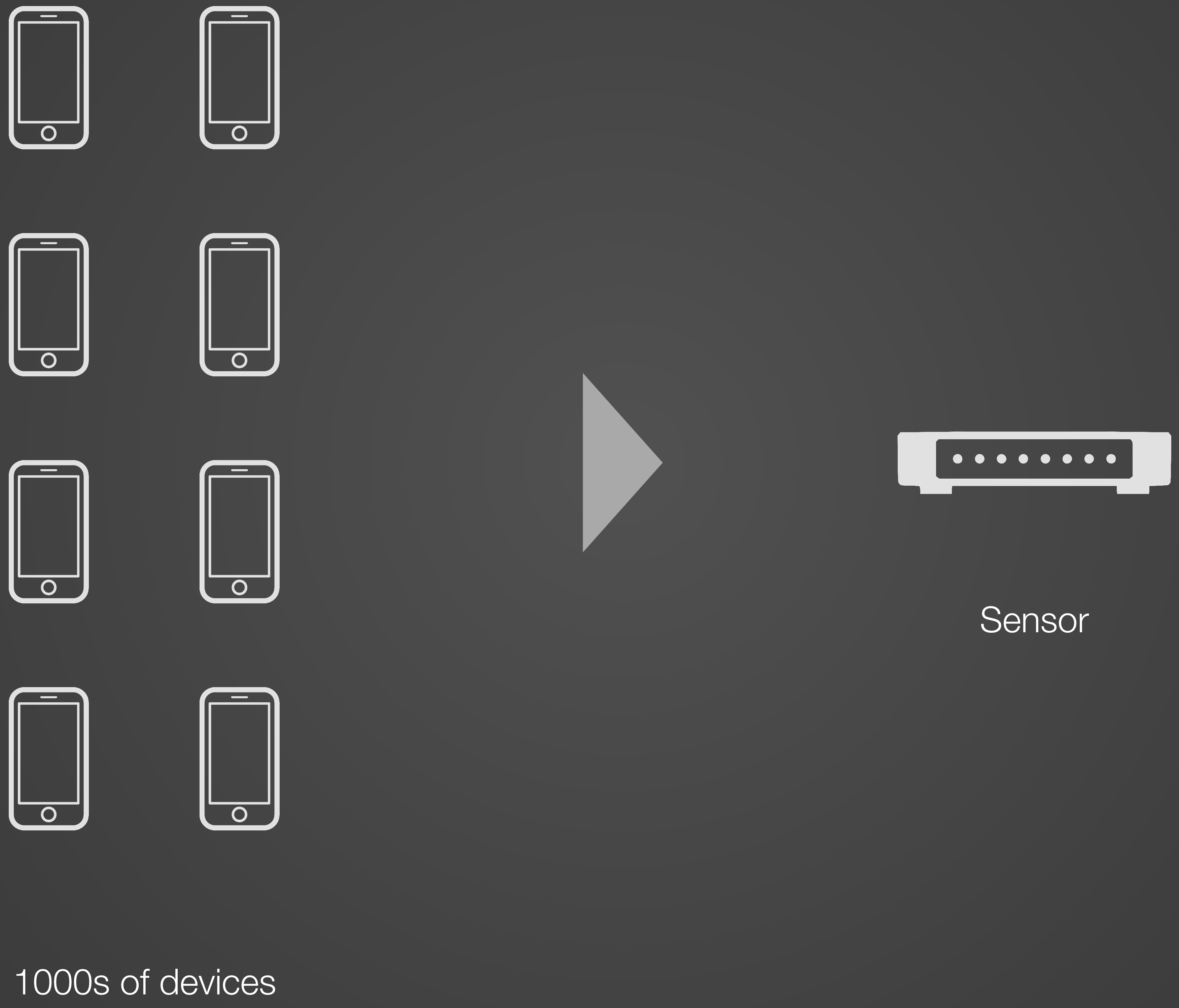
Medium Data Toolkit

Case study on SmartStreetSensors

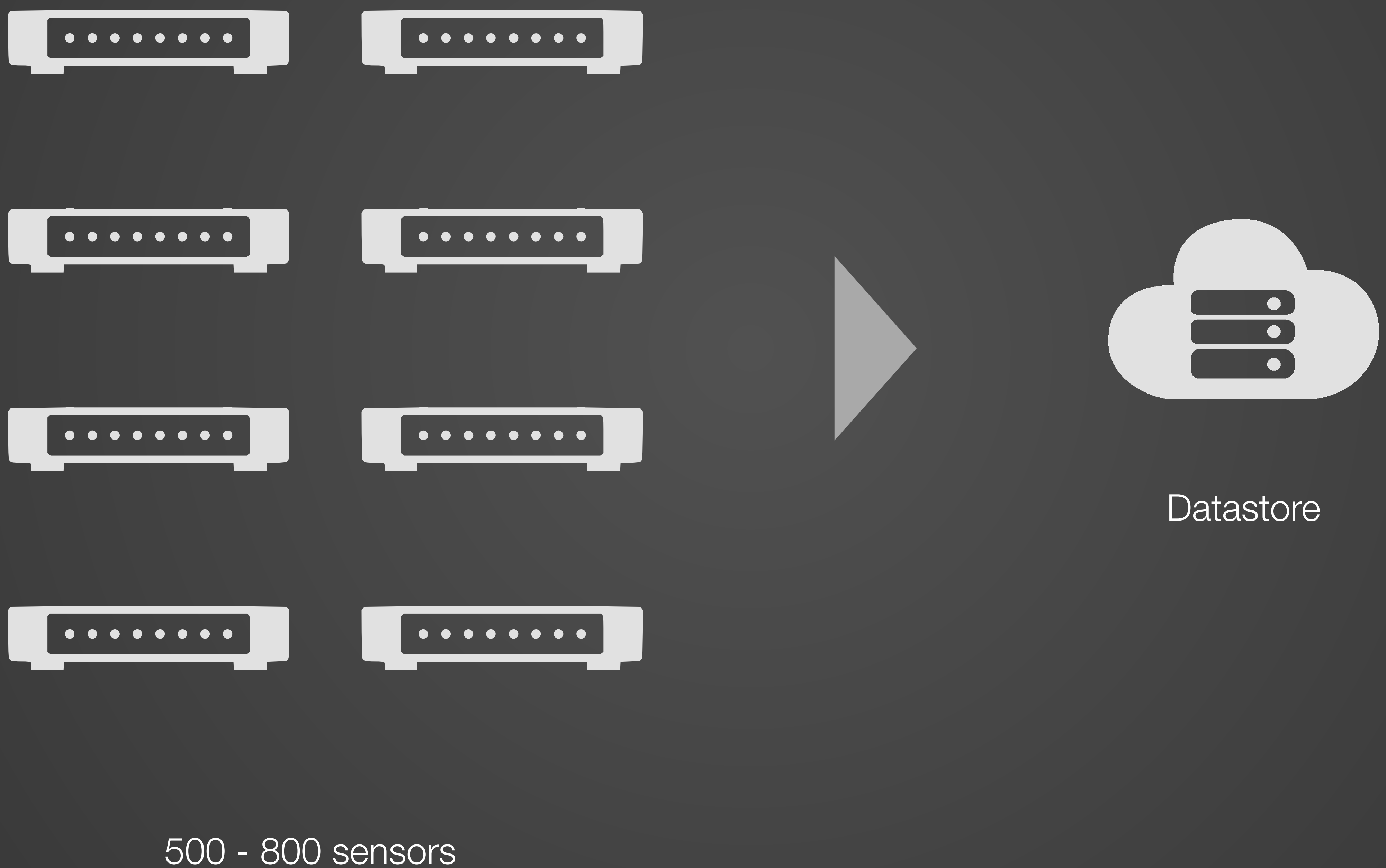
Data Collection

Since Sept 2015

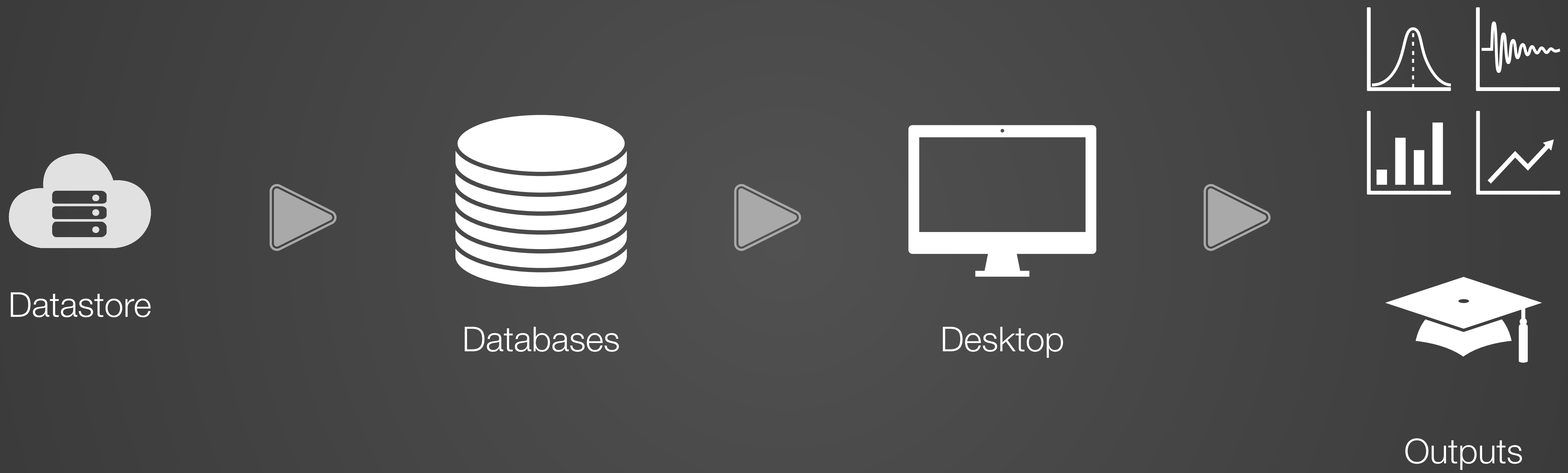
At each Location every 5 mins



At National level every 5 mins



At University



First Attempt

February 2016

40 Locations

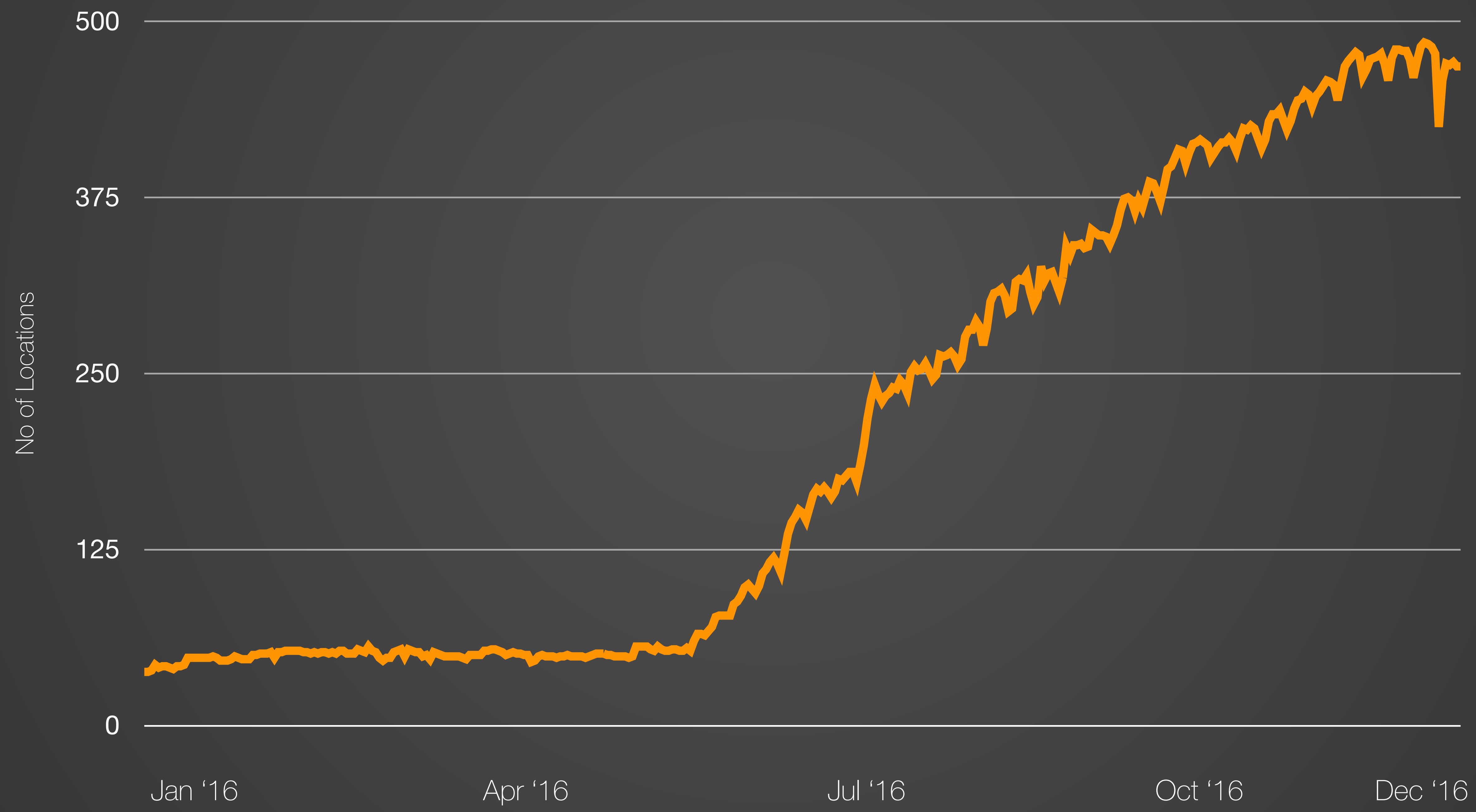
~**1 Mn** records / day

~100 MB / day

~ 20 mins to download

~ 30 Mins to process

The scale of the project grew



December 2018

675 Locations

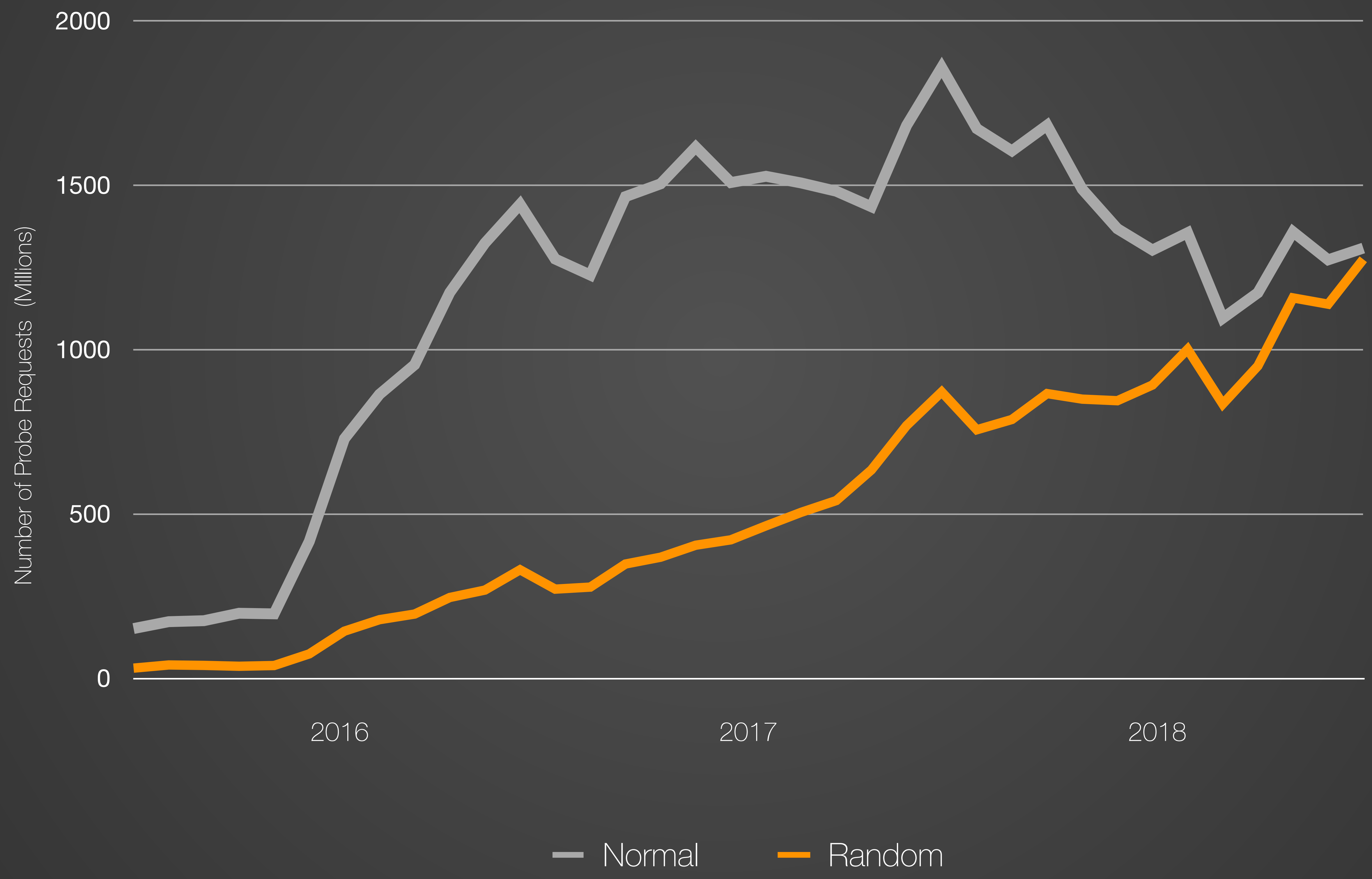
~**31 Mn** records / day

~2 GB / day

> 2 Hours to download

> 5 Hours to process

The complexity of the project grew

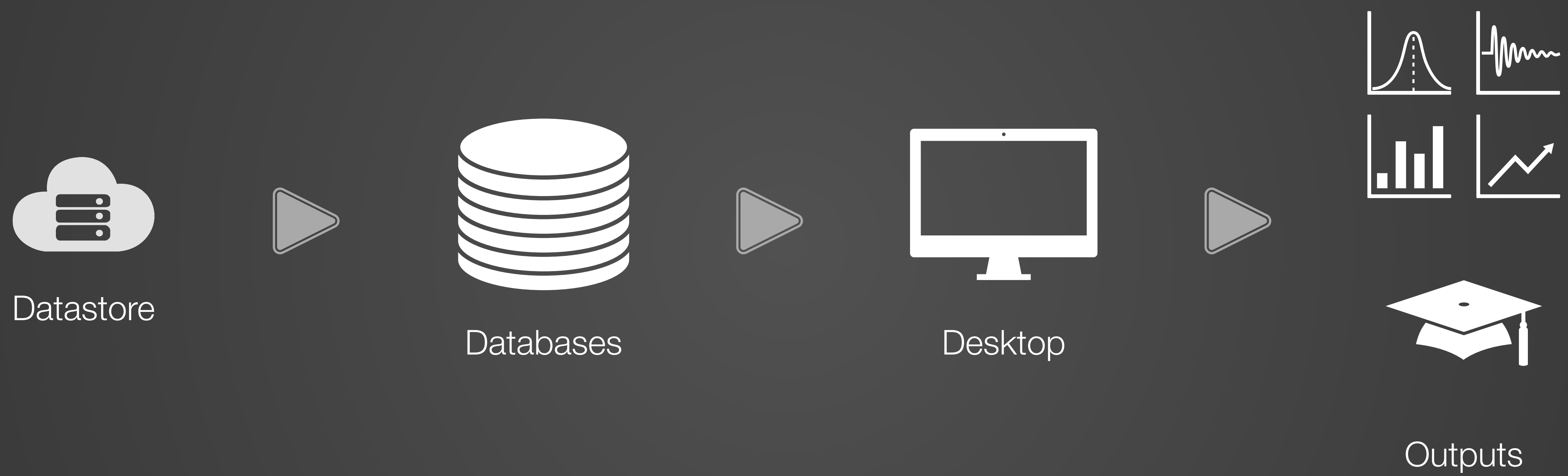


To make **Sense** of the data,

Increase amount of data collected

Use more intensive processing

‘Big data’ approach


Datastore
Databases
Desktop
Outputs


Datastore
Big Data
Infrastructure
Outputs

Big Data Landscape

Data Ingestion
- Apache Flume
- Apache Sqoop
- Facebook Scribe
- Apache Chukwa
- Apache Kafka
- Netflix Suro
- Apache Samza
- Cloudera Morphline
- HIHO
- Apache NiFi
- Apache MainfoldCF

Distributed File System
- HDFS
- OutcastFS
- LustreFS
- GridGain
- ClusterFS
- CephFS
- Alluxio
- XtreemFS

NoSQL Databases

Wide Column
- Apache HBase
- Apache Cassandra
- Hypertable
- Apache Accumulo
- Apache Kudu
- Apache Parquet

Document Store
- MongoDB
- RethinkDB
- ArangoDB
- CouchDB
- DynamoDB
- GemFire

Key-Value
- Redis
- LinkedIn Voldemort
- RocksDB
- Open TSDB

Graph
- Giraph
- Neo4j
- TitanDB
- OrientDB

New SQL Databases
- TokuDB
- Sky
- Handler
- BayesDB
- Akiban
- InfluxDB
- Drizzle
- VoltDB
- Haeinsa
- SAP HANA
- SensieDB

SQL on Hadoop
- Apache Hive
- Facebook Presto
- Apache HCatalog
- Datasalt Splout SQL
- Apache Trafodion
- Apache Tajo
- Apache HAWQ
- Apache Phoenix
- Apache Drill
- Apache MRQL
- Cloudera Impala
- Kylin

Distributed Programming
- Apache Ignite
- Apache REEF
- Apache MapReduce
- Apache Twill
- Apache Pig
- Damballa Parkour
- JAQL
- Apache Hama
- Apache Spark
- Datasalt Pangool
- Apache Spark
- Apache Tez
- Apache Storm
- Apache DataFu
- Apache Flink
- Kangaroo
- Apache Apex
- Tinkerpop
- Netflix PigPen
- Pachyderm MR
- AMPLAB SIMR
- Apache Beam
- Facebook Corona

Service Programming
- Apache Thrift
- Apache Karaf
- Apache Zookeeper
- Twitter Elephant Bird
- Apache Avro
- LinkedIn Norbert
- Apache Curator

Scheduling & DR
- Apache Oozie
- Apache Falcon
- LinkedIn Azkaban
- Shedoscope

Security
- Apache Sentry
- Apache Knox Gateway
- Apache Ranger

Frameworks
- Jumbune
- Spring XD
- Cask Data

Metadata
- Metascope
- Apache Tika

Big Data Landscape

Accumulo Alluxio Deoxys

Haeinsa Kylin Voldemort Farfetchd Parquet

Flume

Piloswine Arango Machoke Norbert

Glaceon

Spark Hive Flink Akiban

Azkaban

Kafka Combee

Tajo

Froakie Tinkerpop Heatmor Ralts

Arcanine

Samza Sqoop

Corona

Kudu Elgyem Zookeeper Honedge

Suro Ninetales

Tika Chukwa

Trafodion

Big Data Tools		Pokemon
Accumulo	Azkaban	Deoxys
Alluxio	Kafka	Farfetchd
Voldemort	Tajo	Machoke
Haeinsa	Tinkerpop	Piloswine
Kylin	Sqoop	Glaceon
Parquet	Samza	Combee
Flume	Corona	Arcanine
Arango	Kudu	Froakie
Norbert	Zookeeper	Heatmor
Spark	Suro	Ralts
Hive	Chukwa	Honedge
Flink	Tika	Elgyem
Akiban	Trafodion	Ninetales

It is **confusing** and involves lot of **cost** !

Researcher time

Resources

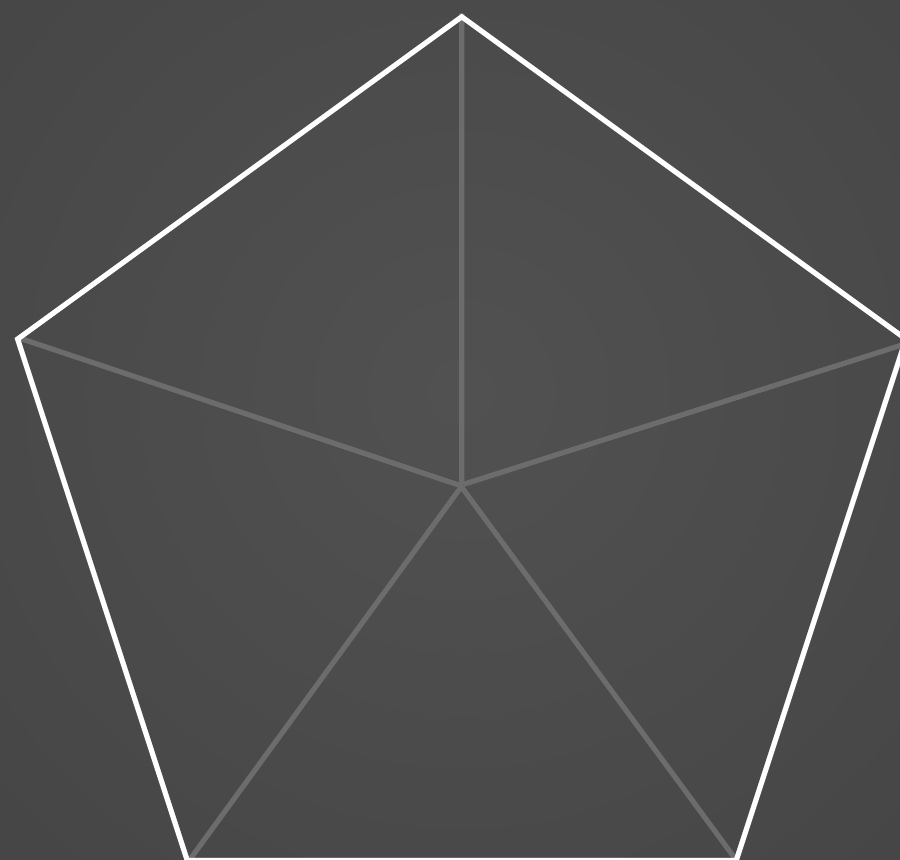
Am I dealing with **big data**?

Evaluating ‘bigness’ of the data

Defining Big Data

Volume

The total amount of
data generated



Visualisation

The challenge of abstracting the
data to simple visuals

Velocity

The rate at which the
data is generated

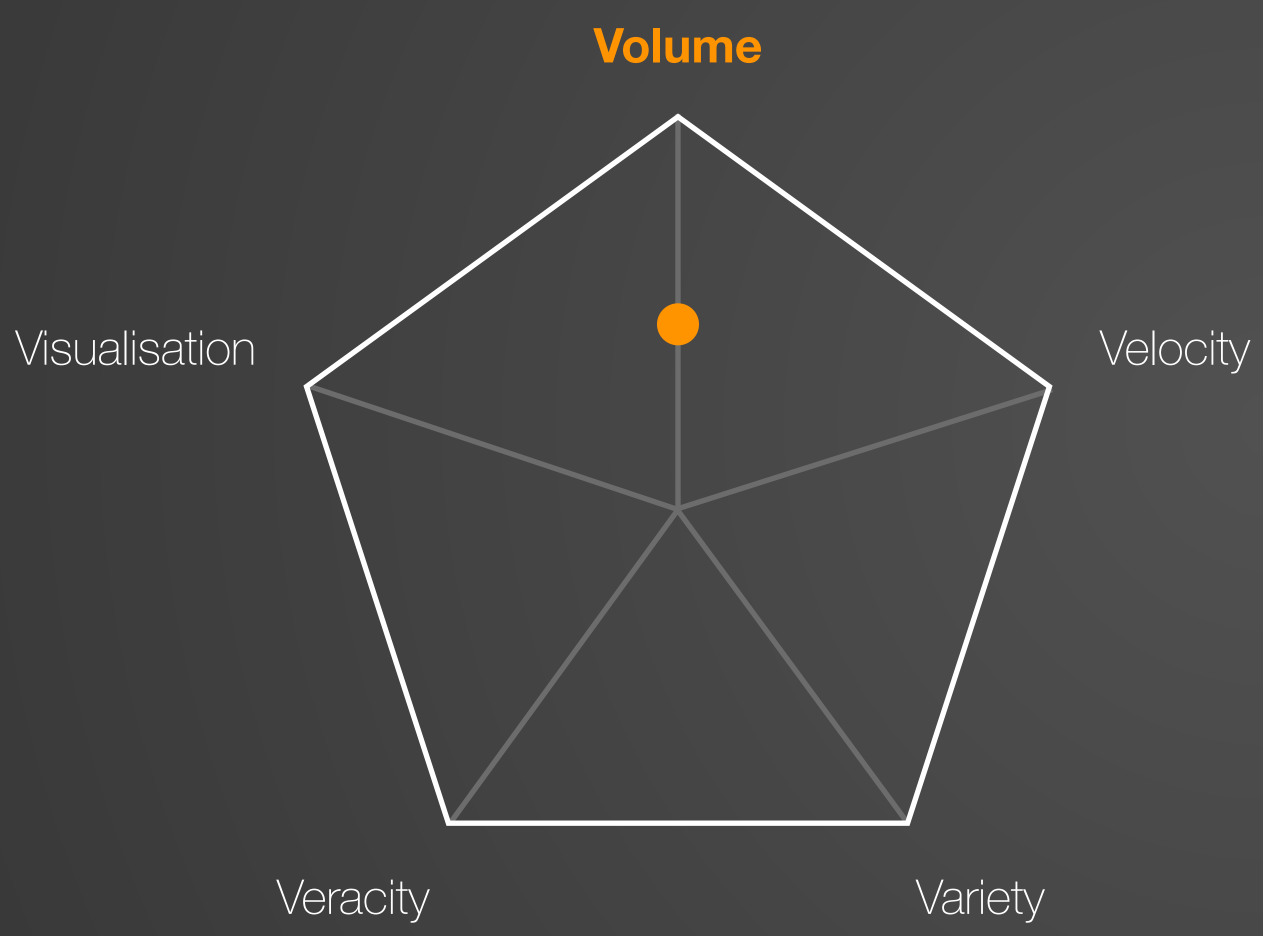
Veracity

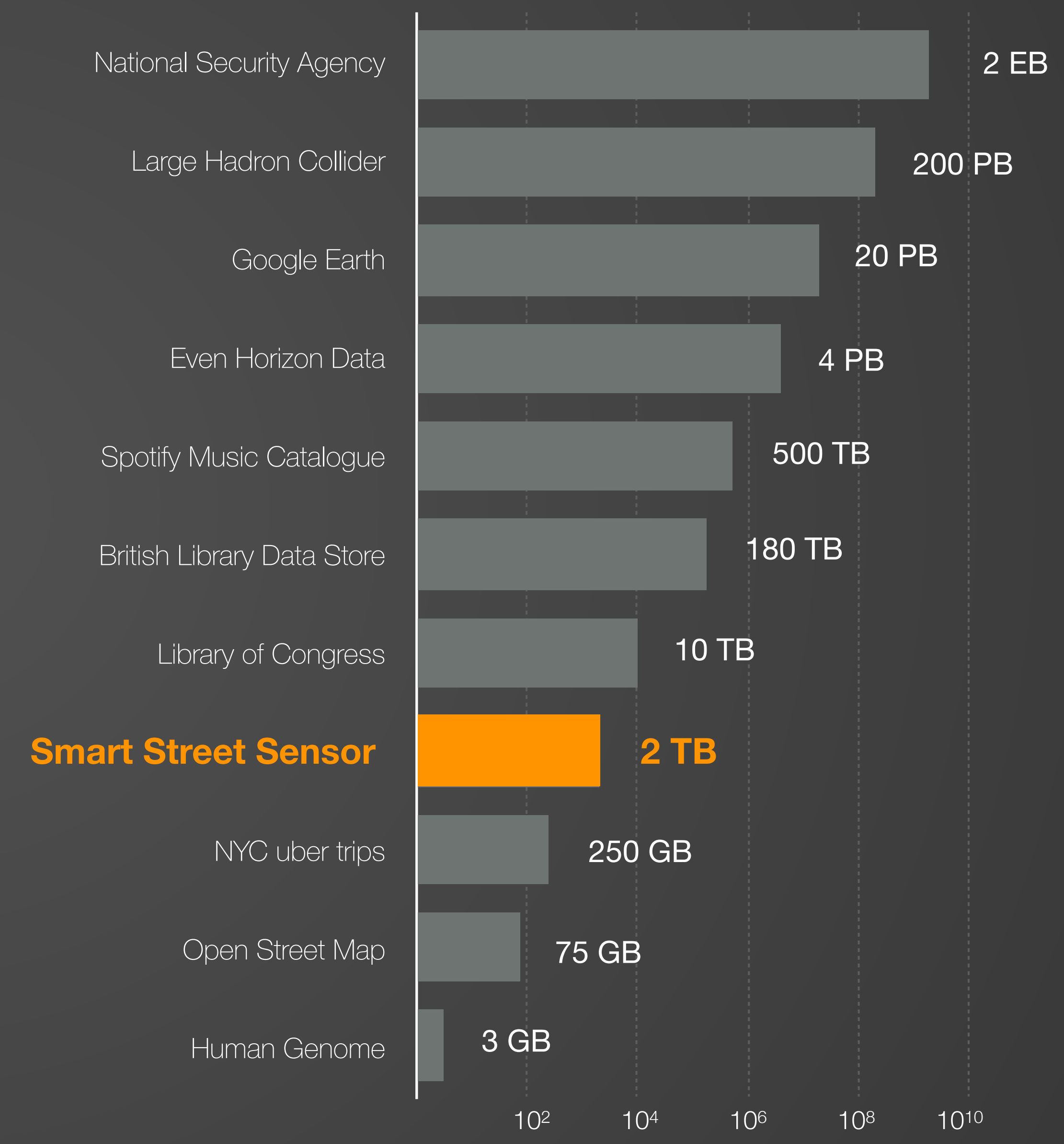
The truthfulness of the data in terms of
bias, inaccuracy and noise

Variety

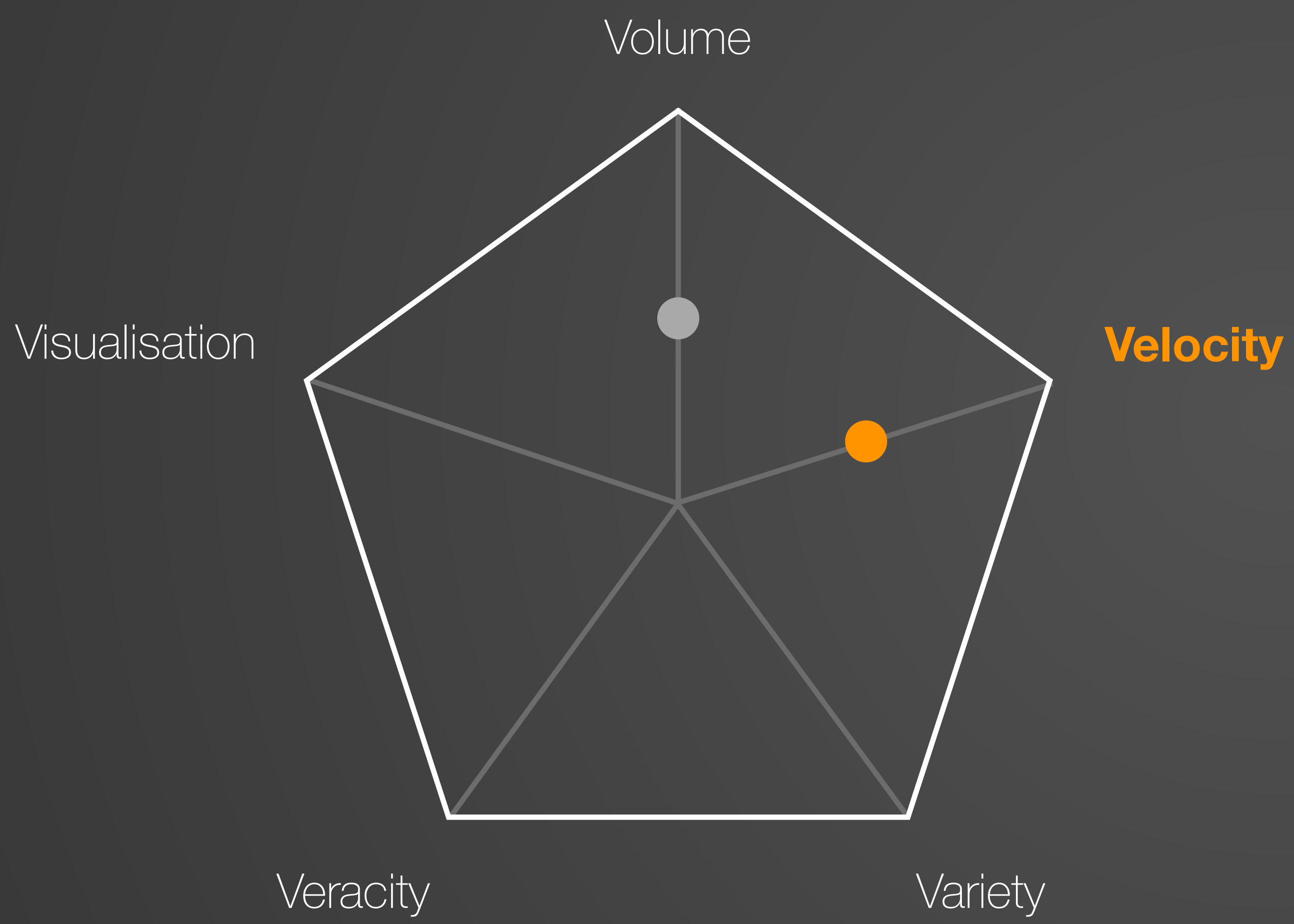
The amount of different type of
data in the data set

Evaluating Wi-Fi data





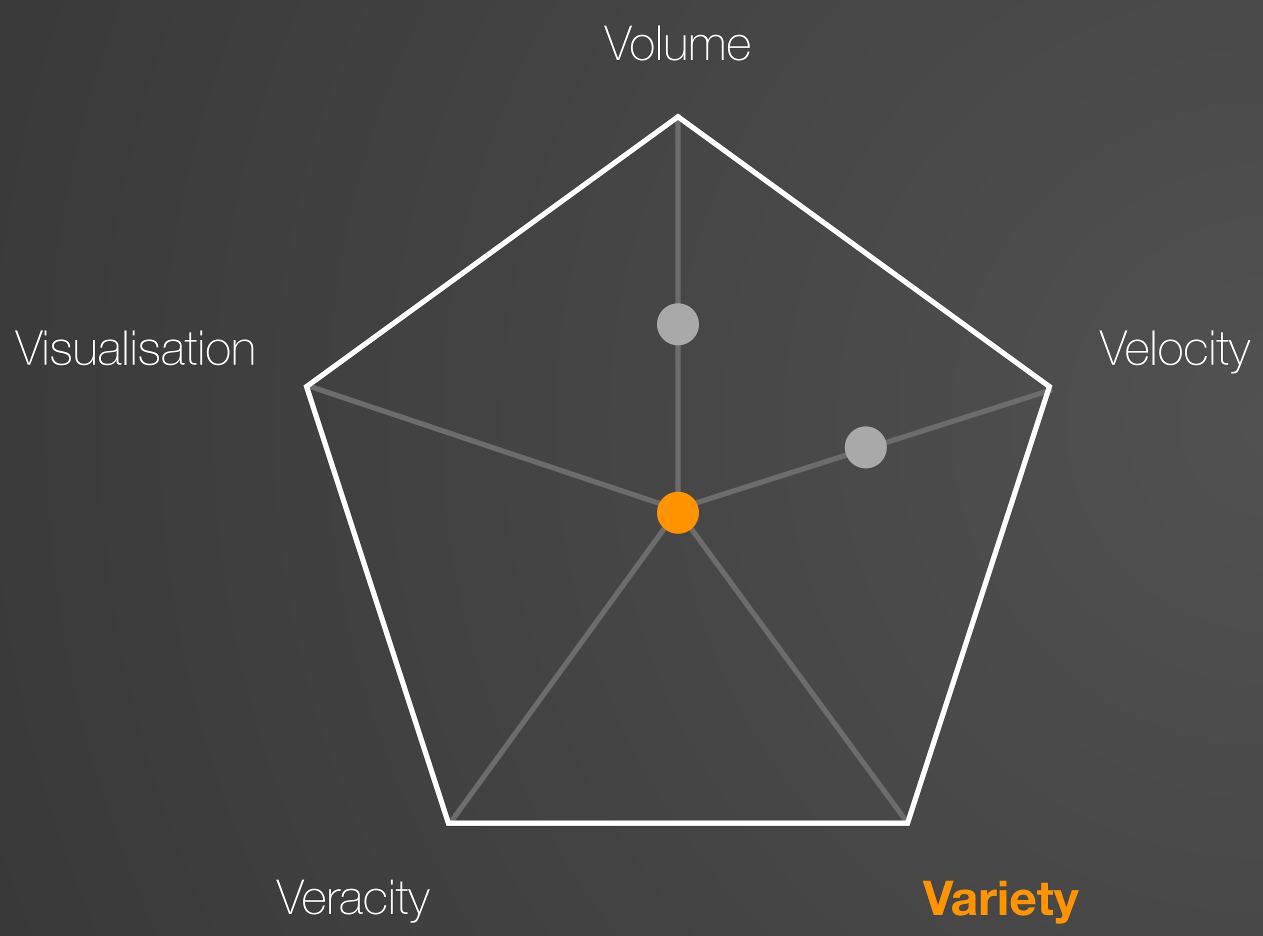
Evaluating Wi-Fi data



250,000 records every 5 mins

2 gb per data per day

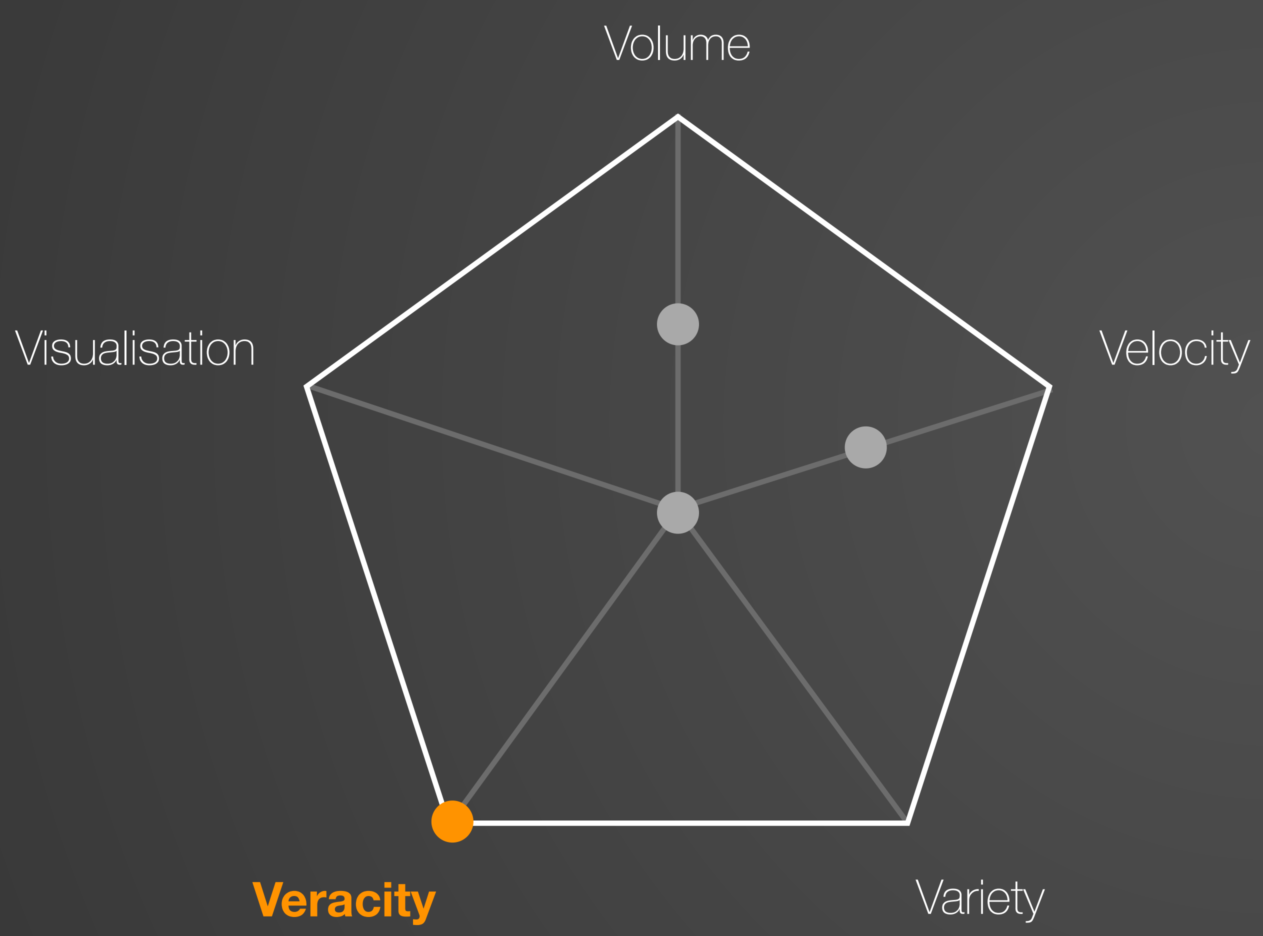
Evaluating Wi-Fi data



Part of the **IEEE Standard**

No variation in the data

Evaluating Wi-Fi data



Bias

Mobile phone ownership across geography and demography

Inaccuracies

Sensor failure, Sensor reboots, Sensor hardware and software versions

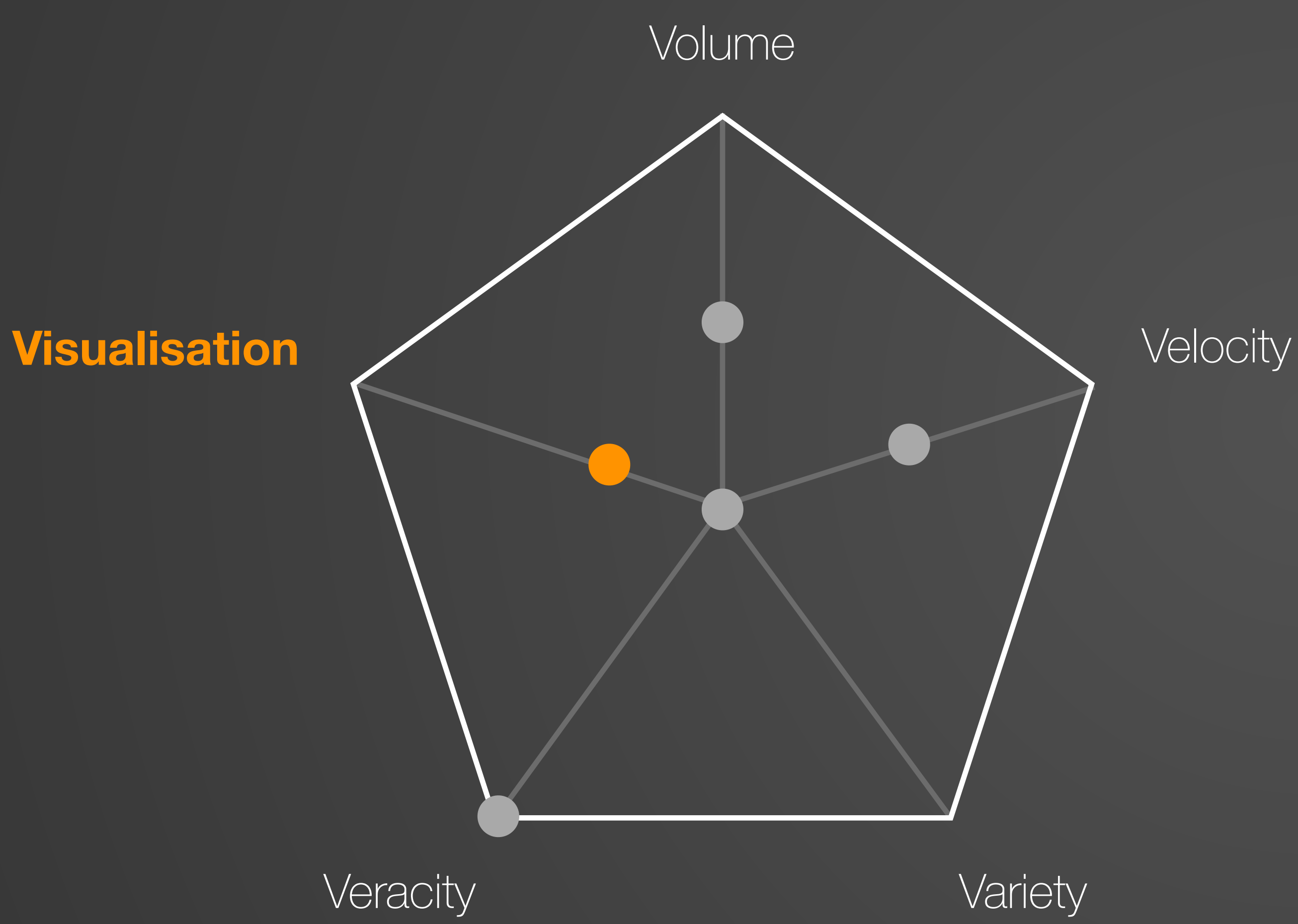
Noise

Variable field of measurement, Variable rate of generation of probe requests

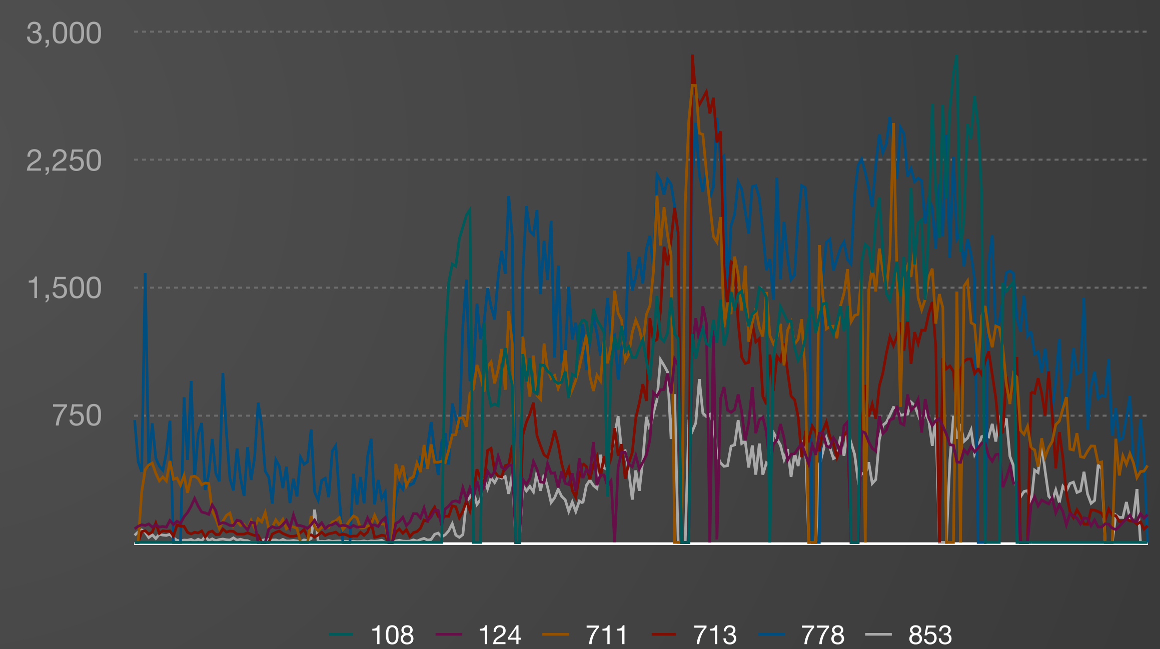
Uncertainties

Change in smart phone landscape, MAC address randomisation, Changes in standards

Evaluating Wi-Fi data

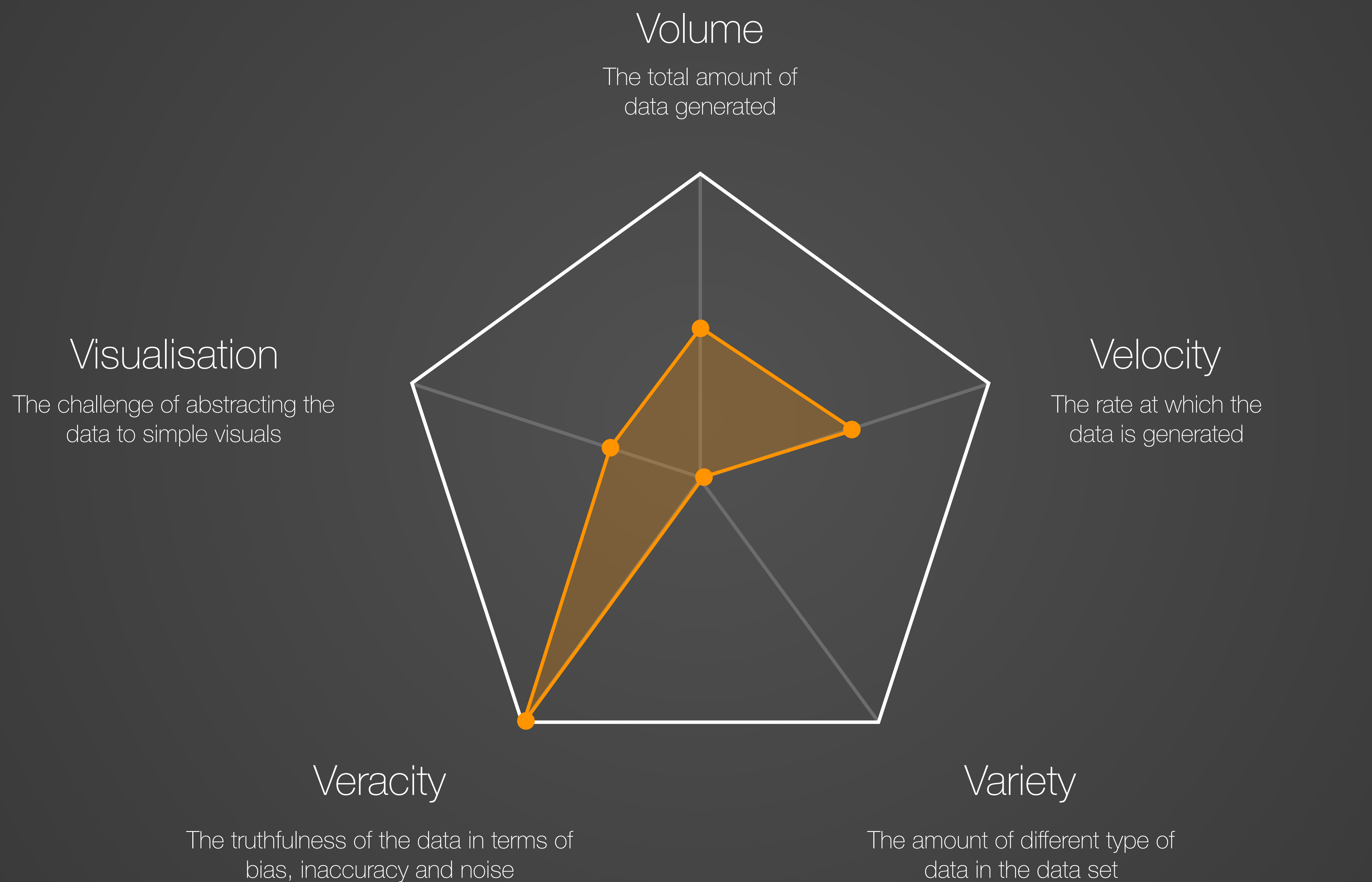


Time dimension of the data



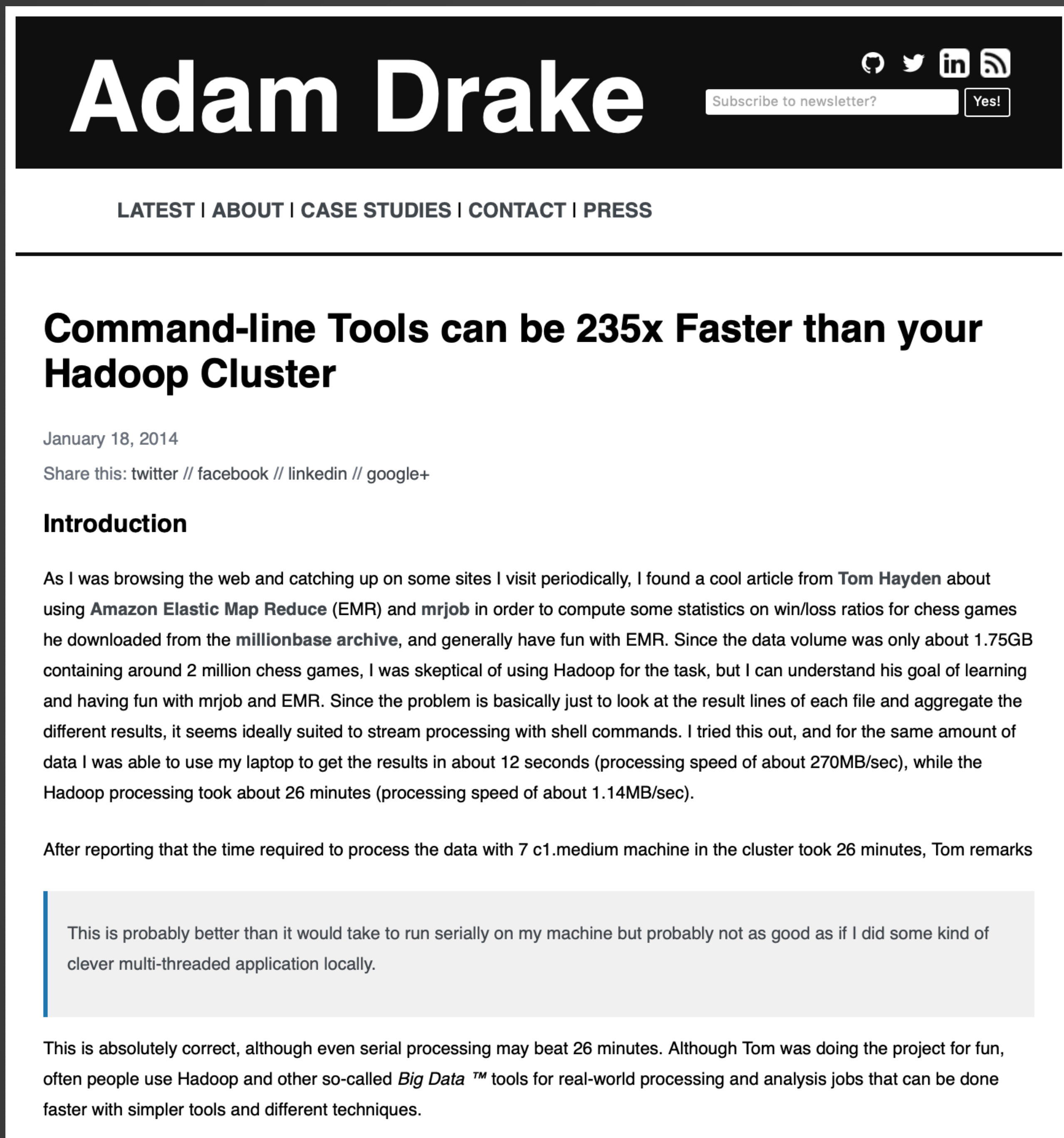
Data on Tottenham Court Road, London
On 15 Jan 2019

Medium Data?


Volume
The total amount of data generated
Velocity
The rate at which the data is generated
Variety
The amount of different type of data in the data set
Veracity
The truthfulness of the data in terms of bias, inaccuracy and noise
Visualisation
The challenge of abstracting the data to simple visuals

The **Toolkit** for medium data

Unix Tools and Big data



Adam Drake

Subscribe to newsletter? Yes!

LATEST | ABOUT | CASE STUDIES | CONTACT | PRESS

Command-line Tools can be 235x Faster than your Hadoop Cluster

January 18, 2014

Share this: twitter // facebook // linkedin // google+

Introduction

As I was browsing the web and catching up on some sites I visit periodically, I found a cool article from **Tom Hayden** about using **Amazon Elastic Map Reduce** (EMR) and **mrjob** in order to compute some statistics on win/loss ratios for chess games he downloaded from the **millionbase archive**, and generally have fun with EMR. Since the data volume was only about 1.75GB containing around 2 million chess games, I was skeptical of using Hadoop for the task, but I can understand his goal of learning and having fun with mrjob and EMR. Since the problem is basically just to look at the result lines of each file and aggregate the different results, it seems ideally suited to stream processing with shell commands. I tried this out, and for the same amount of data I was able to use my laptop to get the results in about 12 seconds (processing speed of about 270MB/sec), while the Hadoop processing took about 26 minutes (processing speed of about 1.14MB/sec).

After reporting that the time required to process the data with 7 c1.medium machine in the cluster took 26 minutes, Tom remarks

> This is probably better than it would take to run serially on my machine but probably not as good as if I did some kind of clever multi-threaded application locally.

This is absolutely correct, although even serial processing may beat 26 minutes. Although Tom was doing the project for fun, often people use Hadoop and other so-called *Big Data ™* tools for real-world processing and analysis jobs that can be done faster with simpler tools and different techniques.

https://adamdrake.com/command-line-tools-can-be-235x-faster-than-your-hadoop-cluster.html

Unix Philosophy

Pipes and text streams

Command-line tools

cat, find, sort, uniq, sed, grep and awk

Parallelisation

Using xargs to utilise all processor cores

Unix Philosophy

Write programs that do **one thing and do it well**.

Write programs to **work together.**

Write programs to handle **text streams**, because that is a universal interface.

Peter H. Salus, A Quarter-Century of Unix (1994)

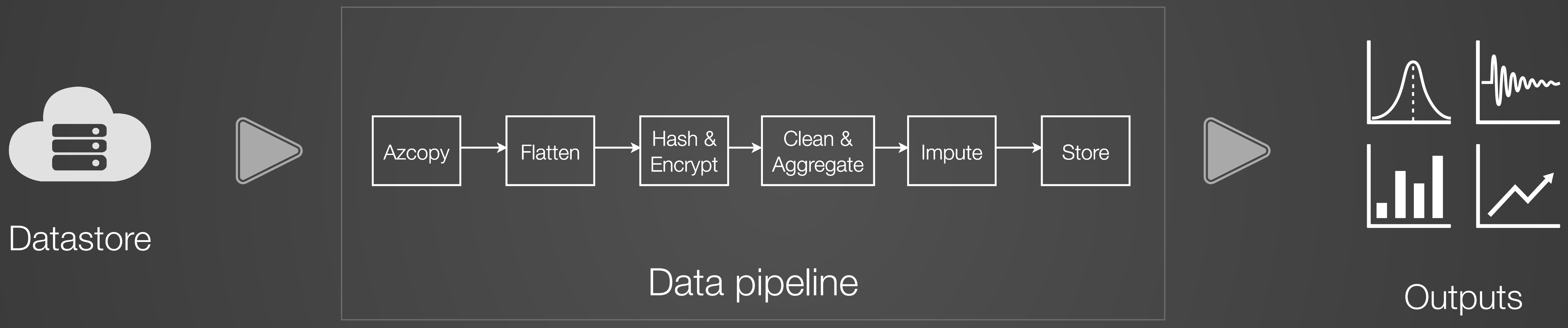
Unix tools

R package	Purpose	Unix tool
azureR & odbc	For downloading data from the azure data store and sql server	azcopy mssql
jsonlite	To convert the json files into data frame for analysis in R	jq
Rcrypt	To encrypt the raw data for transfer to the secure data store	gnupg
digest	To convert personal data into cryptographic hashes	openssl
dplyr	To manipulate and aggregate raw data into counts	find cat cut grep sed awk sort uniq
ImputeTS & ggplot2	To fill in missing data and visualise results	Rscript
RPostgreSQL	To transfer the processed data into a database.	psql

Unix tools

find	Powerful tool for searching filesystem
grep	Searching filesystem contents using regular expressions
cat	Print and concatenate files
cut	To select columns of csv files
sed	Text editor for manipulating streams
awk	Turing complete language for working on text streams
sort	Sorting text streams or csv
uniq	Aggregating text streams by unique values

Data processing pipeline



Parallelising the pipeline

Gnu-parallel To parallelise unix pipelines

Bechmarks

Pipeline in R

```
# ---------------------------------------------------------------
# Loading the libraries
# ---------------------------------------------------------------
library(tidyverse)
library(RJSONIO)
day_folder <- "location_of_the_data"
sensors <- paste(day_folder, dir(day_folder), sep = "/")[1:25]

# ---------------------------------------------------------------
# Read and Parse JSON files
# ---------------------------------------------------------------
for(sensor in sensors) {
  files <- paste(sensor, dir(sensor), sep = "/")
  for( file in files ) {
    records <- fromJSON(file);
    location <- vector();
    timestamp <- vector();
    macaddress <- vector();
    for(record in records) {
      location <- append(location, get_location(file))
      timestamp <- append(time, get_time(file))
      fullmac <- paste0(record$MacAddress, record$VendorMacPart)
      macaddress <- append(macaddress, fullmac); }
    df <- data.frame(location, timestamp, mac)
    probes <- rbind(probes, df) } }

# ---------------------------------------------------------------
# Aggregate the counts for each interval
# ---------------------------------------------------------------
probes %>%
  group_by(location, time) %>%
  summarise(count = length(unique(paste0(vendor, mac)))) %>%
  write.csv("output.csv",row.names=FALSE)
```

20m14s

Pipeline in Unix tools

```
# -------------------------------------------------------
# Get a list of locations
# -------------------------------------------------------
awkc="awk -vFPAT='[^,]*|\"[^\"]*\"' -v OFS=','"
folder="location_of_the_data"
sensors=`ls $FOLDER | head -n 25`


# -------------------------------------------------------
# Loop through locations and parse and output results
# -------------------------------------------------------
for sensor in $sensors;
do
  jq_string=".[] | \
    [\"$sensor\",\
    .timestamp_from_filename,\
    .VendorMacPart+.MacAddress] \
    | @csv";
  cmd="jq -r '$jq_string' $folder$sensor/*.pd \
    | sort | uniq \
    | $awkc '{print \$1,\$2}' \
    | sort | uniq -c";
  echo "$(eval $cmd)" > output.csv;
done
```

18s

Parallelised

```
# -------------------------------------------------------
# Get a list of locations
# -------------------------------------------------------
awkc="awk -vFPAT='[^,]*|\"[^\"]*\"' -v OFS=','"
folder="location_of_the_data"
sensors=`ls $folder | head -n 25`


# -------------------------------------------------------
# Set up the processing pipeline
# -------------------------------------------------------
jq_string=".[] | \
  [\"{}\",\
  .timestamp_from_filename,\
  .VendorMacPart+.MacAddress] \
  | @csv";
cmd="jq -r '$jq_string' $folder{}/*.pd \
  | sort | uniq \
  | $awkc '{print \$1,\$2}' \
  | sort | uniq -c";

# -------------------------------------------------------
# Apply the pipeline in parallel over each location
# -------------------------------------------------------
echo "$sensors" \
  | parallel "$cmd" \
  > output.csv
```

3s

Bechmarks

675 Locations

~**31 Mn** records / day

~2 GB / day

> 2 Hours to download

> 5 Hours to process

Bechmarks

675 Locations

~**31 Mn** records / day

~2 GB / day

~ 15 mins

Conclusions

Lots of data != Big data

Evaluate the data for '**bigness**' in each dimension

Use appropriate tools that **do one thing** and **do it well**

Process **streams of data** rather than blobs / files

Parallelise whenever possible

Bespoke toolkit for the data at hand