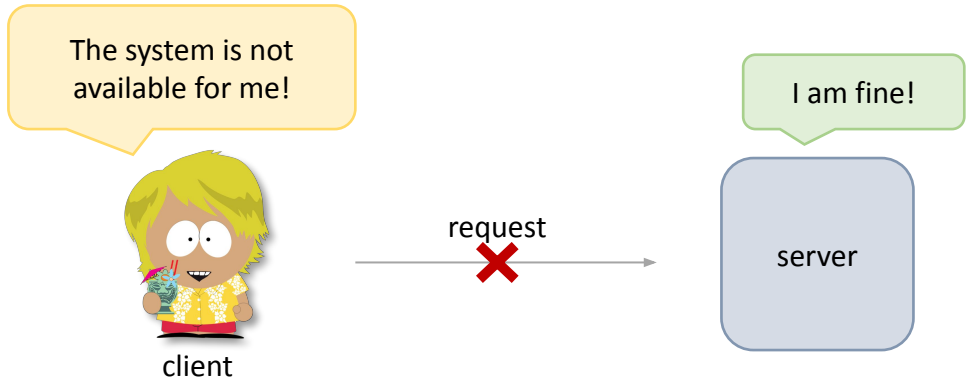
Non-functional requirements – High availability

count-based availability

time-based availability

Success ratio of requests. = availability = System **uptime**, the percentage of time the system has been working and available.

1 request out of 100 fails. = 99% = The system was unavailable about 3.65 days a year.



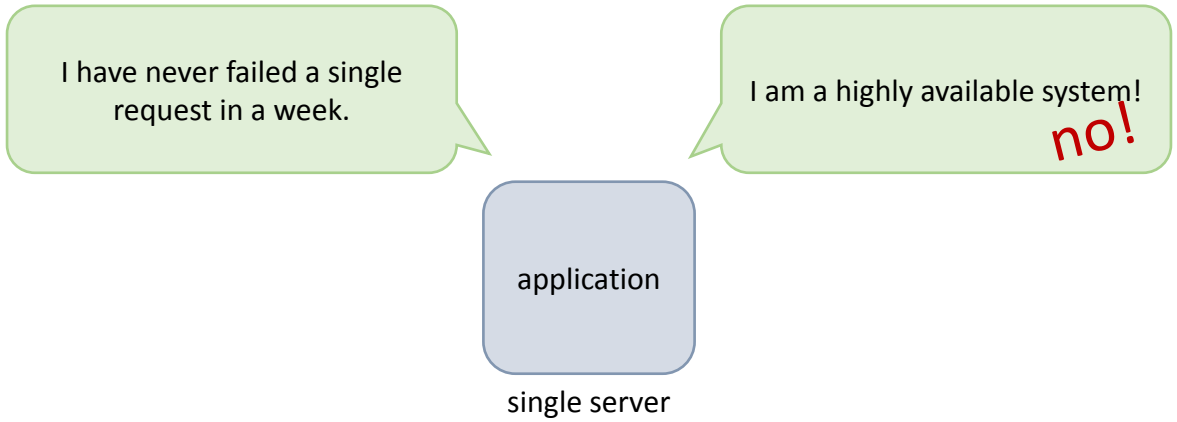
Non-functional requirements – High availability

What is a highly available system?

98% 99% 100% ← very complex and expensive and almost never justified for distributed systems

none of the above

it is not about a number
it is about **architecture** and **process**



Non-functional requirements – High availability

design principles behind high availability

- build redundancy to eliminate single points of failure

  (regions, availability zones, fallback, data replication, high availability pair, …)

- switch from one server to another without losing data

  (DNS, load balancing, reverse proxy, API gateway, peer discovery, service discovery, …)

- protect the system from atypical client behavior

  (load shedding, rate limiting, shuffle sharding, cell-based architecture , …)

- protect the system from failures and performance degradation of its dependencies

  (timeouts, circuit breaker, bulkhead, retries, idempotency, …)

- detect failures as they occur

  (monitoring,
  …)

Non-functional requirements – High availability

processes behind high availability

- **change management**

  all code and configuration changes are reviewed and approved

- **deployment**

  deploy changes to a production environment frequently, quickly, safely;
  automated rollback

- **disaster recovery**

  recover system quickly in the event of a disaster;
  regularly test failover to disaster recovery

- **operational readiness review**

  evaluate system's operational state and identify gaps in operations;
  define actions to remediate risks

- **team culture**

  good team culture promotes process discipline

- **QA**

  regularly exercise tests to validate that newly introduced changes meet functional and non-functional requirements

- **capacity planning**

  monitor system utilization and add resources to meet growing demand

- **root cause analysis**

  establish the root cause of the failure and identify preventive measures

- **game day**

  simulate a failure or event and test system and team responses

Non-functional requirements – High availability

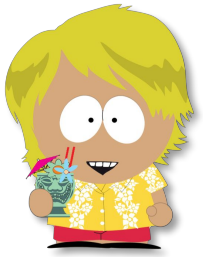
My system is highly available. It guarantees a monthly uptime percentage of 99.99%

service-level objective (**SLO**)

software engineer

If you experience lower availability, I will refund you.

Service Level Agreement (**SLA**)
…
…
…



client