How Netflix Delivers Key-Value and Time-Series Storage *at Any Scale*

Joseph Lynch & Vidhya Arvind

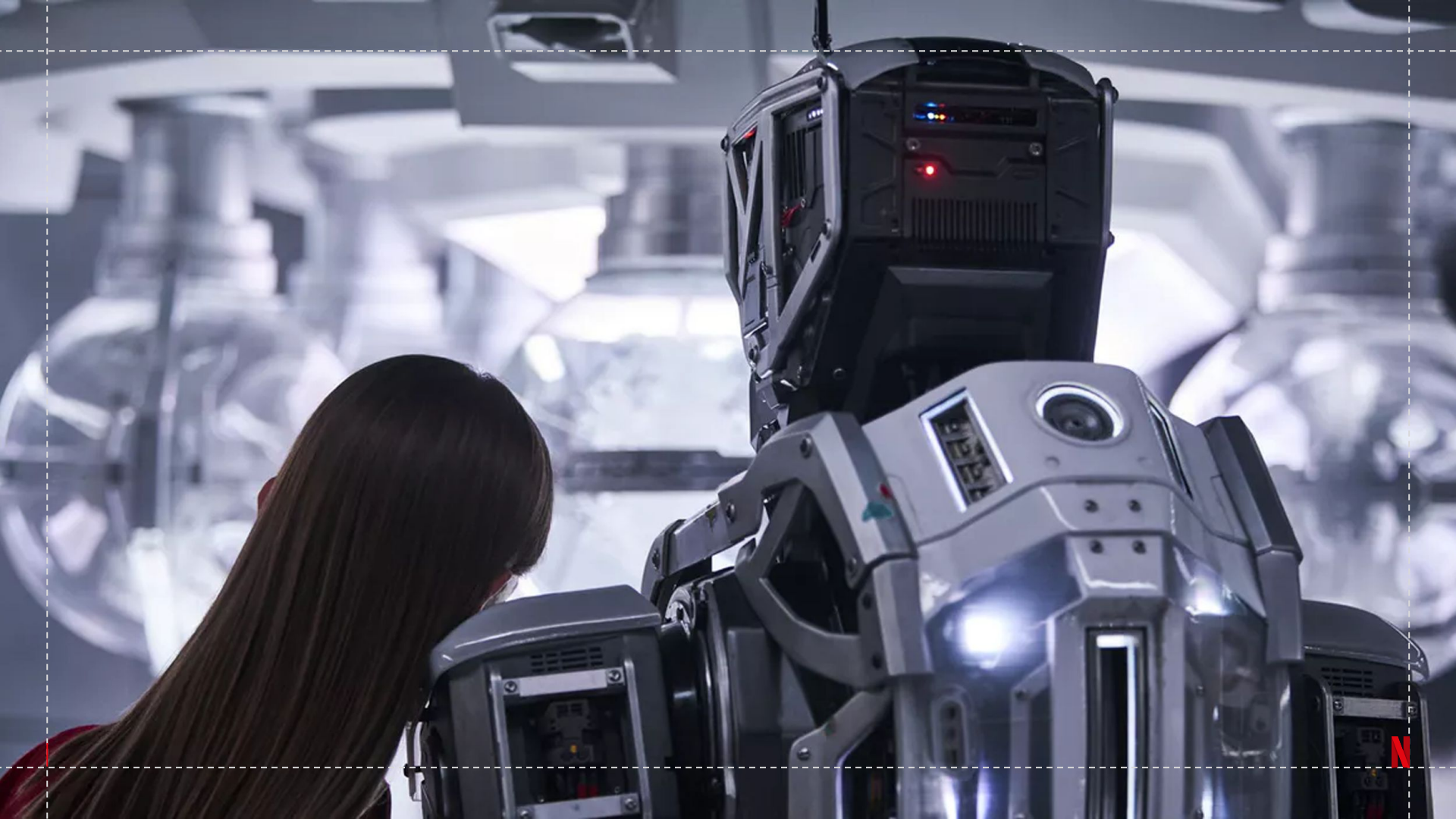
December 12, 2023

Vidhya Arvind

She/Her
Staff Software Engineer
Data Platform @ Netflix

Joseph (Joey) Lynch

He/Him
Principal Software Engineer
Platform @ Netflix
https://jolynch.github.io/



Netflix Storage Problems

All the data

All the scale

All the places

Netflix Online Stateful Scale

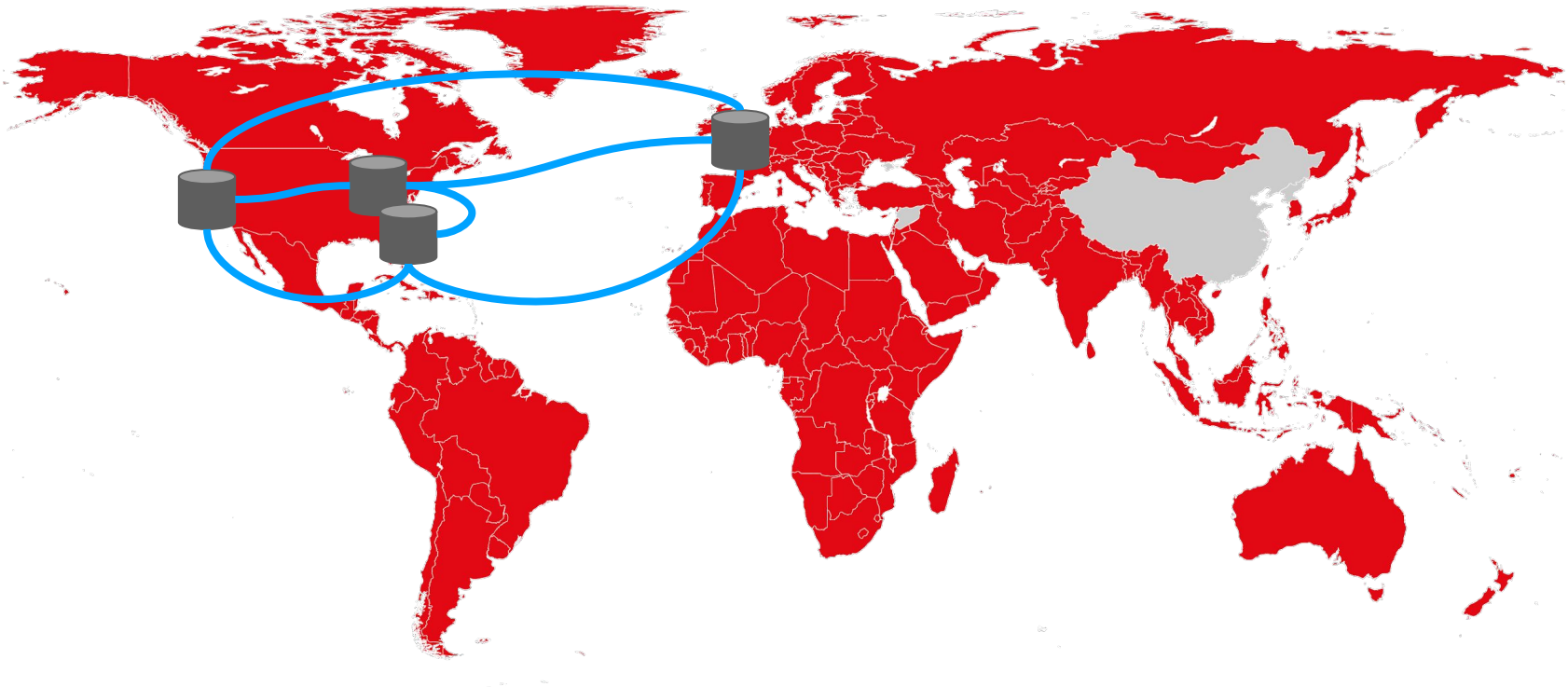
Technology	Traffic	Consistency	Latency SLO	Data Size
Near Cache (e.g. Hollow)	O(1 B/s)	Eventual	O(10us)	O(1 TiB)
Remote Cache (e.g. EVCache)	O(50 M/s)	Best-Effort	O(100us)	O(1 PiB)
Databases (e.g. Apache Cassandra)	O(10 M/s)	In-region RyW, Eventual across	O(1000ms)	O(10 PiB)

https://github.com/Netflix/Hollow

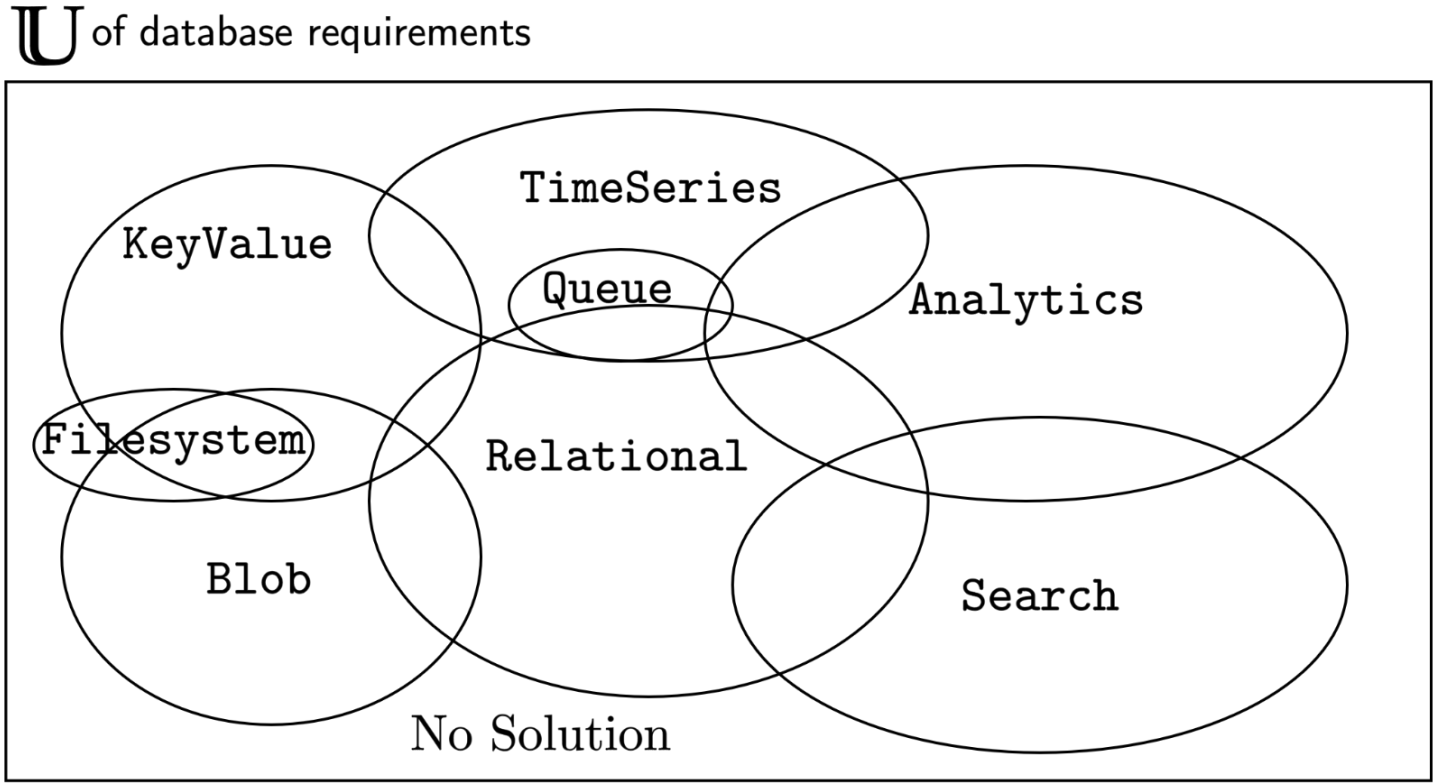
https://github.com/Netflix/EVCache

https://github.com/apache/cassandra

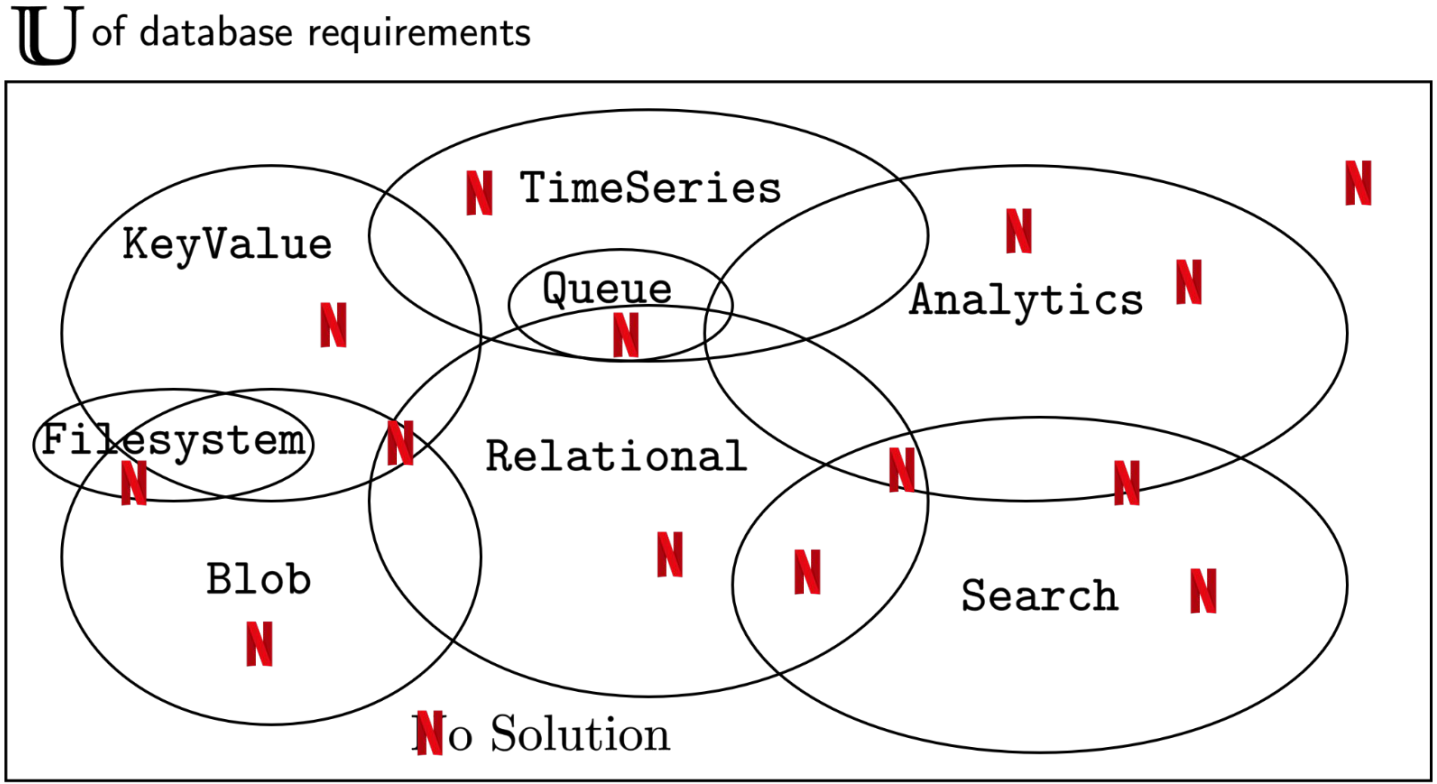
Netflix Online
Stateful Scale



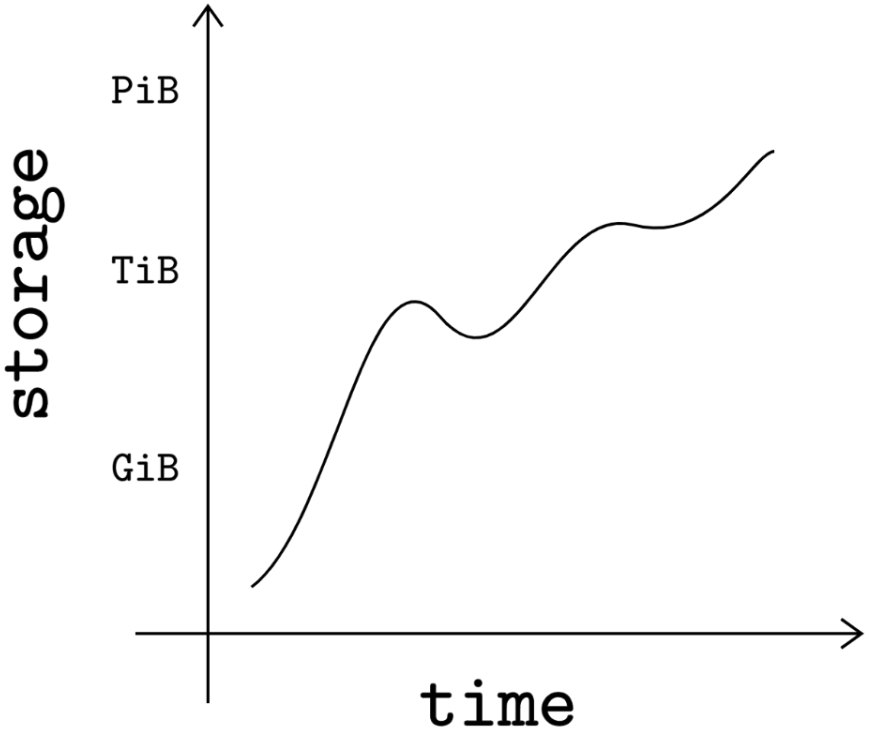
Problem: Varied Requirements

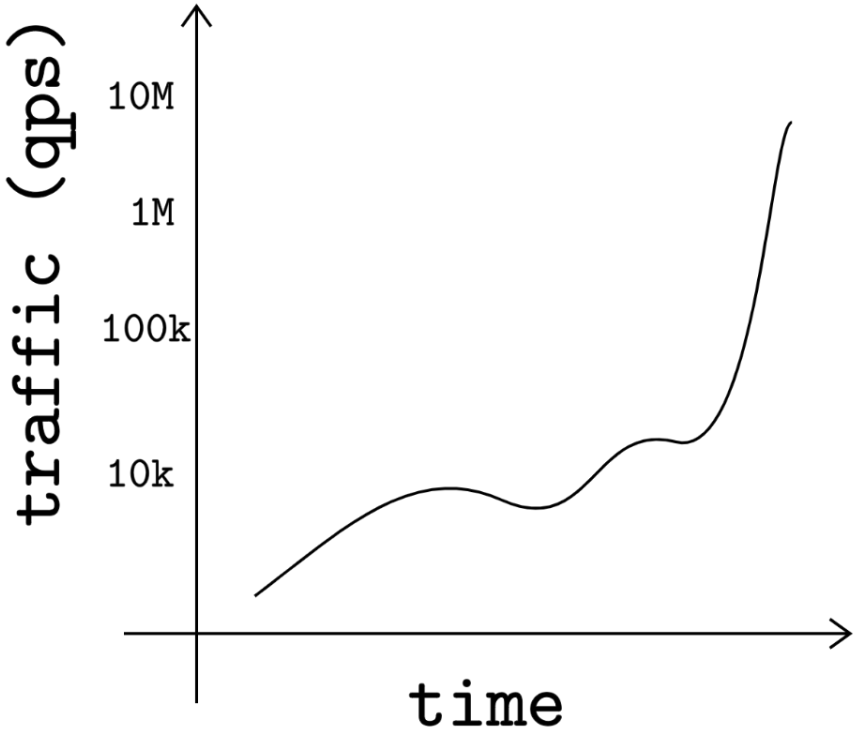


Problem: Varied Use-Cases

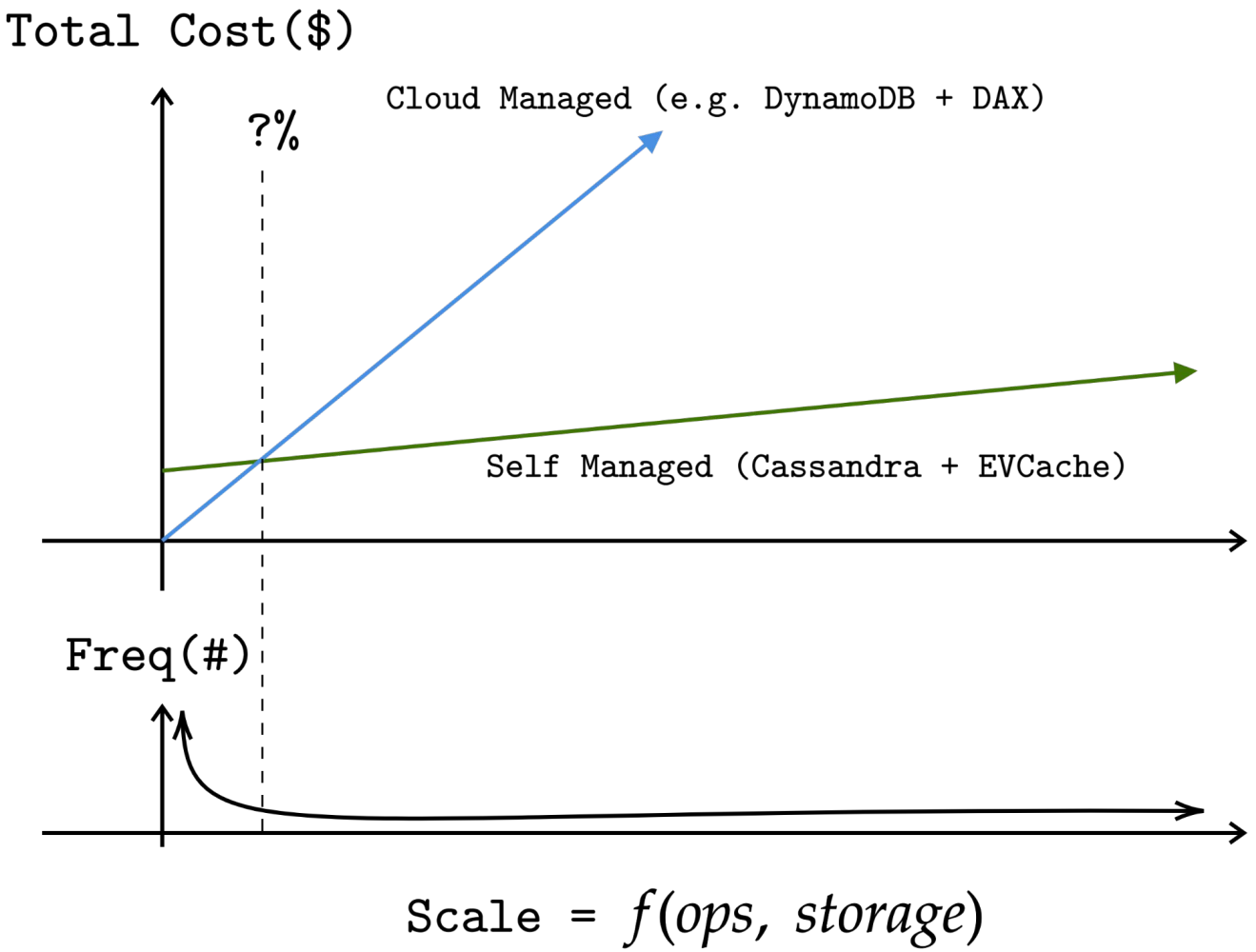


Problem: Scaling





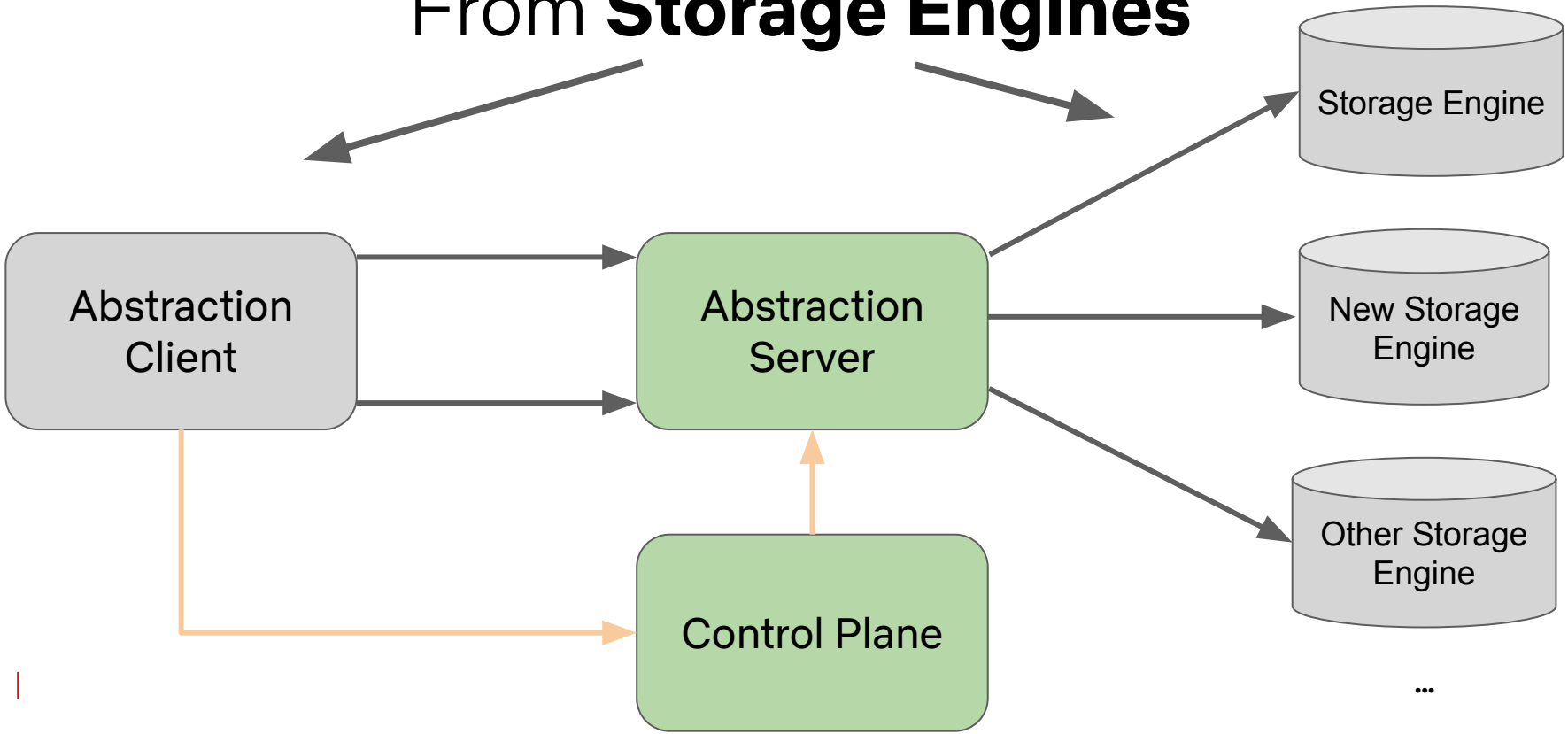
Problem: Scale-Cost Dynamics



Solution?

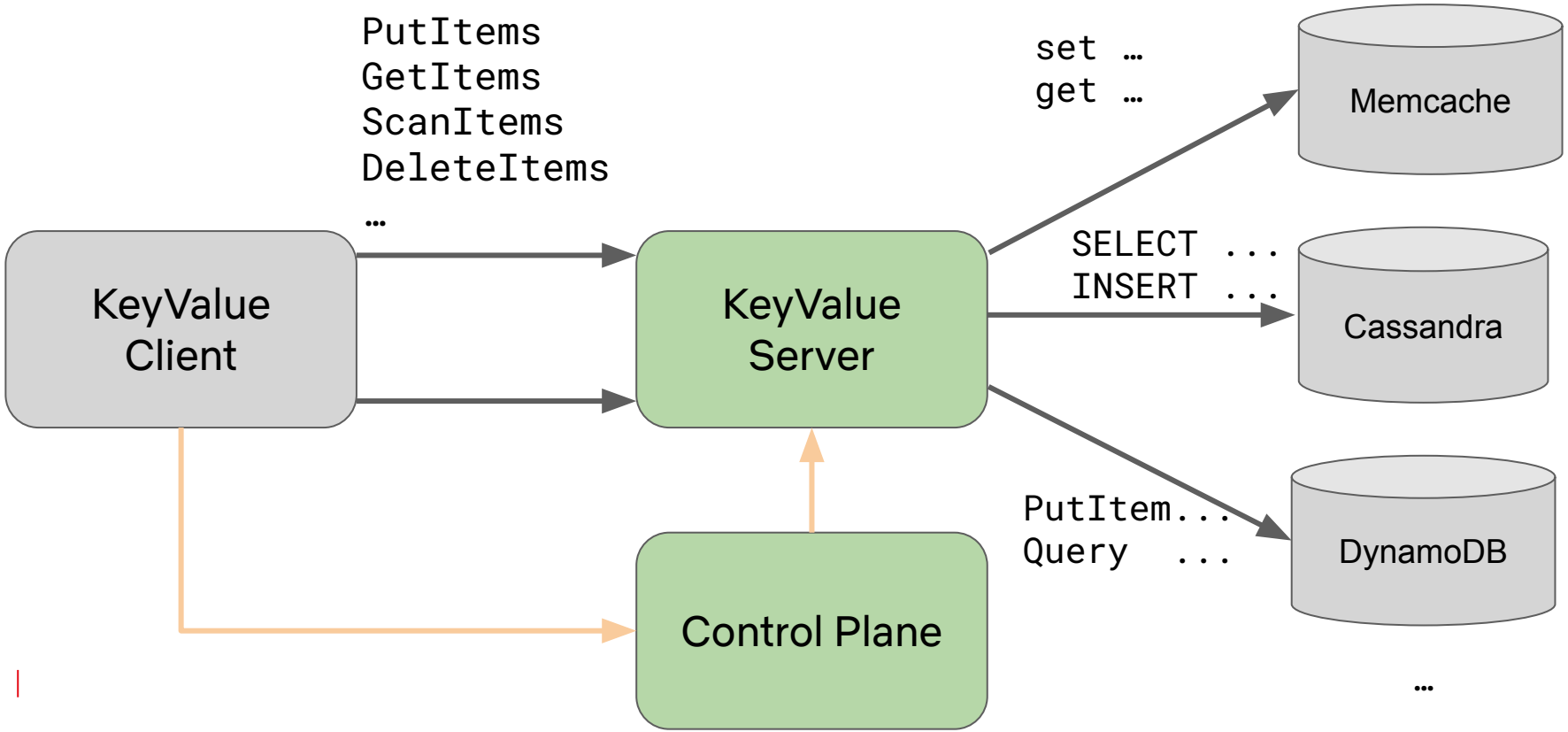
Solution?
Data **Abstraction** Layers!

Abstract **Clients** From **Storage Engines**



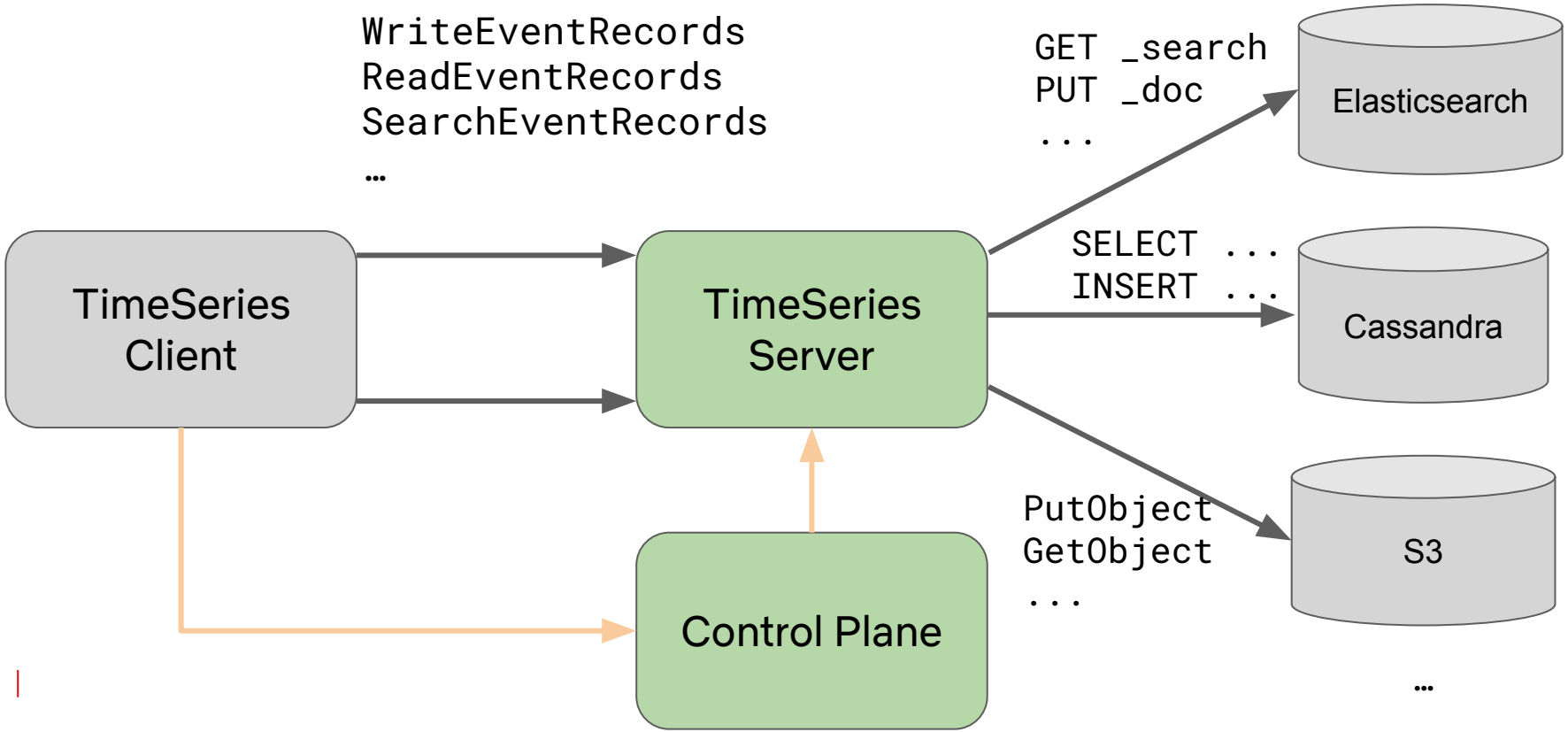
Key-Value

"Two-Level Map as a Service"



Time-Series

"Massively Scalable Event Store"



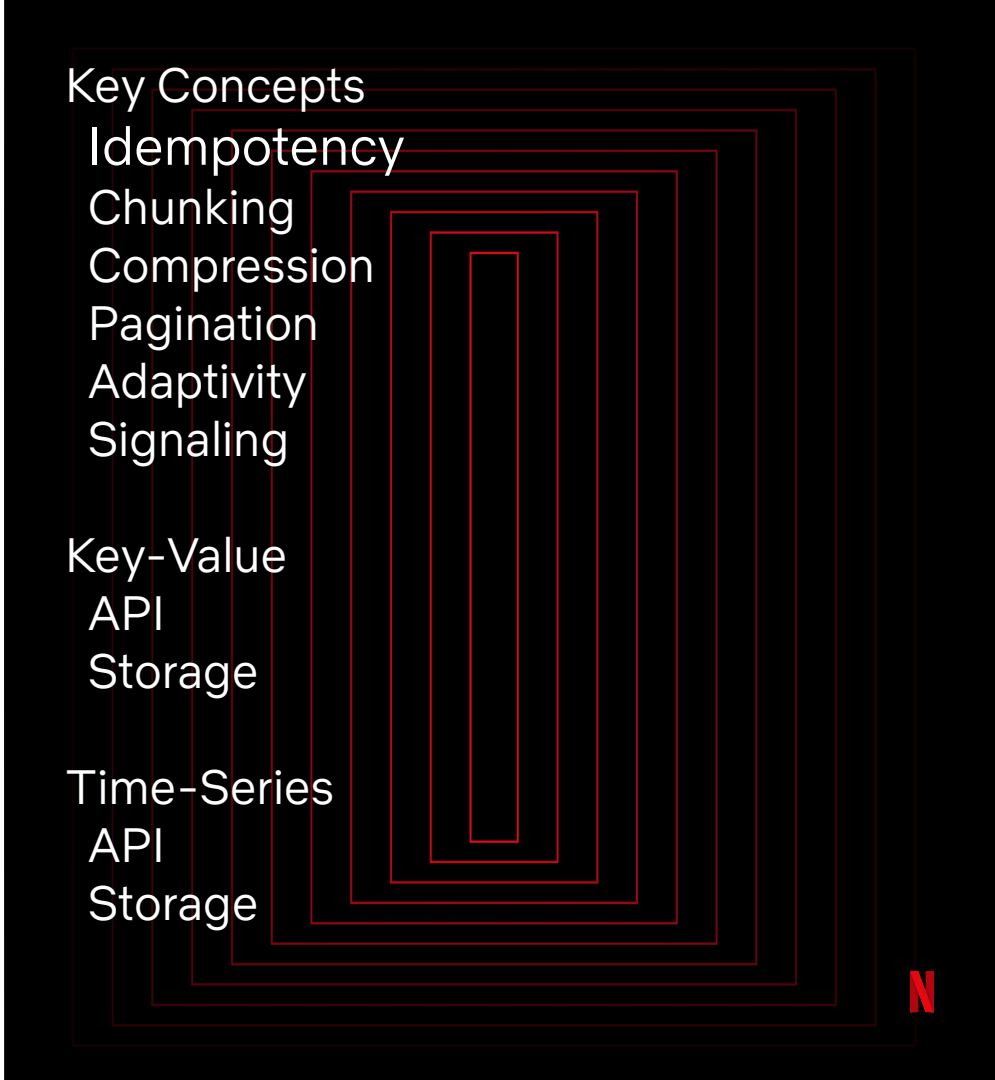

JESUS
SAVES

Storage At Any Scale

Key Concepts
- Idempotency
- Chunking
- Compression
- Pagination
- Adaptivity
- Signaling

Key-Value
- API
- Storage

Time-Series
- API
- Storage

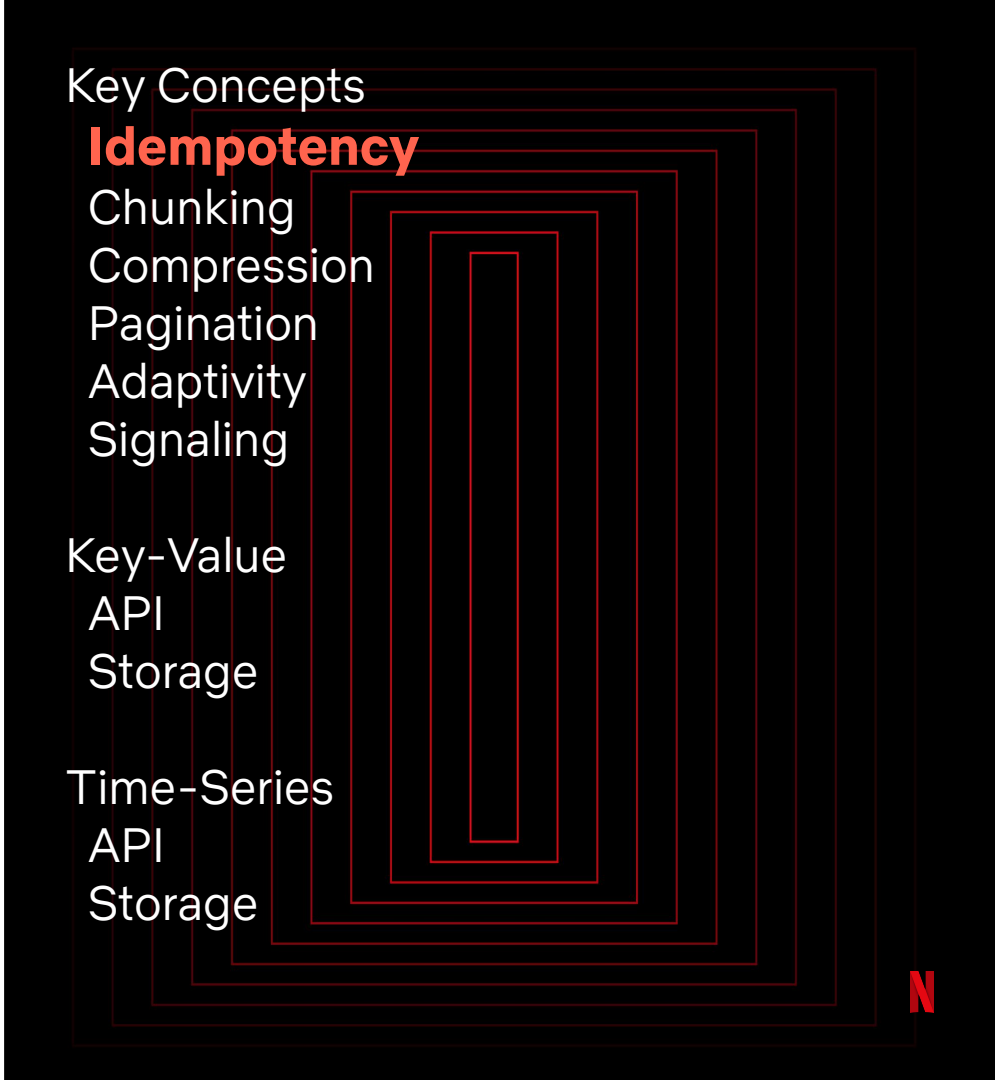


Storage At Any Scale

Key Concepts
- **Idempotency**
- Chunking
- Compression
- Pagination
- Adaptivity
- Signaling

Key-Value
- API
- Storage

Time-Series
- API
- Storage



Retriable Writes

Assume you have to retry, how do you make your mutable API's safe?

Retriable Writes

Assume you have to retry, how do you make your mutable API's safe?

Idempotency Tokens!

```
def do_write():
  token = get_token()
  result = write(
     data=...,
     idempotency_token=token
  )
  # When hedging or retrying
  # Send the same token
  if result.status != SUCCESS:
    retry_write(
       data=...,
       idempotency_token=token)
```

Idempotency Tokens

Uniquely identify the mutations with an idempotency token.

```
// Mutation operations should be idempotent,
// and idempotency tokens are how you can
// achieve that. Writes retry with the same
// token - then storage engines de-duplicate.

// Typically a LWW clock or a version clock.
message IdempotencyToken {
  google.protobuf.Timestamp generation_time = 1;
  string token = 2;
}
```

Clients generate monotonic timestamps

```
last_token = (0, uuid4())

def get_token() -> IdempotencyToken:
  timestamp_micros = max(
    now_in_micros(),
    last_token.generation_time + 10
  )
  nonce = uuid4()
  return (mix_last_digit(timestamp, nonce), nonce)
```

Isolated LWW Tokens

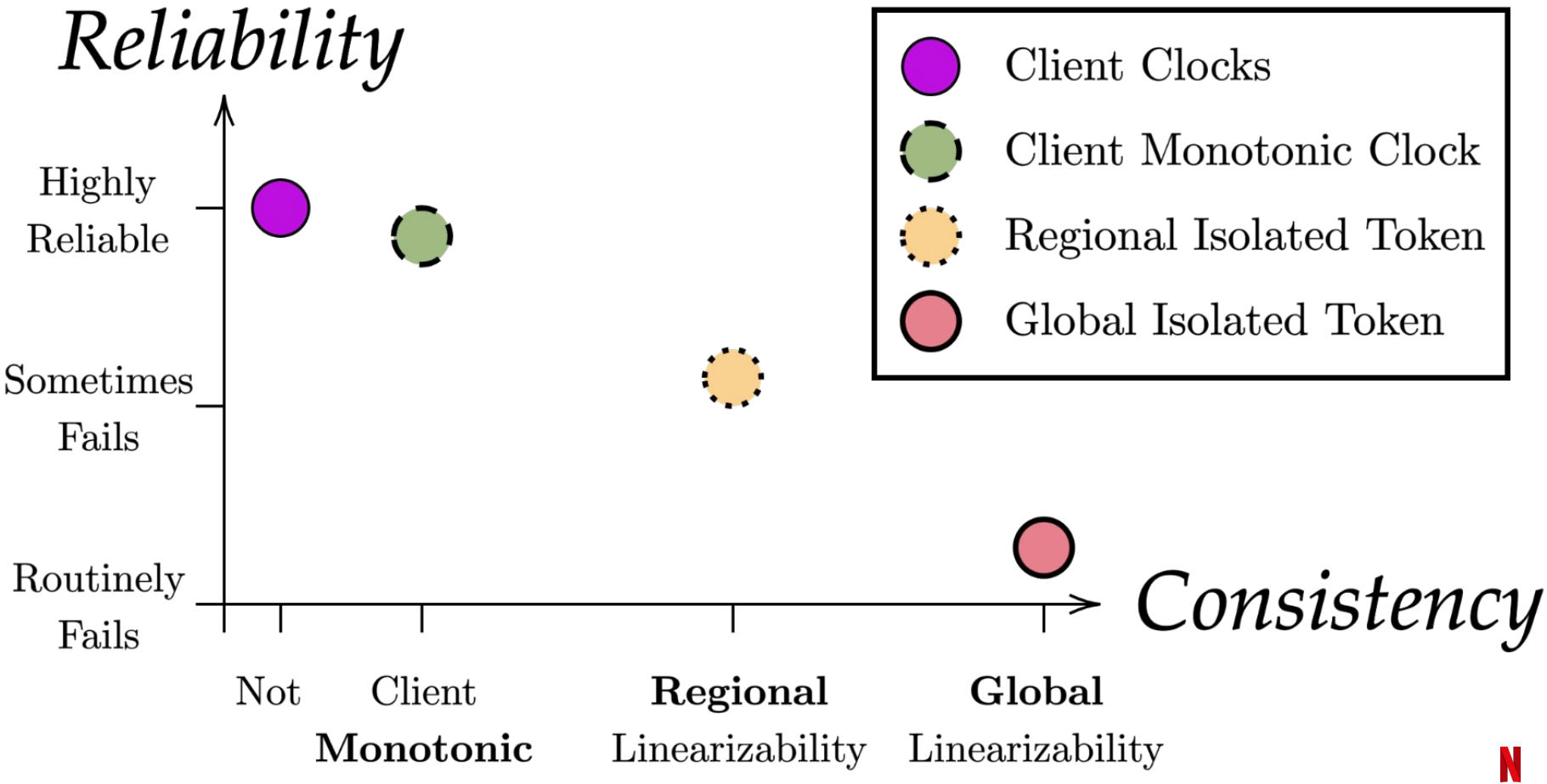
Servers make monotonic timestamps within same region

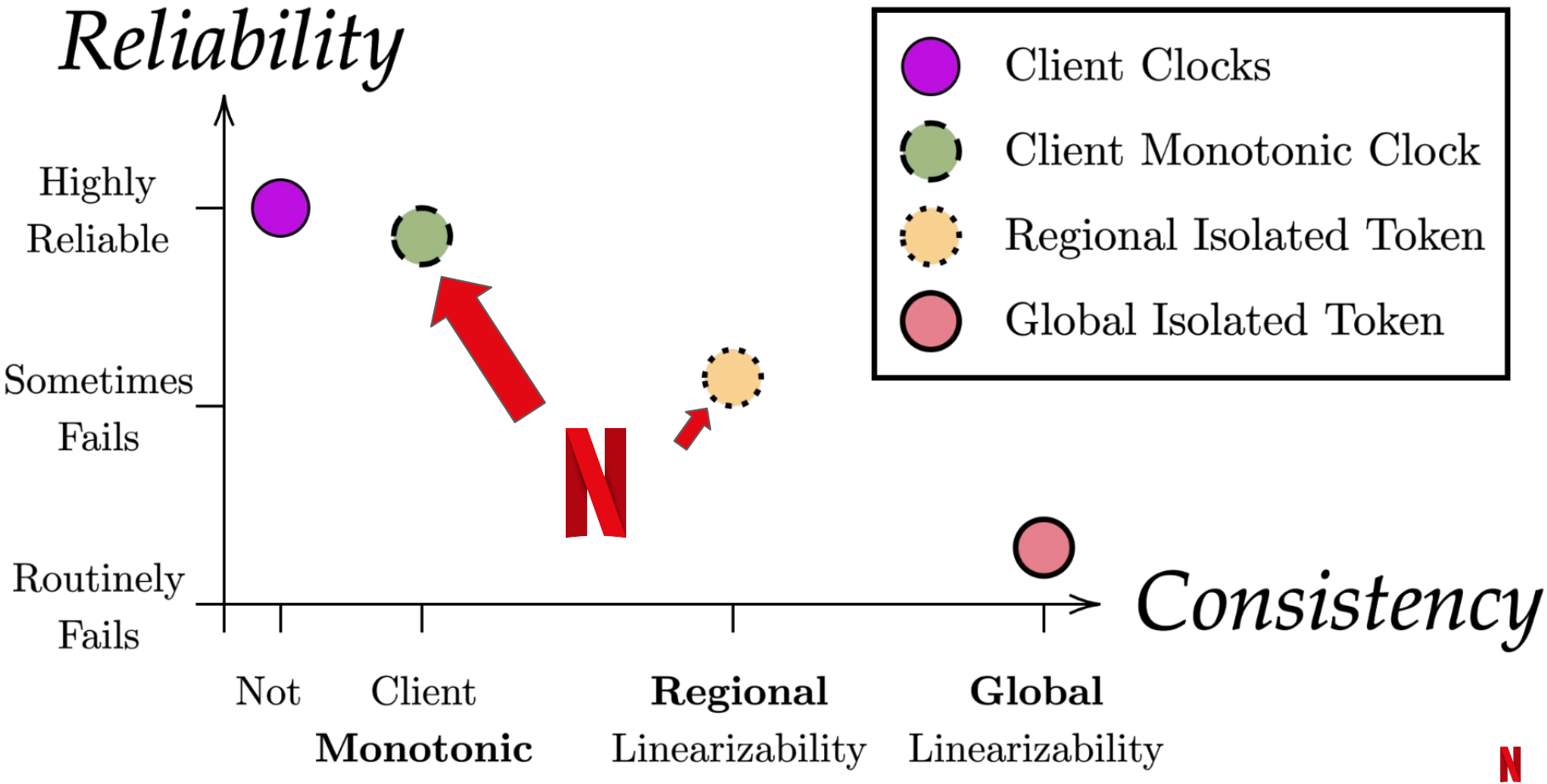
```
last_token = (0, uuid4())

def do_isolated_write(mutations):
  # Share random nonce
  nonce = uuid4()
  # Lock keys, send writes, release
  with acquire_clock(mutations.keys) as timestamp:
    for write in mutations:
      perform_write((timestamp.bump(), nonce), write)
  # close releases at last timestamp + 1
```

Isolated Global Tokens

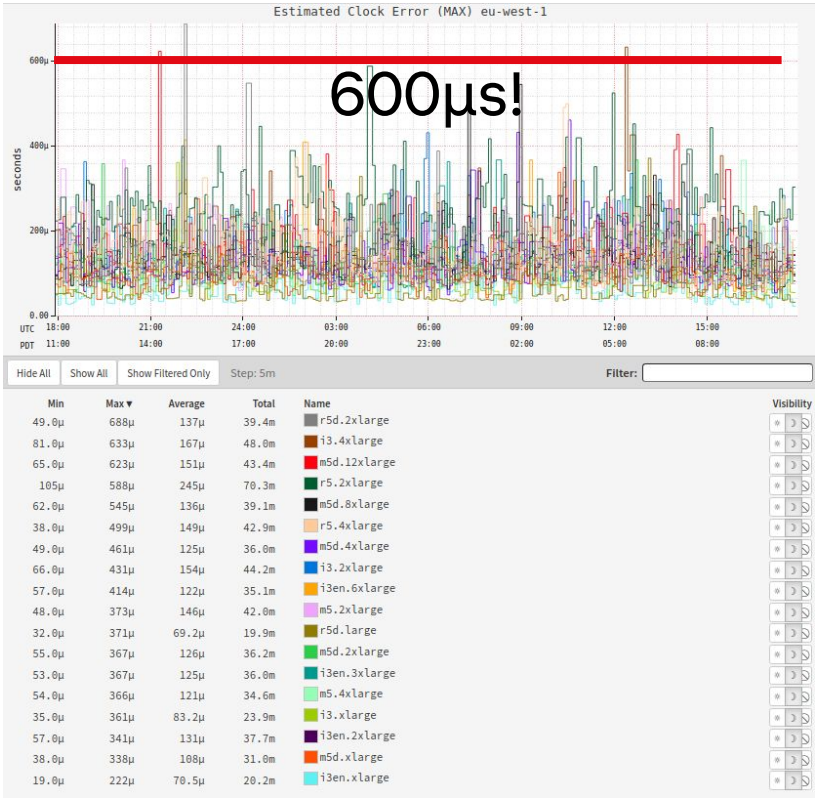
Servers make global monotonic version

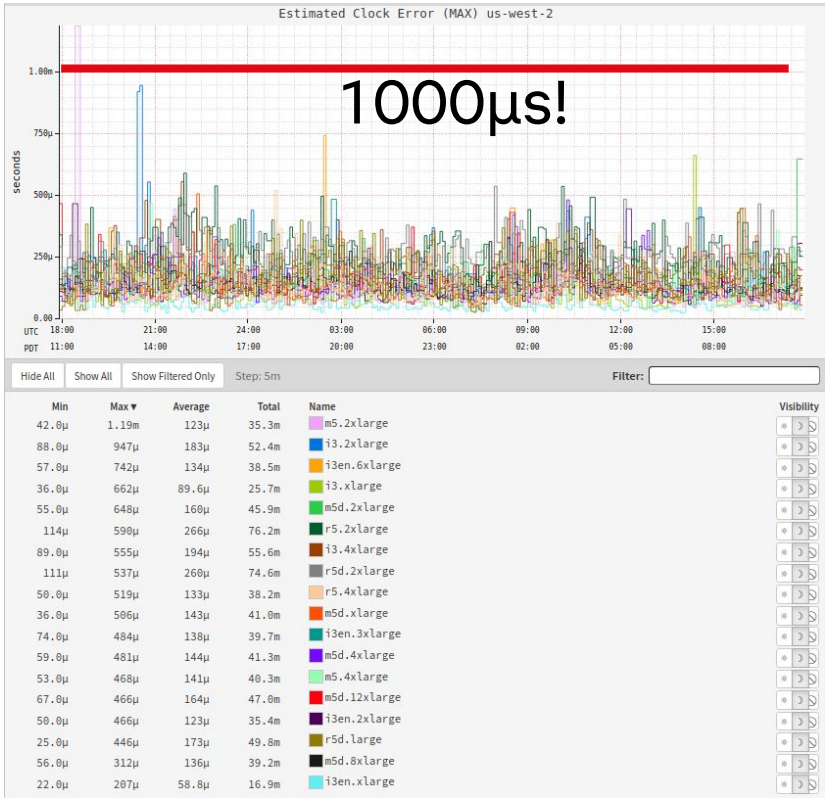
```
last_token = (0, uuid4())

def do_isolated_write(mutations):
  # Share generation timestamp
  timestamp = now()
  # Acquire global transaction id and perform transaction
  with acquire_version(mutations) as token:
    for write in mutations:
      perform_write((timestamp, token), write)
```


Reliability
Highly Reliable
Sometimes Fails
Routinely Fails
Client Clocks
Client Monotonic Clock
Regional Isolated Token
Global Isolated Token
Consistency
Not
Client
Monotonic
Regional
Linearizability
Global
Linearizability


Reliability
Highly Reliable
Sometimes Fails
Routinely Fails
Client Clocks
Client Monotonic Clock
Regional Isolated Token
Global Isolated Token
Consistency
Not
Client
Monotonic
Regional
Linearizability
Global
Linearizability

Clocks Work





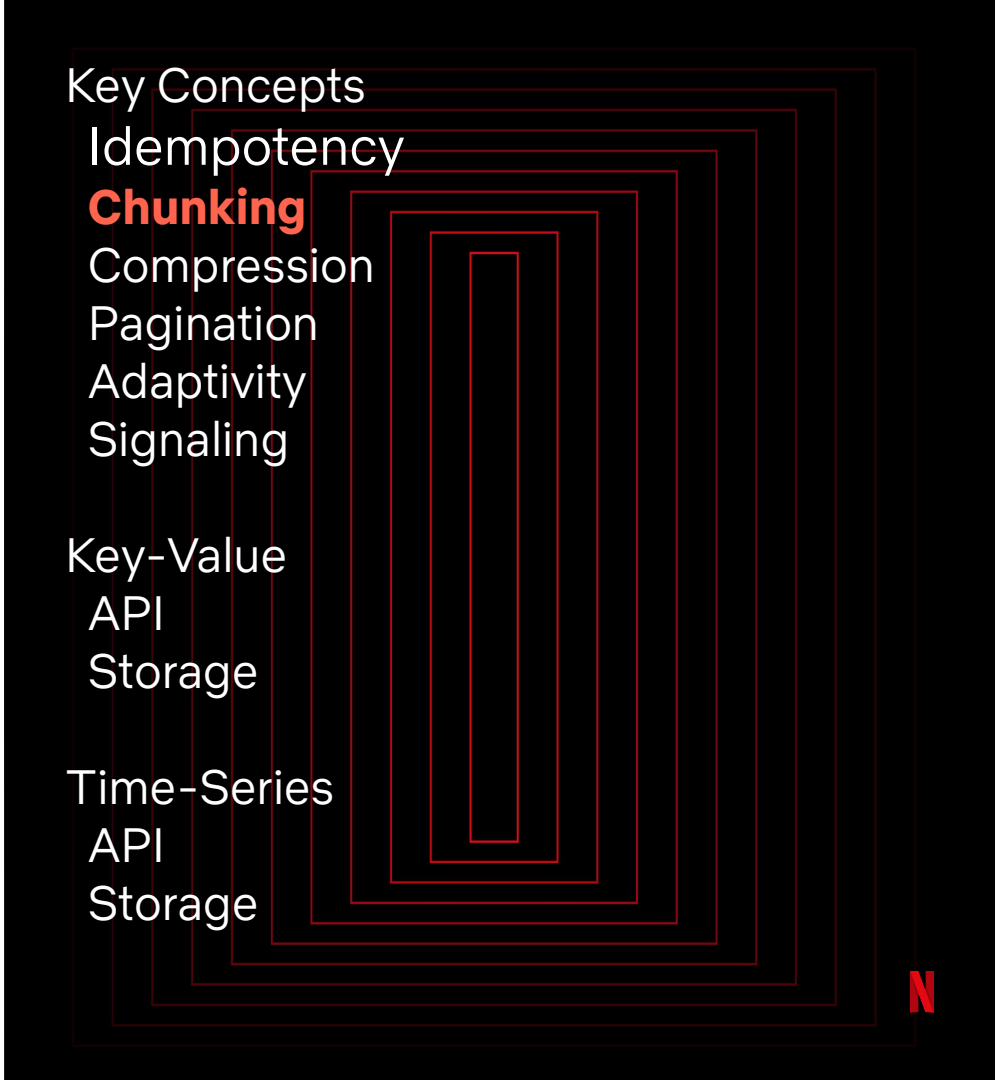
Clock Drift Memo: https://docs.google.com/document/d/1XLBjQ9scZCy-xIo51Rs--CSdFV781fnp5hXdXTBAk1k/edit

Storage At Any Scale

Key Concepts
- Idempotency
- **Chunking**
- Compression
- Pagination
- Adaptivity
- Signaling

Key-Value
- API
- Storage

Time-Series
- API
- Storage



Small data

Payload size < O(1MiB)

```
# Chunk after payload size 1 MiB
chunk_after_size = 1024 * 1024

# Send data to storage engine directly in one
# stage + commit step
if sizeof(payload) < chunk_after_size:
  perform_write(payload)
```

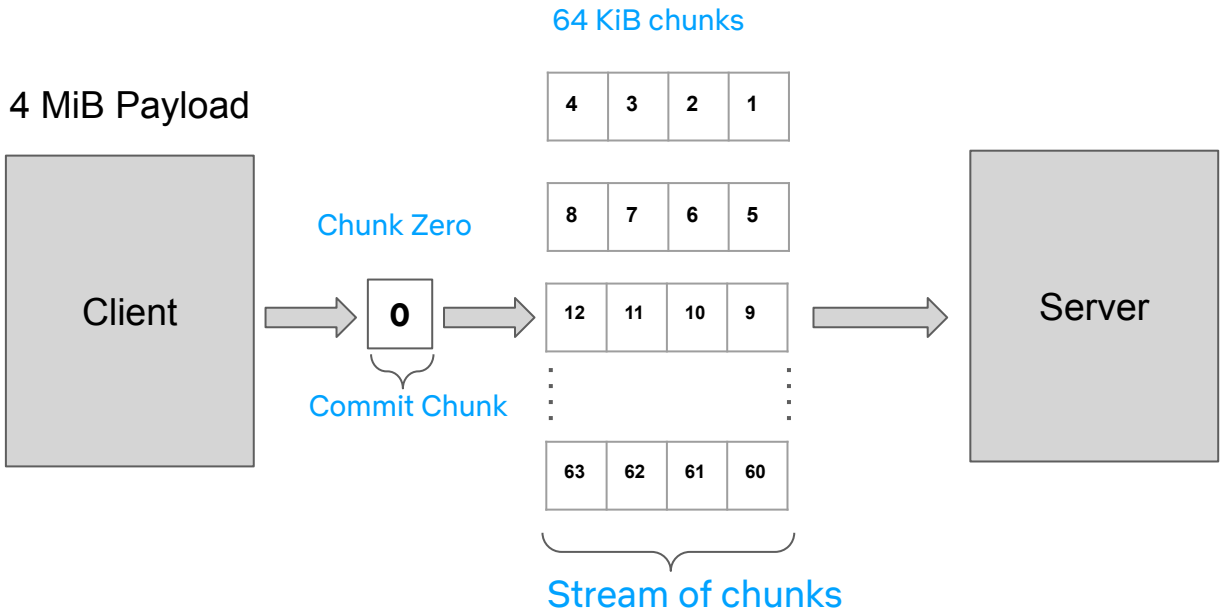
Small data

Payload size < O(1MiB)

```
# Chunk after payload size 1 MiB
chunk_after_size = 1024 * 1024

# Send data to storage engine directly in one
# stage + commit step
if sizeof(payload) < chunk_after_size:
  perform_write(payload)

else ?
```

Chunk Data

Write Path: Payload size: O(> 1MiB)




```
chunk-after-bytes = 1 MiB
chunk-size-bytes  = 64 KiB
```

Chunk Data

Write Path: Payload size: O(> 1Mib)

```
# Step 1: Stage data chunks and use the idempotency
# token to tie chunks together
page, writes = page(), []
for idx, chunk in enumerate(chunk(payload)):
    item = Item(
        key=key, value=chunk,
        chunk=(idx + 1),
        token=idempotency_token
    )
    page.add(item)
    if page.size > 2 * 1024 * 1024:
        writes.add(perform_write(page))
        page.clear()
writes.add(perform_write(page))
```

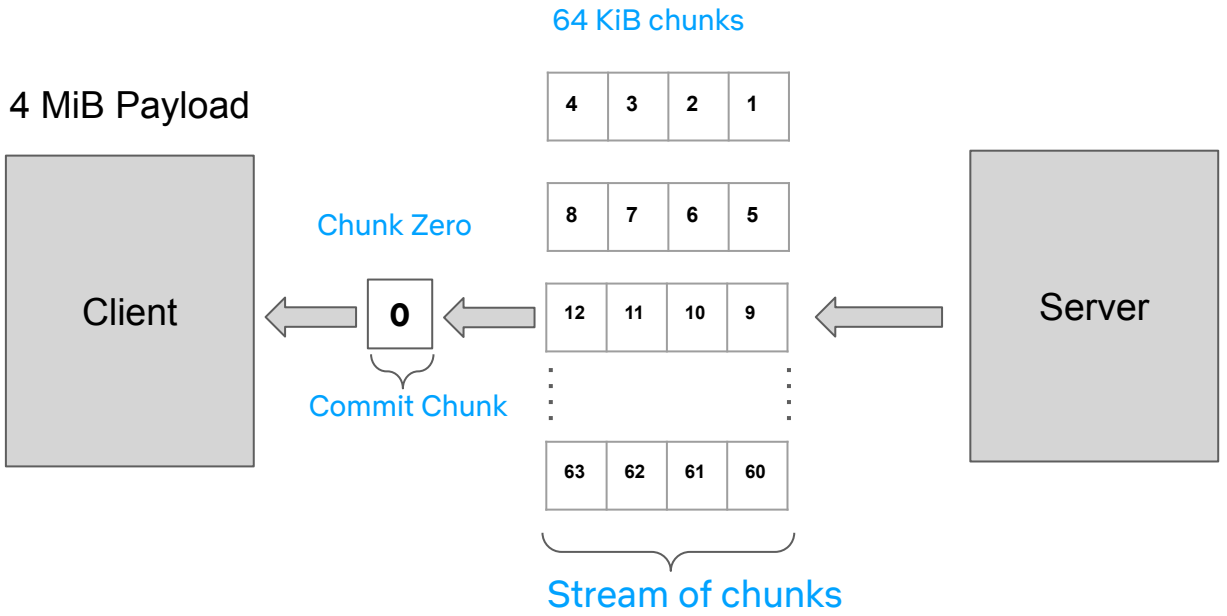
Chunk Data

Write Path: Payload size: O(> 1Mib)

```
# Step 2: Commit the write by writing chunk zero
perform_write(
  Item(
    key        = key,
    chunk      = 0,
    token      = idempotency_token,
    metadata = {
     chunk_count        = 43,
     chunk_size_bytes   = 64KiB,
     # Filled by server
     chunk_token_xxh64 = xxh64(token.nonce),
     chunk_offset       = consistent_hash(key)
    }
  ))
```

Chunk Data: Read Path

Read Path: Payload size: O(> 1Mib)




```
chunk-after-bytes = 1 Mib
chunk-size-bytes  = 64 Kib
```

Chunk Data

Read Path: Payload size: O(> 1Mib)

```
# Server-side: Read chunk 0s value_metadata for
# number of chunks to retrieve
chunks, commit = [], read_chunk_zero(id, key)
version        = commit.metadata.version
num_chunks     = commit.metadata.chunks
chunk          = 1
while sizeof(chunks.value) < page_limit:
  last_chunk = perform_chunk_read(chunk)
  chunks.add(last_chunk)
page_token = last_chunk.chunk < num_chunks
             ? construct_page_token(last_chunk)
             : null
return (chunks, page_token)
```

Chunk Data

Read Path: Payload size: O(> 1Mib)

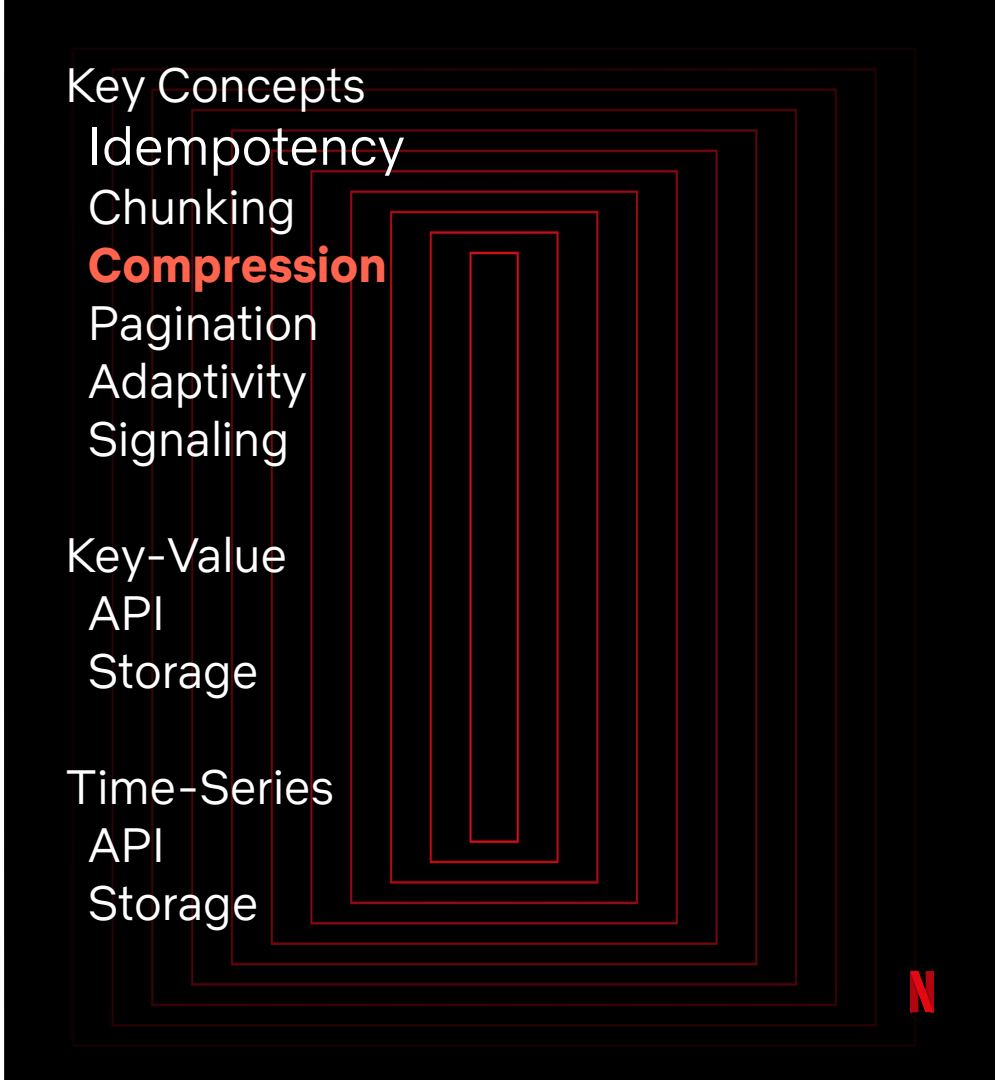
```
# Client-side: retrieve pages and stitch chunks
for page in pages:
 for item in page.items:
   current_item = select(current_item, item)
   # SMALL ITEM: Read the entire item from the single RPC
   if !current_item.metadata.has_chunk_metadata():
     yield current_item
   else
     # LARGE ITEM: append the chunk to the item
     current_item.value.append(item.value)
     if item.chunk == current_item.metadata.chunk_count:
       yield current_item
       current_item = null
```

Storage At Any Scale

Key Concepts
- Idempotency
- Chunking
- **Compression**
- Pagination
- Adaptivity
- Signaling

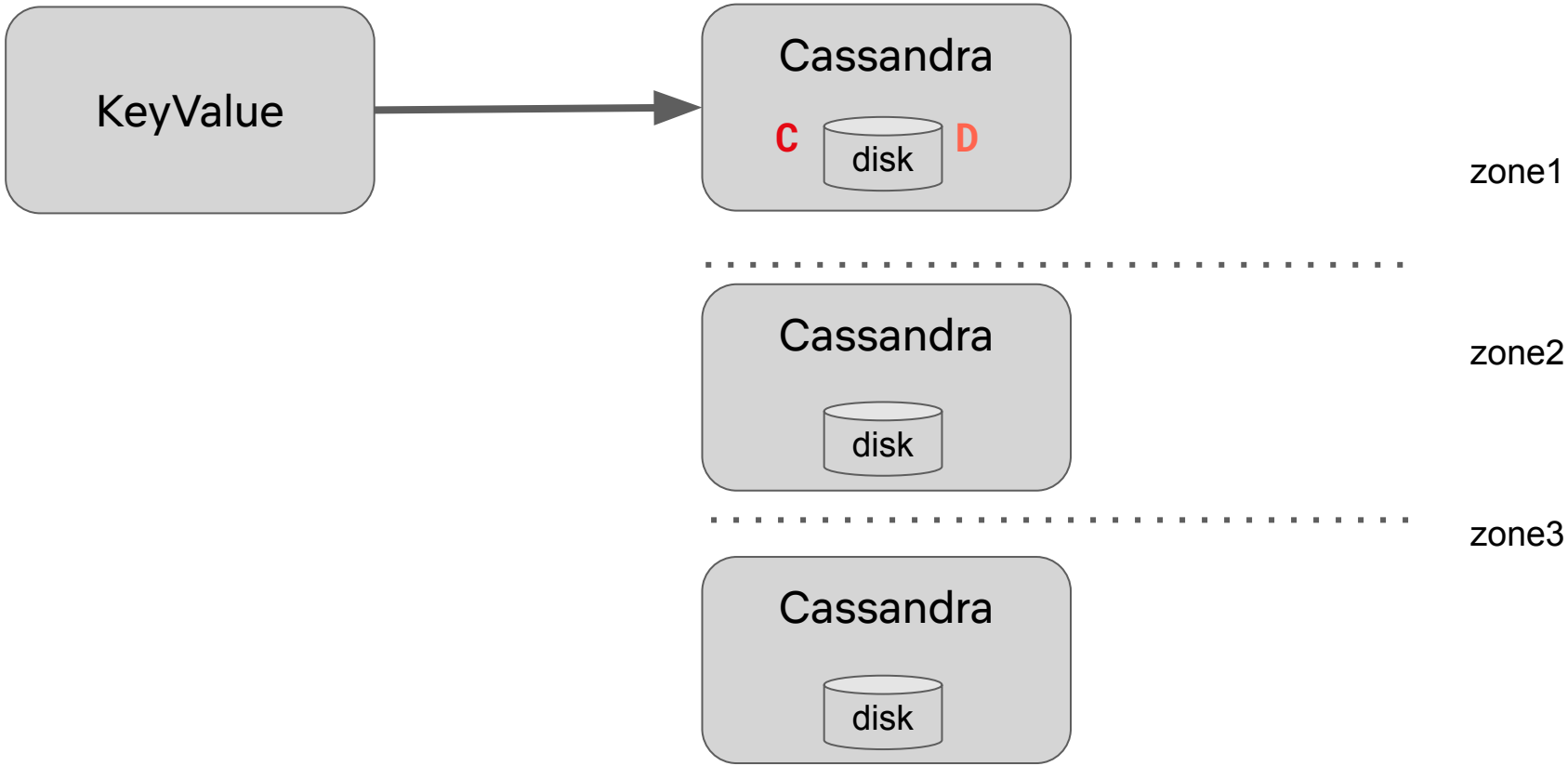
Key-Value
- API
- Storage

Time-Series
- API
- Storage



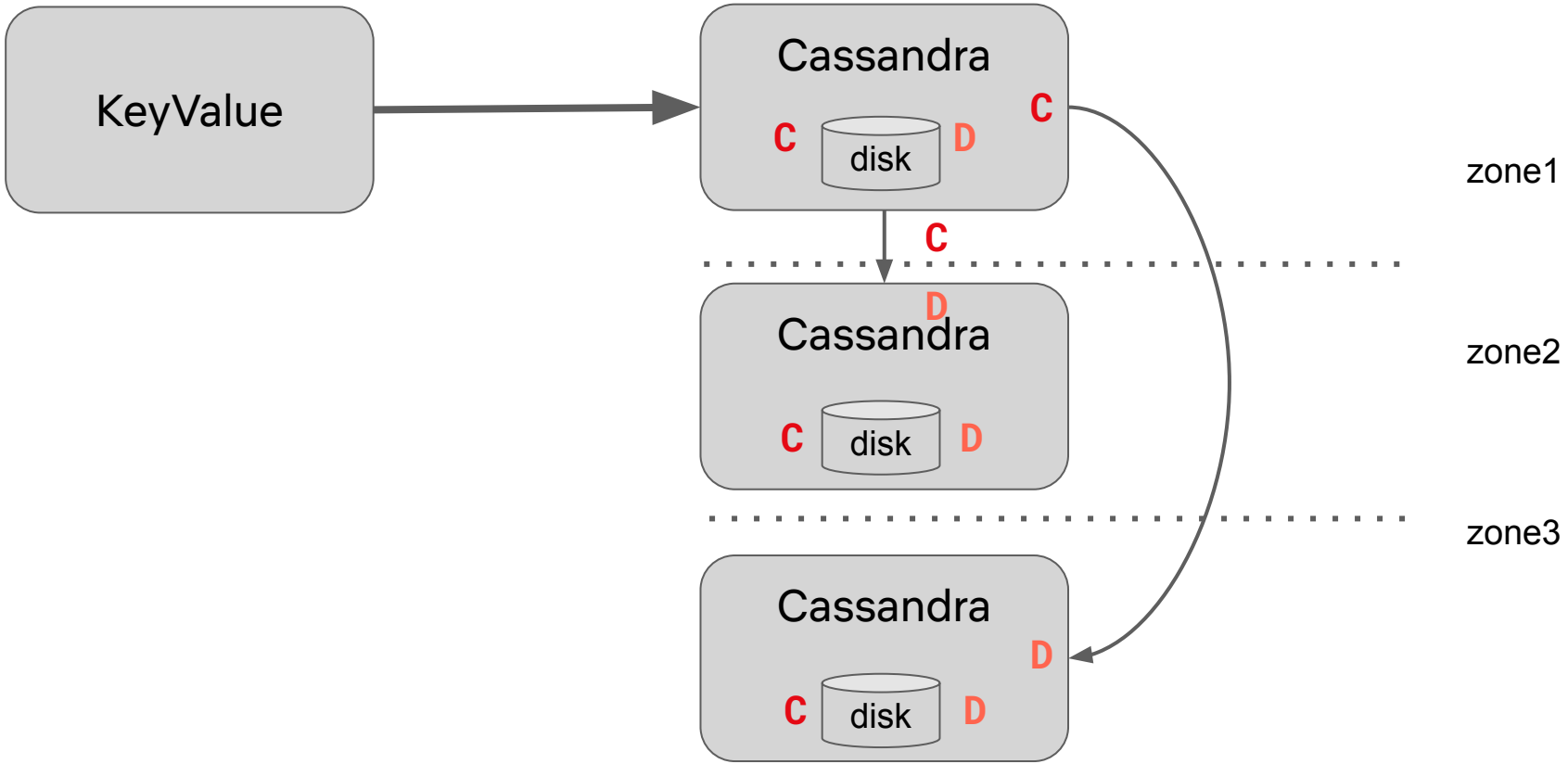
Compression

Server Traditionally Compresses

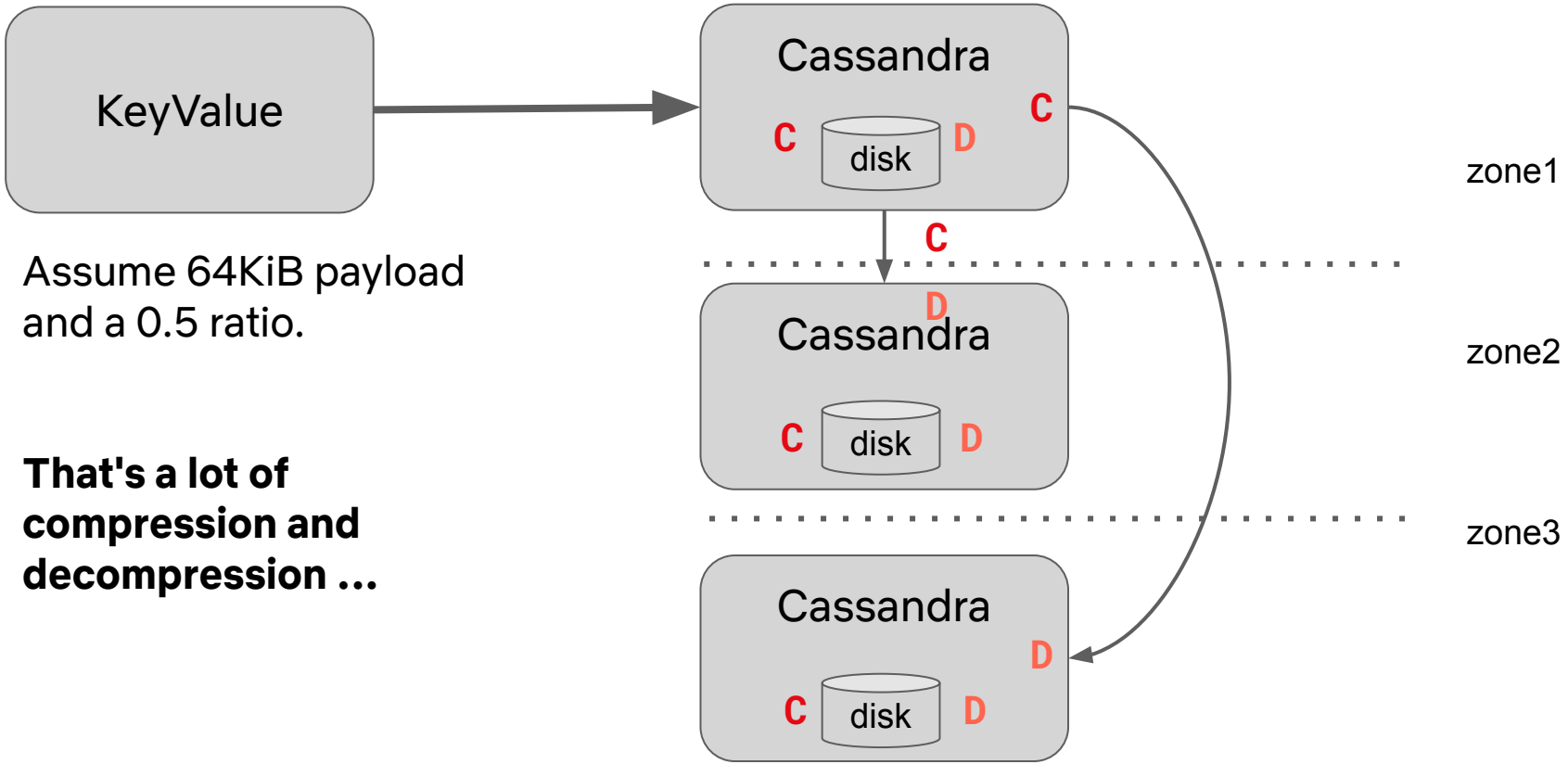


Compression

Server Traditionally Compresses



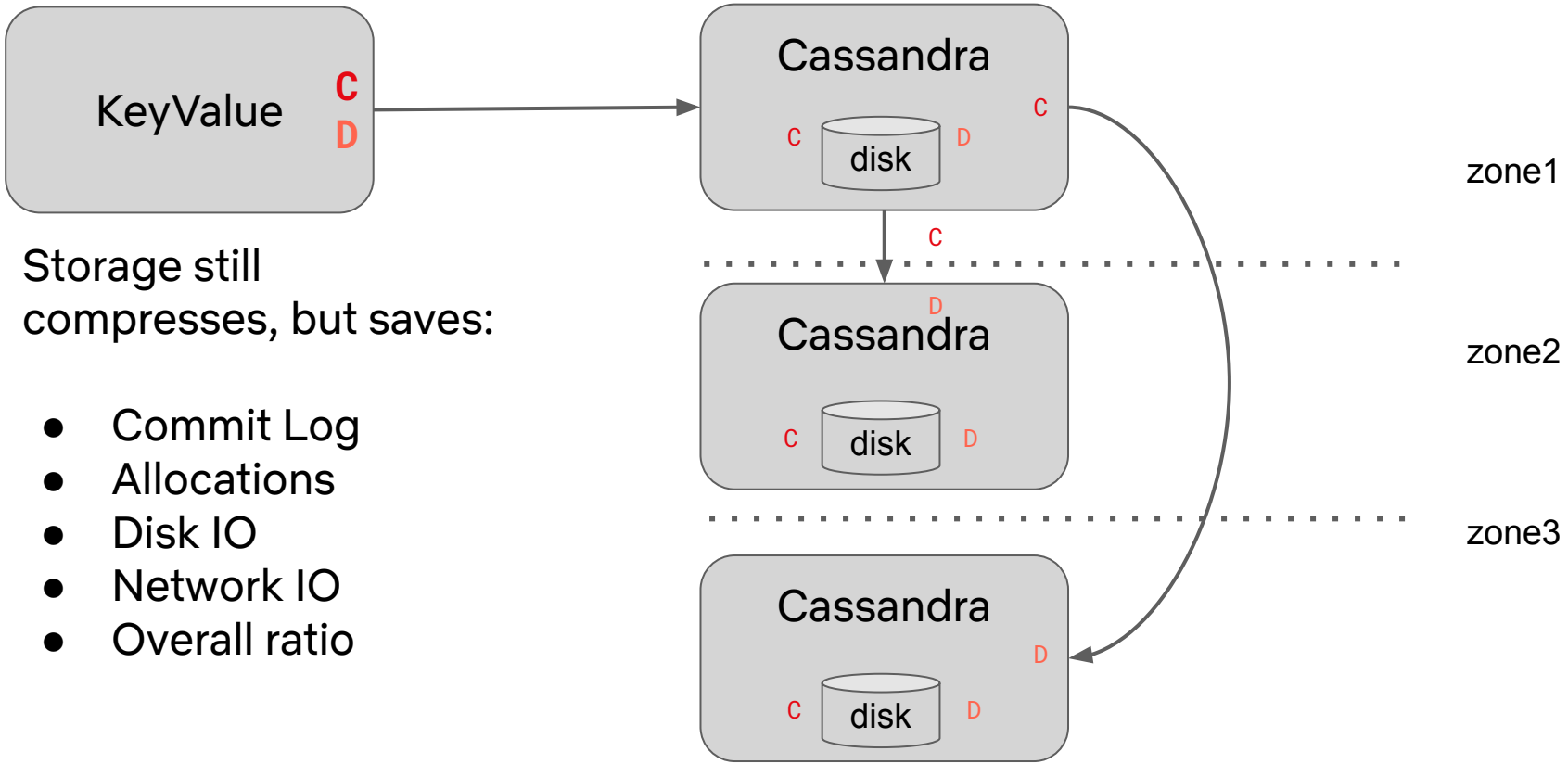
Server Traditionally Compresses



Assume 64KiB payload and a 0.5 ratio.

That's a lot of compression and decompression ...

Better to Compress Chunks at Client

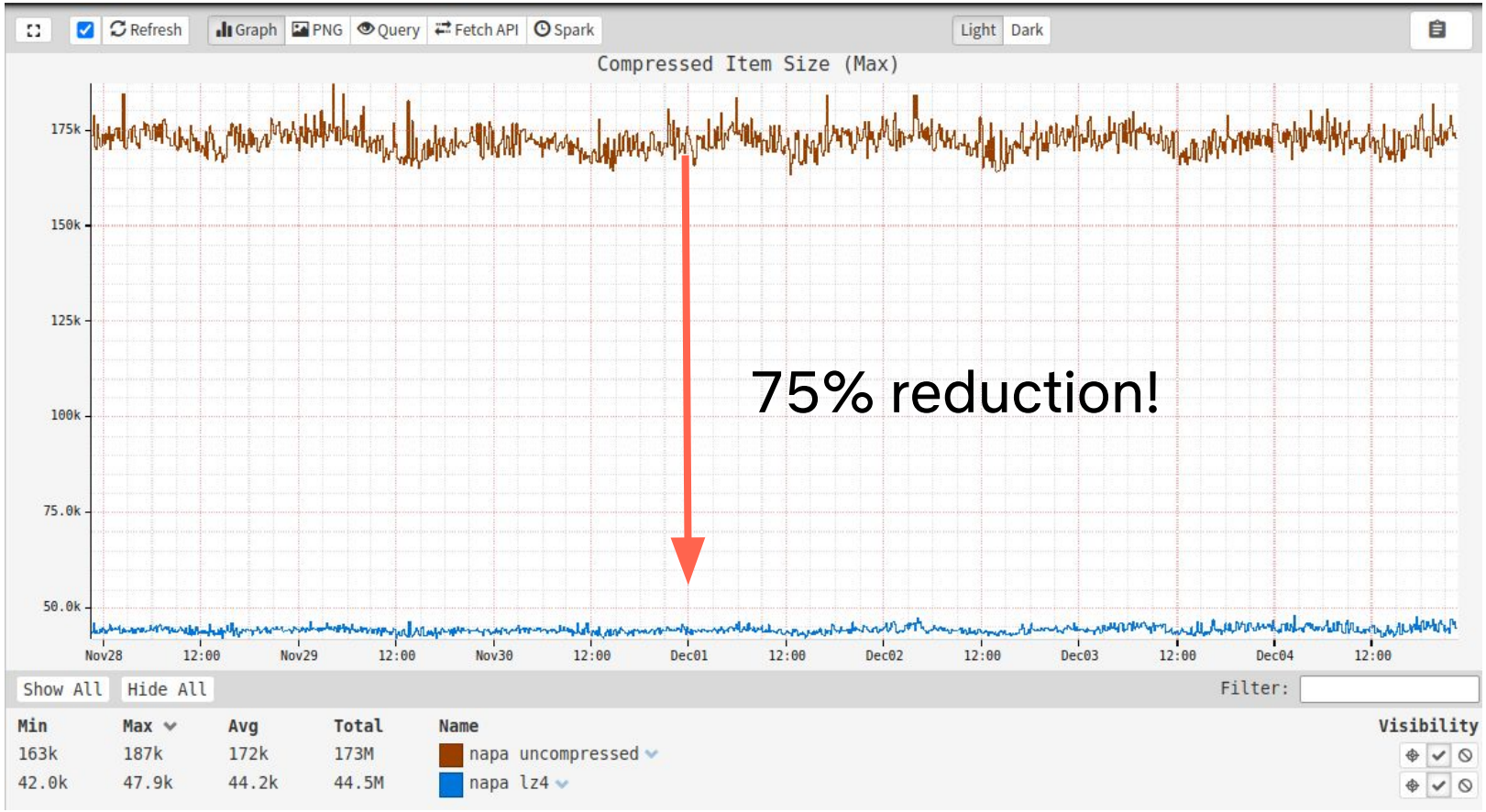


Storage still compresses, but saves:

- Commit Log
- Allocations
- Disk IO
- Network IO
- Overall ratio

Compression

Better to Compress Chunks at Client

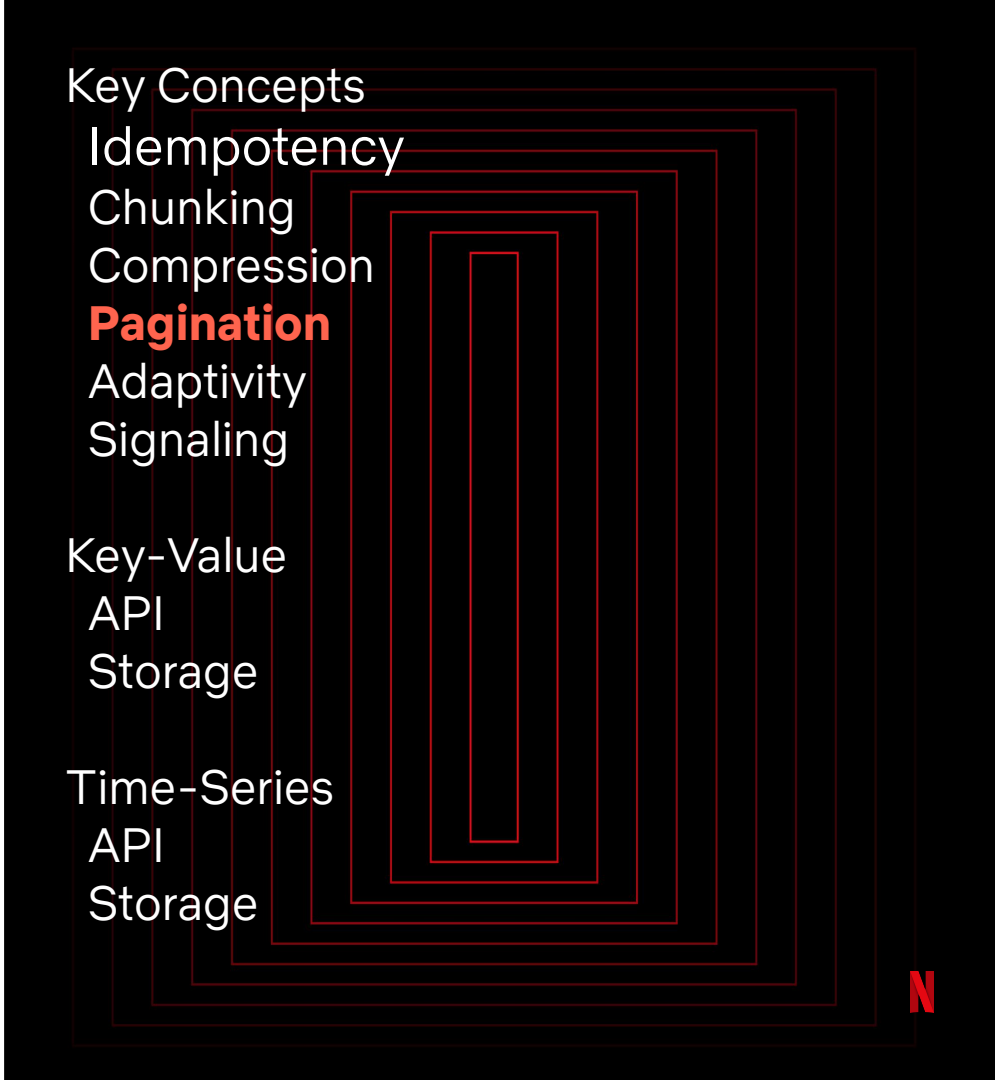


Storage At Any Scale

Key Concepts
- Idempotency
- Chunking
- Compression
- **Pagination**
- Adaptivity
- Signaling

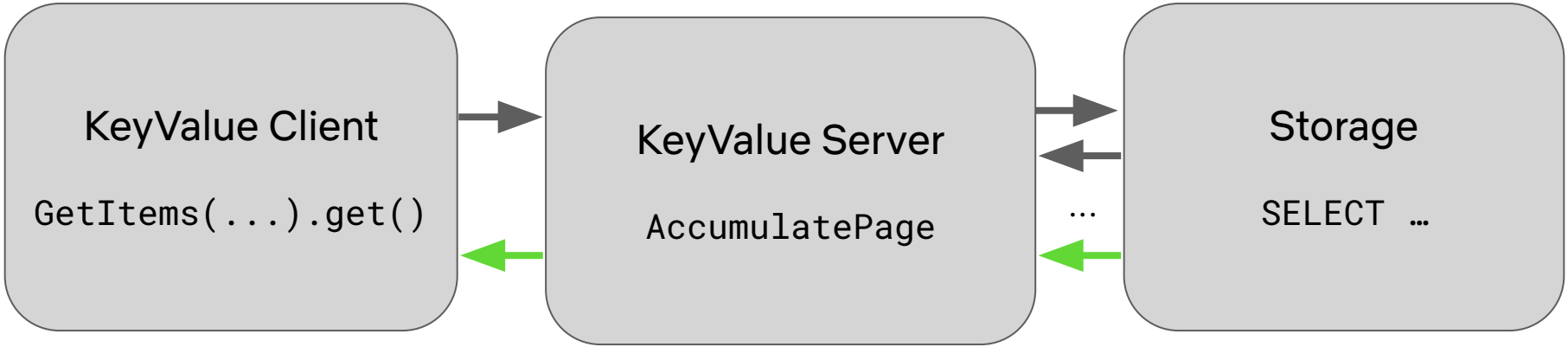
Key-Value
- API
- Storage

Time-Series
- API
- Storage



Pagination

Page Limit of 4MiB




```
{
  "items": [item1, item2, …],
  "next_page_token": <page_token>
}
```

Pagination

Accumulate pages of fixed size work (MiB), clients must ask for more

Storage engines almost always paginate by row count

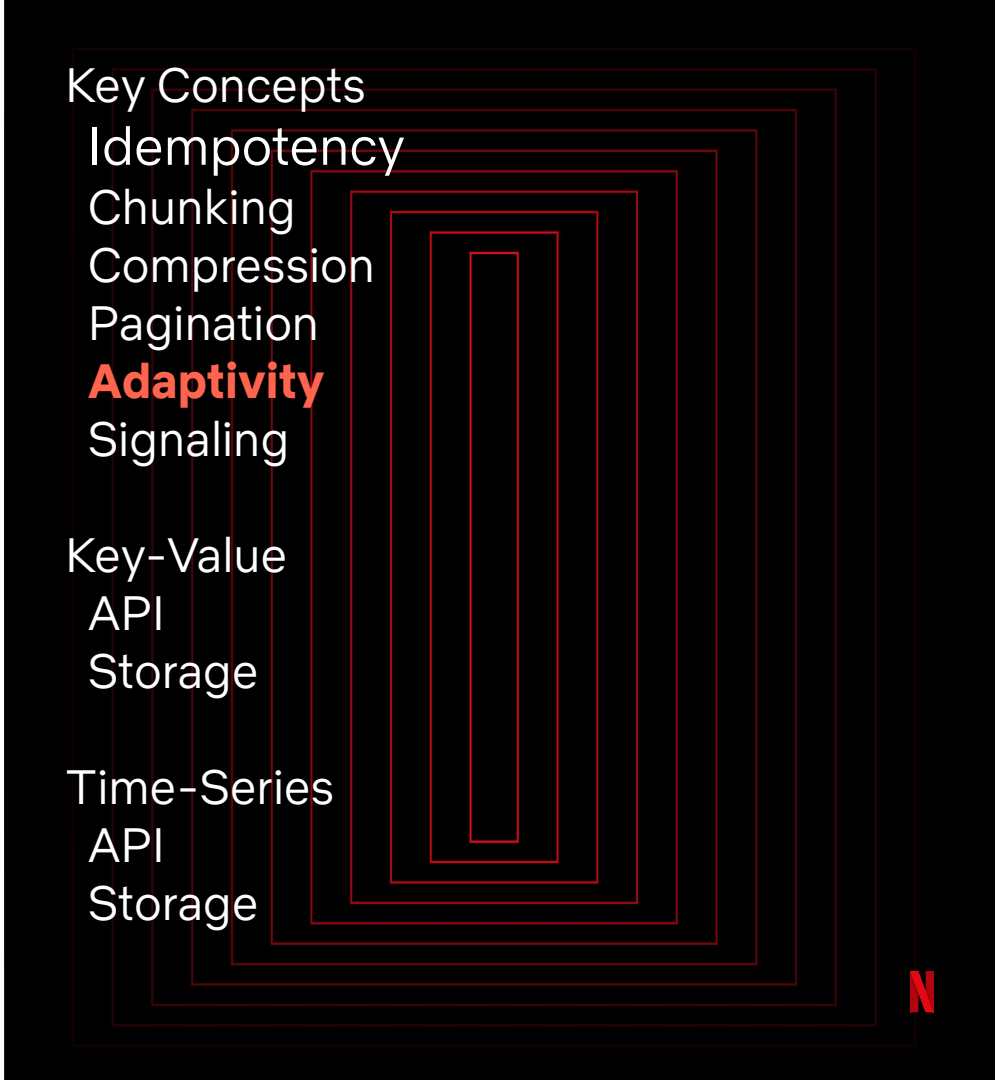
Robust APIs paginate by *size not count*. Translation required.

Storage At Any Scale

Key Concepts
- Idempotency
- Chunking
- Compression
- Pagination
- **Adaptivity**
- Signaling

Key-Value
- API
- Storage

Time-Series
- API
- Storage



Adaptivity:
Target Page size

KeyValue Read Target page size

`HashMap[String, SortedMap[Bytes, Bytes]]`

Server might do many round trips
to DB to accumulate a full page

Adaptivity:
Target Page size

KeyValue Read Target page size

`HashMap[String, SortedMap[Bytes, Bytes]]`

4MiB Page size reached → change the page size via fetch_size to limit DB calls

Adaptivity:
Early SLO Return

KeyValue Read SLO

`HashMap[String, SortedMap[Bytes, Bytes]]`

DB might have more data to fetch but SLO is reached before a page limit is reached

Adaptivity:
Early SLO Return

KeyValue Read SLO

`HashMap[String, SortedMap[Bytes, Bytes]]`

80% SLO_{max} reached → return early

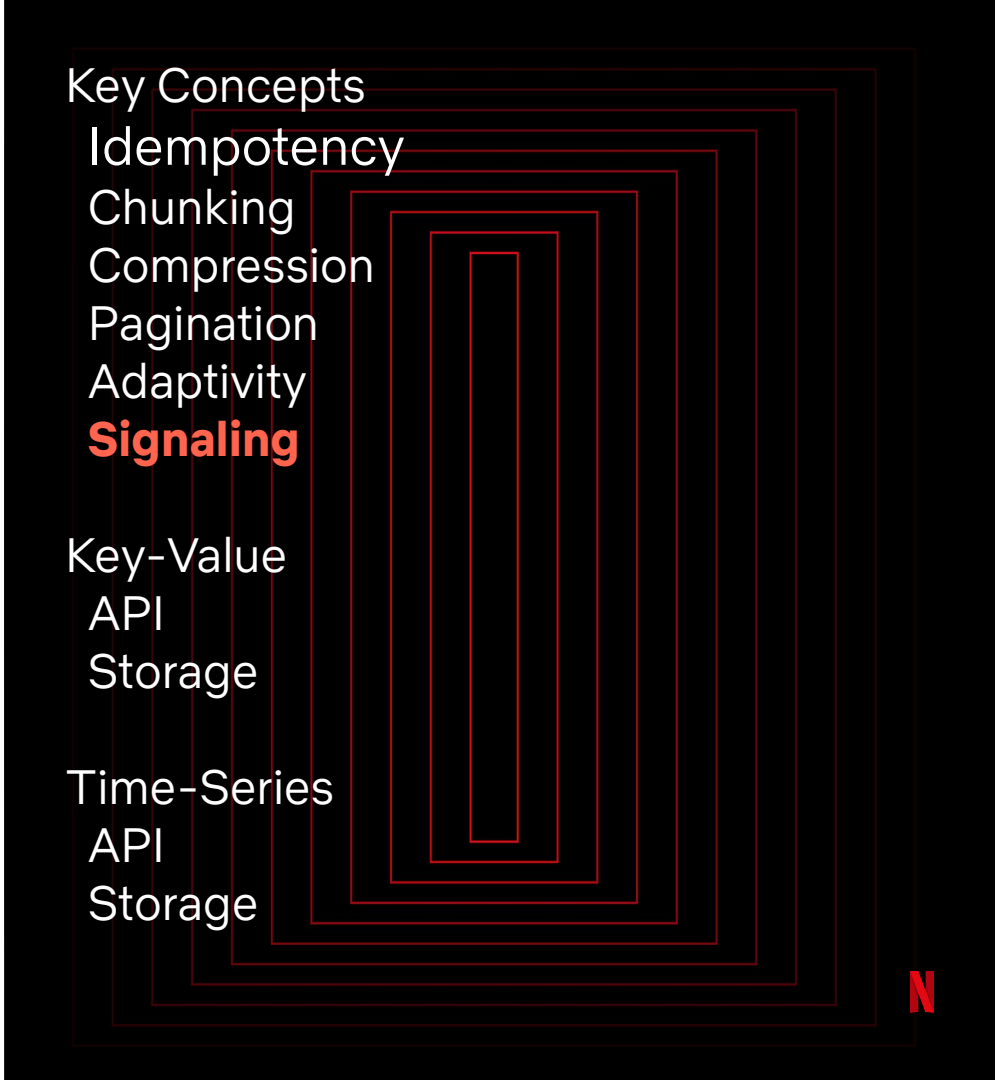
Configuration?

Storage At Any Scale

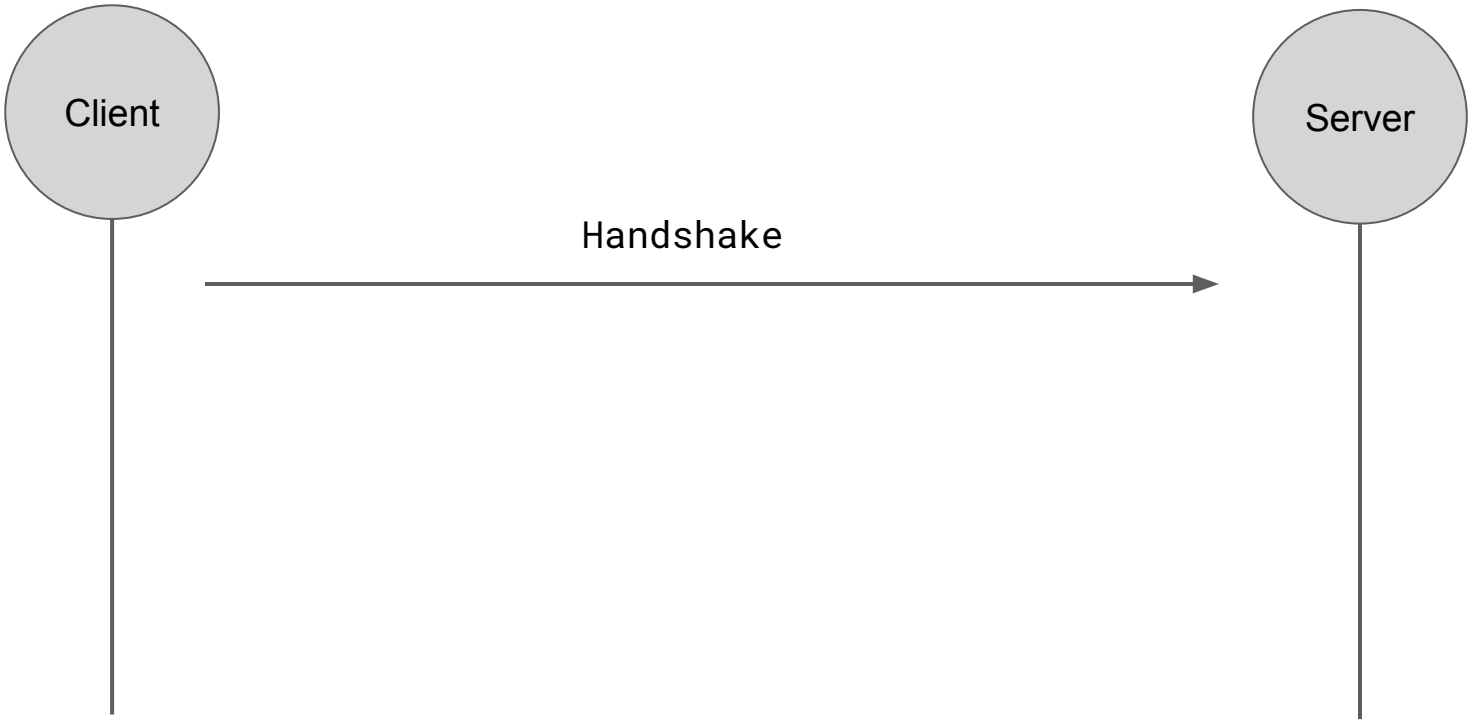
Key Concepts
- Idempotency
- Chunking
- Compression
- Pagination
- Adaptivity
- **Signaling**

Key-Value
- API
- Storage

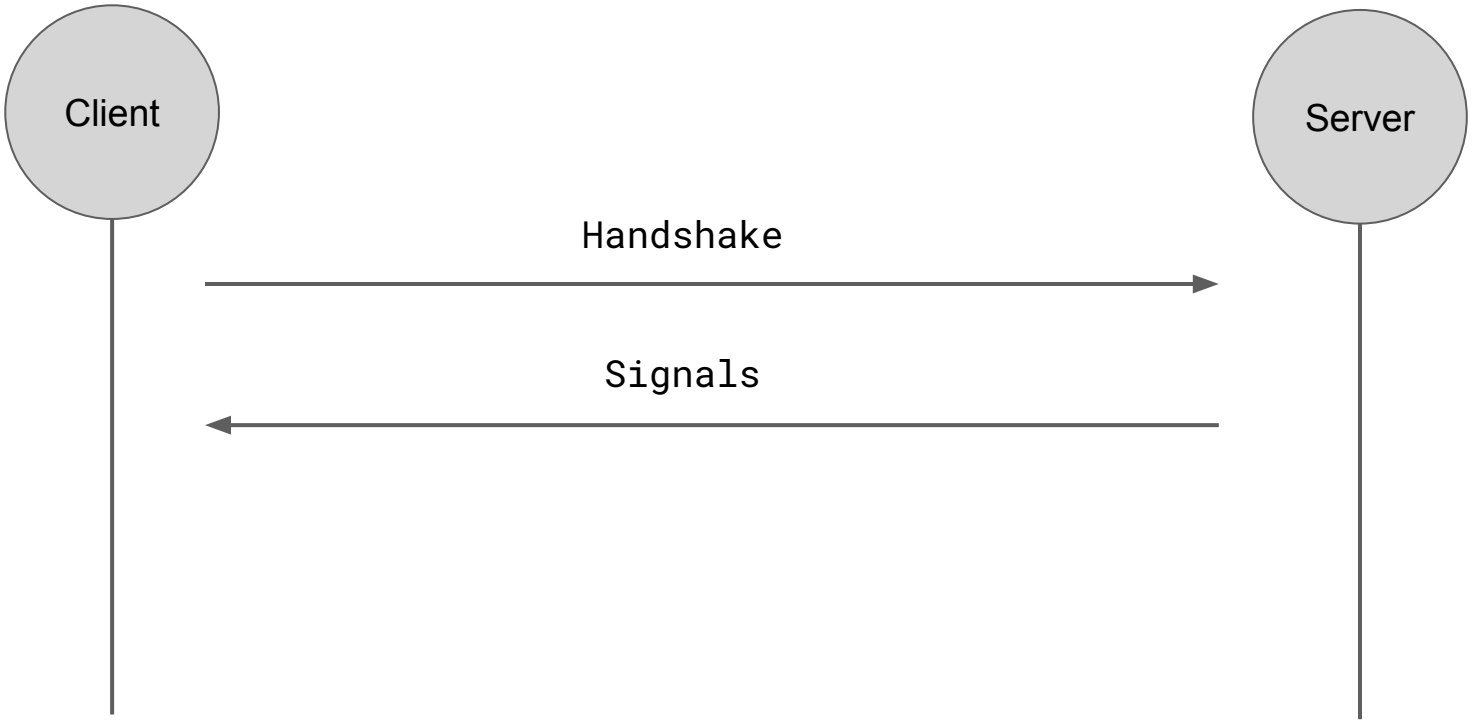
Time-Series
- API
- Storage



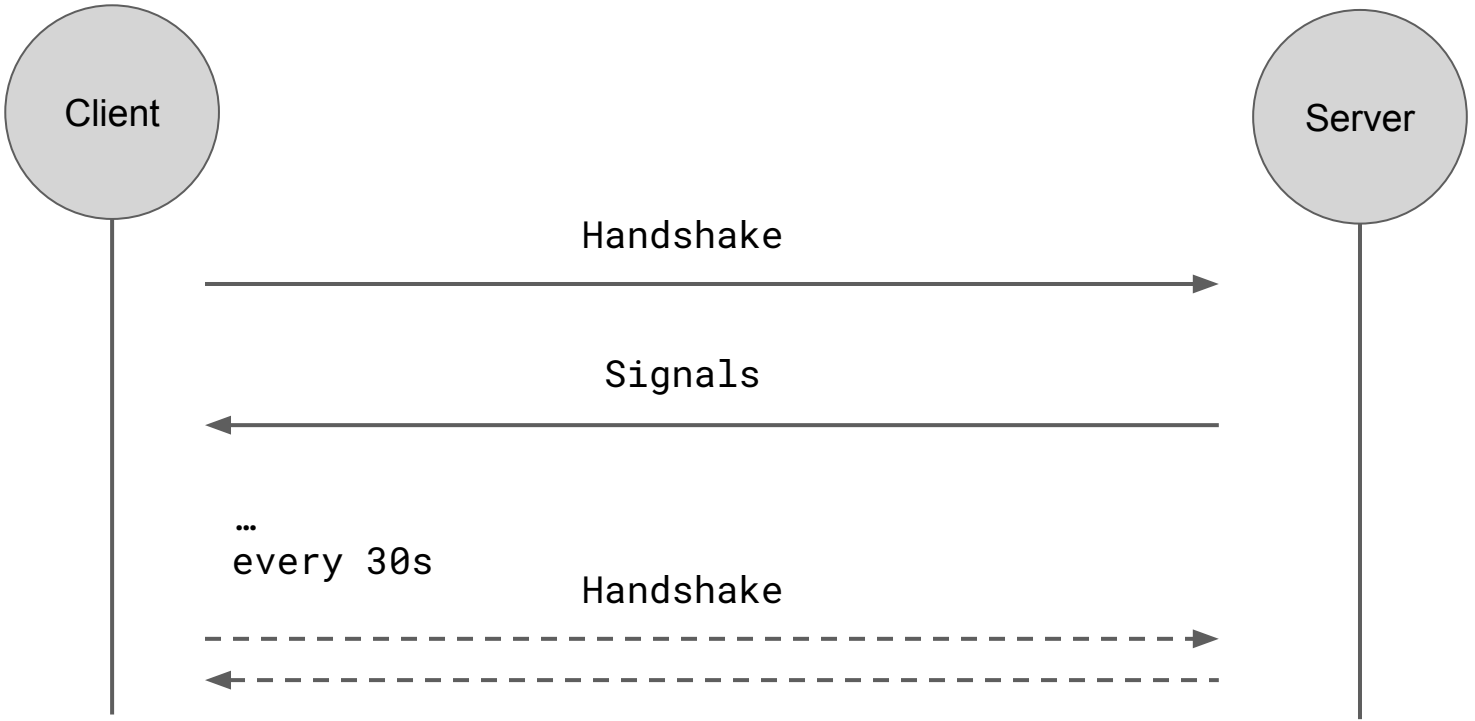
Clients Introspect Servers



Clients Introspect Servers



Clients Introspect Servers



Servers Signal to Clients

Client tuning is error prone
Prefer to send signals from Servers

```
$ grpc -a dgwkv.napa
       -e prod
       -r us-east-1
       com.netflix.dgw.kv.v2.KeyValueServiceV2/Handshake
 | jq .signals
```

```
$ grpc -a dgwts.sessionlogs
       -e prod
       -r us-west-2
       com.netflix.dgw.ts.TimeSeriesService/Handshake
 | jq .signals
```

Servers Signal to Clients

Client tuning is error prone
Prefer to send signals from Servers

```
"payload": {
  "chunked": {
    "chunk-after": 1048576,
    "chunk-size": 65536
  },
  "stats": {
    "record-count": "39371484"
  }
}
```

Clients can transparently chunk payloads to make them fixed amount of work

Large payloads must be broken down into chunks

Servers Signal to Clients

Client tuning is error prone
Prefer to send signals from Servers

```
"compression": [
  {
    "algorithm": "LZ4",
    "parameters": {
      "min-length": 128
    }
  }
]
```

Clients can transparently compress payloads

Fewer bytes = more reliable
(also checksumming)

Servers Signal to Clients

Client tuning is error prone
Prefer to send signals from Servers

```
"slo_by_endpoint": {
  "access": {
    "endpoint": "access",
    "latency": {
      "target": "0.010s",
      "max": "0.500s"
    },
    "concurrency_limits": {
      "normal": 1000,
      "hedge": 50
    }
  }
```

```
"slo_by_endpoint": {
  "access": {
    "endpoint": "access",
    "latency": {
      "target": "0.050s",
      "max": "0.500s"
    },
    "concurrency_limits": {
      "normal": 1000,
      "hedge": 50
    }
  }
```

Article | [The Tail at Scale]

Each Namespace is Different

Lower consistency provides better SLO

```
"namespaces": {
  "eventual_ns": {
    "consistency_scope":
      "LOCAL",
    "consistency_target":
      "EVENTUAL",
    …
    "physical_storage": {
       "type": "CASSANDRA",
       "cluster": "clstr1",
       "dataset": "ks1",
       "table": "tbl1",
    }
```

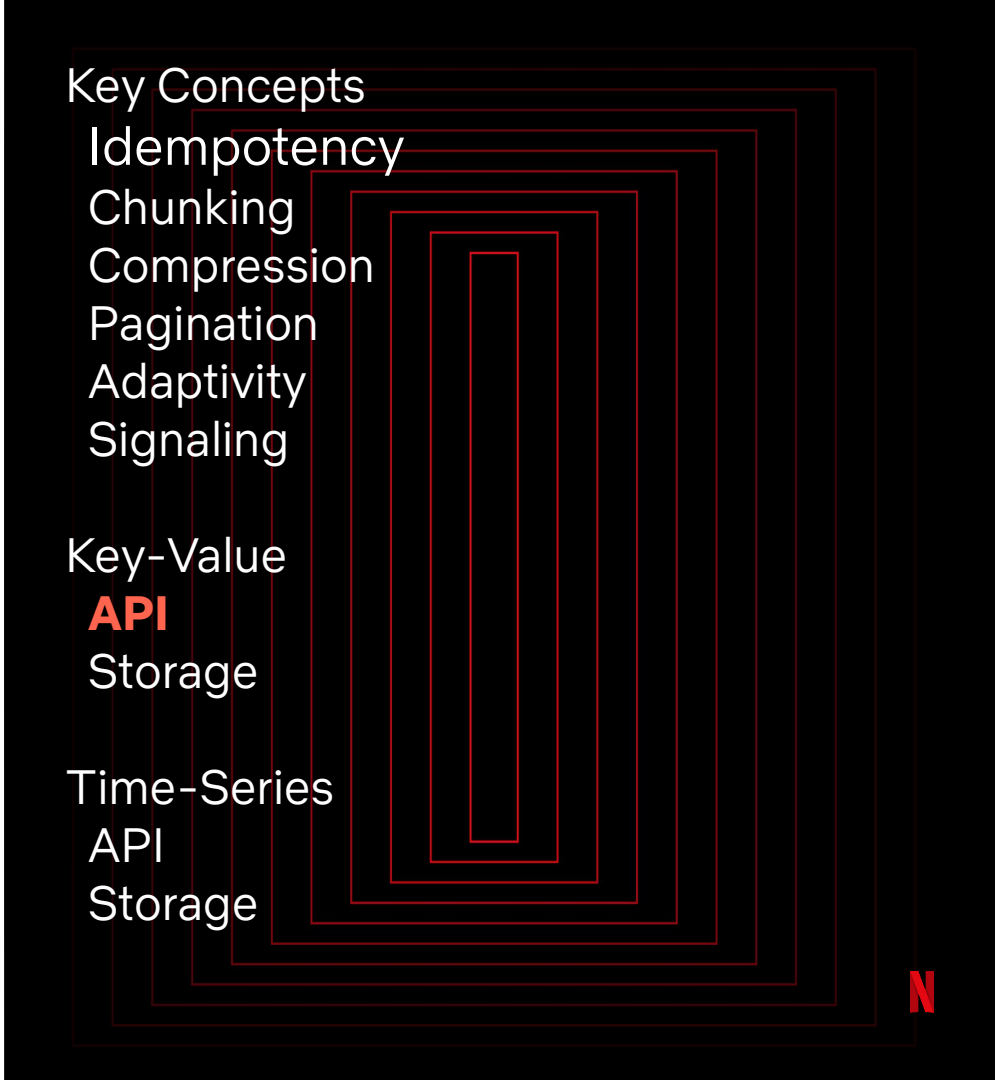
```
"namespaces": {
  "strong_ns": {
    "consistency_scope":
      "GLOBAL",
    "consistency_target":
      "READ_YOUR_WRITES",
    …
    "physical_storage": {
       "type": "CASSANDRA",
       "cluster": "clstr1",
       "dataset": "ks1",
       "table": "tbl1",
    }
```

Storage At Any Scale

Key Concepts
- Idempotency
- Chunking
- Compression
- Pagination
- Adaptivity
- Signaling

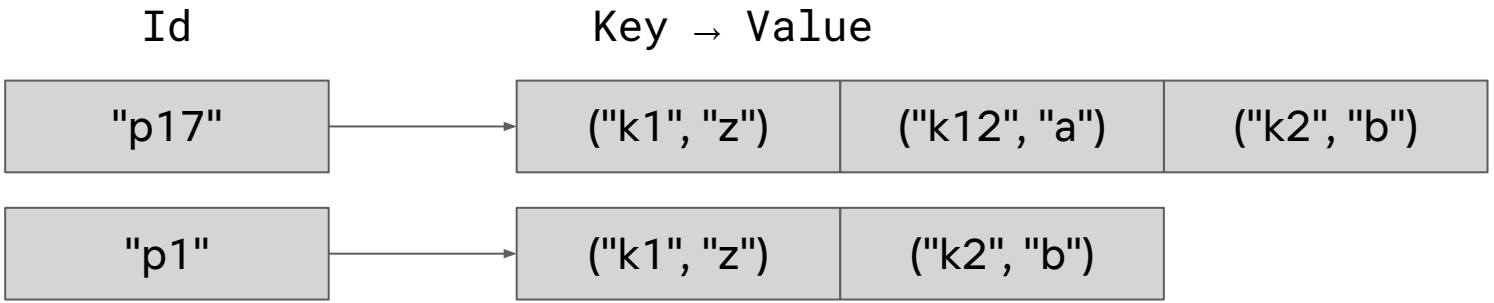
Key-Value
- **API**
- Storage

Time-Series
- API
- Storage



Key-Value Abstraction APIs

```
Server gRPC Abstraction API
Map[String, List[Item]]

Client High-Level API
HashMap[String, SortedMap[Bytes, Bytes]]
```



KeyValue HashMap[String, SortedMap[Bytes, Bytes]]

```
rpc PutItems(
  # Allows retry
  IdempotencyToken token,
  # Primary key
  String namespace,
  String id,
  # Key-Value pairs
  List<Item> items
) -> PutItemsResponse
```

```
PutItemsResponse {
  Trilean durable;
  Trilean visible; …
}

Item {
   bytes key;
   bytes value;
   # Zero implies commit
   int32 chunk;
}
```

KeyValue HashMap[String, SortedMap[Bytes, Bytes]]

```
rpc MutateItems(
  IdempotencyToken token,
  String namespace,
  String id,
  List[MutationRequest]
    mutations, …
) -> MutationResponse
```

```
MutationRequest {
 oneof {
   PutItems    put;
   DeleteItems delete;
 }
}
```

KeyValue HashMap[String, SortedMap[Bytes, Bytes]]

```
rpc GetItems(
  # Primary key
  String namespace,
  String id,

  # What matches
  Predicate predicate,

  # Pagination/Selection
  Selection selection
) -> GetItemsResponse
```

```
GetItemsResponse {
  # Page of results
  # 1 MiB pages
  List[Item] items

  # If set there is
  # more data to read
  String next_page
}
```

KeyValue

HashMap[String, SortedMap[Bytes, Bytes]]

```
rpc ScanItems(
  # Table name
  String namespace,

  # What matches
  Predicate predicate,

  # Pagination & selection
  Selection selection
) -> ScanItemsResponse
```

```
ScanItemsResponse {
  # Page of results
  List[ScanResult] results;

  # Multiple concurrent
  # pages consumed
  List[String] next_page;
}
```

KeyValue `HashMap[String, SortedMap[Bytes, Bytes]]`

Writes

- All writes sent with **IdempotencyToken** so we can deduplicate
- Large values **chunked**, staged, and finally committed

Reads

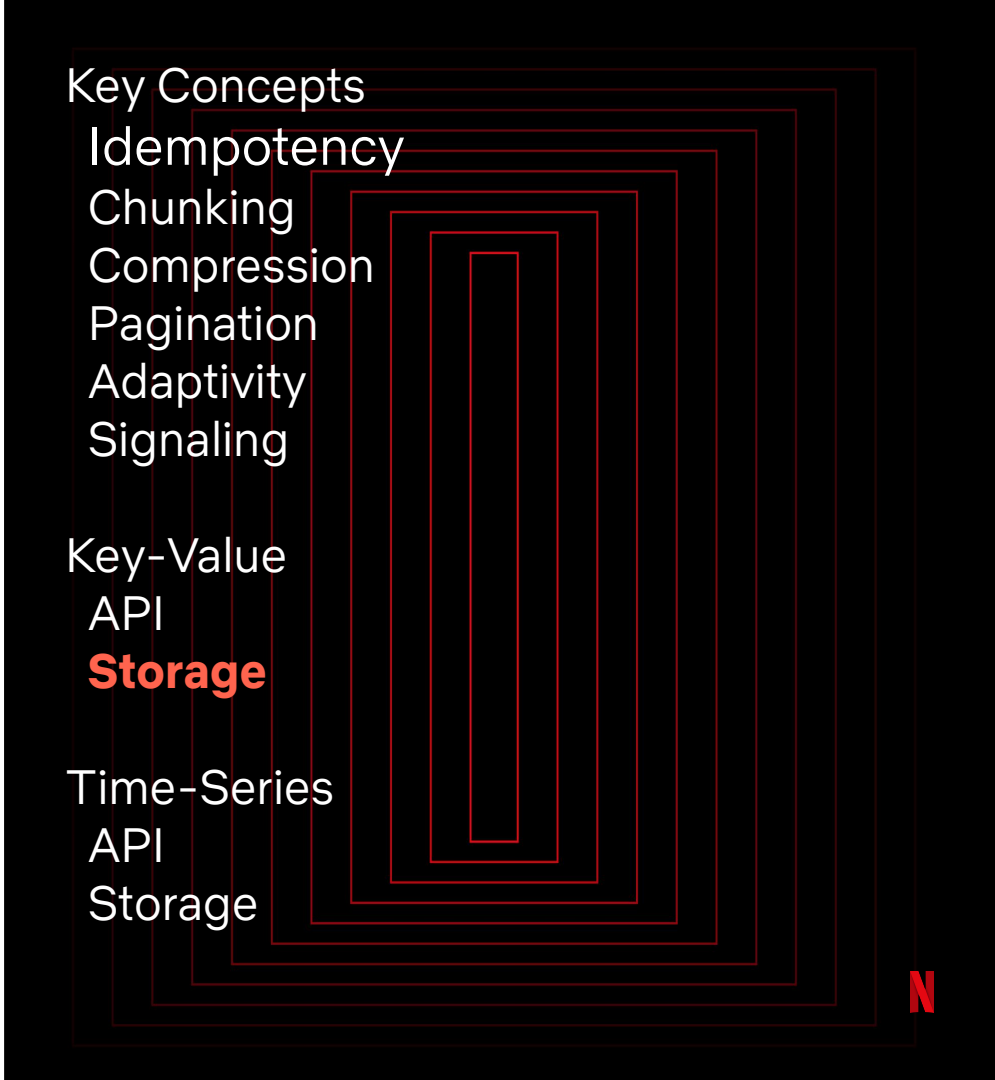
- All **reads return pages** within the SLO
- No SLO across all pages
- **Fixed *size* work**, not count!

Storage At Any Scale

Key Concepts
- Idempotency
- Chunking
- Compression
- Pagination
- Adaptivity
- Signaling

Key-Value
- API
- **Storage**

Time-Series
- API
- Storage



Key-Value Storage Layout

Base Table

id	key	value	value_metadata
1234	key1	value1	{"metadata": …}
1234	key2	<empty>	{"metadata": … }

```
CREATE TABLE IF NOT EXISTS <ns>.<ns> (
  id                text,
  key               blob,
  value             blob,
  value_metadata    blob,
PRIMARY KEY (id, key))
WITH CLUSTERING ORDER BY (key <ASC|DESC>)
  AND compaction  = LCS(256MiB)
  AND compression = LZ4(16KiB)
```

Key-Value Storage Layout

Data Table

id	bucket	key	chunk	version	value	value_metadata
1234	64200	key2	1	timeuuid1	chunk1	{ “metadata”: … }
1234	64201	key2	2	timeuuid1	chunk2	{ “metadata”: … }
1234	64201	key2	3	timeuuid1	chunk2	{ “metadata”: … }

```
CREATE TABLE IF NOT EXISTS <ns>.data_<ns> (
  id              text,
  bucket          int,
  key             blob,
  chunk           int,
  version         timeuuid,
  value           blob,
  value_metadata  blob,
PRIMARY KEY ((id, bucket), key, version, chunk))
WITH CLUSTERING ORDER BY (key <ASC|DESC>, version DESC, chunk ASC)
```

Data Model

Key-Value bucketing per key

id	key	value	value_metadata
1234	**key1**	<empty>	chunk_count: 2 **chunk_offset: 64201**
6789	**key2**	<empty>	chunk_count: 3 **chunk_offset: 34001**
12	**key1**	"small"	{}

id	bucket	key	chunk	version	value	value_metadata
1234	**64201**	key1	1	timeuuid1	chunk value 1	{ “metadata”: … }
1234	**64201**	key1	2	timeuuid1	chunk value 2	{ “metadata”: … }
3443	64202	key1	1	timeuuid3	chunk value 1	{ “metadata”: … }
6789	**34001**	key2	1	timeuuid2	chunk value 1	{ “metadata”: … }
6789	**34001**	key2	2	timeuuid2	chunk value 2	{ “metadata”: … }
6789	**34001**	key2	3	timeuuid2	chunk value 3	{ “metadata”: … }

```
"chunks": {
    "chunk_size_bytes": 65536,
    "chunk_count": 2,
    "chunk_offset": 124431,
    "chunk_token_xxh64": "12595968139585192830",
    "version": "3d52c5c0-6229-11ee-aecd-de49621d037e"
 }
```

Chunking:
Spread the Load

```
offset = consistentHash(xxHash(key), NUM_BUCKETS)
bucket = offset + (chunk / 8)
```

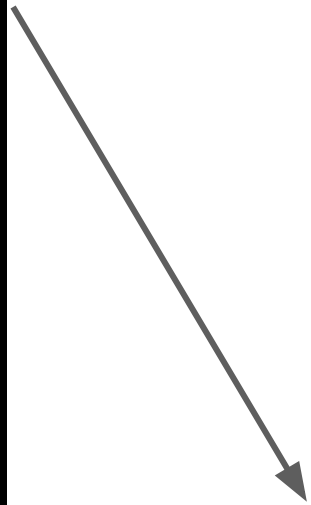
```
( id, key) -> value


("1", "a") -> 1MiB

("1", "b") -> 2MiB

("2", "c") -> 10MiB
```

```
((id, bucket), key, chunk)
```

Chunking:
Spread the Load

```
offset = consistentHash(xxHash(key), NUM_BUCKETS)
bucket = offset + (chunk / 8)
```

```
( id, key) -> value

("1", "a") -> 1MiB

("1", "b") -> 2MiB

("2", "c") -> 10MiB
```

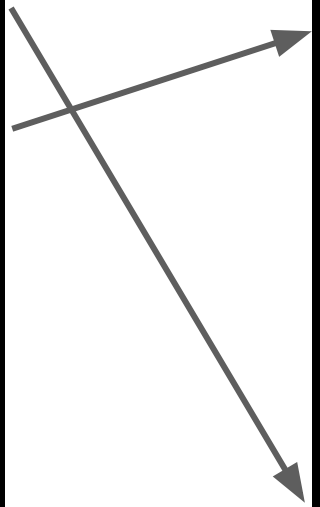


```
((id, bucket), key, chunk)

(("1",    1072), "a",      1)
…
(("1",    1074), "a",      16)
```

Chunking: Spread the Load

```
offset = consistentHash(xxHash(key), NUM_BUCKETS)
bucket = offset + (chunk / 8)
```

```
( id, key) -> value

("1", "a") -> 1MiB

("1", "b") -> 2MiB

("2", "c") -> 10MiB
```

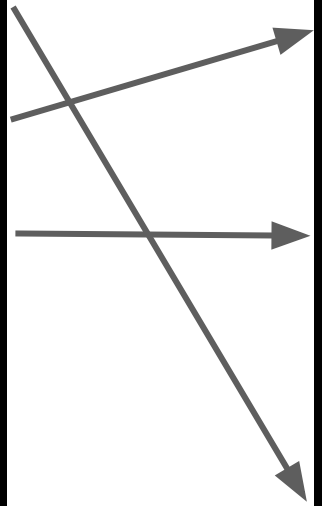


```
((id, bucket), key, chunk)

(("1",       2), "b",      1)
…
(("1",       6), "b",      32)



(("1",    1072), "a",      1)
…
(("1",    1074), "a",      16)
```

Chunking:
Spread the Load

```
offset = consistentHash(xxHash(key), NUM_BUCKETS)
bucket = offset + (chunk / 8)
```

```
( id, key) -> value

("1", "a") -> 1MiB

("1", "b") -> 2MiB

("2", "c") -> 10MiB
```

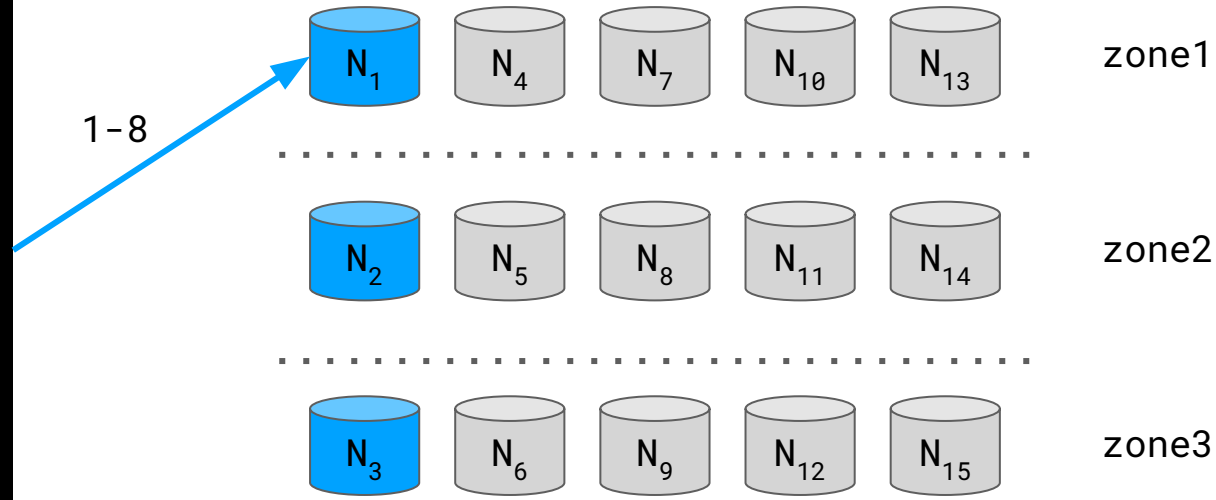


```
((id, bucket), key, chunk)

(("1",       2), "b",      1)
…
(("1",       6), "b",     32)

(("2",      12), "c",      1)
…
(("2",      32), "c",    160)


(("1",    1072), "a",      1)
…
(("1",    1074), "a",     16)
```

Chunking:
Spread the Load

Chunks Distributed Across the Cluster

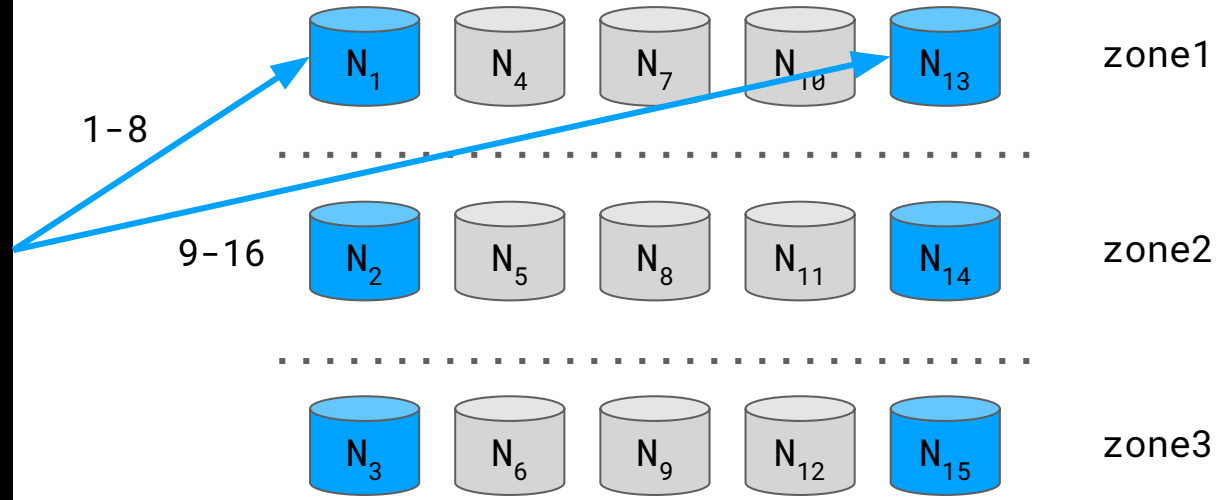
```
( id, key) -> value


("2", "c") -> 10MiB
```



Chunking:
Spread the Load

Chunks Distributed Across the Cluster

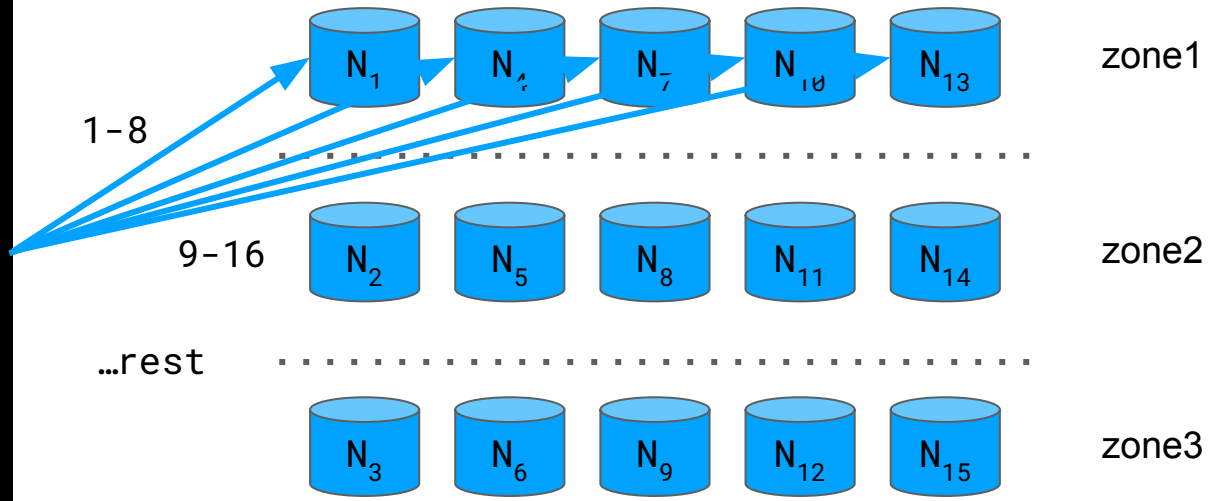
```
( id, key) -> value


("2", "c") -> 10MiB
```



Chunking:
Spread the Load

Chunks Distributed Across the Cluster

```
( id, key) -> value


("2", "c") -> 10MiB
```



Versioning

Key-Value concurrency through versions

id	key	value	value_metadata
1234	**key1**	<empty>	{**version**: **v2**}
6789	**key2**	<empty>	{version: timeuuid()}

id	bucket	key	chunk	version	value	value_metadata
1234	64201	**key1**	1	**V1**	chunk value 1	{ “metadata”: … }
1234	64201	**key1**	2	**V1**	chunk value 2	{ “metadata”: … }
1234	34001	**key1**	1	**V2**	chunk value 1	{ “metadata”: … }
1234	34001	**key1**	2	**V2**	chunk value 2	{ “metadata”: … }
1234	34002	**key1**	3	**V2**	chunk value 3	{ “metadata”: … }

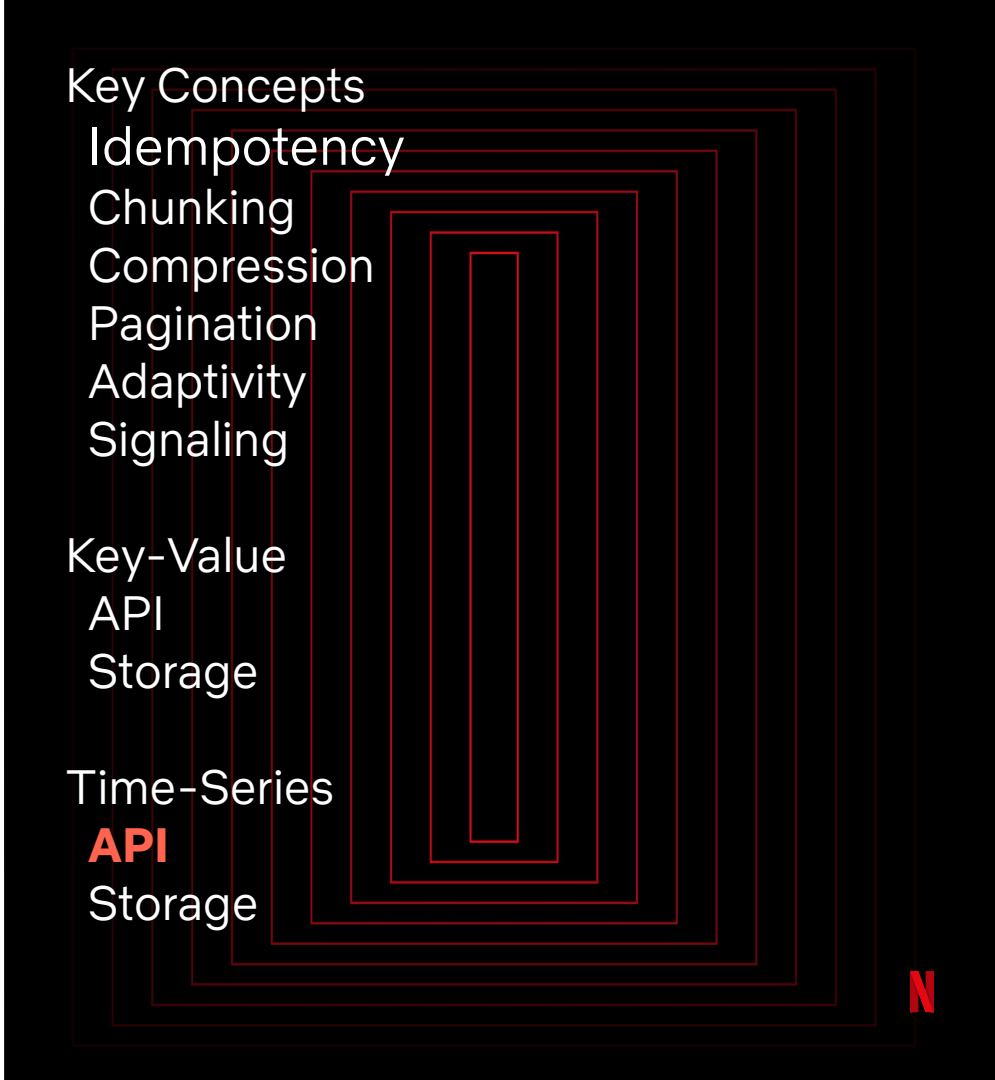
```
"chunks": {
    "chunk_size_bytes": 65536,
    "chunk_count": 2,
    "chunk_offset": 124431,
    "chunk_token_xxh64": "12595968139585192830",
    "version": "3d52c5c0-6229-11ee-aecd-de49621d037e"
 }
```

Storage At Any Scale

Key Concepts
- Idempotency
- Chunking
- Compression
- Pagination
- Adaptivity
- Signaling

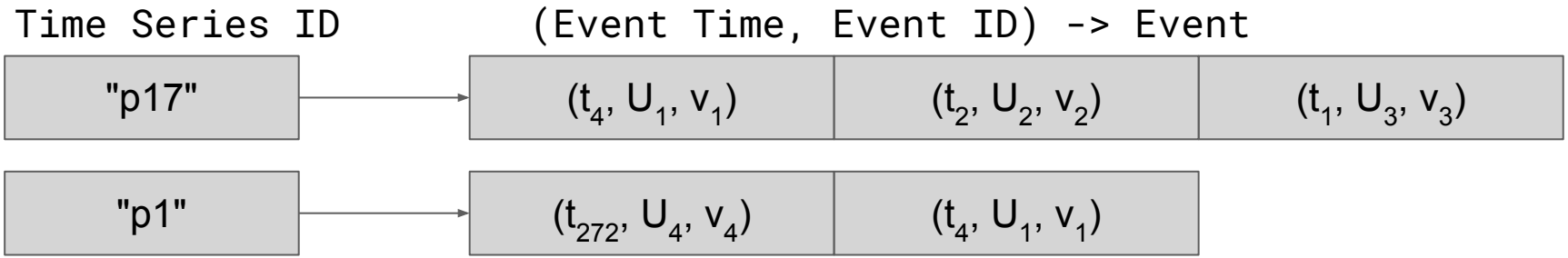
Key-Value
- API
- Storage

Time-Series
- **API**
- Storage



TimeSeries Abstraction APIs

```
Server gRPC Abstraction API
Map[String, List[EventRecord]]

Client High-Level API
HashMap[String,
  SortedMap[[Timestamp, UUID], Event]
```



TimeSeries HashMap[String,
SortedMap[[Timestamp, String], Event]

```
rpc WriteEventRecords[Sync](
  List[EventRecord] events
) -> (Durable, Visible)

message EventRecord {
  String namespace,
  String time_series_id,
  Timestamp time,
  String event_id,
  List[EventItem] items,
}
```

```
rpc ReadEventRecords(
  String namespace,
  String time_series_id,
  Interval time_interval,
) -> Page

rpc SearchEventRecords(
  String namespace,
  SearchQuery query
) -> Page
```

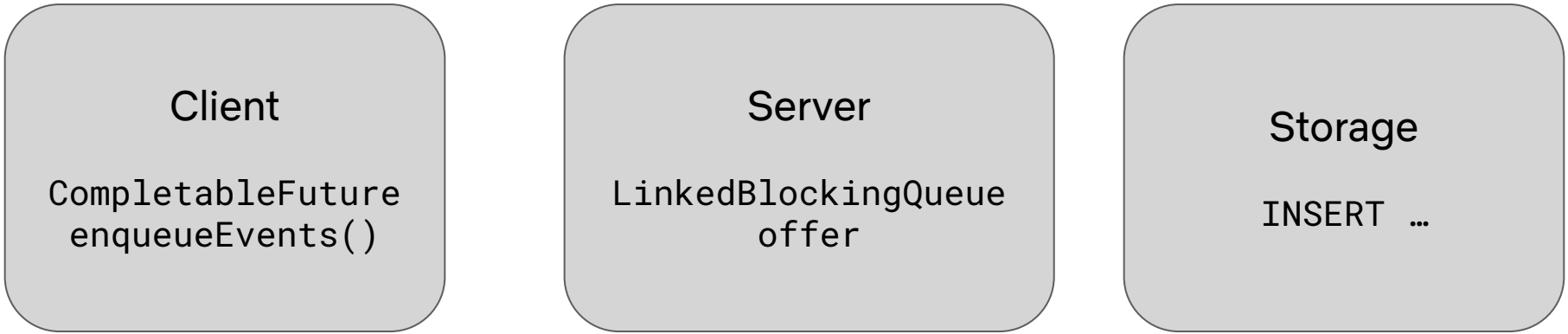
TimeSeries HashMap[String, SortedMap[[Timestamp, String], Event]

```
message EventRecord {
  String namespace,
  String time_series_id,
  # Idempotency Token
  Timestamp time,
  String event_id,
  List[EventItem] items,
}
```

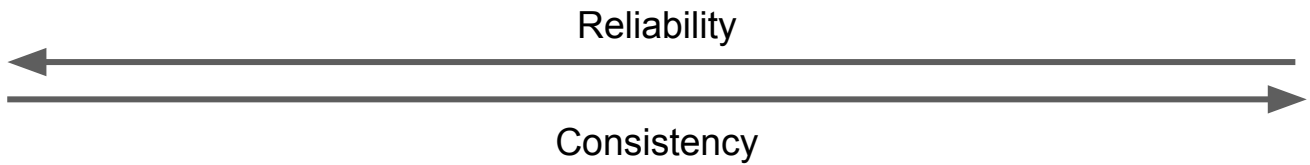
```
message
ReadEventRecordResponse {
  StringValue
    next_page_token;

  # The page of results
  List[EventRecord] records;
}
```

TimeSeries Three-Ack Options

```
HashMap[String,
  SortedMap[[Timestamp, String], Event]
```



1. Fire and Forget
2. Enqueued
3. Durable



TimeSeries `HashMap[String, SortedMap[[Timestamp, String], Event]`

Writes

- All writes sent with components to construct an **IdempotencyToken**
- Three modes of acknowledging
- Large events prohibited (>4MiB)

Reads

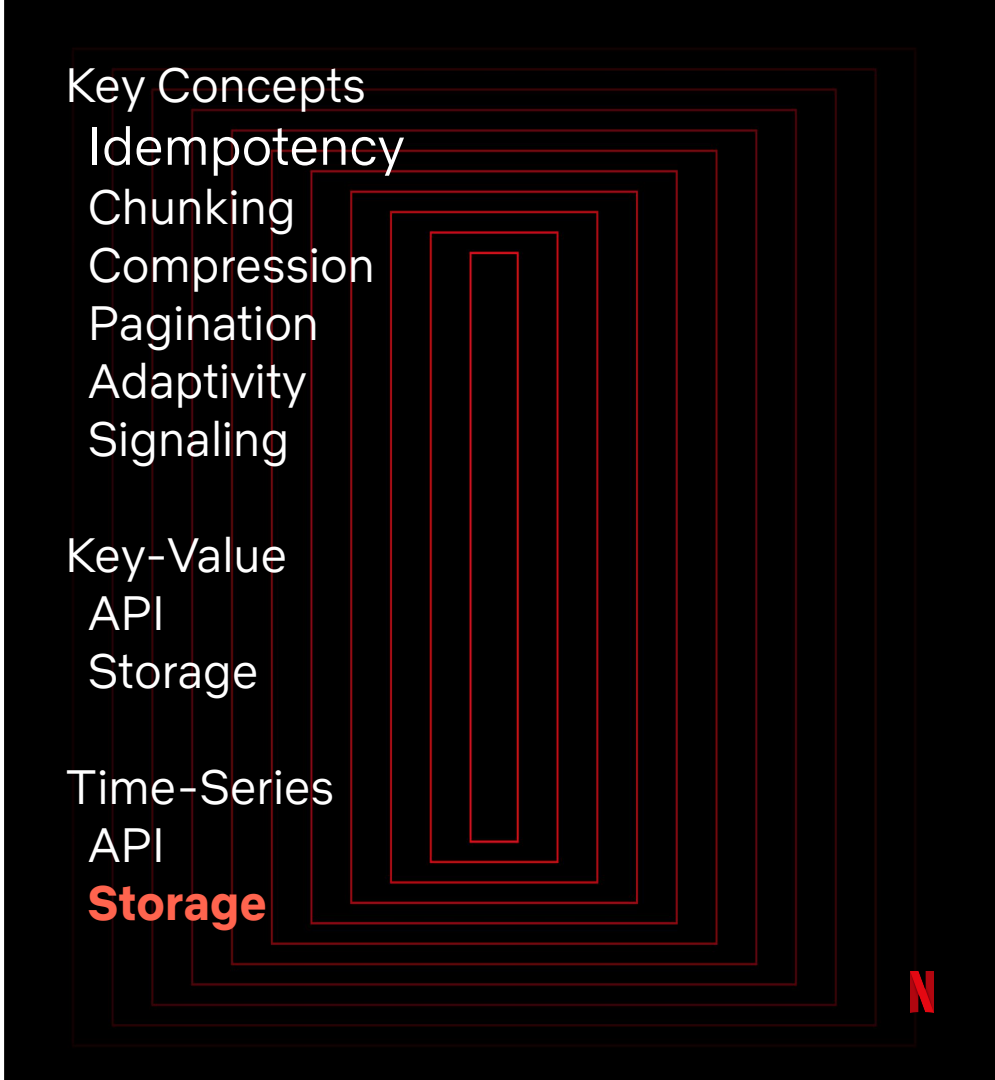
- All **reads return pages** of events within the SLO
- No SLO across all pages
- **Fixed *size* work**, not count!

Storage At Any Scale

Key Concepts
- Idempotency
- Chunking
- Compression
- Pagination
- Adaptivity
- Signaling

Key-Value
- API
- Storage

Time-Series
- API
- **Storage**



Why take on the Complexity?

Save millions of dollars on computers

Flexible operations

Simplify massive scale event datasets for developers

Why take on the Complexity?

Save millions of dollars on computers
(compute efficiency)

Flexible operations
(operator efficiency)

Simplify massive scale event datasets for developers
(developer efficiency)

Time-Series Storage Layout

Metadata table

namespace name	version	start time	end time	metadata	status
ns1	2022_01_01	2023-01-01	2023-01-02	{<config>}	ACTIVE
ns1	2022_01_02	2023-01-02	2023-01-03	{<config>}	ACTIVE
ns2	2022_01_02	2022-01-02	2022-01-07	{<config>}	CLOSED

```
CREATE TABLE IF NOT EXISTS <ns>.metadata (
  namespace_name text,
  version        text,
  start_time     timestamp,
  end_time       timestamp,
  // how many time or event buckets to make
  metadata       map<text, text>,
  status         text,
PRIMARY KEY (namespace_name, version)
) WITH CLUSTERING ORDER BY (version DESC)
```

Time-Series Storage Layout

Metadata table

namespace_name	version	start_time	end_time	status
[redacted]_history_service	data_20240721_00_42	1721522555888000	1729298555888000	ACTIVE
[redacted]_history_service	data_20240422_00_42	1713746555888000	1721522555888000	ACTIVE
[redacted]_history_service	data_20240123_00_42	1705970555888000	1713746555888000	ACTIVE
[redacted]_history_service	data_20231025_00_42	1698194555888000	1705970555888000	ACTIVE
[redacted]_history_service	data_20230727_00_42	1690418555888000	1698194555888000	ACTIVE
[redacted]_history_service	data_20230428_00_42	1682642555888000	1690418555888000	ACTIVE
[redacted]_history_service	data_20230128_00_42	1674866555888000	1682642555888000	ACTIVE
[redacted]_history_service	data_20221030_00_42	1667090555888000	1674866555888000	ACTIVE
[redacted]_history_service	data_20220801_00_42	1659314555888000	1667090555888000	ACTIVE
[redacted]_history_service	data_20220503_00_42	1651538555888000	1659314555888000	DELETED
[redacted]_history_service	data_20220202_00_42	1643762555888000	1651538555888000	DELETED
[redacted]_history_service	data_20211104_00_42	1635986555888000	1643762555888000	DELETED
[redacted]_history_service	data_20210806_00_42	1628210555888000	1635986555888000	DELETED
[redacted]_history_service	data_20210508_00_42	1620434555888000	1628210555888000	DELETED
[redacted]_history_service	data_20210207_00_42	1612658555888000	1620434555888000	DELETED
[redacted]_history_service	data_20201109_00_42	1604882555888000	1612658555888000	DELETED
[redacted]_history_service	data_20200811_00_42	1597106555888000	1604882555888000	DELETED
[redacted]_history_service	data_20200513_00_42	1589330555888000	1597106555888000	DELETED
[redacted]_history_service	data_20200213_00_42	1581554555888000	1589330555888000	DELETED
[redacted]_history_service	FIRST_SLICE	1651538555888000	1651538555888000	ACTIVE

Time Slice - Shard by Table

```
persistence_prod cass_dgw_ts_          _history_service--useast1d us-east-1
(nfsuper) /mnt/data/cassandra/data/impression_history_service $ du -shc *
0       data_20200213_00_42-0b91ad00ca7a11ec9d3efb8c1dedee89
0       data_20200513_00_42-08216570ca7a11ecbadeb1d99c6b1cff
0       data_20200811_00_42-04200170ca7a11ecbb66ed9e09af8275
0       data_20201109_00_42-00b8ba90ca7a11ecad1f2944ed95fca9
0       data_20210207_00_42-fd41bc40ca7911ec9b39db076ee72b0d
0       data_20210508_00_42-f947ab40ca7911ecb296e56a17d142b4
0       data_20210806_00_42-f621da30ca7911ecaa9759e45316053e
0       data_20211104_00_42-f239a380ca7911eca7b30d5f8869e00b
0       data_20220202_00_42-ee1993f0ca7911ec95395fdea5a24fa8
0       data_20220503_00_42-5a5d8ee0e35c11eca081256038c5d02f
1.5T    data_20220801_00_42-5dc216a0e35c11ecb8acff3beb89a17d
1.2T    data_20221030_00_42-62011fe0e35c11ecaca0153c136468d2
1.2T    data_20230128_00_42-90270f431f4435869fc376edf07e0538
1.1T    data_20230428_00_42-530d343cdbb73fb79344af761425f827
1.1T    data_20230727_00_42-04b376573be33ab683a8d17bc6daf455
572G    data_20231025_00_42-5e9ffebc21413ce8b094802352417830
0       data_20240123_00_42-afb6482796343a2bb6e81be80ad37c17
0       data_20240422_00_42-7e17ccc11656396a893cffa94586bcce
0       data_20240721_00_42-4878c769892433ee8c6674683b7821aa
0       metadata-8dfc79b0ca7911ec901035819e30d34e
6.5T    total
```

Slice State Machine

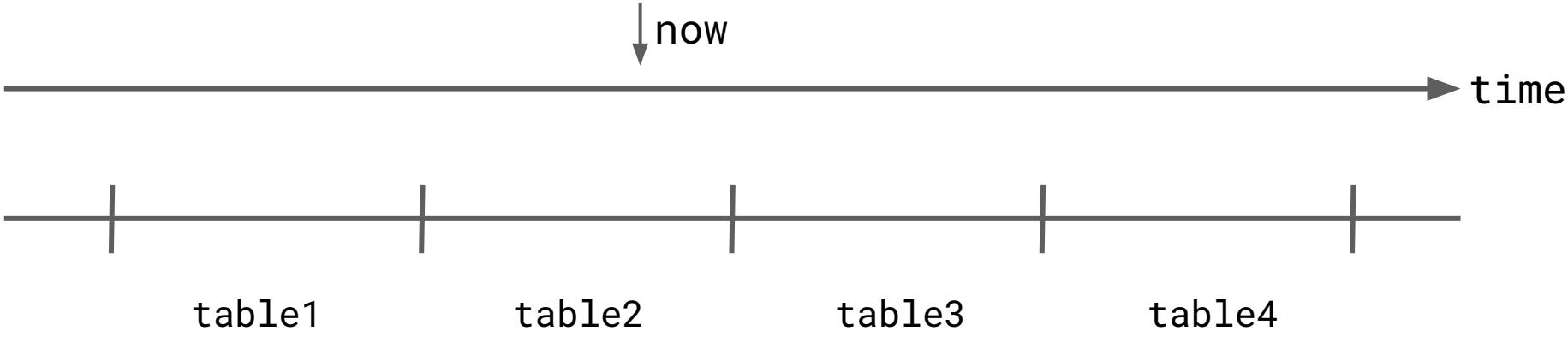
PENDING

ACTIVE

CLOSED

DELETED

Time Slice - Shard by Table



Advantages over TWCS?

- Use up to *80-90% disk*
- Decouple retention from TTL
- Decouple background operations: compression, compaction

Pro-Tips

- Create "runway" in future
- Idempotent CREATE WITH `id = <hash>`

Time-Series Storage Layout

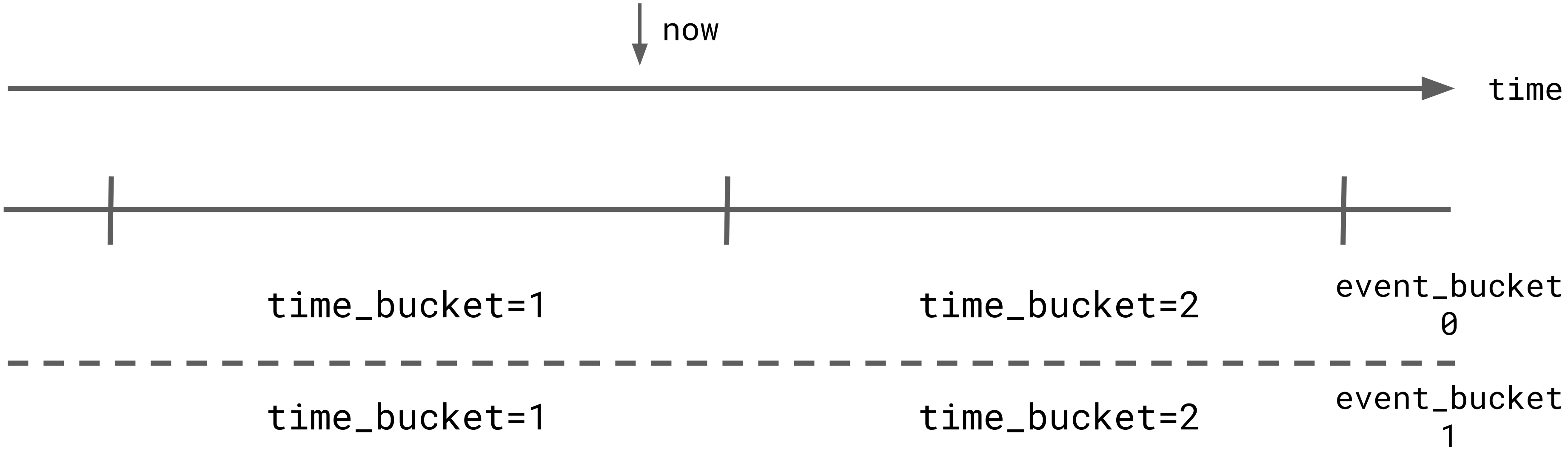
Data table

time series id	time bucket	event bucket	event time	event id	value	value_metadata
1234	19682	1	ts1	uuid1	event1	{ metadata: … }
1234	19681	2	ts2	uuid2	event2	{ metadata: … }
5678	89681	1	ts1	uuid3	event3	{ metadata: … }

```
CREATE TABLE IF NOT EXISTS <ns>.<version> (
  time_series_id text,
  time_bucket    bigint,
  event_bucket   int,
  event_time     timestamp,
  event_id       text,
  value          blob,
  value_metadata blob,
PRIMARY KEY(
   (time_series_id, time_bucket, event_bucket), event_time, event_id))
 WITH CLUSTERING ORDER BY (event_time DESC, event_id DESC, key DESC)
```

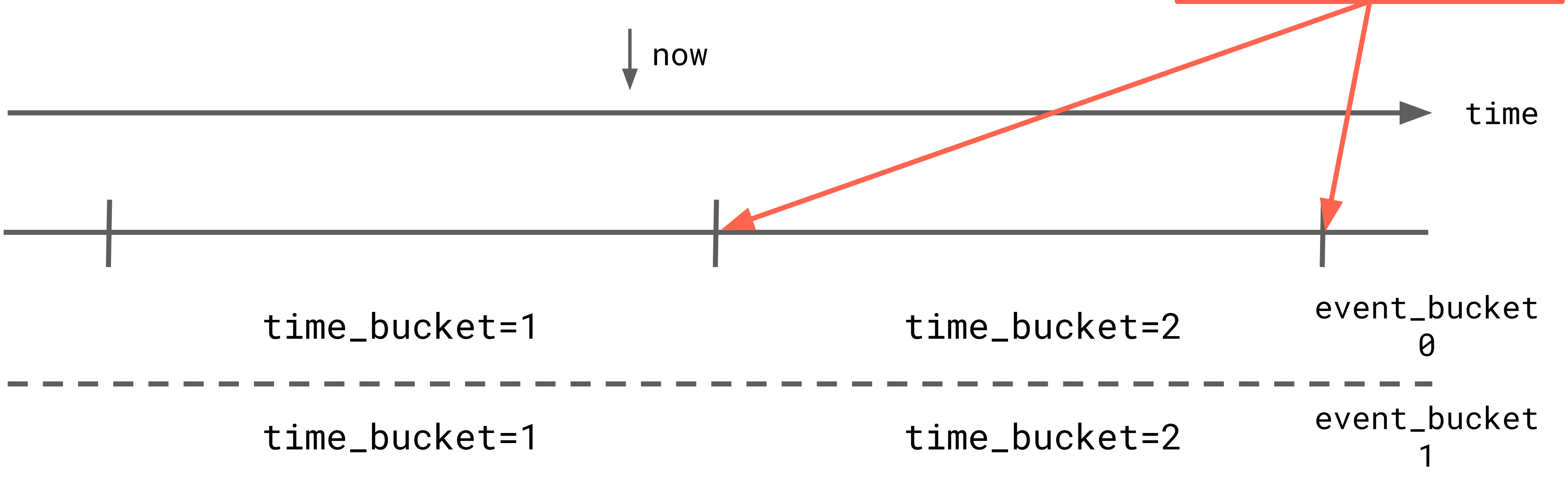
Time-Series Storage Configuration

```
metadata
--------------------------------------------------------------------------------------------------------------------------------------------
{'buckets_per_id': '2', 'buffer_capacity': '4194304', 'buffer_coalesce': '1', 'seconds_per_bucket': '5', 'seconds_per_interval': '7776000'}
{'buckets_per_id': '2', 'buffer_capacity': '4194304', 'buffer_coalesce': '1', 'seconds_per_bucket': '5', 'seconds_per_interval': '7776000'}
{'buckets_per_id': '2', 'buffer_capacity': '4194304', 'buffer_coalesce': '1', 'seconds_per_bucket': '5', 'seconds_per_interval': '7776000'}
{'buckets_per_id': '2', 'buffer_capacity': '4194304', 'buffer_coalesce': '1', 'seconds_per_bucket': '5', 'seconds_per_interval': '7776000'}
```



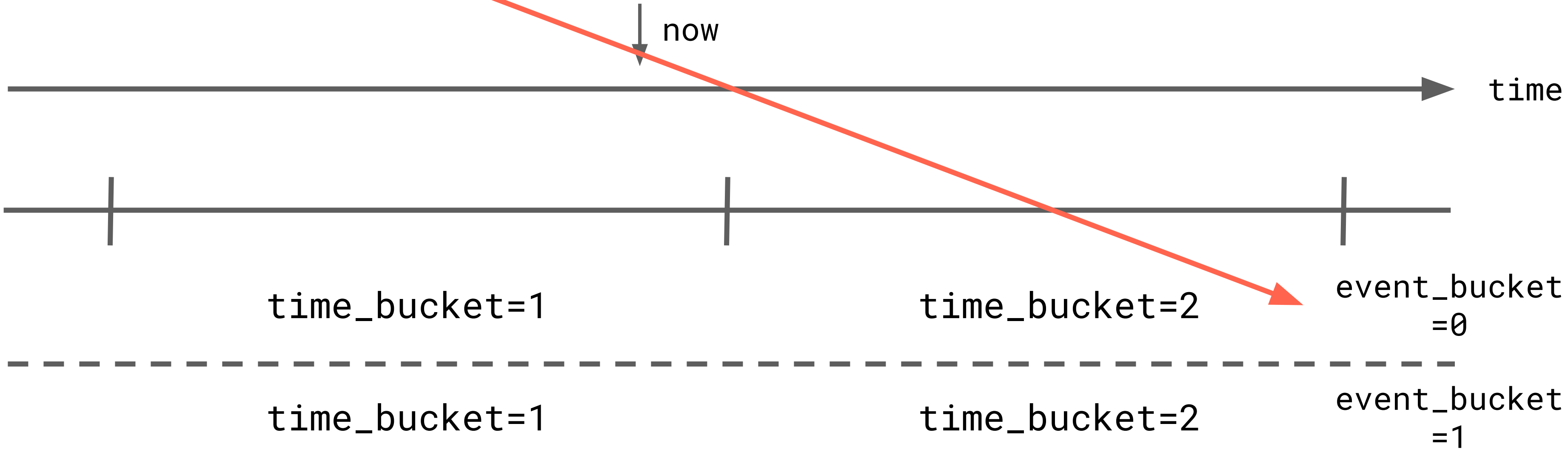
Time-Series Storage Configuration

```
metadata
--------------------------------------------------------------------------------------------------------------------------------------
{'buckets_per_id': '2', 'buffer_capacity': '4194304', 'buffer_coalesce': '1', 'seconds_per_bucket': '5', 'seconds_per_interval': '7776000'}
{'buckets_per_id': '2', 'buffer_capacity': '4194304', 'buffer_coalesce': '1', 'seconds_per_bucket': '5', 'seconds_per_interval': '7776000'}
{'buckets_per_id': '2', 'buffer_capacity': '4194304', 'buffer_coalesce': '1', 'seconds_per_bucket': '5', 'seconds_per_interval': '7776000'}
{'buckets_per_id': '2', 'buffer_capacity': '4194304', 'buffer_coalesce': '1', 'seconds_per_bucket': '5', 'seconds_per_interval': '7776000'}
```



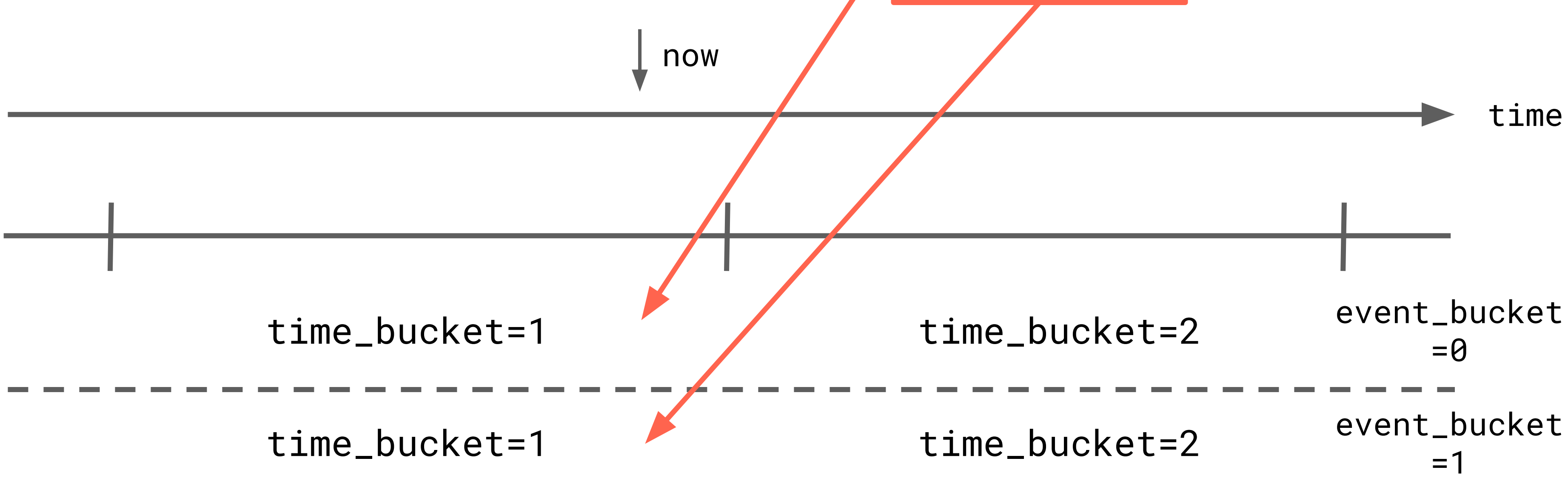
Time-Series Storage Configuration

```
metadata
---------------------------------------------------------------------------------------------------------------------------------------
{'buckets_per_id': '2', 'buffer_capacity': '4194304', 'buffer_coalesce': '1', 'seconds_per_bucket': '5', 'seconds_per_interval': '7776000'}
{'buckets_per_id': '2', 'buffer_capacity': '4194304', 'buffer_coalesce': '1', 'seconds_per_bucket': '5', 'seconds_per_interval': '7776000'}
{'buckets_per_id': '2', 'buffer_capacity': '4194304', 'buffer_coalesce': '1', 'seconds_per_bucket': '5', 'seconds_per_interval': '7776000'}
{'buckets_per_id': '2', 'buffer_capacity': '4194304', 'buffer_coalesce': '1', 'seconds_per_bucket': '5', 'seconds_per_interval': '7776000'}
```

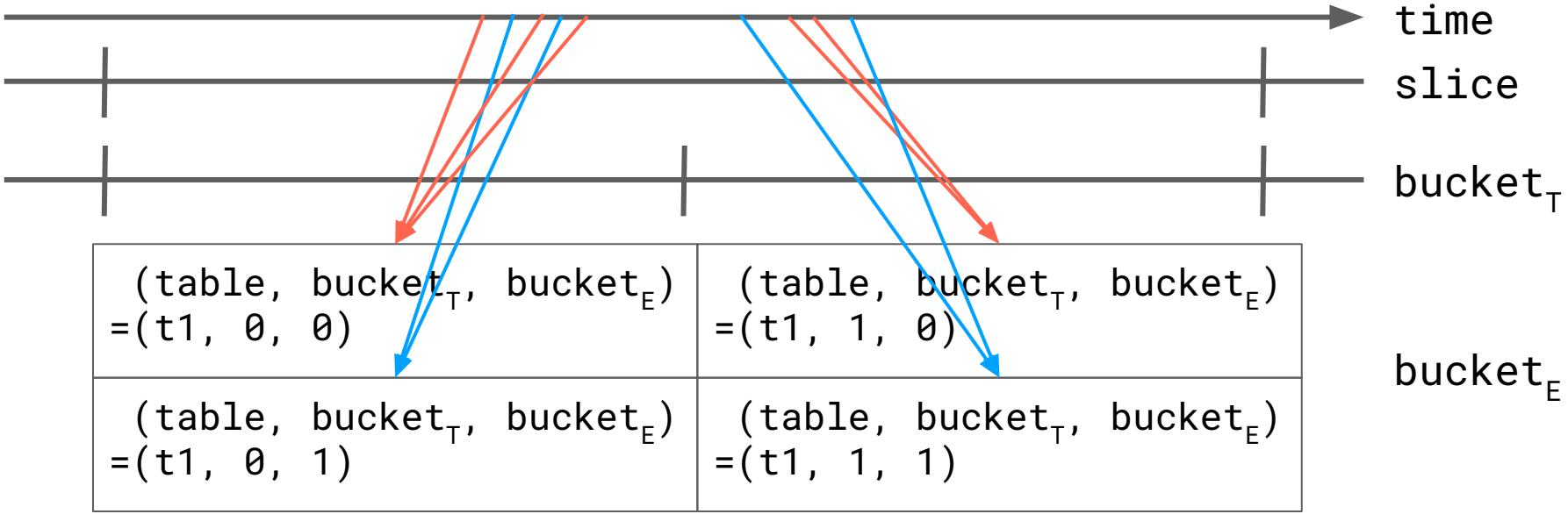


Time-Series Storage Configuration

```
metadata
-----------------------------------------------------------------------------------------------------------------------
{'buckets_per_id': '2', 'buffer_capacity': '4194304', 'buffer_coalesce': '1', 'seconds_per_bucket': '5', 'seconds_per_interval': '7776000'}
{'buckets_per_id': '2', 'buffer_capacity': '4194304', 'buffer_coalesce': '1', 'seconds_per_bucket': '5', 'seconds_per_interval': '7776000'}
{'buckets_per_id': '2', 'buffer_capacity': '4194304', 'buffer_coalesce': '1', 'seconds_per_bucket': '5', 'seconds_per_interval': '7776000'}
{'buckets_per_id': '2', 'buffer_capacity': '4194304', 'buffer_coalesce': '1', 'seconds_per_bucket': '5', 'seconds_per_interval': '7776000'}
```



Time Partition - Tunable



Why So Many Buckets?

Flexible Buckets

Tune for your use case!

Tunable	Primary Purpose	Tradeoff
Tables (slice_interval)	`Retention`	`DDL in Critical Path`
Time Buckets (time_bucket)	`Read amplification`	`Lookback latency`
Random Event Buckets (event_bucket)	`Sharding wide partitions`	`Parallel fan-out`

Key-Value Storage Layout

Data Table - Metadata buckets

id	bucket	key	chunk	version	value	value_metadata
1234	-2	key2	6	timeuuid1	<empty>	{ “metadata”: … }
1234	-2	key2	12	timeuuid2	<empty>	{ “metadata”: … }

Data Table

id	bucket	key	chunk	version	value	value_metadata
1234	64200	key2	1	timeuuid1	chunk1	{ “metadata”: … }
1234	64201	key2	2	timeuuid1	chunk2	{ “metadata”: … }
1234	78902	key2	1	timeuuid2	chunk1	{ “metadata”: … }

Key-Value Storage Layout

Writes

- **Stage** chunks to data table and **commit** to base table after chunks are staged.
- **IdempotencyToken** is used for concurrent writes
- Small values are committed to **base table only**

Reads

- Read base table for version and number of chunks
- Retrieve chunks adaptively
- Fallback to older version if chunks data is not replicated

Success Stories

Cloud Saves

Tracing

Profile Level User Interaction (PLUID)

Cloud Saves

Netflix **Game Data** streaming scale Key-Value Data

30 - 300MB of game progress metadata per call

Enabled by **chunking and compression** features in Key-Value

Tracing

Netflix **Distributed Tracing** data

1000 of services sending IPC traces

Stores 1-2 PiB of uncompressed data

Immutable events

Profile Level User Interaction (PLUID)

Video + Profile play/pause/stop events

Key-Value for active sessions and Time-Series for session history

800k + 10k Writes
150k + 80k Reads
8 hour TTL + 1 year retention

Tuned for heavy write operations

Key-Value deploys - 400+ shards, 3000+ use-cases
Time-Series has 10+ PiB of events stored across 10s of use-cases

Further Work

Summarization

Re-sharding

Compare and Swap

Thank You.

Vidhya Arvind
varvind@netflix.com

Joey Lynch
josephl@netflix.com