KubeCon

CloudNativeCon

THE LINUX FOUNDATION

OPEN SOURCE SUMMIT

AI_dev

Open Source GenAI & ML Summit

China 2024

KubeCon | CloudNativeCon

THE LINUX FOUNDATION OPEN SOURCE SUMMIT

AI_dev Open Source GenAI & ML Summit

China 2024

Building a High-Performance Time Series Database from Scratch

- Aliaksandr Valialkin & Hui Wang, VictoriaMetrics

About me

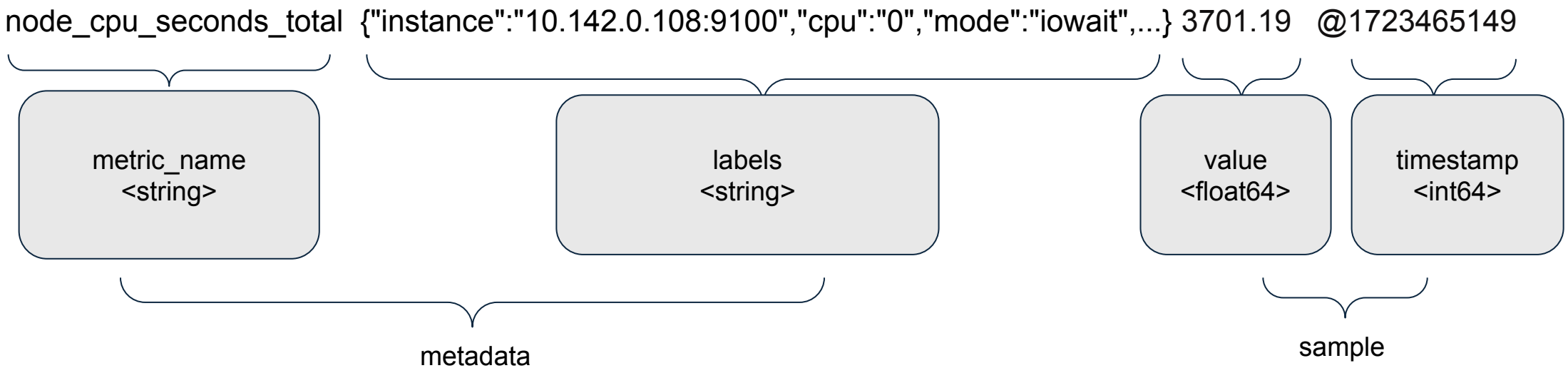
Hui Wang
https://github.com/Haleygo

Working at VictoriaMetrics
Working on monitoring and Kubernetes

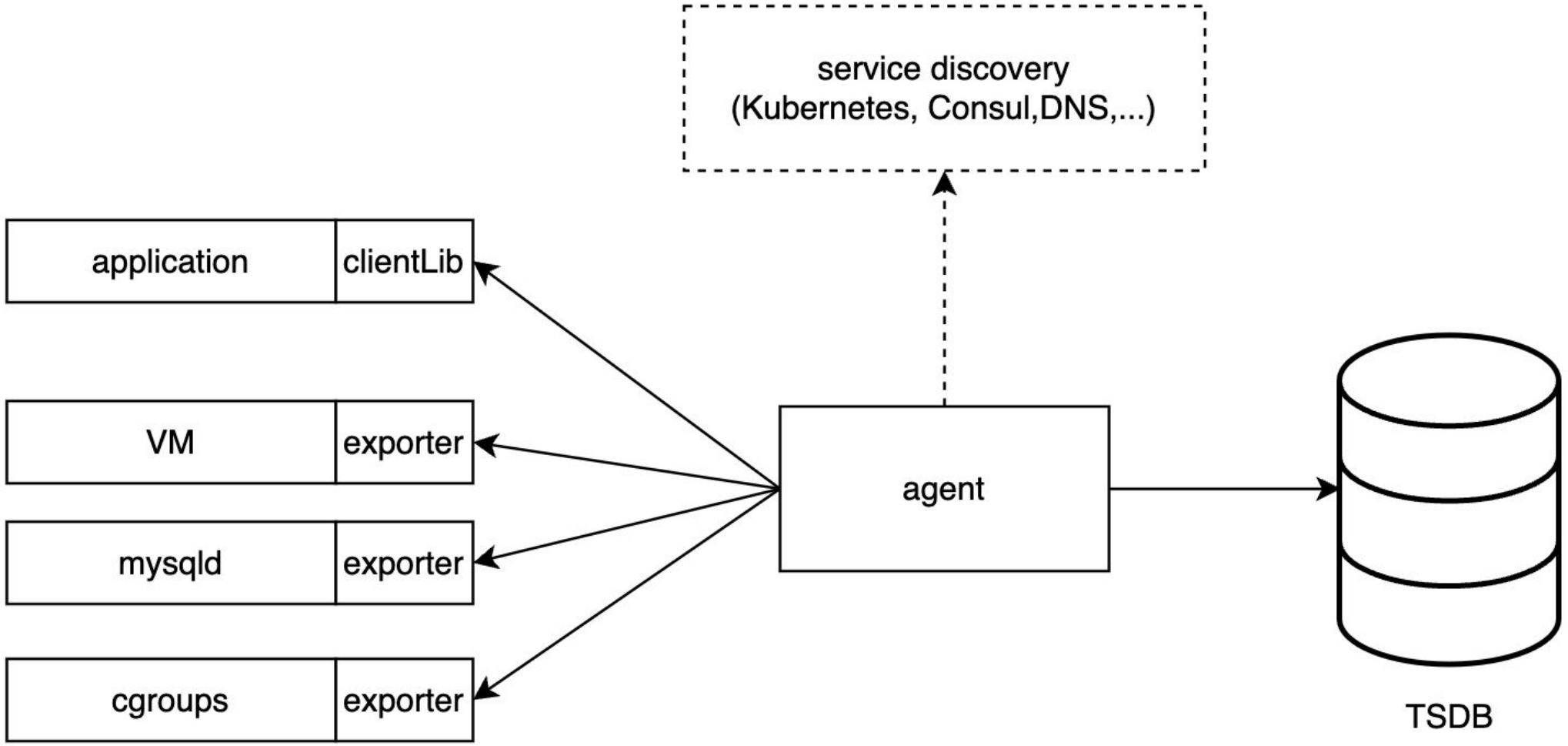
Agenda

1. What’s a time series
2. How to build a TSDB
3. The journey of a time series
4. Ways to improve your TSDB experiences

What’s a time series



How to collect metrics

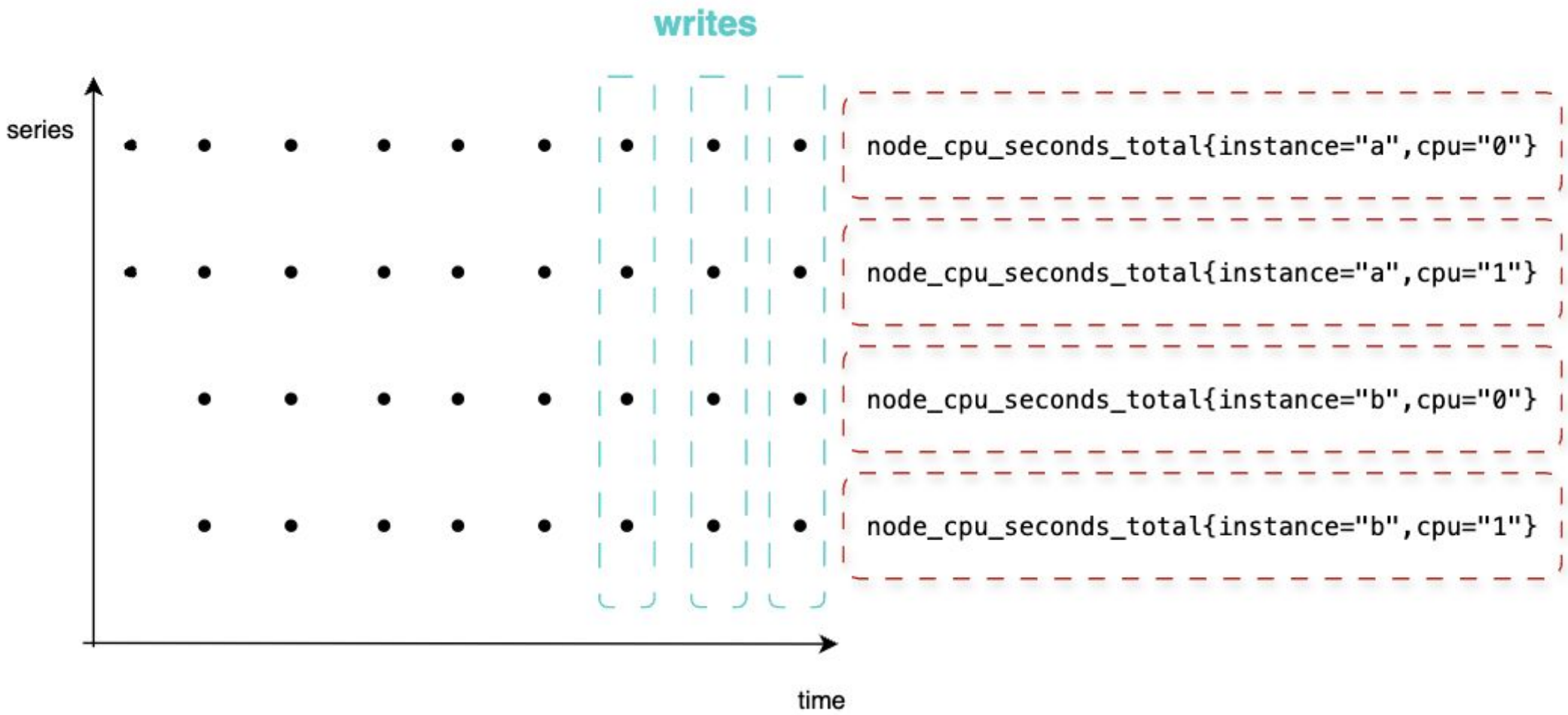


How to collect metrics

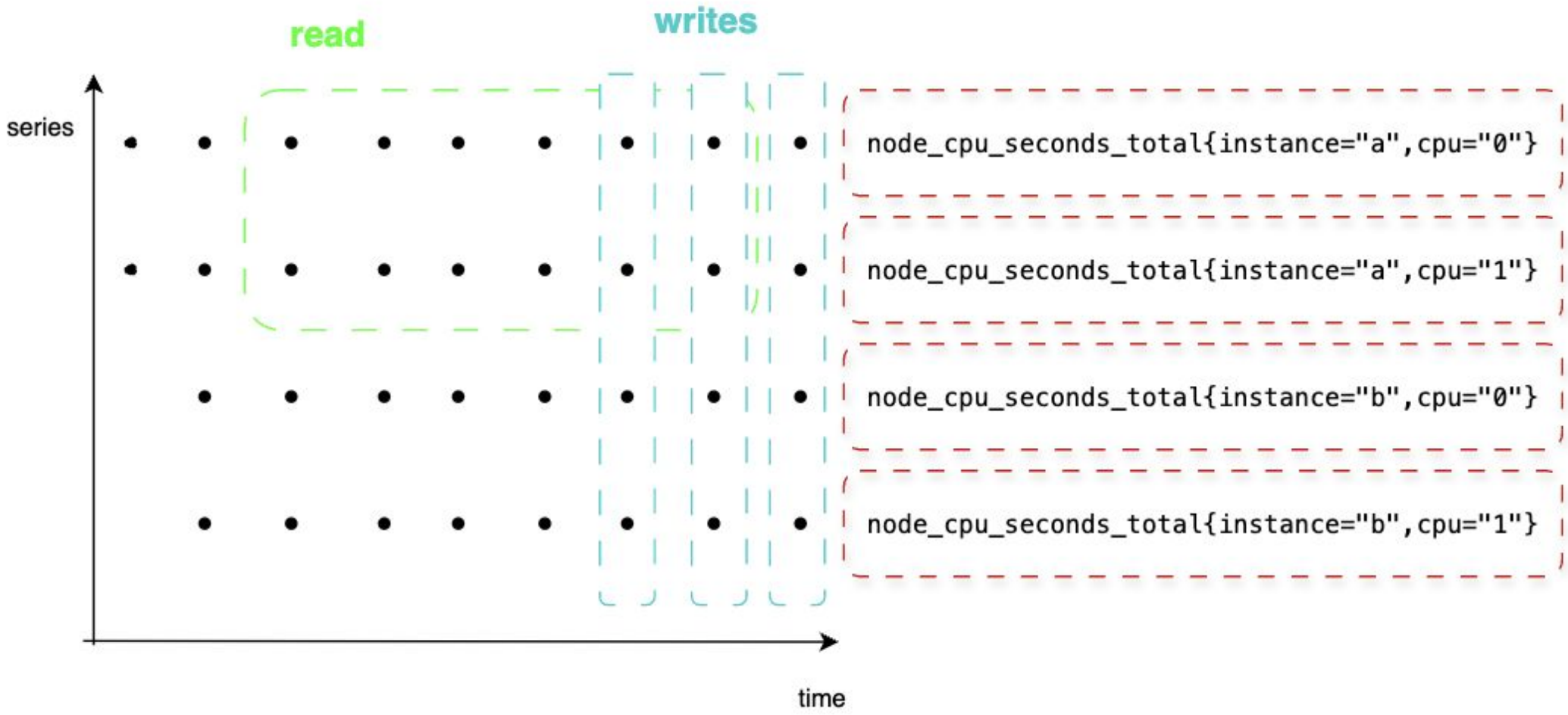
> curl http://{service-address}/metrics

```
# HELP node_cpu_seconds_total Seconds the CPUs spent in each mode.
# TYPE node_cpu_seconds_total counter
node_cpu_seconds_total{cpu="0",mode="idle"} 651947.04
node_cpu_seconds_total{cpu="0",mode="iowait"} 4798.62
node_cpu_seconds_total{cpu="0",mode="irq"} 0
node_cpu_seconds_total{cpu="0",mode="nice"} 0
node_cpu_seconds_total{cpu="0",mode="softirq"} 77037.95
node_cpu_seconds_total{cpu="0",mode="steal"} 3525.07
node_cpu_seconds_total{cpu="0",mode="system"} 44646.12
node_cpu_seconds_total{cpu="0",mode="user"} 225132.33
node_cpu_seconds_total{cpu="1",mode="idle"} 655196.6
node_cpu_seconds_total{cpu="1",mode="iowait"} 4610.05
node_cpu_seconds_total{cpu="1",mode="irq"} 0
node_cpu_seconds_total{cpu="1",mode="nice"} 0
node_cpu_seconds_total{cpu="1",mode="softirq"} 47408.89
node_cpu_seconds_total{cpu="1",mode="steal"} 3417.36
node_cpu_seconds_total{cpu="1",mode="system"} 45130.2
node_cpu_seconds_total{cpu="1",mode="user"} 225970.6
```

Workload of TSDB



Workload of TSDB



Workload of TSDB

1. Write requests are extremely heavy
2. Data is append-only
3. Read load is much lower than the write load
4. Read requests are unpredictable

How to build a TSDB


```
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}   3701.19 @1723465149
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}   3701.32 @1723465179
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}   3701.44 @1723465209
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}   3701.57 @1723465239
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}   3701.68 @1723465269
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}   3701.77 @1723465299
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}   3701.89 @1723465329
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}   3701.98 @1723465359
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}   3702.11 @1723465389
```

metricName labels value timestamp

How to build a TSDB


```
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}   3701.19 @1723465149
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}   3701.32 @1723465179
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}   3701.44 @1723465209
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}   3701.57 @1723465239
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}   3701.68 @1723465269
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}   3701.77 @1723465299
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}   3701.89 @1723465329
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}   3701.98 @1723465359
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}   3702.11 @1723465389
```

TSID

metricName

labels

value

timestamp

index

data

Data compression

1. Timestamp – Delta-of-delta encoding

@1723810130

@1723810145

@1723810160

@1723810174

@1723810190

@1723810205

@1723810220

@1723810235

Data compression

1. Timestamp – Delta-of-delta encoding

@1723810130		@1723810130
@1723810145		+15
@1723810160		+15
@1723810174	$\xrightarrow{\Delta}$	+14
@1723810190		+16
@1723810205		+15
@1723810220		+15
@1723810235		+15

Data compression

1. Timestamp – Delta-of-delta encoding

@1723810130		@1723810130		@1723810130
@1723810145		+15		+15
@1723810160		+15		+0
@1723810174	$\xrightarrow{\Delta}$	+14	$\xrightarrow{\Delta\Delta}$	-1
@1723810190		+16		+2
@1723810205		+15		-1
@1723810220		+15		+0
@1723810235		+15		+0

Data compression

2. Value – XOR-based compression

Decimal	Double Representation	XOR with previous
12	0x4028000000000000	
24	0x4038000000000000	0x0010000000000000
15	0x402e000000000000	0x0016000000000000
12	0x4028000000000000	0x0006000000000000
35	0x4041800000000000	0x0069800000000000

Decimal	Double Representation	XOR with previous
15.5	0x402f000000000000	
14.0625	0x402c200000000000	0x0003200000000000
3.25	0x400a000000000000	0x0026200000000000
8.625	0x4021400000000000	0x002b400000000000
13.1	0x402a333333333333	0x000b733333333333

Figure 4: Visualizing how XOR with the previous value often has leading and trailing zeros, and for many series, non-zero elements are clustered.

Source: Gorilla paper

https://www.vldb.org/pvldb/vol8/p1816-teller.pdf
https://valyala.medium.com/victoriametrics-achieving-better-compression-for-time-series-data-than-gorilla-317bc1f95932

Data compression

node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}	3701.19 @1723465149
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}	3701.32 @1723465179
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}	3701.44 @1723465209
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}	3701.57 @1723465239
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}	3701.68 @1723465269
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}	3701.77 @1723465299
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}	3701.89 @1723465329
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}	3701.98 @1723465359
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}	3702.11 @1723465389

TSID ⇔ node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}

TSID ⇨ **[3701.19,d1,d2,d3,d4,d5,...]**

TSID ⇨ **[1723465149,30,0,0,0,0,0,...]**

Data compression

node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"} 3701.19 @1723465149
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.1[illegible]00","cpu":"0"} 3701.32 @1723465179
node_cpu_seconds_total{"job":"node-exp[illegible]","instance[illegible]00"[illegible]:"0"} 3701.44 @1723465209
node_cpu_seconds_total{"job":"node-expo[illegible]pu":"0"} 3701.57 @1723465239
node_cpu_seconds_total{"job":"node-[illegible]} 3701.68 @1723465269
node_cpu_seconds_total{"job":"no[illegible]":"0"} 3701.77 @1723465299
node_cpu_seconds_total{"job":"node-e[illegible]0"} 3701.89 @1723465329
node_cpu_seconds_total{"jo[illegible]u":"0"} 3701.98 @1723465359
node_cpu_seconds_total{"job":"no[illegible]0"} 3702.11 @1723465389

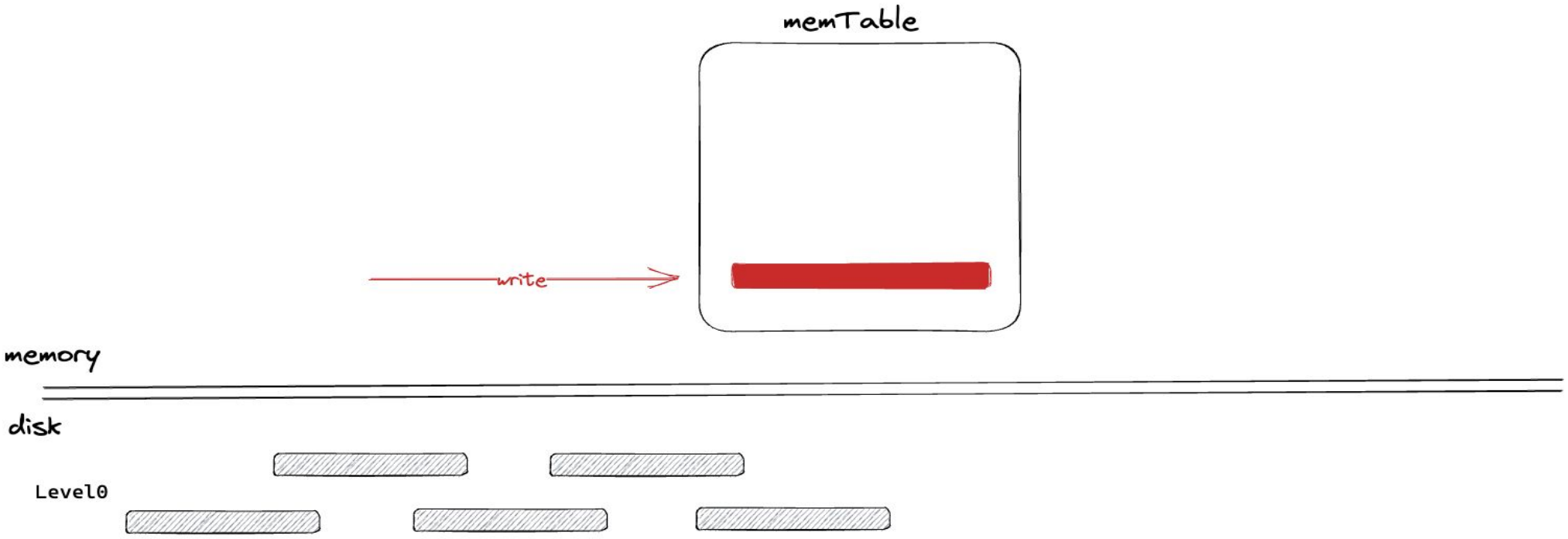
Sample size：
16 bytes -> 1 byte!

TSID ⇔ [illegible]_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"}

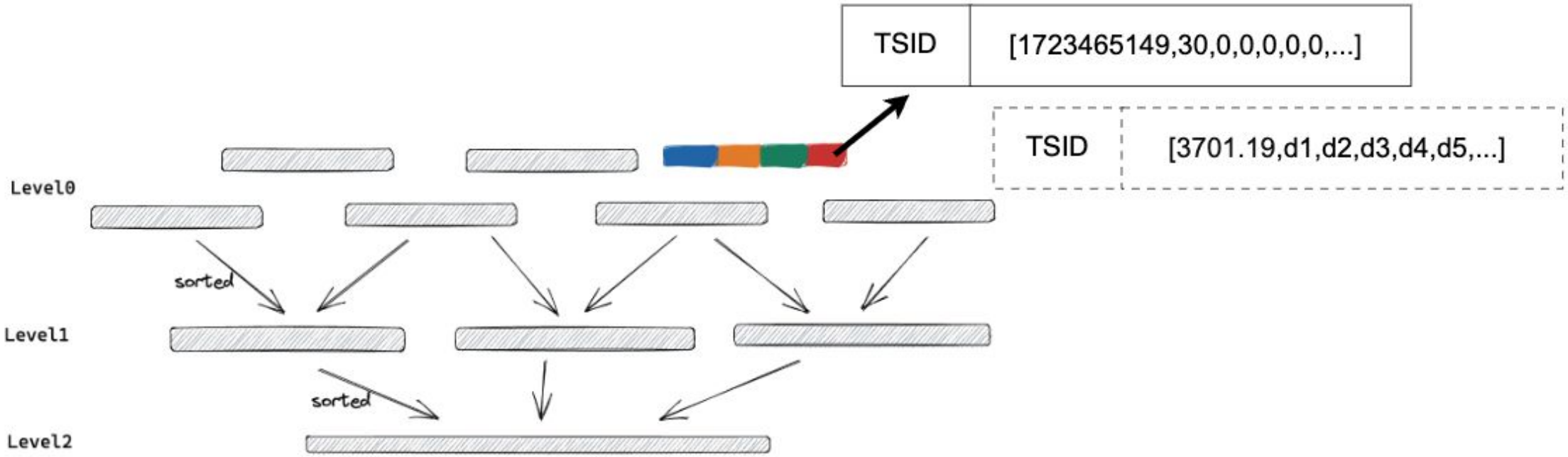
TSID ⇨ [3701.19,d1,d2,d3,d4,d5,...]

TSID ⇨ [1723465149,30,0,0,0,0,0,...]

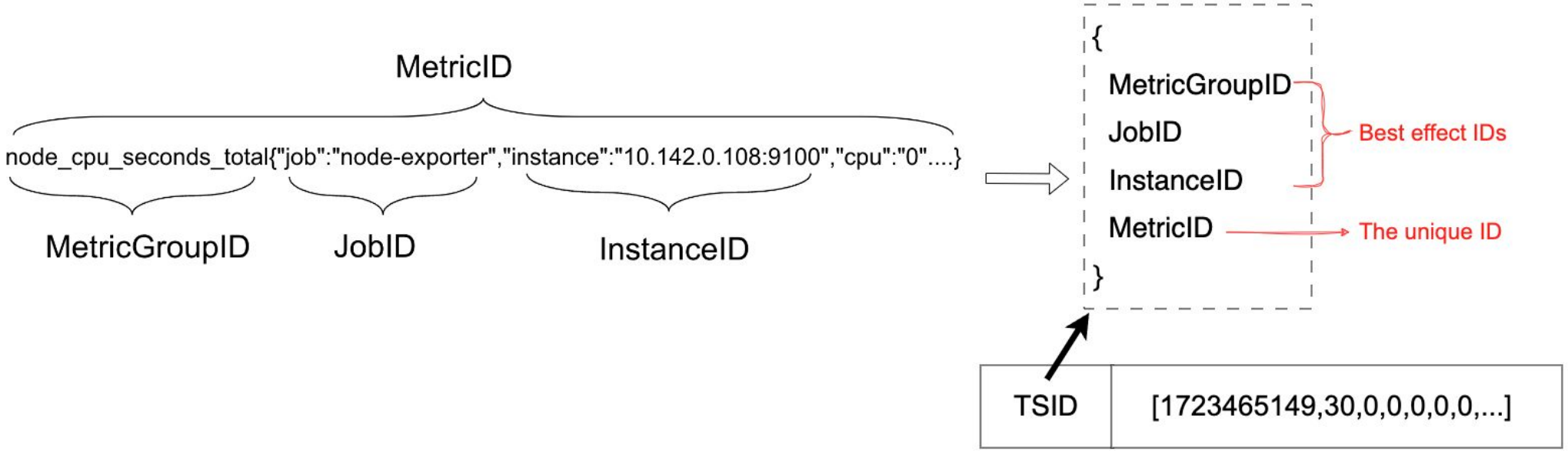
LSM tree

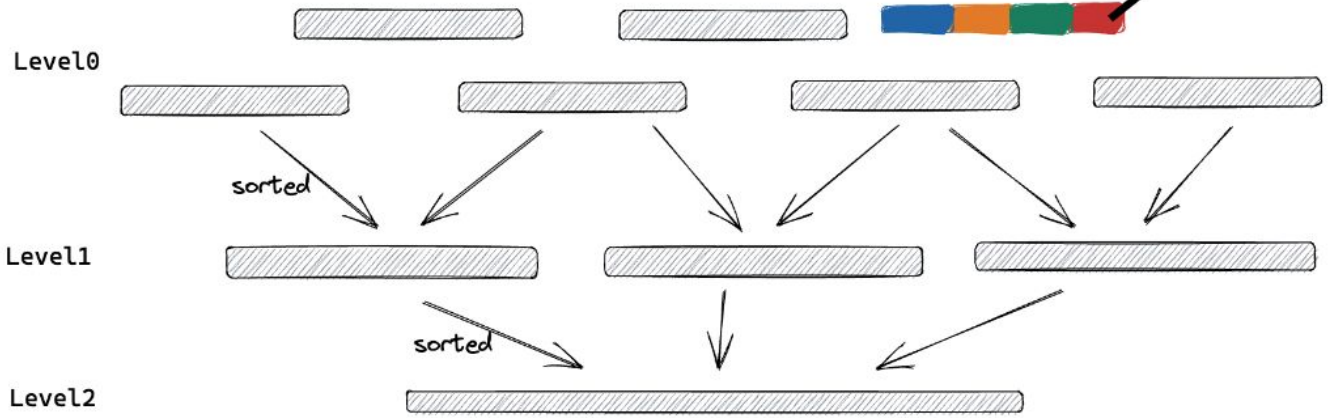


LSM tree


TSID
[1723465149,30,0,0,0,0,0,...]
TSID
[3701.19,d1,d2,d3,d4,d5,...]
Level0
sorted
Level1
sorted
Level2

LSM tree


MetricID
node_cpu_seconds_total{"job":"node-exporter","instance":"10.142.0.108:9100","cpu":"0"....}
MetricGroupID
JobID
InstanceID
{
MetricGroupID
JobID
InstanceID
MetricID
}
Best effect IDs
The unique ID
TSID
[1723465149,30,0,0,0,0,0,...]


Level0
sorted
Level1
sorted
Level2

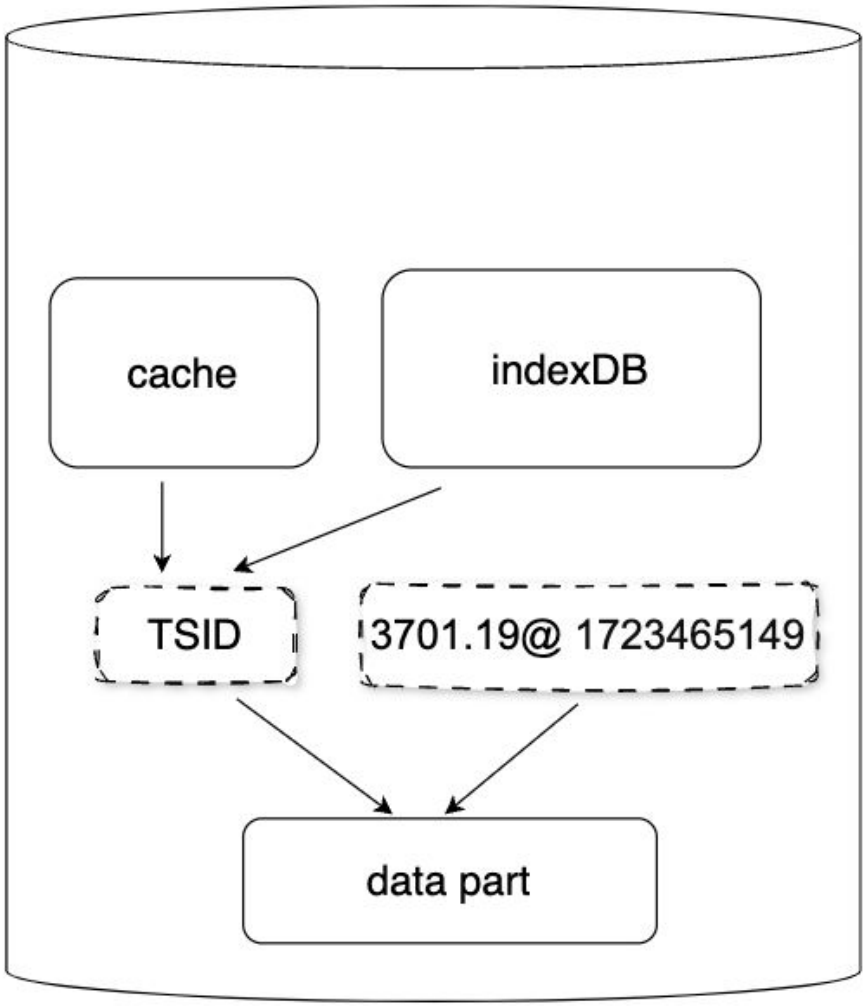
How to build a TSDB

1. Store time series names, timestamps and values separately(aka columnar storage), apply different compression algorithms on them;

2. Store each column in a data structure similar to log-structured merge tree (LSM).

A time series' journey: Write

node_cpu_seconds_total{instance="172.10.0.31:9100",cpu="0",mode="idle"}
3701.19 @ 1723465149



A time series' journey: Query

node_cpu_seconds_total{instance="172.10.0.31:9100",cpu="0"} @ 1723465149

search for TSID

node_cpu_seconds_total	3 7 41 50 101 300 411 1125 1995 2105 3340 5390 ...
instance="172.10.0.31:9100"	3 37 50 111 411 550 870 993 1125 1996 3890 ...
cpu="0"	1 50 111 138 341 411 993 1125 3340 3890 ...

A time series' journey: Query

node_cpu_seconds_total{instance="172.10.0.31:9100",cpu="0"} @ 1723465149

search for TSID

node_cpu_seconds_total: 3 7 41 50 101 300 411 1125 1995 2105 3340 5390 ...

instance="172.10.0.31:9100": 3 37 50 111 411 550 870 993 1125 1996 3890 ...

cpu="0": 1 50 111 138 341 411 993 1125 3340 3890 ...

Intersect

TSID: [50,411,1125]

A time series' journey: Query

node_cpu_seconds_total{instance="172.10.0.31:9100",cpu="0"} @ 1723465149

search TSID

TSID: [50,411,1125]

search data blocks

search metric metadata

50	3701.19 @ 1723465149
411	13467.83 @ 1723465149
1125	148.5 @ 1723465149

50	node_cpu_seconds_total{instance="172.10.0.31:9100.",cpu="0",mode="idle"}
411	node_cpu_seconds_total{instance="172.10.0.31:9100",cpu="0",mode="user"}
1125	node_cpu_seconds_total{instance="172.10.0.31:9100",cpu="0",mode="iowait"}

return

node_cpu_seconds_total{instance="172.10.0.31:9100",cpu="0",mode="idle"} 3701.19 @ 1723465149
node_cpu_seconds_total{instance="172.10.0.31:9100",cpu="0",mode="user"} 13467.83 @ 1723465149
node_cpu_seconds_total{instance="172.10.0.31:9100",cpu="0",mode="iowait"} 148.5 @ 1723465149

With great TSDB comes great data:)

Stats

Total datapoints	Ingestion rate	Read requests	Available CPU
145 Tri	54.0 Mil	19.2 req/s	2.37 K
Active series	**Disk space usage**	**Bytes per point**	**Available memory**
556 Mil	82.0 TiB	0.62 B	3.99 TiB

[VictoriaMetrics: scaling to 100 million metrics per second]

Ingestion samples/s	Active series	Bytes per point	Available CPU
101 Mil	1.01 Bil	0.85 B	1024

job	Count
victoria-metrics-cluster-vminsert	18
victoria-metrics-cluster-vmstorage	46

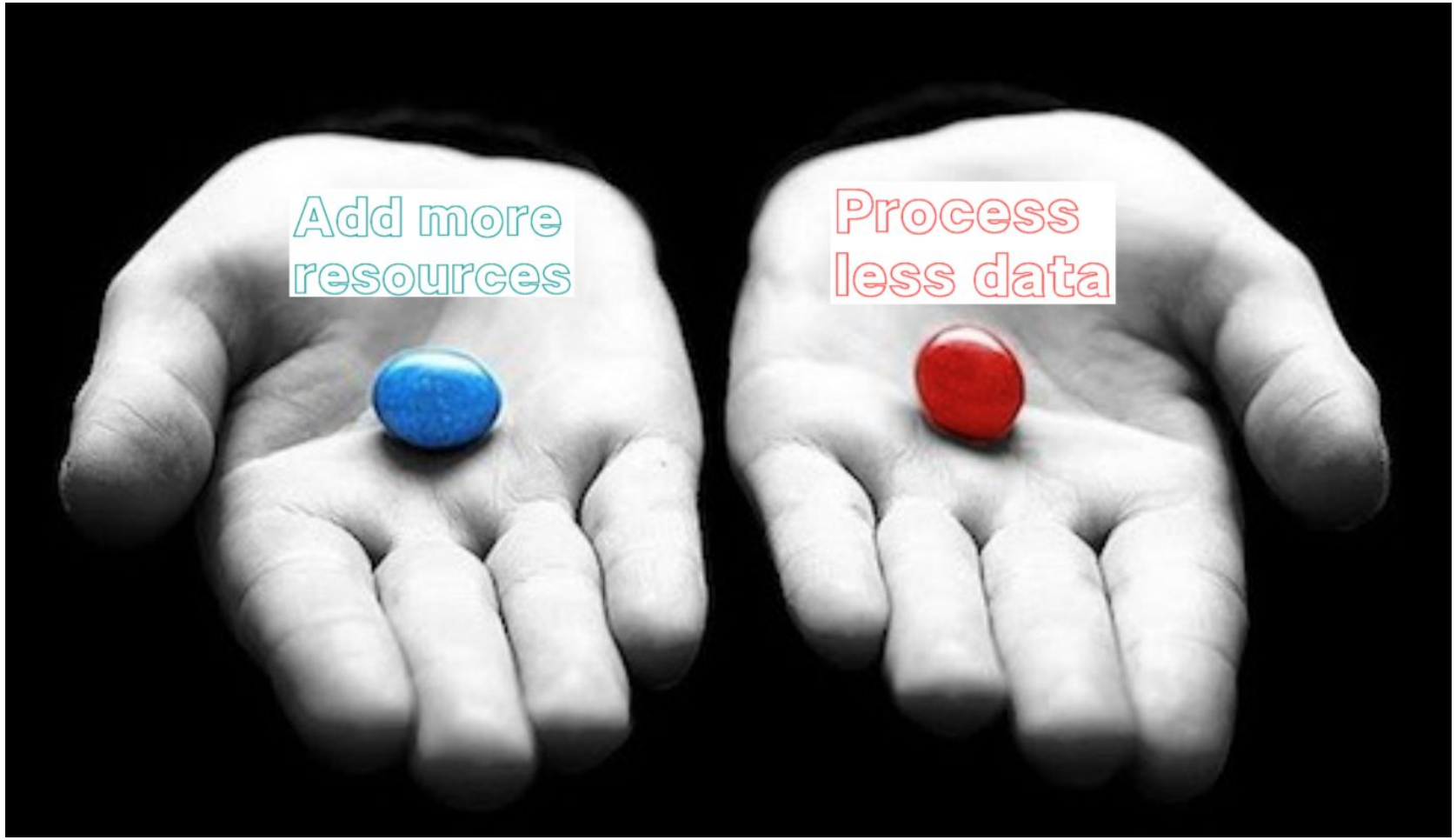
Total datapoints	Disk space usage	Available memory
8.77 Tri	7.48 TiB	3.44 TiB

See more: https://docs.victoriametrics.com/casestudies/

TSDB camp

1. ClickHouse
2. TimescaleDB
3. Prometheus
4. VictoriaMetrics

How to improve experiences



How to improve experiences

VMUI Query Explore Tools Dashboards 2024-08-16

Time series selector

Focus label

Limit entries 10

Total series
397,157 ↑146.01%

Documentation RESET EXECUTE QUERY

Metric names with the highest number of series

Table Graph

Metric name	Number of series	Share in total
apiserver_request_duration_seconds_bucket	18204 +5532	4.58% ↓-3.27%
etcd_request_duration_seconds_bucket	13427 +3995	3.38% ↓-2.46%
apiserver_request_sli_duration_seconds_bucket	11718 +3028	2.95% ↓-2.43%
github_downloads_total	10880 +5440	2.74% ↓-0.63%
flag	8636 +5491	2.17% ↑0.23%
apiserver_request_body_size_bytes_bucket	7282 +2514	1.83% ↓-1.12%
container_memory_failures_total	7259 +4545	1.83% ↑0.15%
storage_operation_duration_seconds_bucket	6320 +3968	1.59% ↑0.13%
grpc_server_handled_total	5561	1.40% ↑1.40%
kubernetes_feature_enabled	2345	0.59% ↑0.59%

Labels with the highest number of series

Table Graph

Label name	Number of series	Share in total
__name__	397157 +235718	100.00% 0.00%
job	392610 +233390	98.86% ↑0.23%
cluster	392338 +233246	98.79% ↑0.24%
prometheus	392338 +233246	98.79% ↑0.24%
instance	384840 +229390	96.90% ↑0.61%
namespace	336266 +195614	84.67% ↓-2.46%
endpoint	255271 +144057	64.27% ↓-4.61%
pod	248435 +165507	62.55% ↑11.19%
service	244203 +139690	61.49% ↓-3.25%

https://play.victoriametrics.com/select/accounting/1/6a716b0f-38bc-4856-90ce-448fd713e3fe/prometheus/graph/#/cardinality

How to improve experiences

Reduce Cardinality !!!

Relabeling


```
metricRelabelings:
- regex: "prometheus_replica"
  action: labeldrop
```

node_cpu_seconds_total{job:"node-exporter",instance="172.10.0.31:9100","cpu":"0",prometheus_replica="0/1"}

=> node_cpu_seconds_total{job:"node-exporter",instance="172.10.0.31:9100","cpu":"0"}

```
metricRelabelings:
- action: drop
  regex: apiserver_request_duration_seconds_bucket;(0.15|0.25|0.3|0.35|0.4|0.8|0.9|1.25|1.75|2.5|3|3.5|4.5)
  sourceLabels:
  - __name__
  - le
```

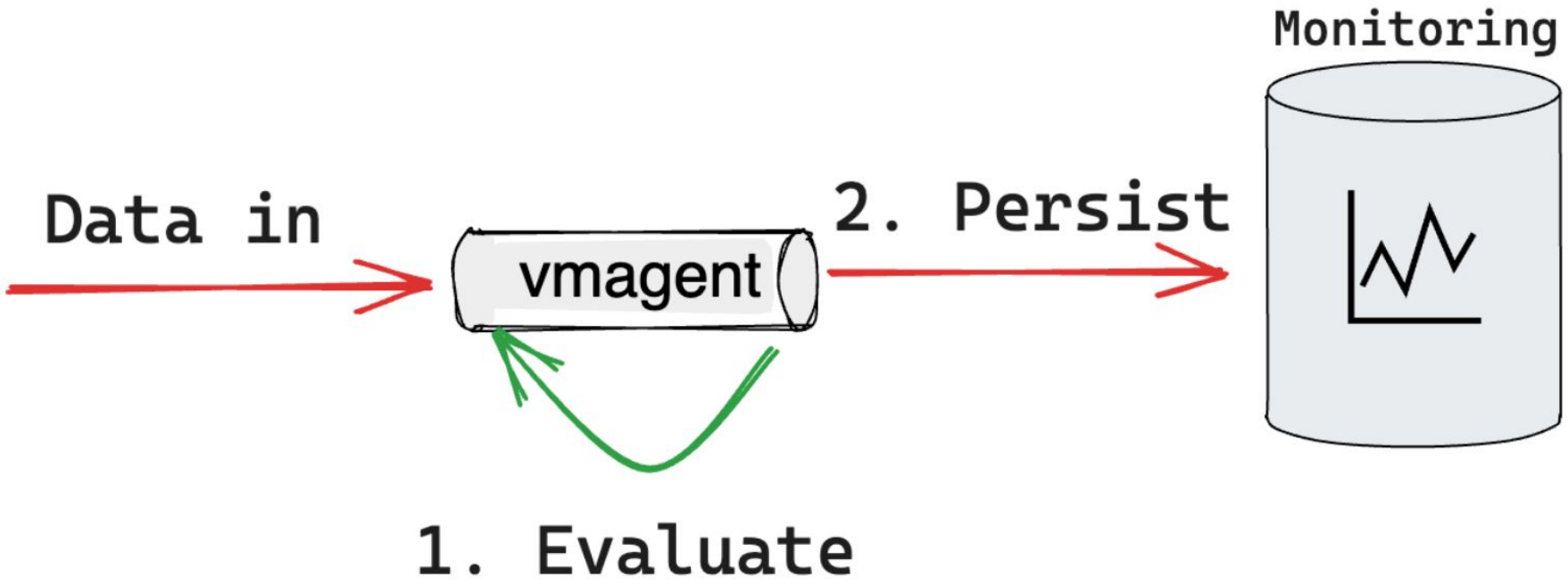
apiserver_request_duration_seconds_bucket{"job":"apiserver",resource:"pods",le="0.025/0.1/0.15/0.3/0.5/0.8/1/1.25/1.5/2/…"}

=> apiserver_request_duration_seconds_bucket{"job":"apiserver",resource:"pods",le="0.025/0.1/0.5/1/1.5/2/…"}

https://docs.victoriametrics.com/relabeling/

Streaming aggregation

The number of **time series or/and samples** stored in TSDB is **what needs to be persisted**



Streaming aggregation

1. Aggregate:

```
- match: "node_cpu_seconds_total"    # time series selector
  interval: "30s"                    # on 30s interval
  outputs: ["total"]                 # aggregate as counter
  without: ["cpu"]                   # group without label
```

node_cpu_seconds_total{job:"node-exporter",instance="172.10.0.31:9100","cpu":"0","mode":"user"}

node_cpu_seconds_total{job:"node-exporter",instance="172.10.0.31:9100","cpu":"1","mode":"user"}

node_cpu_seconds_total{job:"node-exporter",instance="172.10.0.31:9100","cpu":"2","mode":"user"}

=>

node_cpu_seconds_total:30s_without_cpu_total{job:"node-exporter",instance="172.10.0.31:9100","mode":"user"}

Streaming aggregation

2. Deduplicate:

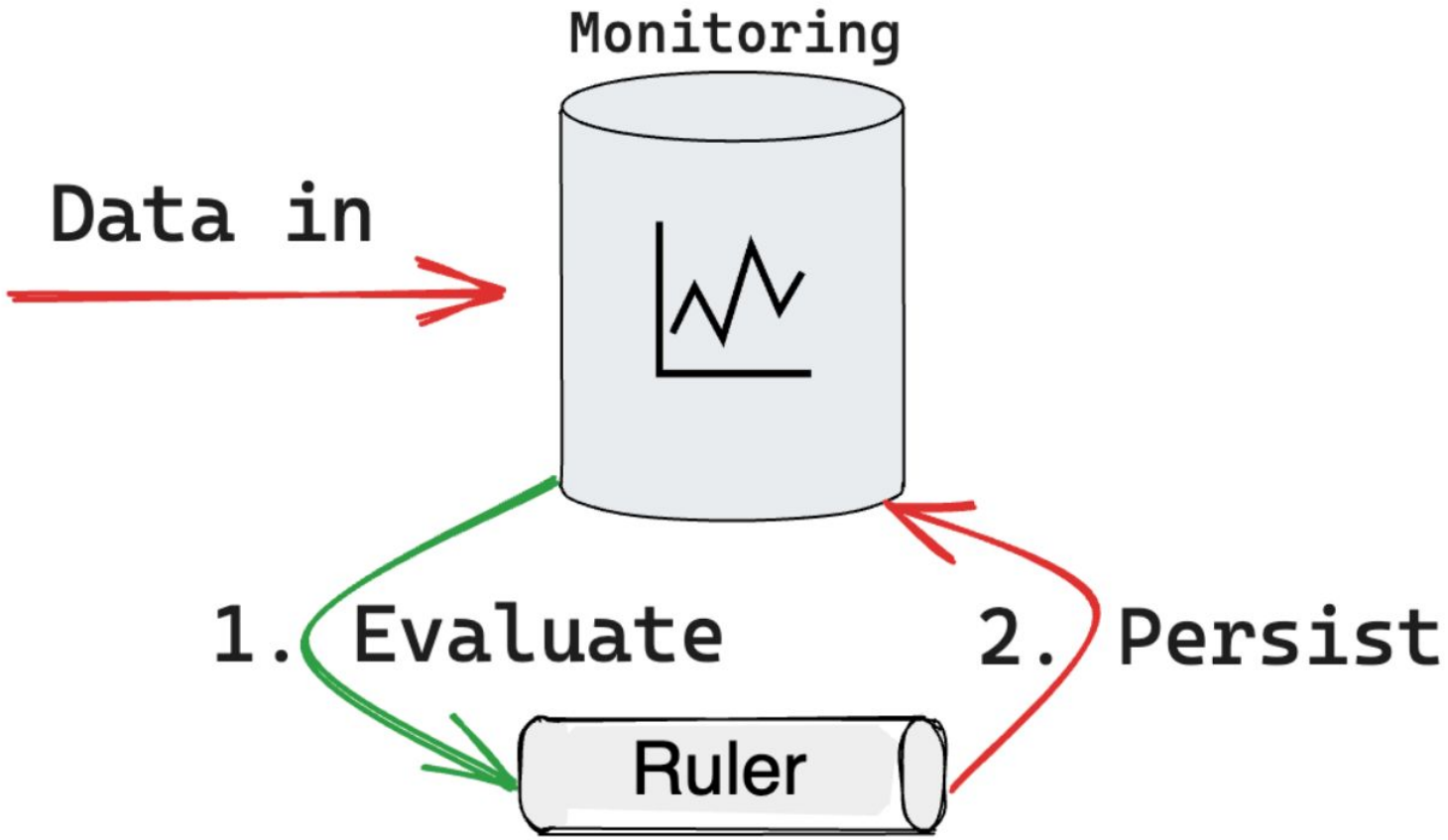
```
- match: "kube_pod_.*"          # time series selector
  interval: "1m"                # on 1m interval
  outputs: ["last"]             # leave the last input sample value over the given interval
  keep_metric_names: true       # keeping the original metric names for the aggregated samples
```

Command-line flags:

```
-remoteWrite.streamAggr.dedupInterval array
  Input samples are de-duplicated with this interval before optional aggregation with -remoteWrite.streamAggr.config
  at the corresponding -remoteWrite.url
-streamAggr.dedupInterval value
  Input samples are de-duplicated with this interval on aggregator before optional aggregation with -streamAggr.config
```

Recording rule

The number of time series stored in TSDB is **Data-in + Recording Rules results**



More to read

- https://web.archive.org/web/20210803115658/https://fabxc.org/tsdb/
- https://valyala.medium.com/how-victoriametrics-makes-instant-snapshots-for-multi-terabyte-time-series-data-e1f3fb0e0282
- https://victoriametrics.com/blog/tsdb-performance-techniques-strings-interning/

Q&A