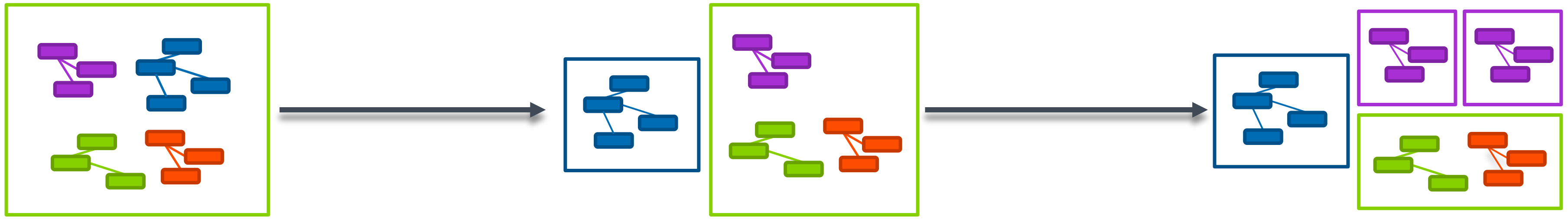
Event Storage in AxonServer

How does it work?

Allard Buijze
CTO & Founder, AxonIQ

Why?!

Location transparency



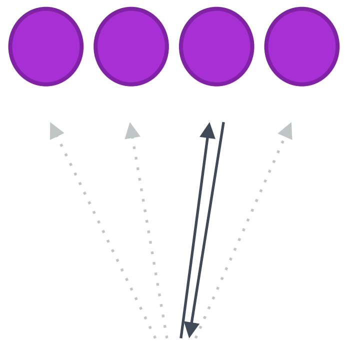
A component should neither be aware of nor make any assumptions about the location of components it interacts with.

Location transparency starts with good API design

(but doesn't end there)

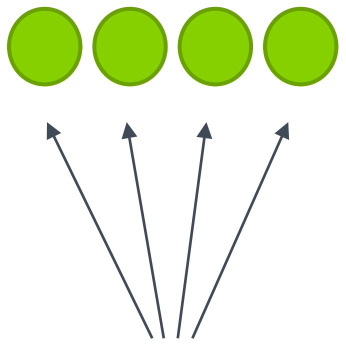
Microservices Messaging

Commands



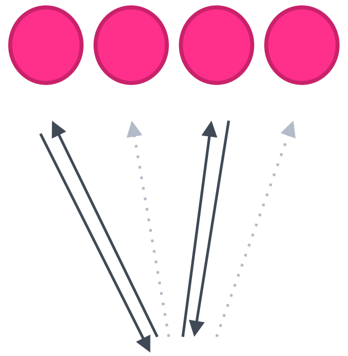
Route to single handler
Use consistent hashing
Provide result

Events



Distribute to all logical handlers
Consumers express ordering req's
No results

Queries



Route with load balancing
Sometimes scatter/gather
Provide result

"Event" and "Message" is not the same thing

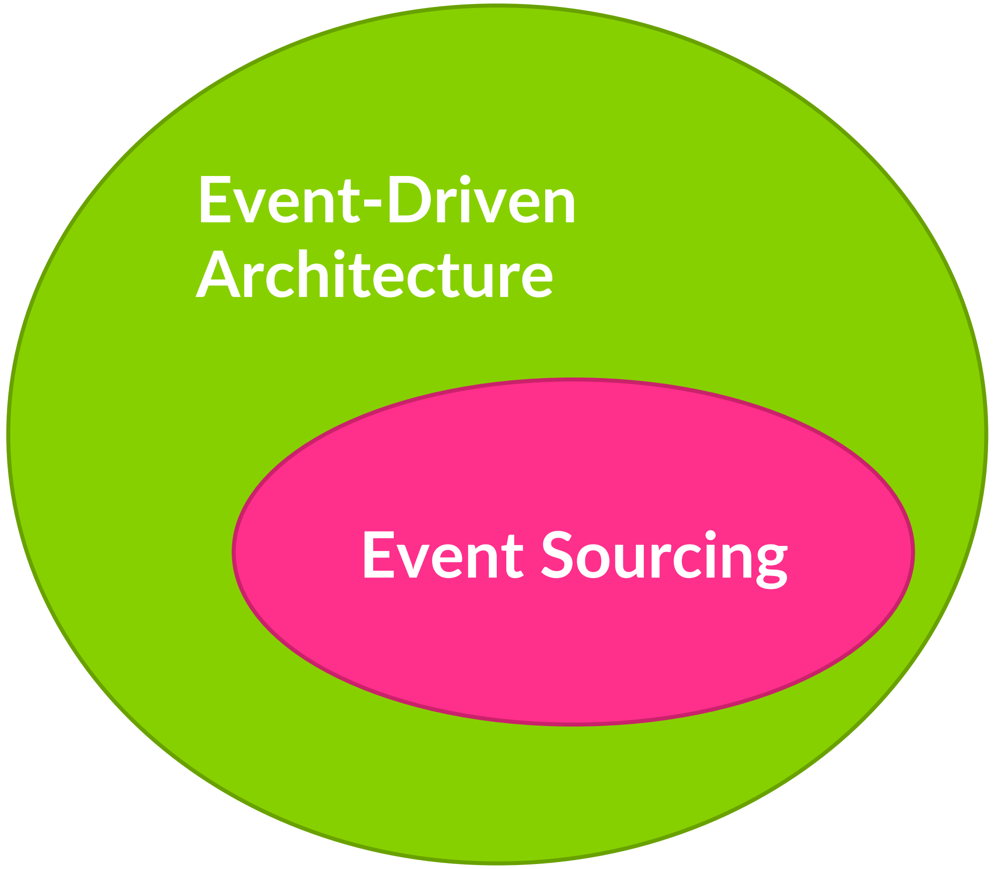
Events…

- Events retain their value of time

- How do we guarantee atomic publication of events **and** state change commits?

- How do we guarantee that our events are a truthful representation of an entity's history

Event Sourcing



is a specific type of Event-Driven Architecture

in which Events are at the heart of the persistence / data storage architecture

Event Sourcing

… is about capturing …

the truth,
the whole truth,
nothing but the truth

Event Sourcing

State storage	Event Sourcing
id: 123	OrderCreated (id: 123)
items	ItemAdded (2x Deluxe Chair, €399)
1x Deluxe Chair - € 399	ItemRemoved (1x Deluxe Chair, €399)
status: return shipment rcvd	OrderConfirmed
	OrderShipped
	OrderCancelledByUser
	ReturnShipmentReceived

Why use event sourcing?

Business reasons

- Auditing / compliance / transparency
- Data mining, analytics: value from data

Technical reasons

- *Guaranteed completeness of raised events*
- *Single source of truth*
- *Concurrency / conflict resolution*
- *Facilitates debugging*
- *Replay into new read models (CQRS)*
- *Easily capture intent*
- *Deal with complexity in models*

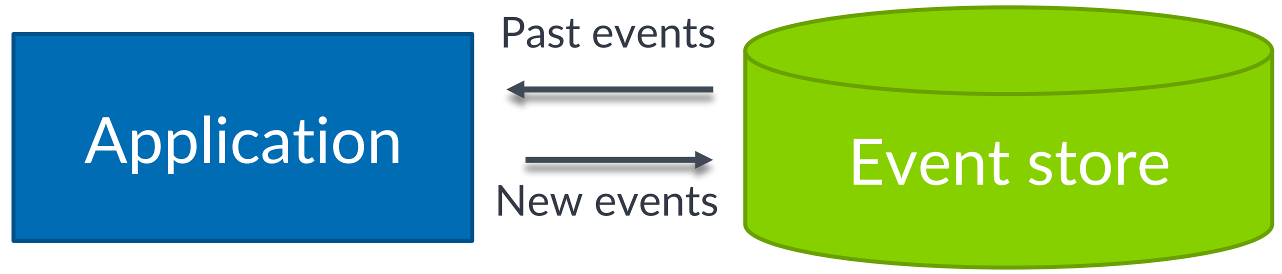
What's an "event store"?

In the architecture of an event-sourced application, the event store is the database system used to store the events.

In terms of implementation, this could be

- General purpose RDBMS technology (Oracle, MySQL, Postgres, etc.)
- General purpose NoSQL technology (Mongo, Cassandra, etc.)
- Specialized event store technology (**AxonServer**, Greg Young's EventStore, PumpkinDB)

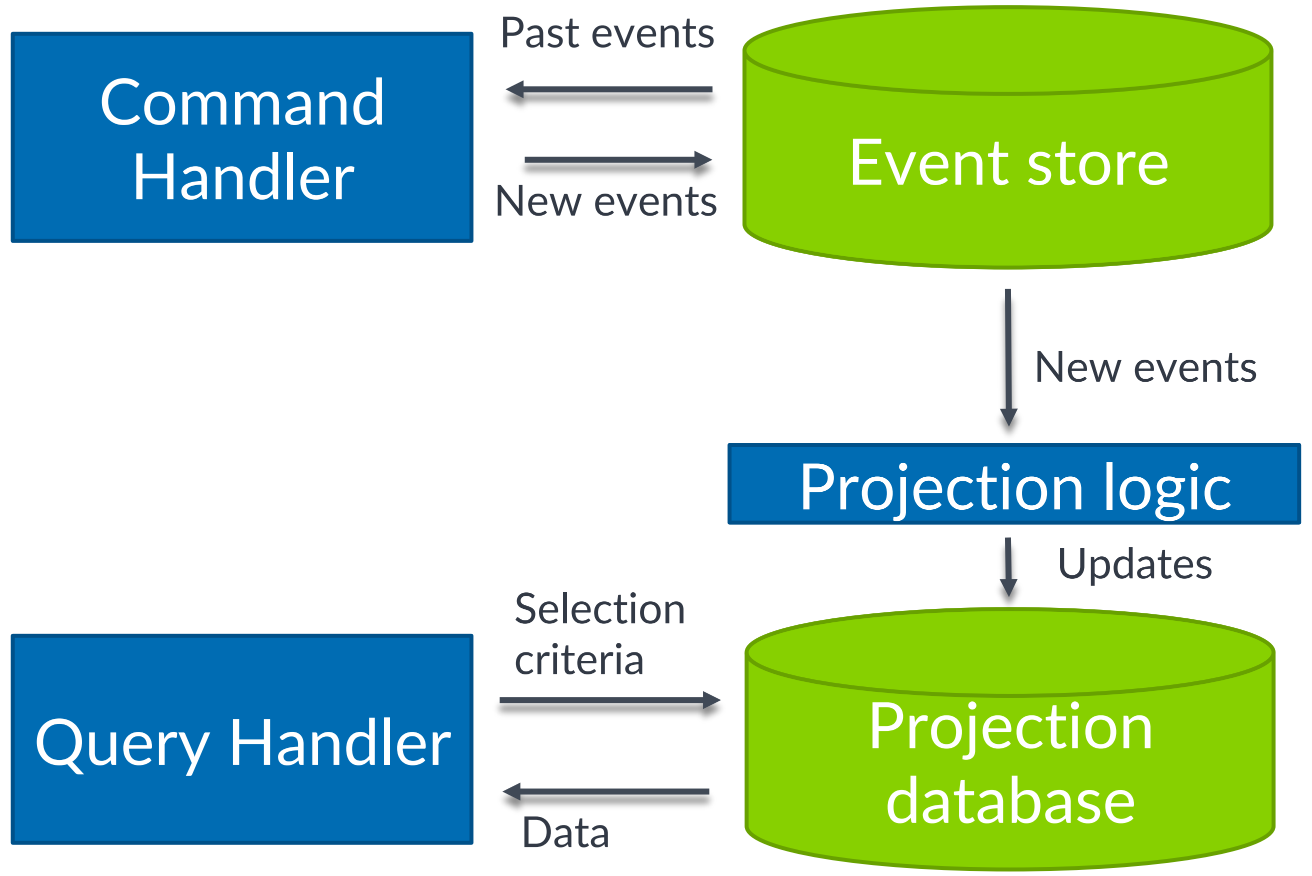
Event store in context



- Works well for processing changes (Commands)
- Does *not* work well for, say, finding all orders with total value > EUR 100

CQRS

Command-Query Responsibility Segregation



Event store requirements

Event Storage Requirements

Read Events

All for an aggregate
(event sourced repository)

All since point in time
(for read model projection)

Read back in write order

Ad-hoc queries
(for debug, monitoring, support)

Write Events

Append Events

Append events

~~Insert events at random point~~

~~Update events~~

~~Delete events~~

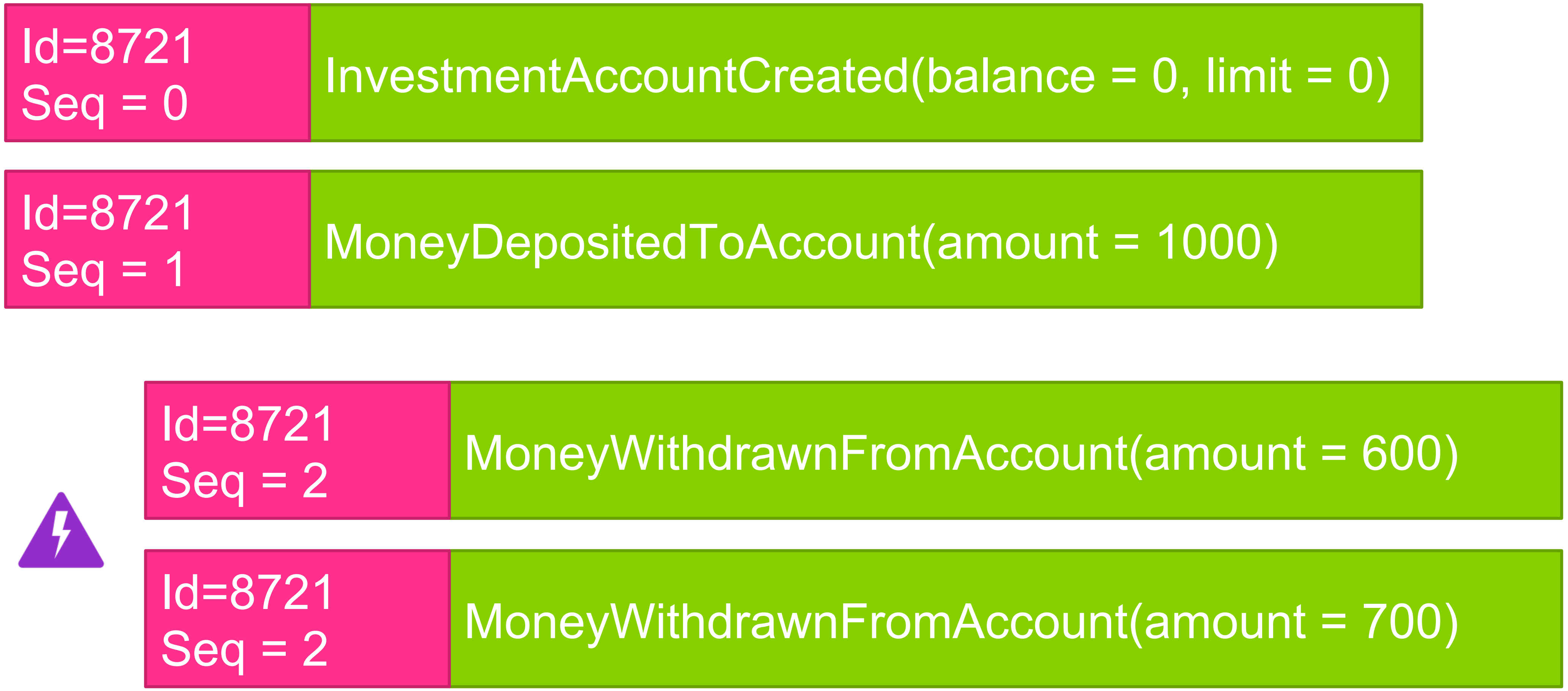
Event Storage Requirements

Id=8721
Seq = 0
	InvestmentAccountCreated(balance = 0, limit = 0)

Id=8721
Seq = 1
	MoneyDepositedToAccount(amount = 1000)

Id=8721
Seq = 2
	MoneyWithdrawnFromAccount(amount = 600)

Event Storage Requirements



Id=8721
Seq = 0
InvestmentAccountCreated(balance = 0, limit = 0)

Id=8721
Seq = 1
MoneyDepositedToAccount(amount = 1000)

Id=8721
Seq = 2
MoneyWithdrawnFromAccount(amount = 600)

Id=8721
Seq = 2
MoneyWithdrawnFromAccount(amount = 700)

Event Storage Requirements

Read Events

All for an aggregate

All since point in time

Read back in write order

Ad-hoc queries

Append Events

Validate aggregate sequence numbers

(consistency)

Event Storage Requirements

Id=8721
Seq = 0
	InvestmentAccountCreated(balance = 0, limit = 0)

Id=8721
Seq = 1
	MoneyDepositedToAccount(amount = 1000)

Command: buy 5 shares of XYZ Corp @ 100

Id=8721
Seq = 2
	MoneyWithdrawnFromAccount(amount = 500)

Id=8721
Seq = 3
	SharesAddedToAccount(symbol = ‘XYZ’, n = 5)

Event Storage Requirements

Read Events

All for an aggregate

All since point in time

Read back in write order

Ad-hoc queries

Only read committed events
(isolation)

Append Events

Validate aggregate sequence numbers
(consistency)

Append multiple events at once
(atomicity)

Committed events protected against loss
(durability)

Event Storage Requirements

Id=8721
Seq = 0
	BankAccountCreated(balance = 0, limit = 0)

Id=8721
Seq = 1
	MoneyDepositedToAccount(amount = 1000)

⋮	**Using the bank account for 10 years** ⋮

Id=8721
Seq = 9102
	MoneyWithdrawnFromAccount(amount = 700)

Id=8721
Seq = 9103
	MoneyDepositedToAccount(amount = 500)

Event Storage Requirements

Id=8721
Seq = 0
	BankAccountCreated(balance = 0, limit = 0)

Id=8721
Seq = 1
	MoneyDepositedToAccount(amount = 1000)

⋮	⋮

Id=8721
Seq = 9080
	BankAccountSnapshot(balance = 5000)

⋮	⋮

Id=8721
Seq = 9102
	MoneyWithdrawnFromAccount(amount = 700)

Id=8721
Seq = 9103
	MoneyDepositedToAccount(amount = 500)

Event Storage Requirements

Read Events/Snapshots

Read Events

All for an aggregate

- Latests snapshot + later events
- All events

All since point in time

Read back in write order

Ad-hoc queries

Only read committed events

(isolation)

Append Events/Snapshots

Append Events

Validate aggregate sequence numbers

(consistency)

Append multiple events at once

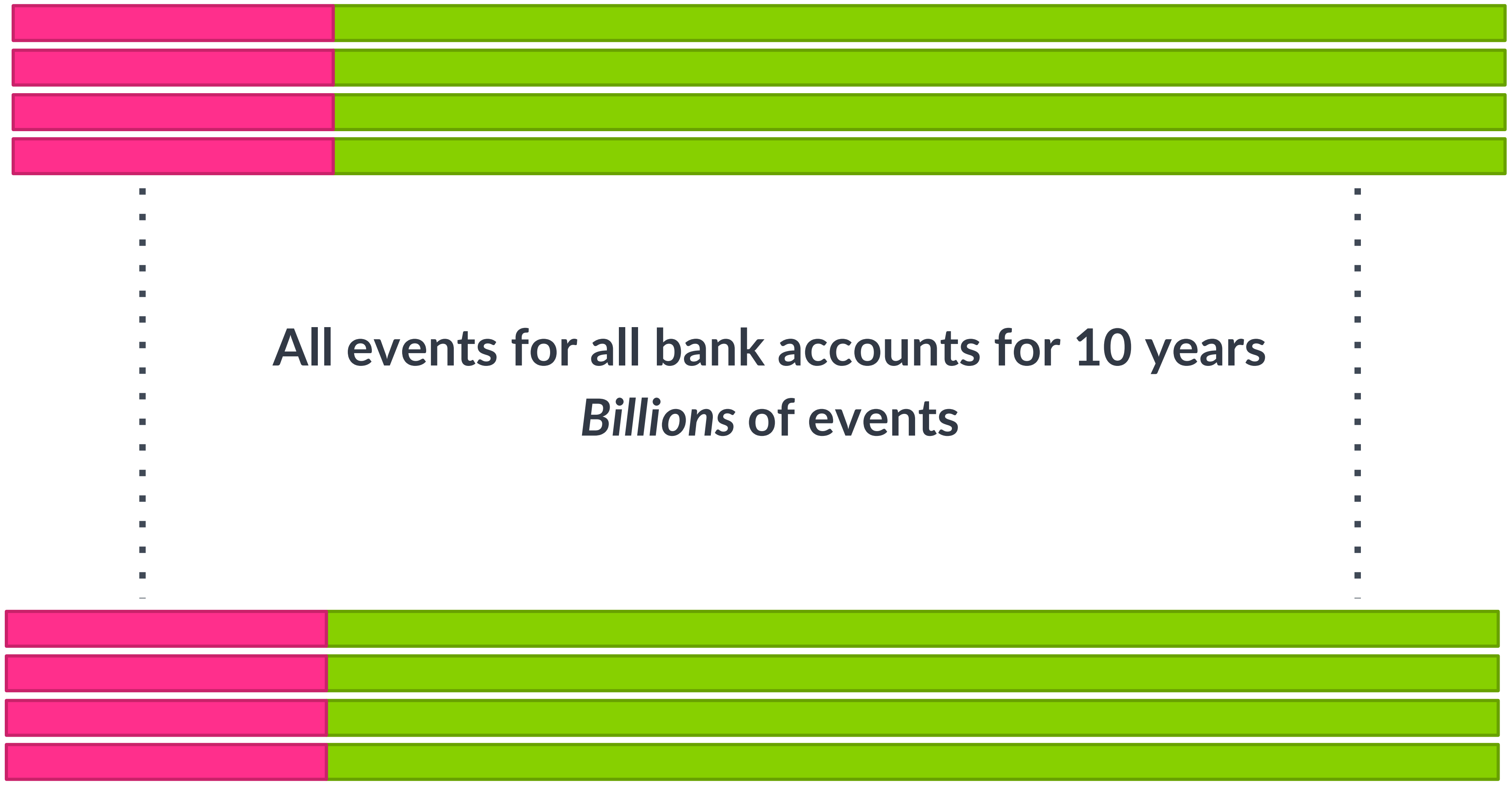
(atomicity)

Committed events protected against loss

(durability)

Append snapshots

Event Storage Requirements



All events for all bank accounts for 10 years

Billions of events

Event Storage Requirements

Read Events/Snapshots

All for an aggregate

- Latests snapshot + later events
- All events

All since point in time

Read back in write order

Ad-hoc queries

Only read committed events

Optimized for recent events

Append Events/Snapshots

Validate aggregate sequence numbers

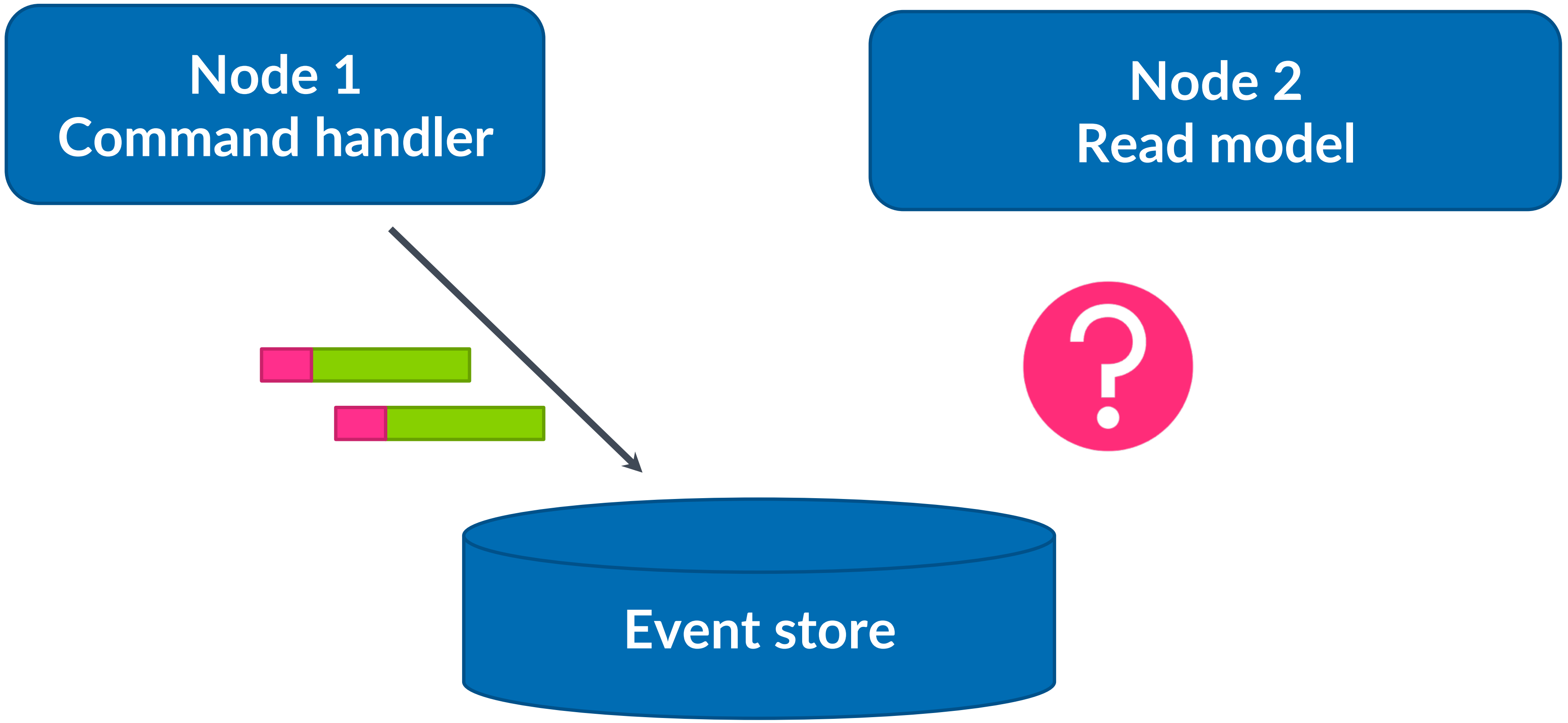
Append multiple events at once

Committed events protected against loss

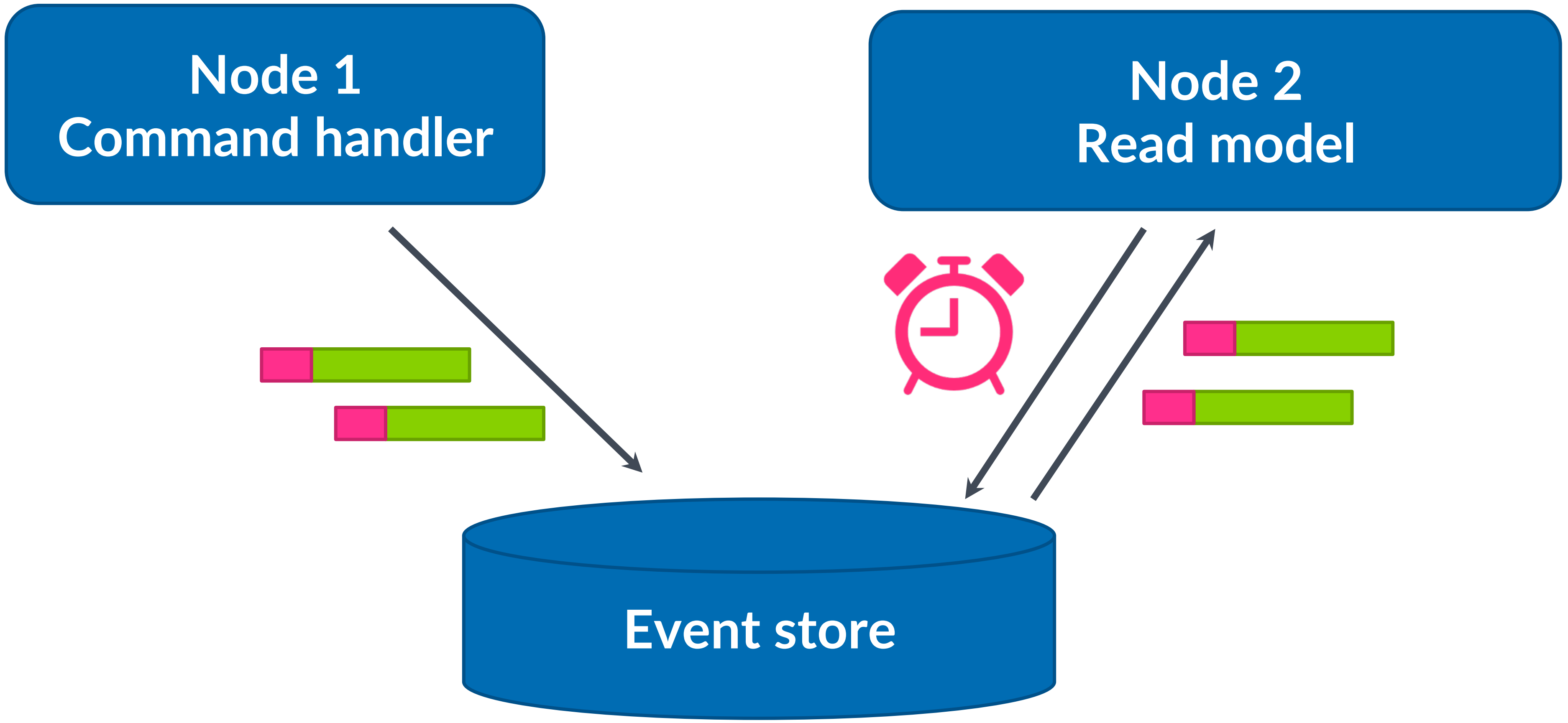
Append snapshots

Constant performance as a function of storage size

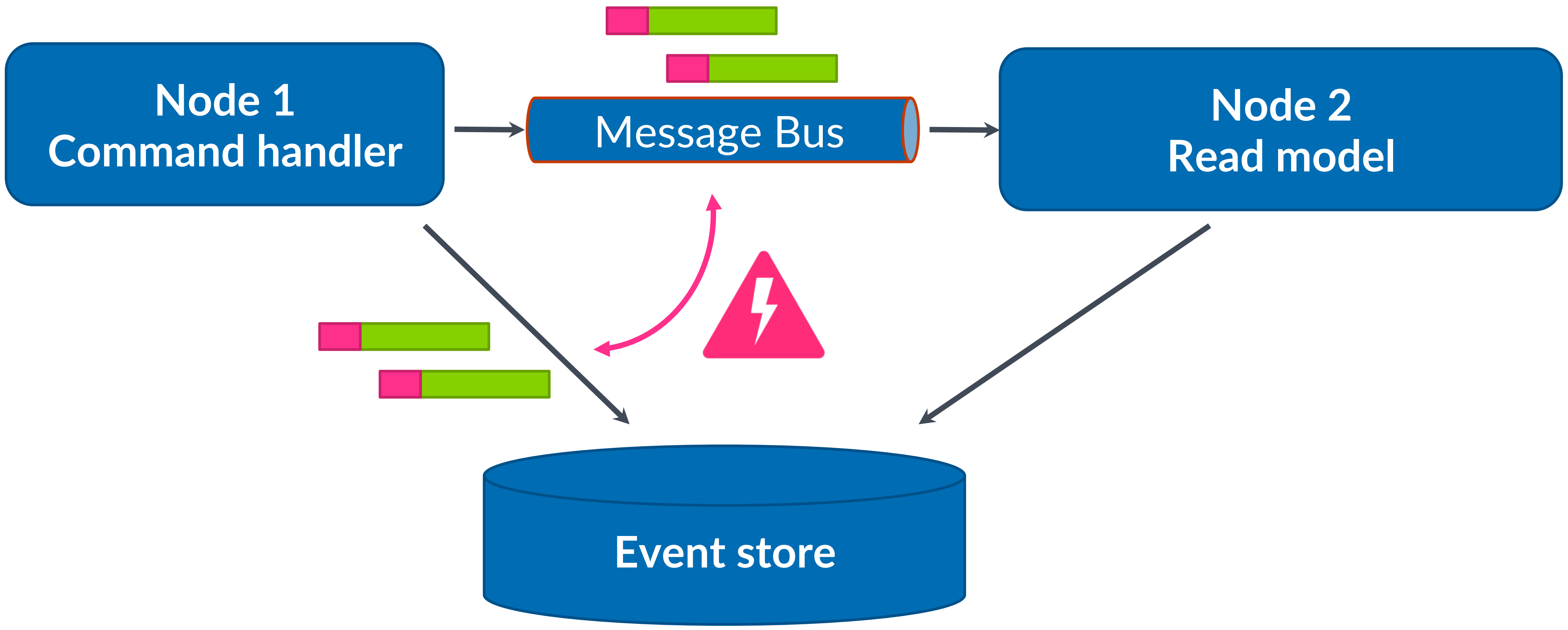
Event Storage Requirements



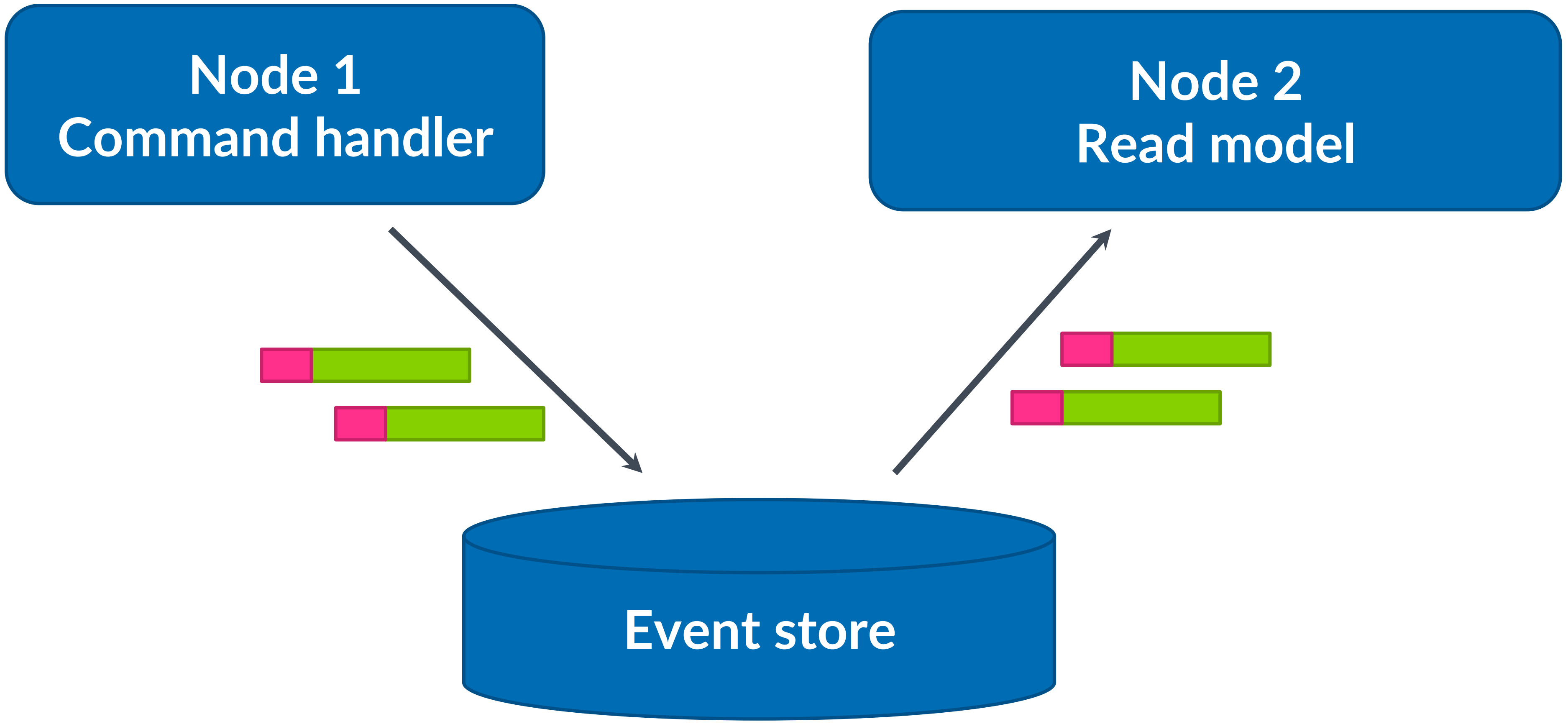
Event Storage Requirements



Event Storage Requirements



Event Storage Requirements



Event Storage Requirements

Read Events/Snapshots

All for an aggregate

- Latests snapshot + later events
- All events

All since point in time, pushing new ones

Read back in write order

Ad-hoc queries

Only read committed events

Optimized for recent events

Append Events/Snapshots

Validate aggregate sequence numbers

Append multiple events at once

Committed events protected against loss

Append snapshots

Constant performance as a function of storage size

Event Storage Requirements

Read Events/Snapshots

- ☐ All for an aggregate
- ☐ • Latests snapshot + later events
- ☐ • All events
- ☐ All since point in time
- ☐ • pushing new ones
- ☐ Read back in write order
- ☐ Ad-hoc queries
- ☐ Only read committed events
- ☐ Optimized for recent events

Append Events/Snapshots

- ☐ Validate aggregate sequence numbers
- ☐ Append multiple events at once
- ☐ Committed events protected against loss
- ☐ Append snapshots
- ☐ Constant performance as a function of storage size

Event store options

Incumbents

- RDBMS (any vendor)
- MongoDB

Contenders

Generic

- Kafka
- Cassandra

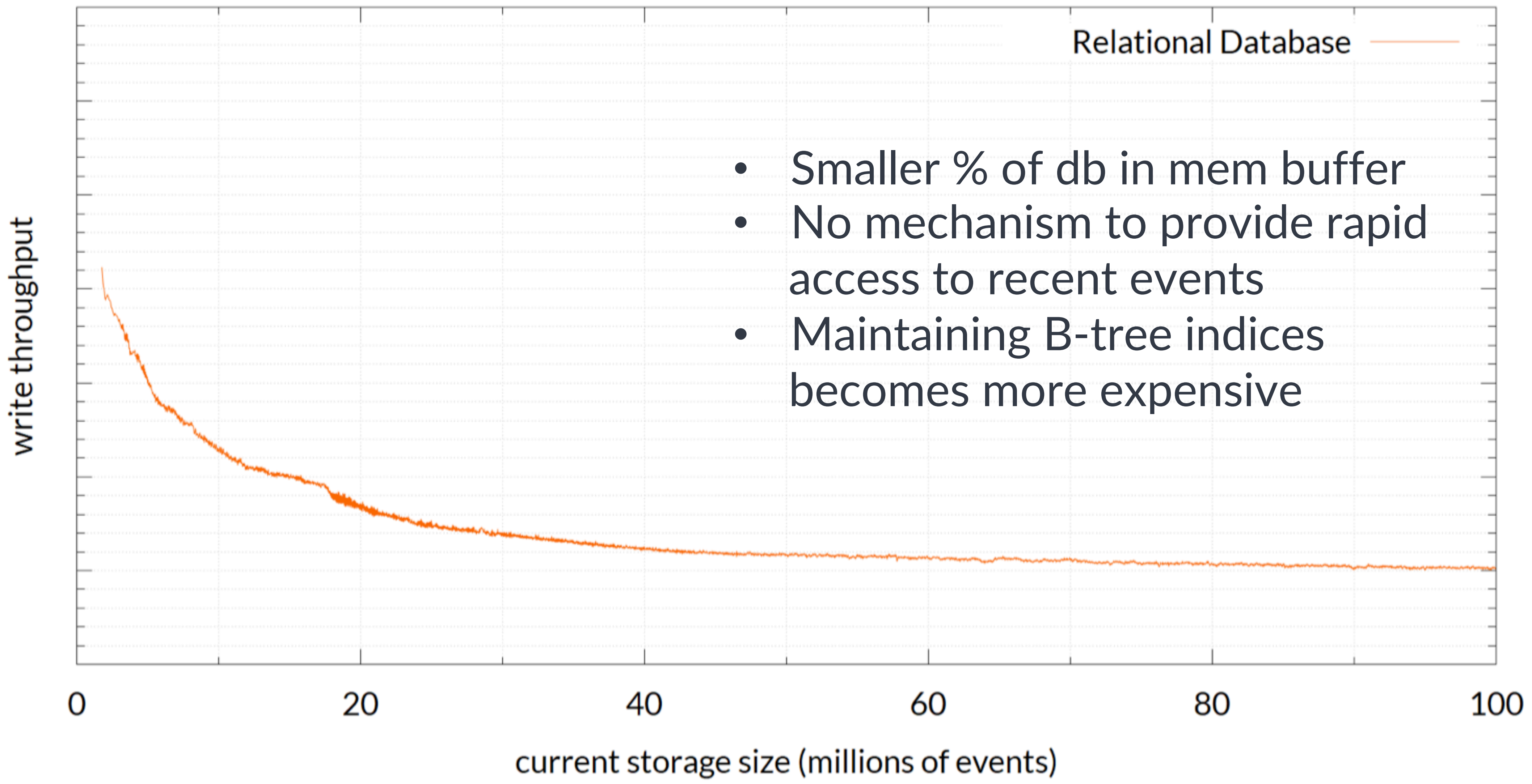
Built-for-purpose

- Greg Young's EventStore
- PumpkinDB

RDBMS

Pros

- Well established tech
- Transactionality



- Smaller % of db in mem buffer
- No mechanism to provide rapid access to recent events
- Maintaining B-tree indices becomes more expensive

AxonIQ

@allardbz

Read Events/Snapshots

All for an aggregate

- ☐ • Latests snapshot + later events
- ☐ • All events
- ☐ All since point in time
- 🟥 • Pushing new events
- ☐ Read back in write order
- ☐ Only read committed events
- 🟥 Optimized for recent events

Append Events/Snapshots

- ☐ Validate aggregate sequence numbers
- ☐ Append multiple events at once
- ☐ Committed events protected against loss
- ☐ Append snapshots
- 🟥 Constant performance as a function of storage size

RDBMS

Pros

- Well established tech
- Transactionality

Cons

- Scalability problems
- No (clean) event push

Read Events/Snapshots

All for an aggregate
- ☐ • Latests snapshot + later events
- ☐ • All events
- ☐ All since point in time
- 🟥 • Pushing new events
- ☐ Read back in write order
- ☐ Only read committed events
- 🟥 Optimized for recent events

Append Events/Snapshot

- ☐ Validate aggregate sequence numbers
- ☐ Append multiple events at once
- ☐ Committed events protected against loss
- ☐ Append snapshots
- 🟥 Constant performance as a function of storage size

MongoDB

Pros

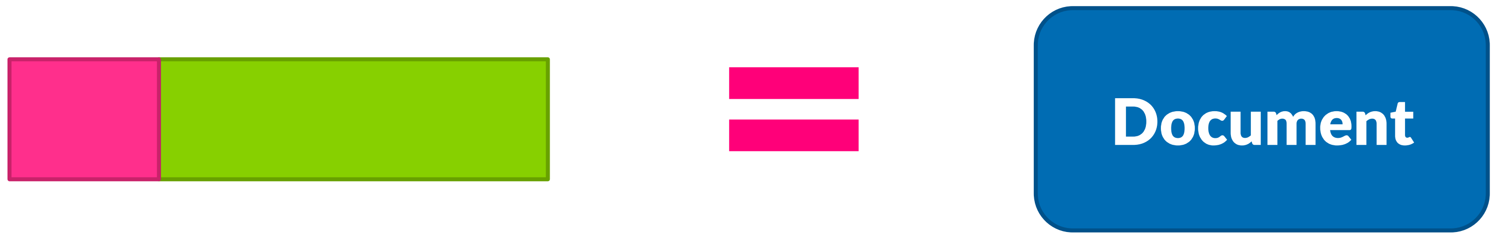
- Horizontal scalability through sharding
- Analysis on events

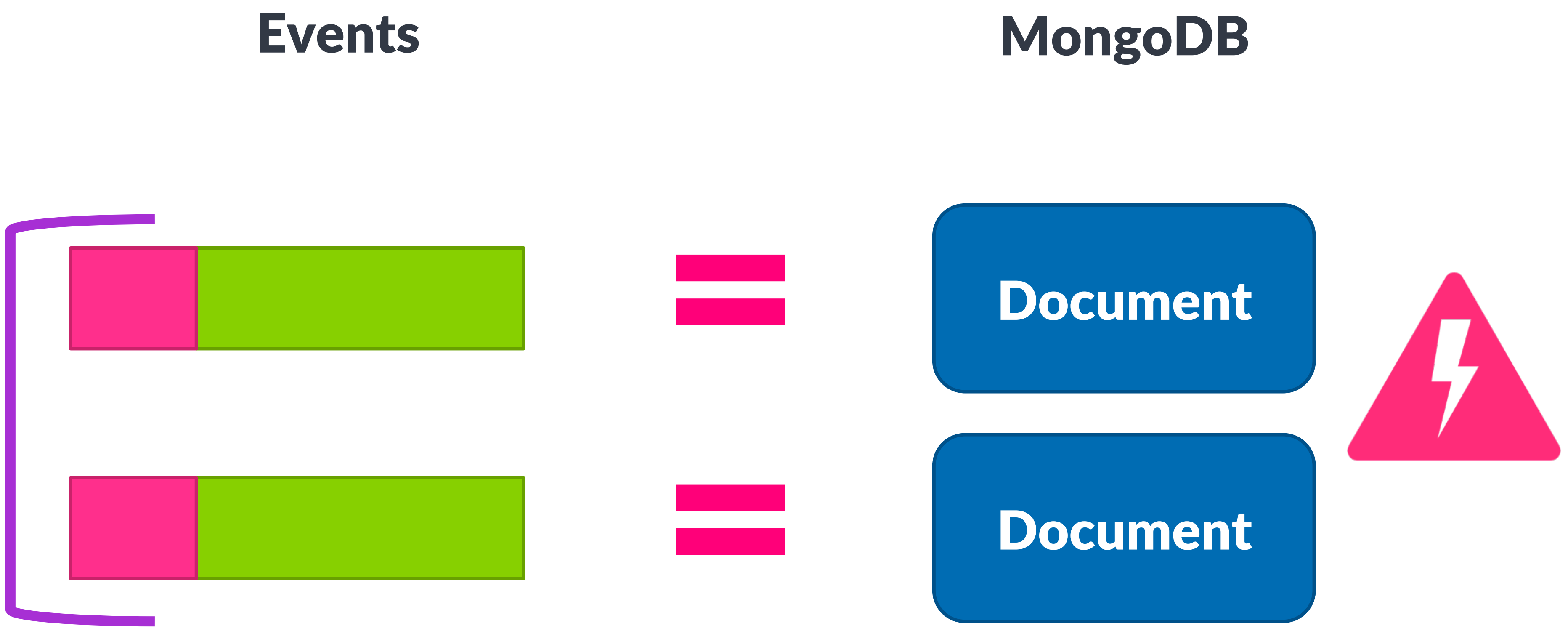
Events

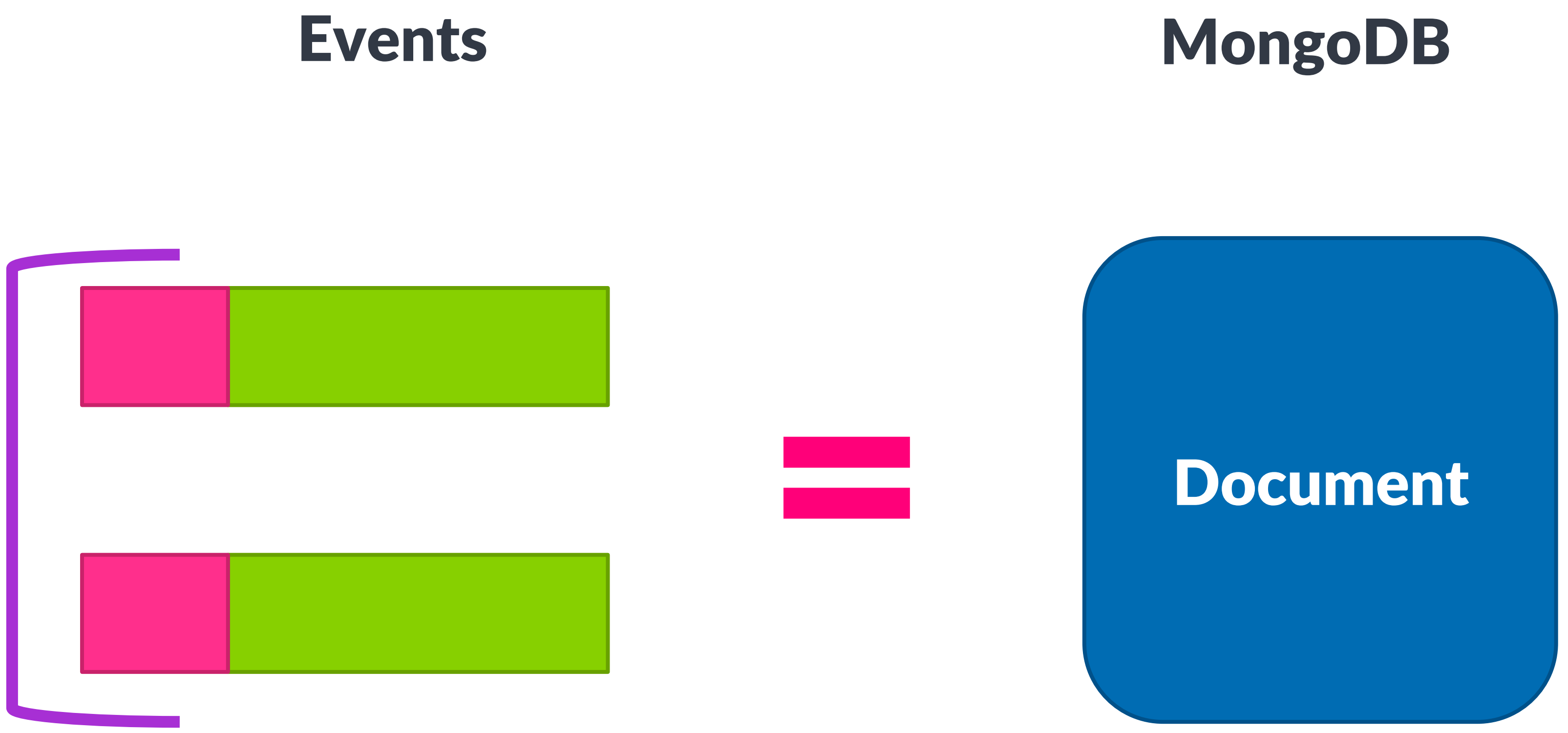
MongoDB

Events

MongoDB




Events
MongoDB
Document
Document


Events
MongoDB
Document

Read Events/Snapshots

All for an aggregate
- ☐ Latests snapshot + later events
- ☐ All events

☐ All since point in time
- 🟨 Pushing new events

🟥 Read back in write order

🟥 Only read committed events

☐ Optimized for recent events

Append Events/Snapshots

☐ Validate aggregate sequence numbers

🟨 Append multiple events at once

☐ Committed events protected against loss

☐ Append snapshots

☐ Constant performance as a function of storage size

MongoDB

Pros

- Horizontal scalability through sharding
- Analysis on events

Cons

- ~~Document transactions~~
- No (easy) event push
- No global sequence #

Read Events/Snapshots

All for an aggregate
- ☐ • Latests snapshot + later events
- ☐ • All events
- ☐ All since point in time
- 🟨 • Pushing new events
- 🟥 Read back in write order
- 🟥 Only read committed events
- ☐ Optimized for recent events

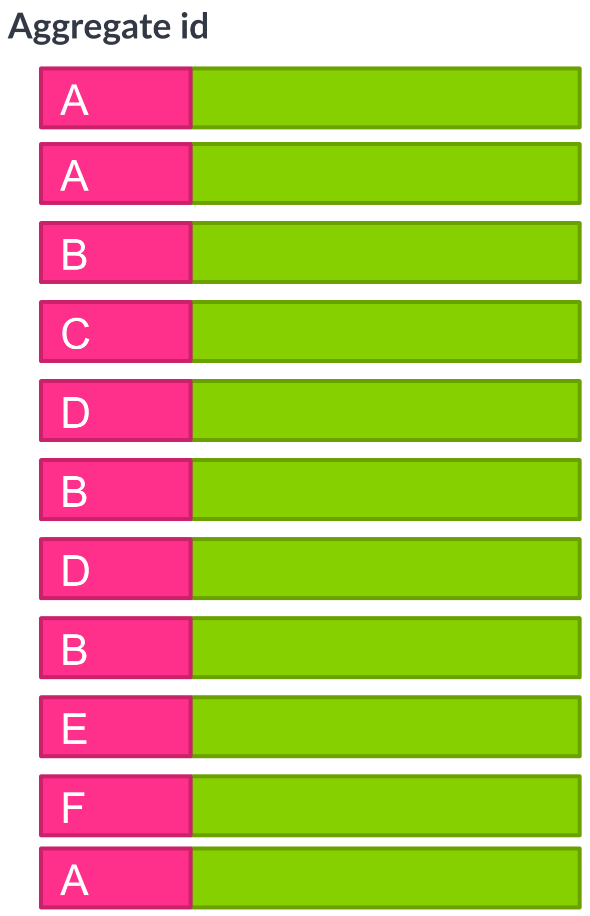
Append Events/Snapshots

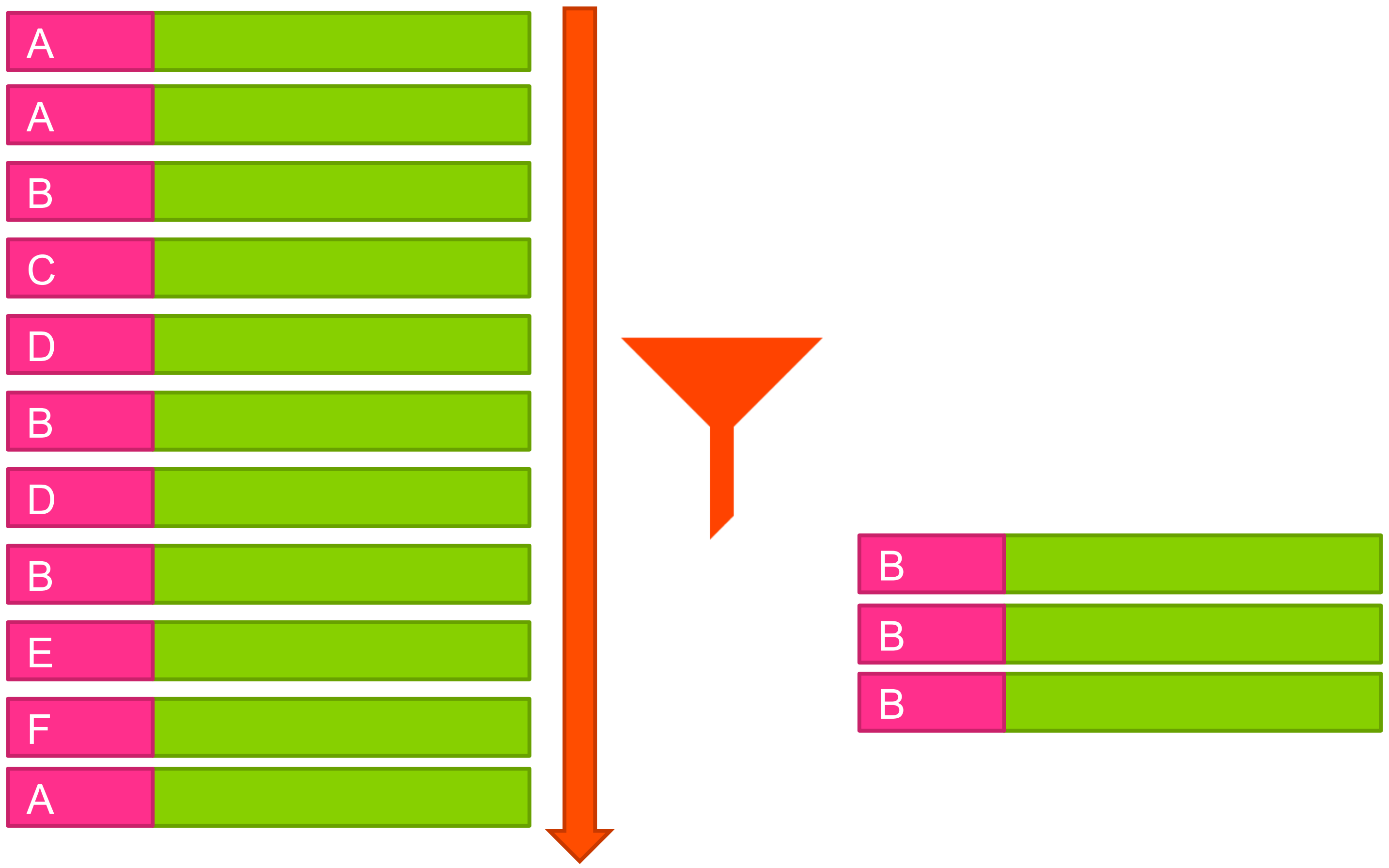
- ☐ Validate aggregate sequence numbers
- 🟨 Append multiple events at once
- ☐ Committed events protected against loss
- ☐ Append snapshots
- ☐ Constant performance as a function of storage size

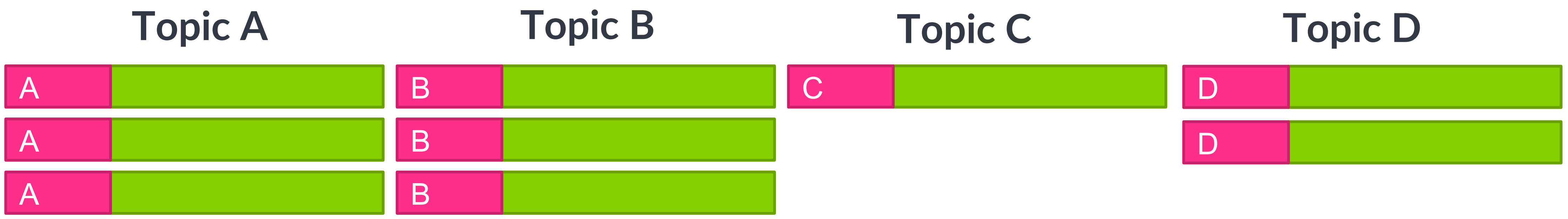
Kafka

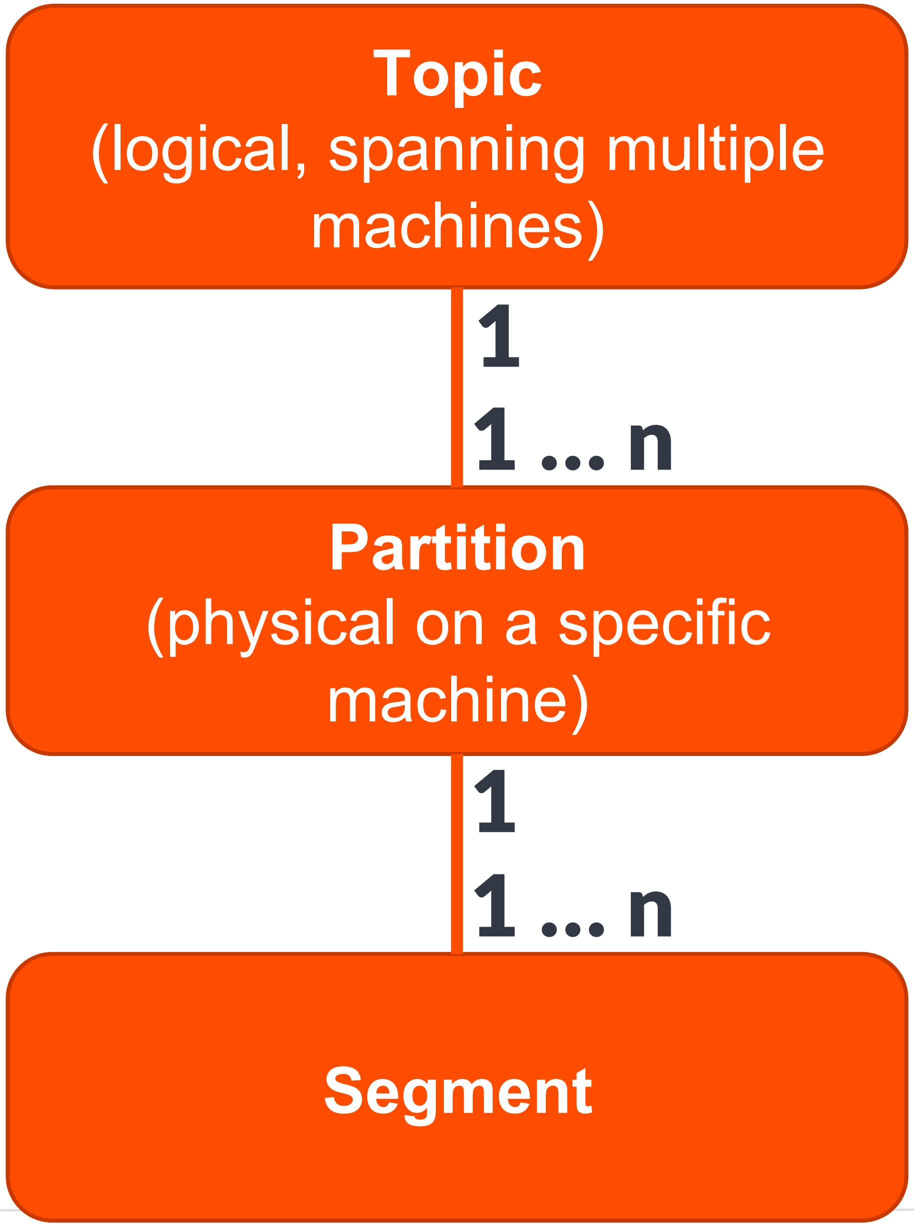
Pros

- Messaging focussed
- Extremely scalable


Aggregate id
A
A
B
C
D
B
D
B
E
F
A


A
A
B
C
D
B
D
B
E
F
A
B
B
B


Topic A
Topic B
Topic C
Topic D
A
A
A
B
B
B
C
D
D


Topic
(logical, spanning multiple machines)
1
1 … n
Partition
(physical on a specific machine)
1
1 … n
Segment

Has an associated directory

Has a log and index file

This doesn't scale to millions of aggregates!

Read Events/Snapshots

All for an aggregate
- 🟥 • Latests snapshot + later events
- 🟥 • All events
- ☐ All since point in time
- ☐ • Pushing new events
- ☐ Read back in write order
- ☐ Only read committed events
- ☐ Optimized for recent events

Append Events/Snapshots

- 🟥 Validate aggregate sequence numbers
- ☐ Append multiple events at once
- ☐ Committed events protected against loss
- ☐ Append snapshots
- ☐ Constant performance as a function of storage size

Kafka

Pros

- Messaging focussed
- Extremely scalable

  in #total events

Cons

- Not scalable in #aggregates

Read Events/Snapshots

All for an aggregate

- 🟥 • Latests snapshot + later events
- 🟥 • All events
- ☐ All since point in time
- ☐ • Pushing new events
- ☐ Read back in write order
- ☐ Only read committed events
- ☐ Optimized for recent events

Append Events/Snapshot

- 🟥 Validate aggregate sequence numbers
- ☐ Append multiple events at once
- ☐ Committed events protected against loss
- ☐ Append snapshots
- ☐ Constant performance as a function of storage size

Cassandra

Pros

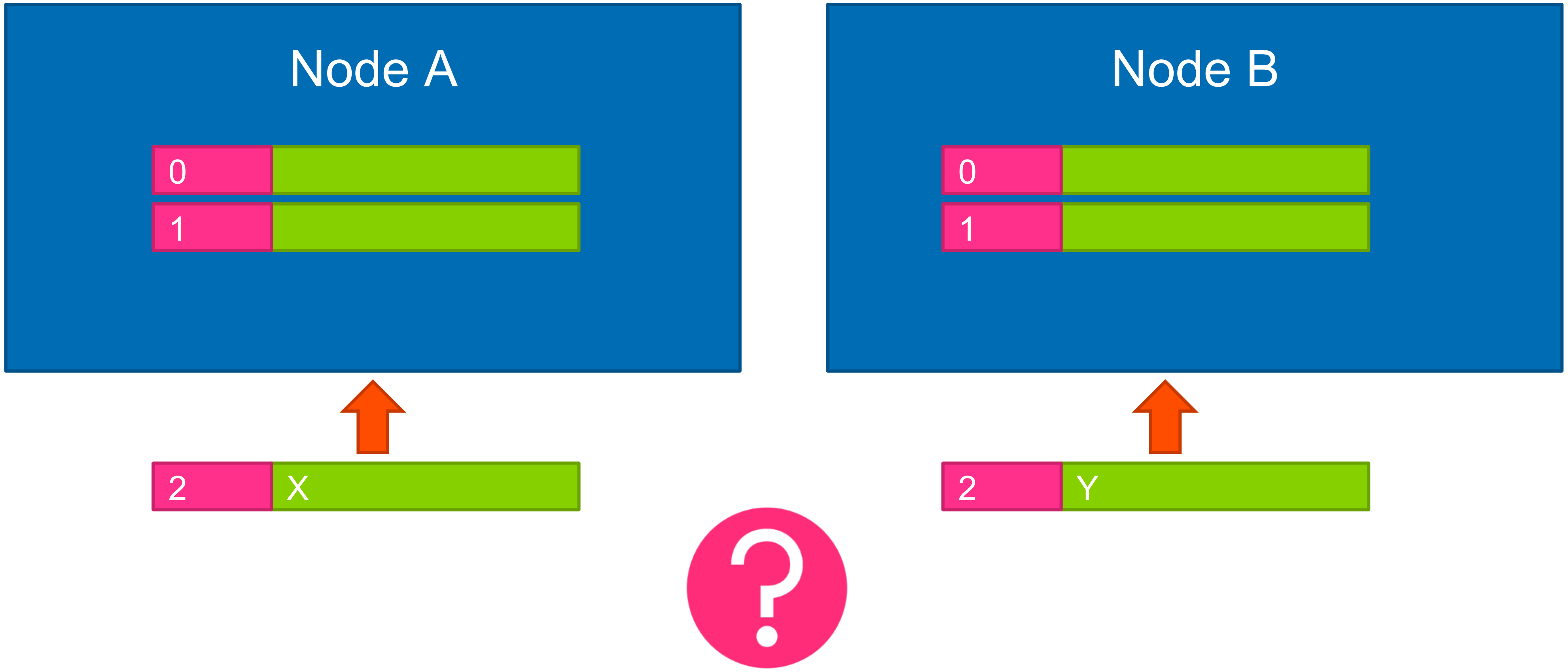
- Extremely scalable
- Multiple global datacenters
- Peer to peer
- Flexible, tunable consistency

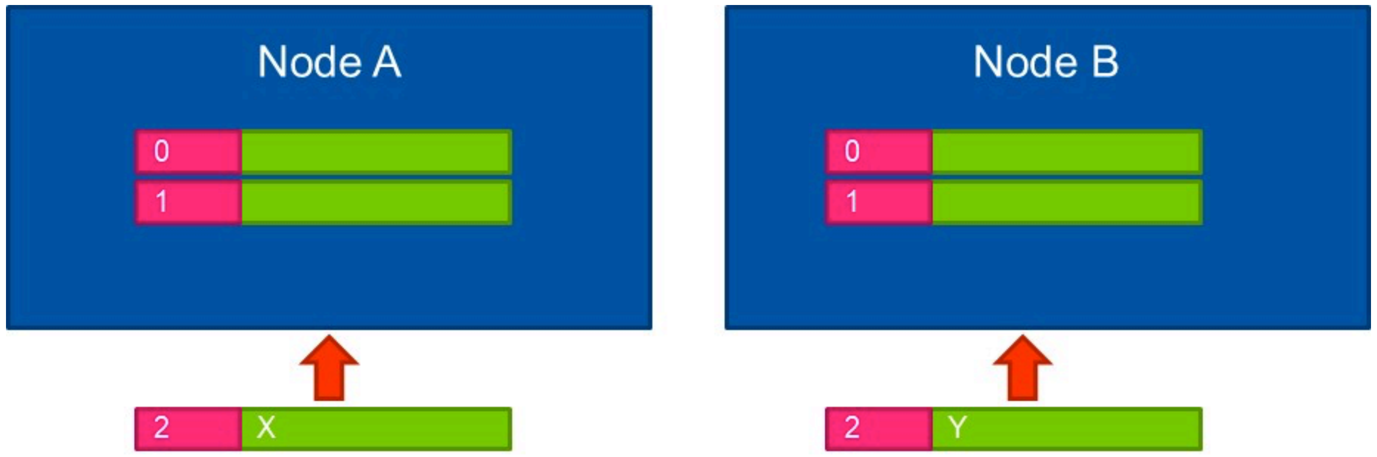
Id=8721
Seq = 0
	InvestmentAccountCreated(balance = 0, limit = 0)

Id=8721
Seq = 1
	MoneyDepositedToAccount(amount = 1000)

Id=8721
Seq = 2
	MoneyWithdrawnFromAccount(amount = 600)

Id=8721
Seq = 2
	MoneyWithdrawnFromAccount(amount = 700)


Node A
Node B
0
1
0
1
2
X
2
Y



```
INSERT INTO events(aggId, aggSeqNo, payload)
VALUES( ‘a’,  2 , ... )
IF NOT EXISTS
```

```
INSERT INTO events(aggId, aggSeqNo, payload)
VALUES( ‘a’,  2 , ... )
IF NOT EXISTS
```

“Behind the scenes, Cassandra is making four round trips between a node proposing a lightweight transaction and any needed replicas in the cluster to ensure proper execution so performance is affected. Consequently, reserve lightweight transactions for those situations where they are absolutely necessary; Cassandra’s normal eventual consistency can be used for everything else.”

Source: https://docs.datastax.com - our highlighting

Read Events/Snapshots

All for an aggregate

- ☐ • Latests snapshot + later events
- ☐ • All events
- ☐ All since point in time
- ☐ • Pushing new events
- ☐ Read back in write order
- ☐ Only read committed events
- ☐ Optimized for recent events

Append Events/Snapshots

- 🟥 Validate aggregate sequence numbers
- ☐ Append multiple events at once
- ☐ Committed events protected against loss
- ☐ Append snapshots
- ☐ Constant performance as a function of storage size

Cassandra

Pros

- Extremely scalable
- Multiple global datacenters
- Peer 2 peer
- Flexible, tunable consistency

Cons

- Can’t guarantee event store consistency efficiently

Read Events/Snapshots

All for an aggregate
- ☐ • Latests snapshot + later events
- ☐ • All events
- ☐ All since point in time
- ☐ • Pushing new events
- ☐ Read back in write order
- ☐ Only read committed events
- ☐ Optimized for recent events

Append Events/Snapshot

- 🟥 Validate aggregate sequence numbers
- ☐ Append multiple events at once
- ☐ Committed events protected against loss
- ☐ Append snapshots
- ☐ Constant performance as a function of storage size

Built-for purpose event stores?

“EventStore” by Greg Young

- Written in .NET, and generally seen as part of .NET ecosystem
- Places heavy emphasis on projection logic (JavaScript) inside the event store.

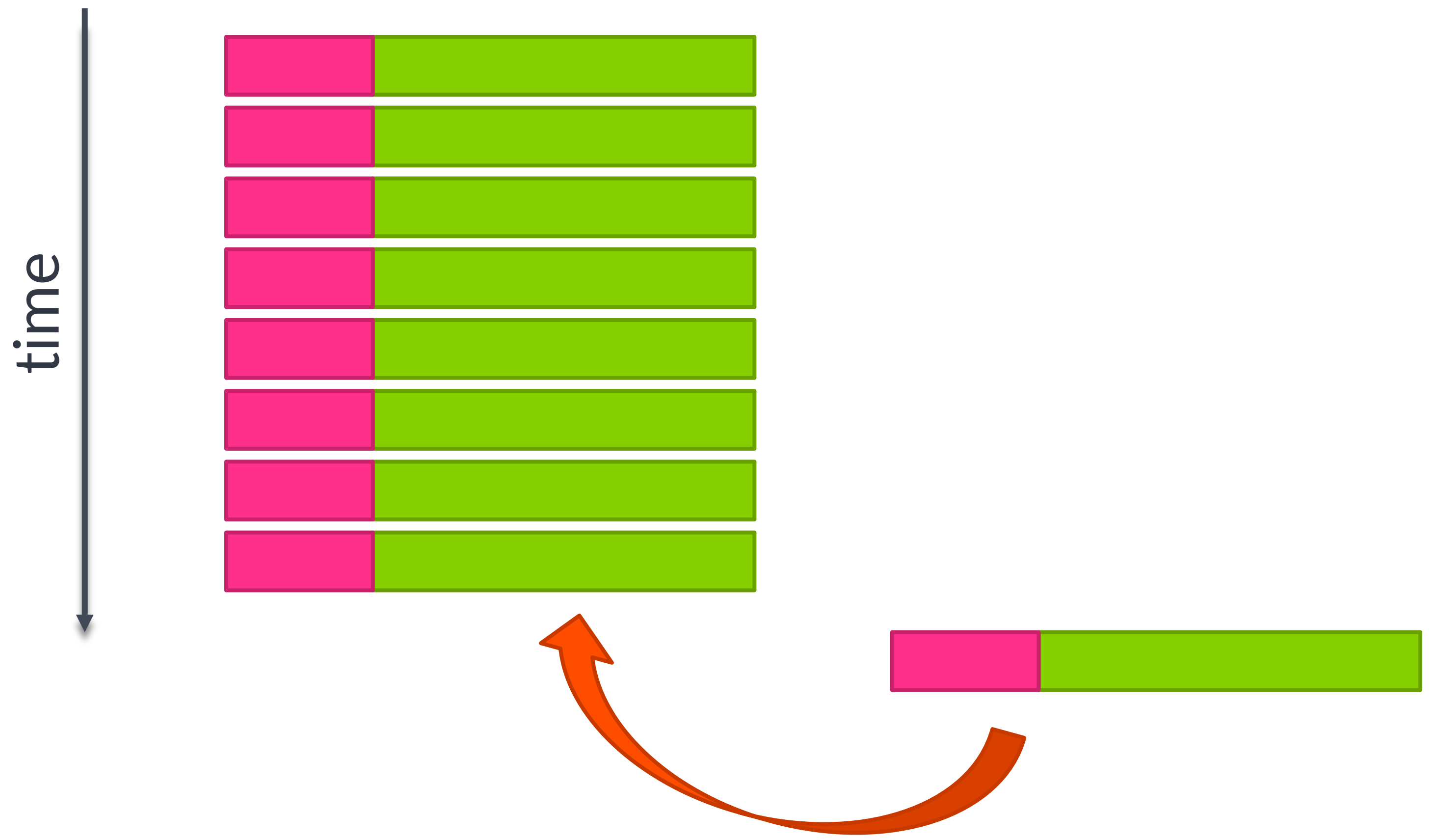
PumpkinDB

- Separate 'database programming environment' inspired by M/MUMPS
- Lots of logic would have to implemented in "PumpkinScript" rather than Java.

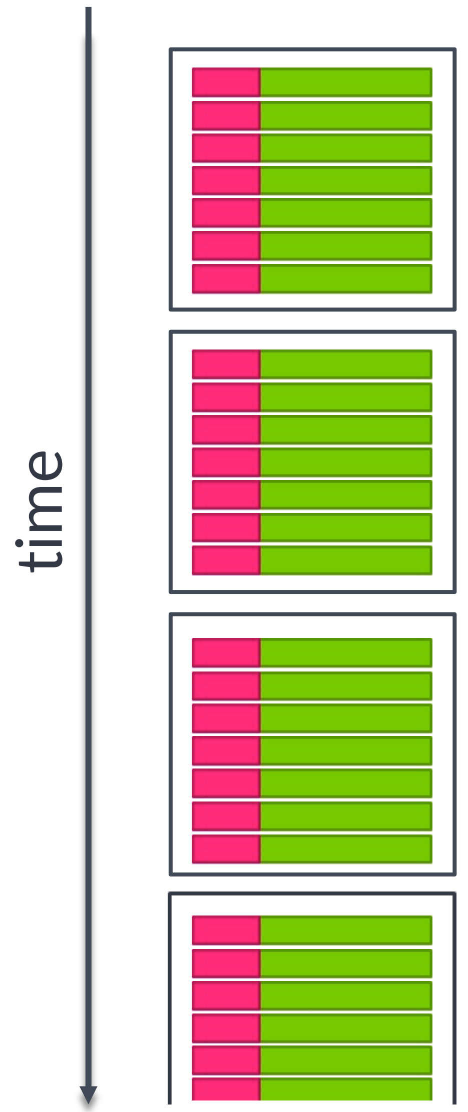
Architecture and features

AxonServer

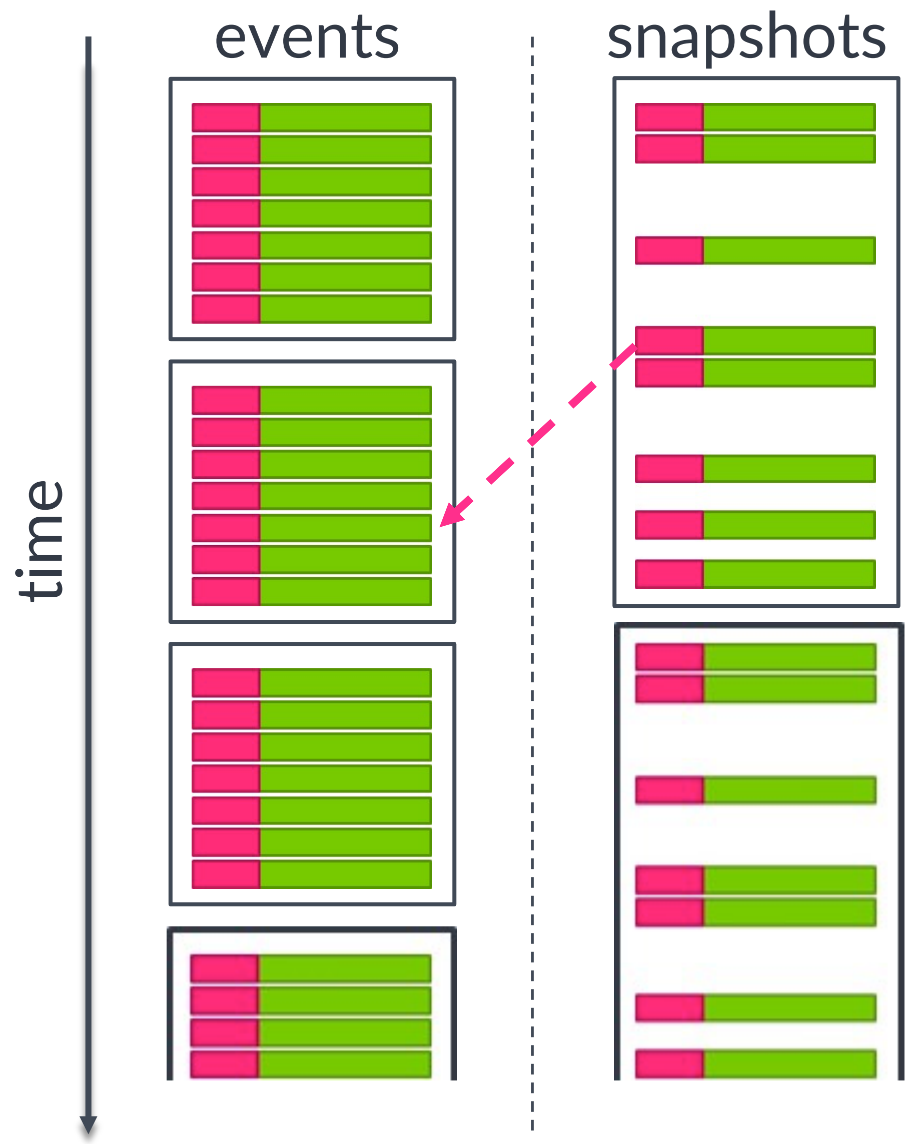
- Built ‘from scratch’ in Java.
- Purpose-built for event sourcing
- Manages files directly - no underlying database system.
- Open interfaces based on HTTP+JSON and gRPC
- Drop-in event store implementation for Axon Framework



Append-only by design



Event-stream split into **segments**



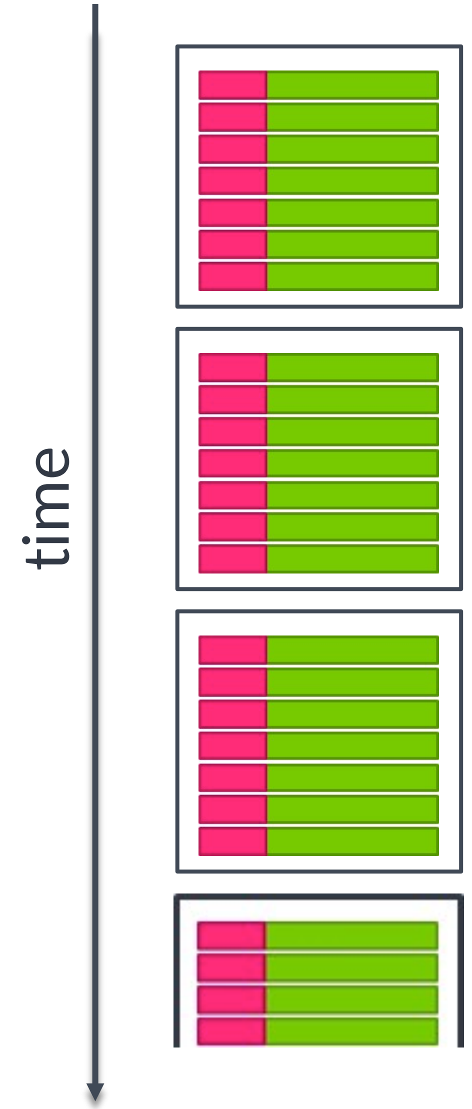
Built-in support for **snapshots**

In each segment, we can efficiently search on **aggregate id + seq no**

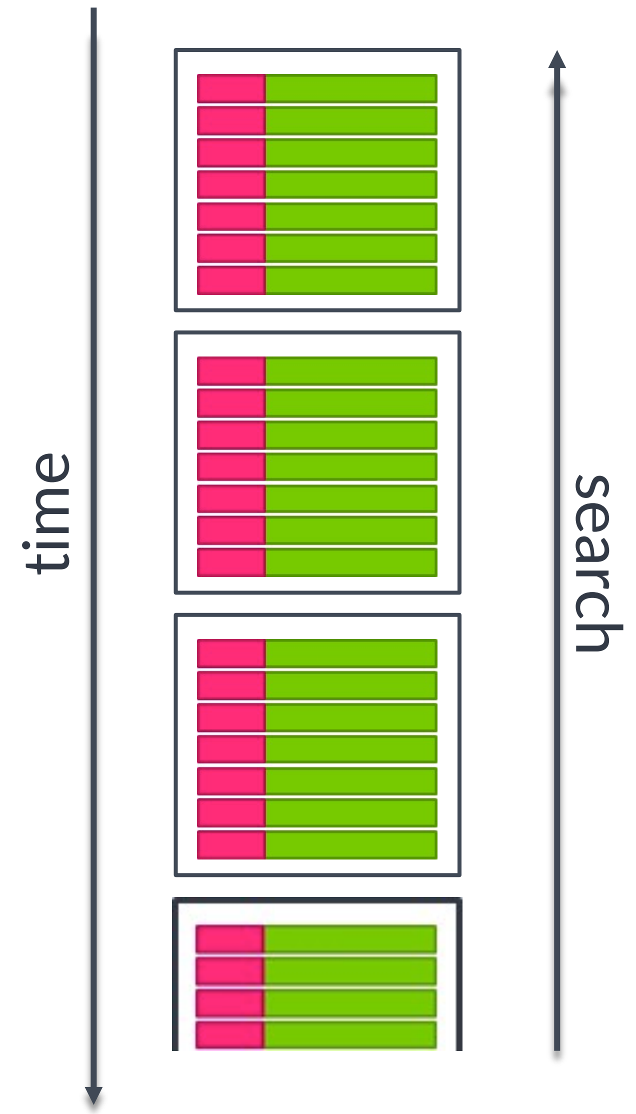
Data

Index

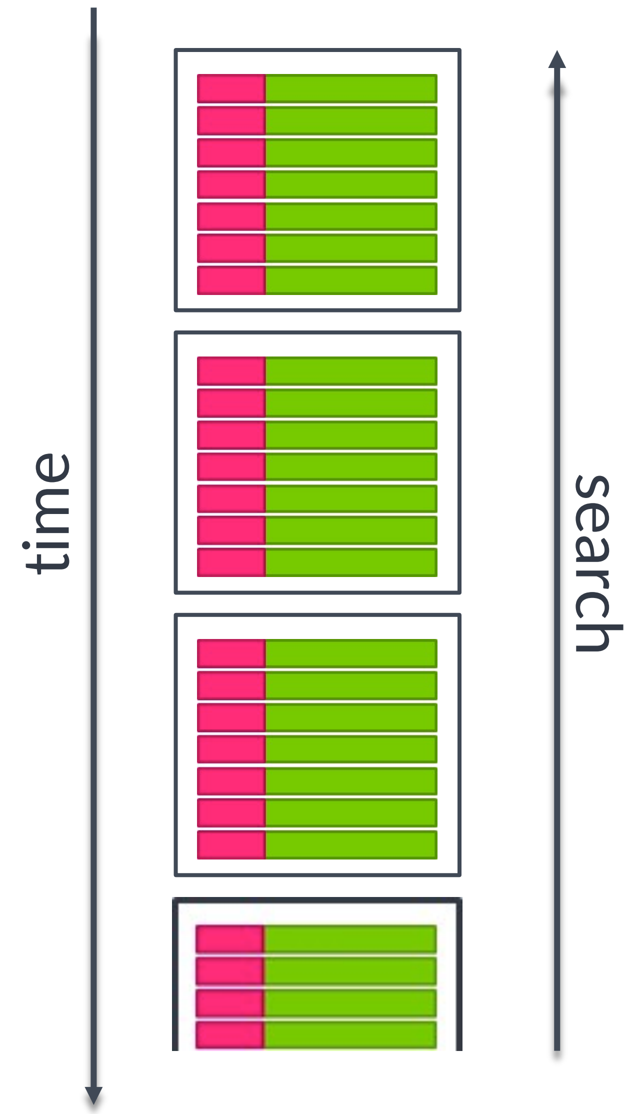
Bloom-filter

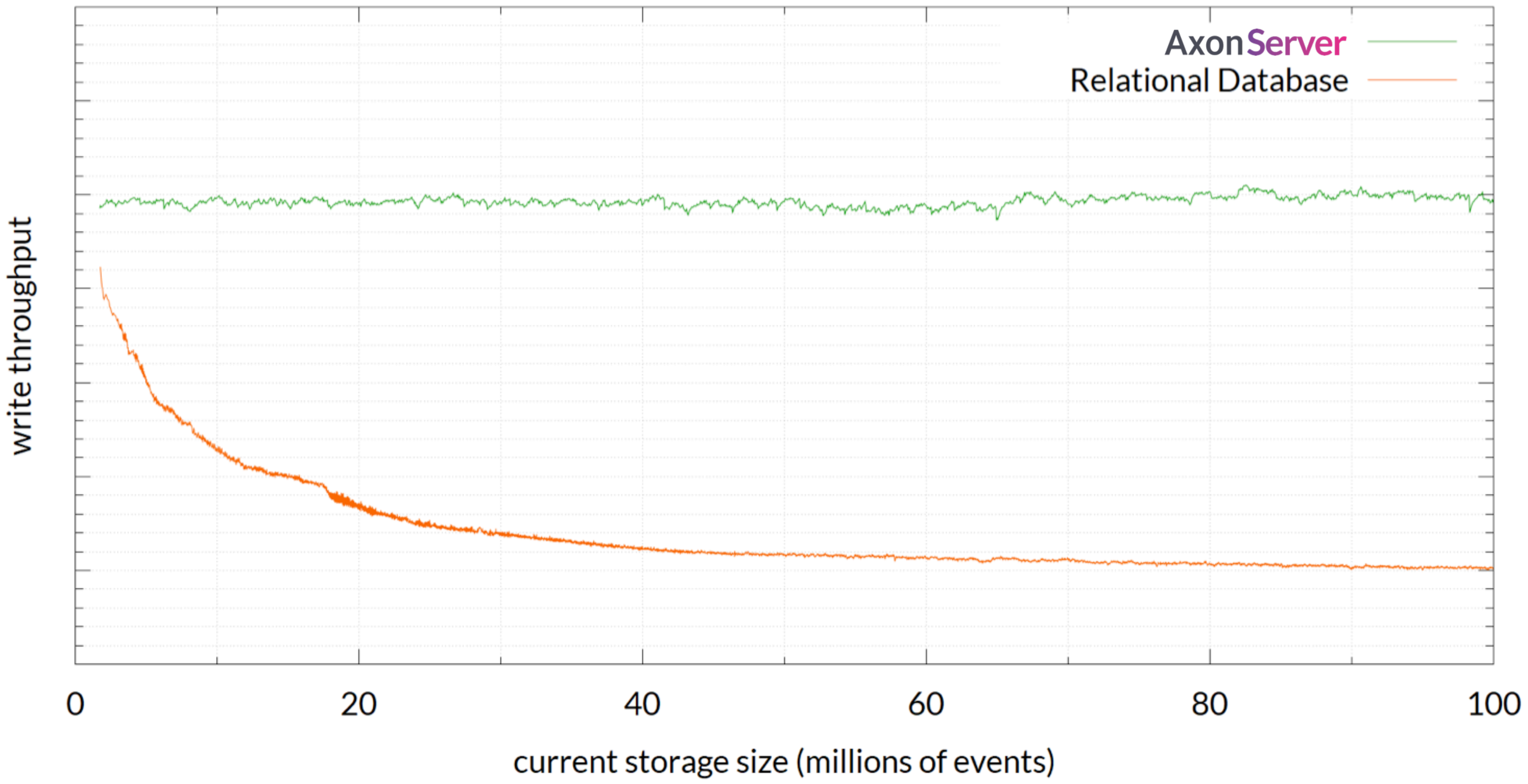


Searching for Aggregate's events goes **backwards in time**

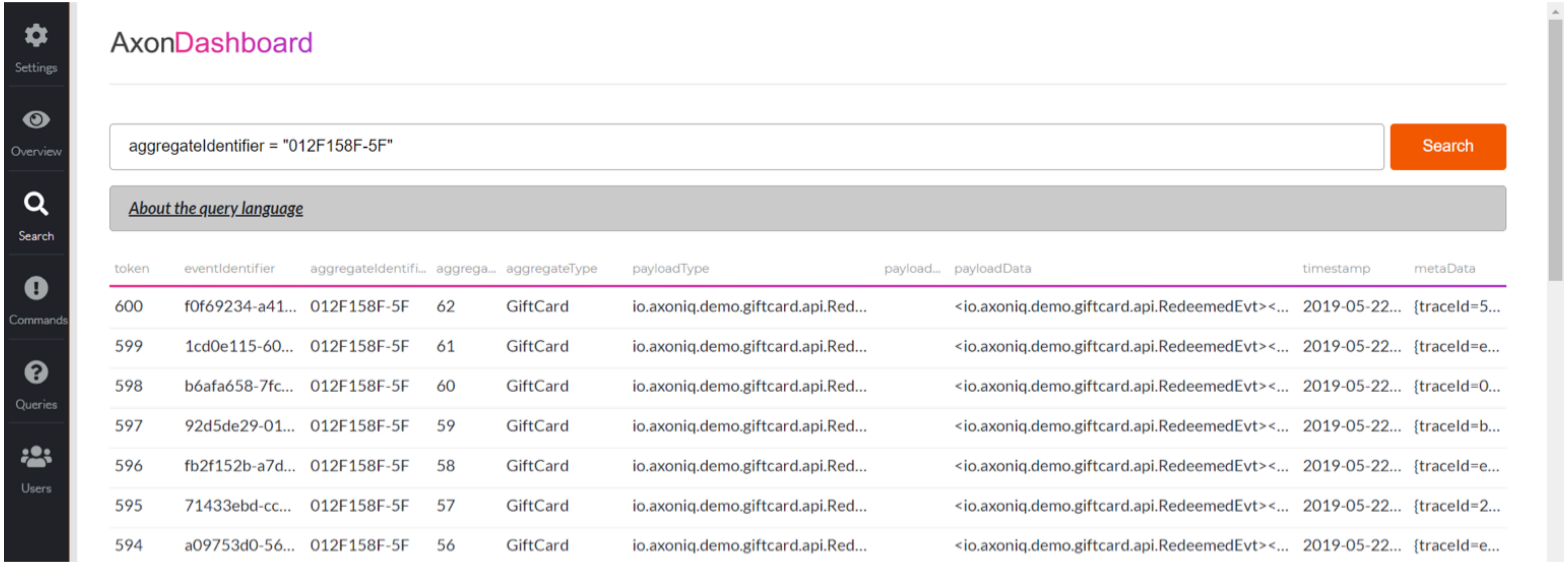


Recent segments are kept **in-memory**




AxonServer
Relational Database
write throughput
0
20
40
60
80
100
current storage size (millions of events)

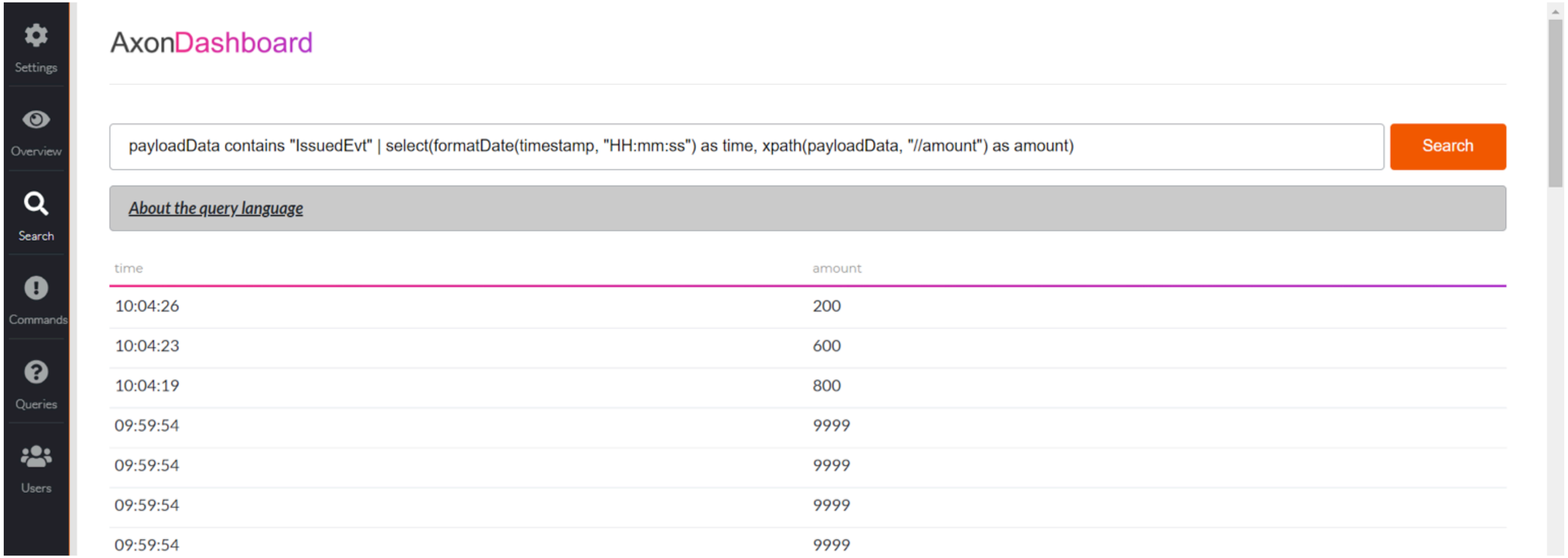
Support for **ad-hoc queries**, through a GUI and an API



aggregateIdentifier = "012F158F-5F"

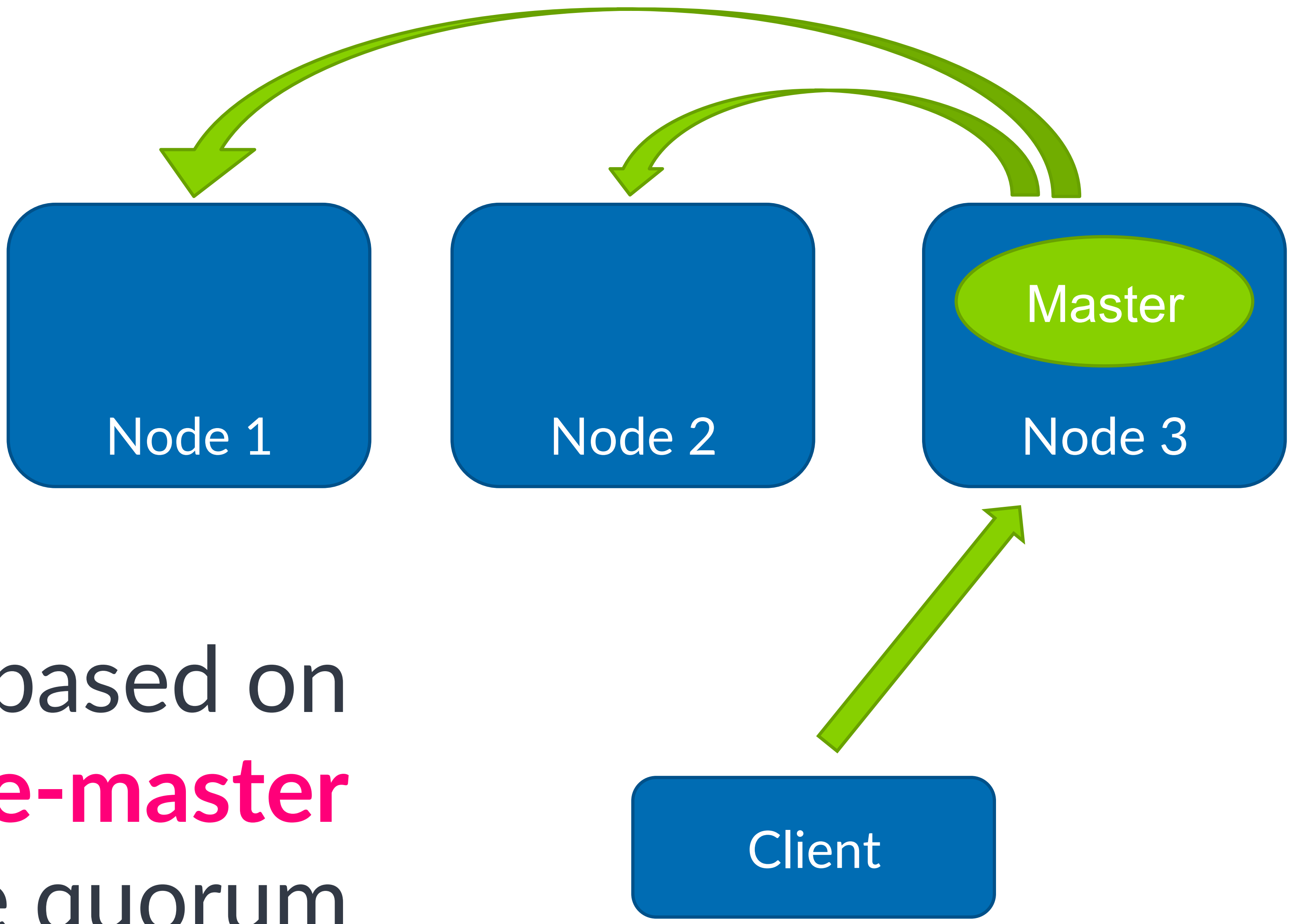
token	eventIdentifier	aggregateIdentifi...	aggrega...	aggregateType	payloadType	payload...	payloadData	timestamp	metaData
600	f0f69234-a41...	012F158F-5F	62	GiftCard	io.axoniq.demo.giftcard.api.Red...		<io.axoniq.demo.giftcard.api.RedeemedEvt><...	2019-05-22...	{traceId=5...
599	1cd0e115-60...	012F158F-5F	61	GiftCard	io.axoniq.demo.giftcard.api.Red...		<io.axoniq.demo.giftcard.api.RedeemedEvt><...	2019-05-22...	{traceId=e...
598	b6afa658-7fc...	012F158F-5F	60	GiftCard	io.axoniq.demo.giftcard.api.Red...		<io.axoniq.demo.giftcard.api.RedeemedEvt><...	2019-05-22...	{traceId=0...
597	92d5de29-01...	012F158F-5F	59	GiftCard	io.axoniq.demo.giftcard.api.Red...		<io.axoniq.demo.giftcard.api.RedeemedEvt><...	2019-05-22...	{traceId=b...
596	fb2f152b-a7d...	012F158F-5F	58	GiftCard	io.axoniq.demo.giftcard.api.Red...		<io.axoniq.demo.giftcard.api.RedeemedEvt><...	2019-05-22...	{traceId=e...
595	71433ebd-cc...	012F158F-5F	57	GiftCard	io.axoniq.demo.giftcard.api.Red...		<io.axoniq.demo.giftcard.api.RedeemedEvt><...	2019-05-22...	{traceId=2...
594	a09753d0-56...	012F158F-5F	56	GiftCard	io.axoniq.demo.giftcard.api.Red...		<io.axoniq.demo.giftcard.api.RedeemedEvt><...	2019-05-22...	{traceId=e...

Support for **ad-hoc queries**, through a GUI and an API

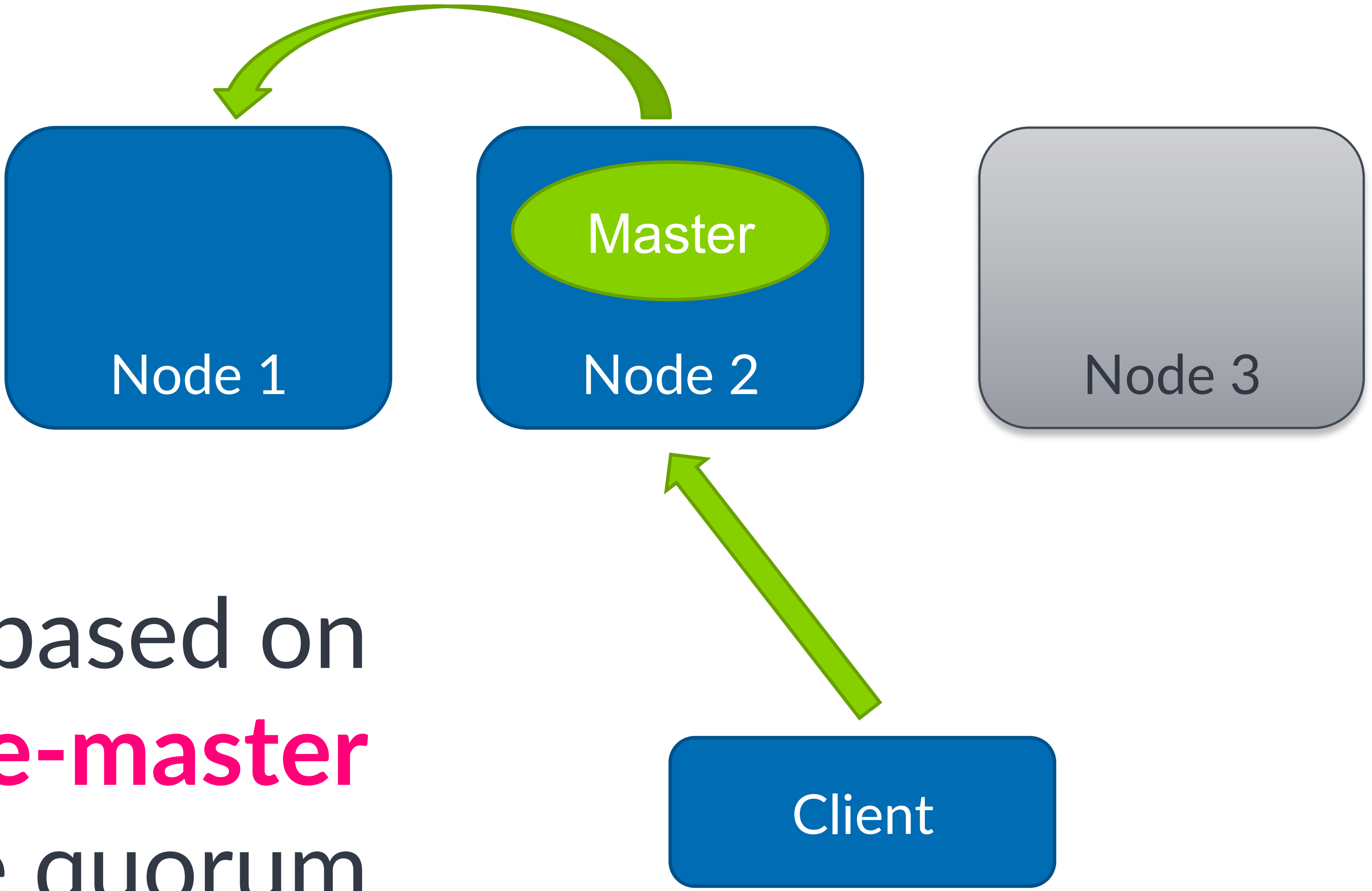


```
payloadData contains "IssuedEvt" | select(formatDate(timestamp, "HH:mm:ss") as time, xpath(payloadData, "//amount") as amount)
```

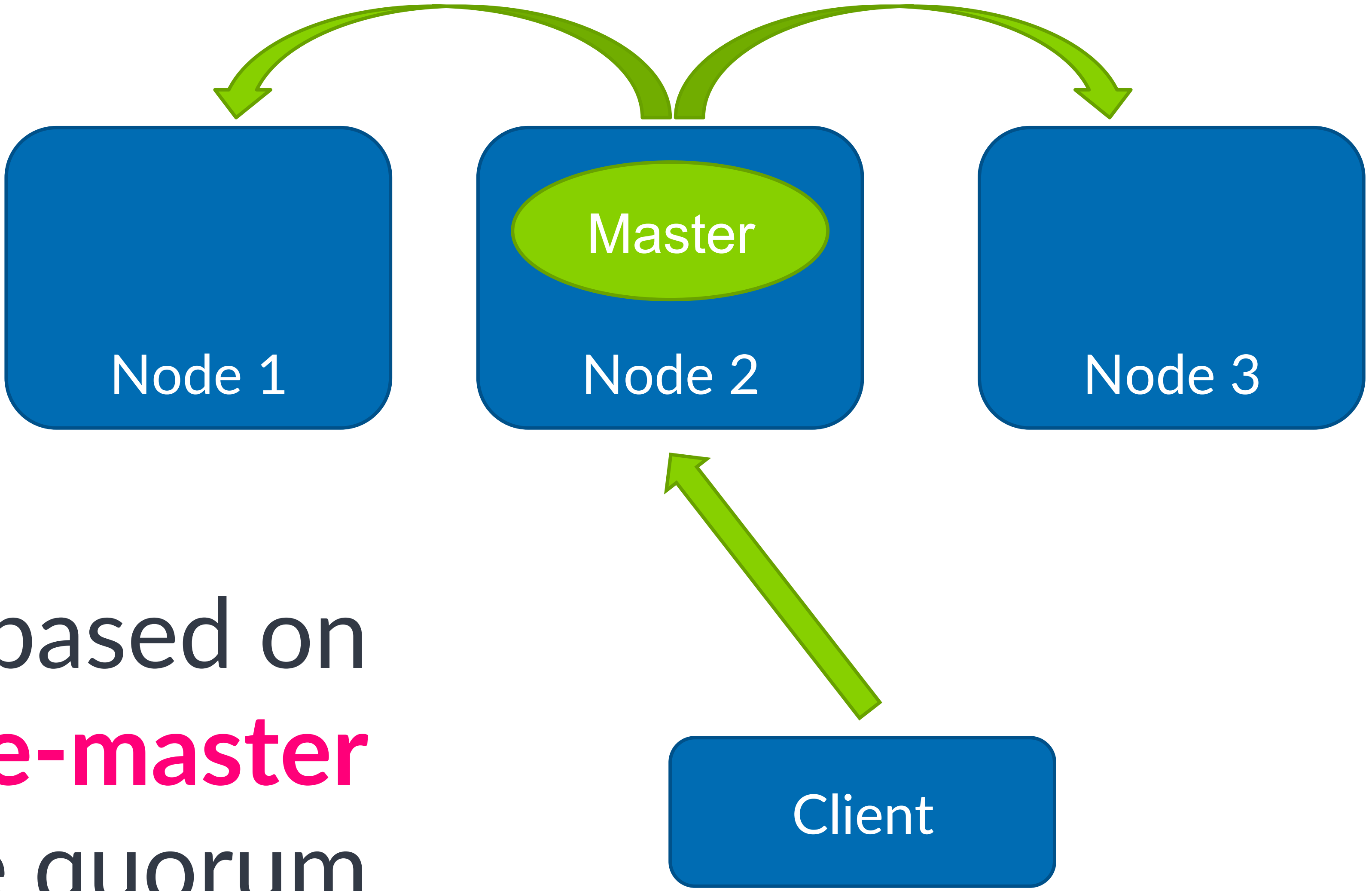
time	amount
10:04:26	200
10:04:23	600
10:04:19	800
09:59:54	9999
09:59:54	9999
09:59:54	9999
09:59:54	9999



Clustering based on **floating single-master** with write quorum



Clustering based on **floating single-master** with write quorum



Clustering based on **floating single-master** with write quorum

How do I get started?

AxonFramework

GitHub

maven Gradle

The Central Repository

AxonServer

GitHub

docker ZIP

AxonServer Enterprise

sales@axoniq.io

ZIP

How do I get started?

axoniq.io/download