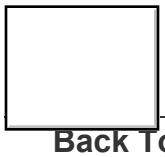


Back To Course Home

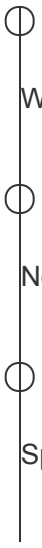
Grokking Modern System Design Interview for Engineers & Managers

0% completed

System Design Interviews

Introduction

Abstractions



Why Are Abstractions Important?

Network Abstractions: Remote Procedure Calls

Spectrum of Consistency Models

The Spectrum of Failure Models

Non-functional System Characteristics

Back-of-the-envelope Calculations

- [ ]

Building Blocks

- [ ]

Domain Name System

- [ ]

Load Balancers

- [ ]

Databases

- [ ]

Key-value Store

- [ ]

Content Delivery Network (CDN)

- [ ]

Sequencer

- [ ]

Distributed Monitoring

- [ ]

Monitor Server-side Errors

- [ ]

Monitor Client-side Errors

- [ ]

Distributed Cache

- [ ]

Distributed Messaging Queue

- [ ]

Pub-sub

- [ ]

Rate Limiter

- [ ]

Blob Store

- [ ]

Distributed Search

- [ ]

Distributed Logging

- [ ]

Distributed Task Scheduler

- [ ]

Sharded Counters

- [ ]

Concluding the Building Blocks Discussion

- [ ]

Design YouTube

- [ ]

Design Quora

- [ ]

Design Google Maps

- [ ]

Design a Proximity Service / Yelp

- [ ]

Design Uber

- [ ]

Design Twitter

- [ ]

Design Newsfeed System

- [ ]

Design Instagram

- [ ]

Design a URL Shortening Service / TinyURL

- [ ]

Design a Web Crawler

- [ ]

Design WhatsApp

- [ ]

Design Typeahead Suggestion

- [ ]

Design a Collaborative Document Editing Service / Google Docs

- [ ]

Spectacular Failures

- [ ]

Concluding Remarks

- [ ]

Course Certificate

Mark Course as Completed

The Spectrum of Failure Models

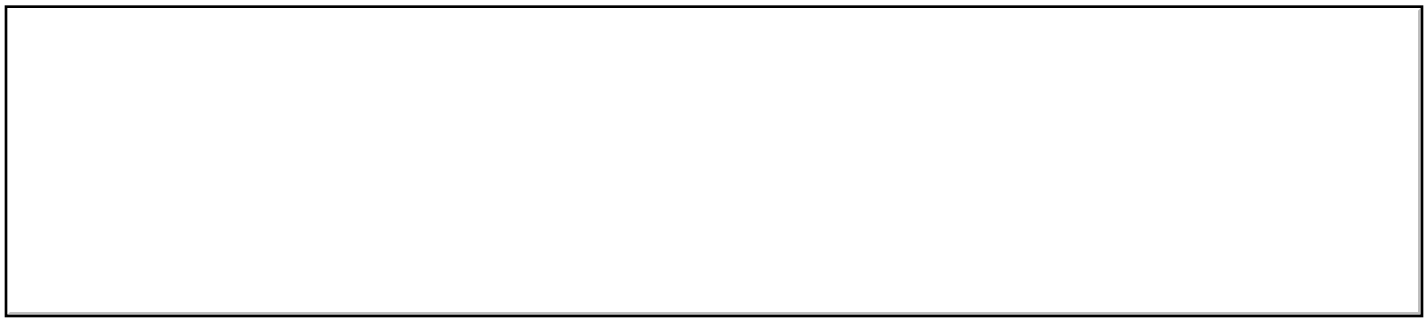
Learn about failures in distributed systems and the complexity of dealing with them.

We'll cover the following

Failures are obvious in the world of distributed systems and can appear in various ways. They might come and go, or persist for a long period.

Failure models provide us a framework to reason about the impact of failures and possible ways to deal with them.

Here is an illustration that presents a spectrum of different failure models:



This is a spectrum of failure models. The difficulty level when dealing with a failure increases as we move to the right

Fail-stop#

In this type of failure, a node in the distributed system halts permanently. However, the other nodes can still detect that node by communicating with it.

From the perspective of someone who builds distributed systems, fail-stop failures are the simplest and the most convenient.

Crash#

In this type of failure, a node in the distributed system halts silently, and the other nodes can't detect that the node has stopped working.

Omission failures#

In **omission failures**, the node fails to send or receive messages. There are two types of omission failures. If the node fails to respond to the incoming request, it's said to be a **send omission failure**. If the node fails to receive the request and thus can't acknowledge it, it's said to be a **receive omission failure**.

Temporal failures#

In **temporal failures**, the node generates correct results, but is too late to be useful. This failure could be due to bad algorithms, a bad design strategy, or a loss of synchronization between the processor clocks.

Byzantine failures#

In **Byzantine failures**, the node exhibits random behavior like transmitting arbitrary messages at arbitrary times, producing wrong results, or stopping midway. This mostly happens due to an attack by a malicious entity or a software bug. A byzantine failure is the most challenging type of failure to deal with.

Spectrum of Consistency Models

Next

Availability

Mark as Completed

Report an Issue