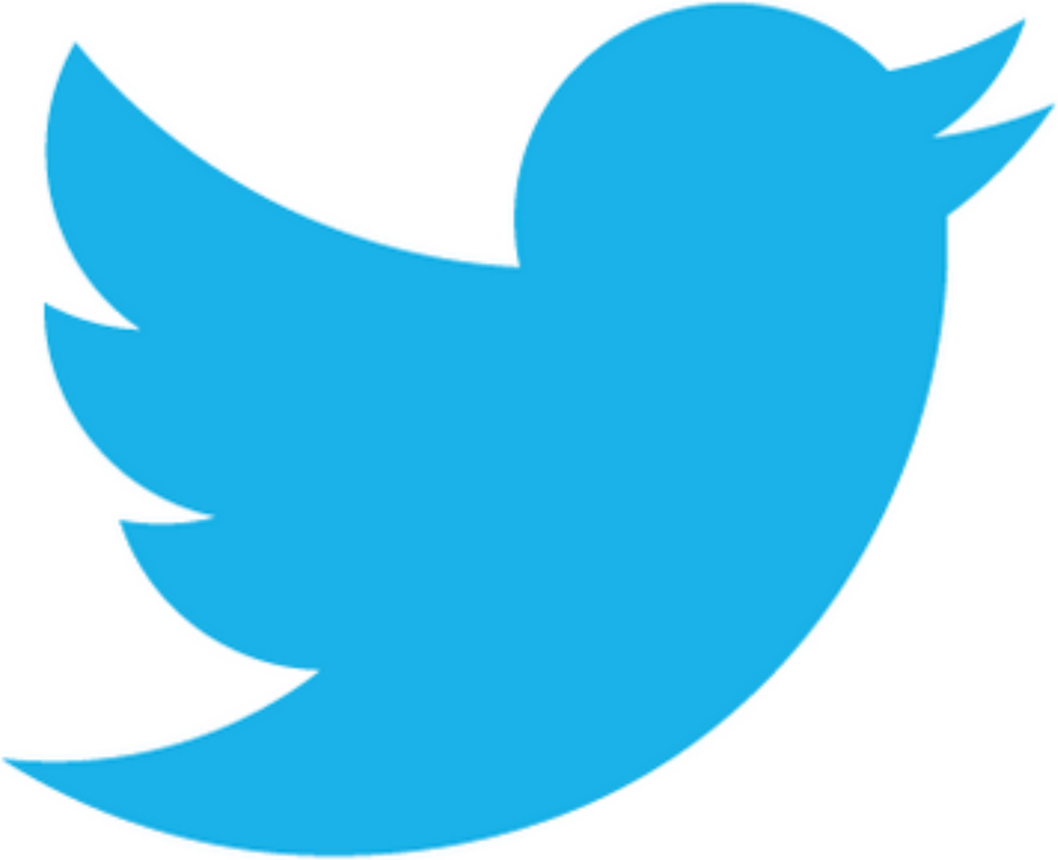
Data Stream Management Systems: An introduction

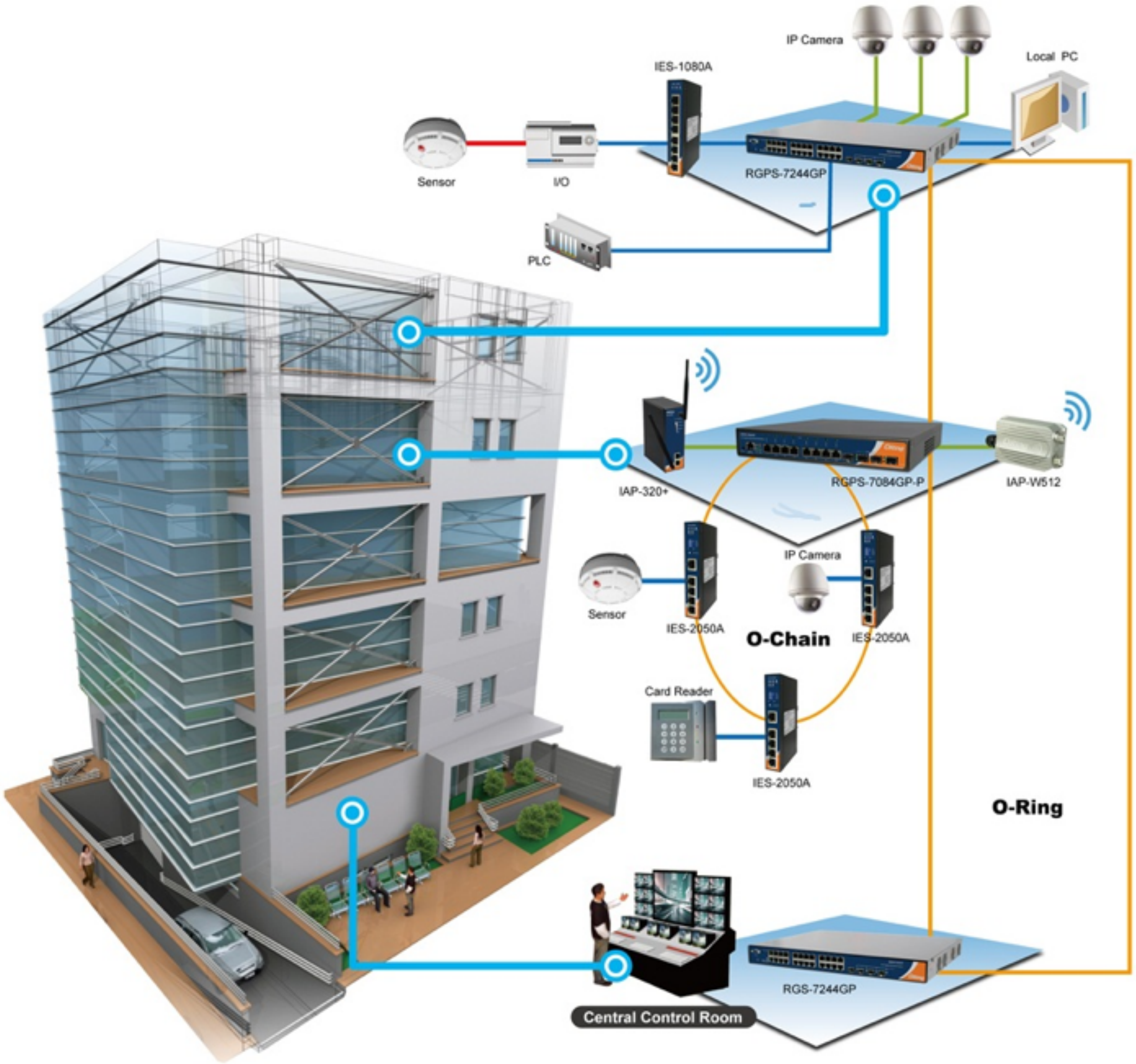
DATA STORM Big Data Summer School
Lisbon, July 14-16th

Paulo Carreira | IST & INESC-ID

Social Web



Smart Buildings



Smart Cities

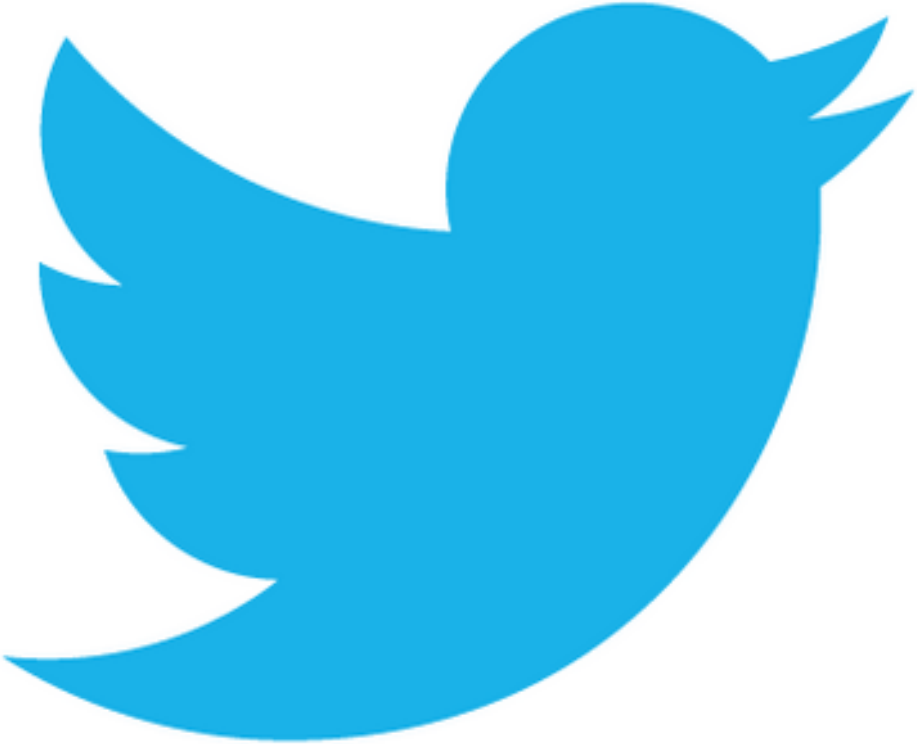
Smart Meters

Smart Everything

Streaming data

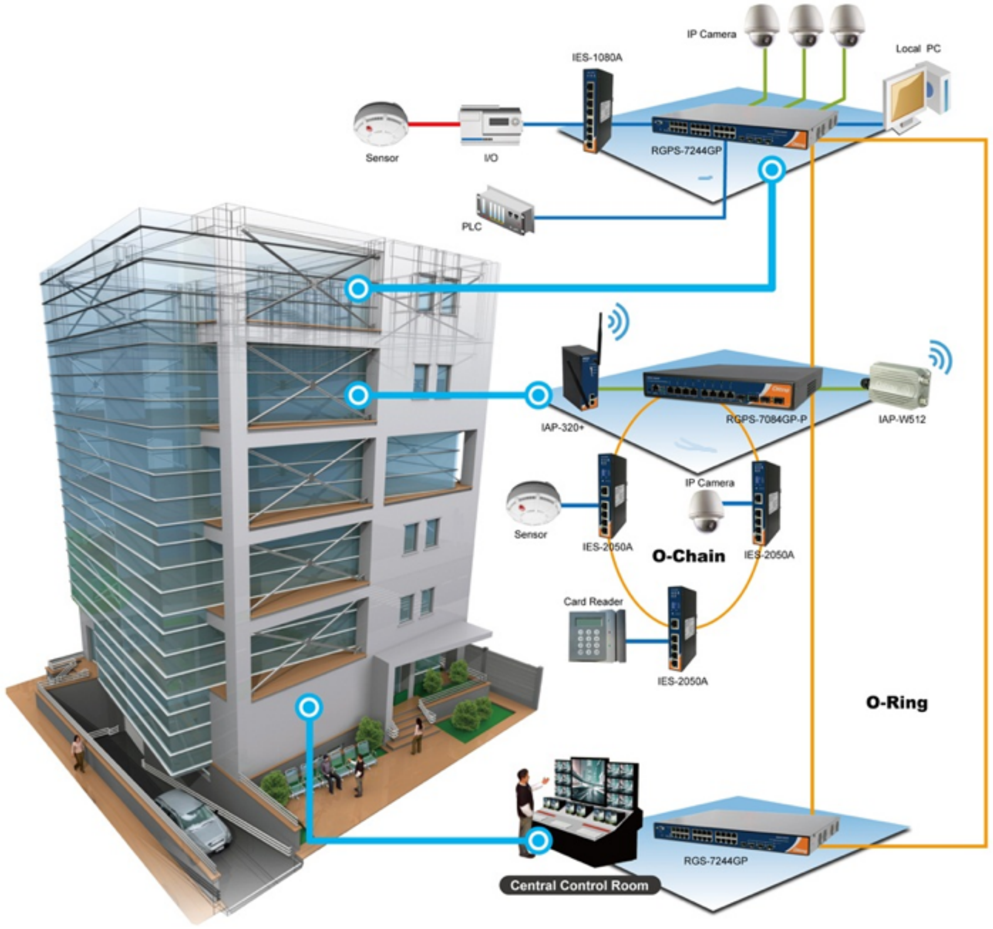
- Arrives out of order
- At varying frequency (sometime bursty)
- Continuously arriving (infinite in nature)
- Transient

Social Web



"Which countries re-tweeted my messages the most over the last 8 hours?"

Smart Buildings



"Which rooms are consuming more energy?"

Smart Cities

“Which neighborhoods have lamps did not start?”

Smart Meters

“Which appliances are consuming most energy?”

Continuous Queries

- Continually evaluated whenever new information arrives

- Continually producing updates to their results

Streaming Data Applications

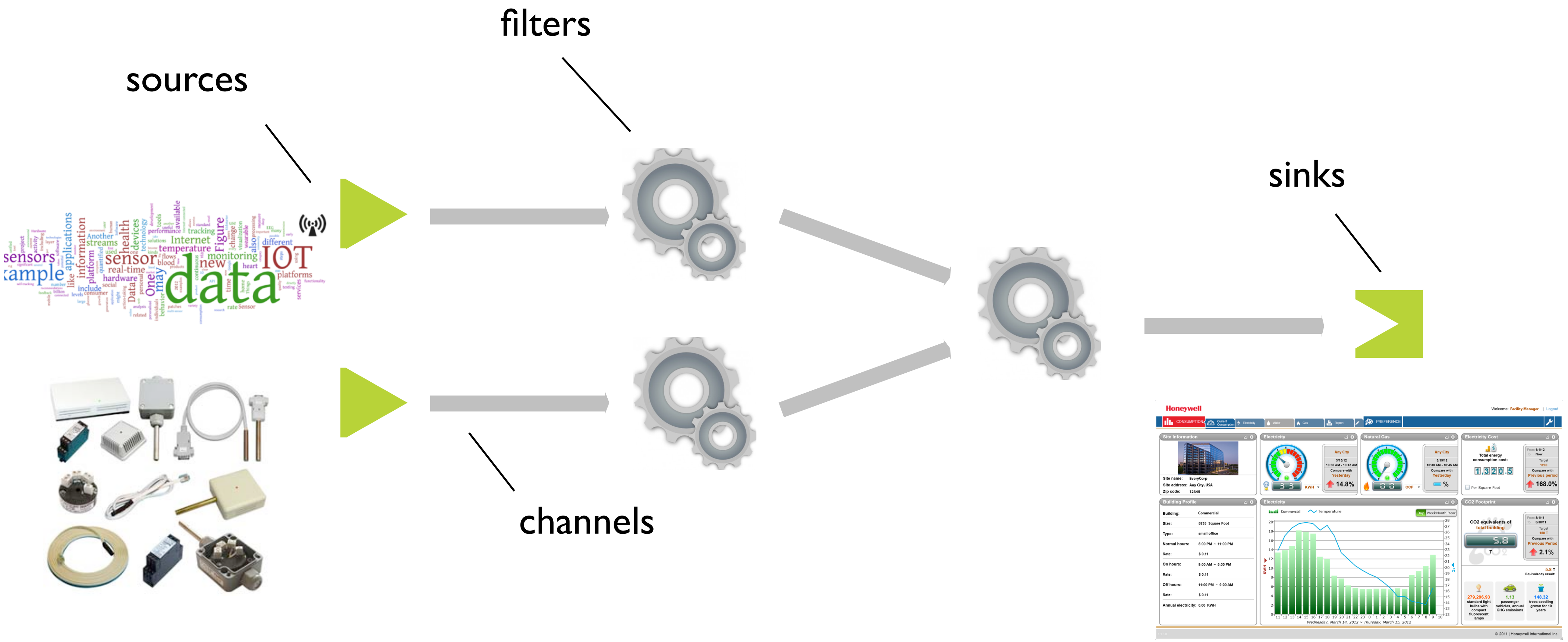
- Finance (real-time trading decisions)
- Fraud detection
- Business Intelligence
- Building Automation
- Assisted Living
- Monitoring
- …

Outline

1. Motivation
2. Stream Processing Fundamentals
3. Querying Data Streams (by example)
    - Output control
    - Windowing
    - Blocking Operators
    - Insert and Remove Streams

Stream Processing Fundamentals

Stream Processing Systems



sources

filters

sinks

channels

Several Challenges

- Processing many events per second (thousands to millions)
- Running thousands of queries
- Detecting very complex event conditions
- Minimization of the development effort (to setup a system that reacts to complex situations)
- …

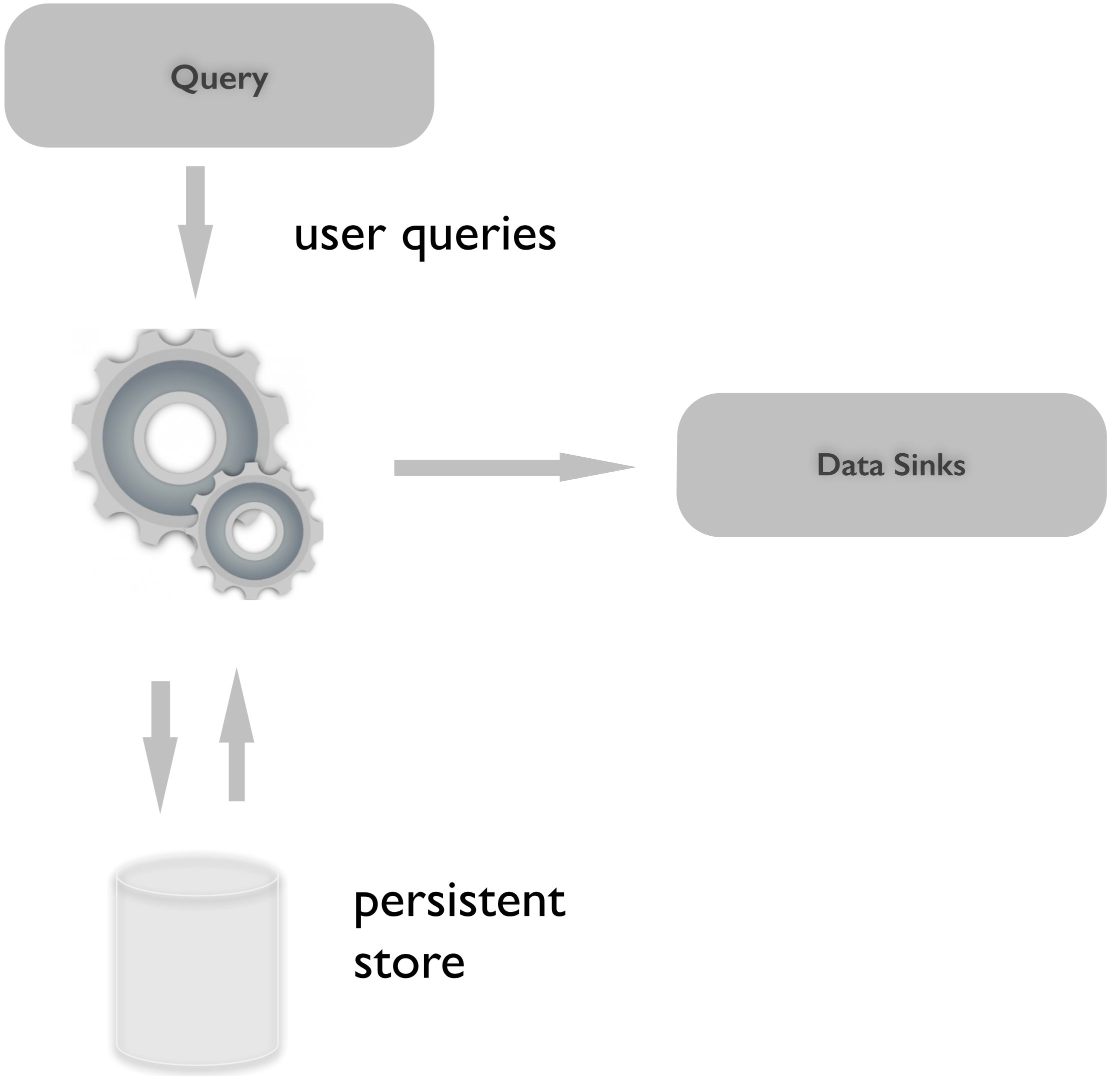
Engineering Stream Processing Software

- Currently Stream Processing software is (still) developed ad-hoc
- In the 1970s Database Management Systems detached applications from data storage and processing logic
- By the 2000s Data Steam Management Systems promise to **detach applications from streaming data processing logic**

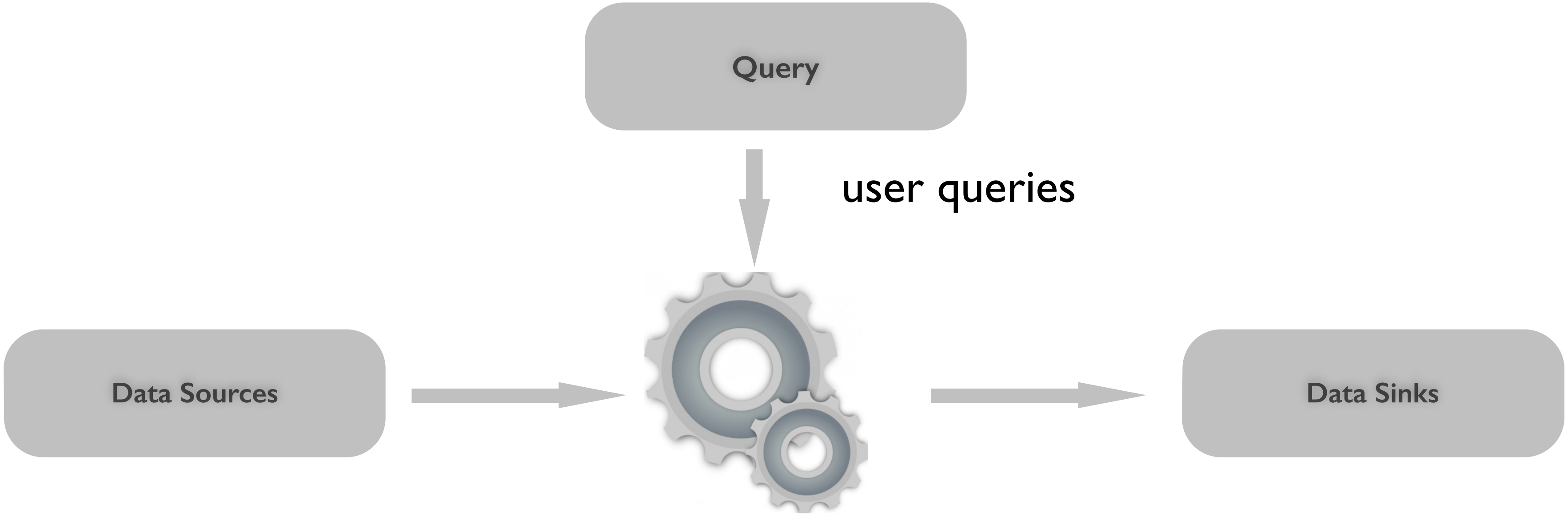
Database Management Systems

Data is stored;

Queries are run



Data Stream Management Systems



Queries are stored;

Data is run

Available DSMSs

- **Research Systems**
  - TelegraphCQ, COUGAR, Borealis, STREAMON
- **Commercial**
  - Oracle 11g RT
  - (IBM DB2, Microsoft SQL-Server soon to follow)
- **Open Source**
  - Esper, Twitter Storm, Apache S4, ...

DBMS vs DSMS

- **DBMS**
  - Queries are run when submitted and terminate
  - Data is pulled
  - Optimization occurs upfront (minimize IO effort)
- **DSMS**
  - Queries are installed and run continuously
  - Data is pushed
  - Optimization is dynamic (minimize latency)

Why are DSMS so great?

- **Simplify the design** of real-time data processing applications

- **Very efficient** at processing a large number of queries over high data flow rates
  - >500K events/sec on dual CPU machine
  - 500 queries at an arrival rate of 1000 events/sec (?)

Challenges

- **Unbounded Memory:** Queries require an unbounded amount of memory to evaluate precisely.
  - Approximate query processing
  - Sliding window query processing
- **High data-flow rate**: Data arrives at a pace (multi-GB) that floods the CPU
  - Sampling
  - Data synopsis

Querying Data Streams (by example)

Events and Streams

- **Event**
  - An occurrence within a particular context
  - Refers to the real-world event and to its digital representation (data)
- **Stream**
  - Abstracted as an **append-only relation** with transient tuples
  - Events on a given stream have **similar structure** known upfront

Continuous Queries

- **Non-blocking** Relational operators extend naturally to stream processing
  - Select, Project, Join are Relational
  - Therefore, SQL also extends naturally to stream processing

- **Blocking operators** such as Grouping and Aggregation require specific operators to be introduced

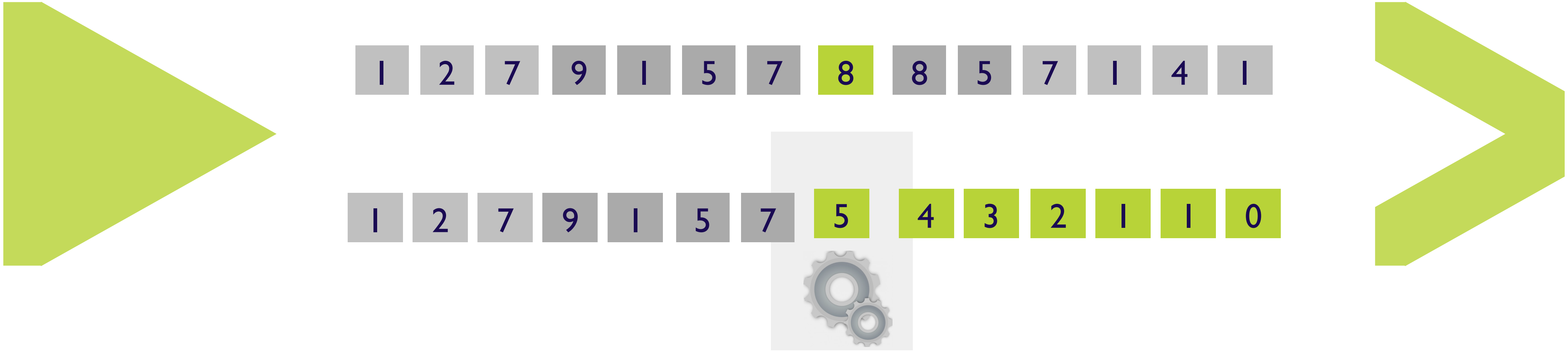
Basic Continuous Query Block

```
select col_expr1, ..., col_exprn
from stream_def
where select_cond
group by aggr_expr
having having_cond
order by ordering_expr
output output_expr
```

The ‘Hello World’ example

The count of withdrawals of amounts greater than 3

```
select count(amount)
from Withdrawal
where amount > 3
```

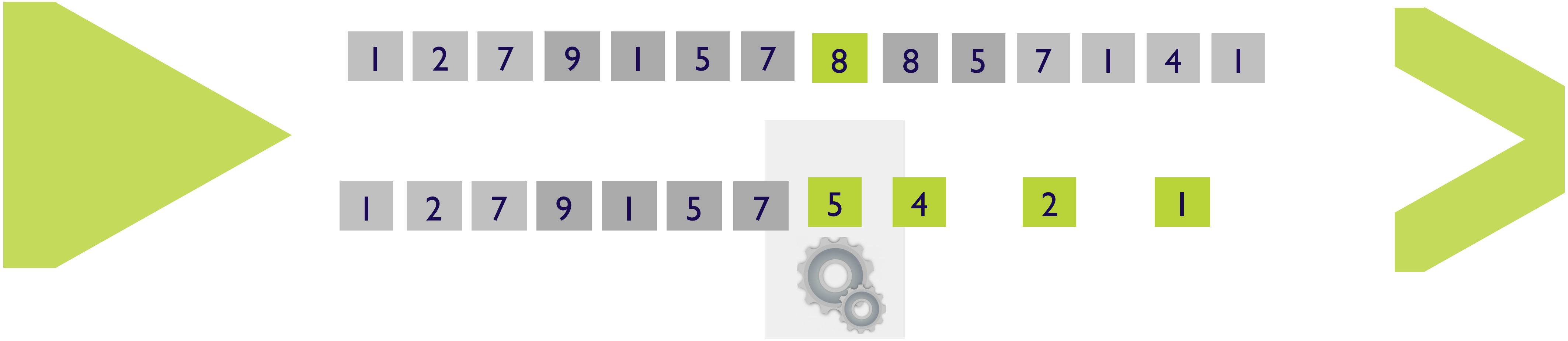


Output control

Output Control

The count of withdrawals of amounts greater than 3, reported at every two events

```
select count(amount)
from Withdrawal
where amount > 3
output every 2 events
```



Output Control

- The **output** clause specifies, *when*, *how* and at what *rate* the output is produced

```
output [after n [seconds | events]]
    [[all | first | last | snapshot]
every output_rate [seconds | events]]
```

Output control

Output at a fixed rate

The ids of all temperature sensor events every 10 seconds

```
select sensorId
from TemperatureSensorEvent
output all every 10 seconds
```

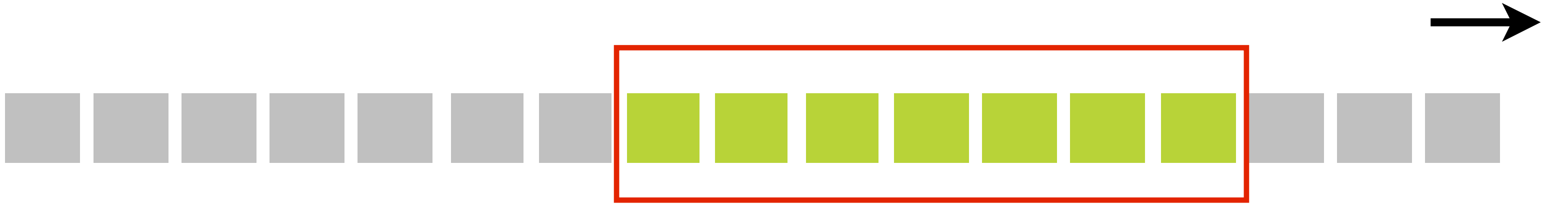
Partial output at fixed rate

The first of a series temperature drops below 21C a the specified output rate

```
select *
from TemperatureSensor where temp<21
output first every 60 seconds
```

Input Control

Windows

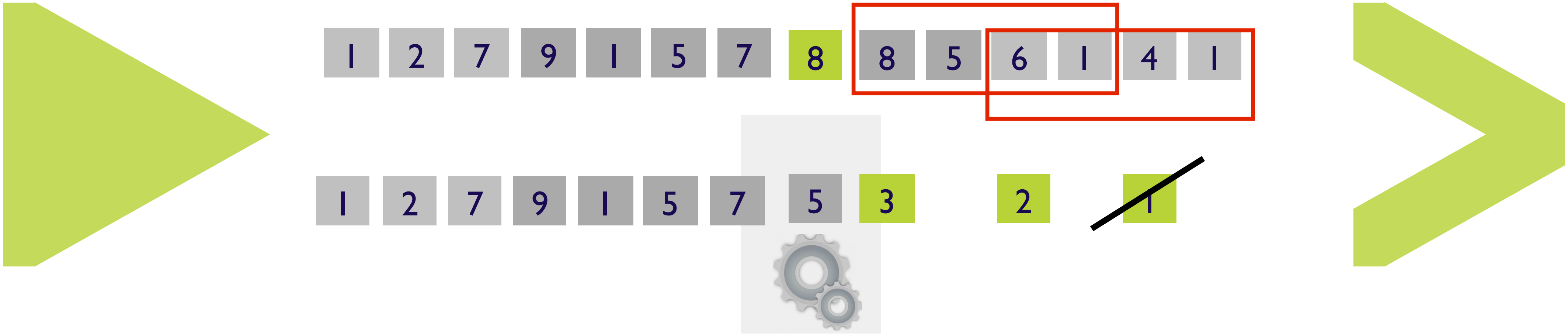


- Windows define a processing context that is updated incrementally
- Type of windows:
  - Batch or Sliding
  - Based on length, based on time, or both

Window queries

The count of withdrawals of amounts greater than 3, on a window of size 2, reported at every two events

```
select count(amount)
Withdrawal.win:length(4)
where amount > 3
output snapshot every 2 events
```



Windows

- A **sliding** window definition

Sum of withdrawals over the last 5 seconds

```
select sum(amount)
from Withdrawal.win:time(5 sec)
```

- A **batch** window definition

Sum of withdrawals at each 5 seconds (batch)

```
select sum(amount)
from Withdrawal.win:time_batch(5 sec)
```

Joins

- Joining two event streams

Which accounts under fraud surveillance have had a withdrawal in the last 30 seconds?

1. Fraud warning events stream for which we keep the last 30 minutes (1800 seconds).

2. Withdrawal events stream for which we consider the last 30 seconds.

The streams are joined on account number.

```
select fraud.accountNumber as accntNum,
       fraud.warning as warn, withdraw.amount as amount,
from FraudWarningEvent.win:time(1800) as fraud,
     WithdrawalEvent.win:time(30) as withdraw
where fraud.accountNumber = withdraw.accountNumber
```

Blocking Operations

Aggregation and Grouping

- A simple aggregate definition

Sums of the amounts every 5-seconds

```
select sum(amount)
from Withdrawal.win:time_batch(5 sec)
```

- Computing aggregates with grouping

Accounts where the average withdrawal amount per account for the last hour of withdrawals is greater then 1000

```
select account, avg(amount)
from Withdrawal.win:time(1 hour)
group by account
having amount > 1000
```

Ordering

- A simple ordering query

Batches of 5 or more stock tick events that are sorted first by price ascending and then by volume descending:

```
select symbol
from StockTickEvent.win:time(60 sec)
output every 5 events
order by price, volume desc
```

Correlated Queries

- A simple correlated query

Get all Order events whose quantity is greater that the sum of all the Order quantities over the last hour

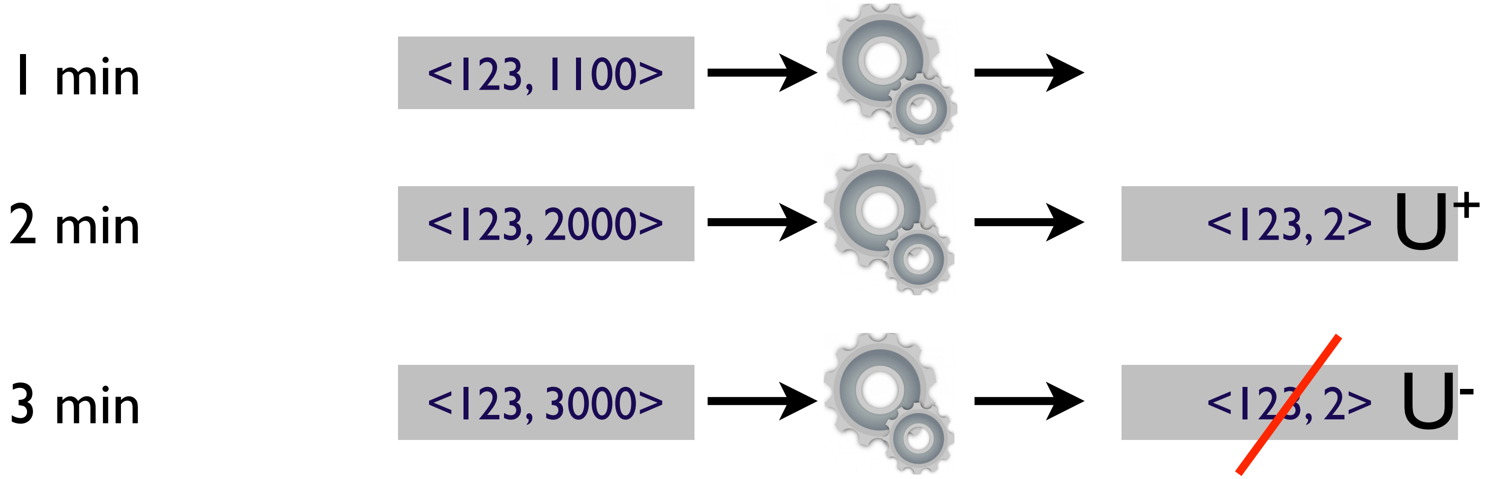
```
select *
from OrderEvent oe
where qty >
  (select sum(qty)
   from OrderEvent.win:time(1 hour) pd
   where pd.client = oe.client)
```

Insert and Remove Streams

Insert and remove streams

What should be result of the query that alerts all accounts that have more that 2 withdrawals with the last 5 mins (of 1000 euro)?

```
select accnt_no, count(*) as n_withdrawals
from Withdrawal.win:time(5 min)
where amount > 1000
group by acct_no
having count(*) = 2
```

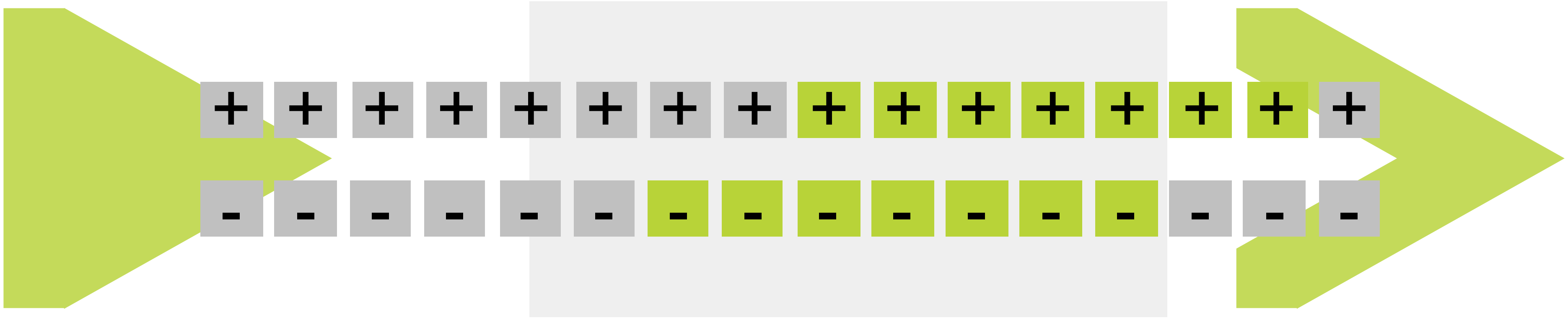


1 min <123, 1100> → →

2 min <123, 2000> → → <123, 2> U^+

3 min <123, 3000> → → ~~<123, 2>~~ U^-

Insert and remove streams



- Sources, channels, operators and sinks must handle positive (insert) and negative (remove) update streams

- The semantics of all operators must be defined accordingly

pjc@inesc-id.pt

http://web.tecnico.ulisboa.pt/paulo.carreira