Grafana Labs

Lifecycle of a Sample in the Prometheus TSDB

Ganesh Vernekar

SRECon, Sydney, 7 Dec 2022

Ganesh Vernekar

Prometheus Team Member
Senior Software Engineer, Grafana Labs

Twitter @_codesome
Mastodon @codesome@hachyderm.io

What is Prometheus?

Metrics-based monitoring & alerting stack.

- Instrumentation your applications and systems
- Metrics collection and storage
- Querying, alerting, dashboarding

Timeseries

http_requests_total{job="nginx", instance="1.2.3.4:80", status="200"}

Series identifier

⇒ [

(1670214381.809, 100) // (unix timestamp, value)

Sample

(1670214396.809, 130)

(1670214411.809, 157)

(1670214426.809, 198)

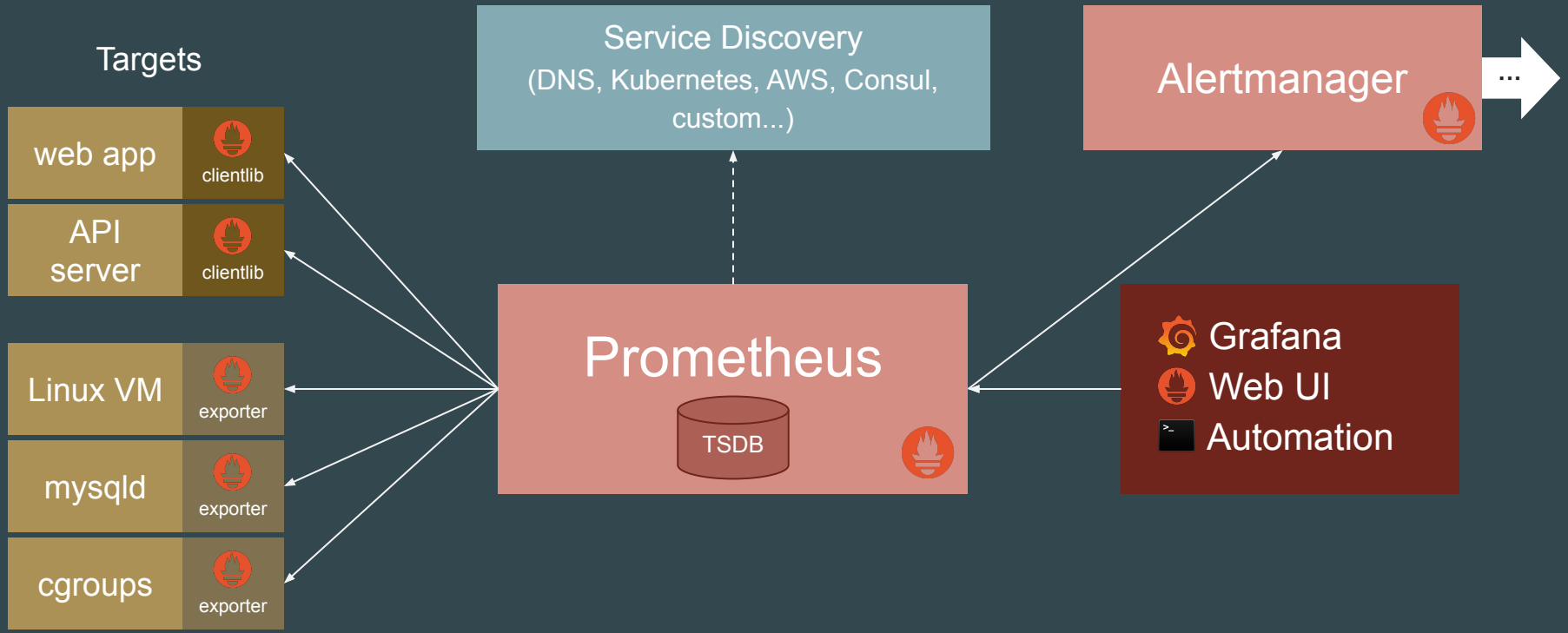
(1670214441.809, 220)

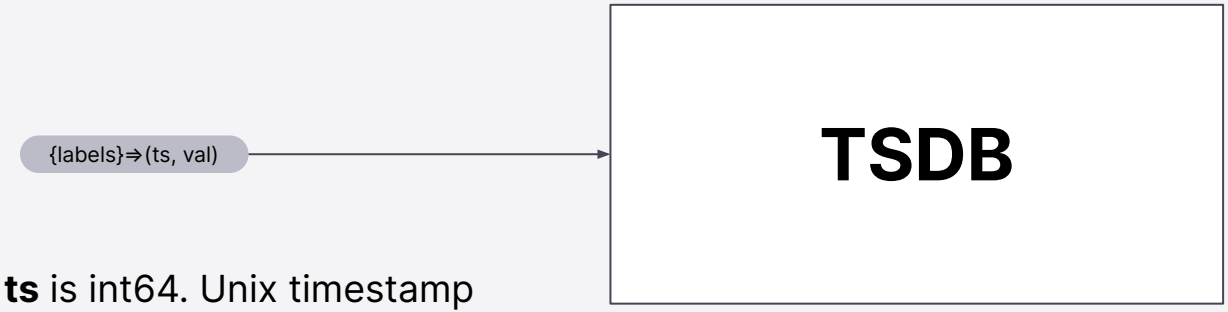
...

]

*Slide from Prometheus intro at KubeCon

Prometheus Architecture

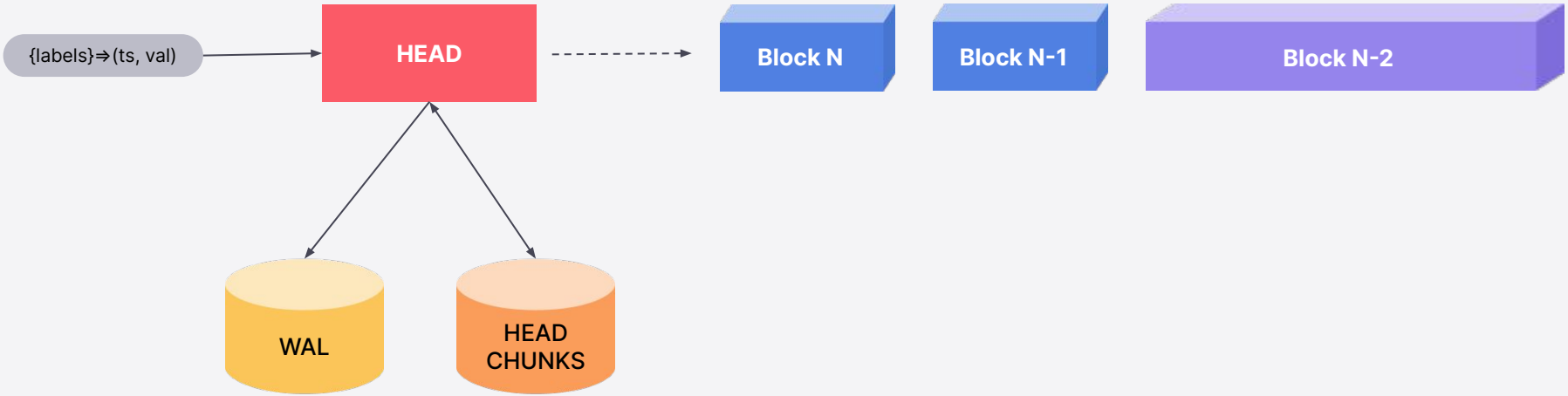




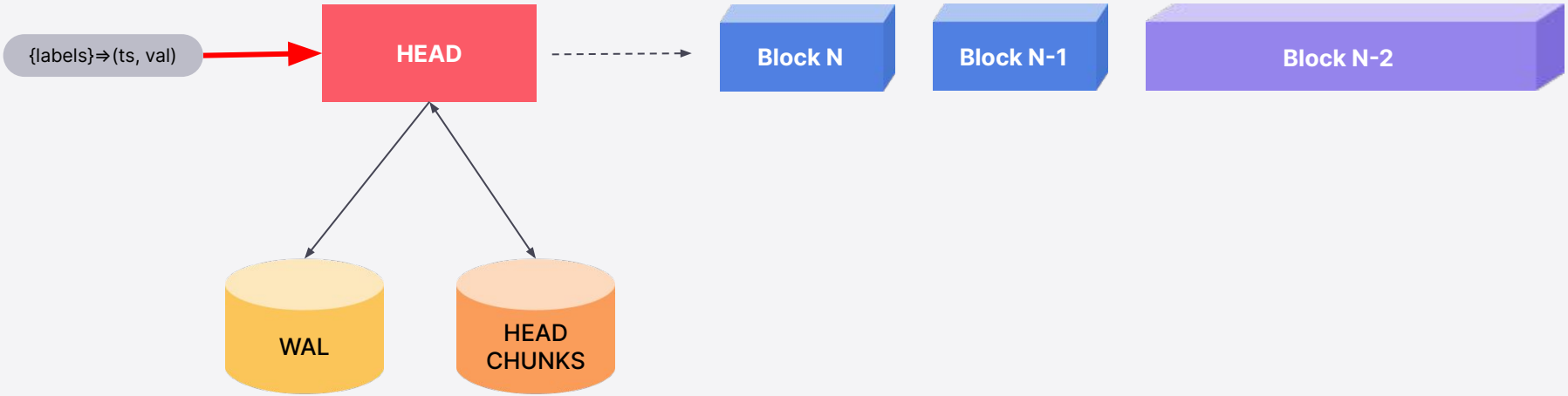
ts is int64. Unix timestamp in milliseconds.

val was only float64; recently got support for native histograms

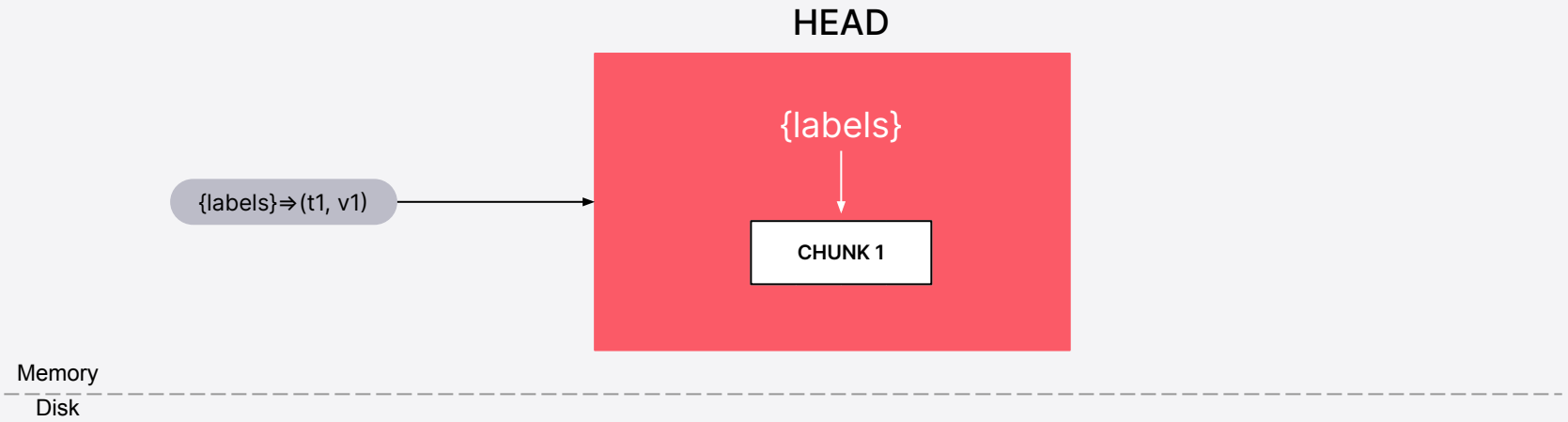
TSDB overview



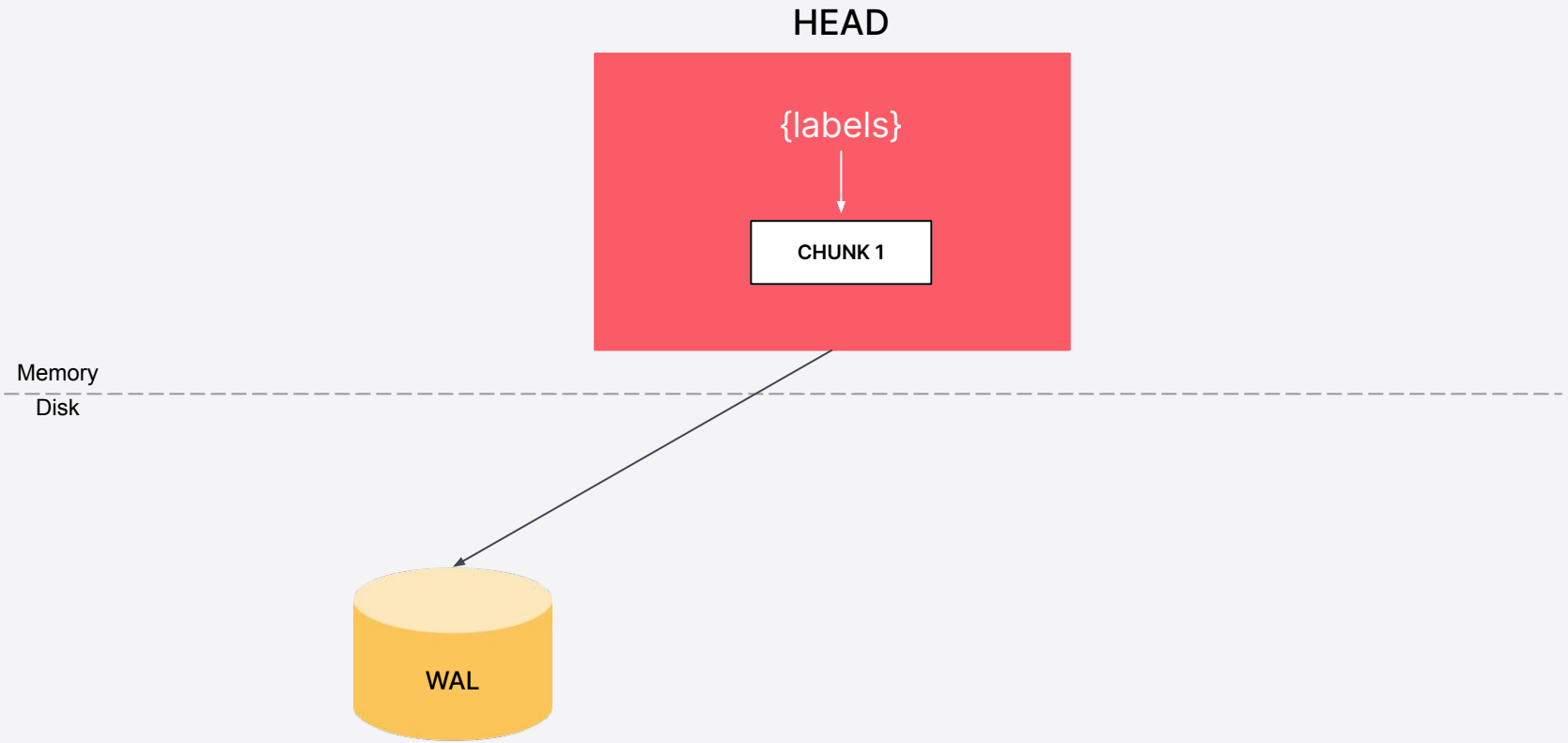
Writing a sample to HEAD



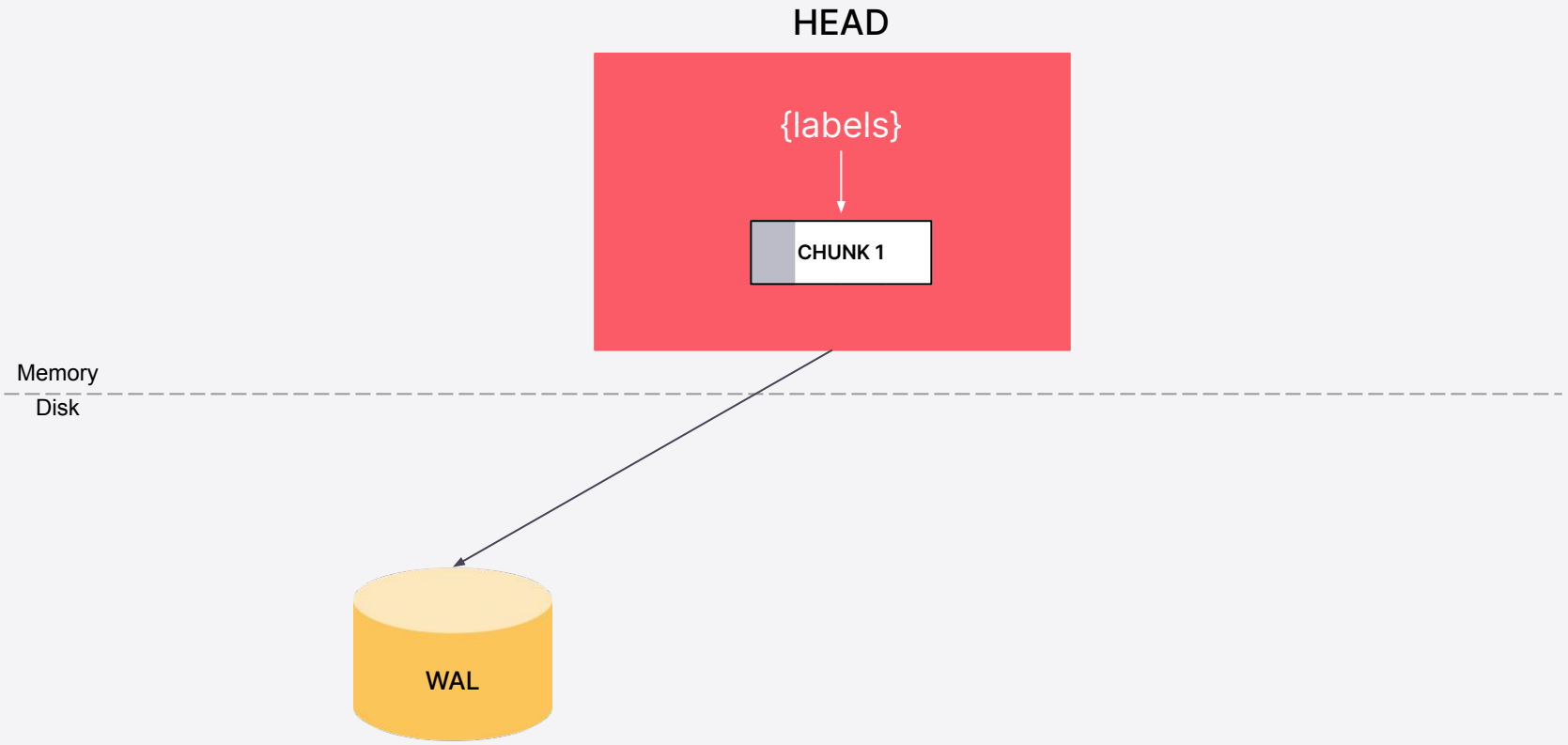
Zooming into Head



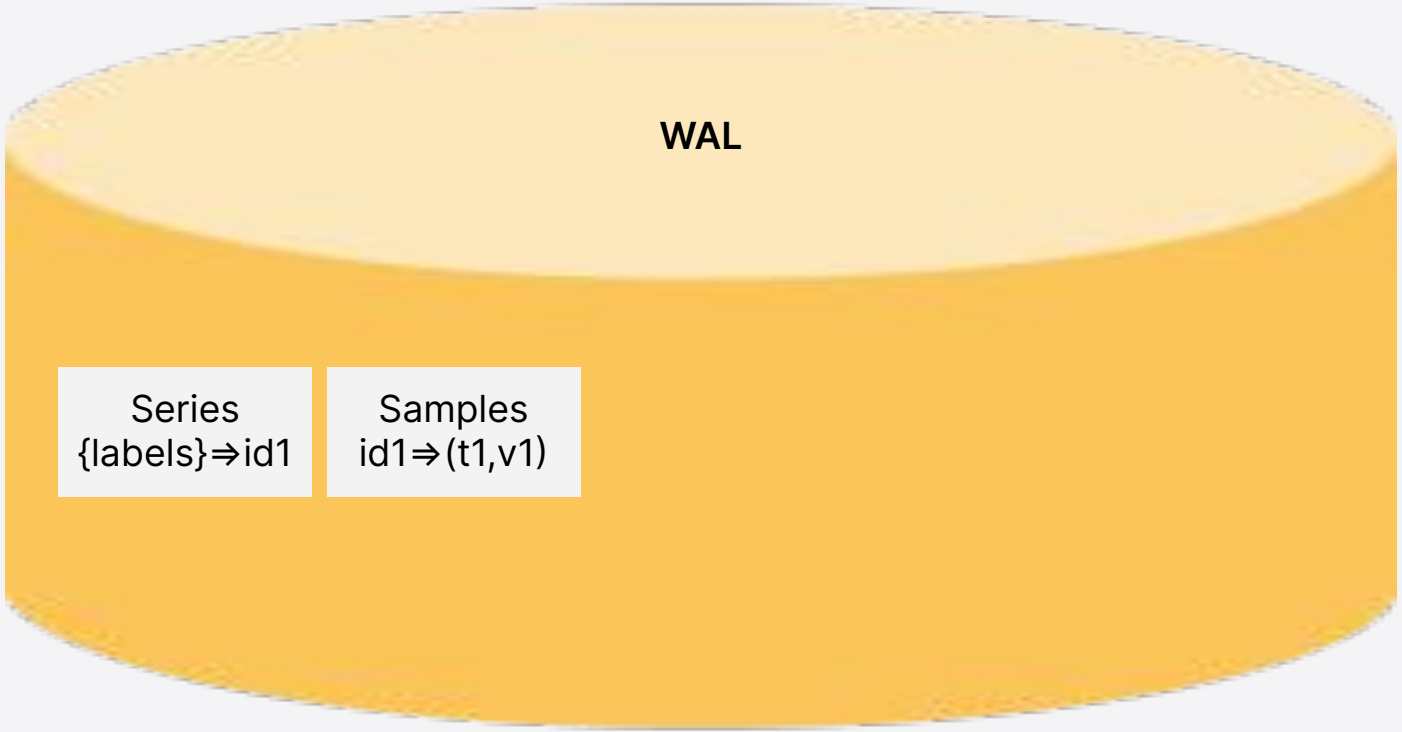
Sample first goes into the WAL



Then the in-memory chunk



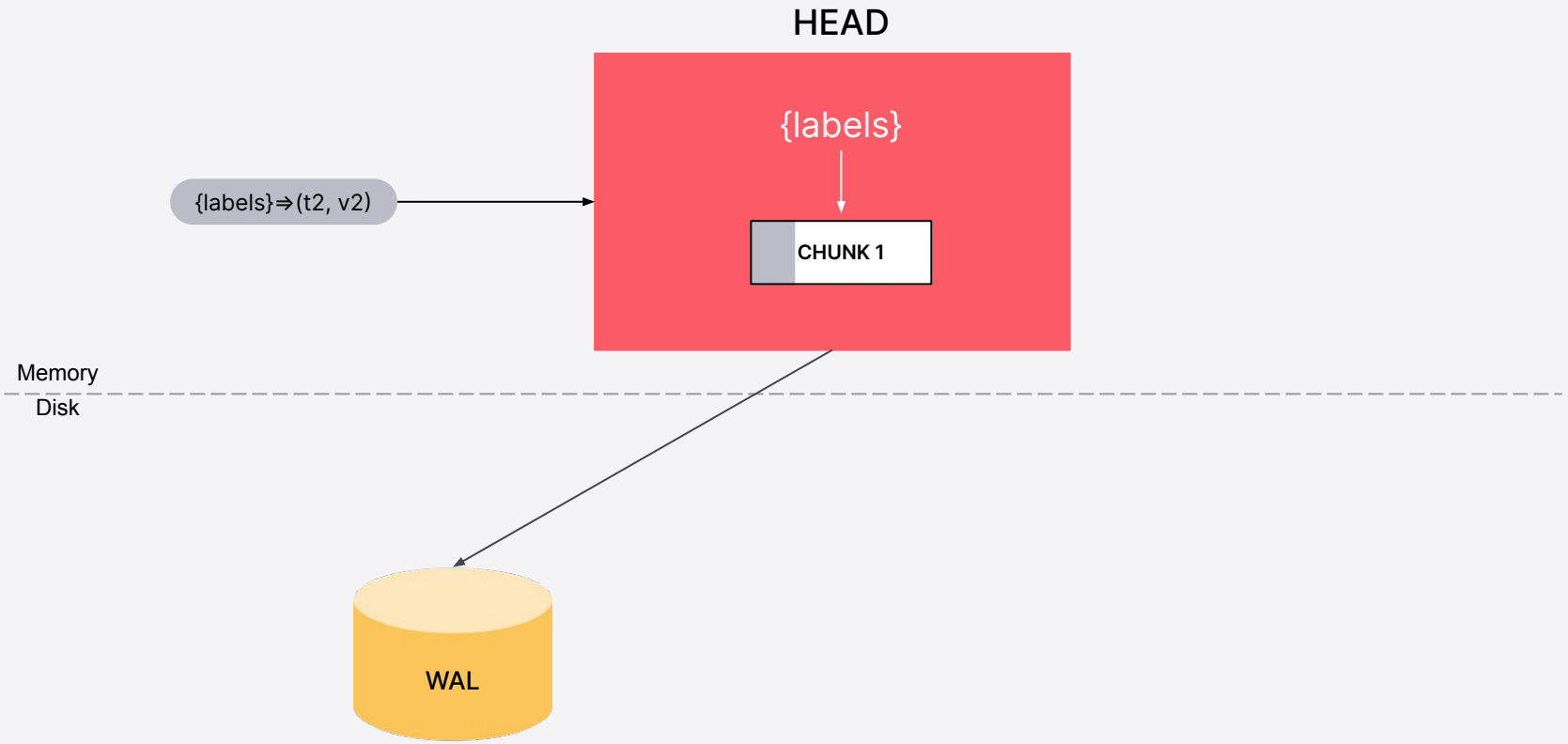
WAL - Write Ahead Log - logs write events



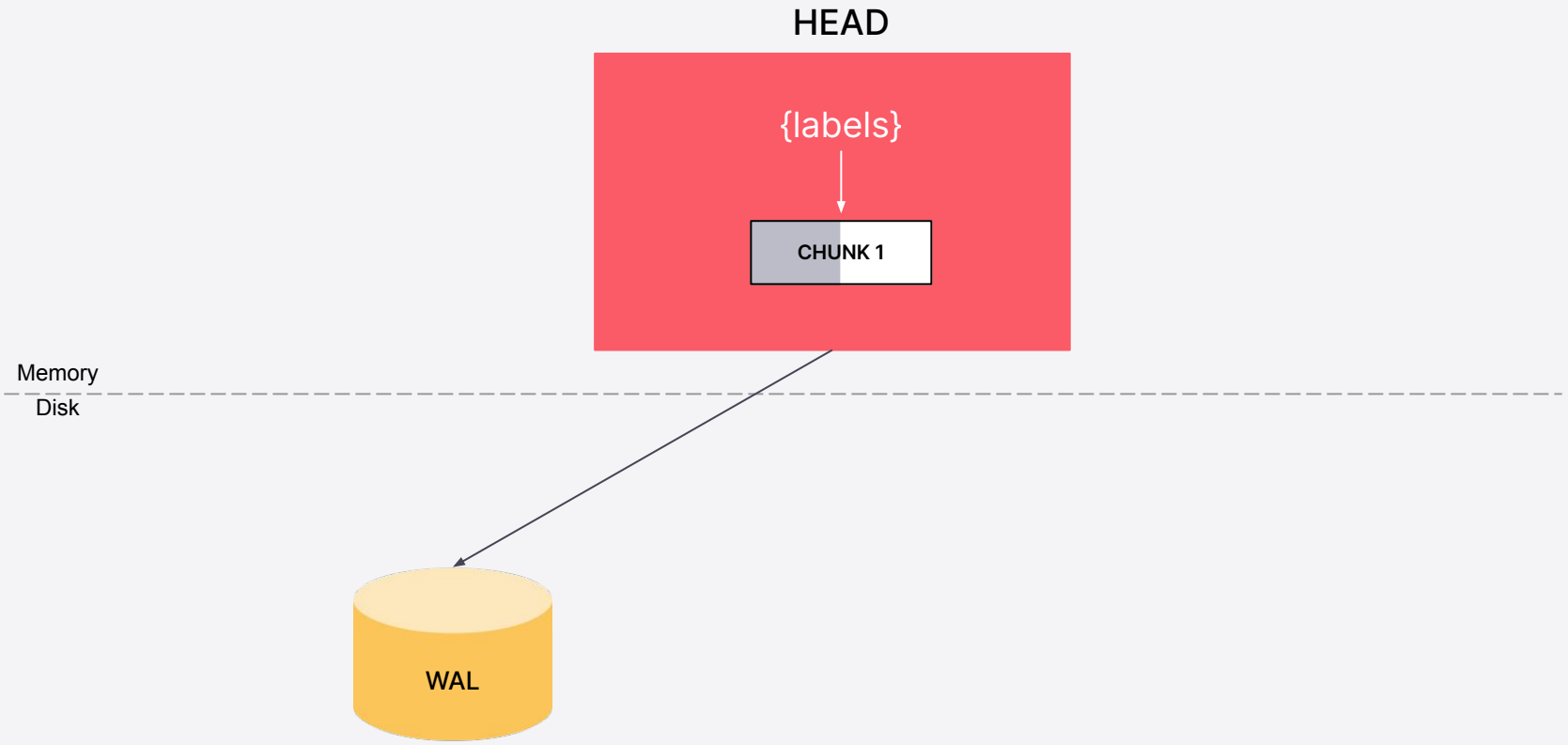
Why do we need WAL?

Durability against crashes

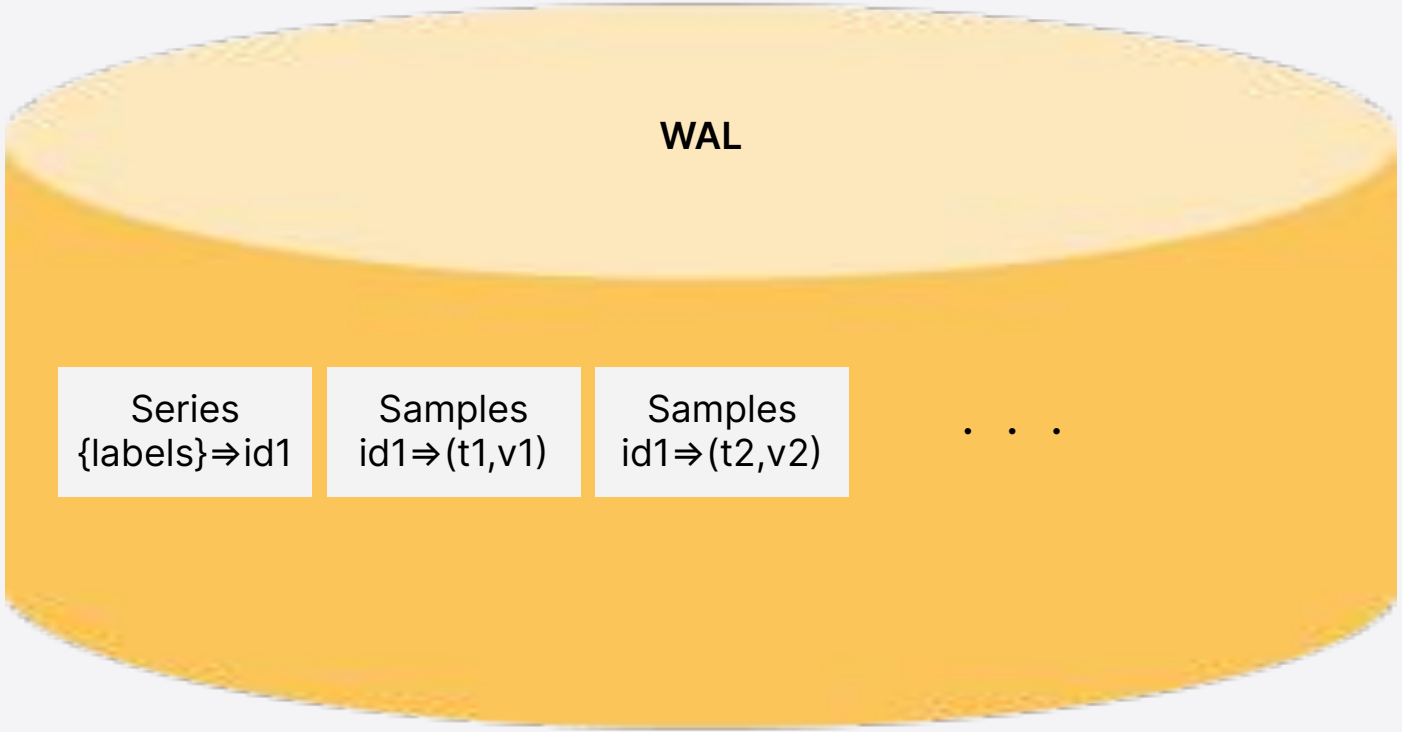
Same process repeated



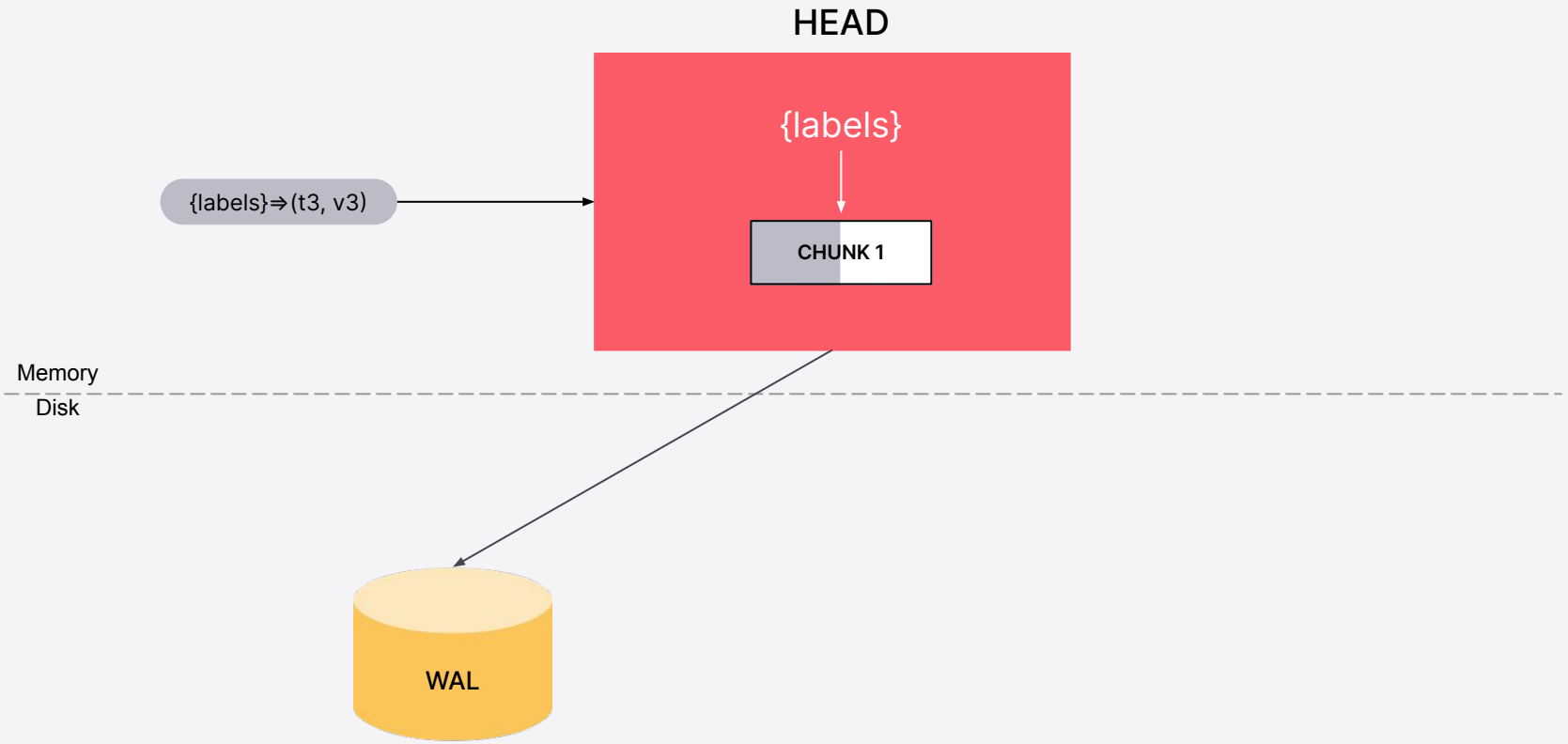
Same process repeated



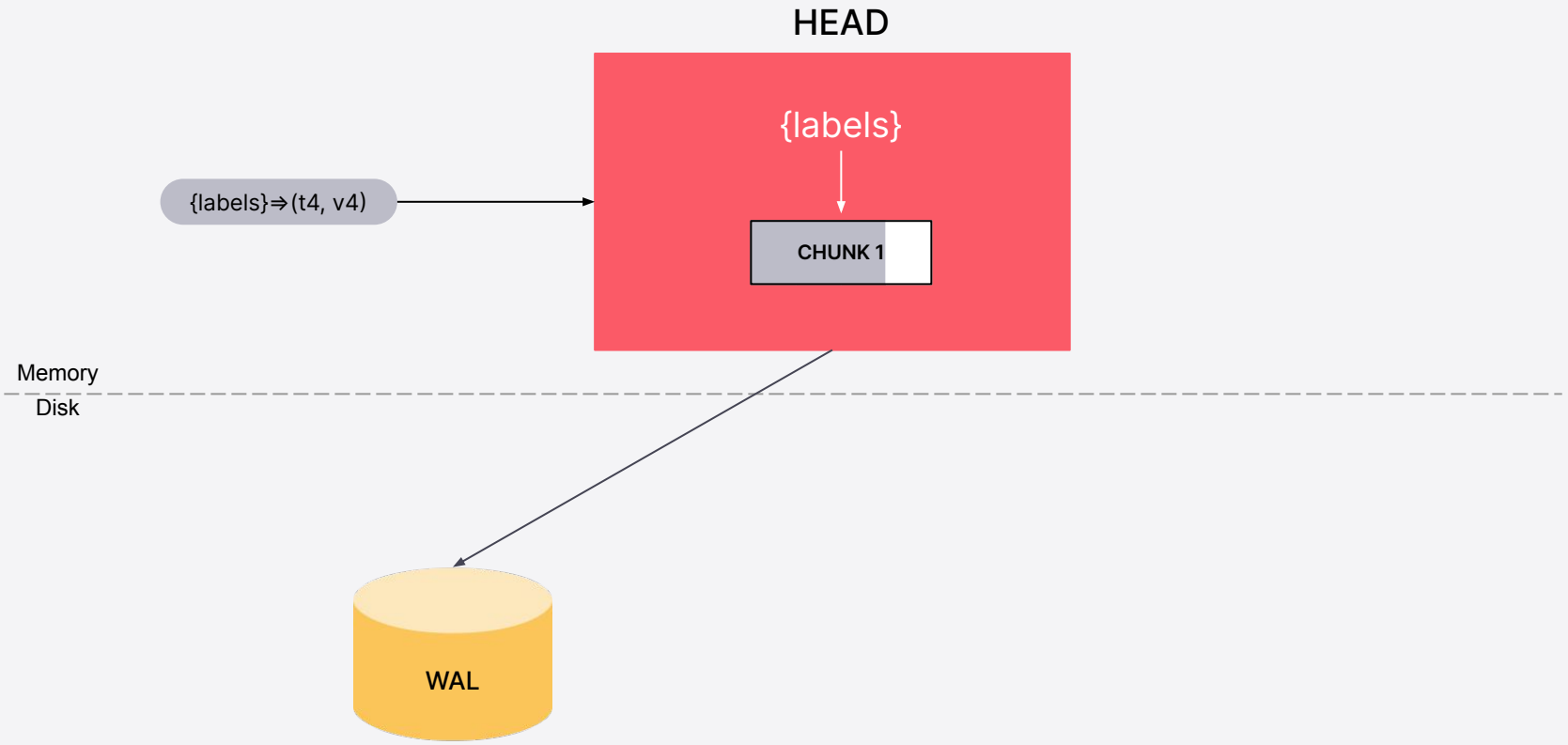
Avoiding duplicate Series record



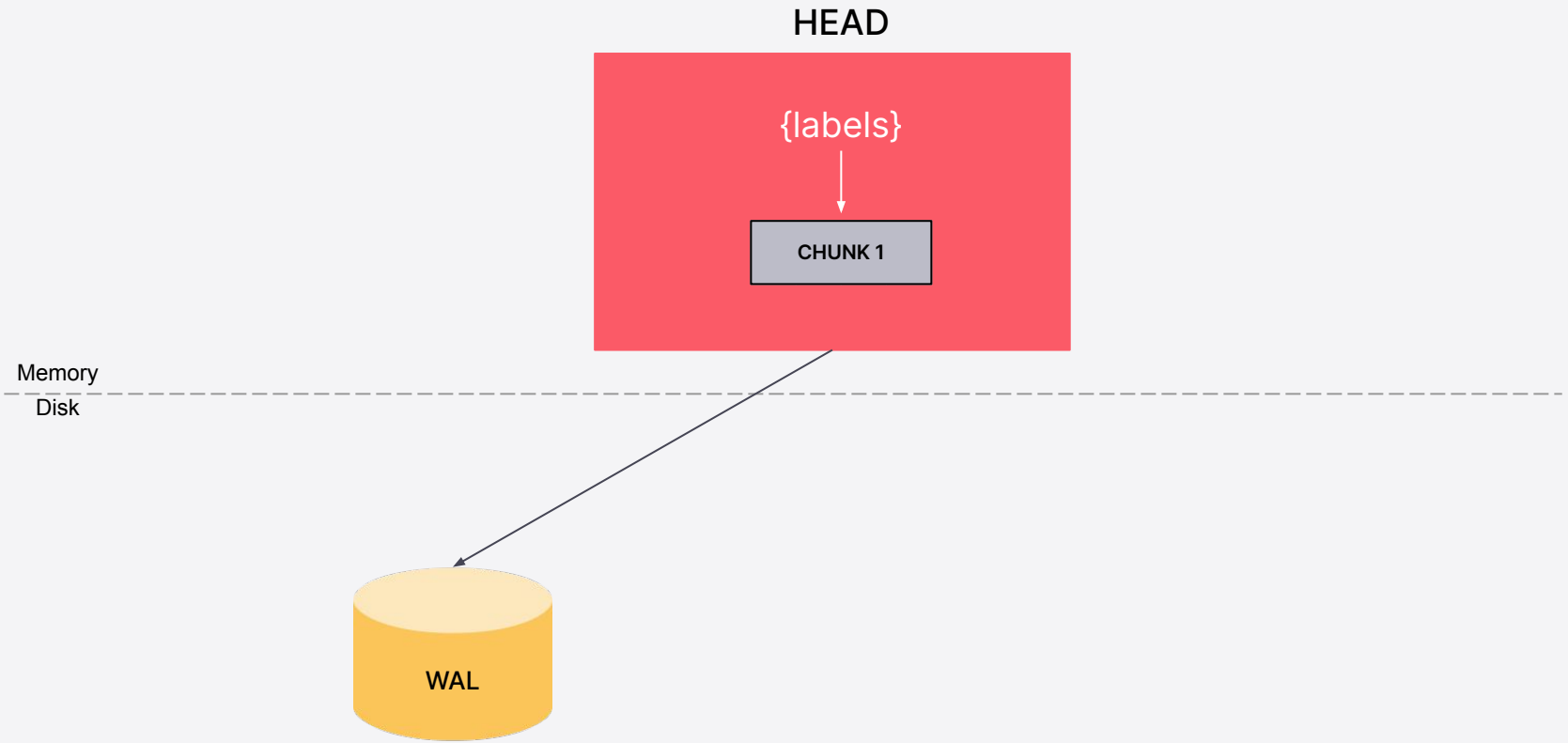
Same process repeated



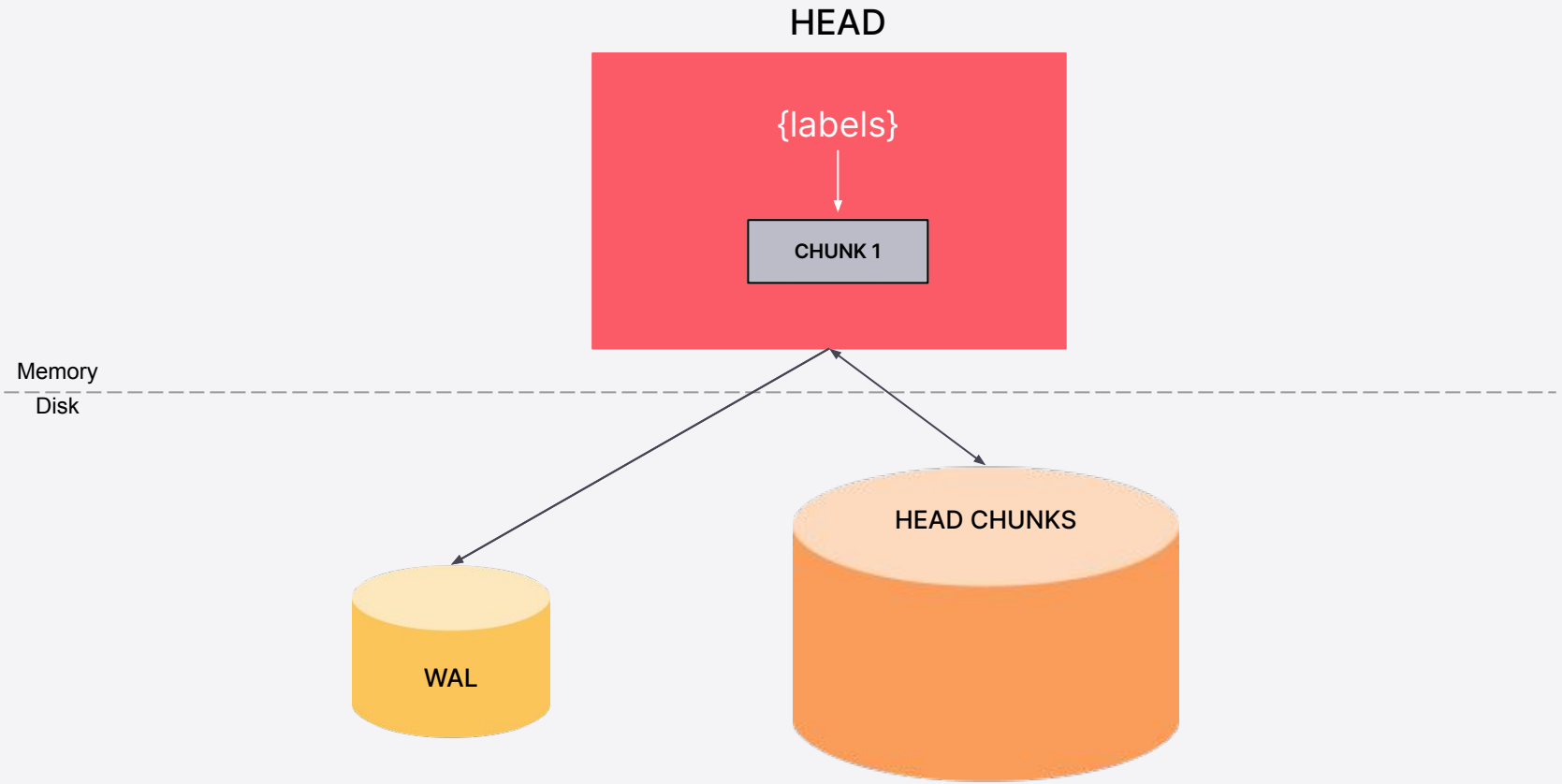
Same process repeated



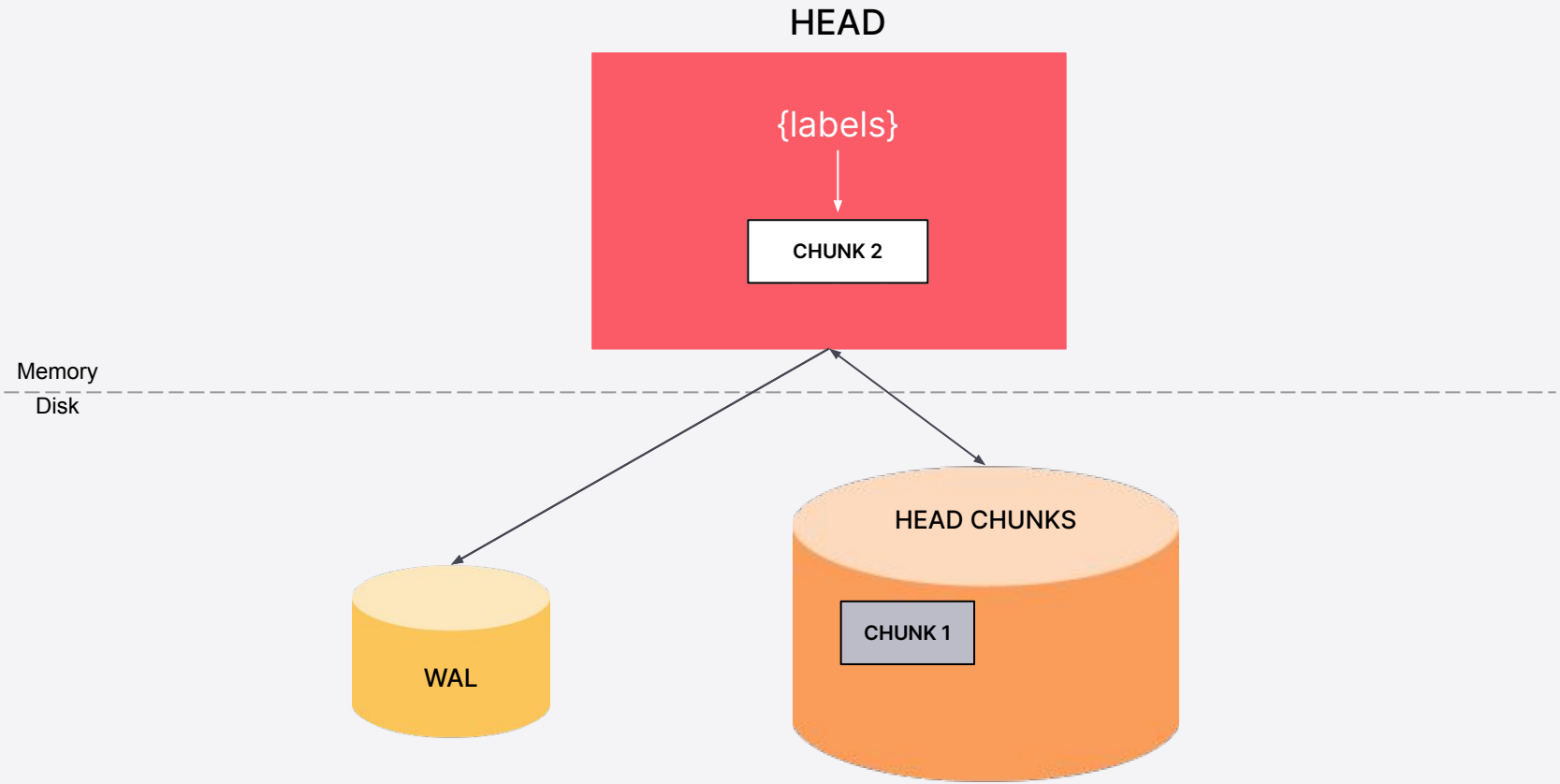
Chunk is now full!



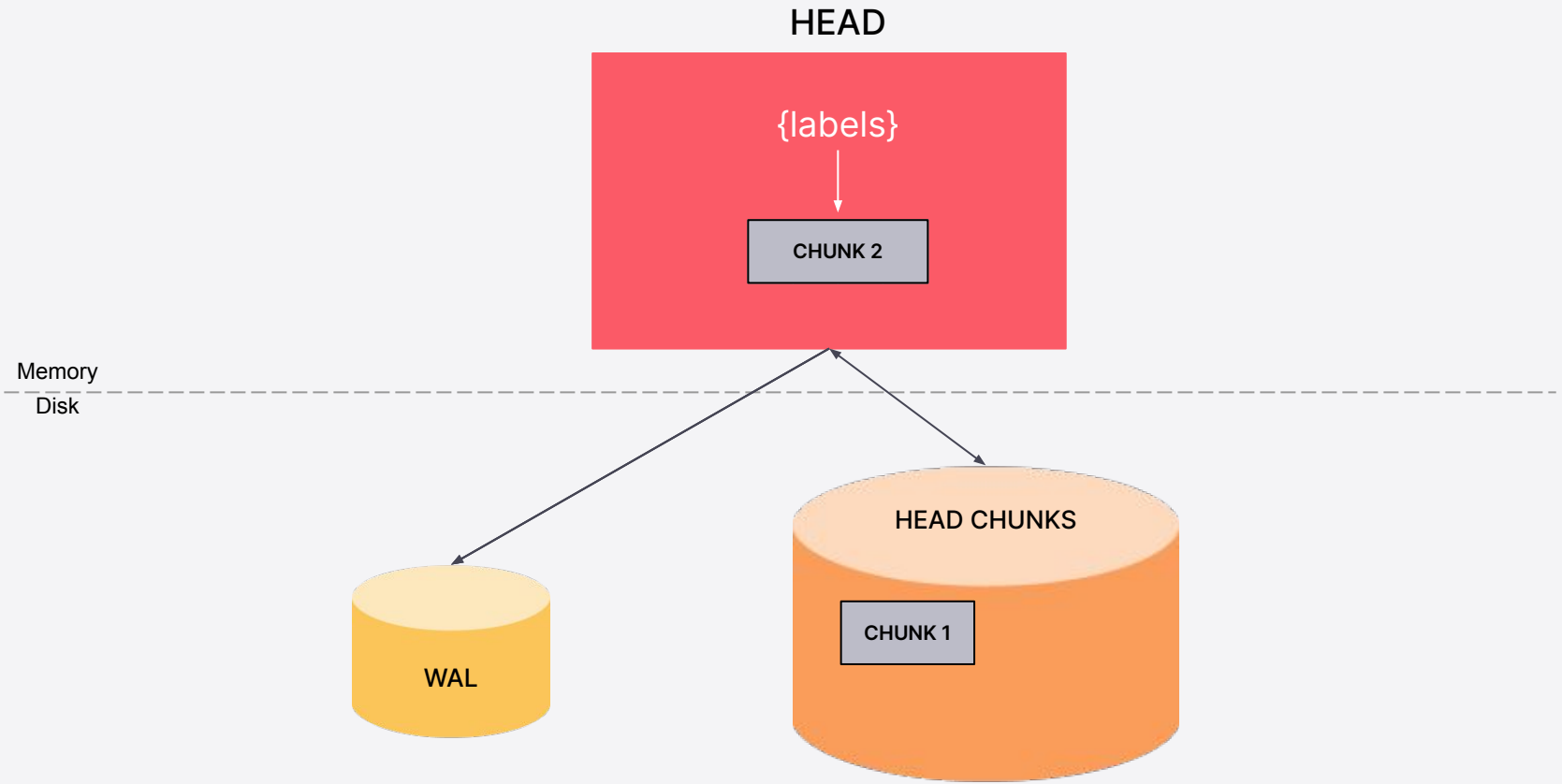
HEAD CHUNKS gets in action



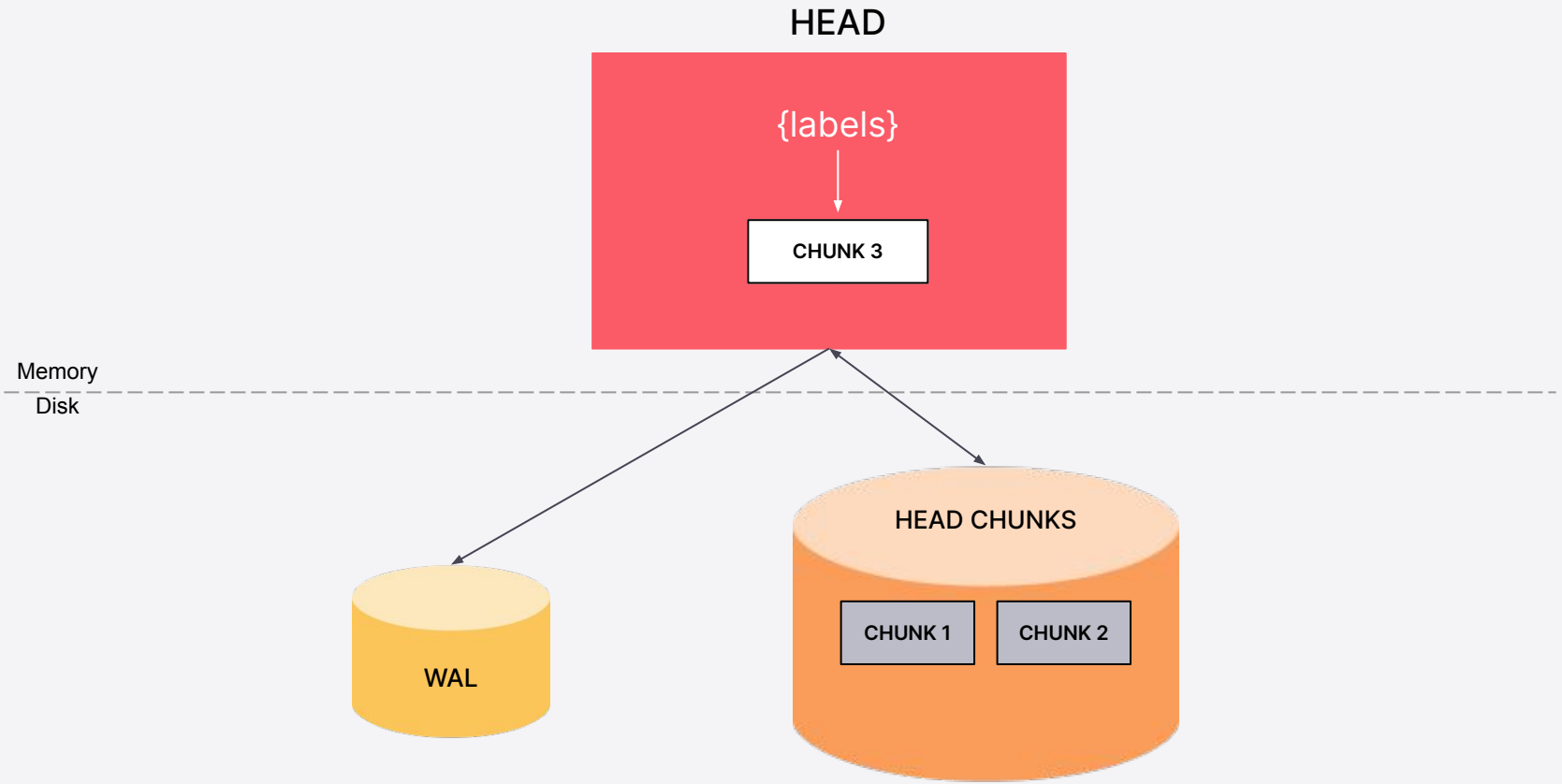
Flush it to the disk and m-map



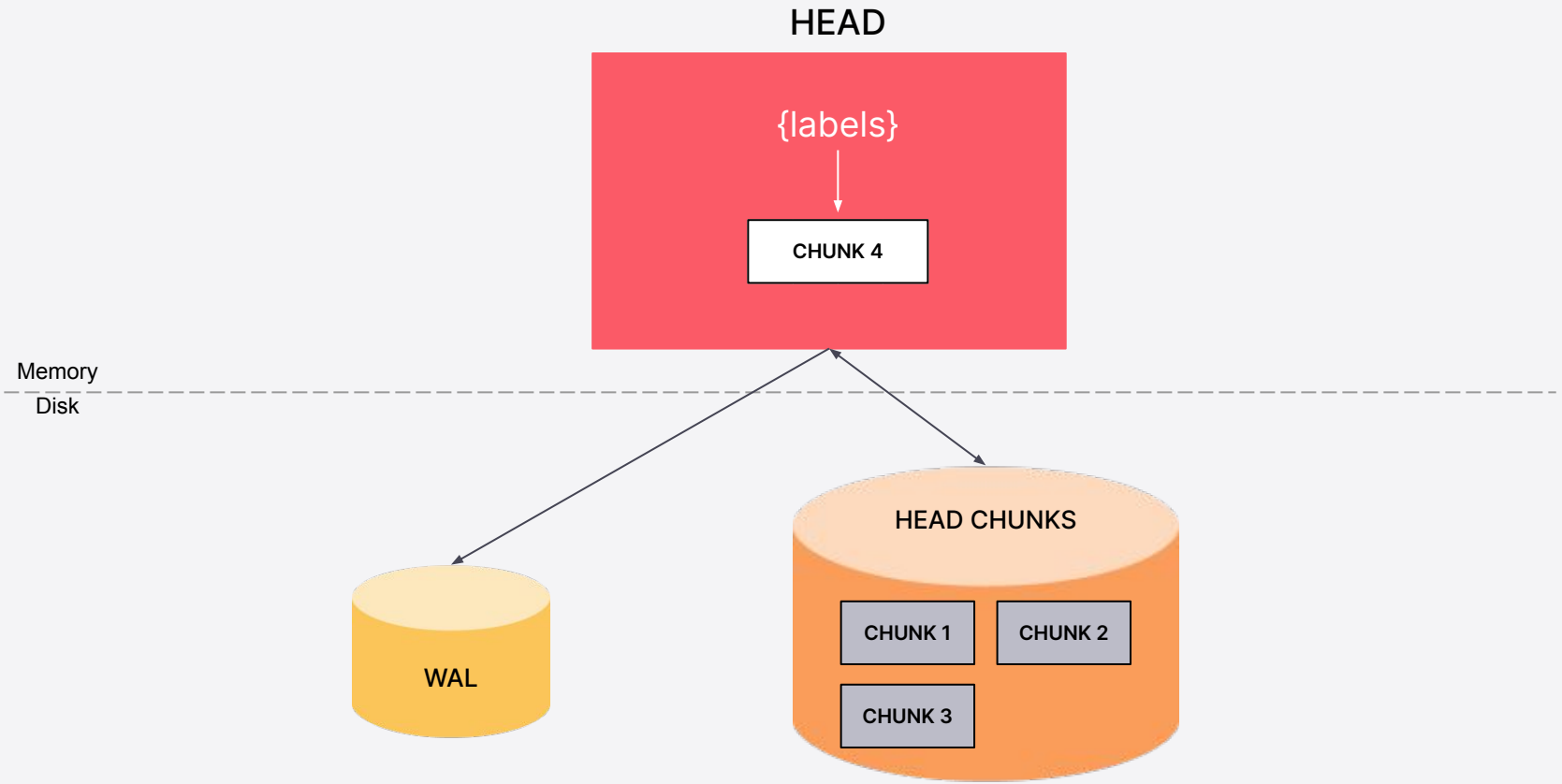
Same process repeated



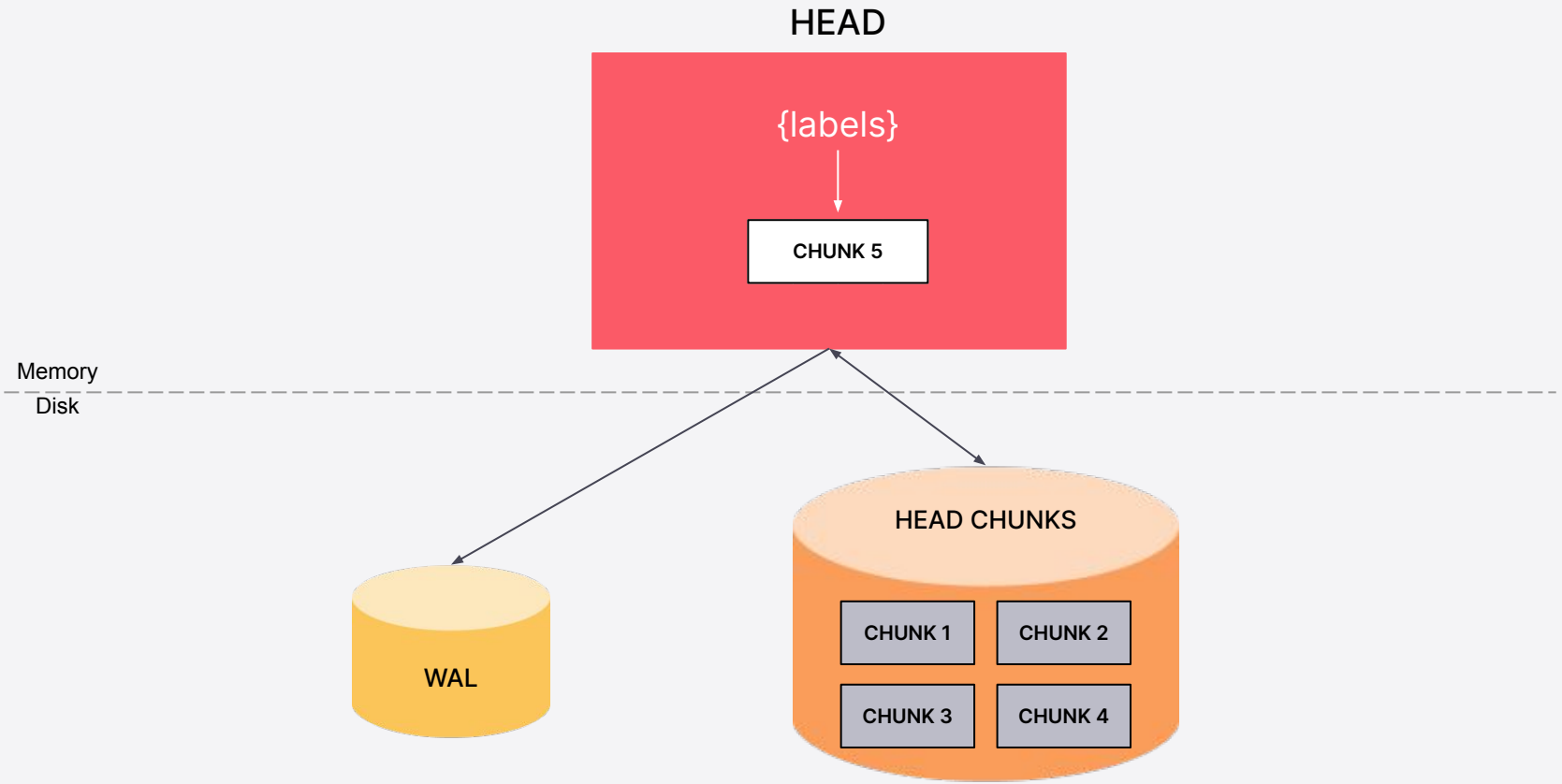
Same process repeated



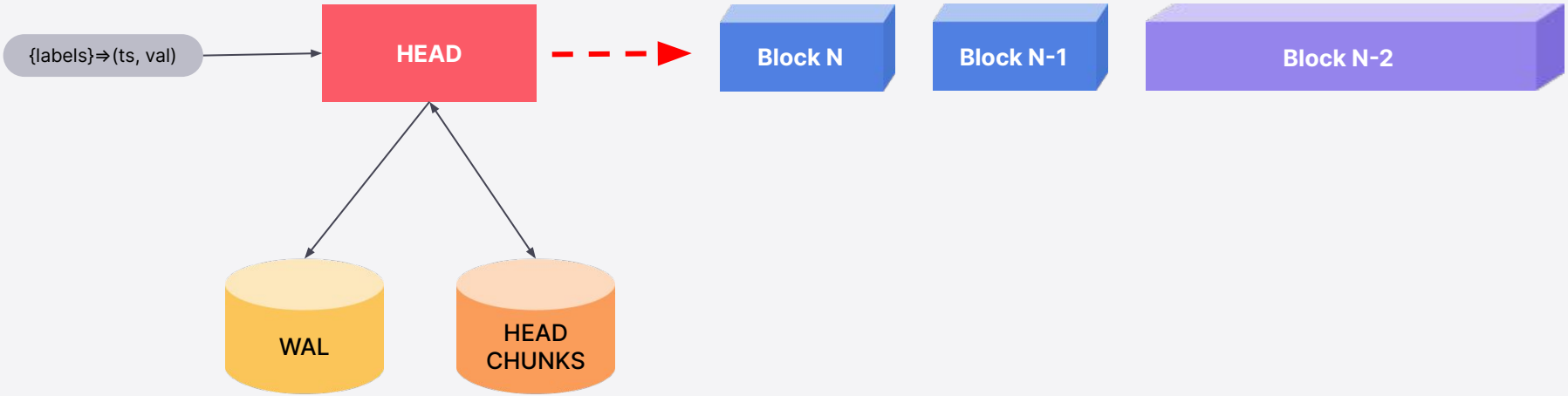
Same process repeated



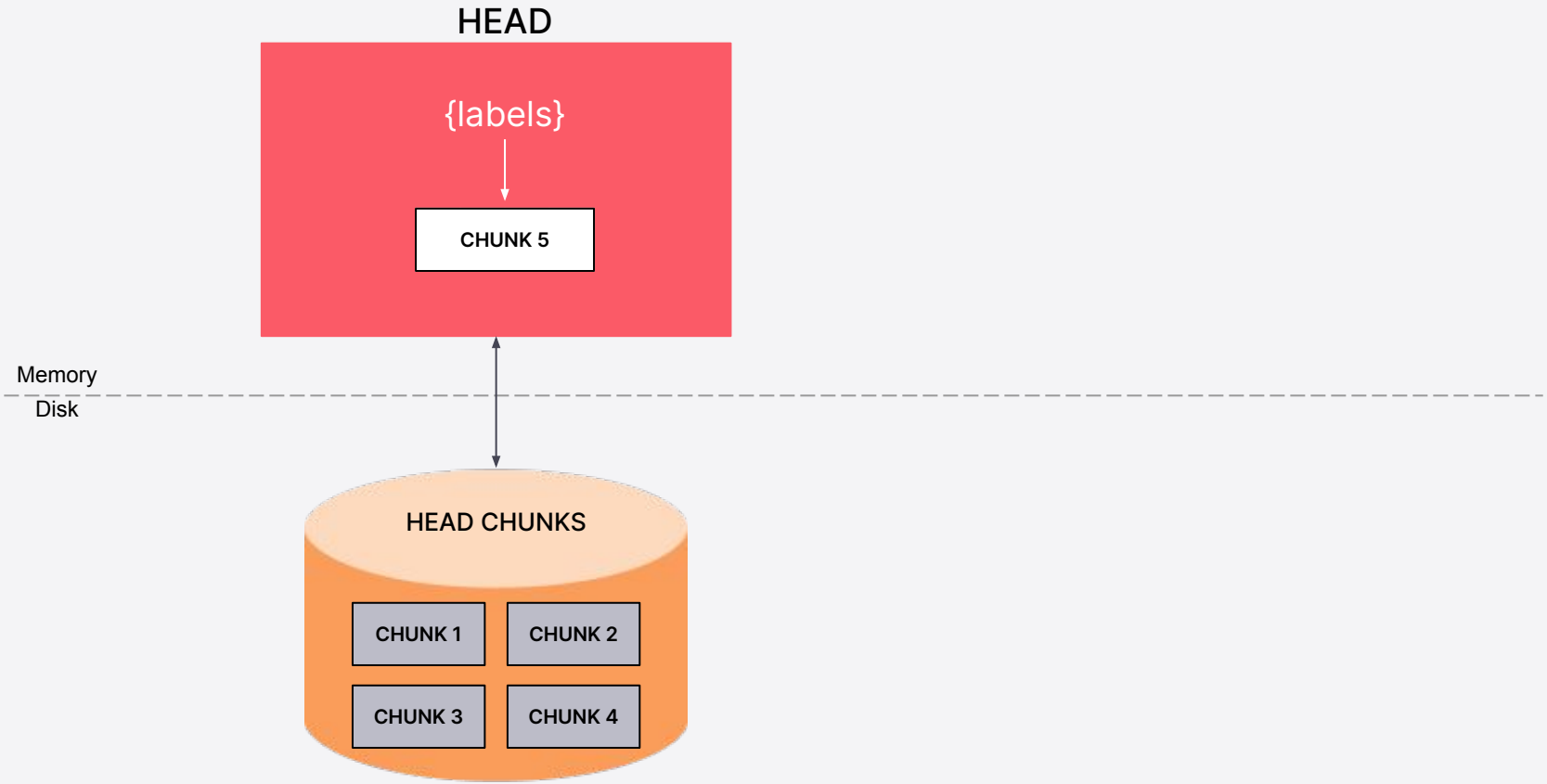
Same process repeated



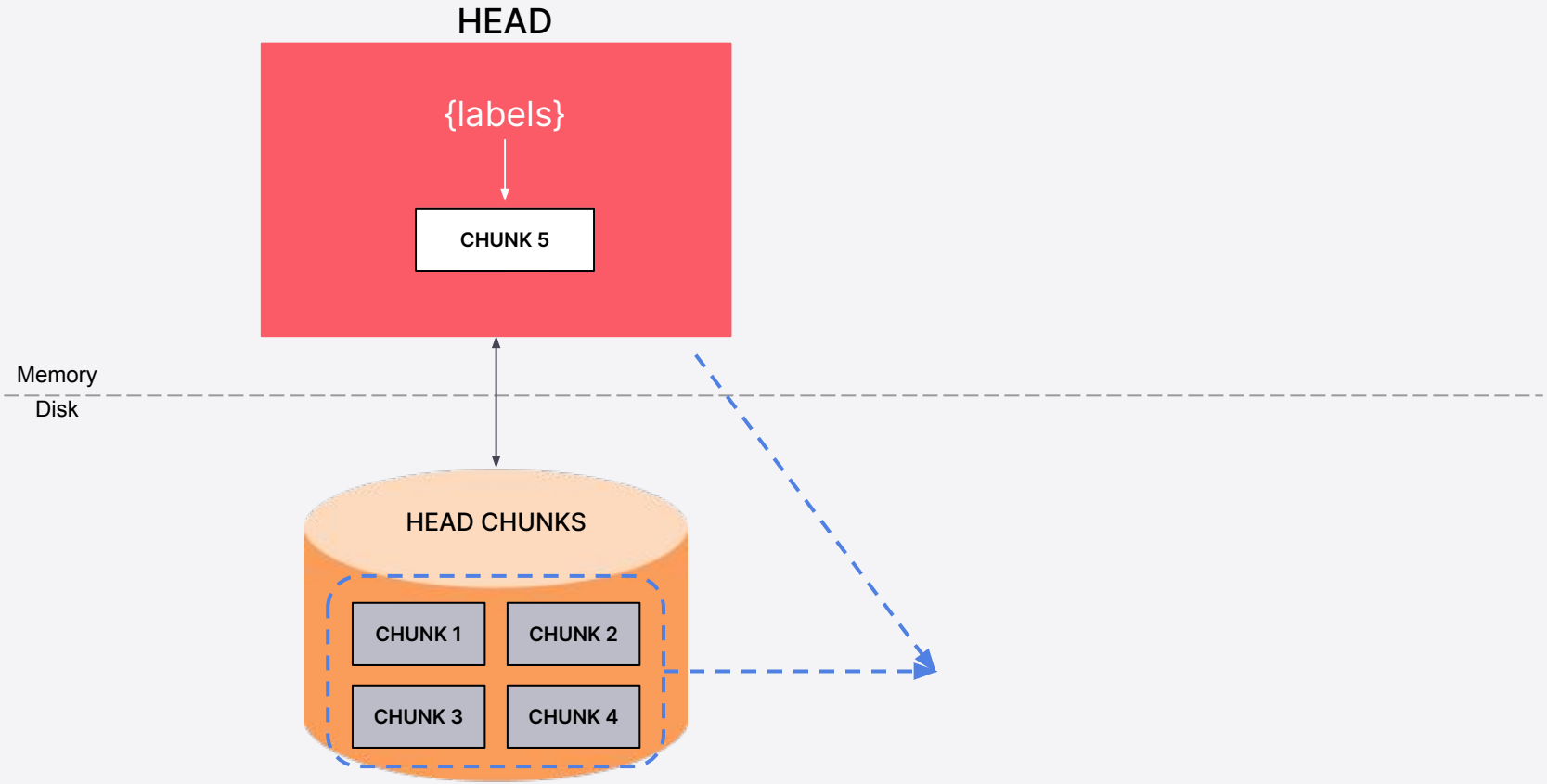
Now we create the Block



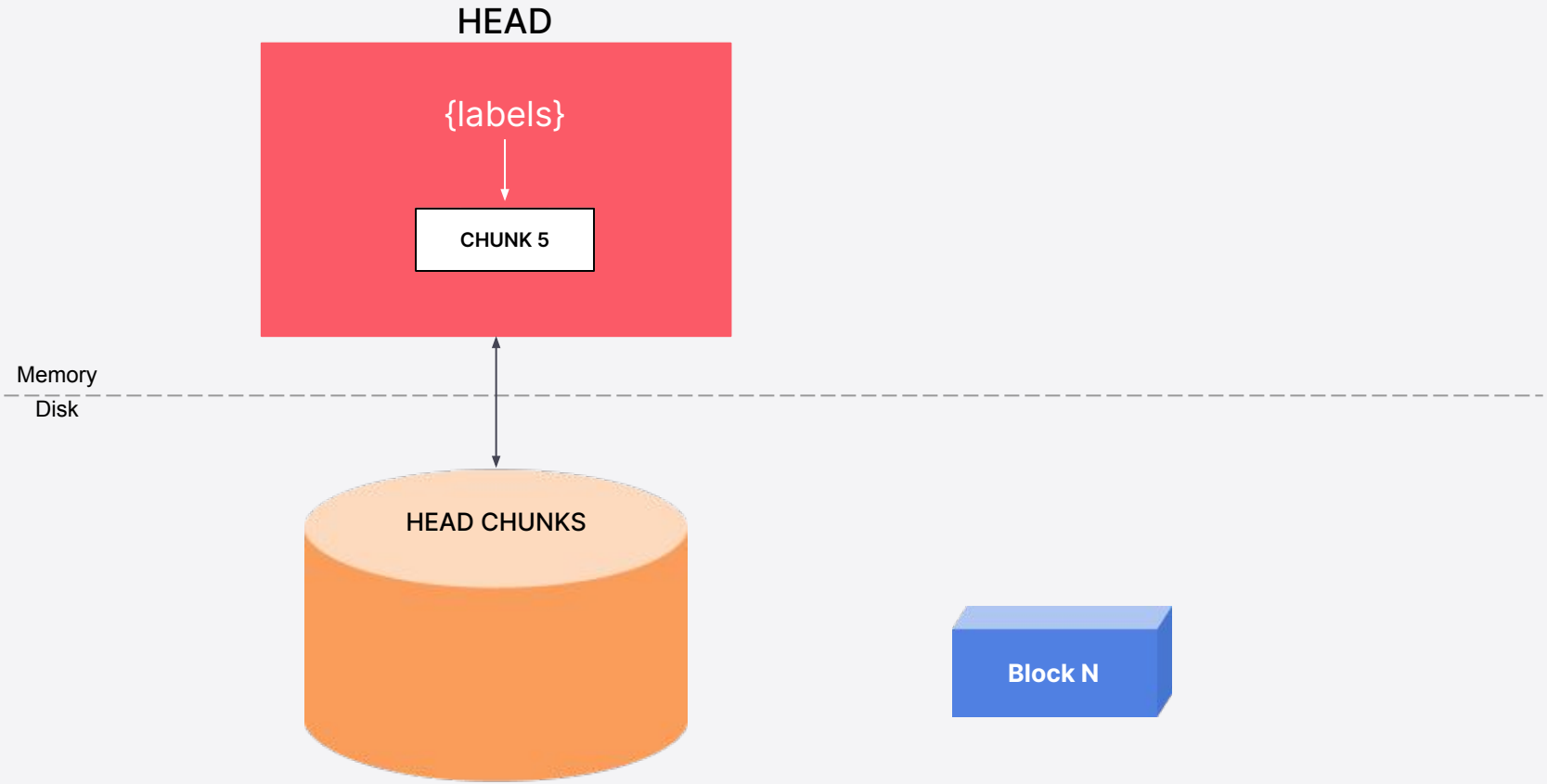
Creating Block from the HEAD - “Head Compaction”



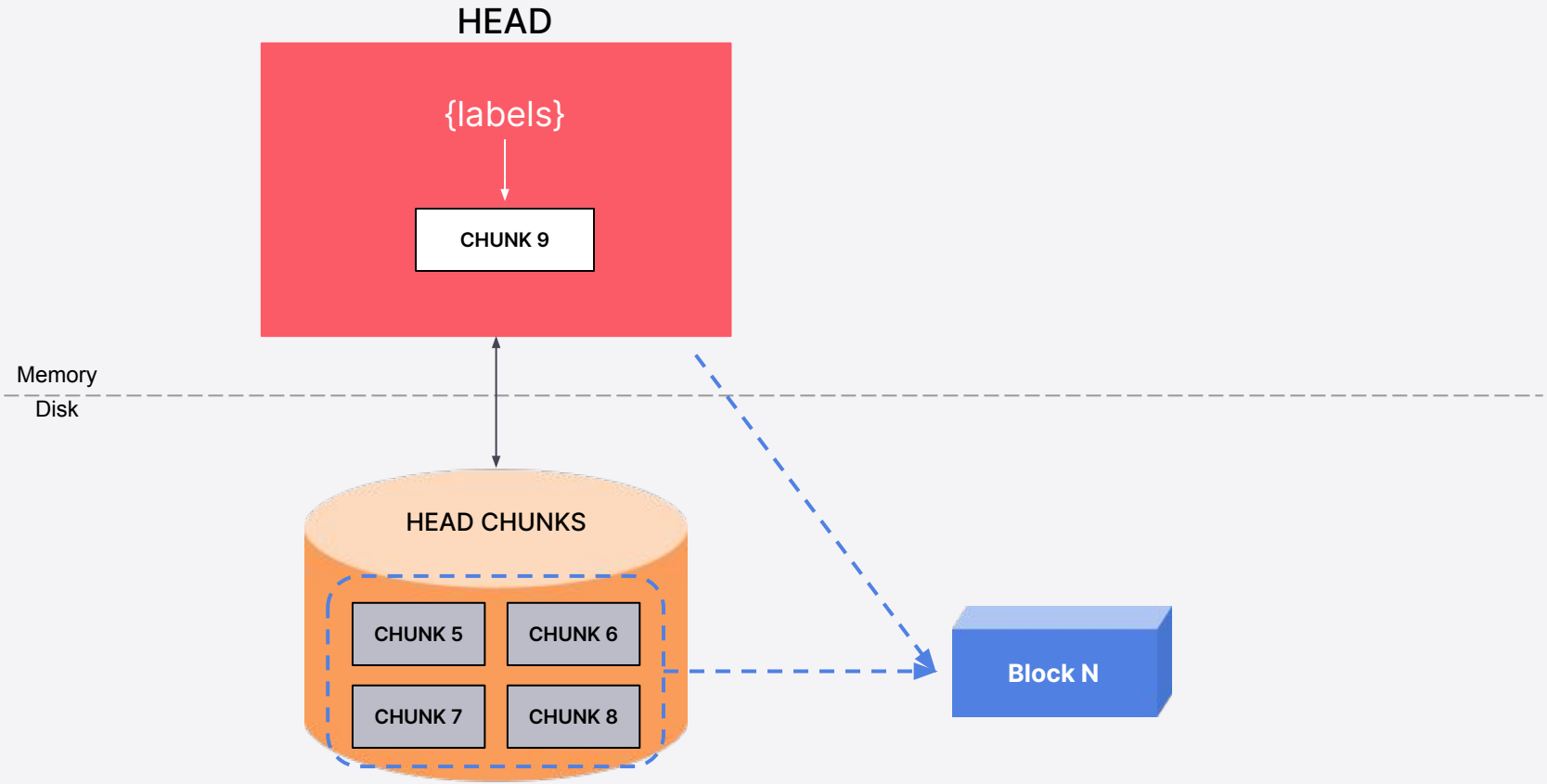
Creating Block from the HEAD - “Head Compaction”



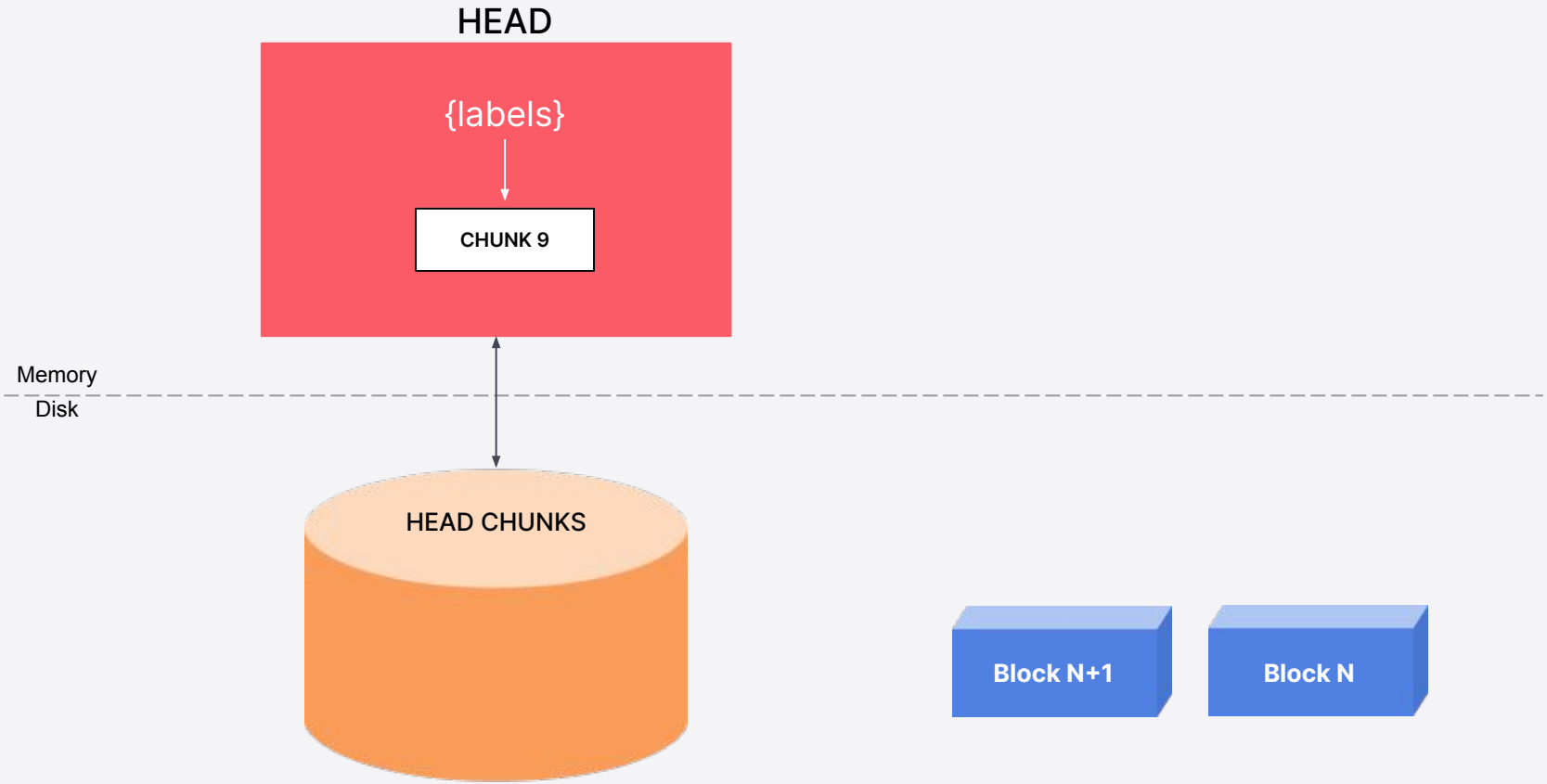
Creating Block from the HEAD - “Head Compaction”



Same process repeated



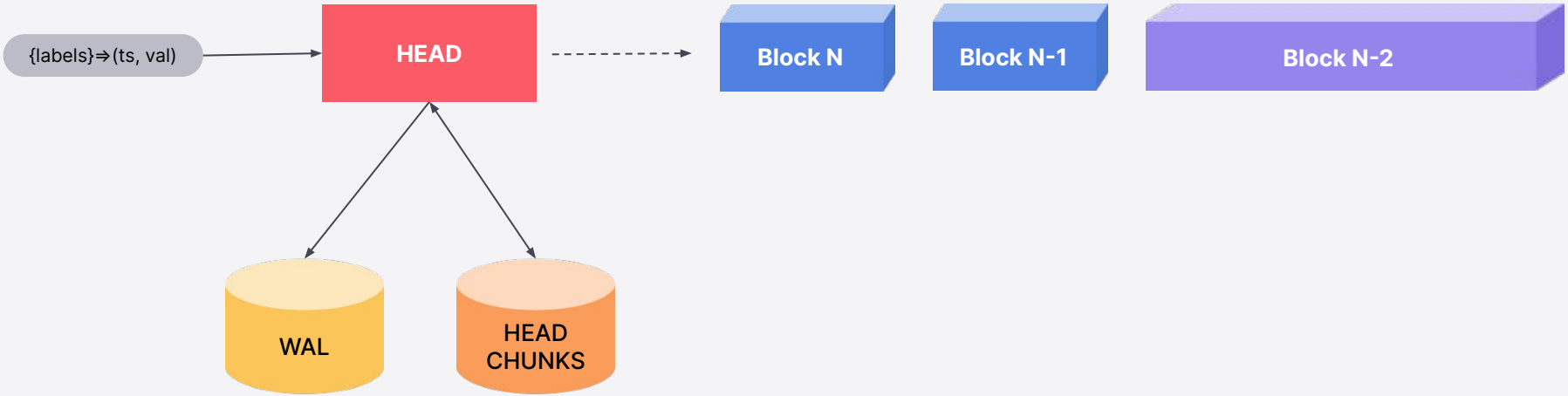
Same process repeated



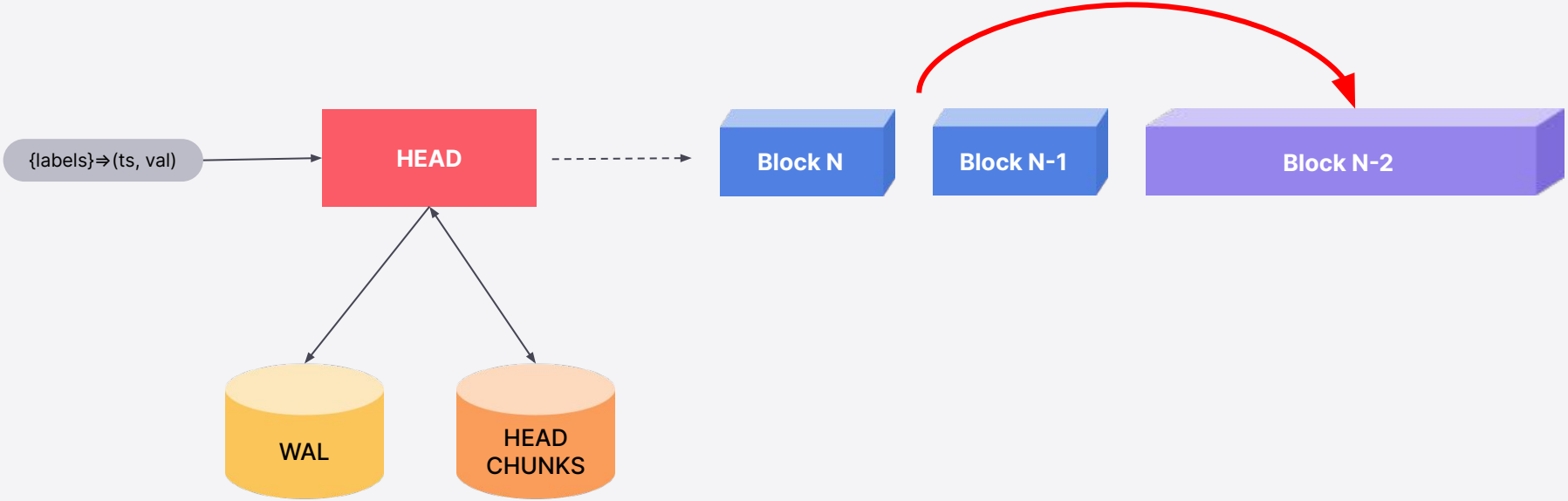
Why do head compaction?

Remove stale series
Clear up memory

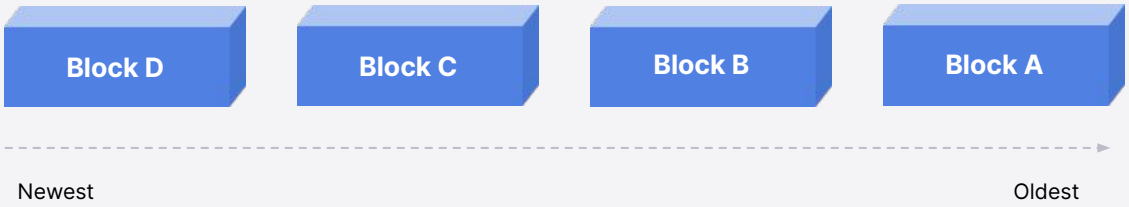
What about that bigger block?



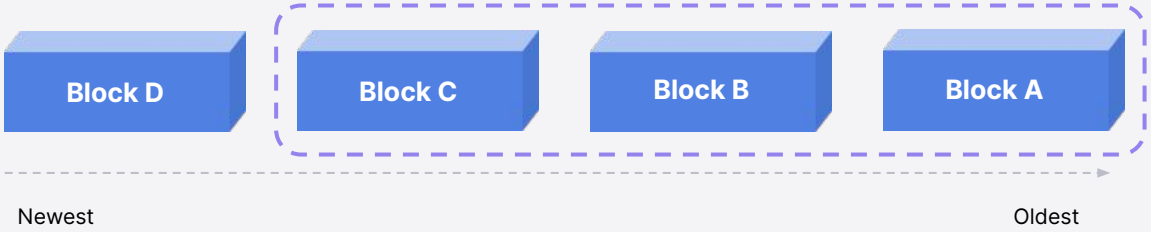
New bigger block is created from smaller blocks



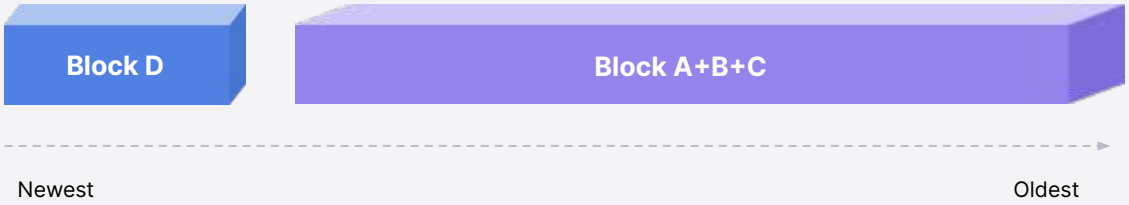
“Block Compaction” or just Compaction



“Block Compaction” or just Compaction



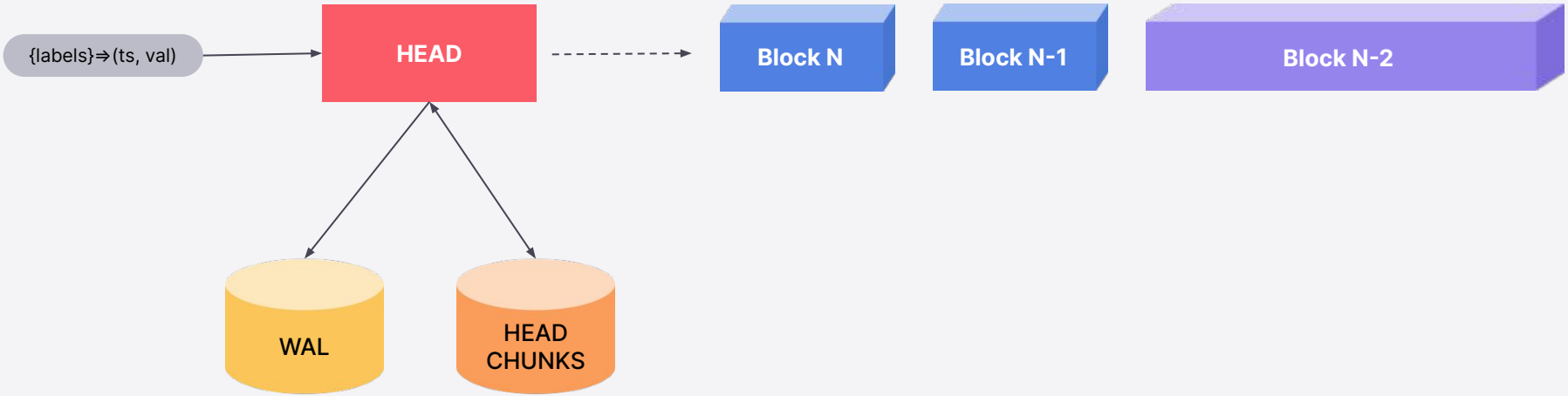
“Block Compaction” or just Compaction



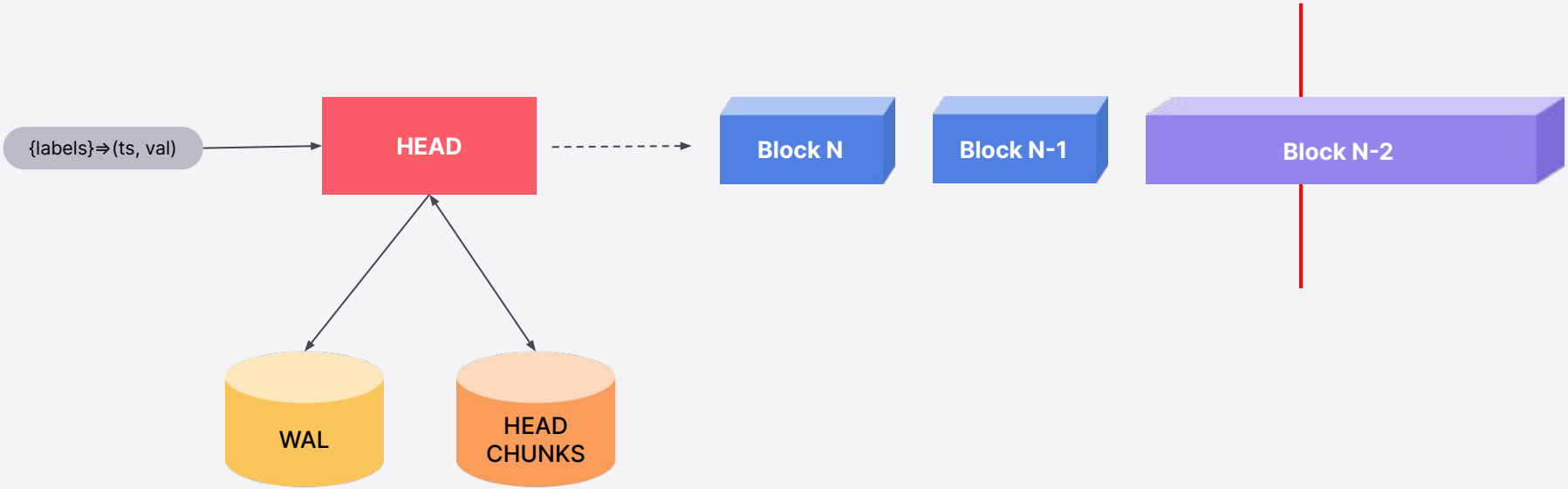
Why do block compaction?

Efficient queries
We will see how...soon

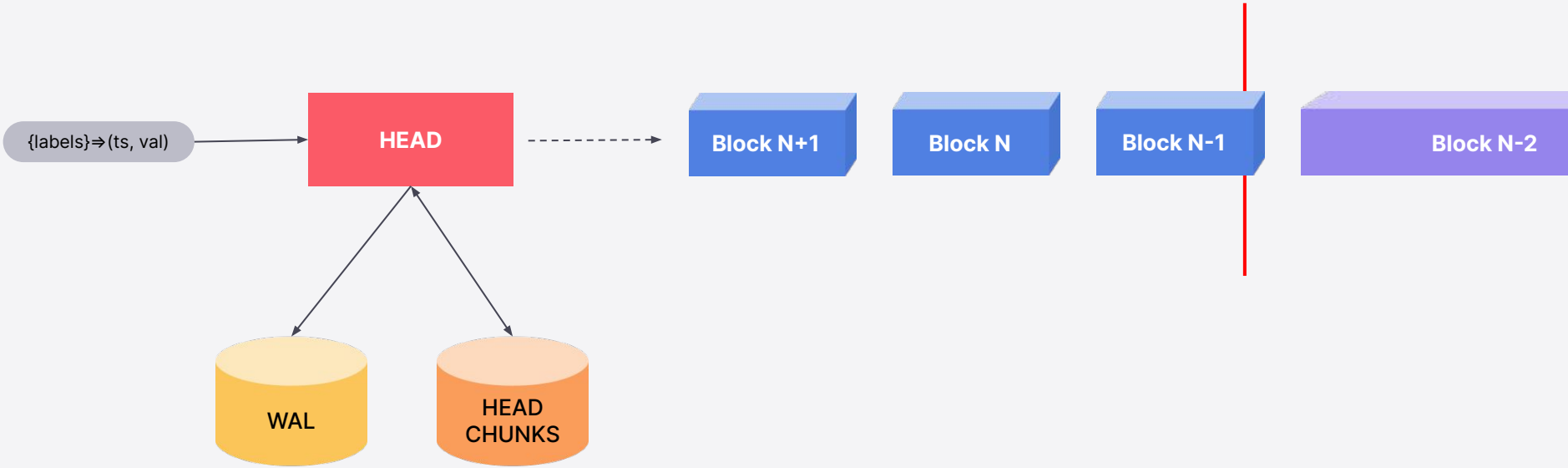
Now we need to get rid of old data



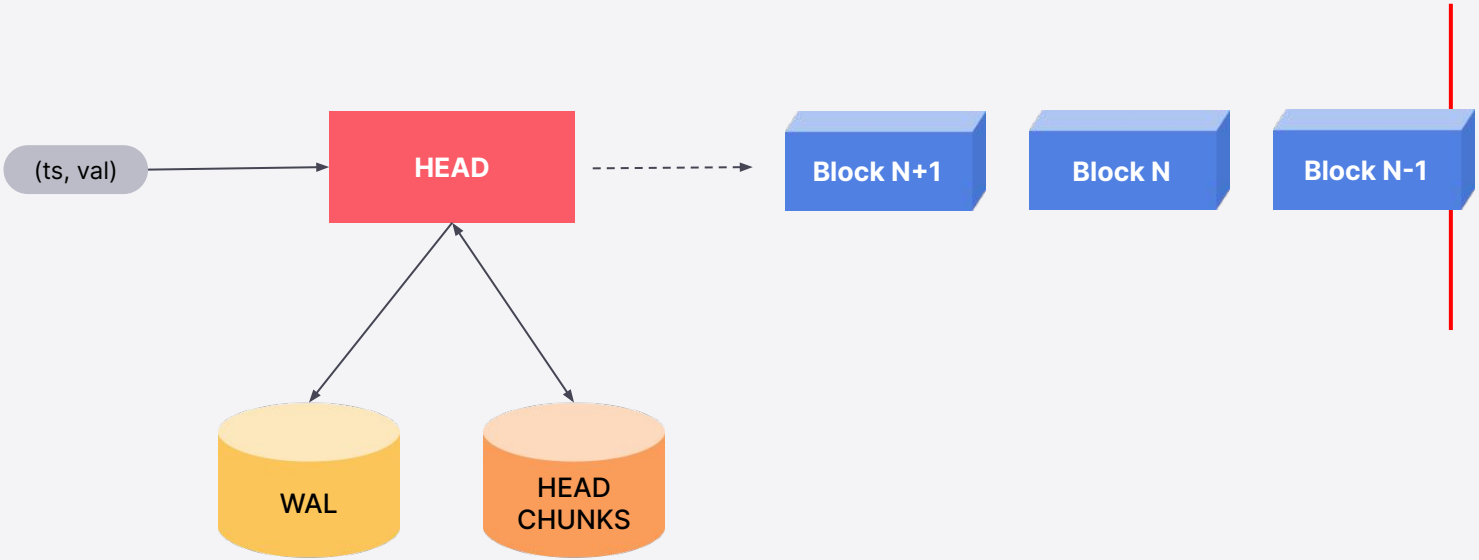
Example of time based retention



Example of time based retention

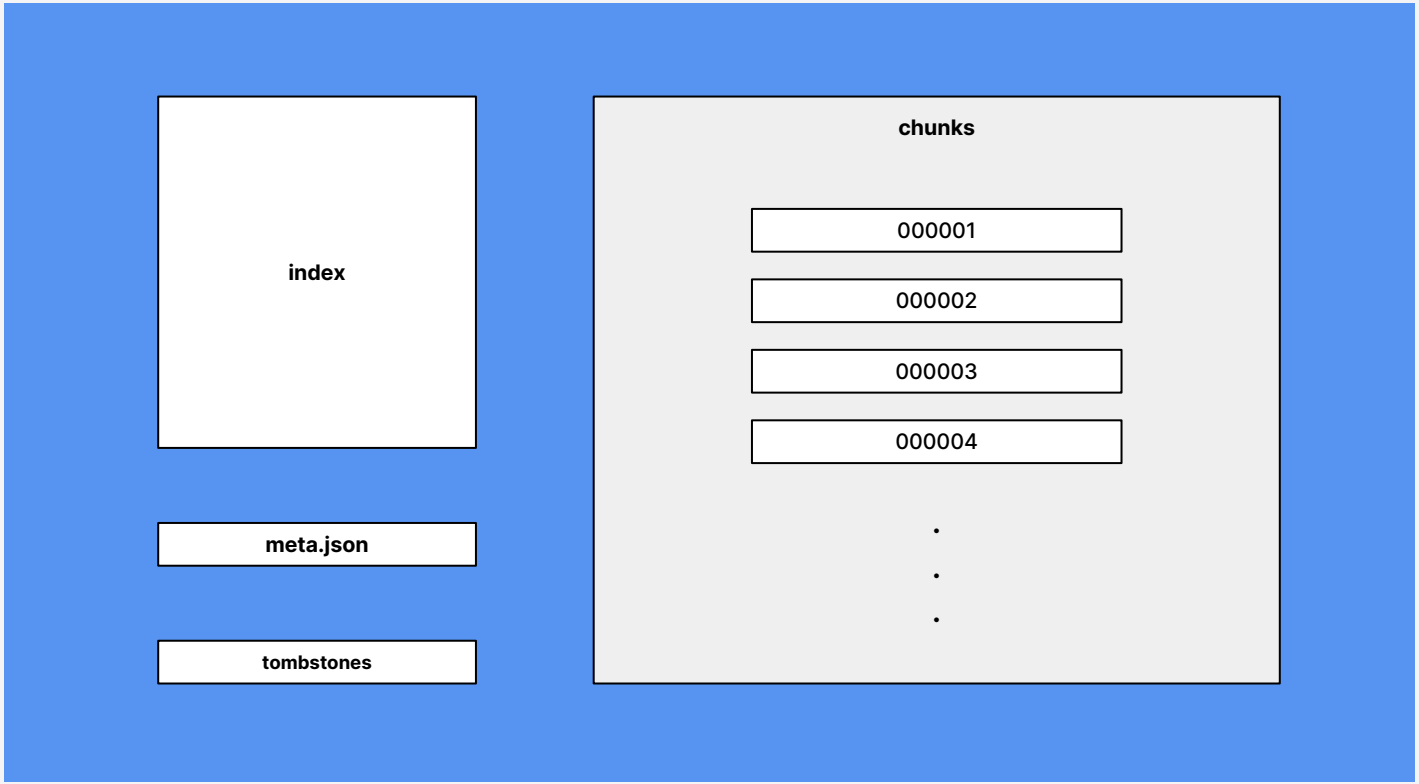


TSDB overview

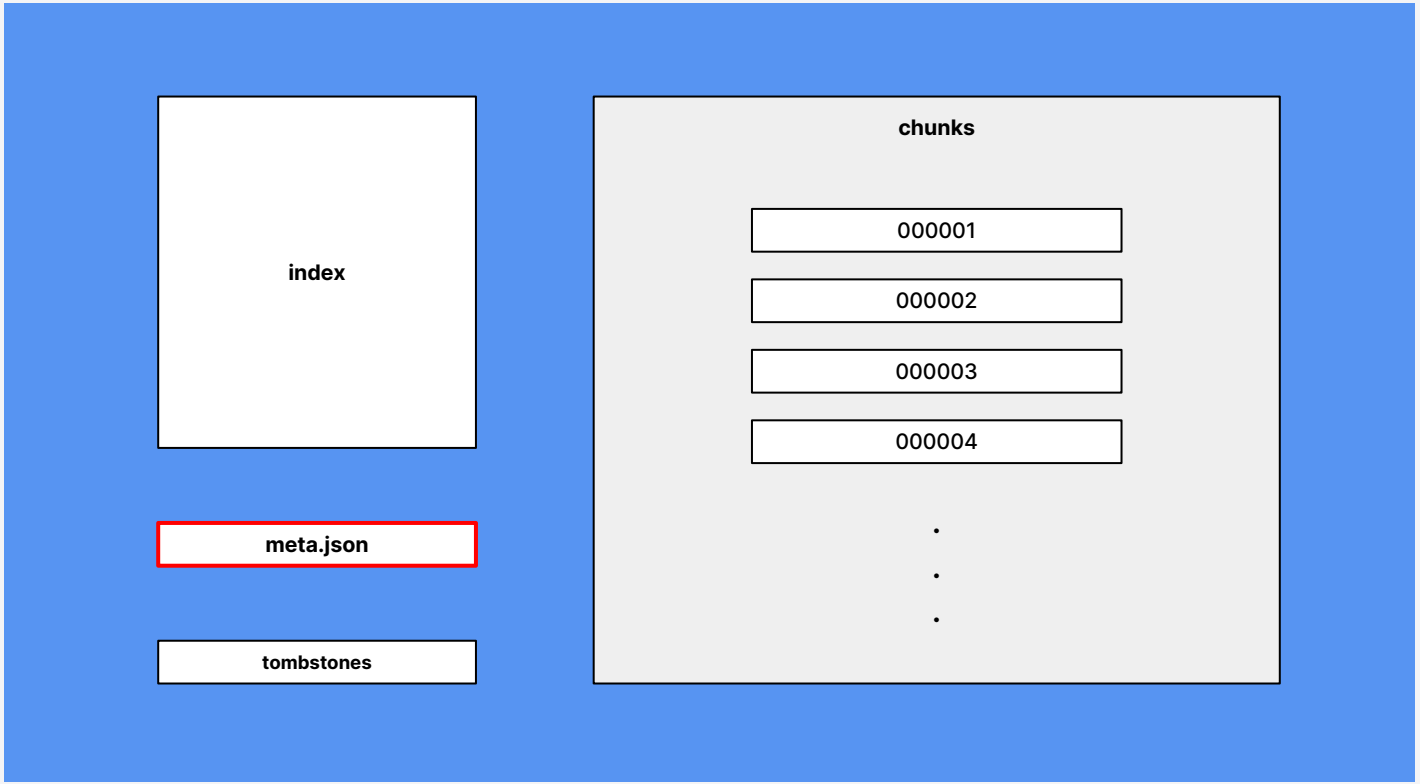


Let’s peek inside a Block

Block N


index
meta.json
tombstones
chunks
000001
000002
000003
000004

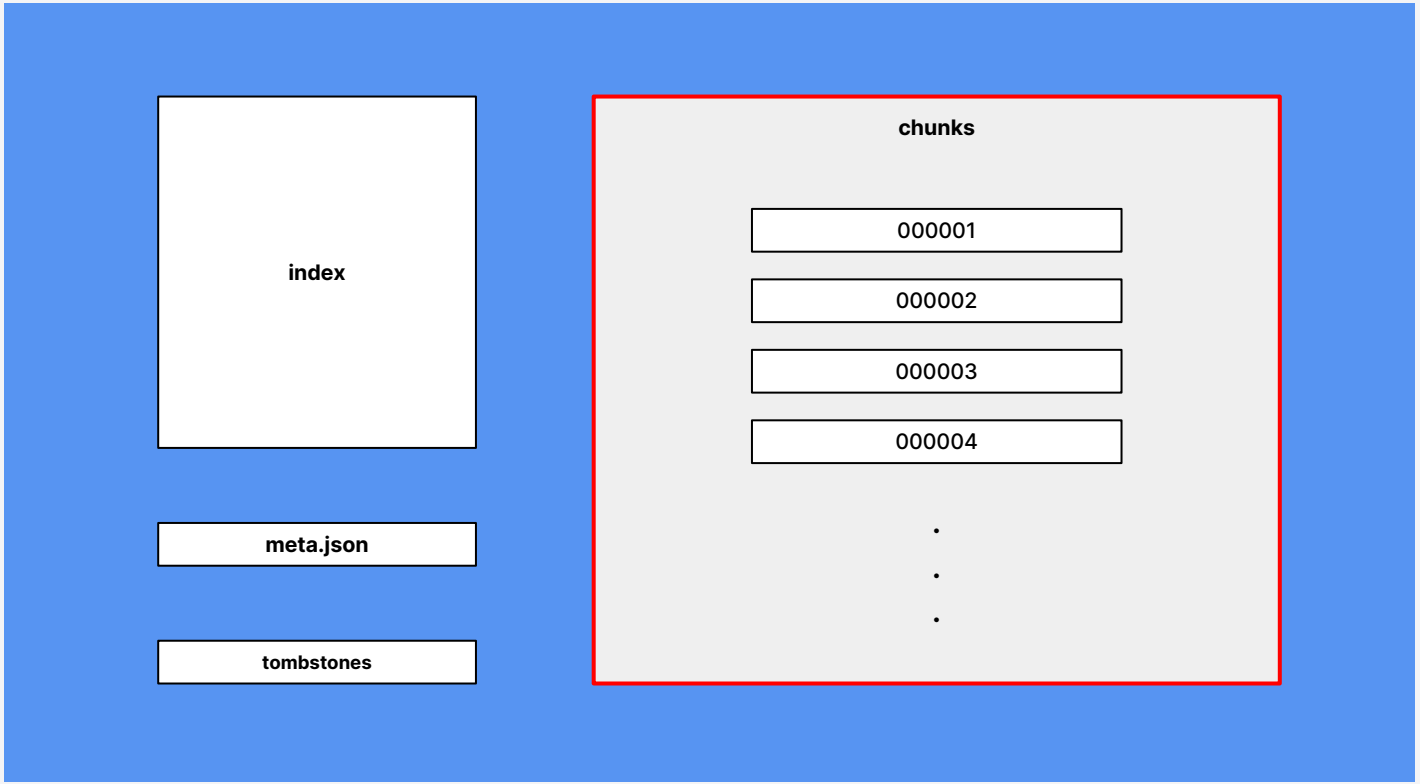
Let’s look at meta.json

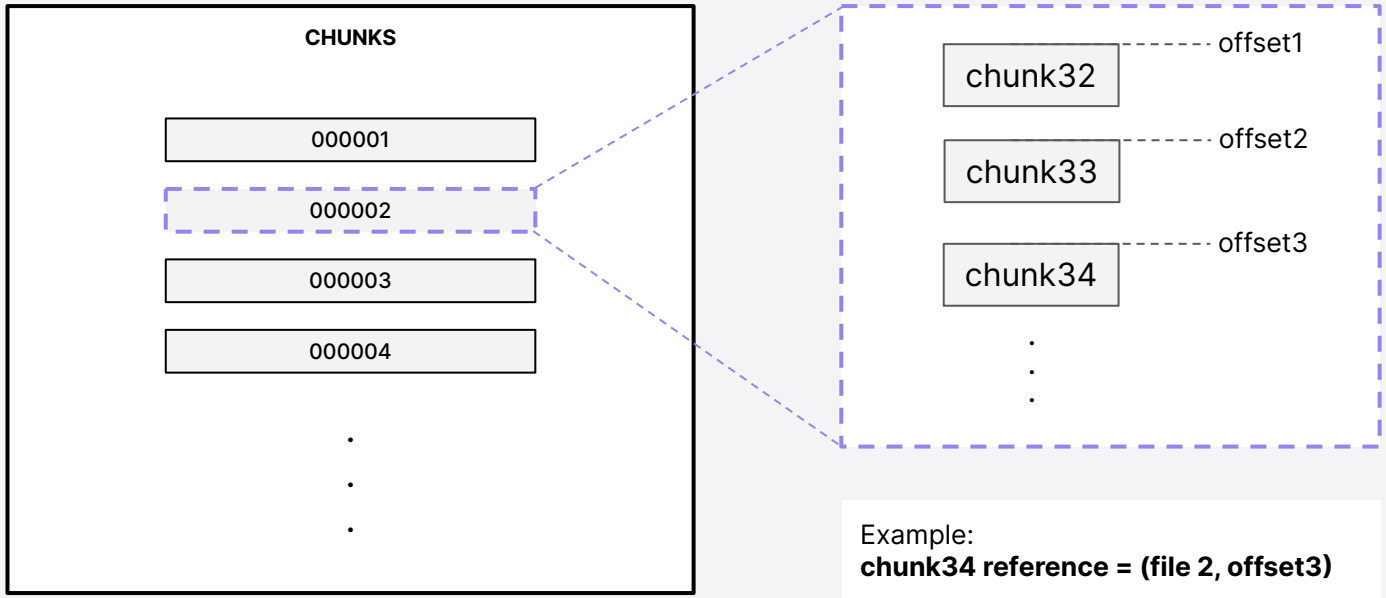


meta.json


```
{
    "ulid": "01GKGKD464MQKE2JY2875Q3D4F",
    "minTime": 1602237600000,
    "maxTime": 1602244800000,
    "stats": {
        "numSamples": 553673232,
        "numSeries":  1346066,
        "numChunks":  4440437
    },
    "compaction": {
        "level": 2,
        "sources": [
            "01GKFYSYE4PVXAHSHQREDN43N0",
            "01GKG5NNP46DBX3XBQT9S4YJ94",
            "01GKGCHCY4AV0QKB0EHYYAT8SG"
        ]
    },
    "version": 1
}
```

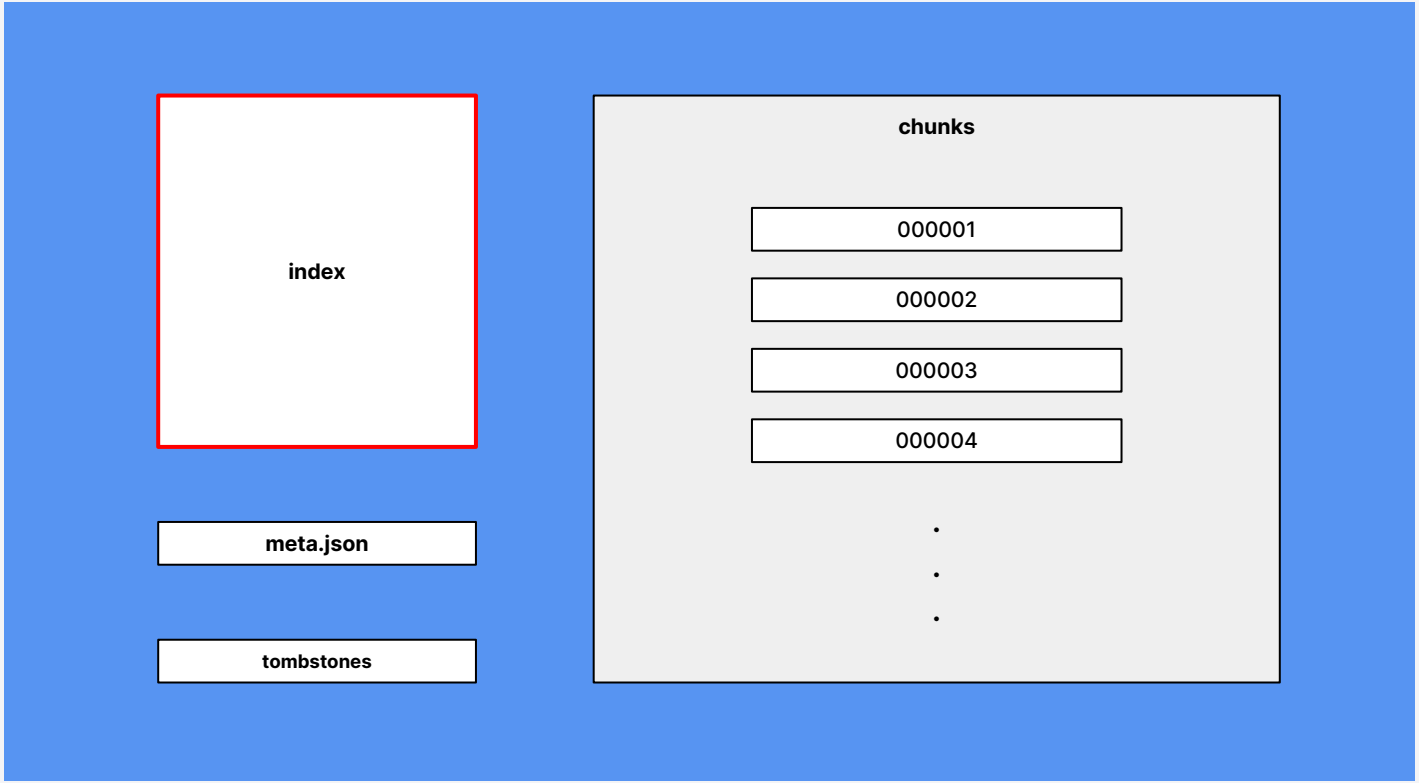

Let’s look at chunks directory

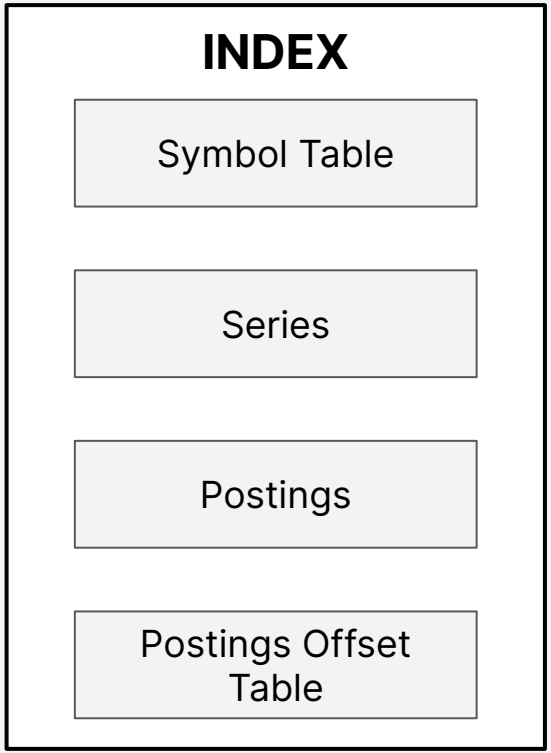


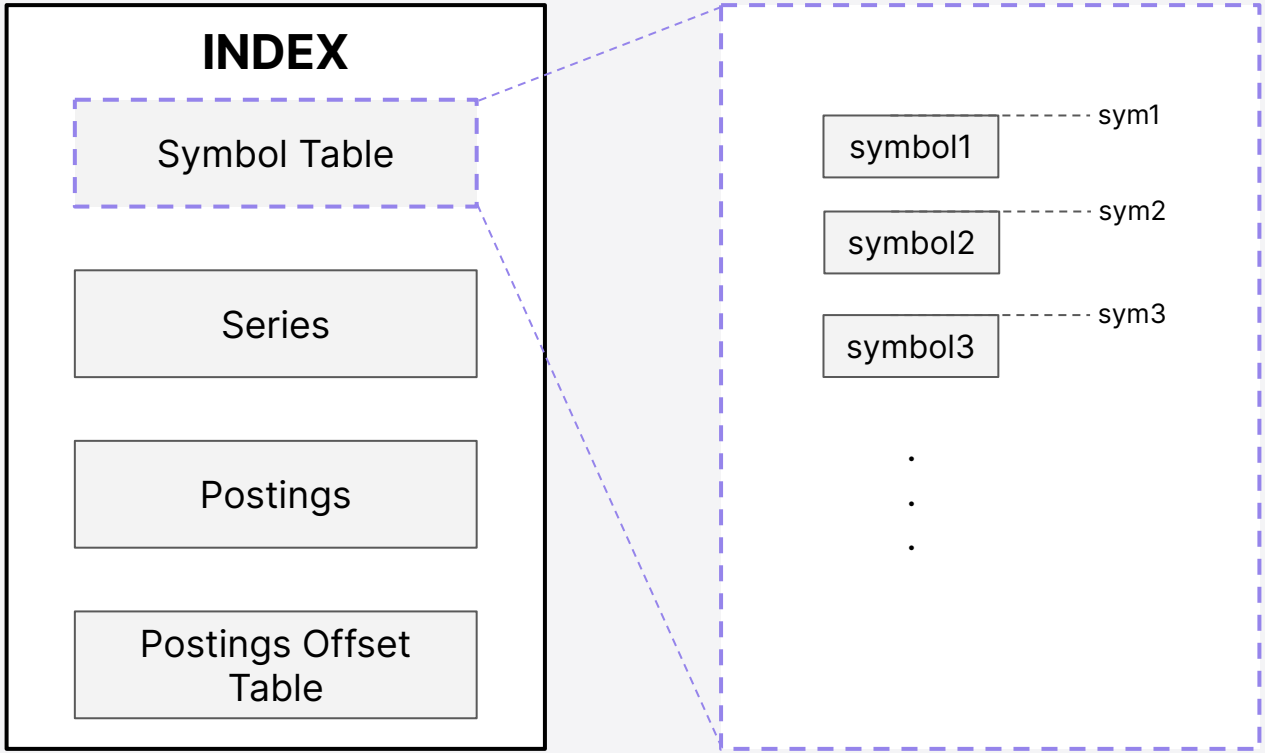


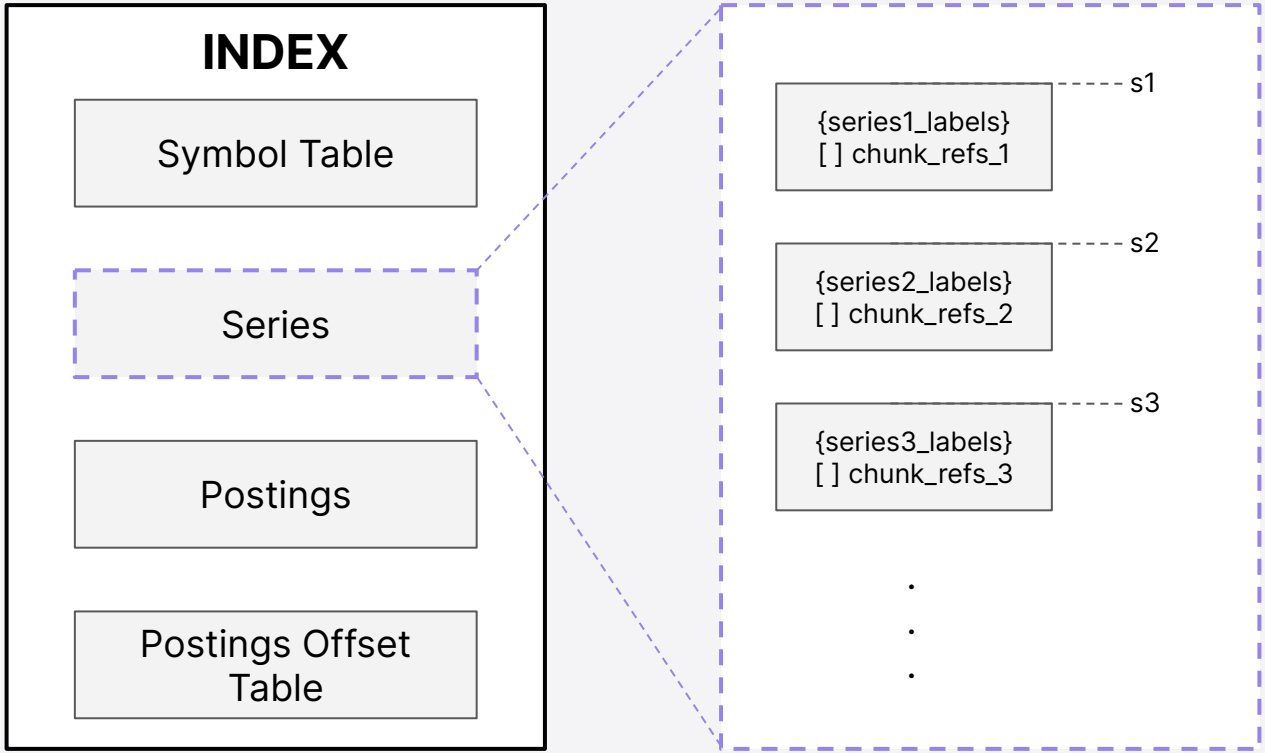
Example:
chunk34 reference = (file 2, offset3)

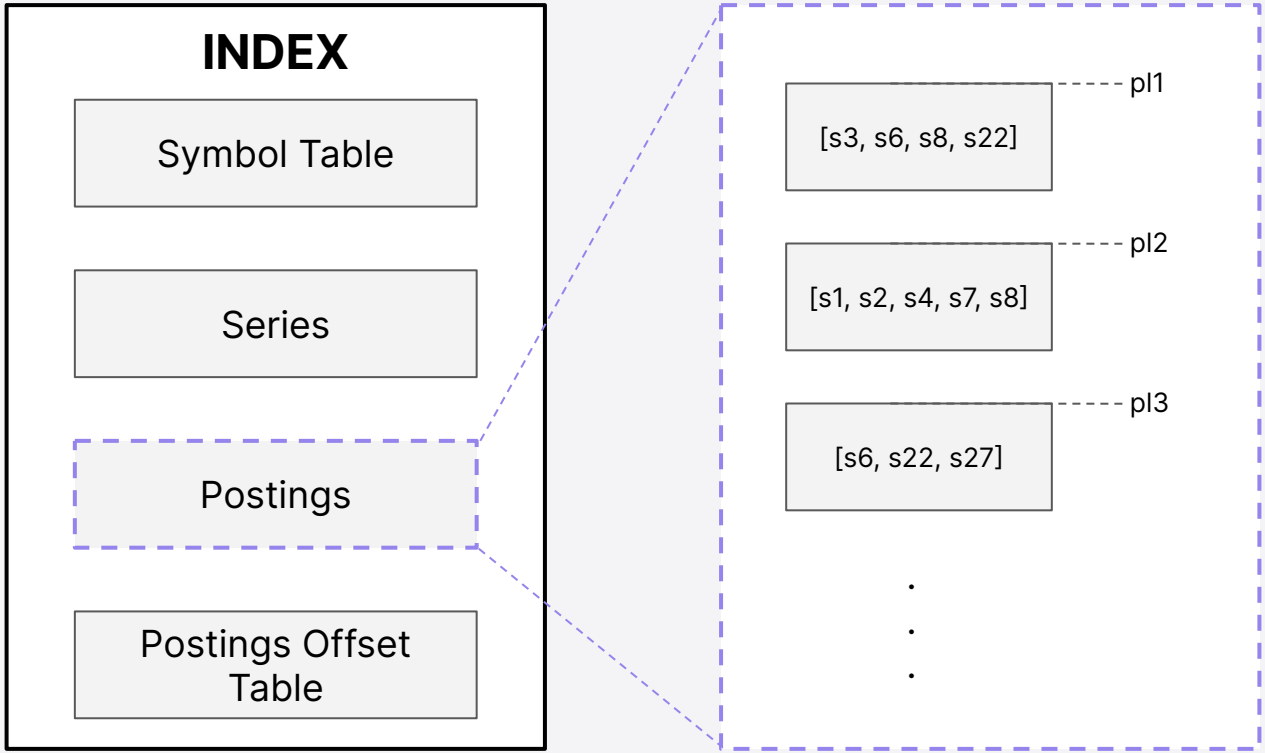
Let’s look at the index

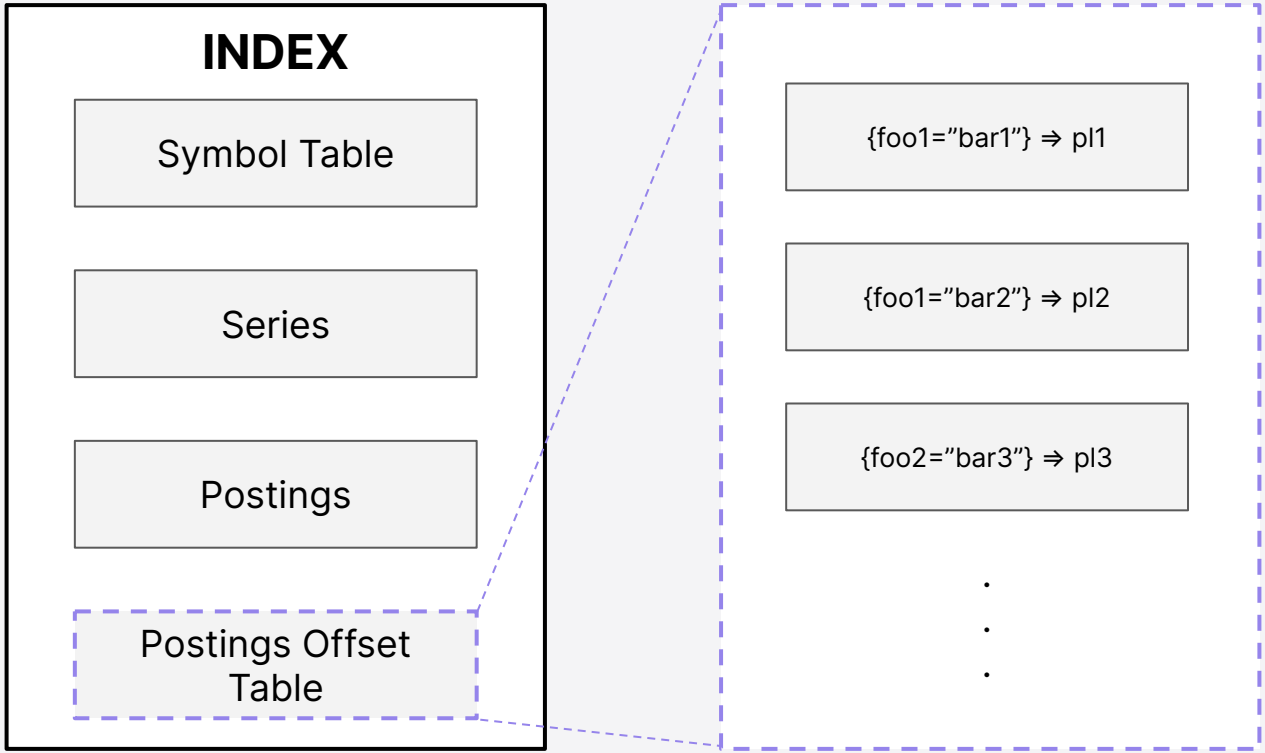



INDEX
Symbol Table
Series
Postings
Postings Offset Table


INDEX
Symbol Table
Series
Postings
Postings Offset Table
symbol1
sym1
symbol2
sym2
symbol3
sym3

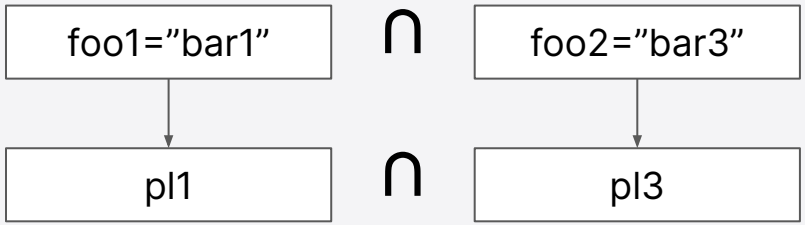

INDEX
Symbol Table
Series
Postings
Postings Offset Table
{series1_labels}
[ ] chunk_refs_1
s1
{series2_labels}
[ ] chunk_refs_2
s2
{series3_labels}
[ ] chunk_refs_3
s3


INDEX
Symbol Table
Series
Postings
Postings Offset Table
pl1
[s3, s6, s8, s22]
pl2
[s1, s2, s4, s7, s8]
pl3
[s6, s22, s27]


INDEX
Symbol Table
Series
Postings
Postings Offset Table
{foo1=”bar1”} ⇒ pl1
{foo1=”bar2”} ⇒ pl2
{foo2=”bar3”} ⇒ pl3

Querying: {foo1=”bar1”, foo2=”bar3”}

Querying: {foo1=”bar1”, foo2=”bar3”}



Postings Offset Table

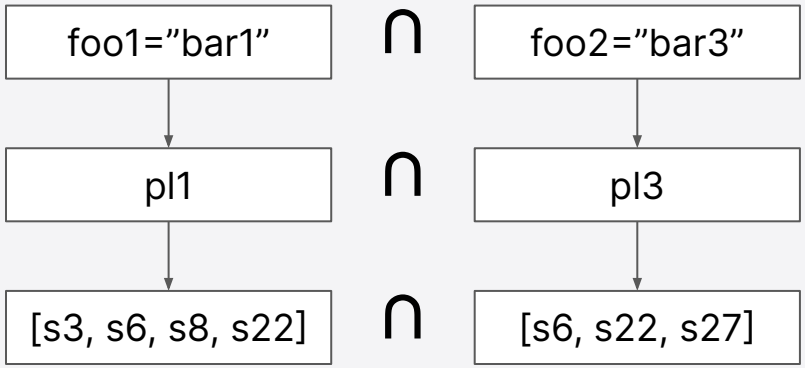
{foo1=”bar1”} ⇒ pl1

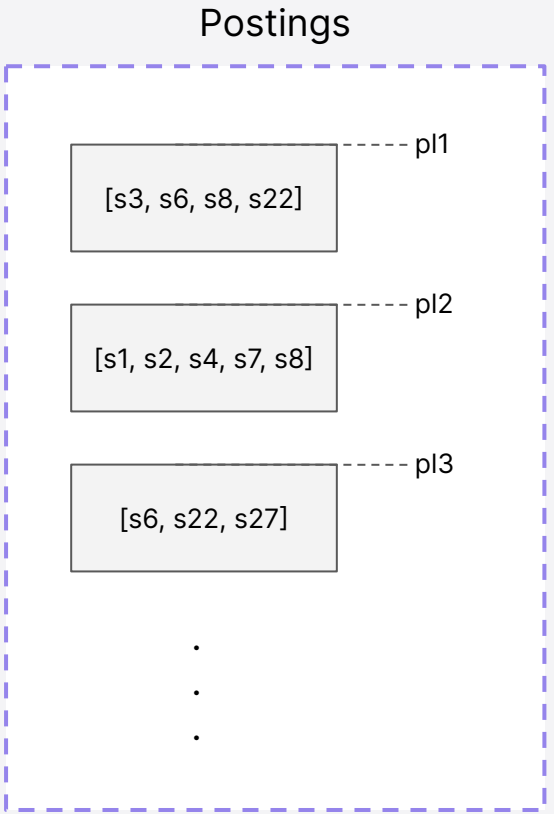
{foo1=”bar2”} ⇒ pl2

{foo2=”bar3”} ⇒ pl3

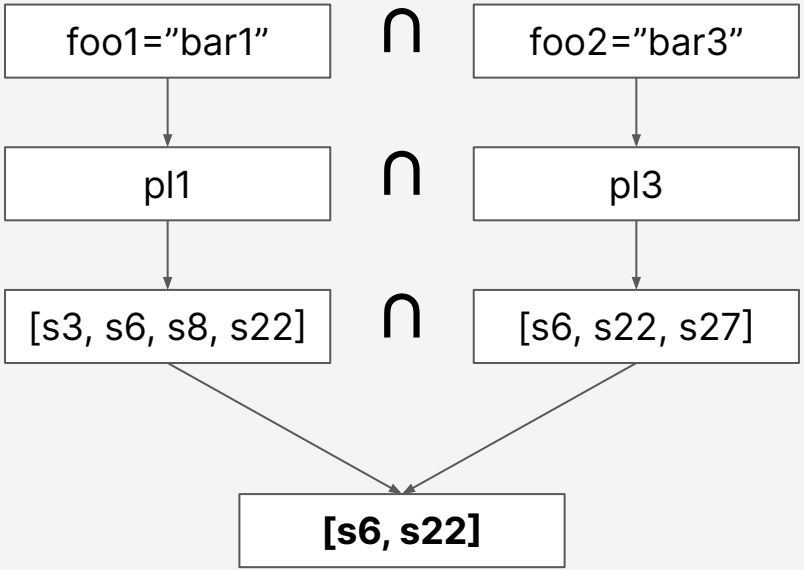
.
.
.

Querying: {foo1=”bar1”, foo2=”bar3”}

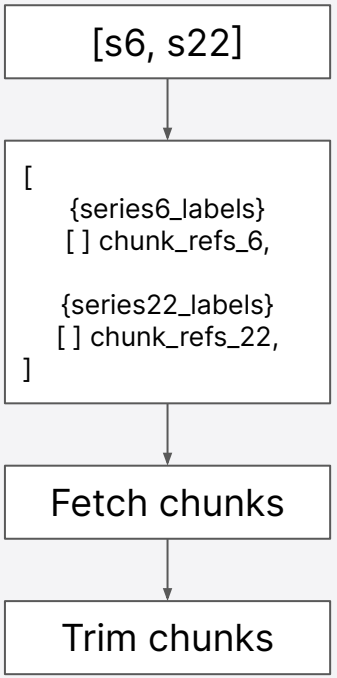




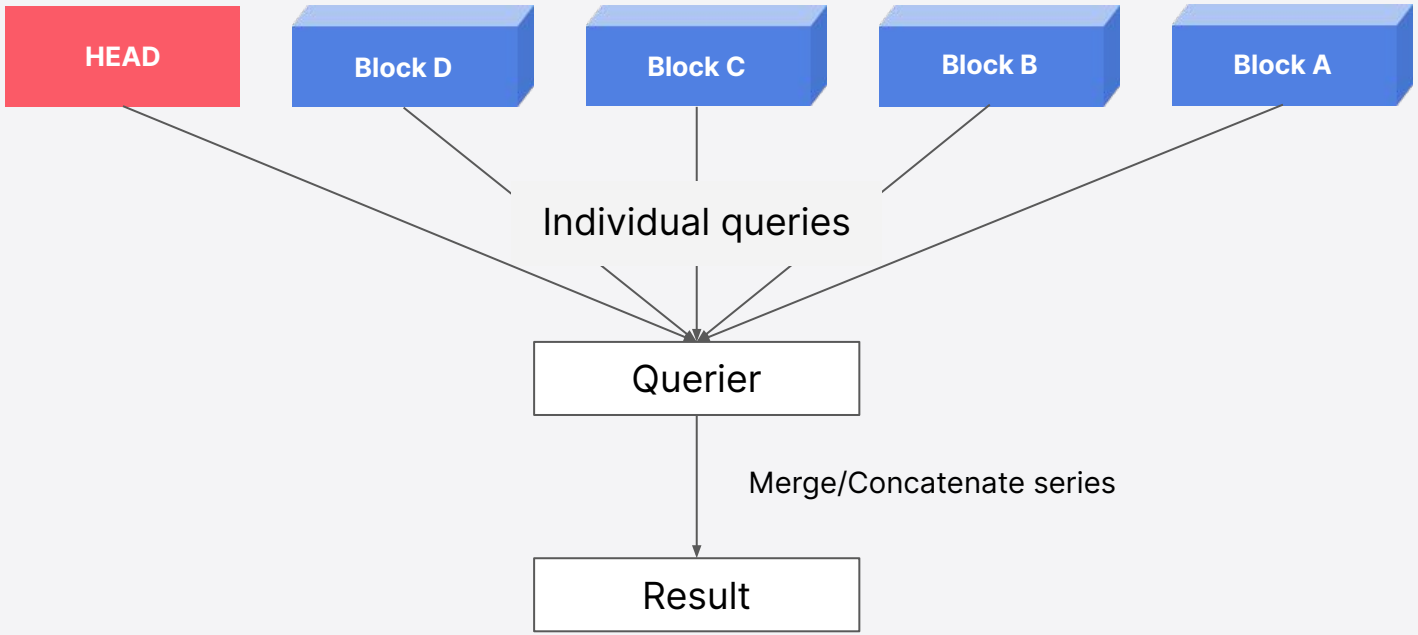
Querying: {foo1=”bar1”, foo2=”bar3”}



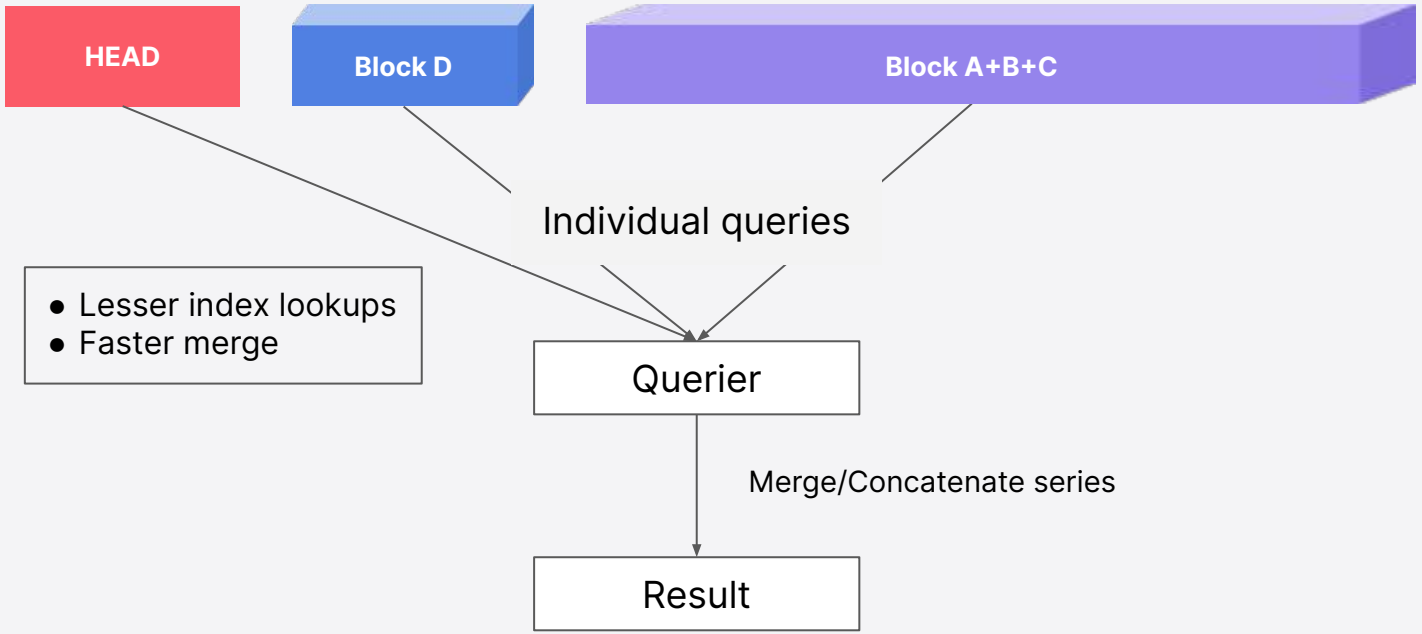
Querying: {foo1=”bar1”, foo2=”bar3”}



Querying multiple blocks



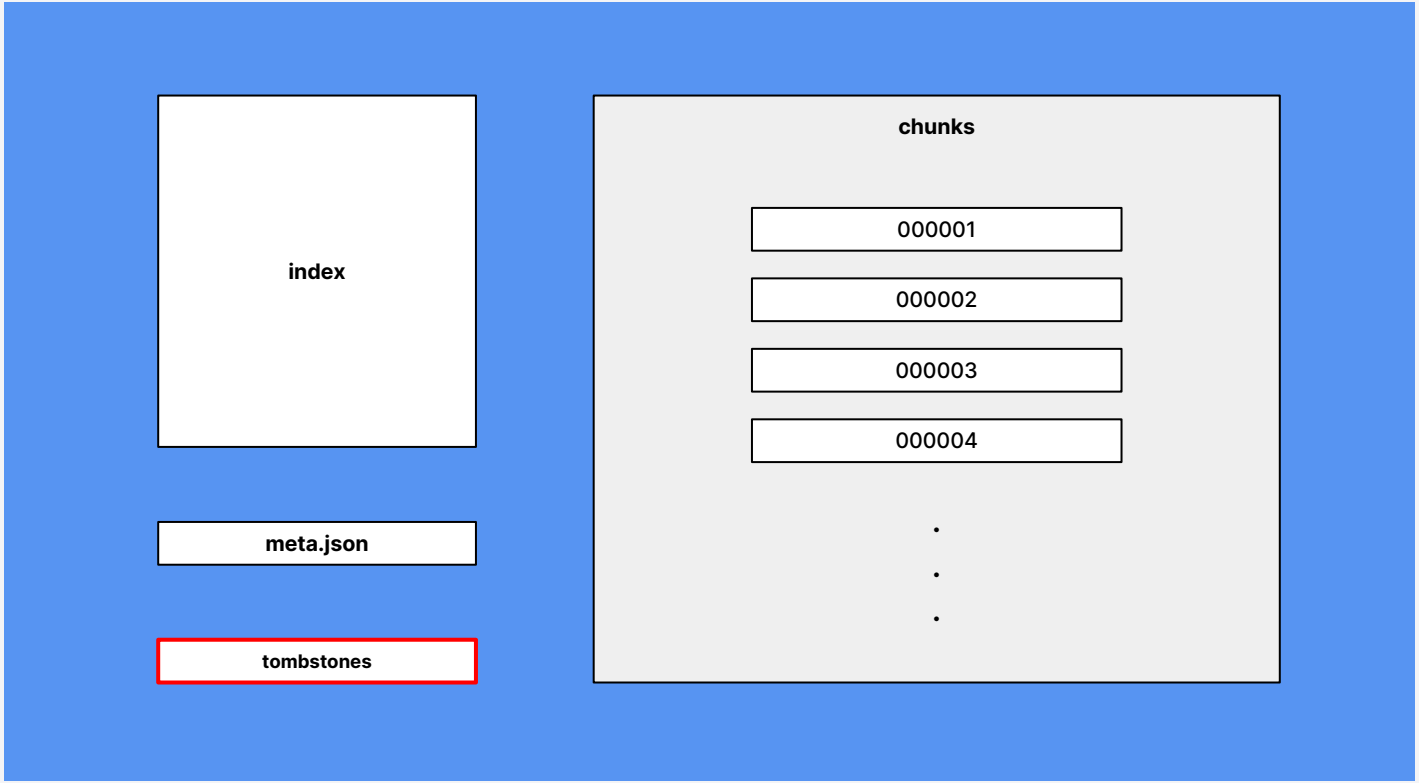
Querying multiple blocks

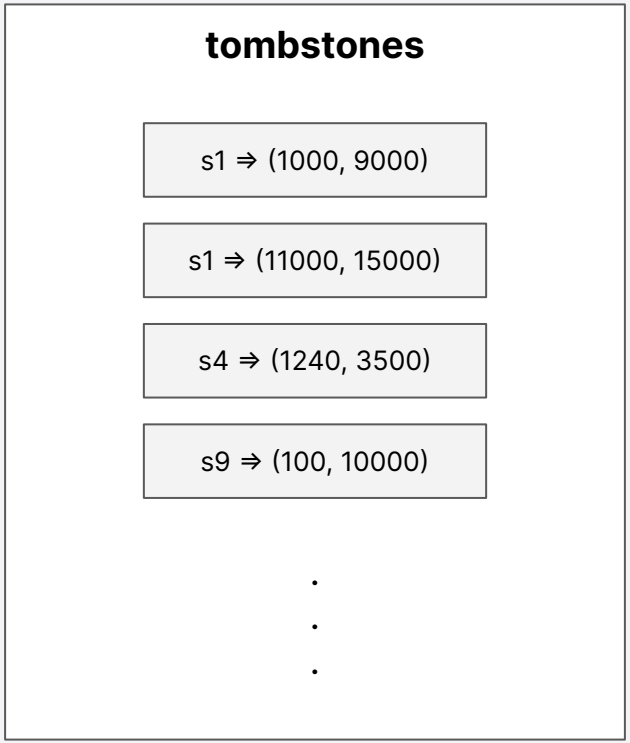


Different query matchers

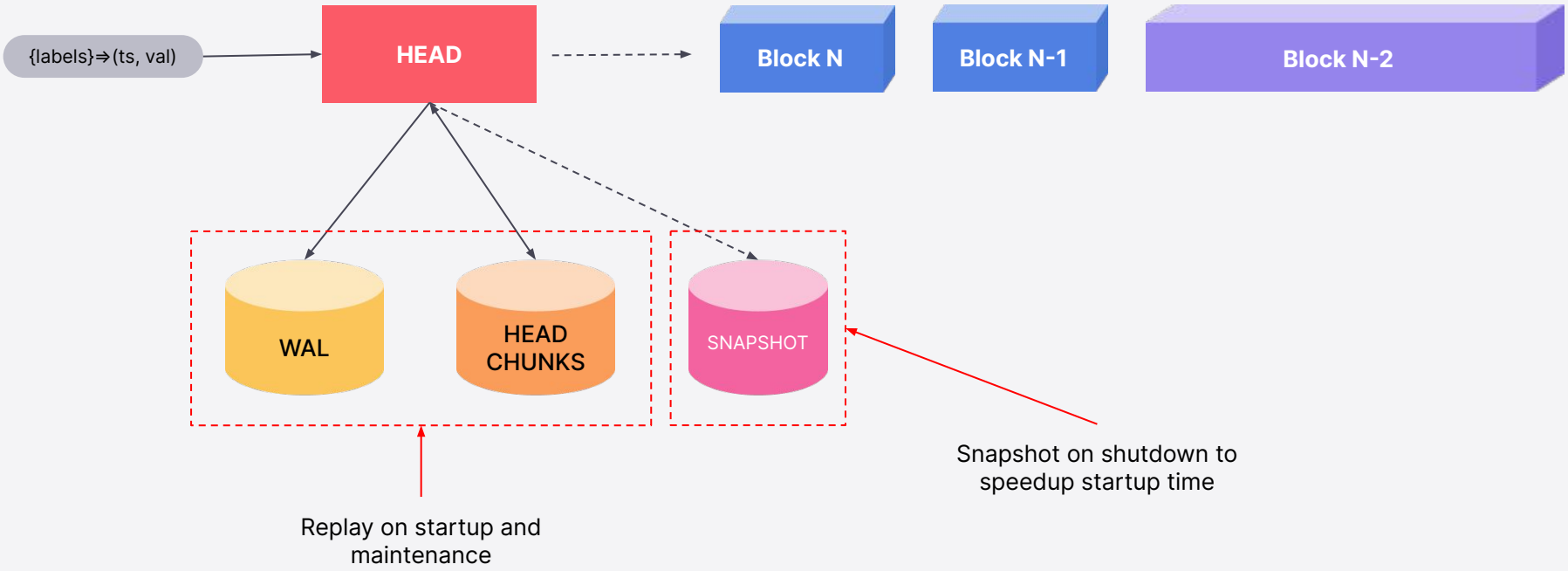
Matcher	Syntax
Equals	{foo=”bar”}
Not Equals	{foo!=”bar”}
Regex Equals	{foo=~”bar.*”}
Regex Not Equals	{foo!~”bar.*”}

Finally, the tombstones



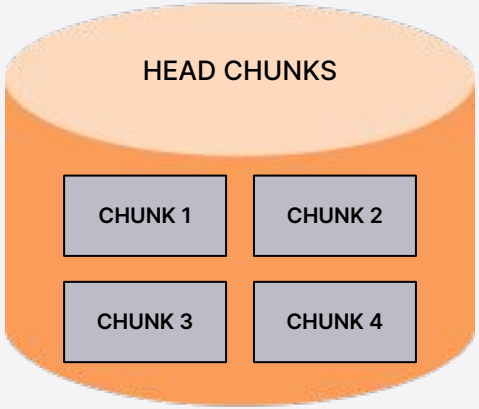

tombstones
s1 ⇒ (1000, 9000)
s1 ⇒ (11000, 15000)
s4 ⇒ (1240, 3500)
s9 ⇒ (100, 10000)

Few more things

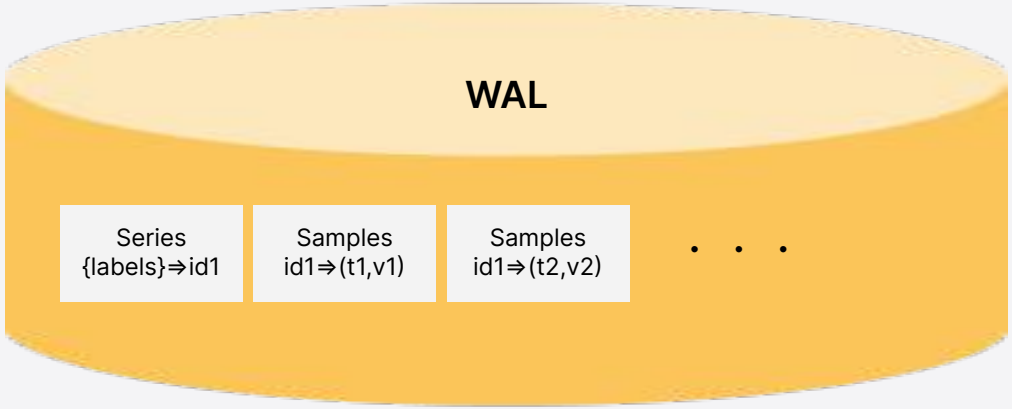


Replay on startup

Replay on startup



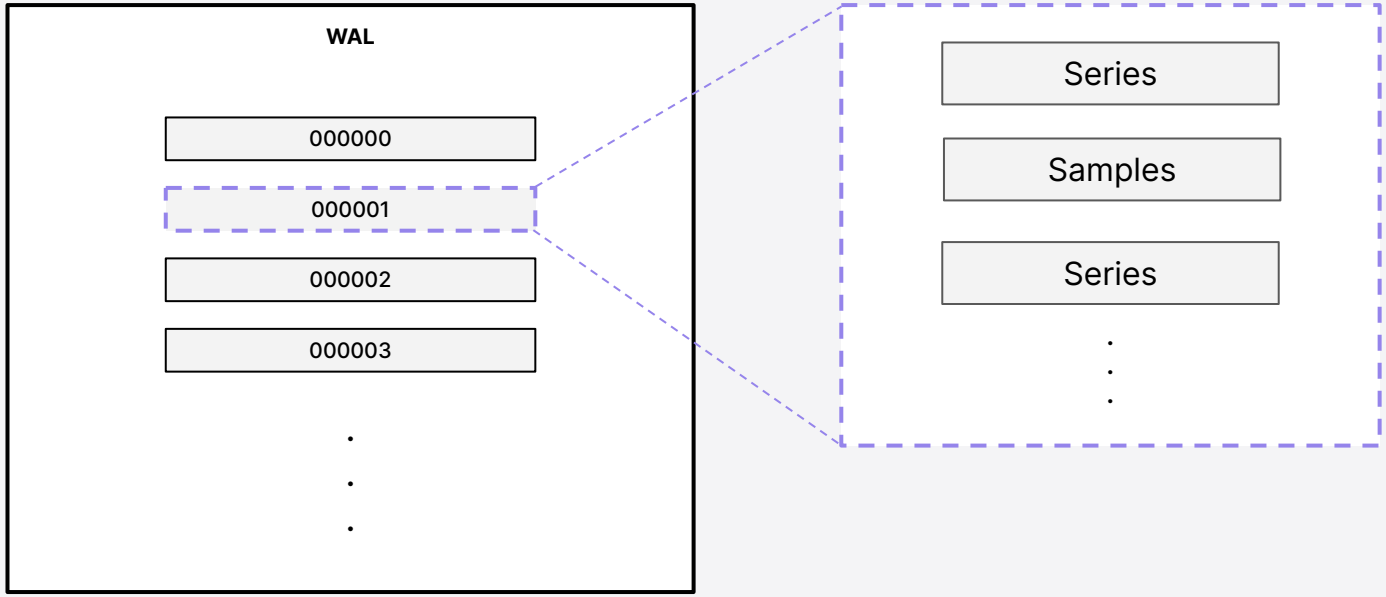
First replay this
Map chunks for all series



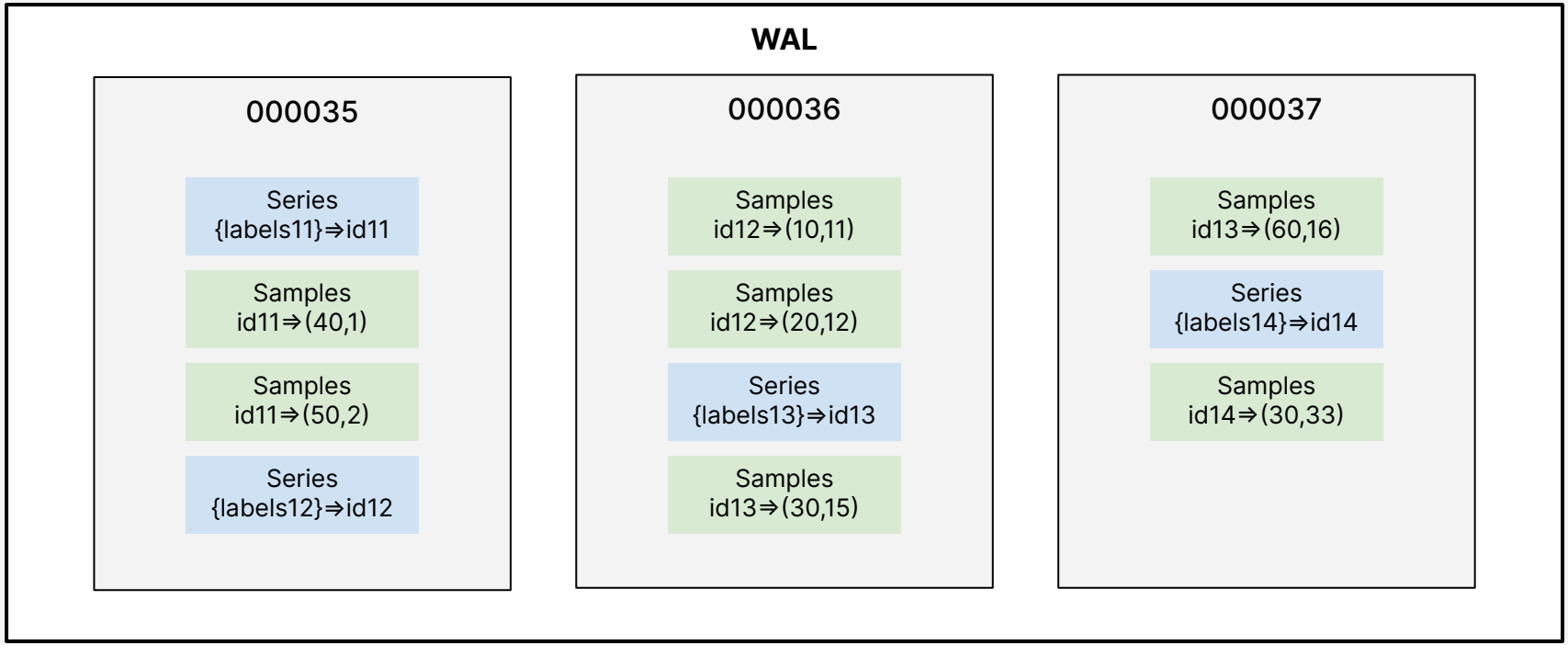
Then replay this
Create series and attach the Head chunks
Replay remaining samples for in-memory chunk

WAL Maintenance

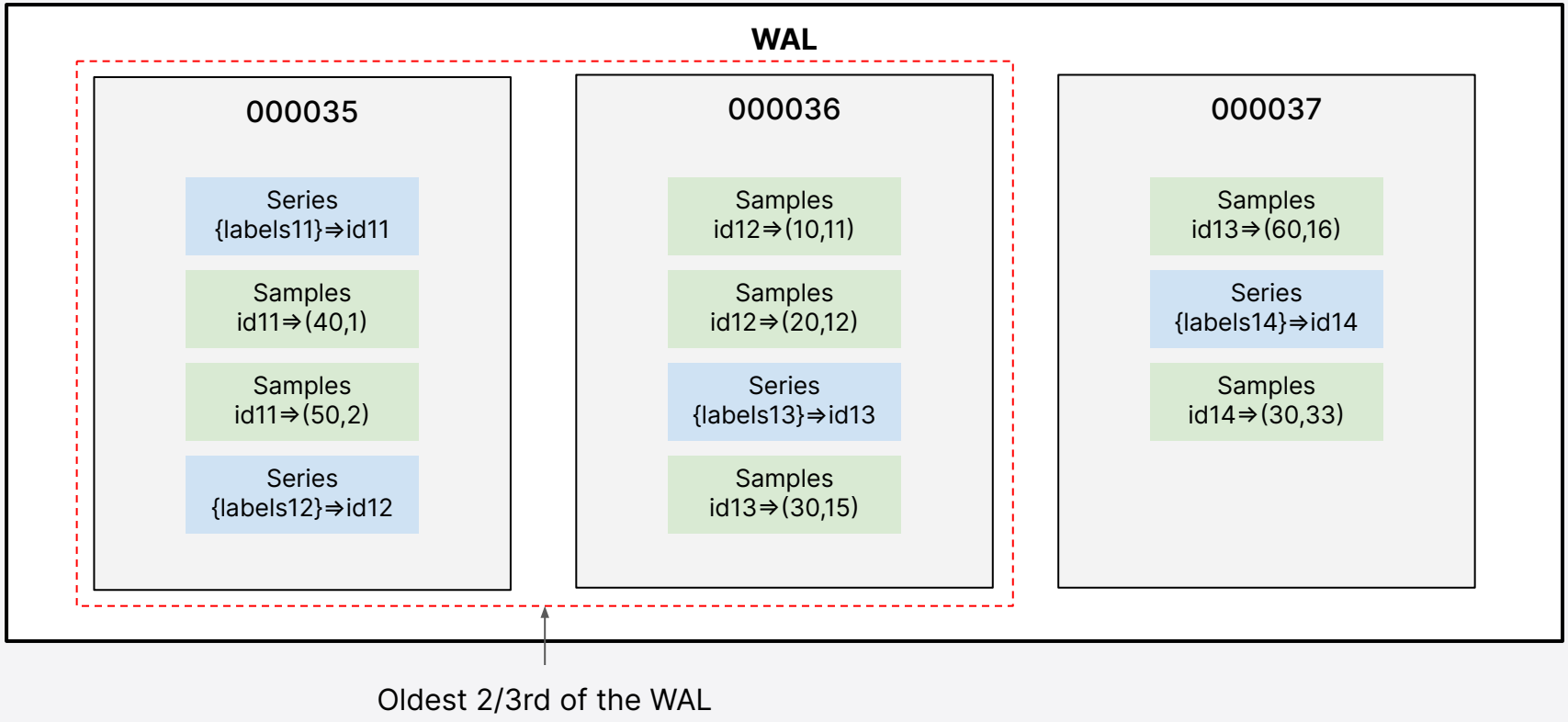
WAL



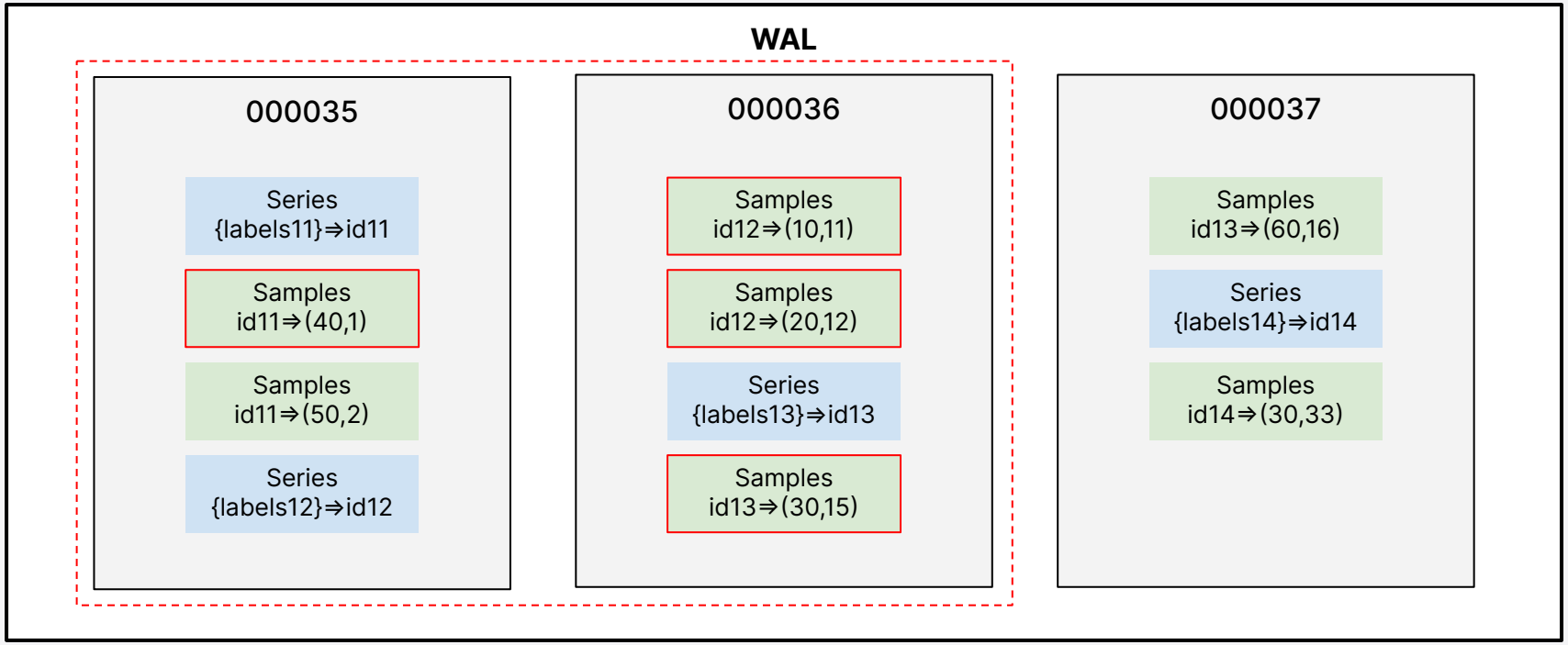
WAL - example file



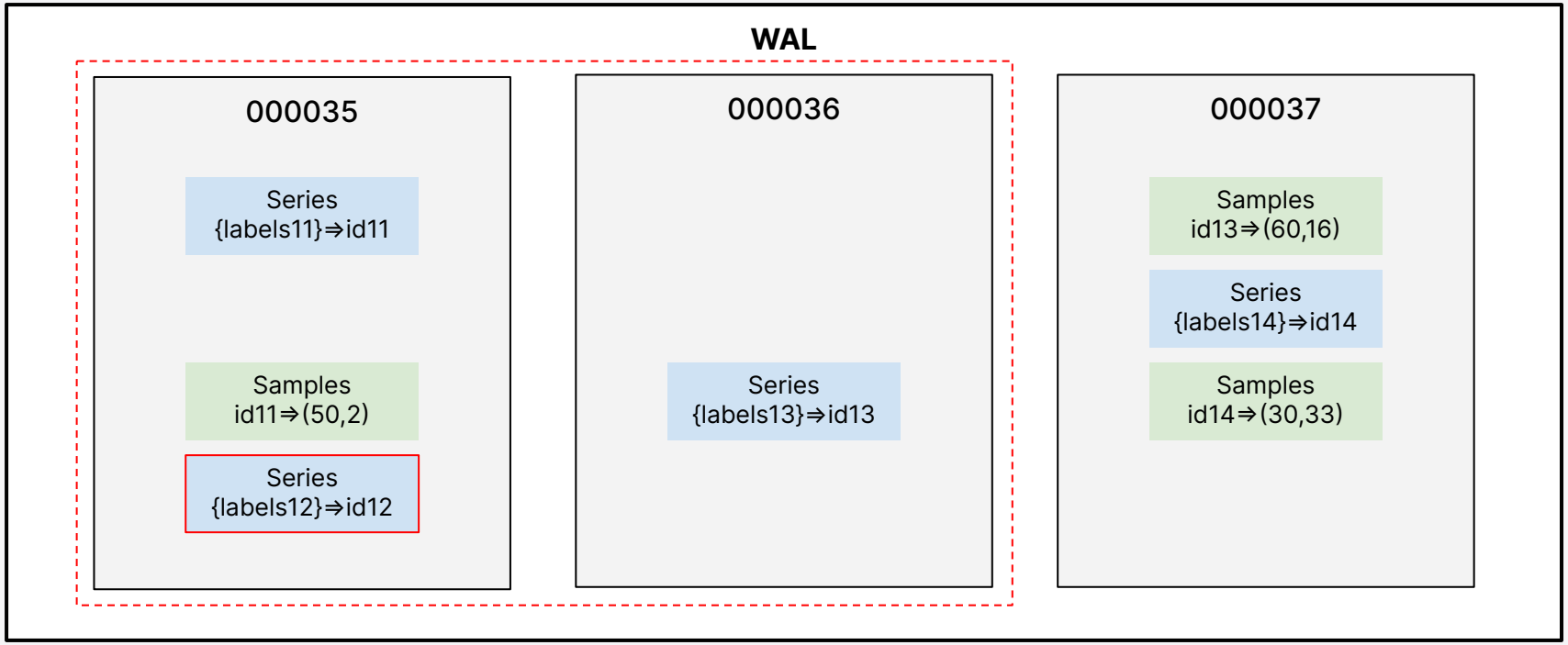
WAL checkpointing



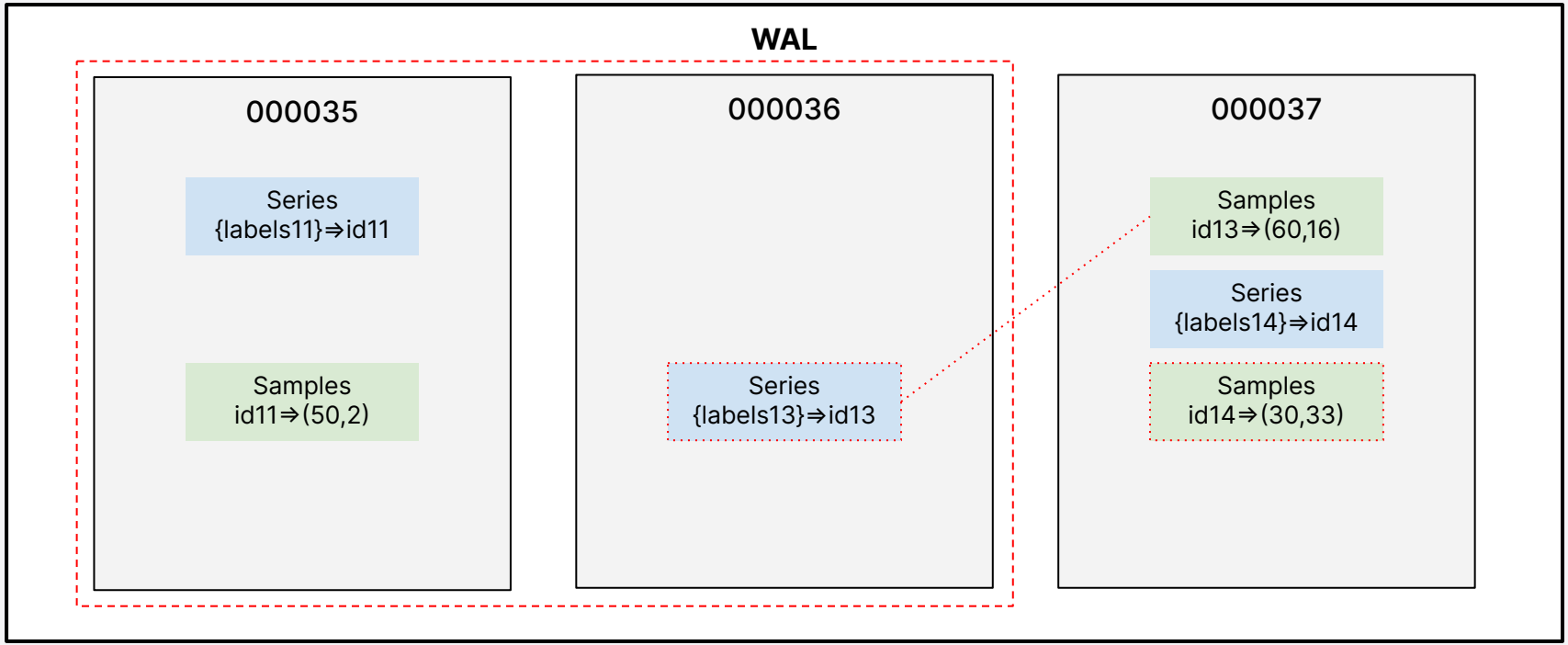
WAL checkpointing, truncate for t<45



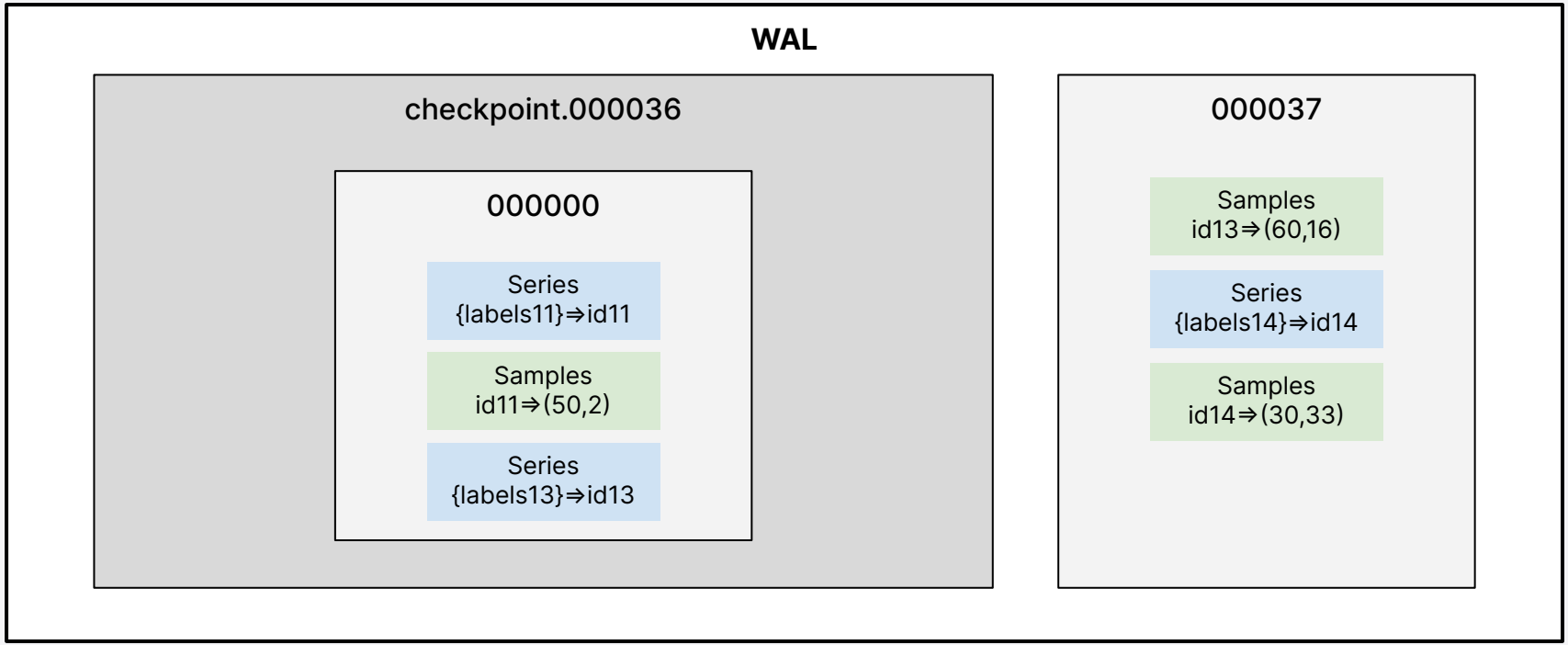
WAL checkpointing, truncate for t<45



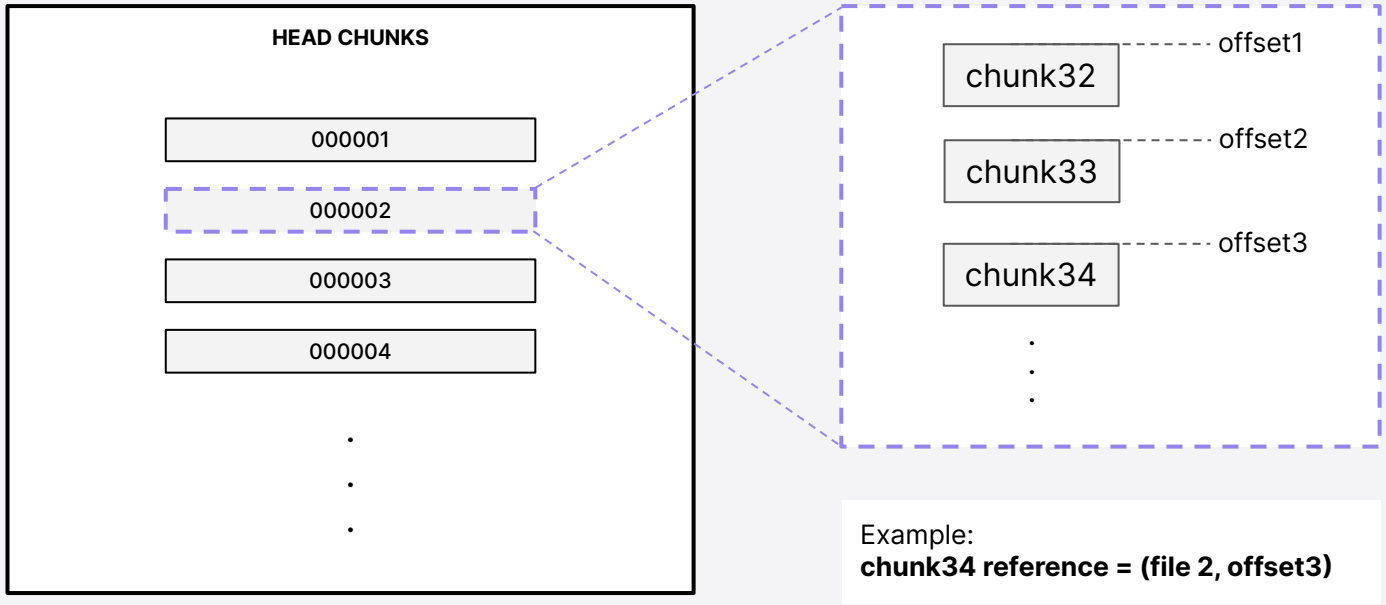
WAL checkpointing, truncate for t<45



WAL checkpointing, truncate for t<45



HEAD CHUNKS
Maintenance

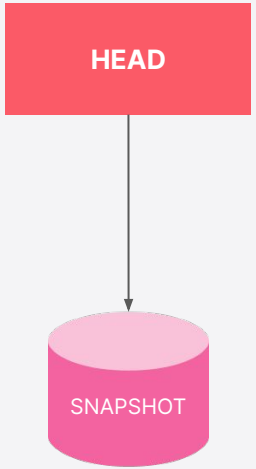


Chunk references in HEAD can tell the oldest file still in use

Snapshot on shutdown

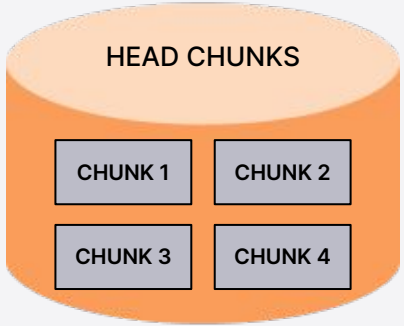
briefly

Snapshot on shutdown

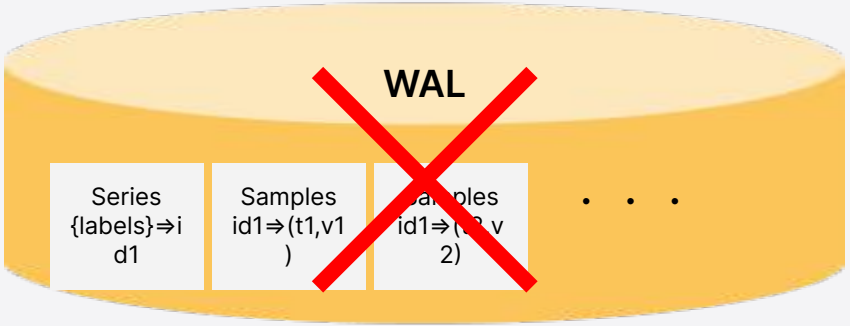


- Snapshot contains all series info and their in-memory chunk
- Taken only during shutdown

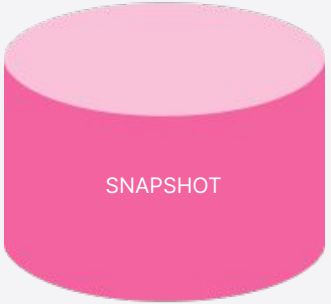
Faster startup times



First replay this
Map chunks for all series



Skip this now



Replay this instead
for the series and
in-memory chunks

This was a movie, here is the novel

https://ganeshvernekar.com/blog/ - 7 part blog post on Prometheus TSDB

1. /blog/prometheus-tsdb-the-head-block
2. /blog/prometheus-tsdb-wal-and-checkpoint
3. /blog/prometheus-tsdb-mmapping-head-chunks-from-disk
4. /blog/prometheus-tsdb-persistent-block-and-its-index
5. /blog/prometheus-tsdb-queries
6. /blog/prometheus-tsdb-compaction-and-retention
7. /blog/prometheus-tsdb-snapshot-on-shutdown

Grafana Labs

Thank you