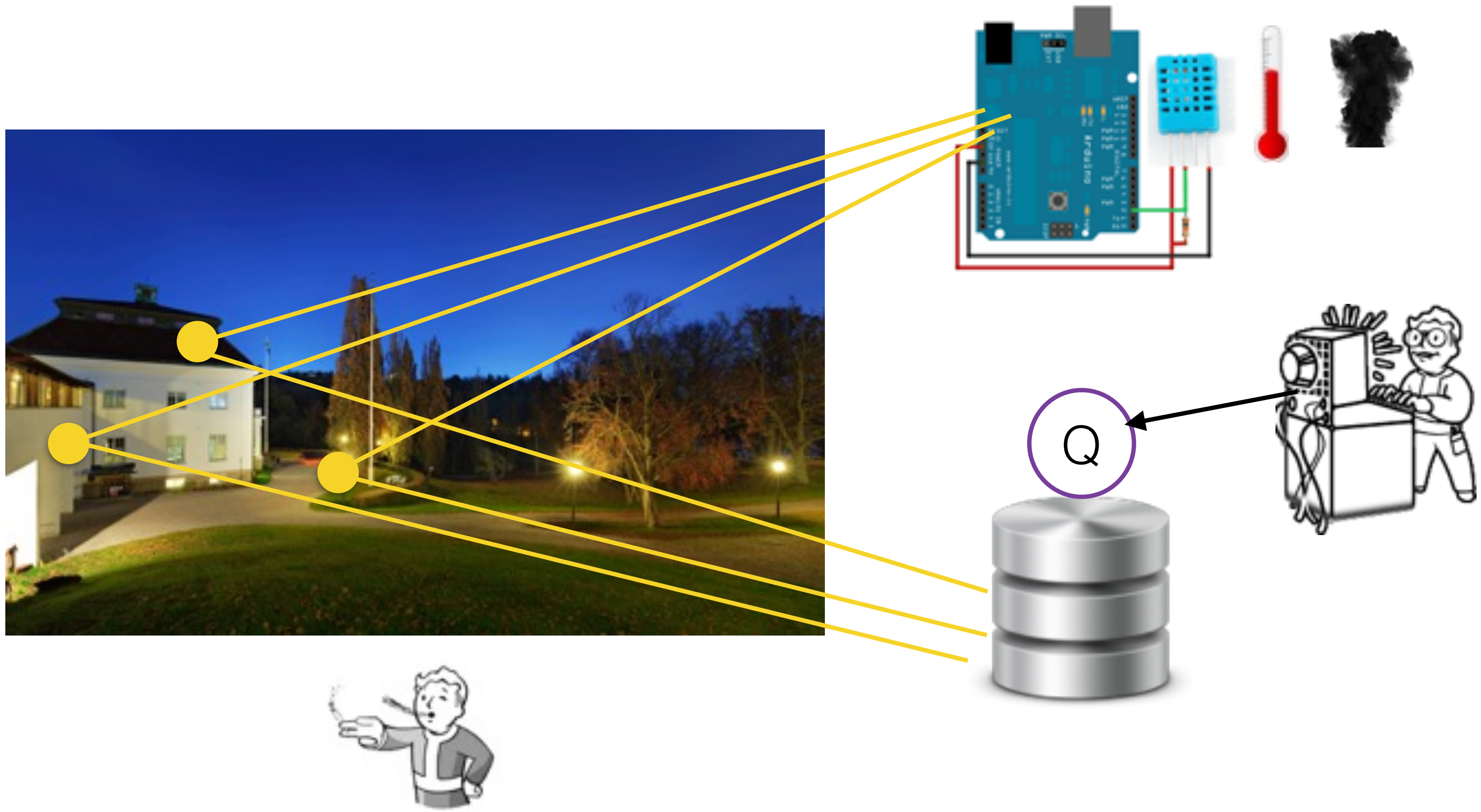
An Introduction to Distributed Data Streaming

Elements and Systems

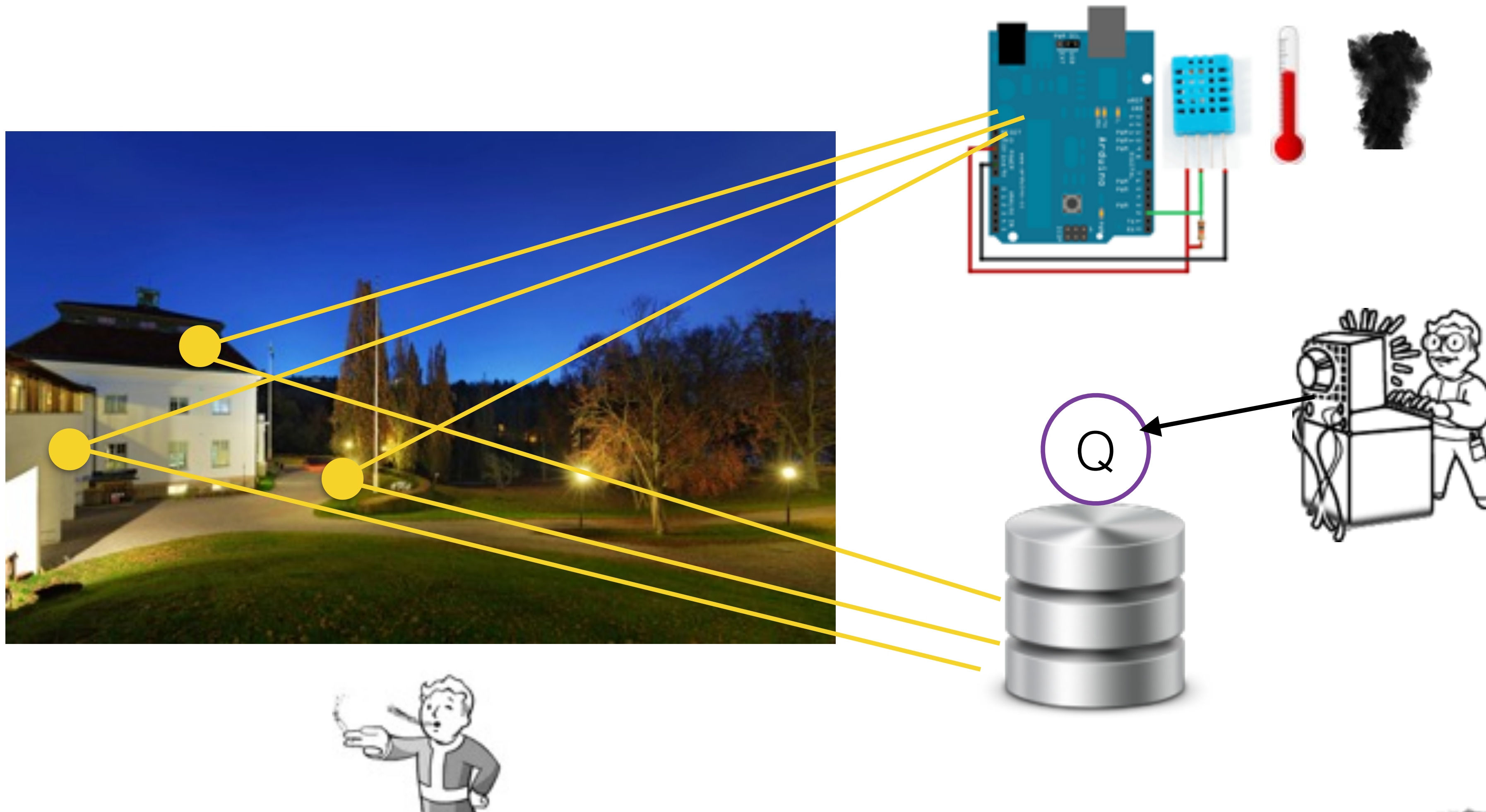
Paris Carbone<parisc@kth.se>
PhD Candidate
KTH Royal Institute of Technology

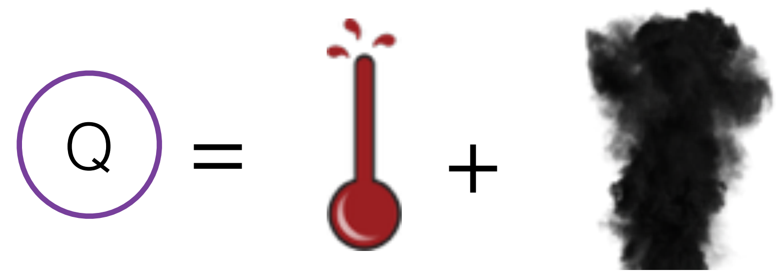
how to avoid this?

how to avoid this?



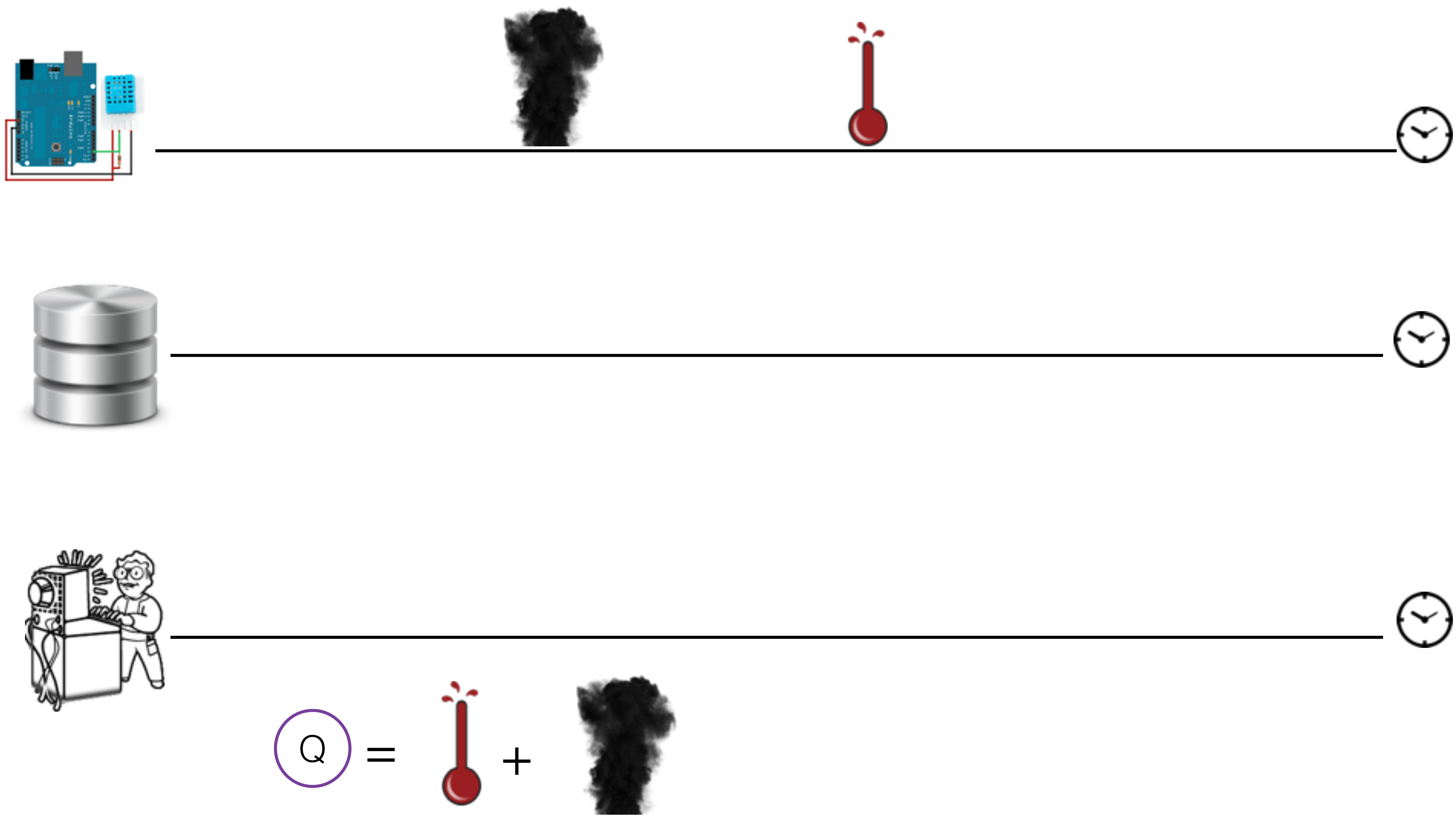
how to avoid this?



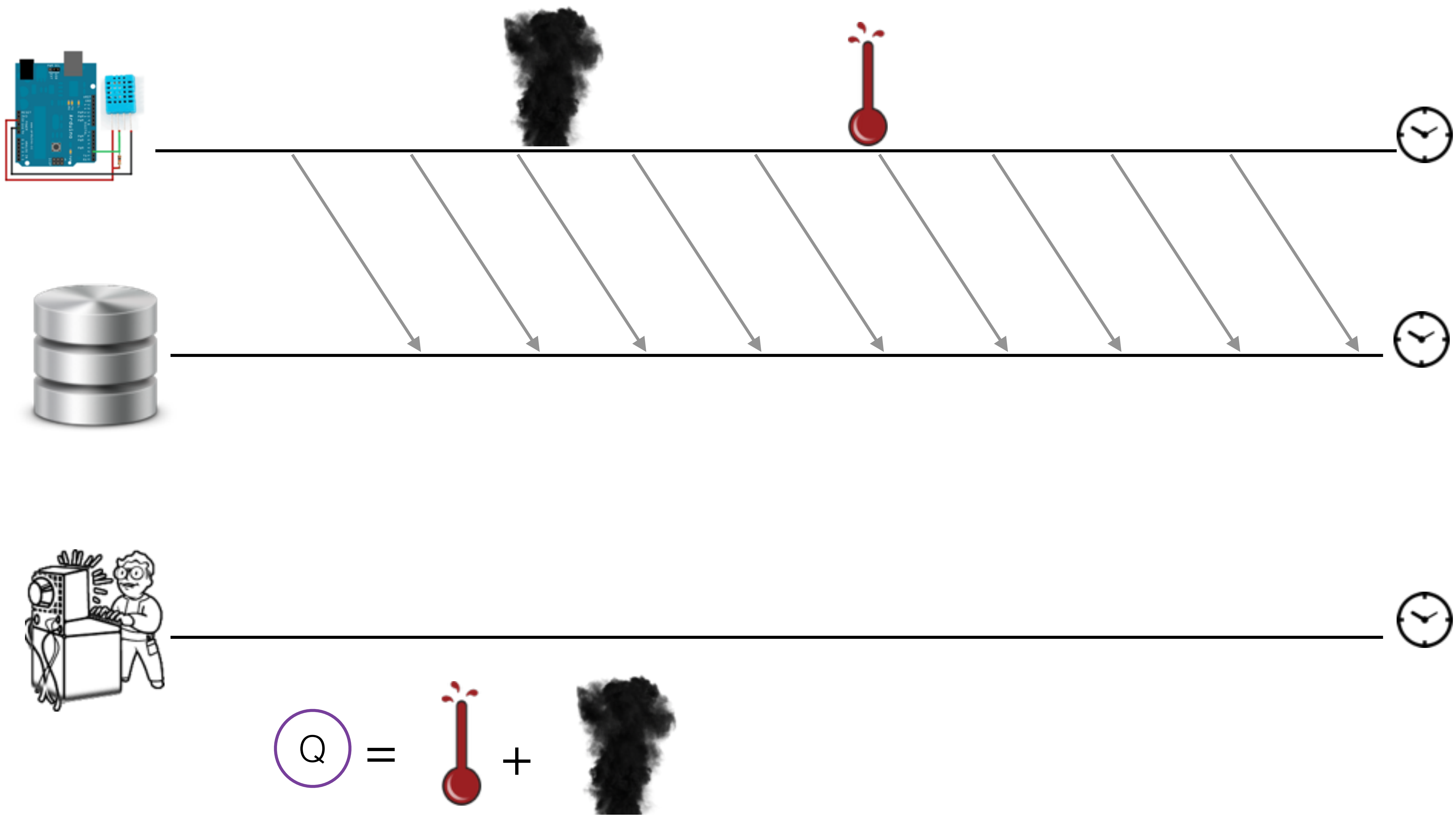


how to avoid this?

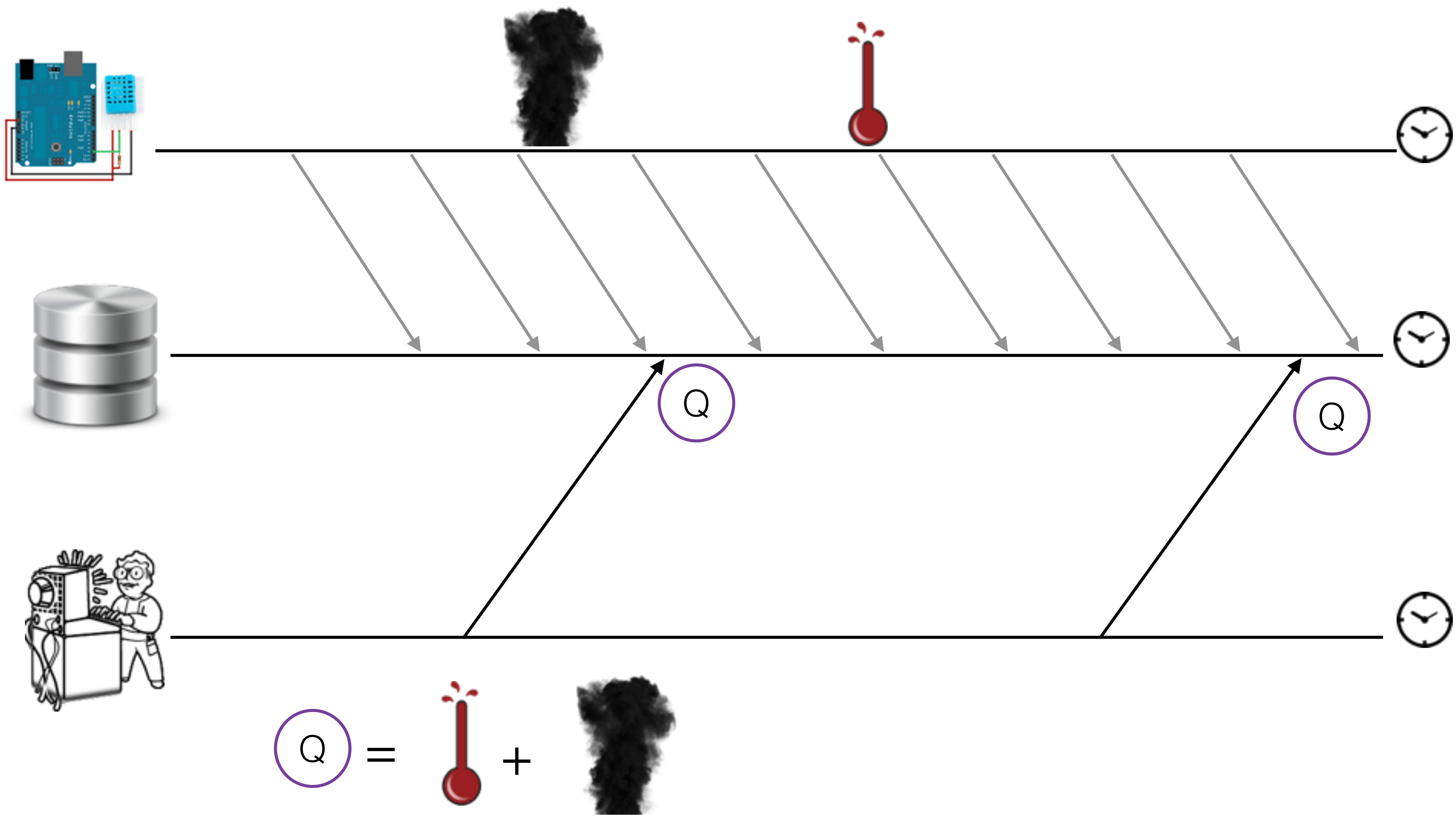
Motivation



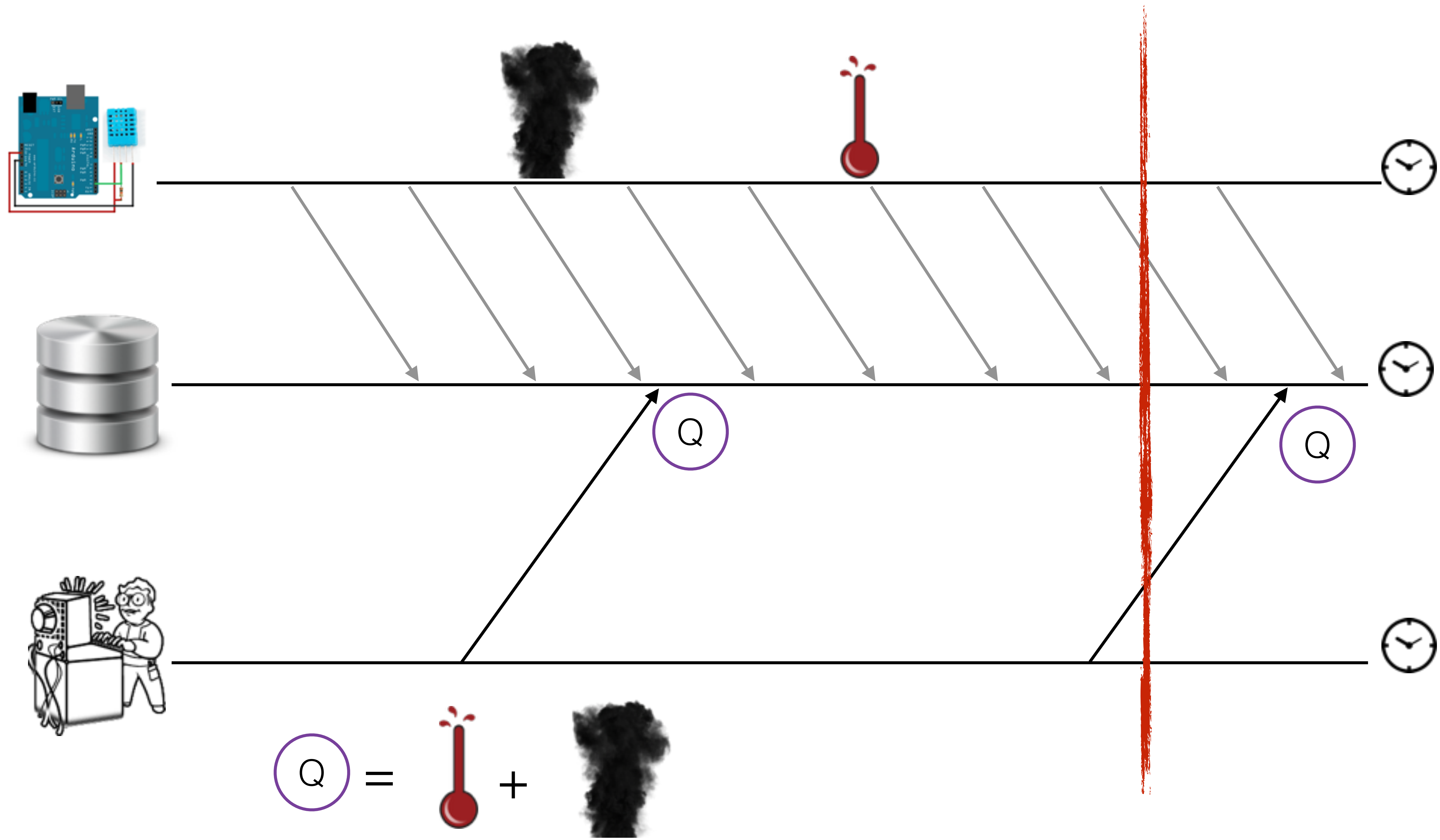
Motivation



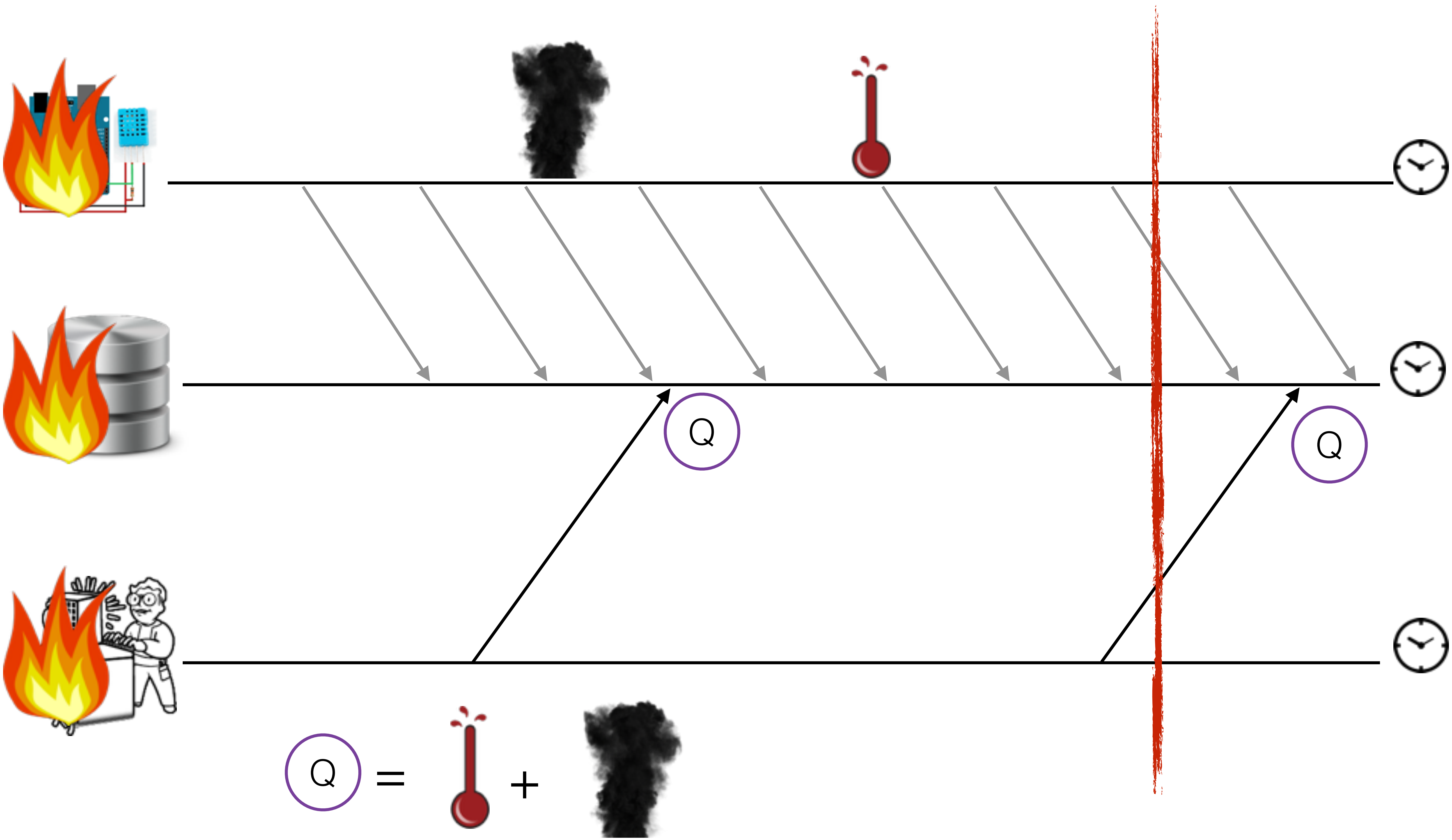
Motivation



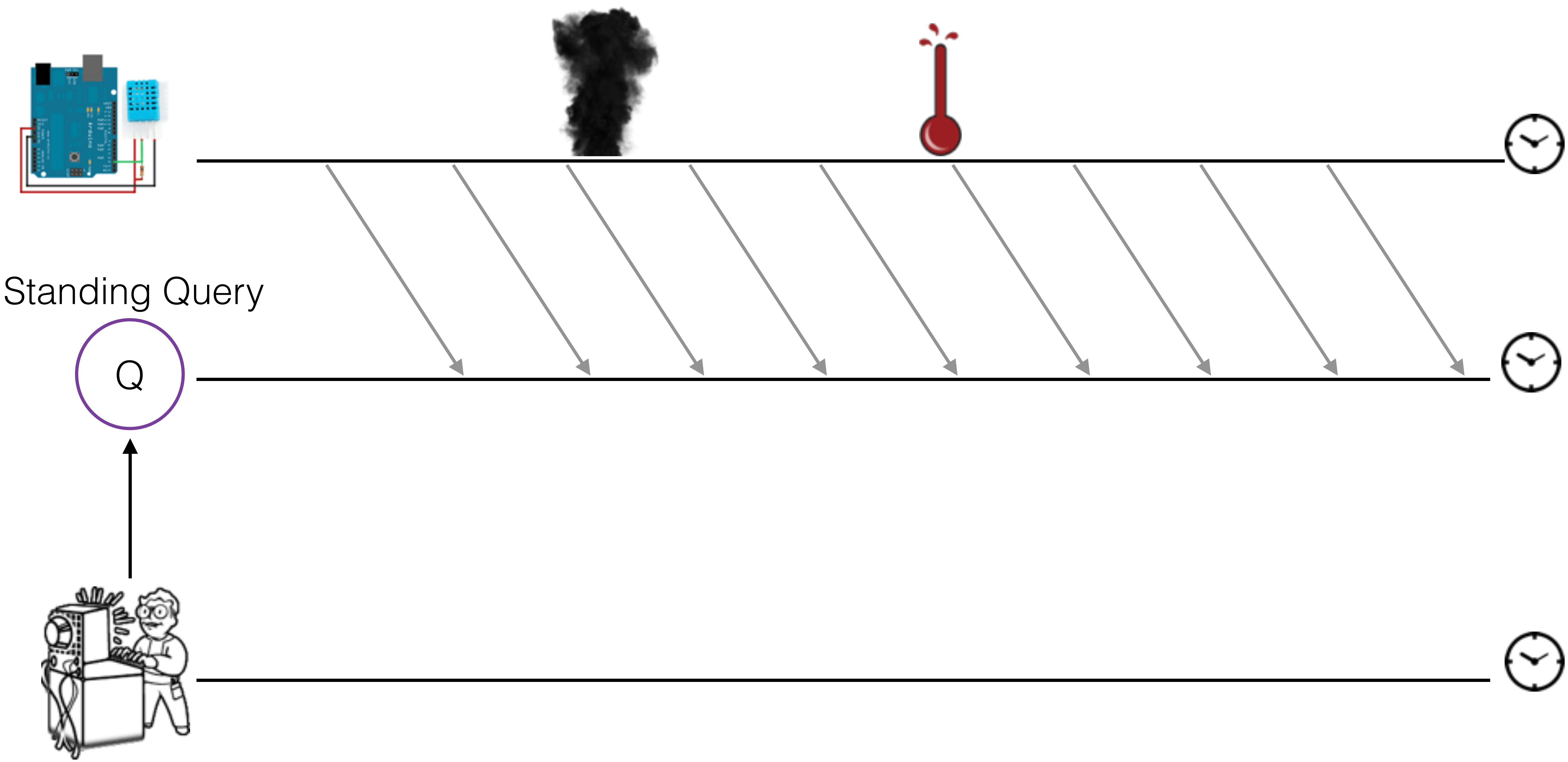
Motivation



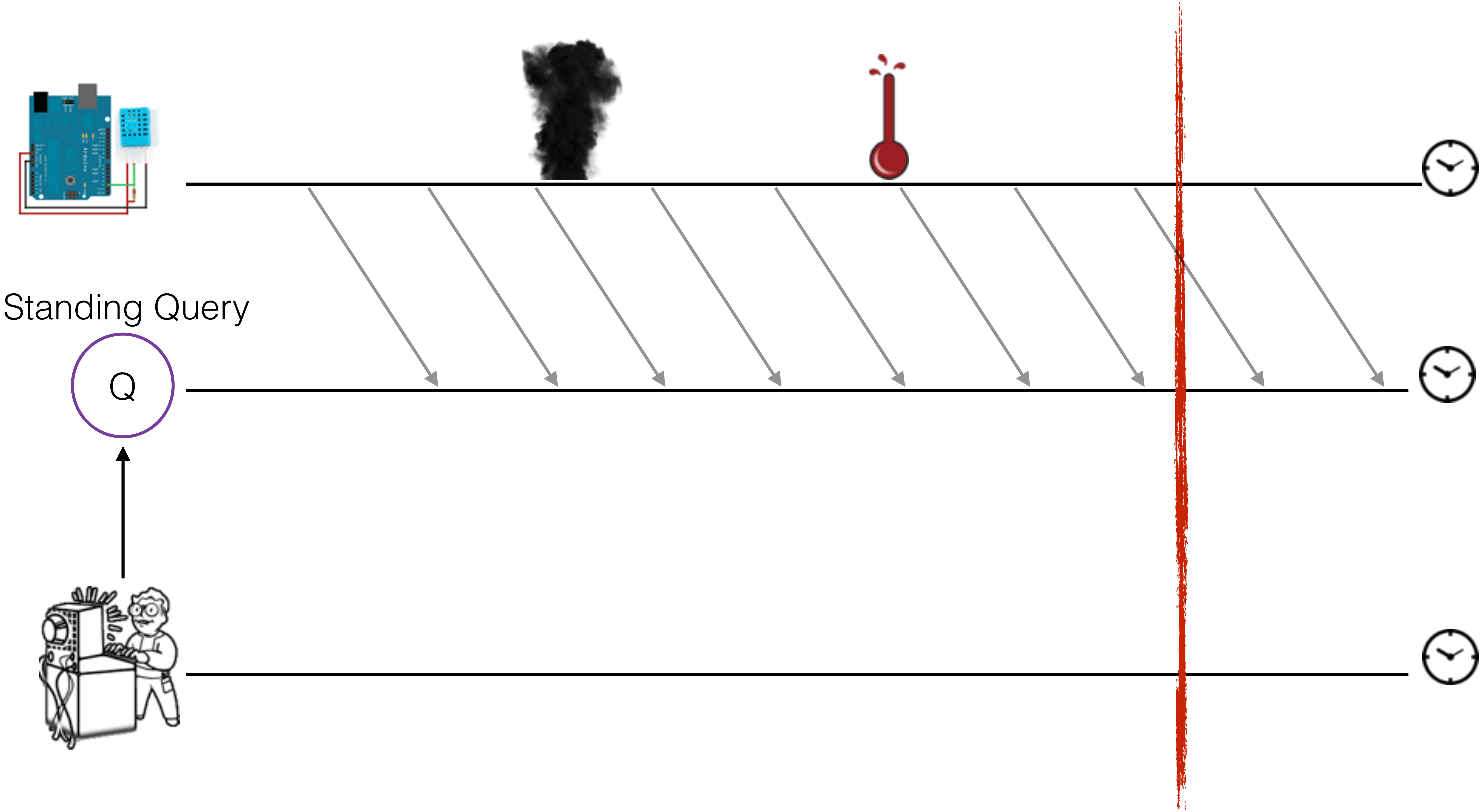
Motivation



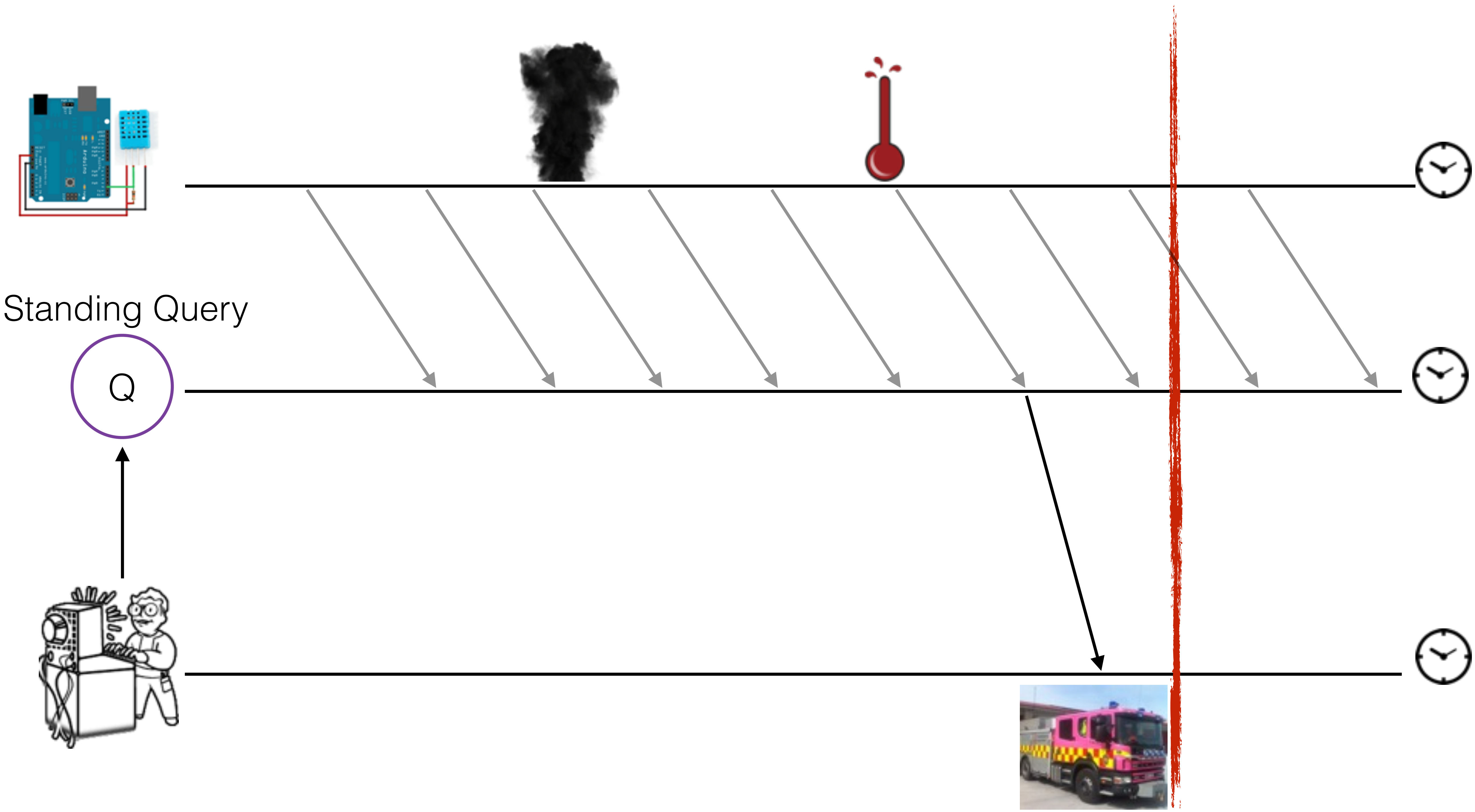
Motivation



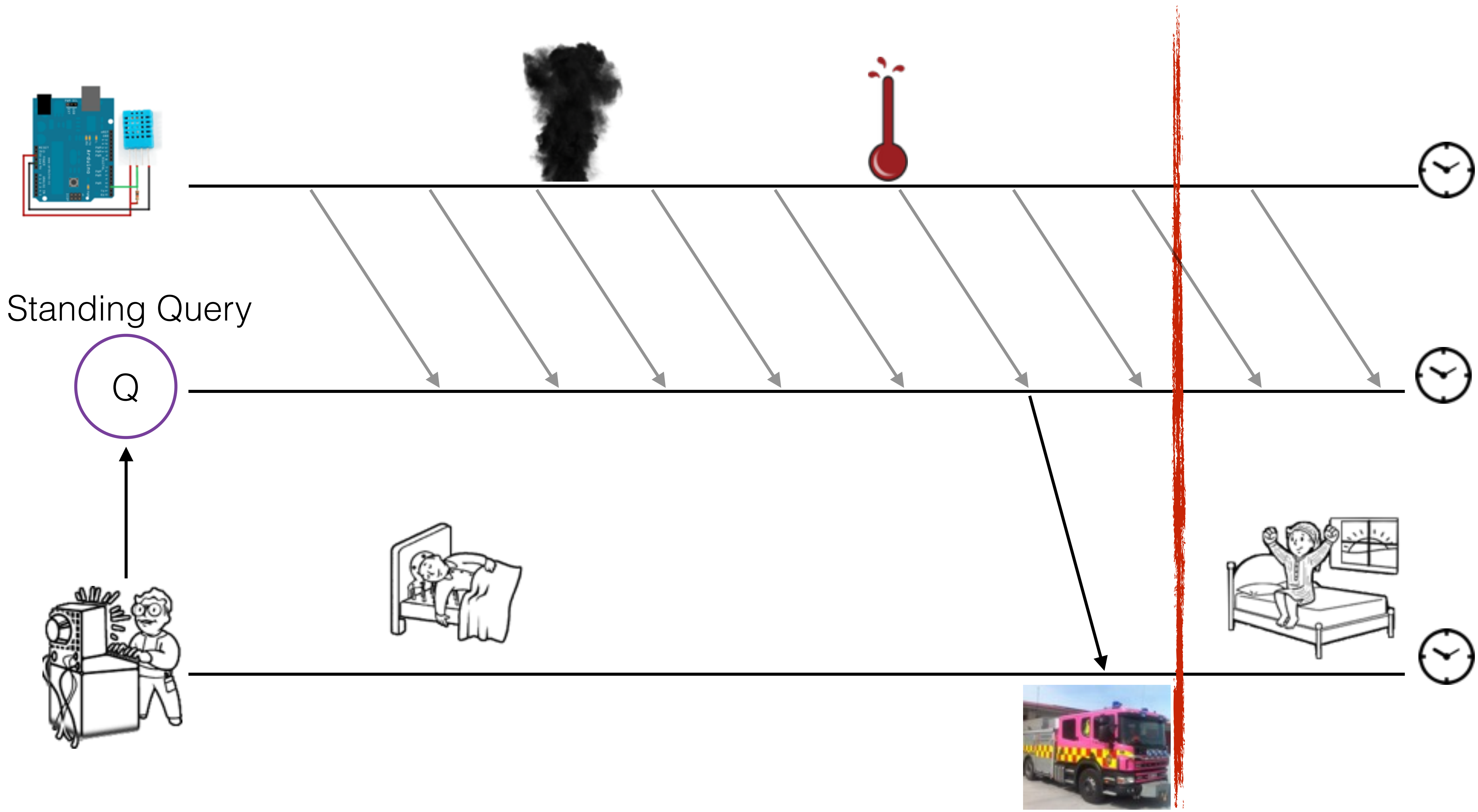
Motivation



Motivation



Motivation

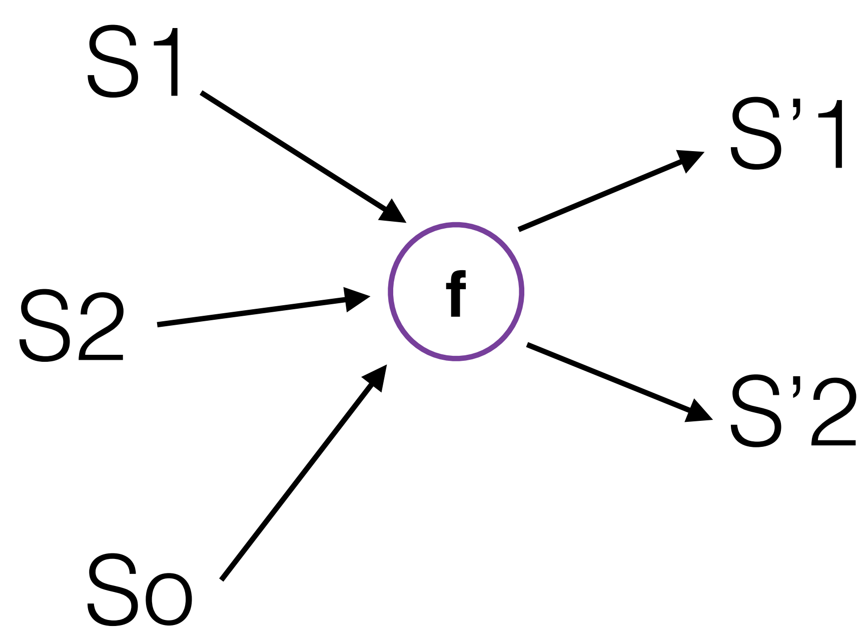


Preliminaries

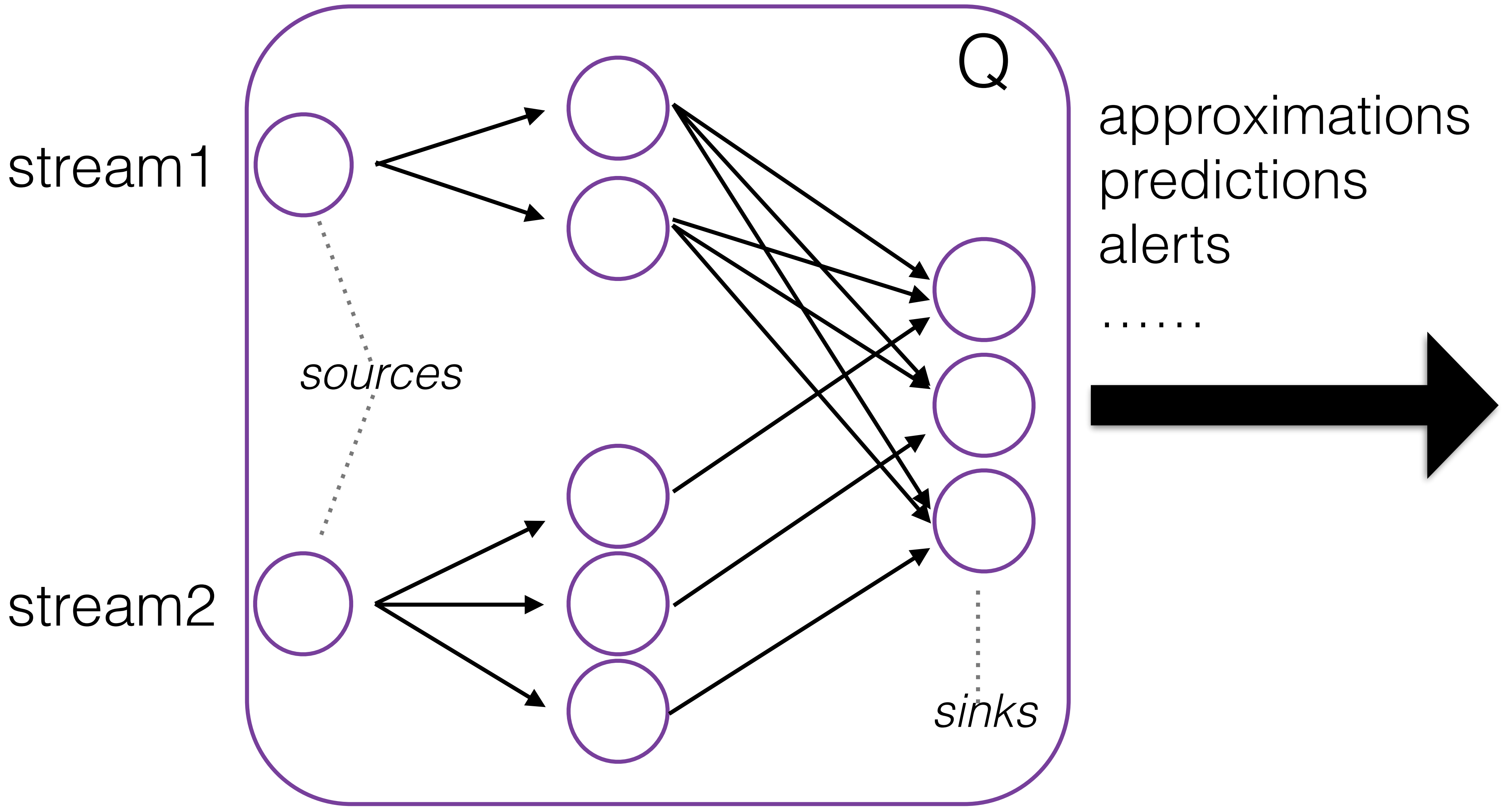
- Data Streaming Paradigm
  - Incoming data is unbound - continuous arrival
  - Standing queries are evaluated <u>continuously</u>
  - Queries operate on the full data stream or on the most recent views of the stream ~ windows

Data Streams Basics

- Events/Tuples : elements of computation - respect a schema
- Data Streams : <u>unbounded</u> sequences of events
- Stream Operators: consume streams and generate new ones.
    - Events are consumed once - no backtracking!



Streaming Pipelines



Core Abstractions

- Windows
- Synopses (summary state)
- Partitioning

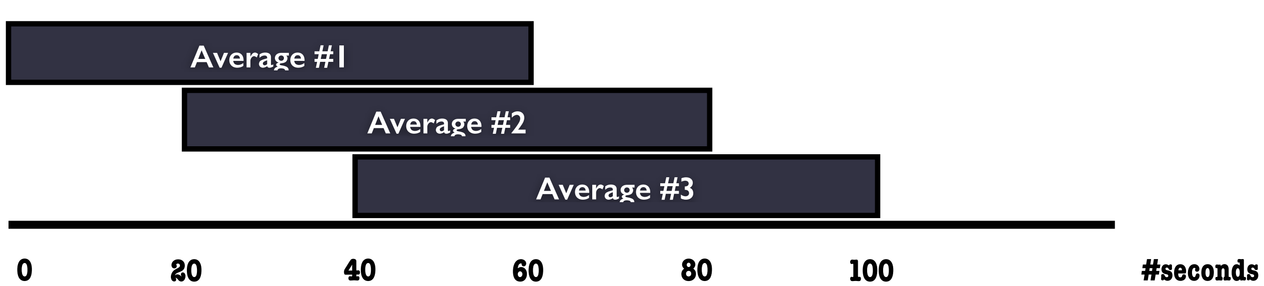
Windows

Discussion

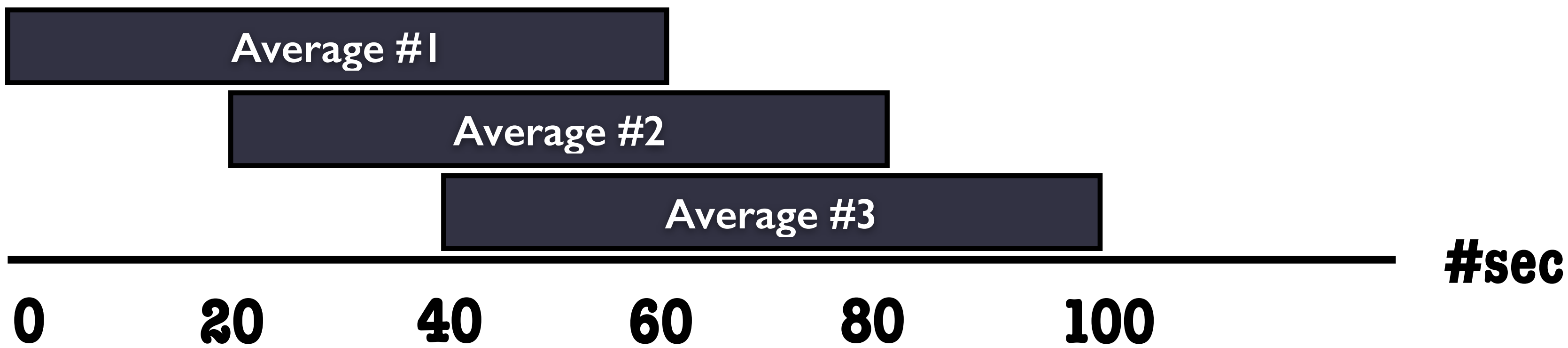
Why do we need windows?

Windows

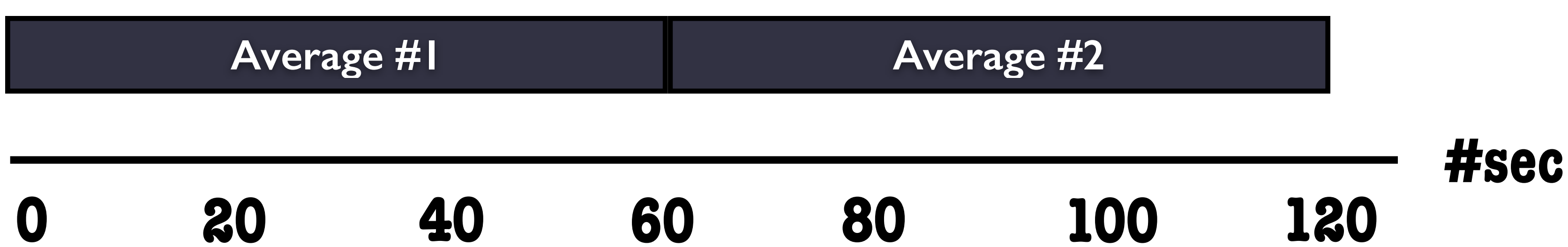
- We are often interested only in **fresh data**
    - f = “*average temperature over the last minute every 20 sec*”
- **Range:** Most data stream processing systems allow window operations on the most recent history (eg. 1 minute, 1000 tuples)
- **Slide:** The frequency/granularity f is evaluated on a given range



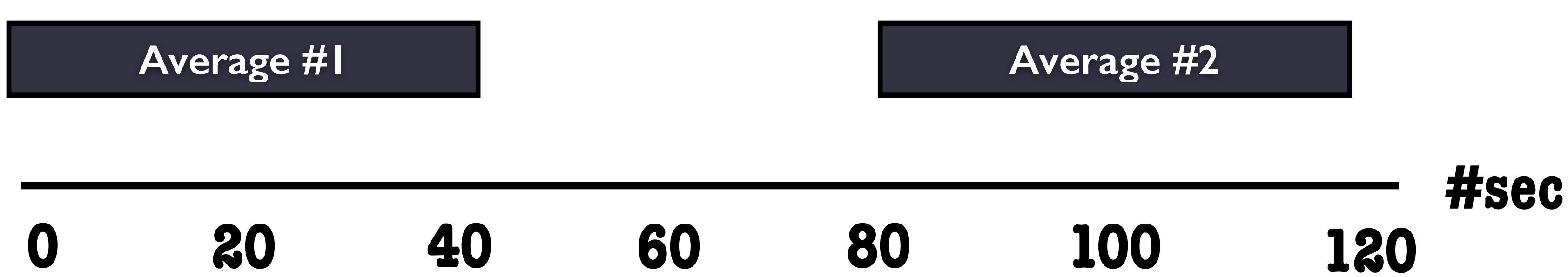
Window Types



Sliding *range > slide*



Tumbling *range = slide*

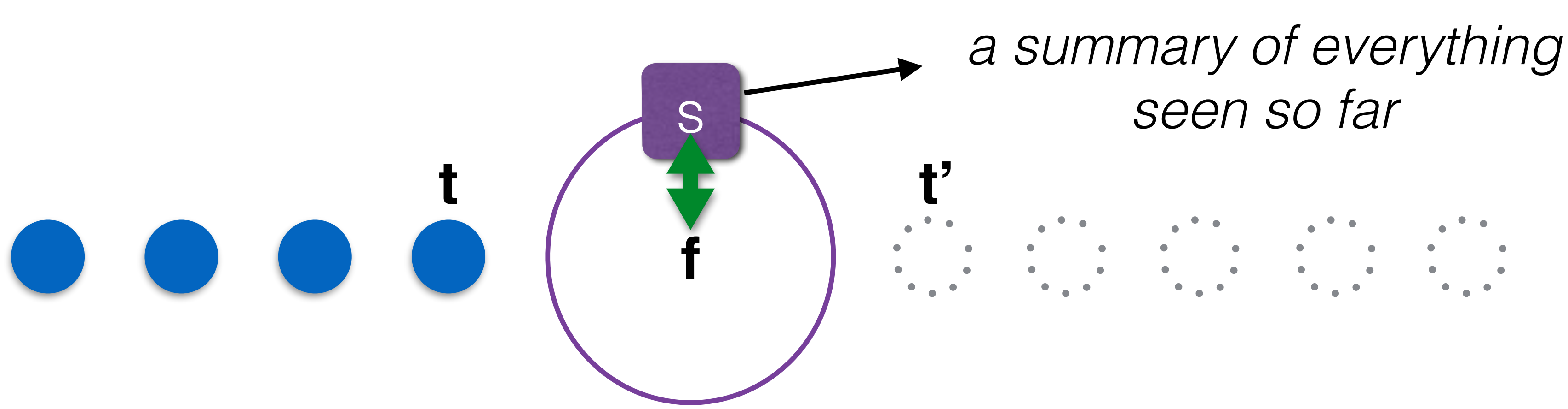


Jumping *range < slide*

Synopses

We cannot infinitely store all events seen

- **Synopsis**: A summary of an infinite stream
  - It is in principle any streaming operator state
- Examples: samples, histograms, sketches, state machines…



1. process t, s
2. update s
3. produce t’

What about window synopses?

Synopses-Aggregations

- **Discussion** - Rolling Aggregations
- Propose a synopsis, s=? when
  - f= max
  - f= ArithmeticMean
  - f= stDev

Synopses-Approximations

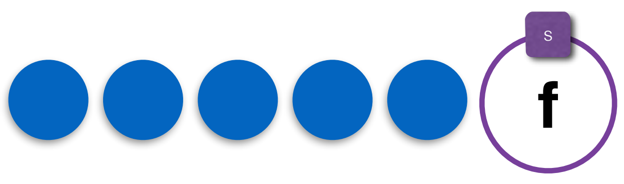
- **Discussion** - Approximate Results
- Propose a synopsis, s=? when
  - f= uniform random sample of k records over the whole stream
  - f= filter distinct records over windows of 1000 records with a 5% error

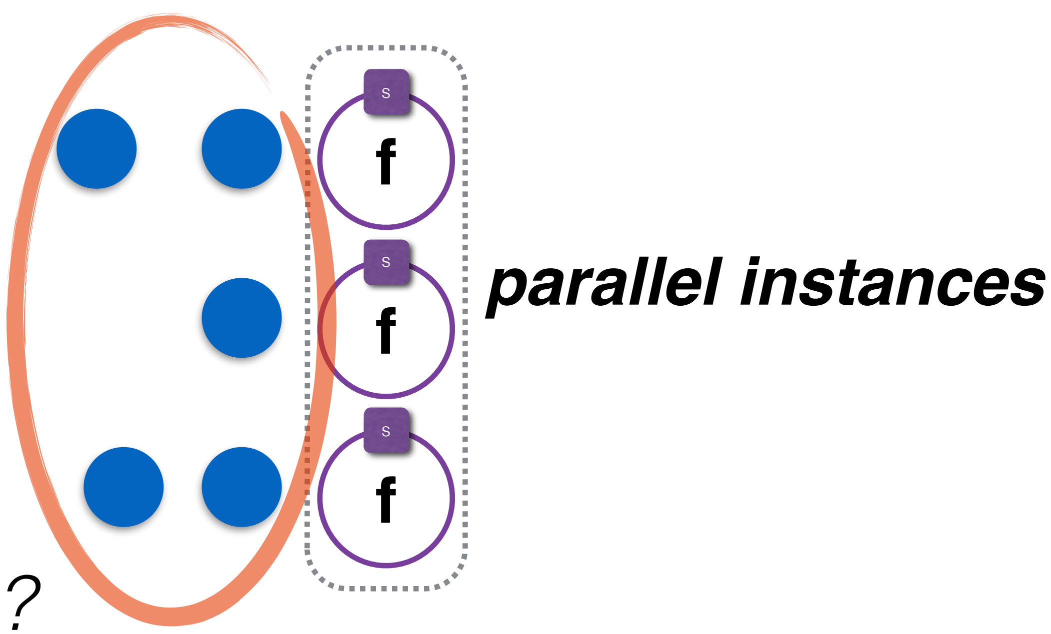
Synopses-ML and Graphs

- Examples of cool synopses to check out
  - **Sparsifiers/Spanners** - approximating graph properties such as shortest paths
  - **Change detectors** - detecting concept drift
  - **Incremental decision trees** - continuous stream training and classification

Partitioning

- One stream operator is not enough
  - **Data** might be too large to process
    - e.g. very high input rate, too many stream sources
  - **State** could possibly not fit in memory



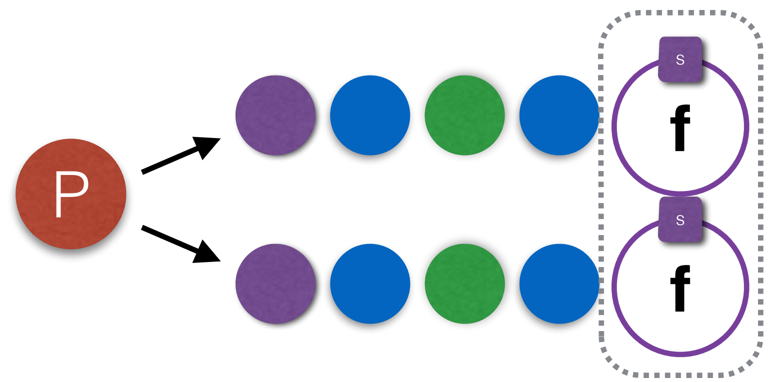


How do we
partition the input streams?

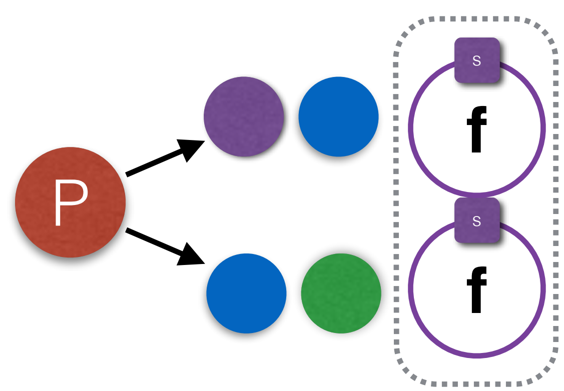
Partitioning

- Partitioning defines how we allocate events to each parallel instance. Typical partitioners are:

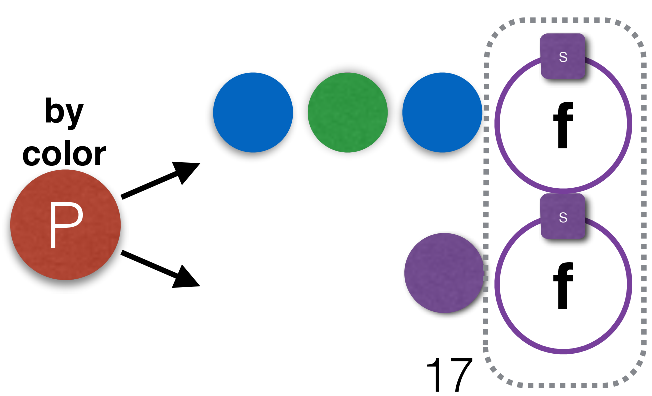
- Broadcast



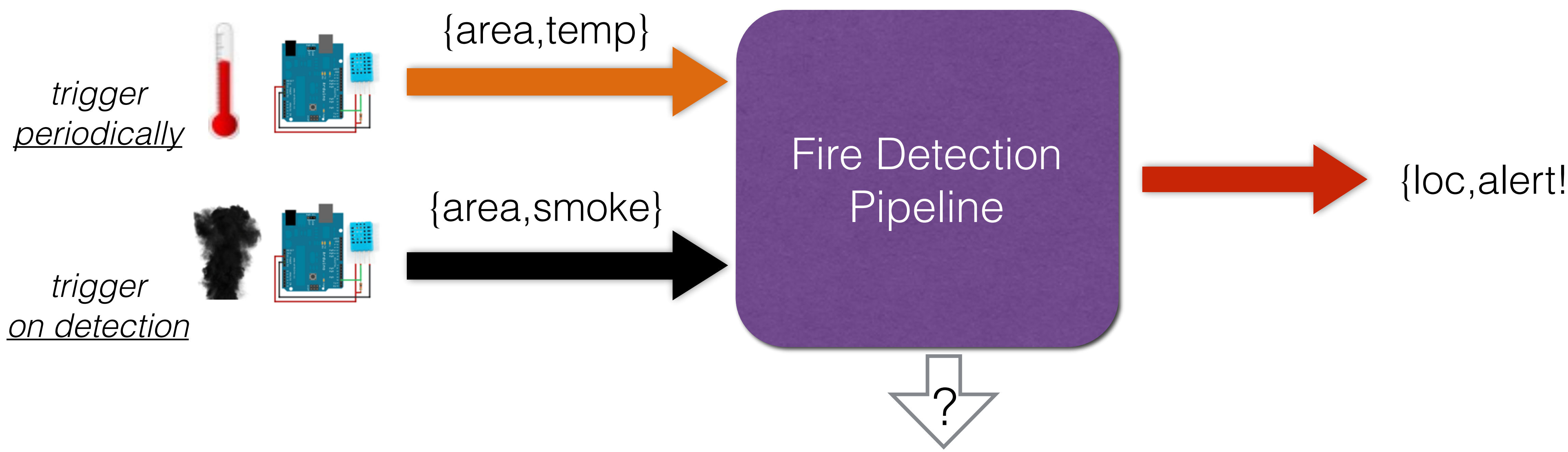
- Shuffle



- Key-based



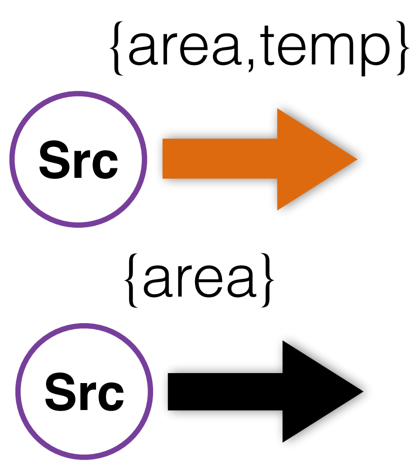
Putting Everything Together



- operators
  - synopses
  - windows
  - partitioning

Operators

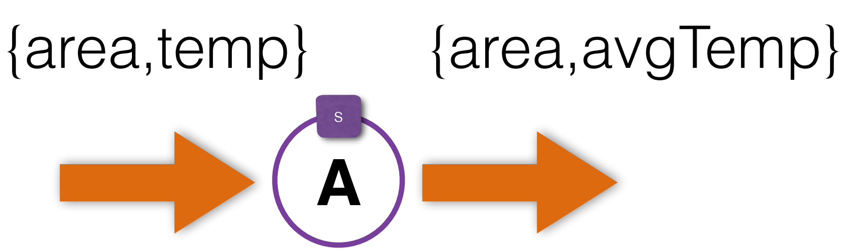
Sensor Data Sources



Periodic Temperature Updates

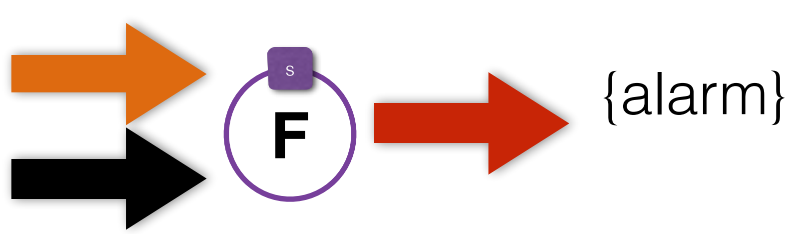
Smoke Detections

Rolling Arithmetic Mean of Temperatures



trivial…

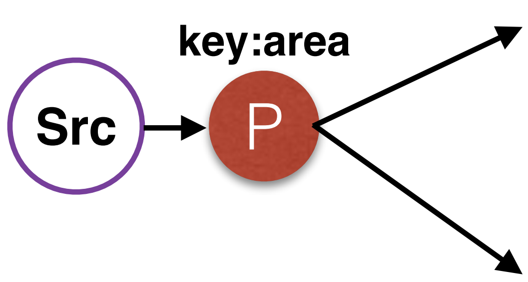
State Machine-based Fire Alarm



What is the state and its transitions?

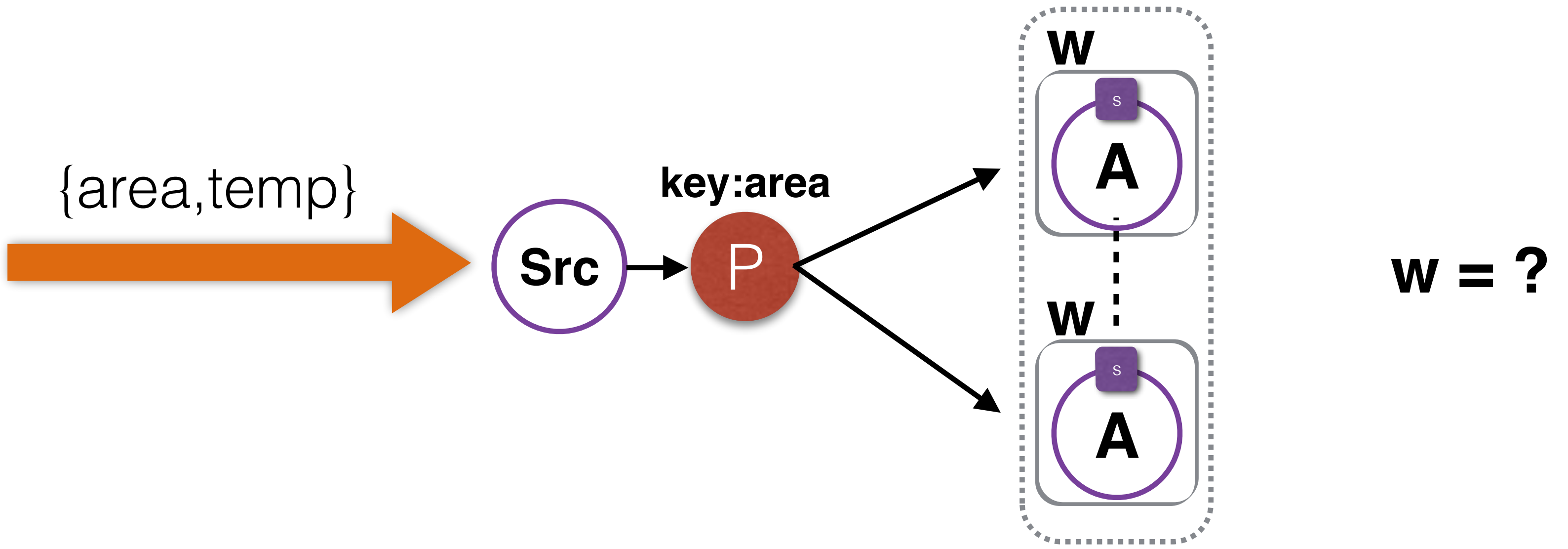
Partitioning

- We are only interested in correlating smoke and high temperature within the same area
- Events carry area information so we can partition our computation by **area**



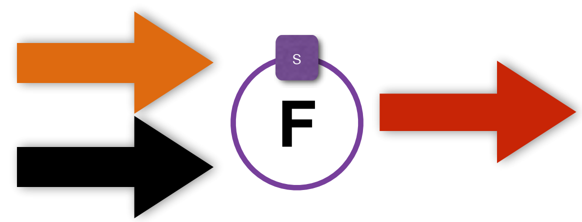
Windowing

- Individual sensor data could be potentially faulty
- We need to gather data from all temperature sensors of an area and produce an average
- We want fresh average temperatures

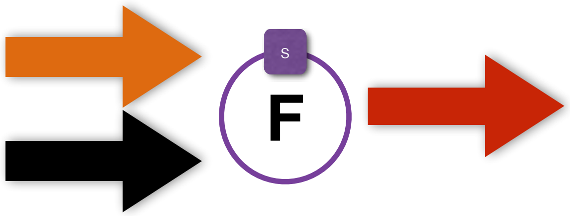


The Fire Alarm

The Fire Alarm



The Fire Alarm

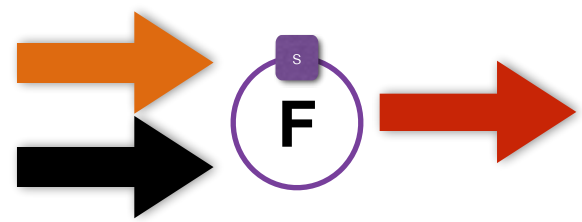


T : avgTemp>40

T : avgTemp<40

S : Smoke

The Fire Alarm



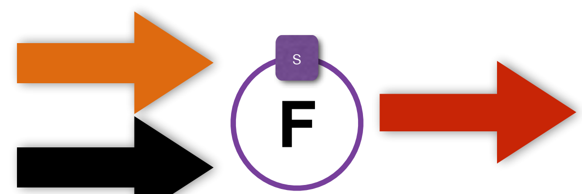
T : avgTemp>40

T : avgTemp<40

S : Smoke

...TTTSTTTSTTTT....

The Fire Alarm

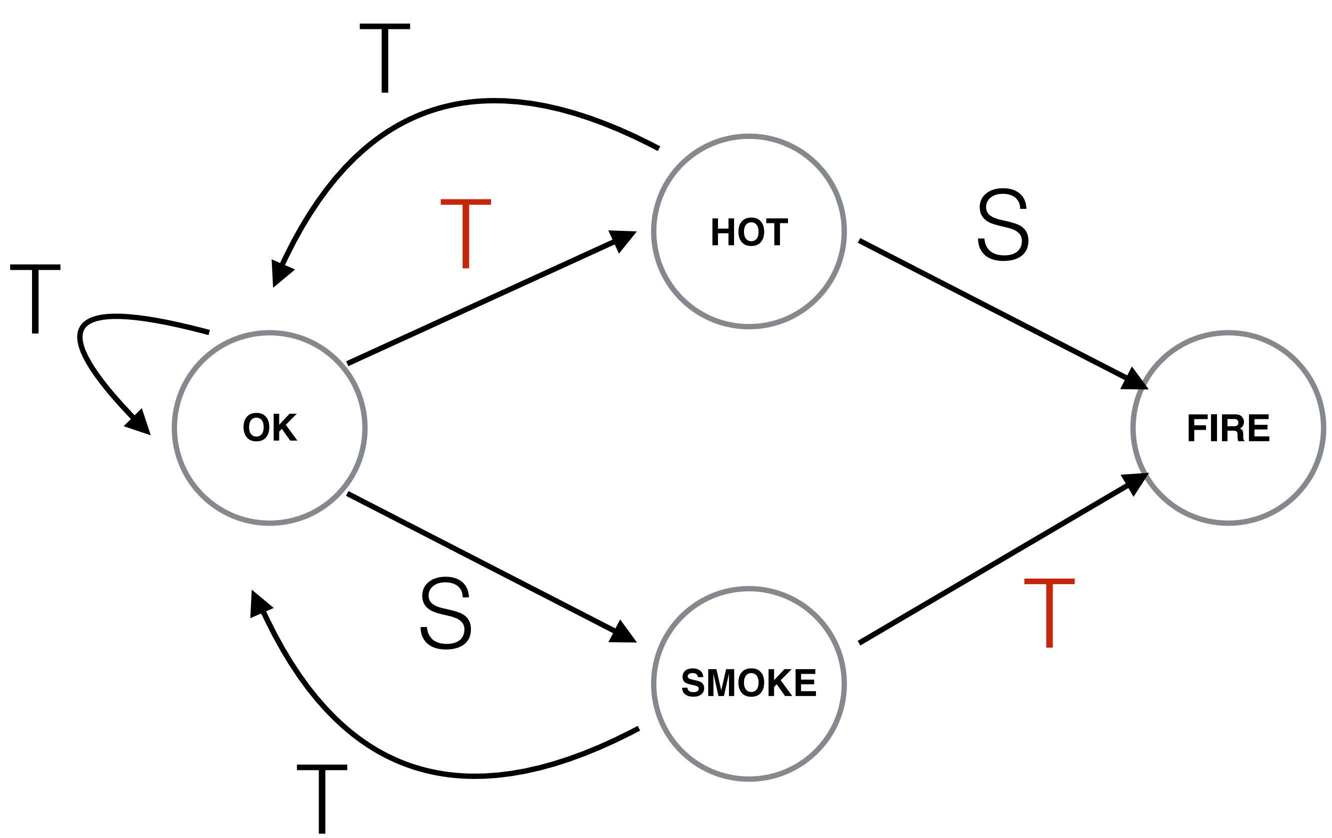


T : avgTemp>40

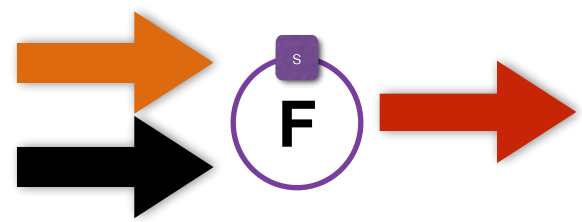
T : avgTemp<40

S : Smoke

…TTTSTTSTTTT….



The Fire Alarm

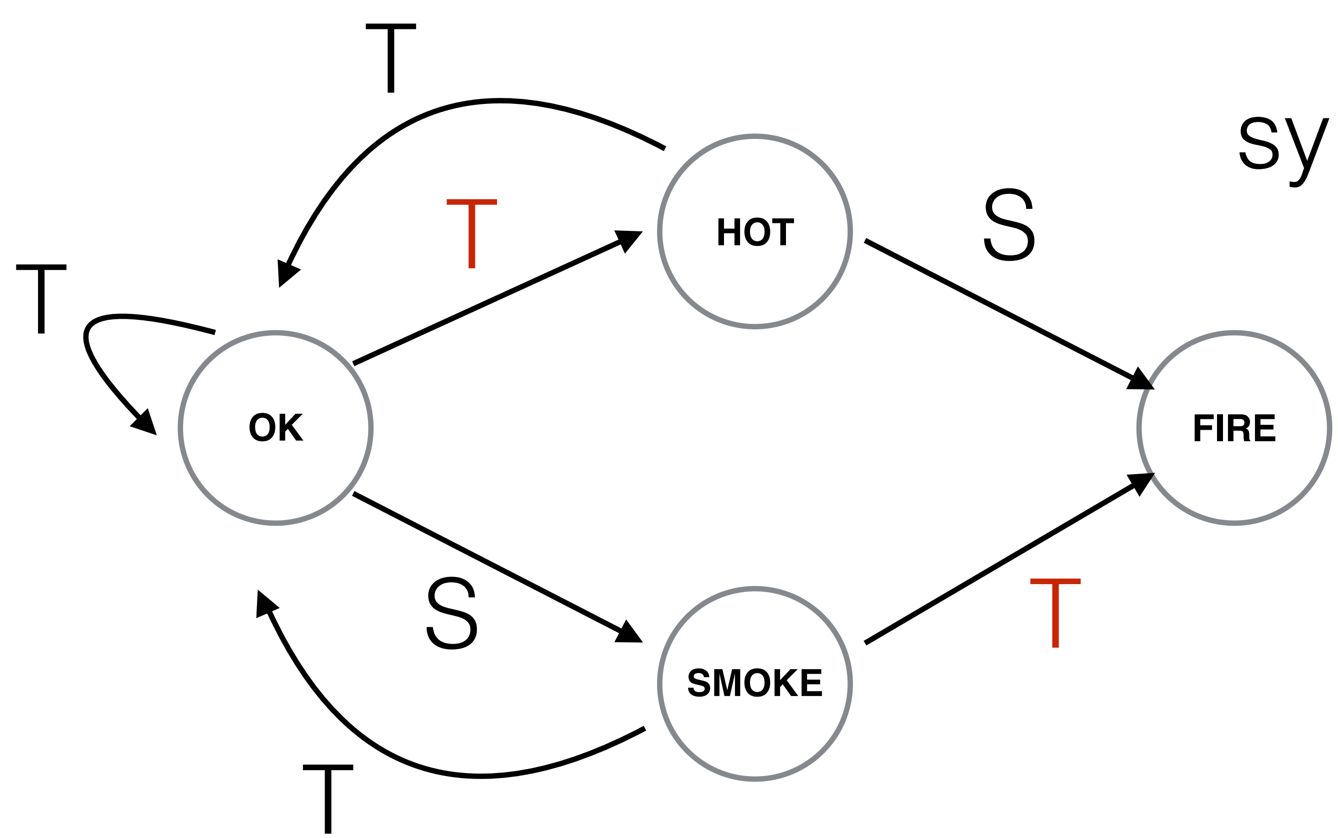


T : avgTemp>40

T : avgTemp<40

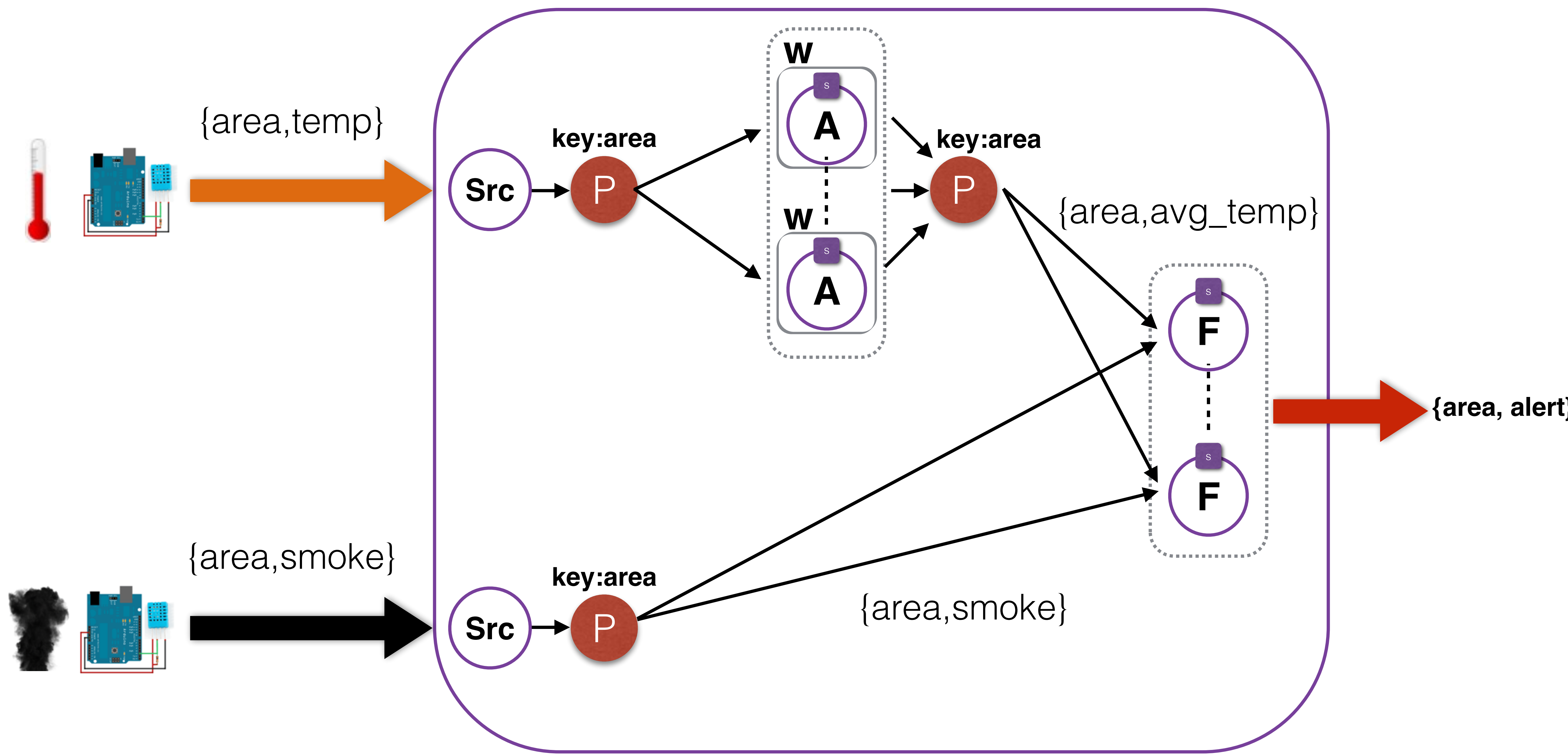
S : Smoke

...TTTSTTSTTTT....



synopsis= 1 state

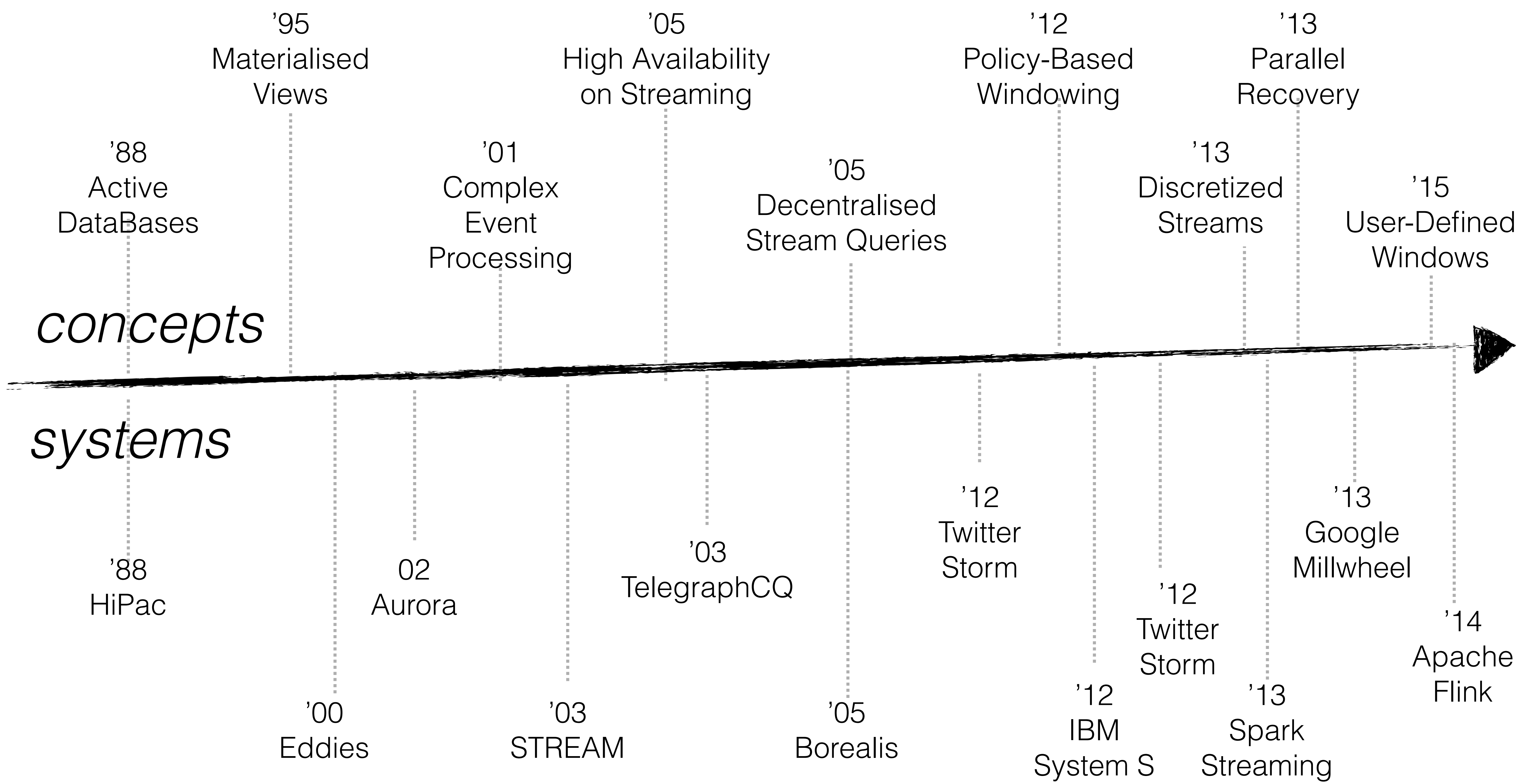
Putting Everything Together



Systems: The Big Picture

Proprietary	**Open Source**
Google DataFlow	Flink
IBM Infosphere	Samza
Microsoft Azure	Spark
	Storm

Evolution


'95 Materialised Views
'05 High Availability on Streaming
'12 Policy-Based Windowing
'13 Parallel Recovery
'88 Active DataBases
'01 Complex Event Processing
'05 Decentralised Stream Queries
'13 Discretized Streams
'15 User-Defined Windows
concepts
systems
'88 HiPac
'00 Eddies
02 Aurora
'03 STREAM
'03 TelegraphCQ
'05 Borealis
'12 Twitter Storm
'12 IBM System S
'12 Twitter Storm
'13 Google Millwheel
'13 Spark Streaming
'14 Apache Flink

Programming Models

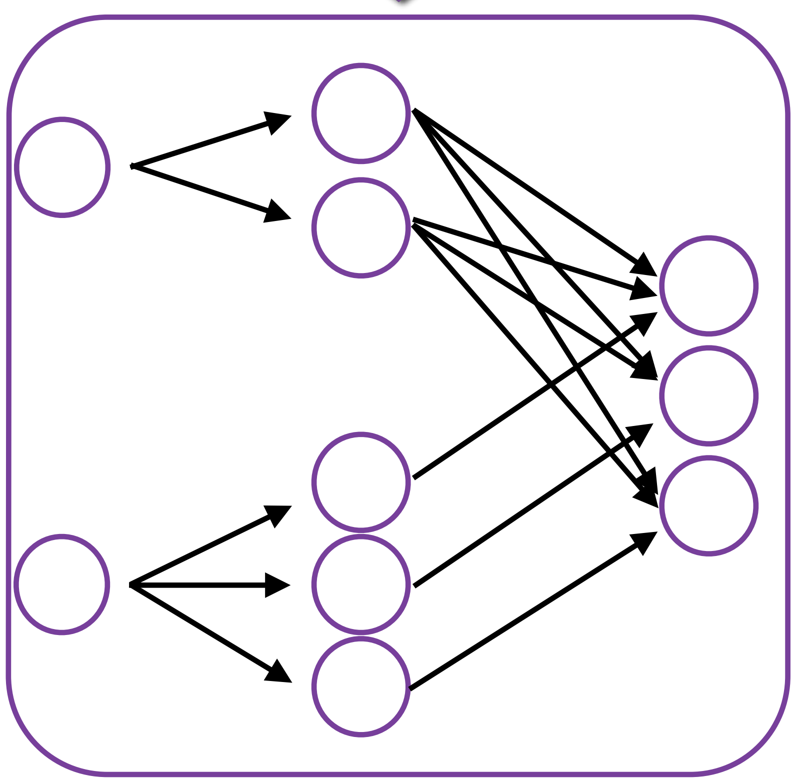
Compositional **Declarative**

- Offer basic building blocks for composing custom operators and topologies
- Advanced behaviour such as windowing is often missing
- Custom Optimisation

- Expose a high-level API
- Operators are higher order functions on abstract data stream types
- Advanced behaviour such as windowing is supported
- Self-Optimisation

Programming Model Types

DStream, DataStream, PCollection…



- Transformations abstract operator details
- Suitable for engineers and data analysts

- Direct access to the execution graph / topology
- Suitable for engineers

Standing Queries with Apache Storm

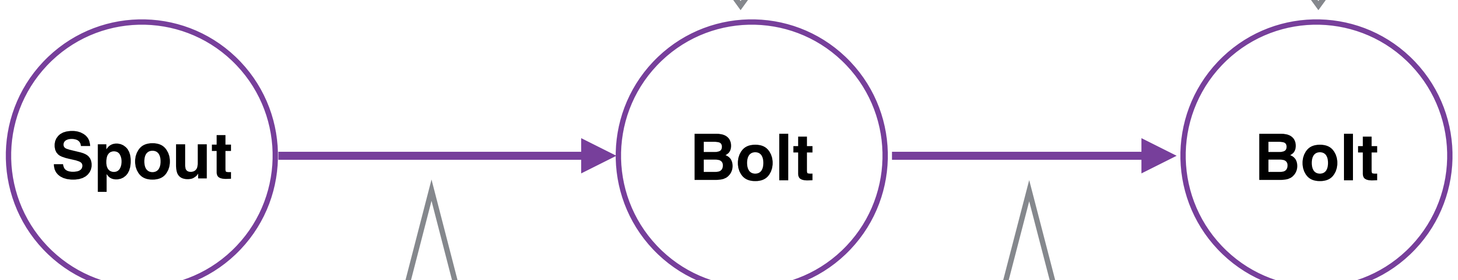
- Step1: Implement input (**Spouts**) and intermediate operators (**Bolts**)
- Step 2: Construct a **Topology** by combining operators

Spouts are the topology sources

The listen to data feeds

Bolts represent all intermediate computation vertices of the topology

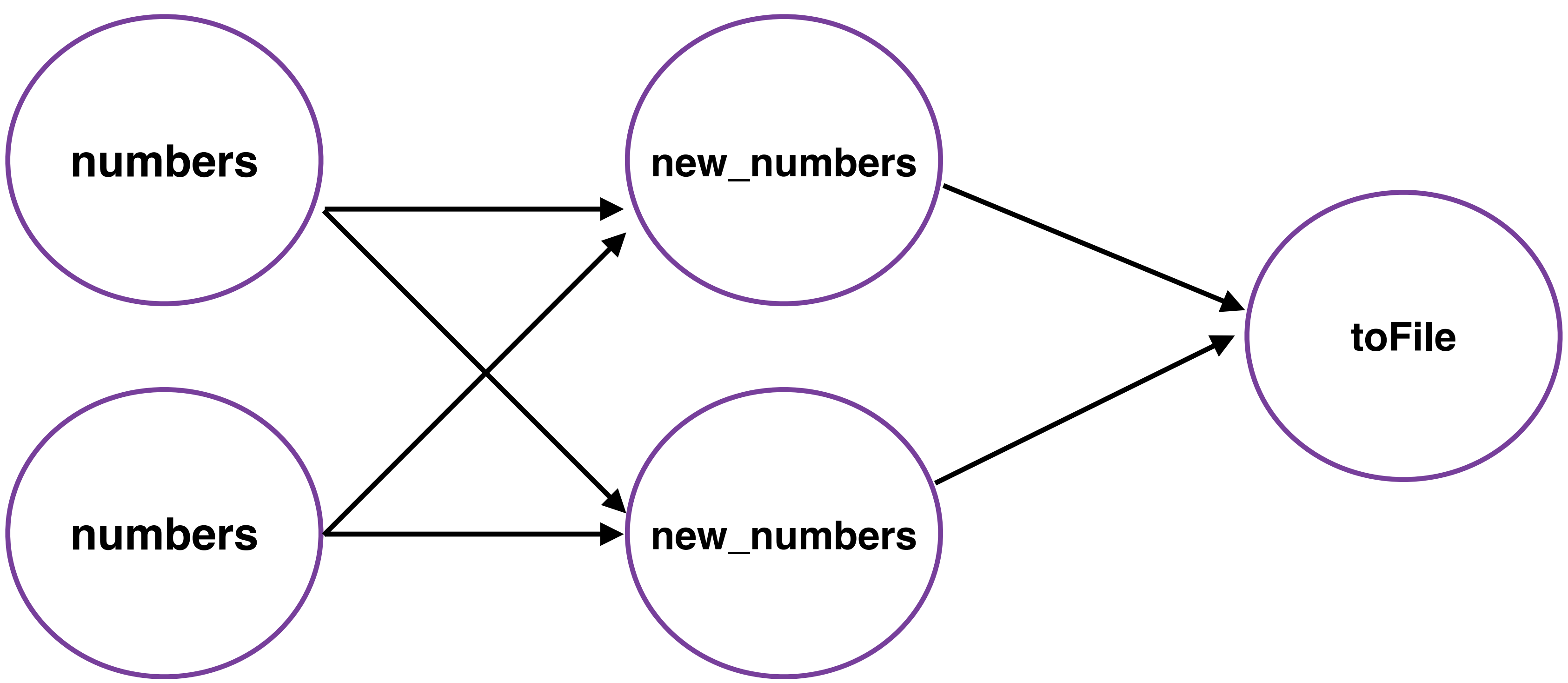
They do arbitrary data manipulation



Each operator can emit/subscribe to **Streams** (*computation results*)

Example: Topology Definition

```
TopologyBuilder builder = new TopologyBuilder()
builder.setSpout("numbers" new NumberGenerator(), 2);
builder.setBolt("new_numbers", new DoubleAndTripleBolt(), 2)
  .shuffleGrouping("numbers");
builder.setBolt("toFile", new DumpToFileBolt(), 1);
  .allGrouping("new_numbers");
```



Standing Queries with Apache Flink

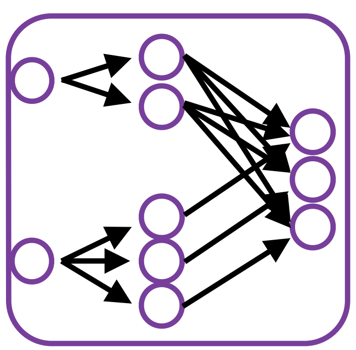
Streaming Program

Flink Client

```
val lines: DataStream[String] = env.fromSocketStream(...)
lines.flatMap {line => line.split(" ")
                              .map(word => Word(word,1))}
      .window(Time.of(5,SECONDS)).every(Time.of(1,SECONDS))
      .groupBy("word").sum("frequency")
      .print()
```

Flink Job Graph Builder/Optimiser

- Operator fusion
- Window Pre-aggregates



Flink Runtime

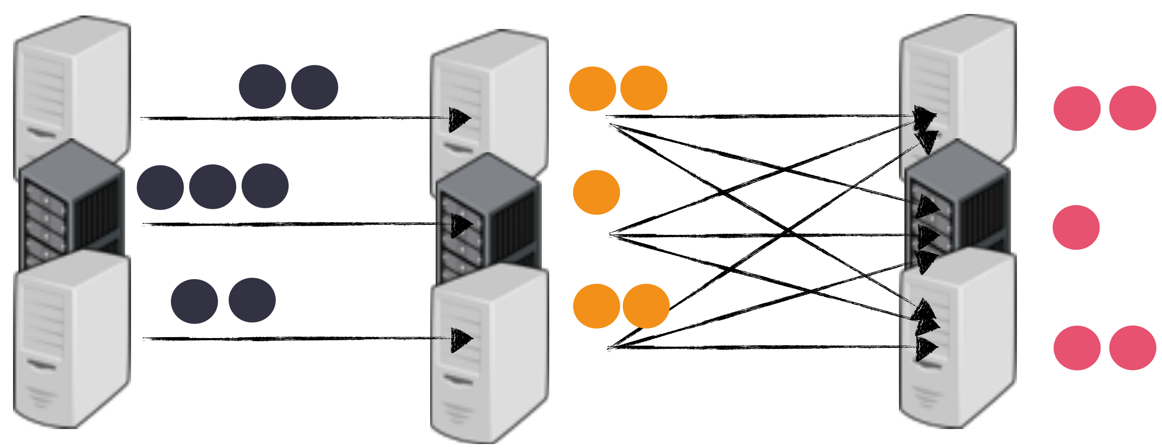
- Deploy Long Running Tasks
- Monitor Execution

Distributed Stream Execution Paradigms

1) Real Streaming (Distributed Data Flow)

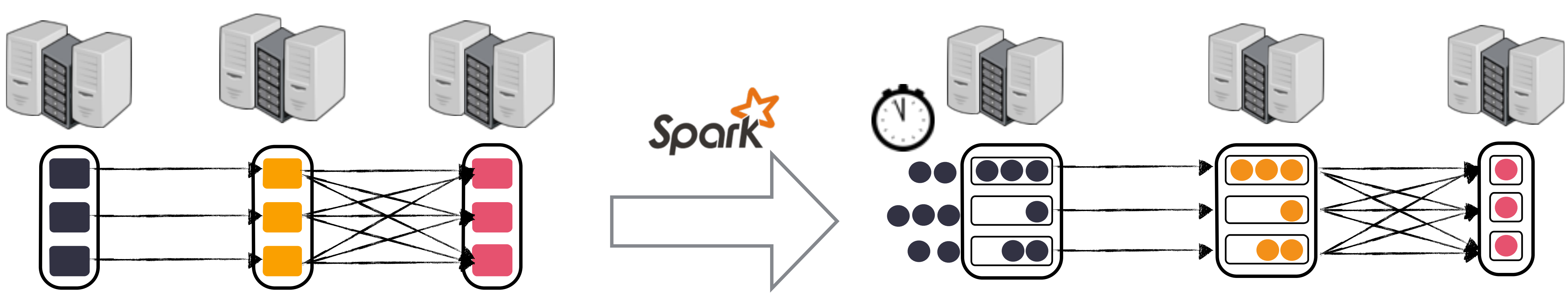
samza

LONG-LIVED TASK EXECUTION



STATE IS KEPT INSIDE TASKS

2) Batched Execution



(Hadoop, Spark)

(Spark Streaming)

Windows in Action

Spark

```
val ssc = new StreamingContext(conf, Seconds(1))

dstream.window(SECONDS(10), SECONDS)(2))
```

range

```
stream.window(Count.of(100))
.every(Time.of(10, SECONDS))
```

slide

- DStreams are already partitioned in time windows
- Only time windows supported

- Windows decomposed into policies
- Policies can be user-defined too

Windows on Storm?

```
@Override
public void execute(Tuple tuple) {
  if (TupleHelpers.isTickTuple(tuple)) {
    LOG.info("Received tick tuple, triggering emit of current window counts");
    emitCurrentWindowCounts();
  }
  else {
    countObjAndAck(tuple);
  }
}

private void emitCurrentWindowCounts() {
  Map<Object, Long> counts = counter.getCountsThenAdvanceWindow();
  ...
  emit(counts, actualWindowLengthInSeconds);
}

private void emit(Map<Object, Long> counts) {
  for (Entry<Object, Long> entry : counts.entrySet()) {
    Object obj = entry.getKey();
    Long count = entry.getValue();
    collector.emit(new Values(obj, count));
  }
}

private void countObjAndAck(Tuple tuple) {
  Object obj = tuple.getValue(0);
  counter.incrementCount(obj);
  collector.ack(tuple);
}
```

src-http://www.michael-noll.com/blog/2013/01/18/implementing-real-time-trending-topics-in-storm/

Partitioning in Action

forward()
shuffle()
broadcast()
keyBy()

partitionCustom()

shuffleGrouping()
allGrouping()
fieldsGrouping()

customGrouping()

full control

repartition(num)
reduceByKey()
updateStateByKey()

no fine-grained control

Synopses in Action

implementing a rolling max per key

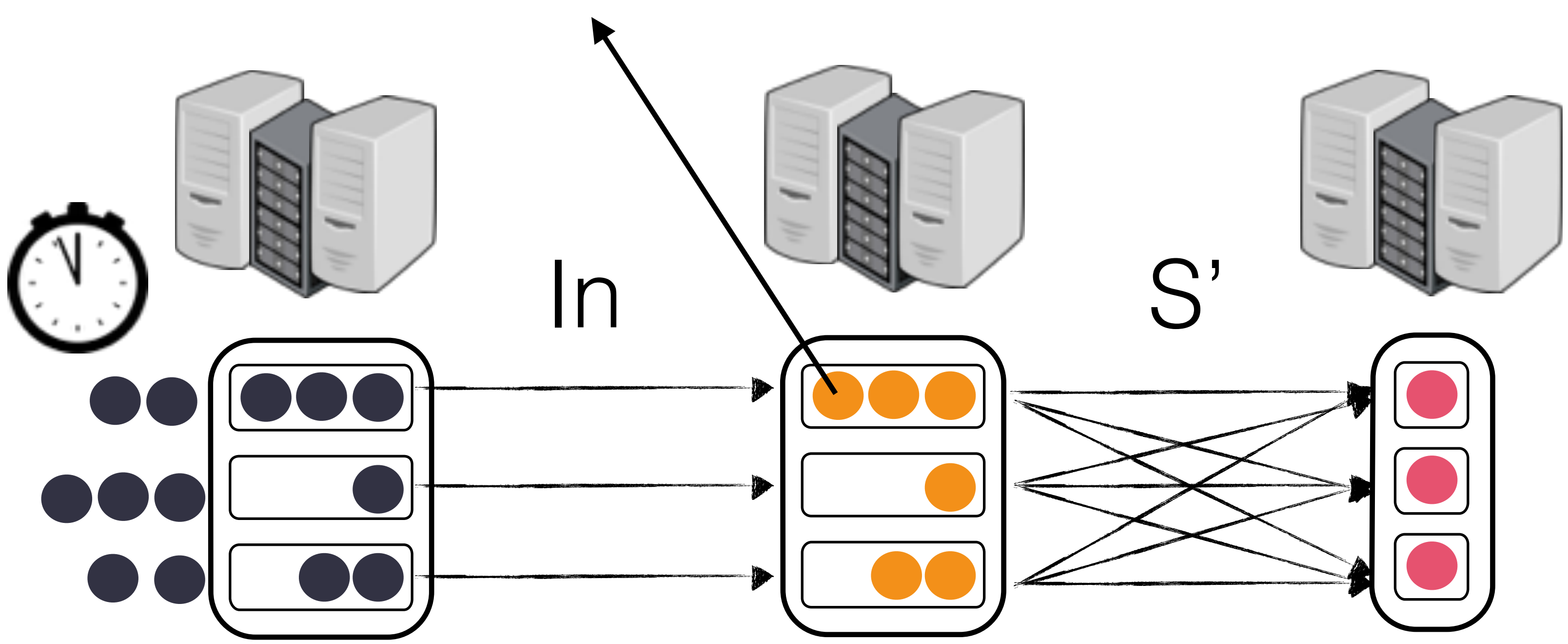
```java
public static class MaxPerKey extends BaseRichBolt {
    private int Map<Integer,Integer> maxVals = ...
    private OutputCollectorBase _collector;
    @Override
    public void execute(Tuple tuple) {
      int key = tuple.getInteger(1);
      int _max = maxVals.contains(key)
        ? maxVals.get(tuple.getInteger(1)) : 0;
        if(_max < tuple.getInteger(2)){
          _max = tuple.getInteger(2);
          maxVals.put(tuple.getInteger(1), _max)
          _collector.emit(tuple, _max);
        }
    }
}
```

```scala
stream.keyBy("key").flatMapWithState(
  (in, state: Option[Int]) => state match {
      case Some(val) => if (val < in.val) (List(in), in.val)
        else (List(),Some(state))
      case None => (List(in.val), Some(in.val))
  })
```

State in Spark?

- Streams are partitioned into small batches
- There is practically no state kept in workers (stateless)
- How do we keep state??

dstream.updateStateByKey(…) *put new states in output RDD*

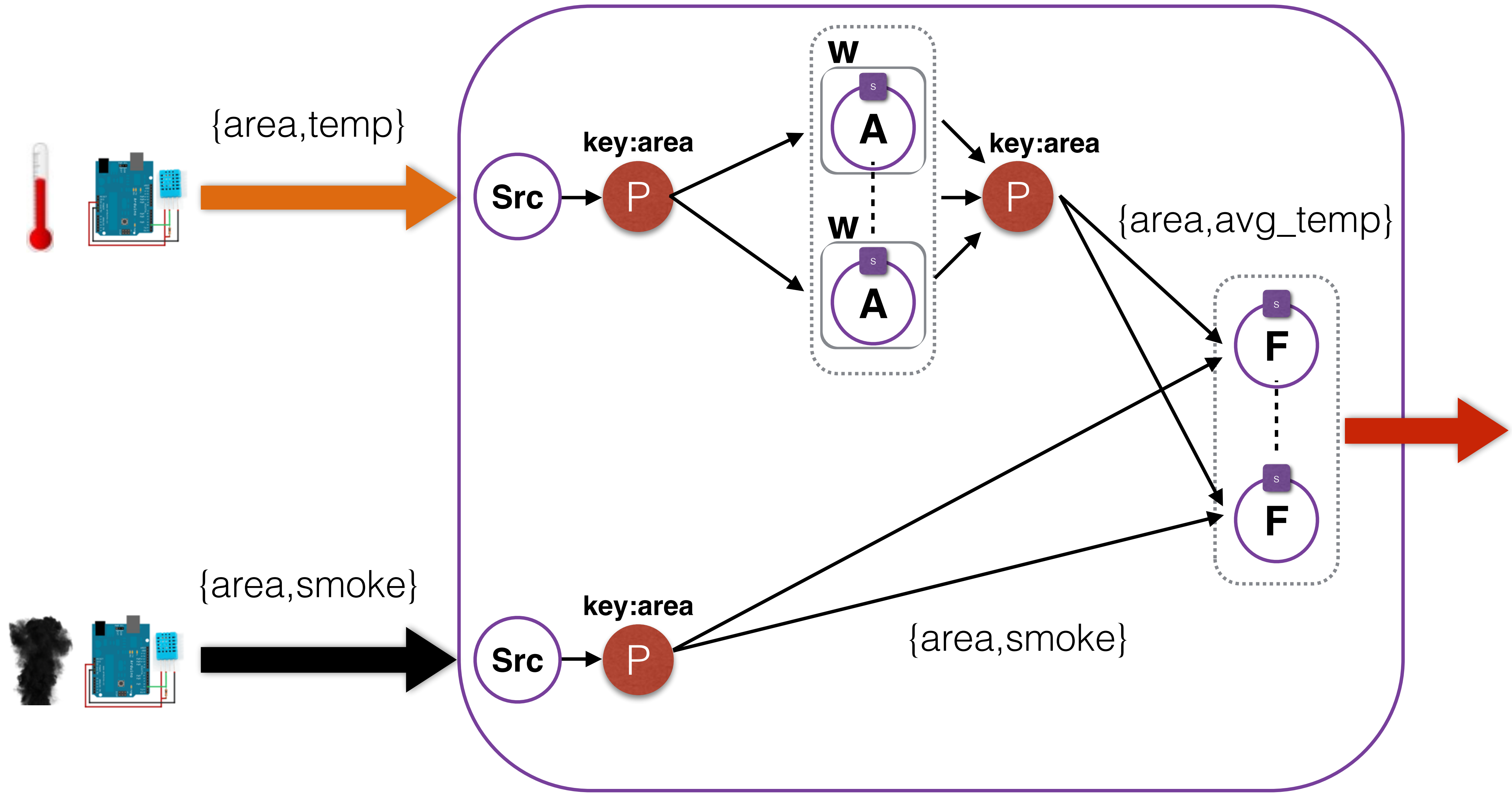


(Spark Streaming)

Implementing the alarm in Flink

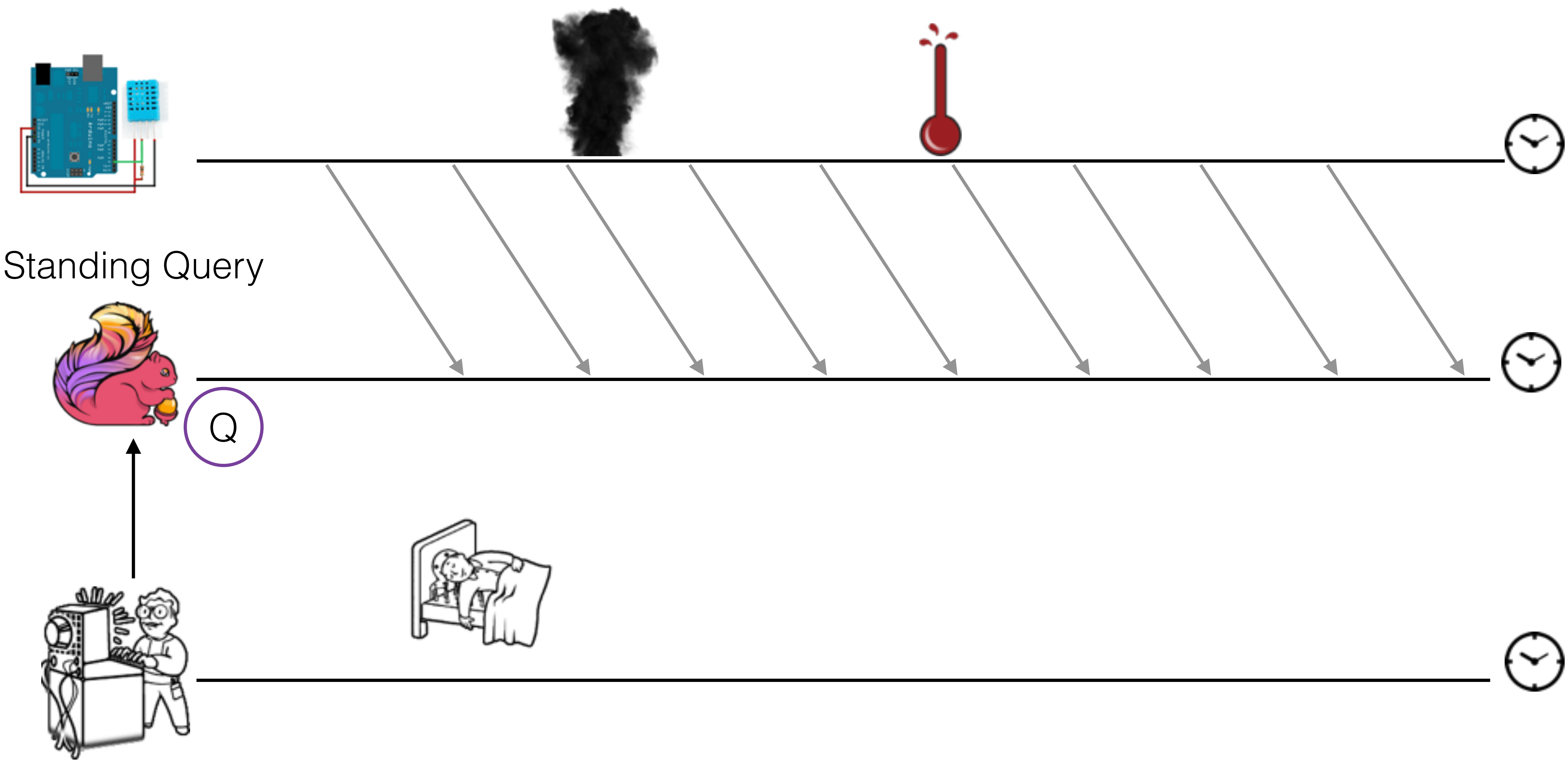
```
val temperatures = env.socketTextStream(...).keyBy("area")
val smokes = env.socketTextStream(...).keyBy("area")

val avgTemp = temperatures
  .window(Time.of(60,  SECONDS))
  .every(Time.of(5,  SECONDS))
  .mapWindow(_avgTemp).flatten().keyBy("area")

avgTemp.connect(smokes).flatMap(sm_alarm).print()
```

So everything works

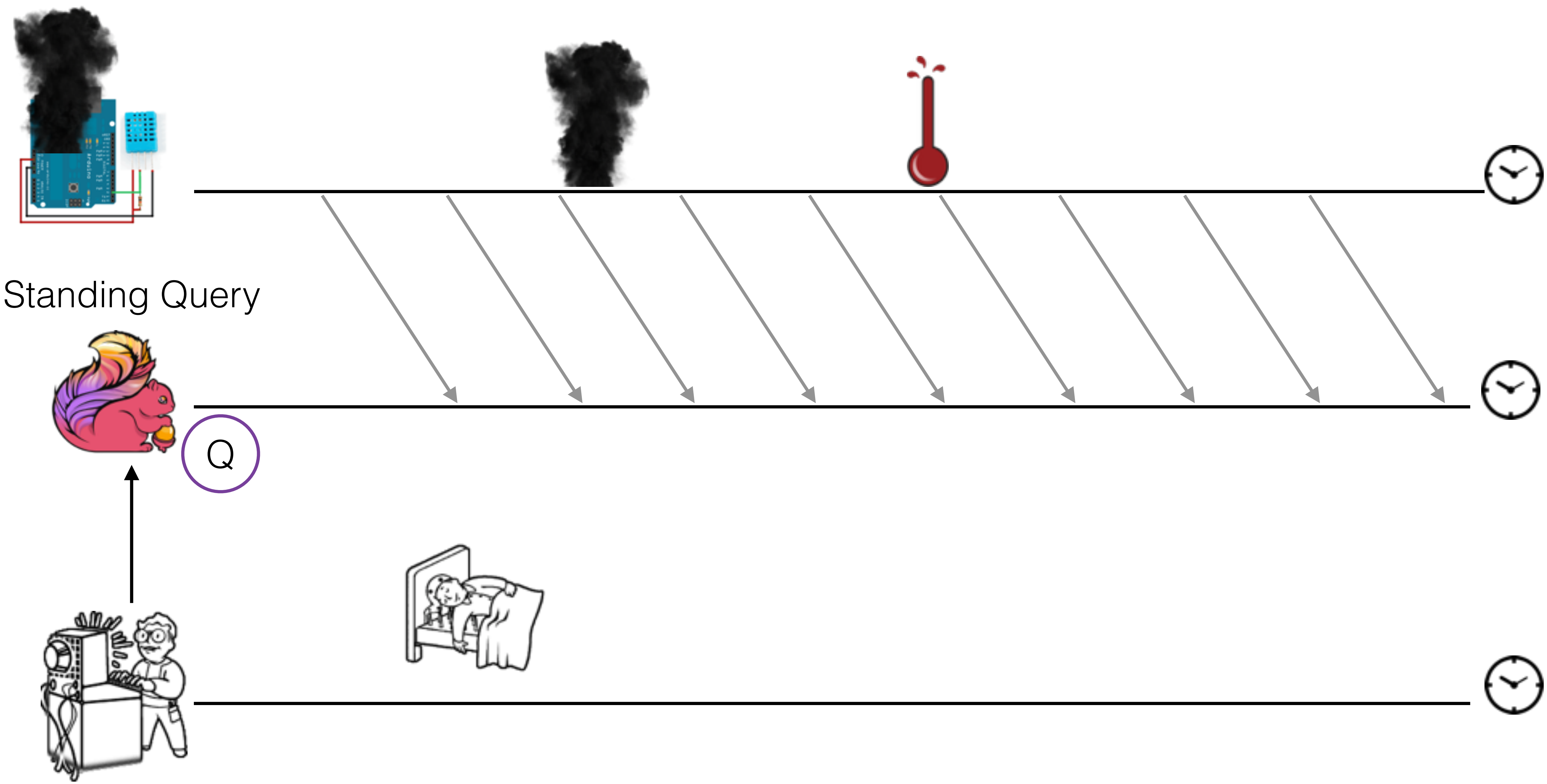


or…

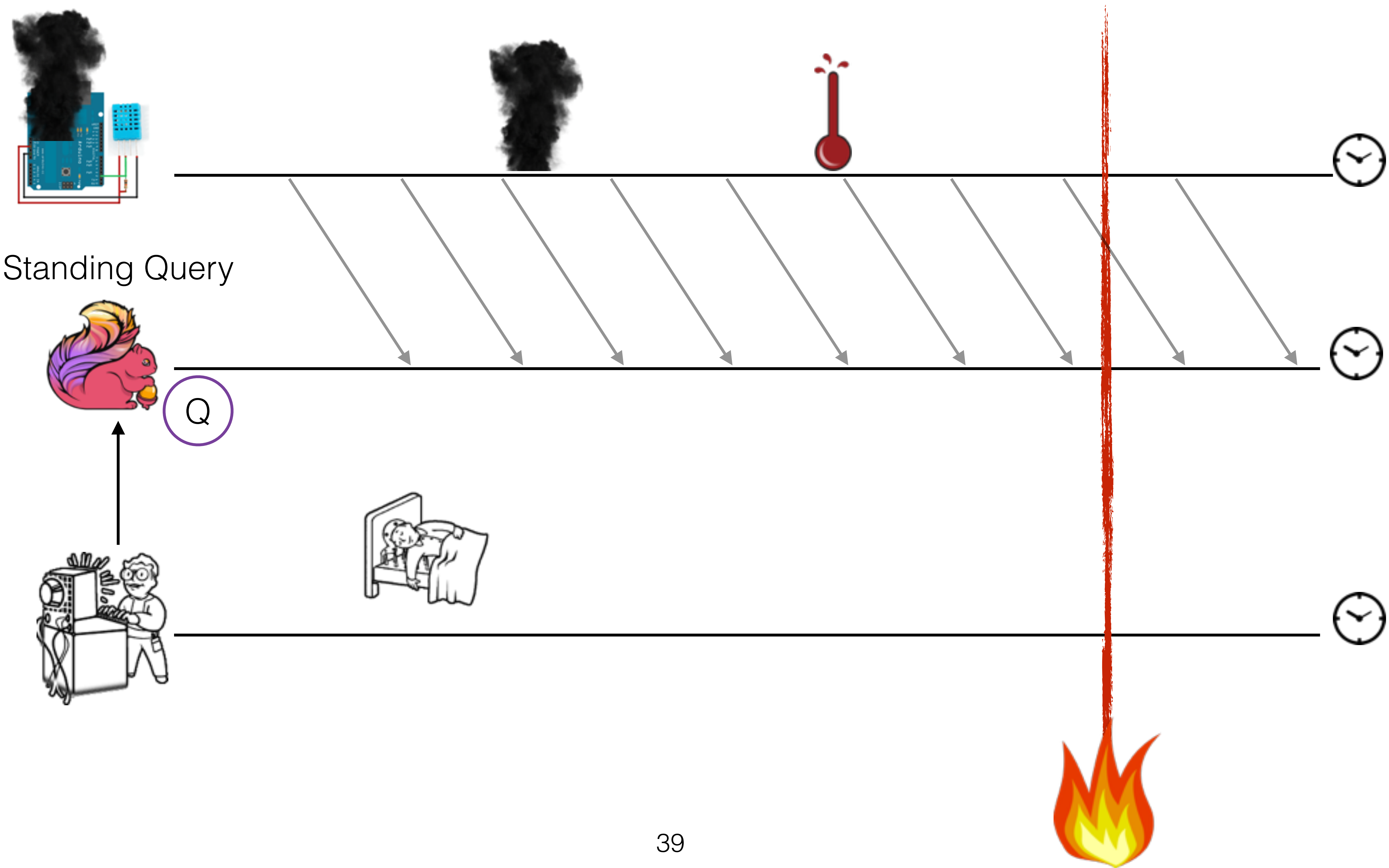
Unreliable Sources



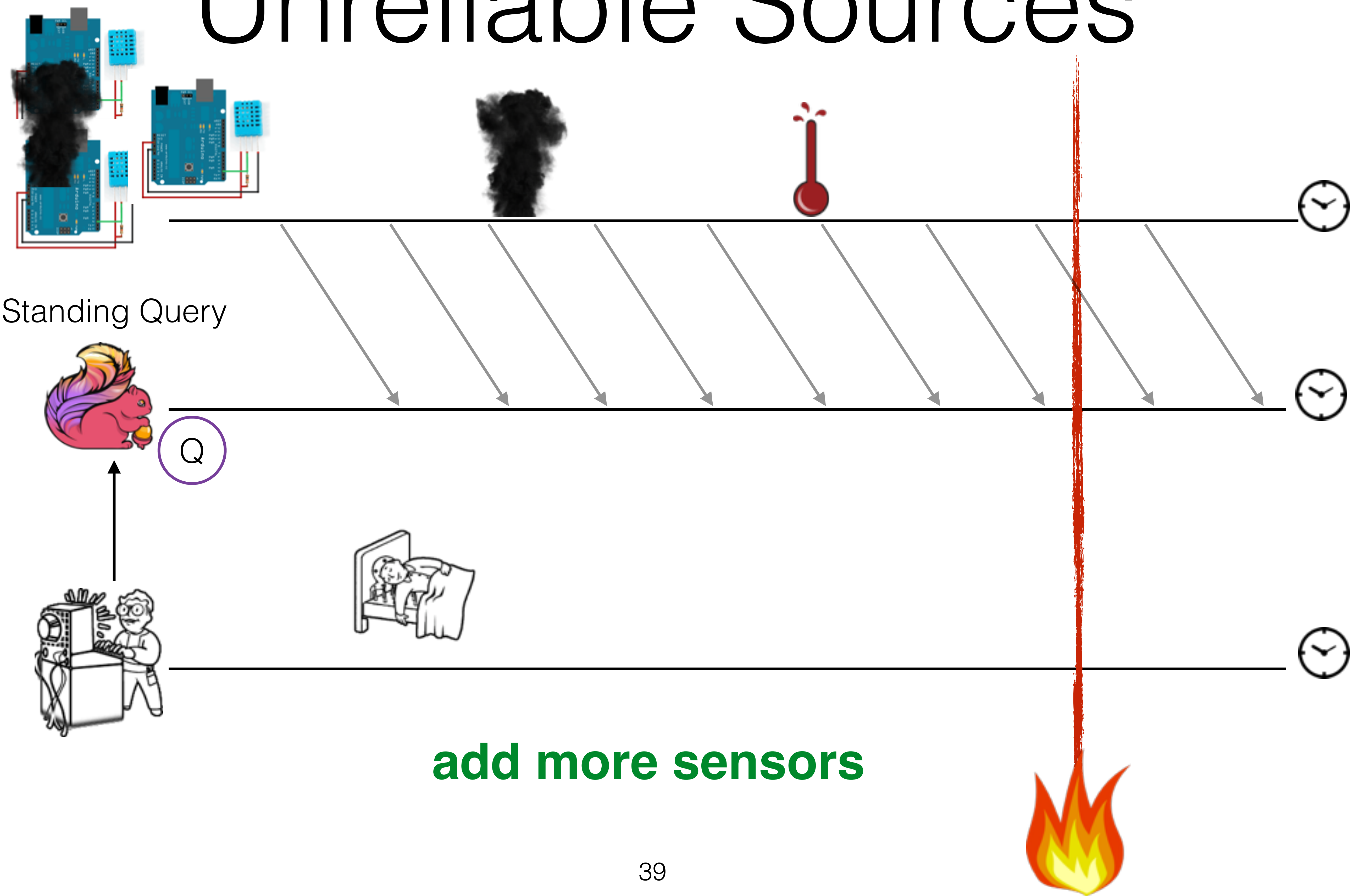
Unreliable Sources



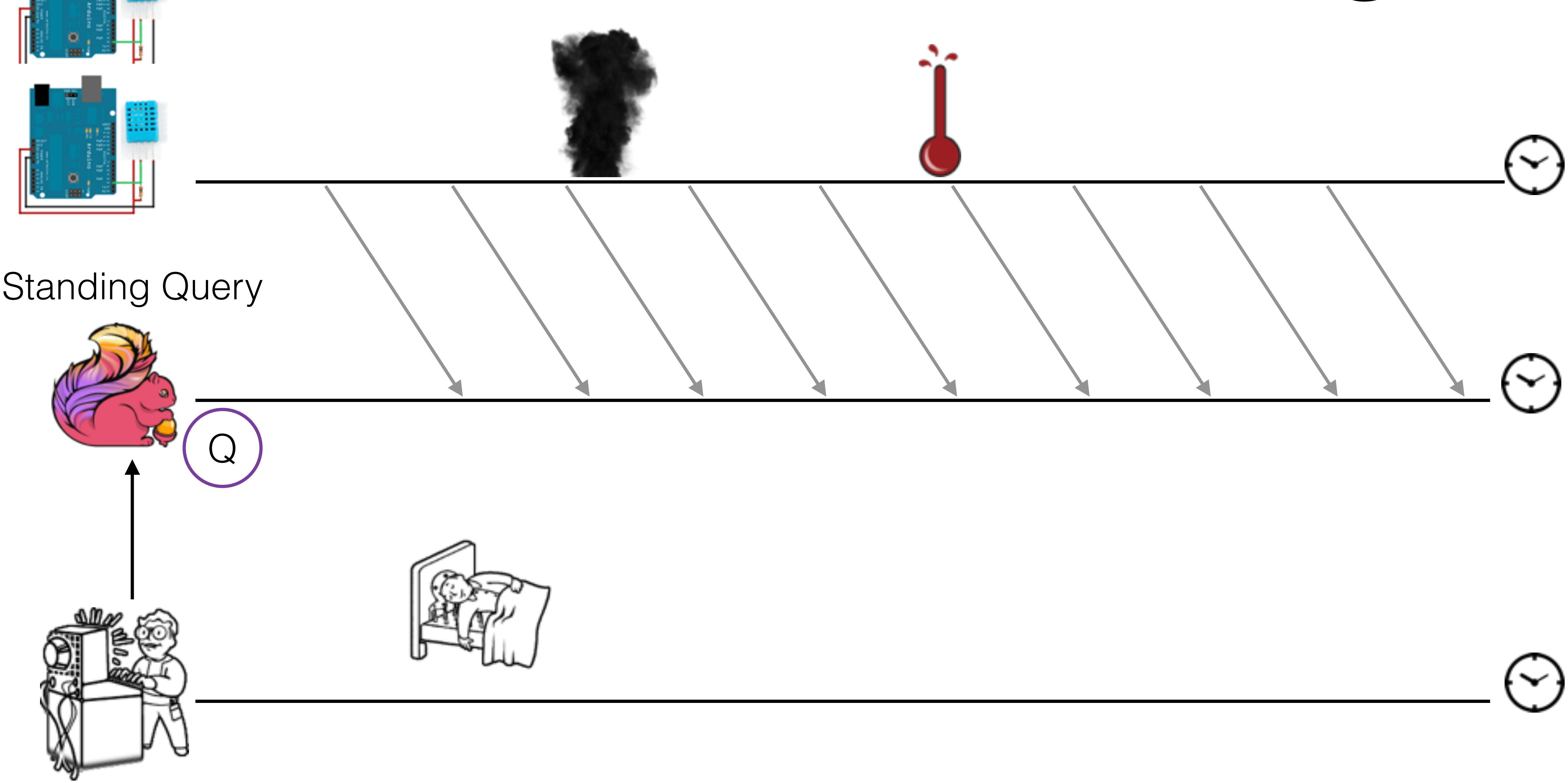
Unreliable Sources



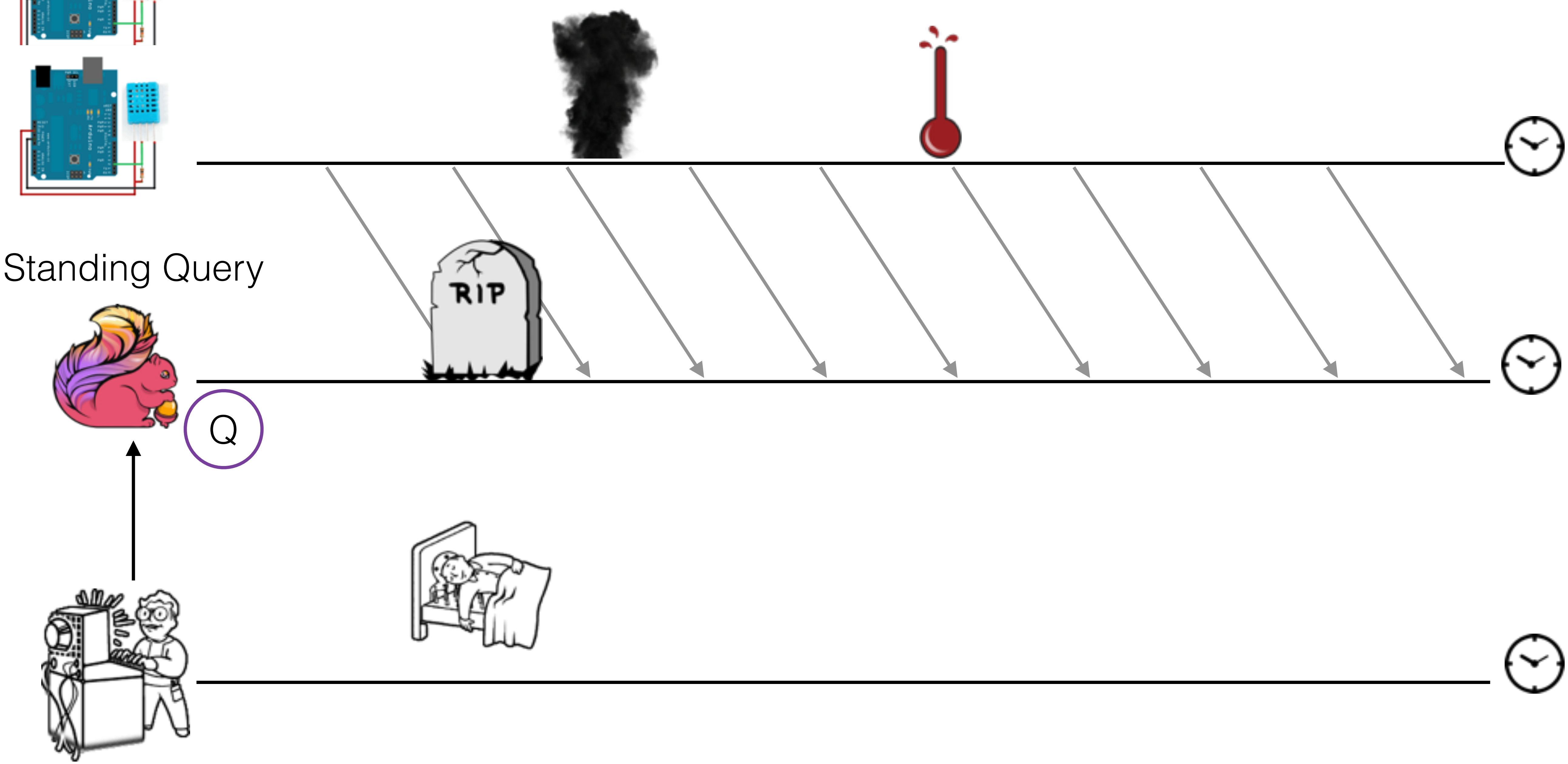
Unreliable Sources



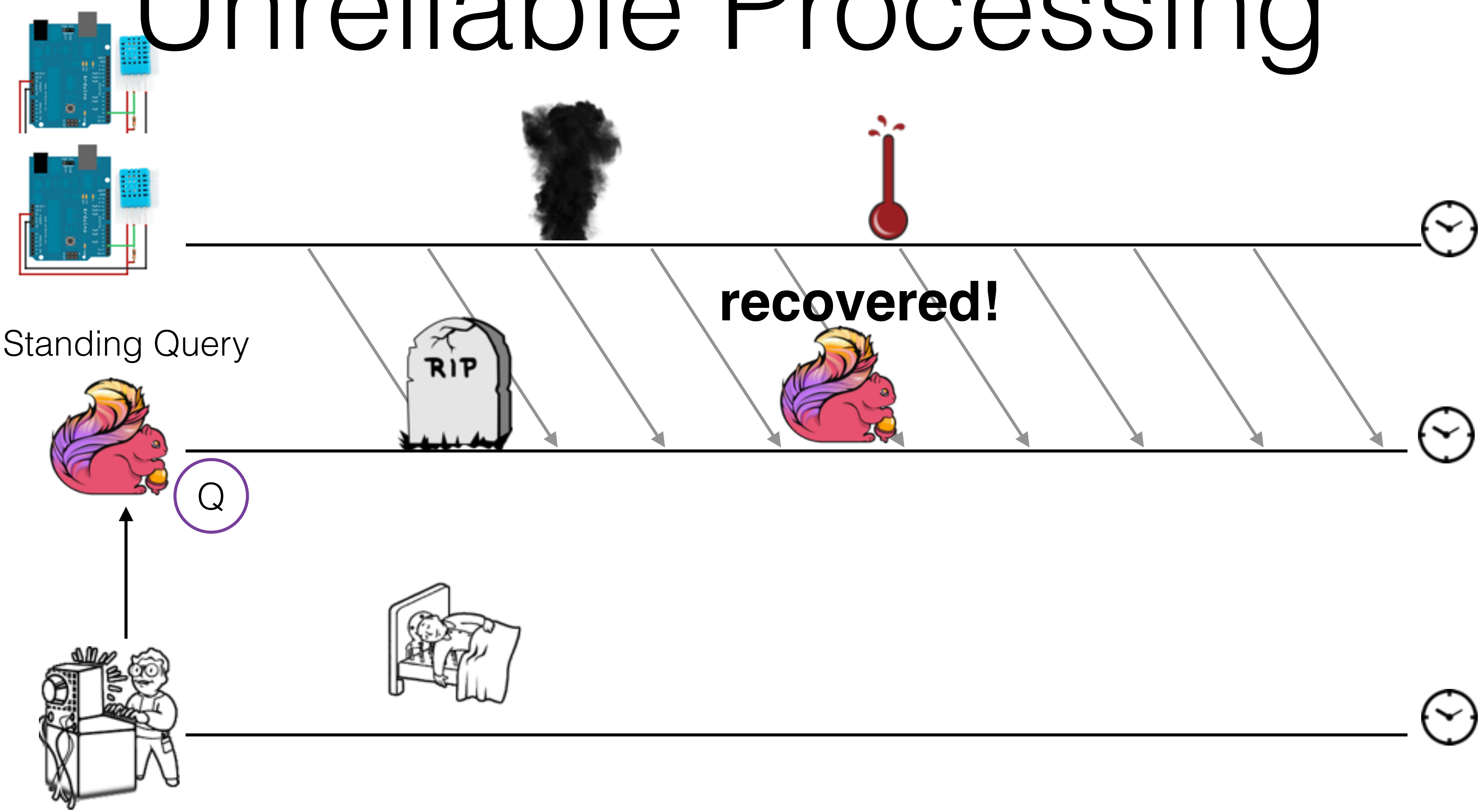
Unreliable Processing



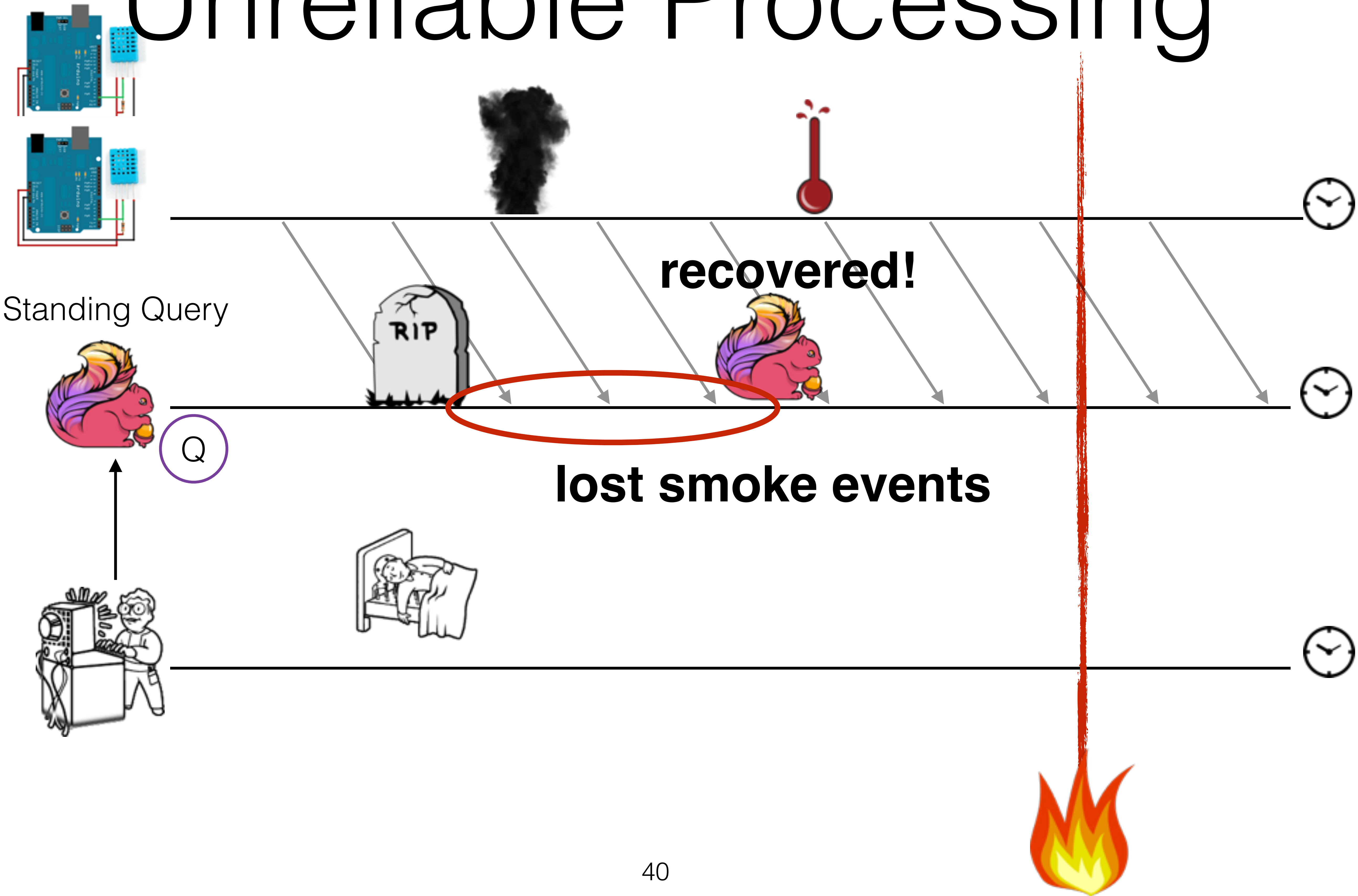
Unreliable Processing



Unreliable Processing



Unreliable Processing



Resilient Brokers

Main Features

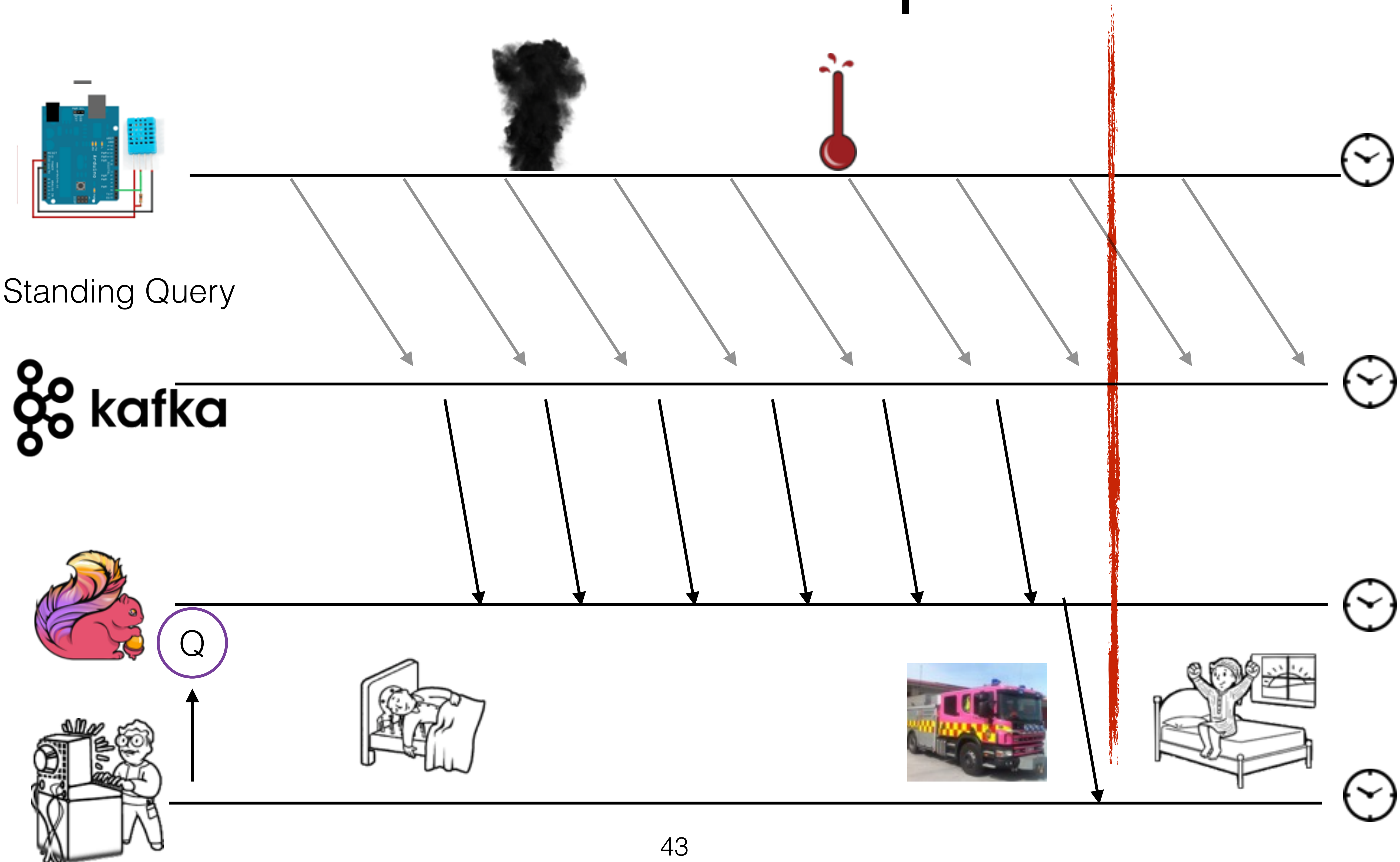
- Topic-based partitioned queues
- Strongly consistent offset mapping to records

Processing Guarantees

- Kafka solves the source consistency problem
- How about the rest of the states of the computation ? (e.g. alert operator state)
- Each system offers different guarantees

	Guarantees	Technique
Storm	at least once	event dependency tracking
Spark	exactly once	source upstream backup
Flink	exactly once	periodic snapshots

Mission Accomplished



Research Topics at KTH/SICS

- Exactly-Once-Output Guarantees
- State management and auto-scaling
- Streaming ML pipelines
- Streaming Graphs