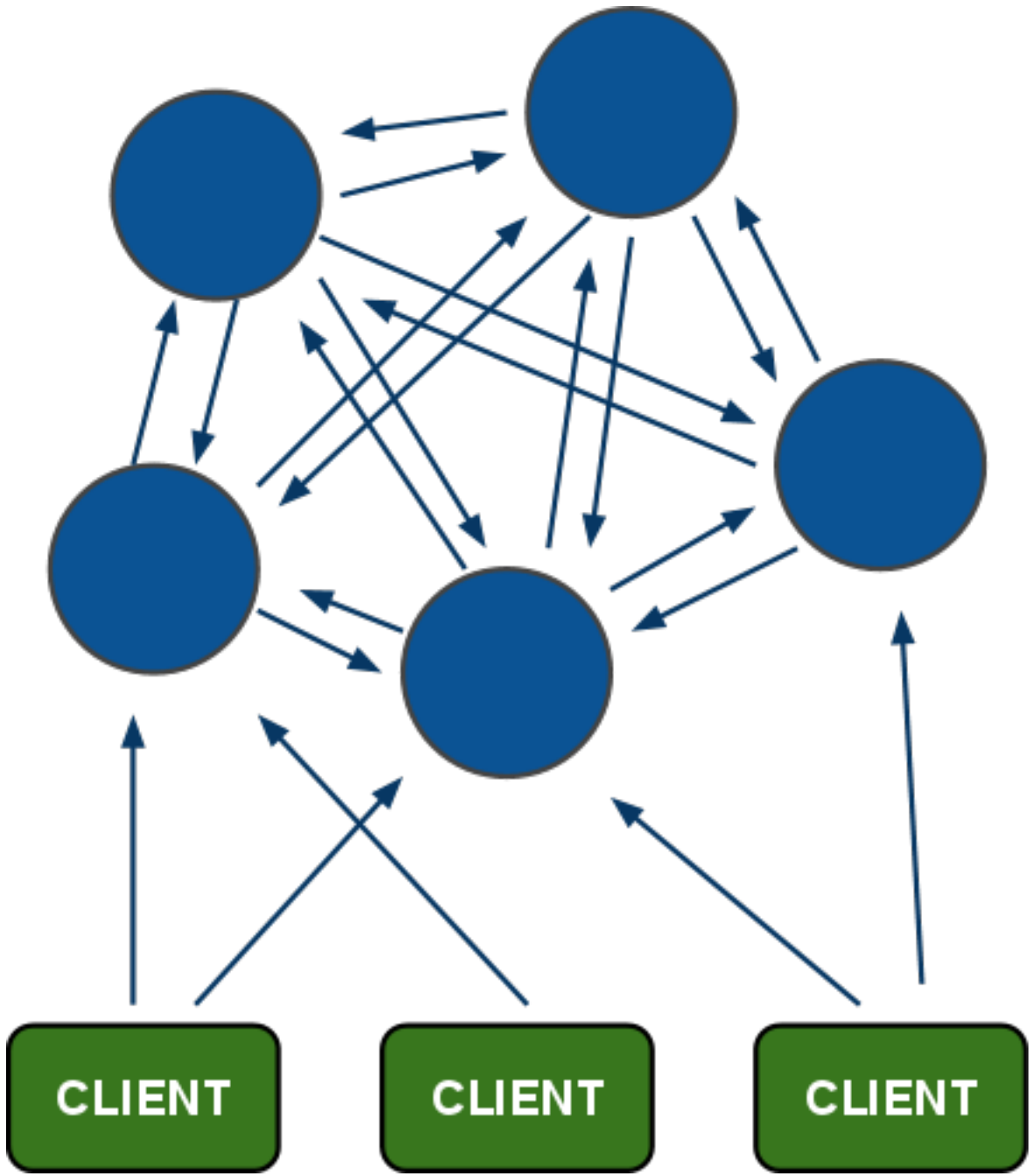
Redis Cluster

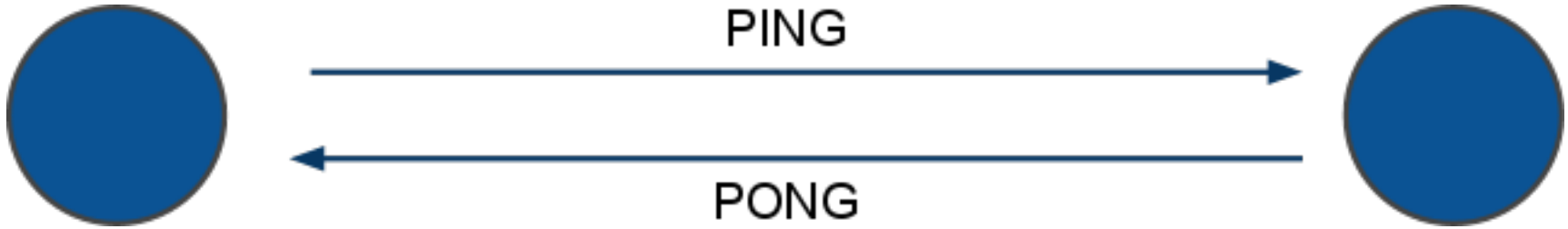
a pragmatic approach to distribution

redis



- All nodes are directly connected with a service channel.
- TCP baseport+4000, example 6379 -> 10379.
- Node to Node protocol is binary, optimized for bandwidth and speed.
- Clients talk to nodes as usually, using ascii protocol, with minor additions.
- Nodes don't proxy queries.

What nodes talk about?



PING: are you ok dude?
I'm master for XYZ hash slots.
Config is FF89X1JK

Gossip: this are info about other nodes I'm in touch with:

A replies to my ping, I think its state is OK.
B is idle, I guess it's having problems but I need some ACK.

PONG: Sure I'm ok dude!
I'm master for XYZ hash slots.
Config is FF89X1JK

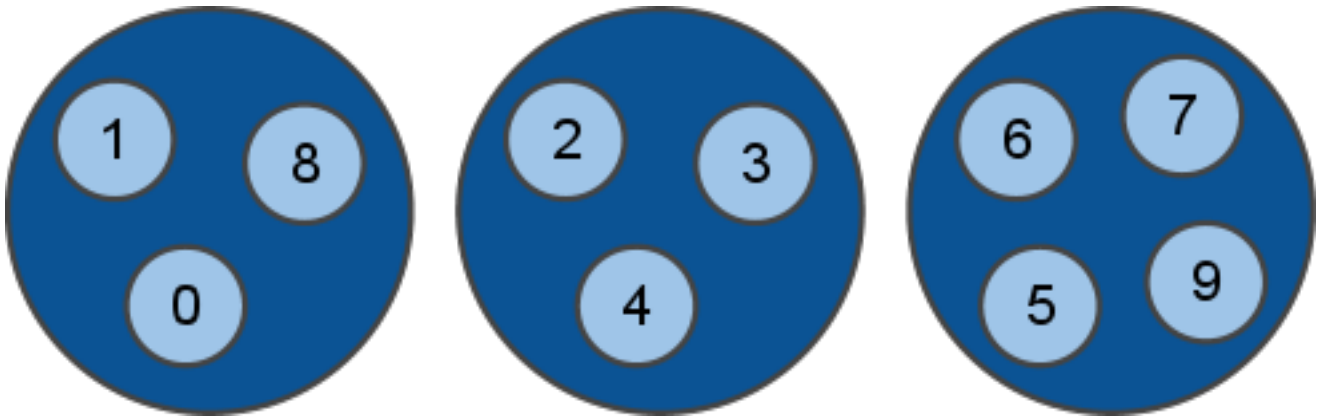
Gossip: I want to share with you some info about random nodes:

C and D are fine and replied in time.
But B is idle for me as well!
IMHO it's down!.

Hash slots

keyspace is divided into 4096 hash slots. But in this example we'll assume they are just ten, from 0 to 9 ;)

Different nodes will hold a subset of hash slots.

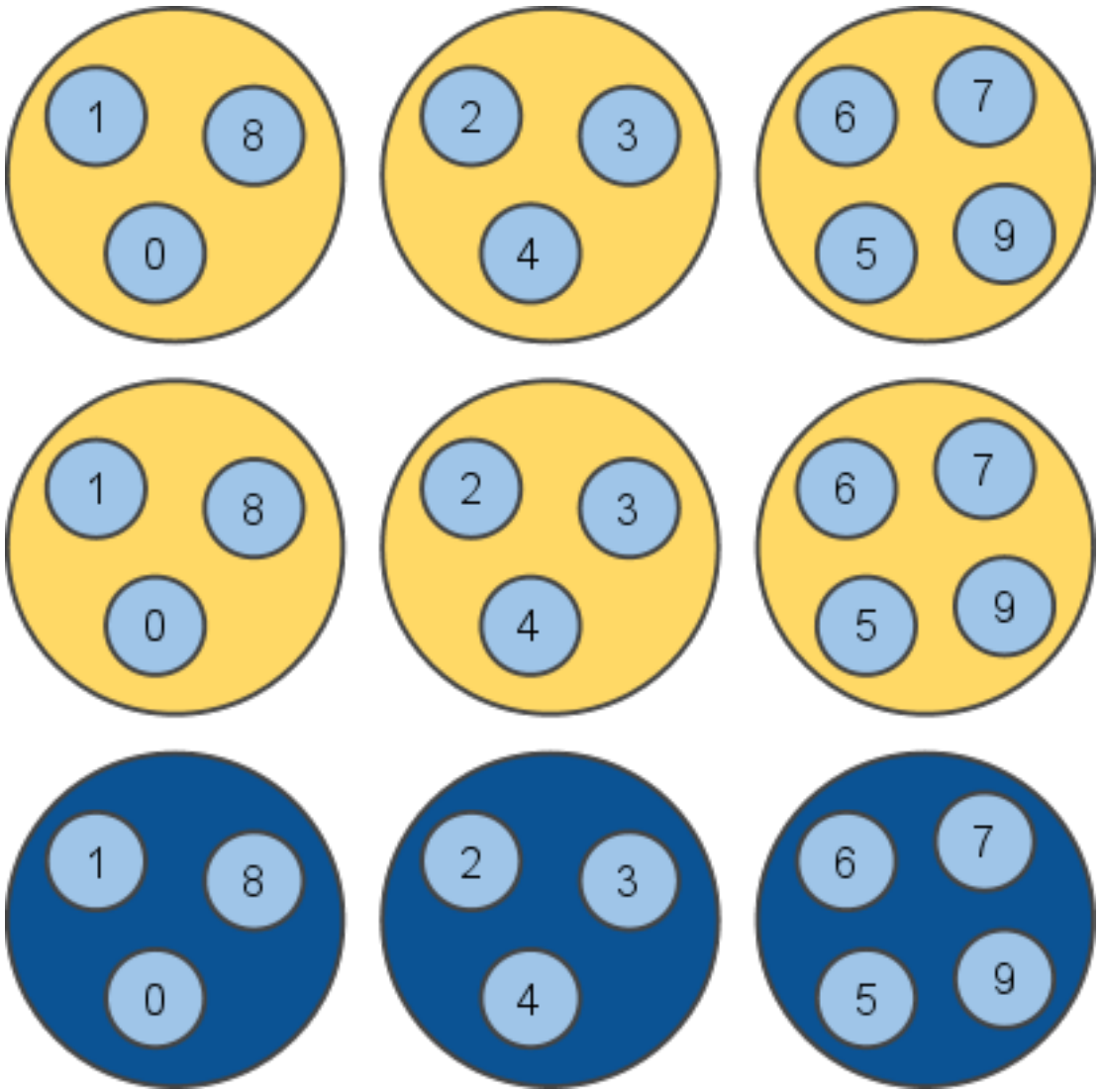


A given key "foo" is at slot:

slot = crc16("foo") mod NUMER_SLOTS

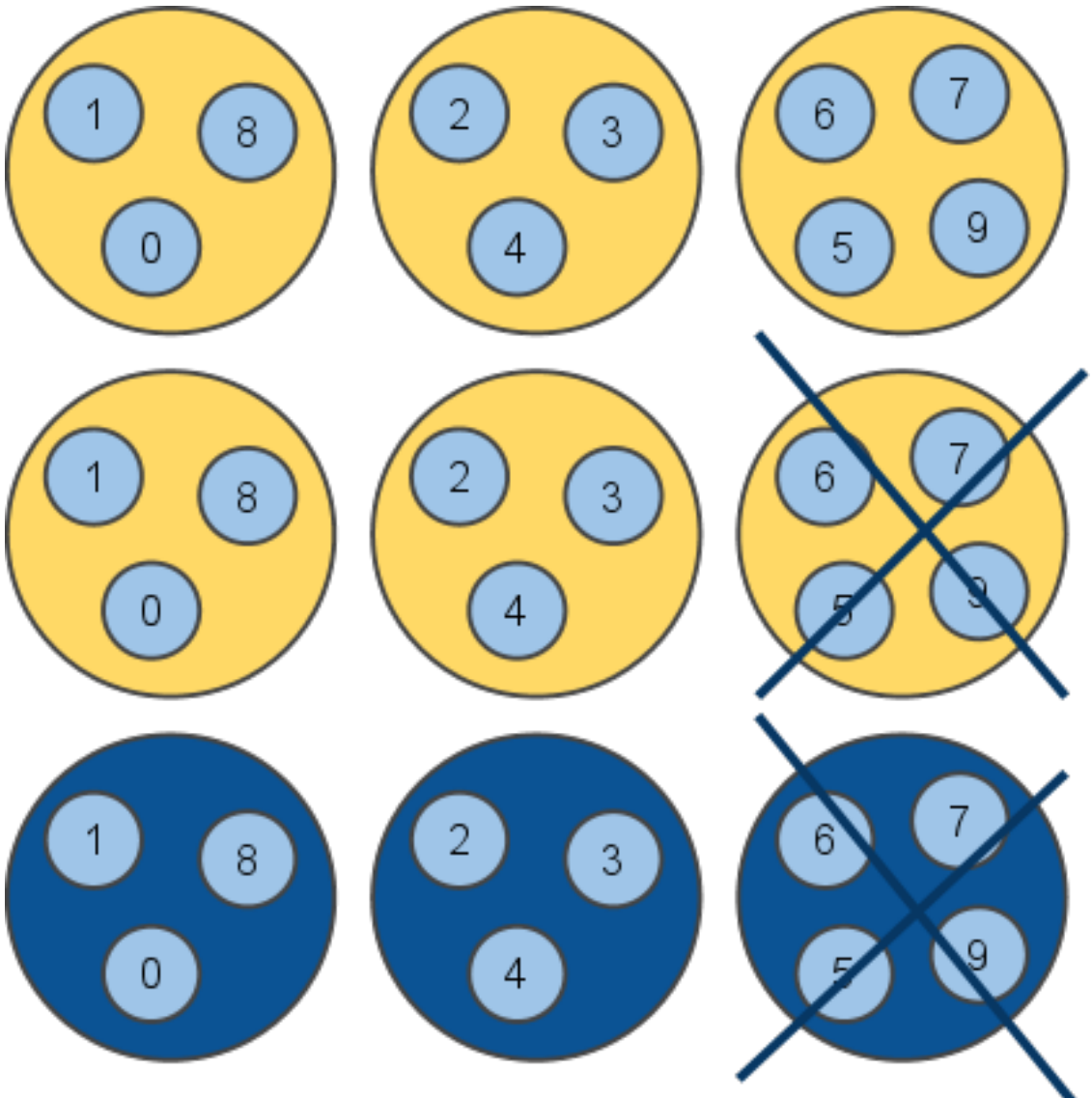
Master and Slave nodes

Nodes are all connected and functionally equivalent, but actually there are two kind of nodes: slave and master nodes:



Redundancy

In the example there are two replicas per every master node, so up to **two random nodes can go** down without issues.

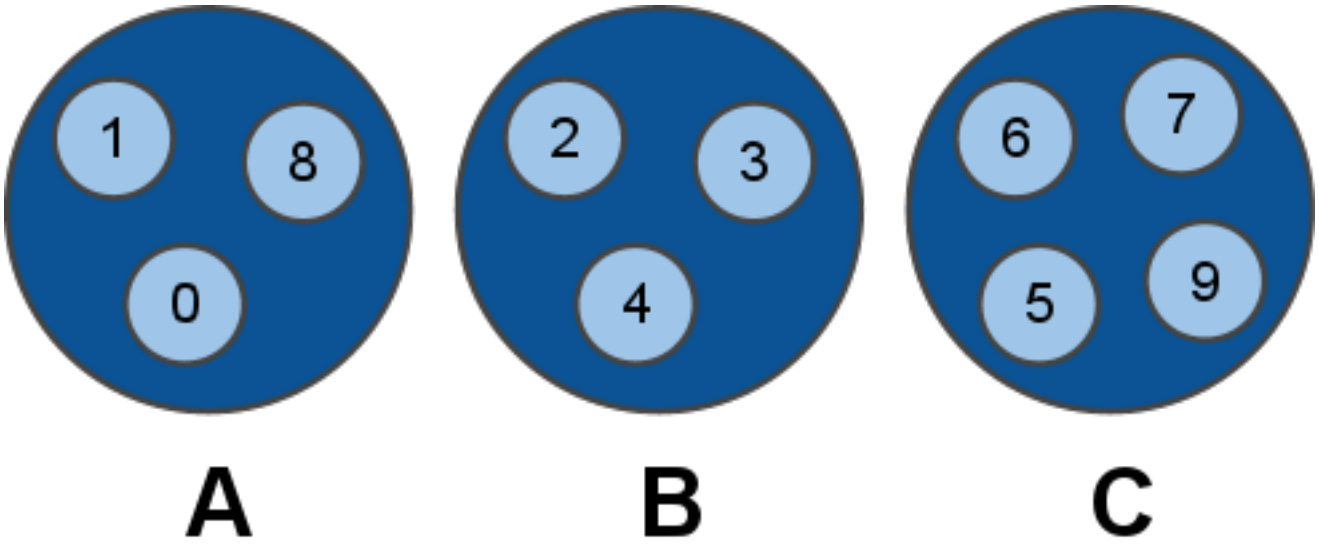


Working with two nodes down is guaranteed, but in the best case the cluster will continue to work as long as there is at least one node for **every hash slot**.

What this means so far?

- Every key only exists in a single instance, plus N replicas that will never receive writes. So there is **no merge, nor application-side inconsistency resolution**.
- The price to pay is not resisting to net splits that are bigger than *replicas-per-hashslot* nodes down.
- Master and Slave nodes use the Redis Replication you already know.
- Every physical server will usually hold multiple nodes, both slaves and masters, but the *redis-trib* cluster manager program will try to allocate slaves and masters so that the replicas are in different physical servers.

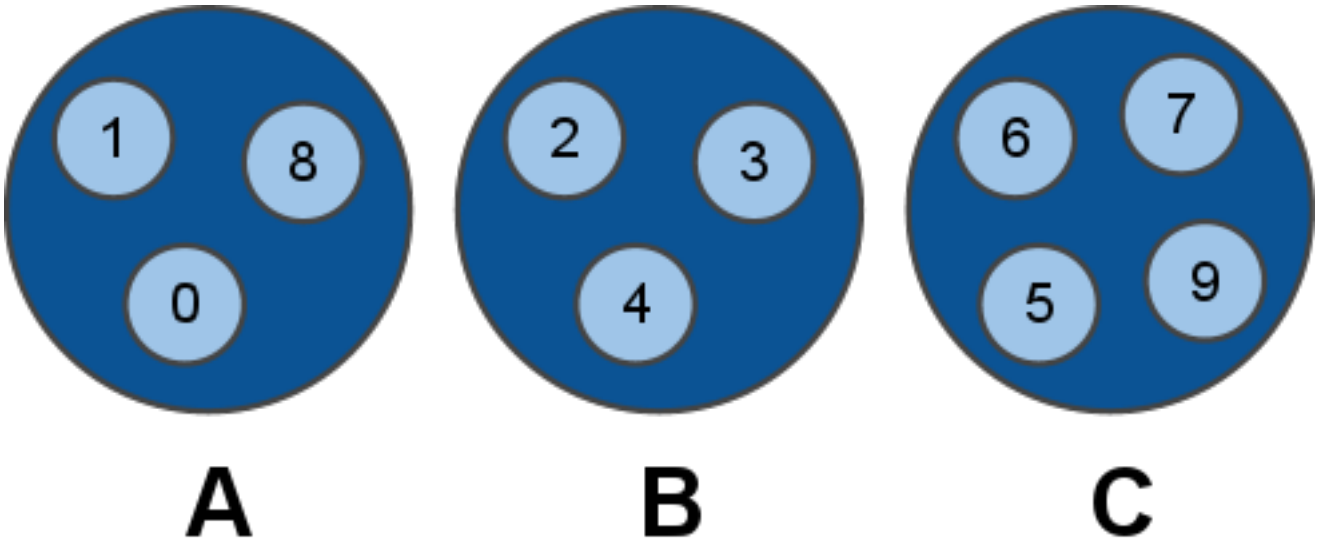
Client requests - dummy client



1. Client => A: **GET foo**
2. A => Client: **-MOVED 8 192.168.5.21:6391**
3. Client => B: **GET foo**
4. B => Client: **"bar"**

-MOVED 8 ... this error means that hash slot 8 is located at the specified IP/port, and the client should reissue the query there.

Client requests - smart client



1. Client => A: **CLUSTER HINTS**
2. A => Client: **... a map of hash slots -> nodes**
3. Client => B: **GET foo**
4. B => Client: **"bar"**

Client requests

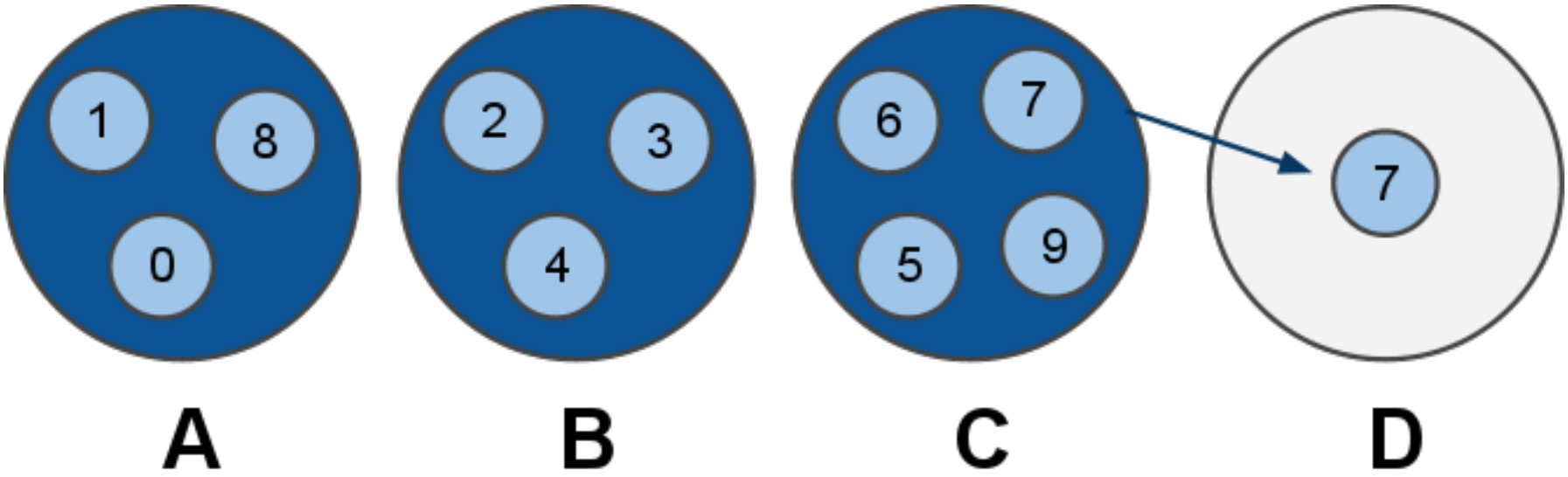
- Dummy, single-connection clients, will work with minimal modifications to existing client code base. Just try a random node among a list, then reissue the query if needed.

- Smart clients will take persistent connections to many nodes, will cache *hashslot -> node* info, and will update the table when they receive a -MOVED error.

- This schema is always horizontally scalable, and low latency if the clients are smart.

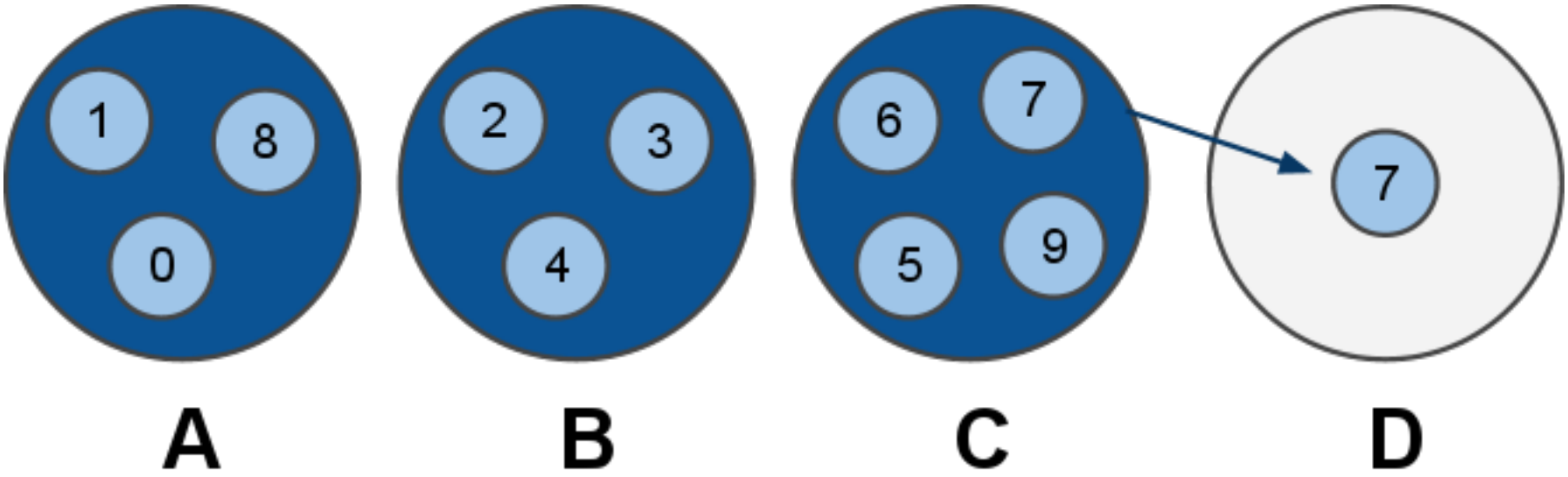
- Especially in large clusters where clients will try to have many persistent connections to multiple nodes, the Redis client object should be shared.

Re-sharding



- We are experiencing too much load. Let's add a new server.
- Node C marks his slot 7 as "MOVING to D"
- Every time C receives a request about slot 7, if the key is actually in C, it replies, otherwise it replies with -ASK D
- -ASK is like -MOVED but the difference is that the client **should retry against D only this query**, not next queries. That means: smart clients should not update internal state.

Re-sharding - moving data

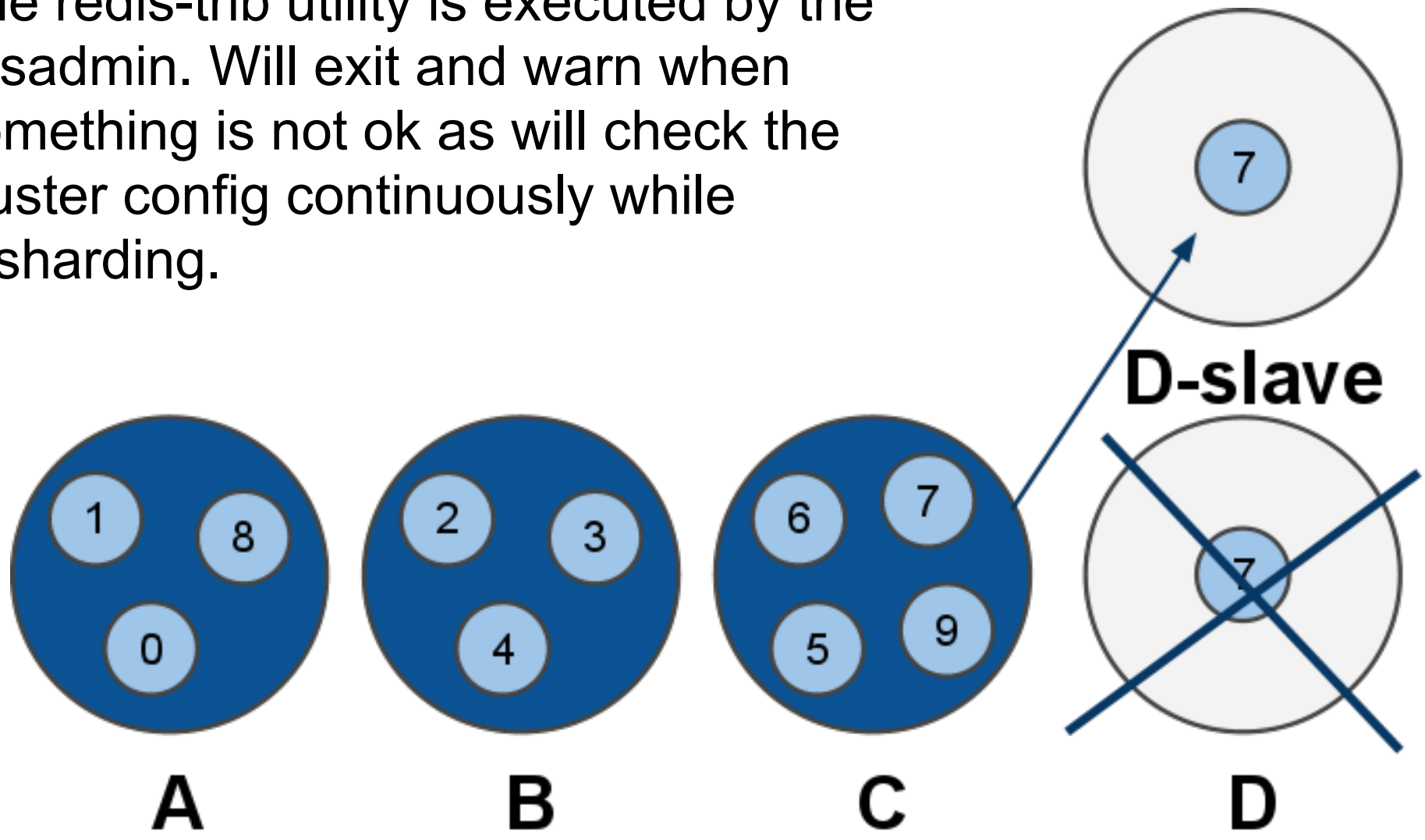


- All the new keys for slot 7 will be created / updated in D.
- All the old keys in C will be moved to D by redis-trib using the MIGRATE command.
- MIGRATE is an atomic command, it will transfer a key from C to D, and will remove the key in C when we get the OK from D. So no race is possible.
- p.s. MIGRATE is an exported command. Have fun...
- Open problem: ask C the next key in hash slot N, efficiently.

Re-sharding with failing nodes

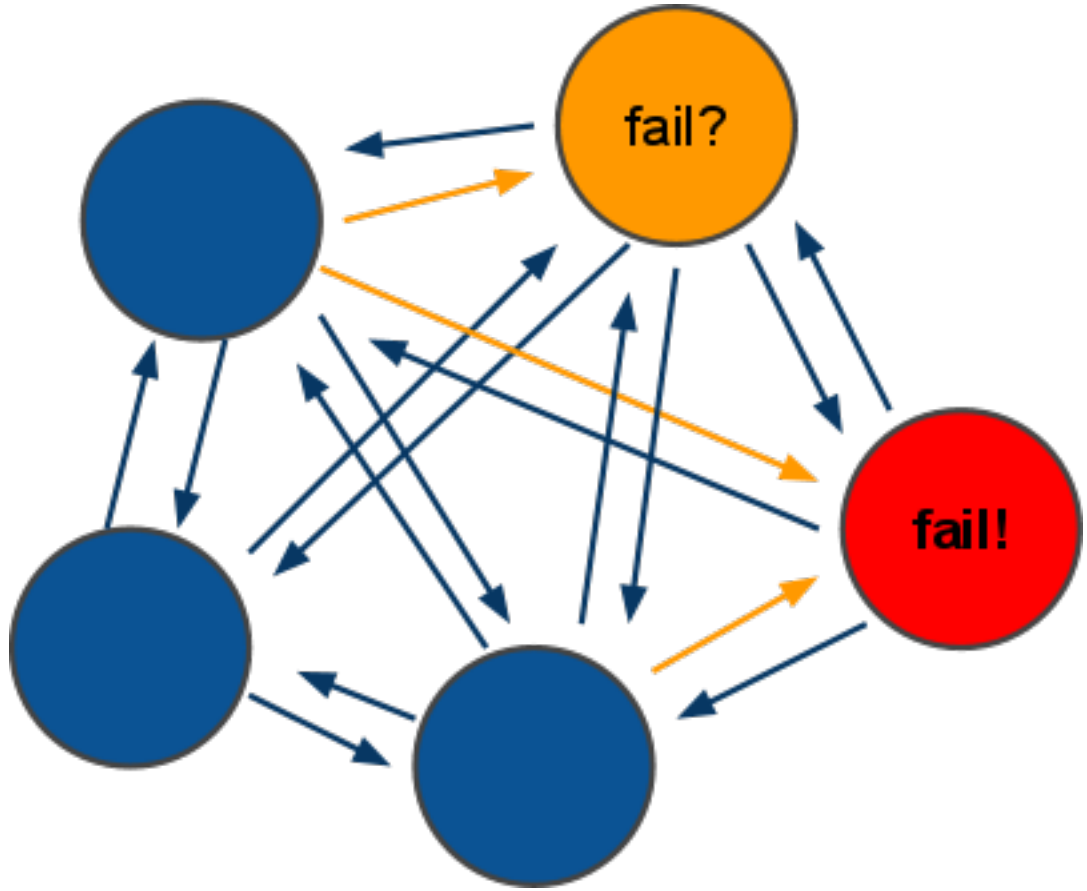
- Nodes can fail while resharding. It's slave promotion as usually.

- The redis-trib utility is executed by the sysadmin. Will exit and warn when something is not ok as will check the cluster config continuously while resharding.



Fault tolerance

- All nodes continuously ping other nodes...
- A node marks another node as *possibly* failing when there is a timeout longer than N seconds.
- Every PING and PONG packet contain a **gossip section:** information about other nodes idle times, from the point of view of the sending node.



Fault tolerance - failing nodes

- A guesses B is failing, as the latest PING request timed out. A will not take any action without any other hint.
- C sends a PONG to A, with the gossip section containing information about B: C also thinks B is failing.
- At this point A marks B as failed, and notifies the information to all the other nodes in the cluster, that will mark the node as failing.
- If B will ever return back, the first time he'll ping any node of the cluster, it will be notified to **shut down** ASAP, as intermitting clients are not good for the clients.
- **Only way to rejoin a Redis cluster after massive crash is: redis-trib by hand**.

Redis-trib - the Redis Cluster Manager

- It is used to setup a new cluster, once you start N blank nodes.
- it is used to check if the cluster is consistent. And to fix it if the cluster can't continue, as there are hash slots without a single node.
- It is used to add new nodes to the cluster, either as slaves of an already existing master node, or as blank nodes where we can re-shard a few hash slots to lower other nodes load.

It's more complex than this...

- there are many details that can't fit a 20 minutes presentation...
- Ping/Pong packets contain enough information for the cluster to restart after graceful stop. But the sysadmin can use CLUSTER MEET command to make sure nodes will engage if IP changed and so forth.
- Every node has a unique ID, and a cluster config file. Everytime the config changes the cluster config file is saved.
- The cluster config file can't be edited by humans.
- The node ID never changes for a given node.
- **Questions?**