redislabs
HOME OF REDIS

RedisGears – Redis in memory data processing

JUNE 2019 | PIETER CAILLIAU

About me

- Produced in Belgium

- (instanceof) SE @ TomTom
- Consultant @ neo4j
- Solution Architect @ Redis Labs
- Product Manager @ Redis Labs

- @cailliaup

Agenda

What is Redis and Redis Enterprise

Stream Processing with RedisGears

RedisGears as a Multimodel Engine

Redis is Fast …

… Extremely Fast

DB-Engines Ranking

345 systems in ranking, September 2018

Rank Sep 2018	Rank Aug 2018	Rank Sep 2017	DBMS	Database Model	Score Sep 2018	Score Aug 2018	Score Sep 2017
1.	1.	1.	Oracle	Relational DBMS	1309.12	-2.91	-49.97
2.	2.	2.	MySQL	Relational DBMS	1180.48	-26.33	-132.13
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1051.28	-21.37	-161.26
4.	4.	4.	PostgreSQL	Relational DBMS	406.43	-11.07	+34.07
5.	5.	5.	MongoDB	Document store	358.79	+7.81	+26.06
6.	6.	6.	DB2	Relational DBMS	181.06	-0.78	-17.28
7.	↑ 8.	↑ 10.	Elasticsearch	Search engine	142.61	+4.49	+22.61
8.	↓ 7.	↑ 9.	Redis	Key-value store	140.94	+2.37	+20.54
9.	9.	↓ 7.	Microsoft Access	Relational DBMS	133.39	+4.30	+4.58
10.	10.	↓ 8.	Cassandra	Wide column store	119.55	-0.02	-6.65

Snapchat

And you've been using it already

GitHub

Redis is Extensively and Diversely Used

twitter

Uses Redis for:
Timeline, following

Scope: 10-20 TB

Snapchat

Uses Redis for:
All messages

Scope: 40 TB

tinder

Uses Redis for:
Geo search, user profiles

Scope: 10-20 TB

GitHub

Uses Redis for:
Repository router

Scope: 10+ TB

stackoverflow

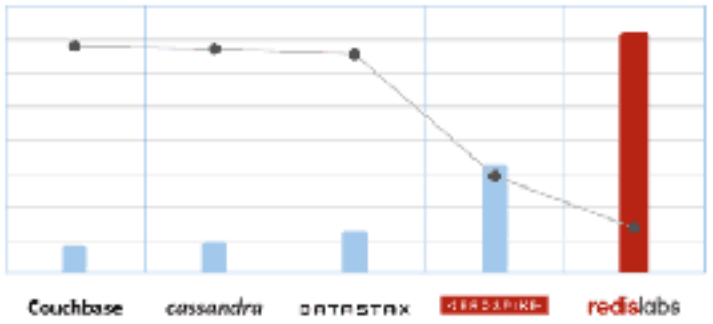
Uses Redis for:
Local/site/global caching

Redis Top Differentiators

1

Performance

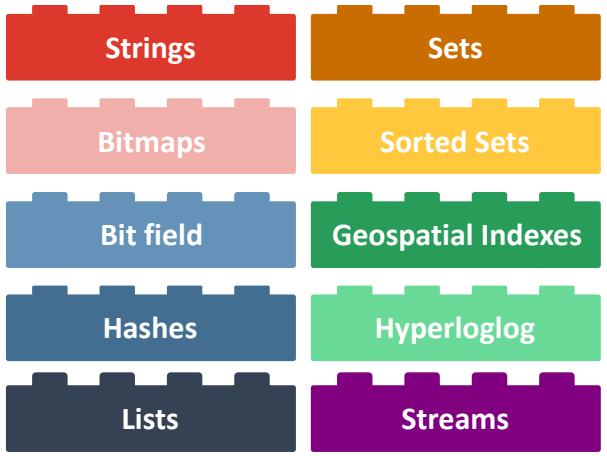
NoSQL Benchmark



2

Simplicity

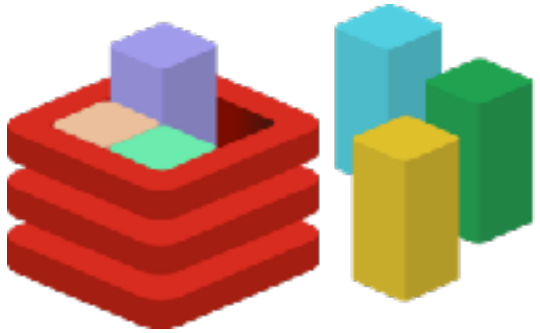
Redis Data Structures



3

Extensibility

Redis Modules



redislabs
HOME OF REDIS

Redis Speed differentiators

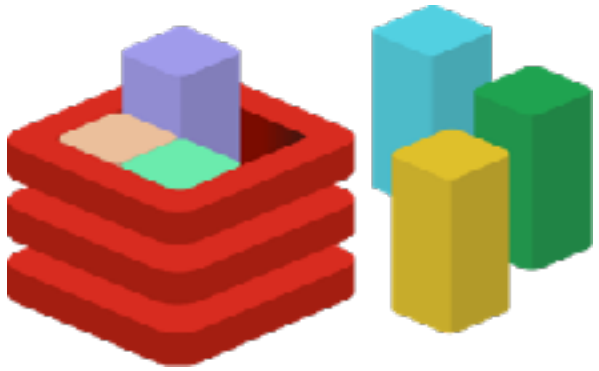
OPTIMIZED ARCHITECTURE	ADVANCED PROCESSING	EFFICIENT OPERATION
✓ Written in C	✓ Most commands are executed with O(1) complexity	✓ Easy to parse networking protocol
✓ Served entirely from memory	✓ Access to discrete elements within objects	✓ Pipelining for reduced network overhead
✓ Single-threaded, lock free	✓ Reduced bandwidth/ overhead requirements	✓ Connection pooling

Redis Speed differentiators

OPTIMIZED ARCHITECTURE	ADVANCED PROCESSING	EFFICIENT OPERATION
✓ Written in C ✓ Served entirely from memory ✓ Single-threaded, lock free	✓ Most commands are executed with O(1) complexity ✓ Access to discrete elements within objects ✓ Reduced bandwidth/ overhead requirements	✓ Easy to parse networking protocol ✓ Pipelining for reduced network overhead ✓ Connection pooling

Modules Extend Redis Infinitely

- Create your own data types and commands
- Reuse Redis' simplicity, performance, scalability and high availability.
- Can be written in C/C++/Go/Python/Rust/Zig
- Leverage existing data structures
- Turn Redis into a Multi-Model database



https://redislabs.com/community/redis-modules-hub/

Redis Modules

RediSearch (GA) redisearch.io

RedisBloom (GA) redisbloom.io

RedisTimeSeries redistimeseries.io

RedisJSON (GA) redisjson.io

RedisAI redisai.io

RedisGraph (GA) redisgraph.io

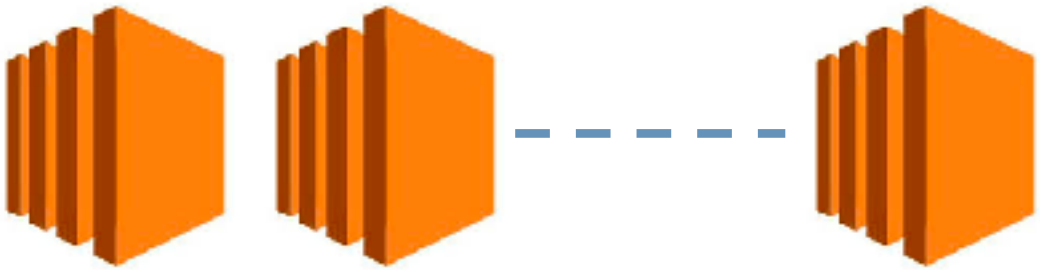
Introducing

Redis Enterprise

redislabs HOME OF REDIS

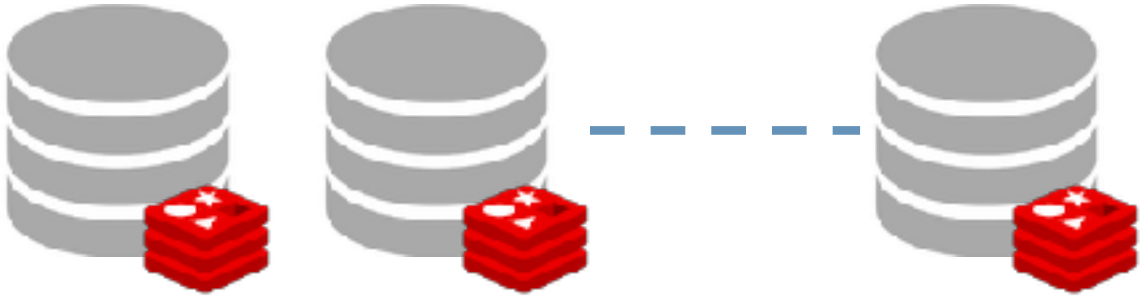
Cloud Providers have different incentives

- Cloud Provider



- Higher margin by
  - Idleness
- Cloud Lock-in

- DBAAS Provider



- Higher margin by better resource utilization
  - Multi-tenancy
  - Reducing RAM
  - CPU utilization

Redis Enterprise : A Unique Primary Database

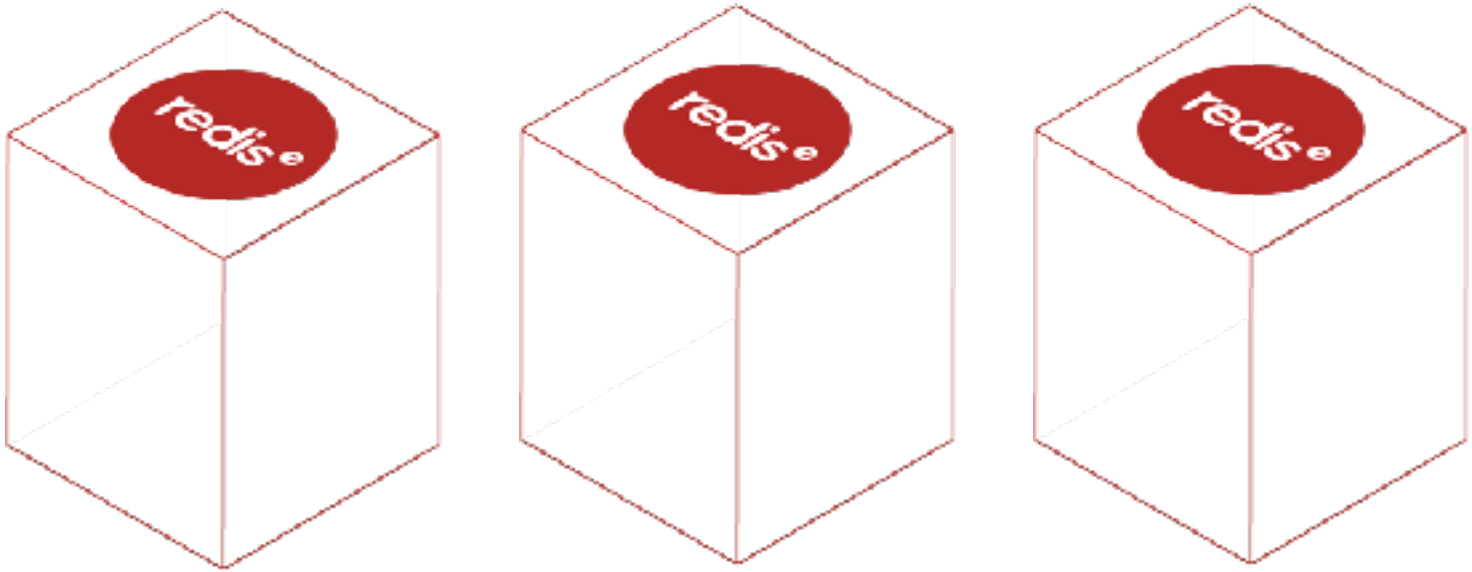
FAST	RELIABLE	FLEXIBLE
HIGHEST PERFORMANCE, LINEAR SCALING		BUILT-IN HIGH PERFORMANCE SEARCH
HIGH AVAILABILITY WITH INSTANT FAILOVER		MULTI-MODEL
DURABILITY AT MEMORY SPEEDS		FLEXIBLE DEPLOYMENT OPTIONS (CLOUD, ON-PREM, HYBRID)
ACTIVE-ACTIVE GEO DISTRIBUTION (CRDT-BASED)		INTELLIGENT TIERED DATA ACCESS (RAM & FLASH MEMORY)

redis

redislabs
HOME OF REDIS

Redis Enterprise Cluster

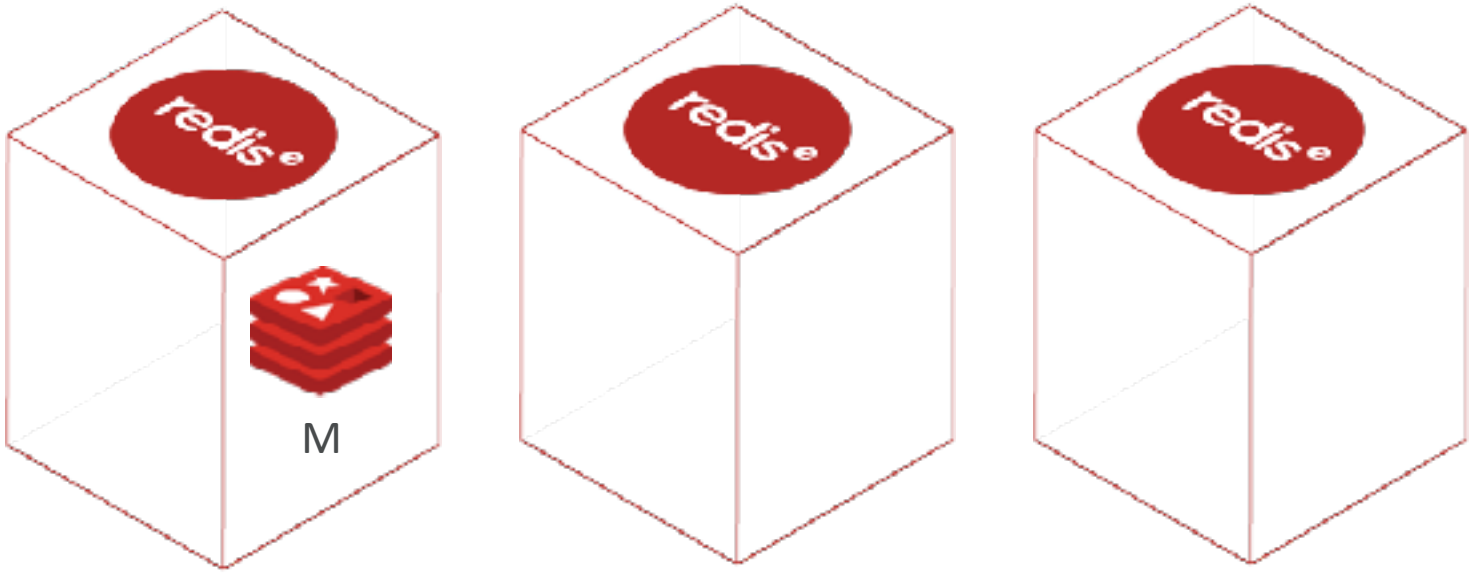
Uneven number of symmetric nodes



Node 1 **Node 2** **Node N (odd number)**

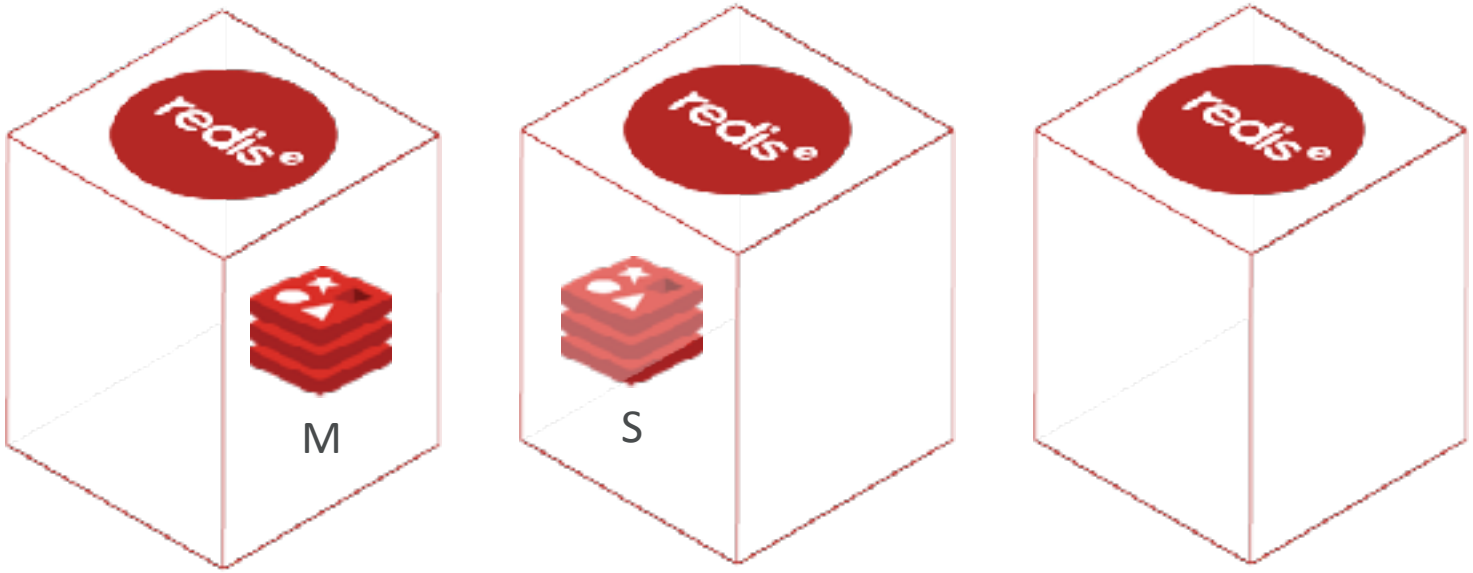
Redis Enterprise Cluster

Single master database



Redis Enterprise Cluster

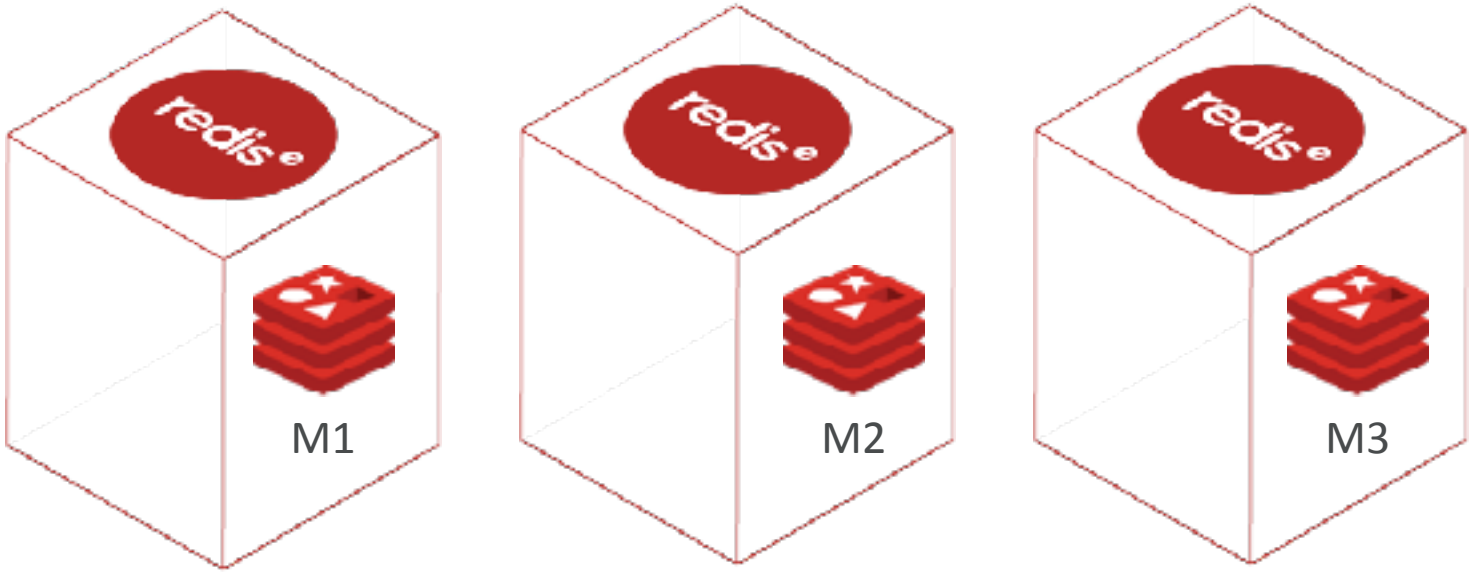
An HA database



Node 1 **Node 2** **Node N (odd number)**

Redis Enterprise Cluster

A Clustered Database



Node 1 **Node 2** **Node N (odd number)**

How do keys get assigned to partitions?

$$hash\_slot = CRC16(key) \bmod 16384$$

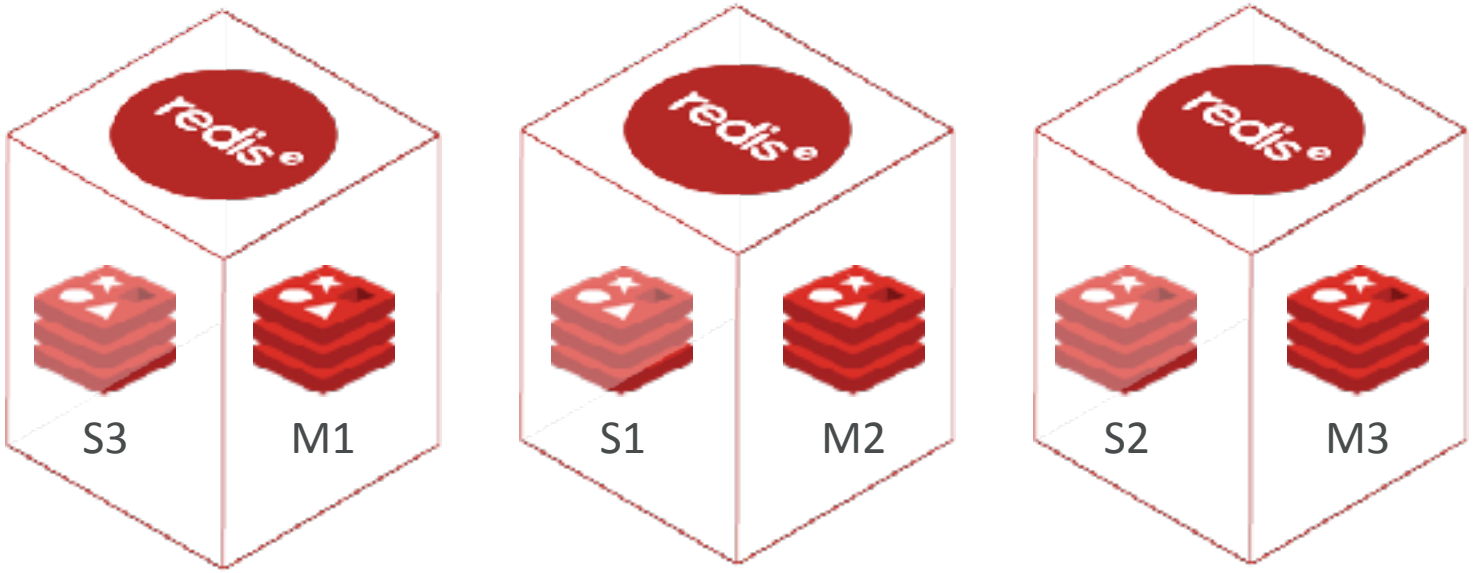
How do keys get assigned to partitions?

$$hash\_slot = CRC16(key) \bmod 16384$$

```
def HASH_SLOT(key)
    s = key.index "{"
    if s
        e = key.index "}",s+1
        if e && e != s+1
            key = key[s+1..e-1]
        end
    end
    crc16(key) % 16384
end
```

Redis Enterprise Cluster

A Highly Available Clustered Database



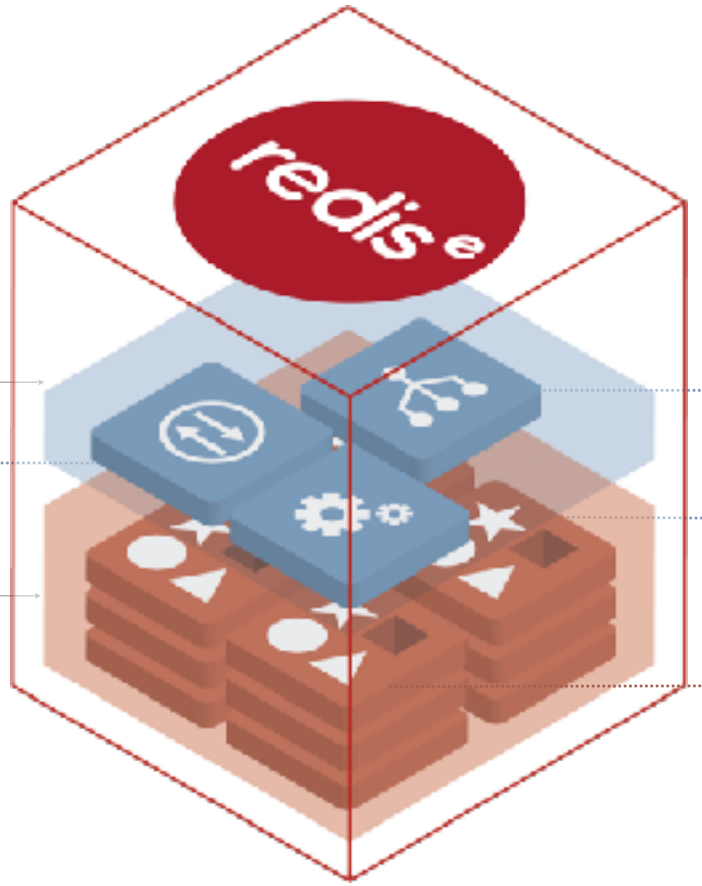
Node 1 **Node 2** **Node N (odd number)**

Redis Enterprise Node

Enterprise Layer

Zero latency proxy

Open Source Layer



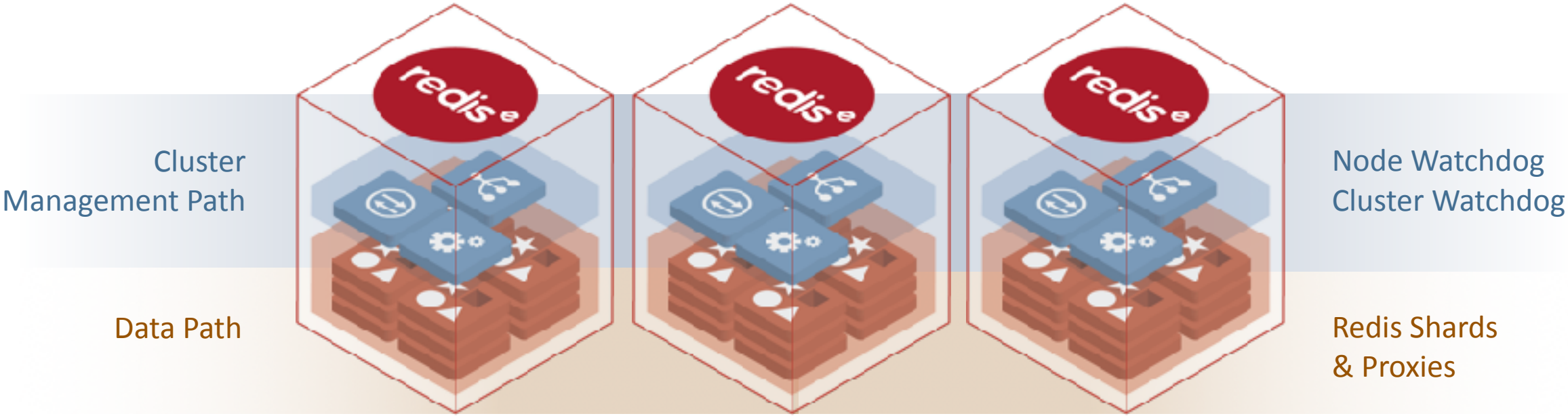
Cluster Manager

REST API

Redis Shards

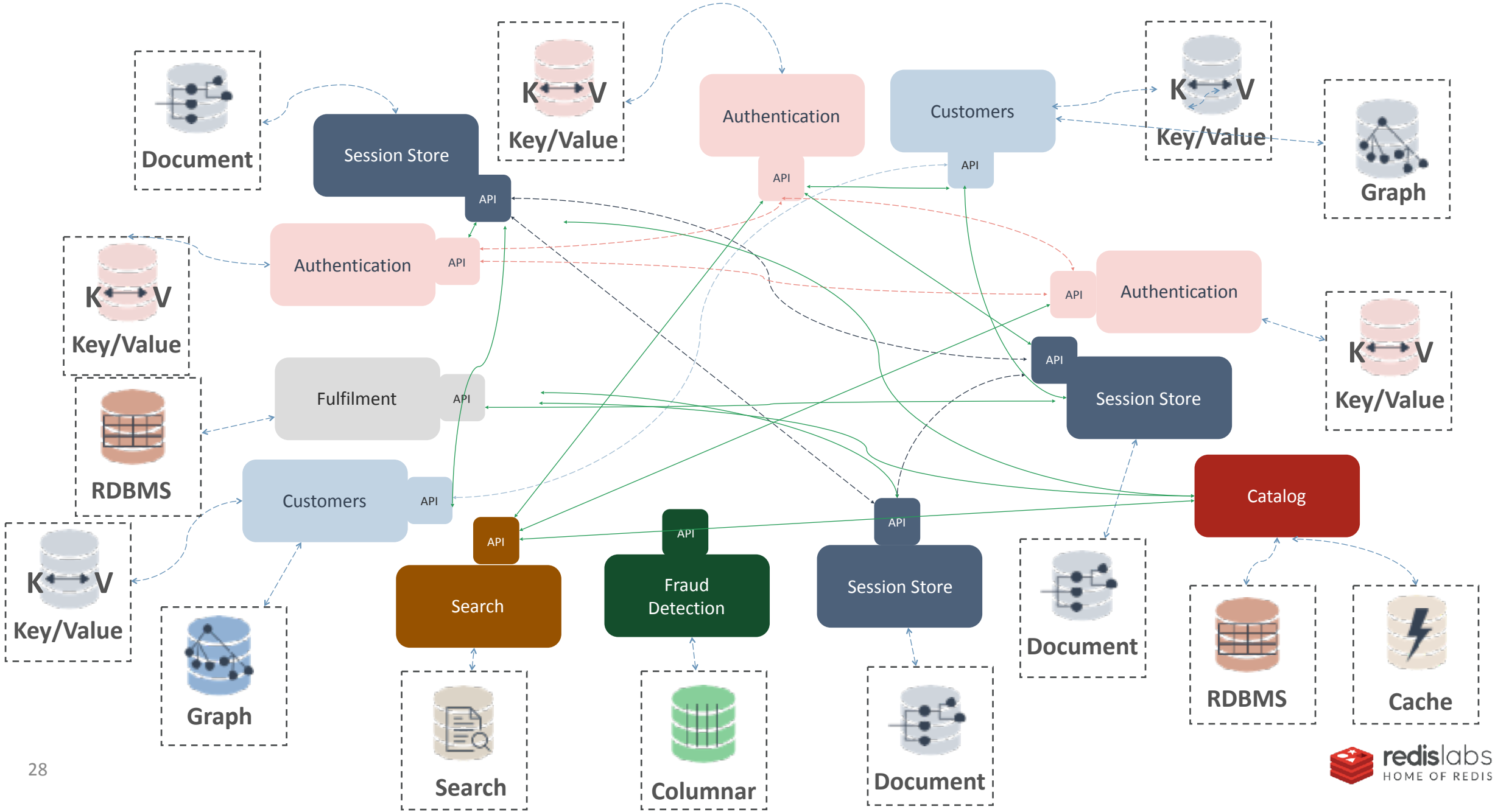
Redis Enterprise: Shared Nothing Symmetric Architecture

Data-Path and Control/Management Path Separation



as a datagrid

Microservices Architecture and Polyglot Persistence



redislabs HOME OF REDIS

The Cost of Polyglot Persistence

Increased application complexity → Costly communication

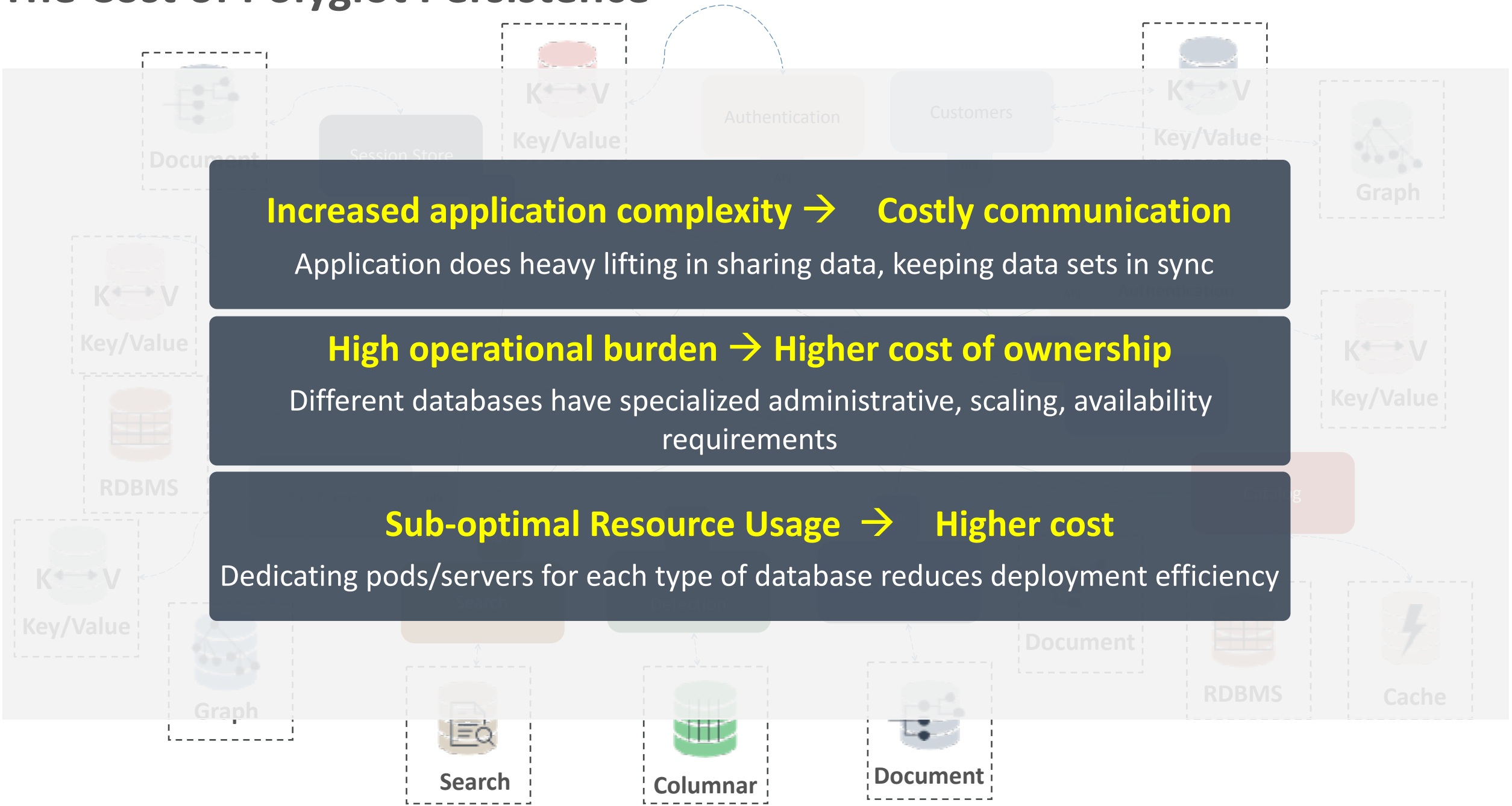
Application does heavy lifting in sharing data, keeping data sets in sync

High operational burden → Higher cost of ownership

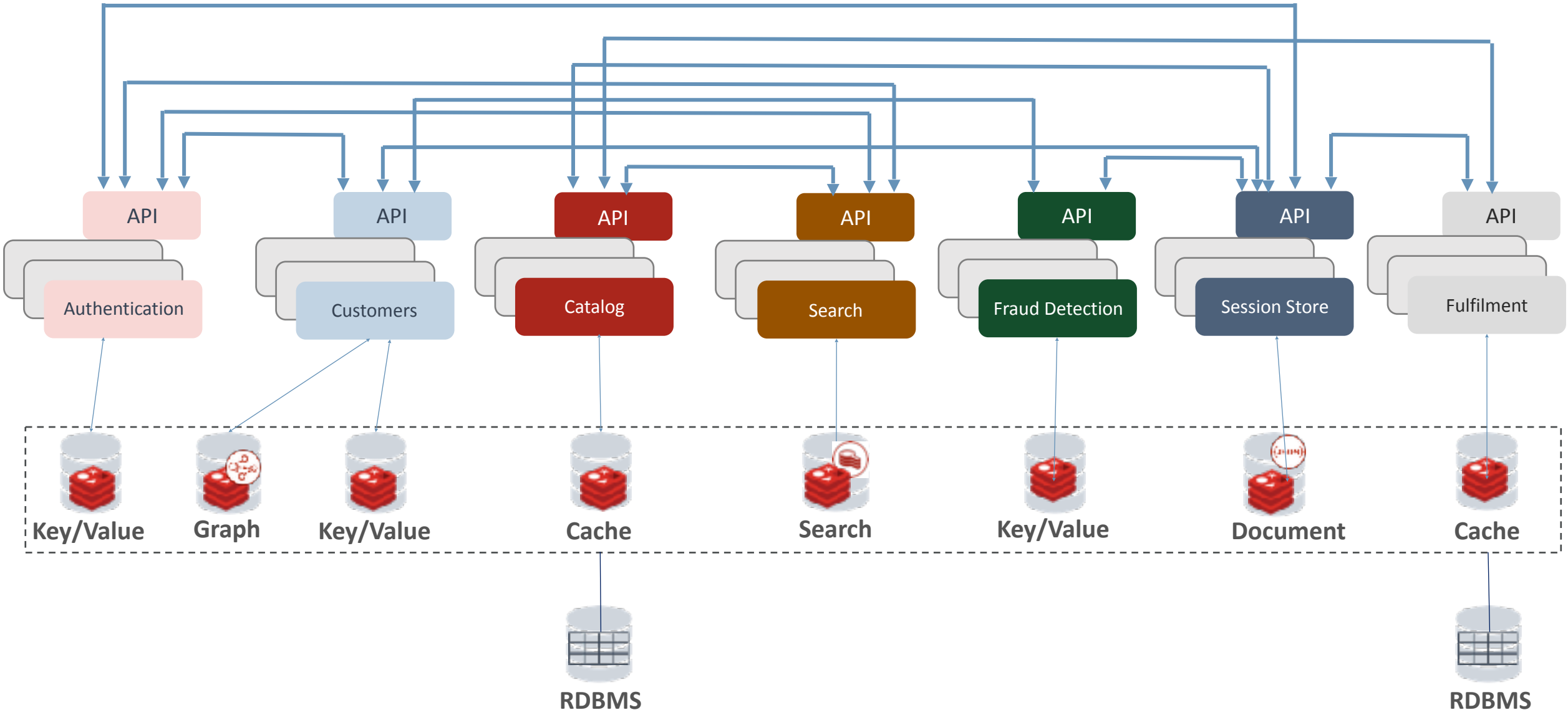
Different databases have specialized administrative, scaling, availability requirements

Sub-optimal Resource Usage → Higher cost

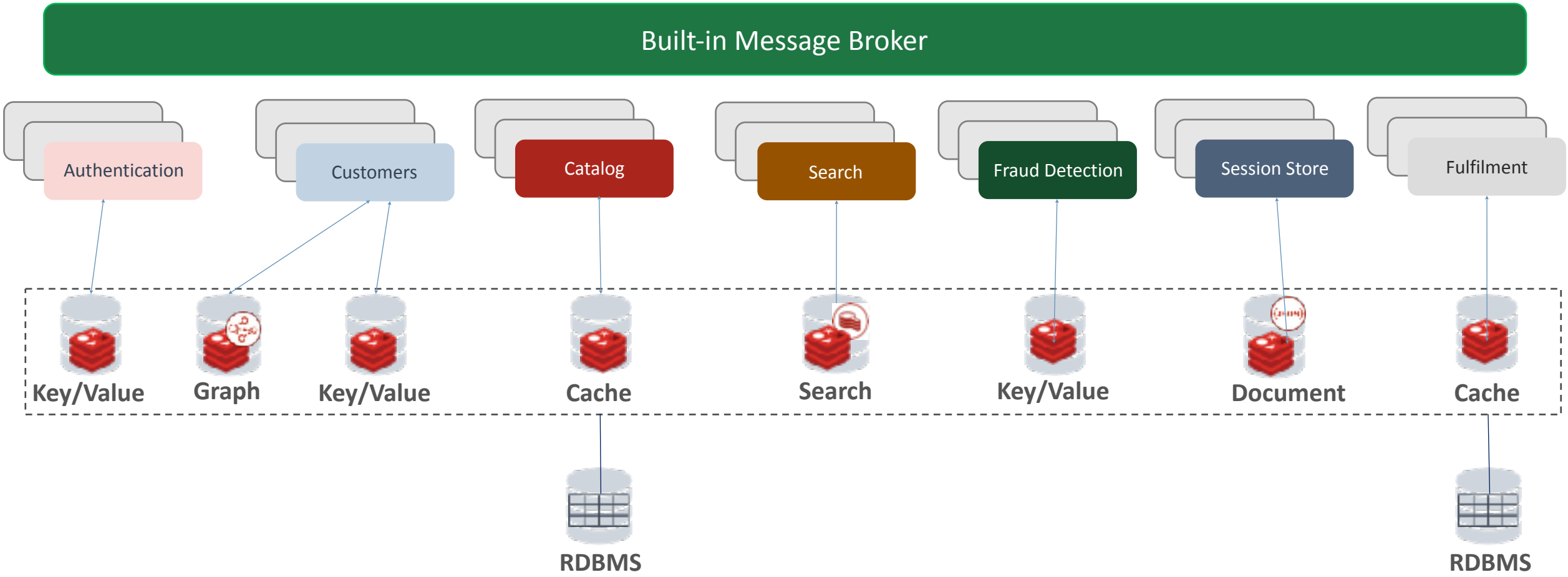
Dedicating pods/servers for each type of database reduces deployment efficiency



Redis Enterprise: A Multi-model Database for Microservices



Built-in Pub-Sub / Streams for event synch across data stores



What are we missing?

- How to consume messages in this *“built in message borker”*
- Given a sharded database, how can I run analytical queries?

- Multi Model database
  - Single copy in core datatypes
  - Inter module communication
  - Component X doing translations between modules.

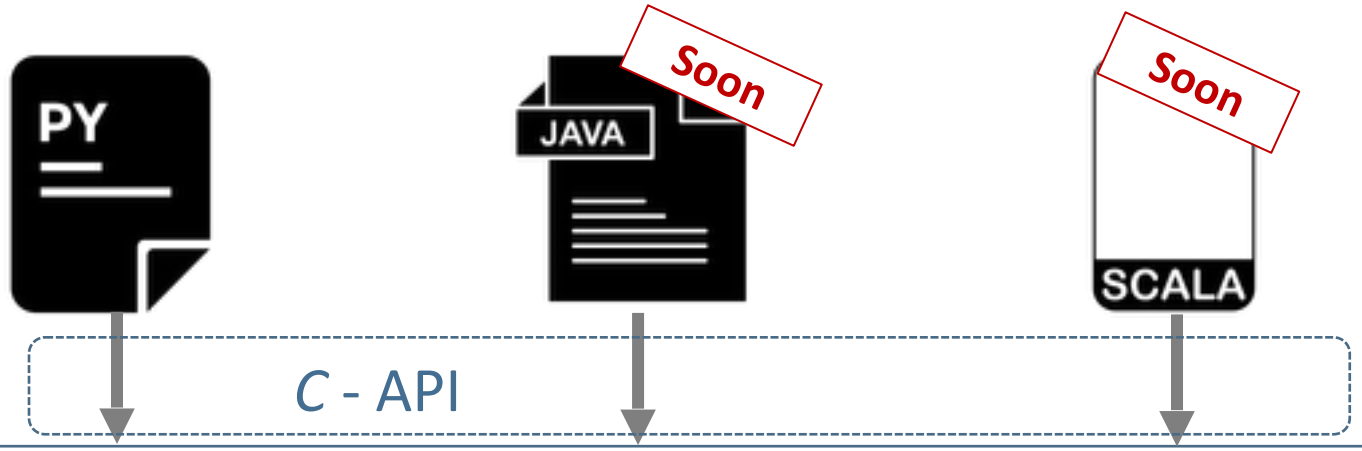
Introducing

RedisGears

What is RedisGears?

RedisGears is a **Serverless** engine for **multi-model and cluster operations** in Redis, supporting both **event driven** as well as **batch operations**

High Performance Architecture



GearsExecuter

GearsCoordinator

MapReducer

Gears infrastructure is written in *C*

Scripting with RedisGears

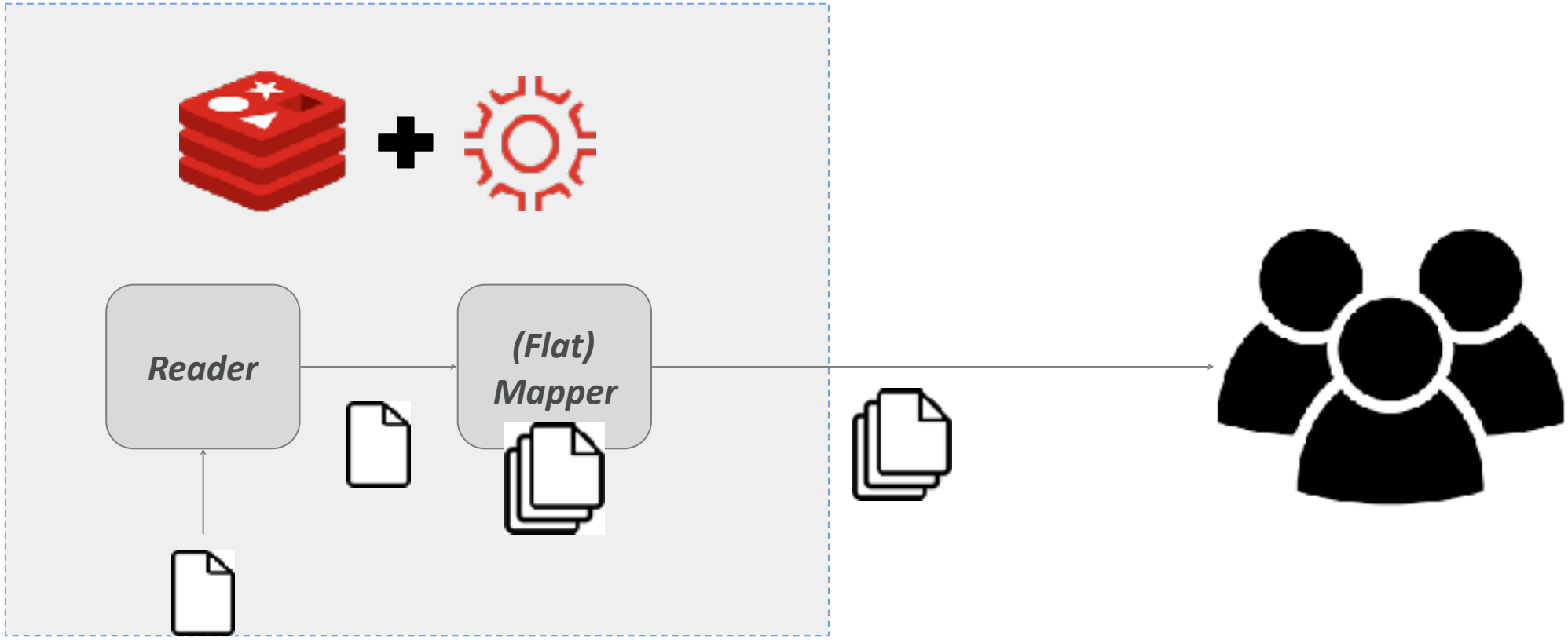
RedisGears allows to define a pipe of operations

- Returning value from one operation is passed to the next operation in the pipe
- Last operation returning the result to the user
- First operation is called ‘reader’ - responsible for providing data
  - Keys reader - read keys from Redis
  - Stream reader - read streams from Redis
  - Python reader - allow to user to write his own readers in python

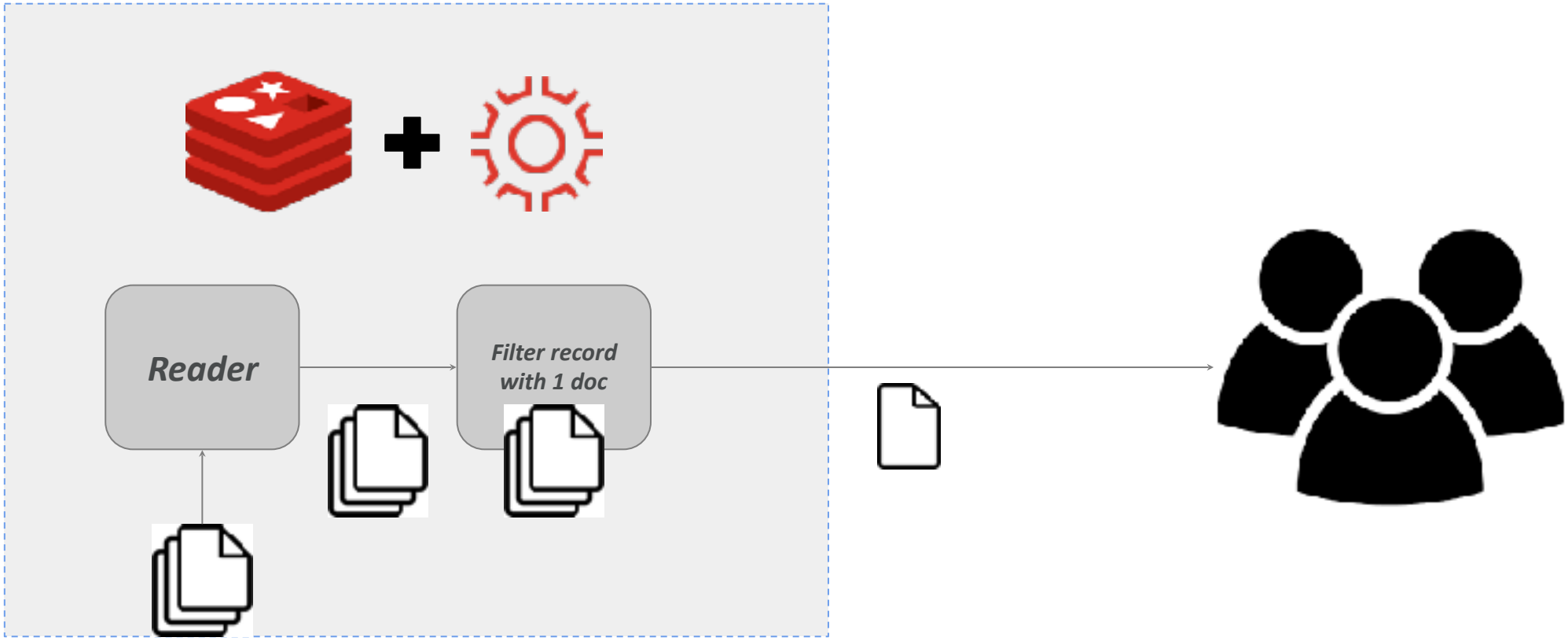
Supported Operations

- Map
- FlatMap
- Filter
- Groupby + Reduce
- Aggregate
- Sort
- Limit
- ForEach
- Distinct

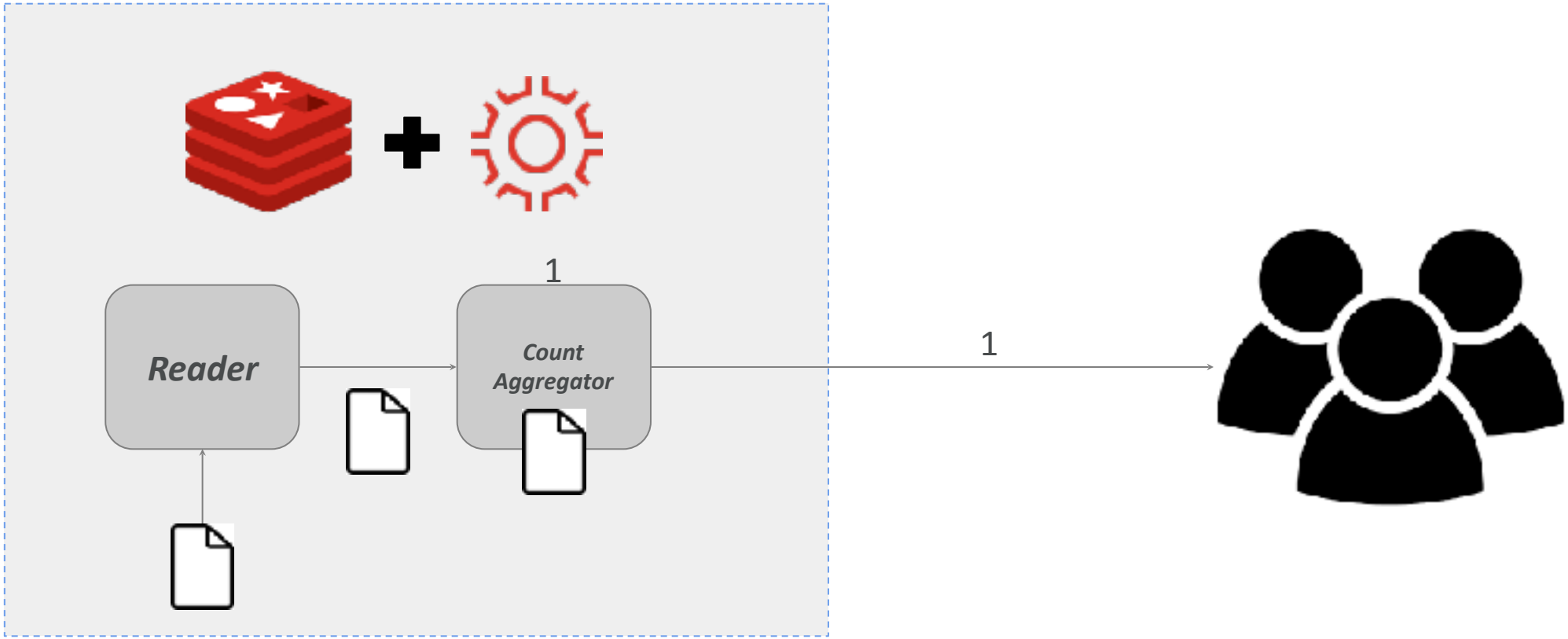
Using RedisGears – (Flat)Mapping



Using RedisGears - Filtering



Using RedisGears - Aggregate

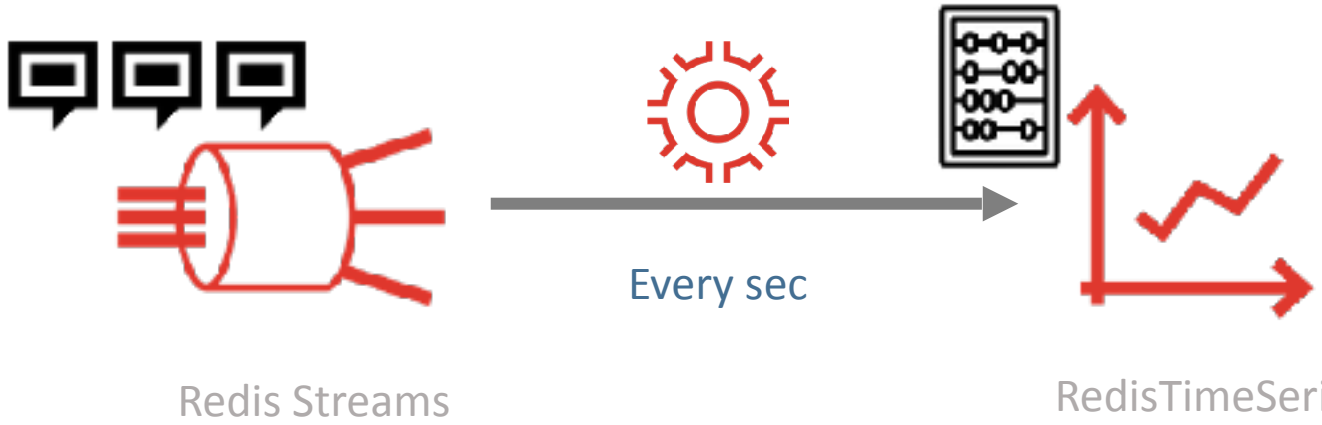


Demo

Use Case #1 – Stream Processing

Gears has a streaming API to allow to trigger gears execution on events.

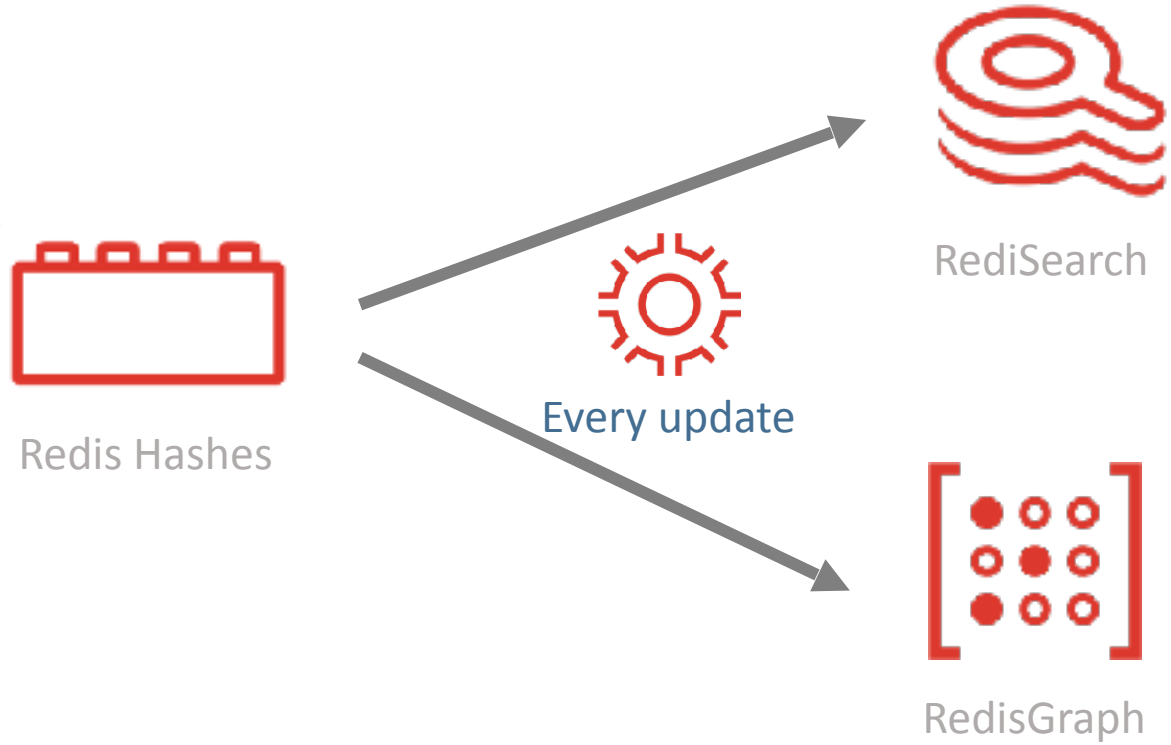
- Redis Stream events - Trigger an execution whenever a new data enters a steam
- Redis Keys events - Trigger an operation whenever a key is touched



Use Case #2 – a MultiModel Engine

Because of RedisGears' flexibility (it's actually running python) you can achieve internal module integration with it:

- Read from hashes and index in RediSearch/RedisGraph
- Read RedisJSON data and pass to RedisTimeSeries
- …



Recipe #1 – even triggering

Build a gear that creates maintains a set of all keys within redis

```
# create the builder
builder = GearsBuilder()
# filter events on key:'all_keys'
builder.filter(lambda x: x['key'] != 'all_keys')
# add the keys to 'all_keys' set
builder.map(lambda x: execute('sadd', 'all_keys', x['key']))
# register the execution on key space notification
builder.register()
```

Recipe #2 – map reducing

Build a gear that counts how often a genre is used within a set of movies

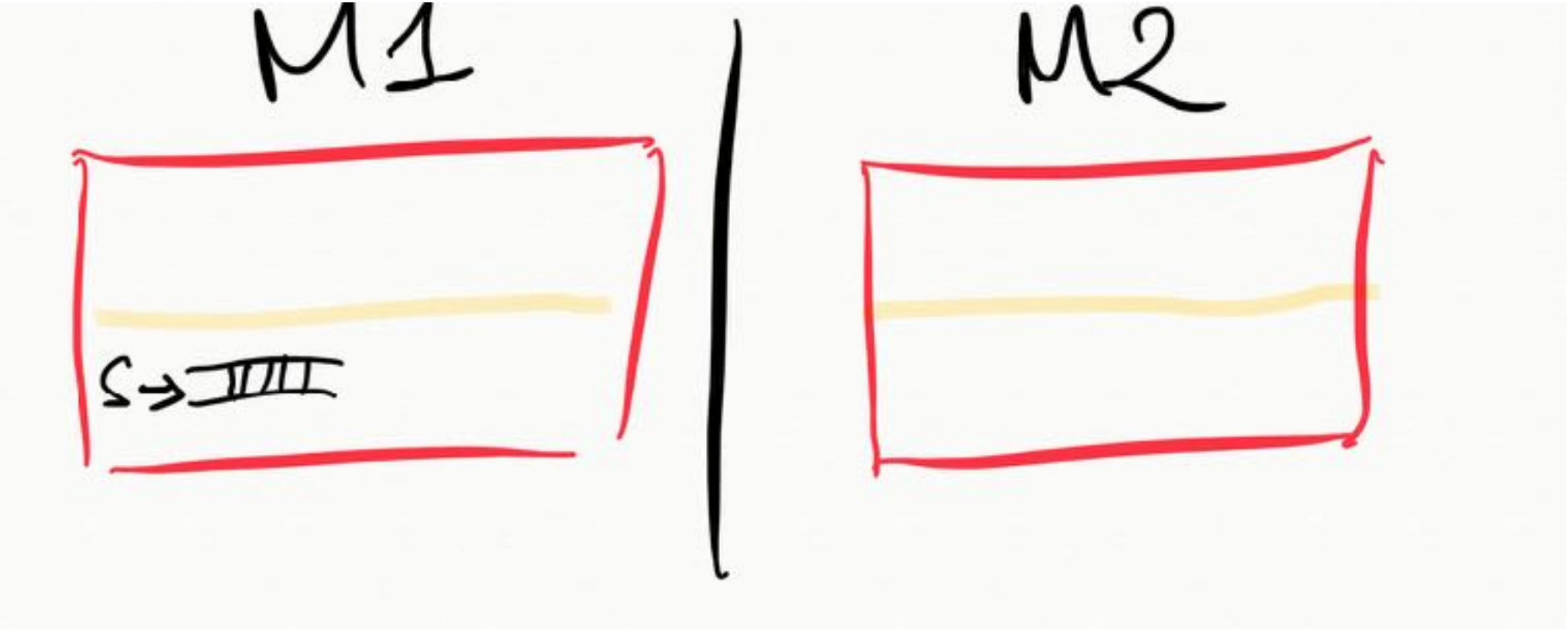
```
# create the pipe builder. KeysOnlyReader is a performance improvement only piping the keys.
builder = GearsBuilder('KeysOnlyReader')
# get from each hash the genres field
builder.map(lambda x: execute('hget', x, 'genres'))
# filter those who do not have genres
builder.filter(lambda x: x is not None)
# split genres by comma
builder.flatmap(lambda x: x.split(','))
# count for each genre the number of times it appears
builder.countby()
# start the execution
builder.run('movie:*')
```

Recipe #3 – stream processing

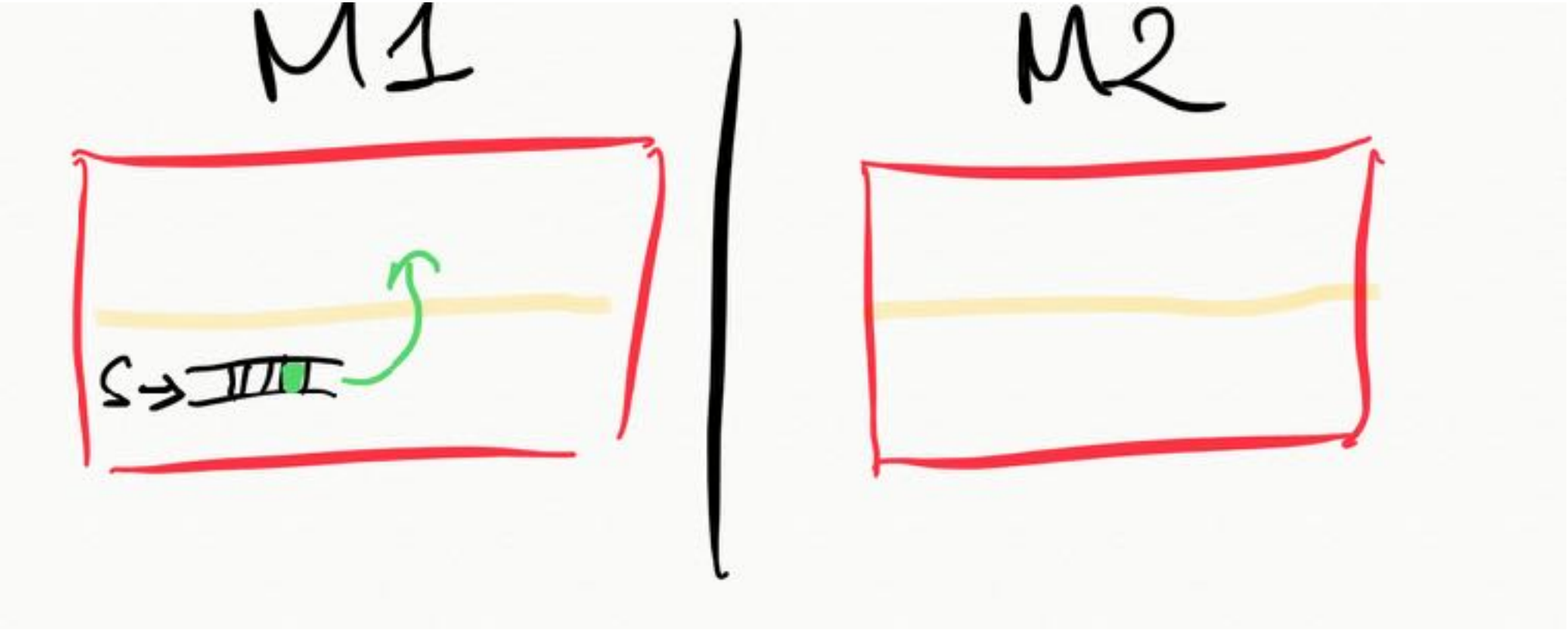
Build a gear that consumes a stream and updates keys accordingly

```
# create the builder with a StreamReader
builder = GearsBuilder('StreamReader')
# extract each field value pair from the message and increase the pipe granularity
builder.flatmap(lambda x: [(a[0], a[1]) for a in x.items()])
# filter out the streamId itself
builder.filter(lambda x: x[0] != 'streamId')
# make sure the gears data lives in the correct shard
builder.repartition(lambda x: x[0])
# apply each field value pair to a key
builder.foreach(lambda x: execute('set', x[0], x[1]))
# register on new messages on the stream 'inputStream'
builder.register('inputStream')
```

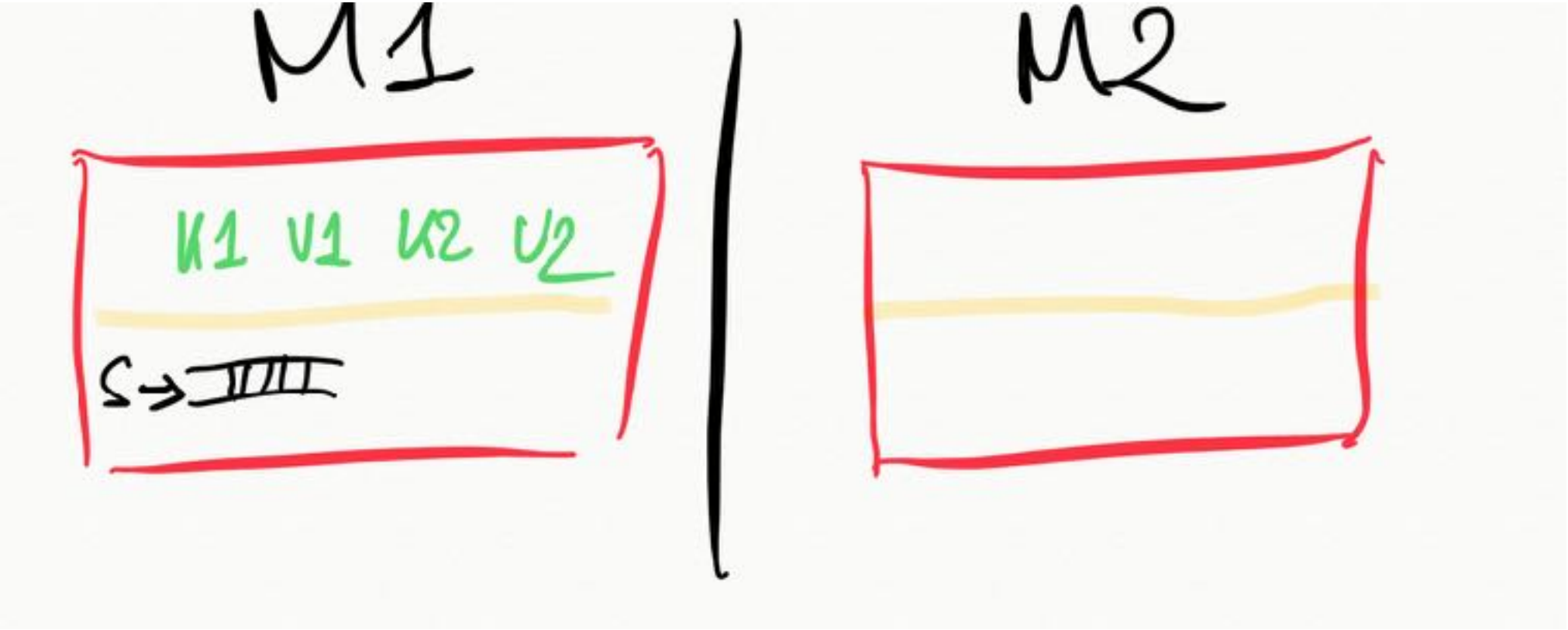
Example Trigger Explained



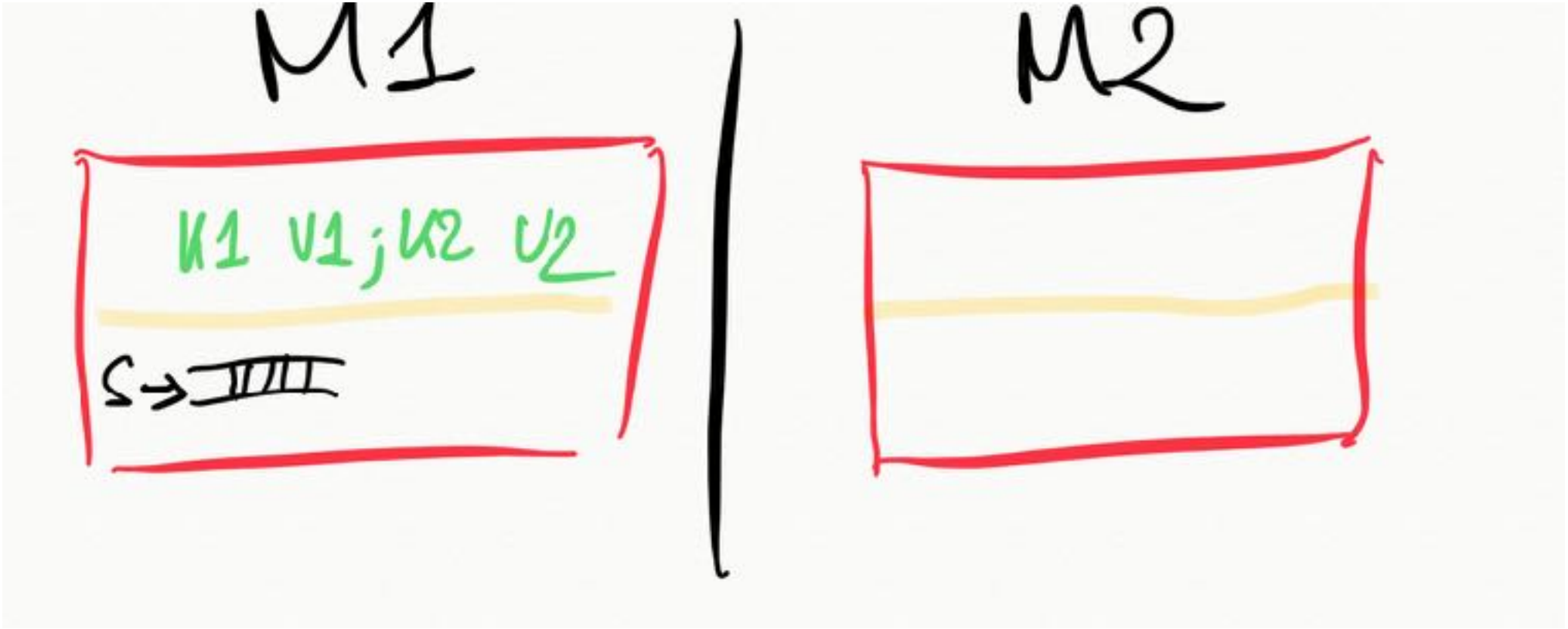
Example Trigger Explained



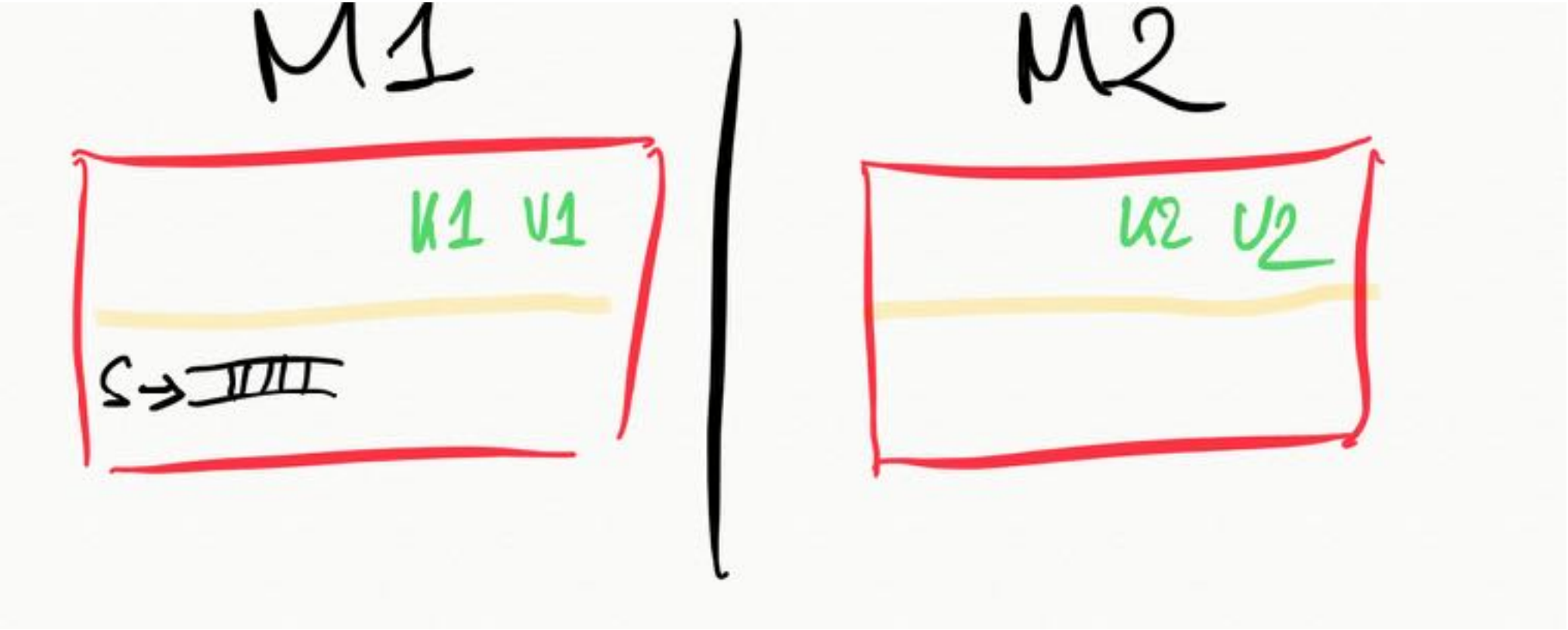
Example Trigger Explained



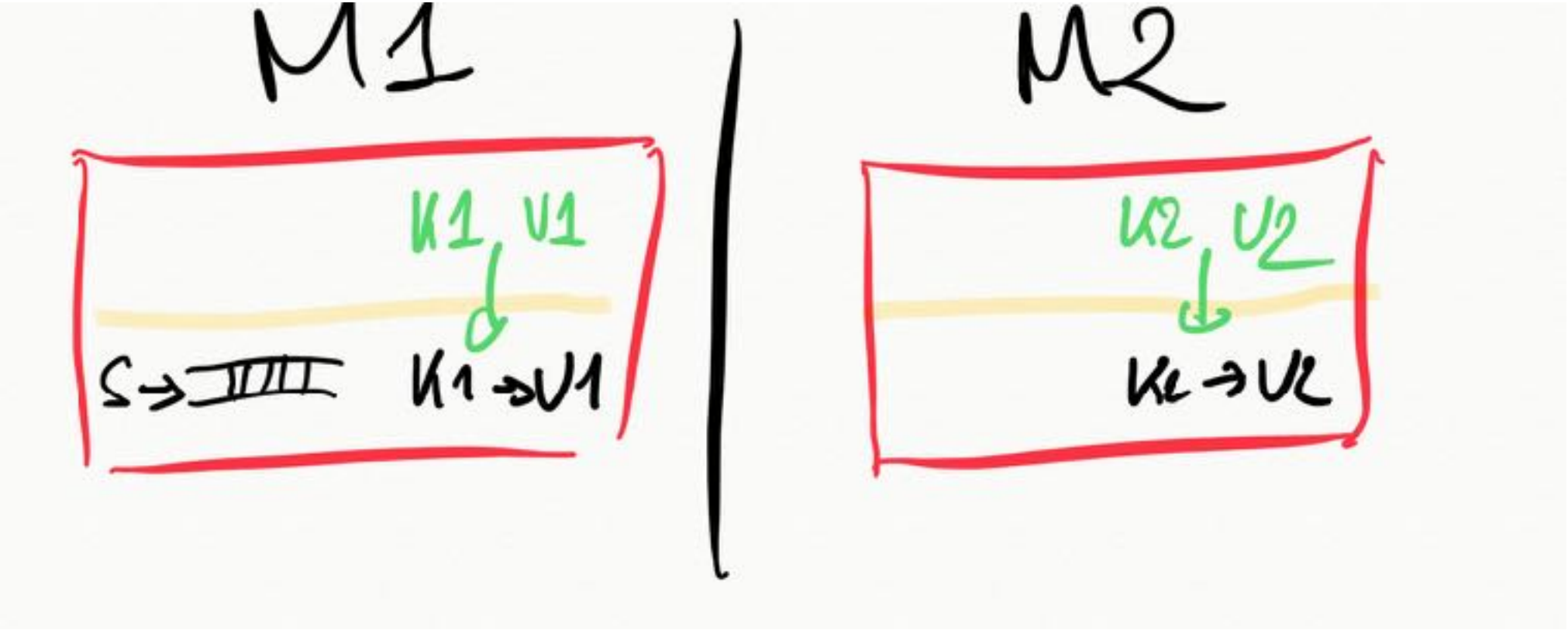
Example Trigger Explained - Flatmap



Example Trigger Explained - Repartition

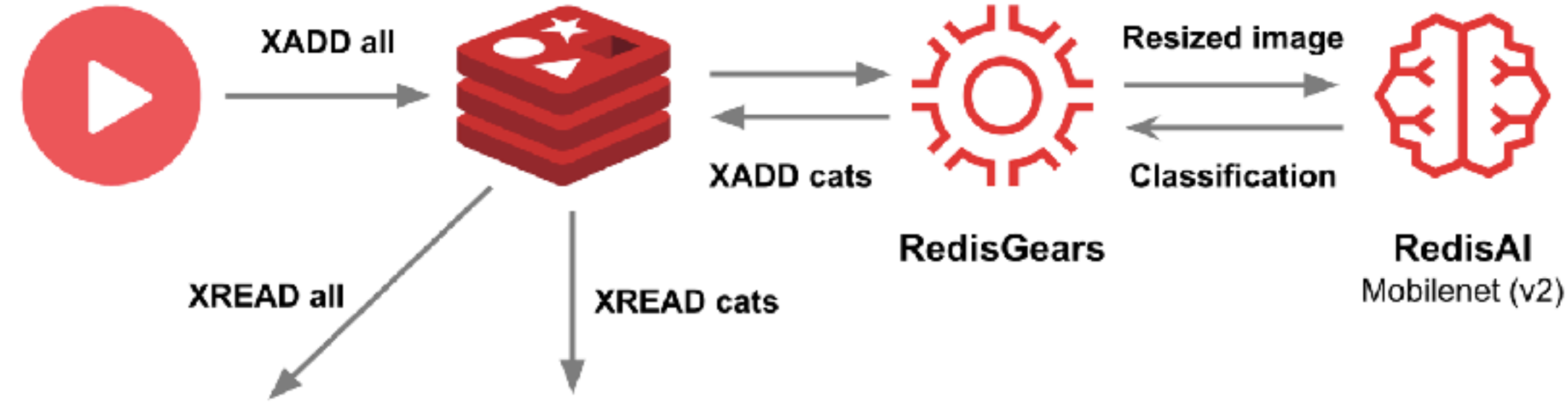


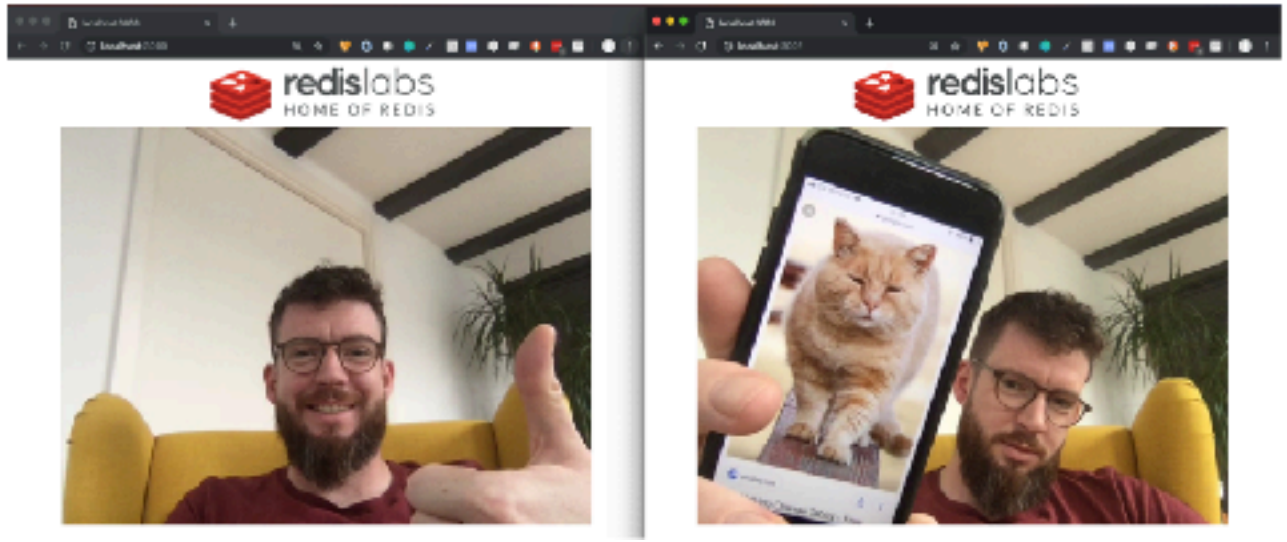
Example Trigger Explained - executeCommand



Demo

Demo Setup


XADD all
XADD cats
Resized image
Classification
RedisGears
RedisAI
Mobilenet (v2)
XREAD all
XREAD cats


redislabs
HOME OF REDIS
redislabs
HOME OF REDIS

Challenge?

- https://github.com/RedisGears/AnimalRecognitionDemo
- #redisfoundmycat

Redis Modules

RediSearch (GA) redisearch.io

RedisBloom (GA) redisbloom.io

RedisTimeSeries redistimeseries.io

RedisJSON (GA) redisjson.io

RedisAI redisai.io

RedisGraph (GA) redisgraph.io

RedisGears redisgears.io

Thank you!

pieter@redislabs.com

redislabs
HOME OF REDIS