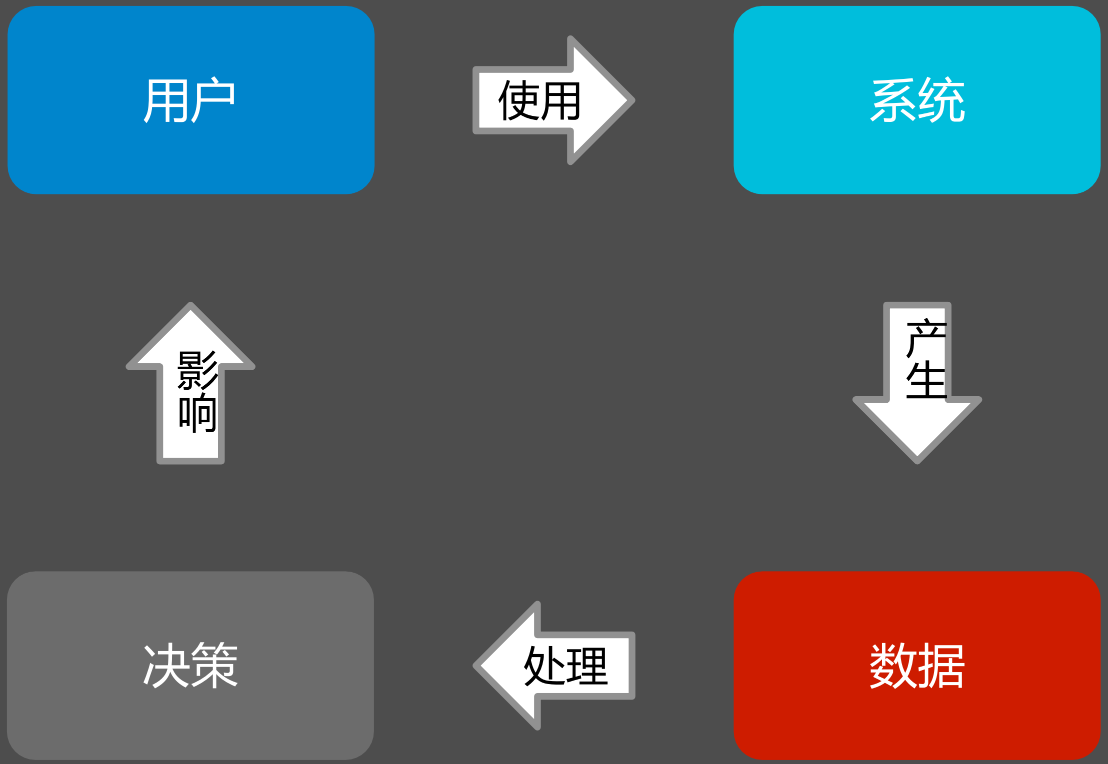
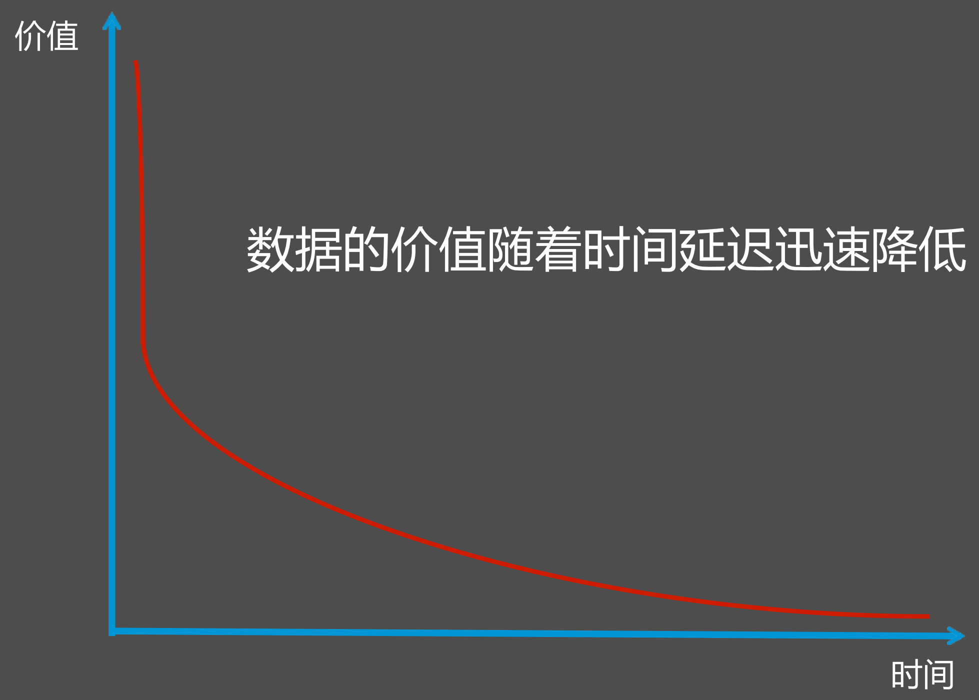
初识流计算Streaming

A Whirlwind Tour of Streaming Data Processing

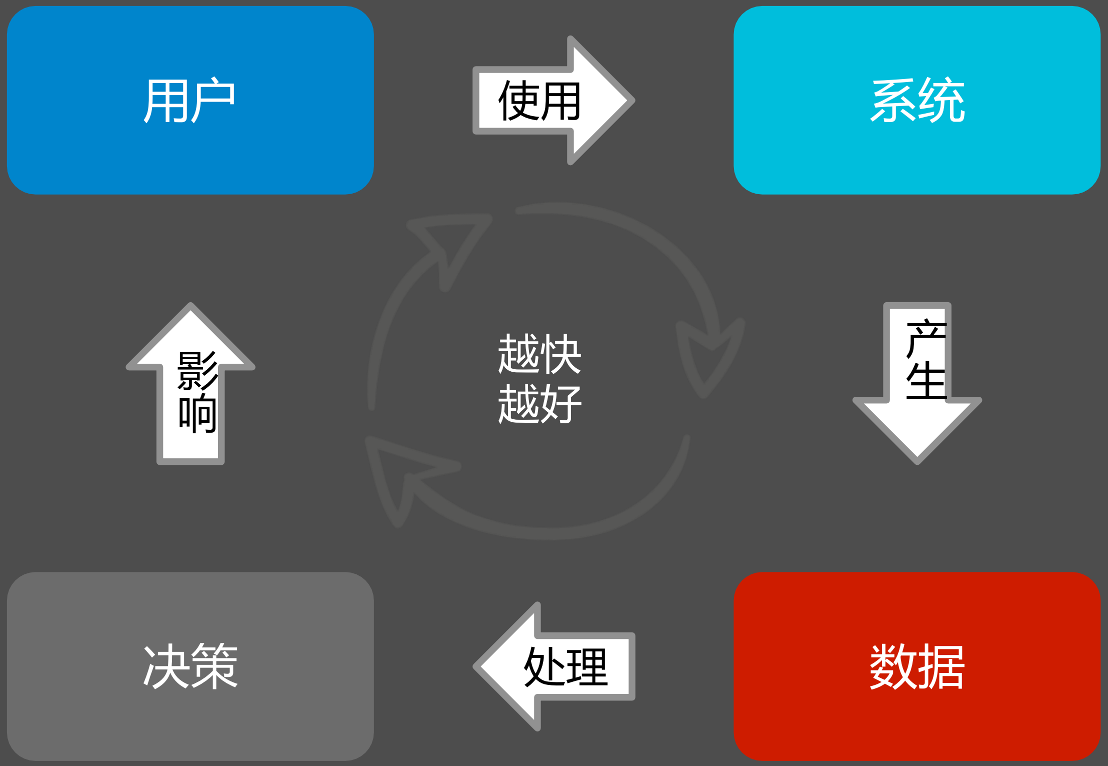


商业和数据形成闭环

业务数据



价值时效

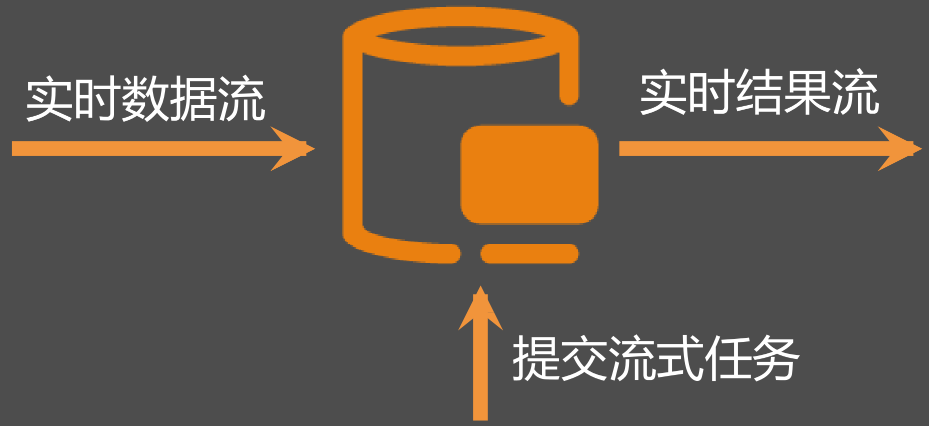


越快越有竞争优势
大数据实时化——流式处理

离线（批量）计算

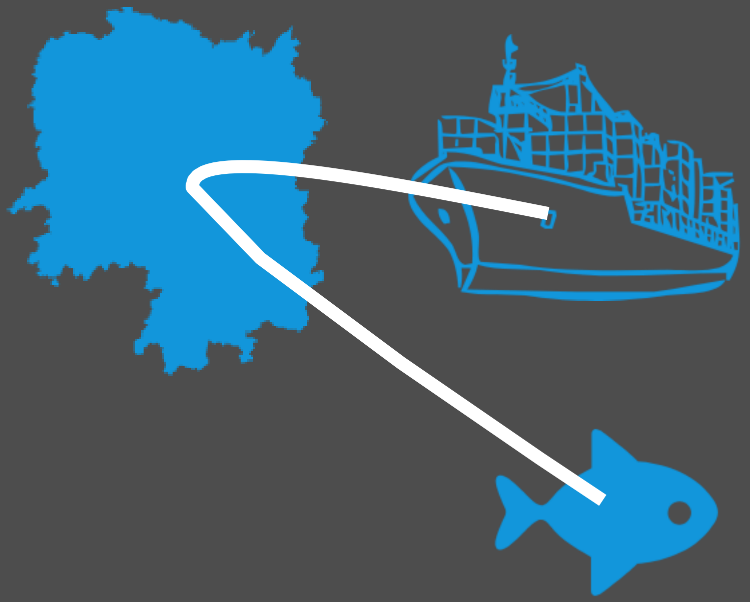
批量计算是一种批量、高时延、主动发起的计算任务

流计算



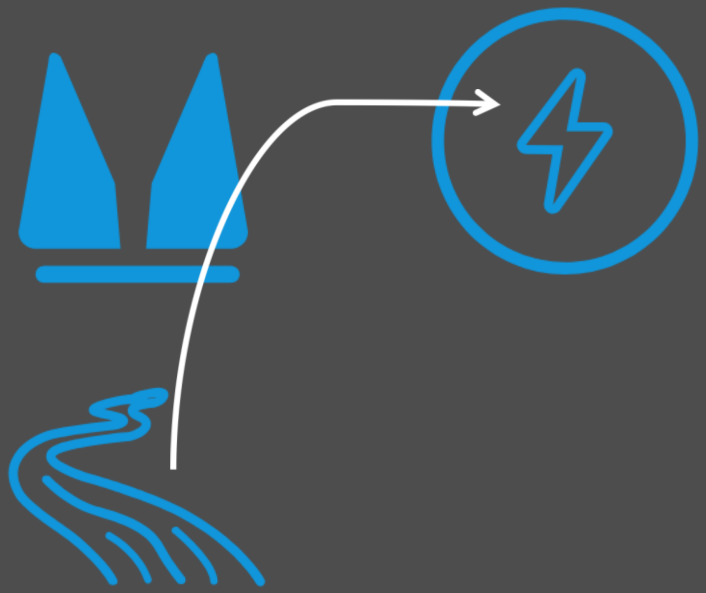
流计算是一种持续、低时延、事件触发的计算任务

离线（批量）计算



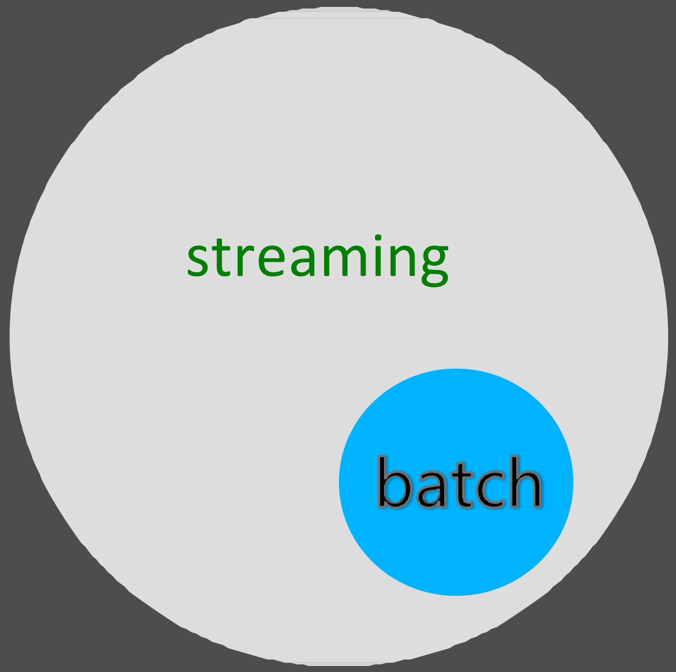
开船去湖里打鱼

流计算

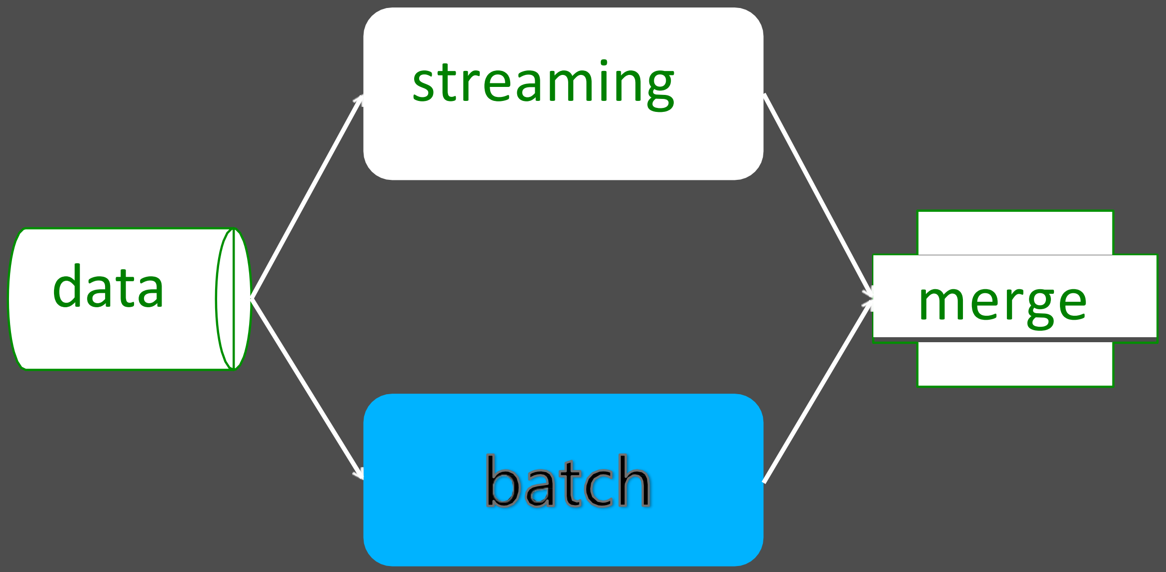


拦河建坝发电

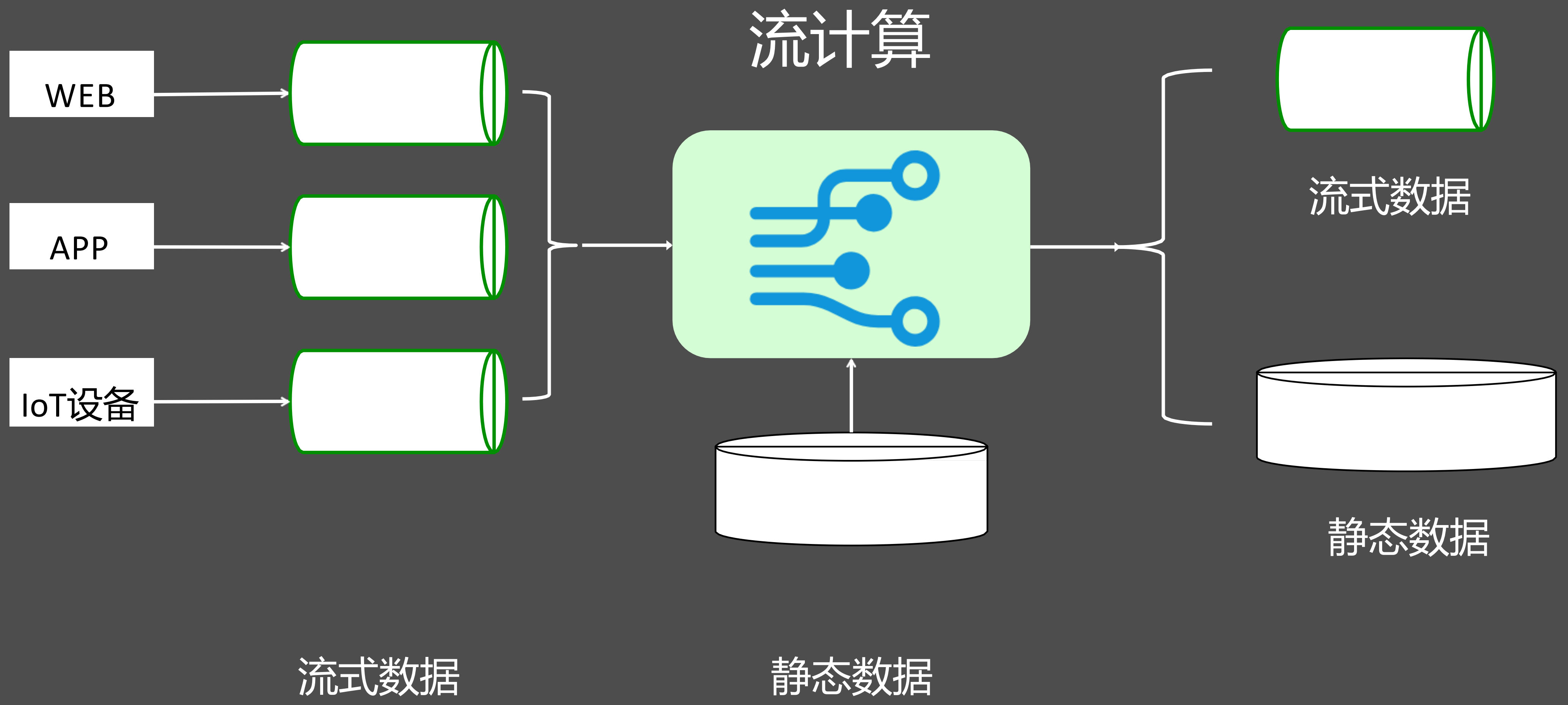
理论



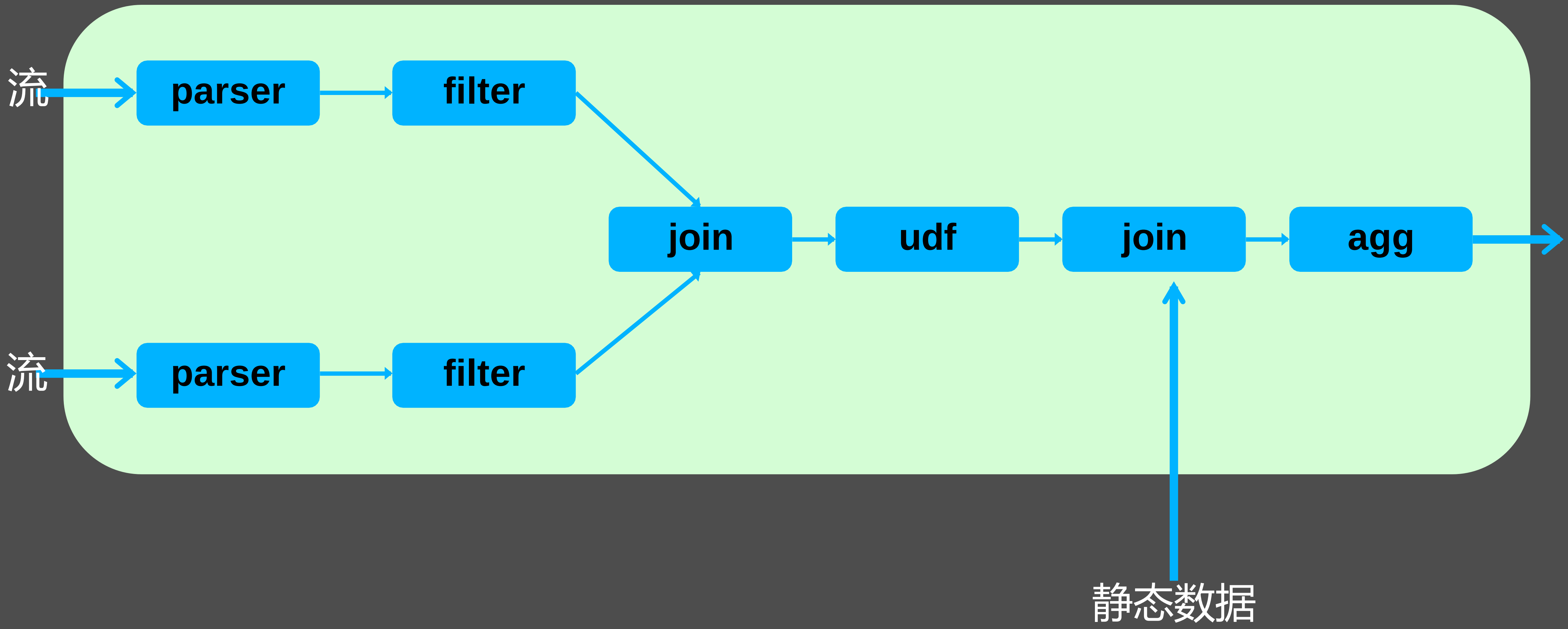
实践



典型架构



流计算



典型场景

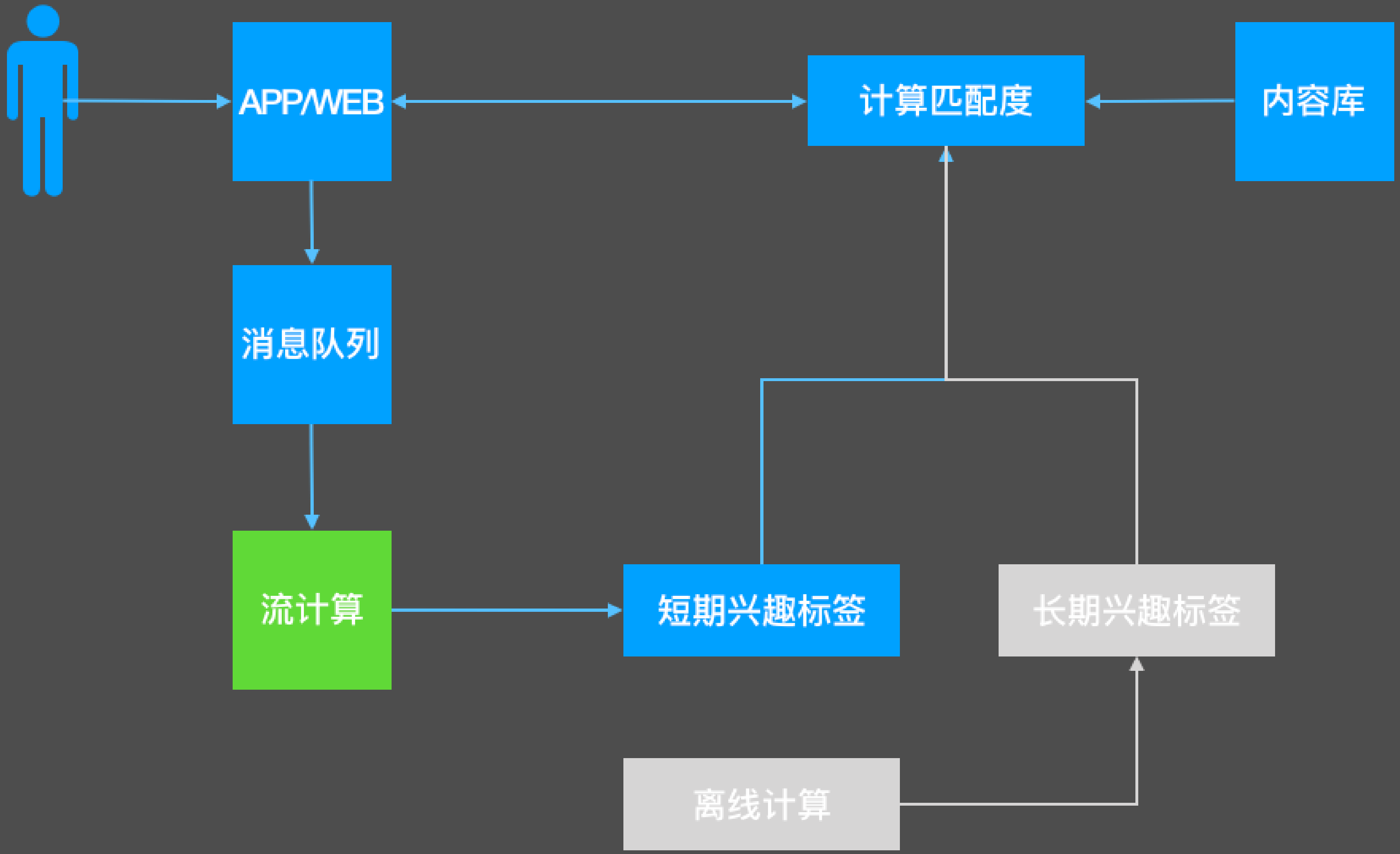
实时推荐

工业IoT

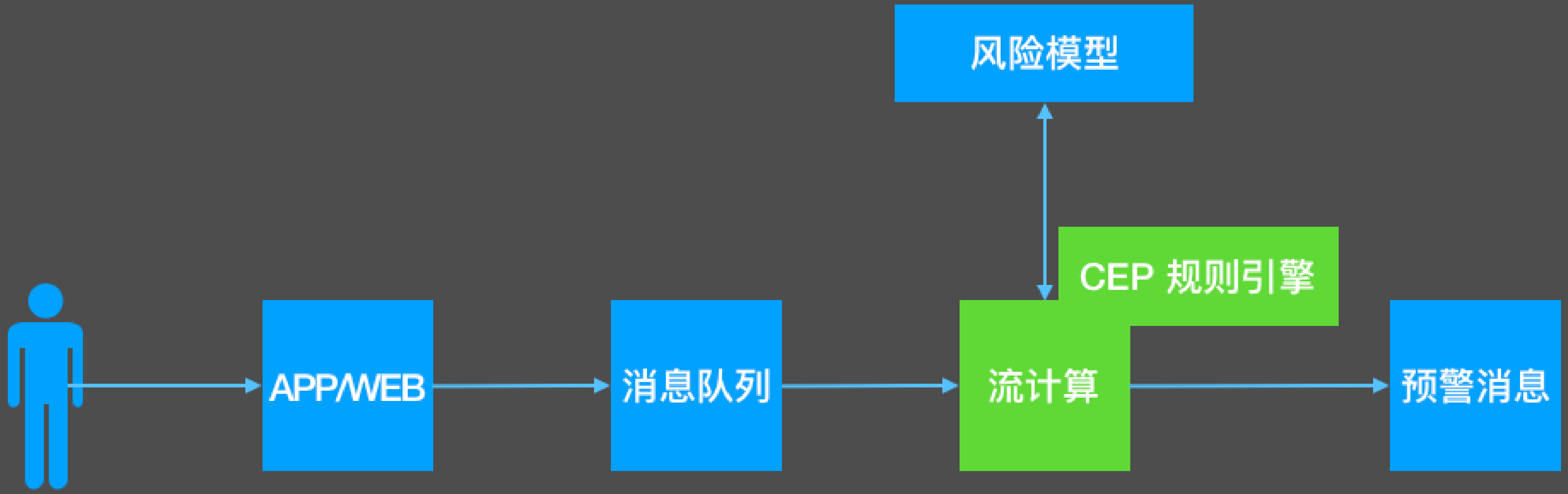
欺诈检测

实时数仓/报表

实时推荐



欺诈检测



相关术语

- Q1: 什么是streaming流式计算

  a type of data processing engine that is designed with infinite data sets in mind (一种被设计用于处理无穷数据集的数据处理引擎)

- Q2：无穷数据集

  a *type of ever-growing, essentially infinite data set* （一种持续生成，本质上是无穷尽的数据集）

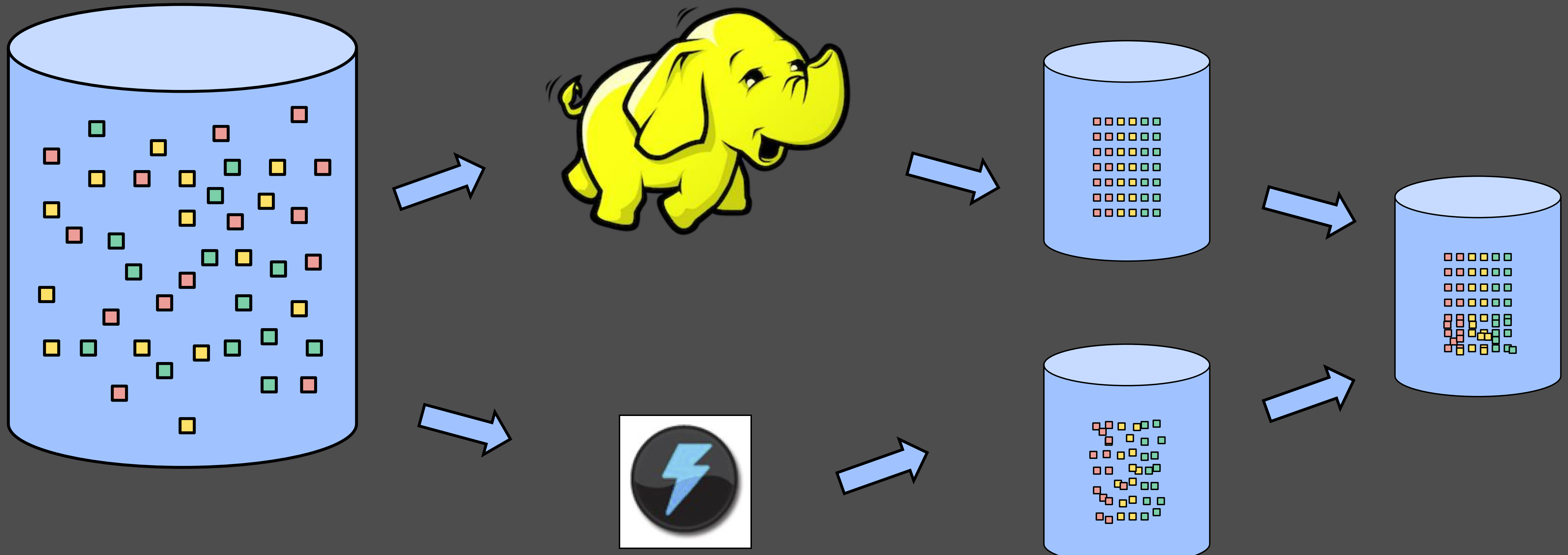
Agenda

1. State of the Streaming World
2. Streaming 101

1

State of the Streaming World

The Lambda Architecture



流计算最夸张的限制

- 长久以来，流计算系统被认为是专为提供低延迟、不精确结果的某些特定场景而设计，并配合一个更强大的批处理系统来提供最终准确的结果，如Lambda架构（Lambda Architecture）。

- Lambda架构的基本思想

  与批处理系统一起运行流计算系统，同时进行几乎一样的计算。流计算系统提供低延迟、不准确的结果（或是因为使用了近似算法，或是因为流计算系统本身没能提供足够准确的结果），而一段时间之后当批处理计算完成，再给出正确的结果。

  设计良好的流计算系统的能力是批处理系统的功能的超集。

http://radar.oreilly.com/2014/07/questioning-the-lambda-architecture.html

Questioning the Lambda Architecture

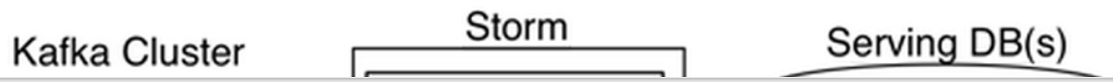
The Lambda Architecture has its merits, but alternatives are worth exploring.

by Jay Kreps | @jaykreps | +Jay Kreps | Comments: 19 | July 2, 2014

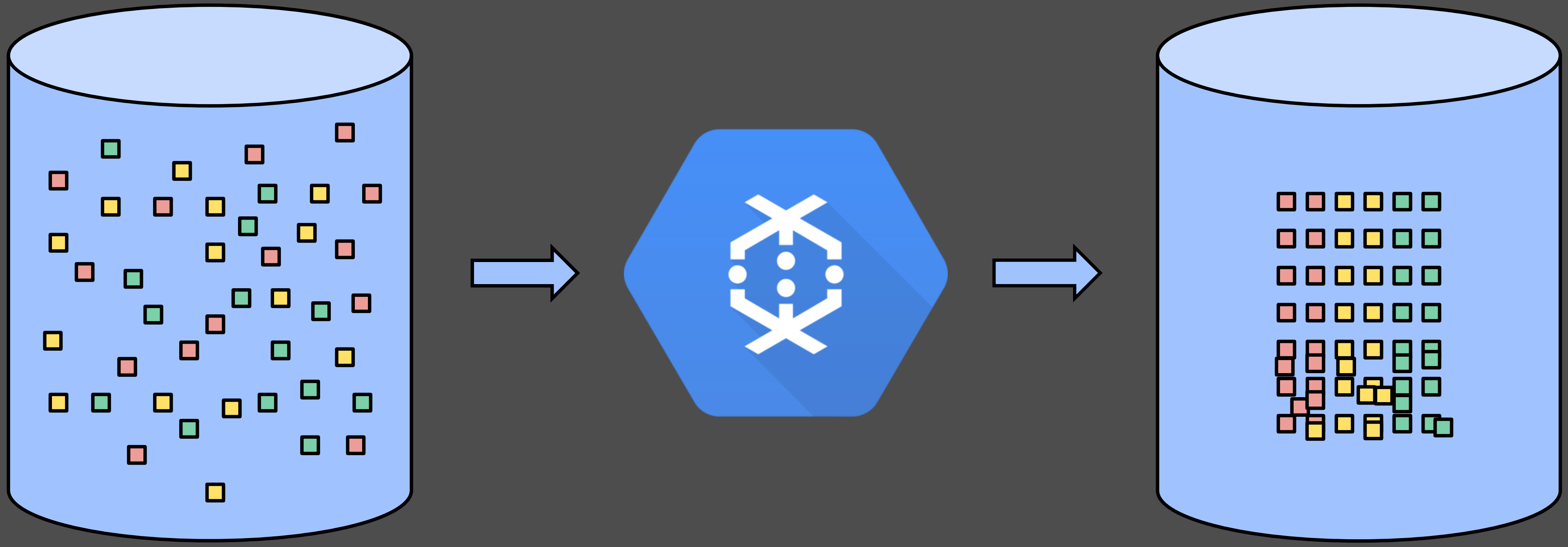
Nathan Marz wrote a popular blog post describing an idea he called the Lambda Architecture ("How to beat the CAP theorem"). The Lambda Architecture is an approach to building stream processing applications on top of MapReduce and Storm or similar systems. This has proven to be a surprisingly popular idea, with a dedicated website and an upcoming book. Since I've been involved in building out the real-time data processing infrastructure at LinkedIn using Kafka and Samza, I often get asked about the Lambda Architecture. I thought I would describe my thoughts and experiences.

What is a Lambda Architecture and how do I become one?

The Lambda Architecture looks something like this:



The Evolution of Streaming



What does it take?

Strong Consistency

Tools for Reasoning About Time

Why consistency is important

- Mostly correct is not good enough
- Required for exactly-once processing
- Required for repeatable results
- Cannot replace batch without it

Fault tolerance in streaming

- How do we ensure the results are always correct?
- Failures should not lead to data loss or incorrect results

Fault tolerance in streaming

- **at least once**: ensure all operators see all。

  Storm: Replay stream in failure case (acking of individual records)

- **Exactly once**: ensure that operators do not perform duplicate updates to their state

  Flink: Distributed Snapshots

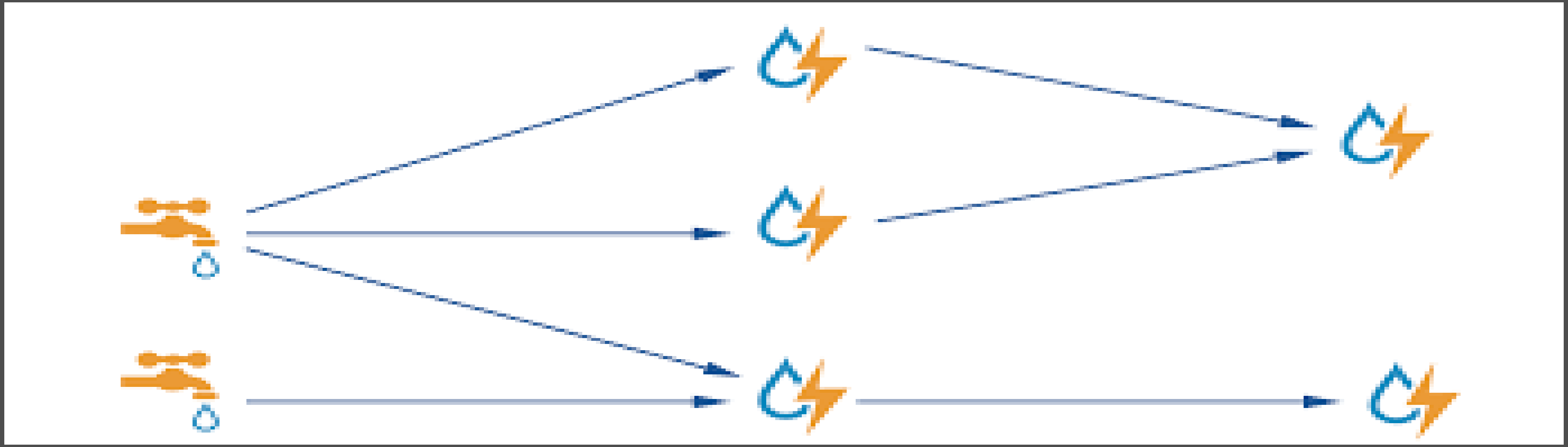
  Spark: Micro-batches on batch runtime

  Google Cloud Dataflow： Transactional updates

Storm Record acknowledgement

Record acknowledgement模式

Topology的Source会保留其产生的所有记录备份用来处理Fail情况。当源头一条记录的所有派生记录都被整个Topology处理完成，Source节点就可以删除其备份；当系统出现部分Fail情况，例如一条记录并没有收到其下游的派生记录的确认，Source就会重新发送该记录到下游的Topology以便重新进行计算。



Storm Record acknowledgement

缺点

1、性能低：每个都会产生一条对应的系统消息到acker，同时有额外的计算消耗，并且acker消息会消耗大量的网络带宽。Single ack注定与高吞吐量无缘。

2、不支持Exactly once语义。（Trident支持Exactly once语义，可以归类为Micro batches，性能下降严重。）

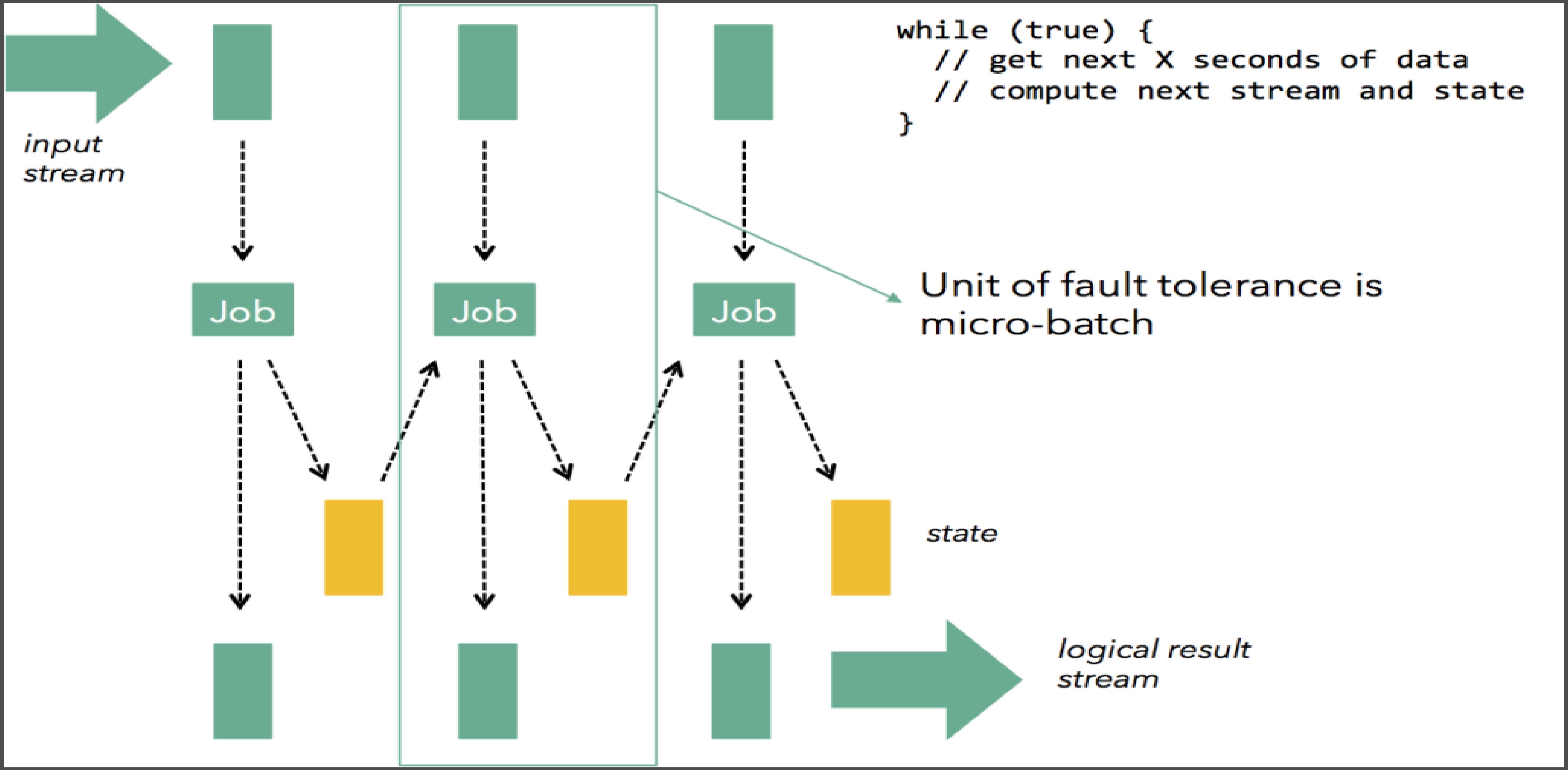
3、Back Pressure：记录确认的容错方式会导致上游节点错误地认为数据处理出现了Fail(实际上仅仅是由于back pressure导致记录处理不及时，而无法ack)

Spark：Micro-batches(stream discretization)

- 流式处理系统中的算子都是在record级别进行计算同步和容错，由此带来了在record如此低层次上进行处理的复杂和开销。

- 把连续的数据流不要切分到record级别，而是收敛切分为一批一批微批的、原子的数据进行类似Batch的计算。这样，每个batch数据可能会成功或者失败处理，我们就对当前失败的这一小批数据进行处理即可。

Spark: Micro-batches(stream discretization)



Spark：Micro-batches(stream discretization)

- 缺点：

1、延迟高

2、窗口受限：不支持count或session窗口

3、Back Pressure：如果某个下游的算子处理较慢，此时如果负责数据流切分的Operator速度快于下游的阻塞节点，就会导致数据切分比原有的配置时间更长。导致越来越多的批次在内存排队等待被处理，最终内存OOM，Runtime不够稳定。

4、微批处理模型的最大限制可能是它连接了两个不应连接的概念：应用程序定义的窗口大小和系统的批处理间隔。

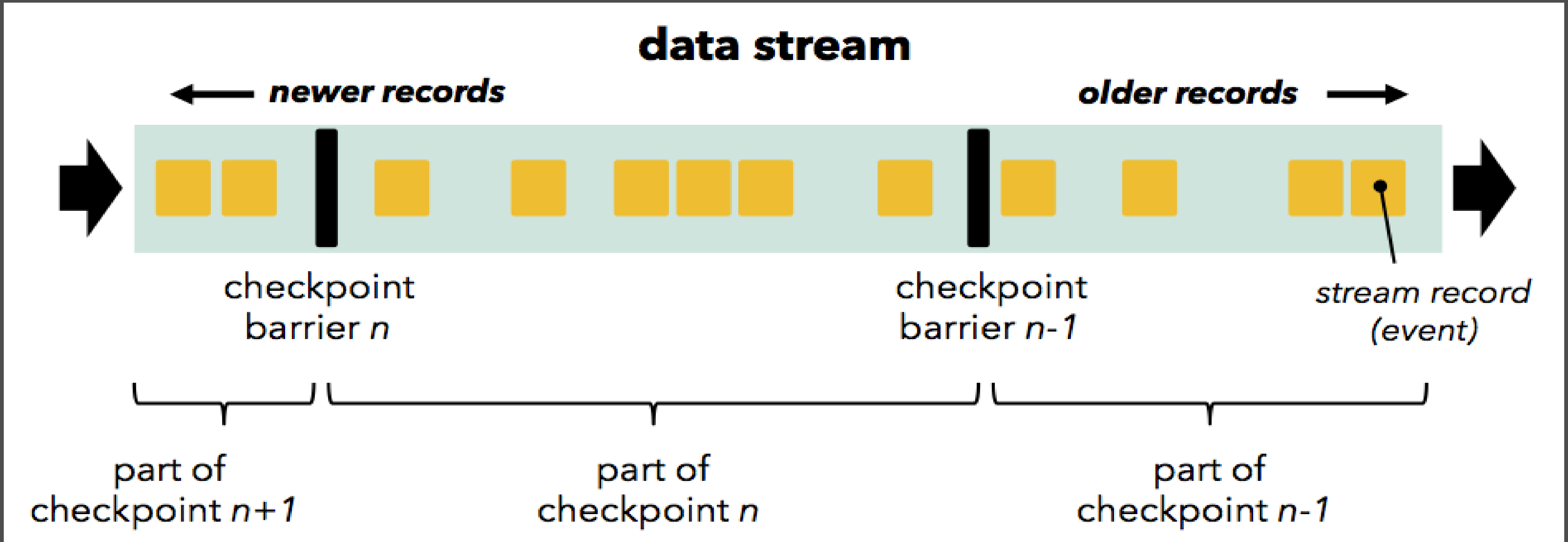
Flink： Distributed Snapshots

- Flink使用的是Chandy Lamport算法的一个变种，定期对正在运行的流拓扑的状态做快照，并将这些快照存储到持久存储。

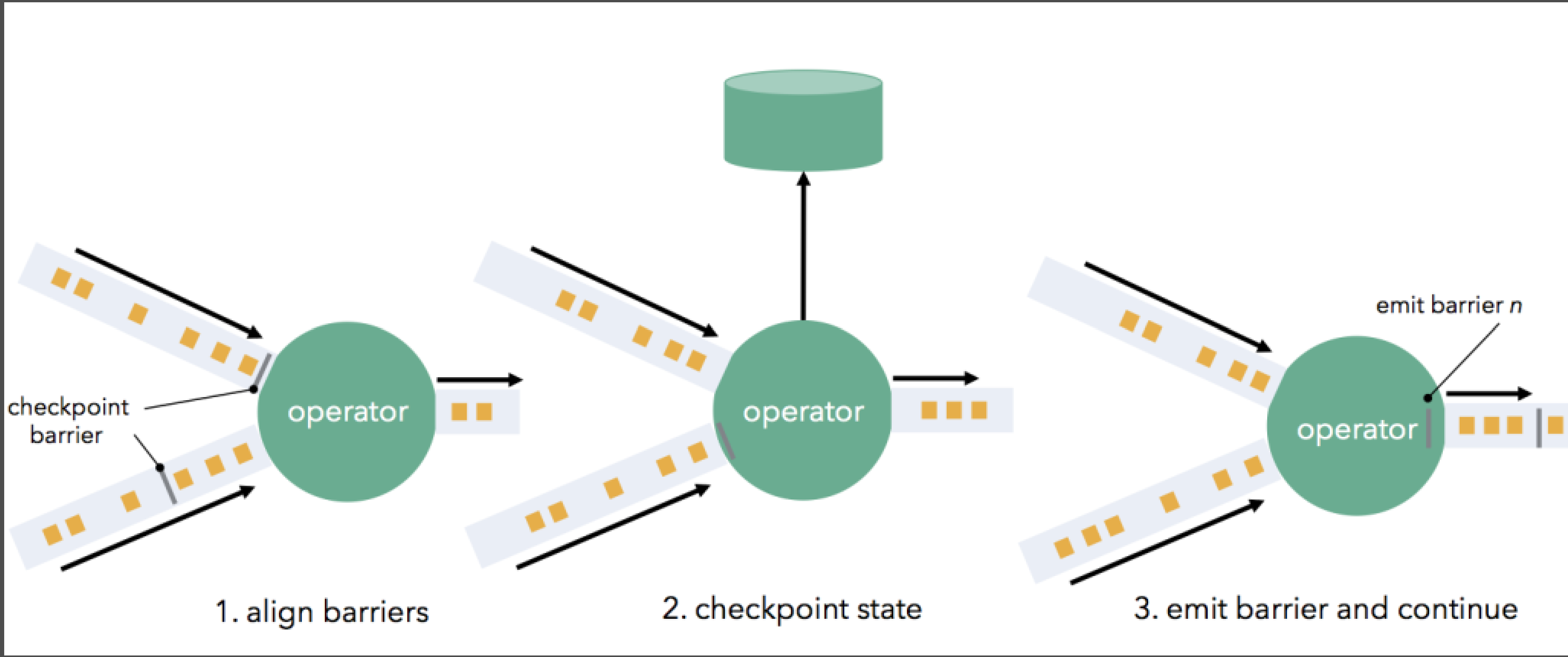
- 遵循真正的持续计算模型（低延迟，流量控制和真正的流编程模型）和高吞吐量的优点，并且也是在Chandy-Lamport算法中被严格证明的Exactly Once保证。

- 除了持久化有状态计算的状态（其他容错机制也需要这样做）之外，这种容错机制几乎没有开销。对于小状态（例如，计数或其他统计摘要），这种持久化开销通常可忽略不计。

Flink: Distributed Snapshots



Flink: Distributed Snapshots



2

Streaming 101

Event Time vs Processing Time

Batch vs Streaming

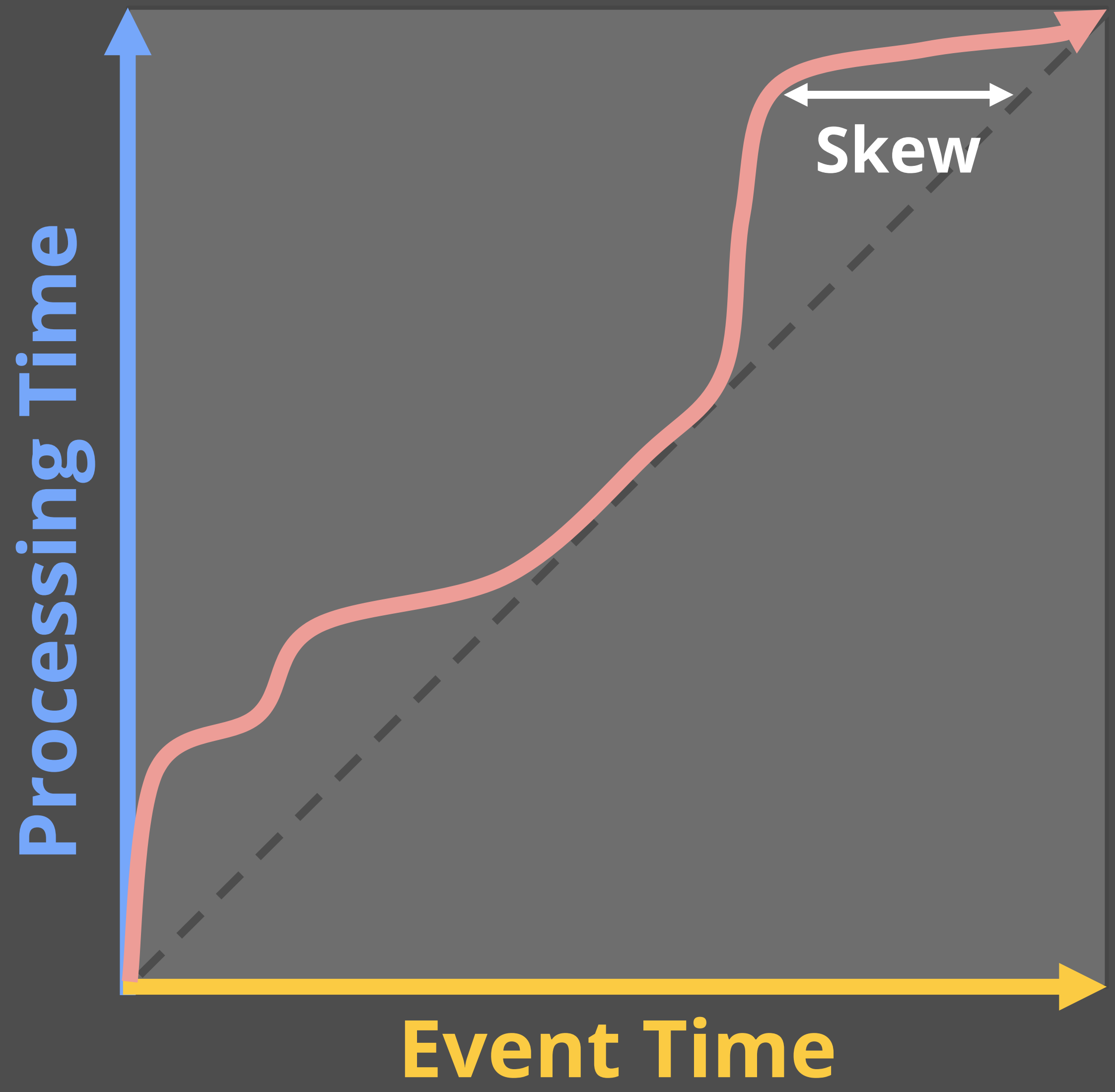
Approaches

Event Time vs Processing Time

Event Time - When Events Happened

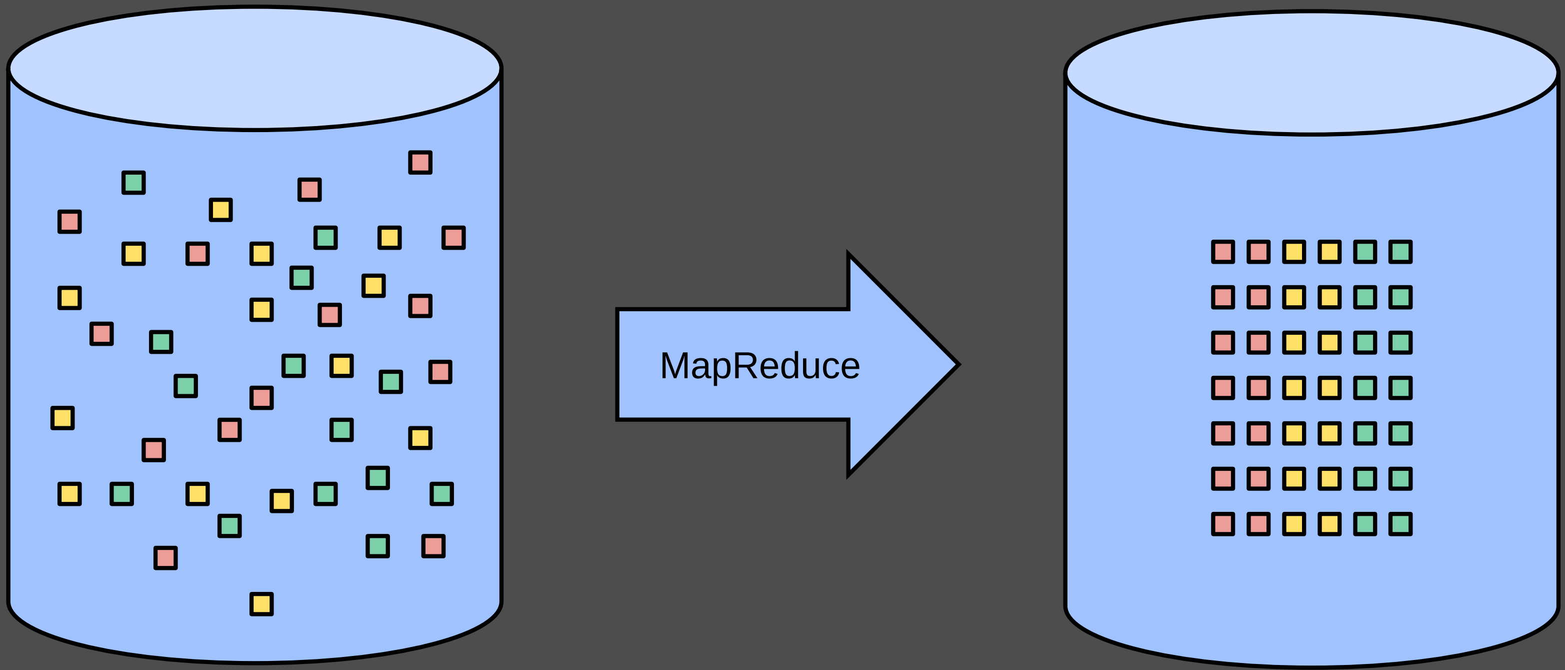
Processing Time - When Events Are Processed

Event Time Skew

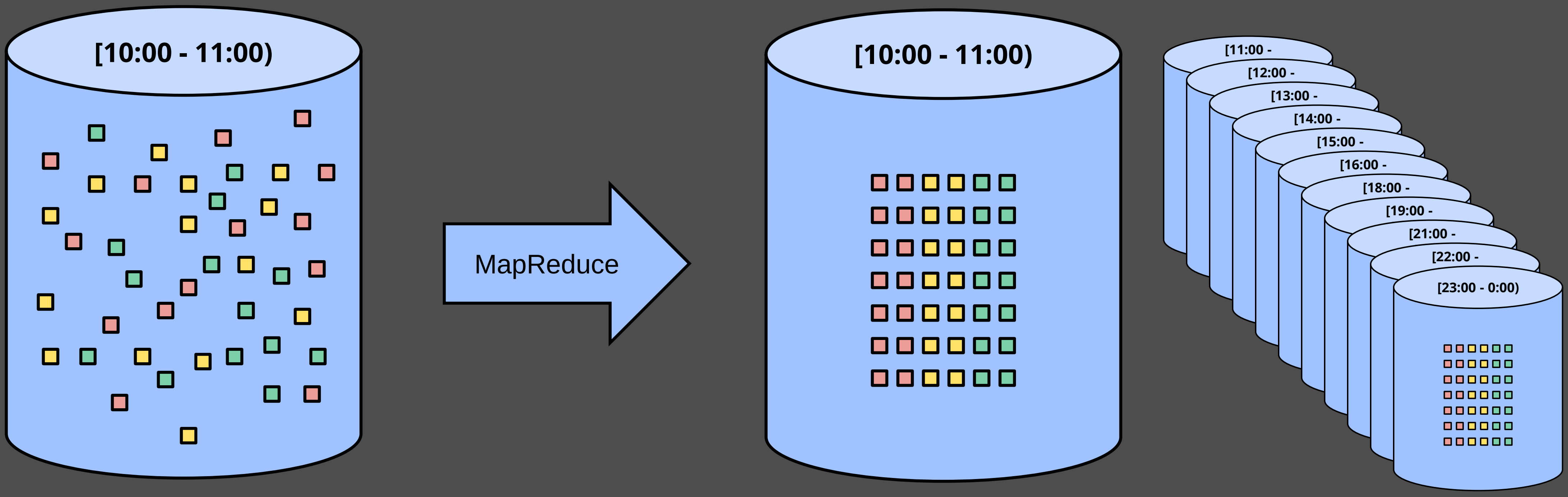


Batch vs Streaming

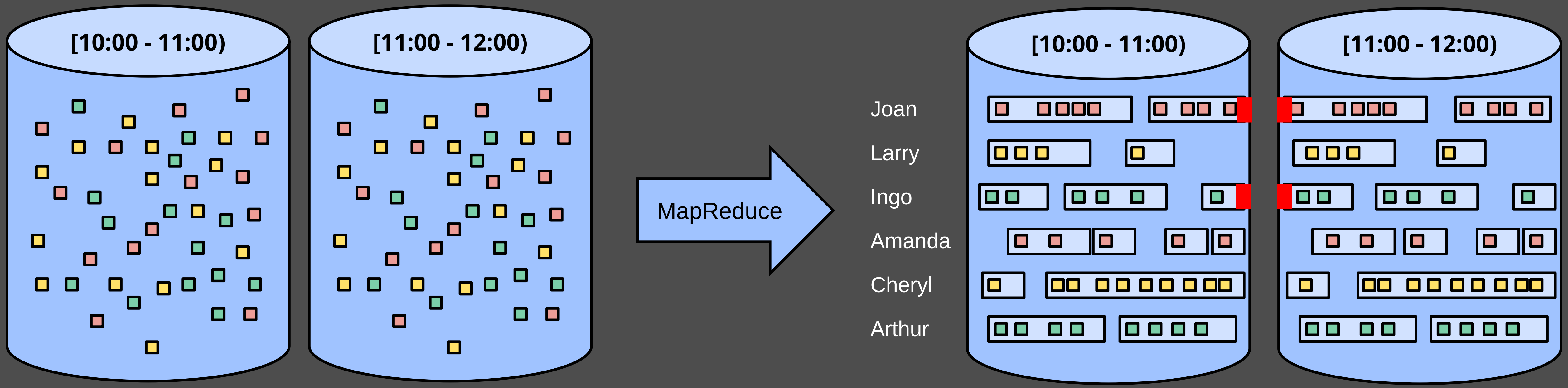
Batch



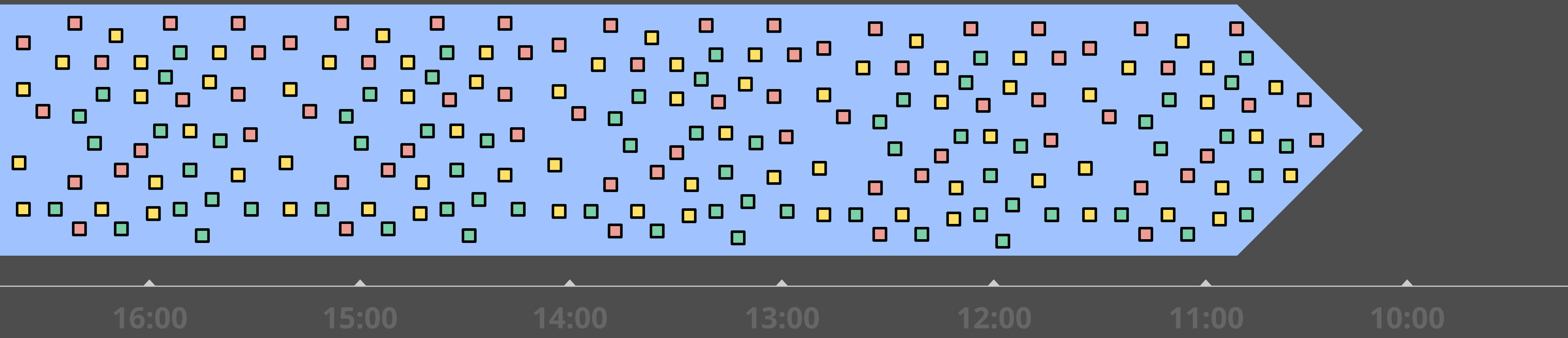
Batch: Fixed Windows



Batch: User Sessions



Streaming



Confounding characteristics of data streams

Unordered

Unbounded

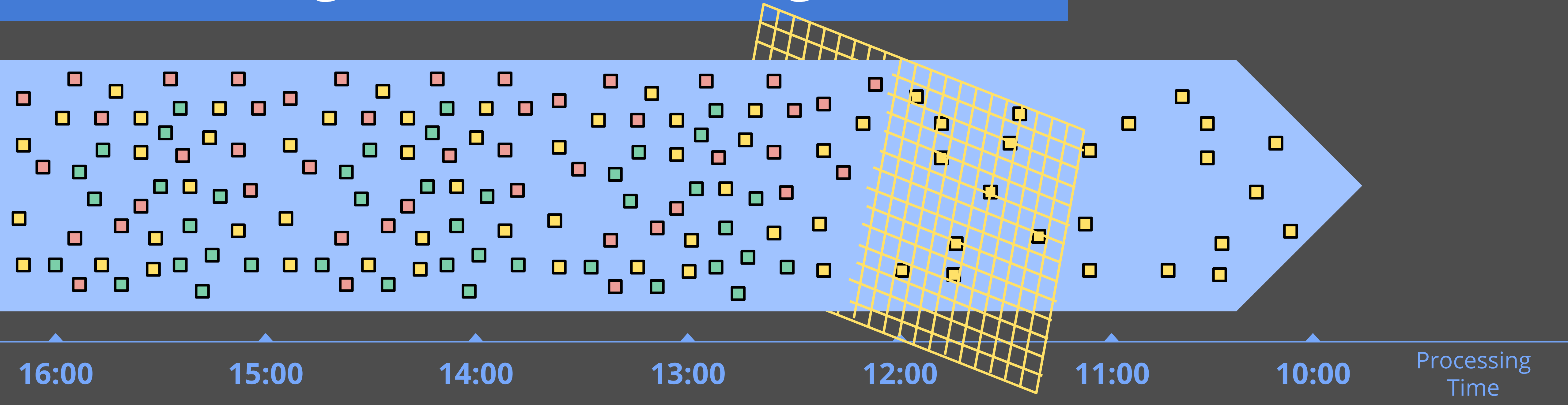
Of Varying Event Time Skew

Approaches

Approaches to reasoning about time

1. Time-Agnostic Processing
2. Approximation
3. Processing Time Windowing
4. Event Time Windowing

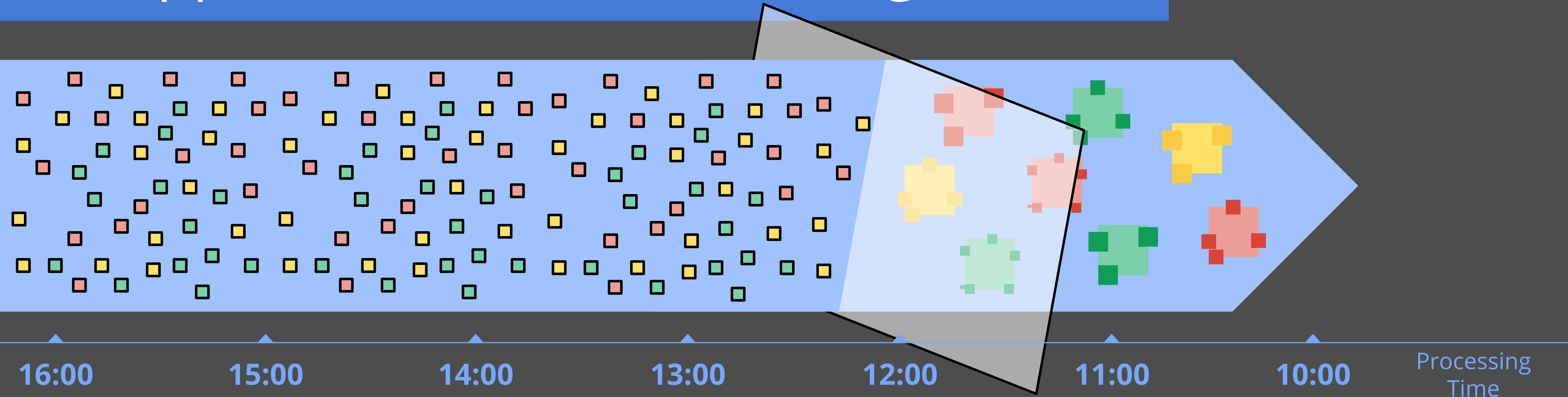
1. Time-Agnostic Processing - Filters



Example Input: Web server traffic logs
Example Output: All traffic from specific domains

Pros: Straightforward
Efficient
Cons: Limited utility

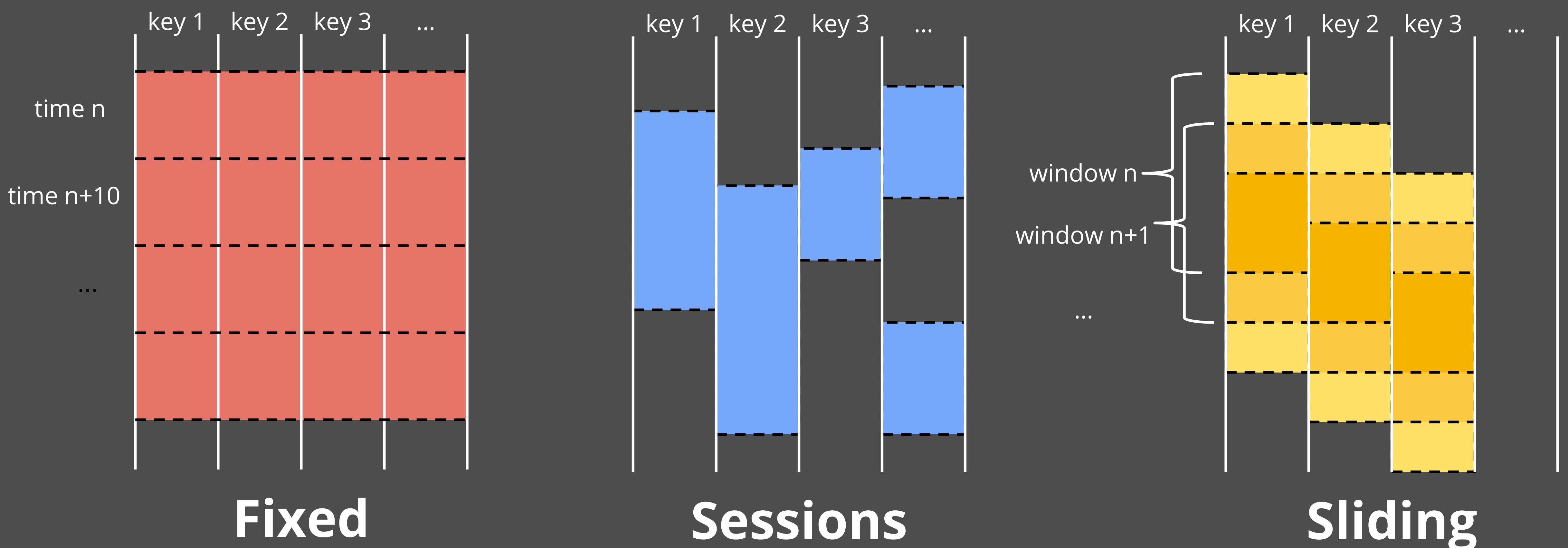
2. Approximation via Online Algorithms



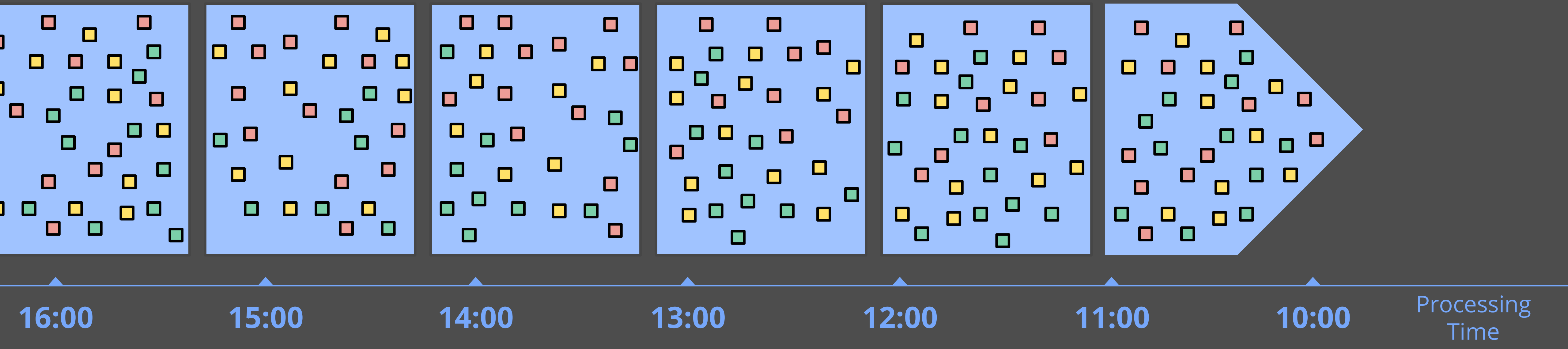
Example Input: Twitter hashtags
Example Output: Approximate top N hashtags per prefix

Pros: Efficient
Cons: Inexact
Complicated Algorithms

Windowing

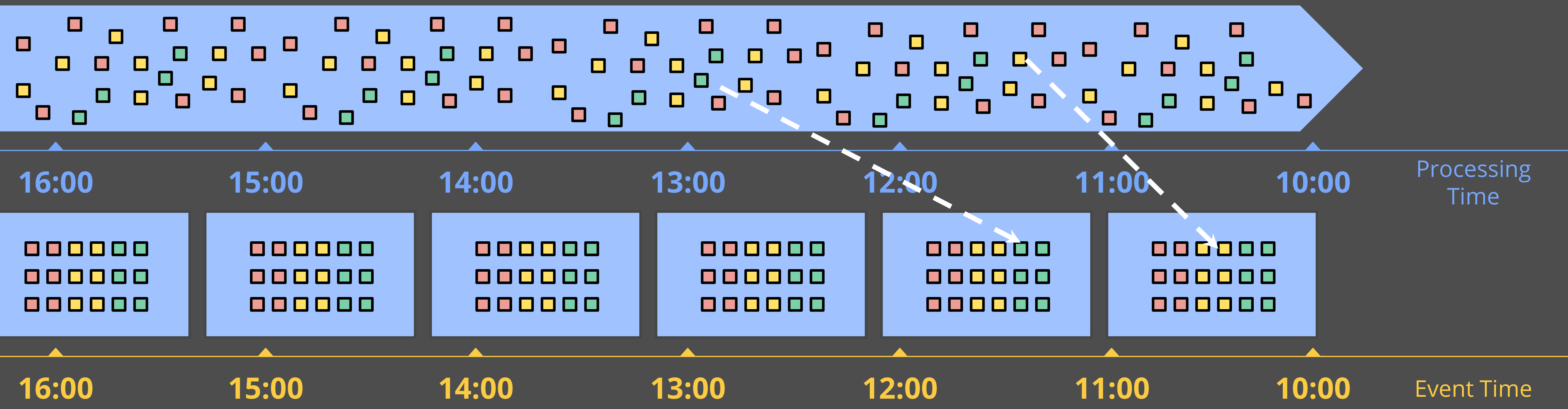

key 1
key 2
key 3
...
time n
time n+10
...
Fixed
key 1
key 2
key 3
...
Sessions
key 1
key 2
key 3
...
window n
window n+1
...
Sliding

3. Windowing by Processing Time



Example Input:	Web server request traffic
Example Output:	Per-minute rate of received requests
Pros:	Straightforward Results reflect contents of stream
Cons:	Results don’t reflect events as they happened If approximating event time, usefulness varies

4. Windowing by Event Time - Fixed Windows



Example Input: Twitter hashtags

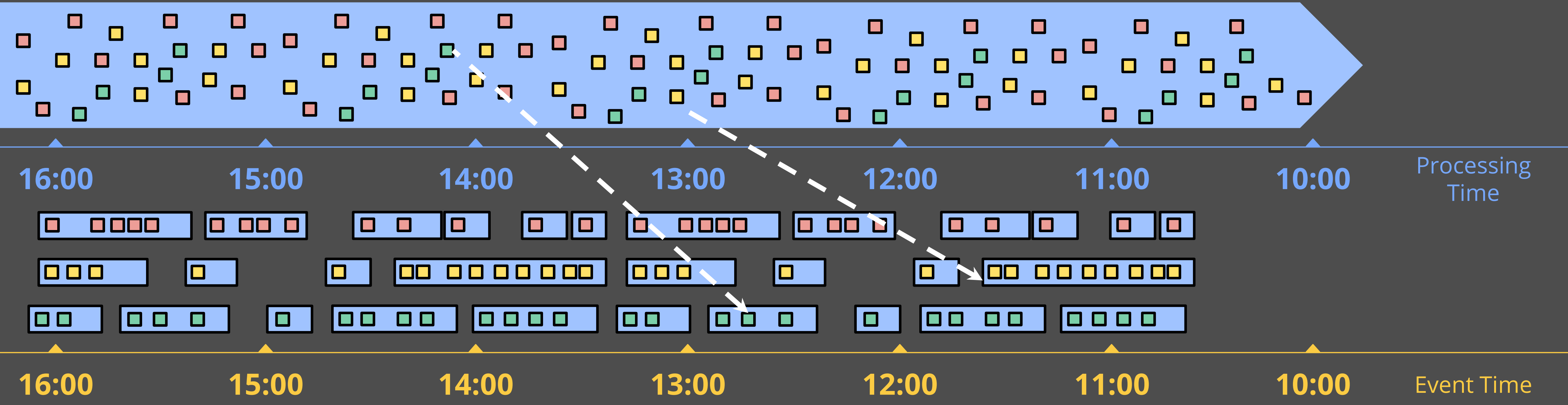
Example Output: Top N hashtags by prefix per hour.

Pros: Reflects events as they occurred

Cons: More complicated buffering

Completeness issues

4. Windowing by Event Time - Sessions



Example Input: User activity stream
Example Output: Per-session group of activities
Pros: Reflects events as they occurred
Cons: More complicated buffering
Completeness issues

谢　谢