BUILD BIG WITH TINY TOOLS: IMMUTABILITY, CHECKSUMS, AND CRDTS

Scott Lystig Fritchie, Basho Japan

Erlang Factory 2016 San Francisco

2016-03-11 Friday

About Scott

- Senior software engineer @ Basho Japan, Tokyo
  - scott@basho.com, @slfritchie on Twitter
  - Tech lead for Basho's distributed file store "Machi"
- Erlang infatuation since 1999
- Co-Chair of the ACM Erlang Workshop 2016, Nara, Japan
  - I urge you to consider writing a paper for the workshop!

Outline

- A very brief introduction to Machi
- Append-only files compared to write-once files
- Immutability changes everything
- What is chain replication?
- Let's make some music: an allegory
- Machi and CRDTs
- Machi and Checksums
- Today's development status

町

Machi

“village” or “town”

Machi

- A distributed, fault tolerant, write-once blob store with file-like API
- Operate in strong consistency mode or eventual consistency mode
  - Eventually consistent files? Are you crazy?

町

Append-Only File Writing

```
[pid  1394]
open(“/tmp/foo",
      O_WRONLY|O_CREAT|O_APPEND,
      0666) = 14
```

The kernel is responsible for ordering all writes in append-only fashion

Not Talking About Log-Structured File Systems

- Sprite LFS
- Solaris/Illumos ZFS
- VAOFS: A Verifiable Append-Only File System for Regulatory Compliance

100% Append-Only Systems

- The Hadoop File System (HDFS)
- The Google File System (GFS)
- Windows Azure Storage (WAS)
    - More blob store than file store

Machi: A File Store/Blob Store Hybrid

- File store-like API
  - Files are ordered collection of bytes
  - Random access at any byte offset
- Blob-store like behavior
  - Server always determines "location" or “name"
  - Location/name examples: file name + offset, opaque string
  - Examples: WAS, Twitter Blobstore, Google Blobstore

Append-Only Vs. Write-Once

- Append-only files
  - Writes ordered by time = writes ordered by offset
- Write-once files
  - A byte/page is writable once
  - Writes can happen in any time order!

Erlang Users Know Immutability

```
foo() ->
    X = 42,
    X = X + 1.
```

Guaranteed to fail, by design.

Immutability Changes Everything

Pat Helland, CIDR 2015

Immutability Changes Everything

Pat Helland
Salesforce.com
One Market Street, #300
San Francisco, CA 94105 USA
01(415) 546-5881
phelland@salesforce.com

3STRACT

re is an inexorable trend towards storing and sending utable data. We *need immutability* to coordinate at a distance we *can afford immutability*, as storage gets cheaper.

s paper is simply an amuse-bouche on the repeated patterns of puting that leverage immutability. Climbing up and down the pute stack really does yield a sense of déjà vu all over again.

INTRODUCTION

vasn't that long ago that computation was expensive, disk age was expensive, DRAM was expensive, but coordination latches was cheap. Now, all these have changed using cheap putation (with many-core), cheap commodity disks, and p DRAM and SSD, while coordination with latches gets der because latch latency loses lots of instruction opportunities.

can now afford to keep immutable copies of lots of data, and payoff is reduced coordination challenges.

More Storage, Distribution, & Ambiguity

have *increasing storage* as the cost per terabyte of disk keeps pping. This means we can keep lots of data for a long time.

have *increasing distribution* as more and more data and work spread across a great distance. Data within a datacenter seems away". Data within a many-core chip may seem "far away".

have *increasing ambiguity* when trying to coordinate with ems that are far away… more stuff has happened since you've d the news. Can you take action with incomplete knowledge? you wait for enough knowledge?

Turtles All the Way Down [17]

various technological areas have evolved recently, they have ponded to these trends of increasing storage, distribution, and biguity by using immutable data in some very fun ways. We explore how apps use immutability in their ongoing work, apps generate immutable DataSets for later offline analysis, SQL can expose and process immutable snapshots, how sively parallel "Big Data" work relies on immutable DataSets. s leads us to looking at the ways in which semantically utable DataSets may be altered while remaining immutable.

t, we consider how updatability is layered atop the creation of immutable files via techniques like LSF (Log Structure File ems), COW (Copy on Write), and LSM (Log Structured ge trees). We examine how replicated and distributed file ems depend on immutability to eliminate anomalies.

his article is published under a Creative Commons Attribution License ttp://creativecommons.org/licenses/by/3.0/), which permits istribution and reproduction in any medium as well allowing erivative works, provided that you attribute the original work to the thor(s) and CIDR 2015.

th Biennial Conference on Innovative Data Systems Research CIDR '15) January 4-7, 2015, Asilomar, California, USA.

Next, we discuss how the hardware folks have joined the party leveraging these tricks in SSD and HDD. See Figure Finally, we look at some trade-offs with using immutable data.

Append-Only Apps	App over Immutable Data: Record Facts then Derive
App Generated DataSets	Generate Immutable Data
Massively Parallel "Big Data"	Read & Write Immutable DataSets
SQL Snapshots & DataSets	Generate Immutable Data
Subjectively Immutable DataSets	Interpret Data as Immutab
LSF, LSM, and COW	Expose Change over Immutable Files by Appen
Immutable Files	Replication of Files/Blocks without Update Anomalies
Wear Leveling on SSD	Change via COW to Spre Physical Update Blocks
Shingles on HDD	Change via COW to Allow Large Physical Rewrites

Figure 1. Immutability is a key architectural concept at many layer of the stack.

2. Accountants Don't Use Erasers

Lots of computing can be characterized as "append-only". T section looks at some of the ways this is commonly accomplish

2.1 "Append-Only" Computing

May kinds of computing are "Append-Only". Observations recorded forever (or for a long time). Derived results calculated on demand (or periodically pre-calculated).

This is similar to a database management system. Transact logs record all the changes made to the database. High-spe appends are the only way to change the log. From t perspective, the contents of the database hold a caching of latest record values in the logs. The truth is the log. The datab is a cache of a subset of the log. That cached subset happens to the latest value of each record and index value from the log.

2.2 Accounting: Observed & Derived Facts

Accountants don't use erasers or they go to jail. All entries i ledger remain in the ledger. Corrections can be made but only making new entries in the ledger. When a company's quarte results are published, they include small corrections to previous quarter. Small fixes are OK! They are append-only, to

Some entries describe observed facts. For example, receivin debit or credit against a checking account is an observed fact.

Some entries describe derived facts. Based on the observatio we can calculate something new. For example, amortized cap expenses based upon a rate and a cost. Another example is current bank account balance with applied debits and credits.

Write-Once Register In Erlang

```
-record(wor,{set=false :: boolean(),
             val :: undefined|val_type()
}).

set(#wor{set=false}=WOR, Val) ->
    WOR#wor{set=true, val=Val}.

get(#wor{set=false}) ->
    undefined;
get(#wor{set=true, val=Val}) ->
    {ok, Val}.
```

Why Write-Once Files?

- Maintaining time-oriented ops in a distributed system is hard
  - Because time is hard
- Avoid “time”, use “space” instead
  - Assign once: file name + offset + byte range size
  - Enforce write-once behavior for every byte
  - Actual write ops can be processed in any time order

Machi API (simplified)

```
-spec append_chunk(
          Prefix:string(),
          Chunk :binary(),
          CSum  :binary()) ->
    {'ok',{FileName:string(),
           Offset:non_neg_integer()}}
    | error_tuple().
```

Machi API (simplified)

```
-spec read_chunk(
          FileName:string(),
          Offset  :non_neg_integer(),
          Size    :non_neg_integer()) ->
    {‘ok’,{Chunk:binary(),
           CSum :binary()}}
    | error_tuple().
```

WHAT IS CHAIN REPLICATION?

Much More About Chain Replication And Humming Consensus

http://ricon.io/archive/2015/

Scott Lystig Fritchie

Basho

Managing chain replication metadata with Humming Consensus

» **SLIDES** | » **VIDEO**

Neil Conway
@neil_conway

Following

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

RETWEETS	FAVORITES
5	13

3:08 PM - 21 Oct 2015

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."

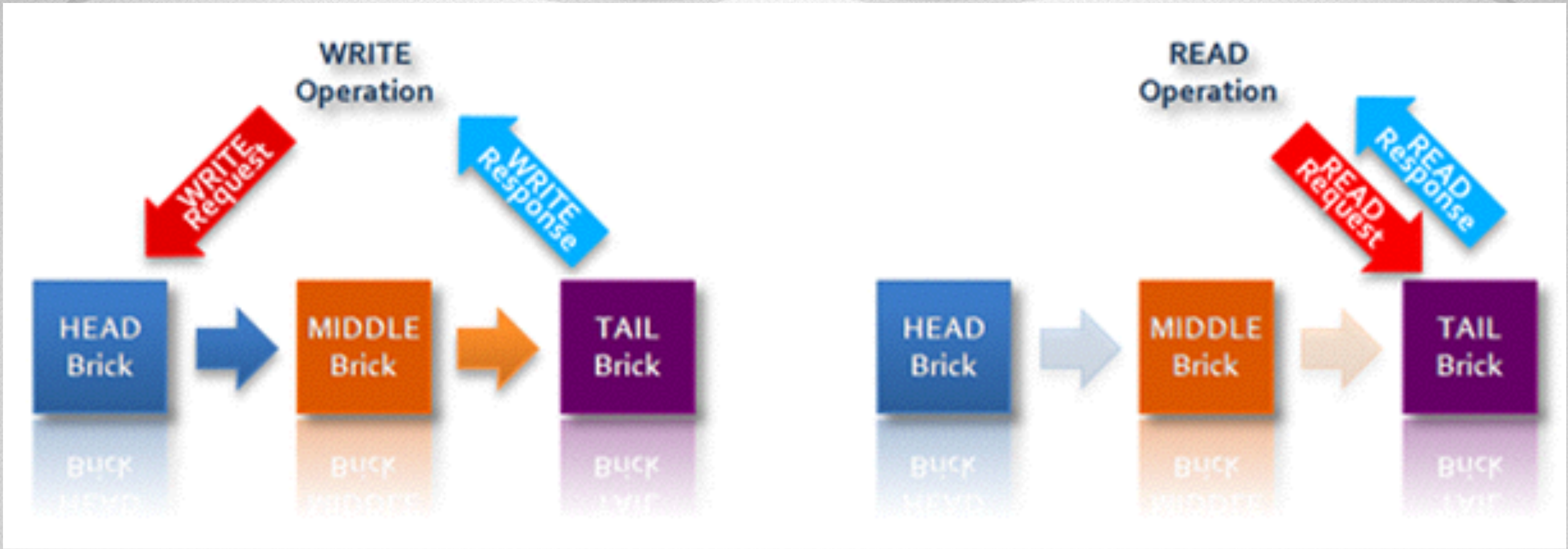
Chain Replication Papers

- 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.

markcallaghan
@markcallaghan

Follow

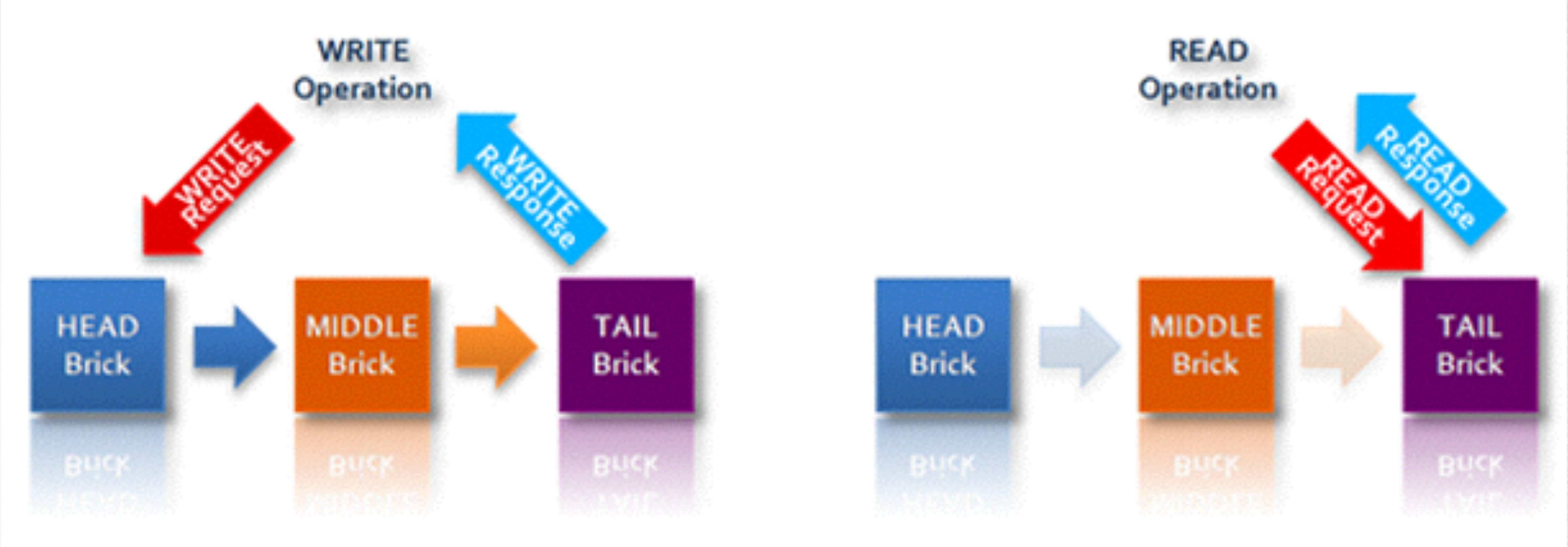
@neil_conway write to front, read from the back. The mullet of replication?

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
  - Rub some Paxos/Raft/ZooKeeper/etcd on it…..
- The **availability of your distributed system** is limited by the **availability of the system’s manager**!
  - Don’t use SC system to manage an EC system like Riak or EC-mode Machi

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

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)

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.

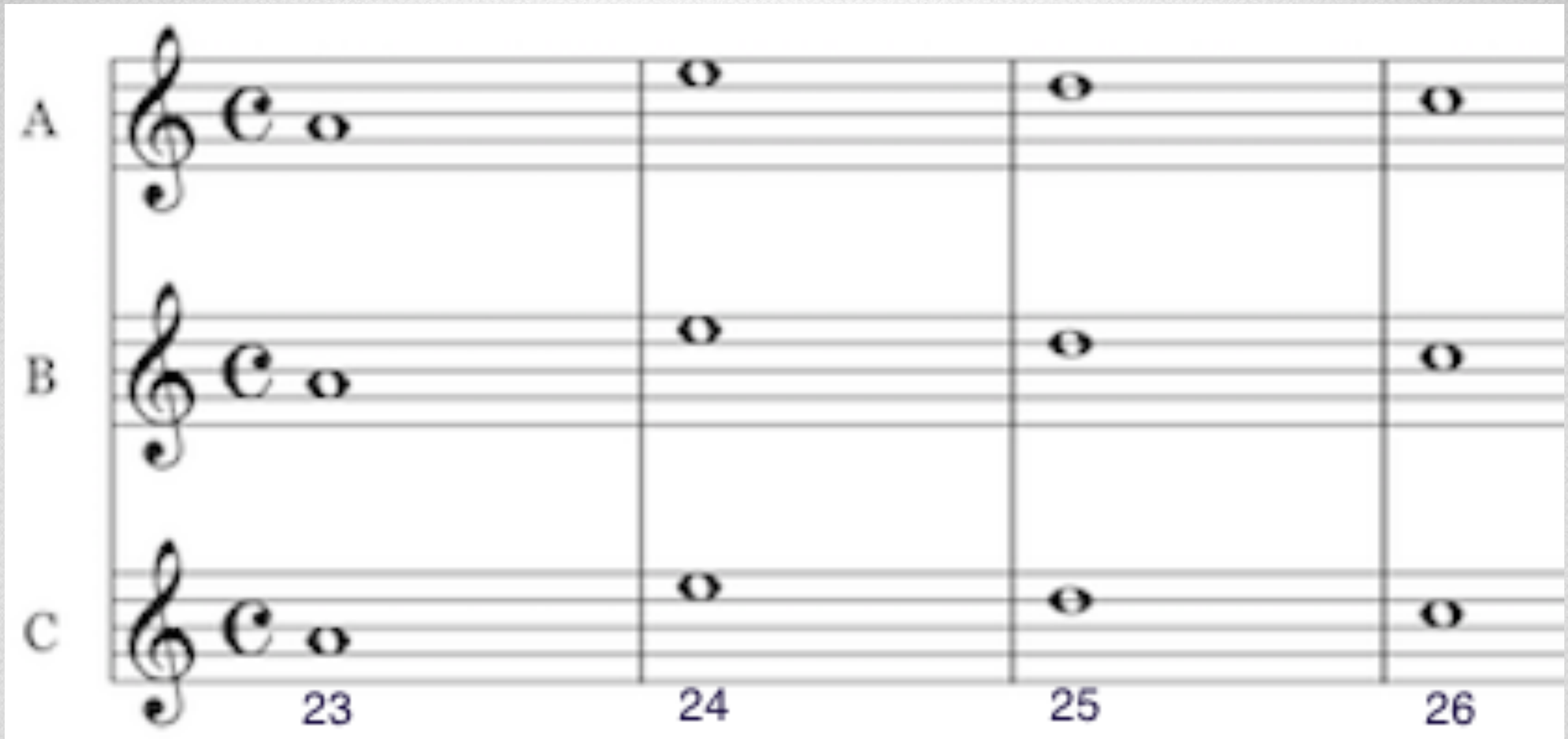
4. If unison, then all agree: write note in private manuscript.

5. If not unison, then there's disagreement

   - Leave the current measure blank, choose the next measure number, go to step #1.

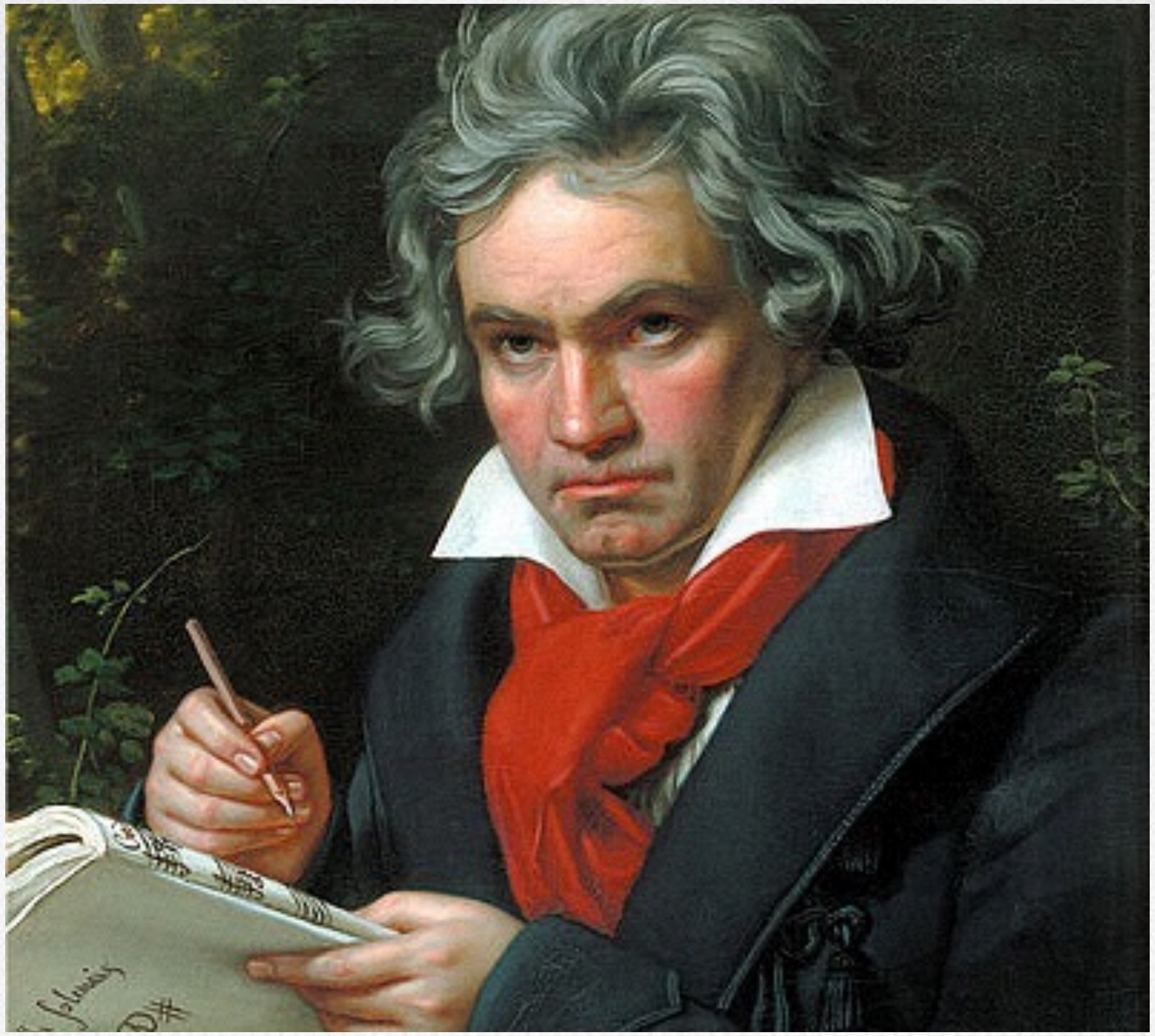
Interruptions, Disagreement, Etc.

- 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

WHAT IF THE COMPOSERS ARE COMPUTERS PROGRAMMED BY… ELVES?

Music To Algorithm

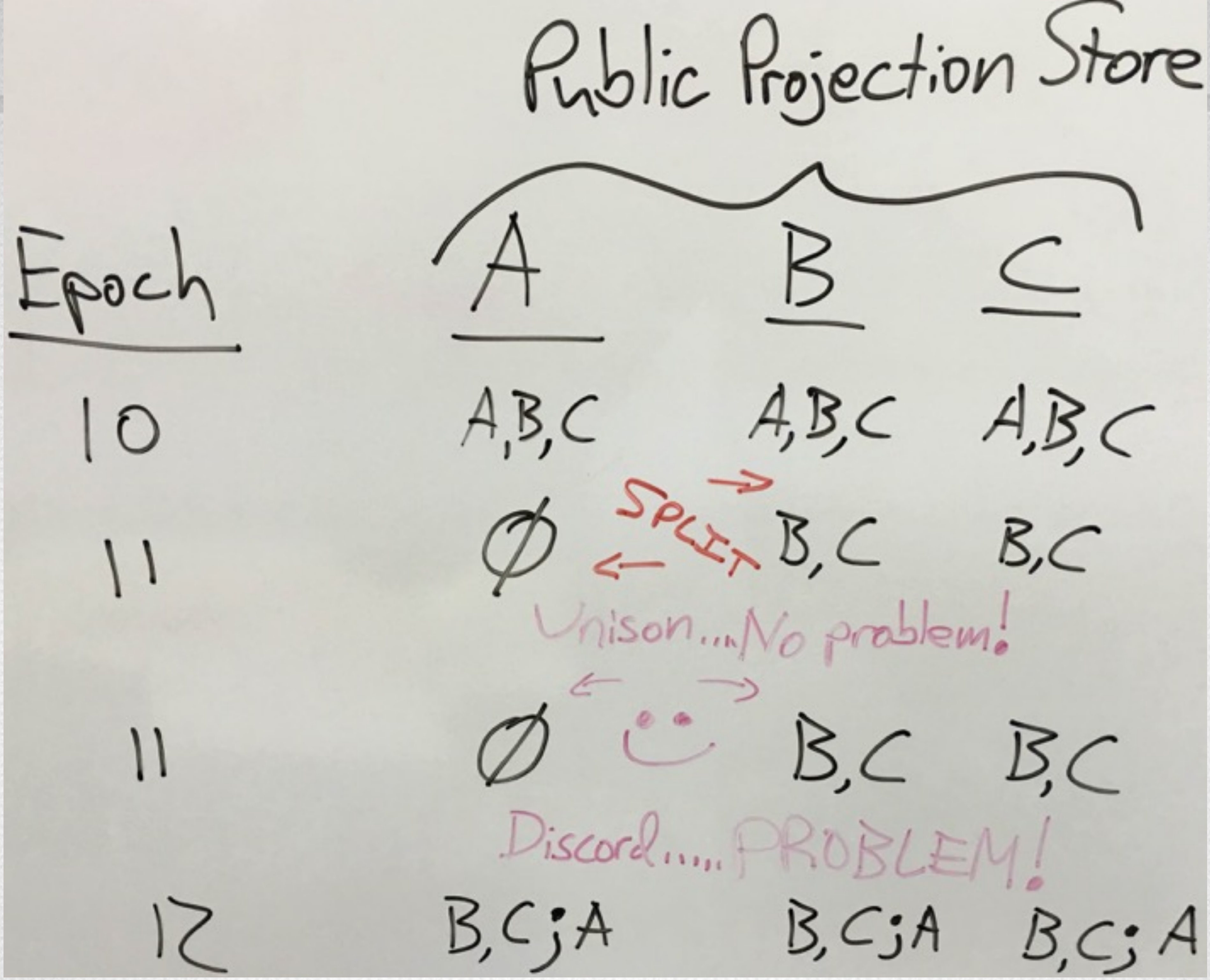
- Measure number -> epoch number
  - Epoch = time period when chain metadata is stable
  - Chain metadata: dynamic membership, chain order, etc.
- Manuscript -> KV store of write-once registers (“Projection Store”)
  - Key = epoch number + `(public | private)`
  - Value = projection data structure

Music To Algorithm

- 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.

Different Modes Of Operation

- Strong consistency: Chain length >= majority quorum size
  - Minimum length prevents split brain syndrome
- 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



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

Humming Consensus Summary

- Built upon write-once registers: the "projection store"
- If you hear unison music (i.e. read identical values from public projection stores), then you have to consider the change.
- If you like the change, accept it & write it to your private store.
- If you don't like the change (safety violation!), propose a new change in a new epoch. Always have the option to ignore a bad ideas/definitely unsafe chain configuration.
- "Hearing unison" may change to discord after a network partition heals. The fix: suggest new change in new a epoch.

MACHI AND CRDTS

C R D T

- Conflict-free Replicated Data Type: https://en.wikipedia.org/wiki/Conflict-free_replicated_data_type
- Basic rules: Commutative, Associative, Idempotent
- “If all updates are received, applying the updates in any order gives the same final result.”

CRDTs in Machi

- Informal use #1: unique file name + offset assignments create CRDT-like, always mergeable files
- Formal use #2: use “map” of “last-write-wins registers” to broadcast up/down visibility status to all chain members
  - Map key = observing server’s name
  - Map value = list of servers believed down by observer
  - riak_dt library: https://github.com/basho/riak_dt

Machi's "Fitness" Service

- Each participant has fitness service
- Fitness service queries all projection stores, any failures are added to local "I think it's down" list
- CRDT map of down lists are spammed to all other participants
- Convert map values -> digraph, then estimate where network partition(s) are located & effect (1-way, 2-way).
- New chain order removes the worst-affected servers

MACHI AND CHECKSUMS: AN ANTI-ENTROPY STORY

Clients Provide The Checksum

```
-spec append_chunk(
          Prefix:string(),
          Chunk :binary(),
          CSum  :binary()) ->
    {'ok',{FileName:string(),
           Offset:non_neg_integer()}}
    | error_tuple().
```

How Machi Uses Checksums

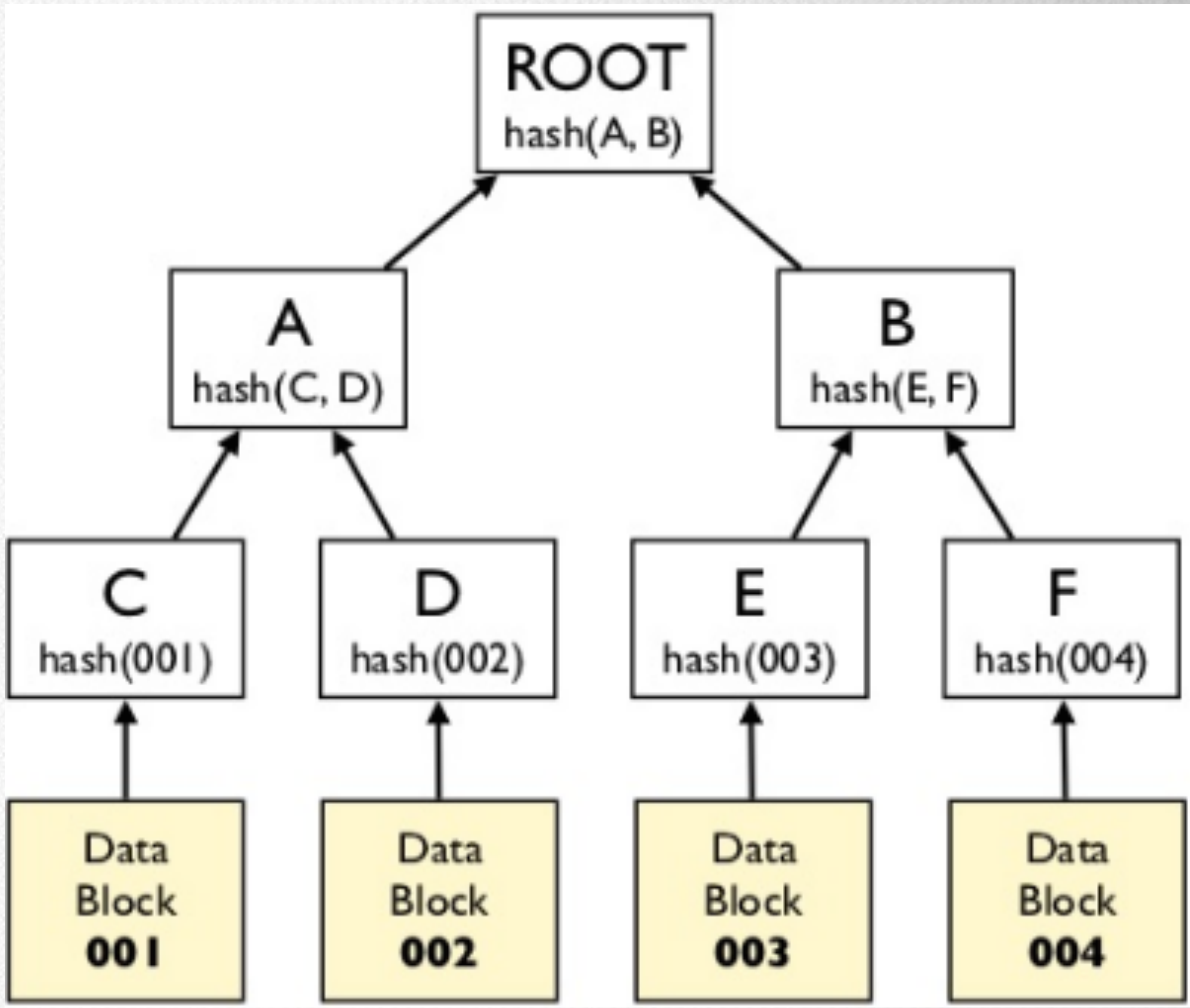
- Server verifies checksum at initial append/write time
- Server “scrubs” local data on disk, re-verifying checksums
  - Similar to RAID array parity scrub/sweep/scan
- Use Merkle-style trees of checksum data for file replication

Merkle Tree (Hash Tree)

- Leaf nodes: hash of original data block

- Interior nodes: hash of concatenation of child hashes

- Sensitive to data block contents **and also tree shape**



Merkle Trees Are Great, But…

- Good news: You have 220 TBytes of data on this modern, high-density server.
- Bad news: You must read all 220 TBytes of data to create a single Merkle tree.

Can we find a short-cut?

Standard Merkle Tree Vs. Machi's

- Leaf nodes: hash of **original data block**
- Interior nodes: hash of concatenation of child hashes
- I/O required is all original data

- Leaf nodes: hash of **concatenation of checksums in block range**
- Interior nodes: hash of concatenation of child hashes
- I/O required is all checksums (~32 bytes each)

Leaf Node Representation

- Unwritten bytes: `<<Length:64, Offset:32, 0>>`
- Written bytes: `<<Length:64, Offset:32, CSum/binary>>`
- Trimmed bytes: `<<Length:64, Offset:32, 1>>`
  - Trimmed = garbage collected & no longer accessible
  - Valid transition: unwritten -> written -> trimmed
  - Valid transition: unwritten -> trimmed

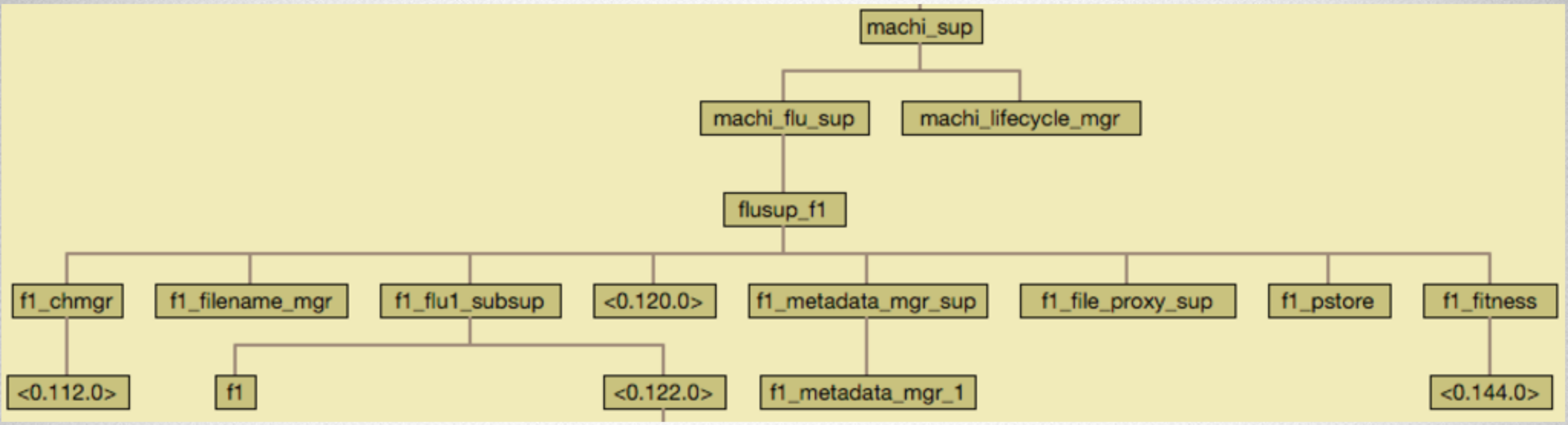
TODAY'S DEVELOPMENT STATUS

Not Finished Yet

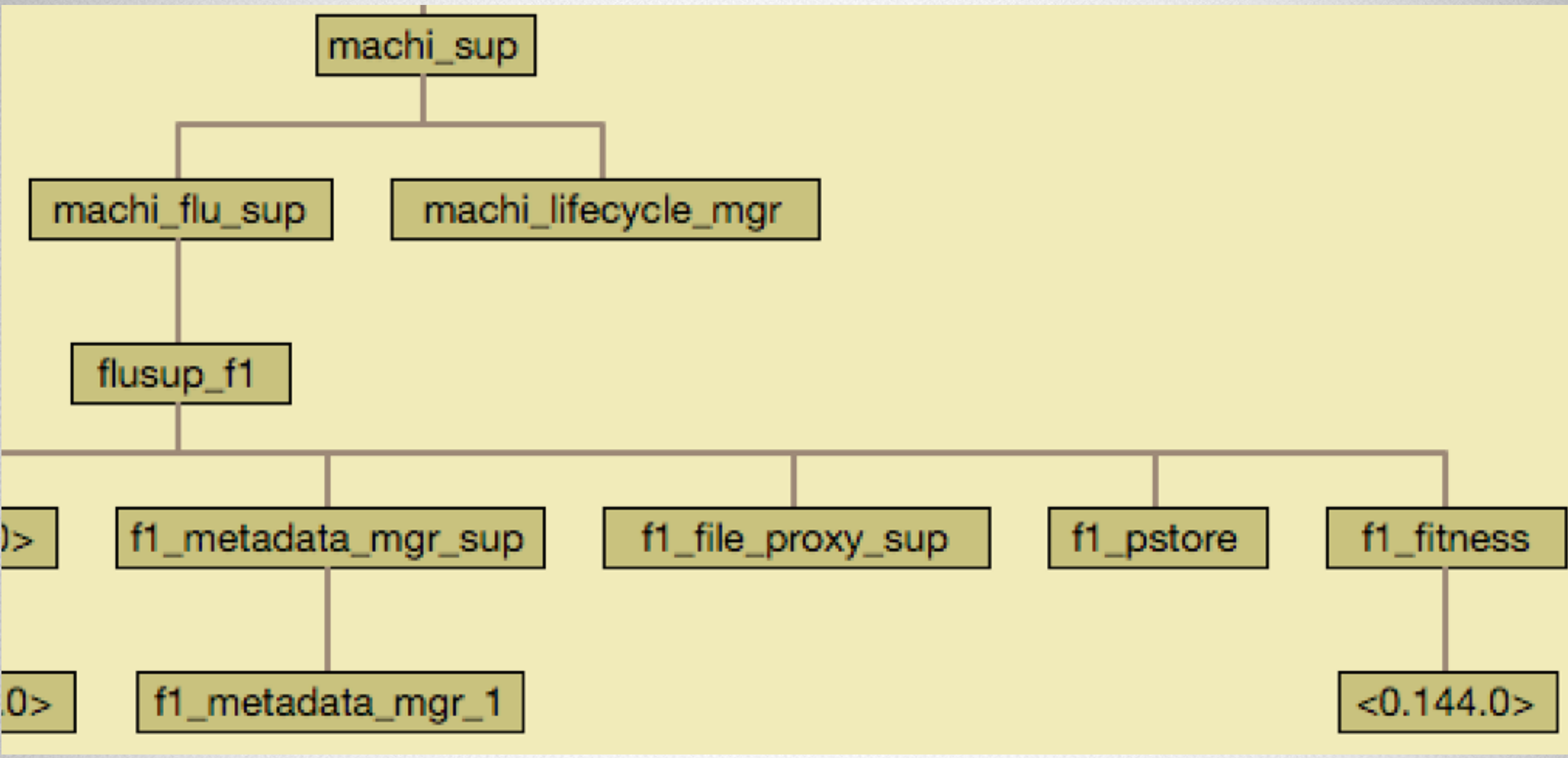
Today’s Humming Consensus

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

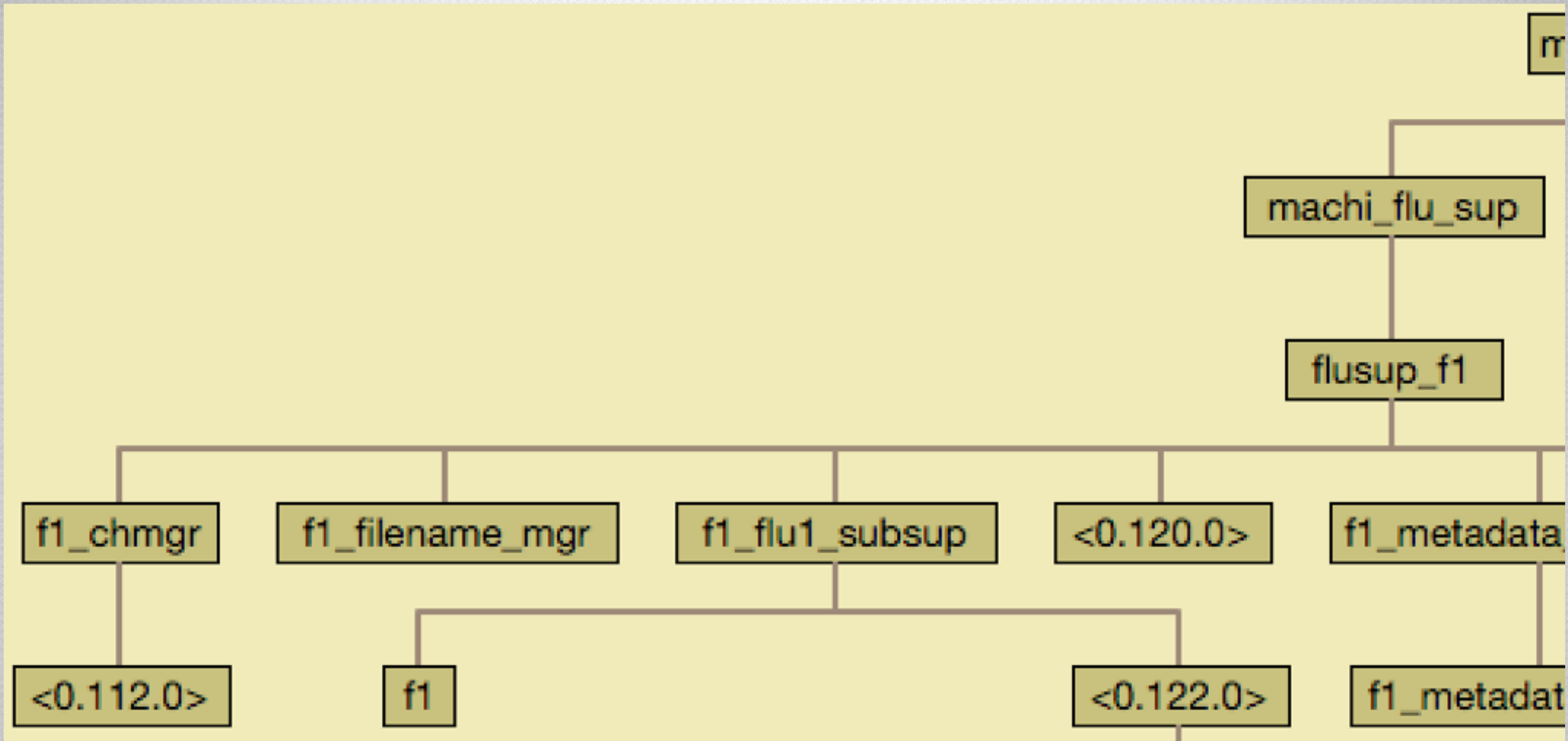
Supervision Tree

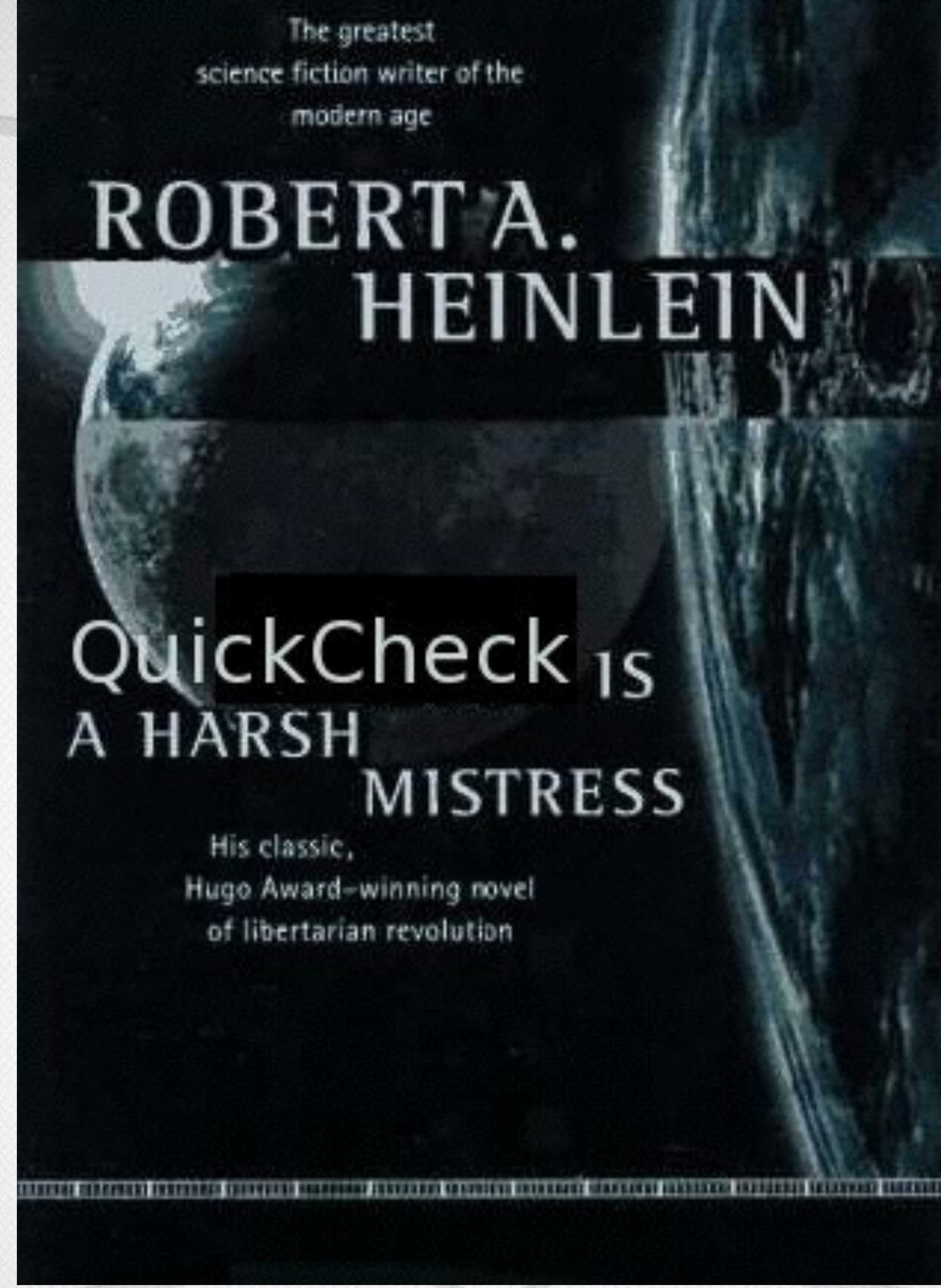


Supervision Tree



Supervision Tree




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

Property-Based Testing: Outline

- Each app/library/function has its invariants
- Identify those invariants! These are your properties.
- **Make the invariants executable**
    - Now you're flexible: plug these functions into EUnit, Common Test, PropEr, QuickCheck, etc.
    - Check invariants at runtime (probes, assertions) and/or after the fact (e.g., post-run analysis of event log)

Invariants For Chain Replication

- Machi-style:
  - 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**

Network Partition Simulator Tests

- One-way network partitions: A -> B fails but B -> A is OK
- Partition definition: [{FromServer, ToServer}, …]
    - List may remain constant or constantly/randomly change
- Run Humming Consensus in variable partitions ("shake the snow globe" random period), then in a fixed partition list.
- Wait for stable & unanimous chain(s). Fail if never stable.
- Check invariants in activity log afterward: chain order, etc.

Thank You!

github.com/basho/machi
https://github.com/basho/machi/tree/master/doc

REFERENCES AND CREDITS

For More Information

- Source code repo: https://github.com/basho/machi/
- Docs: https://github.com/basho/machi/tree/master/doc
- Scott's Ricon 2015 presentation on Humming Consensus: http://ricon.io/archive/2015/index.php
- 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
- 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/
- Immutability Changes Everything: http://www.cidrdb.org/cidr2015/Papers/CIDR15_Paper16.pdf

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

Image Credits

- Merkle hash tree diagram: http://www.cnblogs.com/fxjwind/archive/2012/06/08/2541818.html
- Heinlein book+modification: Orb Books cover, 1997 (?)
- Scott's photo library