Mach
High Performance Observability

$ whoami

- Franco Solleza
  - franco_solleza@brown.edu
  - Office: CIT 345
- Previous life:
  - BSc. Economics 2011
  - Worked for 8 years in not-tech
- What do I do outside of PhDing?



$ whoami

Research Interest in one (long-ish) sentence:

I want to ***observe*** data intensive tasks, identify ***problems*** in these tasks, design ***abstractions*** to solve these problems, and ***systems*** to support these abstractions.

A Terrible, Horrible, No-Good, Very Bad Day at Slack

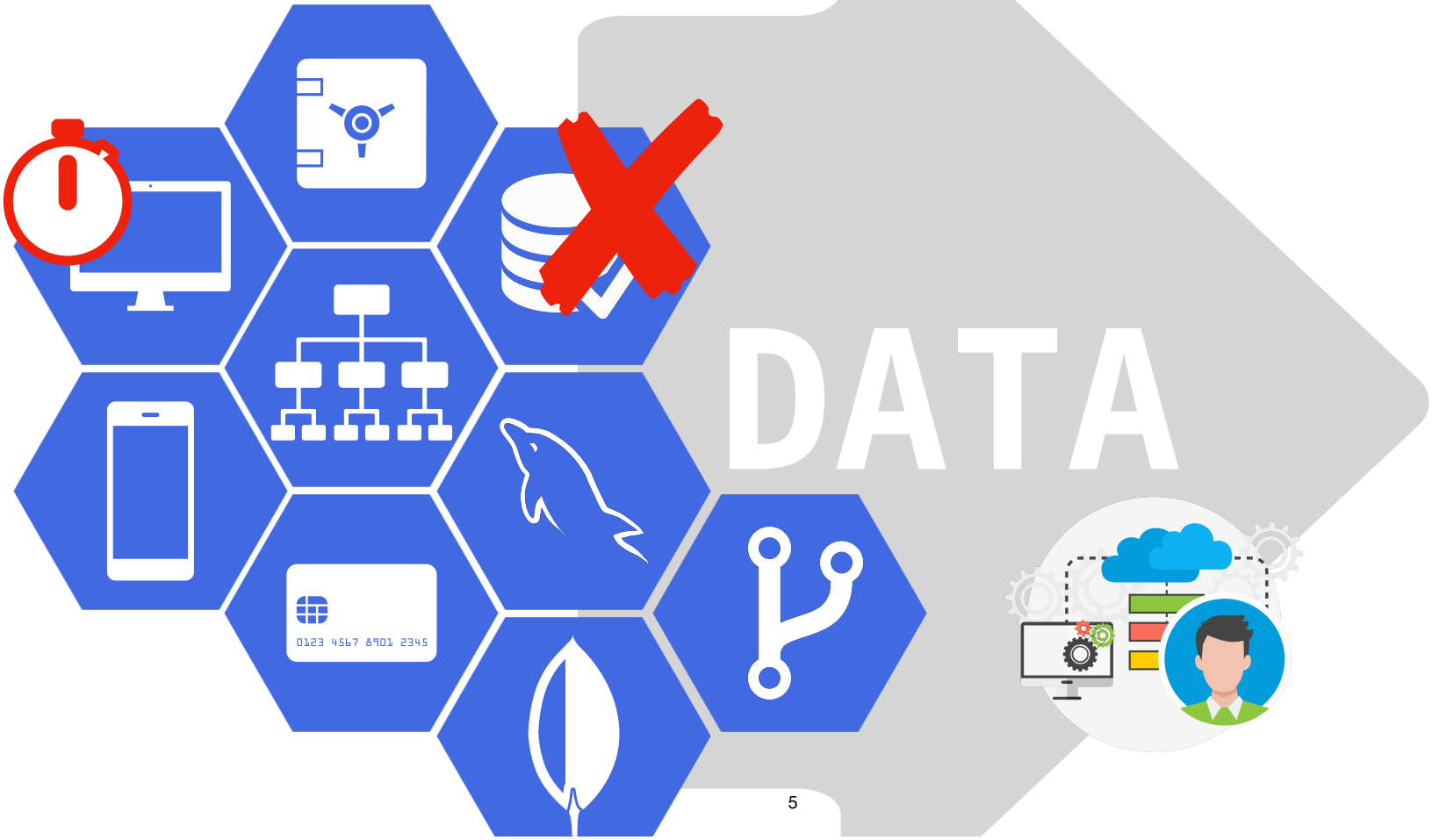
On May 12, 2020, Slack had our first significant outage in a long time. This is a detailed look into the technical issues that caused it.

Laura Nolan Senior Staff Engineer

7 minutes • Written 1 year ago

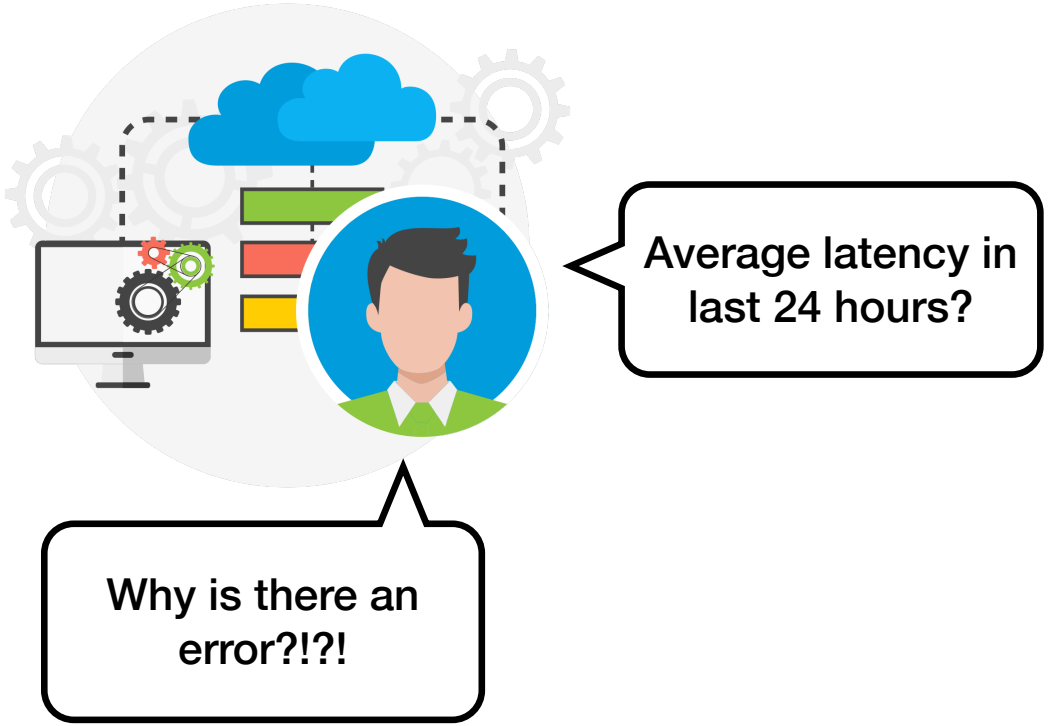
This story describes the technical details of the problems that caused the Slack downtime on May 12th, 2020. To learn more about the process behind incident response for same outage, read Ryan Katkov's post, "All Hands on Deck".

Observability: A data intensive task

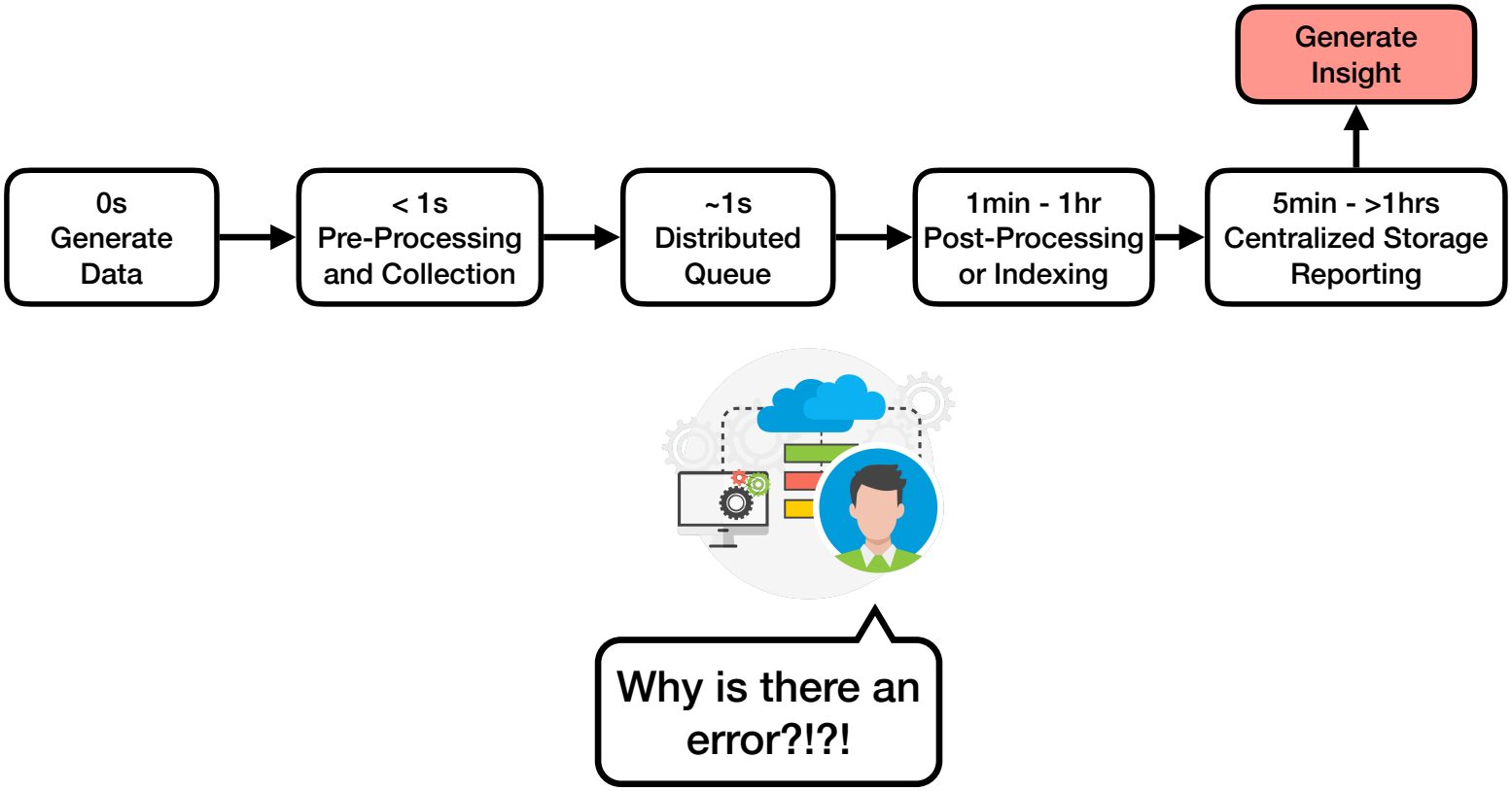


Observability: A data intensive task

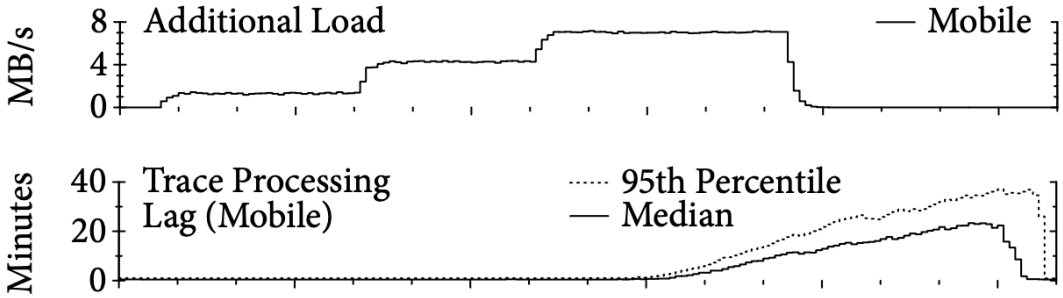
Fresh Data
Unknown Queries
Complete Data
Knowledge development



Problem: Analytics Pipeline and Observability



SotA don’t solve this problem



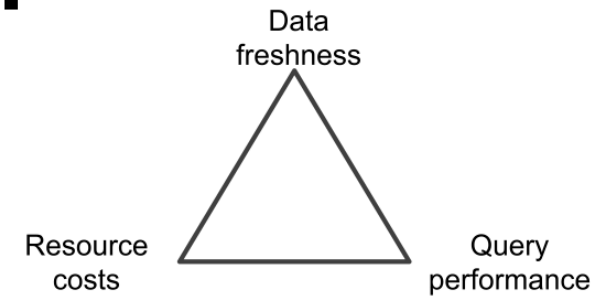
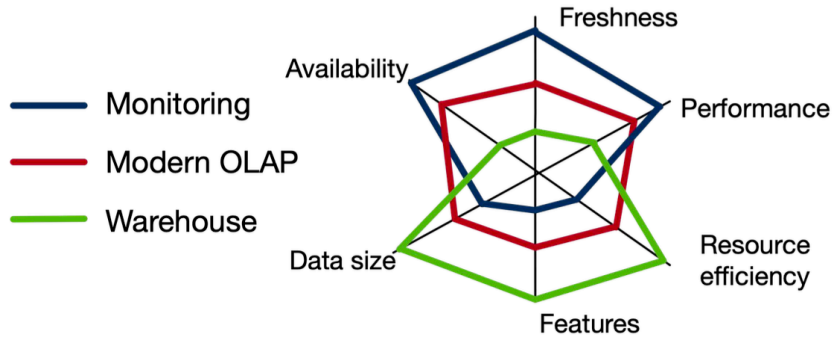


Figure 1: Three-way tradeoffs offered by Napa to maintain databases at an appropriate query performance, data freshness and cost.



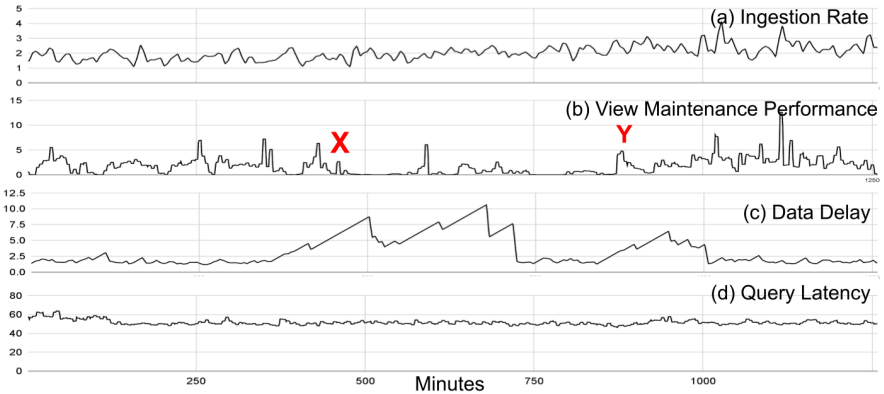


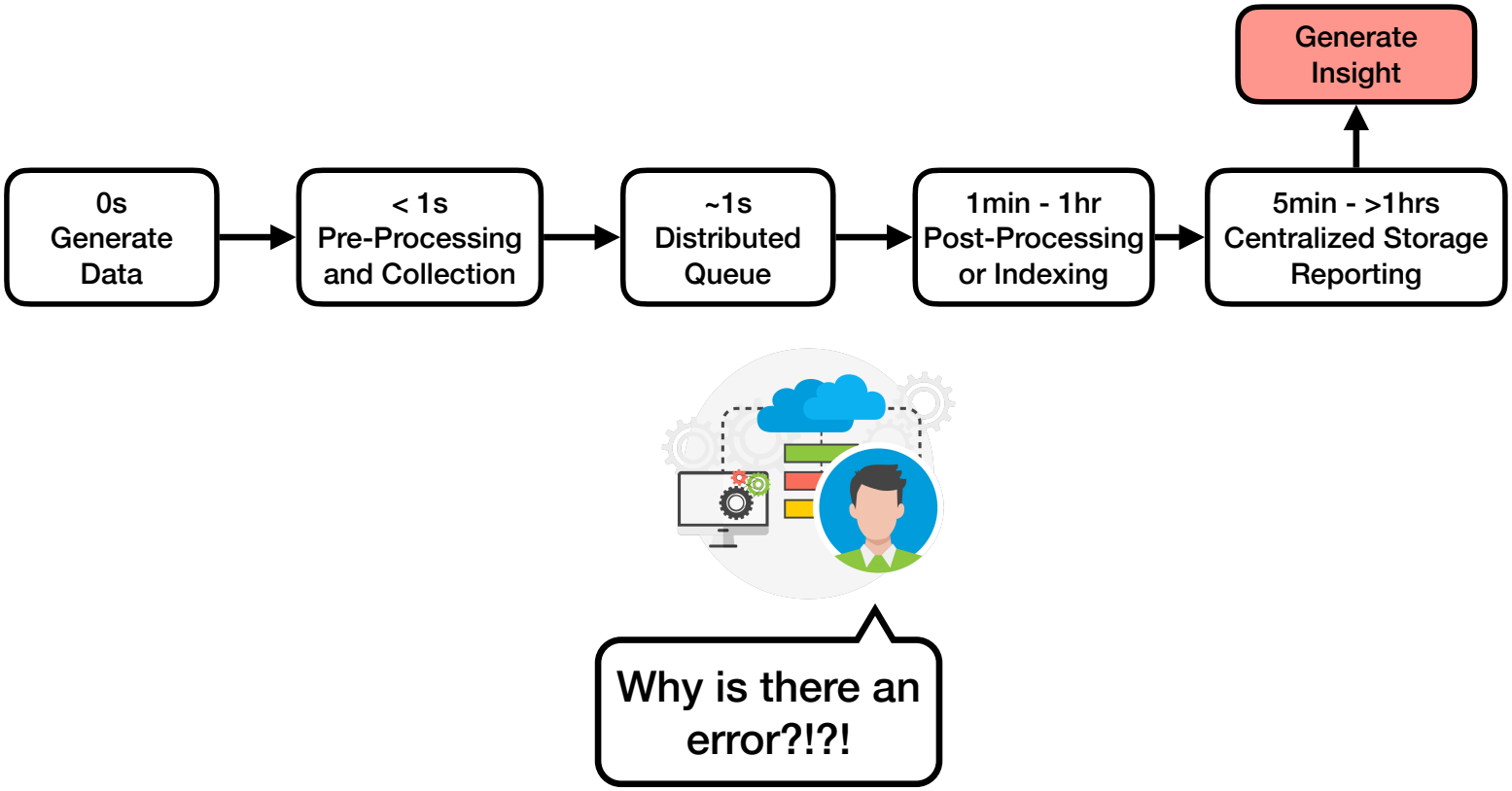
Figure 9: Napa’s decoupling means view maintenance backlog does not impact query performance.

Completeness vs Freshness

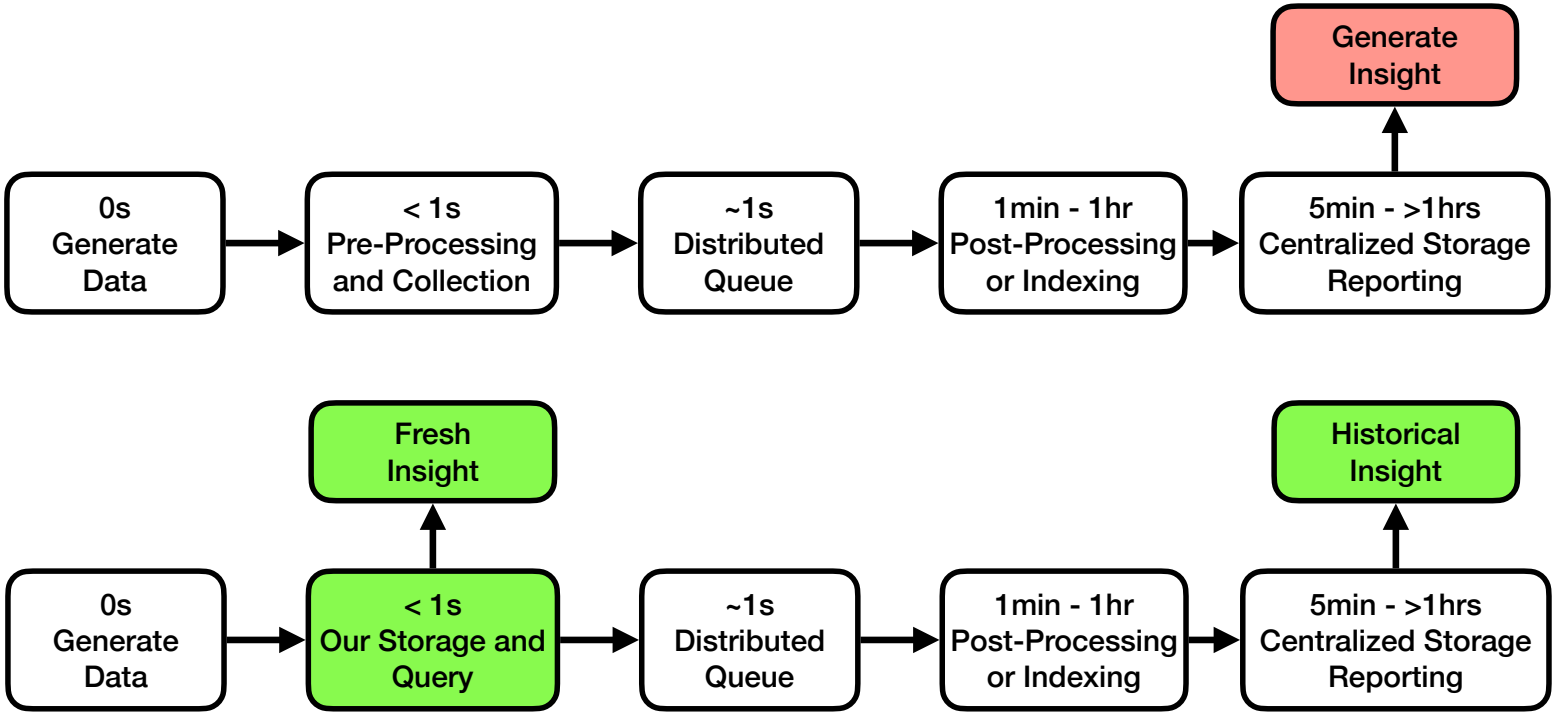
Fresh Data:
Drop Samples to keep up

Complete Data:
Wait for data to show up

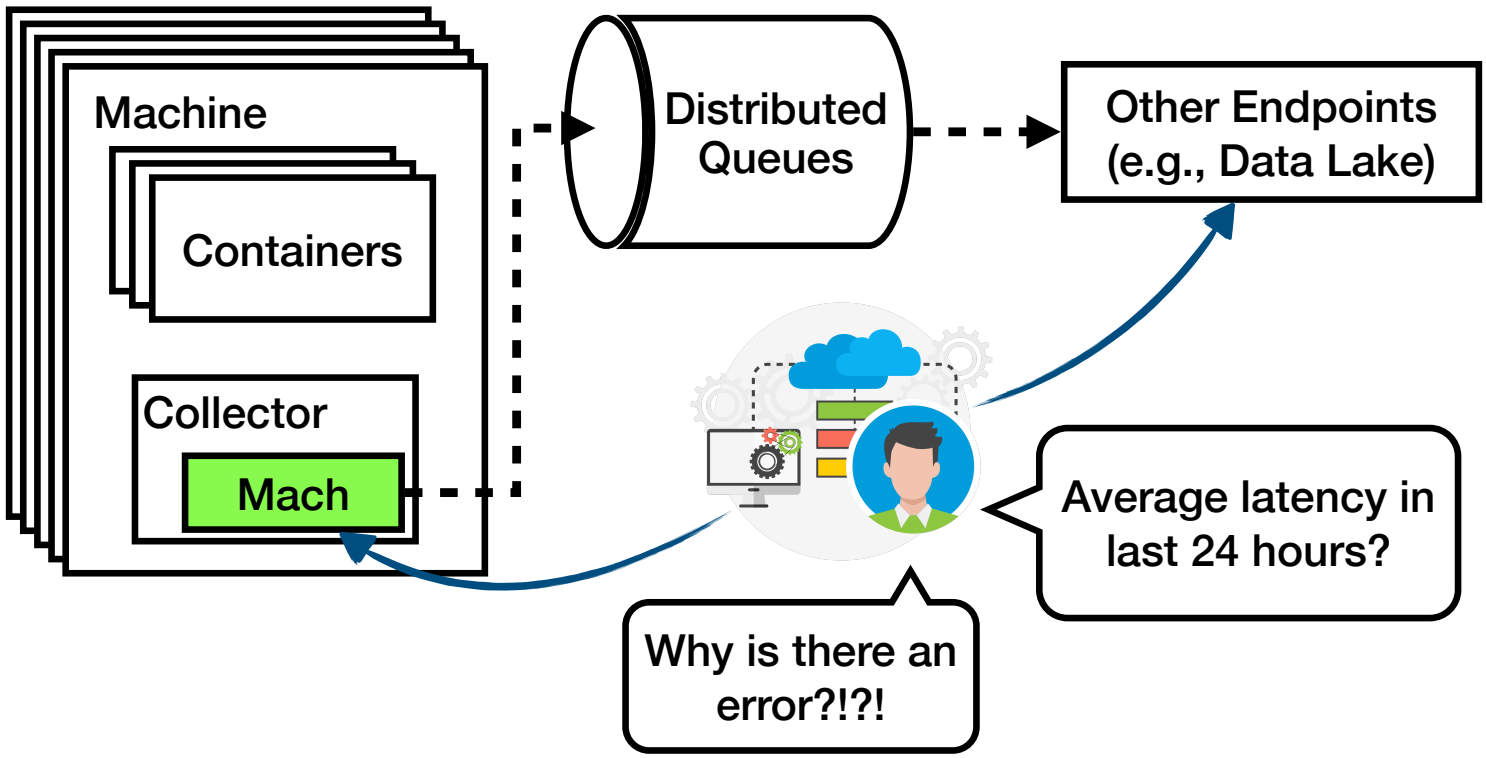
Problem: Analytics Pipeline and Observability



Problem: Analytics Pipeline and Observability

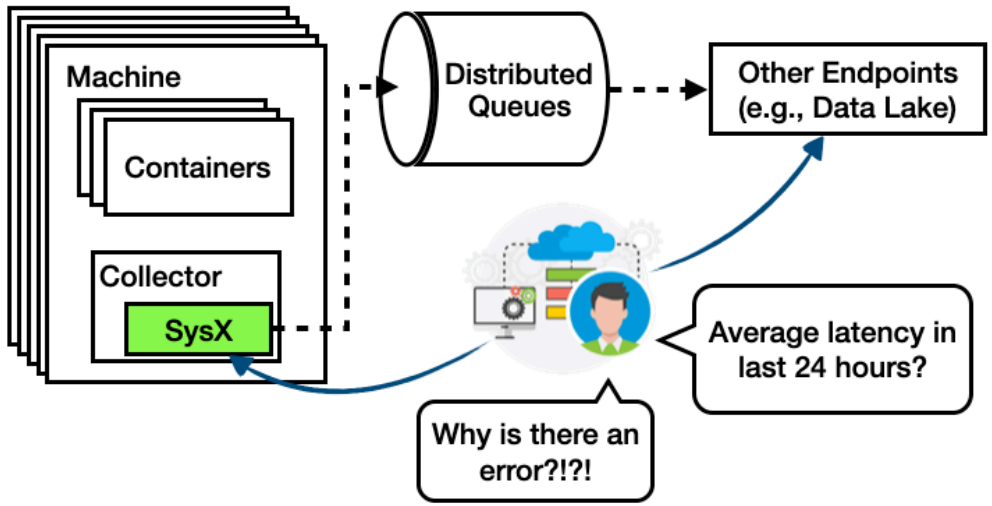


The Abstraction: The Mach Storage Engine



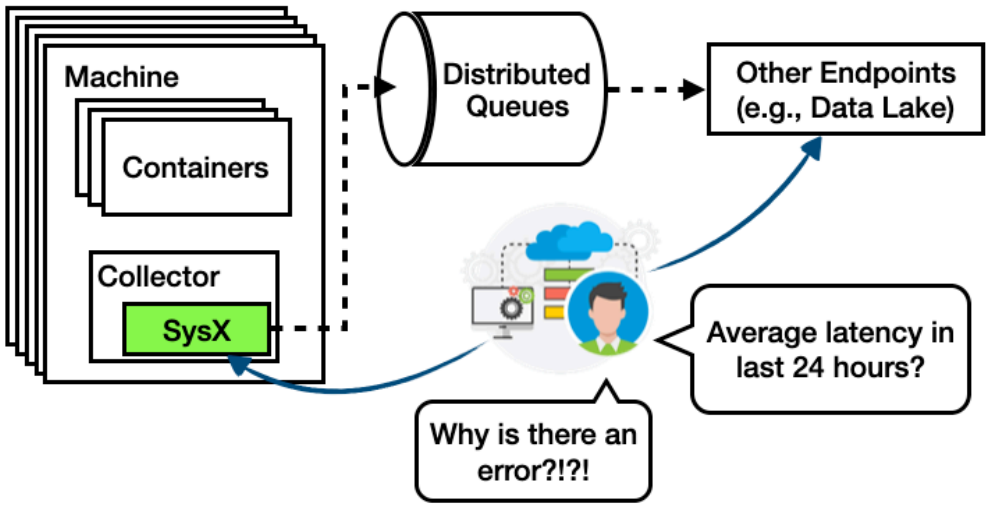
The Abstraction: A simple API

- Prioritize Freshness and Ingest Speed
- Unknown unknowns!!
- **Push** a sample from a source
- **Get** a samples for a given (recent) time range and source

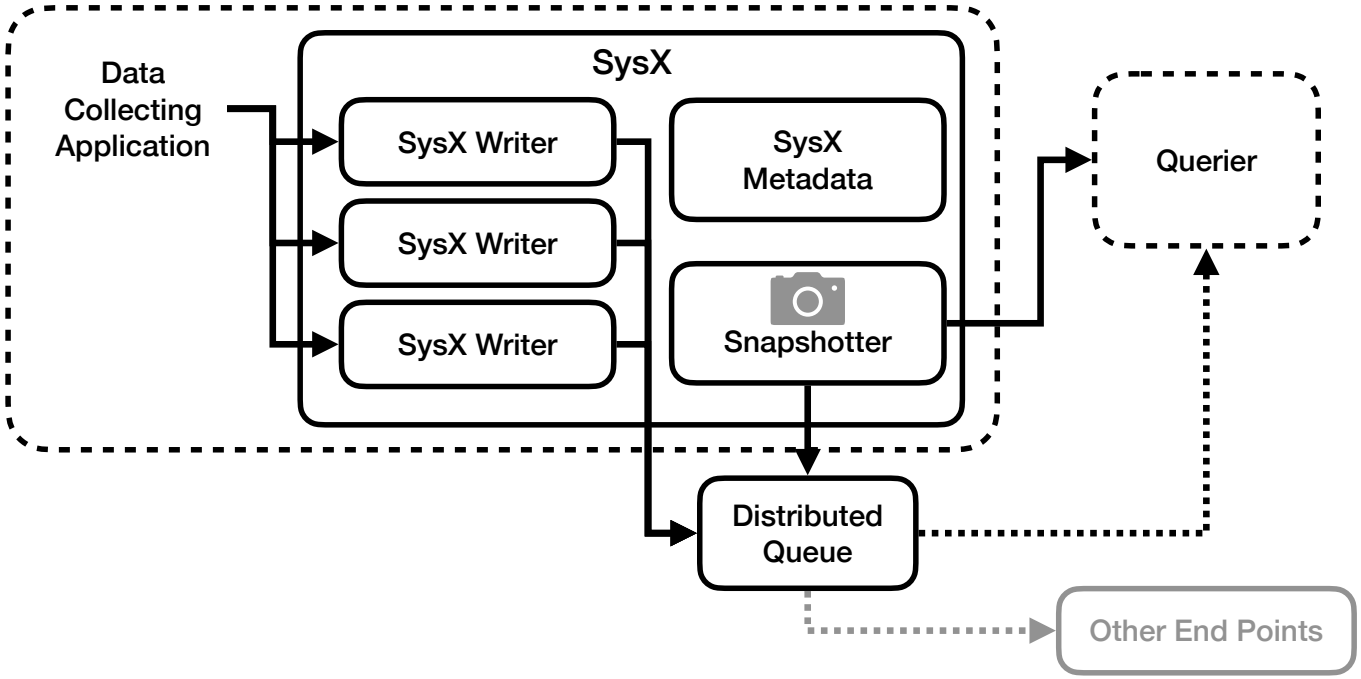


The System: Key Design Requirements

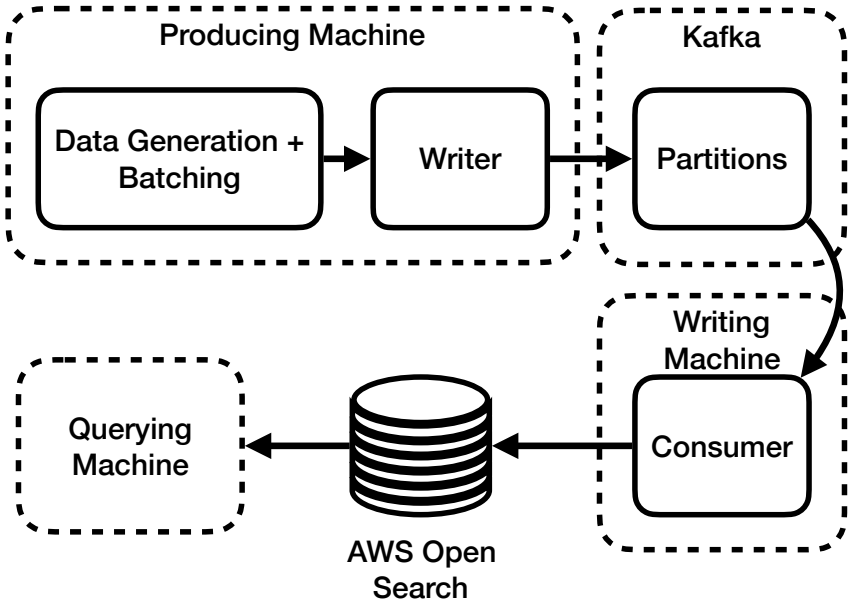
- Low-overhead in-node (both ingest, and querying)
- Fast ingest rates
- Make data available immediately
- No additional data duplication
- Fit right into existing pipelines

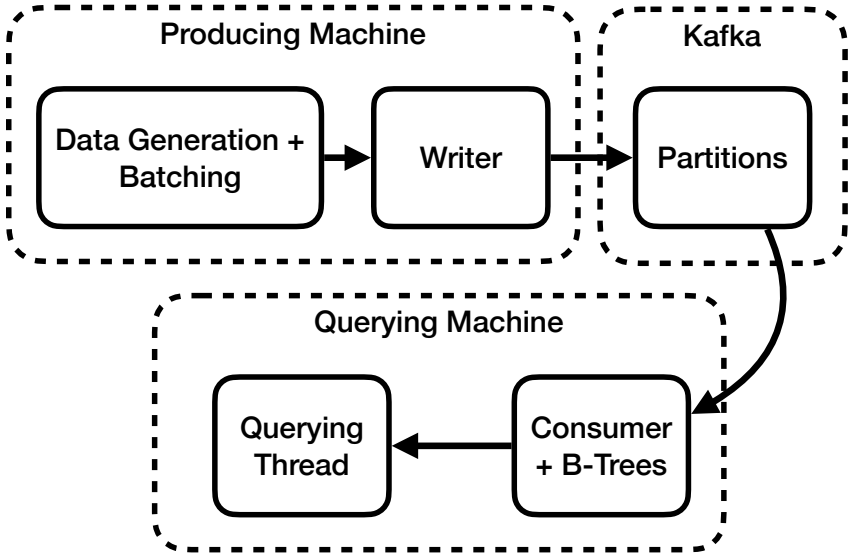


The System: High Level Arch

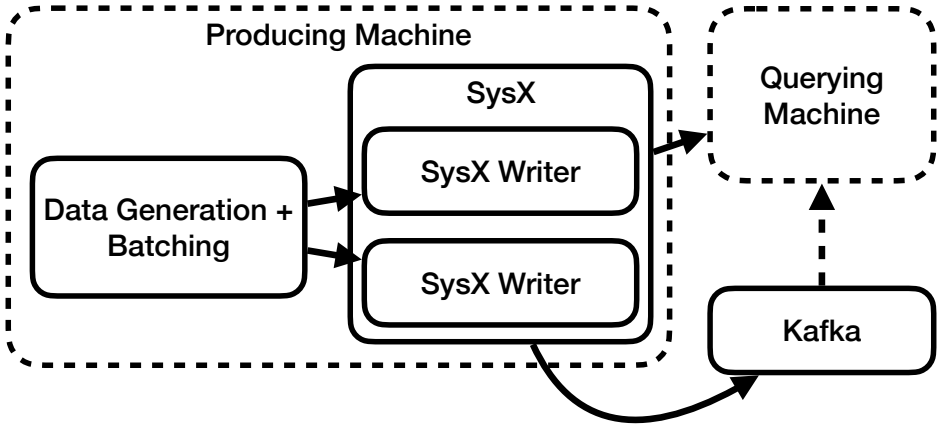


Evaluation Setup: Baselines





Evaluation Setup: Mach



Mach keeps up with high data rates…

- Introduce high data load for 5 minutes.
- Make sure Kafka can keep up
- Baseline compression + serializing cannot keep up
- Queue grows in baseline but not Mach
- CPU Bound

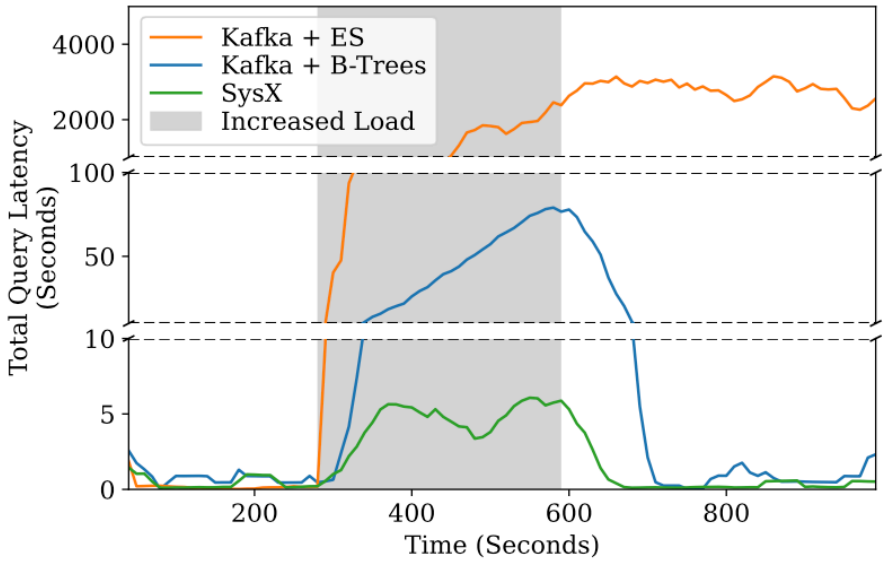


Figure 9: SysX maintains low query latency when the host experiences a load spike in observability data. Unlike the baselines, which wait for data to arrive, SysX's high ingestion throughput and ability to query data in processing reduces latency to a few seconds.

…Even when Kafka is swamped

- Swamp Kafka (i.e., fill the pipe with something else) (i.e., ~250GB/s with 3x replication)
- Baseline builds up an unqueriable queue internally
- Mach builds up queriable internal buffers with low overhead.
- Kafka-bound

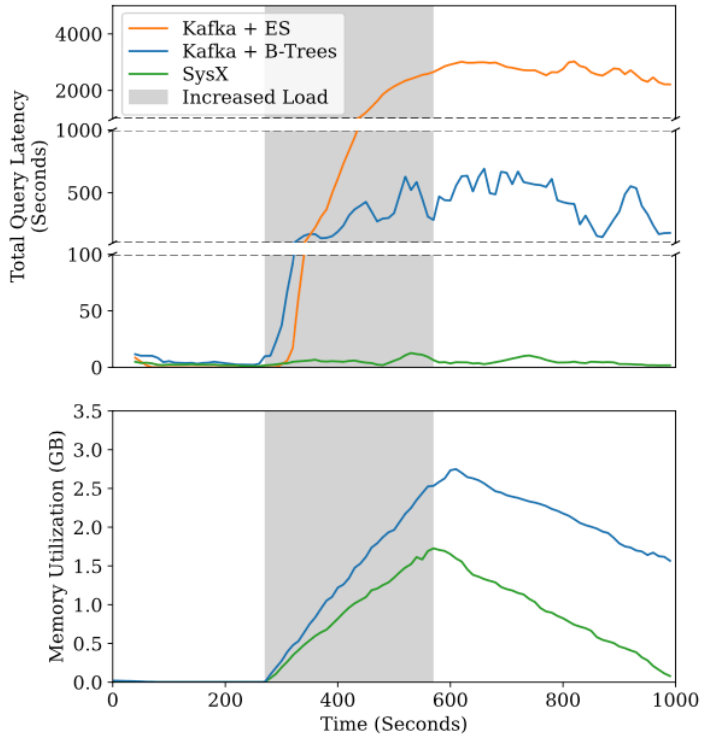


Figure 10: On a saturated Kafka cluster, SysX maintains low query latency and continues to make fresh data accessible. It does so while maintaining lower memory utilization than the Kafka + B-Trees baseline during load spikes when Kafka falls behind.

How fast is fast? Oh it's *fast*

- Need to drop data to maintain freshness
- How much data to can each system keep up with?
- Mach is almost 2x better than hand tuned baseline (blue)
- Kafka + ES is terrible…

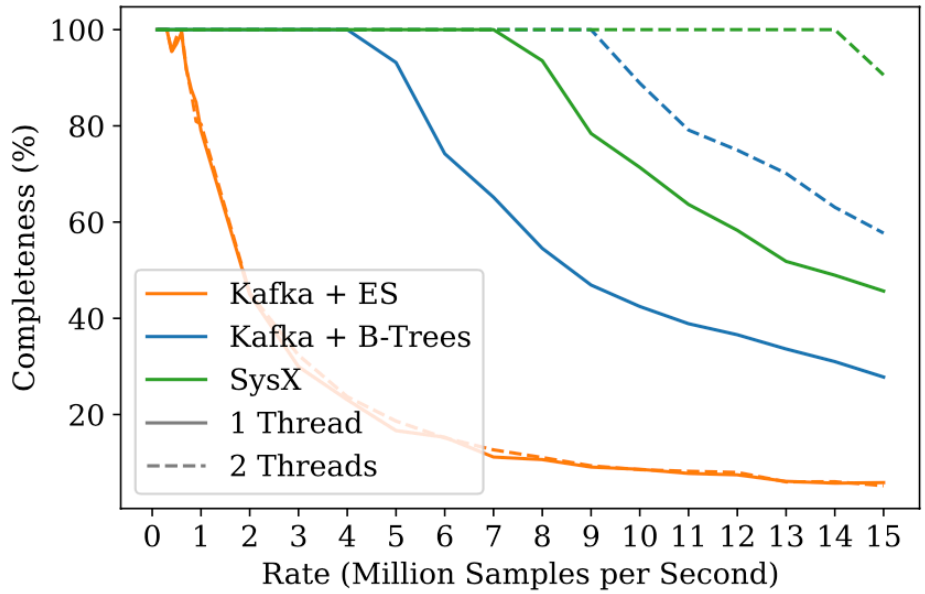
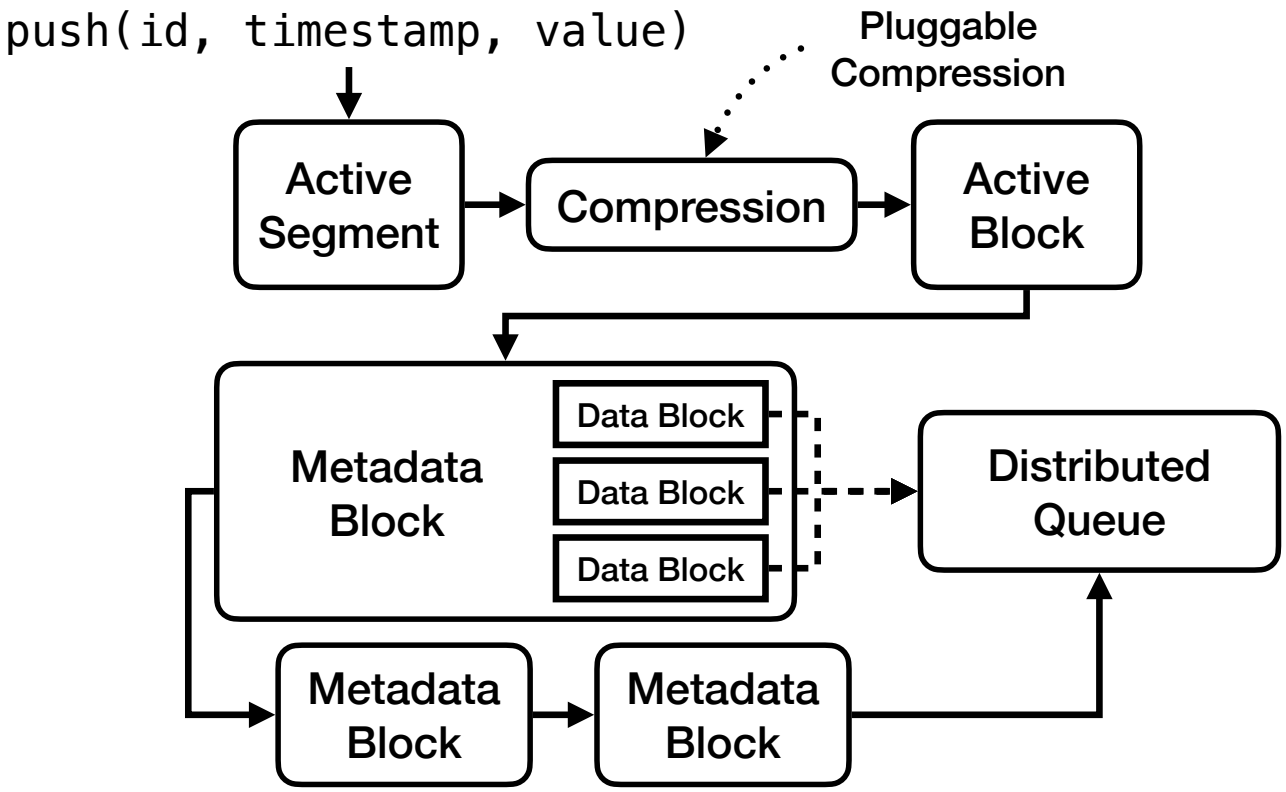


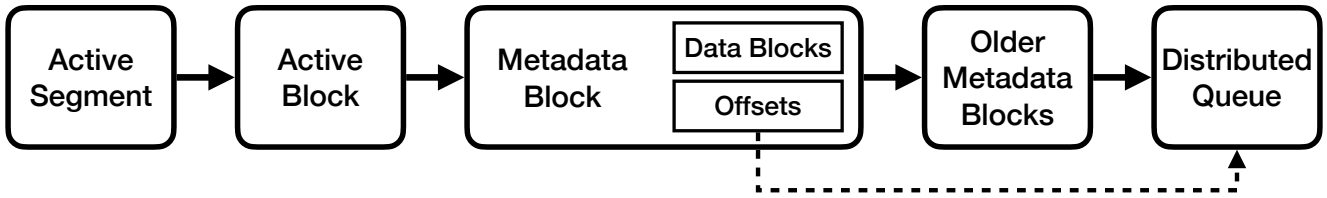
Figure 11: SysX scales to higher loads than the baseline systems, thereby decreasing the need for sampling or dropping data.

Questions?

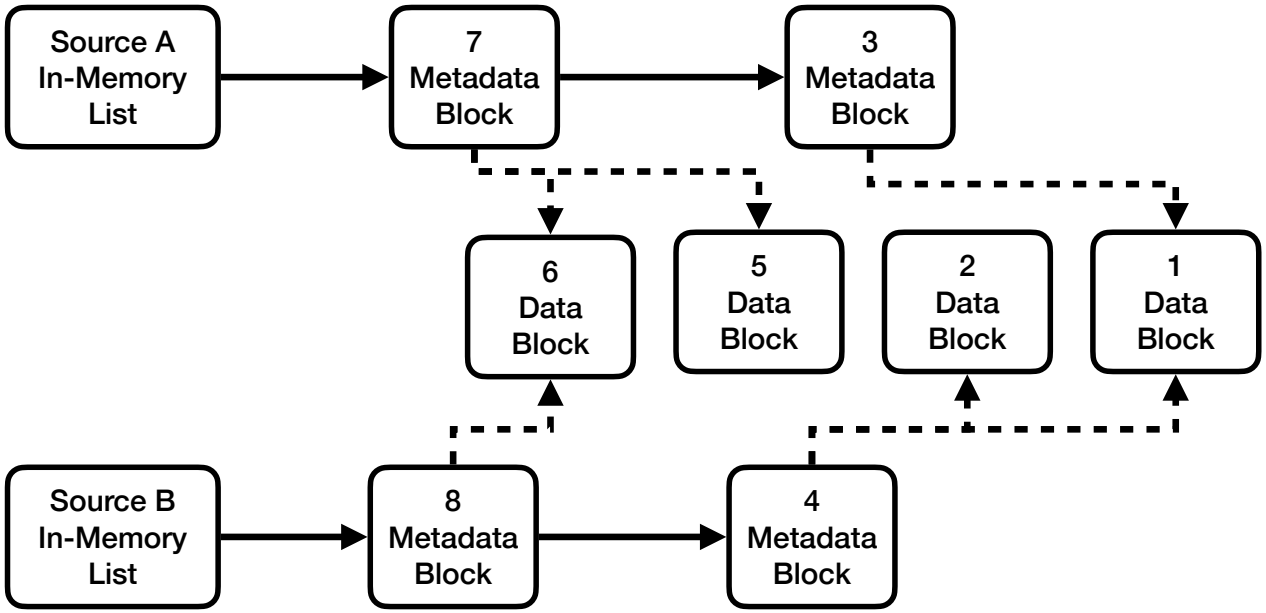
Push API and Write Path



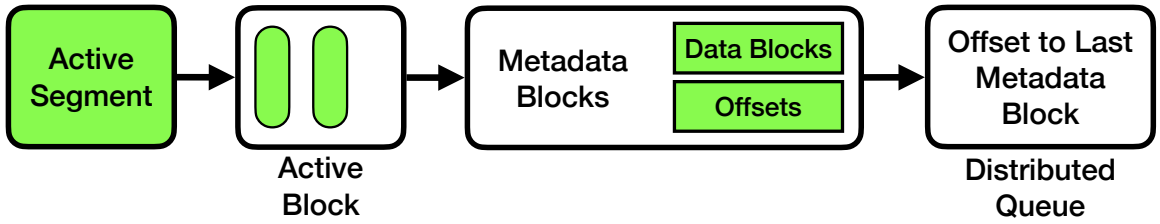
In-memory linked list



Distributed Queue Linked List



Snapshotting Mechanism



Snapshotting Mechanism