Scaling *Mobile Millennium* with BDAS

Timothy Hunter,
Teodor Moldovan, Matei Zaharia, Samy Merzgui, Justin Ma,
Michael J. Franklin, Pieter Abbeel, Alexandre M. Bayen

UC Berkeley

August 22, 2012

Machine learning at scale

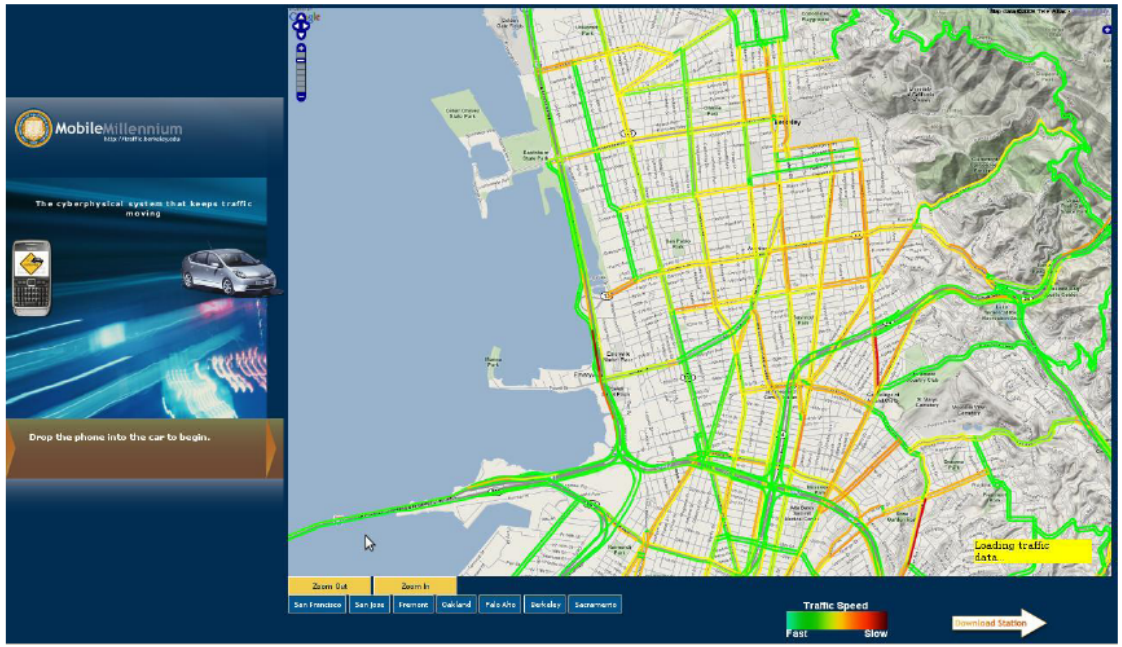
- Combining A, M and P in a real application:
  - Complex models (car traffic estimation)
  - Crowd-sourced data (mobile phones)
  - Computations on the cloud
- How we run Spark inside *Mobile Millennium*

Plan

- Why *car* traffic estimation
- Overview of *Mobile Millennium*
- 2 minutes of applied Machine Learning
- Programming with the Spark framework
- Conclusion: the good, the bad, the not so beautiful

Need for good traffic estimation

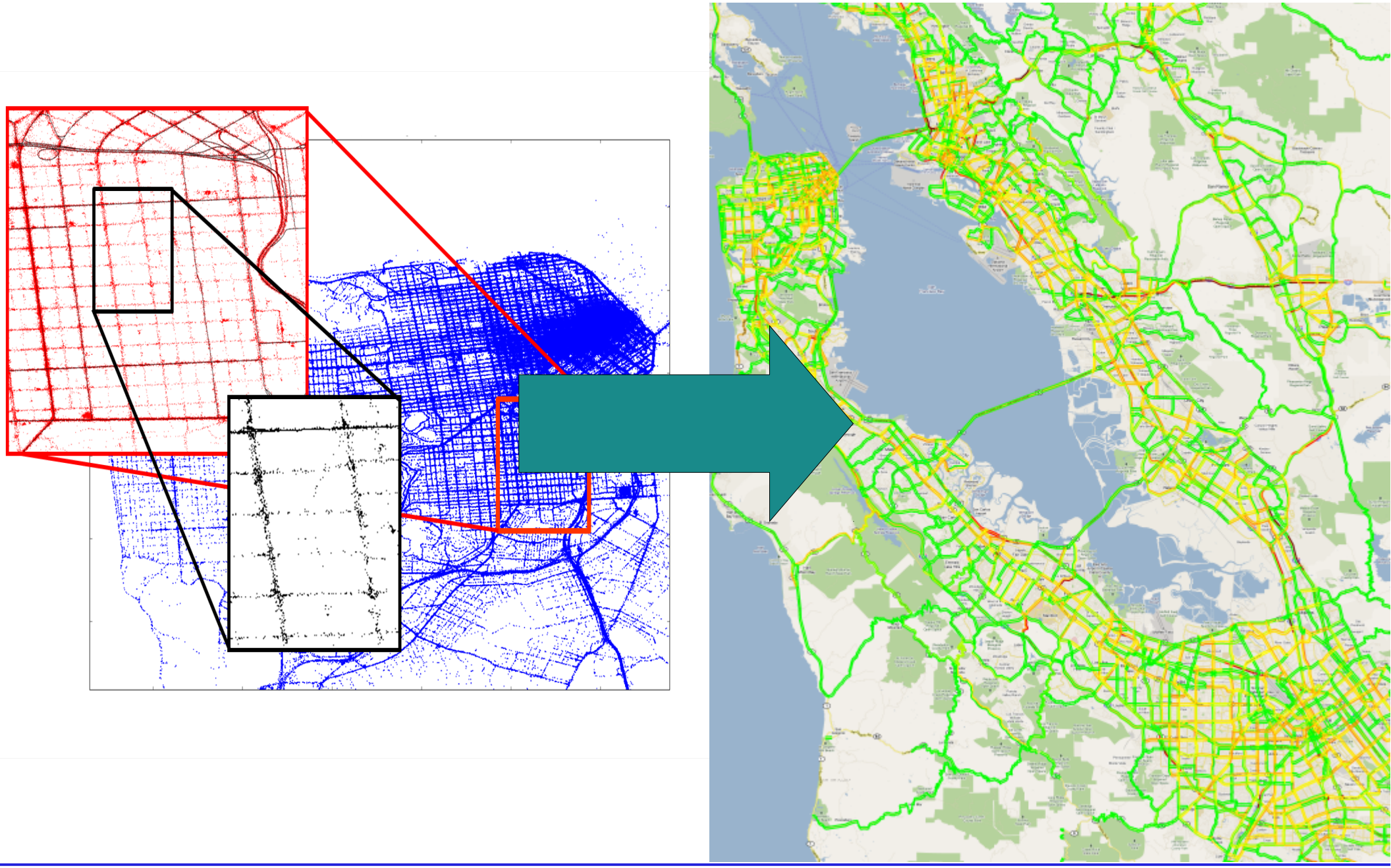
- Traffic congestion affects everyone
- Up-to-date estimation is critical
- Complex for urban streets (*arterial roads*)



Real-time processing of fleet data

- Input: sampled position of taxicabs
- Observed *every minute*

Estimating the travel times

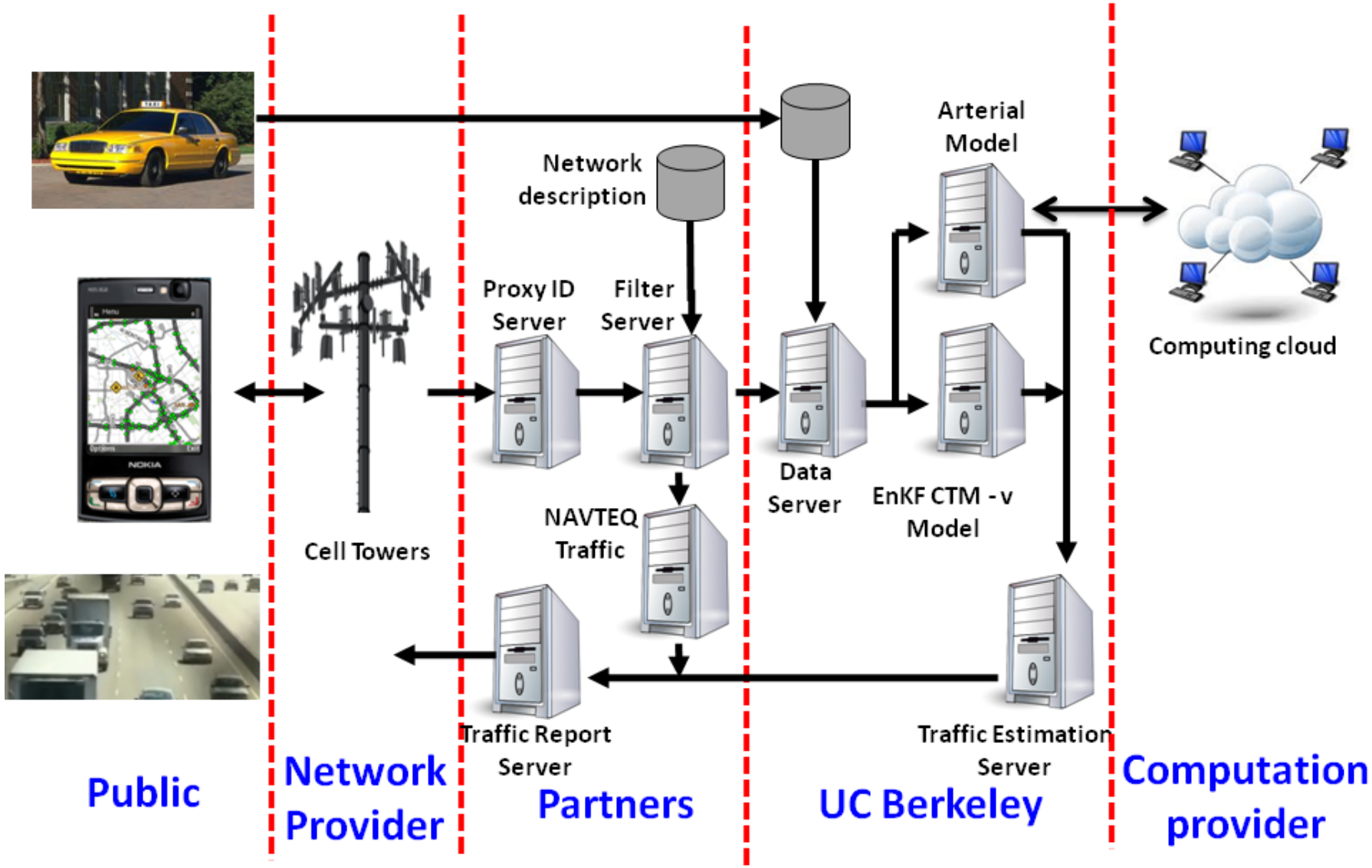


Filtering of fleet data

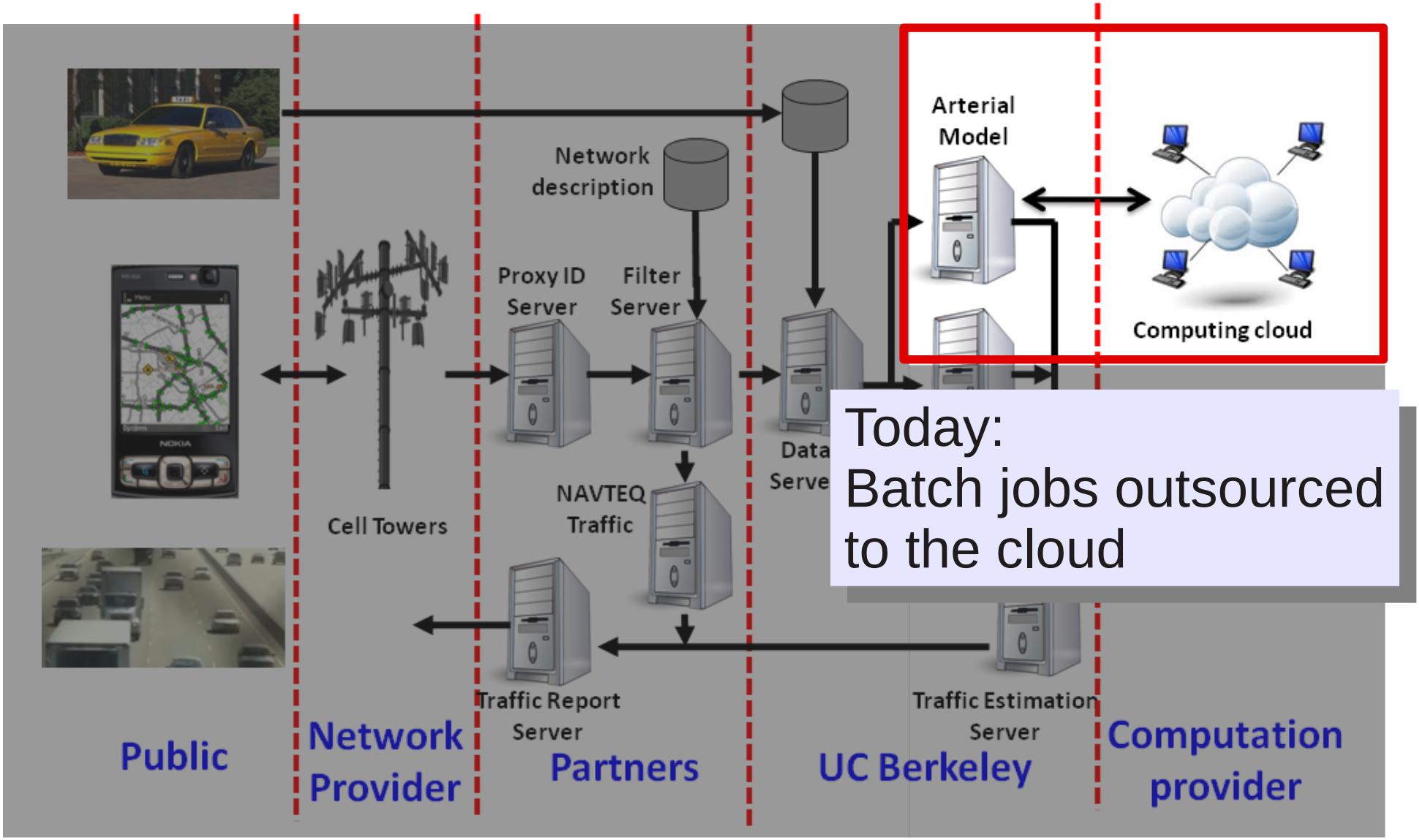
Preprocessing:

- Recovering trajectories from GPS points

- A cyberphysical system for participatory sensing

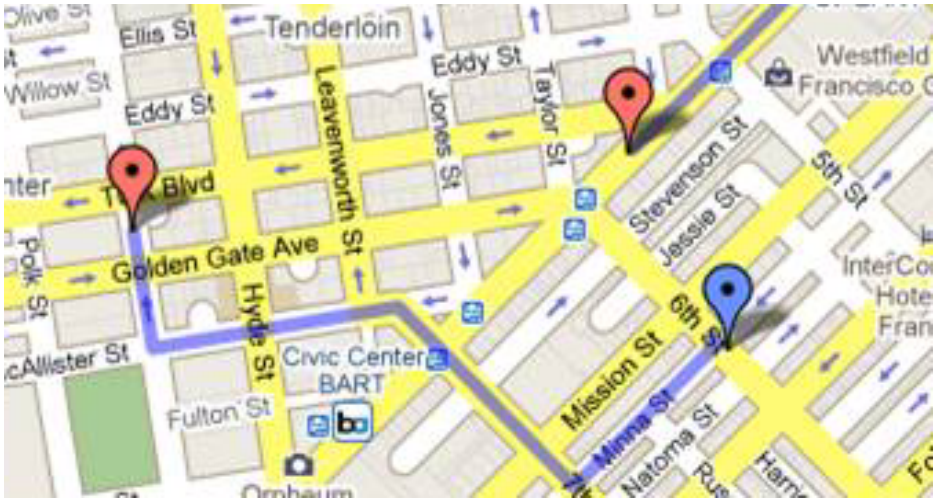


- A cyberphysical system for participatory sensing



Estimation of arterial traffic

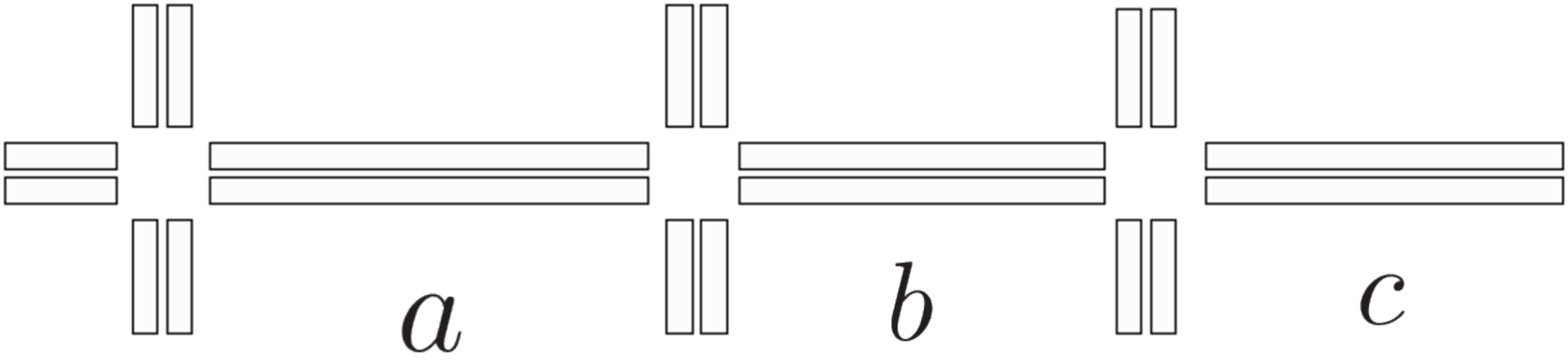
- Input:
  - Pieces of trajectories between GPS points



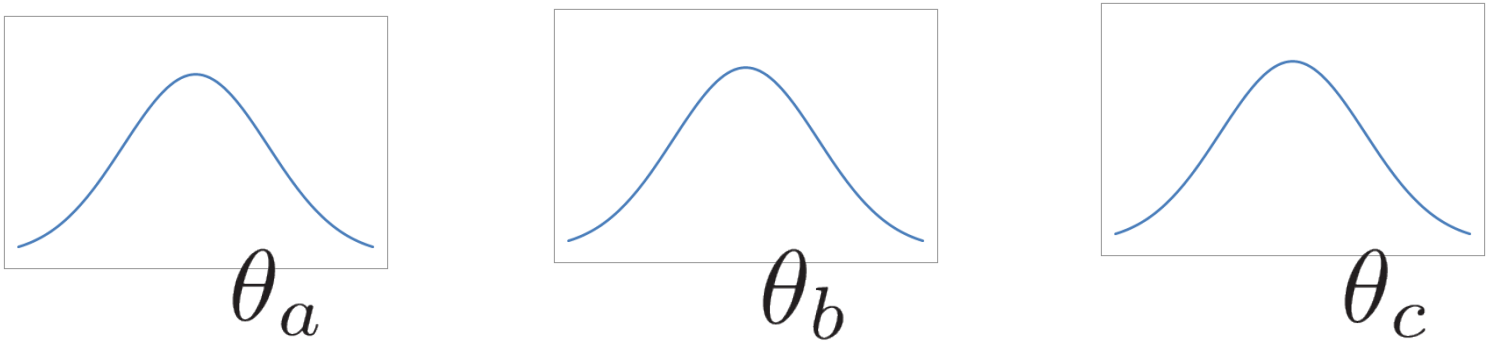
- Output: probability distributions of travel time
  - For each link
  - Parametrized by vector θ (mean and variance of link travel time)

The way things work

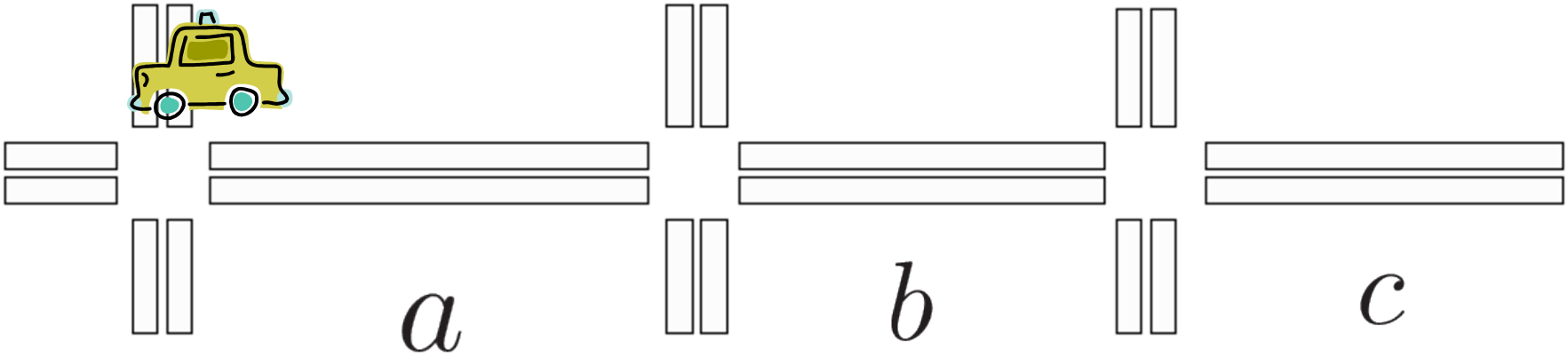
- Example road network



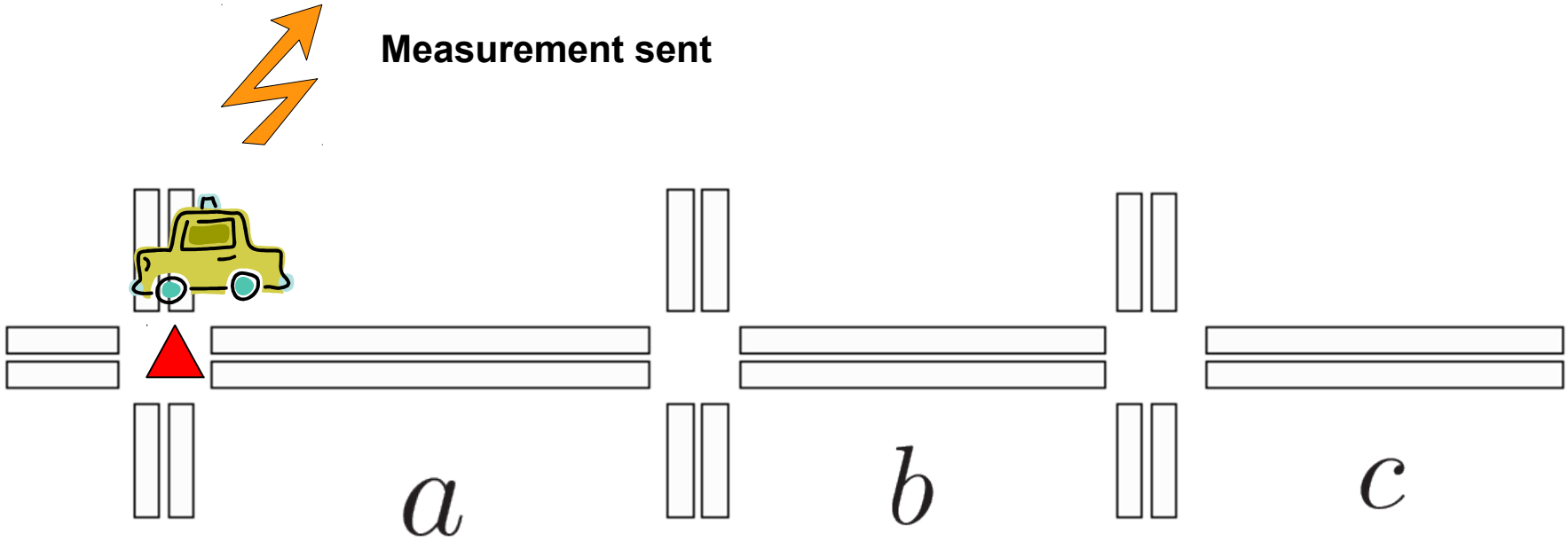
- Associated link travel times:



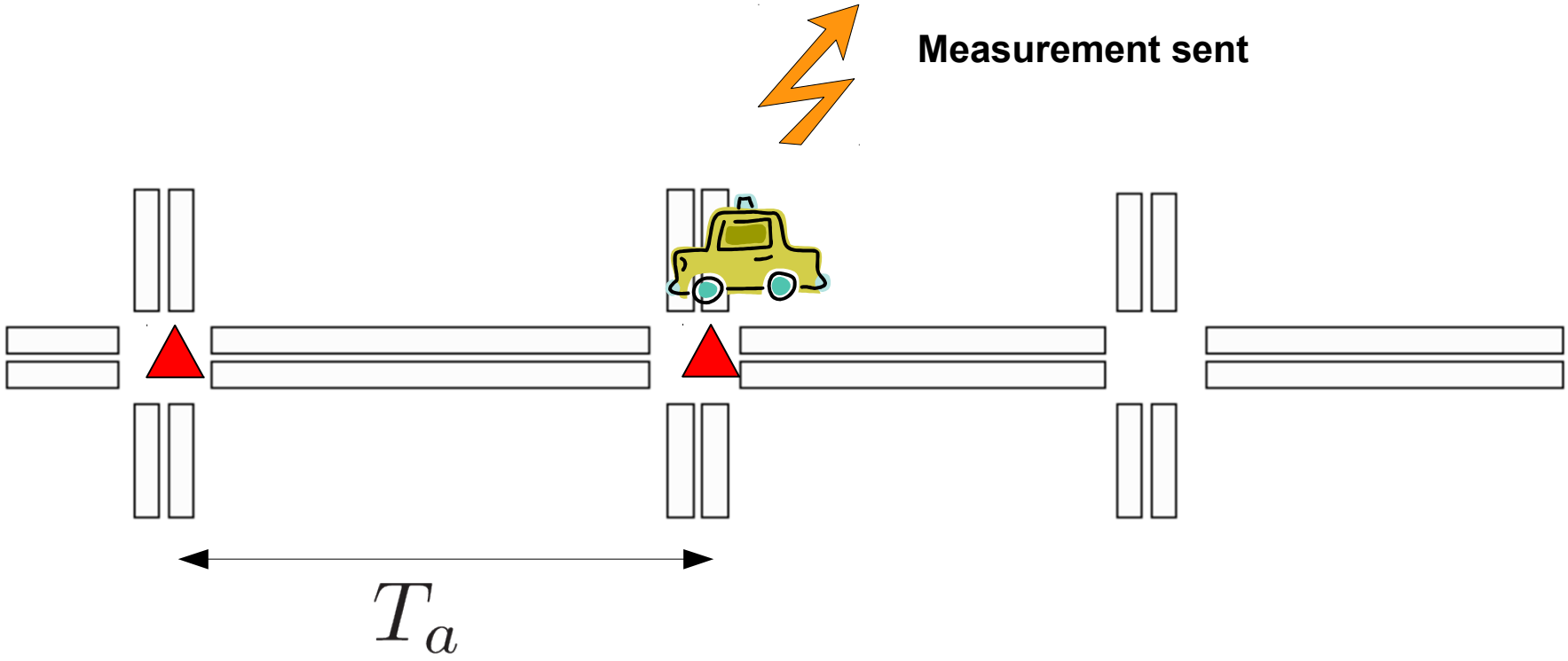
The way things work



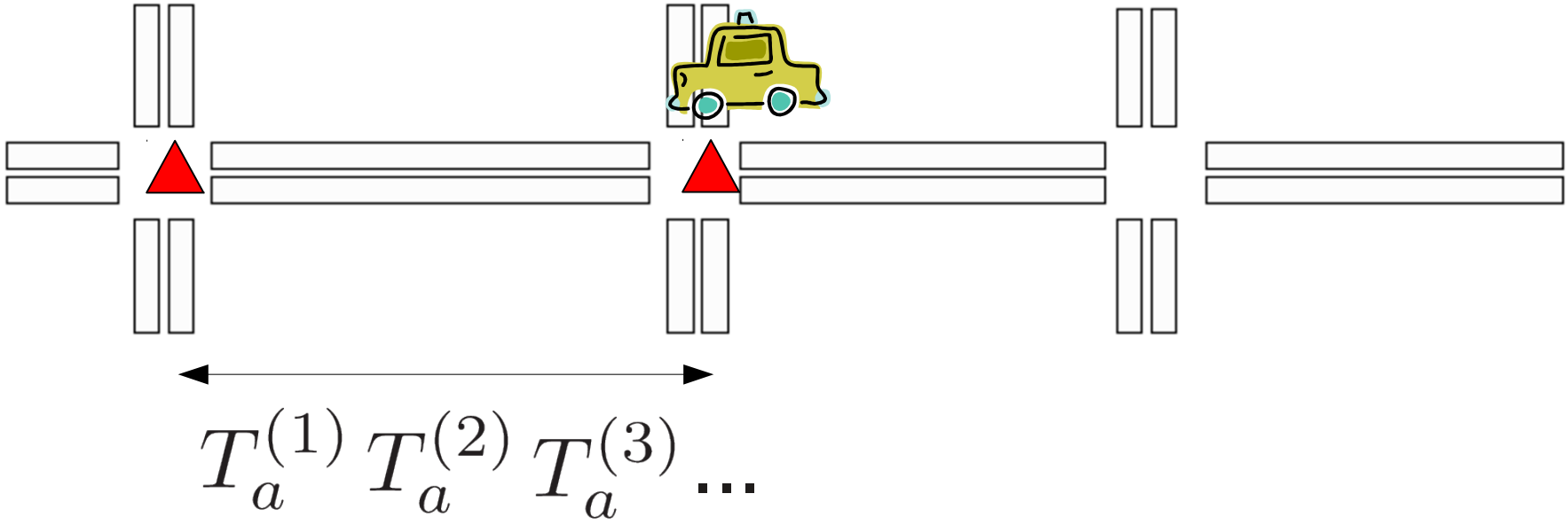
The way things work



The way things work

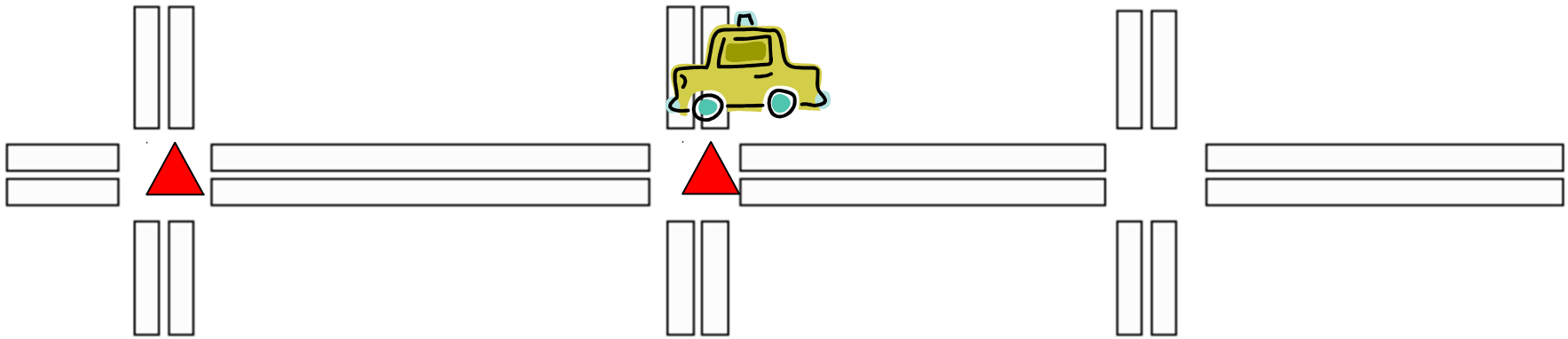


The way things work

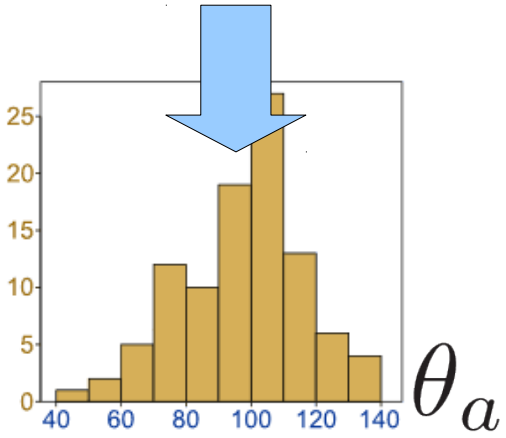


$$T_a^{(1)}\ T_a^{(2)}\ T_a^{(3)}\ \ldots$$

The way things work

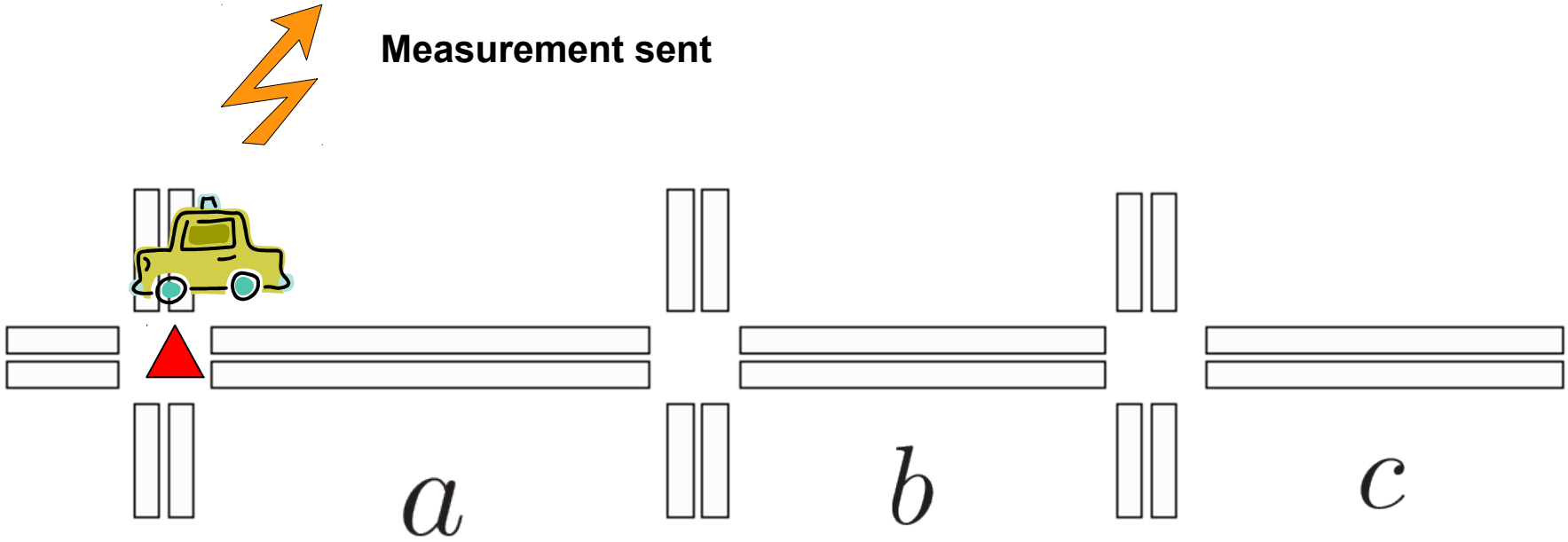


$$T_a^{(1)}\ T_a^{(2)}\ T_a^{(3)}\ \ldots$$



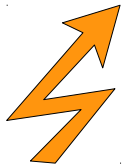
θ_a

Life is not so simple

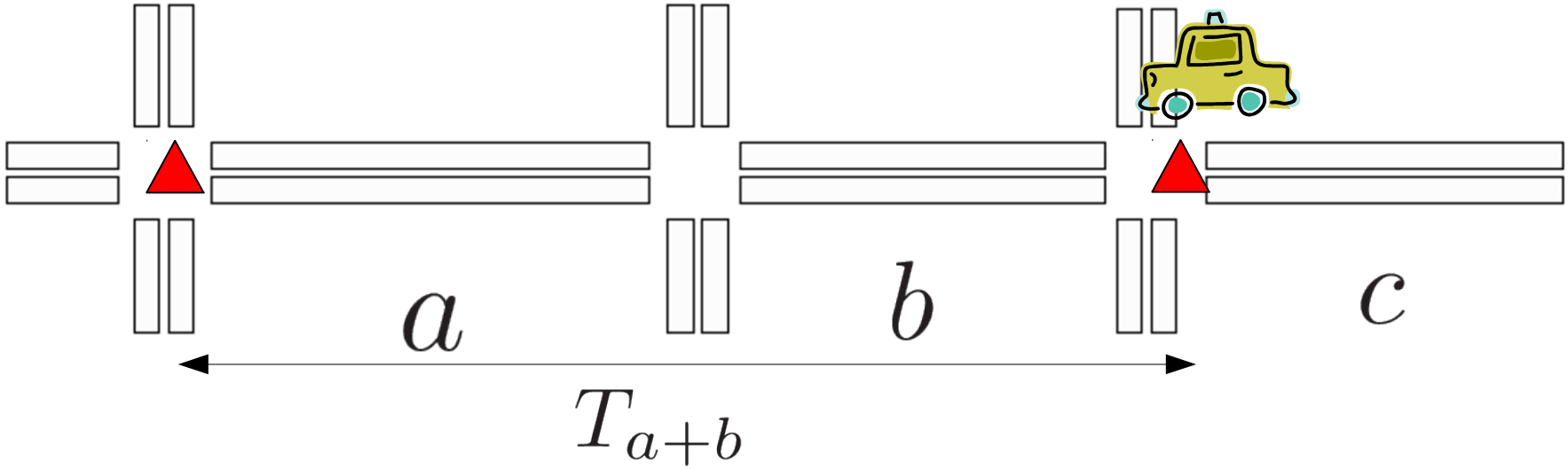


Life is not so simple

- Long time between observations



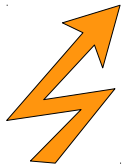
Measurement sent



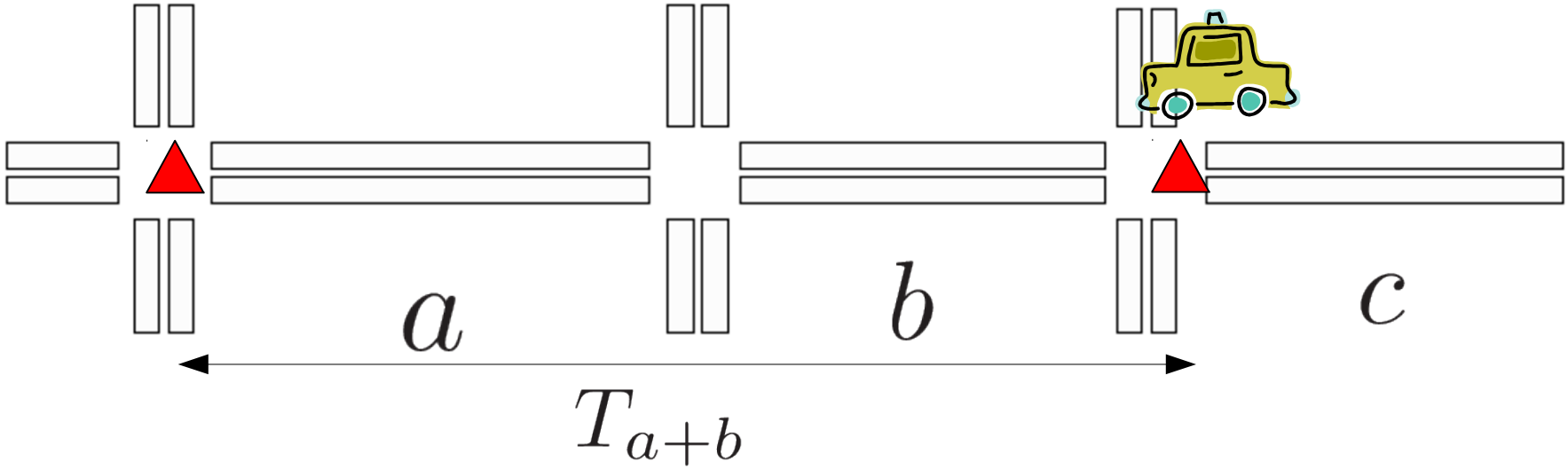
$T_a?$ $T_b?$

Life is not so simple

- Long time between observations



Measurement sent

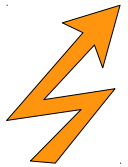


$T_a?$ $T_b?$

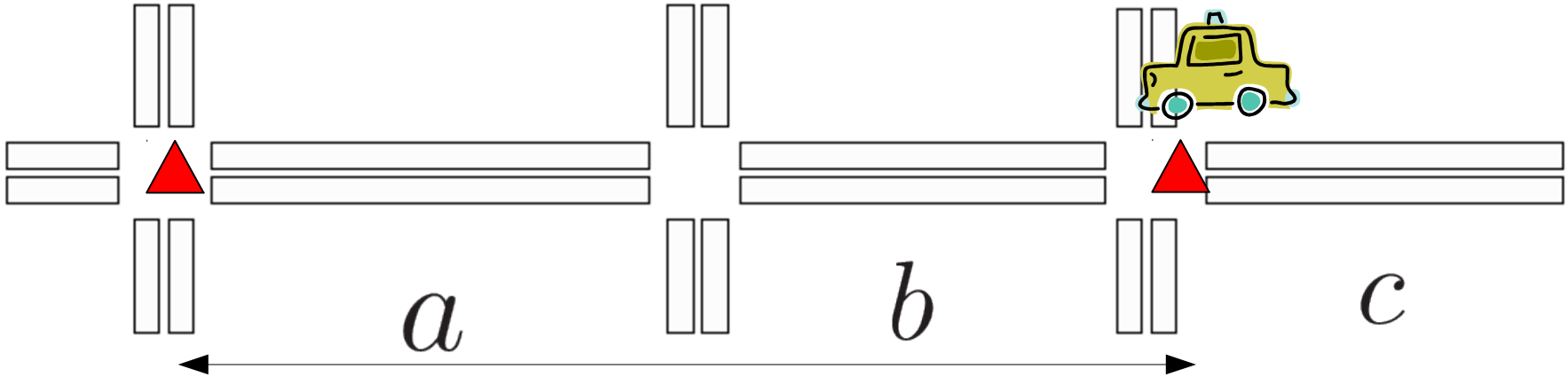
- Solution: sample!

Life is not so simple

- Long time between observations



Measurement sent

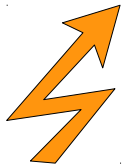


$$T_{a+b}$$

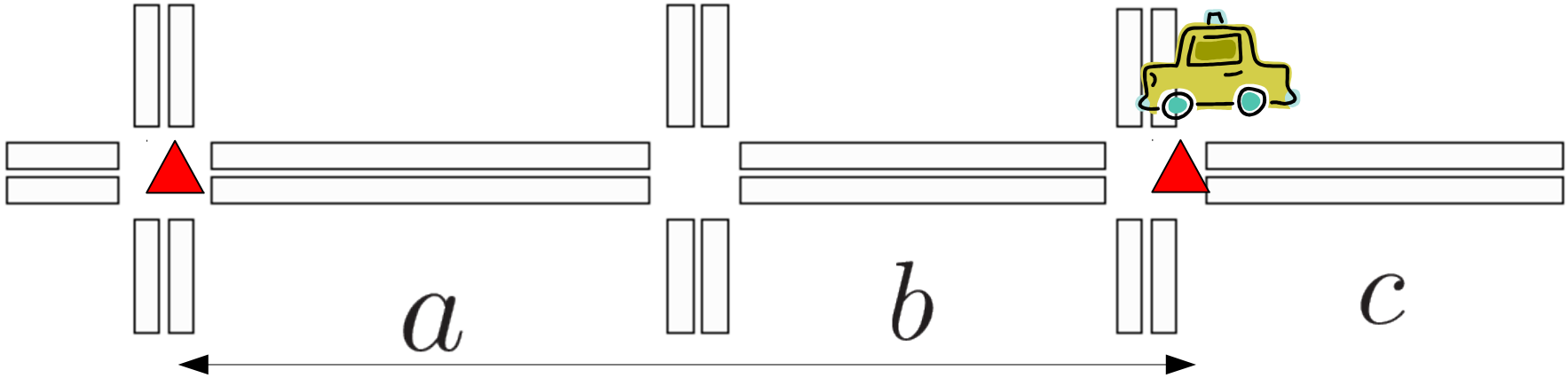
$$t_a^{(1)}, t_a^{(2)}, t_a^{(3)}, \ldots \qquad t_b^{(1)}, t_b^{(2)}, t_b^{(3)}, \ldots$$

Life is not so simple

- Long time between observations



Measurement sent



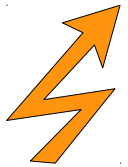
$$T_{a+b}^{(1)}, T_{a+b}^{(2)}, \cdots$$

$$t_a^{(1,1)}, t_a^{(1,2)}, t_a^{(1,3)}, \cdots \qquad t_b^{(1,1)}, t_b^{(1,2)}, t_b^{(1,3)}, \cdots$$

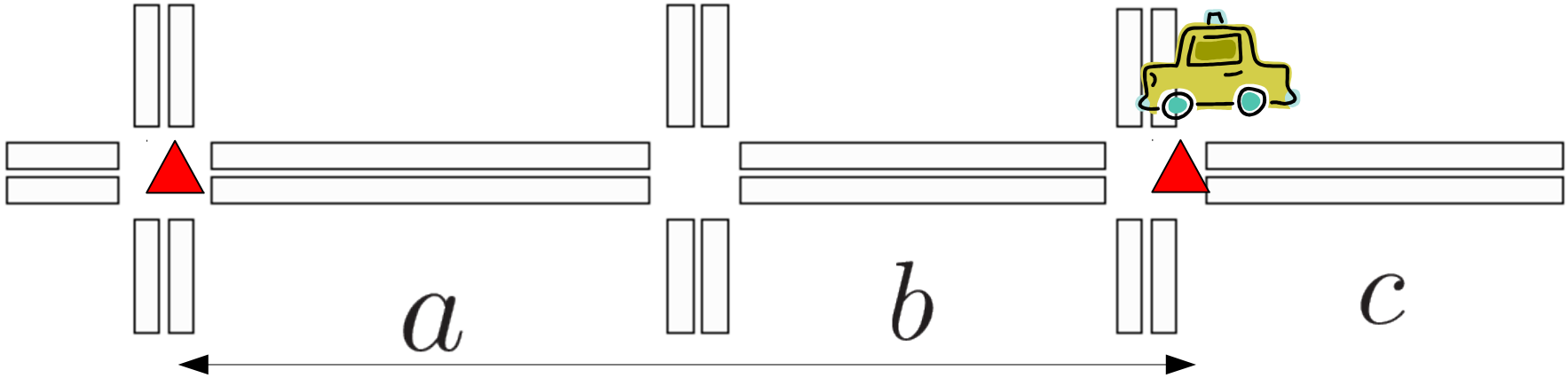
$$t_a^{(2,1)}, t_a^{(2,2)}, t_a^{(2,3)}, \cdots \qquad t_b^{(2,1)}, t_b^{(2,2)}, t_b^{(2,3)}, \cdots$$

Life is not so simple

- Long time between observations



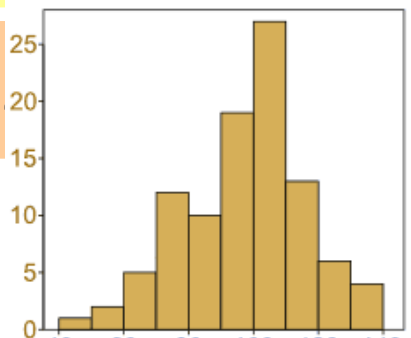
Measurement sent



$$T_{a+b}^{(1)}, T_{a+b}^{(2)}, \cdots$$

$$t_a^{(1,1)}, t_a^{(1,2)}, t_a^{(1,3)}, \cdots$$

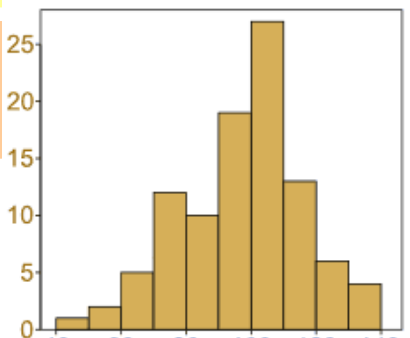
$$t_a^{(2,1)} \quad 3) \quad , \cdots$$



θ_a

$$t_b^{(1,1)}, t_b^{(1,2)}, t_b^{(1,3)}, \cdots$$

$$t_b^{(2,} \quad 2,3) \quad , \cdots$$

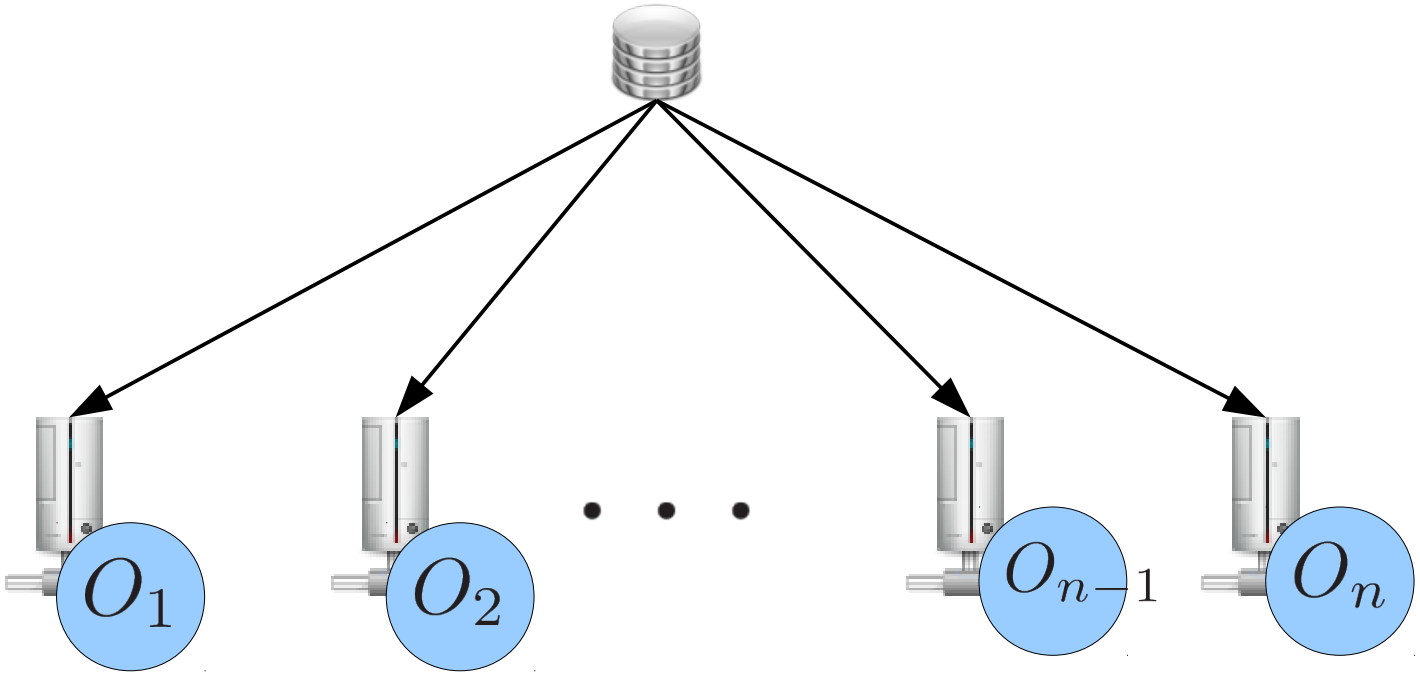


θ_b

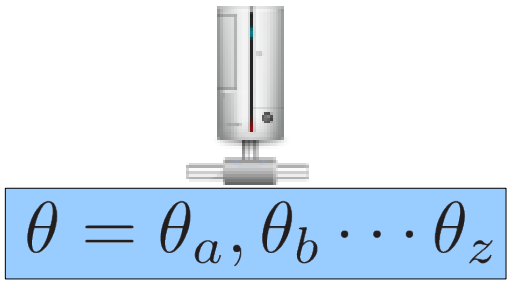
Machine learning without saying it

- Procedure called *Expectation Maximization*
- Iterative in nature:
  - Alternates between sampling (E step) and learning (M step)
- Some figures:
  - 50k road links (parameters)
  - 50M observations (15GB, avg. 4 links / observation)
  - 200M partial travel times
  - x1000 samples per partial travel times

System workflow

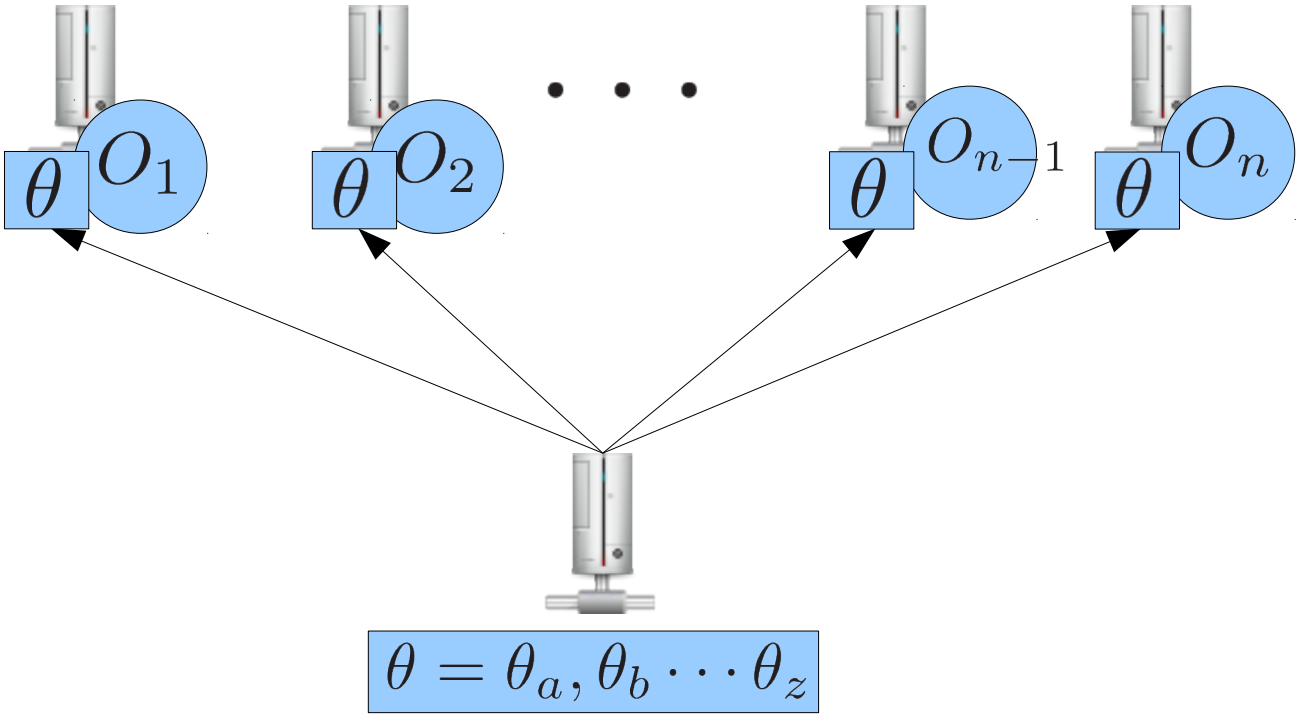


Observations
(distributed, persisted
across nodes)



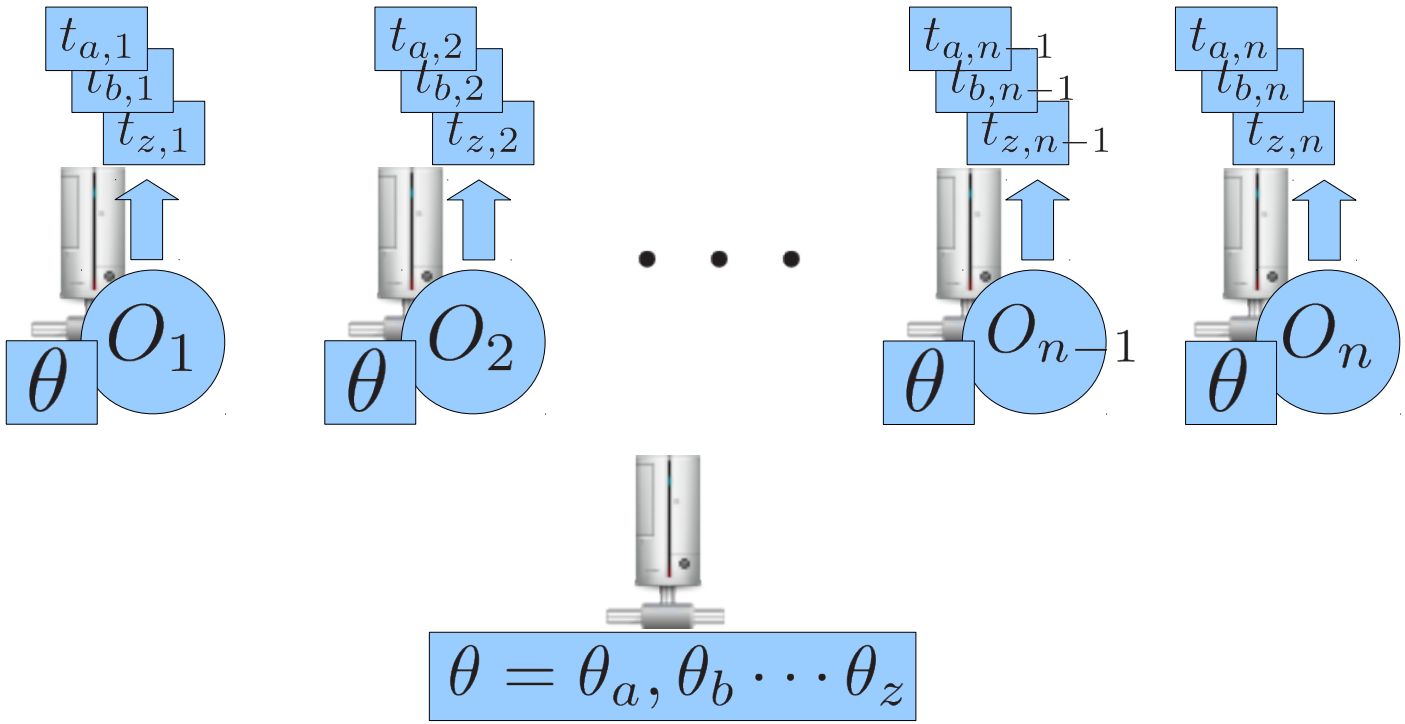
Start link parameters
(on master node)

System workflow



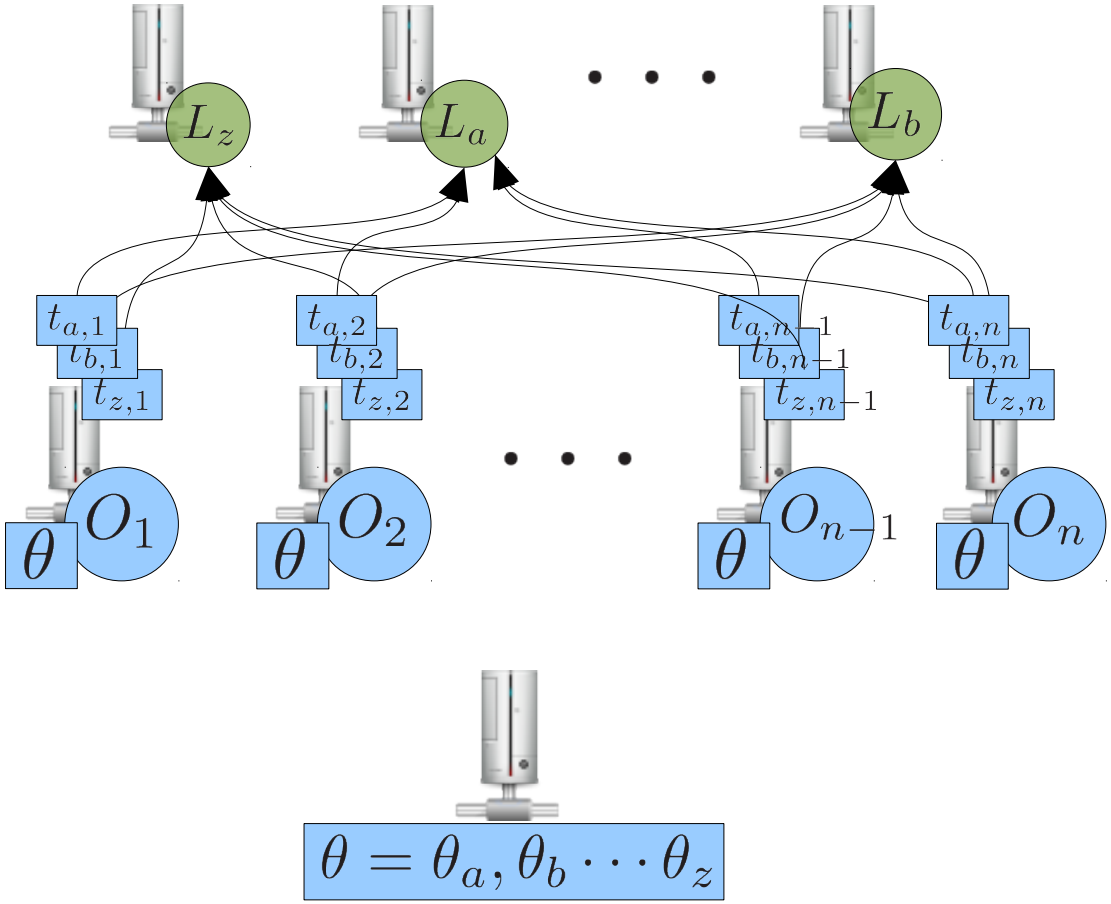
Network parameters
(distributed over the nodes)

System workflow



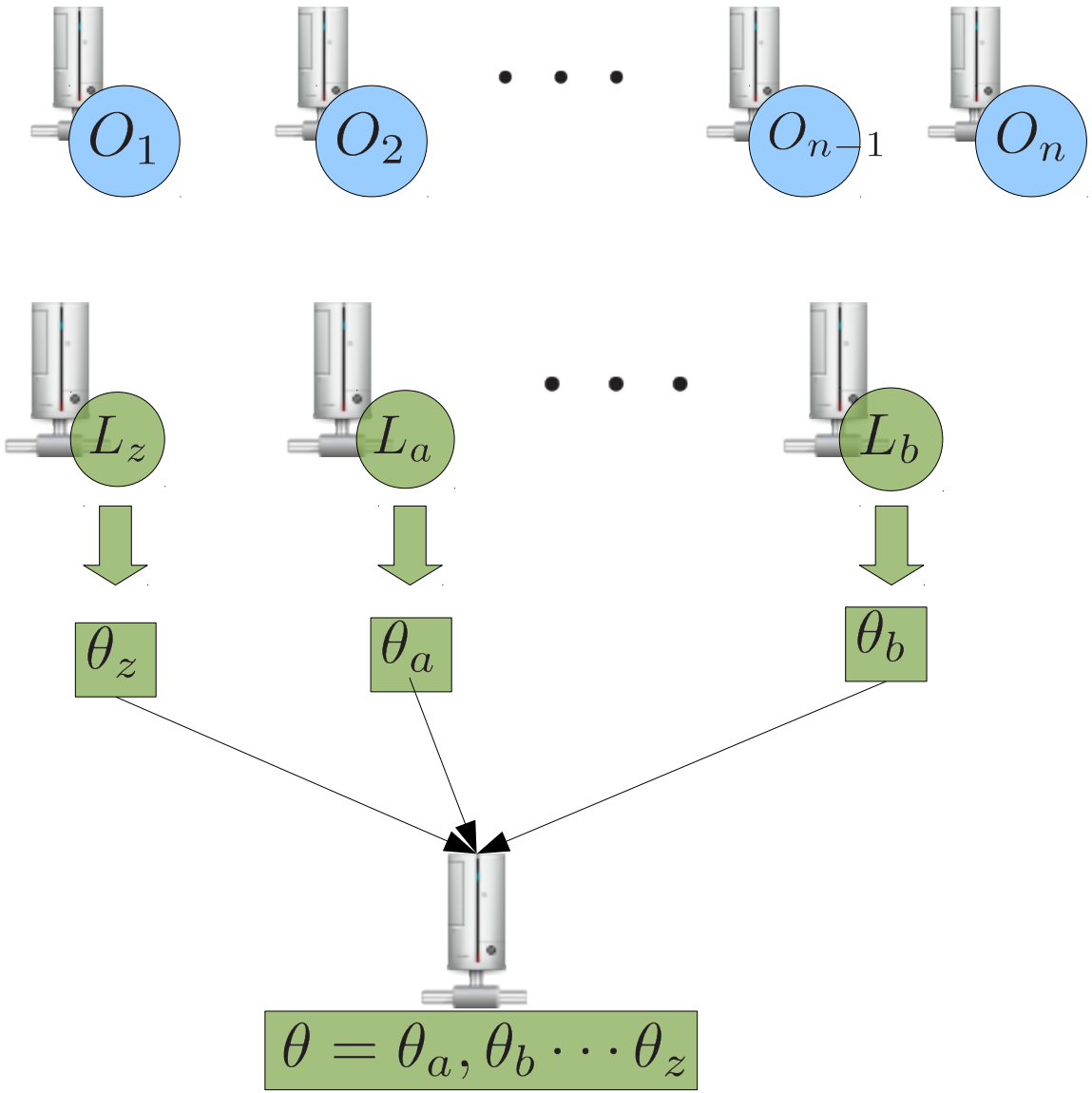
Travel time samples
For each observation link

System workflow



Travel time samples aggregated on a link basis

System workflow



New parameters are generated
The maximize sampled travel times for each link.

The master collects the vector of new parameters.

Main loop of the program

```
val observations = spark.textFile("hdfs:...")
  .map(parseObservation _)
  .cache()
var params = // Initialize models parameters
while (!converged) {
```

Step 1 (E step)

```
  val samples = observations.flatMap( obs =>
    generateSamples(obs, params))
```

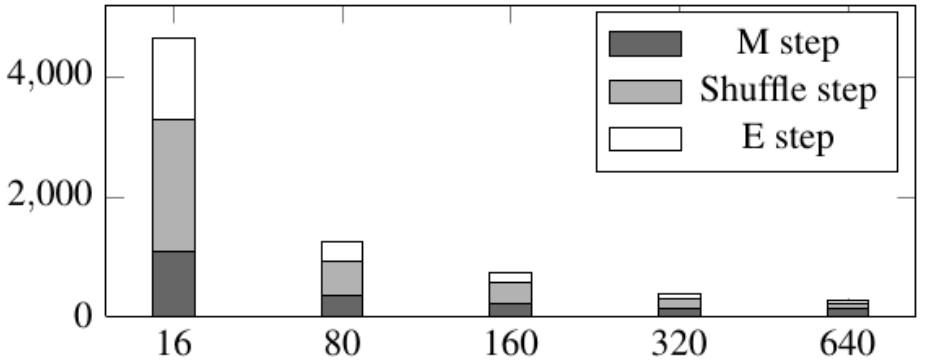
Step 2 (M step)

```
  params = samples.groupByKey(false).map(
    case (linkId, vals) =>
        mostLikelyParam(linkId, vals)
  ).collect()
}
```

The good

- Before using Spark:
  - 3.5x *slower* than real-time
  - Could not even handle all the data
- With Spark:
  - Similar programming interface (methods on scala collections)
  - Very good scalability (near linear)
  - Each iteration 3x faster than reloading from disk

runtime



cores

NERSC cluster:
quad-core Xeon
4X QDR InfiniBand interconnect

Efficient utilization of memory

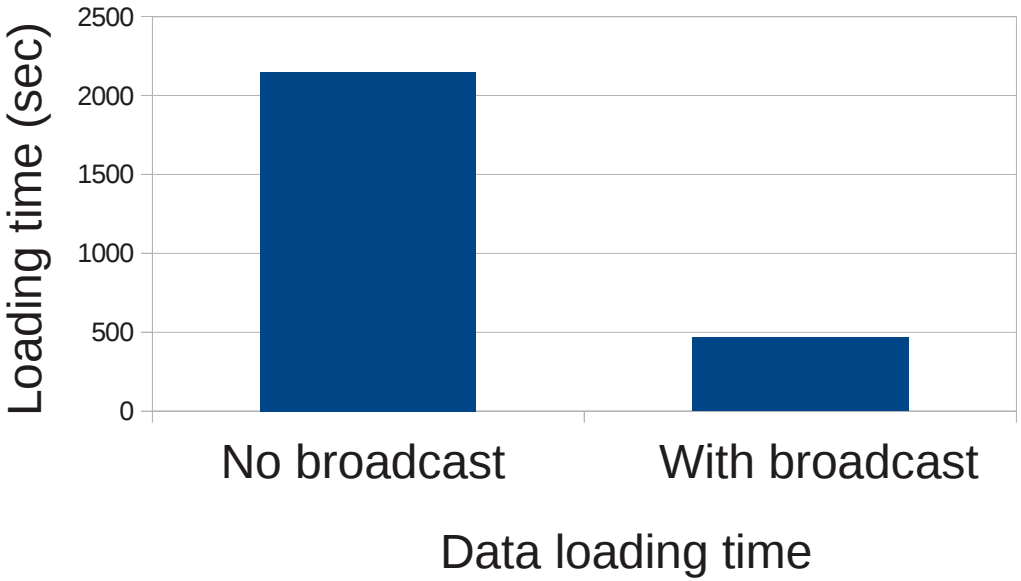
- The observation data is stored in memory:
  - Be careful with the memory footprint
  - Look at logs to monitor GC status
- We cache pointer-based structures
  - Significant overhead in the JVM
- Workaround: use compact collection structures (arrays) and make liberal use of `.toArray()`
- Workaround: RDDs of *serialized data*

Broadcast of large parameters

- Need to share data between all workers:
  - At the start of the job (network description, > 40MB)
  - Between iterations (updated parameters θ)
- Using Spark's broadcast
- Data loading time reduced by 79%

```
val network = // load network
val observations = spark.textFile("...")
  .map(parseObservation(_, network))
```

```
val network = // load network
val bc_net = spark.broadcast(network)
val observations = spark.textFile("...")
  .map(parseObservation(_, bc_net.get()))
```



Conclusion

- An application of Spark:
  - Real-world ML problem
  - Crowd-sourced data
- Implementation now (much) faster than real time
- Not limited by computations:
  - We can use more complex ML tools than before

Thank you