Stream Processing

Lecture 6

2022/2023

Table of Contents

- Storage for Big Data
  - File systems
    - HDFS
  - Databases
    - Key-Value stores
    - Time-series databases
- IoT

Context

- Big data systems need to store huge amounts of data
- Cloud platforms need to be elastic and fault tolerant, supporting the addition and removal of nodes
  – Storage systems must support the same features
- Traditional storage systems are not adequate for such settings

Table of Contents

- Storage for Big Data
  - **File systems**
    - **HDFS**
  - Databases
    - Key-Value stores
    - Time-series databases
- IoT

HDFS

- HDFS is a distributed file system used extensively for storing data to be processed with Hadoop, Spark, etc.
- Design derived from Google File System (GFS).

Goals of HDFS

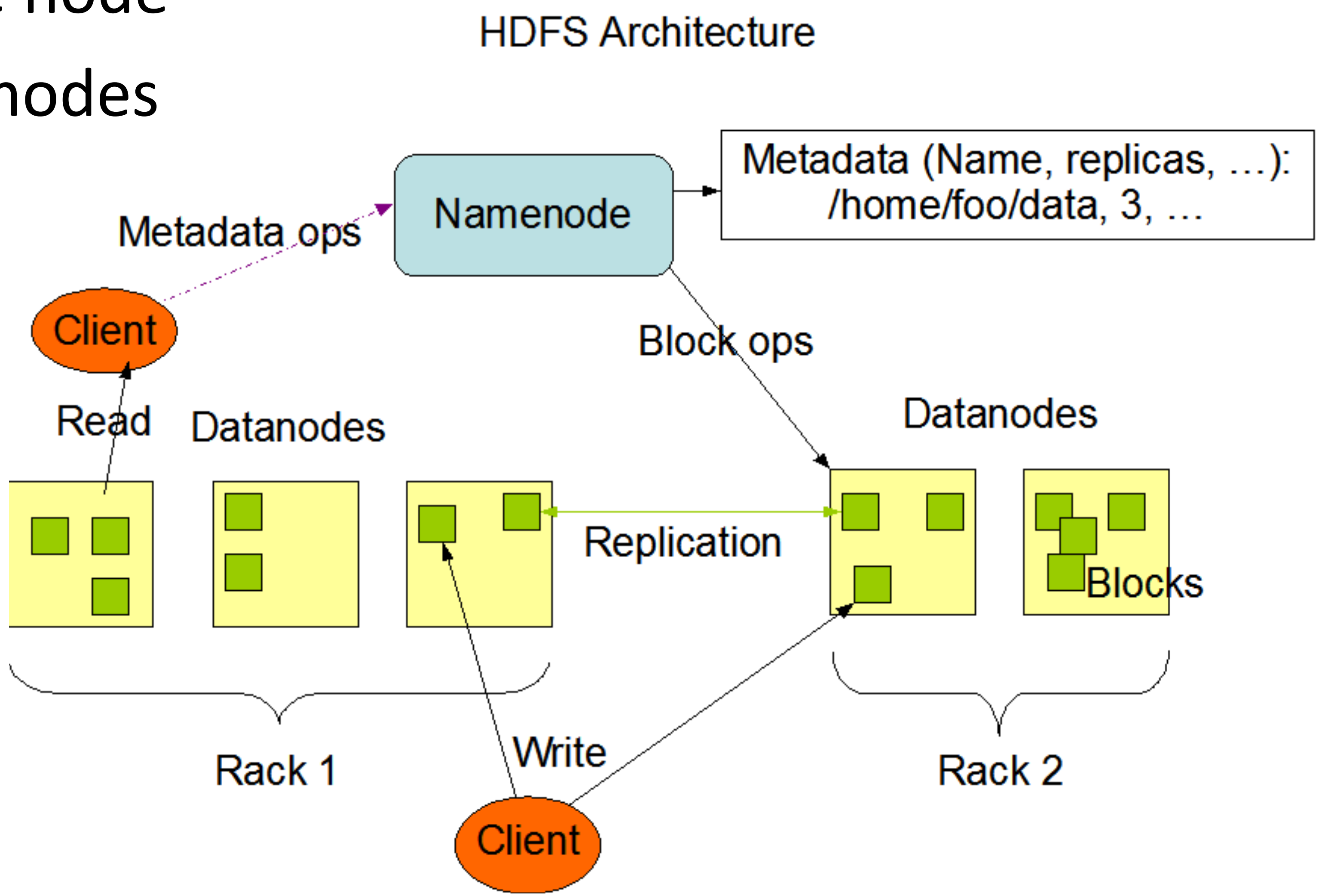
- Very Large Distributed File System
  - 10K nodes, 100 million files, 10PB
- Assumes Commodity Hardware
  - Files are replicated to handle hardware failure
- Optimized for Batch Processing
  - Data locations exposed so that computations can move to where data resides
  - Provides very high aggregate bandwidth

HDFS Model

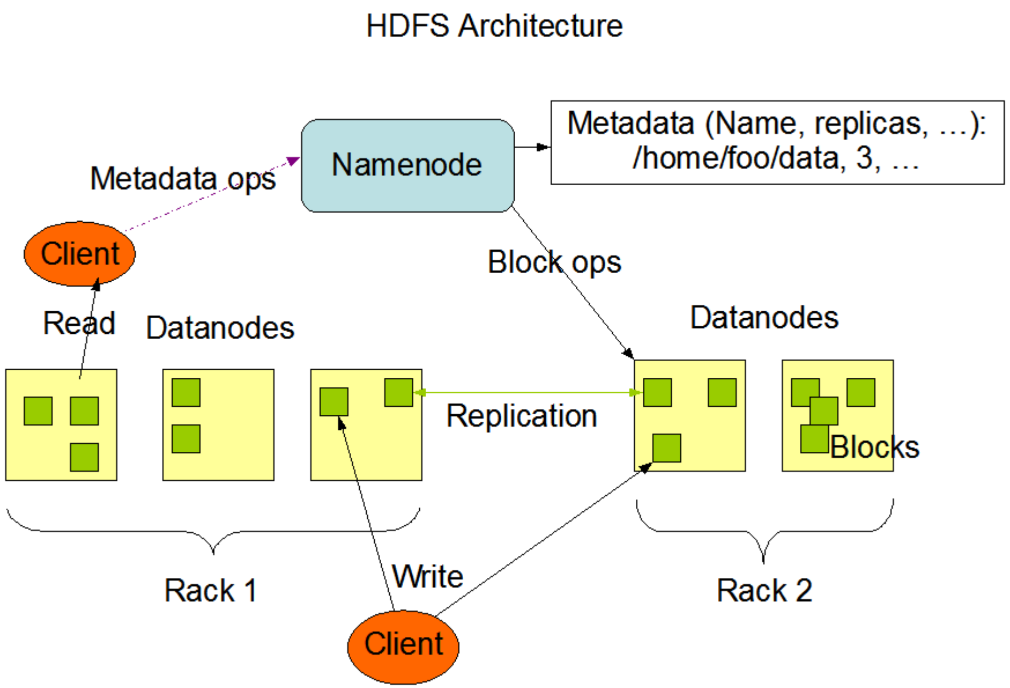
- Single Namespace for entire cluster
- Data Coherency
  - Write-once-read-many access model
  - Client can only append to existing files
- Files are broken up into blocks
  - Typically 64MB block size
  - Each block replicated on multiple DataNodes

HDFS Architecture

- Name node
- Data nodes

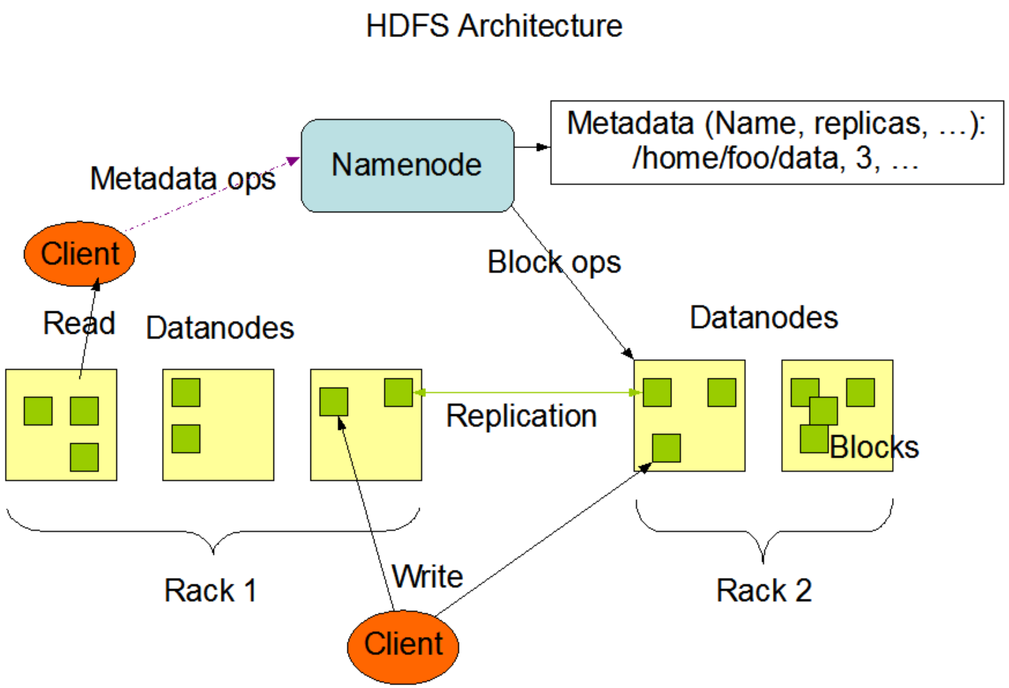


HDFS Architecture: Name Node



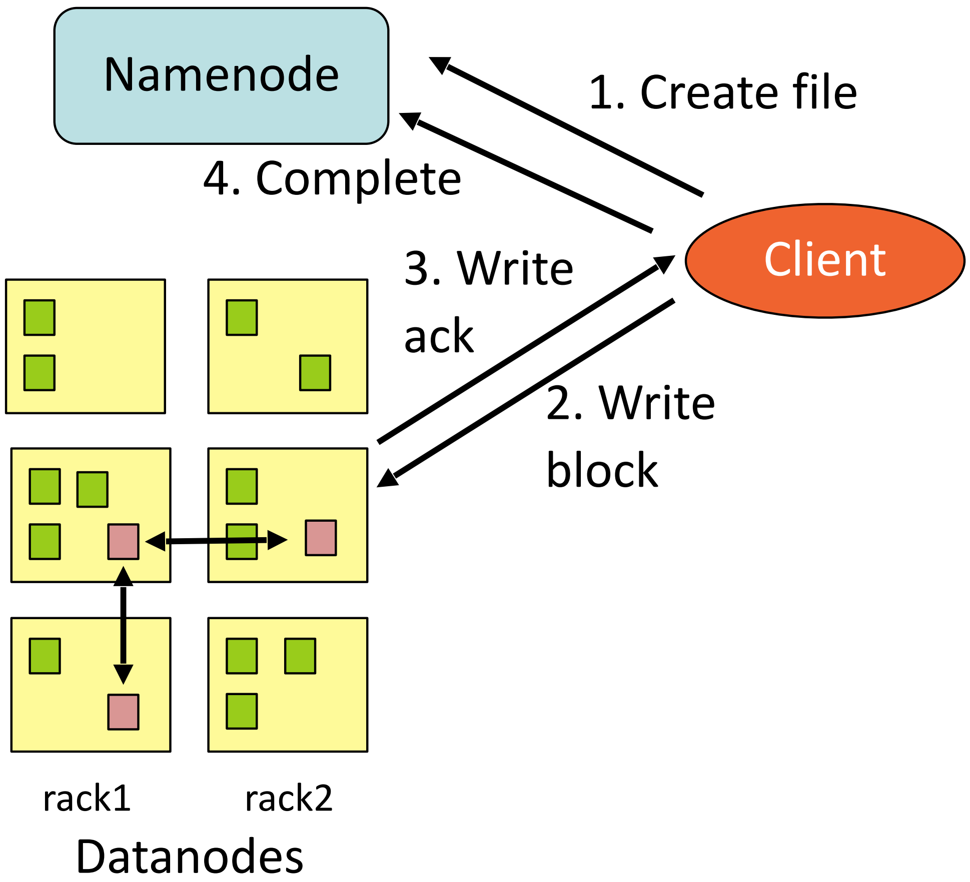
- Manages File System Namespace
  - Maps a file name to a set of blocks
  - Maps a block to the DataNodes where it resides
- Cluster Configuration Management
- Replication Engine for Blocks

HDFS Architecture: Data Nodes



- A Block Server
  - Stores data in the local file system (e.g. ext3)
  - Stores metadata of a block (e.g. CRC)
  - Serves data and metadata to Clients
- Block Report
  - Periodically sends a report of all existing blocks to the NameNode

HDFS Architecture: Write File



1. Create file on NameNode /get information on where to write block (when available)
2. Write block to data on data node
   - Write is replicated in a pipeline
   - Replicated in multiple racks (default: 1 local, 2 remote)
3. Ack returned to client when write complete in a quorum of replicas
4. Notifies the NameNode that write was completed

Replication Engine

- NameNode detects DataNode failures
  - Chooses new DataNodes for new replicas
  - Balances disk usage
  - Balances communication traffic to DataNodes

NameNode Failure

- A single point of failure
- Transaction Log stored in multiple directories
  - A directory on the local file system
  - A directory on a remote file system (NFS/CIFS)

- Several solutions for high availability of the NameNode have been proposed.

Amazon S3

- Object store, with flat namespace
  - Can emulate hierarchical namespace by using names with structure.
- Used as a replacement for file systems
  - E.g. storing static objects in a web site.
- Provides high availability, by storing objects at multiple replicas (in multiple devices and facilities in a given region)
  - Supports for inter-region replication.

Filesystem Events

- HDFS and Amazon S3 include monitoring subsystems
  - It is possible to interface to these subsystems to process these notifications to drive applications

  - Eg. It is possible to monitor files being created/ deleted/replicated and use a processing framework to generate custom reports.

Table of Contents

- Storage for Big Data
  - File systems
    - HDFS
  - **Databases**
    - **Key-Value stores**
    - Time-series databases
- IoT

Key-value databases

- Data model: data is stored as key-value pairs.
- API (variants exist):
  – get(key) -> value
  – put(key, value)
- Some systems provide secondary indexes for faster retrieval of data.

- Simpler model (compared to RDB SQL) simplifies scalable designs.

- Examples: Cassandra, DynamoDB.

Key-value databases

- Interfacing with Processing Streams
  - Connectors allow KV Databases to be used as **sinks** for stream processing results
  - When programmable triggers are available, such as in Cassandra, it is possible to feed KV DB events to Kafka; acting as the **source** of stream processing applications

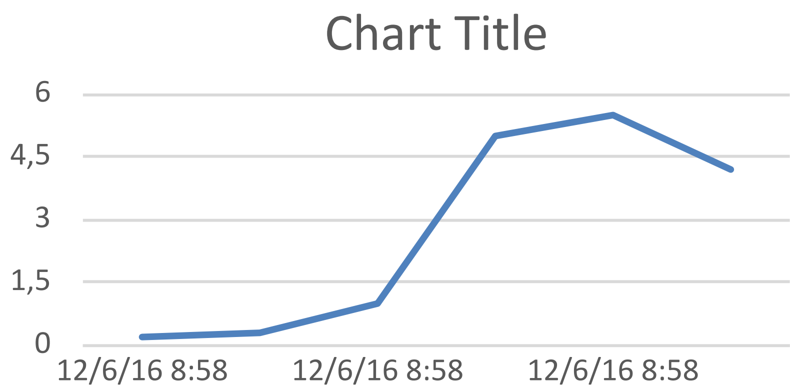
Table of Contents

- Storage for Big Data
  - File systems
    - HDFS
  - **Databases**
    - Key-Value stores
    - **Time-series databases**
- IoT

What are time-series?

- A "Time Series is an ordered sequence of values of a variable (e.g. temperature) with an associated timestamp".
  - Time series can be obtained at equally spaced time intervals or not.
- "Sequence of discrete-time data, ordered on a timeline."
- "Time series data are simply measurements or events that are tracked, monitored, downsampled, and aggregated over time".

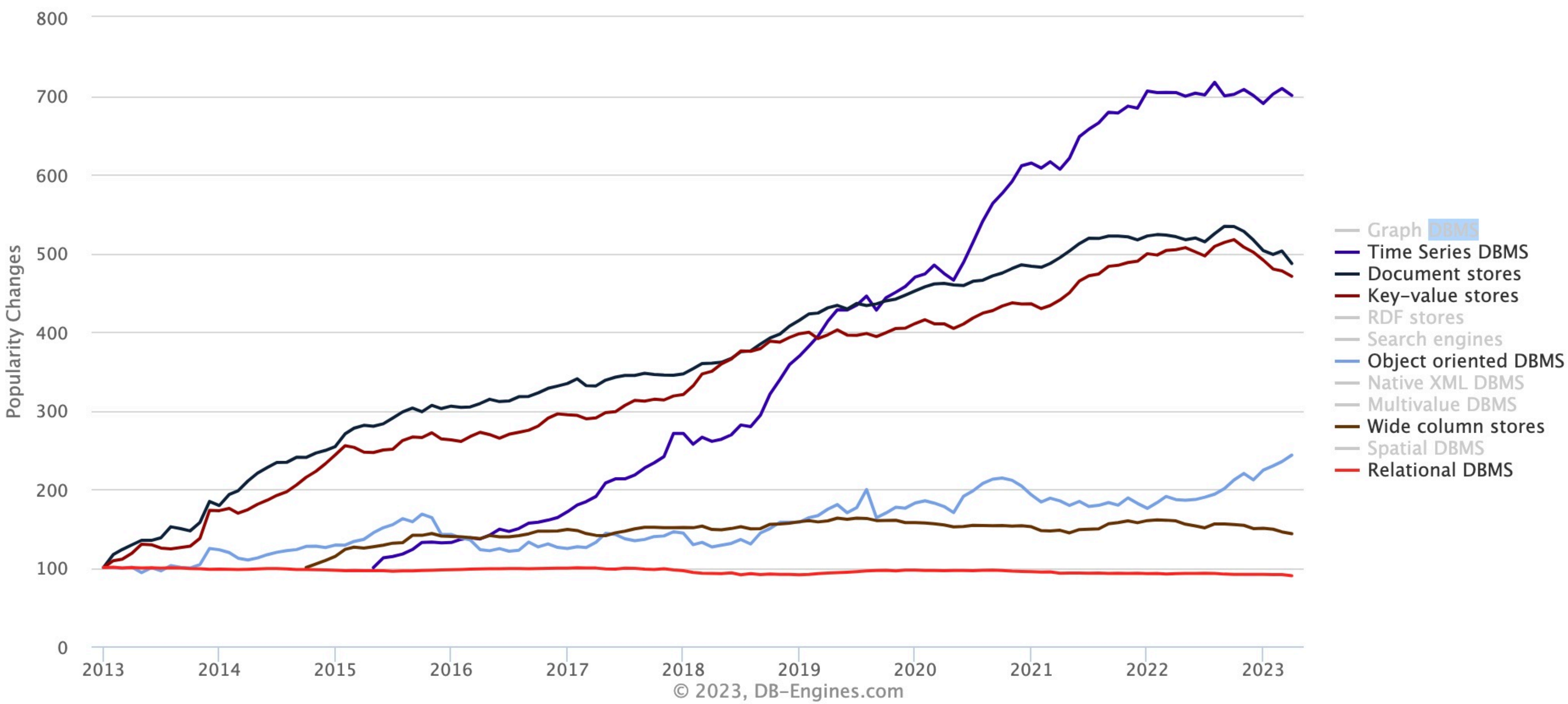
Timestamp	Value
2016-12-06 08:58:00	0.2
2016-12-06 08:58:05	0.3
2016-12-06 08:58:10	1.0
2016-12-06 08:58:15	5.0
2016-12-06 08:58:17	5.5
2016-12-06 08:58:20	4.2



Why are time series important?

- First-generation time series focused mainly on financial markets.
- Current drivers:
  - Monitoring of computing infrastructures in a cluster: performance monitoring, network data;
  - Monitoring of physical world – IoT, sensor networks, etc.
- Emergence of Time-series Databases (TSDB)

Time-series databases popularity



Requirements: writes dominate

- It should always be possible to execute writes.
- Write scale is huge - example from server monitoring

  2,000 servers, VMs, containers, or sensor units

  1,000 measurements per server/unit
  every 10 seconds

  = 17,280,000,000 distinct points per day
- Read scale is smaller
  – E.g. Facebook Gorilla reports “couple orders of magnitude lower”
  – Automated systems watching “important” time series
  – Dashboards for humans
  – Human operators wishing to diagnose an observed problem

Requirements: state transitions

- Identify issues that occur on monitored data.
- TSDB should support fine-grained aggregations over short-time windows.
- TSDB should have the ability to identify state transitions within tens of seconds.

Requirements: high availability and fault tolerance

- TSDB should support write and reads even in the presence of network partitions.
- TSDB should replicate data to survive server failure.

Other requirements

- ACID guarantees are not a requirement, but...
- ...high percentage of writes must succeed at all times (**some may fail**... typically not a problem under high load). Why?
- ... recent data is of higher value than older data.

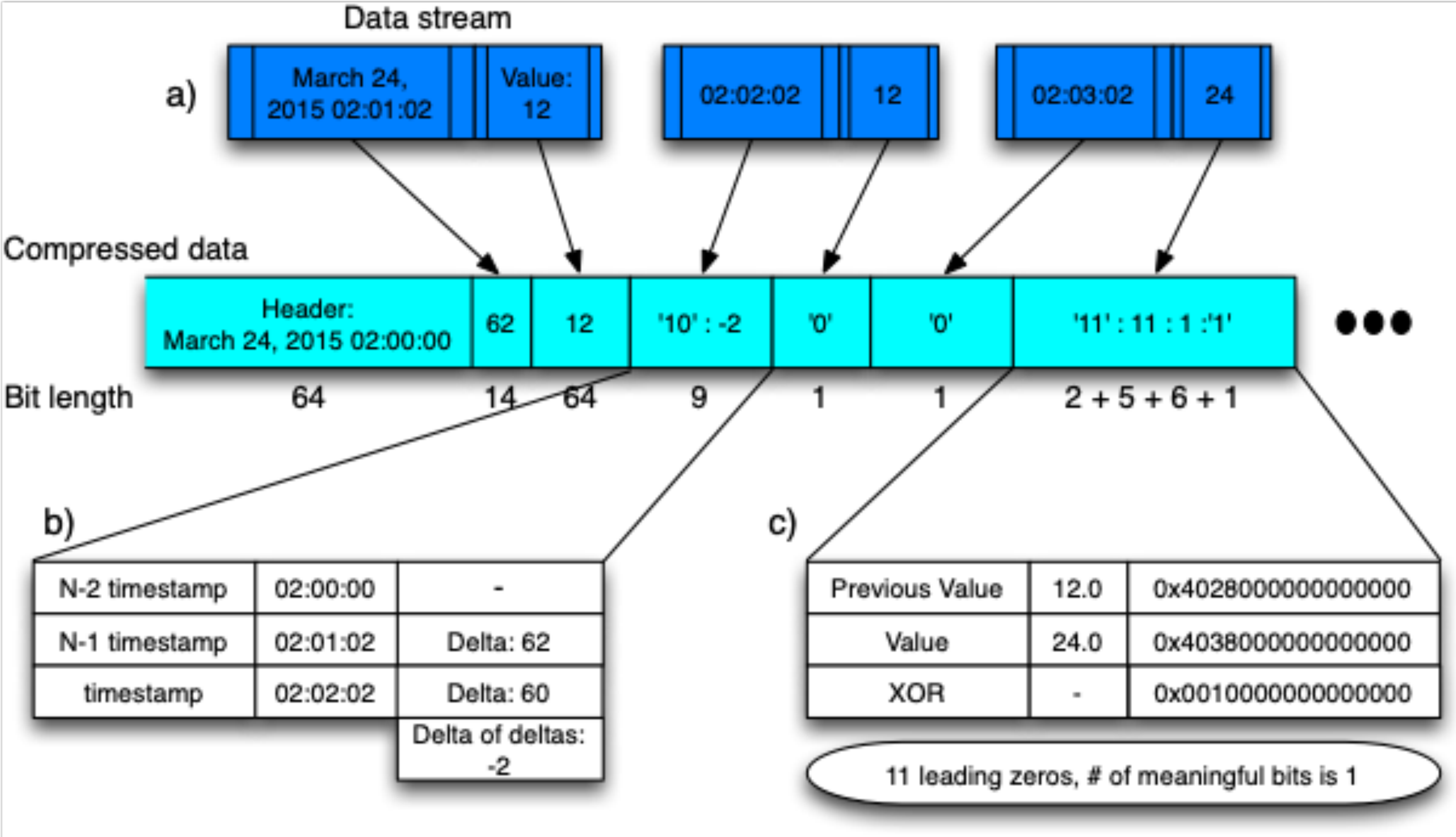
Design of a TSDB

- Problem: scale of data is enormous
- Solution: compression of the data

Time series compression

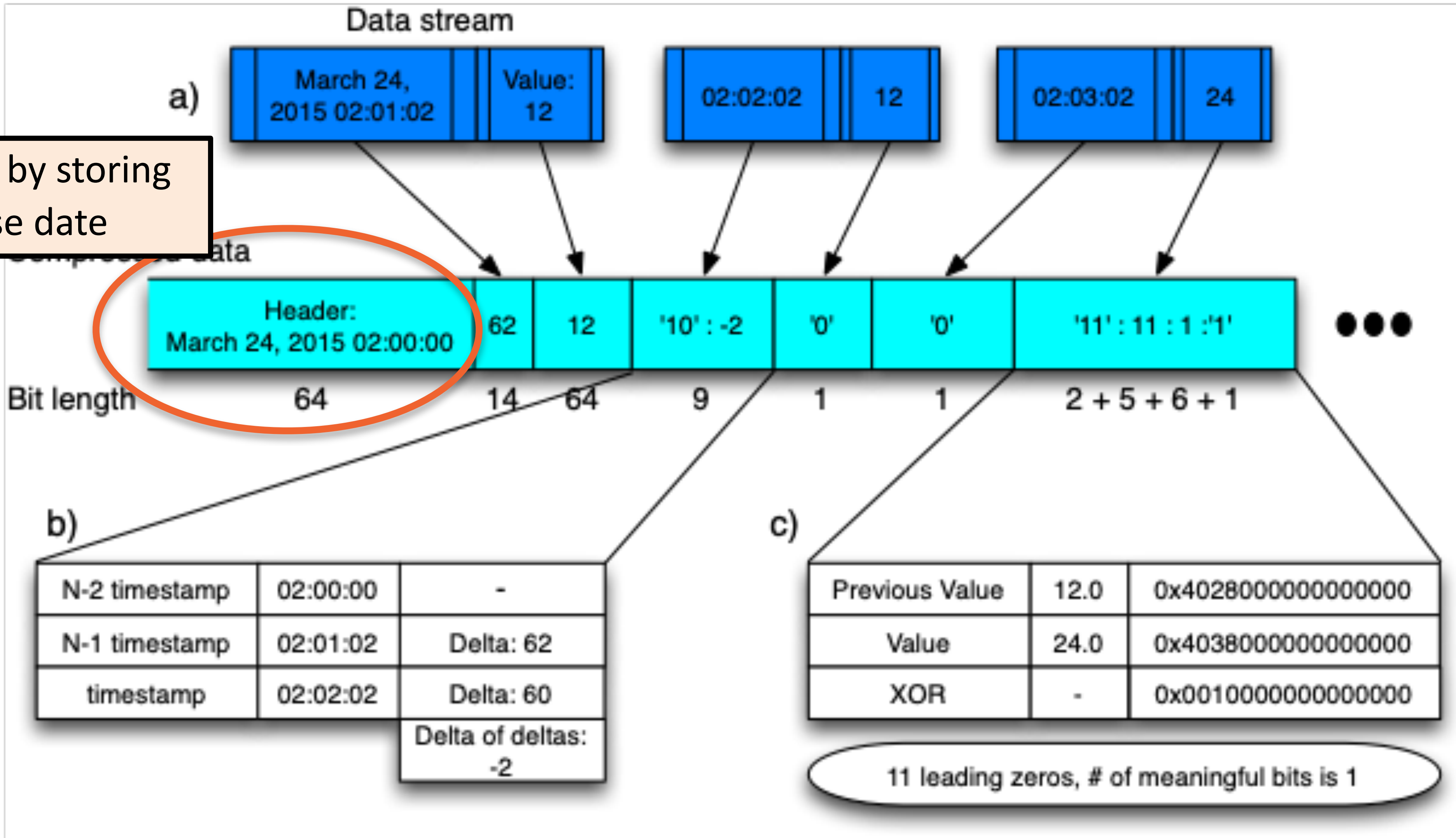
- Compresses data points within a time series.
- e.g.: Facebook Gorilla
  - Each data point is a pair of 64 bit values representing the time stamp and value at that time.
  - Timestamps and values are compressed separately using information about previous values – storing deltas is cheaper.

Time series compression

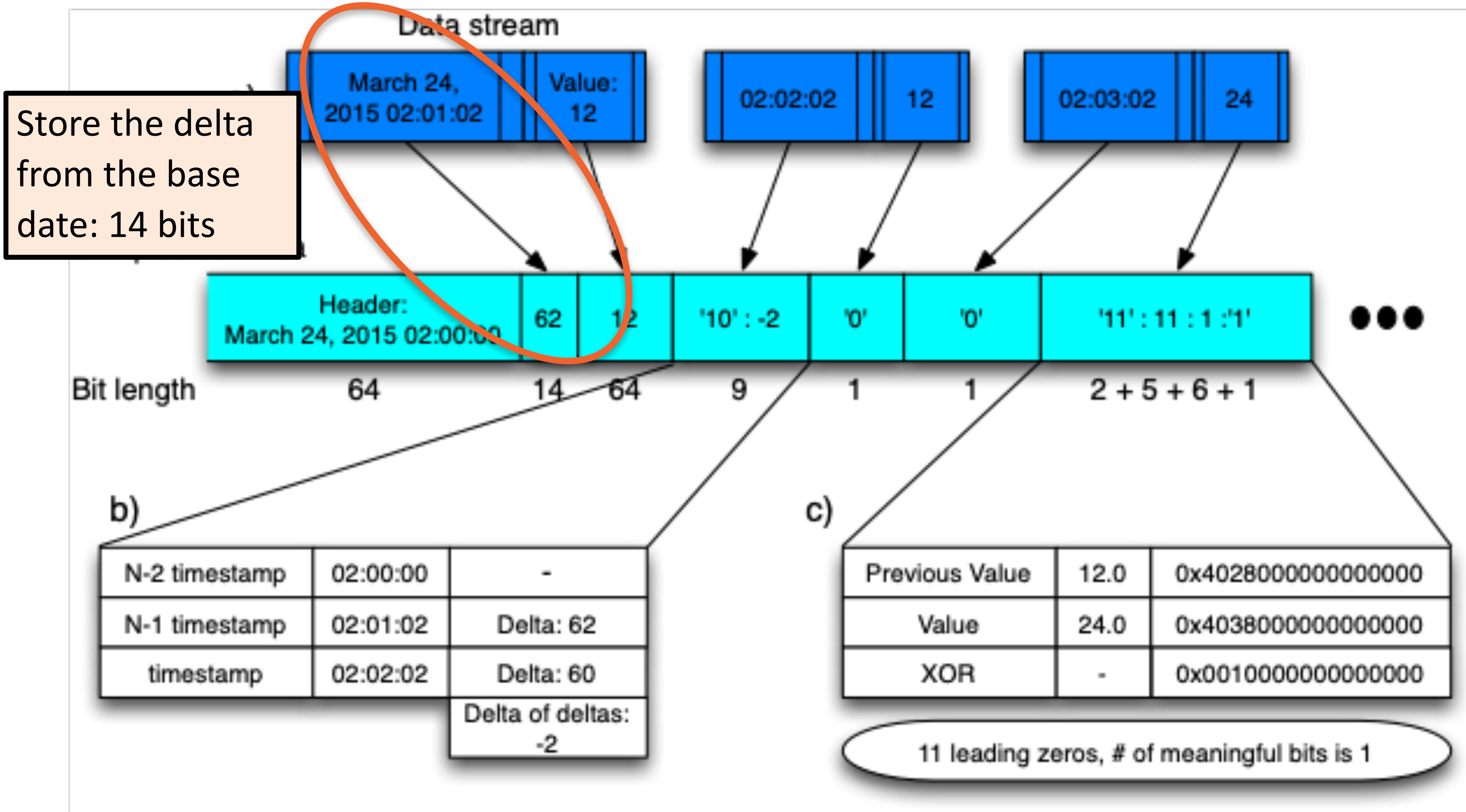


Time series compression

Start by storing a base date

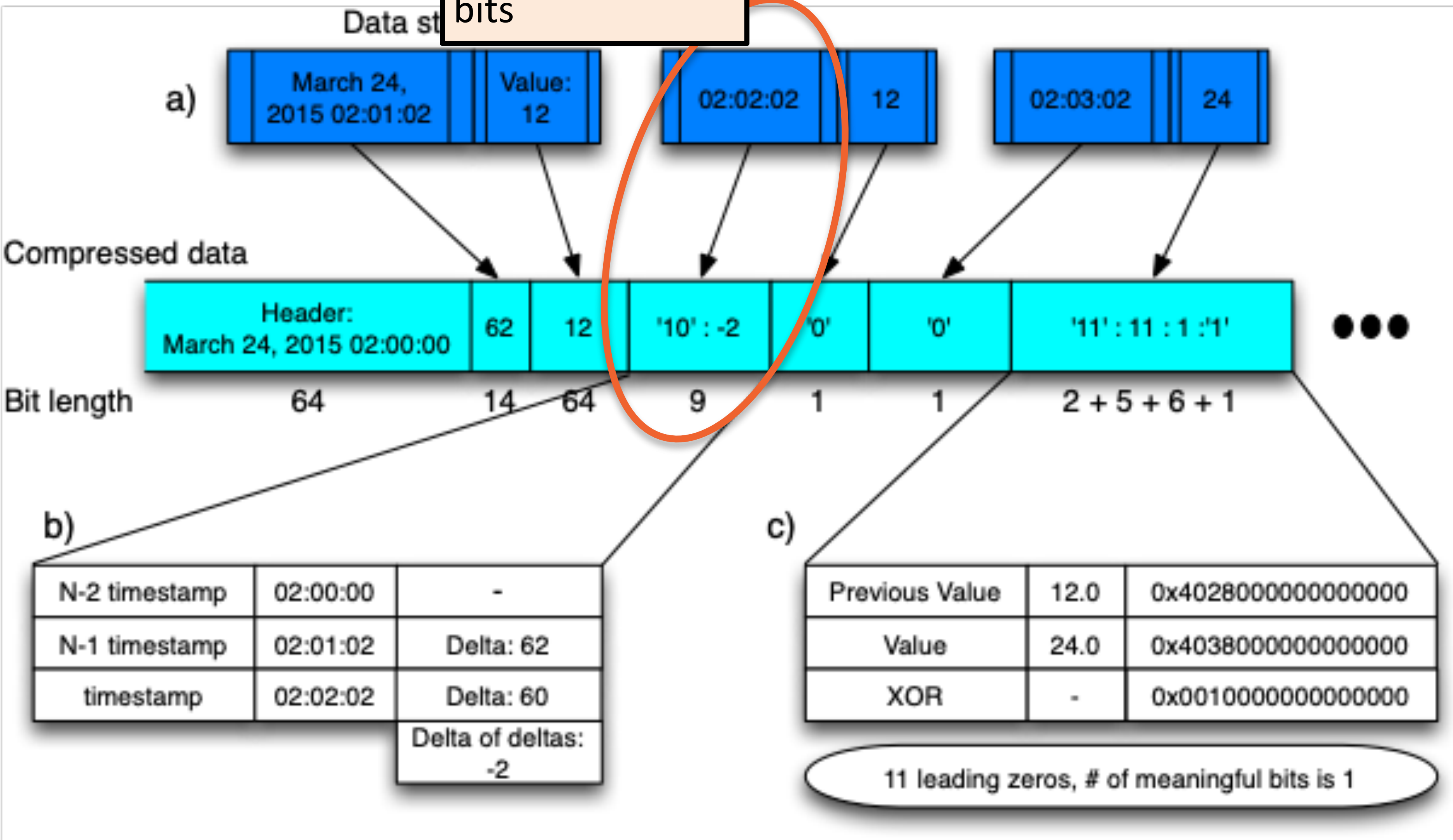


Time series compression



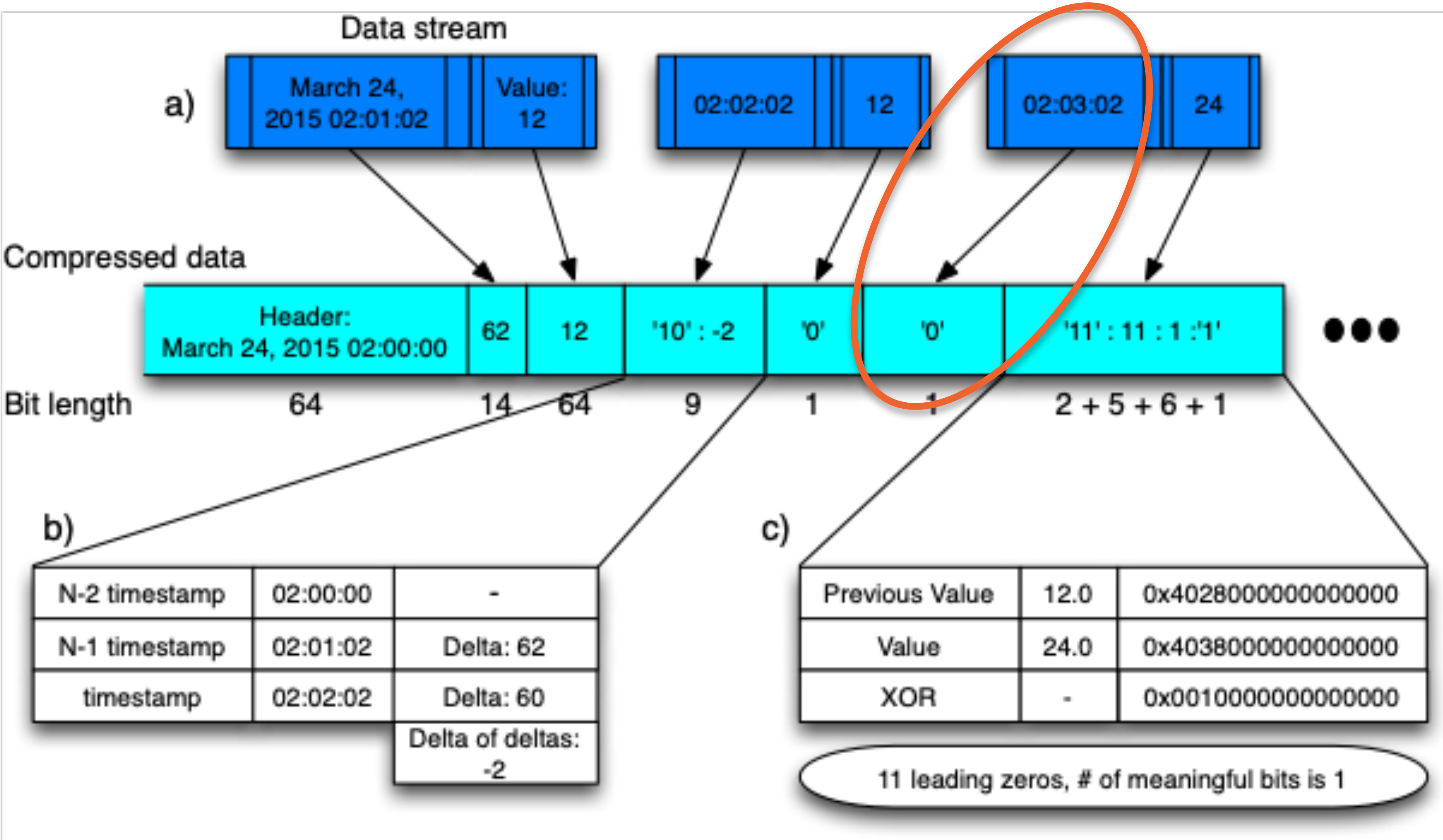
Time series compression

Store the delta from the delta: 9 bits

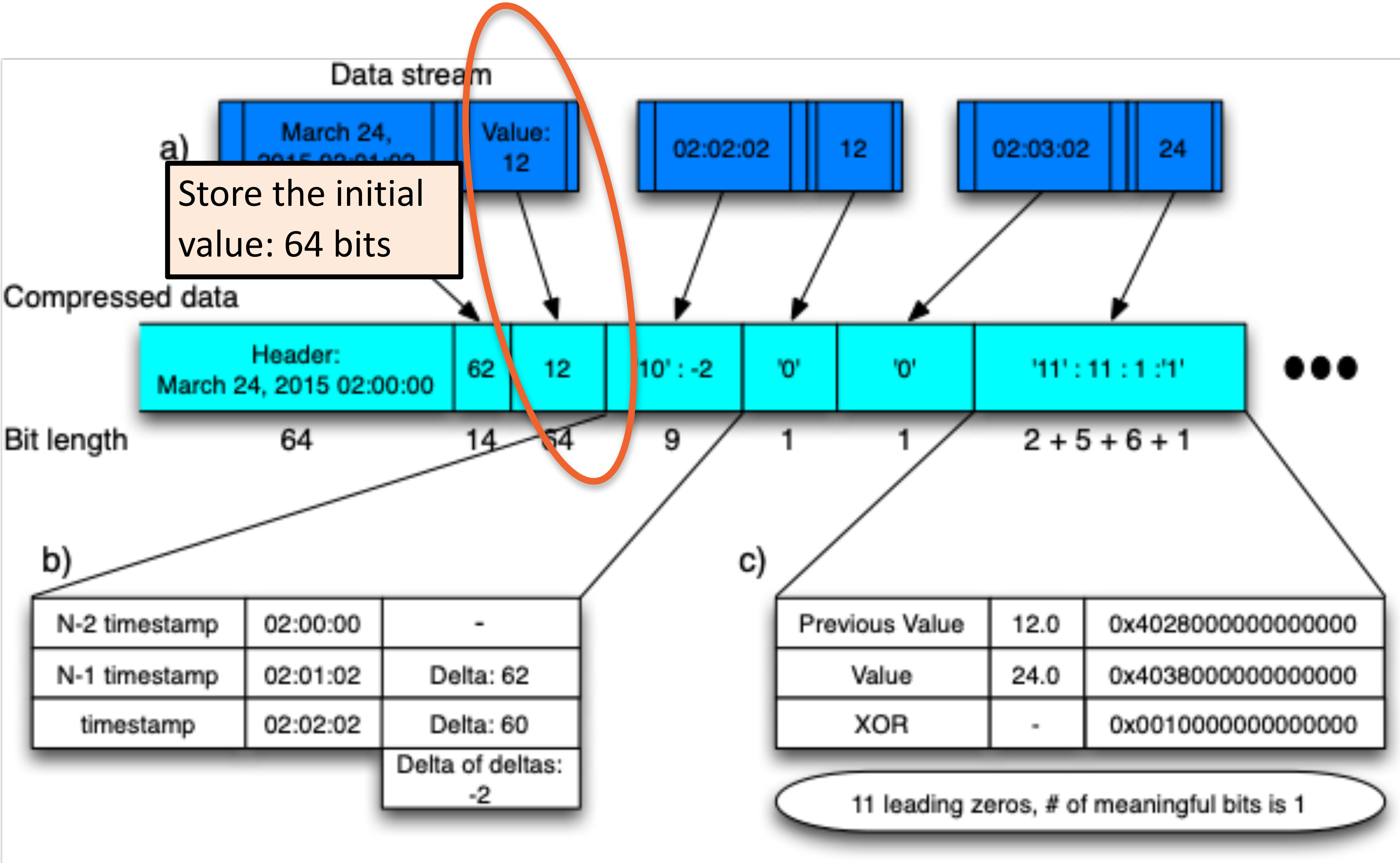


Time series compression

Same delta: 1 bit

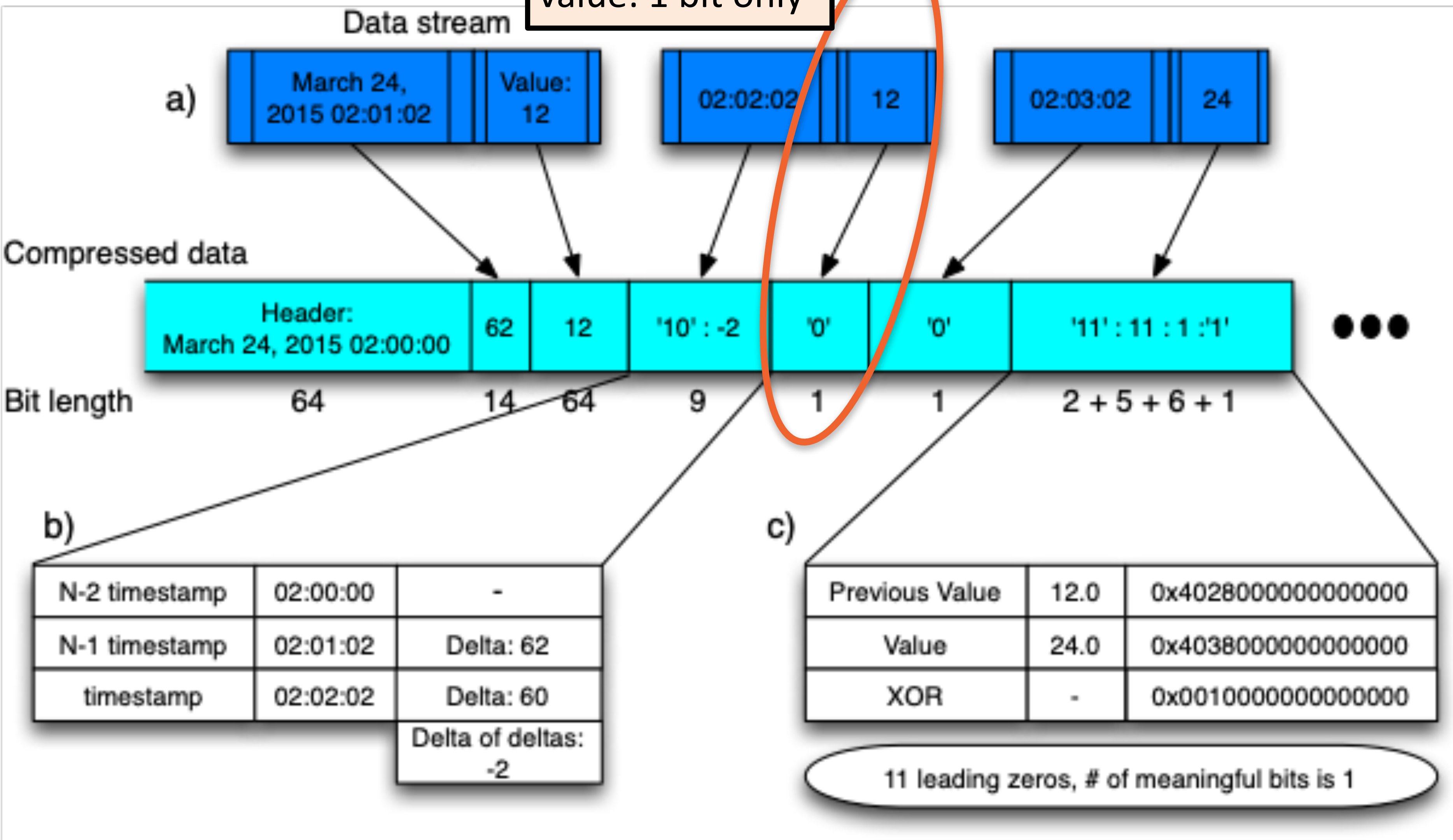


Time series compression

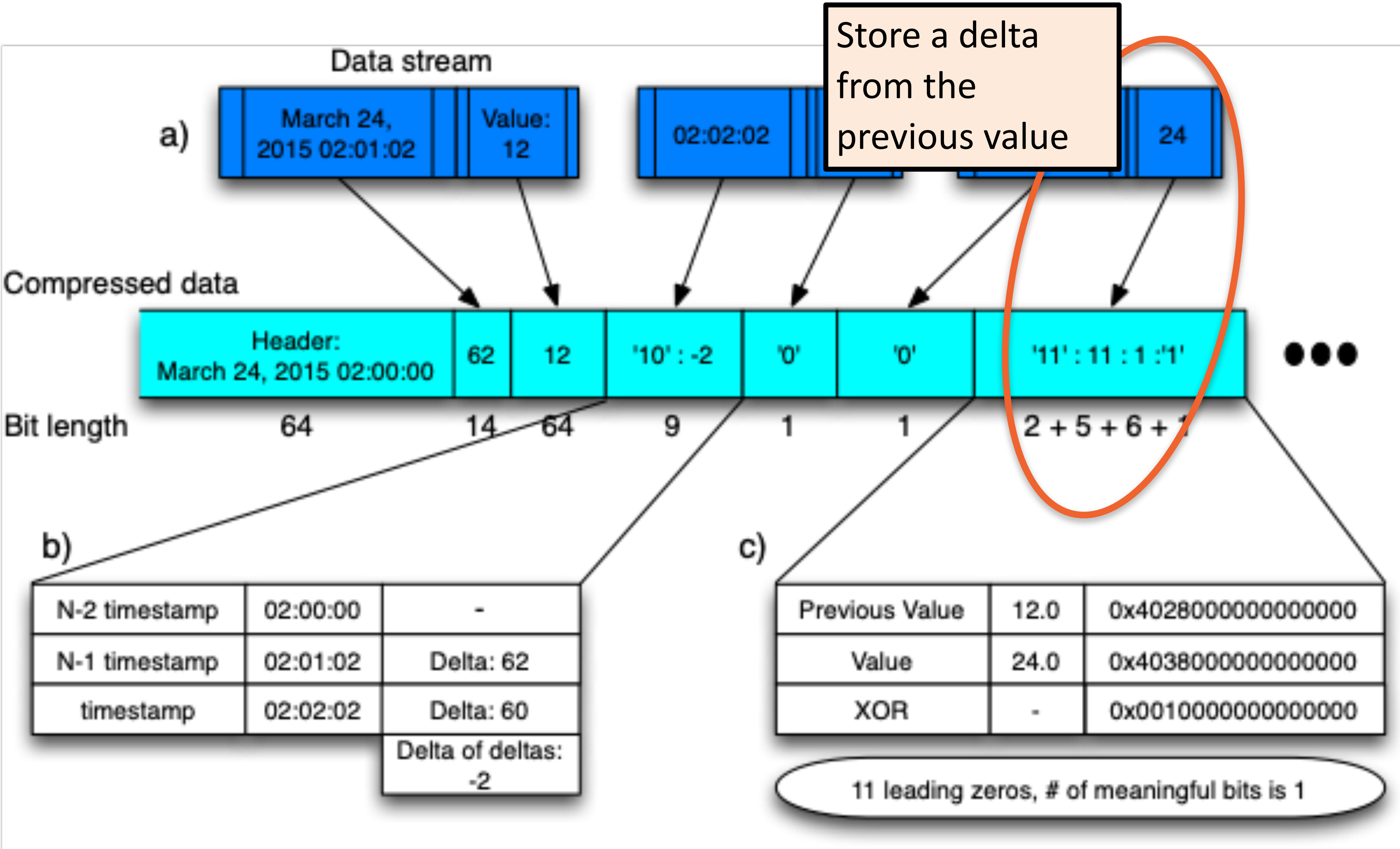


Time series compression

Store the same value: 1 bit only



Time series compression



Design of a TSDB

- Problem: need to write fast, read fast
- Solution: new storage designs, keep indices in memory

Indexing time series

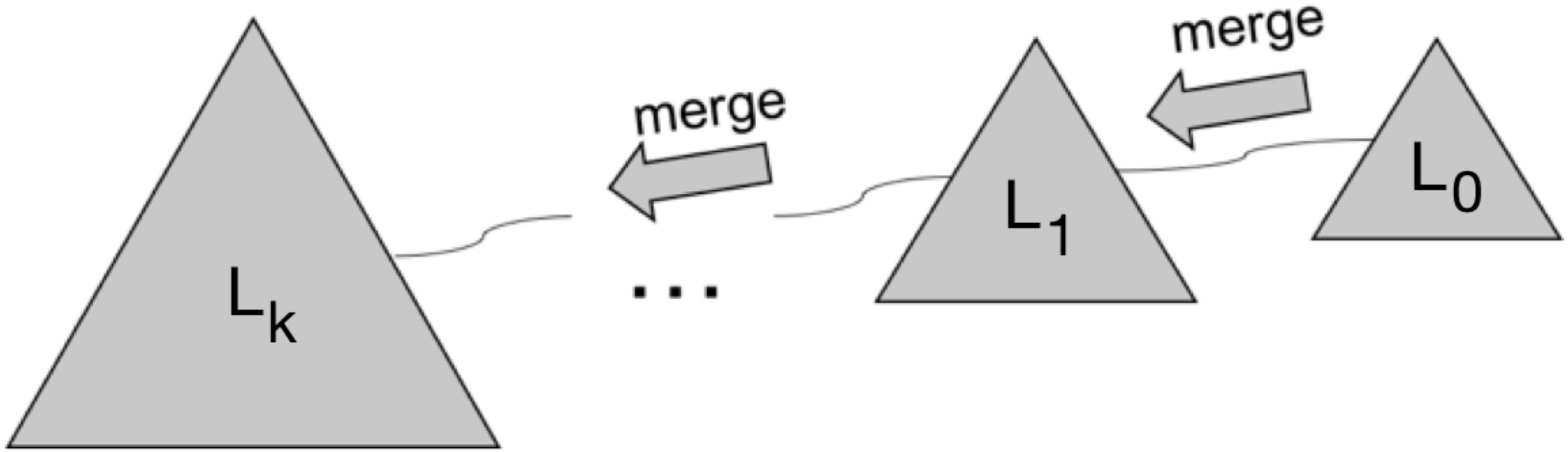
- Need to support fast writes…
- … and fast reads

Indexing time series (cont.)

- Database indexes (B-trees) are not appropriate for time series databases
- Time series databases indexes usually based on LSM trees

Log-structured merge tree (LSM-tree)

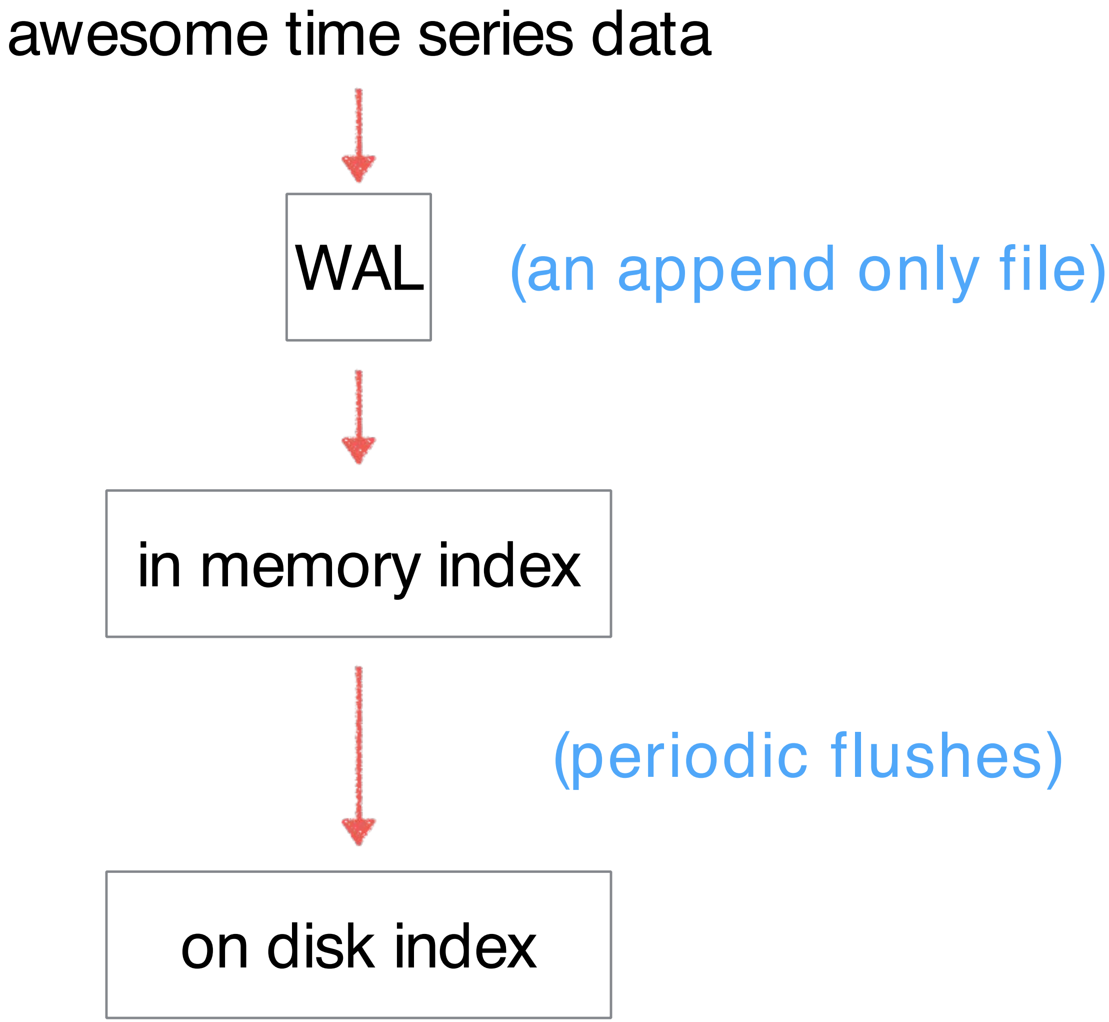
- An LSM-tree consists of a hierarchy of storage levels that increase in size.
- The first level, L0, is stored in memory – used to buffer updates. When this level gets full, it is merged with the other levels.
- The other levels are stored on disk.



LSM-tree: operations (cont.)

- A simple lookup consists in:
  - Searching the value in L0
  - If not found, continue searching in the following levels
    - For efficiency, each level records a summary of the elements present, as a Bloom filter
- Range lookups consist in:
  - Executing a range search in every level
  - Slow, but…
    - If searching for recent values, they will be in L0 (if large enough)
    - The way merging works makes values added at similar times to be in close levels

LSM-based storage in a TSDB (e.g. Influx DB)



How to explore this for indexing in TSDB – e.g. Influx DB

temperature,device=dev1,building=b1 internal=80,external=18 1443782126

Measurement

Tags
(tagset all together)

Fields

Timestamp

Data divided in a sequence of time series.

Each field has its unique identifier.

Key for a value includes the identifier of the field and the timestamp.

temperature,device=dev1,building=b1#internal → 1

1 → (1443782126,80)

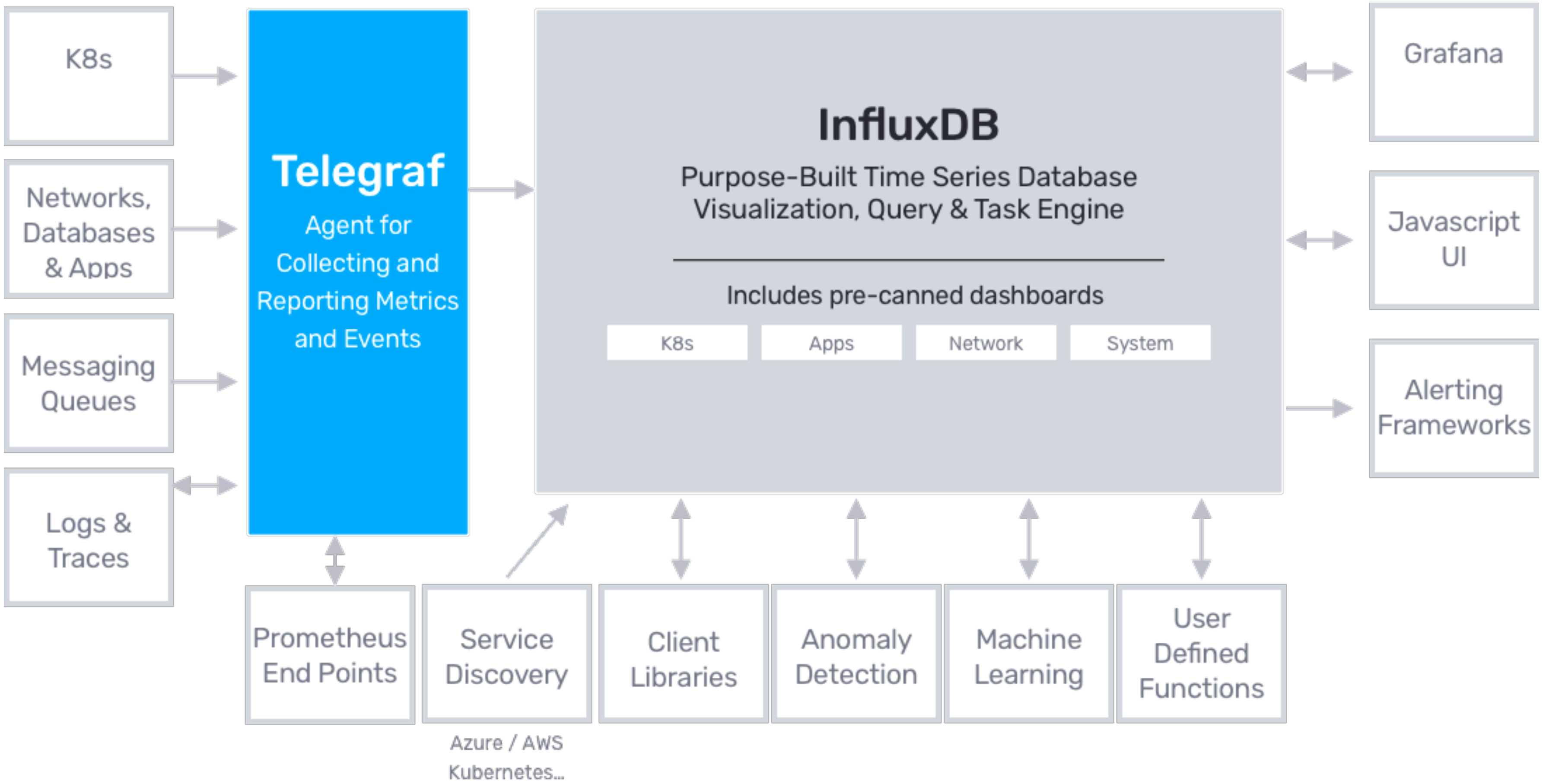
temperature,device=dev1,building=b1#external → 2

2 → (1443782126,18)

key space is ordered

Key	Value
1,1443782126	80
1,1443782127	81
2,1443782126	18

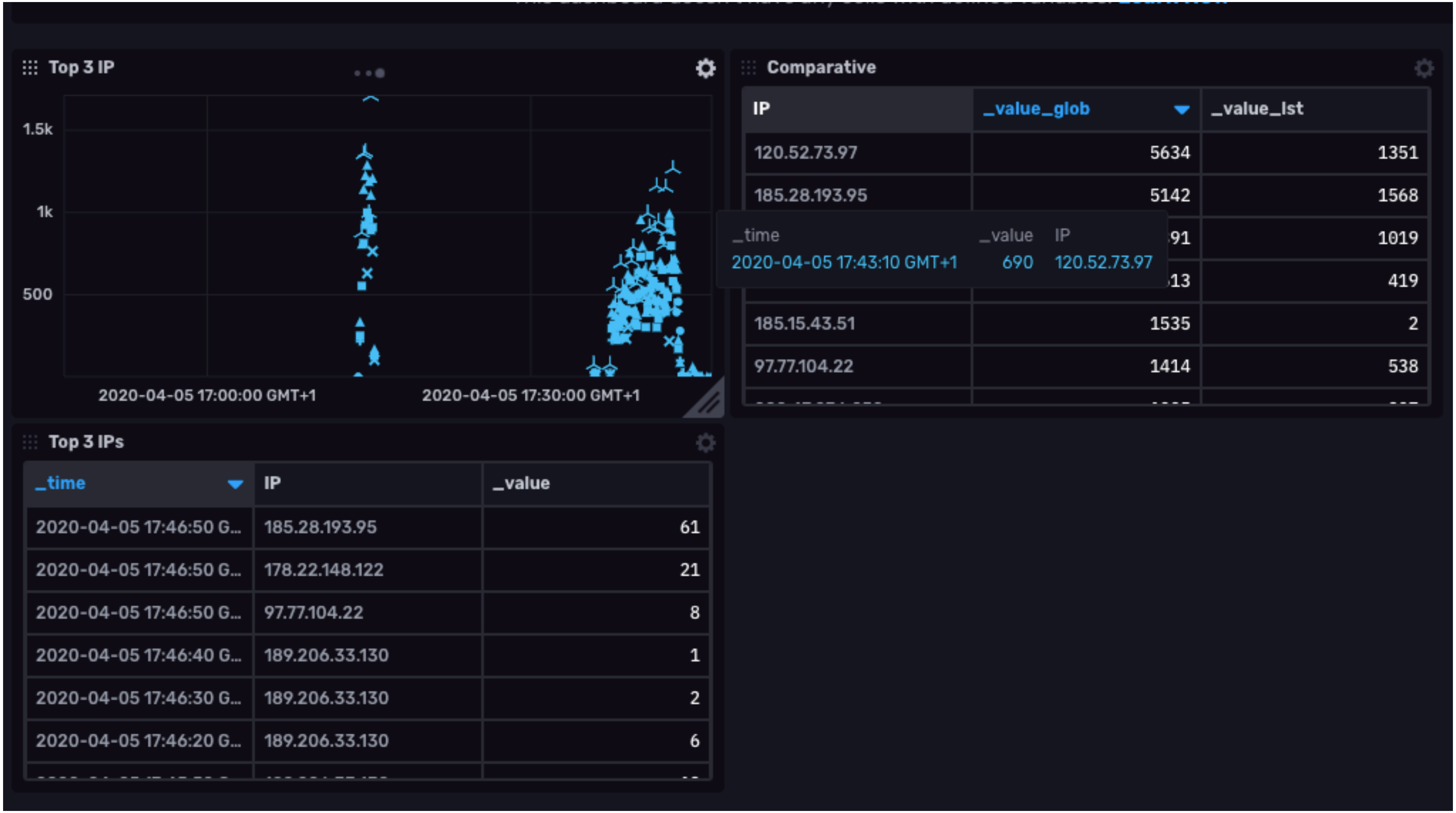
Influx DB ecosystem



InfluxDB 2.0

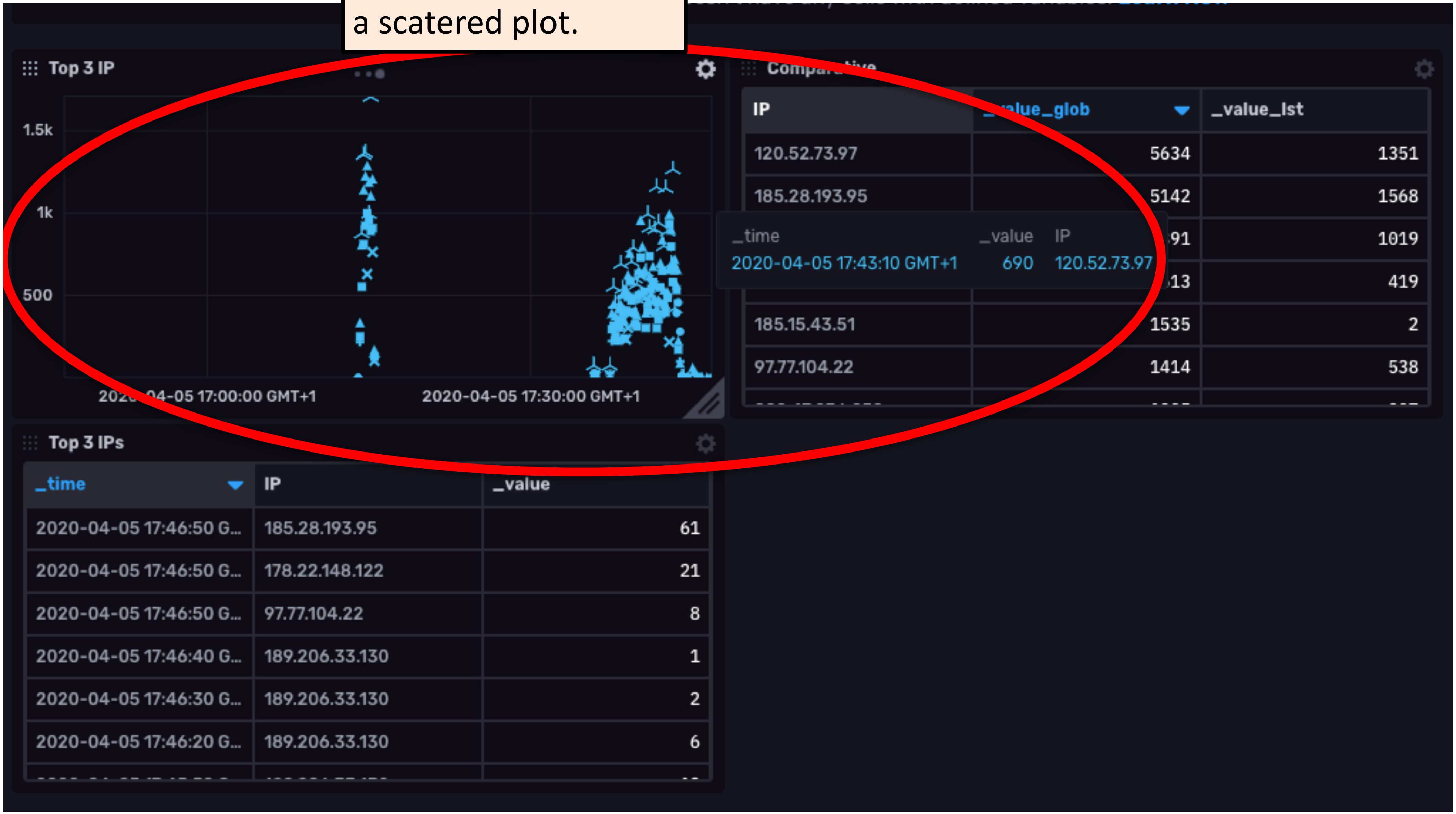
- InfluxDB 2.0 has an interface with integrated querying and displaying of information
  - Also allows to export data for being displayed by other systems – e.g., Grafana (Dashboards).

Weblogs example: kafka + telegraf + influxdb

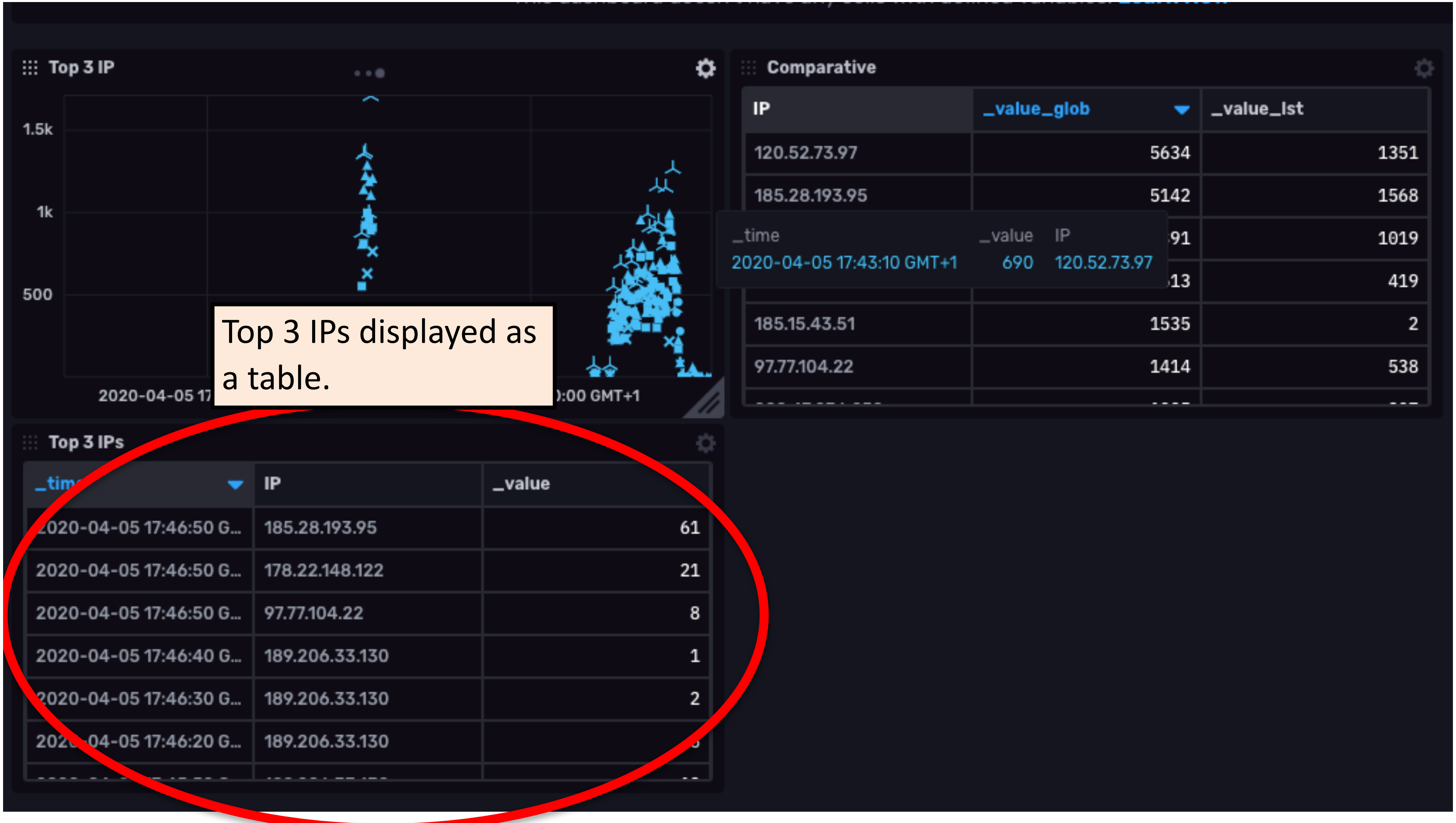


Weblogs example: kafka + telegraf + influxdb

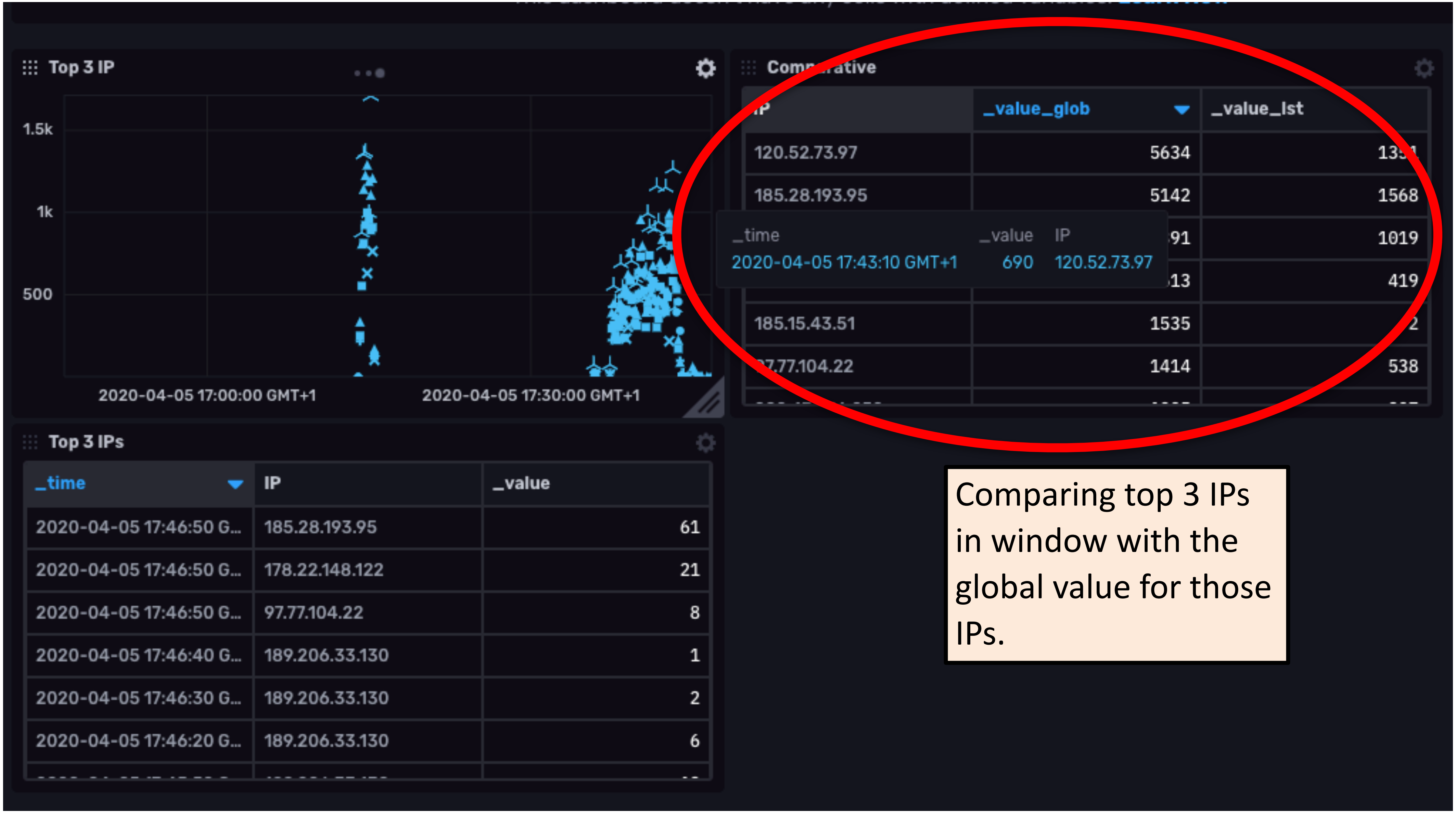
Top 3 IPs displayed as a scatered plot.



Weblogs example: kafka + telegraf + influxdb



Weblogs example: kafka + telegraf + influxdb



Comparing top 3 IPs in window with the global value for those IPs.

First example

- List the top-3 IP sources with more accesses in windows of 30 seconds, every 10 seconds, for the last 15 minutes.

Querying data

- **from (bucket: name)**
  - Select the bucket that stores the data. A bucket may have multiple time series.
    - In the weblog, there is one time series per property (IP,dur,etc.)

```
from(bucket: "weblog")
```

Querying data (cont.)

- **range(start: time[, end: time])**
  - Select the data to be used. e.g.
    - **(start: -5m) :** the last 5 minute
    - (**start: v.timeRangeStart, stop: v.timeRangeStop**) : selected range

```
from(bucket: "weblog")
  |> range(start: -5m)
```

Querying data (cont.)

- **filter (fn)**
  - Filters the data to be queried.
    - **(fn: (r) => r._field == "IP") :** selects the time series of with the IP addresses

```
from(bucket: "weblog")
  |> range(start: -5m)
  |> filter(fn: (r) => r._field == "IP")
```

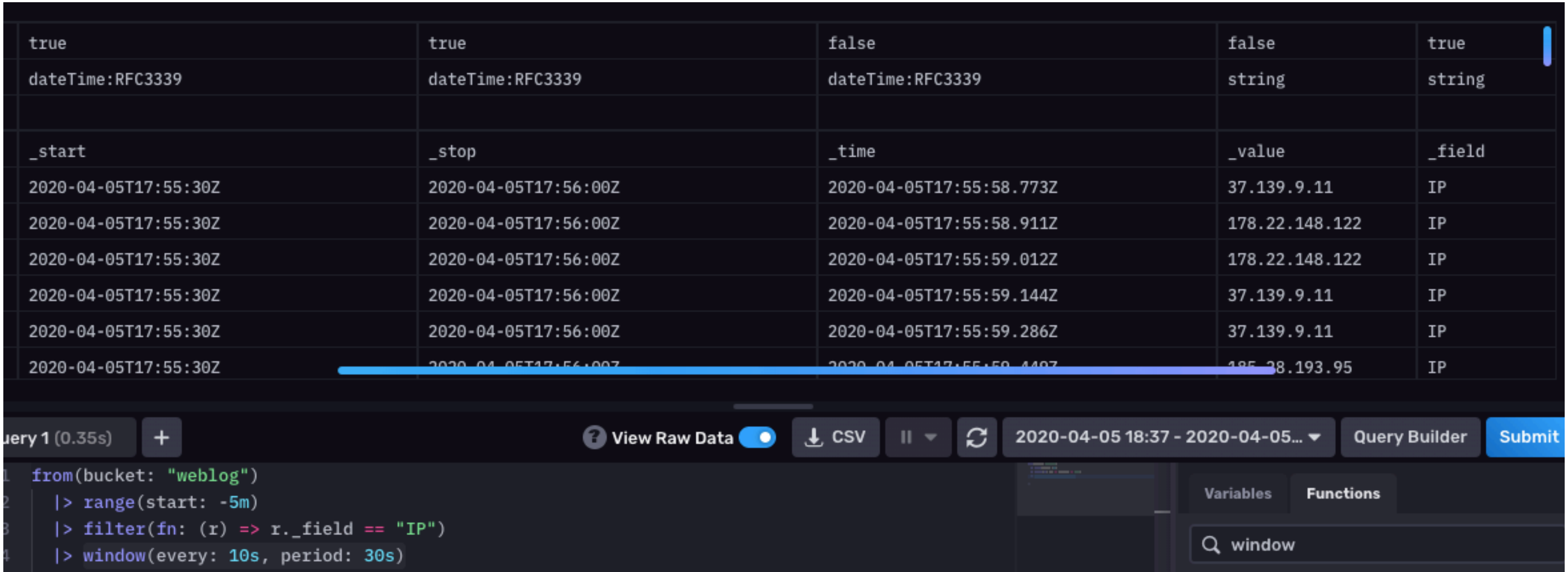
Querying data (cont.)

- **window(every: time, period: time,…)**
  - Group data in windows: **every** specifies the time between windows; **period** specified the window duration, etc.

```
from(bucket: "weblog")
  |> range(start: -5m)
  |> filter(fn: (r) => r._field == "IP")
  |> window(every: 10s, period: 30s)
```

Querying data (cont.)

- **window(every: time, period: time,…)**



Querying data (cont.)

- **group (columns: [...], mode: "by")**
  - Group data by the values of a column for executing an aggregation.
    - Modes: **by** and **except**.

```
from(bucket: "weblog")
  |> range(start: -5m)
  |> filter(fn: (r) => r._field == "IP")
  |> window(every: 10s, period: 30s)
  |> group(columns: ["_start", "_stop", "_value"], mode:"by")
```

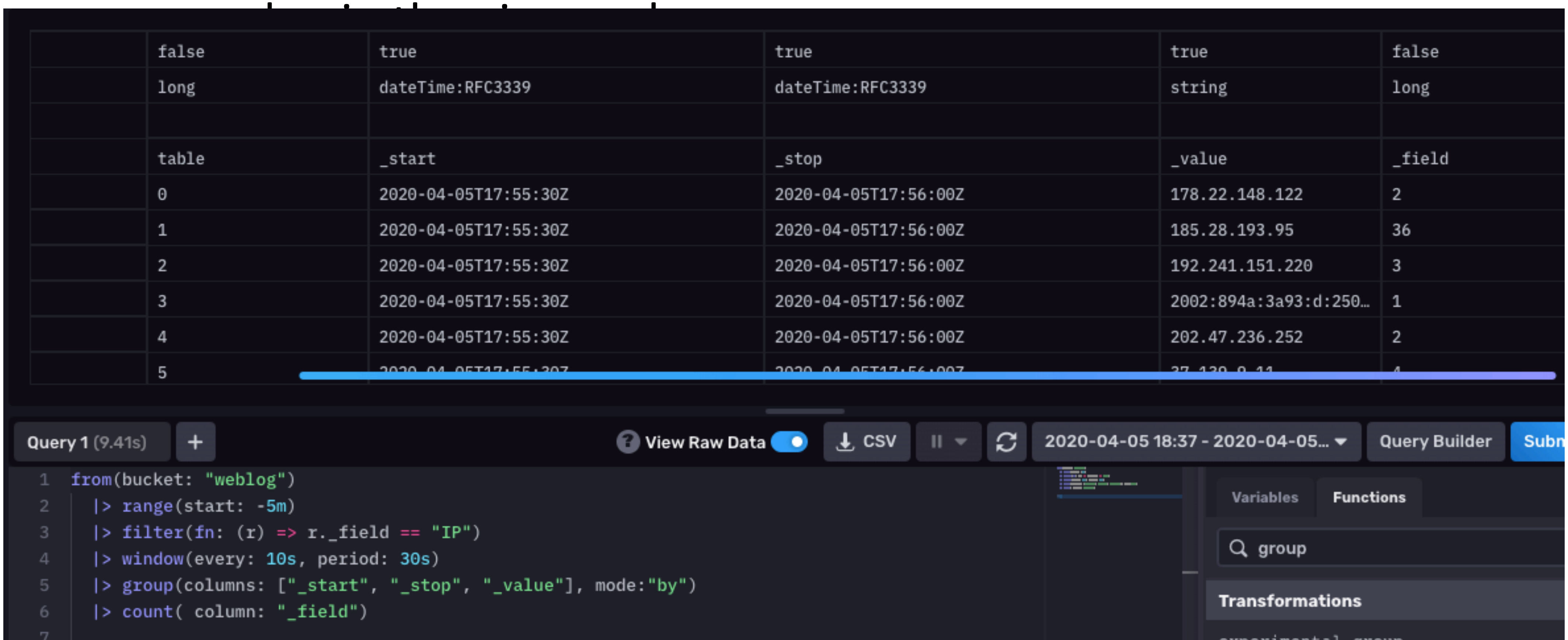
Querying data (cont.)

- **count (column: name)**
  - Counts the number of records in the group and outputs the value in the given column.

```
from(bucket: "weblog")
  |> range(start: -5m)
  |> filter(fn: (r) => r._field == "IP")
  |> window(every: 10s, period: 30s)
  |> group(columns: ["_start", "_stop", "_value"], mode:"by")
  |> count( column: "_field")
```

Querying data (cont.)

- **count (column: name)**
  - Counts the number of records in the group and outputs the

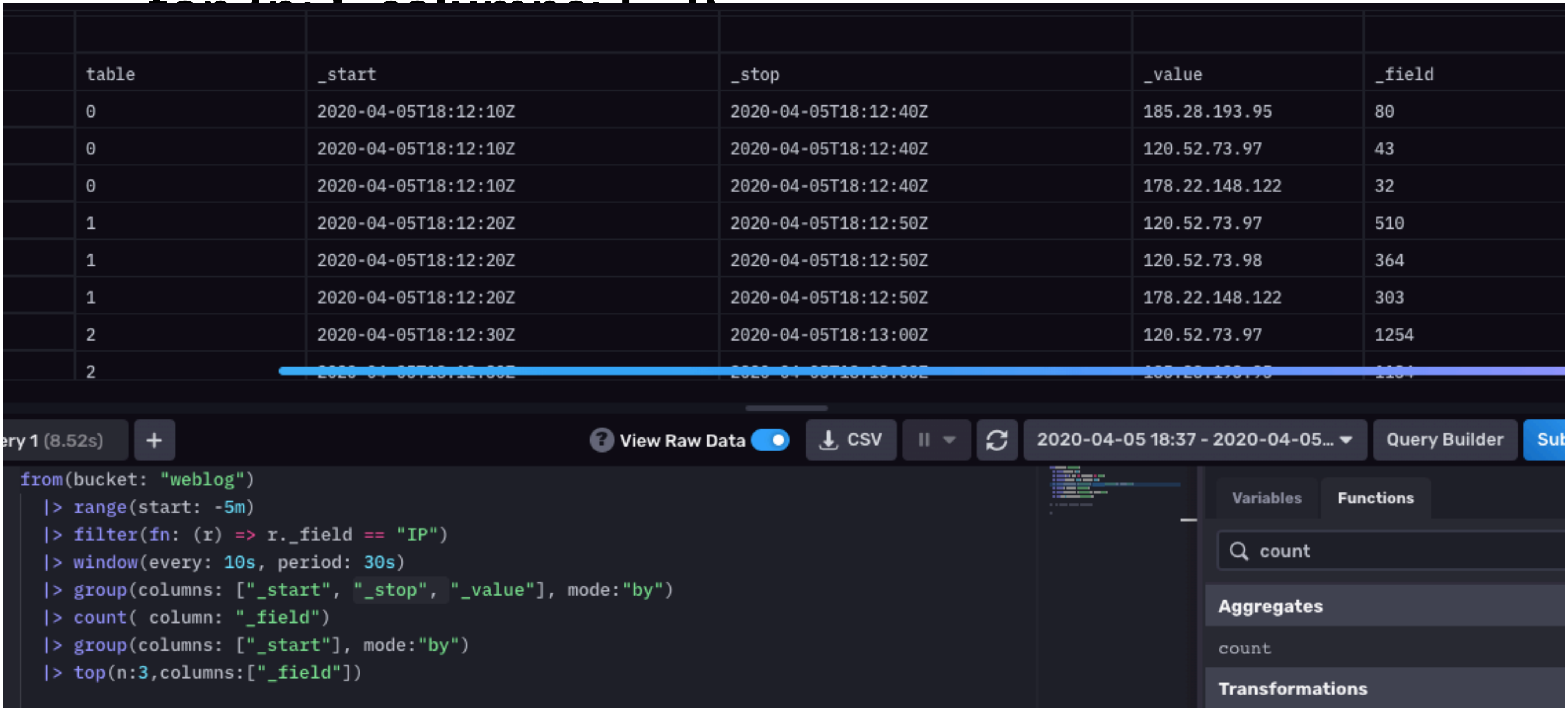


Querying data (cont.)

- **top (n:3, columns: […])**
  - Returns the top **n** element, giving the value of the given columns.
    - E.g. for computing the top element of each window, group by window **_start** first

```
from(bucket: "weblog")
  |> range(start: -5m)
  |> filter(fn: (r) => r._field == "IP")
  |> window(every: 10s, period: 30s)
  |> group(columns: ["_start", "_stop", "_value"], mode:"by")
  |> count( column: "_field")
  |> group(columns: ["_start"], mode:"by")
  |> top(n:3,columns:["_field"])
```

Querying data (cont.)




```
|> top(n:3,columns:["_field"])
```

Querying data (cont.)

- **group ()**
  - Ungroup data.

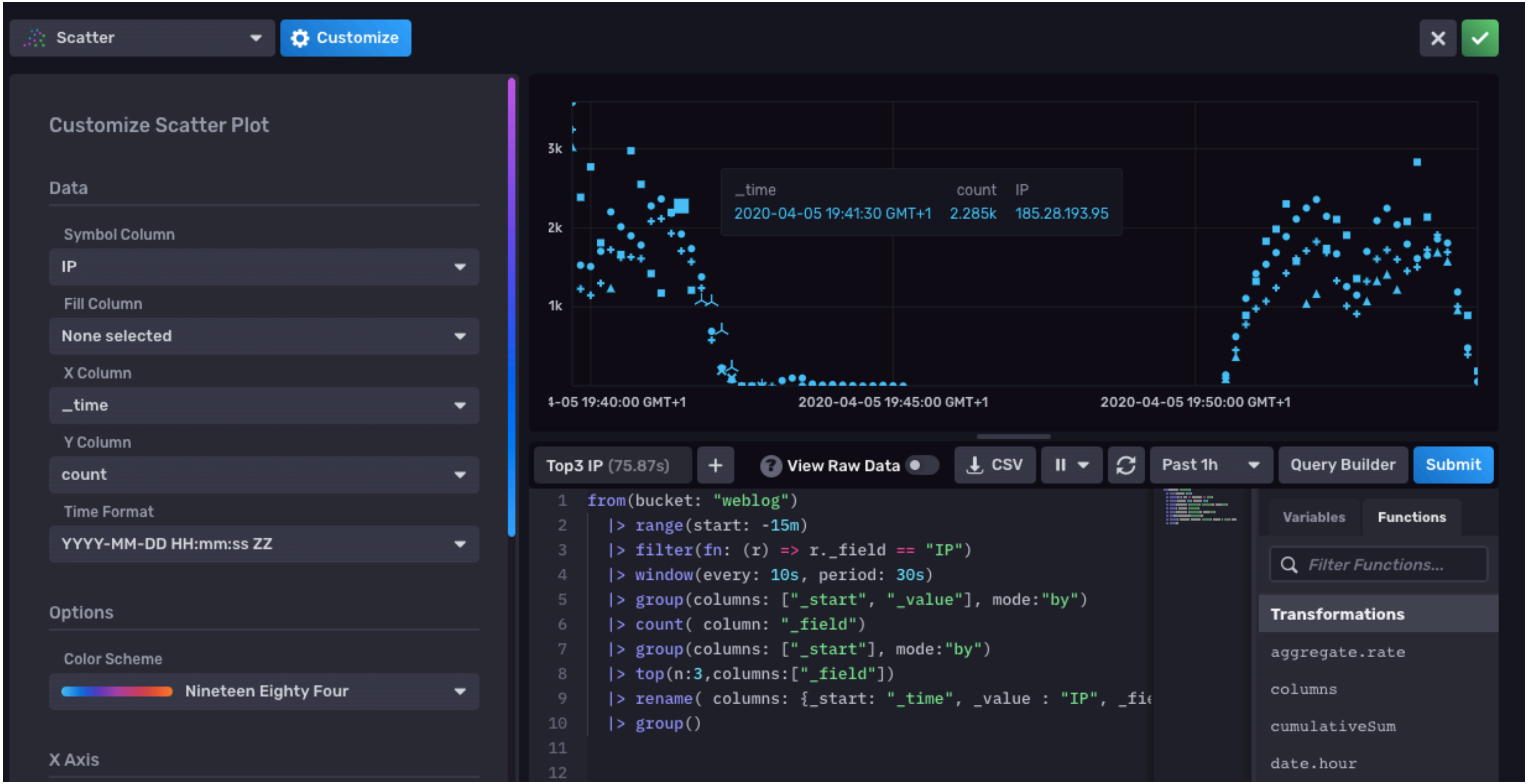
```
from(bucket: "weblog")
  |> range(start: -5m)
  |> filter(fn: (r) => r._field == "IP")
  |> window(every: 10s, period: 30s)
  |> group(columns: ["_start", "_stop", "_value"], mode:"by")
  |> count( column: "_field")
  |> group(columns: ["_start"], mode:"by")
  |> top(n:3,columns:["_field"])
  |> group()
```

Querying data (cont.)

- **rename(columns: …)**
  - Renames column names.

```
from(bucket: "weblog")
  |> range(start: -5m)
  |> filter(fn: (r) => r._field == "IP")
  |> window(every: 10s, period: 30s)
  |> group(columns: ["_start", "_stop", "_value"], mode:"by")
  |> count( column: "_field")
  |> group(columns: ["_start"], mode:"by")
  |> top(n:3,columns:["_field"])
  |> group()
  |> rename( columns: {_value : "IP", _field : "count"})
```

Build a scatter plot with this data



Tasks

- Storing all data forever is not an option, but still interesting to store aggregate information.
- Possible to define a task that periodically downsamples data, by computing aggregate values from data in one bucket and stores them in other bucket.

Table of Contents

- Storage for Big Data
  - File systems
    - HDFS
  - Databases
    - Key-Value stores
    - Time-series databases
- **IoT**

Internet of things (definition)

"***Things* are active participants in business**, information and social processes where they are **enabled to interact and communicate among themselves** and with the environment by exchanging data and information sensed about the environment, while reacting autonomously to the real/physical world events and influencing it by running processes that trigger actions and create services **with or without direct human intervention**."

H. Sundmaeker, P. Guillemin, P. Friess, S. Woelfflé, Vision and challenges for realising the Internet of Things, Cluster of European Research Projects on the Internet of Things—CERP IoT, 2010.

IoT challenges

- IoT creates of an unprecedented amount of data.
- Applications act based on input data.
- Challenges
  - How to manage data
    - Store all data, aggregations, expiration, etc.
  - How to process data
    - Centralized, distributed?

IoT platforms

- IoT is emerging as a key infrastructure in many domains
- Architecture requirements
  - Interconnect many heterogeneous devices
  - Collect data from multiple sources
  - Connect several services

IoT approaches: cloud centric

- Connect devices directly to the cloud
  - Every data is sent to the cloud
  - All computing is performed in the cloud

- Use “standard” stream processing systems/ analytics to process incoming data
- Use “time series” databases to manage sensor data

IoT approaches: edge / fog computing

- Execute computations closer to the devices
  - "Powerful" edge devices process data and execute actions
  - Only part of the data is propagated to the cloud
- Analytics / ML at the edge?
  - Models built on the cloud
  - Models used at the edge to classify / execute actions

Some research questions

- How to distribute computations across multiple devices?
- How to minimize resource consumption – network, storage?
- How to build / evolve models without propagating all data to the cloud?
- How to execute computation while keeping some degree of privacy?

Bibliography

- HDFS Architecture. Dhruba Borthakur.
  - http://svn.apache.org/repos/asf/hadoop/common/tags/release-0.19.2/docs/hdfs_design.pdf
- Gorilla: A Fast, Scalable, In-Memory Time Series
  - https://www.vldb.org/pvldb/vol8/p1816-teller.pdf
- Too detailed references:
  - Log-structured merge trees
    - https://www.cs.umb.edu/~poneil/lsmtree.pdf
  - https://docs.influxdata.com/influxdb/v1.7/concepts/storage_engine/

Acknowledgments

- Some images from:
  - Inside the InfluxDB Storage Engine. Gianluca Arbezzano
  - Tuomas Pelkonen, et. al. Gorilla: A Fast, Scalable, In-Memory Time Series Database. VLDB’15.