Imperial College London

Department of Computing

Drinking From The Fire Hose: The Rise of Distributed Stream Processing Systems

Peter Pietzuch
prp@doc.ic.ac.uk

Large-Scale Distributed Systems Group
http://lsds.doc.ic.ac.uk

Cambridge MPhil – February 2012


THE
END
IS
NEAR

The Data Deluge

150 Exabytes (billion GBs) created in 2005 alone

– Increased to 1200 Exabytes in 2010

Many new sources of data become available

– Sensors, mobile devices
– Web feeds, social networking
– Cameras
– Databases
– Scientific instruments


☛ **How can we make sense of all data ?**

– Most data is not interesting
– New data supersedes old data
– Challenge is not only storage but also querying

Real Time Traffic Monitoring

Instrumenting country's transportation infrastructure

Time-EACM
(Cambridge)

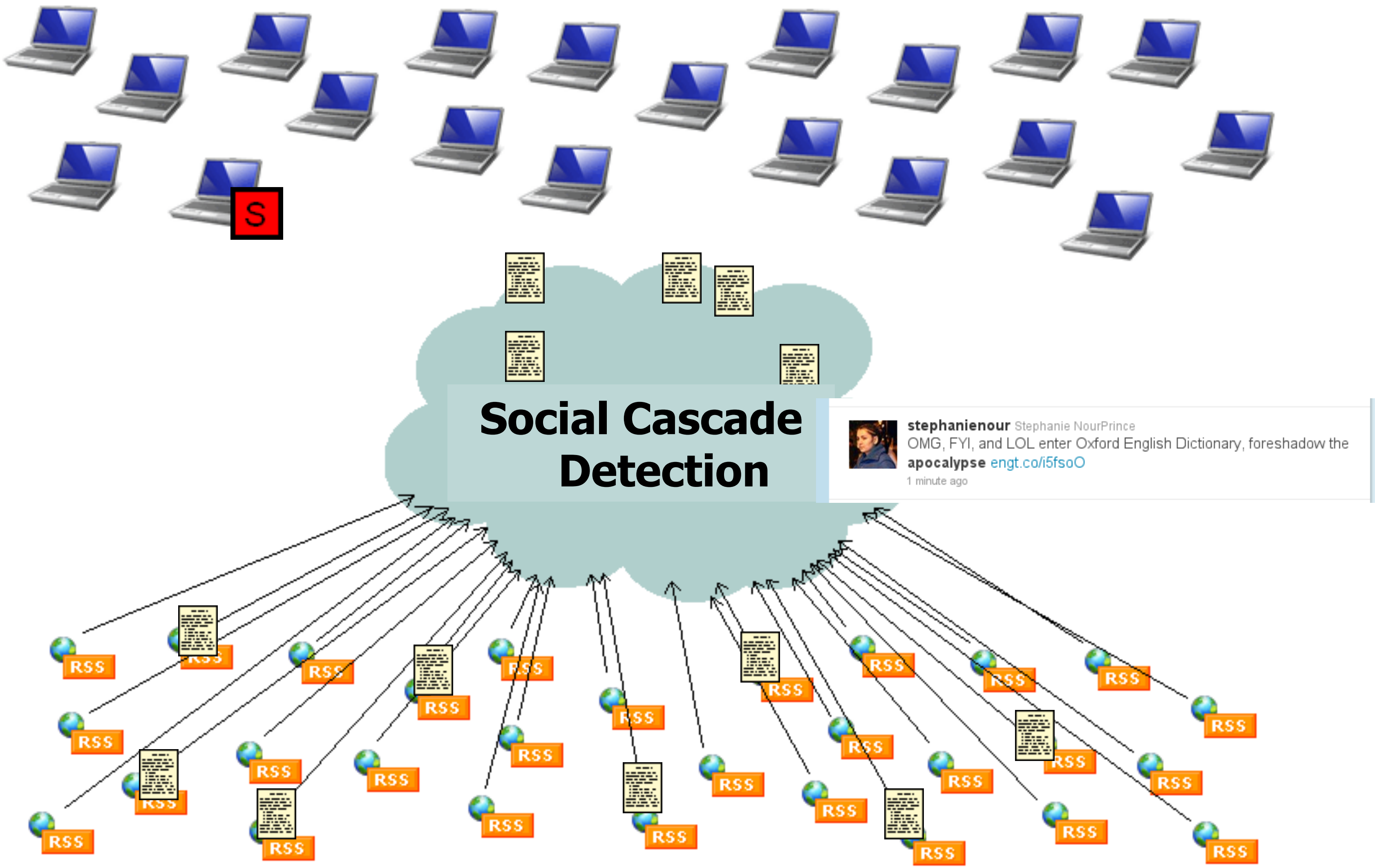
Many parties interested in data

- Road authorities, traffic planners, emergency services, commuters
- But access not everything: Privacy

High-level queries

- "What is the best time/ route for my commute through central London between 7-8am?"

Web/Social Feed Mining



Detection and reaction to social cascades

Fraud Detection

How to detect identity fraud as it happens?

Illegal use of mobile phone, credit card, etc.

- Offline: avoid aggravating customer
- Online: detect and intervene

Huge volume of call records

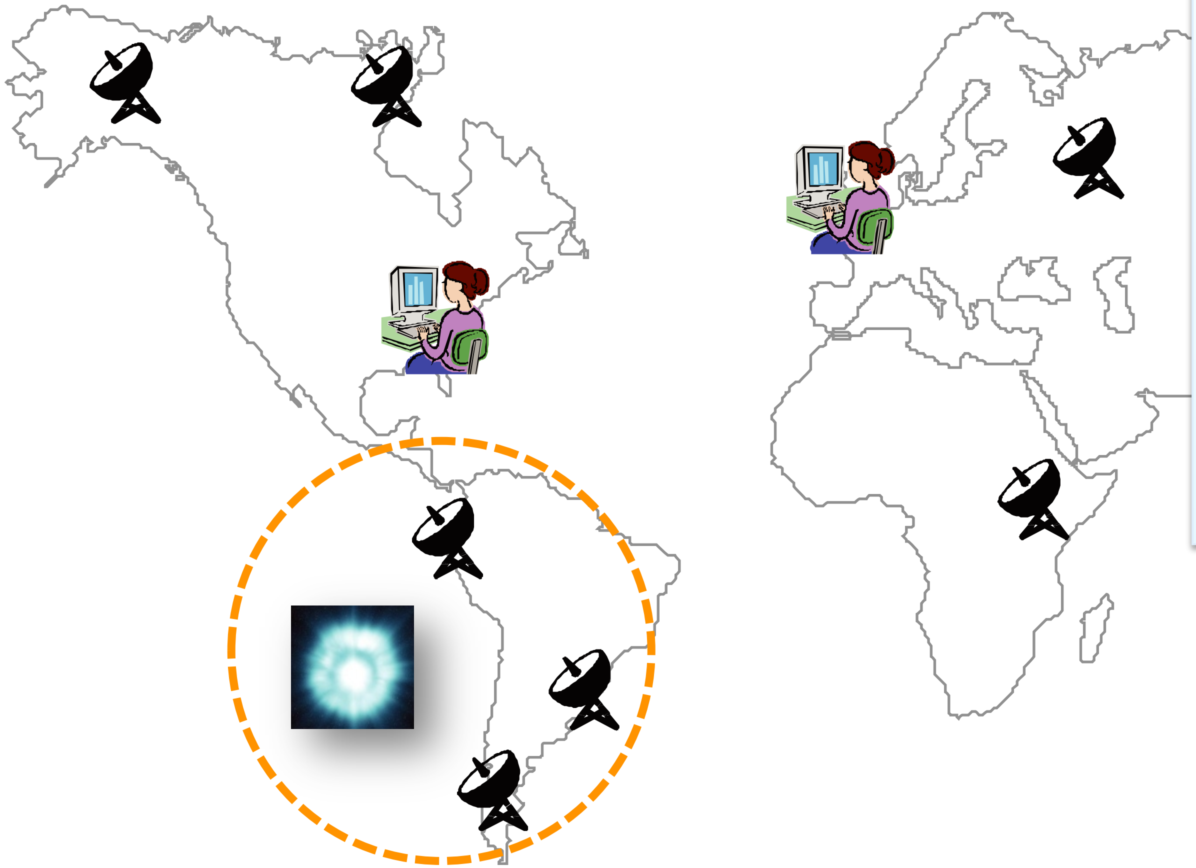
More sophisticated forms of fraud

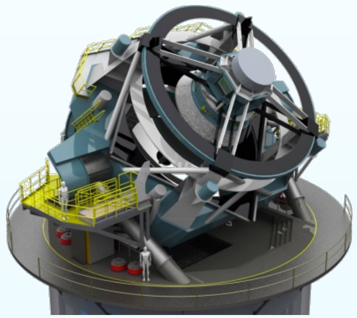
- e.g. insider trading

Supervision of laws and regulations

- e.g. Sabanes-Oxley, real-time risk analysis

Astronomic Data Processing





Large Synoptic Survey Telescope (LSST)

- Generates 1.28 Petabytes per year

Analysing transient cosmic events: γ-ray bursts

Global Sensor Applications: EarthScope

Using sensors to understand geological evolution

– Many sources: 400 seismometers, 1000 GPS stations, …

Stream Processing to the Rescue!

☛ Process data streams on the fly without storage

Stream data rates can be high
- High resource requirements for processing (clusters, data centres)

Processing stream data has real-time aspect
- Latency of data processing matters
- Must be able to react to events as they occur

Traditional Databases (Boring)

D

Qu

es

Data Stream Processing System

result stream

- Indexing?

Overview

Why Stream Processing?

Stream Processing Models

- Streams, windows, operators
- Data mining of streams

Implementation of Stream Processing Systems

- Distributed Stream Processing
- Stream Processing in the Cloud?

Stream Processing

Need to define

1. Data model for streams

2. Processing (query) model for streams

Data Stream

"A **data stream** is a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) **sequence of items**. It is impossible to control the order in which items arrive, nor is it feasible to locally store a stream in its entirety."
[Golab & Ozsu (SIGMOD 2003)]

Relational model for stream structure?

- Can't represent audio/video data
- Can't represent analogue measurements

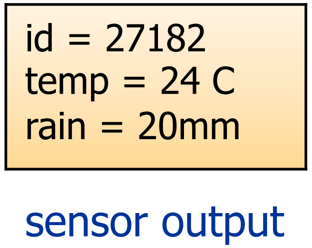
Relational Data Stream Model

Streams consist of infinite sequence of tuples

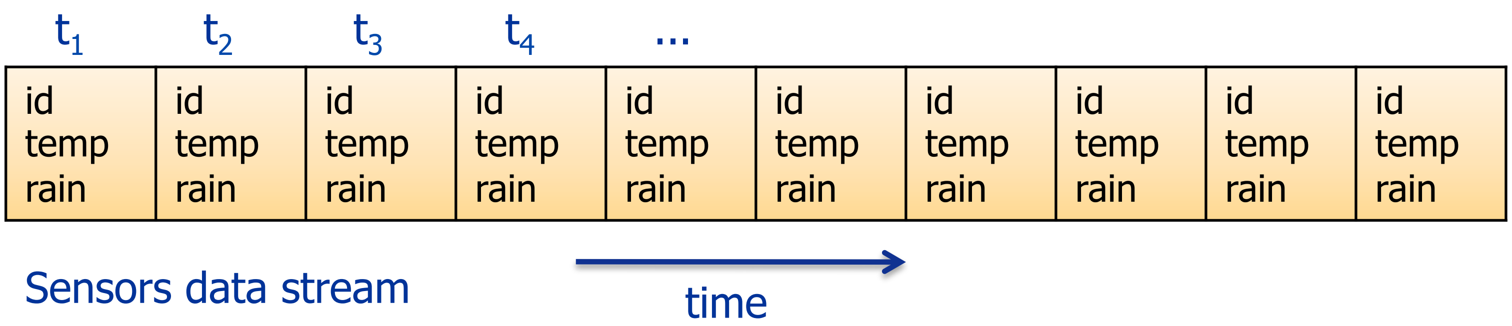
- Tuples often have associated time stamp
  - e.g. arrival time, time of reading, ...

Tuples have fixed relational schema

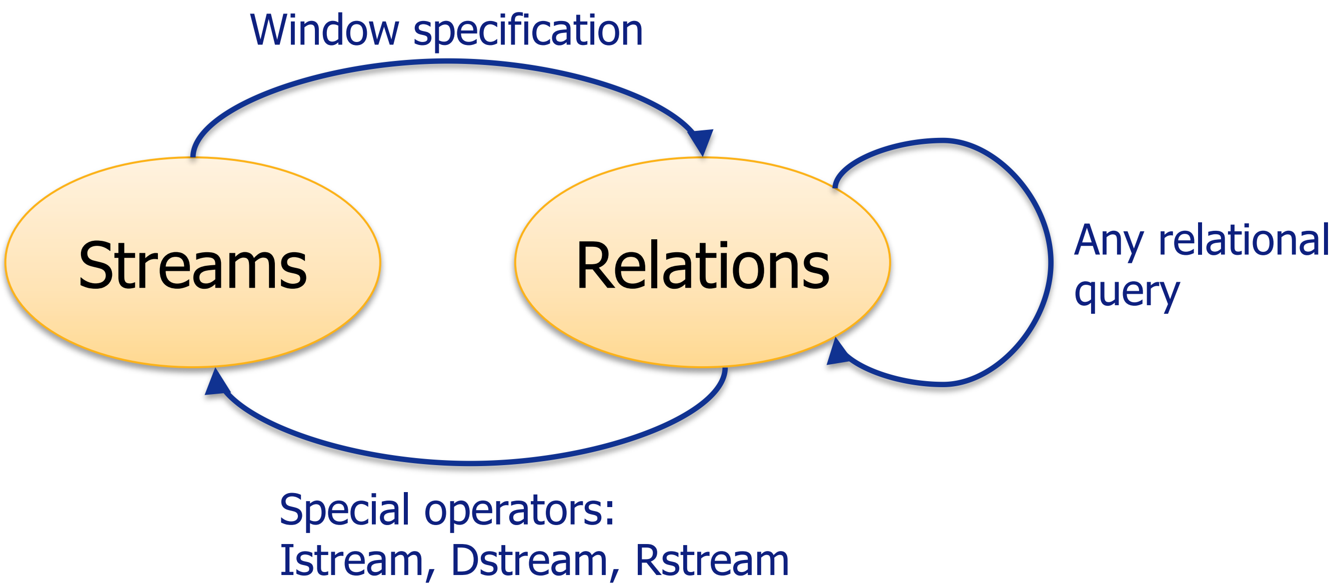
- Set of attributes



`Sensors(id, temp, rain)`



Stream Relational Model



Window converts stream to dynamic relation

- Similar to maintaining view
- Use regular relational algebra operators on tuples
- Can combine streams and relations in single query

Sliding Window I

How many tuples should we process each time?

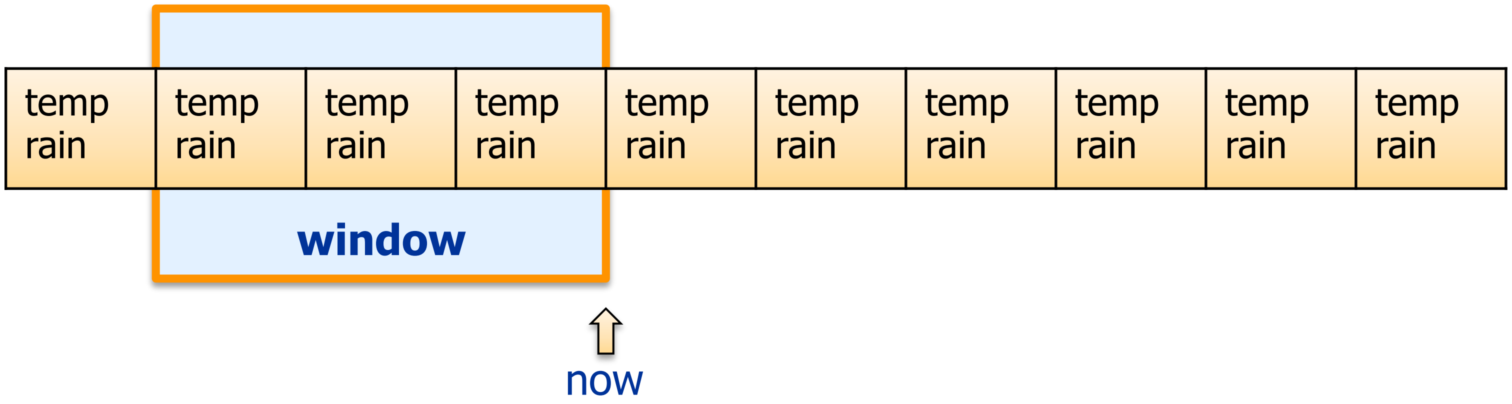
Process tuples in window-sized batches

Time-based window with size τ at current time t

[t - τ : t]	`Sensors [Range τ seconds]`
[t : t]	`Sensors [Now]`

Count-based window with size n:

last n tuples	`Sensors [Rows n]`



Sliding Window II

How often should we evaluate the window?

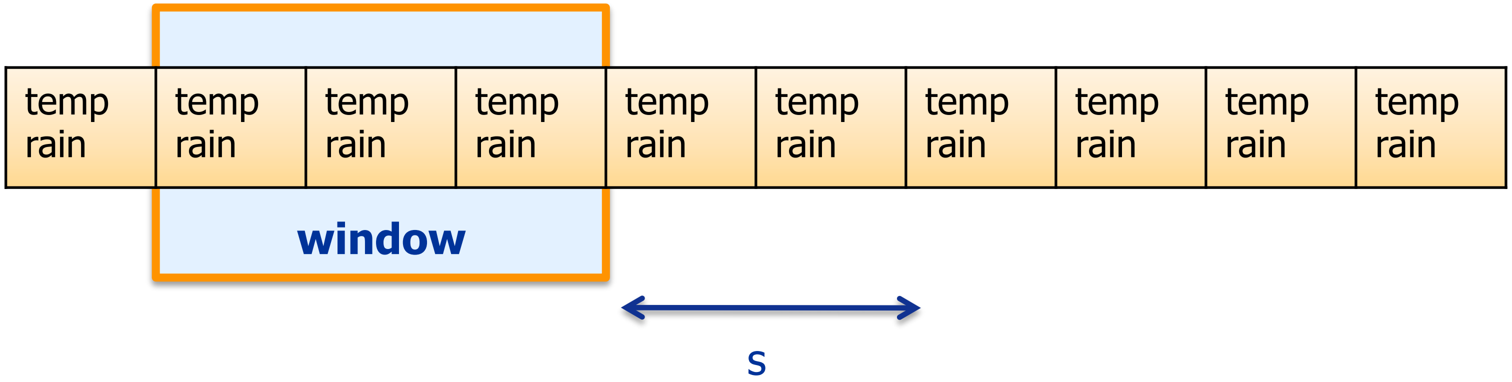
1. Output new result tuples as soon as available
   – Difficult to implement efficiently

2. Slide window by s seconds (or m tuples)

```
Sensors [Slide s seconds]
```

Sliding window: $s < \tau$

Tumbling window: $s = \tau$



Continuous Query Language (CQL)

Based on SQL with streaming constructs

- Tuple- and time-based windows
- Sampling primitives

```
SELECT temp
FROM Sensors [Range 1 hour]
WHERE temp > 42;
```

```
SELECT *
FROM S1 [Rows 1000],
     S2 [Range 2 mins]
WHERE S1.A = S2.A
   AND S1.A > 42;
```

Apart from that regular SQL syntax

Join Processing

Naturally supports joins over windows

```
SELECT *
FROM S1, S2
WHERE S1.a = S2.b;
```

Only meaningful with window specification for streams

- Otherwise requires unbounded state!

```
Sensors(time, id, temp, rain)          Faulty(time, id)
```

```
SELECT S.id, S.rain
FROM Sensors [Rows 10] as S, Faulty [Range 1 day] as F
WHERE S.rain > 10 AND F.id != S.id;
```

Converting Relations → Streams

Define mapping from relation back to stream

- Assumes discrete, monotonically increasing timestamps
  τ, $\tau+1$, $\tau+2$, $\tau+3$, ...

Istream(R)

- Stream of all tuples (r, τ) where $r \in R$ at time τ but $r \notin R$ at time $\tau-1$

Dstream(R)

- Stream of all tuples (r, τ) where $r \in R$ at time $\tau-1$ but $r \notin R$ at time τ

Rstream(R)

- Stream of all tuples (r, τ) where $r \in R$ at time τ

Data Mining in Streams

Stream Data Mining

Often continuous queries relate to long-term characteristics of streams

- Frequency of stock trades, number of invalid sensor readings, ...

May have insufficient memory to evaluate query

- Consider stream with window of 10^9 integers
  - Can store this in 4GB of memory
- What about 10^6 such streams?
  - Cannot keep all windows in memory

☛ Need to compress data in windows

Limitations of Window Compression

Consider window compression for following query:

```
SELECT SUM(num)
FROM Numbers [Rows 10⁹];
```

Assume that W can be compressed as $C(W) = W_C$

- Then $W_1 \neq W_2$ must exist, with $C(W_1) = C(W_2)$
- Let t be oldest time in window for which W1 and W2 differ:

W_1	**3**	5	8	9	2	3	9	7	8	9
W_2	**4**	5	8	2	0	7	0	7	2	1

t

- For W_1: subtract $W_1(t) = 3$; for W_2: subtract $W_2(t) = 4$
  - Cannot distinguish between cases from C(W1) = C(W2)
- No correct compression scheme C(W) possible

Approximate Sum Calculation

Keep sums Σ_i for each n tuples in window

- Compression ratio is 1/n

v_1	v_2	…	v_n	v_{n+1}	v_{n+2}	…	v_{2n}	…	v_{2n+1}	v_{2n+2}

n tuples

n tuples

2 tuples
(incomplete group)

$$\Sigma_W = \boxed{\Sigma_1} + \boxed{\Sigma_2} + \ldots + \boxed{\Sigma_{incomplete}}$$

- Estimate of window sum Σ_W is total of group sums Σ_i

Now v_1 leaves window and v_{2n+3} arrives:

$$\Sigma_W = \boxed{(n\text{-}1/n) * \Sigma_1} + \boxed{\Sigma_2} + \ldots + \boxed{\Sigma_{incomplete}}$$

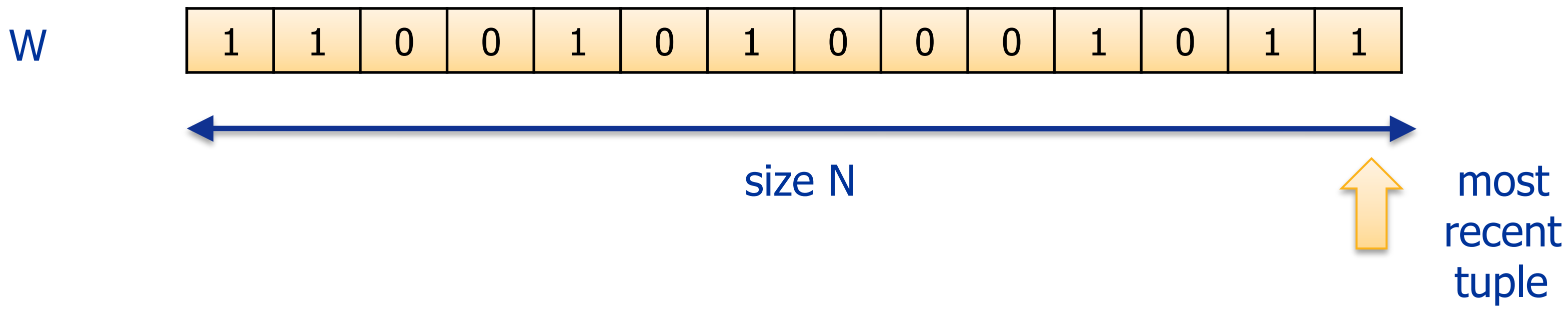
3 tuples
(incomplete group)

- Accuracy of approximation depends on variance

Counting Bits

Assume sliding window W of size N contains bits 1 and 0

– How many 1s are there in the most recent k bits?
  ($1 \leq k \leq N$)



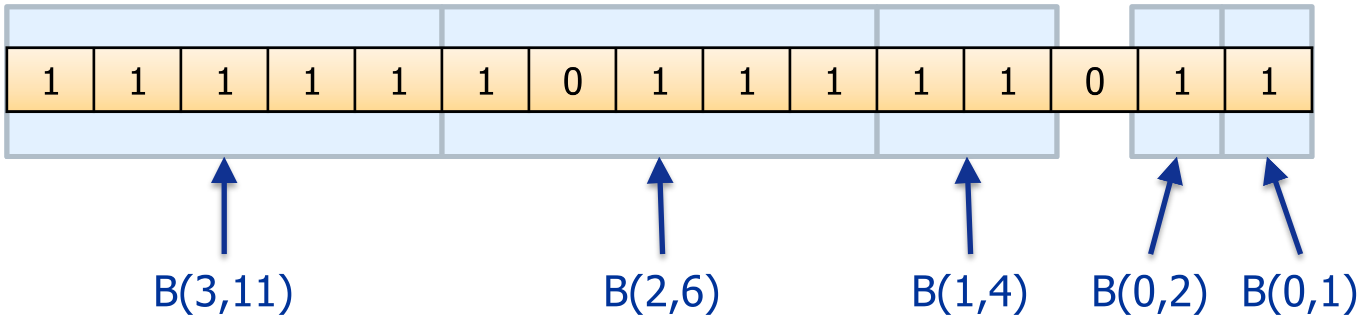
Could answer question trivially with O(N) storage

– But can we approximate answer with, say, logarithmic storage?

Approximate Counting with Buckets

Divide window into multiple buckets B(m, t)

- B(m, t) contains 2^m 1s and starts at t
- Size of buckets does not decrease as t increases
- Either one or two buckets for each size m
- Largest bucket only partially filled



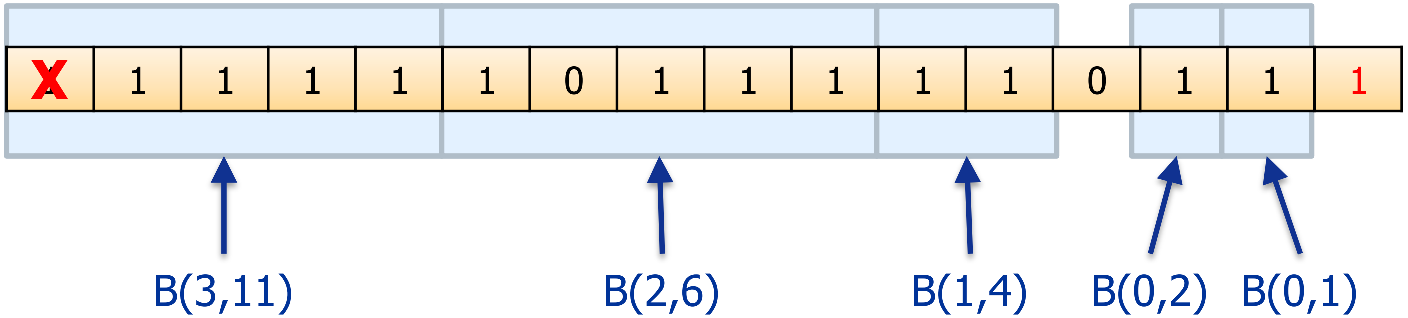
Estimate sum of last k tuples Σ_k:

Σ_k = {sizes of buckets within k} + ½ {last partial bucket}

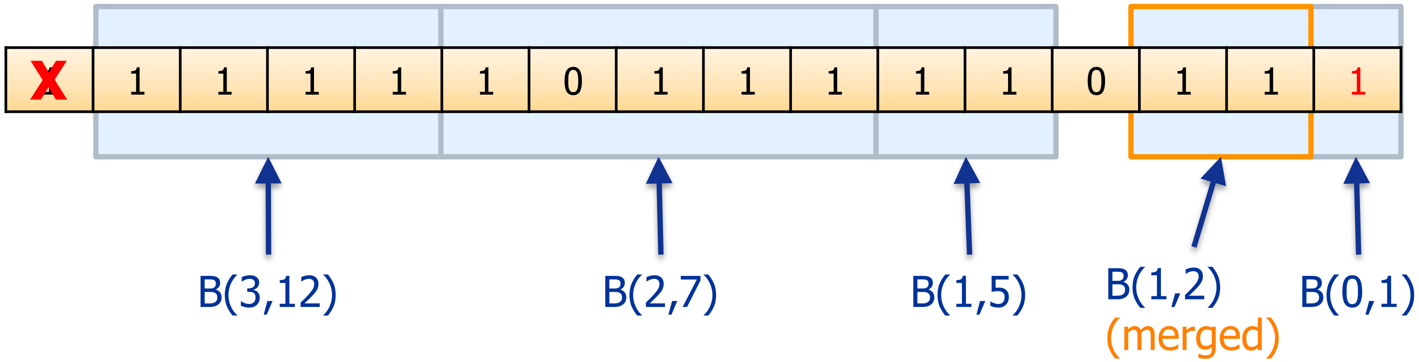
$\Sigma_N = 2^0 + 2^0 + 2^1 + 2^2 + ½ * 2^3 = 12$ (exact answer: 13)

Maintaining Buckets

Discard/merge buckets as window slides



- Discard largest bucket once outside of window
- Create new bucket B(0,1) for new tuple if 1
- Merge buckets to restore invariant of at most 2 buckets of each size m



Space Complexity

Need $O(\log N)$ buckets for window of size N

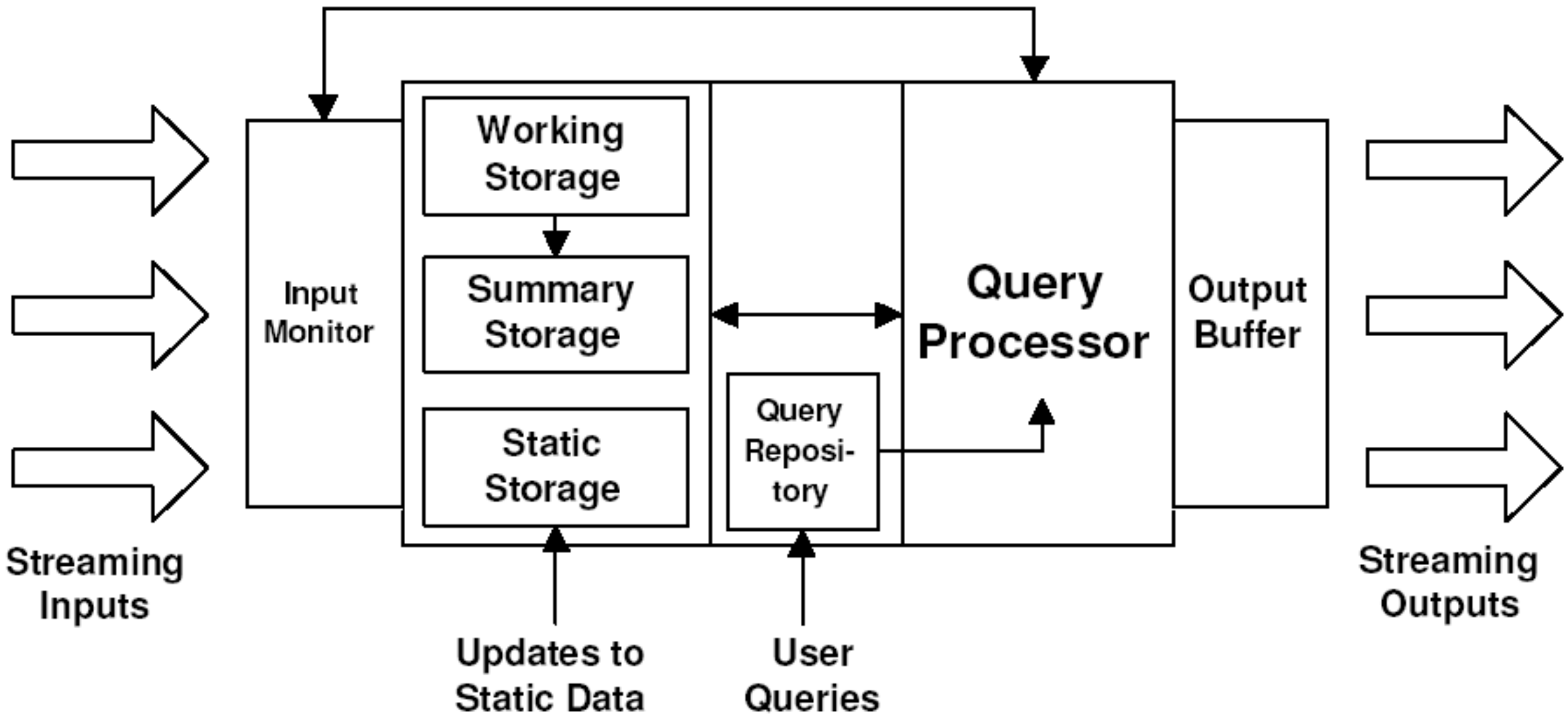
Need $O(\log N)$ bits to represent bucket B(m, t):

- **m** is power of 2, so representable as $\log_2 m$
  m can be represented with $O(\log \log N)$ bits
- t is representable as t mod N
  t can be represented with $O(\log N)$ bits

Overall window compressed to $\mathbf{O(\log^2 N)}$ bits

DSPS Implementation

General DSPS Architecture



Source: Golab & Ozsu 2003

Stream Query Execution

Continuous queries are long-running
➔ properties of base streams may change

- Tuple distribution, arrival characteristics, query load, available CPU, memory and disk resources, system conditions, ...

Solution: Use **adaptive query plans**

- Monitor system conditions
- Re-optimise query plans at run-time

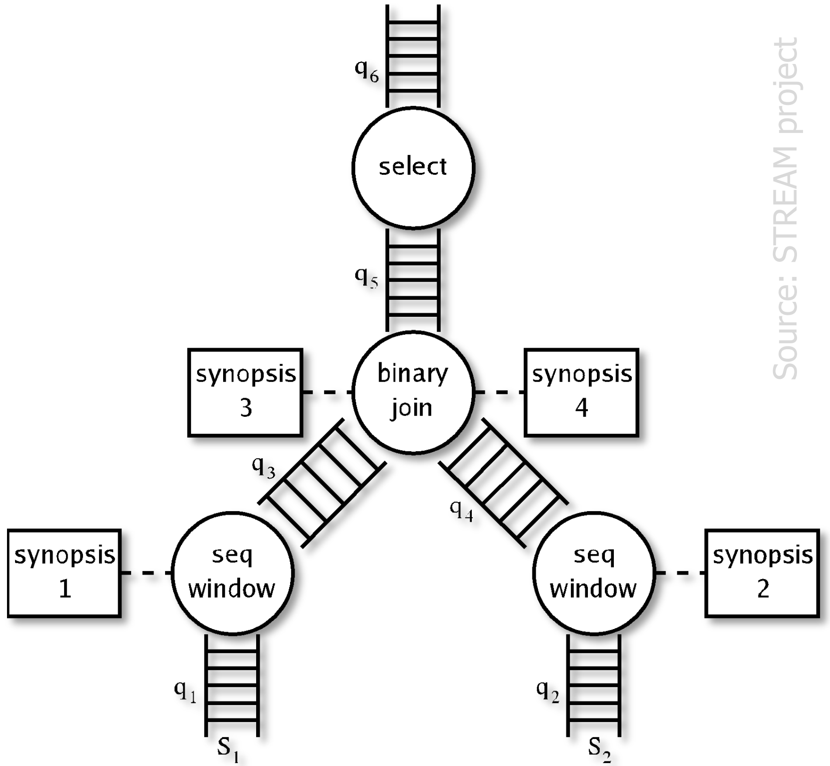
DBMS didn’t quite have this problem...

Query Plan Execution

Executed query plans include:

- **Operators**
- **Queues** between operators
- **State**/"Synposis" (windows, ...)
- **Base streams**

```
SELECT *
FROM S1 [Rows 1000],
     S2 [Range 2 mins]
WHERE S1.A = S2.A
   AND S1.A > 42;
```



Source: STREAM project

Challenges

- State may get large (e.g. large windows)

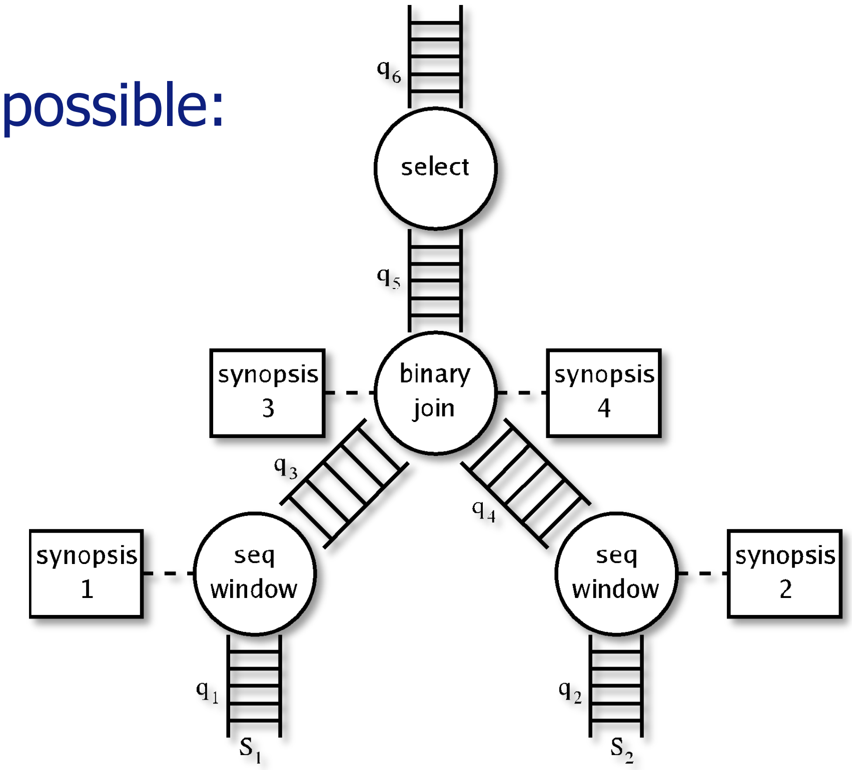
Operator Scheduling

Need scheduler to invoke operators (for time slice)

– Scheduling must be adaptive

Different scheduling disciplines possible:

1. Round-robin
2. Minimise queue length
3. Minimise tuple delay
4. Combination of the above



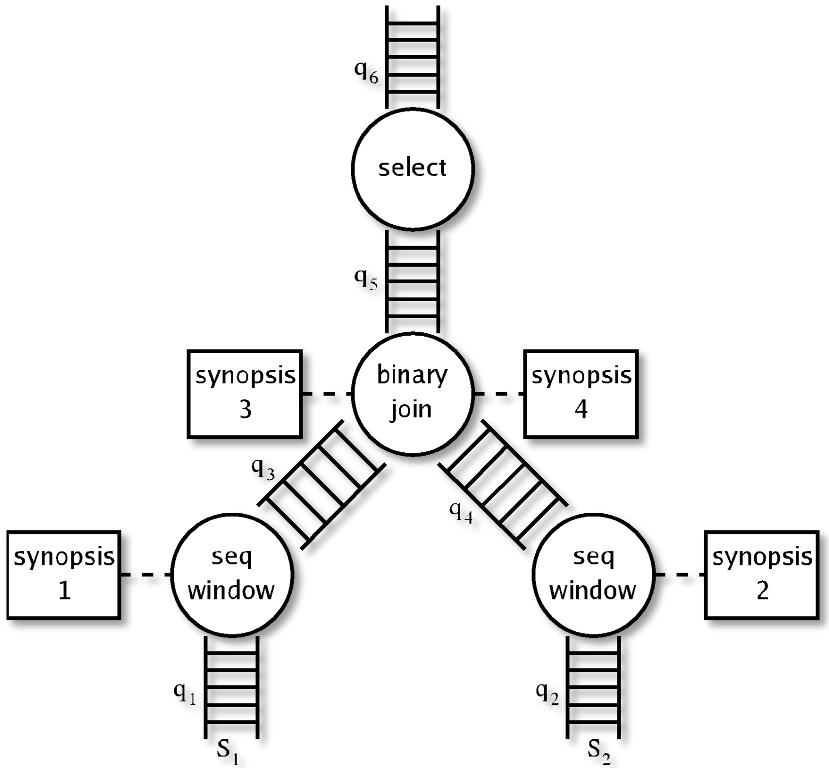
Load Shedding

DSMS must handle overload:
Tuples arrive faster than processing rate

Two options when overloaded:

1. **Load shedding**: Drop tuples
   - Much research on deciding which tuples to drop: c.f. result correctness and resource relief
   - e.g. sample tuples from stream

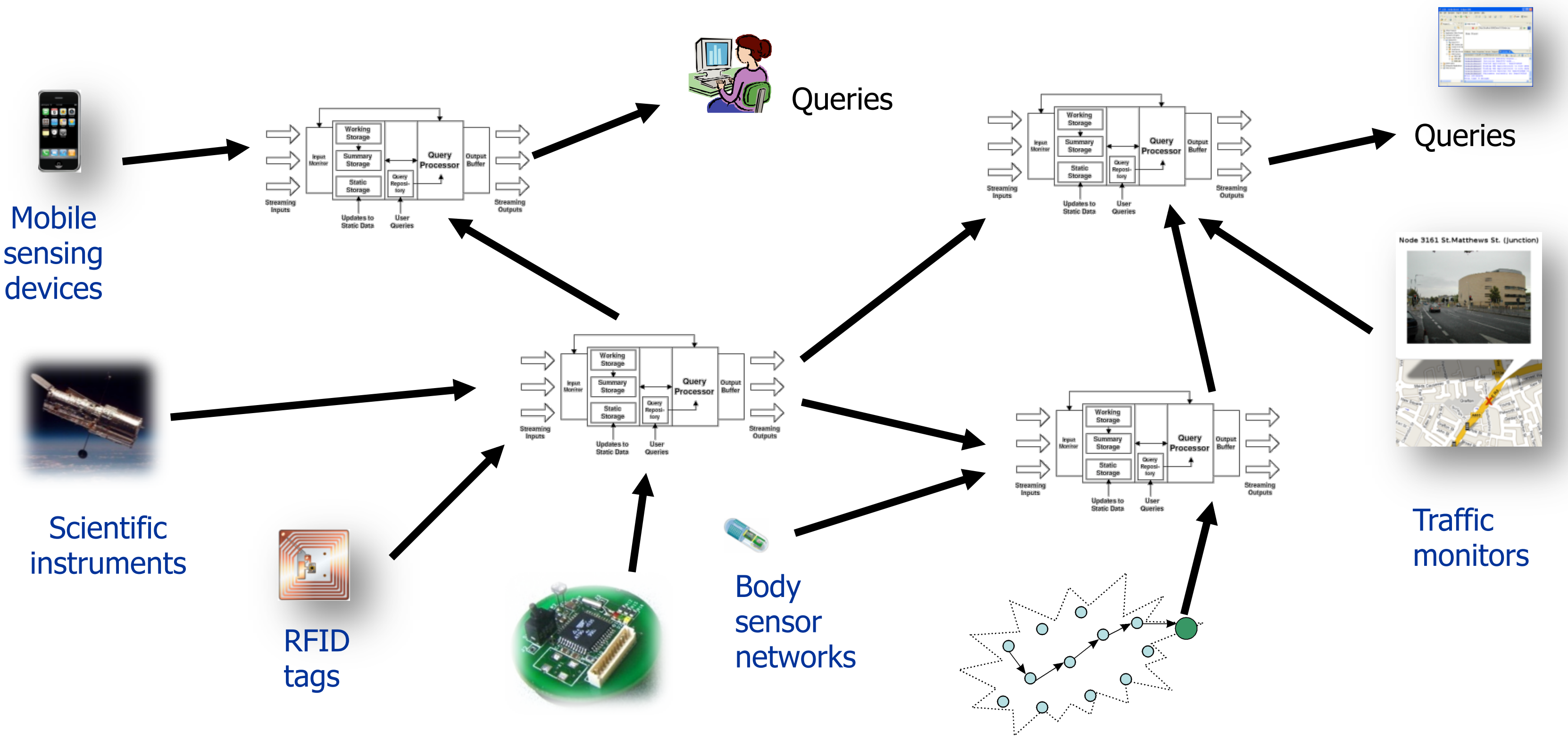
2. **Approximate processing**: Replace operators with approximate processing
   - Saves resources



Distributed DSPS

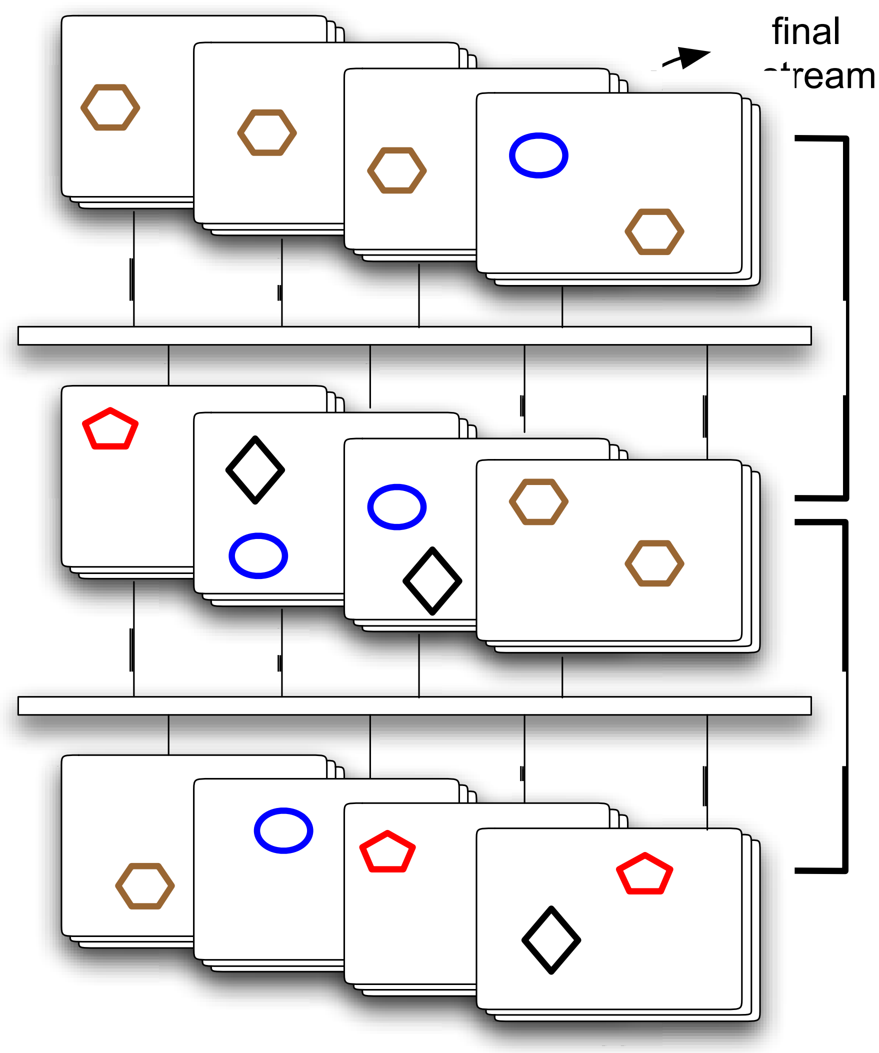
Distributed DSPS

- Interconnect multiple DSPSs with network
  - Better scalability, handles geographically distributed stream sources



- Interconnect on LAN or Internet?
  - Different assumptions about <u>time</u> and <u>failure</u> models

Query Planning in DSPS



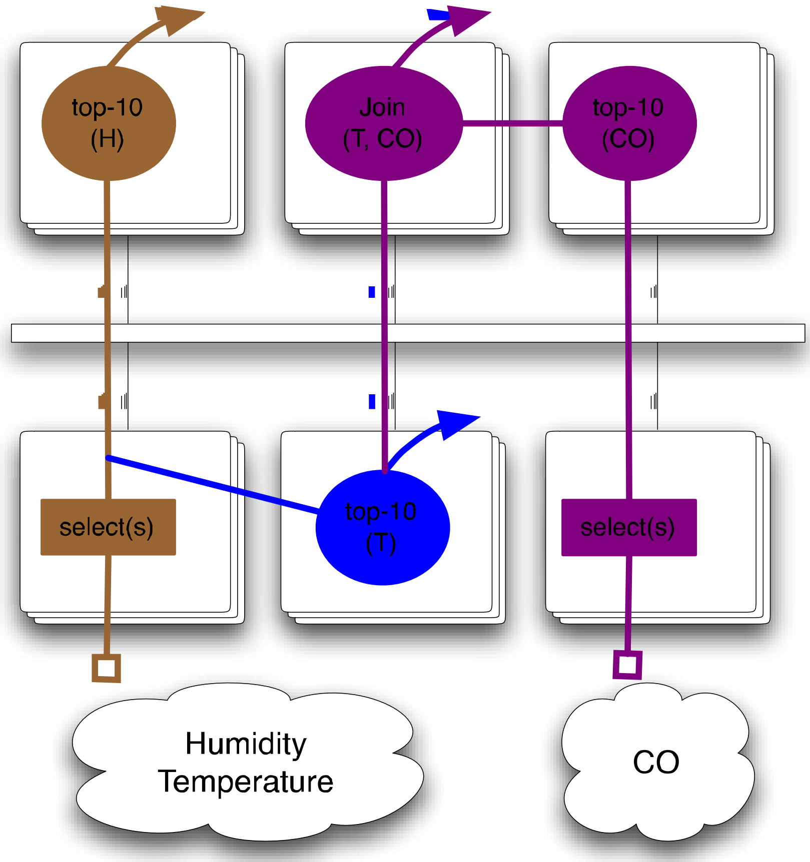
Query Plan

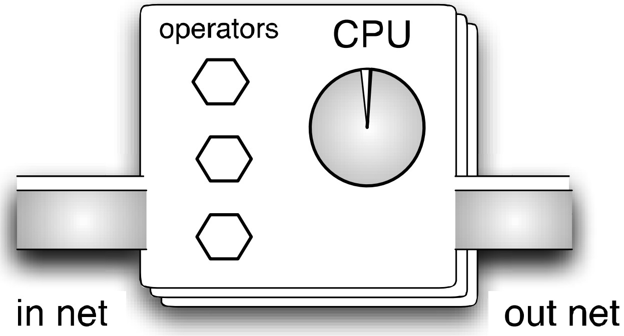
- Operator placement
- Stream connections
- Resource allocation: CPU, network bandwidth, ...

State-of-the-art planners

- Based on heuristics (eg IBM's SODA)
- Assume over-provisioned system
  - Simplifies query planning
  - Not true when you pay for resources...

Planning Challenges





Waste of resources due to query overlap ➔ reuse streams

Premature exhaustion of resources ➔ multi-resource constraints

Optimisation Model

Unified optimisation problem for

- query admission
- operator allocation
- stream reuse

maximise:

λ_1 * (no of satisfied queries) – λ_2 * (CPU usage) – λ_3 * (net usage) – λ_4 * (balance load)

subject to constraints:

1. availability: streams for operators exist on nodes
2. resource: allocations within resource limits
3. demand: final query streams are generated eventually
4. acyclicity: all streams come from real sources

This is hard!

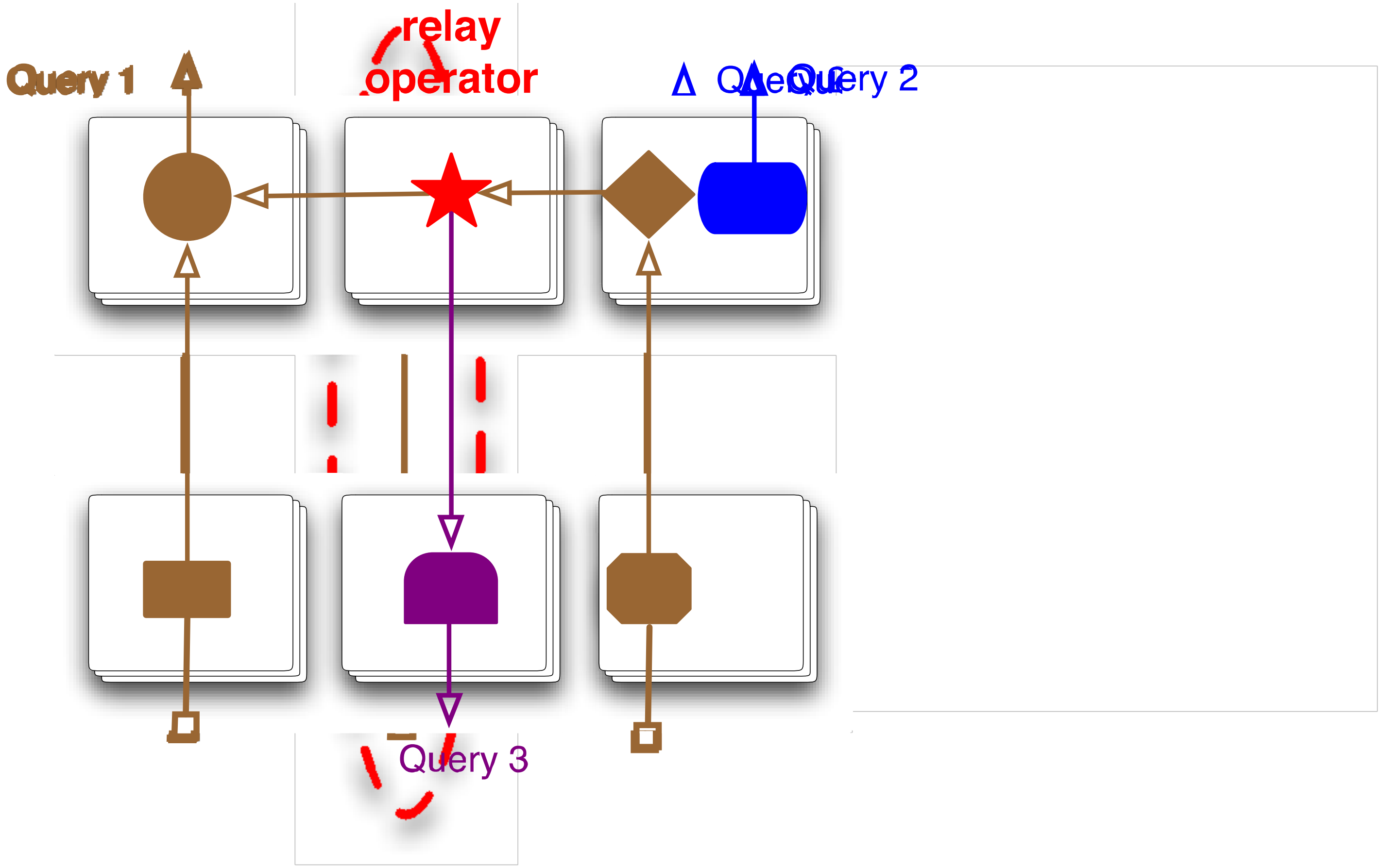
- Solve approximate problem to obtain tractable solution

Evangelia Kalyvianaki, Wolfram Wiesemann, Quang Hieu Vu and Peter Pietzuch, "SQPR: Stream Query Planning with Reuse", IEEE International Conference on Data Engineering (ICDE), Hannover, Germany, April 2011

Tractable Optimisation Model

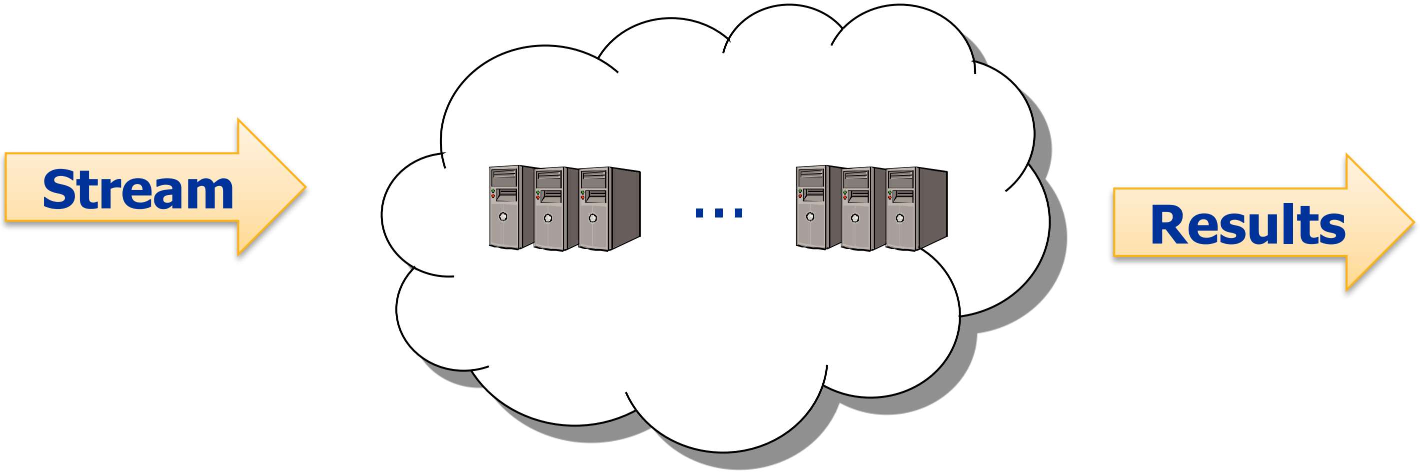
Idea: Only optimise over streams related to new query

- Add relay operators to work around constraints under reuse



Stream Processing in the Cloud

Stream Processing in the Cloud



n servers in cloud DC

Scalability: Scale horizontally across 1000 VMs to support

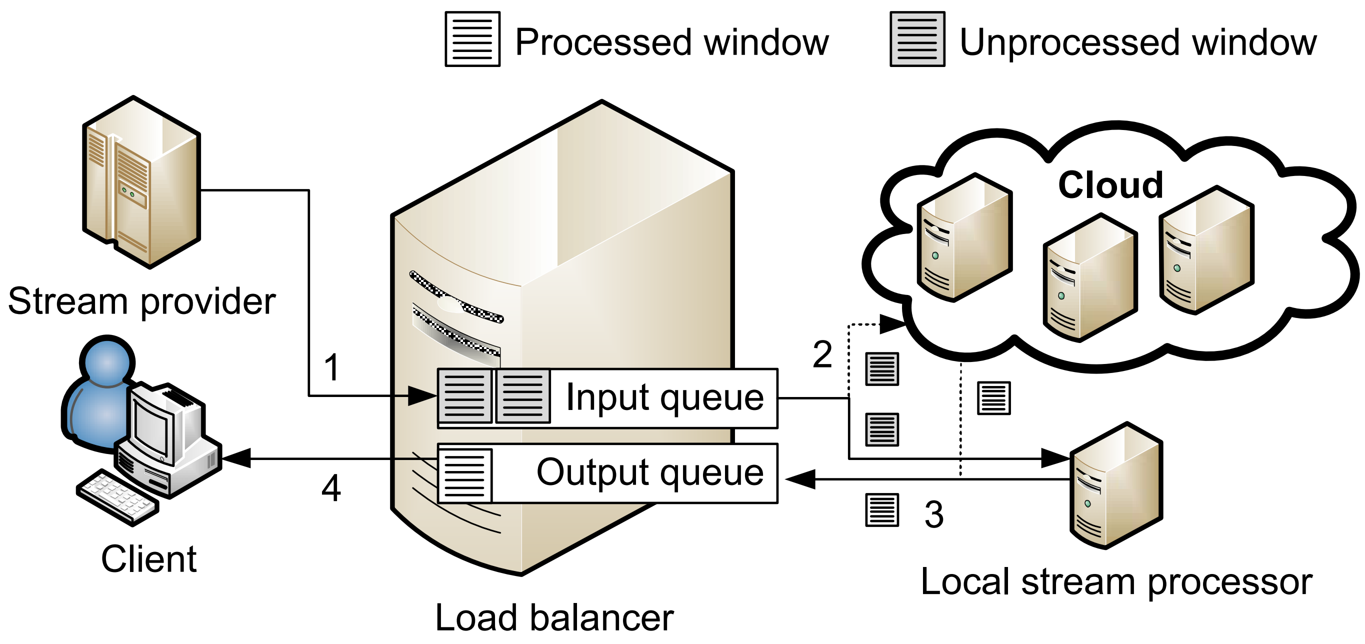
- larger number of queries
- high stream rates

Elasticity: Dynamically tune number of processing servers

- Tune n to affect stream processing throughput

Load Balancing with the Cloud

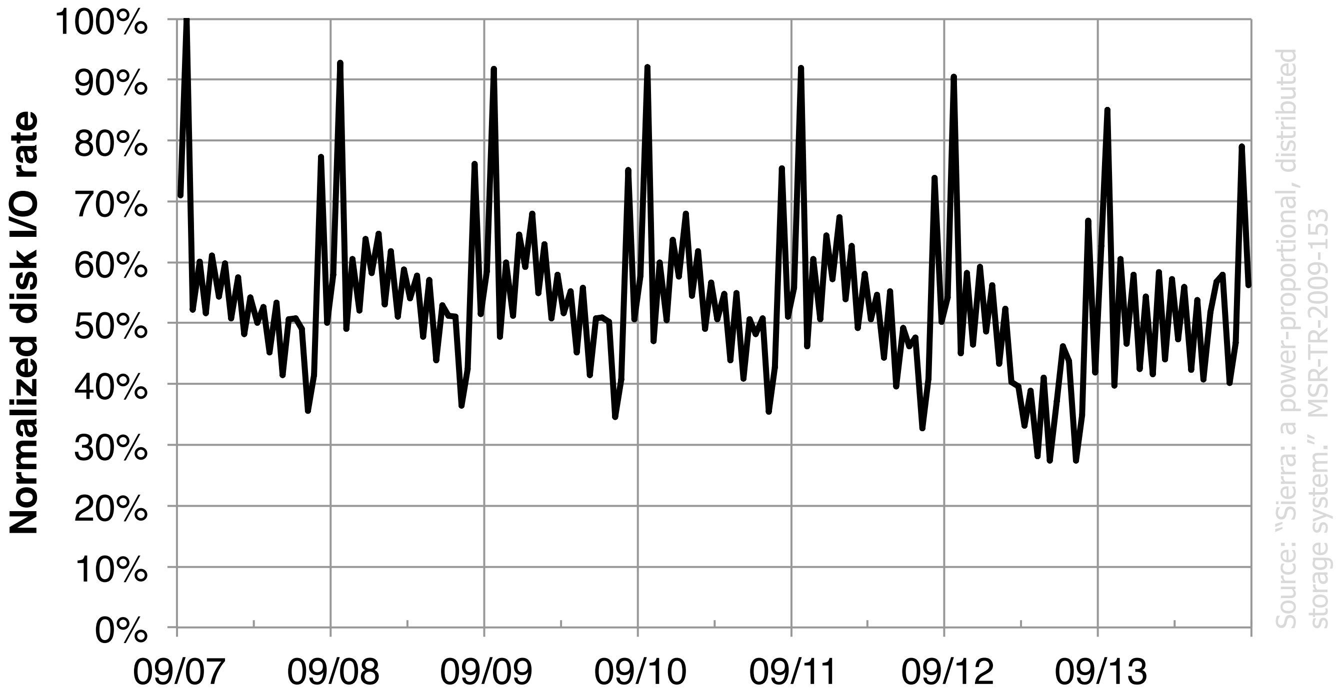
Idea: Using cloud resources for handling peak processing demand



- Network latency to cloud major issue
- Partitioning granularity important

☛ How do you perform stream processing in the cloud?

Typical Processing Workload



Existing workloads have peaks and troughs

– Scope for improvement in terms of **elasticity and adaptability**

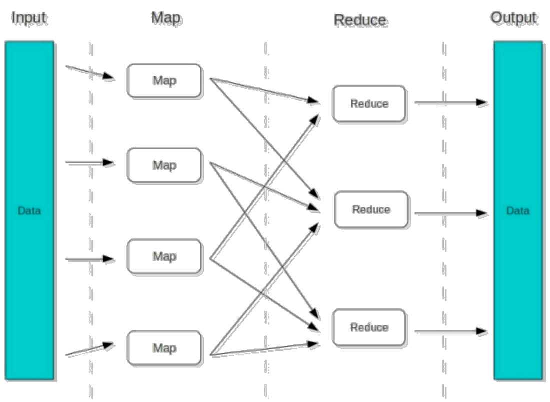
Current solutions in distributed stream processing

– Over-provisioning to handle peak demand
– Load-shedding to discard data during peaks

The Map/Reduce Hammer?

Strawman idea:

- Adapt batch processing model
- Pipelined implementation of map/reduce



Partitioning granularity?

- Window = job?
- Apache Hadoop has large per job overhead

Stream processing semantics?

Data exchange based on distributed file system

Two Layers: Dispatching and Processing

Structured architecture for stream processing

- Separates stream partitioning from computation
- Partitioning reduces amount of data for computation

Simple function in each operators:

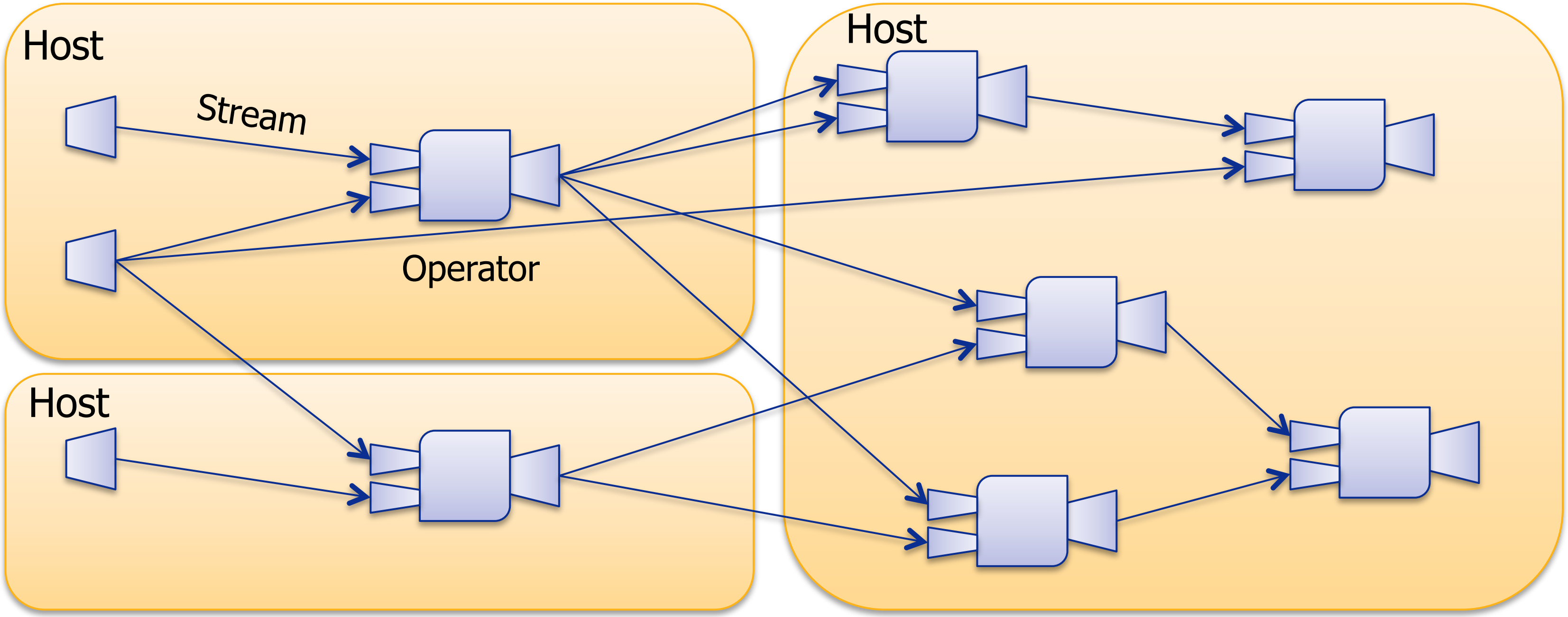
1. Stream partitioning performed by **dispatching layer**
   - Identify relevant data for queries
   - Partitioning of data streams and multicast to multiple operators

2. Computation done by **processing layer**
   - Execution of query operators

SEEP: Scalable & Elastic Event Processing

Decompose queries into multiple stream processing operators

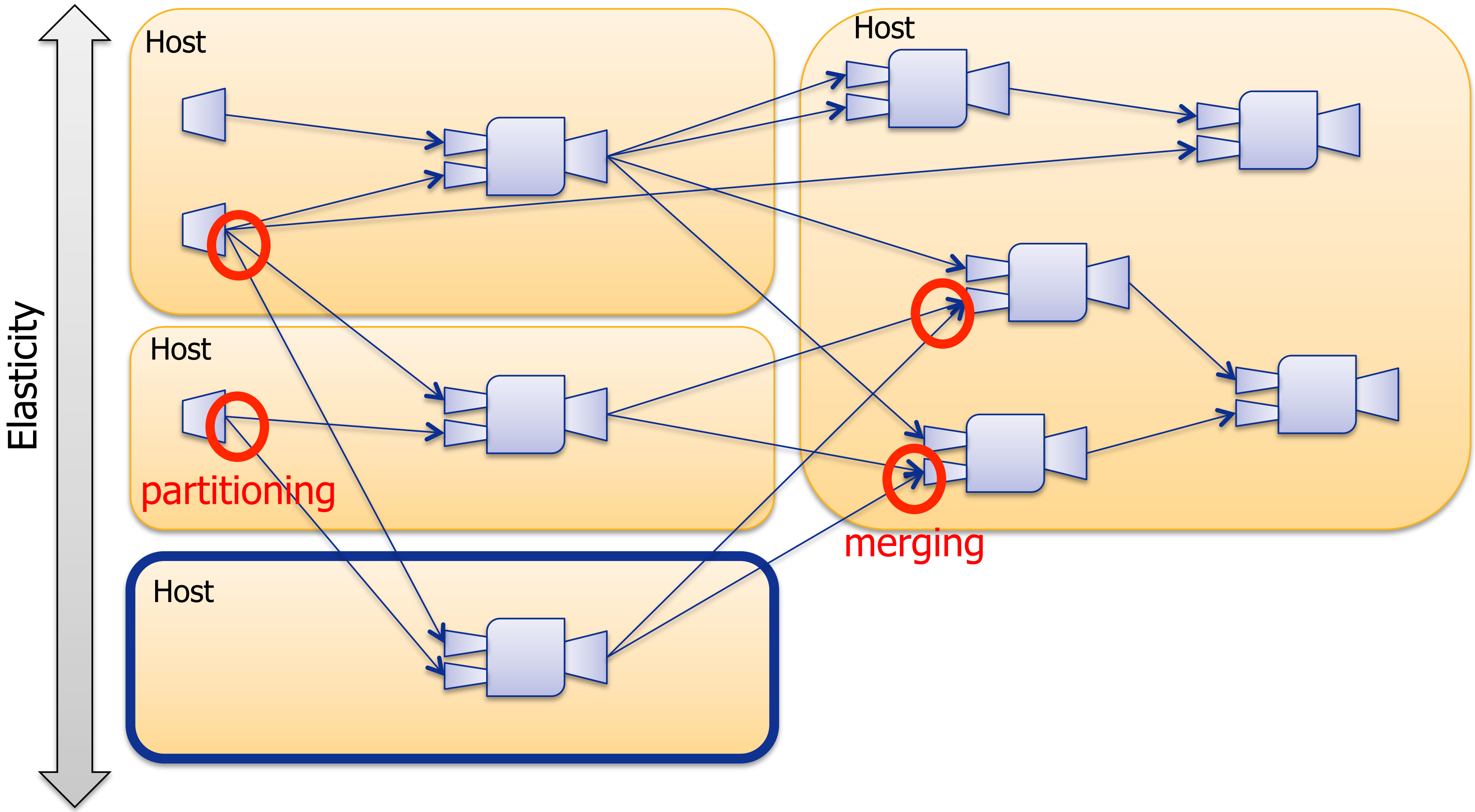
- System exploits intra-query parallelism



Adapt to variations in workload by scaling out

SEEP: Scalable & Elastic Event Processing

Partition and merge streams to utilise more hosts



Twitter Storm & Yahoo S4

Yahoo! S4 (http://incubator.apache.org/s4/)

- Java framework for implementing stream processing applications
- Hides stream "plumbing" from developers
- Uses Zookeeper for coordination

Twitter Storm (https://github.com/nathanmarz/storm)

- Focus on fault-tolerance: acknowledgement of processed tuples
- Spouts produce data; bolts process data
- Different mechanisms for stream partitioning and bolt parallelisation

This is just the beginning... lots of open challenges...

Conclusions

Stream processing will grow in importance
- Handling the data deluge
- Just provide a view/window on subset of data
- Enables real-time response and decision making

Principled models to express stream processing semantics
- Enables automatic optimisation of queries, e.g. finding parallelism
- What is the right model?

Resource allocation matters due to long running queries
- High stream rates and many queries require scalable systems
- Handling overload becomes crucial requirement
- Volatile workloads benefit from elastic DSPS in cloud environments

Thank You! Any Questions?

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