Pravega: Rethinking Storage for Streams

Flavio Junqueira

fpj@pravega.io

Pravega - http://pravega.io

Linux Foundation CNCF – Webinar

April 2020

About the speaker

- Dell EMC
  - Senior Distinguished Engineer
- On Pravega since 2016
- Background
  - Distributed computing
  - Research: Microsoft, Yahoo!
  - Worked on various Apache projects
  - *E.g.*, Apache ZooKeeper, Apache BookKeeper
- Contact
  - Email: fpj@pravega.io
  - Twitter: @fpjunqueira


Many sources continuously generating data

Unbounded data streams

Social networks
Online shopping

Stream of user events
- Status updates
- Online transactions

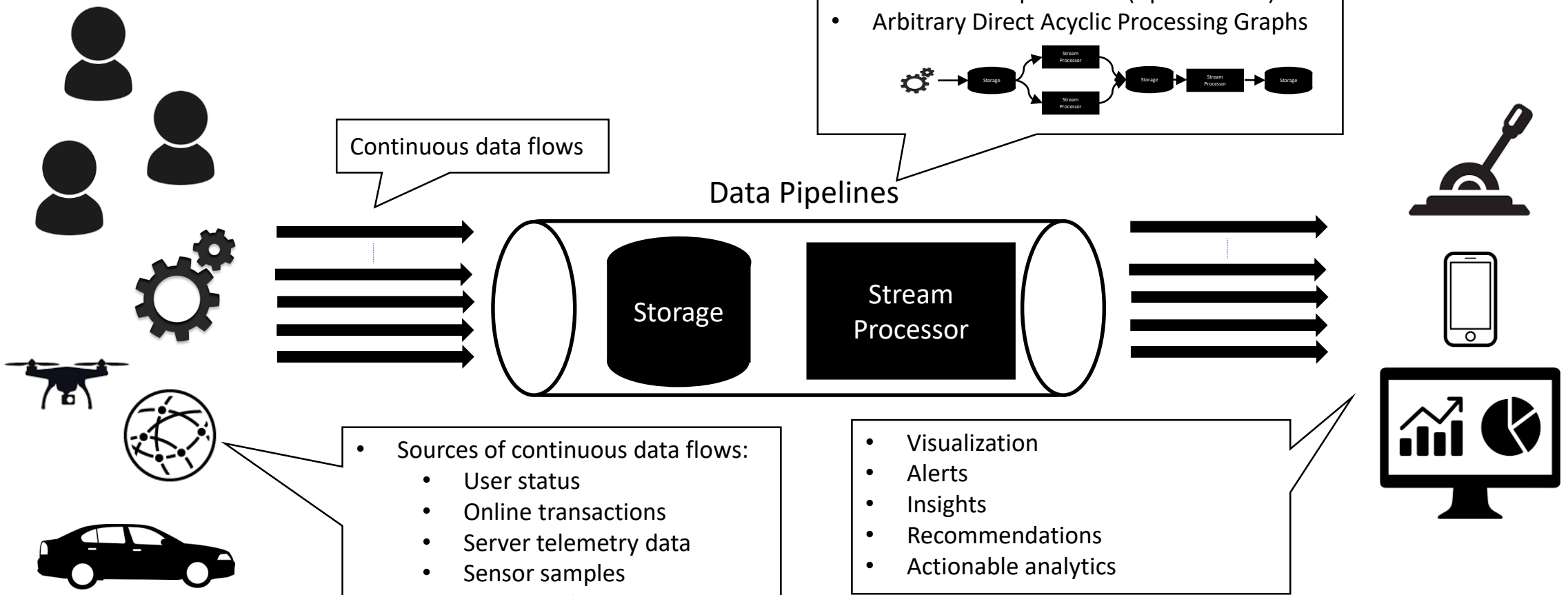
Server monitoring

Telemetry streams
- CPU, memory, disk utilization

Sensors (IoT)

Stream of sensor events
- Temperature samples
- Radar and image

Landscape

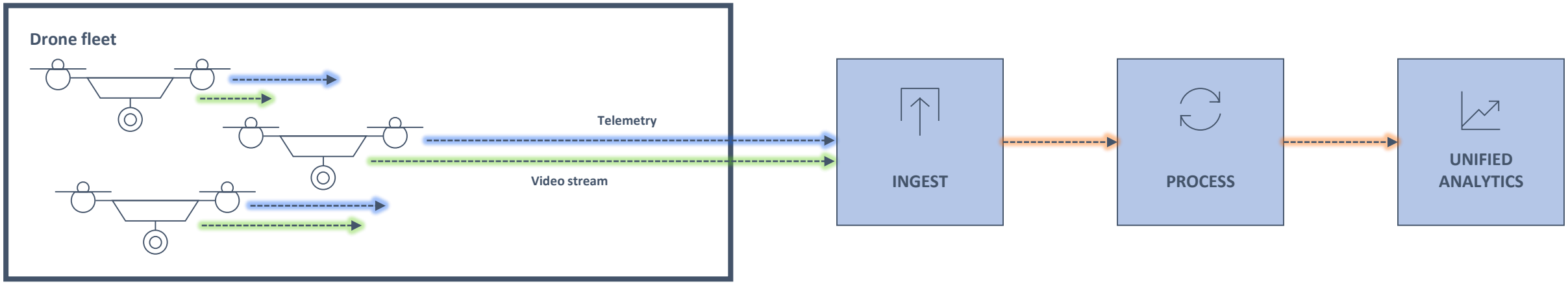


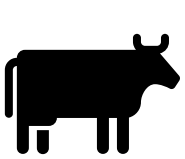
- Core elements
  - Storage (Pravega)
  - Stream processor (Apache Flink)
- Arbitrary Direct Acyclic Processing Graphs

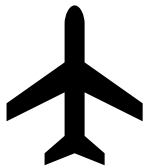
- Sources of continuous data flows:
  - User status
  - Online transactions
  - Server telemetry data
  - Sensor samples
  - Connected cars
  - Drone videos

- Visualization
- Alerts
- Insights
- Recommendations
- Actionable analytics

Unbounded data streams

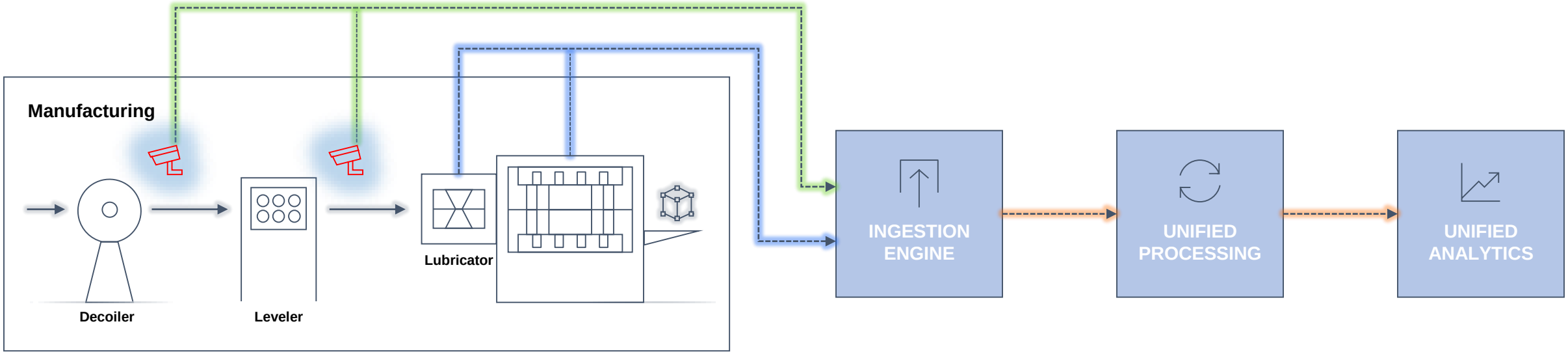






- From cattle health
- To airplane inspection between flights

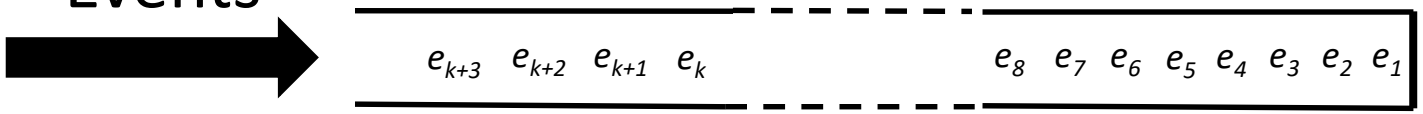
Industrial Sensor Data Anomaly Detection



Data streams – Sequential

Server,
Sensor,
etc.

Events

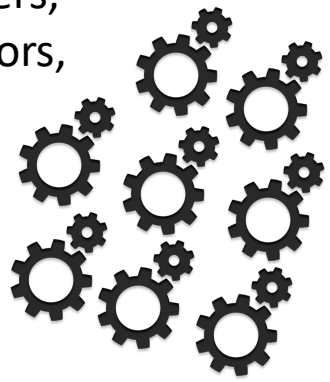


Tail

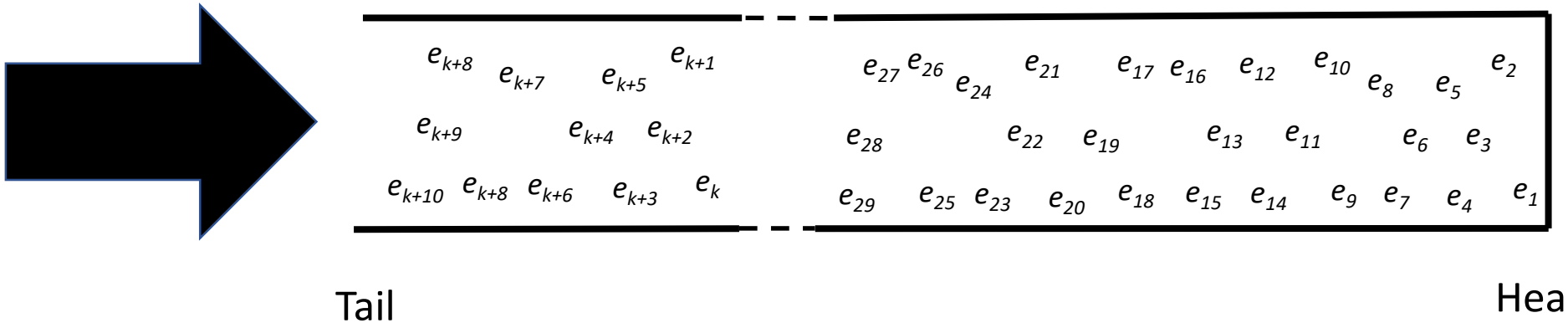
Head

Data streams - Parallelism

Servers,
Sensors,
etc.



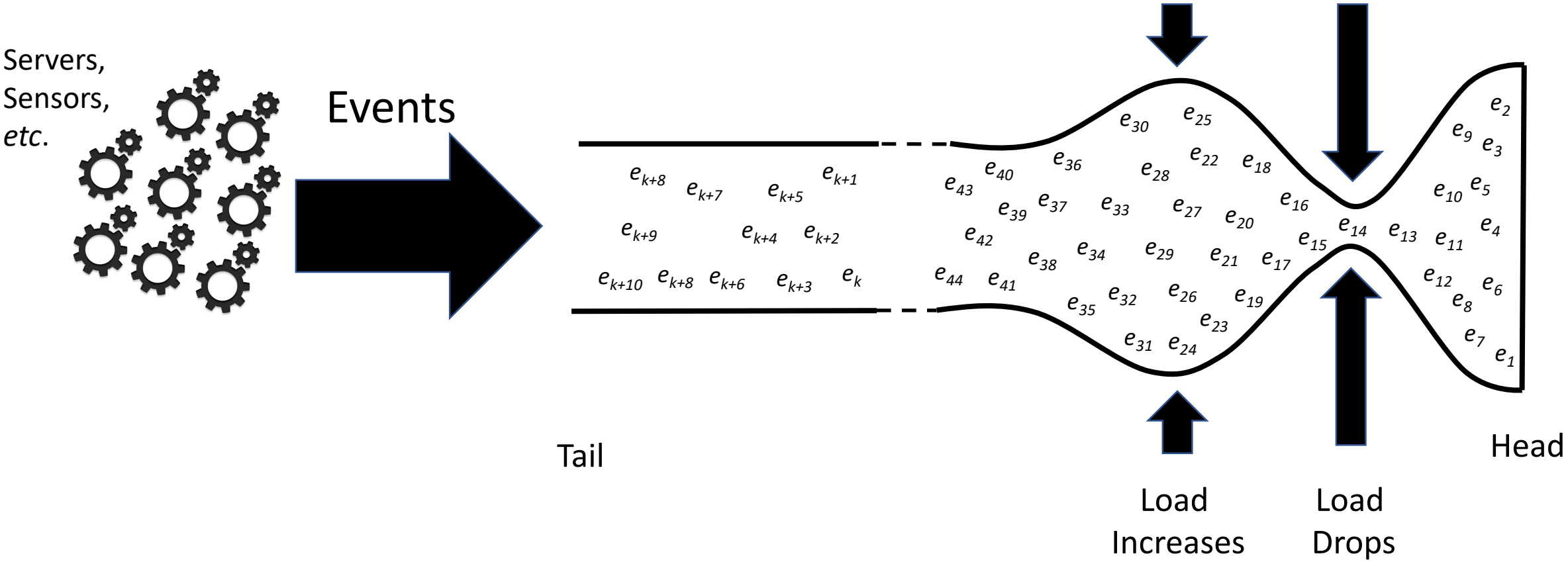
Events



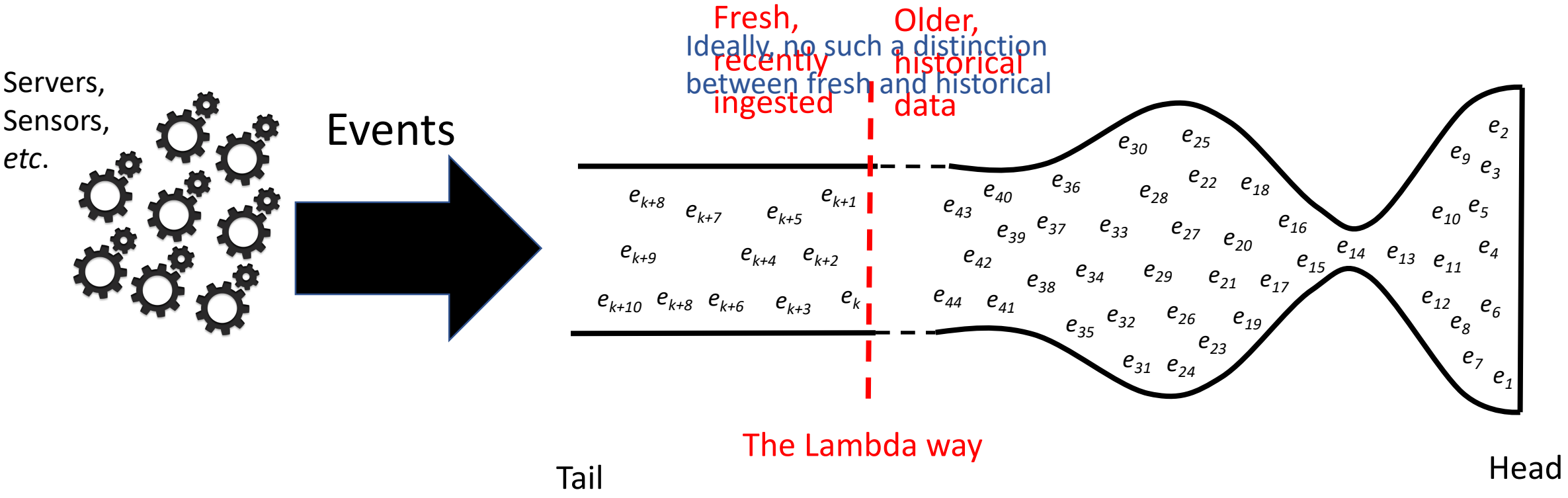
Tail

Head

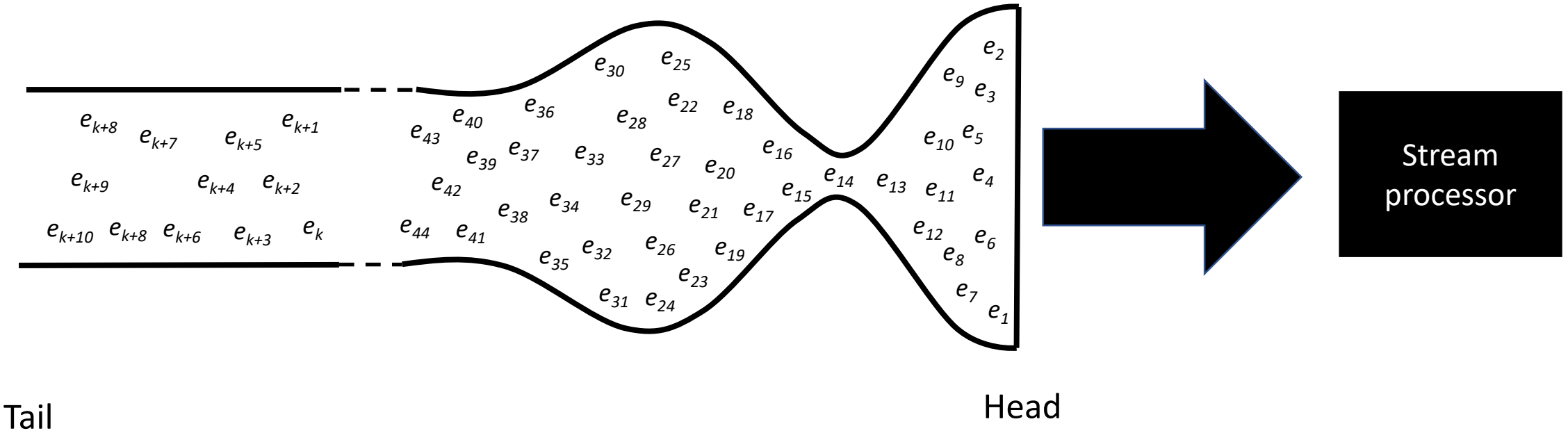
Data streams – Traffic fluctuation



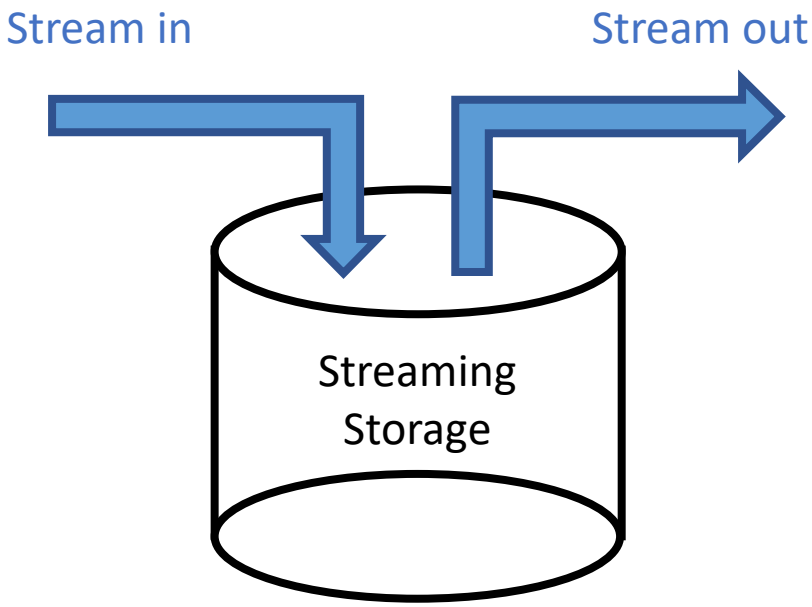
Data streams – Unbounded



Data streams – Read scalability



Data stream: cloud-native, storage primitive



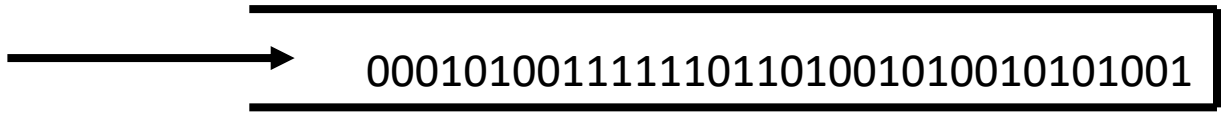
- Unbounded data
- Elastic
- Consistent
- Tailing and historical data analytics

- **Cloud native**

Pravega
Reliable streaming storage

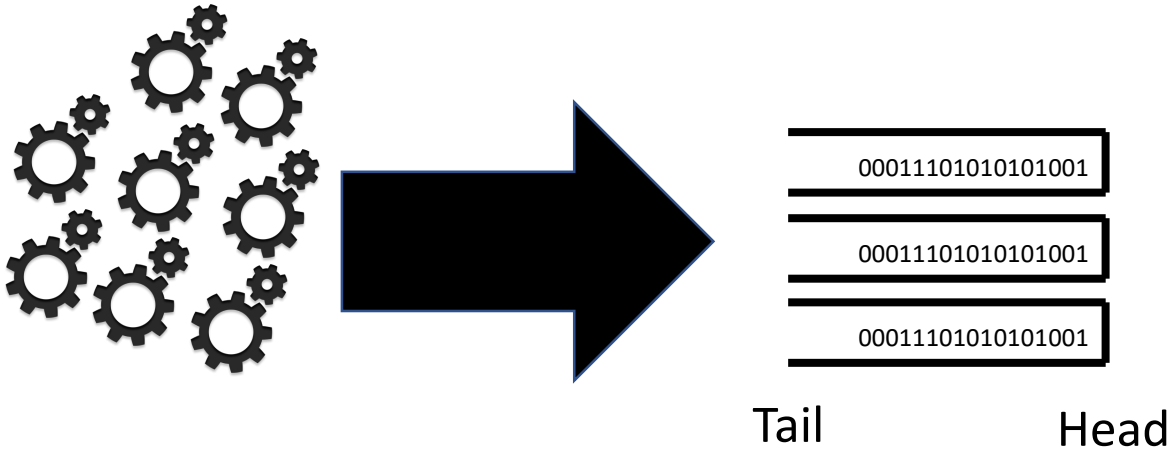
Introducing Pravega

Stream Segment



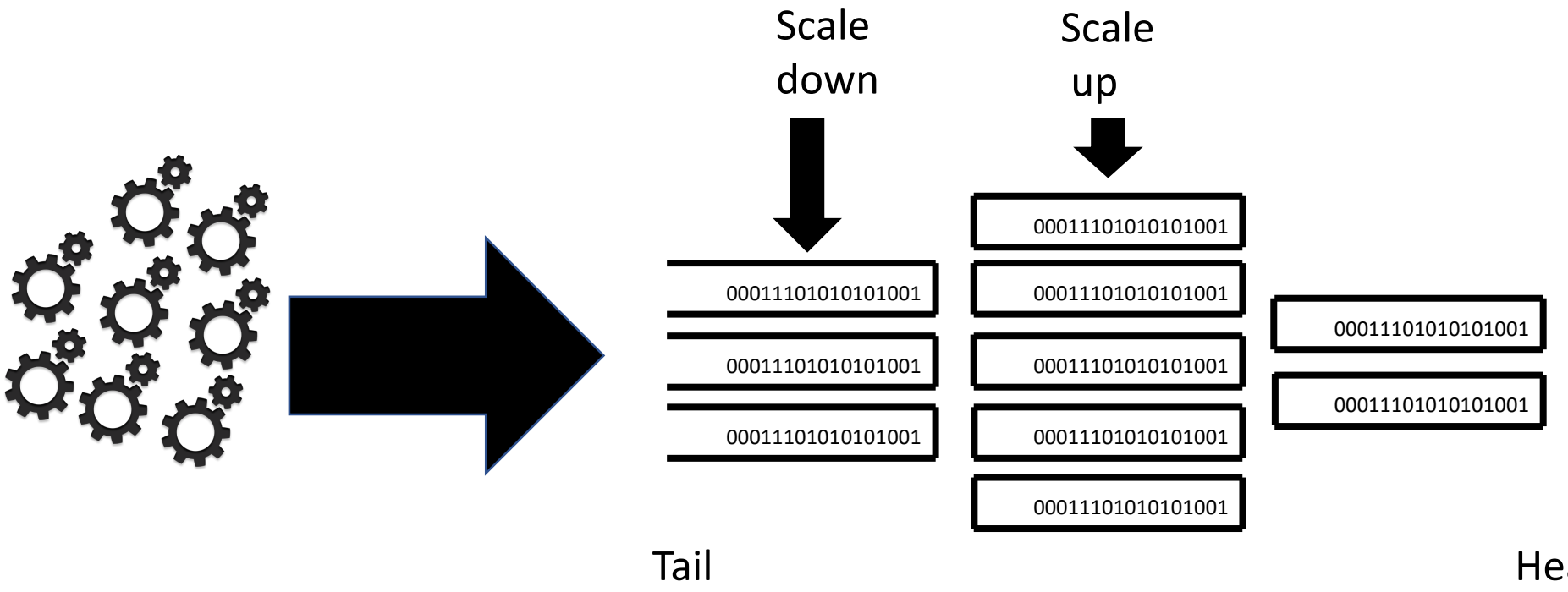
- Storage unit
- Append-only sequences of bytes
- … bytes, not events, messages or records
- Flexible in the formats it can store
- Events: expect serializer implementation

Stream Segment – Parallelism



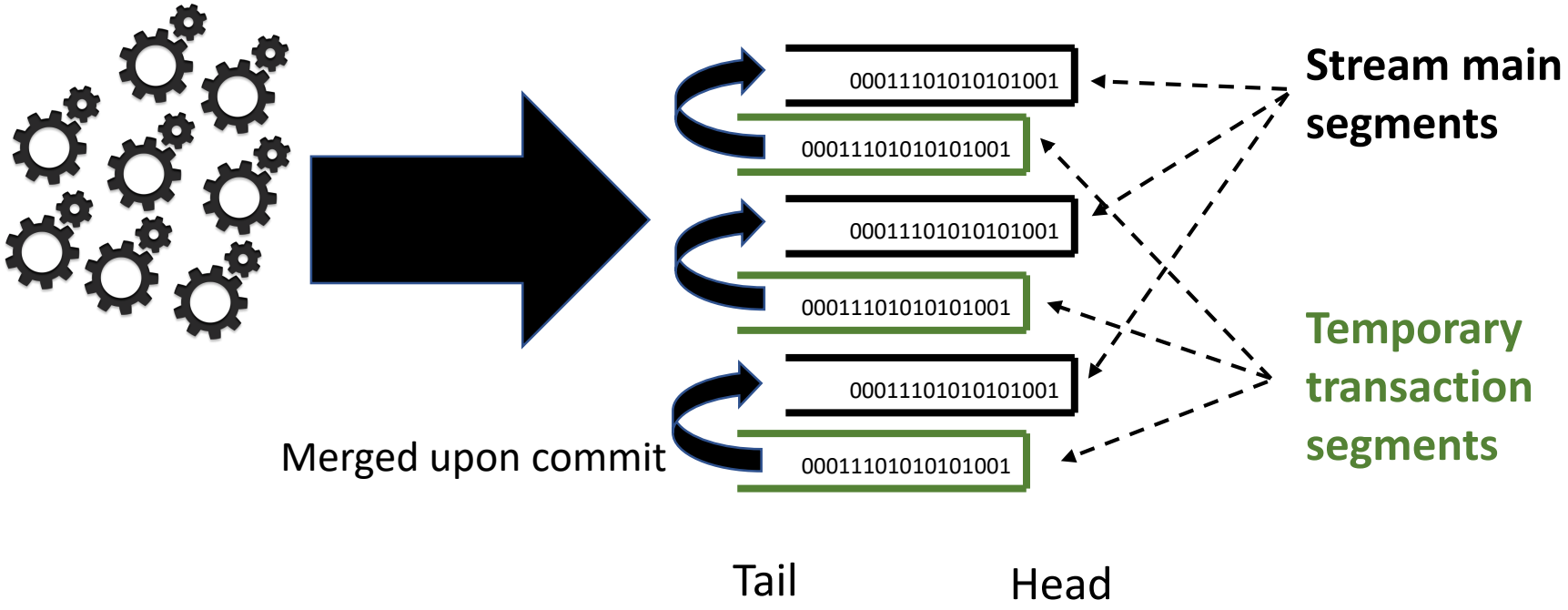
- Sources append to segments in parallel
- **Routing keys** map appends to **segments**

Stream Segment – Scaling

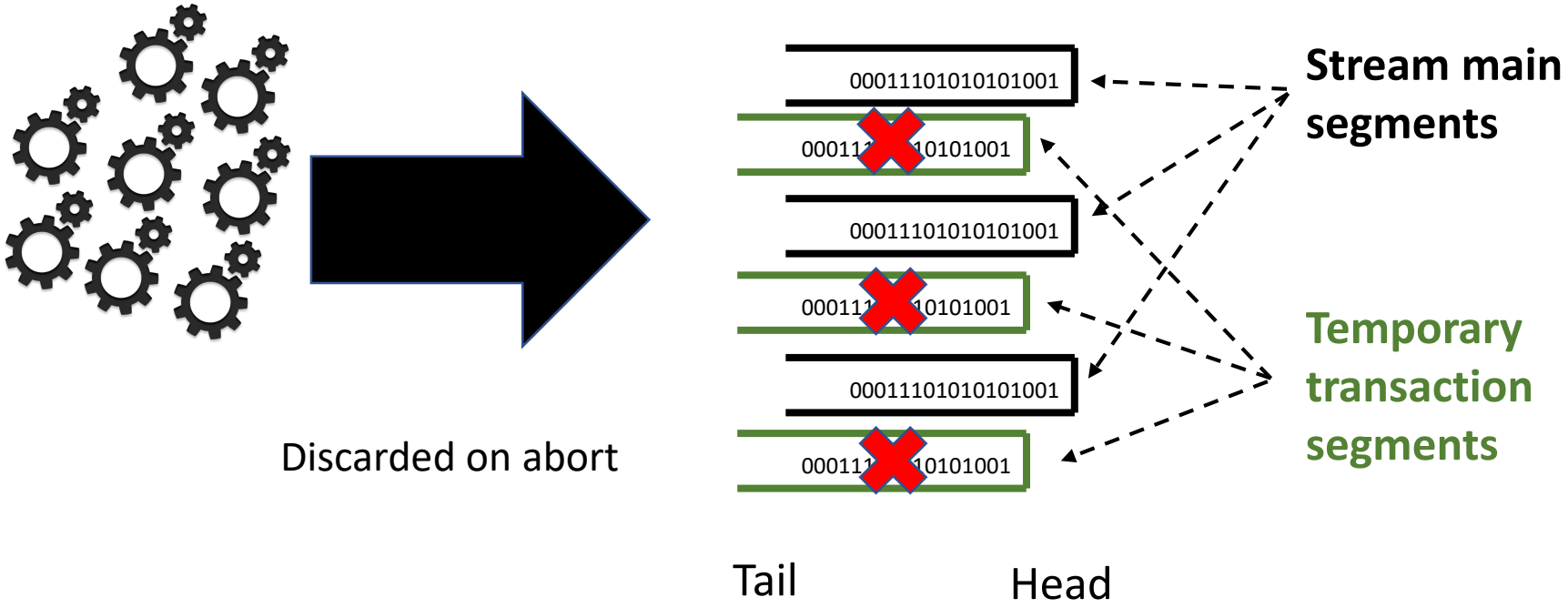


- Degree of parallelism changes dynamically
- **Auto-scaling**: reacts to workload changes

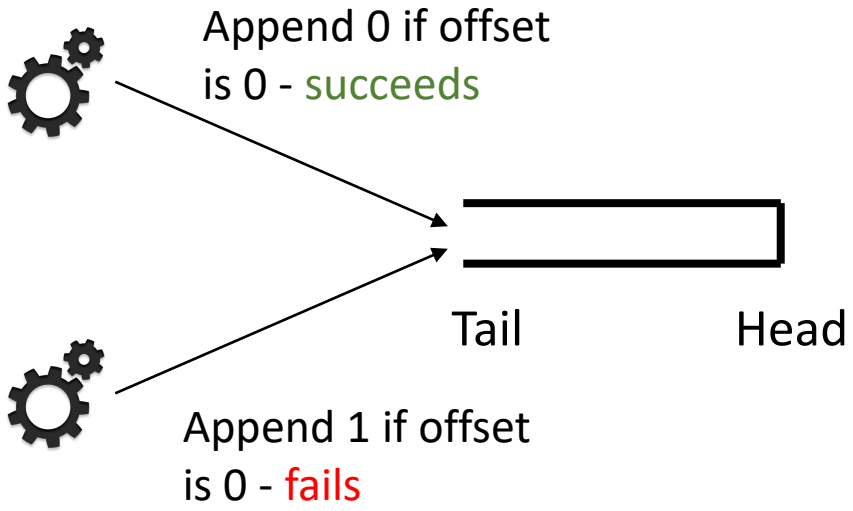
Stream Segment - Transactions



Stream Segment - Transactions



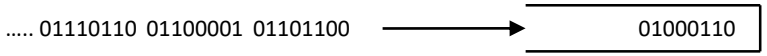
Stream Segment – Appending conditionally



- Revisioned streams
  - Conditional appends
    - Compare expected and current revisions
    - Revision implementation uses segment offset
- State synchronizer
  - Builds on revisioned streams
  - Replicated state machines
  - Optimistic concurrency

Stream scaling
(a.k.a, changing the degree of parallelism)

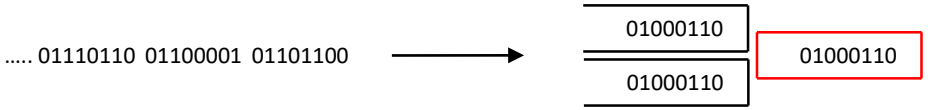
Scaling a stream



- Stream has one segment

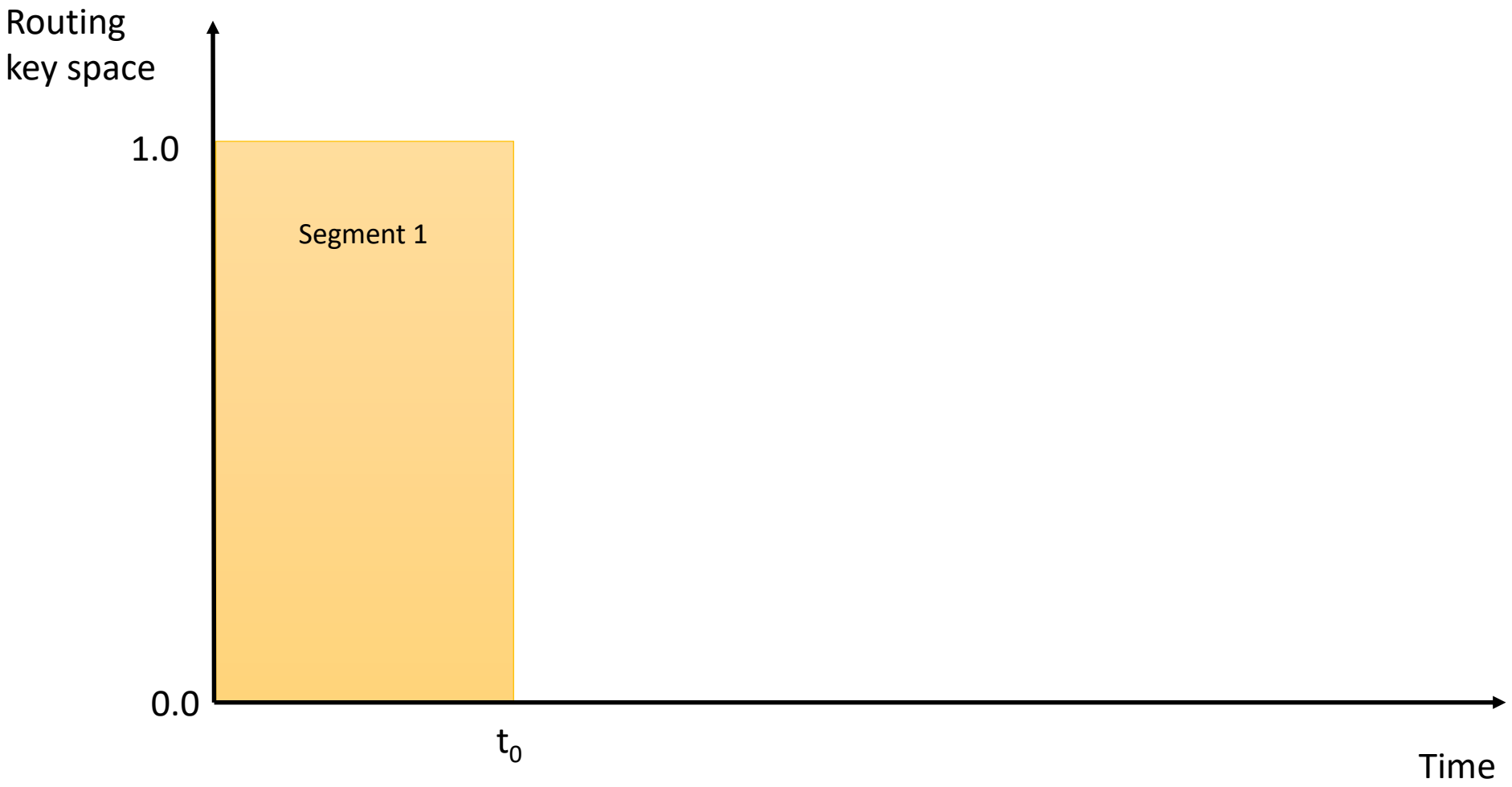
1

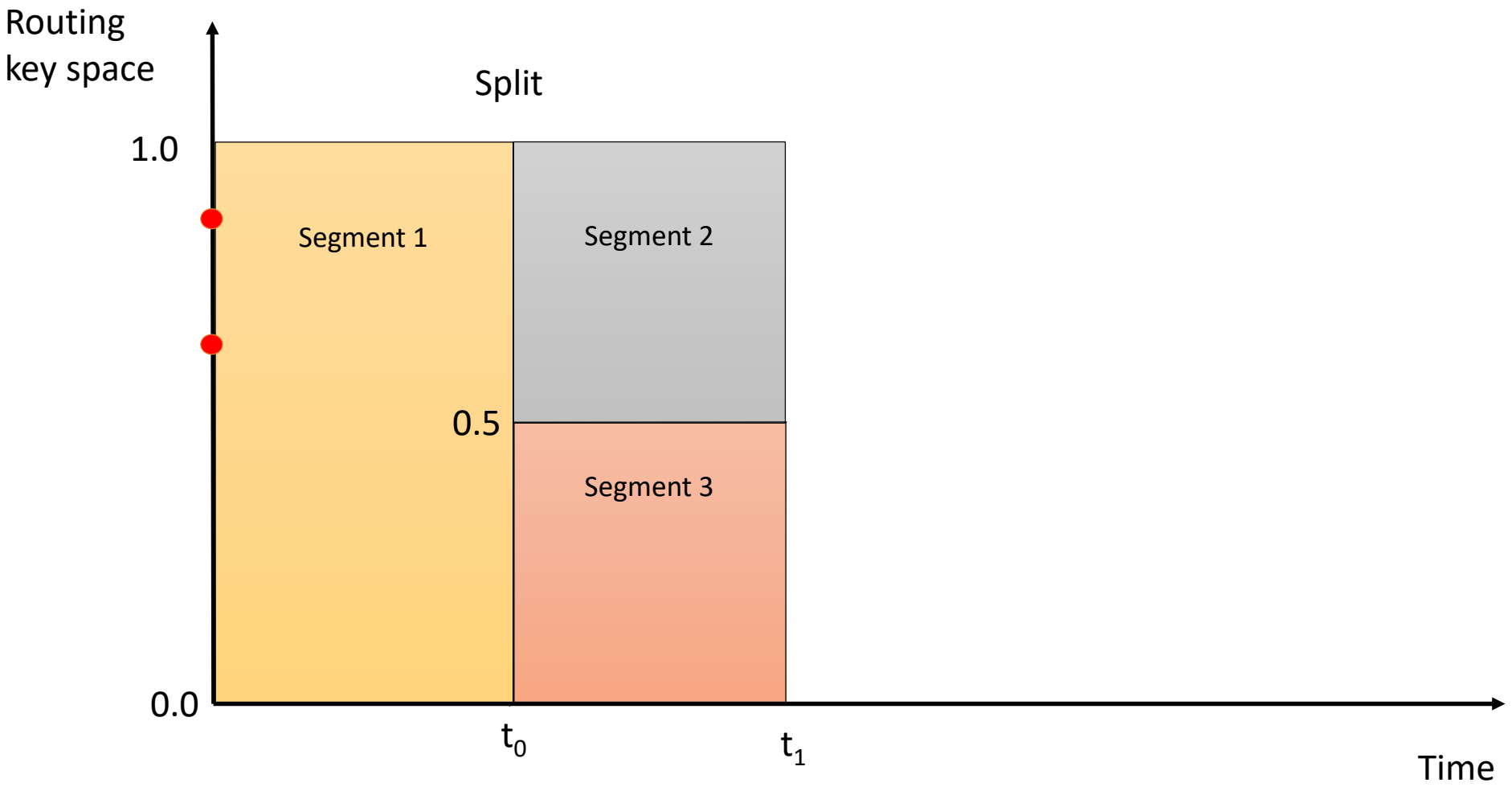
- Auto or manual scaling
- Auto scaling
  - Follows write workload
  - Say input load has increased
  - Increase the degree of parallelism
- Manual scaling
  - API call

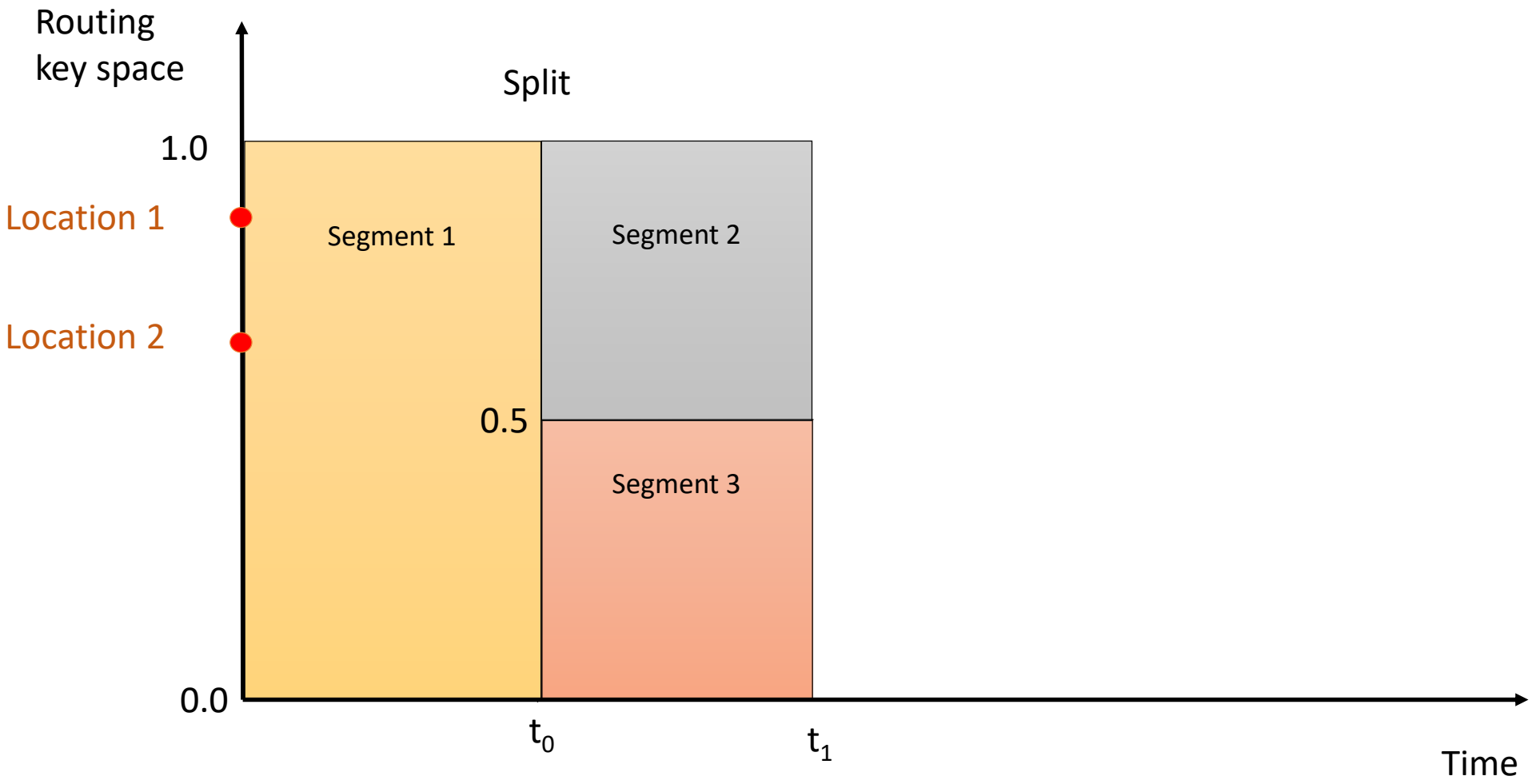


- Seal current segment
- Create new ones

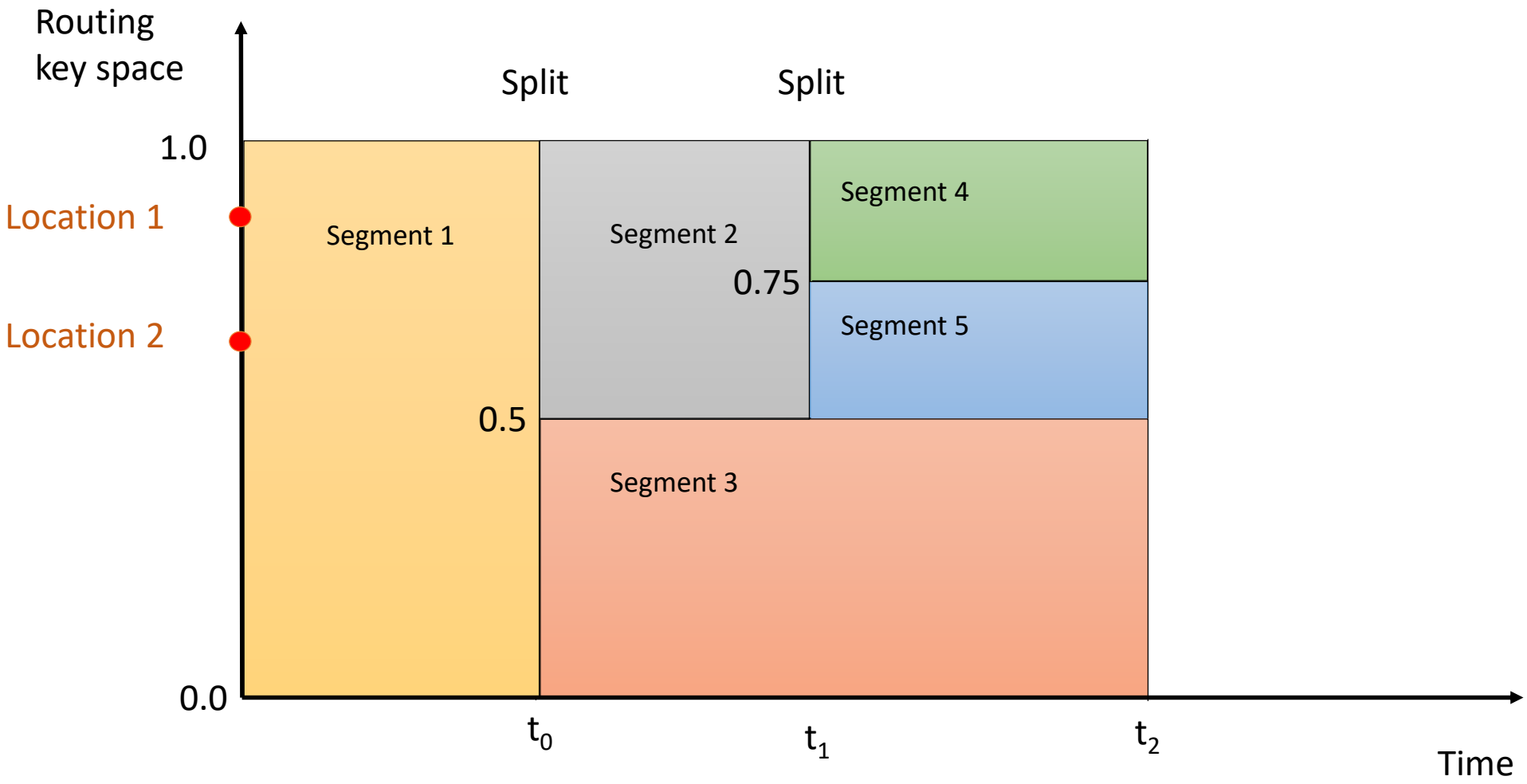
2


Routing
key space
1.0
Segment 1
0.0
t_0
Time

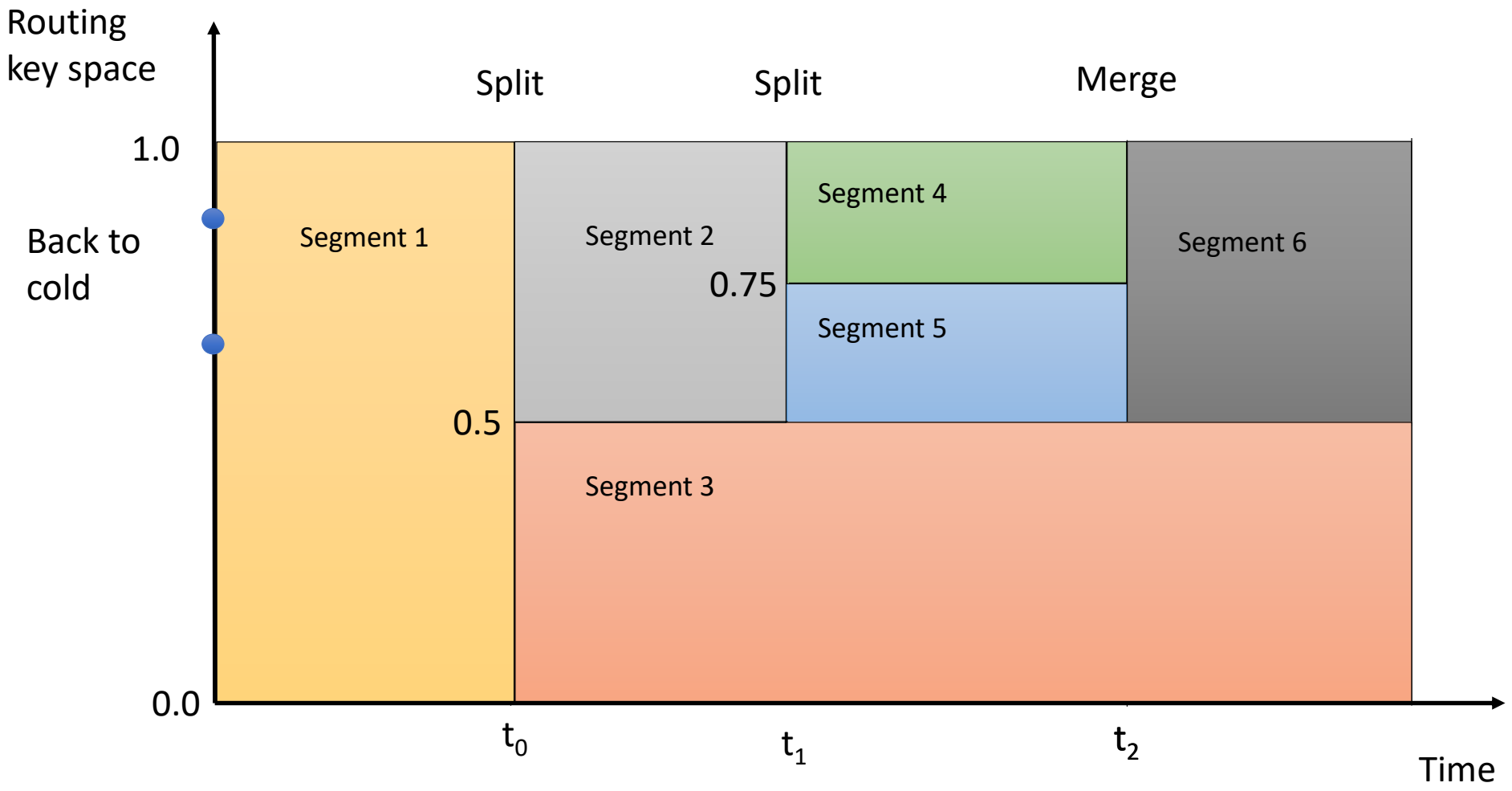

Routing
key space
Split
1.0
Segment 1
Segment 2
0.5
Segment 3
0.0
t_0
t_1
Time



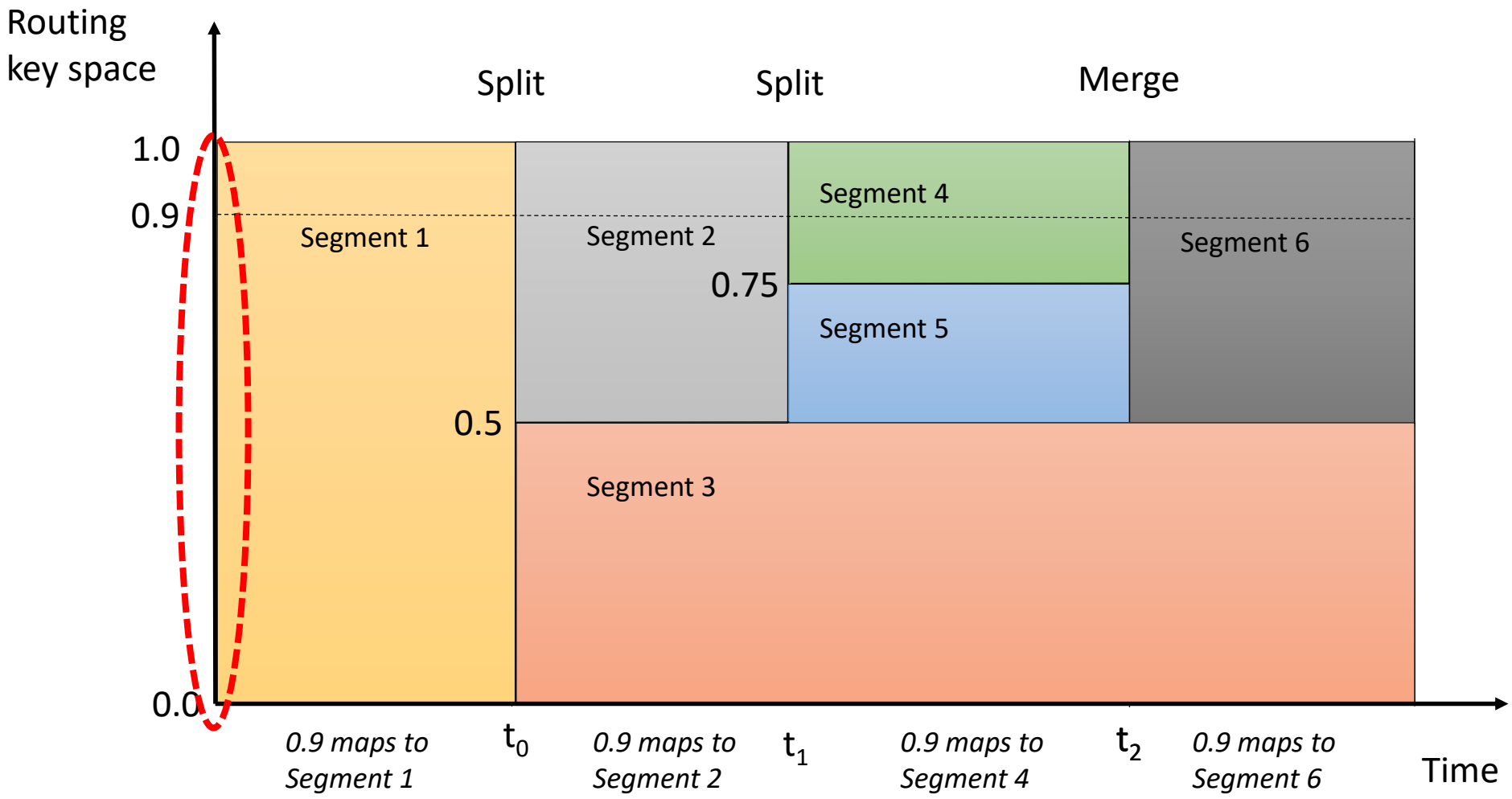
- Keys are coordinates in a geo application
- *E.g.*, taxi rides



- Keys are coordinates in a geo application
- *E.g.*, taxi rides

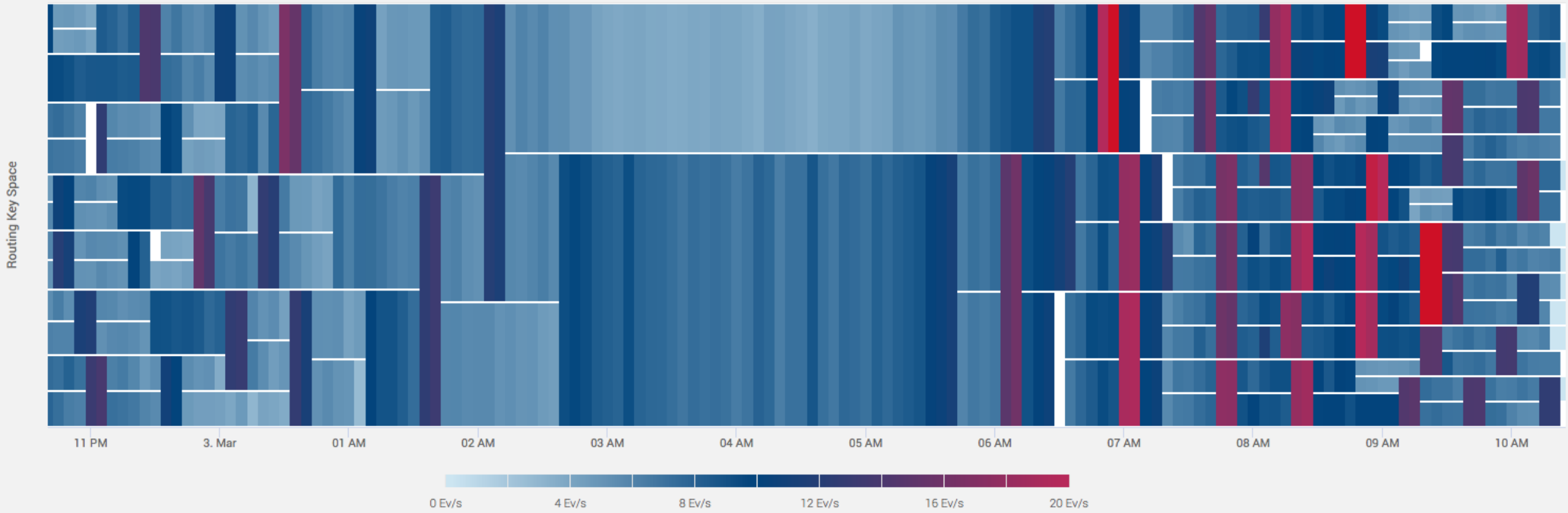

Routing
key space
Split
Split
Merge
1.0
Back to
cold
Segment 1
Segment 2
Segment 4
Segment 6
0.75
Segment 5
0.5
Segment 3
0.0
t_0
t_1
t_2
Time

Key ranges are not statically assigned to segments



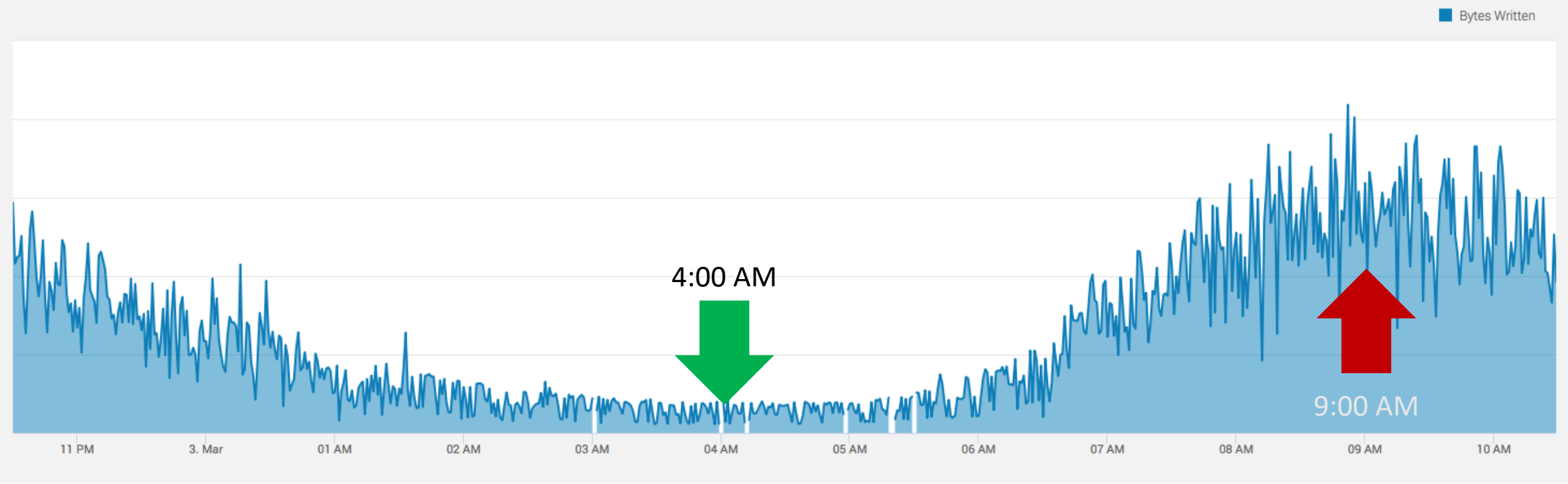
Pravega
Reliable streaming storage

Segment Heat Map



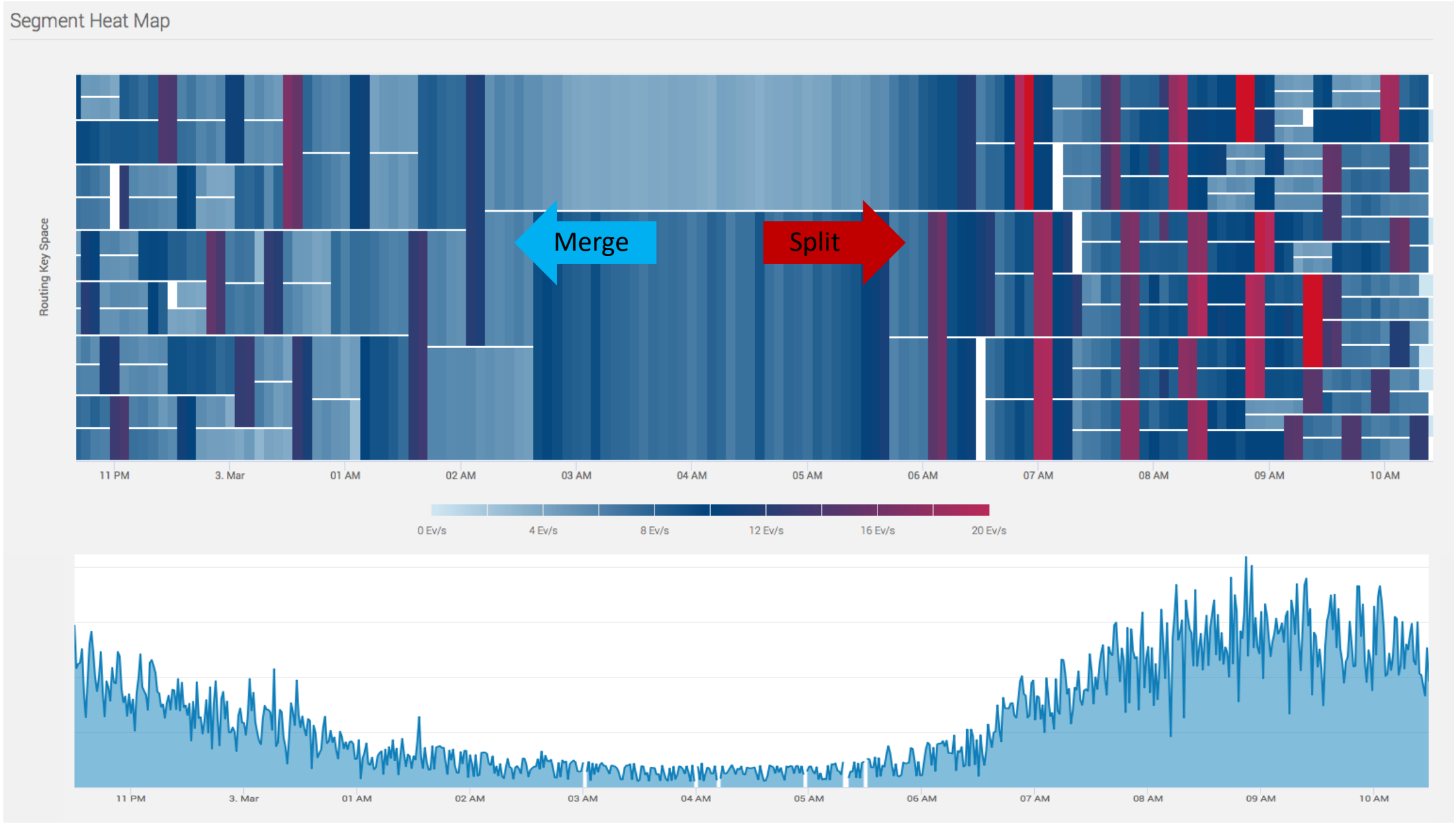
Daily Cycles

Peak rate is 10x higher than lowest rate



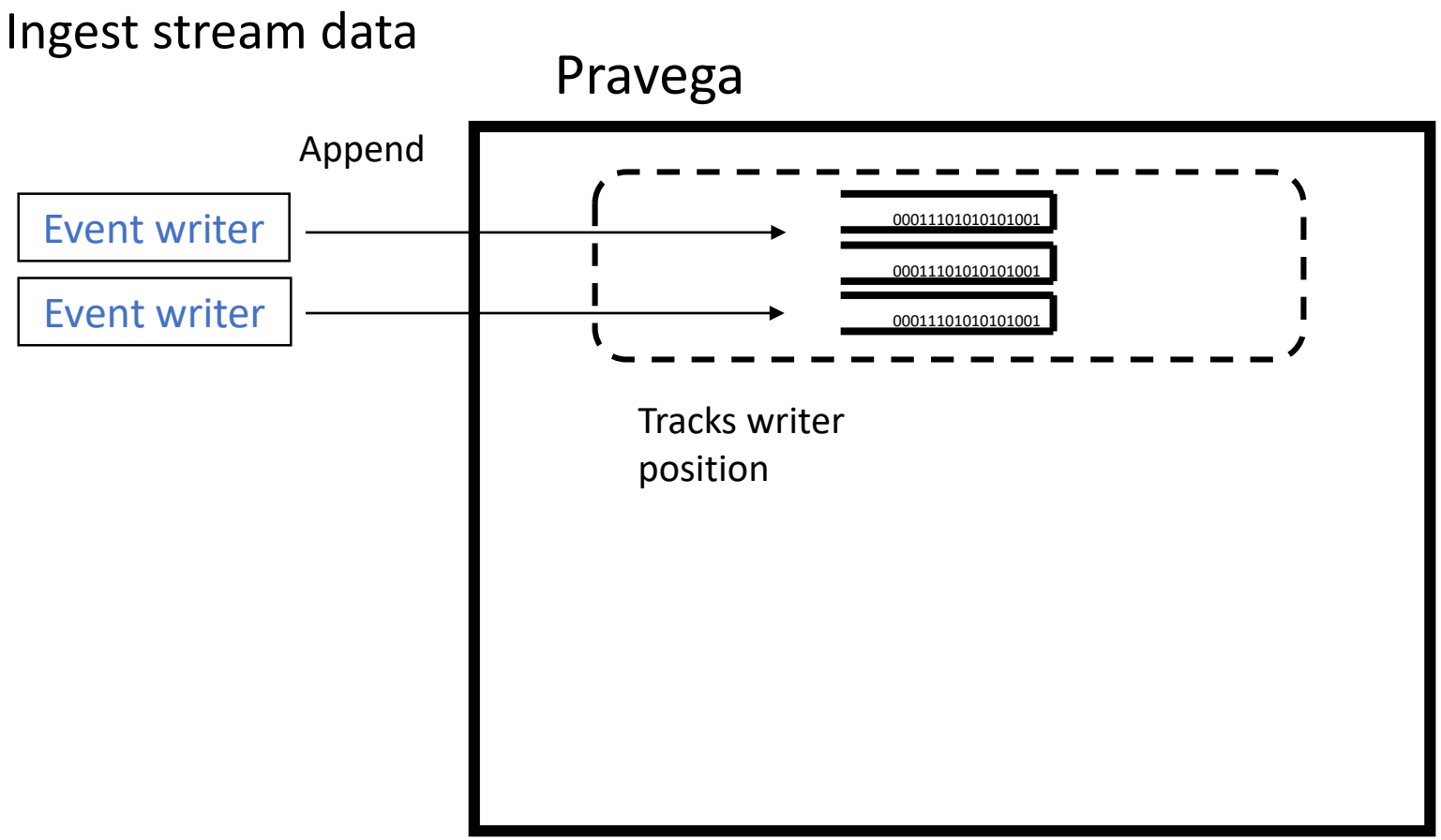
NYC Yellow Taxi Trip Records, March 2015
http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml

Pravega Auto Scaling

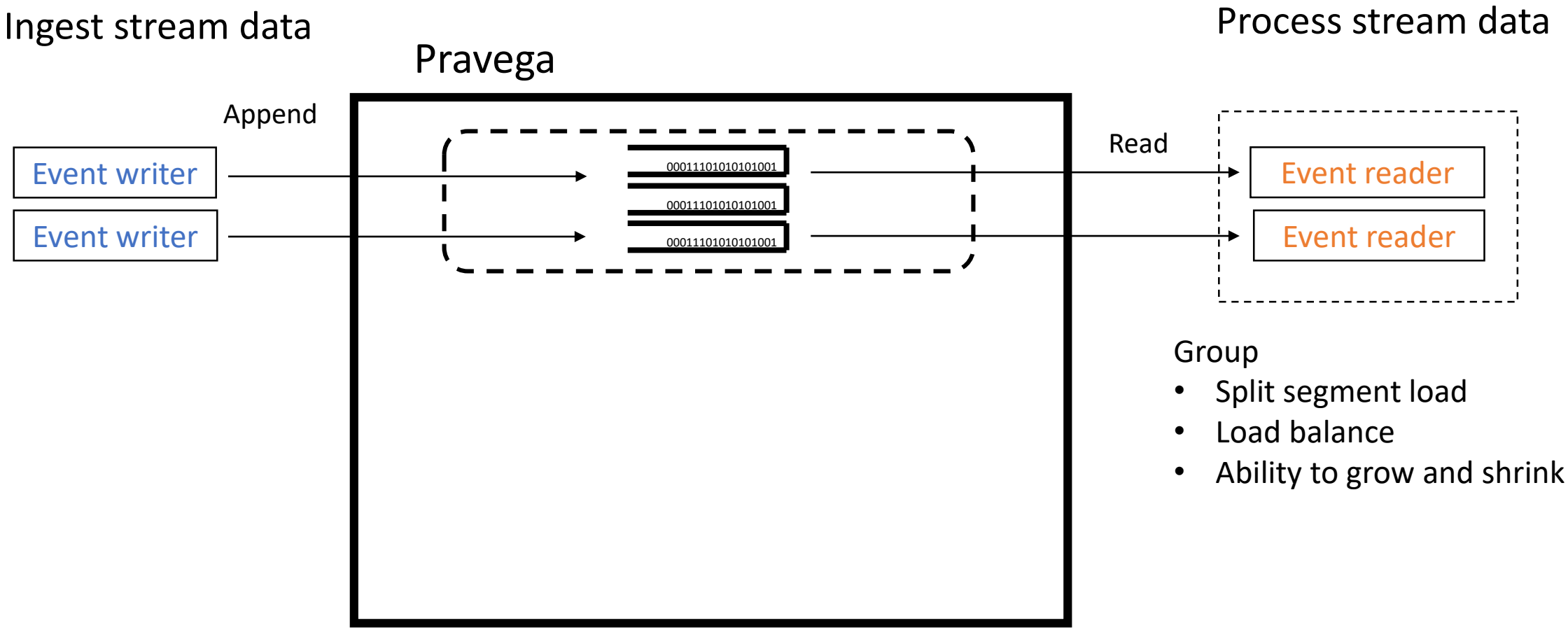


Pravega Architecture

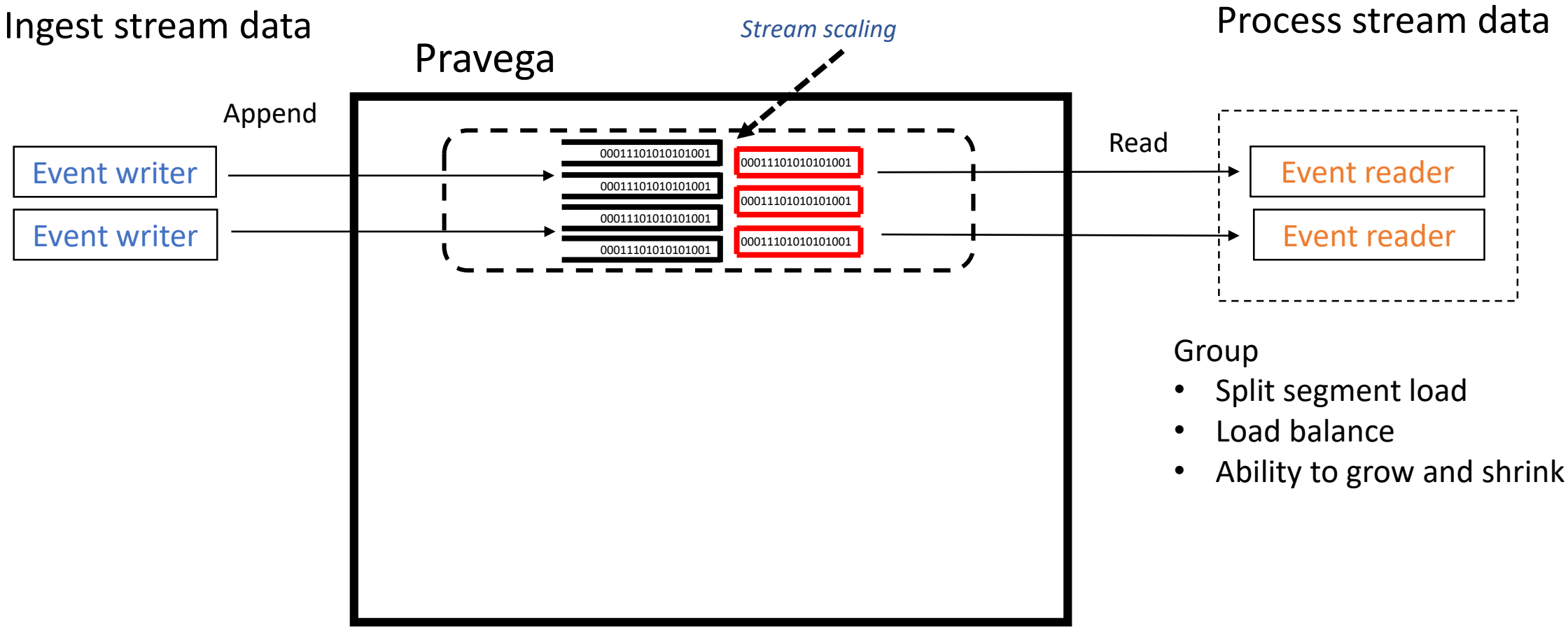
Ingesting to a stream



Reading from a stream



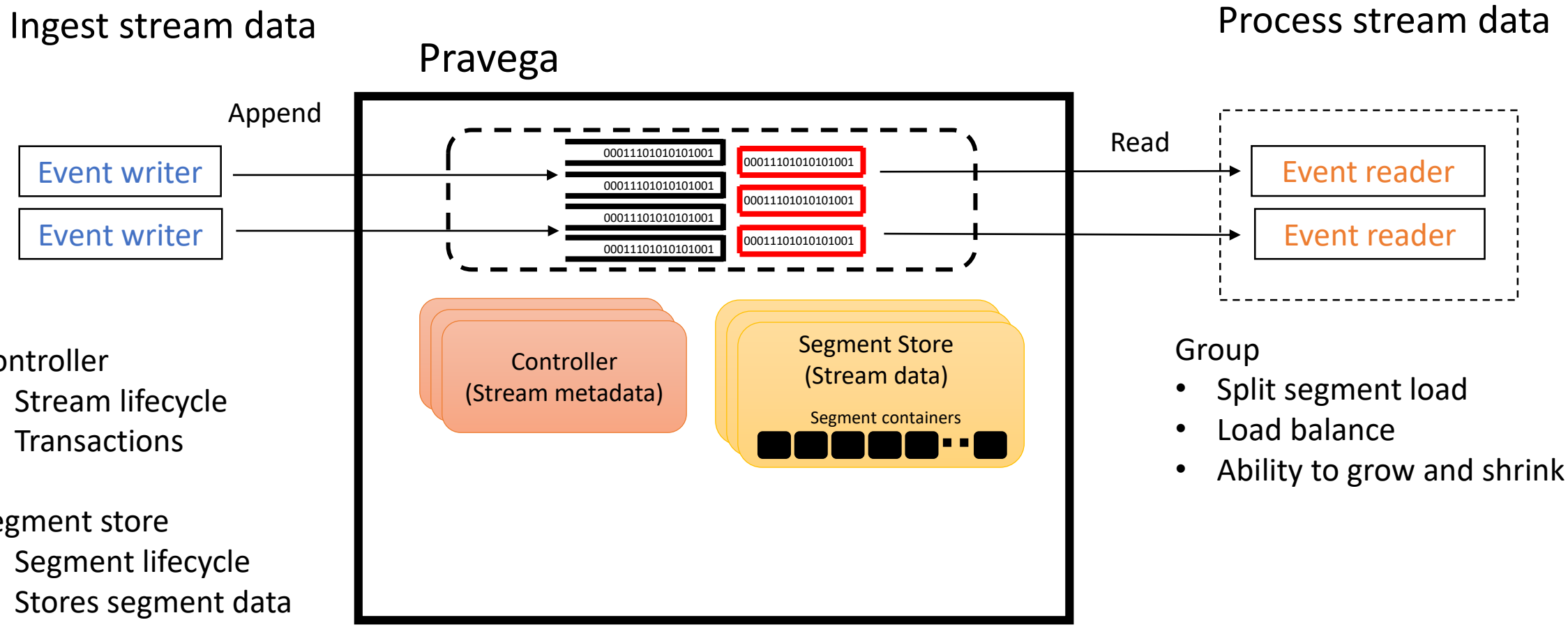
Reading from a stream



Group

- Split segment load
- Load balance
- Ability to grow and shrink

Control and data planes

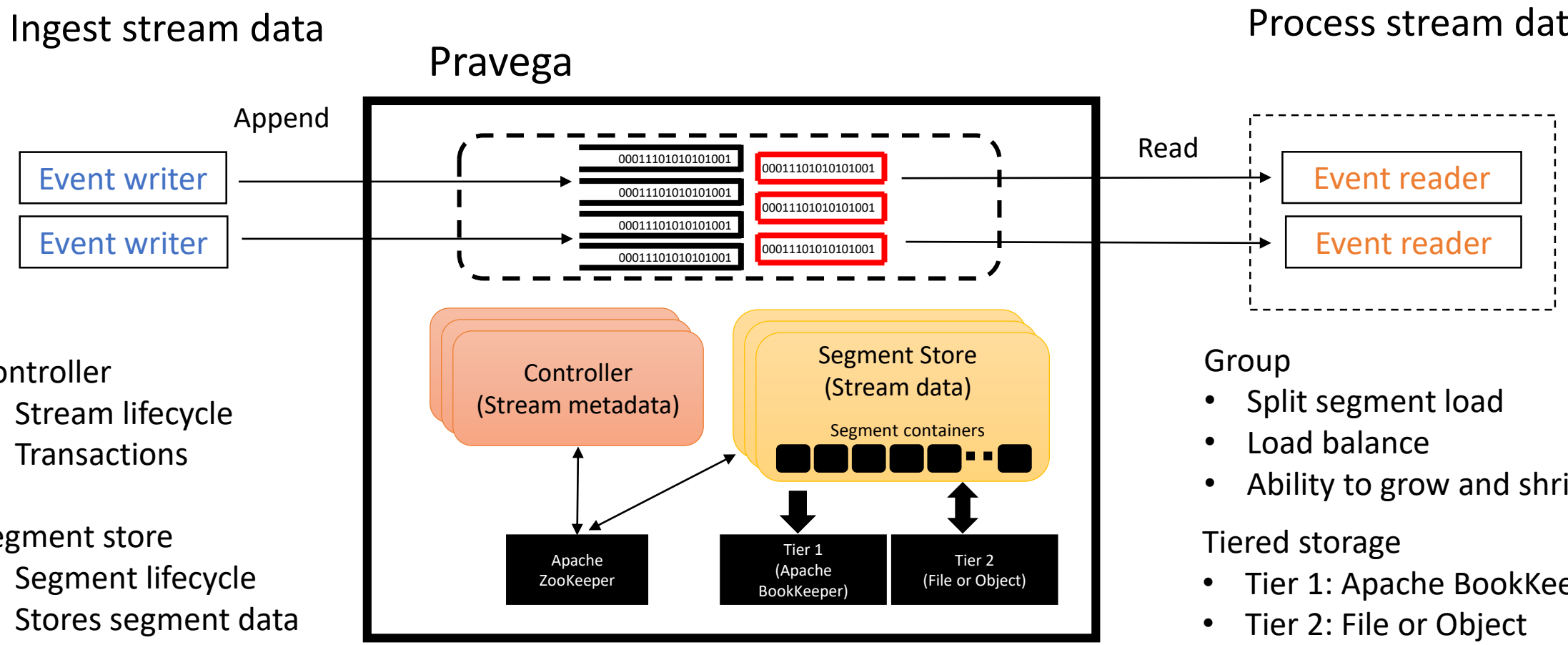


Controller
- Stream lifecycle
- Transactions

Segment store
- Segment lifecycle
- Stores segment data

Group
- Split segment load
- Load balance
- Ability to grow and shrink

Tiered storage



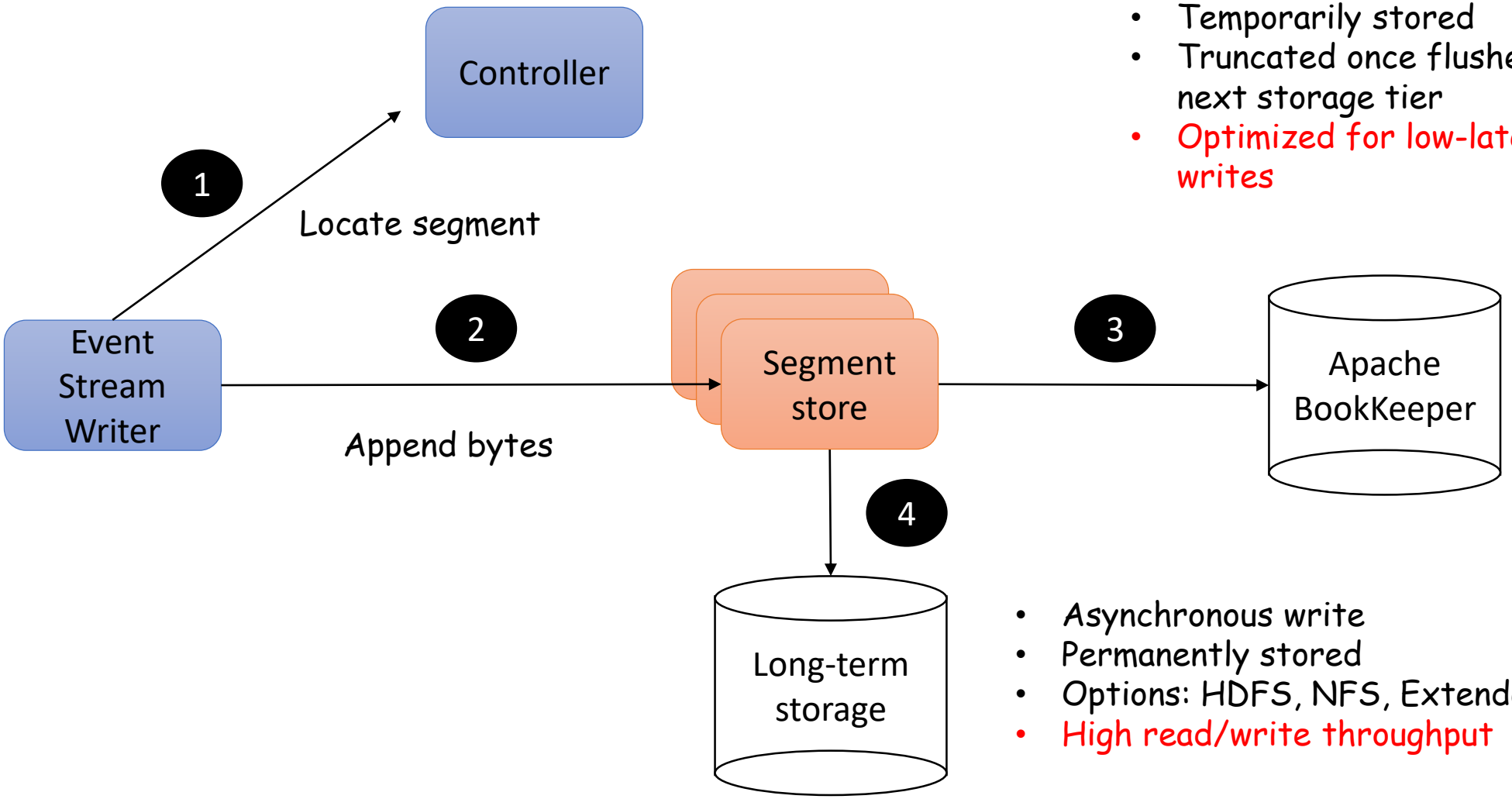
Controller
- Stream lifecycle
- Transactions

Segment store
- Segment lifecycle
- Stores segment data

Group
- Split segment load
- Load balance
- Ability to grow and shrink

Tiered storage
- Tier 1: Apache BookKeeper
- Tier 2: File or Object

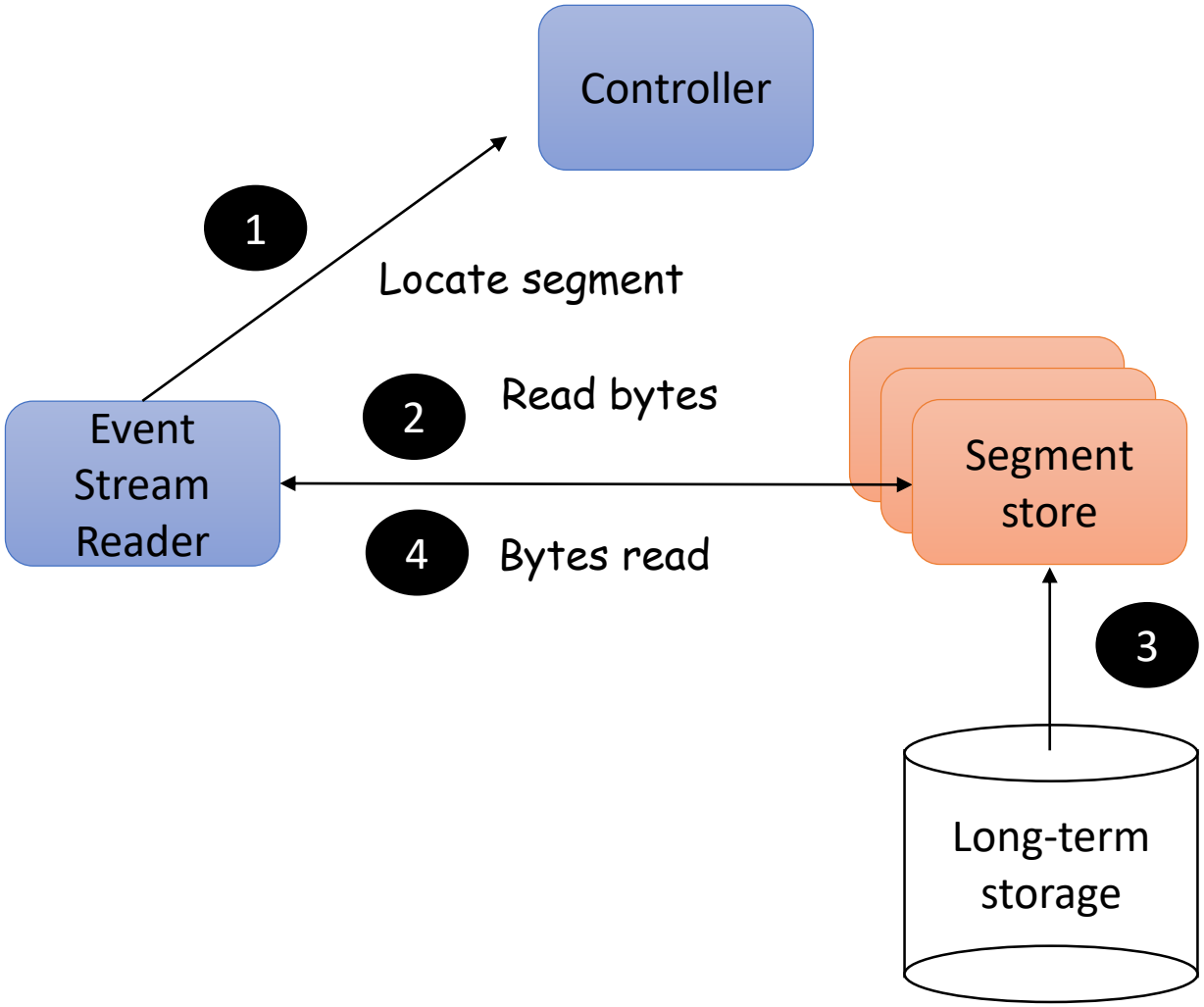
The write path



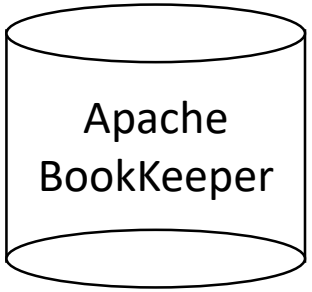
- Synchronous write
- Temporarily stored
- Truncated once flushed to next storage tier
- Optimized for low-latency writes

- Asynchronous write
- Permanently stored
- Options: HDFS, NFS, Extended S3
- High read/write throughput

The read path



- Used for recovery alone
- Not used to serve reads

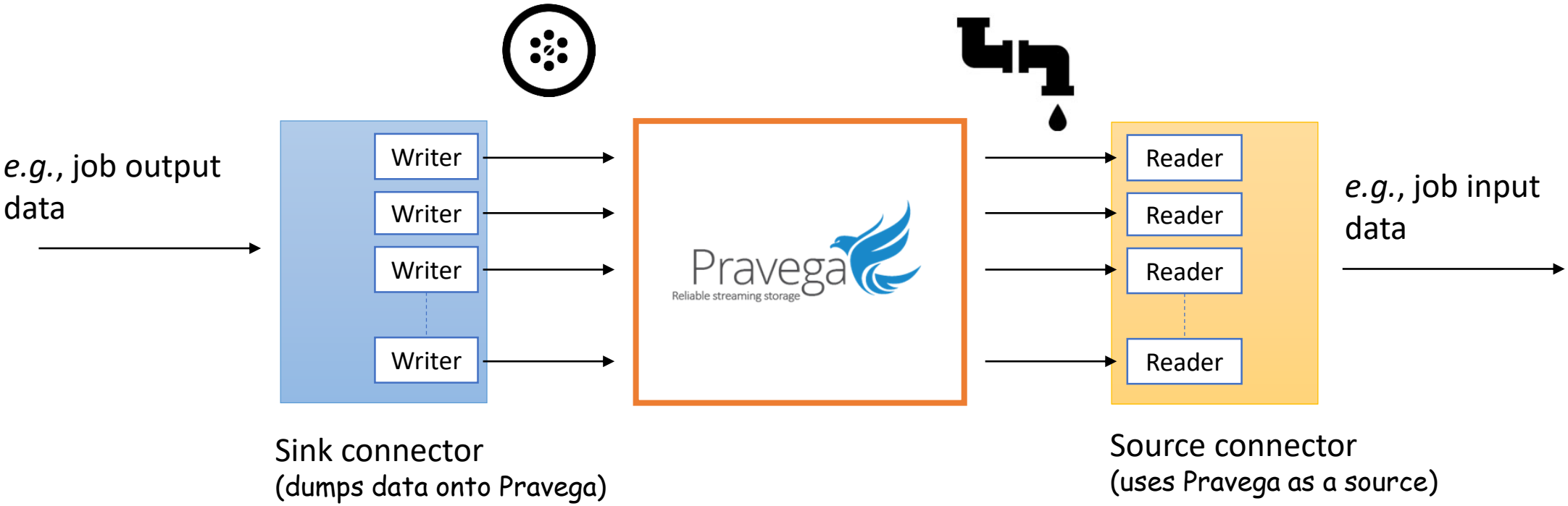


- Bytes read from memory
- If not present, pull data from Tier 2

Pravega
Reliable streaming storage

Connecting to Stream Processors

Connectors



https://github.com/pravega/flink-connectors

Existing connectors

- Apache Flink
- Apache Hadoop
- Logstash plugins
- Alpakka connector
- More to come…

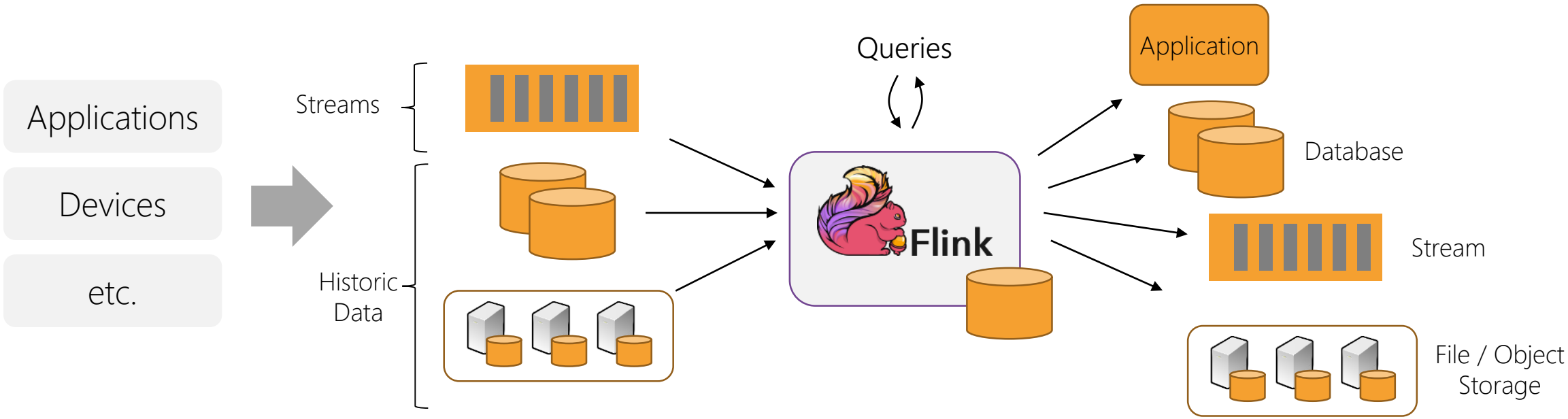
Existing connectors

- **Apache Flink**
- Apache Hadoop
- Logstash plugins
- Alpakka connector
- More to come…

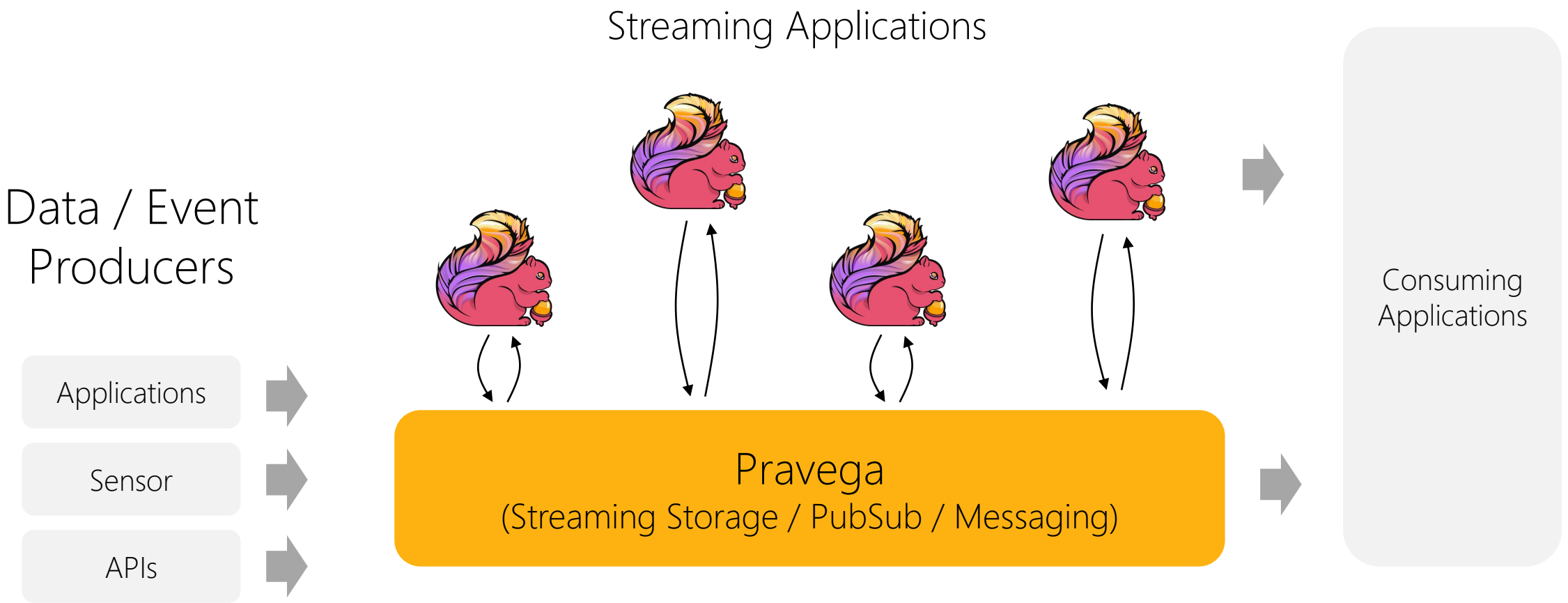
Apache Flink in a nutshell

Stateful computations over streams

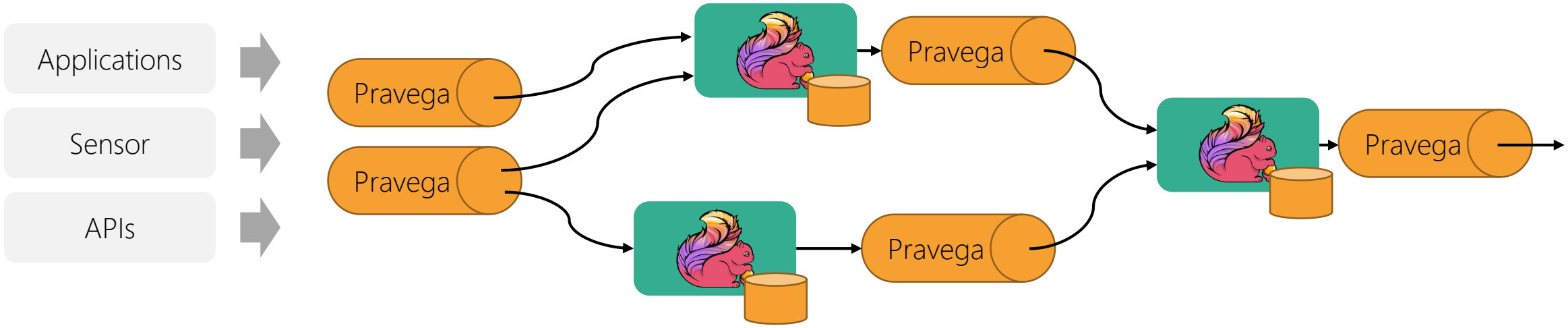
real-time and historic
scalable, fault tolerant, fast, in-memory,
event time, large state, exactly-once

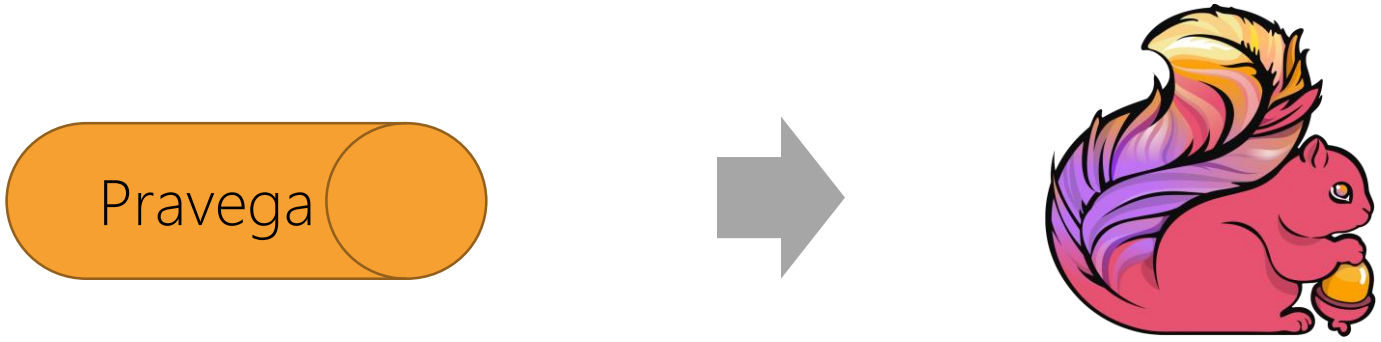


Compute with Apache Flink



Streaming Pipelines

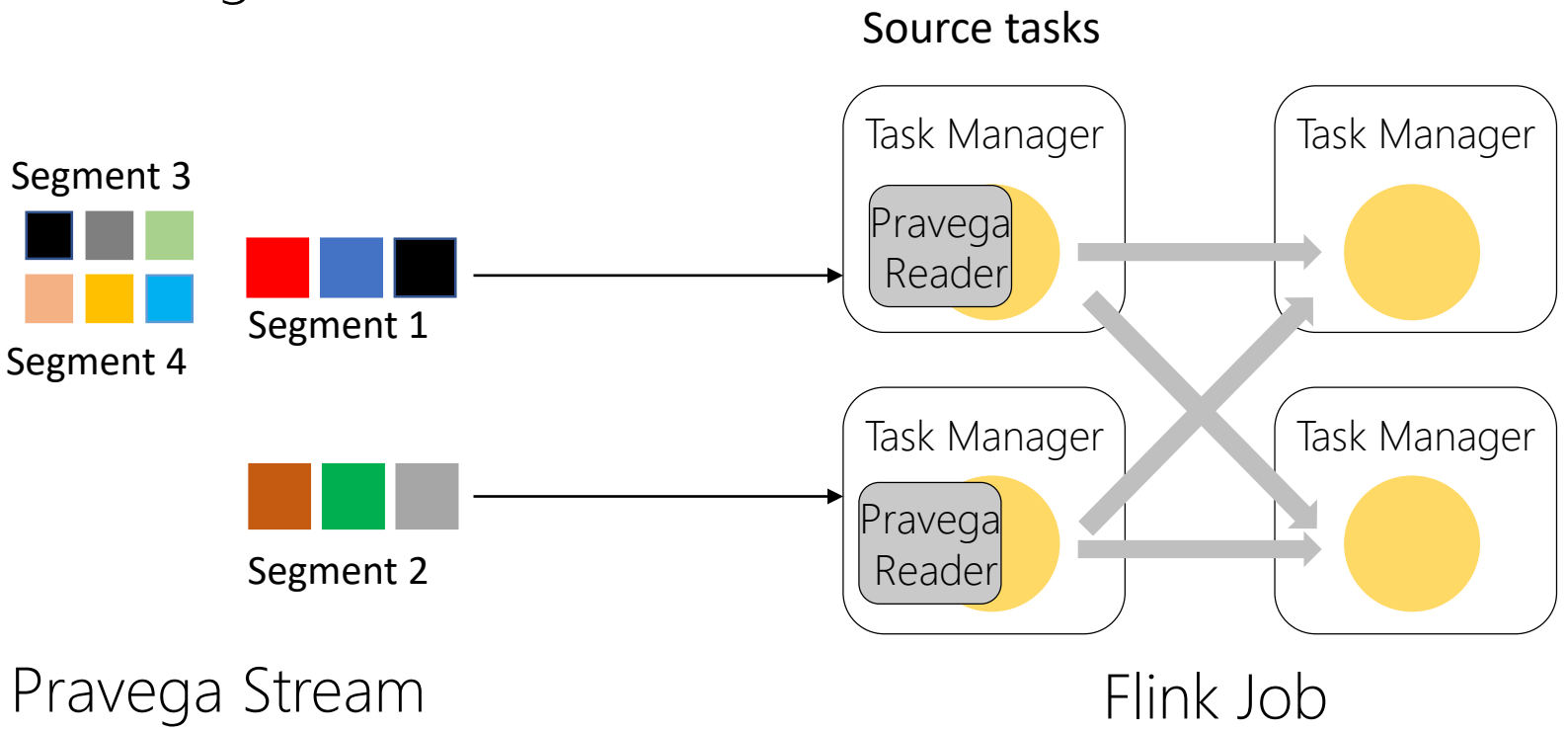




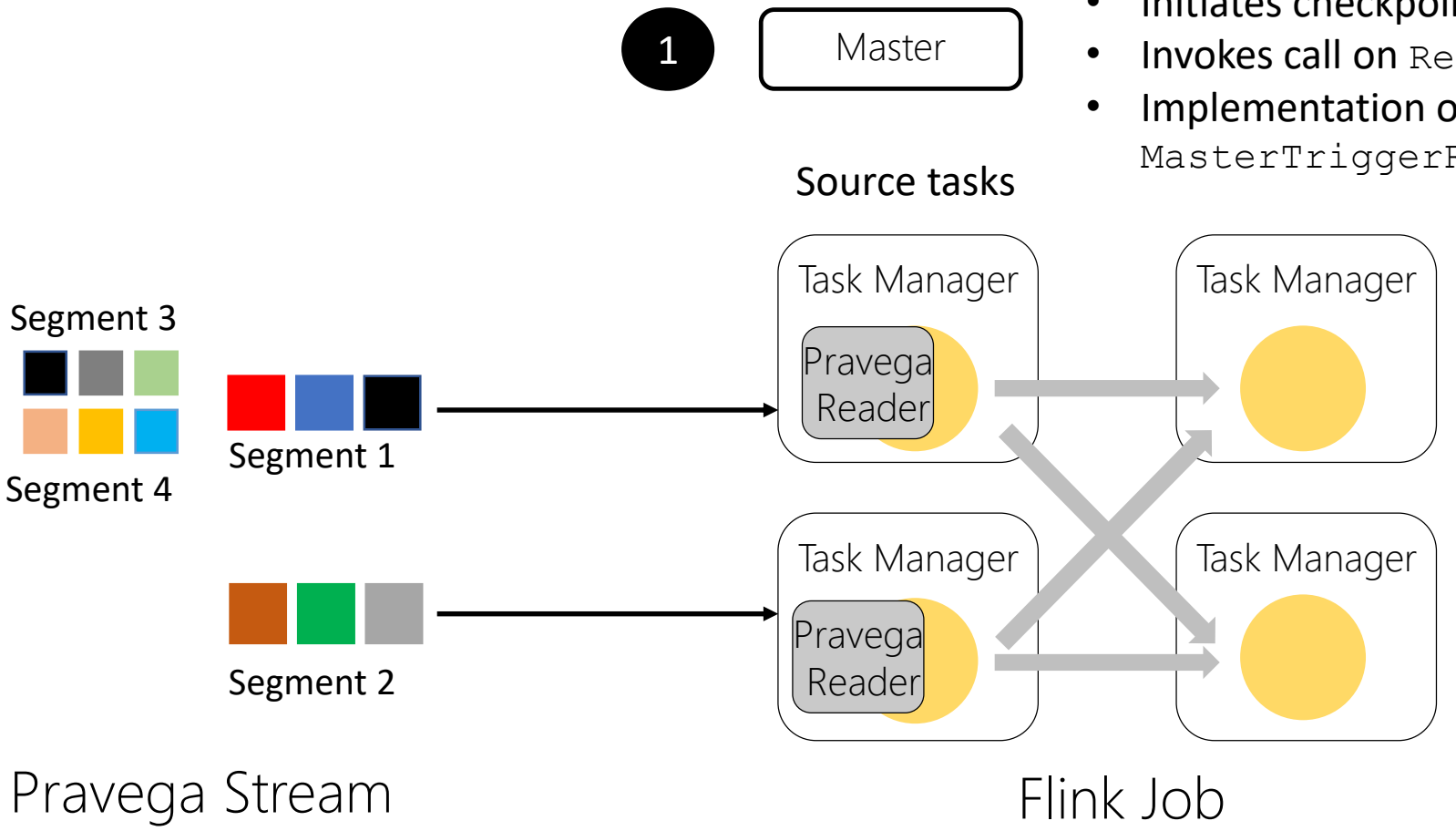
Flink reading from Pravega

Reading via Reader Group

- Reader group automatically assigns and re-balances segments
  - Hides complexity from app
  - Stream scaling



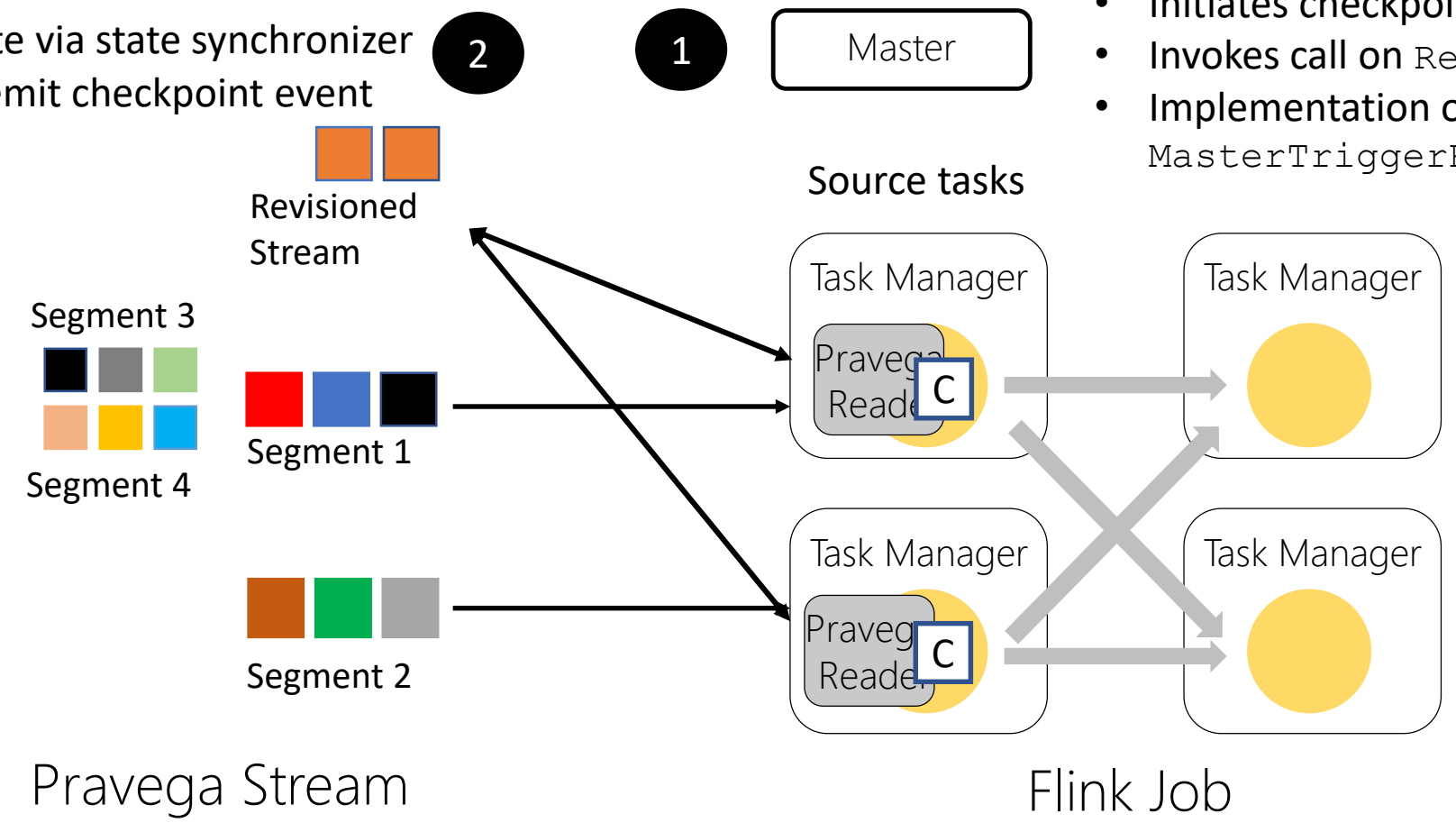
Upon a Flink checkpoint



- Initiates checkpoint
- Invokes call on `ReaderGroup` API
- Implementation of `MasterTriggerRestoreHook`

Upon a Flink checkpoint

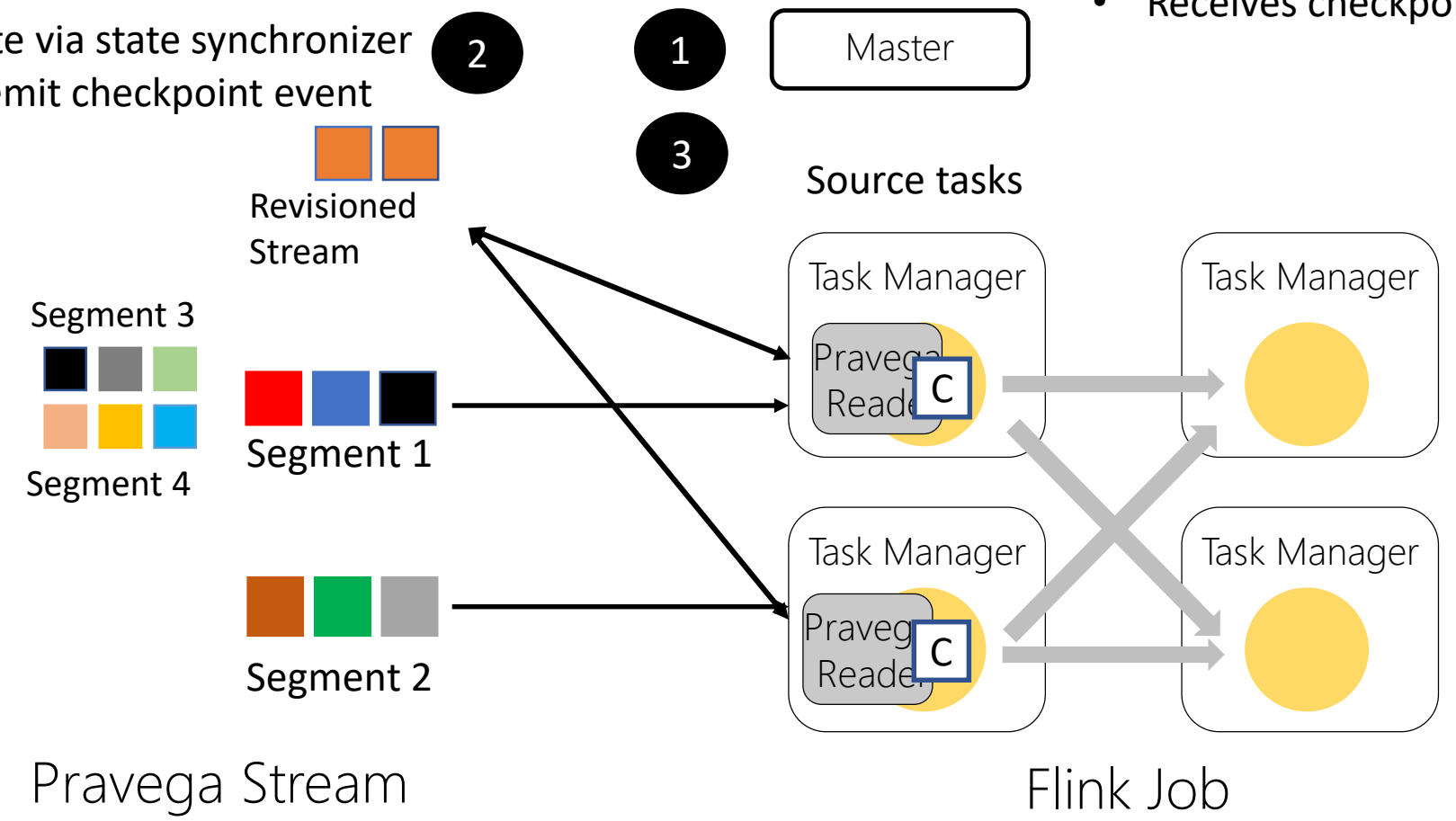
- Coordinate via state synchronizer
- Readers emit checkpoint event



- Initiates checkpoint
- Invokes call on `ReaderGroup` API
- Implementation of `MasterTriggerRestoreHook`

Upon a Flink checkpoint

- Coordinate via state synchronizer
- Readers emit checkpoint event



- Receives checkpoint

Creating a Pravega source

```
// create the Pravega source to read a stream of text
FlinkPravegaReader<String> source = FlinkPravegaReader.<String>builder()
                .withPravegaConfig(pravegaConfig)
                .forStream(stream)
                .withDeserializationSchema(PravegaSerialization.deserializationFor(String.class))
                .build();
```

https://github.com/pravega/pravega-samples

Creating a Pravega source (… and using it)

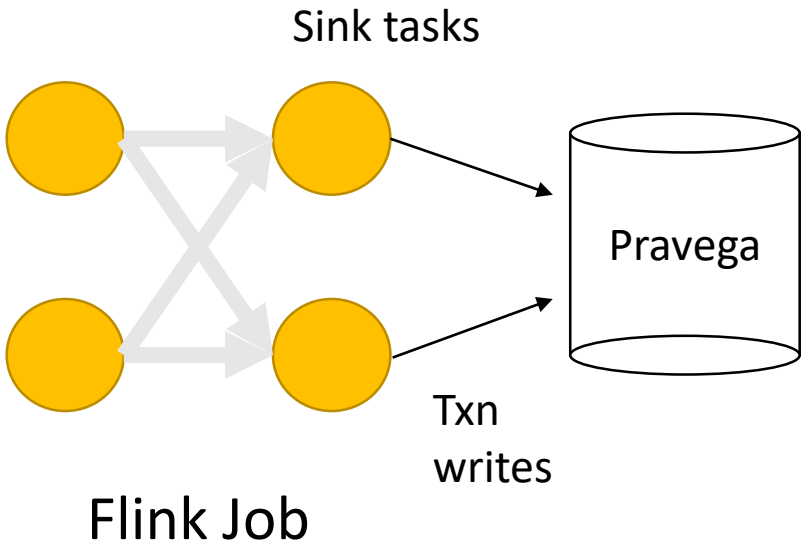
```
// create the Pravega source to read a stream of text
FlinkPravegaReader<String> source = FlinkPravegaReader.<String>builder()
                .withPravegaConfig(pravegaConfig)
                .forStream(stream)
                .withDeserializationSchema(PravegaSerialization.deserializationFor(String.class))
                .build();
```

```
// count each word over a 10 second time period
DataStream<WordCount> dataStream = env.addSource(source).name("Pravega Stream")
                .flatMap(new WordCountReader.Splitter())
                .keyBy("word")
                .timeWindow(Time.seconds(10))
                .sum("count");
```

https://github.com/pravega/pravega-samples

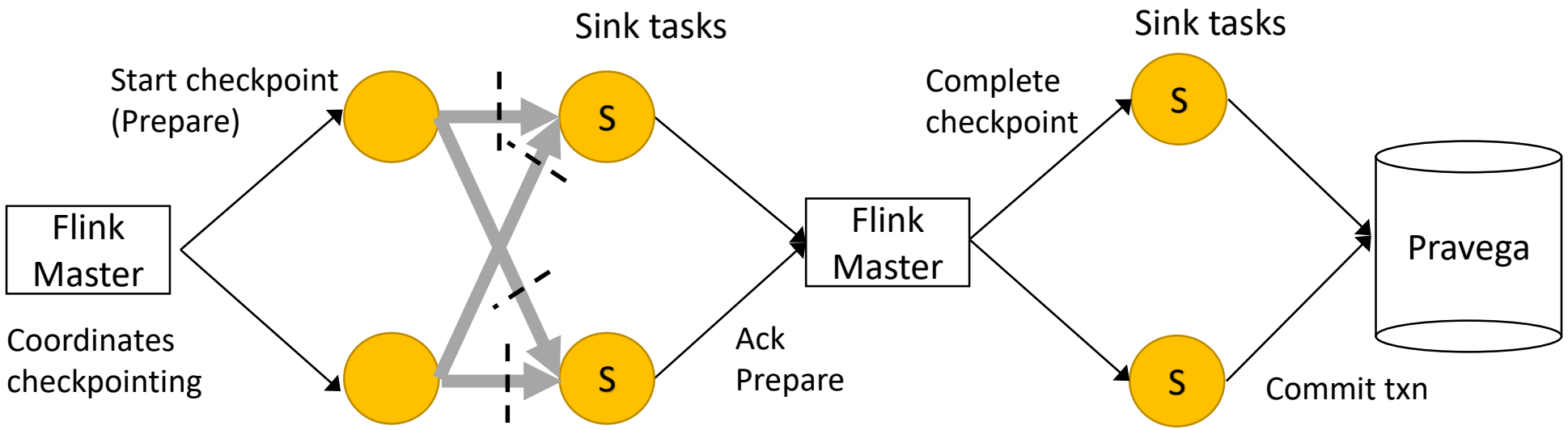
Flink writing to Pravega

Exactly-once with Transactions



- Transactional writes for job output
- Executes a 2PC to commit results
- Option to not use transactions
  - At-least-once semantics

Exactly-once with Transactions



2-Phase commit protocol

Creating a Pravega sink

```
// create the Pravega sink to write a stream of text
FlinkPravegaWriter<String> writer = FlinkPravegaWriter.<String>builder()
                .withPravegaConfig(pravegaConfig)
                .forStream(stream)
                .withEventRouter(new EventRouter())
                .withWriterMode(PravegaWriterMode.EXACTLY_ONCE)
                .withSerializationSchema(PravegaSerialization.serializationFor(String.class))
                .build();
dataStream.addSink(writer).name("Pravega Stream");
```

https://github.com/pravega/pravega-samples

Pravega on Kubernetes

Kubernetes operators

- Operator
  - Custom controller for managing the lifecycle of an application
- Automation
  - Deployment
  - Configuration
  - Scaling
  - Upgrades
  - Monitoring
  - *etc.*

Kubernetes operators

- Pravega Operator

  https://github.com/pravega/pravega-operator

- BookKeeper Operator

  https://github.com/pravega/bookkeeper-operator

- ZooKeeper Operator

  https://github.com/pravega/zookeeper-operator

Wrap up

Conclusion

- Multitude of sources continuously generating data
- Pravega: storage for data streams
  - Unbounded
  - Elastic
  - Consistent
- Connects to streams processors
  - *E.g.*, Apache Flink
  - Exactly-once end to end
- Open source
  - Apache License v2
  - Hosted on github
  - Looking for a home for incubation

Getting started

- Check the web site:
  http://pravega.io
- Check the organization and repository:
  https://github.com/pravega/pravega
- Run Pravega standalone
  ```
  ./gradlew :standalone:startStandalone
  ```
- Try some samples
  https://github.com/pravega/pravega-samples
- Try on Kubernetes
  https://github.com/pravega/pravega-operator

- Feel free to provide feedback and contribute

Questions?

http://pravega.io ← Pravega's Web site

http://github.com/pravega/pravega ← Pravega's code

http://flink.apache.org ← Apache Flink's site

https://github.com/pravega/flink-connectors ← Pravega-Flink connector

{ https://github.com/pravega/pravega-operator
https://github.com/pravega/bookkeeper-operator
https://github.com/pravega/zookeeper-operator } Operators

E-mail: fpj@pravega.io
Twitter: @fpjunqueira