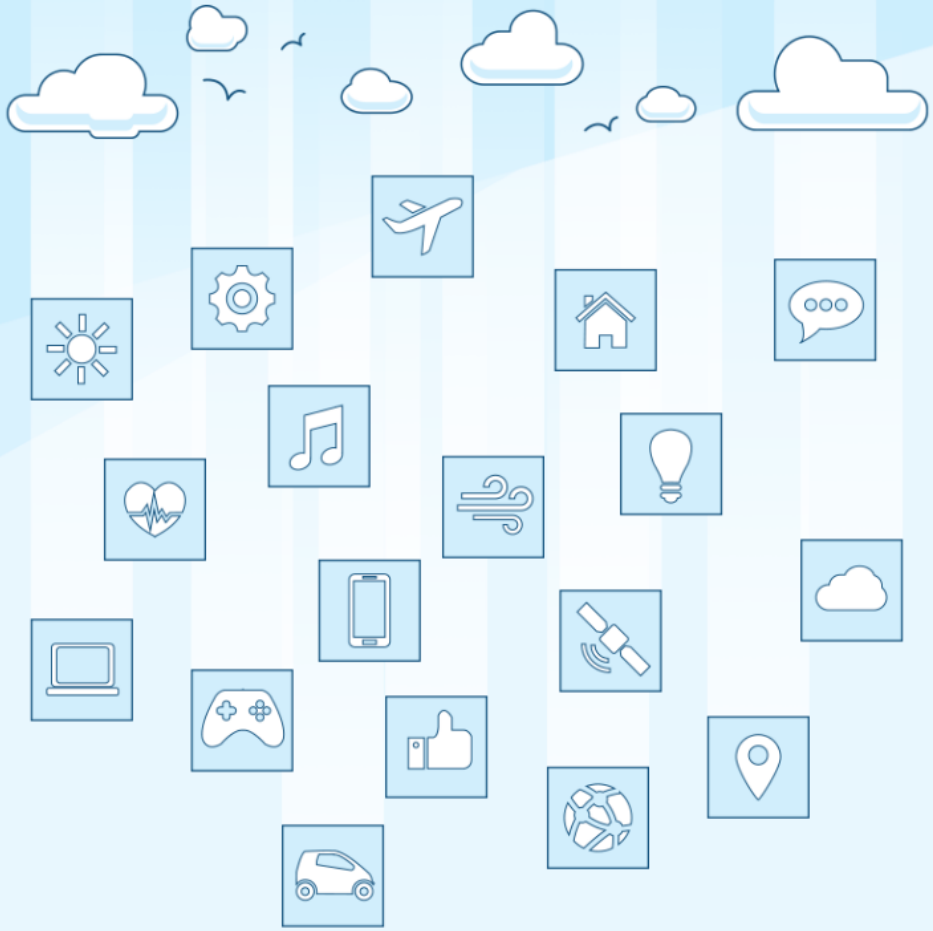


Elastic Data Streams in Pravega for Serverless Computing

Raúl Gracia, Pravega by DellEMC

WOSCx, 2022

Serverless Computing & Streams: A Great Match

- Serverless frameworks allow us to **trigger functions in response to events:**
  - Popular approach to deliver a **reactive programming**.
  - Great deal of **abstraction** from **underlying infrastructure**.
  - **Simplified** programming model for users.

- Data Streams are a **continuous source of events**.
  - Ideal **source of input** for Serverless computing frameworks.
  - **First-class citizen** abstraction for many use-cases (i.e., like object or file).
  - Streaming storage systems need to deal with several challenges (write/read guarantees, parallelism, etc.).

- Several examples of **messaging/streaming systems as a source** for serverless frameworks:
  - Kafka connector for OpenWhisk:
    - https://github.com/apache/openwhisk-package-kafka
  - Kafka event source for AWS Lambda:
    - https://docs.aws.amazon.com/lambda/latest/dg/with-kafka.html

Real Streaming Use-cases

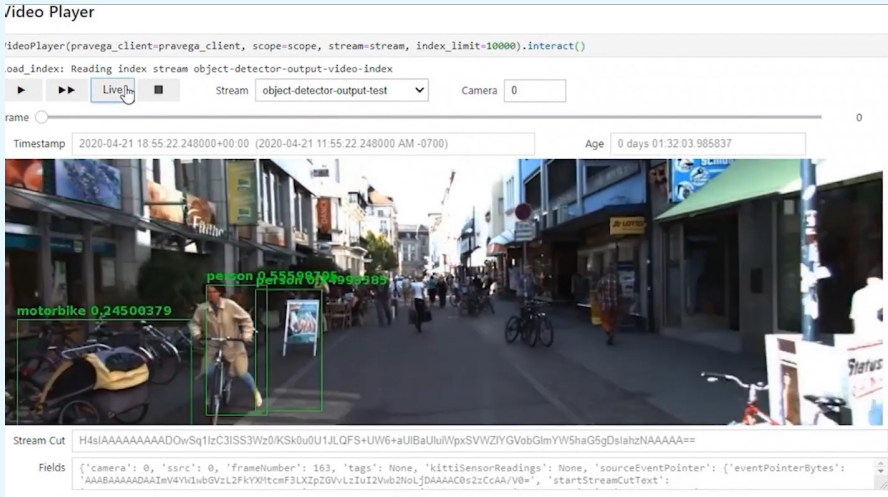
- Many real **streaming use cases** can benefit from the **Serverless computing:**
  - Dell is delivering Knative service on APEX Private Cloud.
  - https://infohub.delltechnologies.com/p/serverless-workload-and-apex-private-cloud

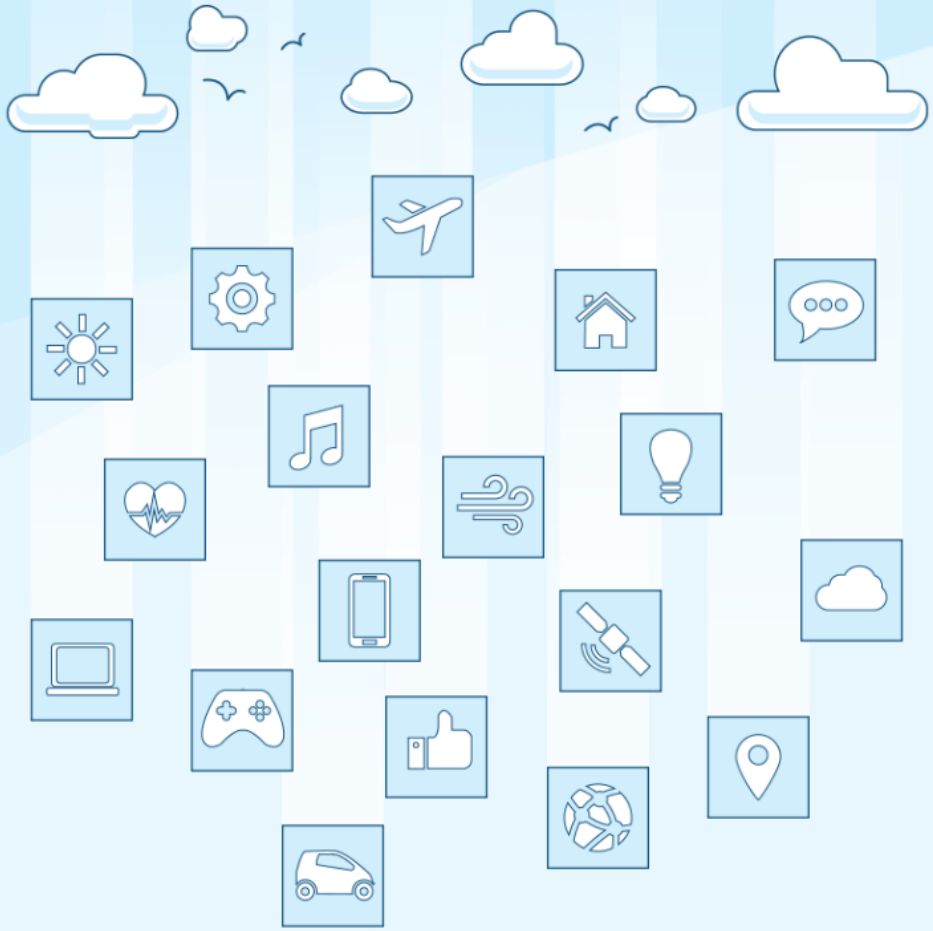
- **Drone images**:
  - Analyze cattle health.
  - Inspect airplanes between flights.
  - Correctness of building construction process.

- **Video analytics**:
  - Storage of surveillance cameras.
  - Real-time identification of objects.

- **Factory sensors**:
  - Anomaly detection in manufacturing.
  - …

Dell Streaming Data Platform
(Streaming storage + analytics product by Dell)





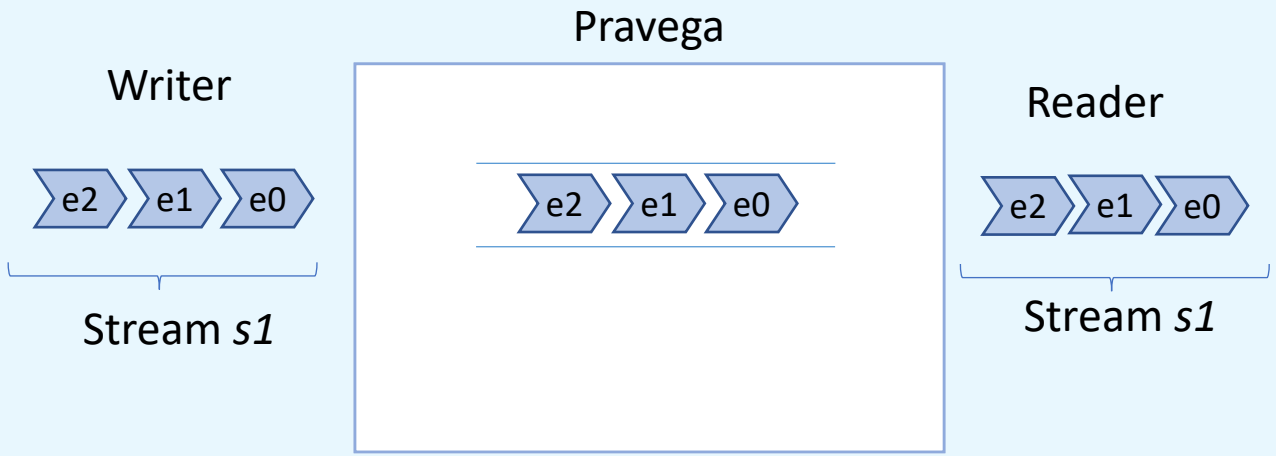
Pravega: A Storage System for Unbounded Data Streams

DELL EMC

http://pravega.io

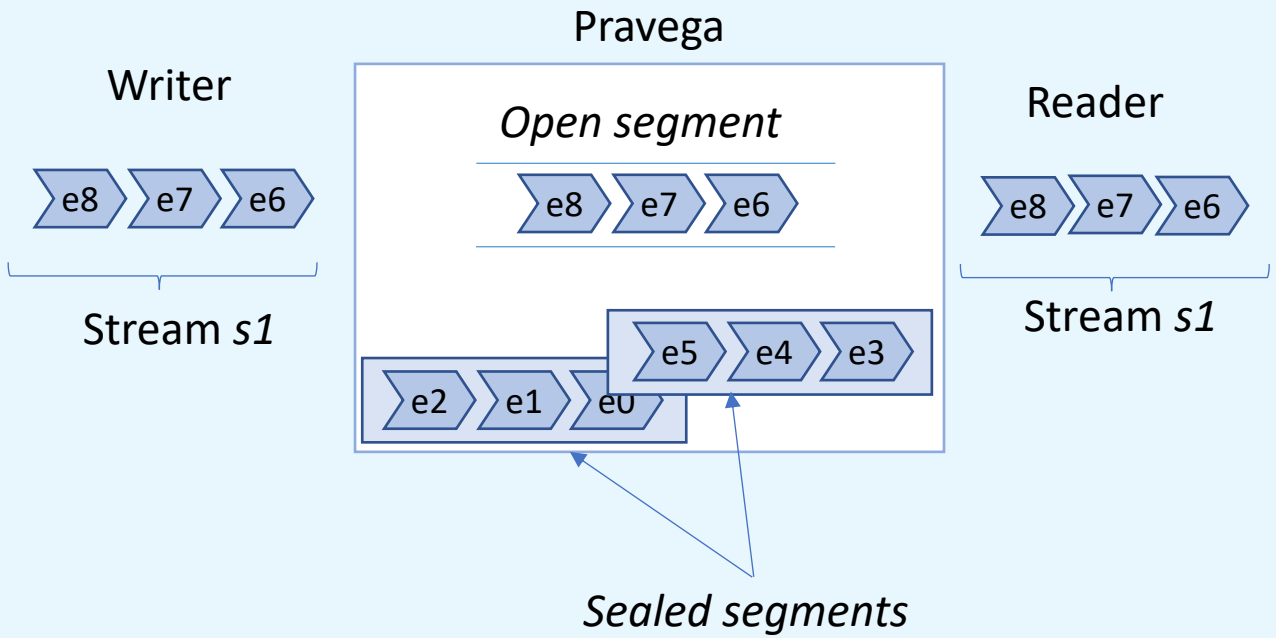
Pravega Concepts I: Streams & Clients

- Pravega is an **open-source storage system** to store/serve **unbounded data streams**.
- **Stream**: Unbounded sequence of bytes.
  - Append-only abstraction (but can be truncated).
- *Clients*: Operate on Streams.
  - **Writer**: `writer.writeEvent(message)`
  - **Reader**: `reader.readNextEvent(timeout)`



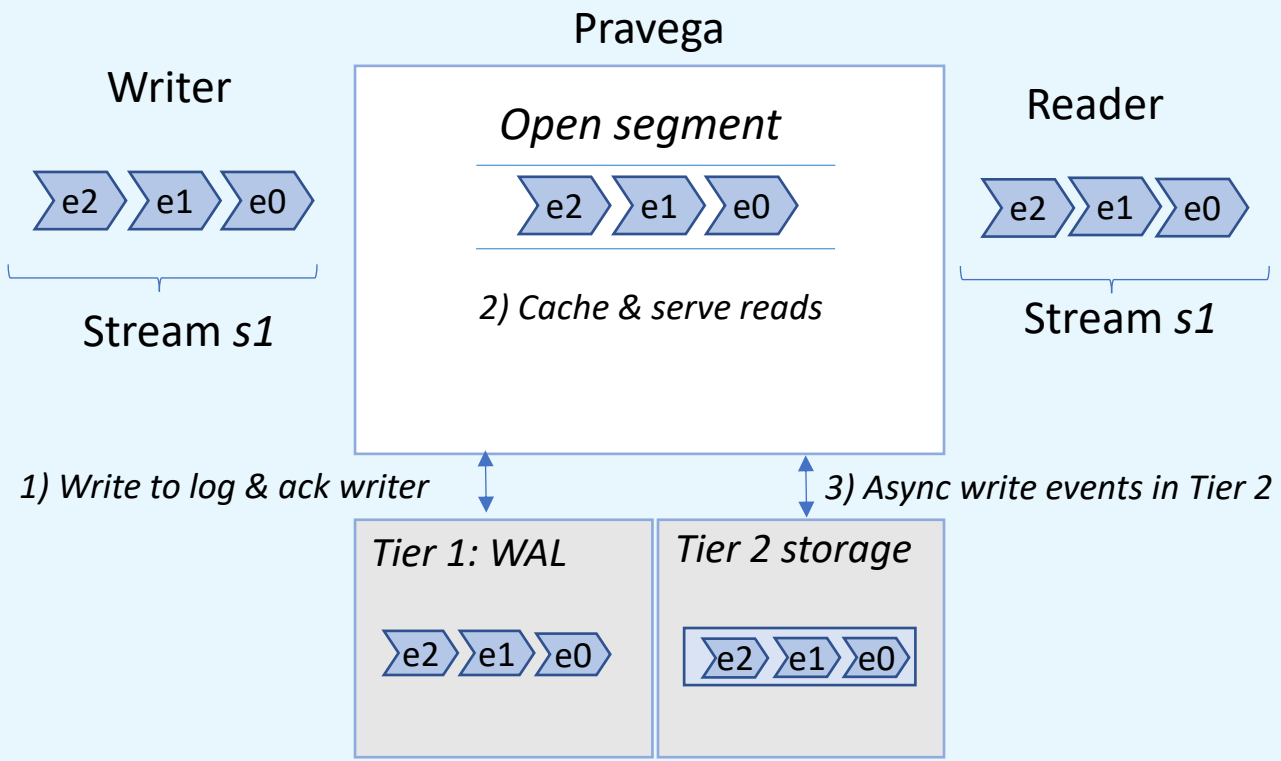
Pravega Concepts II: Stream Segments

- Pravega splits Streams into **segments**:
  - Basic *unit of storage* for Pravega.
- A Stream can be seen as a *sequence of segments*.
- State of segments:
  - *Open* segment: Events are being appended.
  - *Sealed* segment: Read-only.



Pravega Tiered Storage

- Open segments are **durably written** to Tier 1:
  - Low write-to-read latency (*real time analytics*).
  - Write Ahead Log (e.g., Apache Bookkeeper).
  - WAL is only read to recover from failures.
- Segments are **asynchronously stored** in Tier 2:
  - High throughput (*batch analytics*).
  - Pluggable: HDFS, Amazon S3, DellEMC ECS/Isilon.
- Sweet spot in *latency vs throughput trade-off*.

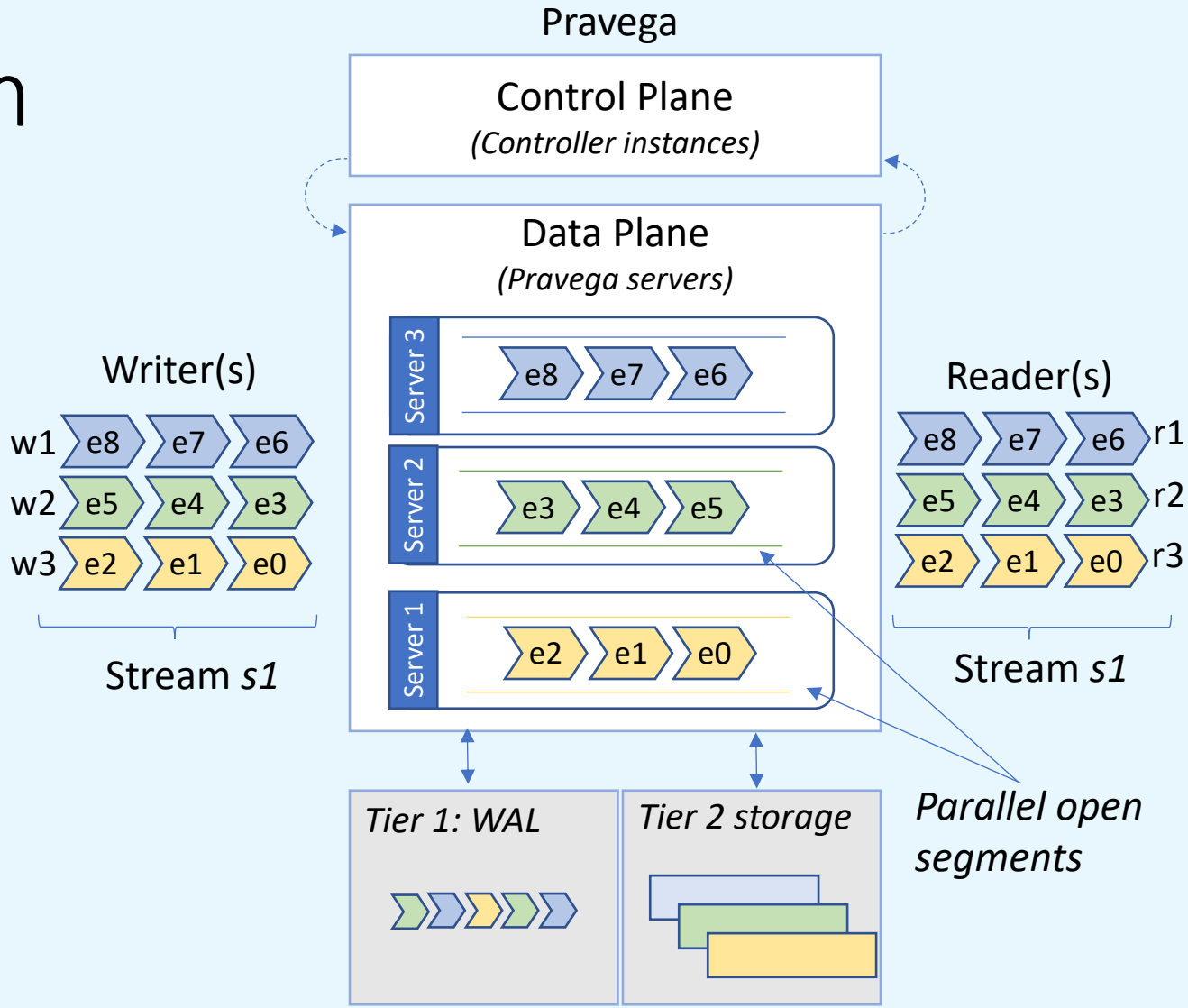


Write/Read Parallelism

- A Stream may have **multiple open segments**.
- **Write guarantees**:
  - *Exactly-once*: No event duplicates (e.g., on reconnections).
  - All events written to a **routing key** will be read in the *same order as they were written*.

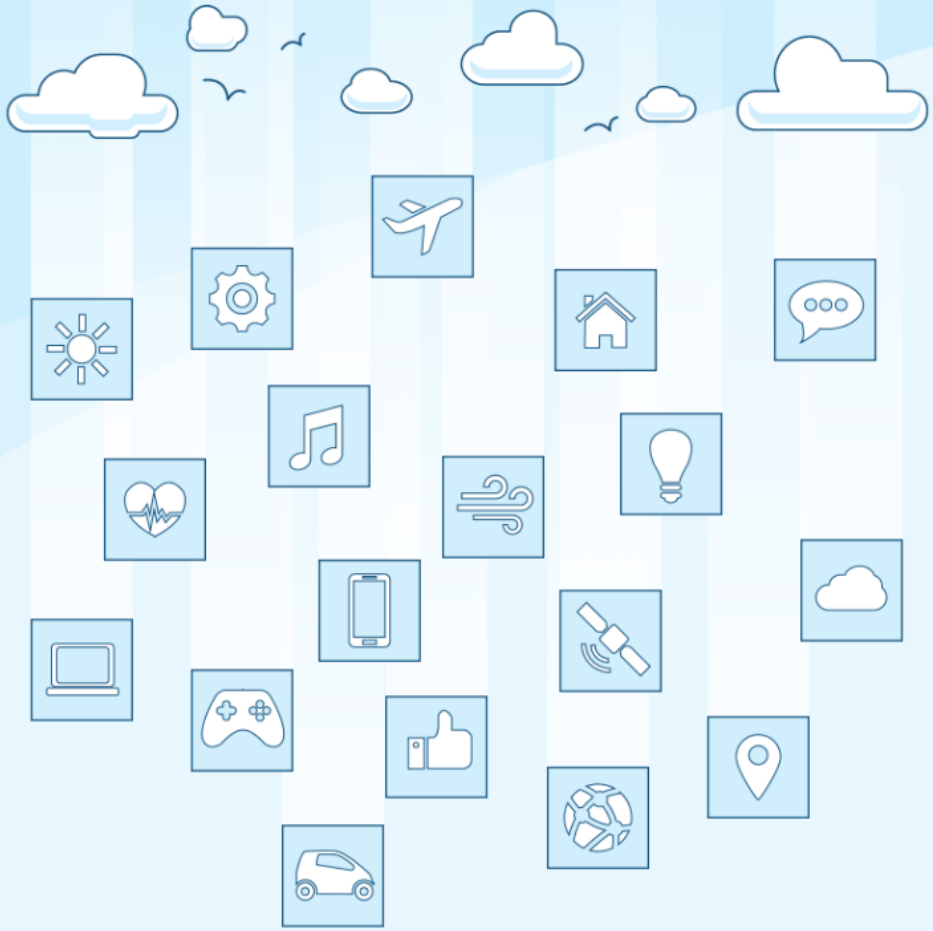
```
writer.writeEvent(routingKey, message)
```

- **Read guarantees**:
  - All the events from a set of Streams will be read by *only one reader in a group of readers*.
  - Application support for reader recovery: Consistent information of reader positions.



Why Pravega?

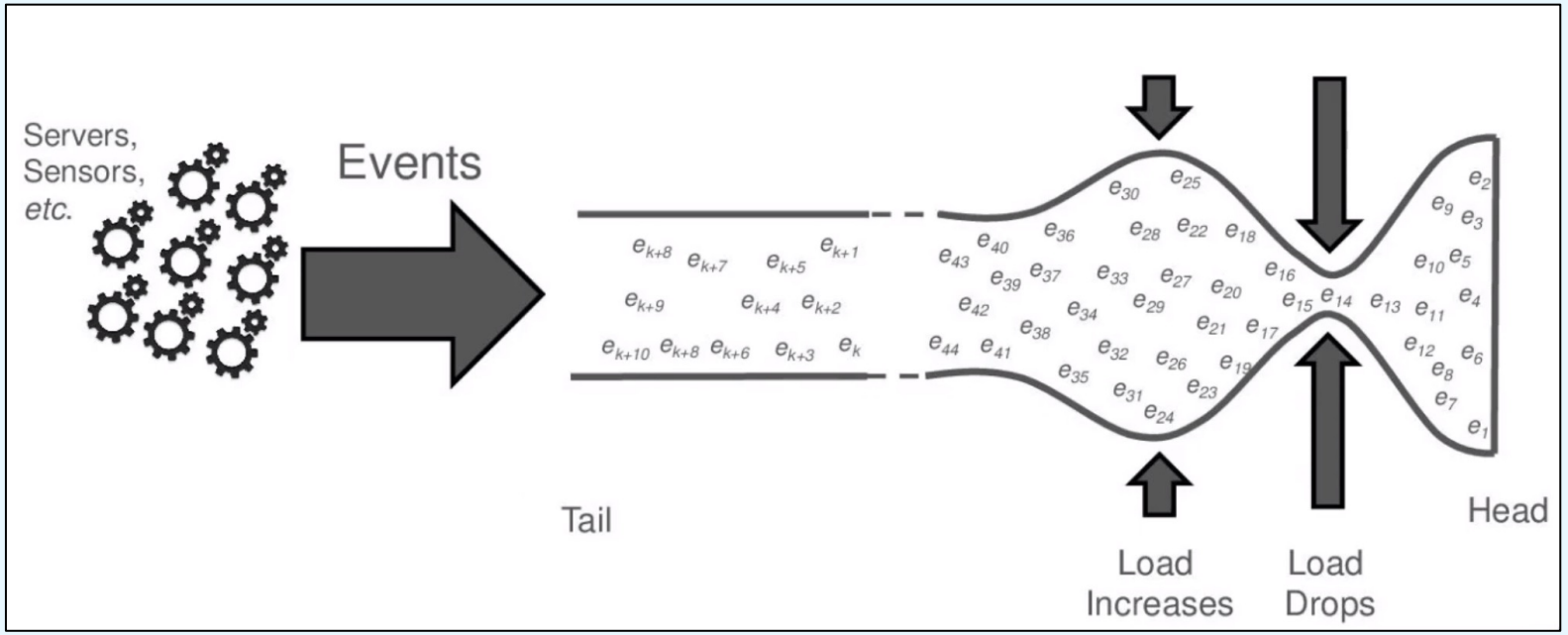
- **Unlimited retention**:
  - Stream segments can be stored in Tier 2 forever.
- **Unified storage primitive**:
  - Sweet spot in latency vs throughput trade-off: copes with both real-time/batch analytics.
- **Data durability:**
  - Data is durably stored in both tiers.
- **Parallelism:**
  - Multiple readers and writers may read/write on the same stream in parallel.
- **Guarantees for data processing**:
  - Exactly-once semantics.
  - Consistent event ordering (enforced via writer routing key).



Stream Autoscaling in Pravega

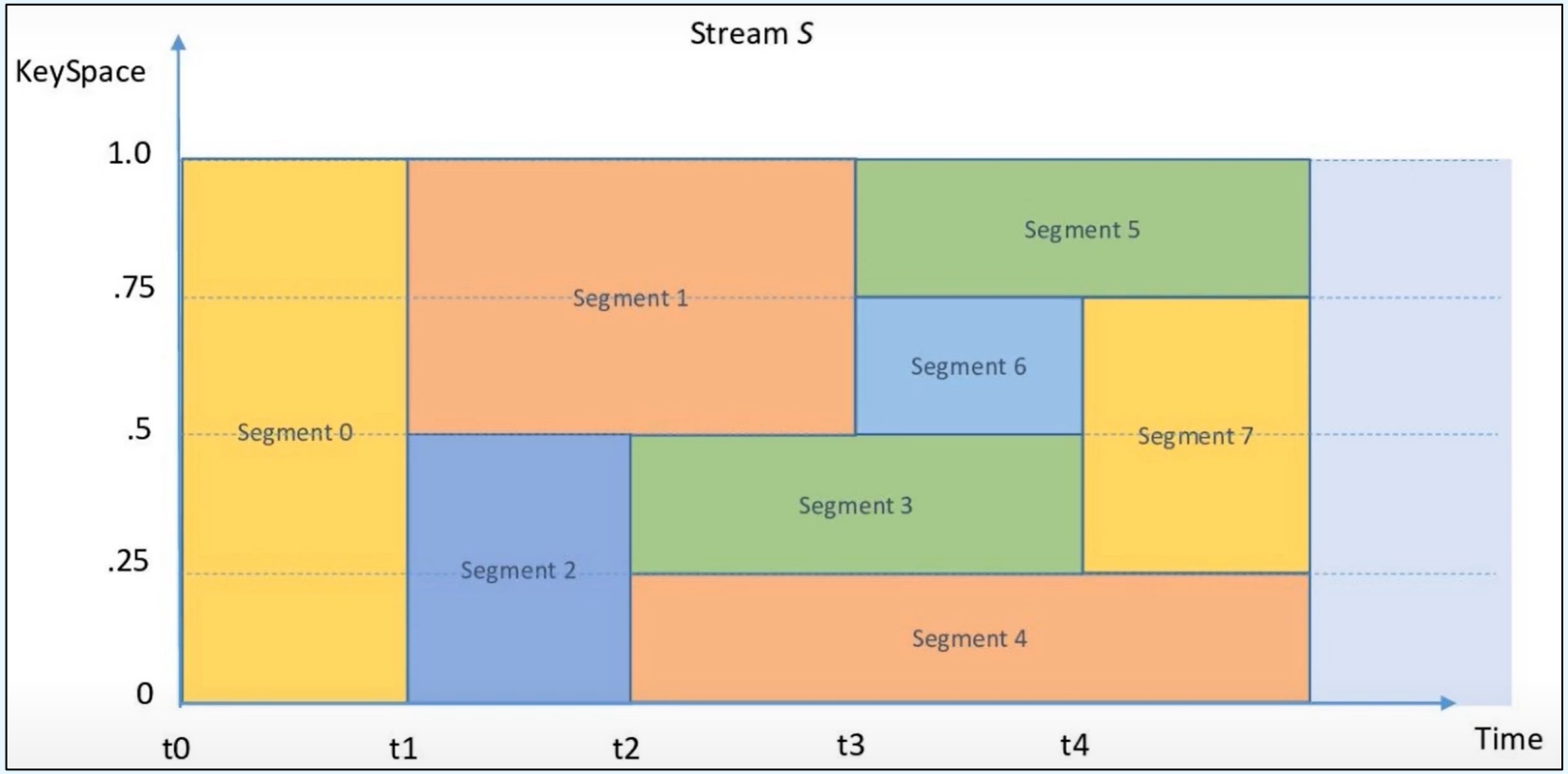
Data Streams: Workload Variations

- Load in a data stream may be **dynamic**.
- Load peaks, daily patterns.
- Ideally, Stream parallelism should vary accordingly.

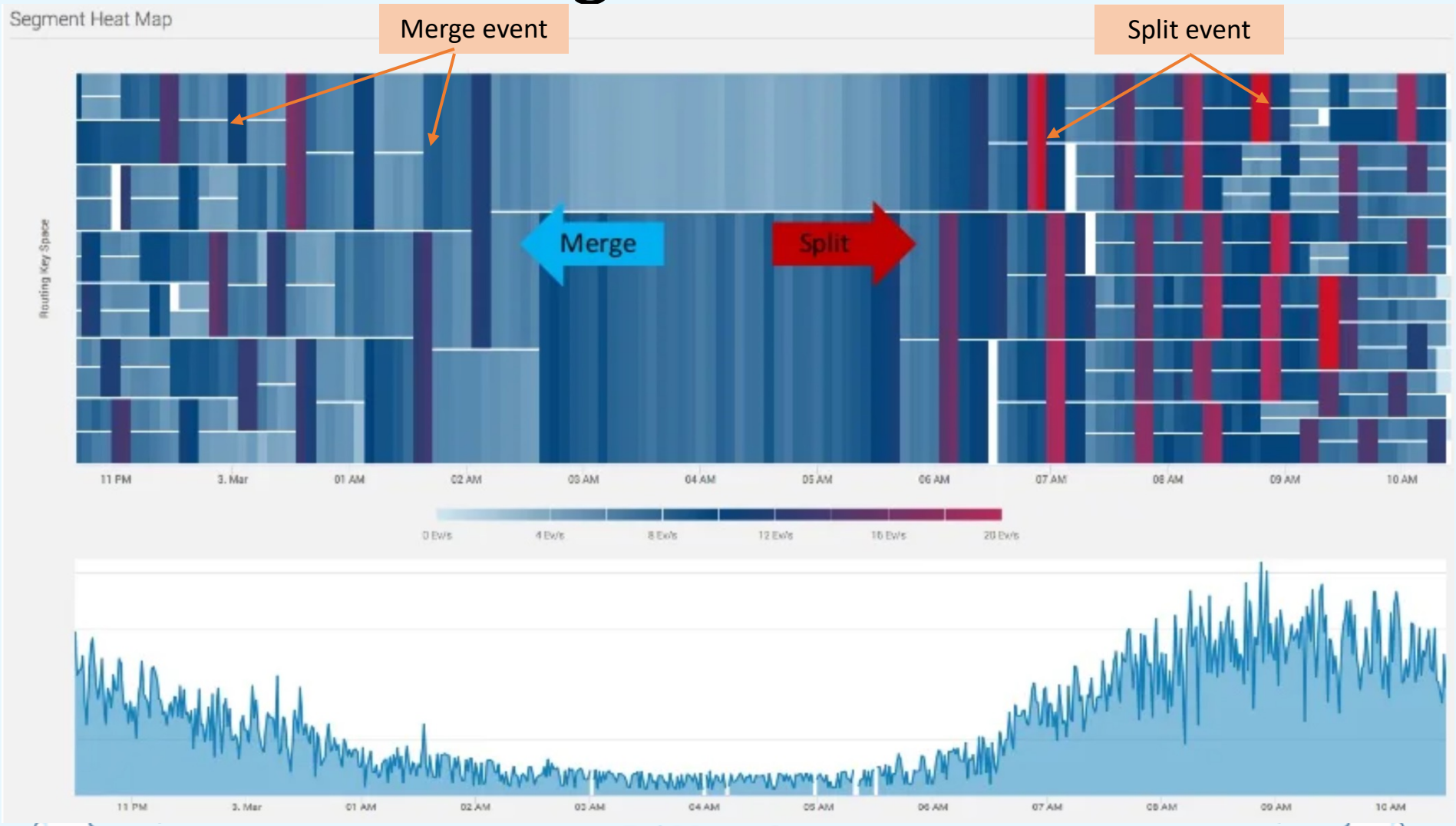


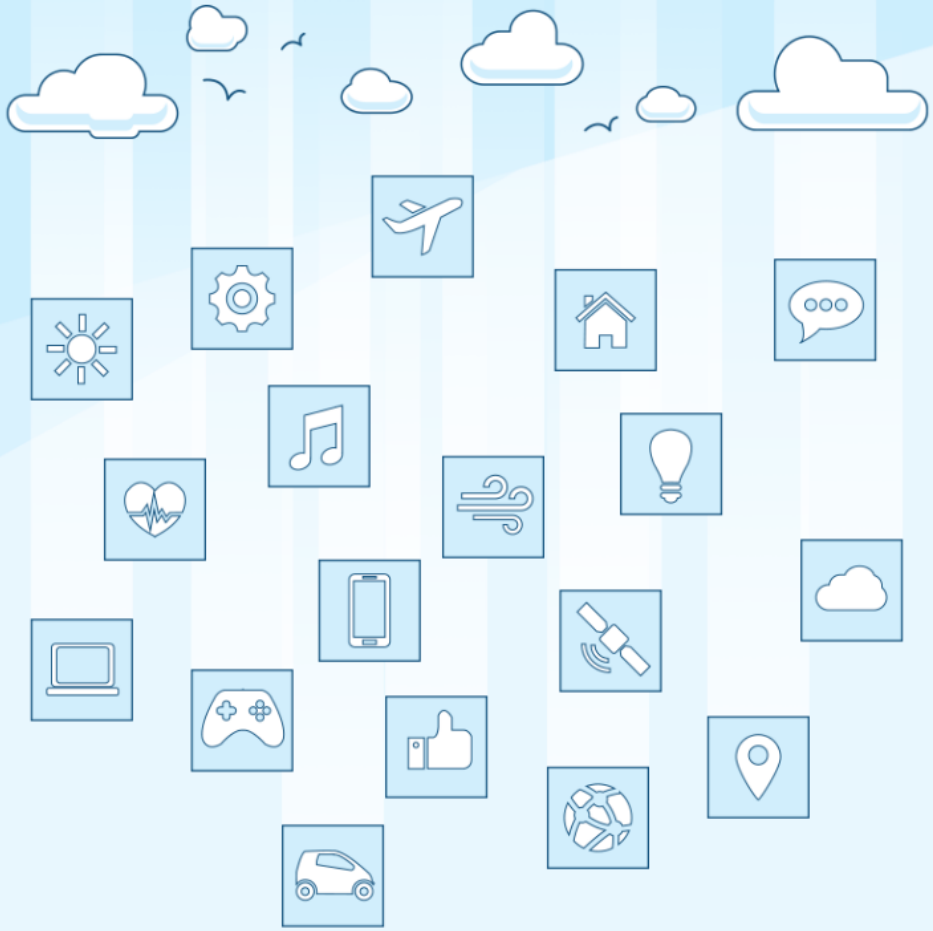
Stream Auto-Scaling

- **Dynamic** number of Segments per Stream.
- Defined via a Stream **policy**.
- Pravega **tracks** the load per-segment.
- It **triggers** up/down scale events.



Stream Auto-Scaling

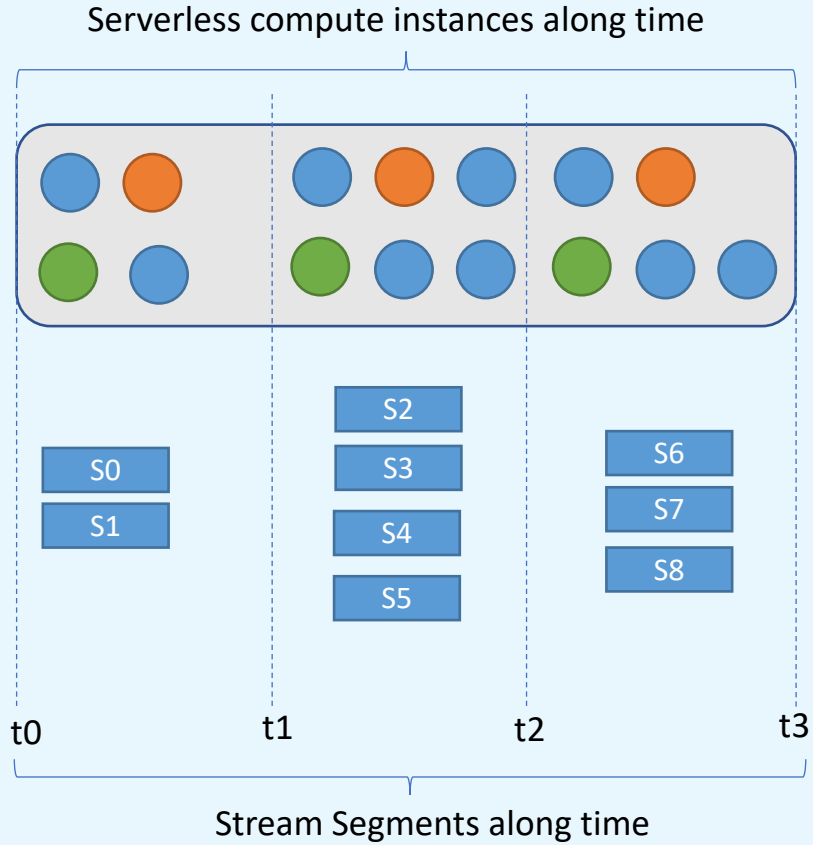


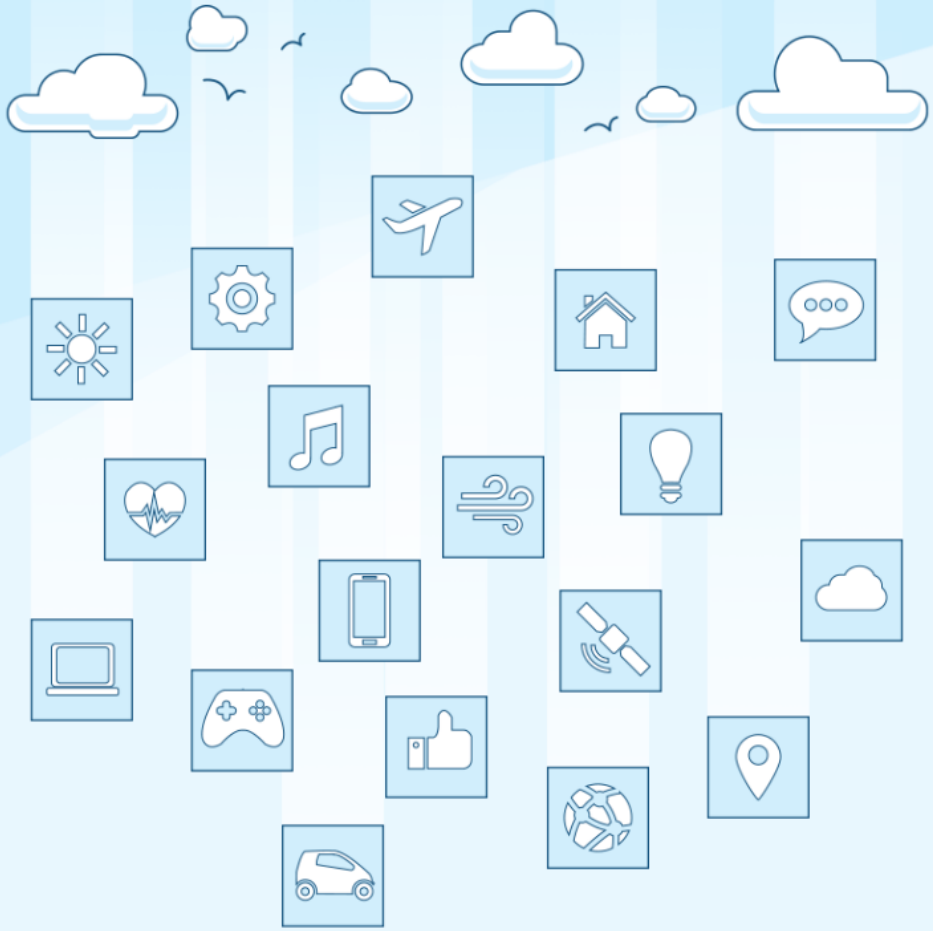


Serverless & Stream Auto-scaling

What's Next? Connecting the Dots

- Serverless frameworks take scale **up/down compute instances**:
  - Usually, such decisions are based on resource metrics (e.g., CPU).
- Pravega Stream parallelism: **software-based metric** to scale upon.
- **Goal**: exploit the **dynamic parallelism of Pravega Streams for scaling Serverless instances**.
  - # of Segments on a Stream as a complementary metric to make scaling decisions.
- **Success story** of Pravega and Apache Flink:
  - Dynamically scale the parallelism of a Flink job based on the number of Segments on a Pravega Stream.





Thanks for your attention!
Q&A