The Past, Present, and Future of Apache Flink®

Aljoscha Krettek, engineering manager & co-founder

Till Rohrmann, engineering manager & co-founder

dataArtisans

Past

It all started in 2014

2009 - 2014

since 2014

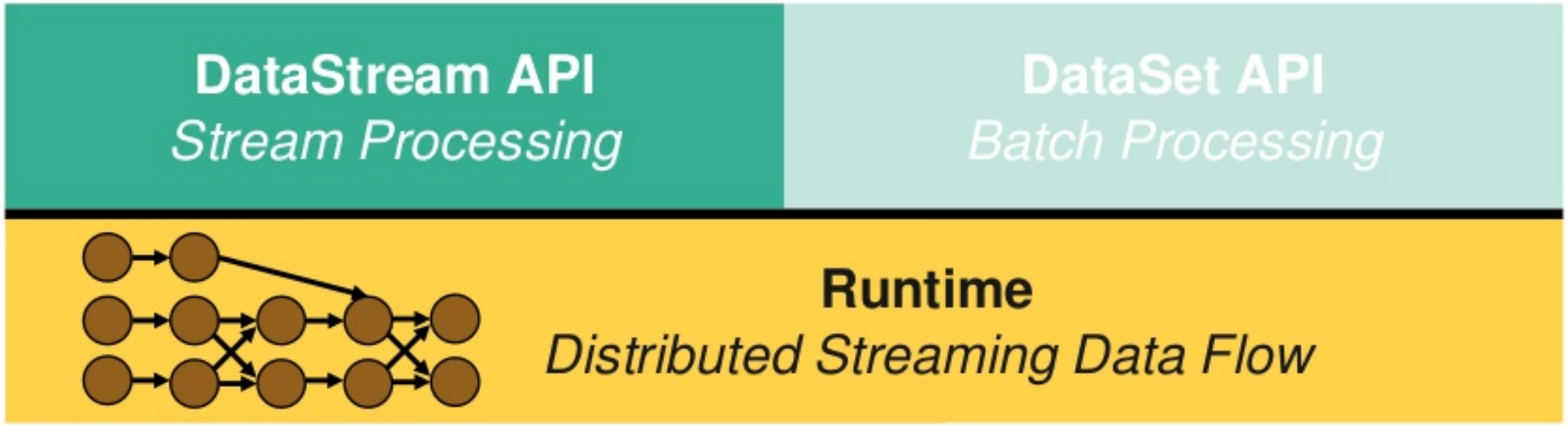
- Batch processor on top of streaming runtime
- First Apache Flink 0.6.0 release August 2014

 |

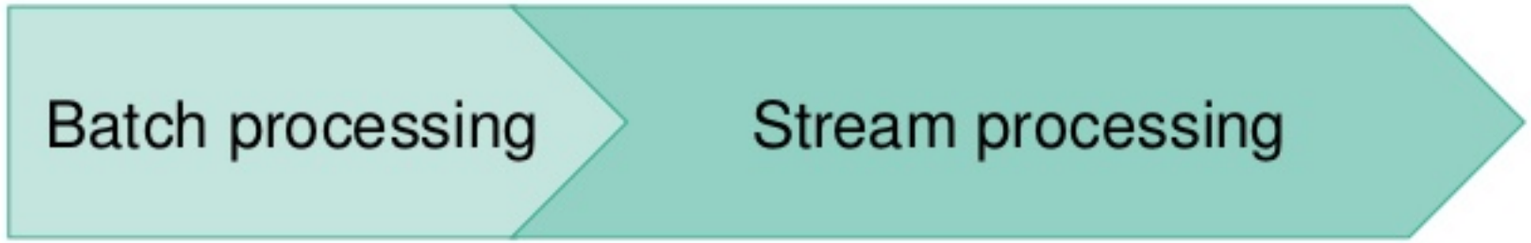
Batch processing

August 2014

Flink learns to stream in real time



 |

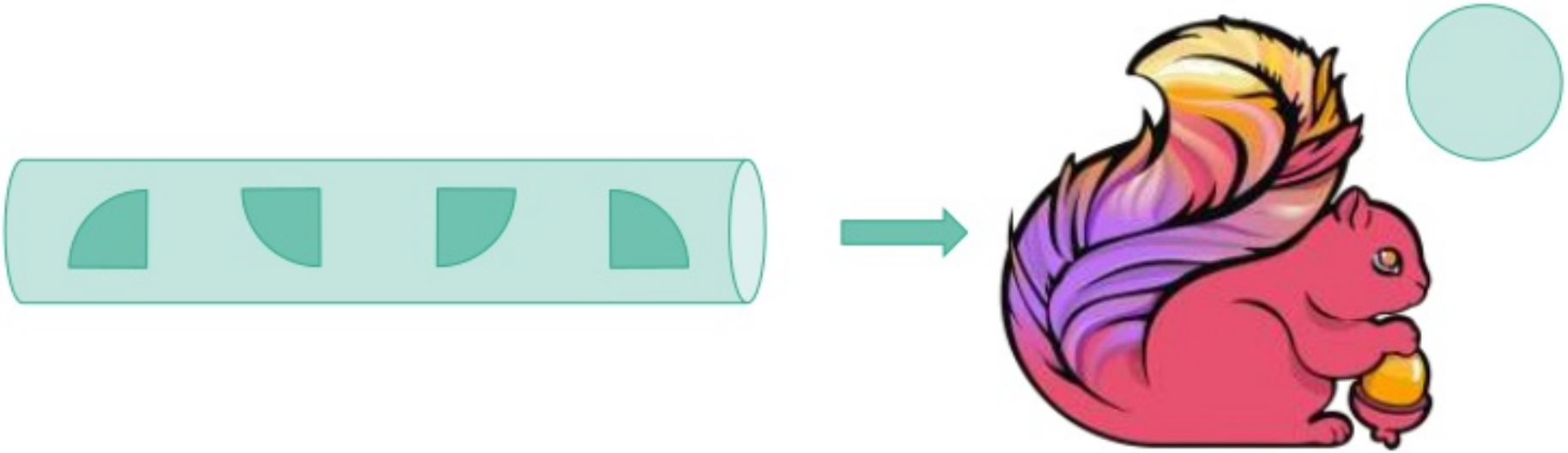


- Continuous & real-time

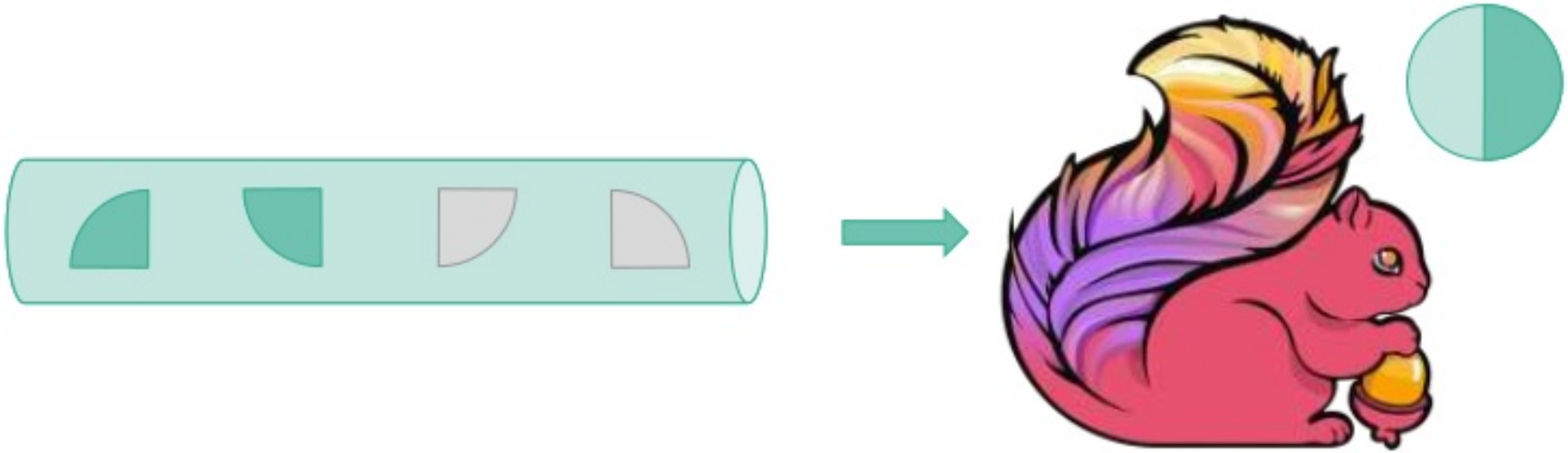
November 2014

 |

Flink learns to remember

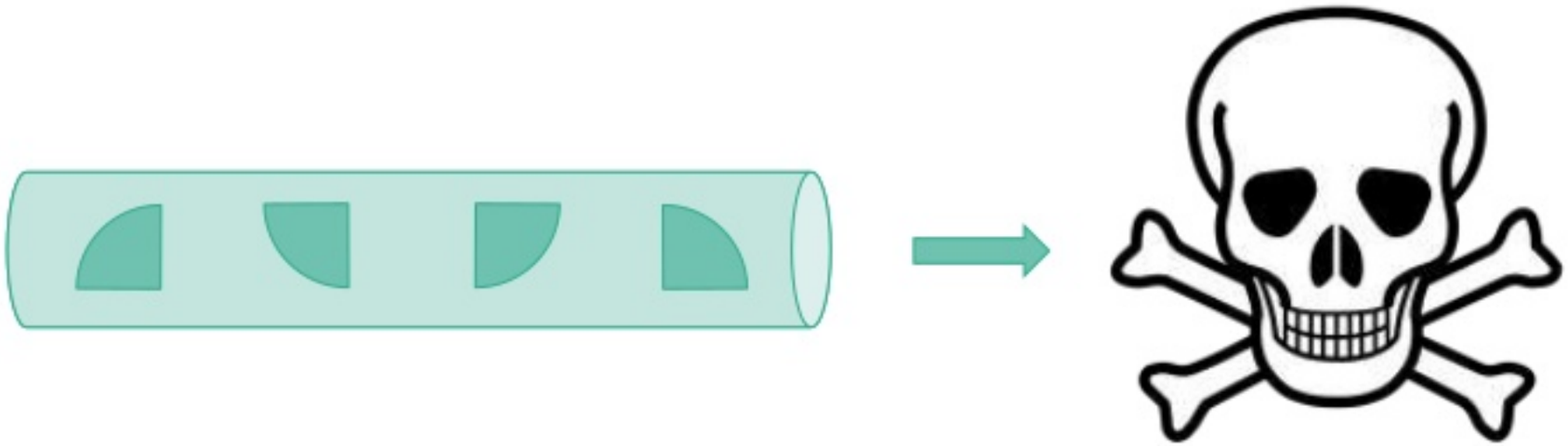


Flink learns to remember



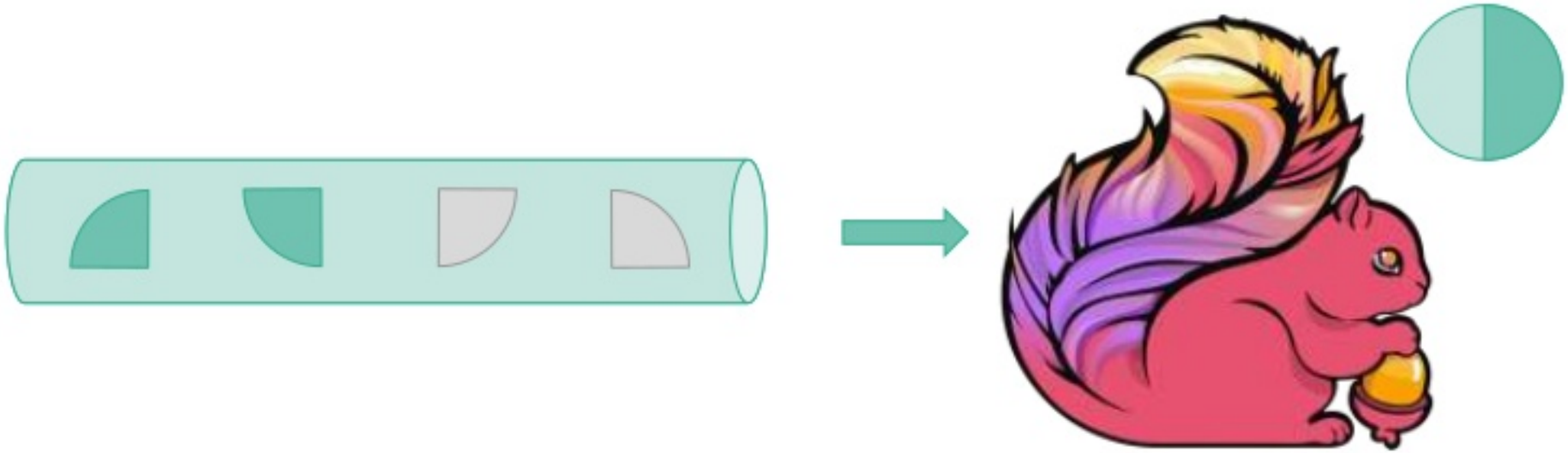
 |

Flink learns to remember



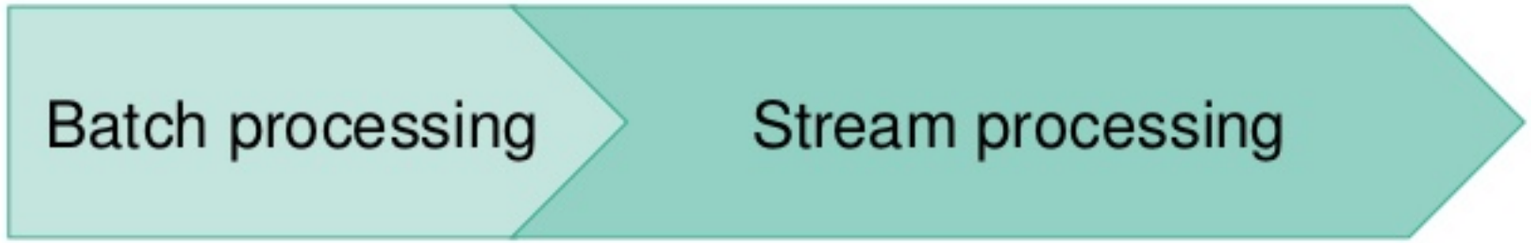
 |

Flink learns to remember



Remember where we left off

 |

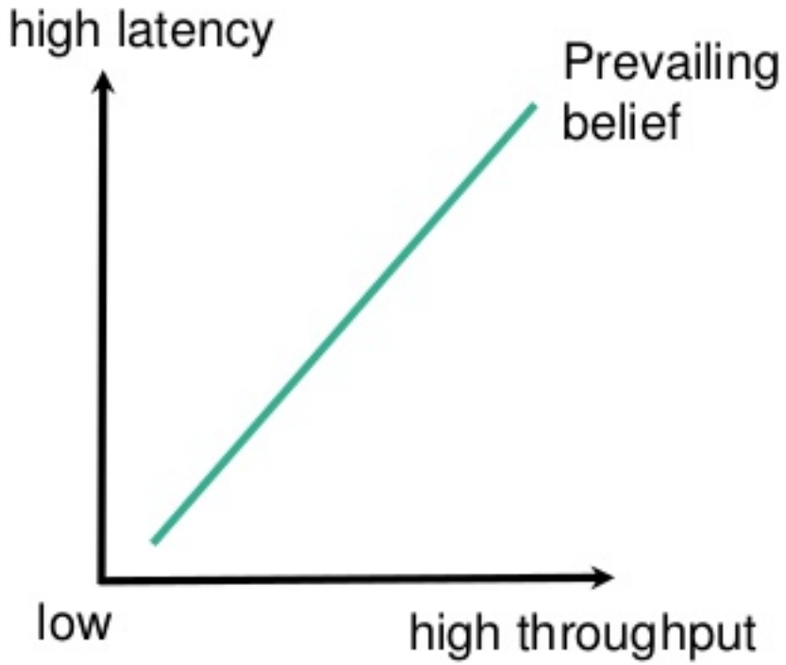


- Continuous & real-time
- Stateful & exactly once

June 2015

 |

Latency vs. Throughput?

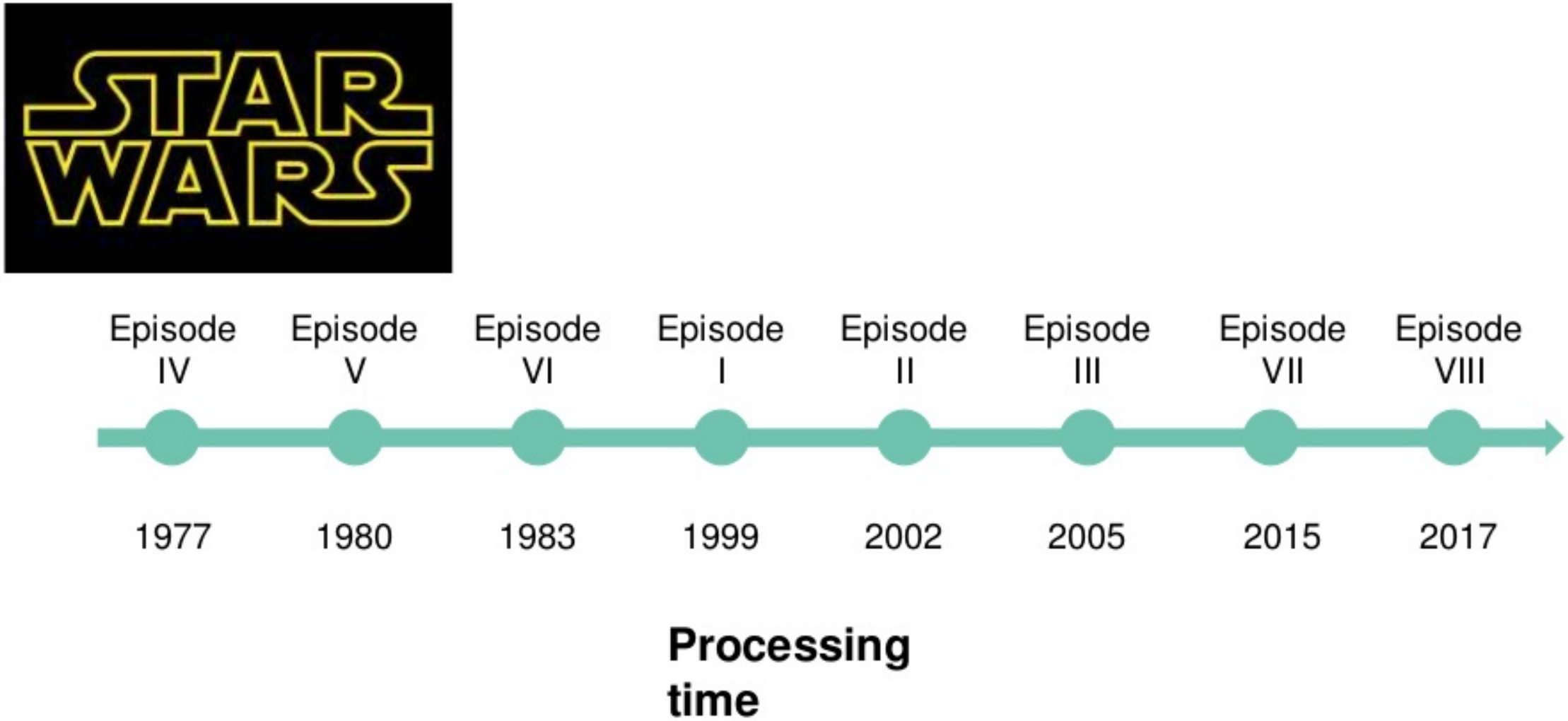


≠

- 10s of millions of events/s
- Latency down to 1 ms

 |

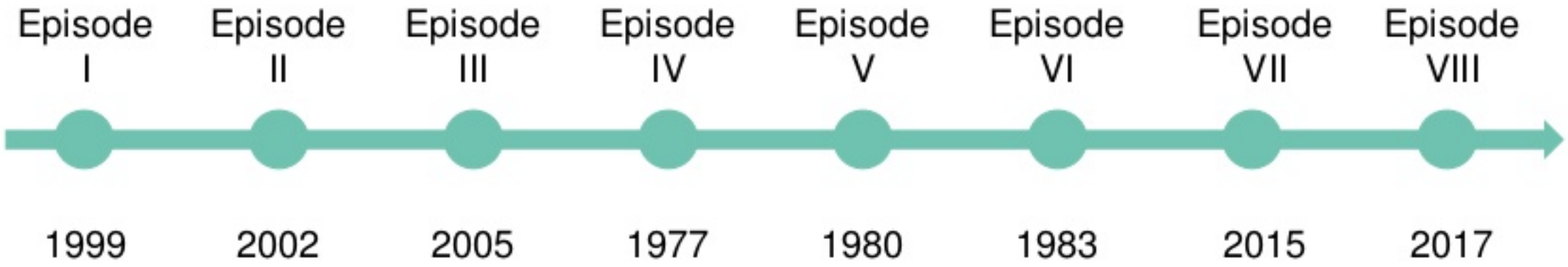
Flink becomes event-time aware



 |

Flink becomes event-time aware

Event time



Processing time

 |

Batch processing → Stream processing

- Continuous & real-time
- Stateful & exactly once
- High throughput & low latency
- Event time

November 2015

 |

More than just analytics: ProcessFunction

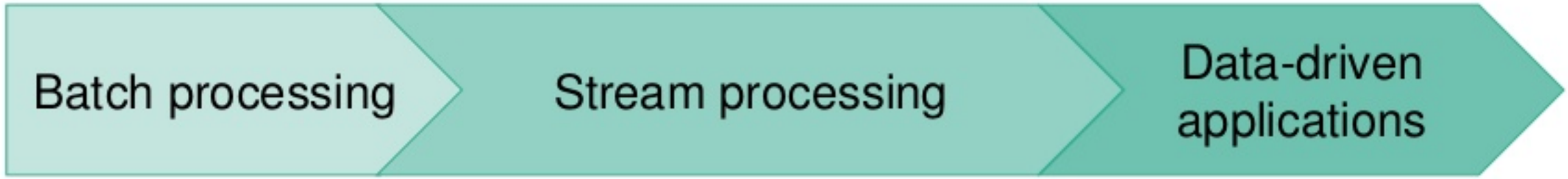
```
class MyFunction extends ProcessFunction[MyEvent, Result] {

    // declare state to use in the program
    lazy val state: ValueState[CountWithTimestamp] = getRuntimeContext().getState(…)

    def processElement(event: MyEvent, ctx: Context, out: Collector[Result]): Unit = {
        // work with event and state and schedule timers
    }

    def onTimer(timestamp: Long, ctx: OnTimerContext, out: Collector[Result]): Unit = {
        // handle callback when event-/processing- time instant is reached
    }
}
```

- ProcessFunction gives access to state, time and events
- Low level API
- Enables data-driven applications

D_TRB

THE SOCIAL NETWORK FOR PETROLHEADS

 |



- Continuous & real-time
- Stateful & exactly once
- High throughput & low latency
- Event time

February 2017

 |

Present & Future

Present in a nutshell

Hardening

Faster network stack

Application level flow control

Resolving dependency hell

Scaling

Incremental snapshots

Local recovery

Scalable timers

Interoperability

Resource elasticity

REST client-server interface

Container entrypoint

Stream SQL

SQL client

User-defined functions

More powerful joins

Misc

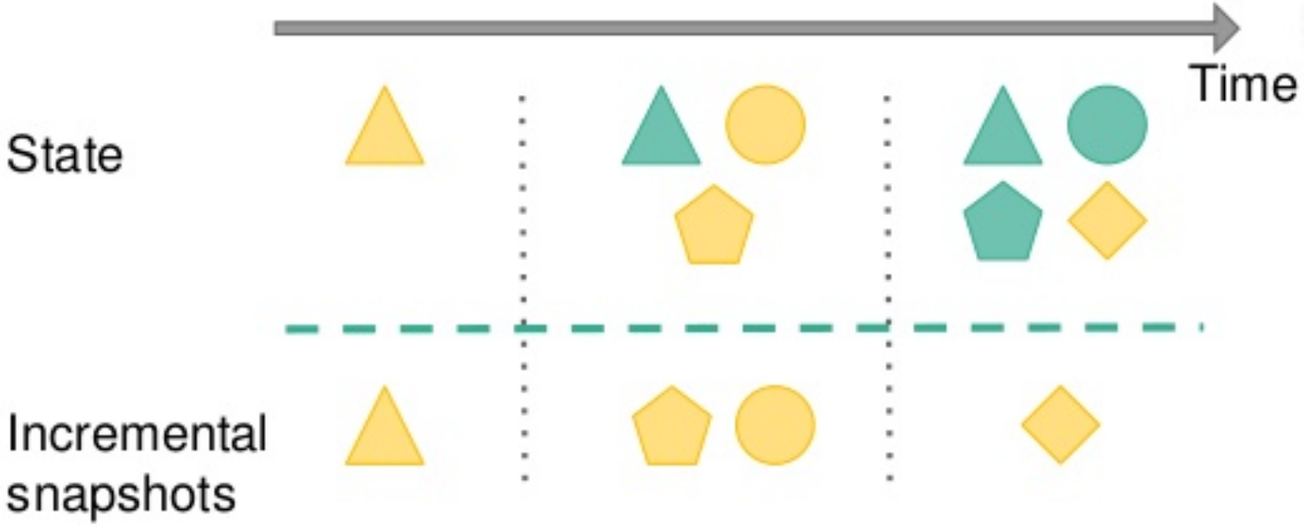
State TTL

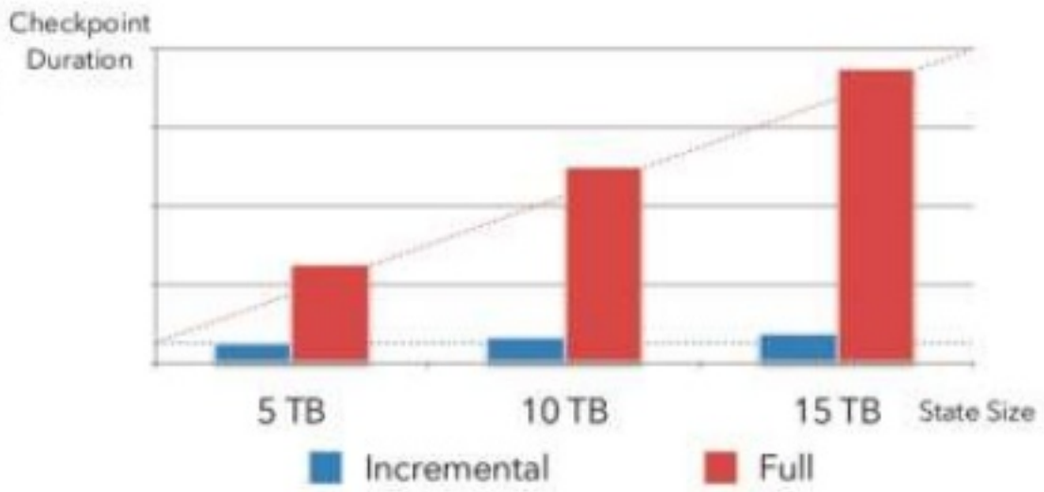
Broadcast state

Kafka exactly-once producer

 |

Large, larger, Flink

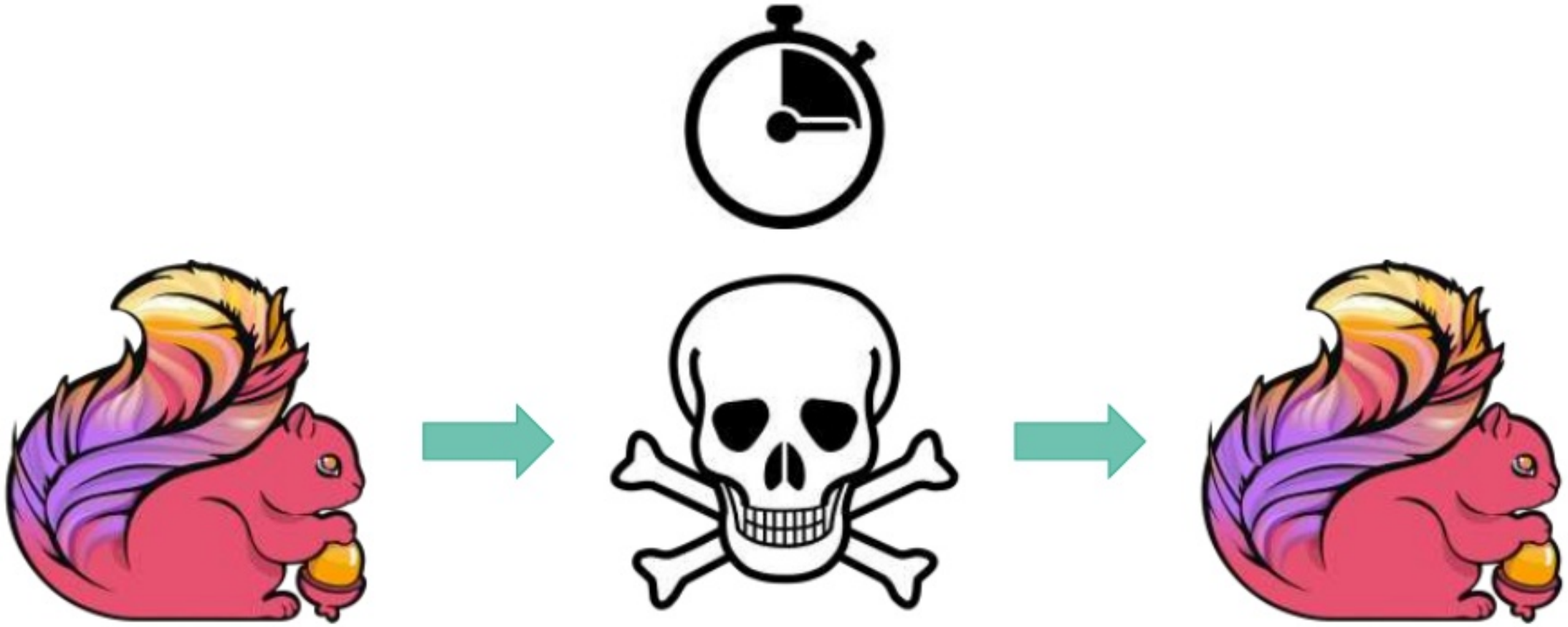




- Snapshot only state diff
- Incremental snapshots allow to handle very large state

 |

Faster failover is always better



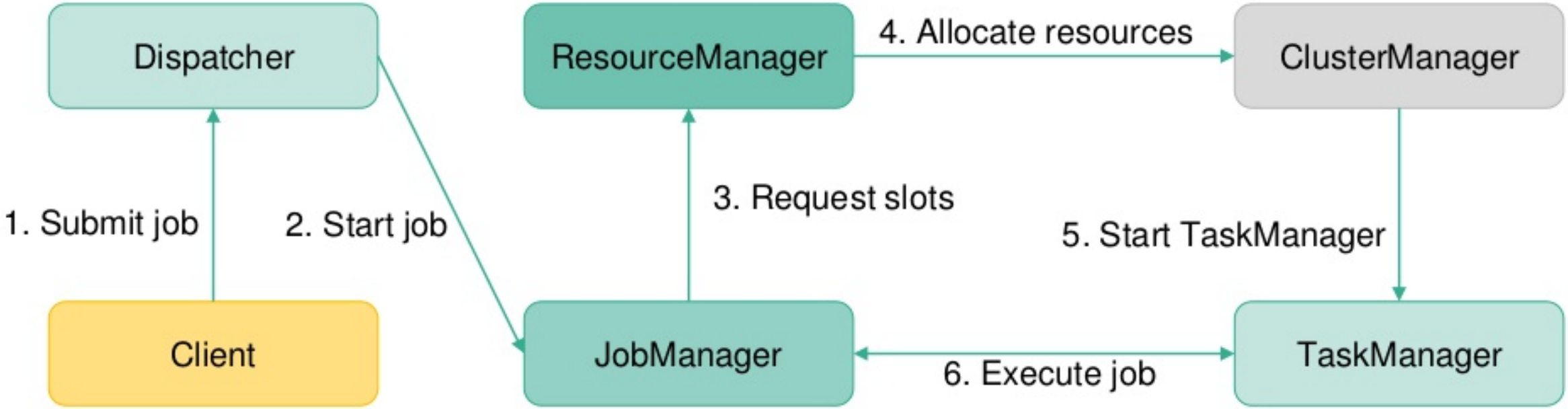
 |

Varying workloads

- Violating SLAs vs. wasting money
- Varying workloads require to adapt resources

 |

Revamped distributed architecture



- Support for full resource elasticity
- Application parallelism can be dynamically changed

 |

- Continuous & real-time
- Stateful & exactly once
- High throughput & low latency
- Event time

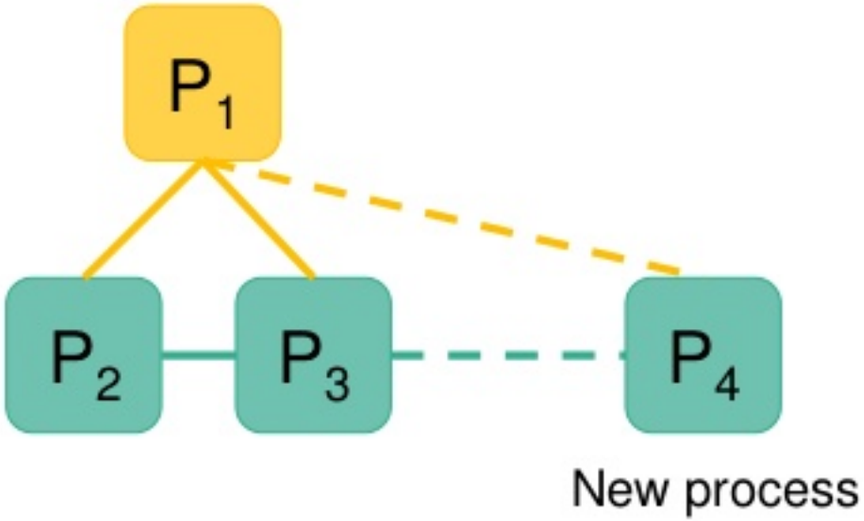
- Applications as first class citizens

Present & Future

 |

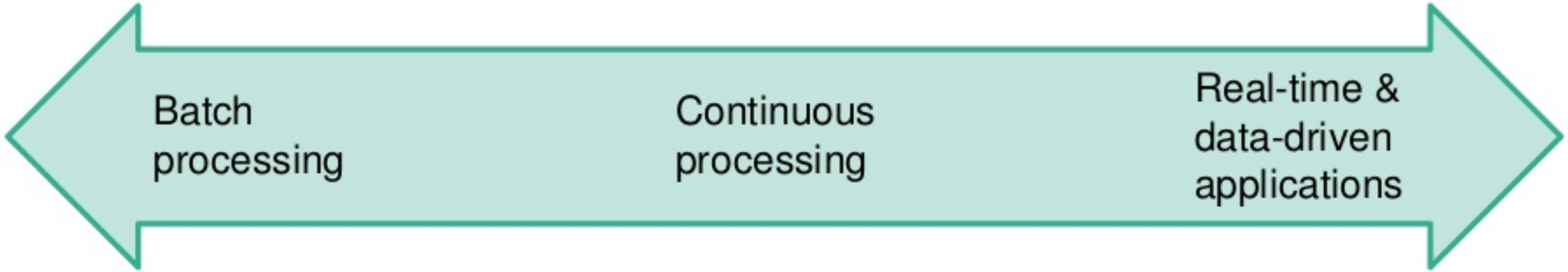
Flink as a library (and still as a framework)

- Deploying Flink applications should be as easy as starting a process
- Bundle application code and Flink into a single image
- Process connects to other application processes and figures out its role
- Removing the cluster out of the equation



 |

How much control do I need?



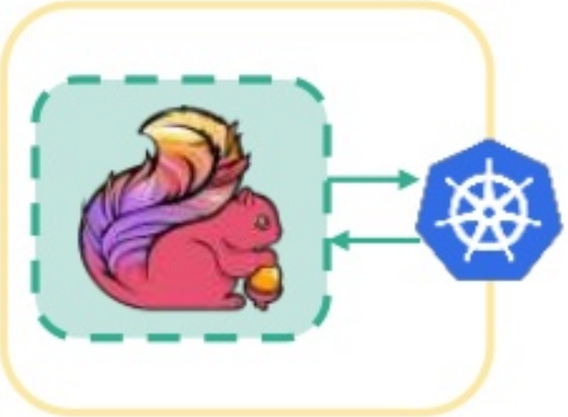
- Multiple short lived stages
- Different resource requirements per stage
- Efficient execution requires control over resources
- Flink allocates actively resources

- Continuously processing operators
- Constrained by external systems, SLAs and application logic
- External system can assign resources
- Flink reacts to available resources

 |

Active vs. reactive mode

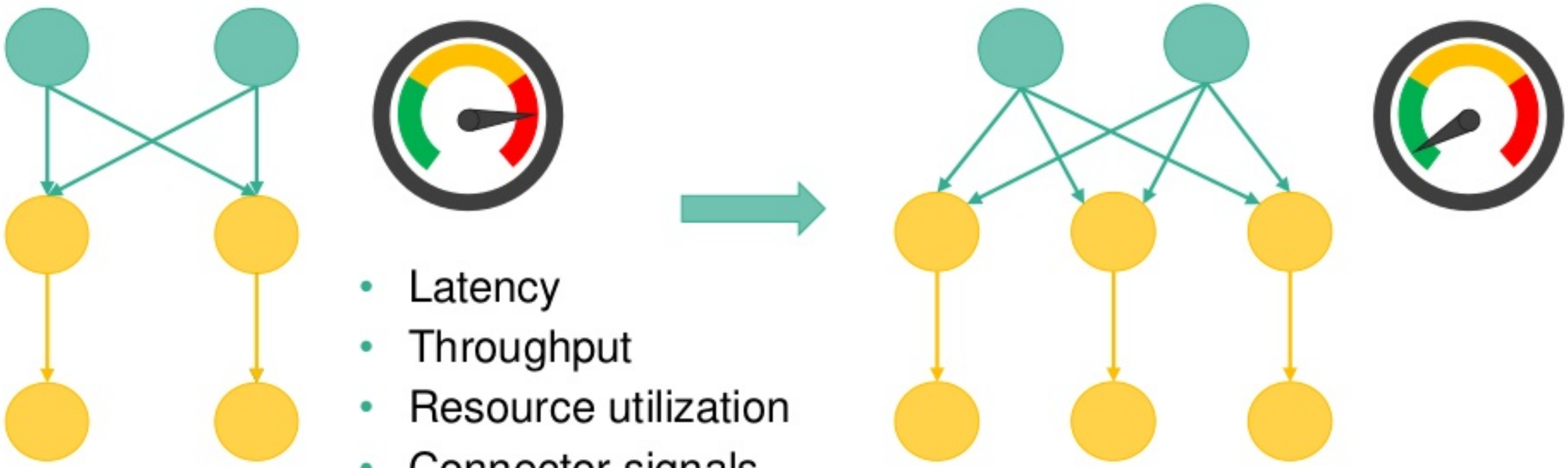
- Active mode
  - Flink is aware of underlying cluster framework
  - Flink allocate resources
  - E.g. existing YARN and Mesos integration



- Reactive mode
  - Flink is oblivious to its runtime environment
  - External system allocates and releases resources
  - Flink scales with respect to available resources
  - Relevant for environments: Kubernetes, Docker, as a library

 |

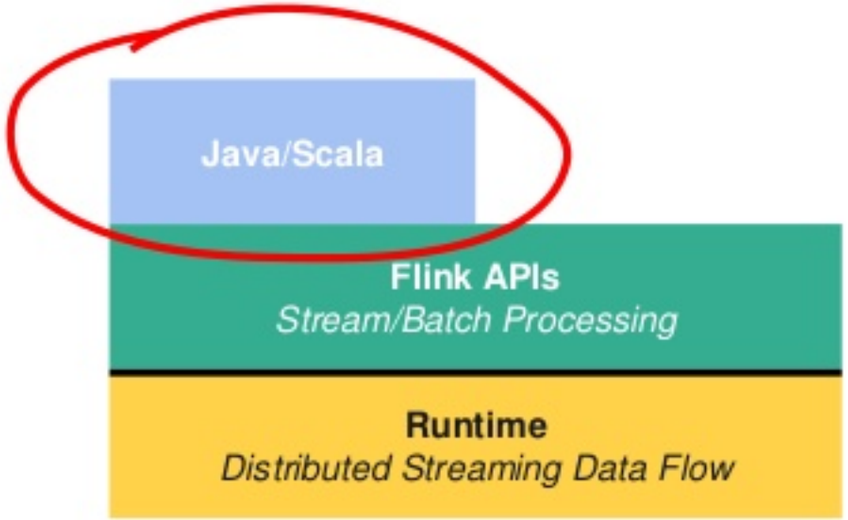
Scaling automatically

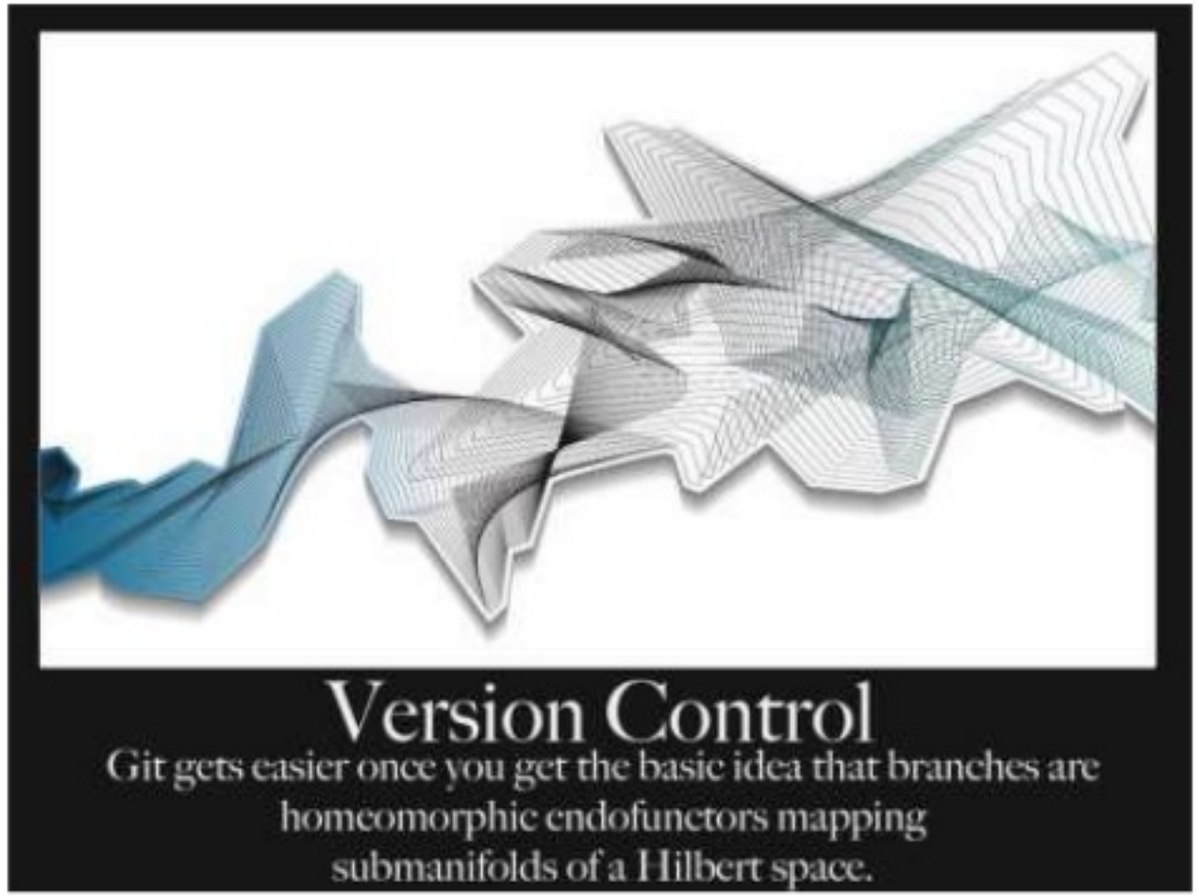


- Latency
- Throughput
- Resource utilization
- Connector signals

 |

How we create Flink Jobs

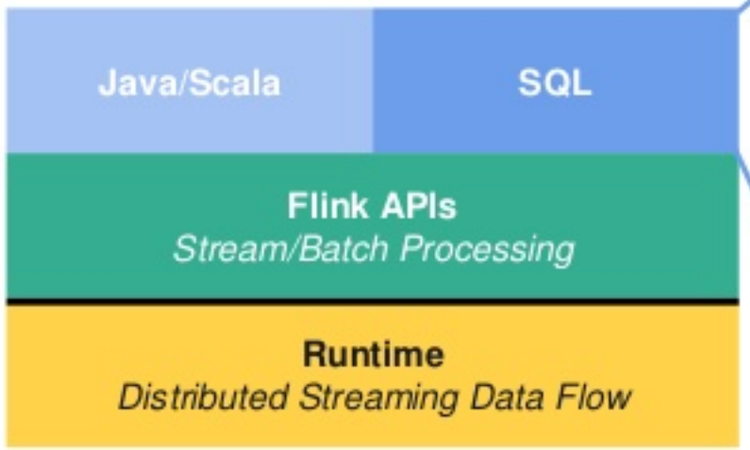




 |

Flink SQL

*since Flink 0.9.0 (June 2015)



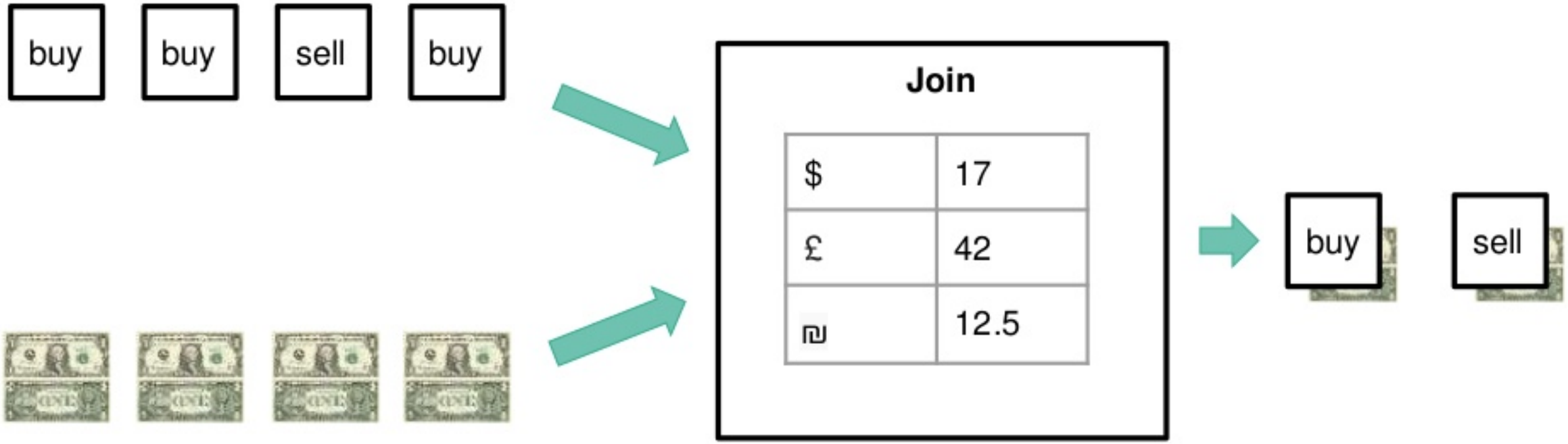
- SQL commandline
- Event time and processing time
- Configuration in YAML
- Streaming and Batch
- Source/Sink definition in YAML
- User-defined functions

"NO CODING REQUIRED"

 |

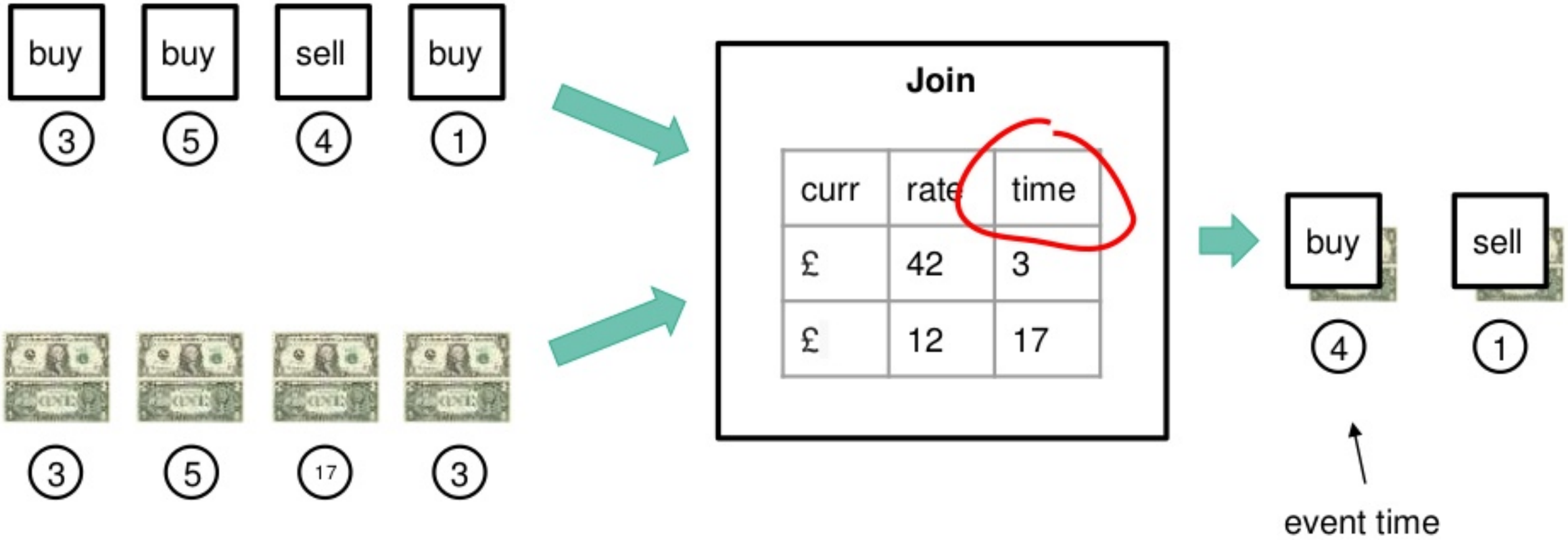
1. fish /Users/aljoscha/Dev/sql-demo (fish)

“Join” me for some trading



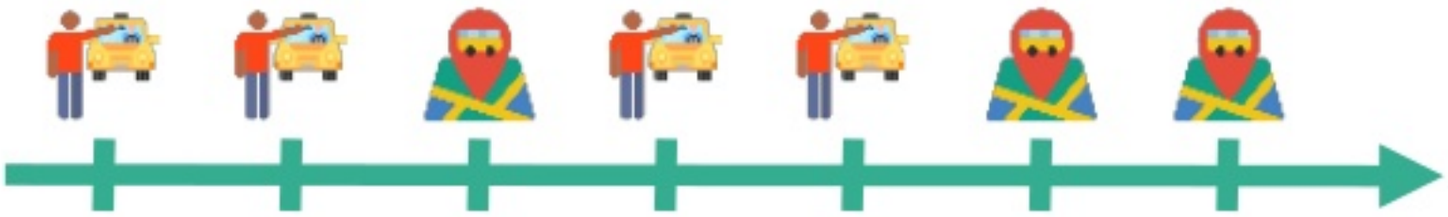
 |

Introducing Time-versioned Table Joins



 |

SQL for pattern analysis?



```
SELECT * from ?
```

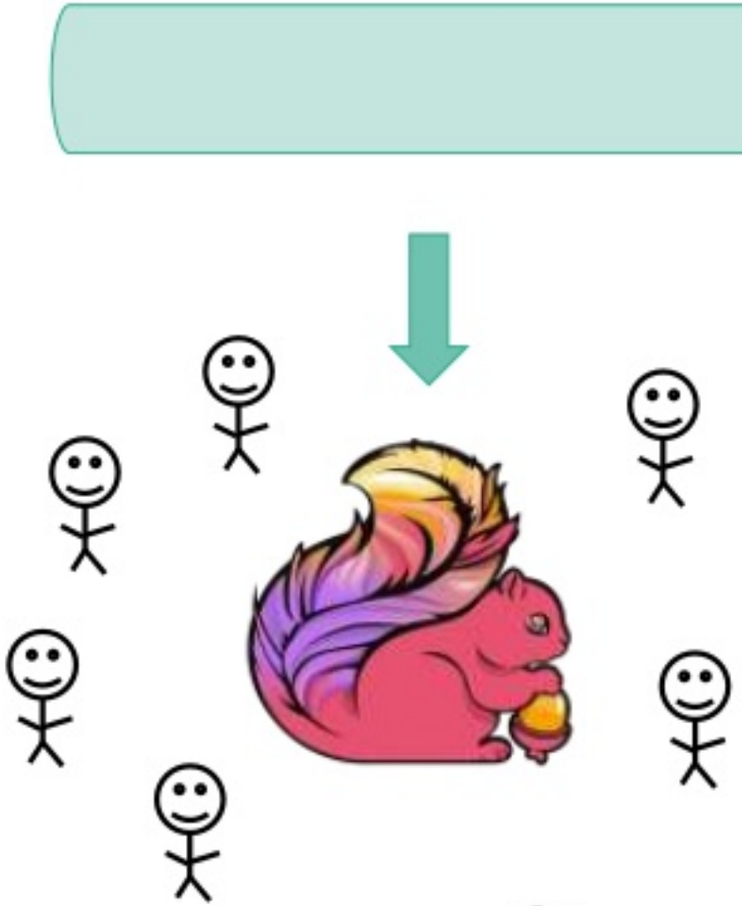
 |

Introducing MATCH_RECOGNIZE

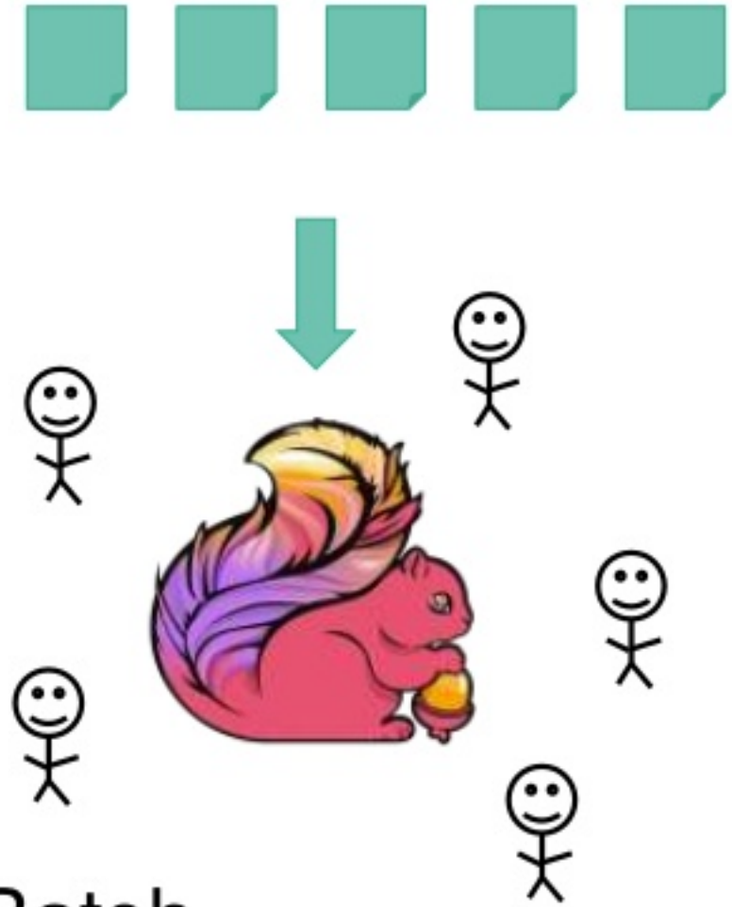
```
SELECT *
FROM TaxiRides
MATCH_RECOGNIZE (
    PARTITION BY driverId
    ORDER BY rideTime
    MEASURES
        S.rideId as sRideId
    AFTER MATCH SKIP PAST LAST ROW
    PATTERN (S M{2,} E)
    DEFINE
        S AS S.isStart = true,
        M AS M.rideId <> S.rideId,
        E AS E.isStart = false
            AND E.rideId = S.rideId
)
```

 |

Todays processing landscape



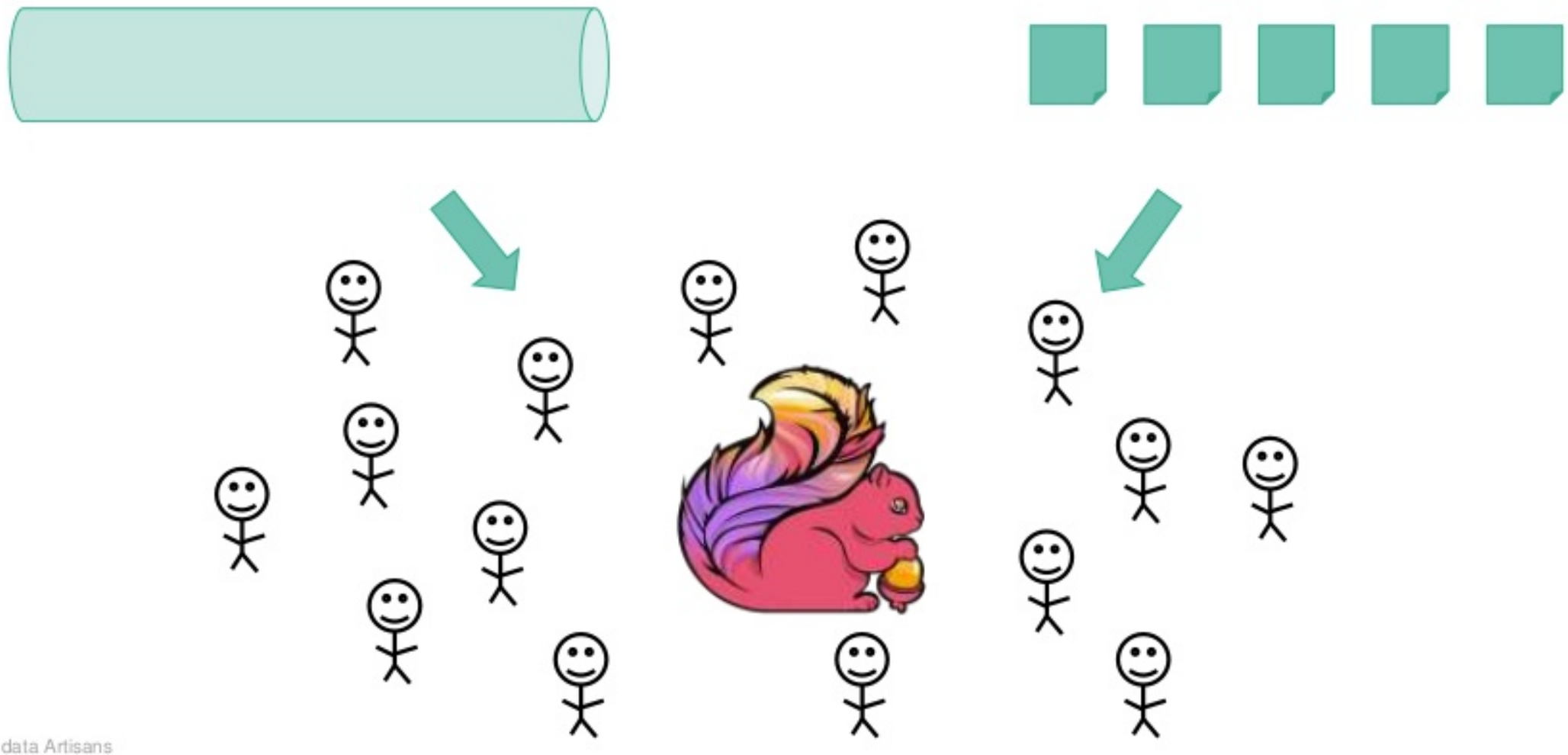
Streaming



Batch

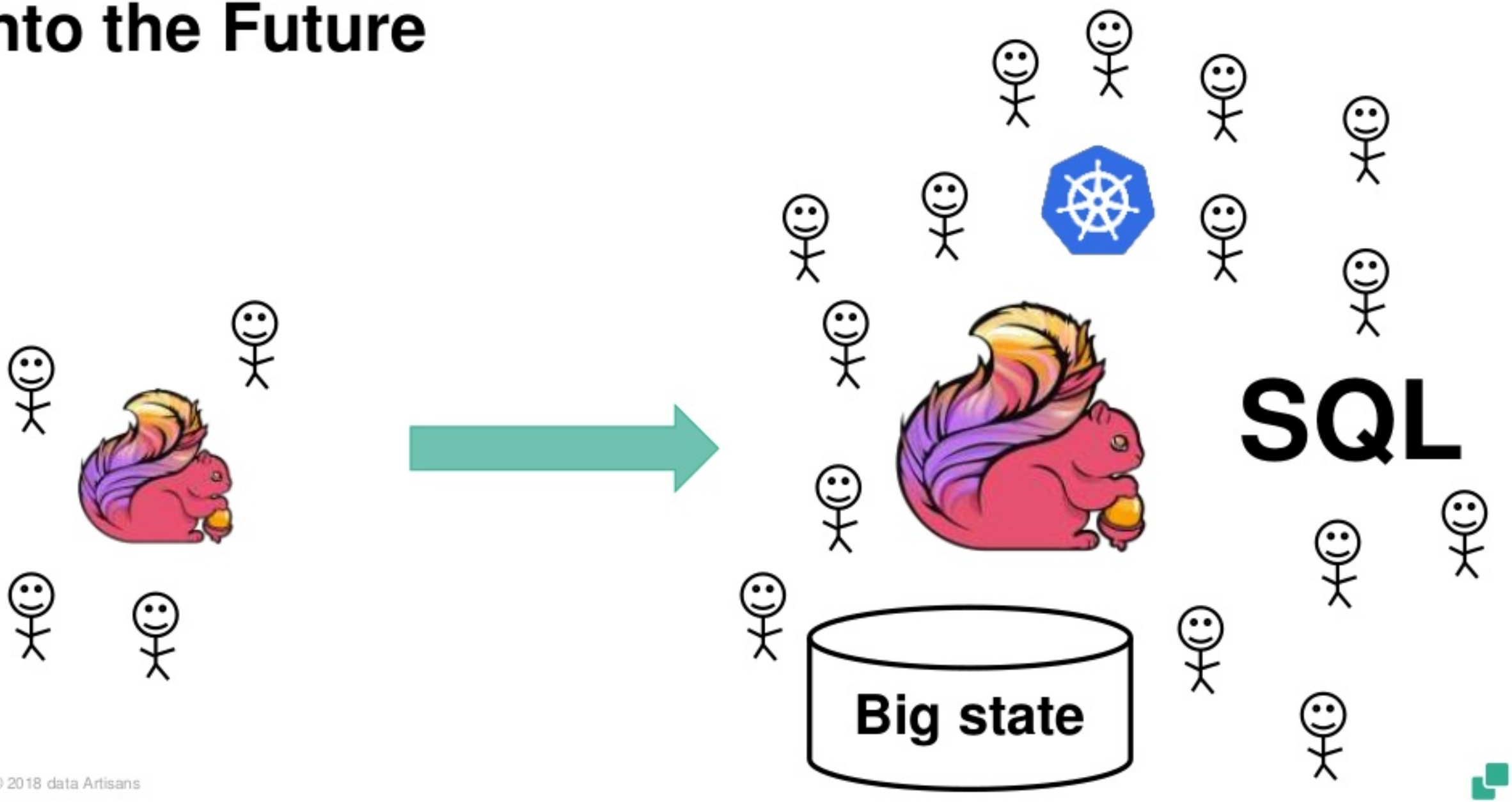
 |

Batch/streaming unification



 |

Into the Future



 |

Thank s!