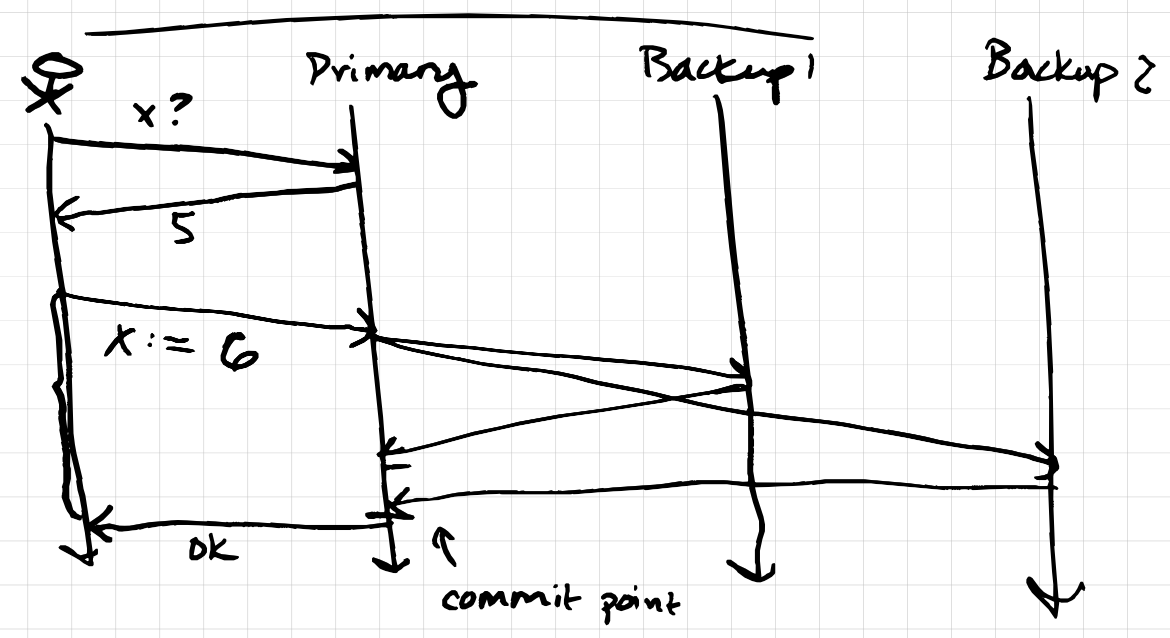
CSE138 Lecture 11

- primary-backup (recap)
- chain replication
  - ↳ relatedly: latency & throughput
- a little about implementing causally consistent replication
- dealing with node failure (crash faults)
- intro to consensus
- FLP result.

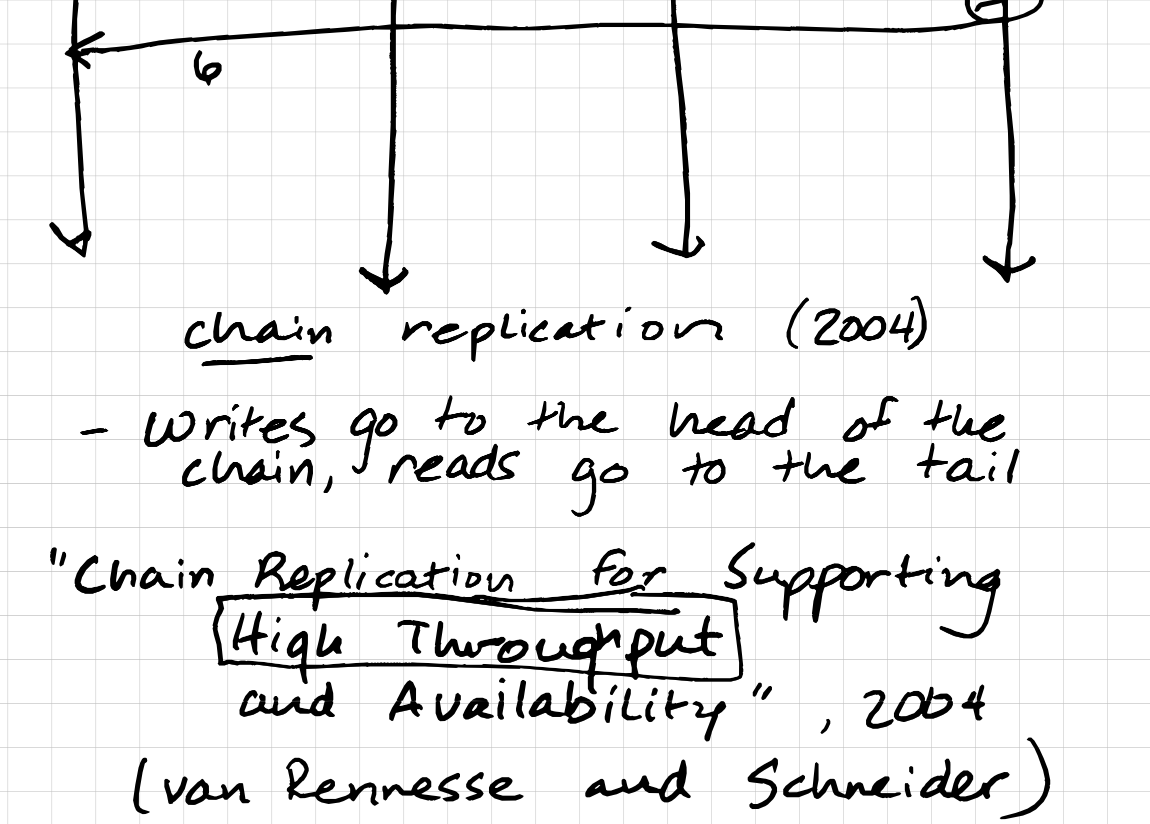


primary-backup replication

pro/con: primary's in charge!

- :) easy to think about total order (primary decides)
- :(primary becomes a bottleneck because it has to handle all requests
- :(what if the primary crashes?

A classic way to improve on primary-backup:



<u>chain</u> replication (2004)

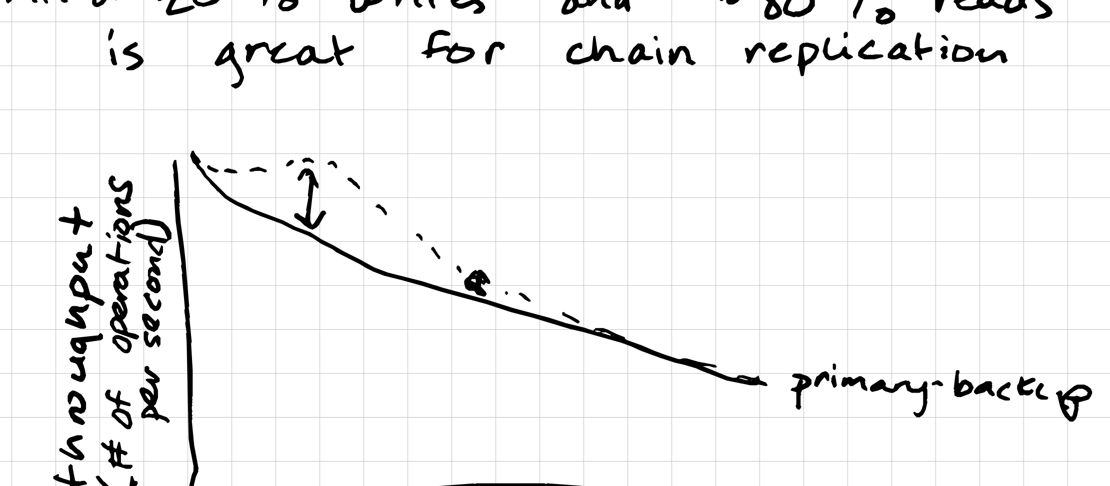
- writes go to the head of the chain, reads go to the tail

"Chain Replication for Supporting High Throughput and Availability", 2004 (van Rennesse and Schneider)

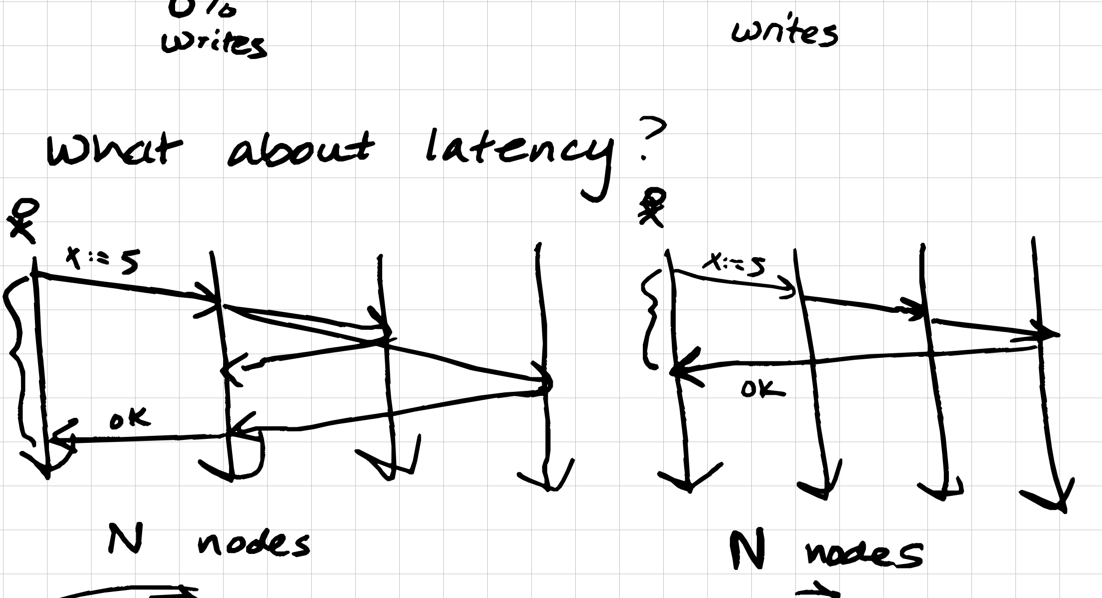
<u>throughput</u>: number of actions (e.g. writes/reads) per unit of time

<u>latency</u>: amount of time per action (← writes/reads)

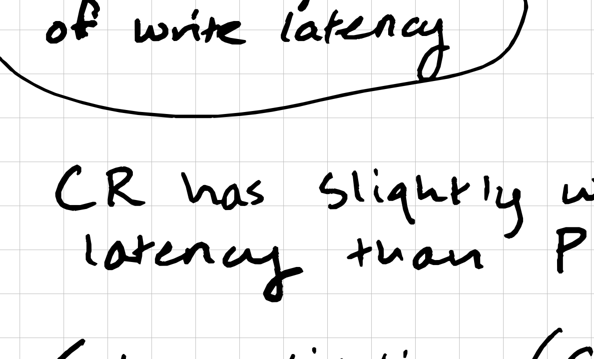
★ writes – 3-4x more time to process than reads

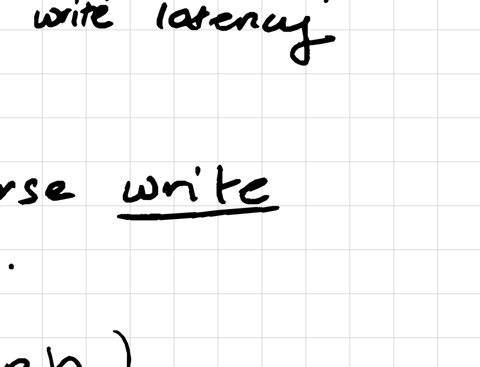


a mix of ~20% writes and ~80% reads is great for chain replication



What about latency?





N nodes ⇒ 4 messages of write latency

N nodes ⇒ N+1 messages of write latency

CR has slightly worse <u>write</u> latency than PB.

Splay replication (Ceph)
↳ something to look up on your own if you want!

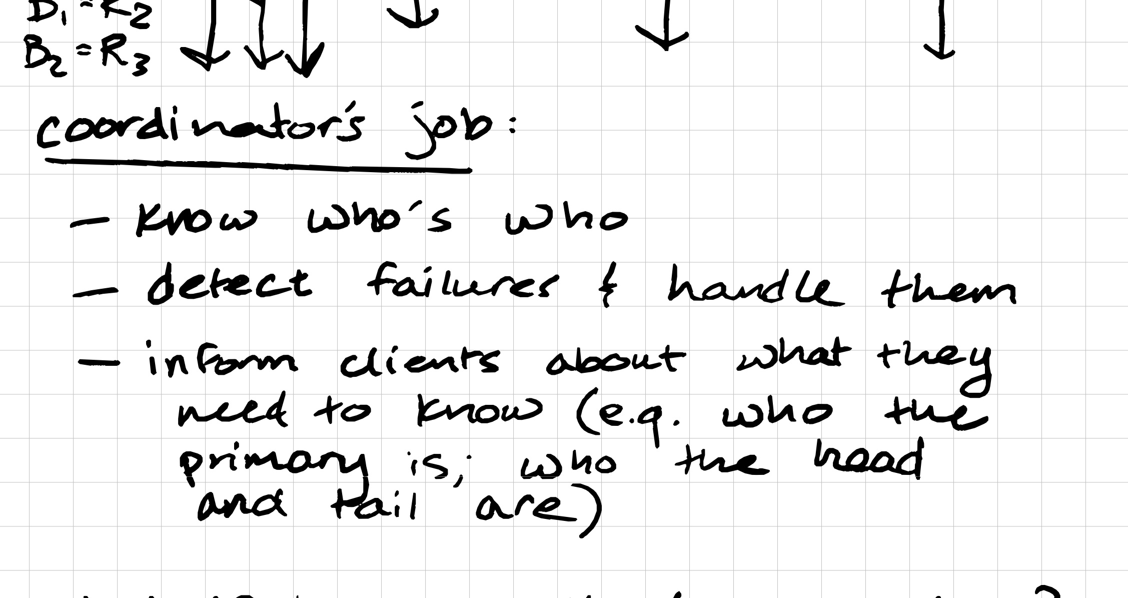
important caveat about both PB and CR:

Nodes must agree on who's playing which role!

Clients also need accurate info on who's playing which role, so they know who to talk to.

Since nodes may <u>change</u> roles over time, we need a way to keep everyone in sync about who's who.

ultimately, both PB and CR will rely on some sort of "coordinator" whose job it is to keep track of who's who.



<u>coordinator's job</u>:

- know who's who
- detect failures & handle them
- inform clients about what they need to know (e.g. who the primary is; who the head and tail are)

what if the coordinator crashes? oh no!

we need multiple coordinator processes.

But how do we keep <u>those</u> in sync?

consensus.

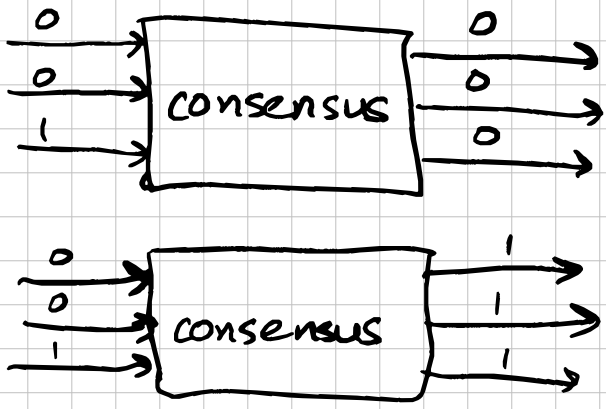
When do you **need** consensus?

You have a distributed system (with multiple processes), and...

- you need to make sure they all deliver messages in the same order. (Totally-ordered delivery)
- you need them all to know which other processes are up and running. (Group membership problem, Failure detection problem)
- you want to elect one to be in charge, and let everyone know who got elected (Leader election problem)
- you want them to take turns accessing some resource that requires exclusive access (Distributed mutual exclusion problem)
- they're all participating in a transaction, and you want them to all agree whether to commit or abort. (Distributed transaction commit problem)

The consensus problem:

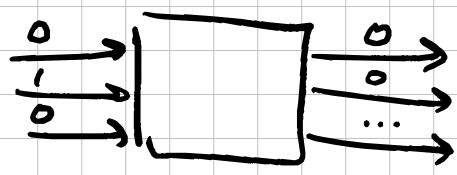
Get everyone to agree on 0 or 1.



Properties that consensus protocols try to satisfy:

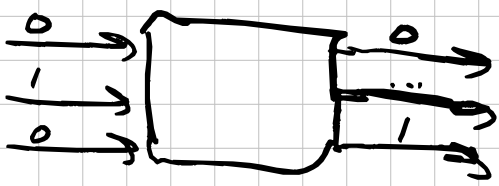
- Termination: each correct process eventually decides on a value.

violation of termination:



- Agreement: every process that decides on a value decides on the same value.

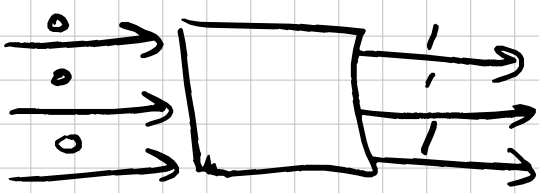
violation of agreement:



- Validity: every process that decides on a value chooses one of the input values.

(the thing we pick must have been proposed!)

violation of validity:



No consensus protocol can actually satisfy all 3 of termination, agreement, and validity, in the asynchronous network model and the crash fault model.

FLP result (1983)

Michael Fischer,
Nancy Lynch,
Michael Patterson.