DESIGNING DATA-INTENSIVE APPLICATIONS

CHAPTER 1

WHAT ARE THE MAIN TOPICS?

STORAGE

DATA PROCESSING

BIG DATA

SHARDING & REPLICATION

EVENTUAL CONSISTENCY

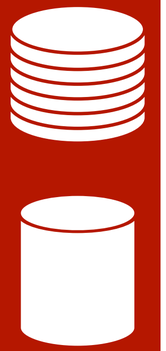
SQL/NOSQL

MAPREDUCE

ACID

CAP

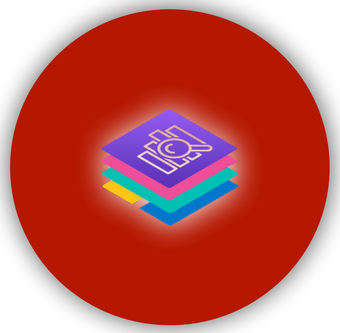
WEB SCALE



WHY WE HAVE TOO MANY TYPES/TOOLS?

DATABASES

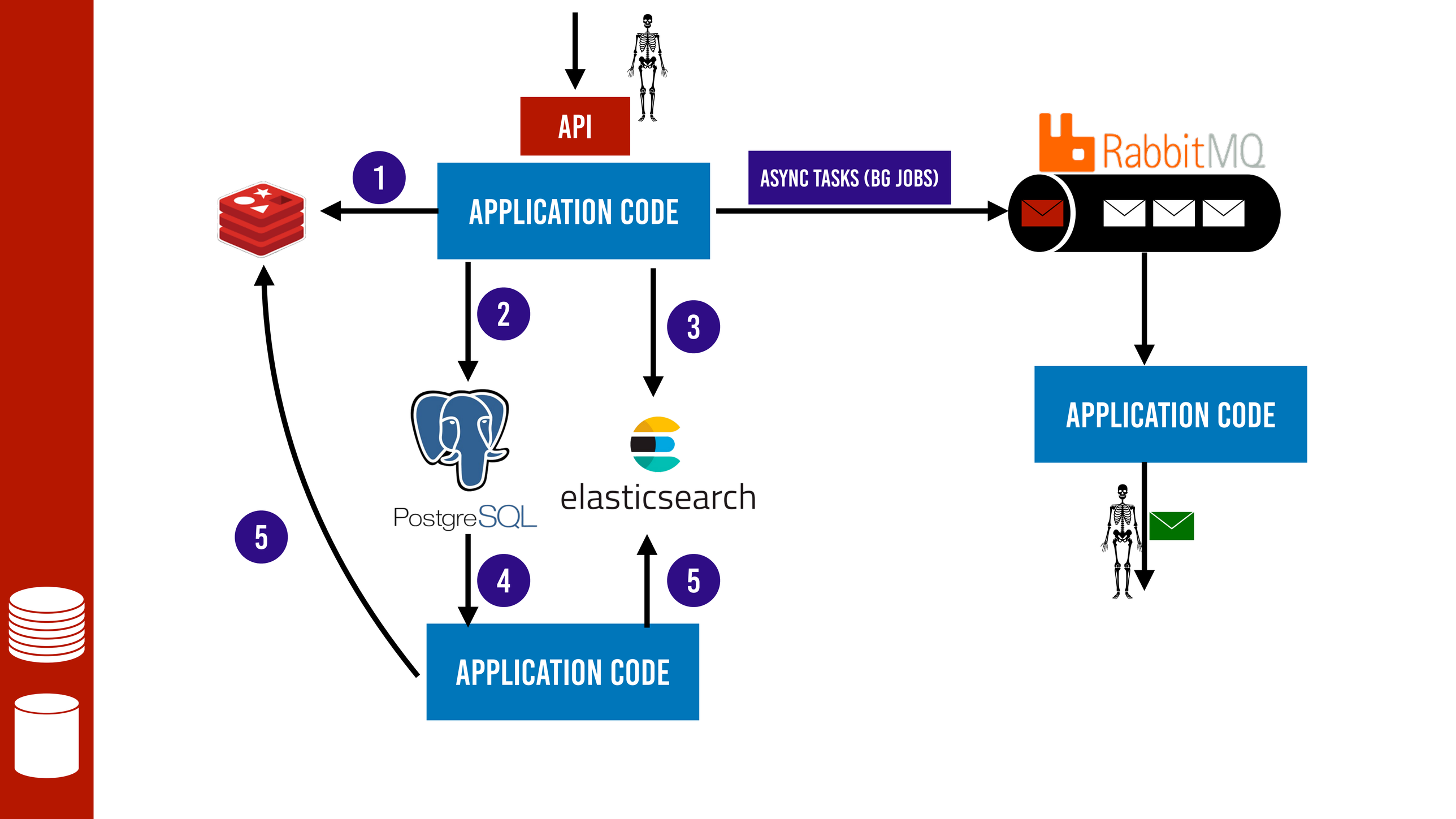
CACHES



SEARCH
INDEXES

STREAM
PROCESSING

BATCH
PROCESSING

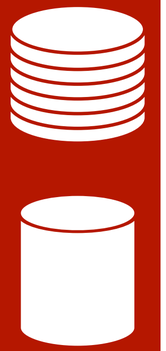

API
APPLICATION CODE
1
ASYNC TASKS (BG JOBS)
RabbitMQ
2
3
PostgreSQL
elasticsearch
APPLICATION CODE
5
4
5
APPLICATION CODE

WHAT ARE WE TRYING TO ACHIEVE WHEN WE BUILD A SYSTEM?

RELIABLE

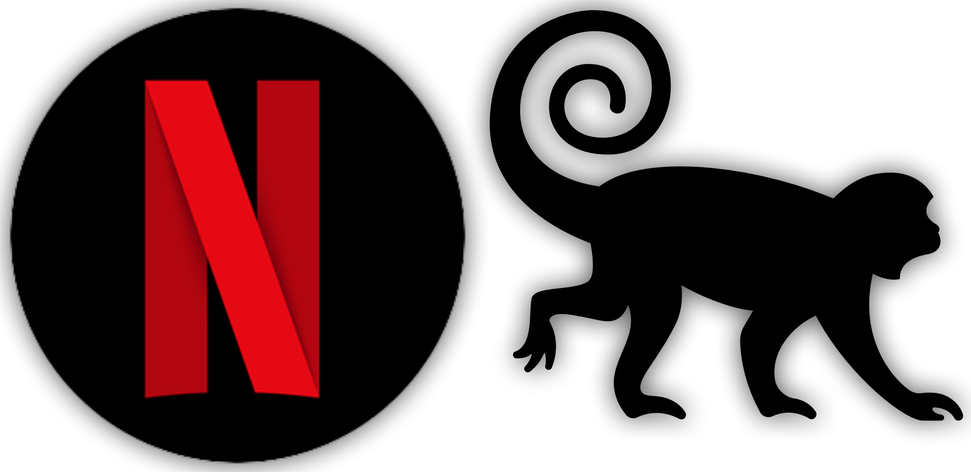
SCALABLE

MAINTAINABLE



RELIABLITY

- CONTINUE WORKING CORRECTLY WHEN THINGS GOES WRONG
- FAULT TOLERANT - RESILIENT
- FAULT NOT FAILURE

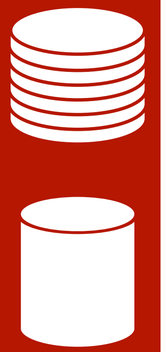


FAULTS TYPES

HARDWARE

SOFTWARE

HUMAN



SCALABILITY

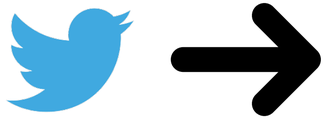
SYSTEM ABILITY TO WORK RELIABLY UNDER LOAD

WHAT IS LOAD?!

POST TWEET (AVG 4.6K) (PEAK +12K) FAN-OUT?!

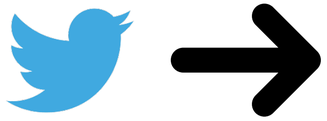
TIMELINE HOMEPAGE 300K REQUEST/SEC

HOW TWITTER FIXED THE SCALABILITY ISSUE?



GLOBAL COLLECTION OF TWEETS

```
SELECT tweets.*, users.* FROM tweets
  JOIN users   ON tweets.sender_id    = users.id
  JOIN follows ON follows.followee_id = users.id
  WHERE follows.follower_id = current_user
```



GLOBAL COLLECTION OF TWEETS

U1 TWEETS

U2 TWEETS

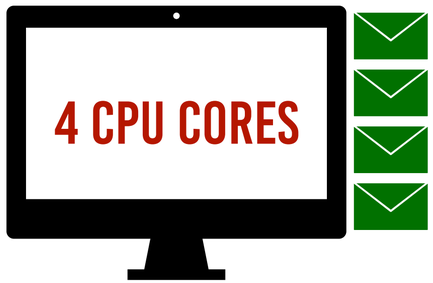
U3 TWEETS

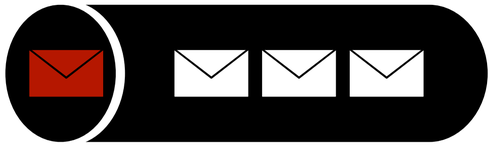
PERFORMANCE

SYSTEM RESOURCES VS INCREASING LOAD

HADOOP THROUGHPUT

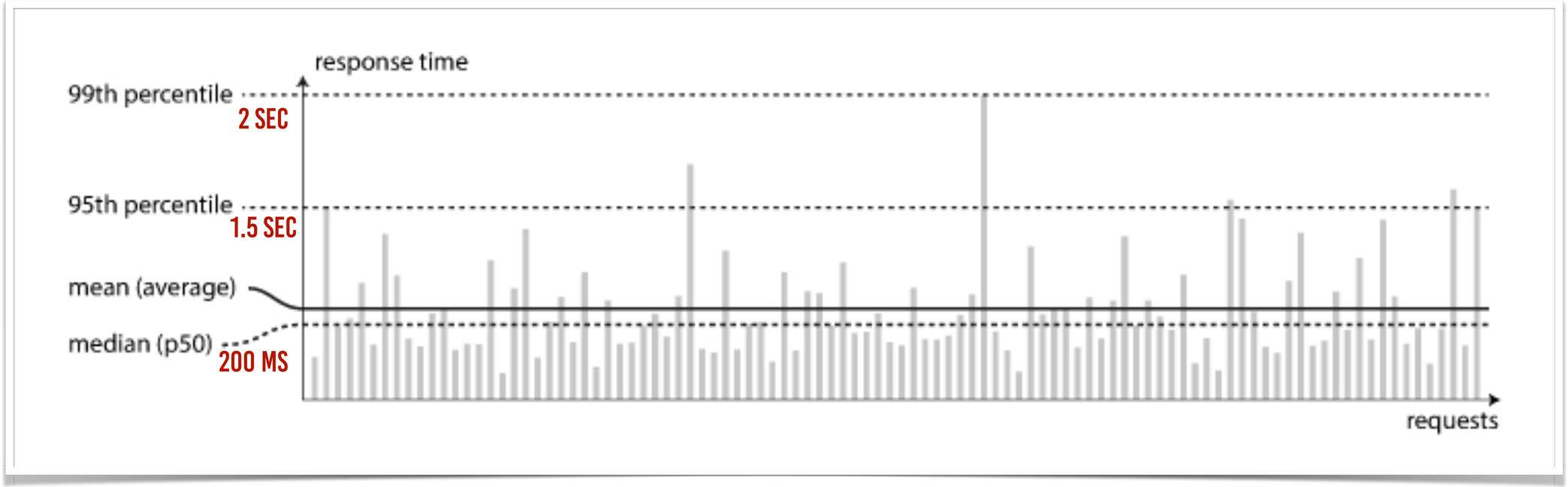
RESPONSE TIME VS LATENCY VS SERVICE TIME





HEAD-OF-LINE BLOCKING

HOW CAN WE MEASURE PERFORMANCE BY RESPONSE TIME?



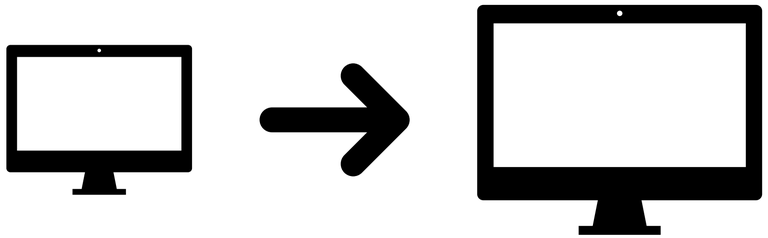
amazon

100 MS RESPONSE TIME

1% DECREASE IN SALE

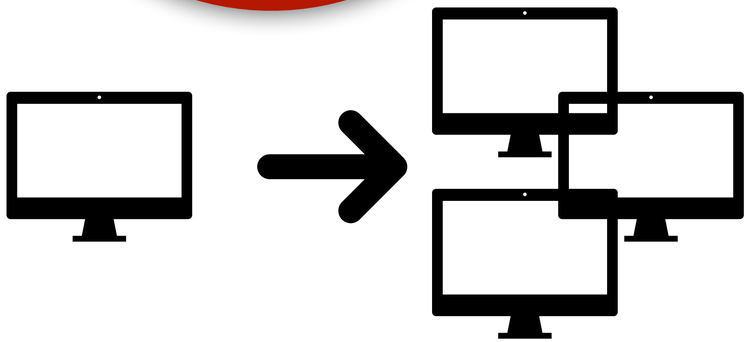
SCALING-UP

VERTICAL



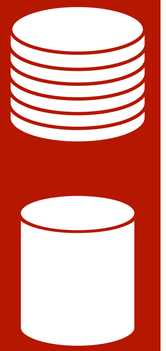
SCALING-OUT

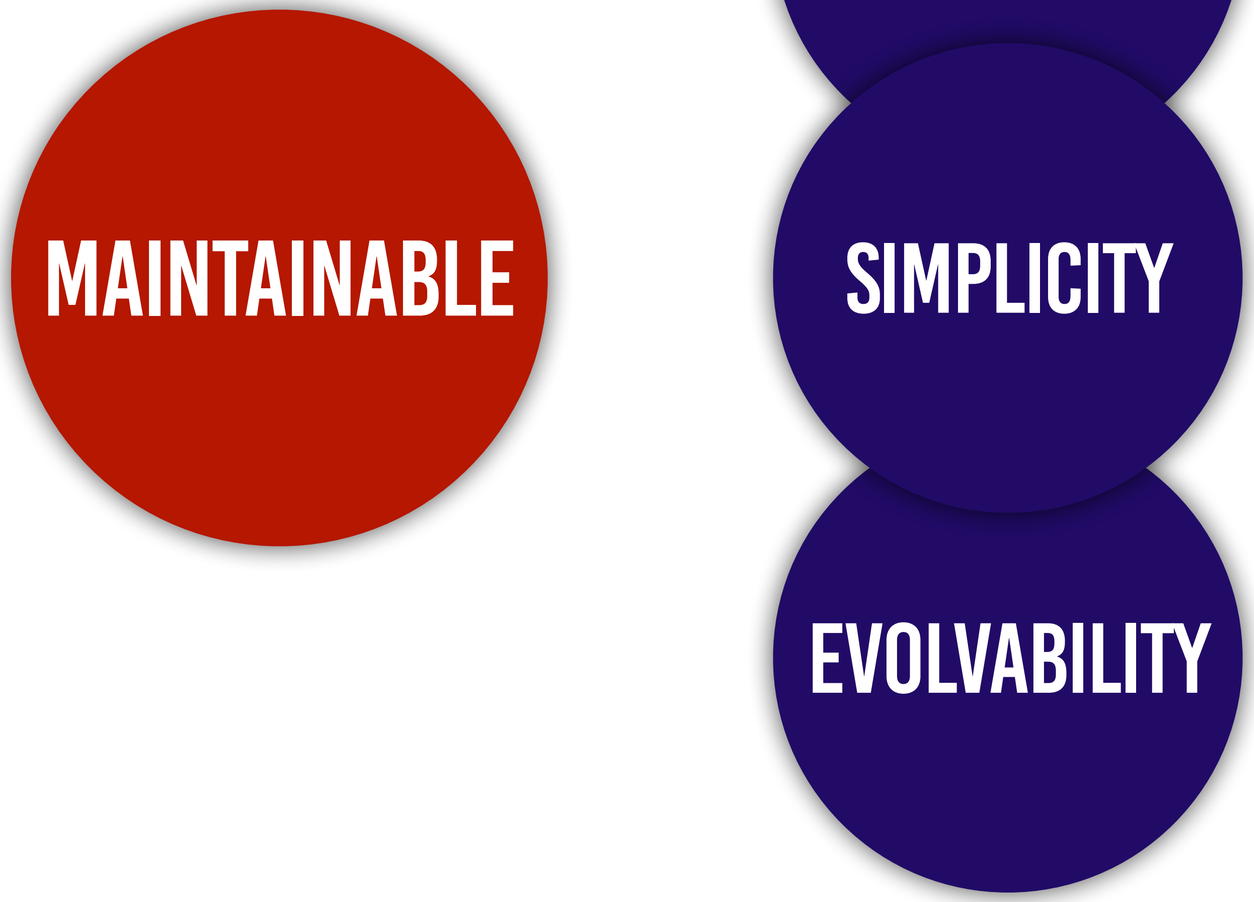
HORIZONTAL



DO WE HAVE MAGIC SCALING CAUSE?

- THE VOLUME OF READS
- RESPONSE TIME REQUIREMENTS
- THE VOLUME OF WRITES
- THE ACCESS PATTERNS
- THE VOLUME OF DATA TO STORE
- THE COMPLEXITY OF THE DATA




MAINTAINABLE
OPERABILITY
SIMPLICITY
EVOLVABILITY