How Event-Based Systems Took Over The World:

Combining Performance with Complex Algorithms

Peter Pietzuch
prp@imperial.ac.uk

Large-Scale Distributed Systems Group
Department of Computing, Imperial College London
http://lsds.doc.ic.ac.uk

DEBS 2016 – Irvine, CA, USA

Events Are Everywhere

More data created than ever

– Generated **2.5 Exabytes** (billion GBs) each day in 2015

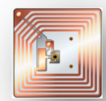
Many new **sources of event data** become available

Internet services, web sites


Social feeds


IoT devices


Cameras


RFID tags


Mobile devices


Scientific instruments


Data repositories

Storage and networking costs become cheaper

– **Hard drive cost per GB** dropped from **$8.93** (2000) to **$0.03** (2014)

☛ Many applications want to exploit these events in real-time…

Intelligent Urban Transport

Instrumentation of urban transport

- Induction loops to measure traffic flow
- Video surveillance of hot spots
- Sensors in public transport

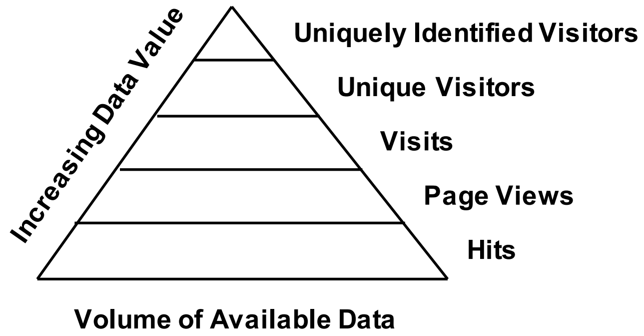
Potential queries

- How to detect traffic congestion and road closures?
- How to explain the cause of congestion (public event, emergency)?
- How to react accordingly (eg by adapting traffic light schedules)?

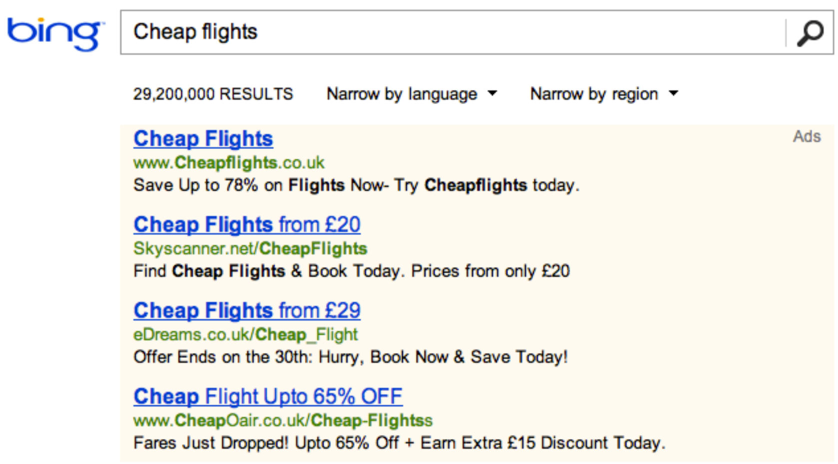
Real-Time Web Analytics

Potential queries

- How to uniquely identify web site visitors?
- How to maximize user experience with relevant content?
- How to analyse “click paths” to trace most common user routes?

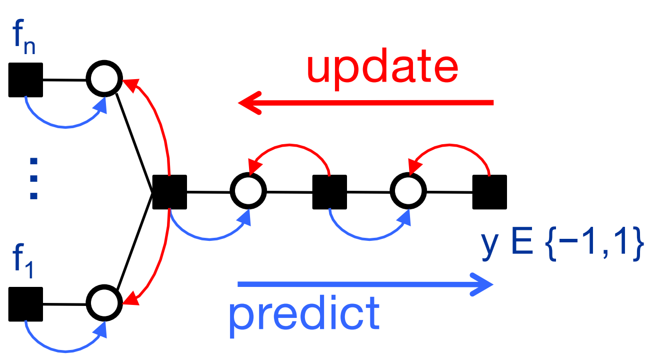


Example: **Online predictions for adverts** to serve on search engines

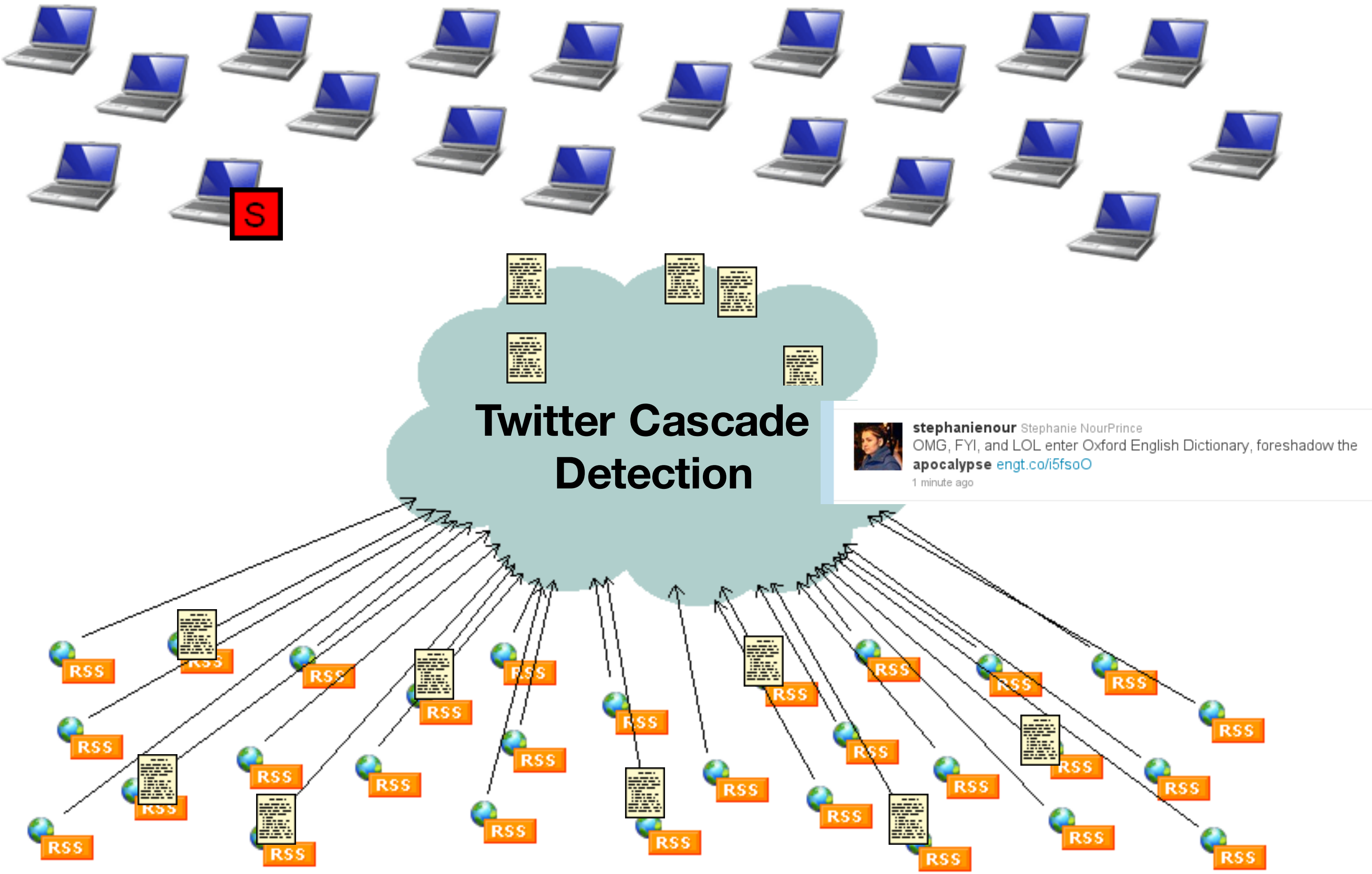


Solution: **AdPredictor**

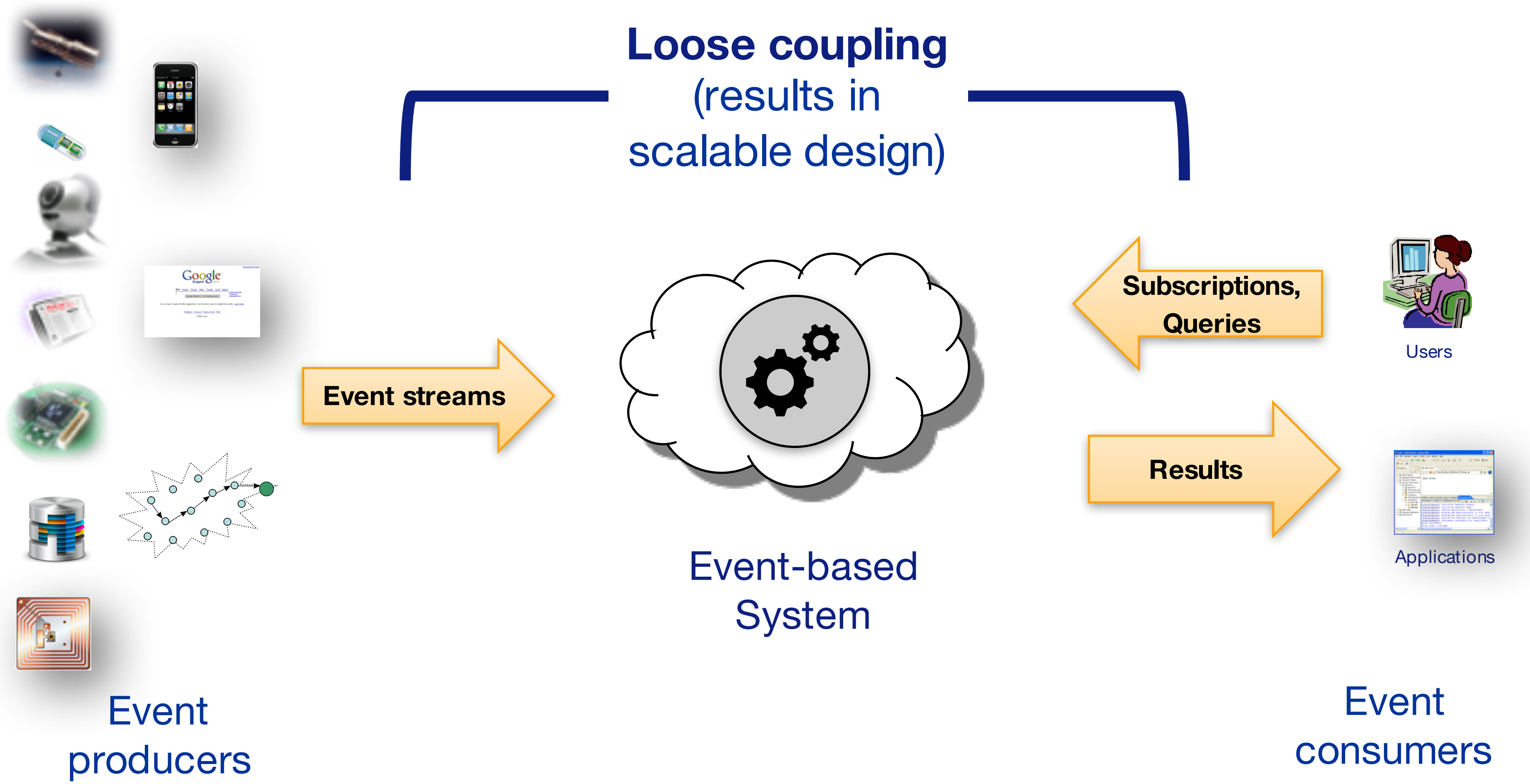
- Bayesian learning algorithm ranks ads according to click probabilities



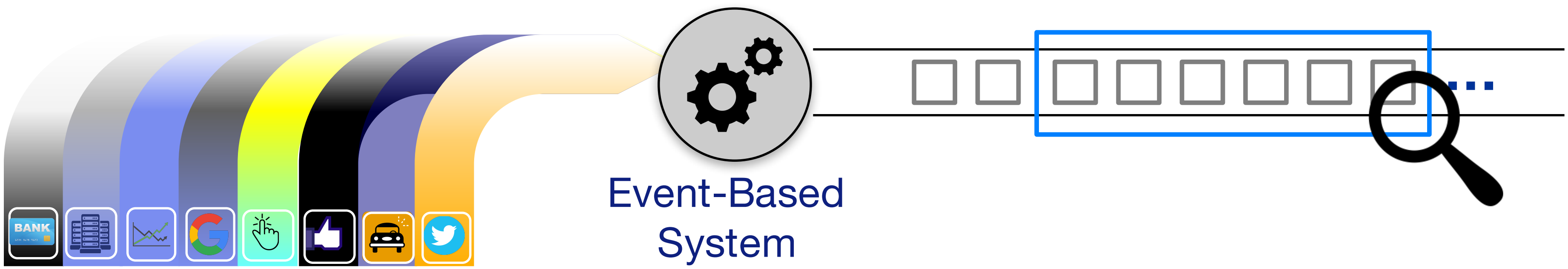
Social Data Mining



Applications Follow An Event-Based Model



Challenge 1: Performance Matters!



High-throughput streams

Low-latency results

Facebook Insights:	Aggregates 9 GB/s	< 10 sec latency
Feedzai:	40K credit card transactions/s	< 25 ms latency
Google Zeitgeist:	40K user queries/s (1 sec windows)	< 1 ms latency
NovaSparks:	150M trade options/s	< 1 ms latency

Challenge 2: Complex Algorithms Matter!

T1

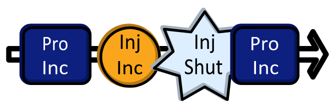
T2

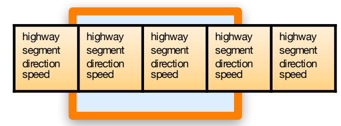
T3

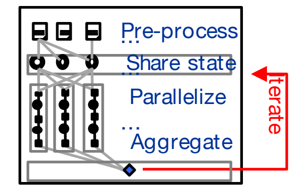
T1(a, b, c)

T2(c, d, e)

T3(g, i, h)







Topic-based filtering → Content-based filtering → Complex pattern matching → Stream queries → Online machine learning, data mining

Publish/Subscribe

Complex Event Processing (CEP)

Stream processing

Roadmap

Introduction to Event-Based Systems

Challenge 1: Performance
How to exploit **parallelism on modern hardware** independently of processing semantics?

Challenge 2: Complex Algorithms
How to support **online machine learning algorithms** over events?

Conclusions

What Is An Event?

An **event** is a happening of interests. An **event type** is a specification of a set of events of the same structure and semantics.
[Etzion and Niblett (2011)]

Events can have fixed **relational schema**

- Payload of event is a set of attributes

highway = M25
segment = 42
direction = north
speed = 85

Vehicle speed data

Vehicles(highway, segment, direction, speed)

What Is An Event Stream?

Event stream is an **infinite sequence of events**

- Assume associated **timestamp** (eg time of reading, time of arrival, …)

t_1	t_2	t_3	t_4	…					
highway segment direction speed	highway segment direction speed	highway segment direction speed	highway segment direction speed	highway segment direction speed	highway segment direction speed	highway segment direction speed	highway segment direction speed	highway segment direction speed	highway segment direction speed

Event stream

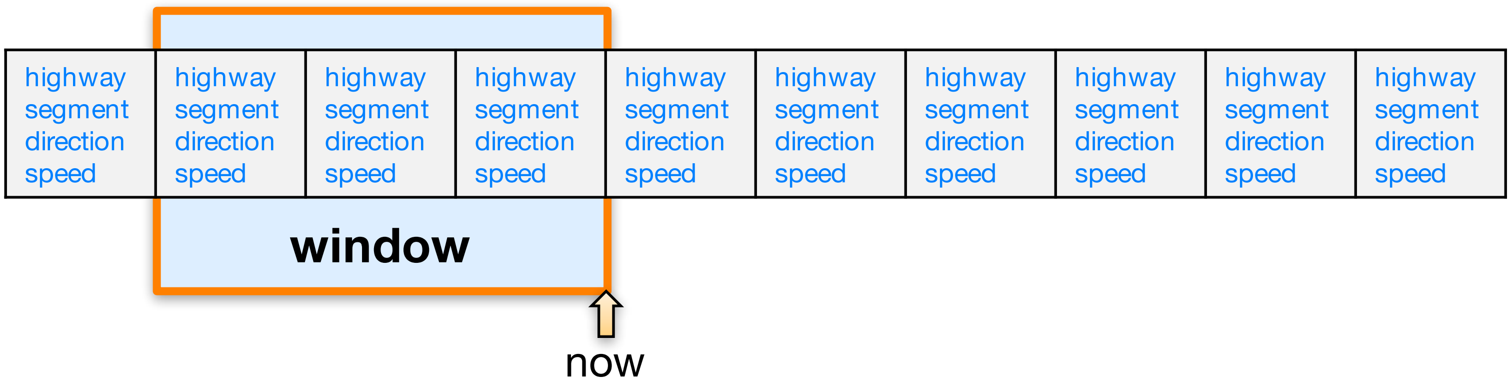
time $\longrightarrow$

☛ But we have an infinite amount of data to process…

How Many Events To Process?

Windows defined **finite set of events** for processing

– Process events in window-sized batches



Time-based window with size τ at current time t

[t - τ : t] **Vehicles[Range τ seconds]**

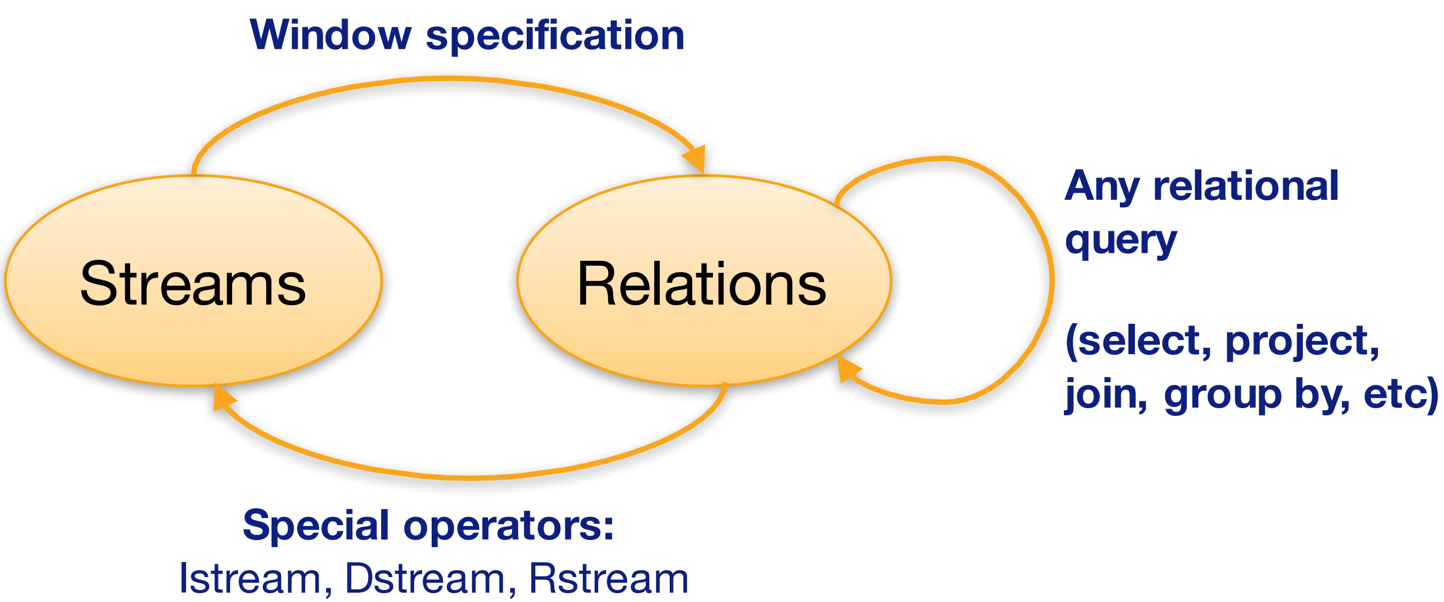
Count-based window with size n:

last n events **Vehicles[Rows n]**

How To Define Event Queries?

Window converts **event stream** to **dynamic relation** (database table)

- Similar to maintaining **database view**
- Use regular relational algebra operators on tuples



Stanford, 2003

CQL: SQL-Based Declarative Queries

CQL provides well-defined semantics for event/stream queries

- Based on well-defined relational algebra (select, project, join, …)

Example: Identify slow moving traffic on highway

- Find highway segments with average speed below 40 km/h

Output

Input stream

```
select      highway, segment,
            direction, AVG(speed) as avg
from        Vehicles[range 5 sec slide 1 sec]
group by    highway, segment, direction
having      avg < 40
```

Operators

☛ Principled way to define event processing semantics…

Roadmap

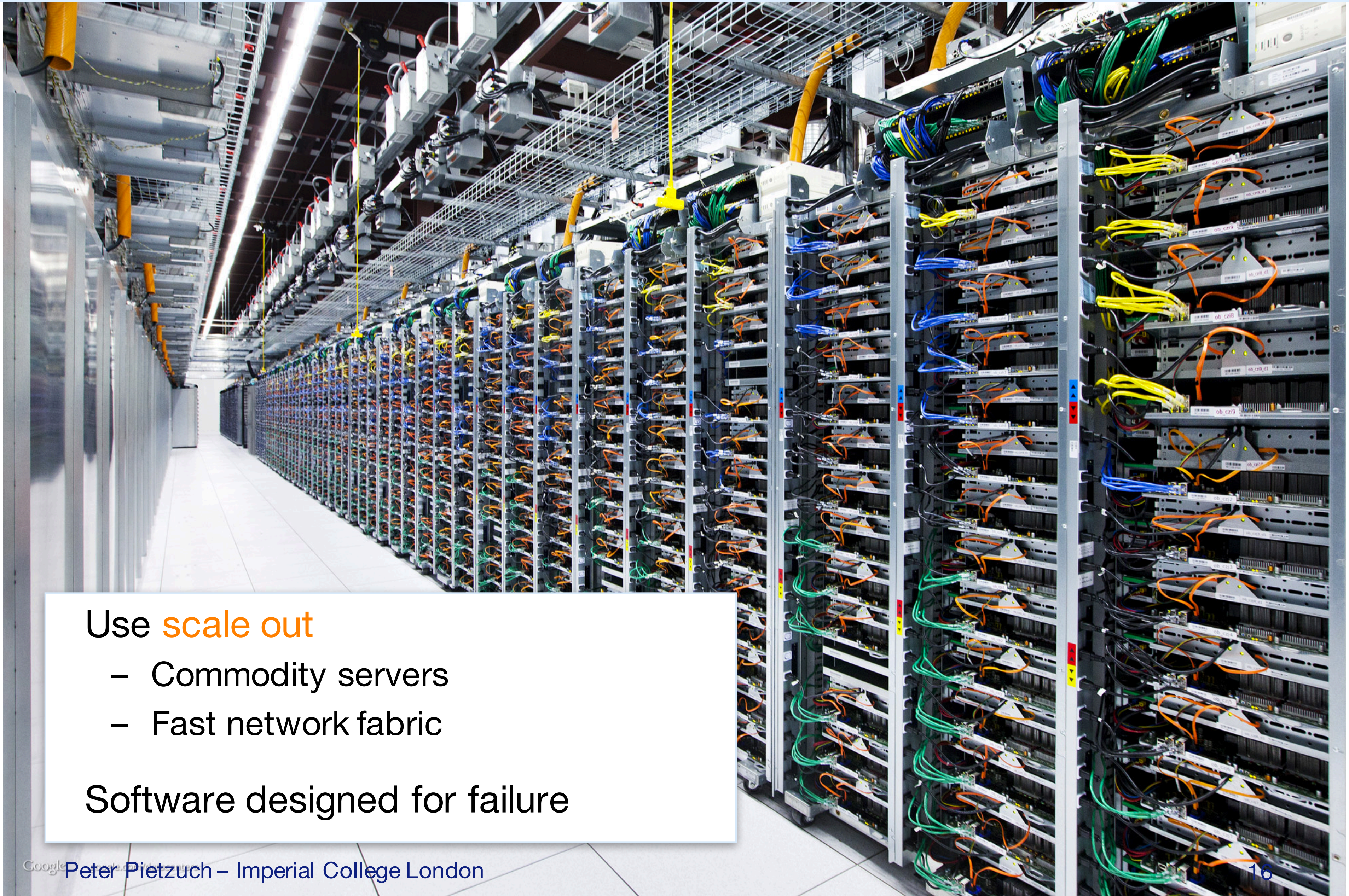
Introduction to Event-Based Systems

Challenge 1: Performance
How to exploit **parallelism on modern hardware** independently of processing semantics?

Challenge 2: Complex Algorithms
How to support **online machine learning algorithms** over events?

Conclusions

How To Scale Big Data Systems?



Use scale out

- Commodity servers
- Fast network fabric

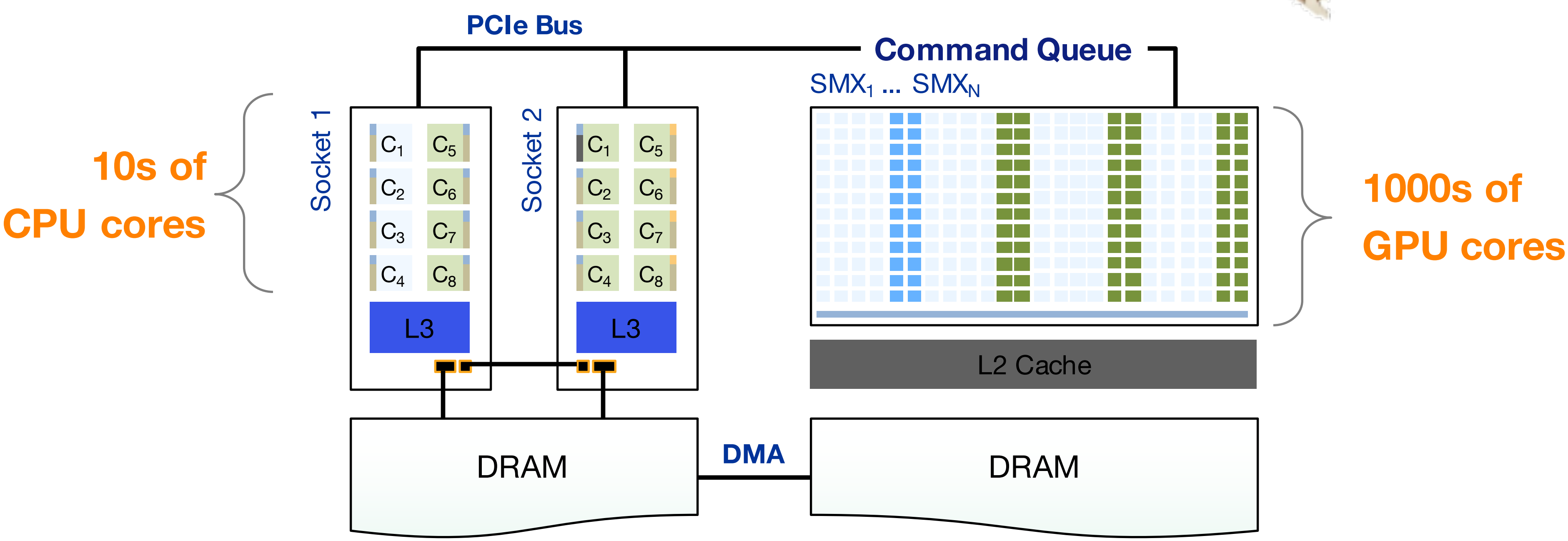
Software designed for failure

But Must Also Exploit Parallel Hardware

Servers have many **parallel CPU cores**

Servers with **GPUs** common

- GPU have even more specialised cores



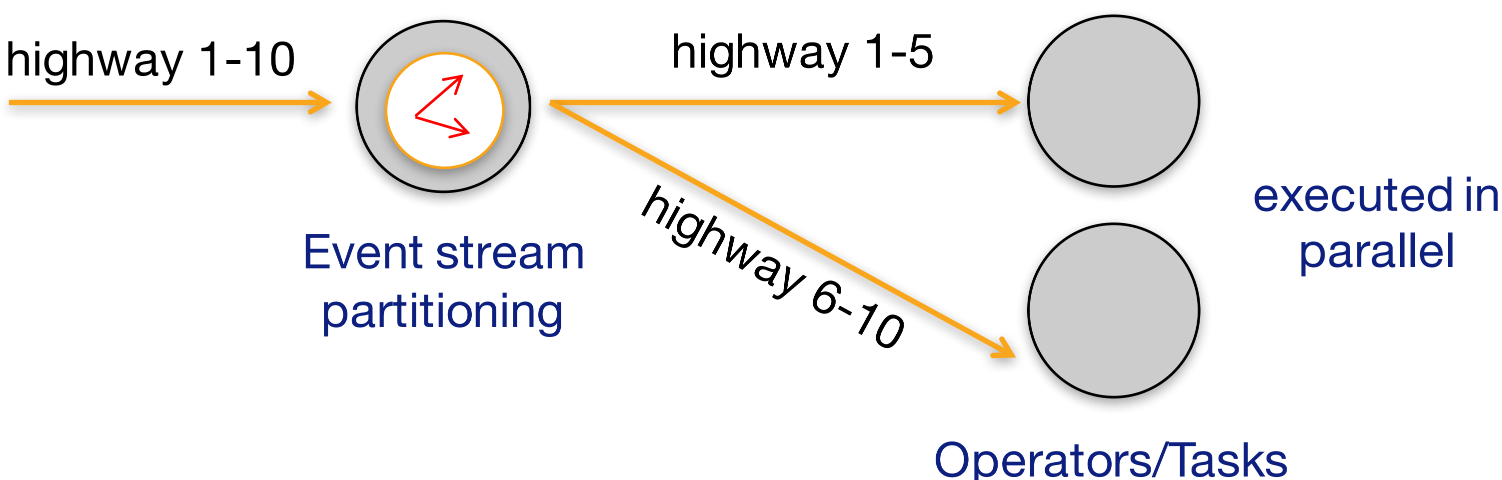
Task Parallelism Vs Data Parallelism

```
select highway, segment, direction, AVG(speed)
from Vehicles[range 5 seconds slide 1 second]
group by    highway, segment, direction
having      avg < 40
```

Task parallelism:
Multiple queries

```
select highway, segment, direction, AVG(speed)
from Vehicles[range 5 seconds slide 1 second]
group by    highway, segment, direction
having      avg < 40
```

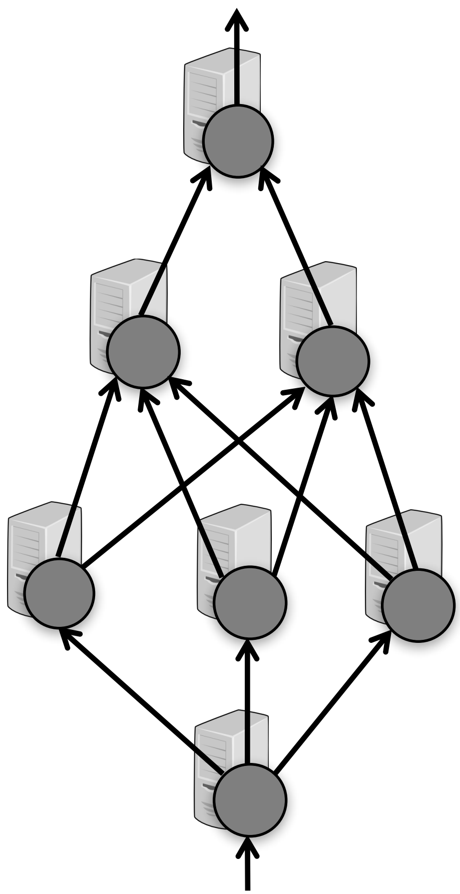
Data parallelism:
Single query



Apache Storm: Dataflow Graphs

parallelism degree 2

parallelism degree 3



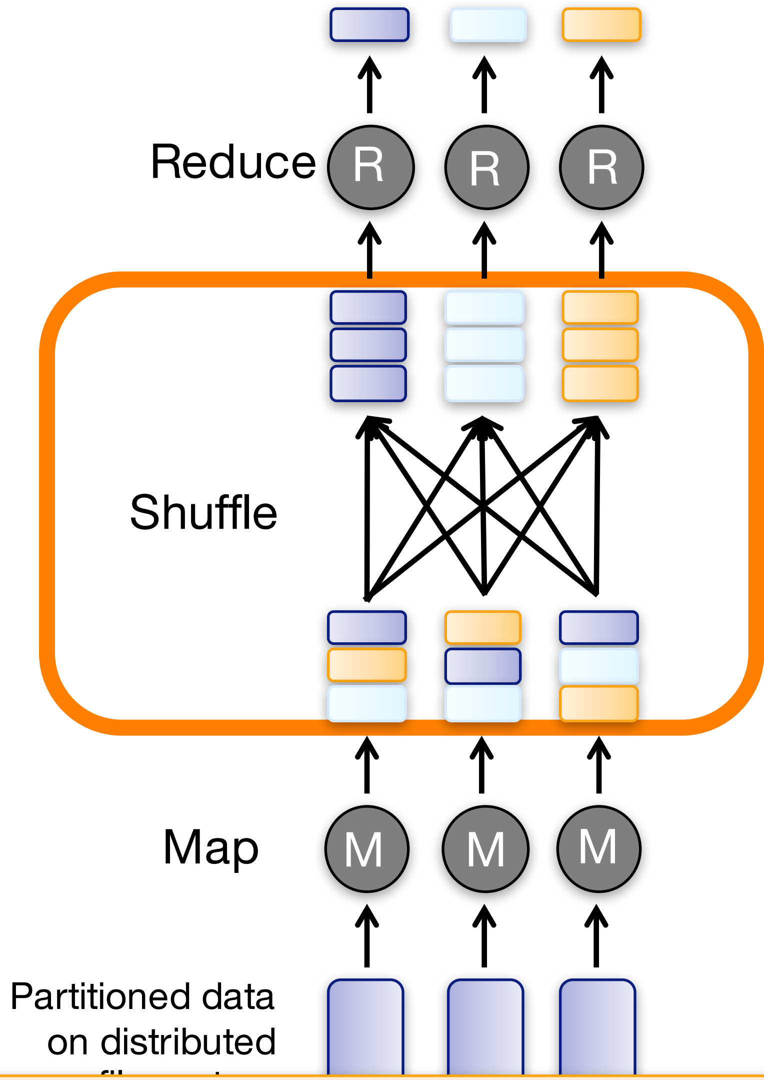
Idea:
Execute event operators as data-parallel **tasks**

Task organised as **dataflow graph**

Many systems do this, e.g. Apache Storm, Apache Flink, Google Dataflow, …

☛ But must manually assign tasks to nodes…

Use Apache Hadoop For Event Processing?



MapReduce model

- Data model: (key, value) pairs
- Two processing functions:

$map(k_1,v_1) \rightarrow list(k_2,v_2)$

$reduce(k_2, list(v_2)) \rightarrow list\ (v_3)$

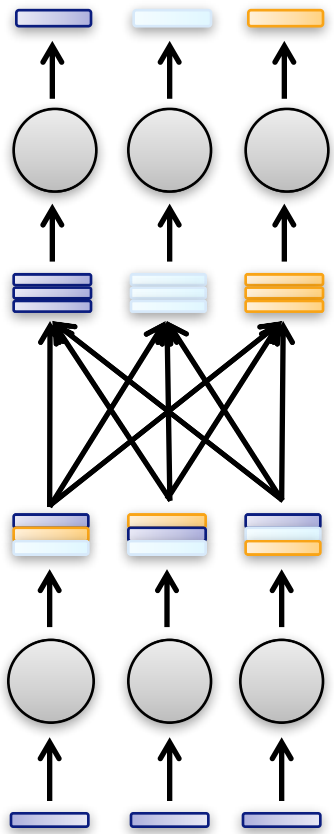
Benefits

- Simple programming model
- Transparent parallelisation
- Fault-tolerant processing

☛ Shuffle phase introduces synchronisation barrier (batch processing)

UC Berkeley, SOSP'13

Apache Spark: Micro-Batching



Event stream, divided into micro-batches

Idea:
Reduce size of data partitons to produce up-to-date, incremental results

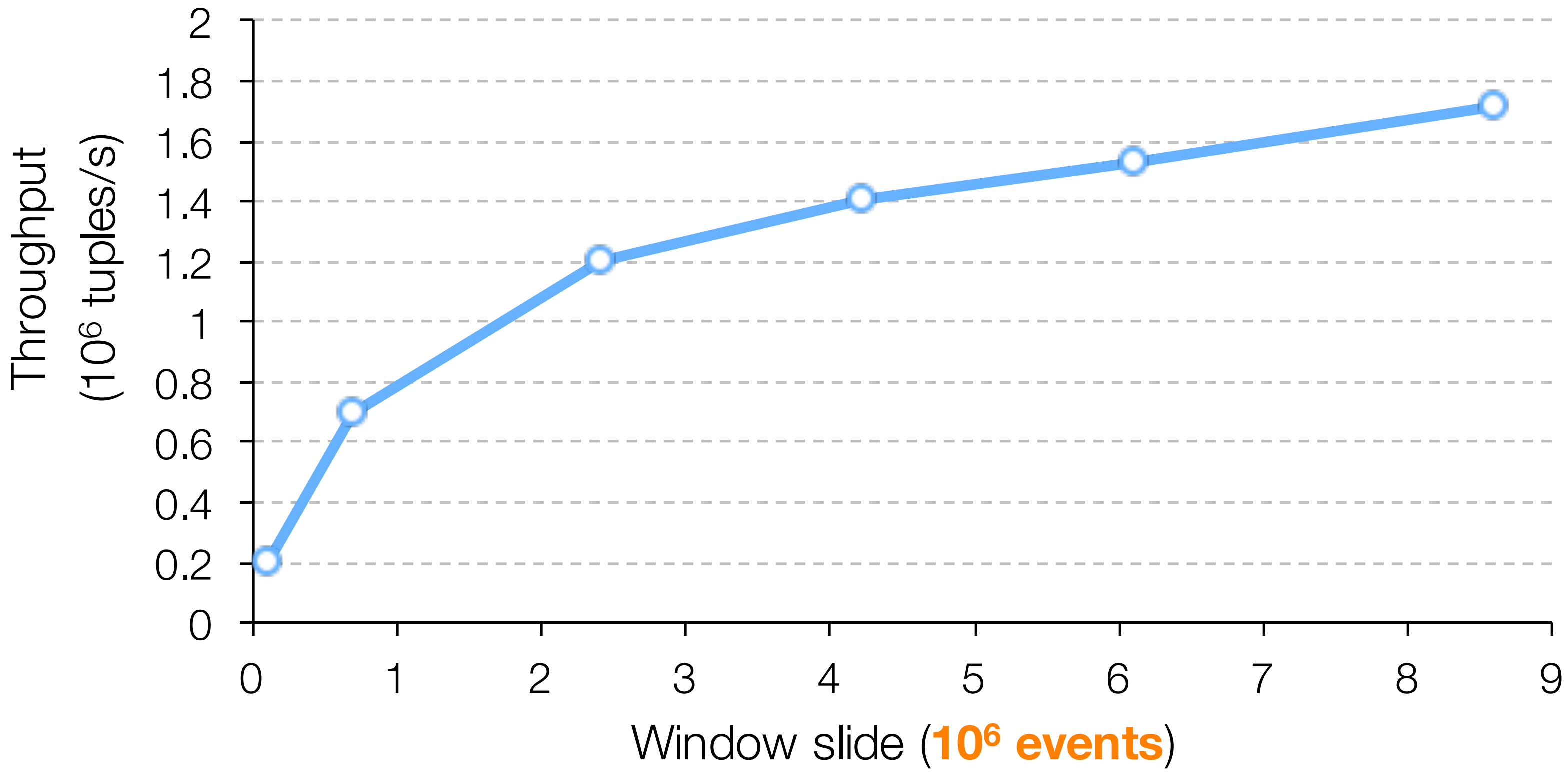
Micro-batching for event data

- Tasks operate on micro-batch partitions
- Results produced with low latency

☛ Interaction of query windows and micro-batches?

Spark: Small Slides Result In Low Throughout

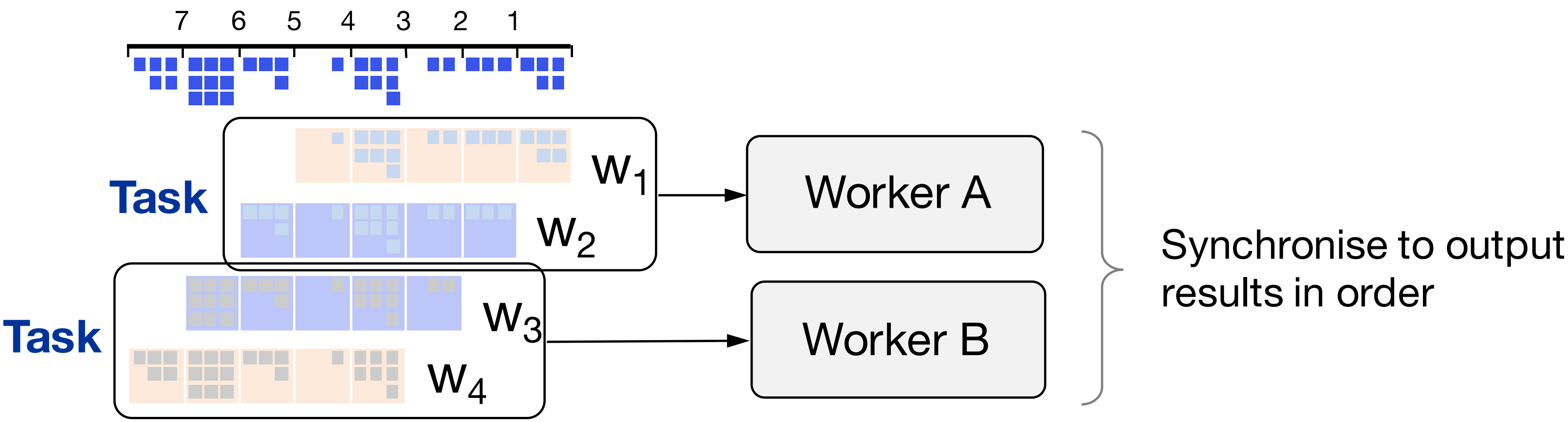
select AVG(S.1) **from** S [**rows** 1024 **slide** ***x***]



☛ Want to avoid coupling performance with query definition

How To Parallelise Sliding Windows?

select highway, segment, direction, AVG(speed) **as** avg
from Vehicles[**range** 5 seconds **slide** 1 second]
group by highway, segment, direction
having avg < 40

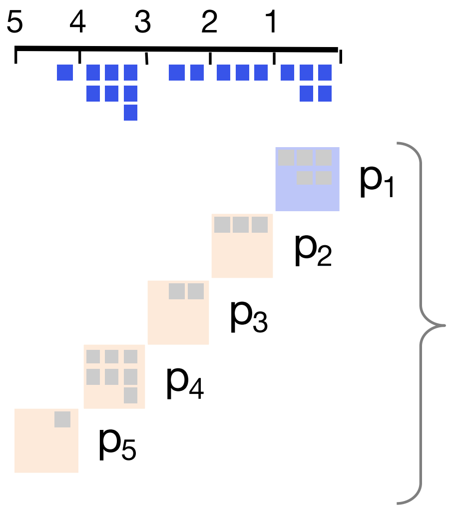


☛ Leads to **redundant computation**

Avoiding Redundant Computation

Use **panes** to remove window overlap between tasks

- Smallest unit of parallelism without data dependencies between windows



Panes processed in parallel

Window results assembled from pane results

Apache Spark uses **panes** for **micro-batches** with windowed queries

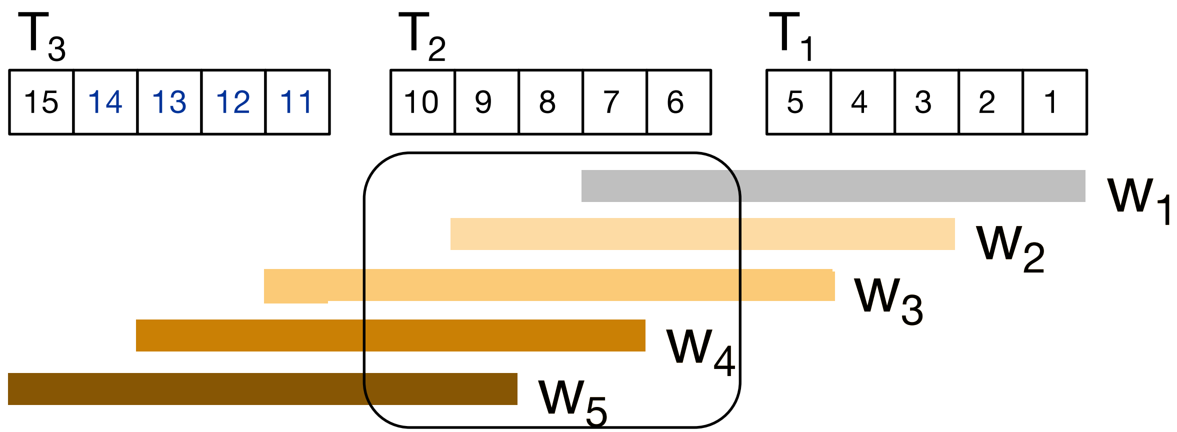
☛ **Micro-batch size** limited by pane size

☛ **Window slide** limited by minimum micro-batch size (~500 ms)

SABER: Window Fragment Model

Idea: **Decouple task size from window size/slide**

- e.g. 5 events/task, window size 7 rows, slide 2 rows



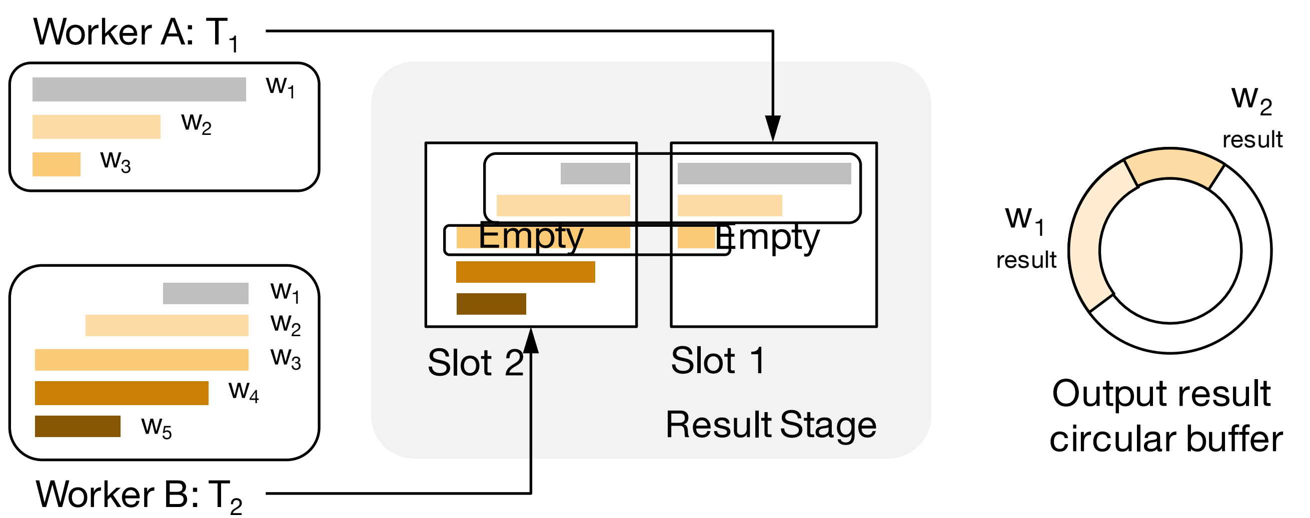
Task contains one or more **window fragments**

- Closing/pending/opening windows in T_2
- Workers process fragments incrementally

Merging Window Fragment Results

Idea: **Decouple task size from window size/slide**

- Assemble window fragment results
- Output them in correct order

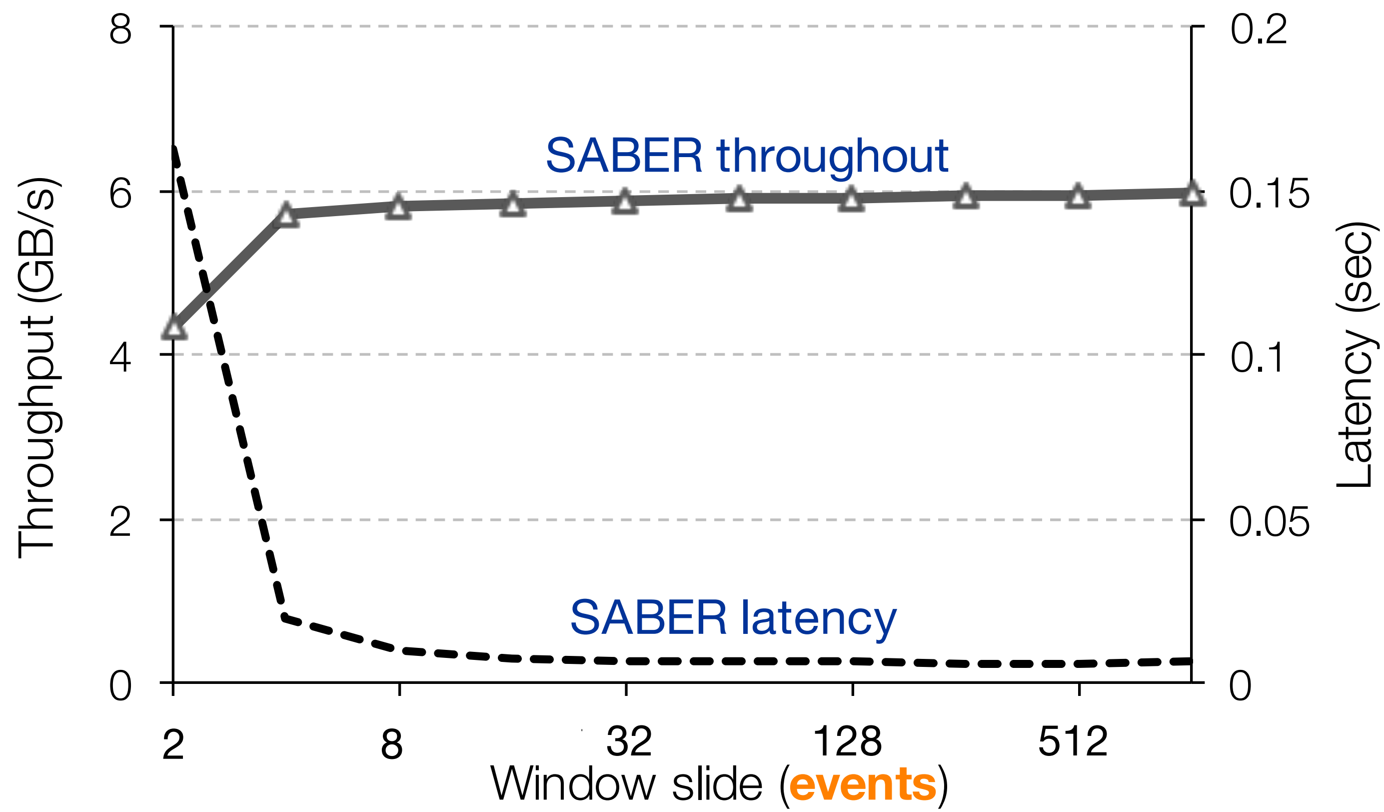


Worker A stores T_1 results and exits (nothing to forward)

Worker B stores T_2 results, merges window fragment results and forwards complete windows downstream

SABER: Window Performance

select AVG(S.1) **from** S [**rows** 1024 **slide** ***x***]



☛ Performance with windowed queries remains predictable

Roadmap

Introduction to Event-Based Systems

Challenge 1: Performance
How to exploit **parallelism on modern hardware** independently of processing semantics?

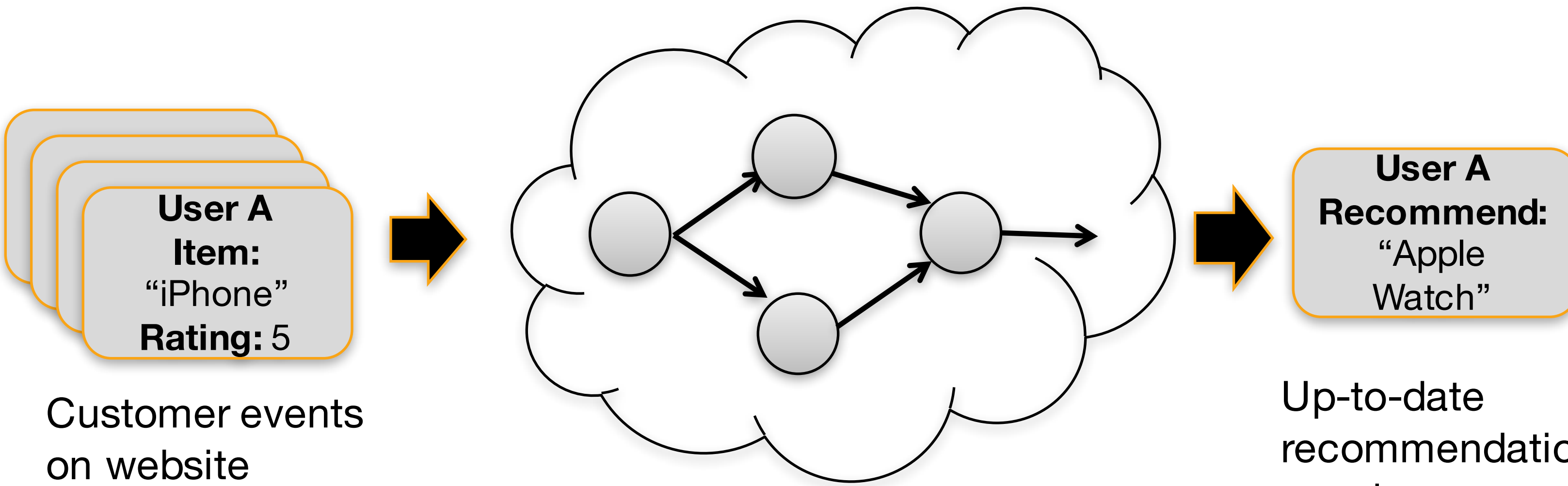
Challenge 2: Complex Algorithms
How to support **online machine learning algorithms** over events?

Conclusions

Supporting Online Machine Learning

Online recommender system

- Recommendations based on past user ratings
- Eg based on **collaborative filtering** (cf Netflix, Amazon, …)



Customer events on website

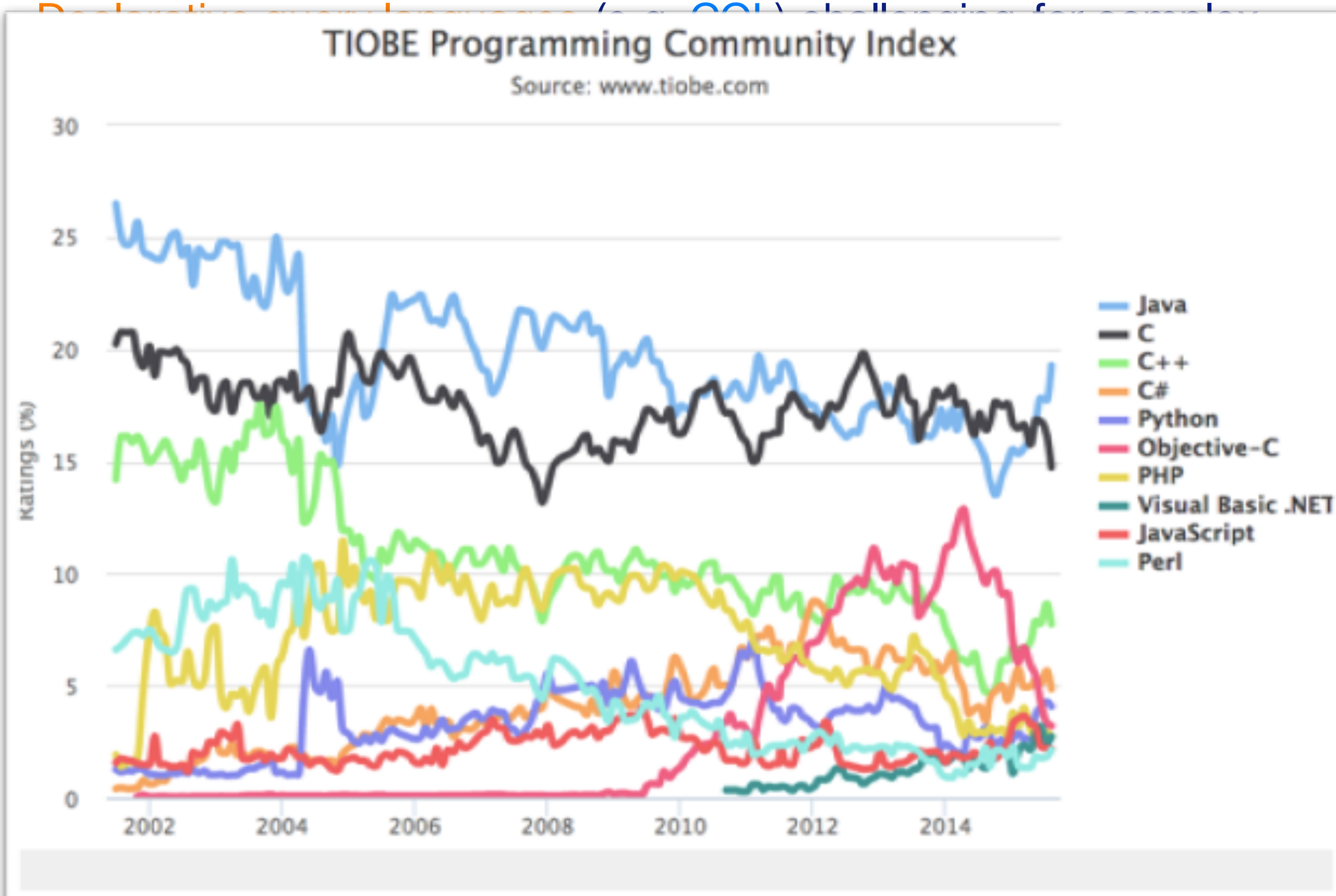
Executed as dataflow graph

(eg Storm, Spark, Flink, …)

Up-to-date recommendation events

☛ What programming abstraction to use to specify the algorithm?

Programming Models For Event Processing?



Online Collaborative Filtering In Java

Update with new ratings

	Item-A	Item-B
User-A	4	5
User-B	0	5

User-Item matrix (**UI**)

Multiply for recommendation

User-B	1	2

x

	Item-A	Item-B
Item-A	1	1
Item-B	1	2

Co-Occurrence matrix (**CO**)

```
Matrix userItem = new Matrix();
Matrix coOcc = new Matrix();


void processRatingEvent(int user, int item,
                        int rating) {
    userItem.setElement(user, item, rating);
    updateCoOccurrence(coOcc, userItem);
}
Vector processRecEvent(int user) {
    Vector userRow = userItem.getRow(user);
    Vector userRec = coOcc.multiply(userRow);
    return userRec;
}
```

Collaborative Filtering In Spark (Java)

```
// Build the recommendation model using ALS
int rank = 10;
int numIterations = 20;
MatrixFactorizationModel model = ALS.train(JavaRDD.toRDD(ratings), rank, numIterations, 0.01);

// Evaluate the model on rating data
JavaRDD<Tuple2<Object, Object>> userProducts = ratings.map(
  new Function<Rating, Tuple2<Object, Object>>() {
    public Tuple2<Object, Object> call(Rating r) {
      return new Tuple2<Object, Object>(r.user(), r.product());
    }
  }
);
JavaPairRDD<Tuple2<Integer, Integer>, Double> predictions = JavaPairRDD.fromJavaRDD(
  model.predict(JavaRDD.toRDD(userProducts)).toJavaRDD().map(
    new Function<Rating, Tuple2<Tuple2<Integer, Integer>, Double>>() {
      public Tuple2<Tuple2<Integer, Integer>, Double> call(Rating r){
        return new Tuple2<Tuple2<Integer, Integer>, Double>(
          new Tuple2<Integer, Integer>(r.user(), r.product()), r.rating());
      }
    }
));
JavaRDD<Tuple2<Double, Double>> ratesAndPreds = JavaPairRDD.fromJavaRDD(ratings.map(
    new Function<Rating, Tuple2<Tuple2<Integer, Integer>, Double>>() {
      public Tuple2<Tuple2<Integer, Integer>, Double> call(Rating r){
        return new Tuple2<Tuple2<Integer, Integer>, Double>(
          new Tuple2<Integer, Integer>(r.user(), r.product()), r.rating());
      }
    }
)).join(predictions).values();
```

Collaborative Filtering In Spark (Scala)

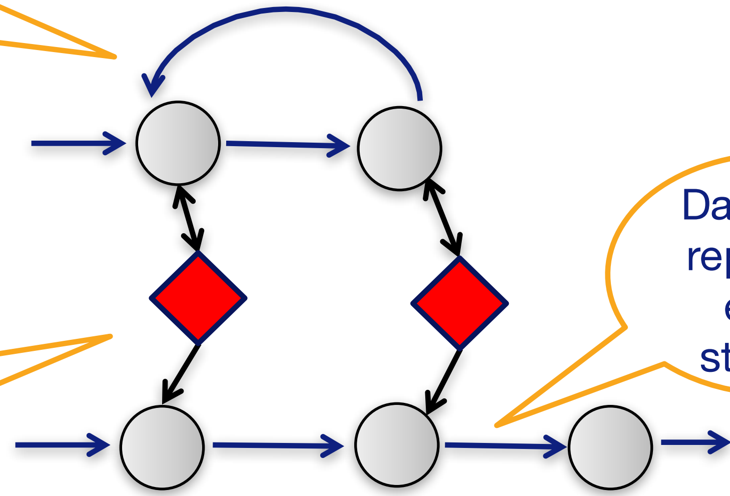
```
// Build the recommendation model using ALS
val rank = 10
val numIterations = 20
val model = ALS.train(ratings, rank, numIterations, 0.01)

// Evaluate the model on rating data
val usersProducts = ratings.map {
  case Rating(user, product, rate) => (user, product)
}
val predictions =
  model.predict(usersProducts).map {
    case Rating(user, product, rate) => ((user, product), rate)
  }
val ratesAndPreds = ratings.map {
  case Rating(user, product, rate) => ((user, product), rate)
}.join(predictions)
```

☛ All event data is immutable, no fine-grained model updates

Processing State As First Class Citizen

Tasks process event data

	Item 1	Item 2
User A	2	5
User B	4	1



Dataflows represent event streams

State Elements (SEs) represent transient state

State elements (SEs) are mutable in-memory data structures

- Tasks have **local access** to SEs
- SEs can be shared between tasks

Challenges With Large Processing State

```
Matrix userItem = new Matrix();
Matrix coOcc = new Matrix();
```

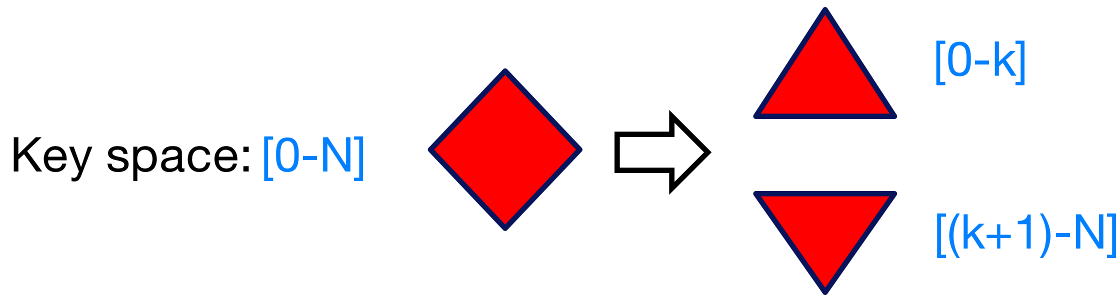
Big Data problem: **Matrices become large**

State will not fit into single node

☛ How to handle distributed state in a scalable fashion?

Partitioned State Elements

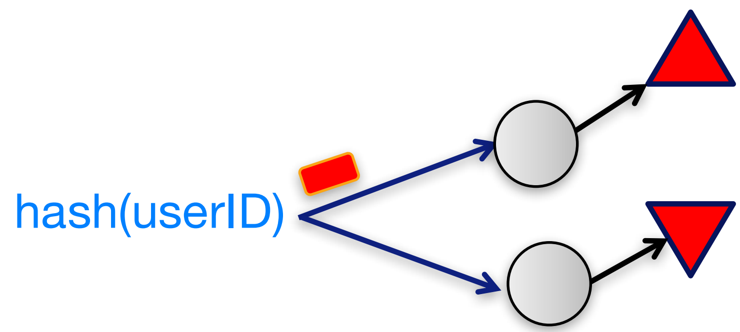
Idea: **Partitioned SEs** are split into disjoint partitions



Access by key

User-Item matrix (UI)

	Item-A	Item-B
User-A	4	5
User-B	0	5



State **partitioned** according to partitioning key

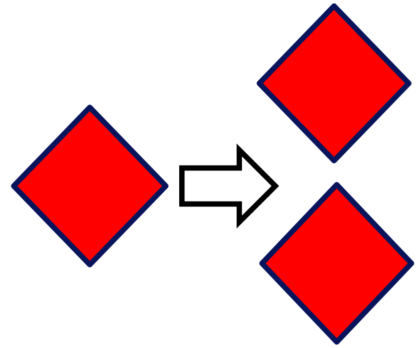
Dataflow **routed** according to hash function

Partial State Elements

Partial SEs are replicated (when partitioning is not possible)

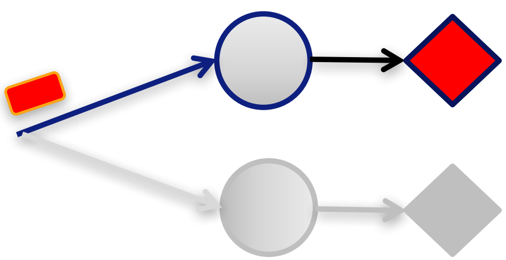
Co-Occurrence matrix (**CO**)

	Item-A	Item-B
Item-A	1	1
Item-B	1	2

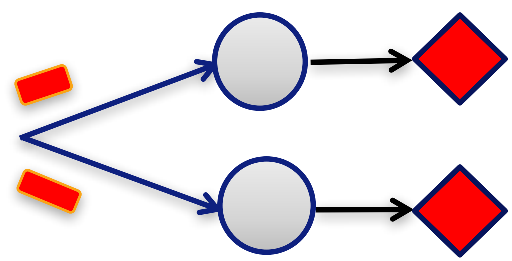


– Replicas kept **weakly consistent**

Access to partial SEs either **local** or **global**

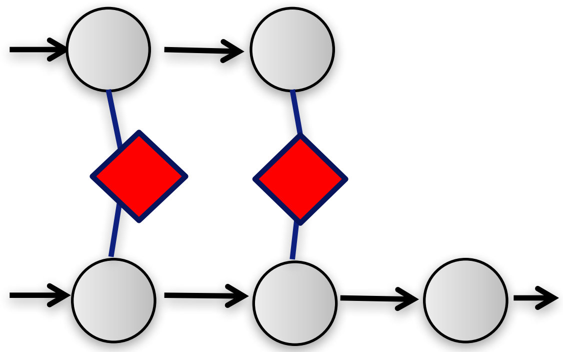


Local access:
Events sent to one



Global access:
Events sent to all

Scalable & Elastic Event Processing (SEEP)



Annotated Java program

(@Partitioned, @Partial, @Global, ...)

Program.java

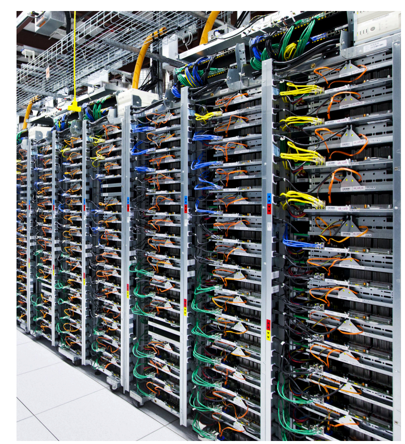
Static program analysis

Data-parallel
Stateful Dataflow Graph (SDG)

SEEP distributed dataflow framework

Translation & checkpoint-based fault tolerance

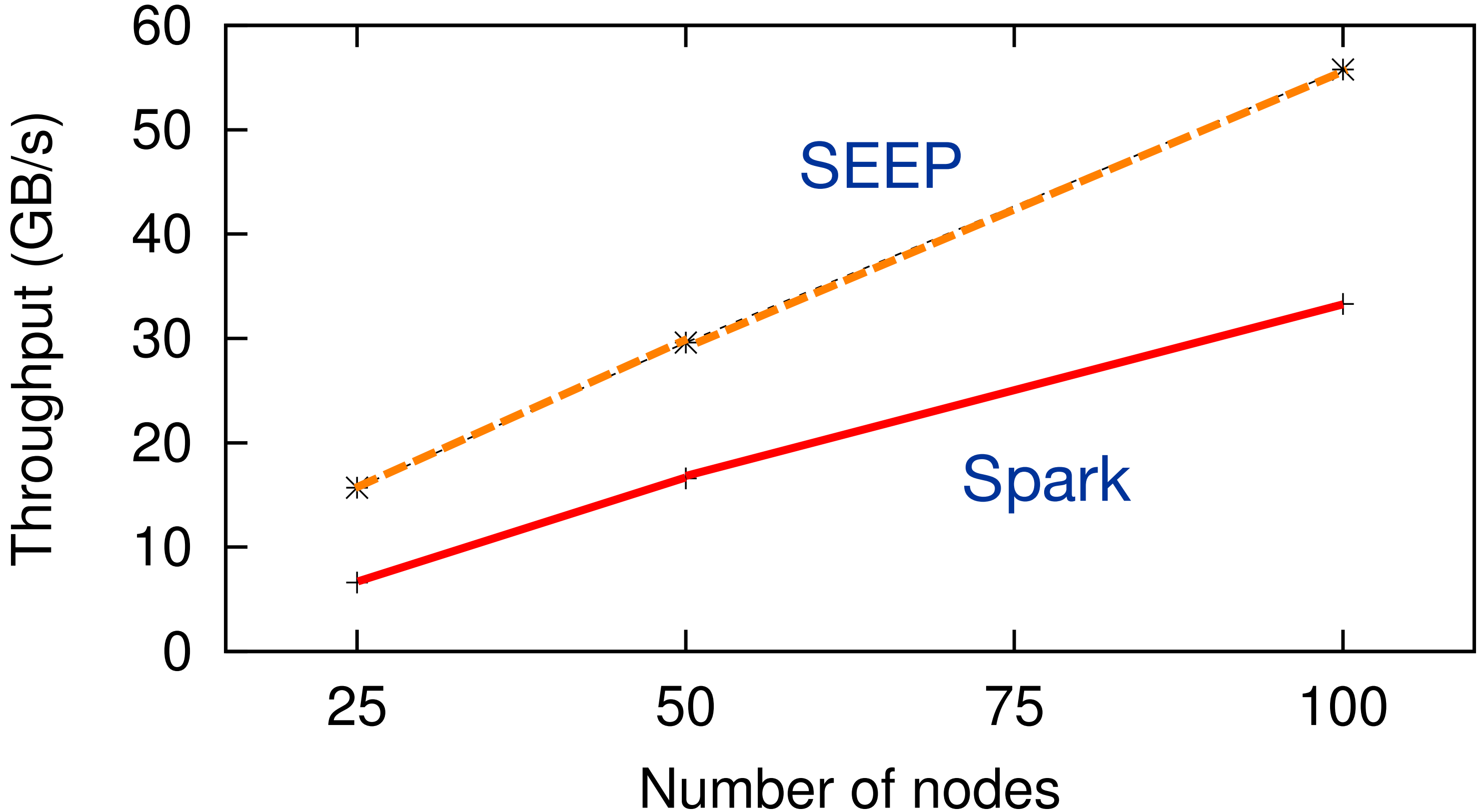
Cluster



SEEP: Online Logistic Regression

100 GB training dataset for classification

Deployed on Amazon EC2 (“m1.xlarge” VMs with 4 vCPUs and 16 GB RAM)



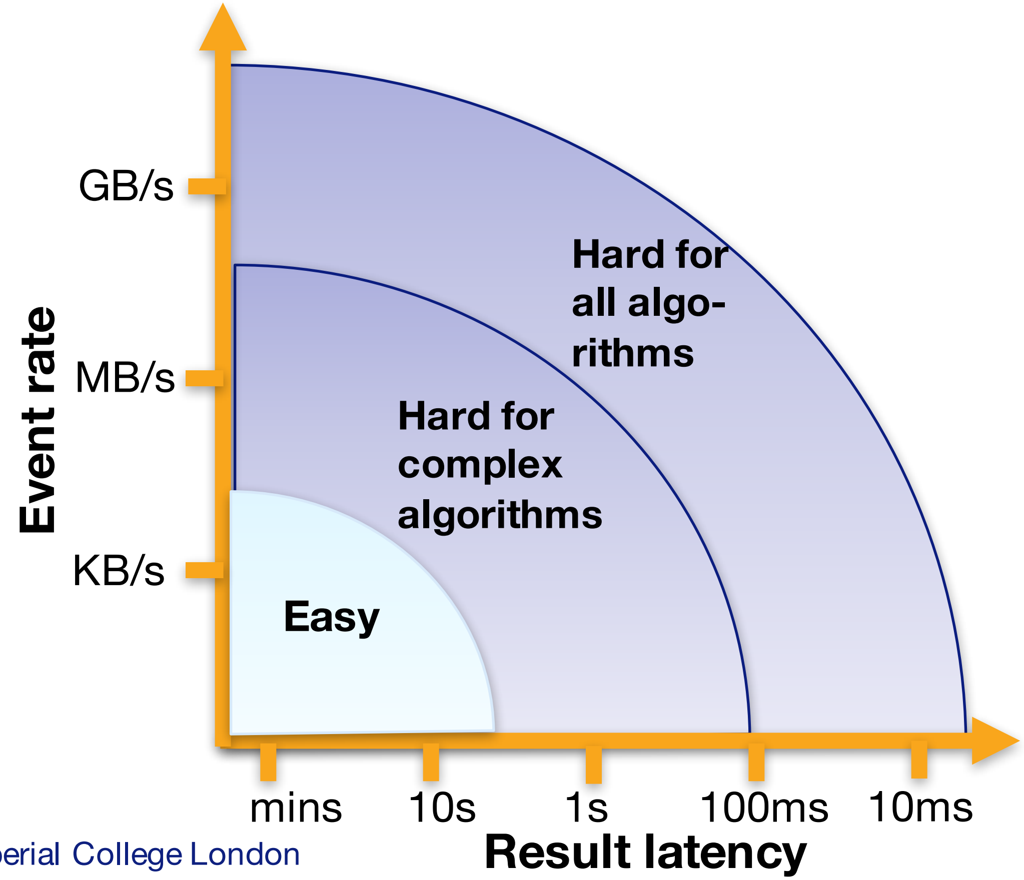
☛ SEEP has comparable throughput to Spark despite mutable state

Conclusions I

Event-based systems are a crucial part of many data processing stacks

- Many applications and services require real-time view of event streams
- Batch processing models increasingly replaced by event processing

Interesting tension between **performance** and **algorithmic complexity**



Conclusions II

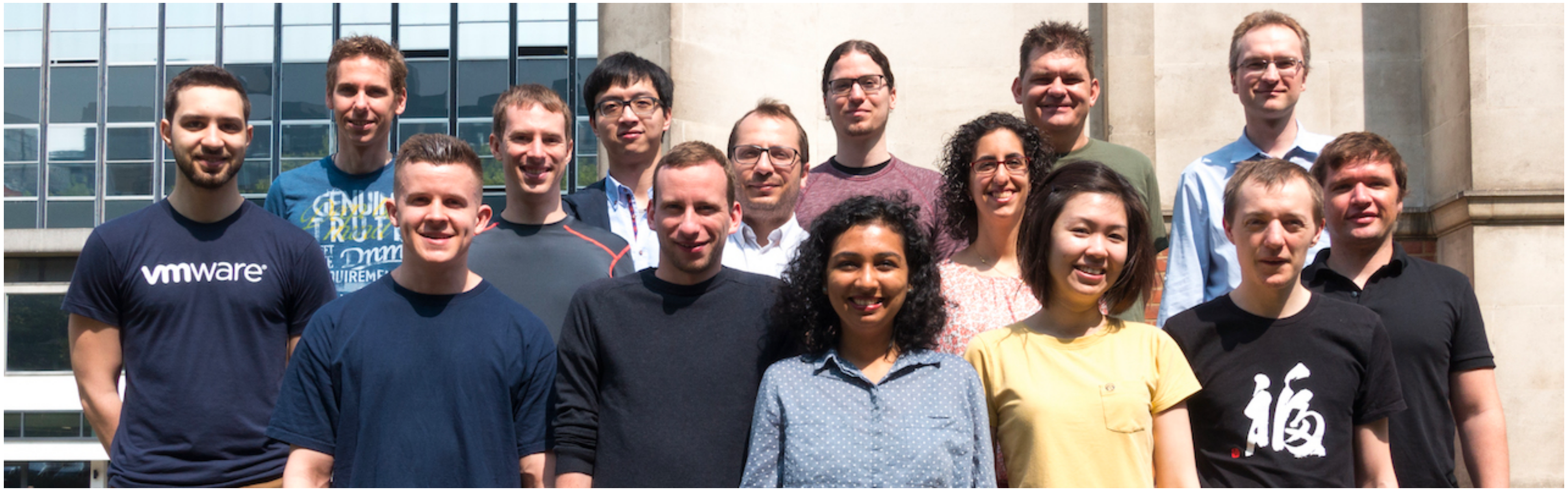
1. Modern **parallel hardware** (multicore CPUs/GPUs) raises challenges
   - New event-based system designs must exploit **data parallelism**
   - But must not couple performance with **processing semantics**

☛ **Principled window handling in parallel event processing**

2. **Online machine learning** over events is killer app
   - Complex streaming applications require **expressive programming models**
   - Want to offer **natural programming abstractions** to users

☛ **Stateful event processing for machine learning**

Acknowledgements – LSDS Group



Thank you! Any Questions?

Peter Pietzuch
<prp@doc.ic.ac.uk>
http://lsds.doc.ic.ac.uk