Designing for Understandability: the Raft Consensus Algorithm

Diego Ongaro
John Ousterhout

Stanford University

PLATFORMLAB

Algorithms Should Be Designed For …

Correctness?

Efficiency?

Conciseness?

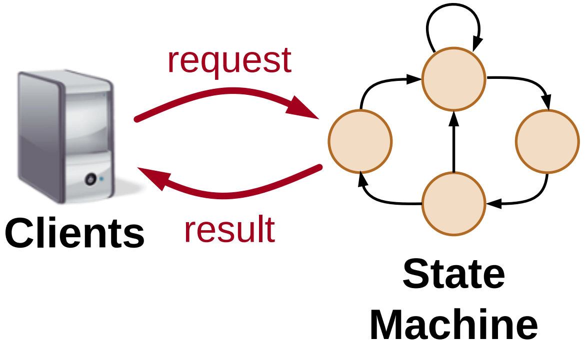
Understandability!

Overview

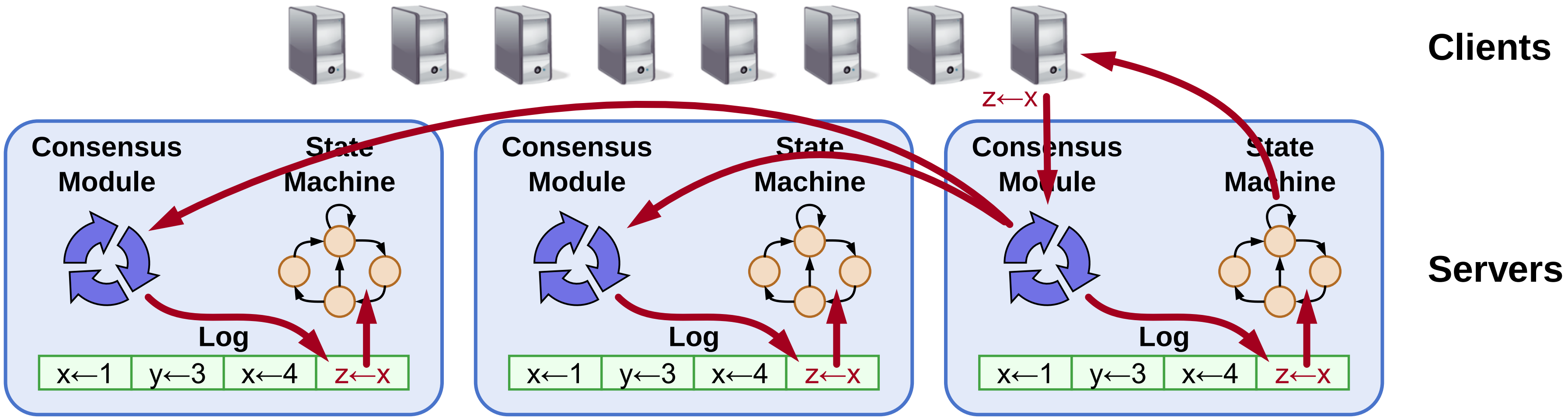
- **Consensus:**
  - Allows collection of machines to work as coherent group
  - Continuous service, even if some machines fail
- **Paxos has dominated discussion for 25 years**
  - Hard to understand
  - Not complete enough for real implementations
- **New consensus algorithm: Raft**
  - Primary design goal: understandability (intuition, ease of explanation)
  - Complete foundation for implementation
  - Different problem decomposition
- **Results:**
  - User study shows Raft more understandable than Paxos
  - Widespread adoption

State Machine

- **Responds to external stimuli**
- **Manages internal state**
- **Examples: many storage systems, services**
  - Memcached
  - RAMCloud
  - HDFS name node
  - ...



Replicated State Machine

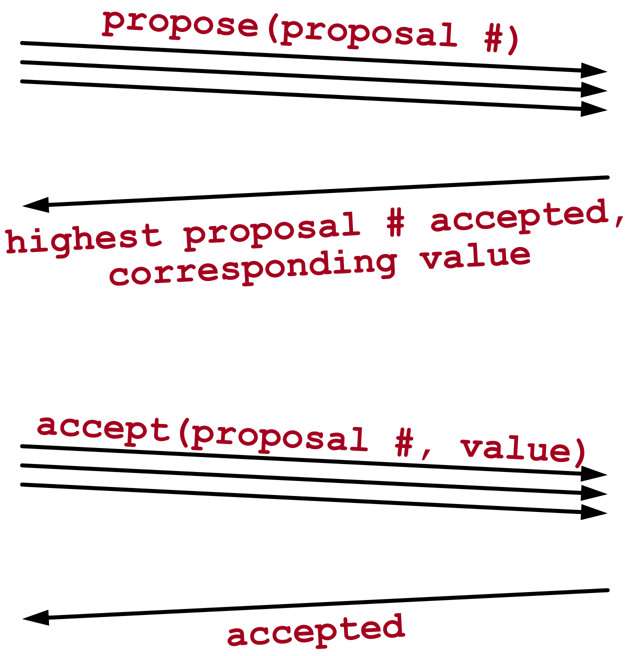


- **Replicated log ensures state machines execute same commands in same order**
- **Consensus module ensures proper log replication**
- **System makes progress as long as any majority of servers are up**
- **Failure model: delayed/lost messages, fail-stop (not Byzantine)**

Paxos (Single Decree)

Proposers | **Acceptors**

Choose unique proposal #



proposal # > any previous?

Majority? Select value for highest proposal # returned; if none, choose own value

proposal # >= any previous?

Majority? Value chosen

Paxos Problems

- **Impenetrable: hard to develop intuitions**
  - Why does it work?
  - What is the purpose of each phase?
- **Incomplete**
  - Only agrees on single value
  - Doesn't address liveness
  - Choosing proposal values?
  - Cluster membership management?
- **Inefficient**
  - Two rounds of messages to choose one value
- **No agreement on the details**

> *"The dirty little secret of the NSDI community is that at most five people really, truly understand every part of Paxos :-)"*
> *— NSDI reviewer*

> *"There are significant gaps between the description of the Paxos algorithm and the needs of a real-world system ... the final system will be based on an unproven protocol"*
> *— Chubby authors*

Not a good foundation for practical implementations

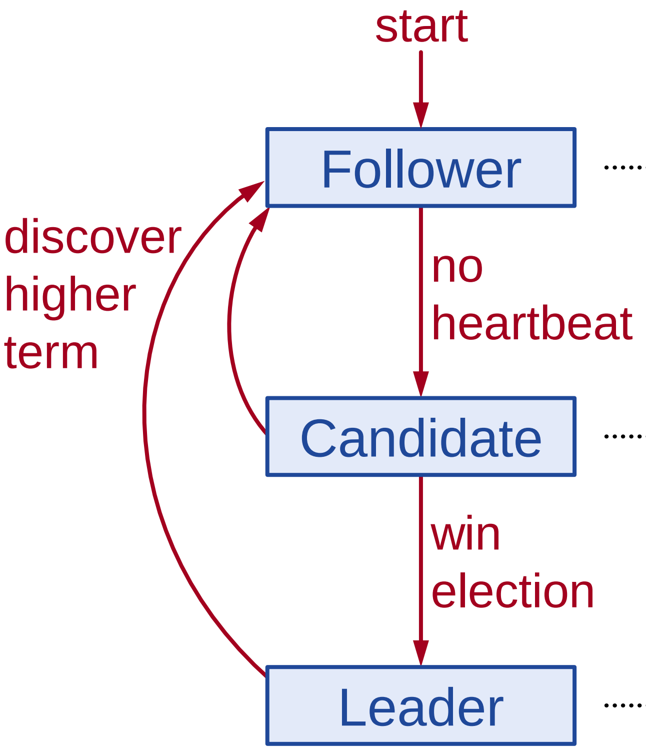
Raft Challenge

- **Is there a different consensus algorithm that's easier to understand?**
- **Make design decisions based on understandability:**
  - Which approach is easier to explain?
- **Techniques:**
  - Problem decomposition
  - Minimize state space
    - Handle multiple problems with a single mechanism
    - Eliminate special cases
    - Maximize coherence
    - Minimize nondeterminism

Raft Decomposition

1. **Leader election:**
   - Select one server to act as leader
   - Detect crashes, choose new leader
2. **Log replication (normal operation)**
   - Leader accepts commands from clients, appends to its log
   - Leader replicates its log to other servers (overwrites inconsistencies)
3. **Safety**
   - Keep logs consistent
   - Only servers with up-to-date logs can become leader

Server States and RPCs

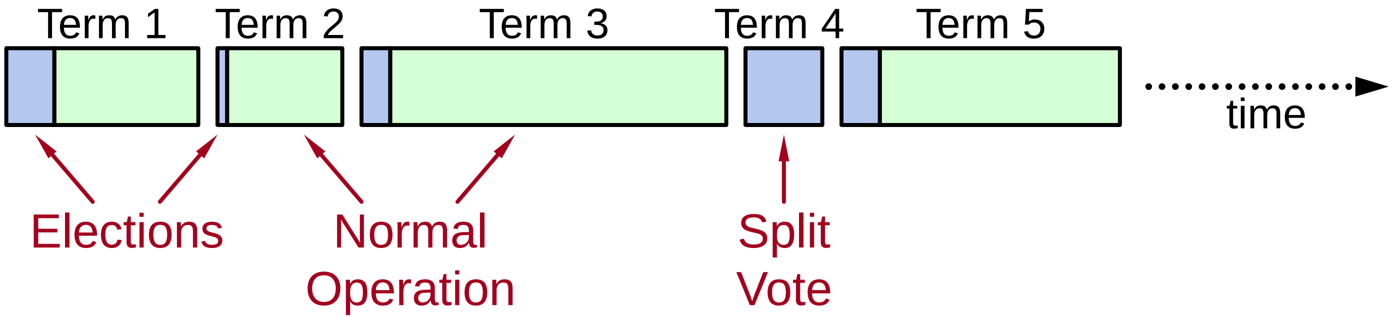


Passive (but expects regular heartbeats)

Issues **RequestVote** RPCs to get elected as leader

Issues **AppendEntries** RPCs:
- Replicate its log
- Heartbeats to maintain leadership

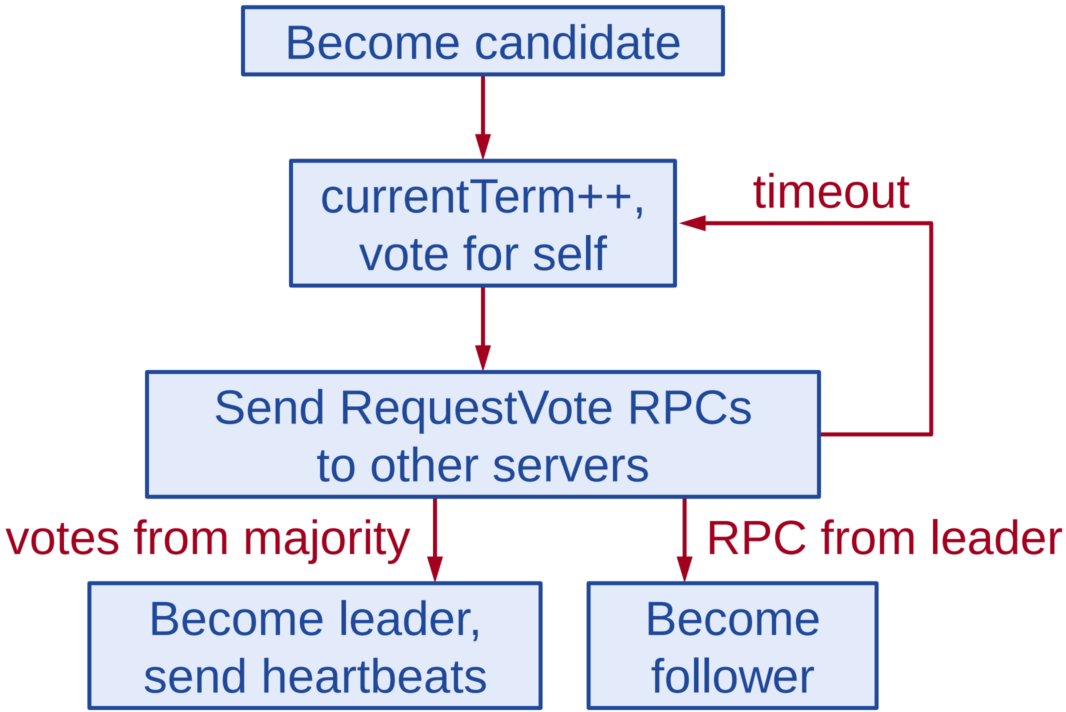
Terms



- **At most 1 leader per term**
- **Some terms have no leader (failed election)**
- **Each server maintains current term value (no global view)**
  - Exchanged in every RPC
  - Peer has later term? Update term, revert to follower
  - Incoming RPC has obsolete term? Reply with error

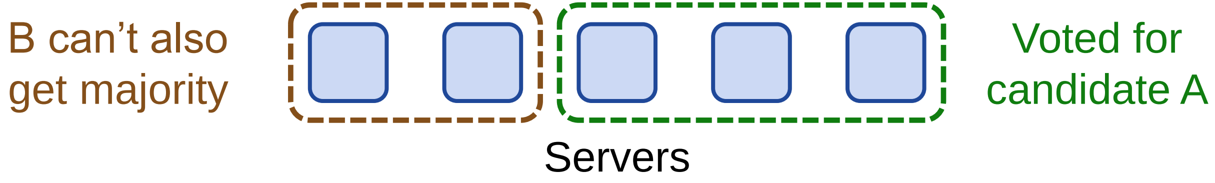
Terms identify obsolete information

Leader Election



Election Correctness

- **Safety: allow at most one winner per term**
  - Each server gives only one vote per term (persist on disk)
  - Majority required to win election

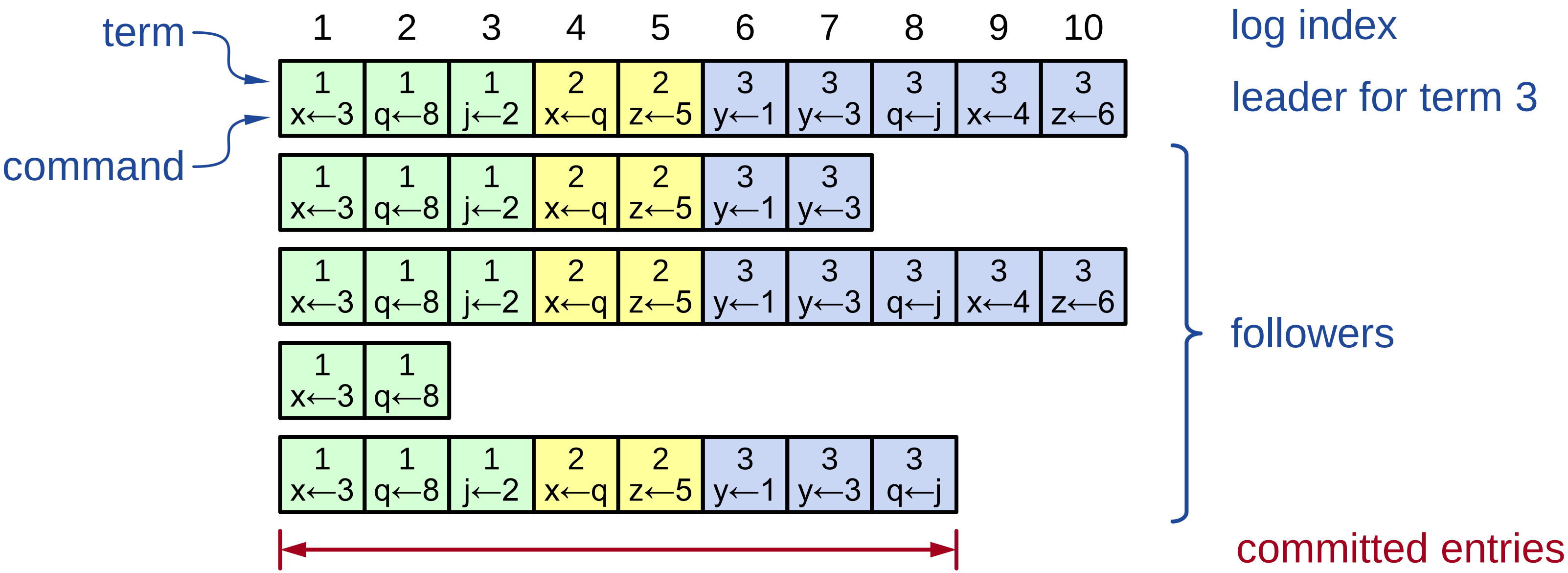


- **Liveness: some candidate must eventually win**
  - Choose election timeouts randomly in [T, 2T] (e.g. 150-300 ms)
  - One server usually times out and wins election before others time out
  - Works well if T >> broadcast time
- **Randomized approach simpler than ranking**

Normal Operation

- **Client sends command to leader**
- **Leader appends command to its log**
- **Leader sends AppendEntries RPCs to all followers**
- **Once new entry committed:**
  - Leader executes command in its state machine, returns result to client
  - Leader notifies followers of committed entries in subsequent AppendEntries RPCs
  - Followers execute committed commands in their state machines
- **Crashed/slow followers?**
  - Leader retries AppendEntries RPCs until they succeed
- **Optimal performance in common case:**
  - One successful RPC to any majority of servers

Log Structure



- **Must survive crashes (store on disk)**
- **Entry committed if safe to execute in state machines**
  - Replicated on majority of servers by leader of its term

Log Inconsistencies

Crashes can result in log inconsistencies:

log index	1	2	3	4	5	6	7	8	9	10
s_1 (leader for term 4)	1 x←3	1 q←8	1 j←2	2 x←q	2 z←5	3 y←1	3 y←3	3 q←j		
s_2 (followers)	1 x←3	1 q←8	1 j←2	2 x←q	2 z←5	3 y←1	3 y←3			
s_3 (followers)	1 x←3	1 q←8	1 j←2	2 x←q	2 z←5	3 y←1	3 y←3	3 q←j	3 x←4	
s_4 (followers)	1 x←3	1 q←8								
s_5 (followers)	1 x←3	1 q←8	1 j←2	2 x←q	2 z←5	2 y←3	2 q←j	2 x←8	2 x←4	

Raft minimizes special code for repairing inconsistencies:

- Leader assumes its log is correct
- Normal operation will repair all inconsistencies

Log Matching Property

Goal: high level of consistency between logs

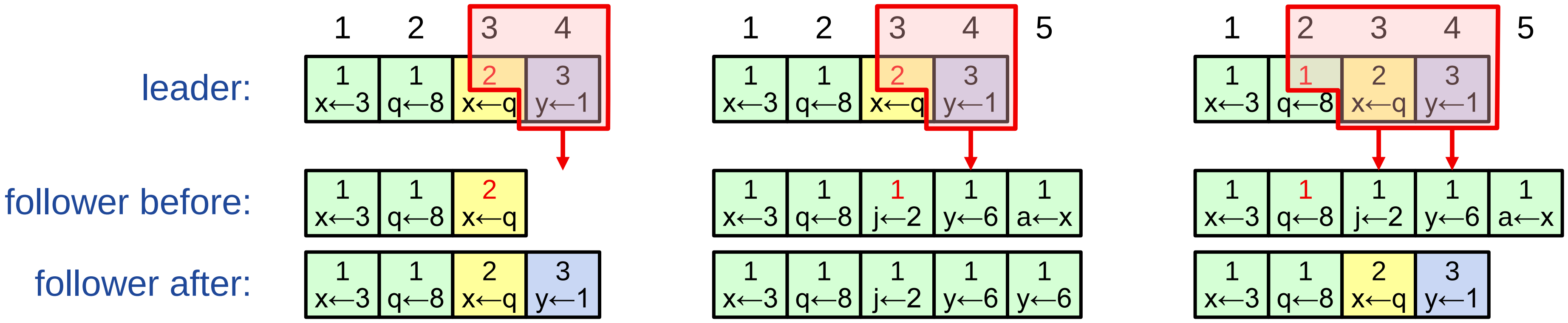
- **If log entries on different servers have same index and term:**
  - They store the same command
  - The logs are identical in all preceding entries

1	2	3	4	5	6	7	8	9	10
1 x←3	1 q←8	1 j←2	2 x←q	2 z←5	3 y←1	3 y←3	3 q←j	3 x←4	3 z←6
1 x←3	1 q←8	1 j←2	2 x←q	2 z←5	3 y←1	4 x←z	4 y←7		

- **If a given entry is committed, all preceding entries are also committed**

AppendEntries Consistency Check

- **AppendEntries RPCs include <index, term> of entry preceding new one(s)**
- **Follower must contain matching entry; otherwise it rejects request**
  - Leader retries with lower log index
- **Implements an induction step, ensures Log Matching Property**



Example #1: success

Example #2: mismatch

Example #3: success

Safety: Leader Completeness

- **Once log entry committed, all future leaders must store that entry**
- **Servers with incomplete logs must not get elected:**
  - Candidates include index and term of last log entry in RequestVote RPCs
  - Voting server denies vote if its log is more up-to-date
  - Logs ranked by <lastTerm, lastIndex>

Leader election for term 4:

	1	2	3	4	5	6	7	8	9
s_1	1	1	1	2	2	3	3	3	
s_2	1	1	1	2	2	3	3		
s_3	1	1	1	2	2	3	3	3	3
s_4	1	1	1	2	2	3	3	3	
s_5	1	1	1	2	2	2	2	2	2

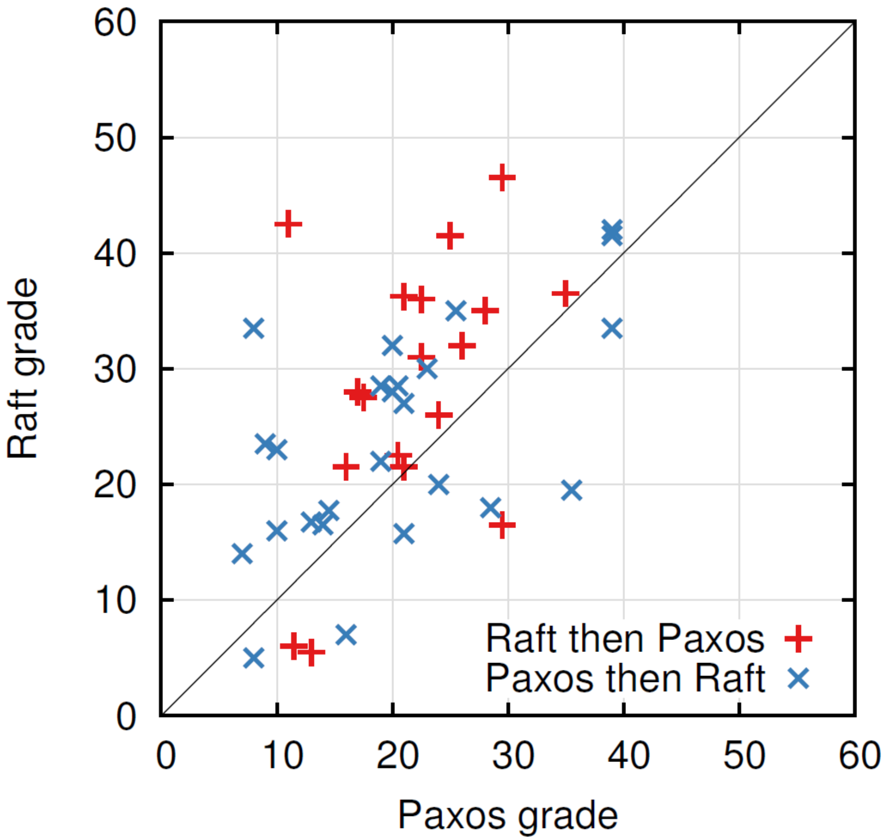
Raft Evaluation

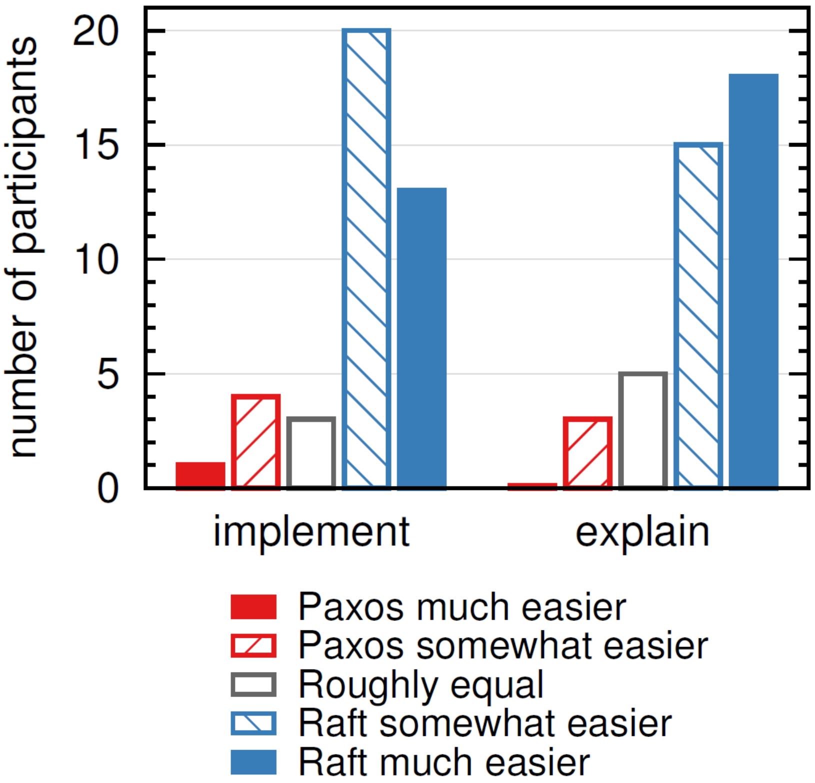
- **Formal proof of safety**
  - Ongaro dissertation
  - UW mechanically checked proof (50 klines)
- **C++ implementation (2000 lines)**
  - 100’s of clusters deployed by Scale Computing
- **Performance analysis of leader election**
  - Converges quickly even with 12-24 ms timeouts
- **User study of understandability**

User Study: Is Raft Simpler than Paxos?

- **43 students in 2 graduate OS classes (Berkeley and Stanford)**
  - Group 1: Raft video, Raft quiz, then Paxos video, Paxos quiz
  - Group 2: Paxos video, Paxos quiz, then Raft video, Raft quiz
- **Instructional videos:**
  - Same instructor (Ousterhout)
  - Covered same functionality: consensus, replicated log, cluster reconfiguration
  - Fleshed out missing pieces for Paxos
  - Videos available on YouTube
- **Quizzes:**
  - Questions in 3 general categories
  - Same weightings for both tests
- **Experiment favored Paxos slightly:**
  - 15 students had prior experience with Paxos

User Study Results





Impact

Hard to publish:

- **Rejected 3 times at major conferences**
- **Finally published in USENIX ATC 2014**
- **Challenges:**
  - PCs uncomfortable with understandability as metric
  - Hard to evaluate
  - Complexity impresses PCs

Widely adopted:

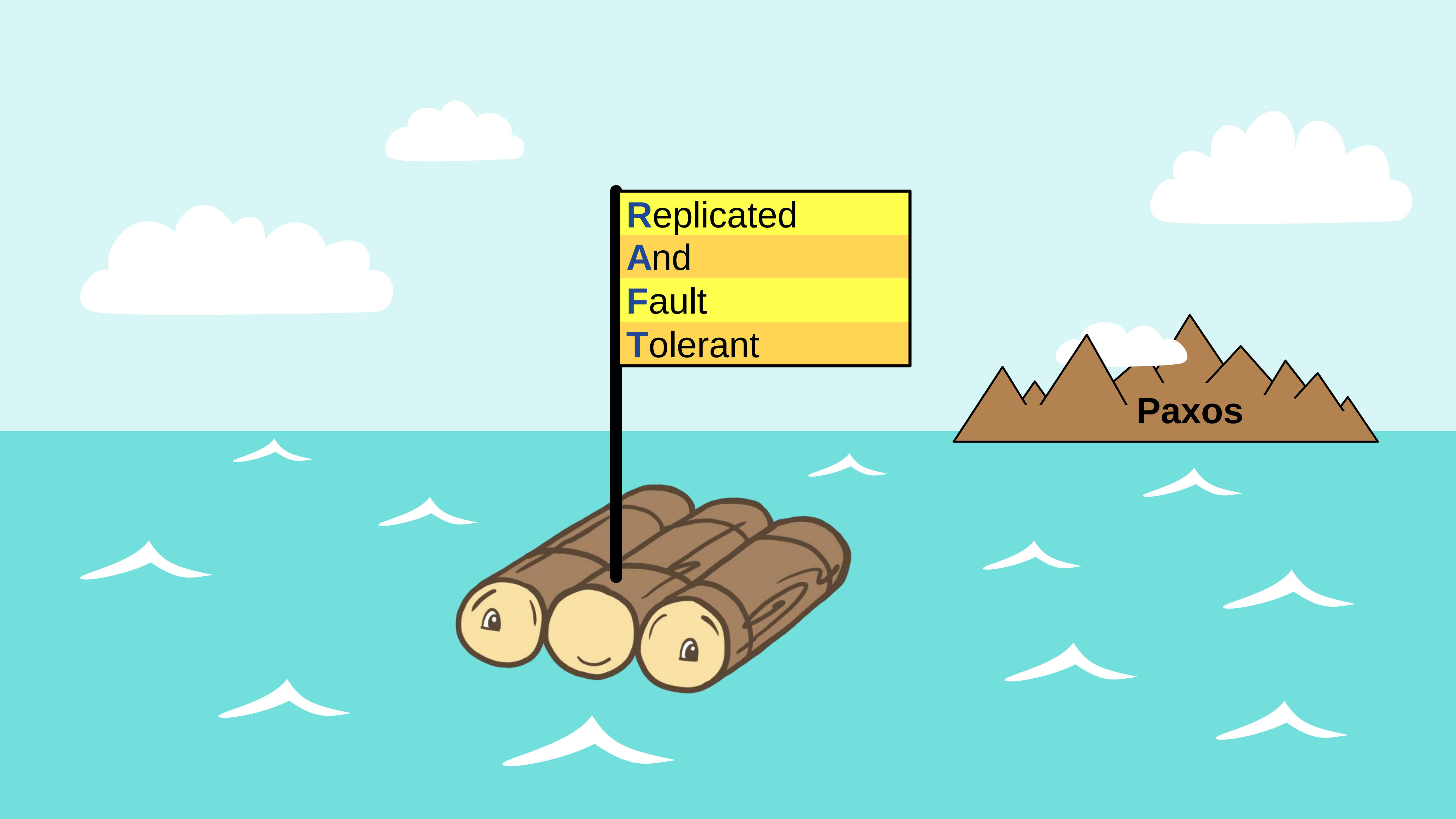
- **25 implementations before paper published**
- **83 implementations currently listed on Raft home page**
- **>10 versions in production**
- **Taught in graduate OS classes**
  - MIT, Stanford, Washington, Harvard, Duke, Brown, Colorado, ...

Additional Information

- **Other aspects of Raft (see paper or Ongaro dissertation):**
  - Communication with clients (linearizability)
  - Cluster liveness
  - Log truncation
- **Other consensus algorithms:**
  - Viewstamped Replication (Oki & Liskov, MIT)
  - ZooKeeper (Hunt, Konar, Junqueira, Read, Yahoo!)

Conclusions

- **Understandability deserves more emphasis in algorithm design**
  - Decompose the problem
  - Minimize state space
- **Making a system simpler can have high impact**
- **Raft better than Paxos for teaching and implementation:**
  - Easier to understand
  - More complete


Replicated
And
Fault
Tolerant
Paxos

Extra Slides

PLATFORMLAB

Raft Properties

- **Election Safety:** at most one leader can be elected in a given term
- **Leader Append-Only:** a leader never modifies or deletes entries in its log
- **Log Matching:** if two logs contain an entry with the same index and term, then the logs are identical in all entries up through the given index
- **Leader Completeness:** if a log entry is committed, then that entry will be present in the logs of all future leaders
- **State Machine Safety:** if a server has applied a log entry at a given index to its state machine, no other server will ever apply a different log entry for the same index

Leader Changes

- **Logs may be inconsistent after leader change**
- **No special steps by new leader:**
  - Start normal operation
  - Followers' logs will eventually match leader
- **Leader's log is "the truth"**