From Big Data to Fast Data: Efficient Stream Data Management

Alexandru Costan

HDR Defense, ENS Rennes, March 14, 2019

Big Data

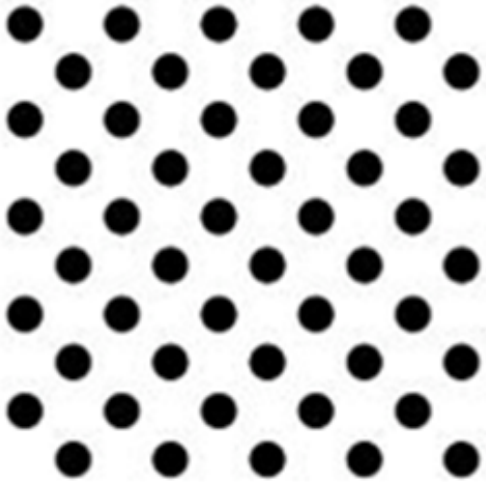
2011

2017

2018

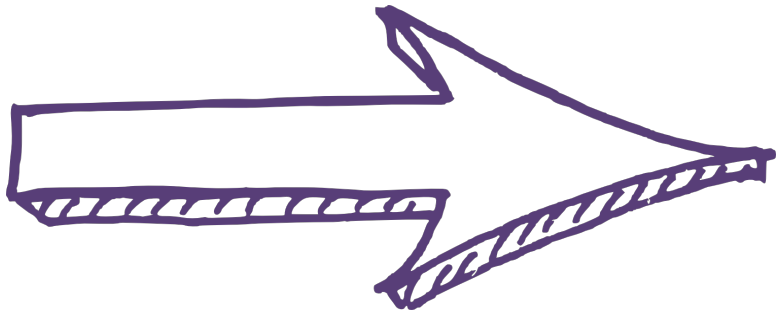
From Big Data to Fast Data

Volume

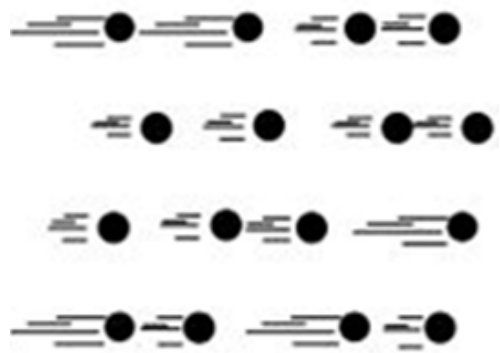


Data at rest

Stationary
Static

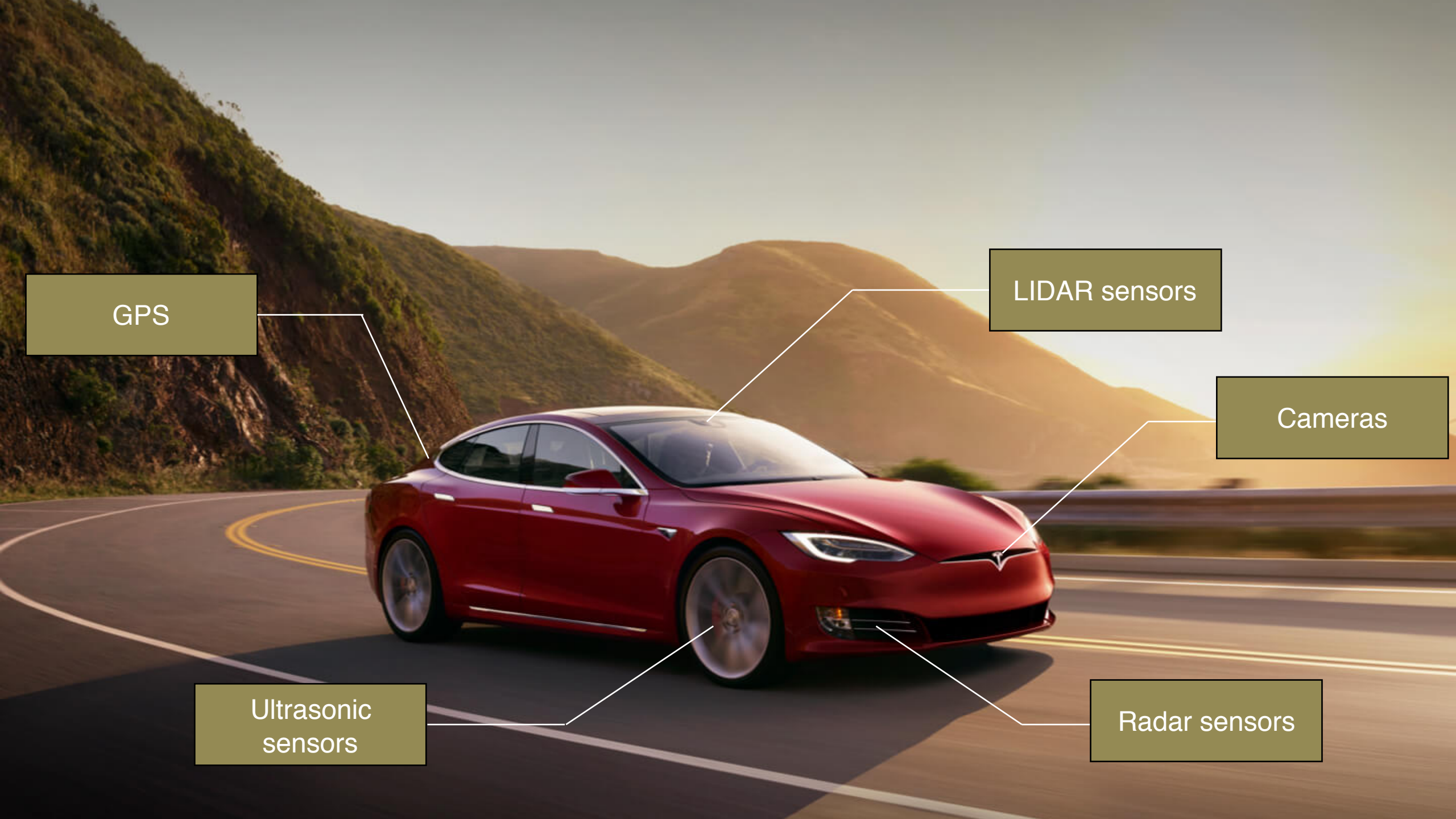


Velocity



Data in motion

Fluid
Dynamic


GPS
LIDAR sensors
Cameras
Ultrasonic sensors
Radar sensors

Sensor type	Quantity	Data generated
Radar	4–6	0.1–15 Mbit/s
LIDAR	1–5	20–100 Mbit/s
Camera	6–12	500–3,500 Mbit/s
Ultrasonic	8–16	<0.01 Mbit/s
Vehicle motion, GNSS, IMU	-	<0.1 Mbit/s

TOTAL ESTIMATED BANDWIDTH

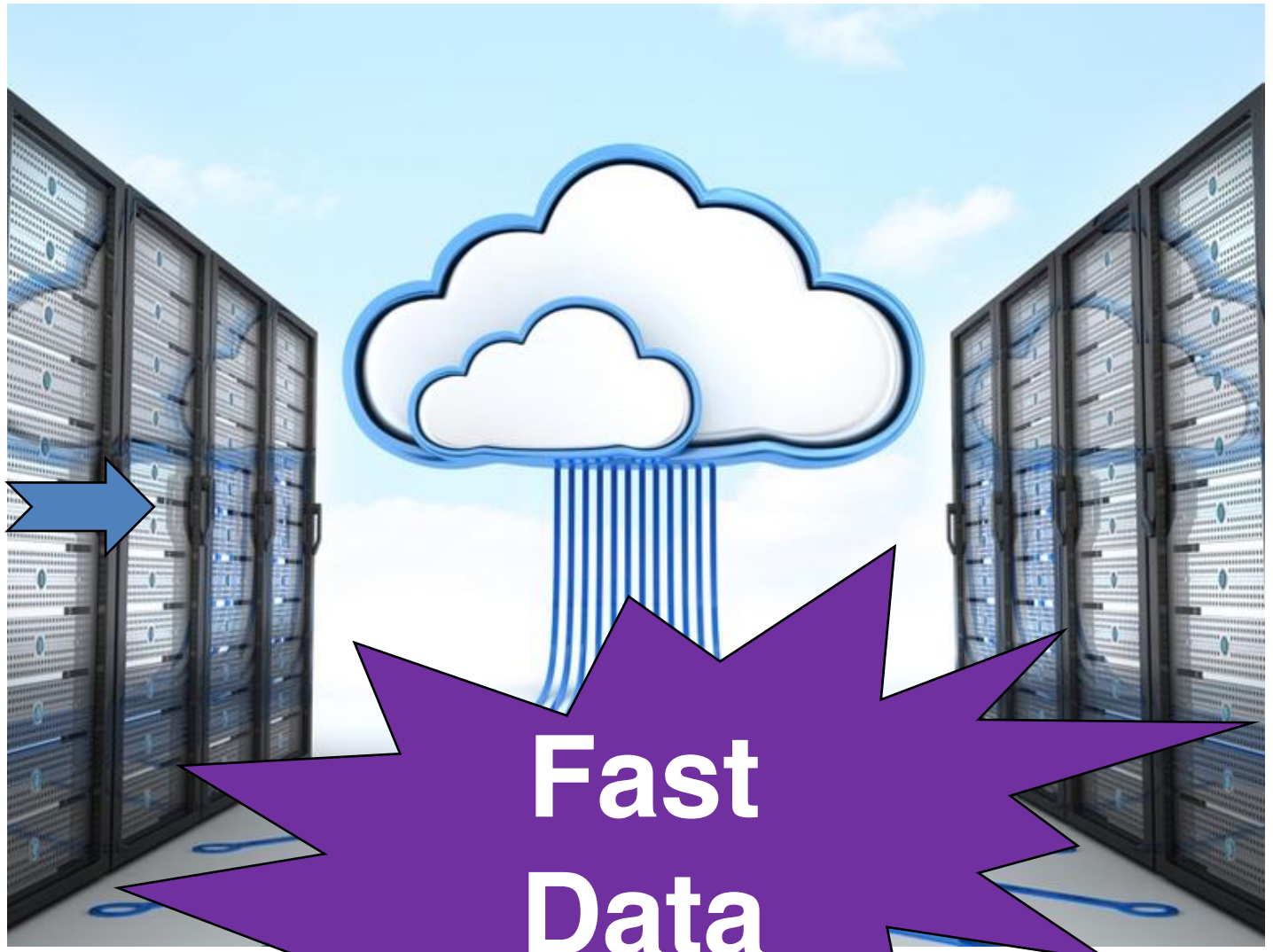
3 Gbit/s (~1.4TB/h) to 40 Gbit/s (~19 TB/h)

Cars

Devices

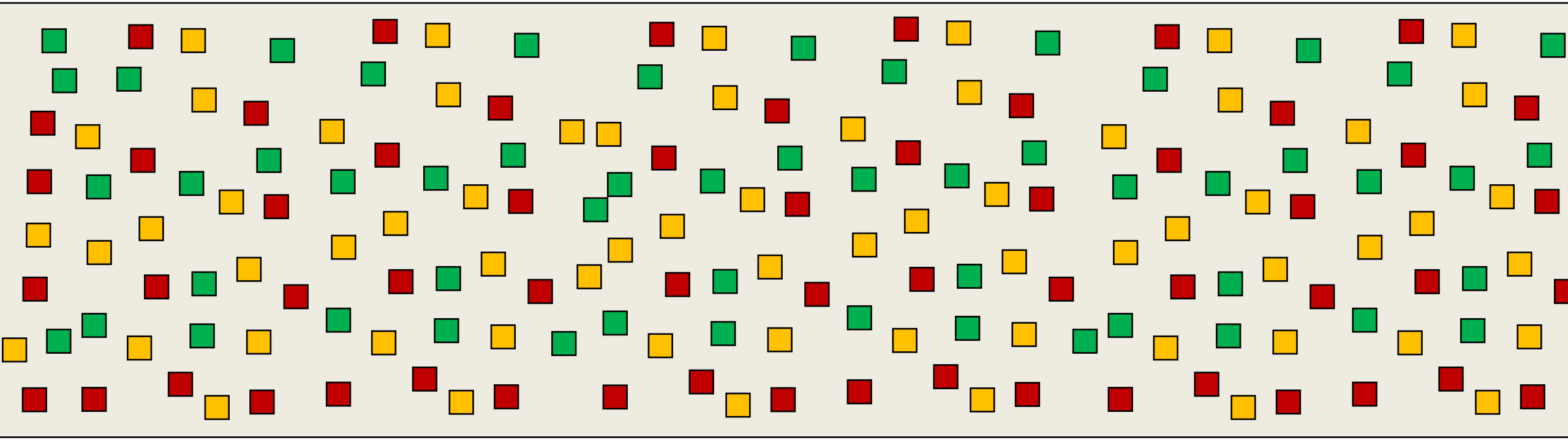
IoT and Smart City

SMALL and MANY



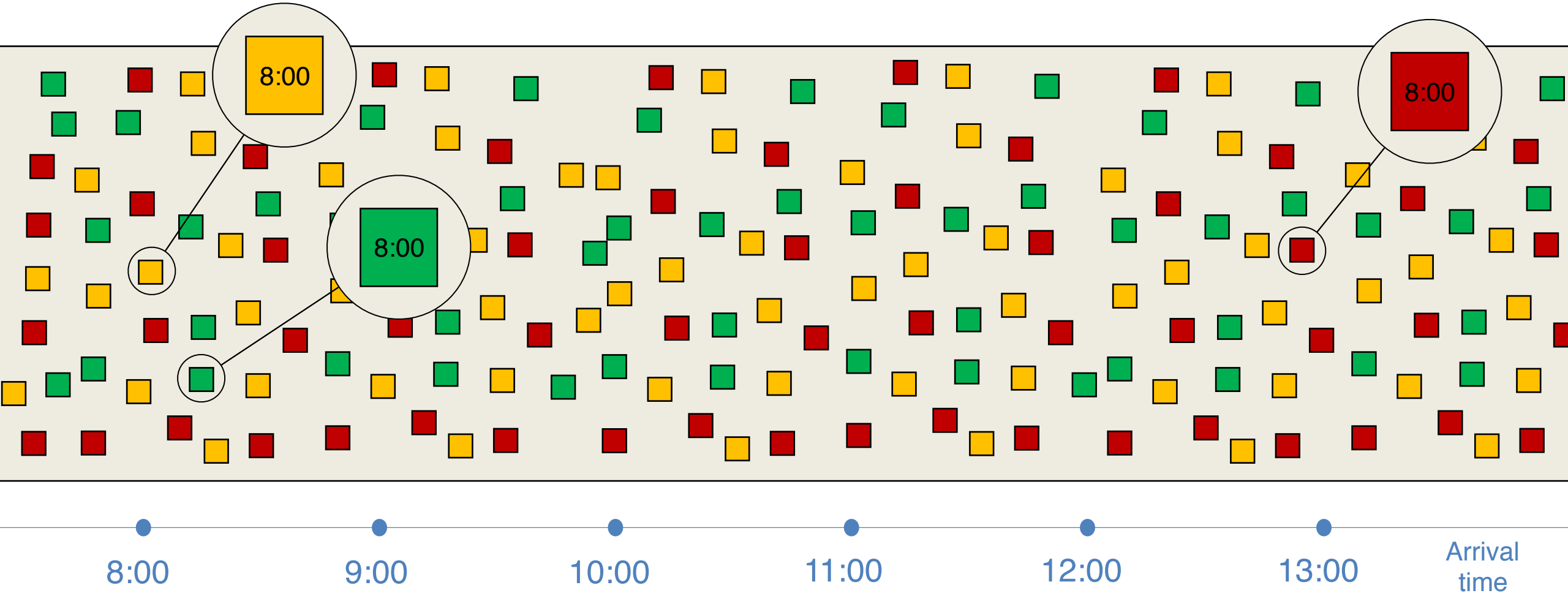
Fast Data

Streams: the model for Fast Data



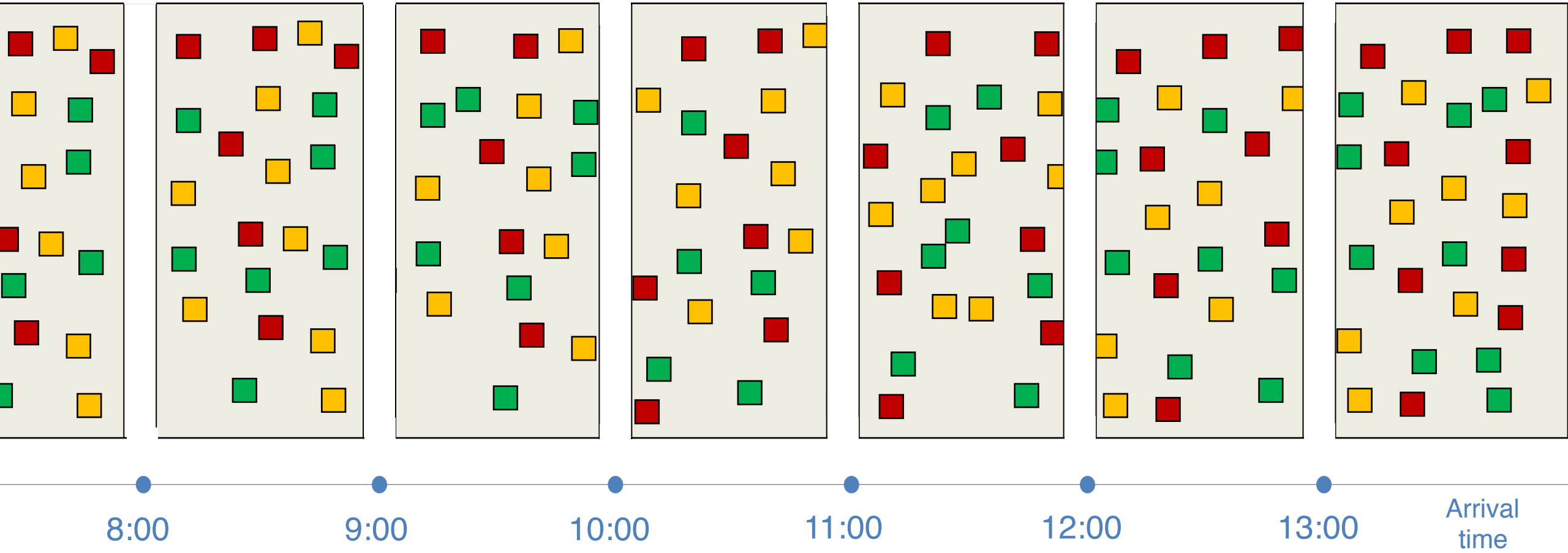
- Continuous, unbounded, unordered, global-scale datasets made up of events
- Small size per event (*i.e.,* bytes and kilobytes)
- High arrival rate (*i.e.,* million items per second)

Streams: the model for Fast Data



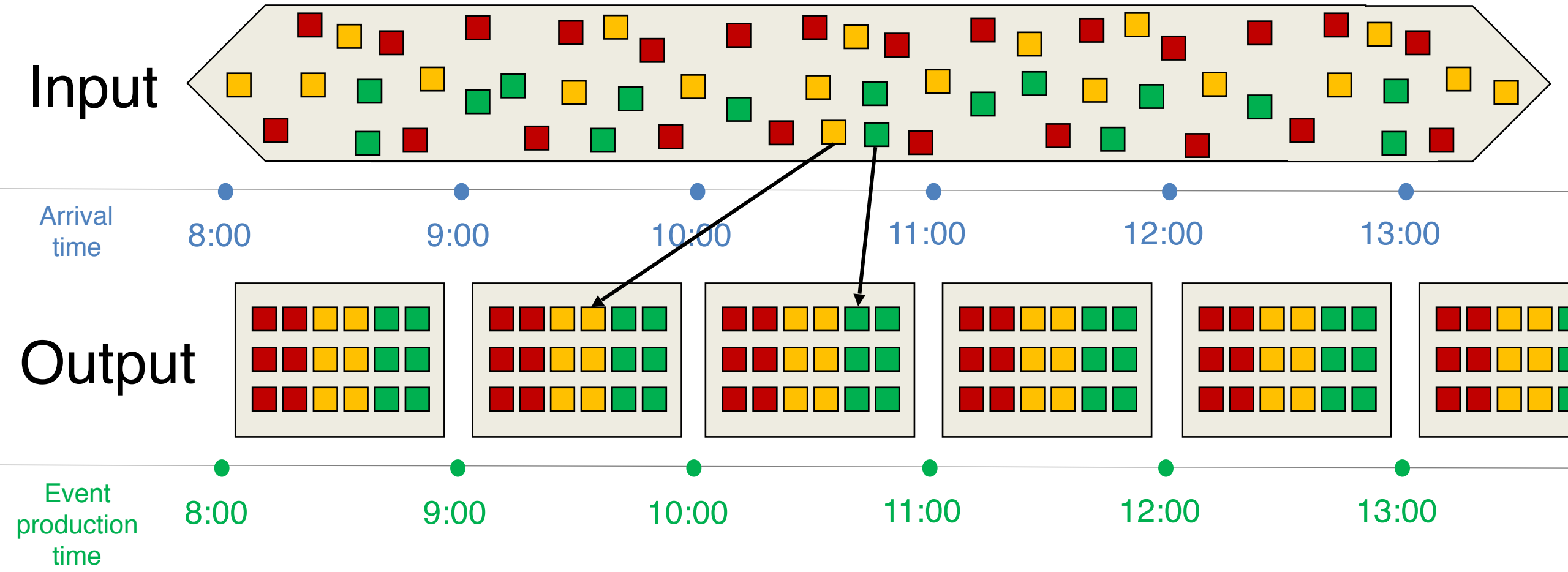
Events arrive with unknown delays

How to deal with this unboundedness ?



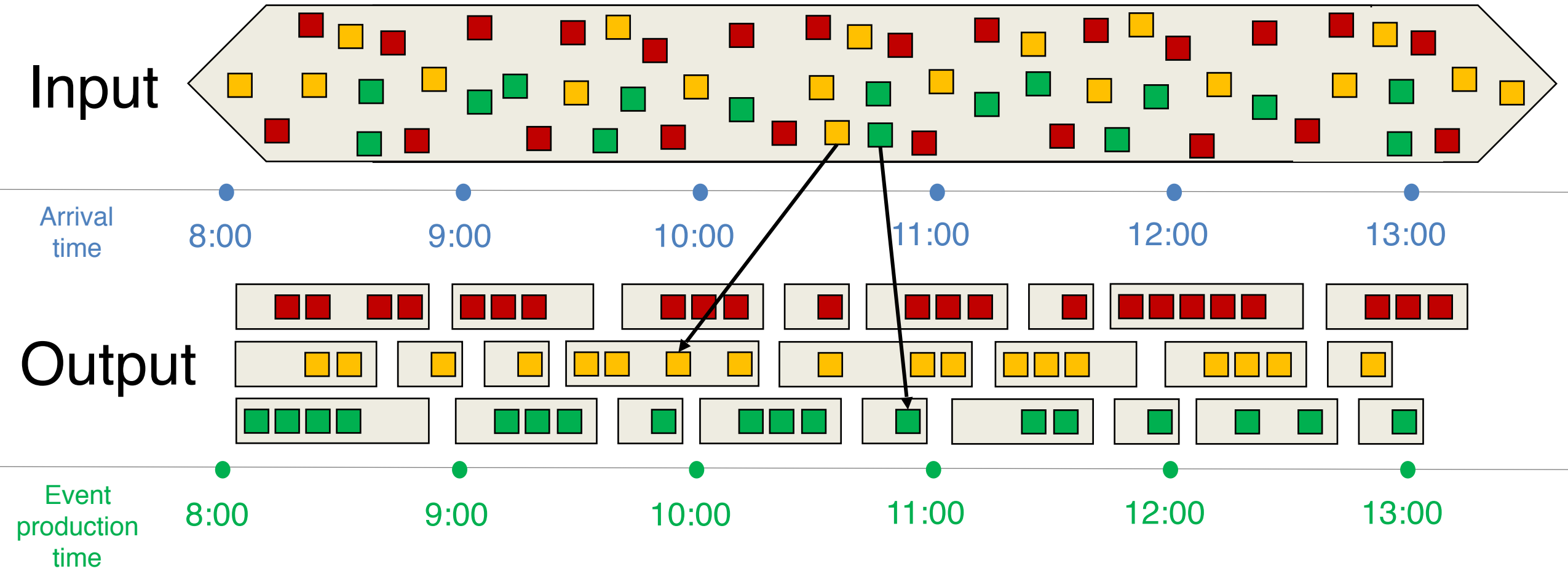
Aggregating **time-based windows**

How to deal with this unboundedness ?



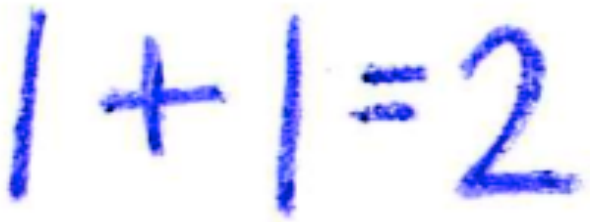
Aggregating **event-based windows**

How to deal with this unboundedness ?



Aggregating **session-based windows**

Batch vs. streaming



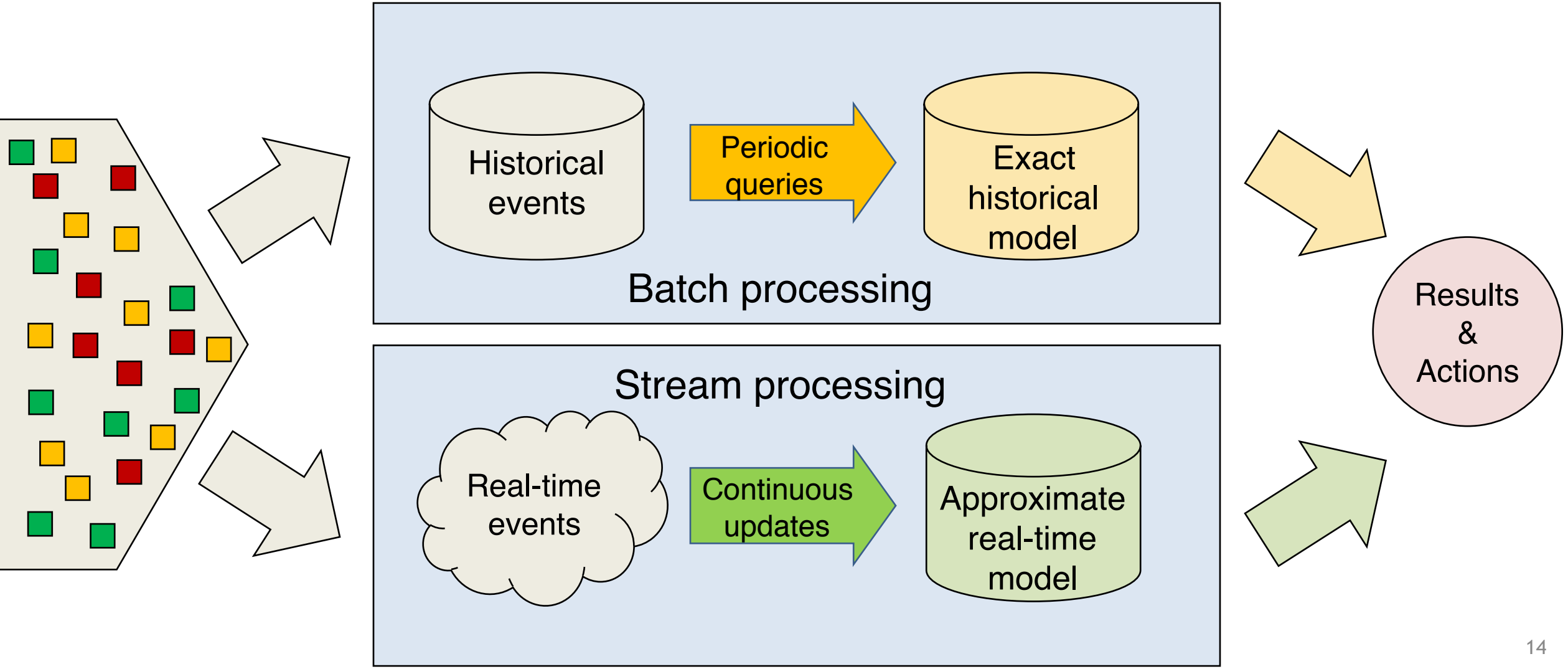
Correctness

Latency

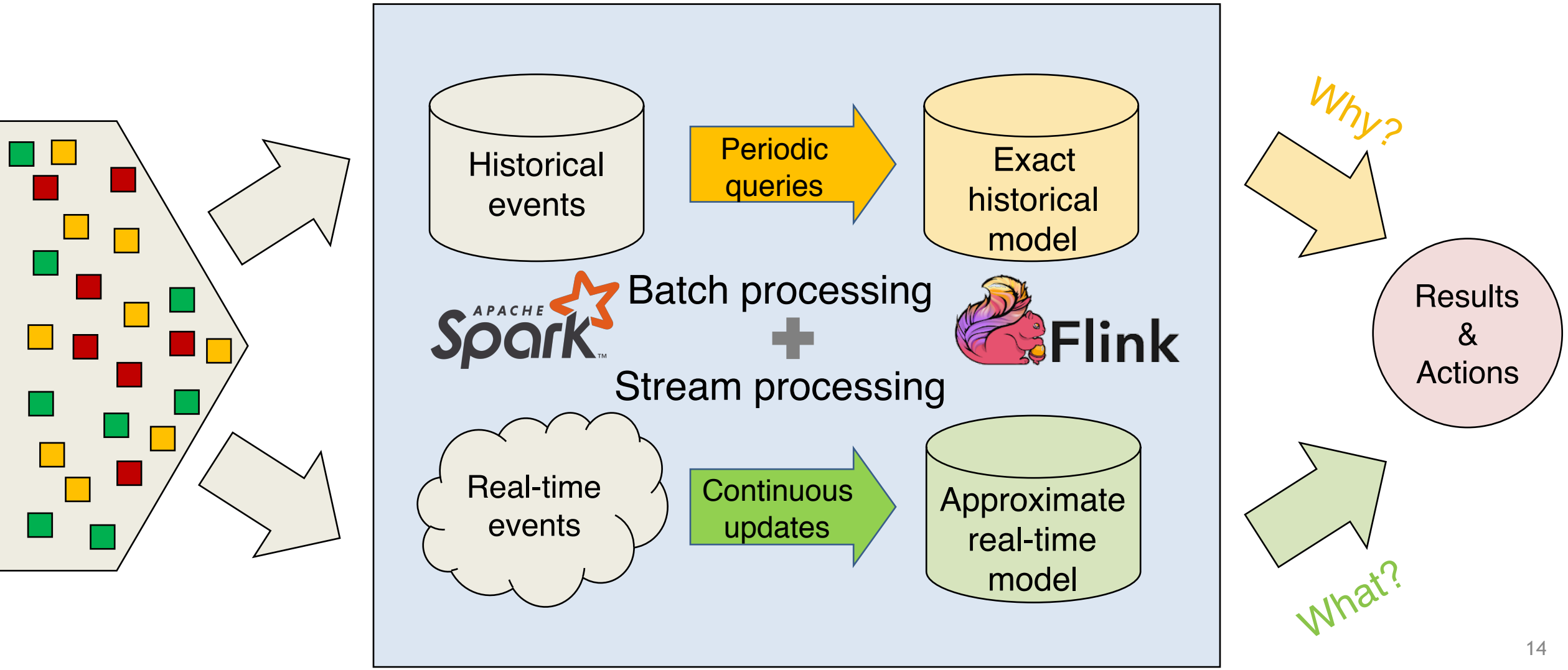
Cost

	Batch	*Streaming*
		
Correctness	Exact results	Approximate results
Latency	High-latency	Low-latency
Cost	Stateless	Stateful

State of the art until recently: Lambda Architectures

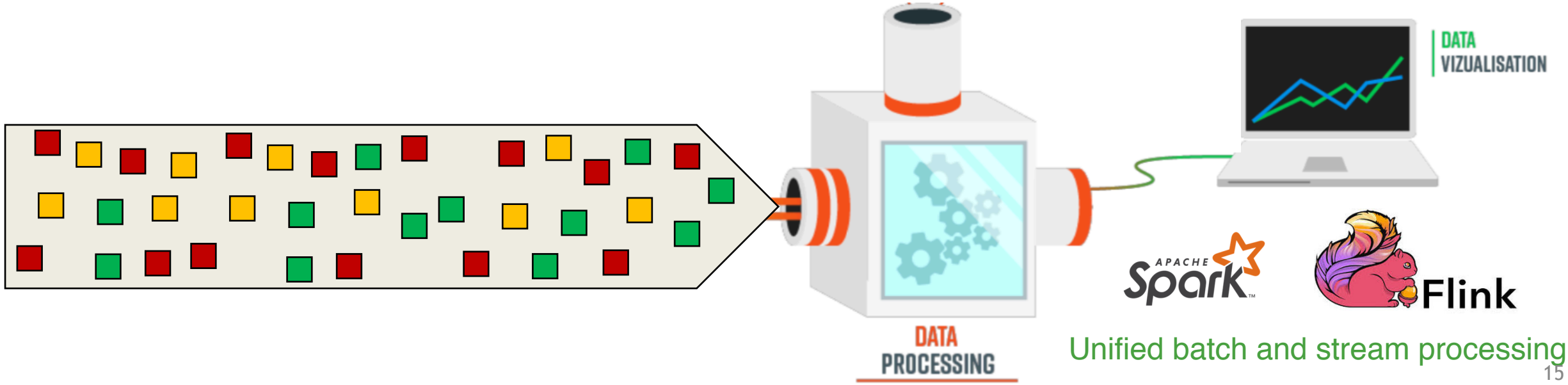


State of the art until recently: Lambda Architectures



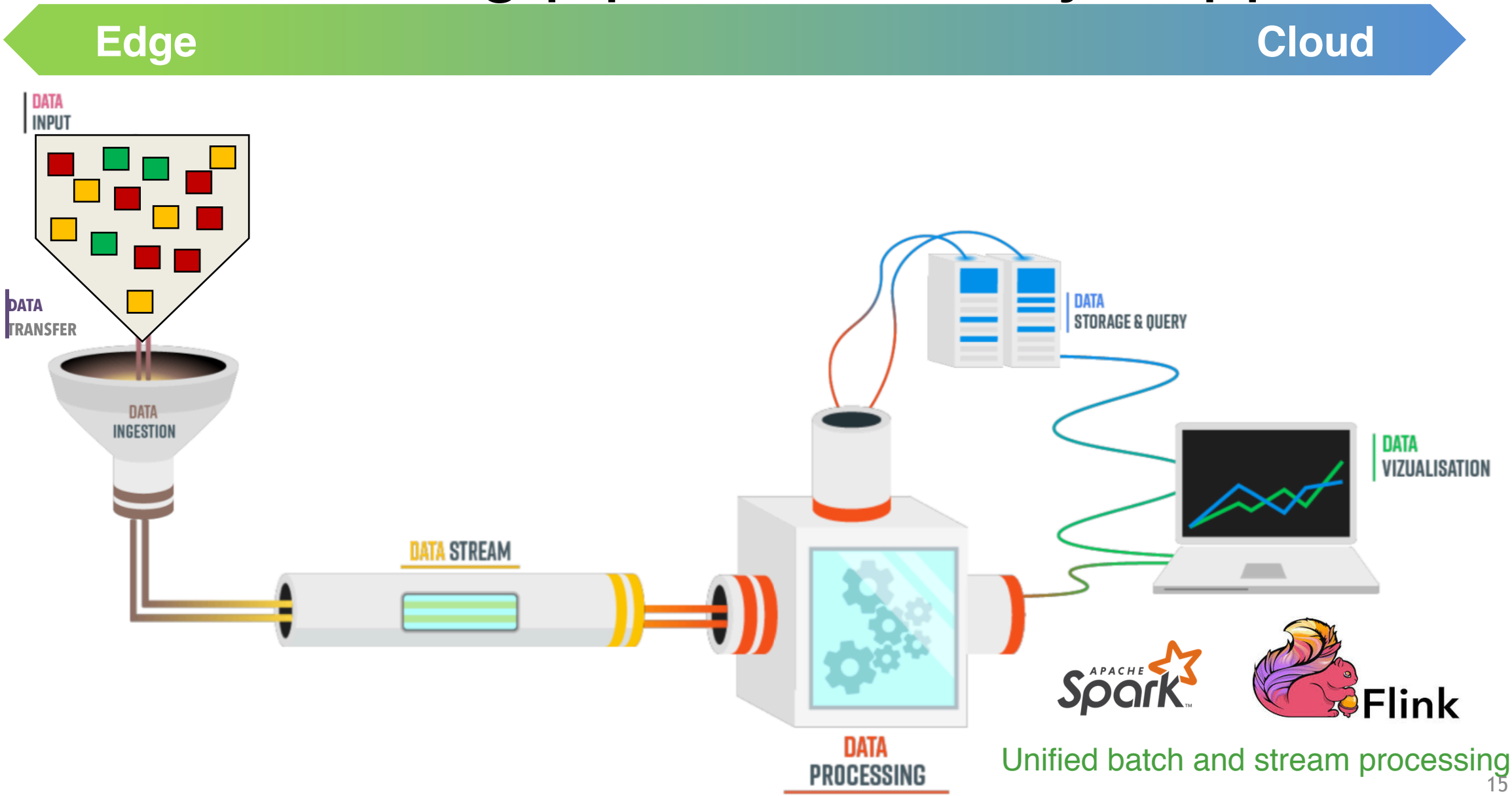
The streaming pipeline: latency happens

Cloud



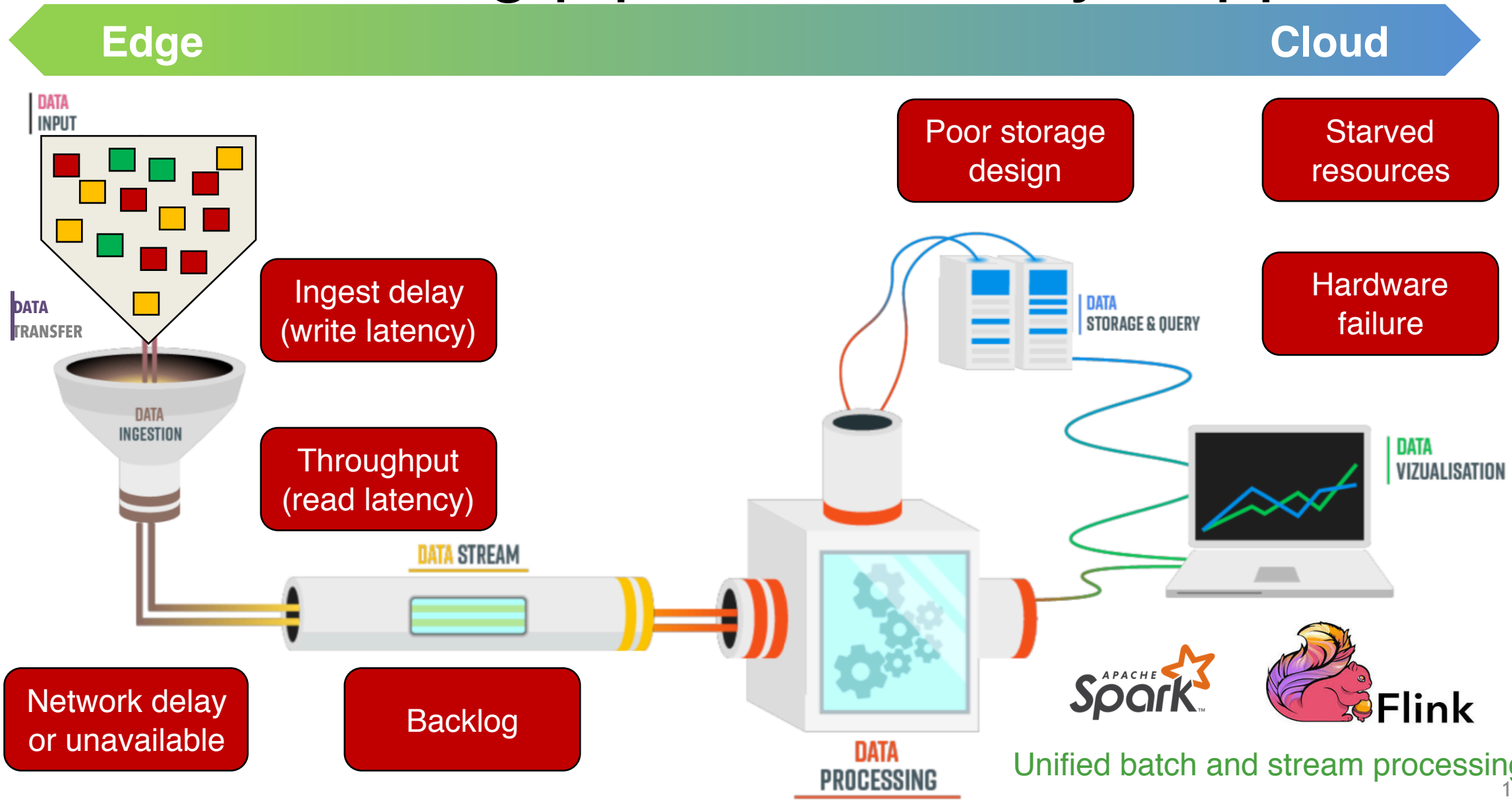
Unified batch and stream processing

The streaming pipeline: latency happens

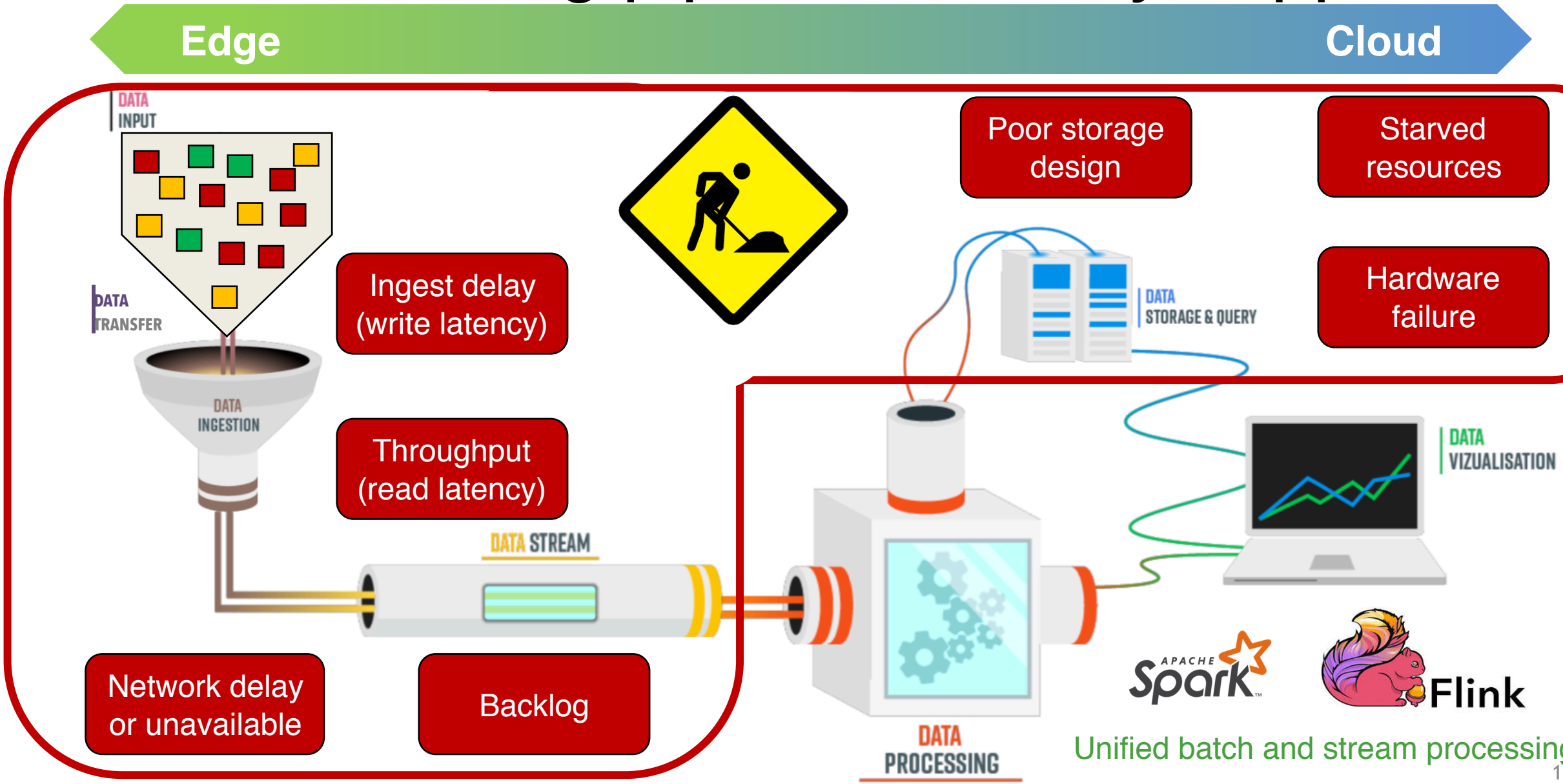


Unified batch and stream processing

The streaming pipeline: latency happens

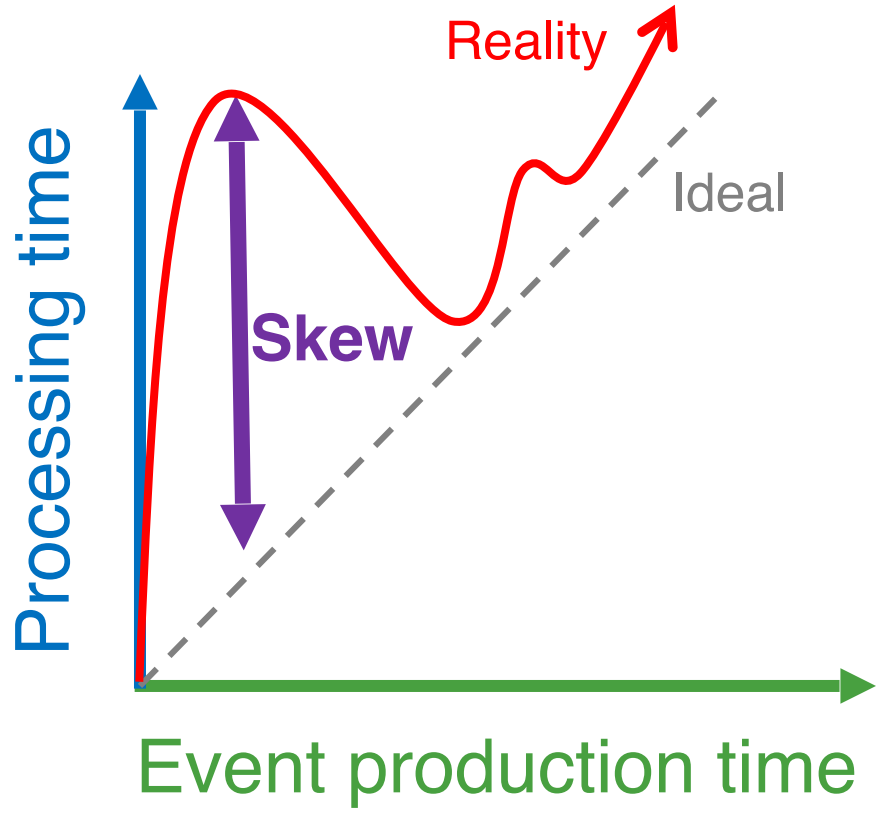


The streaming pipeline: latency happens

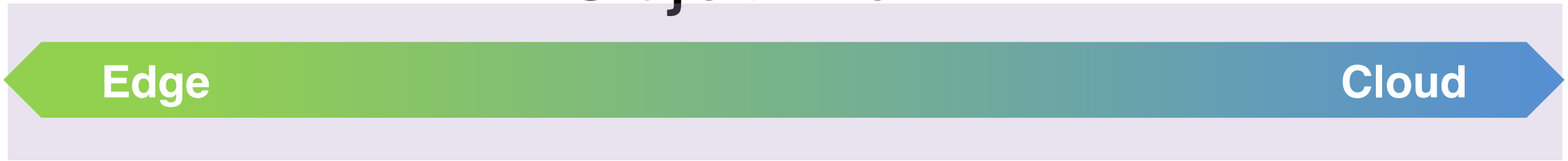


Objective

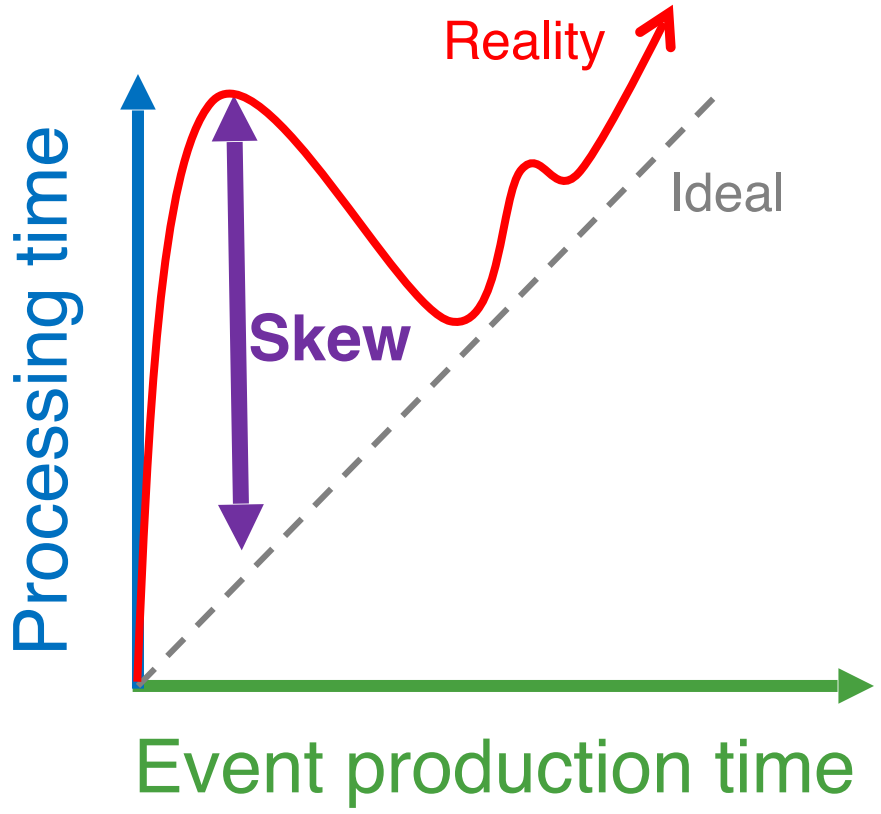

Edge
Cloud


Reality
Ideal
Skew
Processing time
Event production time

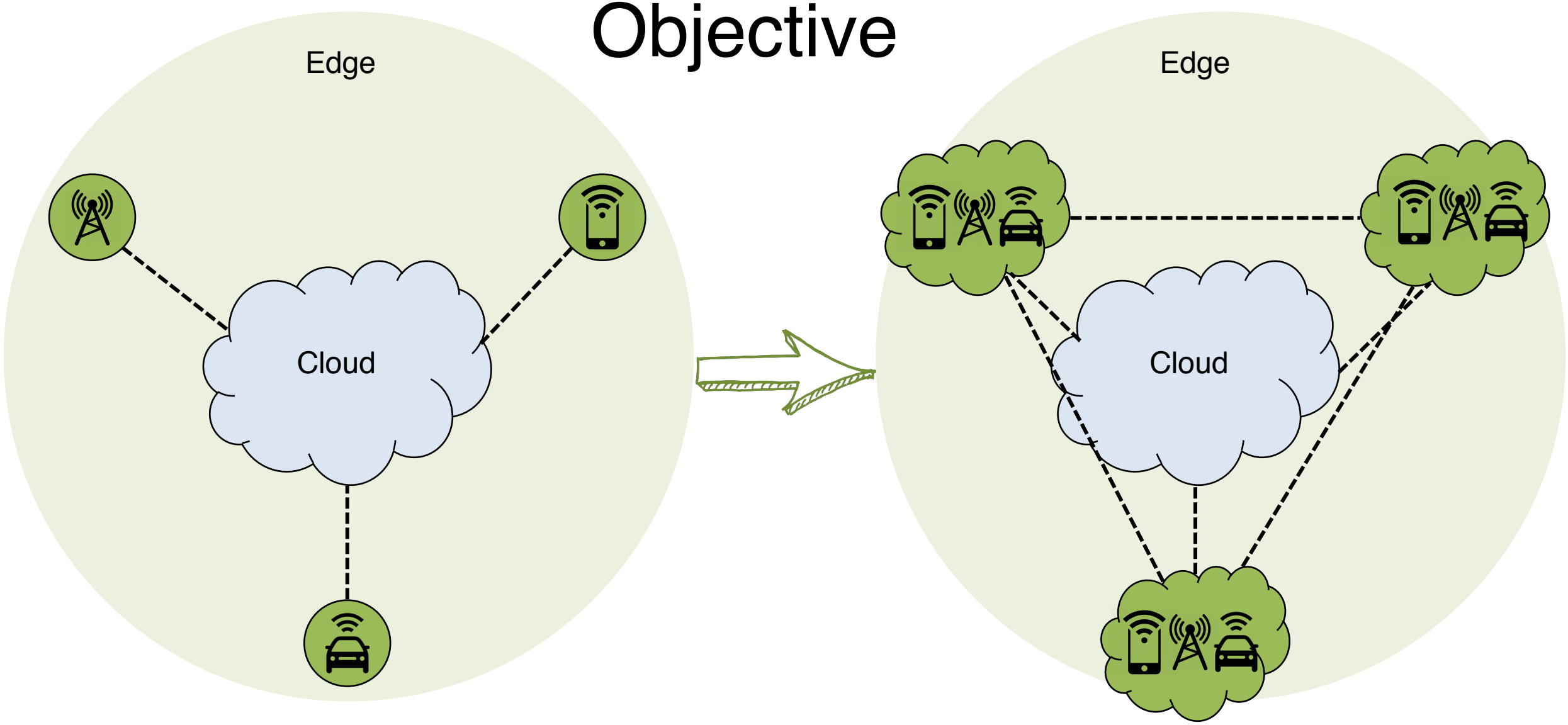
Objective



Reduce the processing time **skew** by means of **dedicated stream data management across Edge and Cloud**



Objective

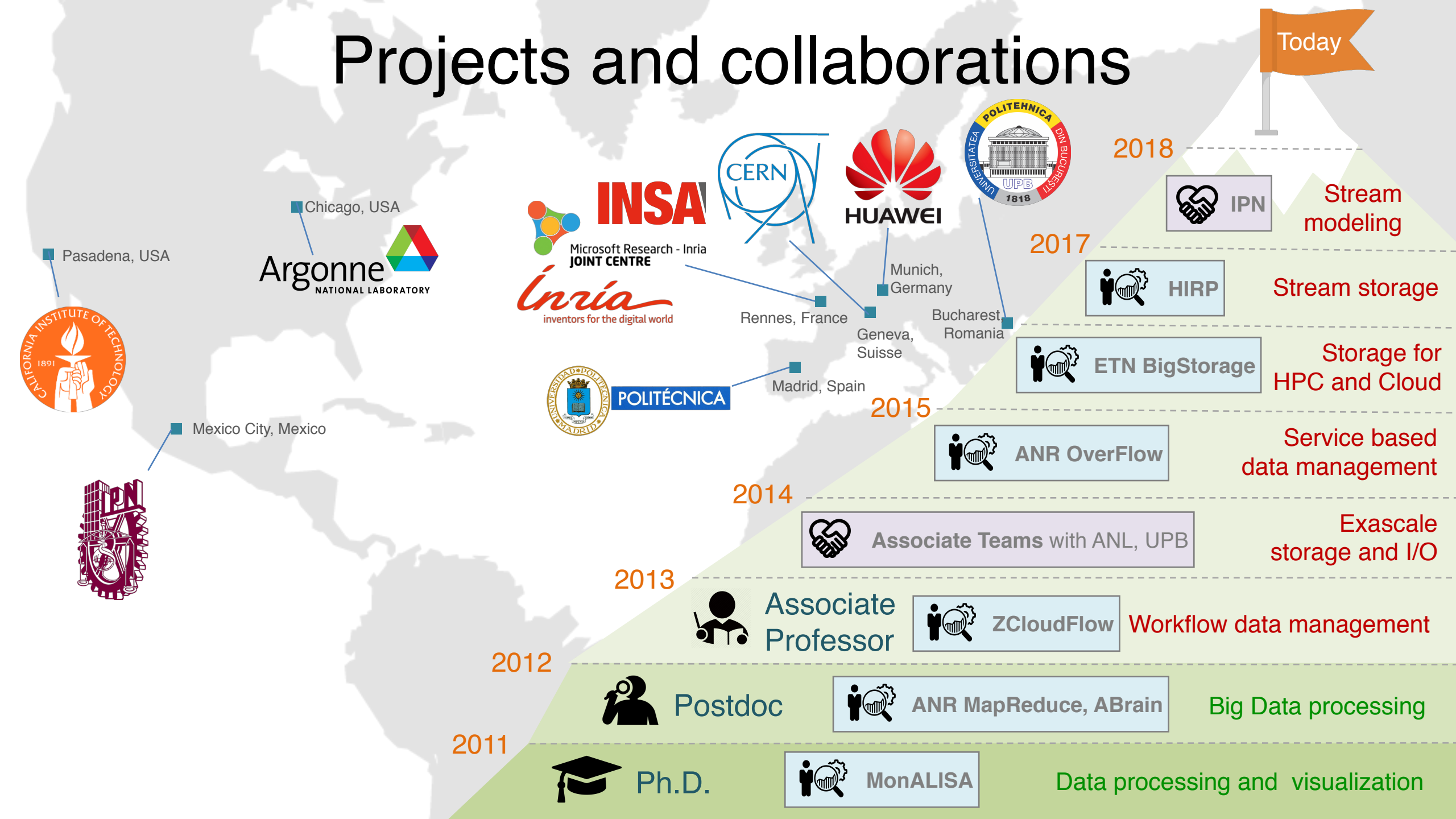


Today: centralized

Tomorrow: decentralized

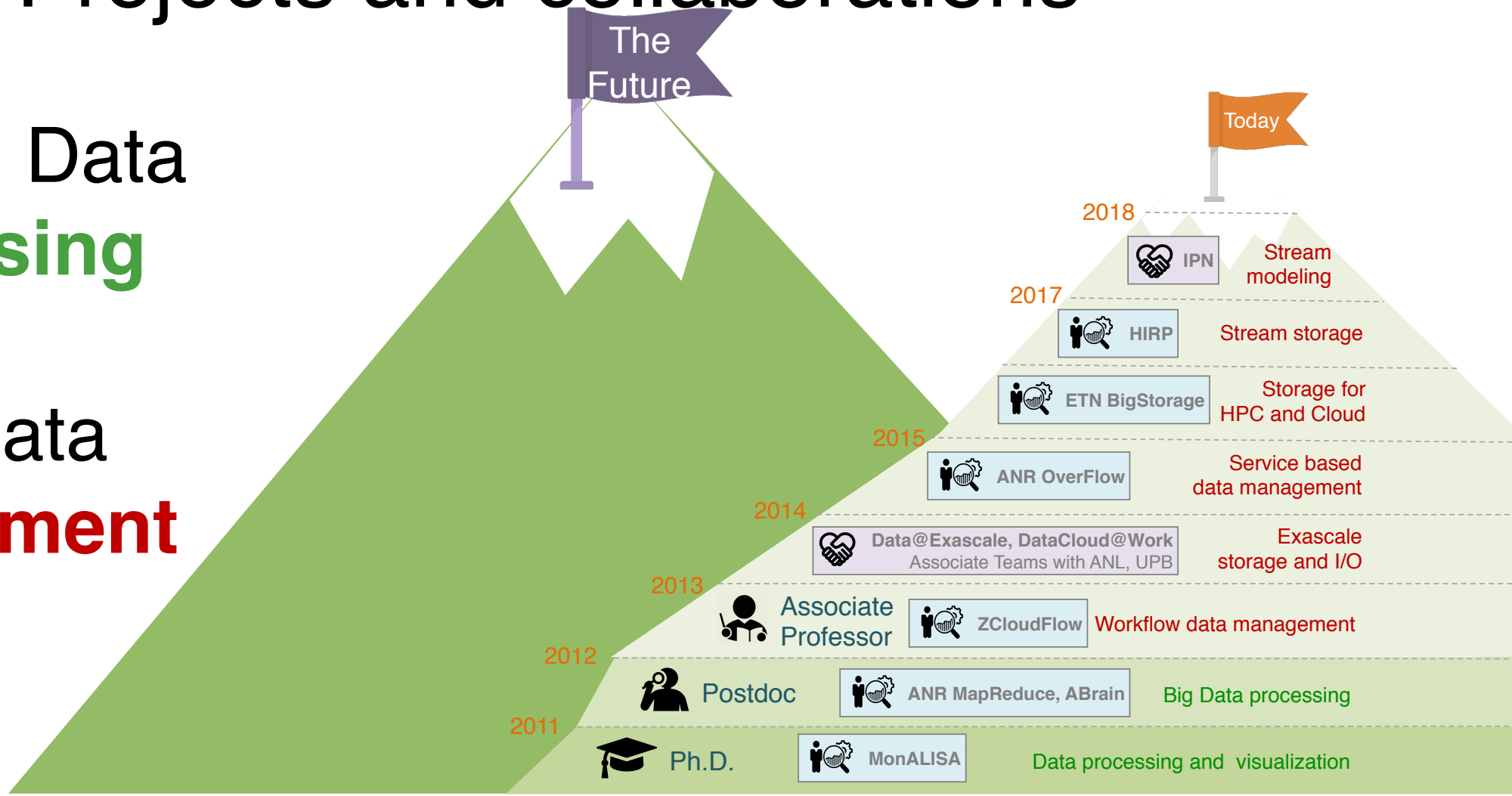
My research path

Projects and collaborations



Projects and collaborations

From Big Data
processing
to
Fast Data
management



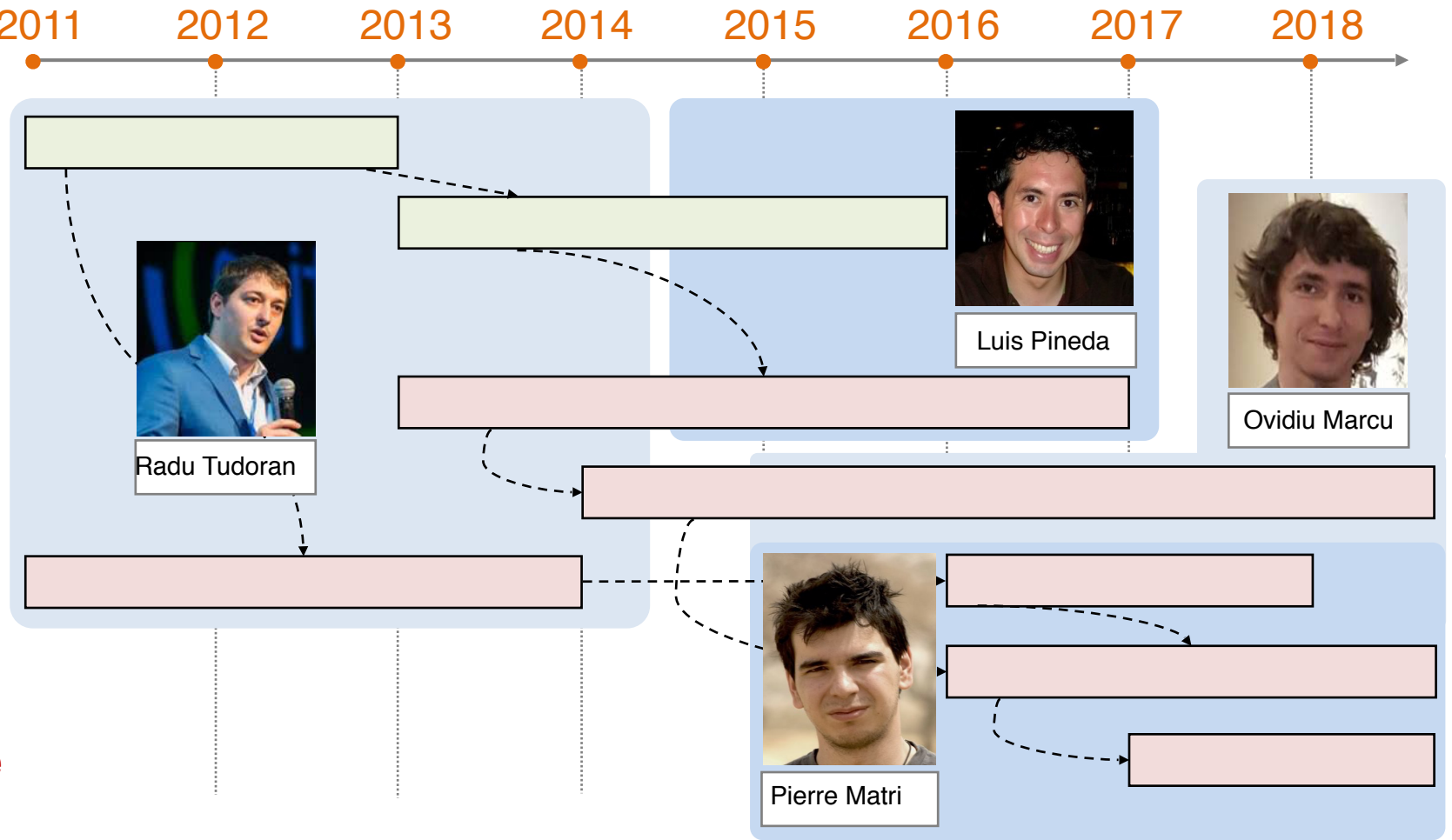
Research topics and PhD co-supervision

Big Data processing

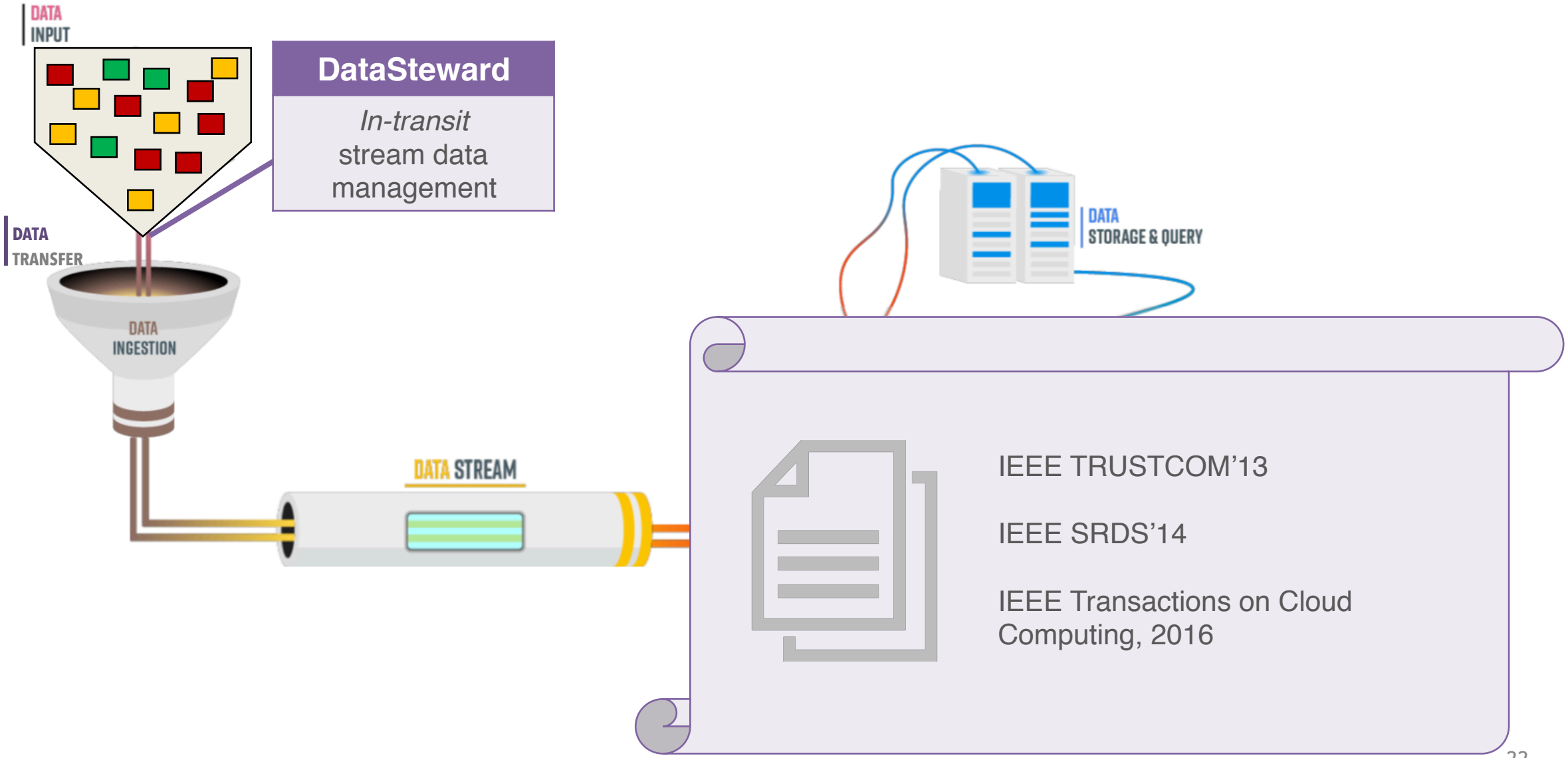
- Iterative MapReduce
- Geo-distributed processing

Fast Data management

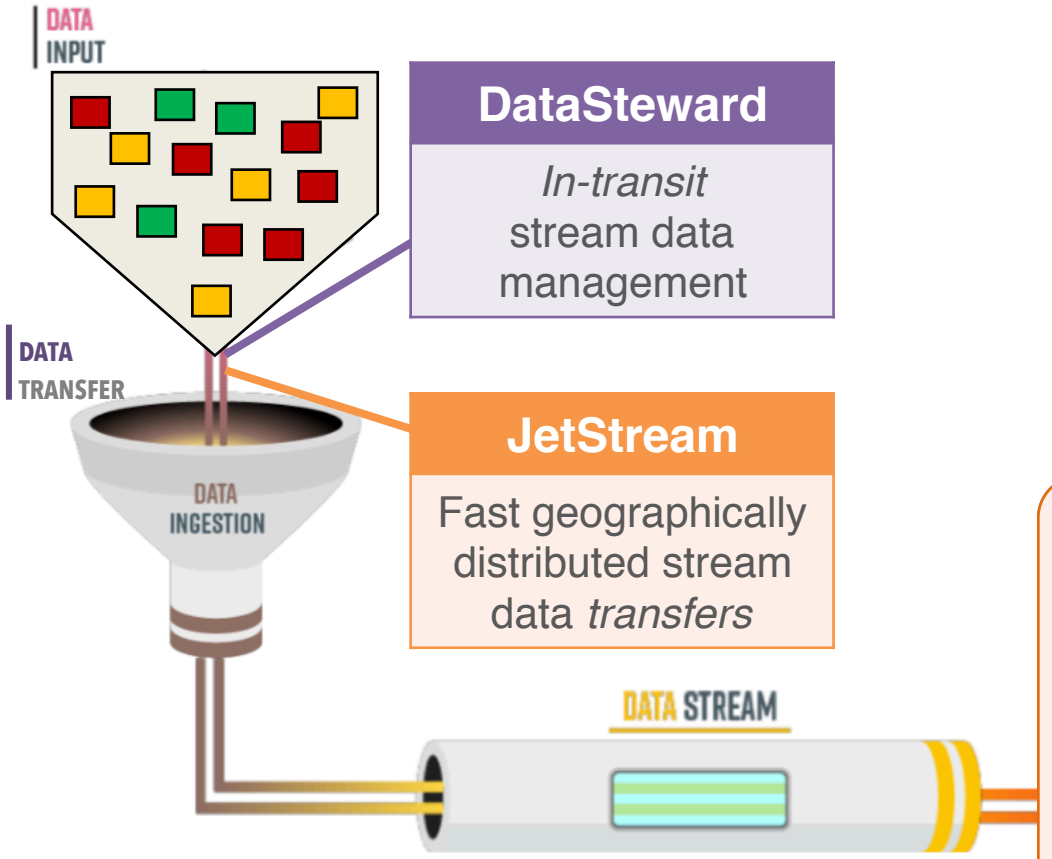
- Workflow data management
- Stream data management
- Blob storage
- Transactional storage
- HPC and Big Data convergence



My contributions



My contributions



DataSteward

In-transit stream data management

JetStream

Fast geographically distributed stream data *transfers*

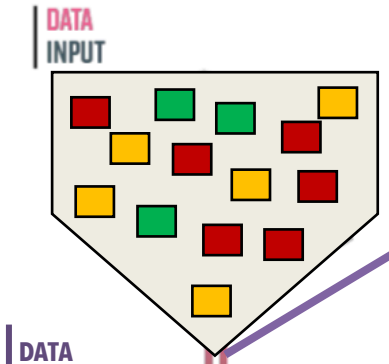
IEEE BIGDATA'13

ACM DEBS'14

IEEE/ACM CCGRID'14

Future Generation of Computer Systems, 2014

My contributions

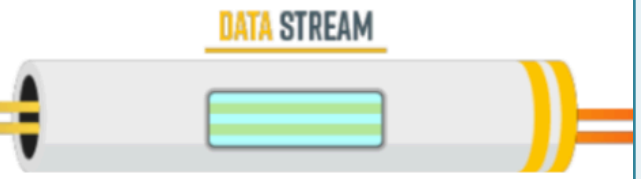


DataSteward

In-transit stream data management

JetStream

Fast geographically distributed stream data *transfers*



KerA

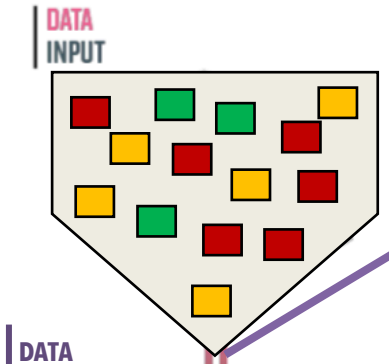
Scalable stream *ingestion*

IEEE BIGDATA'17

EBDMA 2017 workshop (with IEEE/ACM CCGRID'17)

IEEE ICDCS'18

My contributions



DataSteward

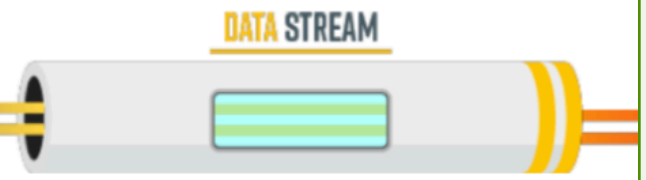
In-transit stream data management

ZCloudFlow

Adaptive distributed *metadata management*

JetStream

Fast geographically distributed stream data *transfers*

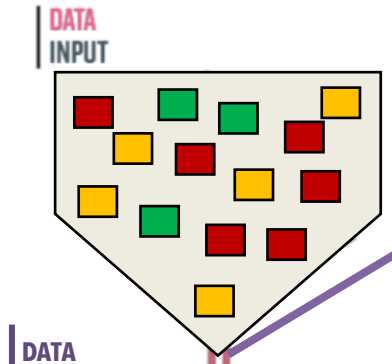


KerA

Scalable stream *ingestion*

- IEEE CLUSTER'15
- IEEE BIGDATA'16
- IEEE Transactions on Knowledge and Data Engineering, 2018

My contributions



DataSteward

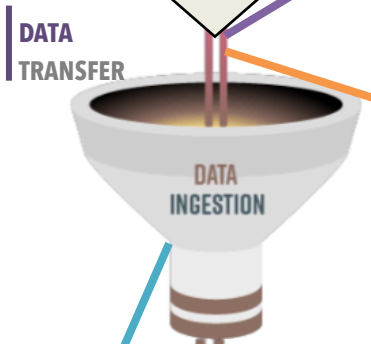
In-transit stream data management

ZCloudFlow

Adaptive distributed *metadata management*

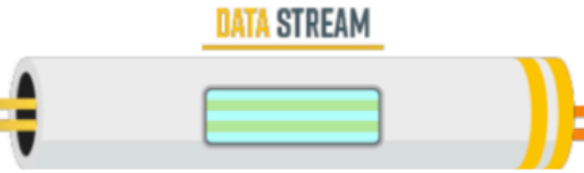
Týr

Transactional blob storage



JetStream

Fast geographically distributed stream data *transfers*



KerA

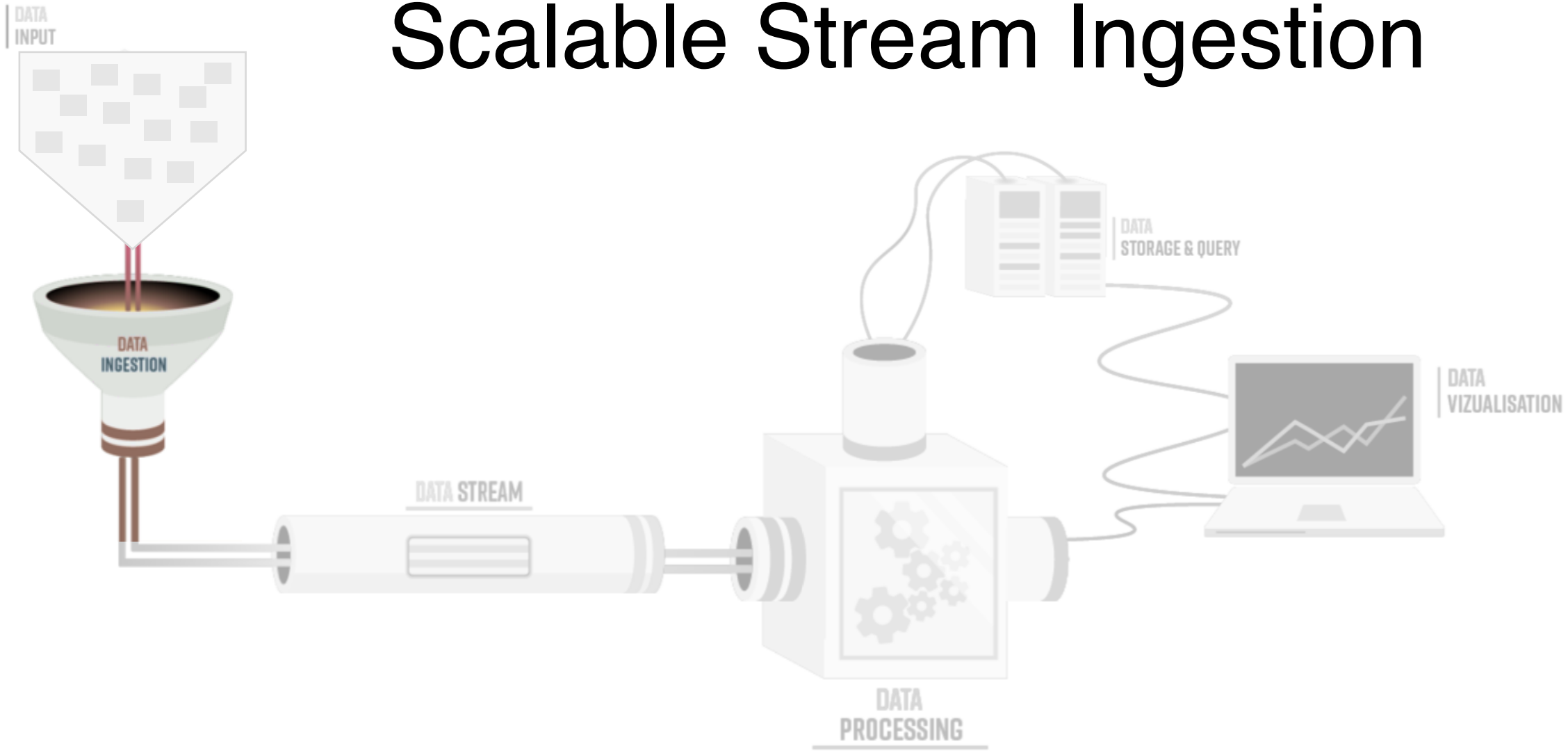
Scalable stream *ingestion*

IEEE/ACM SC'16

IEEE/ACM CCGRID'18

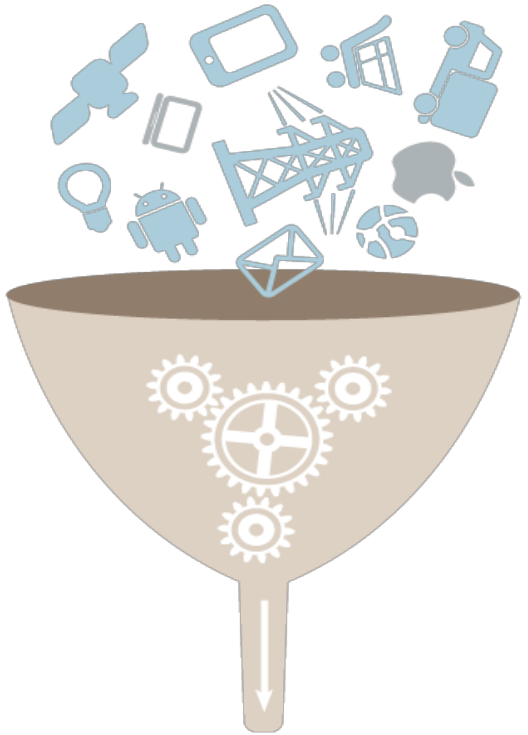
Future Generation Computer Systems, 2018

KerA:
Scalable Stream Ingestion



What is ingestion ?

- Collect data from various sources
  → *producers*
- Deliver them for processing / storage
  → *consumers*
- Optionally: buffer, log, pre-process



Ingestion determines the processing performance

State of the art: Apache Kafka

50 nodes, average 200K events/s

400 nodes, peak 3.2M events/s

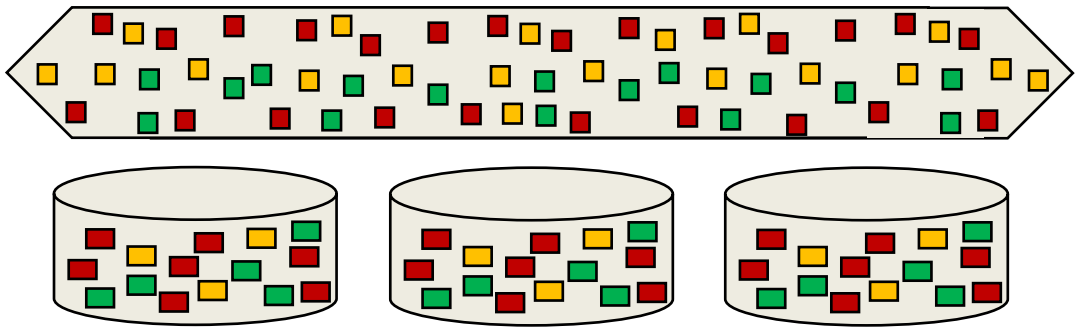
Limitations

- Scalability
- Data duplication

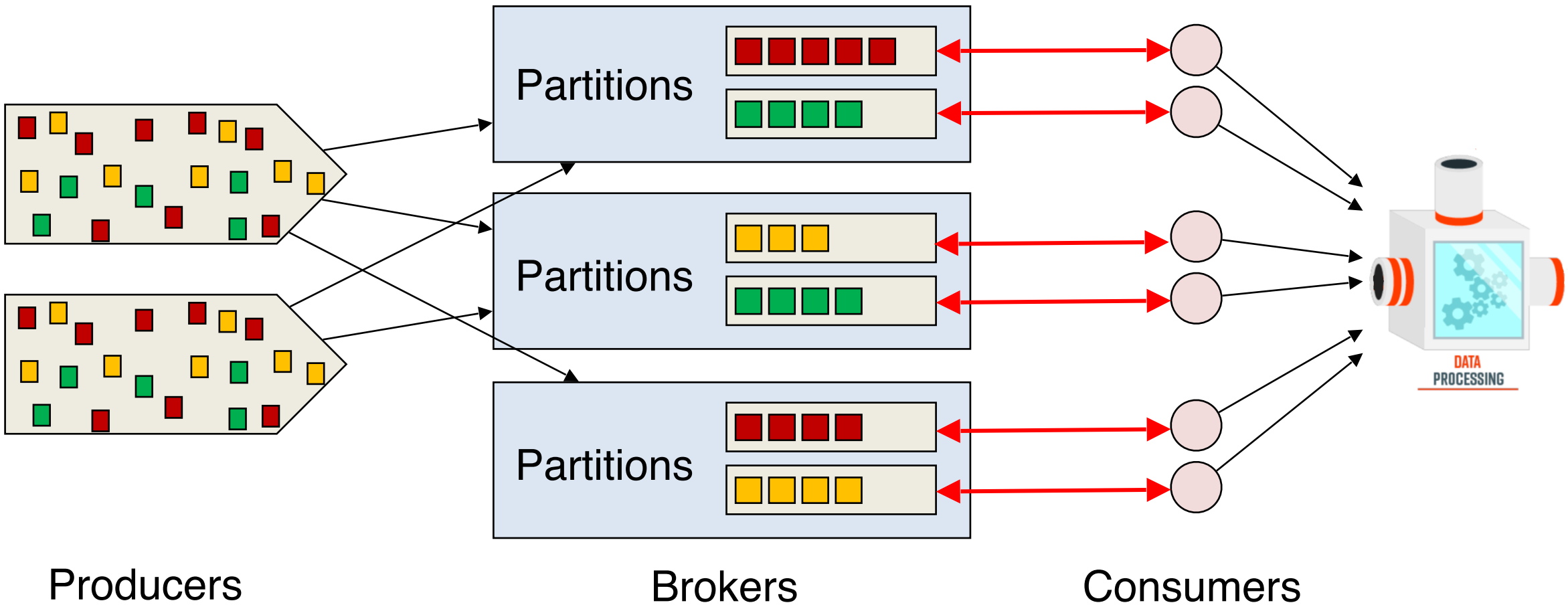
The KerA approach to ingestion

- Scalability → **Dynamic partitioning**
  - Enables seamless elasticity

- Data duplication → **Unified ingestion and storage**
  - Support for both
    - Streams (unbounded data)
    - Objects (bounded data)

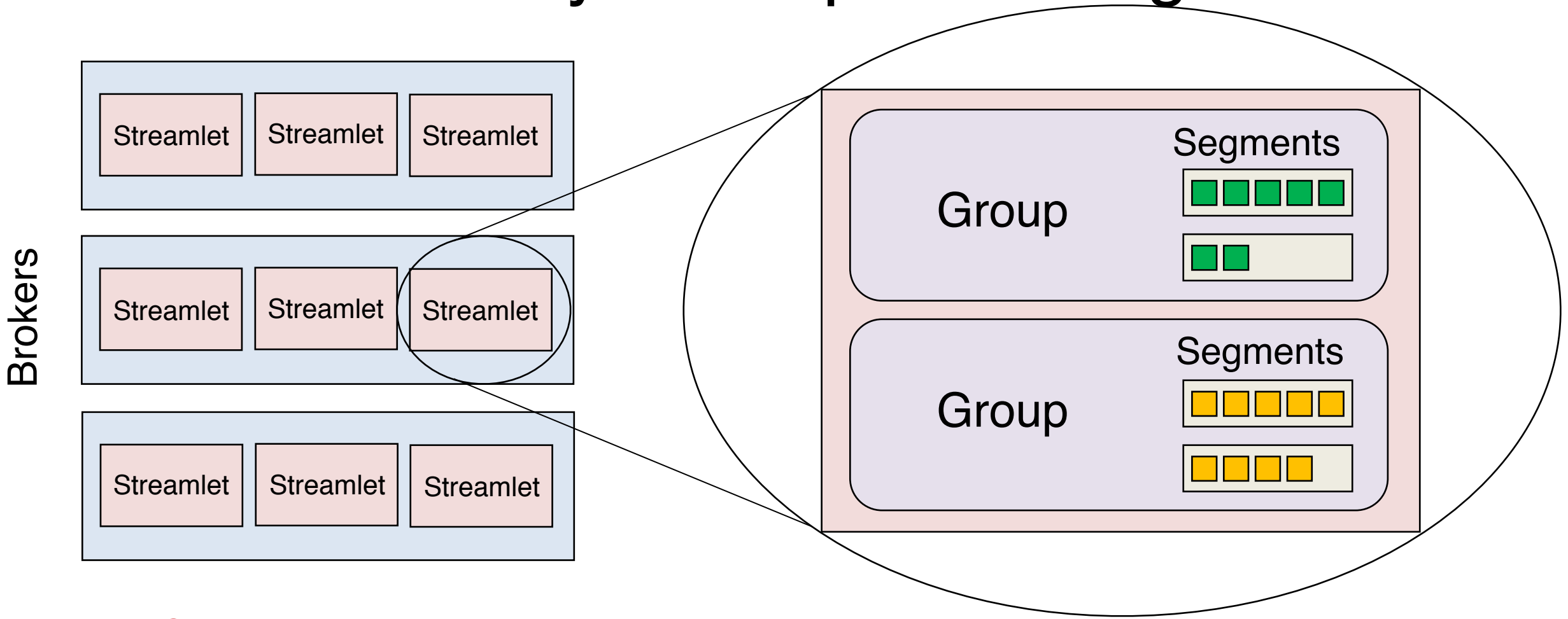


Issue: scalability



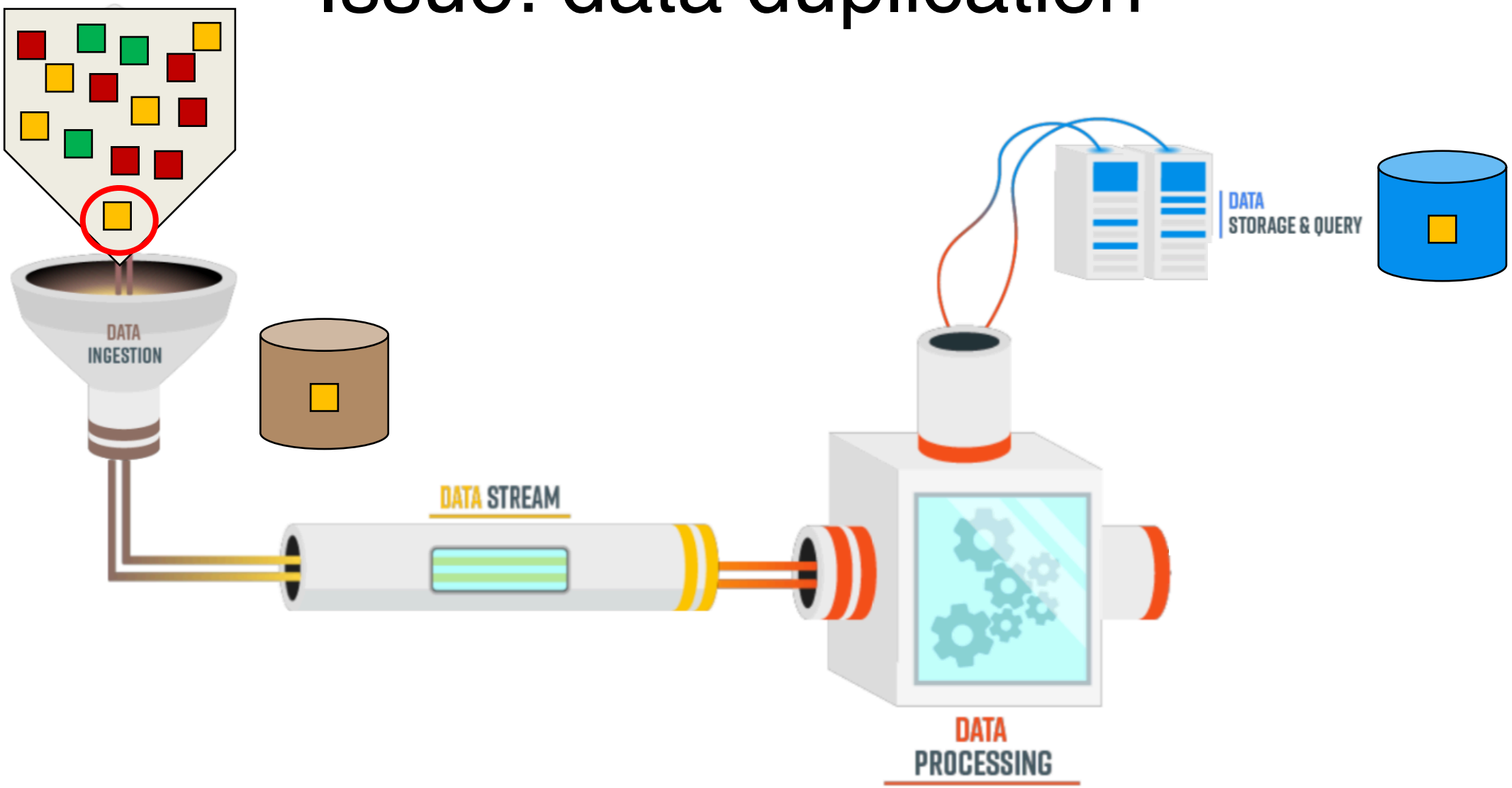
Each partition is statically associated with one consumer: limited scalability

KerA: dynamic partitioning



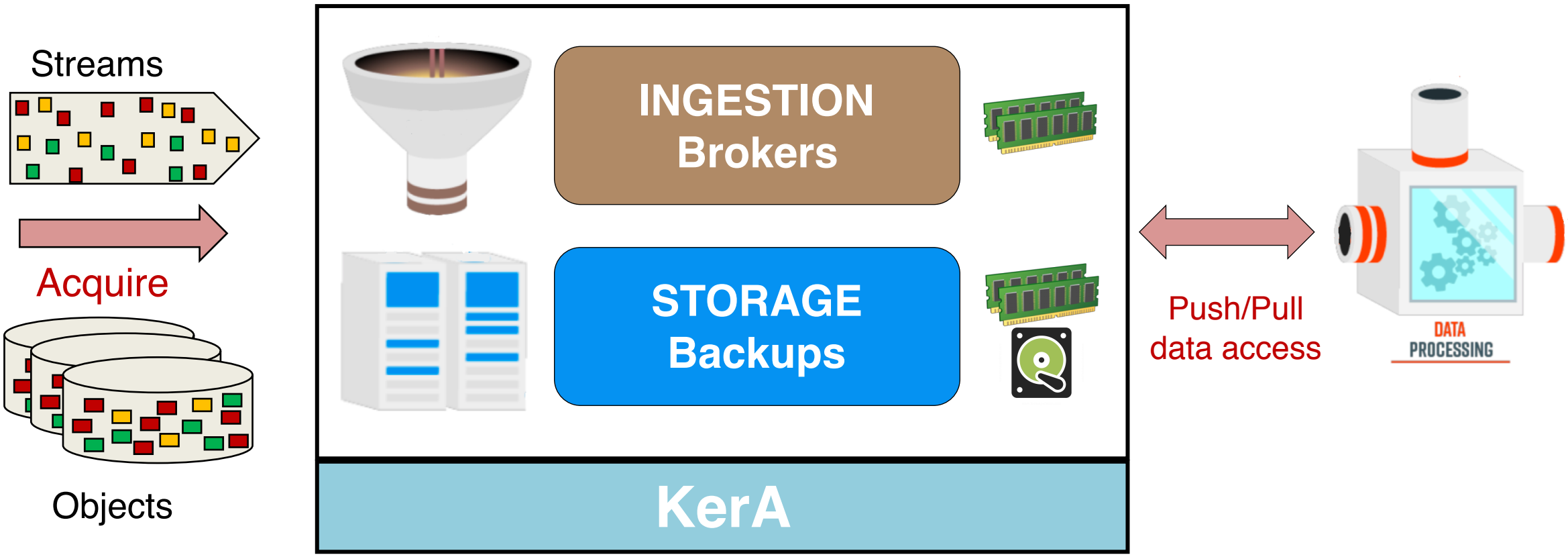
- **Streamlets:** logical stream containers; #streamlets > #brokers
- **Groups:** created and processed dynamically; up to a maximum number per broker
- **Segments:** stream partitions of fixed size; configurable #segments per group

Issue: data duplication



Increased network and storage overheads

KerA: unified ingestion and storage

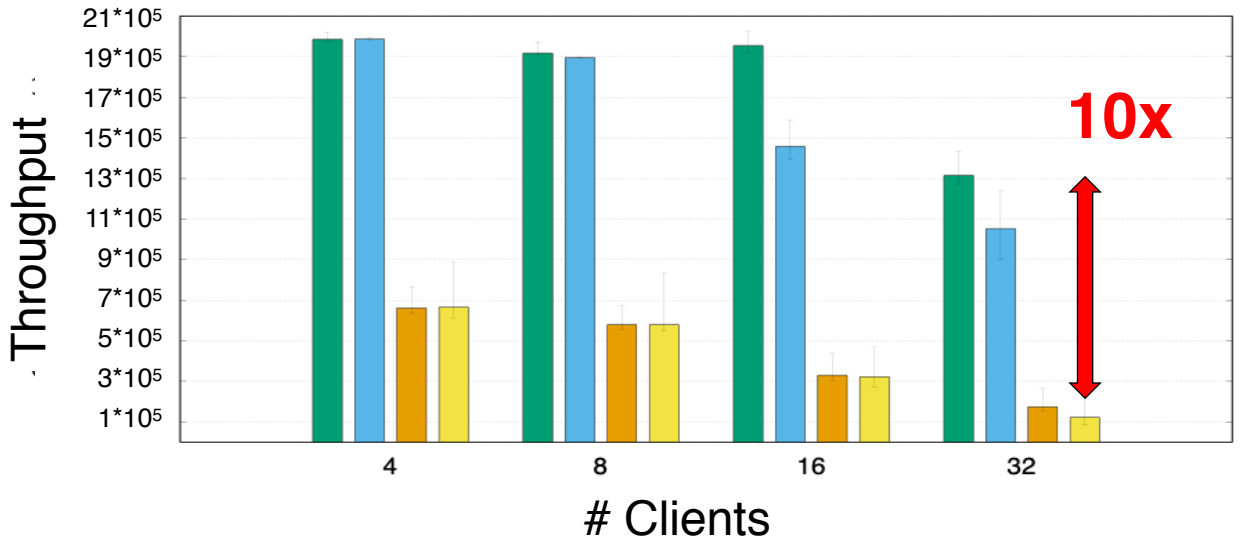


Move less data, process them faster

Common data model for streams and objects

Evaluating scalability

Vertical



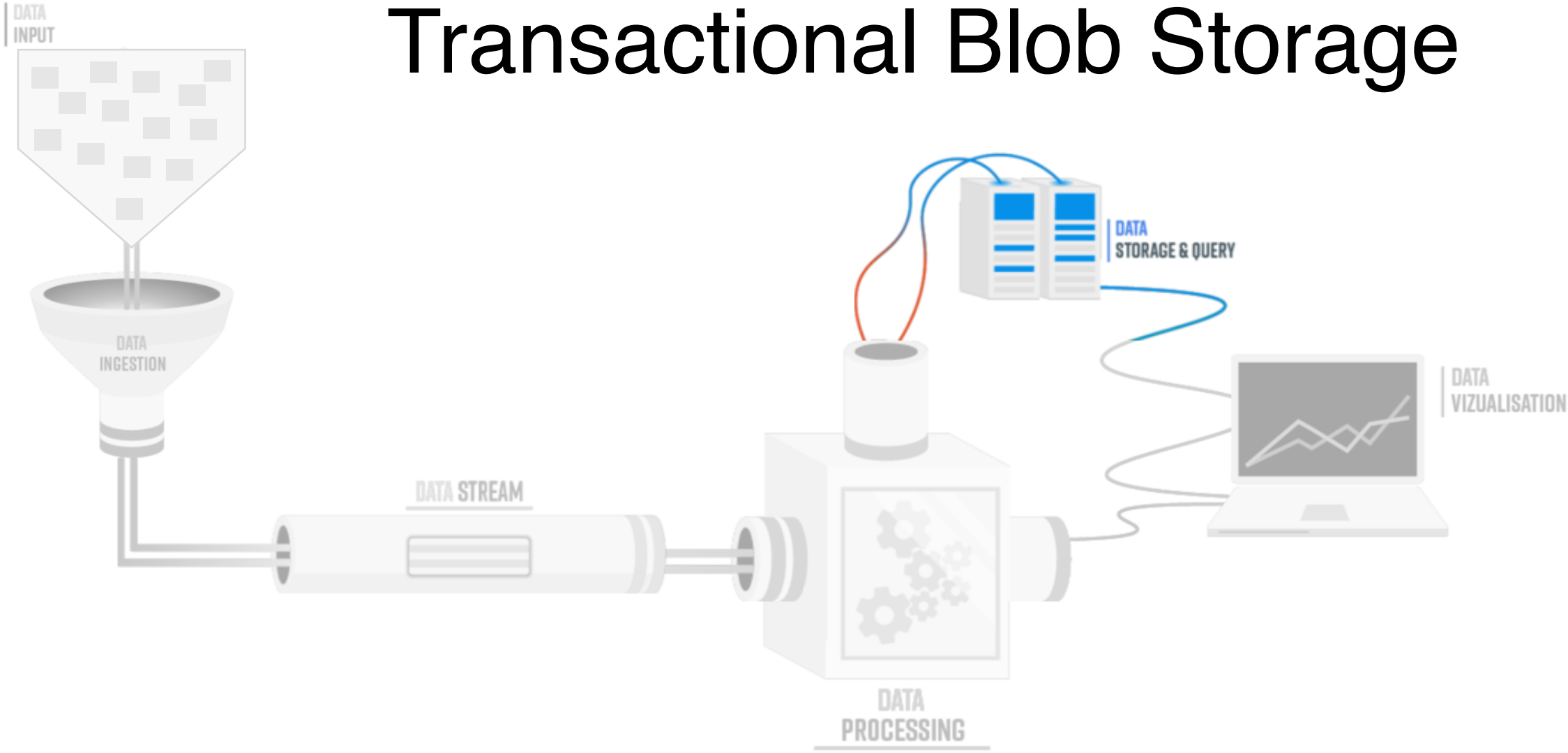
4 brokers, 32 partitions,
128KB request size, 100B records

Horizontal

64 clients, 32 partitions,
1MB request size, 100B records

2x better throughput
with 75% less resources

Týr:
Transactional Blob Storage

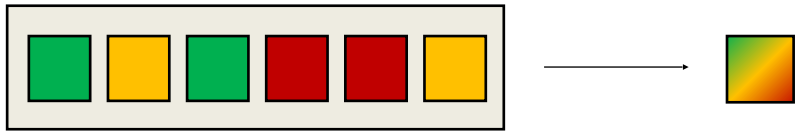


Motivating use-case: MonALISA

A large-scale monitoring and analytics service
for the CERN LHC ALICE experiment

Ingests and stores telemetry events at 13 GB/s

Computes 35,000+ aggregates of events in real-time



MonALISA RDBMS platform does not scale

Multiple storage requirements

Write atomicity for aggregate updates

Atomic, lock-free writes

High-performance reads

Horizontal scalability

Write atomicity for aggregate updates

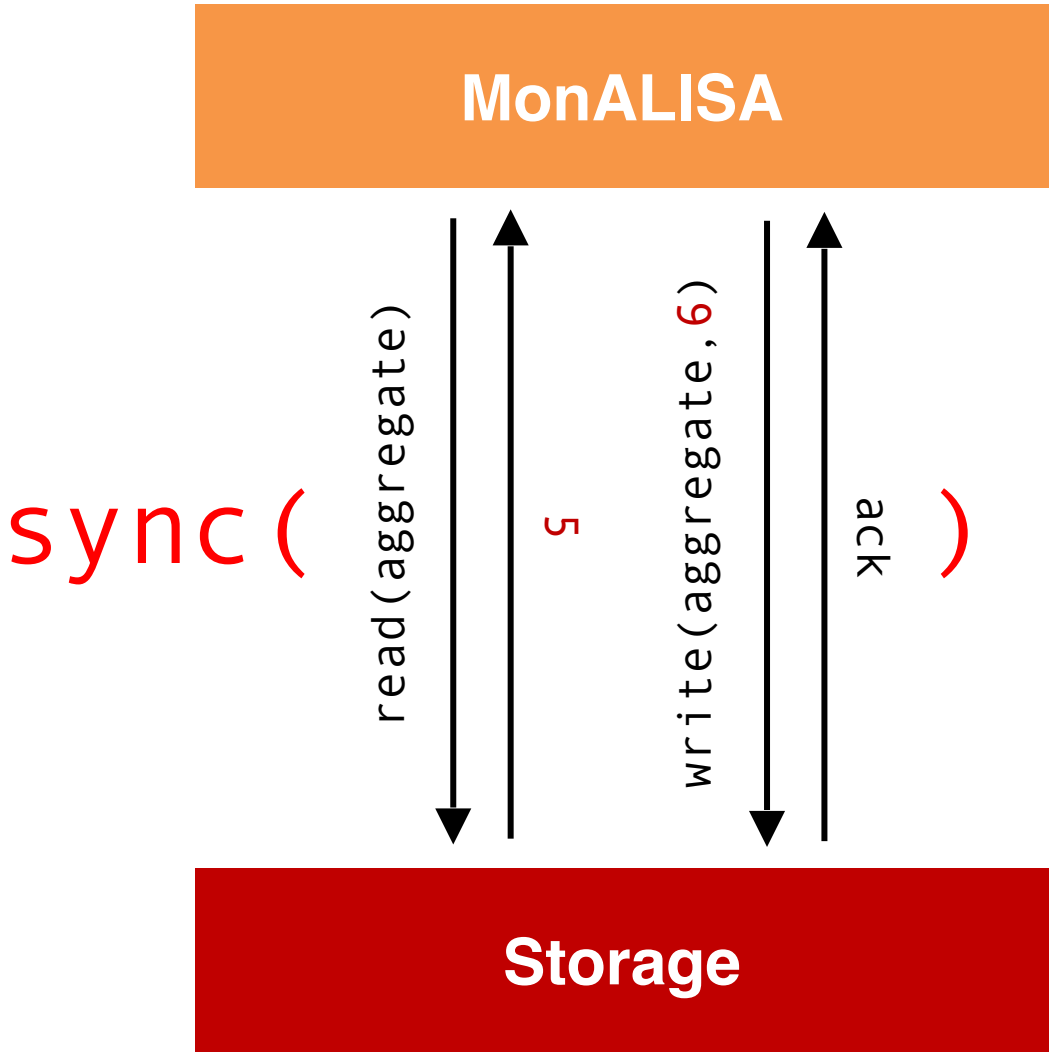
Aggregate update is a two-step operation

- Read current value remotely from storage
- Write the updated value remotely to storage

Aggregate update needs to be atomic

- Concurrent writers!

Synchronization is mandatory

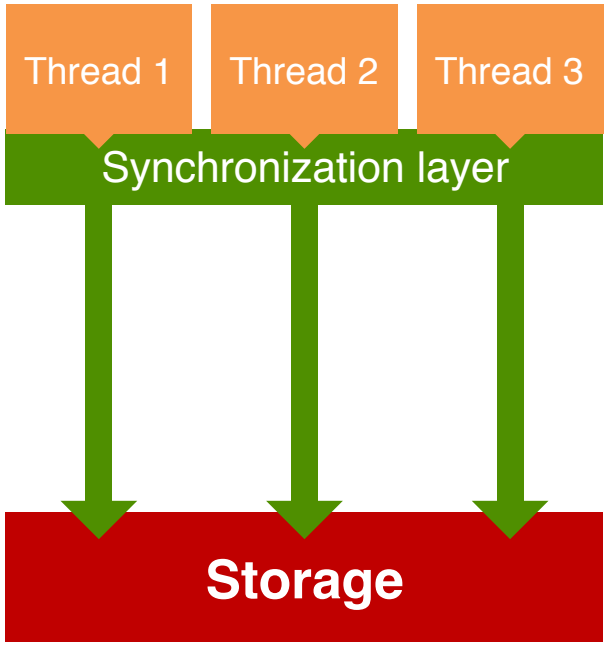


At which level to handle synchronization?

At application level?

Fine-grained synchronization
Application-specific optimization
…but increases app complexity, error-prone

Common on HPC (e.g., explicit locking)



At which level to handle synchronization?

At application level?

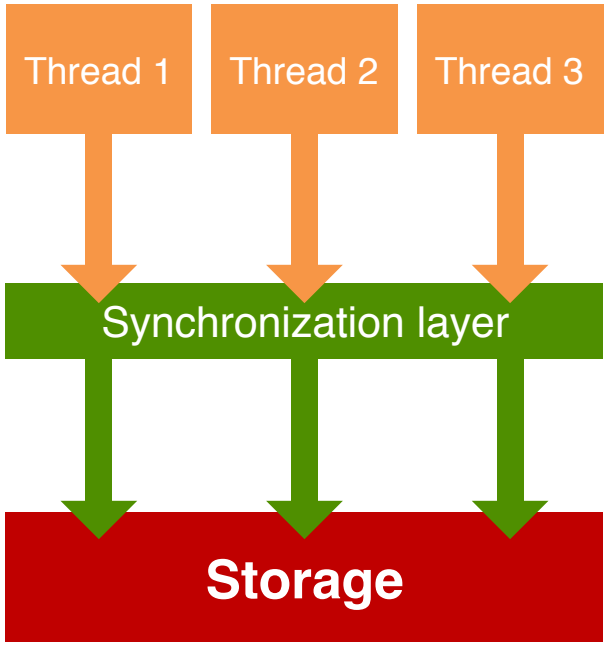
Fine-grained synchronization
Application-specific optimization
…but increases app complexity, error-prone

Common on HPC (e.g., explicit locking)

At middleware level?

Eases application design
…but typically substantial performance overhead

Also common on HPC (e.g., MPI collective I/O)



At which level to handle synchronization?

At application level?

Fine-grained synchronization
Application-specific optimization
…but increases app complexity, error-prone

Common on HPC (e.g., explicit locking)

At middleware level?

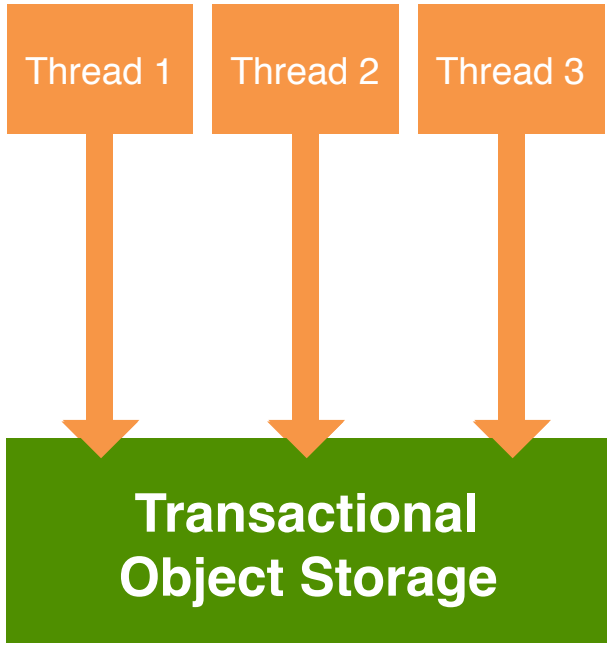
Eases application design
…but typically substantial performance overhead

Also common on HPC (e.g., MPI collective I/O)

At storage level?

Also eases application design
Storage-specific optimization
…but less customizable than app-level synchronization

Common on BDA (e.g., transactional systems)



Transactional API

```
> begin()
> current = read(aggregate, …)
> write(aggregate, current+1, …)
> commit()
```

Týr read protocols

Direct read	**Multi-chunk read**	**Transactional read**
1 RTT Similar to key-value stores Low latency No multi-chunk consistency guarantees	1 RTT + 1 Additional cost Multi-chunk consistency No repeatable reads *i.e.* no consistency guarantees between successive reads	First read 1 RTT + 1 Additional cost Subsequent reads 1 RTT Multi-chunk consistency Repeatable reads

`> read(blob, 0, 10kb)`	`> read(blob, 0, 100mb)`	`> begin()`
		`> read(blob, 0, 10kb)`
		`> read(blob, 100mb, 10kb)`
		`> commit()`

The developer can select an algorithm offering lesser guarantees

Results in a substantial performance increase

Useful for example for append logs, in which multi-chunk operations are not needed

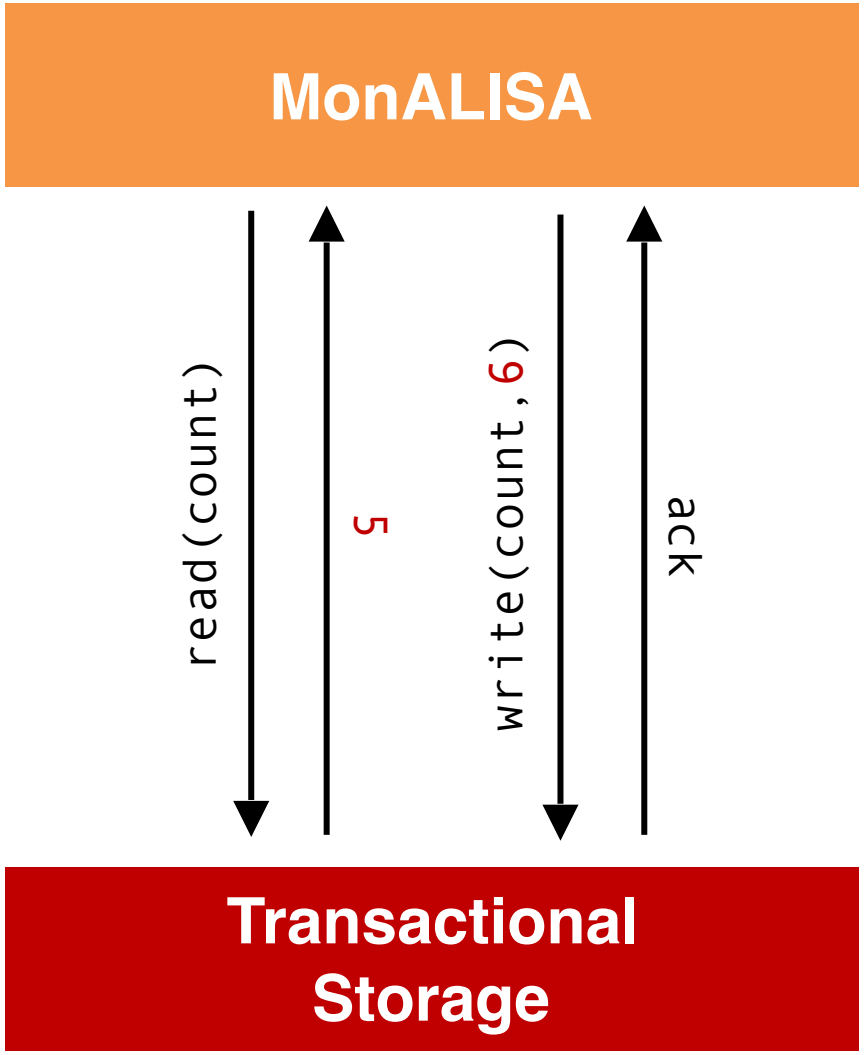
Týr atomic writes

MonALISA: aggregate updates could be performed atomically and efficiently

Týr enables these writes to be performed with one round-trip instead of two

Atomic operations: in-place data modification

Integrated with the transaction protocol



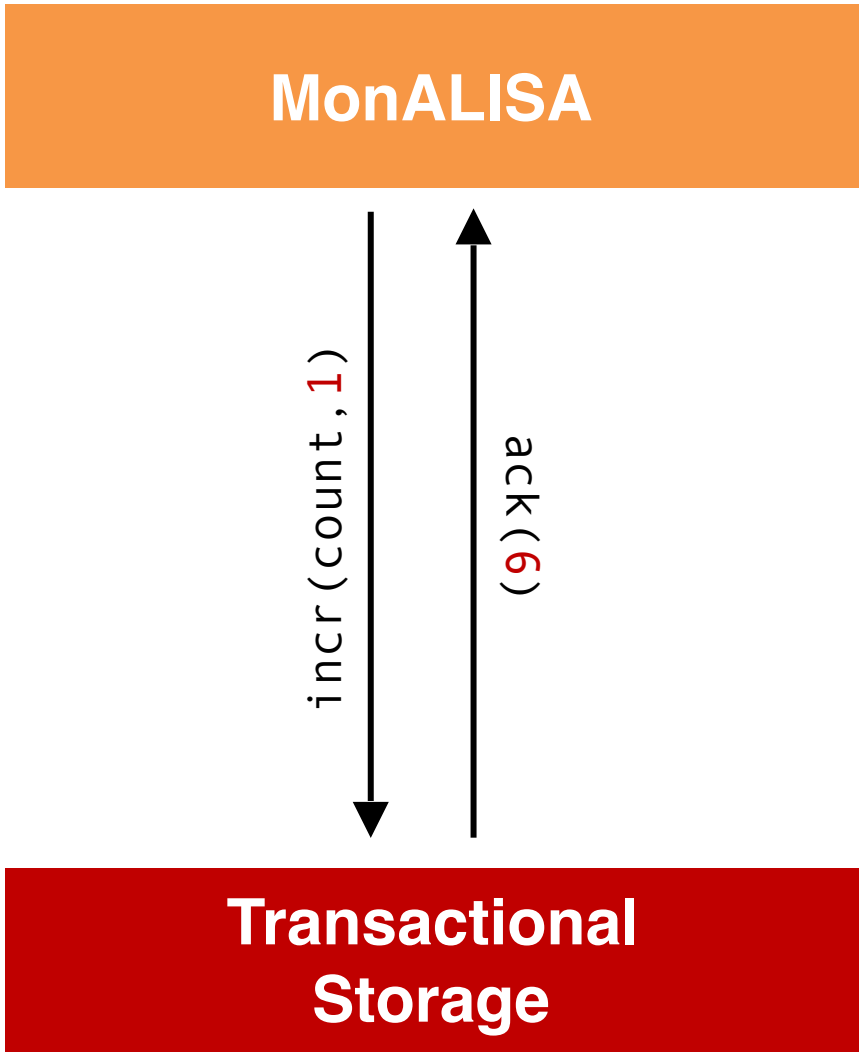
Týr atomic writes

MonALISA: aggregate updates could be performed atomically and efficiently

Týr enables these writes to be performed with one round-trip instead of two

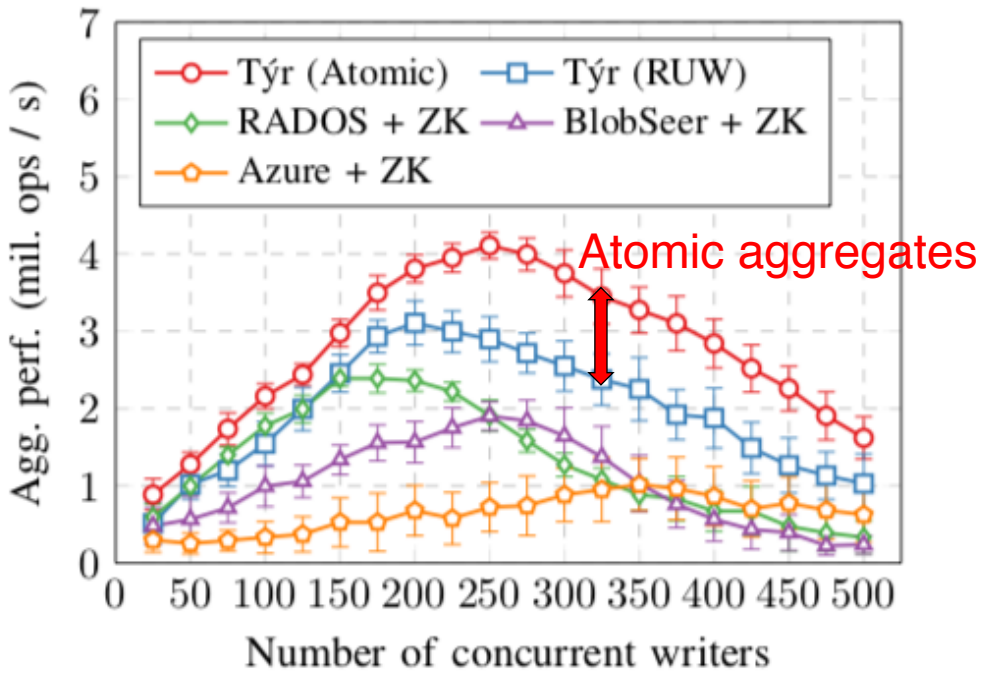
Atomic operations: in-place data modification

Integrated with the transaction protocol

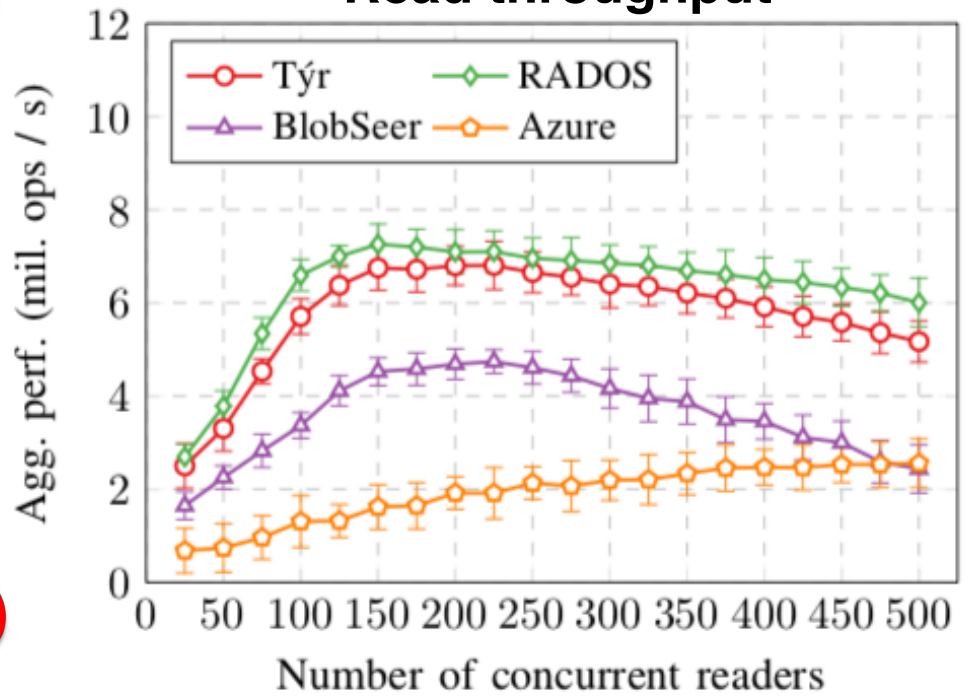


Read / Write performance

Transactional write throughput



Read throughput



Only Týr is transactional

Other systems with fine-grained locking

Clear performance gain for transactions

Less network overhead, +23% *vs.* Rados

Atomic operations are very effective

One RTT instead of two, +25% throughput

Only Týr provides multi-chunk reads

Slightly lower performance than Rados

Cost of read protocols, -5% *vs.* Rados

BlobSeer: metadata management cost

No direct reads

Azure: internals are not documented

Perspectives:
HPC and Big Data Convergence

Cars

Devices

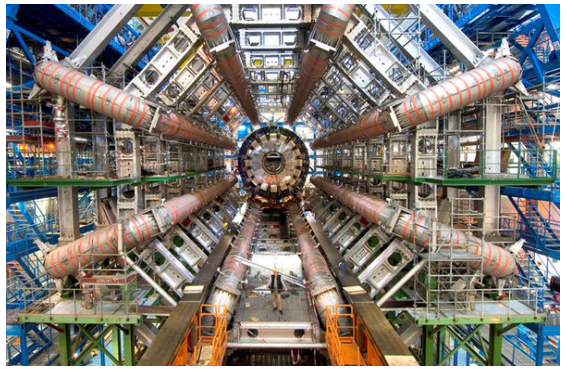
IoT / Smart City

SMALL and MANY

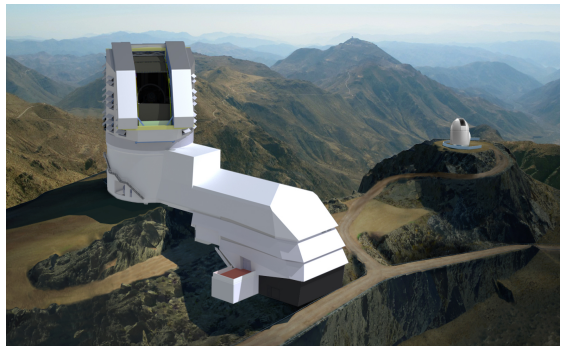
Multidisciplinary Data Fusion

BIG and FEW

LHC

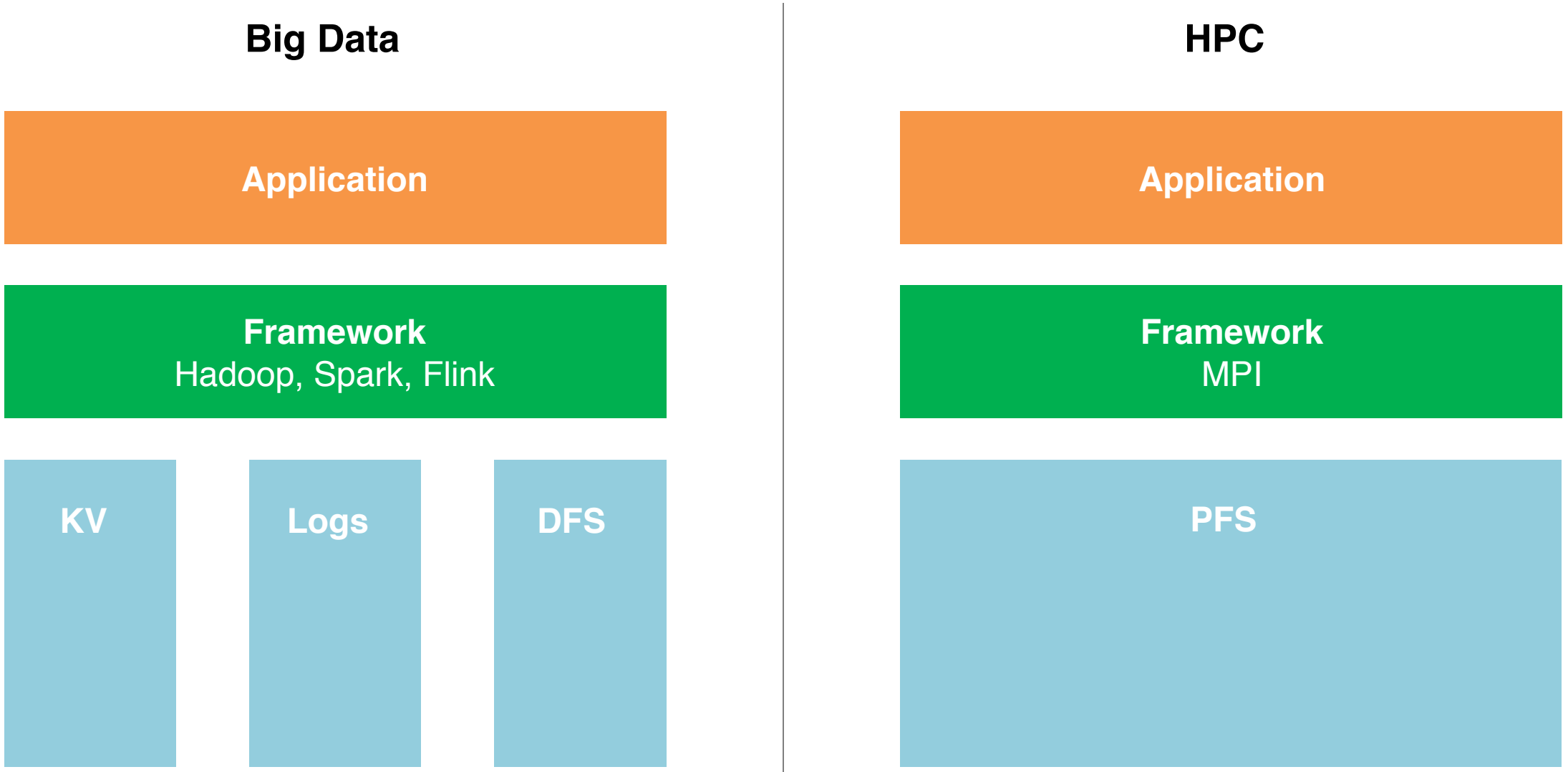


LSST

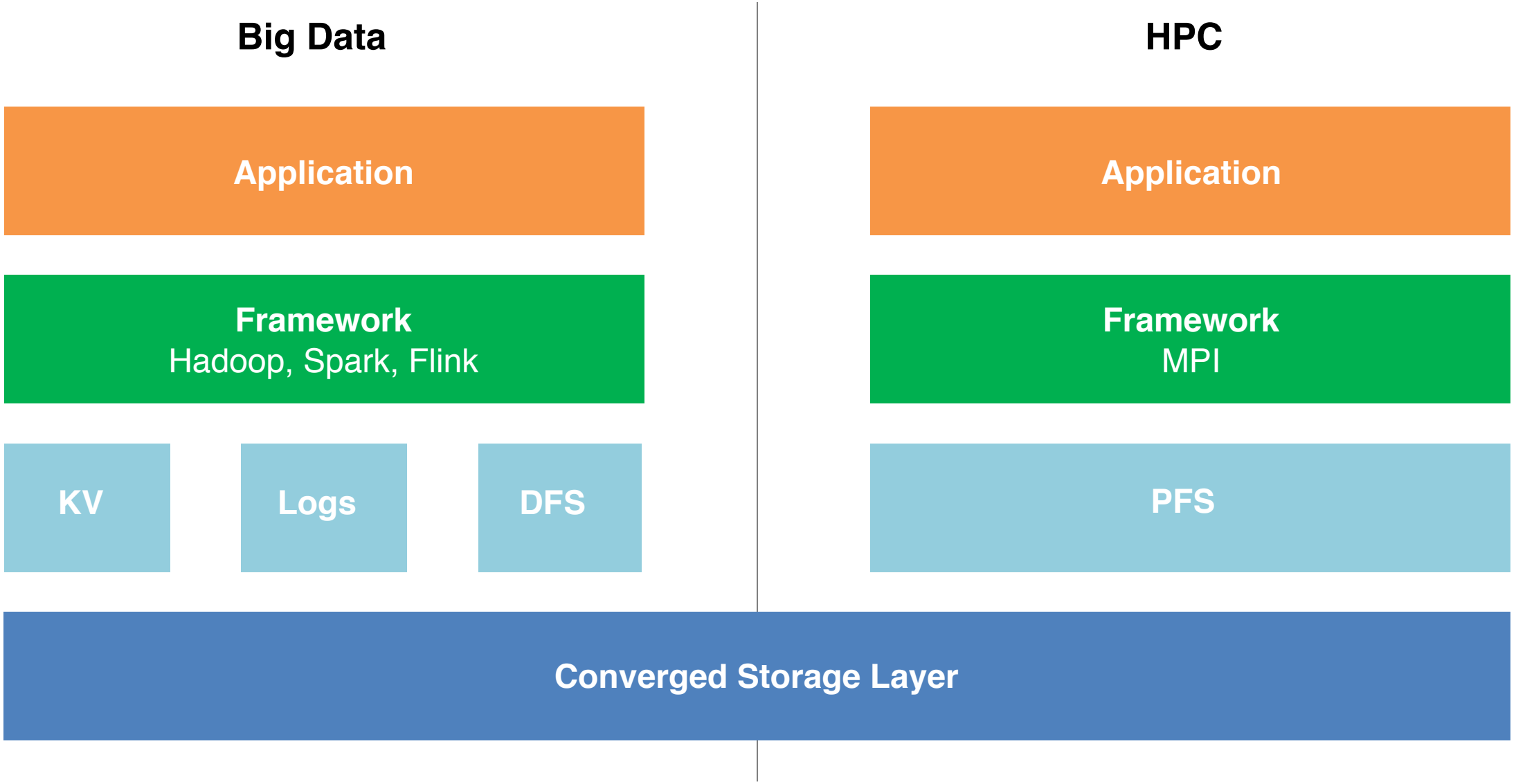


SKA

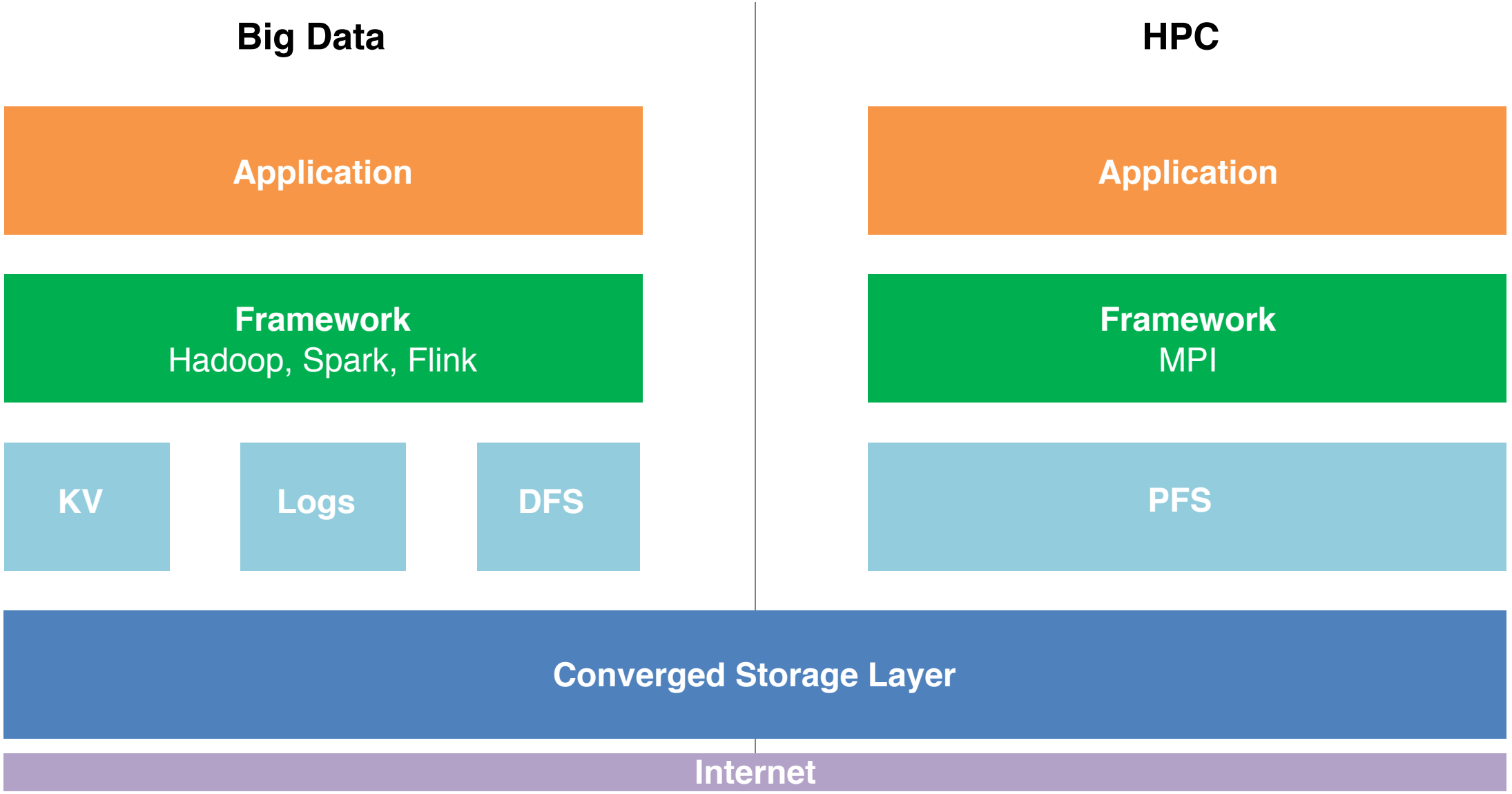
HPC / Big Data convergence



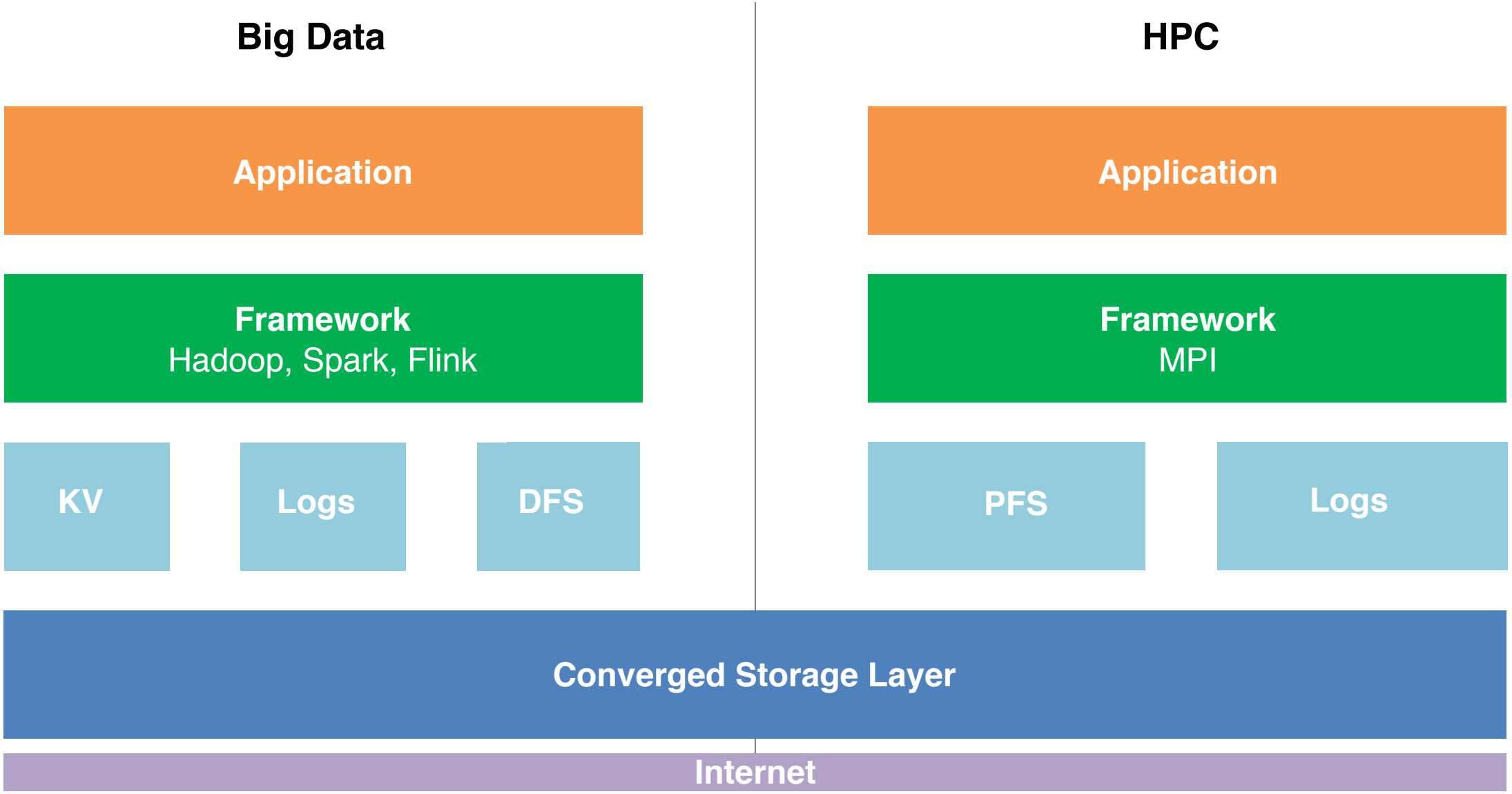
HPC / Big Data convergence



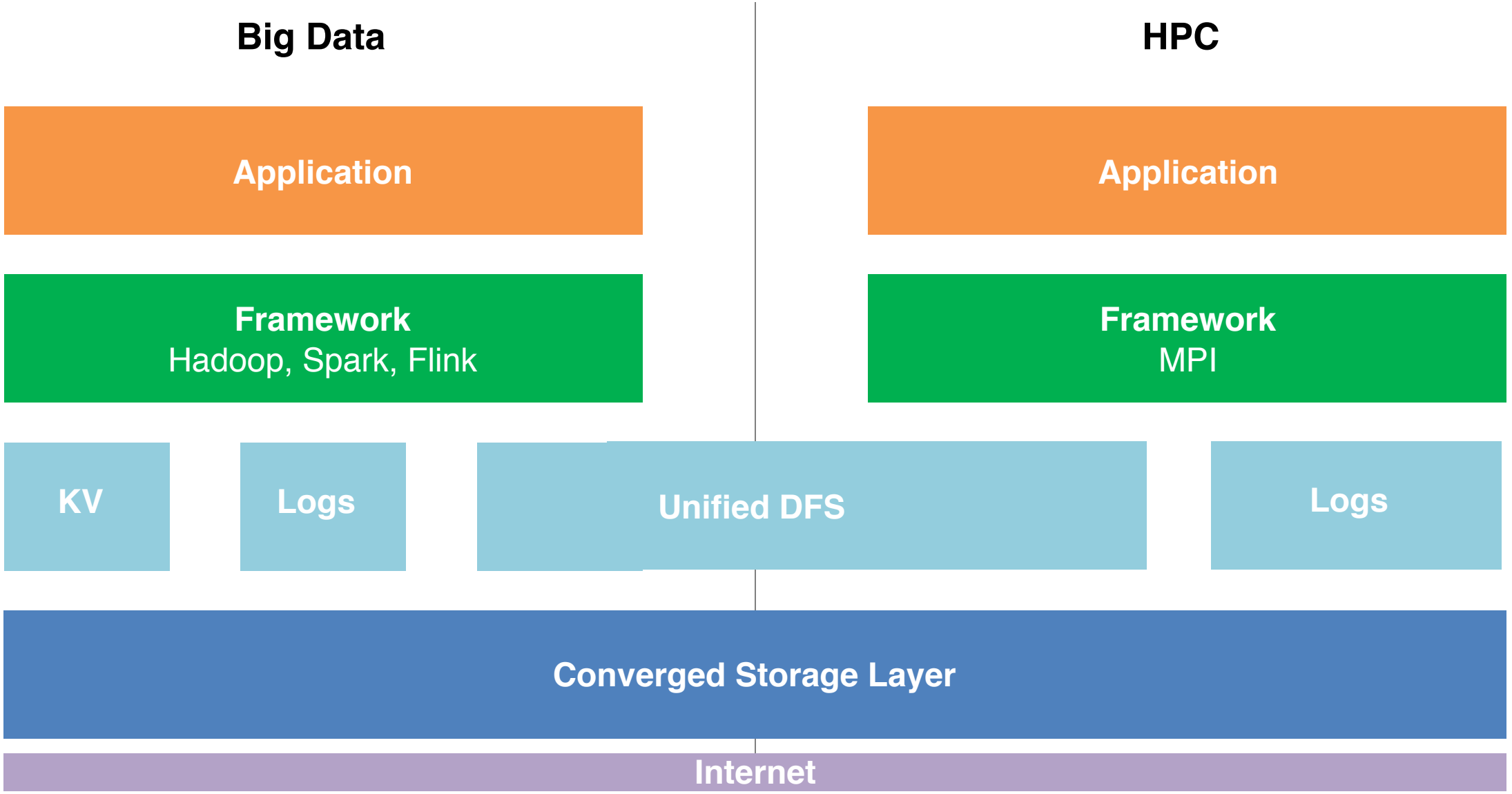
HPC / Big Data convergence



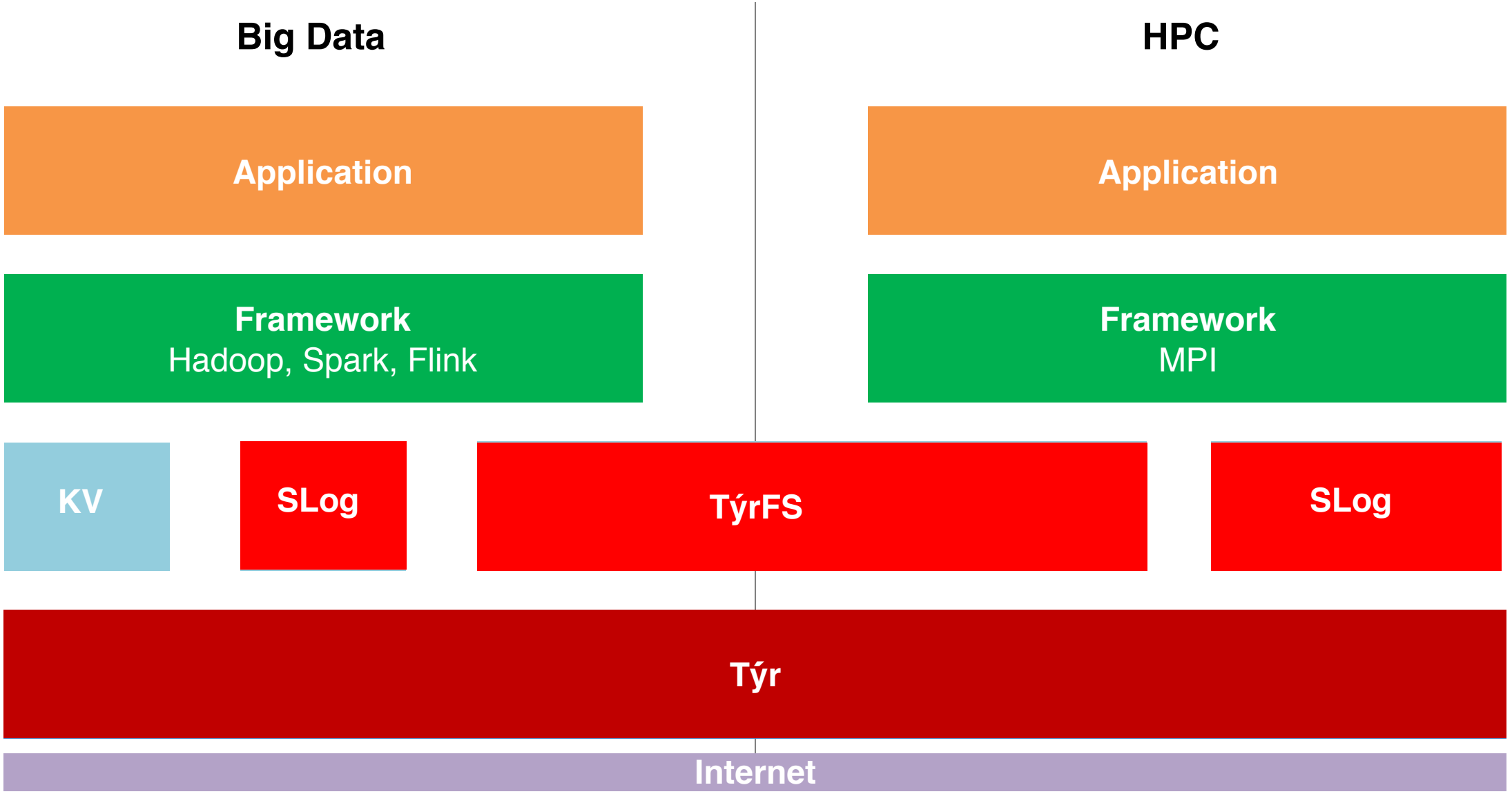
HPC / Big Data convergence



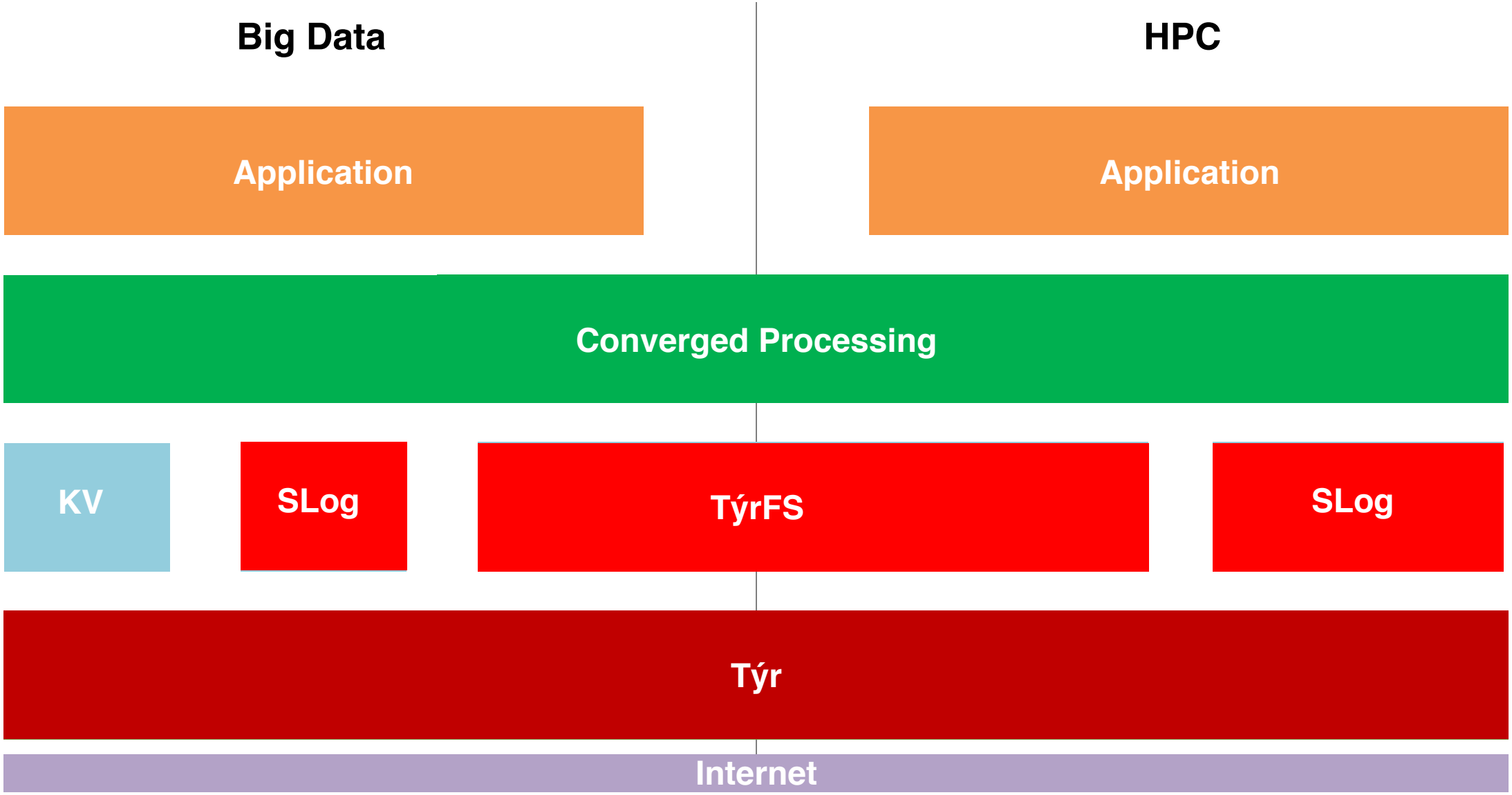
HPC / Big Data convergence



HPC / Big Data convergence



HPC / Big Data convergence



My Research Project:

Converged Processing

… or how ***Past, Present*** and ***Future*** data could jointly enable disruptive analytics on Extreme-scale infrastructures

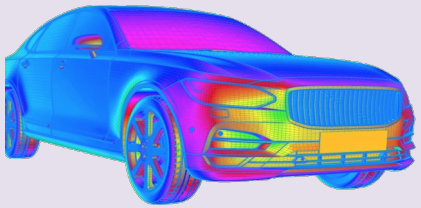
Scenario: digital twins

Real World Entities

Scenario: digital twins

Real World Entities

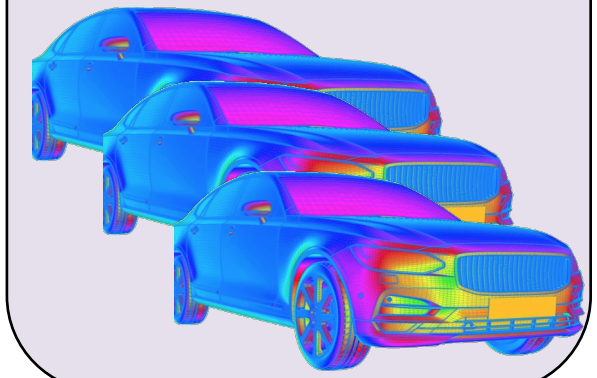
Digital Twins



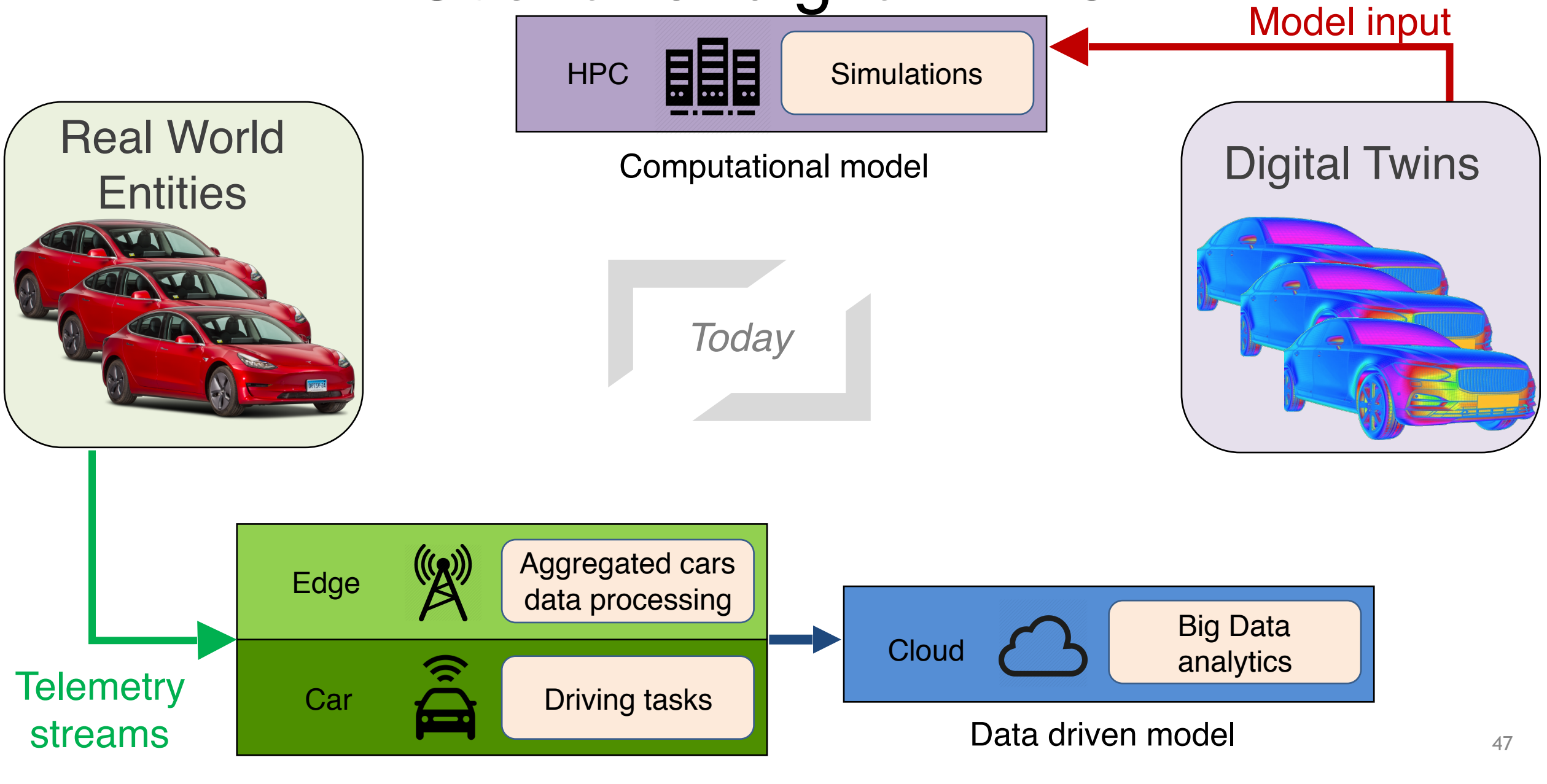
Scenario: digital twins

Real World Entities

Digital Twins



Scenario: digital twins



Scenario: digital twins

Feedback and control

HPC

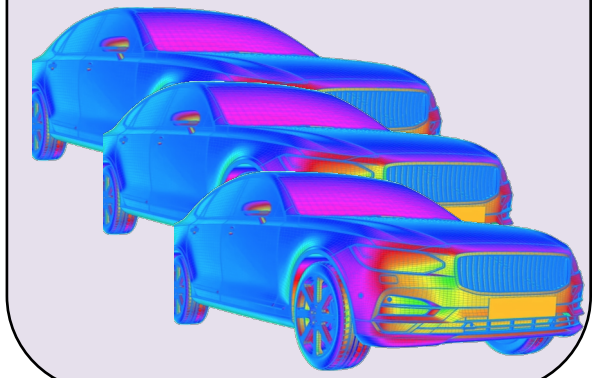
Simulations

Model input

Computational model

Real World Entities

Digital Twins



Tomorrow

Model update

Edge

Aggregated cars data processing

Car

Driving tasks

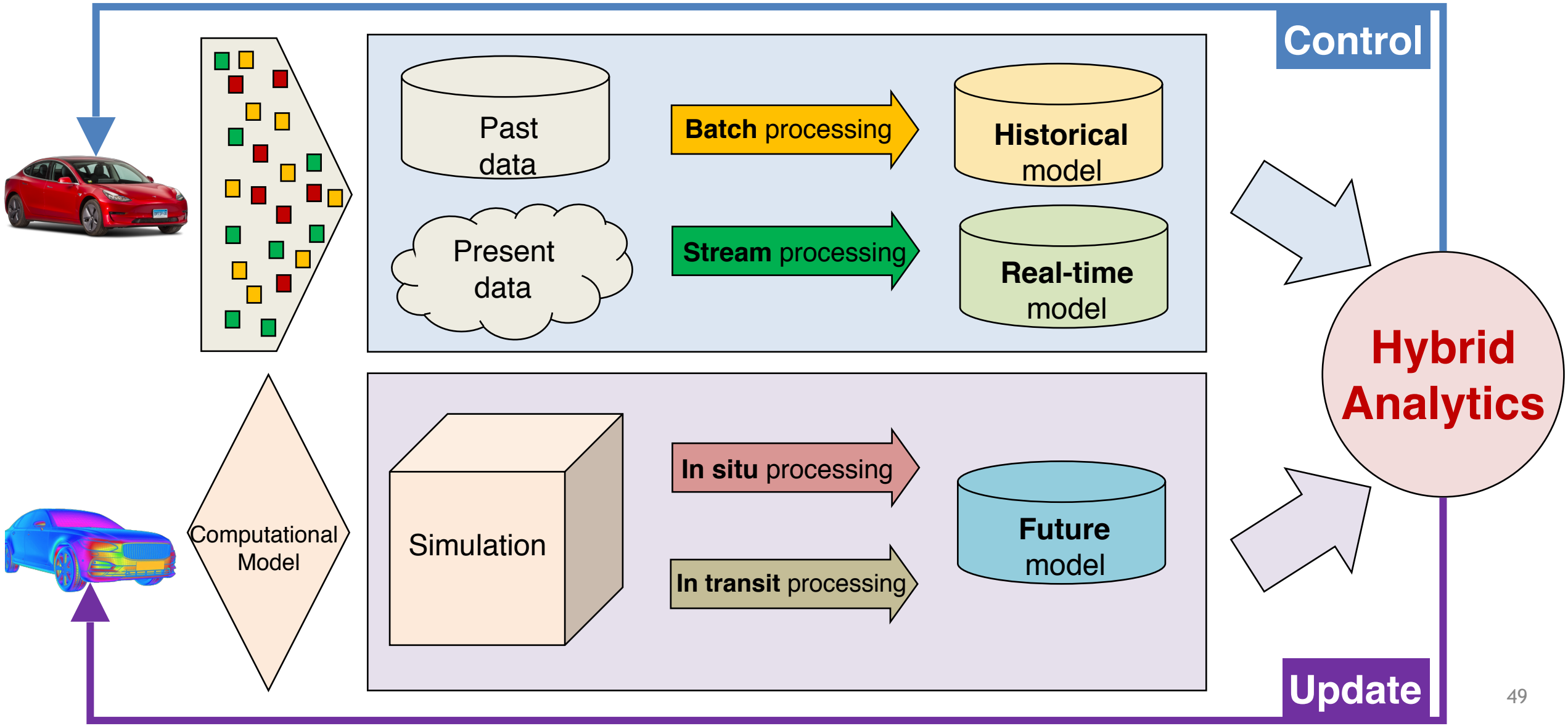
Telemetry streams

Cloud

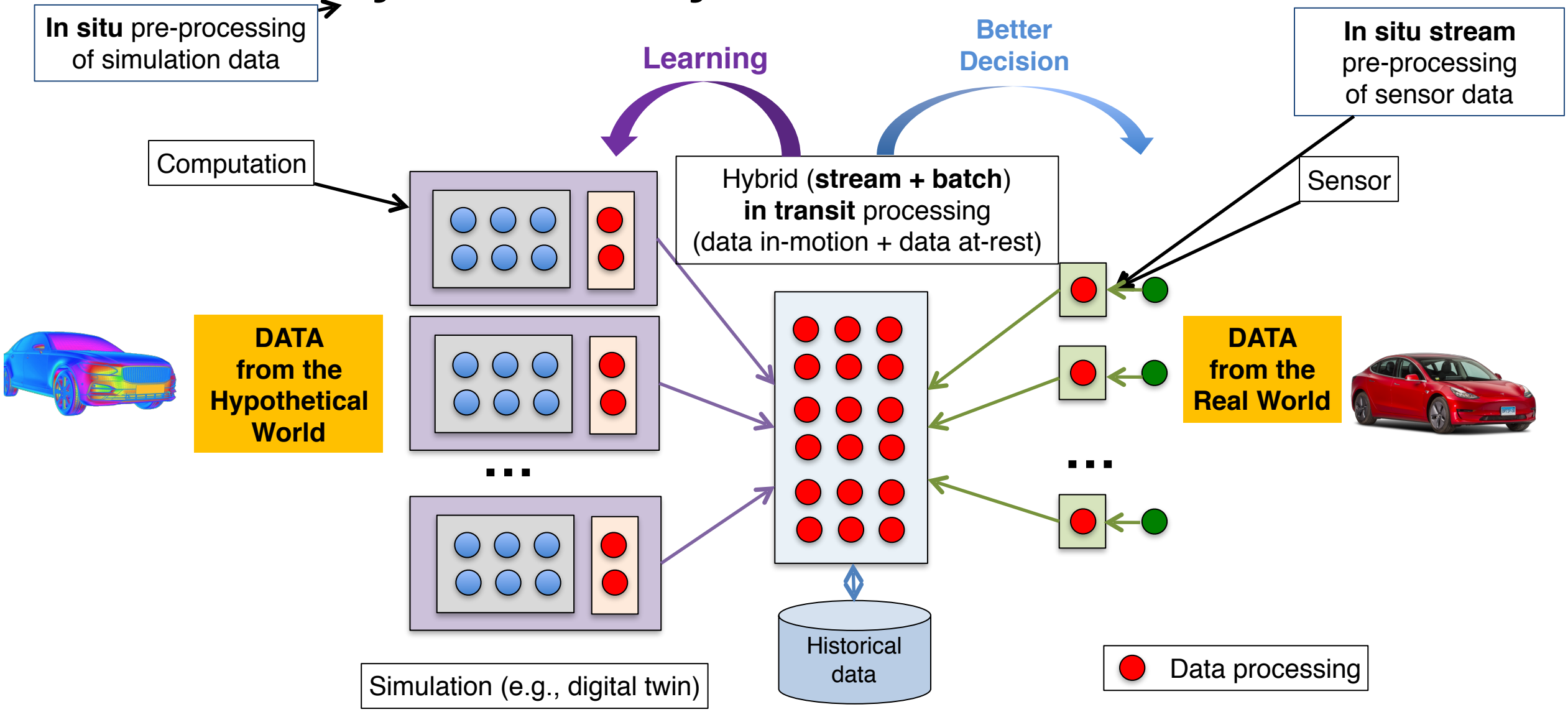
Big Data analytics

Data driven model

Our vision: hybrid analytics architecture



Hybrid analytics architecture



Hybrid analytics architecture

In situ pre-processing of simulation data

Learning

Better Decision

In situ stream pre-processing of sensor data

Computation

Hybrid (**stream + batch**)

Sensor

DATA from the Real World

Postdoc (ANR OverFlow project)

- Investigating Edge vs. Cloud computing trade-offs for stream processing
- Methodology for benchmarking Edge processing frameworks

Pedro Silva

Ph.D. (co-supervised with UPB)

- Uniform Cloud and Edge stream processing for fast Big Data analytics

DataSteward++

+ Edge analytics (e.g., data aggregations)
+ uniform Edge/Cloud processing

Hybrid analytics architecture

In situ pre-processing of simulation data

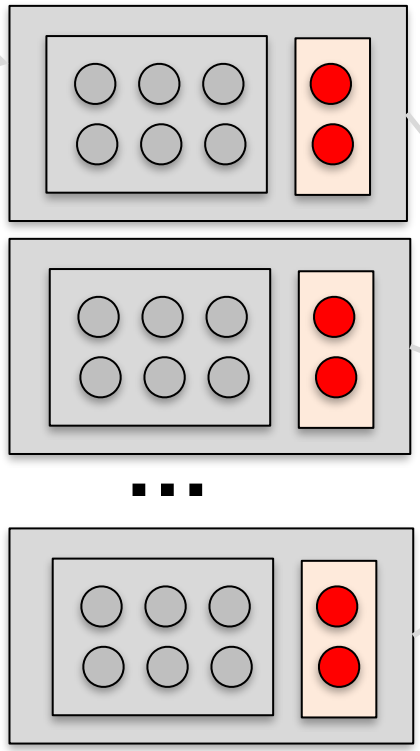
Learning

Better Decision

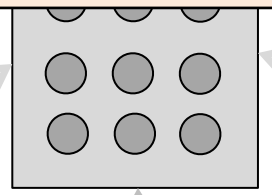
In situ stream pre-processing of sensor data

Computation

DATA from the Hypothetical World



Simulation (e.g., digital twin)



Historical data

Research Engineer (ADT project)

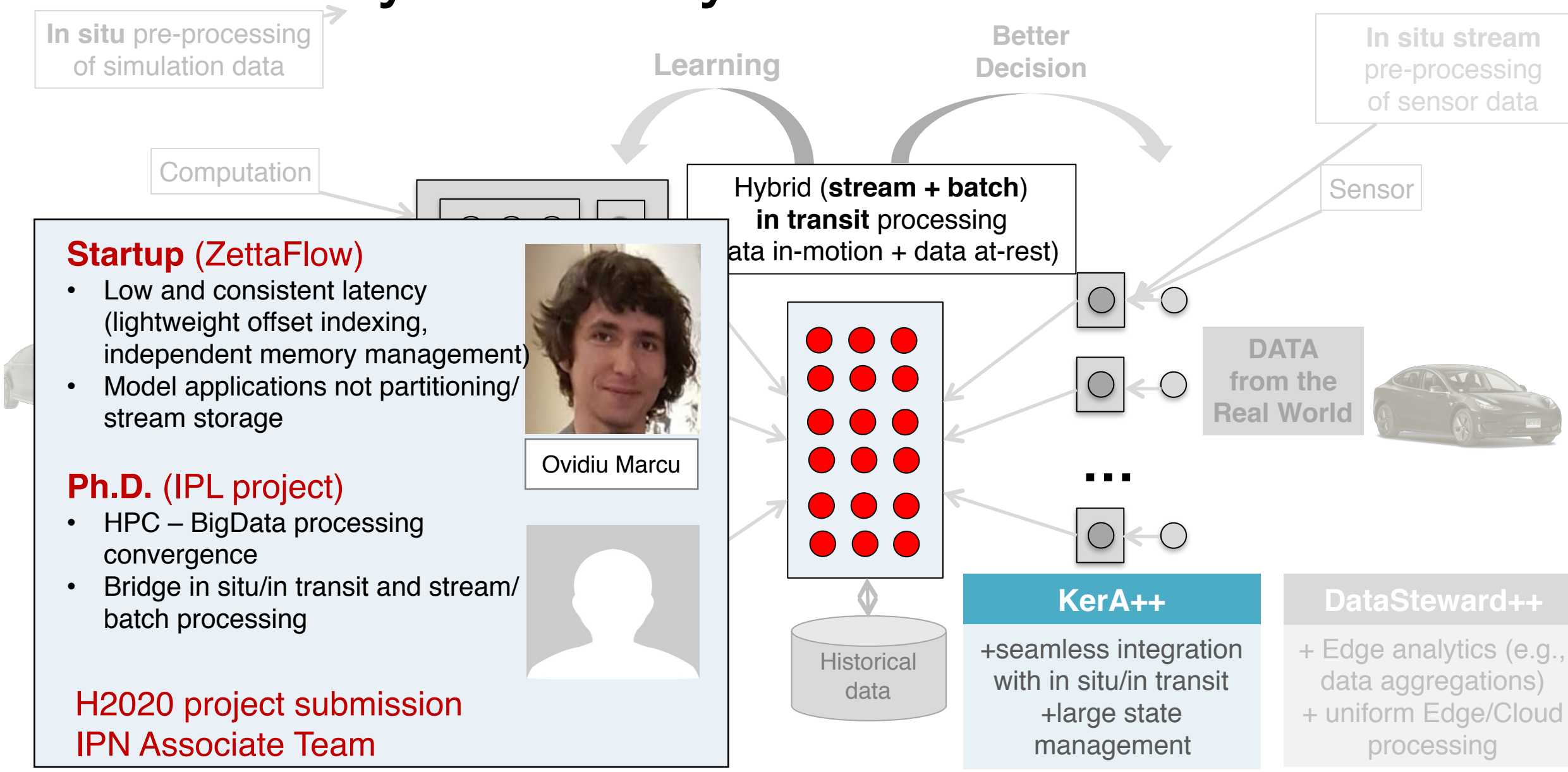
- Enable support for in situ Big Data analytics
- Elastic allocation of dedicated resources (cores/nodes)

Ovidiu Marcu

DataSteward++

+ Edge analytics (e.g., data aggregations)
+ uniform Edge/Cloud processing

Hybrid analytics architecture



Startup (ZettaFlow)

- Low and consistent latency (lightweight offset indexing, independent memory management)
- Model applications not partitioning/ stream storage

Ovidiu Marcu

Ph.D. (IPL project)

- HPC – BigData processing convergence
- Bridge in situ/in transit and stream/ batch processing

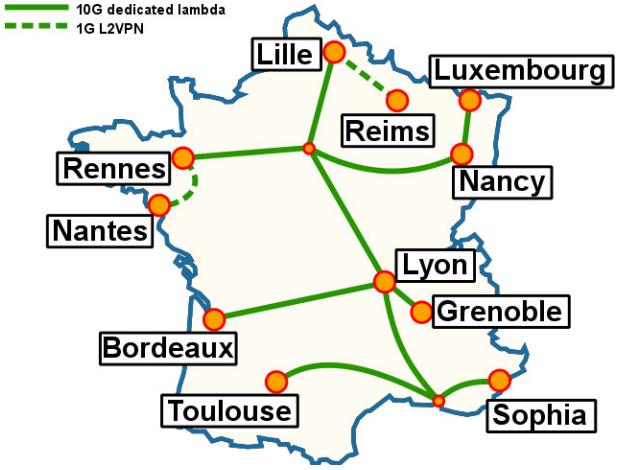
H2020 project submission
IPN Associate Team

My scientific methodology

- Analyze trends and **state of the art**
- Intuition
- Identify **realistic use cases**
- Define research questions
- Develop a **real piece of software**
- Evaluate research questions with **synthetic benchmarks**
- Evaluate research questions with **real-life use cases**

Platforms

- Academic testbeds and supercomputers



Grid'5000

Chameleon

- Public clouds

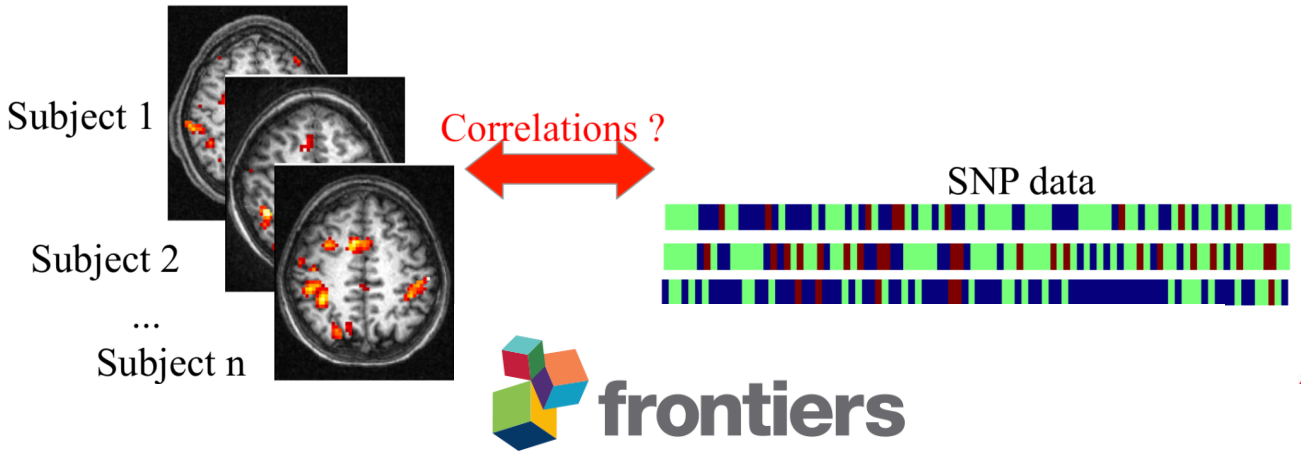
Microsoft Azure

amazon web services

Impact: interdisciplinarity

Contributions to **healthcare**

- **DataSteward** and **JetStream** used to prove for the first time the correlation between brain regions and genetic data
- Enables early diagnostic of psychiatric illnesses



frontiers in Neuroscience

Formal dialogue with the **HPC community**

- Member of the Big Data Value Association (BDVA)
- Contributions to the joint white paper with the European Association for HPC (ETP4HPC)

ETP4HPC EUROPEAN TECHNOLOGY PLATFORM FOR HIGH PERFORMANCE COMPUTING

BDV BIG DATA VALUE ASSOCIATION

Impact: industry

Transfer

- **JetStream** integrated in Microsoft Azure for control message transfers
- Azure SignalR provides real-time functionality using several dedicated connections – inspired by **DataSteward**
- Huawei studies **KerA** for potential integration in the stream layer of the Huawei Cloud

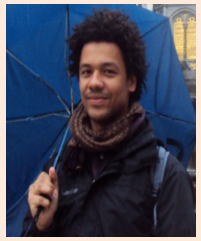
Startup: **ZettaFlow**

- Fast Big Data stream ingestion to power real-time applications

No one else is creating software services specifically for science
Otherwise, we must adapt/adopt other solutions

Luc Bougé

Gabriel Antoniu

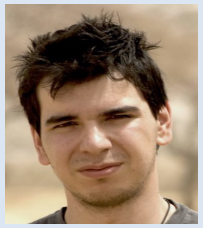


Pedro Silva

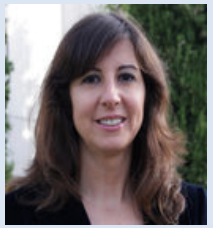
Luis Pineda

Ovidiu Marcu

Inria
inventors for the digital world



Pierre Matri



Maria S. Perez

UNIVERSIDAD POLITÉCNICA MADRID
POLITÉCNICA

Thank You!

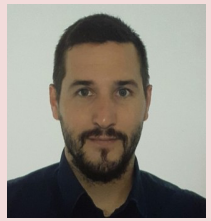
Rob Ross

Bogdan Nicolae

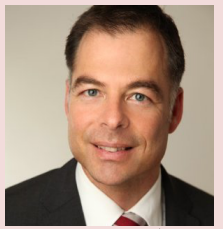
Kate Keahey

Argonne
NATIONAL LABORATORY

Radu Tudoran



Stefano Bortoli



Goetz Brasche

HUAWEI