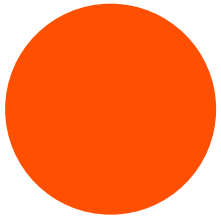
CRITEO

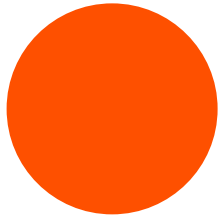
Streaming 101

Ilyas Toumlilt
i.toumlilt@criteo.com

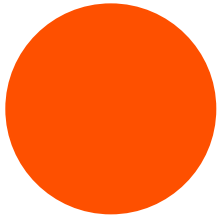
Session 1 – 07/02/2024



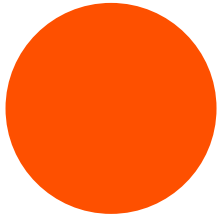
Survey



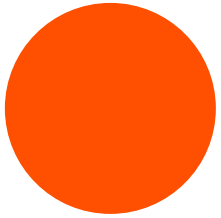
Reviewing CS Basics



☕☕☕☕☕



Data-intensive streaming systems



Streaming Demo

What should I expect?

Subtitle

- 7 study days: Streaming, Kafka, Flink and company
- Strong foundation with (almost real) practical labs
- An email a day before class about what you will learn

Source:

What I will learn

Introduction to Streaming (101)	
Introduction to Messaging Systems	
Deep Architecture Kafka	
Kafka Lab Exam (Compulsory)	
Distributed Consensus Algorithms	
Introduction to Flink	
Flink High-Level Customisation	
Final Exam (Compulsory)	

Source:

References

- Designing Data-Intensive Applications – Martin Kleppmann
- Kafka: The Definitive Guide – Gwen Shapira et al.
- Stream Processing with Apache Flink – Fabian Hueske

CRITEO

Let’s start with a survey

CRITEO

Reviewing CS Basics

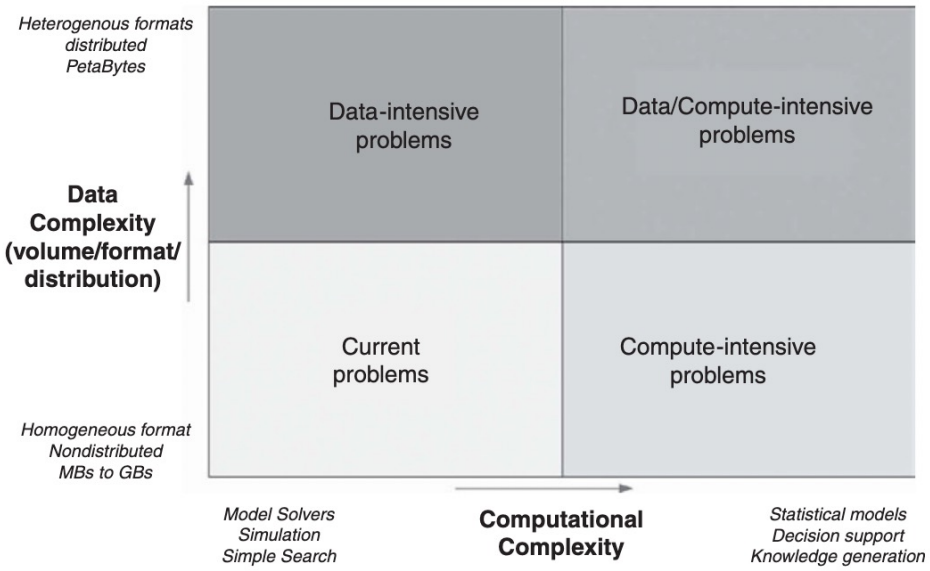
Computing in General

- Data-intensive computing

  e.g. HDFS, SPARK, KAFKA, PRESTO ,FLINK etc.

- Compute-intensive computing

  e.g.HPC, OPENMP, MPI



Gorton, Ian, and Deborah K. Gracio, eds. Data-intensive computing: architectures, algorithms, and applications. Cambridge University Press, 2012.

Data-driving decision making

- Information Theory: Data is a frozen information.
- Data drives decisions not every day, every sec!
- Decisions could be make by humans or by software

Real Numbers

- CERN LHC detectors generates 300 GBps [1]
- The New York Stock Exchange generates about 4-5 terabytes of data per day [2]
- The Internet Archive stores around 18.5 petabytes of data[3]

[1] https://wlcg.web.cern.ch/

[2] http://bit.ly/nyse_data_deluge

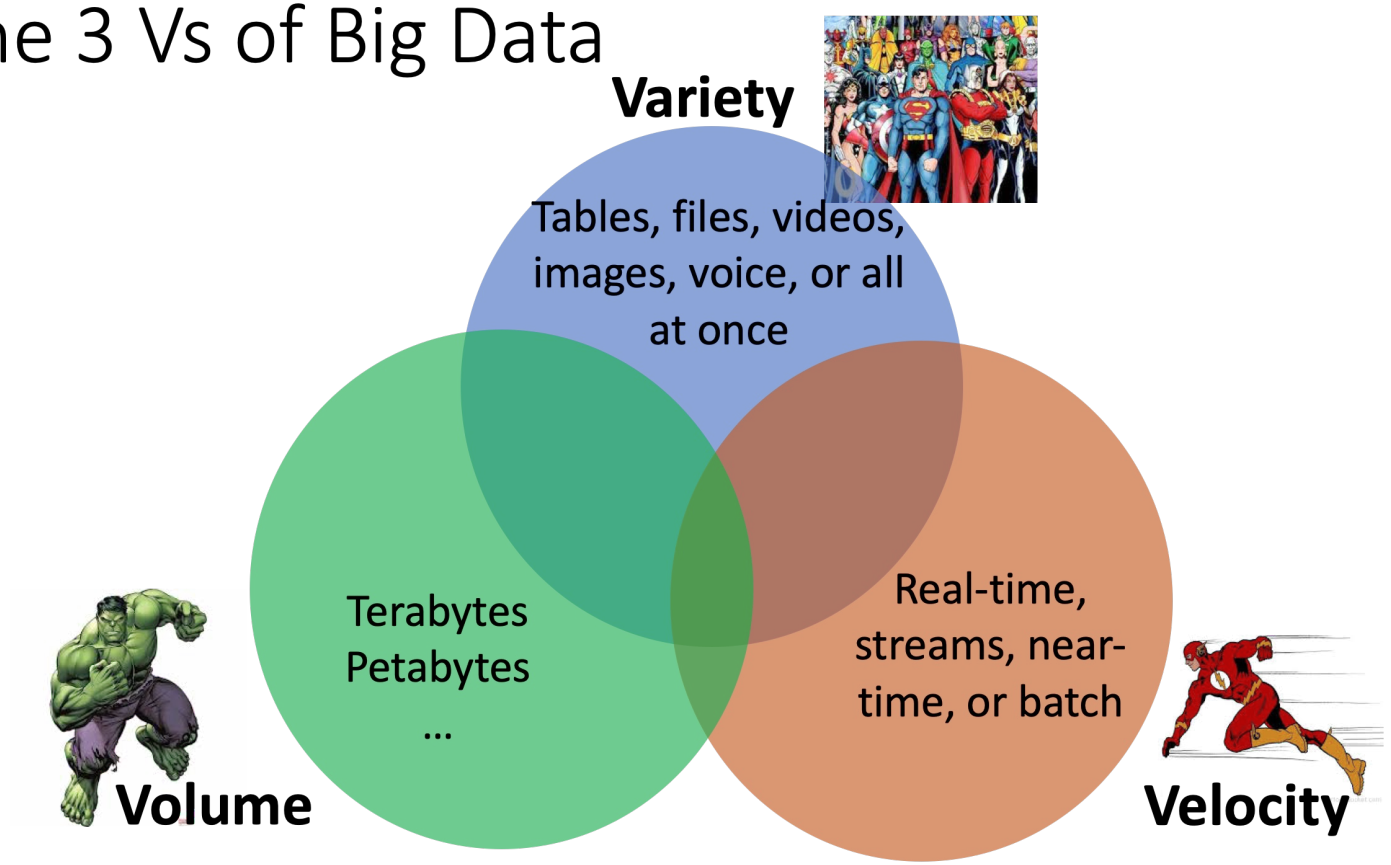
[3] https://archive.org/web/petabox.php

Problems of Data-intensive Distributed Computing

- The programming model: MapReduce
- Reliability and availability constraints
- Scalability
- Data locality

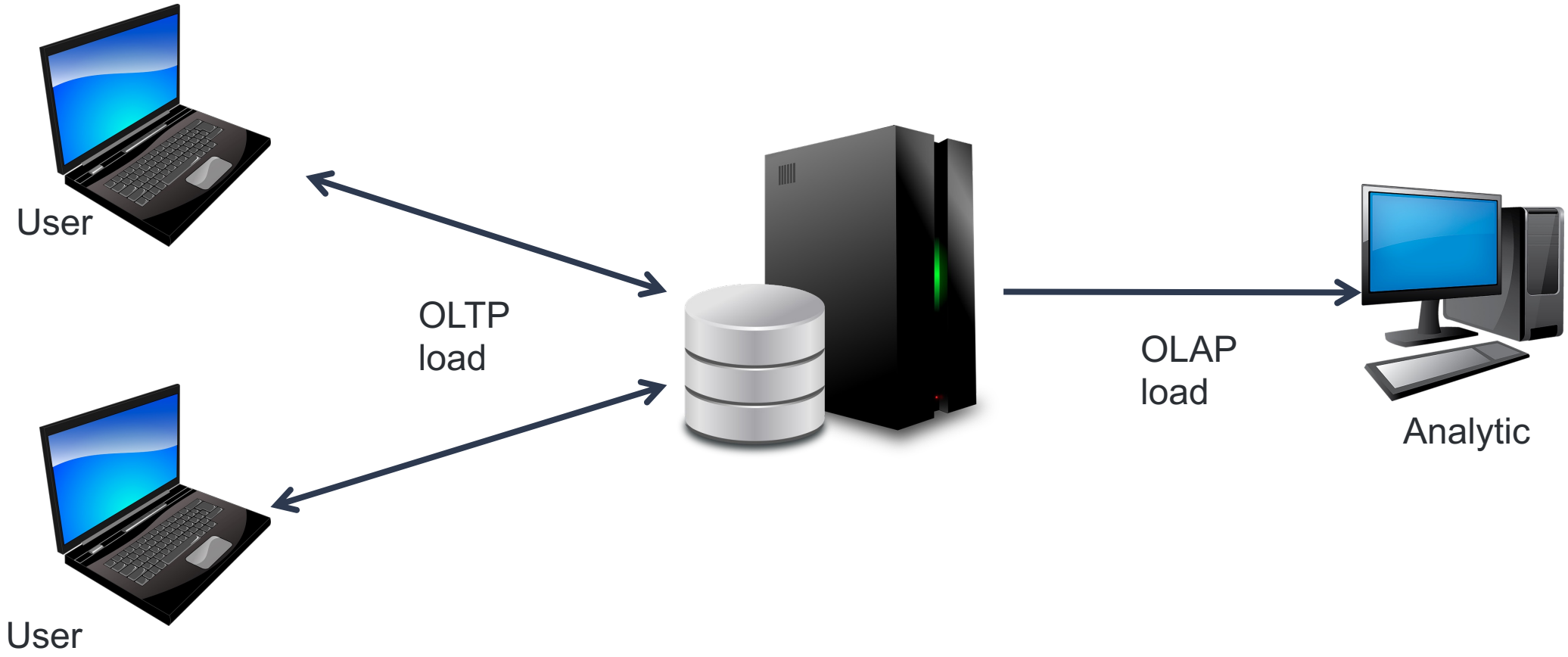
Dimensions of Data-intensive Computing

The 3 Vs of Big Data



CRITEO

Data systems evolution

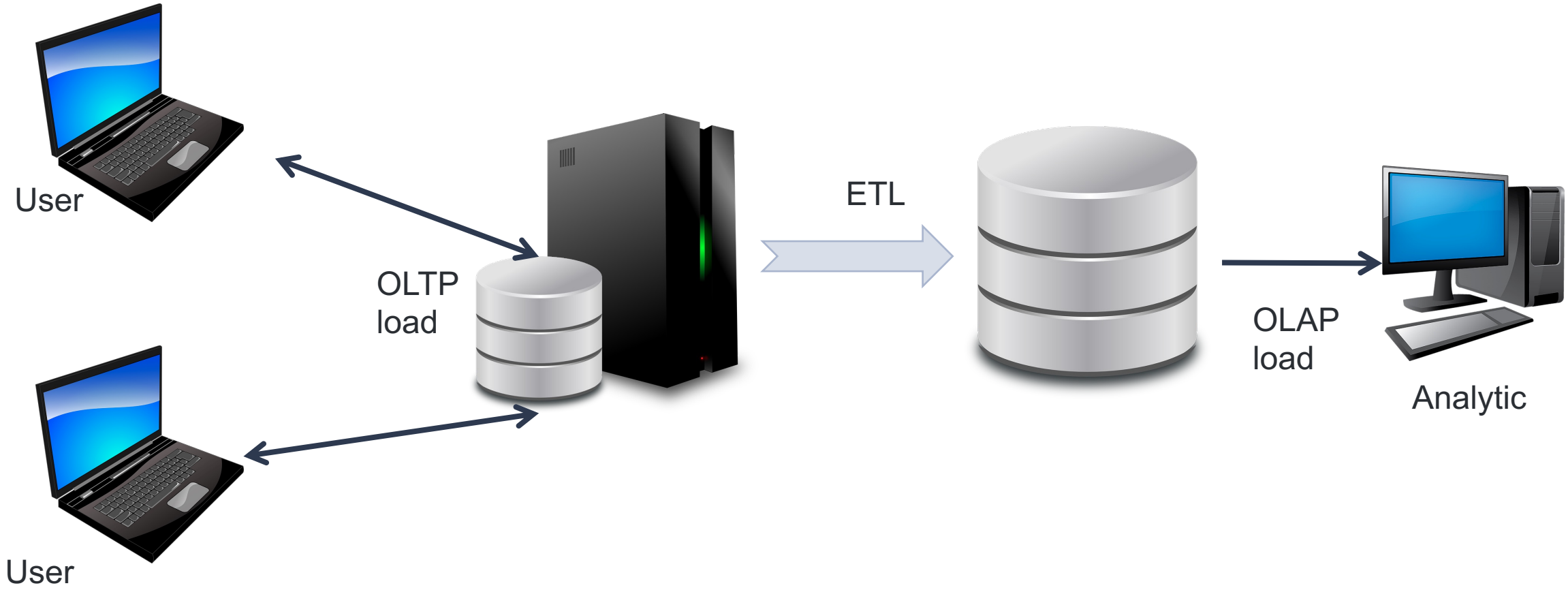


CRITEO

OLAP vs OLTP

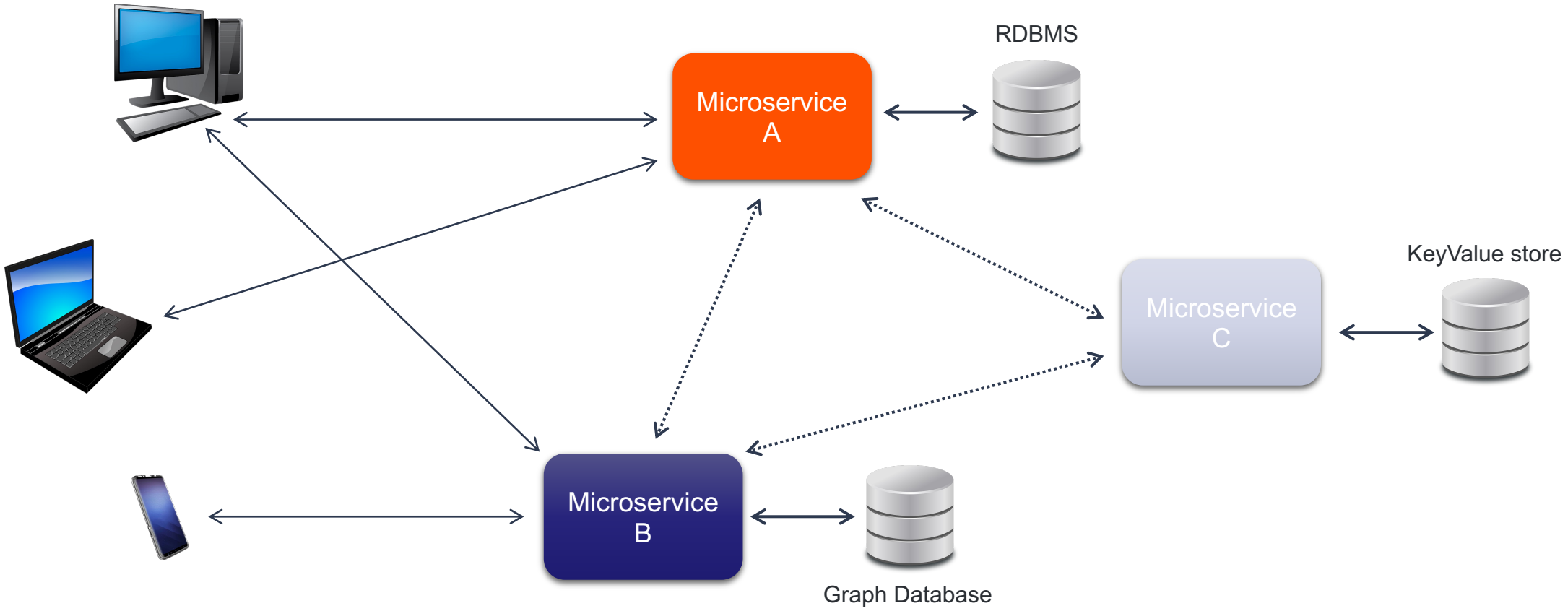
	OLAP	OLTP
Characteristics & Query Types	Complex involves more joins	Relatively simple
Response time	Minutes maybe even hours	Milliseconds level
Constraints	Relaxed integrity constraints (not normalized)	Database follows integrity constraint (normalized)

Data systems evolution



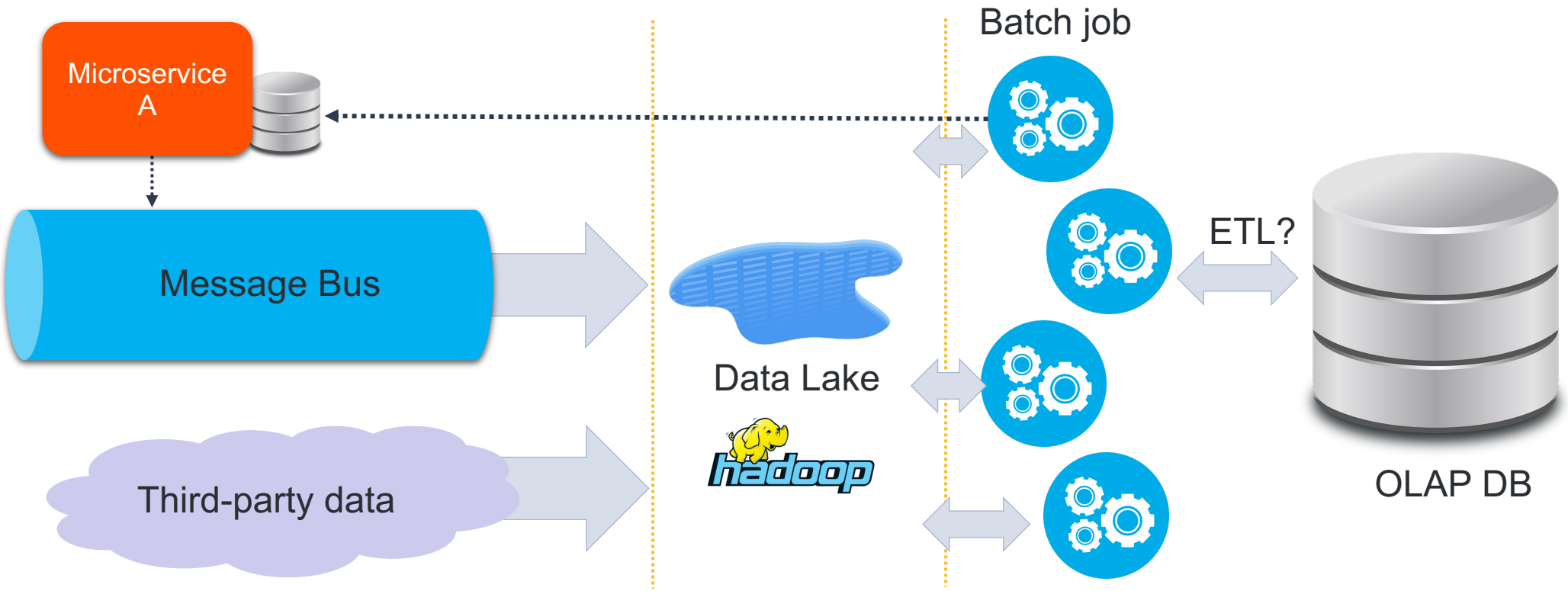
CRITEO

Microservices



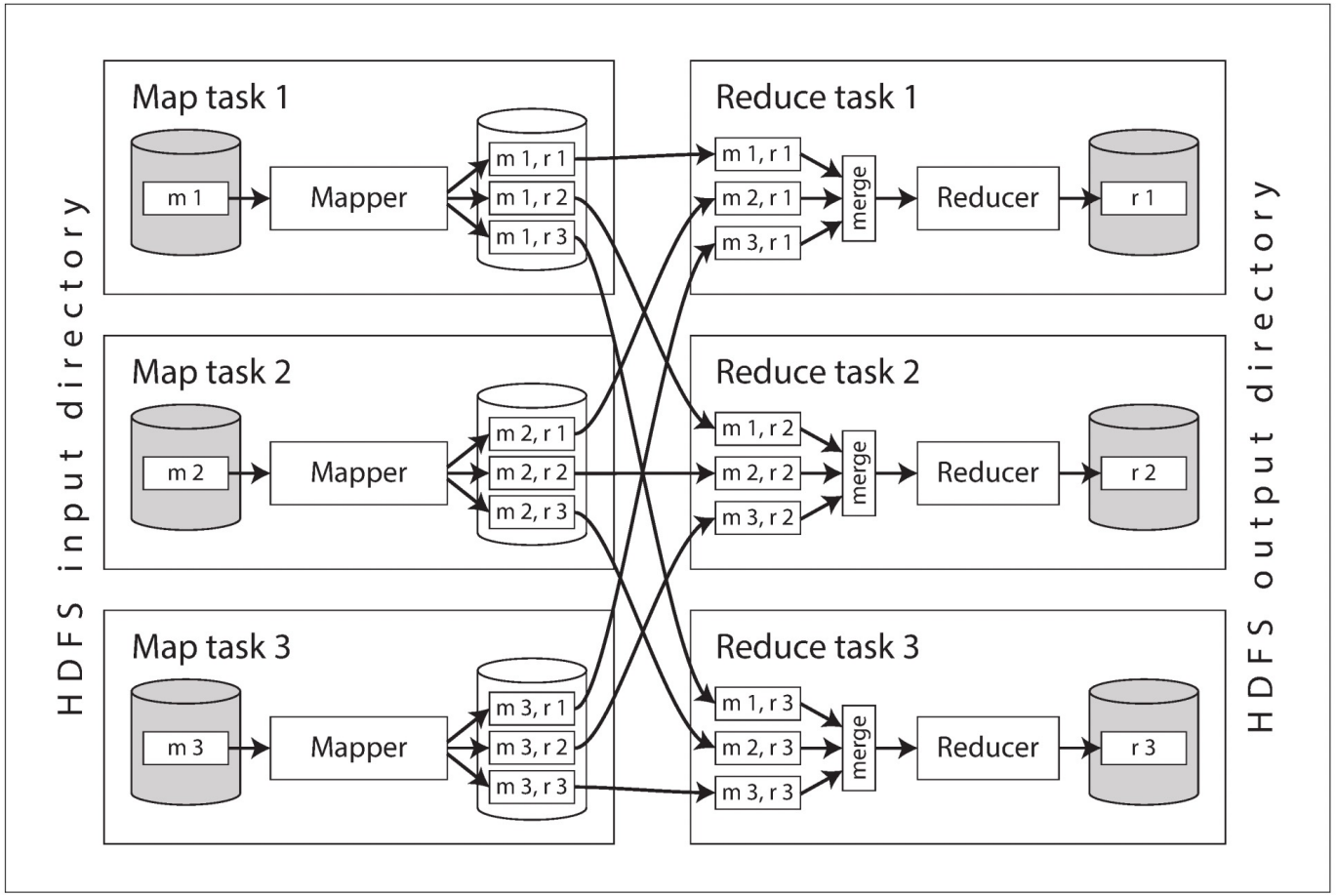
CRITEO

Batch Processing



CRITEO

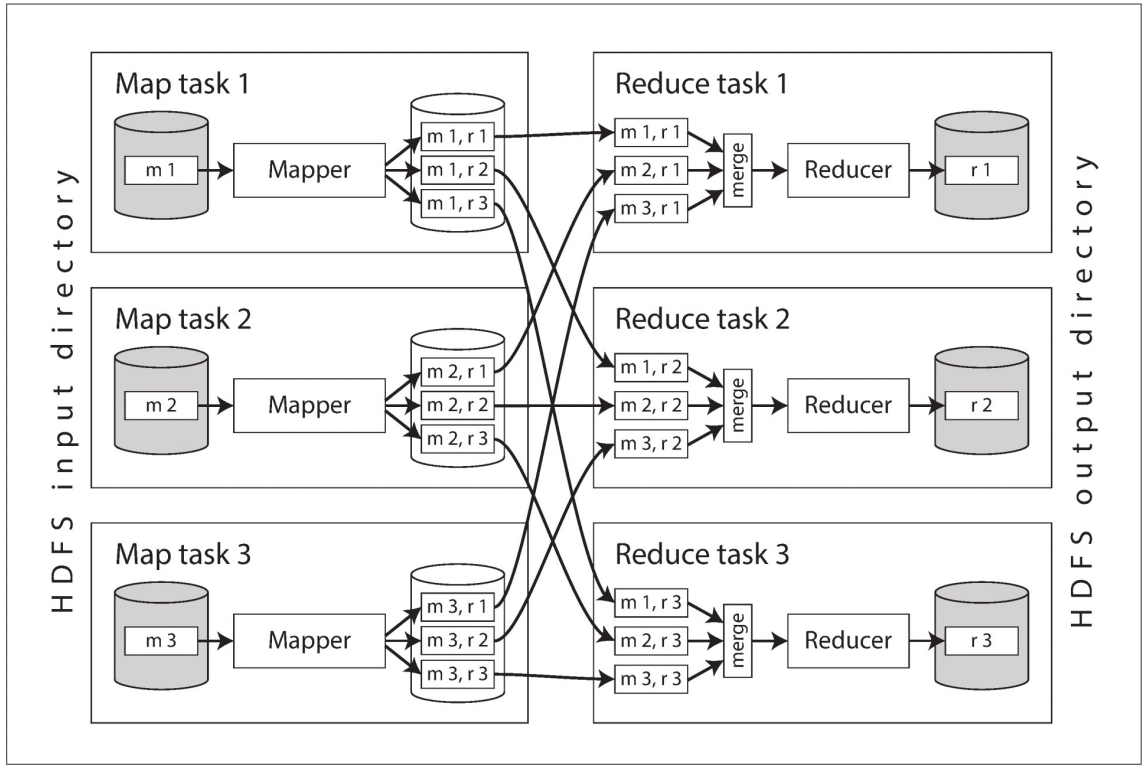
Batch Processing(MapReduce)



CRITEO

Well-known MR Problems

- Handling skew
- Map-side joins
- Broadcast hash joins



CRITEO

Real Use-case:

- Daily-batch, analysing production logs.
- Building search index
- Building machine learning algorithms
- Recommender systems algorithms

http://wiki.apache.org/hadoop/PoweredBy

Streaming

- "A type of data processing engine that is designed with infinite datasets in mind."
- Streaming systems may have lower latency than batch jobs, but this may be seen as a grateful side effect. In fact, streaming systems are made to deal with endless datasets by design.
- This means that even very small batches may be seen as a streaming system.

Streaming Formal Definition

$$stream(t) = \frac{\mathrm{d}\, state(t)}{\mathrm{d}t}$$

$$state(now) = \int_{t=0}^{now} stream(t)\ \mathrm{d}t$$

CRITEO

Streaming 101

Streaming vs Batch Processing

Problems arise in streaming: *time, windows, state…*

Common distributed system problems: *Dumb ways to die*

Real classic problem

You are the owner of an application (imagine any app, remember flappy bird?) in production

It's tested, up and running with a workload of 100 users

Big buzz came and you are getting +100.000 active users

Your clients say the response time is increasing... 😕

Where to start looking?

Real classic problem

You are the owner of an application (imagine any app, remember flappy bird?) in production

It's tested, up and running with a workload of 100 users

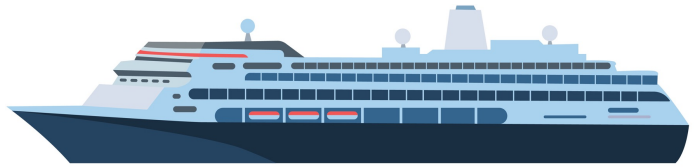
Big buzz came and you are getting +100.000 active users

Your clients say the response time is increasing... 😕

Where to start looking?

Resource Usage

Vertical scaling



The simplest approach to scale to higher load is to buy more powerful machines

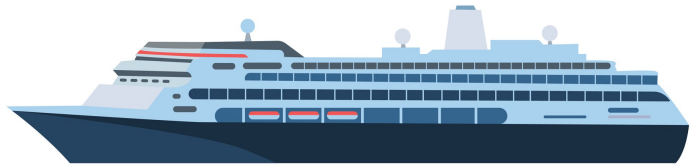
- vertical scaling or scaling up

Many CPUs, many RAM chips, and many disks can be joined together under one operating system, and a fast interconnect allows any CPU to access any part of the memory or disk.

- In this kind of shared-memory architecture, all the components together can be treated as a single machine.

Photo: Designed by pch.vector / Freepik

Vertical scaling

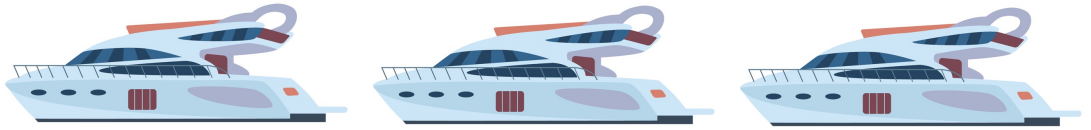


The problem with a shared-memory approach is that the cost grows faster than linearly

- A machine with twice as many CPUs, RAM, and disk capacity costs significantly more than twice the price.
- Due to bottlenecks, a machine twice the size cannot necessarily handle twice the load.

A shared-memory architecture may offer limited fault tolerance.

Horizontal scaling



In a shared-nothing architecture multiple machines are used.

- horizontal scaling or scaling out

Each node uses its CPUs, RAM, and disks independently.

No special hardware is required.

Any coordination between nodes is done at the software level, through the network.

While a distributed shared-nothing architecture has many advantages, it adds more complexity.

More machines may lead to more failures too.

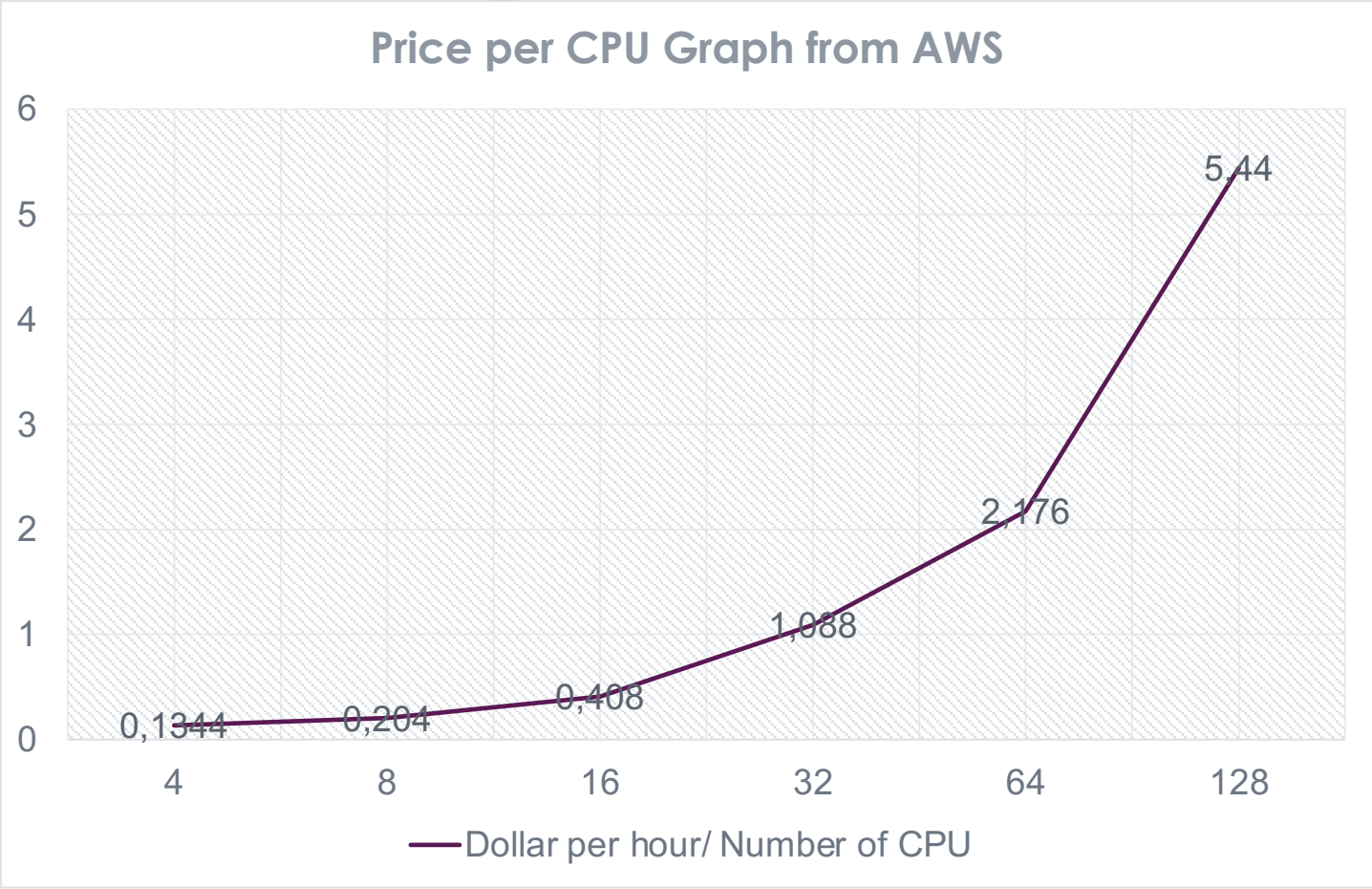
What about a break? ☕

CRITEO

Distributed Systems

- There are several reasons why you might want to distribute your data across multiple machines:
  - Scalability
  - Fault tolerance/high availability
  - Latency
  - Price

What about pricing



CRITEO

Replication Versus Partitioning

There are two common ways data is distributed across multiple nodes:

- Replication:
  - Keeping a copy of the same data on several different nodes, potentially in different locations.
  - Replication provides redundancy: if some nodes are unavailable, the data can still be served from the remaining nodes.
  - Replication can also help to improve performance(in general).
- Partitioning (*sharding*) e.g. partition by Country
  - Splitting a big data into smaller subsets called partitions so that different partitions can be assigned to different nodes.

Unbounded data streams

- Unbounded data is an ever-growing, essentially infinite data set.
- It reflects the reality in a much natural way.
- In a batch world, the data must be finite, with a beginning and an end. Usually this is defined by partitions (by hour, by client etc).
- Unbounded data means there will always be new data arriving.
- The rate of new data usually is non-deterministic.
- We can think a bounded dataset as a subset of the unbounded dataset.

Let's talk about events!

- Events are stored in the message system one after the other.
- The stream of events can also be seen as a log of messages.
- Each message describe an event and contains the informations required to be processed:

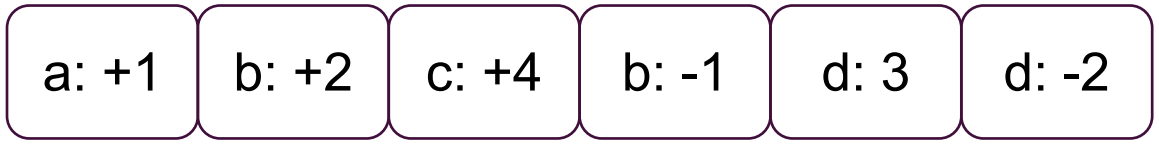
  User 1 added the product X to the shopping cart;

  User 2 logged in;

  Sensor Y recorded the temperature Z;

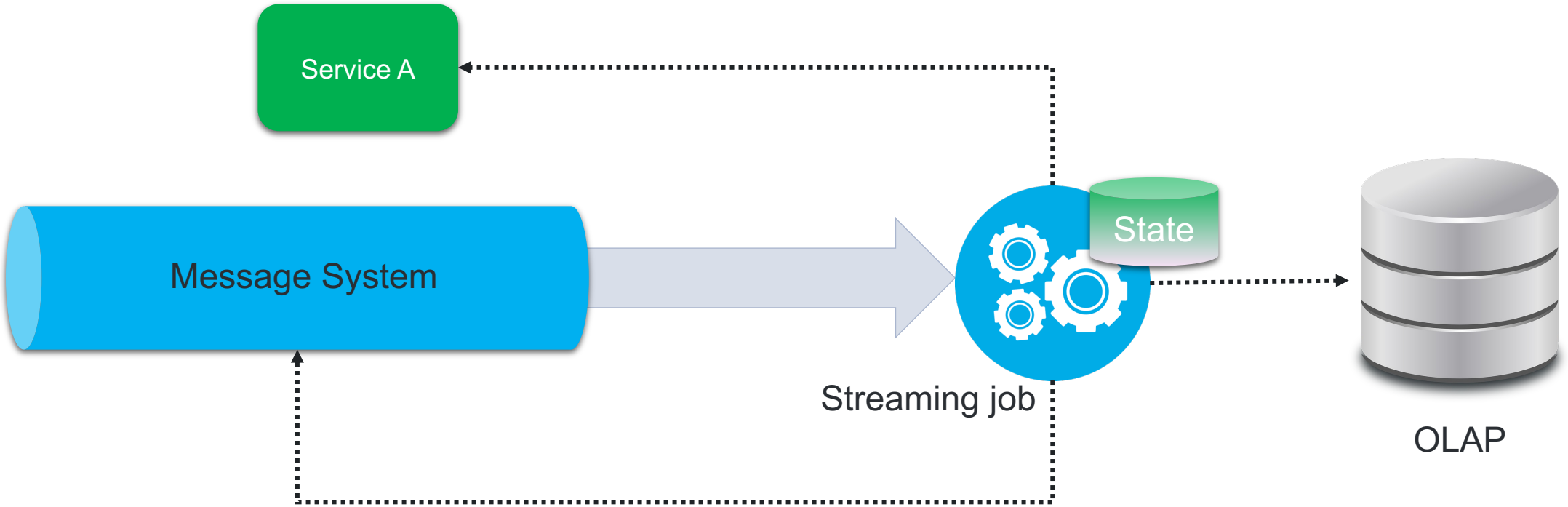
  User 4 paused the video X at the position Y.

Log - Table Duality



a	1	1
b	2-1	1
c	4	4
d	3-2	1

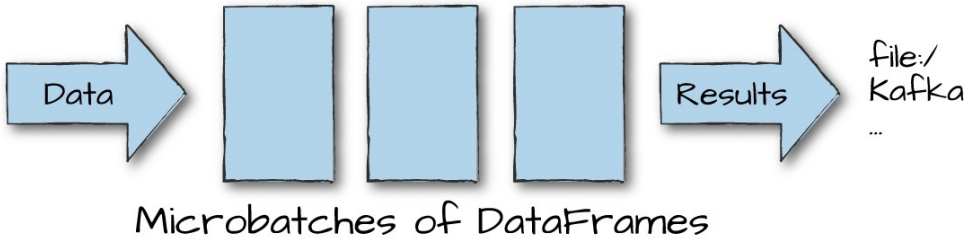
Stream Processing



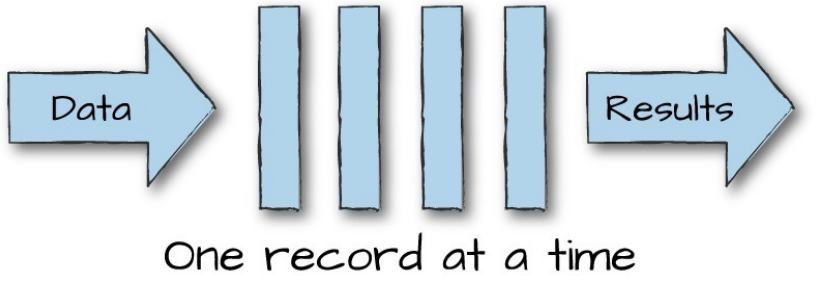
CRITEO

Stream Processing Patterns

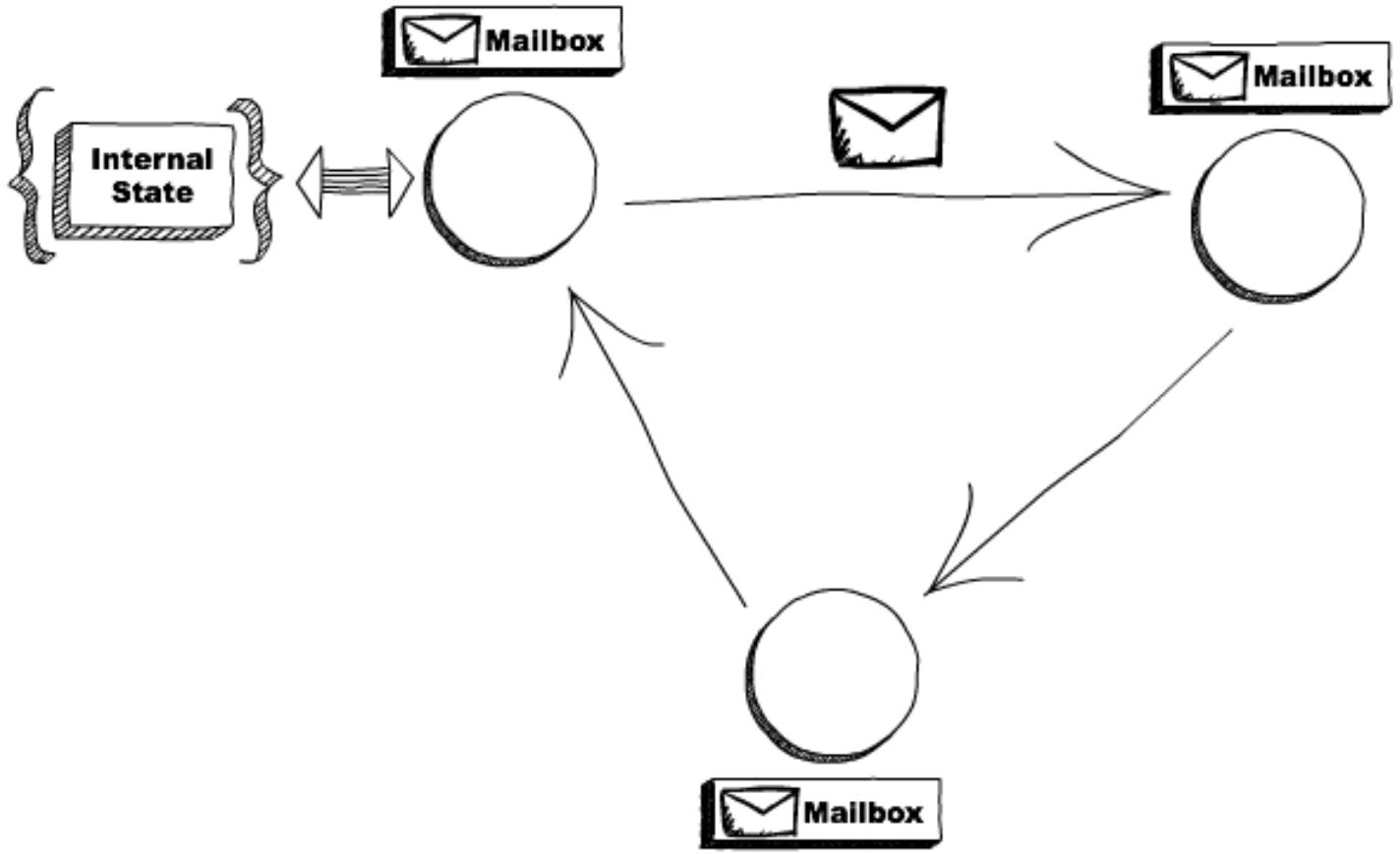
- Micro-batch systems



- Continuous processing-based systems



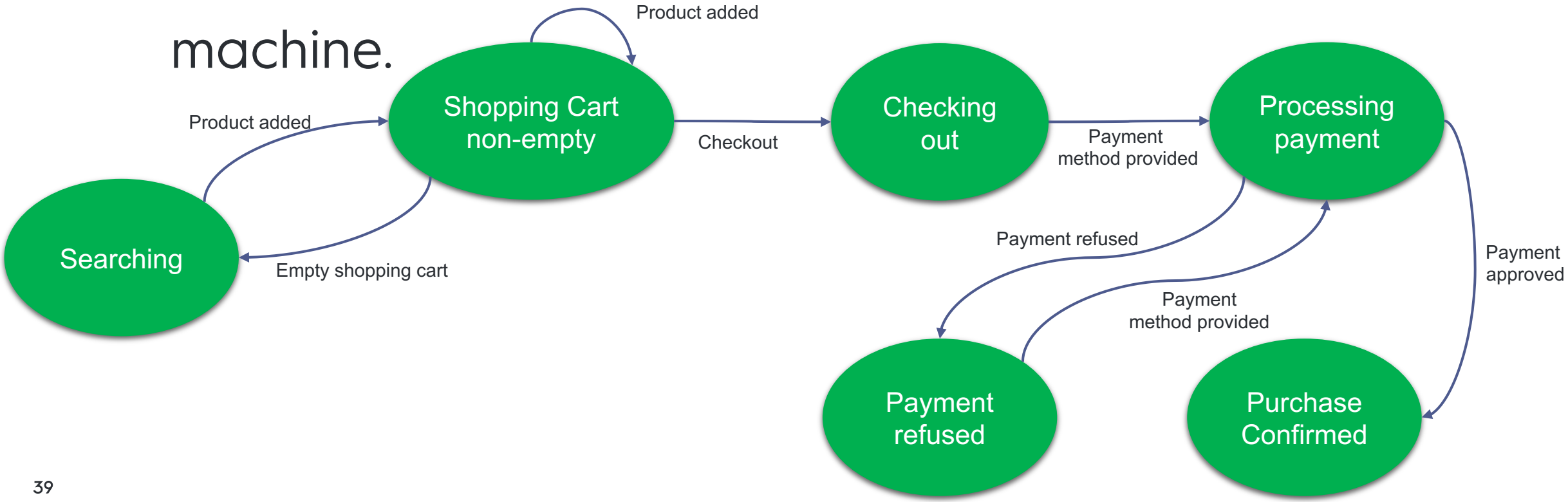
Streaming and Actor Models



CRITEO

Finite State Machine

- A stream of events can be used to feed a state machine.



CRITEO

Stream Processing vs Batch Processing

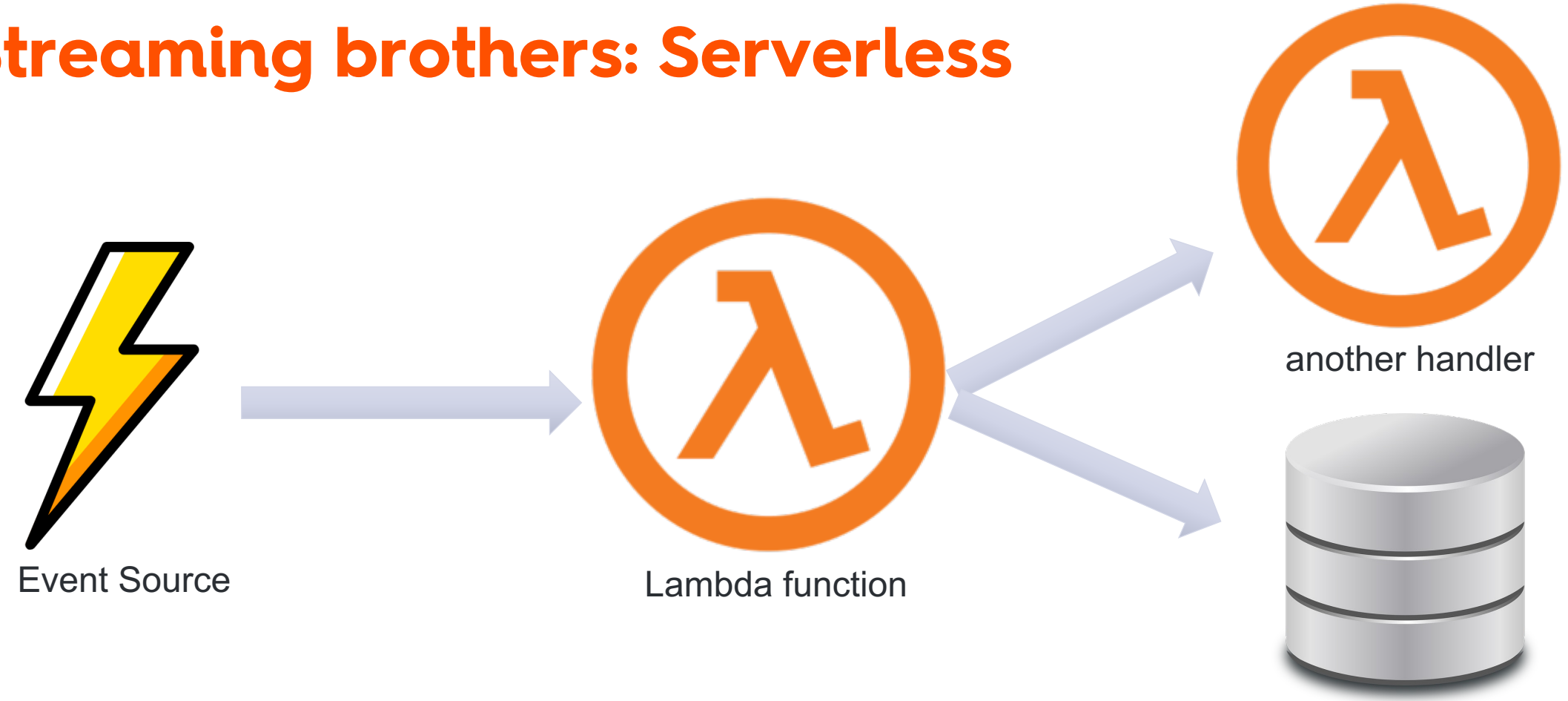
Batch

- Event time processing is for free
- Optimizations built on plan of executions
- Batches allows to catch up faster
- Scales

Streaming

- Less moving components
- Flexible data partitioning and windowing
- Less overhead -> reactivity
- Faster feedback loop during the development
- Time based joins are easy

Streaming brothers: Serverless



CRITEO

Streaming Details

Distributed Log storage Zoo

- Apache Kafka
- Apache Pulsar
- Pravega
- Redpanda

Problems arise in Streaming

- Time
- Windows and late messages
- State

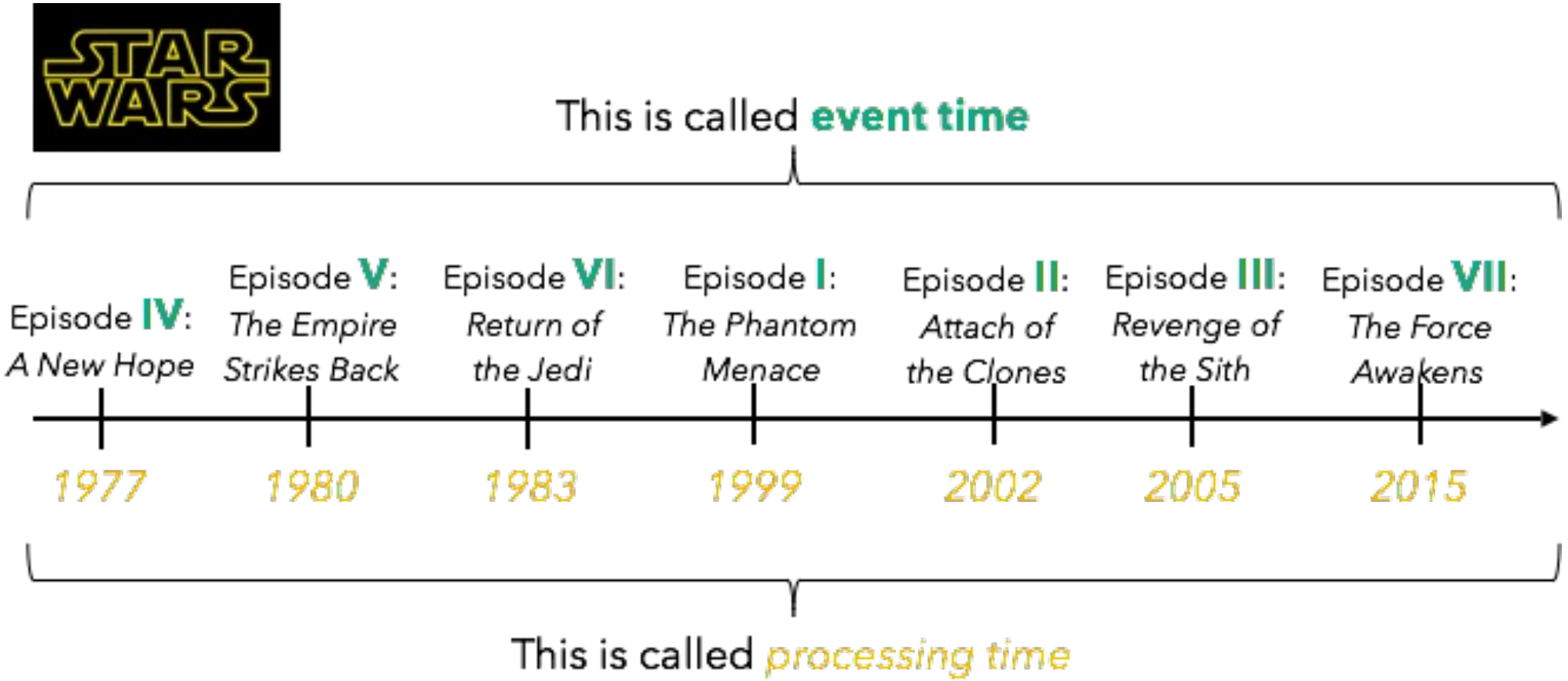
Time

- What is being computed?
- Where in event time?
- When in processing time?
- How do refinements relate?

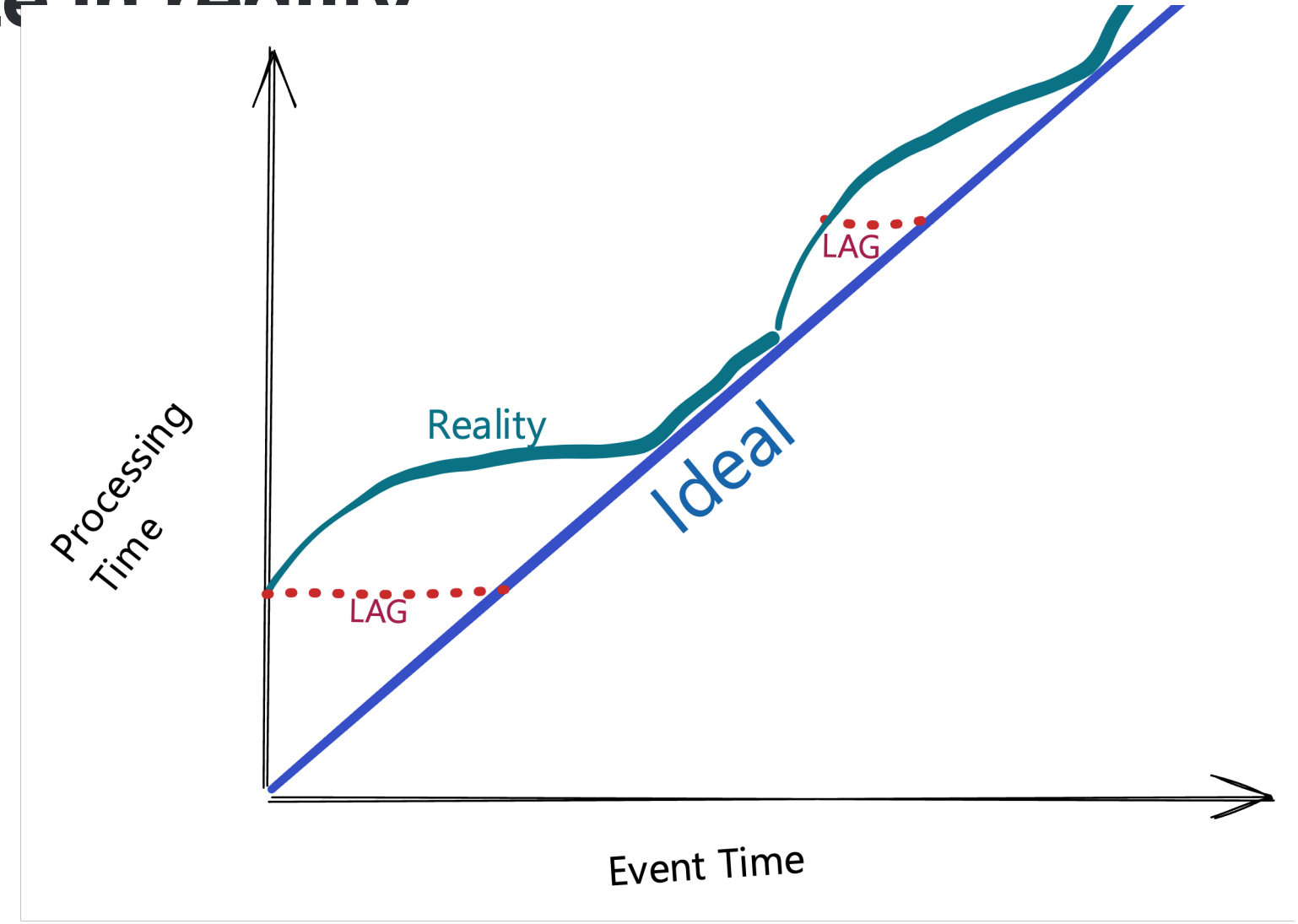
Time

- **Event time** - the time at which events actually occurred.
- **Processing time** - the time at which events are processed by the application.
- **Ingestion time** - the time at which events entered the message system.
- Not all use cases care about Event time.
- In an ideal world, Event time and Processing time would always be equal, with events being processed immediately as they occur.

Event time vs processing time

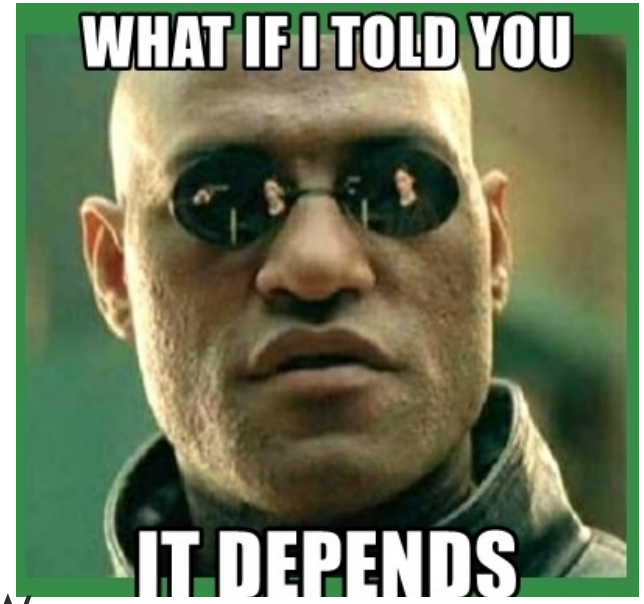


Time in reality



How much we should care about time ?

- Time-agnostic jobs:
  filtering, joins,map-only
- Approximation algorithms:
  top-N, streaming K means
- Windowing:
  fixed windows, sliding window, session window



CRITEO

Time Windows