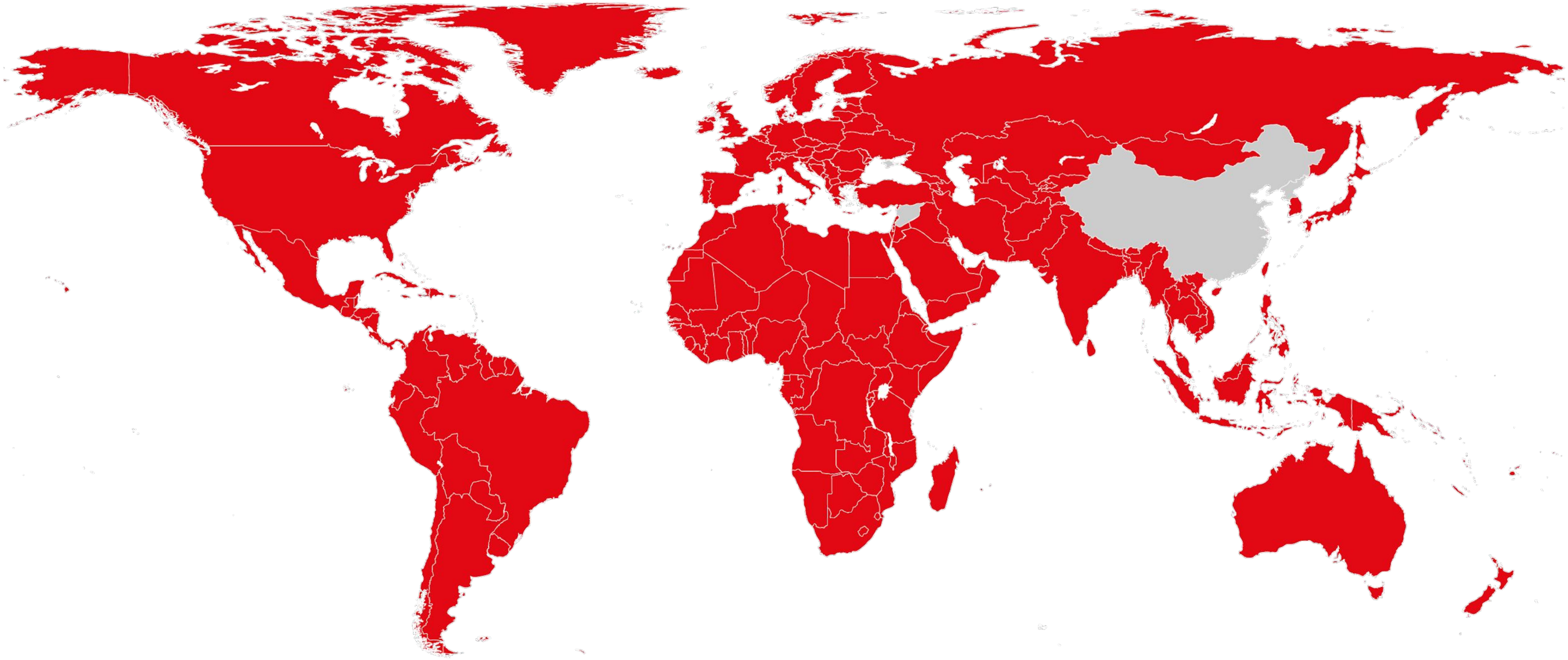
How Netflix Uses Data Abstractions to scale to 100s of use cases

Vidhya Arvind

March 12, 2024

How to efficiently scale when there are 1000s of applications?

Netflix Scale



Speaker

Vidhya Arvind

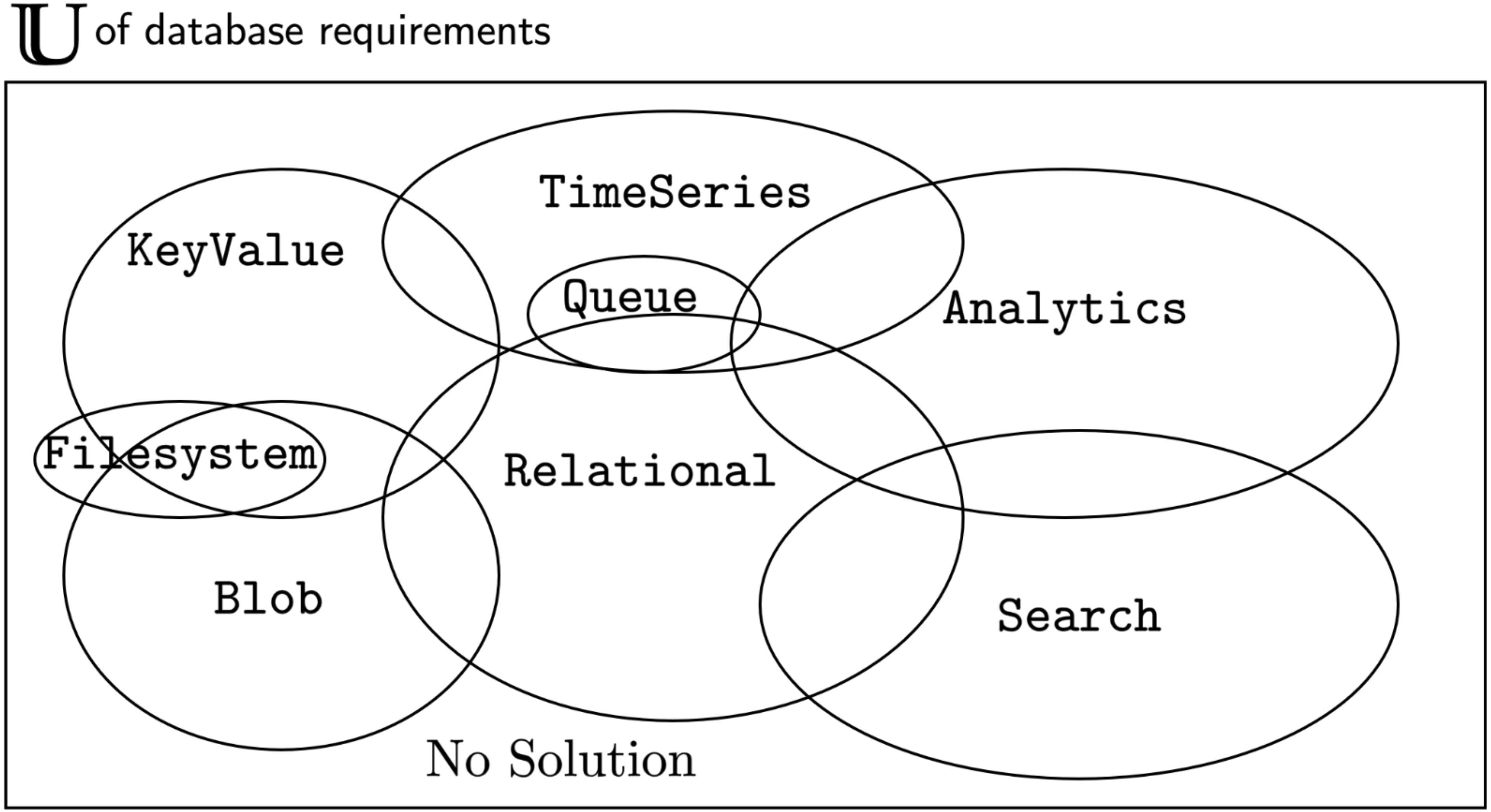
Staff Software Engineer
Data Platform at Netflix

Founding member of Data abstractions at Netflix

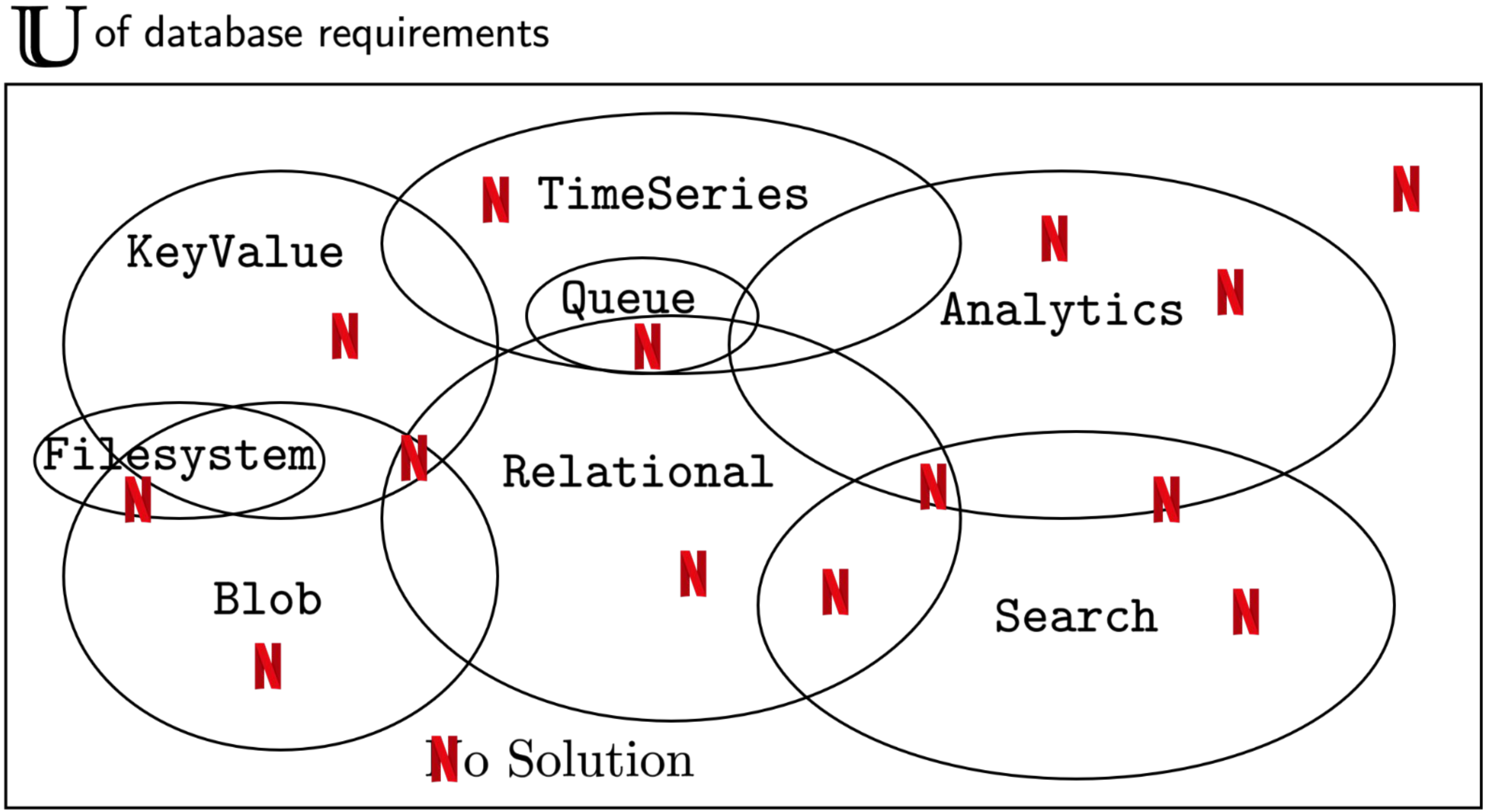
https://vidhyaarvind.github.io/

Can we scale to *all* our data use cases?

Problem: Varied Requirements



Problem: Varied Use-Cases

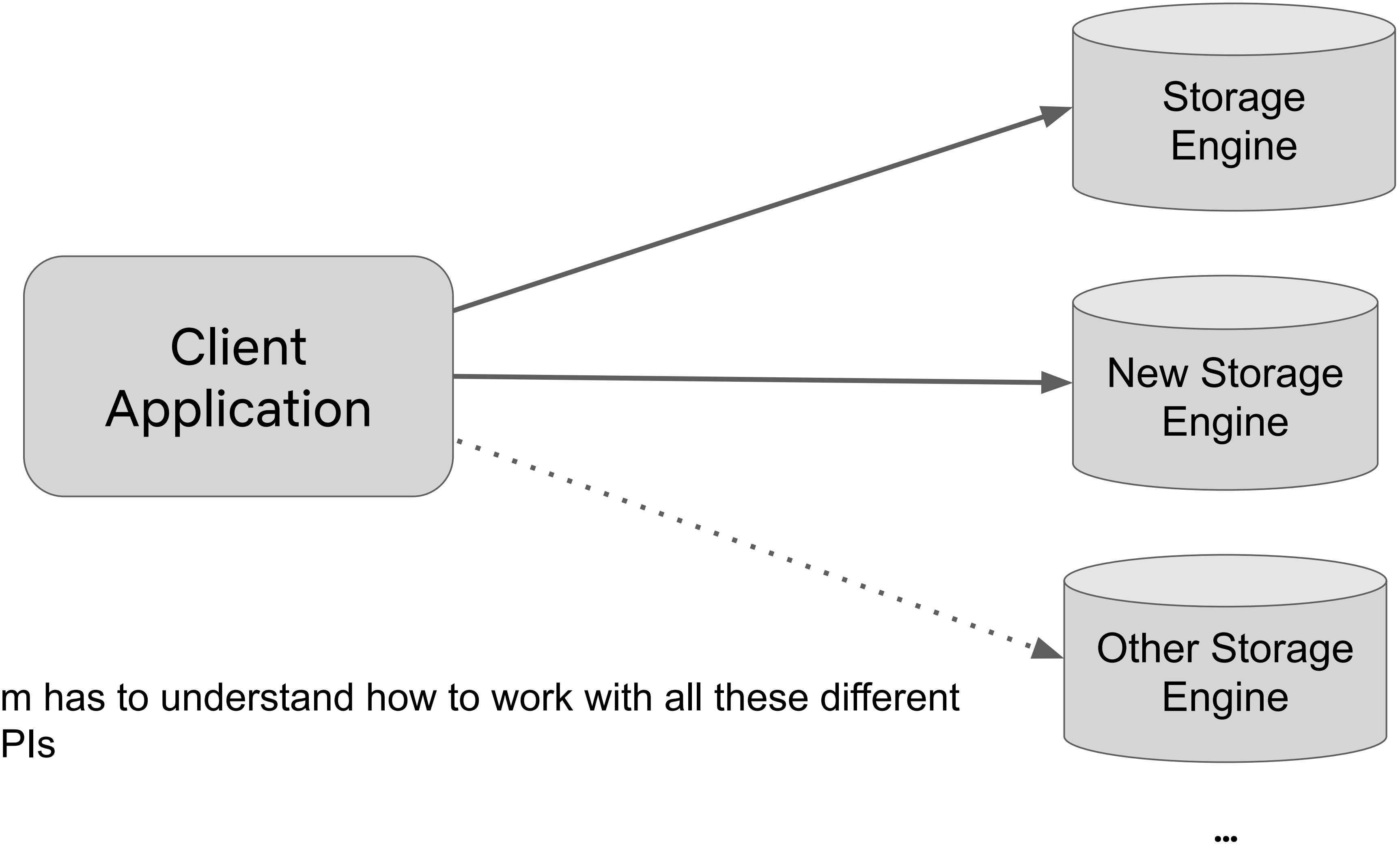


Can take the common patterns and provide a common solution?

Can take the common patterns and provide a common solution?

Can the solution be generic and storage agnostic?

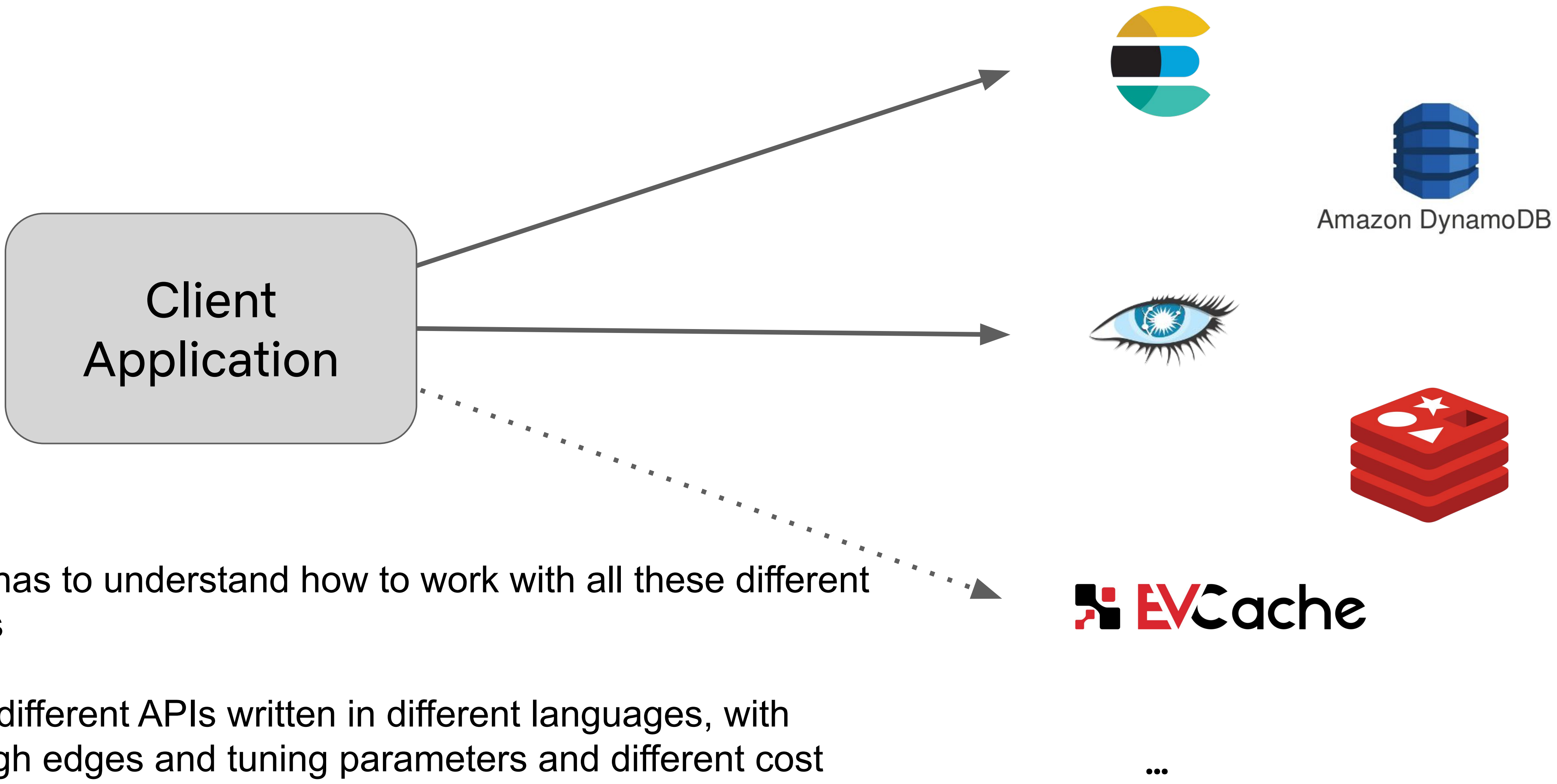
Problem: Variety of Storage engines



Client team has to understand how to work with all these different storage APIs

Problem: Variety of Storage engines

What if we have 100s of services?

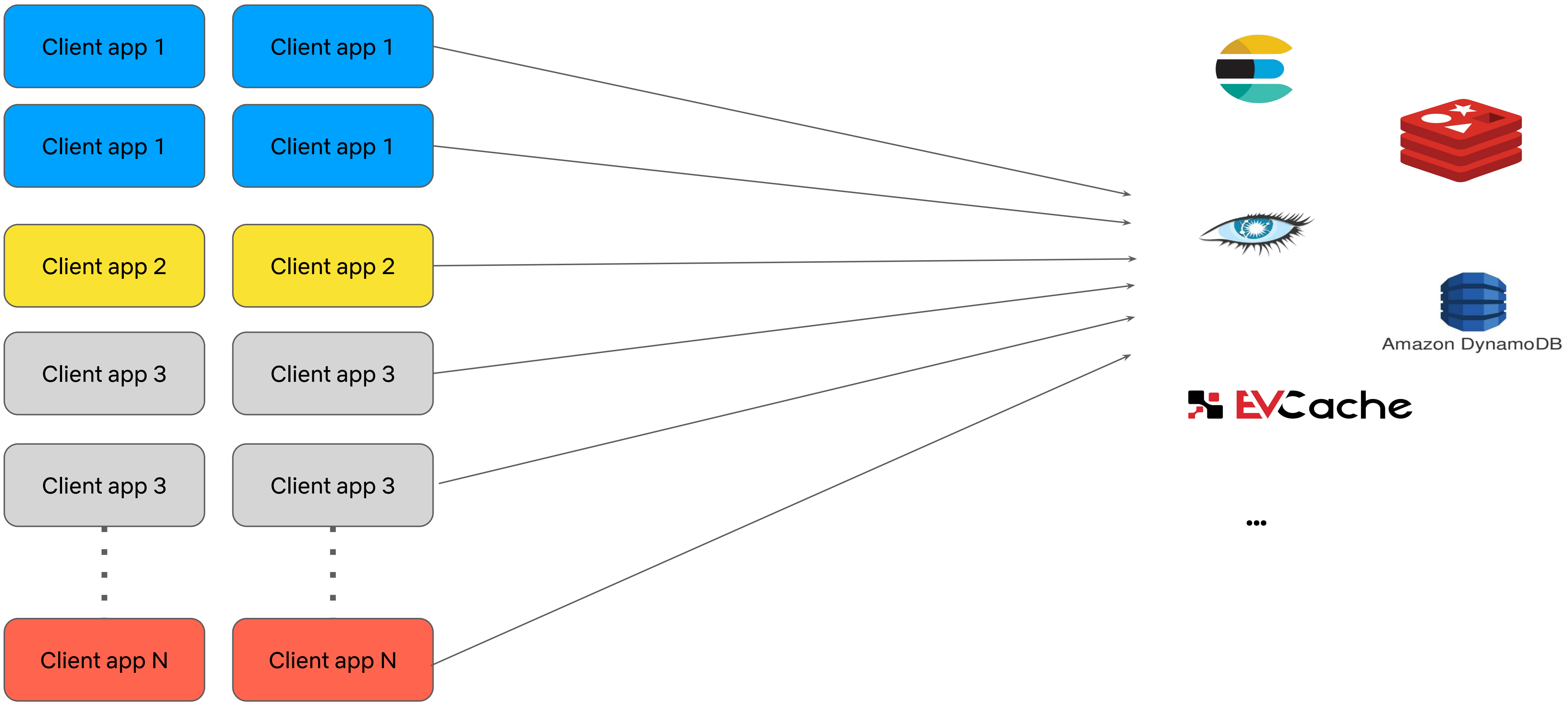


Client team has to understand how to work with all these different storage APIs

Understand different APIs written in different languages, with different rough edges and tuning parameters and different cost models

Without Virtualization

Can we isolate the use cases?



Solution?

Data **Abstraction** Layers!

We can solve any problem by introducing an extra **level of indirection**.

- David J. Wheeler

Fundamental theorem of software engineering

Level of Indirection

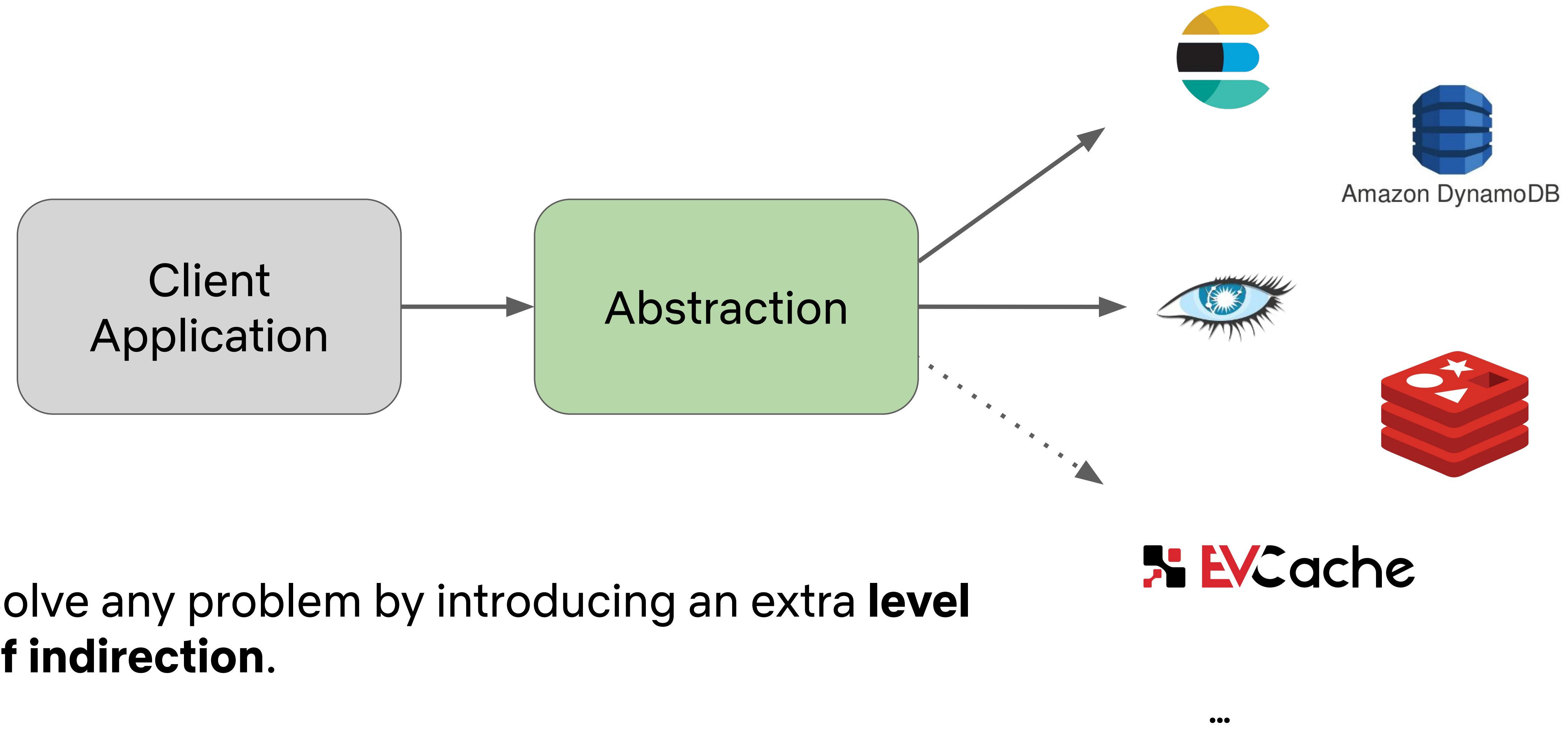
Abstraction

- Level of indirection
- Take the complex system and break into smaller pieces with clearly defined boundaries

Virtualization the abstraction

- Virtualize it by defining the implementation
- Switching between implementations
- Layer systems to solve bigger problems

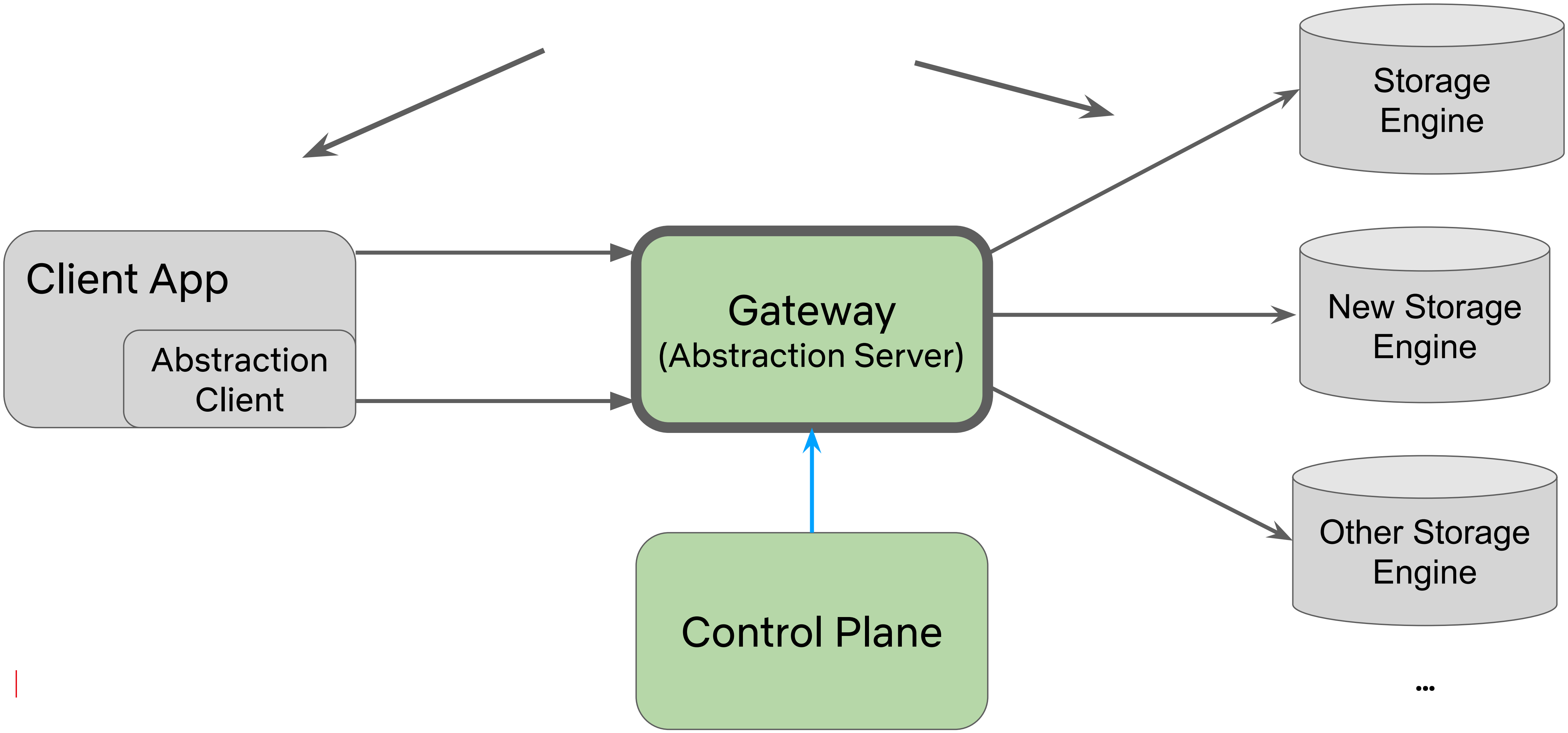
Level of Indirection



Solve any problem by introducing an extra **level of indirection.**

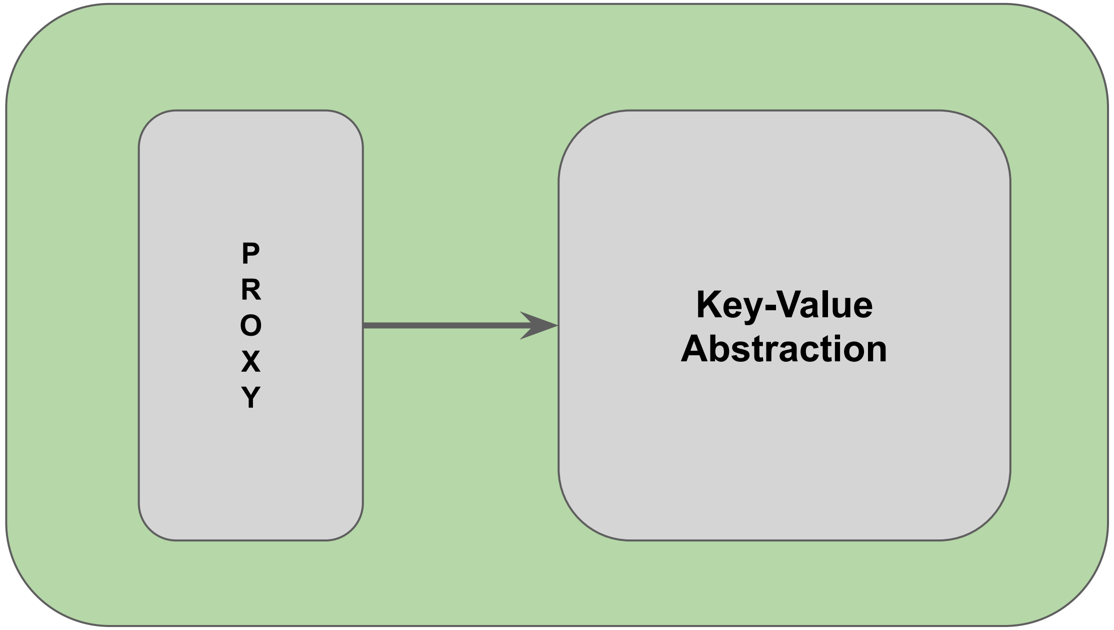
Data Gateway

Abstract **Clients**

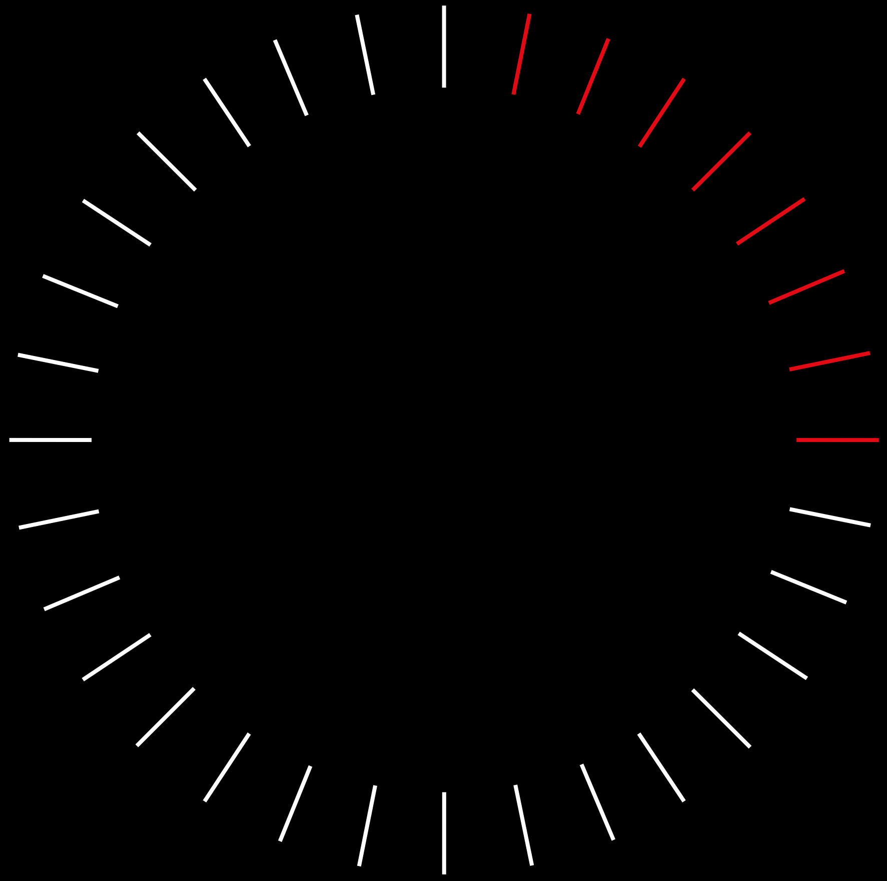


Data Gateway

Gateway (Key-Value Abstraction)

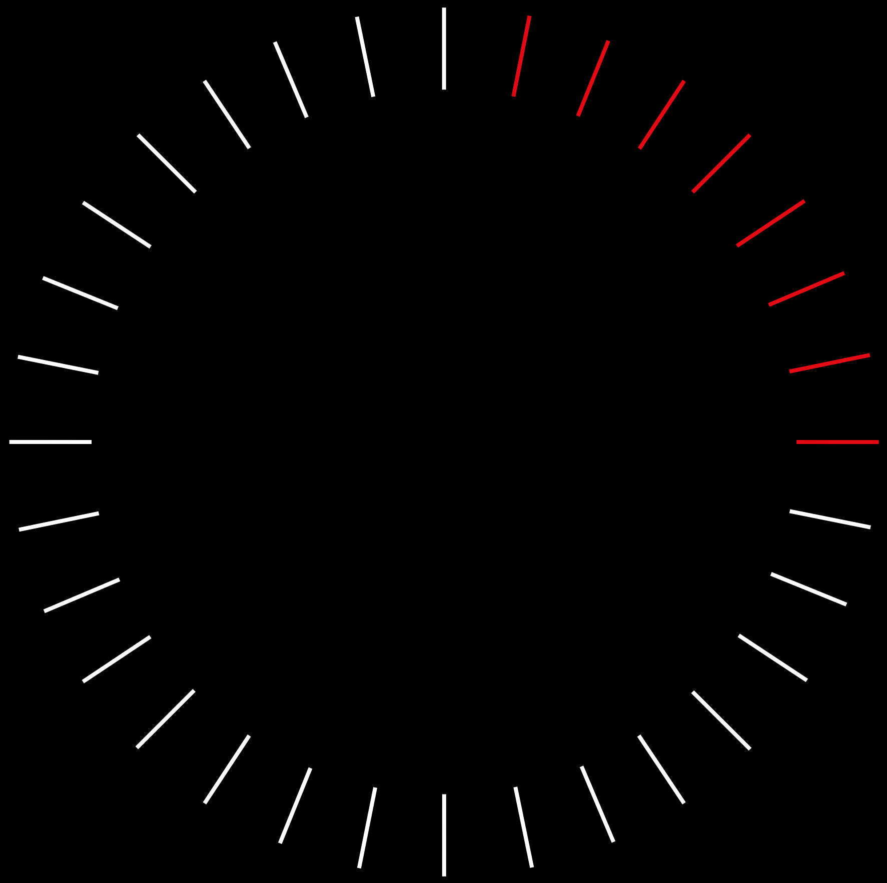


- □ **Virtualization**
- □ Abstraction
- □ Clean APIs



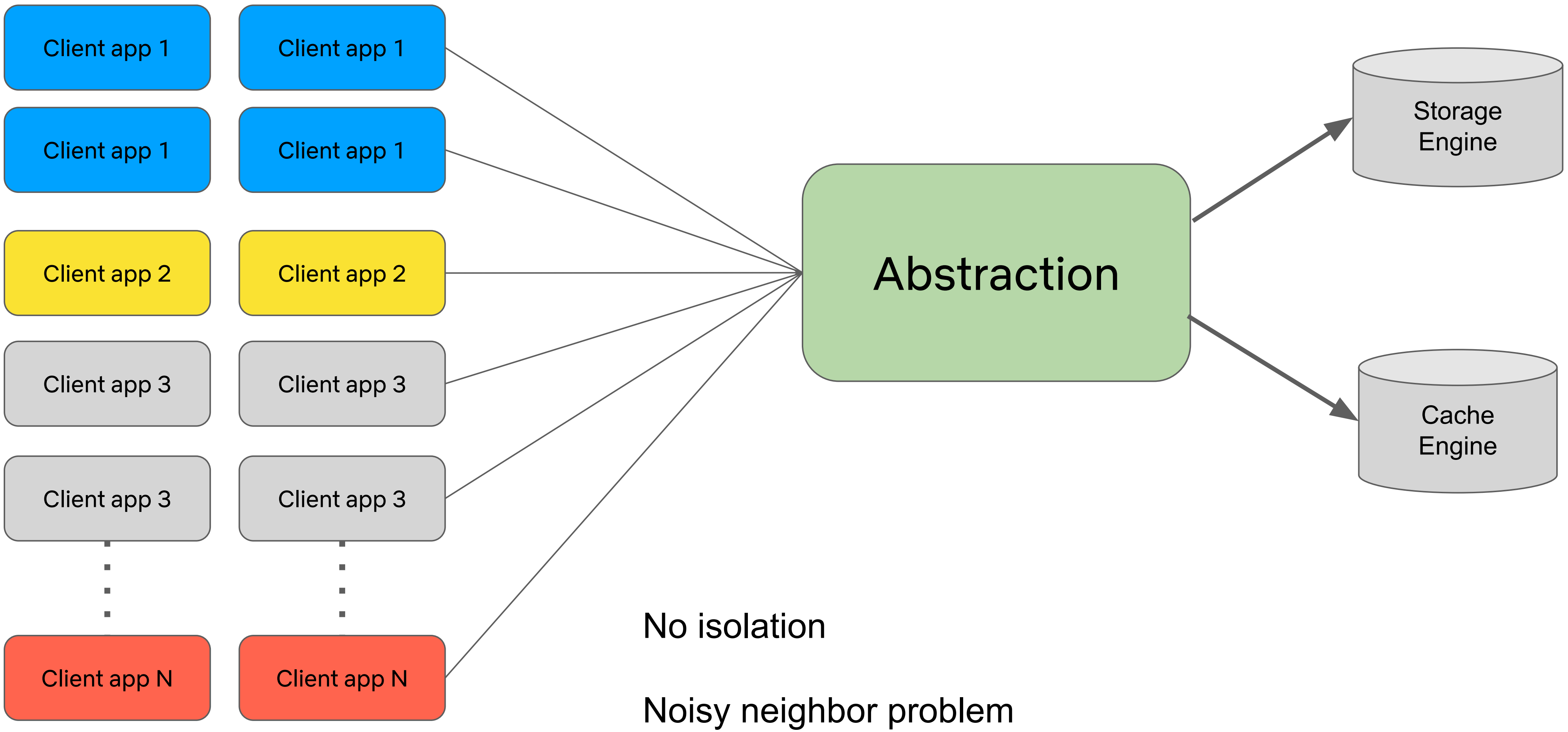
Virtualization

- □ **Sharding**
- □ Composition
- □ Configuration



Without Virtualization

Single point of failure

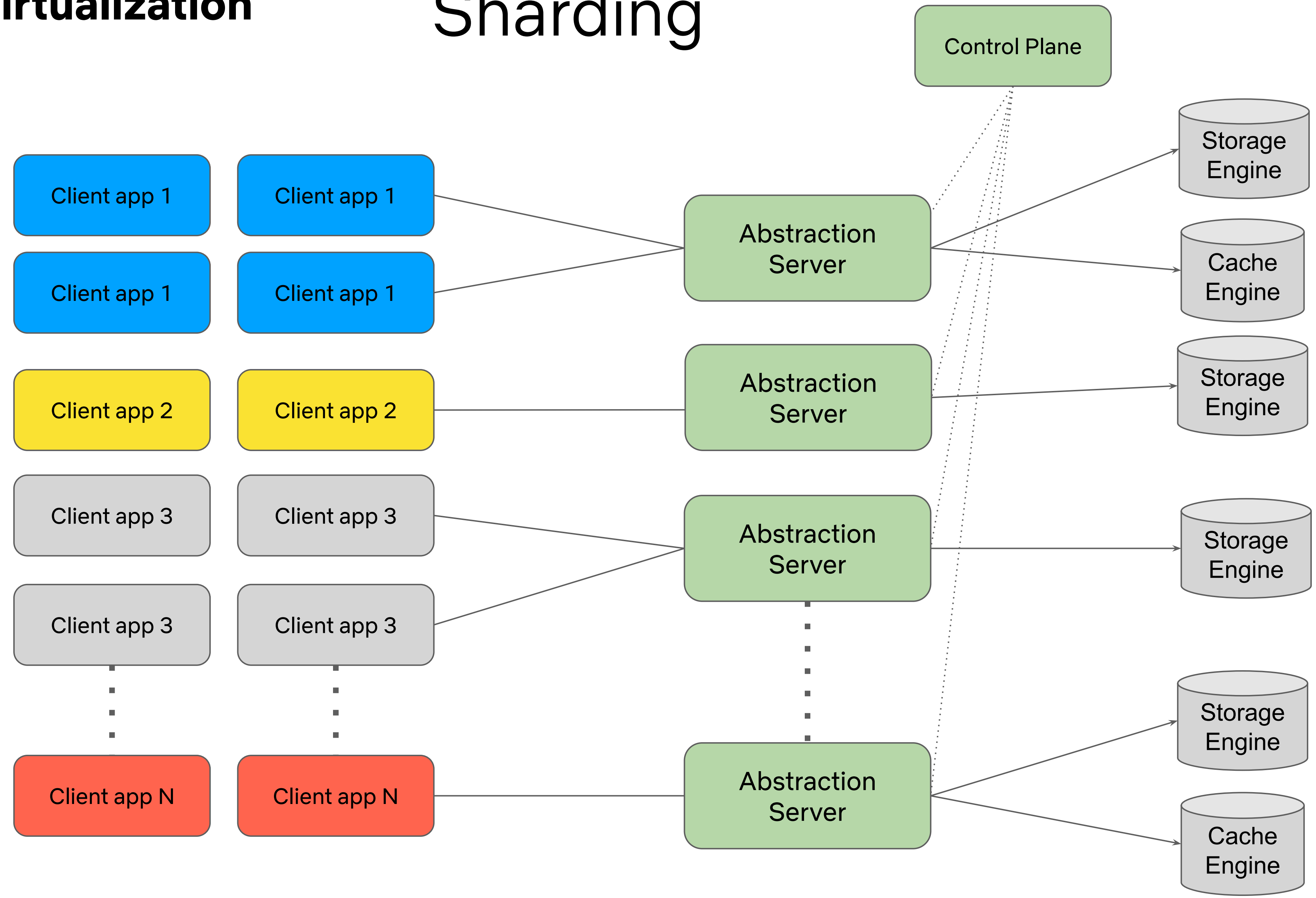


No isolation

Noisy neighbor problem

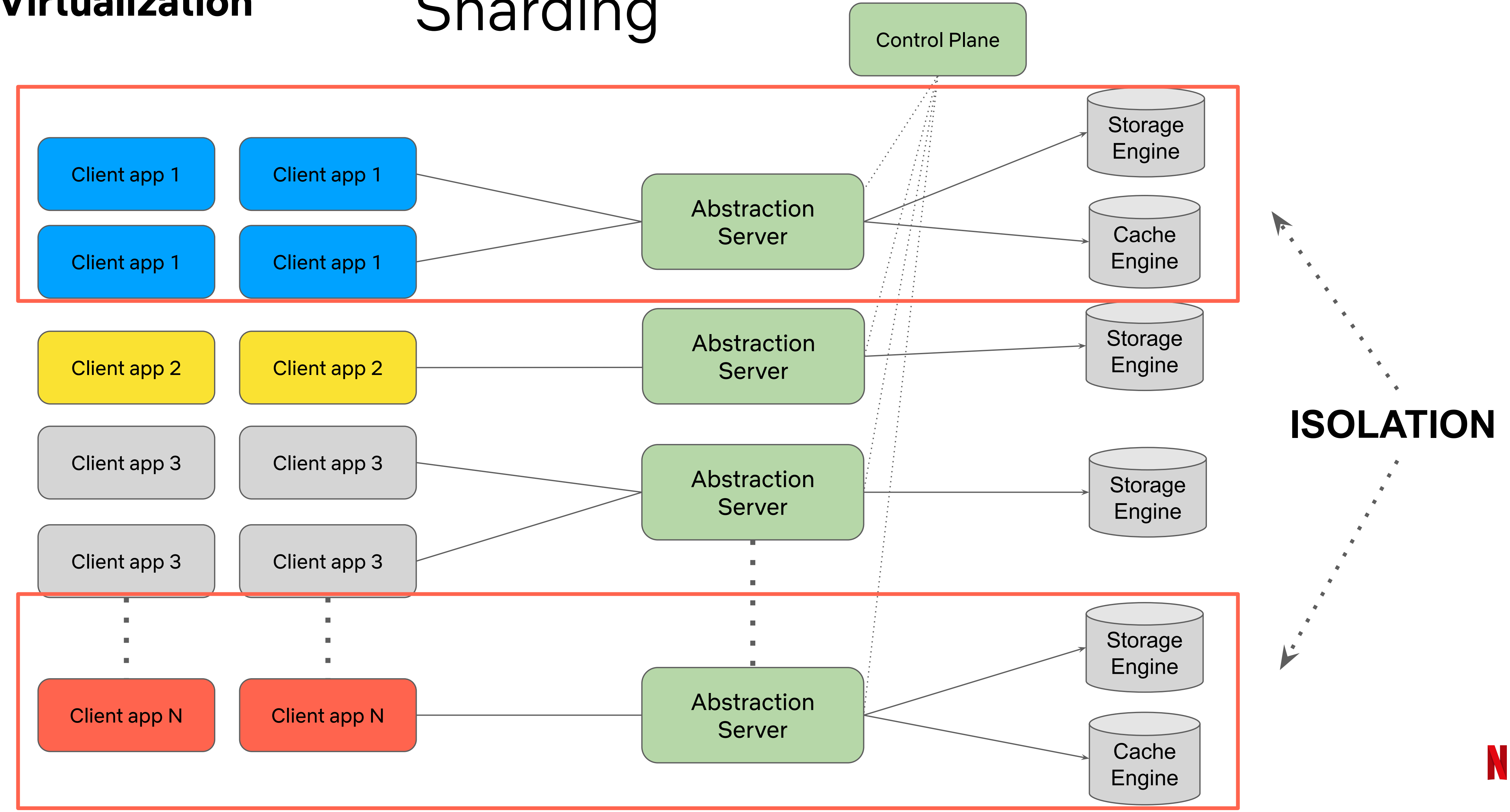
Virtualization

Sharding



Virtualization

Sharding



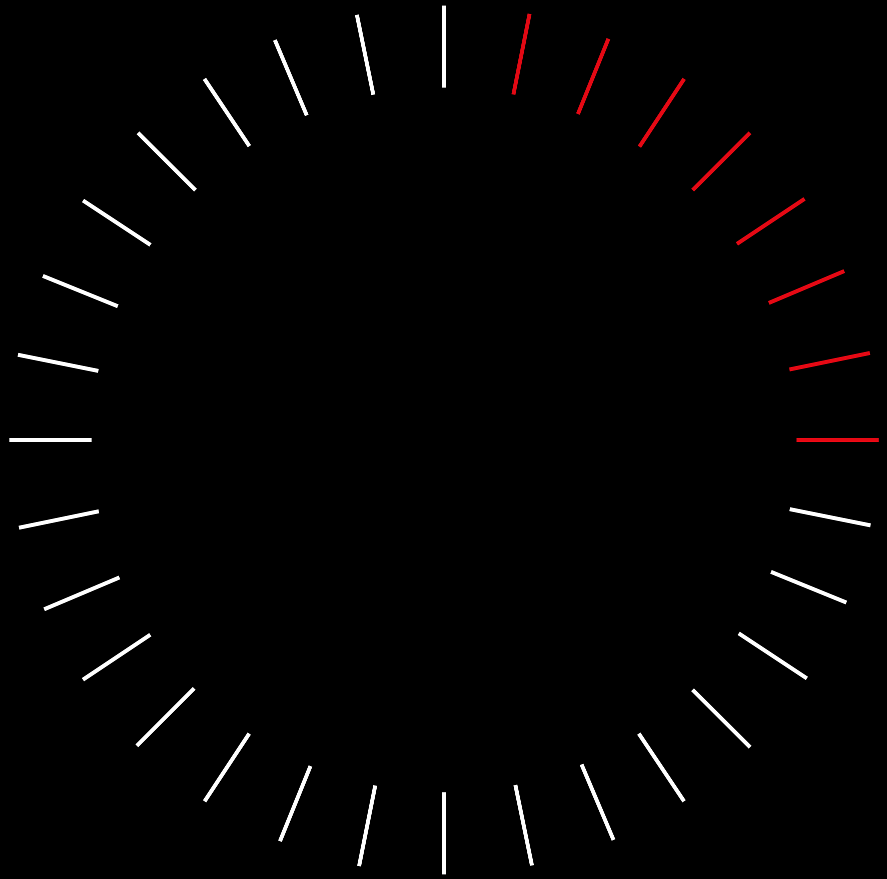
Virtualization

✓ Sharding

□ **Composition**

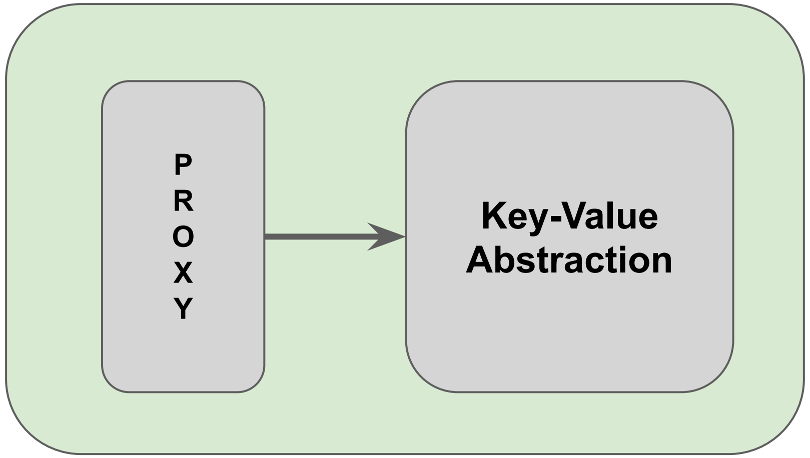
□ Configuration

Checkout talk:
[Application architecture as code (GBL301)]

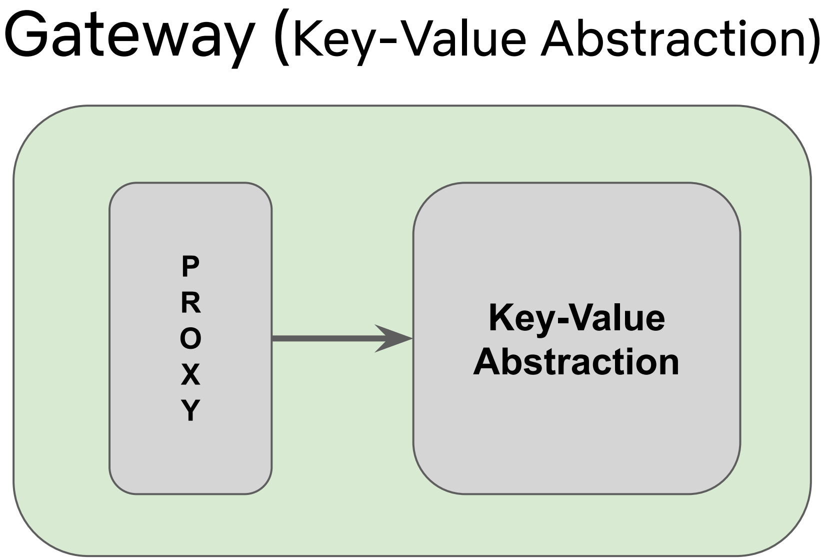


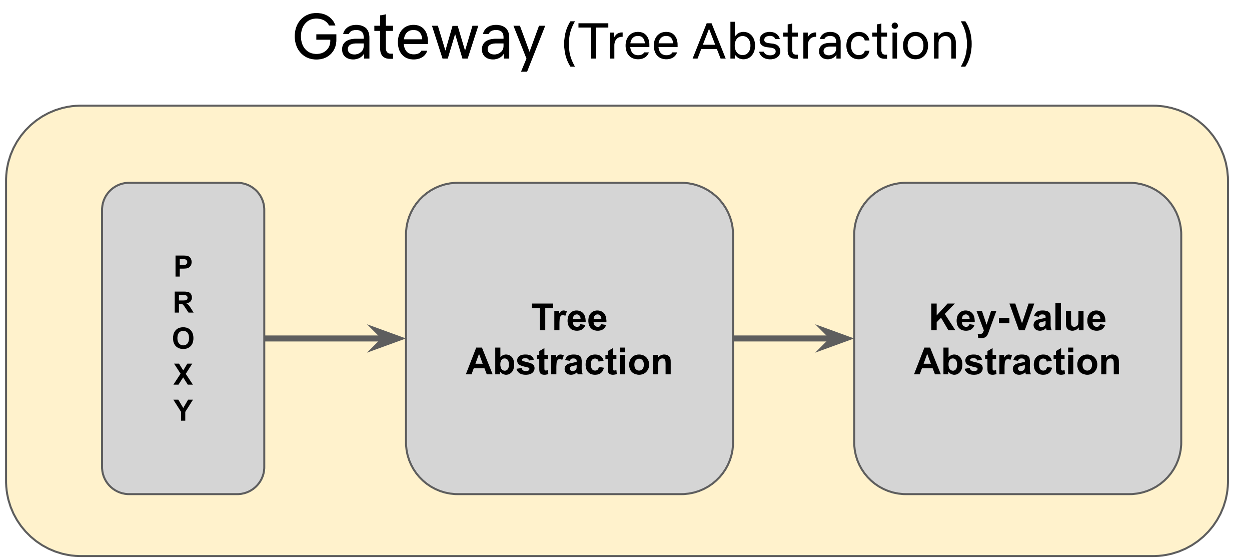
Composition of services

Gateway (Key-Value Abstraction)



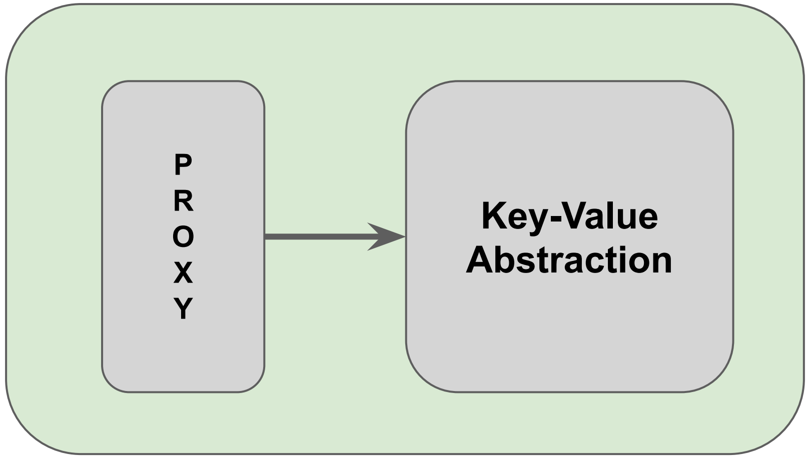
Composition of services



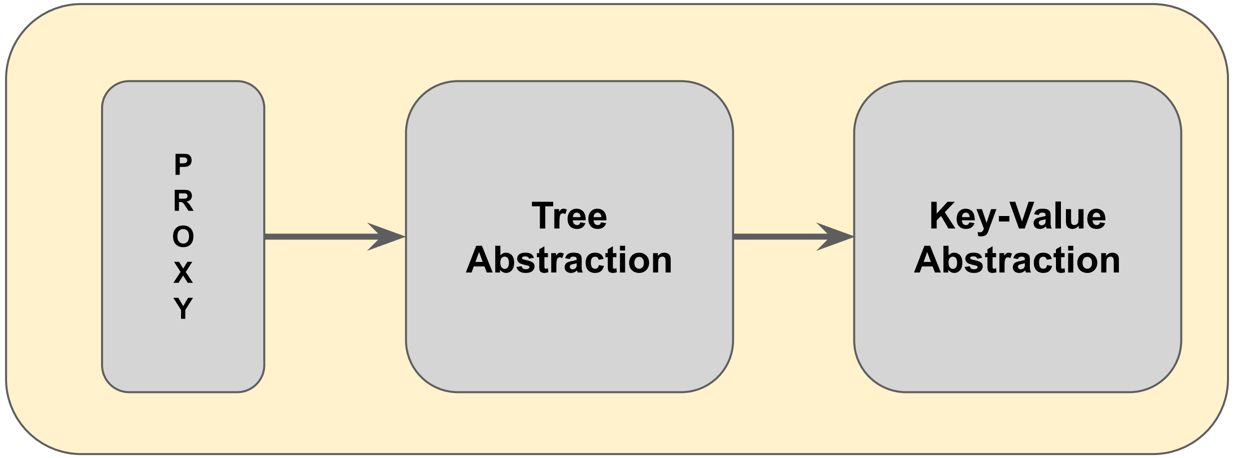


Composition of services

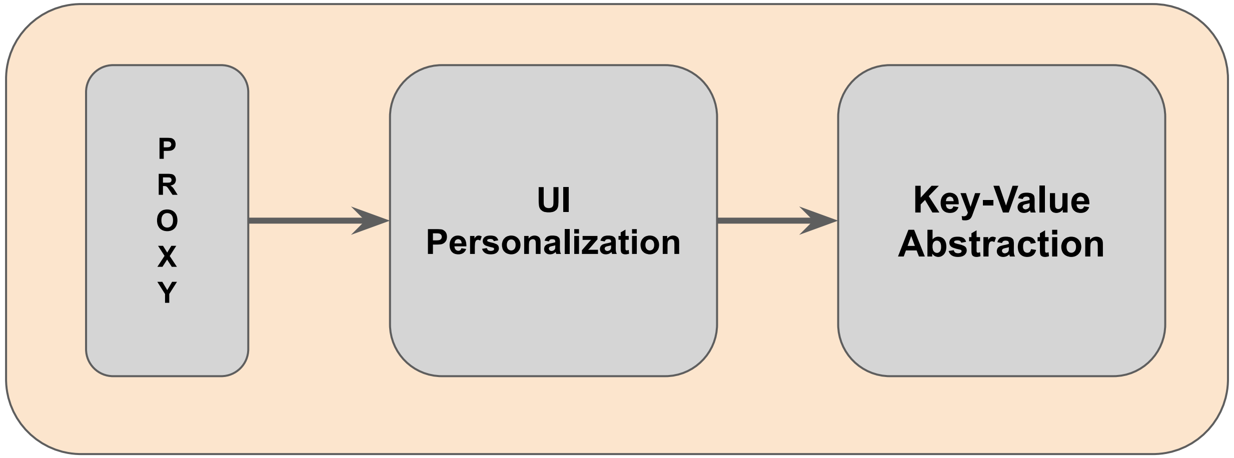
Gateway (Key-Value Abstraction)



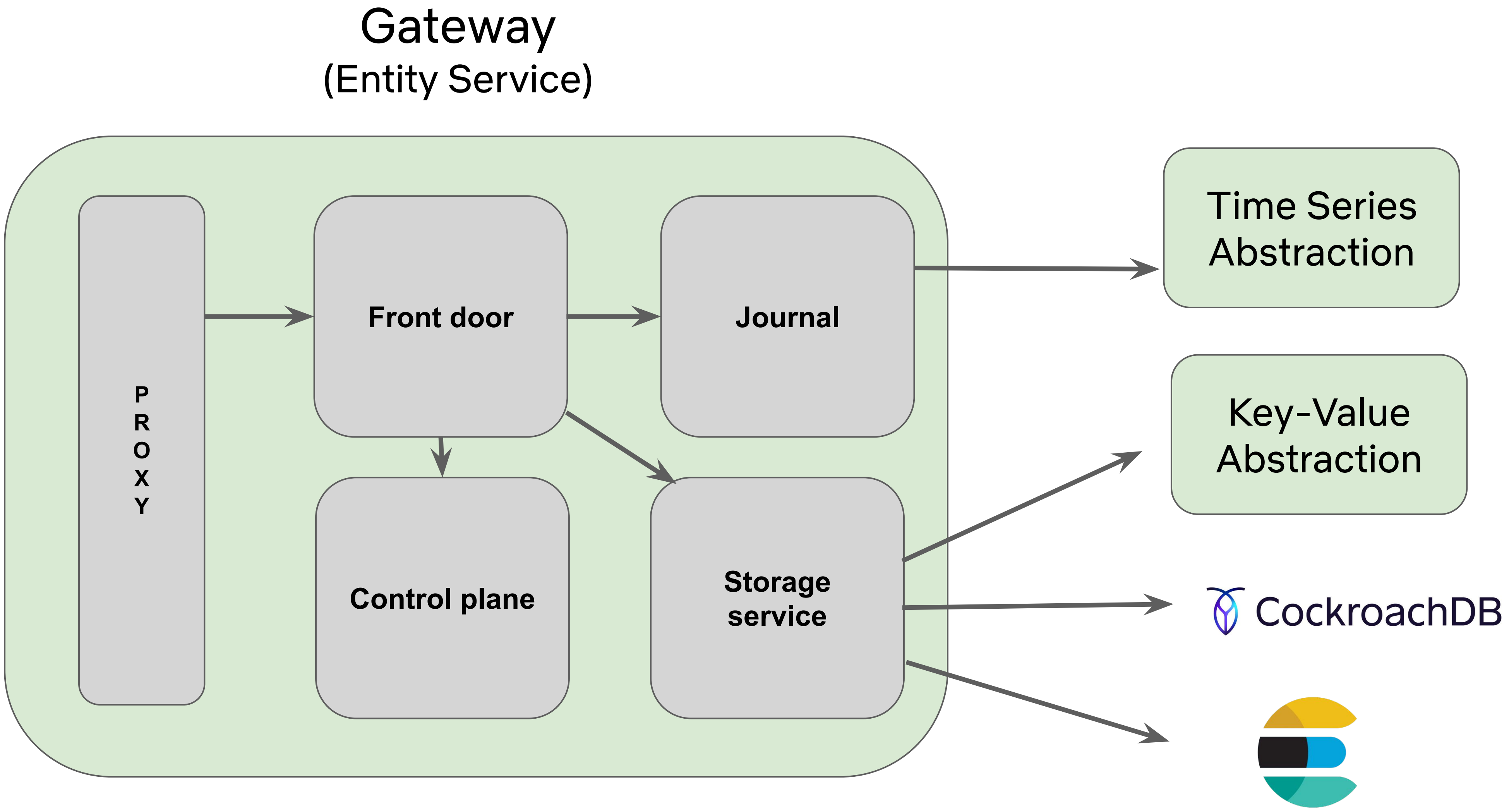
Gateway (Tree Abstraction)



Gateway (Composing custom apis along with abstraction)

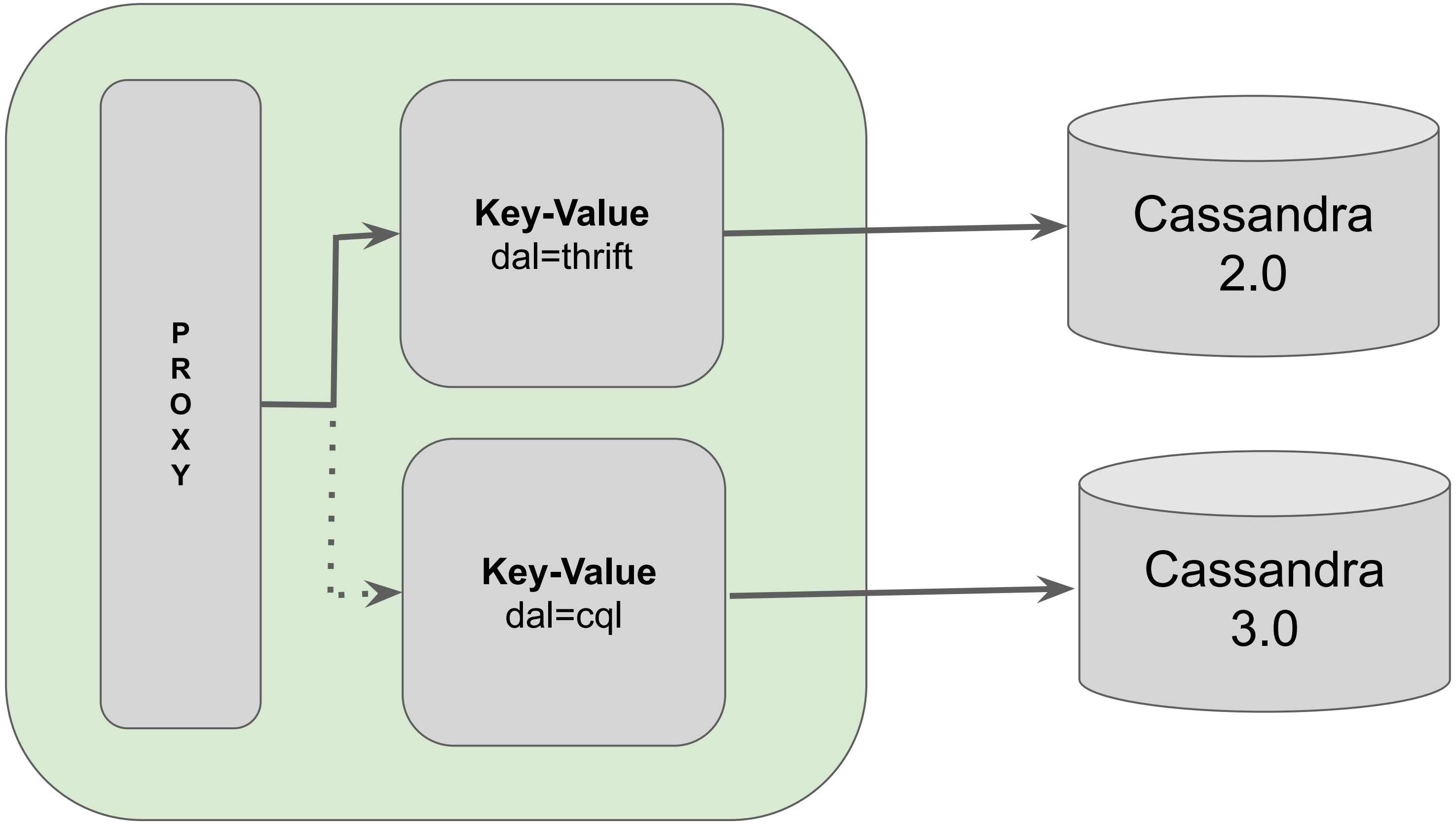


Composition of services



Data Gateway

Shadow Writes

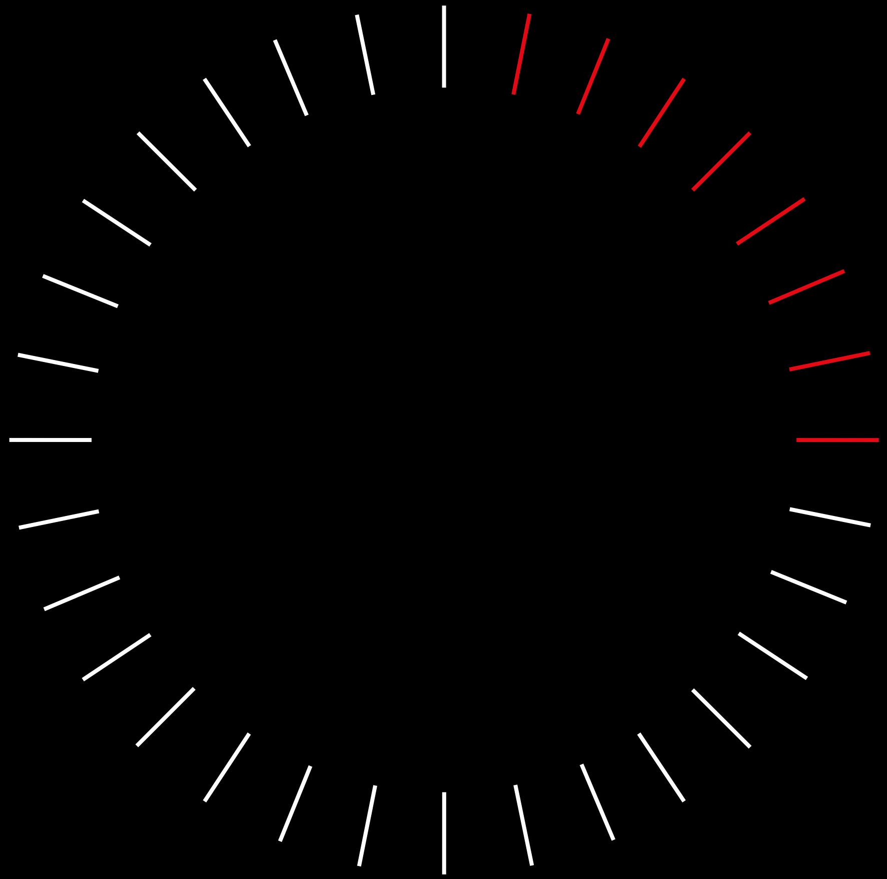


✓ Shading

✓ Composition

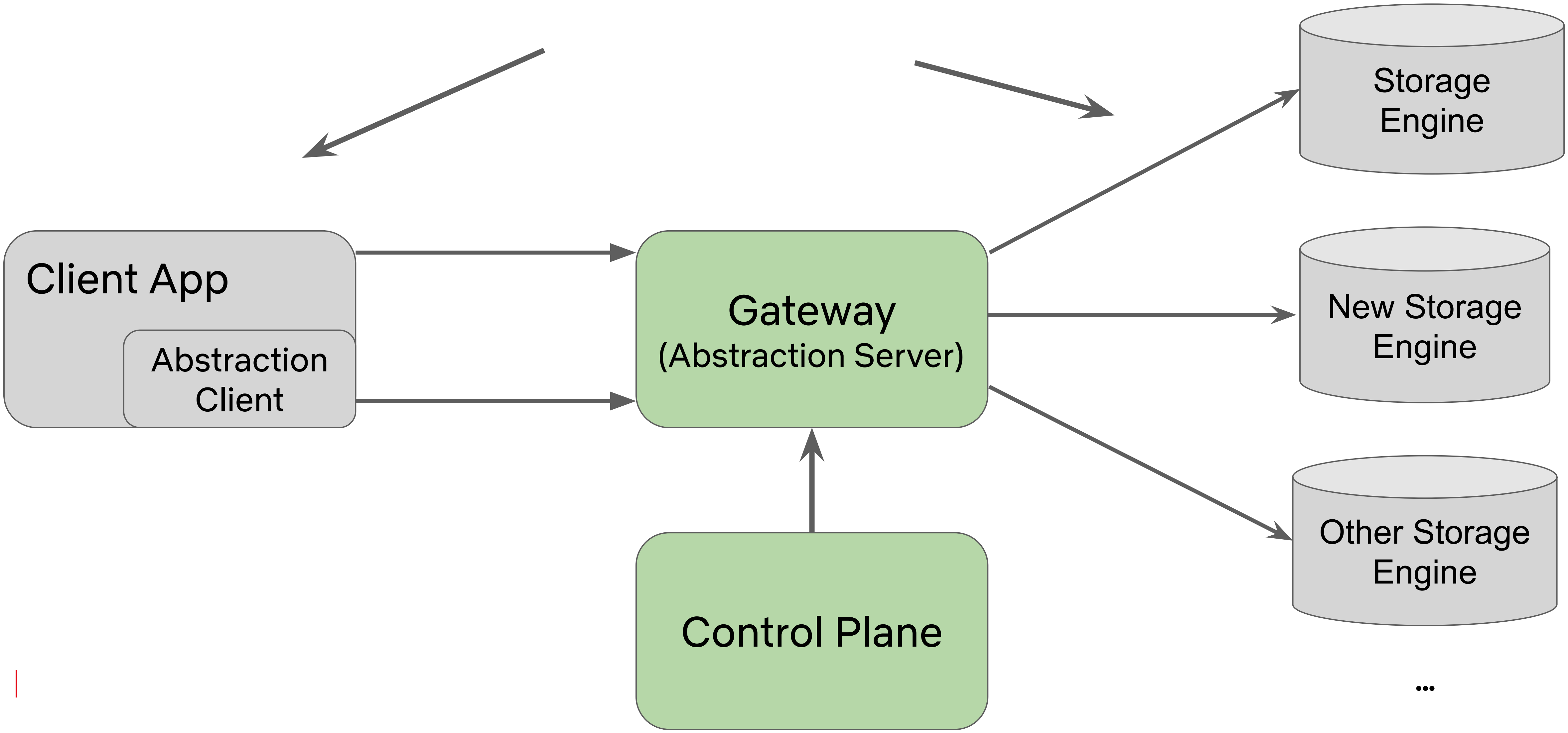
❑ **Configuration**

Virtualization



Data Gateway

Abstract **Clients**



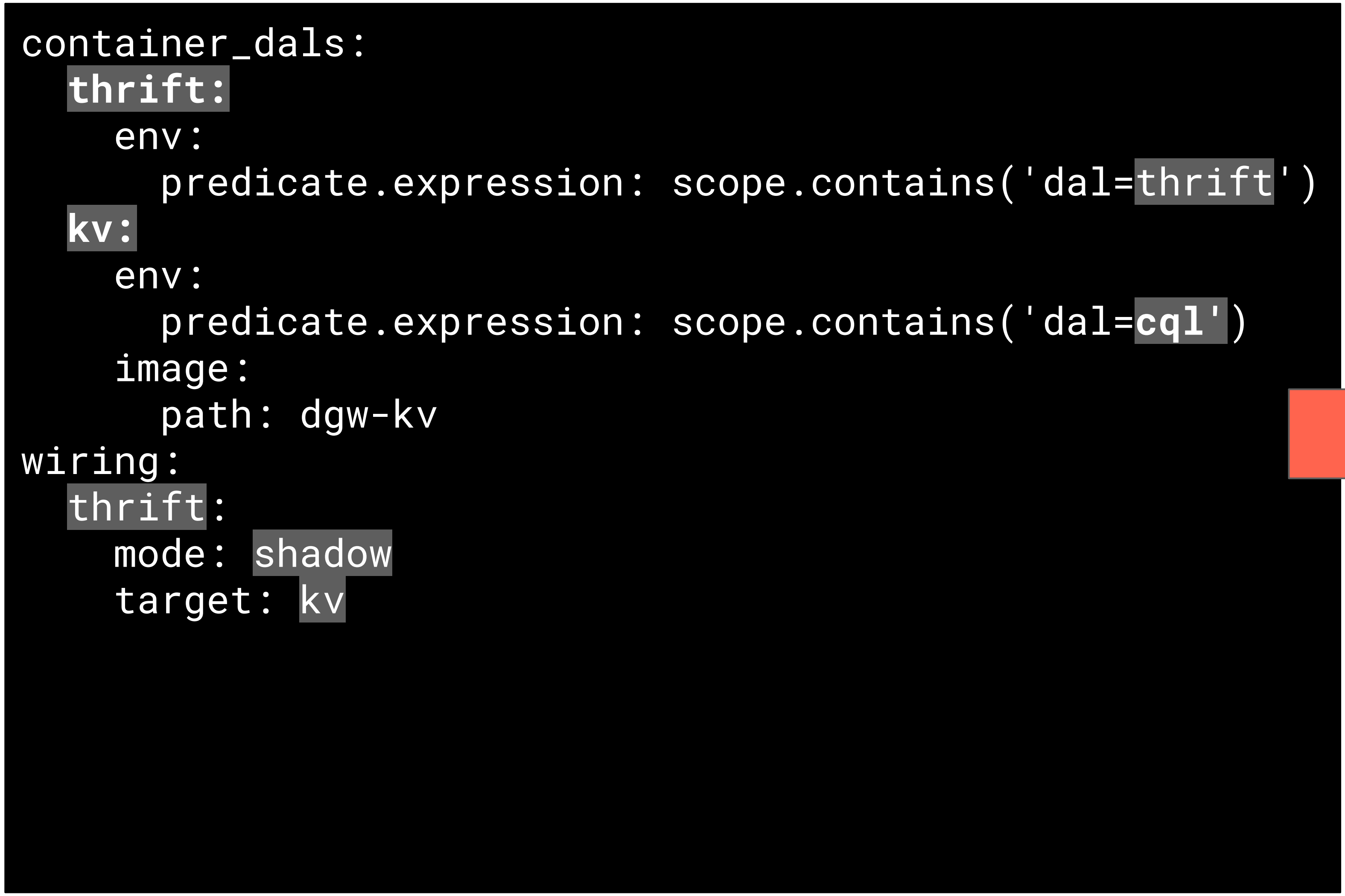
Configuration for composition

Write down - how to compose services

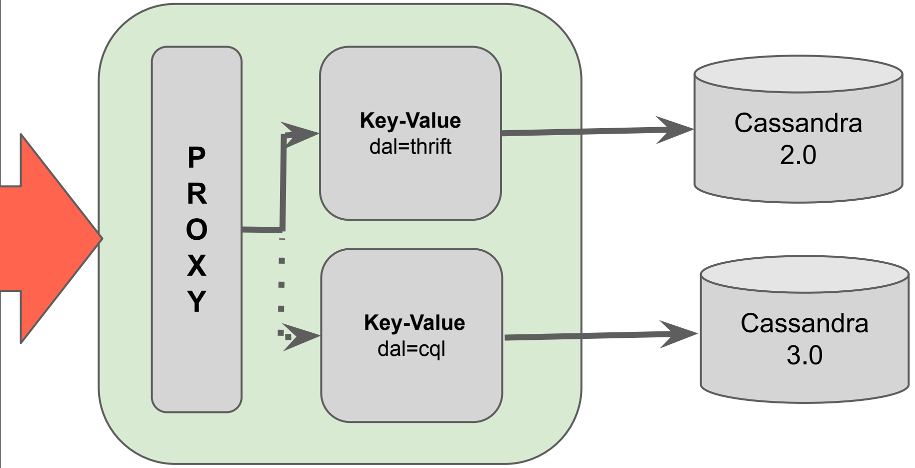
Use configuration to deploy

Deployment Configuration

Runtime Configuration



```
container_dals:
  thrift:
    env:
      predicate.expression: scope.contains('dal=thrift')
  kv:
    env:
      predicate.expression: scope.contains('dal=cql')
    image:
      path: dgw-kv
wiring:
  thrift:
    mode: shadow
    target: kv
```



Control Plane

Persistence Configuration

```
{
 "namespace_name": "<namespace>",
 "persistence_configurations": {
  "persistence_configuration": [{
    "id": "PRIMARY_STORAGE",
    "version": 2,
    "level": 4,
    "scope": "dal=kv",
    "physical_storage": {

       …
     },
     “config”: {
       …
     }
}
```

Persistence Configuration Contd..

```
"physical_storage": {
 "type": "CASSANDRA",
 "cluster": "<cass_cluster>",
 "dataset": "<keyspace>",
 "table": "<table>",
 "schema_id": "kv:cassandra:desc"
},

"config": {
 "consistency_scope": "LOCAL",
 "consistency_target":"READ_YOUR_WRITES",
 "contacts": "<dev@netflix.com>",
 "context": "Device history service"
}
```

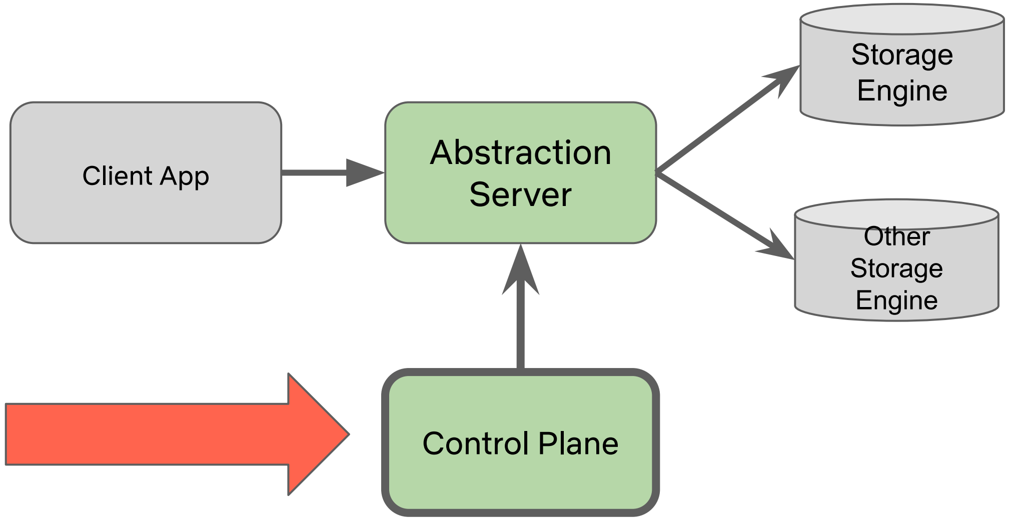
```
"physical_storage": {
  "type": "EVCACHE",
  "cluster":"<cache_cluster>",
  "table": "<cache_name>",
}
```

Control Plane

Persistence Configuration

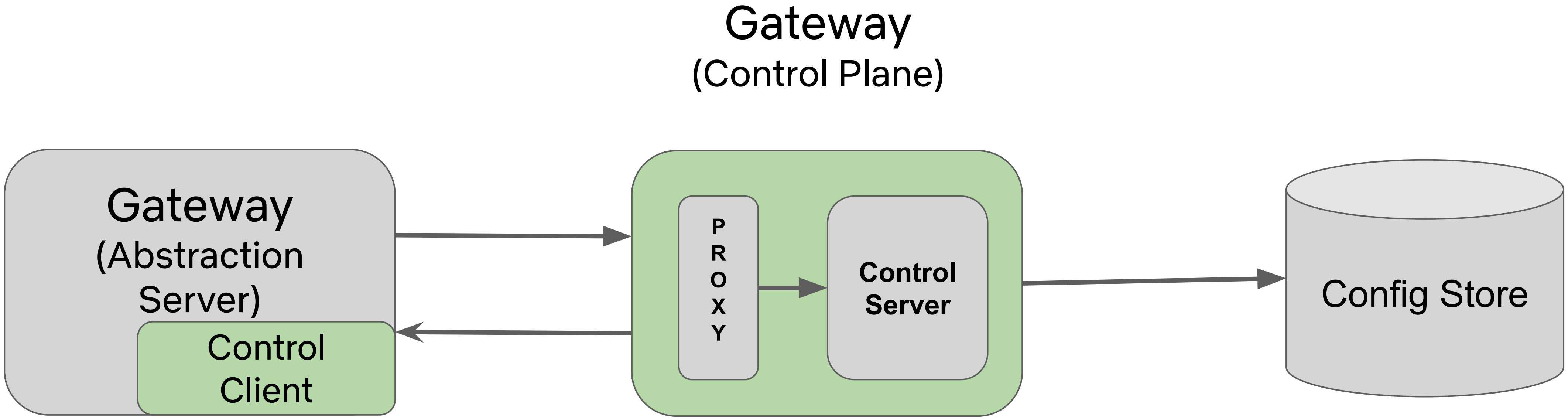
```
{ "version": "5",
  "namespaces": [{
      "namespace_name": "interstitial_feedback",
      "persistence_configurations": {
        "persistence_configuration": [{
            "id": "PRIMARY_STORAGE",
            "version": 1,
            "level": 4,
            "scope": "dal=kv",
            "physical_storage": {
              "type": "CASSANDRA",
              "cluster": "cass_dgw_kv_interstitial_feedback",
              "dataset": "interstitial_feedback",
              "table": "interstitial_feedback",
              "schema_id": "kv:cassandra"
            },
            "config": {
                "consistency_scope": "LOCAL",
                "consistency_target": "READ_YOUR_WRITES",
                "default_ttl": 5184000,
                "enforce_max_ttl": 5184000
              }
        }]},
      "capabilities": ["ALL"],"status": "ACTIVE",
      "create_ts": "2023-07-14T21:14:30Z",
}]}
```

```
grpc -a dgwcontrol.kv WatchNamespaces -d "
{
  "shard_identity": "<shard_identity>"
  "last_seen_version": 0
}"
```



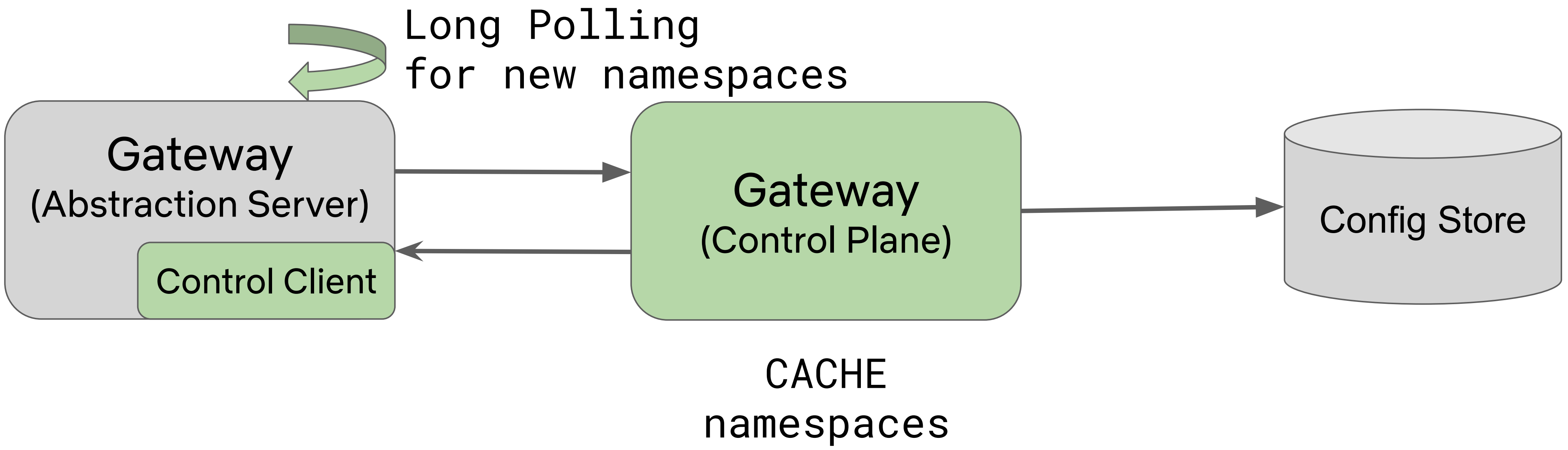
Control Plane

"Configuration as service"



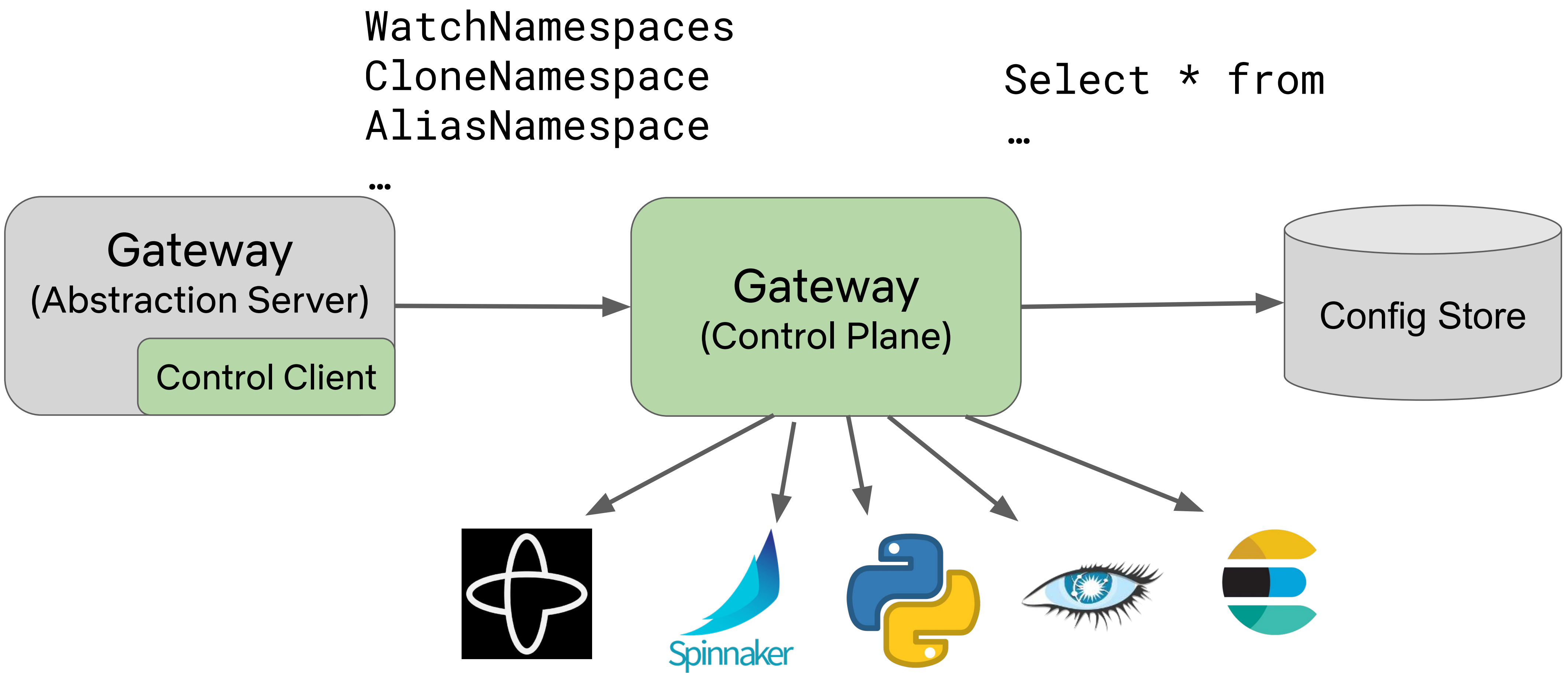
Control Plane

"Configuration as service"



Control Plane

"Configuration as service"



```
WatchNamespaces
CloneNamespace
AliasNamespace
…
```

```
Select * from
…
```

Control Plane Apis

```
rpc WatchNamespaces(

 String shard_identity;
 long last_seen_version;

) ->
WatchNamespacesResponse
```

```
message WatchNamespacesResponse(

 List<Namespace> namespaces=1;
 int version=2;

)
```

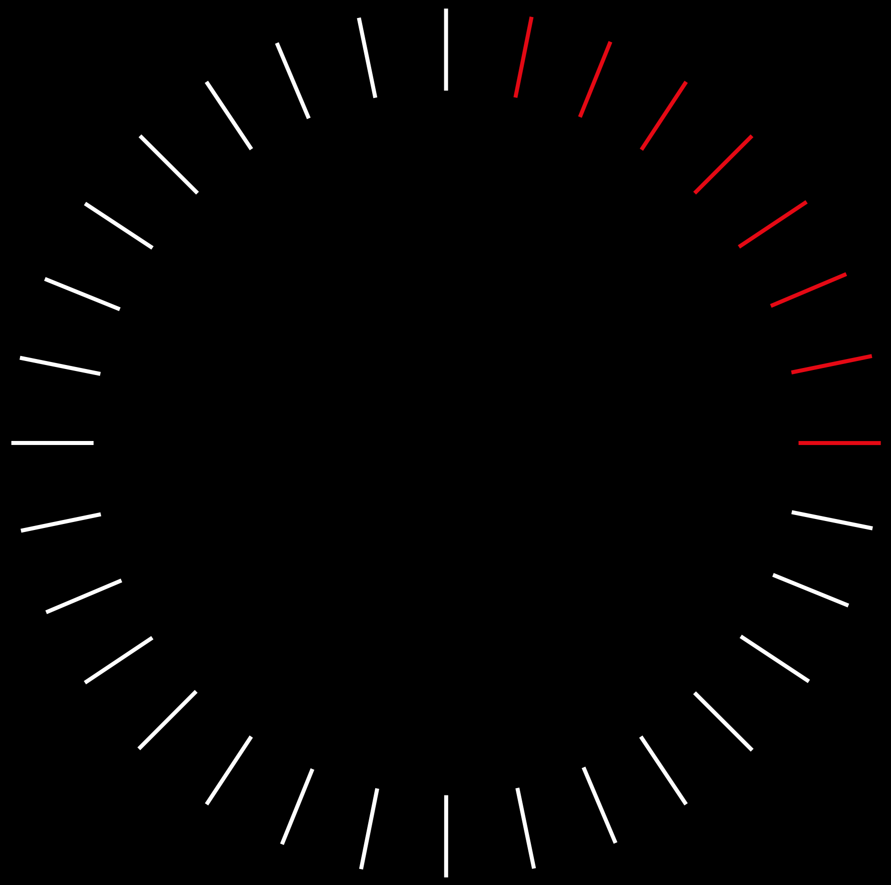
Control Plane Apis

```
rpc CloneNamespaces(

 String source;
 String target;

) ->
CloneNamespacesResponse
```

```
message CloneNamespacesResponse(

 String job_id=1;
 Status status=2;

)
```

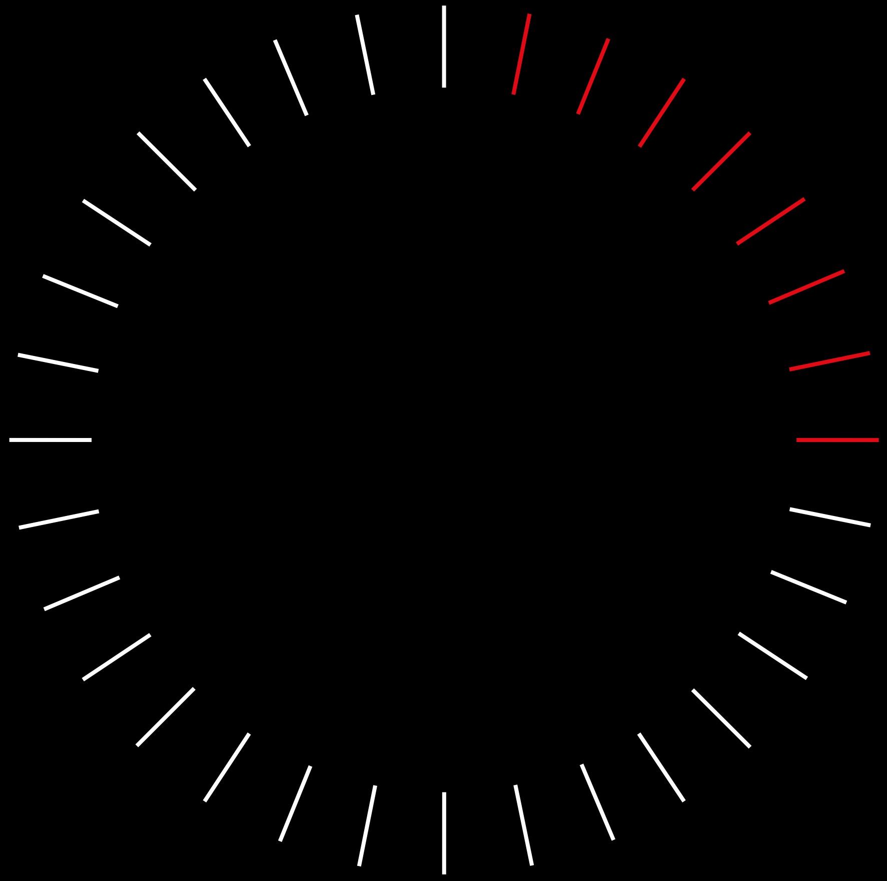
- ✓ **Virtualization**
- □ **Abstraction**
- □ Clean APIs

Virtualization



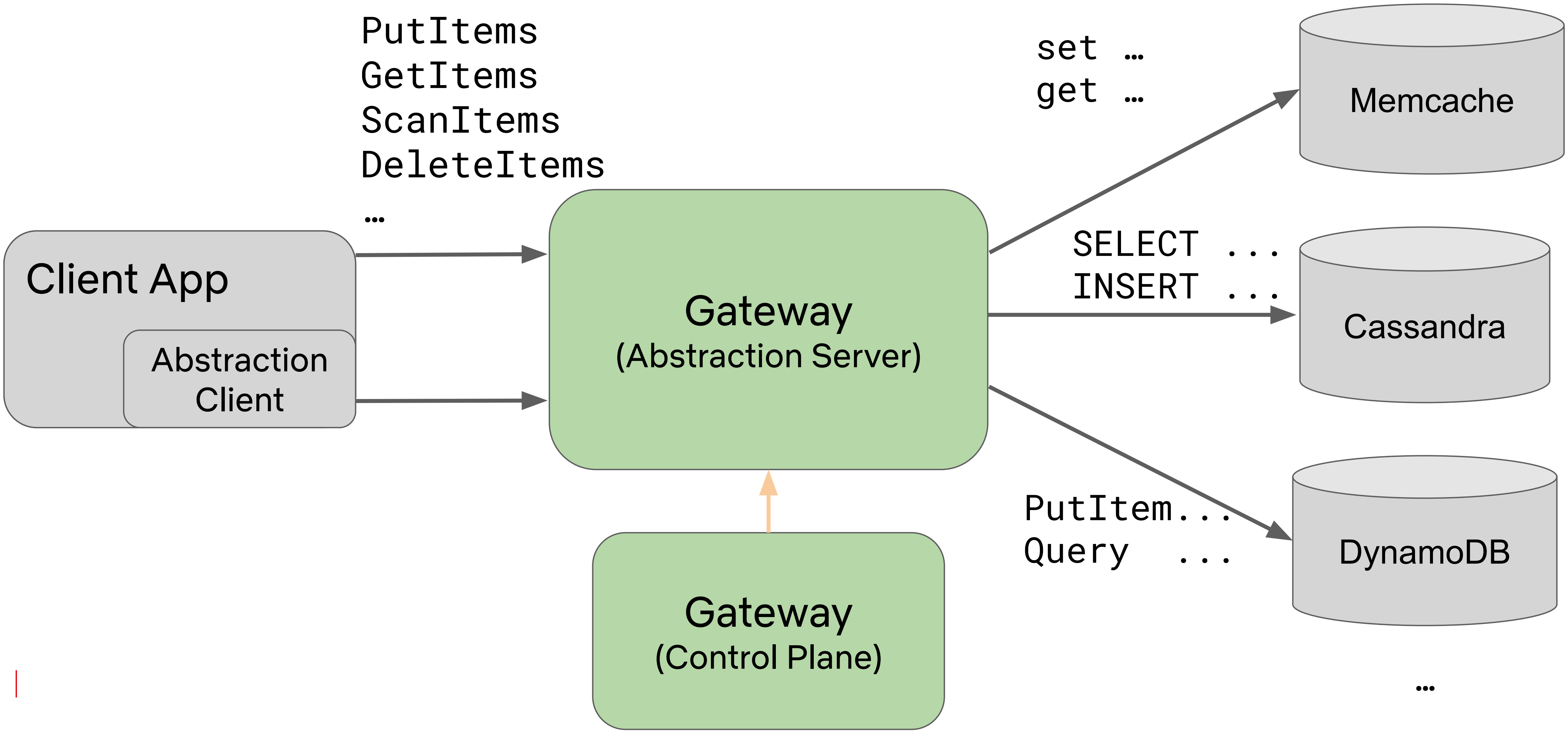
Abstraction

- □ **Storage Agnostic**
- □ Dual Writes & Data Migration



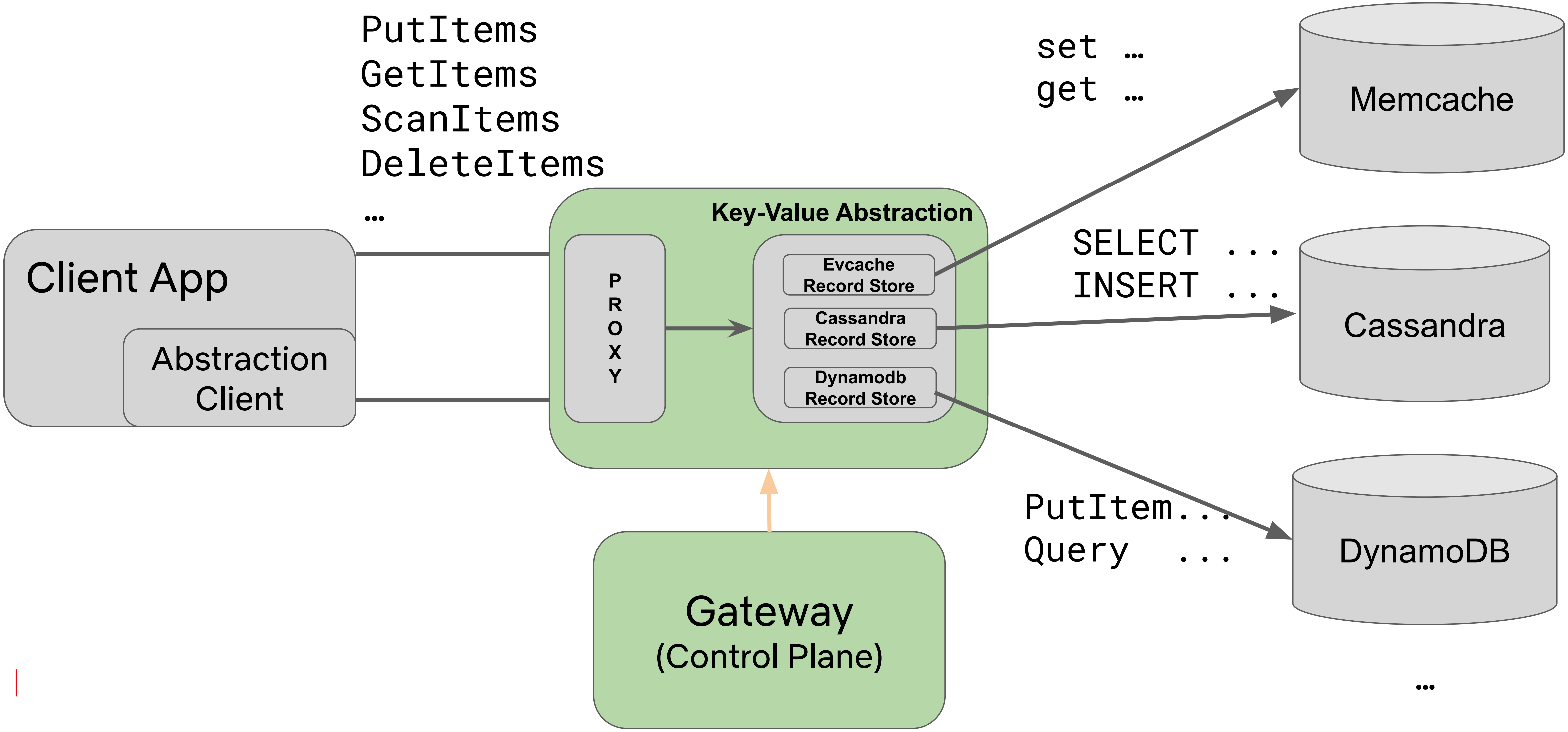
Key-Value

"Two-Level Map as a Service"



Key-Value

"Two-Level Map as a Service"



Control Plane

Namespace Persistence Configuration

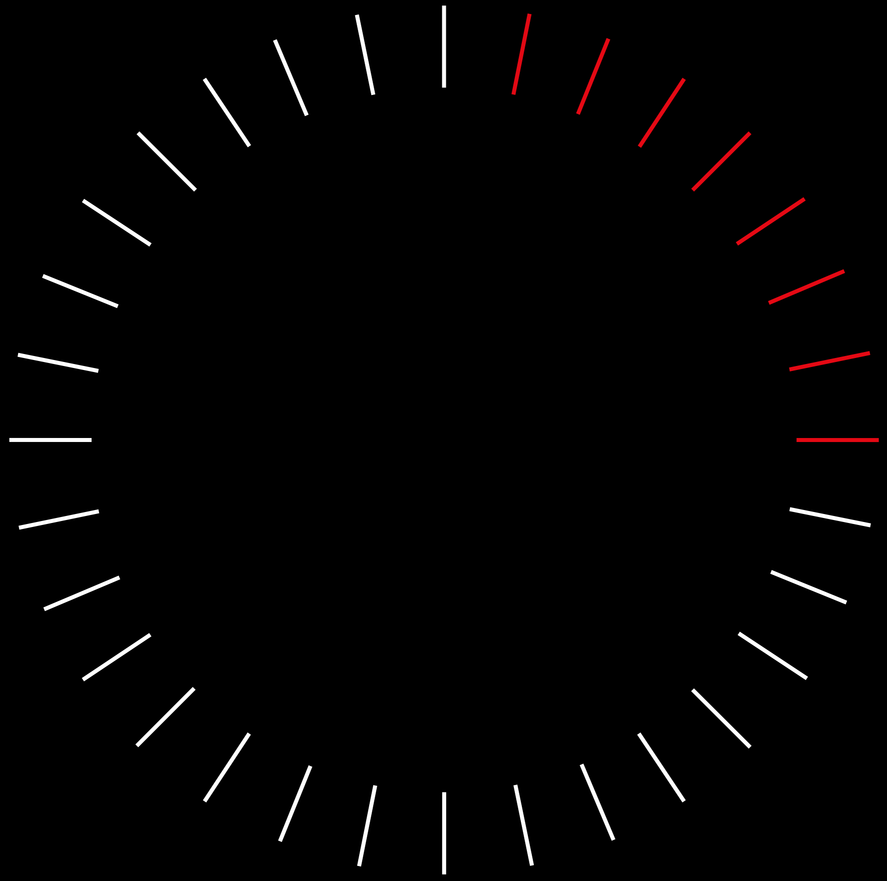
```
{"namespace_name": "ns1",
 "persistence_configurations": {
  "persistence_configuration": [{
   "id": "PRIMARY_STORAGE",
   "version": 1,
   "level": 4,
   "scope": "dal=cql",
   "physical_storage": {
    "type": "CASSANDRA",
    "cluster": "<cass_cluster>",
    "dataset": "<keyspace>",
    "table": "<table>",
    "schema_id":
   "kv:cassandra:desc"
   }}]
 }}
```

```
{"namespace_name": "ns2",
 "persistence_configurations":{
  "persistence_configuration":[{
   "id": "PRIMARY_STORAGE",
   "version": 1,
   "level": 4,
   "scope": "dal=dynamodb",
   "physical_storage": {
    "type": "DYNAMODB",
    "table": "<table>",
    "schema_id":
   "kv:dynamodb:desc"
   }}]
 }}
```

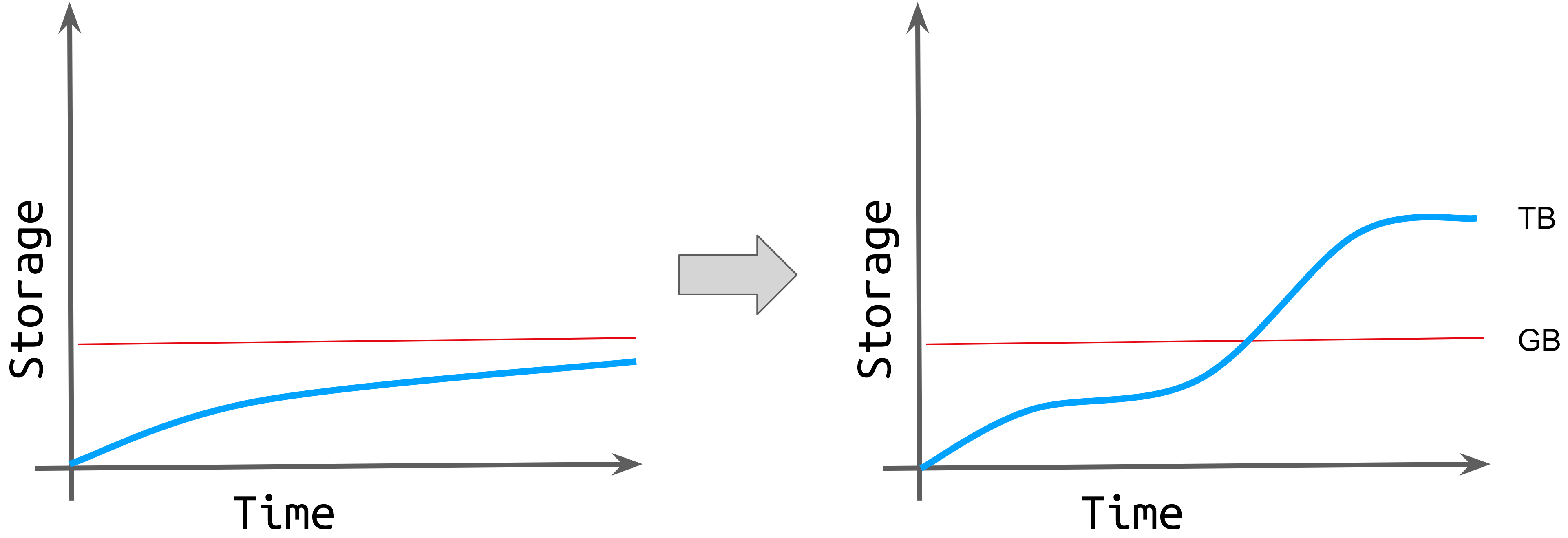
Abstraction

✓ Storage Agnostic

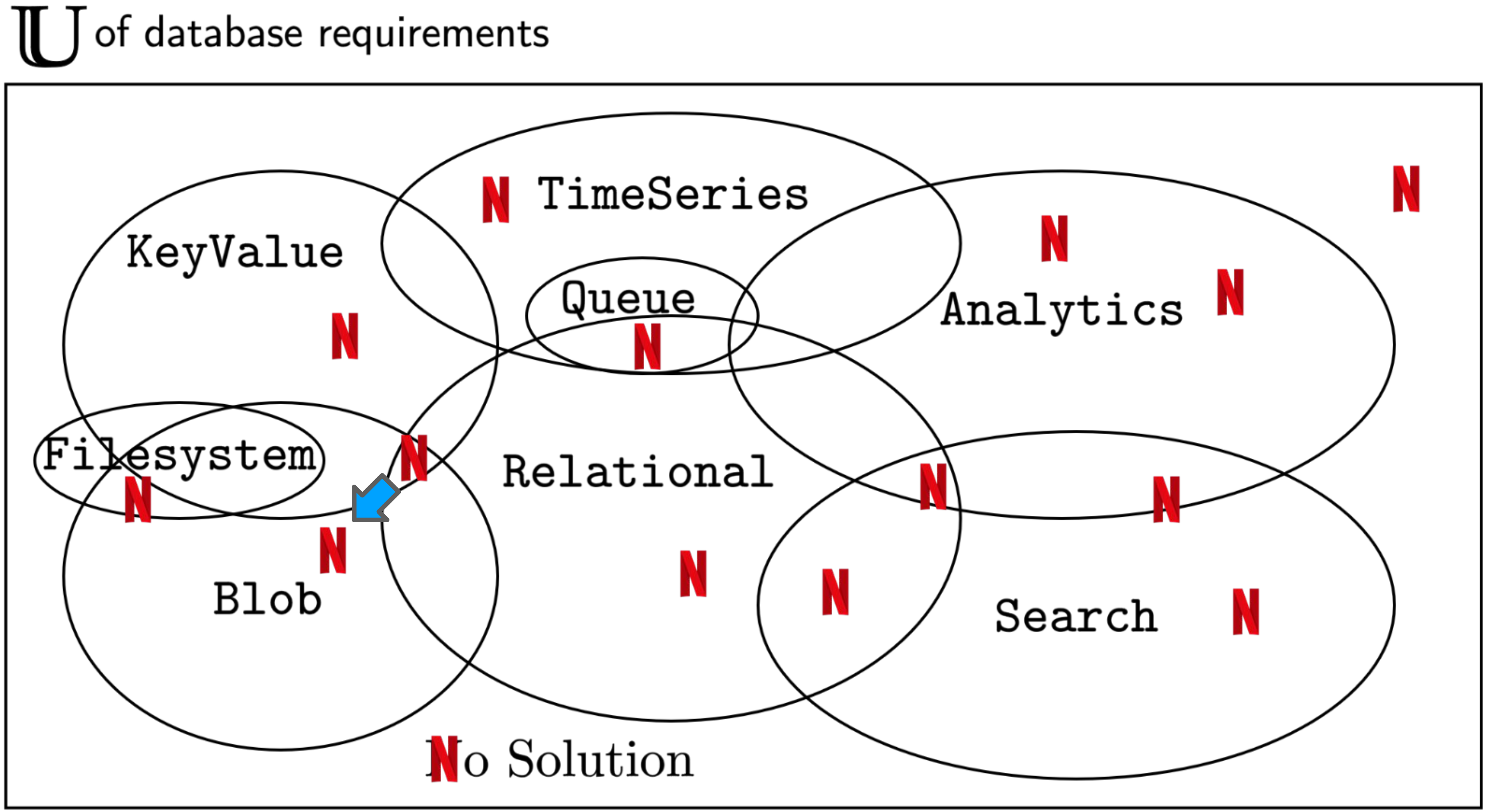
□ **Dual Writes & Data Migration**



Use case: Change overtime

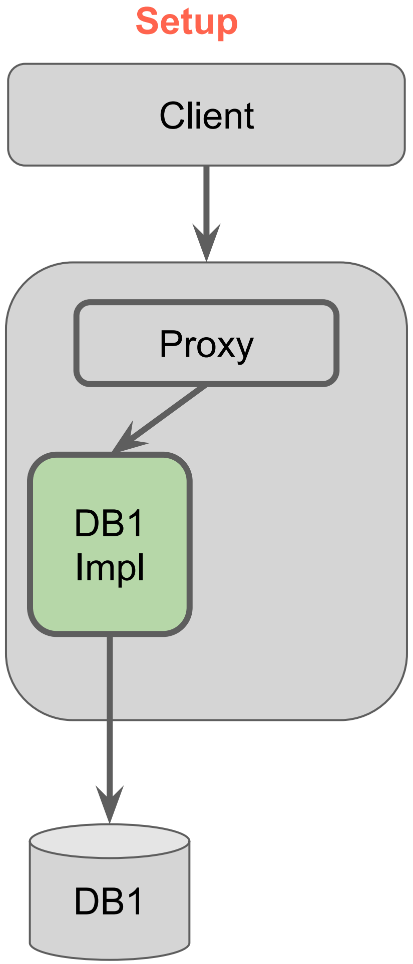


Problem: Varied Use-Cases



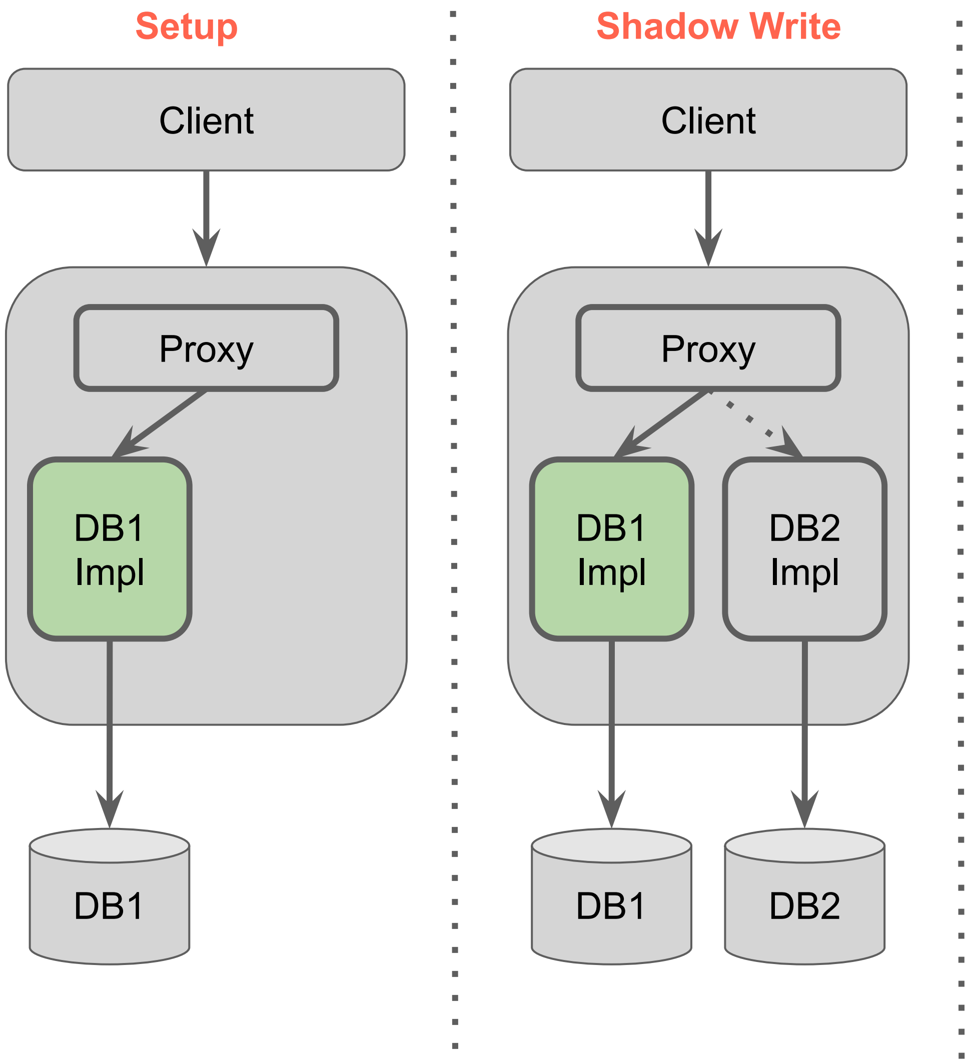
Data Gateway

Data Migrations



Data Gateway

Data Migrations

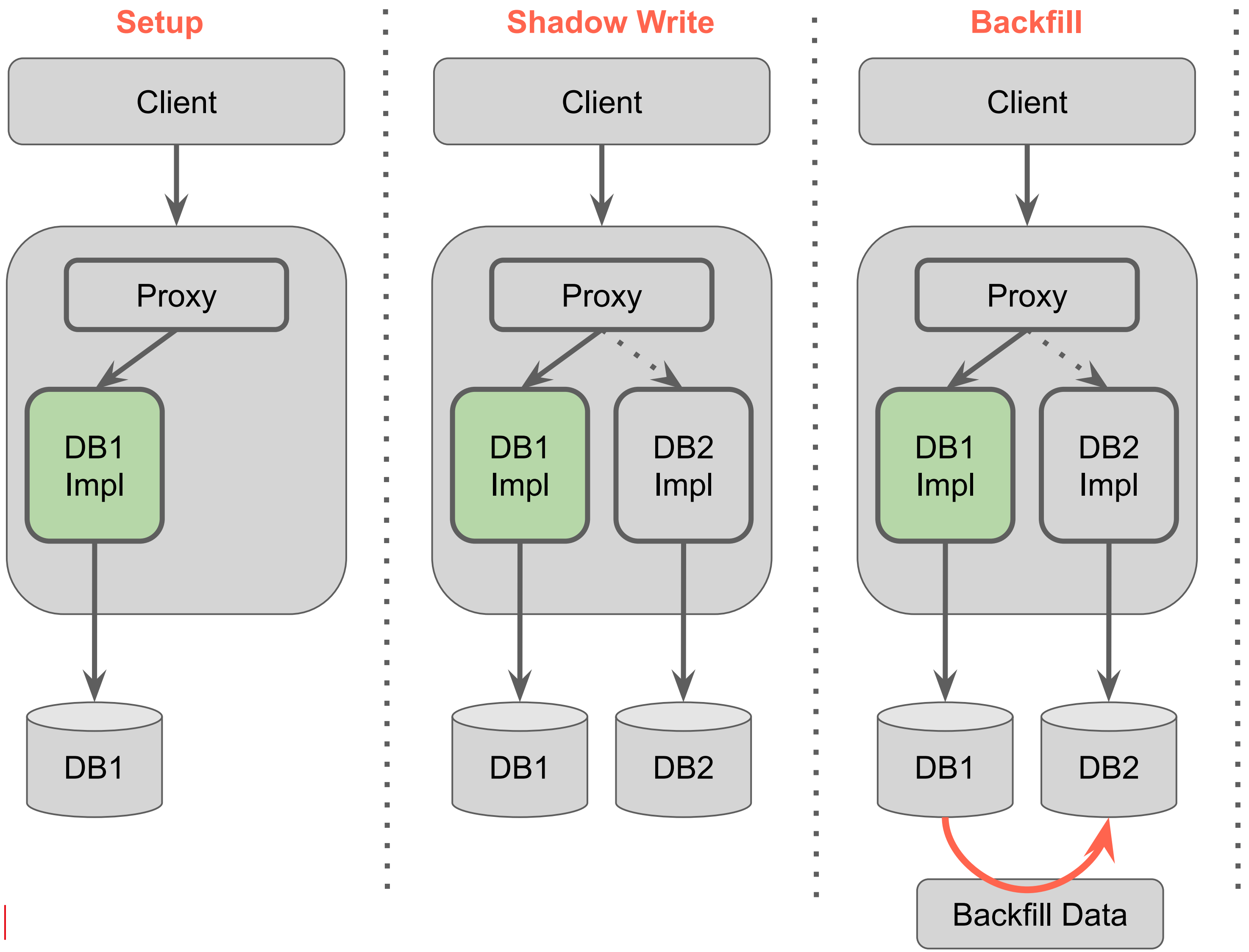


Setup

Shadow Write

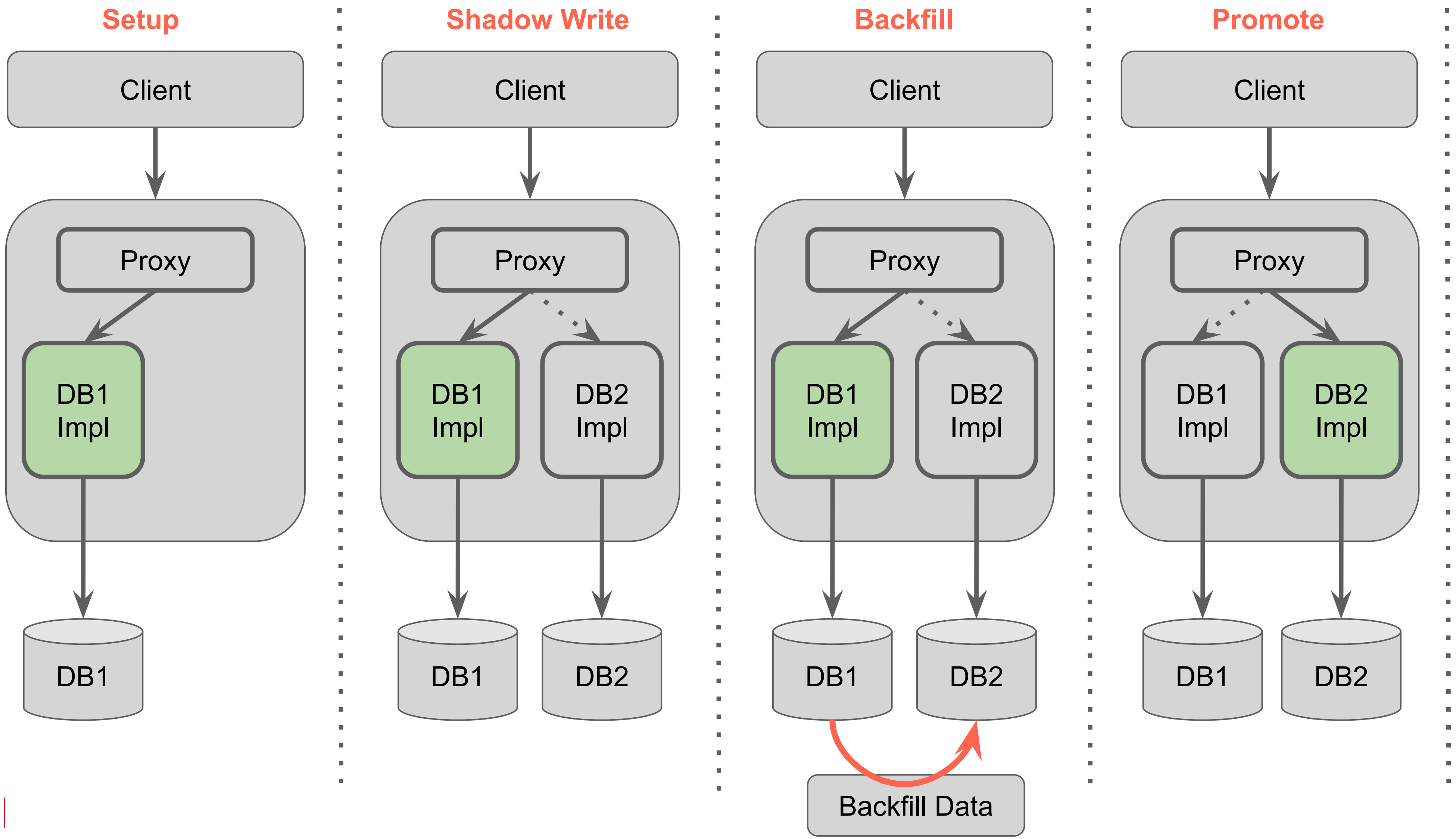
Data Gateway

Data Migrations



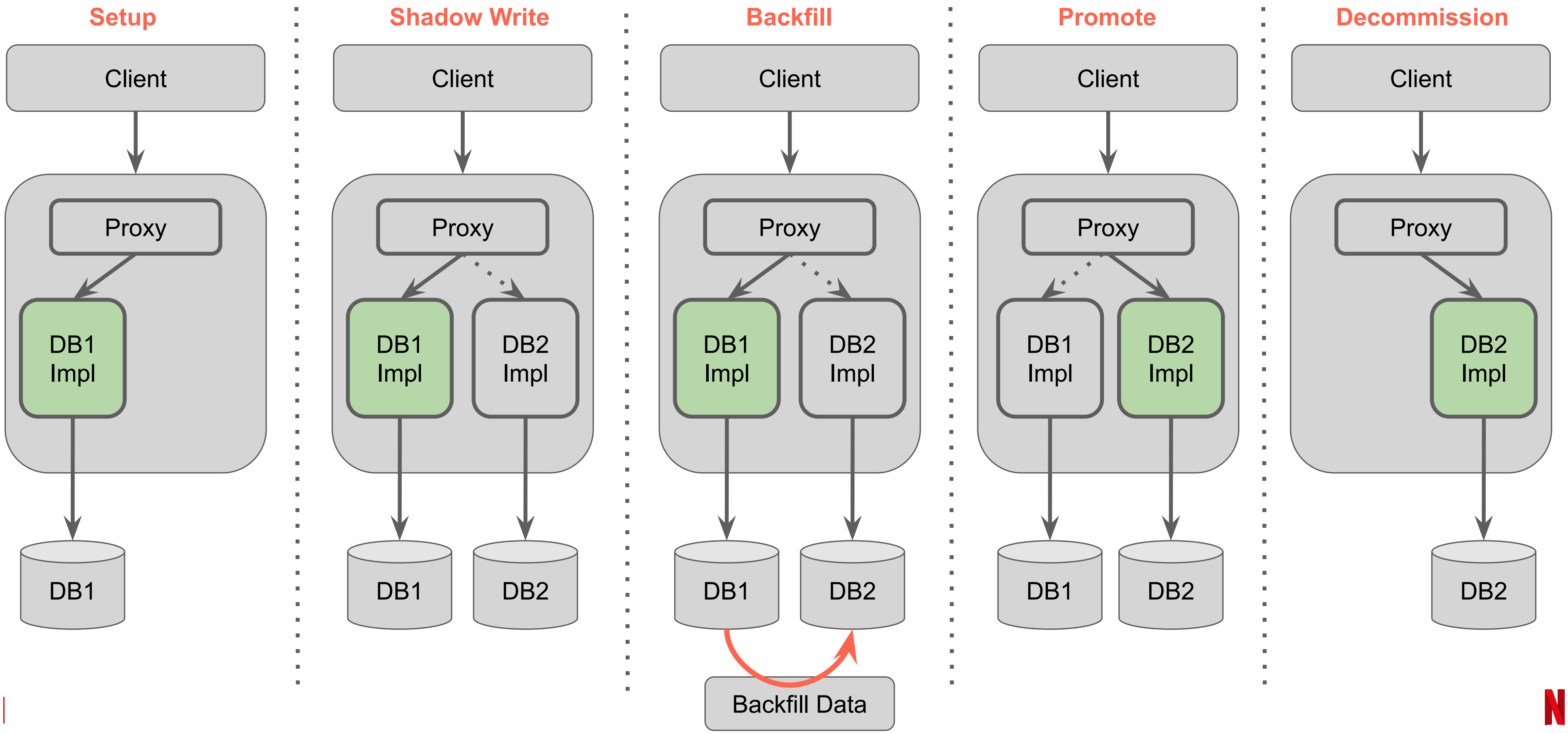
Data Gateway

Data Migrations



Data Gateway

Data Migrations



Control Plane Persistence Configurations for dual writes

```
{"namespace_name": "ns1",
 "persistence_configurations": {
  "persistence_configuration": [{
   "id": "PRIMARY_STORAGE",
   "version": 1,
   "level": 4,
  "scope": "dal=cql",
  "physical_storage": {
   "type": "CASSANDRA",
   "cluster": "<cass_cluster>",
   "dataset": "<keyspace>",
   "table": "<table>",
   "schema_id":
  "kv:cassandra:desc"
  },
```

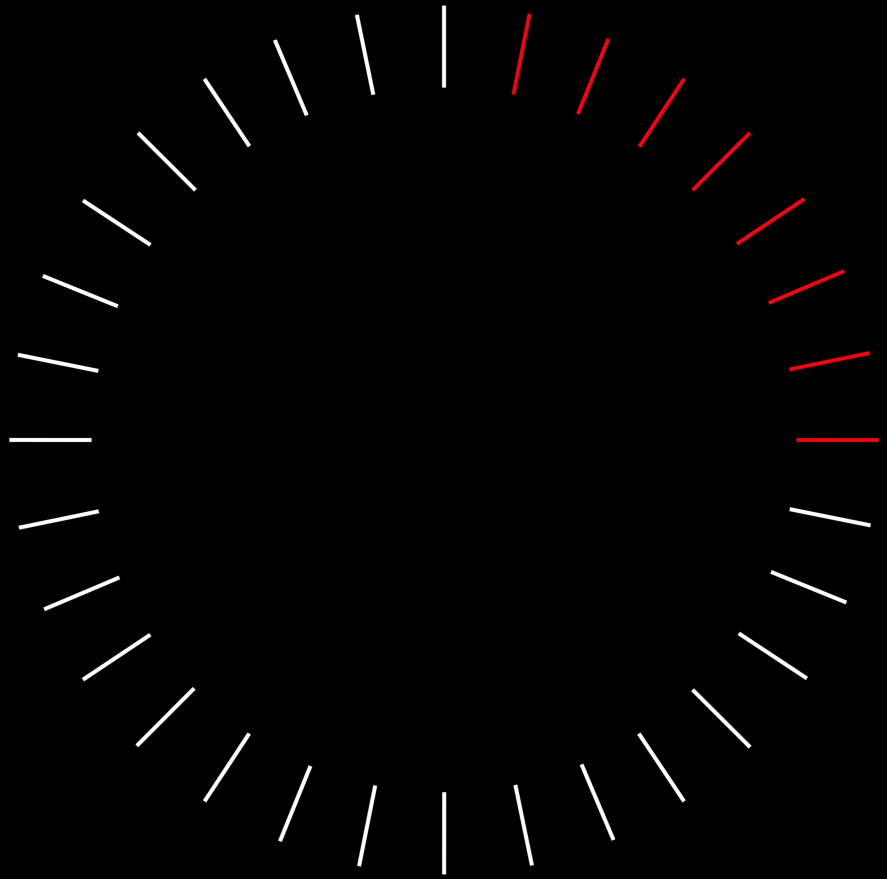
1

```
{
  "id": "PRIMARY_STORAGE",
  "version": 1,
  "level": 4,
 "scope": "dal=dynamodb",
 "physical_storage": {
  "type": "DYNAMODB",
  "table": "<table>",
  "schema_id":
 "kv:dynamodb:desc"
 }}]
}}
```

2

- ✓ Virtualization
- ✓ Abstraction
- □ Clean APIs

Clean APIs

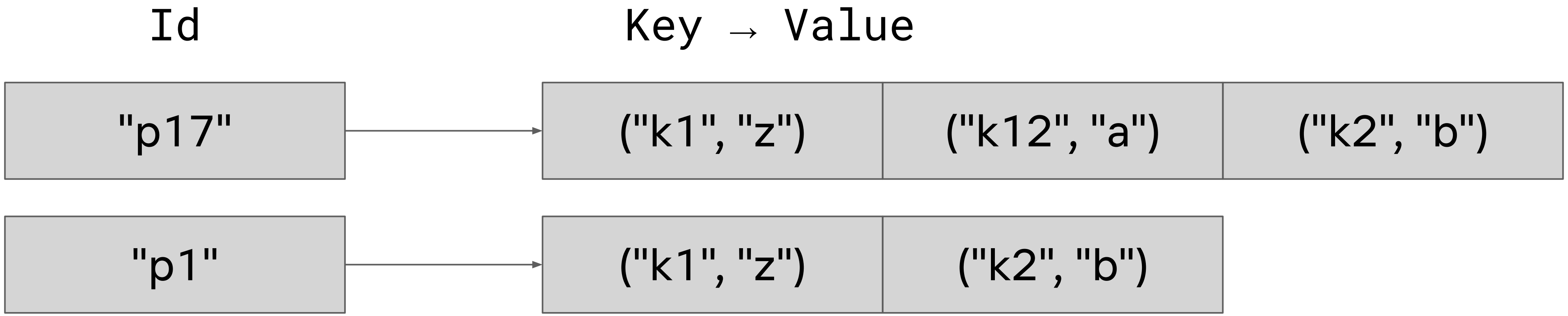


Key-Value Abstraction APIs

```
PutItems -> Trilean

GetItems -> List[Item]

MutateItems -> Boolean

ScanItems -> BlockingQueue[Map[Id, List[Item]]]

PutIfAbsent -> Optional[Boolean]

Compute -> Optional[Item]
```

Key-Value Abstraction APIs

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

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



Id

Key → Value

"p17" → ("k1", "z") ("k12", "a") ("k2", "b")

"p1" → ("k1", "z") ("k2", "b")

Key-Value Storage Layout

Base Table

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

```
CREATE TABLE IF NOT EXISTS <keyspace>.<table_name> (
  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

```
Map[String, List[Item]]

message Item {
    bytes key = 1;
    bytes value = 2;
    Metadata value_metadata = 3;
    int32 chunk = 4;
}
```

Key-Value

```
message Metadata {

  // How this data is compressed
  CompressionMetadata compression = 1;

  // Written in time, Expires at time etc.
  LifecycleMetadata lifecycle = 2;

  // If this data references other chunks of data
  ChunkMetadata chunks = 3;

  // Encoding and content metadata, headers
  ContentMetadata content = 4;

}
```

Key-Value

```
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; …
}
```

Key-Value

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

```
rpc MutateItems(

   IdempotencyToken token,
   String namespace,
   String id,
   List[MutationRequest]
     mutations, …

) -> MutationResponse
```

```
MutationRequest {

  oneof {
      PutItems       put;
      DeleteItems delete;
  }

}
```

Key-Value

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
}
```

Key-Value 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;

}
```

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)
```

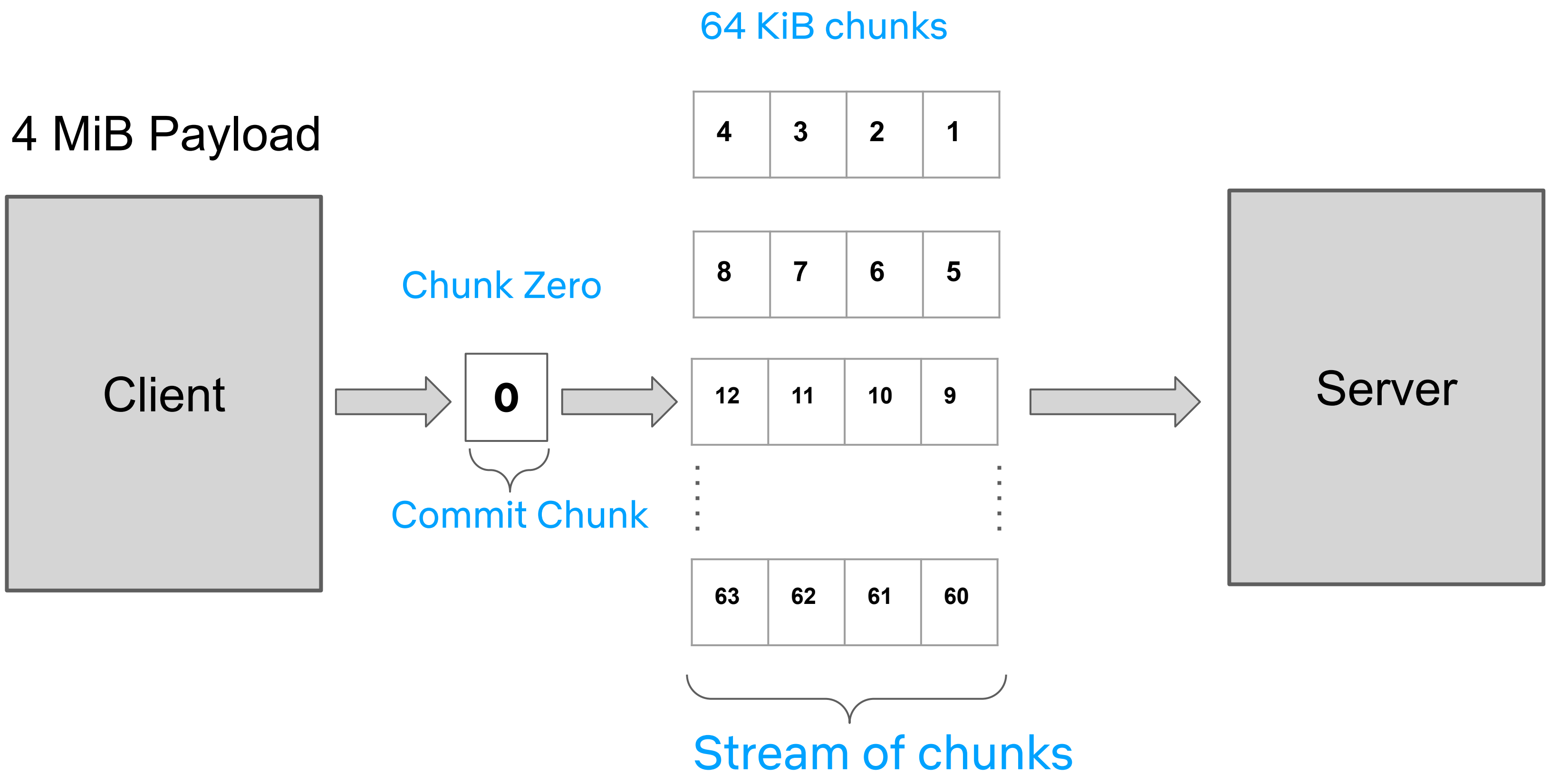
Small data

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))
```

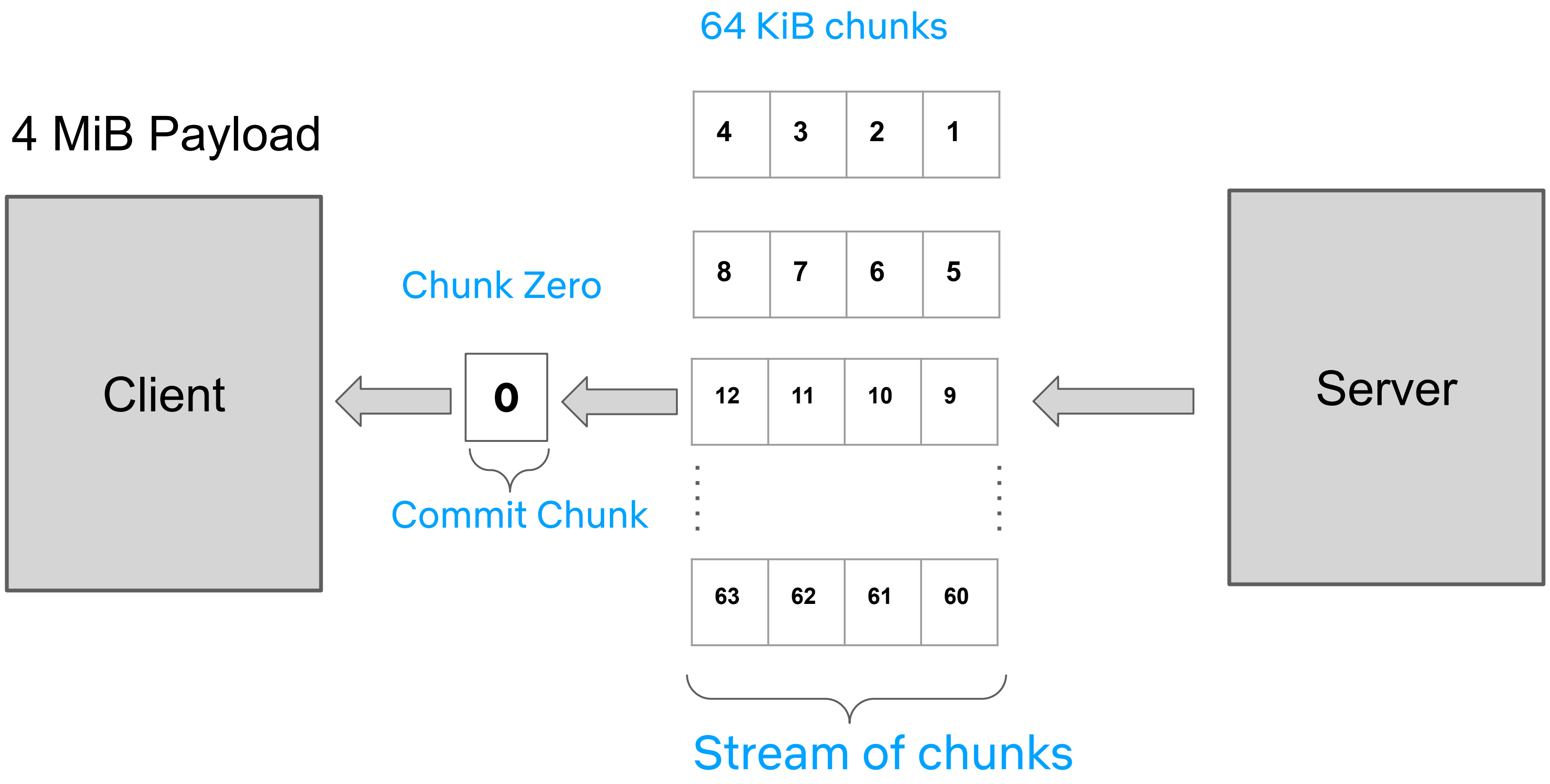
Chunk Data

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
```

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

Chunk Data

```
# 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
```

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!

Key-Value Storage Layout

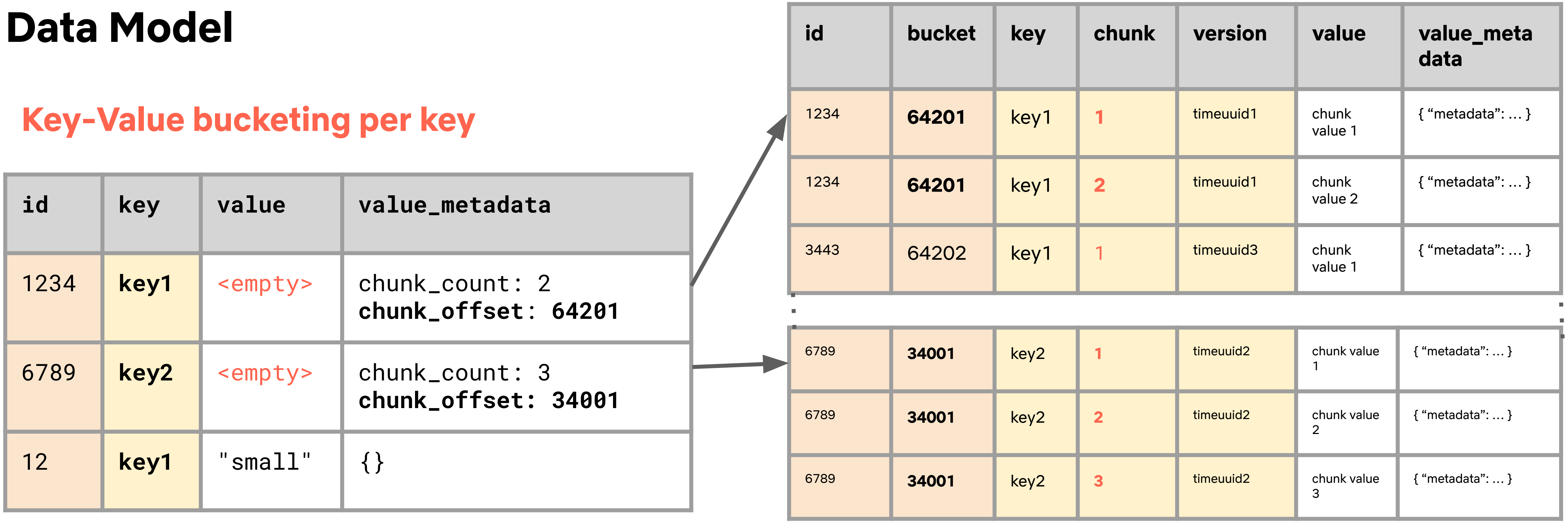
Data Table

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_meta data
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": ... }

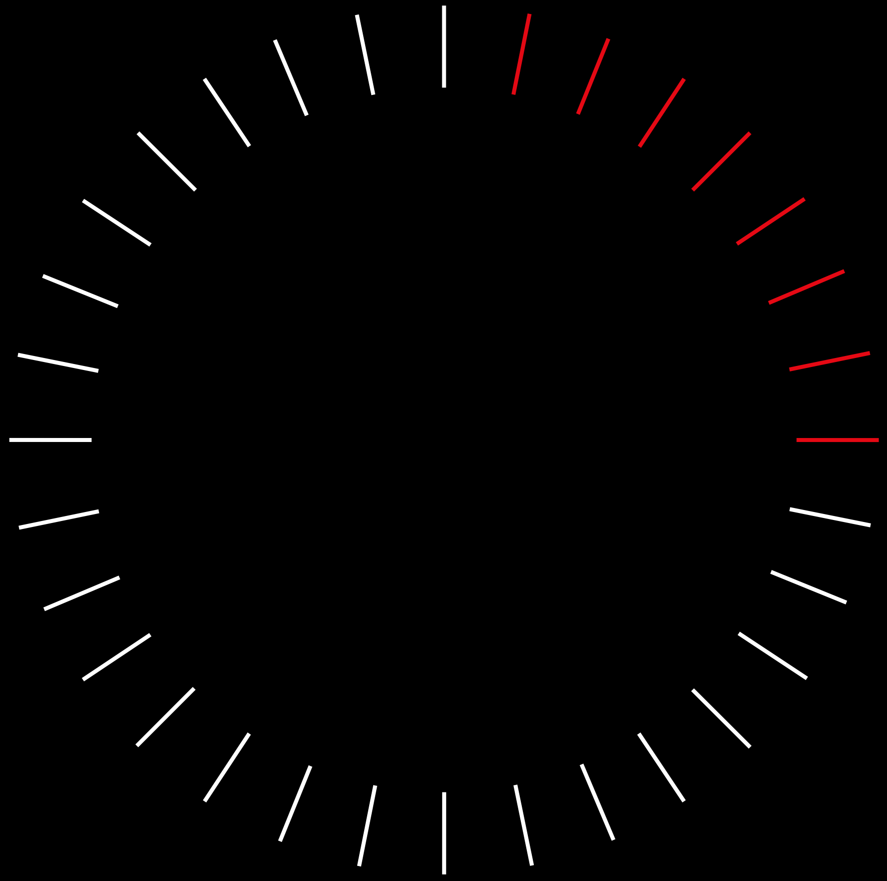
```
"chunks": {
    "chunk_size_bytes": 65536,
    "chunk_count": 2,
    "chunk_offset": 124431,
    "chunk_token_xxh64": "12595968139585192830",
    "version": "3d52c5c0-6229-11ee-aecd-de49621d037e"
 }
```

✓ Virtualization

✓ Abstraction

✓ Clean APIs

Clean APIs



Building Blocks

Chunking

Adaptive Pagination

Compression

Dictionary Compression

Caching

Signaling

SLO Signaling

Summarization

Nearline Caching

Building Blocks

Chunking

Adaptive Pagination

Compression

Dictionary Compression

Caching

Signaling

SLO Signaling

Summarization

Nearline Caching

Abstraction with many tunable features

Timeseries

Counter

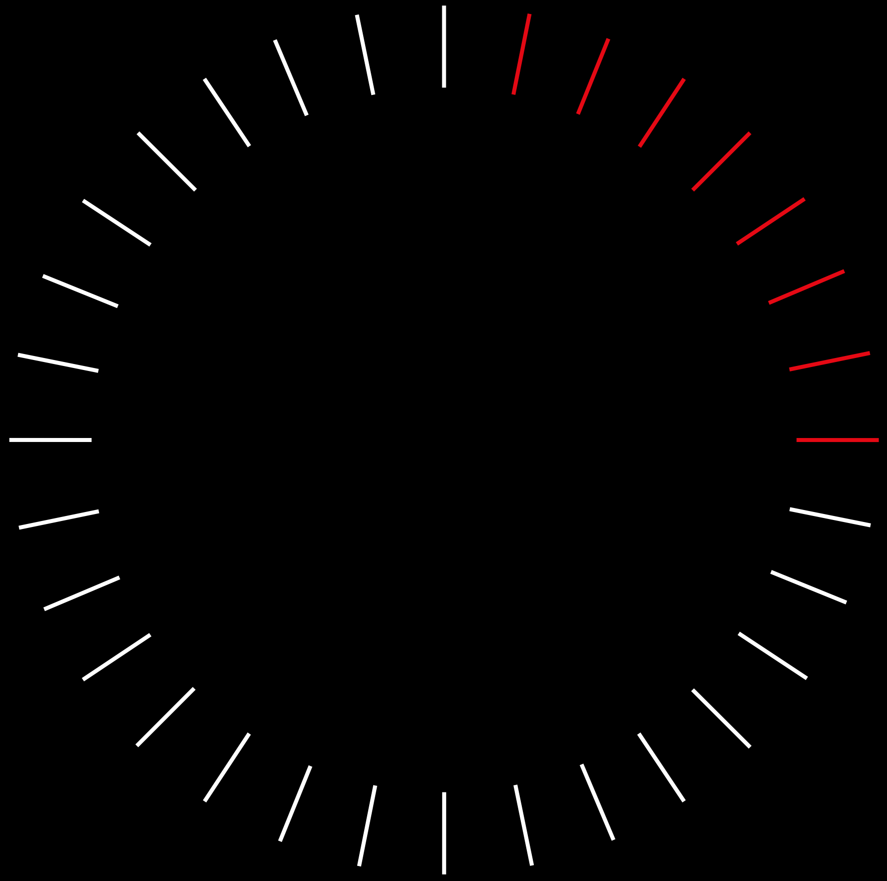
Identifier

Entity

Tree

Graph

More Abstraction



Every new layer of abstraction is a new chance for a **clean-slate** redesign of everything, making everything a **little faster**, less power hungry, more elegant, **easier to use**, cheaper."

- Marc Andreessen

Thank You.

Vidhya Arvind
varvind@netflix.com