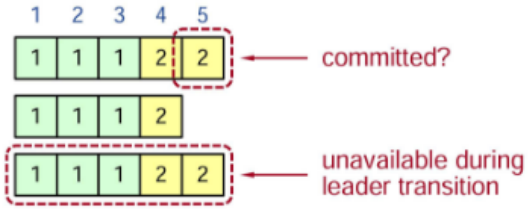
Raft: 领导选举 理解 配图（1）

➢ 选举安全性

已经被大多数机器确认提交的数据不能被覆盖

Picking the Best Leader

- Can't tell which entries are committed!



- During elections, choose candidate with log most likely to contain all committed entries
  - Candidates include log info in RequestVote RPCs (index & term of last log entry)
  - Voting server V denies vote if its log is "more complete":
    $(lastTerm_V > lastTerm_C)$ ||
    $(lastTerm_V == lastTerm_C)$ && $(lastIndex_V > lastIndex_C)$
  - Leader will have "most complete" log among electing majority

March 3, 2013 Raft Consensus Algorithm Slide 17

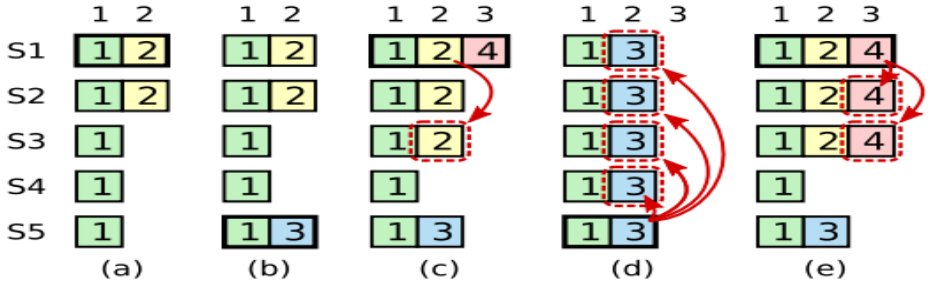


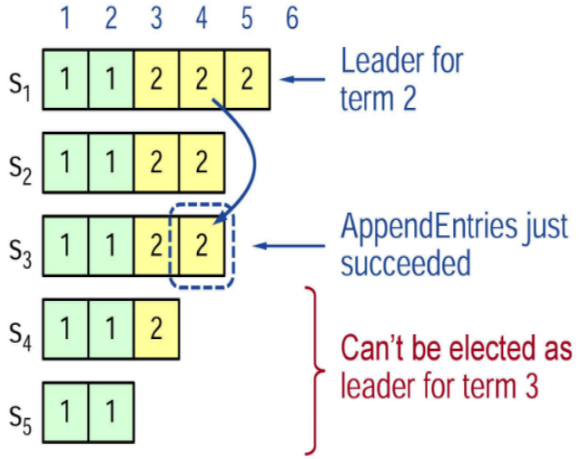
Figure 8: A time sequence showing why a leader cannot determine commitment using log entries from older terms. In (a) S1 is leader and partially replicates the log entry at index 2. In (b) S1 crashes; S5 is elected leader for term 3 with votes from S3, S4, and itself, and accepts a different entry at log index 2. In (c) S5 crashes; S1 restarts, is elected leader, and continues replication. At this point, the log entry from term 2 has been replicated on a majority of the servers, but it is not committed. If S1 crashes as in (d), S5 could be elected leader (with votes from S2, S3, and S4) and overwrite the entry with its own entry from term 3. However, if S1 replicates an entry from its current term on a majority of the servers before crashing, as in (e), then this entry is committed (S5 cannot win an election). At this point all preceding entries in the log are committed as well.

➢ 左图只有三台机器，右图 c中提交数据，在d中被覆盖，选举领导算法有缺陷。

Raft: 领导选举 理解 配图（2）

Committing Entry from Current Term

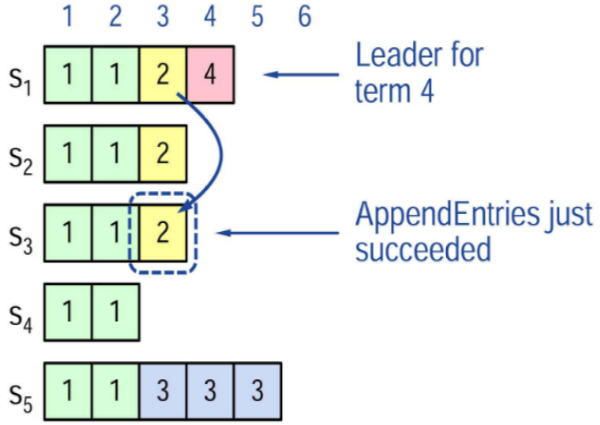
- Case #1/2: Leader decides entry in current term is committed



- Safe: leader for term 3 must contain entry 4

Committing Entry from Earlier Term

- Case #2/2: Leader is trying to finish committing entry from an earlier term



- Entry 3 not safely committed:
  - s_5 can be elected as leader for term 5
  - If elected, it will overwrite entry 3 on s_1, s_2, and s_3!

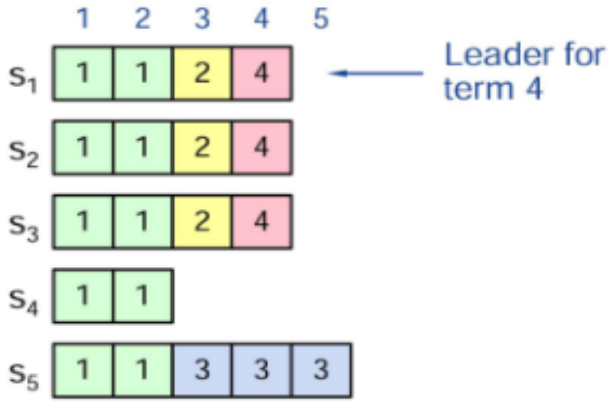
➢ 左图 S4只能获取一个投票 ok。右图 S5 因为任期号3大2 ， 获取3投票 有问题。

Raft: 领导选举 理解 配图（3）

➢ **Election restriction 。**

New Commitment Rules

- **For a leader to decide an entry is committed:**
  - Must be stored on a majority of servers
  - At least one new entry from leader's term must also be stored on majority of servers
- **Once entry 4 committed:**
  - s_5 cannot be elected leader for term 5
  - Entries 3 and 4 both safe



Combination of election rules and commitment rules makes Raft safe

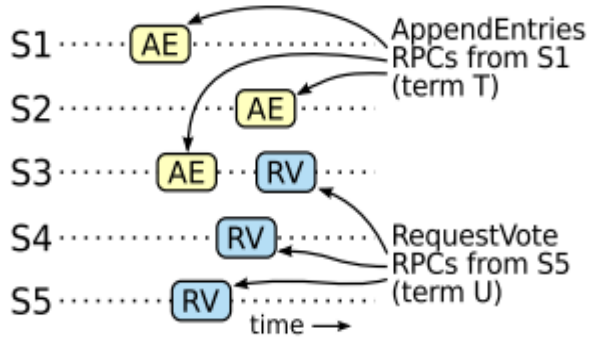
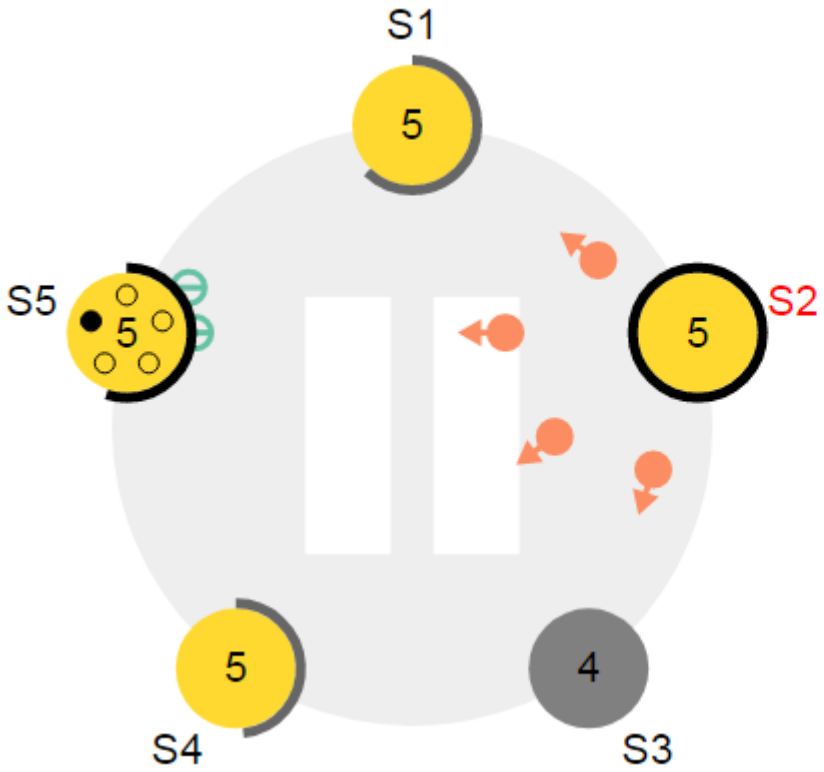


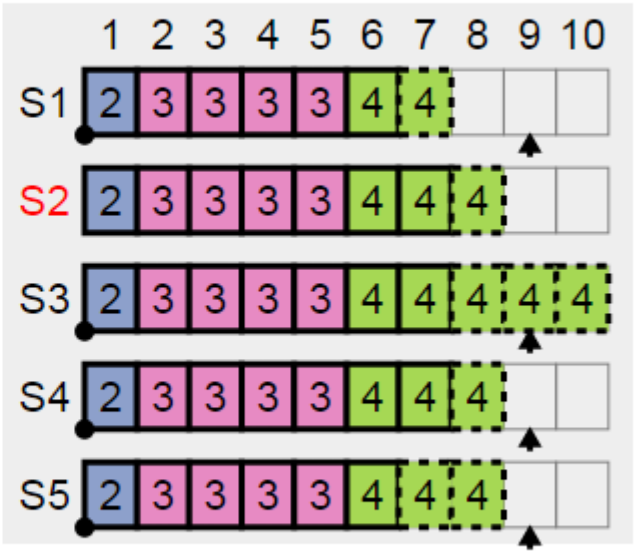
Figure 9: If S1 (leader for term T) commits a new log entry from its term, and S5 is elected leader for a later term U, then there must be at least one server (S3) that accepted the log entry and also voted for S5.

➢ 不同任期，必须能看见至少有一条来自于它本任期内的记录也存于大多数服务器。

Raft: 日志复制 理解 配图（1）

- **Log replication 。**





说明：

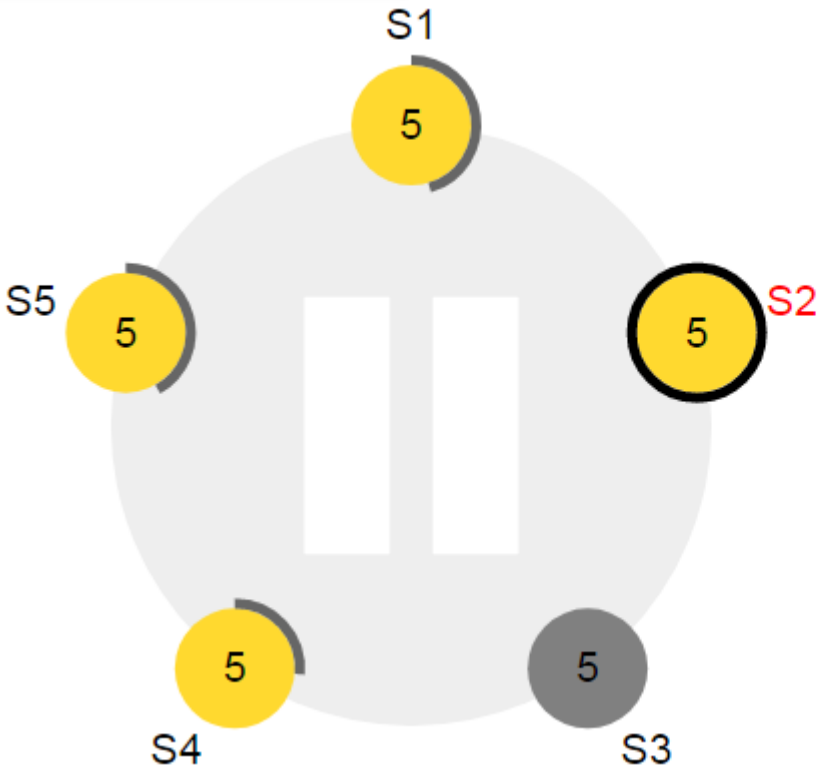
虚框表示：数据未提交

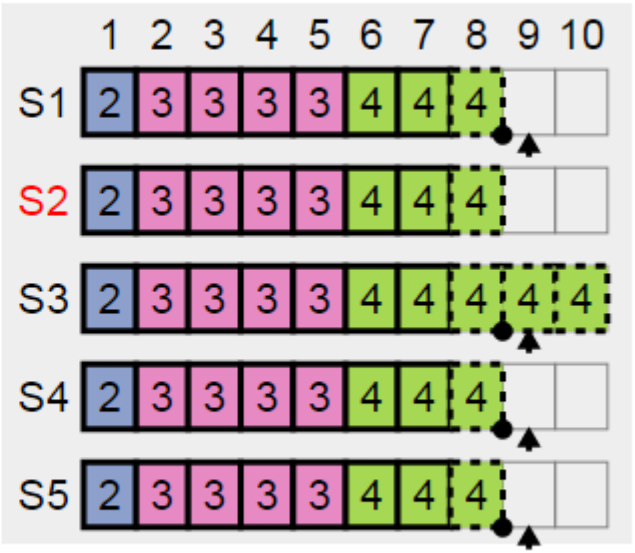
实框表表示：数据已经提交

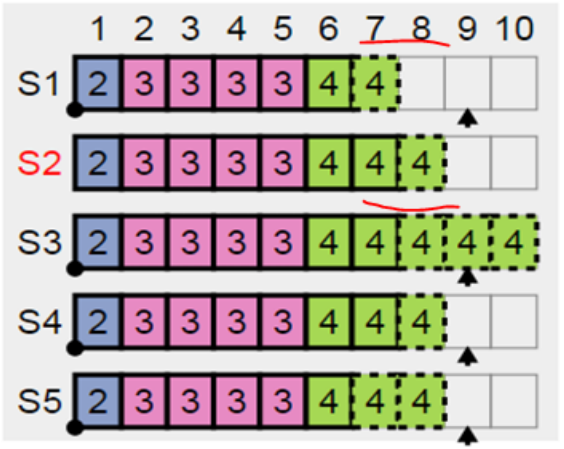
- **S3故障，S2选举新领导，raft 依然按照正常日志复制流，省去故障恢复时间。**

Raft:日志复制 理解 配图（2）

➢ **Log replication 。**





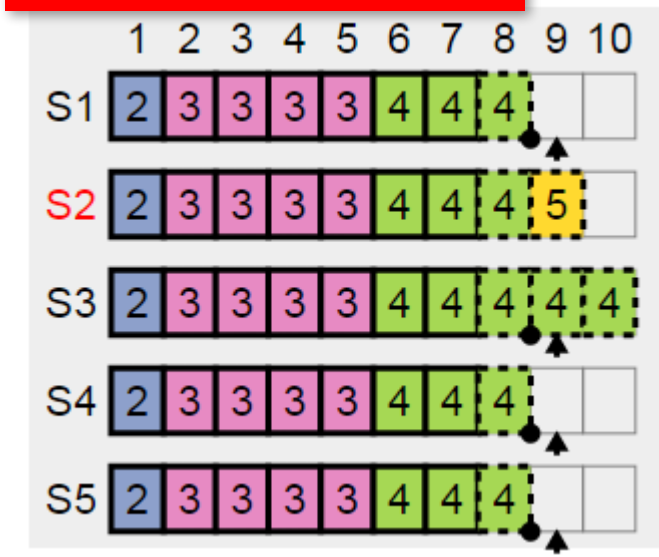


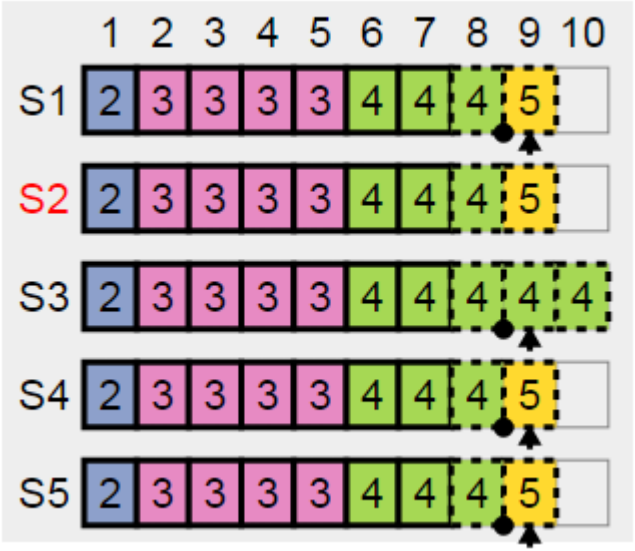
- Raft never commits log entries from previous terms by counting replicas
- If the follower does not find an entry in its log with the same index and term, then it refuses the new entries 。

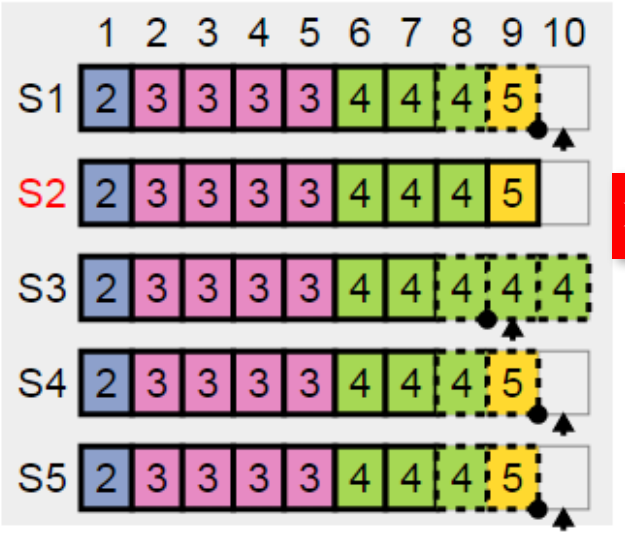
➢ **客户端无感知，不考虑客户造成不一致，** normal operation ， **AppendEntries consistency check**

Raft:日志复制 理解 配图（3）

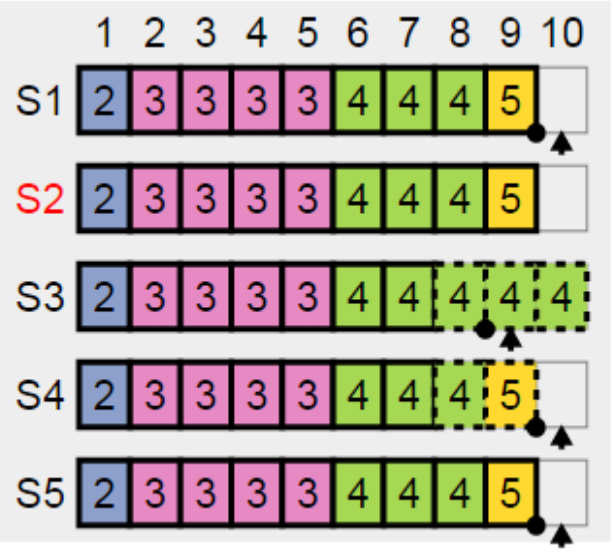
- Log Matching Property。

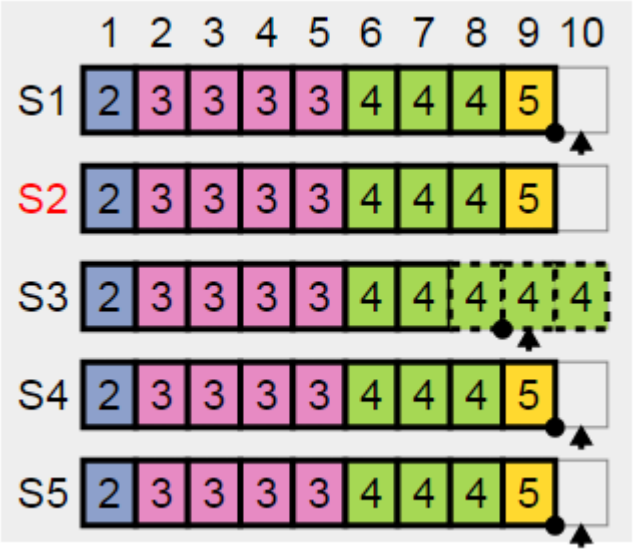


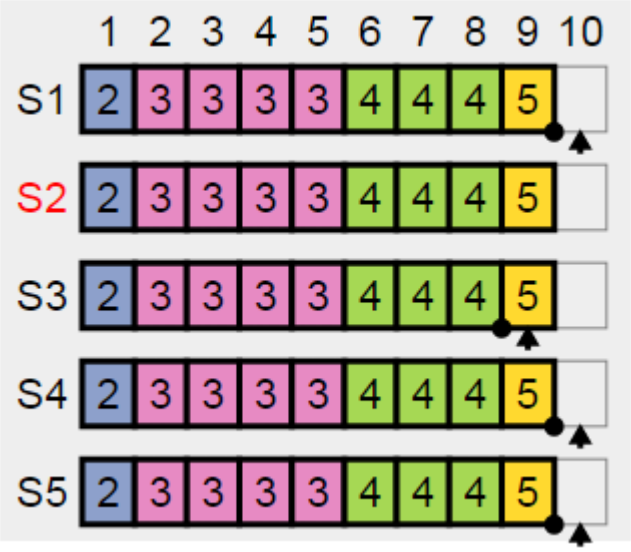




- 8和9一块同步的



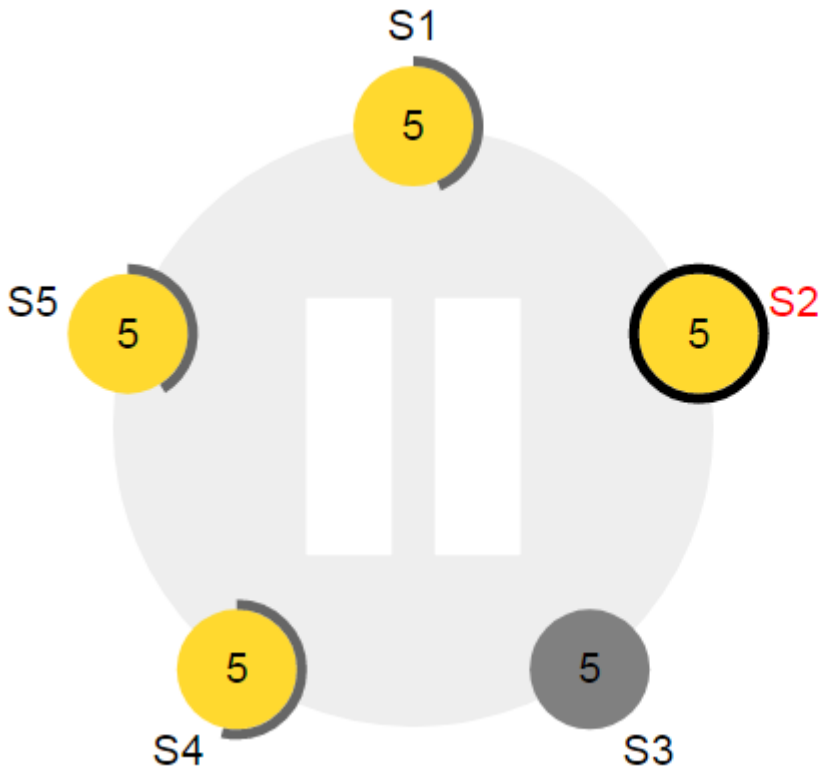


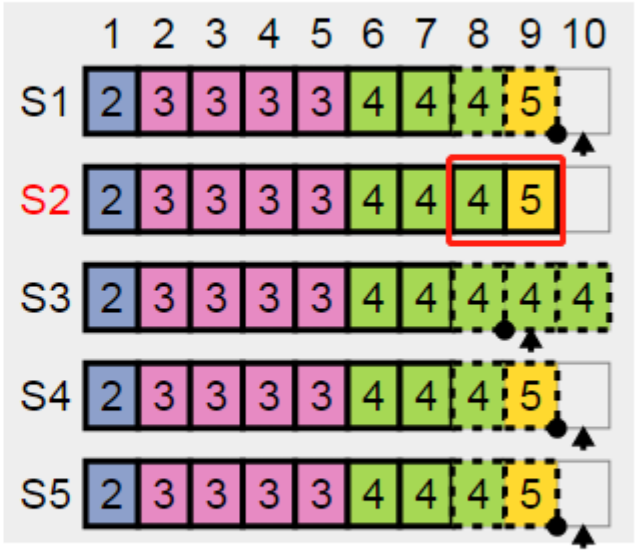


- S3 恢复

Raft:日志复制理解 配图（4）

➢ **Log Matching Property。**





If two entries in different logs have the same index and term,

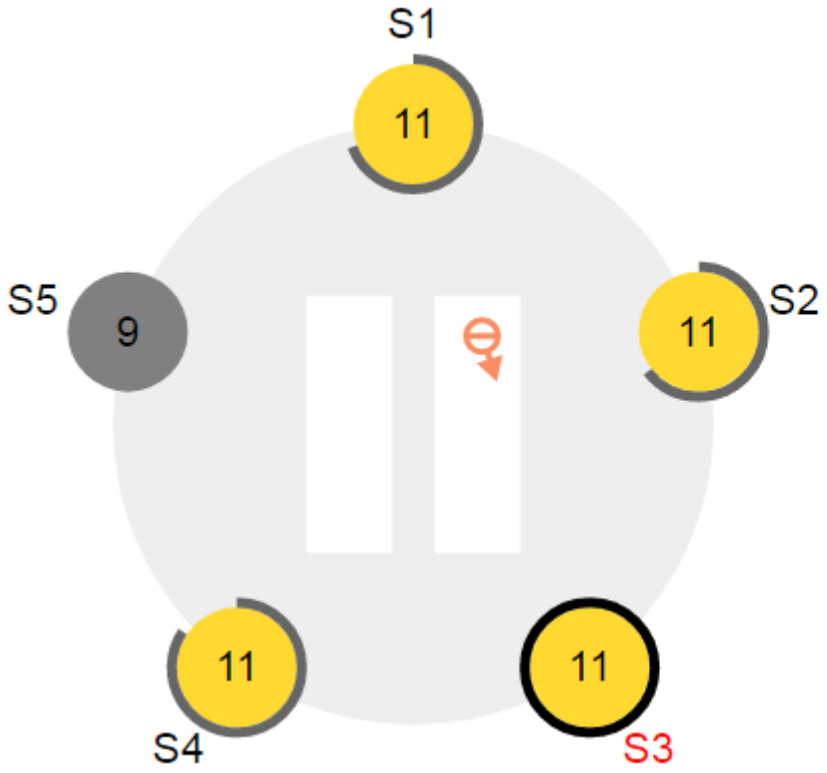
✓ they store the same command.
✓the logs are identical in all preceding entries

• If the logs have last entries with different terms, then the log with the later term is more up-to-date

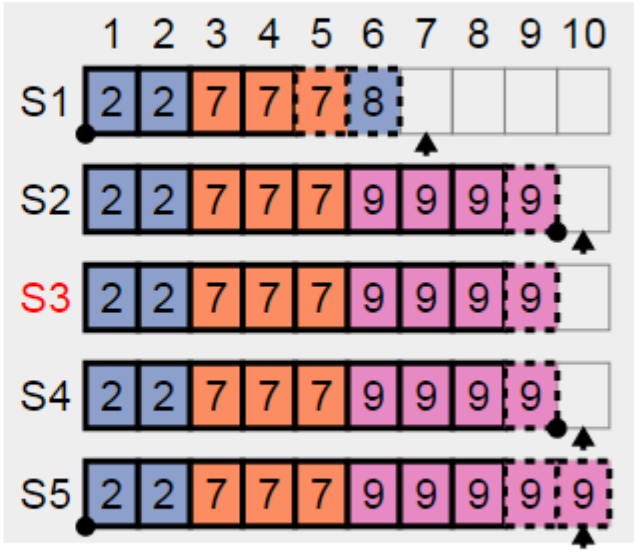
•If the logs end with the same term, then whichever log is longer is more up-to-date

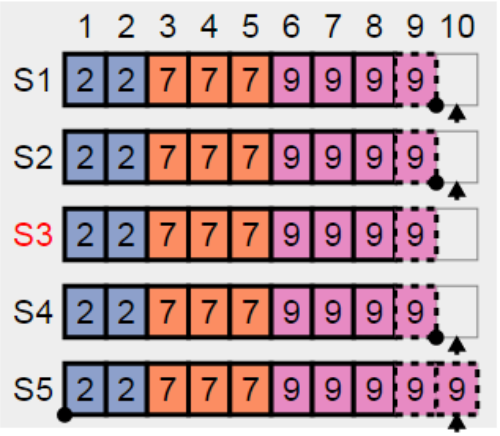
➢ 当前任期5索引9 已经提交，上一任期4的索引8状态默认变成已经提交状态

Raft:一致性协议 理解 配图（5）

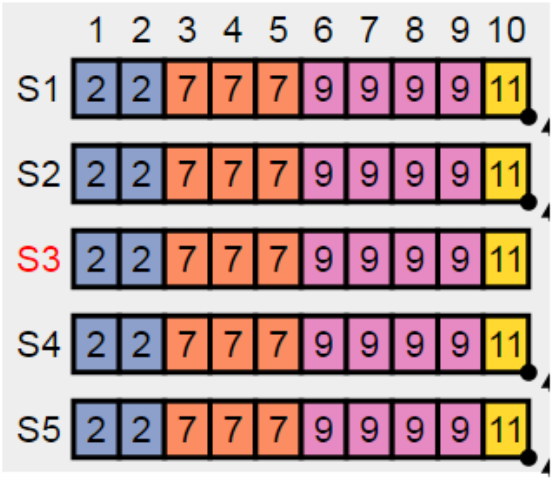


➢ Client no req





➢ Client new req



➢ 总结：故障转移后，新领导优先处理当前任期请求，然后处理上一任期遗留数据。

QA

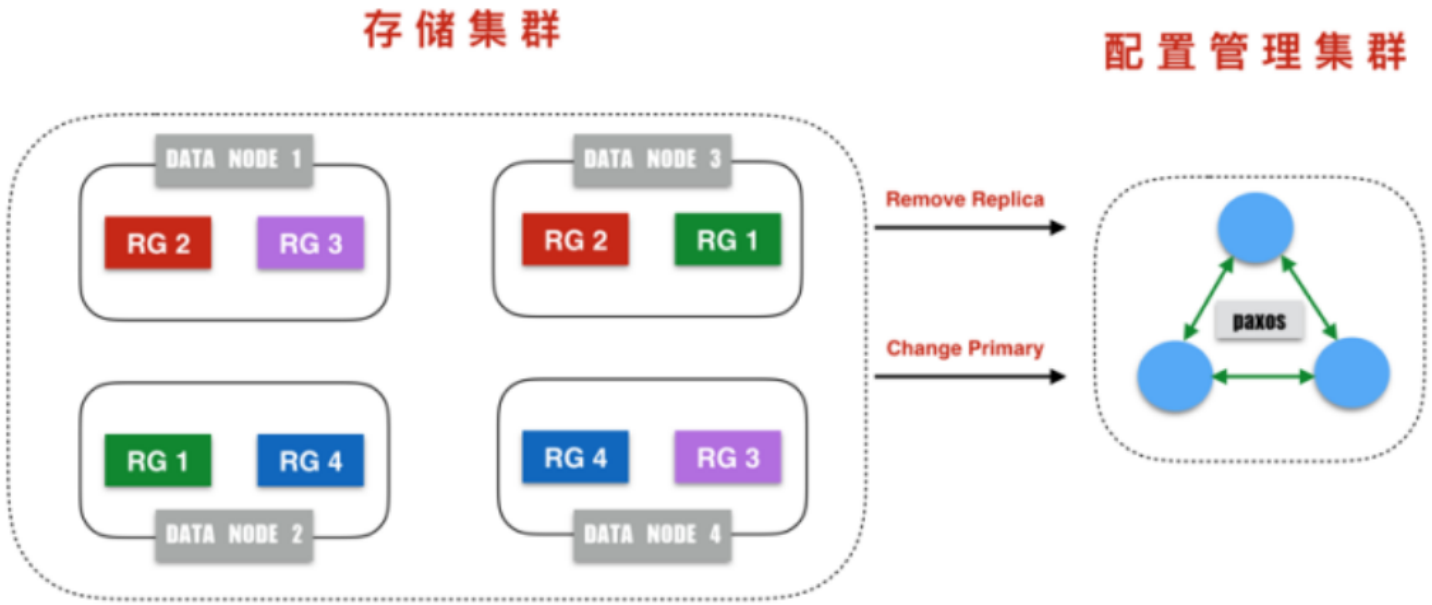
1. 假如 raft-A对客户端没有相应。请求sq 在ack后只有部分写入成功C，但是没有被选举到领导，已经提交记录，被新领导B同步成没有提交,此时如何恢复呢？该记录最后是否提交呢?

➢ 总结：故障转移后，新领导优先处理当前任期请求，然后处理上一任期遗留数据。

Pacific A: Replication 协议 理解 配图（1）

➢ 产品

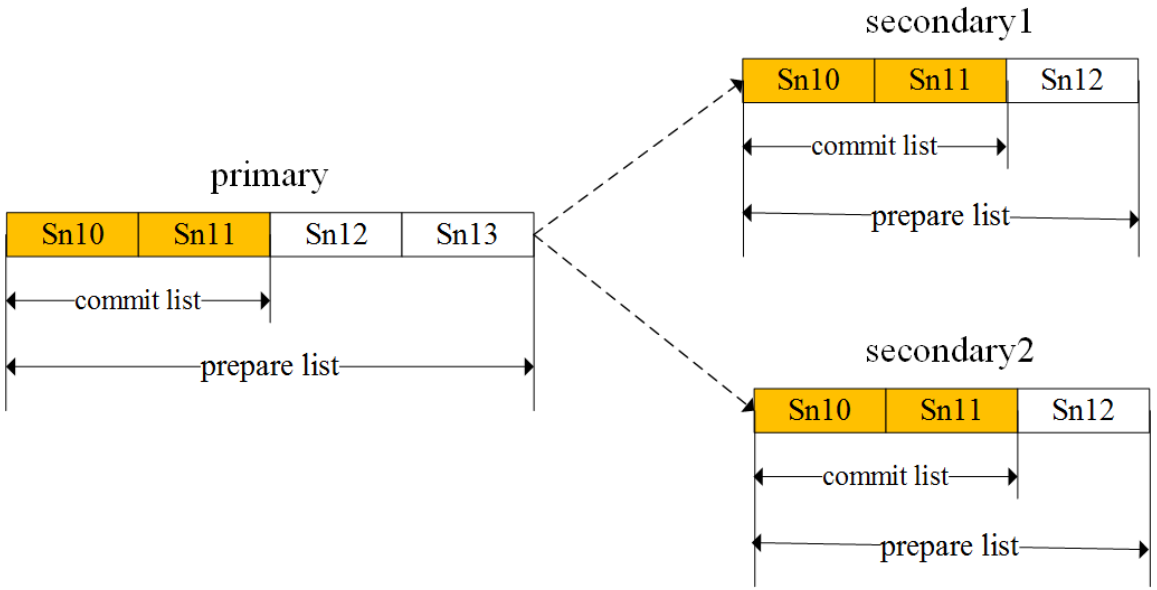
国内小米的开源分布式 key-value pegasus 也是基于 Pacifica 构建。



➢ 重要应用

➢ Commit Invariant

Commit Invariant: Let p be the primary and q be any replica in the current configuration, $\text{committed}_q \subseteq \text{committed}_p \subseteq \text{prepared}_q$ holds.



写入模型

a primary adds a request into its committed list (when moving the committed point forward) only after all replicas have inserted it into their prepared list,
the committed list on the primary is always a prefix of the prepared list on any replica

➢ 提交不变性 一般主的commitd index大于 任何其他节点的。最终保持统一

Pacific A: Replication 一致性协议 理解 配图（3）

➢ **Change of Primary**

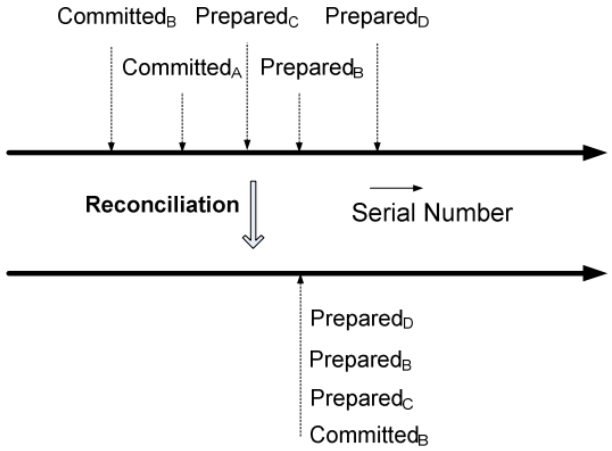


Figure 1: Reconciliation: An Example. A was the primary in the old configuration. In the new configuration, an old secondary B is promoted to be the new primary with A removed from the configuration. The first line shows the state of the prepared lists and the committed lists of the replicas based on the highest serial numbers on the lists. The second line shows the corresponding state after reconciliation.

故障模型

如何保证已经 committed 的数据不会被删除

During reconciliation, p sends prepare messages for uncommitted requests in its prepared list and have them committed on the new configuration

new Primary (Leader B)， 虽然知道 commitB point

并不知道节点C commitC point， 尤其Old leader commit A point 在哪里、这样模糊了！

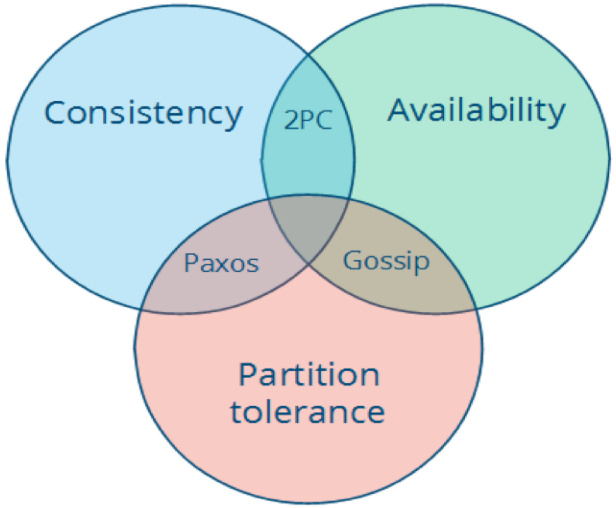
➢ **Leader B 讲当前的 prepare B list 上的所有 log entry 都提交，PreB=ComitB**

如果像raft那样，完全同步新leader 对吗？

CAP 理论线性一致性 配图（1）

➢ Linearizability vs Serializability

En:http://www.bailis.org/blog/linearizability-versus-serializability/



隔离级别	脏读（Dirty Read）	不可重复读（NonRepeatable Read）	幻读（Phantom Read）
未提交读（Read uncommitted）	可能	可能	可能
已提交读（Read committed）	不可能	可能	可能
可重复读（Repeatable read）	不可能	不可能	可能
可串行化（Serializable ）	不可能	不可能	不可能

Linearizability: single-operation, single-object, real-time order， CAP中的C ,读取到一定是最新写入的值

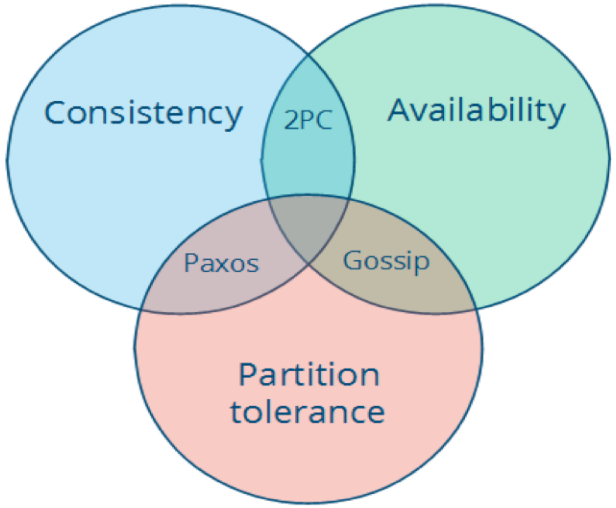
Serializability: multi-operation, multi-object, arbitrary total order， Serializability是ACID中Isolation

➢ AICD CAP

CAP 理论线性一致性 配图（2）

➢ Linearizability vs Serializability

En:http://www.bailis.org/blog/linearizability-versus-serializability/



隔离级别	脏读（Dirty Read）	不可重复读（NonRepeatable Read）	幻读（Phantom Read）
未提交读（Read uncommitted）	可能	可能	可能
已提交读（Read committed）	不可能	可能	可能
可重复读（Repeatable read）	不可能	不可能	可能
可串行化（Serializable ）	不可能	不可能	不可能

Linearizability: single-operation, single-object, real-time order， CAP中的C ,读取到一定是最新写入的值

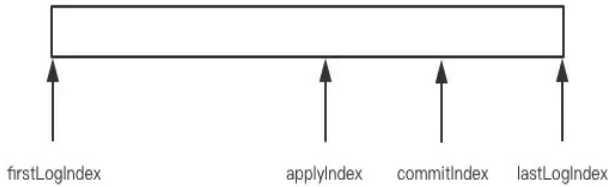
Serializability: multi-operation, multi-object, arbitrary total order， Serializability是ACID中Isolation

➢ AICD CAP

CAP 理论线性一致性 配图（2）

➢ Raft 日志同步

当状态机的 apply index 大于或等于 commited index 时才读取数据并返回.



Leader当选后为什么要立即同步一个no-op日志?

➢ AICD CAP

CAP 理论线性一致性 配图（2）

➢ Raft 日志同步

	may send more than one for efficiency)
leaderCommit	leader's commitIndex

Results:

term	currentTerm, for leader to update itself
success	true if follower contained entry matching prevLogIndex and prevLogTerm

Receiver implementation:

1. Reply false if term < currentTerm (§5.1)
2. Reply false if log doesn't contain an entry at prevLogIndex whose term matches prevLogTerm (§5.3)
3. If an existing entry conflicts with a new one (same index but different terms), delete the existing entry and all that follow it (§5.3)
4. Append any new entries not already in the log
5. If leaderCommit > commitIndex, set commitIndex = min(leaderCommit, index of last new entry)

➢ AICD CAP