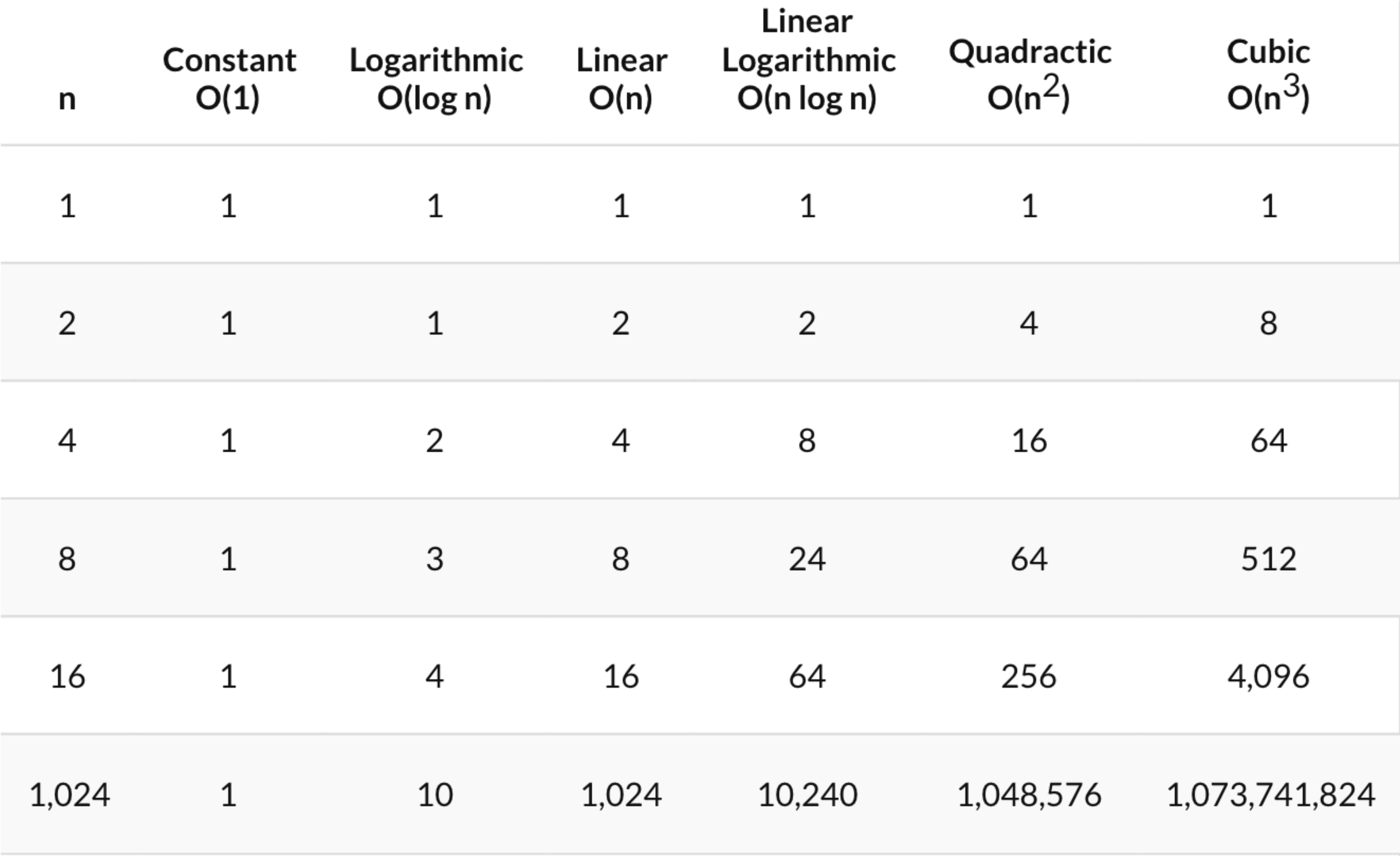
REDIS 101 - INTRO

ISMAIL ANJRINI

BIG O NOTATION



n	Constant O(1)	Logarithmic O(log n)	Linear O(n)	Linear Logarithmic O(n log n)	Quadractic O(n^2)	Cubic O(n^3)
1	1	1	1	1	1	1
2	1	1	2	2	4	8
4	1	2	4	8	16	64
8	1	3	8	24	64	512
16	1	4	16	64	256	4,096
1,024	1	10	1,024	10,240	1,048,576	1,073,741,824

OVERVIEW

- It means REmote DIctionary Server
- Originally Redis was started in order to scale LLOOGG
- Redis data types are closely related to fundamental data structures and are exposed to the programmer as such, without additional abstraction layers
- Redis is an in-memory but persistent on disk database
- Data sets can't be larger than memory
- The two on-disk storage formats (RDB and AOF) don't need to be suitable for random access
- There are no plans to create an on disk backend for Redis

MODULES

- RediSearch
- neural-redis
- RedisGraph
- rediSQL
- RedisJSON
- RedisBloom
- redis-cell

REDIS MEMORY FOOTPRINT

- An empty instance uses ~ 3MB of memory.
- 1 Million small Keys: String Value pairs use ~ 85MB of memory.
- 1 Million Keys: Hash value, representing an object with 5 fields, use ~ 160 MB of memory.
- Use the redis-benchmark utility to generate random data sets then check the space used with the INFO memory command

REDIS PERSISTENCE

REDIS PERSISTENCE

- The RDB persistence performs point-in-time snapshots of your dataset at specified intervals
- The AOF persistence logs every write operation received by the server, that will be played again at server startup, reconstructing the original dataset
- You can disable persistence completely
- It is possible to combine both AOF and RDB in the same instance
- AOF will be used to reconstruct the dataset after Redis restarts

REDIS PERSISTENCE - RDB

Advantages	**Disadvantages**
Very compact single-file point-in-time representation of your Redis data	RDB is NOT good if you need to minimize the chance of data loss
RDB is very good for disaster recovery	Fork() can be time consuming if the dataset is big
RDB maximizes Redis performances	
RDB allows faster restarts with big datasets compared to AOF	

REDIS PERSISTENCE - AOF

Advantages	**Disadvantages**
Using AOF Redis is much more durable	AOF files are usually bigger than the equivalent RDB files for the same dataset
The AOF log is an append only log	We experienced rare bugs in specific commands
Redis is able to automatically rewrite the AOF in background when it gets too big	
AOF contains a log of all the operations one after the other in an easy to understand and parse format	
You can even easily export an AOF file	

REDIS-CLI

- A simple program that allows to send commands to Redis, and read the replies sent by the server, directly from the terminal
- Determine host, port, password and database
- Getting input from other programs
- Continuously run the same command
- CSV output
- Interactive mode
  - Write redis-cli THEN enter
- Monitoring commands executed in Redis
- Remote backups of RDB files
- Slave mode
- redis-cli MONITOR

REDIS-CLI MONITOR

- Not monitored commands
  - AUTH
  - EXEC
  - HELLO
  - QUIT

KEYS

OVERVIEW

- Very long keys are not a good idea
- Use pattern for key names
  - user:2435:settings
- The maximum allowed key size is 512 MB

KEYS PATTERN

- O(N) with N being the number of keys in the database
- Returns all keys matching pattern
- Redis running on an entry level laptop can scan a 1 million key database in 40 milliseconds
- Should only be used in production environments with extreme care

SCAN CURSOR [MATCH PATTERN] [COUNT COUNT] [TYPE TYPE]

- O(1) for every call.
- O(N) for a complete iteration
  - N is the number of elements inside the collection
- SCAN is a cursor based iterator
- Returns only a small number of elements per call
- An iteration starts when the cursor is set to 0
- Terminates when the cursor returned by the server is 0

INFO [SECTION]

- Returns information and statistics about the server
- INFO section_name

DEL - UNLINK

- DEL - O(N)
- UNLINK - O(1)

EXISTS KEY [KEY ...]

- O(1)
- EXISTS k1
- EXISTS k1 k2

TYPE KEY

- O(1)
- SET name ismail
  - TYPE name -> string

FLUSHDB [ASYNC]

- O(N)
- Delete all the keys of the currently selected DB

DBs

DATA TYPES

OVERVIEW

- In Redis the value is not limited to a simple string
- Redis is actually a data structures server
  - Binary-safe strings.
  - Lists
  - Sets
  - Sorted sets
  - Hashes
  - Bit arrays (or simply bitmaps)
  - HyperLogLogs
  - Streams

STRINGS

- The Redis String type is the simplest type of value you can associate with a Redis key
- SET and the GET commands are the way we set and retrieve a string value
- Values can be strings (including binary data) of every kind
- A value can't be bigger than 512 MB
- Commands
  - set k1 value1
  - get k1

STRINGS (CONT.)

- SET
  - O(1)
  - EX, XX, NX, PX
- GET
  - O(1)
- INCR, INCRBYFLOAT, HINCRBY, HINCRBYFLOAT, DECR, DECRBY
  - O(1)

LISTS

- Redis lists are implemented via Linked Lists
  - The speed of adding a new element to the head of a list with ten elements is the same as adding an element to the head of list with 10 million elements.
- A List is just a sequence of elements
  - 1,4,67,0,3
- LPUSH (head),RPUSH (tail), LRANGE
- RPOP, LPOP
- LTRIM, RTRIM
- BLPOP, BRPOP

HASHES

- field-value pairs
- hmset, hmget
- hset, hget
- hincrby

SETS

- Sets are unordered collections of strings
- Intersection, union or difference between multiple sets
- Redis returns the elements in any order at every call from the same set

SORTED SETS

- Is similar to a mix between a Set and a Hash
- Every element in a sorted set is associated with a floating point value, called the score
- Sorted sets are ordered according to the following rule
    - If A and B are two elements with a different score, then A > B if A.score is > B.score.
    - If A and B have exactly the same score, then A > B if the A string is lexicographically greater than the B string.
        - A and B strings can't be equal since sorted sets only have unique elements.
- zadd, zrange

BITMAPS

- Bitmaps are not an actual data type, but a set of bit-oriented operations defined on the String type
- Bit operations
  - Constant-time single bit operations O(1)
    - Setting a bit to 1 or 0
    - Getting its value
  - Operations on groups of bits O(N)
    - Counting the number of set bits in a given range of bits

BITFIELDS

- GETBIT - O(1)
  - Returns 1 or 0
- SETBIT - O(1)
  - Change and returns the original bit value stored at offset
- BITCOUNT - O(N)
- BITFIELD - O(1)
- BITOP O(N)

0	0	0	0	0	0	0	0	0	0

```
setbit visits:2020:12:05 9 1
```

0	0	0	0	0	0	0	0	0	1

```
setbit visits:2020:12:06 9 1
```

0	0	0	0	0	0	1	1	0	0

```
setbit visits:2020:12:10 9 1
```

1	0	1	0	0	1	1	0	1	1

BITOP OR visits:2020:12 visits:2020:12:05 visits:2020:12:06 visits:2020:12:10

1	0	1	0	0	1	1	1	0	1

BITOP AND visits:2020:12 visits:2020:12:05 visits:2020:12:06 visits:2020:12:10

0	0	0	0	0	0	0	0	0	0

INDEXING

OVERVIEW

- Redis is not exactly a key-value store
- Implementing and maintaining indexes with Redis is an advanced topic
- Types
  - Sorted Sets as Indexes
  - Lexicographical indexes
  - Composite indexes

SIMPLE NUMERICAL INDEXES WITH SORTED SETS

34	66	64	22	38	80	21	15	99	44
Ismail	Ahmad	Fadi	Riyadh	Kasem	Ave	Sami	John	Emmy	So

```
ZRANGEBYSCORE employees 0 +inf
```

```
ZRANGEBYSCORE employees 22 50
```

USING OBJECTS IDs AS ASSOCIATED VALUES (HASHES)

age	34	66	64	22	38	80	21	15	99	44
name	Ismail	Ahmad	Fadi	Riyadh	Kasem	Ave	Sami	John	Emmy	So
score	33	22	56	45	90	132	98	243	59	59
id	1	2	3	4	5	6	7	8	9	10

34	66	64	22	38	80	21	15	99	44
1	2	3	4	5	6	7	8	9	10

```
ZRANGEBYSCORE users 0 +inf

ZRANGEBYSCORE users 22 50
```

```
HGET emps:2 name

HGETALL emps:1
```

LEXICOGRAPHICAL INDEXES

- When elements are added with the same score to set they are sorted lexicographically
- Redis is comparing the strings as binary data with the memcmp() function
- If the common prefix of two strings is the same then the longer string is considered the greater of the two
    - school, schooling
        - schooling is greater than school
    - John, J
        - John is greater than J

0	0	0	0	0	0	0	0	0	0
Ismail	Ahmad	Fadi	Riyadh	Kasem	Ave	Sami	John	Emmy	So

```
ZRANGEBYSCORE employees 0 +inf
```

```
 1) "Ahmad"
 2) "Ave"
 3) "Emmy"
 4) "Fadi"
 5) "Ismail"
 6) "John"
 7) "Kasem"
 8) "Riyadh"
 9) "Sami"
10) "So"
```

```
ZRANGEBYLEX employees [A [John
```

```
1) "Ahmad"
2) "Ave"
3) "Emmy"
4) "Fadi"
5) "Ismail"
6) "John"
```

```
ZRANGEBYLEX employees [A "[A\xff"
```

```
1) "Ahmad"
2) "Ave"
```

65890345	9678	5475234	785432	76400923
math	dev	ops	micro	lang

```
ZRANGEBYSCORE numbers 0 +inf

1) "5475234:ops"
2) "65890345:math"
3) "76400923:lang"
4) "785432:micro"
5) "9678:dev"
```

65890345	00009678	05475234	00785432	76400923
math	dev	ops	micro	lang

```
ZRANGEBYSCORE numbers 0 +inf

1) "00009678:dev"
2) "00785432:micro"
3) "05475234:ops"
4) "65890345:math"
5) "76400923:lang"
```

COMPOSITE INDEXES

age	0034	0066	0066	0066	0038	0038	0038	0015	0015	0015
score	0033	0022	0056	0045	0090	0132	0098	0243	0059	0059
id	1	2	3	4	5	6	7	8	9	10

```
ZRANGEBYLEX emps [0040:0030.00 [0040:0070.00

(empty array)
```

```
ZRANGEBYLEX emps [0066:0030.00 [0066:0050.00

0066:0045:4
```

RELATIONSHIP - ONE-TO-MANY

user:user_id:products

```
sadd user:1:products 1

sadd user:1:products 2
```

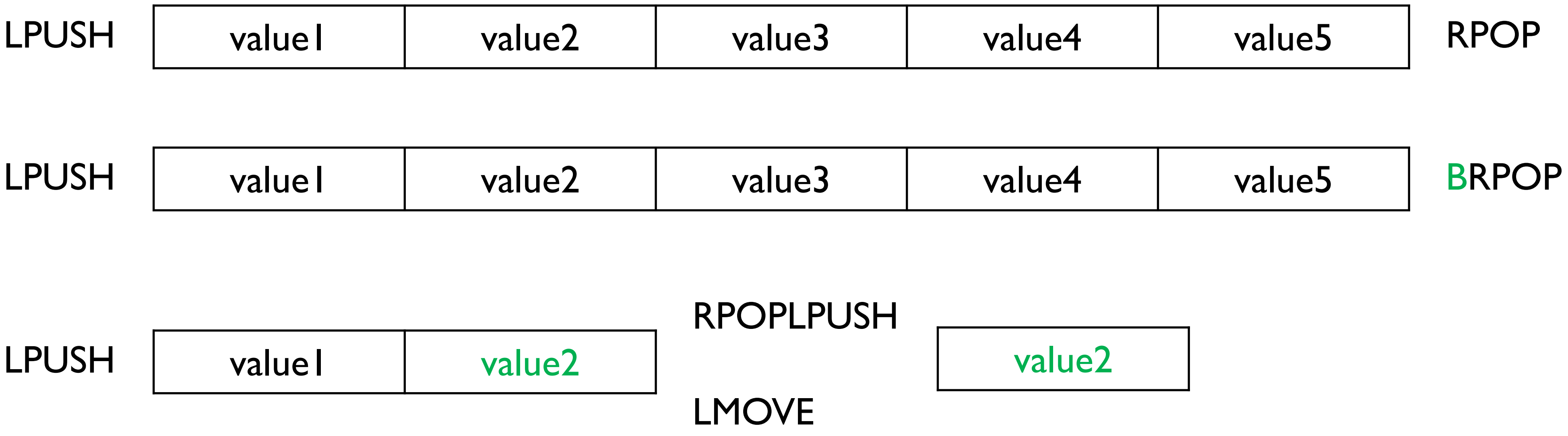
product:prodeuct_id:users

```
sadd product:1:users 45

sadd product:1:users 33

sadd product:3:users 33
```

QUEUES



LPUSH value1 value2 value3 value4 value5 RPOP

LPUSH value1 value2 value3 value4 value5 BRPOP

LPUSH value1 value2 RPOPLPUSH LMOVE value2

PUB/SUB

Senders (publishers) are not programmed to send their messages to specific receivers

Publishing on db 10, will be heard by a subscriber on db 1

Consumers (subscribers) will not receive old messages which have been sent before subscription

SUBSCRIBE

UNSUBSCRIBE

PUBLISH

PSUBSCRIBE

PUNSUBSCRIBE

PUBSUB

STREAMS

Streams are primarily an append only data structure

xadd requests * key/value key/value

stream id: <millisecondsTime>-<sequenceNumber>

Read mode

- Fan out messages to multiple clients
- Get messages by ranges of time
- Reading messages via consumer groups

Fan out messages to multiple clients

- Every new item, by default, will be delivered to every consumer that is waiting for data in a given stream
- XREAD
- XREAD BLOCK

Consumer groups

- Each message is served to a different consumer so that it is not possible that the same message will be delivered to multiple consumers
- Consumers are identified, within a consumer group, by a name
- Each consumer group has the concept of the first ID never consumed
- Consuming a message, requires an explicit acknowledgment using a XACK command
- A consumer group tracks all the messages that are currently pending

USE CASES

Leaderboard

```
hset player:100 name ismail mobile 0559876543

hset player:200 name fadi mobile 0545670987

hset player:300 name othman mobile 0523463445
```

→ Hash

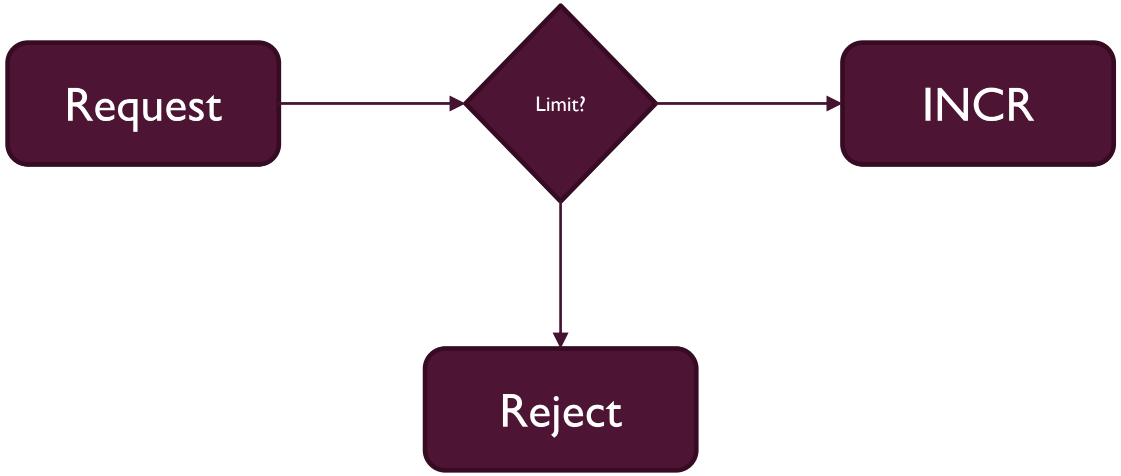
```
zadd players 250 100

zadd players 310 200

zadd players 200 300
```

→ Sorted set

rate-limiter



get key:10

incr key:10

expire key:10 60

ttl key:10

DEBUGGING

TYPE	MONITOR	OBJECT	INFO

PERFORMANCE

MEMORY USAGE key [SAMPLES count]

Reports the number of bytes that a key and its value require to be stored in RAM

set counter 1

memory usage counter

↓

(integer) 56

set c 1

memory usage c

↓

(integer) 48

zadd games 1 fortnite

memory usage games

↓

(integer) 71

zadd games 1 duty

memory usage games

↓

(integer) 79

EVICTION POLICIES

maxmemory

maxmemory-policy

LRU CACHE

noeviction

allkeys-lru

volatile-lru

allkeys-random

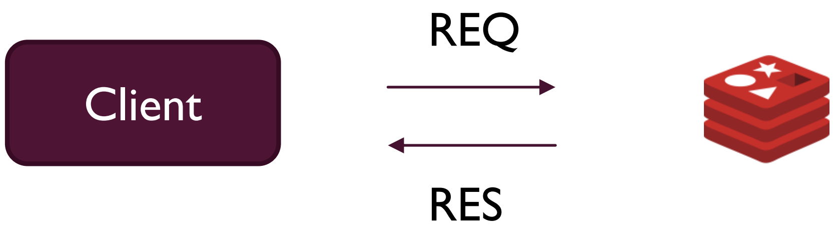
volatile-ttl

LFU CACHE

volatile-lfu

allkeys-lfu

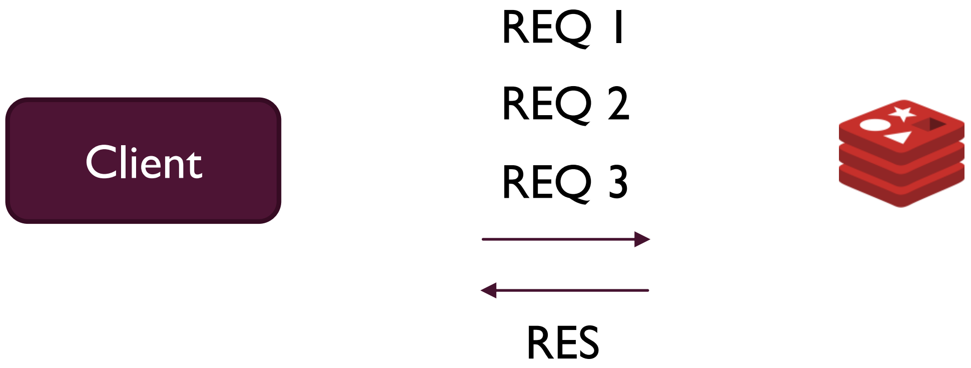
PIPELINING



Client

REQ

RES



Client

REQ 1

REQ 2

REQ 3

RES

RTT (Round Trip Time)

TRANSACTIONS

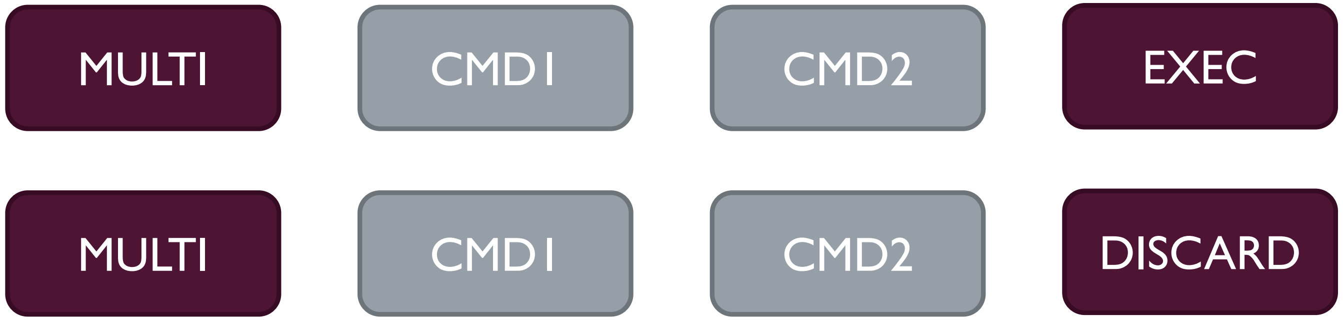
All the commands in a transaction are serialized and executed sequentially

Either all of the commands or none are processed

Redis makes sure to use a single syscall to write the transaction on disk

Using the redis-check-aof tool to the partial transaction

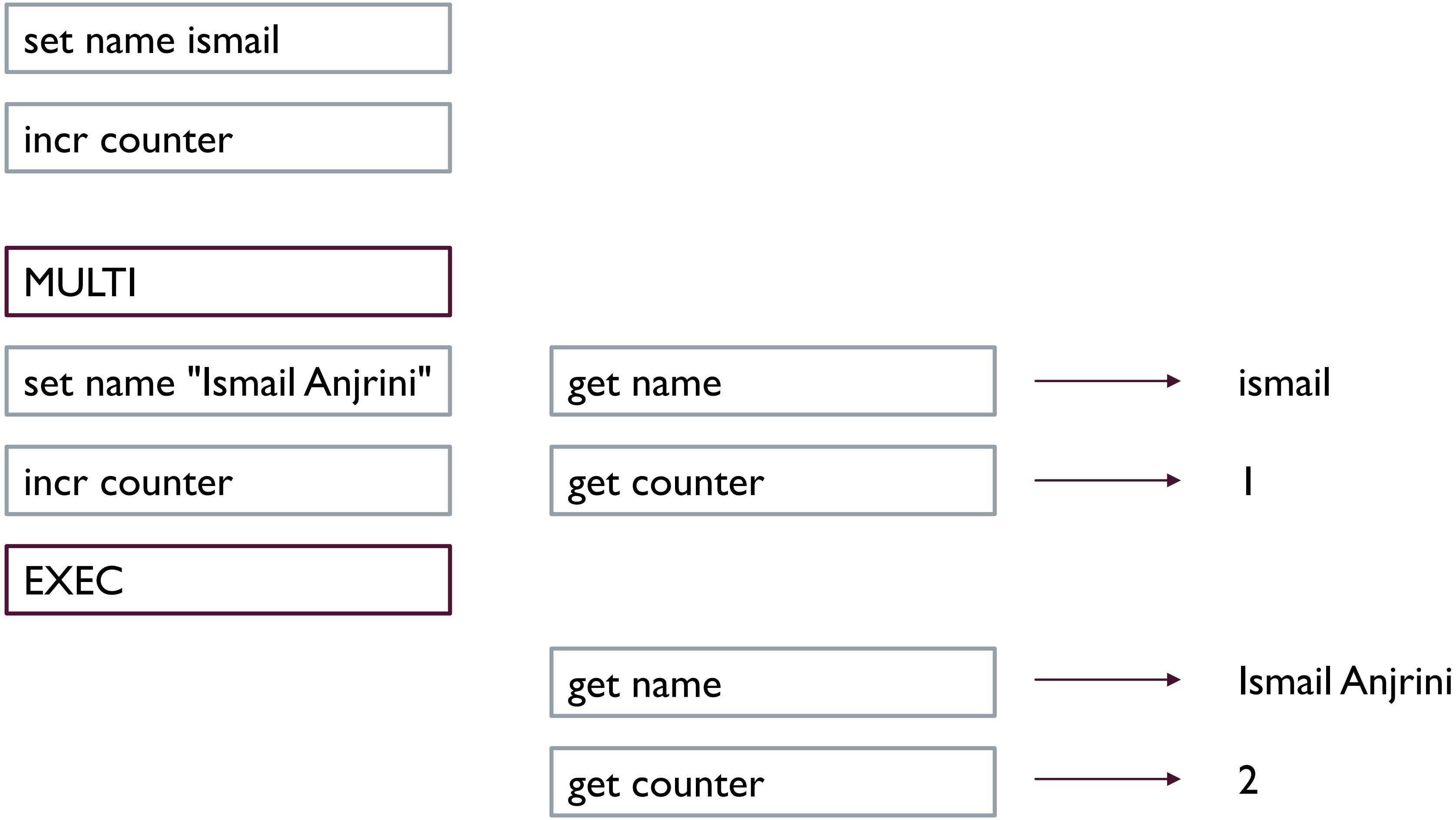
Redis allows optimistic locking to a check-and-set (CAS) operation



MULTI CMD1 CMD2 EXEC

MULTI CMD1 CMD2 DISCARD

Even when a command fails, all the other commands in the queue are processed



set name ismail

incr counter

MULTI

set name "Ismail Anjrini"

incr counter

EXEC

get name → ismail

get counter → 1

get name → Ismail Anjrini

get counter → 2

TRANSACTION ERRORS

A command may fail to be queued

```
inc counter
```

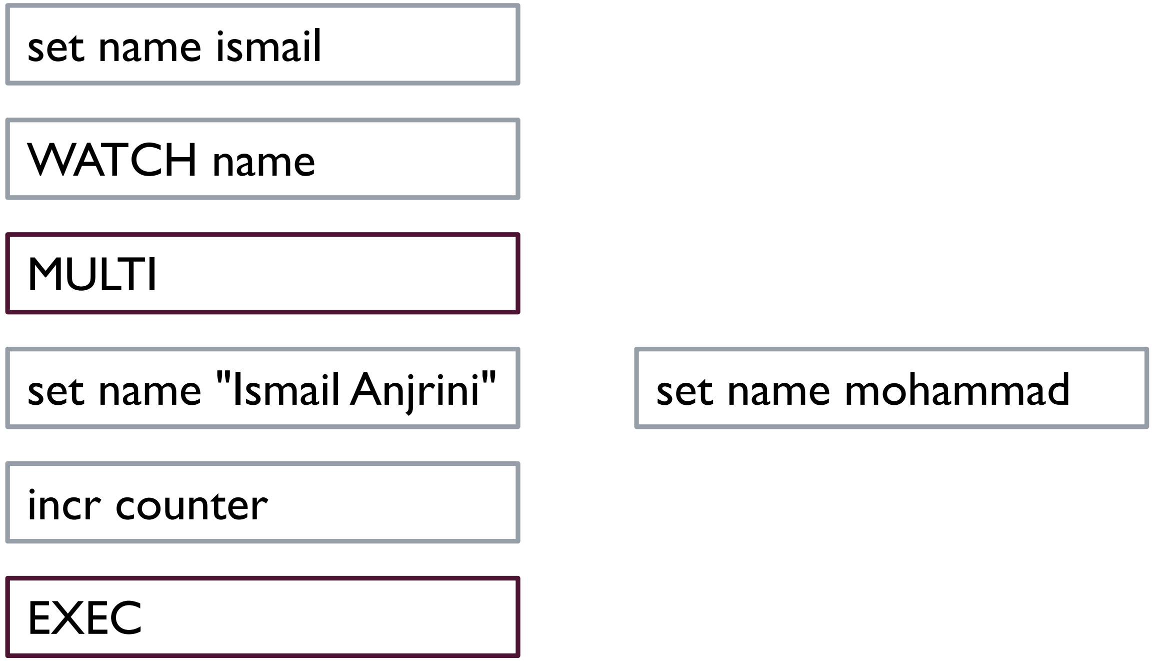
A command may fail after EXEC is called

```
set name ismail

incr name
```

Redis does not support roll backs

OPTIMISTIC LOCKING

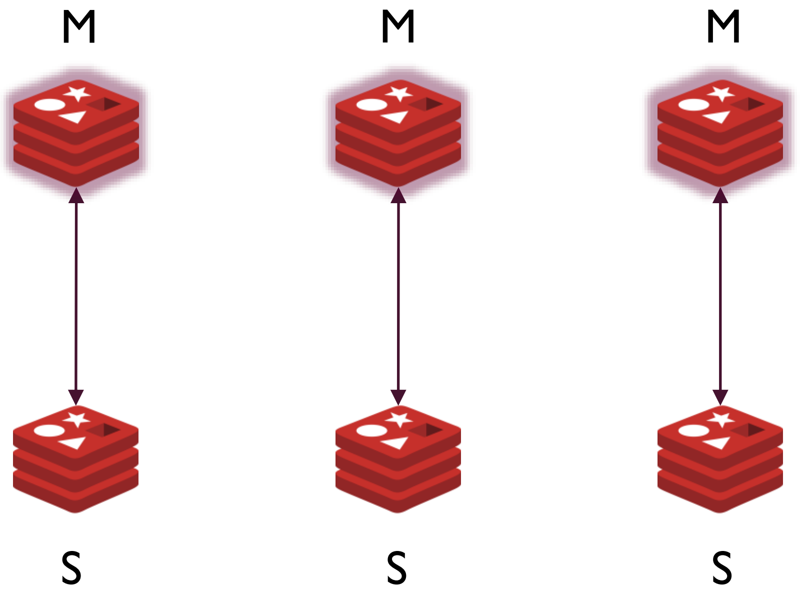


```
set name ismail
WATCH name
MULTI
set name "Ismail Anjrini"        set name mohammad
incr counter
EXEC
```

When EXEC is called, all keys are UNWATCHed

Use the UNWATCH command (without arguments) to flush all the watched keys

REDIS CLUSTERS


M
M
M
S
S
S

REDIS CLUSTER GOALS

High performance

Linear scalability up to 1000 nodes

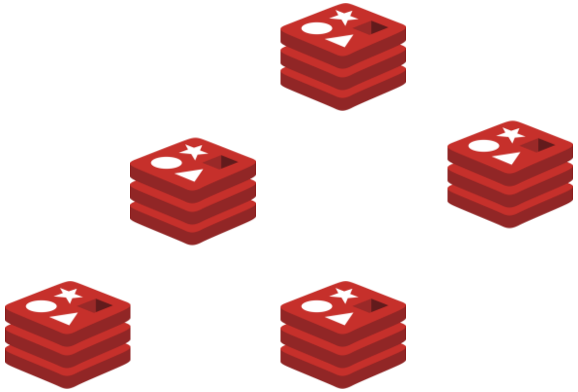
There are no proxies

Asynchronous replication

No merge operations are performed on values

Acceptable degree of write safety

Availability



NODES ROLES

Nodes are responsible for holding the data

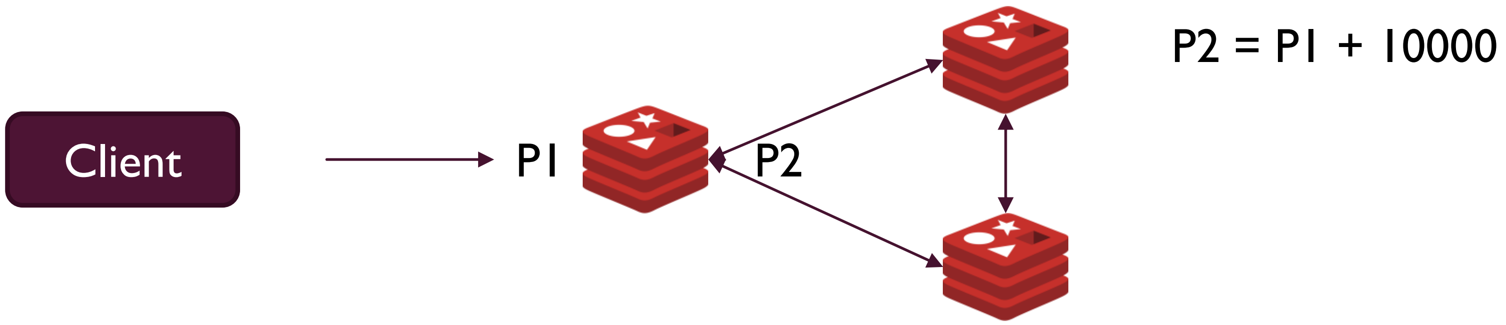
Holding the state of the cluster

Nodes are also able to auto-discover other nodes

Detect non-working nodes, and promote slave nodes to master when needed

Every node is connected to every other node in the cluster using the cluster bus

Every node requires two TCP connections open



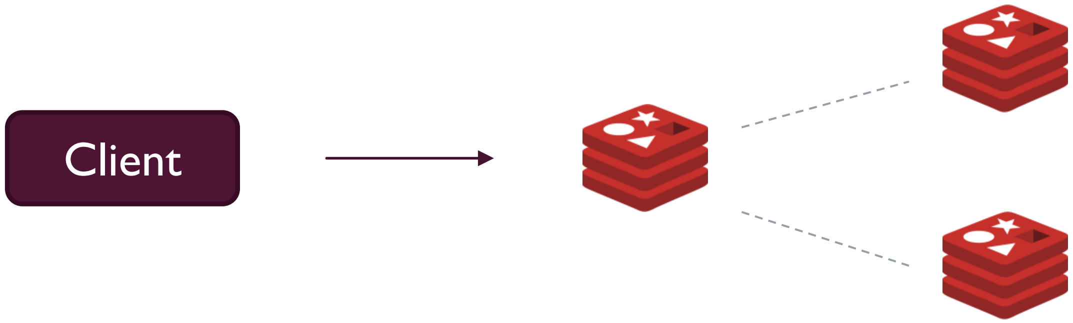
Redis Cluster Bus

WRITE SAFETY

Redis Cluster uses Asynchronous replication between nodes

Last failover wins

Losing writes during partitions

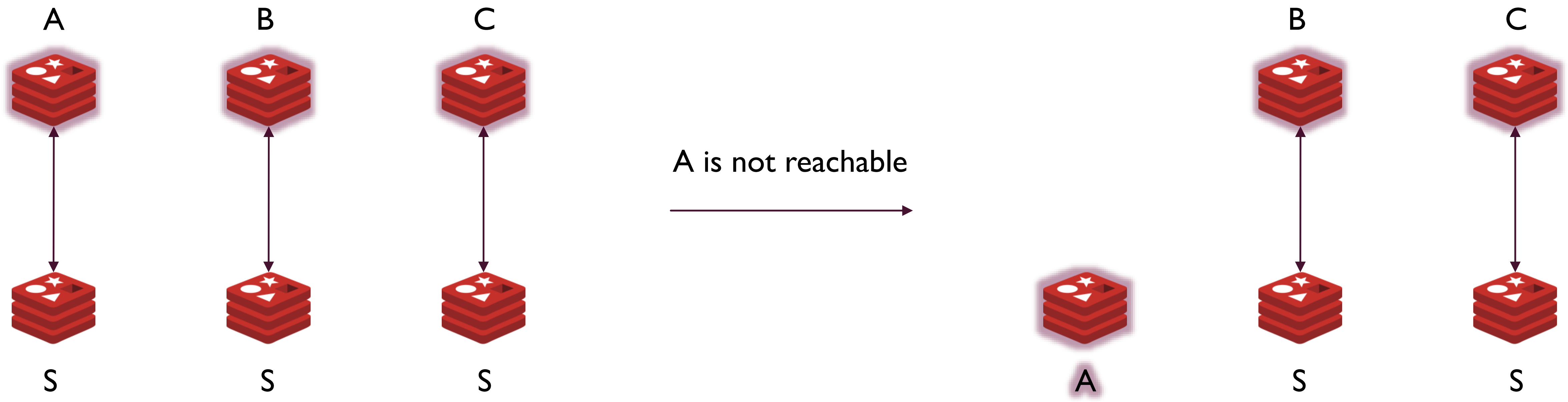


Master node try to reply to clients and slaves at about the same time

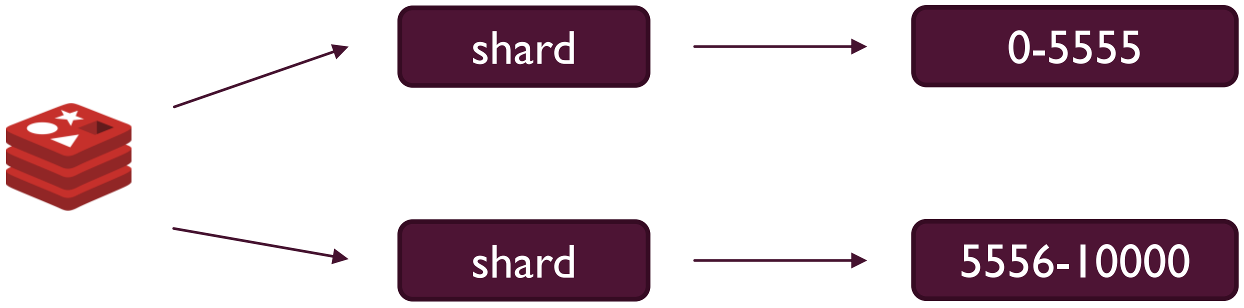
AVAILABILITY

Every master node in a Redis cluster has at least one slave node

Redis Cluster is designed to survive failures of a few nodes in the cluster



KEYS DISTRIBUTION



shard

0-5555

shard

5556-10000

The key space is split into 16384 slots

Each master node in a cluster handles a subset of the 16384 hash slots

HASH_SLOT = CRC16(key) mod 16384

KEYS DISTRIBUTION

Hash tags are a way to ensure that multiple keys are allocated in the same hash slot

What is between the first occurrence of { and the following first occurrence of } is hashed

IF the key contains a { character

AND IF there is a } character to the right of {

AND IF there are one or more characters between the first occurrence of { and the first occurrence of }

{user1000}.following {user1000}.followers → user1000

foo{}{bar} → bar

foo{{bar}}zap → {bar

foo{bar}{zap} → bar

NODES ATTRIBUTES

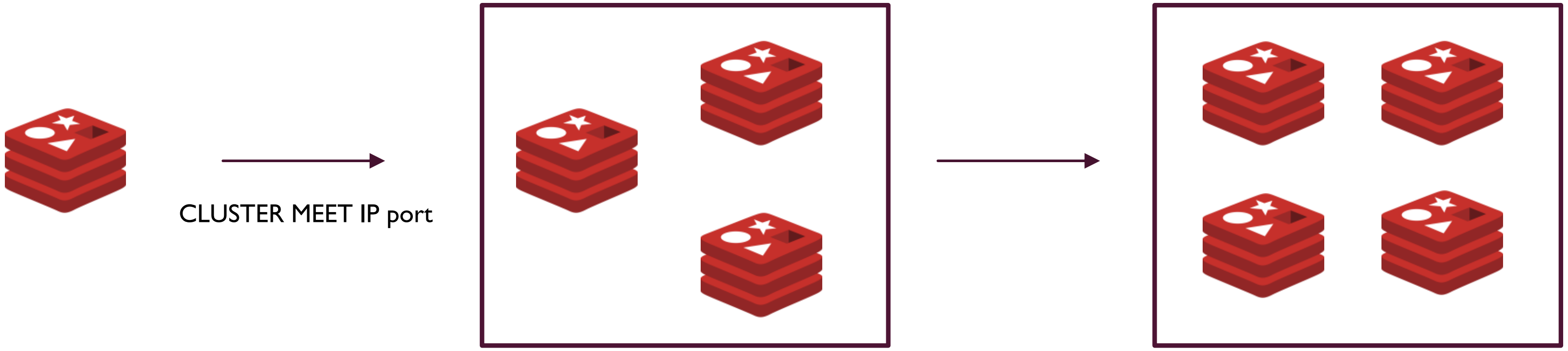
Every node has a unique name - hex representation of a 160 bit random number

The node ID is used to identify every node across the whole cluster

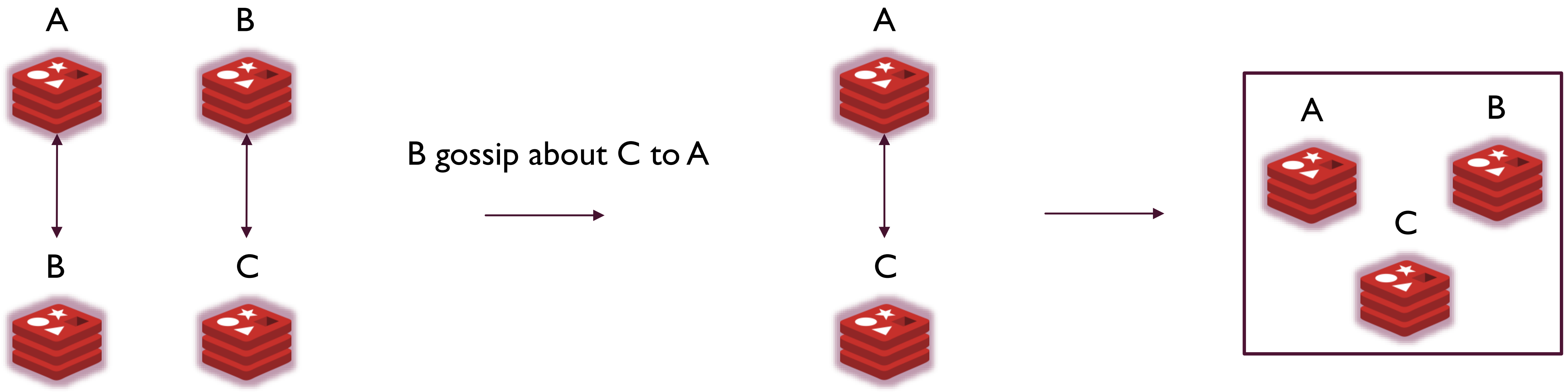
Every node maintains the following information about other nodes

- Node ID
- IP
- Port
- Set of flags
- Last time the node was pinged
- Set of hash slots served

NODES HANDSHAKE

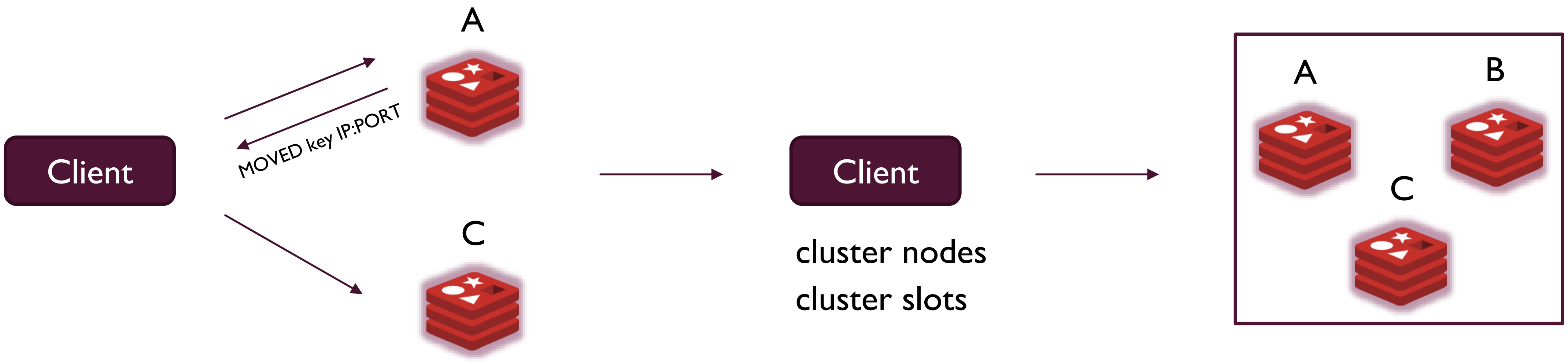


NODES HANDSHAKE



B gossip about C to A

REDIRECTION



Client

A

MOVED key IP:PORT

C

Client

cluster nodes

cluster slots

A

B

C

SECURITY

Redis is designed to be accessed by trusted clients inside trusted environments

Network security Port Bind IP

Authentication feature AUTH Password

TLS support

Disabling/Renaming of specific commands

ACL