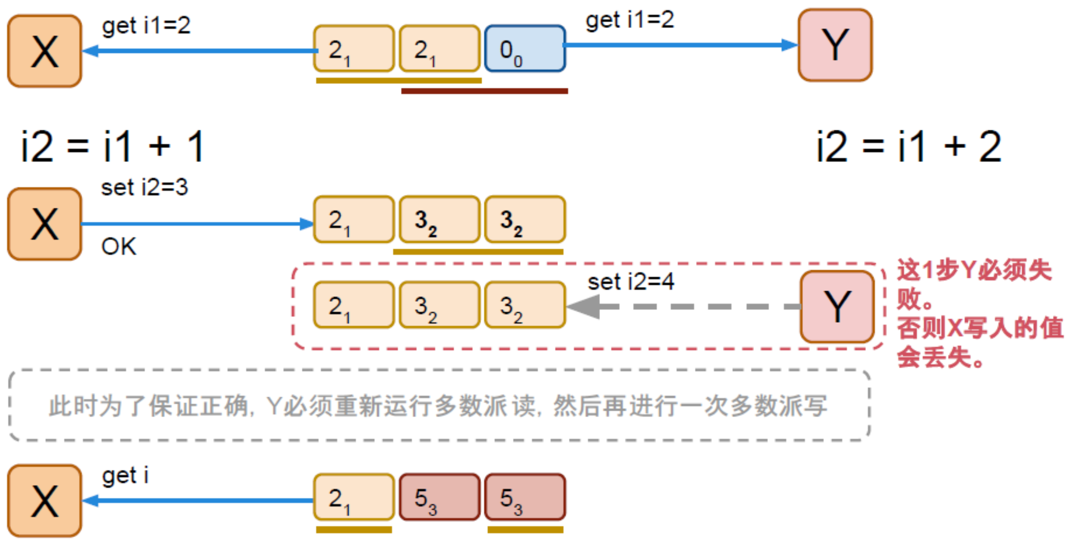
"set" → 直接对应多数派写.

"inc" → (最简单的事务型操作):

1. 通过多数派读, 读取最新的 “i”: i1
2. Let i2 = i1 + n
3. set i2



# 假想的并发访问




我们期待最终X可以读到i3=5，
这需要Y能知道X已经写入了i2。如何实现这个机制？

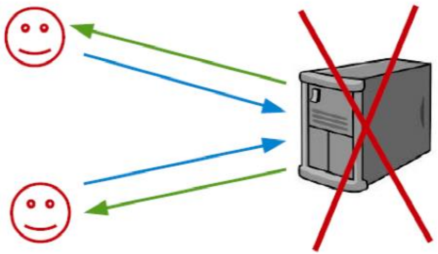
# 共识 Paxos

# What is the Problem?

# Two Generals' Problem

- Two generals need to be agree on time to attack to win.
- They communicate through messengers, who may be killed on their way.
- Agreement is the problem.

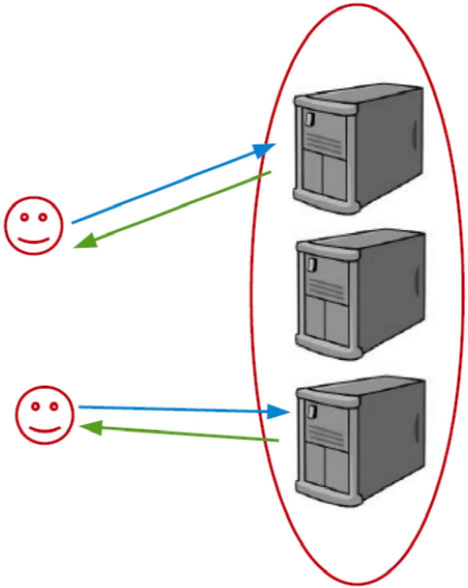
# Replicated State Machine Problem (1/2)



- The solution: replicate the server.

# Replicated State Machine Problem (2/2)

- Make the server deterministic (state machine).
- Replicate the server.
- Ensure correct replicas step through the same sequence of state transitions (How?)
- Agreement is the problem.



# The Agreement Problem

# The Agreement Problem

- Some nodes propose values (or actions) by sending them to the others.
- All nodes must decide whether to accept or reject those values.

- But, ...

- Concurrent processes and uncertainty of timing, order of events and inputs.
- Failure and recovery of machines/processors, of communication channels.

# Agreement Requirements

- Safety
  - Validity: only a value that has been proposed may be chosen.
  - Agreement: no two correct nodes choose different values.
  - Integrity: a node chooses at most once.
- Liveness
  - Termination: every correct node eventually choose a value.

# Agreement in Distributed Systems: Possible Solutions

- Two-Phase Commit (2PC)
- Paxos

# Two-Phase Commit (2PC)

# The Two-Phase Commit (2PC) Problem

- The problem first was encountered in database systems.
- Suppose a database system is updating some complicated data structures that include parts residing on more than one machine.
- System model:
  - Concurrent processes and uncertainty of timing, order of events and inputs (asynchronous systems).
  - Failure and recovery of machines/processors, of communication channels.

# Intuitive Example (1/3)

- You want to organize outing with 3 friends at 6pm Tuesday.
  - Go out only if all friends can make it.

# Intuitive Example (2/3)

- What do you do?
  - Call each of them and ask if can do 6pm on Tuesday (voting phase)
  - If all can do Tuesday, call each friend back to ACK (commit)
  - If one cannot do Tuesday, call other three to cancel (abort)

# Intuitive Example (3/3)

- Critical details
  - While you were calling everyone to ask, people who have promised they can do 6pm Tuesday must reserve that slot.
  - You need to remember the decision and tell anyone whom you have not been able to reach during commit/abort phase.
- That is exactly how 2PC works.

# The 2PC Players

- Coordinator (Transaction Manager)
  - Begins transaction.
  - Responsible for commit/abort.
- Participants (Resource Managers)
  - The servers with the data used in the distributed transaction.

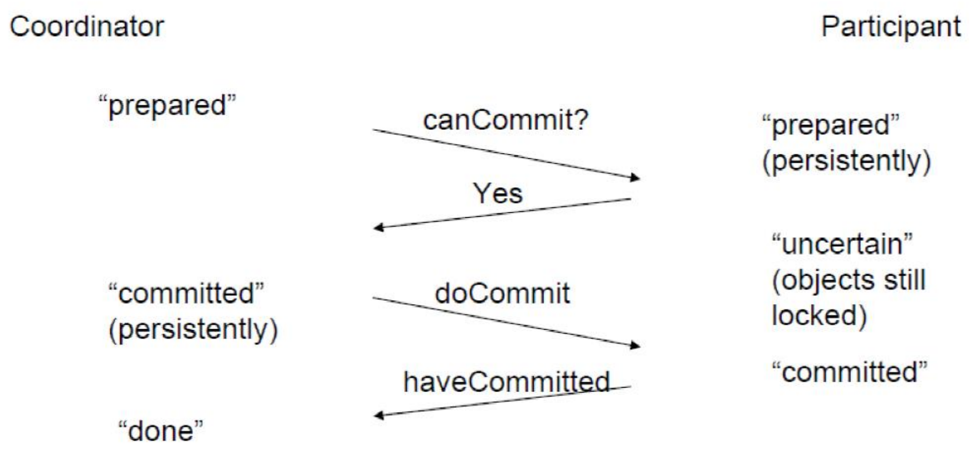
# The 2PC Algorithm - Prepare Phase

- Coordinator asks each participant `canCommit`.
- Participants must prepare to commit using permanent storage before answering `yes`.
  - Lock the objects.
  - Participants are not allowed to cause an abort after it replies `yes` to `canCommit`.
- Outcome of the transaction is uncertain until `doCommit` or `doAbort`.
  - Other participants might still cause an abort.

# The 2PC Algorithm - Commit Phase

- The coordinator collects all votes.
  - If unanimous yes, causes commit.
  - If any participant voted no, causes abort.
- The fate of the transaction is decided atomically at the coordinator, once all participants vote.
  - Coordinator records fate using permanent storage.
  - Then broadcasts `doCommit` or `doAbort` to participants.

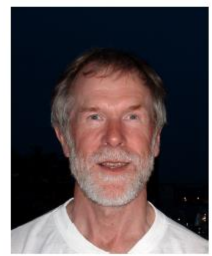
# 2PC Sequence of Events



# Impossibility of Distributed Consensus with One Faulty Process

# FLP

- Fischer-Lynch-Paterson (FLP)
  - M.J. Fischer, N.A. Lynch, and M.S. Paterson, Impossibility of distributed consensus with one faulty process, Journal of the ACM, 1985.



[PDF] Impossibility of Distributed Consensus with One Faulty Process
https://groups.csail.mit.edu/tds/papers/Lynch/jacm85.pdf ▾ 翻译此页
作者：MJ FISCHER - 1985 - 被引用次数：4703 - 相关文章
Impossibility of Distributed Consensus with One Faulty. Process. MICHAEL J. FISCHER. Yale University, New Haven, Connecticut. NANCY A. LYNCH.

# FLP Impossibility Result

- It is impossible for a set of processors in an asynchronous system to agree on a binary value, even if only a single process is subject to an unannounced failure.
- The core of the problem is asynchrony.

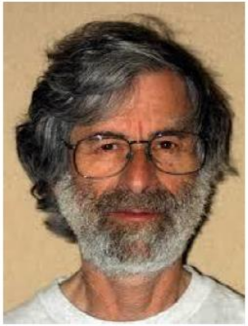
# FLP Impossibility Result

- What FLP says: you cannot guarantee both safety and progress when there is even a single fault at an inopportune moment.
- What FLP does not say: in practice, how close can you get to the ideal (always safe and live)?
- So, Paxos ...

# Paxos

- The only known completely-safe and largely-live agreement protocol.
- L. Lamport, The part-time parliament, ACM Transactions on Computer Systems, 1998.

希腊岛屿Paxon 上的执法者（legislators，后面称为牧师priest）在议会大厅（chamber）中表决通过法律，并通过服务员传递纸条的方式交流信息，每个执法者会将通过的法律记录在自己的账目（ledger）上。问题在于执法者和服务员都不可靠，他们随时会因为各种事情离开议会大厅，并随时可能有新的执法者进入议会大厅进行法律表决，使用何种方式能够使得这个表决过程正常进行，且通过的法律不发生矛盾。



# The Paxos Players

- Proposers
  - Suggests values for consideration by acceptors.
- Acceptors
  - Considers the values proposed by proposers.
  - Renders an accept/reject decision.
- Learners
  - Learns the chosen value.
- A node can act as more than one roles (usually 3).

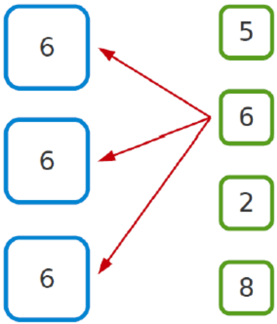
# Single Proposal, Single Acceptor

- Use just one acceptor
  - Collects proposers' proposals.
  - Decides the value and tells everyone else.


- Sounds familiar?
  - two-phase commit (2PC)
  - acceptor fails = protocol blocks

# Single Proposal, Multiple Acceptors

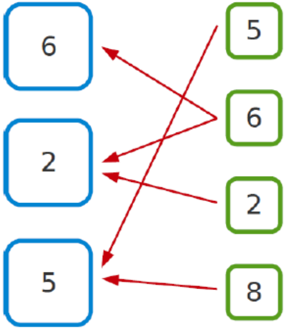
- One acceptor is not fault-tolerant enough.
- Let's have multiple acceptors.




- From there, must reach a decision. How?
- Decision = value accepted by the majority.
- P1: an acceptor must accept first proposal it receives.

## Multiple Proposals, Multiple Acceptors

- If there are multiple proposals, no proposal may get the majority.
  - 3 proposals may each get 1/3 of the acceptors.




- Solution: acceptors can accept multiple proposals, distinguished by a unique proposal number.

# Paxos 定义

**Proposer**: 发起paxos的进程.

**Acceptor**: 存储节点，接受、处理和存储消息.

**Quorum**(Acceptor的多数派) : n/2+1个Acceptors.

**Round**: 1轮包含2个阶段: Phase-1 & Phase-2

每1轮的编号 (**rnd**):

单调递增; 后写胜出; 全局唯一(用于区分Proposer);

# Paxos 定义

Acceptor看到的最大**rnd** (**last_rnd**):

Acceptor记住这个值来识别哪个proposer可以写。

Value (**v**): Acceptor接受的值.

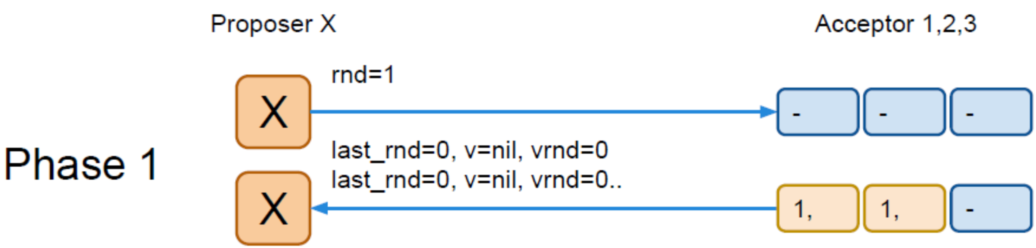
Value round number (**vrnd**):

Acceptor接受的**v**的时候的**rnd**

值‘被确定的’定义:

有多数(多于半数)个Acceptor接受了这个值.

# 典型 Paxos：阶段1

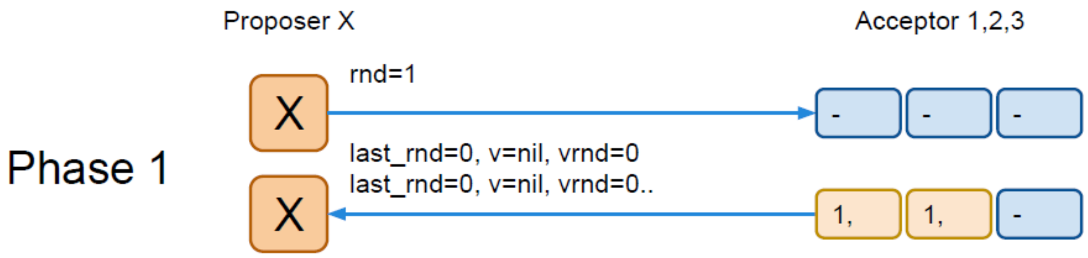



当Acceptor收到phase-1的请求时：

- 如果请求中**rnd**比Acceptor的**last_rnd**小，则拒绝请求
- 将请求中的**rnd**保存到本地的**last_rnd**.

  从此这个Acceptor只接受带有这个**last_rnd**的**phase-2**请求。
- 返回应答，带上自己之前的**last_rnd**和之前已接受的**v**.

# 典型 Paxos：阶段1

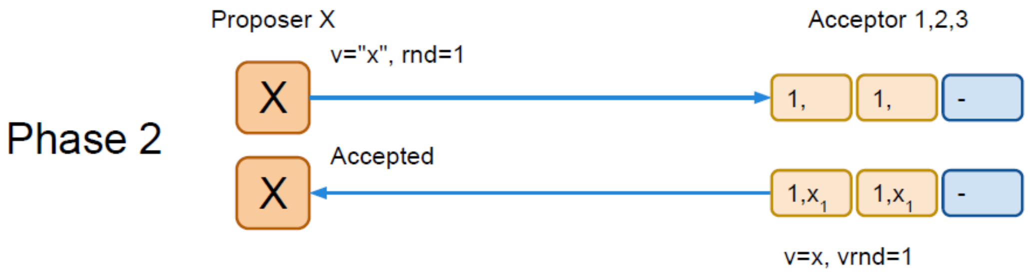



当Proposer收到Acceptor发回的应答：

- 如果应答中的**last_rnd**大于发出的**rnd**: 退出.
- 从所有应答中选择**vrnd**最大的**v**:

  不能改变(可能)已经确定的值
- 如果所有应答的**v**都是空，可以选择自己要写入**v**.
- 如果应答不够多数派，退出

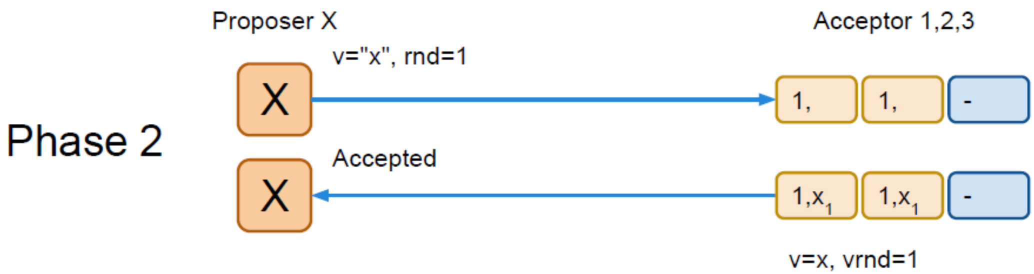
# 典型 Paxos：阶段2




Proposer:

发送**phase-2**，带上**rnd**和上一步决定的**v**

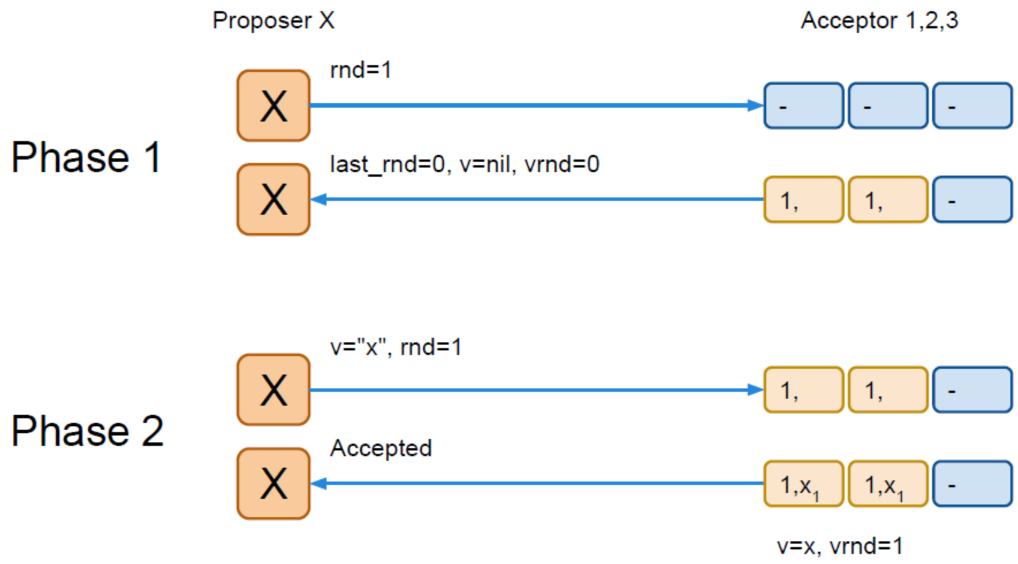
# 典型 Paxos：阶段2



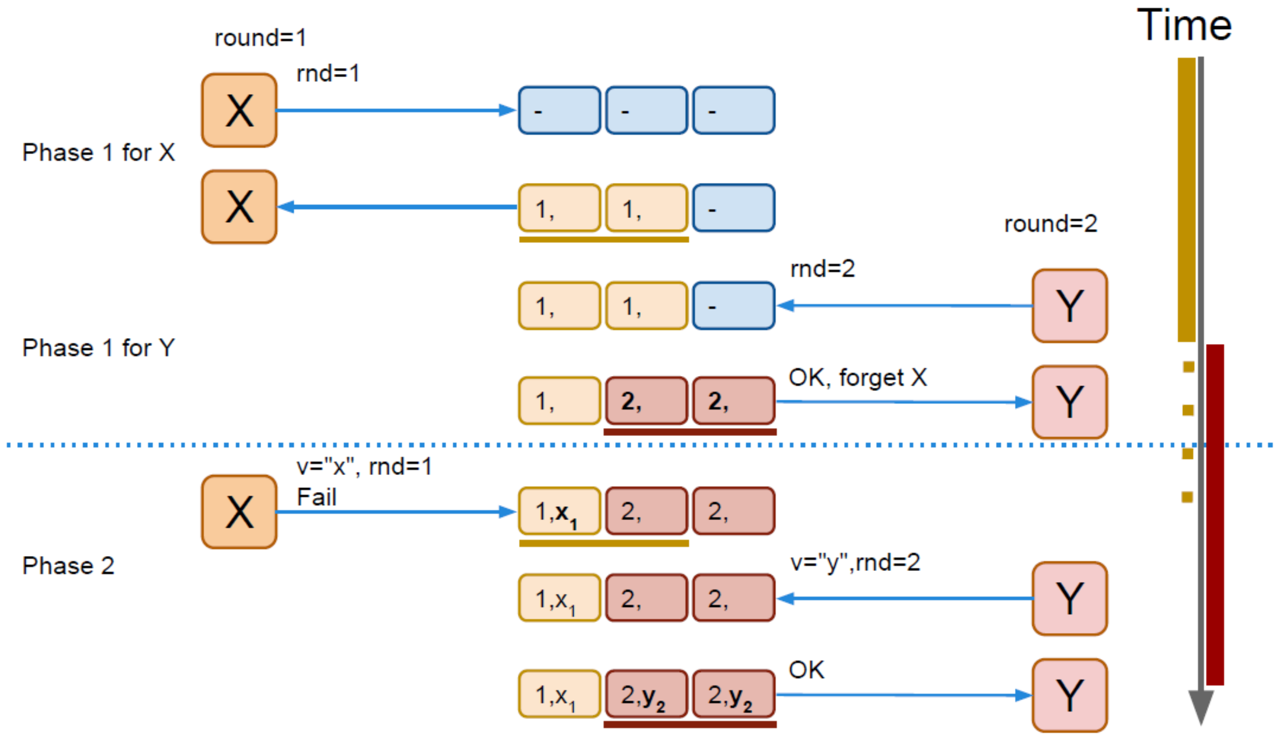

Acceptor:

- 拒绝**rnd**不等于Acceptor的**last_rnd**的请求
- 将**phase-2**请求中的**v**写入本地，记此**v**为‘已接受的值’
- **last_rnd==rnd** 保证没有其他Proposer在此过程中写入过其他值

# 例子： 无冲突

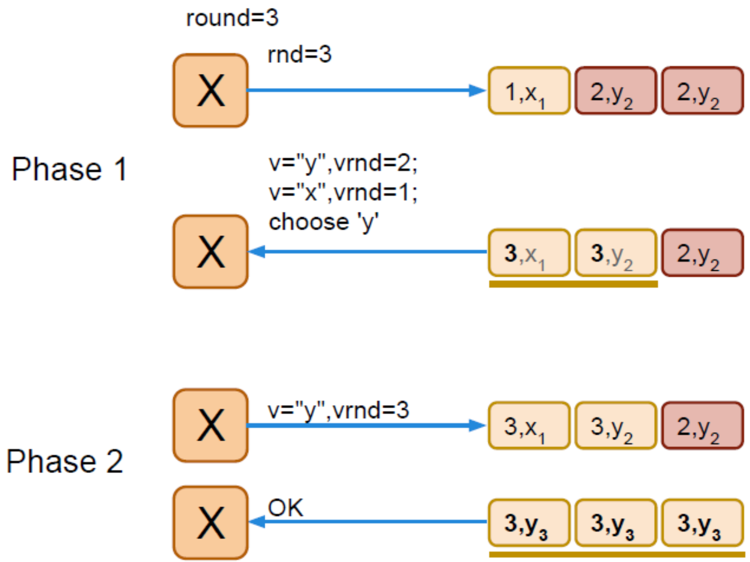

Proposer X
Acceptor 1,2,3
Phase 1
rnd=1
X
-
-
-
last_rnd=0, v=nil, vrnd=0
X
1,
1,
-
Phase 2
v="x", rnd=1
X
1,
1,
-
Accepted
X
1,x1
1,x1
-
v=x, vrnd=1

# 例子2： 解决并发写冲突



Time
round=1
rnd=1
X
-
-
-
Phase 1 for X
X
1,
1,
-
round=2
rnd=2
1,
1,
-
Y
Phase 1 for Y
1,
2,
2,
OK, forget X
Y
v="x", rnd=1
Fail
X
1,x1
2,
2,
Phase 2
v="y",rnd=2
1,x1
2,
2,
Y
OK
1,x1
2,y2
2,y2
Y

# 例子3：X 不会修改确定的 V




X只能选择v="y", 因为它可能是一个被确定的值。

# 其他

Learner角色:

- Acceptor发送**phase-3** 到所有learner角色，让learner知道一个值被确定了.
- 多数场合Proposer就是1个Learner.

Livelock:

多个Proposer并发对1个值运行paxos的时候，可能会互相覆盖对方的**rnd**，然后提升自己的rnd再次尝试，然后再次产生冲突，一直无法完成

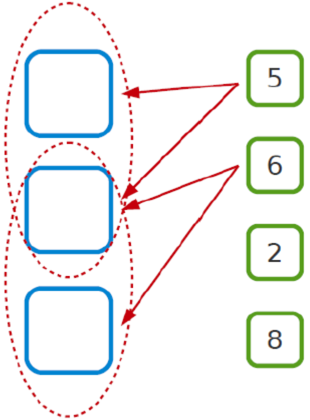
# Paxos - Safety (1/3)

- If a value $v$ is chosen at proposal number $n$, any value that is sent out in phase 2 of any later proposal numbers must be also $v$.

# Paxos - Safety (2/3)

- Decision = Majority (any two majorities share at least one element)

- Therefore after the first round in which there is a decision, any subsequent round involves at least one acceptor that has accepted $v$.



# Paxos - Safety (3/3)

- Now suppose our claim is not true, and let $m$ is the first proposal number that is later than $n$ and in 2nd phase, the value sent out is $w \neq v$.

- This is not possible, because if the proposer $P$ was able to start 2nd phase for $w$, it means it got a majority to accept round for $m$ (for $m > n$). So, either:
  - $v$ would not have been the value decided, or
  - $v$ would have been proposed by $P$

- Therefore, once a majority accepts $v$, that never changes.

# Liveness

- If two or more proposers race to propose new values, they might step on each other toes all the time.
  - $P_1$: `prepare`$(n_1)$
  - $P_2$: `prepare`$(n_2)$
  - $P_1$: `accept`$(n_1, v_1)$
  - $P_2$: `accept`$(n_2, v_2)$
  - $P_1$: `prepare`$(n_3)$
  - $P_2$: `prepare`$(n_4)$
  - ...
  - $n_1 < n_2 < n_3 < n_4 < \cdots$
- With randomness, this occurs exceedingly rarely.

# Paxos：种类

## Classic Paxos

1个实例(确定1个值)写入需要2轮RPC.

## Multi Paxos

约为1轮RPC，确定1个值(第1次RPC做了合并).

## Fast Paxos

没冲突:1轮RPC确定一个值.

有冲突:2轮RPC确定一个值.

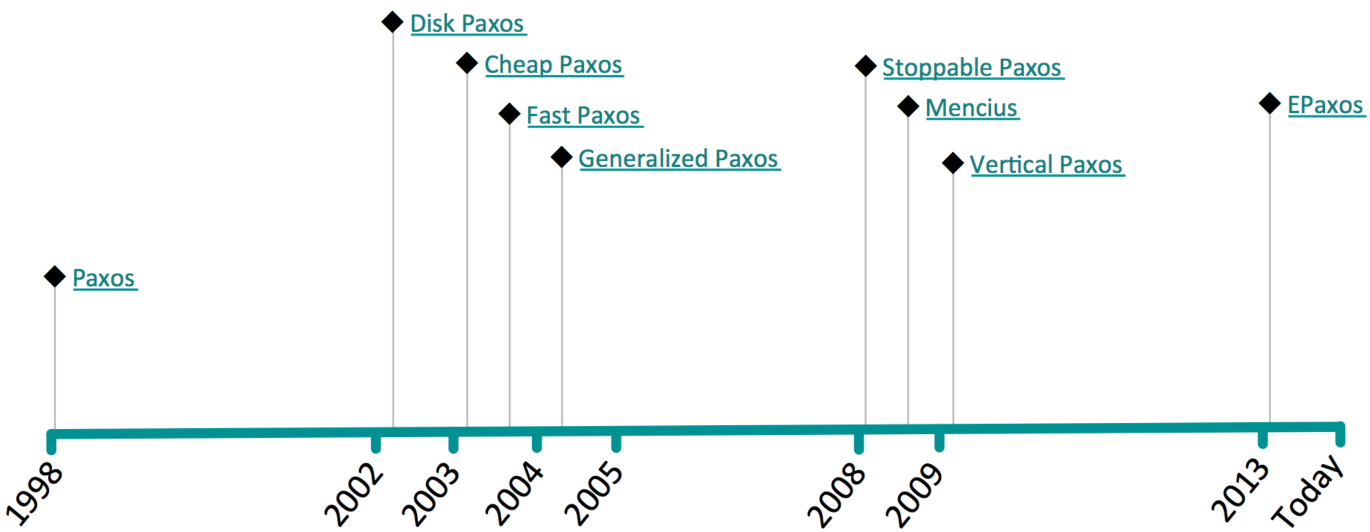
# Multi Paxos

将多个paxos实例的phase-1合并到1个RPC;
使得这些paxos只需要运行phase-2即可。

应用:

chubby zookeeper megastore spanner

# 其他变种




**http://paxos.systems/variants.html**

谢谢！