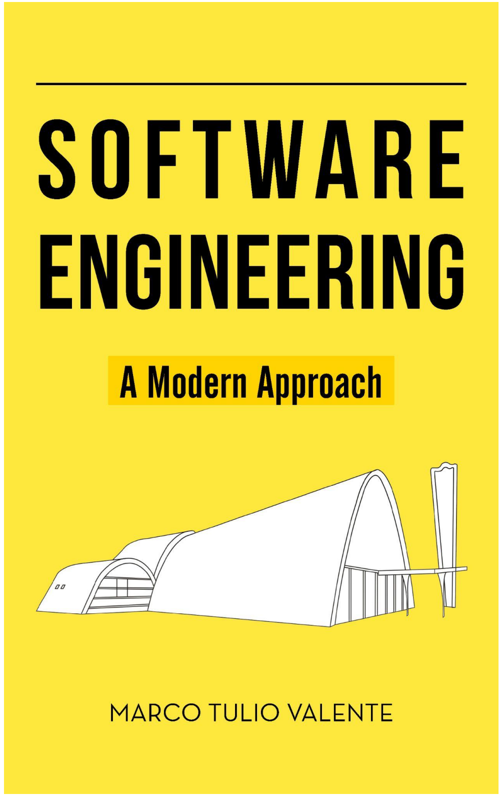


Chapter 7 - Architecture

Prof. Marco Tulio Valente

https://softengbook.org

Architecture is about the important stuff. Whatever that is. – Ralph Johnson

Architecture = high-level design

- The focus is no longer on small units (e.g., classes)
- But on larger and more relevant units
- Packages, modules, subsystems, layers, services, ...

Architectural Patterns = predefined architectures

- Layered
- Model-View-Controller (MVC)
- Microservices
- Message-Oriented
- Publish/Subscribe

Linus-Tanenbaum Debate (1992)

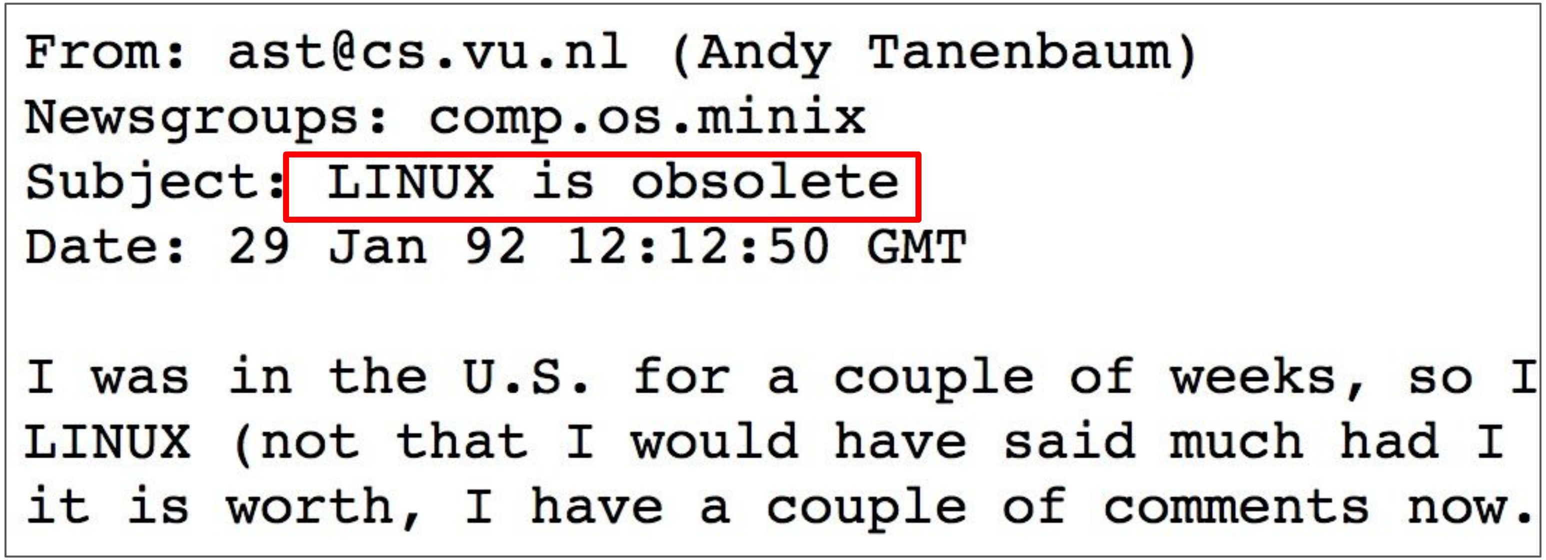
Linux

Minix OS

On the Importance of Software Architecture

https://www.oreilly.com/openbook/opensources/book/appa.html

Beginning of the debate: Tanenbaum's msg (1992)

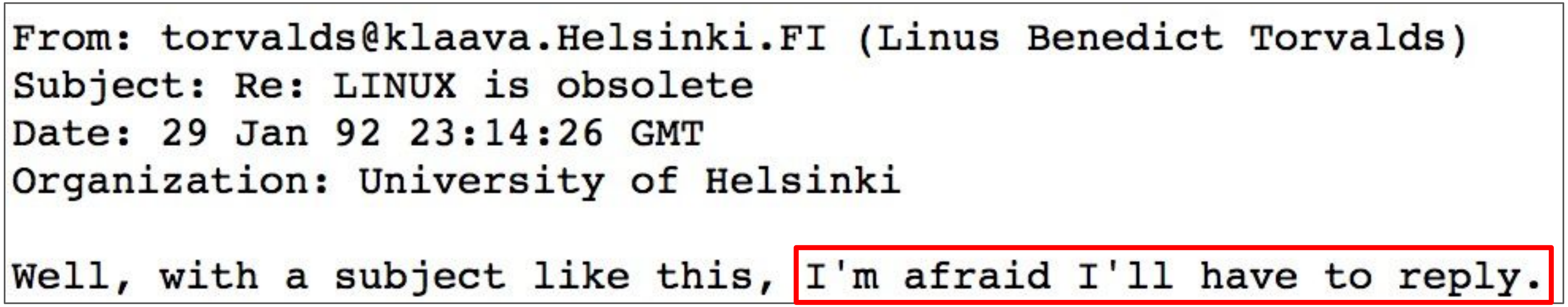


```
From: ast@cs.vu.nl (Andy Tanenbaum)
Newsgroups: comp.os.minix
Subject: LINUX is obsolete
Date: 29 Jan 92 12:12:50 GMT

I was in the U.S. for a couple of weeks, so I
LINUX (not that I would have said much had I
it is worth, I have a couple of comments now.
```

Tanenbaum's Argument

- Linux has a monolithic architecture
    - OS is a single executable file
    - Process management, memory, files, etc
- Microkernel architecture is better:
    - Kernel only contains essential services
    - Other services run as independent processes

Linus's Reply

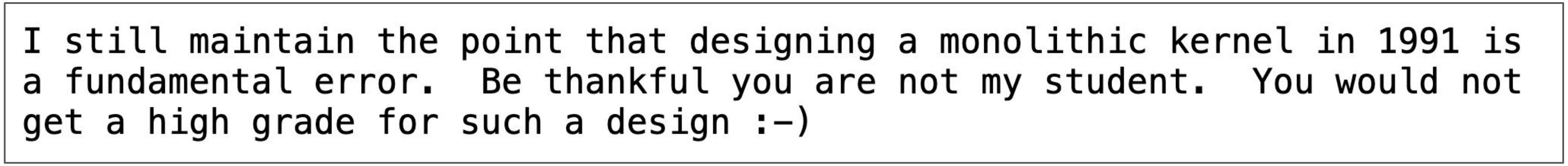


```
From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Subject: Re: LINUX is obsolete
Date: 29 Jan 92 23:14:26 GMT
Organization: University of Helsinki

Well, with a subject like this, I'm afraid I'll have to reply.
```

Linus's Argument

- In theory, microkernel architecture is more interesting
- But other criteria have to be considered
- For instance, Linux is a reality and not just a promise

New message from Tanenbaum

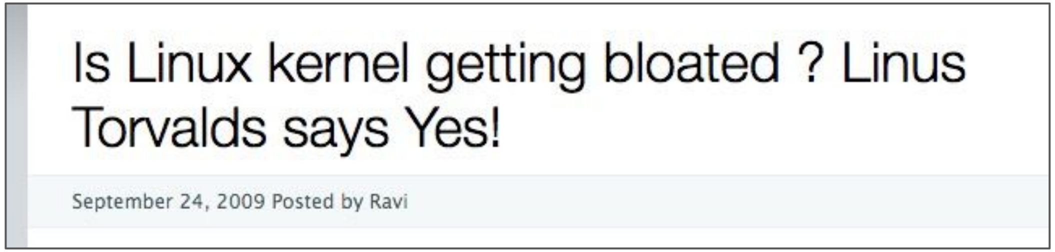


```
I still maintain the point that designing a monolithic kernel in 1991 is
a fundamental error.  Be thankful you are not my student.  You would not
get a high grade for such a design :-)
```

Comentário do Ken Thompson (Unix)

> I would generally agree that microkernels are probably the wave of the future. However, it is in my opinion easier to implement a monolithic kernel. It is also easier for it to turn into a mess in a hurry as it is modified.

Ken Thompson predicted the future: 17 years later (2009) see Torvalds' statement at a Linux conference


Is Linux kernel getting bloated ? Linus Torvalds says Yes!

September 24, 2009 Posted by Ravi

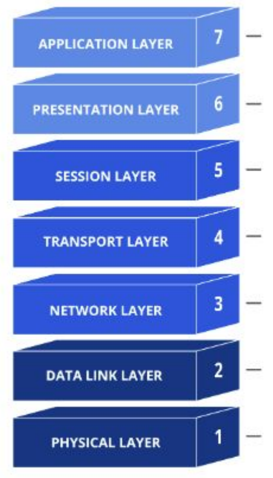
We are definitely not the streamlined, small, hyper-efficient kernel that I envisioned 15 years ago. The kernel is huge and bloated… And whenever we add a new feature, it only gets worse.

Takeaway: the costs of architectural decisions can take years to appear…

Layered Architecture

Layered Architecture

- System is organized in a hierarchical way
- Layer *n* can only use services from layer *n-1*
- Widely used in networks and distributed systems



Advantages: divide and conquer

- Breaks down system complexity and facilitates:
    - Understanding
    - Layer exchange (e.g., TCP to UDP)
    - Layer reuse (e.g., multiple apps use TCP)

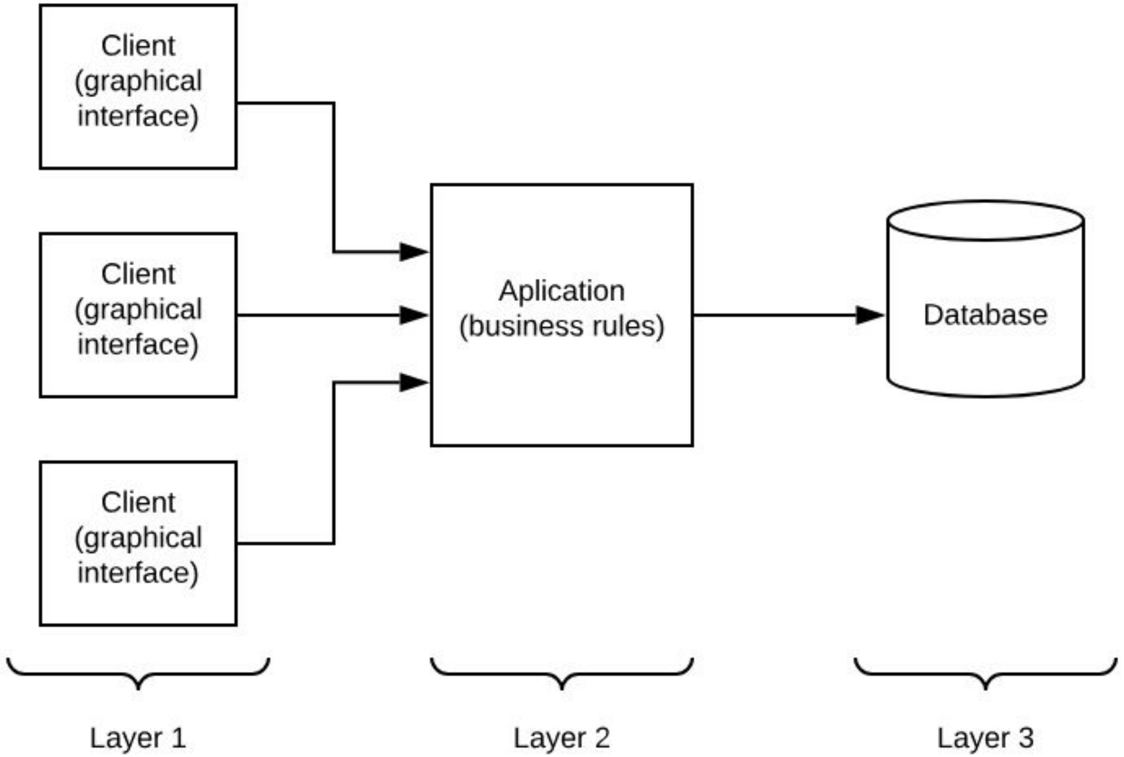
Variations

- Three-Tier Architecture
- Two-Tier Architecture

Three-Tier Architecture

- Common when downsizing enterprise apps in 80s and 90s
- Downsizing: migration from mainframes to Unix servers

Three-Tier Architecture


Client (graphical interface)
Client (graphical interface)
Client (graphical interface)
Aplication (business rules)
Database
Layer 1
Layer 2
Layer 3

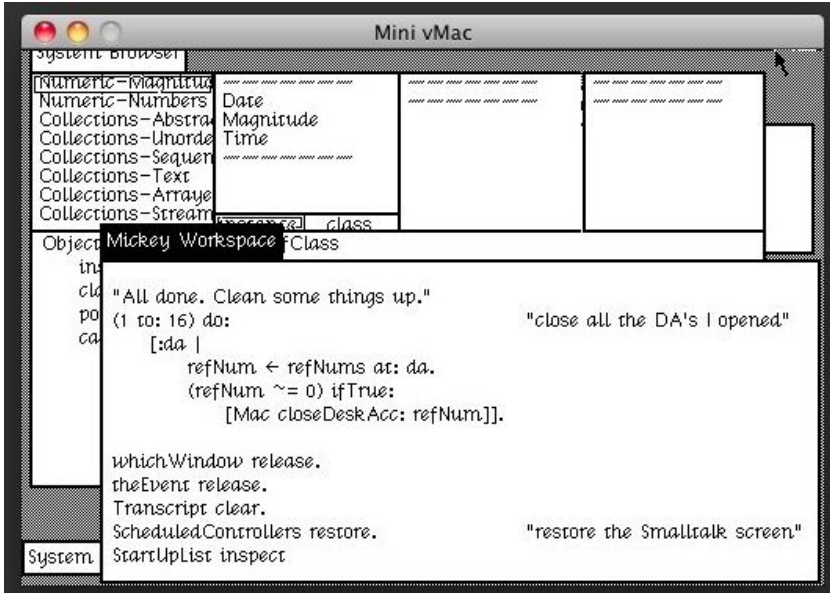
Two-Tier Architecture

- Simpler:
  - Tier 1: client (interface + logic)
  - Tier 2: database server
- Disadvantage: all processing is done on the client

Model-View-Controller (MVC)

MVC Architecture

- Introduced in the 1980s, with Smalltalk
- To implement graphical interfaces (GUIs)



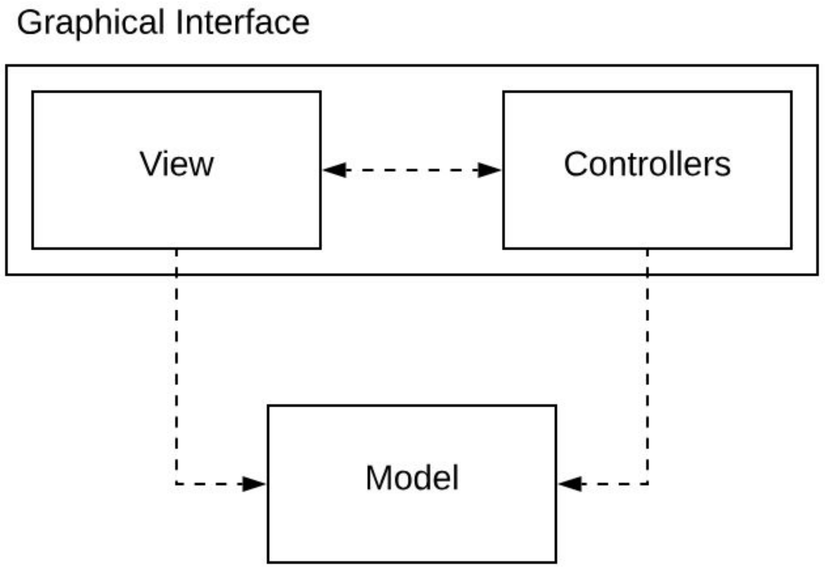
```
"All done. Clean some things up."
(1 to: 16) do:                              "close all the DA's I opened"
    [:da |
        refNum ← refNums at: da.
        (refNum ~= 0) ifTrue:
            [Mac closeDeskAcc: refNum]].

whichWindow release.
theEvent release.
Transcript clear.
ScheduledControllers restore.               "restore the Smalltalk screen"
StartUpList inspect
```

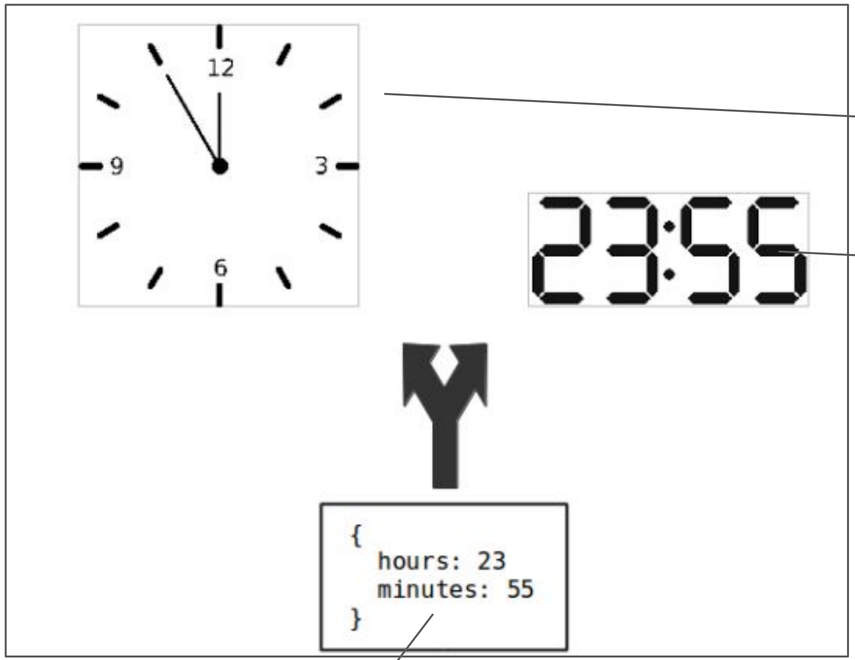
MVC divides classes into 3 groups

- View: classes for implementing GUIs, like windows, buttons, menus, scroll bars, etc
- Control: classes that handle events produced by input devices such as mouse and keyboard
- Model: classes with application logic and data

MVC = (View + Controllers) + Model

= Graphical Interface + Model





```
{
  hours: 23
  minutes: 55
}
```

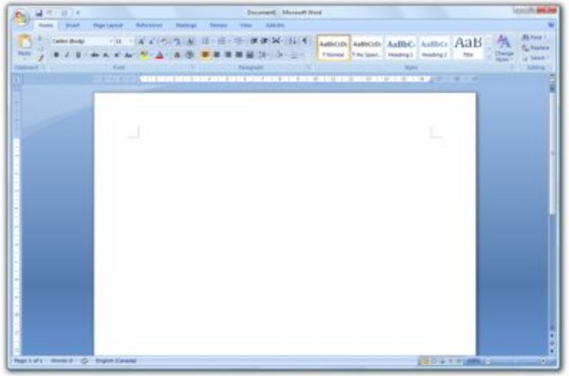
GUI #1

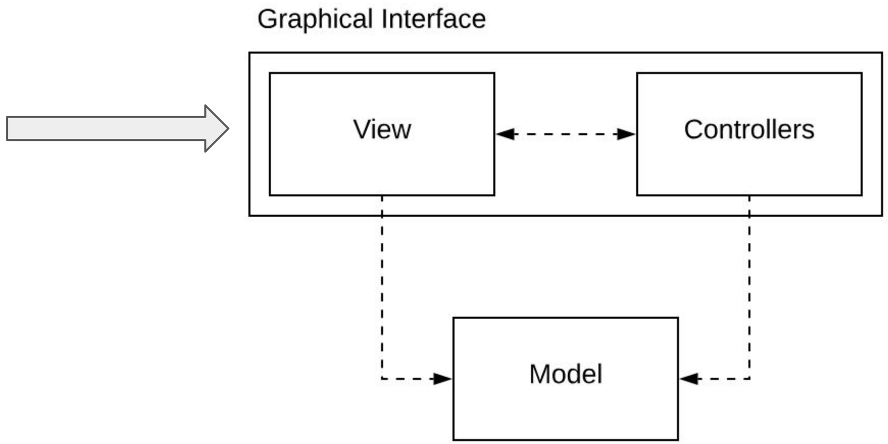
GUI #2

Model

Traditional MVC apps

- MVC was designed for desktop applications
- Example: Microsoft Word, Google Chrome, etc





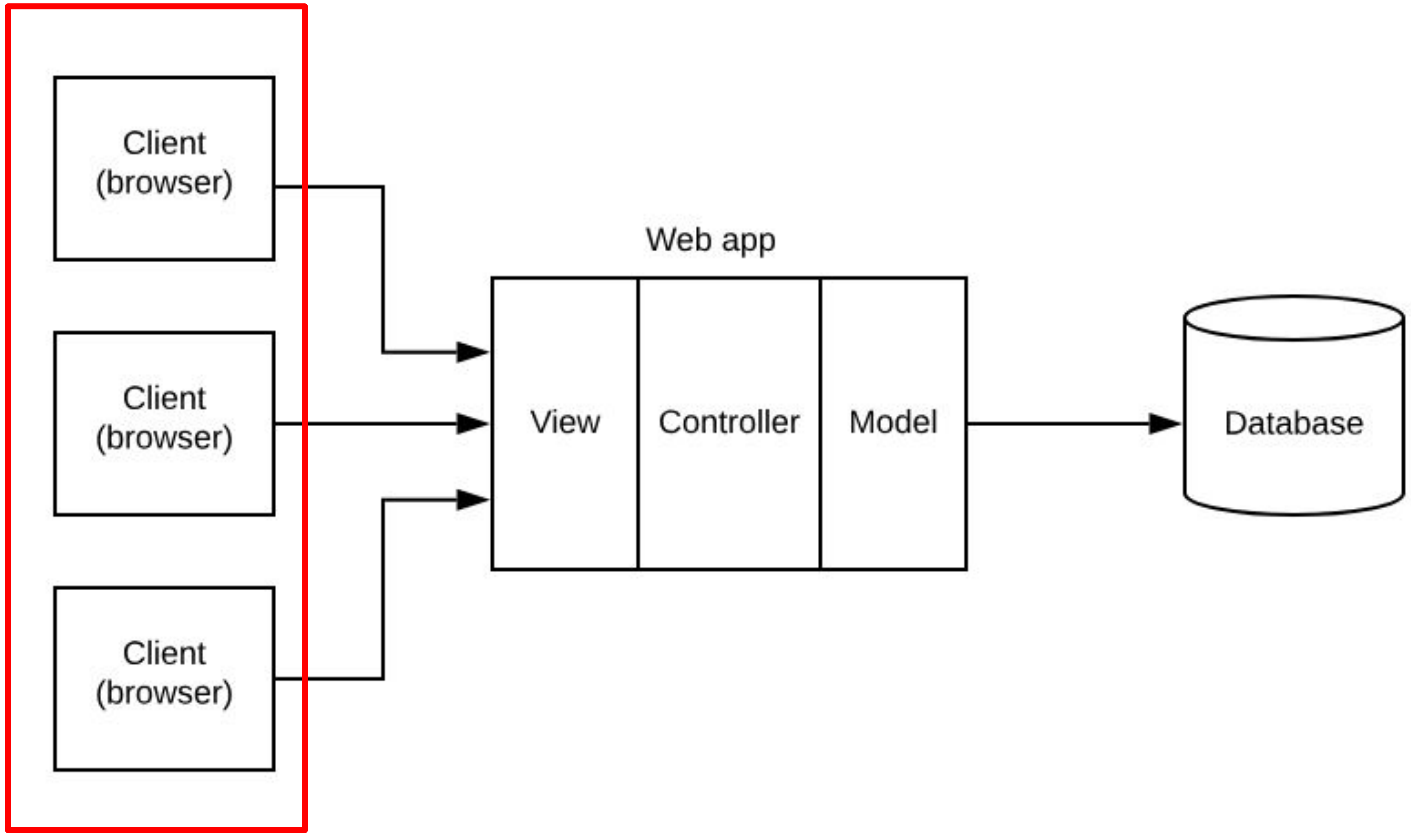
MVC Today

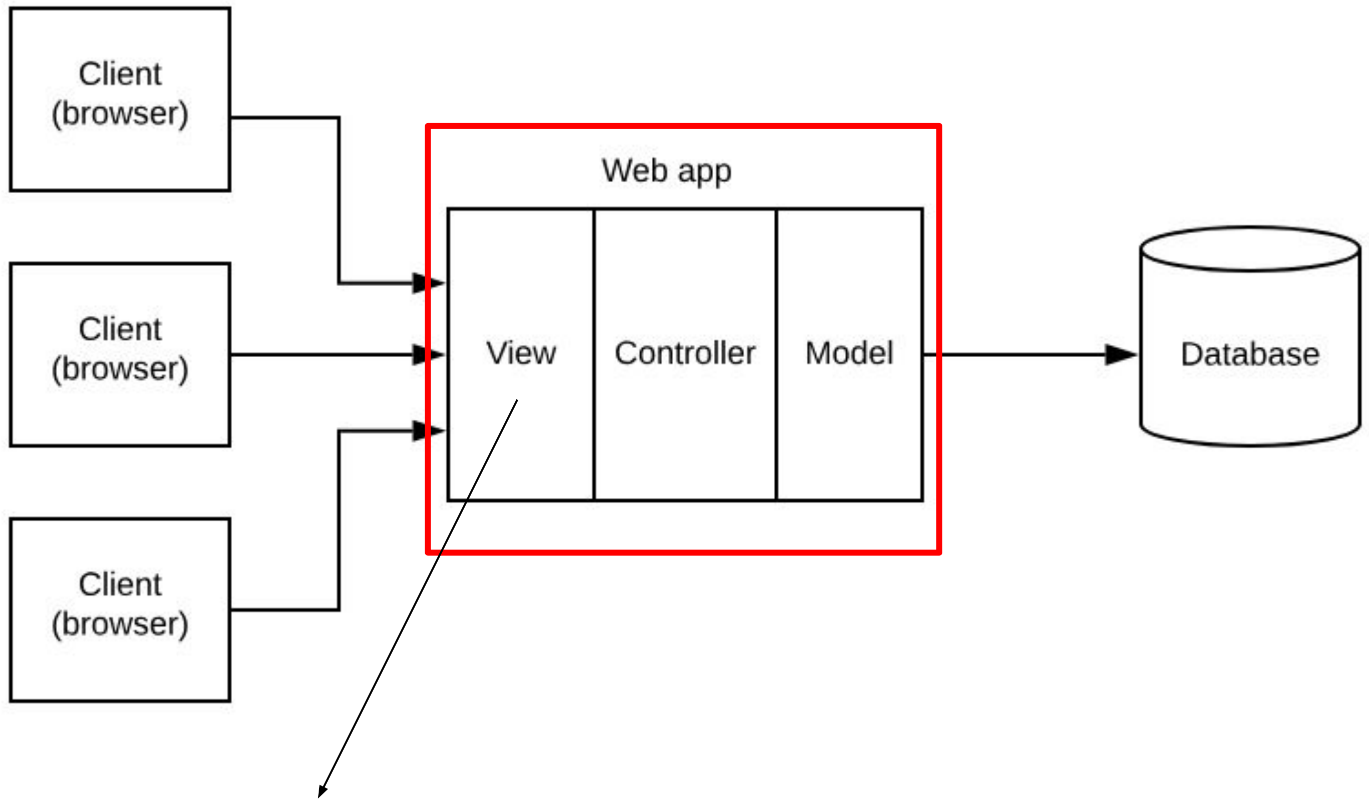
- MVC Web
- Single Page Applications

MVC Web

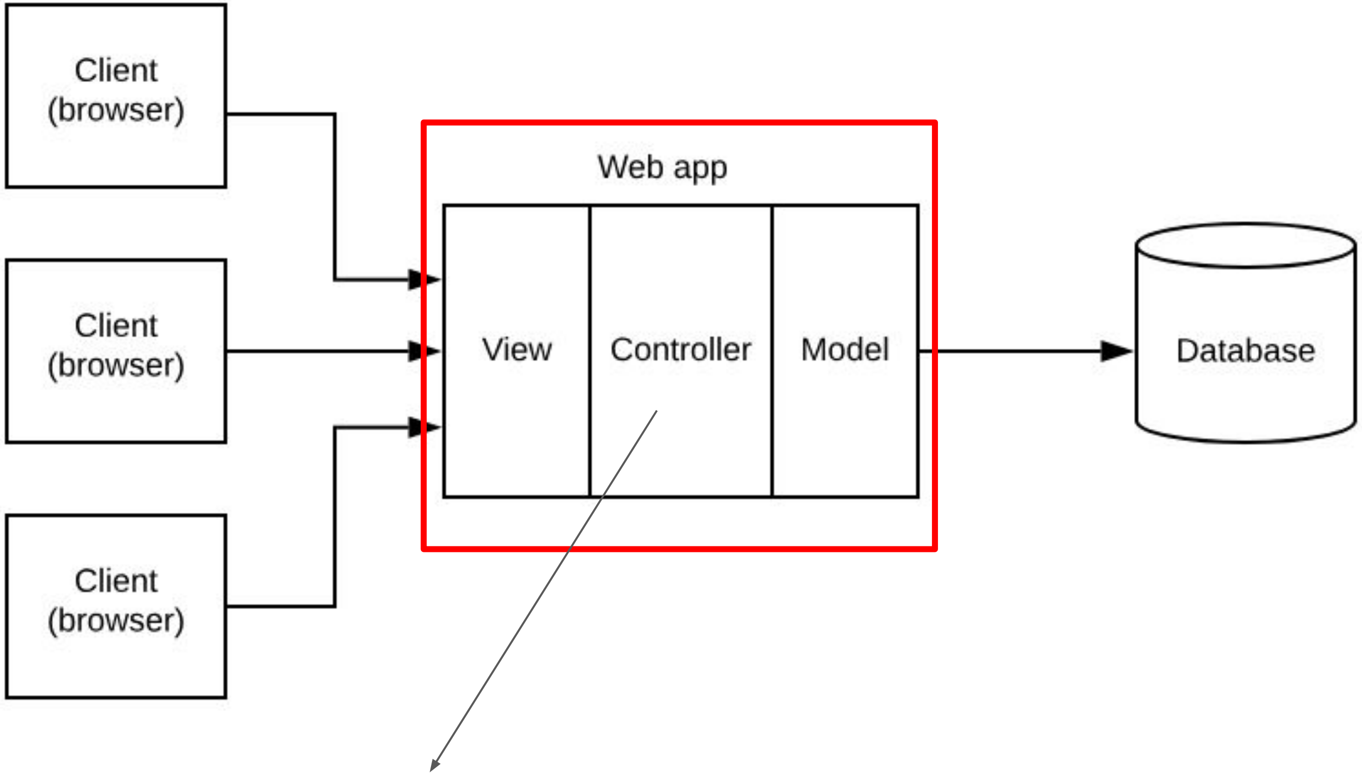
MVC Web

- MVC adaptation for the Web
- Ruby on Rails, DJango, Spring, PHP Laravel, etc

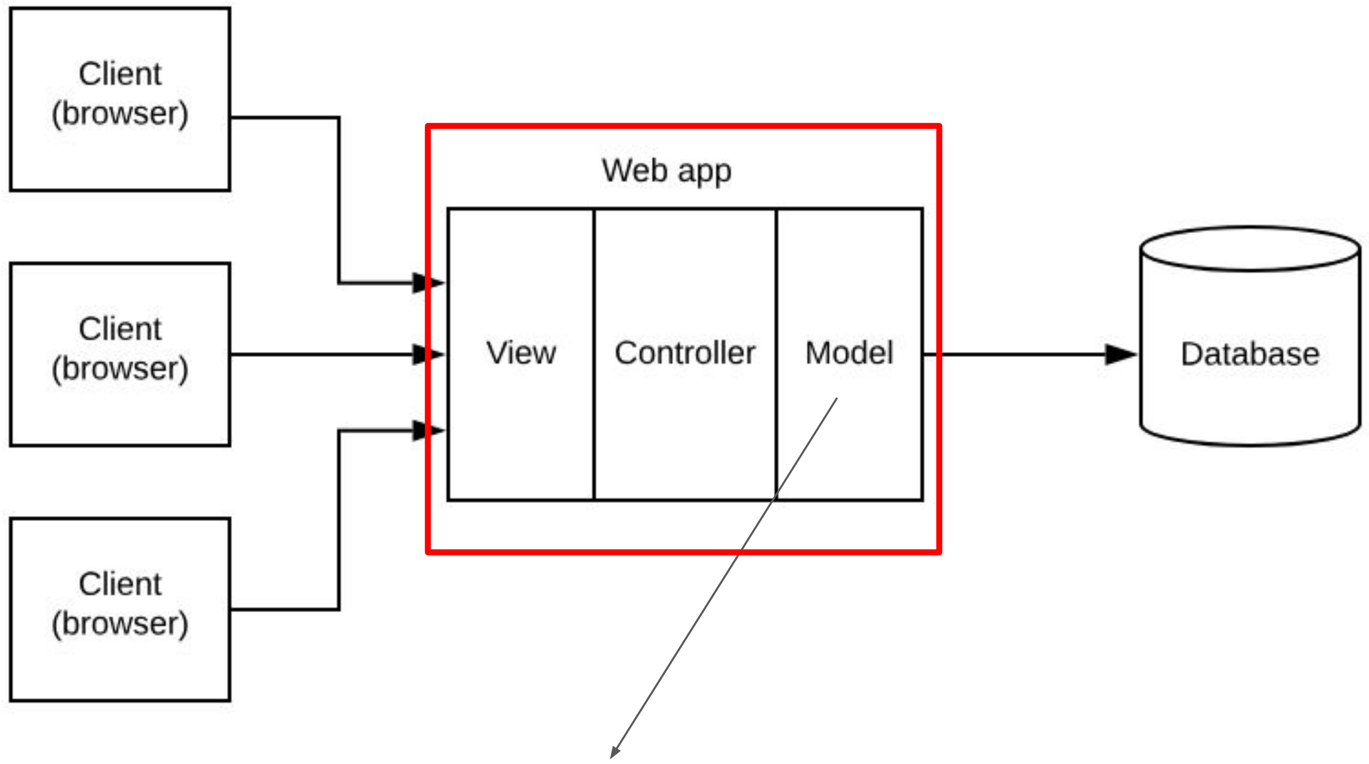

Client (browser)
Client (browser)
Client (browser)
Web app
View
Controller
Model
Database



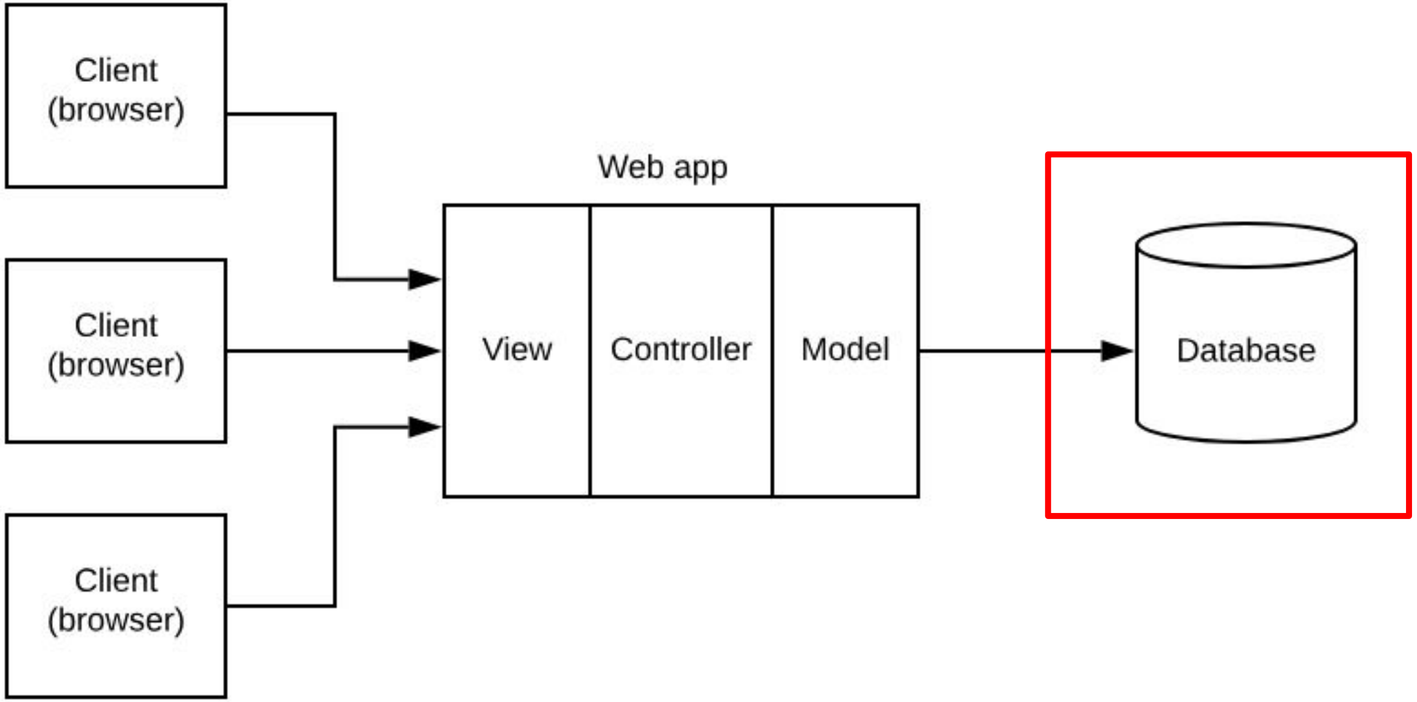
HTML, CSS, JavaScript pages
(what the user sees)



Receive input data and provide information to generate output page



Application logic (business rules) and interface with the database


Client (browser)
Client (browser)
Client (browser)
Web app
View
Controller
Model
Database

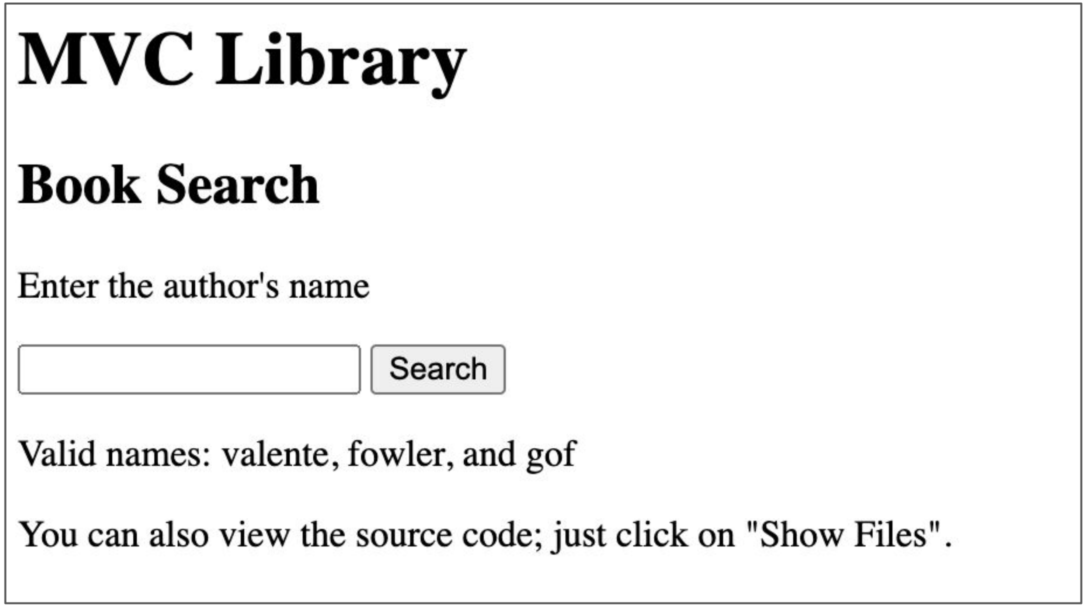
A simple Web-based MVC system

This example does not use any framework and has a very simple web interface, only for didactic purposes.

Controller

```
public class BookSearchController {
  ...
  public void start() {
    ...
    get("/", (req, res) -> {
      res.redirect("index.html");
      return null;
    });
    ...
  }
}
```

Browser (index.html)


MVC Library

Book Search

Enter the author's name

Valid names: valente, fowler, and gof

You can also view the source code; just click on "Show Files".

Controller

```
public class BookSearchController {
  BookSearchService searchService;
  BookPage bookPage;
  ...
  public void start() {
    ...
    get("/search", (req, res) -> {
      String author = req.queryParams("author");
      Book book = searchService.searchByAuthor(author);
      return bookPage.displayBook(book.getTitle(),
                                  book.getAuthor(),
                                  book.getISBN());
    });
    ...
  }
}
```

Model

```
public class BookSearchService {

  public Book searchByAuthor(String author) {
    try (Connection con = DriverManager.getConnection(...)) {
      String query = "SELECT * FROM books WHERE author = ?";
      PreparedStatement stmt = con.prepareStatement(query);
      stmt.setString(1, author);
      ResultSet rs = stmt.executeQuery();
      String isbn = rs.getString("isbn");
      String title = rs.getString("title");
      return new Book(isbn, author, title);
    } catch (SQLException e) {
      System.out.println(e.getMessage());
      return null;
    }
  }
}
```

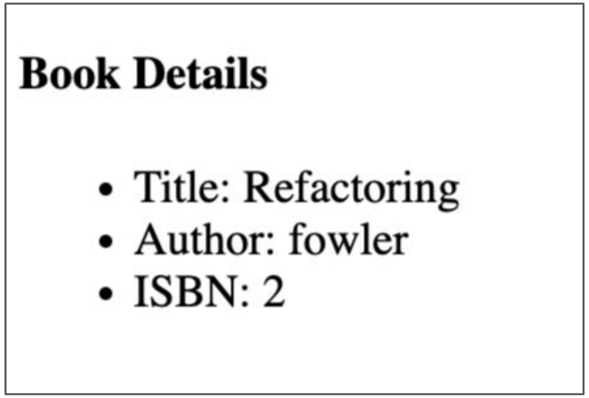
Controller

```
public class BookSearchController {
  BookSearchService searchService;
  BookPage bookPage;
  ...
  public void start() {
    ...
    get("/search", (req, res) -> {
      String author = req.queryParams("author");
      Book book = searchService.searchByAuthor(author);
      return bookPage.displayBook(book.getTitle(),
                                  book.getAuthor(),
                                  book.getISBN());

    });
    ...
  }
}
```

View

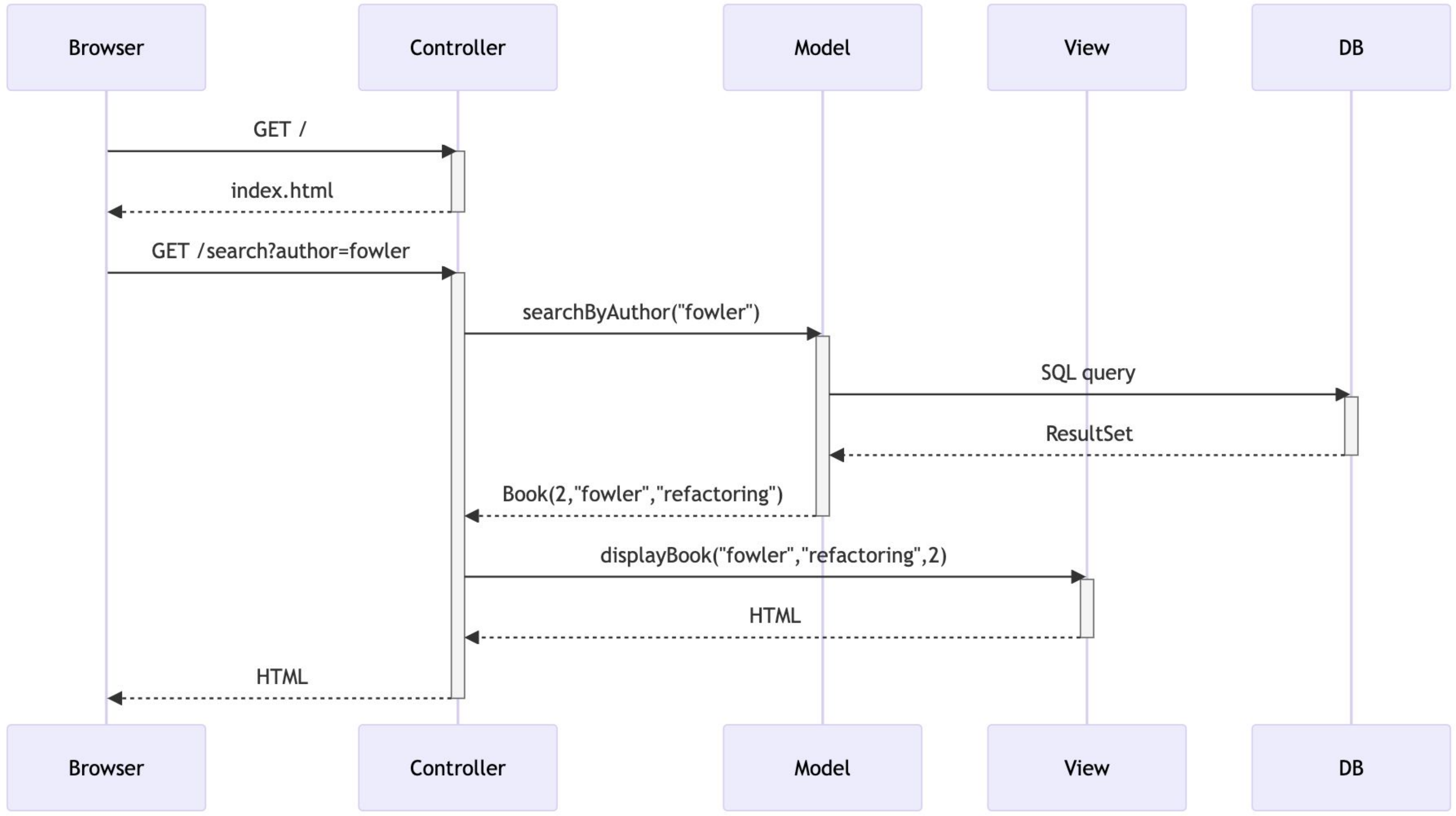
```
public class BookPage {

  public String displayBook(String title, String author, String isbn) {
    String res = "<h4> Book Details </h4>";
    res += "<ul>";
    res += "<li> Title: " + title + " </li>";
    res += "<li> Author: " + author + " </li>";
    res += "<li> ISBN: " + isbn + " </li>";
    res += "</ul>";
    return res;
  }
}
```

Browser

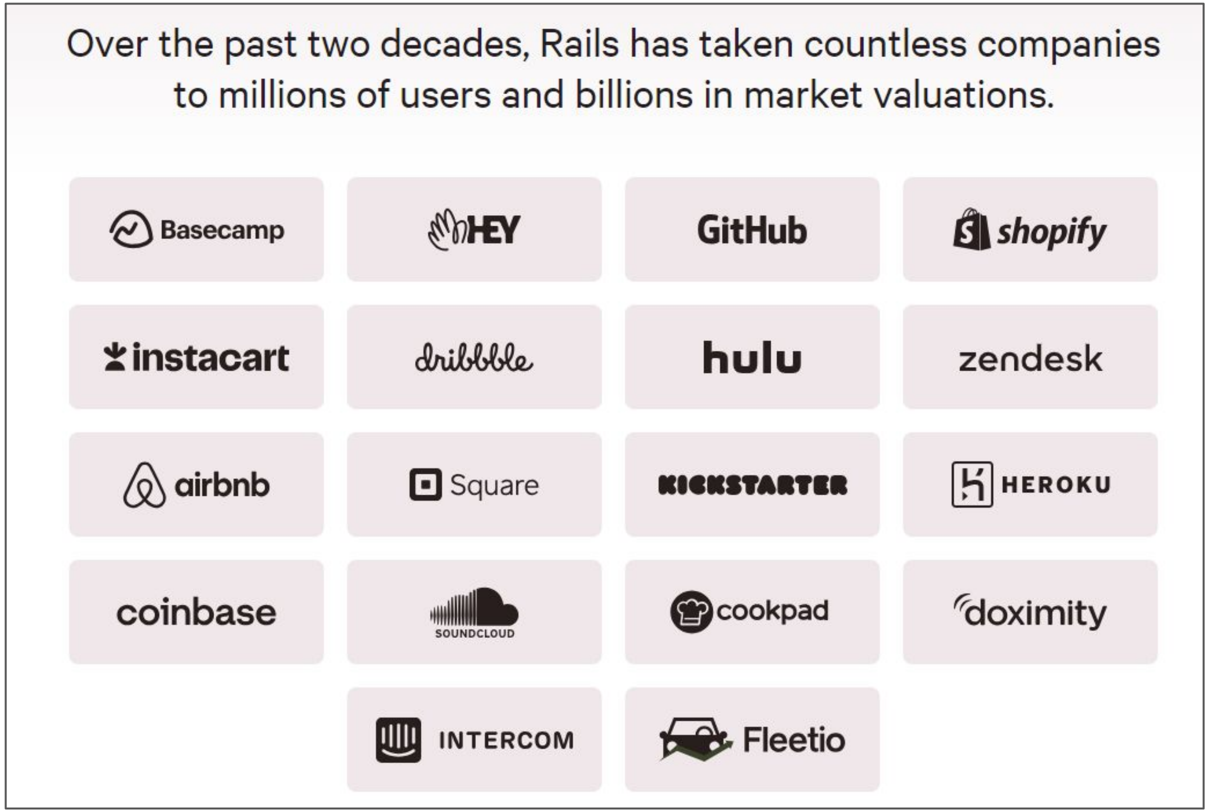


Book Details

- Title: Refactoring
- Author: fowler
- ISBN: 2


Browser
Controller
Model
View
DB
GET /
index.html
GET /search?author=fowler
searchByAuthor("fowler")
SQL query
ResultSet
Book(2,"fowler","refactoring")
displayBook("fowler","refactoring",2)
HTML
HTML
Browser
Controller
Model
View
DB

MVC Frameworks remain relevant

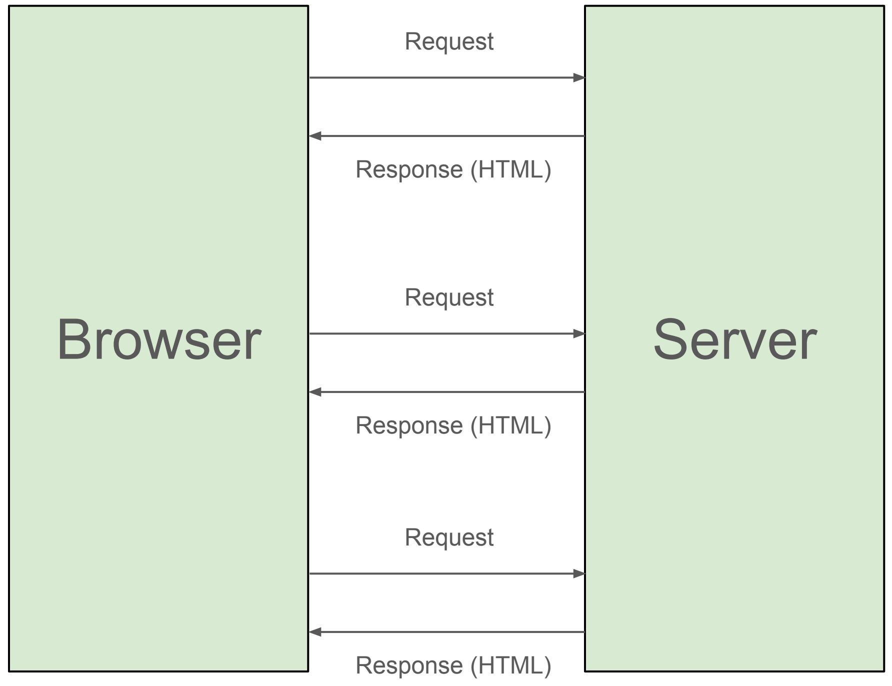


Over the past two decades, Rails has taken countless companies to millions of users and billions in market valuations.

https://rubyonrails.org (April 2024)

Single Page Applications (SPAs)

Traditional Web Apps



Multiple Page Applications

Problem: less responsive interfaces

Single Page Applications

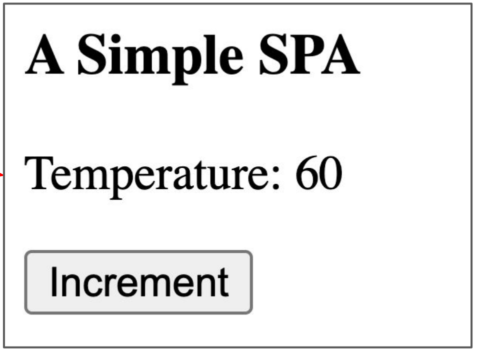
- Run in the browser, but is more independent of the server
    - Manipulates its own interface
    - Stores its data
    - Access the server to fetch more data
- Example: GMail, Google Docs, Facebook, Figma, etc
- Implemented in JavaScript

Simple Application using Vue.js

```
<h3>A Simple SPA</h3>

<div id="ui">
  Temperature: {{ temperature }}
  <p><button v-on:click="incTemperature">Increment
  </button></p>
</div>

<script>
var model = new Vue({
  el: '#ui',
  data: {
    temperature: 60
  },
  methods: {
    incTemperature: function() {
      this.temperature++;
    }
  }
})
</script>
```

Interface (Web, HTML)

```
<h3>A Simple SPA</h3>

<div id="ui">
  Temperature: {{ temperature }}
  <p><button v-on:click="incTemperature">Increment
  </button></p>
</div>

<script>
var model = new Vue({
  el: '#ui',
  data: {
    temperature: 60
  },
  methods: {
    incTemperature: function() {
      this.temperature++;
    }
  }
})
</script>
```

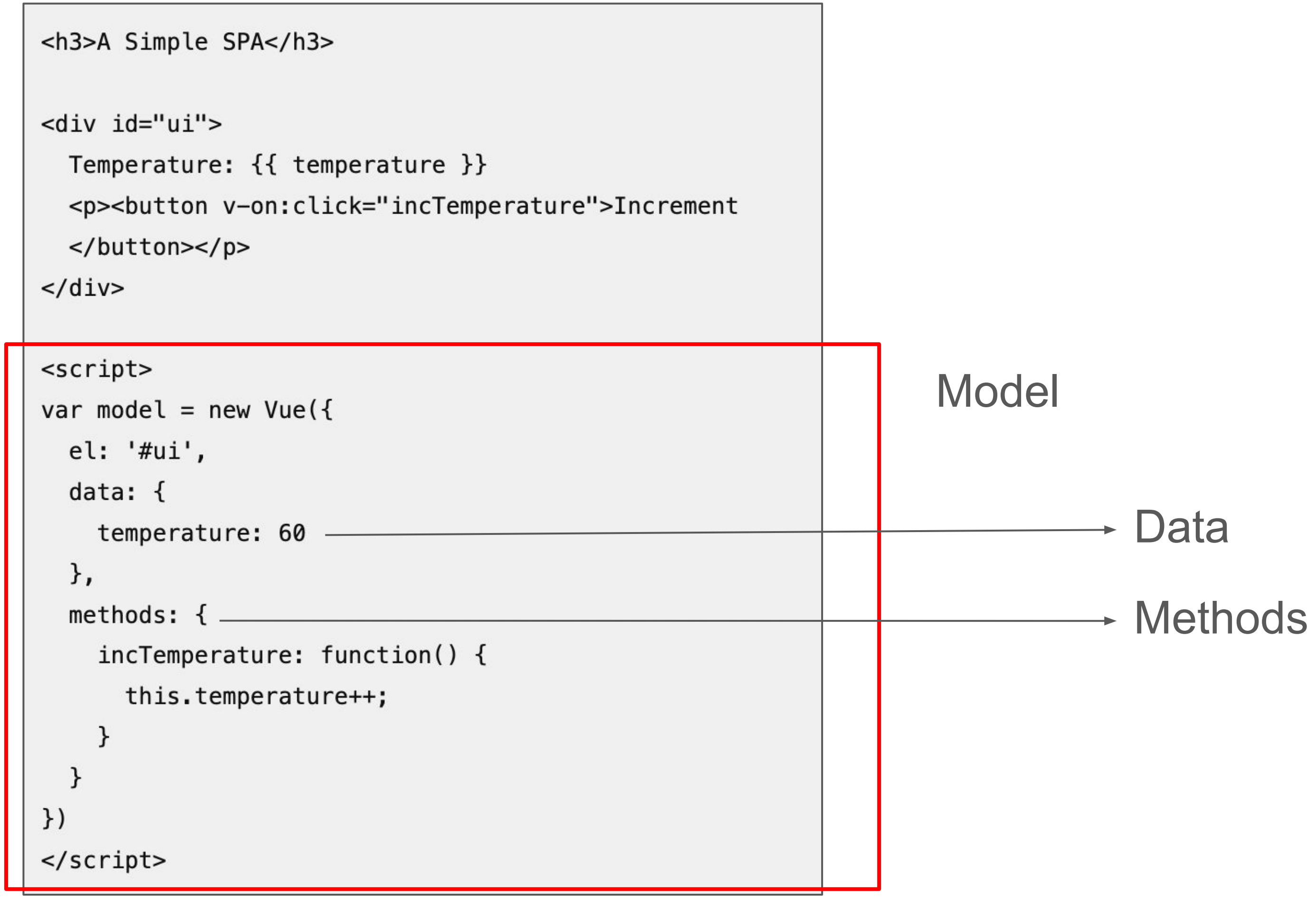


A Simple SPA

Temperature: 60

```
<h3>A Simple SPA</h3>

<div id="ui">
  Temperature: {{ temperature }}
  <p><button v-on:click="incTemperature">Increment
  </button></p>
</div>

<script>
var model = new Vue({
  el: '#ui',
  data: {
    temperature: 60
  },
  methods: {
    incTemperature: function() {
      this.temperature++;
    }
  }
})
</script>
```

Model



```
<h3>A Simple SPA</h3>

<div id="ui">
  Temperature: {{ temperature }}
  <p><button v-on:click="incTemperature">Increment
  </button></p>
</div>

<script>
var model = new Vue({
  el: '#ui',
  data: {
    temperature: 60
  },
  methods: {
    incTemperature: function() {
      this.temperature++;
    }
  }
})
</script>
```

```
<h3>A Simple SPA</h3>

<div id="ui">
  Temperature: {{ temperature }}
  <p><button v-on:click="incTemperature">Increment
  </button></p>
</div>

<script>
var model = new Vue({
  el: '#ui',
  data: {
    temperature: 60
  },
  methods: {
    incTemperature: function() {
      this.temperature++;
    }
  }
})
</script>
```

```
<h3>A Simple SPA</h3>

<div id="ui">
  Temperature: {{ temperature }}
  <p><button v-on:click="incTemperature">Increment
  </button></p>
</div>

<script>
var model = new Vue({
  el: '#ui',
  data: {
    temperature: 60
  },
  methods: {
    incTemperature: function() {
      this.temperature++;
    }
  }
})
</script>
```

```
<h3>A Simple SPA</h3>

<div id="ui">
  Temperature: {{ temperature }}
  <p><button v-on:click="incTemperature">Increment
  </button></p>
</div>

<script>
var model = new Vue({
  el: '#ui',
  data: {
    temperature: 60
  },
  methods: {
    incTemperature: function() {
      this.temperature++;
    }
  }
})
</script>
```

Summary

Traditional MVC (Smalltalk): desktop apps, pre-Web

MVC Web: MVC adaptation for the Web (fullstack)

RAILS

django

Laravel

SPA: MVC adaptation for responsive apps (frontend)

React

Vue.js

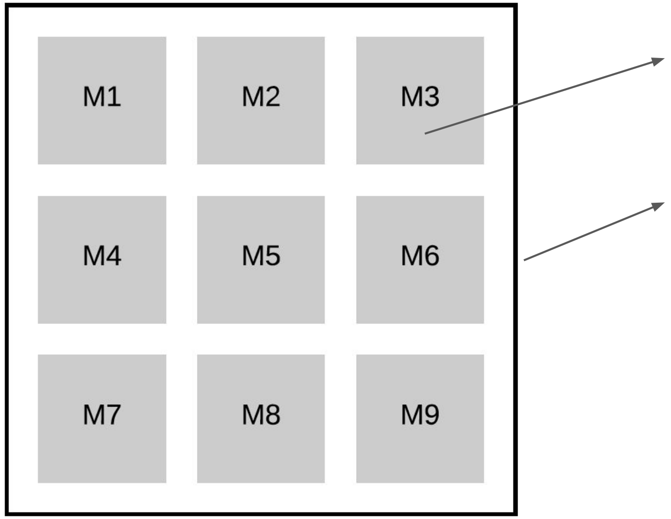
Summary: MVC Variants

- Traditional: Smalltalk, before Web
- Web: similar to 3-Tier
- SPA: similar to traditional MVC

Microservices

Monoliths

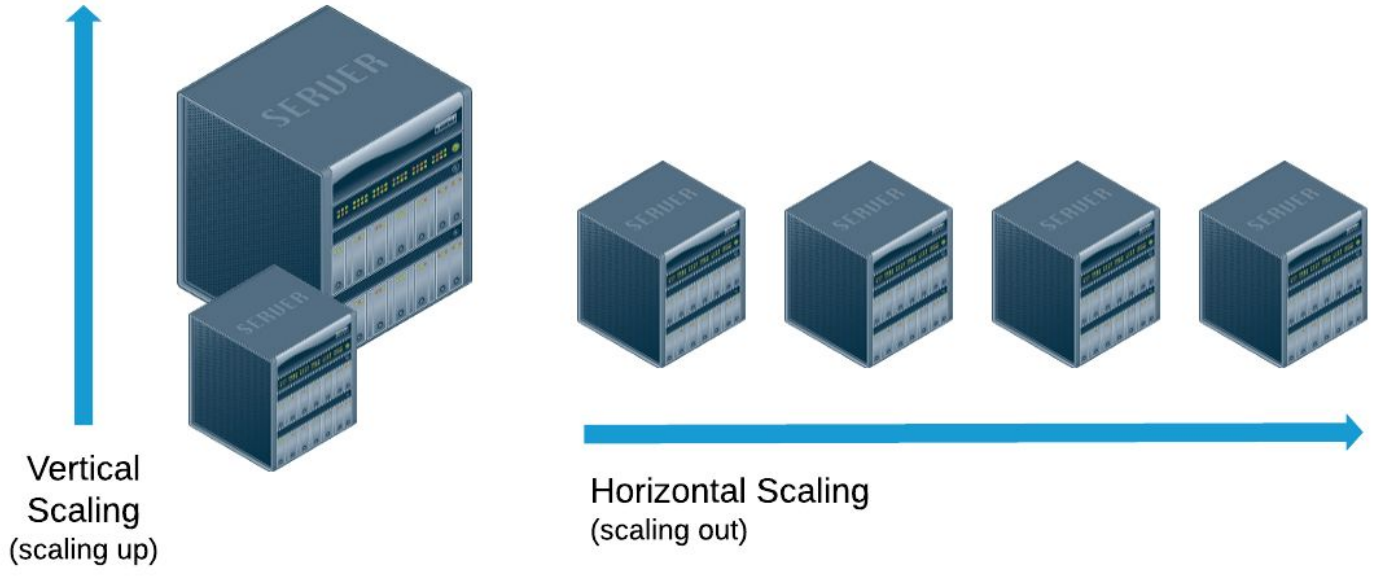
- Monoliths: system is a single process at run-time
- Process: operating system process



Modules: compilation units

At runtime, everything runs in a single process

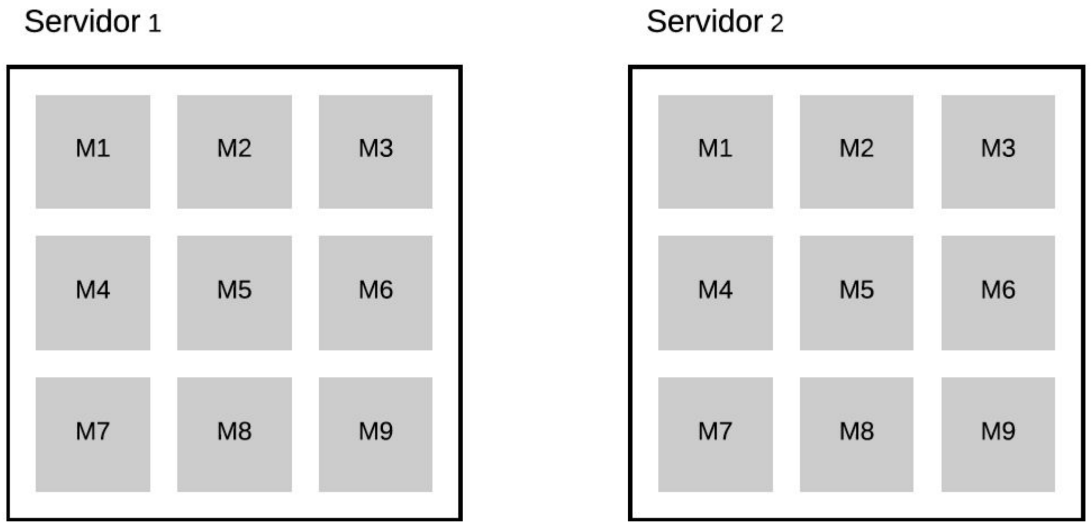
Vertical vs Horizontal Scalability



https://www.section.io/blog/scaling-horizontally-vs-vertically/

Problem #1 with Monoliths: Scalability

- Horizontal scalability requires scaling the entire monolith
- Even when the bottleneck is in a single module



Problem #2 with Monoliths: Releases are slower

- The release process is slow, centralized, and bureaucratic
- Teams don't have autonomy to put modules into production
- Reason: changes can impact other teams' modules
- As a result:
  - Predefined dates for release
  - Several tests, sometimes manual, before release

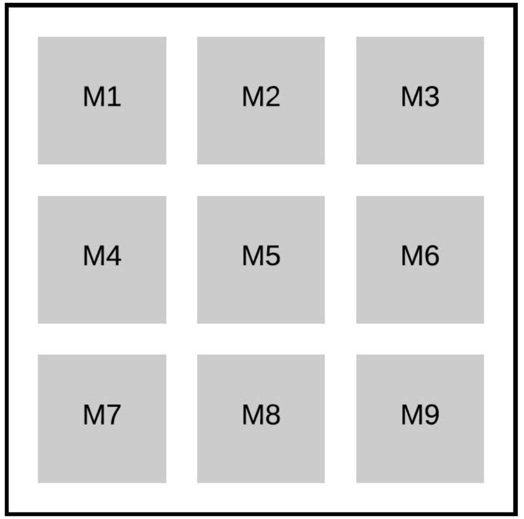
particularly, in a monolithic codebase

Microservices

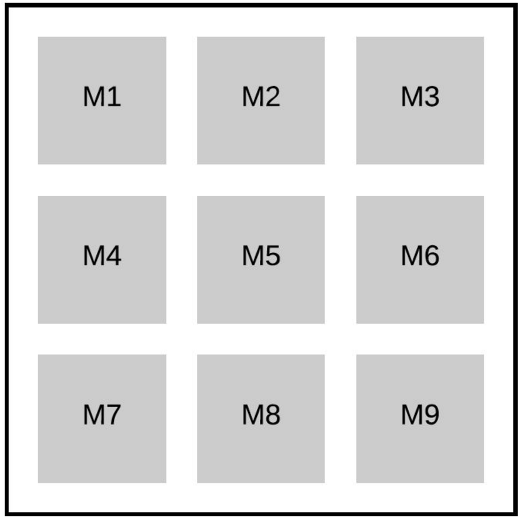
Microservices

- Services ⇒ Modules are independent processes
- Micro ⇒ small modules

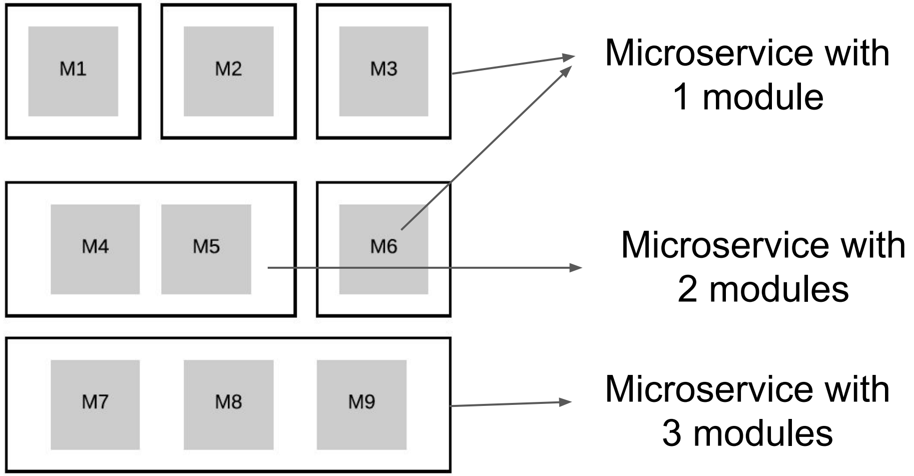
Monolithic Architecture



Monolithic Architecture



Microservices-based Architecture


M1
M2
M3
M4
M5
M6
M7
M8
M9
Microservice with 1 module
Microservice with 2 modules
Microservice with 3 modules

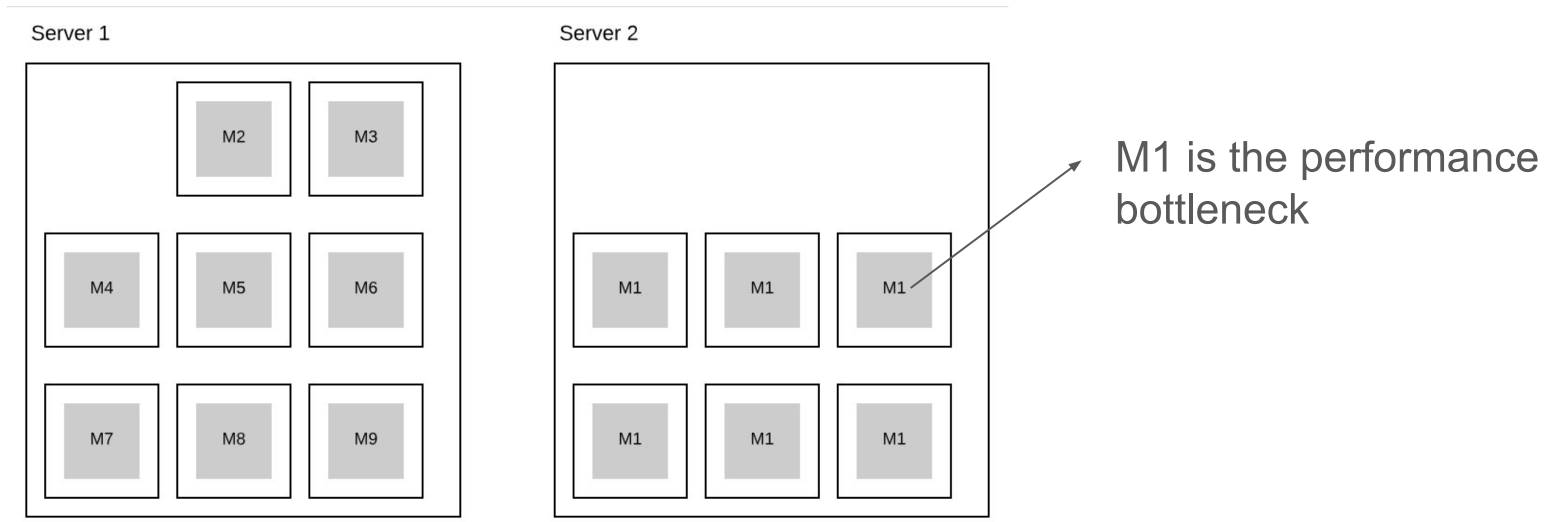
microservice = process (run-time, operating system)

Explaining it another way

- Suppose a system with *n* endpoints
- Monolith: all *n* endpoints in the same process
- Microservices: each endpoint is a separate process

Advantage #1: Scalability

- We can scale just the module with performance problem



Advantage #2: Flexibility for Releases

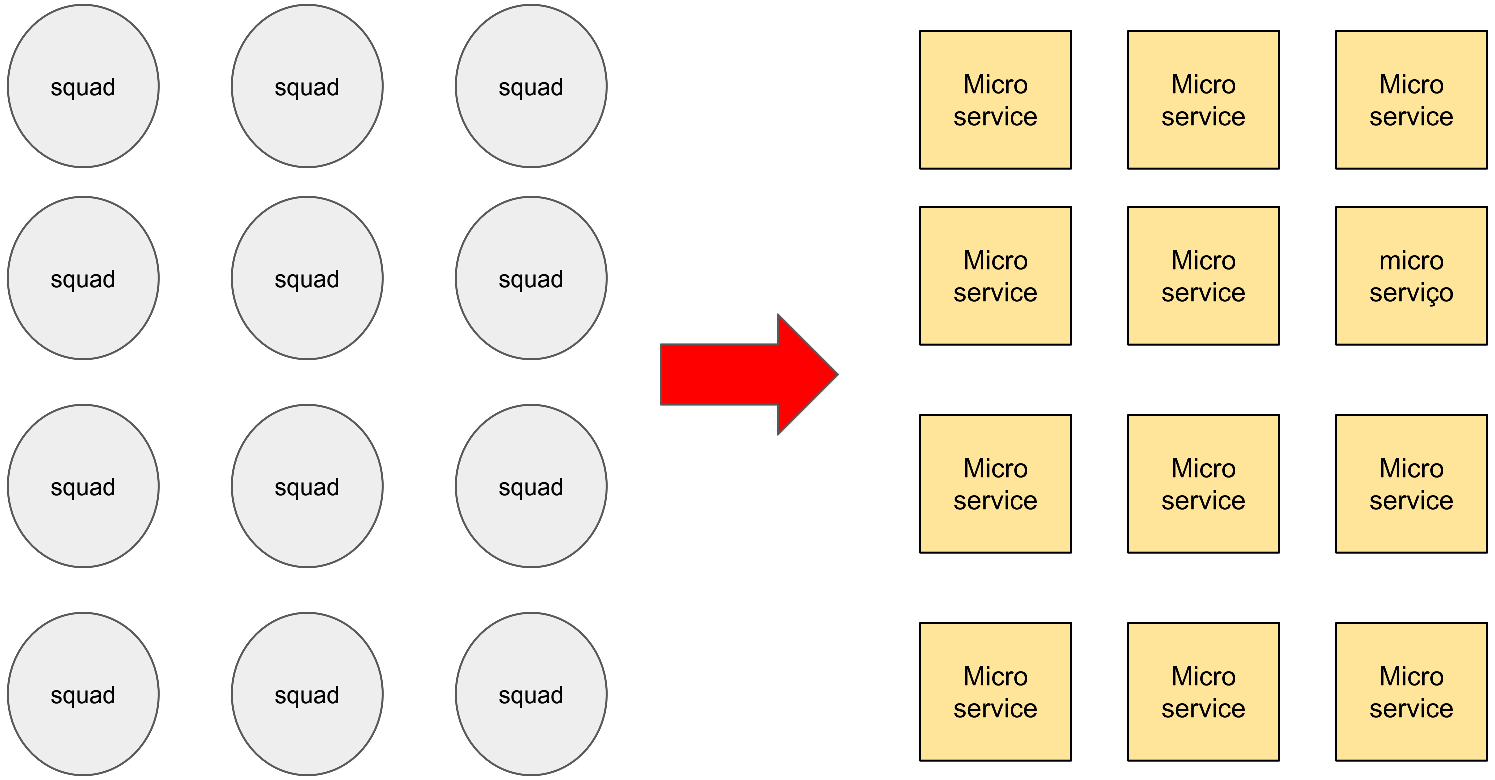
- Chances of interference between processes are smaller
- Reason: each process has its own address space
- Thus, teams have autonomy to put microservices into production

Other advantages

- Microservices can use different technologies
- Partial failures (e.g., only one microservice can be offline)

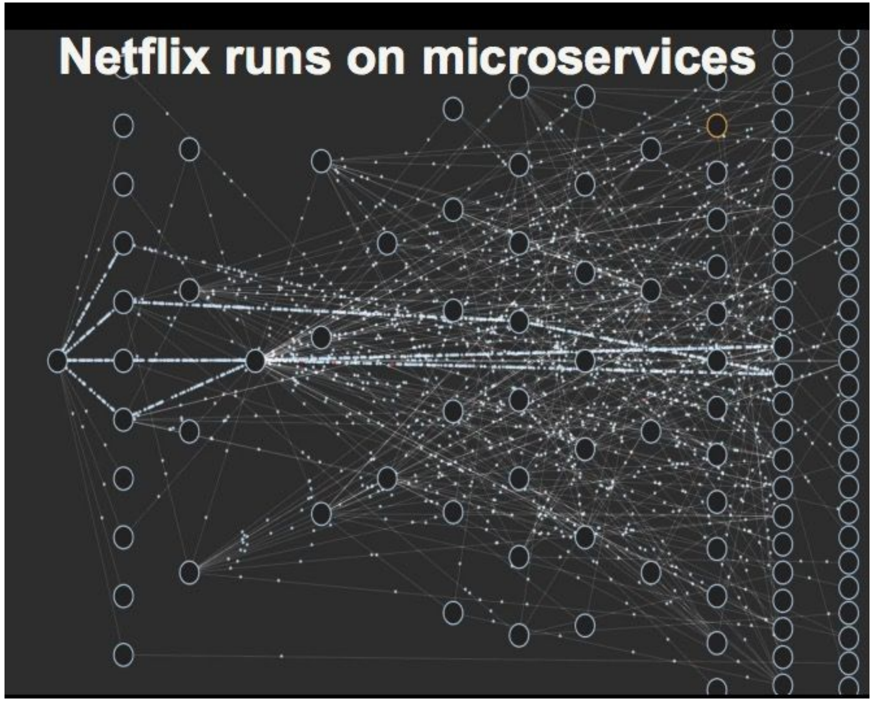
Lei de Conway (1968)

- Software architecture mirrors the organization's architecture



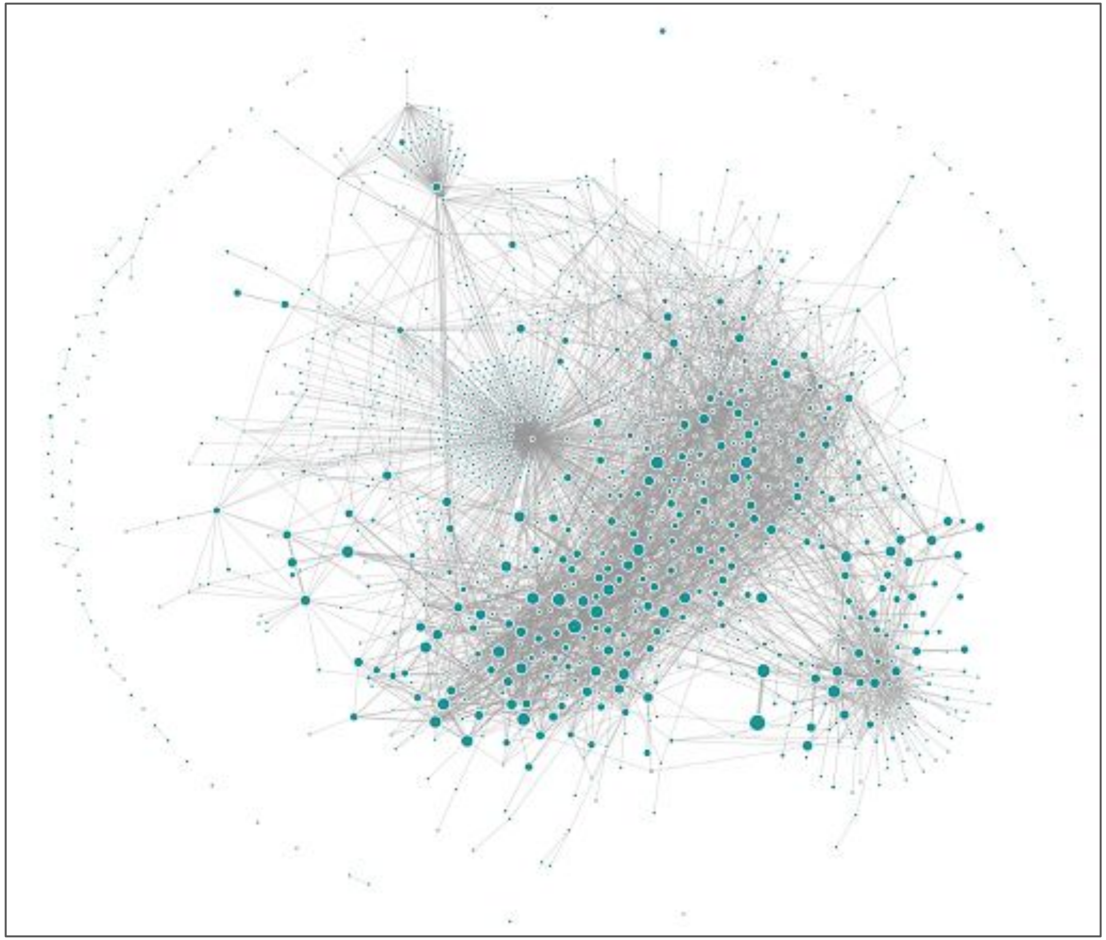
Who uses microservices?

- Large companies like Netflix, Amazon, Google, etc



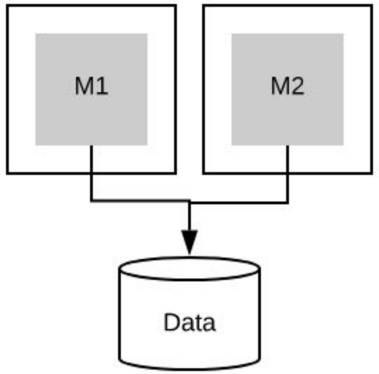
Each node is a microservice

Example: Uber (~2018)



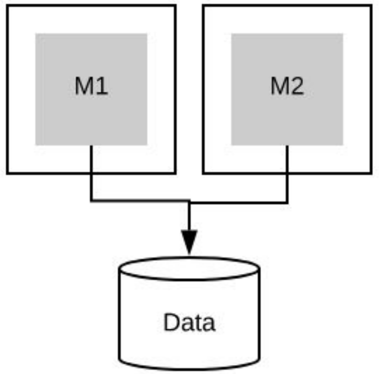
https://eng.uber.com/microservice-architecture/

Microservices & Databases

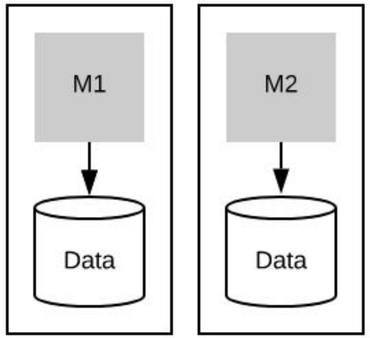


- Architecture that is **not** recommended
- Reason: it increases coupling between M1 and M2

Microservices & Databases



- Architecture that is **not** recommended.
- Reason: it increases coupling between M1 and M2



- Recommended architecture
- Reason: there is no data coupling between M1 and M2. Thus, M1 and M2 can evolve independently
- If M1 needs to use M2 (or vice versa), this should occur via interfaces

When not to use microservices?

- Microservices-based architecture is more complex
    - Distributed system (manage hundreds of processes)
    - Latency (communication via network)
    - Distributed transactions

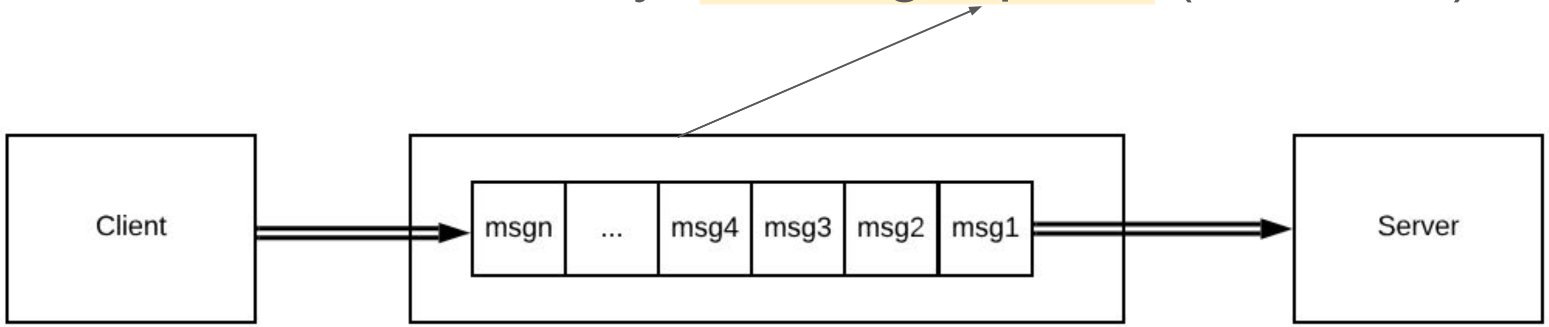
Recommendation: start with monoliths

- Only migrate to microservices if:
  - Monolith with performance problems
  - Monolith is delaying releases
- Migration can be gradual…

Message-Oriented Architecture

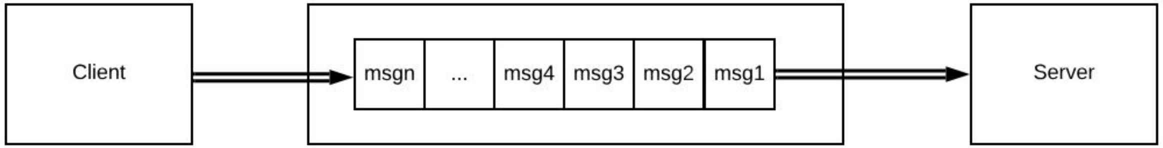
Message-Oriented Architecture

- Context: distributed applications
- Clients do not communicate directly with the servers
- But with an intermediary: message queue (or broker)



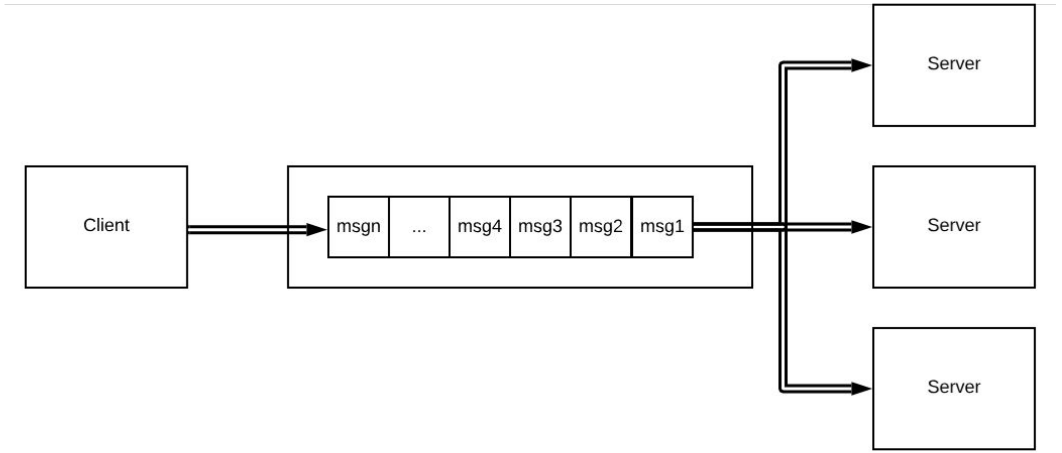
Advantage #1: Fault Tolerance

- No more "server is down" message
- Assuming the message queue runs on a reliable server



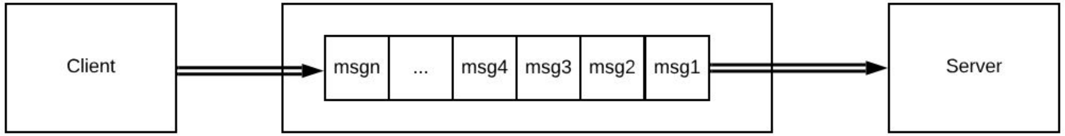
Advantage #2: Scalability

- Easier to add new servers and harder to overload a server with too many messages



Asynchronous Communication

- Loose coupling between clients and servers
- Space decoupling: clients do not know servers and vice versa
- Time decoupling: clients and servers do not need to be simultaneously available



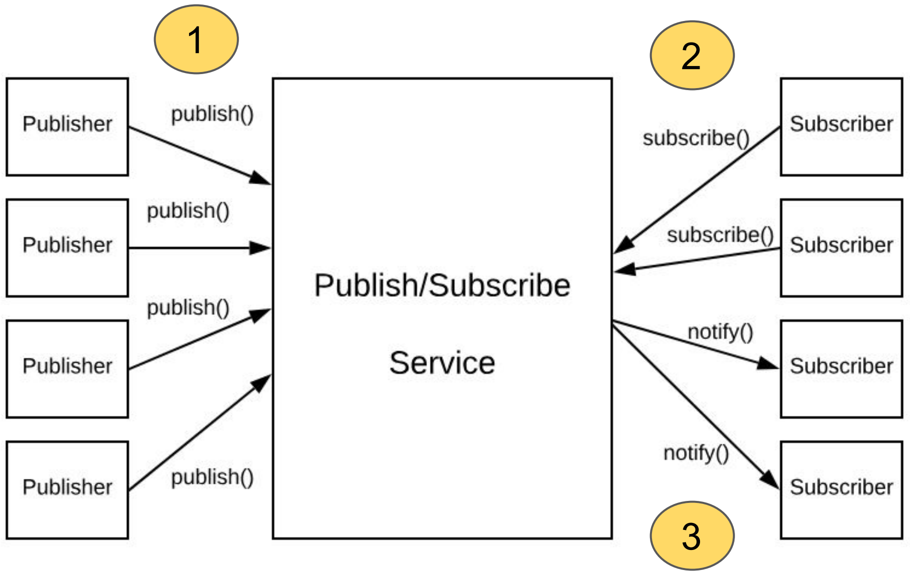
Publish/Subscribe Architecture

Publish/Subscribe

- Improvement of message queue
- Messages are called events

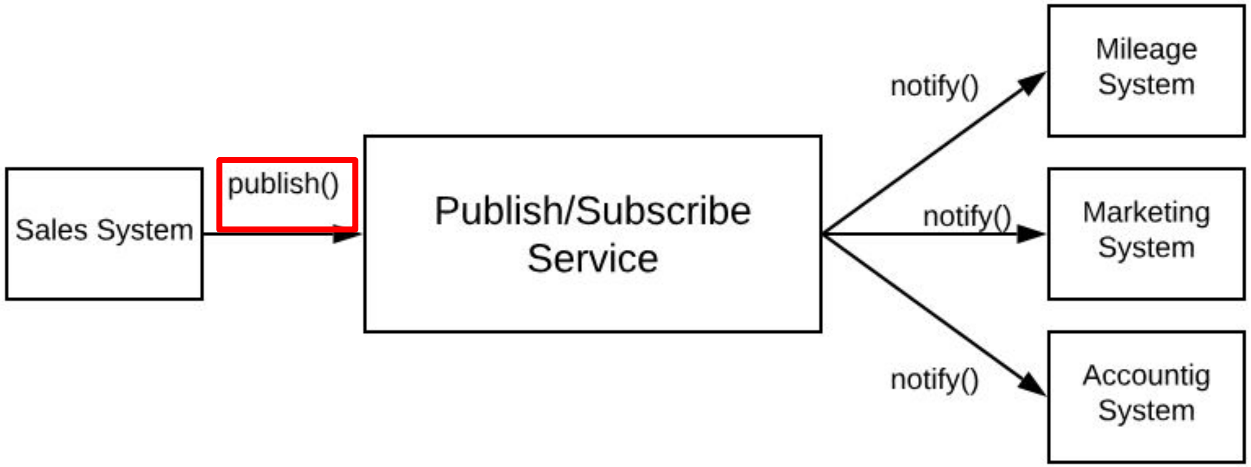
Publish/Subscribe

- Systems can: (1) publish events; (2) subscribe to events; (3) be notified about the occurrence of events

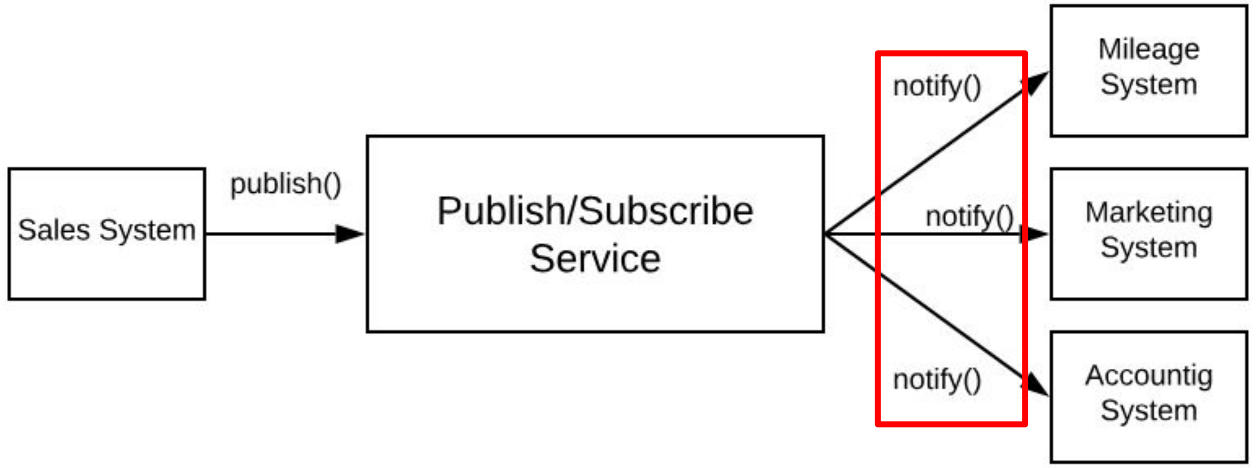


Example: Airline System

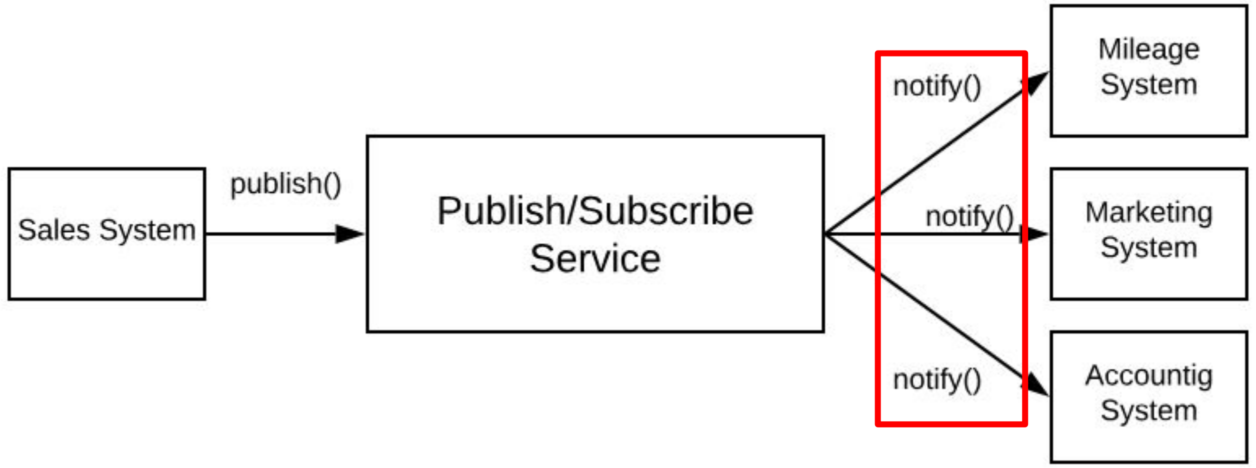
- Event: ticket sale



Example: Airline System



Example: Airline System

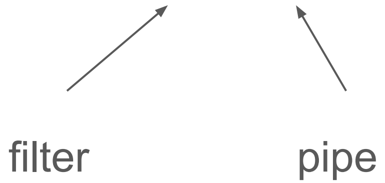


Group communication: 1 system publishes events, *n* systems subscribe and are notified of this publication

Other Architectural Patterns

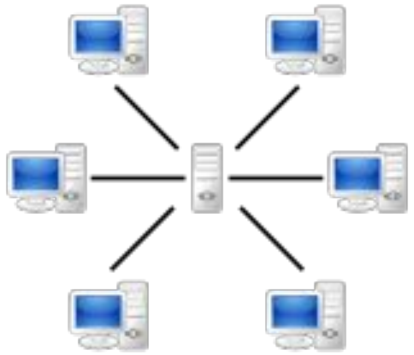
(1) Pipes and Filters

- Programs are called **filters** and communicate via **pipes** (which act as buffers)
- Flexible architecture. Used by unix commands.
- Example: `ls | grep csv | sort`



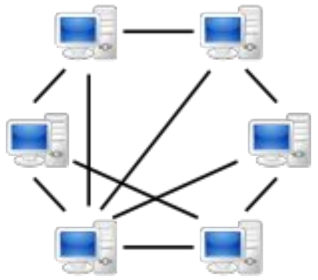
(2) Client/Server

- Common in network services
- Examples: print service, file service, web service



(3) Peer-to-Peer

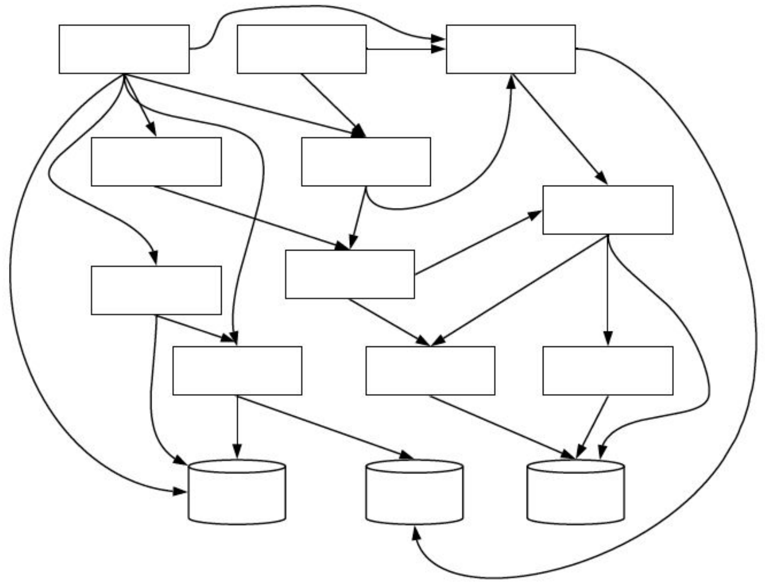
- Every node is a client and server
- Consumer and provider of resources
- Example: file sharing using BitTorrent; Blockchain



Architectural Anti-Patterns

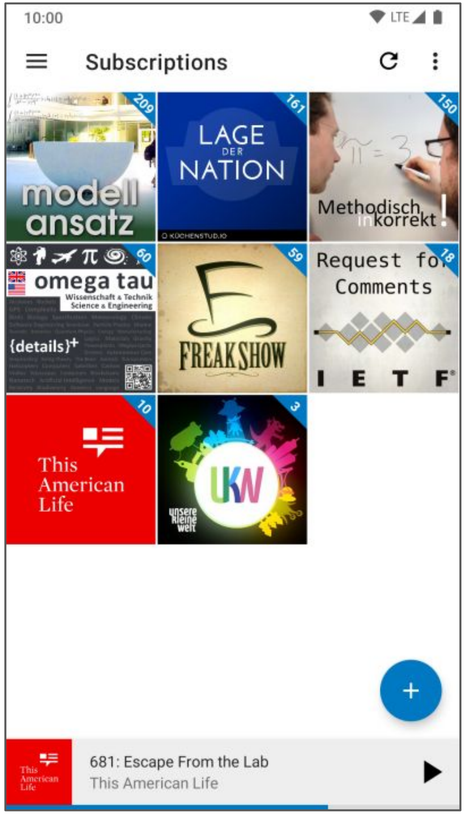
Big Ball of Mud

- A module can use any other module of the system

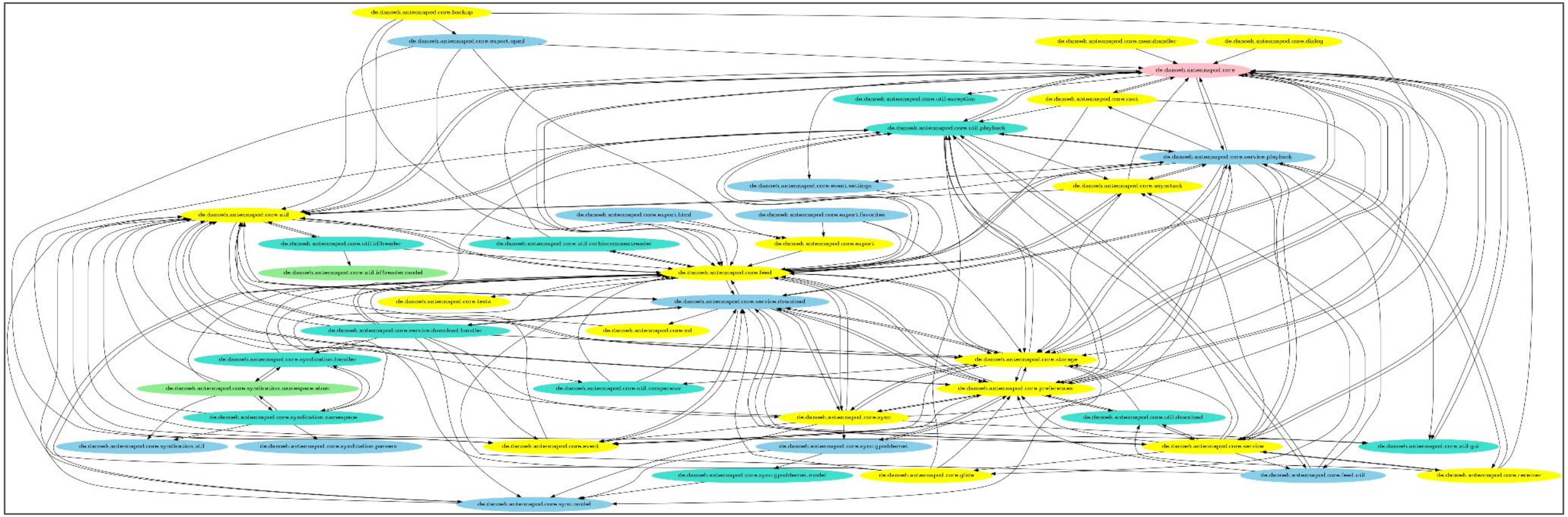


Example of Remodularization

- AntennaPod: open-source podcast player
- Android and Java



November, 2020 - Big Ball of Mud (with many circular dependencies)


de.danoeh.antennapod.core.backup
de.danoeh.antennapod.core.export.opml
de.danoeh.antennapod.core.menuhandler
de.danoeh.antennapod.core.dialog
de.danoeh.antennapod.core
de.danoeh.antennapod.core.util.exception
de.danoeh.antennapod.core.cast
de.danoeh.antennapod.core.util.playback
de.danoeh.antennapod.core.service.playback
de.danoeh.antennapod.core.event.settings
de.danoeh.antennapod.core.asynctask
de.danoeh.antennapod.core.util
de.danoeh.antennapod.core.export.html
de.danoeh.antennapod.core.export.favorites
de.danoeh.antennapod.core.util.id3reader
de.danoeh.antennapod.core.util.vorbiscommentreader
de.danoeh.antennapod.core.export
de.danoeh.antennapod.core.util.id3reader.model
de.danoeh.antennapod.core.feed
de.danoeh.antennapod.core.tests
de.danoeh.antennapod.core.service.download
de.danoeh.antennapod.core.service.download.handler
de.danoeh.antennapod.core.ssl
de.danoeh.antennapod.core.syndication.handler
de.danoeh.antennapod.core.storage
de.danoeh.antennapod.core.syndication.namespace.atom
de.danoeh.antennapod.core.util.comparator
de.danoeh.antennapod.core.preferences
de.danoeh.antennapod.core.syndication.namespace
de.danoeh.antennapod.core.sync
de.danoeh.antennapod.core.util.download
de.danoeh.antennapod.core.syndication.util
de.danoeh.antennapod.core.syndication.parsers
de.danoeh.antennapod.core.event
de.danoeh.antennapod.core.sync.gpoddernet
de.danoeh.antennapod.core.service
de.danoeh.antennapod.core.util.gui
de.danoeh.antennapod.core.sync.gpoddernet.model
de.danoeh.antennapod.core.glide
de.danoeh.antennapod.core.feed.util
de.danoeh.antennapod.core.receiver
de.danoeh.antennapod.core.sync.model

https://antennapod.org/blog/2024/05/modernizing-the-code-structure

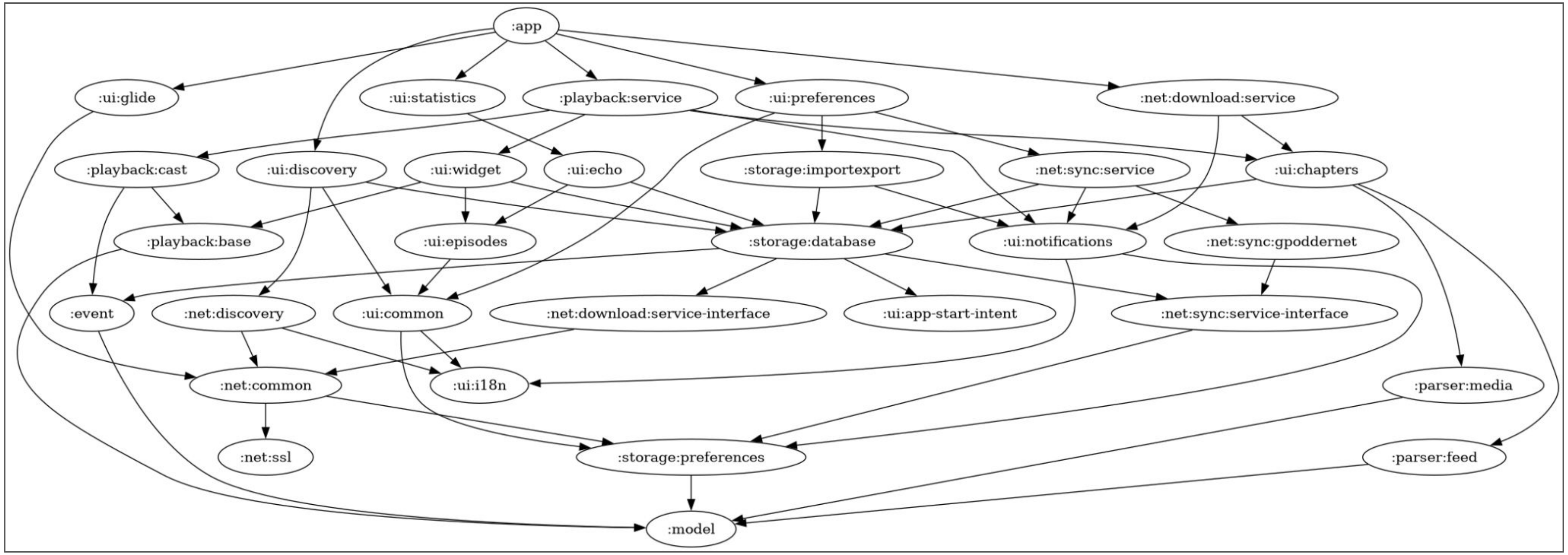
Developers' Comments

- "To test the database, for example, one normally wouldn’t have to launch the full app. However, because the database basically depended on everything else, most of our tests required starting up a full Android device."

Developers' Comments

- "A particularly problematic aspect of the structure was that there were many “utility” classes. These utility classes caused many of the cycles visible in the structure."

May, 2024



https://antennapod.org/blog/2024/05/modernizing-the-code-structure

Exercises

1. What is the likely architecture of the following systems? Briefly justify your answer.

   (a) Microsoft Excel (desktop version)

   (b) A Banking App (mobile)

   (c) Twitter Web (front-end)

   (d) Google Slides (front-end)

   (e) Twitter (backend)

   (f) Moodle

2. Answer on microservices:

(a) Why does microservices provide flexibility for teams to independently deploy their code?

(b) Why is this independence less feasible with monoliths? In other words, why is it not recommended for a team to immediately deploy a modification made in a monolith?

(c) Why should microservices not share the same database?

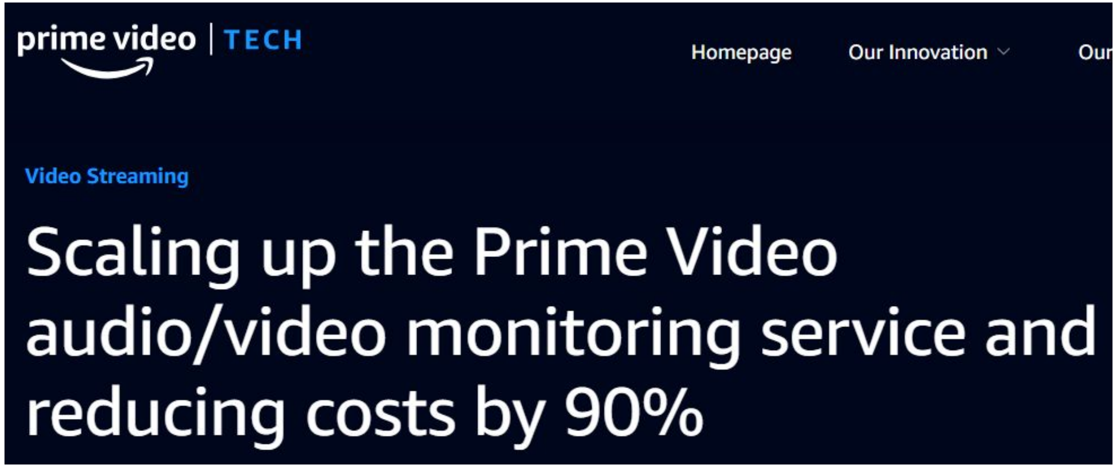
3. Suppose a streaming company that plans to implement a system to detect videos with quality problems (for example, issues in captions, audio, frozen images, etc). The videos are stored in a storage system, i.e., secondary memory. The company is evaluating two architectures:

Architecture #1: each detector is a microservice, which receives the name of the video as a parameter, loads it from storage, and executes a particular quality detection algorithm on that video.

Architecture #2: the quality detectors are modules of a monolith. Thus, the video is loaded only once from storage to main memory and shared by all detectors.

Assuming that the streaming company has millions of customers, which architecture is more scalable? Justify.

Note: this exercise is based on a post from the Amazon Prime Video blog


prime video | TECH

Homepage Our Innovation Our

Video Streaming

Scaling up the Prime Video audio/video monitoring service and reducing costs by 90%

March, 2023

End