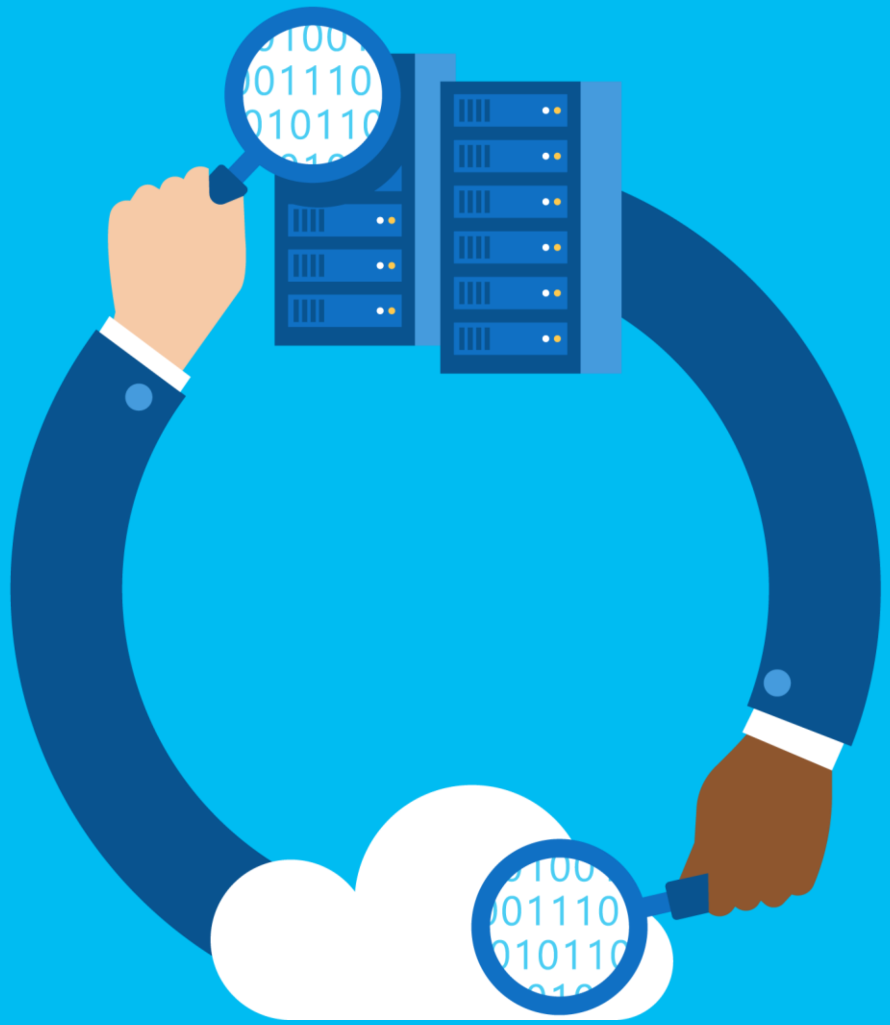
Microsoft

Rethinking Storage

for the cloud, edge, serverless, and big data era

Badrish Chandramouli

Microsoft Research, Redmond
badrishc@microsoft.com | @badrishc
https://badrish.net/



Introduction

- The **cloud** has matured as the platform for compute and data processing
- The **edge** is becoming important as a source, destination, and conduit for cloud computation
- There is increased focus on simplicity, ease of adoption and deployment, and auto-scaling with **serverless** abstractions
- We are ingesting, storing, processing rich **big data** with dynamic schema, such as JSON

The Compute – Storage Gap

- Storage (be it main memory, local disk, or cloud storage) is not keeping up with advances in compute simplification

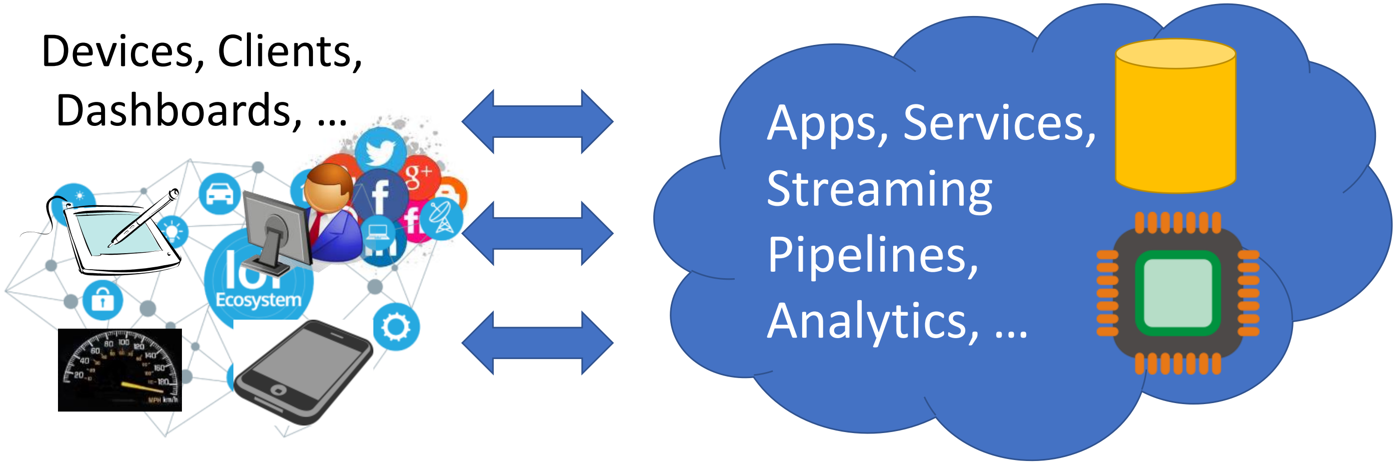
- Today's state of the practice
  - Use auto-scaling compute (Lambda, Functions) or Kubernetes
  - Keep everything in memory: use input replay or tolerate data loss
  - Or, use remote elastic SQL/storage services (Aurora, Socrates, BigTable, …) on every invocation/event
  - Throw in a cache as an afterthought – always Redis

The Landscape Today

- A kitchen sink of storage services and design patterns for stateful apps over modern compute substrates
- Poor memory & storage utilization, latency (**last mile** is longest)
- Unclear recovery & consistency guarantees in distributed deployments with caches
- An inability to ingest, store, process modern & rich evolving datasets quickly (e.g., the Twitter firehose)
- Too much user effort: choosing indices, storage formats, and data layouts, …

Case Study: Trill for Bing Ads

- Trill is a high speed in-mem columnar streaming analytics library
- Now OSS; used across Microsoft: Azure, Bing, Office, Windows, …
  - Library model of Trill was a huge success
  - Used with a variety of distributed fabrics (Orleans, Scope/Cosmos, Kubernetes, …)
- Bing Ads uses Trill in scaled-out Scope compute infra
- Temporal Locality of State
  - Search engine maintains per-user stats over last week
  - Billions of users "alive" at given instant
  - But, only millions actively surfing
  - Everything stored in main memory
  - Storage is the main reason to scale out



The SimpleStore Research Agenda

Simplify app view of [storage + cache]; high performance
Build single-node embedded storage artifacts

- Use by **end-user apps** or **cloud services**
- Use as **storage accelerator** or **point of truth**

- Compute Workloads
  - Unified log/storage abstraction across memory, local, cloud storage (FASTER Log)
  - Embedded KV store + cache (FASTER KV)
  - Scalable consistency & recovery models for such workloads (CPR)
  - Resilient stateful actors (CRA / Ambrosia)
  - **In progress**: auto-scaling and zero-config library for serverless storage

- Big Data Analytics Workloads
  - Embedded library for ingesting, storing, querying flexible-schema data (FishStore)
  - Fast partial parsing techniques for flexible schema data (Mison)
  - **In progress**: ML-driven automatic data layout and indexing of high-dimensional data

Talk Outline

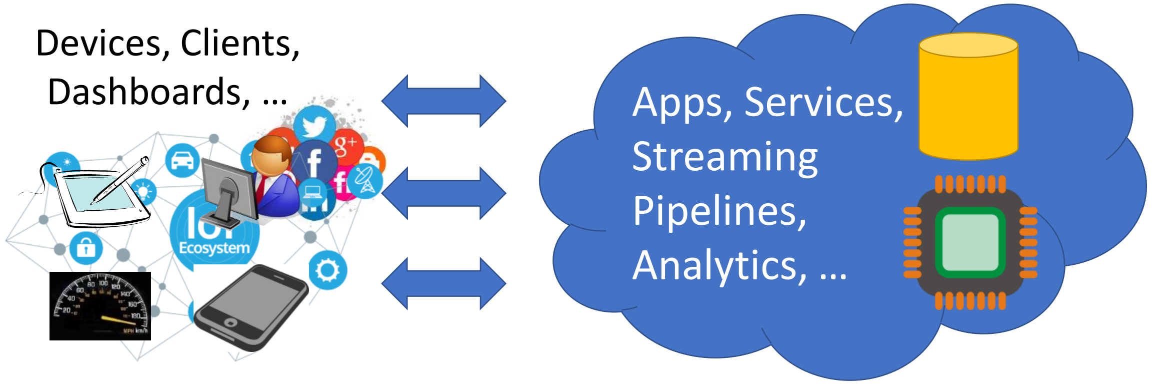
- Introduction & Motivation
- SimpleStore for Compute Workloads
  - FASTER Log
  - FASTER KV + cache
  - Concurrent Prefix Recovery
  - Library for Serverless
- SimpleStore for Big Data Analytics
  - FishStore for flexible schema data
  - Learned data layouts for storage & caching
- Conclusions

Talk Outline

- Introduction & Motivation
- SimpleStore for Compute Workloads
  - FASTER Log
  - FASTER KV + cache
  - Concurrent Prefix Recovery
  - Library for Serverless
- SimpleStore for Big Data Analytics
  - FishStore for flexible schema data
  - Learned data layouts for storage & caching
- Conclusions

Compute over Fine-Grained Objects

- Many apps operate over billions of fine-grained objects
  - IoT device tracking, data center monitoring, streaming, online services, …
- State consists of independent objects – *devices, users, ads*
  - Overall state doesn't fit in memory
  - Point ops with lots of updates
    e.g., update per-device average CPU reading
  - Atomic read-modify-write (RMW)
  - State exhibits temporal locality
  - State needs to be recoverable



- Problem across edge, cloud, multi-tenant, and serverless applications

What is FASTER

- An open-source library for accelerating object storage
  - High performance, concurrent, latch free, shared memory
  - Two sub-components

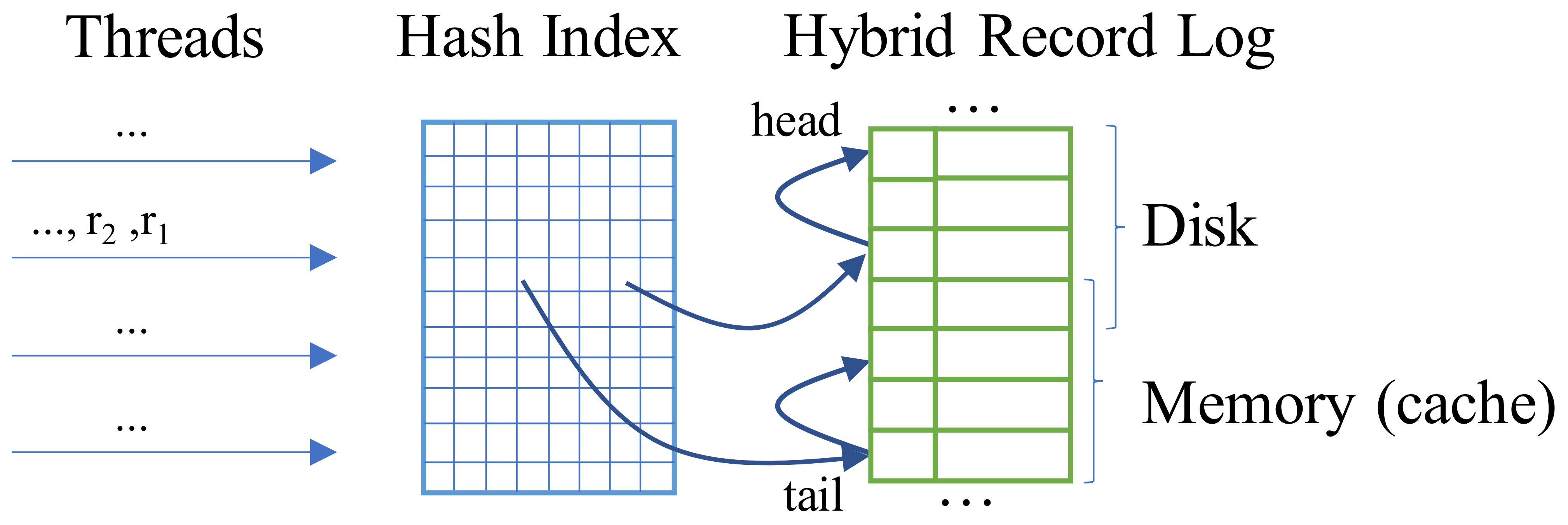
1) FASTER Log

  - Record log abstraction over tiered storage: enqueue, commit, scan, read, truncate
  - "Hybrid" support: tail may optionally be modified in-memory safely, as **mutable region**
  - Can be used independently as a *persistent queue*

2) FASTER KV

  - Hash key-value store over the record log
  - Shapes the (changing) hot working set in memory → integrated cache
  - Performance: up to 200 million ops/sec for YCSB variants
    - One Intel Xeon machine, two sockets, 72 threads
    - Exceeds throughput of pure in-memory systems when working set fits in memory

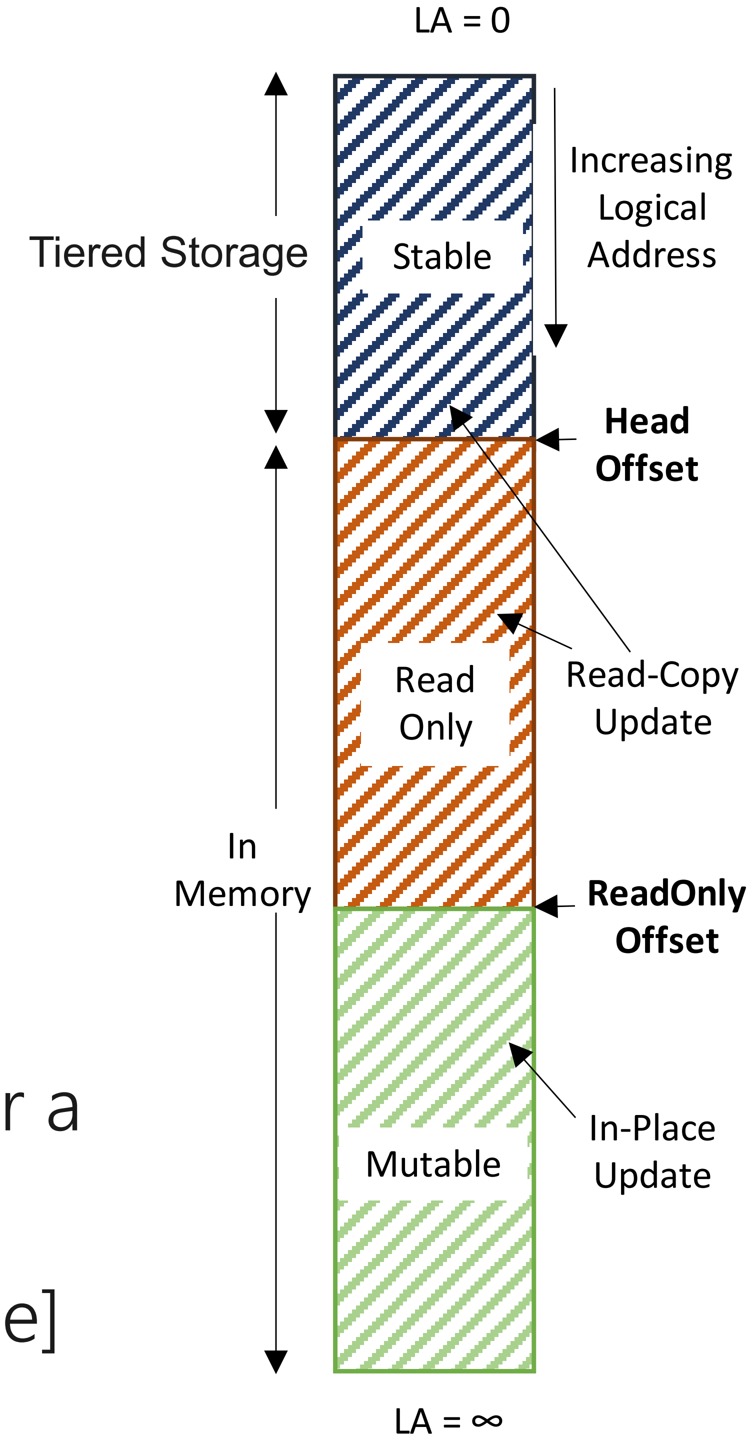
Architecture & Components



- Technical Innovations
  - **Indexing**: Concurrent Hash Index
  - **Record Storage**: "Hybrid Log" Record Allocator
  - **Threading**: Epoch Protection Framework with Trigger Actions

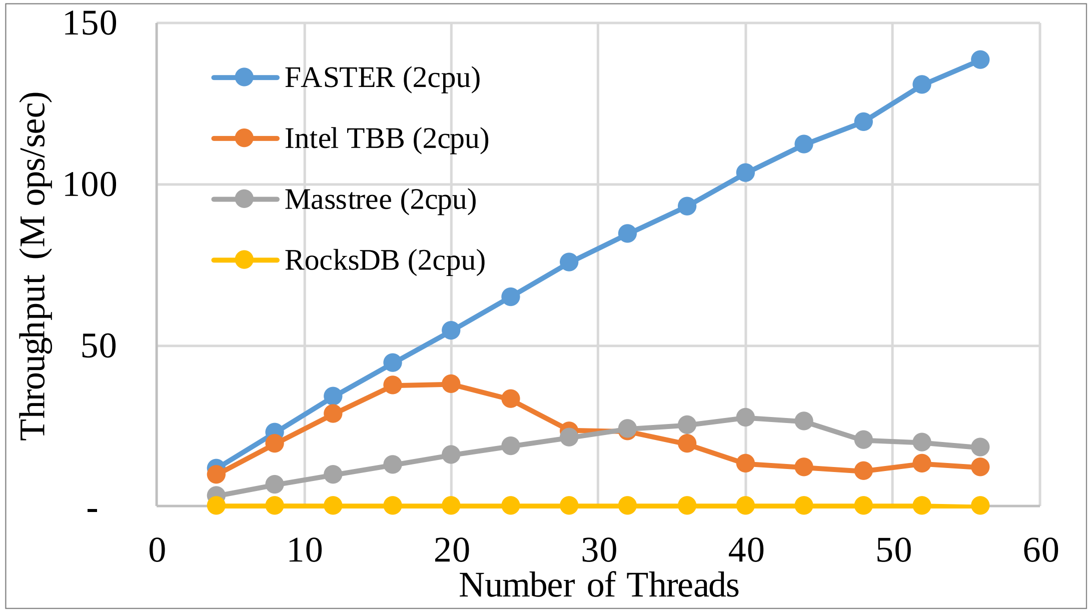
Hybrid Log in Brief

- Divide memory into three regions
  - Stable (on disk) → Read-Copy-Update (RCU)
  - Mutable (in memory) → In-Place Update (IPU) - *optional*
  - Read-only (in memory) → Read-Copy-Update (RCU)
- Hybrid concurrency model
  - RCU: compare-and-swap on index
  - IPU: user record-level concurrency
- Tail grows → offsets grow as well
  - New records allocated at tail
- New & updated records stay in mutable region for a while → captures temporal locality
- Supports tiering, e.g., [memory, SSD, cloud storage]

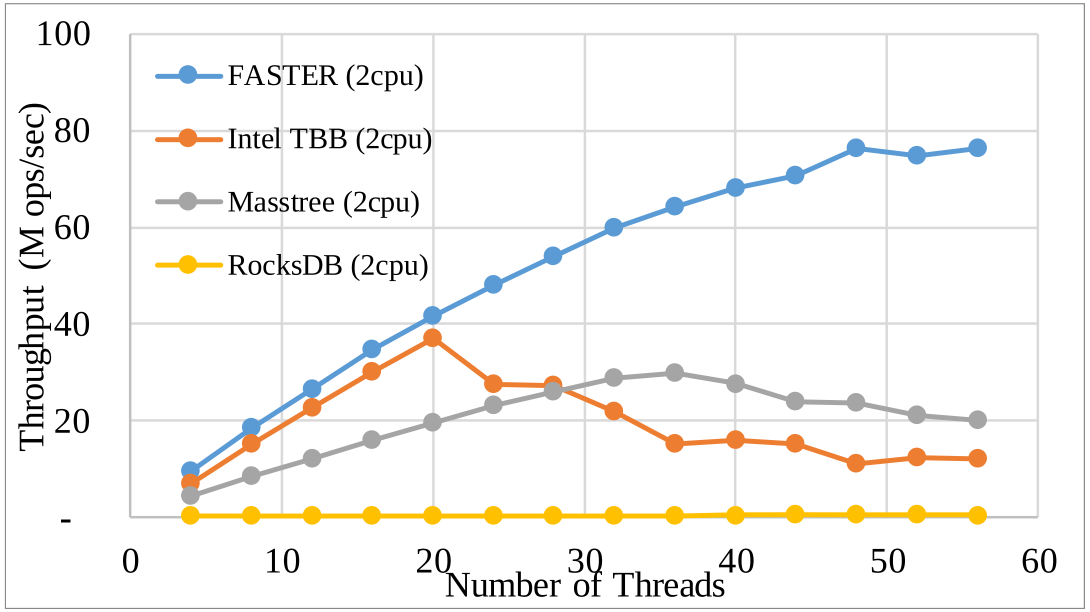


Scalability of FASTER KV with # Threads

- When current working set “happens to fit” in hybrid log memory



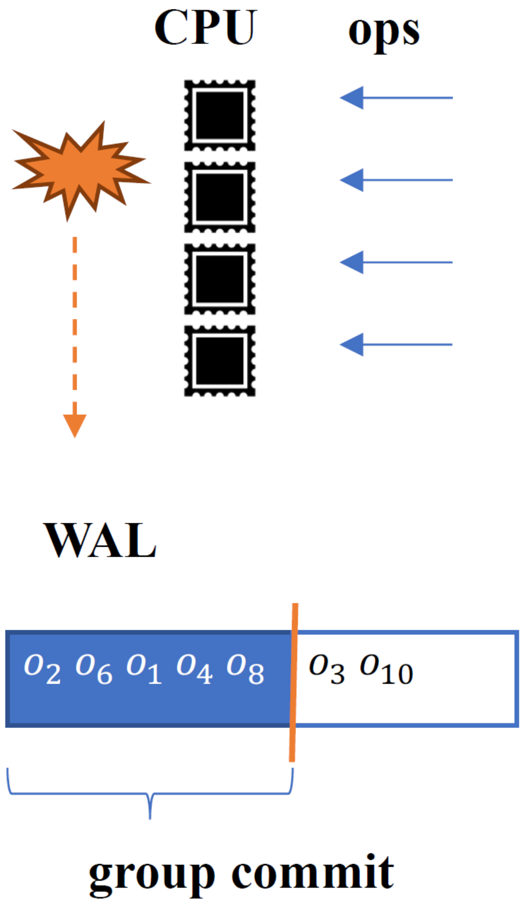
100% RMW; 8 byte payloads



100% blind updates; 100 byte payloads

What About Durability?

- Write Ahead Log? Every change is recorded in WAL
- Stresses write bandwidth; log is a scalability bottleneck; fine-grained commit acks

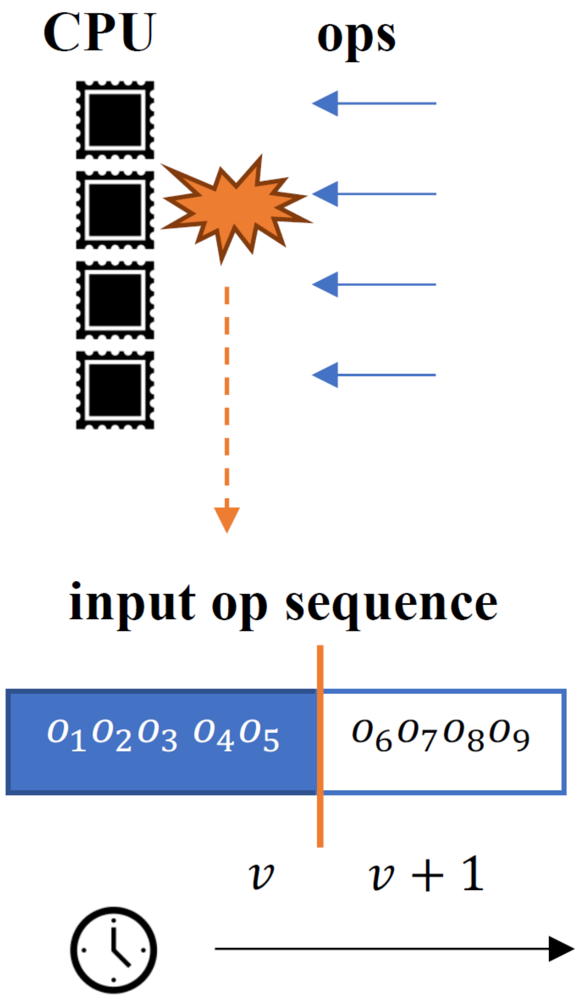


FASTER + WAL:
>150M ops/sec → ~15M ops/sec

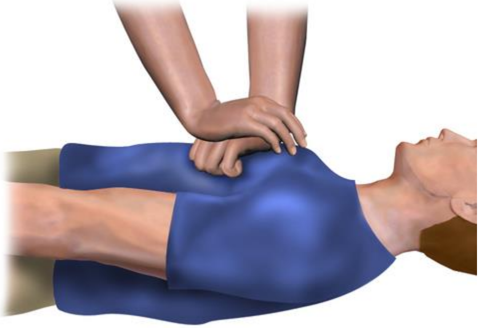
Custom in-mem txn database + WAL:
bottleneck at ~20M single-key txns per sec

Towards Our Approach: Prefix Recovery

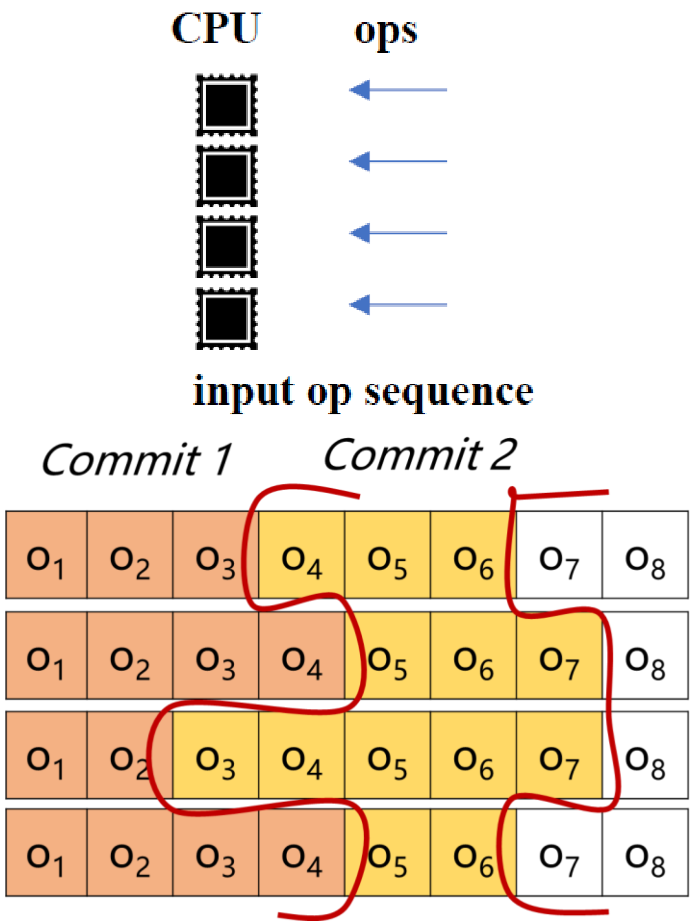
- Adopt the semantics of group commit
- Prefix Recovery (PR) based commit
  - Commit = { all ops issued up to time t }
  - Clients can prune in-flight op log until t, expose commit
- Compatible with reliable messaging systems (e.g., Kafka)
- Today's PR approaches are not scalable
  - Using WAL: { fuzzy chkpt + WAL }
  - Atomic commit log of ops → scalability bottleneck
  - Quiesce the database → not desirable



Concurrent Prefix Recovery (CPR)



- System notifies each thread S_i of a commit point t_i in its local operation timeline
  - Eliminates system-wide single time point t
- All ops before t_i are committed, and none after, $\forall i$
- Same consistency as PR, but allows scalable multi-threaded implementation
- System, not user, chooses exact CPR point per thread → key to non-blocking



Using CPR to Build Systems

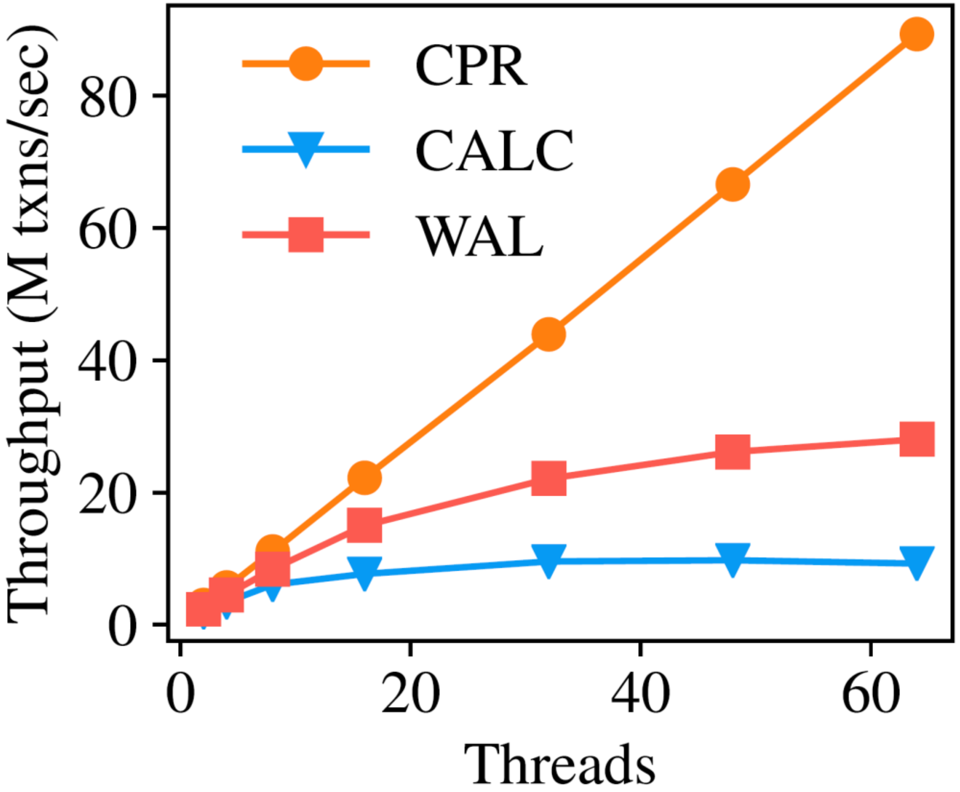
- CPR makes it possible to implement scalable group commit
- But, non-trivial to design systems that achieve this scalability!
- We used CPR to add durability to
  - Simple concurrent shared-memory transactional database
  - FASTER KV
- Non-trivial details; based on epochs + state machine; see paper

- CPR model is interesting for distributed/serverless storage as well

In-mem DB Prototype + CPR

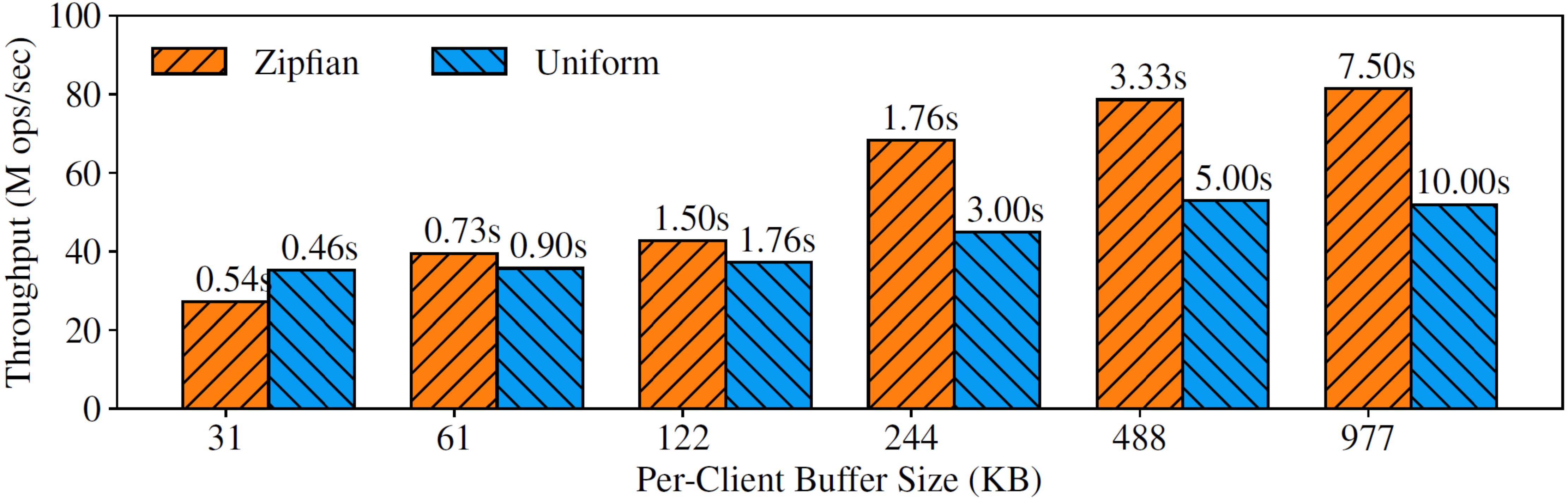
- Compared CPR against:
  - WAL
  - CALC (point-in-time checkpoints using atomic commit log of ops)

- Summary: CPR scales linearly with #threads
  - See paper for details



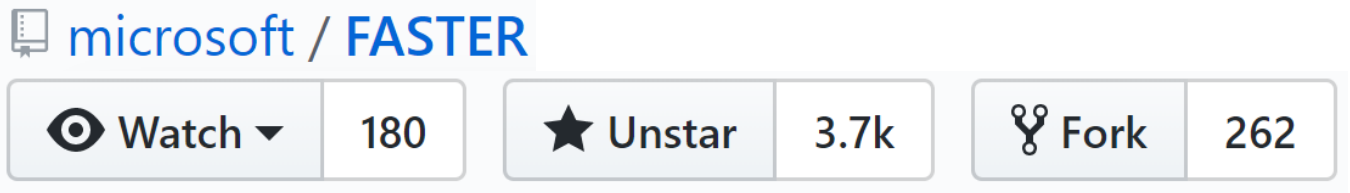
FASTER + CPR: End-to-End Experiment

- Vary client op buffer size; issue commit when buffer 80% full
- Use 36 client threads, YCSB 50:50 workload
- Figure shows a *commit latency vs. throughput tradeoff*



Current Status of FASTER

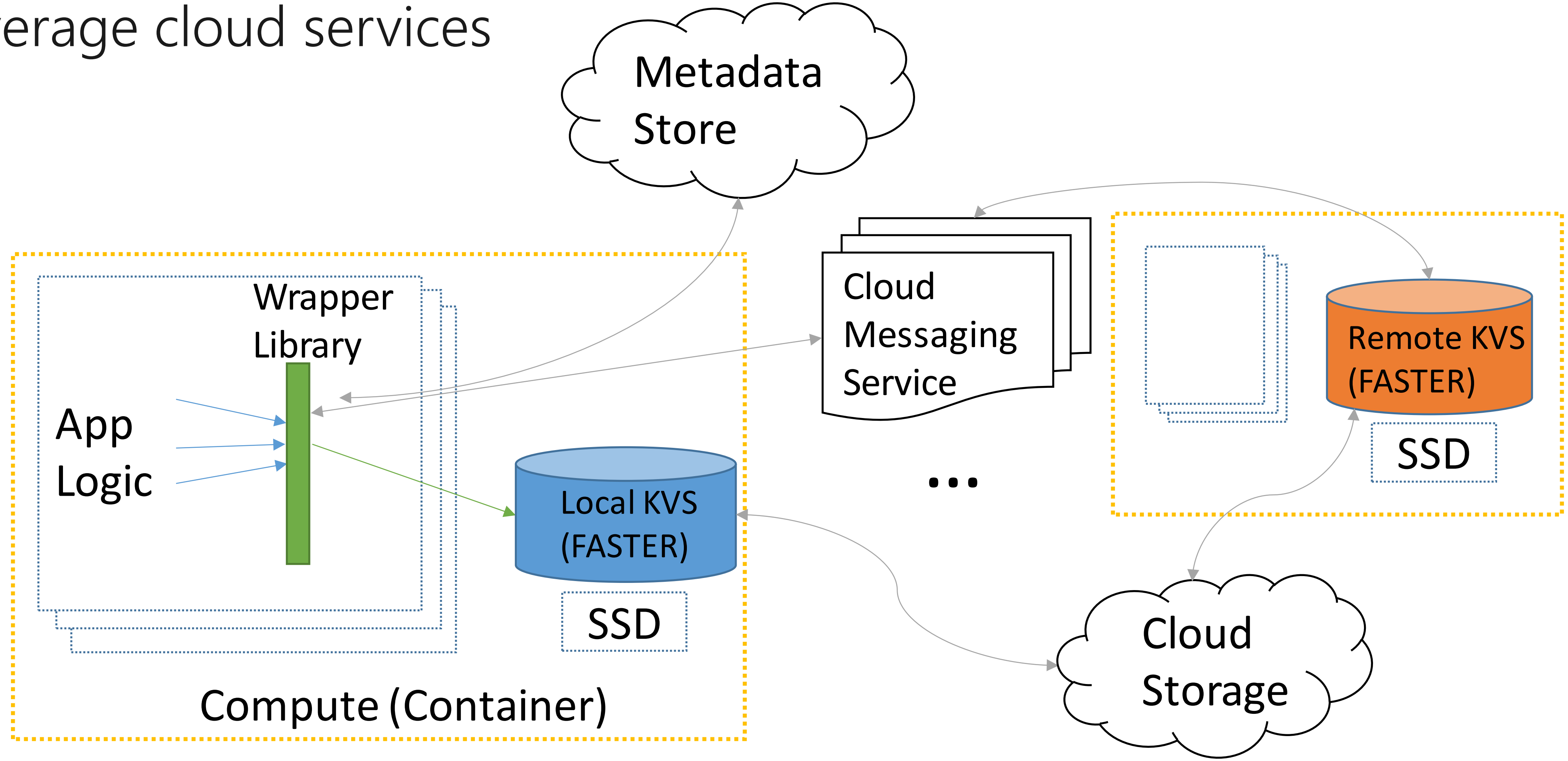
- Open sourced at https://github.com/microsoft/FASTER



- Research papers: SIGMOD 2018, VLDB 2018 demo, SIGMOD 2019
- Summary of Use Cases
  - State store for streaming pipelines
  - Edge cache^{++} in front of point-of-truth database backends
  - Scalable persistent queue abstraction for edge-cloud (FasterLog)
  - Integrated into Timely Dataflow (with Rust wrapper over FASTER C++)
  - Presented and evaluated recently as alternative to RocksDB (Flink Forward 2019)

Future: Storage for Serverless/Actor Apps

- CPR → Distributed CPR
- Leverage cloud services
- Decentralized Storage Library



Stateful Actor Frameworks

- Actor-oriented systems (Orleans, Ray, Durable Functions, Ambrosia) are helping simplify stateful applications
- Expose abstraction of [resilient compute + local memory]
  - Use DB ideas of checkpoint/replay or active-active for state recovery
- Reusable storage artifacts help build such systems, make it easier to manage app state
- Users still need storage + cache libraries
  - Applications do not always live within the confines of specific framework
  - Elasticity is easier, quicker, more reliable, manageable with stateless fabrics
  - Applications have diverse remote storage needs (e.g., store truth in CosmosDB, access larger-than-memory shards on compute node, map-reduce)

Talk Outline

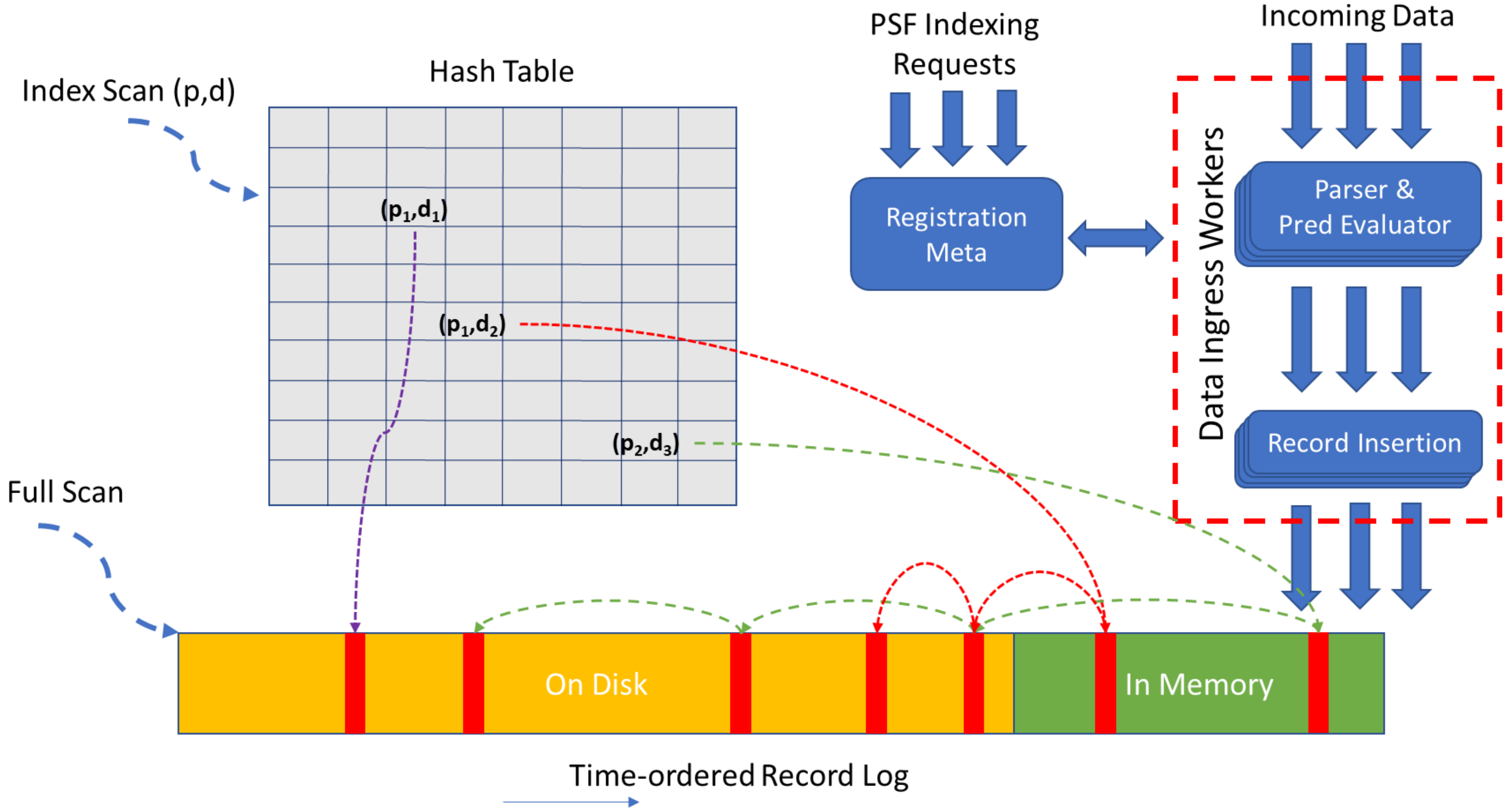
- Introduction & Motivation
- SimpleStore for Compute Workloads
  - FASTER Log
  - FASTER KV + cache
  - Concurrent Prefix Recovery
  - Library for Serverless
- SimpleStore for Big Data Analytics
  - FishStore for flexible schema data
  - Learned data layouts for storage & caching
- Conclusions

Simplifying Analytics: FishStore

- Stands for Faster Ingestion with Subset Hashing
- Storage library for dynamic flexible-schema data, e.g., JSON, CSV
  - Based on registration of dynamic predicates/query templates over data
  - Query-driven dynamic schema inference
  - Rockset talk provided great motivational use cases
- Two bottlenecks: indexing & parsing
  - Extended FASTER to index “interesting subsets” of data in chains
  - Generic parser interface to parse only “interesting” fields → we use Mison & simdjson
- Ingests at 10GB/sec, saturates 2GB/sec SSD with < 8 cores
  - Details: SIGMOD 2019 paper, VLDB 2019 demo
  - Open source at https://github.com/microsoft/FishStore

FishStore Architecture

- Ingest + index: fast path
- Dynamically reg/dereg templates
- Query on registered templates



Future: Automatic Data Layout, Caching, Indexing

- Ultimate Goal
  - Ingest high-dim flexible schema data, impose access workload (queries) on library
  - Storage auto-optimizes layout/access methods over time
- First attempt: workload-driven data layout for OLAP
  - Leverage reinforcement learning
- Initial results are surprising
  - Data layouts are an order-of-magnitude better than traditional layouts
  - Produces data blocks: form basis for caching at storage clients
  - Supports advanced layouts where tuples may be in multiple blocks

Thank You

microsoft/FASTER

Fast persistent recoverable log and key-value store, in C# and C++, from Microsoft Research.

C# C++ ★ 3.7k Forks 258

microsoft/Trill

Trill is a single-node query processor for temporal or streaming data.

C# ★ 1k Forks 86

microsoft/FishStore

FishStore is a prototype fast ingestion and querying layer for flexible-schema data

C++ ★ 60 Forks 3

microsoft/AMBROSIA

Robust Distributed Programming Made Easy and Efficient

C# ★ 306 Forks 28

microsoft/CRA

Common Runtime for Applications (CRA) is a software layer (library) that makes it easy to create and deploy distributed dataflow-style applications on top of resource managers such as Kubernetes, Y...

C# ★ 25 Forks 10

Find our open-source work at https://github.com/badrishc

Pubs at https://badrish.net/

Interested in working on SimpleStore? Contact me for internships @MSR.

Thanks to Present & Past Collaborators

Yinan Li, Donald Kossmann, Dong Xie, Guna Prasaad, Justin Levandoski, James Hunter, Chi Wang, Johannes Gehrke, Zongheng Yang, Tianyu Li, Sam Madden, Umar F. Minhas, Jonathan Goldstein, Ibrahim Sabek, Ryan Stutsman, Chinmay Kulkarni, and others.