System Design

1) Characteristics → Data Intensive Applications

2) Principles
   - Reliable
   - Scalable
   - Maintainable

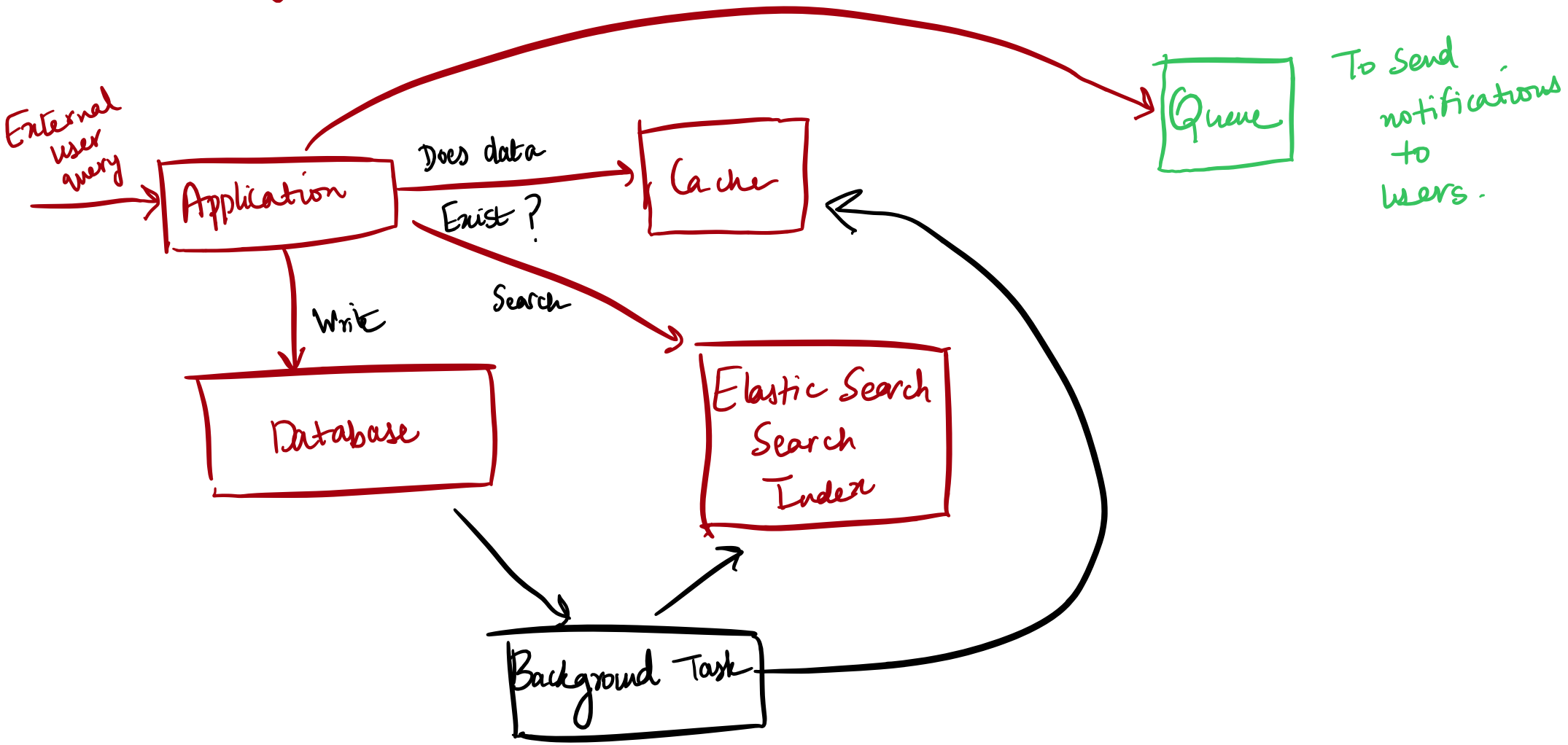
Data Intensive

- What is the alternative type → compute Intensive
- What is the bottleneck?
  - large amounts of Data → Facebook

Building Blocks

1) Databases → storage & retrieval (optimizations considerations)
2) Cache
3) Load Balancers
4) Message Transmission → to process async

Stitching the above components



What are the goals of System Design?

(i) Robust

(ii) Trustable

(iii) Data Integrity

1) Reliable → What?

- System performs correctly in the face of Faults
  - Corrected output in expected time frame
  - System prevents unauthorized access
- Tolerate certain Faults

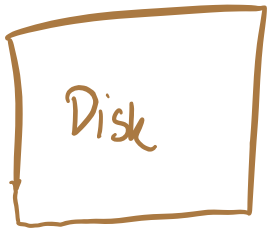
Fault vs **Failure**

- Fault → Single component
- Failure → Stops Functioning completely

→ Types of Faults
- Software
- hardware → usually independent → events are unrelated
- human

MTTF ⟶ mean time to failure

MTTR ⟶ mean time to recover.



30 years

MTTF

lets say a system uses 10 disks

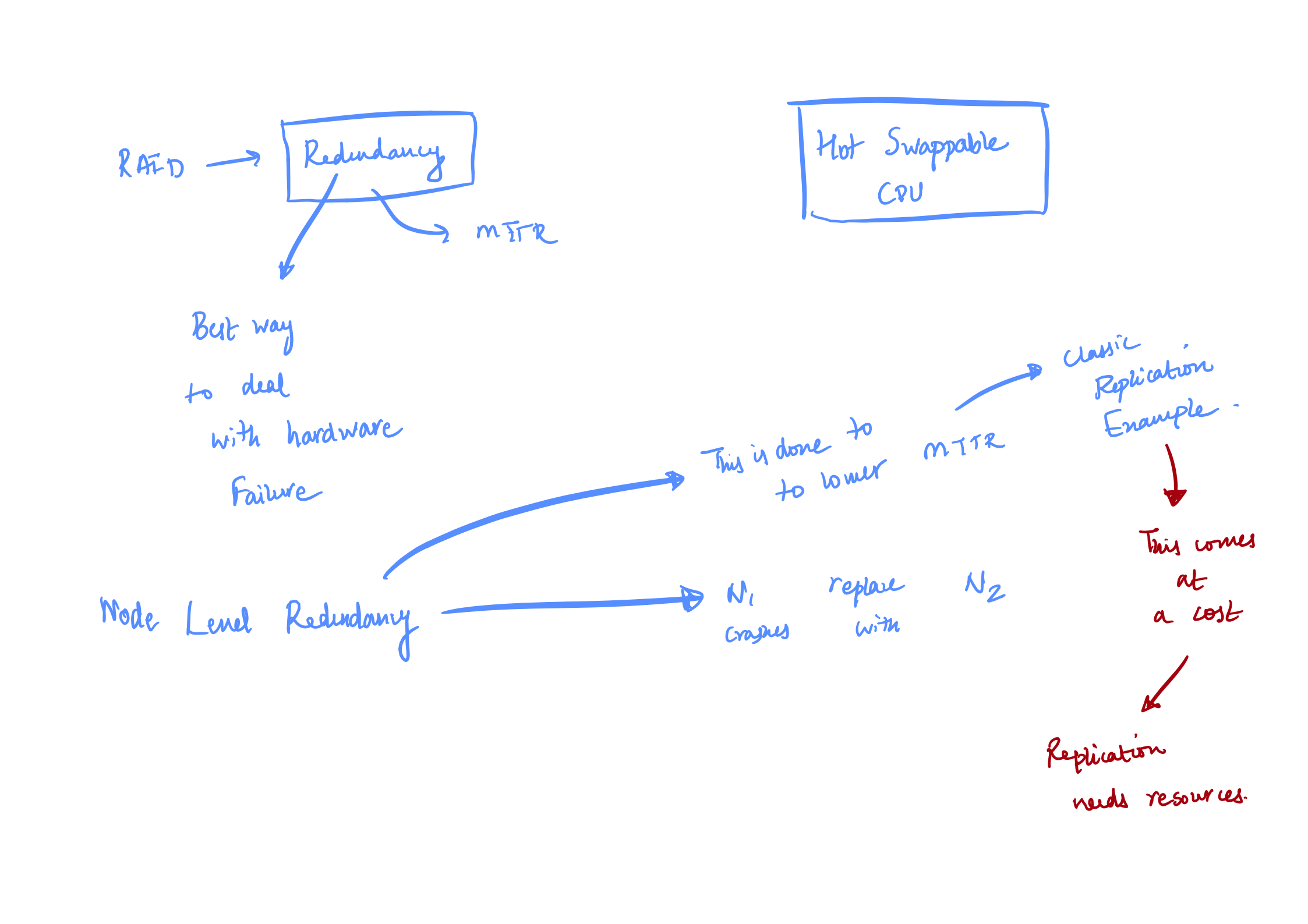
probability of atleast 1 single disk failure

$= 1 - P(\text{None})$

$= 1 - P(D_1)P(D_2)P(D_3)$

D_1 → disk 1 Failure

we multiply because they are independent events


RAID
Redundancy
MTTR
Hot Swappable CPU
Best way to deal with hardware Failure
Node Level Redundancy
This is done to to lower MTTR
Classic Replication Example.
This comes at a cost
Replication needs resources.
N_1 crashes
replace with
N_2

Scalability

- how does the system perform when you increase the load?
- Is your architecture designed to be able increase the resources and handle more load?

Load Parameters

- Metrics to quantify the load
  - RPS (For a web server)
    - webServers also use QPS, WPS
  - Read to Write Ratio (For a Database)
  - Cache — Hits is a good param

Exercise

Twitter
- Post a thread (write)
- Fetch their feed (read)

Approach

Tradeoffs → Design around Read Heavy or Write Heavy.

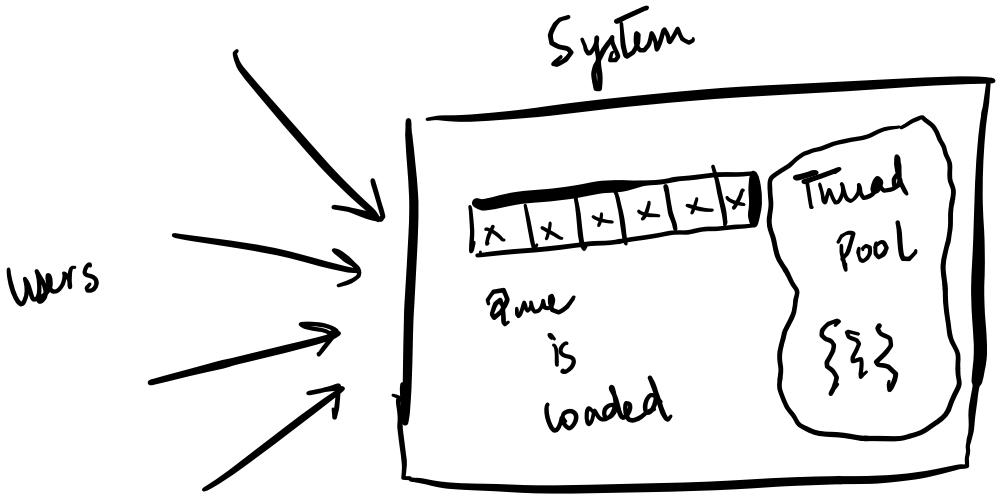
Approach 1

Approach 2

Performance → How do you evaluate ?

1) Throughput → Batch Processing System → How many jobs does a system complete in a time frame?

2) Response Time → Any online system → time b/w request & response. → A very important metric



response

Time can be multiple parts that can go wrong.

If Queue is loaded → RT will be affected

Designs can look like:

1) For high throughput

2) For low response time

Evaluating Performance Parameters

Response Time

(i) Distributions

Percentiles

P50 P90 P99

Px

x % of requests took less than some time

Throughput

(i) Average

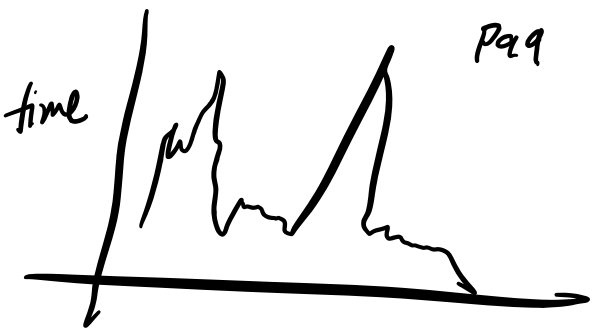
basically how many requests you process over fixed time

This can also be over intervals

Look at grafana metrics

P99

99% of requests took less than 5 milliseconds



usually a minute

Cadence is a tradeoff because of time period can be changed

The timeframe can be varied

Approach for coping with Load

1) Vertical Scaling → There is a certain hardware limit you will hit. → Moore's law

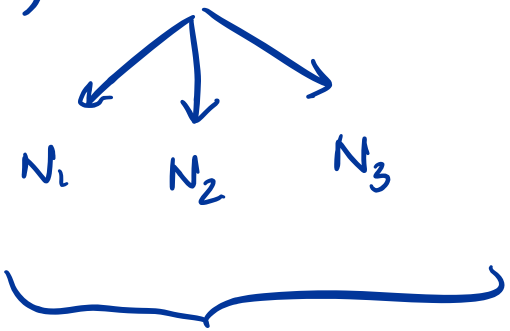
2) Horizontal Scaling

- Just add more nodes. →  Throw more money at a problem.
- There is a potential problem here → Stateful Service vs Stateless Service (C* vs random service)

Stateless

1) No state

2) Auto Scale

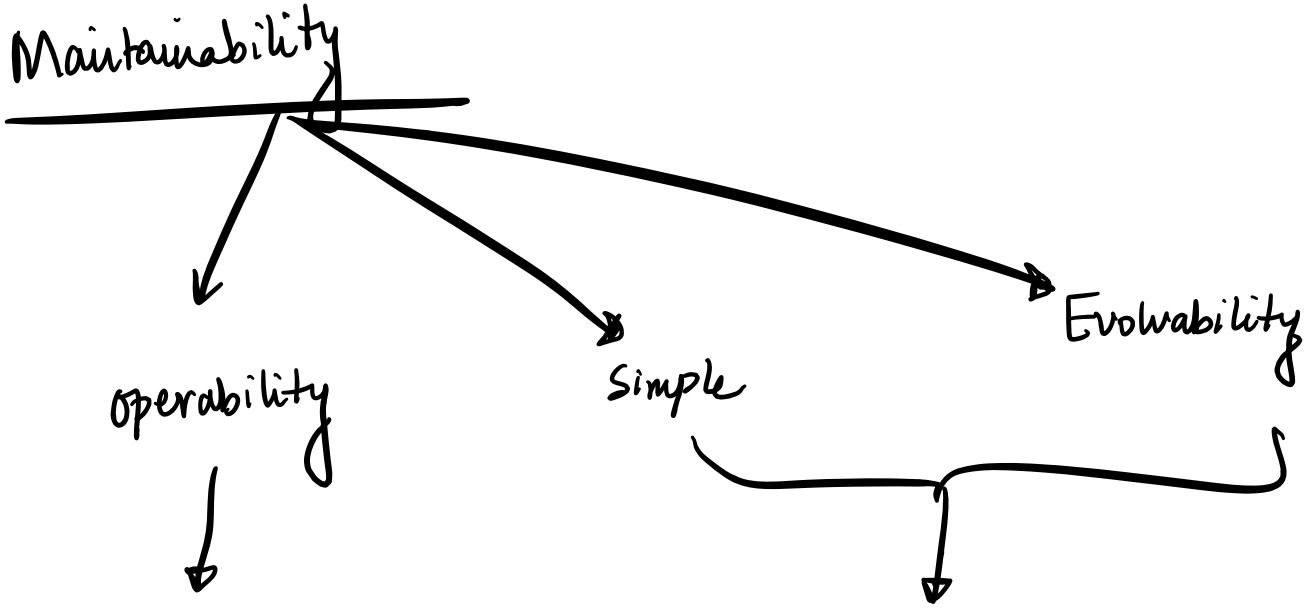


Amazon ASG - Auto Scaling Group

Stateful

1) State needs to maintained (DBs)
   - Example: C*
   - Cache
   - Zookeeper
     - locking
     - service

2) Scaling needs processes
   - C* scale incidents



1) UI controls

2) Documentation

Similar concepts lead to this

(i) Good Design Patterns

(ii) modular / Extensible components.