



Innovation is Freedom

# Introducing Time Series & Warp 10

Horacio Gonzalez  
@LostInBrittany

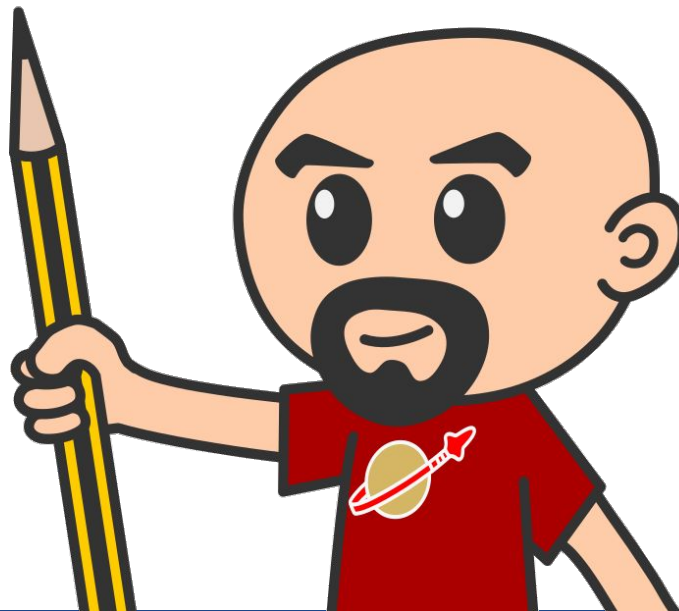


# Horacio Gonzalez



@LostInBrittany

Spaniard lost in Brittany, developer,  
dreamer and all-around geek



# Time Series

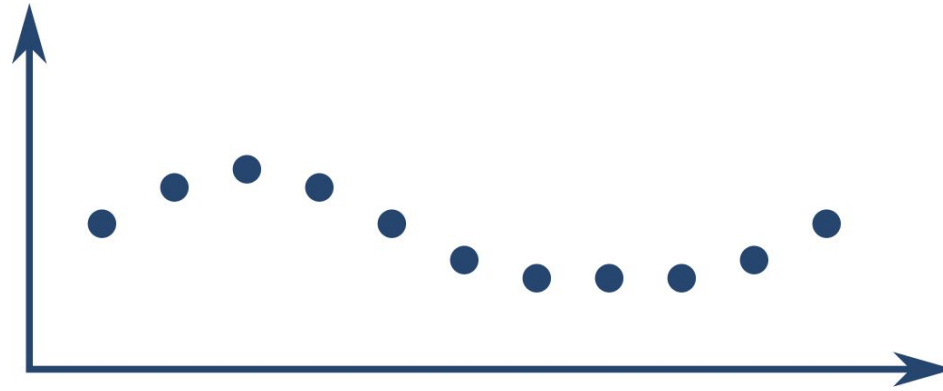
---

## What are they?



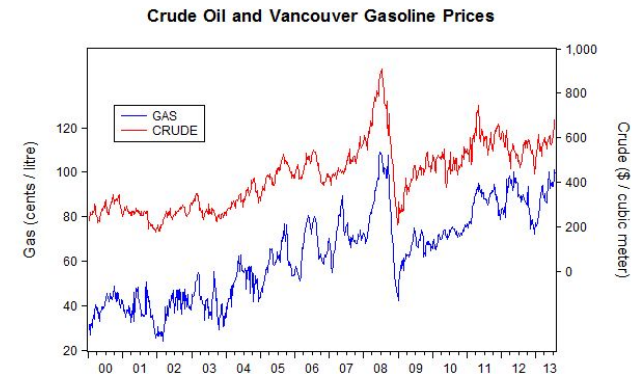
## Definition of Time Series:

An ordered sequence of values of a variable at equally spaced time intervals.



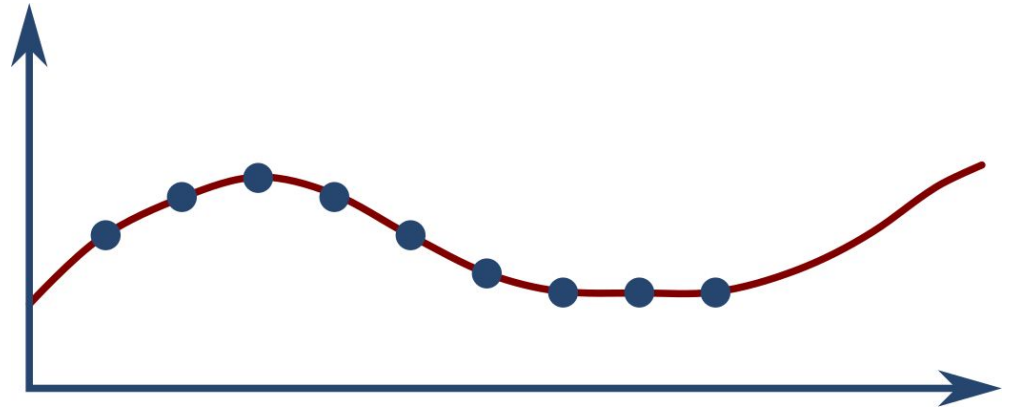
# Time Series

- Stock Market Analysis
- Economic Forecasting
- Budgetary Analysis
- Process and Quality Control
- Workload Projections
- Census Analysis
- ...



## Applications:

- Understanding the data
- Fit a model
  - Monitoring
  - Forecasting



Stock market Analytics  
Economic Forecasting



\$\$\$



Study & Research



## Many specific analytical tools:

- Moving average
- ARMA (AutoRegressive Moving Average)
- Multivariate ARMA models
- ARCH (AutoRegressive Conditional Heteroscedasticity)
- Dynamic time warping
- ...





## Specific application of general tools

- Artificial neural networks
- Hidden Markov model
- Fourier & Wavelets transforms
- Entropy encoding
- ...



# Time Series Databases

---

## What tools do I use?



## General purpose data analytics tools:

- Matlab
- Python
- R
- ...

## General purpose relational database engine



# Time Series Databases

---

Does not scale!



# Time Series Databases

Many type of database engine

- Relational databases
- Key-value databases
- Document databases
- Graph databases
- ...



What about Time Series?

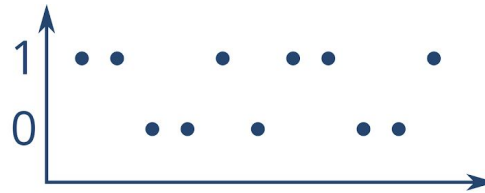
Time Series databases!



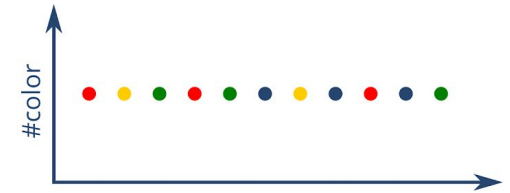
## Data model: time series



Long / Double



Boolean

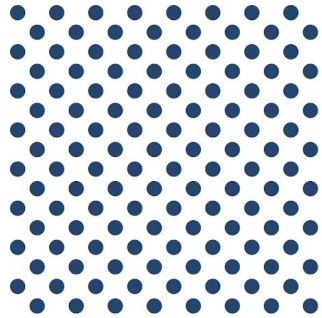


String

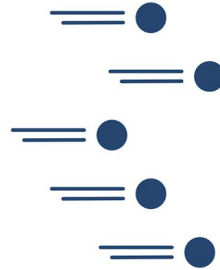


# Time Series Databases

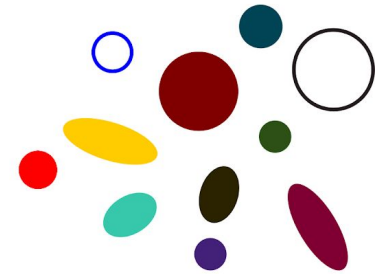
## The 3 'v'



Volume



Velocity



Variety





# Time Series Databases

Many options



# Time Series Databases



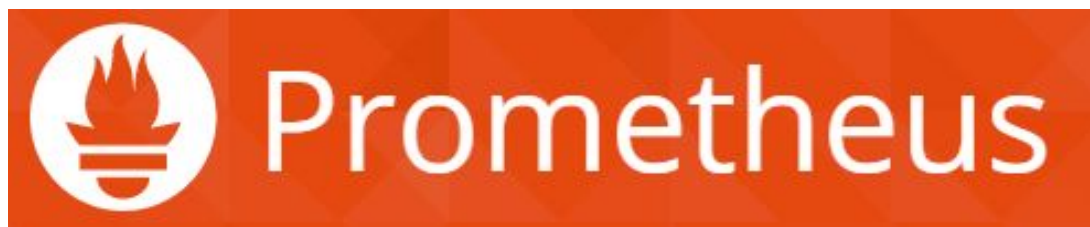
# Time Series Databases



OPENTSDDB



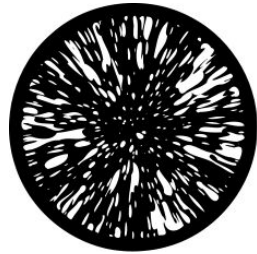
# Time Series Databases



# Time Series Databases



# Time Series Databases



***WARP 10***



# Warp 10

---

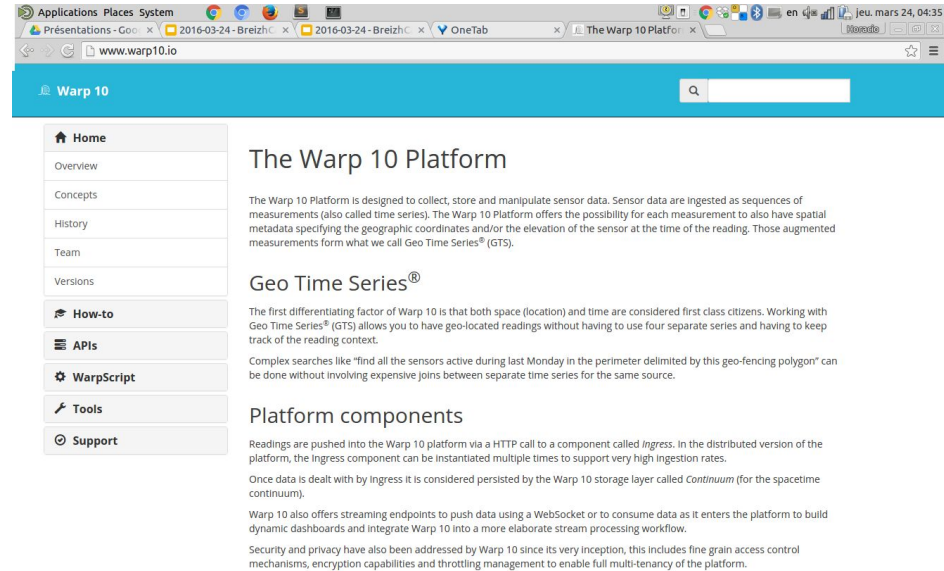
## Open-source Time Series Database



# More than a Time Series DB

Warp 10 is a software platform that

- Ingests and stores time series
- Manipulates and analyzes time series



The screenshot shows a web browser displaying the Warp 10 Platform website. The browser's address bar shows the URL [www.warp10.io](http://www.warp10.io). The website has a blue header with the text "Warp 10" and a search bar. A navigation menu on the left includes links for Home, Overview, Concepts, History, Team, Versions, How-to, APIs, WarpScript, Tools, and Support. The main content area features the title "The Warp 10 Platform" followed by a paragraph describing its purpose: "The Warp 10 Platform is designed to collect, store and manipulate sensor data. Sensor data are ingested as sequences of measurements (also called time series). The Warp 10 Platform offers the possibility for each measurement to also have spatial metadata specifying the geographic coordinates and/or the elevation of the sensor at the time of the reading. Those augmented measurements form what we call Geo Time Series® (GTS)." Below this is the "Geo Time Series®" section, which states: "The first differentiating factor of Warp 10 is that both space (location) and time are considered first class citizens. Working with Geo Time Series® (GTS) allows you to have geo-located readings without having to use four separate series and having to keep track of the reading context." This is followed by a "Platform components" section, which explains: "Readings are pushed into the Warp 10 platform via a HTTP call to a component called *Ingress*. In the distributed version of the platform, the *Ingress* component can be instantiated multiple times to support very high ingestion rates. Once data is dealt with by *Ingress* it is considered persisted by the Warp 10 storage layer called *Continuum* (for the spacetime continuum)." The final paragraph notes: "Warp 10 also offers streaming endpoints to push data using a WebSocket or to consume data as it enters the platform to build dynamic dashboards and integrate Warp 10 into a more elaborate stream processing workflow. Security and privacy have also been addressed by Warp 10 since its very inception, this includes fine grain access control mechanisms, encryption capabilities and throttling management to enable full multi-tenancy of the platform."





# #collect

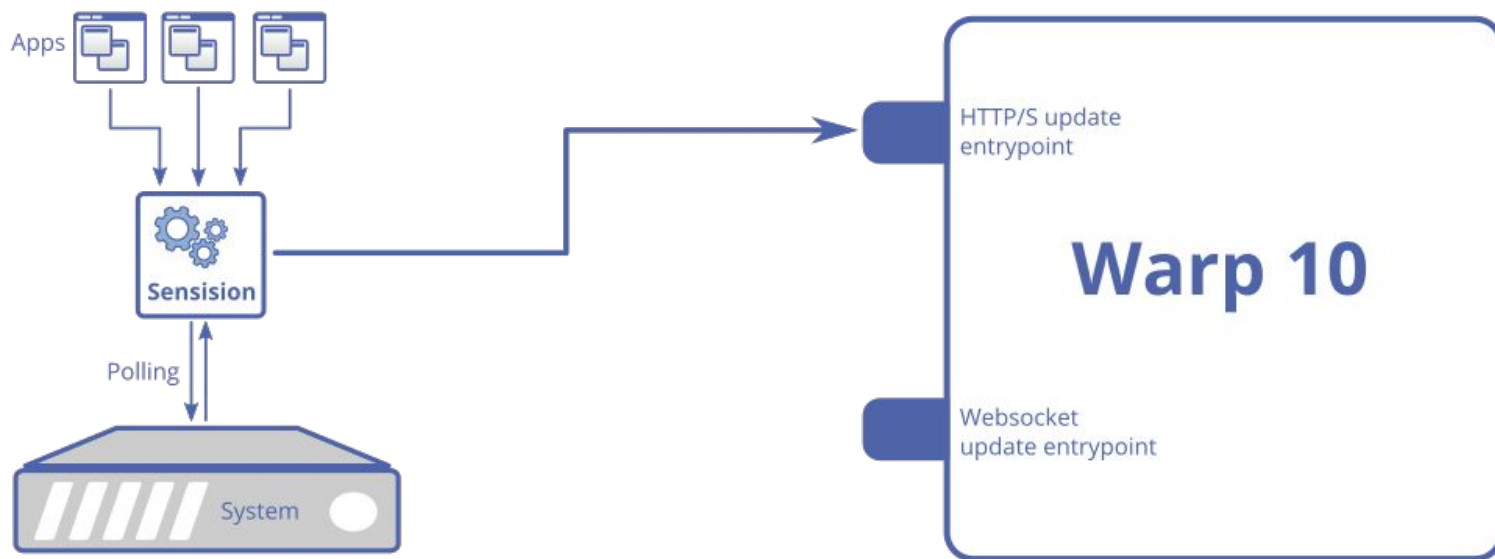
## How do you get these metrics?



Image: [Games Radar](#)



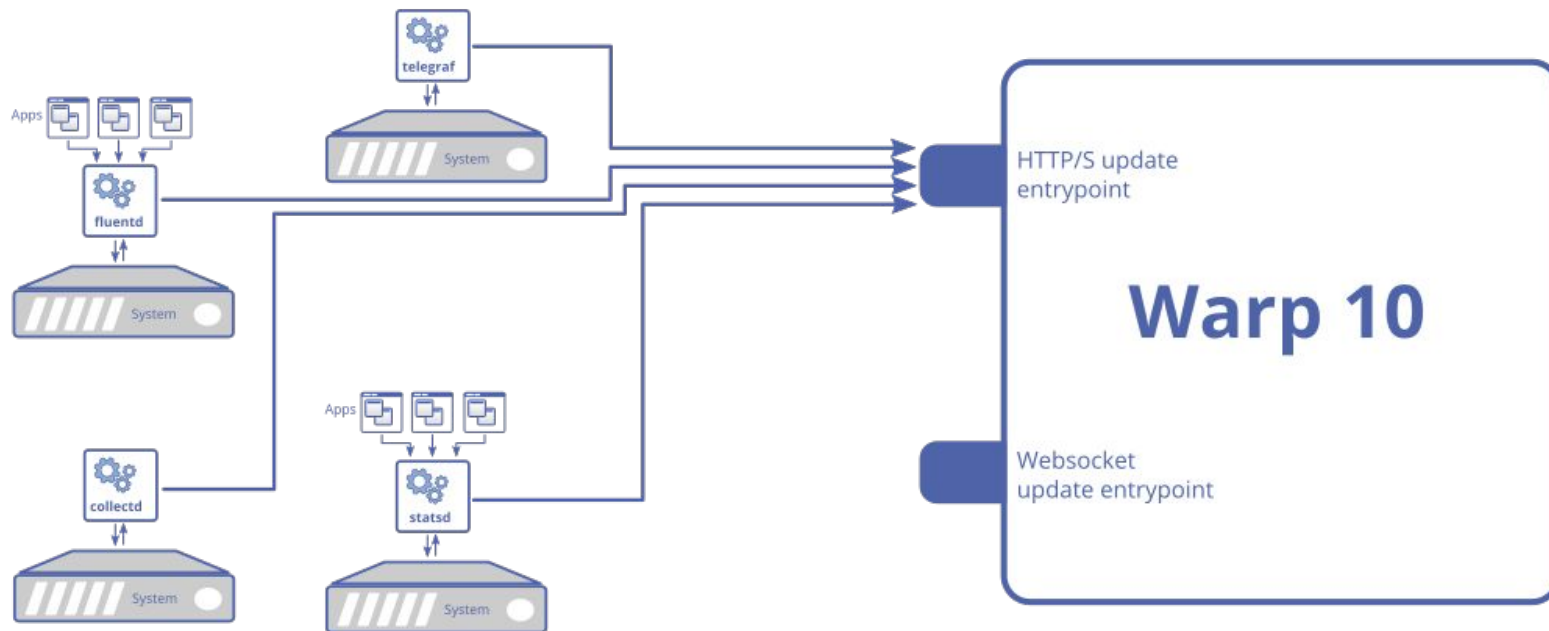
# Collecting data



## Warp 10's Sensision agent



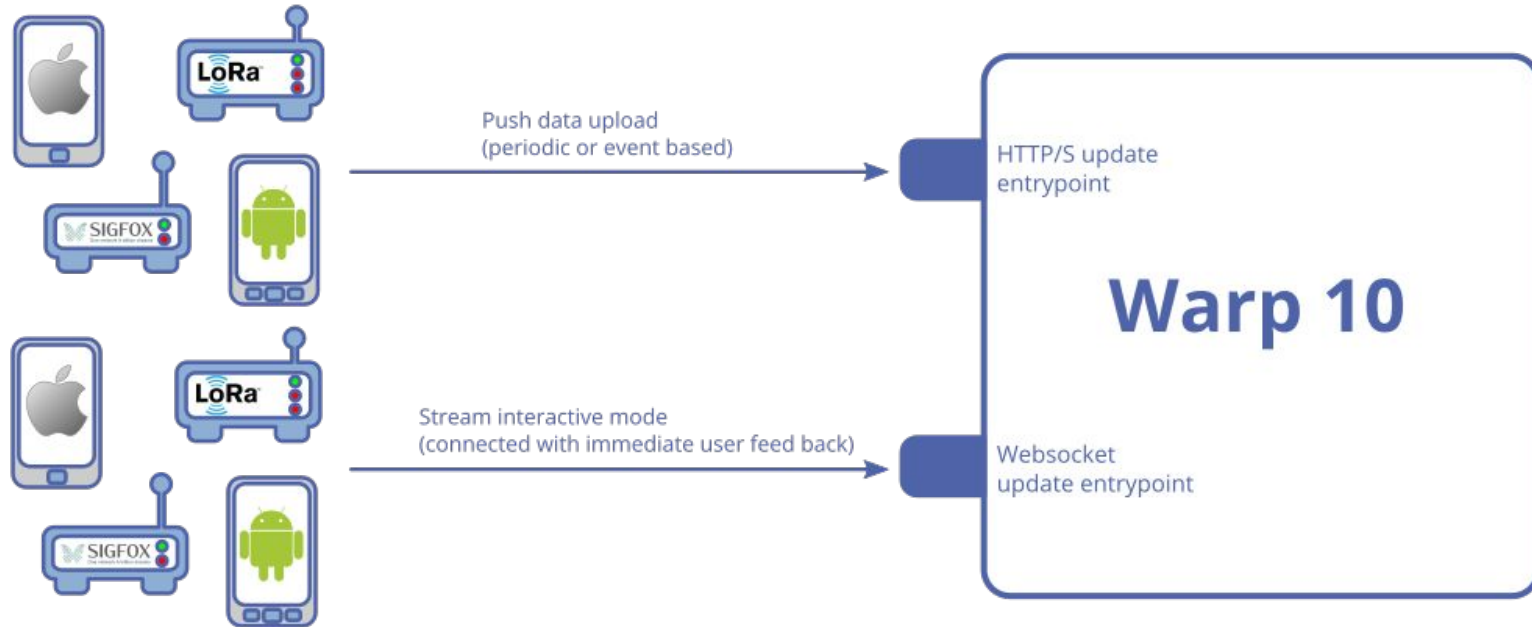
# Collecting data



## 3rd party collectors



# Collecting data



Any HTTP capable device



# Choosing an input format



# XML? JSON?

```
<gts>
  <ts>1457097328123456</ts>
  <lat>45.0</lat>
  <lon>-0.01</lon>
  <e>100</e>
  <c>foo.bar</c>
  <ls>
    <l>
      <k>label1</k>
      <v>val1</v>
    </l>
  </ls>
  <v>4.2</v>
</gts>
```

**139 bytes**

```
{
  "ts": 1457097328123456,
  "lat": 45.0,
  "lon": -0.01,
  "e": 100,
  "c": "foo.bar",
  "l": [
    {
      "k": "label1",
      "v": "val1"
    }
  ],
  "v": 4.2
}
```

**108 bytes**



# Warp 10 GTS Input Format

1457097328123456/45.0:-0.01/100 foo.bar{label1=val1} 4.2

**57 bytes**

But size isn't the most important reason

**parsing time** is way more important

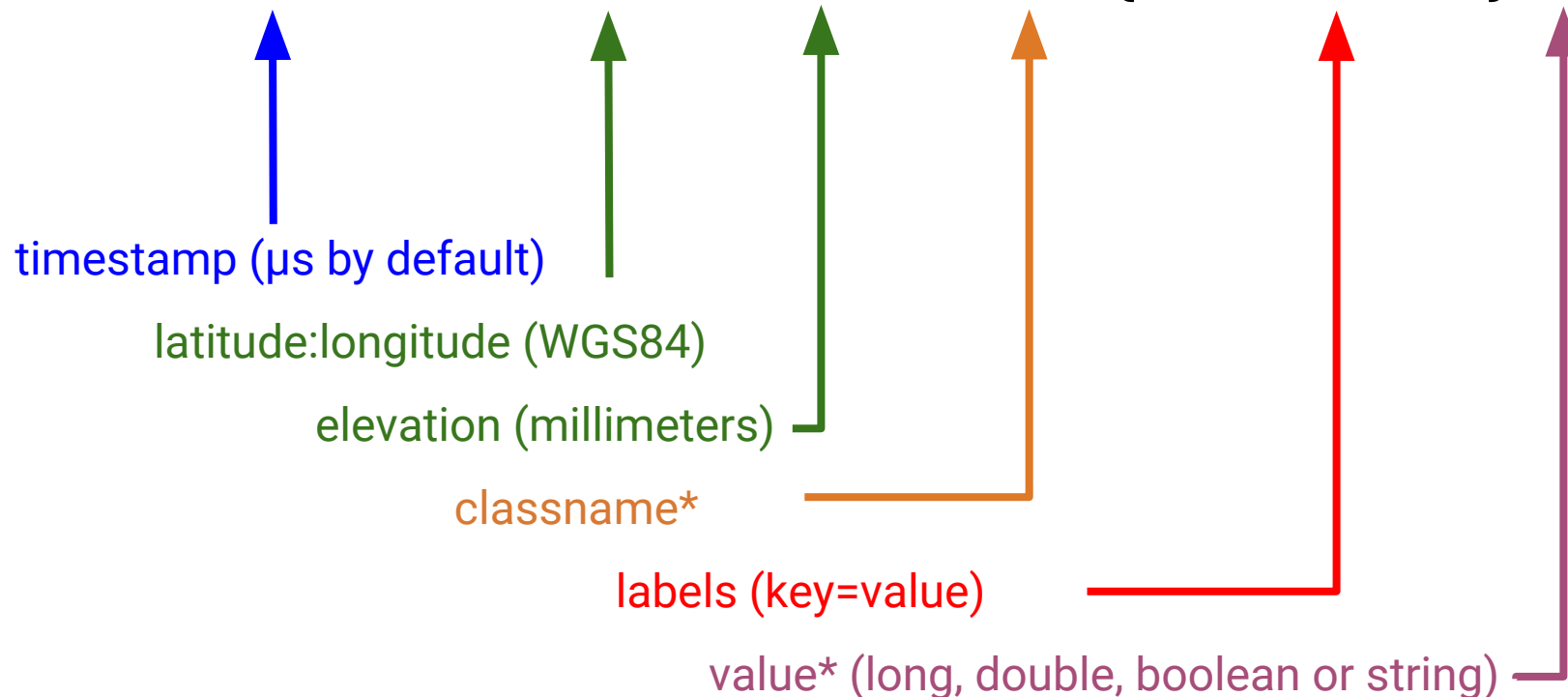
XML or even JSON parsing is slow and costly

Warp 10 GTS input format isn't



# Warp 10 GTS Input Format

1457097328123456/45.0:-0.01/100 foo.bar{label1=val1} 4.2



\* mandatory fields





# #store

---

## From tiny to huge



Image: [Games Radar](#)



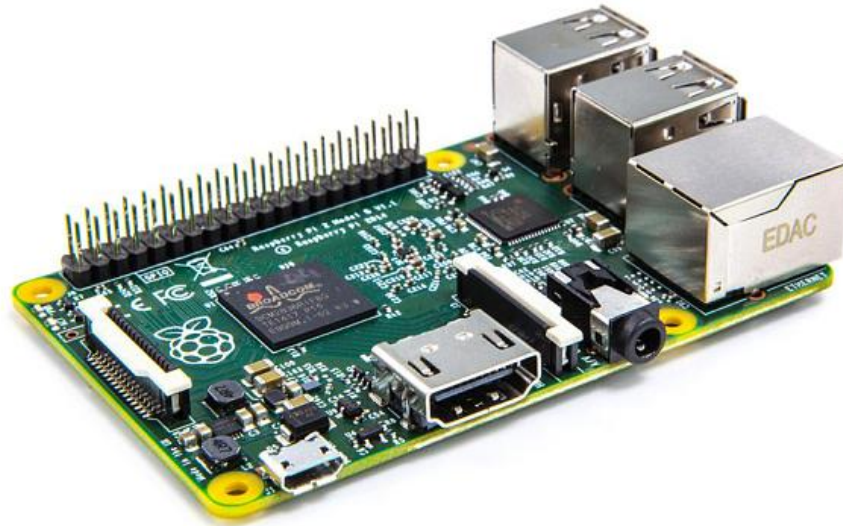
# Warp 10 on Raspberry Pi B+



1 000 datapoints per second



# Warp 10 on Raspberry Pi 2 B



3 000 datapoints per second



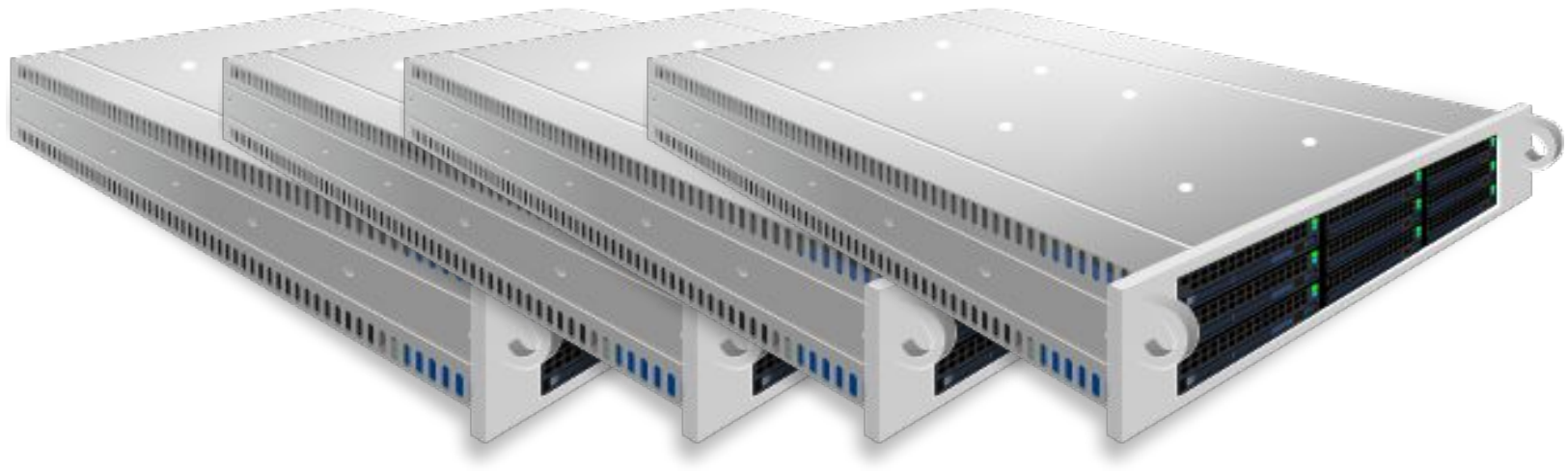
# Warp 10 on a modern server



120 000 datapoints per second



# Warp 10 on a cluster



5 millions of datapoints per second



# #analyse

---

## From tiny to huge



Image: [Amazon](#)



# Many time-series solutions

TSAR



*cassandra*

OpenTSDB



Predix



# But they are only stores...



Fetching data is only the tip of the iceberg





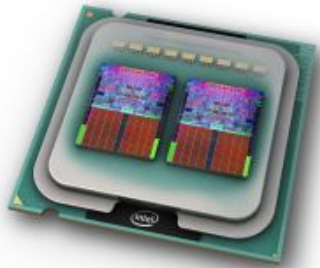
# Analysing the data



High level analysis must be done elsewhere



# Algorithms are resource hungry



# Your computer is not a datacenter



# Manipulating GTS



To be scalable, analysis must be done in  
Warp 10 platform, not in user's computer



## A true GTS analysis toolbox

- Hundreds of functions
- Manipulation frameworks
- Analysis workflow



## Why not a simple REST API?

- One endpoint by function?
- How to chain a workflow analysis?



REST API not suitable for  
complex manipulations



## Why not a SQL dialect?

- How do you do a simple moving average in SQL?
- How do you geo-time fencing in SQL?



SQL is not adapted to (G)TS analysis!



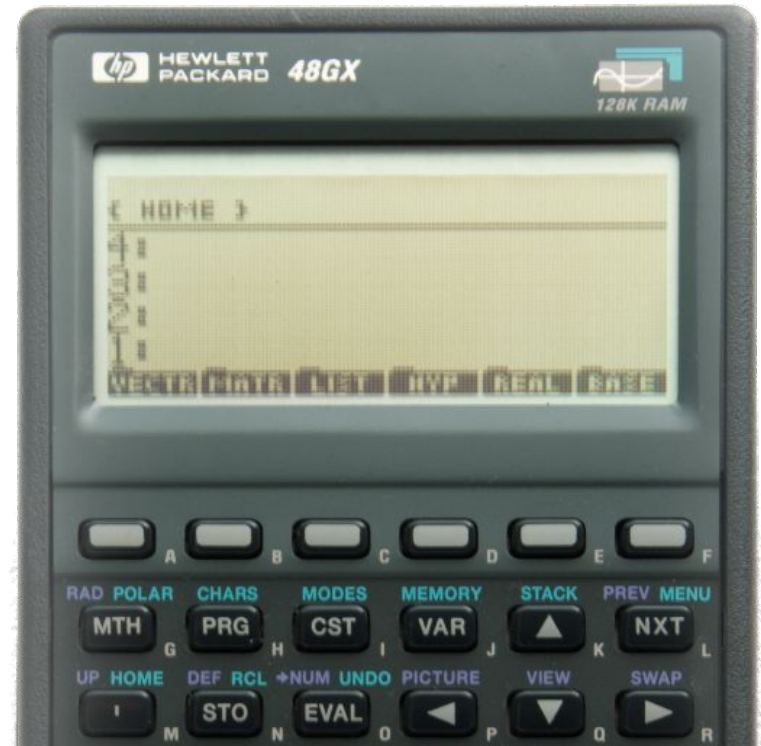
Our solution: a GTS manipulation language





# A stack based language

<b>Input</b>	2	3	add	11	mul	1	add
<b>Stack</b>	2	3		11		1	
	2	2	5	5	55	55	56



# Basic operations

```
// This is a commentary  
'a' // string value  
true // boolean value  
42 // long value  
3.14159 // double value  
+ // operations  
  
20 22 + // several items in one line
```



# Five frameworks

- BUCKETIZE
- MAP
- REDUCE
- FILTER
- APPLY



# More than 800 functions

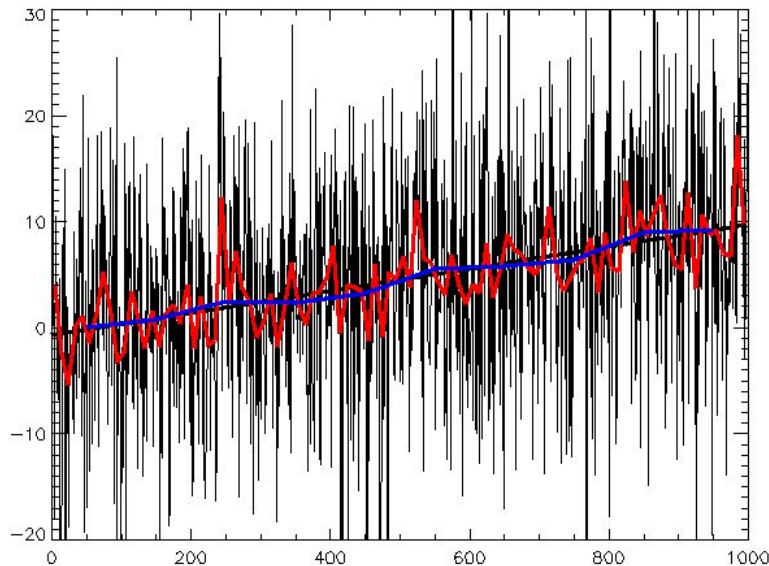
Trigonometry

Data & Time

Macros

List & Maps

Logic  
Structures



Strings

Maths &  
statistics

Loop  
Structures



# Time series functions



TIMECLIP

TIMESPLIT

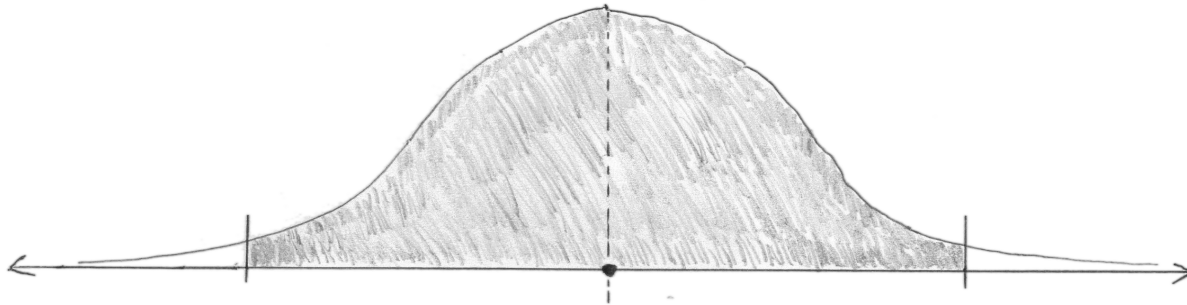
SHRINK

MERGE

...



# Time series functions



MUSIGMA

NORMALIZE

NSUMSUMQ

STANDARDISE

ZSCORE



# Geo-Time Series functions



Geo mapping (WKT)



Horizontal & vertical speed

Horizontal & vertical distance

Haversine

...



 Quantum 

WarpScript
Ingress
Delete

**Choose your backend:**

- Distributed Warp
- Choose another backend

**WarpScript**

```
1
2 // This is a commentary
3 'a' // string value
4 true // boolean value
5 42 // long value
6 3.14159 // double value
7 + // operations
8
9 20 22 + // several items in one line
10
```

**Permalink:**  
[Ci8vIFRoaXMgaXMgYsBjb21tZW50YXJ5CidhJyAgCS8vIHNoCmluZyB2YWx1ZQp0cnVlIAkvLyBib29s...](https://ovh.com/quantum/ide/Ci8vIFRoaXMgaXMgYsBjb21tZW50YXJ5CidhJyAgCS8vIHNoCmluZyB2YWx1ZQp0cnVlIAkvLyBib29s...)

[Execute!](#)





# Enough teasing...



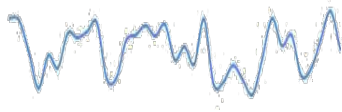
# Fuel prices data



data.gouv.fr IOVH  
ion is Freedom



11 379 fuel stations



42 885 Geo Time Series



16 297 448 metrics



# Basic analysis

## Average diesel fuel prices in France since 2007



Image: [LEGO Ideas](#)



# First Fetch Data (SQL vs WarpScript )

```
SELECT 'gazole_price', 'station_id'  
FROM opendata_fuel WHERE fuel_date between  
'2007-01-01T00:00:00.000000Z' AND  
'2015-12-31T23:59:59.999999Z'
```

---

```
[  
  'API TOKEN'  
  'opendata.fuel' { 'type' 'gazole' }  
  '2007-01-01T00:00:00.000000Z'  
  '2015-12-31T23:59:59.999999Z'  
] FETCH
```



# FETCH gives us a GTS list

```
[
  {
    "c": "data.fuel", "l": { "id": "7380001", "type": "gazole", ".app": "gazoline" },
    "v": [ [1451452320000000, 45.526001, 5.974660, 1.159], [1450501920000000, 45.526001, 5.974660, 1.169], ... ]
  },
  {
    "c": "data.fuel", "l": { "id": "4130001", "type": "gazole", ".app": "gazoline" },
    "v": [ [1450771934000000, 47.423929, 2.053439, 0.988], [1450426320000000, 47.423929, 2.053439, 0.999], ... ]
  },
  {
    "c": "data.fuel", "l": { "id": "4080005", "type": "gazole", ".app": "gazoline" },
    "v": [ [1419685260000000, 43.732249, -0.238590, 1.249], [1419426000000000, 43.732249, -0.238590, 1.259], ... ]
  },
  {
    "c": "data.fuel", "l": { "id": "1216002", "type": "gazole", ".app": "gazoline" },
    "v": [ [1449220339000000, 44.278179, 2.429619, 1.2], [1446631880000000, 44.278179, 2.429619, 1.22], ... ]
  },
  {
    "c": "data.fuel", "l": { "id": "8549001", "type": "gazole", ".app": "gazoline" },
    "v": [ [1451560370000000, 46.369959, -0.594400, 0.969], [1451459537000000, 46.369959, -0.594400, 0.969], ... ]
  },
  ...
]
```



# FETCH gives us a GTS list

```
[
  {
    "c": "data.fuel", "l": { "id": "7380001", "type": "gazole", ".app": "gazoline" },
    "v": [ [1451452320000000, 45.526001, 5.974660, 1.159], [1450501920000000, 45.526001, 5.974660, 1.169], ... ]
  },
  {
    "c": "data.fuel", "l": { "id": "4130001", "type": "gazole", ".app": "gazoline" },
    "v": [ [1450771934000000, 47.423929, 2.053439, 0.988], [1450426320000000, 47.423929, 2.053439, 0.999], ... ]
  },
  {
    "c": "data.fuel", "l": { "id": "4080005", "type": "gazole", ".app": "gazoline" },
    "v": [ [1419685260000000, 43.732249, -0.238590, 1.249], [1419426000000000, 43.732249, -0.238590, 1.259], ... ]
  },
  {
    "c": "data.fuel", "l": { "id": "1216002", "type": "gazole", ".app": "gazoline" },
    "v": [ [1449220339000000, 44.278179, 2.429619, 1.2], [1446631880000000, 44.278179, 2.429619, 1.22], ... ]
  },
  ...
]
```

## Timestamp

(microseconds since epoch)



# FETCH gives us a GTS list

```
[
  {
    "c": "data.fuel", "l": { "id": "7380001", "type": "gazole", ".app": "gazoline" },
    "v": [ [1451452320000000, 45.526001, 5.974660, 1.159], [1450501920000000, 45.526001, 5.974660, 1.169], ... ]
  },
  {
    "c": "data.fuel", "l": { "id": "4130001", "type": "gazole", ".app": "gazoline" },
    "v": [ [1450771934000000, 47.423929, 2.053439, 0.988], [1450426320000000, 47.423929, 2.053439, 0.999], ... ]
  },
  {
    "c": "data.fuel", "l": { "id": "4080005", "type": "gazole", ".app": "gazoline" },
    "v": [ [1419685260000000, 43.732249, -0.238590, 1.249], [1419426000000000, 43.732249, -0.238590, 1.259], ... ]
  },
  {
    "c": "data.fuel", "l": { "id": "1216002", "type": "gazole", ".app": "gazoline" },
    "v": [ [1449220339000000, 44.278179, 2.429619, 1.2], [1446631880000000, 44.278179, 2.429619, 1.22], ... ]
  },
  ...
]
```

## Location

(latitude, longitude)  
@LostInBrittany



# FETCH gives us a GTS list

```
[
  {
    "c": "data.fuel", "l": { "id": "7380001", "type": "gazole", ".app": "gazoline" },
    "v": [ [1451452320000000, 45.526001, 5.974660, 1.159], [1450501920000000, 45.526001, 5.974660, 1.169], ... ]
  },
  {
    "c": "data.fuel", "l": { "id": "4130001", "type": "gazole", ".app": "gazoline" },
    "v": [ [1450771934000000, 47.423929, 2.053439, 0.988], [1450426320000000, 47.423929, 2.053439, 0.999], ... ]
  },
  {
    "c": "data.fuel", "l": { "id": "4080005", "type": "gazole", ".app": "gazoline" },
    "v": [ [1419685260000000, 43.732249, -0.238590, 1.249], [1419426000000000, 43.732249, -0.238590, 1.259], ... ]
  },
  {
    "c": "data.fuel", "l": { "id": "1216002", "type": "gazole", ".app": "gazoline" },
    "v": [ [1449220339000000, 44.278179, 2.429619, 1.2], [1446631880000000, 44.278179, 2.429619, 1.22], ... ]
  },
  ...
]
```

Value





# Calculate the average

## Using Groovy:

```
int sumValue = 0
int counter = 0
for (def station: stations) {
    int stationSum = 0
    int stationCounter = 0
    for (def value: values) {
        sumValue += value.price
        counter ++
        stationSum += value.price
        stationCounter ++
    }
    station.meanPrice = stationSum / stationCounter
}
```

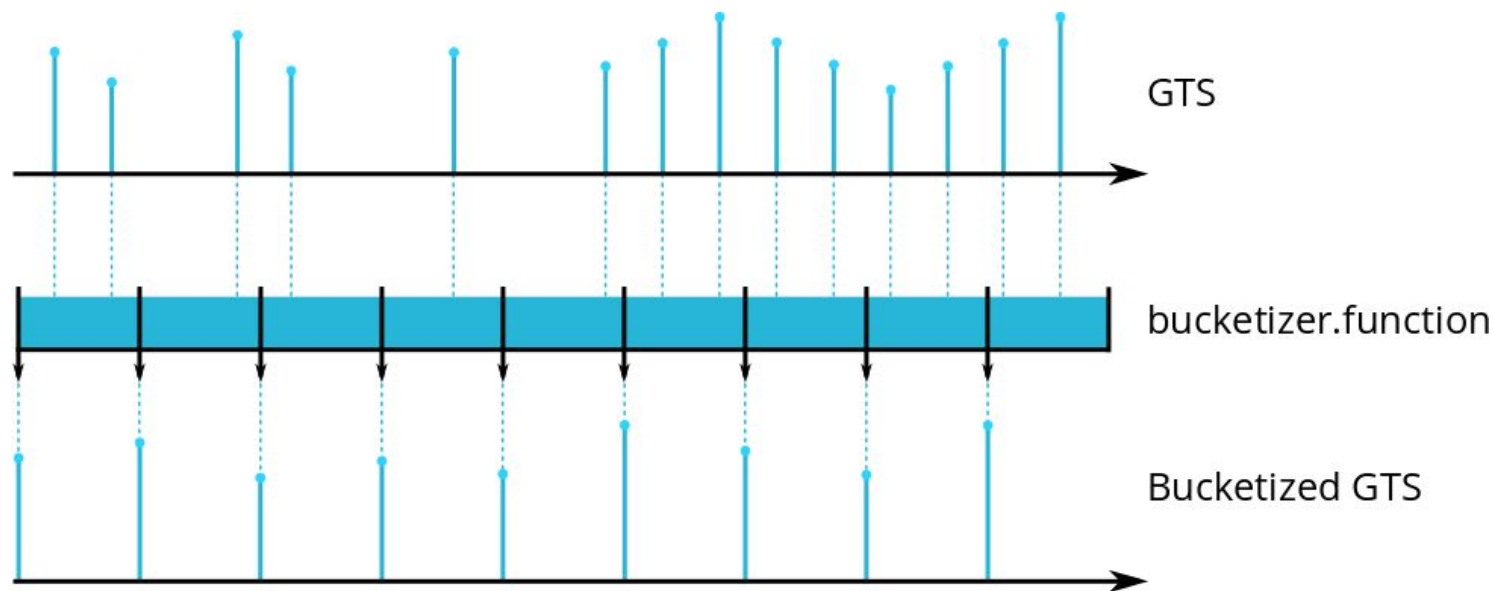


## 1- Calculate the mean price by station

```
// mean fuel price by station
[
  $gts          // gts list from fetch
  bucketizer.mean // average computing
  NOW          // align all buckets
  0           // bucket span auto
  1           // number of buckets
] BUCKETIZE
```



## BUCKETIZE framework



Put the data of a GTS into regularly spaced buckets



# Calculate the average with WarpScript

```
[
  {
    "c": "data.fuel", "l": { "id": "7380001", "type": "gazole", ".app": "gazoline" },
    "v": [ [1457451425216748, 45.526001, 5.974660, 1.164] ]
  },
  {
    "c": "data.fuel", "l": { "id": "4130001", "type": "gazole", ".app": "gazoline" },
    "v": [ [1457451425216748, 47.423929, 2.053439, 0.9935] ]
  },
  {
    "c": "data.fuel", "l": { "id": "4080005", "type": "gazole", ".app": "gazoline" },
    "v": [ [1457451425216748, 43.732249, -0.238590, 1.254] ]
  },
  {
    "c": "data.fuel", "l": { "id": "1216002", "type": "gazole", ".app": "gazoline" },
    "v": [ [1457451425216748, 44.278179, 2.429619, 1.21] ]
  },
  {
    "c": "data.fuel", "l": { "id": "8549001", "type": "gazole", ".app": "gazoline" },
    "v": [ [1457451425216748, 46.369959, -0.594400, 0.969] ]
  },
  ...
]
```



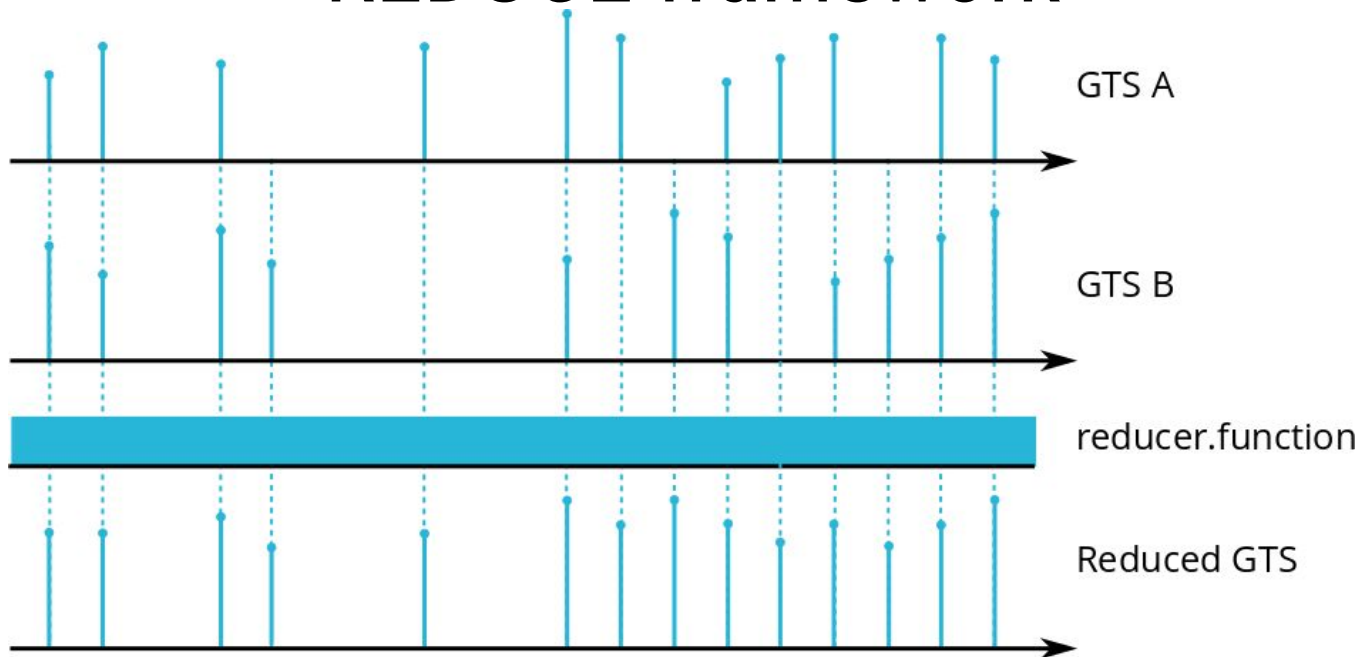
## 2- Reduce to get the global average

```
// mean fuel price for all stations  
[] // use all labels  
reducer.mean // mean function  
3 ->LIST // to list  
REDUCE // execute reducer
```

```
0: [{"c":"","l":{"type":"gazole",".app":"prixdescarburants"},"a":{"v":  
[[1451606399999999,48.09619389474392,0.7937734387814999,1.1012307692307692]]}]}
```



## REDUCE framework



Apply a function on a set of GTS tick by tick



# Too verbose? Write it differently

```
// mean fuel price for all stations
[
  [ $gts bucketizer.mean NOW 0 1 ] BUCKETIZE
  [ ]
  reducer.mean
] REDUCE
```

```
0: [{"c":"","l":{"type":"gazole",".app":"prixdescarburants"},"a":{"v":
[[145160639999999,48.09619389474392,0.7937734387814999,1.1012307692307692]]}]}
```



# Even more concise

```
// mean fuel price by station
[ $gts bucketizer.mean NOW 0 1 ] BUCKETIZE
// mean fuel price for all stations
[ SWAP [ ] reducer.mean ] REDUCE
```

```
0: [{"c":"","l":{"type":"gazole",".app":"prixdescarburants"},"a":{"},"v":
[[145160639999999,48.09619389474392,0.7937734387814999,1.1012307692307692]]}]
```





# Basic analysis

## Mean of the last available diesel fuel prices in France



Image: [LEGO Ideas](#)



# Fetching Data (SQL vs WarpScript )

```
SELECT t1.* FROM opendata_fuel t1
JOIN (
  SELECT 'gazole_price', 'station_id', MAX(fuel_date)
  FROM opendata_fuel t2
  GROUP BY 'station_id')
ON t1.station_id = t2.station_id AND
   t1.fuel_date = t2.fuel_date
```

---

```
[
  'API TOKEN'
  'opendata.fuel' { 'type' 'gazole' }
  NOW -1
] FETCH
```



# FETCH gives us a GTS list

```
[
  {
    "c": "data.fuel", "l": { "id": "7380001", "type": "gazole", ".app": "gazoline" }, "a": {},
    "v": [[1451452320000000, 45.526001, 5.974660, 1.159]]
  },
  {
    "c": "data.fuel", "l": { "id": "4130001", "type": "gazole", ".app": "gazoline" }, "a": {},
    "v": [[1450771934000000, 47.423929, 2.053439, 0.988]]
  },
  {
    "c": "data.fuel", "l": { "id": "4080005", "type": "gazole", ".app": "gazoline" }, "a": {},
    "v": [[1419685260000000, 43.732249, -0.238590, 1.249]]
  },
  {
    "c": "data.fuel", "l": { "id": "1216002", "type": "gazole", ".app": "gazoline" }, "a": {},
    "v": [[1449220339000000, 44.278179, 2.429619, 1.2]]
  },
  {
    "c": "data.fuel", "l": { "id": "8549001", "type": "gazole", ".app": "gazoline" }, "a": {},
    "v": [[1451560370000000, 46.369959, -0.594400, 0.969]]
  },
  {
    "c": "data.fuel", "l": { "id": "3970001", "type": "gazole", ".app": "gazoline" }, "a": {},
    "v": [[1420002013000000, 47.117552, 5.540754, 1.072]]
  },
  ...
]
```



# Mean of those last prices

align ticks with **BUCKETIZE** framework

compute the average with **REDUCE**

```
0: [{"c":"","l":{"type":"gazole",".app":"prixdescarburants"},"a":{"v":  
[[145743307757713,46.52869887650013,2.8918789699673653,1.0454783074450689]]}]
```

X



# Geo-time analysis

---

Find the cheapest fuel station near here

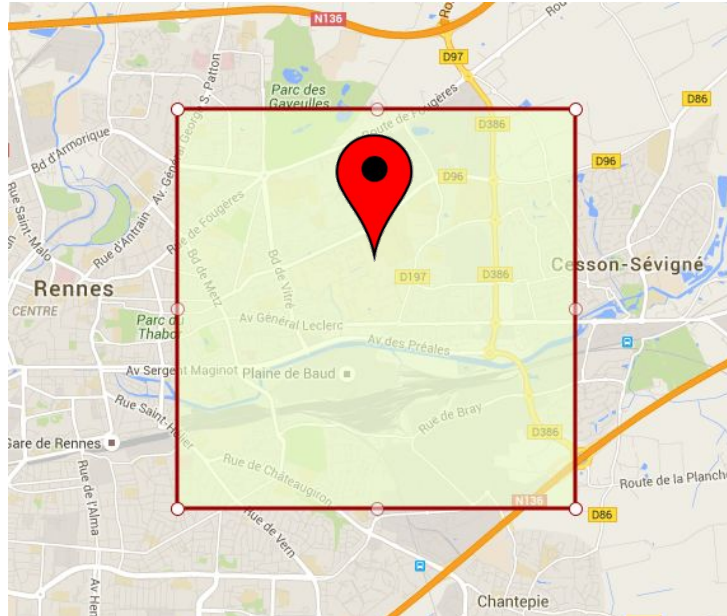


48.115434, -1.636877



# geometry

```
POLYGON (( -1.66378 48.13340, -1.60996 48.13340,  
-1.60996 48.09746, -1.66378 48.09746,  
-1.66378 48.13340))
```



# WKT in WarpScript

```
// converts WKT into geographical zone  
'POLYGON (( ... ))'  
0.1 // error percentage  
true // true for inside  
GEO.WKT
```



# Geo-filtering points of GTS

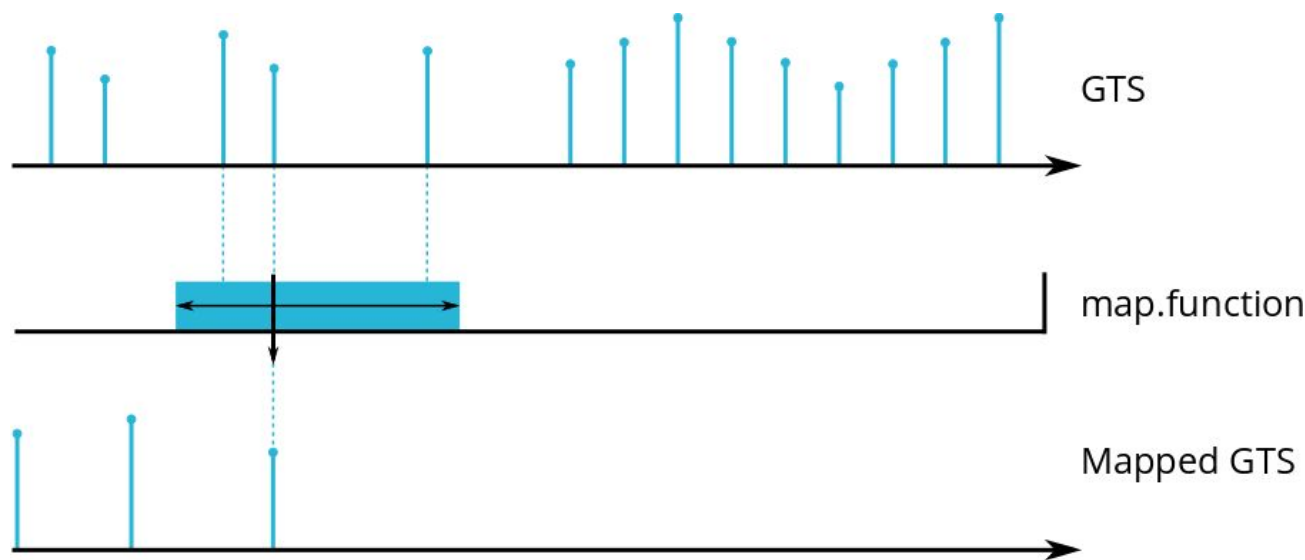
```
[
  $gts           // GTS list
  $geozone       // from GEO.WKT
  Mapper.geo.within // keep inner points
  0              // pre
  0              // post
  0              // occurrences
] MAP NONEMPTY
```





# Geo-filtering points of GTS

## MAPPER framework



Apply a function on values of a GTS  
that fall into a sliding window



# The stations near my position

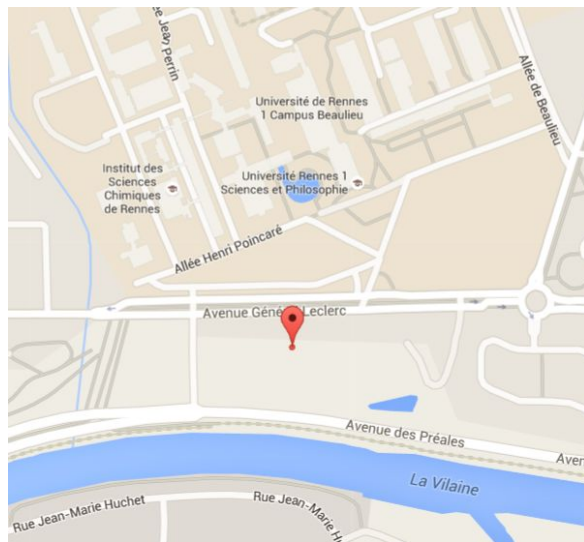
```
[
  {
    "c": "data.fuel", "l": {"id": "35000020", "type": "gazole", ".app": "gazoline"}, "a": {},
    "v": [[1451517120000000, 48.113564, -1.637989, 0.962]]
  }, {
    "c": "data.fuel", "l": {"id": "35000022", "type": "gazole", ".app": "gazoline"}, "a": {},
    "v": [[1451517120000000, 48.127478, -1.650935, 0.979]]
  }, {
    "c": "data.fuel", "l": {"id": "35000004", "type": "gazole", ".app": "gazoline"}, "a": {},
    "v": [[1419919968000000, 48.113564, -1.637989, 1.069]]
  }, {
    "c": "data.fuel", "l": {"id": "35000003", "type": "gazole", ".app": "gazoline"}, "a": {},
    "v": [[1419878728000000, 48.127478, -1.650935, 1.068]]
  }
]
```



# There can only be one

VALUESORT // Sort by value  
0 GET // Take the first

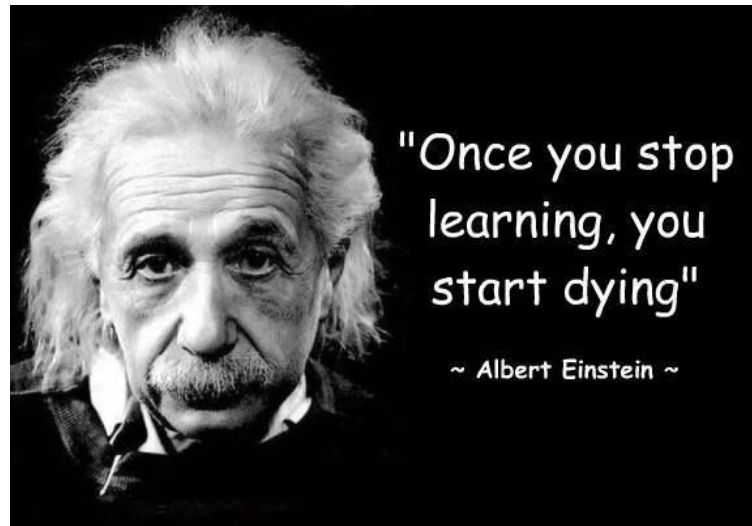
```
0: {"c":"data.fuel","l":{"id":"35000020","type":"gazole",".app":"prixdescarburants"},"a":{"},"v":  
[[1451517120000000,48.113564173690975,-1.6379893384873867,0.962]]}
```



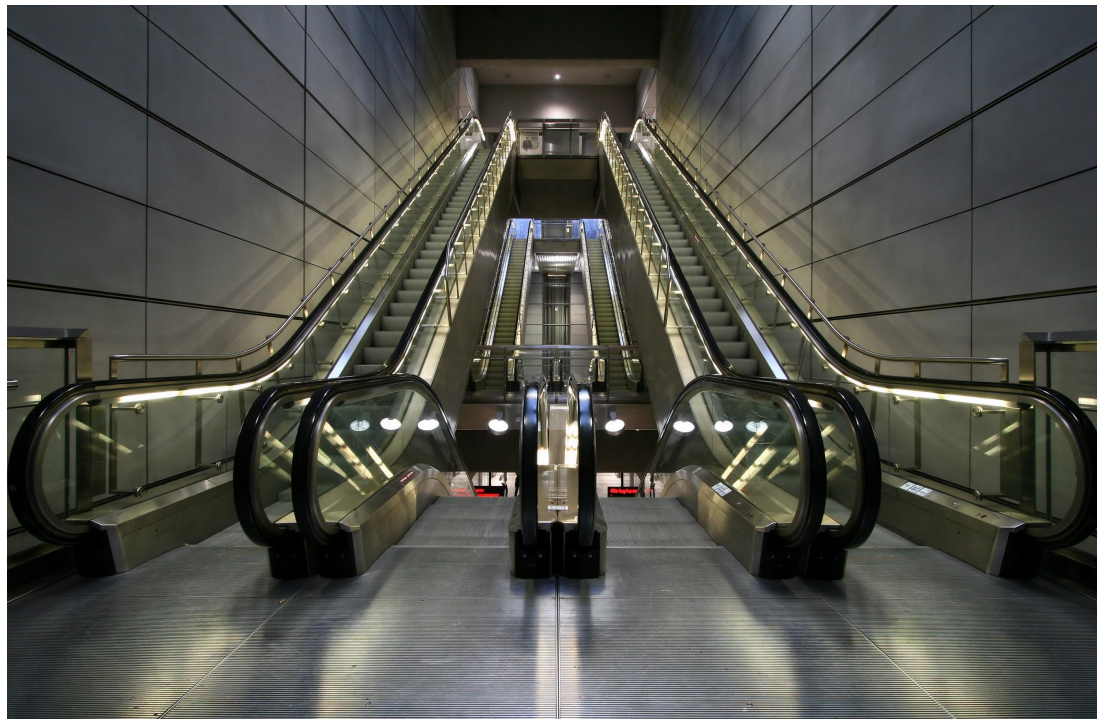
# And this is only the surface

---

## Possibilities are endless



# Think differently



Geo-Time Series are everywhere



# Warp 10 platform and tools



```
WarpScript
// GET PROFILES
[
  $data_Fuel?
  [ "type" "geojson" ]
  $map ""
]
RETURN "geo" STORE

// BRIGGS CAMP IS HERE
[[ 114488 -14.66667 ]]
ZPLIST ONLY STATIONS AROUND ME (5000 meters)
2009
[[ GET2_BROUENCOUX ]]
[[ FROM GET2_BROUENCOUX ]]
[ $data_Sensor? mapper-geo-walkin @ @ ] MAP NODATA?
VALIDDATE @ GET ?
```

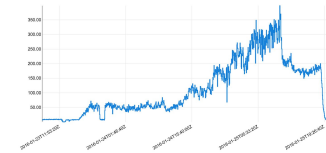
Permalink: [2JnaUjUuNDuRQI?BEALZJMmZ4VERWVWfYfYmuhedf65EdYmMOZP4LYGImeZEPfQESKvZSRMqH...](#)

Plot Execute!

Your script execution took 2.903 s server-side

Response:

```
0: [ {"type": "Point", "coordinates": [ 114488, -14.666666666666667 ], "properties": { "name": "BROUENCOUX" } }, { "type": "Point", "coordinates": [ 114517, -14.666666666666667 ], "properties": { "name": "BROUENCOUX" } } ]
```



 Warp 10



# OVH Metrics

---

## What did we choose?



# What's OVH Metrics

Managed Cloud Platform for Geo Time Series®





# What's a metric?

[**me**-trik] : the science of measuring



# What's a metric?

“To measure is to know”

William “Lord Kelvin” Thomson



# What's a metric?

Actionable metric  $\neq$  vanity metric



# What is a metric?

Metrics are Time Series!



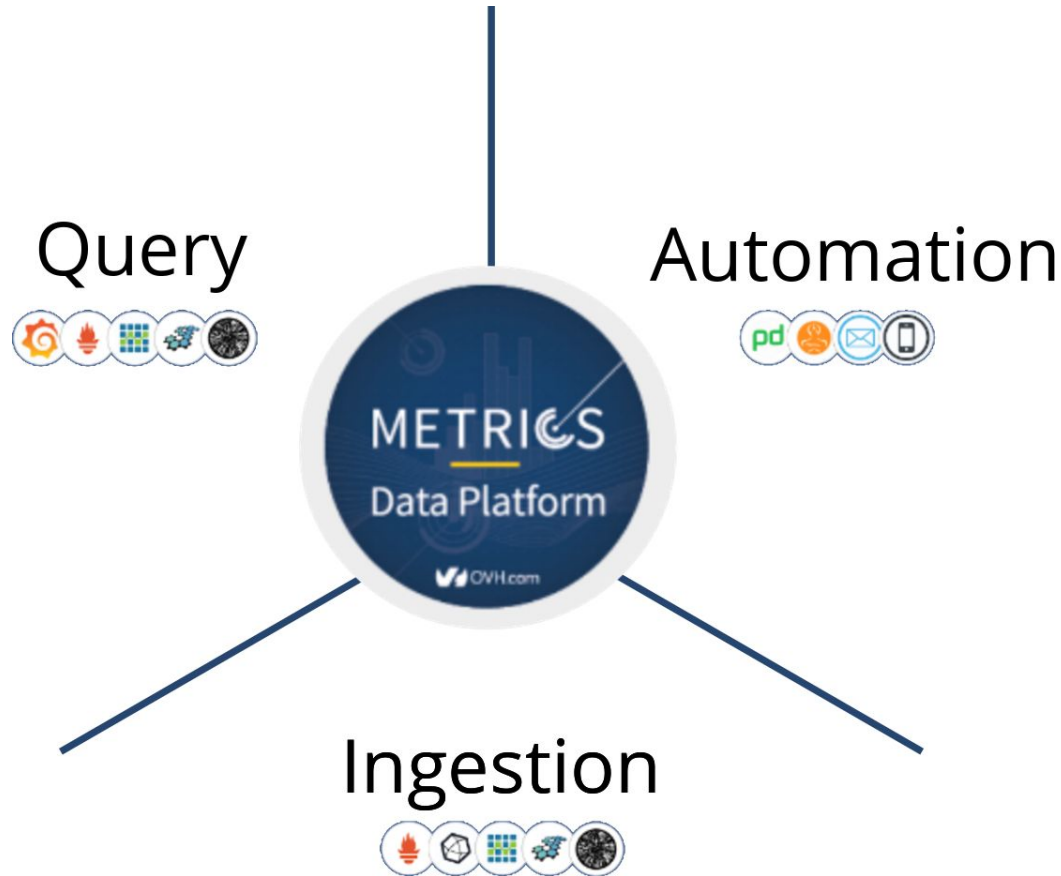
Using a Time Series Database!

But... which one?

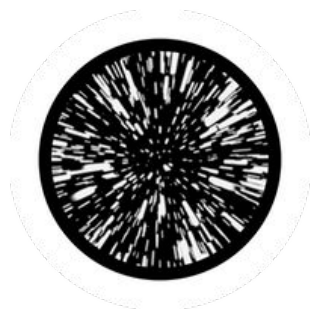
Why choose? Let's support all of them!



# Metrics Data Platform



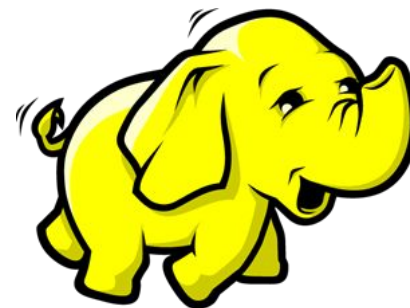
# Metrics Data Platform



+



+



# Conclusion

---

**That's all folks!**







