FlexLog

A Shared Log for Stateful Serverless Computing

Dimitra Giantsidi, Manos Giortamis, Nathaniel Tornow,
Florin Dinu and Pramod Bhatotia

THE UNIVERSITY of EDINBURGH

Serverless (FaaS) computing

- Pay-as-you-go execution model
- Programmability and ease
  - Upload simple functions
  - Hide complexities
- Performance and scalability
  - No overheads to manage the infrastructure

Serverless computing infrastructure is offered by all major cloud providers

Serverless workloads characteristics

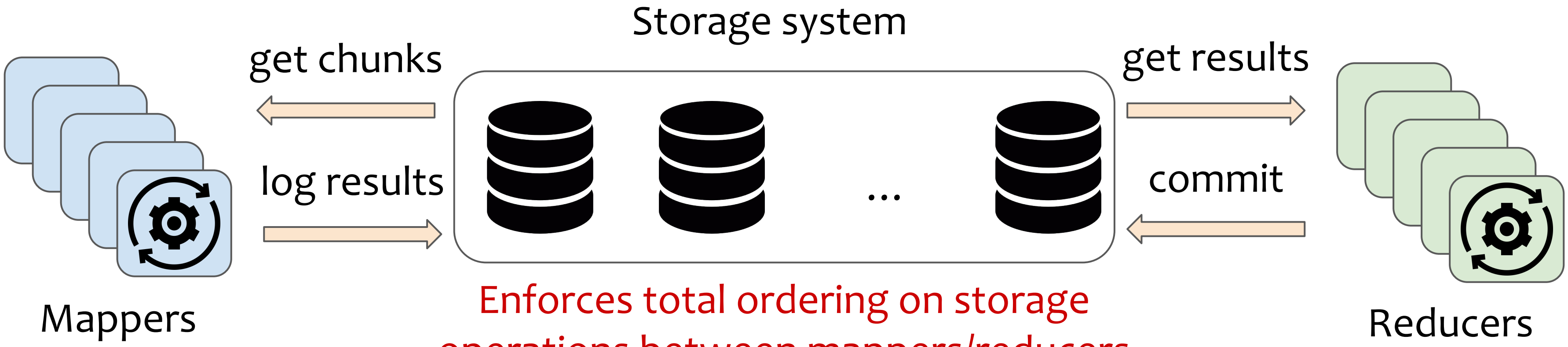
- Storage access for data persistence
  - ~40% of execution is all about storage
- Low-latency
  - Functions are short-lived, < 1 sec
- Distribution and flexible consistency
  - e.g., chained applications

Amazon S3 object store

Google cloud storage

Serverless workloads require fast storage systems with configurable semantics

Strict ordering in serverless chained applications

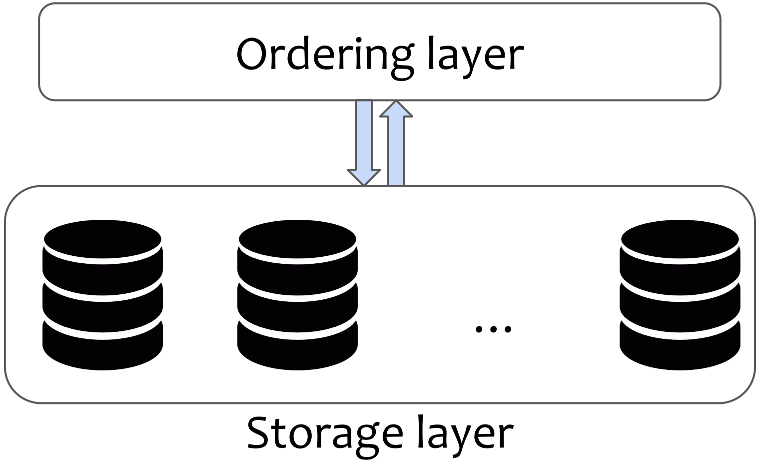


Enforces total ordering on storage operations between mappers/reducers

Total ordering is unnecessarily strict for serverless applications

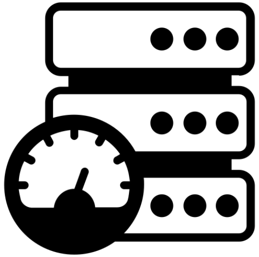
The beauty of shared log abstraction

- Distributed storage system
  - Append-only sequence of records
- Fault tolerance
- Strong programming model
  - Put/Get on append-only memory
- Performance (and scalability)
  - Consensus hidden behind the API

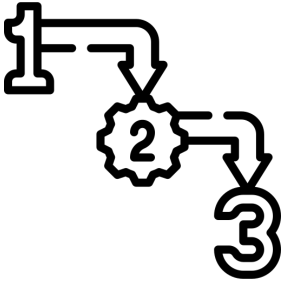


Shared logs can benefit serverless in terms of performance and semantics

Challenges



#1: Fast storage access



#2: Fast (and flexible) ordering

Problem statement

How to design a **fast** storage system **with flexible ordering**

for serverless computing infrastructure?

Our proposal

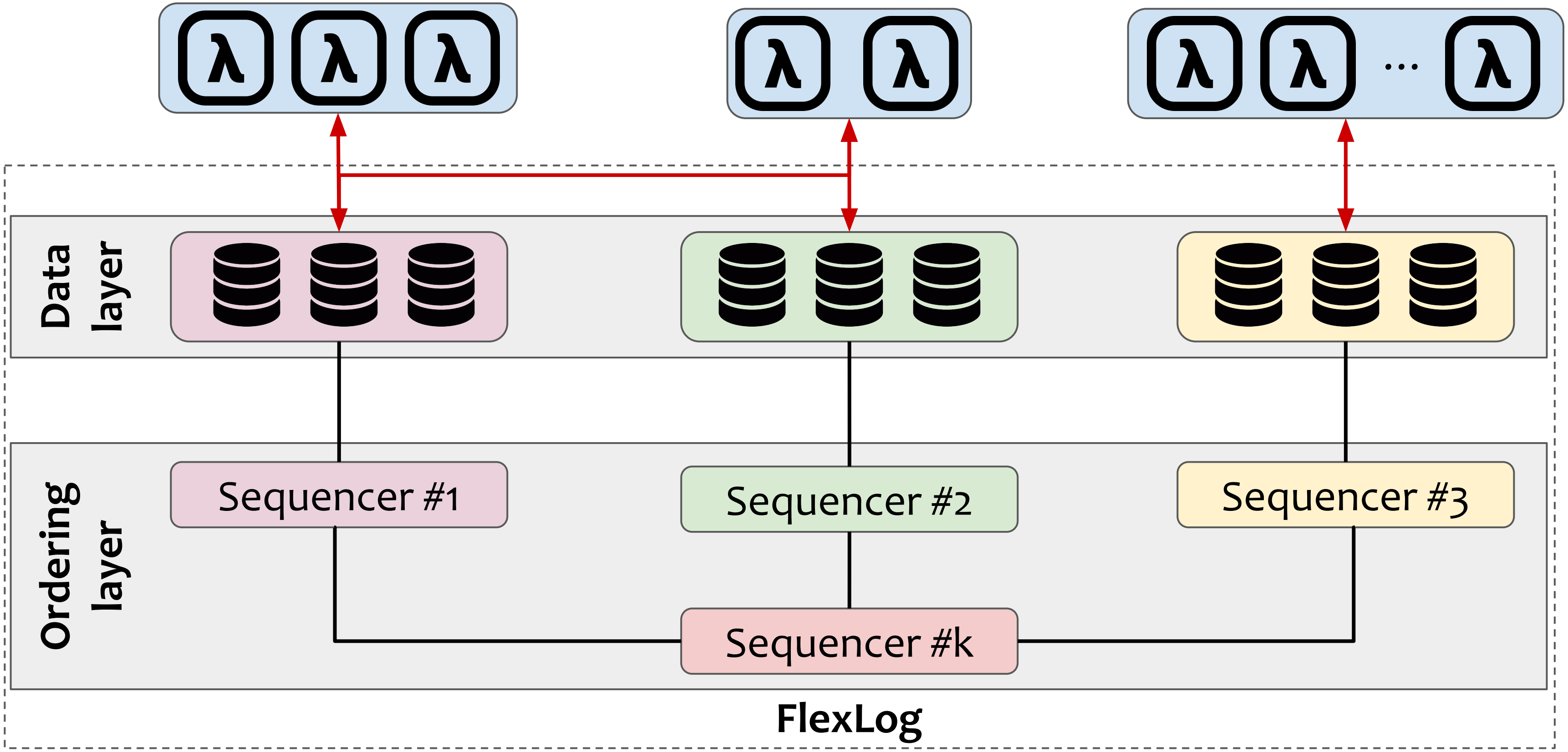
FlexLog
A Shared Log for Stateful Serverless Computing

Properties:

- Performance
- Flexible ordering semantics
- Formally proven consistency

Check the paper for the formal proof!

FlexLog overview

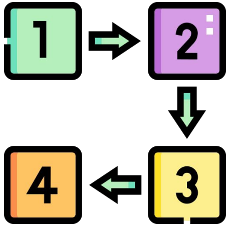


Outline

- ~~Motivation~~
- System components
- Example execution
- Evaluation

FlexLog

Data layer



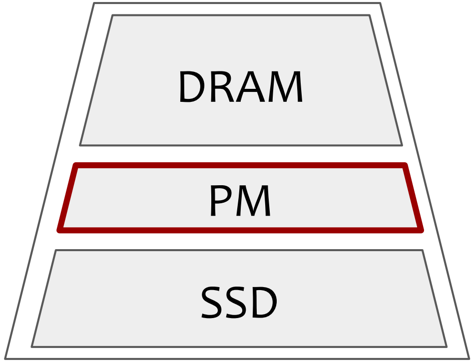
Ordering layer

FlexLog builds a fast data layer and a flexible ordering layer

Storage layer design

- Persistent memory (PM):
  - Durability
  - Low-latency I/O

PM transactions (TXs) for crash-consistency

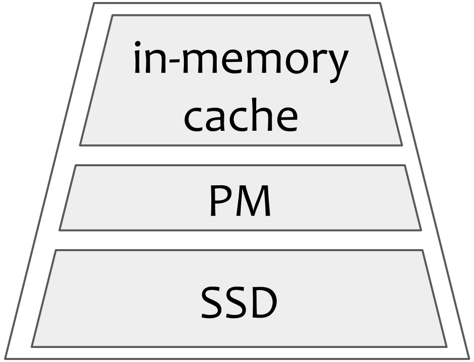


Memory and storage hierarchy

Storage layer design

- Persistent memory (PM):
  - Durability
  - Low-latency I/O

- (Storage) Replica:
  - In-memory cache
  - PM for the log
  - SSD to checkpoint and truncate



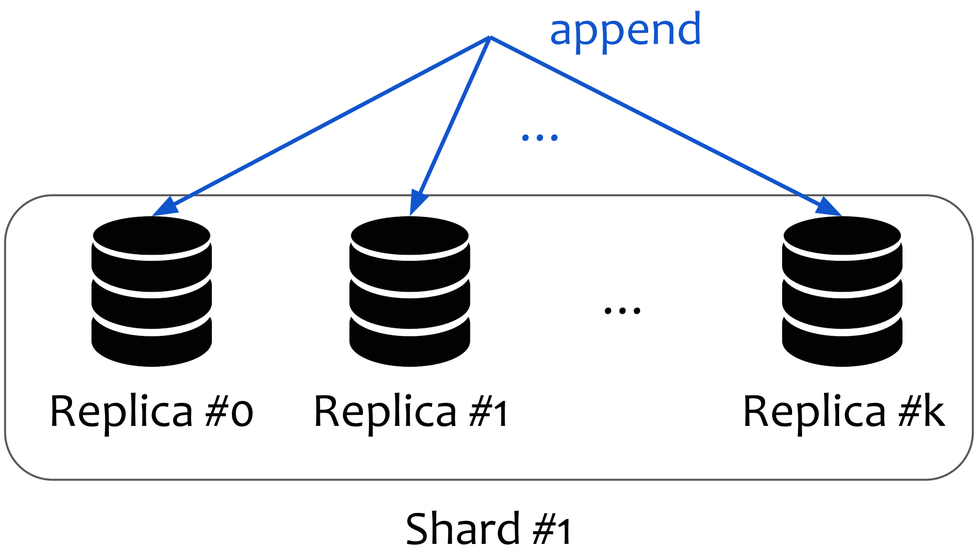
Replica storage stack

Replicas integrate PM for fast I/O and run PM-TXs for crash consistency

Data layer design

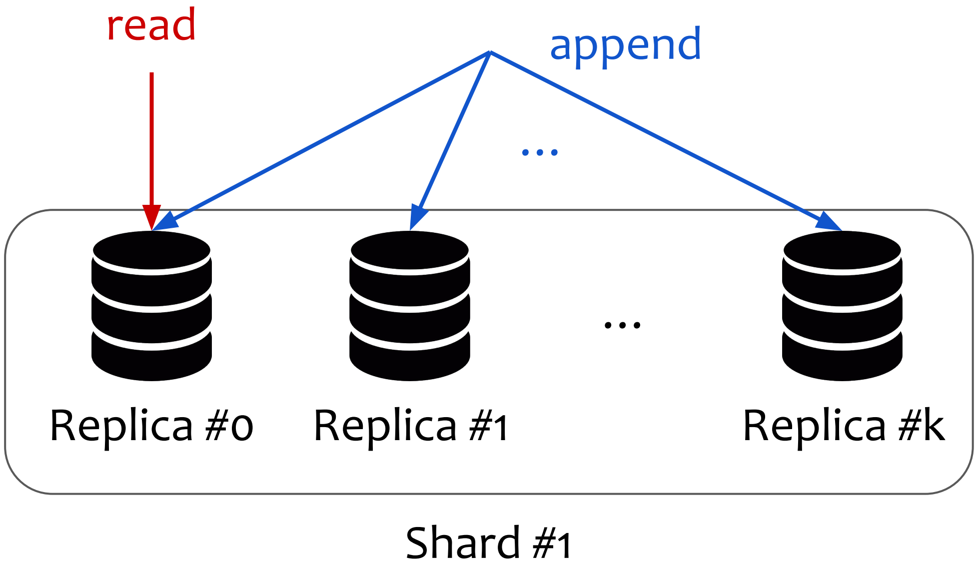
- Shard
  - A set of (storage) replicas

- Write-all/read-one protocol
  - Local lin. reads



Data layer design

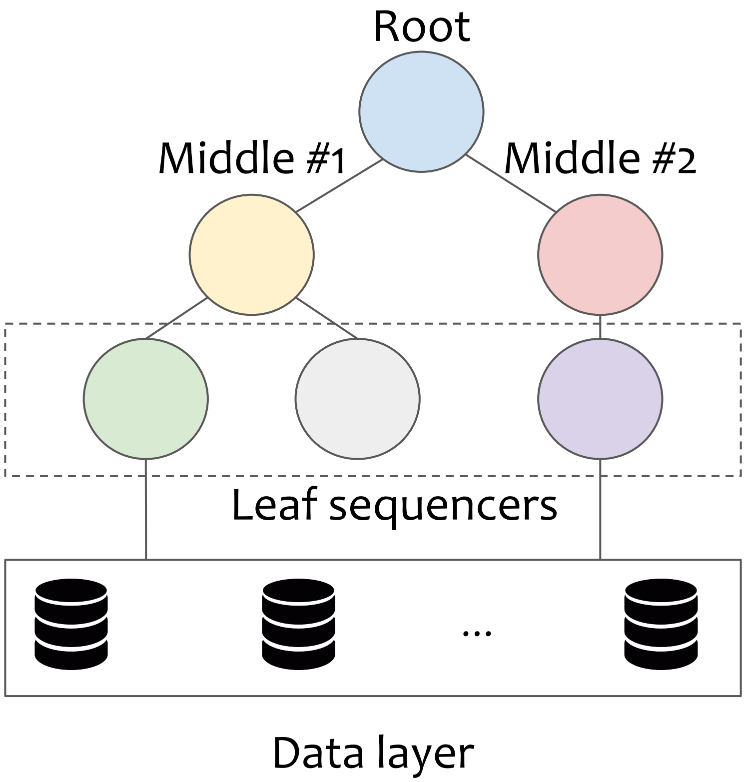
- Shard
  - A set of (storage) replicas
- Write-all/read-one protocol
  - Local lin. reads



Replicas execute a write-all/read-one replication protocol for performance

Ordering layer design

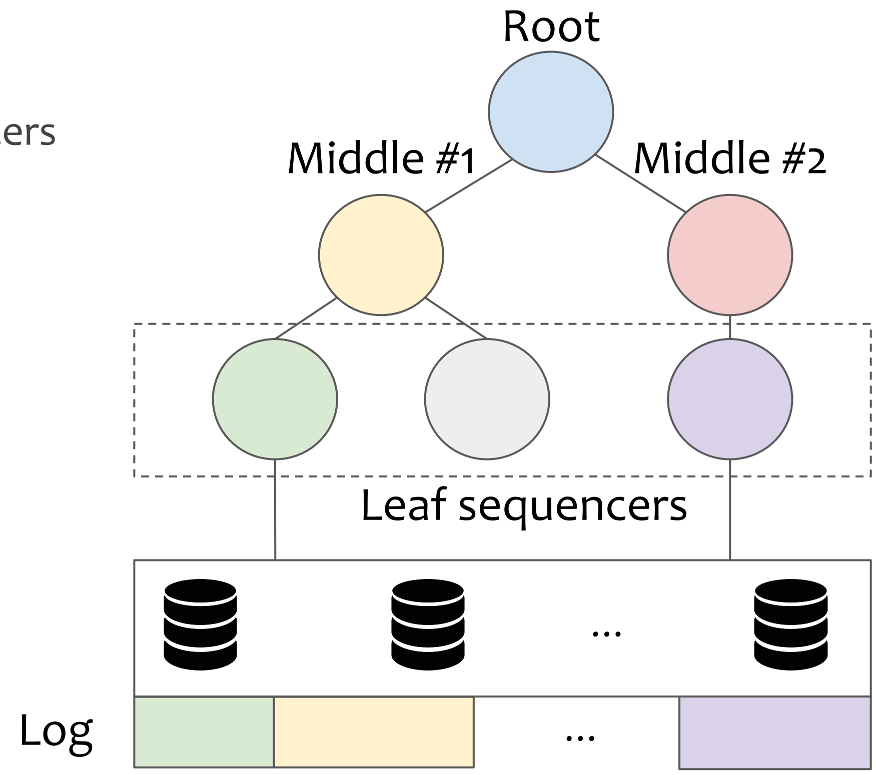
- Sequencers in a tree hierarchy
  - Shards communicate w/ leaf sequencers
- Color abstraction
  - Denote ordering semantics
- Sequence number (SN)



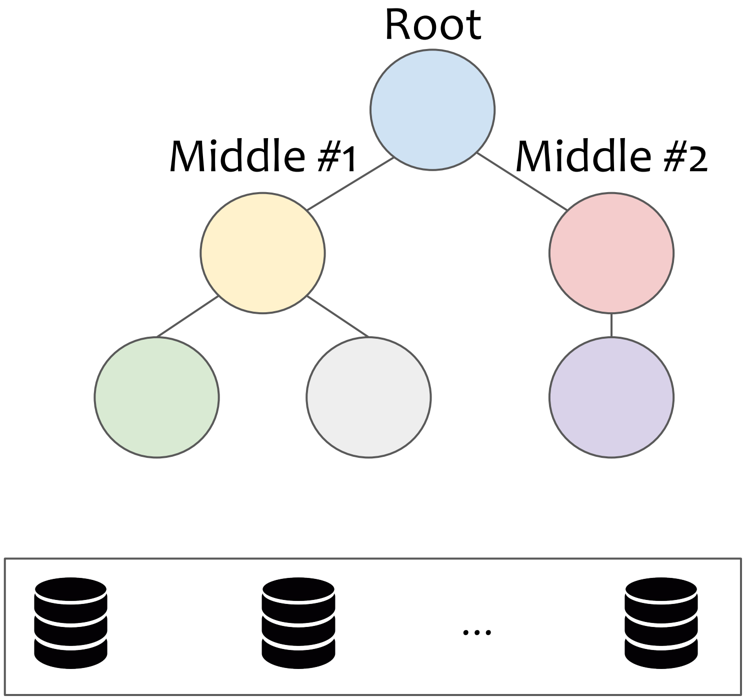
Ordering layer design

- Sequencers in a tree hierarchy
  - Shards communicate w/ leaf sequencers
- Color abstraction
  - Denote ordering semantics
- Sequence number (SN)

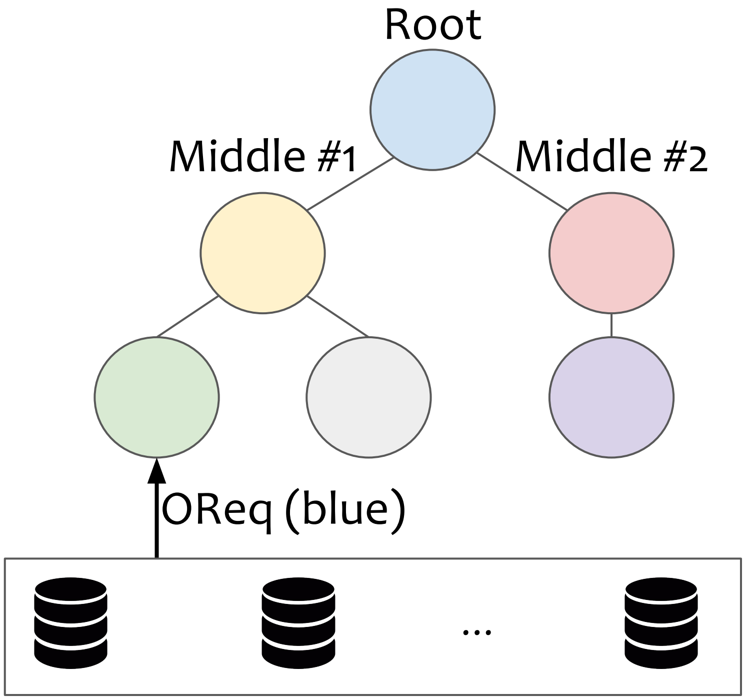
- SNs from different sequencers are unrelated
  - Division into independent regions



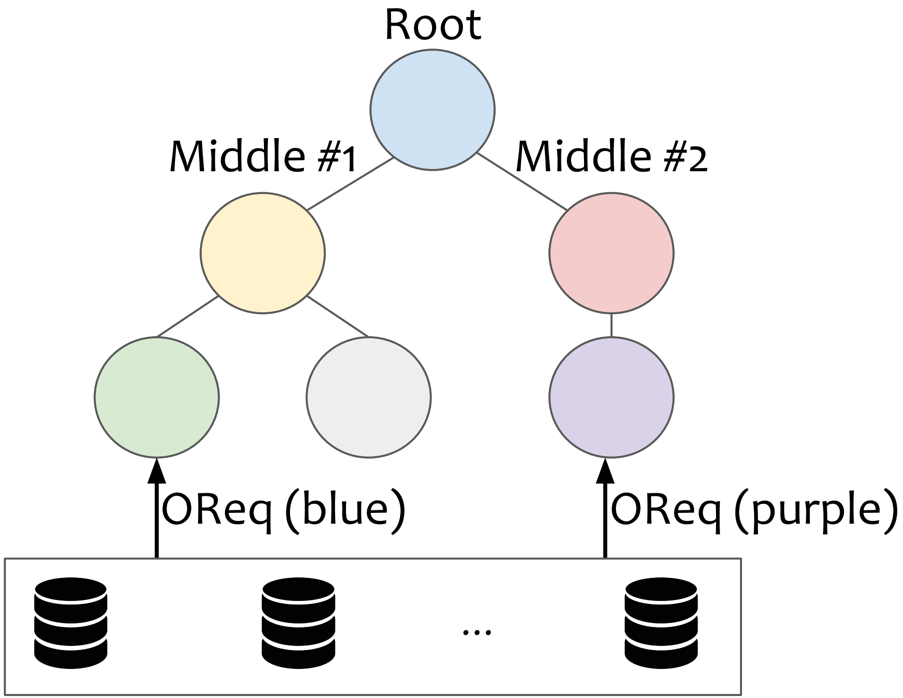
Ordering layer design



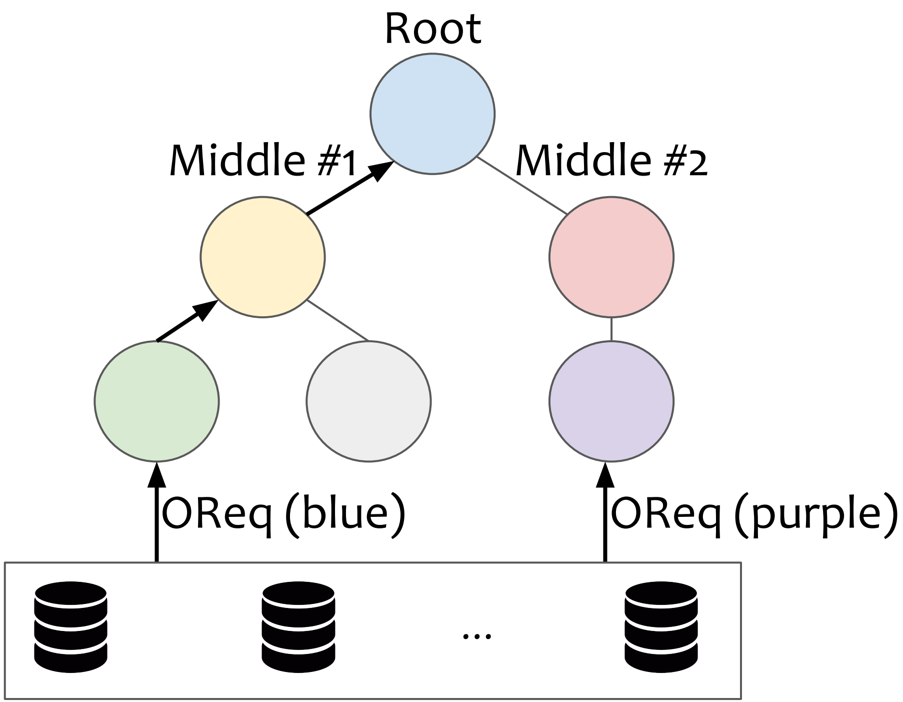
Ordering layer design



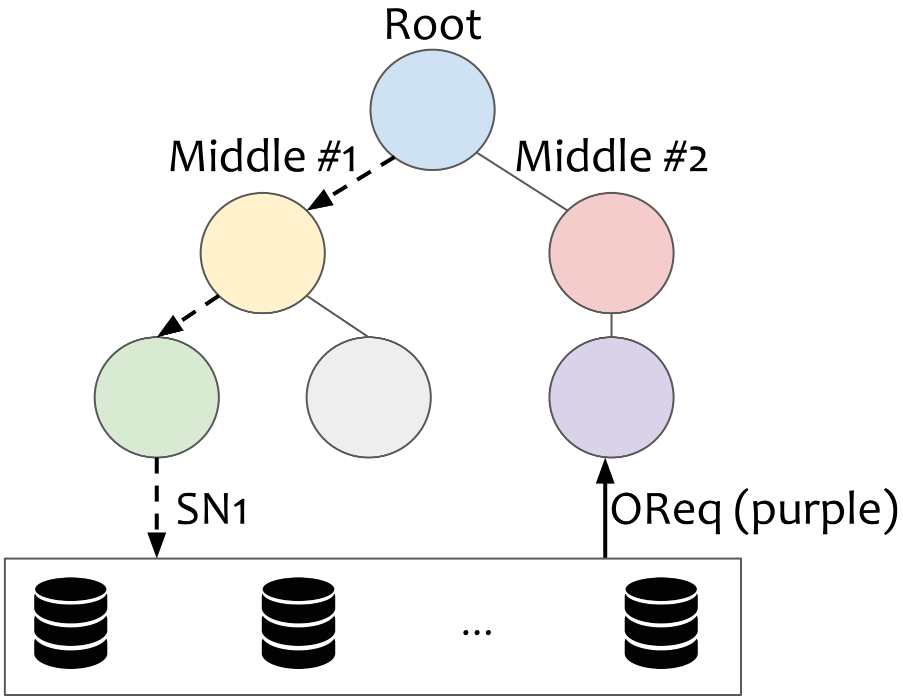
Ordering layer design



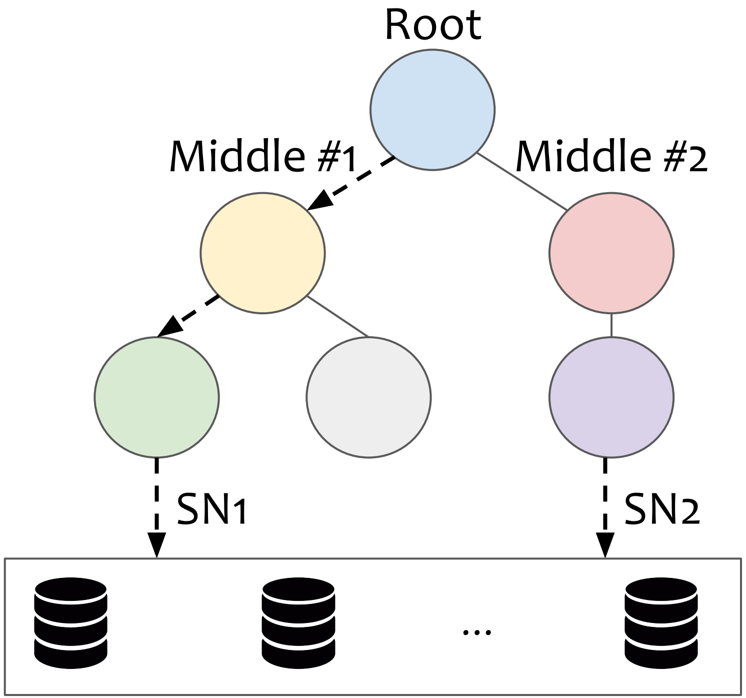
Ordering layer design



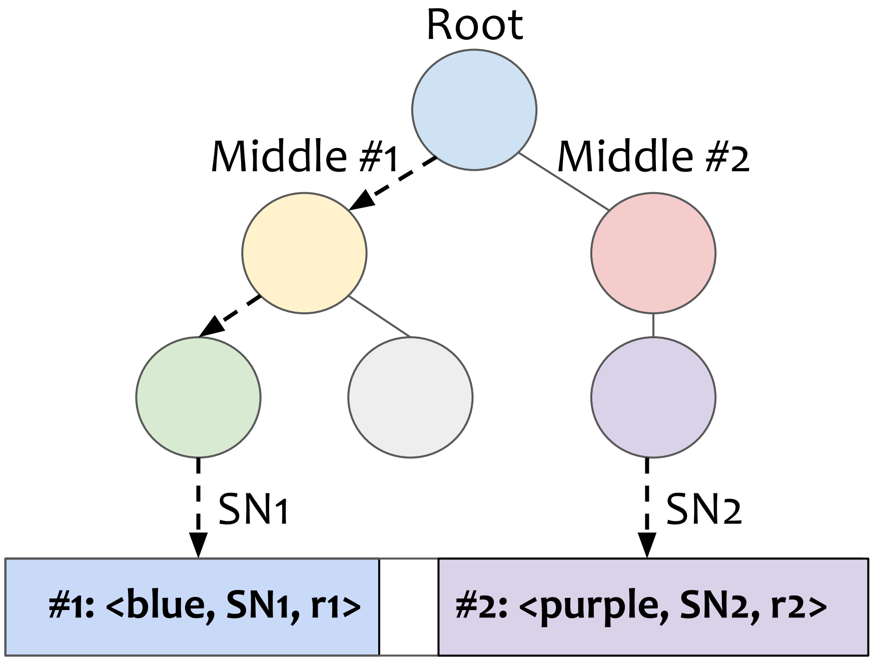
Ordering layer design



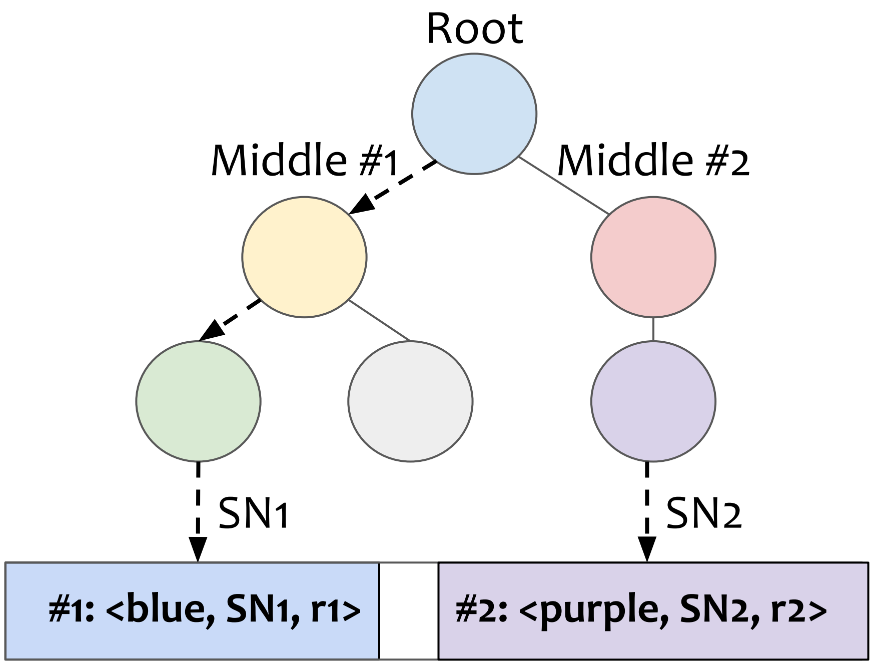
Ordering layer design



Ordering layer design



Ordering layer design



FlexLog ordering layer allows for per-color ordering to boost performance

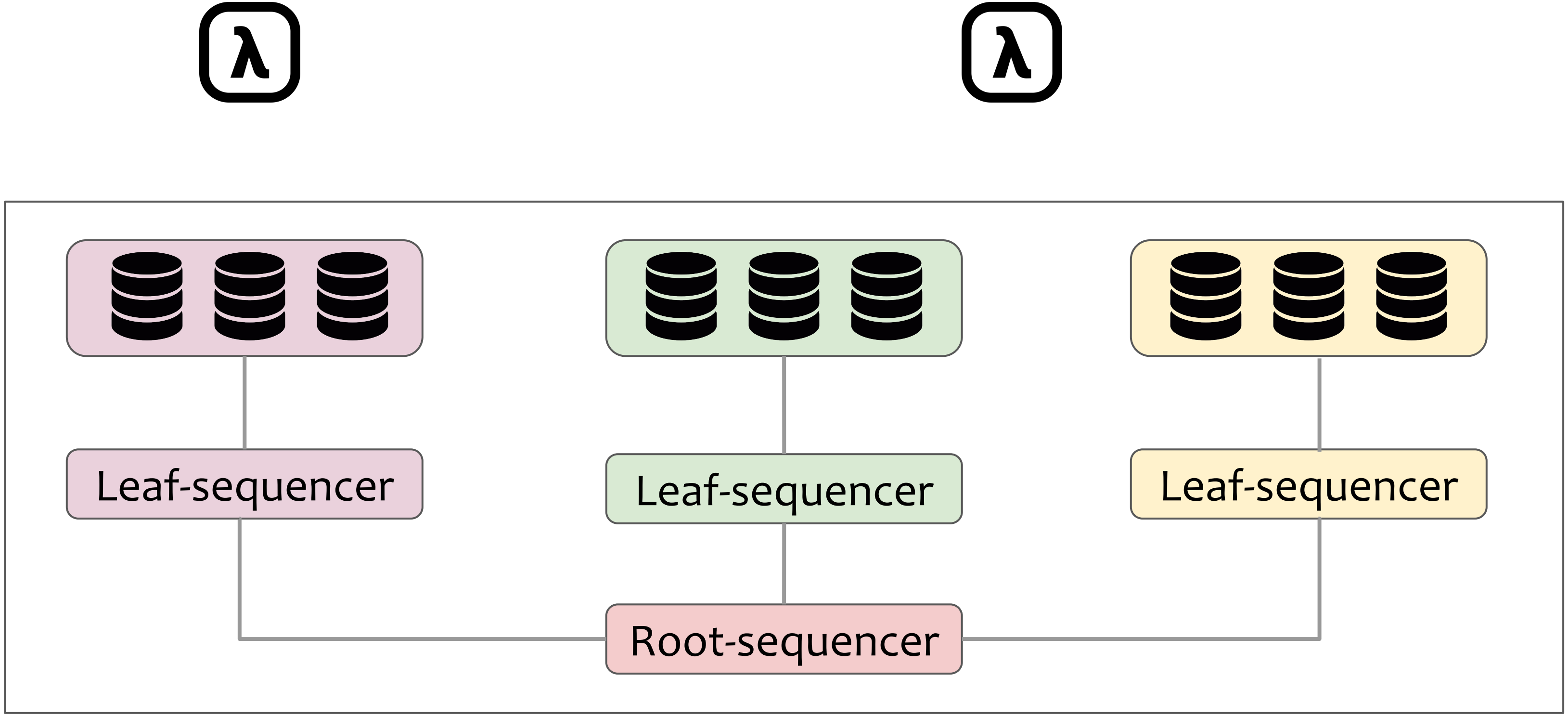
Outline

- ~~Motivation~~
- ~~System components~~
- Example execution
- Evaluation

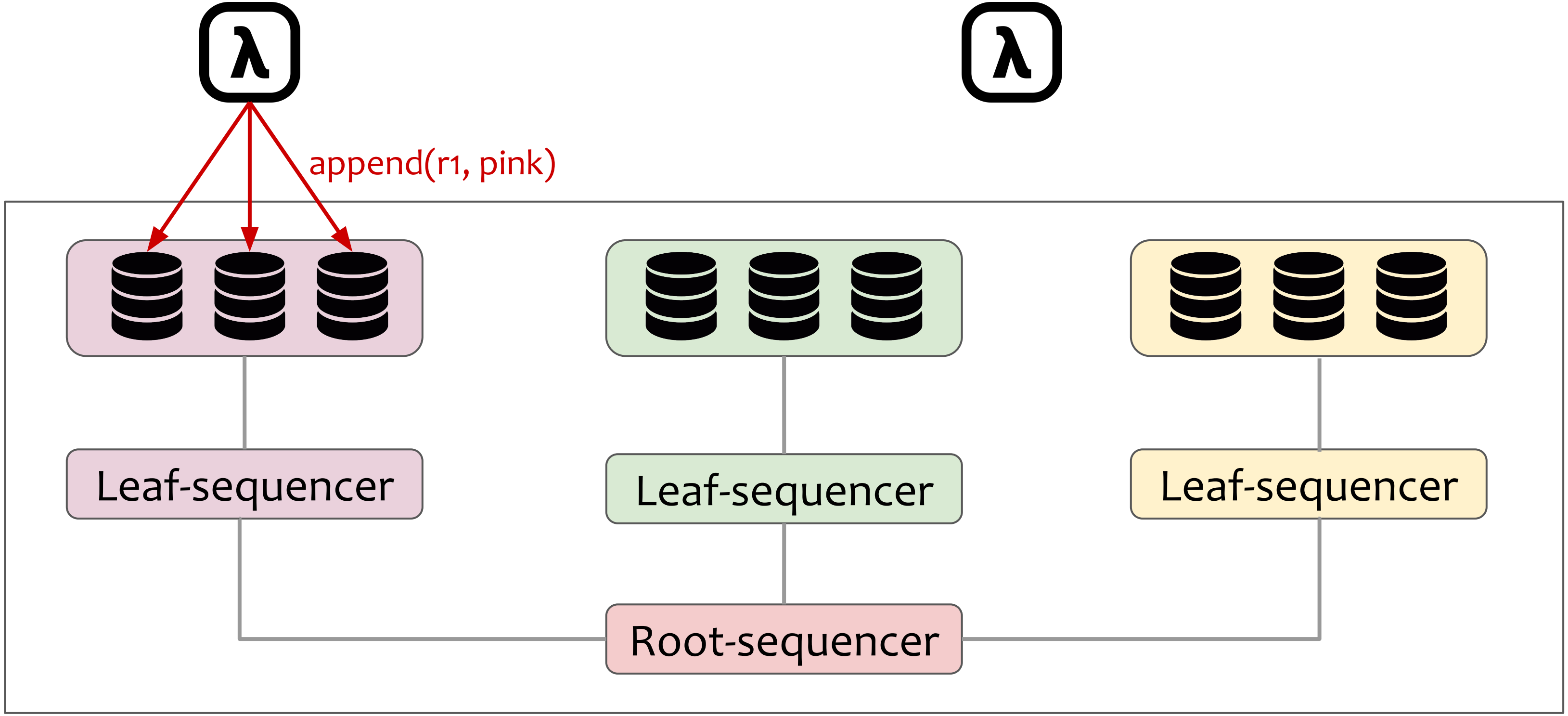
FlexLog API

append(records[], color)	Appends records and returns SN upon completion
read(SN, color)	Reads a record with SN from the color-ed log
subscribe(color)	Receives all records of the color-ed log
trim(SN, color)	Garbage collects the log of color-ed log by deleting all records with sn ≤ SN

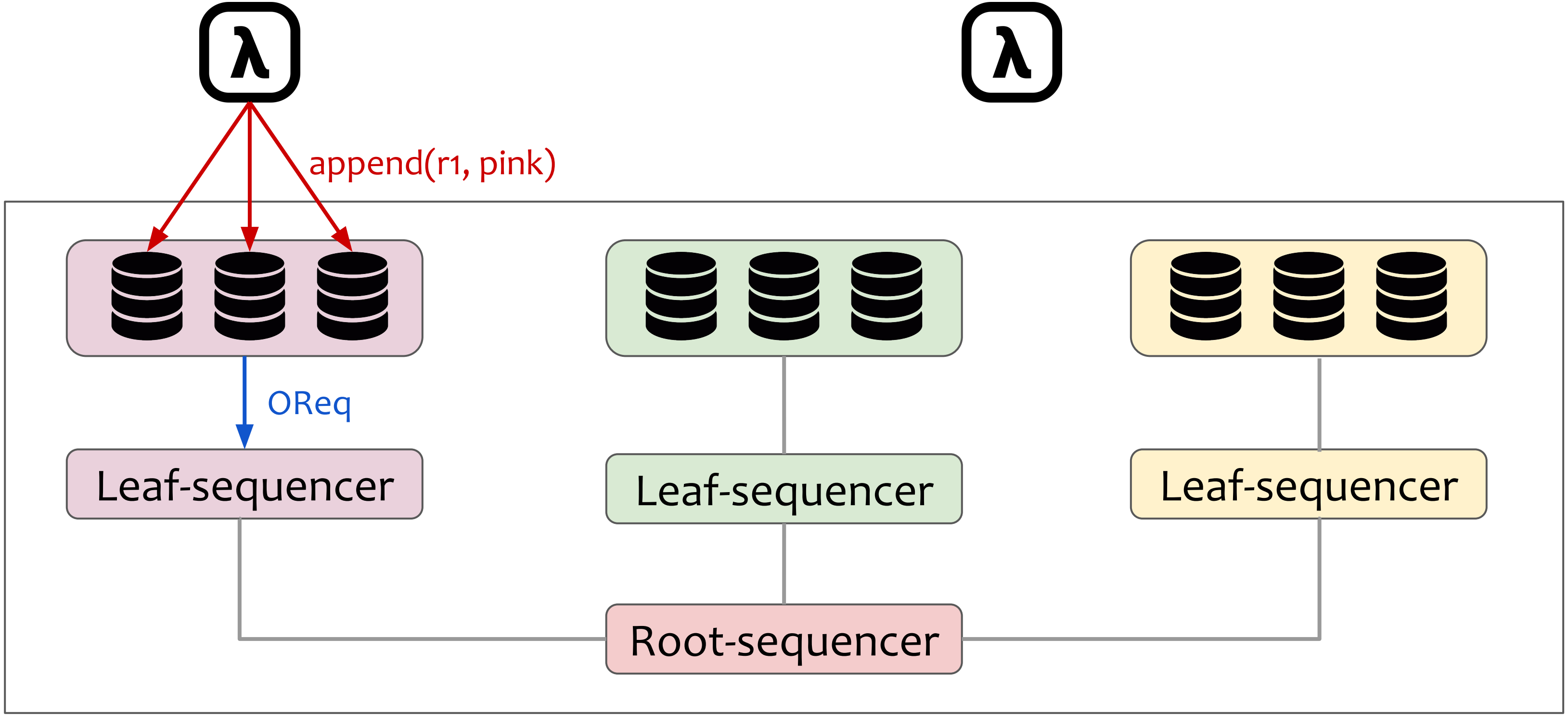
FlexLog in action



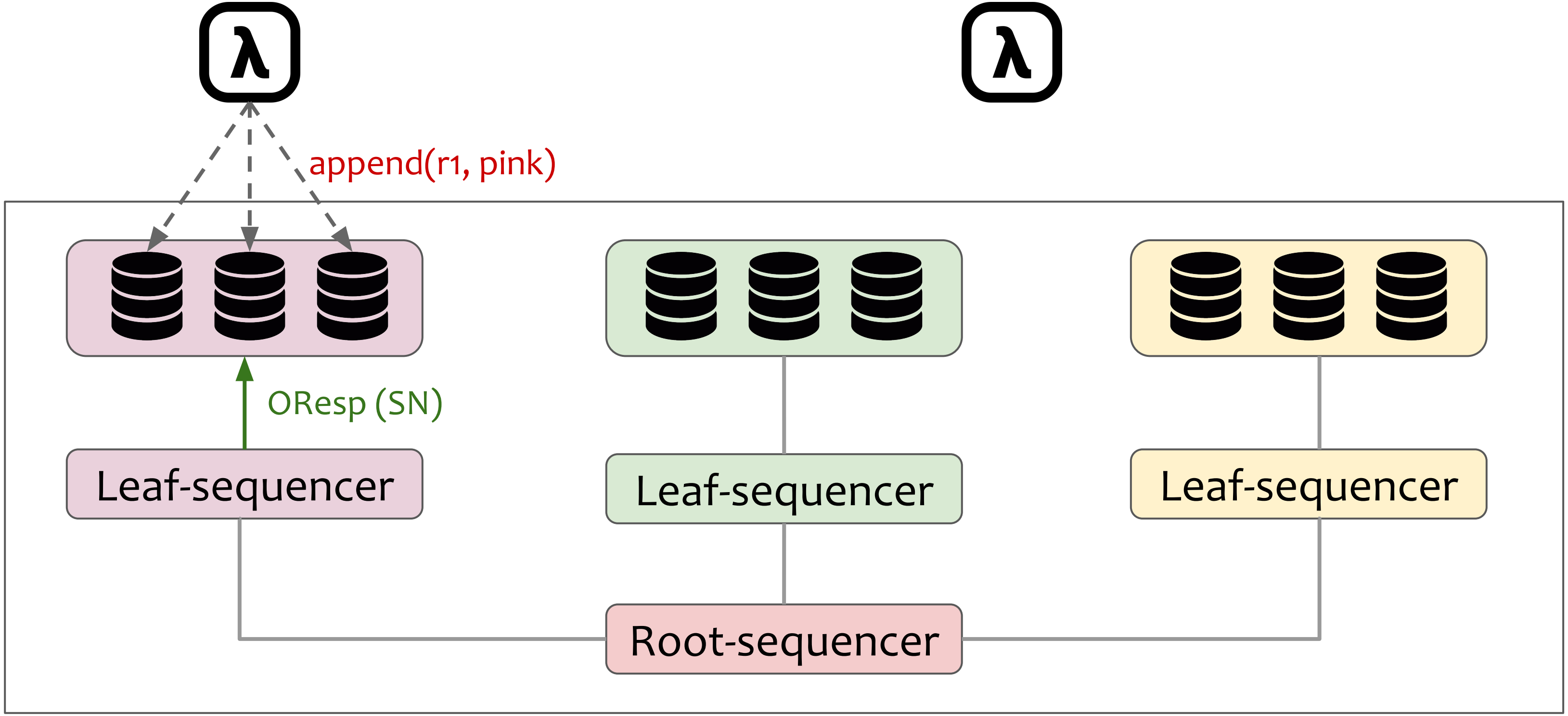
FlexLog in action



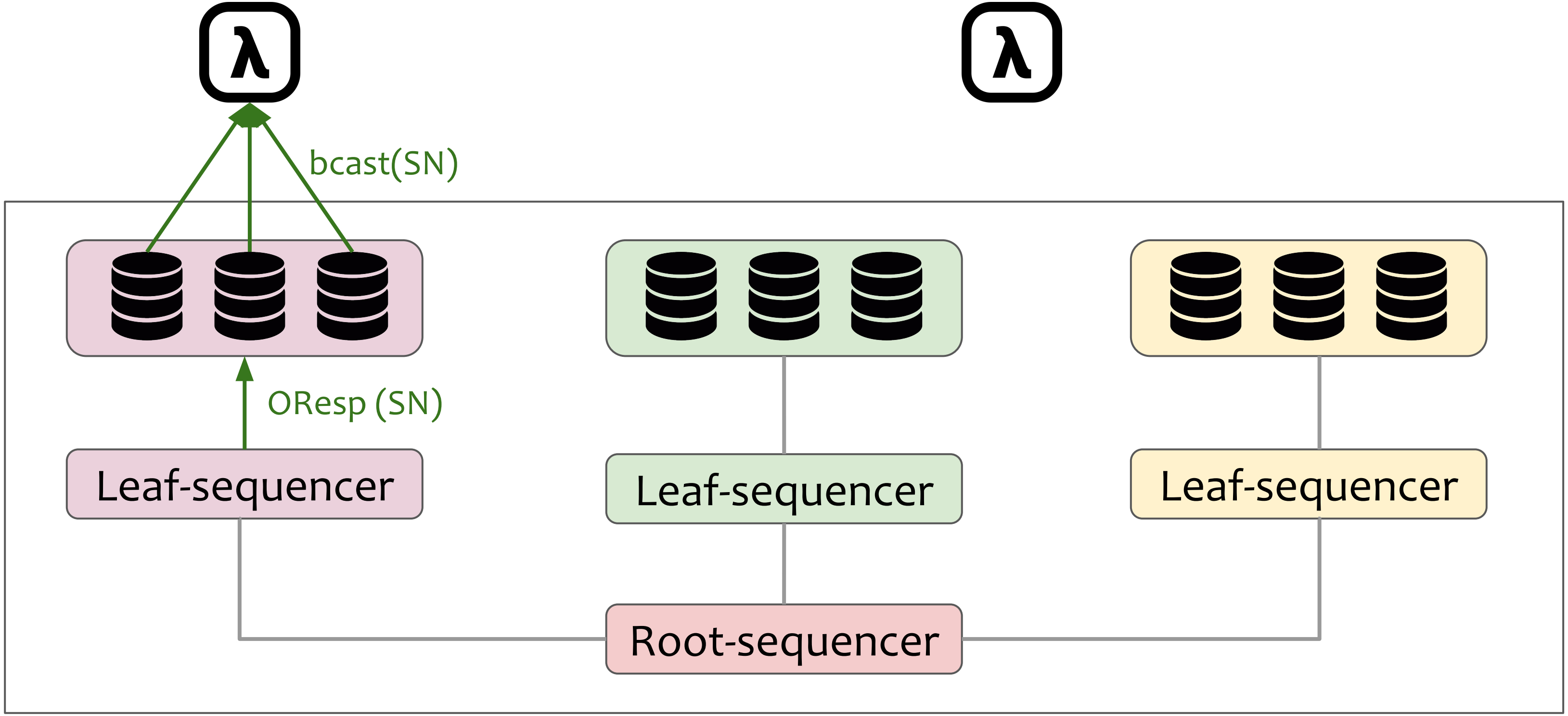
FlexLog in action



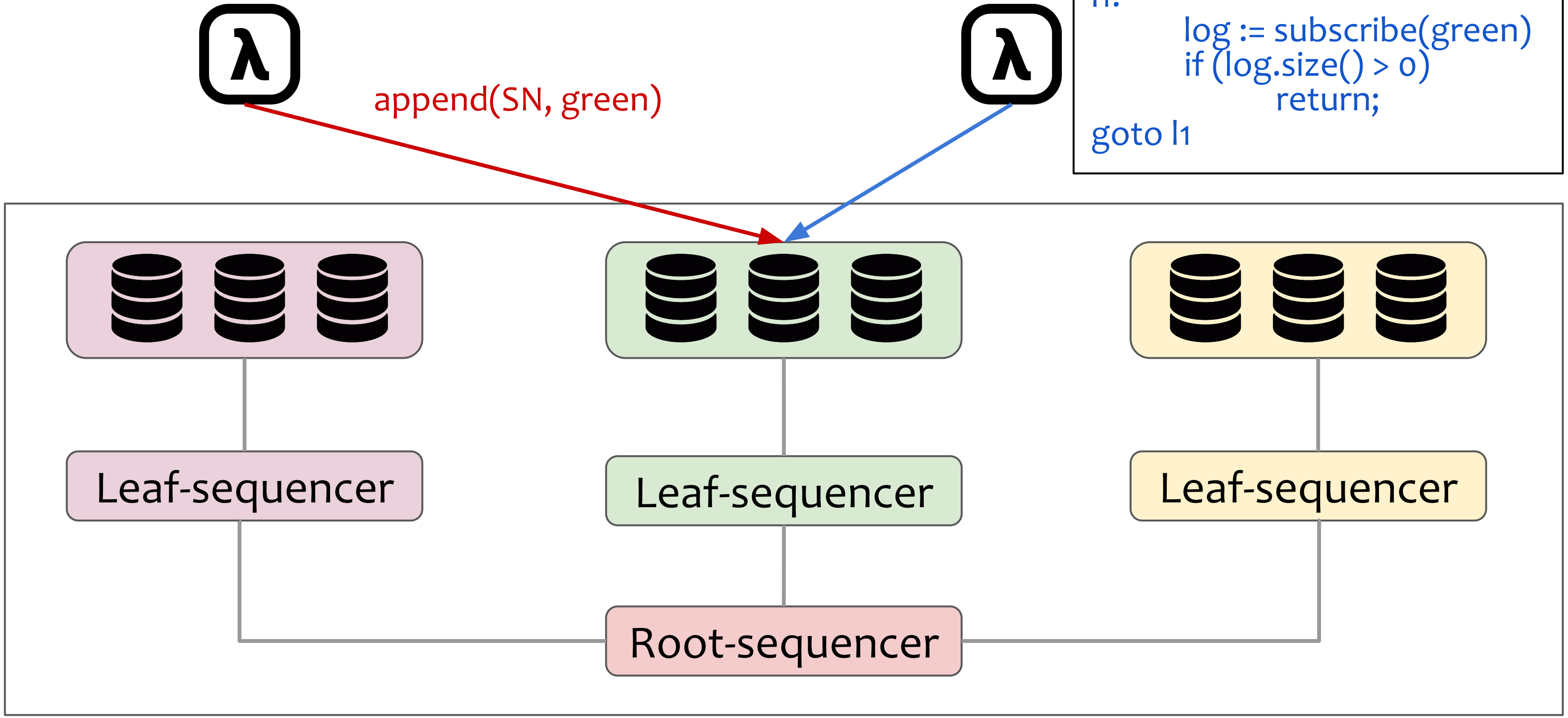
FlexLog in action



FlexLog in action

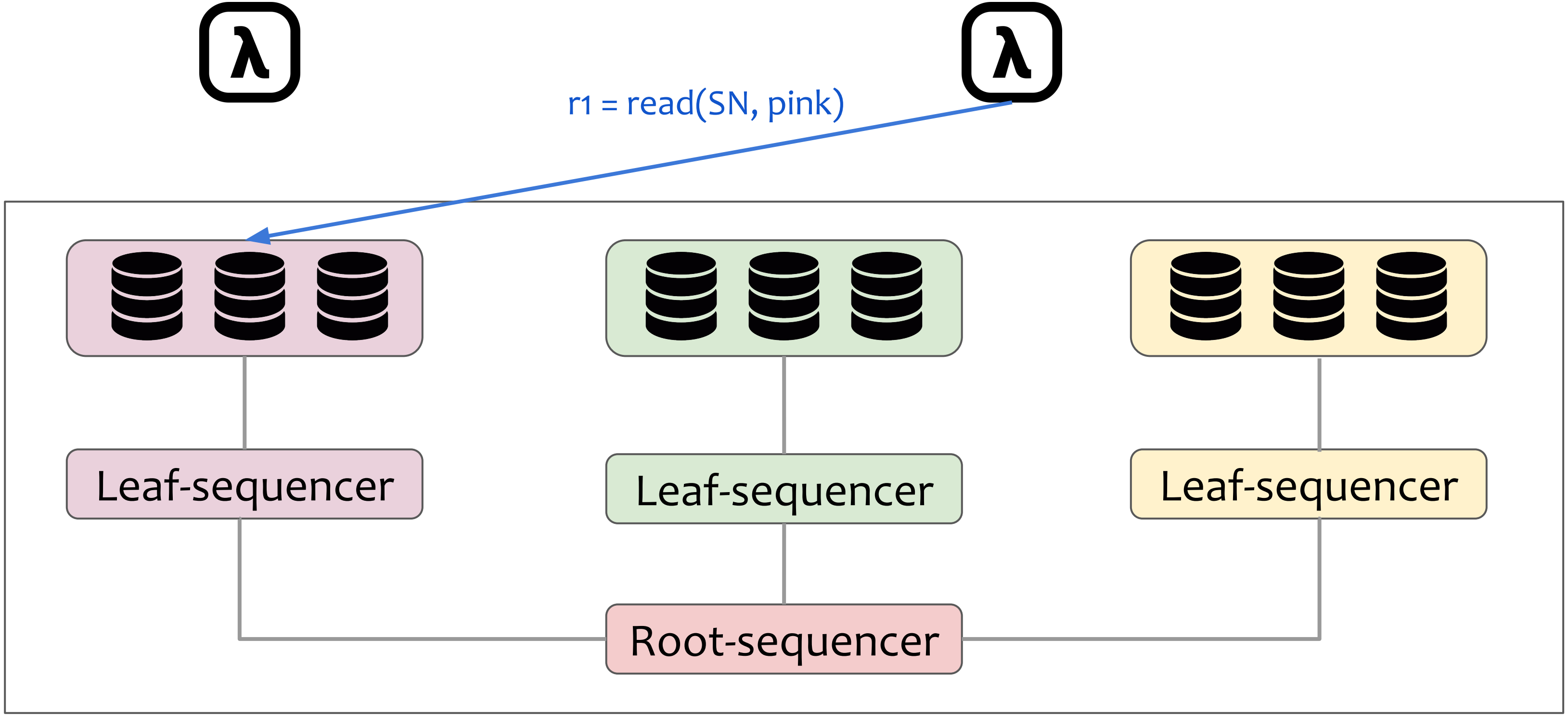


FlexLog in action




```
l1:
        log := subscribe(green)
        if (log.size() > 0)
                return;
goto l1
```

FlexLog in action



Outline

- ~~Motivation~~
- ~~System components~~
- ~~Example execution~~
- Evaluation

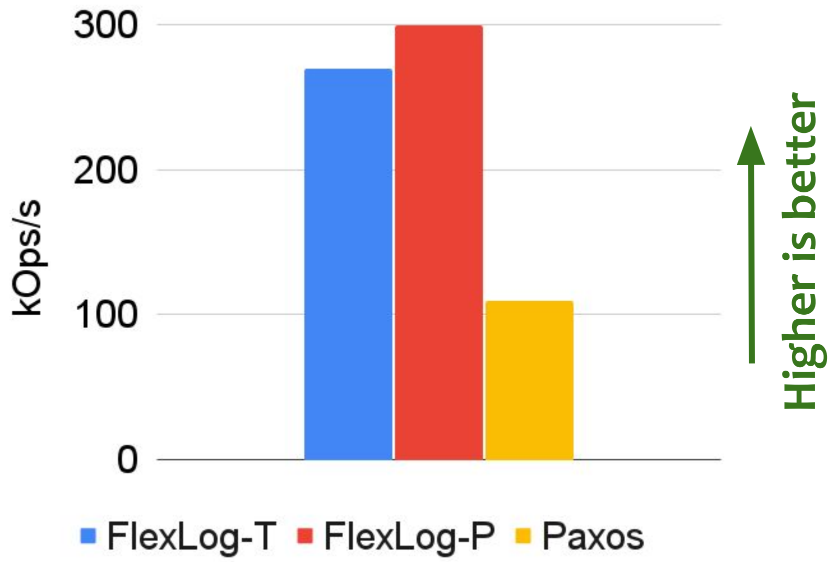
Evaluation

- Implementation
  - PMDK (libpmemobj++)
  - gRPC for networking

- Questions
  - What is the ordering layer's performance?
  - What is the storage layer's performance?

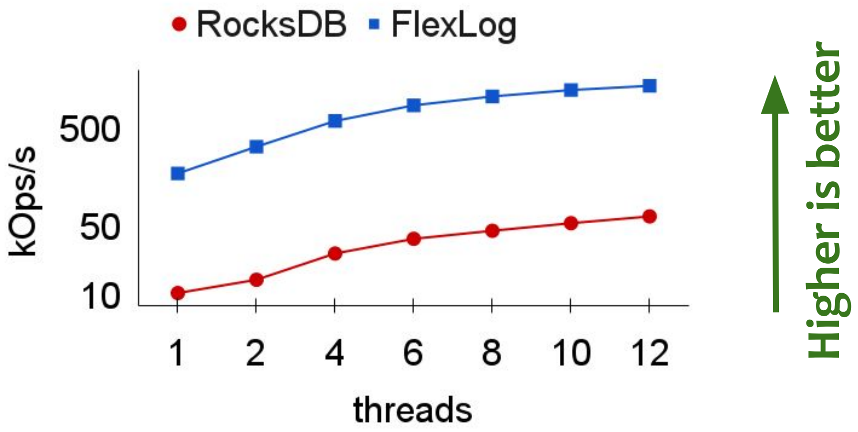
- H/W setup: 800 GB Intel Optane DC PM (x6) over a 10Gbps network

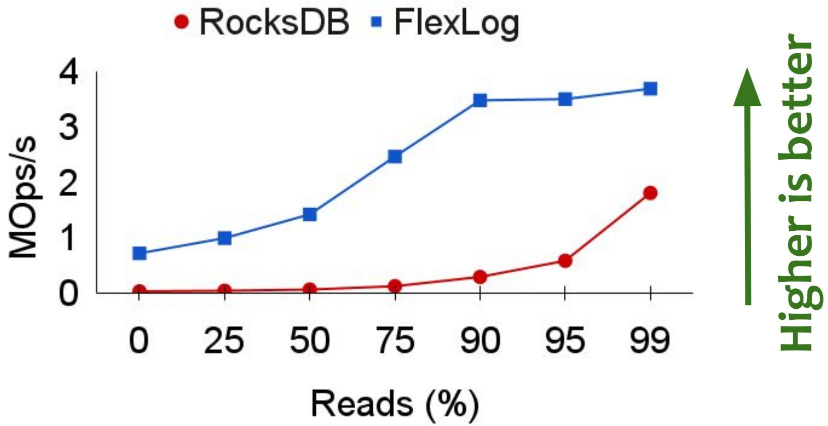
Q1: Ordering layer performance



FlexLog ordering is up to 3x faster w.r.t. the state-of-the-art

Q2: Storage layer performance





FlexLog storage layer performs up to 10x better w.r.t. the state-of-the-art

Summary

General purpose storage systems are not well-fitted to serverless computing

- Limited performance due to slow I/O (SSDs)
- Strict and expensive total ordering

FlexLog: A Shared Log for Stateful Serverless Computing

- Builds a data layer on top of fast PM
- Builds a fast and flexible ordering layer

Source code: https://github.com/TUM-DSE/FlexLog