Announcements

- CATME Post Project 1 Survey Released
- 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 5
- Extra Credit Opportunity today

CS3300 Introduction to Software Engineering

Lecture 11: Software Architecture

Dr. Nimisha Roy ▸ nroy9@gatech.edu

Slides adapted from Alessandro Orso

What is Software Architecture?

Perry and Wolf
SWA = { Elements, Form, Rationale}

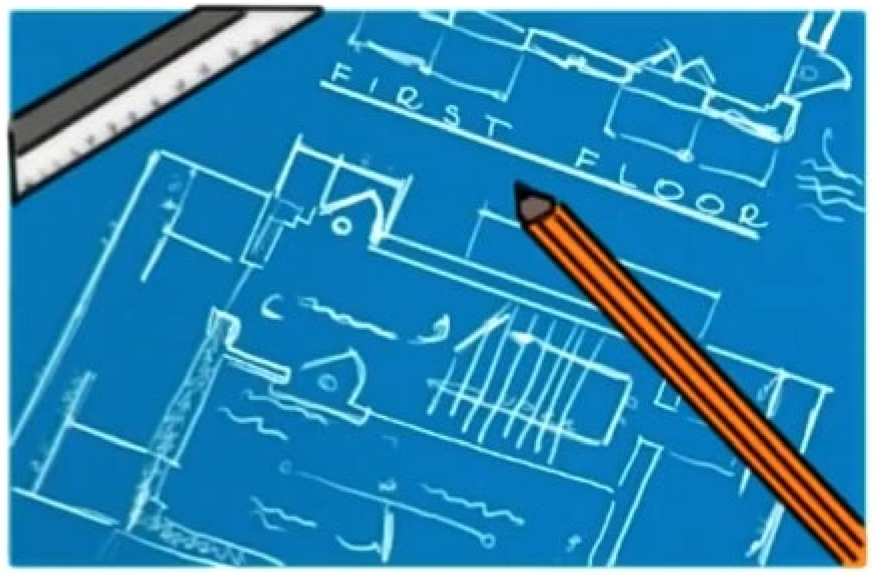
What (processes, data, connectors) ; How (properties, relationship between elements) ; Why (justification for elements and relationships)

Shaw and Garland
SWA = [is a level of design that] involves

- Description of elements from which systems are built
- Interactions among those elements
- Patterns that guide their composition
- Constraints on these patterns

A general definition of SWA

Set of principal design decisions about the system



Blueprint of a software system

- Structure
- Behavior
- Interaction
- Nonfunctional properties

Temporal Aspect

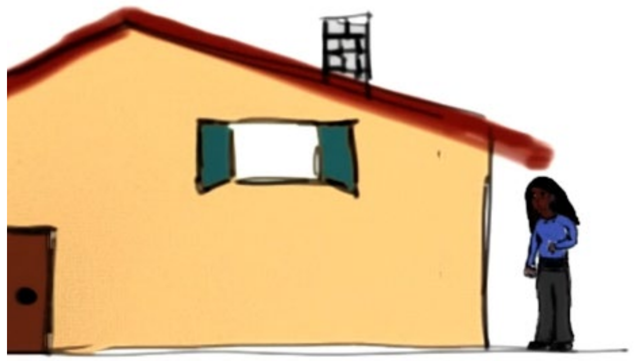
A SWA is not defined at once, but iteratively, over time

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

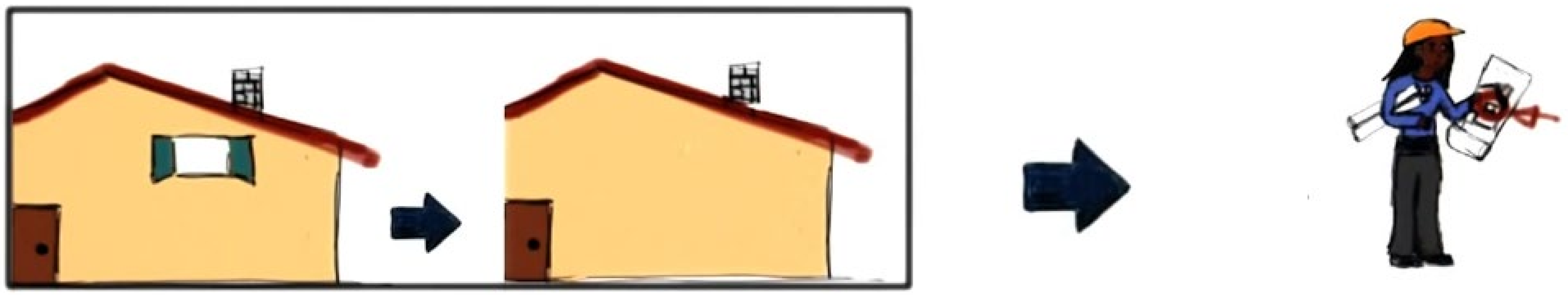
A prescriptive architecture captures the design decisions made prior to the system's construction
=> as- conceived SWA



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

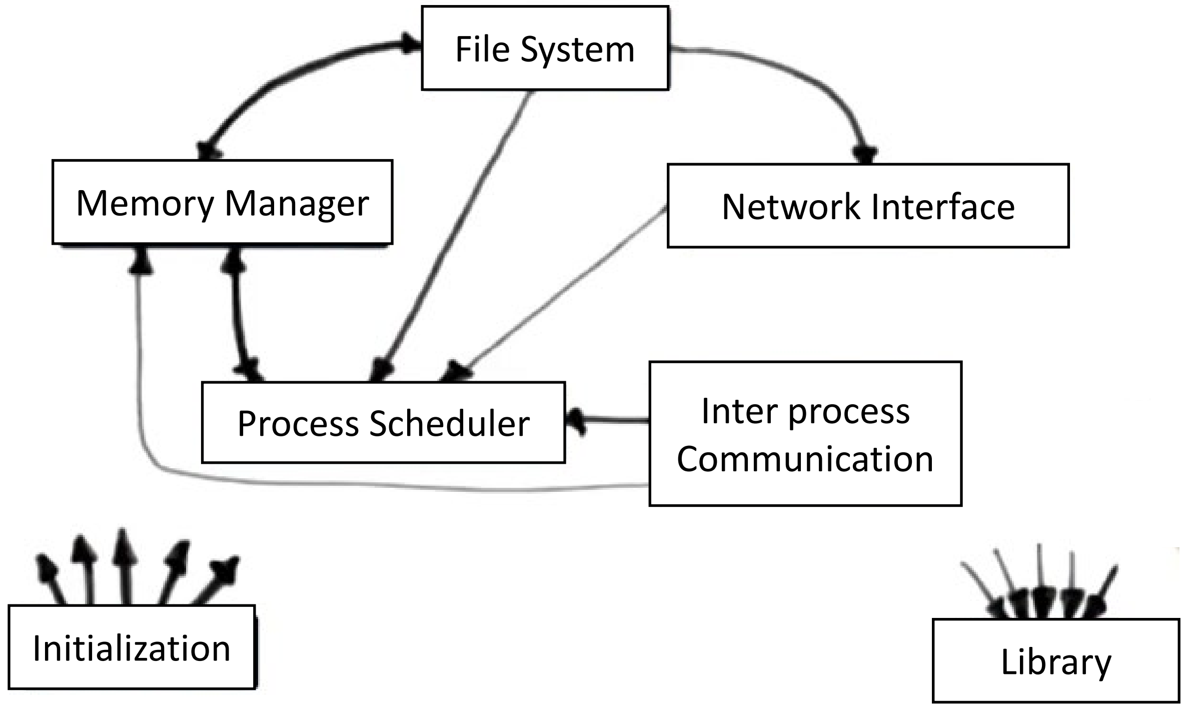
Architectural Recovery

Drift and Erosion => Degraded architecture

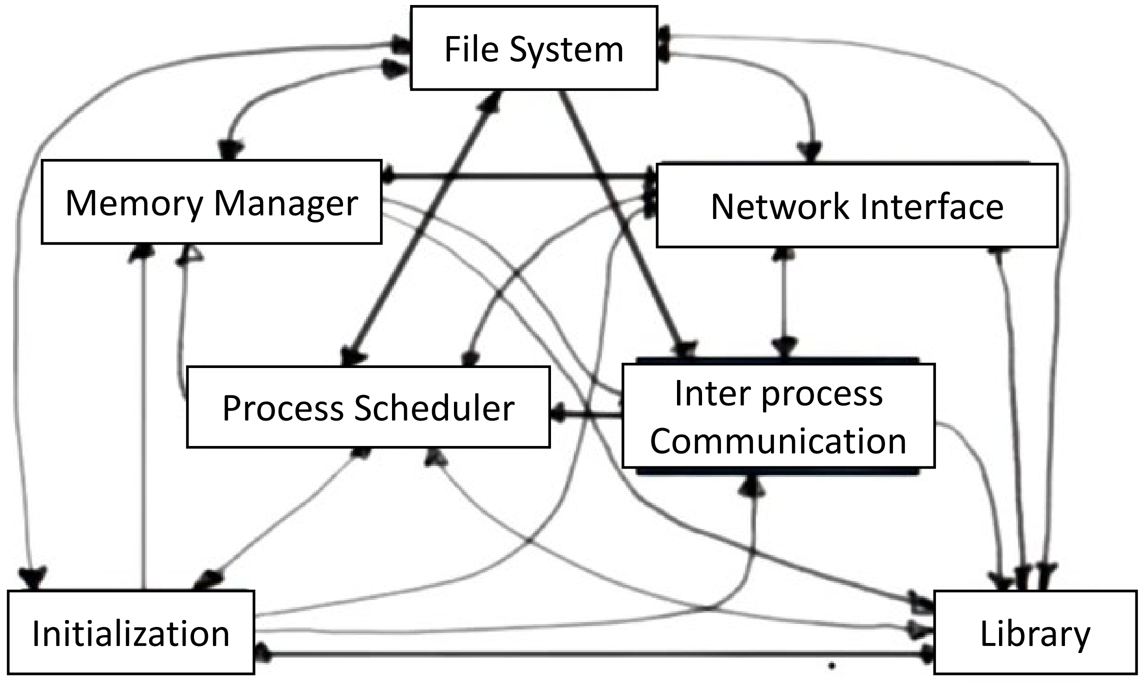
Keep tweaking the code (typically disastrous)

Architectural recovery: determine SWA from implementation and fix it

An example from the Linux Kernel



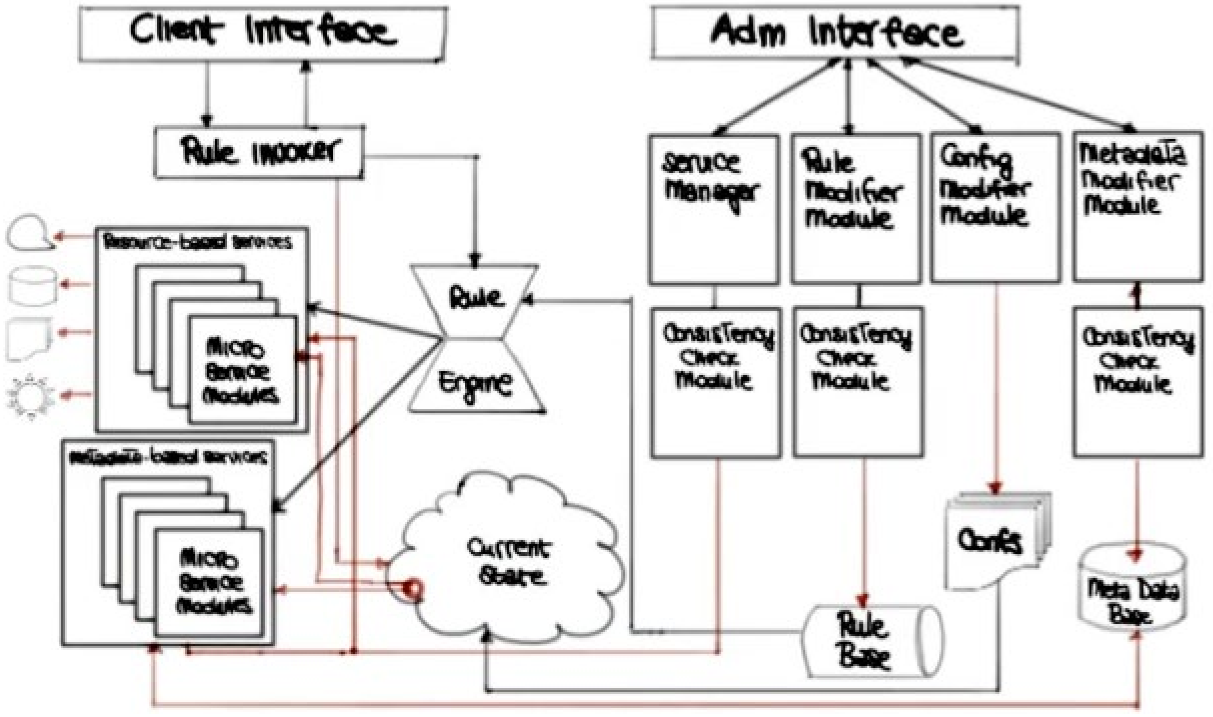
Prescriptive Architecture



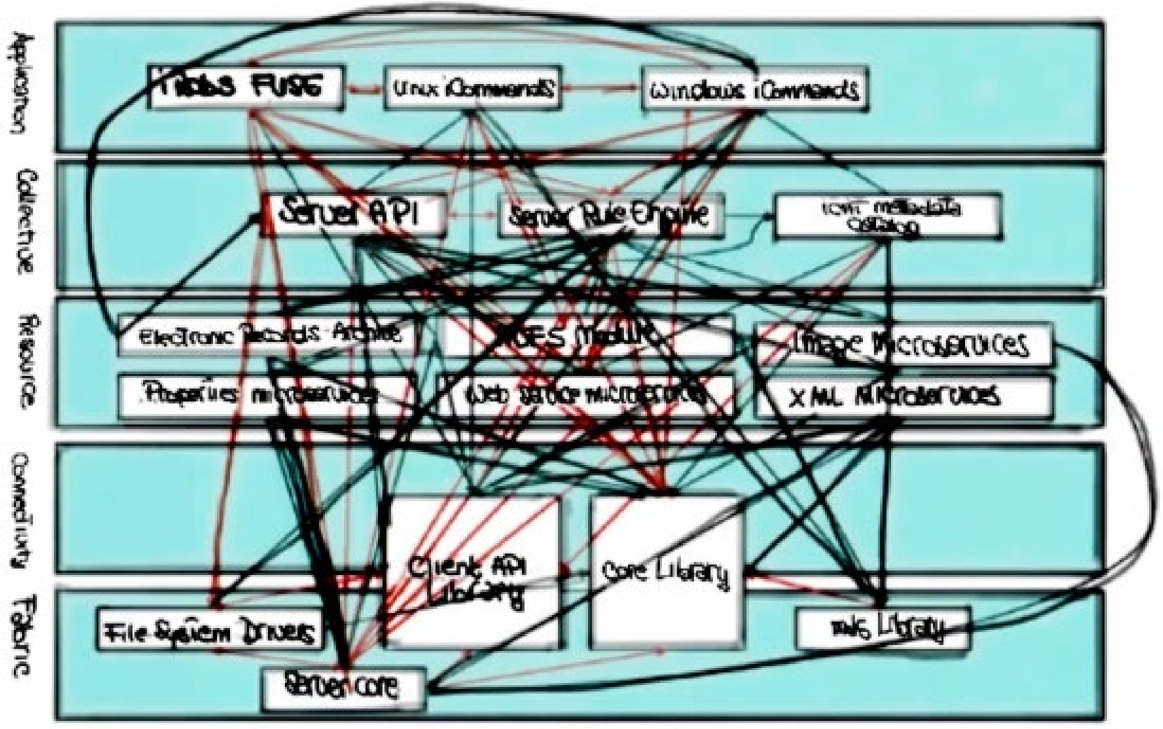
Descriptive Architecture

Another example: iRODS

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



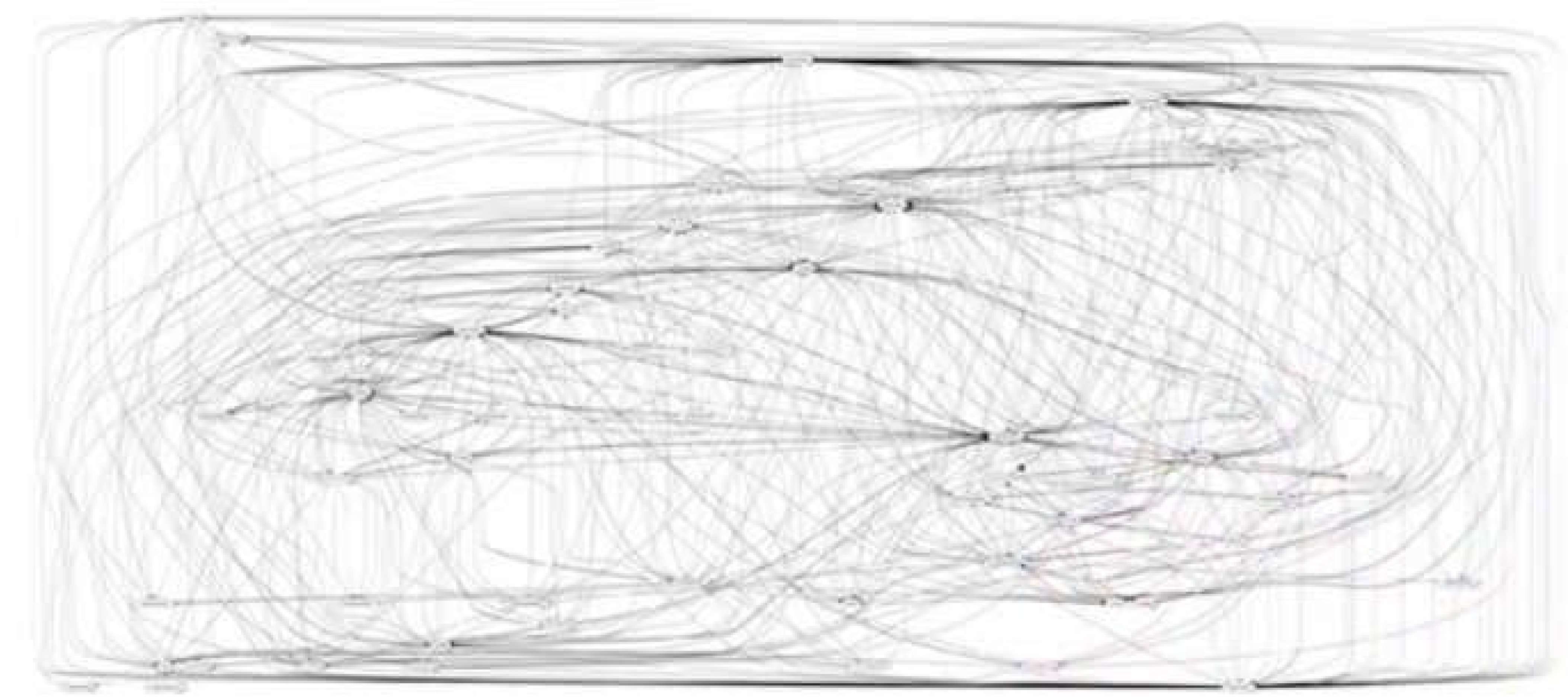
Prescriptive Architecture



Descriptive Architecture

More examples: Hadoop

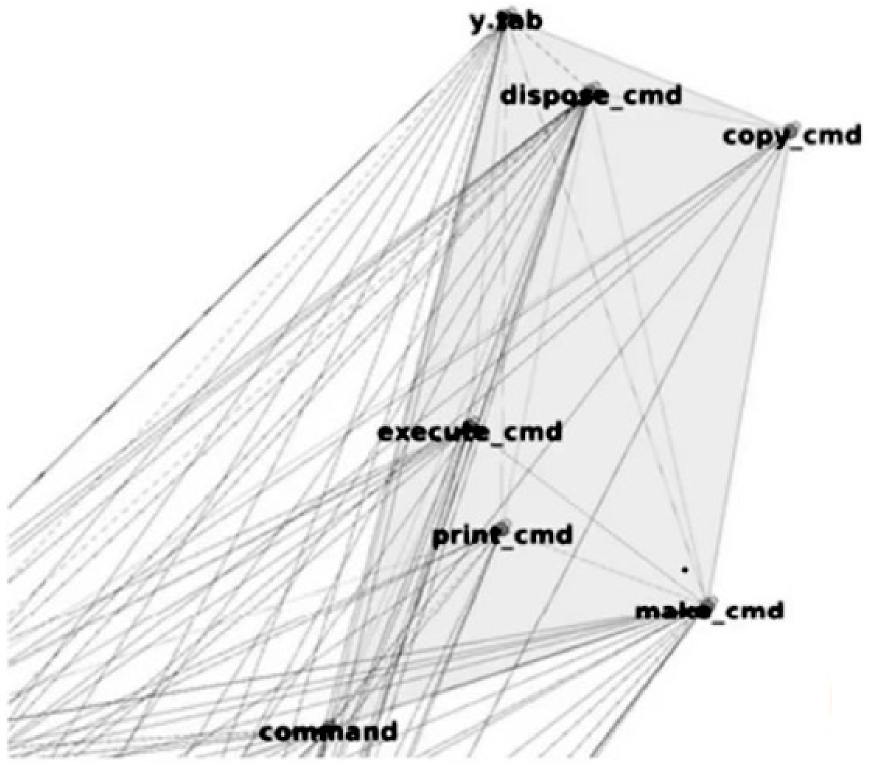
Open-source software framework for storage and large-scale processing of data sets



Descriptive Architecture

Final example: Bash

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

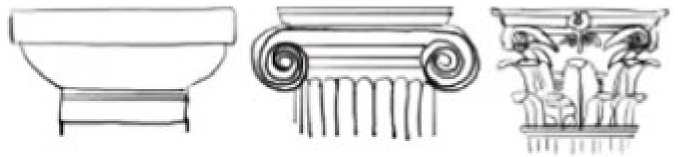


Lack of cohesion in the component

High coupling among components

Descriptive Architecture of the command component of Bash.

Architectural Styles



An architectural style 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.

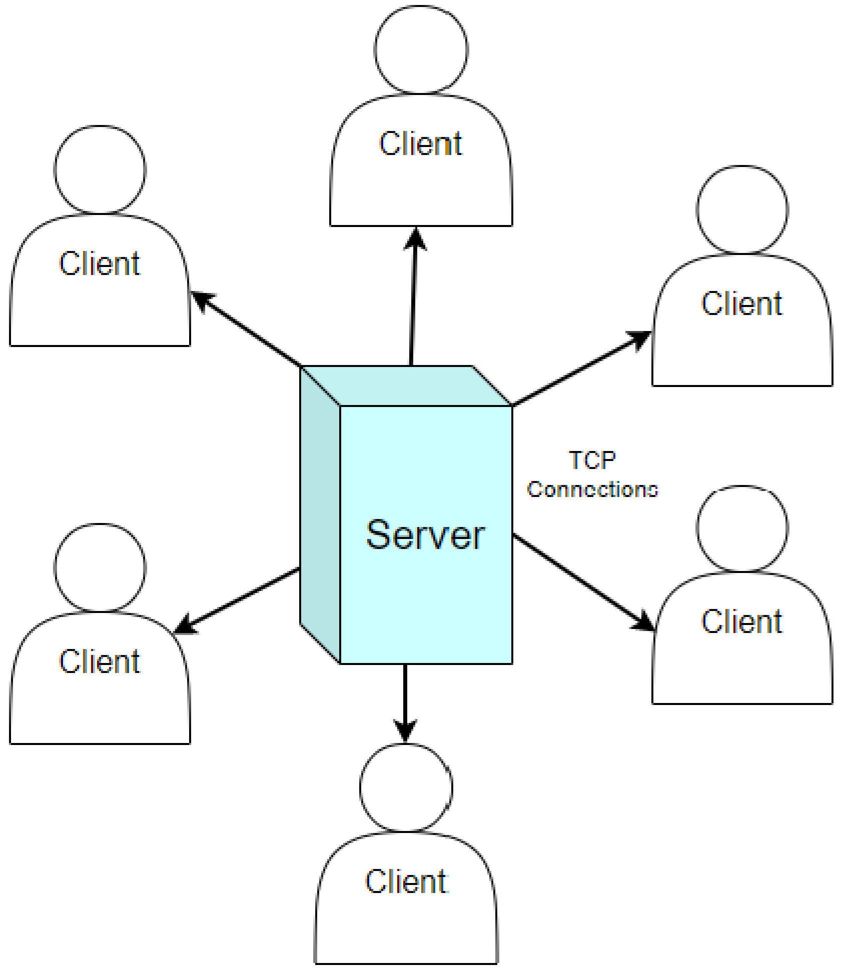
Top Architectural Styles

- N-Tier
  - Client – Server
- Layered
- Peer-to-Peer
- Pipe and Filter
- Implicit Invocation/Event-Driven
- Microservices

Client-Server/ N-Tier Architecture

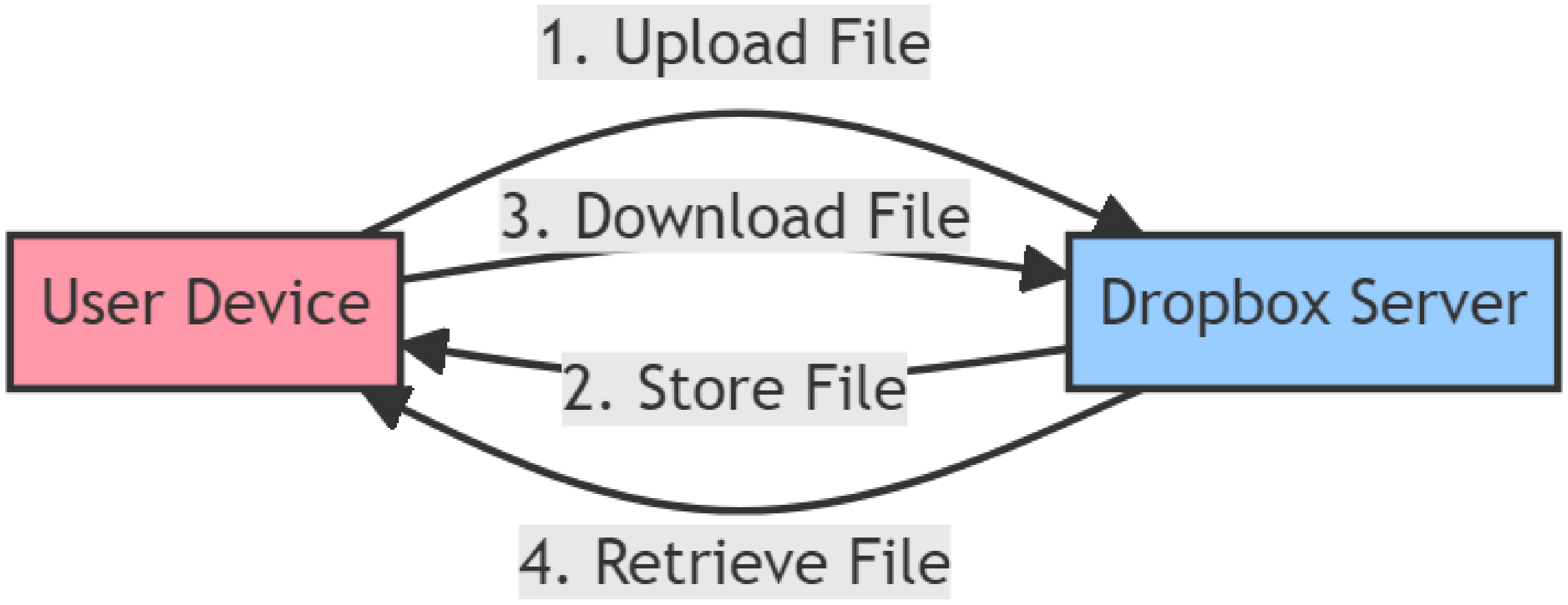
Client-Server Architecture

- A system is organized as set of services and associated servers, and clients that access and use the services
- Major components:
  - Set of servers that offer services to other components, e.g., *print servers, file servers, email servers*
  - Set of clients that call on the services offered by servers
  - A network that allows the clients to access these services



https://www.educative.io/blog/software-architecture-diagramming-and-patterns

Client-Server Architecture Example: Dropbox

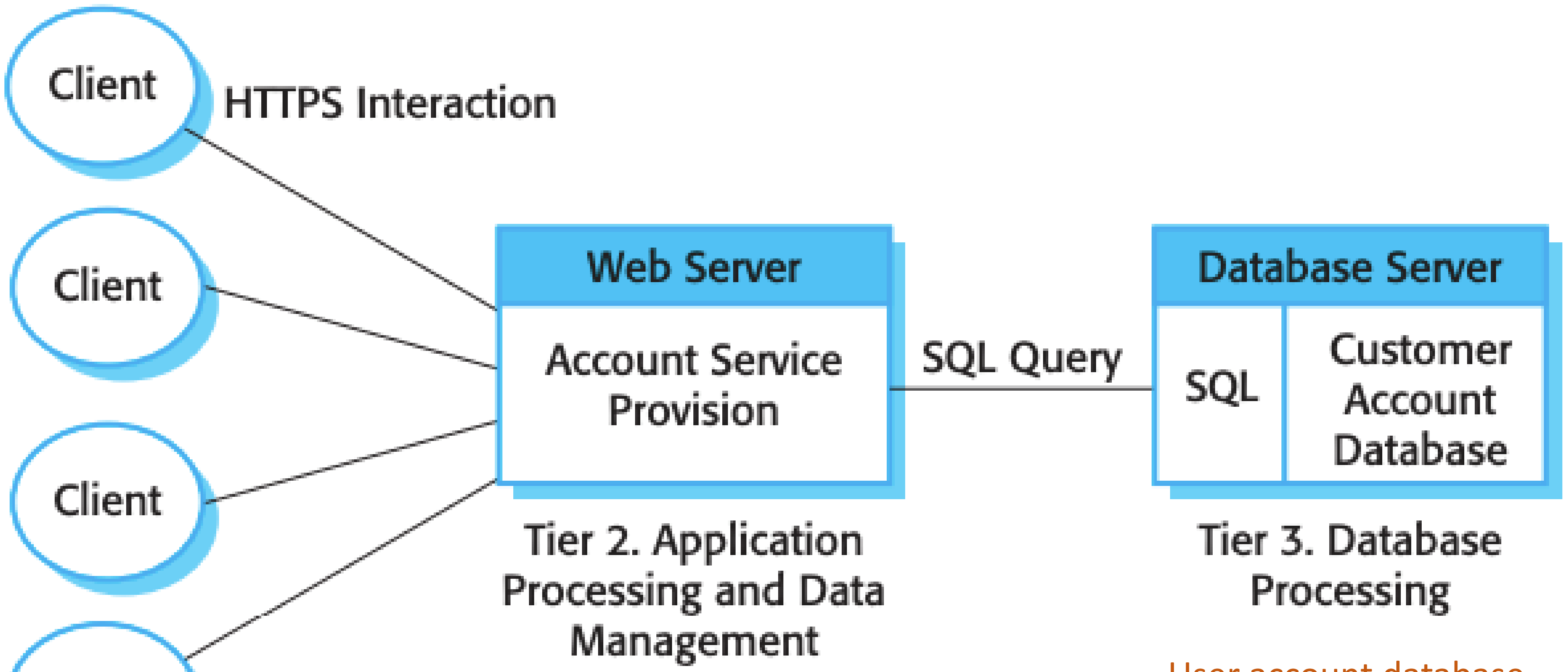


- User device (integrated UI) is the client
- Dropbox is the server
- This architectural style is monolithic, meaning a single executable performs all of the server-side functions for the application

- Does the Client and Server have to be on separate hardware?

- Client-Server is a 2-tier architectural style
- Can be implemented on a single computer; but should be different processors
- Tier vs layer:
    - tier – physical separation;
    - layer – logical separation

3-tier Architecture Example: an Internet banking system


Client
HTTPS Interaction
Client
Client
Client
Web Server
Account Service Provision
SQL Query
Database Server
SQL
Customer Account Database
Tier 2. Application Processing and Data Management
Tier 3. Database Processing

Login page, account summery, settings, bill payment

facilities to transfer cash, generate statements, pay bills

User account database, transaction database, historical records

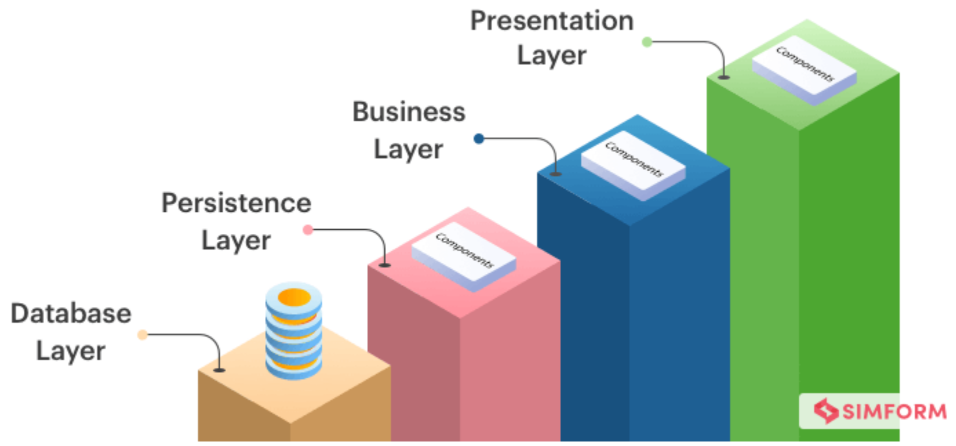
Client-server is a 2-tier architectural style where there is no Business logic layer or immediate layer in between client and server.

The three-tier model can be extended to a multi-tier variant, where additional servers are added to the system. This may involve using a web server for data management and separate servers for application processing and database services.

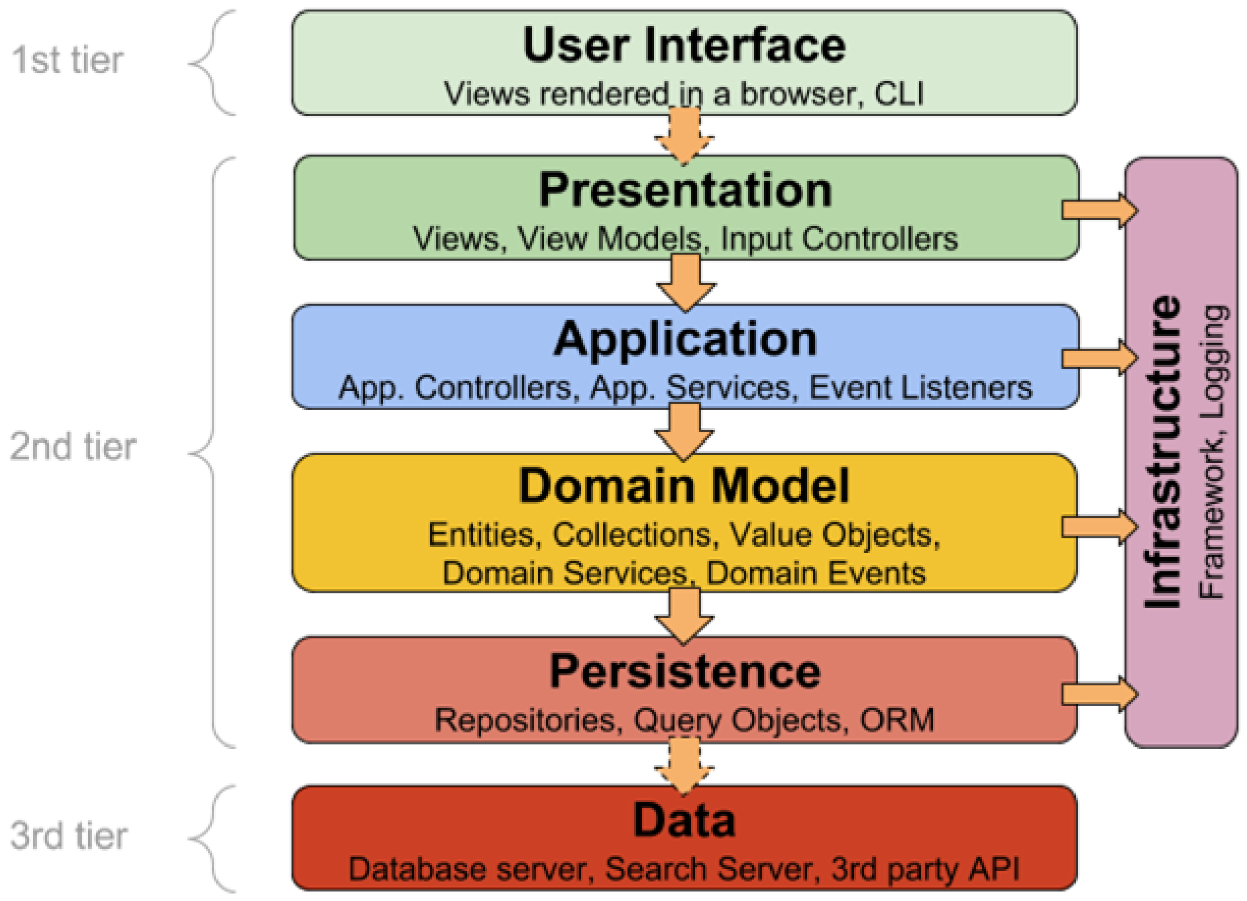
Layered Architecture

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



Layered Architecture



1st tier
2nd tier
3rd tier
User Interface
Views rendered in a browser, CLI
Presentation
Views, View Models, Input Controllers
Application
App. Controllers, App. Services, Event Listeners
Domain Model
Entities, Collections, Value Objects, Domain Services, Domain Events
Persistence
Repositories, Query Objects, ORM
Data
Database server, Search Server, 3rd party API
Infrastructure
Framework, Logging

https://herbertograca.com/2017/08/03/layered-architecture/

- 3-Tiered AND 6-Layered Architecture
  - tier – physical separation; layer – logical separation
- A native browser application, rendering and running the **user interface**, sending requests to the server application;
- An **application server**, containing the presentation layer, the application layer, the domain layer, and the persistence layer;
- A **database server**, which would be used by the application server for the persistence of data.
- It is a scalable solution and solves the problem of updating the clients as the user interface lives and is compiled on the server, although it is rendered and run on the client browser.

Layered Architecture: Pros and Cons

- Advantages:
  - We only need to understand the layers beneath the one we are working on
  - Each layer is replaceable by an equivalent implementation, with no impact on the other layers
  - A layer can be used by several different higher-level layers
- Disadvantages:
  - Unorganized source code
  - Coupling of code over time
  - Considerable amount of time to build/test/deploy

Peer-to-Peer Architecture

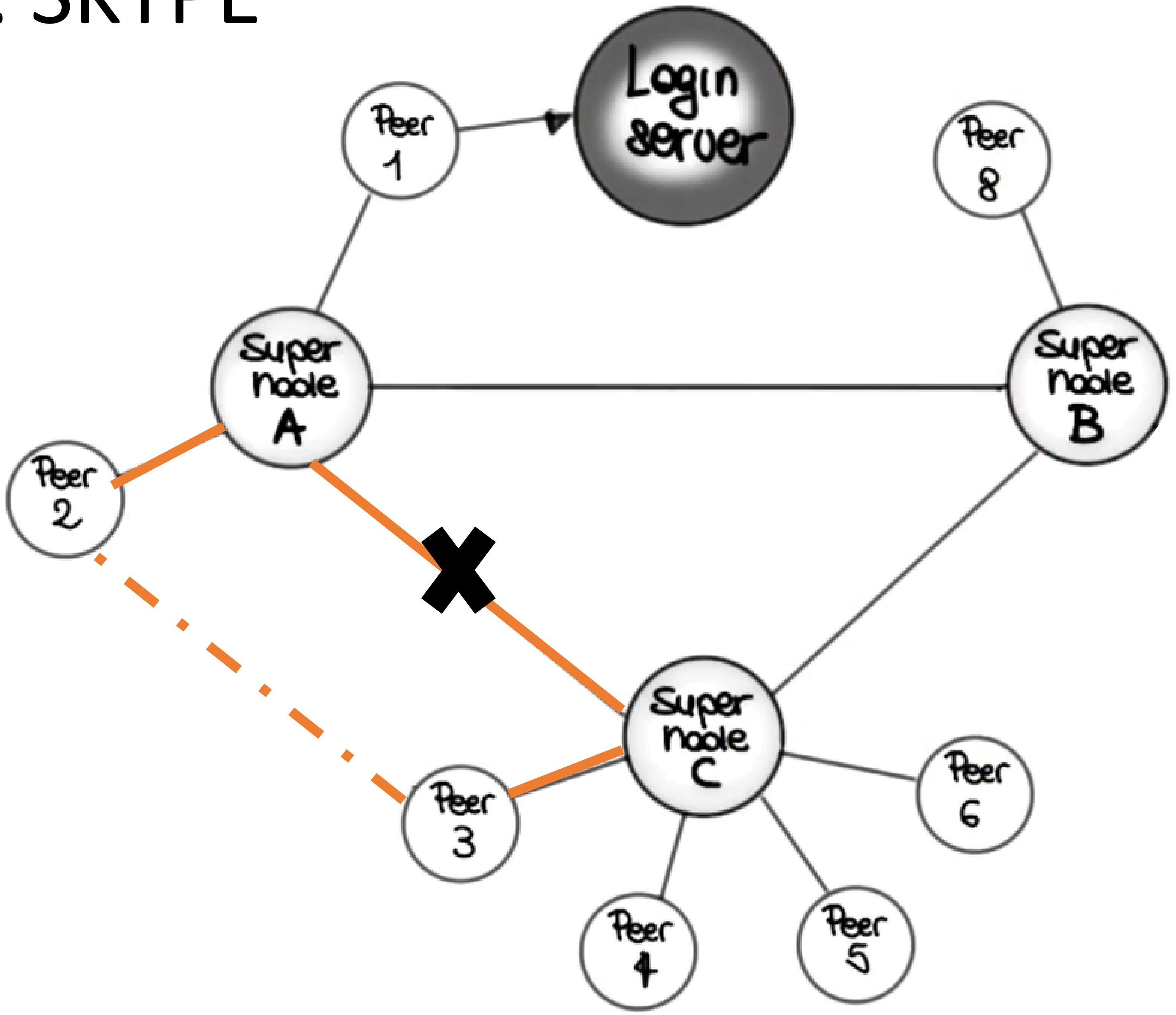
Peer-to-Peer Architecture

- Peer-to-peer (P2P) systems are decentralized systems in which computations can be carried out by any node on the network
- No clear distinction between clients and servers
- Examples: Skype, Gnutella and BitTorrent, ICQ and Jabber, SETI@home

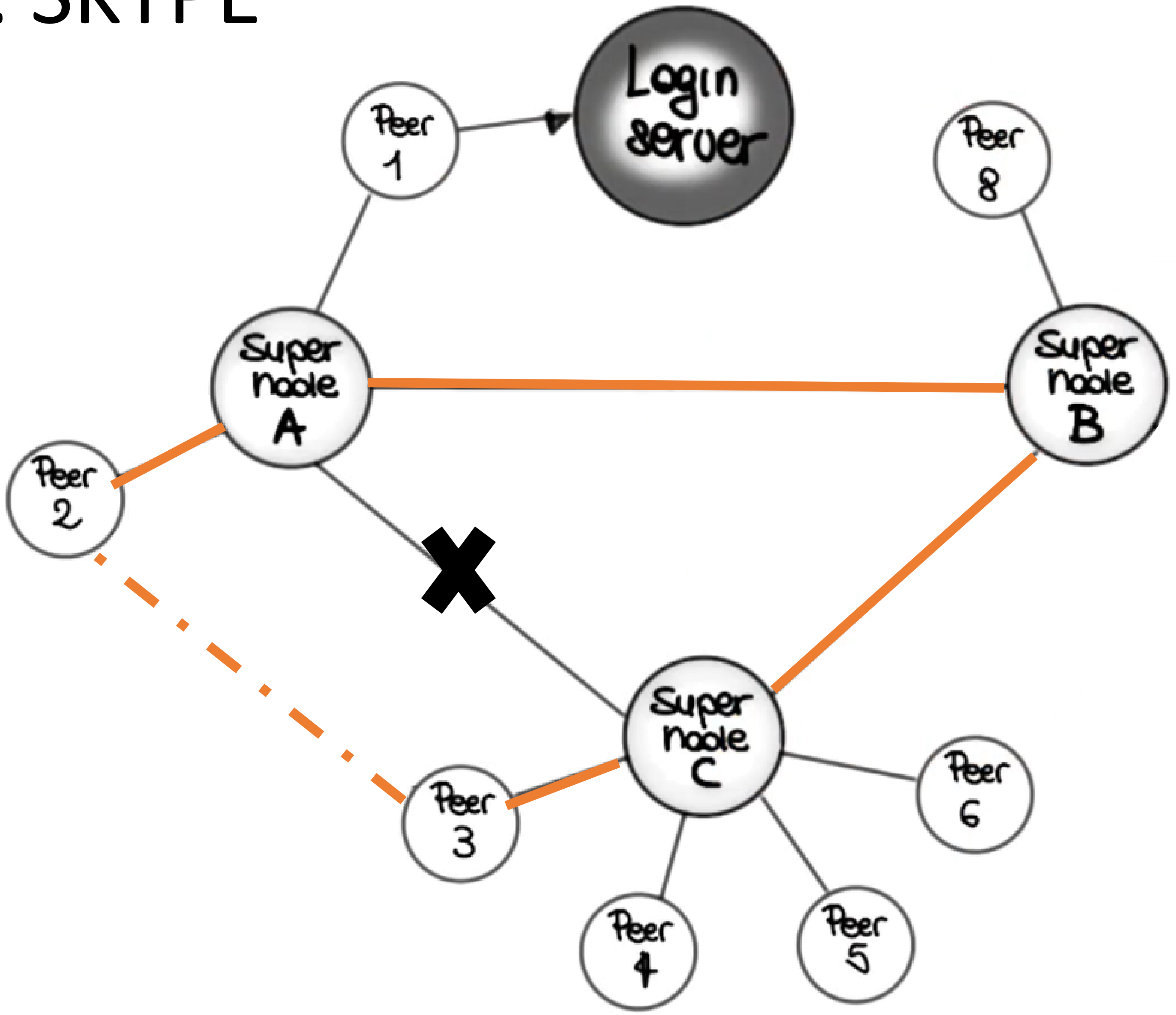
Peer-to-Peer Architecture: Pro and Con

- It is appropriate in two circumstances:
  - Where the system is **computationally intensive**, and it is possible to separate the processing required into a large number of **independent** computations
  - Where the system involves the **exchange of information** between individual computers on a network and there is no need for this information to be centrally stored or managed
- Con
  - Achieving robust security is challenging.
  - Performance depends on the number of nodes connected to the network.
  - No way to backup files or folders.

Example: SKYPE

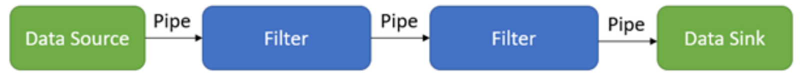


Example: SKYPE



Pipe and Filter Architecture

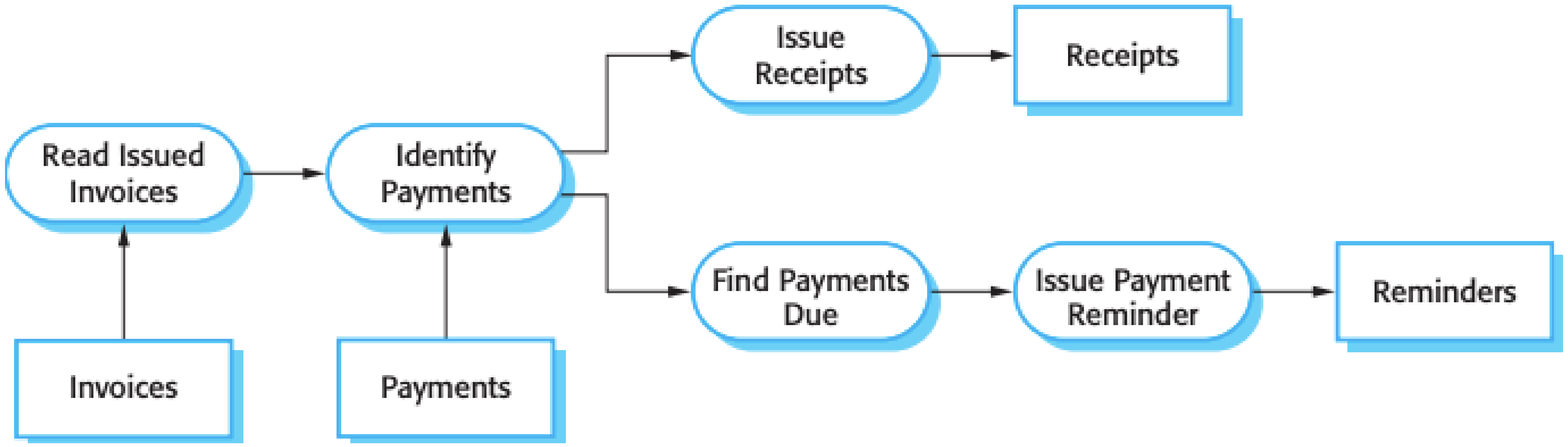
Pipe and Filter Architecture



- The pipe and filter architecture style is data-centric, structured around how data flows through the application. In this architecture, applications employ a pipeline as follows:
  - Firstly, the application takes in data as input.
  - Next, a series of transformations on the data are applied sequentially.
  - Finally, the application returns the processed data as output.
- Name 'pipe and filter' comes from Unix where it is possible to link processes using 'pipes'

https://jhocx.medium.com/pipe-and-filter-software-architecture-87eb6cee8c86

Pipe and Filter Example: Invoice



An organization has issued invoices to customers. Once a week, payments that have been made are reconciled with the invoices. For those invoices that have been paid, a receipt is issued. For those invoices that have not been paid within the allowed payment time, a reminder is issued.

Pipe and Filter: Pros and Cons

Advantages

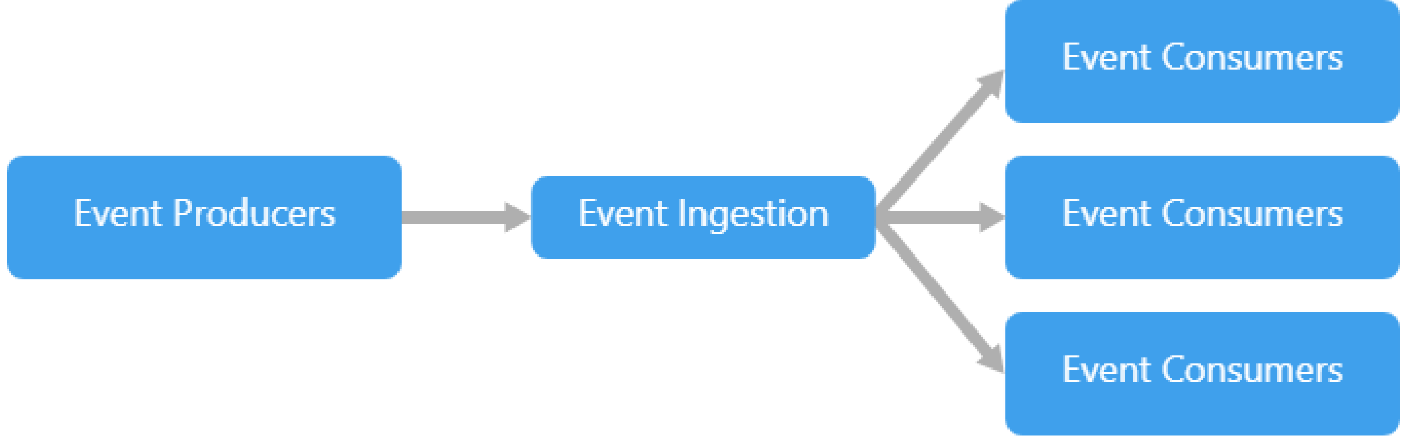
- Can be used for applications facilitating a simple, one-way data processing and transformation.
- Applications using tools like Electronic Data Interchange and External Dynamic List.
- To perform advanced operations in Operating Systems like UNIX, where the output and input of programs are connected in a sequence.

Disadvantages

- There can be a loss of data in between filters if the infrastructure design is not reliable.
- The slowest filter limits the performance and efficiency of the entire architecture.
- System overhead increases due to multiple parsing and unparsing data with complex I/O formats

Implicit Invocation/Event-Driven Architecture (EDA)

Implicit Invocation/ Event-Driven Architecture (EDA)



- An event-driven architecture consists of event producers that generate a stream of events, and event consumers that listen for the events.
- Traditionally, components interact with each other by explicitly invoking routines
- With EDA, instead of invoking a procedure directly, a component can announce (broadcast) one or more events. Other components register an interest in an event. When the event is announced, the system invokes all the interested procedures.
- Should be used when multiple subsystems must process the same events and Real-time processing is needed with minimum time lag. e.g. GUI

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.

Microservices

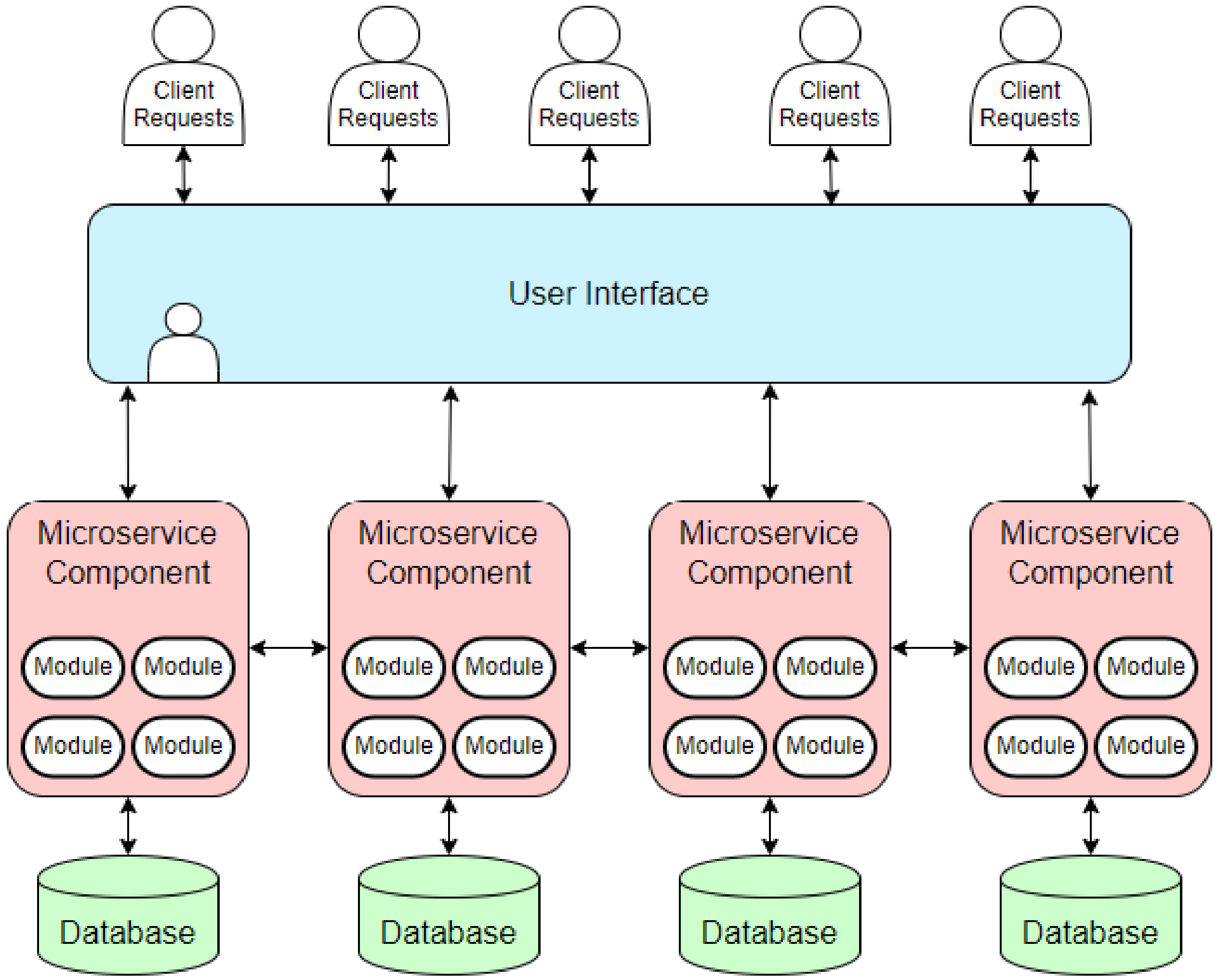
Microservices

One of the most popular software trends at the moment

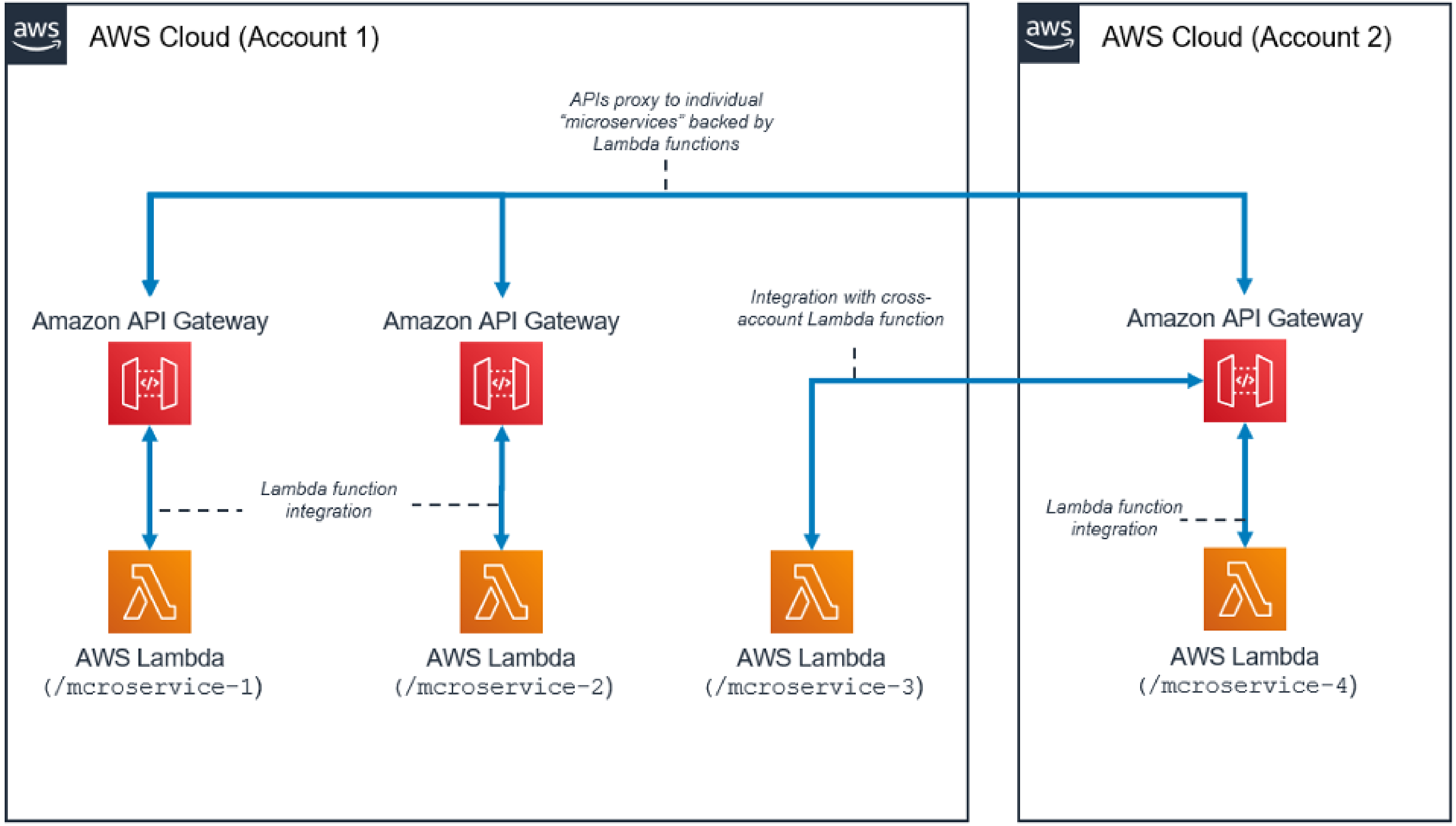
consists of groups of small, independent, self-contained services with small code bases (not monolithic)

Minimizes number of dependencies

Allows easy scalability of development



Microservices example: AWS Cloud & Netflix


AWS Cloud (Account 1)
AWS Cloud (Account 2)
APIs proxy to individual "microservices" backed by Lambda functions
Amazon API Gateway
Amazon API Gateway
Integration with cross-account Lambda function
Amazon API Gateway
Lambda function integration
Lambda function integration
AWS Lambda (/mcroservice-1)
AWS Lambda (/mcroservice-2)
AWS Lambda (/mcroservice-3)
AWS Lambda (/mcroservice-4)

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

Architectural Styles: Summary

- **N-Tier (Client – Server)** - different layers of the system (presentation, application processing, data management, and database) are separate processes that execute on different processors
- **Layered -** Organizes the system into layers with related functionality/logic associated with each layer
- **Peer-to-Peer -** decentralized systems in which computations can be carried out by any node on the network
- **Pipe and Filter -** data-centric style structured around how data flows through the application and transformations are performed on it.
- **Event-Driven-** consists of event producers that generate a stream of events, and event consumers that listen for the events.
- **Microservices-** groups of small, independent, self-contained services

Software Architecture Quizizz