THE UNIVERSITY *of* EDINBURGH

Distributed Systems Fall 2024

Yuvraj Patel

Who am I?

Lecturer (Assistant Professor) in ICSA, School of Informatics

Ph.D. from the University of Wisconsin, Madison, USA

- Thesis: Fair and Secure Synchronization for Non-Cooperative Concurrent Systems
- Ph.D. Minor in Psychology

Research Area - Operating Systems (Classical & Quantum Computing), Storage & File Systems, Concurrency, Security, Distributed Systems

Who am I (contd..)?

Prior to Grad School:

Spent 9 years writing Operating System and File System code

Code written by me executes thousands of times every second

- Bank transactions
- Car production – BMW, Daimler Chrysler
- Movie production – Avatar
- Large Hadron Collider and many more….

Call Me

Yuvraj or Yuvi or UV (but not IR)

Please do not address me as

- Professor/Prof. Yuvraj Prof. Patel/ Dr. Patel

Today's Agenda

What will you do in this course?

- **How will you be successful?**

Understand what distributed systems are? What are the different architectures for distributed systems?

Read

Read the recommended material comprising

- Book: Distributed Systems by Maarten Van Steen & Andrew S. Tanenbaum (Online free book available)
- Book: Distributed systems concepts and design by George Coulouris, Jean Dollimore, Tim Kindberg & Gordon Blair (Book available online through Library)
- Papers cited on the Schedule Page

Reading will help you improve your thought process

Start Coursework Promptly

Coursework (25% of final grade)

1 project involving building a distributed component

- Release date - 21/10/2024
- 4 weeks to work on the project
- Group Project
- Will have to spend 40 hours individually

Don’t Cheat: Academic Integrity

It is OK to:

- Discuss project or specification in general terms (when to return an error?)
- Discuss how different library routines/system calls work
- Ask TAs, Professor for help

It is NOT OK to:

- Use code samples for similar problems you may find on-line including stackoverflow, ChatGPT, etc.
- Bug someone else for a lot of help
- Share your code directly with other people/project groups
- Post your code in a public place

We will run tools to check for similar code across groups/individuals

Exam

Exam (worth 75% of the final grade)

- Assess Distributed Systems concepts discussed in the class
- One final exam
- Will share more details towards the end of the semester

Learn through

- Lecture Material
- Coursework
- Piazza discussion
- Group study
- Discussion with others (which I highly recommend)

Ask for Additional Help

Two TAs

- Leping Li & Xueheng Wang
- Past students
- Office hours specifically during coursework

Leping Li
2nd year Ph.D. student
Research: Concurrency, OS
Tips to succeed: Read the course material; Attend the lectures; Do not hesitate to ask questions; Start the project early

Xueheng Wang
2nd year Ph.D. student
Research: Concurrency, DS, LLM
Tips to succeed: Always attend classes; Think more and ask more questions; Focus on understanding than memorizing; Start working on project early

Ask for Additional Help

Two TAs

- Leping Li & Xueheng Wang
- Past students
- Office hours specifically during coursework

Piazza

- Tends to be active and prompt
- Learn yourself and help others
- School will use Piazza participation to gauge engagement
- Consider Piazza as a knowledge repository
- Check Piazza regularly for course updates

My office hours ???

Course Outline

Introduction and overview - Need for distributed systems

Architecture & Communication - Scalability, Load balancing, Partitioning, RPC

Fault Tolerance - Failure models, Reliability, Recovery

Coordination - Ordering & Causality, Distributed transactions, Concurrency Control, Consensus/Agreement

Consistency & Replication - Epidemic algorithms, Consistency Models, Replica management

Distributed Storage - File systems, Large Scale systems

Issues - Energy/Power, Security, Local OS Support, Verification, Testing

Learning Outcomes

Develop an understanding of the principles of distributed systems and be able to demonstrate this by explaining them

Being able to give an account of the trade-offs which must be made when designing a distributed system, and make such trade-offs in you own designs

Develop practical skills of implementation of distributed algorithms in software so that you will be able to take an algorithm description and realize it in software

Being able to give an account of the models used to design distributed systems and to manipulate those models to reason about such systems

Being able to design efficient algorithms for distributed computing tasks

Today's Agenda

What will you do in this course?

- How will you be successful?

Understand what distributed systems are? What are the different architectures for distributed systems?

What is a Distributed System?

A distributed system

- Multiple computers (or nodes) communicate via a network
- Work together to achieve some task together/collectively
- Appears as a single coherent system to the users
- Failure of a node you didn't even know existed can render your own node unusable

Examples

- Distributed Systems are ubiquitous

Why study Distributed Systems?

Inherently distributed

- Either necessarily or sufficiently

For better reliability

- If one node fails, the system continues to work

For better performance

- Get things done faster; for example, due to replication (from a nearby datacenter)

To solve bigger and complex problems

- Single node cannot handle all the data/processing capacity, etc.
- Efficiently solve a problem

Why ”Not” make a Distributed System?

Communication can fail (network link, network partition, etc.)

Nodes can crash

Faults can happen randomly or in a coordinated fashion

Security & Privacy concerns

Perspectives on Distributed Systems

Architecture: common organizations

Process: what kind of processes, and their relationships

Communication: facilities/protocols for exchanging data

Coordination: application-independent algorithms

Naming: how to identify resources?

Consistency & Replication: how to replicate for better performance; manage different replicas

Fault Tolerance: keep the show running in the presence of failures

Security: ensure authorization/authentication of users

Design Goals

Support sharing of resources
- Achieve high efficiency, cost effective

Distribution transparency
- Hide the fact that the processes and resources are physically distributed across multiple nodes

Openness
- Components can easily be used or integrated into other systems

Dependability
- Degree that a computer system can be relied upon to operate as expected

Security
- Ensure confidentiality and integrity, provide trust

Scalability
- Scale well with more resources, users,

Resource Sharing

Sharing resources crucial to achieve effectiveness, manage costs

Examples – Cloud-based shared storage and files, peer-to-peer multimedia sharing and streaming (BitTorrent), shared web hosting

Distribution Transparency

Different types

- Access
- Location
- Relocation
- Migration
- Replication
- Concurrency
- Failure

Dependability Requirements

Four main requirements

- Availability – Readiness for usage
- Reliability – Continuity of service delivery
- Safety – Very low probability of catastrophes
- Maintainability – How easy can a failed system be repaired

Scale in Distributed Systems

Think scalability from three perspectives

- Size scalability – Number of users/processes/nodes
- Geographical scalability – Maximum distance between nodes
- Administrative scalability – Number of administrative domains

Issues with scalability

Size scalability

- Limited by computational/storage/network capacity
- Scale-up vs Scale-out

Geographical scalability

- Hard to go from LAN to WAN
- High latency an issue; some apps/protocols may not work
- Unreliable WAN links

Administrative scalability

- Conflicting policy concerning usage management, security, etc.
- Many systems do not suffer from issues – BitTorrent, Skype calls, Spotify

Techniques for scaling

Hide communication latency

- Use asynchronous communication; Delegate work (Edge servers)

Partitioning & Distribution

- Split data and spread across many nodes; Decentralization

Replication

- Replicate components/data/resources
- Caching – a special form of replication
- Increases availability, load balancing, better performance
- Drawbacks – consistency problems

Replication & Consistency

Architecture

Agenda

Four aspects

- Architectural Styles
- Centralized Architecture
- Decentralized Architecture
- Hybrid System Architecture

Architectural Style

A style is formulated in terms of

- (Replaceable) components with well-defined interfaces
- The way that components are connected to each other
- The data exchanged between components
- How these components and connectors are jointly configured into a system

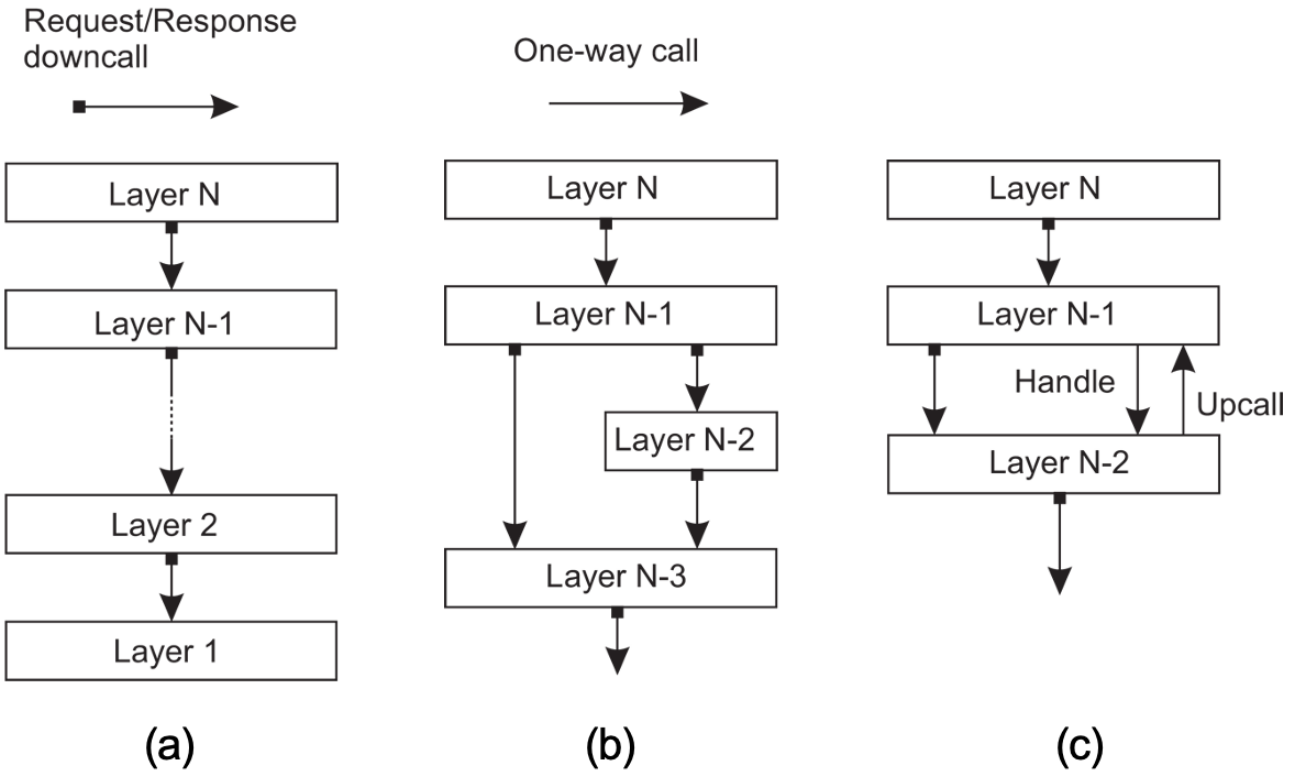
Connector

- A mechanism that mediates communication, coordination, or cooperation among components

Different styles – Layered, Service-Oriented, Publish-Subscribe

Layered Style

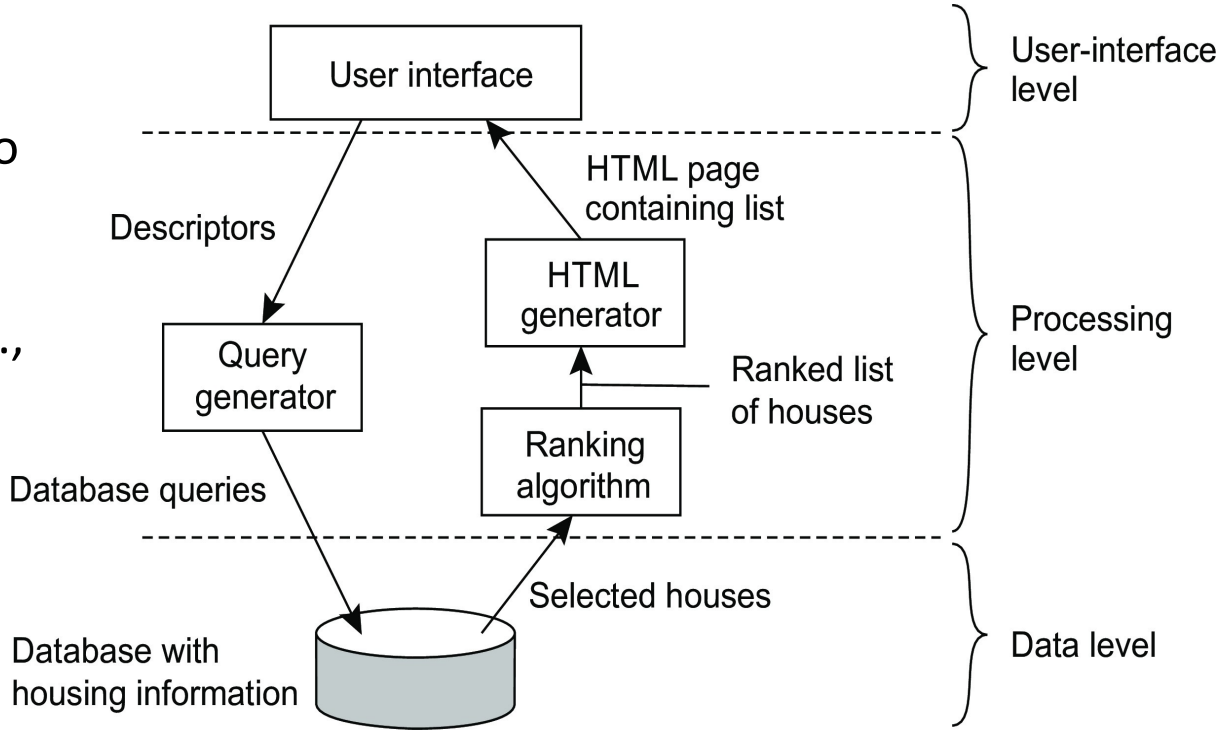
Components organized in a layered style



Application Layering

Three-layered view

- Application-interface layer contains units for interfacing to users or external applications
- Processing layer contains the functions of an application, i.e., without specific data
- Data later contains the data that a client wants to manipulate through the application components



Service-Oriented Style

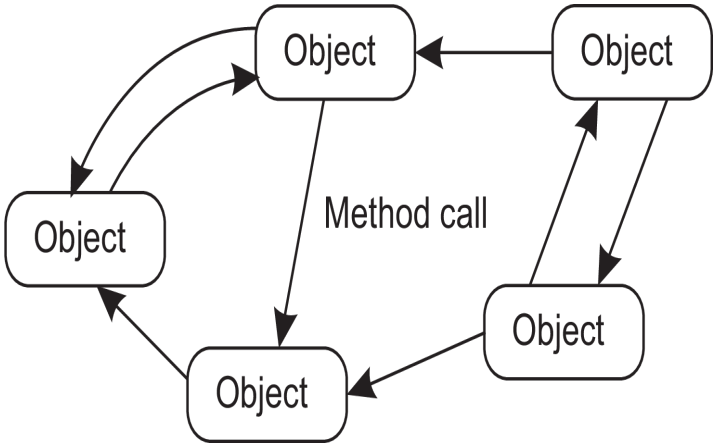
Multiple types – Object-based, Microservices, Resource-based

Object-based

- Objects represent components; connected to each other through procedure calls
- Objects may be placed on different nodes

Microservices

- Allow a large application to be separated into smaller independent parts
- Each part have its own realm of responsibility

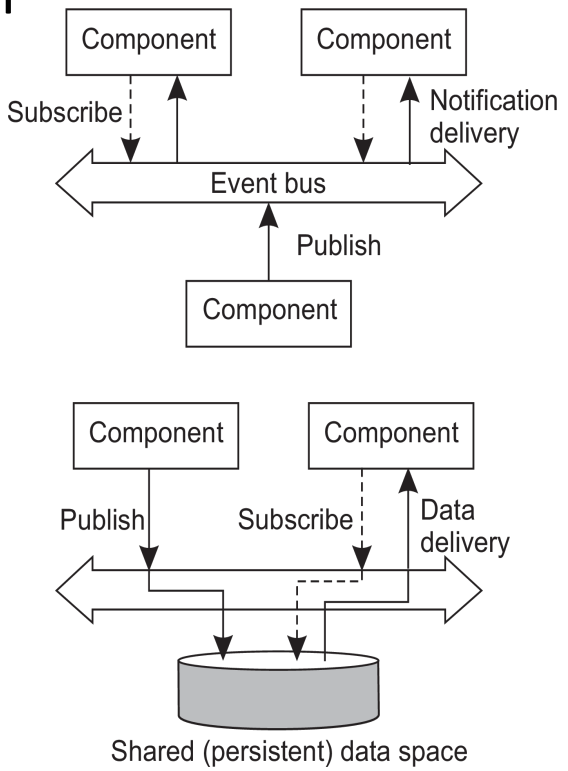


Publish-Subscribe Style

Strong separation between processing and coordination

View system as a collection of autonomously operating processes

Two types – Event-based, Shared data-space



System Architecture

Organize system based on where the software components are placed

Three main types

- Centralized – Client-server
- Decentralized – Peer-to-peer
- Hybrid -- Cloud, Edge

Centralized Architecture

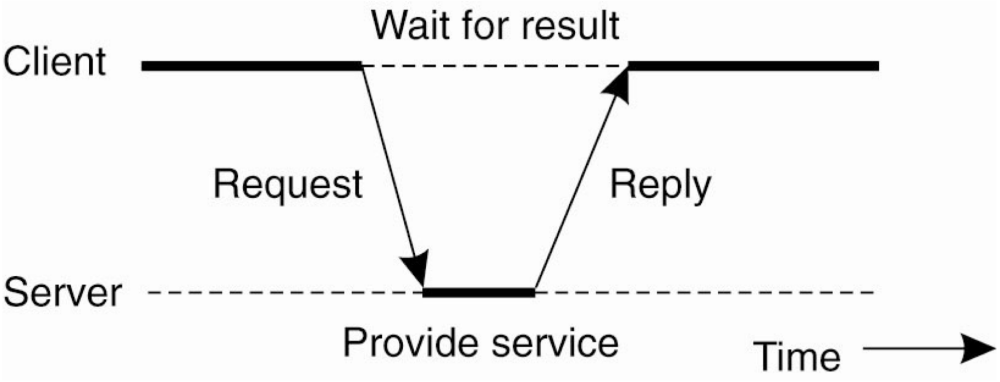
Basic Client-Server Model

Servers – Processes offering services

Clients – Processes using services

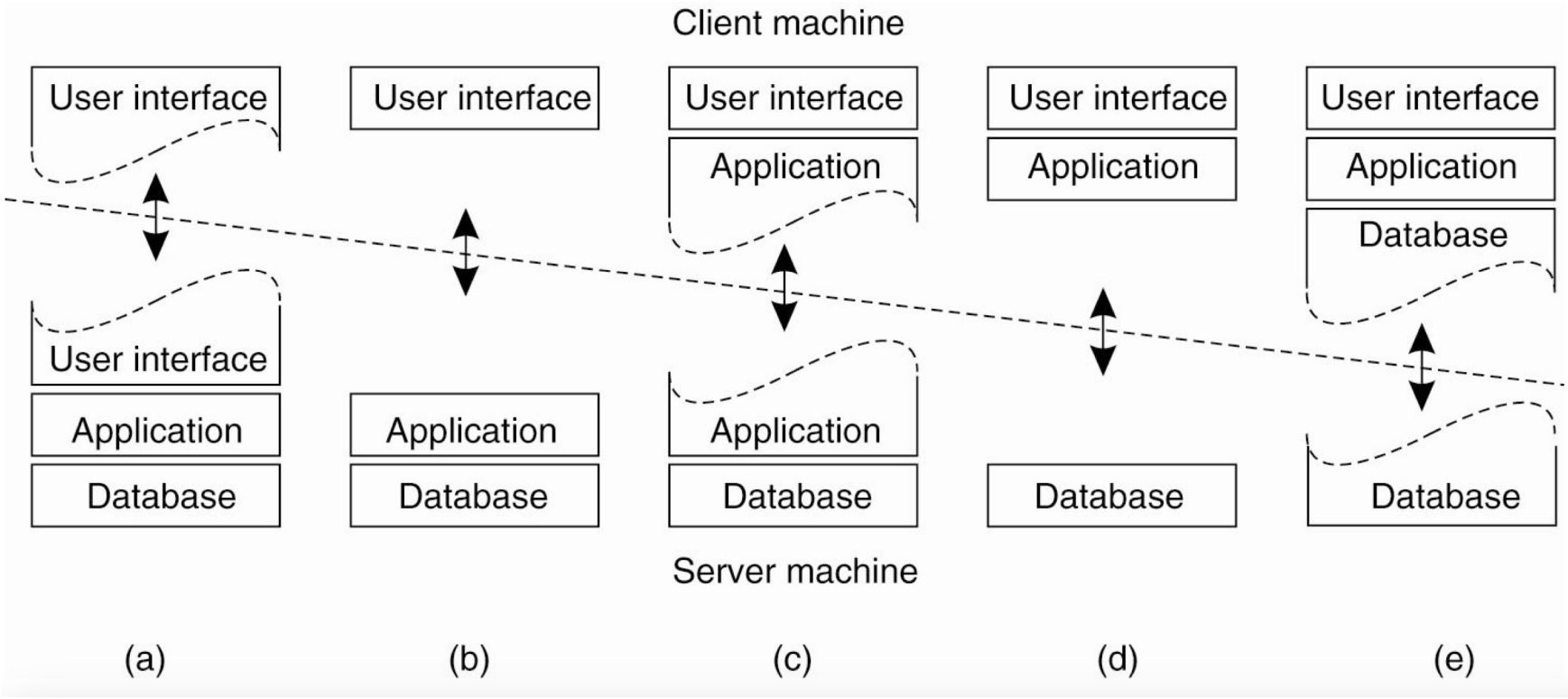
Clients and Servers can be on different machines

Clients follow request/reply model regarding using services



Multi-tiered Centralized Architectures

Spectrum of choices – Browser-based to phone-based to desktop apps

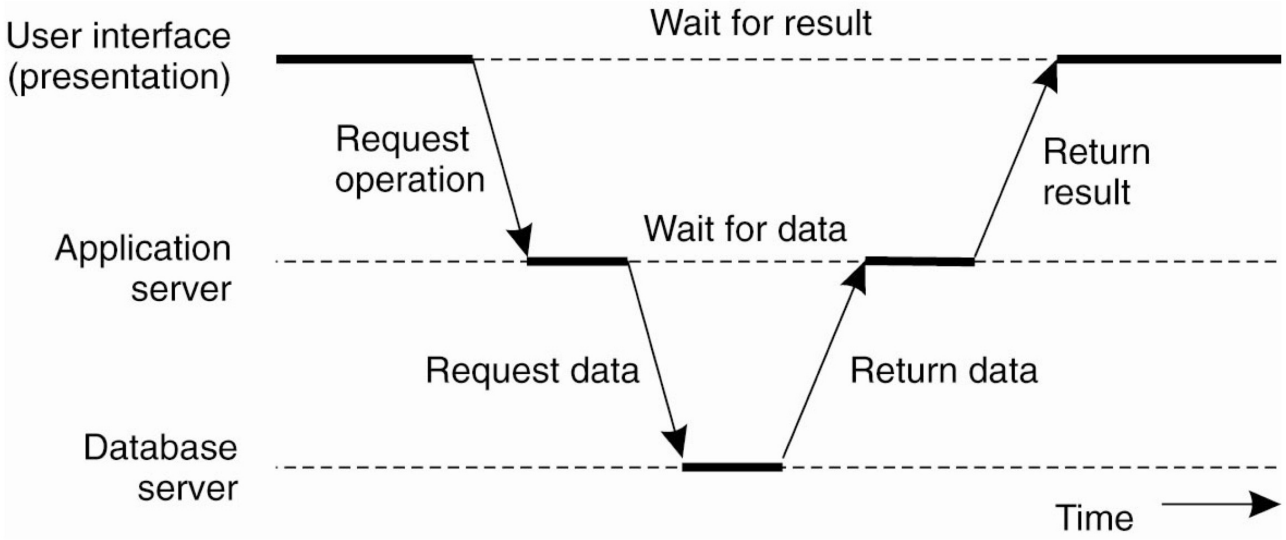


Three-tier Web Applications

Very common in most web-based applications

Server itself uses a “client-server” architecture

3 tiers – HTTP, J2EE and DB



Decentralized Architecture

Peer-to-peer systems

- Client and Server physically split up into logically equivalent parts
- Each part operates on its own share of the complete data set

Better for load-balancing

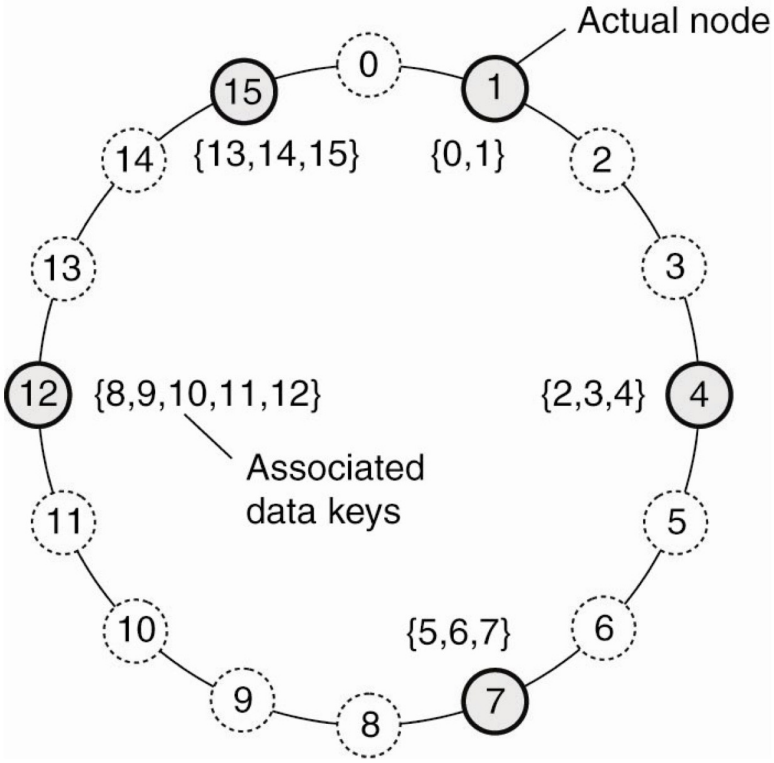
Two types depending on the topology

Structured Peer-to-Peer Systems

Adheres to specific deterministic topology – ring, tree, grid, etc.

Semantic-free index – each data item associated with a key; key used as an index

Example: Chord – Uses distributed Hash Table to locate data/objects; Data item with key k → smallest node with id >= k

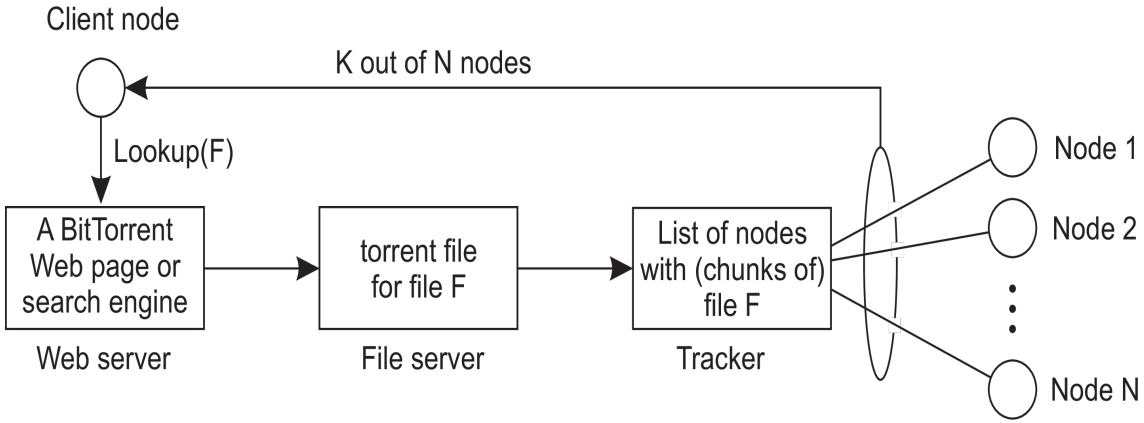


Unstructured Peer-to-Peer Systems

Adheres to randomly picking the neighbors → Random graph

Use flooding or random walks to search for specific data

Example: BitTorrent for file sharing



Hybrid Architectures

Mix of both centralized and decentralized architecture

Examples: Cloud Computing, Edge Computing, Blockchain/Distributed Ledgers

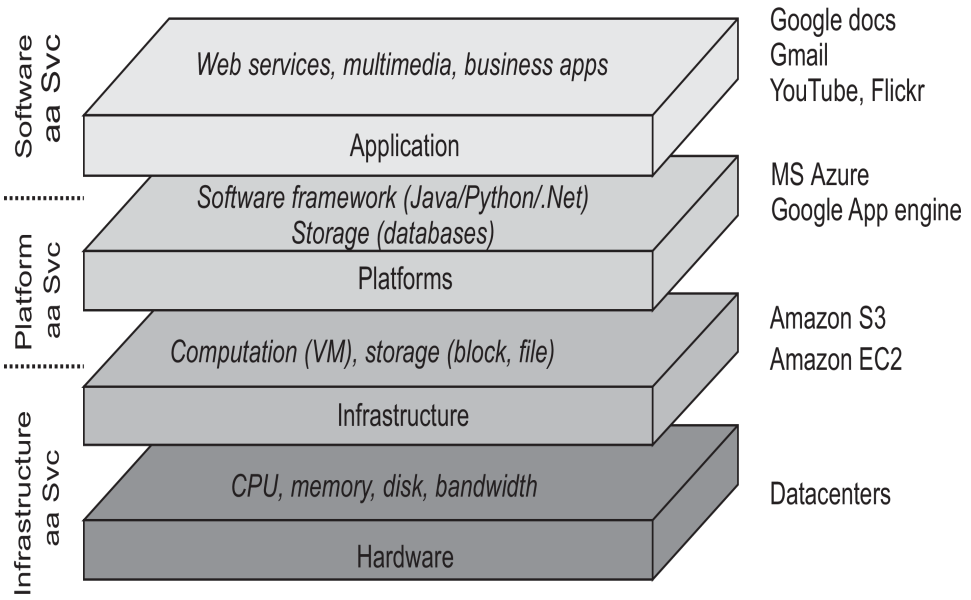
Cloud Computing

Layered approach

- Hardware – Processors, Routers, etc.
- Infrastructure – Virtualization techniques dealing with CPU, storage
- Platform – Provide higher-level abstractions for storage
- Application – Actual applications such as Google Docs, Office 365, YouTube, etc.

Pay-per-use model

Guaranteed Service-level Agreements (SLAs)



Edge Computing

Systems deployed on the Internet where servers are placed at the edge of the network: the boundary enterprise networks and the actual Internet

Why Edge

- Performance
- Reliability
- Security & Privacy

Edge Orchestration – Trickier than in the Cloud

- Resource allocation guarantees; Service Placement; Edge Selection