



AgileLog: A Forkable Shared Log for Agents on Data Streams



Shreesha G. Bhat



Tony Hong



Michael Noguera



Aishwarya Ganesan



Ram Alagappan

University of Illinois Urbana-Champaign

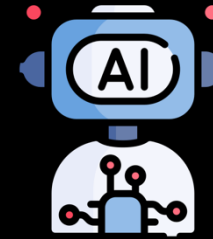
AI Agents: New Users of Data Systems

Traditional Human-written Programs



- Fixed, pre-programmed logic

AI Agents Powered by LLMs



- Accomplish high-level data tasks specified in natural language
- Use LLM reasoning to decide steps on-the-fly
- Invoke tools [1] to interact with data system

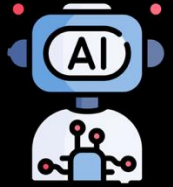
Modern data systems must **evolve for this new kind of user!** [2, 3]

[1] Schick et al. Toolformer: Language Models Can Teach Themselves to Use Tools. *NeurIPS*, 2023. Jane Dwivedi-Yu, Roberto

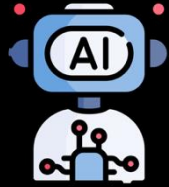
[2] Liu et al. Supporting Our AI Overlords: Redesigning Data Systems to be Agent-First. arXiv:2509.00997, 2025.

[3] Parameswaran et al. <https://bair.berkeley.edu/blog/2026/07/07/intelligence-is-free-now-what/>

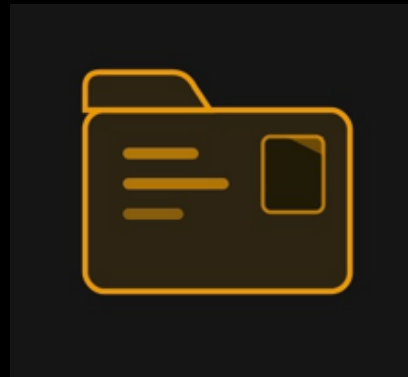
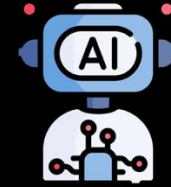
Agents on Data Systems



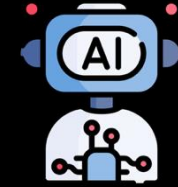
OLTP



OLAP



File System



Lakehouse

Our Focus: Live, Streaming Data



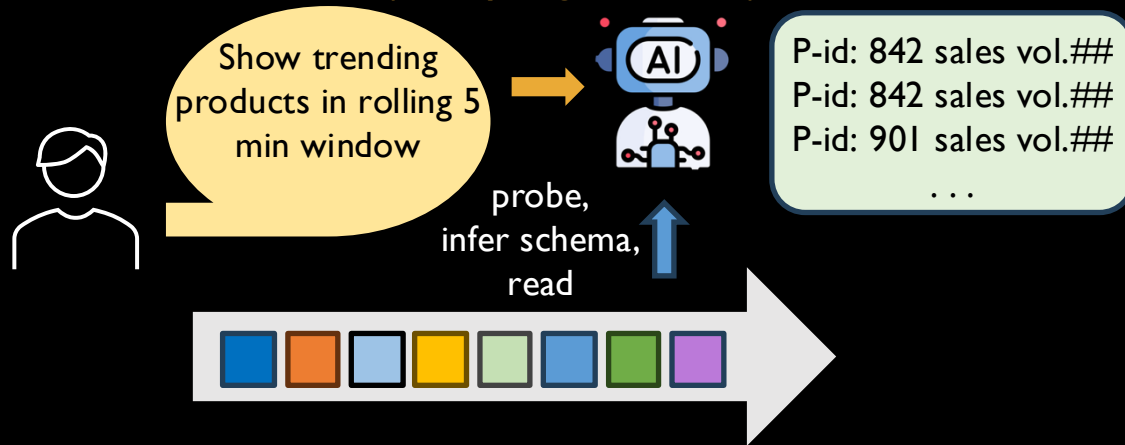
Streaming systems process **real-time data**

Much value to be gained by having agents directly act on them

All these systems already allow agents to interact with streaming data
(via MCP and tools)

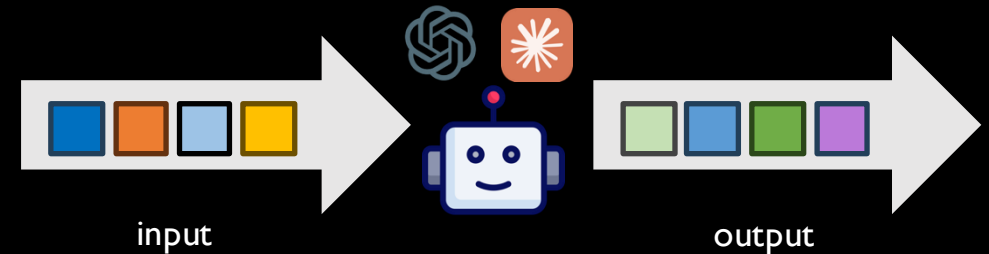
Agents Interacting with Data Streams

Agentic Analysis over Streaming Data via NL (non-programmers)

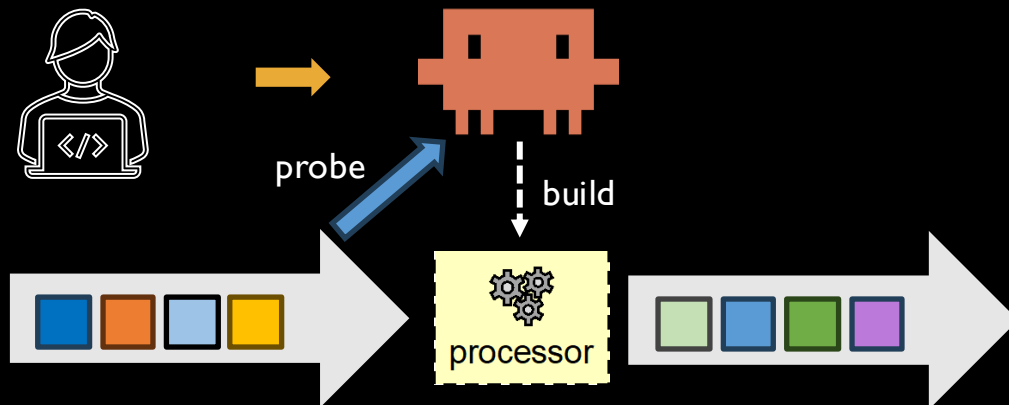


Agentic Stream Processors

Invoke LLM on each record
(e.g., for content moderation)
and write output to another stream

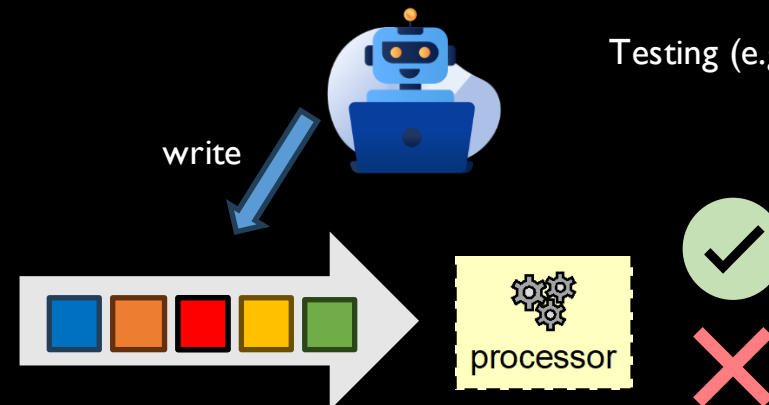


Coding Agents



Agentic Testing of Stream Processors

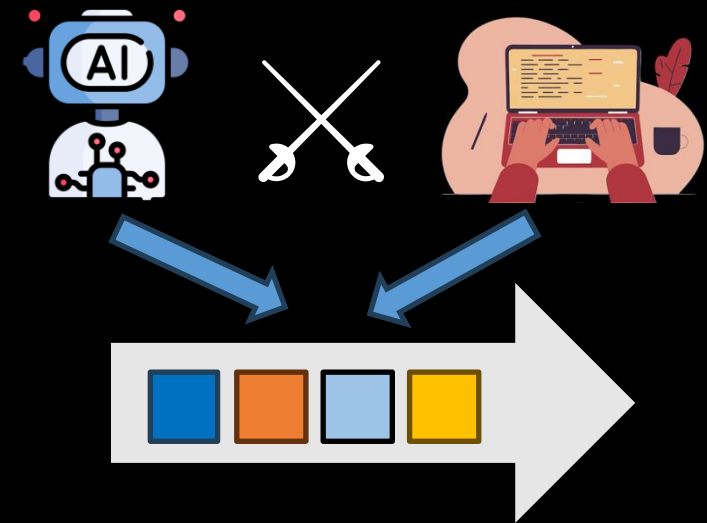
Testing (e.g., fraud txns)



Agents on Streams: Problems

Unfortunately, today's streaming systems fall short in enabling agentic use cases!

Performance Interference
Unsafe Writes
Inability to Explore Writes
Lack of Realistic Sandboxes



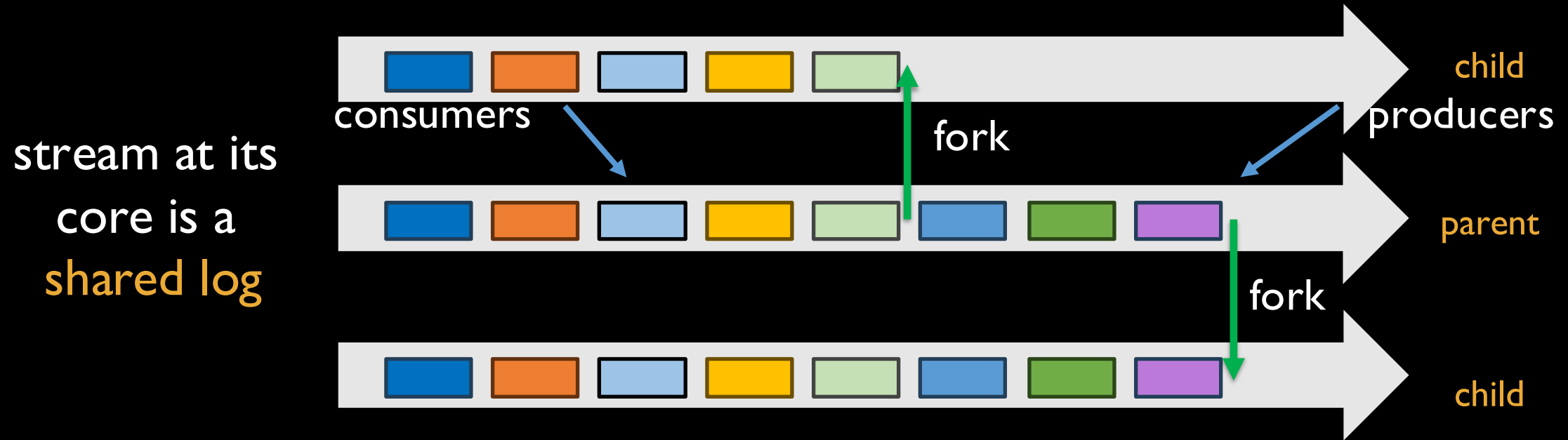
How to Solve These Problems?



AgileLog – rethinks the fundamental abstraction underlying streaming data to solve these problems

Bolt – an implementation of AgileLog, and some eval

A Forkable Shared Log is a Good Substrate for Agents



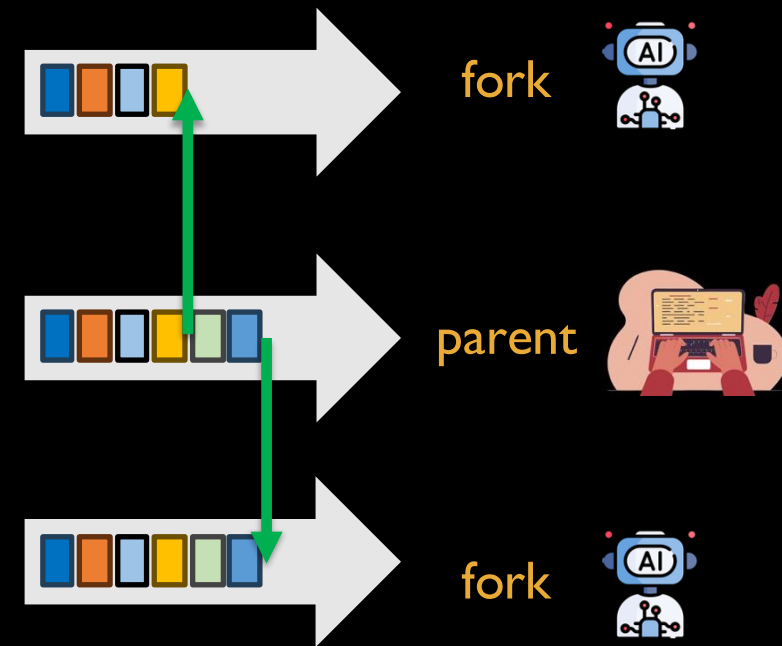
A shared log that offers **forking** as a core primitive
cheap, logically separate, performance-isolated *child* copies of itself

AgileLog is a new forkable shared log abstraction

AgileLog: A Good Substrate for Agents

Performance Isolation

traditional apps → parent log
each agent → own fork



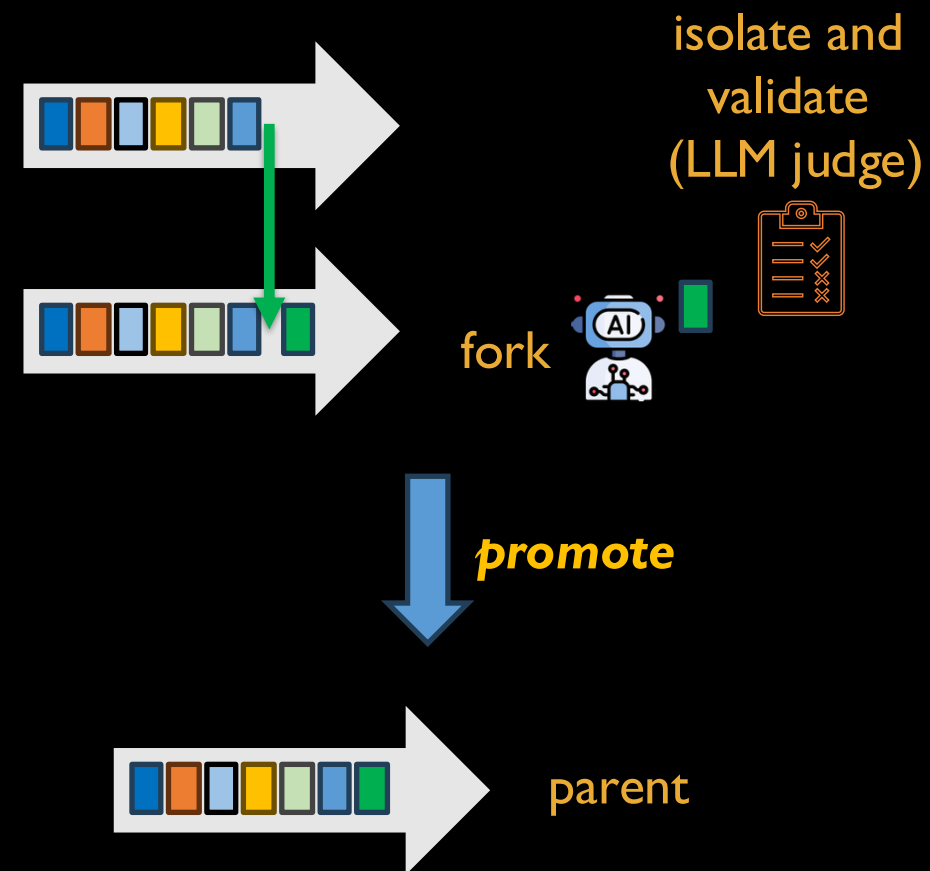
AgileLog: A Good Substrate for Agents

Performance Isolation

traditional apps → parent log
each agent → own fork

Handling Unsafe Writes

agentic writes isolated/validated within fork



AgileLog: A Good Substrate for Agents

Performance Isolation ✓

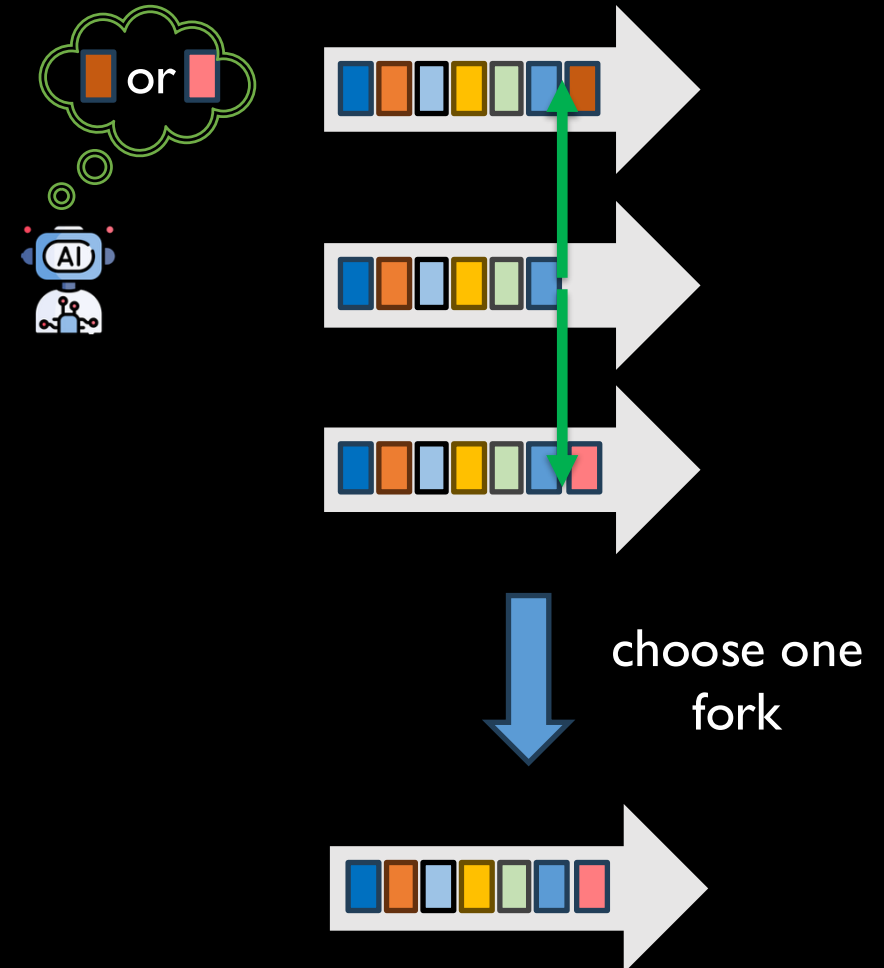
traditional apps → parent log
each agent → own fork

Handling Unsafe Writes ✓

agentic writes isolated/validated within fork

Handling Exploratory Writes ✓

agents create N forks and promote one fork



AgileLog: A Good Substrate for Agents

Performance Isolation ✓

traditional apps → parent log
each agent → own fork

Handling Unsafe Writes ✓

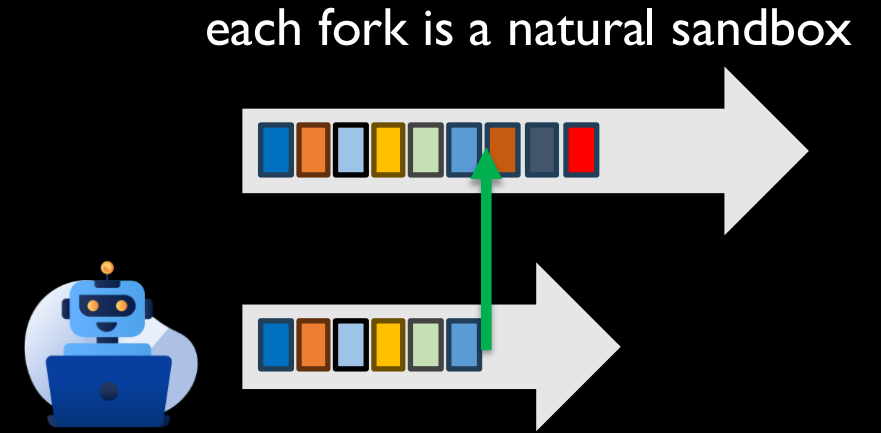
agentic writes isolated/validated within fork

Handling Exploratory Writes ✓

agents create N forks and promote one fork

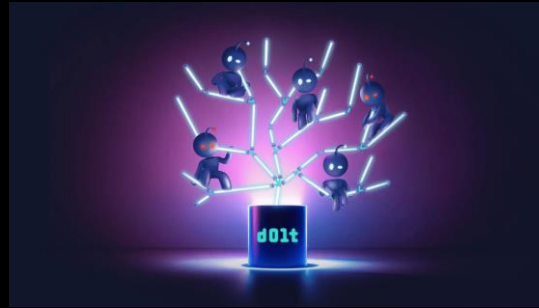
Realistic Sandboxes ✓

fork is a natural sandbox with real data



Forking or Branching is Not a New Idea!

OLTP Databases



Lakehouse



Block and Object Stores



Academic Works:

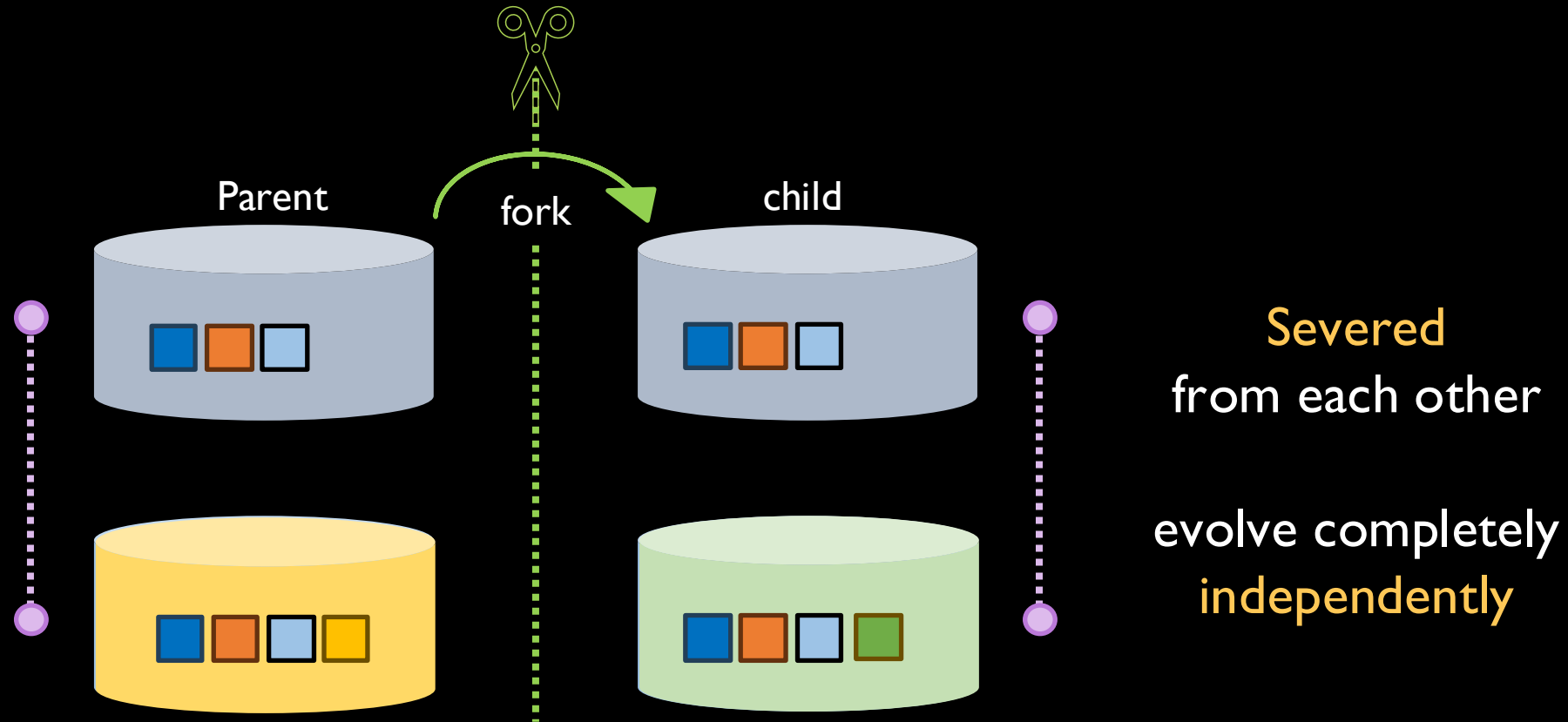
Olive (NSDI 2006)

Decibel (VLDB 2016)

OrpheusDB (VLDB 2017)

How is AgileLog Different

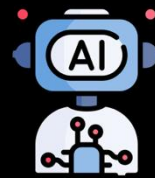
Traditionally, systems offer **severed** forks



AgileLog provides this usual severed forks
(or *sForks*) but...

How is AgileLog Different

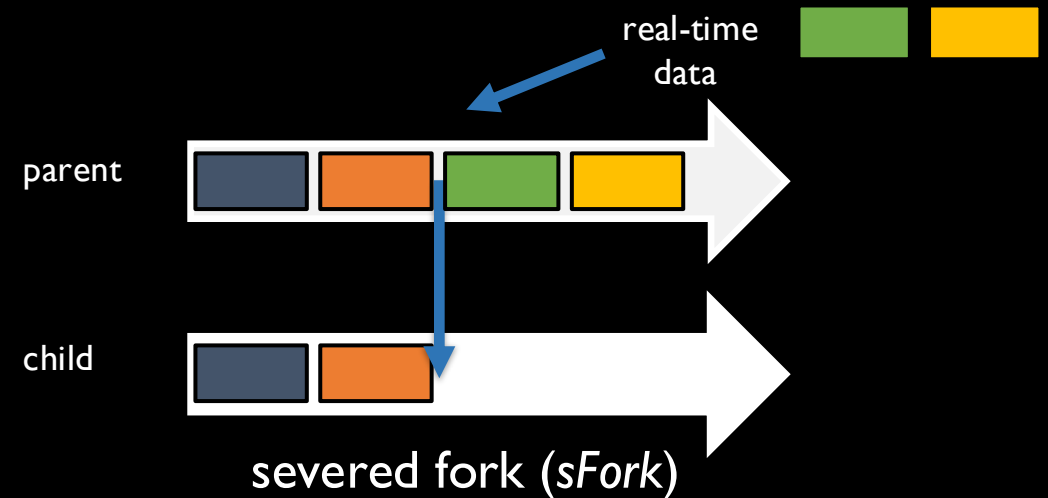
Severed forks are **insufficient** for **streaming data**



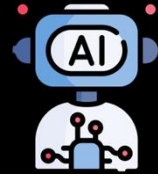
real-time analytics agent



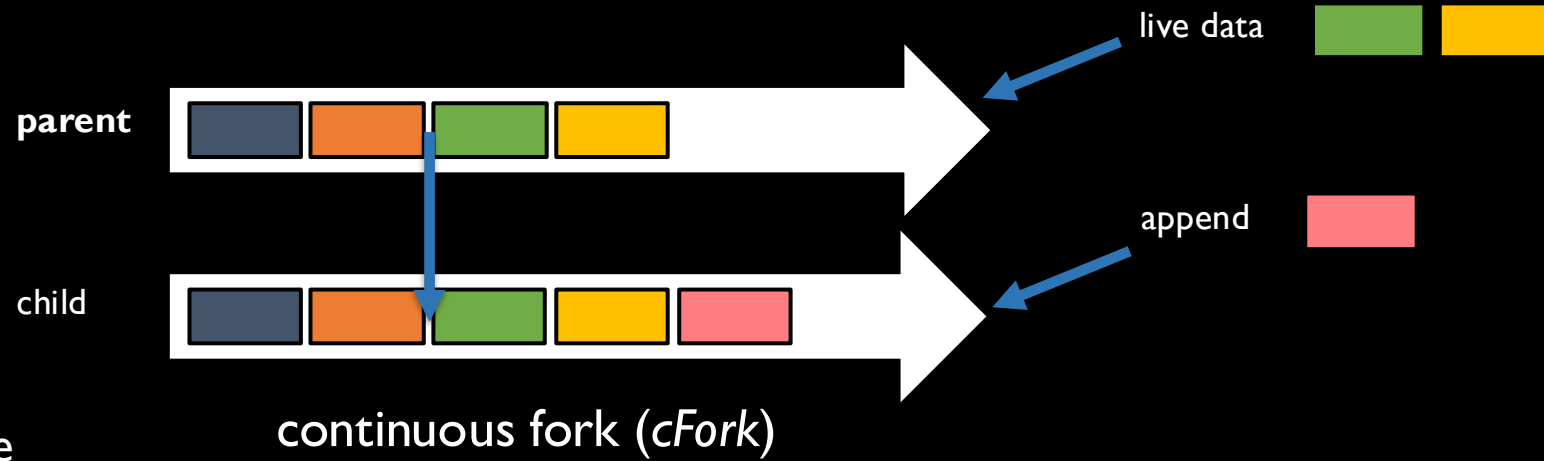
does not have the
real-time context/data



Continuous Forks



Has real-time
data & context!



Like traditional fork, inherits parent's history
But also, **continuously inherits** subsequent records

Writes on the cFork still remain isolated!

cFork: **unidirectional isolation**

sFork: bidirectional isolation

AgileLog Interface

```
interface AgileLog:  
    // Traditional shared-log API calls.  
    Position append(Record r);  
    List<Record> read(Position from, Position to);
```

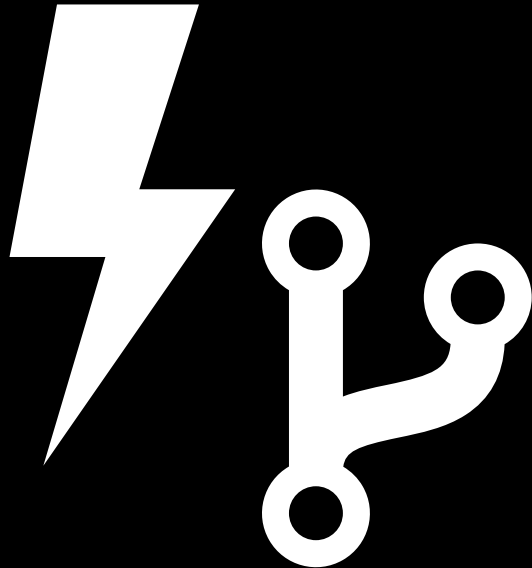
How to Solve These Problems?

AgileLog – rethinks the fundamental abstraction underlying streaming data to solve these problems

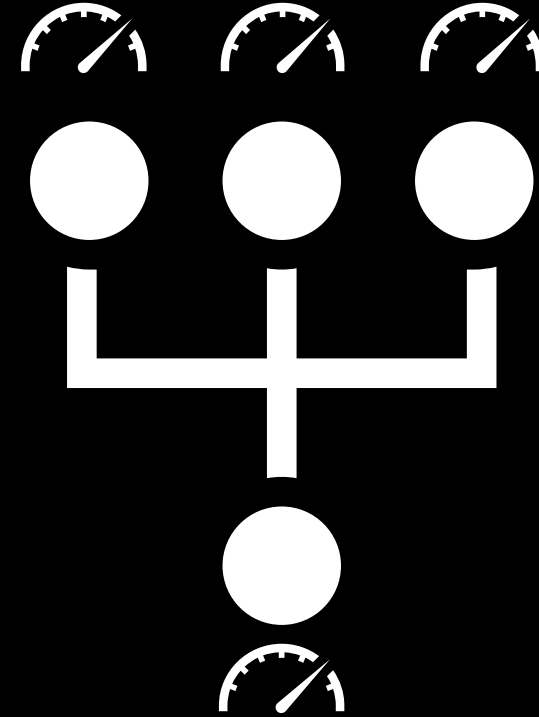


Bolt – an implementation of AgileLog, and some eval

Bolt: Goals



Forks creation must be quick

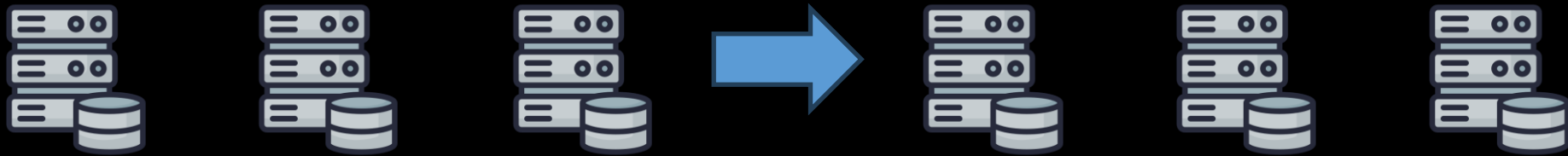


Fork must be perf-isolated
from other forks and parent

Conflicting Goals

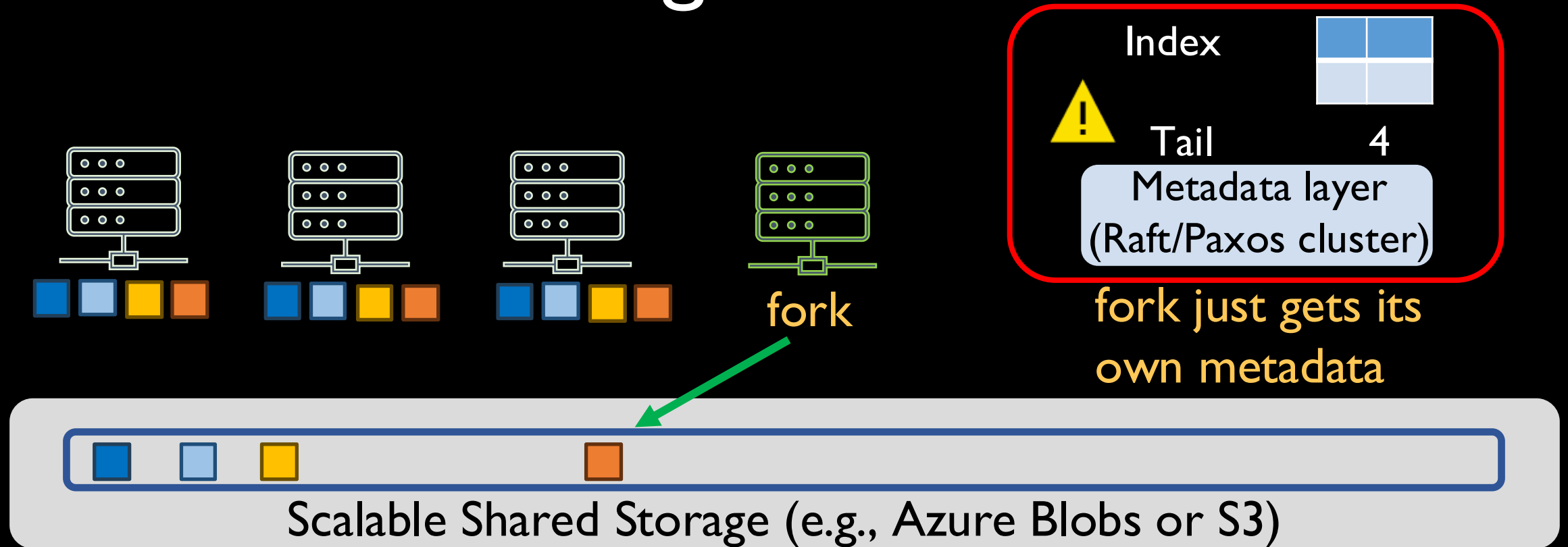


Host forks on the same servers as parent?
fast forks (no data copy) ✓ but interference ✗



Copy data to new cluster?
no interference ✓ but forks slow ✗

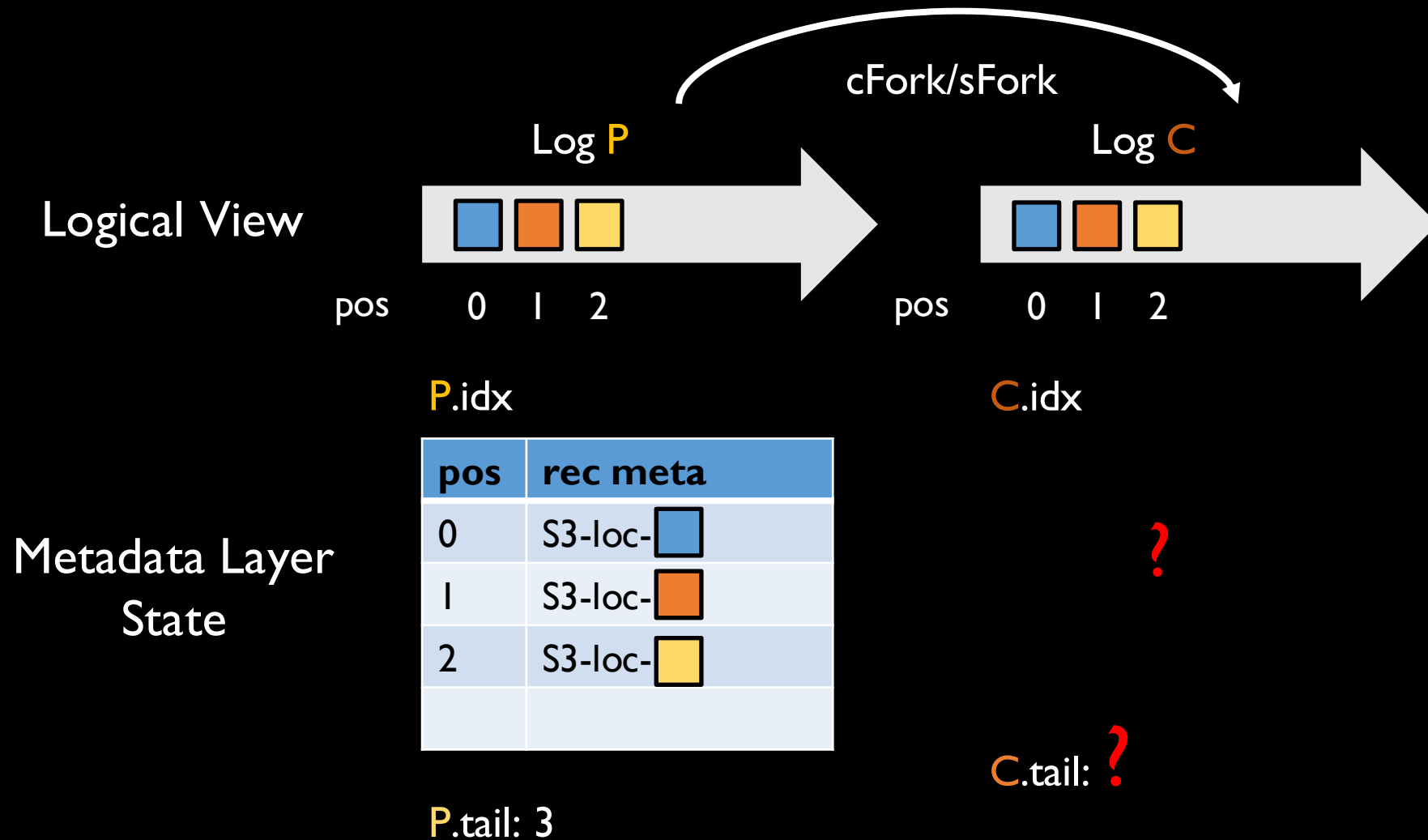
Architectural Insight: Go Diskless



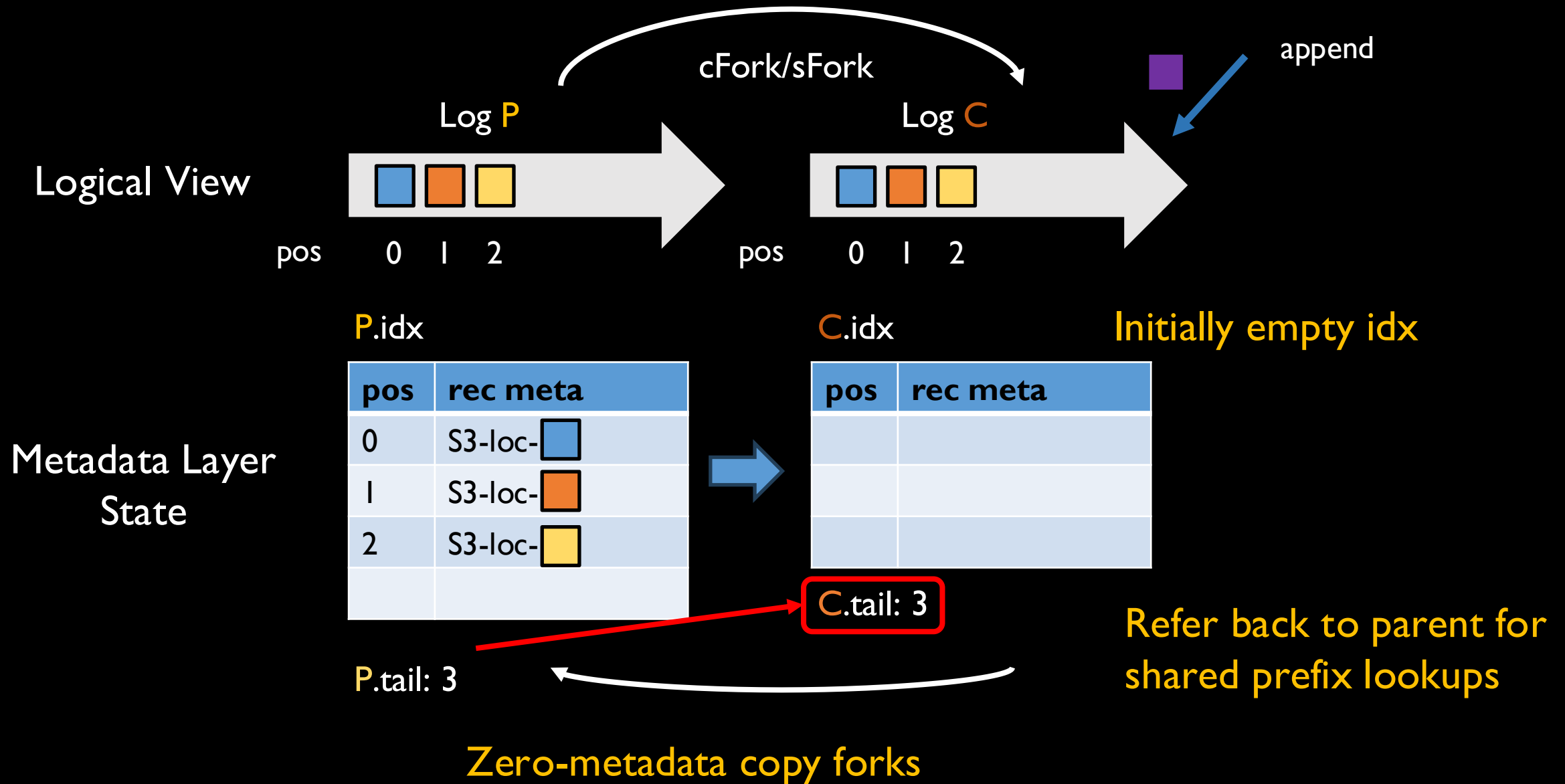
Fast forks: only metadata operation ✓

Isolation: diff compute end point + scalable storage layer ✓

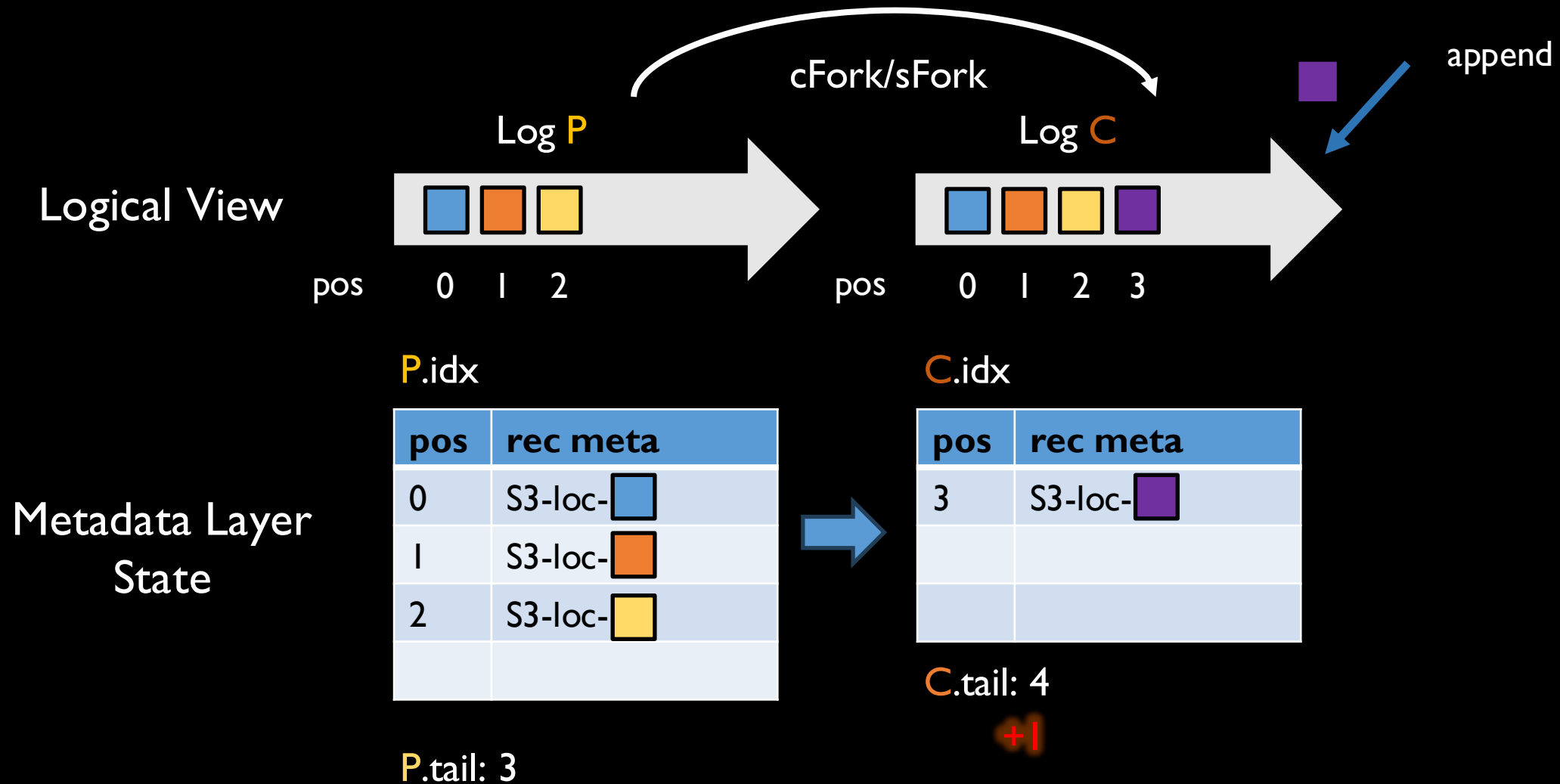
How to Create Forks?



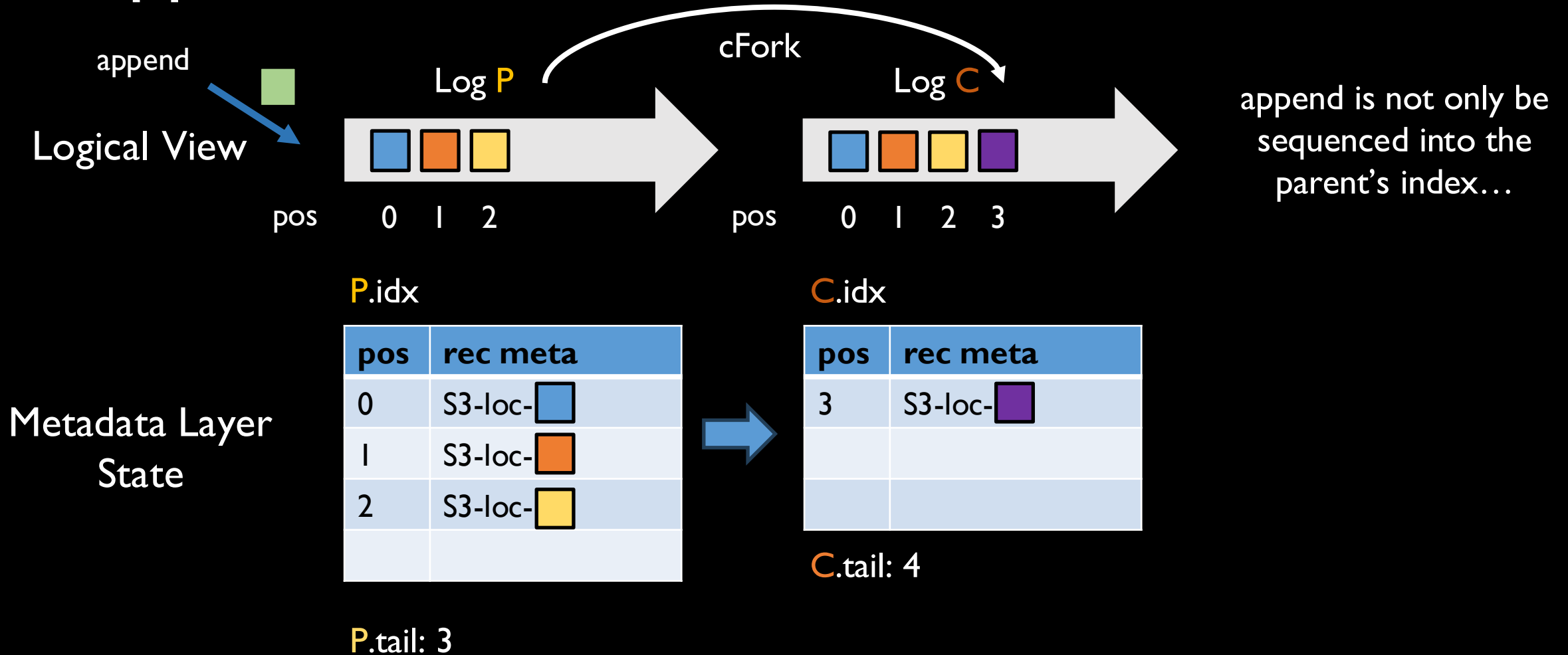
Efficient Zero-Metadata Copy Fork Creation



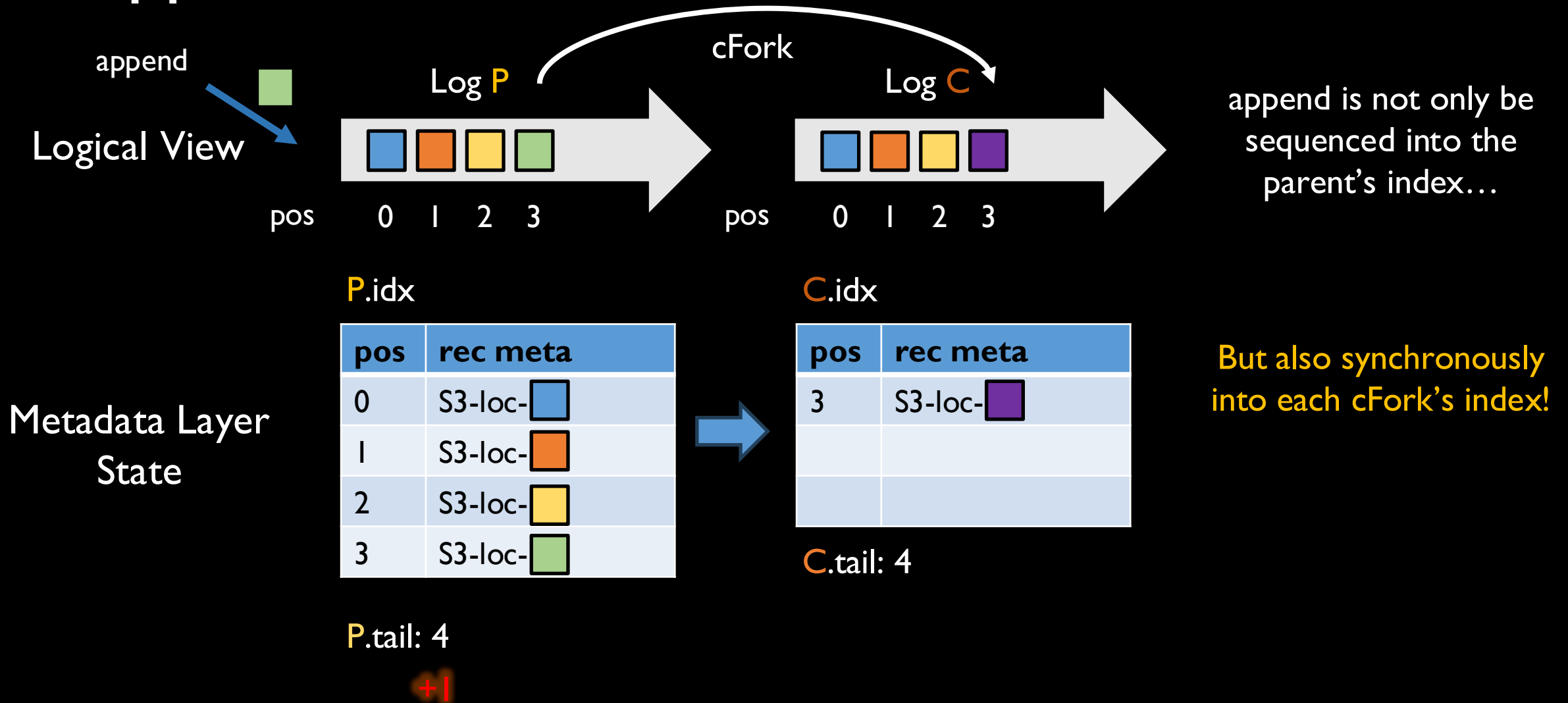
Efficient Zero-Metadata Copy Fork Creation



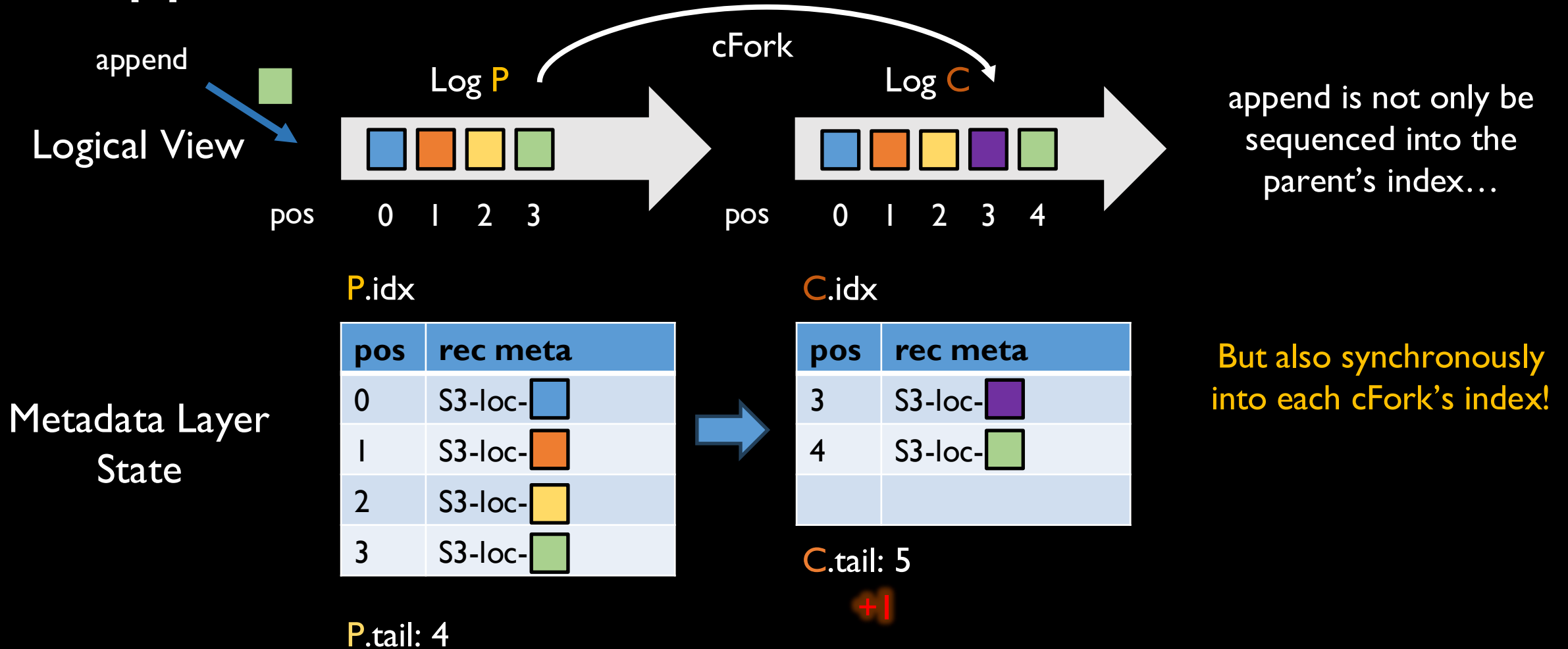
How to Implement cForks: A Naïve Approach



How to Implement cForks: A Naïve Approach



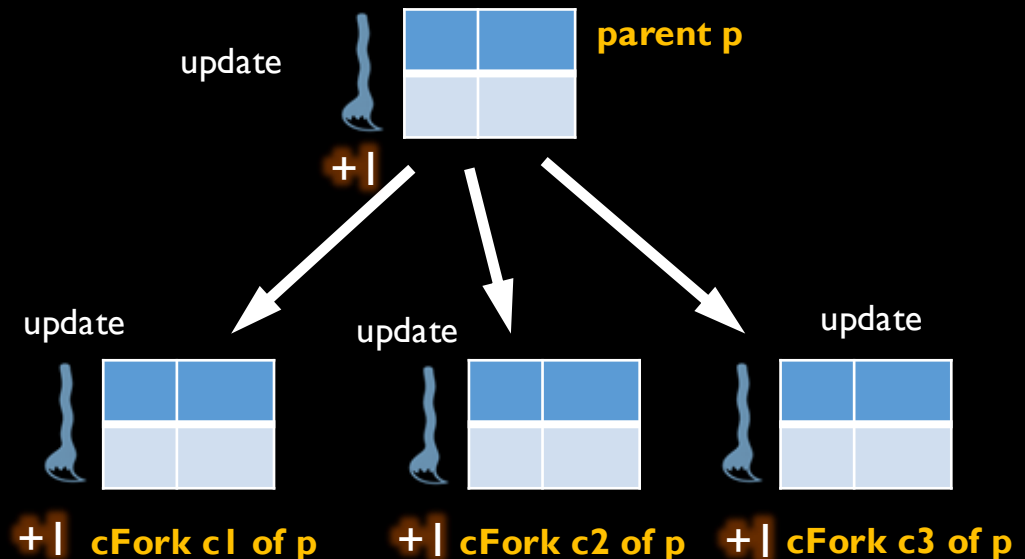
How to Implement cForks: A Naïve Approach



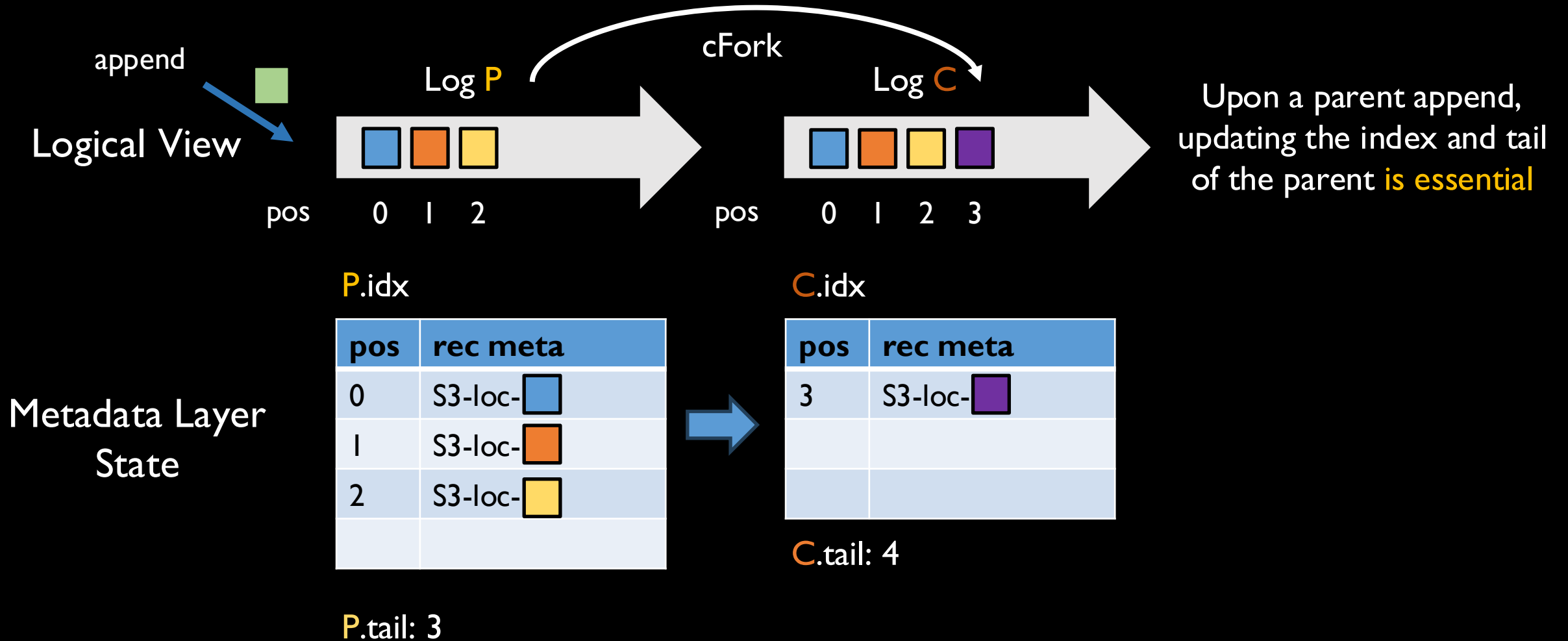
How to Implement cForks: A Naïve Approach

With wide branching and many many cForks synchronous index updates will

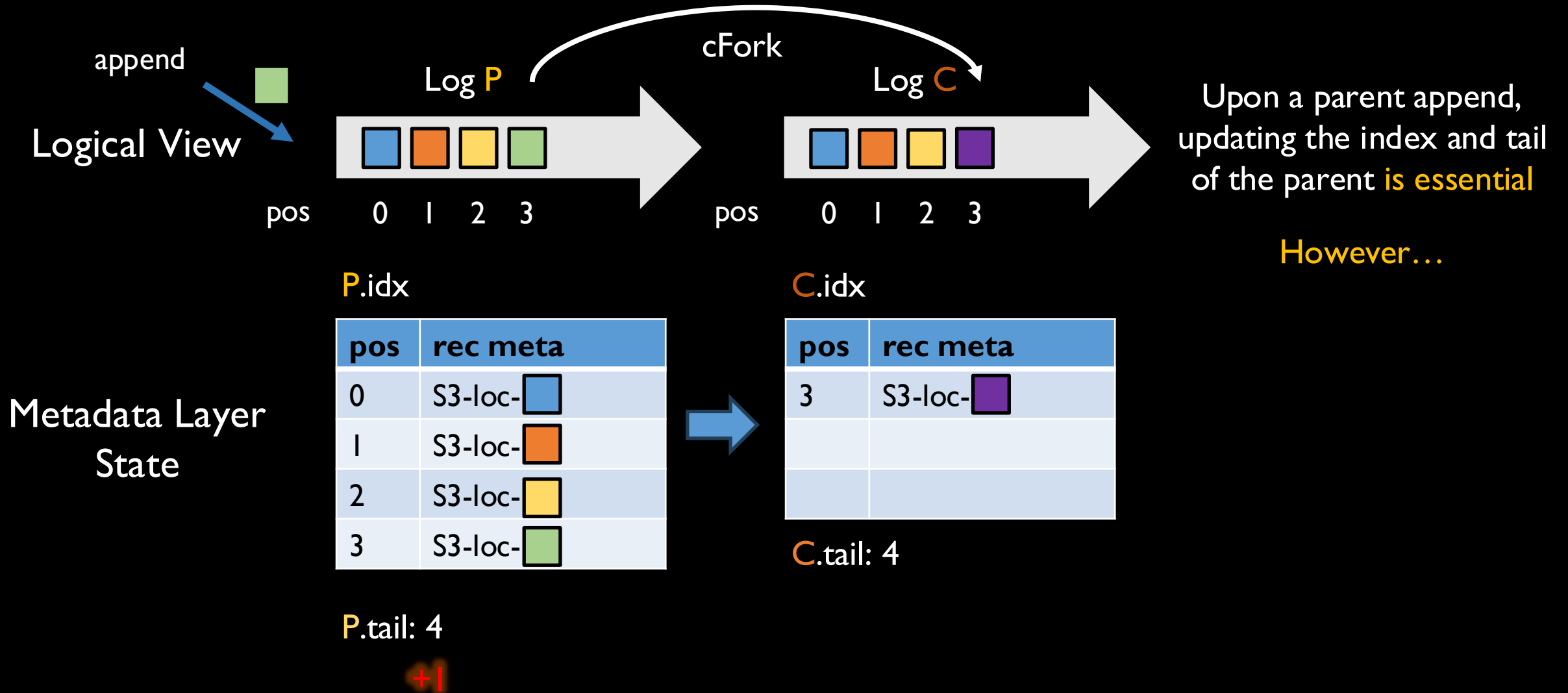
- slow down parent updates
- uses excessive metadata memory (redundant metadata across parents and children)



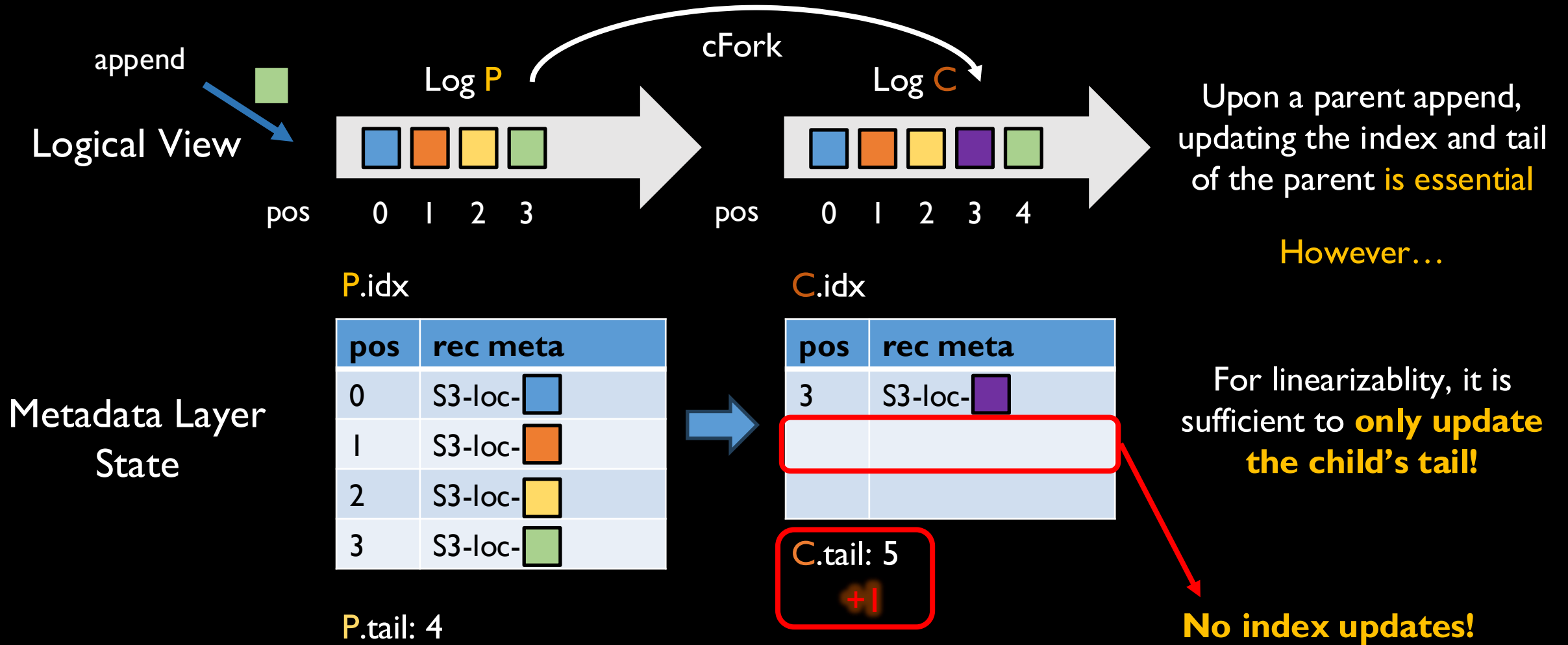
Efficient cForks with Tail-Only Updates



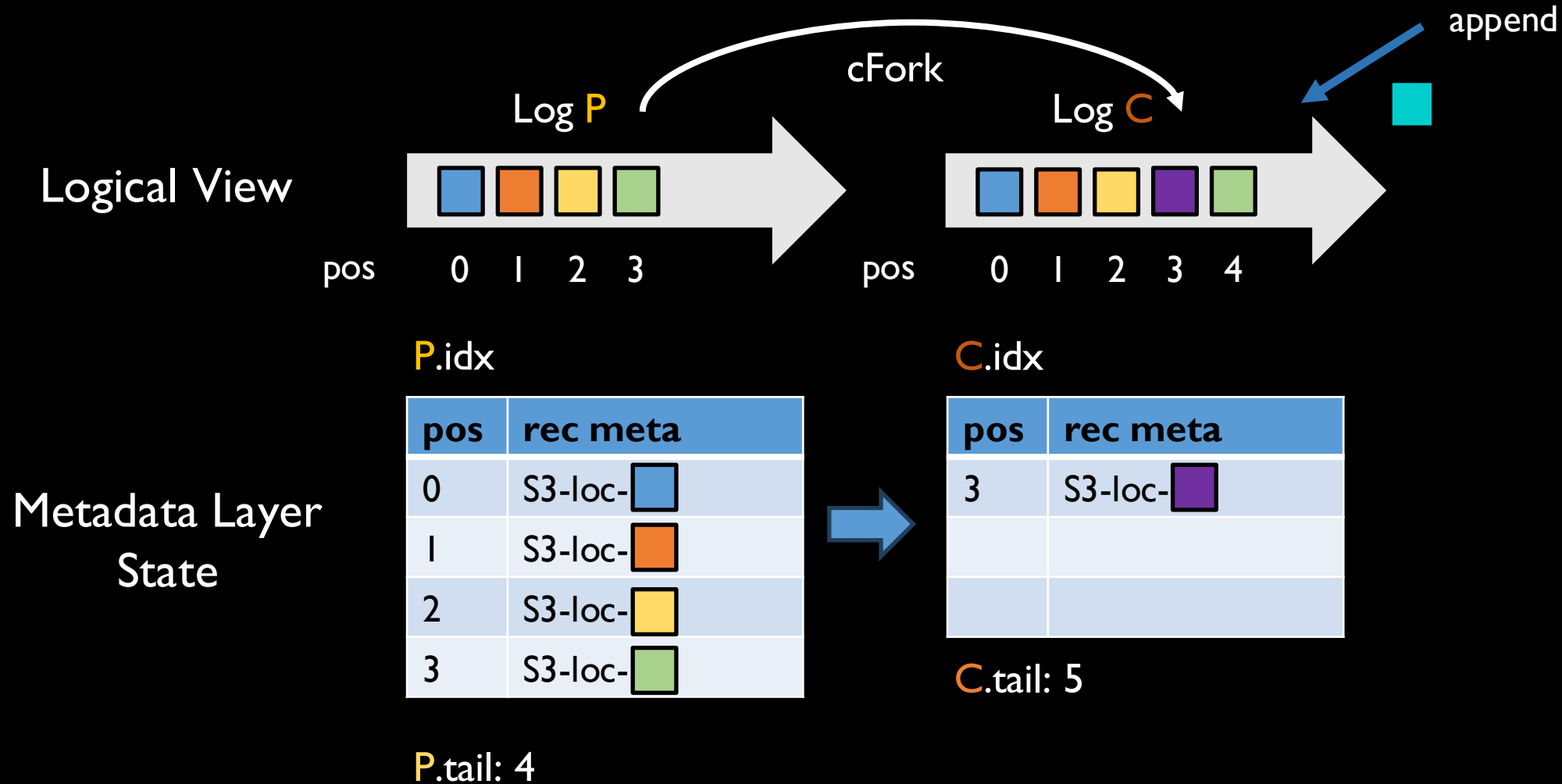
Efficient cForks with Tail-Only Updates



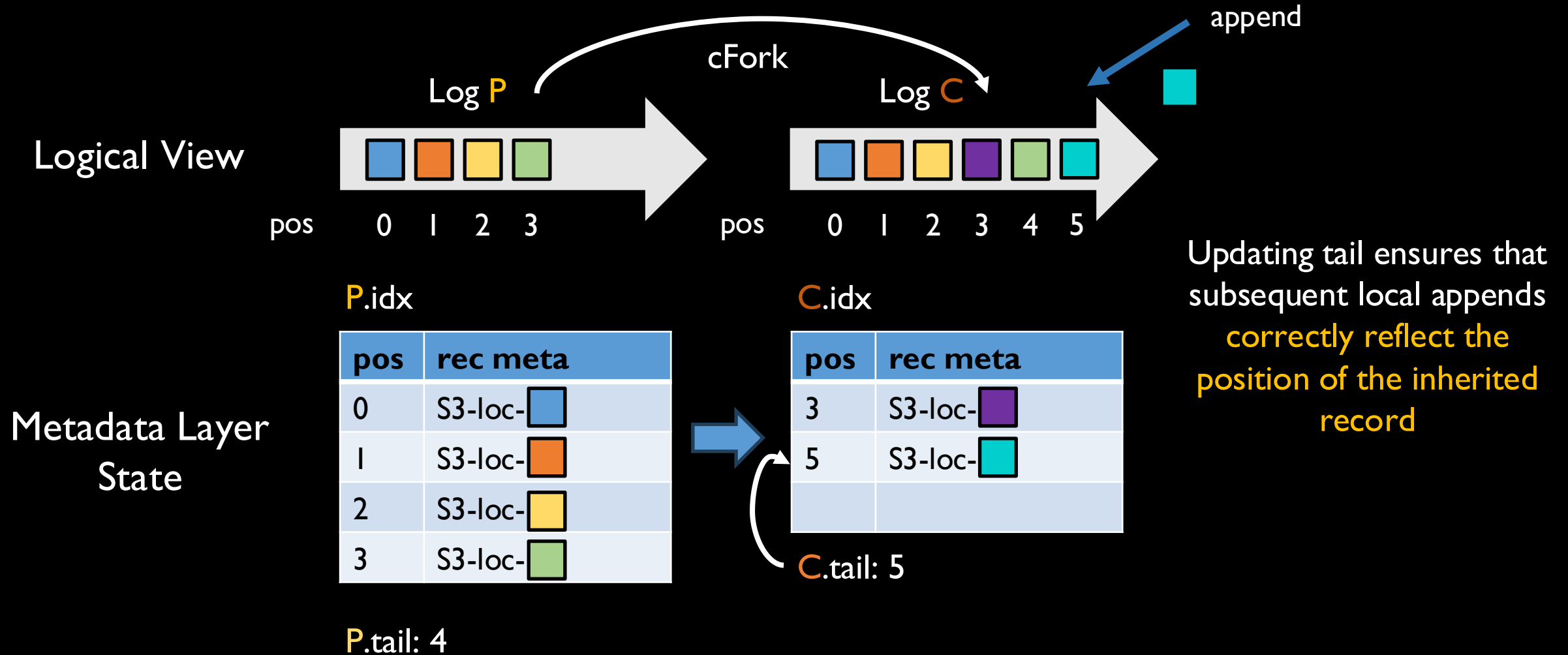
Efficient cForks with Tail-Only Updates



Efficient cForks with Tail-Only Updates



Efficient cForks with Tail-Only Updates



More Techniques in the Design!

Hierarchical log indexes for lookups

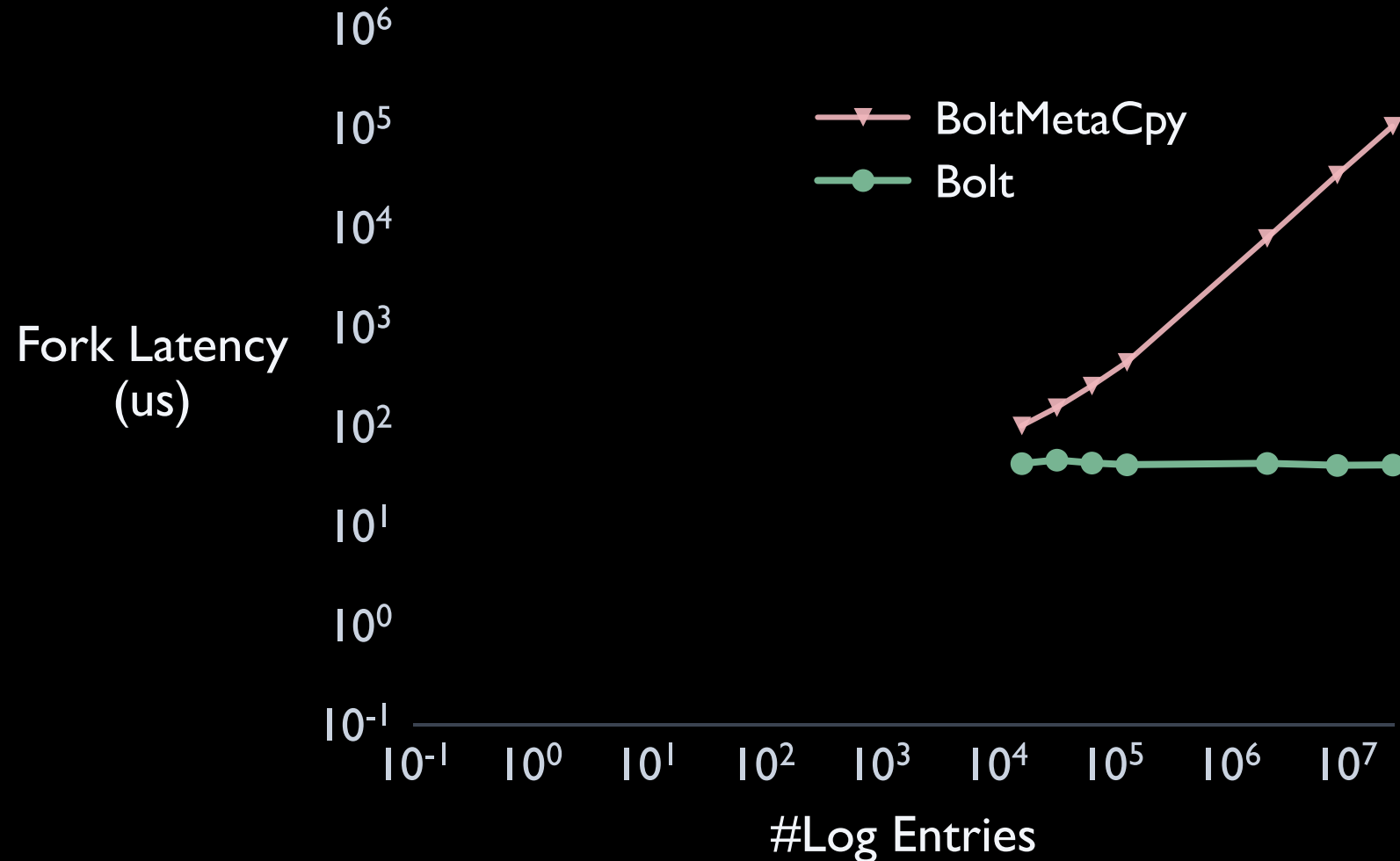
Lazy tail updates for cForks

Efficient promotes

Lease-based squashes to deal with rogue agents

Performance Benchmarks

Bolt Creates Forks in 10s of μ s

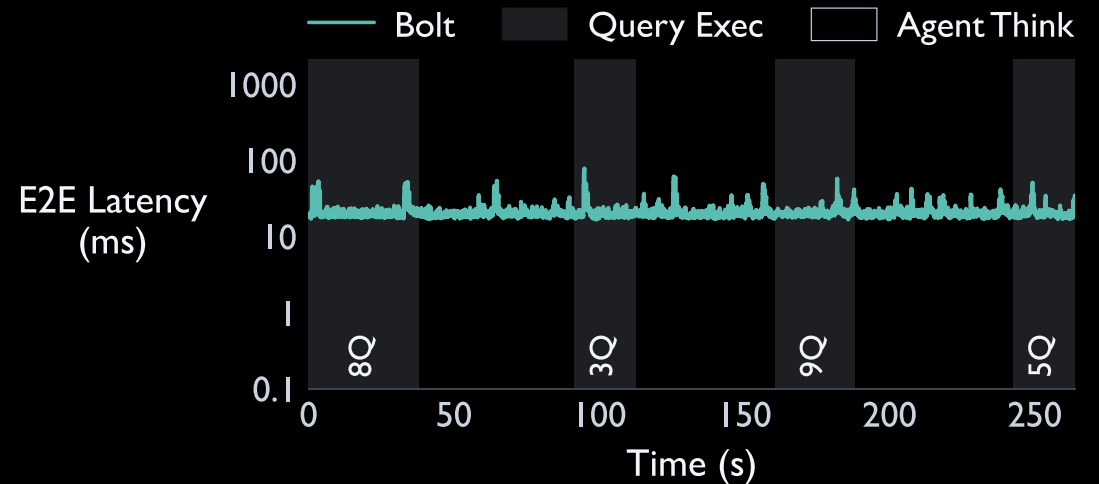
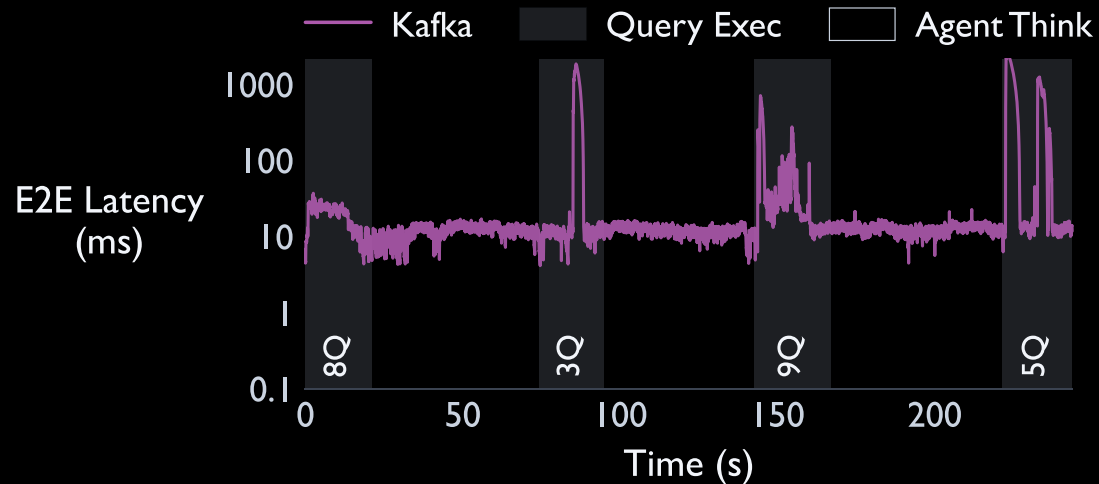


cForks Impose Minimal Overhead

Agentic Analytics No Longer Makes Production Slower

Workload: adhoc-query agent + production latency-sensitive workload

Measure latency of a traditional workload alongside the agent



Summary

AI agents are the users → rethink data systems in this lens!

AgileLog is a step in this direction for live, streaming data

Forking is a key enabler; continuous forks are necessary for real-time data

Bolt builds atop a diskless architecture to realize forks



Shreesha G.
Bhat



Tony
Hong



Michael
Noguera



Aishwarya
Ganesan



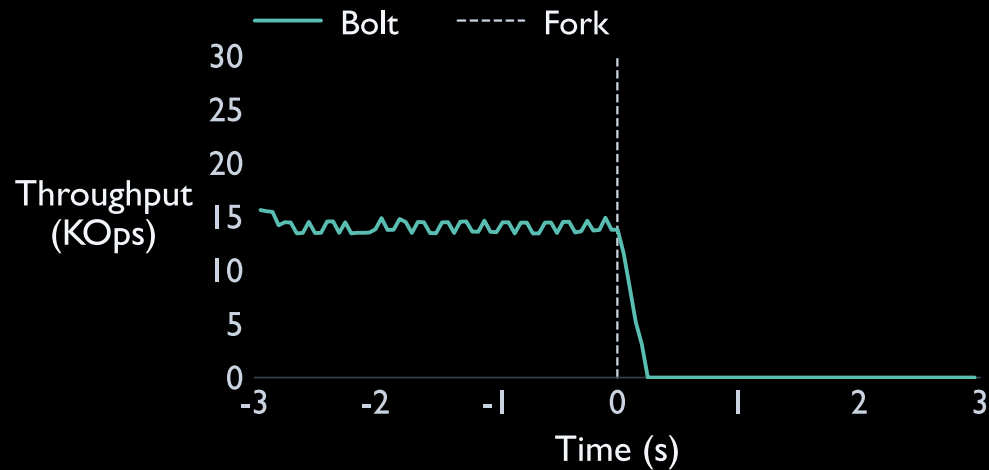
Ram
Alagappan

Artifact
Link:

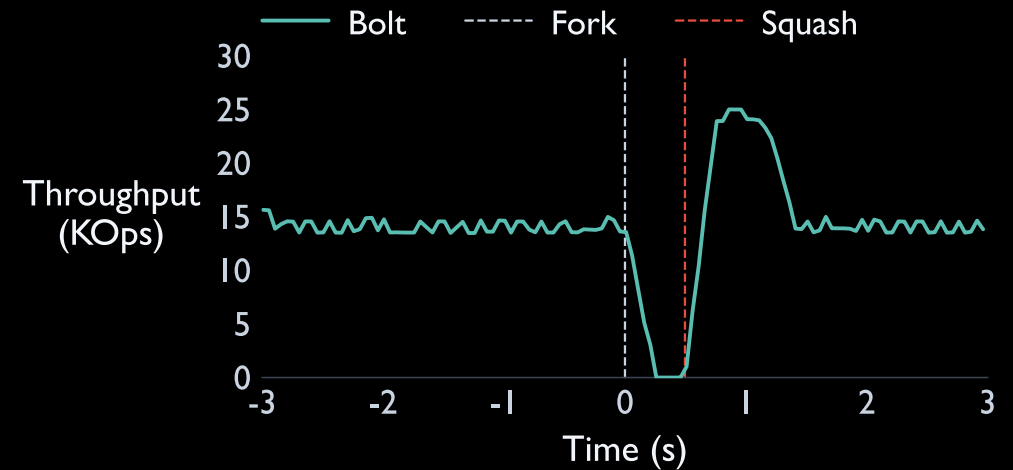


Backup

Dealing with Rogue Agents: Leases

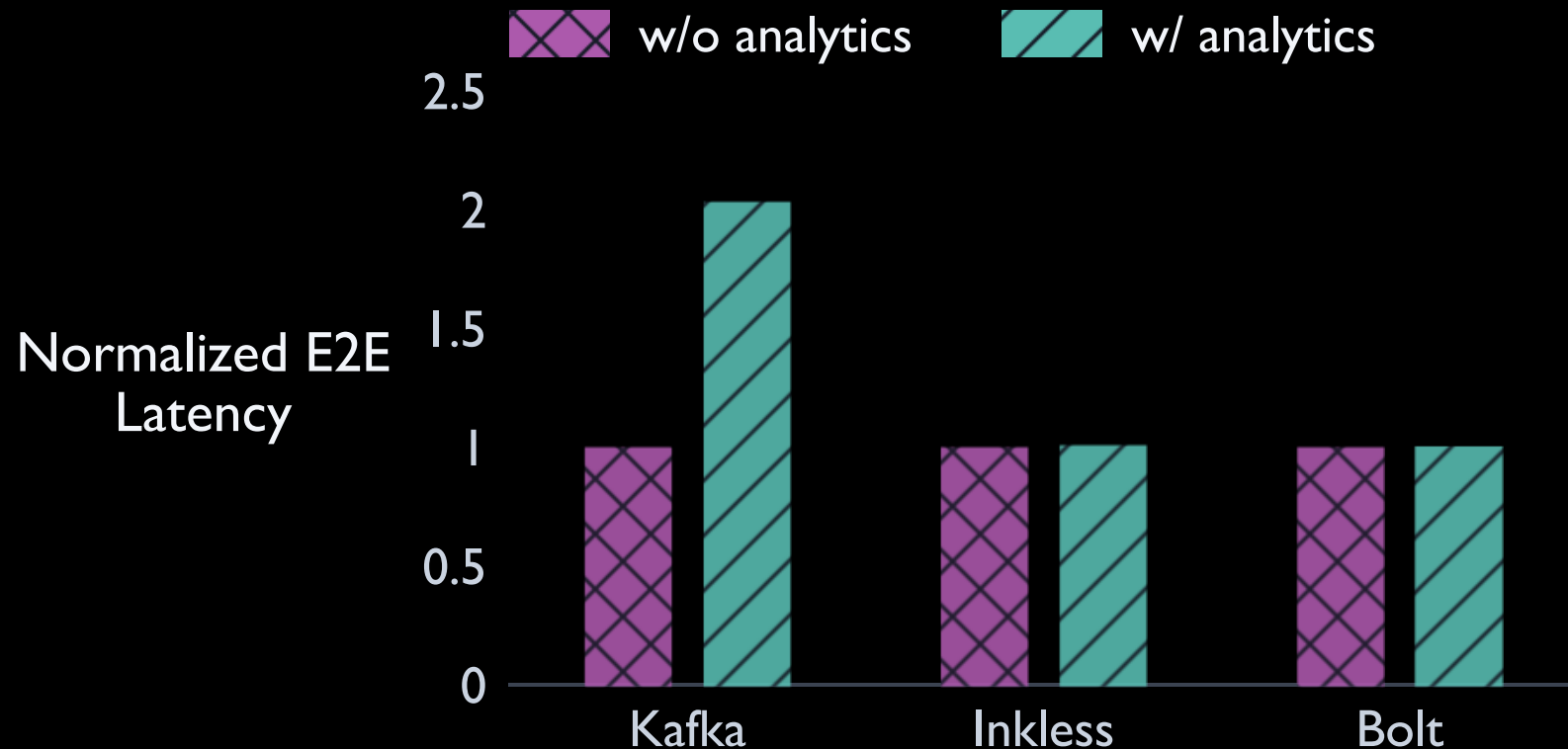


w/o Internal Lease-based Squash



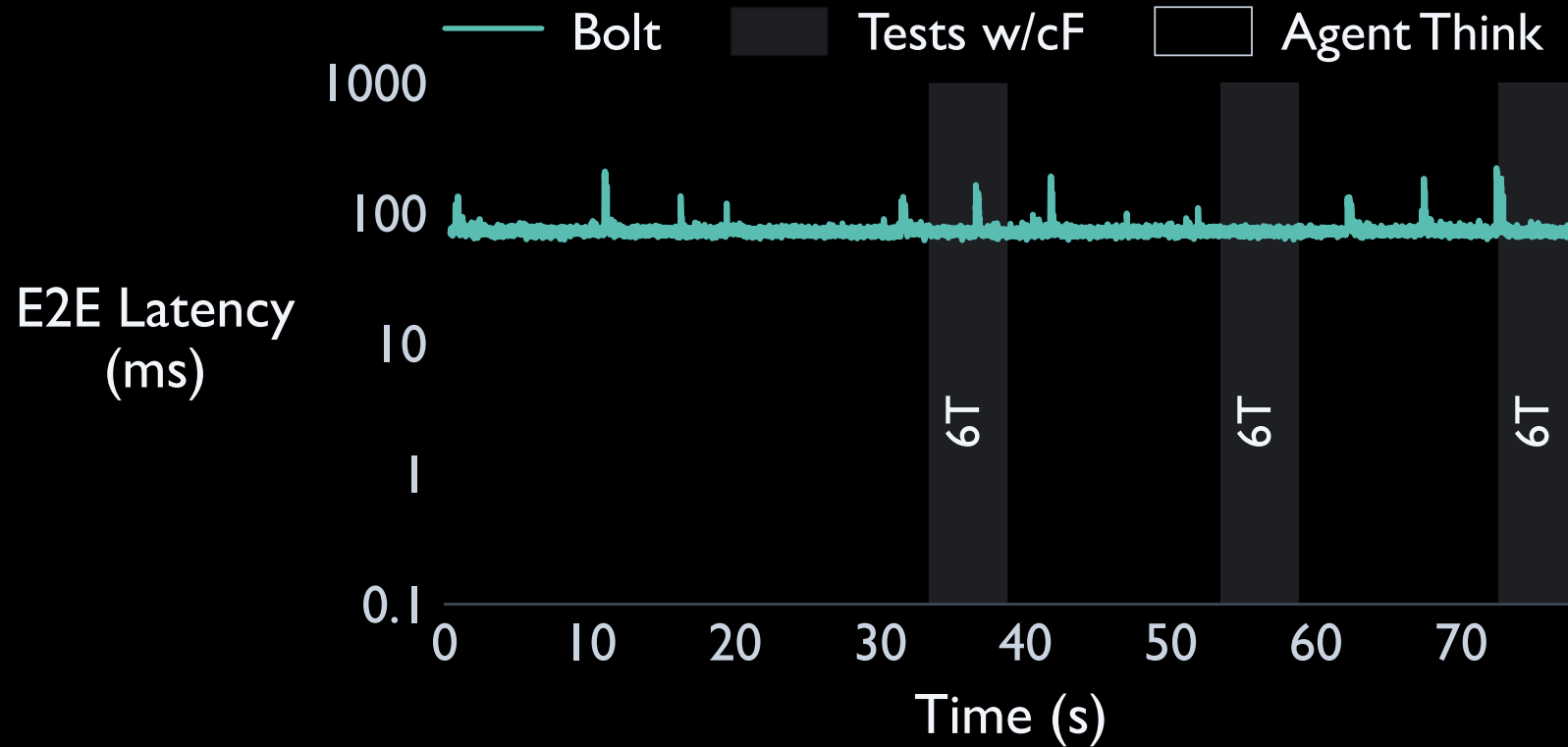
with Internal Lease-based Squash

Comparison to Other Diskless Architectures



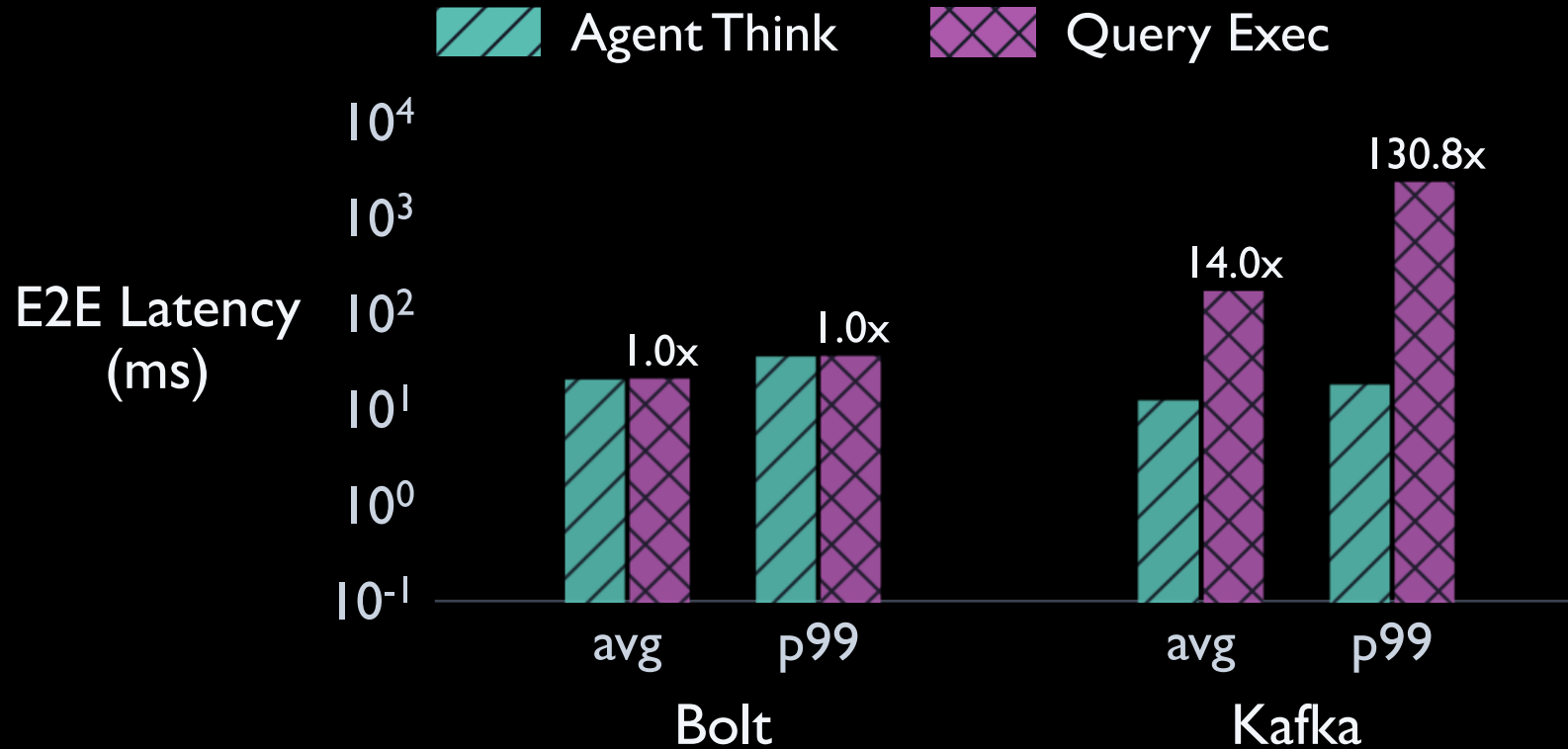
Inkless and other diskless shared logs can mimic read-only cForks, but cannot handle writes

Stream Testing Agent

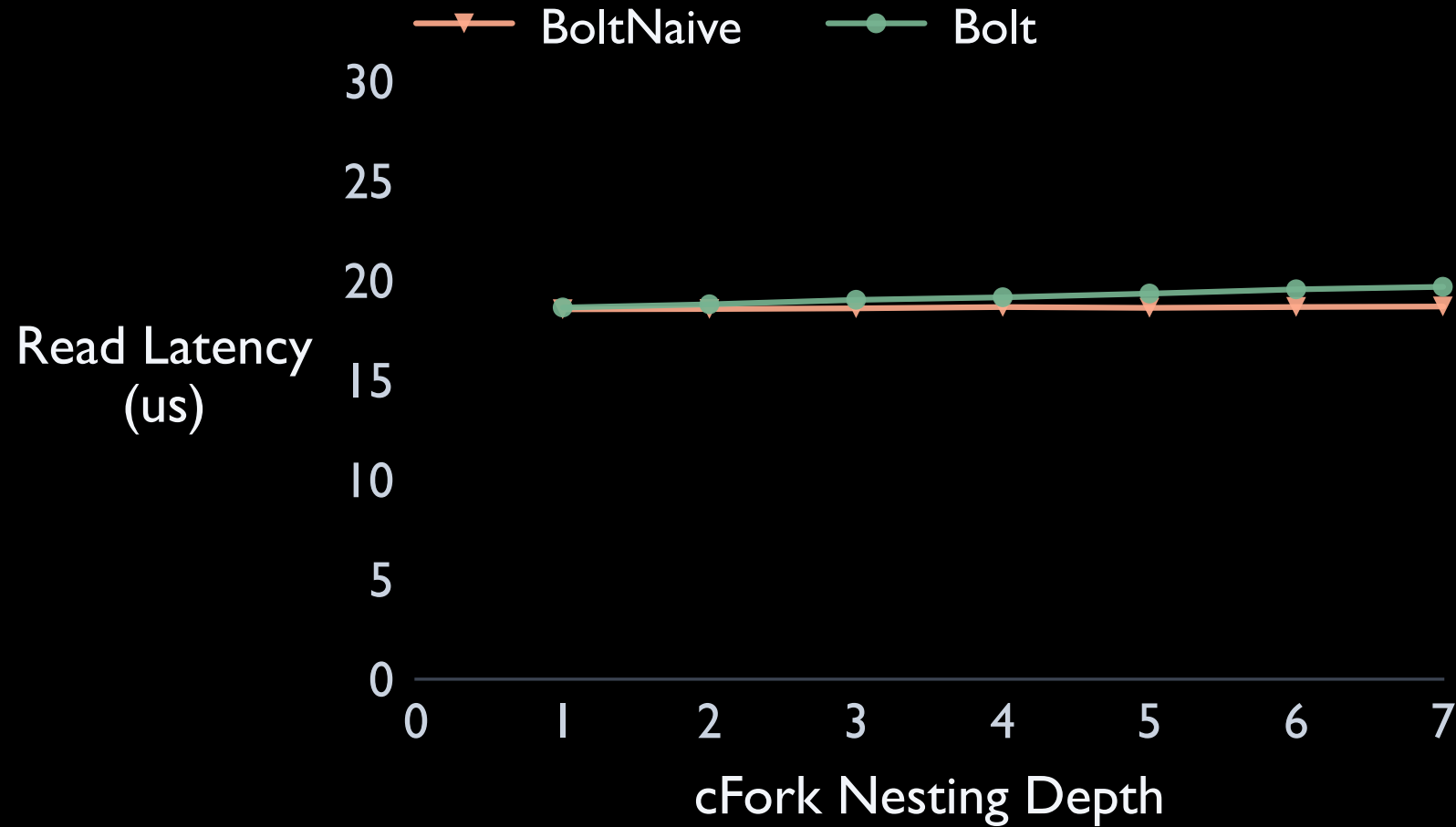


Tests are exercising edge case behavior, but no impact to production

Per-phase Distributions for Analytics Agent

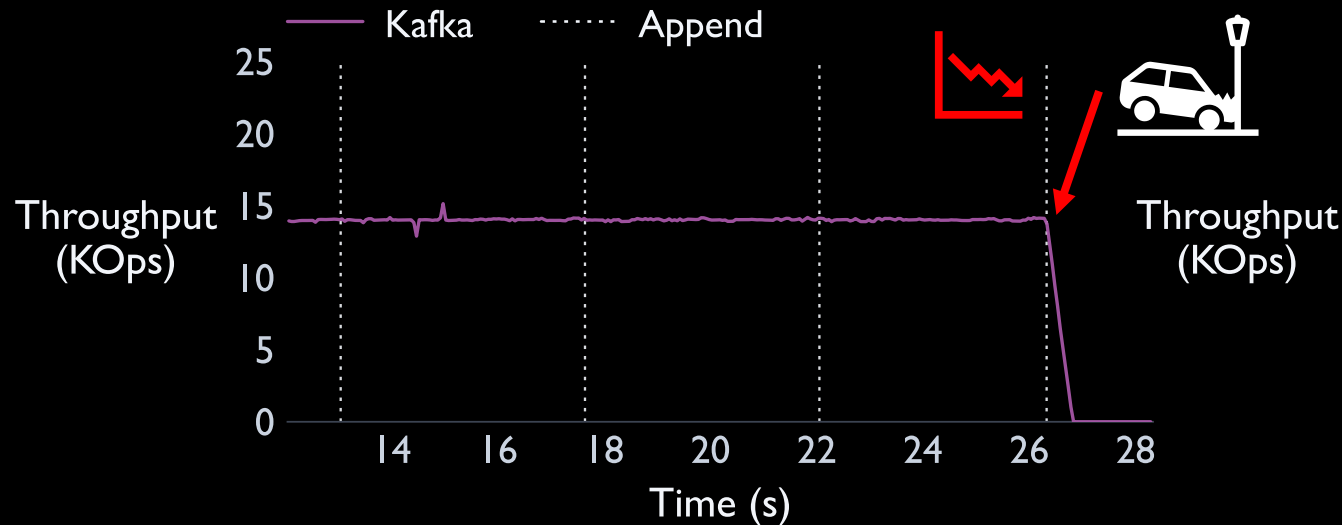


Metadata Query Latency with Deep Forks

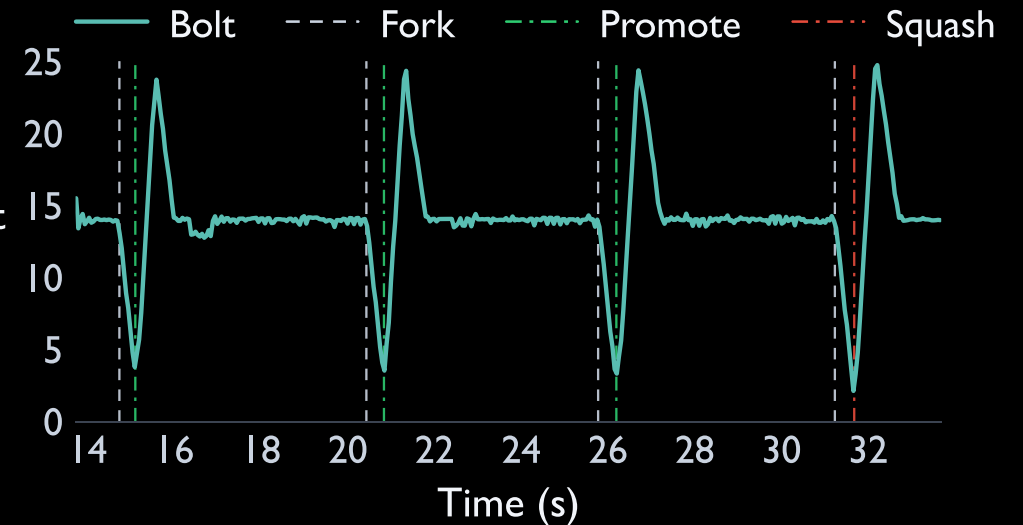


Supply Chain Restock Agent

Inject a schema error to simulate an agent mistake



Direct Agent Writes with Kafka



Isolated Writes with cFork and Promote

Impact of Complex Nested Forks

Various nesting hierarchies
at each point

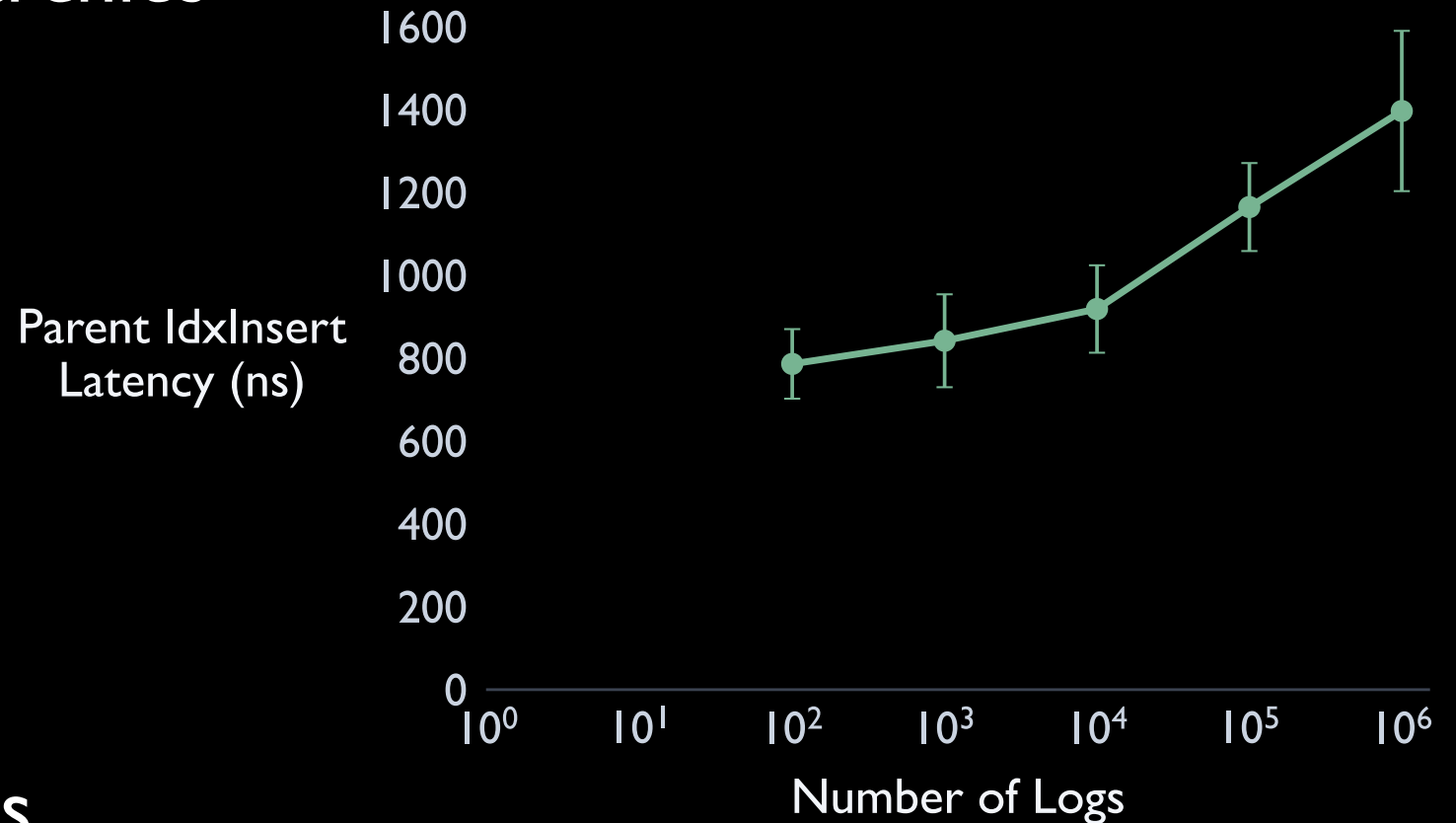
Linear (deep forks)

Star (wide forks)

Random

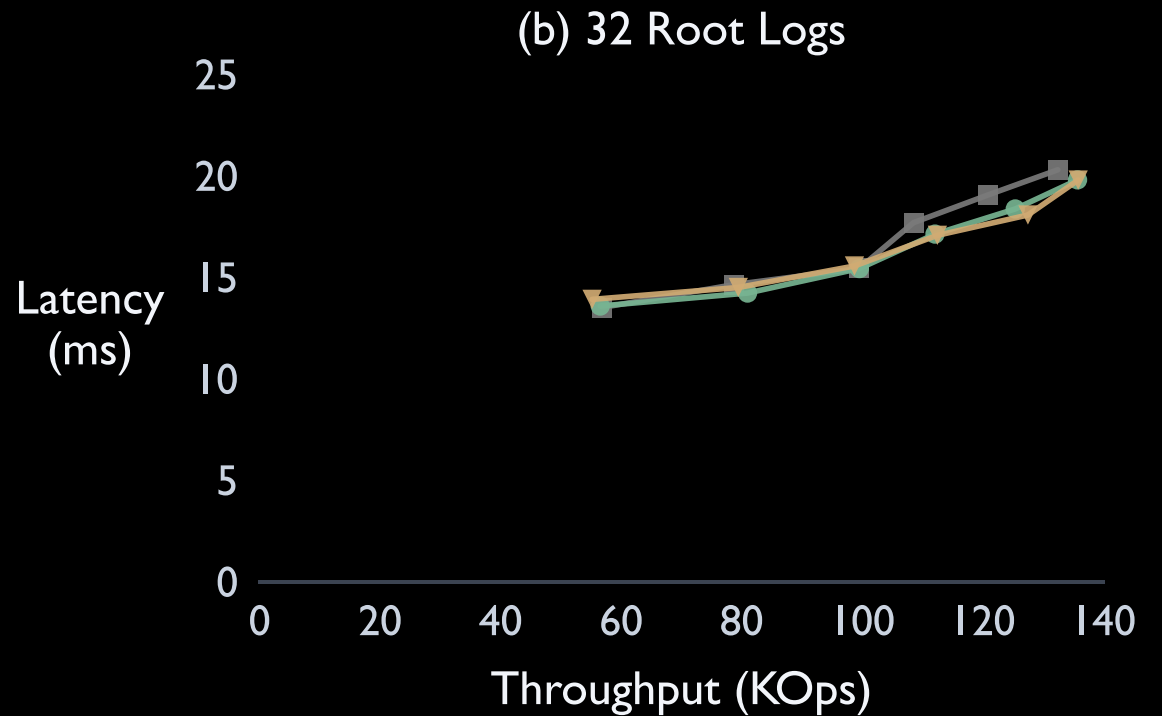
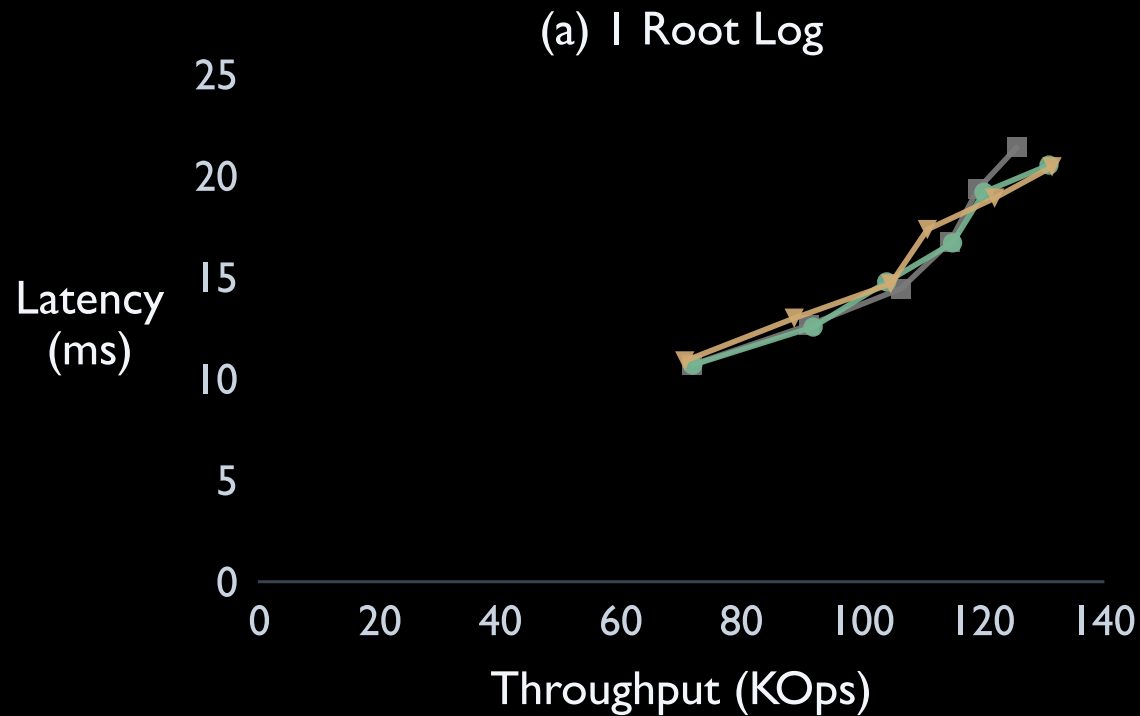
Balanced

Variance under 400ns

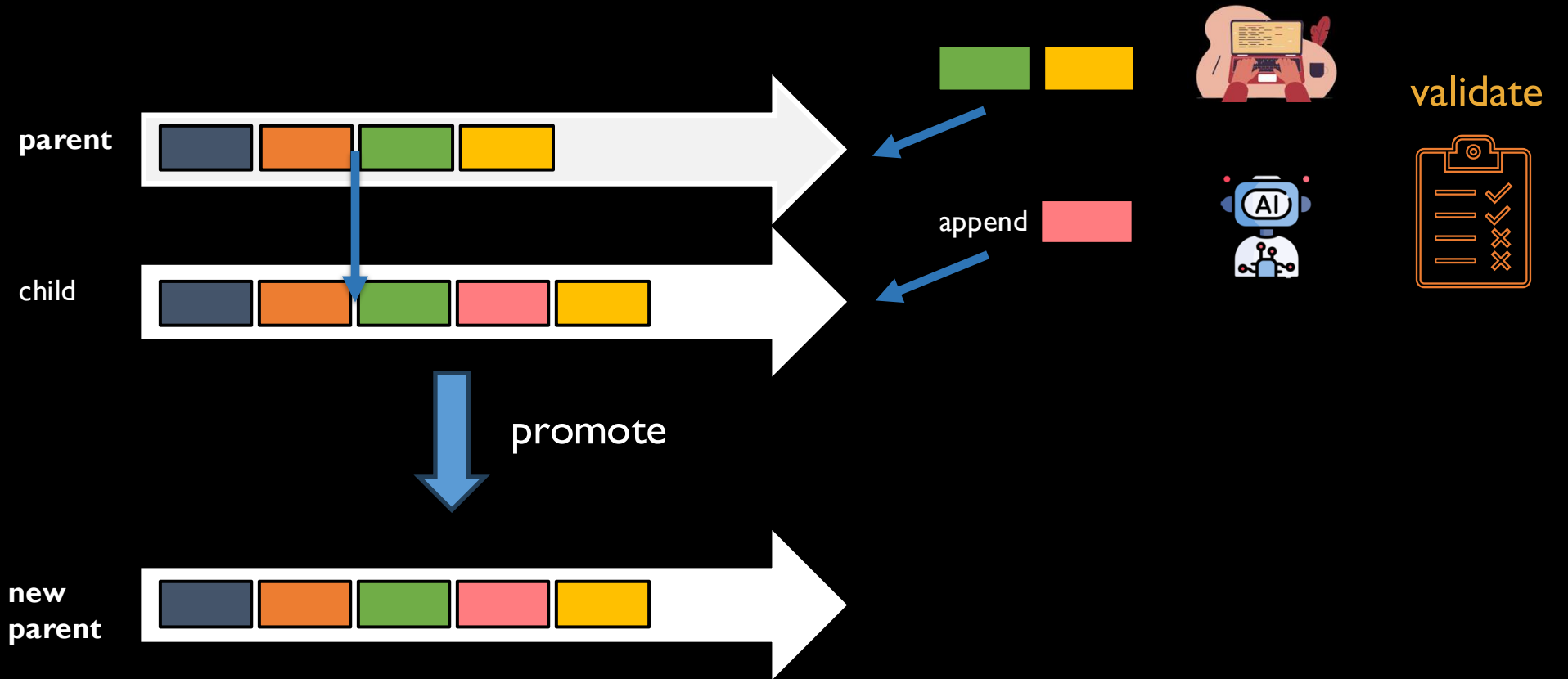


Parent Performance Impact E2E

—■— 0 cForks —●— 10 cForks —▼— 100 cForks



cForks + Promote



The cFork interleaves agentic and non-agentic records in a **linearizable order**

Agents on Streams: Problems

Performance Interference

Aren't agents "just another client"?

No! Highly **speculative** and **exploratory** [1, 2, 3]

try many paths to accomplish task: partial attempts at task, suboptimal solutions, validation of steps

Interferes with the performance of traditional workloads on the same infra

Requirement: Performance Isolation

[1] Liu et al. Supporting Our AI Overlords: Redesigning Data Systems to be Agent-First. arXiv:2509.00997, 2025.

[2] Giurgiu Nidd. Supporting Dynamic Agentic Workloads: How Data and Agents Interact. arXiv preprint arXiv:2512.09548, 2025.

[3] Parameswaran et al. <https://bair.berkeley.edu/blog/2026/07/07/intelligence-is-free-now-what/>

Agents on Streams: Problems

Unsafe Writes

Direct writes can be catastrophic

LLMs can generate incorrect actions, prone to write **malformed** or **incorrect records** [1, 2]

Can cause traditional consumers to crash or hang!

Requirement: A way to isolate and validate writes before they take effect

[1] Liu et al. Supporting Our AI Overlords: Redesigning Data Systems to be Agent-First. arXiv:2509.00997, 2025.

[2] Safe Agent Writes in Workbench Demo. <https://www.dolthub.com/blog/2025-11-25-safe-agent-writes-video/>

Agents on Streams: Problems

Inability to Explore Writes

Agents may explore different paths to write to streams

Examples:

- different sub-agents building hypothetical parallel worlds

- try different what-if analyses in parallel

Today's systems expose one history, unable to represent multiple futures

Requirement: The ability to explore writes

Agents on Streams: Problems

Lack of Realistic Sandboxes

Testing agents need to inject events that must be isolated from the production streams

However, agents can effectively test only with real data

Crucial in streaming where **temporal context is critical**

example: fraudulent txn interleaved with real txns

Requirement: A way to create cheap, realistic sandboxes