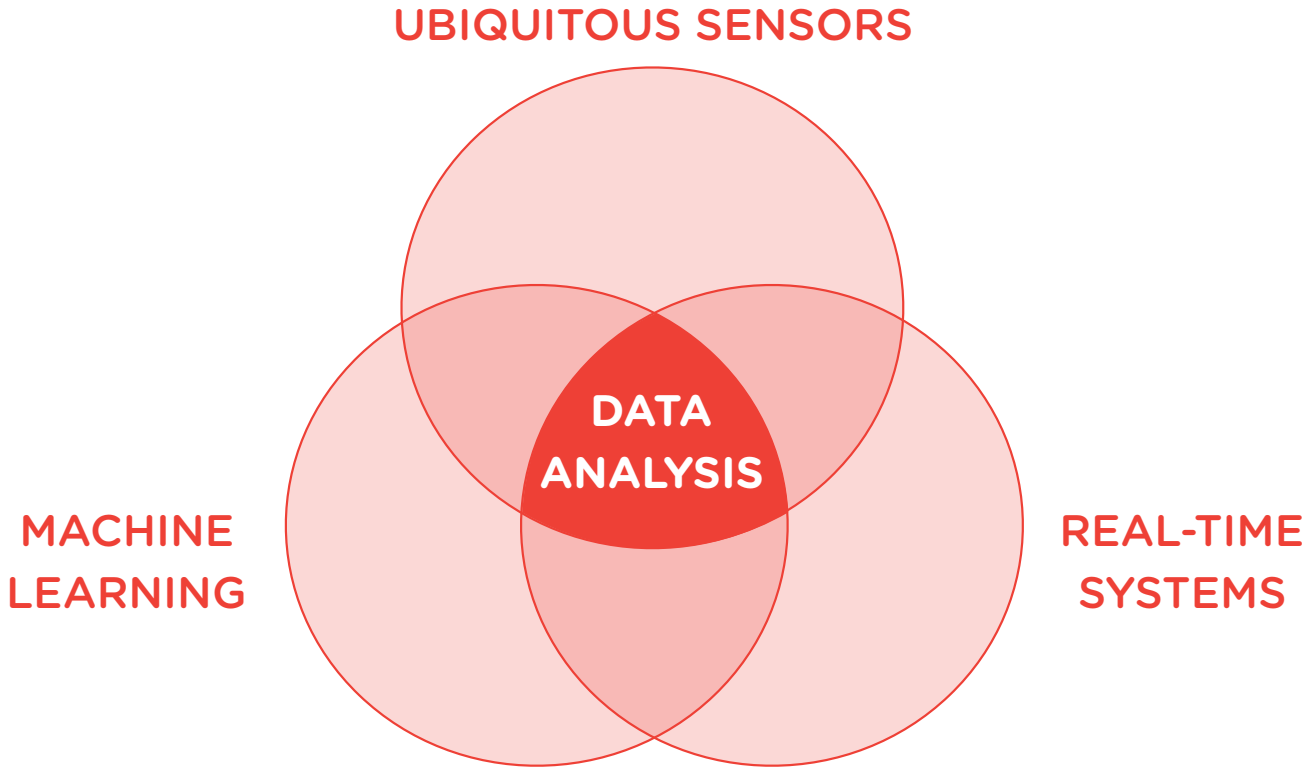
Spark Streaming and IoT

Michael J. Freedman
iobeam

SPARK SUMMIT EAST
DATA SCIENCE AND ENGINEERING AT SCALE
FEBRUARY 16-18, 2016 NEW YORK CITY

Technology confluence in IoT

INTERSECTION OF 3 MAJOR TRENDS



Data analysis is the killer app

CASE STUDY: PREDICTIVE MAINTENANCE

Predicting motor failure through analysis of vibration data

CASE STUDY: SMART CITIES

Traffic and air quality monitoring via GPS and environmental sensor

CASE STUDY: HEALTH & FITNESS

Exercise identification based on 3D motion data analysis

CASE STUDY: SMART GRID

Demand-response optimizations on supply-side capacity, spot prices

Challenges in applying Spark to IoT

REQUIREMENTS

1. One IoT app performs tasks at different time intervals
2. Devices send data at varying delays and rates
3. Within org, multiple IoT apps run concurrently

CHALLENGES

1. Supporting full spectrum of batch to real-time analysis
2. Handling delayed data transparently
3. Processing many low-volume, independent streams
4. Multi-tenancy with low-volume apps and high utilization

Potential economic impact of IoT is >$11 trillion per year, even while 99% of IoT data goes unused today.

— 2015 McKinsey study

Required: Programming + data infra abstractions

INFRA

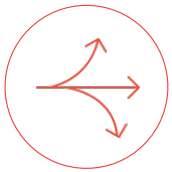
1

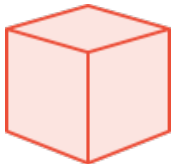
Supporting full spectrum of batch to real-time analysis

IoT analysis spans many intervals

STREAM PROCESSING
(REAL-TIME)

BATCH PROCESSING
(HOURS, NIGHTLY)





Fire / Hazard Detection
Immediately

Bus Location Updates
15 sec

Traffic Conditions
1 min

Environmental Conditions
15 min

Traffic Optimizations
Daily

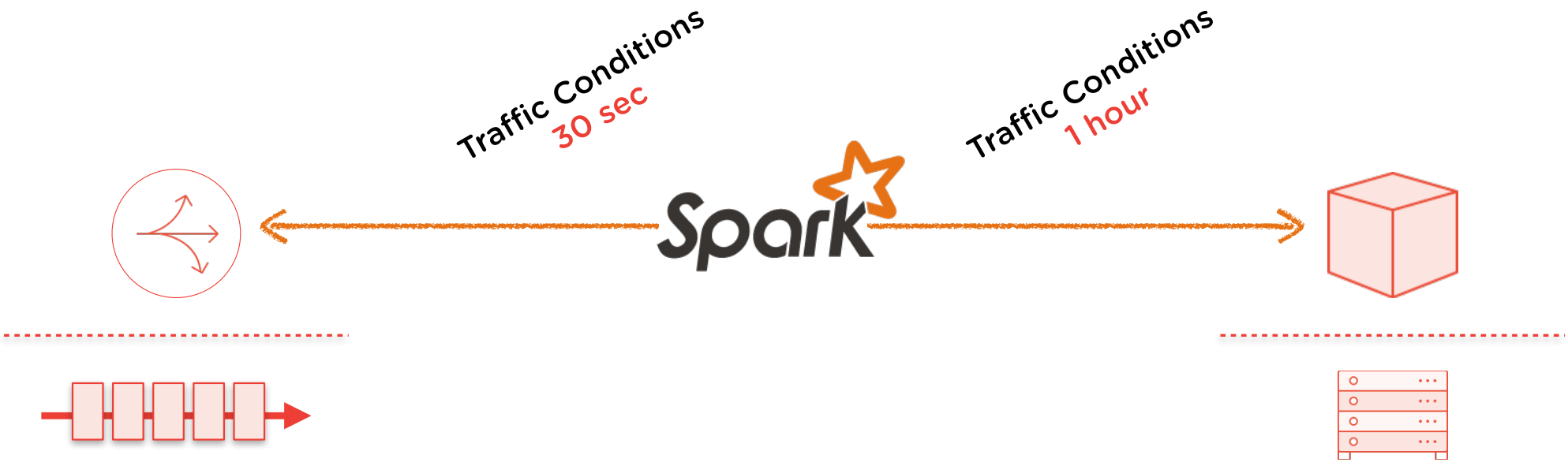
Spark simplifies programming across intervals

```
val readings = iobeamInterface.getInputStreamRecords()

// Trigger temperatures that fall outside acceptable conditions
val bad_temps = readings.filter(t => t > highTempThreshold || t < lowTempThreshold)
val triggers = new TriggerEventStream(bad_temps.map(t => new TriggerEvent("bad_temperature", t)))

// Compute mean temperatures over 30 min windows
val windows = readings.groupByKeyAndWindow(Seconds( 1800 ), Seconds(60))
val mean_temps = new TimeSeriesStream("mean_temperature", windows.map(t => t.sum / t.length))

new OutputStreams(mean_temps, triggers)
```

But programming != data abstractions

Programming != data abstractions

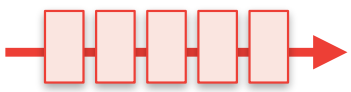
1 Frequencies change as products evolve

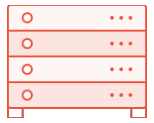


Programming != data abstractions

1. Frequencies change as products evolve
2. Joining real-time with historical data

5 min mean vs. trailing
- hourly mean
- hourly mean from yesterday
- hourly mean from last week





Programming != data abstractions

1. Frequencies change as products evolve
2. Joining real-time with historical data
3. Supporting backfill for delayed data

Programming != data abstractions

1. Frequencies change as products evolve
2. Joining real-time with historical data
3. Supporting backfill for delayed data

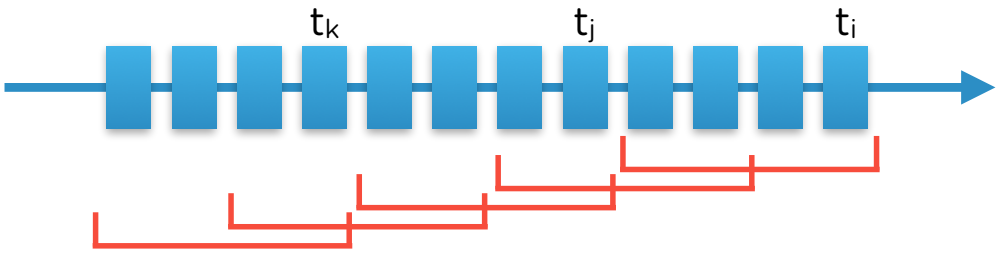
Data Series Abstraction

2 Handling delayed data transparently

Windows in streaming DBs

Tumbling windows

Windows in streaming DBs



Sliding windows

- Defined over # of tuples
- Defined over time period

  ...using arrival_time of tuples

But IoT data is often delayed

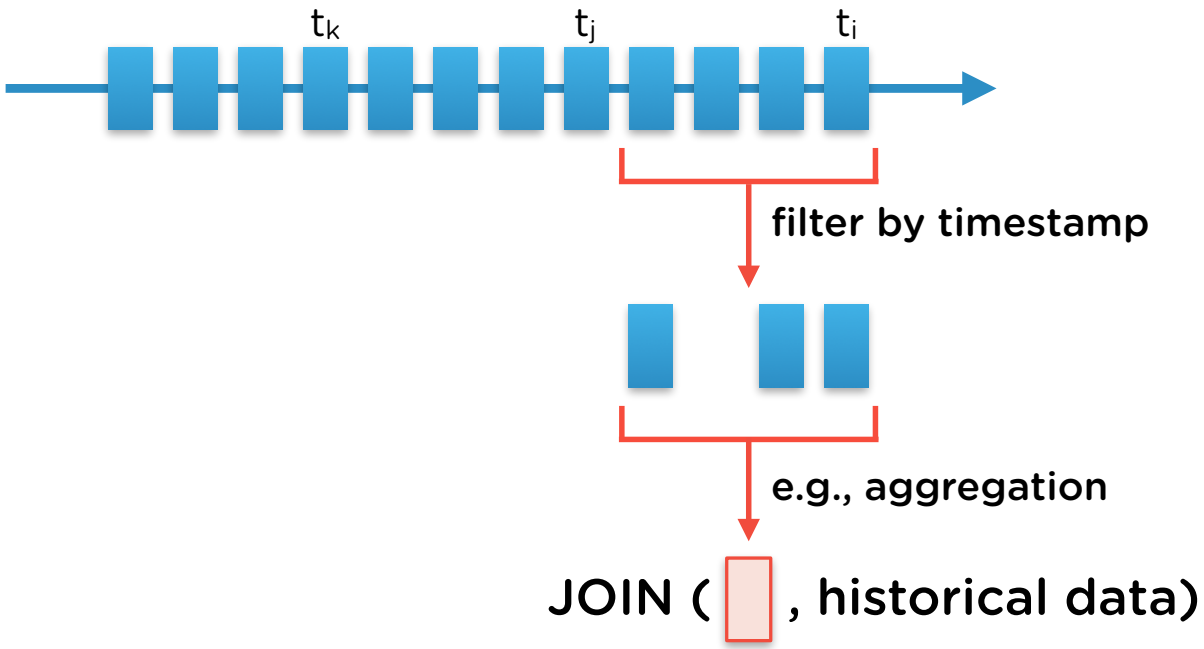
Seconds due to network congestion

Minutes due to duty cycling for energy savings

Minutes to hours due to intermittent connectivity

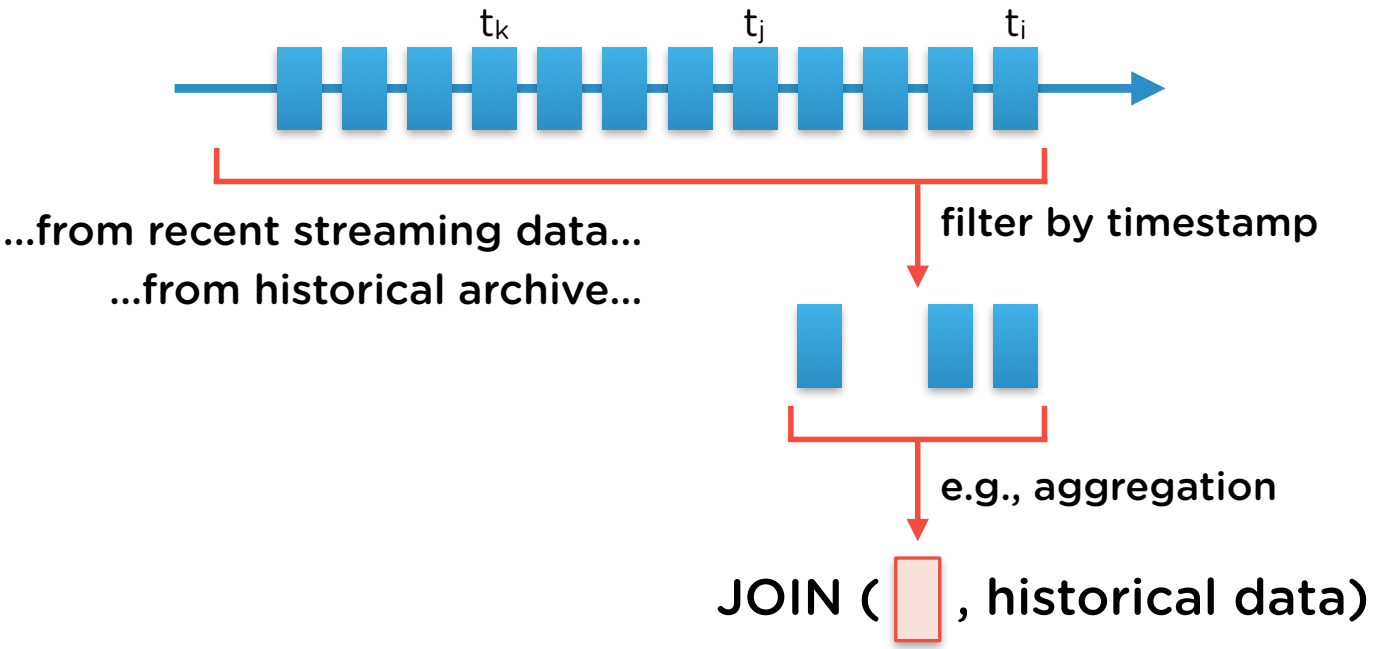
Windowing data by arrival time has no semantic meaning

Wanted: Data generation time, not arrival time



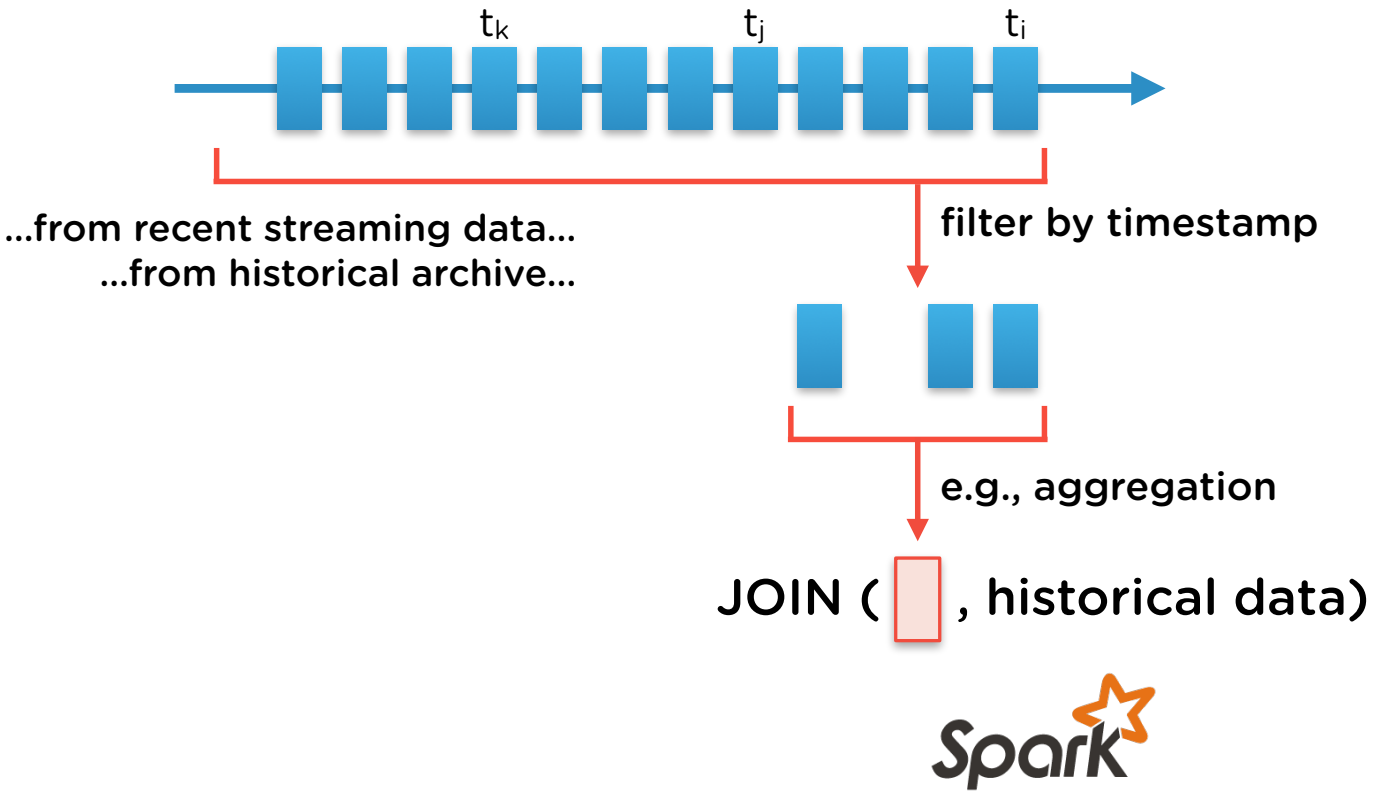
Data semantics defined over timestamp

Wanted: Backfill does not change semantics



Data semantics defined over timestamp

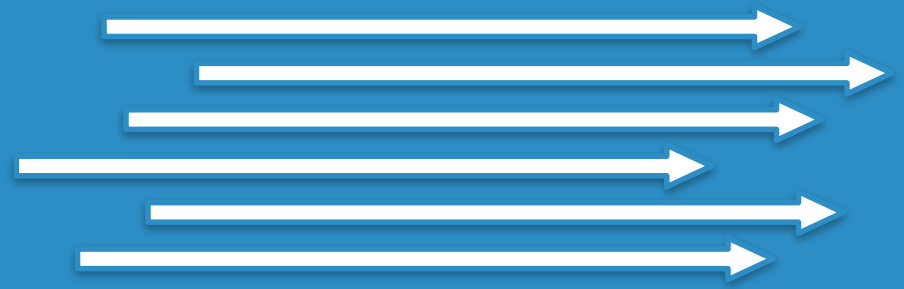
Wanted: Better data infra abstractions



Data Series Abstraction

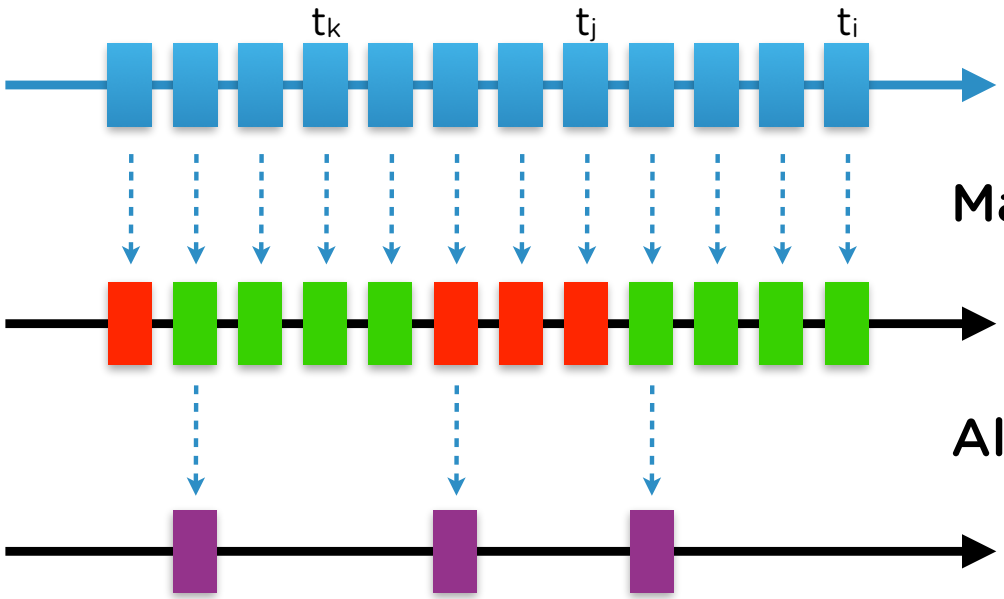
3

Processing many low-volume, independent streams



IoT Device Streams

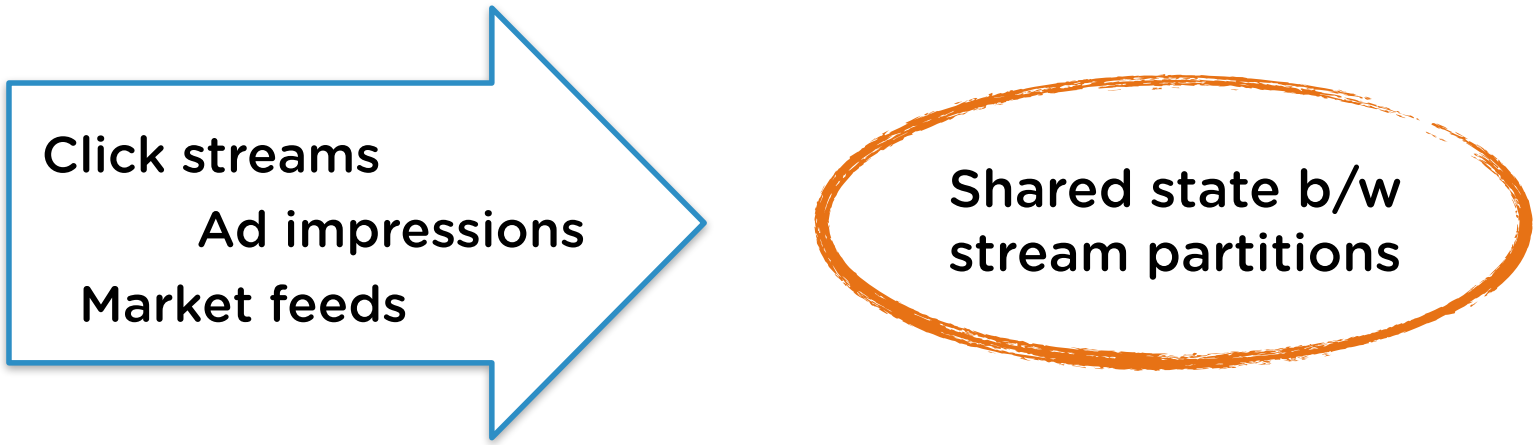
Wanted: Maintain state across batches



Map to good/bad conditions

Alert on condition transition

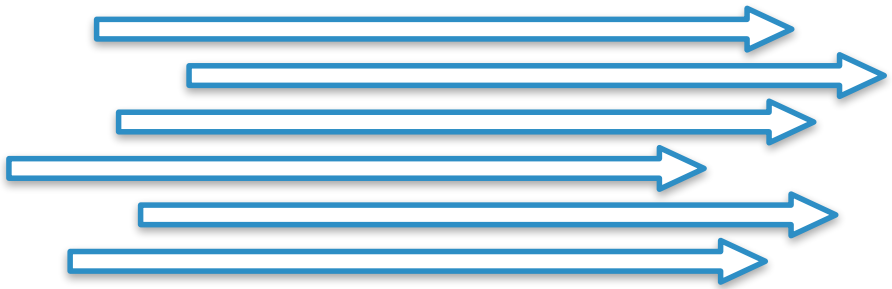
Spark: Share state through RDDs



Maintain shared state via *updateKeyByState()*

- Transforms RDD, makes state available across cluster
- Many great uses, e.g., learning parameters in iterative ML
- But increases data lineage ⇨ increases checkpointing cost

IoT: Many independent streams



IoT Device Streams

Often only need to maintain state *within* individual streams

- Each worker handles 1+ streams, not multiple workers per stream
- Use language data structures (e.g., Java Map) to maintain state within worker
- No RDD transform ⇨ no lineage increase ⇨ no increased checkpointing cost

4 Multi-tenancy with low-volume apps and high utilization

Multi-tenancy for batch processing

Goal: Minimize time-to-completion

Spark: 1 worker = 1 server core

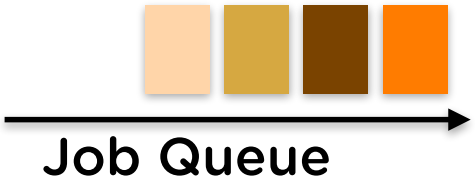
Job Queue

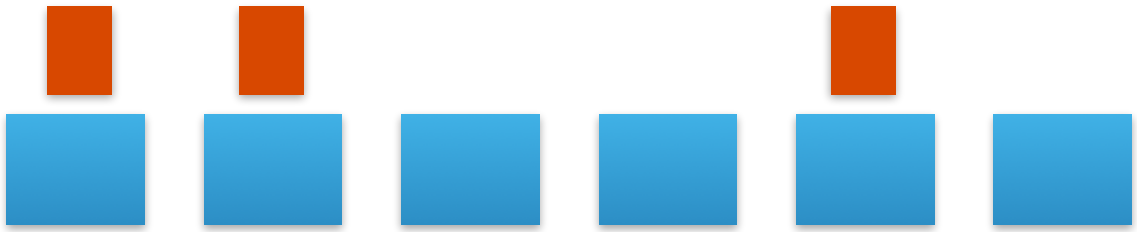
Server Cores

Multi-tenancy for batch processing

Goal: Minimize time-to-completion

Spark: 1 worker = 1 server core

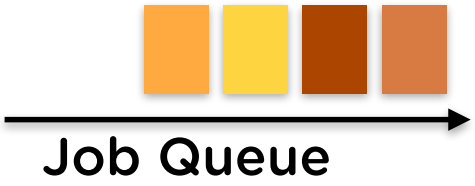


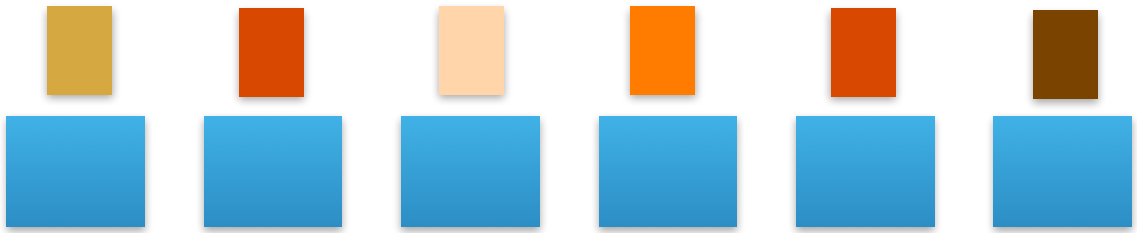


Multi-tenancy for stream processing

Problem: Low utilization with low-rate apps

Spark: 1 worker = 1 server core



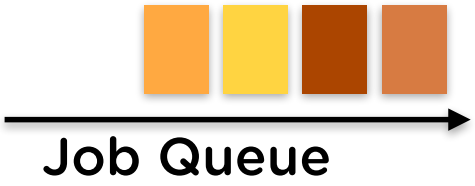


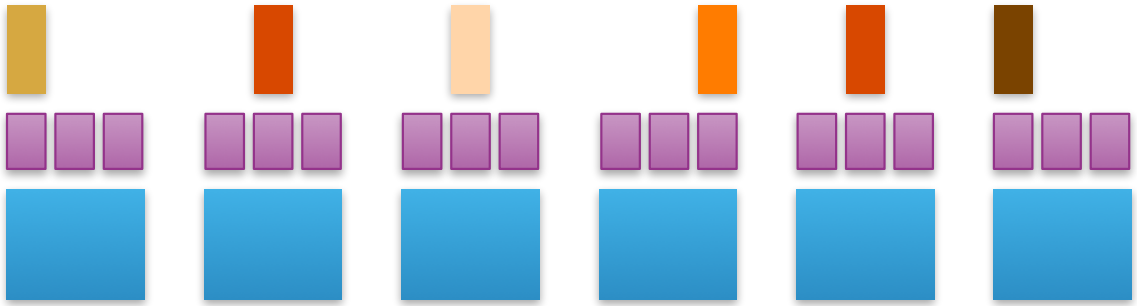
Multi-tenancy for stream processing

Goal: Improve utilization with low-rate apps

1 worker = 1 virtual core

N workers = 1 server core



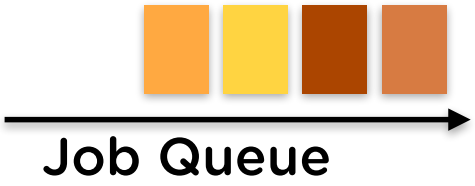


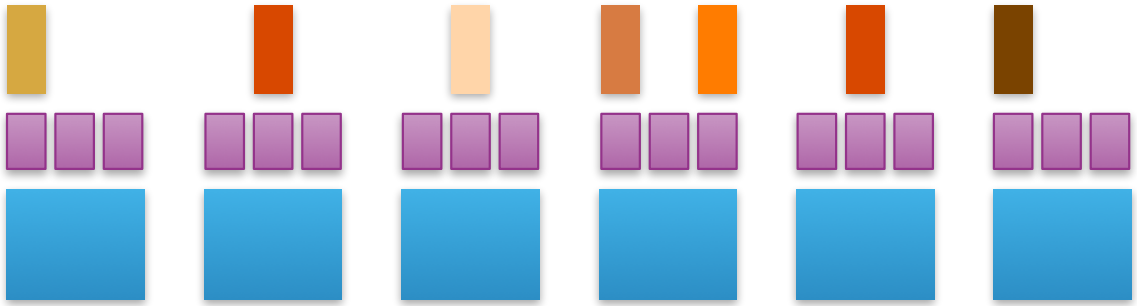
Multi-tenancy for stream processing

Goal: Improve utilization with low-rate apps

1 worker = 1 virtual core

N workers = 1 server core



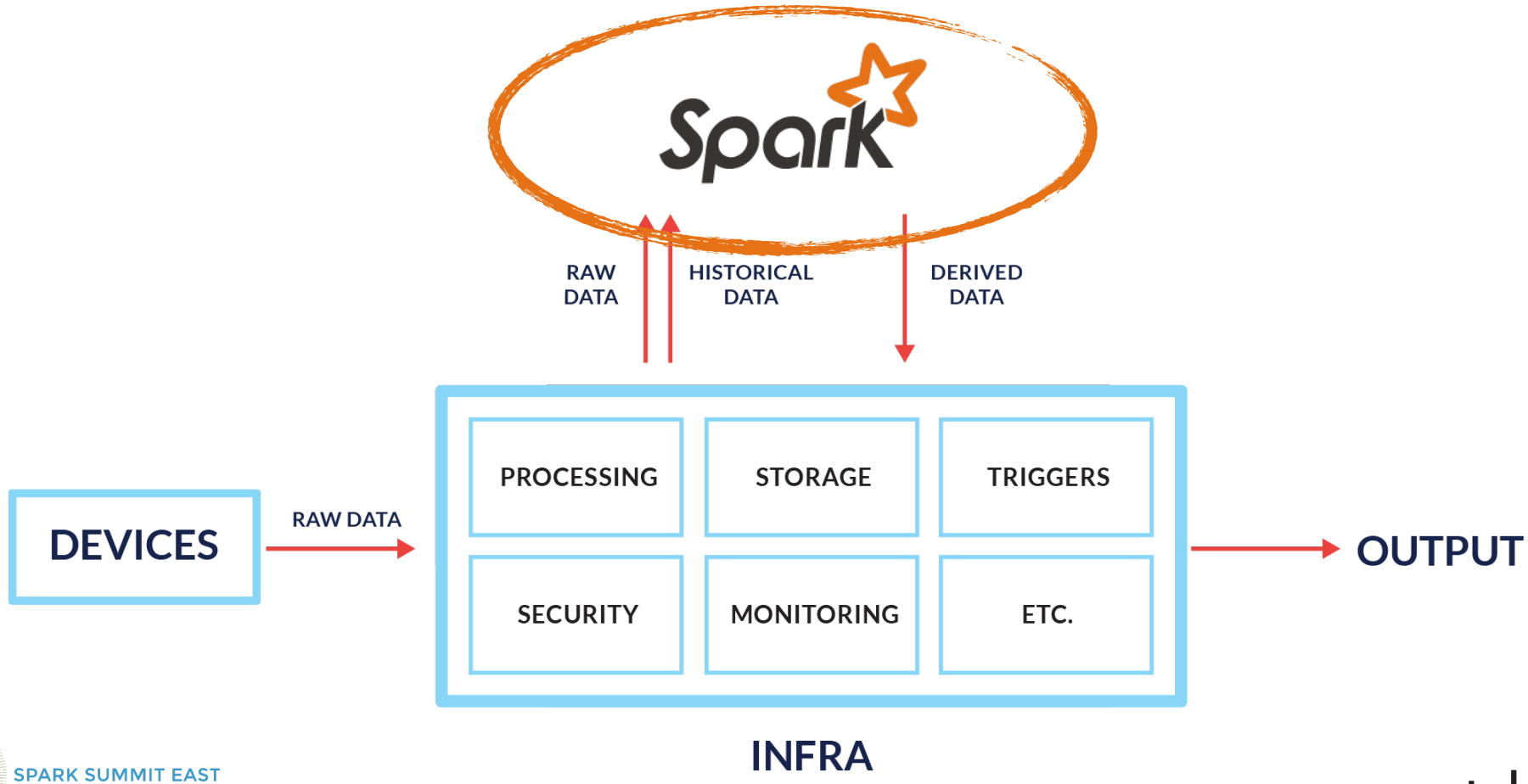


Spark + Unified Data Infrastructure

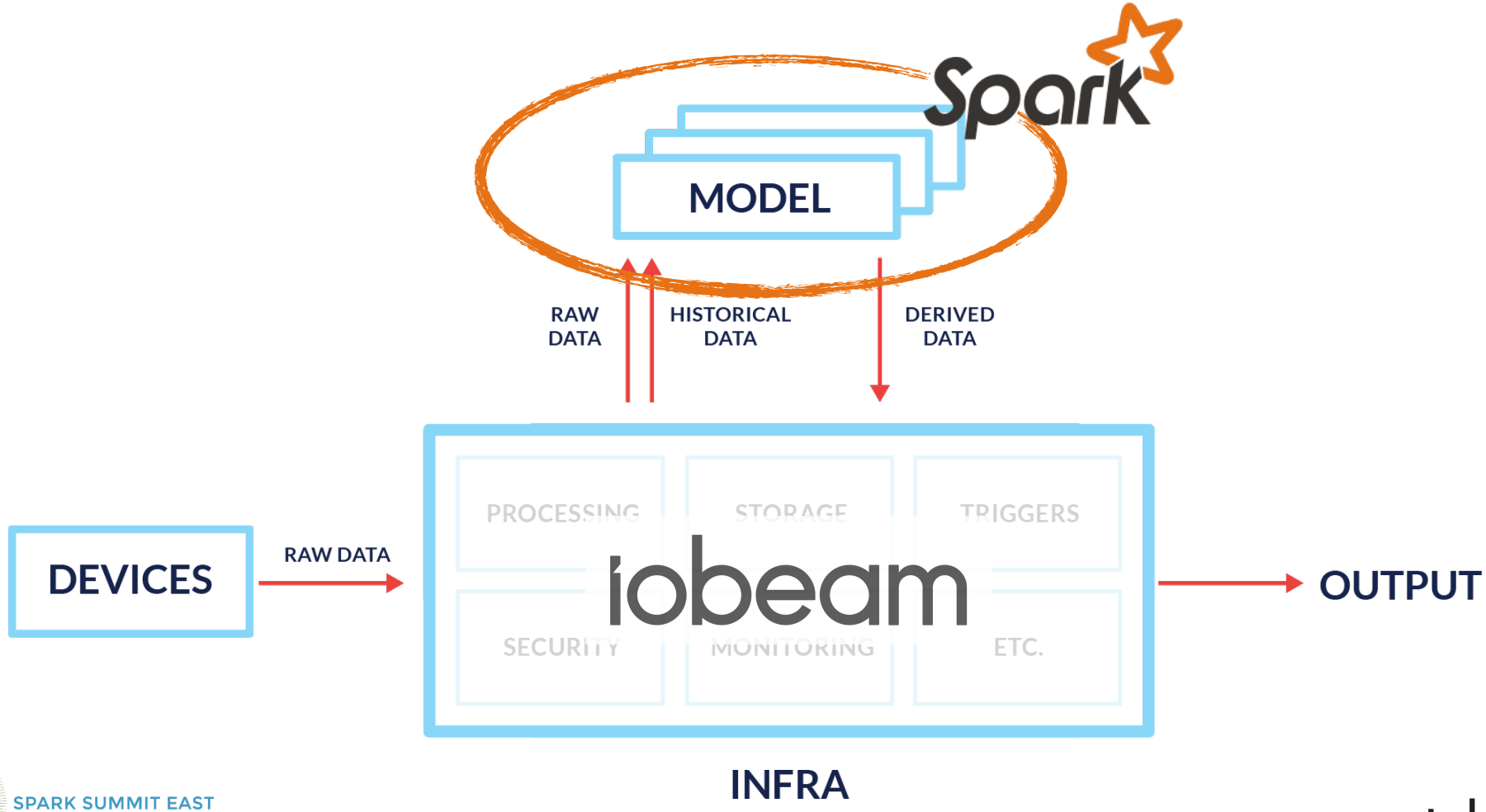
Required: Programming + data infra abstractions

INFRA

Required: Programming + data infra abstractions



Device-Model-Infra (DMI) framework for IoT



Questions?

Developers: docs.iobeam.com

Whitepaper: www.iobeam.com/docs/iobeam-DMI.pdf