dependable evolvable pervasive software engineering group

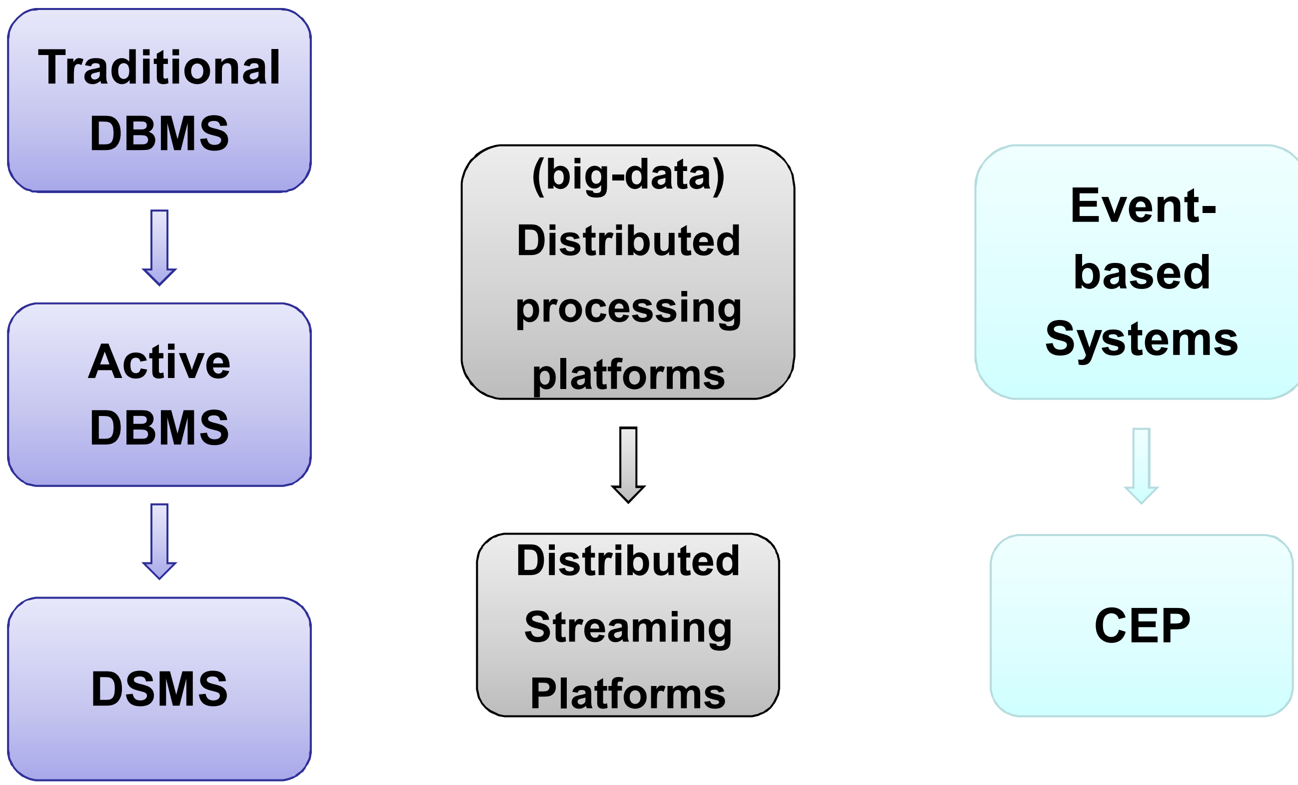
Complex Event Processing @ PoliMI

Gianpaolo Cugola

DEIB – Politecnico di Milano

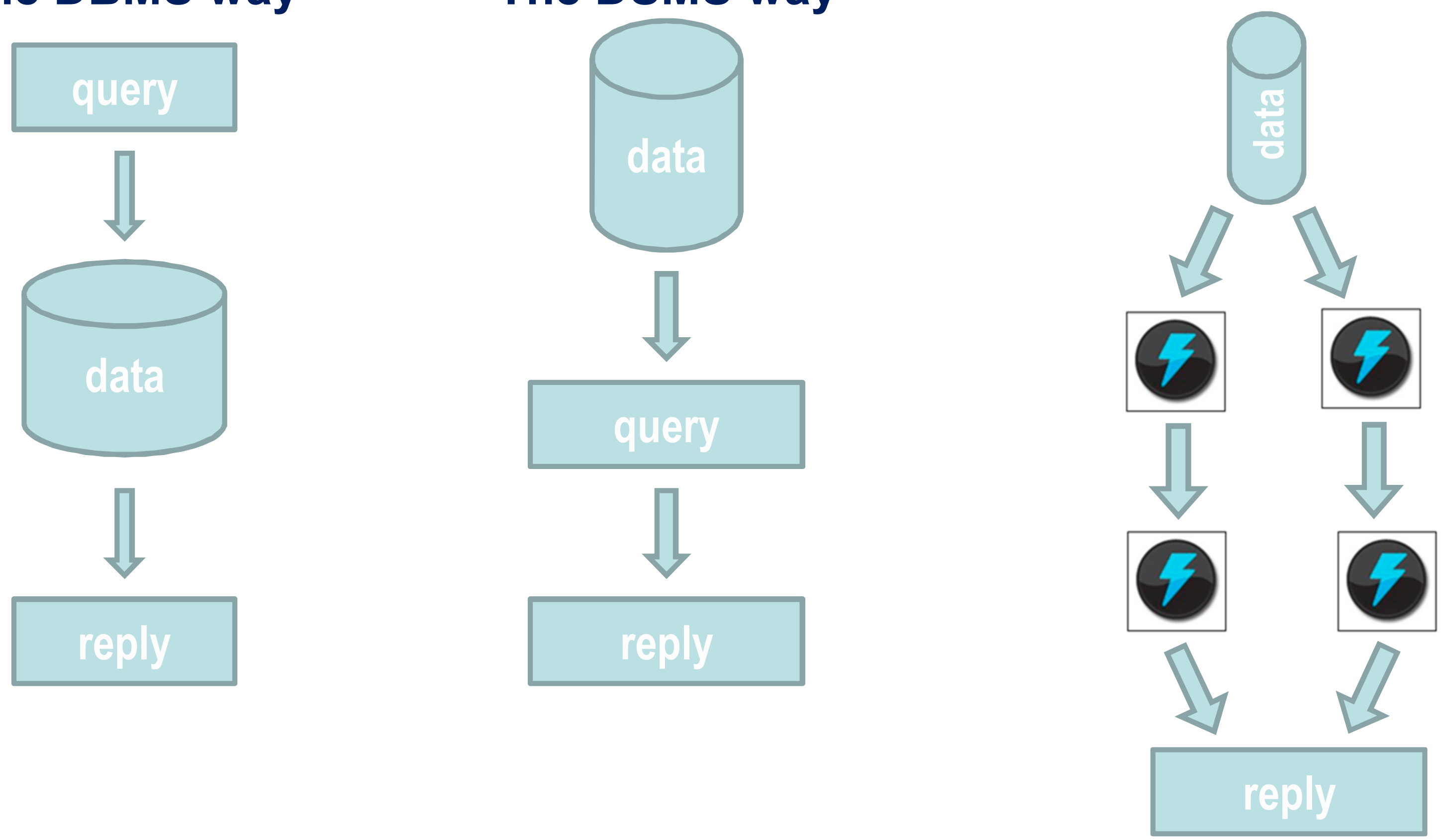
gianpaolo.cugola@polimi.it

On-line processing of (big) data: Two approaches



From DBMS to DSMS and Streaming Platforms

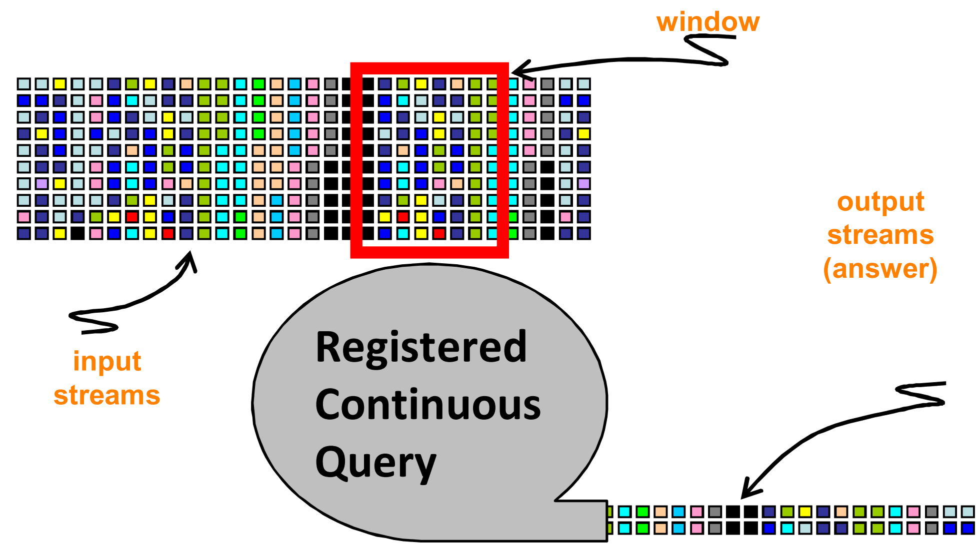
The DBMS way **The DSMS way** **The Distributed Streaming way**



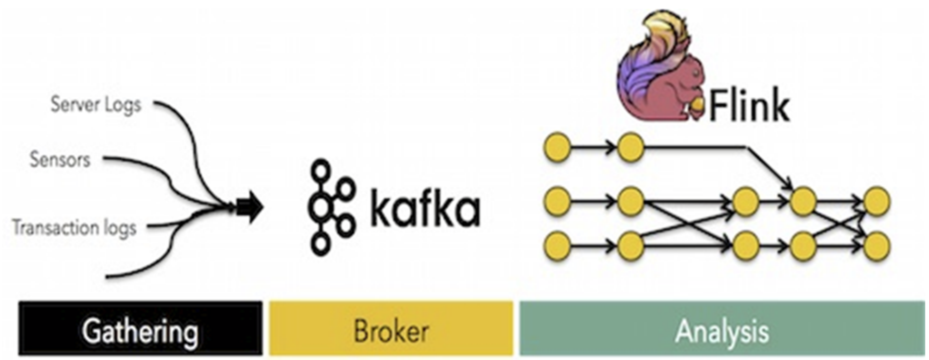
Data Stream Management Systems

- The continuous nature of streams requires a paradigmatic change:
  - from persistent data stored and queried on demand
    - *One-time semantics*
  - to transient data consumed on the fly by continuous queries
    - *Continuous semantics*
- Continuous queries often operates through *windows*

```
CQL/Stream:
Select IStream(*)
From      F1[Rows 5],
          F2[Rows 10]
Where  F1.A = F2.A
```



Distributed Stream Processing



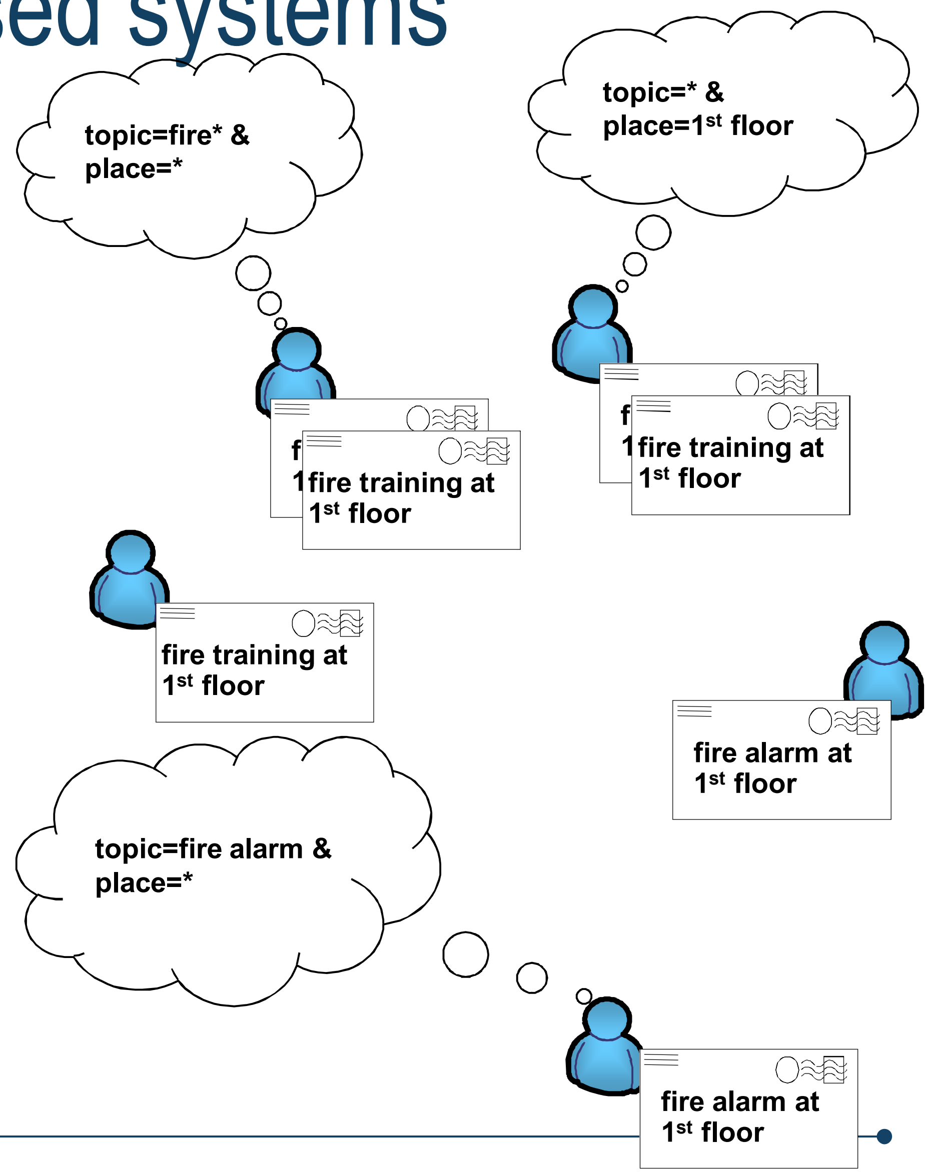
General Architecture

Streaming API

```
case class Event(location: Location, numVehicles: Long)

val stream: DataStream[Event] = …;

stream
    .filter { evt => isIntersection(evt.location) }
    .keyBy("location")
    .timeWindow(Time.minutes(15), Time.minutes(5))
    .sum("numVehicles")

    .keyBy("location")
    .mapWithState { (evt, state: Option[Model]) => {
      val model = state.orElse(new Model())
      (model.classify(evt), Some(model.update(evt)))
    }}
```

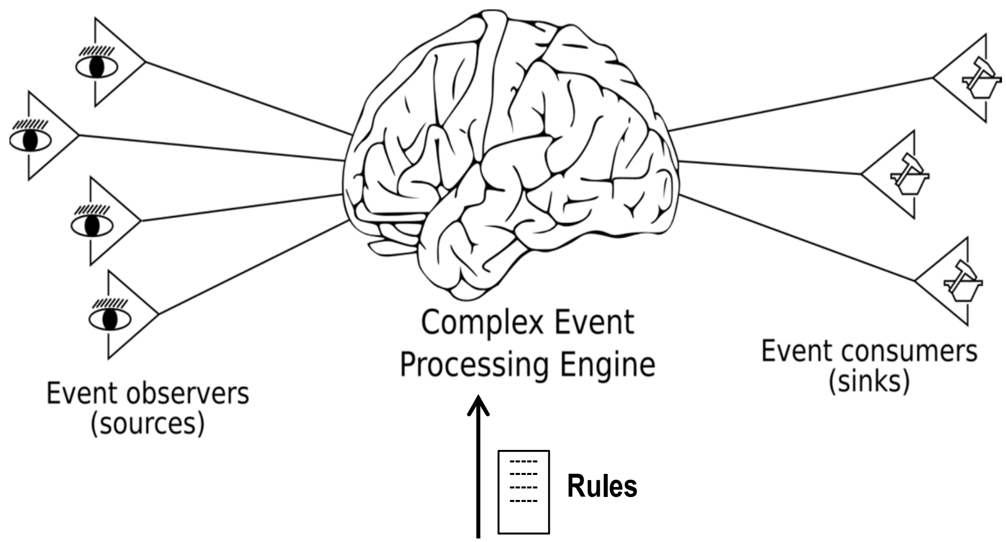
Event-based systems

- Components collaborate by exchanging information about occurrent *events*. In particular:
  - Components *publish* notifications about the events they observe, or
  - they *subscribe* to the events they are interested to be notified about
- Communication is:
  - Purely message based
  - Asynchronous
  - Multicast
  - Implicit
  - Anonymous



Complex Event Processing (CEP)

- CEP systems adds the ability to deploy *rules* that describe how composite events can be generated from primitive (or composite) ones
- Typical CEP rules search for *sequences of events*
  - Raise C if A→B
- Time is a key aspect in CEP



Several tools

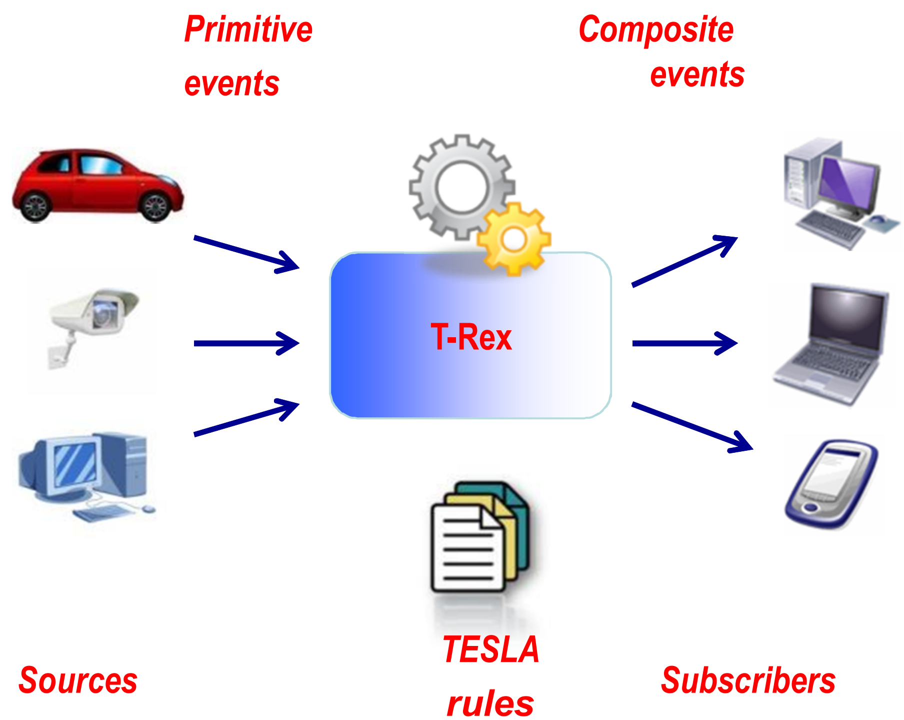
- Distributed stream computing platforms/frameworks
  - STORM: http://storm-project.net/
  - SPARK STREAMING: https://spark.apache.org/streaming/
  - Apache Samza: http://samza.apache.org/
  - Apache Flink: https://flink.apache.org/
- Open source DSMS/CEP
  - Esper: http://www.espertech.com/esper/
  - WSO2 Complex Event Processor http://wso2.com/products/complex-event-processor/
  - T-Rex (PoliMI)
- Commercial DSMS/CEP
  - IBM InfoSphere Streams, TIBCO StreamBase, Oracle CEP, SAP's Sybase CEP, Microsoft StreamInsight

Distributed Stream Processing @ PoliMI

- Streaming operators are typically stateless
  - Such that they can be easily replicated/distributed operating on different stream partitions
- In several applications it is necessary to have stateful operators…
- …and share state among different operators
  - Especially true for data mining and machine learning alg.
- Goal: Extend existing platforms (namely Apache Flink) to support such shared state…
- … with minimal impact on performance

CEP @ PoliMI: T-Rex

- T-Rex receives *primitive events published* by one or more *sources*
  - Embedded sensors, but also legacy systems...
- Processes those events
  - Using a set of *rules* written in an ad-hoc language: *TESLA*
  - To derive new information as a set of *composite events*
- Delivers events to interested components (i.e., *subscribers*)
  - E.g., mobile devices, …



TESLA: The rule language of T-Rex

```
Define      CE(Att₁ : Type₁, …, Attₙ : Typeₙ)
From        Pattern
Where       Att₁ = f₁(..), …, Attₙ = fₙ(..)
Consuming   e₁, …, eₘ
```

TESLA: An example

```
Define GrowingDelay(train_id: string, newDelay: int, oldDelay: int)
From TrainDelay(train_id = $t, delay = $d) as T1
  and last TrainDelay(train_id=$t, delay<$d) as T2
       within 10m from T1
Where train_id := T1.train_id, newDelay:=T1.delay,
      oldDelay:=T2.delay;
```

```
Define GrowingDelay(train_id: string, newDelay: int, oldDelay: int)
From TrainDelay(train_id = $t, delay = $d, delay>10) as T1
  and last TrainDelay(train_id=$t, delay<$d) as T2
       within 10m from T1
  and not TrainDelay(train_id=$t, delay>=$d)
       between T1 and T2
Where train_id := T1.train_id, newDelay:=T1.delay,
      oldDelay:=T2.delay;
```

Thanks f  attention!

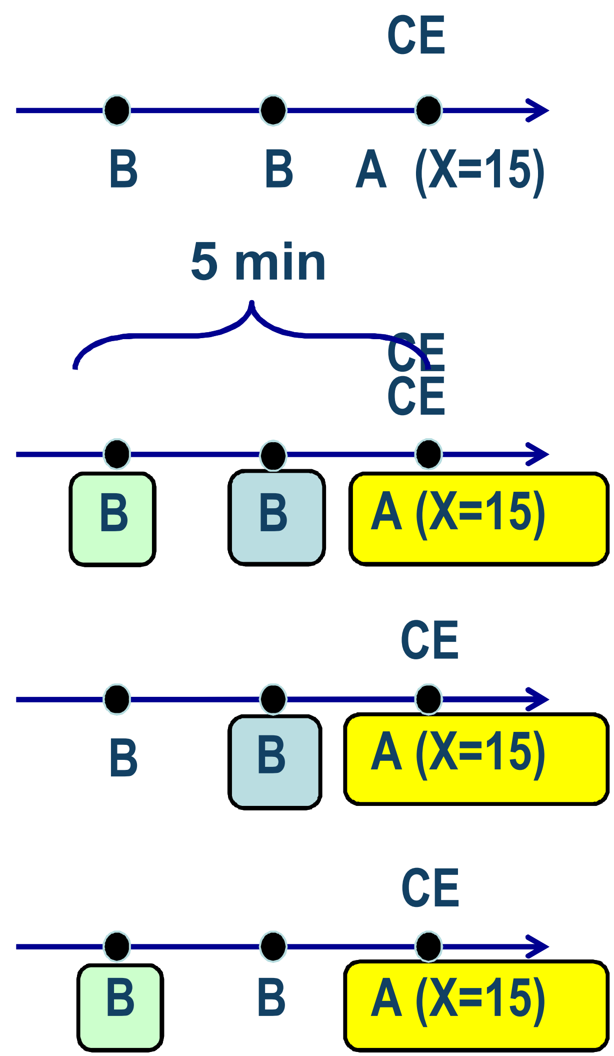
Gianpaolo Cugola

DEIB – Politecnico di Milano

gianpaolo.cugola@polimi.it

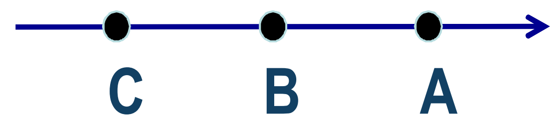
Patterns in TESLA

- Selection of a single event
  - `A(x>10)`
  - `Timer()`
- Selection of sequences
  - `A(x>10) and each B within 5 min from A`
  - `A(x>10) and last B within 5 min from A`
  - `A(x>10) and first B within 5 min from A`
  - Generalization
    - n-first / n-last

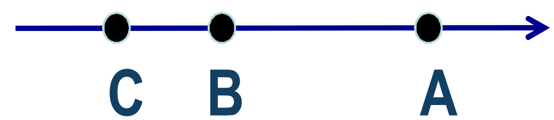


Patterns in TESLA

- TESLA allows *-within operators to be composed with each other:
  - In chains of events
    - `A and each B`
      `within 3 min from A`
      `and last C`
      `within 2 min from B`



  - In parallel
    - `A and each B`
      `within 3 min from A`
      `and last C`
      `within 4 min from A`



- Parameters can be added between events in a pattern

Parameters

- Parameters can be added between events in a pattern
  - `A(a=$x) and each B(a=$x) within 3 min from A`
    `and last C(a=$x) within 4 min from A`

Negations and Aggregates

- Two kinds of negations:
  - Interval based:
    - `A and last B`
`within 3 min from A`
`and not C between B and A`
  - Time based:
    - `A and not C within 3 min from A`
- Similarly, two kinds of aggregates
  - Interval based
    - Use values appearing between two events
  - Time based
    - Use values appearing in a time interval

Hierarchies of events

- TESLA allows to define hierarchies of events
  - Composite events can be used to define (new) composite events