Announcements

- Requirements Assignment Due Tonight
- Keep in mind that Code Review Assignment is also due with Project 1
  - PRs should be well spaced in time for full credit
- Mid term Feedback Survey Released Today
  - Please express your comments/concerns about the course so far.
  - Your responses will be stored anonymously
  - Due Oct 1
- Extra Credit Opportunity today

CS3300 Introduction to Software Engineering

Lecture 10: Software Architecture

Dr. Nimisha Roy ▸ nroy9@gatech.edu

Slides adapted from Alessandro Orso

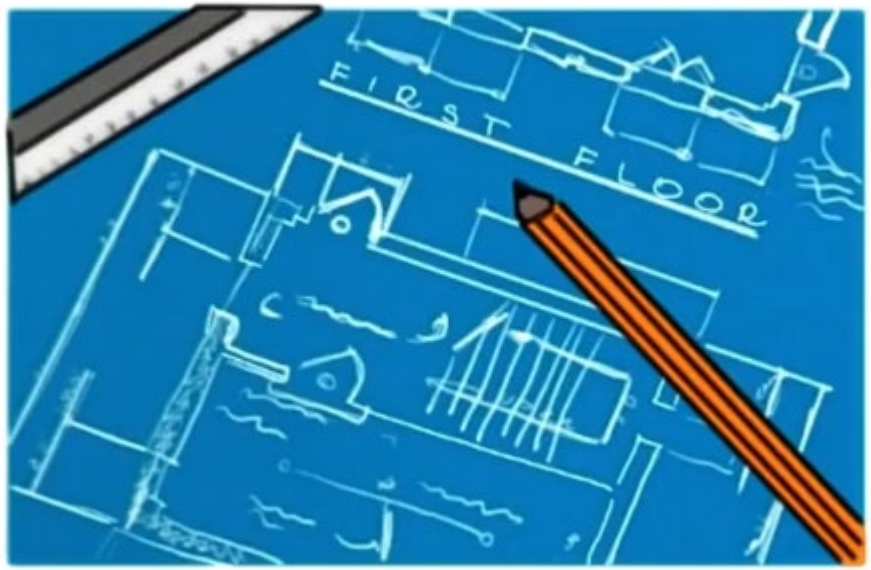
Software Architecture

- The fundamental **organization of a system**, embodied in its components, their **relationships to each other** and the environment, and the principles governing its **design** and evolution. [ANSI/IEEE Std 1471-2000]

- Expert developers' understanding of the system design. [Ralph Johnson]

A general definition of SWA

Set of principal design decisions about the system



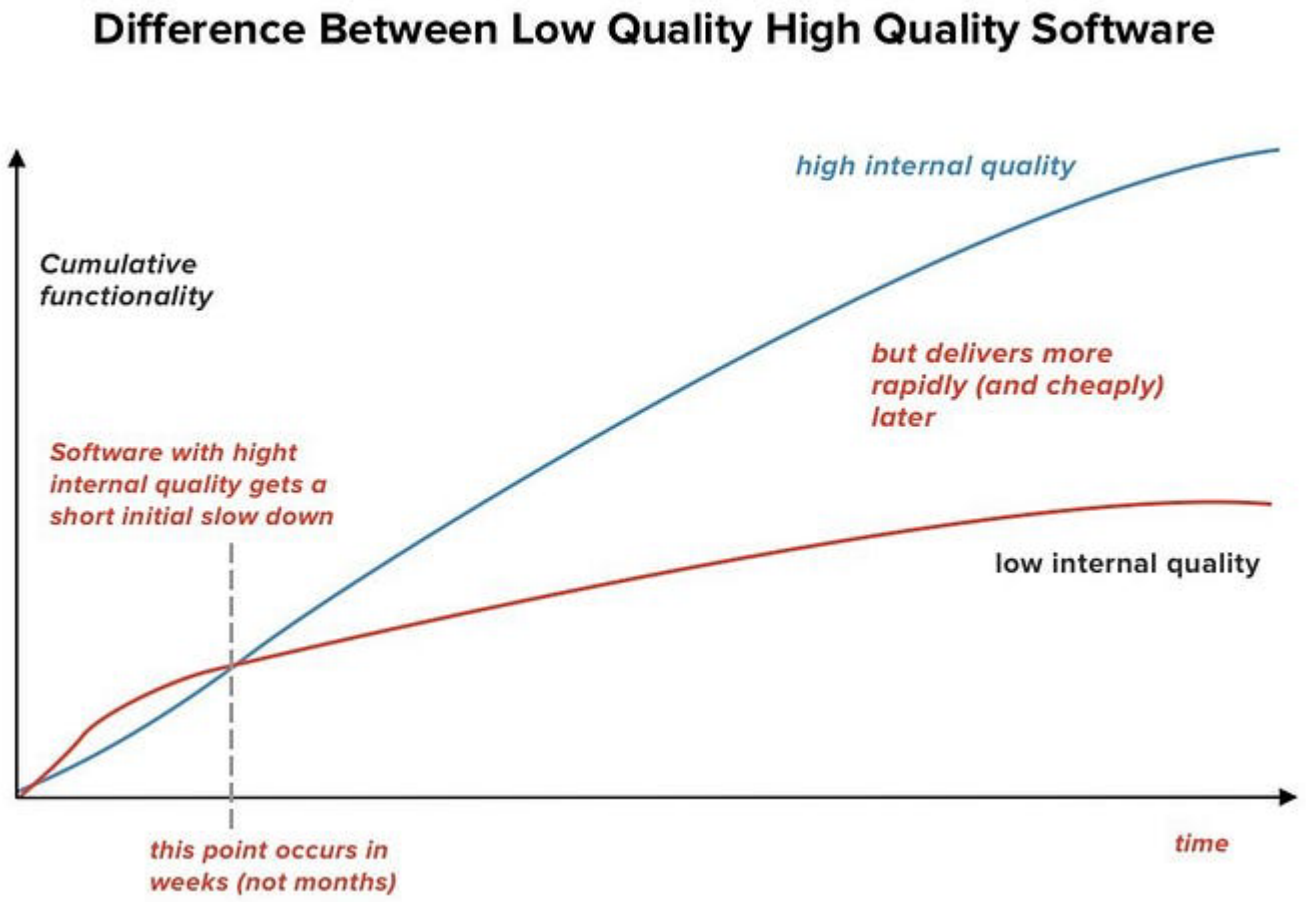
Blueprint of a software system

- Structure
- Behavior
- Interaction
- Follow business requirements

In Netflix, it’s their microservice architecture that empowers them to manage availability, whereas, in Salesforce or Google, it’s a domain-driven design that helps them run domain logic complexity.

Why is SWA important?

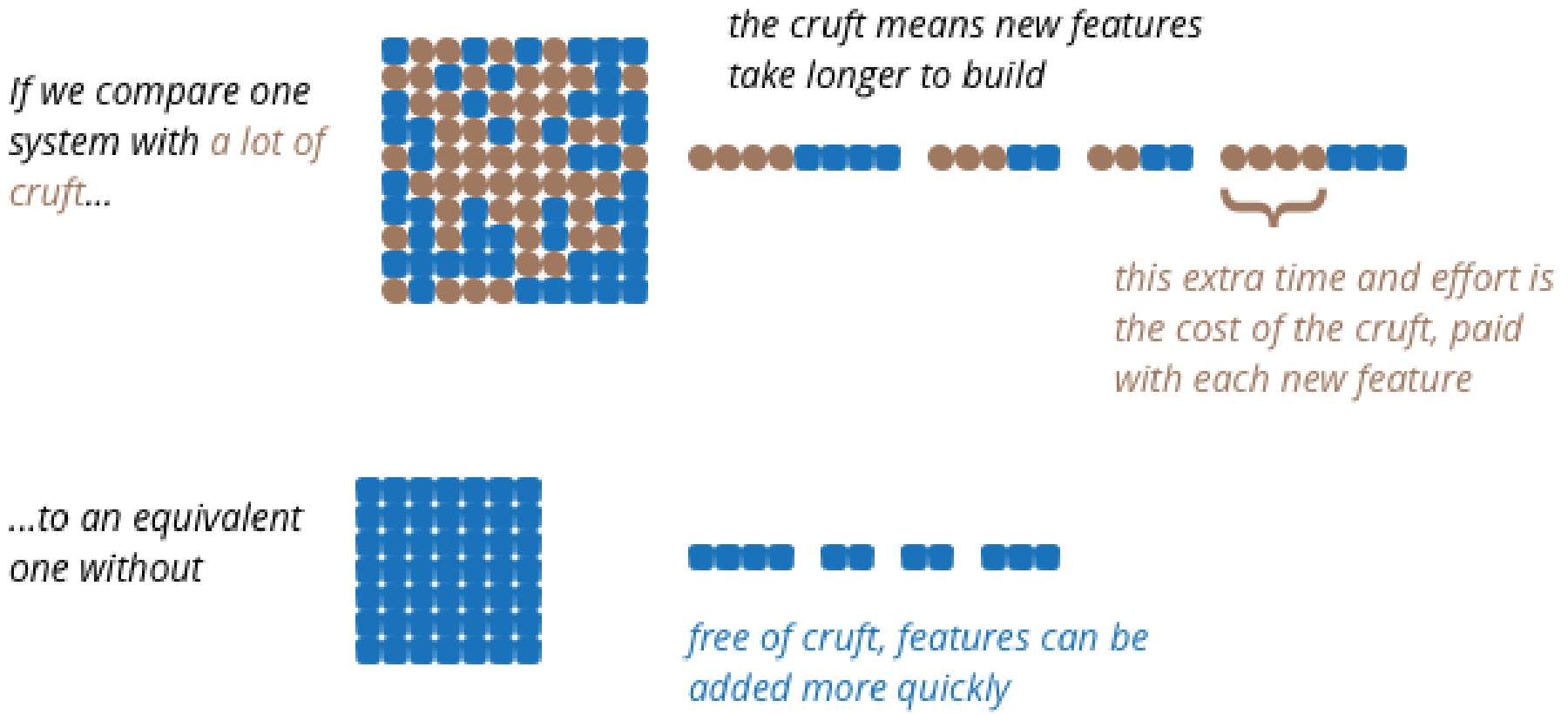
- Helps in maintaining high-quality code, facilitating future changes, and enhancing the longevity of the system.
- Crucial for managing complex systems at scale, enabling smooth feature integration over time.



Why is SWA important?

Tricky subject for the clients of software products - as it isn't something they immediately perceive. Poor architecture is a major contributor to the growth of cruft - elements of the software that impede the ability of developers to understand the software.

- Software that contains a lot of cruft **is much harder to modify**, leading to features that arrive more slowly and with more defects.
- High internal quality leads to faster delivery of new features



Why is SWA important?

Scenario: Suppose two similar products have been launched within the gap of a month. After three months, they require new features to be added.

- Launch on May 31: Messy, tangled code leads to quicker launch but complicates later changes.
- Launch on June 30: Clean, well-architected code is easier to maintain but leads to a slightly delayed launch.

What would a software development company choose to do?

Usually, despite the messy code blocks, team would go for an earlier launch → that's what would matter for the time being - quicker launch → better opportunities to monopolize market. In the second scenario, the changes would take time as quality performance and quality code have been given equal importance →This would disturb time-to-market unfavorably. But, a well-defined system architecture design in the form of microservices will help in easier maintenance. → your company saves time, but it will also satisfy the users with fast and regular updates for new features.

A well-defined architecture allows for easier maintenance and consistent updates, saving time and delivering timely results.

Temporal Aspect

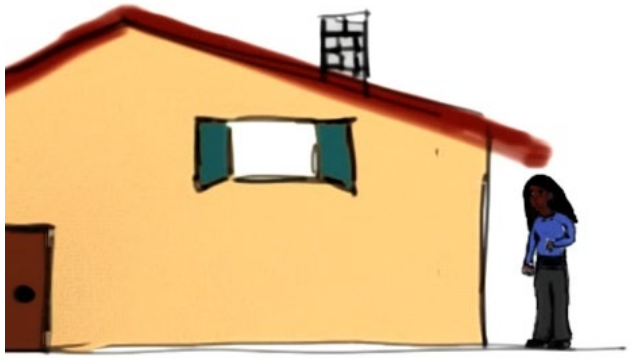
SWA evolves over time.

At any point in time, there is a SWA, but it will change over time

Design decisions are made, unmade, and changed over a system’s lifetime.

Prescriptive vs. Descriptive Architecture

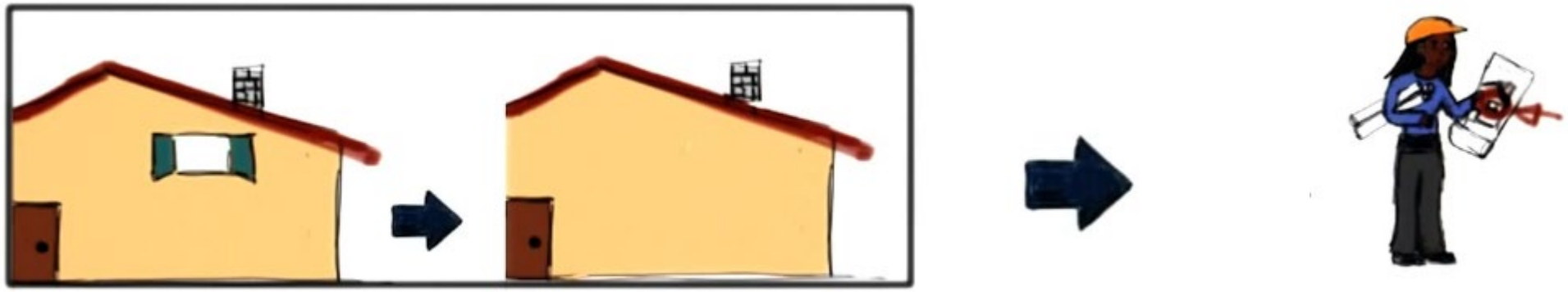
A prescriptive architecture captures the design decisions made prior to the system's construction
=> as- conceived SWA



A descriptive architecture describes how the system has actually been built
=> as- implemented SWA

Architectural Evolution

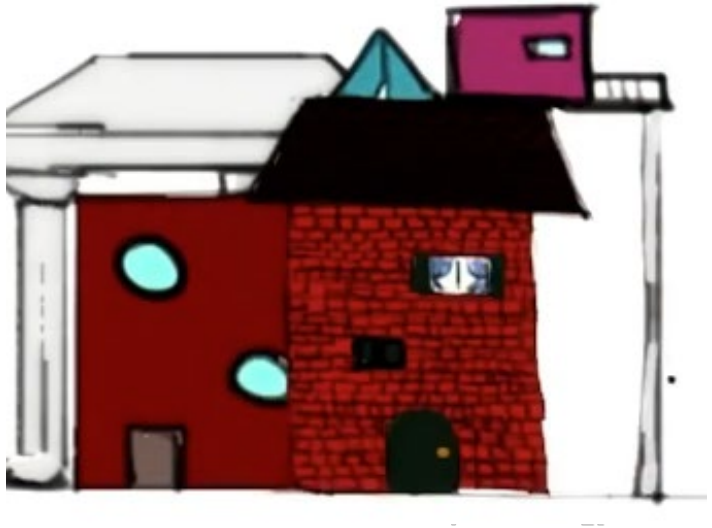
When a system evolves, ideally its prescriptive architecture should be modified first



In practice, this rarely happens

- Developer’s sloppiness
- Short deadlines
- Lack of documented prescriptive architectures

Architectural Degradation



Architectural drift : Introduction of architectural design decisions orthogonal to a system’s prescriptive architecture

Architectural erosion : Introduction of architectural design decisions that violate a system’s prescriptive architecture

Maintaining alignment between prescriptive and descriptive architecture is key to avoiding "drift" and "erosion."

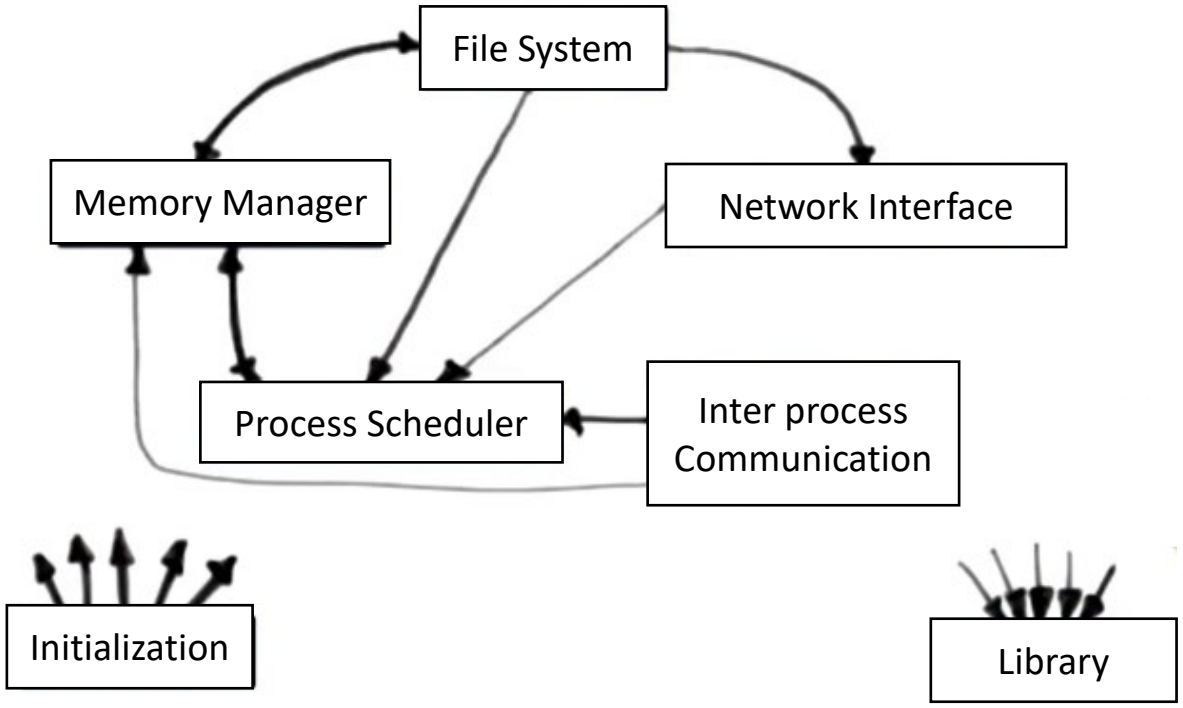
Architectural Recovery

Drift and Erosion => Degraded architecture

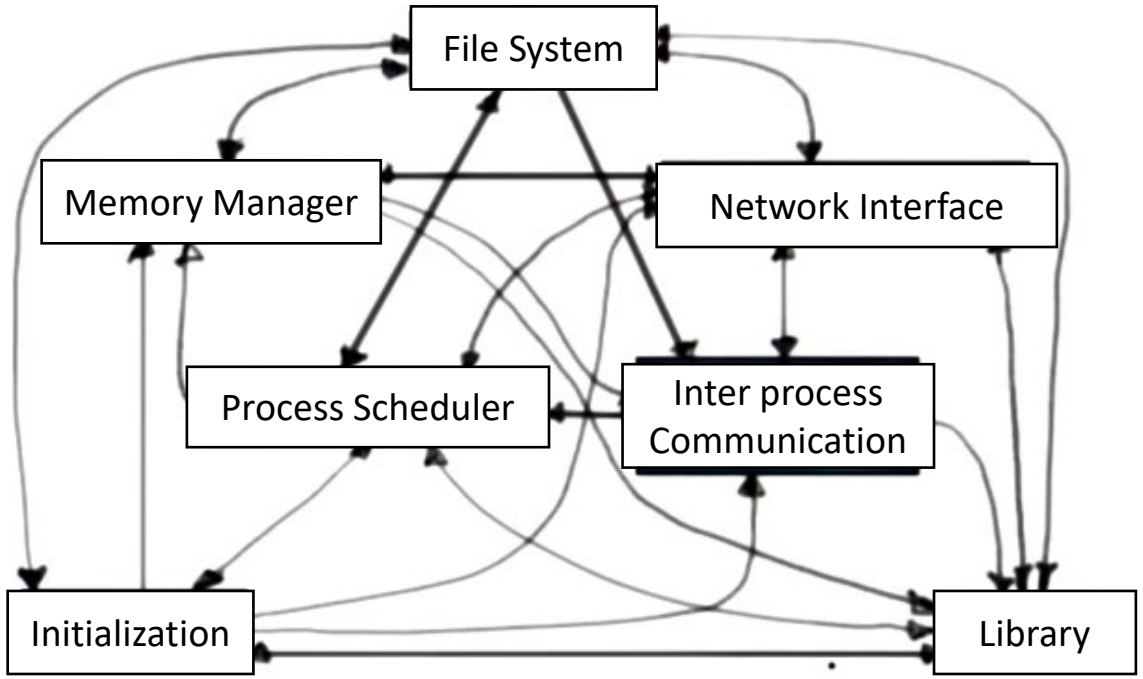
Keep tweaking the code (typically disastrous)

Architectural recovery: determine SWA from implementation and fix it

An example from the Linux Kernel



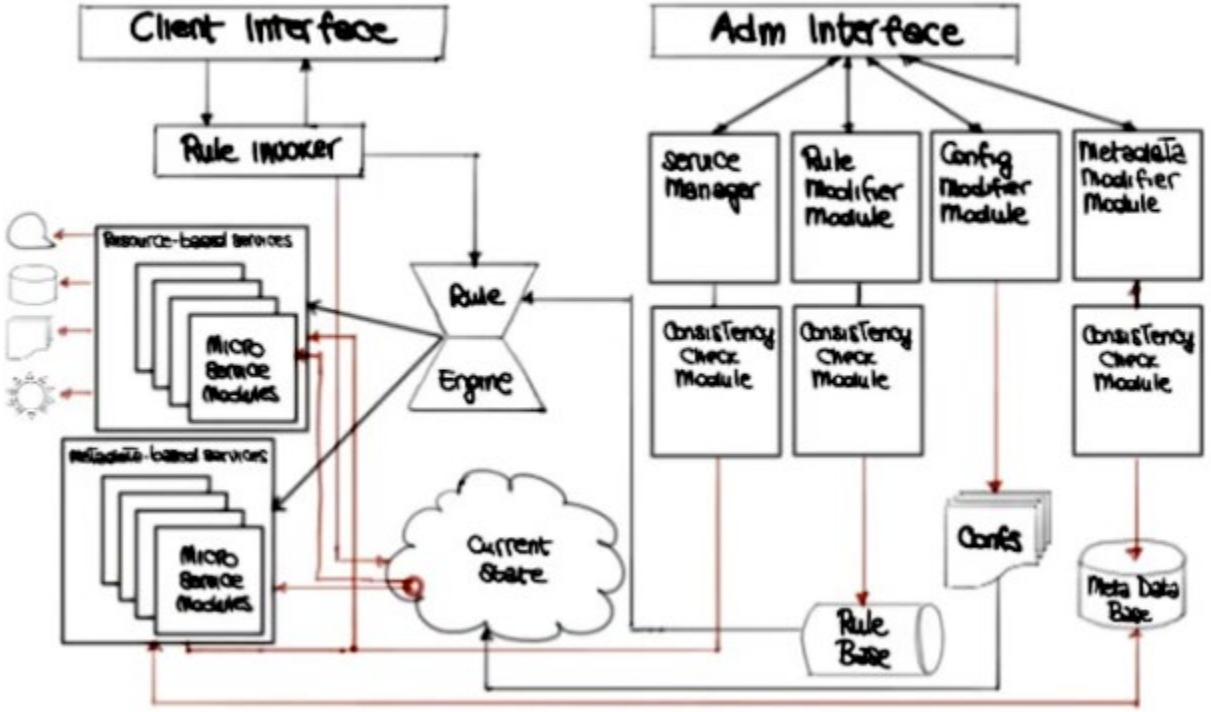
Prescriptive Architecture



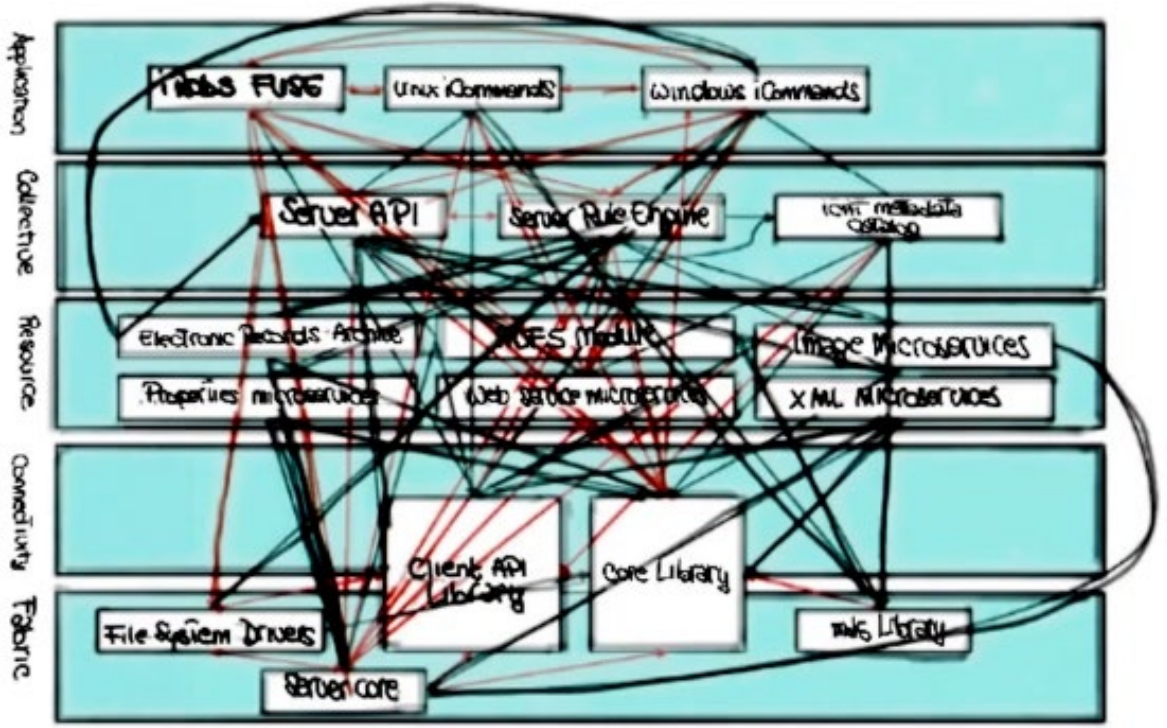
Descriptive Architecture

Another example: iRODS

Data grid system that was built by a biologist. It's a system for storing and accessing big data.



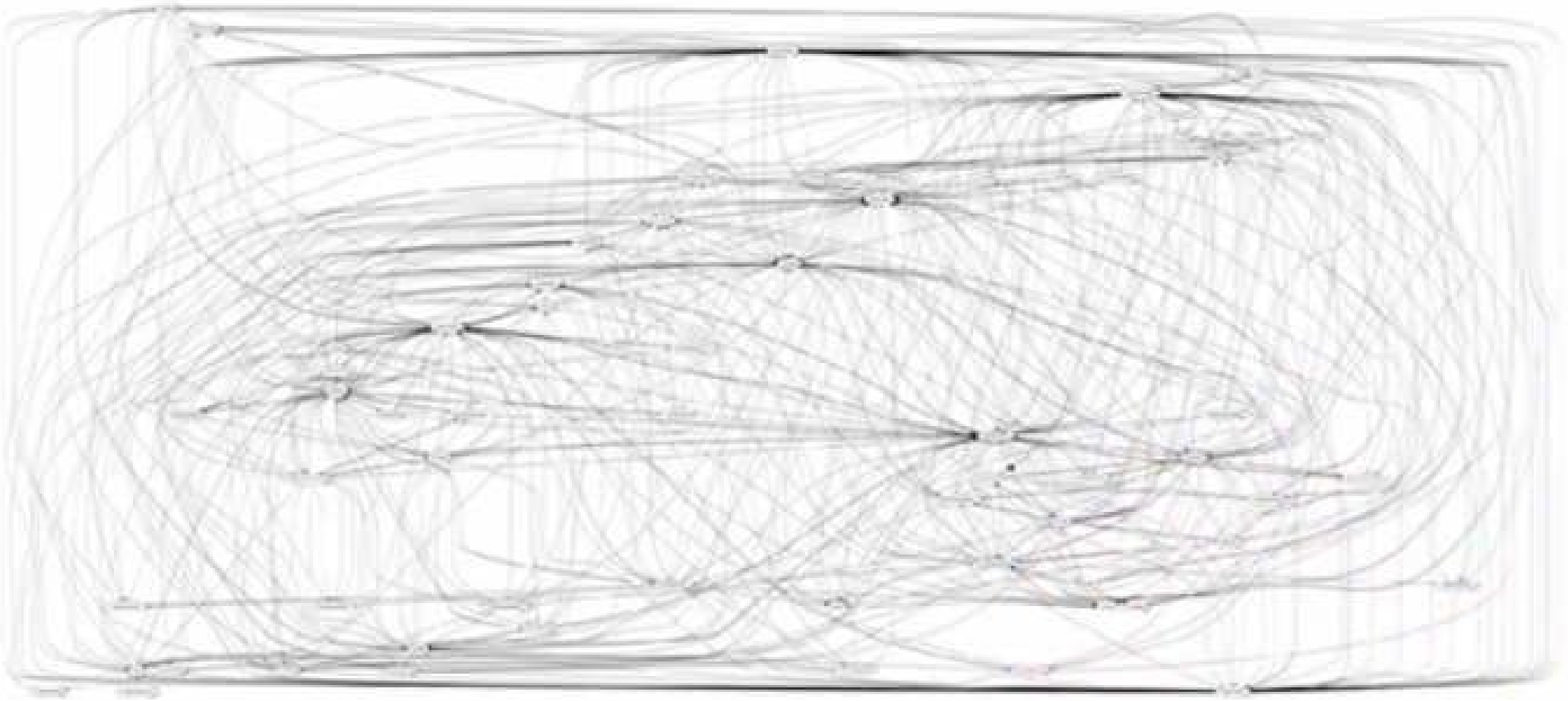
Prescriptive Architecture



Descriptive Architecture

More examples: Hadoop

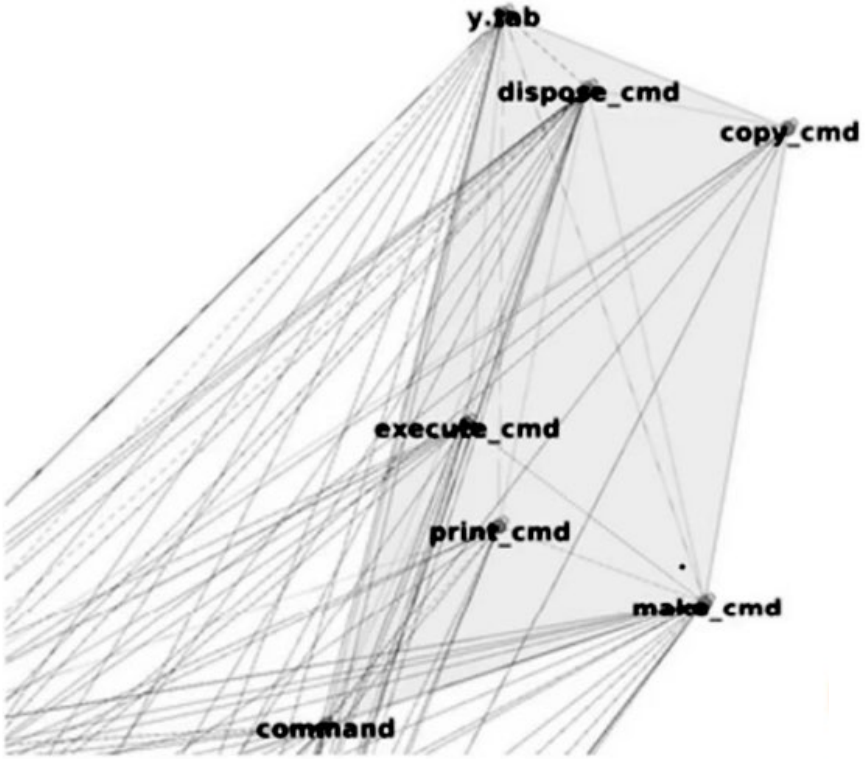
Open-source software framework for storage and large-scale processing of data sets



Descriptive Architecture

Final example: Bash

Unix shell written as a free software replacement for the traditional Bourne shell



Lack of cohesion in the component

High coupling among components

Descriptive Architecture of the command component of Bash.

How AI can Help Maintain Architecture

- AI can automate documentation generation and code reviews, ensuring descriptive architecture evolves alongside system changes.
- AI-based tools monitor system changes and suggest updates to maintain architectural coherence.

Example: AI tools like DeepCode help developers avoid architecture degradation by automatically detecting bad practices early in the development cycle.

Architectural Patterns

An architectural pattern defines “a family of systems in terms of a pattern of structural organization; a vocabulary of components and connectors, with constraints on how they can be combined”

M. Shaw and D. Garlan, 1996

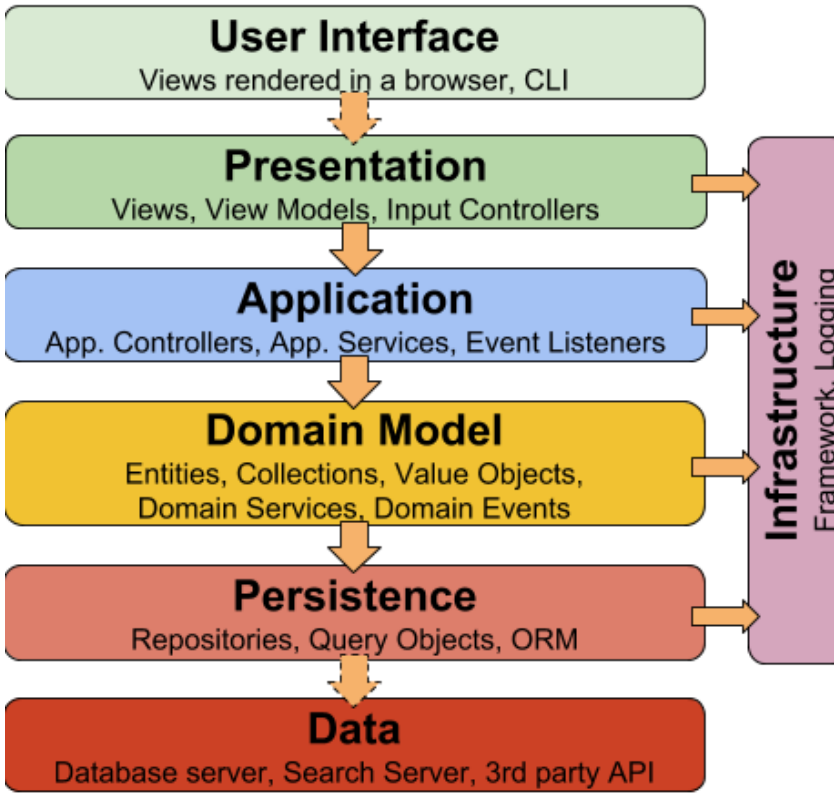
Basically, named collection of architectural design decisions applicable in a given context.

Top Architectural Patterns

- Layered
- Microkernel
- Microservices
- Event-Driven
- Serverless

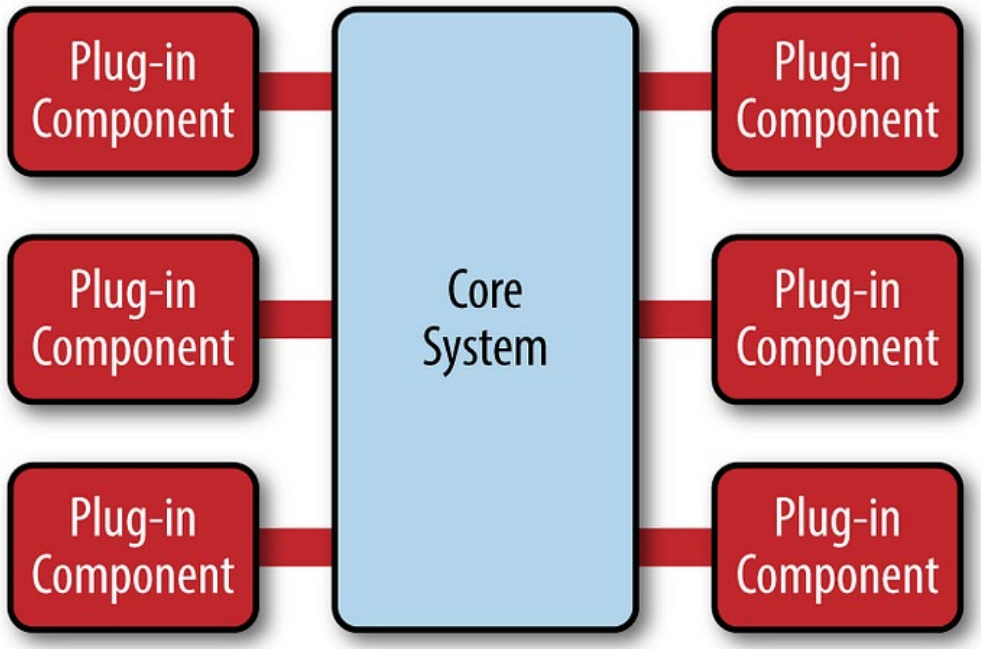
Layered Architecture

- Organizes the system into layers with related functionality/logic associated with each layer
- In a layered system, each layer:
  - Depends on the layers beneath it
  - Is independent of the layers on top of it, having no knowledge of the layers using it
- Example: Wordpress
- Standard usage: Standard apps for quick development with fewer and inexperienced developers; Apps with strict testability and maintainability standards



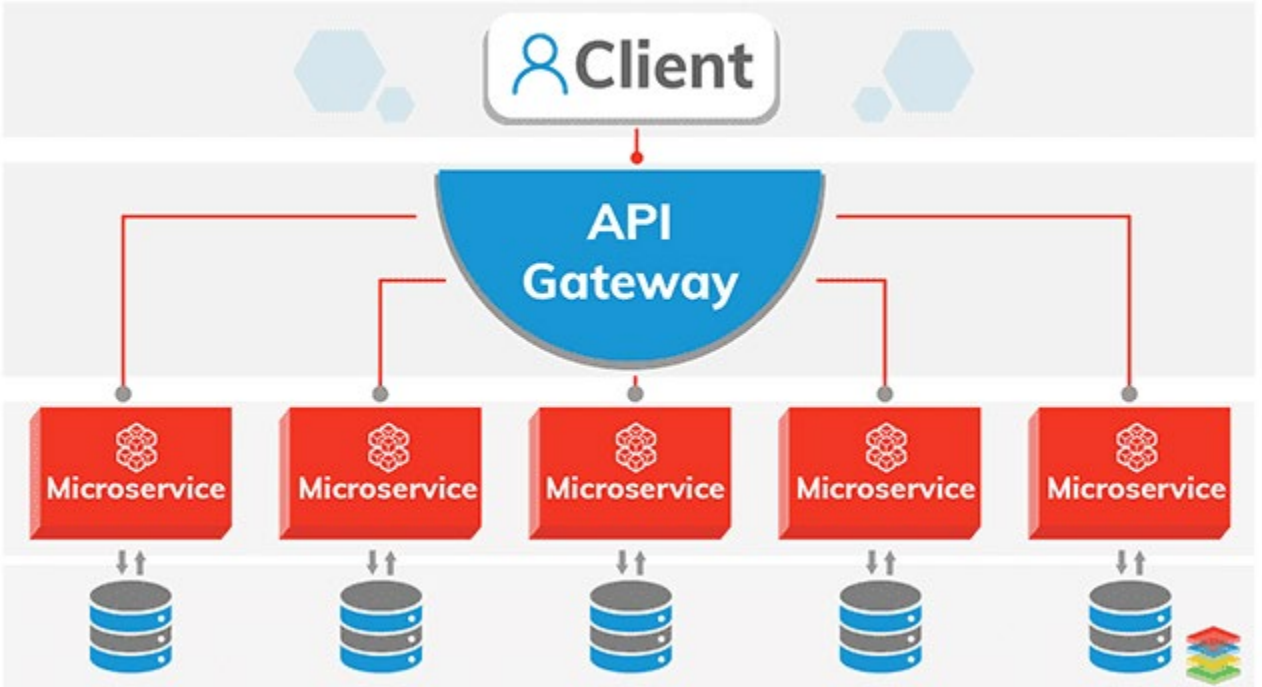
Microkernel Architecture

- separated into an extended functionality (plug-ins) and a minimal functional core comprising standard business logic without any custom code for complex conditional processes or exceptional cases.
- plug-ins consist of independent components that support the core by offering specialized processing added features
- best suited for apps that need to be adaptive and flexible enough to frequently changing system requirements.
- best usage - Task and job scheduling apps, Workflow apps, Apps that incorporate data from various sources, transform the data and send it to different destinations

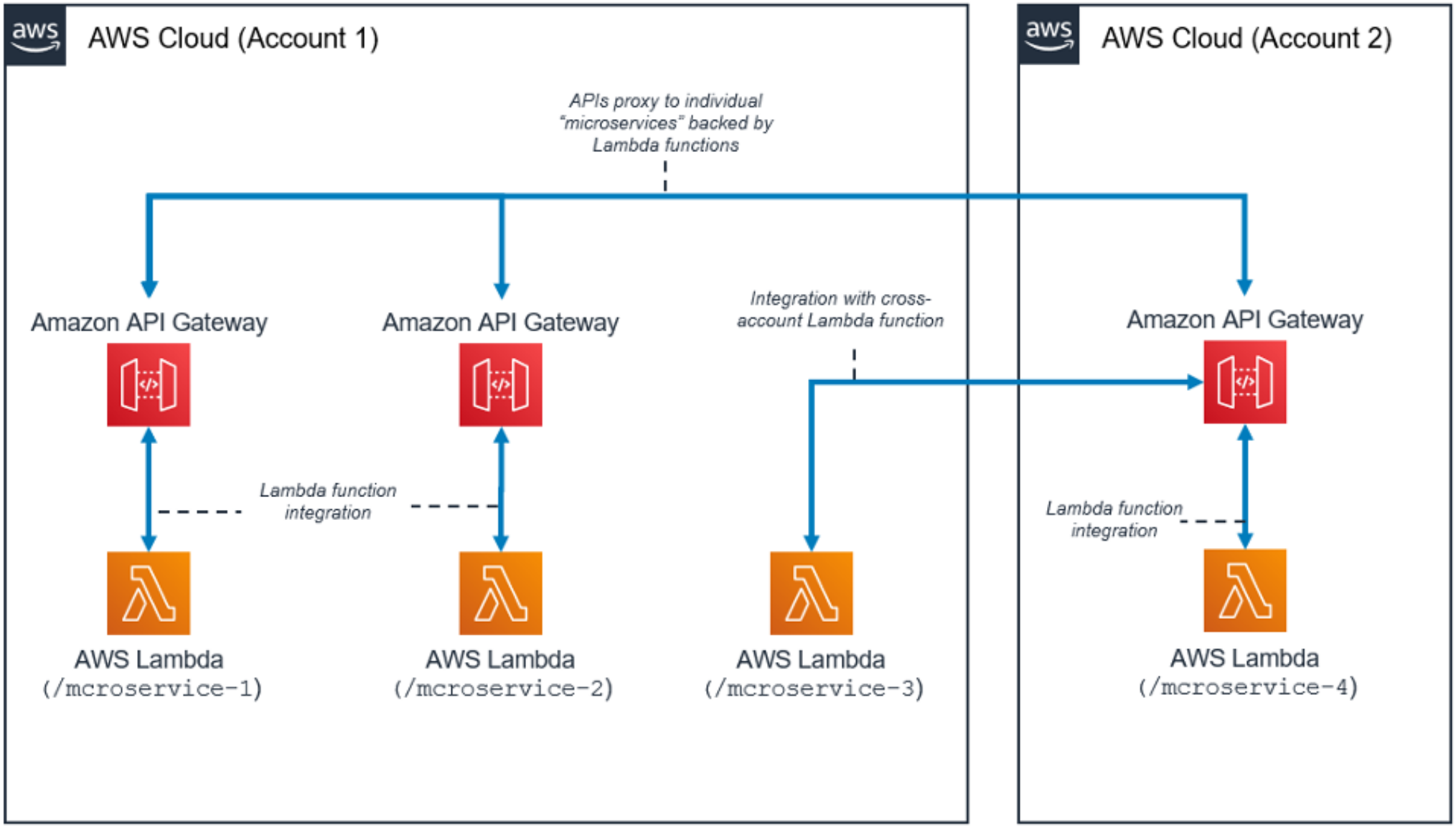


Microservices

- approaches the building of multiple small and independent apps that work together under an entire system. Each app or microservice has its own responsibility, and the only dependence between them is to communicate.
- Allows easy scalability of development
- Best usage: Fast developing web and business apps; Corporate data hubs with well-established boundaries; Websites with small components

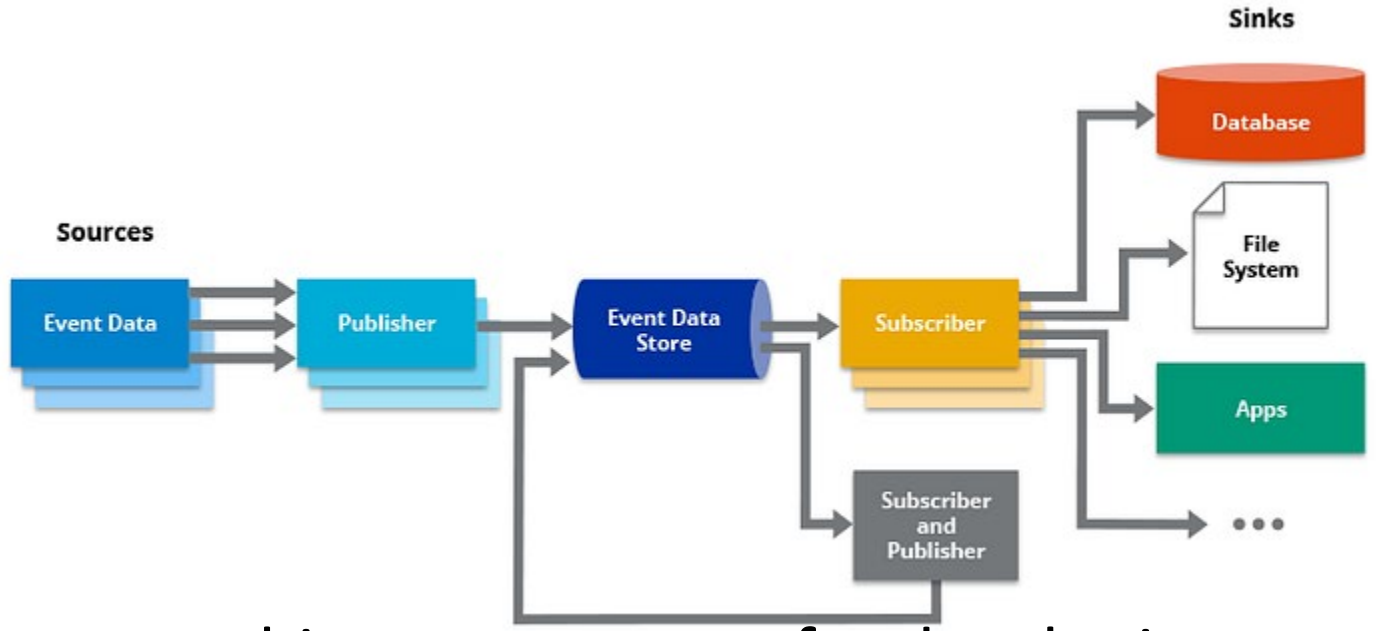


Microservices example: AWS Cloud & Netflix



https://docs.aws.amazon.com/whitepapers/latest/serverless-multi-tier-architectures-api-gateway-lambda/microservices-with-lambda.html

Event-Driven Architecture (EDA)



- most commonly allocated asynchronous architecture pattern for developing highly scalable systems.
- Its approach is based on data that defines ‘events’, such as moving the scroll bar, clicking a button, etc., and processes them asynchronously.
- involves single-purpose event processing elements that build a central unit. The central unit then accepts all data and assigns it to separate modules handling the specific type.
- Best usage: User interfaces; Apps that have asynchronous data flow; Complex apps that require seamless data flow and would eventually grow

EDA Example: e-commerce website

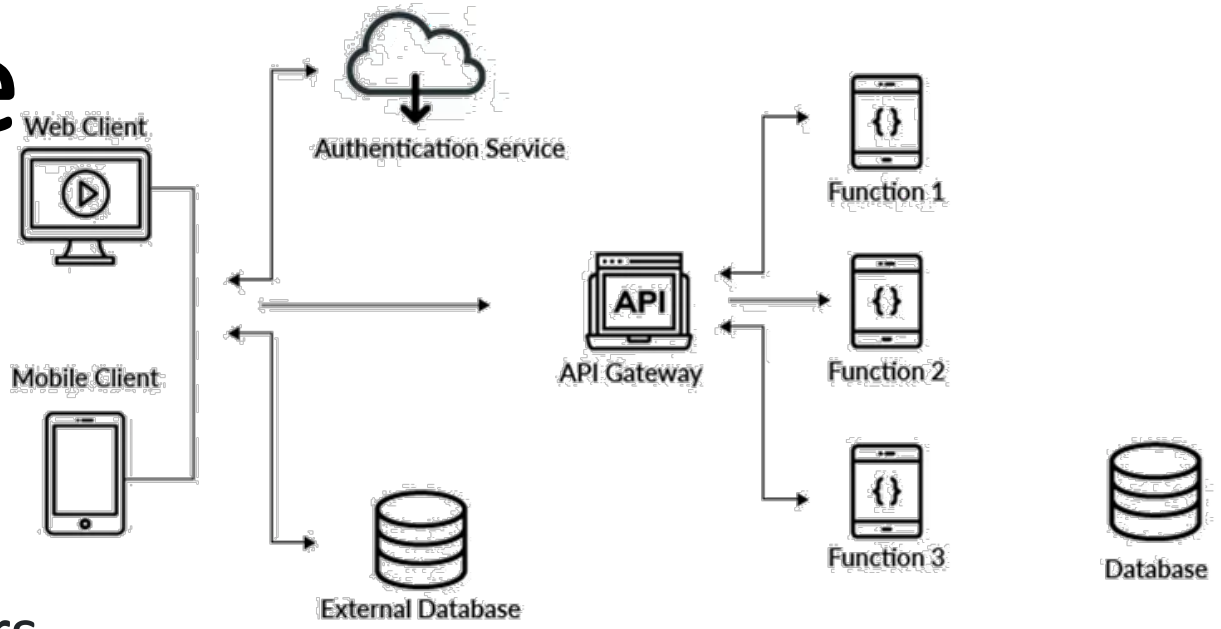
A good example is an e-commerce site.

- Enables the e-commerce website to react to various sources at a time of high demand.
- Simultaneously, it avoids any crash of the application or any over-provisioning of resources.

Usage:

- For applications where individual data blocks interact with only a few modules.
- Helps with user interfaces.

Serverless Architecture



- Allows developers to build and run services without having to manage the underlying infrastructure.
- Developers can write and deploy code, while a cloud provider provisions servers to run their applications, databases, and storage systems at any scale.

- Serverless functions are triggered by events, such as HTTP requests, database changes, or file uploads. They execute in response to these events, making serverless architecture inherently event-driven and highly responsive.
- E.g. is AWS Lambda, Google Cloud Functions (GAE Standard) and Azure Functions

What makes a good software architecture?

1. **Functionality:** The extent the software performs against its needed purposes.
2. **Usability:** The level the software can be used with ease and convenience.
3. **Reliability:** The capability of the product to provide intended functionality under given circumstances.
4. **Supportability:** The facility with which developers can transfer the software from one platform to another with minimal or no changes.
5. **Performance:** The approximation by considering resource utilization, processing speed, response time, productivity, and throughput.
6. **Self-Reliance:** The capability of independent activities for optimal performance even if one is going through a downtime.

Leveraging AI for Software architecture

Architectural Visualization

- AI tools can automatically generate visual representations of complex architectures **from code**, helping developers understand and improve system design - detect bottlenecks, dependencies, or design flaws
- AI-generated architecture diagrams **from prompts** (**Lucidchart, Eraser.io**) significantly reduces the time spent on tech solution design but also provides architects with a foundational blueprint to kickstart their design.

Real-Time Monitoring

- AI-based tools (**AWS X-ray, Dynatrace**) continuously monitor the system's architecture to detect drift and architectural degradation.
- These tools can provide real-time alerts and recommendations for corrective actions when architectural principles are violated, helping to maintain consistency.

Leveraging AI for Software architecture

Legacy Systems

- AI (**Grok, Cast AI**) can help refactor and upgrade legacy systems by identifying outdated patterns and suggesting modern design solutions.
- Legacy applications are often hard to maintain due to their monolithic design.
- AI tools can analyze the code and provide recommendations for migrating to modern architectures like microservices or serverless.

Code Insights

- AI can automatically detect code smells and restructure messy codebases to make them more maintainable and efficient.
- For example, tools like **DeepCode** or **Codex** help refactor code to improve readability and performance.

Leveraging AI for Software architecture

Other insights:

https://medium.com/inspiredbrilliance/generative-ai-in-software-architecture-dont-replace-your-architects-yet-cde0c5d462c5

https://www.linkedin.com/pulse/generative-ai-software-architecture-design-lev-z-/

Software Architecture Quizizz