2110625 - Data Science Architecture

Key-Value Stores

Asst.Prof. Natawut Nupairoj, Ph.D.
Department of Computer Engineering
Chulalongkorn University
natawut.n@chula.ac.th

CHULA ENGINEERING
Foundation toward Innovation

Cassandra ก็เก็บ memory แต่สามารถเขียนลง disk เพื่อ recovery ได้

Dynamo เน้นเก็บใน memory

Key-Value Stores

ทุกตัวมองเป็น key value แต่ต่างที่วิธีการเก็บ เน้น memory เป็นหลัก เก็บลง storage ได้ แต่ primary เป็น memory

- A data store designed for storing, retrieving, and managing associative arrays (aka. dictionary or hash table)

- Main concept is to store data as a collection of key-value pairs in which a key serves as a unique identifier

- Simple and fast, often use in-memory architecture, ability to scale-out

↑ เก็บใน localStorage หรือ key:value store ได้

- Use cases: session store, shopping cart, SQL/API caching, etc.

↳ session ของเว็บ

↳ ทุกการ call API มี overhead เอาผลลัพธ์ที่เคย query เก็บเอาไว้

db มี cache เหมือนกัน แต่เจอปัญหา db ไม่ไหวก็ app sever ทำให้ไม่เกิด ปฏิบัติ

Redis

Key-Value Stores

Redis (Remote Dictionary Server)

เห็น failover กับ scale

- In-memory data structure store with clustering, transactional, time-to-live limiting, and auto-failover capabilities

key มี expire ได้ไหม

- Support wide-range of data structure with lots of related operations for each structure

รองรับหลายโครงสร้าง

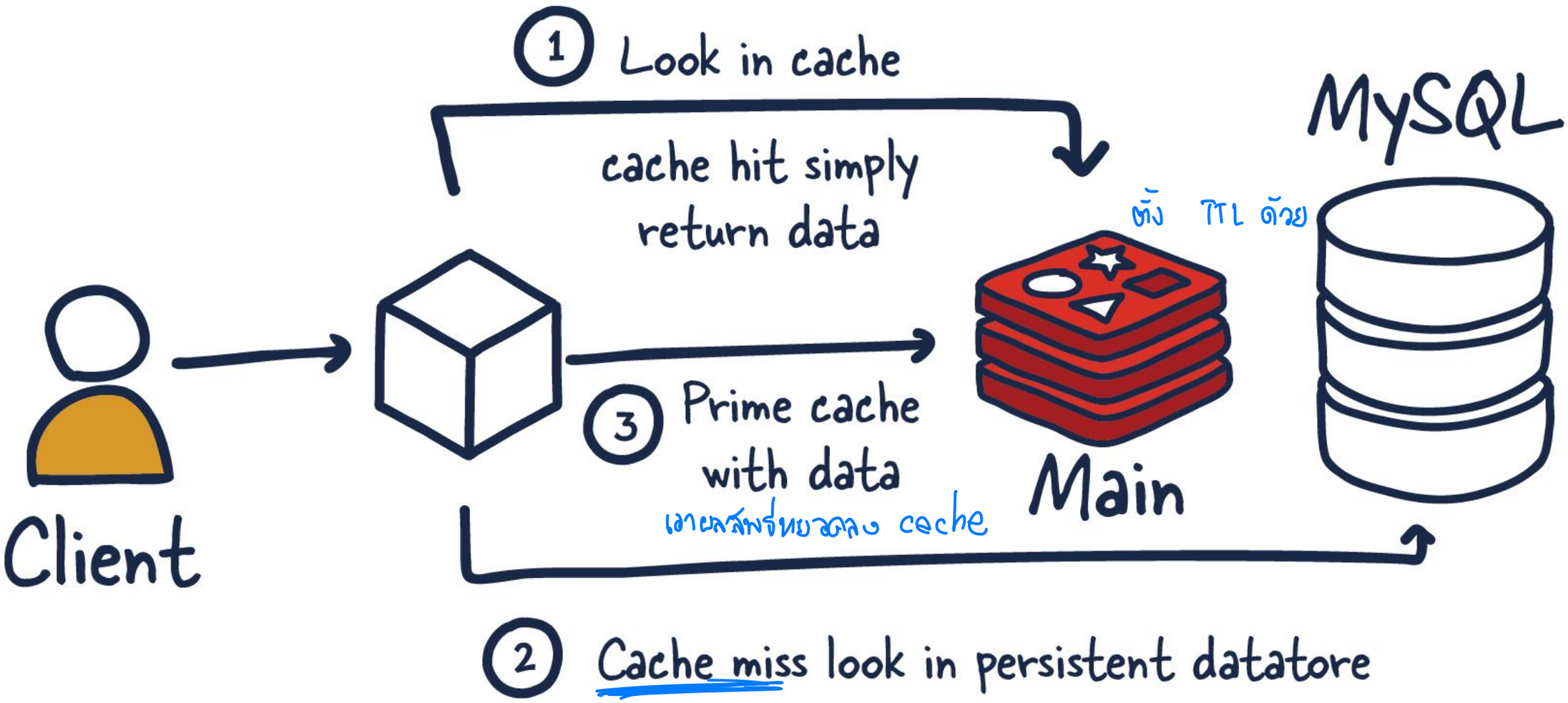
- Being used for database cache, message broker, streaming engine, feature store engine, etc.

- Provide CLI and support many programming languages

- Many useful modules are available to extend the functionality of Redis core e.g. RedisJSON, RedisSearch, RedisTimeSeries, Redis OM, etc.

feature store เก็บ dataset ไว้ train model ได้

How is redis traditionally used



Source: https://architecturenotes.co/redis/

ถ้า memory เต็มจะทำงานต่อไม่ได้

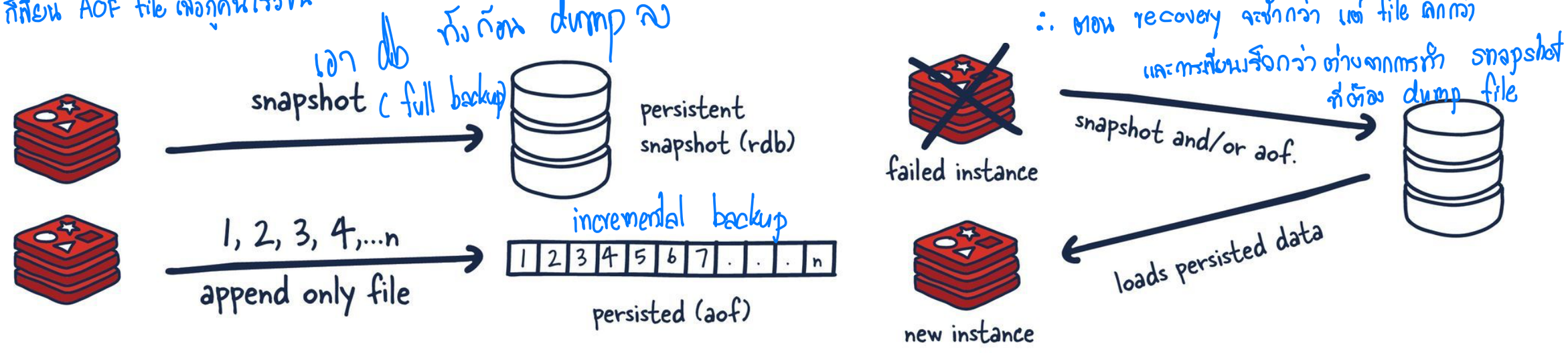
Redis Persistence

- Redis supports many level of persistence: no persistence, RDB (point-in-time snapshot), AOF (log every write), RDB+AOF

ทำ snapshot ทุก 1 ชม. แล้วระหว่าง ชม. ก็เขียน AOF file เผื่อกู้คืนเร็วขึ้น

append only file → เวลาที่มีการ write redis จะเขียนต่อท้ายใน AOF file เผื่อจับทึกความเปลี่ยนแปลง ที่เกิดขึ้น

∴ ตอน recovery จะช้ากว่า แต่ file เล็กกว่า



snapshot 1 ชม. รหว่นมีเพิ่ม AOF

redis มีการ compress ตัวเอง

Running Redis

- The simplest way to run a redis instance is to use docker

```
docker pull redis
docker run -d --rm --name redis -p 6379:6379 redis
```

- This will start redis in your docker at port 6379 and map the port to your localhost

- You can also use docker-compose.yml and other example files in redis folder in datasci_architecture repo

Working with Redis

- Redis CLI
  - Standard client program to connect to any redis server
  - Come with any redis installation (see: https://redis.io/docs/getting-started/)

```
redis-cli

redis-cli -h 34.143.227.66
```

  - You can type in redis command in the CLI input

Working with Redis

- Redis-Py
  - Standard python package for redis client

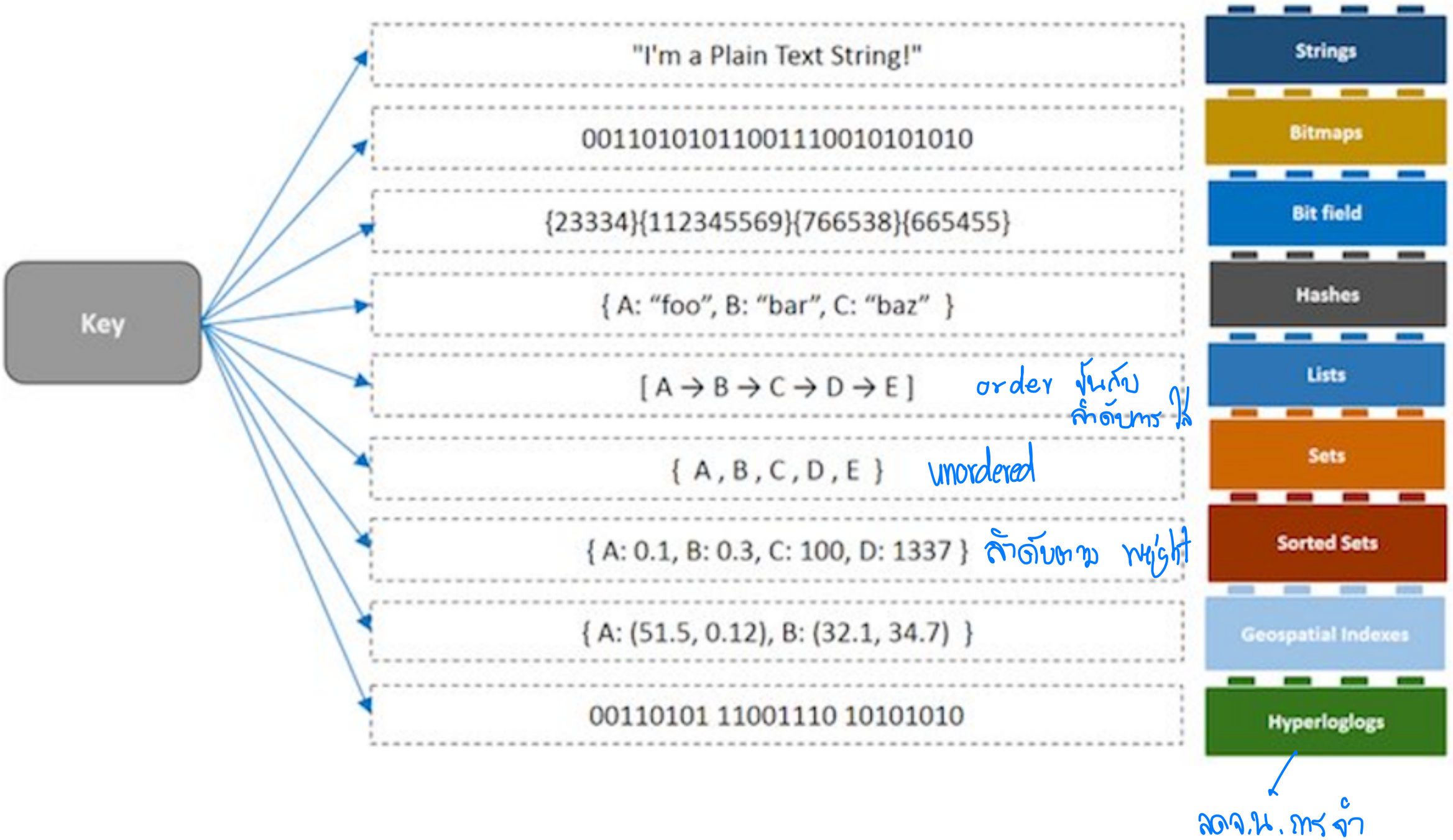
```
pip install redis
```

- Example

```
import redis

r = redis.Redis(host='hostname', port=port)
```

Redis Data Types

รองรับหลายโครงสร้าง value

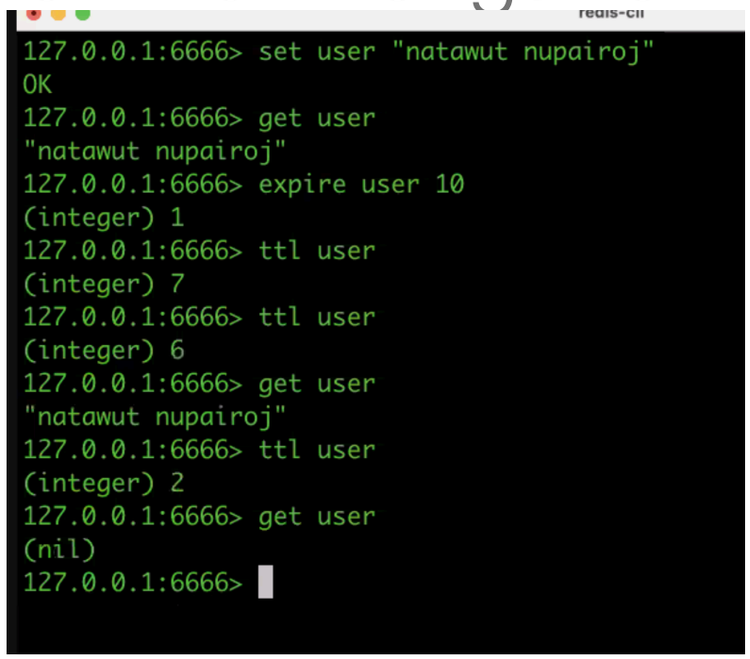


String

- Similar to Python or Java Strings, maximum length of 512MB

```
SET “user” “Natawut Nupairoj”
GET “user”
DEL “user”
```

key value



- Use cases
  - Server-side object cache e.g. HTML fragments, shopping cart, user profile
  - ~~Queues~~
  - Activity tracking

Other String Commands

APPEND

DECR

DECRBY

GET

GETDEL

GETEX

GETRANGE

GETSET

INCR

INCRBY

INCRBYFLOAT

LCS

MGET

MSET

MSETNX

PSETEX

SET

SETEX

SETNX

SETRANGE

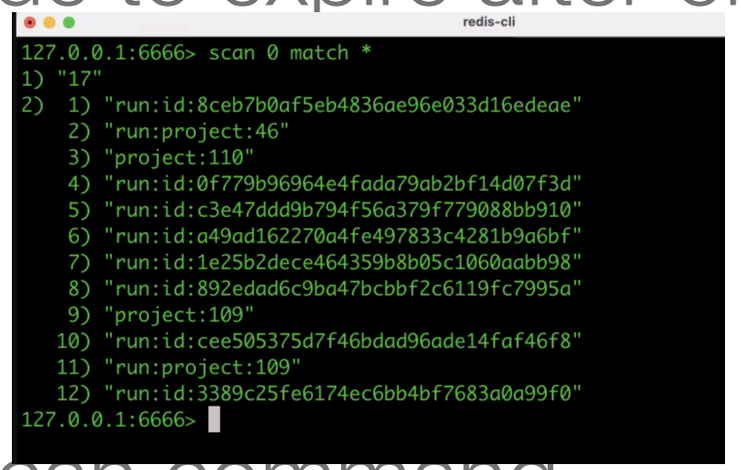
STRLEN

SUBSTR

Useful Commands

- Any item in Redis can be made to expire after or at a certain time

```
EXPIRE user 60.  # in seconds
TTL user
```



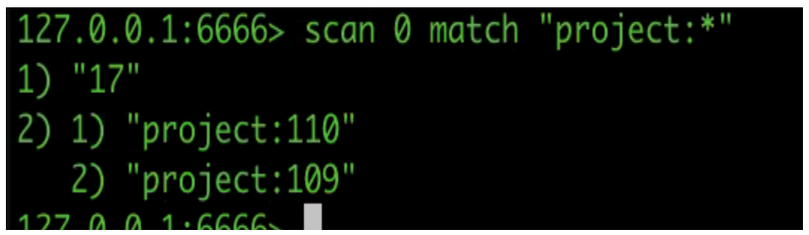
- You can scan all index with scan command

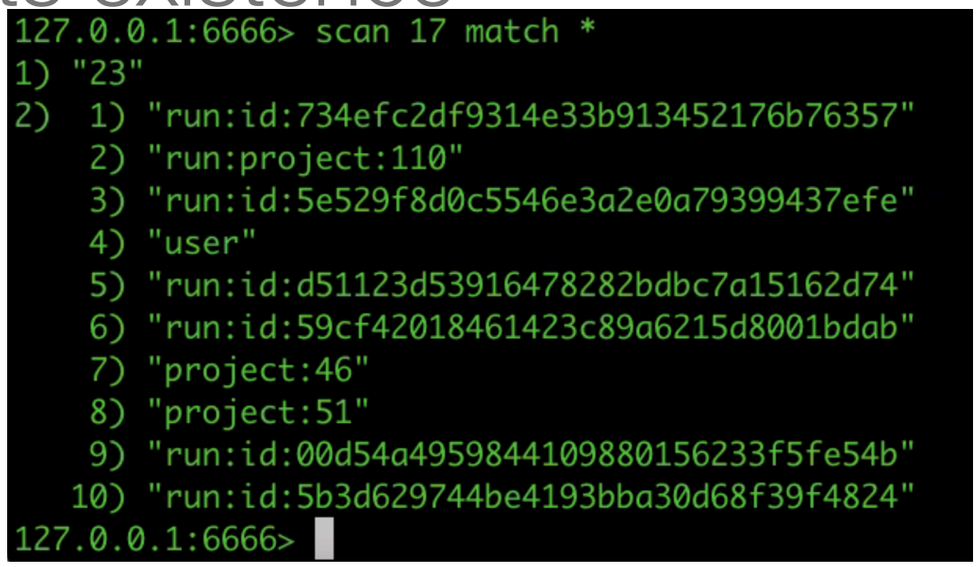
```
SCAN 0
```

scan 0 = scan key ตั้งแต่ต้น

- You can delete item or test its existence

```
DEL mykey
EXISTS mykey
```





```
127.0.0.1:6666> type run:id:8ceb7b0af5eb4836ae96e033d16edeae
hash
```

```
127.0.0.1:6666> hgetall run:id:8ceb7b0af5eb4836ae96e033d16edeae
 1) "id"
 2) "8ceb7b0af5eb4836ae96e033d16edeae"
 3) "p_id"
 4) "51"
 5) "parameters"
 6) "{\"begin\":-1.0,\"end\":-1.0}"
 7) "snapshottype"
 8) "brief"
 9) "status"
10) "completed"
11) "last_updated"
12) "1713695310.318542"
127.0.0.1:6666>
```

```
22) "run:id:59cf42018461423c89a6215d8001bdab"
23) "run:id:892edad6c9ba47bcbbf2c6119fc7995a"
24) "project:109"
25) "run:id:a12a3080aeb1410b961ed28de323cb68"
127.0.0.1:6666> type run:project:46
list
127.0.0.1:6666> lrange run:project:46 0 -1
1) "c3e47ddd9b794f56a379f779088bb910"
2) "a49ad162270a4fe497833c4281b9a6bf"
3) "1e25b2dece464359b8b05c1060aabb98"
127.0.0.1:6666> hgetall run:1e25b2dece464359b8b05c1060aabb98
(empty array)
127.0.0.1:6666> hgetall run:id:1e25b2dece464359b8b05c1060aabb98
 1) "id"
 2) "1e25b2dece464359b8b05c1060aabb98"
 3) "p_id"
 4) "46"
 5) "parameters"
 6) "{\"begin\":1711542601.0,\"end\":1713772955.72234}"
 7) "snapshottype"
 8) "brief"
 9) "status"
10) "completed"
11) "last_updated"
12) "1713772967.819626"
127.0.0.1:6666>
```

List

```
127.0.0.1:6666> sadd team natawut john
(integer) 2
127.0.0.1:6666> scard team
(integer) 2
127.0.0.1:6666> sadd team natawut john
(integer) 0
127.0.0.1:6666> scard team
(integer) 2
127.0.0.1:6666> smembers team
1) "natawut"
2) "john"
127.0.0.1:6666>
```

ใส่ key ผิด redis ทำงานต่อได้ ไม่ error ข้อเสีย redis

List

- List of strings, sorted by insertion order
- Can be used as list, queue, stack

LPUSH → push ทางซ้าย
RPUSH → push ทางขวา (ต่อท้าย)

```
LPUSH mylist abc  # mylist contains “abc”
LPUSH mylist xyz # mylist contains “xyz”, “abc”
RPUSH mylist 123 # mylist contains “xyz”, “abc”, “123”
```

- Use cases
  - Queue
  - Timelines

List Commands

BLMOVE

BLMPOP

BLPOP

BRPOP

BRPOPLPUSH

LINDEX

LINSERT

LLEN

LMOVE

LMPOP

LPOP

LPOS

LPUSH

LPUSHX

LRANGE

LREM

LSET

LTRIM

RPOP

RPOPLPUSH

RPUSH

RPUSHNX

~~Set~~

- Powerful data types for unordered non-duplicated keys
- Support many set operations e.g. intersection, union, etc.

```
SADD user_set natawut
SCARD user_set
SMEMBERS user_set
```

- Use cases
  - Set of user profiles
  - Set of inappropriate words for inappropriate content filtering

Set Commands

SADD

SCARD

SDIFF

SDIFFSTORE

SINTER

SINTERCARD

SINTERSTORE

SISMEMBER

SMEMBERS

SMISMEMBER

SMOVE

SPOP

SRANDMEMBER

SREM

SSCAN

SUNION

SUNIONSTORE

Sorted Set

เรียงตาม weight ไม่เรียงตามลำดับการใส่

- Set of sorted items based on the score associated to each member

```
ZADD my_sortedset 5 data1
ZADD my_sortedset 1 data2 10 data3
ZRANGEBYSCORE my_sortedset 5. +inf WITHSCORES
```

- Use
  - Leader scoreboard
  - Priority queue

Sorted Set Commands

BZMPOP

BZPOPMAX

BZPOPMIN

ZADD

ZCARD

ZCOUNT

ZDIFF

ZDIFFSTORE

ZINCRBY

ZINTER

ZINTERCARD

ZINTERSTORE

ZLEXCOUNT

ZMPOP

ZMSCORE

ZPOPMAX

ZPOPMIN

ZRANDMEMBER

ZRANGE

ZRANGEBYLEX

ZRANGEBYSCORE

Hash

set ของ data ที่ value เป็น string เท่านั้น

- A container of unique fields and their values

```
HMSET profile:12345 user nnp id 12345  name “Natawut Nupairoj”
balance 10
```

key value

```
HGETALL profile:12345

HINCRBYprofile:12345 balance 5
```

- Use
  - User profile information
  - Post. Information

Hash Commands

HDEL

HEXISTS

HGET

HGETALL

HINCRBY

HINCRBYFLOAT

HKEYS

HLEN

HMGET

HMSET

HRANDFIELD

HSCAN

HSET

HSETNX

HSTRLEN

HVALS

Redis ทำ Aggregation ไม่ได้

Example: Recipes in Redis

- Create data model for recipe

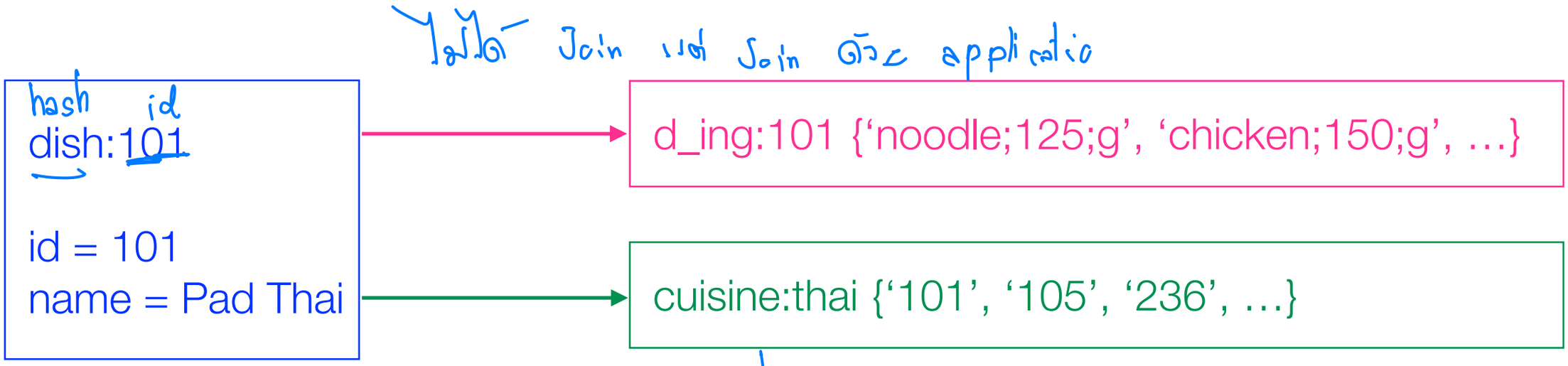
take over ดีๆครับสิ่งที่ทำงาน

- Data items
  - Main data
    - Dish - hash - key = dish:dish_id, value = dish_id, name, category, cuisine
    - Dish Ingredients - set - key = d_ing:dish_id, value = string(name; size; unit)
  - For query
    - Cuisine - set - key = cuisine:cuisine_value, value = string(dish_id)
- Relationship
  - 1 dish can have multiple ingredients
  - Cuisine, as a helper data item, points back to dishes belong in the cuisine
- This is only one example of model design

Example: Recipes in Redis

การตั้งชื่อ key สำคัญมาก เพราะตัวนี้ value ไม่ใช่

ไม่ได้ Join แต่ Join ด้วย application

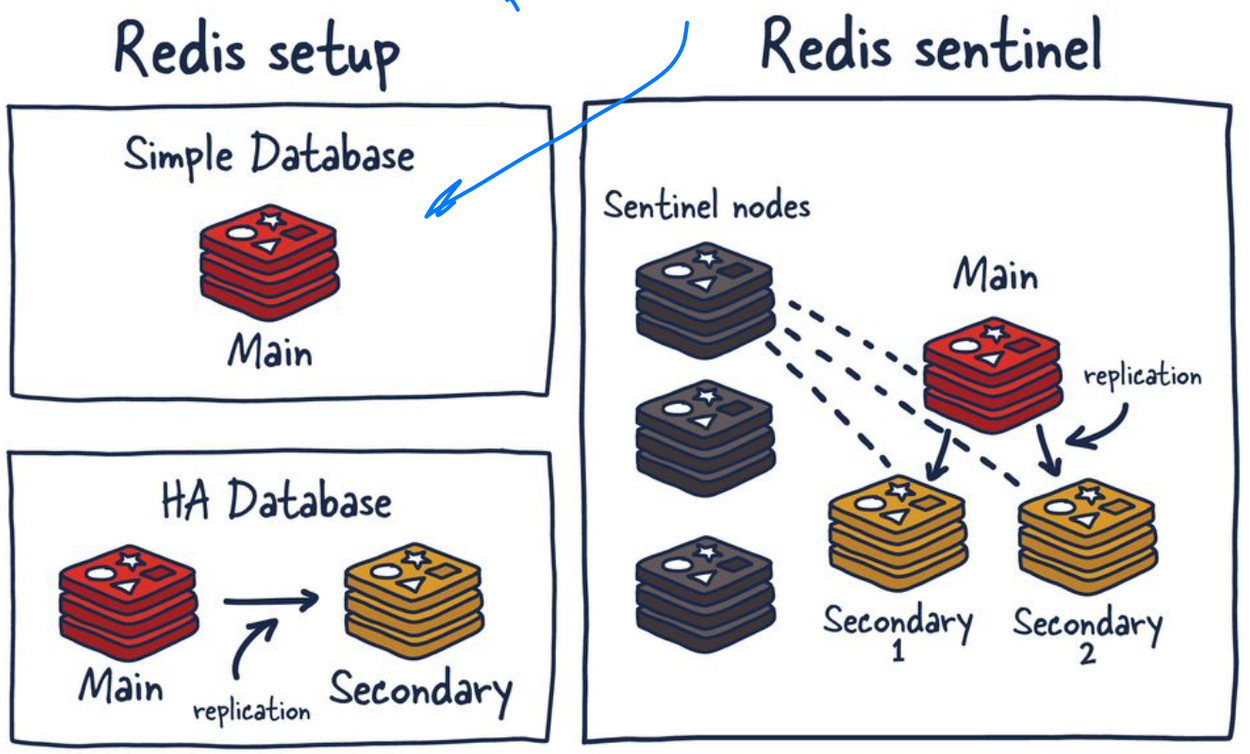


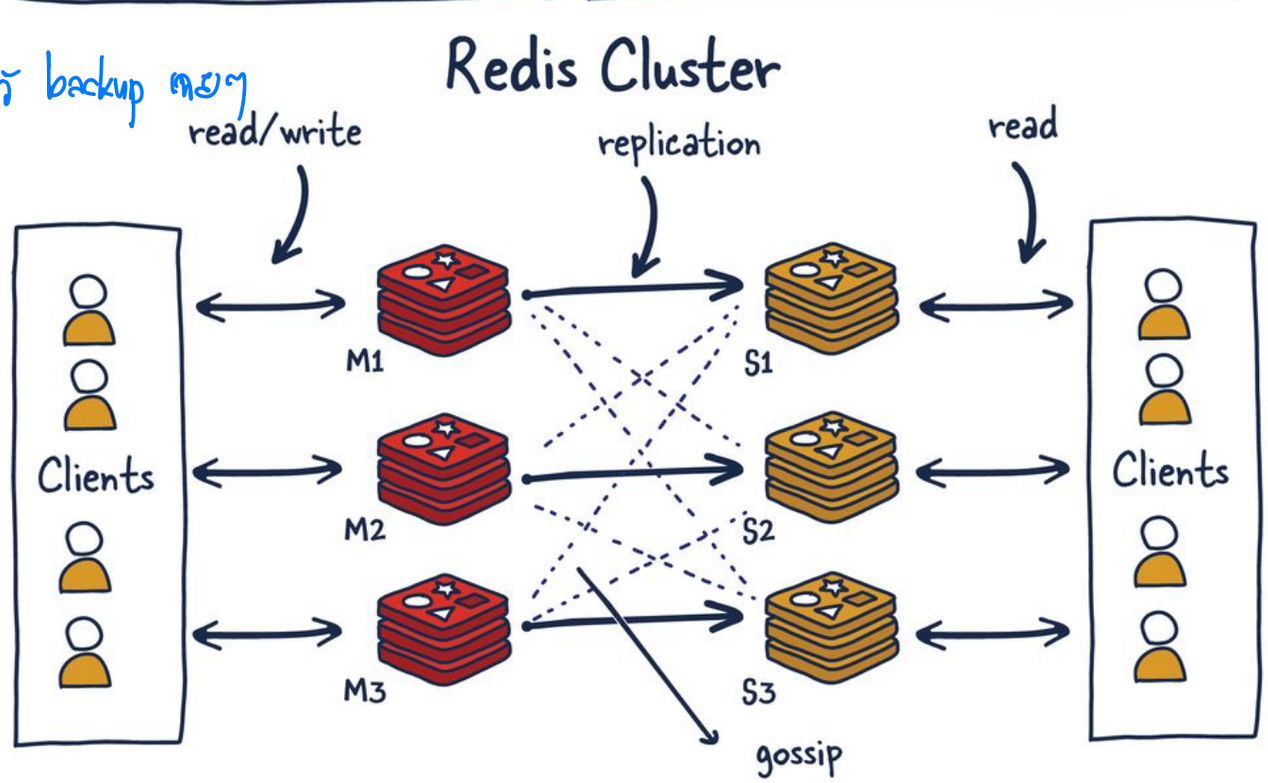

```
HMSET dish:101 id 101 name “Pad Thai” cuisine “thai” category “noodle”

SADD d_ing:101 “noodle;125;g”
SADD d_ing:101 “shrimp;150;g”
SADD d_ing:101 “brown sugar;3;tbsp”

SADD cuisine:thai “101”
```

ออกแบบโครงสร้าง Redis อย่างไรจะตอบยาก

Redis Architecture

- Redis supports flexible architecture
  - Single instance
  - High Availability
  - Sentinel
  - Cluster





Source: https://architecturenotes.co/redis/

Redis High Availability

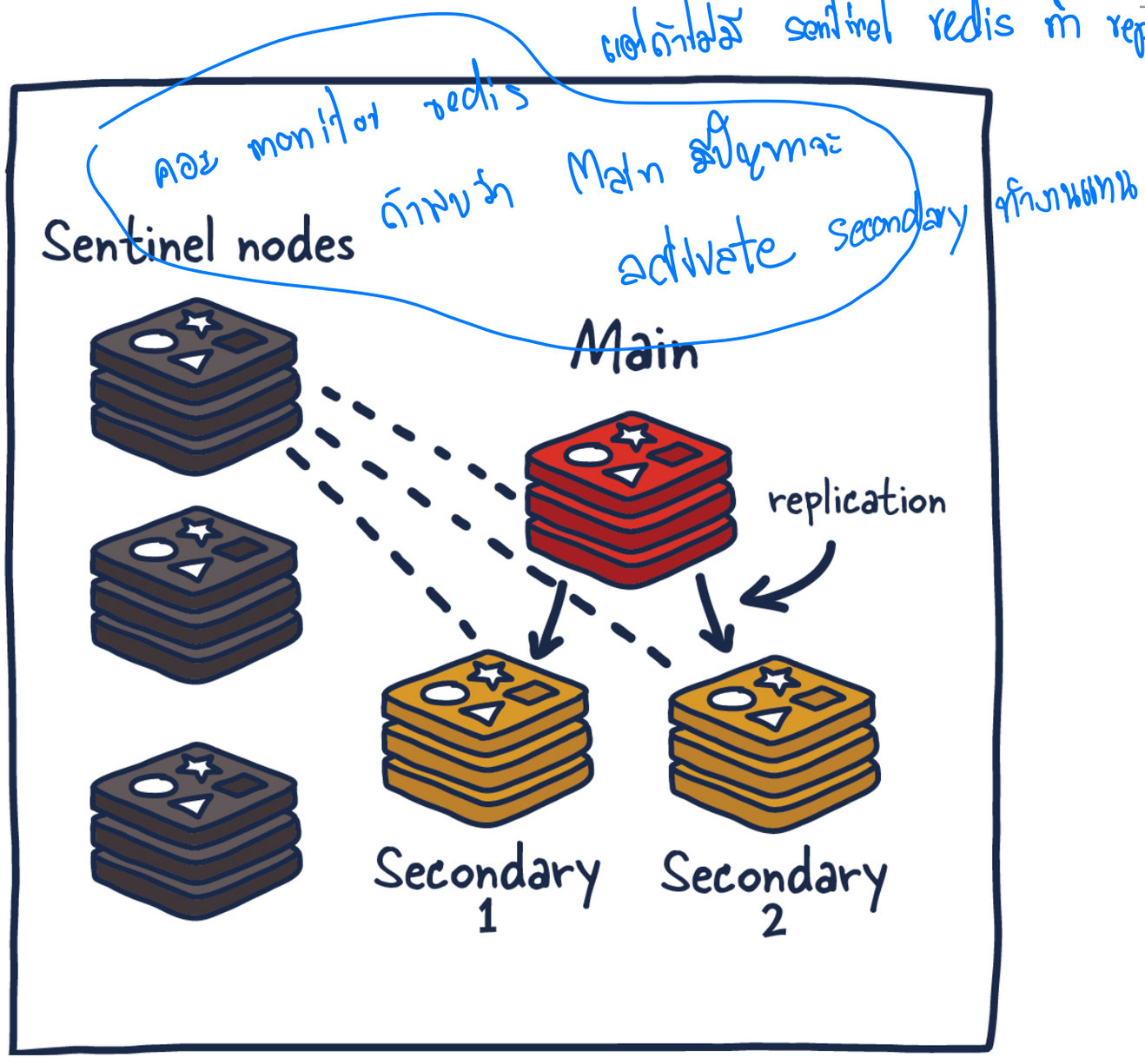
- When setup for HA, Redis consists of a master and one or more replicas and follows the following rules
  - Master performs all commands (read/write/etc.), replica performs only read commands
  - When a master and a replica instances are well-connected, the master keeps the replica updated by sending a stream of commands to the replica to replicate the effects on the dataset
- When the link between the master and the replica breaks, for network issues or because a timeout is sensed in the master or the replica, the replica reconnects and attempts to proceed with a partial resynchronization
- When a partial resynchronization is not possible, the replica will ask for a full resynchronization
- Redis uses asynchronous replication, with asynchronous replica-to-master acknowledges of the amount of data processed

Redis High Availability

- To ensure data synchronization, all servers keep data version (replication id, offset); equal data versions mean both servers hold the same data
  - When a server becomes a master, it generates a new replication ID; replica server inherits replication ID from its master once connected
  - When master updates its data, it increments offset by one
- When replicas connect to masters, they use the PSYNC command to send their old master replication ID and the offsets they processed so far
  - If the differences of offsets are not too far behind, partial synchronization is performed
  - If too far or mismatched replication ID, a full resynchronization happens: in this case the replica will get a full copy of the dataset, from scratch

สังเกตุ ถ้าเป็น main แล้วเป็น replica มันจะไม่ takeover

Redis Sentinel



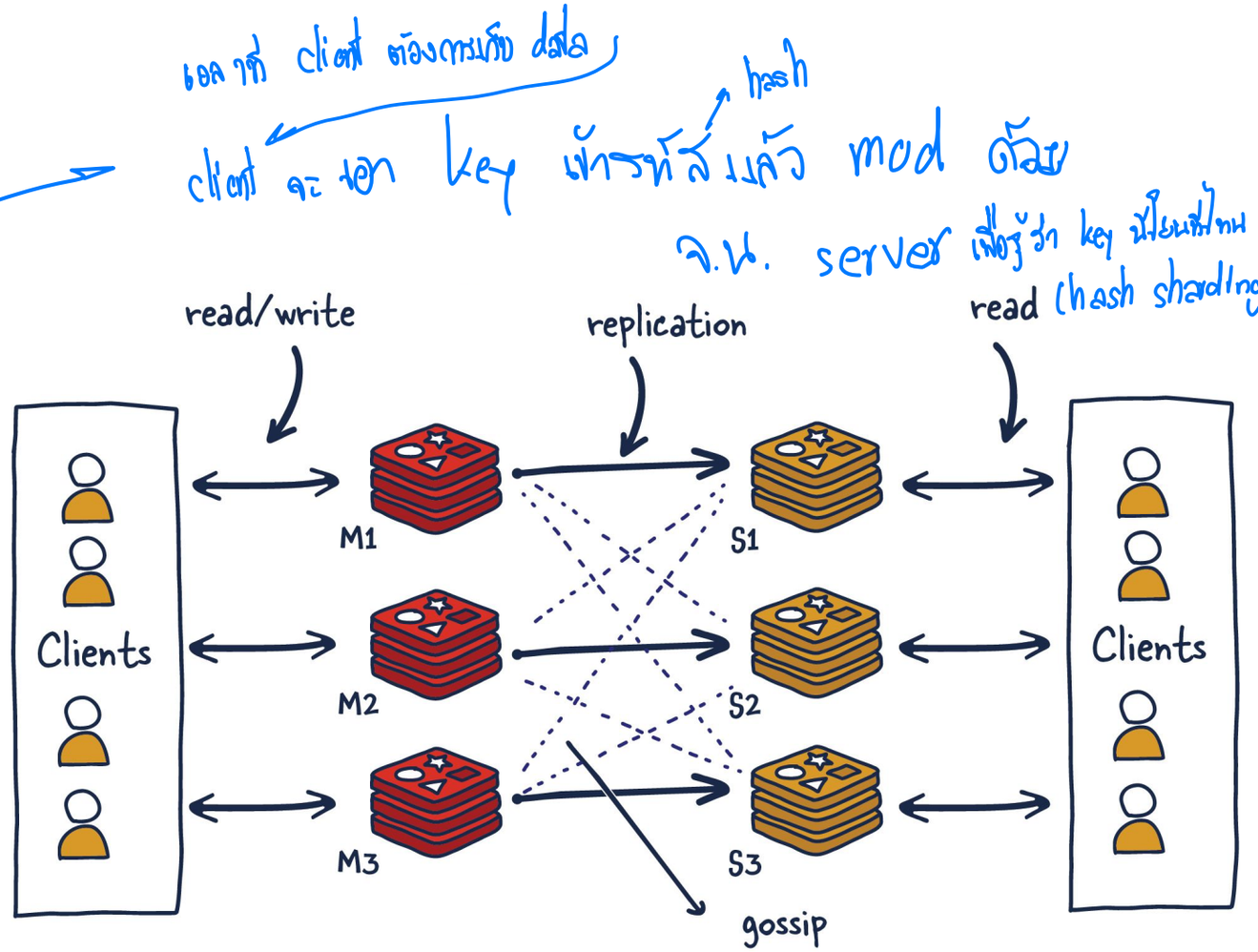
Source: https://architecturenotes.co/redis/

- A cluster of sentinel processes working together to coordinate states of Redis master (main) and replicas (secondaries)
- Focus on HA aspects with minimum number of nodes
- Support single master with multiple replicas with monitoring, notification, auto-failover, configuration providing
- Need at least three Sentinel instances for a robust deployment (quorum or agreement protocol)
- Can be deployed in Redis and client nodes

มี main หลาย node ทำงานพร้อมกัน (scale out)

Redis Cluster

- Support algorithmic sharding for horizontal scaling
  - Incoming data is hashed into a hash slot using deterministic hash function and mod with number of nodes
  - Data with a given key always maps to the same shard map ลง hash slot
  - Perform resharding when adding new nodes
- There are 16k hashslot; all data in the same hashslot always moves to the new node when resharding ชุดข้อมูลที่อยู่ใน กลุ่มเดียวกัน
- Nodes communicate with one another using gossiping protocol to determine the entire cluster's health
- Redis Cluster does not guarantee strong consistency



ข้อมูลเก็บทั้ง main และ secondary แต่ที่ที่ main ก่อน

Source: https://architecturenotes.co/redis/

และ main จะ gossip secondary
↳ propagate

user 1 คนใช้ทุก main

hash slot อยู่ในช่วง 0- 16k โดย redis จัดการให้

redis เลือกเป็น worker เพื่อเก็บงานที่ต้องทำ

References

- ScaleGrid, “Top Redis Use Cases by. Core Data Structure Types”, https://scalegrid.io/blog/top-redis-use-cases-by-core-data-structure-types/

- Jerry An, “The most important Redis data structures you must understand”, https://medium.com/analytics-vidhya/the-most-important-redis-data-structures-you-must-understand-2e95b5cf2bce

- Brad Solomon, “How to use Redis with python”, https://realpython.com/python-redis/

- M. Yusuf, “Redis Explained”, https://architecturenotes.co/redis/