



Introduction to Stream Processing With Apache Flink

Marta Paes (@morsapaes)
Developer Advocate

Data Scientist



Data Analyst

Introduction to Stream Processing With Apache Flink

Marta Paes (@morsapaes)
Developer Advocate

Data Engineer

About Ververica



Original Creators of
Apache Flink®



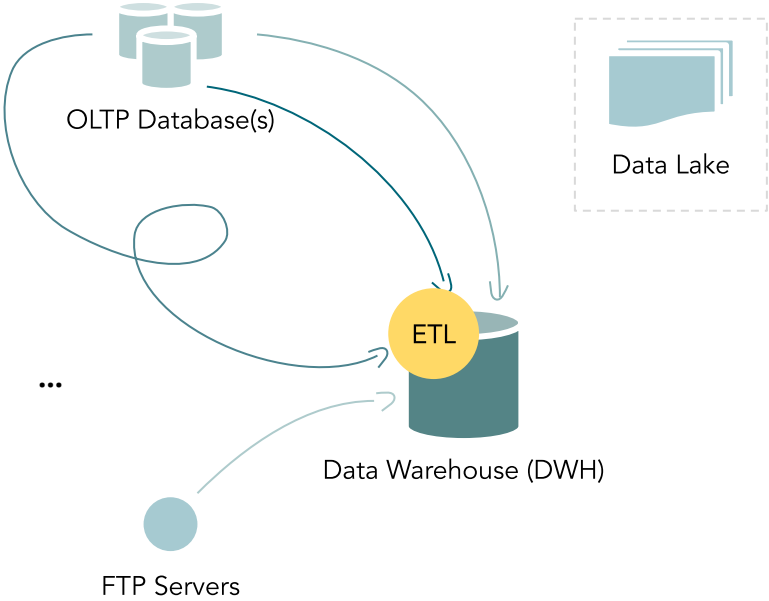
Enterprise Stream Processing
With Ververica Platform



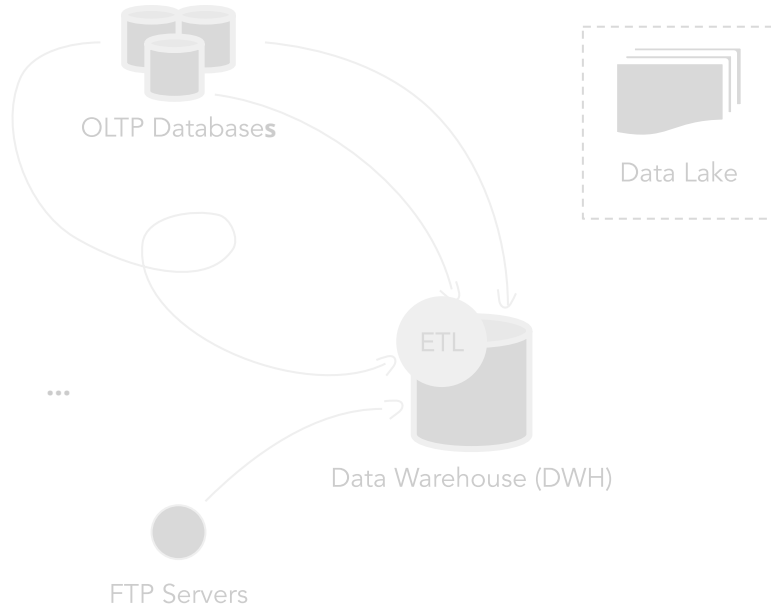
Part of
Alibaba Group



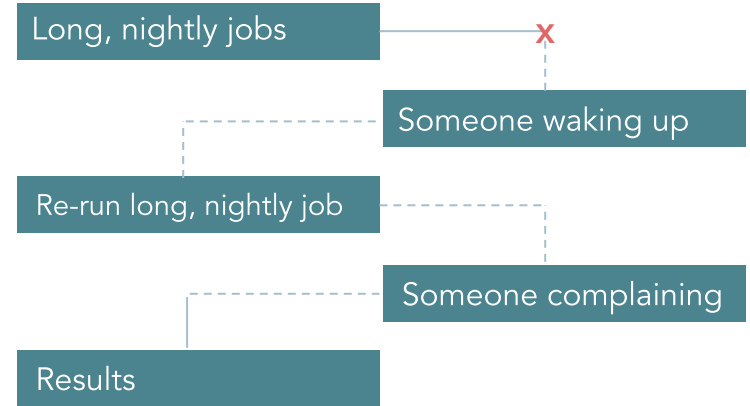
Analytics...Not that Long Ago



Analytics...Not that Long Ago



The quest for data...



But in the end...

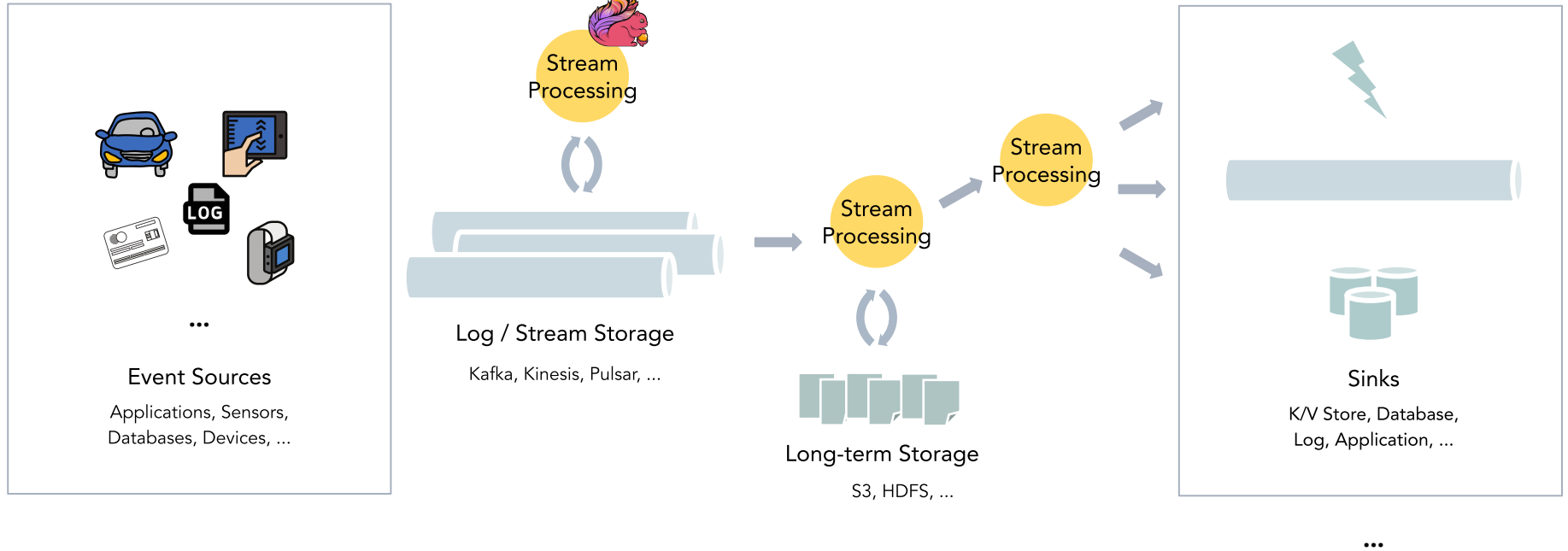
- Most source data is continuously produced
- Not everyone can wait for yesterday's data
- Most logic is not changing that frequently



Everything is a Stream

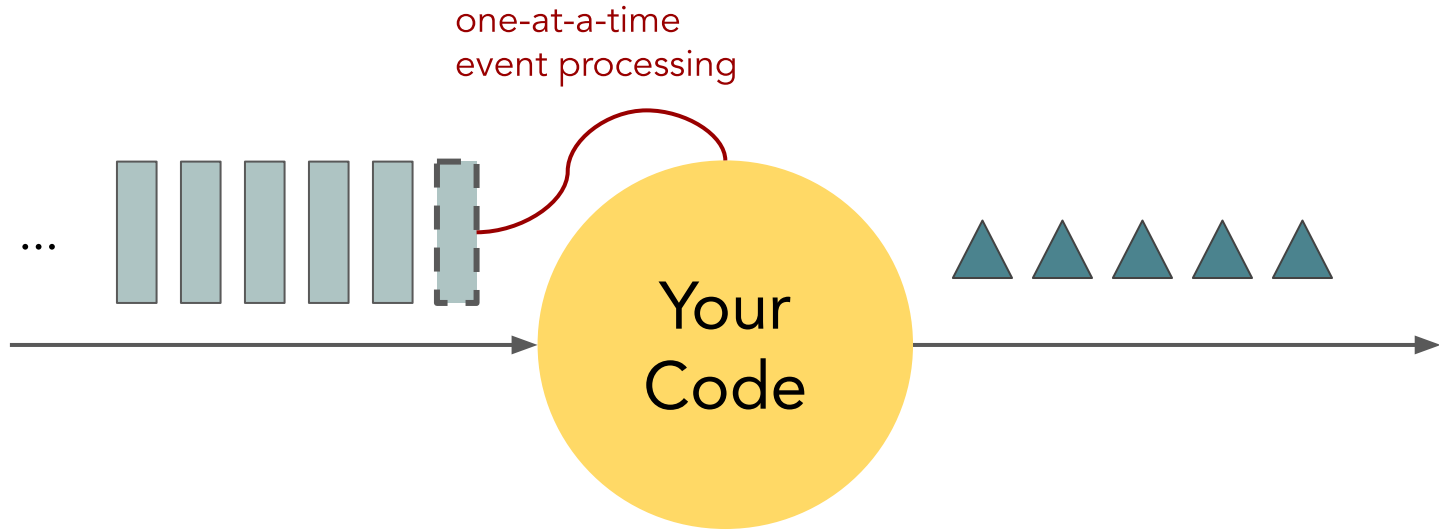
Everything is a Stream

Your static data records become **events** that are **continuously produced** and should be **continuously processed**.



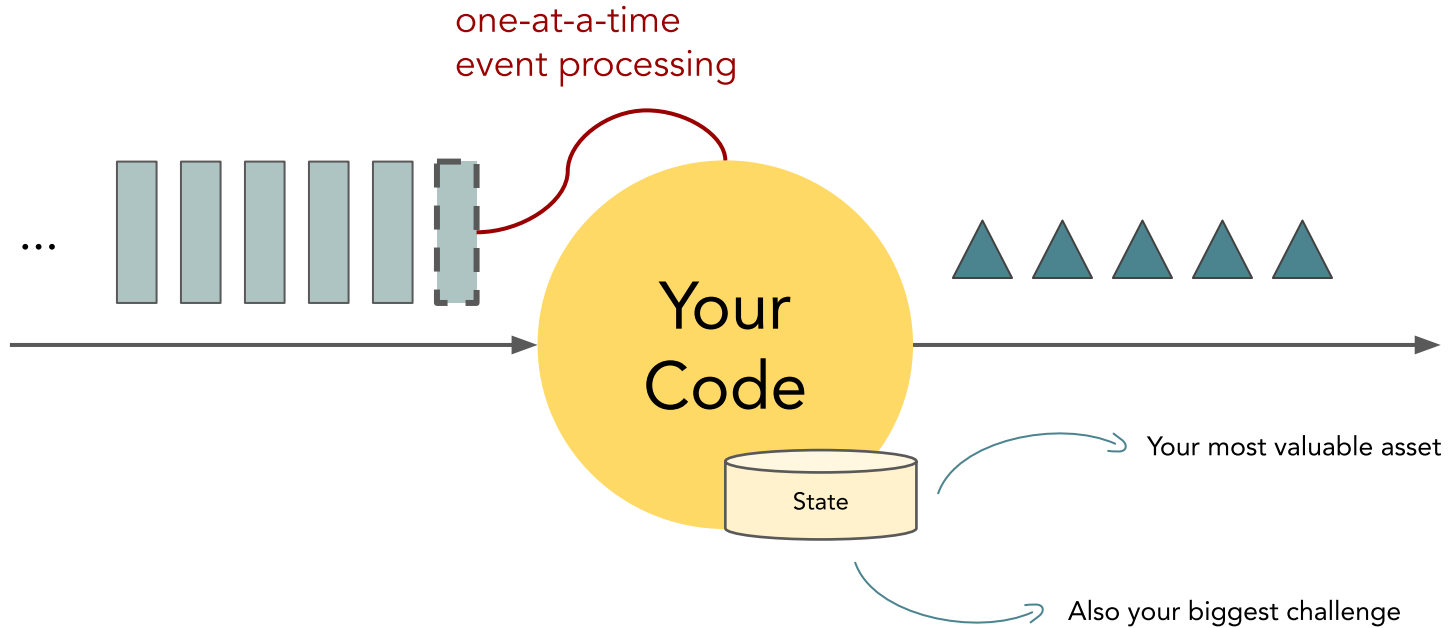
Stream Processing

Continuous processing of unbounded streams of events, one-at-a-time.



Stateful Stream Processing

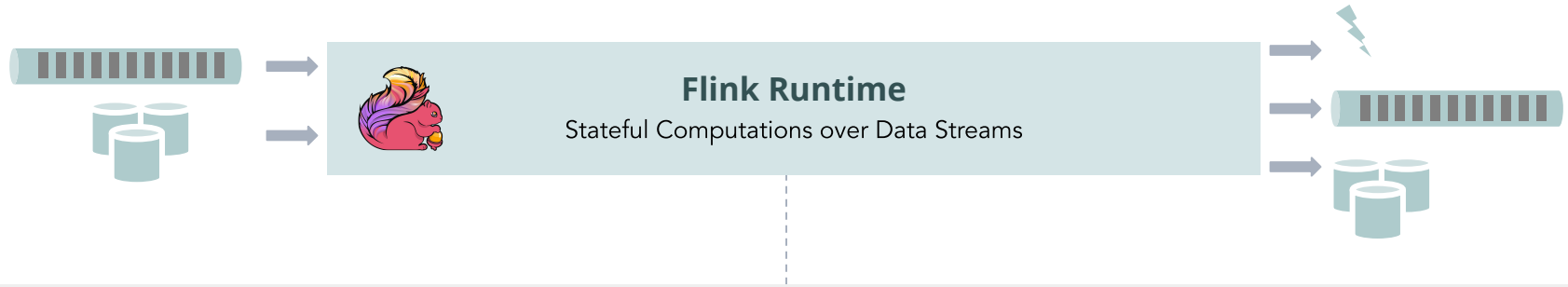
What if this simple model could “remember” events as they flow through?



So...what is Apache Flink?

What is Apache Flink?

Flink is an **open source** framework and distributed engine for **stateful stream processing**.



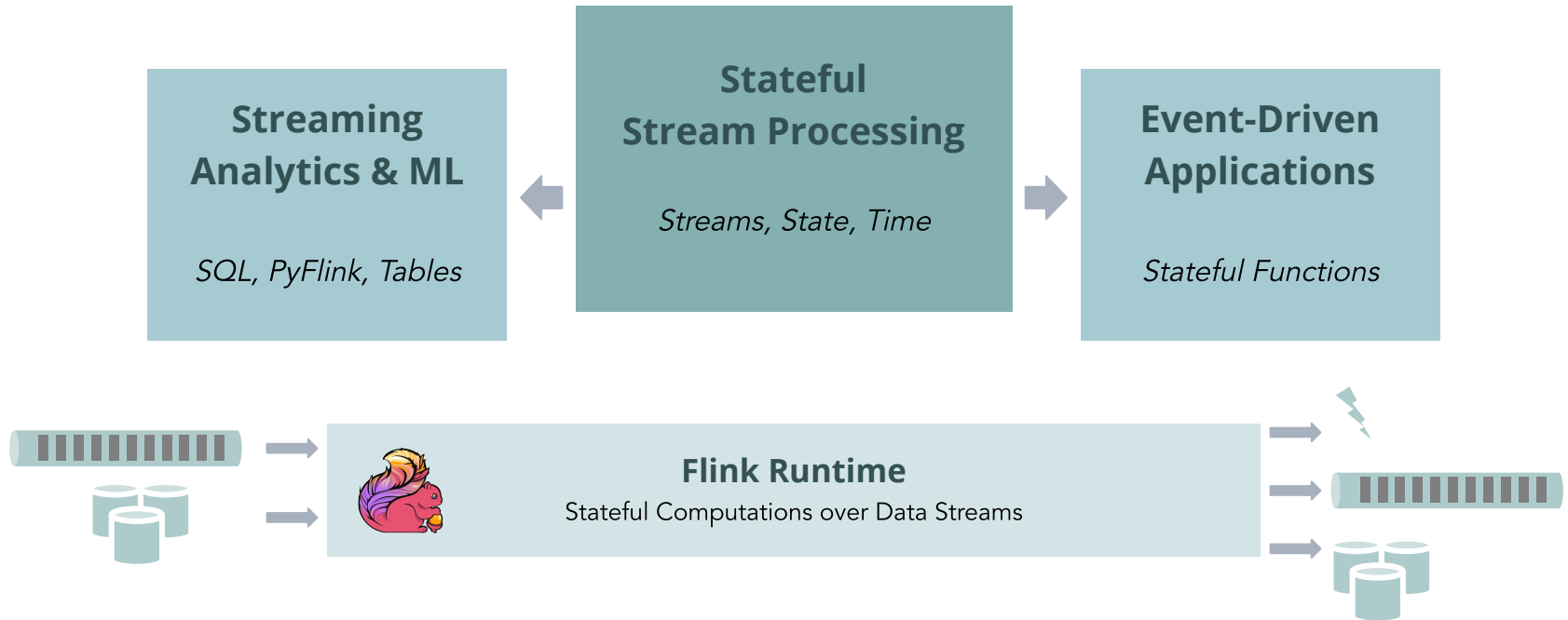
Stateful Computations over Data Streams

- State management is what makes Flink **powerful**.
- Consistent, one-at-a-time event processing is what makes Flink **flexible**.



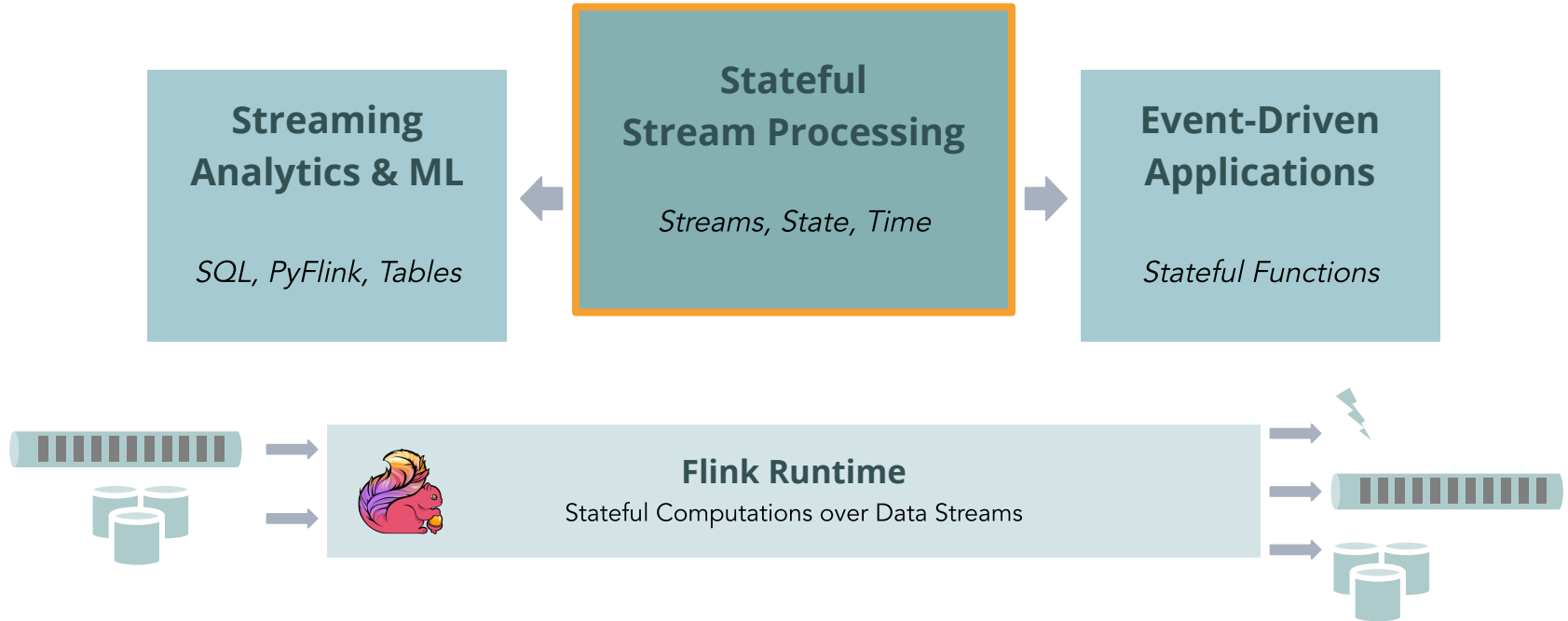
Stateful Computations over Data Streams

This gives you a robust foundation for a wide range of use cases:



Stateful Stream Processing

Classical, core stream processing use cases that build on the primitives of **streams**, **state** and **time**.



Stateful Stream Processing

Classical, core stream processing use cases that build on the primitives of **streams**, **state** and **time**.

- Explicit control over these primitives
- Complex computations and customization
- Maximize **performance** and **reliability**

Example Use Cases

NETFLIX

[Large-scale Data Pipelines](#)

ING 

[ML-Based Fraud Detection](#)

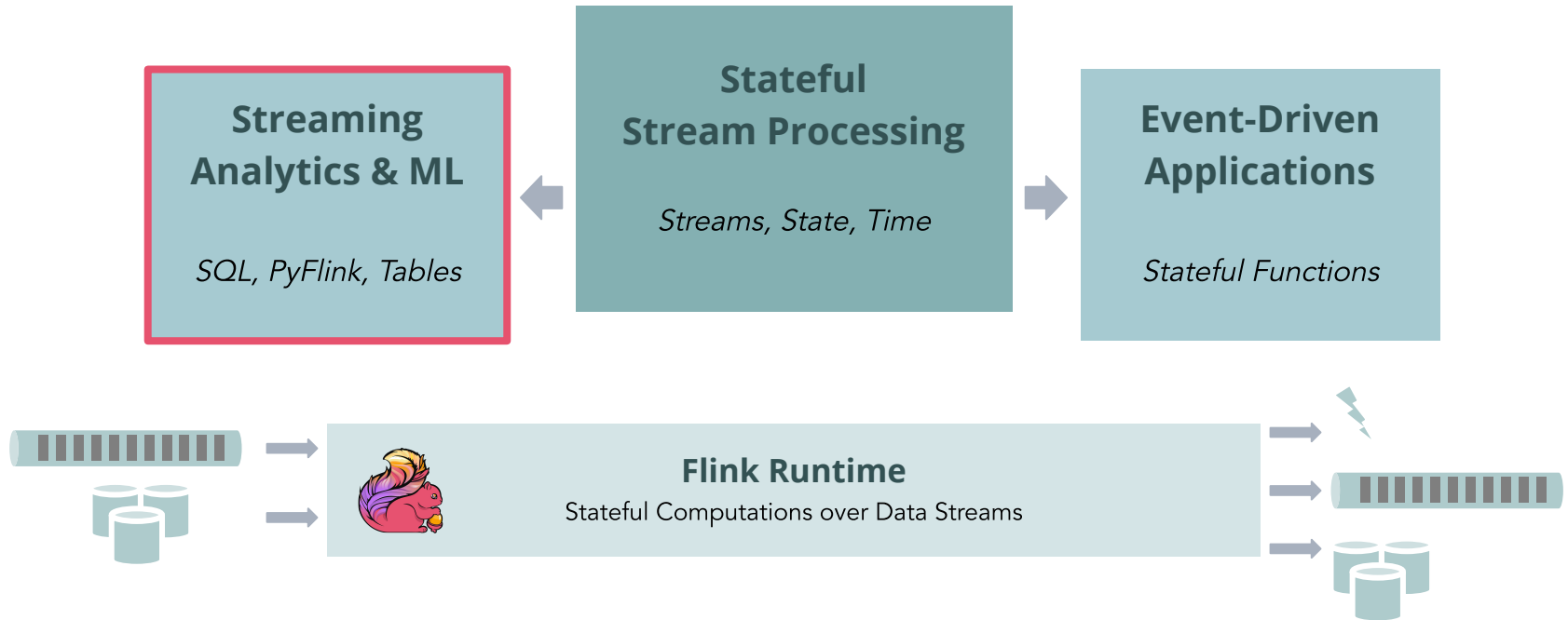
aws 

[Service Monitoring & Anomaly Detection](#)



Streaming Analytics & ML

More high-level or domain-specific use cases that can be modeled with SQL/Python and dynamic tables.



Streaming Analytics & ML

More high-level or domain-specific use cases that can be modeled with SQL/Python and dynamic tables.

- Focus on logic, not implementation
- Mixed workloads (batch and streaming)
- Maximize **developer speed** and **autonomy**

Example Use Cases

Uber

[E2E Streaming Analytics Pipelines](#)

 eventador.io

[Build and Maintain Materialized Views](#)

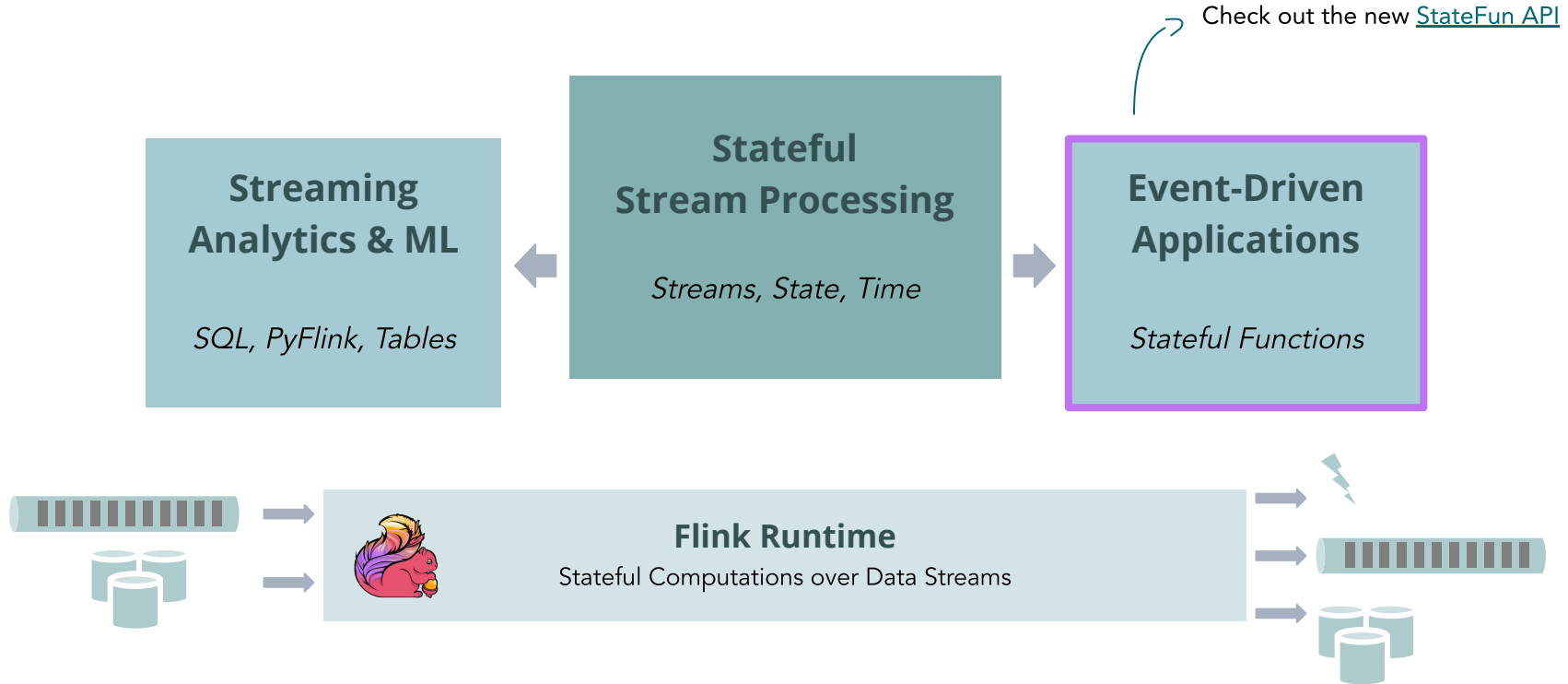
criteoL.

[ML Feature Generation](#)



Event-Driven Applications

Use cases that extend stream processing to stateful serverless applications.



More Apache Flink Users



How big can you go?

Alibaba: Double 11 / Singles Day

Real-time Data Applications

incl. sub-second updates to the GMV dashboard

Search

Rec.

Ads

BI

Security



Infrastructure

>5K
nodes



>500K
CPU cores

Data Size



985PB

State Size (Biggest)



100TB

Throughput (Peak)



2.5B
events/sec

Latency



Sub-sec



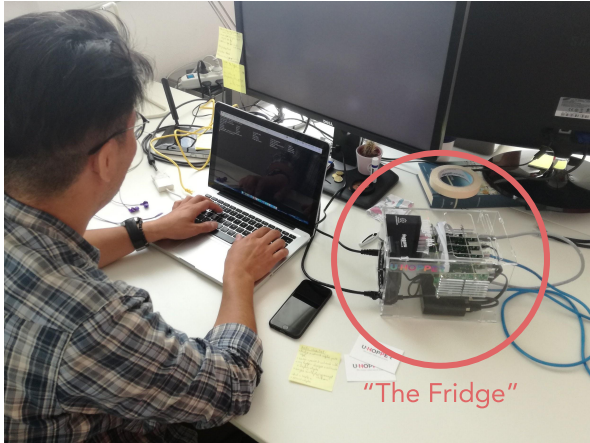
...but you can also go small...

U-Hopper: FogGuru

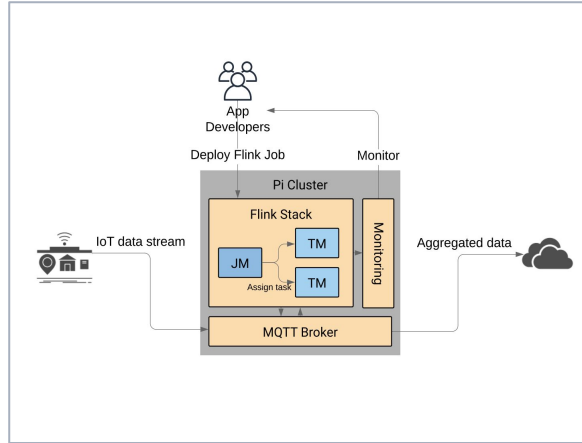
FogGuru is a platform for developing and deploying fog applications in **resource-constrained devices**.

Demo

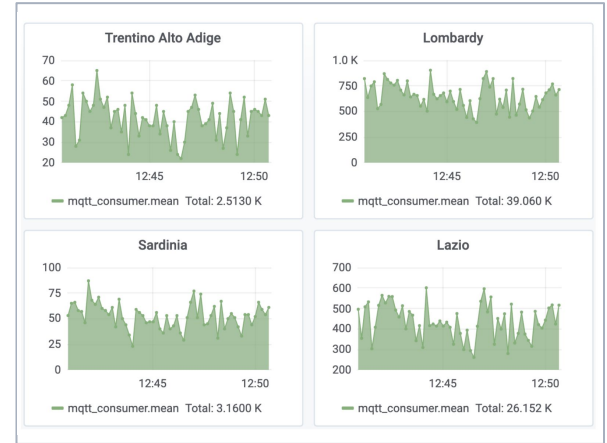
Cluster of 5 Raspberry Pi 3b+



Docker Swarm + Flink + Mosquitto



Data volume: 800 events/sec



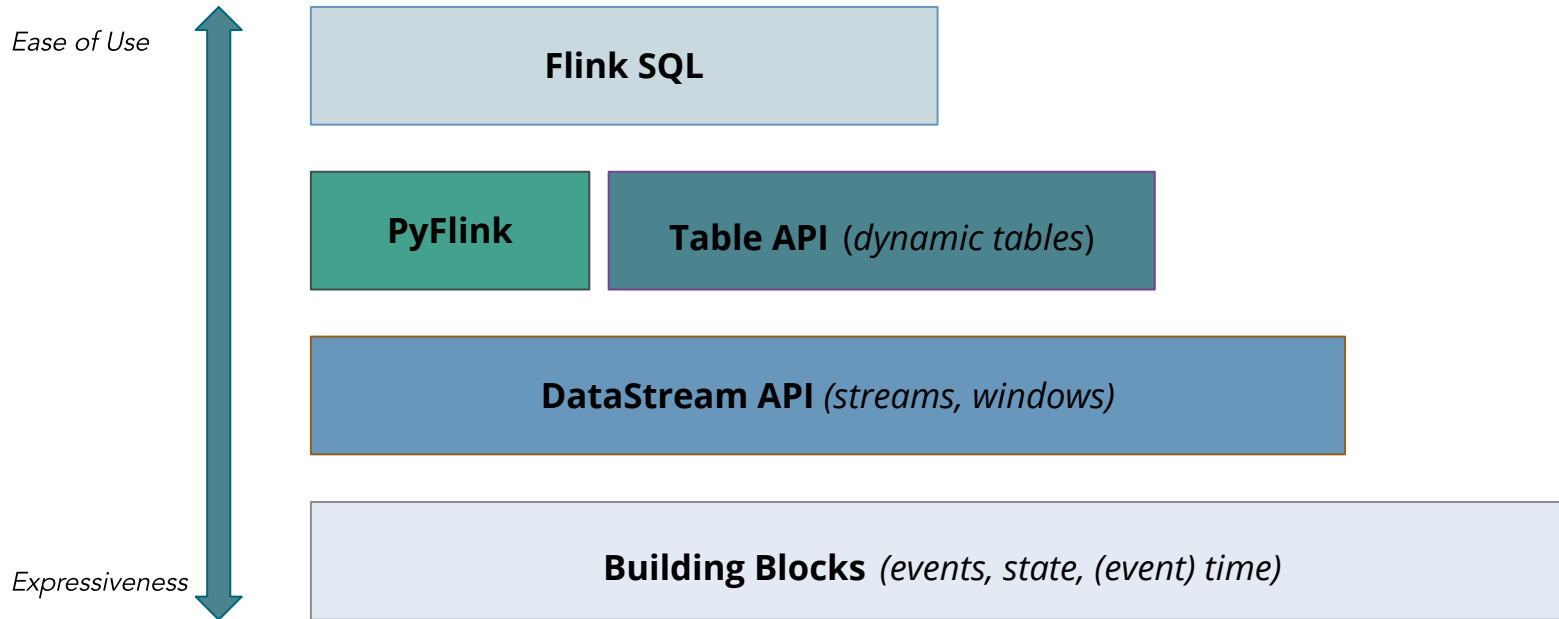
...or just use your laptop + an IDE.

What Makes Flink...Flink?



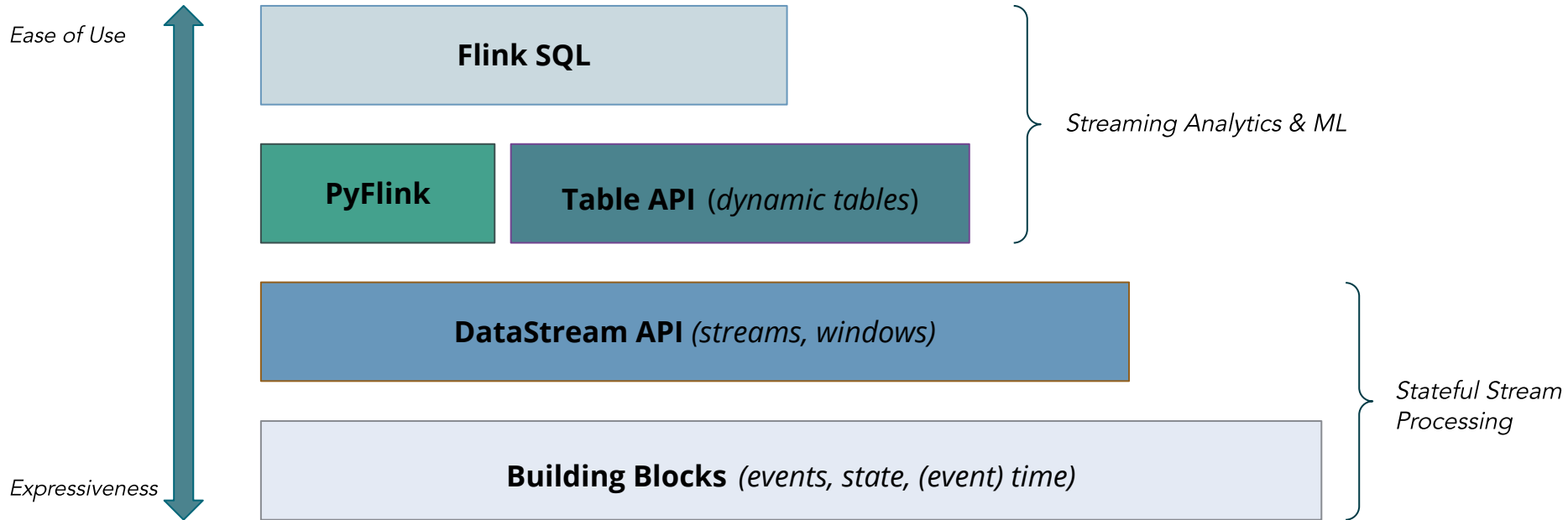
The Flink API Stack

Flink has layered APIs with different tradeoffs for **expressiveness** and **ease of use**. You can mix and match all the APIs!



The Flink API Stack

Flink has layered APIs with different tradeoffs for **expressiveness** and **ease of use**. You can mix and match all the APIs!



At the Core: Streaming Dataflows

```
DataStream<SensorReading> sensorData = env.addSource(new FlinkKafkaConsumer(...));
```

Source

```
DataStream<SensorReading> avgTemp = sensorData  
    .map(r -> new SensorReading(r.id, r.timestamp, (r.temperature-32) * (5.0/9.0)))  
    .keyBy(r -> r.id)  
    .timeWindow(Time.seconds(5))  
    .apply(new TemperatureAverager());
```

Transformations

```
avgTemp.addSink(new ElasticSearchSink(...));
```

Sink



At the Core: Streaming Dataflows

```
DataStream<SensorReading> sensorData = env.addSource(new FlinkKafkaConsumer(...));
```

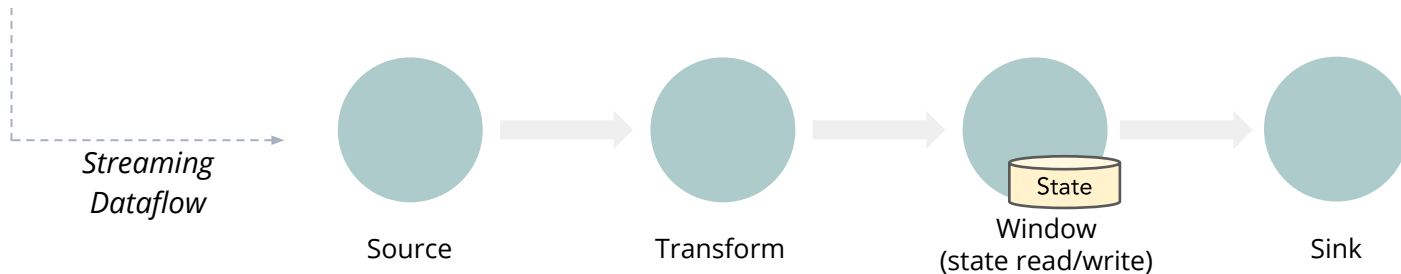
Source

```
DataStream<SensorReading> avgTemp = sensorData  
    .map(r -> new SensorReading(r.id, r.timestamp, (r.temperature-32) * (5.0/9.0)))  
    .keyBy(r -> r.id)  
    .timeWindow(Time.seconds(5))  
    .apply(new TemperatureAverager());
```

Transformations

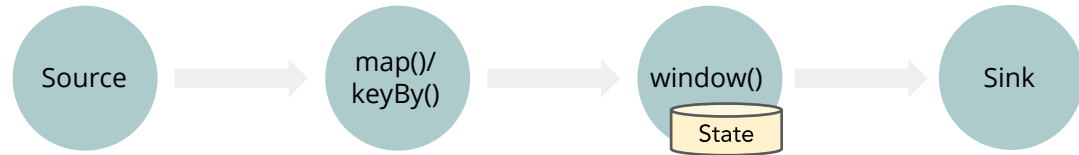
```
avgTemp.addSink(new ElasticSearchSink(...));
```

Sink



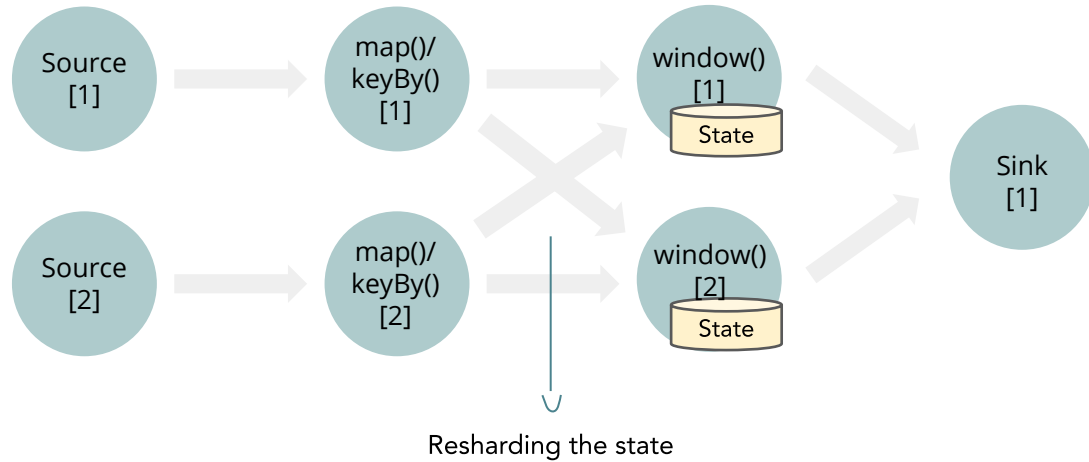
At the Core: Streaming Dataflows

Flink takes care of transforming your topology into a parallel dataflow that can run **distributed** on multiple machines.



At the Core: Streaming Dataflows

Flink takes care of transforming your topology into a parallel dataflow that can run **distributed** on multiple machines.

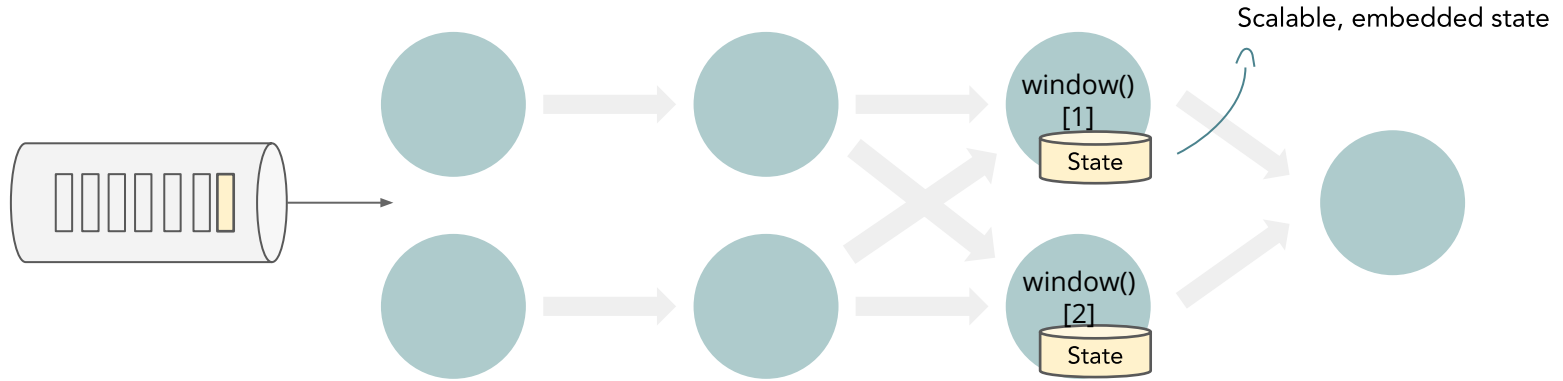


- State is re-scaled automatically with parallel operators



At the Core: State

Flink stores your state **locally** in-memory (on the JVM heap) or on disk (RocksDB).

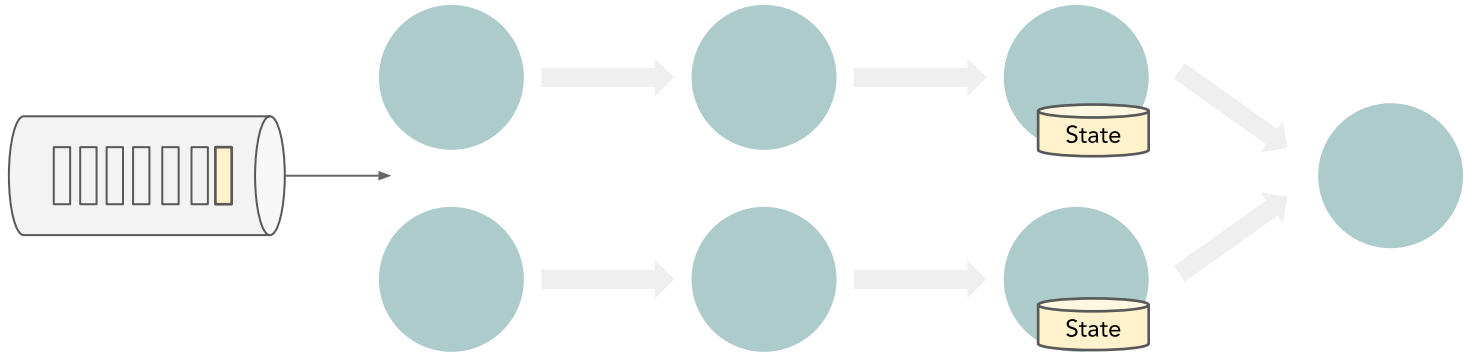


- State access at memory/disk speed
- The amount of state you can keep is only limited by heap/disk size



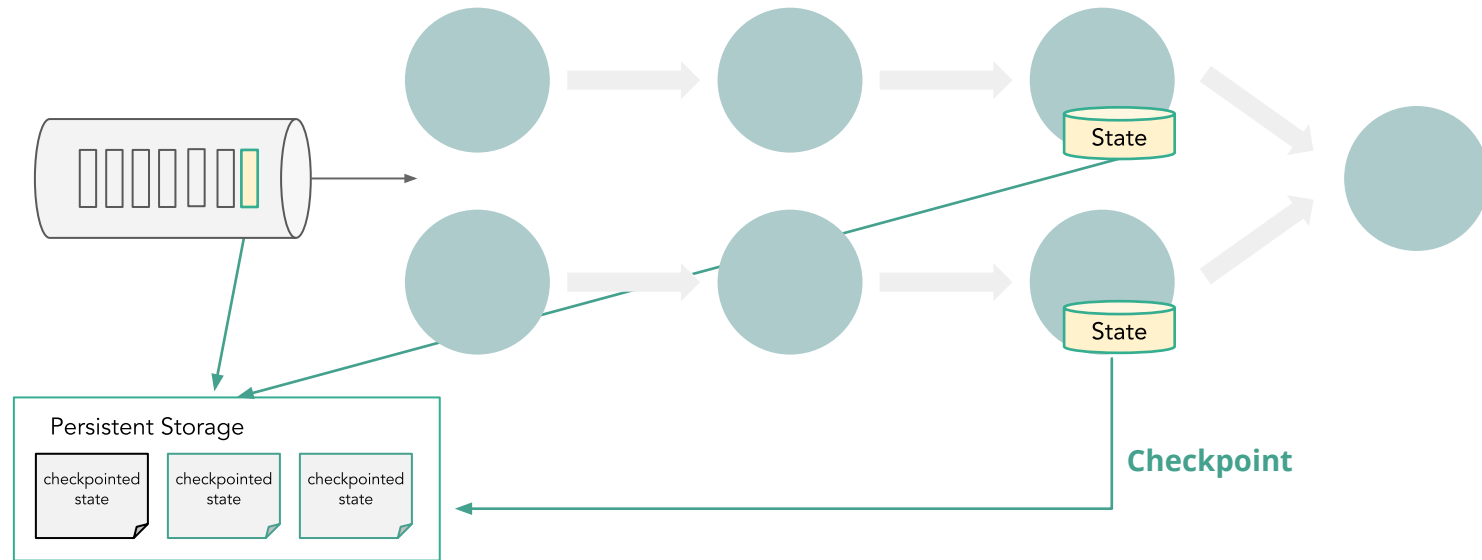
Fault Tolerance

What happens when something goes wrong? How does Flink guarantee that this state is **fault tolerant**?



Fault Tolerance: Checkpointing

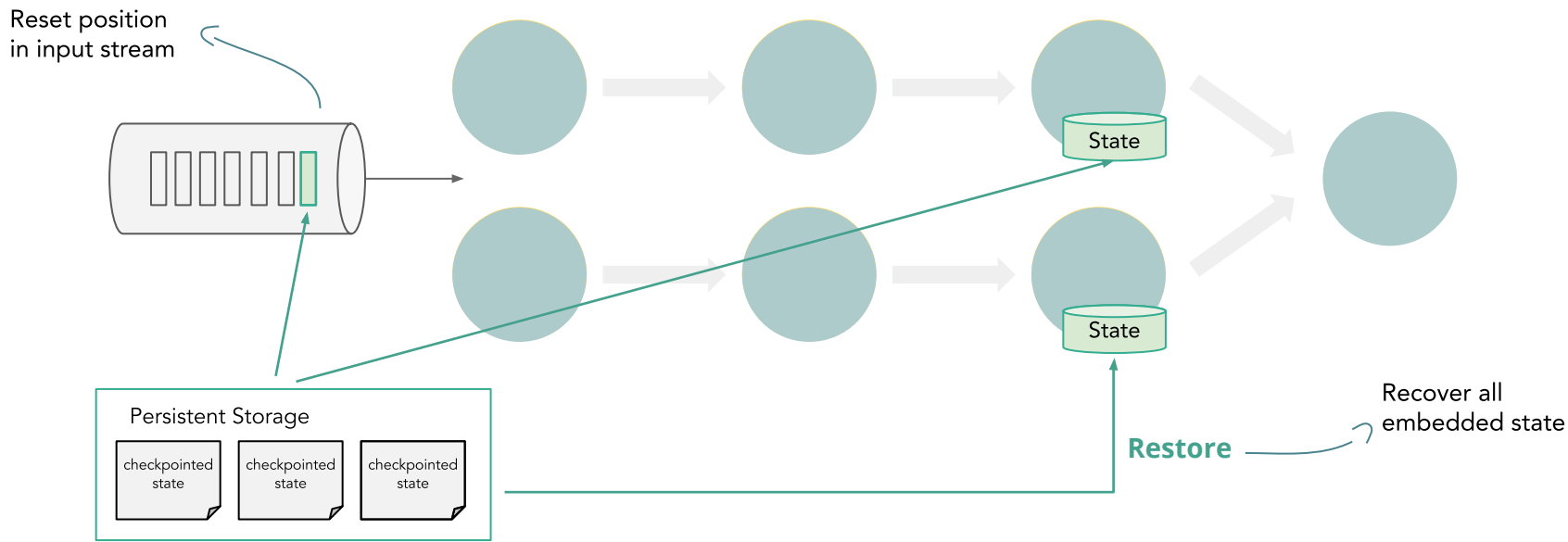
Flink takes **periodic snapshots** (i.e. checkpoints) of your application state to guarantee state consistency in case of failures.





Fault Tolerance: Recovery

Flink recovers all embedded state and positions in the input streams, giving you **failure-free execution** semantics with **exactly-once** consistency guarantees.



Beyond Fault Tolerance

You can also explicitly trigger these snapshots (i.e. savepoints) for planned, manual backup.

● Upgrades and
Rollbacks

● Cross Datacenter
Failover

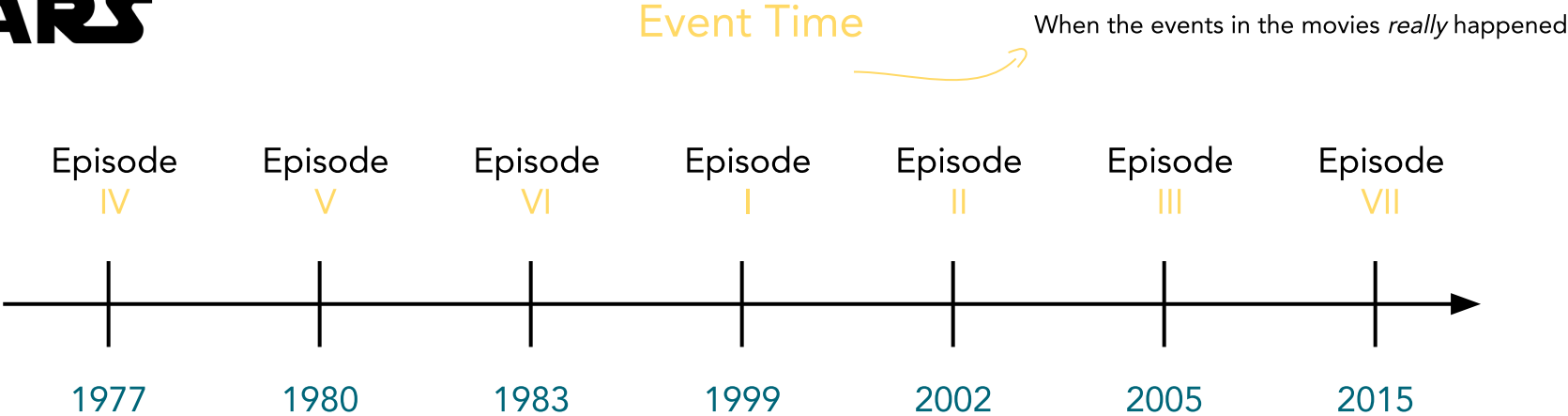
● State Archiving

● Blue / Green
Deployments

● Schema Evolution



At the Core: Time



When the movies were released

Processing Time



At the Core: Time

STAR WARS

Event Time

Time embedded in the records when they are produced



- Deterministic results
- Handle out-of-order or late events
- Trade-off result completeness/correctness and latency



At the Core: Time

STAR WARS

- Non-deterministic results
- Best performance and lowest latency
- Speed > completeness/correctness



System time of the processing machine

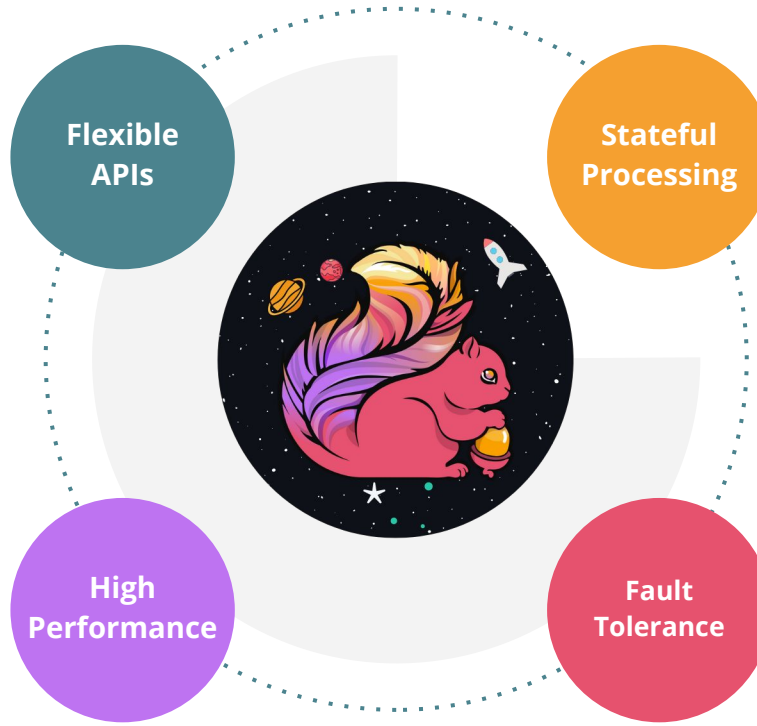
Processing Time



What Makes Flink...Flink?

- Ease of use/Expressiveness
- Wide Range of Use Cases

- Local State Access
- High Throughput/Low Latency



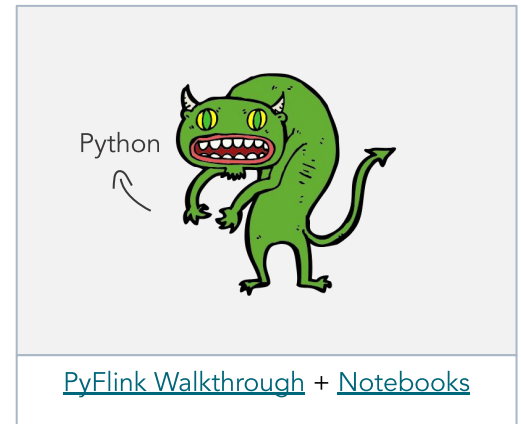
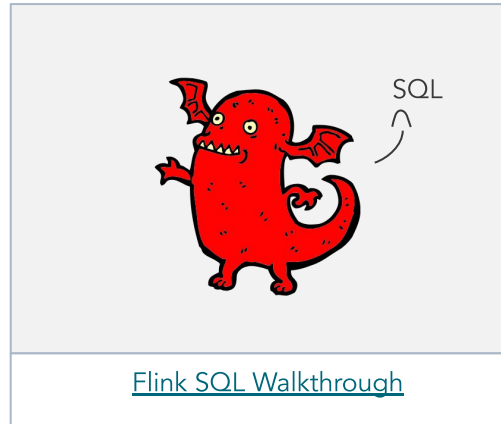
- State = First-class Citizen
- Event-time Support

- Distributed State Snapshots
- Exactly-once Guarantees



How to Get Started?

There are many ways to get started with Flink — and you don't have to know Java/Scala.

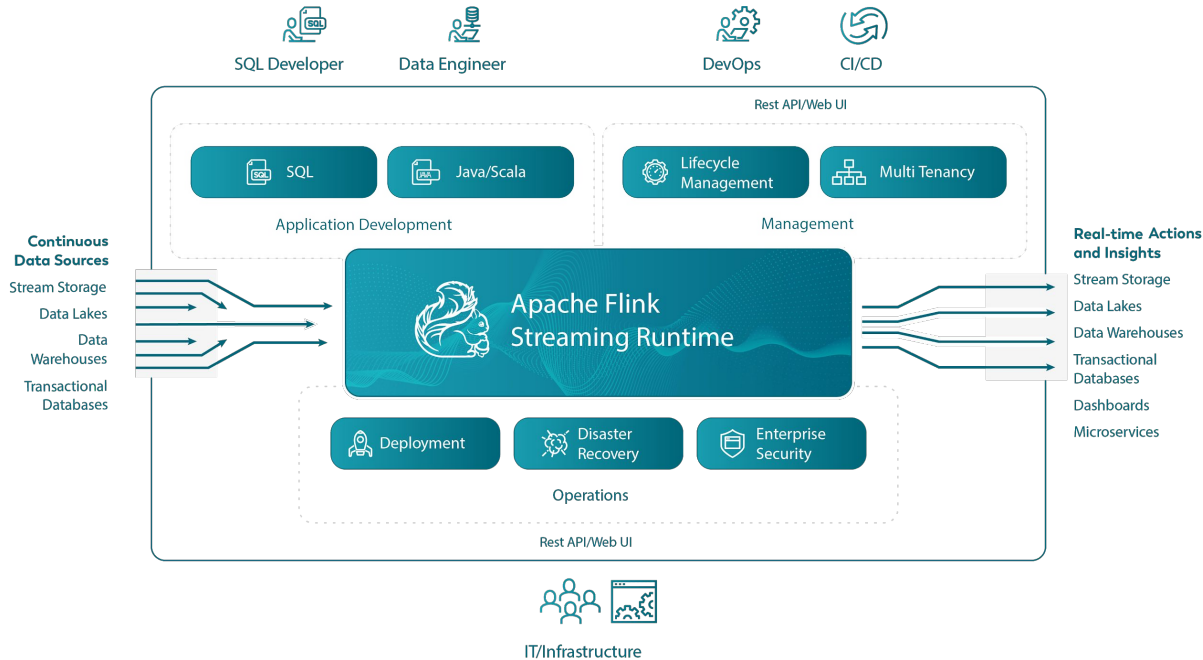


- Visit flink.apache.org
- Subscribe to the User Mailing List (for help!) or use the `apache-f1ink` tag on SO
- Follow [@ApacheFlink](https://twitter.com/ApacheFlink)



How to Get Started?

Get up and running with Flink on Kubernetes with Ververica Platform Community Edition!



- Permanently free
- Unlimited application size
- Commercial use

[Download](#)





Thank you!

Marta Paes (@morsapaes)
Developer Advocate