~~Architecture~~
Simple Design !

What is architecture?

A short story

📅 9 years ago…

I tried to build **Netflix…**

Tiii séhla: Server / Client…

It worked! 🥳

POC

A short story

📅 9 years ago…

I tried to build **Netflix…**

Tiii séhla: Server / Client…

It worked!🥳

It never scaled…😰

Why?

What is architecture?

We don’t know…

But it’s important! ❗

More specifically: **A good architecture is important, otherwise it becomes slower and more expensive to add new capabilities in the future.**

Let’s break it down:

Software Architecture :

Software architecture is the blueprint of building software. It shows the overall structure of the software, the collection of components in it, and how they interact with one another while hiding the implementation.

Lézemna exemple

A quick example: bit.ly

Bit chénhou?

Long URL: https://www.yoursite.com/?/post/2021/12/08/$5982-! #_ref;di(@%

Short URL: https://bit.ly/3l71d3o

Requirements

Functional requirements

- **Given a URL, our service should generate a shorter and unique alias for it.**
- **Users should be redirected to the original URL when they visit the short link.**
- **Links should expire after a default timespan.**

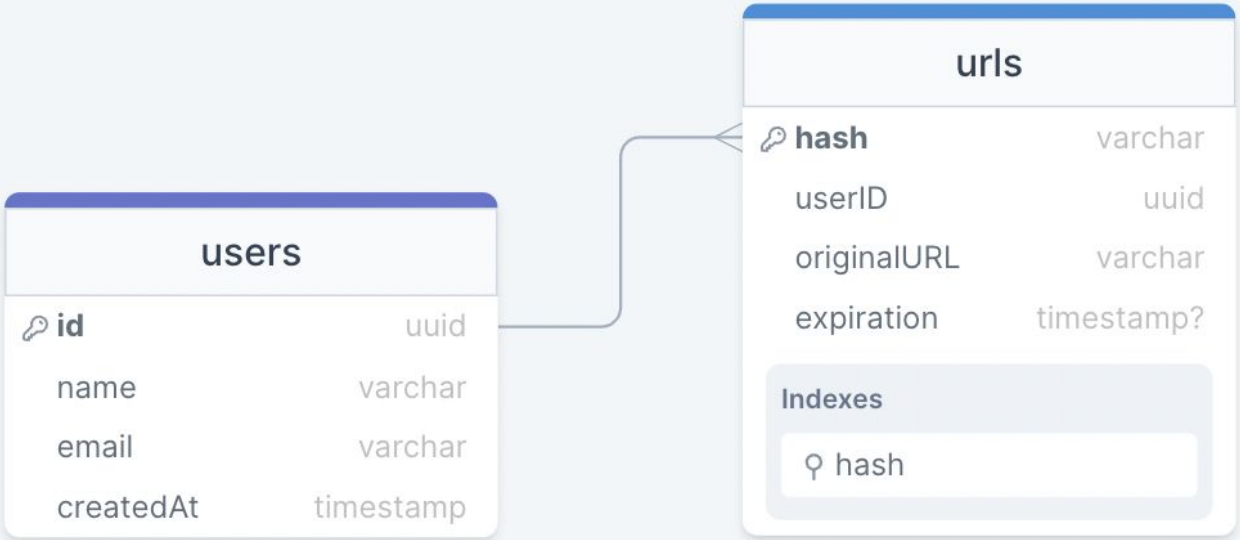
Non-functional requirements

- High availability with minimal latency.
- The system should be scalable and efficient.

Extended requirements

- Prevent abuse of services.
- Record analytics and metrics for redirections.

Data modeling



API design

- Create URL
  - createURL(apiKey: string, originalURL: string, expiration?: Date): string
- Get URL
  - getURL(apiKey: string, shortURL: string): string

Where to store the data? How to query it?

Relational database?

Graph database?

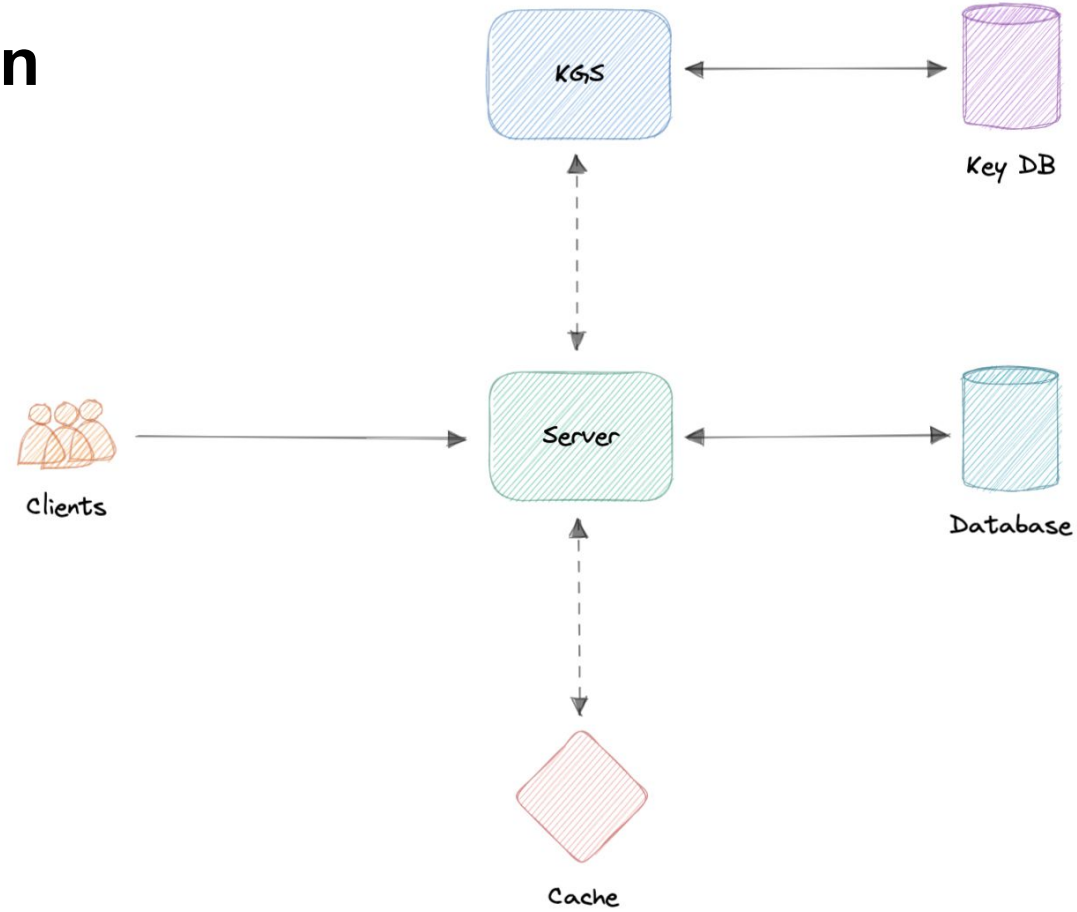
Data types

What do we have to store?

- The short string
- The target URL
- The email of the user
- The API key
- Dates (creation and expiration)

Question: what data types will-we use?

High-level design



Requirements

Functional requirements

- Given a URL, our service should generate a shorter and unique alias for it.
- Users should be redirected to the original URL when they visit the short link.
- Links should expire after a default timespan.

Non-functional requirements

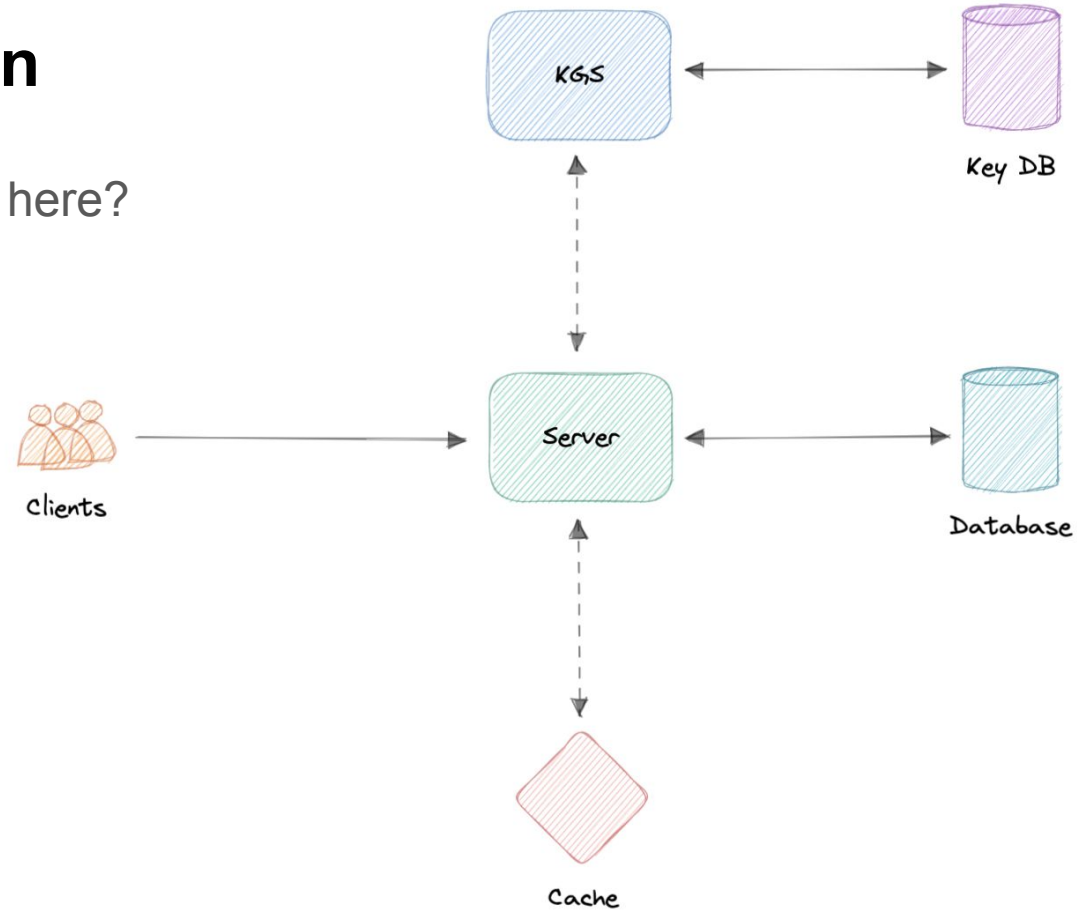
- **High availability with minimal latency.**
- **The system should be scalable and efficient**.

Extended requirements

- **Prevent abuse of services.**
- Record analytics and metrics for redirections.

High-level design

Do you see the problem here?



Estimations…

It’s a read-heavy system… let’s assume **100:1 read/writes**

Let’s expect **100 Million links generated per month**

….

So for around 100 million urls shortened, we will have 10 billion visits…

That’s around 40 inserted urls/second, and 4000 read requests / second

…

Each record will cost ~512 Ko

So after the 5 years, we will have 6 billion records in the DB ~ 3 TB

…

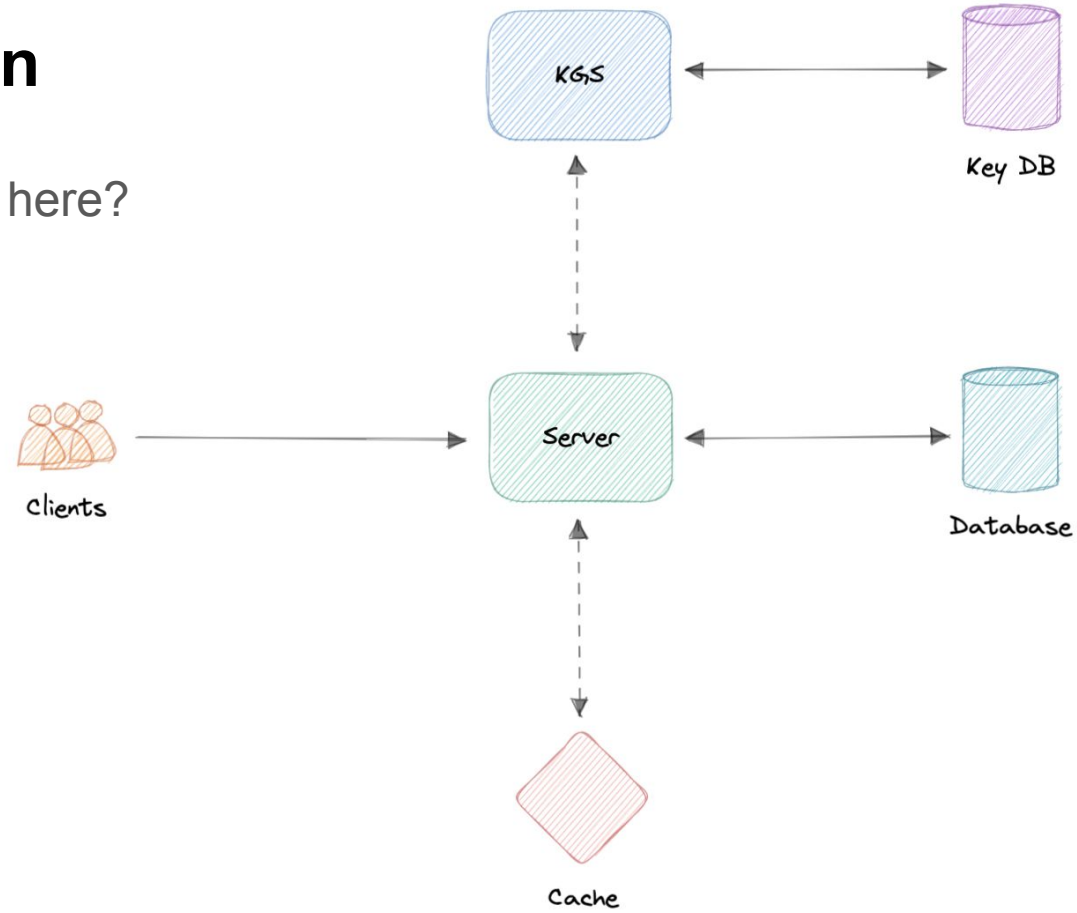
Incoming bandwidth: 20KB/s

Outgoing bandwidth: 2MB/s

High-level design

Do you see the problem here?

How to improve this?



Patterns

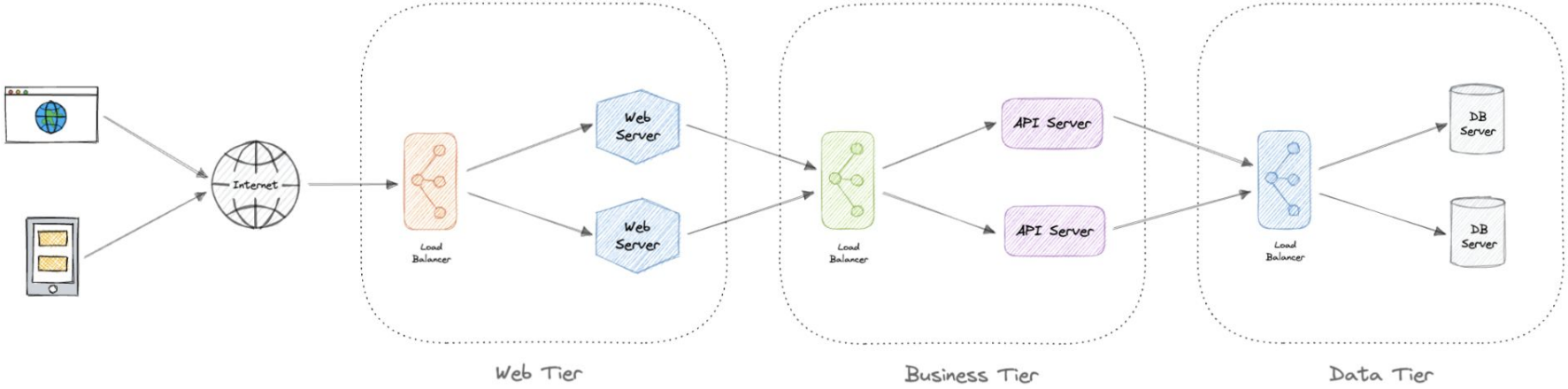
Patterns

- Layered
- Client-Server
- Event-Driven
- Monoliths
- Microservices
- Service Oriented Architecture
- Micro Frontends

Client / Server

It’s as easy as WWW, emails, FTP, …

Layered Pattern: also called N-tier architecture



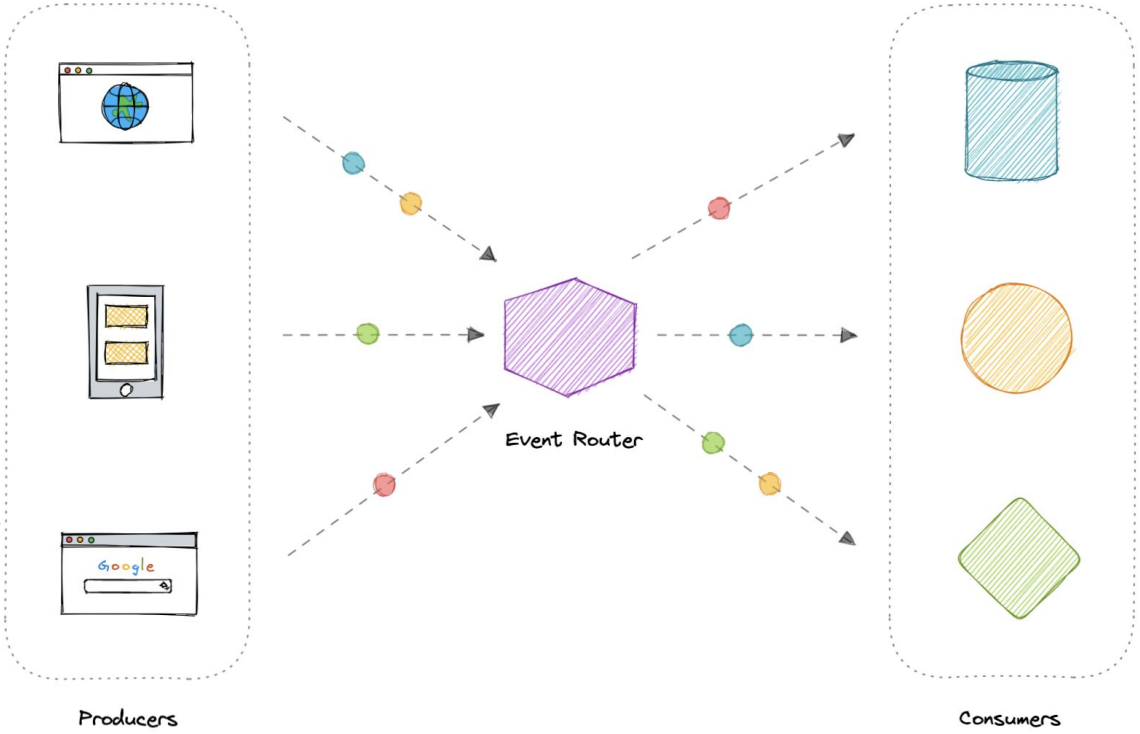
Generally speaking, we will have 3 layers… but we can go down to 2, or 1

Event-Driven

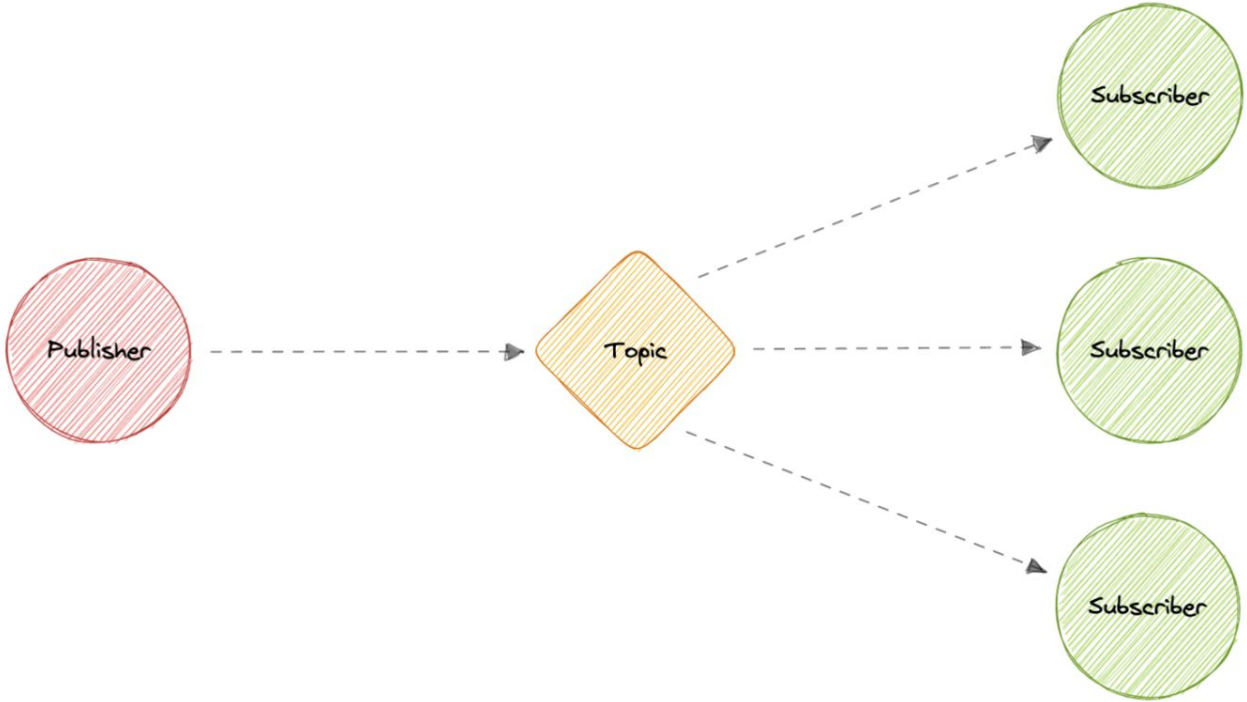
- Event producers: Publishes an event to the router.
- Event routers: Filters and pushes the events to consumers.
- Event consumers: Uses events to reflect changes in the system

Here we have different approaches:

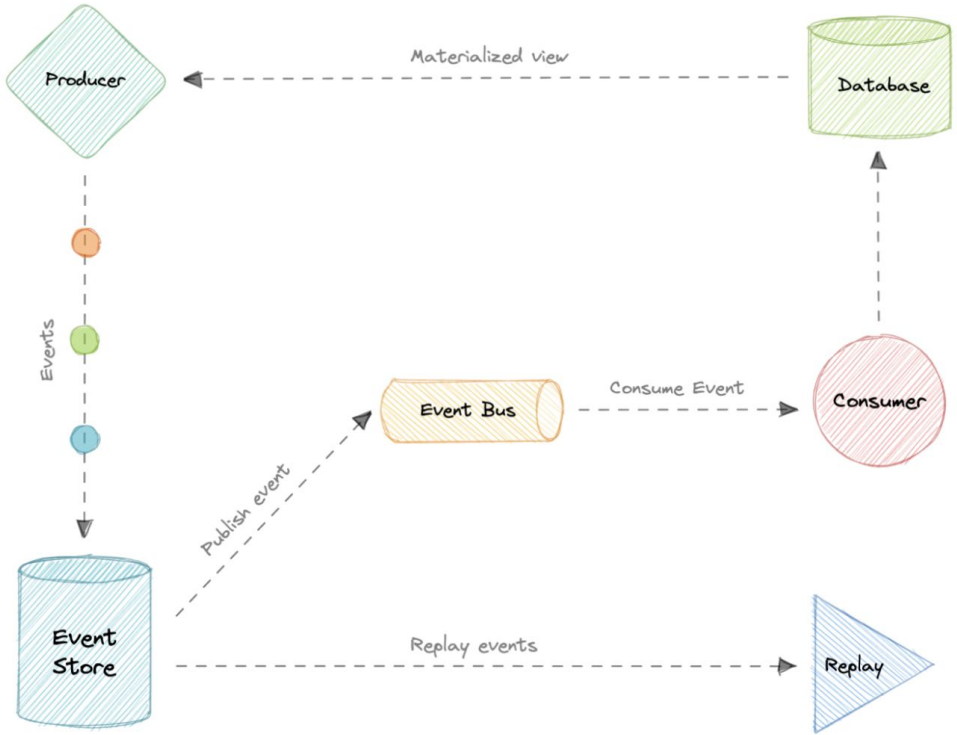
- **Publish-Subscribe**
- **Event Sourcing**
- **Command and Query Responsibility Segregation (CQRS)**



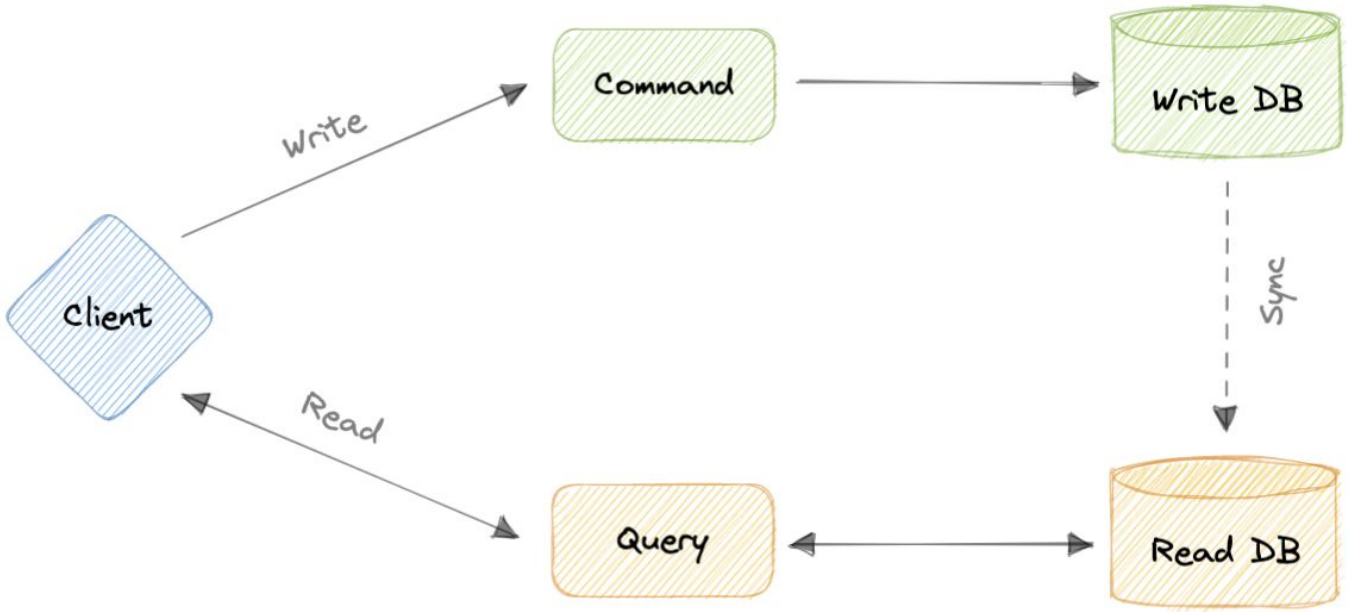
Publish & Subscribe



Event Sourcing



Command and Query Responsibility Segregation (CQRS)



Pro / Cons of CQRS

Advantages

Let's discuss some advantages of CQRS:

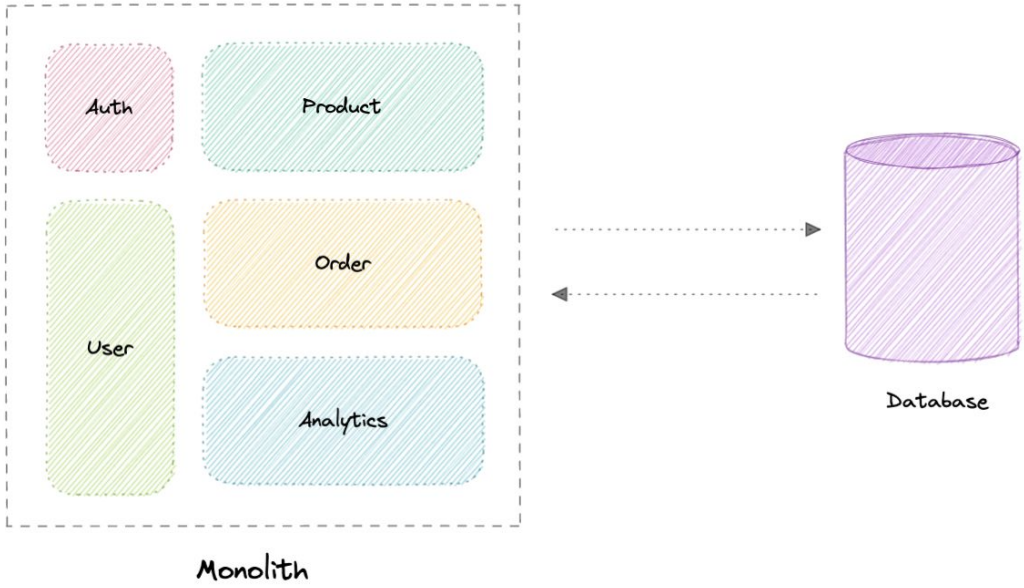
- Allows independent scaling of read and write workloads.
- Easier scaling, optimizations, and architectural changes.
- Closer to business logic with loose coupling.
- The application can avoid complex joins when querying.
- Clear boundaries between the system behavior.

Disadvantages

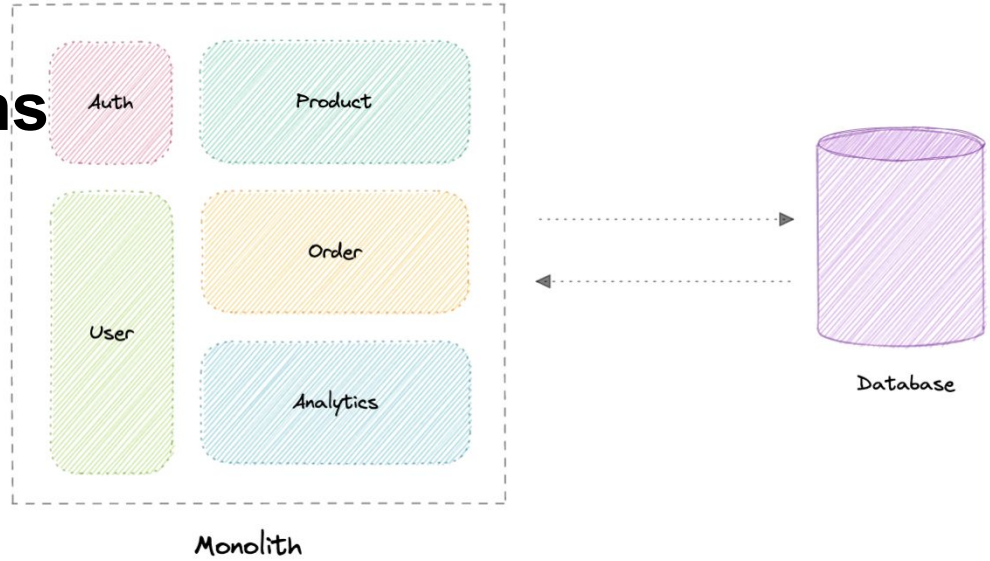
Below are some disadvantages of CQRS:

- More complex application design.
- Message failures or duplicate messages can occur.
- Dealing with eventual consistency is a challenge.
- Increased system maintenance efforts.

Monoliths (micro kernel)



Advantages of monoliths



Advantages

Following are some advantages of monoliths:

- Simple to develop or debug.
- Fast and reliable communication.
- Easy monitoring and testing.
- Supports ACID transactions.

Disadvantages

Some common disadvantages of monoliths are:

- Maintenance becomes hard as the codebase grows.
- Tightly coupled application, hard to extend.
- Requires commitment to a particular technology stack.
- On each update, the entire application is redeployed.
- Reduced reliability as a single bug can bring down the entire system.
- Difficult to scale or adopt new technologies.

Microservices

Advantages

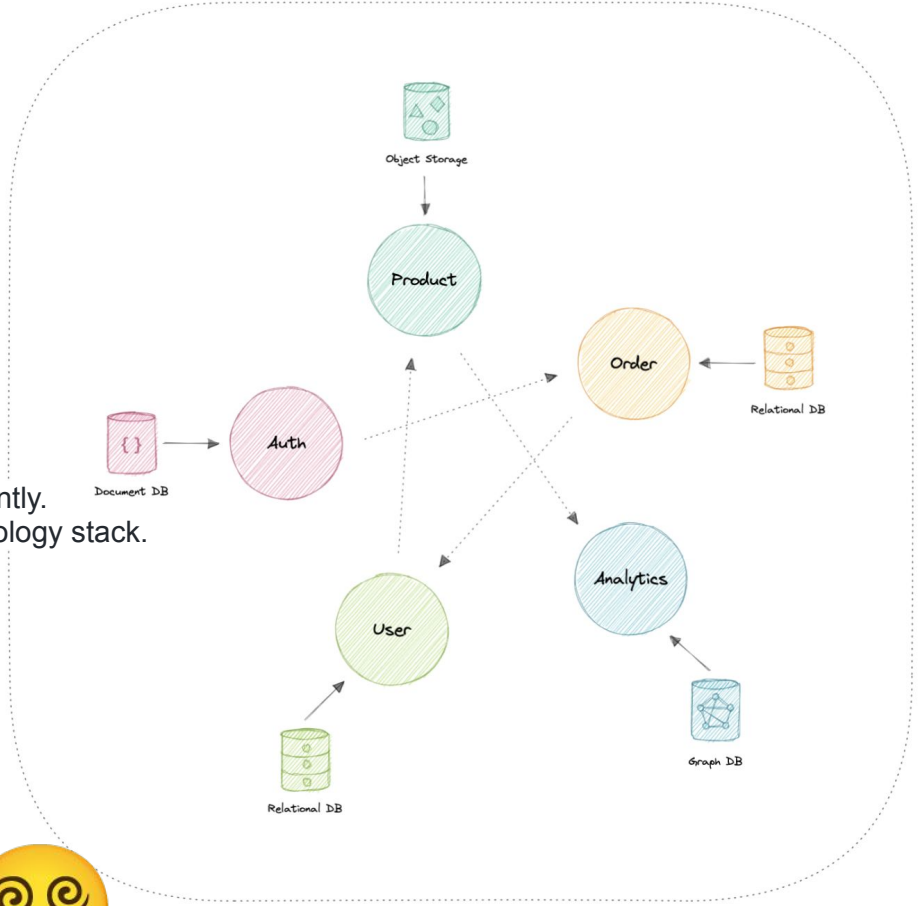
Here are some advantages of microservices architecture:

- Loosely coupled services.
- Services can be deployed independently.
- Highly agile for multiple development teams.
- Improves fault tolerance and data isolation.
- Better scalability as each service can be scaled independently.
- Eliminates any long-term commitment to a particular technology stack.

Disadvantages

Microservices architecture brings its own set of challenges:

- Complexity of a distributed system.
- Testing is more difficult.
- Expensive to maintain (individual servers, databases, etc.).
- Inter-service communication has its own challenges.
- Data integrity and consistency.
- Network congestion and latency.



Distributed monoliths…

It’s built like microservices… but it’s tightly coupled with itself.

How to know it’s a distributed monolith:

- Requires low latency communication.
- Services don't scale easily.
- Dependency between services.
- Sharing the same resources such as databases.
- Tightly coupled systems.

=>most of kubernetes nowadays 

Service Oriented Architecture

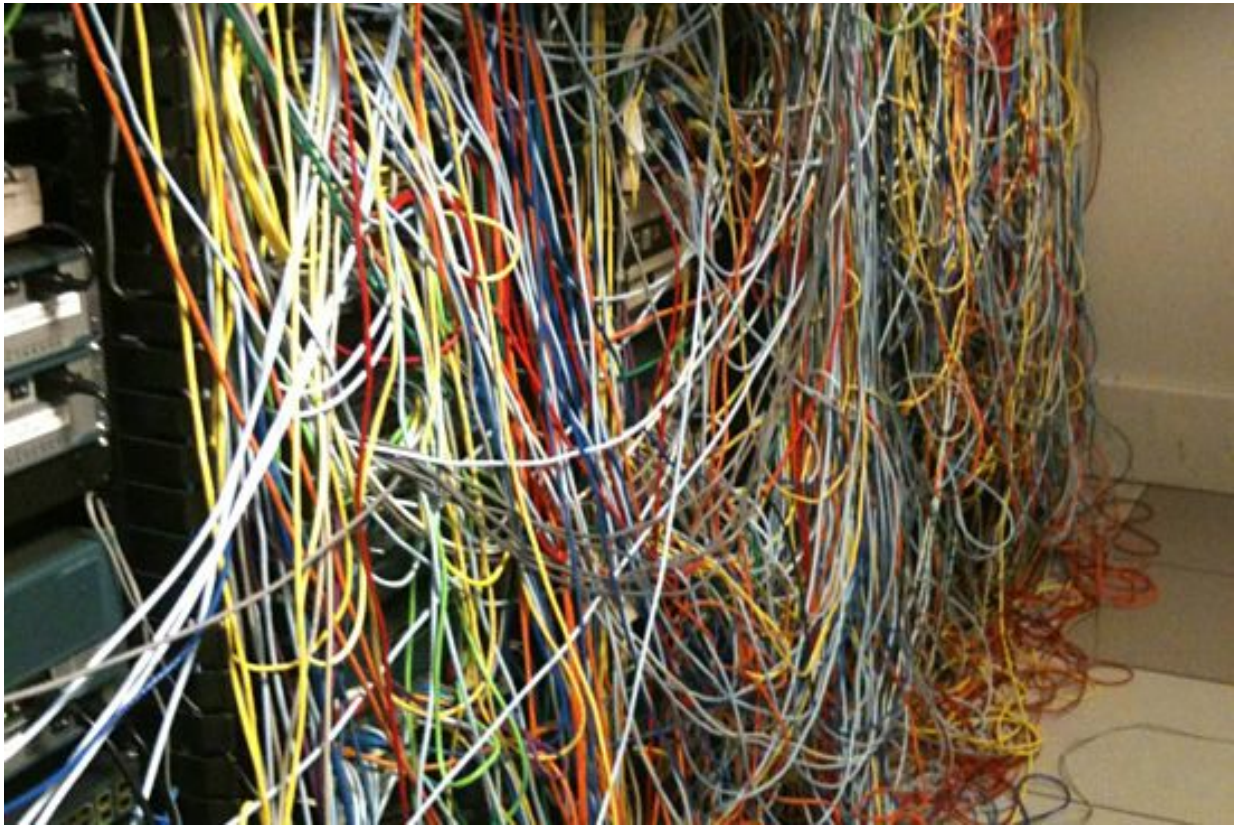
Software components are reusable via service interfaces…

Modular monoliths…

Trah tall3ouha hévvi !

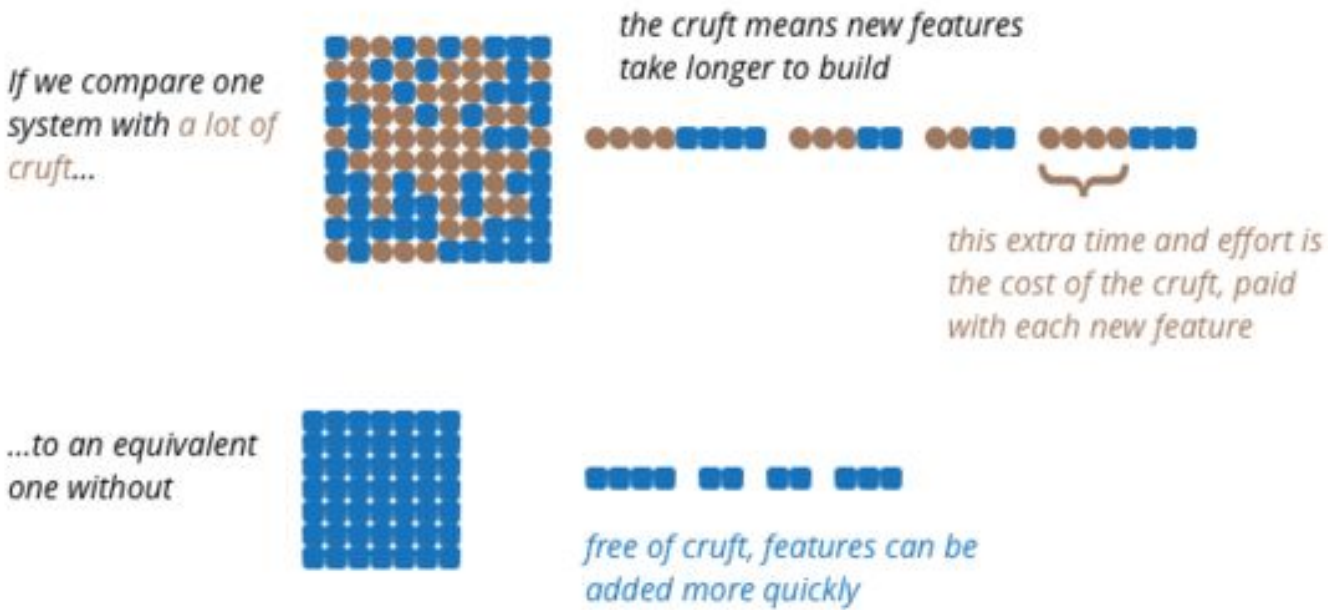
Wrap Up

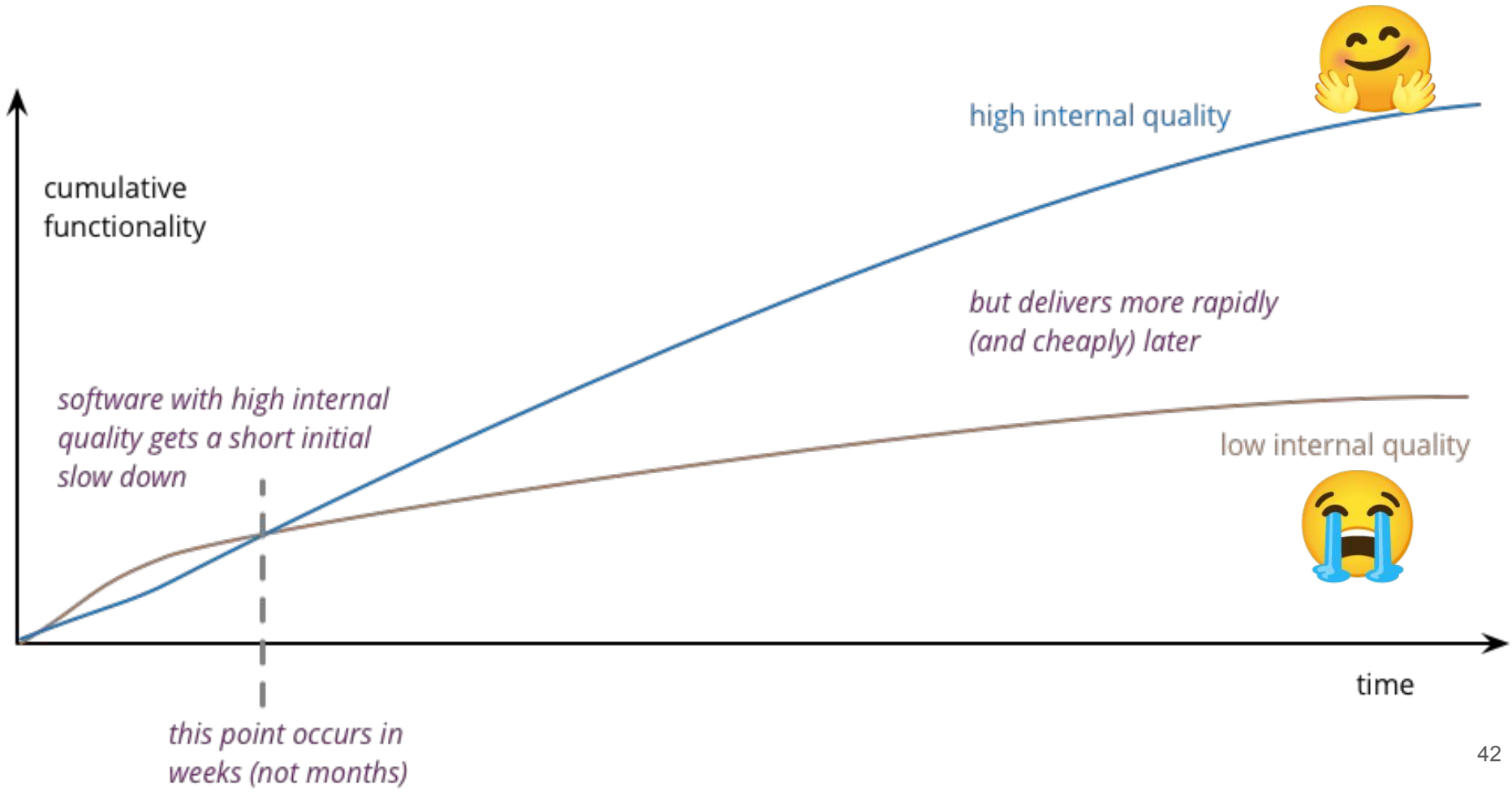
Architecture *mta3 3amm 3li*…

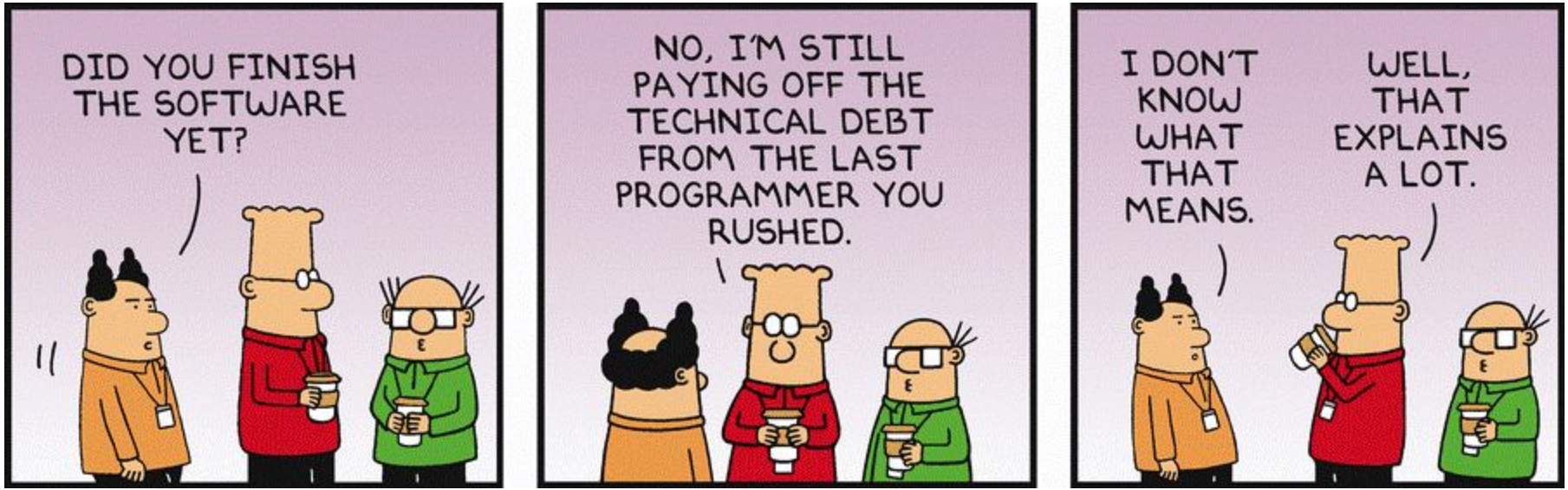


Internal Quality

Quality matters!



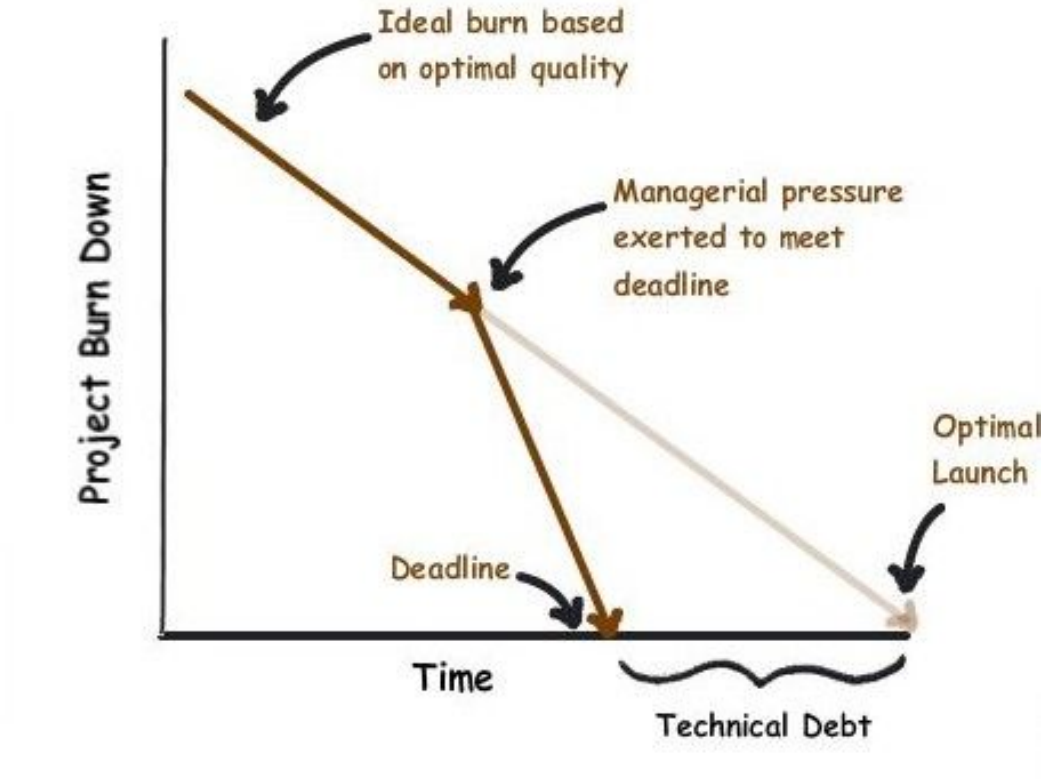

high internal quality
cumulative
functionality
but delivers more rapidly
(and cheaply) later
software with high internal
quality gets a short initial
slow down
low internal quality
time
this point occurs in
weeks (not months)

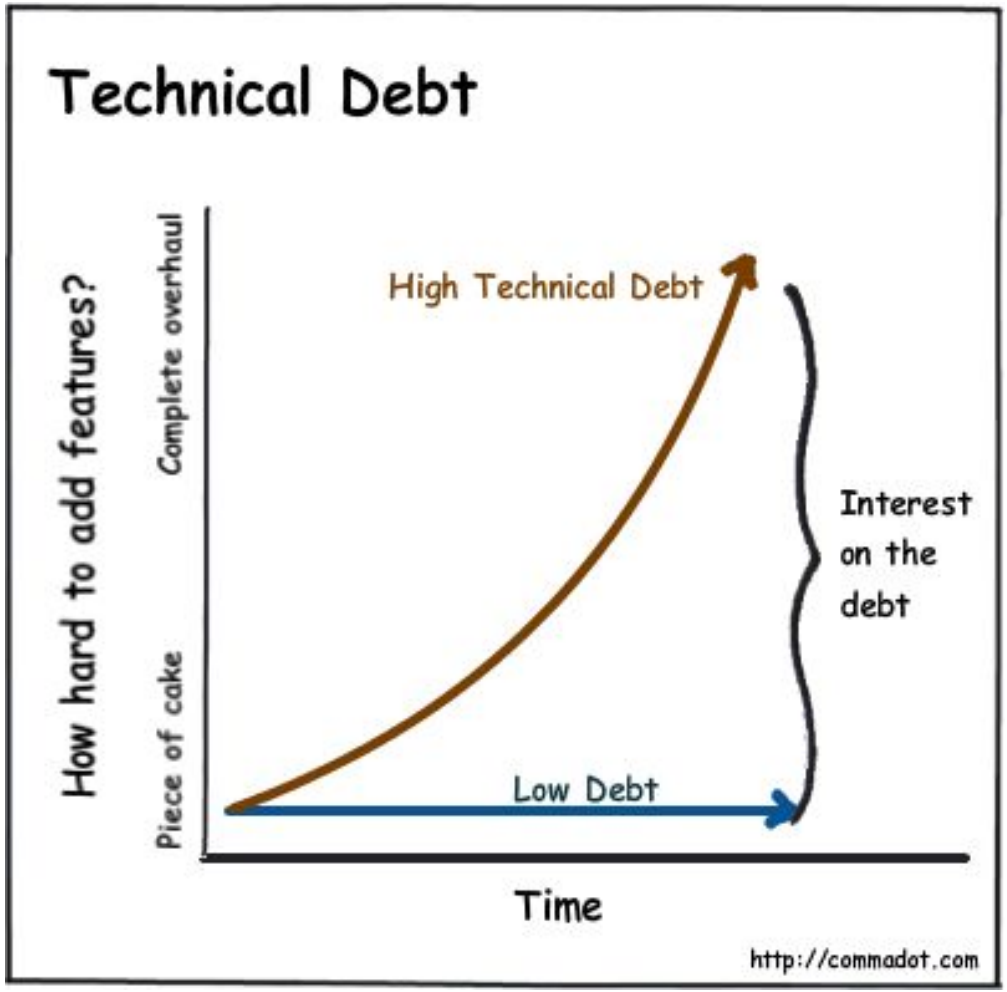

DID YOU FINISH THE SOFTWARE YET?
NO, I'M STILL PAYING OFF THE TECHNICAL DEBT FROM THE LAST PROGRAMMER YOU RUSHED.
I DON'T KNOW WHAT THAT MEANS.
WELL, THAT EXPLAINS A LOT.

What is technical debt?

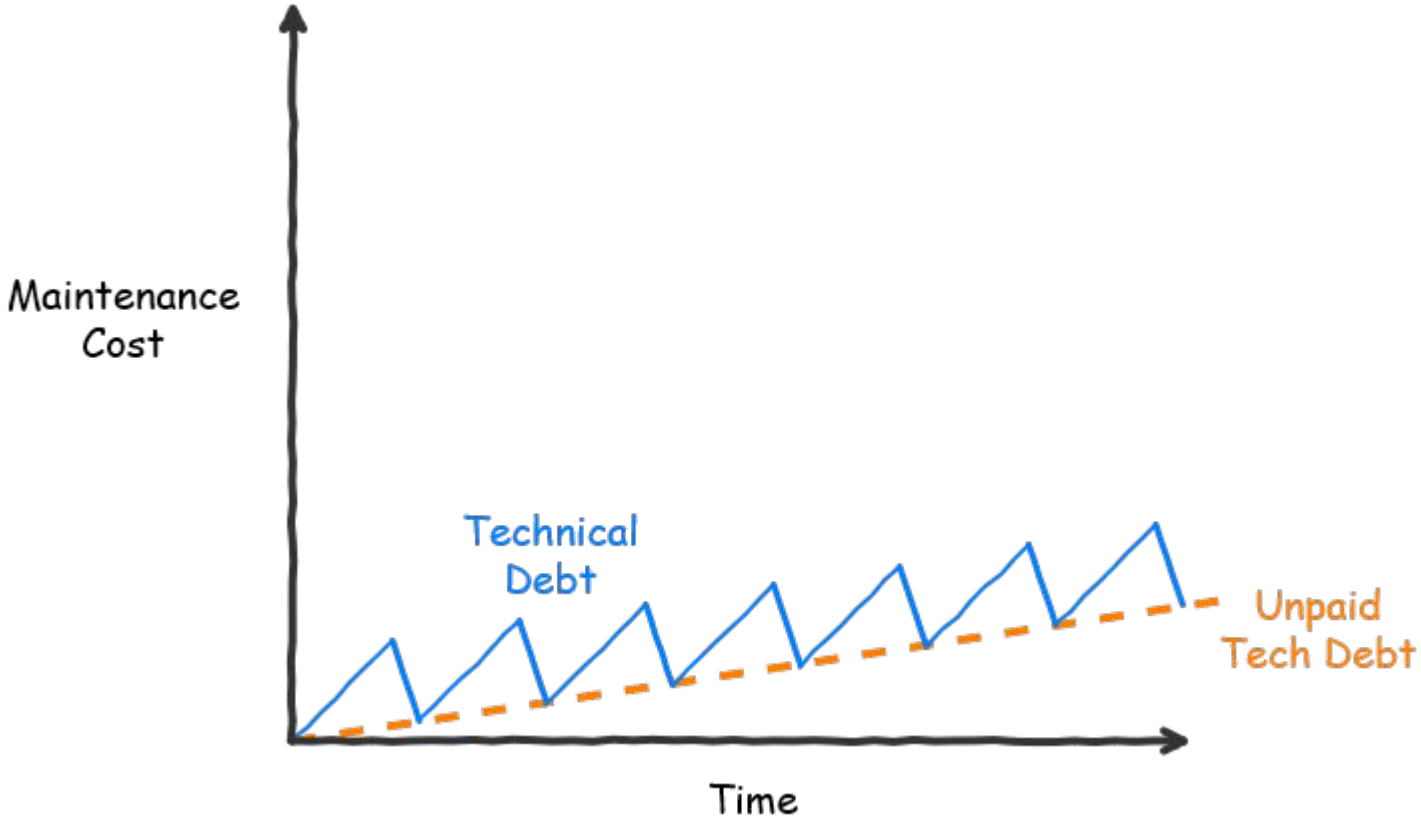
...and how to manage it

Technical Debt

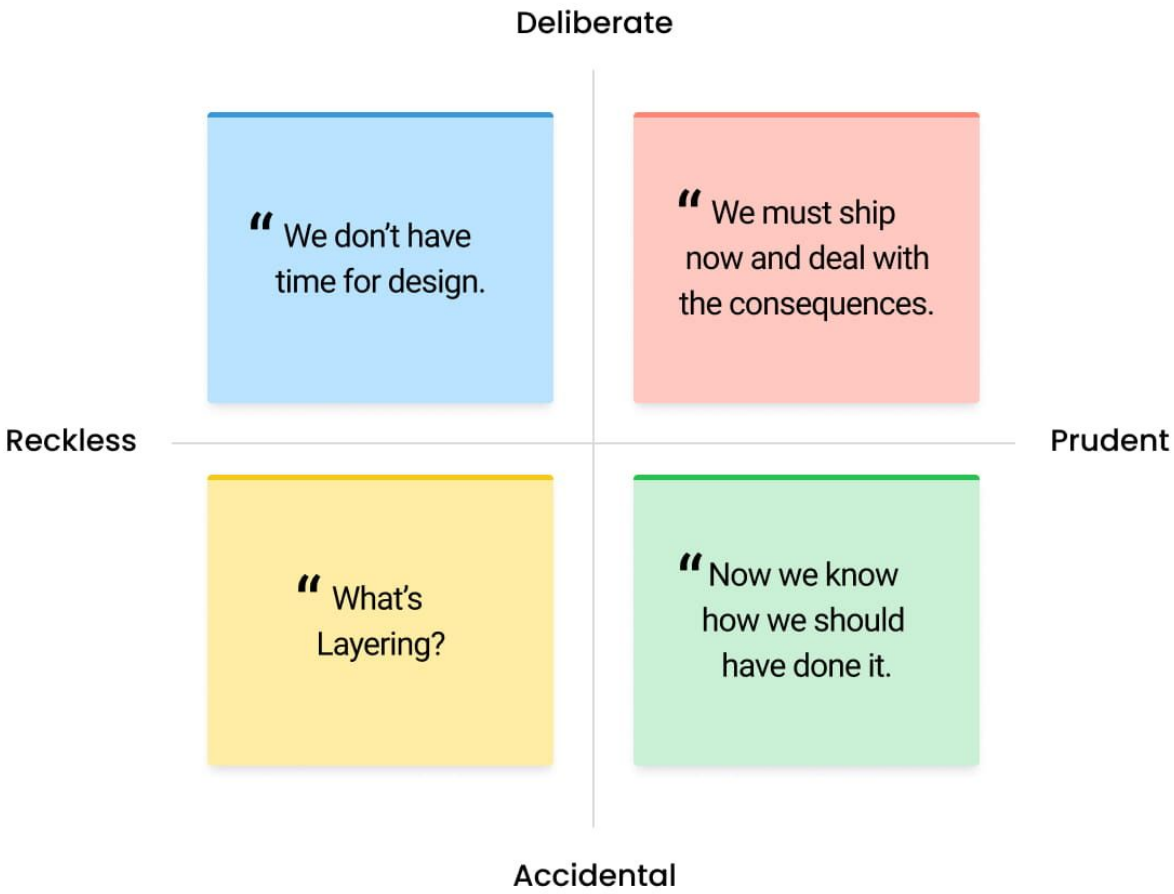



Technical Debt
How hard to add features?
Complete overhaul
Piece of cake
High Technical Debt
Interest on the debt
Low Debt
Time
http://commadot.com

We manage it this way…



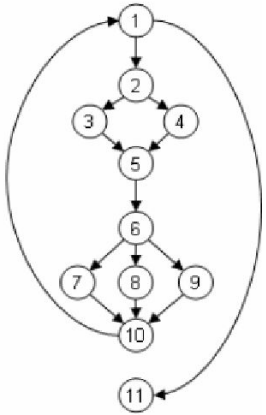
Technical Debt Quadrants



Debt comes from complexity…

Cyclomatic Complexity

Node	Statement
(1)	while(x<100){
(2)	if (a[x] % 2 == 0) {
(3)	parity = 0;
	}
	else {
(4)	parity = 1;
(5)	}
(6)	switch(parity){
	case 0:
(7)	println("a[" + i + "] is even");
	case 1:
(8)	println("a[" + i + "] is odd");
	default:
(9)	println("Unexpected error");
	}
(10)	x++;
	}
(11)	p = true;



Cognitive Complexity

```
function recordProduct (product, productCount, selectedProducts) {
  if (!productCount.has(product)) {
    selectedProducts.push(product);
    productCount.set(product, 1);
  } else {
    productCount.set(product, productCount.get(product) + 1);
  }
}

function printInFormat (productCount, selectedProducts) {
  const sortedSelectedProducts = selectedProducts.sort((p1, p2) => p1.barcode.localeCompare(p2.barcode));
  const totalPrice = selectedProducts.reduce(
    (totalprice, p) => totalprice + p.price * productCount.get(p),
    0);

  return '================== Receipt ==================\n' +
  (sortedSelectedProducts.length === 0
    ? '\n'
    : getLines(sortedSelectedProducts, productCount)) +
  '==================== Total ====================\n' +
  `${totalPrice.toFixed(2)}`;
}

function getLines (sortedSelectedProducts, productCount) {
  let resultString = '';
  for (const product of sortedSelectedProducts) {
    resultString += `${product.name.padEnd(30)}x${productCount.get(product).toString().padEnd(9)}${product.unit}\n`;
  }
  return resultString;
}
```

Application Architecture

What is a good application ? 🔍

- A body of code that's seen by developers as a single unit
- A group of functionality that customers see as a single unit
- An initiative that those with the money see as a single budget

How to make a good application ?

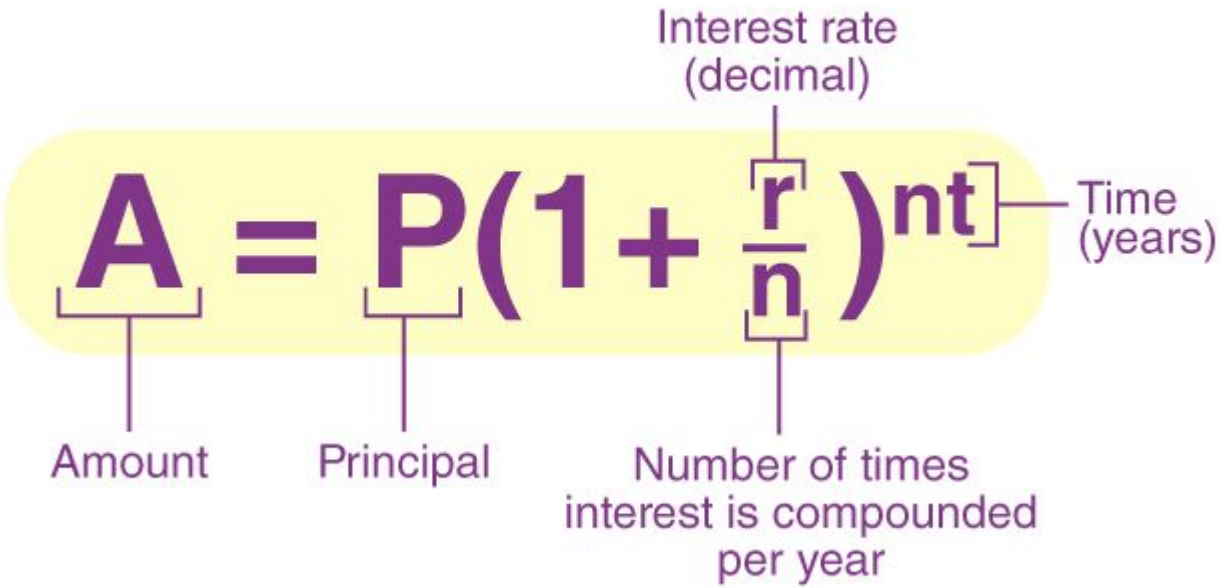
Universal Principles in life

Understand time : the 80/20 Rule

Universal Principles in life

The 80/20 Rule

Compound Interest



Universal Principles in life

The 80/20 Rule

Compound Interest

Optimism & curiosity

Universal Principles in life

The 80/20 Rule

Compound Interest

Optimism & curiosity

There is no magical tool… most of the quality is you…

Universal Principles in life

The 80/20 Rule

Compound Interest

Optimism & curiosity

There is no magical tool… most of the quality is you…

Following simple, good practices

Universal Principles in life

The 80/20 Rule

Compound Interest

Optimism & curiosity

There is no magical tool… most of the quality is you…

Following simple, good practices

Communication makes things easier

Computer stuff…

Look for answers into the documentation, then github issues, then specialised forums, then deep space, then youtube and then stack overflow.

Computer stuff…

Look for answers into the documentation, then github issues, then specialised forums, then deep space, then youtube and then stack overflow.

Make loosely coupled entities, with simple “piping”

Computer stuff…

Look for answers into the documentation, then github issues, then specialised forums, then deep space, then youtube and then stack overflow.

Make loosely coupled entities, with simple “piping”

If it takes too long, split it

Computer stuff…

Look for answers into the documentation, then github issues, then specialised forums, then deep space, then youtube and then stack overflow.

Make loosely coupled entities, with simple “piping”

If it takes too long, split it

Use standard ways (ex:git-flow for source control)

Computer stuff…

Look for answers into the documentation, then github issues, then specialised forums, then deep space, then youtube and then stack overflow.

Make loosely coupled entities, with simple “piping”

If it takes too long, split it

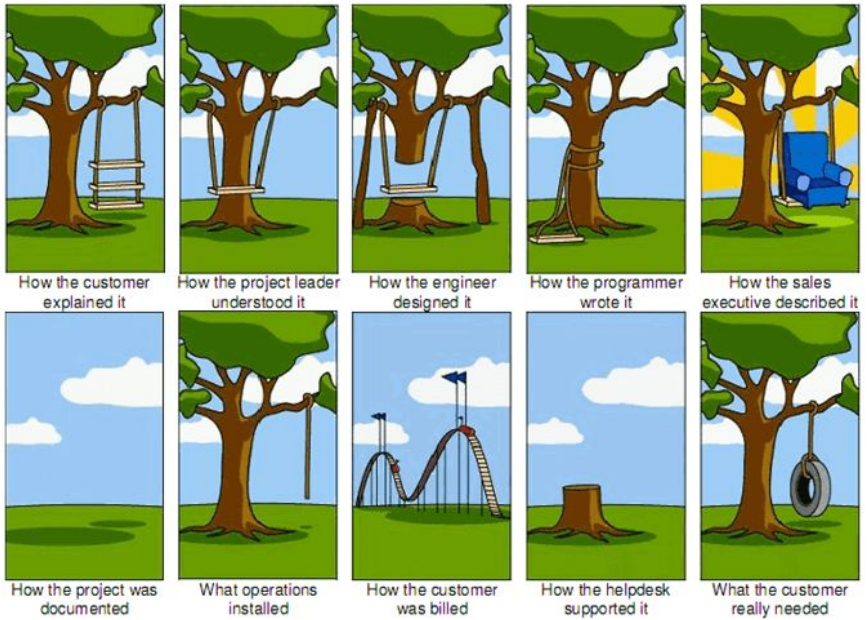
Use standard ways (ex:git-flow for source control)

Explain it to the rubber ducky (also known as alone-peer-programming)

Communication

We loose only get 10% of what has been said.

If unsure, prefer a written answer.



Quizz 

Imagine you review this…

```
if(userEndDate.getTime() != null
    && userEndDate.getTime() > today.getTime()
    && endDateFromRequest != null ? endDateFromRequest.getTime() !== userEndDate.getTime() : true){

            //some code
     }
```

```
this.onSubmit = this.onSubmit.bind(this)
this.onClose = this.onClose.bind(this)
```

```
function IsNumeric(sText) {
    var ValidChars = "0123456789.";
    var IsNumber = true;
    var Char;
    for (i = 0; i < sText.length && IsNumber == true; i++) {
        Char = sText.charAt(i);
        if (ValidChars.indexOf(Char) == -1) {
            IsNumber = false;
        }
    }
    return IsNumber;
}
```

Some Best Practices

What i personally do…


FRONT-END DEVELOPER
THE LIGHTBULB IS BROKEN
COVERS LIGHTBULB WITH YELLOW PAINT
TELLS BACKEND THAT LIGHTBULB WORKS, BUT ONLY DURING THE DAY
BACKEND DEVELOPER
WONDERS WHY EVERYBODY'S HAVING PROBLEMS WITH LIGHTBULBS
REALIZES HE FORGOT TO DEPLOY LIGHT TO DATABASE
KEEP CALM & BLAME FRONTEND
MARKETING
THE LIGHTBULB IS BROKEN
CONVINCES YOU THAT DARK IS THE NEW LIGHT
LEAVES YOU TO READ BLOG POSTS, GOES BACK TO PLAYING CANDY CRUSH
CONTENT MARKETING
THE LIGHTBULB IS BROKEN
HAS NO IDEA HOW TO FIX IT
WRITES A 2,000 WORD EXPERT'S GUIDE TO CHANGING A LIGHTBULB

In Nest JS

📑 Follow the docs

🤖 Use the command line tools to bootstrap your application (less errors, fast)

Keep controllers as thin as possible

Use services for business logic

Use pipes for input validation/control

🗂️ Organise your code into modules

Use DTOs

🦋 Keep your API beautiful

In Next JS

📑 Follow the docs

⚠️ No warnings in the browser console

♻️ Components should be reusable

Maintain clear folder structure

Keep it fast and responsive

Separate logic from rendering

In GUI programming [React]

The list is soo long that i need to write a document for that…

But you can check some in here **https://github.com/alan2207/bulletproof-react**

In JS in general

```
let a = 42;
```

```
let age = 42;
```

```
let wWidth = 640;
let w_height = 480;
```

```
let windowWidth = 640;
let windowHeight = 480;
```

```
const cdr = 700;
```

```
// The number of 700ms has been calculated empirically based on UX A/B test result
// @see: <link to experiment or to related JIRA task or to something that explains
const callbackDebounceRate = 700;
```

```
// t5abbi el modal sa7bi
toggleModal(false);
```

```
// Hide modal window on error.
toggleModal(false);
```

```
document.location.search.replace(/(^\?)/,'').split('&').reduce(function(o,n){n=n.s
```

```
document.location.search
  .replace(/(^\?)/, '')
  .split('&')
  .reduce((searchParams, keyValuePair) => {
    keyValuePair = keyValuePair.split('=');
    searchParams[keyValuePair[0]] = keyValuePair[1];
    return searchParams;
  },
  {}
)
```

```
try {
  // Something unpredictable.
} catch (error) {
  // tss... 🤫
}
```

```
try {
  // Something unpredictable.
} catch (error) {
  setErrorMessage(error.message);
  // and/or
  logError(error);
}
```

```
let x = 5;

function square() {
  x = x ** 2;
}

square(); // Now x is 25.
```

```
let x = 5;

function square(num) {
  return num ** 2;
}

x = square(x); // Now x is 25.
```

```
function sum(a, b) {
  return a + b;
}

// Having untyped fun here.
const guessWhat = sum([], {}); // -> "[object Object]"
const guessWhatAgain = sum({}, []); // -> 0
```

```
function sum(a: number, b: number): ?number {
  // Covering the case when we don't do transpilation and/or Flow type checks in J
  if (typeof a !== 'number' && typeof b !== 'number') {
    return undefined;
  }
  return a + b;
}

// This one should fail during the transpilation/compilation.
const guessWhat = sum([], {}); // -> undefined
```

```
function square(num) {
  if (typeof num === 'undefined') {
    return undefined;
  }
  else {
    return num ** 2;
  }
  return null; // This is my "Plan B".
}
```

```
function square(num) {
  if (typeof num === 'undefined') {
    return undefined;
  }
  return num ** 2;
}
```

```
function someFunction() {
  if (condition1) {
    if (condition2) {
      asyncFunction(params, (result) => {
        if (result) {
          for (;;) {
            if (condition3) {
            }
          }
        }
      })
    }
  }
}
```

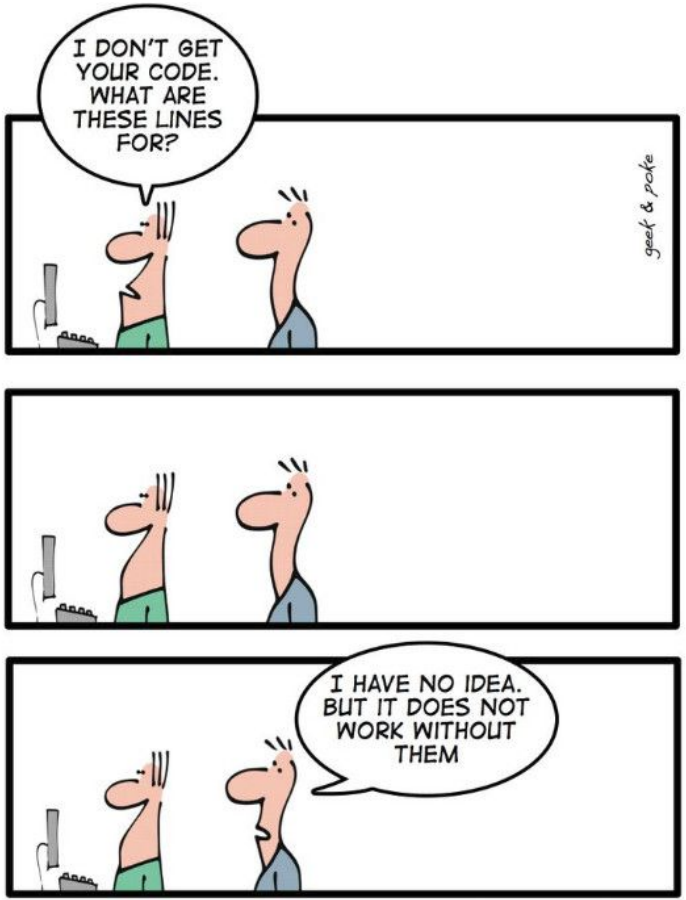
```
async function someFunction() {
  if (!condition1 || !condition2) {
    return;
  }

  const result = await asyncFunction(params);
  if (!result) {
    return;
  }

  for (;;) {
    if (condition3) {
    }
  }
}
```

```
const fruits = ['apple',
  'orange', 'grape', 'pineapple'];
  const toppings = ['syrup', 'cream',
                    'jam',
                    'chocolate'];
const desserts = [];
fruits.forEach(fruit => {
toppings.forEach(topping => {
    desserts.push([
fruit,topping]);
    });})
```

```
const fruits = ['apple', 'orange', 'grape', 'pineapple'];
const toppings = ['syrup', 'cream', 'jam', 'chocolate'];
const desserts = [];

fruits.forEach(fruit => {
  toppings.forEach(topping => {
    desserts.push([fruit, topping]);
  });
})
```

```
$ ls -la

package.json
```

```
$ ls -la

package.json
package-lock.json
```


I DON'T GET YOUR CODE. WHAT ARE THESE LINES FOR?
geek & poke
I HAVE NO IDEA. BUT IT DOES NOT WORK WITHOUT THEM

In Project Structure

🗂️ Structure your code by component

Layer your components

Use common utilities as npm packages

Use environment aware, hierarchical config

In Error Handling 🛑

Use Async-Await or promises for async error handling

Distinguish operational vs programmer errors

Document API errors using Swagger

Fail fast

Defencive programming

For code style 💃

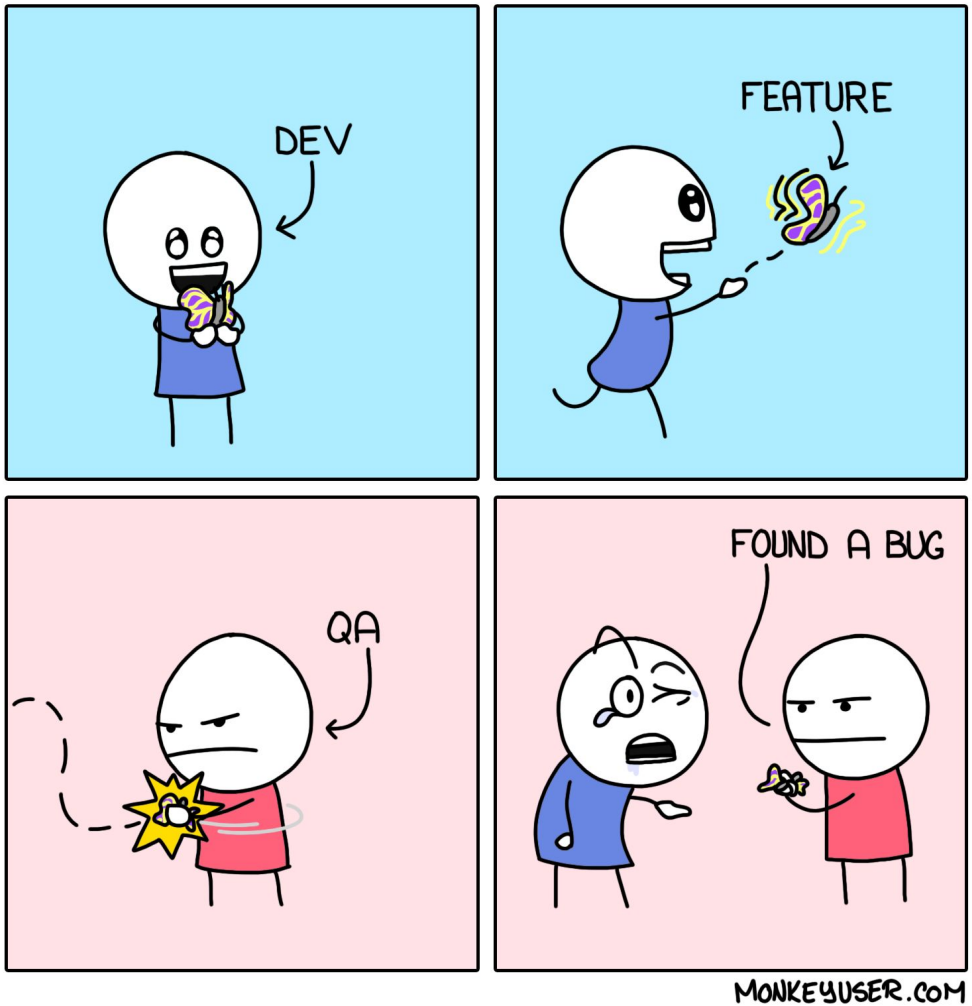
ESLint

Function naming is strategic (use naming conventions)

Never use var

Use Async Await, avoid callbacks

=> arrow functions are okay !


THE STRUGGLE
DEV
FEATURE
QA
FOUND A BUG
MONKEYUSER.COM

For Testing and overall Quality (next session)

At the very least: write API testing !

Test on average 3 things

Be aware of vulnerabilities in dependencies

Check your test coverage

Inspect for outdated packages

Refactor regularly

In Production (overview, not for you)

Monitoring

Smart Logging

Use reverse proxies

Lock dependencies

Monitor CPU and MEMORY usage of processes

Kill servers almost every day (to forget the state)

Have and emergency plan

Tools that will help you work efficiently

Git-flow

sonarLint

Bracket Pair Colorizer

GitLens

Better Comments

Import Cost

Markdown

Take time to set up your workspace

Further links

General mindset:

- https://martinfowler.com
- https://news.ycombinator.com (the comments)

Technology specific coding best practices

- https://dev.to
- https://github.com/airbnb/javascript

Download this presentation @ **https://bit.ly/3jwsowv**


Thank You