UNIVERSITY of VIRGINIA

CS4740
CLOUD COMPUTING

Reliability

Prof. Chang Lou, UVA CS, Fall 2025

AGENDA

- What is reliability
- Motivation for reliability research
- Software techniques to improve cloud reliability

- End of semester concluding remarks :)

WHAT IS RELIABILITY

— What are some common qualities we measure on systems?

WHAT IS RELIABILITY

- **Reliability is not**
  - Performance: make systems faster
  - Usability: make systems more user-friendly
  - Security: make systems safer against intrusions
  - Cost-effectiveness: make systems more affordable
- **Reliability is**
  - the system's ability to consistently perform its intended function without **failure** over a given period.

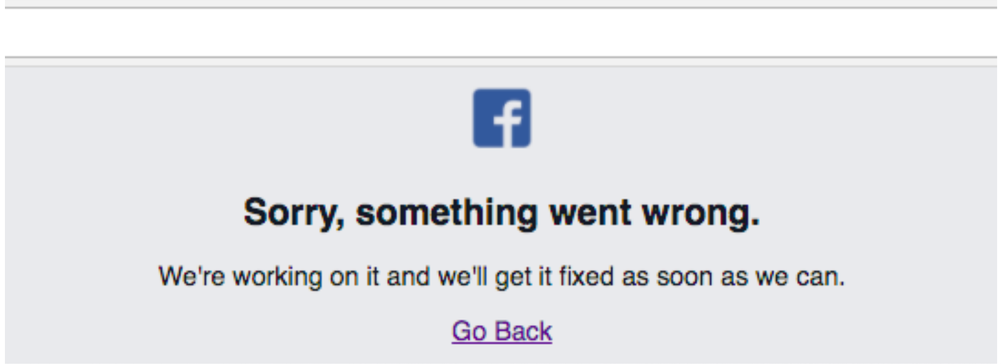
WHAT IS RELIABILITY

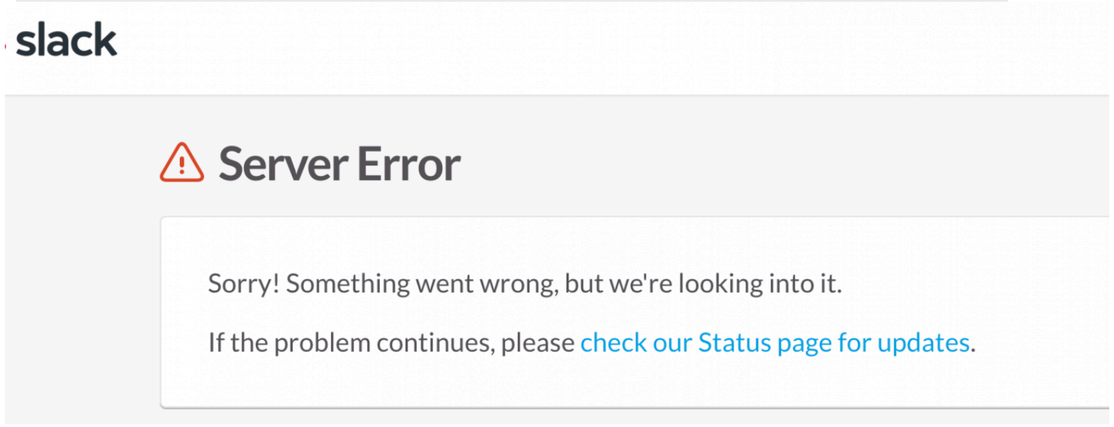
- **Reliability**
  - measured with the probability that a system operates without failure in a given period of time.
  - how to compute probability: Mean Time Between Failures (MTBF)

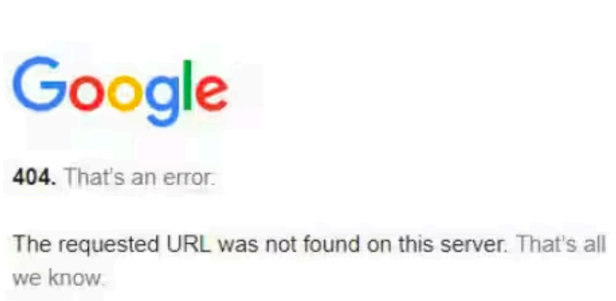
$$Reliability = 1 - \frac{1}{MTBF} = 1 - \frac{NumofBreakdowns}{E[uptime]}$$

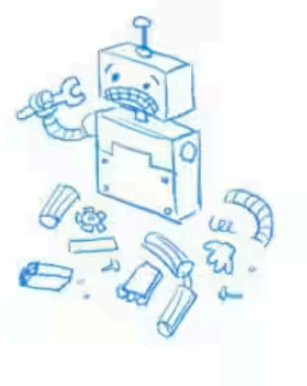
CLOUD FAILURES

— Cloud failures are prevalent









CLOUD FAILURES

— Bad user experience

CLOUD FAILURES

— Huge economic loss and service unavailability

Microsoft's MFA is so strong, it locked out users for 8 hours

23 May 2013

3 difficult days for Rackspace Cloud Load Balancers

Posted by **iwgcr**

After almost 24 hours of technical difficulties, Facebook is back

Facebook blamed the issue on a "server configuration change."

Amazon 'missed out on $34m in sales during internet outage'

The e-commerce giant generates $9,615 in sales per second – but not when it's website is down

Ben Chapman • Tuesday 08 June 2021 16:54 • 1 Comments

Millions online hit by Microsoft 365 outages

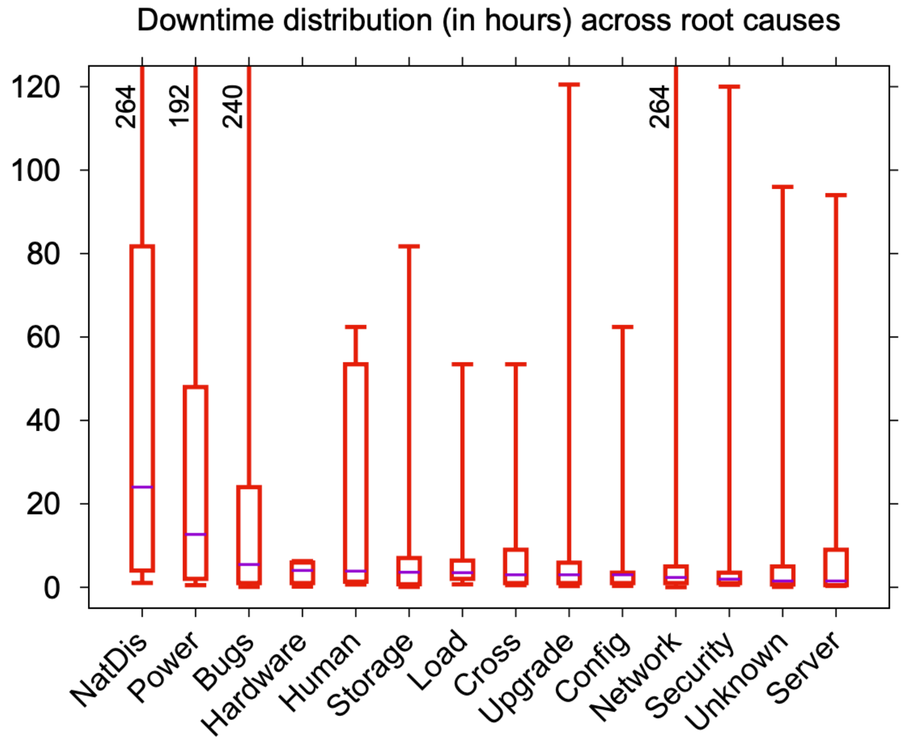
911 emergency services go down across the US after CenturyLink outage

Zack Whittaker @zackwhittaker / 4 months ago

Comment

CLOUD FAILURES

— Cloud systems fail due to different root causes



• H. S. Gunawi et al., Why Does the Cloud Stop Computing? Lessons from Hundreds of Service Outages, In Proceedings of the 6th ACM Symposium on Cloud Computing (SOCC '16), October 2016.

CLOUD FAILURES

— .. sometimes very weird root causes

TECH TECHNOLOGY GOOGLE FIBER

Google Fiber Shot Down By 'Bored' Hunters

'Bored' Hunters Shoot Down Google Fiber

By Bianca Bosker

Nov 22, 2010, 05:12 AM EST | **Updated** May 25, 2011, 05:50 PM EDT

Google reinforces undersea cables after shark bites

Sharks have been biting down on fibre optic cables under the Pacific, possibly confused by electrical signals that resemble fish

REMAINING PART OF LECTURE

– We focus on solutions for software bugs

TACKLING SOFTWARE ISSUES IN DIFFERENT WAYS

Bug finding	Formal methods	Runtime
Fuzz testing	**Model checking**	Failure detection
Static analysis	Symbolic execution	Failure diagnosis
Dynamic analysis	Theorem proving	Failure recovery
...	...	...
Can we automatically find bugs in the codes?	Can we prove the codes are bug-free?	Can we better handle failures at runtime?

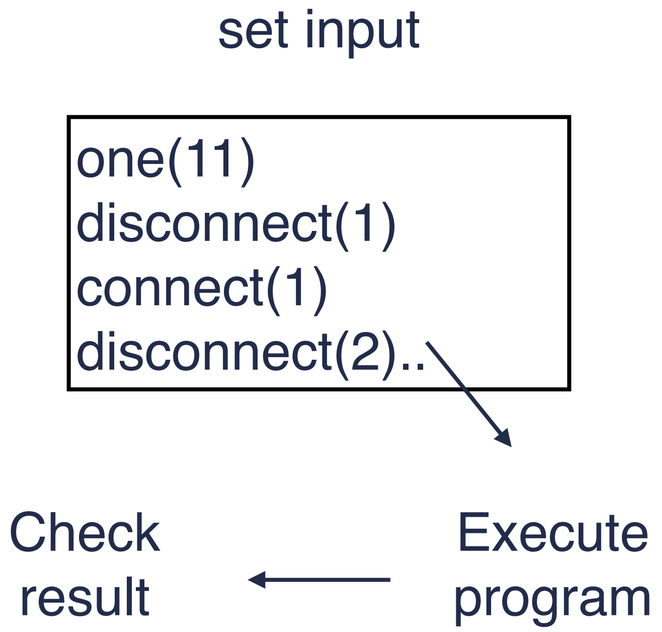
Testing (fuzzy)

TESTING

```go
func TestPersist12C(t *testing.T) {
    servers := 3
    cfg := make_config(t, servers, unreliable: false, snapshot: false)
    defer cfg.cleanup()

    cfg.begin( description: "Test (2C): basic persistence")

    cfg.one( cmd: 11, servers, retry: true)

    // crash and re-start all
    for i := 0; i < servers; i++ {
        cfg.start1(i, cfg.applier)
    }
    for i := 0; i < servers; i++ {
        cfg.disconnect(i)
        cfg.connect(i)
    }

    cfg.one( cmd: 12, servers, retry: true)

    leader1 := cfg.checkOneLeader()
    cfg.disconnect(leader1)
    cfg.start1(leader1, cfg.applier)
    cfg.connect(leader1)

    cfg.one( cmd: 13, servers, retry: true)
```

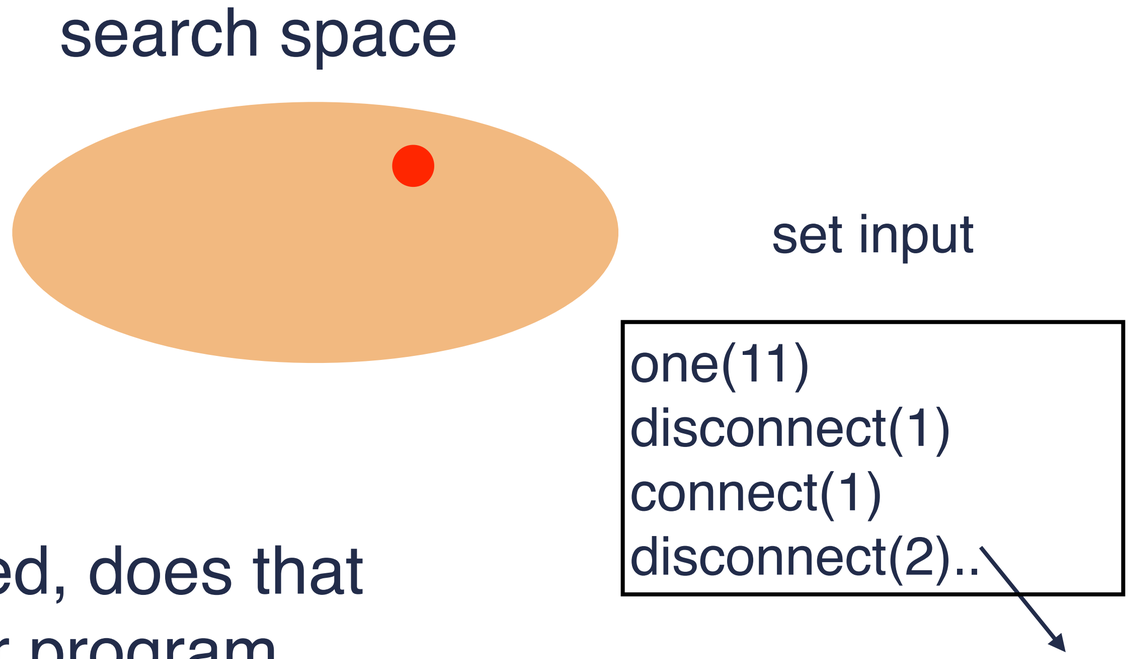
TESTING

```go
func TestPersist12C(t *testing.T) {
    servers := 3
    cfg := make_config(t, servers, unreliable: false, snapshot: false)
    defer cfg.cleanup()

    cfg.begin( description: "Test (2C): basic persistence")

    cfg.one( cmd: 11, servers, retry: true)

    // crash and re-start all
    for i := 0; i < servers; i++ {
        cfg.start1(i, cfg.applier)
    }
    for i := 0; i < servers; i++ {
        cfg.disconnect(i)
        cfg.connect(i)
    }

    cfg.one( cmd: 12, servers, retry: true)
```



TESTING

```go
func TestPersist12C(t *testing.T) {
    servers := 3
    cfg := make_config(t, servers, unreliable: false, snapshot: false)
    defer cfg.cleanup()

    cfg.begin( description: "Test (2C): basic persistence")

    cfg.one( cmd: 11, servers, retry: true)

    // crash and re-start all
    for i := 0; i < servers; i++ {
        cfg.start1(i, cfg.applier)
    }
    for i := 0; i < servers; i++ {
        cfg.disconnect(i)
        cfg.connect(i)
    }

    cfg.one( cmd: 12, servers, retry: true)
```

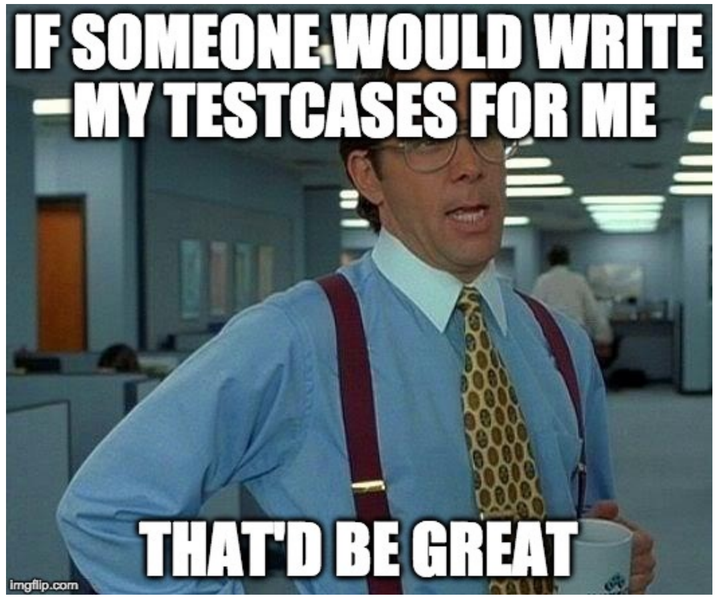


Test passed, does that mean your program has no bug?

tests only cover a small portion of possibilities!

FUZZ TESTING

- Goal:
  - To find program inputs that reveal a bug
- Approach:
  - Generate inputs randomly until program reports errors



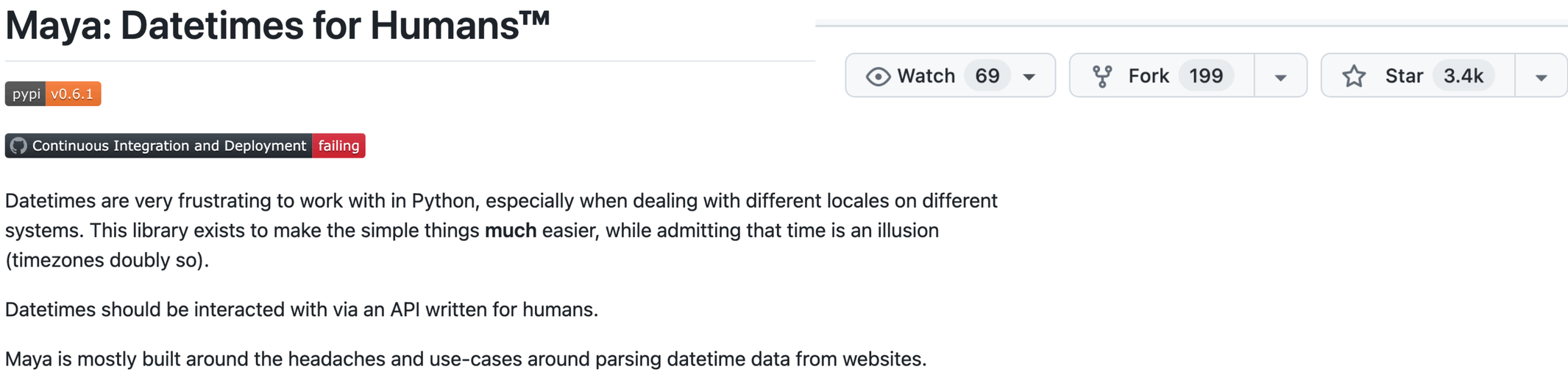
FUZZ TESTING EXAMPLE

- Standard HTTP GET request
  - § GET /index.html HTTP/1.1

- Fuzzing HTTP GET request
  - § AAAAAA...AAAA /index.html HTTP/1.1
  - § GET ///////index.html HTTP/1.1
  - § GET %n%n%n%n%n%n.html HTTP/1.1
  - § GET /AAAAAAAAAAAAA.html HTTP/1.1
  - § GET /index.html HTTTTTTTTTTTTTP/1.1

FUZZ TESTING EXAMPLE 2: OPEN-SOURCE SOFTWARE

- Many open-sourced fuzzer implementation
  - e.g., Atheris: A Coverage-Guided, Native Python Fuzzer from Google



FUZZ TESTING EXAMPLE 2: OPEN-SOURCE SOFTWARE

- Many open-sourced fuzzer implementation
  - e.g., Atheris: A Coverage-Guided, Native Python Fuzzer from Google

```
>>> scraped = '2016-12-16
18:23:45.423992+00:00'
>>> maya.parse(scraped).datetime()

datetime.datetime(2016, 12, 16, 13, 23, 45,
423992)
```

Maya: Python
Datetimes Library

```
>>> maya.parse('may15,2021').datetime()

datetime.datetime(2022, 5, 15, 0, 0, tzinfo=)
```

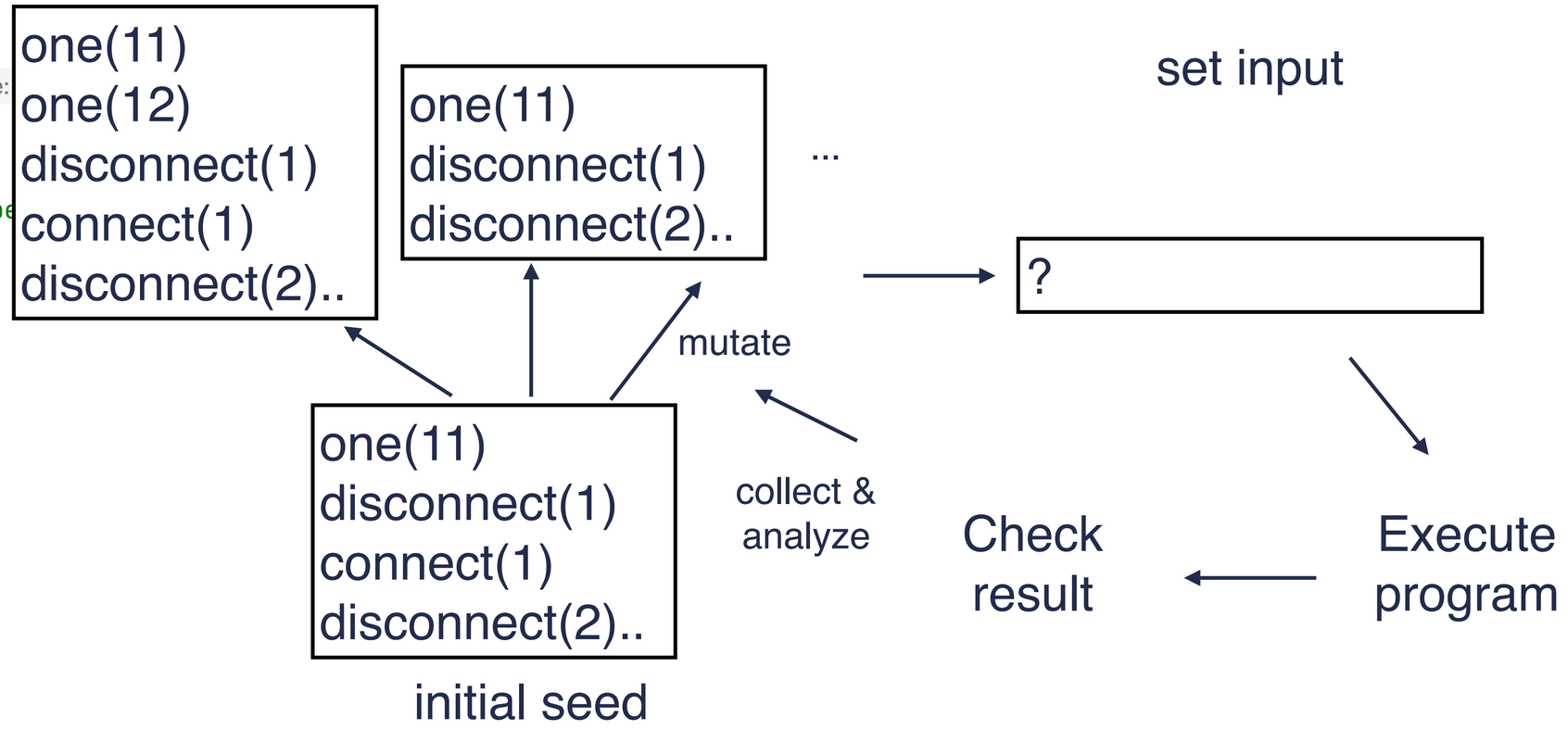
Applying fuzzer to find a
triggering input

FUZZ TESTING EXAMPLE

- How to fuzz testing a distributed system?
- Very challenging, especially considering all concurrency and non-determinism
  - here we show an intuitive approach

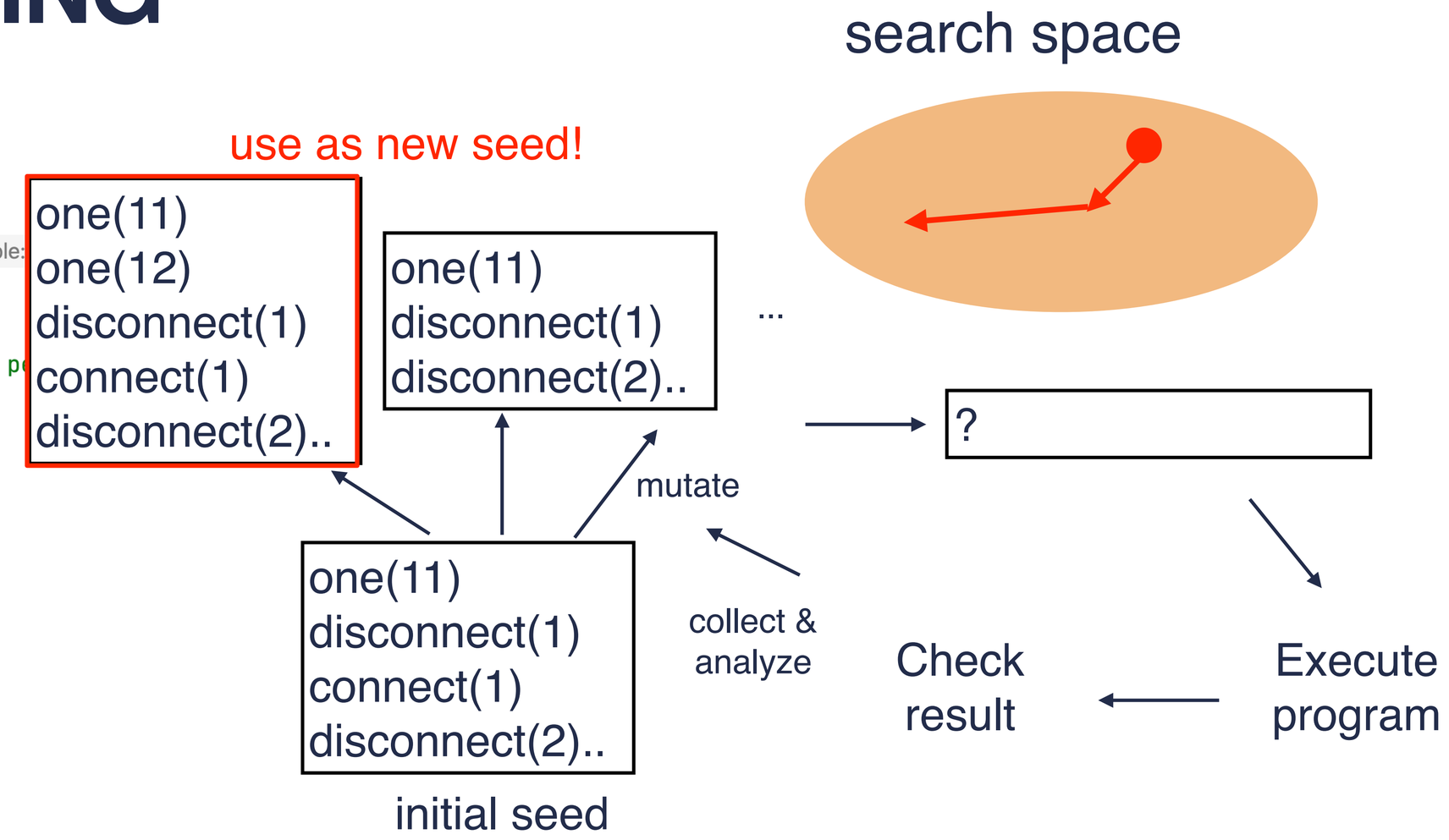
FUZZ TESTING

```go
func TestPersist12C(t *testing.T) {
    servers := 3
    cfg := make_config(t, servers, unreliable:
    defer cfg.cleanup()

    cfg.begin( description: "Test (2C): basic pe

    cfg.one( cmd: 11, servers, retry: true)

    // crash and re-start all
    for i := 0; i < servers; i++ {
        cfg.start1(i, cfg.applier)
    }
    for i := 0; i < servers; i++ {
        cfg.disconnect(i)
        cfg.connect(i)
    }

    cfg.one( cmd: 12, servers, retry: true)
```



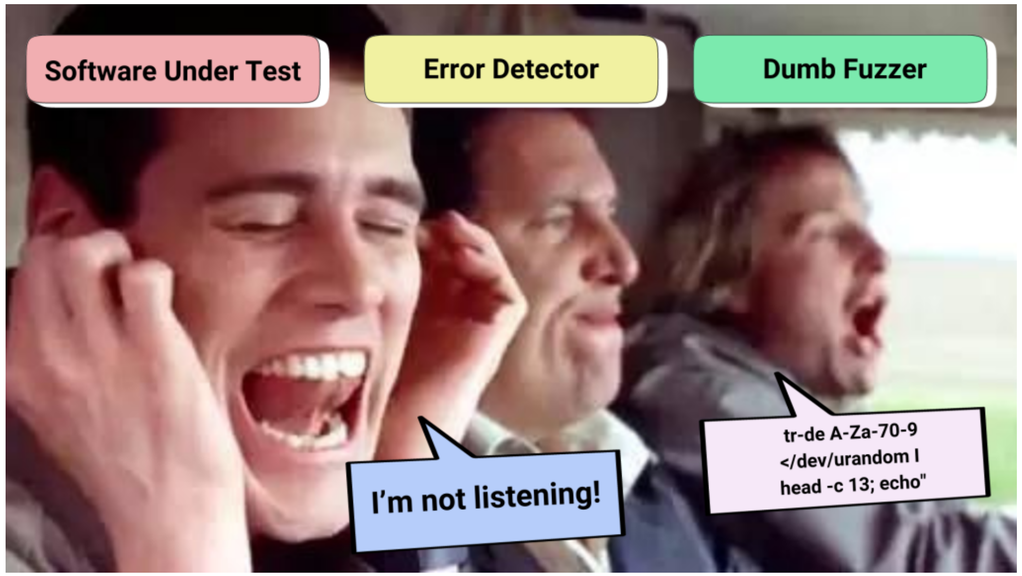
FUZZ TESTING

```go
func TestPersist12C(t *testing.T) {
    servers := 3
    cfg := make_config(t, servers, unreliable:
    defer cfg.cleanup()

    cfg.begin( description: "Test (2C): basic p

    cfg.one( cmd: 11, servers, retry: true)

    // crash and re-start all
    for i := 0; i < servers; i++ {
        cfg.start1(i, cfg.applier)
    }
    for i := 0; i < servers; i++ {
        cfg.disconnect(i)
        cfg.connect(i)
    }

    cfg.one( cmd: 12, servers, retry: true)
```



FUZZ TESTING

- Strength
  - low cost, easy-to-implement
  - practical for large programs
- Weakness
  - randomness
  - complexity of structured input
  - wasted efforts on rejected input

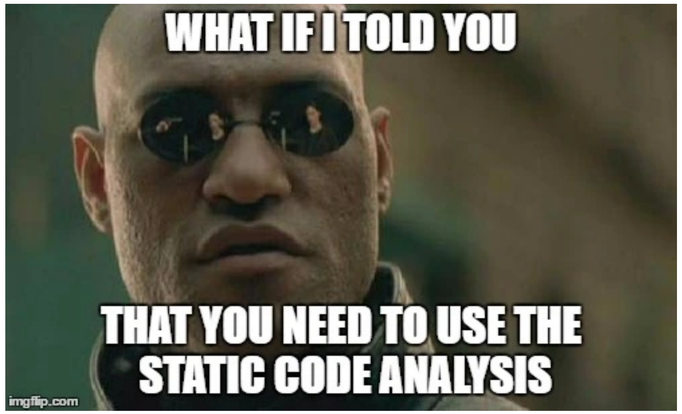


Static analysis

STATIC ANALYSIS

```
func (rf *Raft) RequestVote(args *RequestVoteArgs, reply *RequestVoteReply) {

   rf.mu.Lock()
   log.Printf("Worker%d: receive %v \n", rf.me, args)

   rf.CheckBehind(args.Term)

   reply.Term = rf.currentTerm
   if (rf.votedFor == -1 || rf.votedFor == args.CandidateId) && (args.LastLogTerm > rf.log[len(rf.log)-1].Term ||
      (args.LastLogTerm == rf.log[len(rf.log)-1].Term && args.LastLogIndex >= len(rf.log)-1)) {
      log.Printf("Worker%d: grant true %v %v %v \n", rf.me, rf.votedFor, rf.currentTerm, rf.commitIndex)
      rf.votedFor = args.CandidateId
      rf.currentTerm = args.Term
      rf.ifLeaderAlive = true
      rf.recentVoted = true
      log.Printf("Worker%d: become follower\n", rf.me)
      rf.role = Follower

      rf.persist()

      reply.VoteGranted = true
      return
   }
   reply.VoteGranted = false
   log.Printf("Worker%d: grant false %v %v %v \n", rf.me, rf.votedFor, rf.currentTerm, rf.commitIndex)

   rf.mu.Unlock()
}
```

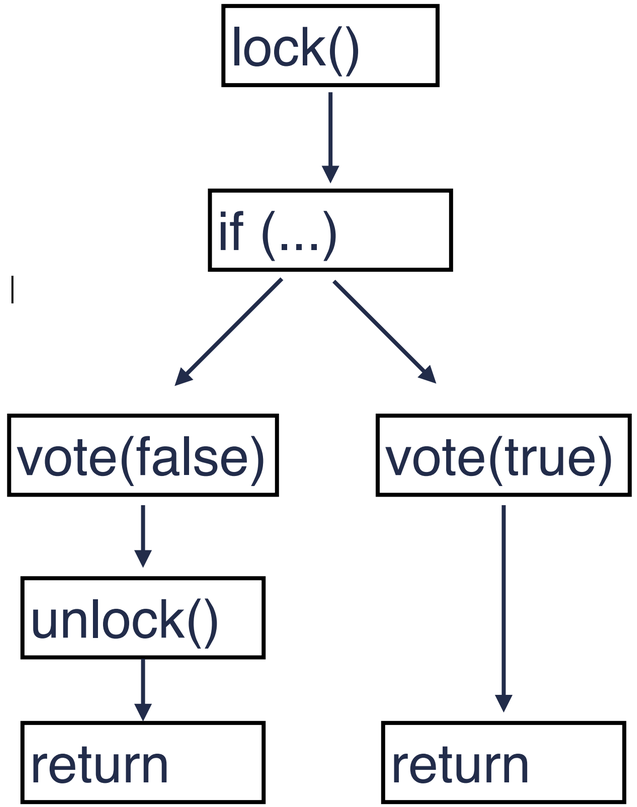
anything wrong with this code?



STATIC ANALYSIS

```
func (rf *Raft) RequestVote(args *RequestVoteArgs, reply *RequestVoteReply) {

   rf.mu.Lock()
   log.Printf("Worker%d: receive %v \n", rf.me, args)

   rf.CheckBehind(args.Term)

   reply.Term = rf.currentTerm
   if (rf.votedFor == -1 || rf.votedFor == args.CandidateId) && (args.LastLogTerm > rf.log[len(rf.log)-1].Term ||
      (args.LastLogTerm == rf.log[len(rf.log)-1].Term && args.LastLogIndex >= len(rf.log)-1)) {
      log.Printf("Worker%d: grant true %v %v %v \n", rf.me, rf.votedFor, rf.currentTerm, rf.commitIndex)
      rf.votedFor = args.CandidateId
      rf.currentTerm = args.Term
      rf.ifLeaderAlive = true
      rf.recentVoted = true
      log.Printf("Worker%d: become follower\n", rf.me)
      rf.role = Follower

      rf.persist()

      reply.VoteGranted = true
      return
   }
   reply.VoteGranted = false
   log.Printf("Worker%d: grant false %v %v %v \n", rf.me, rf.votedFor, rf.currentTerm, rf.commitIndex)

   rf.mu.Unlock()
}
```

no unlock() before return!



static analysis uses "patterns" to fine bugs

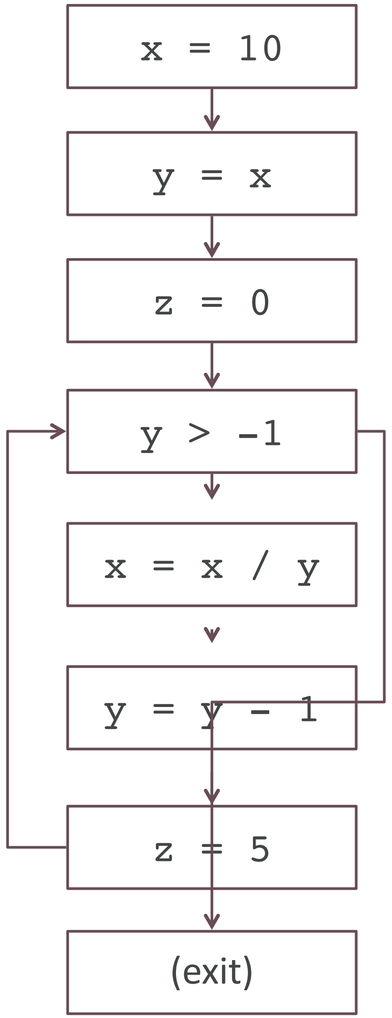
ANOTHER EXAMPLE

```
x = 10;
y = x;
z = 0;
while (y > -1) {
  x = x / y;
  y = y - 1;
  z = 5;
}
```

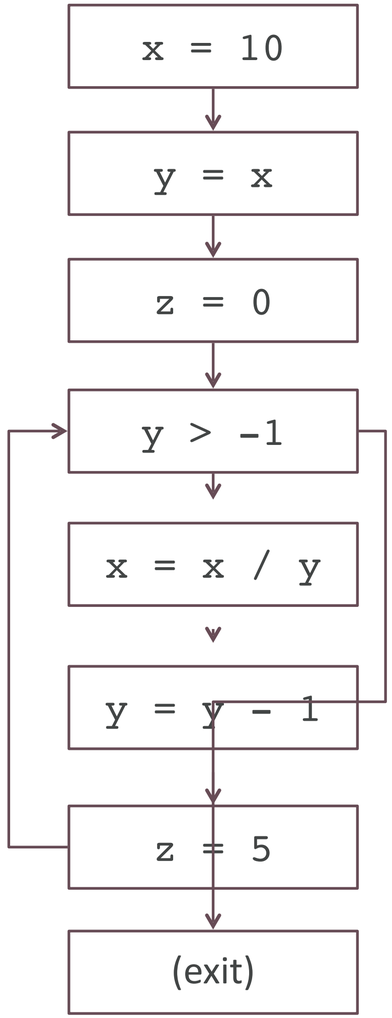
can x be zero?

ANOTHER EXAMPLE

```
x = 10;
y = x;
z = 0;
while (y > -1) {
  x = x / y;
  y = y - 1;
  z = 5;
}
```



ANOTHER EXAMPLE



x:NZ

x:NZ, y:NZ

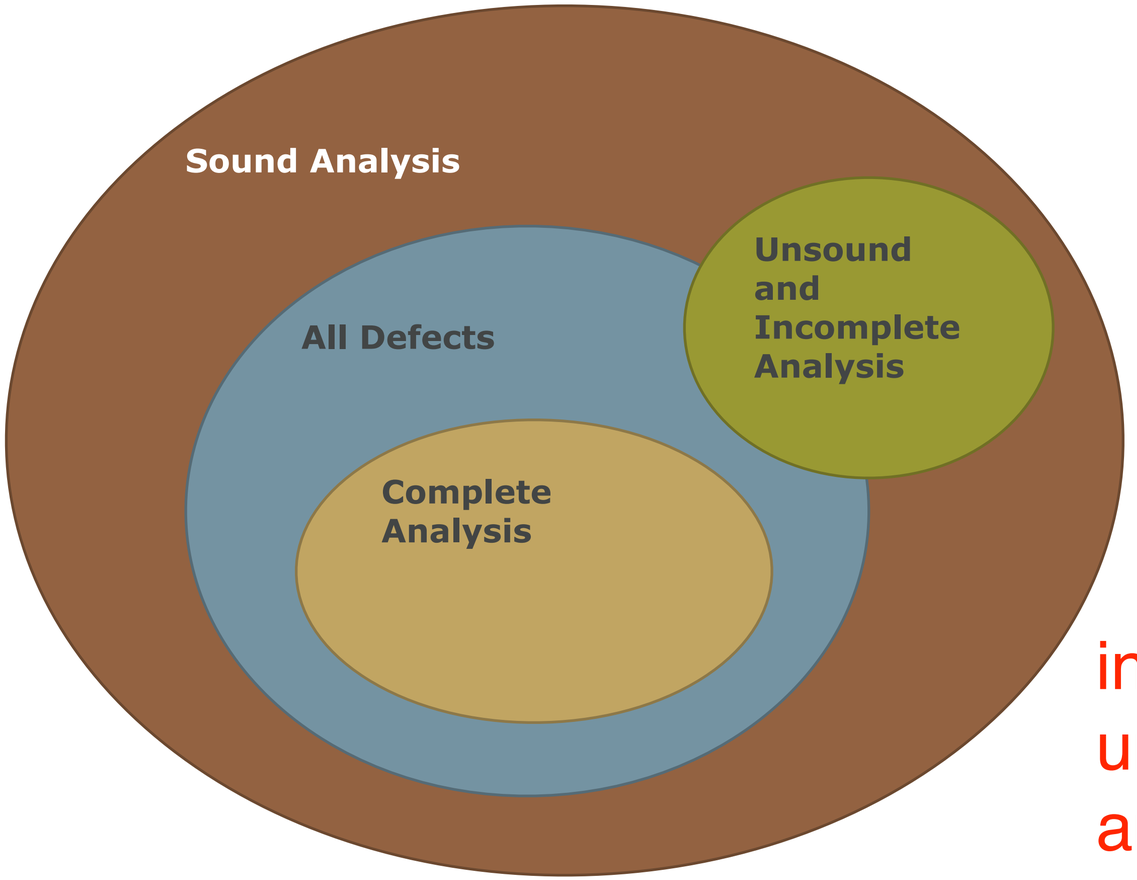
x:NZ, y:NZ, z:Z	x:NZ, y:MZ, z:MZ	x:NZ, y:MZ, z:MZ
x:NZ, y:NZ, z:Z	x:NZ, y:MZ, z:MZ	x:NZ, y:MZ, z:MZ
x:NZ, y:NZ, z:Z	x:NZ, y:MZ, z:MZ	x:NZ, y:MZ, z:MZ
x:NZ, y:MZ, z:Z	x:NZ, y:MZ, z:MZ	x:NZ, y:MZ, z:MZ
x:NZ, y:MZ, z:NZ	x:NZ, y:MZ, z:NZ	x:NZ, y:MZ, z:NZ

SOUNDNESS, COMPLETENESS

Property	Definition
Soundness	“Sound for reporting correctness” Analysis says no bugs → No bugs or equivalently There is a bug → Analysis finds a bug
Completeness	“Complete for reporting correctness” No bugs → Analysis says no bugs

Recall: $A \rightarrow B$ is equivalent to $(\neg B) \rightarrow (\neg A)$

SOUNDNESS, COMPLETENESS



in practice, often settle for unsound and incomplete analysis

STATIC ANALYSIS

- Strength
  - scalability
  - fault localization
- Weakness
  - require specific bug pattern (false negative)
  - lack runtime information (false positive)

Model checking

TESTING IS USEFUL, HOWEVER..

— “Testing can only show the presence of errors, not their absence.”

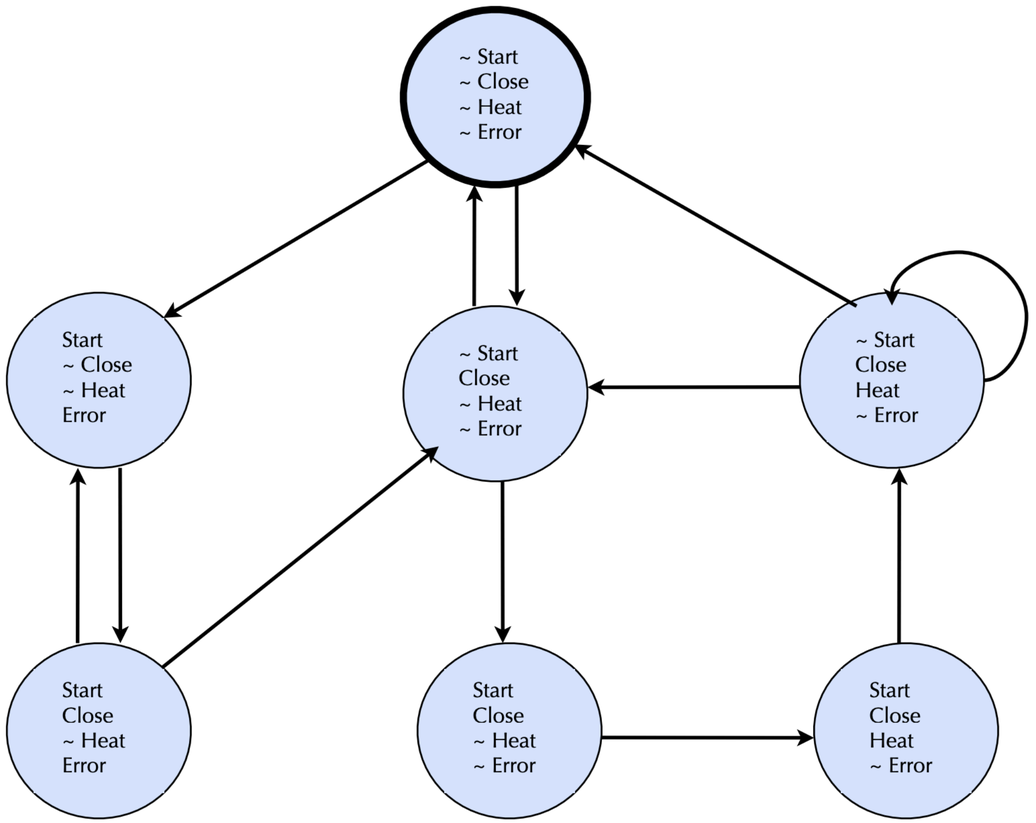
Edsger Dijkstra

1930–2002

MOTIVATION EXAMPLE

- Many techniques focus on checking **implementation**, not **design**
- What if the system design is incorrect?
- Example: Microwave oven
  - Start: “start” button pressed
  - Close: is door closed?
  - Heat: microwave active
  - Error: error state
- Safety property: the oven doesn’t heat up until the door is closed
  - (¬Heat) U Close

MOTIVATION EXAMPLE



DEMO: CHECK CHANG'S MICROWAVE OVEN WITH TLA+

MODEL CHECKING PROBLEM

- Given state transition graph M
- Let ϕ be specification (a temporal logic formula)
- Find all states s of M such that for all execution sequences x starting from s, $x,0 \models \phi$

MODEL CHECKING STEPS

- 1. Write a specification of the system in a formal specification language (think math).

- 2. Specify correctness properties as invariants on states or behaviors.

- 3. Use a model checker to exhaustively check that every state/behavior of the system, within a bounded range of configurations, satisfies your invariants.
  - e.g., TLA+ (by Leslie Lamport)

MODEL CHECKING RAFT

https://github.com/Vanlightly/raft-tlaplus/blob/main/specifications/standard-raft/Raft.tla

raft-tlaplus / specifications / standard-raft / **Raft.tla**

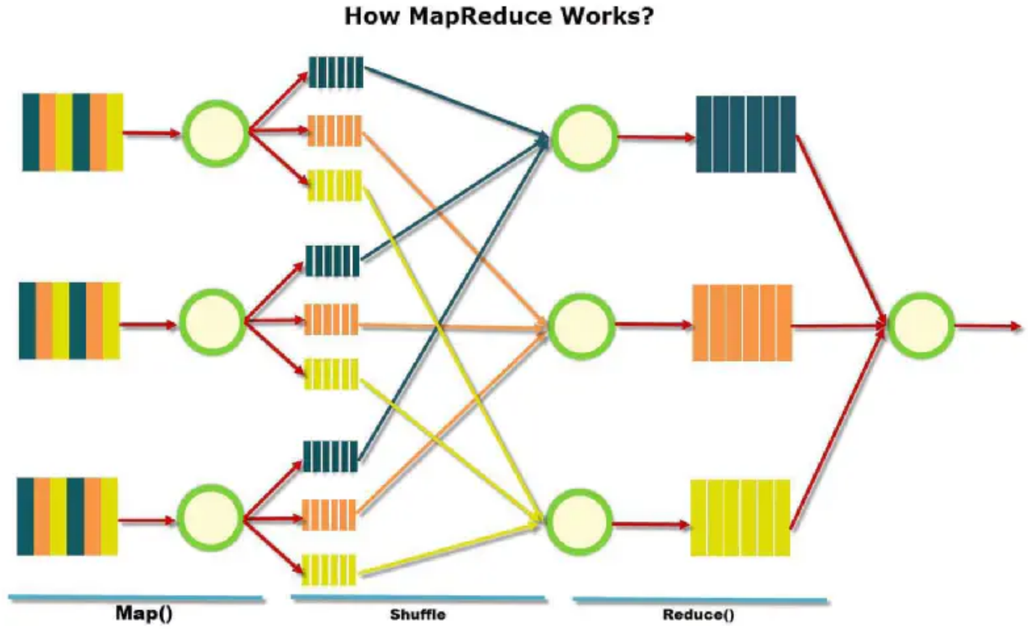
Code | Blame | 653 lines (582 loc) · 26.3 KB

```
257         /\ UNCHANGED <<acked, leaderVars, logVars, restartCtr>>
258
259     \* ACTION: AppendEntries ----------------------------------------
260     \* Leader i sends j an AppendEntries request containing up to 1 entry.
261     \* While implementations may want to send more than 1 at a time, this spec uses
262     \* just 1 because it minimizes atomic regions without loss of generality.
263     AppendEntries(i, j) ==
264         /\ i /= j
265         /\ state[i] = Leader
266         /\ pendingResponse[i][j] = FALSE \* not already waiting for a response
267         /\ LET prevLogIndex == nextIndex[i][j] - 1
268                prevLogTerm == IF prevLogIndex > 0 THEN
269                                   log[i][prevLogIndex].term
270                               ELSE
271                                   0
272                \* Send up to 1 entry, constrained by the end of the log.
273                lastEntry == Min({Len(log[i]), nextIndex[i][j]})
274                entries == SubSeq(log[i], nextIndex[i][j], lastEntry)
275            IN
276               /\ pendingResponse' = [pendingResponse EXCEPT ![i][j] = TRUE]
277               /\ Send([mtype          |-> AppendEntriesRequest,
278                        mterm          |-> currentTerm[i],
279                        mprevLogIndex  |-> prevLogIndex,
280                        mprevLogTerm   |-> prevLogTerm,
281                        mentries       |-> entries,
282                        mcommitIndex   |-> Min({commitIndex[i], lastEntry}),
283                        msource        |-> i,
284                        mdest          |-> j])
285         /\ UNCHANGED <<serverVars, candidateVars, nextIndex, matchIndex, logVars, auxVars>>
```

Concluding remarks

IT HAS BEEN A LONG JOURNEY..

WE BUILT TWO CLOUD SYSTEMS..



MapReduce

Raft Protocol Summary

Followers

- Respond to RPCs from candidates and leaders.
- Convert to candidate if election timeout elapses without either:
  - Receiving valid AppendEntries RPC, or
  - Granting vote to candidate

Candidates

- Increment currentTerm, vote for self
- Reset election timeout
- Send RequestVote RPCs to all other servers, wait for either:
  - Votes received from majority of servers: become leader
  - AppendEntries RPC received from new leader: step down
  - Election timeout elapses without election resolution: increment term, start new election
  - Discover higher term: step down

Leaders

- Initialize nextIndex for each to last log index + 1
- Send initial empty AppendEntries RPCs (heartbeat) to each follower; repeat during idle periods to prevent election timeouts
- Accept commands from clients, append new entries to local log
- Whenever last log index ≥ nextIndex for a follower, send AppendEntries RPC with log entries starting at nextIndex, update nextIndex if successful
- If AppendEntries fails because of log inconsistency, decrement nextIndex and retry
- Mark log entries committed if stored on a majority of servers and at least one entry from current term is stored on a majority of servers
- Step down if currentTerm changes

Persistent State

Each server persists the following to stable storage synchronously before responding to RPCs:

currentTerm	latest term server has seen (initialized to 0 on first boot)
votedFor	candidateId that received vote in current term (or null if none)
log[]	log entries

Log Entry

term	term when entry was received by leader
index	position of entry in the log
command	command for state machine

RequestVote RPC

Invoked by candidates to gather votes.

Arguments:

candidateId	candidate requesting vote
term	candidate's term
lastLogIndex	index of candidate's last log entry
lastLogTerm	term of candidate's last log entry

Results:

term	currentTerm, for candidate to update itself
voteGranted	true means candidate received vote

Implementation:

1. If term > currentTerm, currentTerm ← term (step down if leader or candidate)
2. If term == currentTerm, votedFor is null or candidateId, and candidate's log is at least as complete as local log, grant vote and reset election timeout

AppendEntries RPC

Invoked by leader to replicate log entries and discover inconsistencies; also used as heartbeat .

Arguments:

term	leader's term
leaderId	so follower can redirect clients
prevLogIndex	index of log entry immediately preceding new ones
prevLogTerm	term of prevLogIndex entry
entries[]	log entries to store (empty for heartbeat)
commitIndex	last entry known to be committed

Results:

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

Implementation:

1. Return if term < currentTerm
2. If term > currentTerm, currentTerm ← term
3. If candidate or leader, step down
4. Reset election timeout
5. Return failure if log doesn't contain an entry at prevLogIndex whose term matches prevLogTerm
6. If existing entries conflict with new entries, delete all existing entries starting with first conflicting entry
7. Append any new entries not already in the log
8. Advance state machine with newly committed entries

Raft

PLAYED WITH COMMERCIAL CLOUD SYSTEMS..

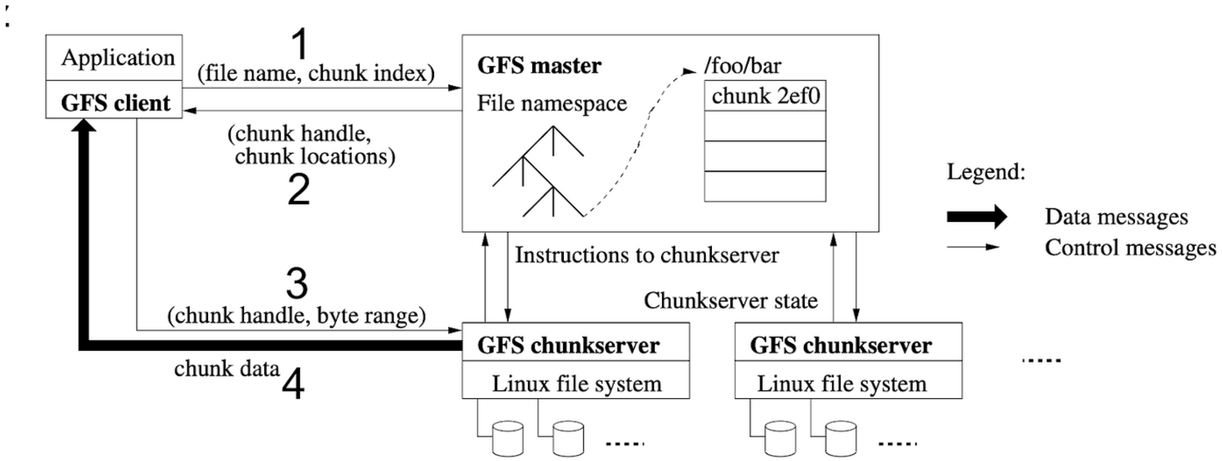
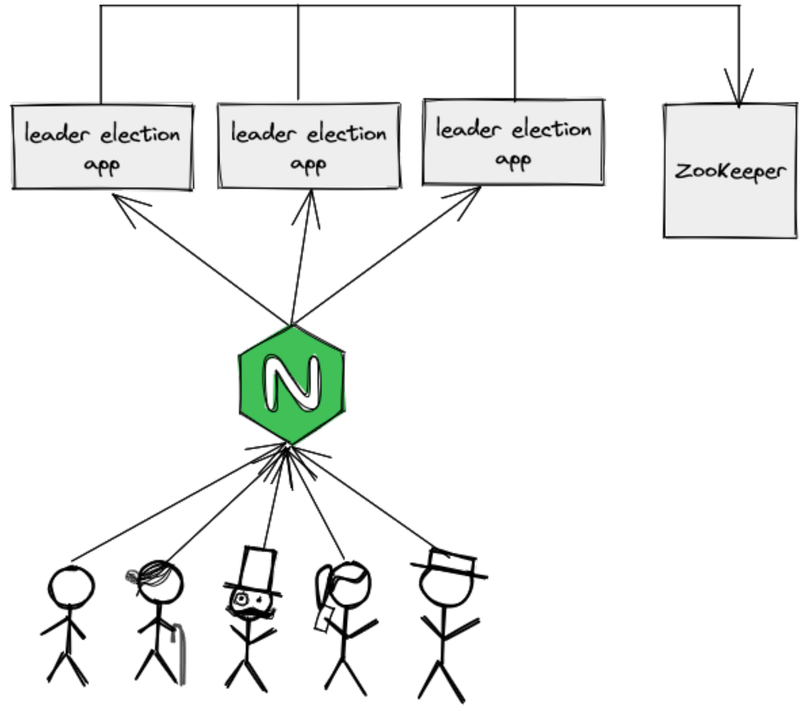


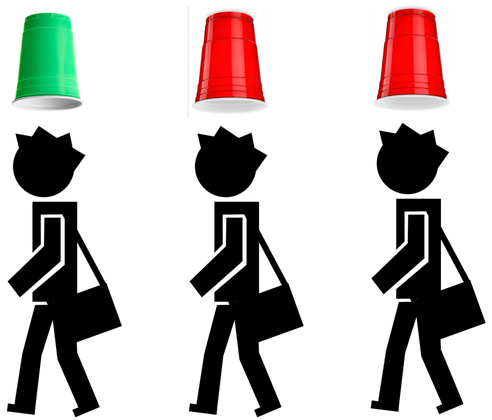
Figure 1: GFS Architecture

Google File System

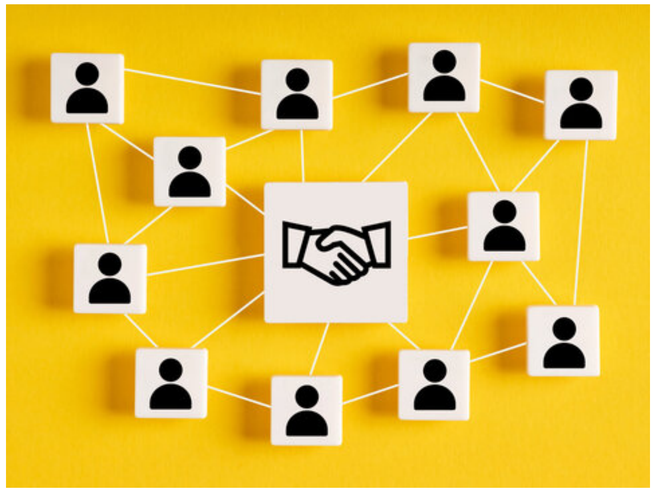


ZooKeeper (Lab Day I, II, III)

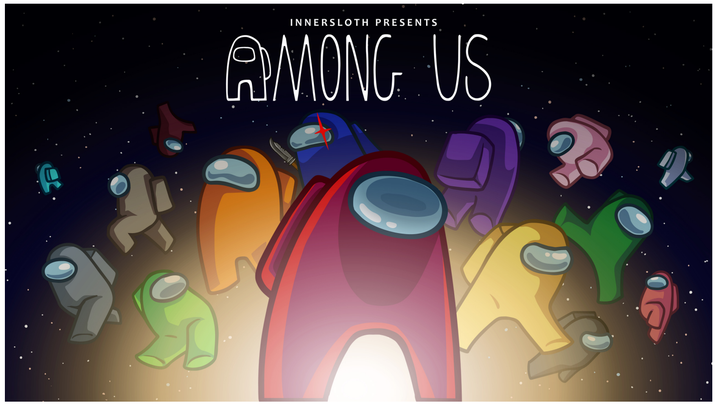
GAMES..



Green cup, Red cup

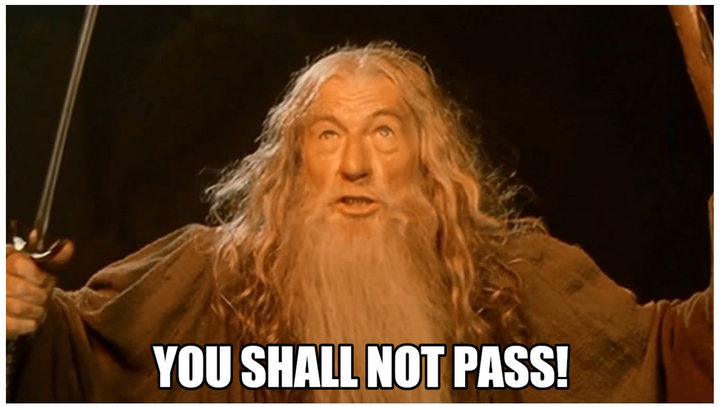


Consensus



Consensus
(w/ malicious peers)

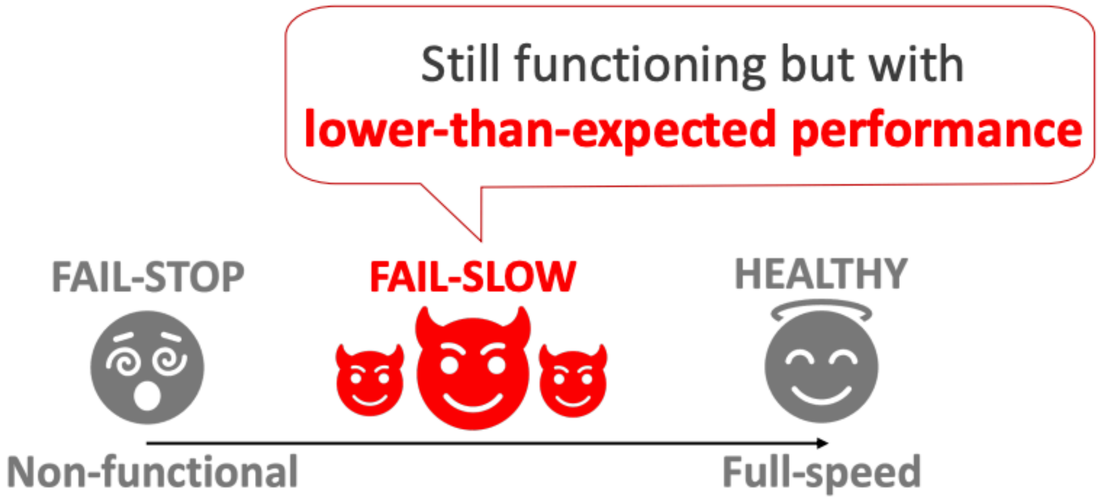
2PC failures w/ Donut



Gandalf

DISCUSSION WITH CLOUD EXPERTS..

- **Failures in The Wild**
  - **Fail-Slow**
  - Fail-Stop
  - Metastable
  - …



What if I'd like to learn more

FUTURE STUDY

- 1. Online resources
  - cloud/distributed system course, e.g., MIT 6.824
  - follow up latest progress on top system conferences, e.g., SOSP/OSDI

6.5840 Schedule: Spring 2024

E25-111, TR1-2:30

Here is the tentative schedule of lectures and due dates. The lecture notes and paper questions for future dates are copies from previous years, and may change. Lectures are in E25-111, Tues/Thurs 1:00 to 2:30.

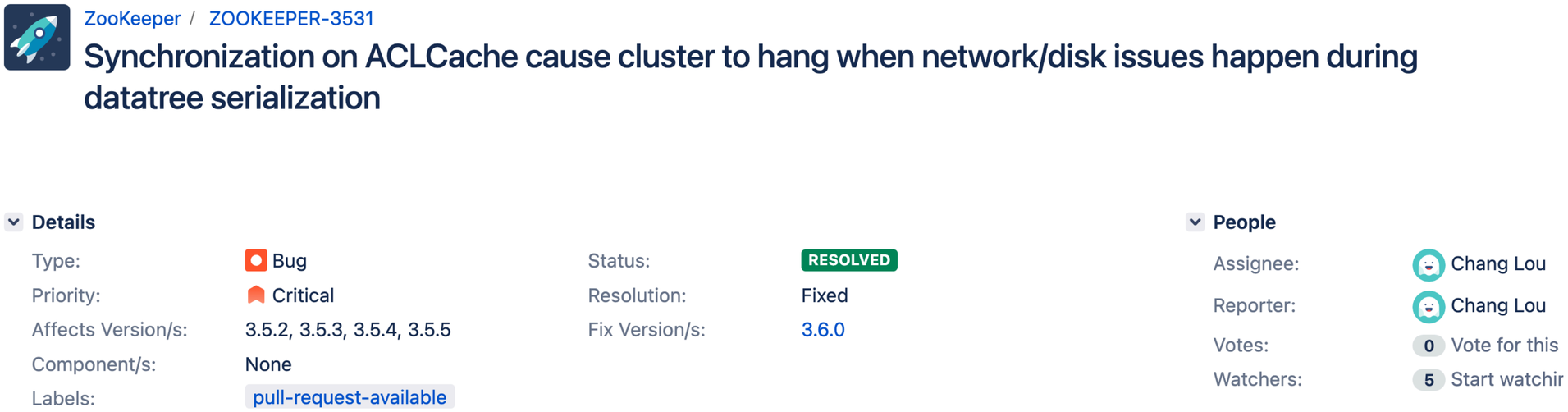
Monday	Tuesday	Wednesday	Thursday	Friday
feb 5 *First day of classes*	feb 6 **LEC 1 (rtm):** Introduction, video **Preparation:** Read MapReduce (2004) **Assigned:** Lab 1: MapReduce	feb 7	feb 8 **LEC 2 (rtm):** RPC and Threads, crawler.go, kv.go, vote examples, video **Preparation:** Do Online Go tutorial (FAQ) (Question)	feb 9
feb 12	feb 13 **LEC 3 (snowstorm):** None **Assigned:** Lab 2: Key/Value server	feb 14	feb 15 **LEC 4 (rtm):** Consistency and Linearizability **Preparation:** Linearizability Testing (FAQ) (Question)	feb 16 **DUE:** Lab 1. All labs are due at 11:59pm.
feb 19 President's day	feb 20 **Assigned:** Lab 3: Raft *Monday schedule*	feb 21	feb 22 **LEC 5 (guest lecture):** (Russ Cox of Google/Go) Go patterns **Preparation:** Read The Go Programming Language and Environment (FAQ) (Question)	feb 23 **DUE:** Lab 2

FUTURE STUDY

— 2. Contribute to open-source cloud software

  — for example, download and play with Kubernetes today

  — even submitting a small PR is a big achievement and a good start!



ZooKeeper / ZOOKEEPER-3531

Synchronization on ACLCache cause cluster to hang when network/disk issues happen during datatree serialization

Details

Type:	Bug	Status:	RESOLVED
Priority:	Critical	Resolution:	Fixed
Affects Version/s:	3.5.2, 3.5.3, 3.5.4, 3.5.5	Fix Version/s:	3.6.0
Component/s:	None		
Labels:	pull-request-available		

People

Assignee:	Chang Lou
Reporter:	Chang Lou
Votes:	0 Vote for this
Watchers:	5 Start watchir

FUTURE STUDY

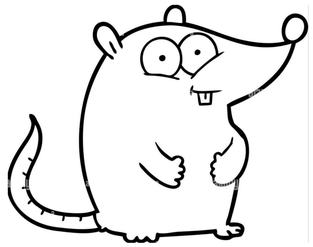
- 3. Continue exploring cloud in our grad-level course!
  - Focus on Reliability
  - Paper reading + Project
  - **No exam :)**
  - Undergraduate students are welcomed

CS6501: Cloud System Reliability

Fall 2024, UVA CS

.. A FEW MORE WORDS

— This is a class "in progress".



you

me

— Thank you so much for supporting and improving this course!

Share your thoughts for future students on Student Experiences of Teaching!

https://go.blueja.io/tCkk_V3wNk-JEn23zAT3sA

Extra credits for Completed SET!

TAKEAWAYS

- Next class: **Final Review**
- **Deadline** of Lab2C: 12/8, Monday

ACKNOWLEDGEMENT

THIS COURSE IS DEVELOPED HEAVILY BASED ON COURSE MATERIALS SHARED BY PROF. INDRANIL GUPTA, PROF. ROBERT MORRIS, PROF. MICHAEL FREEDMAN, PROF. KYLE JAMIESON, PROF. WYATT LLOYD AND PROF. ROXANA GEAMBASU. MANY APPRECIATIONS FOR GENEROUSLY SHARING THEIR MATERIALS AND TEACHING INSIGHTS.

SOME CONTENTS ARE FROM OSDI'21 PREVIEW SESSION VIDEO MADE BY CHENGCHENG WAN AND LEFAN ZHANG