

CSE P 552

Distributed Systems

Replication

~ fault tolerance

- Lab 1: network faults ✓

~ machine faults?

- all critical tasks need to be able to
run on > 1 machine

- What if all machines crash?

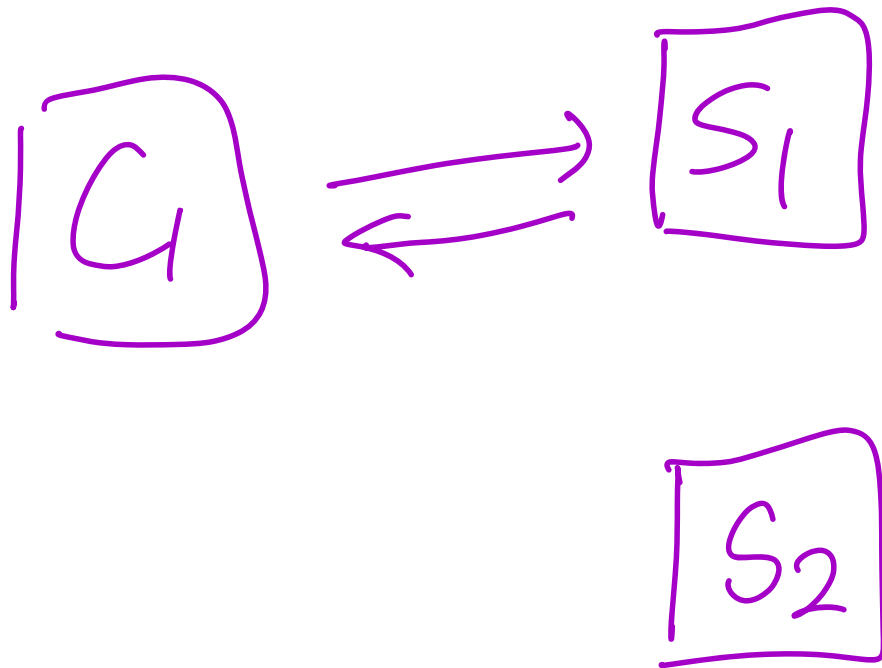
- progress is not possible (liveness violation)

- maintain safety properties

Safety vs Liveness

- safety = bad things don't happen
- liveness = good things eventually happen
 - availability = % of time when I can get work done

Tolerating one machine crash



Replication = do same thing on ≥ 1 server



Strategy: State machine replication (Application)



- deterministic
- no side effects

Determinism = running same seq of
cmds from same starting
state produces same results
and same ending state

Bad idea: primary decide when to
failover to backup

Failure detection is impossible

- indistinguishable whether a lack of communication is because of network or machine faults

Success detection is possible

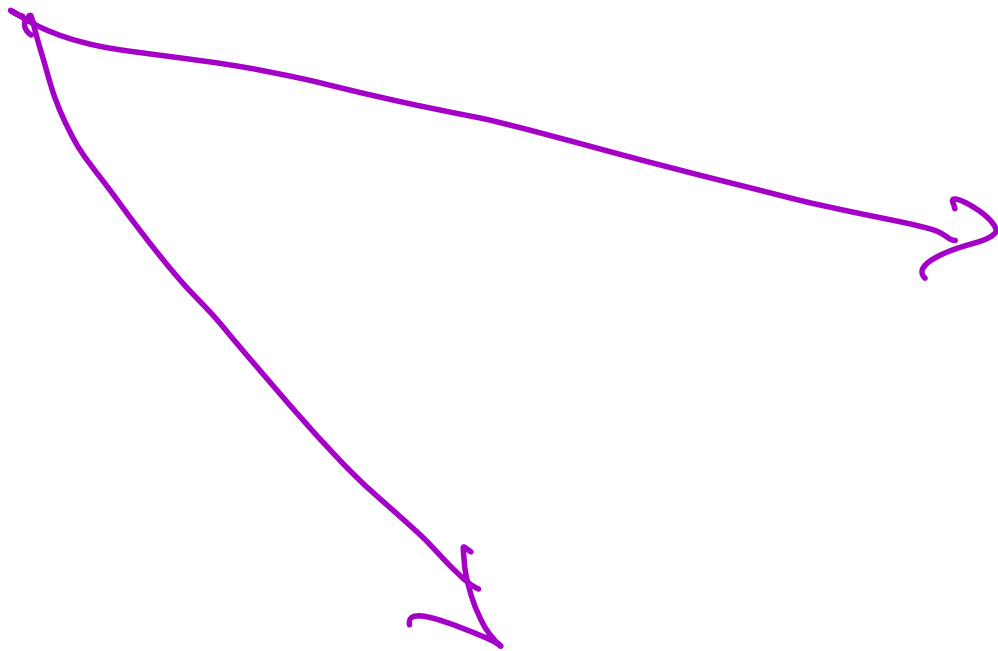
- just send a message saying you succeeded

Approximating Failure Detection — who detects?

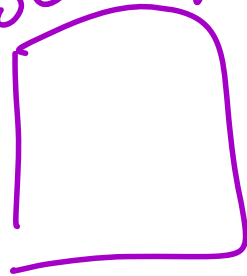
C

P

B

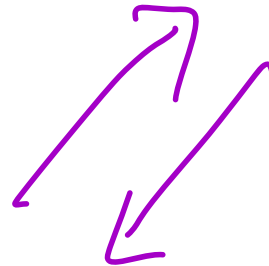


View
server



Failover:

- view server leader to switch to backup
- client contacts backup, who operates independently



Lab 1 Debrief: how did it go?

- challenges? insights?

- Java.

- * Part 3 harder than expected

- Shim b/w Server and app vs server and network

- figuring out spec (esp. part 3)

- debugging search tests

- heisenbugs

Advice on Debugging

of run-test.py -s FINEr
> log.txt

	Search	run
safety	① viz	② log
liveness	④	③

Invariant property: always true

- true in initial state of the system
- true in all reachable states

Example in Lab 1

- if request from C w/ seq num 42
has been sent, server's cached seq num
for C is ≥ 41

Stable Properties (Monotonic properties)

- ~ might be false, but on an execution if it becomes true, it stays true for the rest of that execution
- node N sent message M
- ~ labl: server's cached seqnum only goes up:
"seqnum for C is $\geq K$ " is stable (for all K)

How to prove a property unstable?

give an example of $\text{true} \rightarrow \text{false}$

Linearizability

(consistency model: answers the question of what results are "correct" for a bunch of concurrent client requests)

- results have to agree w/ a sequential execution

- if one request r_1 was sent after the response to another req r_2 , then r_1 must come after r_2 in the seq order

linearizability: results $\rightarrow$ bool

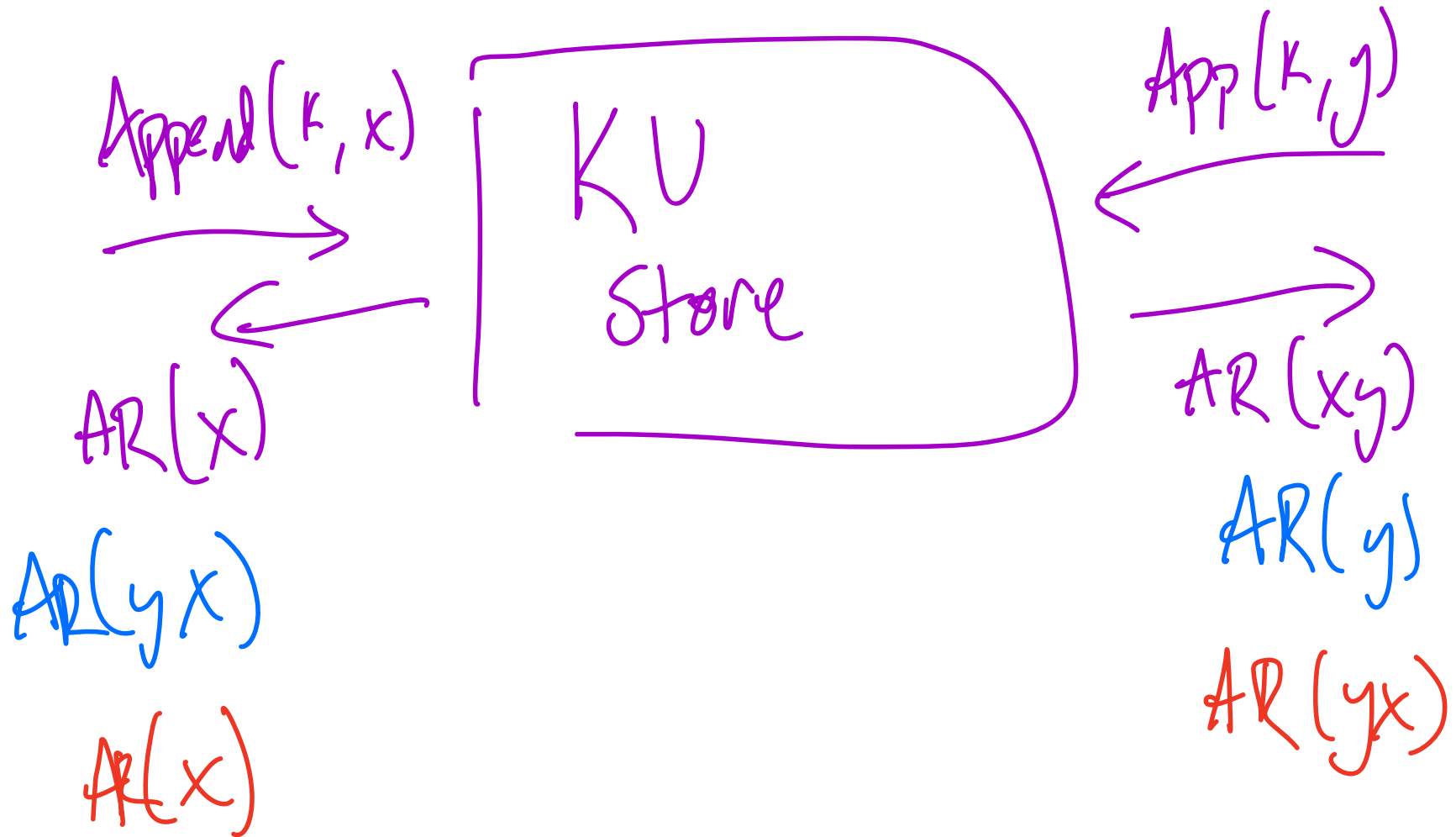
list of interleaved req/resp pairs

$\{A_{C_1}(x), A_{C_2}(y), AR_{C_2}(y), AR_{C_1}(yx)\}$

Sequential order = each request is immmed.
followed by its resp

$\rightarrow [(A_{C_2}(y), AR_{C_2}(y)), (A_{C_1}(x), AR_{C_1}(yx))]$

§3



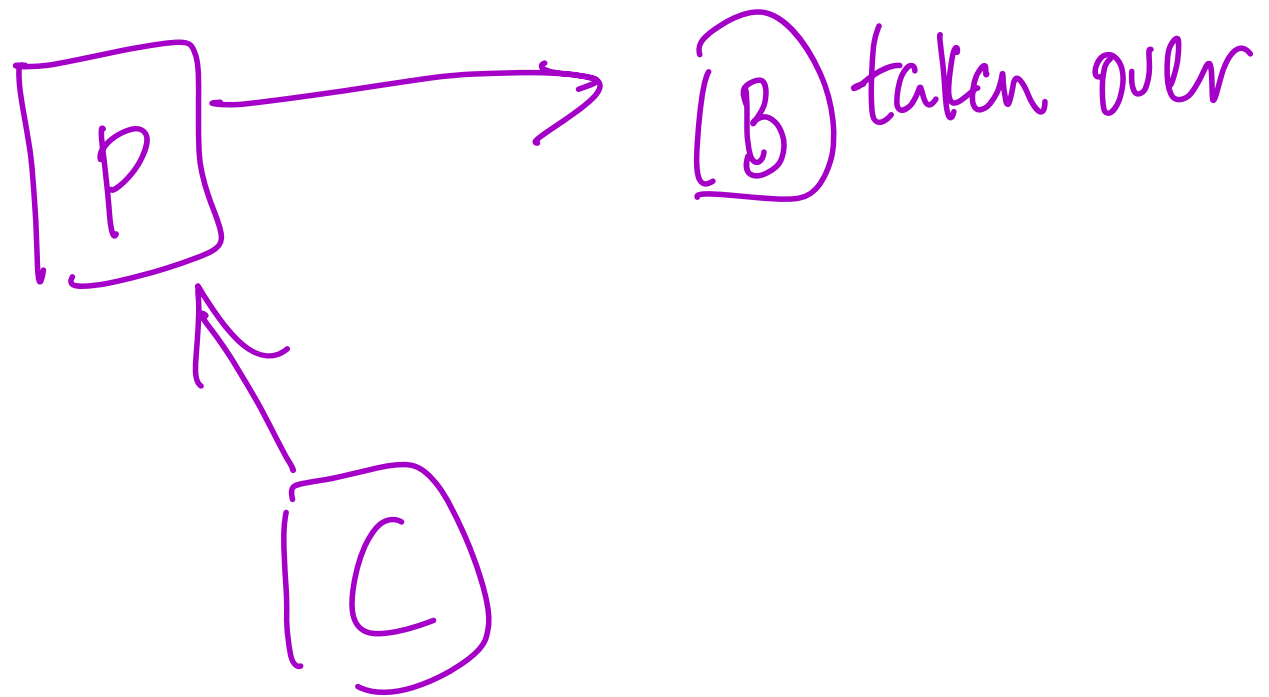
$$[A_{C_1}(x), AR_{C_1}(yx), A_{C_2}(y), AR_{C_2}(y)]$$

$$\rightarrow [A_{C_2}(y), AR_{C_2}(y), A_{C_1}(x), AR_{C_1}(yx)]$$

not linearizable

- how do clients know when to switch to the backup?
 - ask the view server
 - when? when they think P is failed
- how does the backup know when to take over?
 - ask the view server

(Vs) backup should take over



- Number our VILL

view:

- view number

- primary address

- backup address

- null if primary is unilateral

S_3

VS

View 1: $P = S_1$
 $B = S_2$

view 2: $P = S_2$
 $B = \text{null}$

S_1

S_2

view 3: $P = S_2$
 $B = S_3$

C

State Transfer: new P sends entire DB to the new B

view $P = S_1$
 $B = S_2$

S_1 fails

can VS make S_3 new P ?

Ping protocol

