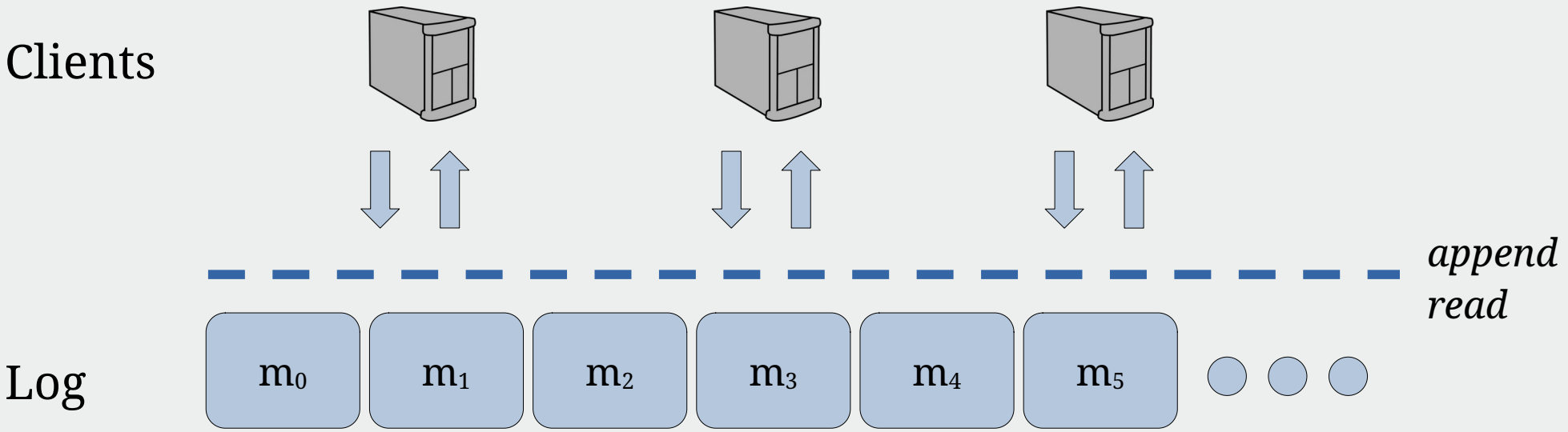
Designing a Log for the Datacenter

Micah Murray, Wen Zhang, Aisha Mushtaq,
Natacha Crooks, Aurojit Panda*, Scott Shenker
UC Berkeley *New York University

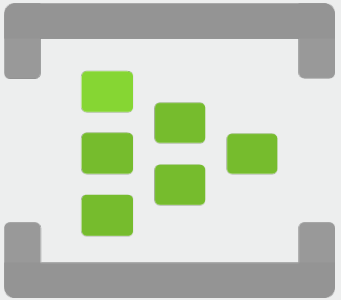
What are distributed shared logs?



A *simple* interface to totally order operations

Where are shared logs used?

kafka



Azure Event Hubs

Where are shared logs used?

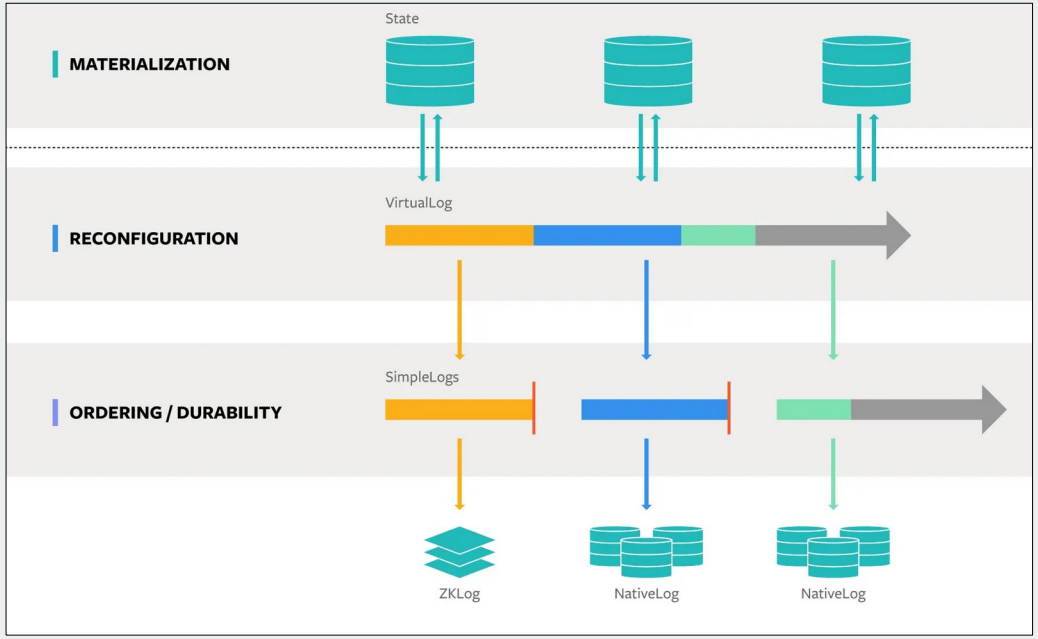
- In microservice architectures for interoperability
  - e.g., Heroku uses Kafka to message between microservices

Where are shared logs used?

- In Change-Data-Capture for interoperability
  - e.g., Debezium uses Kafka for streaming data

Where are shared logs used?

- In database systems for simplicity
  - e.g., Delos simplifies implementing distributed applications



Everyone has their own log...

Why not just have one?

<u>Proposal</u>*: One* log for the entire datacenter!

Challenge: Scale??

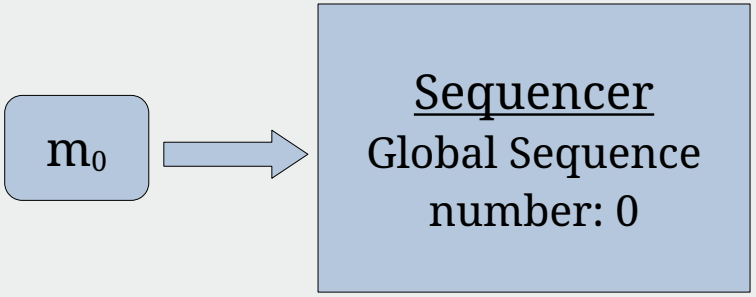
Log must serve ~10s of billions req/s

Problem: Prior solutions don’t support this throughput

Shared Logging Protocols

Sequencing

Replication

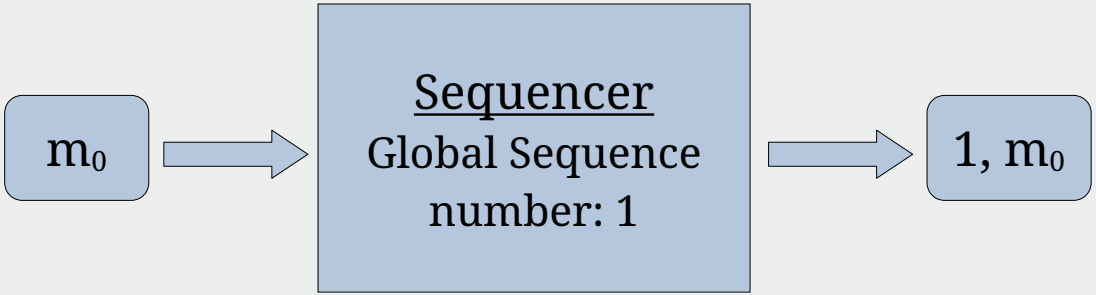


Sequencing: Enforces total order

Shared Logging Protocols

Sequencing

Replication

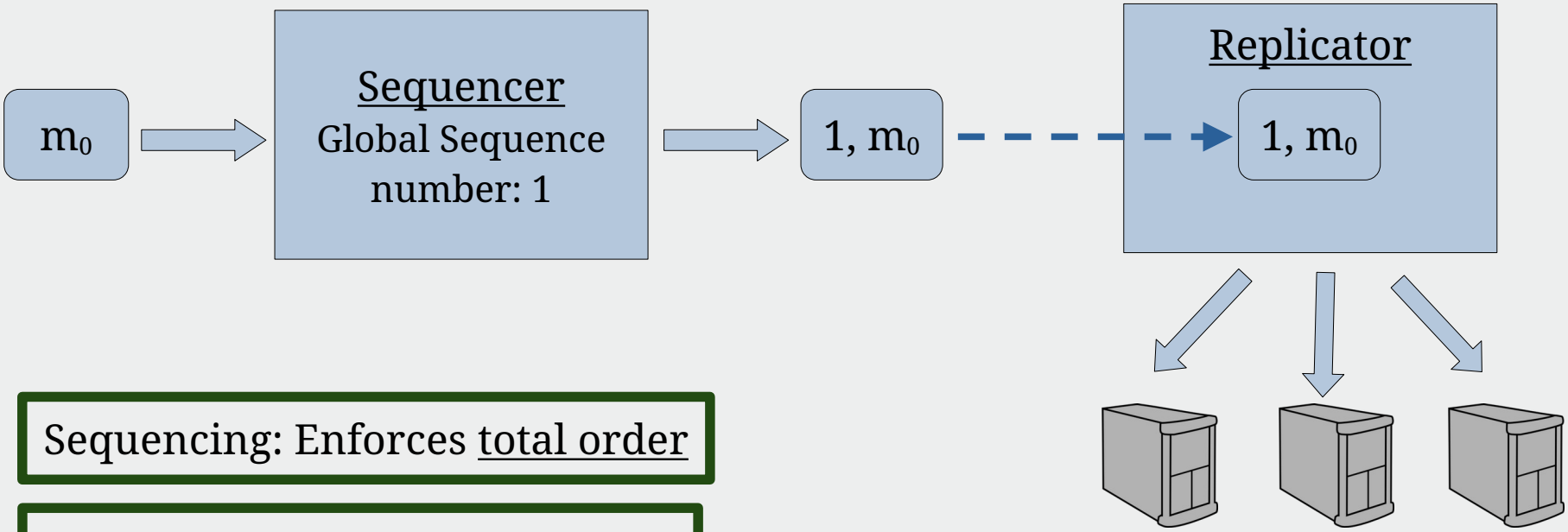


Sequencing: Enforces total order

Shared Logging Protocols

Sequencing

Replication

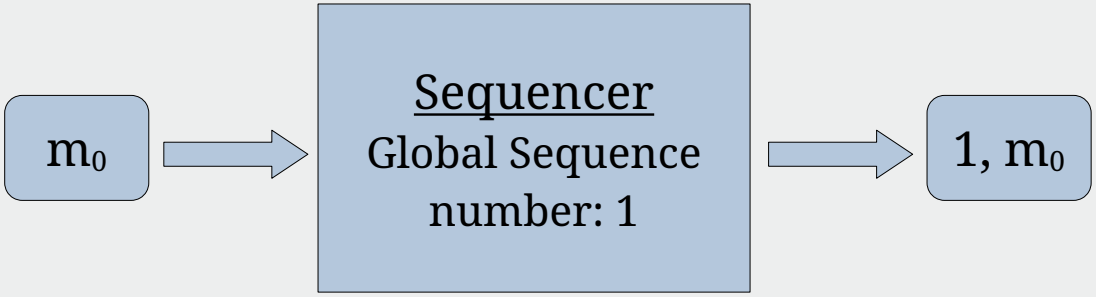


Sequencing: Enforces total order

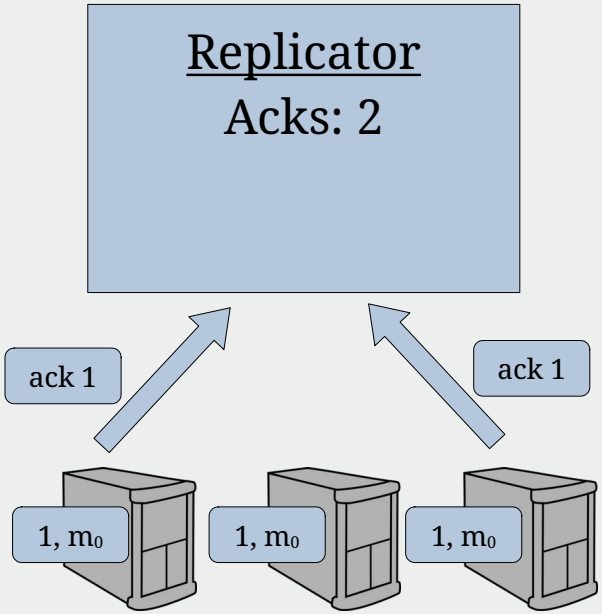
Replication: Ensures durability

Shared Logging Protocols

Sequencing



Replication

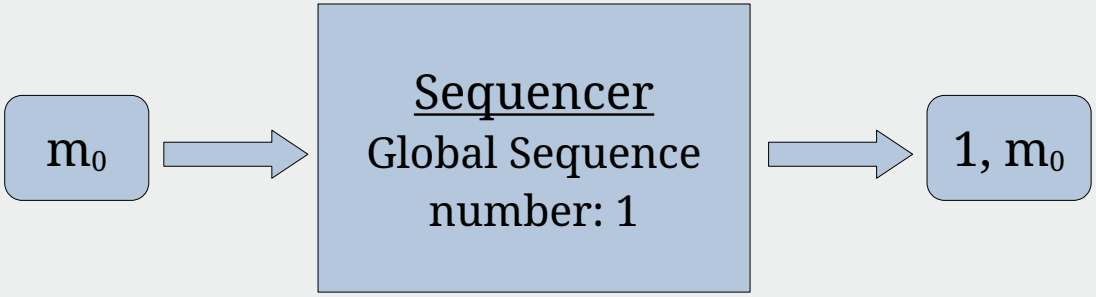


Sequencing: Enforces total order

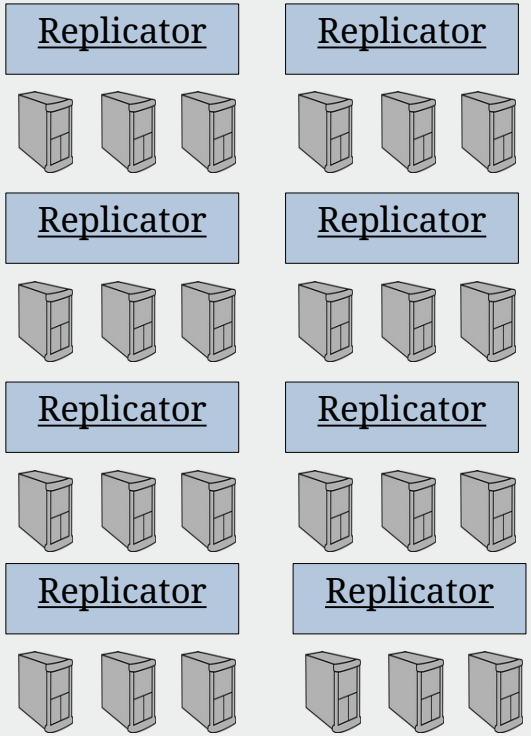
Replication: Ensures durability

Shared Logging Protocols

Sequencing



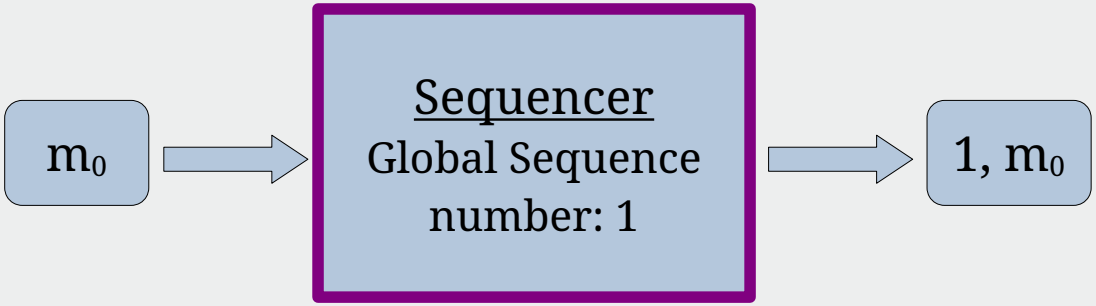
Replication



Replication: Infinitely parallelizable!

Shared Logging Protocols

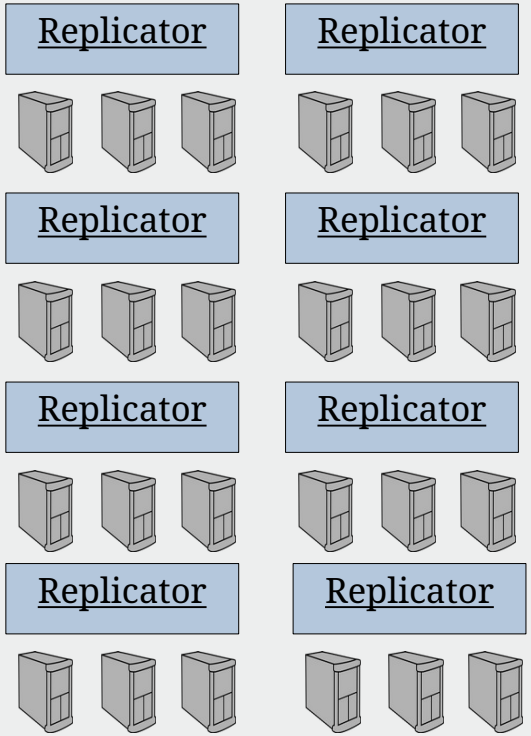
Sequencing



Sequencer: **Not parallelizable**!

Replication: Infinitely parallelizable!

Replication



Can we scale sequencing?

Insight #1: Sequencing is simple and purely compute

Action #1: ‘Cheat’ with specialized hardware

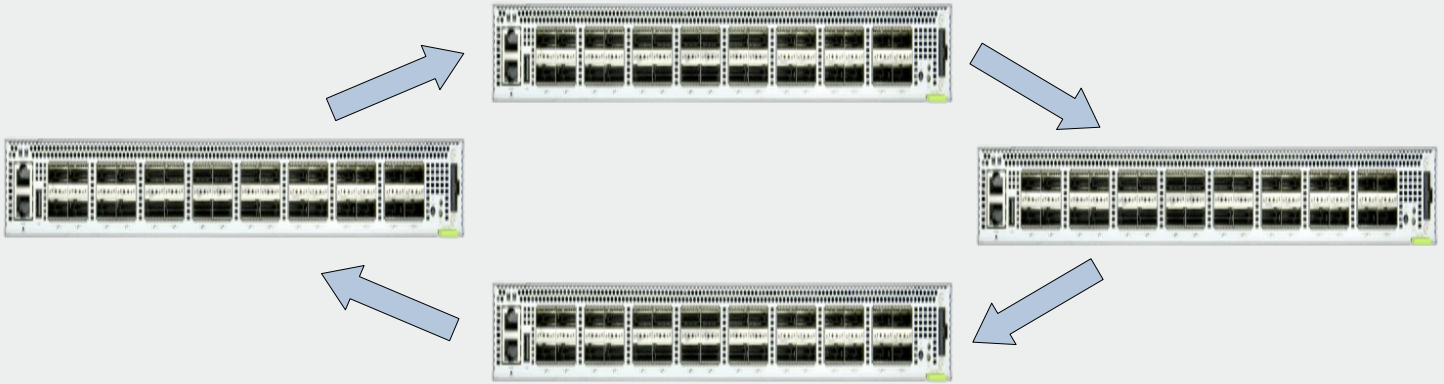
Can we scale sequencing infinitely?

Insight #2: Need multiple sequencer nodes

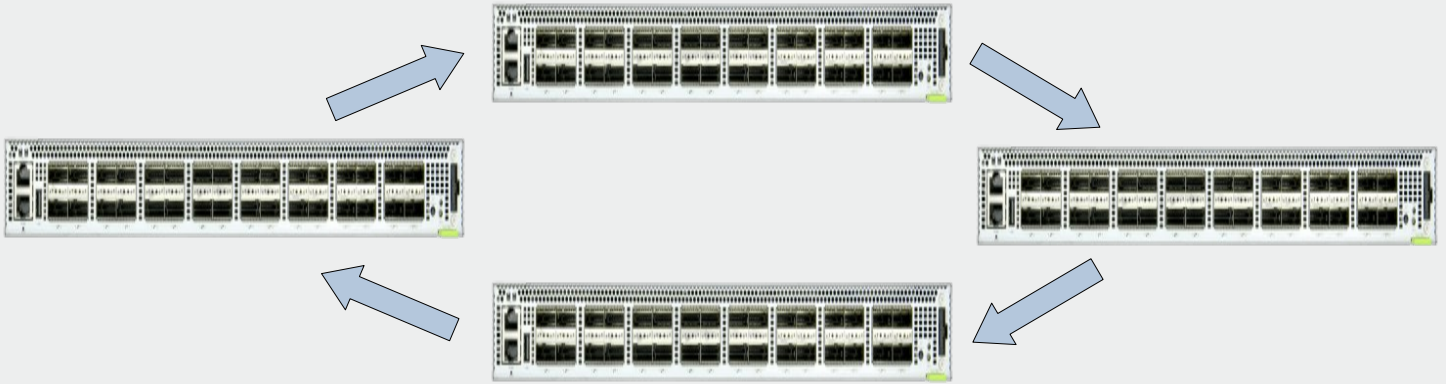
Action #2: Distribute sequencing using a ring

Which accelerated hardware?

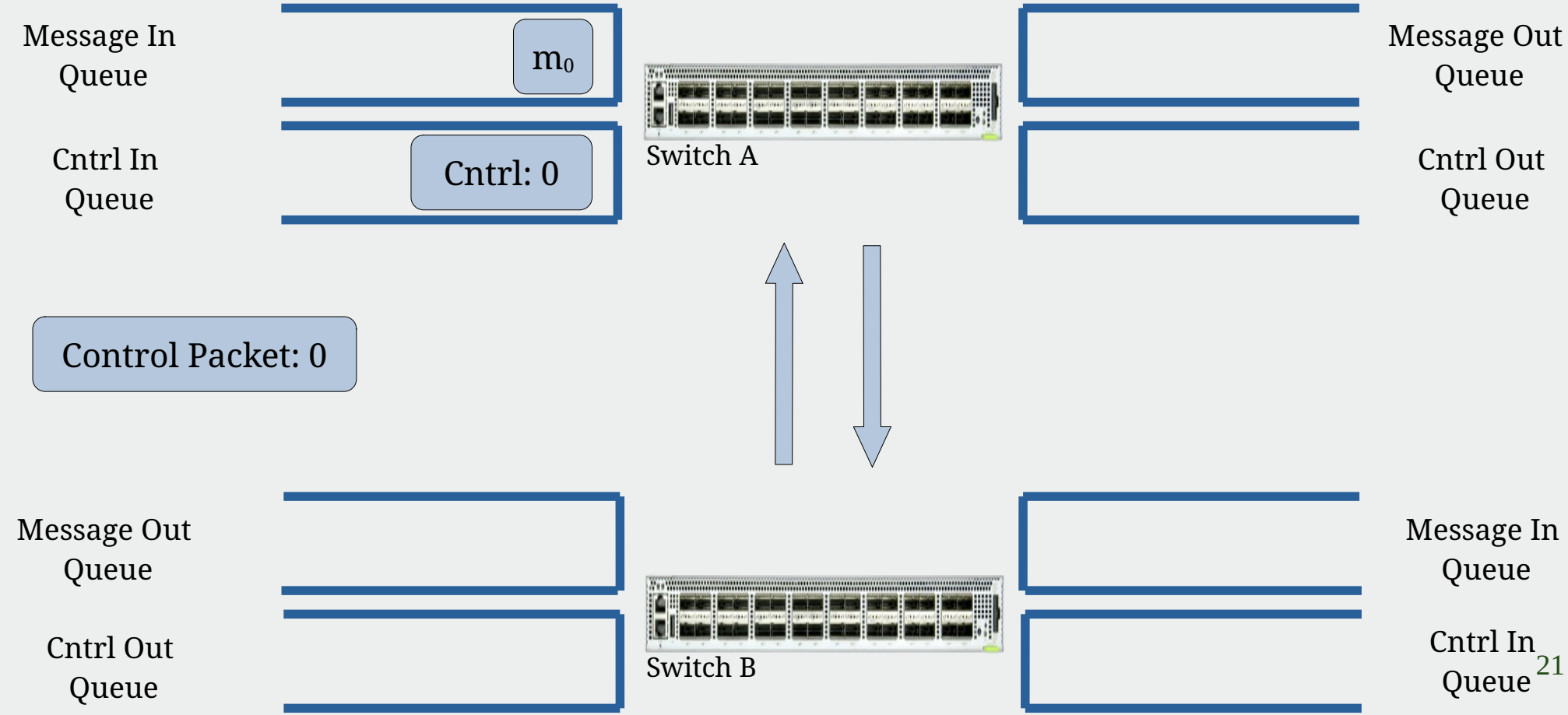
Programmable switches!



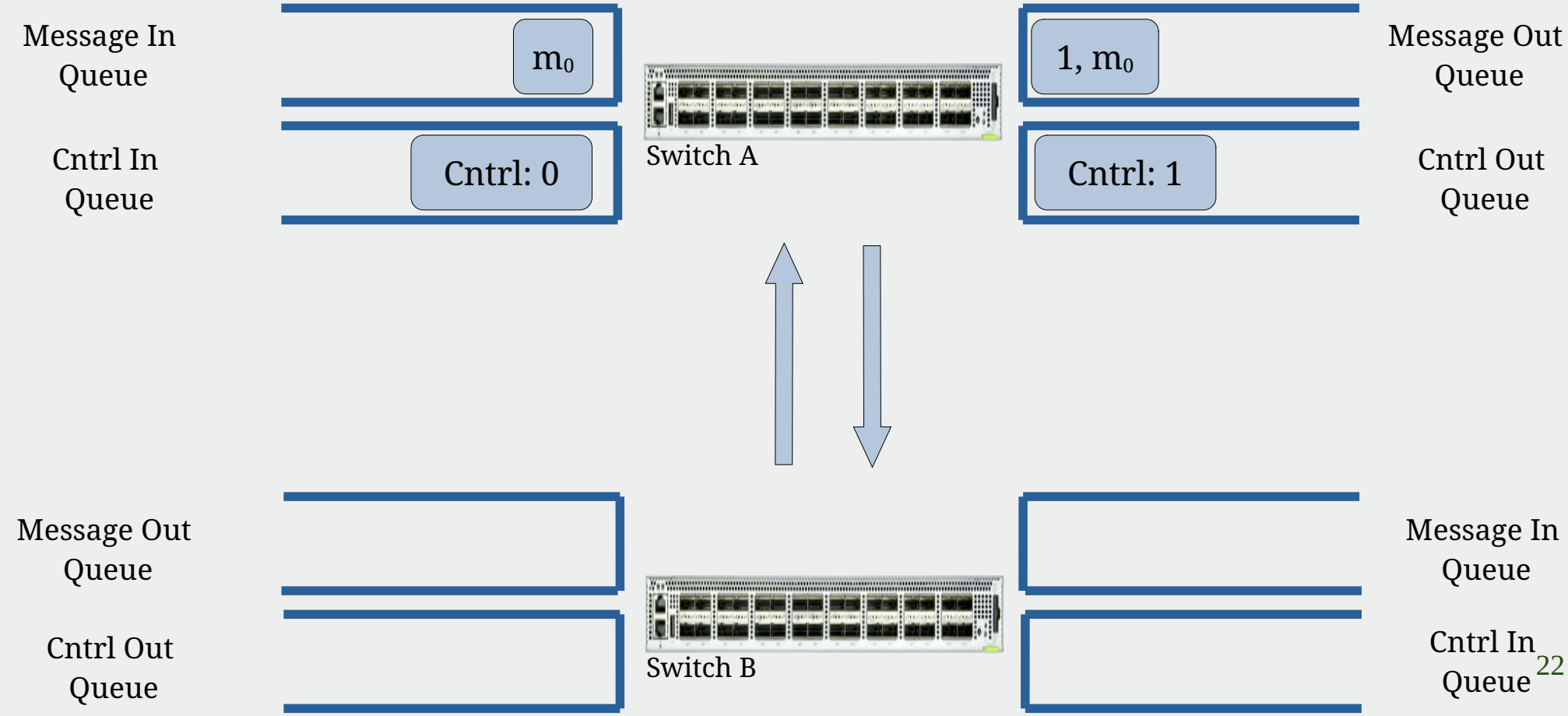
How does this new sequencer work?



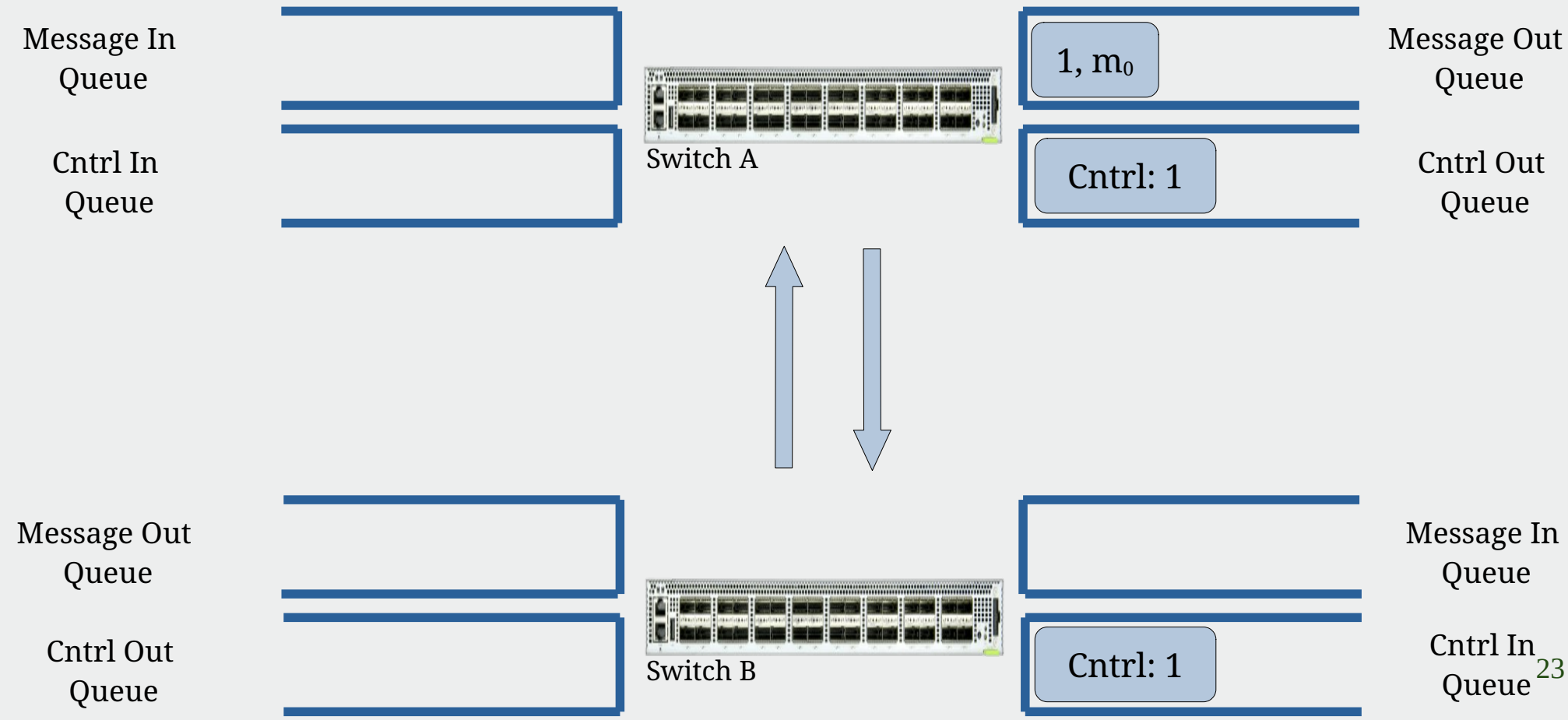
Ring: Example



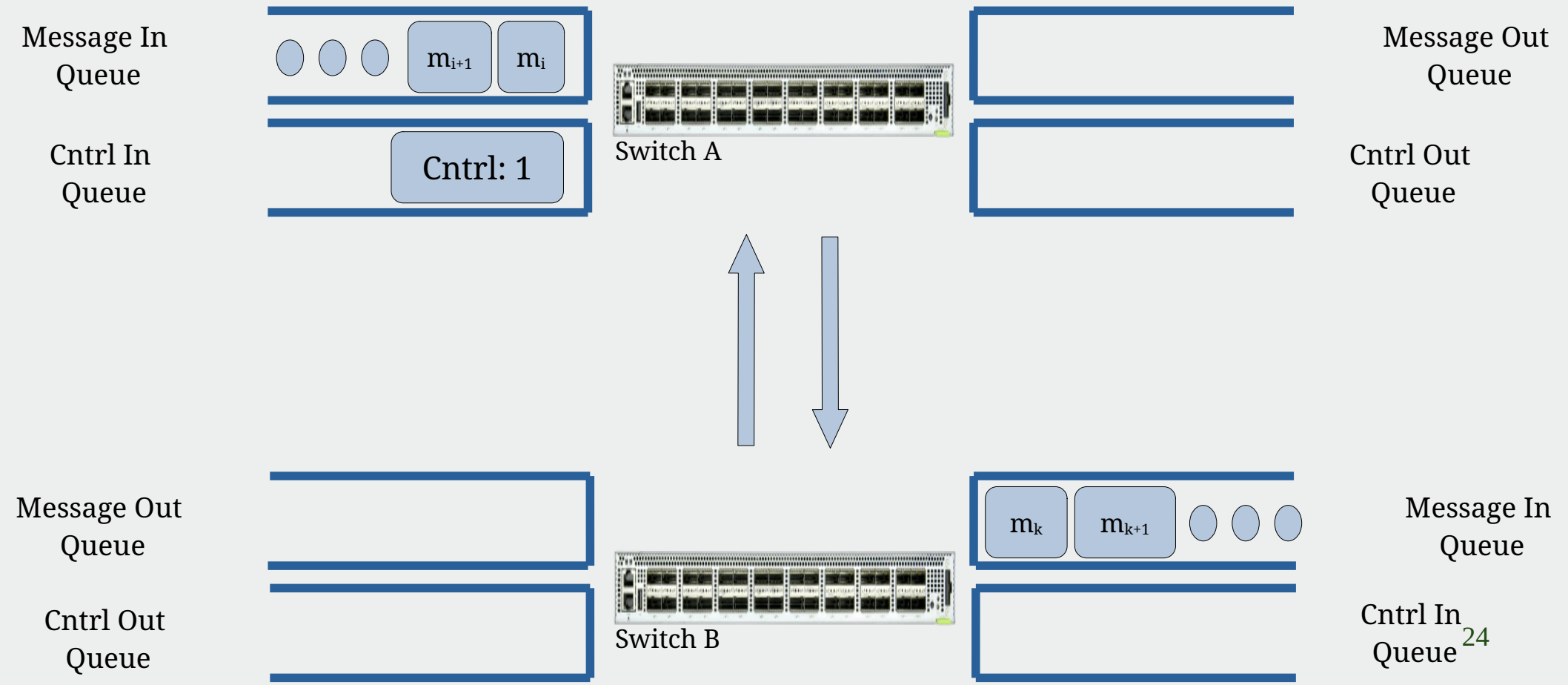
Ring: Example



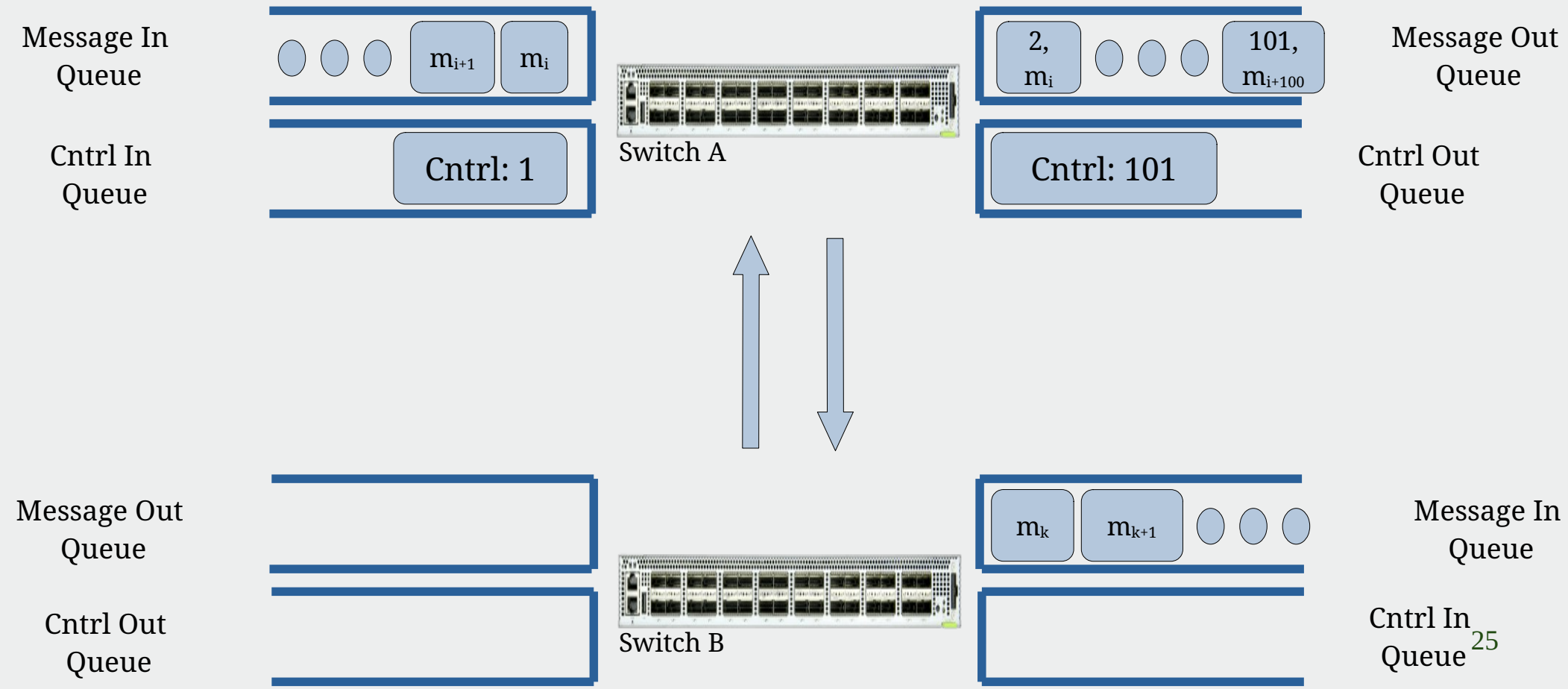
Ring: Example



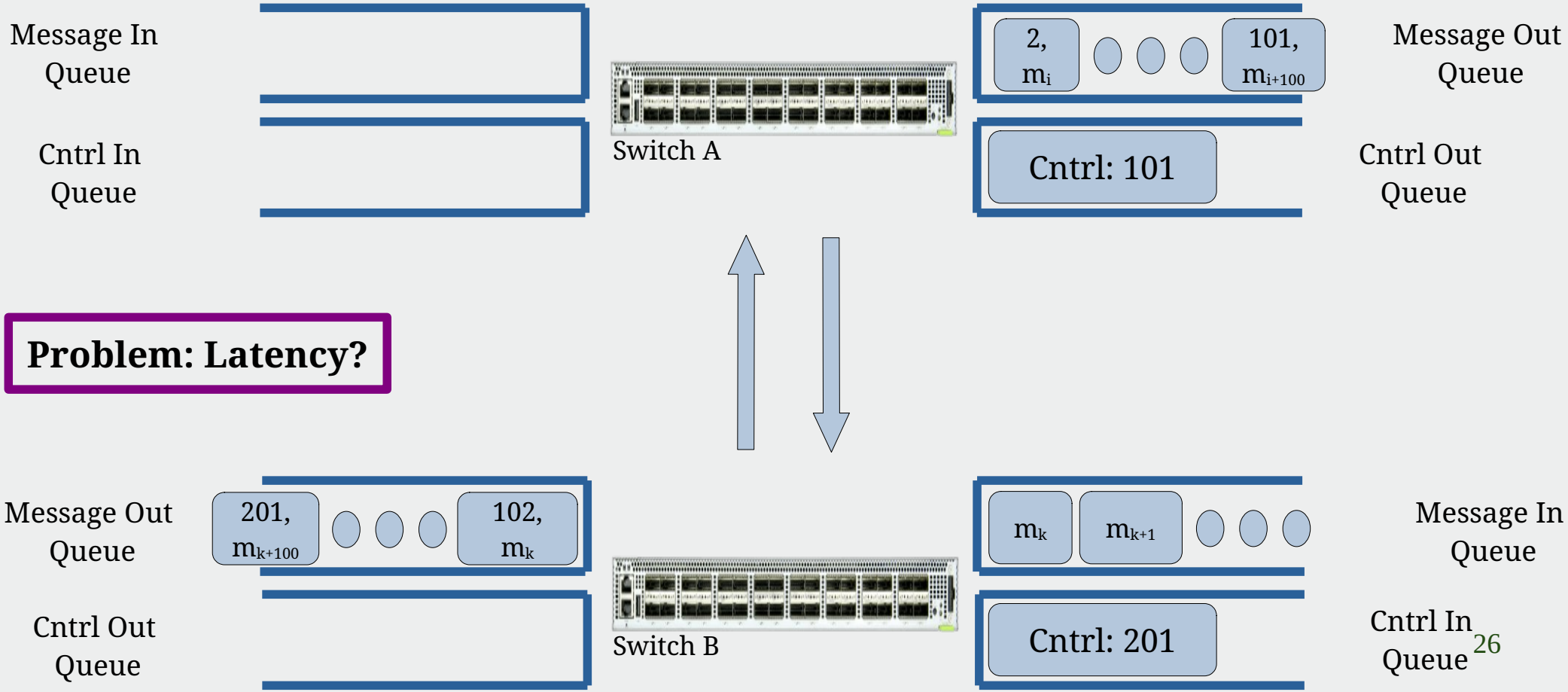
Ring: Scale Via Batching



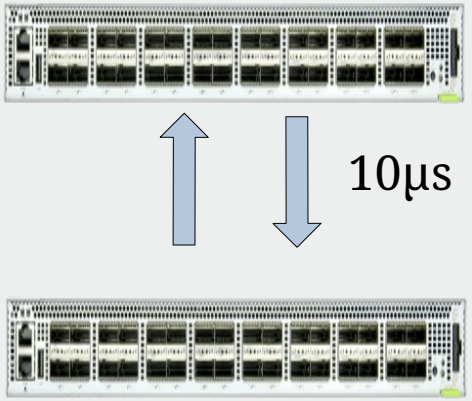
Ring: Scale Via Batching



Ring: Scale Via Batching

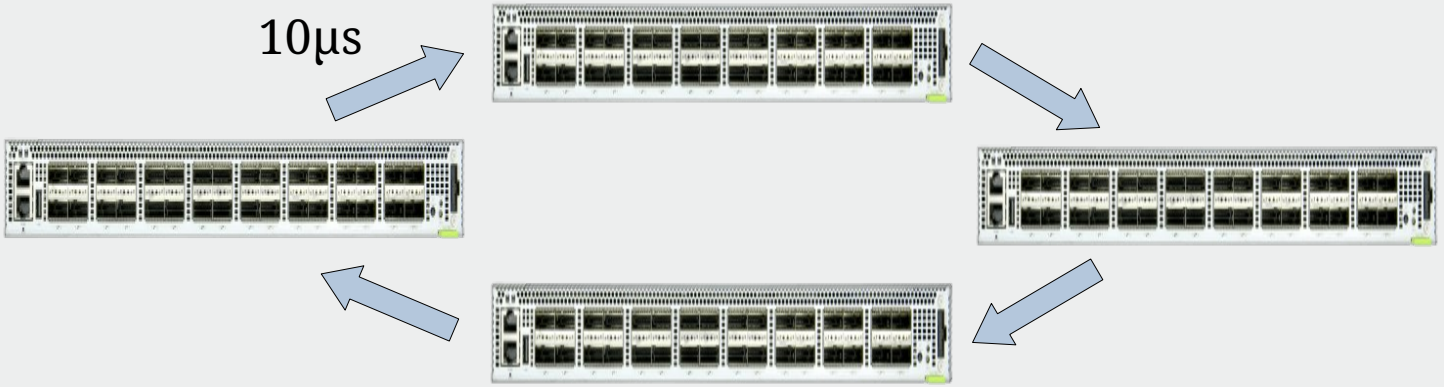


Ring: Latency



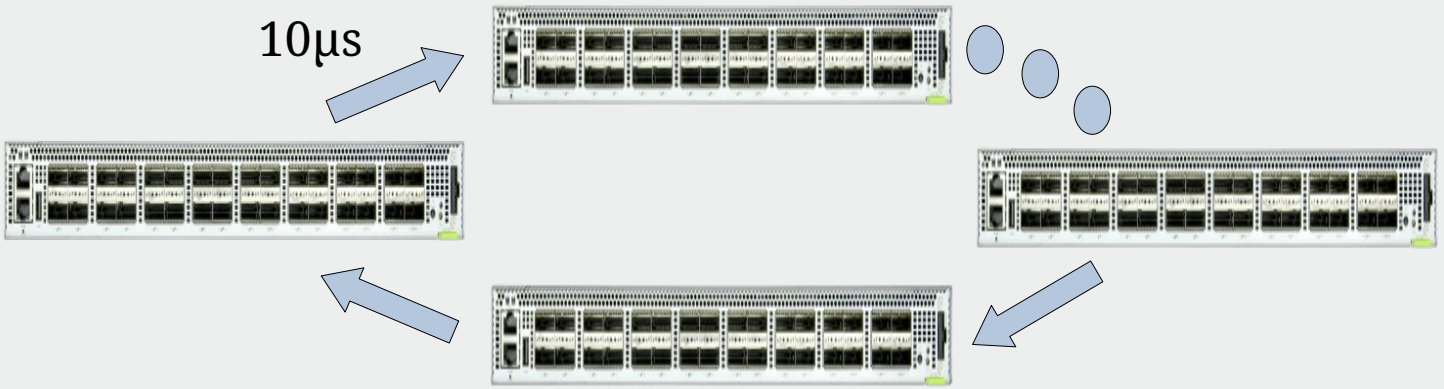
# Switch	Tput (req/s)	Latency
2	1.25 billion	20µs

Ring: Latency



# Switch	Tput (req/s)	Latency
2	1.25 billion	20µs
4	2.5 billion	40µs

Ring: Latency



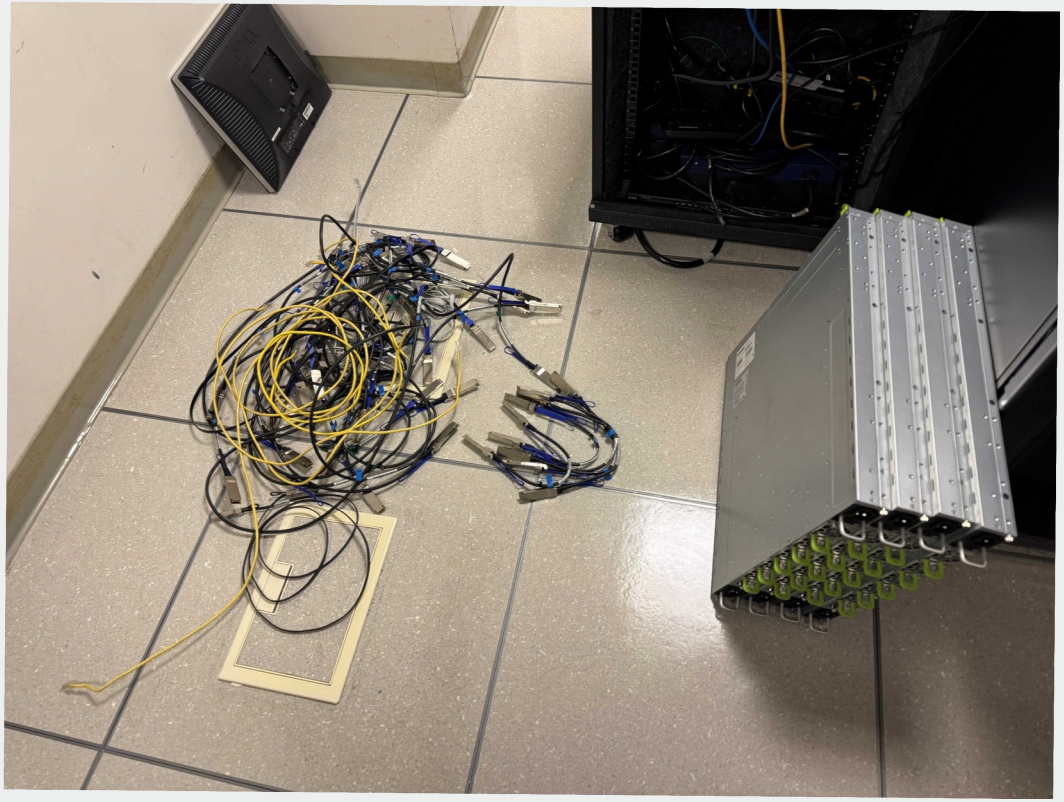
Comparison: RocksDB write takes ~.5ms

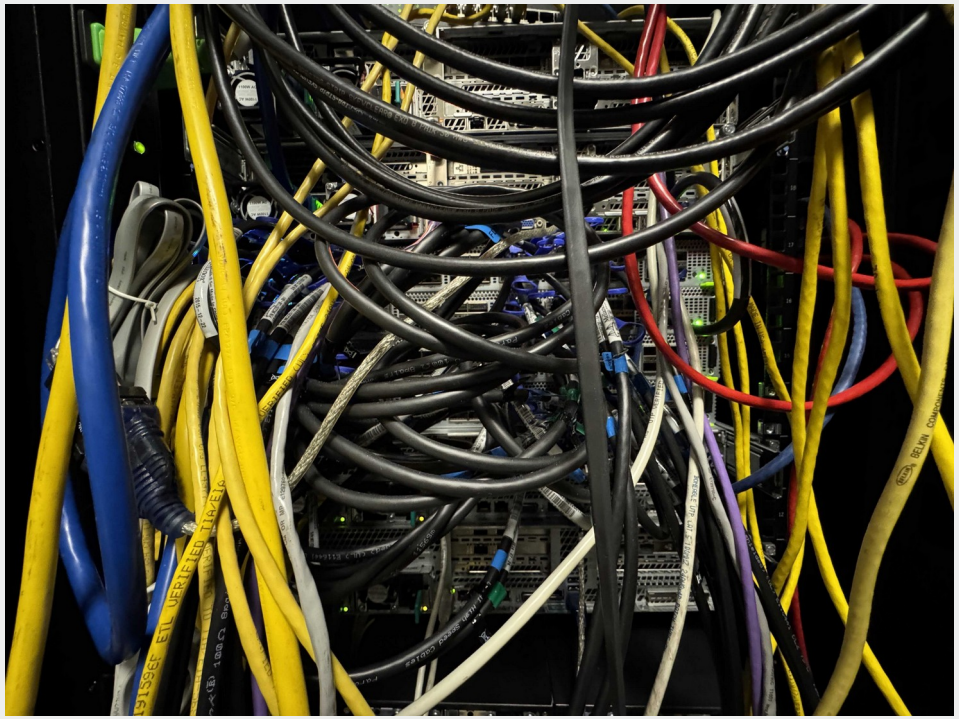
# Switch	Tput (req/s)	Latency
2	1.25 billion	20µs
4	2.5 billion	40µs
100	62.5 billion	1ms

Key Challenges

- Failure handling and ring recovery [done]
- Streams [in progress]
- Augmented ring designs [in progress]
- And more!

Now time to build it! 🚧





Email: micahmurray@berkeley.edu

Goal: One log for the datacenter

Challenge: Need to scale sequencing

Insight #1: Sequencing is simple
Action #1: 'Cheat' with specialized hardware

Insight #2: Need multiple sequencer nodes
Action #2: Distribute sequencing using a ring

Other insight: Accelerated hardware keeps ring latency comparable to replication

Extra Slide!

Why the total order?

- Kafka: Need to shard for scalability, but want total order across partitions
- Cross data store transactions in the age of ever expanding microservice architectures
- LLM Agents: Accessing many applications at once and will need to know the order in which application operations occurred