MANAGING CHAIN REPLICATION WITH HUMMING CONSENSUS

Scott Lystig Fritchie, Basho Japan KK

Ricon 2015 San Francisco

2015-11-05 Thursday 5pm US PST

About Me

- Senior software engineer @ Basho Japan KK, Tokyo
  - scott@basho.com, @slfritchie on Twitter
  - Tech lead for Basho's distributed file store "Machi"
- Author of HibariDB (which uses chain replication)
- UNIX sysadmin & software developer since 1986
- Erlang infatuation (infection?) since 1999

Outline

- Problem statement
- What is CR?
- Why use CR?
- Managing CR is a problem?
- Why improve CR?
- An allegory of composing music (in fan-fic style)
- Move from story to code
- Today's code status
- References, credits
- Questions!

Problem Statement

- We wish to make a self-contained manager for Chain Replication metadata (e.g., chain membership, chain order, safe chain state transitions) that supports both strong consistency **and** eventual consistency.
- We solve this problem as distributed musicians might compose music.

Neil Conway
@neil_conway

Following

Chain replication: strange that it is so well-known among academics and yet seemingly obscure to practitioners.

RETWEETS 5 FAVORITES 13

3:08 PM - 21 Oct 2015

Problem Statement Problems

- What is Chain Replication?
- Why use Chain Replication?
- Why is managing Chain Replication a problem?

WHAT IS CHAIN REPLICATION?

Chain Replication Papers

- Van Renesse and Schneider. "Chain Replication for Supporting High Throughput and Availability." USENIX OSDI. Vol. 4. 2004.
- Bickford & Guaspari, "Formalizing Chain Replication", tech report, 2006.
- Bickford, "Verifying Chain Replication using Events", tech report, 2006.
- Terrace and Freedman. "Object Storage on CRAQ: High-Throughput Chain Replication for Read-Mostly Workloads."

Chain Replication Papers

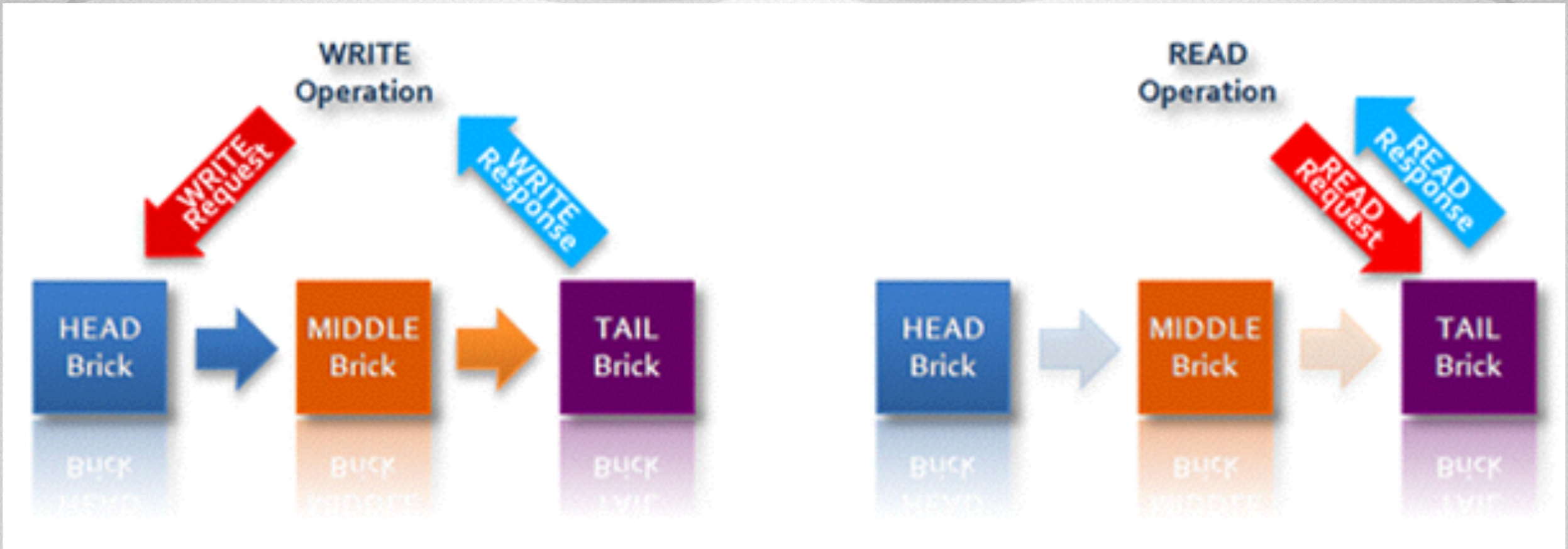
- Van Renesse, Ho, and Schiper. "Byzantine chain replication." Principles of Distributed Systems. Springer Berlin Heidelberg, 2012. 345-359.
- Abu-Libdeh, van Renesse, and Vigfusson. "Leveraging sharding in the design of scalable replication protocols." Proceedings of the 4th annual Symposium on Cloud Computing. ACM, 2013.

Chain Replication Users

- FAWN
- CRAQ
- HibariDB
- Hyperdex
- CORFU & CorfuDB
- ChainReaction

- Synrc App Stack
- Machi
- … perhaps more? …

Chain Replication On One Slide



- Variant of primary/secondary replication: strict chain order!
- Sequential read @ tail. Linearizable read @ all.
  Dirty read @ head or middle.

C.R.: A Paxos Cousin

- "Niobe, Chain replication, and the Google File System [...] While these protocols are seemingly unrelated, the first two can be viewed as Vertical Paxos algorithms."
  - "Vertical Paxos and Primary-Backup Replication", Lamport, Malkhi, Zhou

markcallaghan
@markcallaghan

Foll

@neil_conway write to front, read from the back. The mullet of replication?

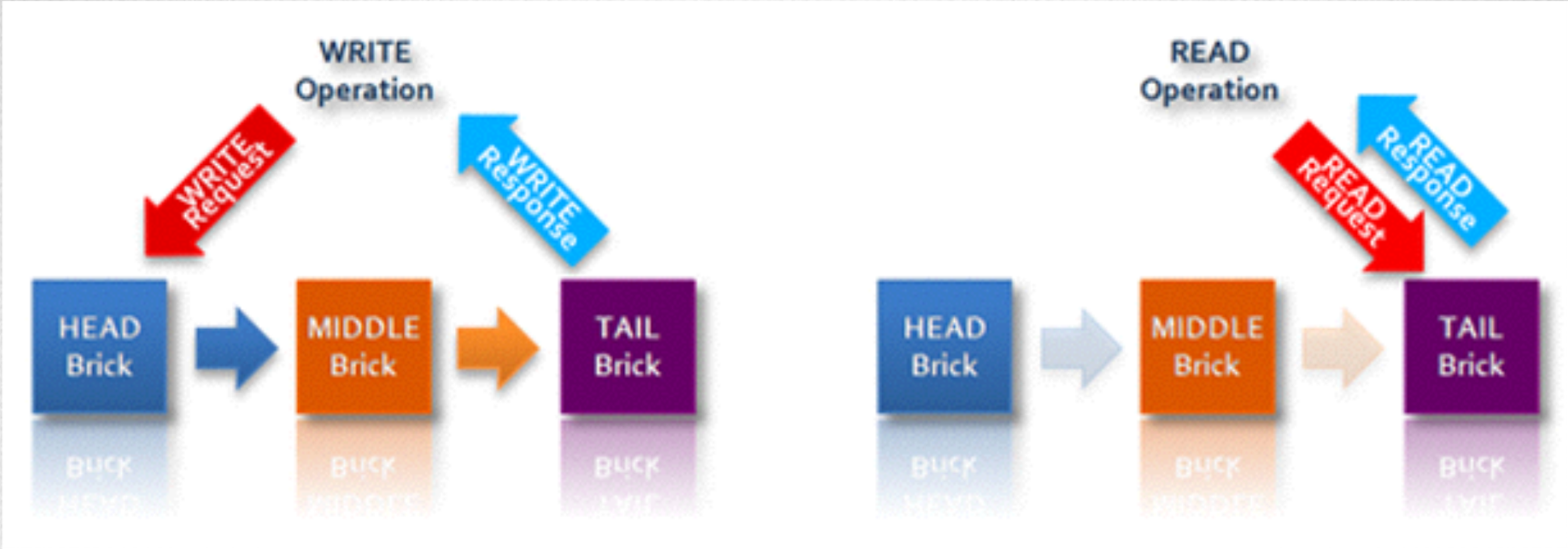
RETWEETS
2

FAVORITE
1

The Other “One Slide”

WHY USE CHAIN REPLICATION?

Cheap! Easy! Free! Kittens!



- “Cheap”: **f+1** replicas to survive **f** failures.
- “Easy”: Strong consistency is a nice side-effect
- “Free": Anti-entropy is an **under-valued side-effect**

Cheap! Easy! Free! Kittens!

WHY IS MANAGING CHAIN REPLICATION A PROBLEM?

Managing Chain Replication

- Screw up chain order -> screw up consistency
- “State of the art” isn't ideal

Review: State Of The Art

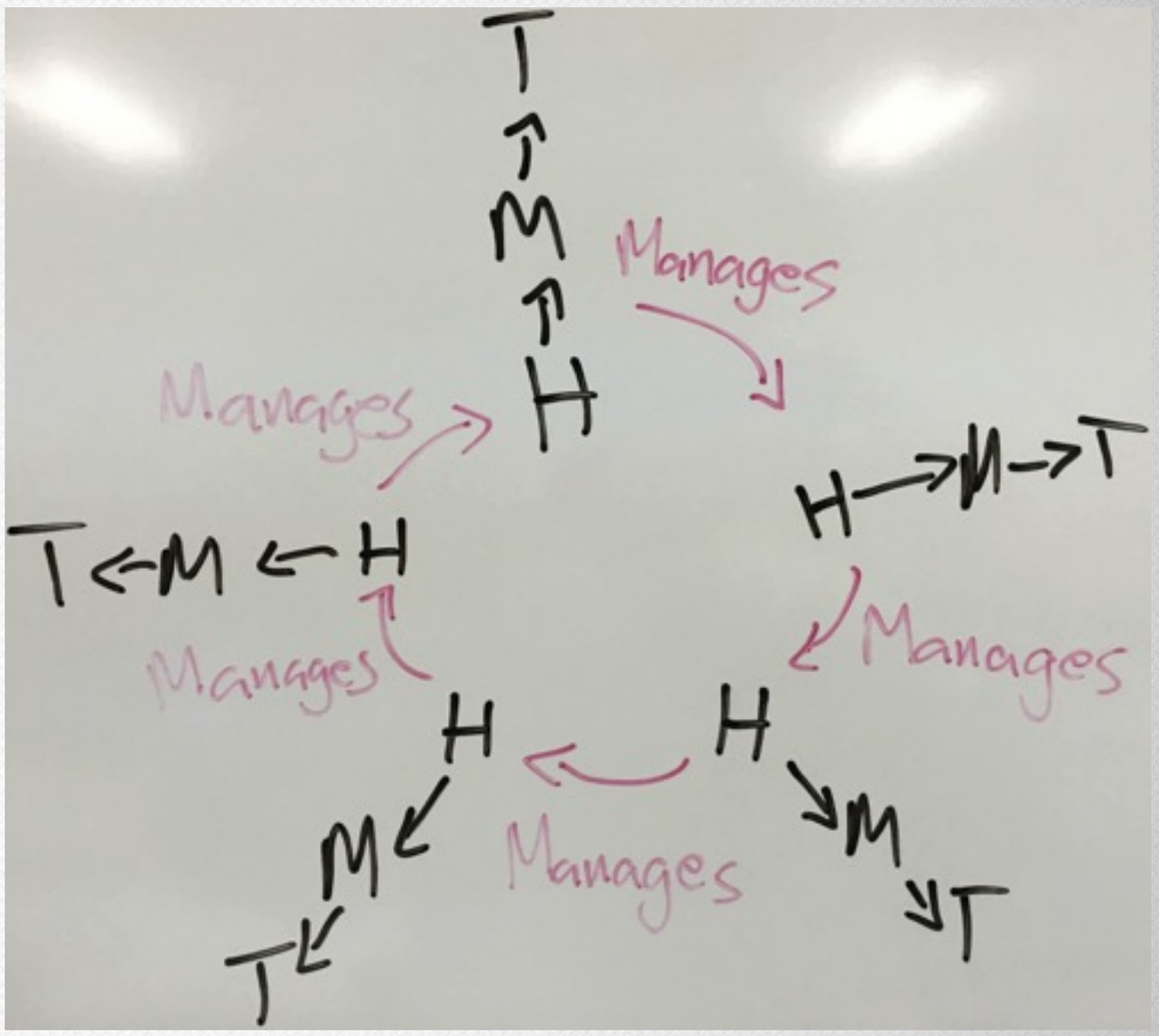
- **The oracle**: exactly one omniscient, infallible agent/program.
  - Definitely bad for always-available/eventual consistency
- **Active-standby oracle**
  - Not so helpful if chain is length > 2
  - Even number -> split brain problem
  - Config & monitoring is pain/nightmare/pure-evil/….

Review: State Of The Art

- Use strong consistency system to create **a distributed oracle**
  - Example tools: Zookeeper, etcd
  - ZK/etcd servers on separate boxes -> more stuff can break
  - Awfully big & complex for Machi's design space

Review: State Of The Art

- Elastic Replication: "elastic band" of chain managers
  - If at least one chain is running, all chains can be bootstrapped
  - Corner case: all fail simultaneously



WHY DO WE WANT TO IMPROVE THE STATE OF THE ART?

Dumb File Service

- Dumb, 26 years ago: NFSv2
- Dumb, today: Machi

```
/* https://tools.ietf.org/html/rfc1094
*/
program NFS_PROGRAM {
  version NFS_VERSION {
      void
      NFSPROC_NULL(void)          = 0;
      attrstat
      NFSPROC_GETATTR(fhandle)    = 1;
      attrstat
      NFSPROC_SETATTR(sattrargs) = 2;
/* .... */
      statfsres
      NFSPROC_STATFS(fhandle)     = 17;
```

町

Machi

“village” or “town”

```
// Protocol Buffers API requests:
//
// echo()            :basic test
// auth()            :start auth handshake
// append_chunk()    :append bytes to file
// read_chunk()      :read bytes from file
// trim_chunk()      :delete bytes in file
// checksum_list():get checksum metadata
// list_files()      :list files
//
// FIN. The end. That's all.
```

How Is This Better Than Hadoop?

- … or NFS or QFS or WTF or SeaweedFS or …

1. First: Replicate bits correctly 100% of the time
   - **Do cool stuff only after replication *works***
2. **Checksum everything**, end to end!
3. Two modes: strong consistency and eventual consistency
   - Eventual consistency: CRDT-like, always-mergeable file operations as a (dumb & robust) service

CONSENSUS AND HUMMING IN THE IETF

RFC 7282

To reinforce that we do not vote, we have also adopted the tradition of “humming”: When, for example, we have face-to-face meetings and the chair of the working group wants to get a “sense of the room”, instead of a show of hands, sometimes the chair will ask for each side to hum on a particular question, either “for” or “against”.

Once Upon A Time, There Were Some Distributed Music Composers

<Fanfic Mode="Lamport" Allusion="The Part-Time Parliament">

There Will Be A Quiz At The End

- How frequently do composers talk directly to each other?

About Our Music Composers

- Everyone follows strict rules for composition
  - Voice leading, chord progression, rhythm, instrumentation…
- Need rough consensus on each measure of music
- All work in the same room ... unless they don't
- Small groups break out to rehearsal rooms. Or at coffee shop.
  - For a few seconds. Or hours. Or years.

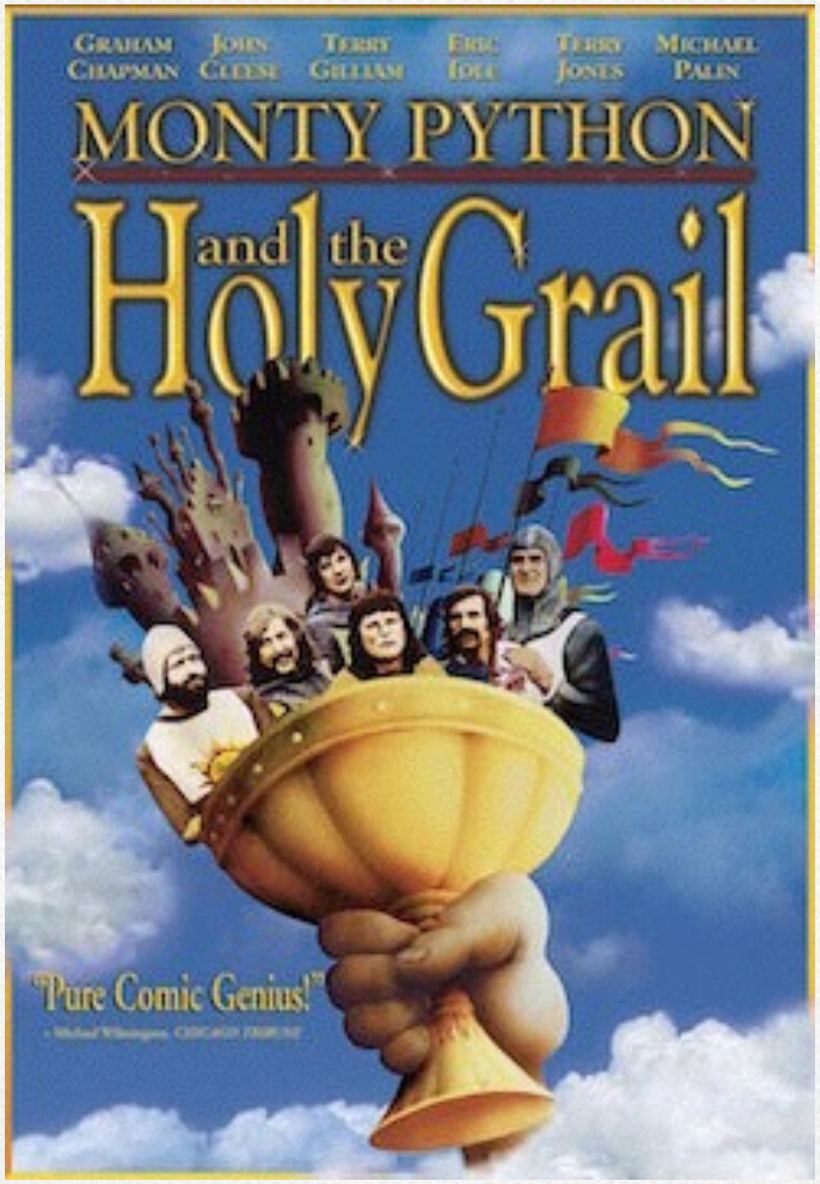
About The Composers' Workflow

- Each measure of a manuscript is numbered
- Music is written only from beginning to end
  - One measure at a time
  - Blank measures will be removed by publisher, no worries
- Each measure is ranked for beauty, lyricism, etc.
- For lyricism, immediate earlier measures are important
  - No mixing Happy Birthday + Thriller + Tijuana Taxi

Let’s Simplify: Plain Chant

- a.k.a. Gregorian plainsong or Byzantine chant
- Monophonic
  - No tritones ("diabolus in musica") because … no chords!
- Strict voice leading rules
- Vocal only (no instrumentation to worry about)

Pie Jesus Domine, Dona Eis Requiem ... {Headsmack}



Composer’s Workflow, Part 2

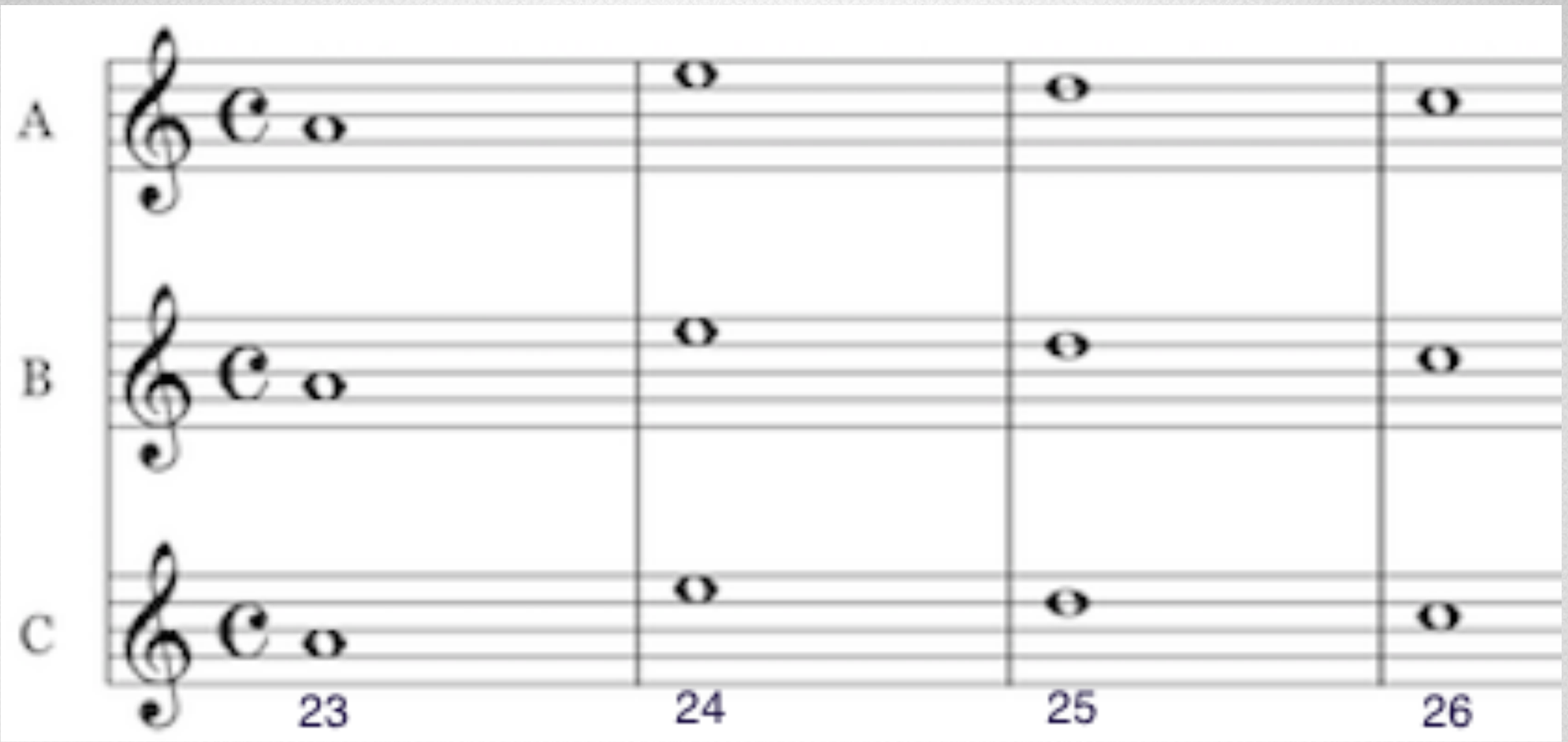
- Each composer acts independently
- All composers can hear humming in the same room
  - But cannot hear humming in other rooms or coffee shop
- Each composer has a private manuscript to copy **consensus music measures**
- All use indelible ink, impossible to change once written.
- Ignore anachronisms, e.g. music measures didn't exist in 6th century

Composing A Measure Of Music

1. Check who is in the room & music in earlier measures
2. Check rules, tastes of composers in the room, …
3. Choose a note for the next measure and hum it.
4. If unison, then all agree: write note in private manuscript.
5. If not unison, then there's disagreement
   - Leave the current measure blank, choose the next measure number, go to step #1.

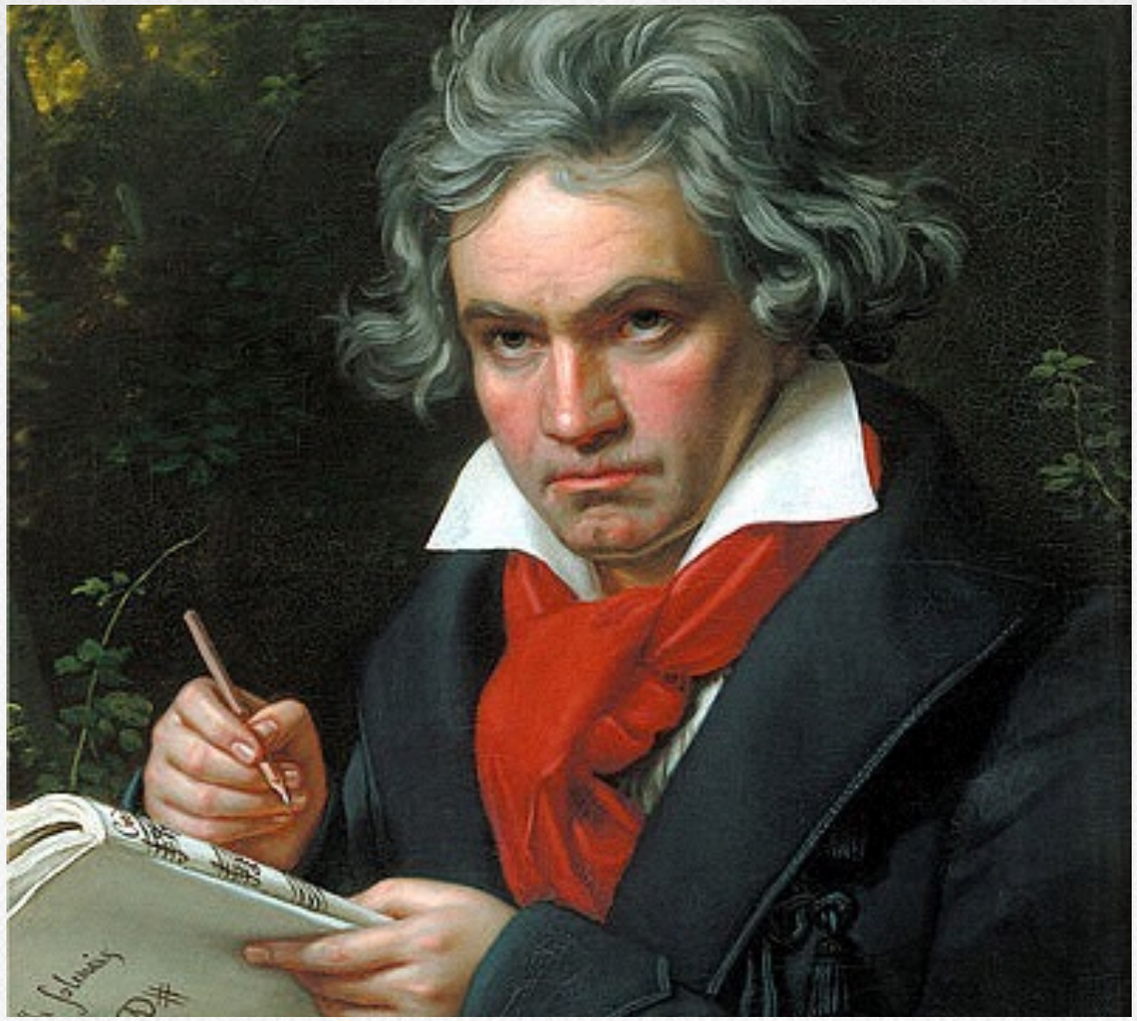
Interruptions, Disagreement, Etc.

- Each group in each room acts independently.
- If someone leaves the room? Write a new measure.
- If someone enters the room? Write a new measure.
- If someone takes a nap in the room? Write a new measure.
  - If they try to (re)use an old measure number, scold them, refuse the idea, and choose a new number



The Results Might Be...

WHAT IF THE COMPOSERS ARE DEAF?



For Example: Ludwig Von Beethoven

Use Two Manuscripts!

- “Public” manuscript: write here instead of humming
  - “Listen” by reading public manuscripts
  - **Anyone can read and write** a public manuscript
    - Helps us with slow/sleeping composers….
- “Private” manuscript: same use as our allegory
  - Anyone can read from it, only the owner can write to it

</FANFIC MODE="LAMPORT">

s/Lamport/Dijkstra/ if $MarkAllen_p

Question

- How frequently do composers talk directly to each other?

WHAT IF THE COMPOSERS ARE COMPUTERS PROGRAMMED BY… ELVES?

Our Model

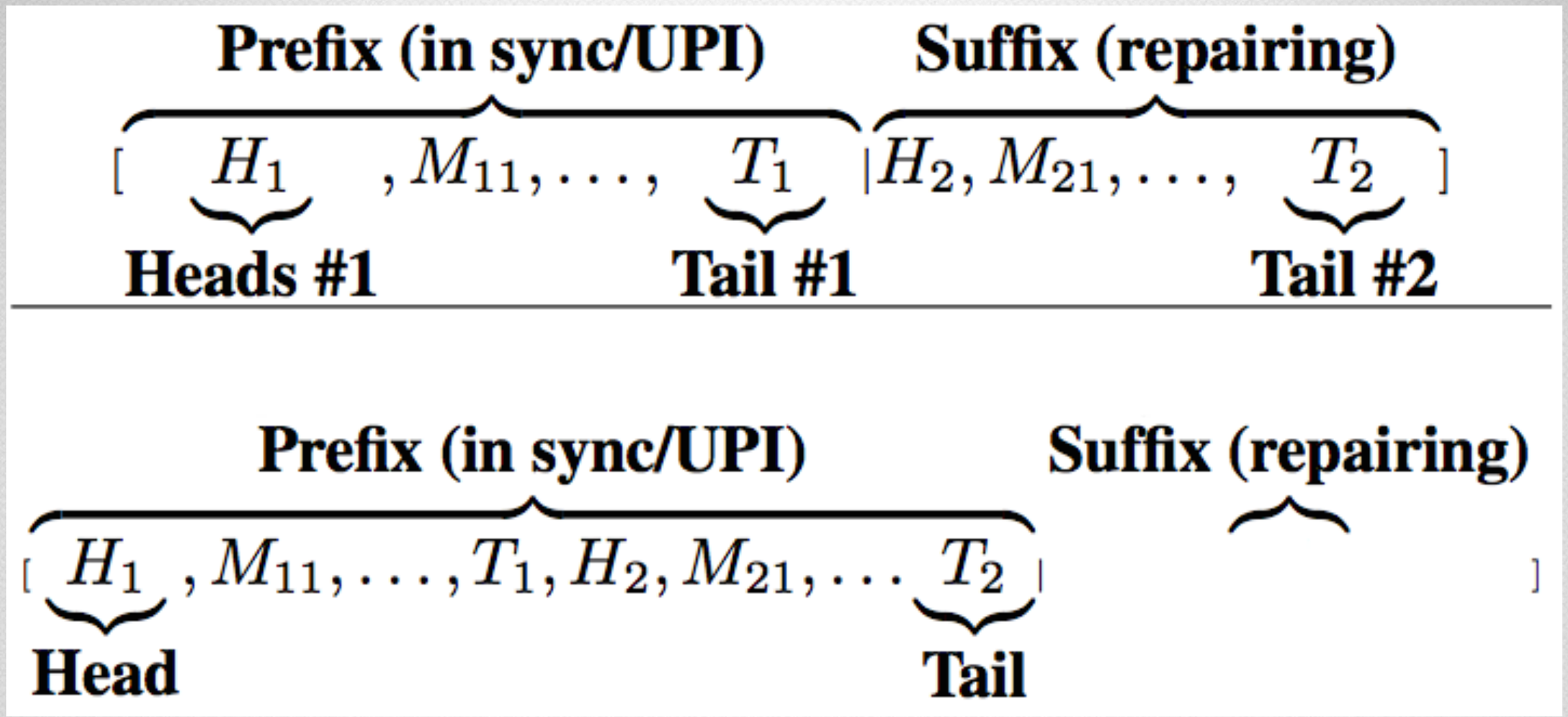
- Hosts & processes: If a client query times out, server can be considered down (weaker than fail-stop).
- Failure detection is not accurate or timely.
- Network: Messages can be dropped or reordered
  - Message corruption is detectable via checksum verification

Creeping Formality

- Measure number -> epoch number
  - Epoch = time period when chain metadata is stable
  - Chain metadata: membership, order, etc.
- Manuscript -> KV store of write-once registers (“Projection Store”)
  - Key = epoch number + `(public | private)`
  - Value = projection data structure

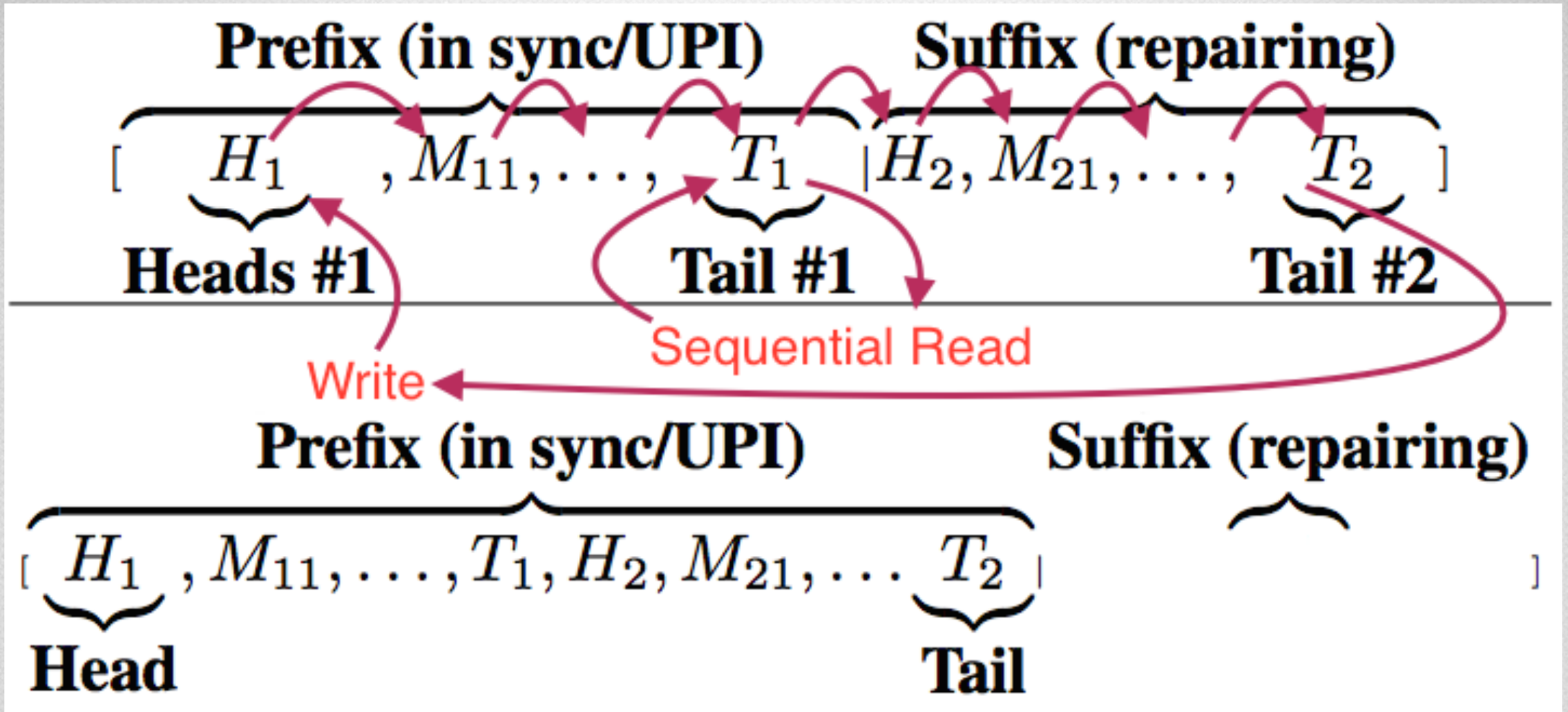
Creeping Formality

- Music composition rules -> chain state transition safety rules
  - Strict separation: “in sync” prefix, “out of sync/repairing” suffix
  - Never re-order “in sync” portion of chain
  - Move “in sync” -> “repairing” at any time
  - Move “repairing” -> “in sync” only after repair effort is OK
  - Move “repairing” -> “in sync” **only to end of in sync list**



Chain State Transition, Before & After Repair

UPI = “Update Propagation Invariant”

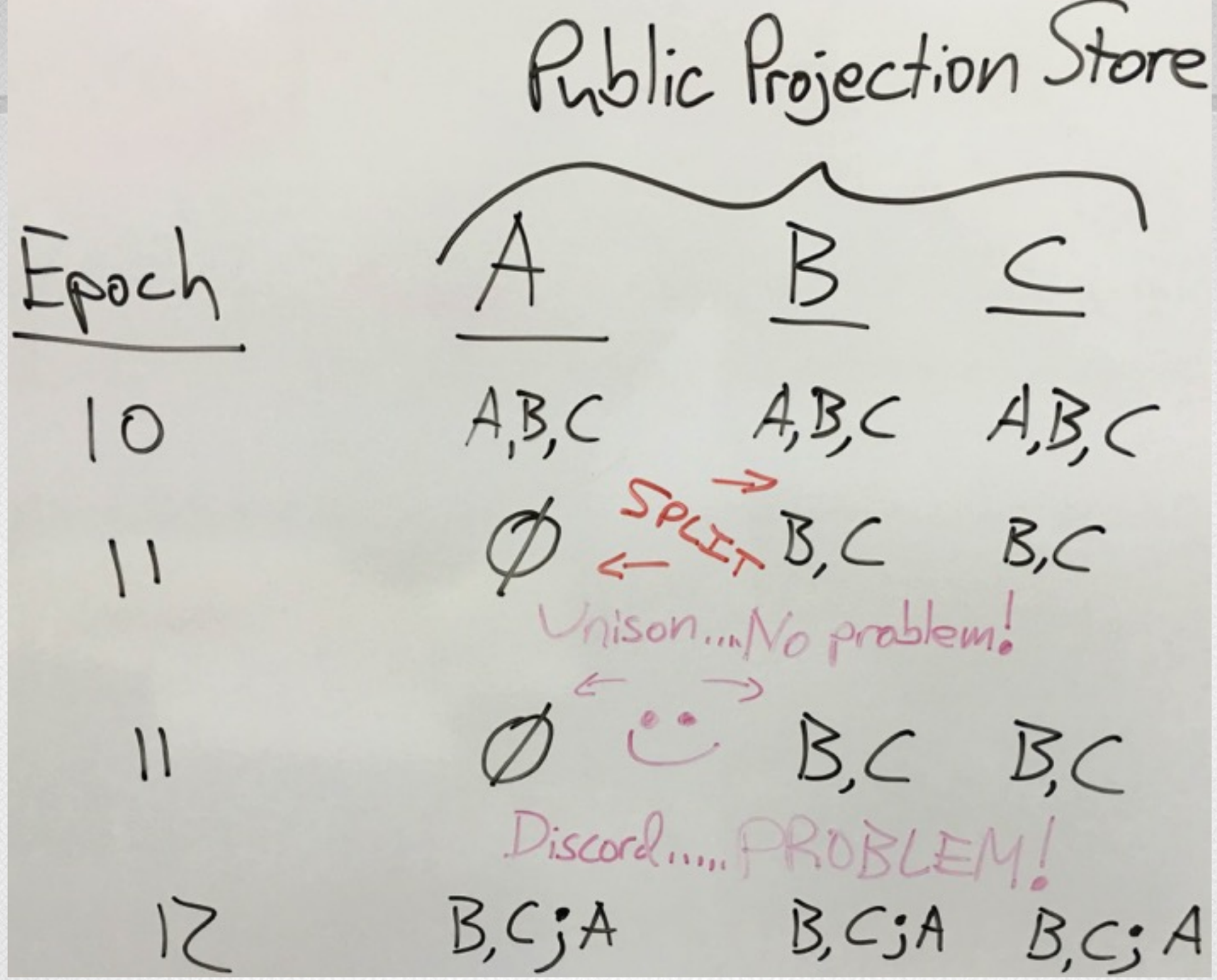


Chain State Transition, Before & After Repair

UPI = “Update Propagation Invariant”

Creeping Formality

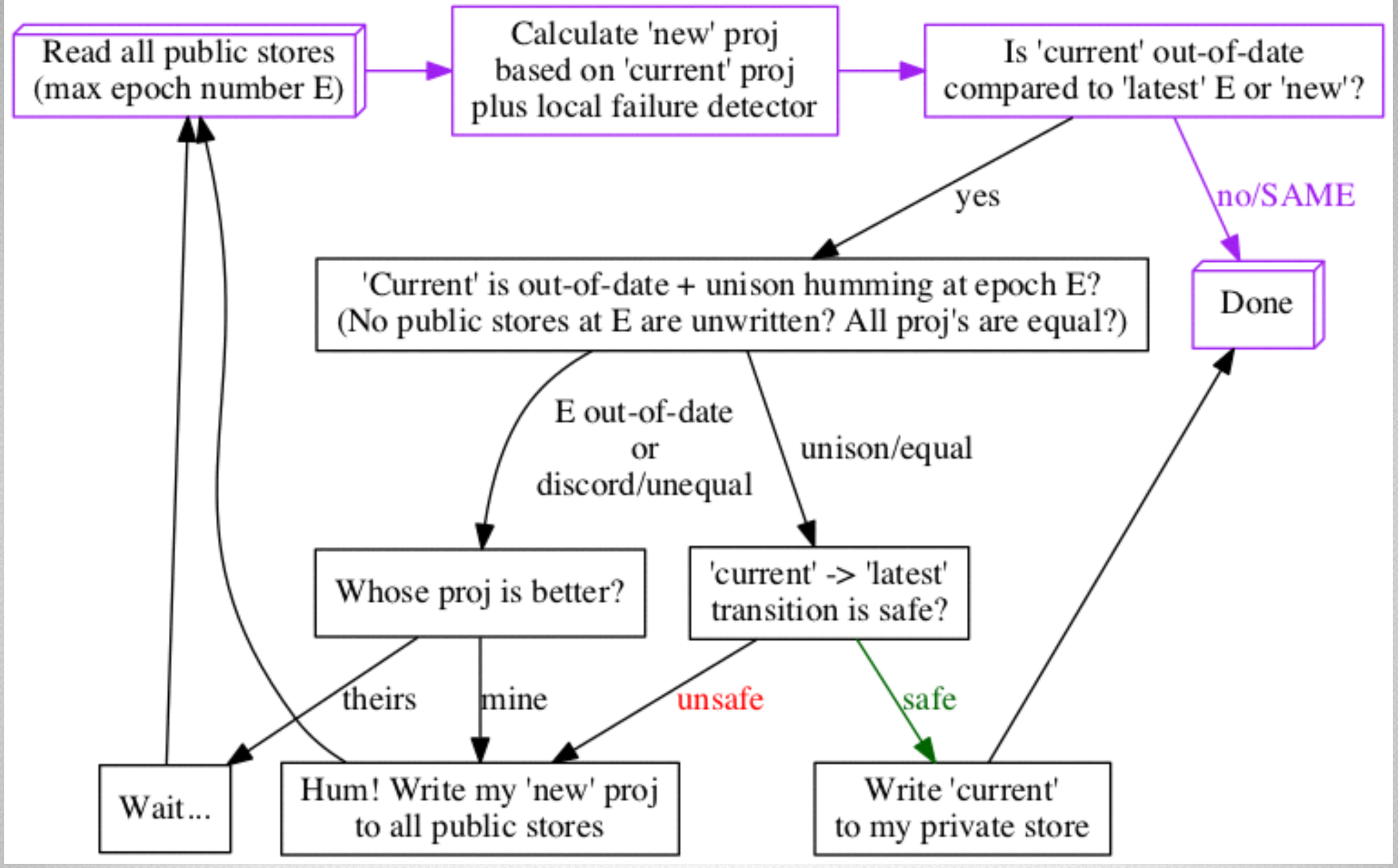
- A computer writes to **all available public** projection stores
  - **All available public** projections at epoch number **E** are equal -> “humming” in unison for epoch number **E**
- Private projection store remains writable only by owner
  - After writing highest private epoch number, use that projection for subsequent operation.

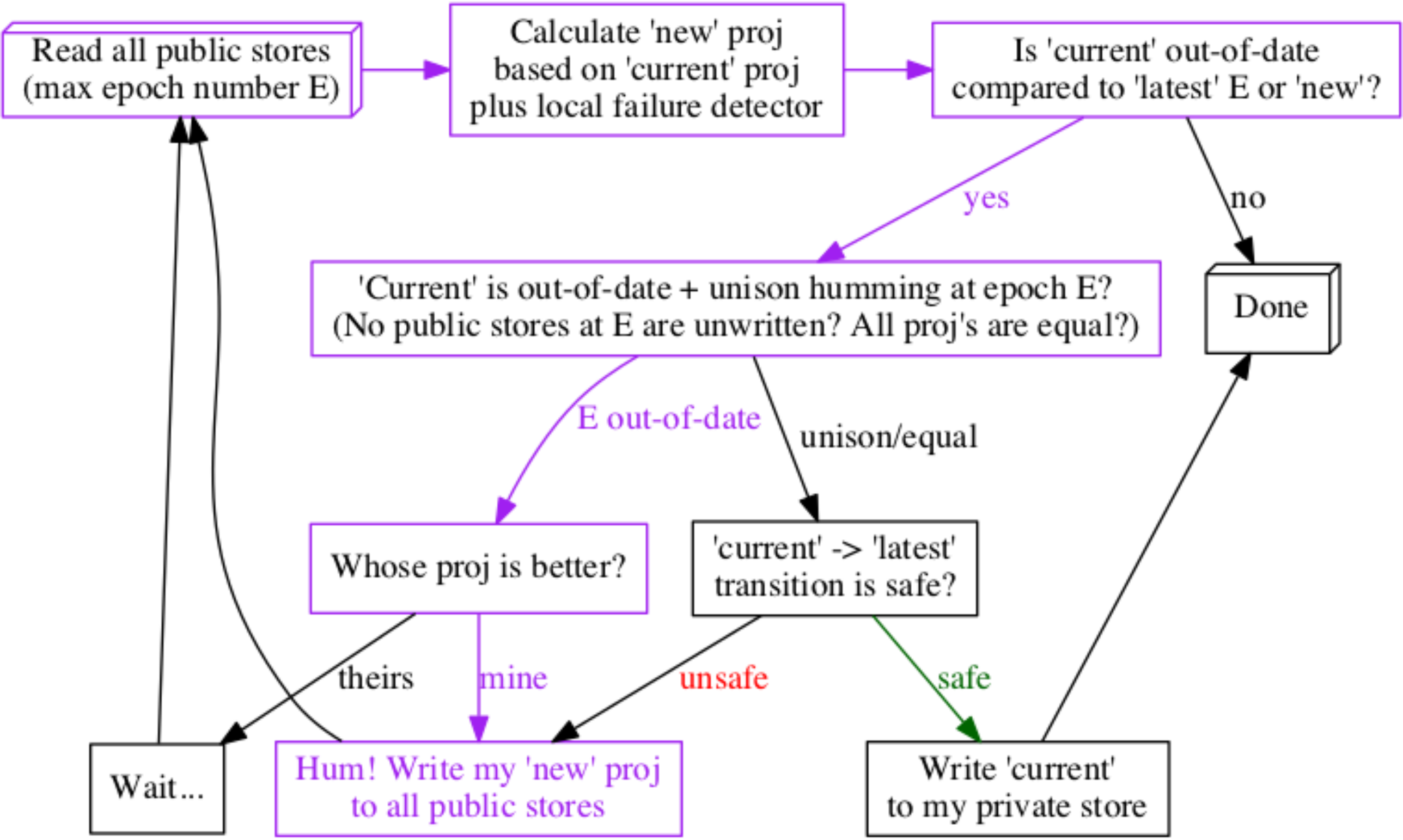


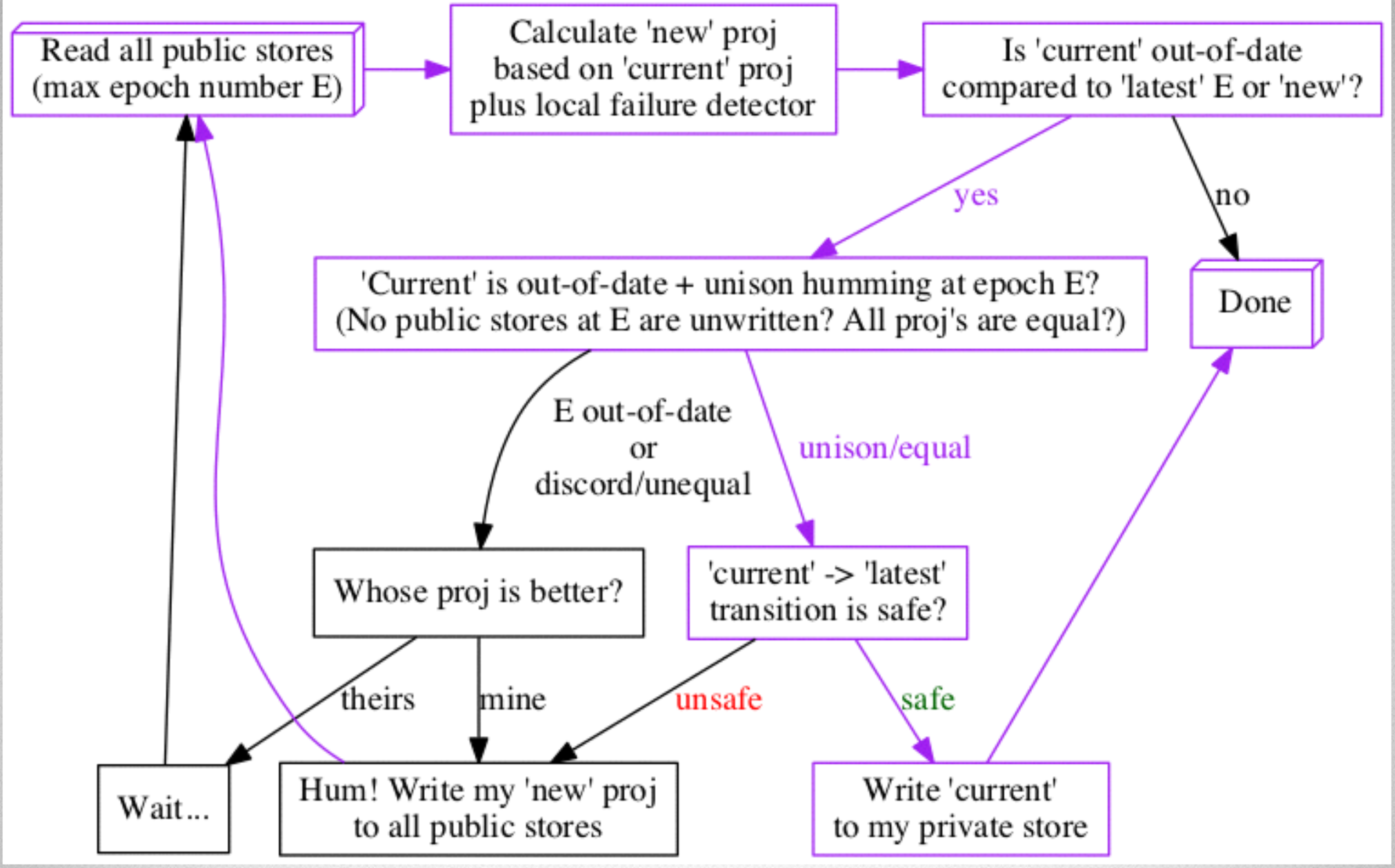
Sneak peek … we will return to this slide

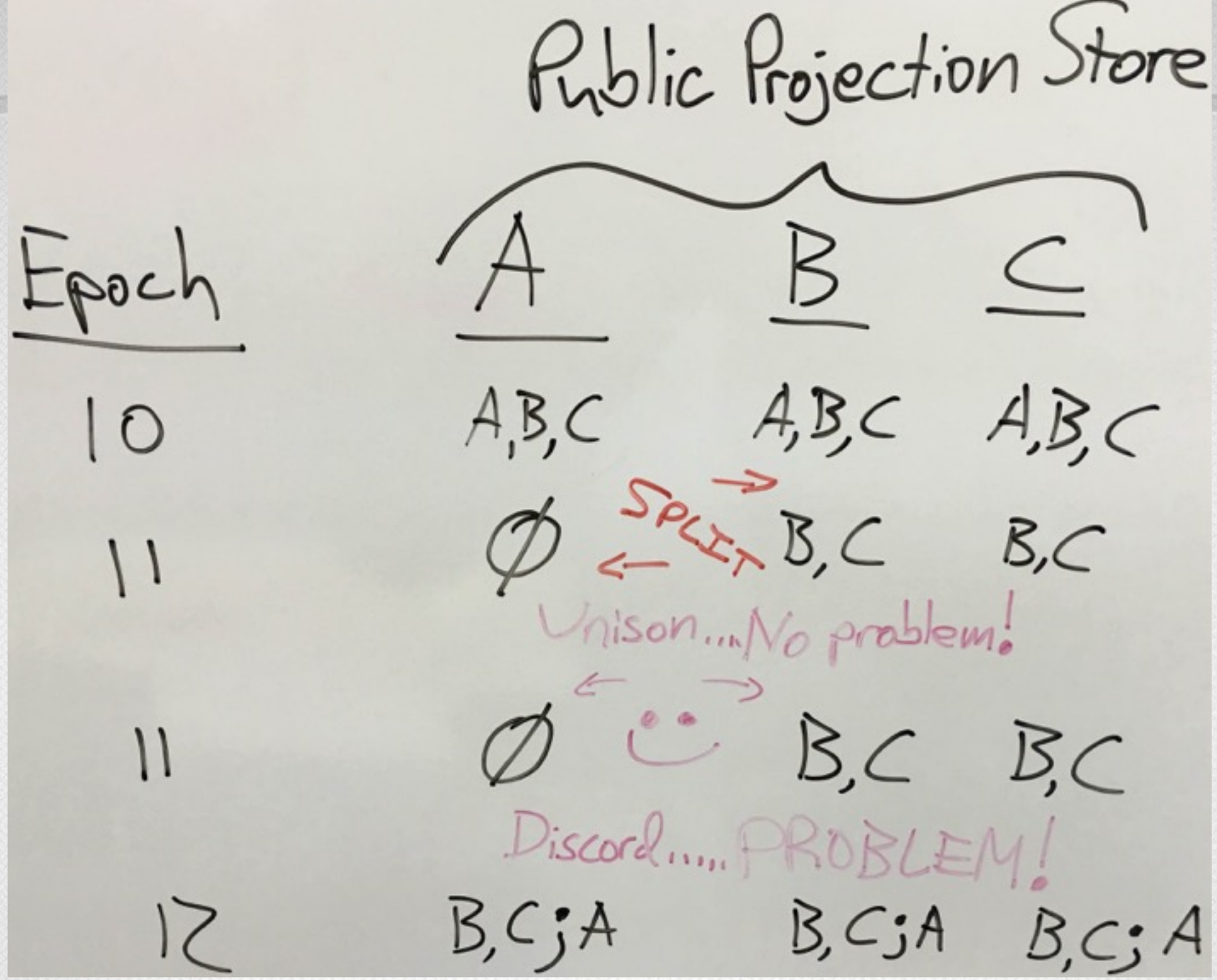
Yeah, Another Quiz Question

- How frequently do humming consensus participants communicate directly with each other?


Read all public stores
(max epoch number E)
Calculate 'new' proj
based on 'current' proj
plus local failure detector
Is 'current' out-of-date
compared to 'latest' E or 'new'?
yes
no/SAME
'Current' is out-of-date + unison humming at epoch E?
(No public stores at E are unwritten? All proj's are equal?)
Done
E out-of-date
or
discord/unequal
unison/equal
Whose proj is better?
'current' -> 'latest'
transition is safe?
theirs
mine
unsafe
safe
Wait...
Hum! Write my 'new' proj
to all public stores
Write 'current'
to my private store


Read all public stores
(max epoch number E)
Calculate 'new' proj
based on 'current' proj
plus local failure detector
Is 'current' out-of-date
compared to 'latest' E or 'new'?
yes
no
'Current' is out-of-date + unison humming at epoch E?
(No public stores at E are unwritten? All proj's are equal?)
Done
E out-of-date
unison/equal
Whose proj is better?
'current' -> 'latest'
transition is safe?
theirs
mine
unsafe
safe
Wait...
Hum! Write my 'new' proj
to all public stores
Write 'current'
to my private store


Read all public stores
(max epoch number E)
Calculate 'new' proj
based on 'current' proj
plus local failure detector
Is 'current' out-of-date
compared to 'latest' E or 'new'?
yes
no
'Current' is out-of-date + unison humming at epoch E?
(No public stores at E are unwritten? All proj's are equal?)
Done
E out-of-date
or
discord/unequal
unison/equal
Whose proj is better?
'current' -> 'latest'
transition is safe?
theirs
mine
unsafe
safe
Wait...
Hum! Write my 'new' proj
to all public stores
Write 'current'
to my private store



No conflict at epoch 11 … until the net-split heals

Question

- How frequently do humming consensus participants communicate directly with each other?

Different Modes Of Operation

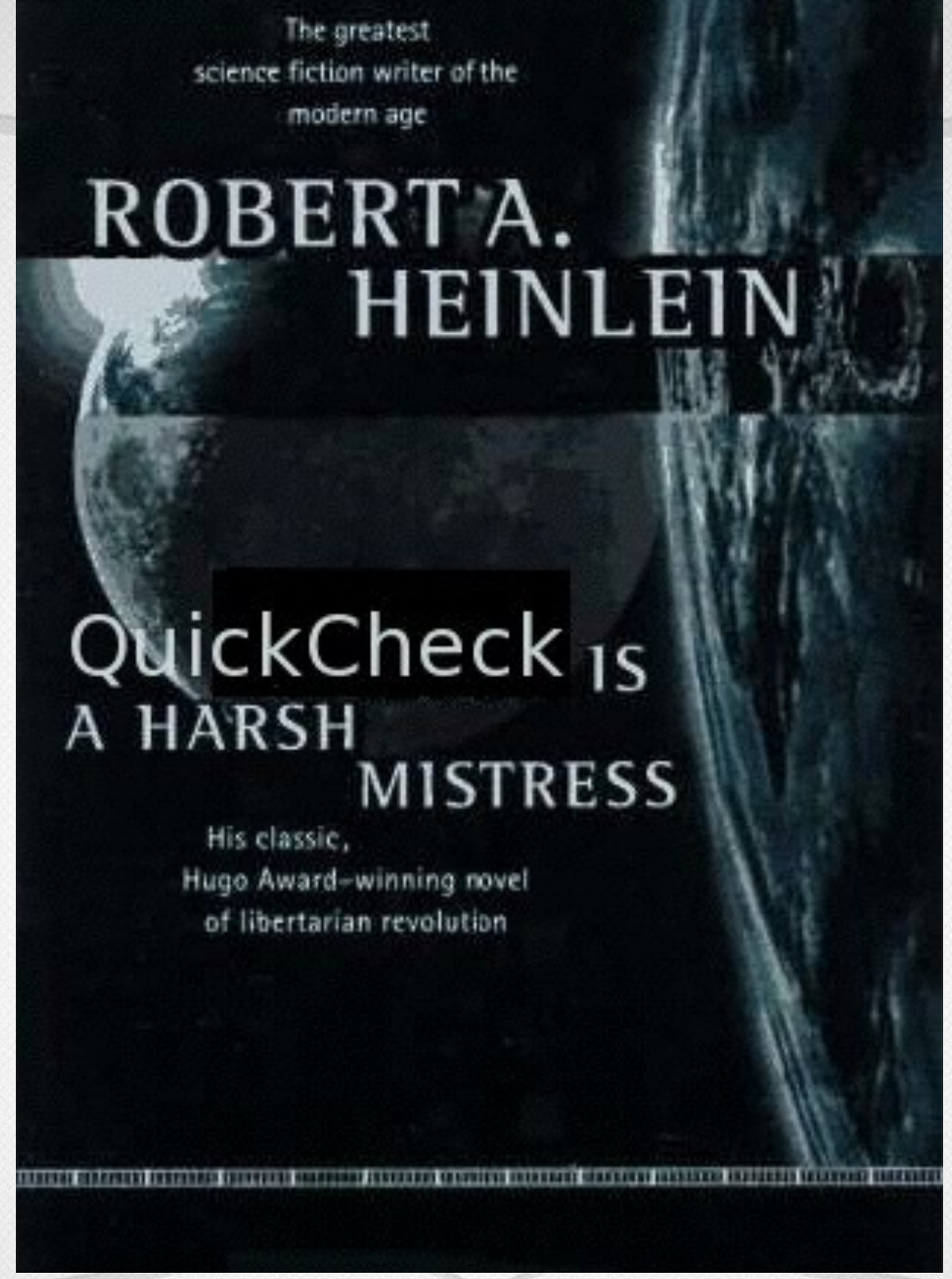
- Strong consistency: Chain length >= majority quorum size
  - CP mode minimum length prevents split brain syndrome
  - **2f+1** servers to tolerate **f** failures: no longer “cheap”
- Eventual consistency: Chain length = 1 is OK!
  - Machi files are write-once registers at byte level, all Machi file ops are CRDT-like, always mergeable
  - Humming Consensus can merge and repair chains after network partition

Cheating The 2F+1 Chain Length

- Avoiding split brain: 2f+1 of “real” servers + “witness” servers
  - A, B, and C are real servers: humming consensus + file service; W1 & W2 are “witness servers” (humming consensus only + quick epoch number check on read/write)
- Zero real server failures: **A -> B -> C**, 5 of 5 in h.c., 3 real
- One real server failure: **W1 -> B -> C**, 4 of 5 in h.c., 2 real
- Two real server failures: **W1 -> W2 -> C**, 3 of 5 in h.c., 1 real

TODAY’S DEVELOPMENT STATUS

No Formal Proofs Yet


The greatest
science fiction writer of the
modern age
ROBERT A.
HEINLEIN
QuickCheck is
A HARSH
MISTRESS
His classic,
Hugo Award-winning novel
of libertarian revolution

Today’s Humming Consensus

- Fully implemented (Erlang, service-agnostic (mostly))
  - Works well in network partition simulator
  - **Property-based testing has been invaluable,** with & without using QuickCheck
- Hasn't seen The Real World yet!
- Source & docs: https://github.com/basho/machi

Network Partition Simulator

- Map: simulate uni-directional message drops between actors
  - Example: **A->B** drop messages but **B->A** is OK
- Partition map may change at random intervals
- Partition map may remain frozen/stable
- Asymmetric partitions cause more chatter & churn, but HC copes well enough today, still much room for improvement.
- Today's code's worst case: 7 or 9 actors (livelock struggle)

Thank You!

github.com/basho/machi … lots more in the ‘docs’ directory

bit.ly/humming-2015

REFERENCES AND CREDITS

For More Information

- Source code repo: https://github.com/basho/machi/
- Docs: https://github.com/basho/machi/tree/master/doc
- Chain replication and CORFU: section 11 of https://github.com/basho/machi/blob/95437c2f0b6ce2eec9824a44708217a266e880b6/doc/high-level-machi.pdf also, that paper's bibliography
- On Consensus and Humming in the IETF: https://www.ietf.org/rfc/rfc7282.txt
- NFS v2 RFC: https://www.ietf.org/rfc/rfc1149.txt
- Elastic Replication: https://www.cs.cornell.edu/projects/quicksilver/public_pdfs/er-socc.pdf
- The Part-time Parliament: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.132.2111&rank=1

For More Information

- HDFS: https://en.wikipedia.org/wiki/Apache_Hadoop#HDFS
- QFS: https://en.wikipedia.org/wiki/Quantcast_File_System
- WTF: http://arxiv.org/abs/1509.07821
  - Preprint of "The Design and Implementation of the Wave Transactional Filesystem"
- SeaweedFS: https://github.com/chrislusf/seaweedfs
- The original allegory: http://www.snookles.com/slf-blog/2015/03/01/on-humming-consensus-an-allegory/

Image Credits

- Composers: http://blog.mymusictheory.com/wp-content/uploads/2012/12/composers-mix-529x300.jpg
- Neil Conway: https://twitter.com/neil_conway/status/656713576422379520
- Mark Callaghan: https://twitter.com/markcallaghan/status/656810474365841410
- Chain replication diagram: https://github.com/hibari/hibari-doc
- Beethoven: https://upload.wikimedia.org/wikipedia/commons/thumb/6/6f/Beethoven.jpg/399px-Beethoven.jpg
- Monty Python: http://images4.static-bluray.com/movies/covers/23375_front.jpg
- Under construction: https://github.com/h5bp/lazyweb-requests/issues/99
- Heinlein book+modification: Orb Books cover, 1997 (?)
- Scott's photo library

Corfu-Style Epoch Management

- All client ops tagged with current epoch # E
- If client op **E < E_current**, then **server refuses op**
- Any hosed client is OOS until newer epoch is found.
  - ... by reading from servers' private projection stores
- If client op **E > E_current**, then **server wedges self**
- Any wedged server is OOS until newer epoch is chosen
  - ... by humming consensus