ELASTICSEARCH

inside out

ELASTICSEARCH

- Building blocks for fulltext search
- Easy data aggregations
- Distributed
- Scalable (read/write)
- Fault-tolerant

VOCABULARY

SCALING

- Scaling = capacity change
- Vertical (scaling up/down)
- Horizontal (scaling out/in)

HORIZONTAL SCALING (SCALING UP)

- Keep single server
  - Add more CPU, more RAM
- This only works so far..
- Reverse operation:
  - scaling down

VERTICAL SCALING (SCALING OUT)

- Adding more servers

- .. but
  - More complicated
  - More overhead
  - More maintenance
  - More trouble
  - ..

- You want to scale up first, as far as possible
- Reverse operation: scaling in

SHARDING

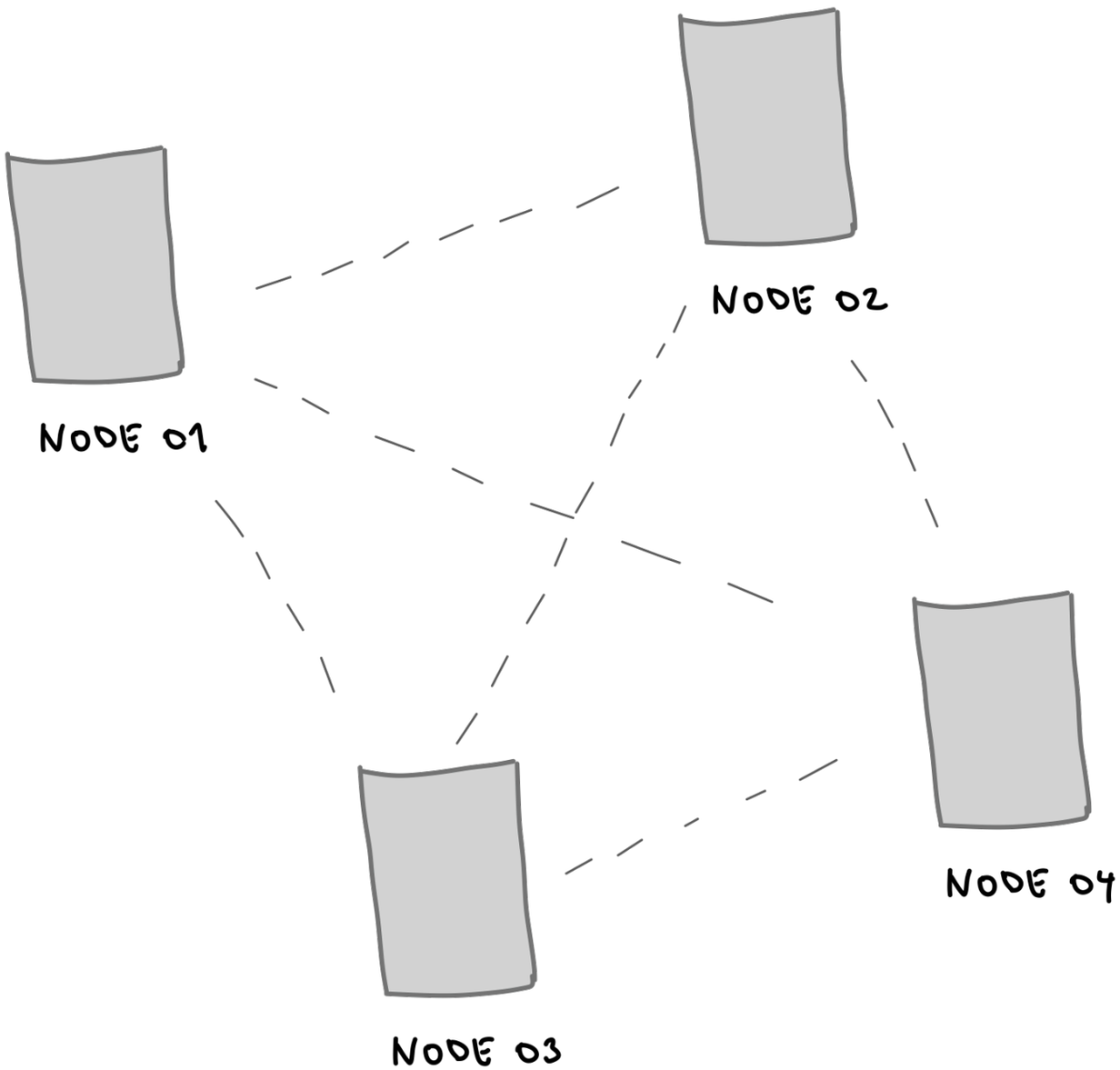
- Relatively simple scale-out technique
- Keep separate databases (they don't know about each other)
- Your application decides which **shard** (which database) will store the data
- Your application manages data distribution, e.g.
  - shard = record_id modulo num_shards
- Number of shards must be known in advance and fixed
- Adding a shard requires resharding data (moving data around)
- Problematic joins (cross-database joins?), requires data co-locality

ELASTICSEARCH

2 MODES OF OPERATION

- Single-node
- Cluster

CLUSTER (SIMPLIFIED)



INDEX

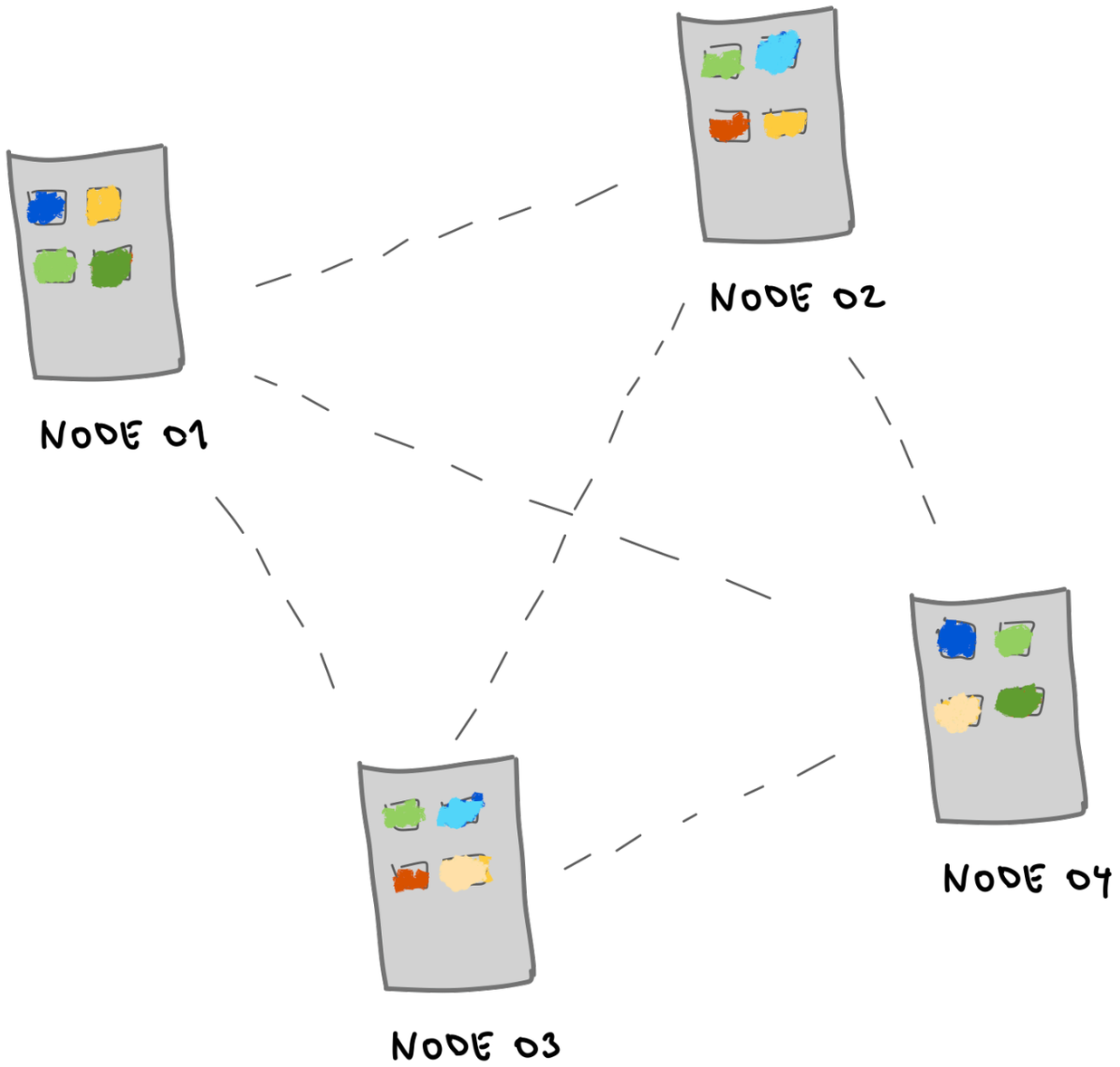
- 1 index = N shards
- 1 shard = M segments
- 1 segment = 1 Lucene index

- Primary
- N replicas, N >= 0

SHARD ALLOCATION / MANAGEMENT

- Handled by Elasticsearch (not your app)
- Allocation deciders (rules), e.g.:
  - primary should be on a different server than replica
  - in a different data center
  - in a different rack
  - each server should have roughly the same number of shards
  - disk space usage on each server should be < 90%

CLUSTER STATE (WHO HAS WHAT)



- idx01:
  - pri/1: node01
  - pri/2: node04
  - rep/1: node02
  - rep/2: node03
- idx02:
  - pri/1: node01
  - pri/2: node02
  - rep/1: node03
  - rep/2: node04

INDEX SETTING

```
curl -XPUT localhost:9200/my-index -d '{
  "settings": {
     "index.number_of_shards": 2,
     "index.number_of_replicas": 1
  }
}'
```

STATE INSPECTION

```
curl 'localhost:9200/_cat/indices?v'

health status index    uuid
pri rep docs.count docs.deleted store.size
pri.store.size

green  open   my-index 4B8NUyxKTraHknDImae5AA
2   1          0            0       920b
460b
```

STATE INSPECTION

```
curl 'localhost:9200/_cat/shards?v'

index    shard prirep state   docs store ip
node

my-index 1     p      STARTED    0  230b
172.18.0.5 es03

my-index 1     r      STARTED    0  230b
172.18.0.2 es02

my-index 0     p      STARTED    0  230b
172.18.0.6 es01

my-index 0     r      STARTED    0  230b
172.18.0.2 es02
```

FAULT TOLERANCE

- Primary goes down? Use replica.
- Data center (DC) goes down? Replicas exist in another DC
  - Thanks to allocation deciders
- Rack goes down? Replicas exist in another rack
  - Thanks to allocation deciders
- Must be somehow supported by your app
  - If a server is unreachable, connect to another

INDEX STATE

- Green
  - All good
- Yellow
  - Replicas are missing, if you lose primary = problem
- Red
  - Some (or all) shards are completely gone. Serving partial data (or none at all).

CLUSTER STATE

- Green
  - All indices green
- Yellow
  - At least one index is yellow, none is red
- Red
  - At least one index is red

FAULT TOLERANCE (AGAIN)

- Elasticsearch will correct the state if possible
  - Server gone? Allocate replicas to other servers
  - May not always be possible due to allocation rules

SCALING READS

- Node accepts read request (reads are from index)
- Checks cluster state - where are shards for this read/index?
- Sends requests to specific shards
  - This node now acts as a "coordinating" node
- After all nodes responded with partial results, "aggregate" the results and send response

AGGREGATING SEARCHES

- Searches in shards are separate
- Each hit (found doc) gets assigned a score
- Aggregation = sort partial shard results and select top N docs

AGGREGATING AGGREGATIONS

```
"aggregations": {
    "top_tags": {
        "terms": {
            "field": "tags"
        }
    }
}
```

AGGREGATING AGGREGATIONS

- Each shard runs the aggregation
  - returns partial results
- Coordinating node aggregates partial results
- Inaccurate results

AGGREGATIONS EXAMPLE

- terms aggregation, size: 3

- Shard 1:
  - A: 33
  - B: 21
  - C: 20
  - D: 19
- Shard 2:
  - B: 40
  - D: 30
  - A: 21
  - C: 6
- After agg:
  - B: 61
  - A: 54
  - D: 30
  - C: 20
- Real numbers:
  - B: 61
  - A: 54
  - D: 49
  - C: 26

AGGREGATIONS INACCURACIES

- Can get more accuracy with shard_size
  - Increases the "window" size of each shard
  - Accuracy vs. performance

SCALING WRITES

- Node receives write request
  - There must be a single primary shard which receives all writes
  - Which?
    - A special node type - master - for cluster management

MASTER NODE

- Coordinates cluster state
- Single node - decision maker - modifies and broadcasts cluster state
- 2-phase cluster state commit

MASTER ELECTION

- Other master-eligible nodes exists
- In case of master going down, cluster elects new master from master-eligible nodes
- No master
  - writes are failing
  - reads are ok

CLUSTER INSPECTION

```
curl 'localhost:9200/_cat/nodes?v'

ip         heap.percent ram.percent cpu load_1m load_5m load_15m node.role master name

172.18.0.7           20          93   0    0.04    0.11     0.09 ilm       *      esmaster02

172.18.0.4           22          93   0    0.04    0.11     0.09 ilm       -      esmaster01

172.18.0.5           26          93   0    0.04    0.11     0.09 dil       -      es03

172.18.0.3           27          93   0    0.04    0.11     0.09 ilm       -      esmaster03

172.18.0.2           29          93   0    0.04    0.11     0.09 dil       -      es02

172.18.0.6           23          93   0    0.04    0.11     0.09 dil       -      es01
```

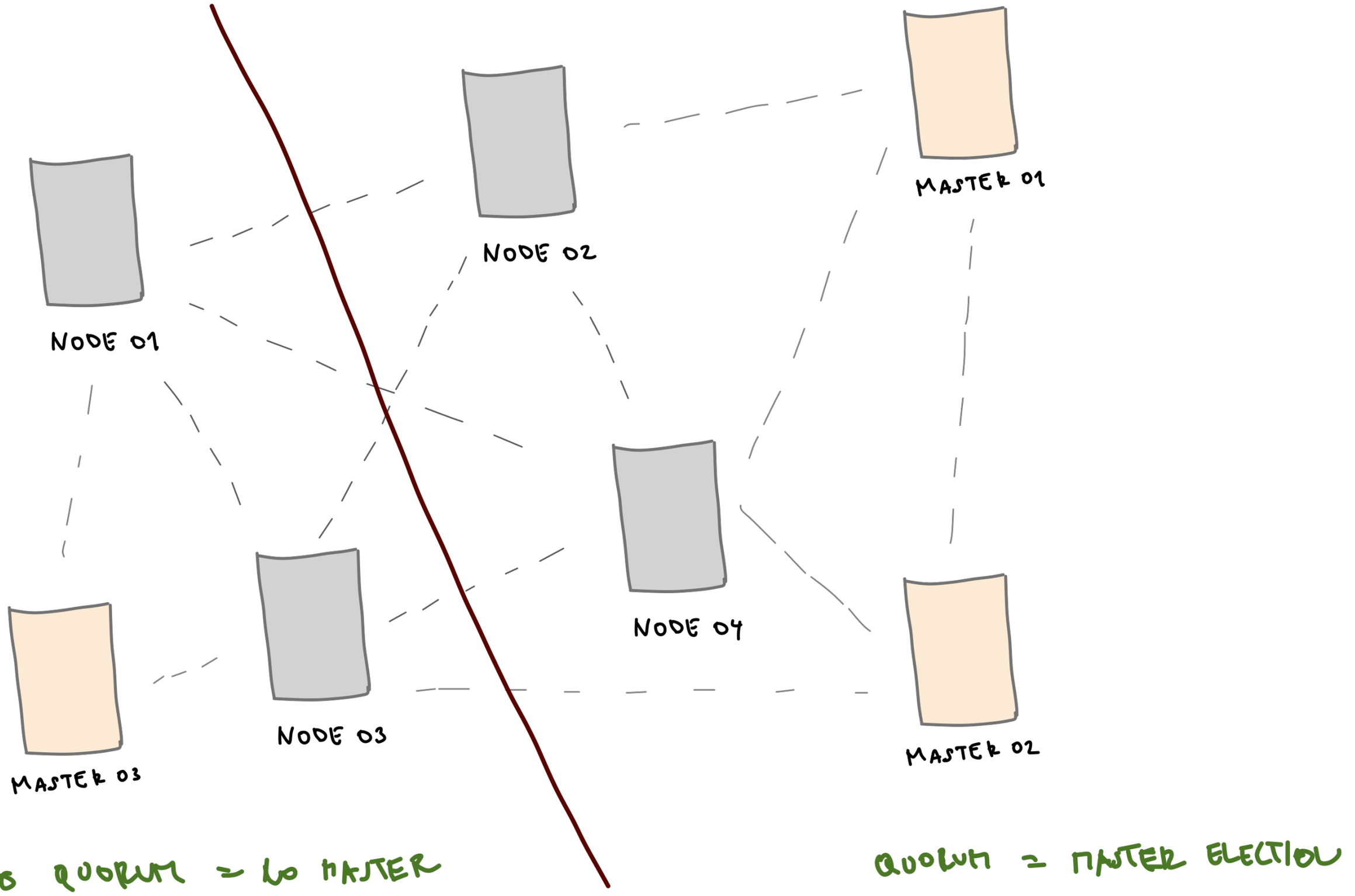
DISTRIBUTED FAILURES

- Networks do fail sometimes
  - network partitions
- You want to avoid split-brain situation
  - some nodes in cluster lose connectivity to master, and elect a new master
  - cluster is now split to 2 parts -> data inconsistency

QUORUM OF MASTERS

- Master-eligible nodes can elect a master only if at least N nodes participate in election (quorum)
- Robust case
  - master-eligible nodes: 3
  - quorum: 2

NETWORK PARTITION



BASIC TENET OF A DISTRIBUTED SYSTEM

- A distributed system is (almost) never 100% down
- Your application must deal with it
  - E.g. by retrying query on a different host

BASIC PROCESSES

- Segment = Lucene data = immutable
  - update = delete + create
- Segment merges
  - merge data, avoid deleted docs
    - CPU-intensive (compression/decompression)
- Flush new segment to disk
  - IO-intensive (writing new blocks to disk)
- Indexing = merge, flush, merge, flush, merge, flush...

BASIC PROCESSES

- Query
  - CPU??
  - IO??
- Snapshots
- Rollups
- Refresh
- ...

THREAD POOLS

- Resource contention control
- Each node has several threads (configurable, dynamic defaults) for main operations
- Thread pool size = amount of operation concurrency = amount of resource utilization
- Queues = tasks wait in queue for available threads
- Rejections = Queue full

THREAD POOLS

```
curl -s 'localhost:9200/_cat/thread_pool?v' | head
node_name  name                        active queue rejected
esmaster02 analyze                          0     0        0
esmaster02 ccr                              0     0        0
esmaster02 data_frame_indexing              0     0        0
esmaster02 fetch_shard_started              0     0        0
esmaster02 fetch_shard_store                0     0        0
esmaster02 flush                            0     0        0
esmaster02 force_merge                      0     0        0
esmaster02 generic                          0     0        0
esmaster02 get                              0     0        0
```

WHAT YOU SHOULD NOW

- Understand types of scaling and their trade-offs
- Understand principles of sharding
- Understand the role of a master node in cluster and its impact on writes
- Understand basic principles of fault tolerance in Elasticsearch, understand what happens in case a node disconnects from the cluster.