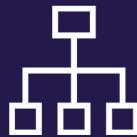


FREE

SYSTEM DESIGN

THE BIG ARCHIVE



MAY-17-2022

System Design

What are database isolation levels? What are they used for?	4
What is IaaS/PaaS/SaaS?	6
Most popular programming languages	7
What is the future of online payments?	9
What is SSO (Single Sign-On)?	11
How to store passwords safely in the database?	13
How does HTTPS work?	16
How to learn design patterns?	18
A visual guide on how to choose the right Database	20
Do you know how to generate globally unique IDs?	22
How does Twitter work?	24
What is the difference between Process and Thread?	26
Interview Question: design Google Docs	28
Deployment strategies	30
Flowchart of how slack decides to send a notification	32
How does Amazon build and operate the software?	33
How to design a secure web API access for your website?	35
How do microservices collaborate and interact with each other?	38
What are the differences between Virtualization (VMware) and Containerization (Docker)?	40
Which cloud provider should be used when building a big data solution?	42
How to avoid crawling duplicate URLs at Google scale?	44
Why is a solid-state drive (SSD) fast?	47
Handling a large-scale outage	49
AWS Lambda behind the scenes	51

HTTP 1.0 > HTTP 1.1 > HTTP 2.0 > HTTP 3.0 (QUIC).	53
How to scale a website to support millions of users?	55
DevOps Books	58
Why is Kafka fast?	60
SOAP vs REST vs GraphQL vs RPC.	62
How do modern browsers work?	63
Redis vs Memcached	64
Optimistic locking	65
Tradeoff between latency and consistency	67
Cache miss attack	68
How to diagnose a mysterious process that's taking too much CPU, memory, IO, etc?	70
What are the top cache strategies?	71
Upload large files	74
Why is Redis so Fast?	76
SWIFT payment network	77
At-most once, at-least once, and exactly once	80
Vertical partitioning and Horizontal partitioning	82
CDN	84
Erasure coding	87
Foreign exchange in payment	89
Block storage, file storage and object storage	94
Block storage, file storage and object storage	95
Domain Name System (DNS) lookup	97
What happens when you type a URL into your browser?	99
AI Coding engine	101
Read replica pattern	103

Read replica pattern	105
Email receiving flow	107
Email sending flow	109
Interview Question: Design Gmail	111
Map rendering	113
Interview Question: Design Google Maps	115
Pull vs push models	117
Money movement	119
Reconciliation	122
Which database shall I use for the metrics collecting system?	126
Metrics monitoring and altering system	129
Reconciliation	131
Big data papers	134
Avoid double charge	136
Payment security	138
System Design Interview Tip	139
Big data evolvement	140
Quadtree	142
How do we find nearby restaurants on Yelp?	144
How does a modern stock exchange achieve microsecond latency?	147
Match buy and sell orders	149
Stock exchange design	151
Design a payment system	153
Design a flash sale system	155
Back-of-the-envelope estimation	157

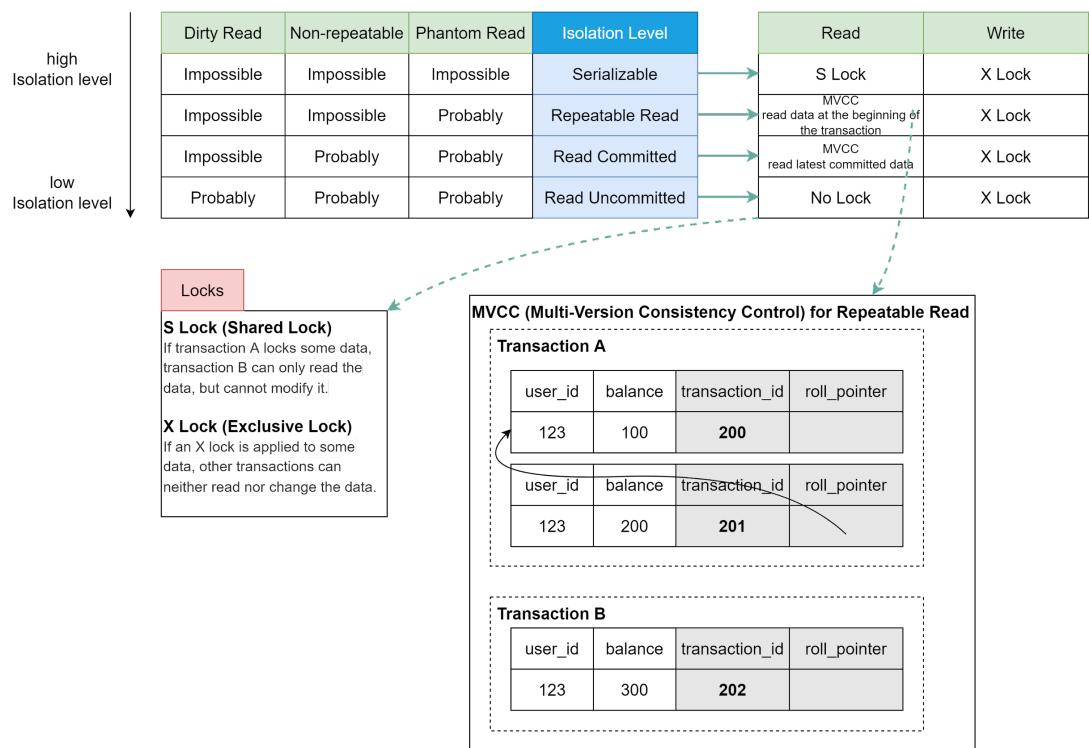
What are database isolation levels? What are they used for?

Database isolation allows a transaction to execute as if there are no other concurrently running transactions.

The diagram below illustrates four isolation levels.

Database Isolation Level Illustrated

 blog.bytebytego.com



- ◆ **Serializable**: This is the highest isolation level. Concurrent transactions are guaranteed to be executed in sequence.
- ◆ **Repeatable Read**: Data read during the transaction stays the same as the transaction starts.
- ◆ **Read Committed**: Data modification can only be read after the transaction is committed.

- ♦ Read Uncommitted: The data modification can be read by other transactions before a transaction is committed.

The isolation is guaranteed by MVCC (Multi-Version Consistency Control) and locks.

The diagram below takes Repeatable Read as an example to demonstrate how MVCC works:

There are two hidden columns for each row: `transaction_id` and `roll_pointer`. When transaction A starts, a new Read View with `transaction_id=201` is created. Shortly afterward, transaction B starts, and a new Read View with `transaction_id=202` is created.

Now transaction A modifies the balance to 200, a new row of the log is created, and the `roll_pointer` points to the old row. Before transaction A commits, transaction B reads the balance data. Transaction B finds that `transaction_id 201` is not committed, it reads the next committed record(`transaction_id=200`).

Even when transaction A commits, transaction B still reads data based on the Read View created when transaction B starts. So transaction B always reads the data with `balance=100`.

Over to you: have you seen isolation levels used in the wrong way?
Did it cause serious outages?

Check out our bestselling system design books.

Paperback: [Amazon](#) Digital: [ByteByteGo](#).

What is IaaS/PaaS/SaaS?

The diagram below illustrates the differences between IaaS (Infrastructure-as-a-Service), PaaS (Platform-as-a-Service), and SaaS (Software-as-a-Service).

Cloud Computing Services: Who Manages What?



For a non-cloud application, we own and manage all the hardware and software. We say the application is on-premises.

With cloud computing, cloud service vendors provide three kinds of models for us to use: IaaS, PaaS, and SaaS.

IaaS provides us access to cloud vendors' infrastructure, like servers, storage, and networking. We pay for the infrastructure service and install and manage supporting software on it for our application.

PaaS goes further. It provides a platform with a variety of middleware, frameworks, and tools to build our application. We only focus on application development and data.

SaaS enables the application to run in the cloud. We pay a monthly or annual fee to use the SaaS product.

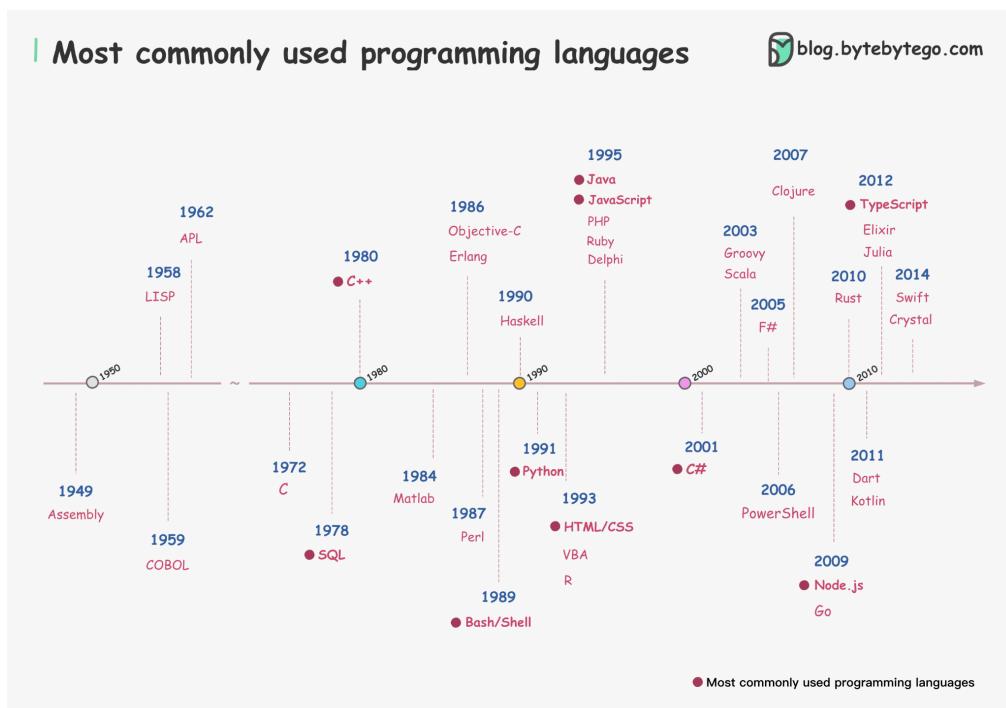
Over to you: which IaaS/PaaS/SaaS products have you used? How do you decide which architecture to use?

Image Source: <https://www.ibm.com/cloud/learn/iaas-paas-saas>

Most popular programming languages

Programming languages come and go. Some stand the test of time. Some already are shooting stars and some are rising rapidly on the horizon.

I draw a diagram by putting the top 38 most commonly used programming languages in one place, sorted by year. Data source: StackOverflow survey.



- 1 JavaScript
- 2 HTML/CSS
- 3 Python
- 4 SQL
- 5 Java
- 6 Node
- 7 TypeScript
- 8 C
- 9 Bash/Shell
- 10 C
- 11 PHP

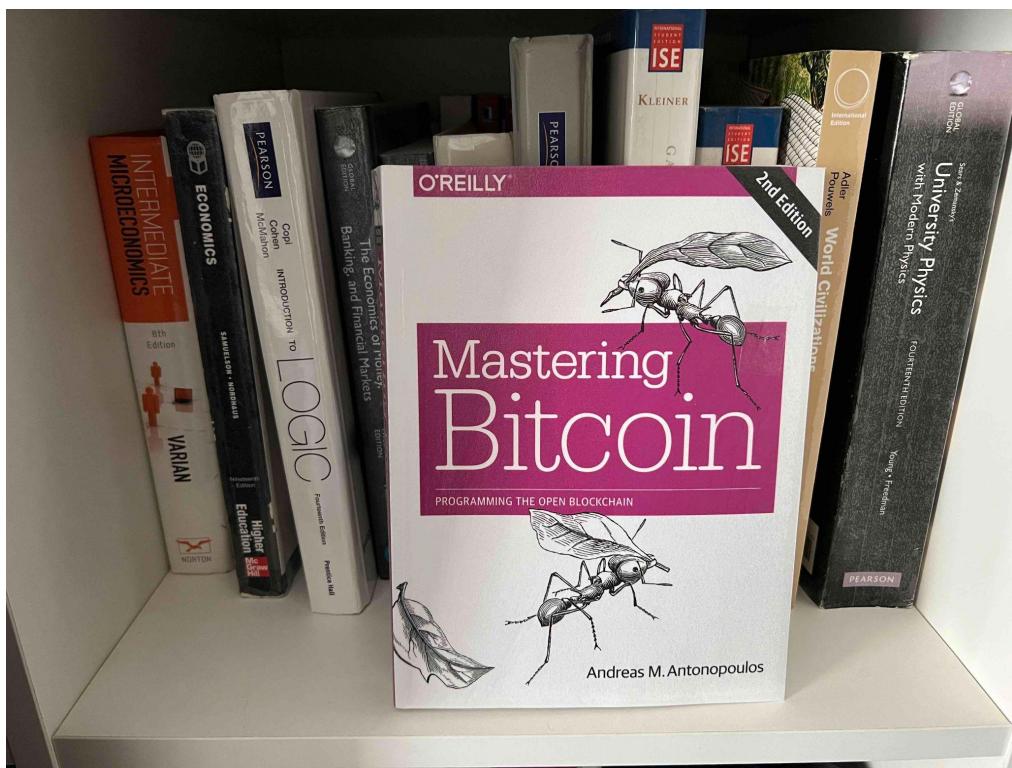
- 12 C
- 13 PowerShell
- 14 Go
- 15 Kotlin
- 16 Rust
- 17 Ruby
- 18 Dart
- 19 Assembly
- 20 Swift
- 21 R
- 22 VBA
- 23 Matlab
- 24 Groovy
- 25 Objective-C
- 26 Scala
- 27 Perl
- 28 Haskell
- 29 Delphi
- 30 Clojure
- 31 Elixir
- 32 LISP
- 33 Julia
- 34 F
- 35 Erlang
- 36 APL
- 37 Crystal
- 38 COBOL

Over to you: what's the first programming language you learned? And what are the other languages you learned over the years?

What is the future of online payments?

I don't know the answer, but I do know one of the candidates is the blockchain.

As a fan of technology, I always seek new solutions to old challenges. A book that explains a lot about an emerging payment system is 'Mastering Bitcoin' by Andreas M. Antonopoulos. I want to share my discovery of this book with you because it explains very clearly bitcoin and its underlying blockchain. This book makes me rethink how to renovate payment systems.



Here are the takeaways:

1. The bitcoin wallet balance is calculated on the fly, while the traditional wallet balance is stored in the database. You can check chapter 12 of System Design Interview Volume 2, on how to implement a traditional wallet (<https://amzn.to/34G2vmC>).

2. The golden source of truth for bitcoin is the blockchain, which is also the journal. It's the same if we use Event Sourcing architecture to build a traditional wallet, although there are other options.
3. There is a small virtual machine for bitcoin - and also Ethereum. The virtual machine defines a set of bytecodes to do basic tasks such as validation.

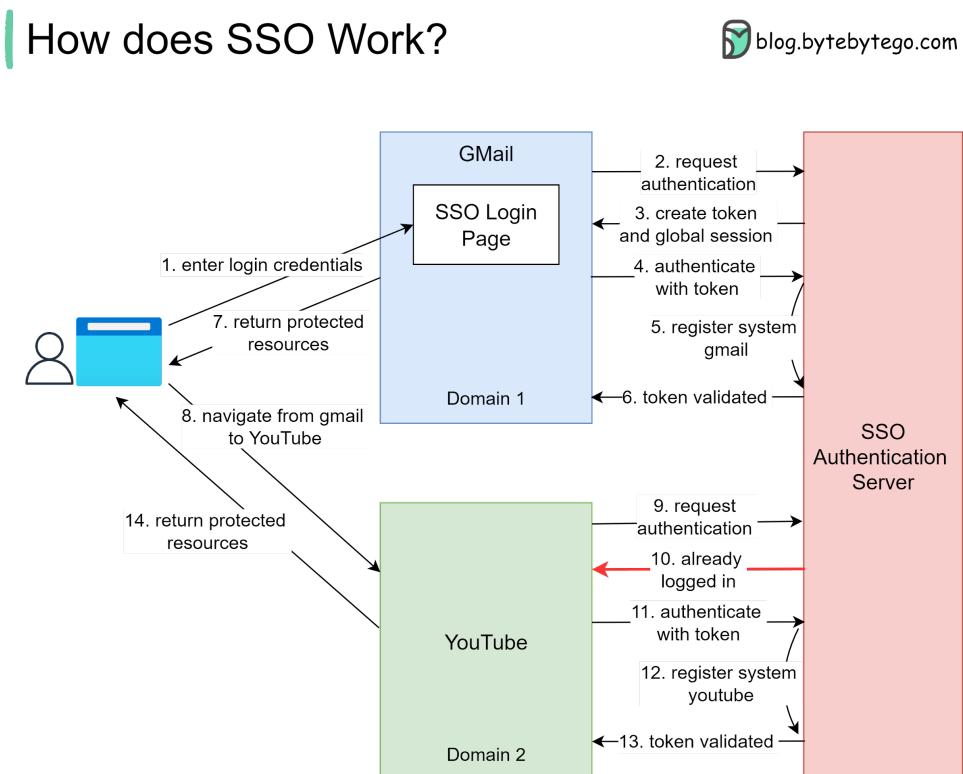
Over to you: if Elon Musk set up a base on planet Mars, what payment solution will you recommend?

What is SSO (Single Sign-On)?

A friend recently went through the irksome experience of being signed out from a number of websites they use daily. This event will be familiar to millions of web users, and it is a tedious process to fix. It can involve trying to remember multiple long-forgotten passwords, or typing in the names of pets from childhood to answer security questions. SSO removes this inconvenience and makes life online better. But how does it work?

Basically, Single Sign-On (SSO) is an authentication scheme. It allows a user to log in to different systems using a single ID.

The diagram below illustrates how SSO works.



Step 1: A user visits Gmail, or any email service. Gmail finds the user is not logged in and so redirects them to the SSO authentication server, which also finds the user is not logged in. As a result, the user

is redirected to the SSO login page, where they enter their login credentials.

Steps 2-3: The SSO authentication server validates the credentials, creates the global session for the user, and creates a token.

Steps 4-7: Gmail validates the token in the SSO authentication server. The authentication server registers the Gmail system, and returns “valid.” Gmail returns the protected resource to the user.

Step 8: From Gmail, the user navigates to another Google-owned website, for example, YouTube.

Steps 9-10: YouTube finds the user is not logged in, and then requests authentication. The SSO authentication server finds the user is already logged in and returns the token.

Step 11-14: YouTube validates the token in the SSO authentication server. The authentication server registers the YouTube system, and returns “valid.” YouTube returns the protected resource to the user.

The process is complete and the user gets back access to their account.

Over to you:

Question 1: have you implemented SSO in your projects? What is the most difficult part?

Question 2: what's your favorite sign-in method and why?

Check out our bestselling system design books.

Paperback: [Amazon](#) Digital: [ByteByteGo](#).

How to store passwords safely in the database?

Let's take a look.

Things NOT to do

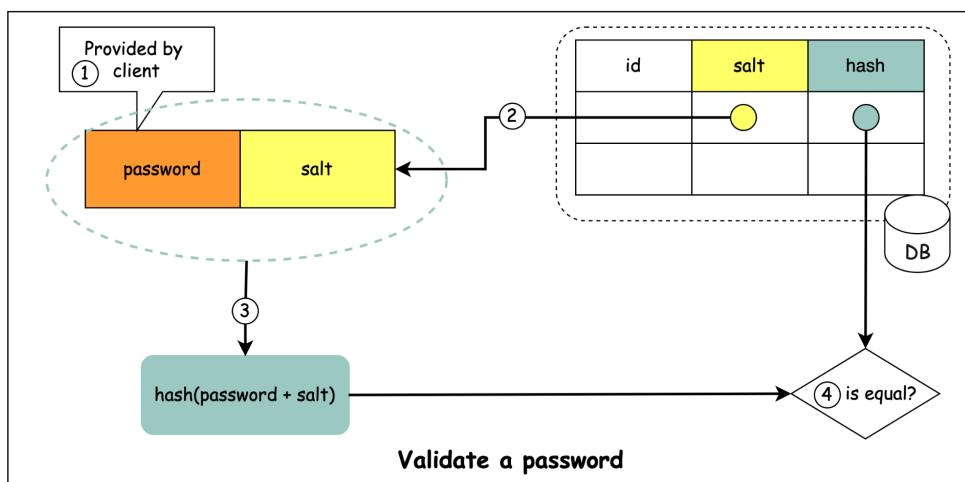
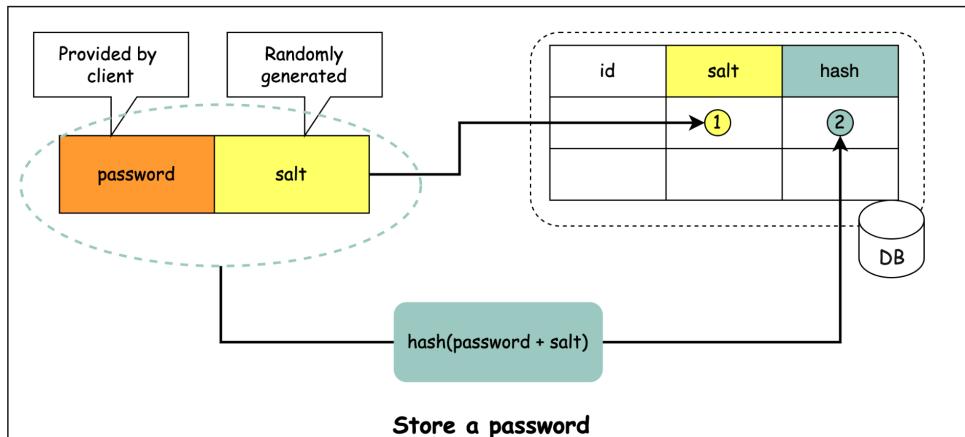
- ◆ Storing passwords in plain text is not a good idea because anyone with internal access can see them.
- ◆ Storing password hashes directly is not sufficient because it is prone to precomputation attacks, such as rainbow tables.
- ◆ To mitigate precomputation attacks, we salt the passwords.

What is salt?

According to OWASP guidelines, “a salt is a unique, randomly generated string that is added to each password as part of the hashing process”.

How to store passwords in DB?

 blog.bytebytego.com



How to store a password and salt?

- ① A salt is not meant to be secret and it can be stored in plain text in the database. It is used to ensure the hash result is unique to each password.
- ② The password can be stored in the database using the following format: $\text{hash}(\text{password} + \text{salt})$.

How to validate a password?

To validate a password, it can go through the following process:

- ① A client enters the password.
- ② The system fetches the corresponding salt from the database.

- ③ The system appends the salt to the password and hashes it. Let's call the hashed value H1.
- ④ The system compares H1 and H2, where H2 is the hash stored in the database. If they are the same, the password is valid.

Over to you: what other mechanisms can we use to ensure password safety?

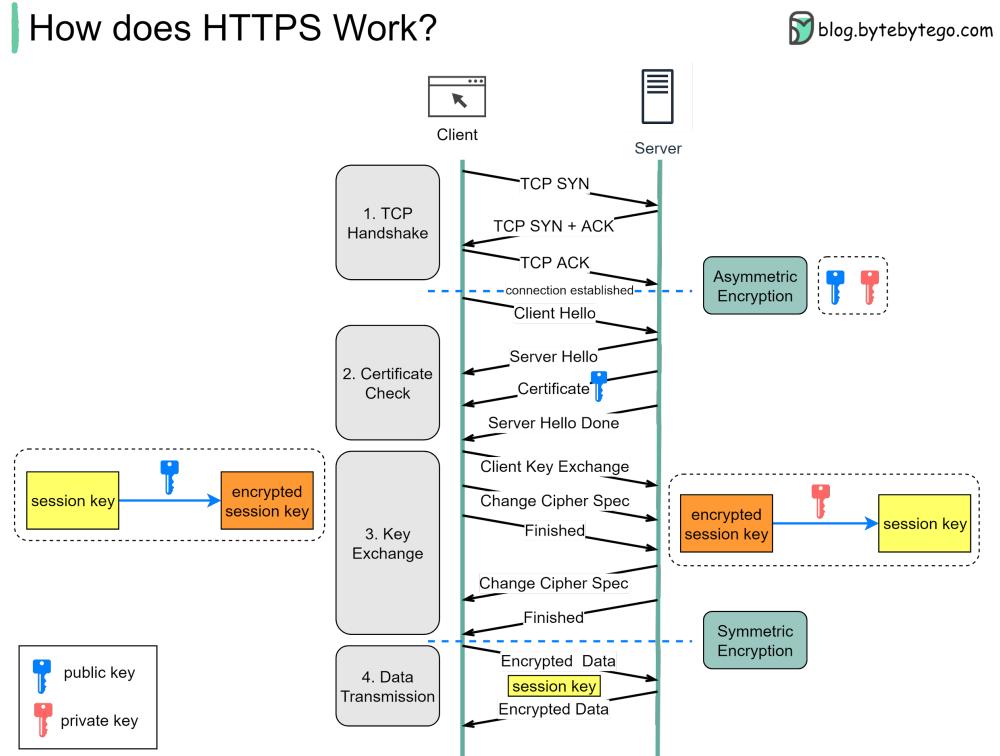
—

Check out our bestselling system design books.

Paperback: [Amazon](#) Digital: [ByteByteGo](#).

How does HTTPS work?

Hypertext Transfer Protocol Secure (HTTPS) is an extension of the Hypertext Transfer Protocol (HTTP.) HTTPS transmits encrypted data using Transport Layer Security (TLS.) If the data is hijacked online, all the hijacker gets is binary code.



How is the data encrypted and decrypted?

Step 1 - The client (browser) and the server establish a TCP connection.

Step 2 - The client sends a “client hello” to the server. The message contains a set of necessary encryption algorithms (cipher suites) and the latest TLS version it can support. The server responds with a “server hello” so the browser knows whether it can support the algorithms and TLS version.

The server then sends the SSL certificate to the client. The certificate contains the public key, host name, expiry dates, etc. The client validates the certificate.

Step 3 - After validating the SSL certificate, the client generates a session key and encrypts it using the public key. The server receives the encrypted session key and decrypts it with the private key.

Step 4 - Now that both the client and the server hold the same session key (symmetric encryption), the encrypted data is transmitted in a secure bi-directional channel.

Why does HTTPS switch to symmetric encryption during data transmission? There are two main reasons:

1. Security: The asymmetric encryption goes only one way. This means that if the server tries to send the encrypted data back to the client, anyone can decrypt the data using the public key.
2. Server resources: The asymmetric encryption adds quite a lot of mathematical overhead. It is not suitable for data transmissions in long sessions.

Over to you: how much performance overhead does HTTPS add, compared to HTTP?

Check out our bestselling system design books.

Paperback: [Amazon](#) Digital: [ByteByteGo](#).

How to learn design patterns?

Besides reading a lot of well-written code, a good book guides us like a good teacher.

Head First Design Patterns, second edition, is the one I would recommend.



When I began my journey in software engineering, I found it hard to understand the classic textbook, **Design Patterns**, by the Gang of Four. Luckily, I discovered Head First Design Patterns in the school library. This book solved a lot of puzzles for me. When I went back to the Design Patterns book, everything looked familiar and more understandable.

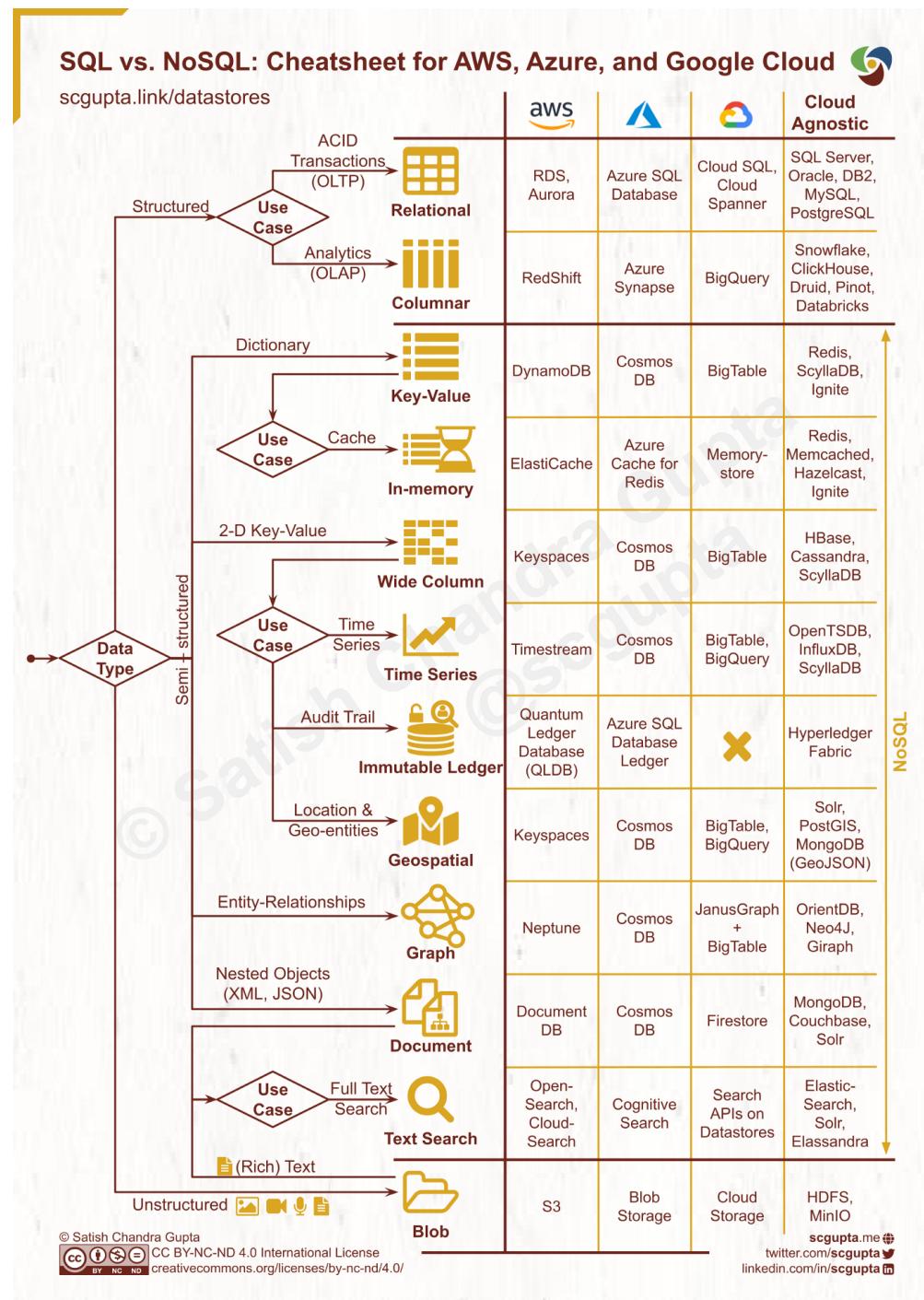
Last year, I bought the second edition of Head First Design Patterns and read through it. Here are a few things I like about the book:

- ◆ This book solves the challenge of software's abstract, "invisible" nature. Software is difficult to build because we cannot see its architecture; its details are embedded in the code and binary files. It is even harder to understand software design patterns because these are higher-level abstractions of the software. The book fixes this by using visualization. There are lots of diagrams, arrows, and comments on almost every page. If I do not understand the text, it's no problem. The diagrams explain things very well.
- ◆ We all have questions we are afraid to ask when we first learn a new skill. Maybe we think it's an easy one. This book is good at tackling design patterns from the student's point of view. It guides us by asking our questions and clearly answering them. There is a Guru in the book and there's also a Student.

Over to you: which book helped you understand a challenging topic?
Why do you like it?

A visual guide on how to choose the right Database

Picking a database is a long-term commitment so the decision shouldn't be made lightly. The important thing to keep in mind is to choose the right database for the right job.



Data can be structured (SQL table schema), semi-structured (JSON, XML, etc.), and unstructured (Blob).

Common database categories include:

- ◆ Relational
- ◆ Columnar
- ◆ Key-value
- ◆ In-memory
- ◆ Wide column
- ◆ Time Series
- ◆ Immutable ledger
- ◆ Geospatial
- ◆ Graph
- ◆ Document
- ◆ Text search
- ◆ Blob

Thanks, [Satish Chandra Gupta](#)

Over to you - Which database have you used for which workload?

Do you know how to generate globally unique IDs?

In this post, we will explore common requirements for IDs that are used in social media such as Facebook, Twitter, and LinkedIn.

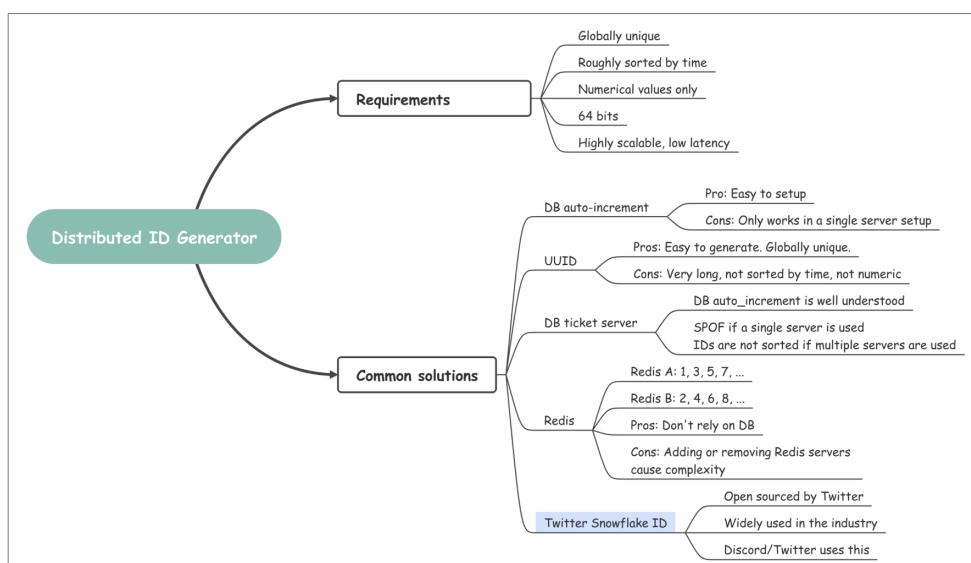
Requirements:

- ◆ Globally unique
- ◆ Roughly sorted by time
- ◆ Numerical values only
- ◆ 64 bits
- ◆ Highly scalable, low latency

Unique ID Generator

 blog.bytebytogo.com

Reasons	
Globally unique	If IDs are not globally unique, there could be collisions.
Roughly sorted by time	So user IDs, post IDs can be sorted by time without fetching additional info
Numerical values only	Naturally sortable by time
64 bits	$2^{32} = \sim 4$ billion \rightarrow not enough IDs. 2^{64} is big enough 2^{128} wastes space and is too long
Highly scalable, low latency	Ability to generate a lot of IDs per second in low latency fashion is critical.



The implementation details of the algorithms can be found online so we will not go into detail here.

Over to you: What kind of ID generators have you used?

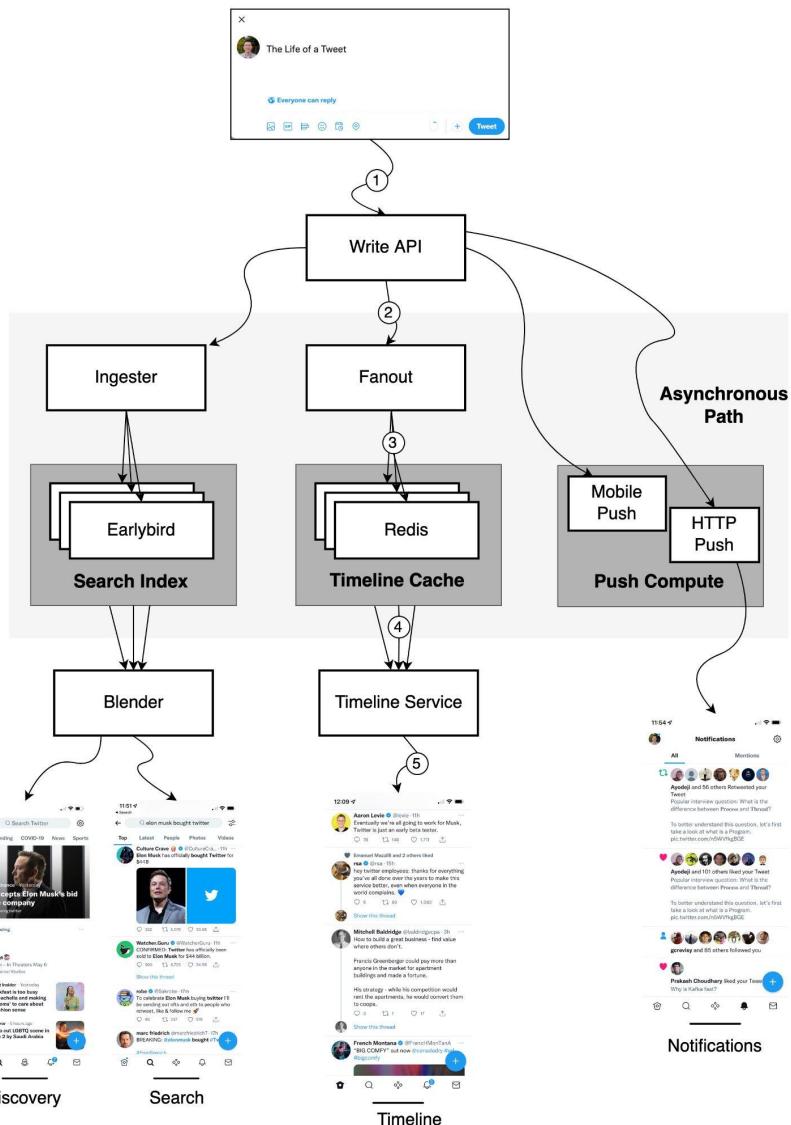
—

Check out our bestselling system design books.

Paperback: [Amazon](#) Digital: [ByteByteGo](#).

How does Twitter work?

This post is a summary of a tech talk given by Twitter in 2013. Let's take a look.



The Life of a Tweet:

- ① A tweet comes in through the Write API.
- ② The Write API routes the request to the Fanout service.
- ③ The Fanout service does a lot of processing and stores them in the Redis cache.

- 4 The Timeline service is used to find the Redis server that has the home timeline on it.
- 5 A user pulls their home timeline through the Timeline service.

Search & Discovery

- ◆ Ingester: annotates and tokenizes Tweets so the data can be indexed.
- ◆ Earlybird: stores search index.
- ◆ Blender: creates the search and discovery timelines.

Push Compute

- ◆ HTTP push
- ◆ Mobile push

Disclaimer: This article is based on the tech talk given by Twitter in 2013 (<https://bit.ly/3vNfjRp>). Even though many years have passed, it's still quite relevant. I redraw the diagram as the original diagram is difficult to read.

Over to you:

Do you use Twitter? What are some of the biggest differences between LinkedIn and Twitter that might shape their system architectures?

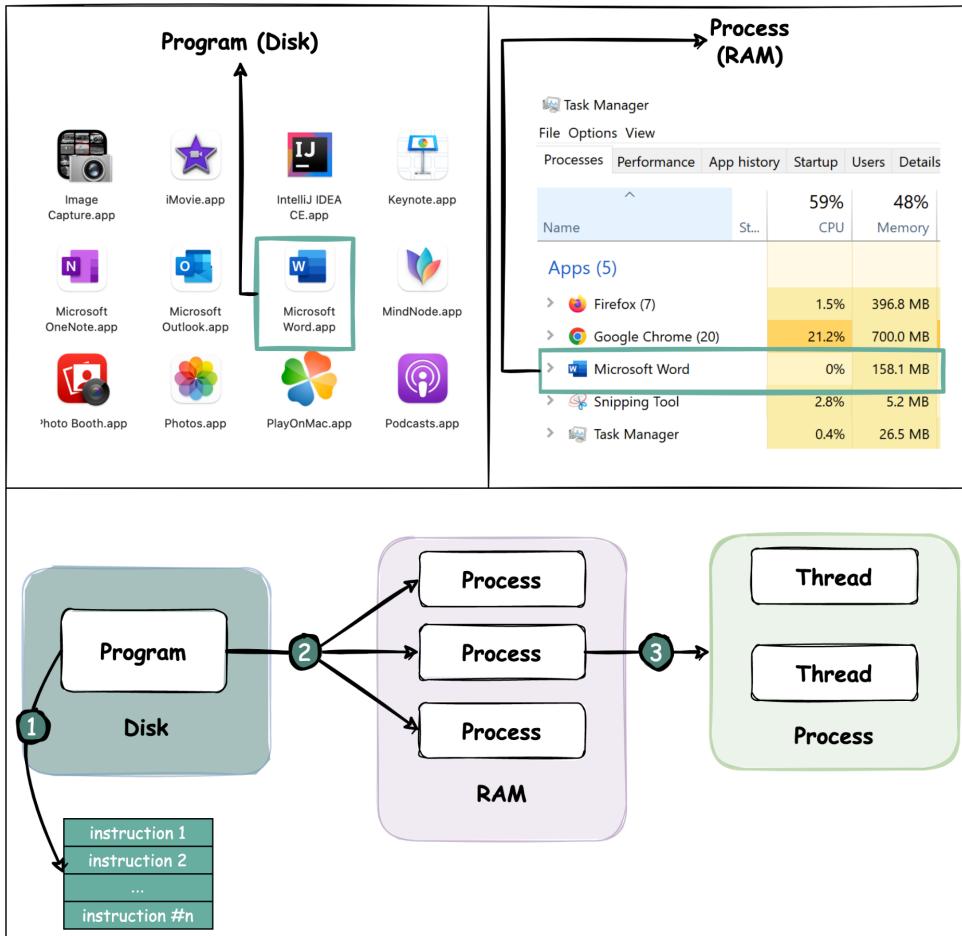
Check out our bestselling system design books.

Paperback: [Amazon](#) Digital: [ByteByteGo](#).

What is the difference between Process and Thread?

Program vs Process vs Thread

 blog.bytebytogo.com



To better understand this question, let's first take a look at what is a Program. A **Program** is an executable file containing a set of instructions and passively stored on disk. One program can have multiple processes. For example, the Chrome browser creates a different process for every single tab.

A **Process** means a program is in execution. When a program is loaded into the memory and becomes active, the program becomes a process. The process requires some essential resources such as registers, program counter, and stack.

A Thread is the smallest unit of execution within a process.

The following process explains the relationship between program, process, and thread.

1. The program contains a set of instructions.
2. The program is loaded into memory. It becomes one or more running processes.
3. When a process starts, it is assigned memory and resources. A process can have one or more threads. For example, in the Microsoft Word app, a thread might be responsible for spelling checking and the other thread for inserting text into the doc.

Main differences between process and thread:

- ◆ Processes are usually independent, while threads exist as subsets of a process.
- ◆ Each process has its own memory space. Threads that belong to the same process share the same memory.
- ◆ A process is a heavyweight operation. It takes more time to create and terminate.
- ◆ Context switching is more expensive between processes.
- ◆ Inter-thread communication is faster for threads.

Over to you:

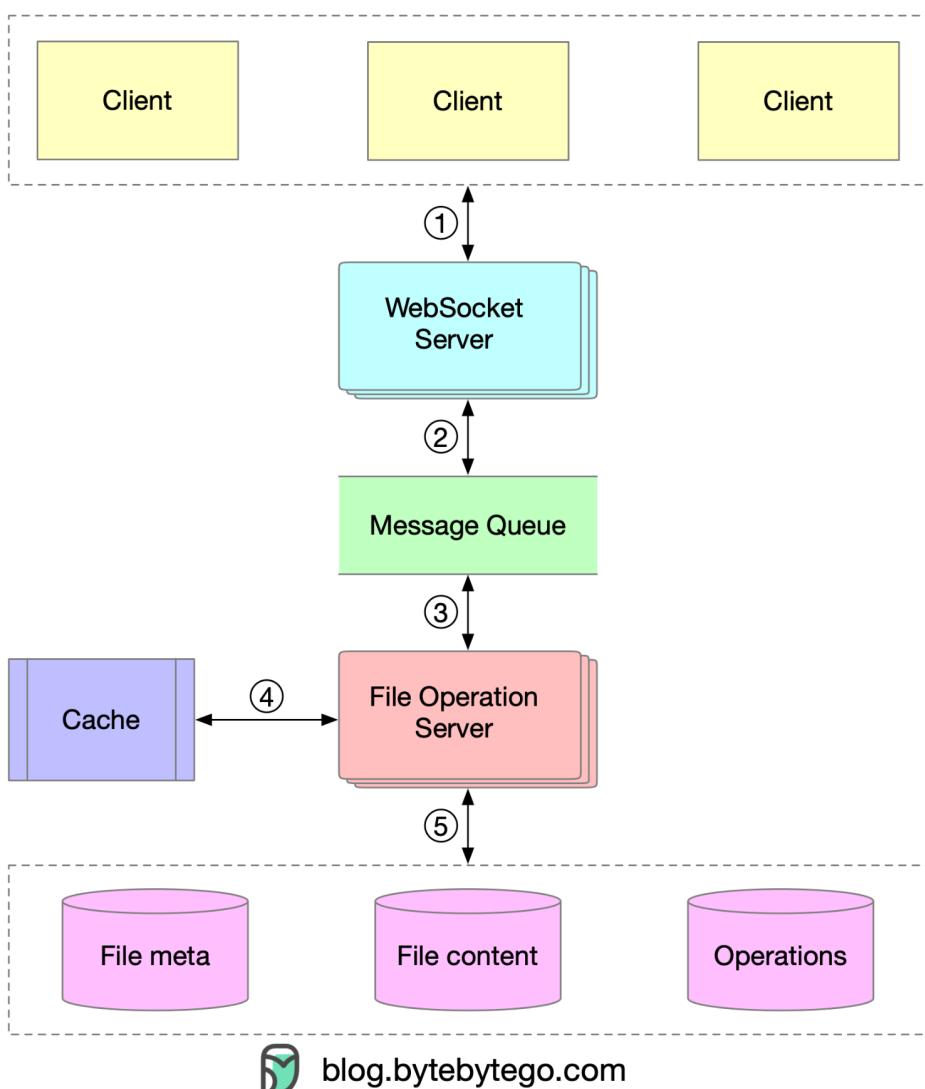
- 1). Some programming languages support coroutine. What is the difference between coroutine and thread?
- 2). How to list running processes in Linux?

Check out our bestselling system design books.

Paperback: [Amazon](#) Digital: [ByteByteGo](#).

Interview Question: design Google Docs

How to Design Google Doc?



- ① Clients send document editing operations to the WebSocket Server.
- ② The real-time communication is handled by the WebSocket Server.
- ③ Documents operations are persisted in the Message Queue.

- ④ The File Operation Server consumes operations produced by clients and generates transformed operations using collaboration algorithms.
- ⑤ Three types of data are stored: file metadata, file content, and operations.

One of the biggest challenges is real-time conflict resolution. Common algorithms include:

- ◆ Operational transformation (OT)
- ◆ Differential Synchronization (DS)
- ◆ Conflict-free replicated data type (CRDT)

Google Doc uses OT according to its Wikipedia page and CRDT is an active area of research for real-time concurrent editing.

Over to you - Have you encountered any issues while using Google Docs? If so, what do you think might have caused the issue?

Check out our bestselling system design books.

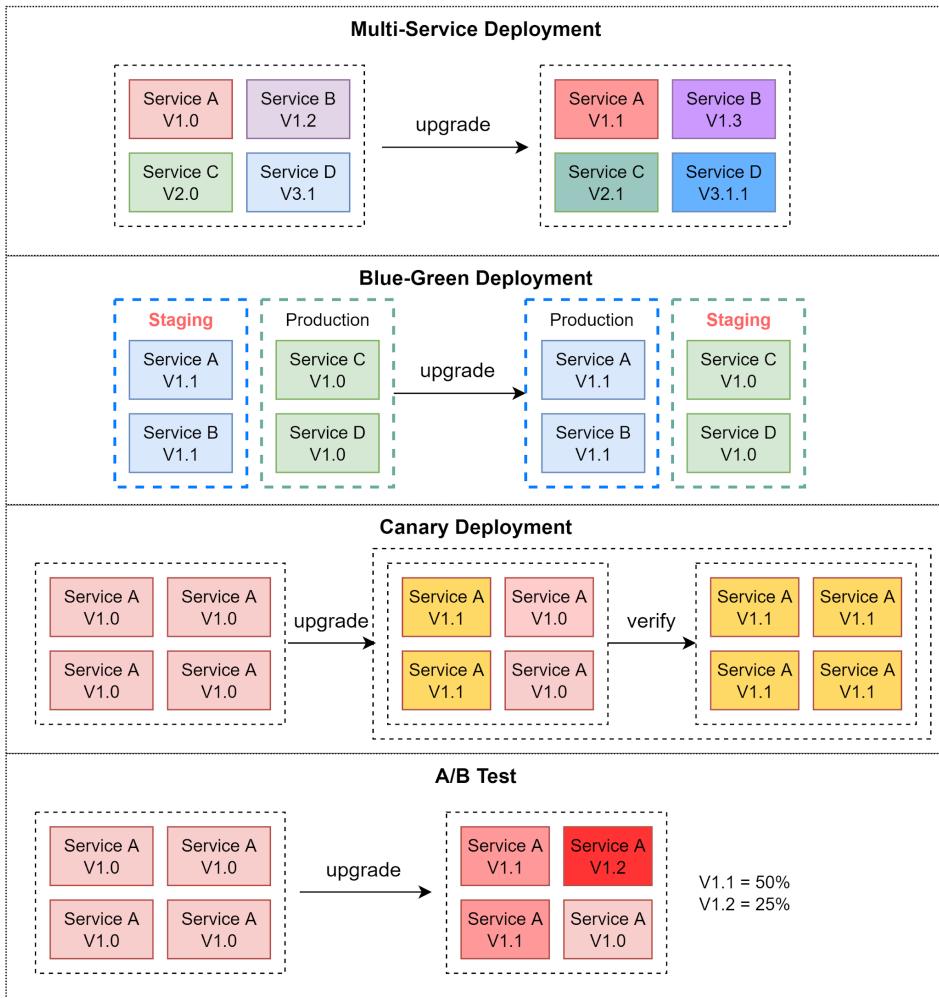
Paperback: [Amazon](#) Digital: [ByteByteGo](#).

Deployment strategies

Deploying or upgrading services is risky. In this post, we explore risk mitigation strategies.

The diagram below illustrates the common ones.

How to Deploy Services?



Multi-Service Deployment

In this model, we deploy new changes to multiple services simultaneously. This approach is easy to implement. But since all the services are upgraded at the same time, it is hard to manage and test dependencies. It's also hard to rollback safely.

Blue-Green Deployment

With blue-green deployment, we have two identical environments: one is staging (blue) and the other is production (green). The staging environment is one version ahead of production. Once testing is done in the staging environment, user traffic is switched to the staging environment, and the staging becomes the production. This deployment strategy is simple to perform rollback, but having two identical production quality environments could be expensive.

Canary Deployment

A canary deployment upgrades services gradually, each time to a subset of users. It is cheaper than blue-green deployment and easy to perform rollback. However, since there is no staging environment, we have to test on production. This process is more complicated because we need to monitor the canary while gradually migrating more and more users away from the old version.

A/B Test

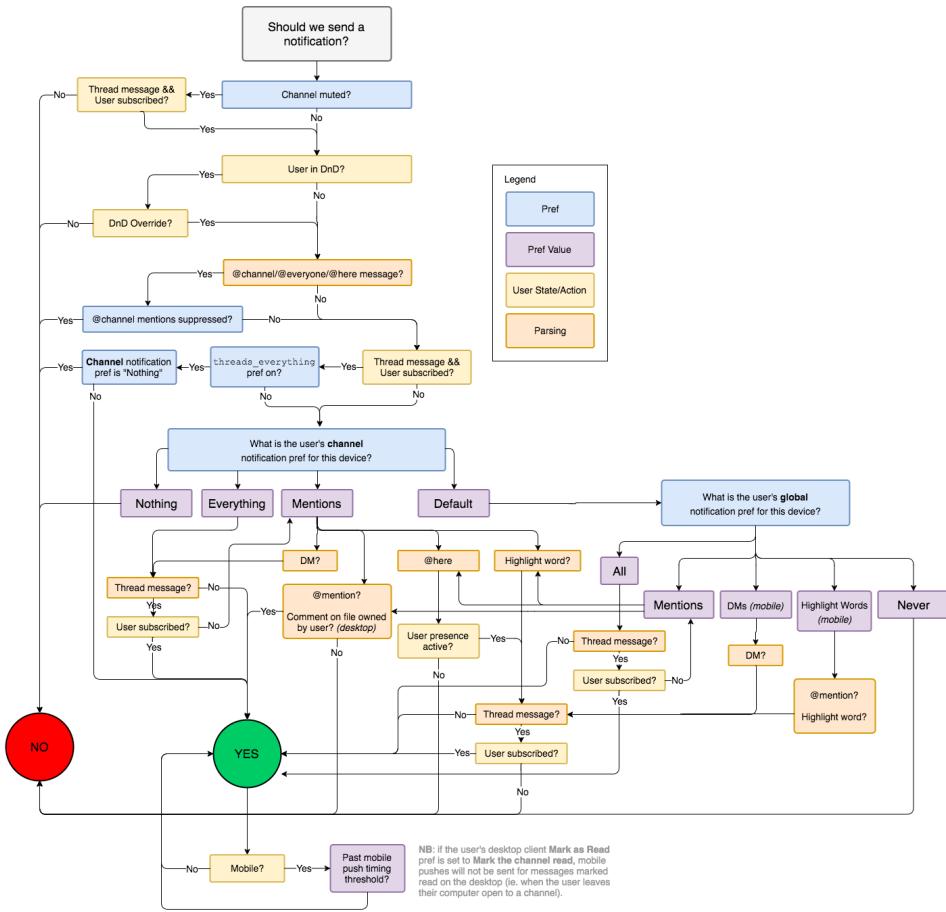
In the A/B test, different versions of services run in production simultaneously. Each version runs an “experiment” for a subset of users. A/B test is a cheap method to test new features in production. We need to control the deployment process in case some features are pushed to users by accident.

Over to you - Which deployment strategy have you used? Did you witness any deployment-related outages in production and why did they happen?

Flowchart of how slack decides to send a notification

It is a great example of why a simple feature may take much longer to develop than many people think.

When we have a great design, users may not notice the complexity because it feels like the feature is just working as intended.



What's your takeaway from this diagram?

Image source:

<https://slack.engineering/reducing-slacks-memory-footprint/>

How does Amazon build and operate the software?

In 2019, Amazon released The Amazon Builders' Library. It contains architecture-based articles that describe how Amazon architects, releases, and operates technology.

 <p>LEVEL 400</p> <p>Workload isolation using shuffle-sharding</p> <p>Author: Colm MacCarthaigh</p> <p>Shuffle Sharding is one of our core techniques for drastically limiting the scope of impact of operational issues.</p> <p>PDF Kindle</p>	 <p>LEVEL 400</p> <p>Architecting and operating resilient serverless...</p> <p>Author: David Yanacek</p> <p>In this video, we cover what AWS does to build reliable and resilient services, including avoiding modes and overload, performing bounded work, throttling at multiple layers, guarding concurrency,</p> <p>▼</p>	 <p>LEVEL 300</p> <p>Caching challenges and strategies</p> <p>Authors: Matt Brinkley, Jas Chhabra</p> <p>Improving latency and availability with caching while avoiding the modal behavior they can introduce.</p> <p>PDF Kindle</p>
 <p>LEVEL 300</p> <p>Amazon's approach to security during...</p> <p>Author: Colm MacCarthaigh</p> <p>In this video, learn about how AWS</p>	 <p>LEVEL 400</p> <p>Avoiding insurmountable queue backlogs</p> <p>Author: David Yanacek</p> <p>Prioritizing draining important</p>	 <p>LEVEL 400</p> <p>Implementing health checks</p> <p>Author: David Yanacek</p> <p>Automatically detecting and mitigating</p>

As of today, it published 26 articles. It took me two weekends to go through all the articles. I've had great fun and learned a lot. Here are some of my favorites:

- ◆ Making retries safe with idempotent APIs
- ◆ Timeouts, retries, and backoff with jitter
- ◆ Beyond five 9s: Lessons from our highest available data planes
- ◆ Caching challenges and strategies
- ◆ Ensuring rollback safety during deployments
- ◆ Going faster with continuous delivery

- ◆ Challenges with distributed systems
- ◆ Amazon's approach to high-availability deployment

Over to you: what's your favorite place to learn system design and design principles?

Link to The Amazon Builders' Library: aws.amazon.com/builders-library