CSCI 1380 : Day 16 : Lazy Replication

last Week

Raft (passive replication)

This Week

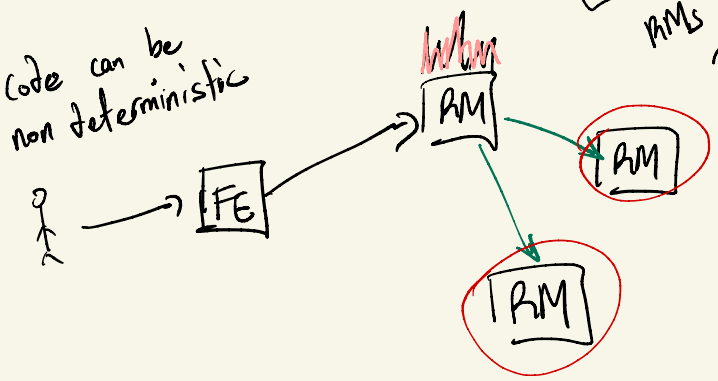
Lazy Replication (Gossip, Vector clocks, motivating)

Practical Consensus (Practical active/passive replication)

Passive

leader/follower RMs have different roles

code can be non deterministic

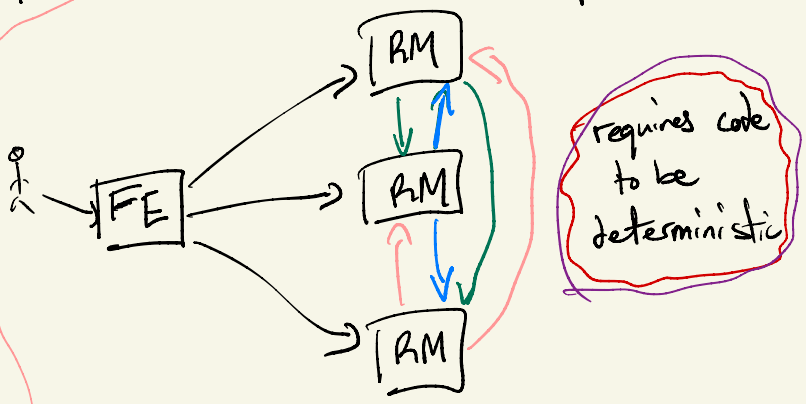


key issues: under failure it takes time to find new leaders & commit

Passive ⟹ data centers/clusters

network latency ⟹ 4ms

Active all RMs are equal



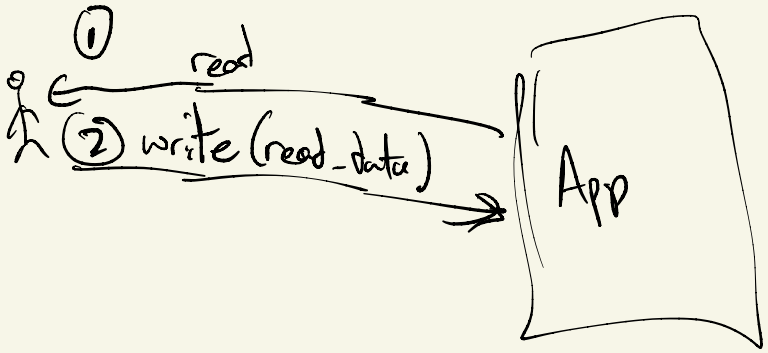
key issues: lots of messages & high overhead

Wide area network (Internet)

network latency ⟹ 40–100ms

Geo distributed database/storage will use active replication

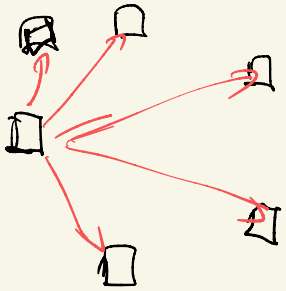
Application (Lazy replication / Causal ordering)



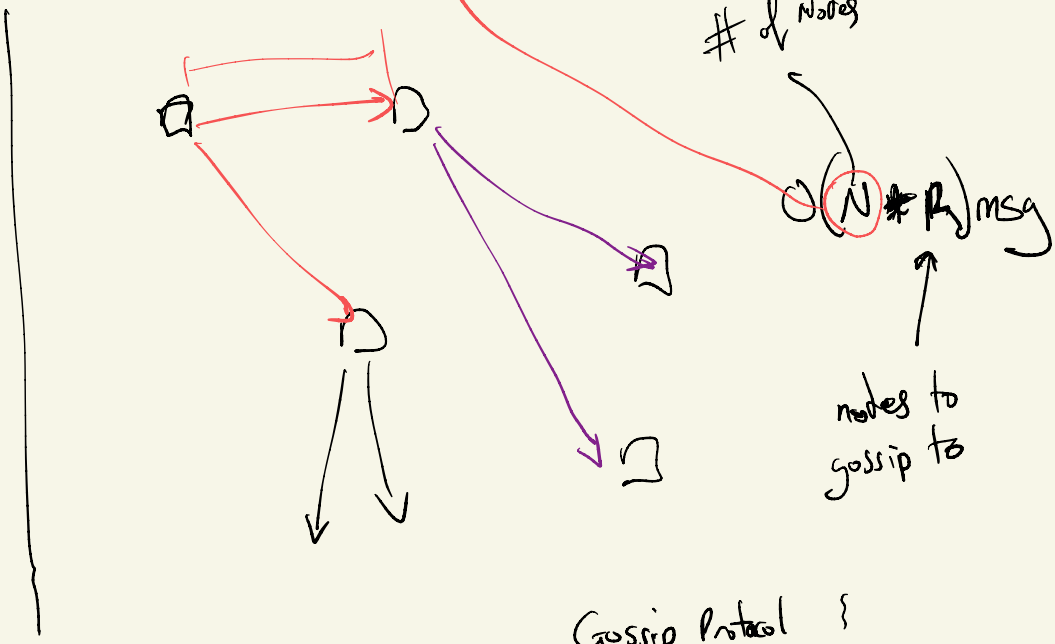
1. email/messages
2. forums
3. social network

Causal relationship: you can only respond to what you read or what you received

Gossip



$O(N * N-1)$ msg

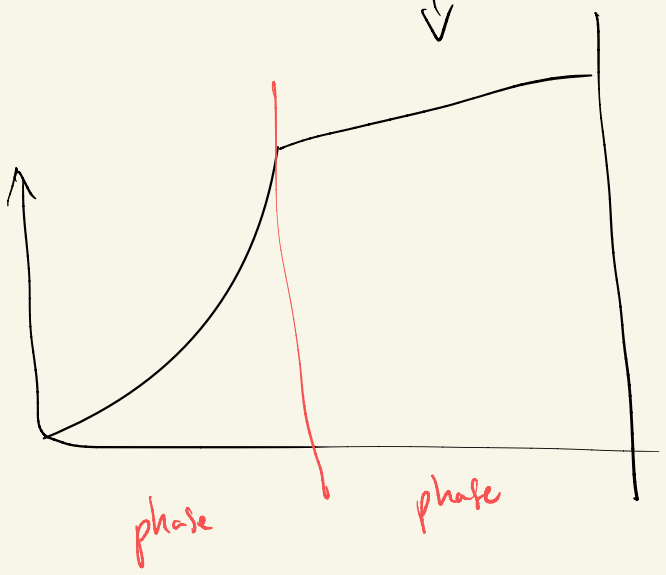


Gossip Protocol {
pick R random nodes
Send updates to R
sleep(t)
}

# of nodes to gossip to

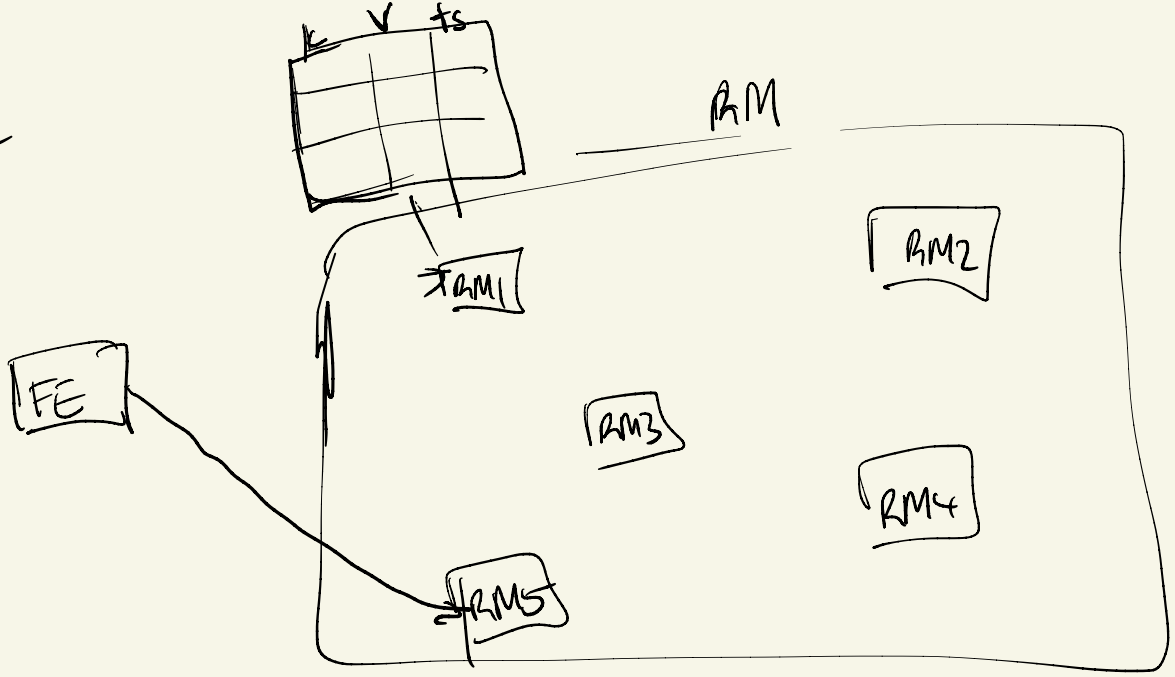
How frequently to Gossip

# of epochs or round = $\frac{\text{phase1}}{\text{exponential}} + \frac{\text{phase2}}{}$



query → get

update → set



① FE picks & interacts with one RMs

② before you update; you must first read
↳ you get a VC from the RM

③ when you update then you provide VC with the key & value

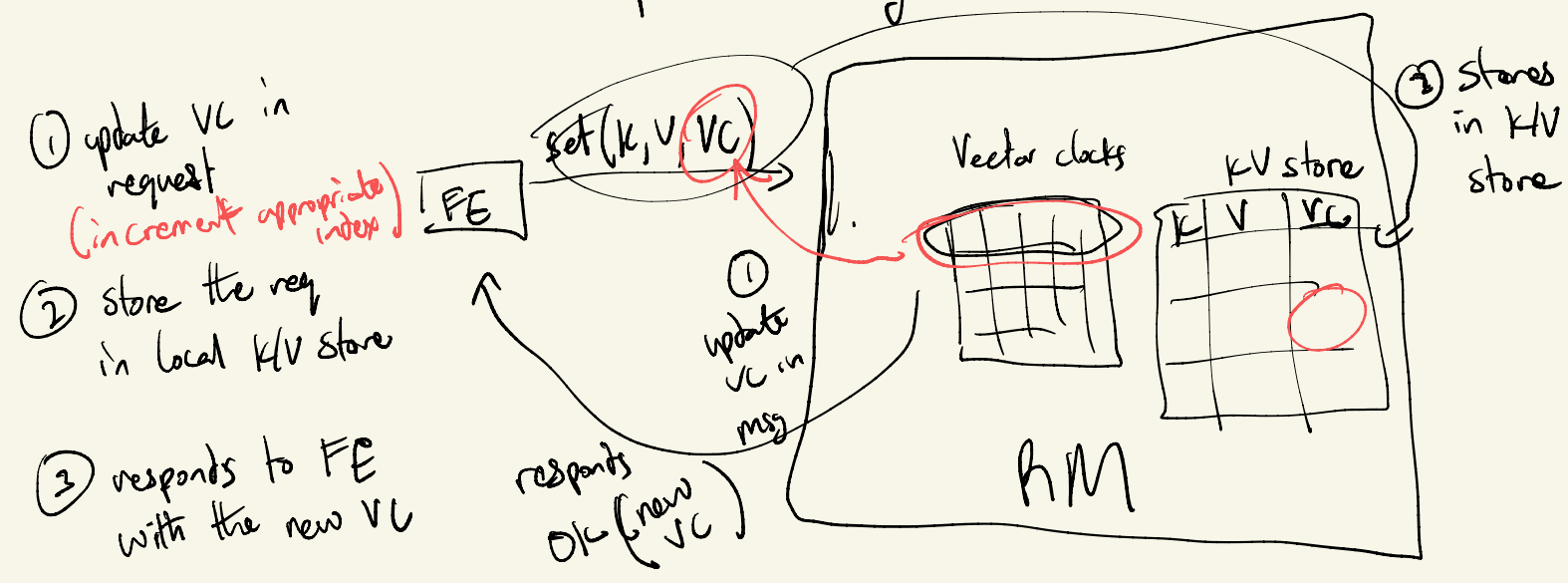
email / forums / messages

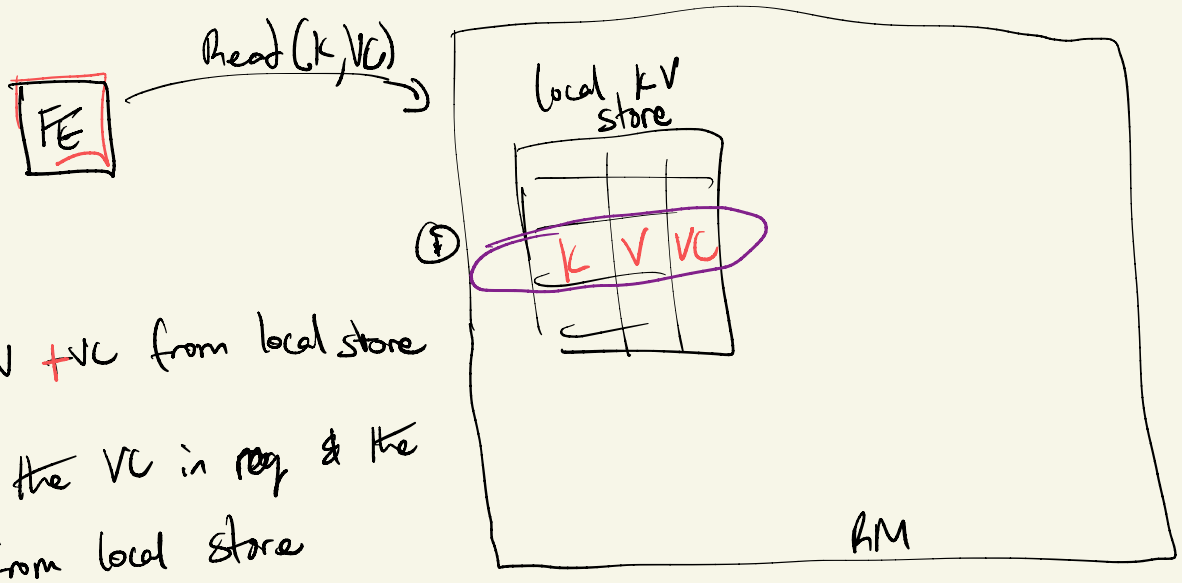
① you need to first / receive

② before you can respond

A read (or query) in Lazy R. returns Value & V.C.

Update in Lazy R.





① get K,V +VC from local store

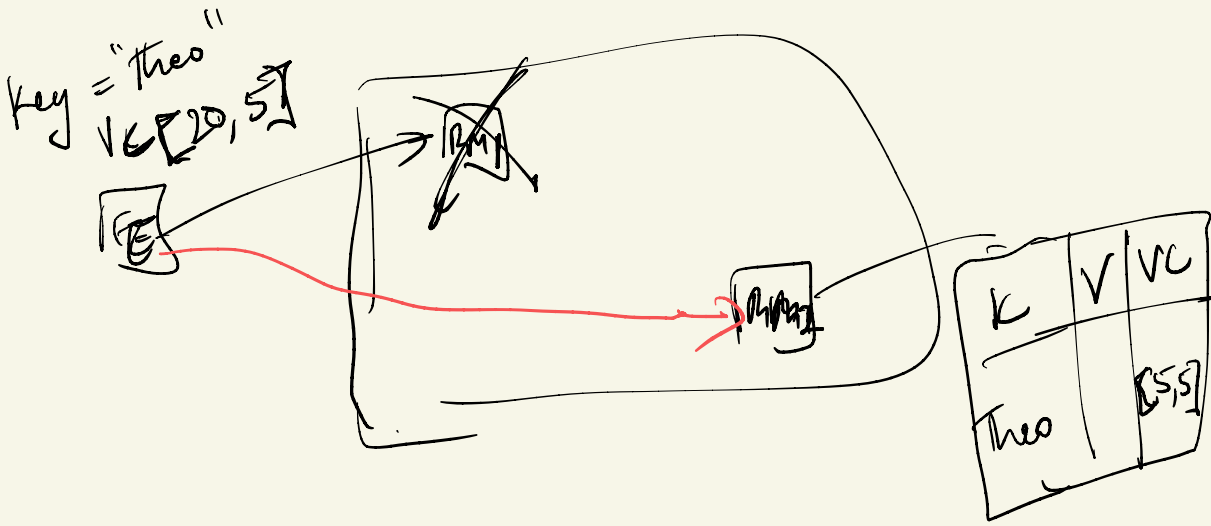
② compare ~~the~~ VC in req & the VC from local store

if req.VC > local.VC

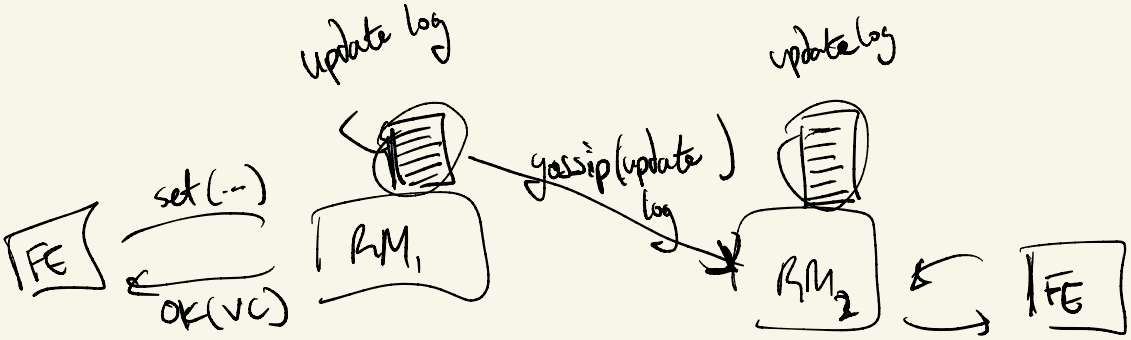
wait for gossip msgs

else return value +new VC

As long as one RM is alive ⇒ you can update always



Update replication with Gossip



Gossip msg in Lazy replication include "update logs"

RM3

Parallel but not identical to AppendEntries in RAFT

Because

1. Gossip / randomness factor
2. updates return immediately with "OK"

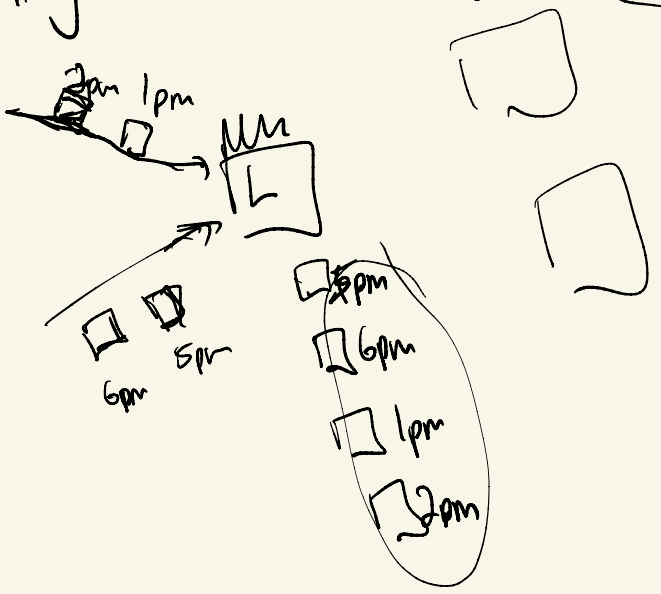
Commit procedure to decide when to respond back to say things are OK

Data loss can happen

Ordering

Linearizability = Total ordering + real time + FIFO

leader clock



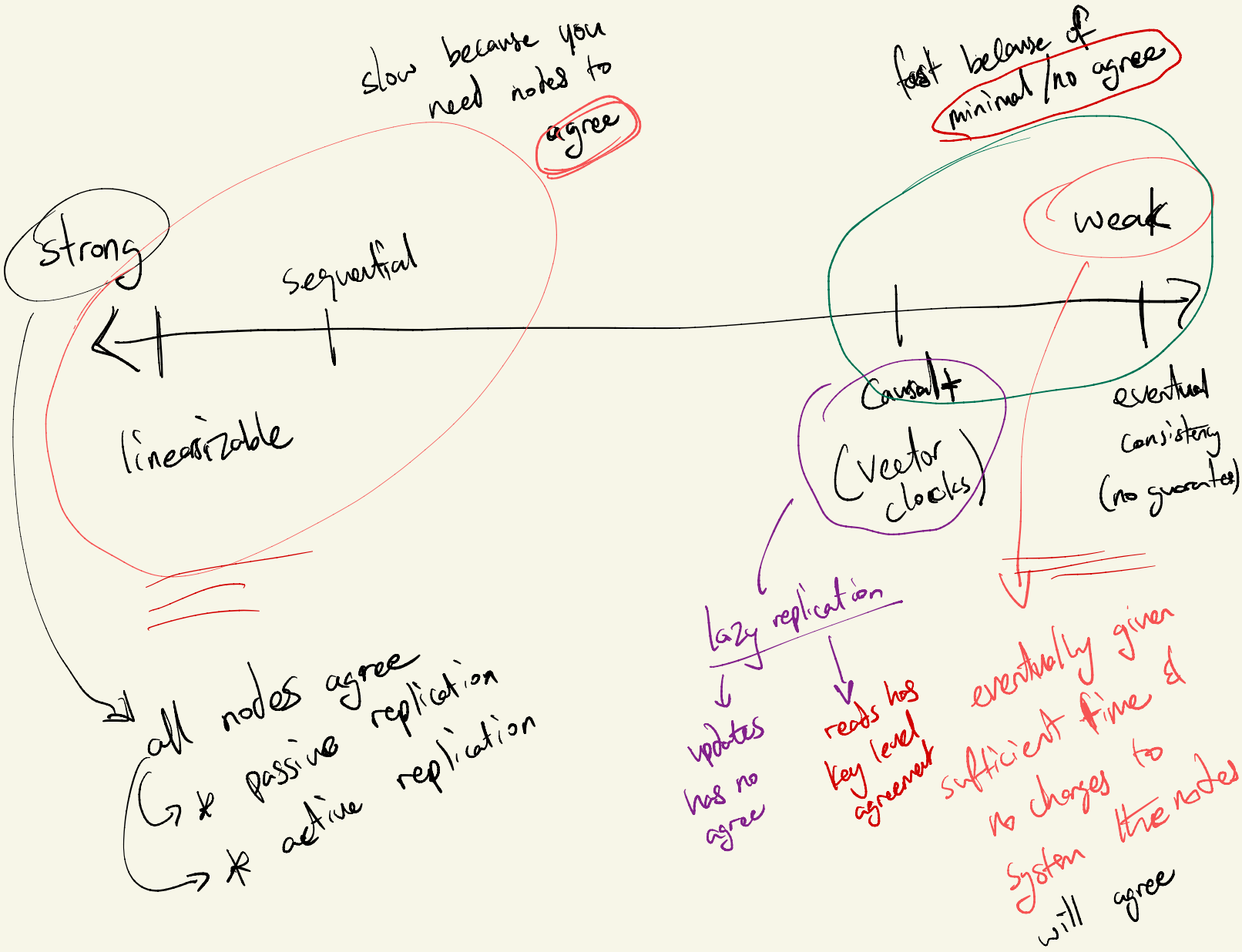
Lazy != linearizable ordering

(1) some updates can get lost from

(2) edgecase => conflicting updates

(3) VC to determine ordering
different orderings can happen to
unrelated events

Sequential = Total + FIFO


slow because you need nodes to agree
fast because of minimal/no agree
strong
sequential
linearizable
weak
causal+
(vector clocks)
eventual consistency
(no guarantee)
all nodes agree
* passive replication
* active replication
lazy replication
updates has no agree
reads has key level agreement
eventually given sufficient time & no changes to system the nodes will agree

Summary

Gossip

Lazy Replication

* motivating examples
* query (get) requests (waits until RM's local VC is as up to date as VC in get)
* ~~update~~ (set) requests (returns instantaneously with new VC)
* Propagating ~~update~~ log btw RMs
* Availability Guarantees

Ordering & Consistency