

# Lecture 2 - Distributed System Architectures

## Centralized (Client-Server)

- **Associated Architectural Style: Layered Architecture**
  - Matches the vertical/hierarchical organization of client-server systems.
  - Separates functions into distinct layers (e.g., presentation, business logic, data layers).
  - Example: Traditional web applications where clients request resources from servers.

## Decentralized (Peer-to-Peer)

- **Associated Architectural Style: Event-Based Architecture**
  - Supports horizontal communication and symmetric functionality in P2P systems.
  - Relies on events/messages for coordination between peers without centralized control.
  - Example: File-sharing networks where nodes communicate directly.

## Hybrid (C/S + P2P)

- **Associated Architectural Styles:**
  1. **Component-Based Architecture**
    - Enables modular integration of client-server and P2P components.
    - Example: Edge-server systems that combine centralized cloud services with decentralized edge devices.
  2. **Data-Centered Architecture**
    - Facilitates collaboration through shared data spaces (e.g., databases, repositories).
    - Example: Collaborative systems where users interact with shared resources.

# Definitions

- **Software Architectures** – describe the organization and interaction of software components; focuses on logical organization of software (component interaction, etc.)

Examples include layered architecture, microservices architecture, and MVC (Model-View-Controller).

- **System Architectures** - describe the placement of software components on physical machines

Examples include cloud-based architecture, client-server architecture, and distributed systems.

Feature	Software Architecture	System Architecture
Focus	Logical structure and interaction of software components	Physical deployment and placement of software components on hardware
Concerned With	Component interaction, data flow, design patterns, modularity	Hardware, networking, cloud/on-premise deployment, scalability
Key Elements	Layers, services, modules, APIs, frameworks	Servers, databases, networks, data centers, cloud infrastructure
Examples	Microservices, MVC (Model-View-Controller), Event-Driven Architecture	Client-Server, Distributed Systems, Cloud Computing, Edge Computing
Scope	Internal structure and communication of software	Infrastructure that supports software execution
Decisions Impact	Maintainability, scalability, code reusability	Performance, availability, reliability, security

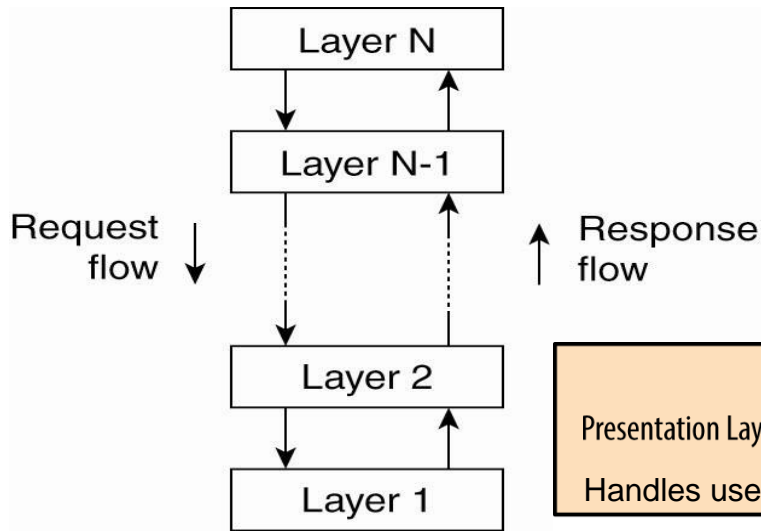
# Architectural Styles

- An **architectural style** describes a particular way to configure a collection of components and connectors.
  - **Component** - a module with well-defined interfaces; reusable, replaceable
  - **Connector** – communication link between modules
- Architectures suitable for distributed systems:
  - Layered architectures\*
  - Component-based architectures\*
  - Data-centered architectures
  - Event-based architectures

# Layered Architecture

- Each layer/tier is allocated a specific responsibility of the system
- Each layer can only interact with the neighboring layers (important for integrity and security)
- Each layer may contain layers of its own (e.g. Presentation layer could contain two layers: client layer and client presentation layer)

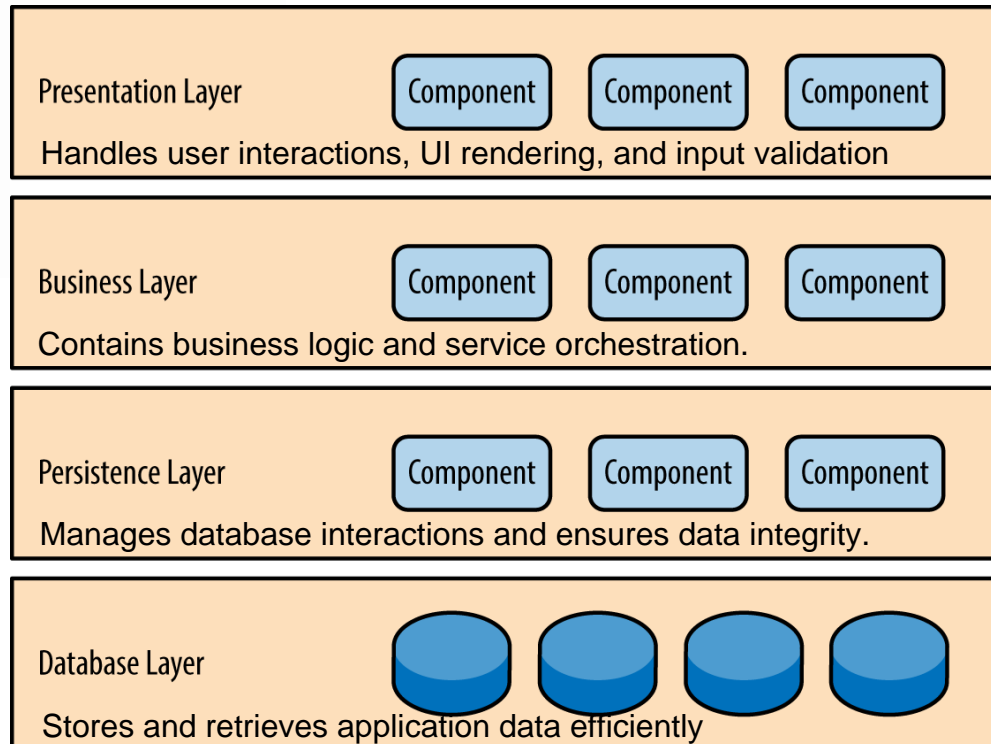
# Layered Architecture



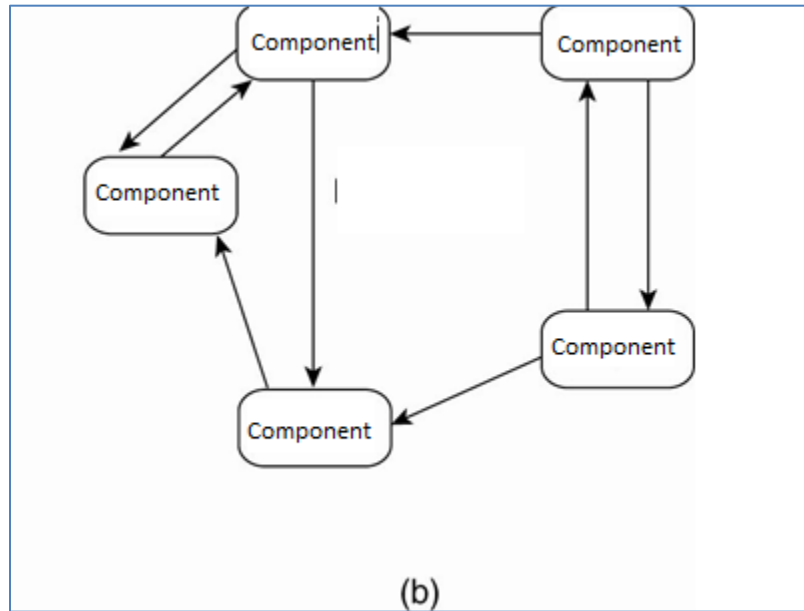
(a)

## Example: E-Commerce System

Layer	Example Responsibilities
Presentation Layer	Displays product catalog, shopping cart, order forms
Application Layer	Processes orders, applies discounts, manages user sessions
Data Access Layer	Fetches product details, updates order statuses in the database
Database Layer	Stores user info, product data, and purchase history



# Component based architecture

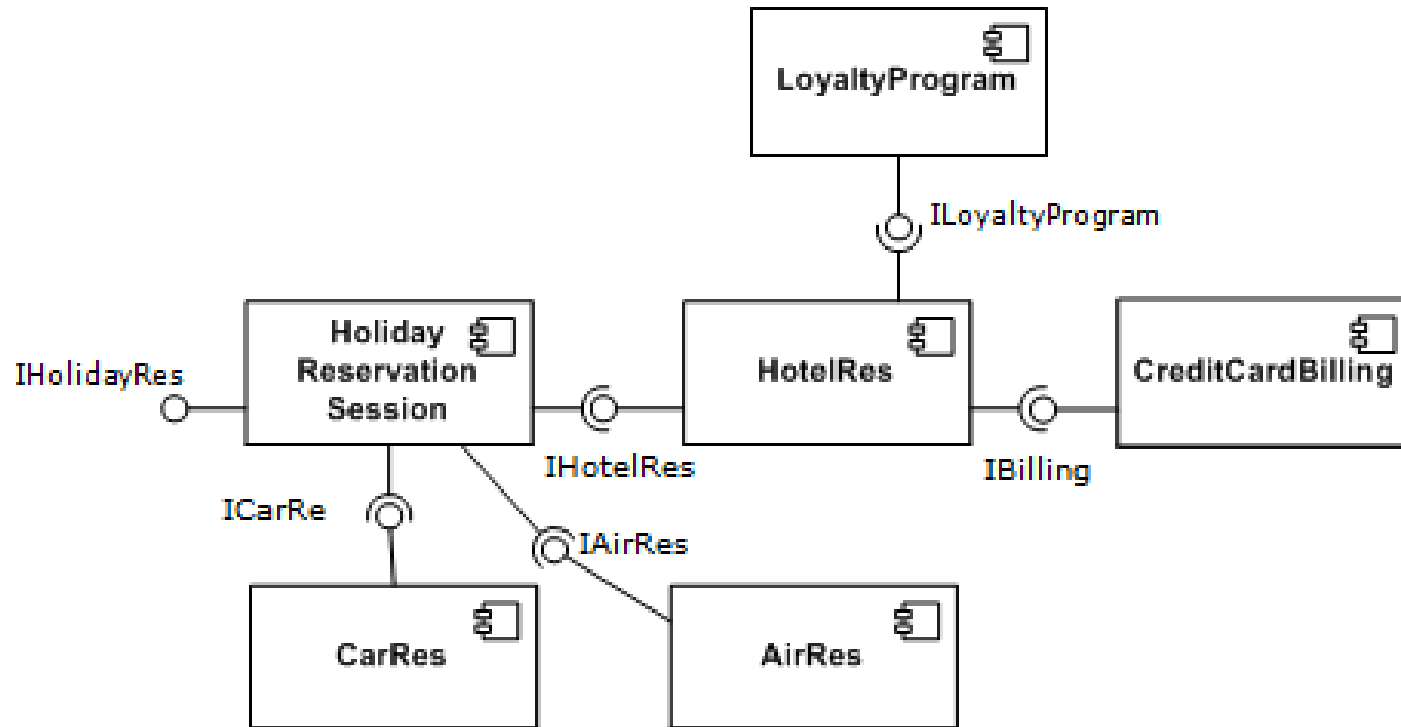


A Component-Based Architecture structures a system into independent, reusable components that communicate via connectors.

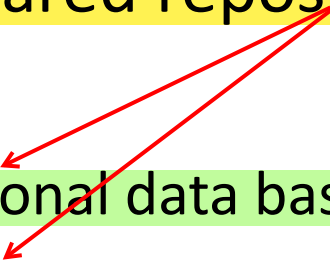
It is less structured than other architectures, offering flexibility and modularity.

- Consists of components and connectors
- Component based is less structured

# Component based architecture

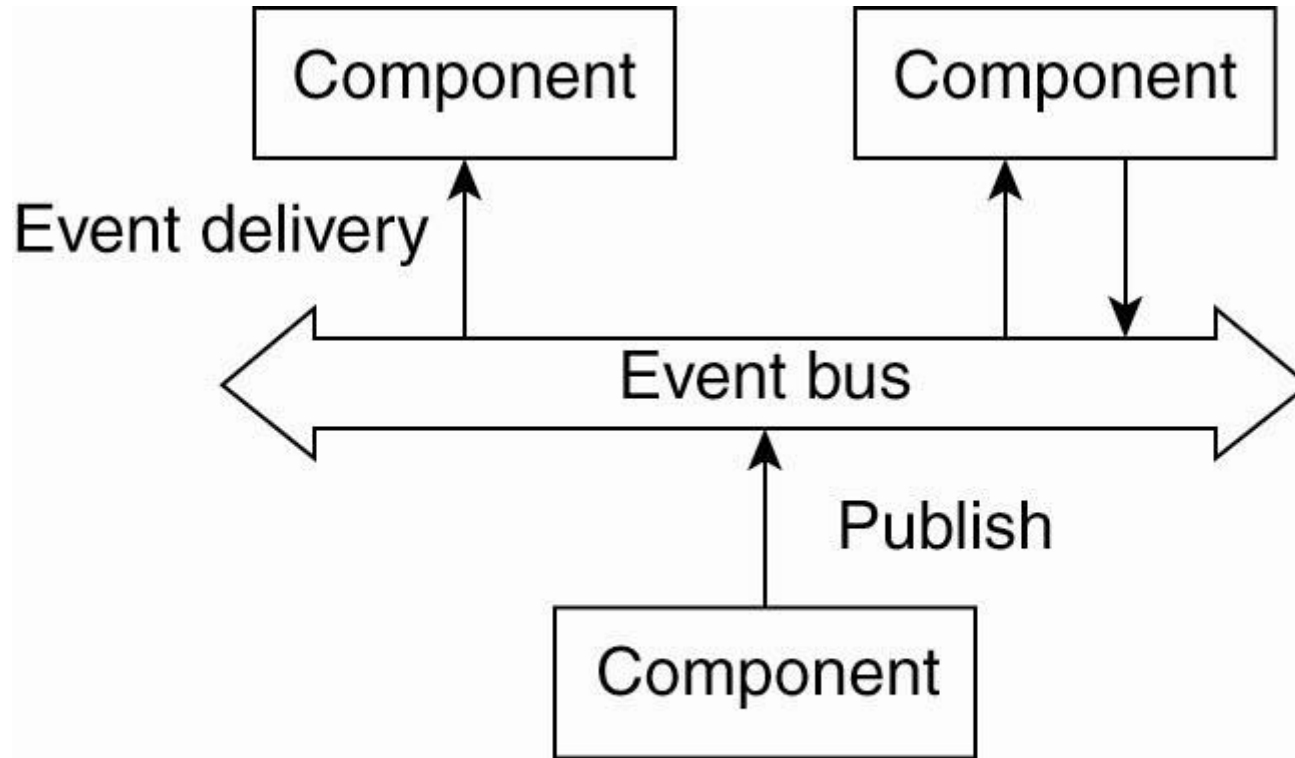


# Data-Centered Architectures

- Main purpose: data access and update
  - Processes interact by reading and modifying data in some shared repository (active or passive)
    - Traditional data base (passive): responds to requests
    - Blackboard system (active): clients solve problems collaboratively; system updates clients when information changes.
- 



# Event based Architectures



(a)

# Event based Architectures

High scalability

- Communication via event propagation, in dist. systems seen often in Publish/ Subscribe; e.g., register interest in market info; get email updates
- Decouples sender & receiver; asynchronous communication
- Event-based architecture supports several communication styles:
  - Publish-subscribe** Components publish events that multiple subscribers listen to.  
eg . Stock market updates, push notifications
  - Broadcast** An event is sent to all components, regardless of interest.  
eg - Emergency alerts, live streaming
  - Point-to-point** One-to-one event communication between components.  
eg - Email notifications, direct messaging

# Shared Data-Space Architecture

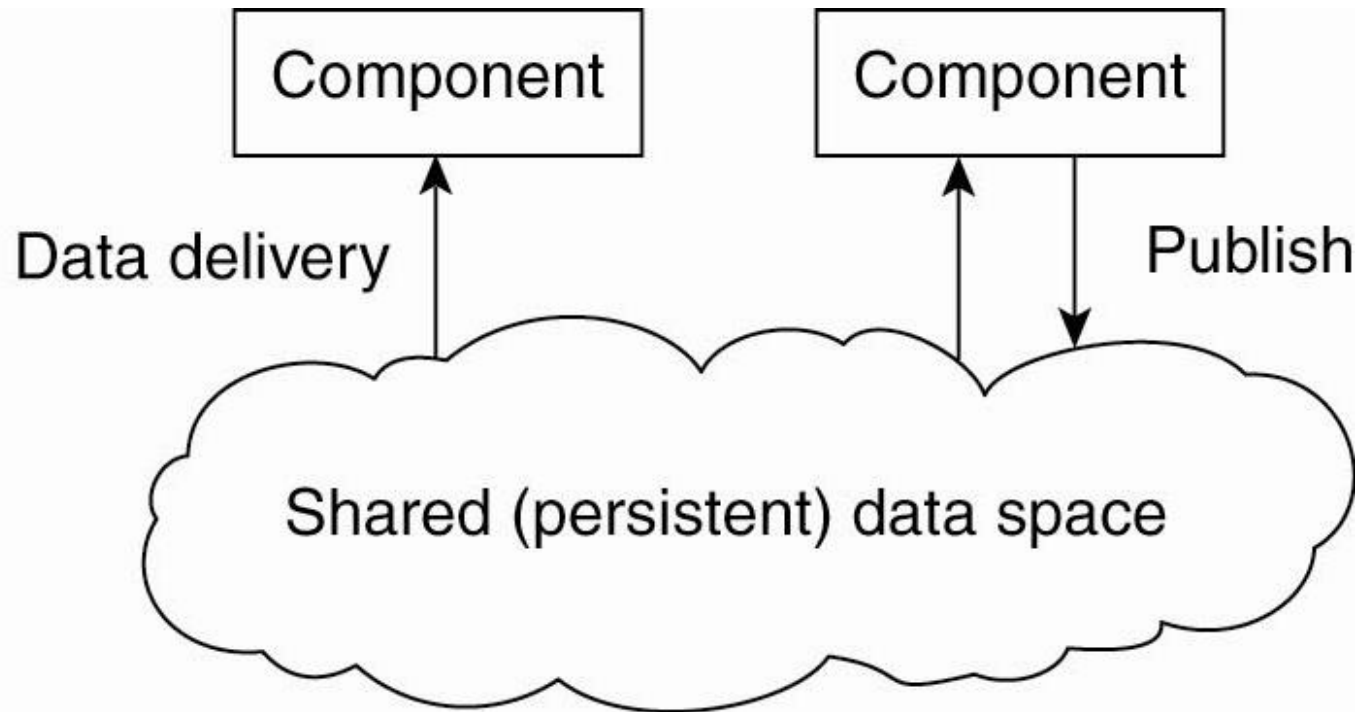
- Multiple Architectures can be combined in the same system architecture

E.g. A component in the object based architecture may have a layered architecture

A Shared Data-Space Architecture is a hybrid approach that combines both Data-Centric and Event-Based Architectures to enable asynchronous communication through a shared data repository.

- Shared Data-Space Architecture combines the Data-centric architecture and Event based architecture

# Shared Data-Space Architecture



(b)

- E.g., shared distributed file systems or Web-based distributed systems
- Processes communicate asynchronously

# Which Software Architectural style?

\* final exam question

- An online forum to share travel information among users

Data-Centered Architectures - he forum relies on a centralized database to store posts, comments, and user data.

- A remote monitoring system that monitors the health of an elderly person.

Event based - The system must continuously monitor health data (heart rate, blood pressure, etc.).

- A music file sharing system among a group of users.

data centered

- A mobile taxi app

Event-Based:

Drivers and riders receive real-time ride requests and location updates.

Architectural Style	Key Characteristics	Examples
Layered Architecture	System is divided into multiple layers (UI, Business Logic, Data), each layer interacts only with adjacent layers, ensures modularity, maintainability, and security.	Web applications, Banking systems, E-commerce platforms.
Component-Based Architecture	System is built using independent, reusable components, each component has a well-defined interface, allows scalability and flexibility.	Hospital Management System, E-commerce systems, ERP software.
Data-Centered Architecture	Centralized data storage (database, repository), processes interact by reading and writing to shared data, ensures consistency and reliability.	Online forums, Social Media, University Management System.
Event-Based Architecture	Communication via event propagation, supports asynchronous messaging, loosely coupled components.	Stock Market Systems, IoT Smart Home Systems, Notification Systems.
Shared Data-Space Architecture	Combination of Data-Centered and Event-Based architectures, processes communicate asynchronously through a shared repository.	Peer-to-peer file sharing, Cloud-based distributed systems.

# Distribution Transparency

see lecture 01

- An important characteristic of software architectures in distributed systems is that they are designed to support distribution transparency.
- Transparency involves trade-offs
- Different distributed applications require different solutions/architectures
  - There is no “silver bullet” – no one-size-fits-all system.  
(Compare NOW, Seti@home, Condor)

# System Architectures

# System Architectures for Distributed Systems

- **Centralized:** traditional client-server structure C/S
  - Vertical (or hierarchical) organization of communication and control paths (as in layered software architectures)
  - Logical separation of functions into client (requesting process) and server (responder)
- **Decentralized:** peer-to-peer P2P
  - Horizontal rather than hierarchical comm. and control
  - Communication paths are less structured; symmetric functionality
- **Hybrid:** combine elements of C/S and P2P C/S + P2P
  - Edge-server systems
  - Collaborative distributed systems.
- Classification of a system as centralized or decentralized refers to communication and control organization, primarily.



# Traditional Client-Server

Traditional Client-Server Communication:

- Processes are divided into two groups (clients and servers).
- Synchronous communication: request-reply protocol

Communication Protocols:

- In LANs, often implemented with a connectionless protocol (unreliable) faster
- In WANs, communication is typically connection-oriented TCP/IP (reliable)
  - High likelihood of communication failures

# C/S Architectures

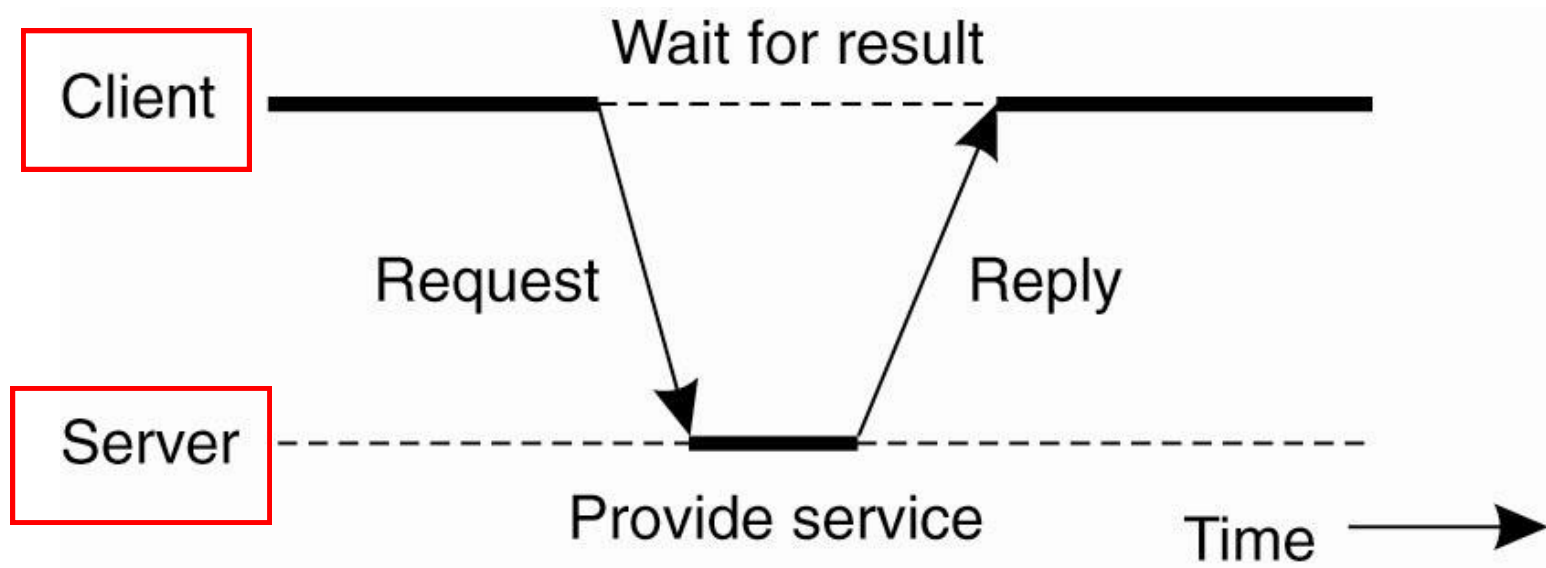


Figure 2-3. General interaction between a client and a server.

# Two-tiered C/S Architectures

small functionality run in client side workload has server

- Server provides processing and data management; client provides simple graphical display (**thin-client**)
  - Perceived performance loss at client
  - Easier to manage, more reliable, client machines don't need to be so large and powerful

large functionality run on client side reduce workload has server

- At the other extreme, all application processing and some data resides at the client (**fat-client** approach)
  - Pro: reduces workload at server; more scalable
  - Con: harder to manage by system admin, less secure

# Three-tiered Architectures

- In some applications servers may also need to be clients, leading to a three-level architecture
  - Distributed transaction processing
  - Web servers that interact with database servers
- Distribute functionality across three levels of machines instead of two.

# Multitiered Architectures

## (3 Tier Architecture)

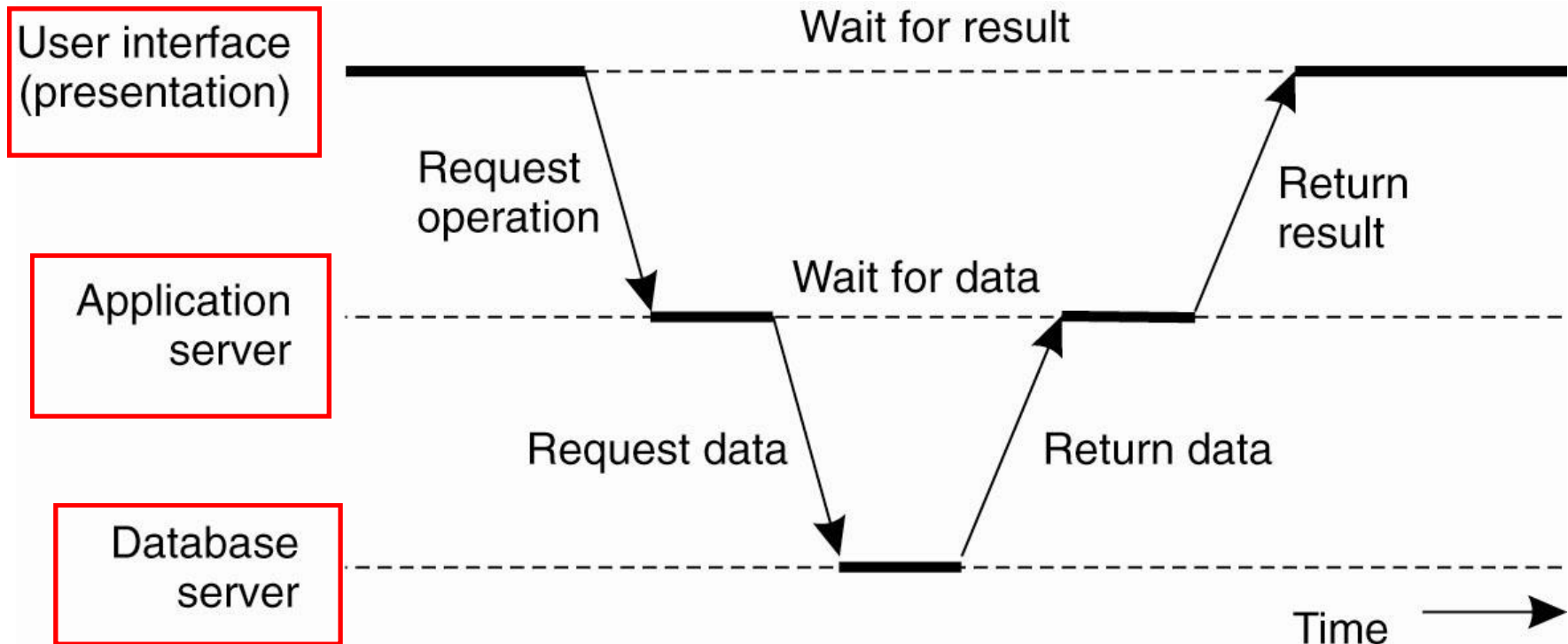


Figure 2-6. An example of a server acting as client.

# Multitiered Architectures

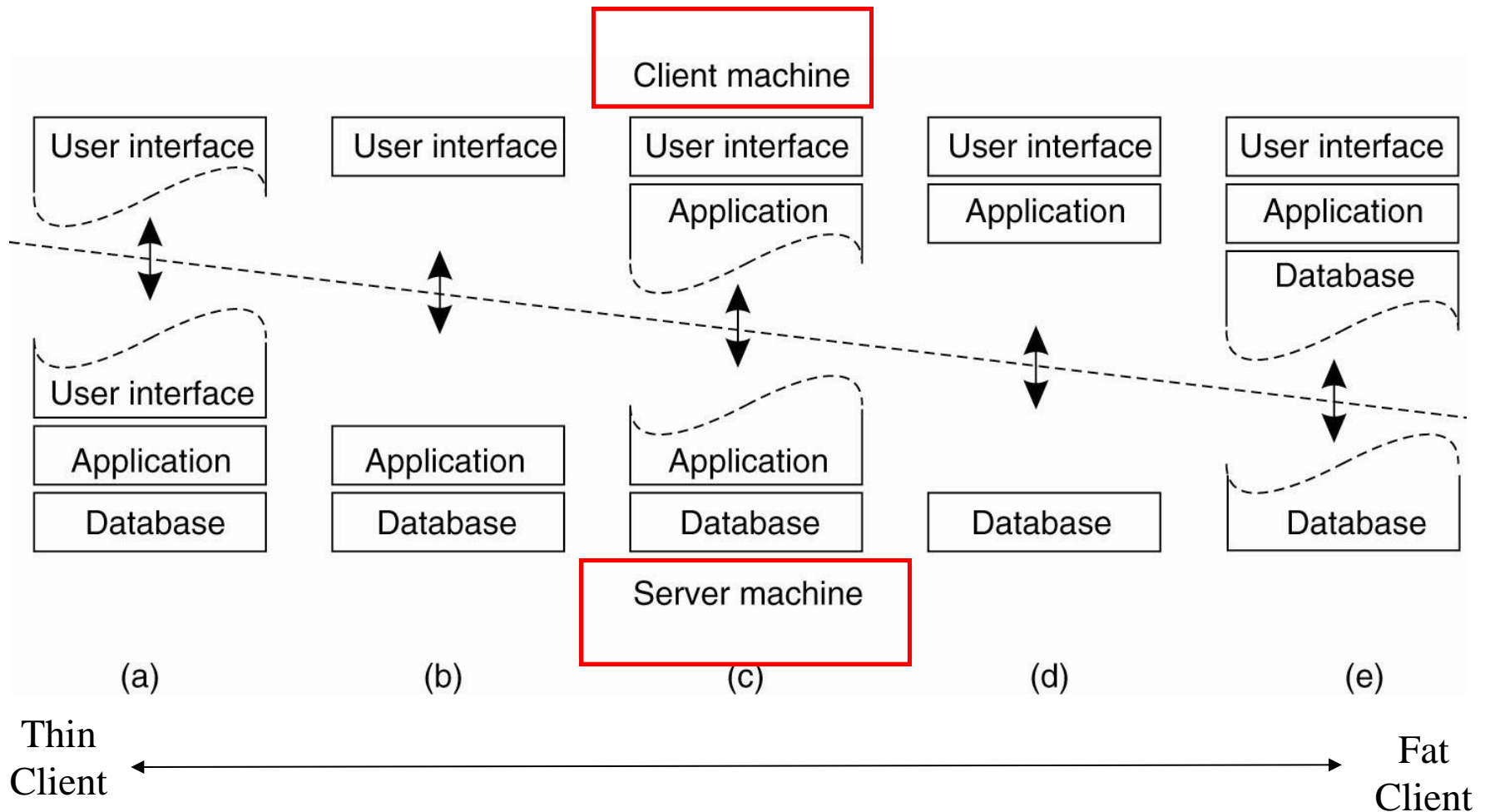


Figure 2-5. Alternative client-server organizations (a)–(e).

# Layered (software) Architecture for Client-Server Systems

- **User-interface level:** GUI's (usually) for interacting with end users
- **Processing level:** data processing applications – the core functionality
- **Data level:** interacts with data base or file system
  - Data usually is persistent; exists even if no client is accessing it
  - File or database system

# Examples

- Web search engine
  - Interface: type in a keyword string
  - Processing level: processes to generate DB queries, rank replies, format response
  - Data level: database of web pages
- Stock broker's decision support system
  - Interface: likely more complex than simple search
  - Processing: programs to analyze data; rely on statistics, AI perhaps, may require large simulations
  - Data level: DB of financial information
- Cloud based “office suites”
  - Interface: access to various documents, data,
  - Processing: word processing, database queries, spreadsheets,...
  - Data : file systems and/or databases



# Application Layering

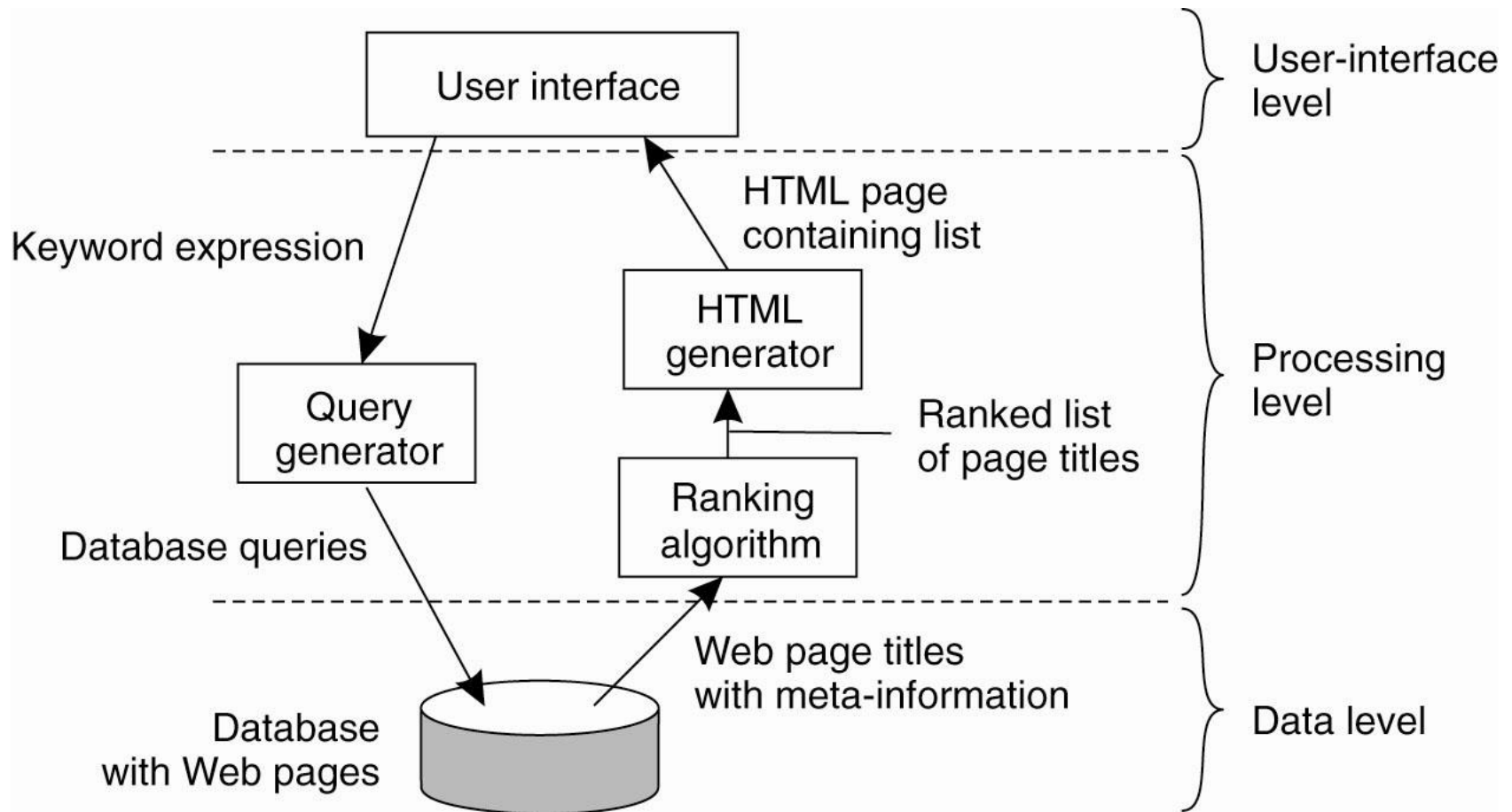
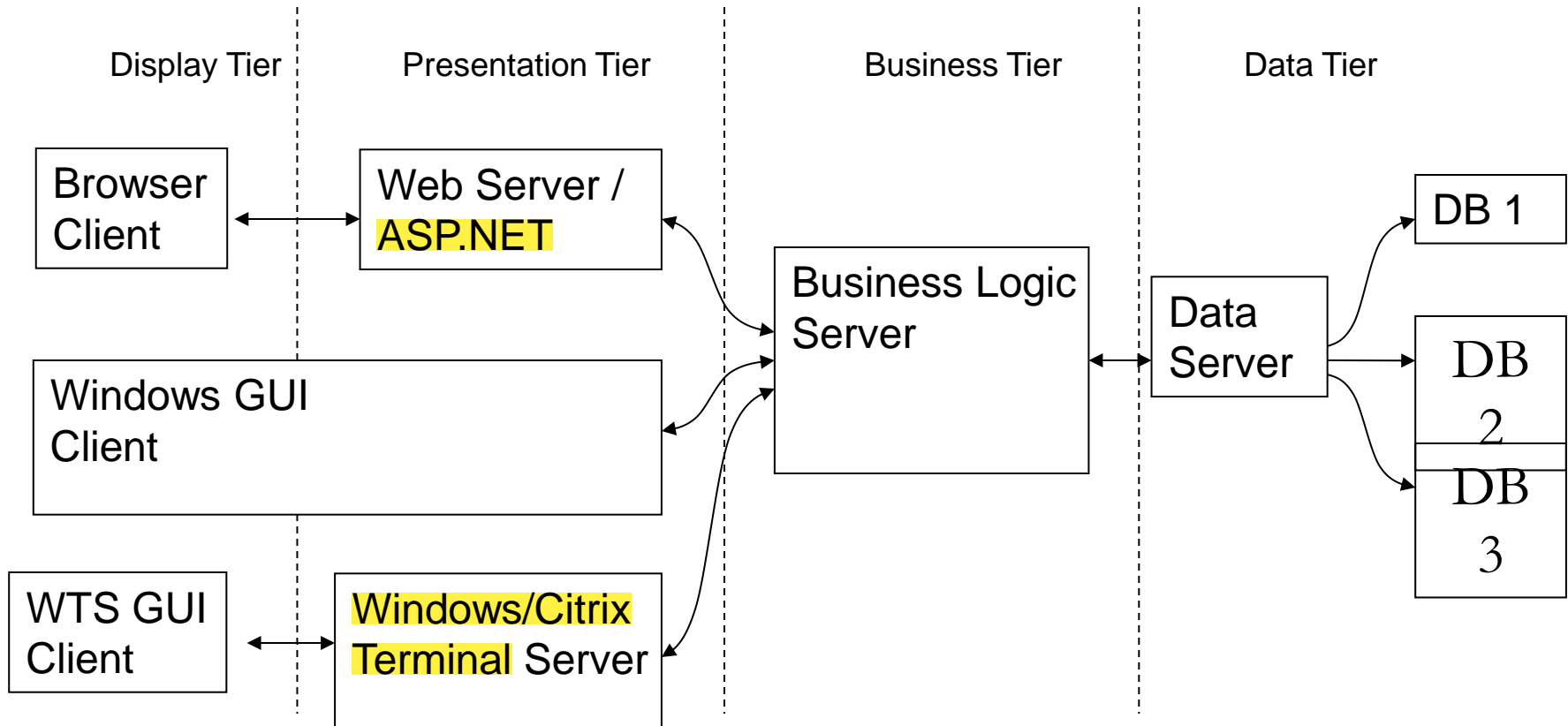


Figure 2-4. The simplified organization of an Internet search engine into three different layers.

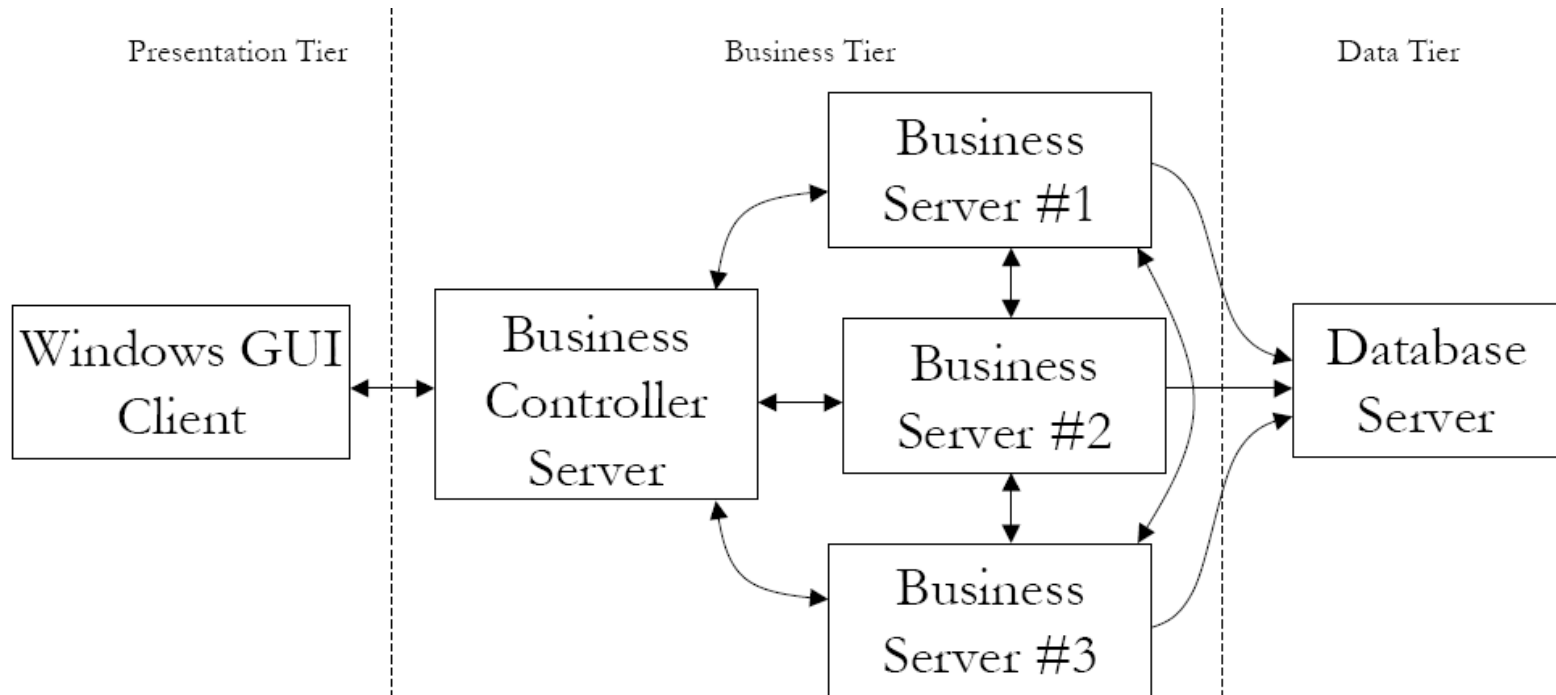
# Flexible/Scalable Architecture



# Distributing the Business Tier

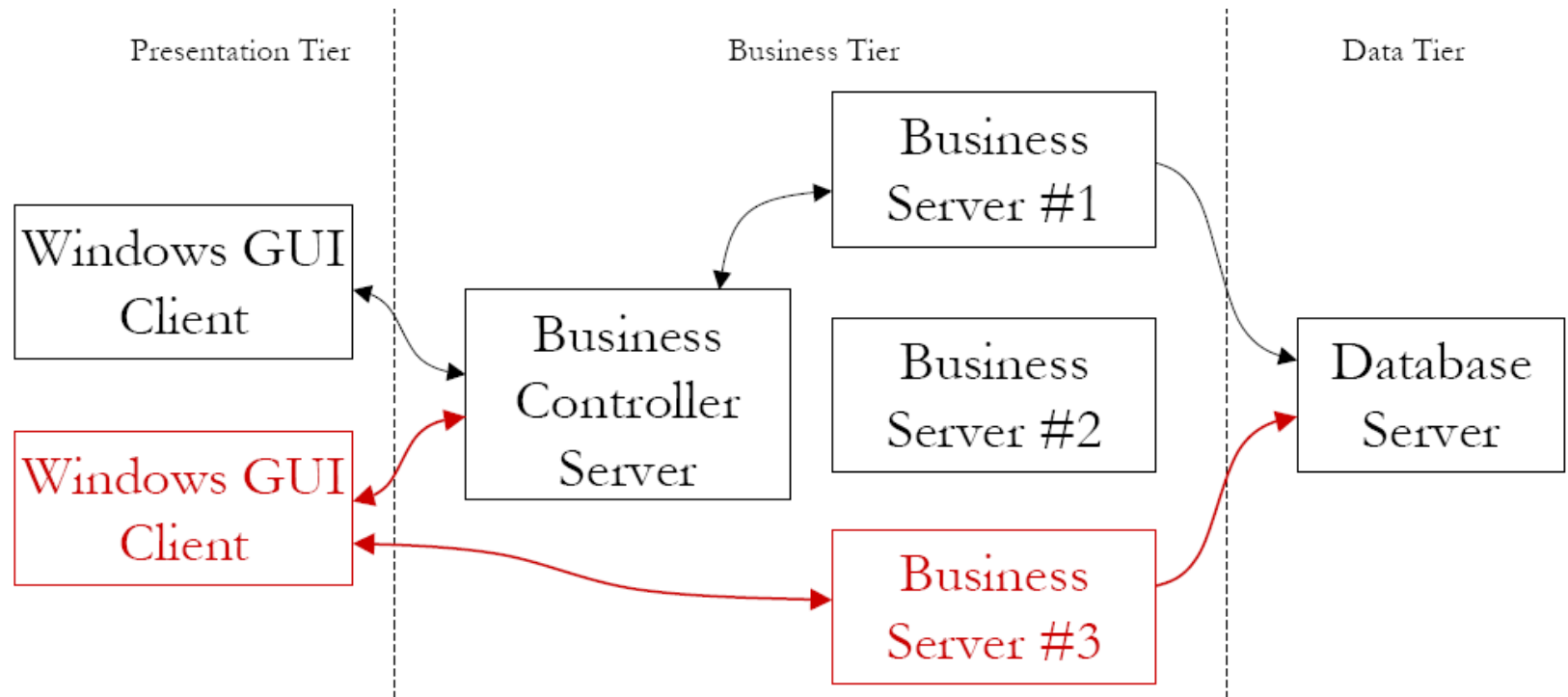
- Business tier could be distributed for various reasons:
  - **Parallel computing** - split one job among many servers
  - **Load balancing** - have a controller component redirect presentation clients to a least-utilized business tier server
  - **Fault tolerance** - robustness to system faults or data faults

# Parallel Computing Architecture



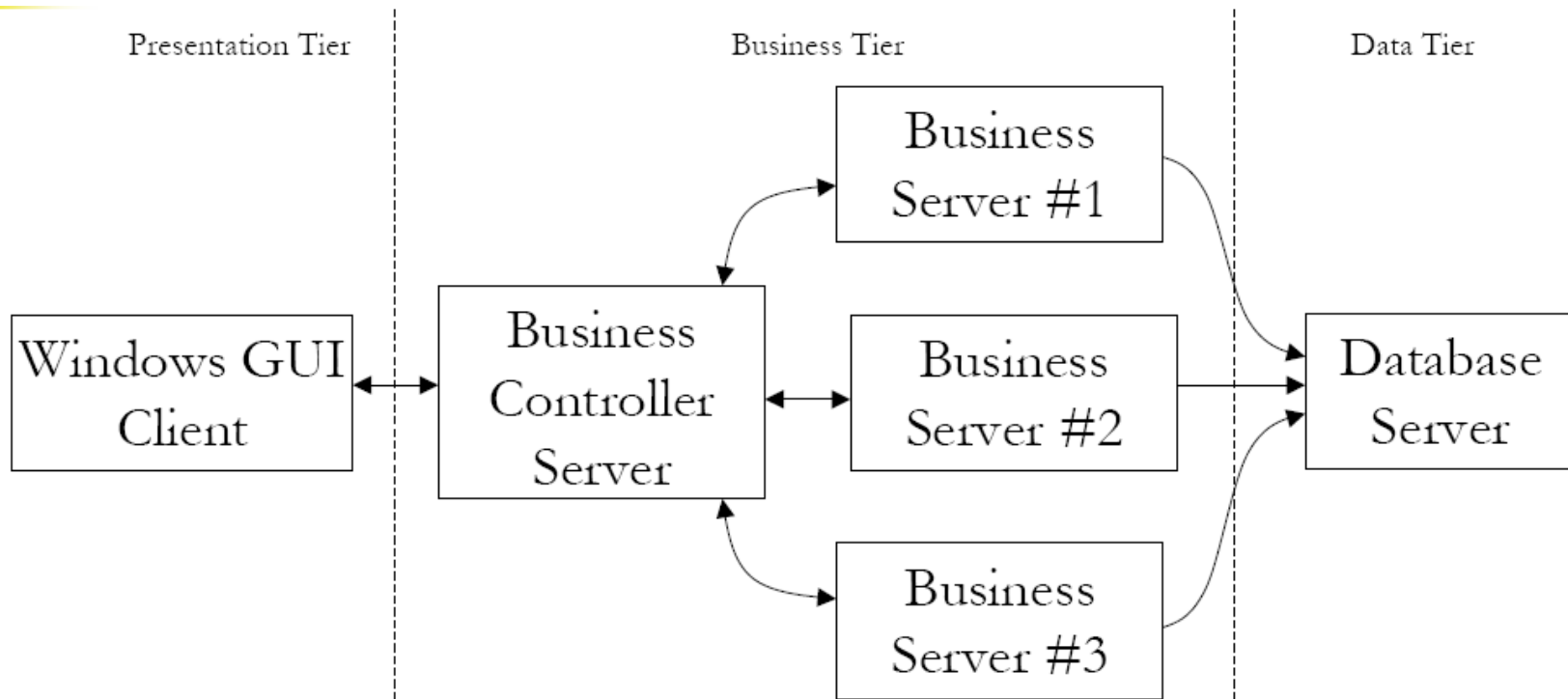
- Controller passes job to server farm, returns response
  - Business servers collaborate via data sharing

# Load Balancing Architecture



- Controller passes **job to one server, returns response**
- **Alternative:** Controller tells client which server to use

# Fault Tolerant Architecture



- Data faults: Controller asks all servers to do the same job
  - Then compares results to detect faults
- System faults: Controller uses any server that is up

# Whether to tier or not?

## ■ Advantages

- Can reuse code (high coherence, low coupling) similler function together      similler function independent
- Scalable
- Integrity
- Fault tolerance

## ■ Disadvantages

- Increases communication overhead
- Errors/Losses in message transmission
- Can pose security risks (e.g. packet sniffing)
- Increased Complexity

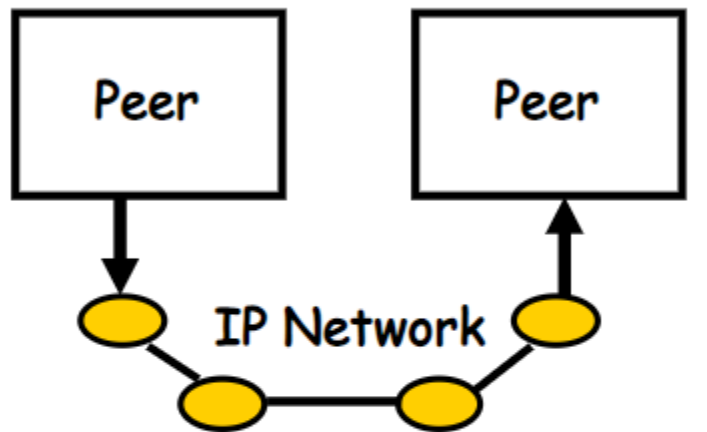
# Decentralized Architecture **Peer-to-Peer**

A peer-to-peer (P2P) system operates without a central server, distributing processing and data across multiple nodes.

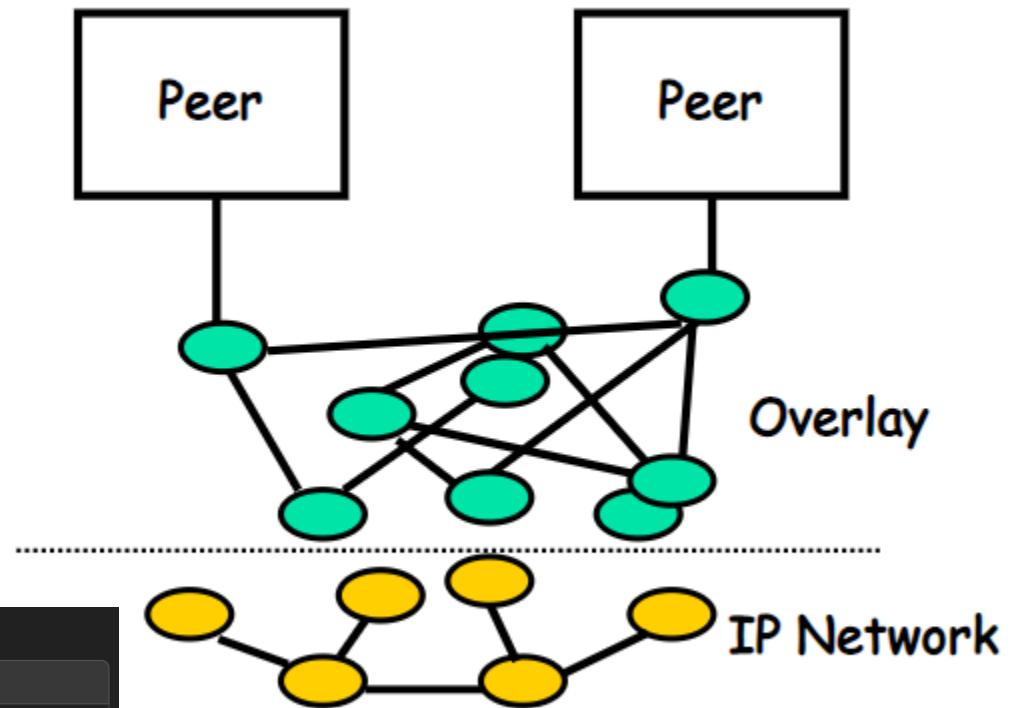
- Nodes act as both client and server; interaction is symmetric (e.g. Pastry, Chord)
- Each node acts as a server for part of the total system data
- **Overlay networks** connect nodes in the P2P system
  - Nodes in the overlay use their own addressing system for storing and retrieving data in the system
  - Nodes can route requests to locations that may not be known by the requester.



# P2P Overlay networks



Traditional Communication



Tunneling Communication

## P2P vs. Client-Server

Feature	Client-Server	Peer-to-Peer
Centralization	Centralized	Decentralized
Communication	Structured	Unstructured
Failure Handling	Single point of failure	More fault-tolerant
Performance Bottlenecks	Possible	Less likely
Security	Easier to secure	Harder to control

# P2P v Client/Server

- P2P computing allows end users to communicate without a dedicated server.
- P2P overcomes the issue of Single point of failure in C/S
- Communication is still usually synchronous (blocking)
- There is less likelihood of performance bottlenecks since communication is more distributed.
  - Data distribution leads to workload distribution.
- Resource discovery is more difficult than in centralized client-server computing & look-up/retrieval is slower
- P2P can have freeloading/free-riding issue (particularly in resource sharing)
- P2P can be more fault tolerant, more resistant to denial of service (DoS) attacks because network content is distributed.
  - Individual hosts may be unreliable, but overall, the system should maintain a consistent level of service

# P2P architectures

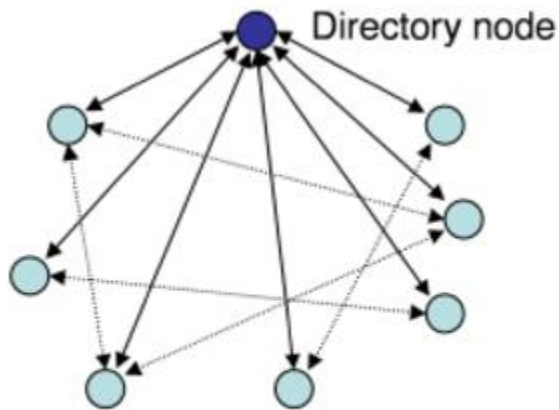
Types of P2P Architectures,

- **Structured** – E.g. **Chord**, **Pastry**, uses **Distributed Hash Tables (DHT)**, has **Circular structures**
- **Unstructured** – E.g. **BitTorrent**, **Napster**, has **unstructured complex structures**, **both pure P2P and Hybrid are unstructured**

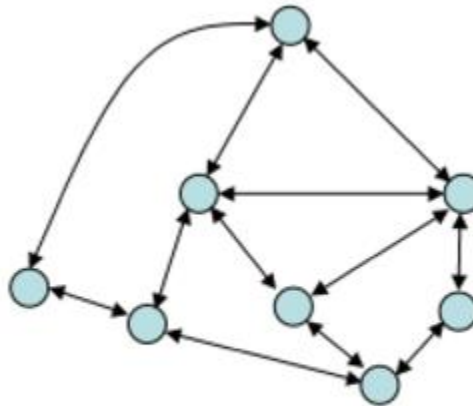
Nodes organize into a structured topology.  
Efficient lookup and routing.

Nodes communicate freely.  
Less efficient but more adaptable.

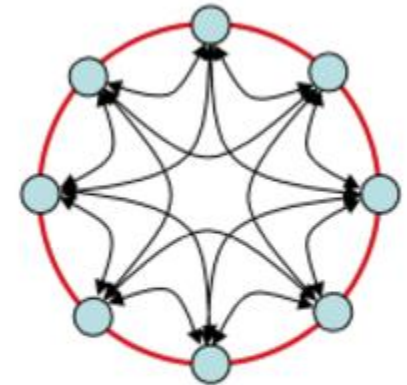
# Structured vs Unstructured P2P



Hybrid (Napster)



Unstructured overlay



Structured overlay

## Challenges in P2P Systems

- Resource Discovery – No centralized index.
- Freeloading – Some users consume but do not contribute resources.
- Slower Lookups – Unlike client-server, requests must traverse multiple nodes.

## Advantages of P2P

- Fault Tolerance – System remains functional even if nodes fail.
- DoS Resistance – Attacks on individual nodes don't take down the network.
- Better Load Distribution – Data spread across multiple nodes reduces bottlenecks.

# Hybrid Architectures

- Combine client-server and P2P architectures
  - Edge-server systems; e.g. ISPs, which act as servers to their clients, but cooperate with other edge servers to host shared content Clients interact with local servers, which cooperate with other edge servers.
  - Collaborative distributed systems; e.g., BitTorrent, which supports parallel downloading and uploading of chunks of a file. First, interact with C/S system, then operate in decentralized manner.

# Superpeers

- Maintain indexes to some or all nodes in the system
- Supports resource discovery
- Act as servers to regular peer nodes, peers to other super-peers
- Improve scalability by controlling floods
- Can also monitor state of network
- Example: Napster
- Edge-Server computing (e.g. ISP)

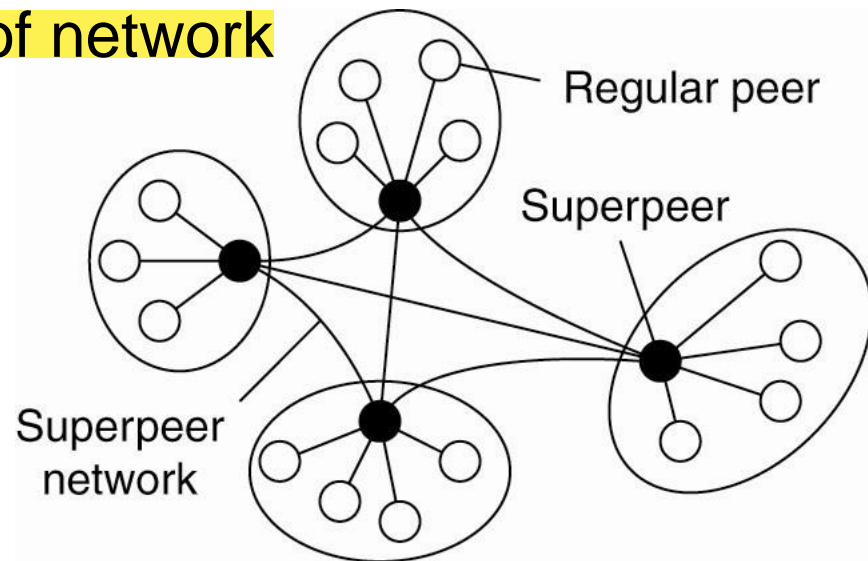


Figure 2-12.

# Hybrid (System Architecture) example

## - BitTorrent

<https://www.youtube.com/watch?v=6PWUCFmOQwQ>

# Distributed Hash Tables

- A fully decentralized routing mechanism (unlike TCP/IP routing)

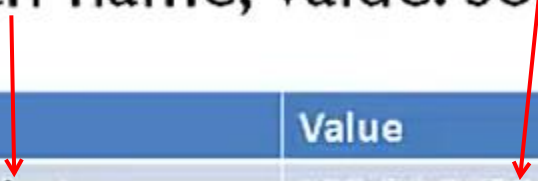
decentralized system for storing and retrieving key-value pairs across multiple nodes in a network. Unlike traditional hash tables that reside in a single system, DHTs distribute data among multiple peers in a structured way, allowing for efficient lookups and scalability.



# Distributed Hash Table (DHT)

Simple database with (key, value) pairs:

- key: human name; value: social security #

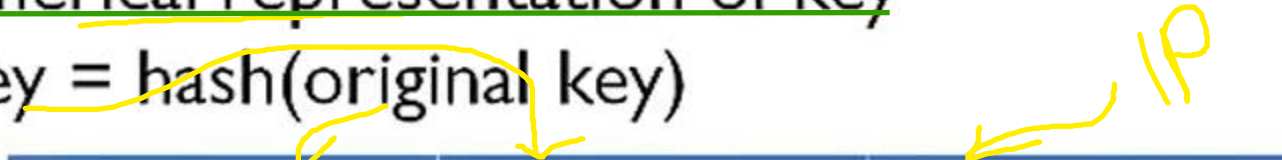


Key	Value
John Washington	132-54-3570
Diana Louise Jones	761-55-3791
Xiaoming Liu	385-41-0902
Rakesh Gopal	441-89-1956
Linda Cohen	217-66-5609
.....	.....
Lisa Kobayashi	177-23-0199

- key: movie title; value: IP address

# Hash Table

- More convenient to store and search on numerical representation of key
- $\text{key} = \text{hash}(\text{original key})$



Handwritten yellow arrows point from the text 'key = hash(original key)' to the 'Original Key' and 'Key' columns of the table. A handwritten 'IP' is written to the right of the table, with an arrow pointing to the 'Value' column.

Original Key	Key	Value
John Washington	8962458	132-54-3570
Diana Louise Jones	7800356	761-55-3791
Xiaoming Liu	1567109	385-41-0902
Rakesh Gopal	2360012	441-89-1956
Linda Cohen	5430938	217-66-5609
.....		.....
Lisa Kobayashi	9290124	177-23-0199

# Distributed Hash Table (DHT)

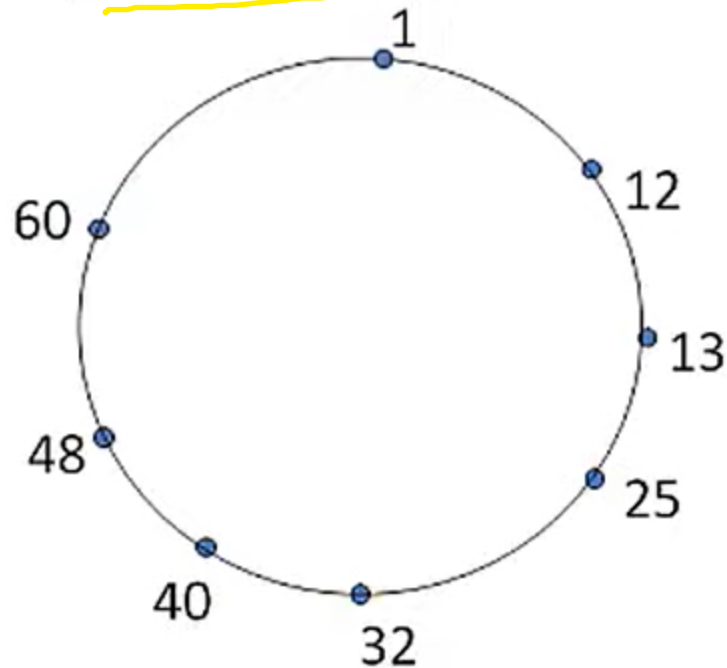
- Distribute (key, value) pairs over millions of peers
  - pairs are evenly distributed over peers
- Any peer can query database with a key
  - database returns value for the key
  - To resolve query, small number of messages exchanged among peers
- Each peer only knows about a small number of other peers
- Robust to peers coming and going (churn)

# Assign key-value pairs to peers

- rule: assign key-value pair to the peer that has the closest ID.
- convention: closest is the immediate successor of the key.
- e.g., ID space  $\{0, 1, 2, 3, \dots, 63\}$
- suppose 8 peers: 1, 12, 13, 25, 32, 40, 48, 60
  - If key = 51, then assigned to peer 60
  - If key = 60, then assigned to peer 60
  - If key = 61, then assigned to peer 1

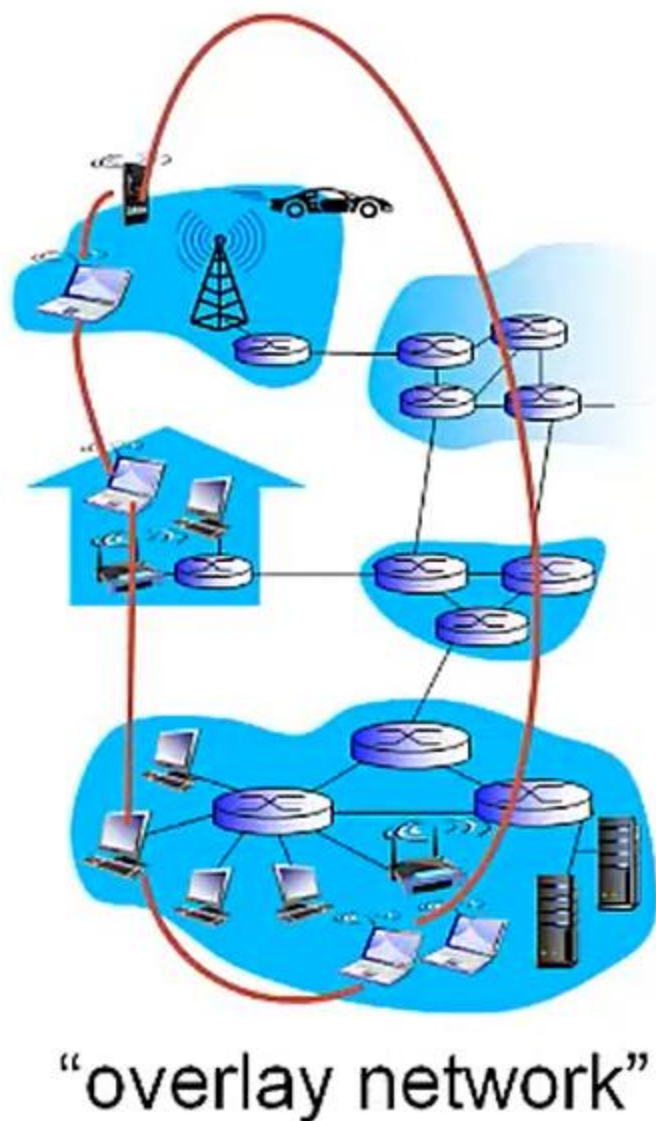
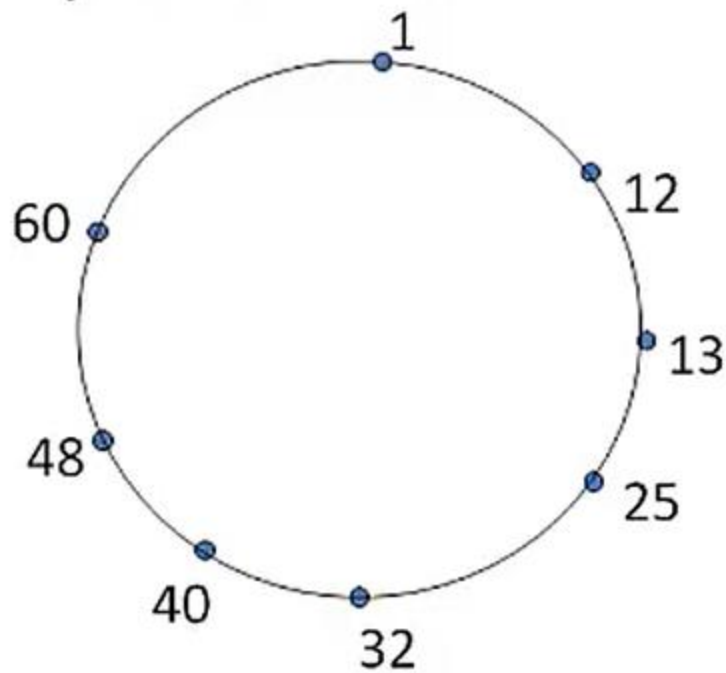
# Circular DHT

- each peer only aware of immediate successor and predecessor.



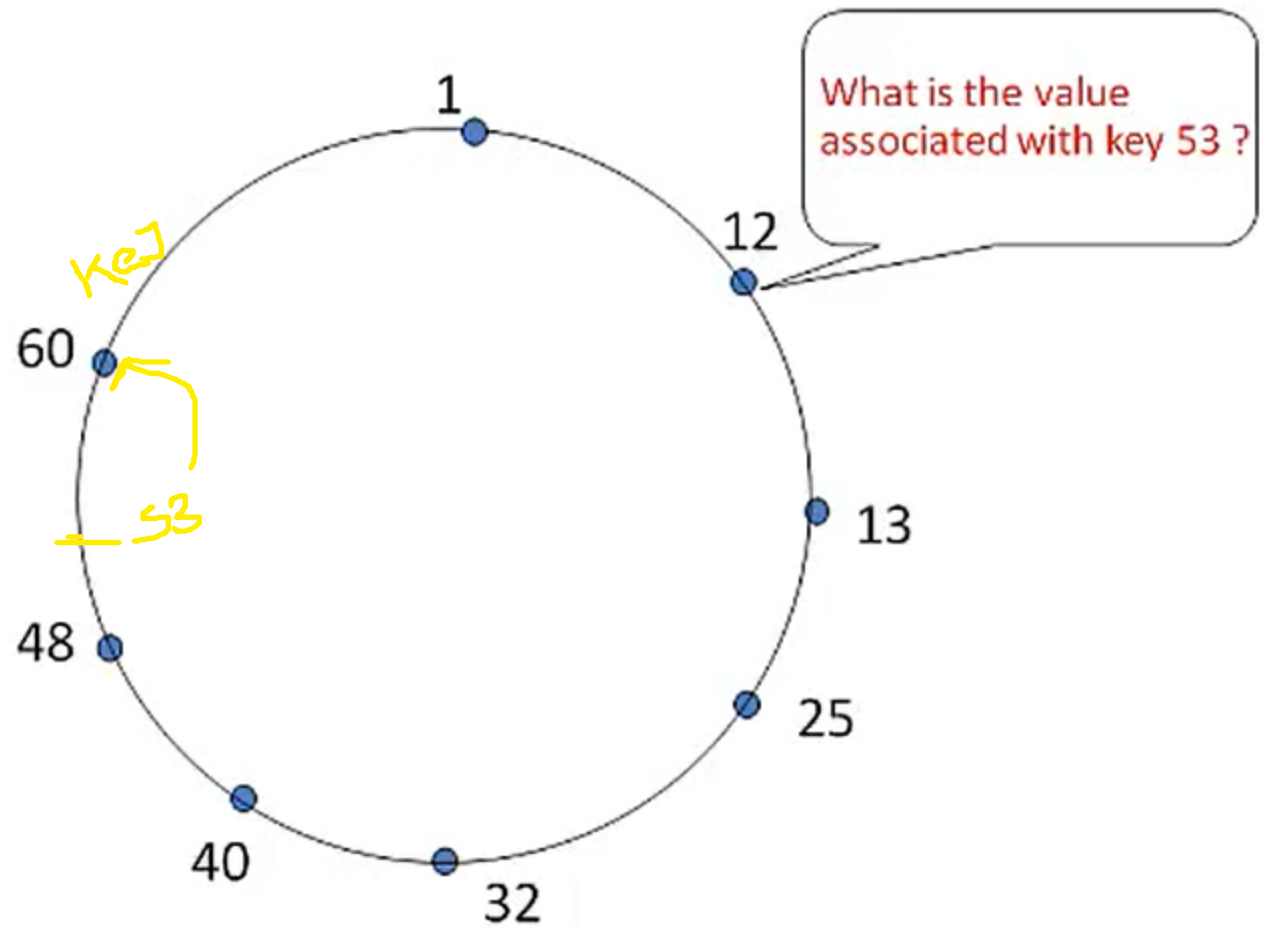
# Circular DHT

- each peer *only* aware of immediate successor and predecessor.

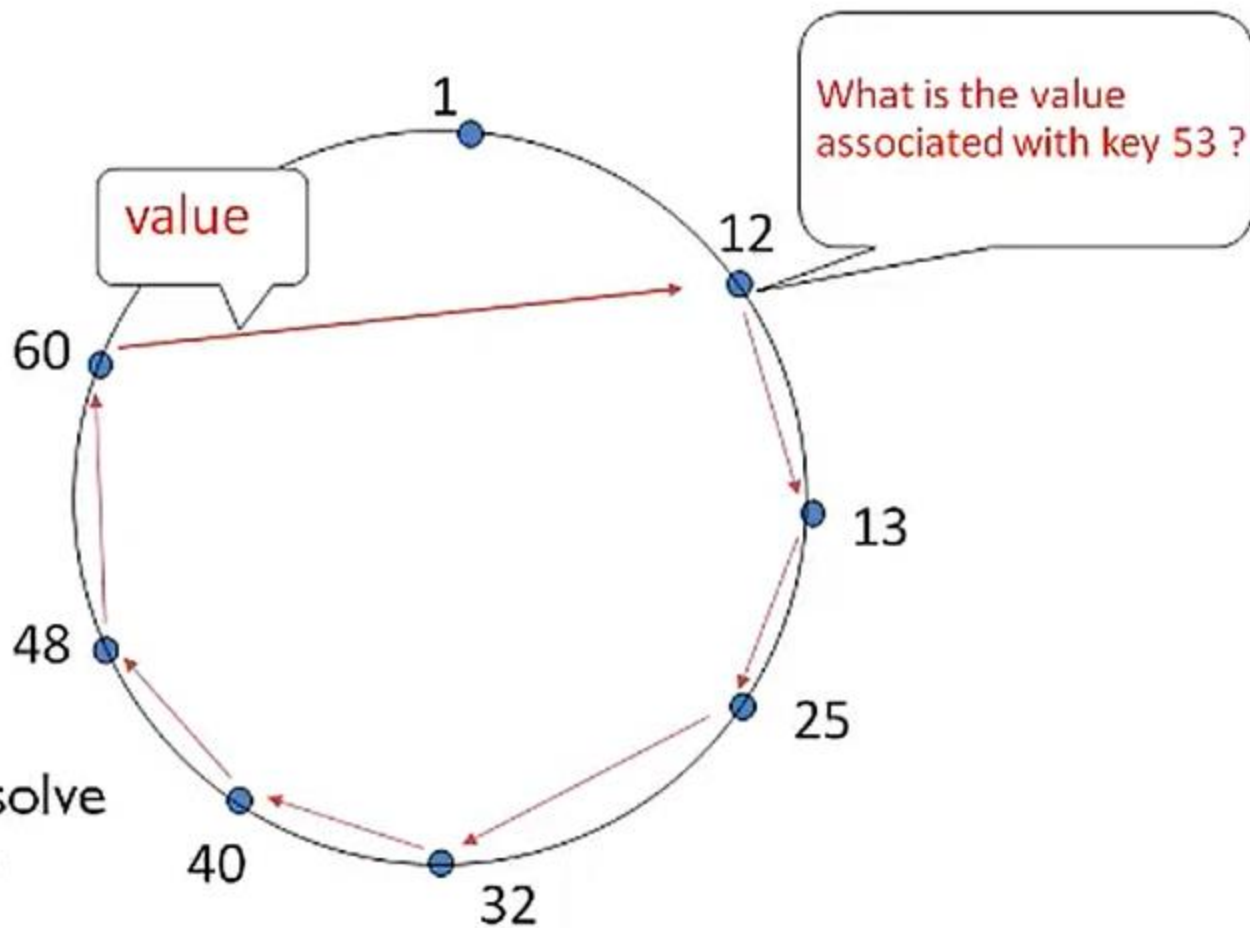




# Resolving a query



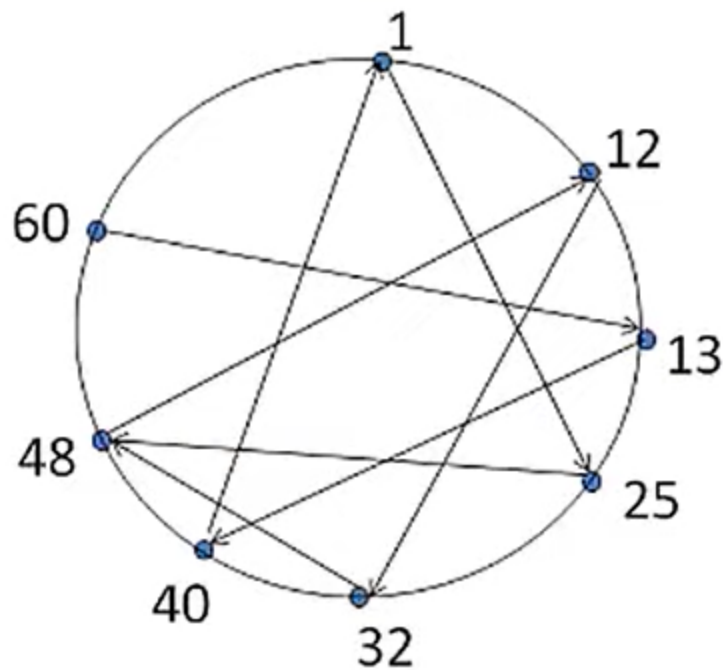
# Resolving a query



$O(N)$  messages  
on average to resolve  
query, when there  
are  $N$  peers

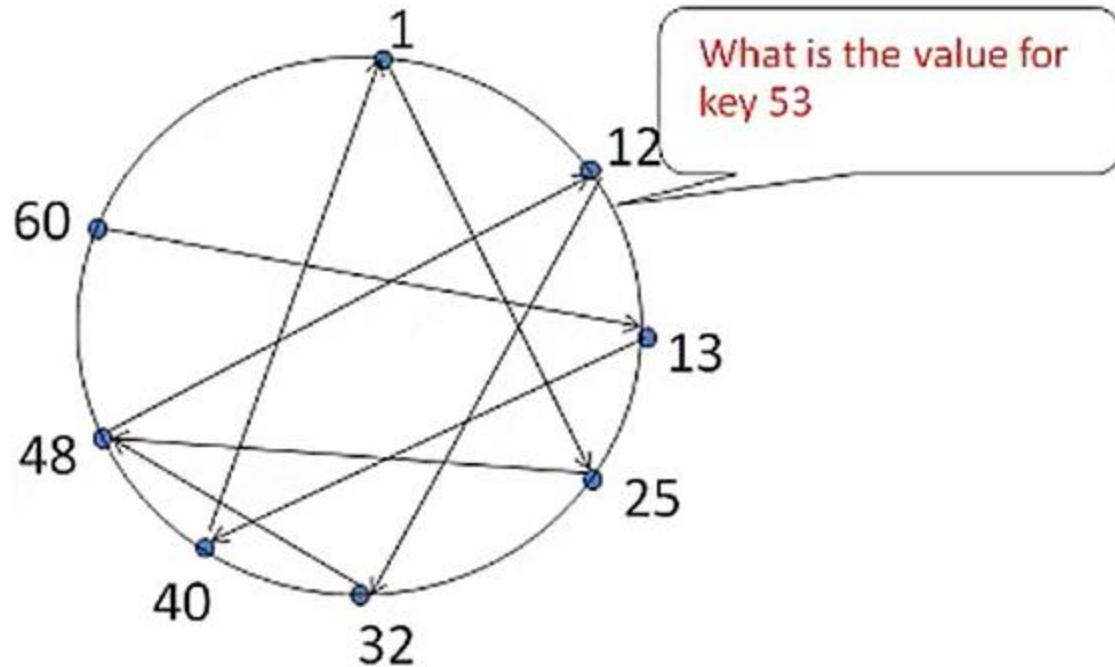


# Circular DHT with shortcuts



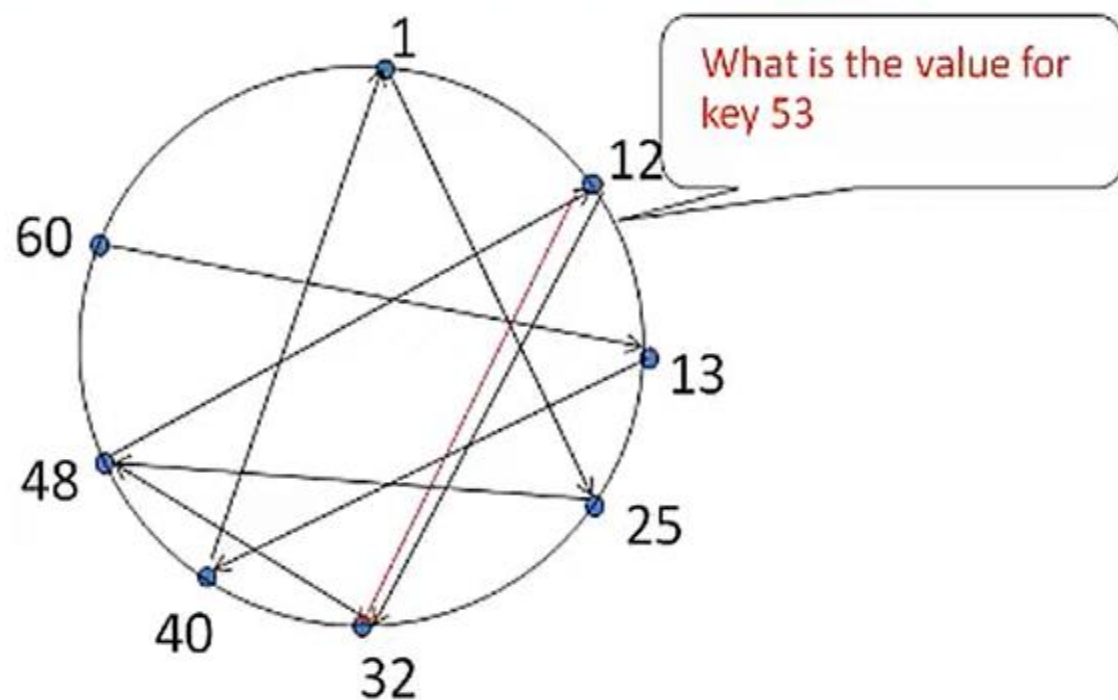
- each peer keeps track of IP addresses of predecessor, successor, short cuts.

# Circular DHT with shortcuts



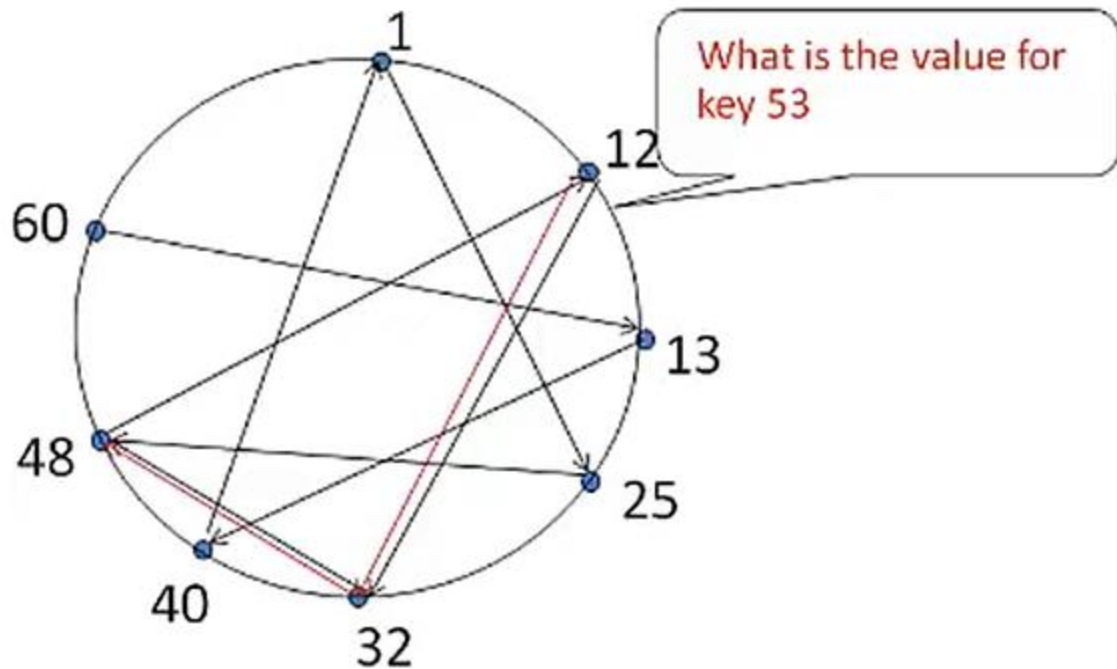
- each peer keeps track of IP addresses of predecessor, successor, short cuts.

# Circular DHT with shortcuts



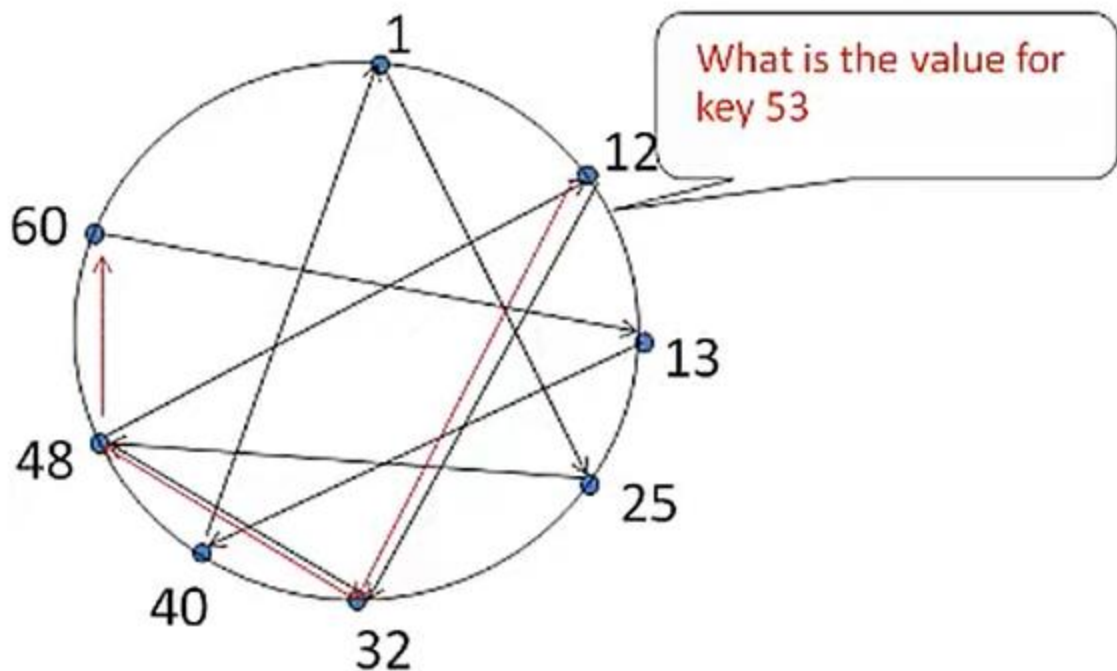
- each peer keeps track of IP addresses of predecessor, successor, short cuts.

# Circular DHT with shortcuts



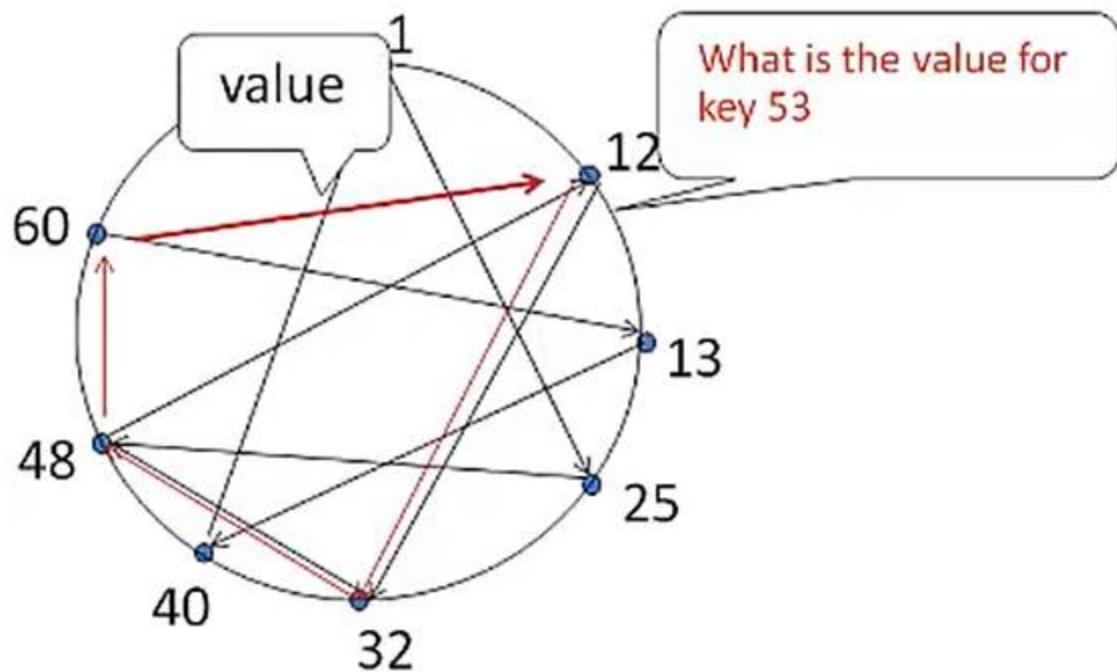
- each peer keeps track of IP addresses of predecessor, successor, short cuts.

# Circular DHT with shortcuts



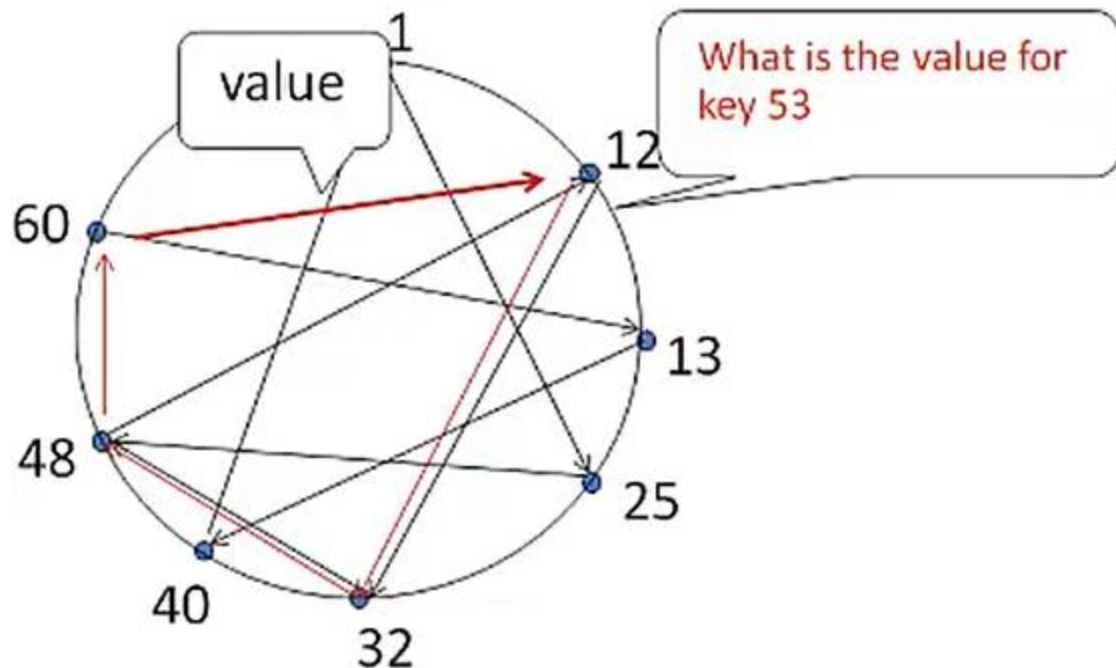
- each peer keeps track of IP addresses of predecessor, successor, short cuts.

# Circular DHT with shortcuts



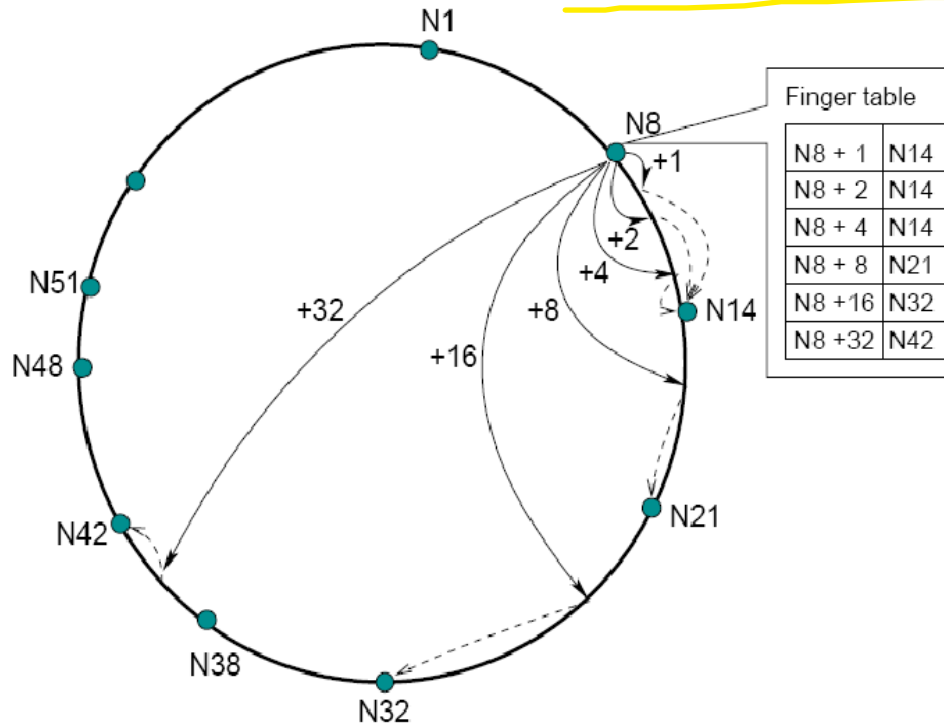
- each peer keeps track of IP addresses of predecessor, successor, short cuts.

## Circular DHT with shortcuts



- each peer keeps track of IP addresses of predecessor, successor, short cuts.
- reduced from 6 to 3 messages.

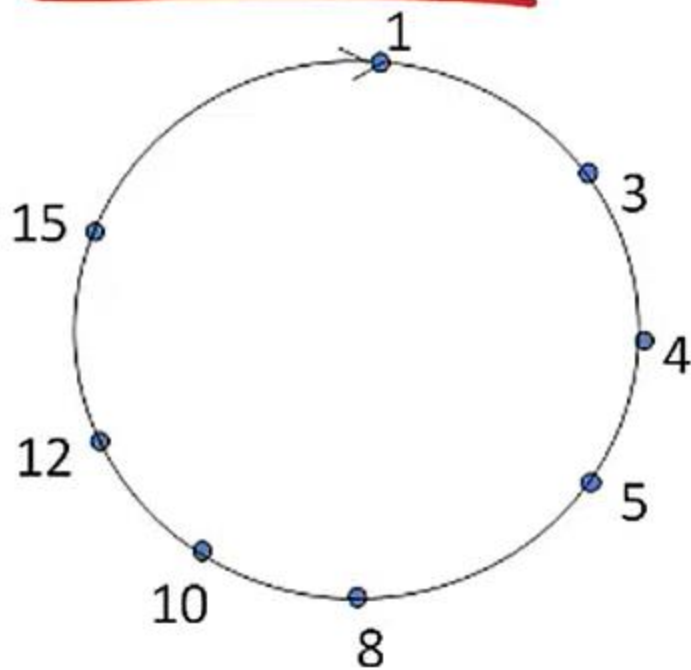
# Chord - Finger table



- <https://pdos.csail.mit.edu/papers/ton:chord/paper-ton.pdf>
- Reduces time complexity of lookup to  $O(\log n)$



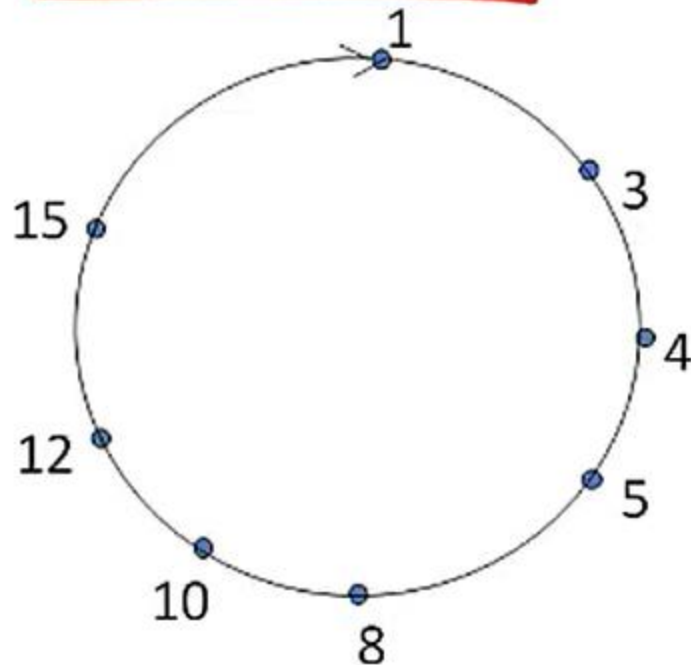
# Peer churn



handling peer churn:

❖ peers may come and go (churn)

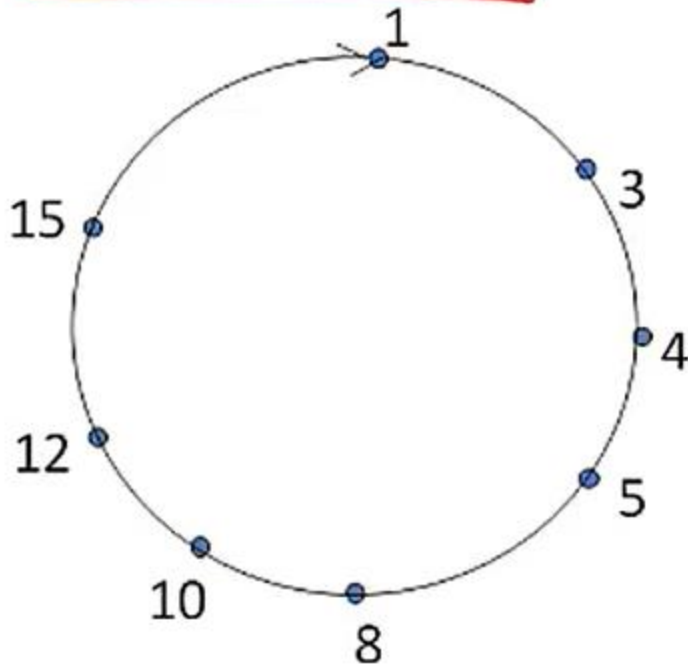
# Peer churn



## handling peer churn:

- ❖ peers may come and go (churn)
- ❖ each peer knows address of its two successors

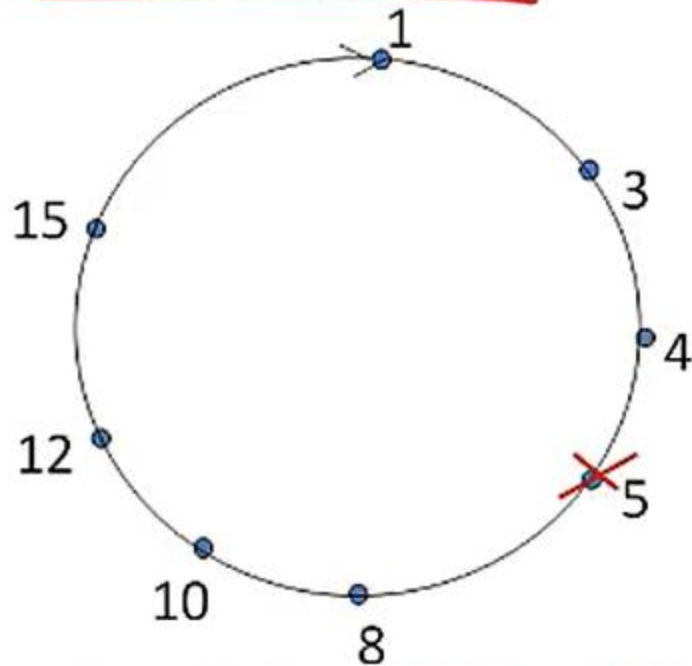
# Peer churn



## handling peer churn:

- ❖ peers may come and go (churn)
- ❖ each peer knows address of its two successors
- ❖ each peer periodically pings its two successors to check aliveness

# Peer churn

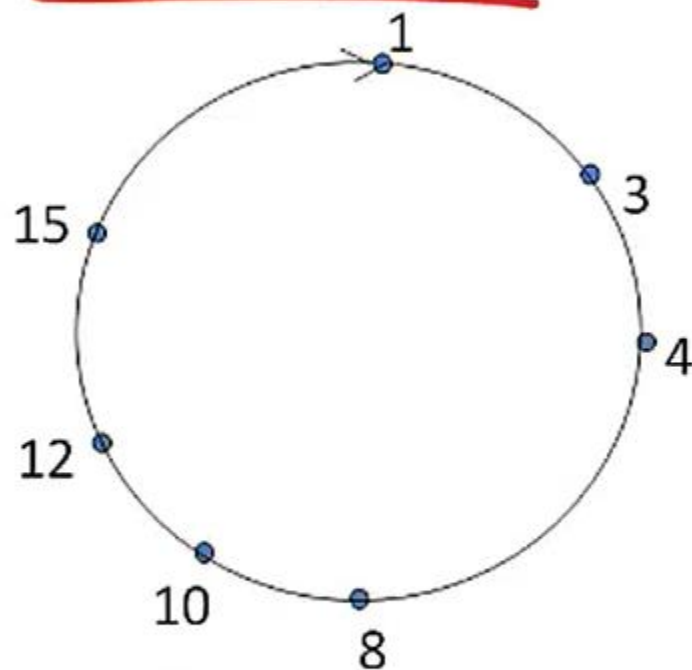


*example: peer 5 abruptly leaves*

## handling peer churn:

- ❖ peers may come and go (churn)
- ❖ each peer knows address of its two successors
- ❖ each peer periodically pings its two successors to check aliveness
- ❖ if immediate successor leaves, choose next successor as new immediate successor

# Peer churn



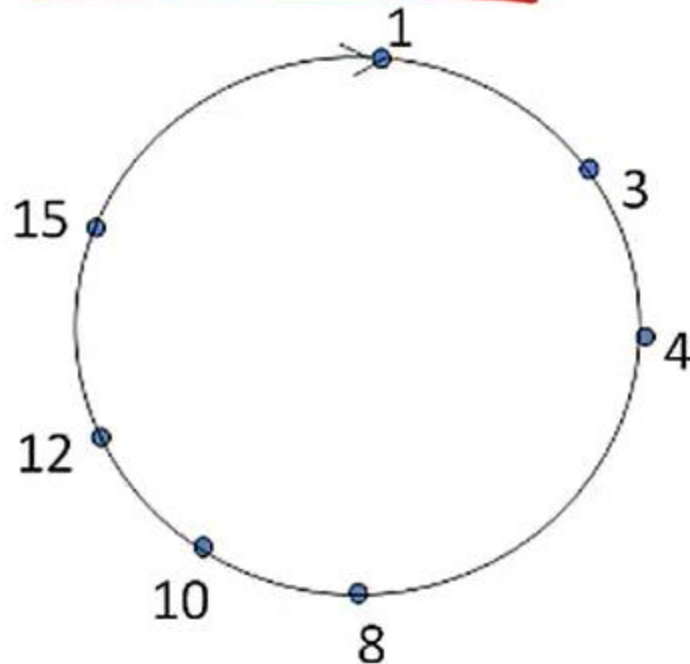
## handling peer churn:

- ❖ peers may come and go (churn)
- ❖ each peer knows address of its two successors
- ❖ each peer periodically pings its two successors to check aliveness
- ❖ if immediate successor leaves, choose next successor as new immediate successor

## *example: peer 5 abruptly leaves*

- peer 4 detects peer 5's departure; makes 8 its immediate successor

# Peer churn



## handling peer churn:

- ❖ peers may come and go (churn)
- ❖ each peer knows address of its two successors
- ❖ each peer periodically pings its two successors to check aliveness
- ❖ if immediate successor leaves, choose next successor as new immediate successor

## *example: peer 5 abruptly leaves*

- peer 4 detects peer 5's departure; makes 8 its immediate successor
- 4 asks 8 who its immediate successor is; makes 8's immediate successor its second successor.

# Peer Churn

- What happens to the data (values) that were stored in peer 5 in the previous example?
- Have to duplicate data in multiple peers (maybe the successors)

# Member join

- Have to update the ids when new members join the network
- P2P is generally more complex than C/S due to peer join, peer churn and distributed routing



# Possibilities with P2P?

- Currently mostly used for file sharing
- Blockchain uses P2P
- Online Social networks?
- Taxi applications?
- Etc etc
- <https://blockonomi.com/youtube-alternative/>
- <https://dailycoin.com/best-decentralized-social-media-platforms-top-10-alternatives-to-consider/>

# Vertical vs Horizontal Distribution

- Traditional client-server architectures exhibit **vertical distribution across tiers**. Each tier serves a different purpose in the system. Ex- 3-tier architecture
  - *Logically* different components reside on different nodes
- **Horizontal distribution**: each node has roughly the same processing capabilities and stores/manages part of the total system data.
  - Better load balancing, more resistant to denial-of-service attacks, harder to manage than C/S
  - Communication & control is **not hierarchical**; all about equal
  - **P2P/Hybrid** are also examples of horizontal distribution

# Volunteer computing

- Seti@Home
- BOINC (<https://boinc.berkeley.edu/>)
- [https://en.wikipedia.org/wiki/Volunteer\\_computing](https://en.wikipedia.org/wiki/Volunteer_computing)

# Architecture versus Middleware

Volunteer computing is a form of horizontal distribution, where individuals donate their computing power to a distributed system.

- Where does middleware fit into an architecture?
- **Middleware:** the software layer between user applications and distributed platforms.
- Purpose: to provide distribution transparency
  - Applications can access programs running on remote nodes without understanding the remote environment

# Architecture versus Middleware

- Middleware may also have an architecture
  - e.g., CORBA has an component-based style.
- Use of a specific architectural style can make it easier to develop applications, but it may also lead to a less flexible system.
- Possible solution: develop middleware that can be customized as needed for different applications with different architectures.

# Summary

- Software Architecture Vs System Architecture
- Different Software Architectural styles – Layered, Component based, event driven, data centered...
- Can have combinations of these styles
- Different System Architectures – Client Server, Peer to Peer, Hybrid
- P2P – structured/unstructured, Distributed Hash Tables (DHT)

