Distributed Systems Design
COMP 6231

Architectures

Lecture 2

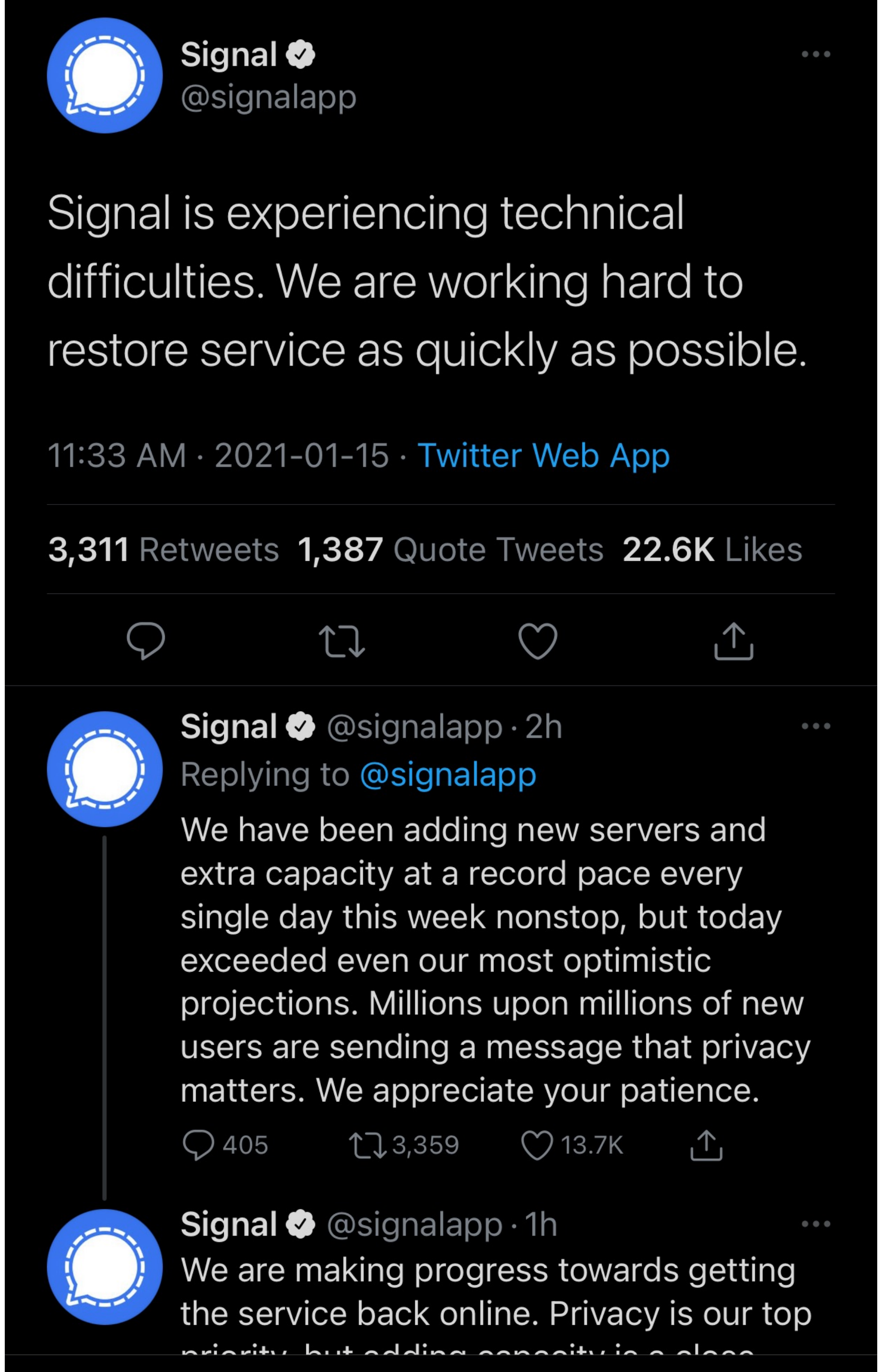
Essam Mansour

Today…

- **Last Session:**
  - Introduction

- **This Session:**

- Architectures for distributed systems *(Chapter 2)*
  - Architectural styles
  - Client-server architectures
  - Decentralized and peer-to-peer architectures



Signal ✓
@signalapp

Signal is experiencing technical difficulties. We are working hard to restore service as quickly as possible.

11:33 AM · 2021-01-15 · Twitter Web App

3,311 Retweets **1,387** Quote Tweets **22.6K** Likes

Signal ✓ @signalapp · 2h
Replying to @signalapp

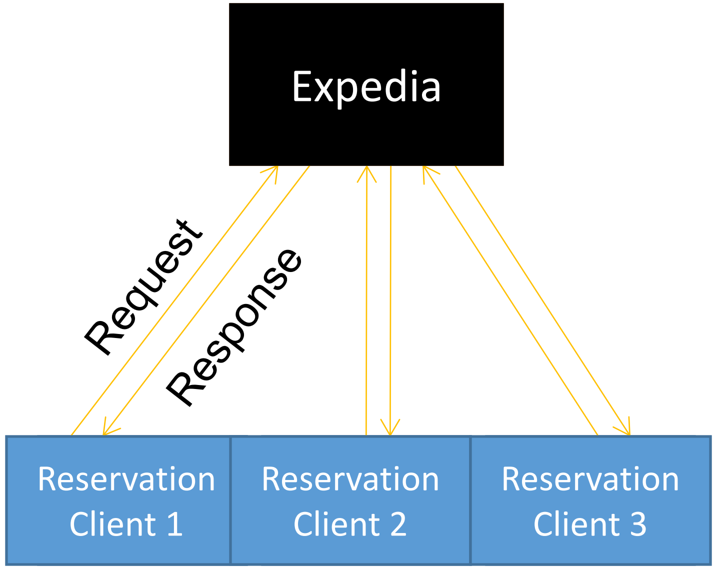
We have been adding new servers and extra capacity at a record pace every single day this week nonstop, but today exceeded even our most optimistic projections. Millions upon millions of new users are sending a message that privacy matters. We appreciate your patience.

405 3,359 13.7K

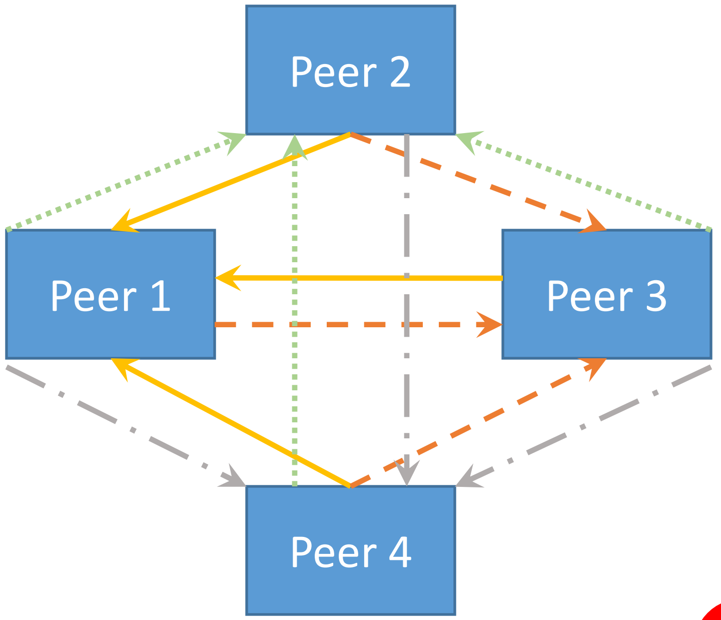
Signal ✓ @signalapp · 1h

We are making progress towards getting the service back online. Privacy is our top priority, but adding capacity is a close

Bird’s Eye View of Some Distributed Systems



Google Search
Airline Booking



Bit-torrent
Skype

Is it still the case ?

How would one characterize these distributed systems?

Simple Characterization of Distributed Systems

- What are the entities that are communicating in a DS?
  a) Communicating entities (system-oriented vs. problem-oriented entities)

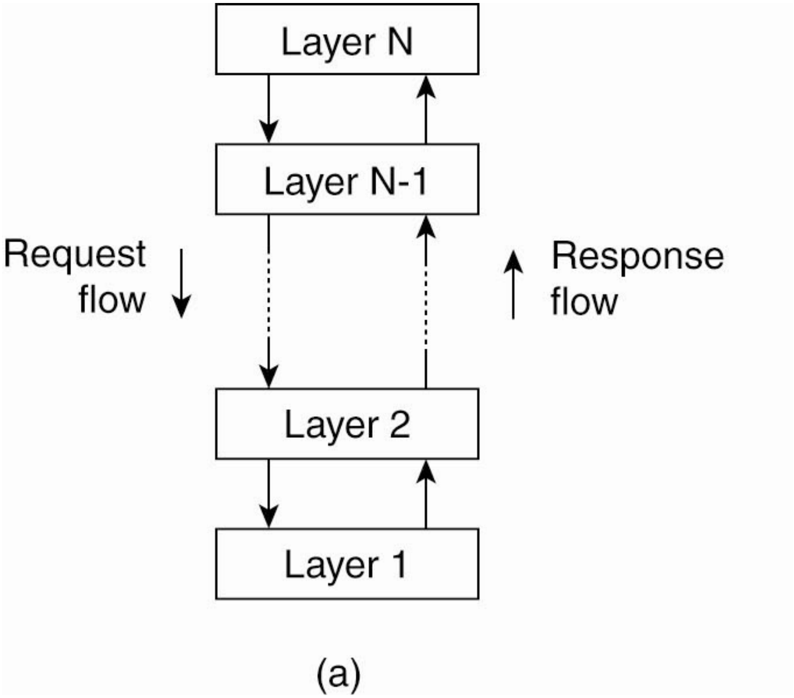
- How do the entities communicate?
  b) Communication paradigms (sockets and RPC)

- What roles and responsibilities do the entities have?
  c) This could lead to different organizations (referred, henceforth, to as *architectures*)

Architectural Styles

- Important styles of architecture for distributed systems
  - Layered architectures
  - Object-based architectures
  - Data-centered architectures
  - Event-based architectures
  - Resource-based architectures

Layered Design



(a)

- Each layer uses previous layer to implement new functionality that is exported to the layer above
- Example: Multi-tier web apps

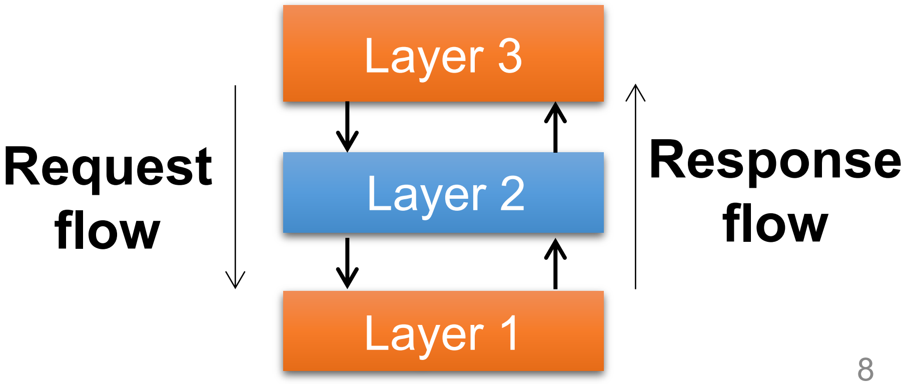
Layering

- A complex system is partitioned into layers
  - Upper layer utilizes the services of the lower layer
  - A *vertical organization* of services

- For example, a three-layer solution could easily be deployed on a single tier, such as a personal workstation.

- Layering simplifies the design of complex distributed systems by hiding the complexity of below layers

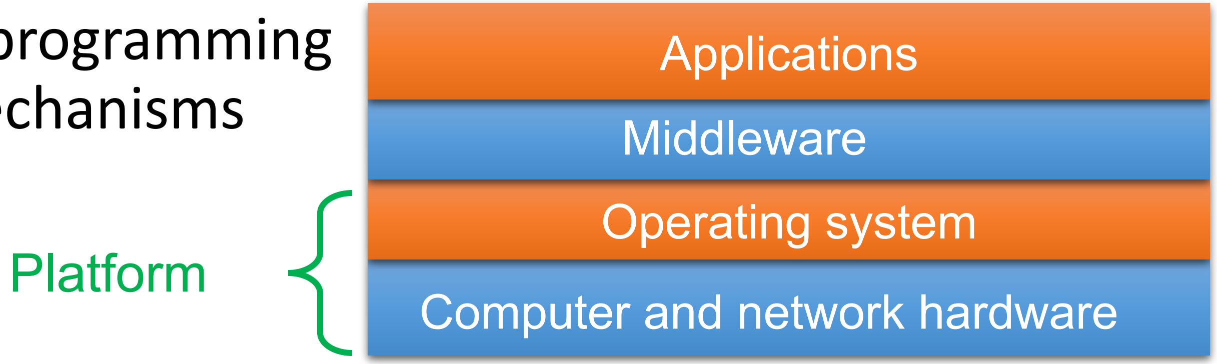
- Control flows from layer to layer



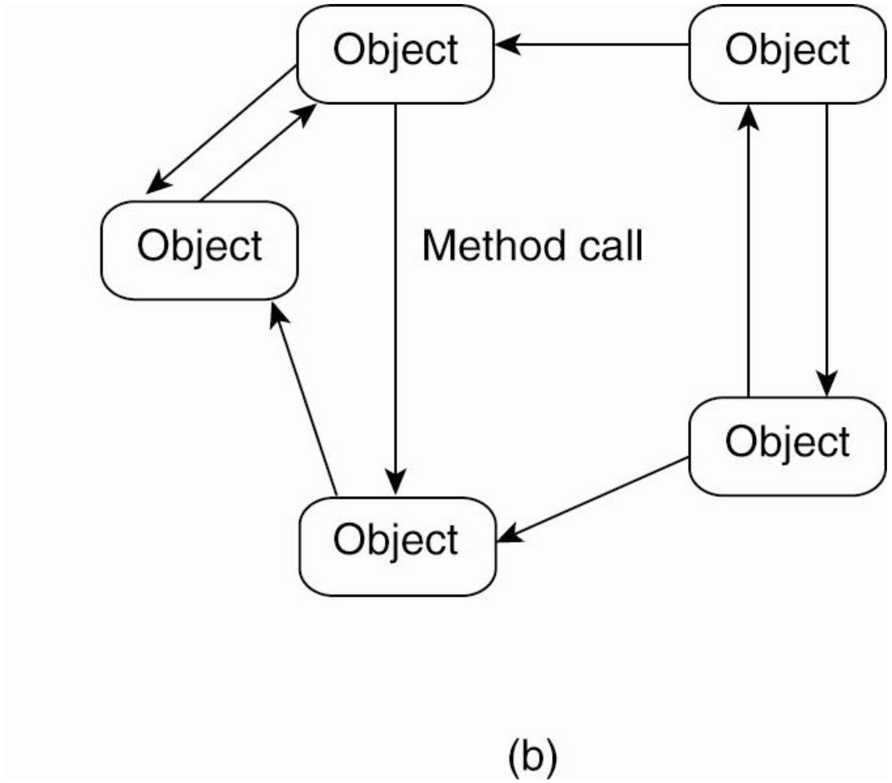
Layering – Platform and middleware

- Distributed systems can be organized into three layers:

  1. Platform
     - Low-level hardware and software layers
     - Provides common services for higher layers
  2. Middleware
     - Masks heterogeneity and provides convenient programming models to application programmers
     - Typically, it simplifies application programming by abstracting communication mechanisms
  3. Applications



Object-based Style

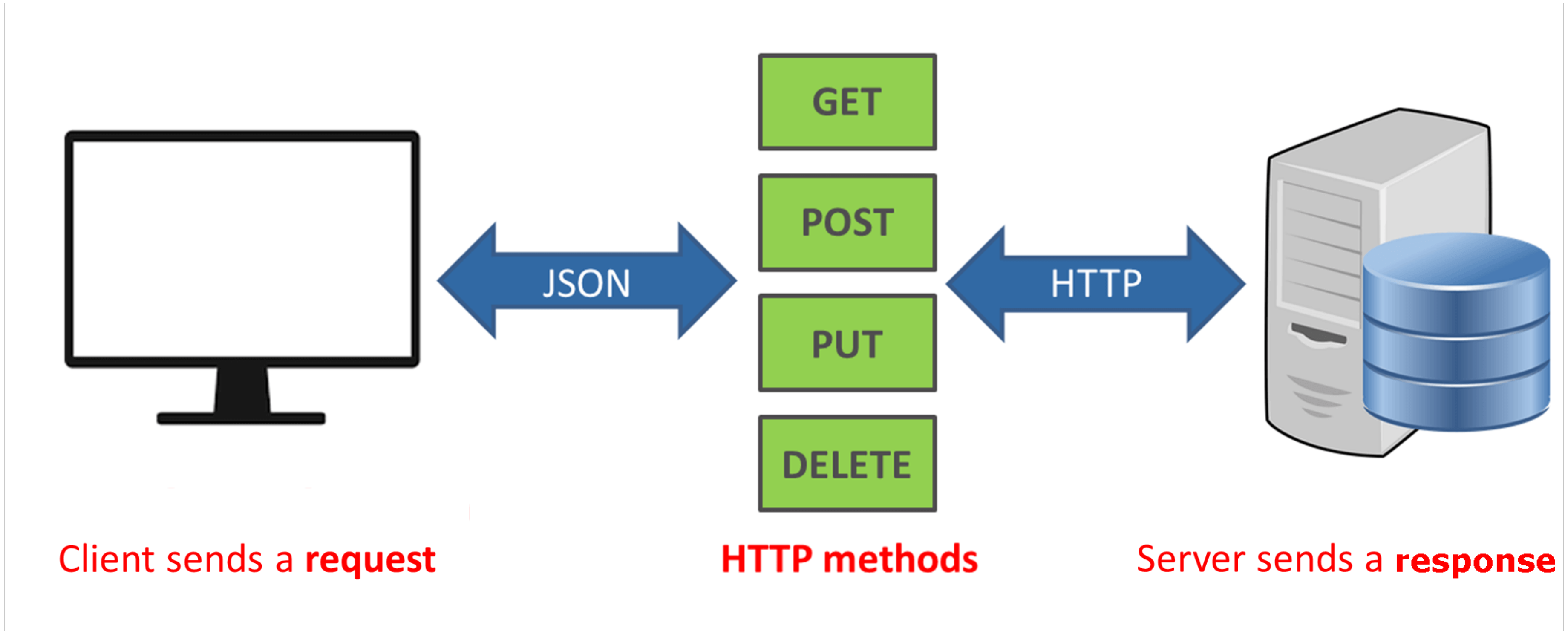


(b)

- Each object corresponds to a components
- Components interact via remote procedure calls
  - Popular in client-server systems

Resource-oriented Architecture

- Example of ROA: Representational State Transfer (REST)



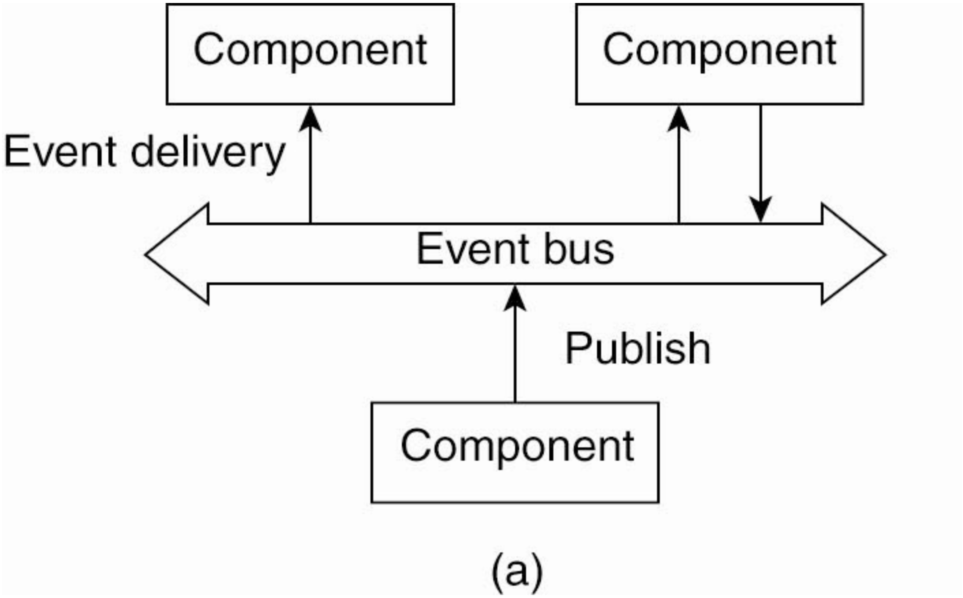
Resource-oriented Architecture

- Example of ROA: Representational State Transfer (REST)
  - Basis for RESTful web services
  - Resources identified through a single naming scheme
    - Uniform Resource Identifier (URI)
  - All services offer same interface (e.g., 4 HTTP operations)
    - **Get** / **Put** / **Delete** / **Post** HTTP operations
  - Messages are fully described
  - No state of the caller is kept (stateless execution)
  - Example: use HTTP for API
    - http://bucketname.s3.aws.com/objName
  - Return JSON objects

```
{"name":"test.com","messages":["msg 1","msg 2","msg 3”],"age":100}
```

  - **Discuss**: Service-oriented (SOA) vs. Resource-oriented (ROA)

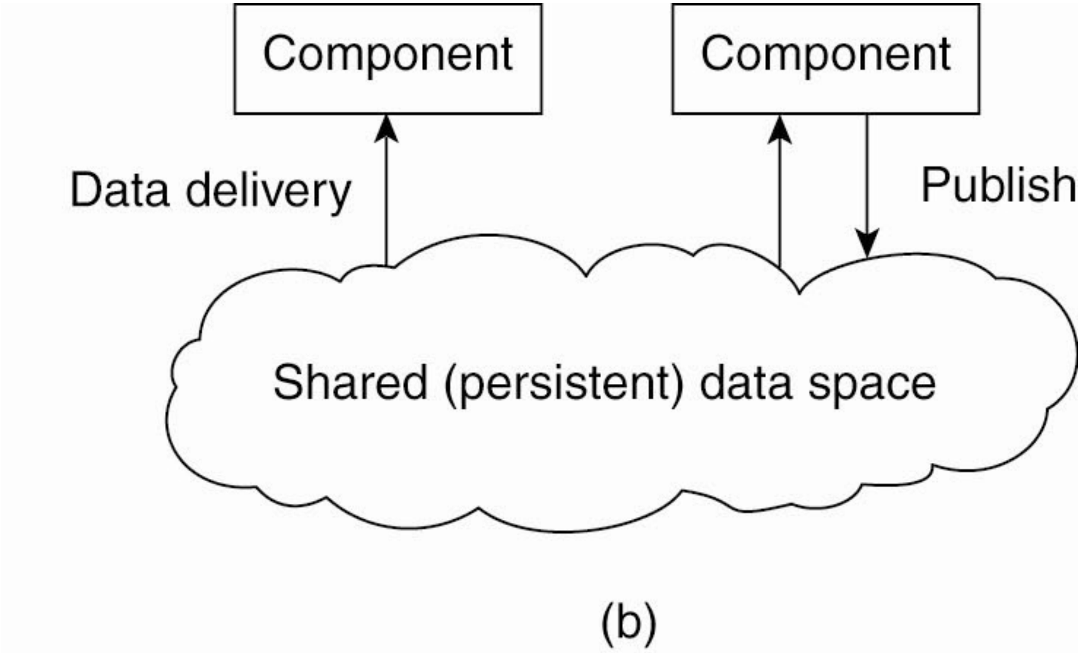
Event-based architecture



(a)

- Communicate via a common repository
  - Use a publish-subscribe paradigm
  - Consumers subscribe to types of events
  - Events are delivered once published by any publisher

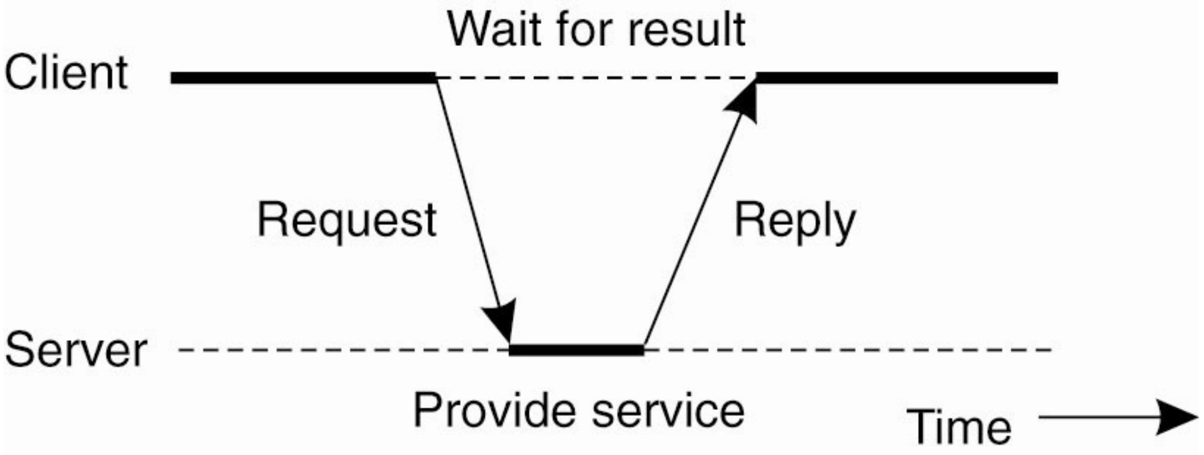
Shared data-space



(b)

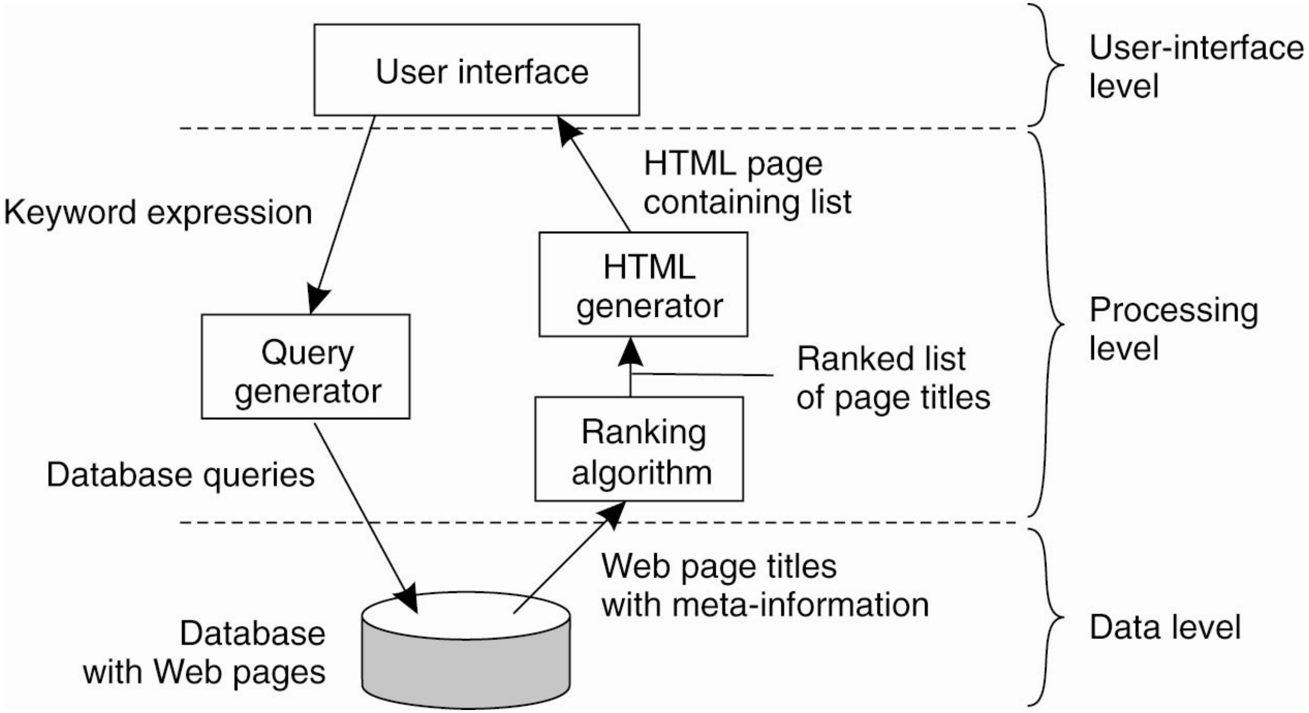
- “Bulletin-board” architecture
    - Decoupled in space and time
    - Post items to shared space; consumers pick up at a later time

Client-Server Architectures



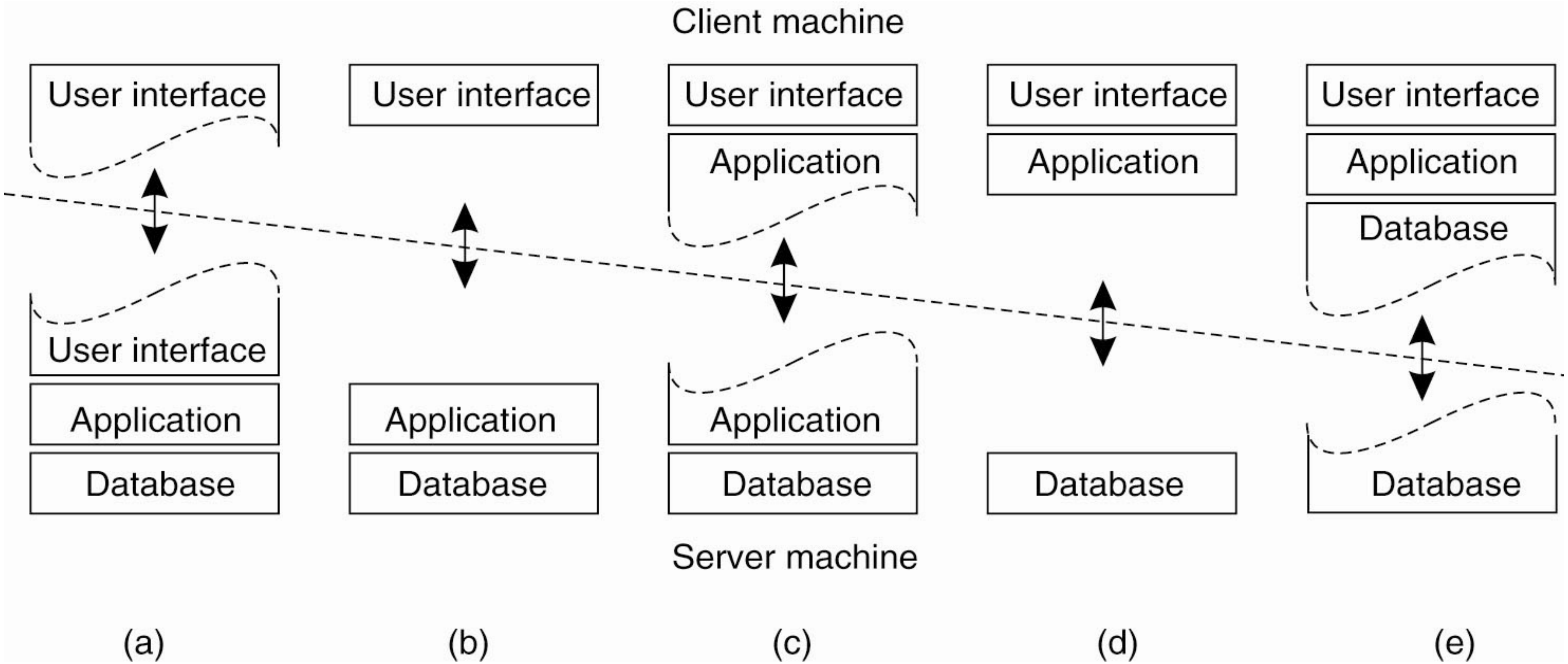
- Most common style: client-server architecture
- Application layering
    - User-interface level
    - Processing level
    - Data level

Search Engine Example



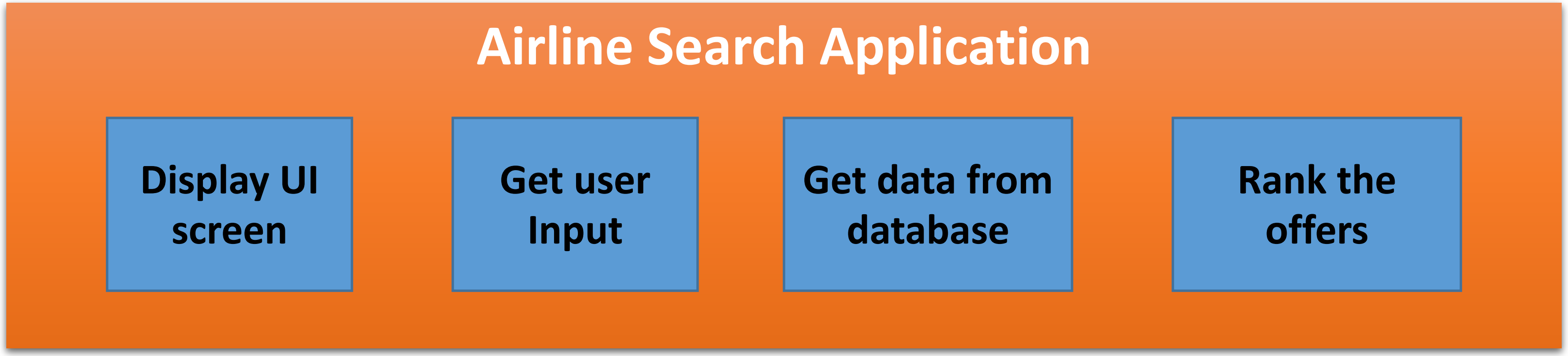
Search engine architecture with 3 layers

A Spectrum of Choices



Tiering

- Tiering is a technique to:
  1. Organize the functionality of a service,
  2. and place the functionality into appropriate servers
  3. a tier is a physical structuring mechanism for the system infrastructure



Airline Search Application

Display UI screen

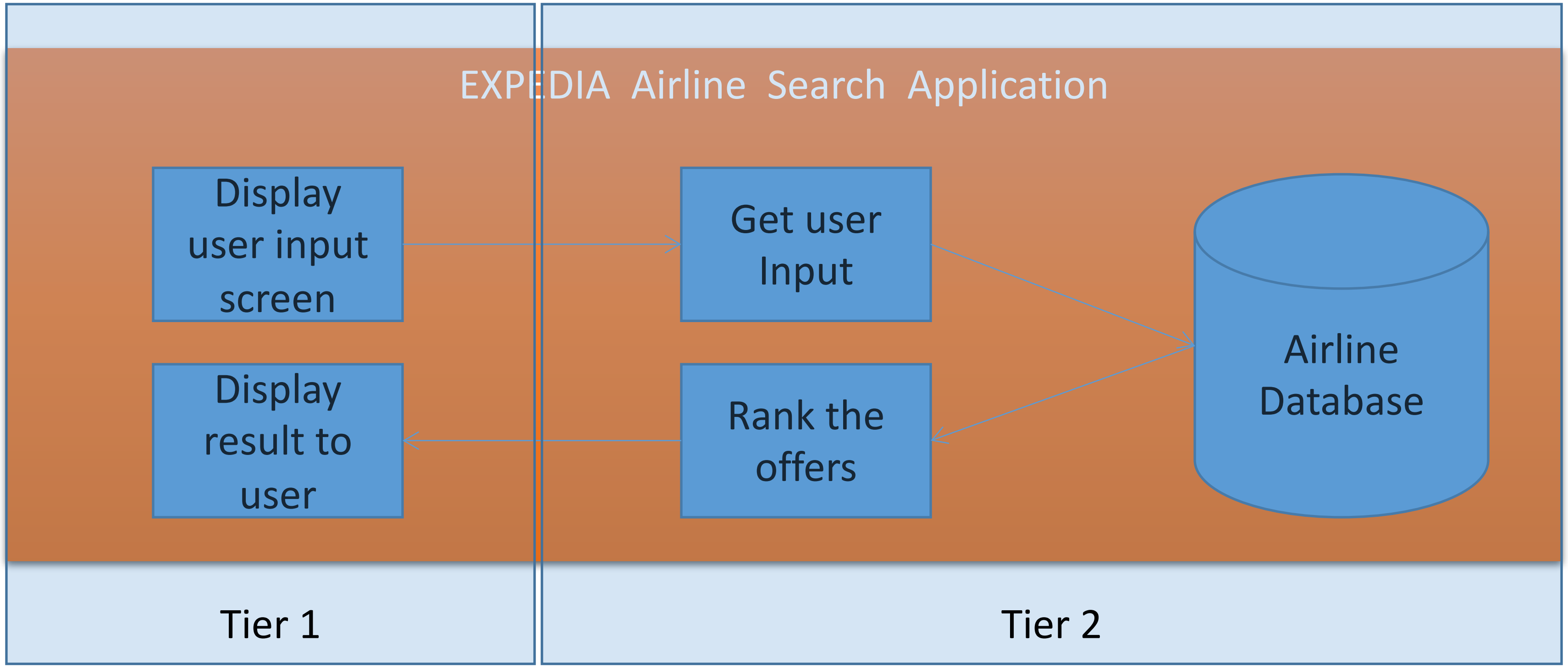
Get user Input

Get data from database

Rank the offers

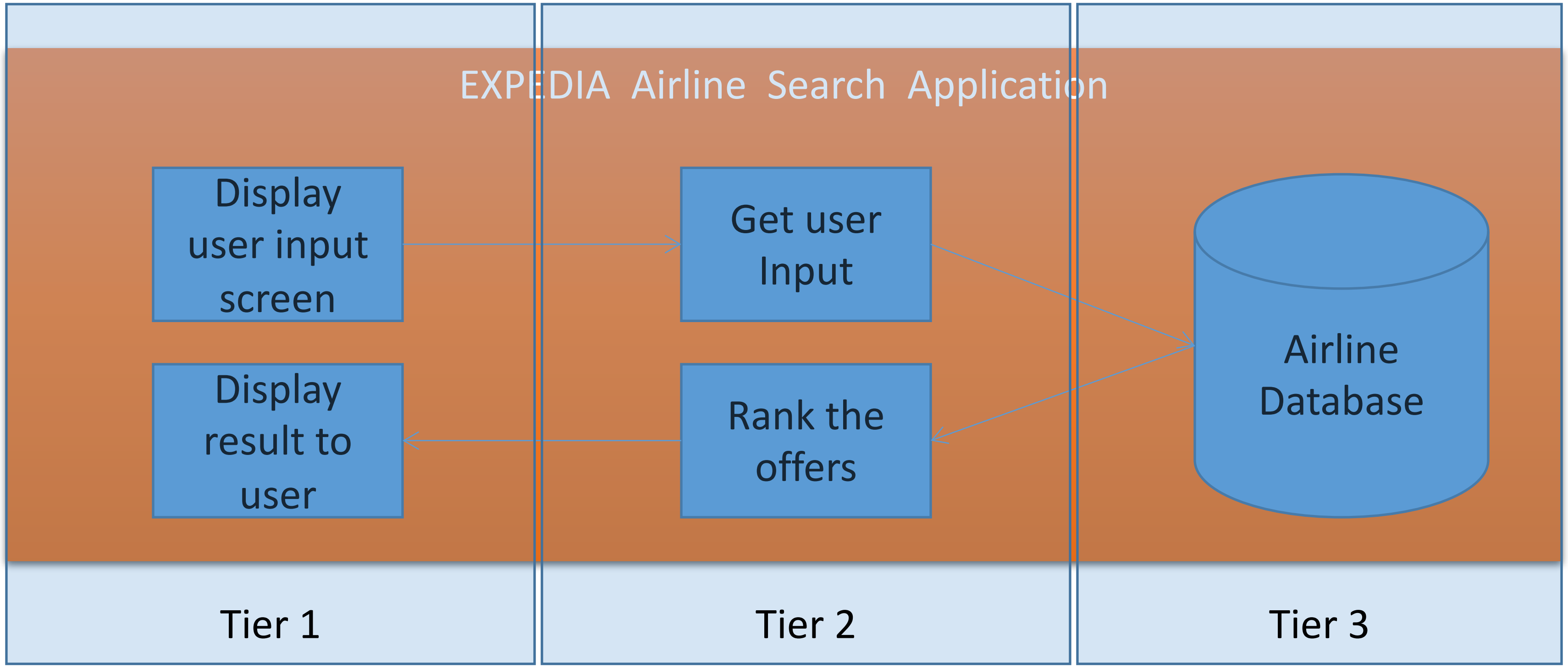
A Two-Tiered Architecture

- How would you design an airline search application?

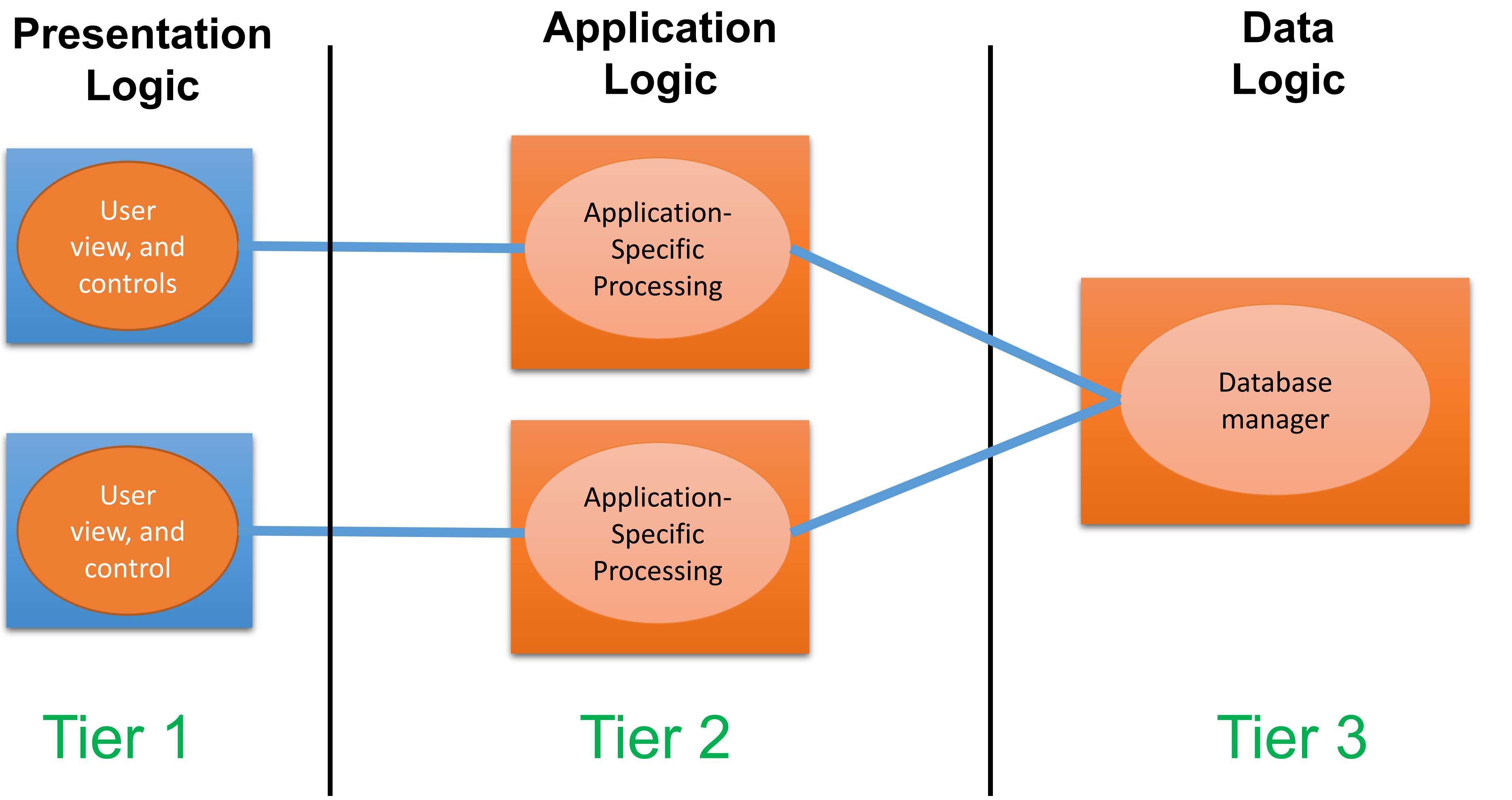


A Three-Tiered Architecture

- How would you design an airline search application?



A Three-Tiered Architecture



Presentation Logic

Application Logic

Data Logic

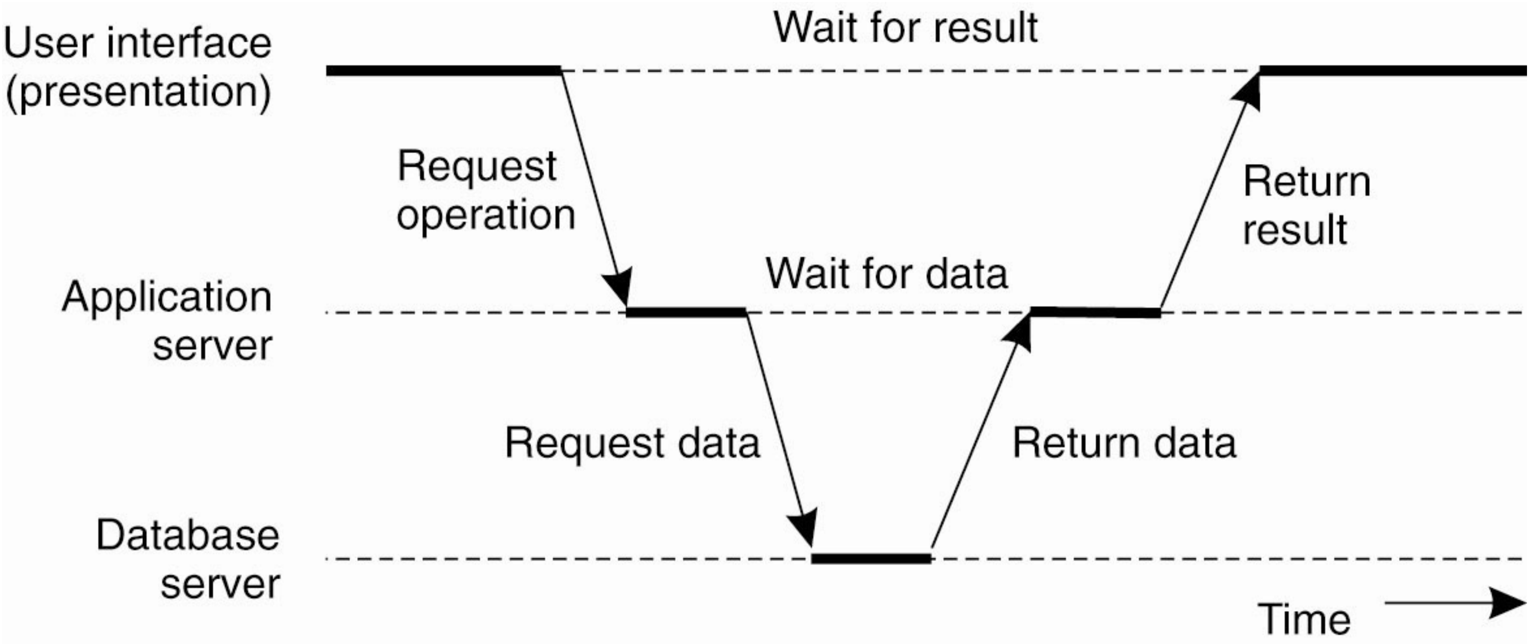
User view, and controls
User view, and control
Application-Specific Processing
Application-Specific Processing
Database manager

Tier 1

Tier 2

Tier 3

Three-tier Web Applications



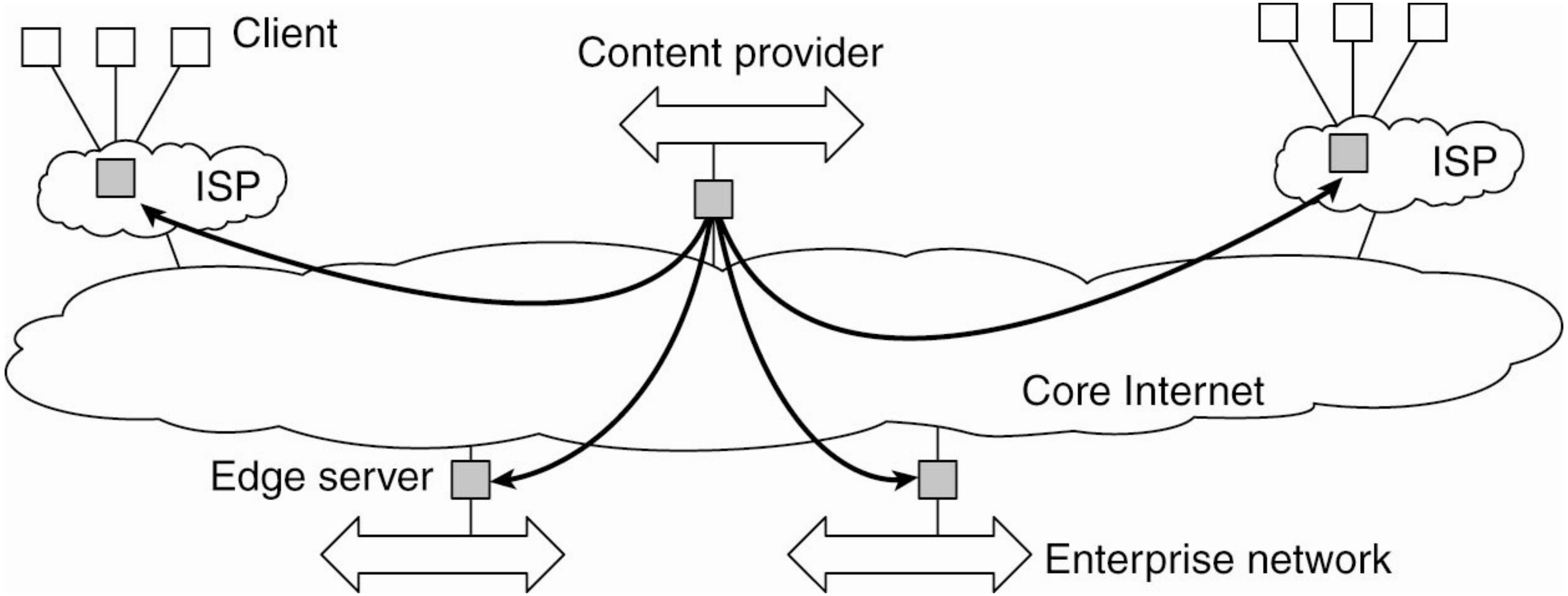
- Server itself uses a “client-server” architecture
- 3 tiers: HTTP, J2EE and database
  - Very common in most web-based applications

Three-Tiered Architecture: Pros and Cons

- Advantages:
  - Enhanced maintainability of the software (one-to-one mapping from logical elements to physical servers)
  - Each tier has a well-defined role

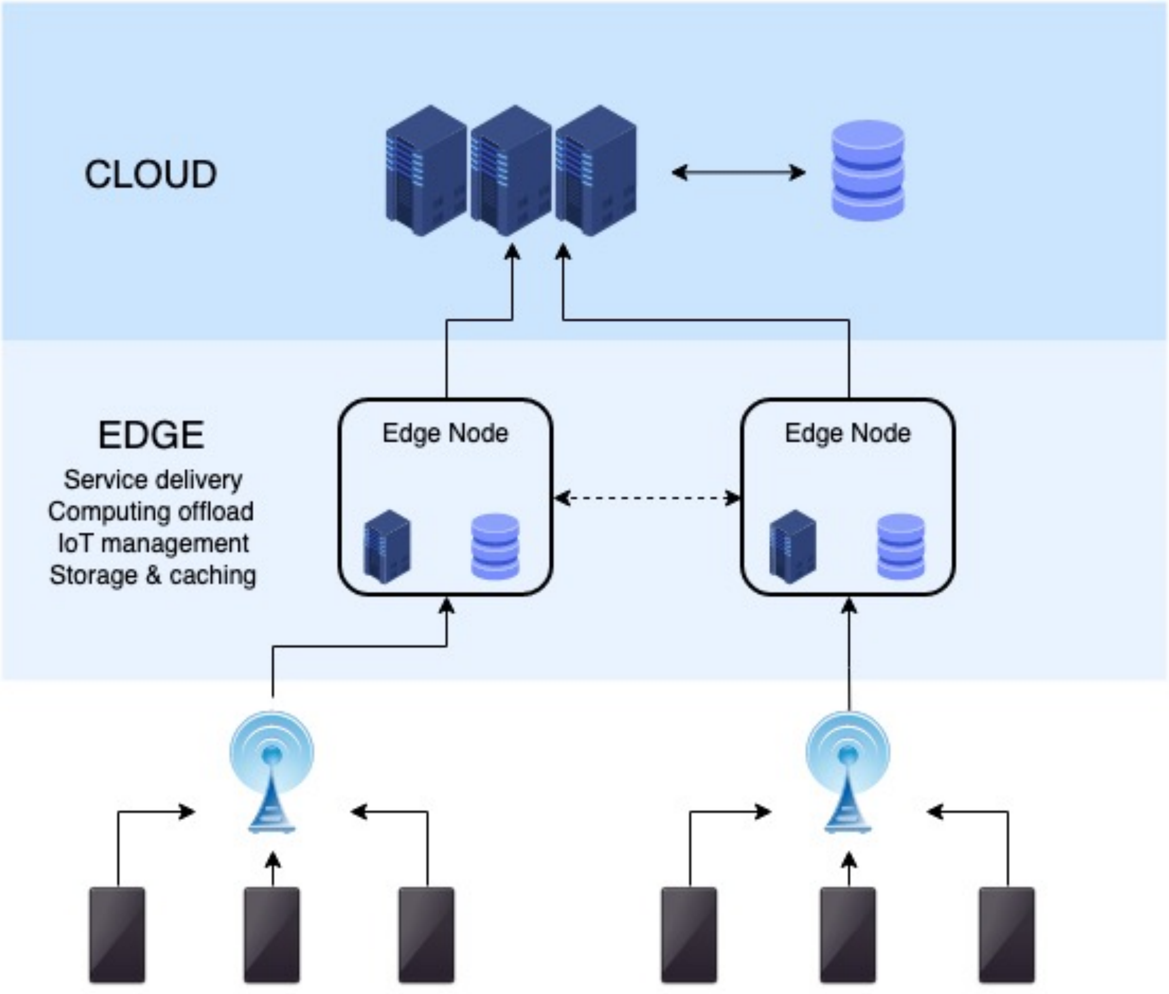
- Disadvantages:
  - Added complexity due to managing multiple servers
  - Added network traffic
  - Added latency

Edge-Server Systems



- Edge servers: from *client-server* to *client-proxy-server*
- Content distribution networks: proxies cache web content near the edge

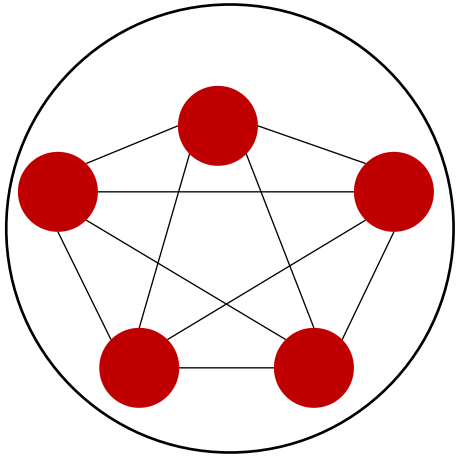
Edge-Server Systems



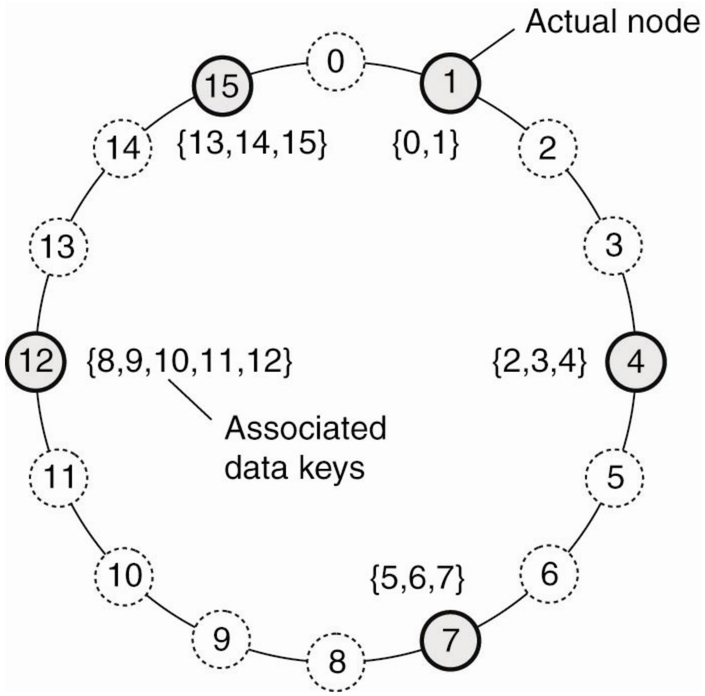
https://en.wikipedia.org/wiki/Edge_computing

Decentralized Peer-to-Peer Architecture

- A peer-to-peer (P2P) architecture can be characterized as follows:
  1) All nodes are equal (no hierarchy)
     - No Single-Point-of-Failure (SPOF)
  2) A central coordinator is not needed
     - But, decision making becomes harder
  3) The underlying system can scale out indefinitely
     - In principle, no performance bottleneck

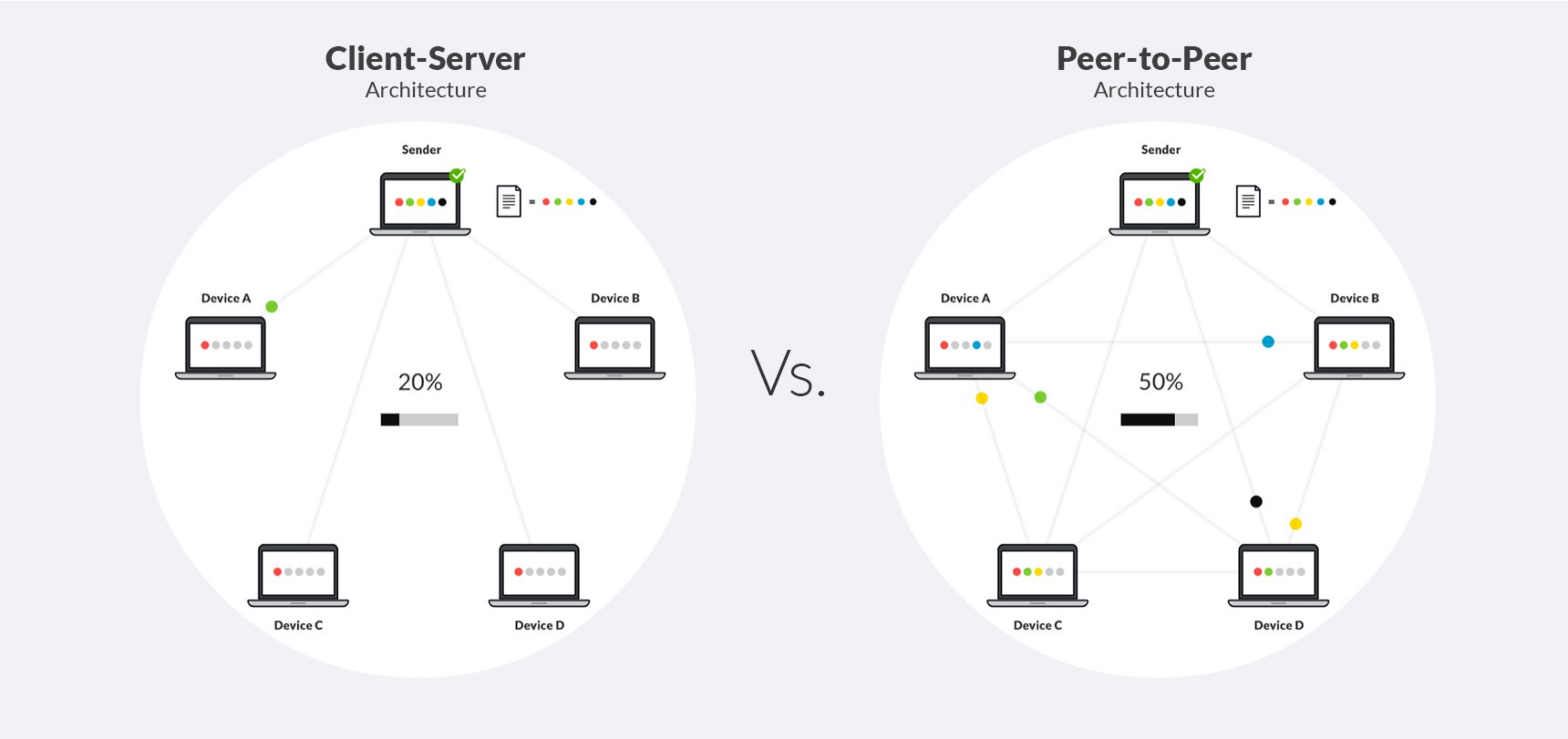


Decentralized Peer-to-Peer Architecture



- Peer-to-peer systems
  - Removes distinction between a client and a server
  - Overlay network of nodes
- Chord: structured peer-to-peer system
  - Use a distributed hash table to locate objects
    - Data item with key *k* -> smallest node with id >= k

Client-Server vs Peer-to-Peer



Peer-to-Peer Architecture

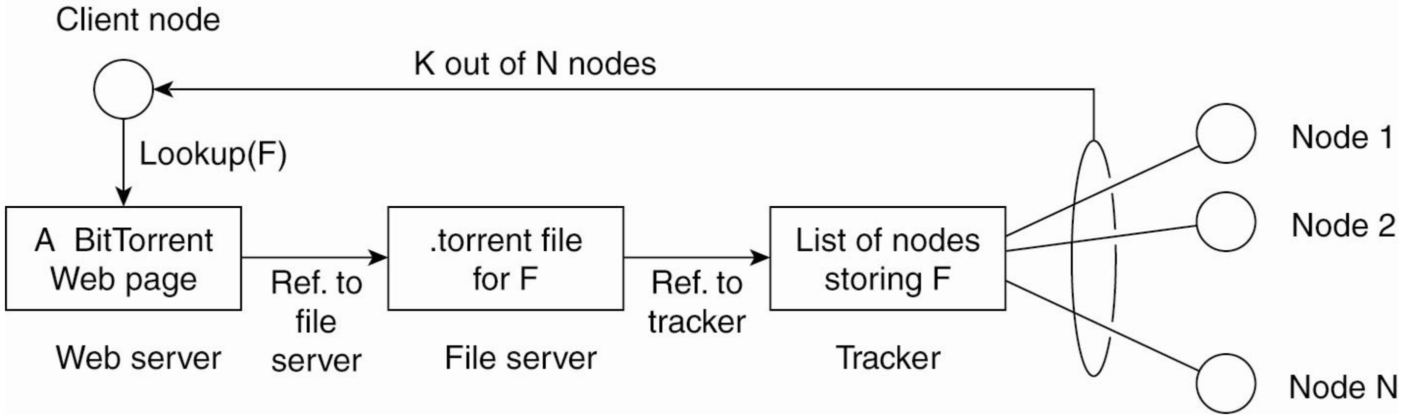
- A peer-to-peer (P2P) architecture can be characterized as follows:

4) Peers can interact directly, forming groups and sharing contents (or offering services to each other)
   - At least one peer should share the data, and this peer should be accessible
   - Popular data will be highly available (it will be shared by many)
   - Unpopular data might eventually disappear and become unavailable (as more users/peers stop sharing them)

5) Peers can form a virtual *overlay network* on top of a physical network topology
   - *Logical paths* do not usually match *physical paths* (i.e., higher latency)
   - Each peer plays a role in routing traffic through the overlay network

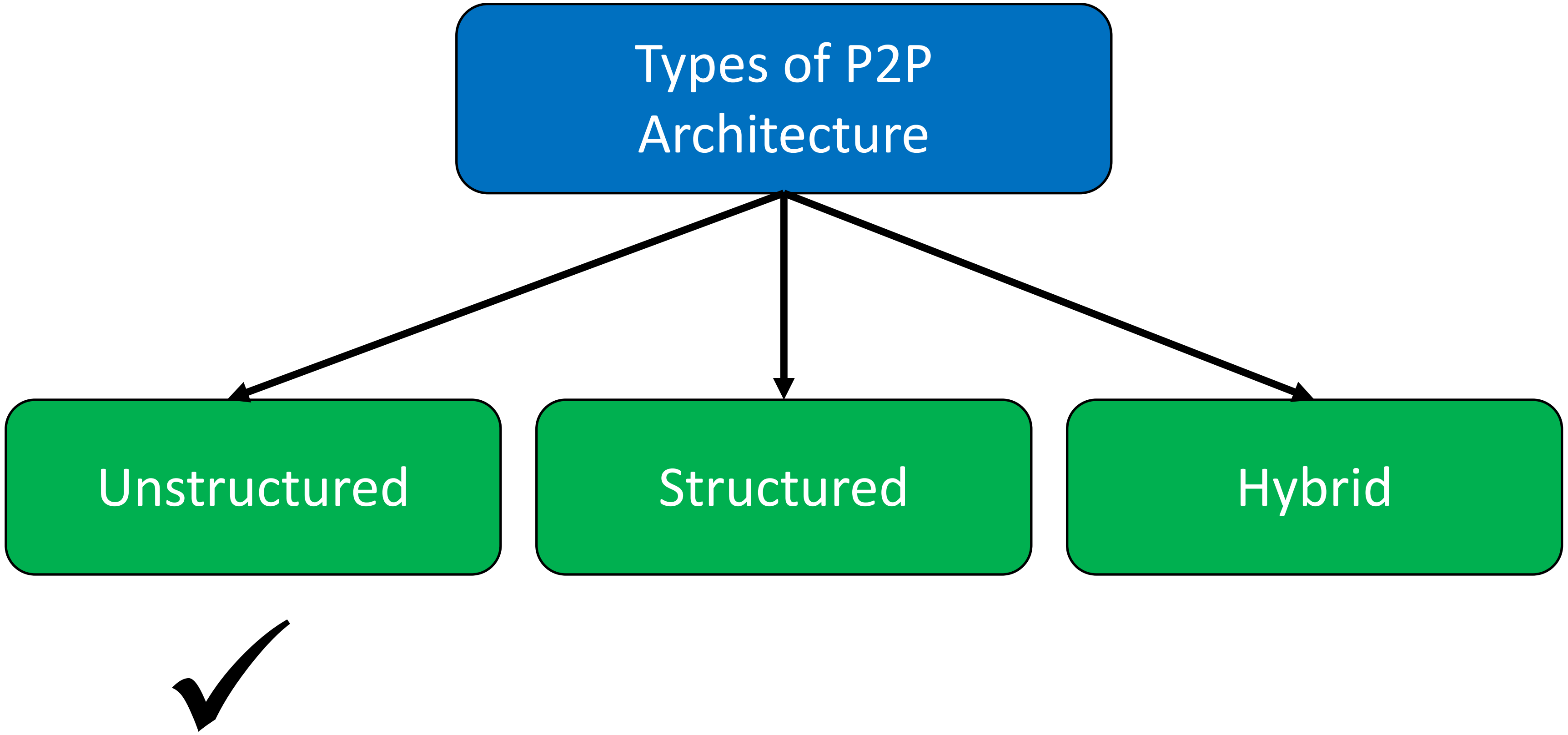
Peer-to-Peer Architecture

Collaborative Distributed Systems



- BitTorrent: Collaborative P2P downloads
  - Download chunks of a file from multiple peers
    - Reassemble file after downloading
  - Use a global directory (web-site) and download a .torrent
    - .torrent contains info about the file
      - Tracker: server that maintains active nodes that have requested chunks
      - Force altruism:
        - If *P* sees *Q* downloads more than uploads, reduce rate of sending to Q

P2P Types



Types of P2P Architecture

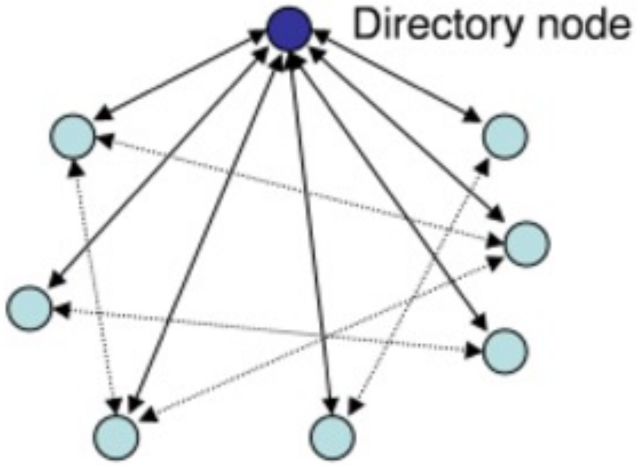
Unstructured

Structured

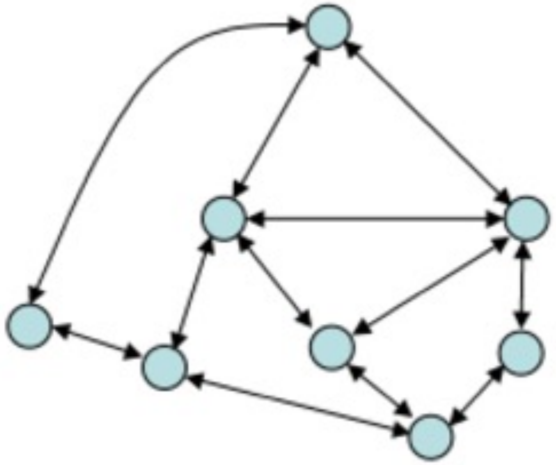
Hybrid

✓

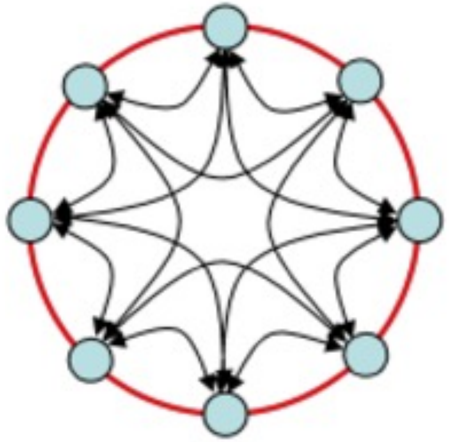
P2P Types



Hybrid (Napster)



Unstructured overlay



Structured overlay

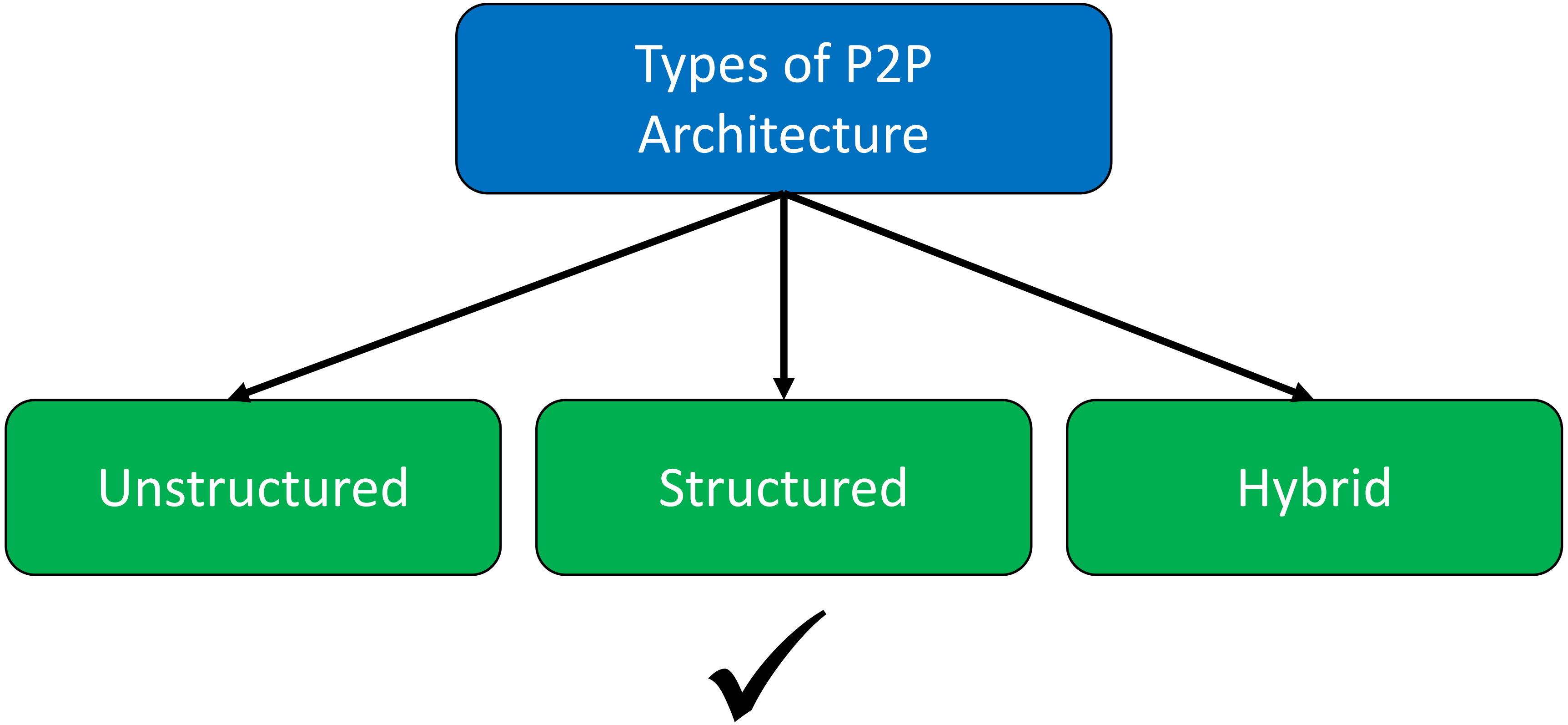
P2P Types

- Unstructured P2P:
  - The architecture does not impose any particular structure on the overlay network
    - Each node pick a random set of nodes and becomes their neighbors
    - Choice of degree impacts network dynamics

  - Advantages:
    - Easy to build
    - Highly robust against high rates of <u>*churn*</u> (i.e., when a great deal of peers frequently join and leave the network)

  - Main disadvantage:
    - Peers and contents are *loosely-coupled*, creating a data location problem
      - Searching for data might require broadcasting

P2P Types


Types of P2P Architecture
Unstructured
Structured
Hybrid
✓

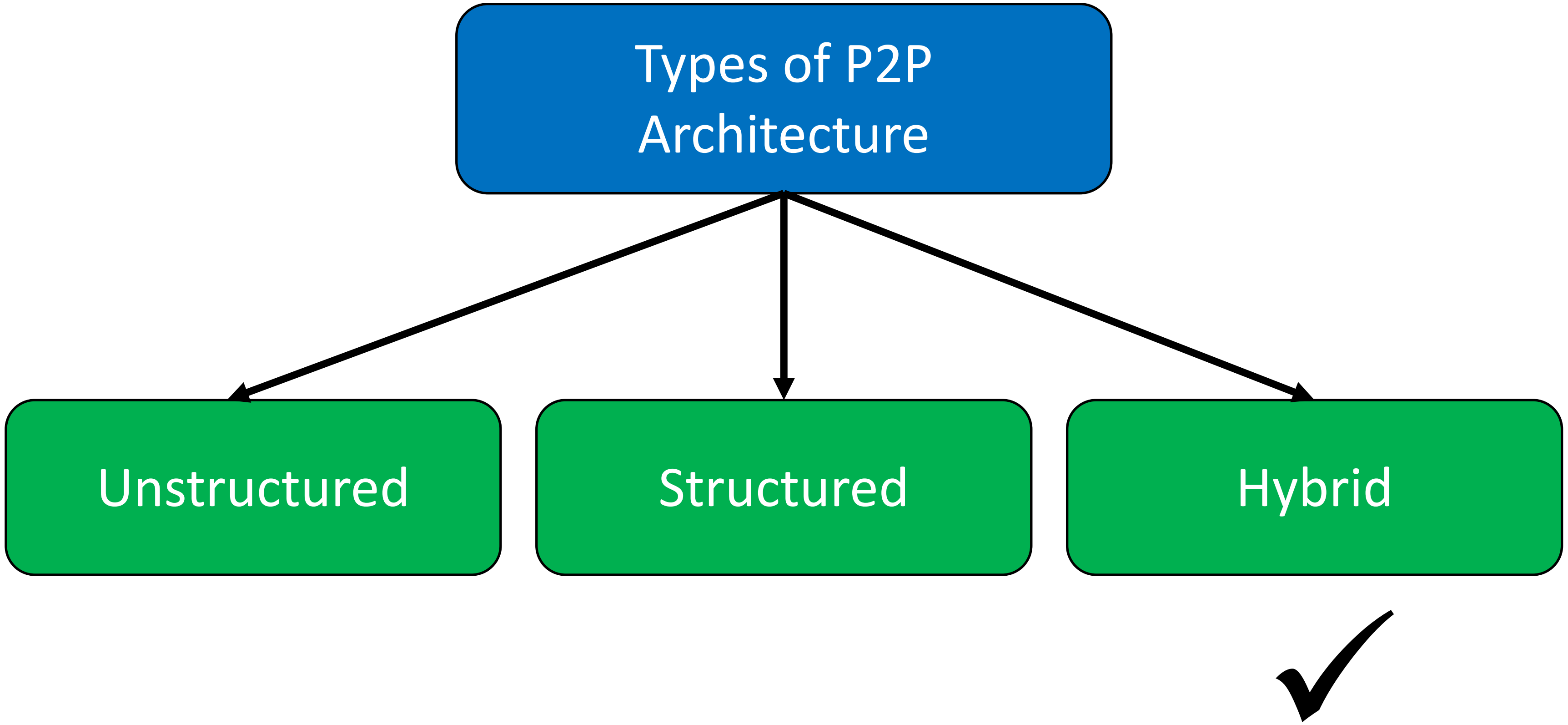
P2P Types

- Structured P2P:
  - The architecture imposes some structure on the overlay network topology

  - Main advantage:
    - Peers and contents are *tightly-coupled* (e.g., through hashing), simplifying data location

  - Disadvantages:
    - Harder to build
    - For optimized data location, peers must maintain extra metadata (e.g., lists of neighbors that satisfy specific criteria)
    - Less robust against high rates of churn

P2P Types



Types of P2P Architecture

Unstructured

Structured

Hybrid

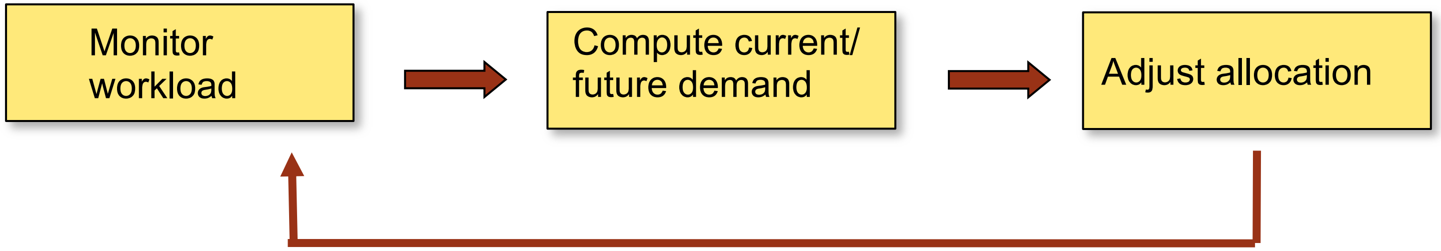
✓

P2P Types

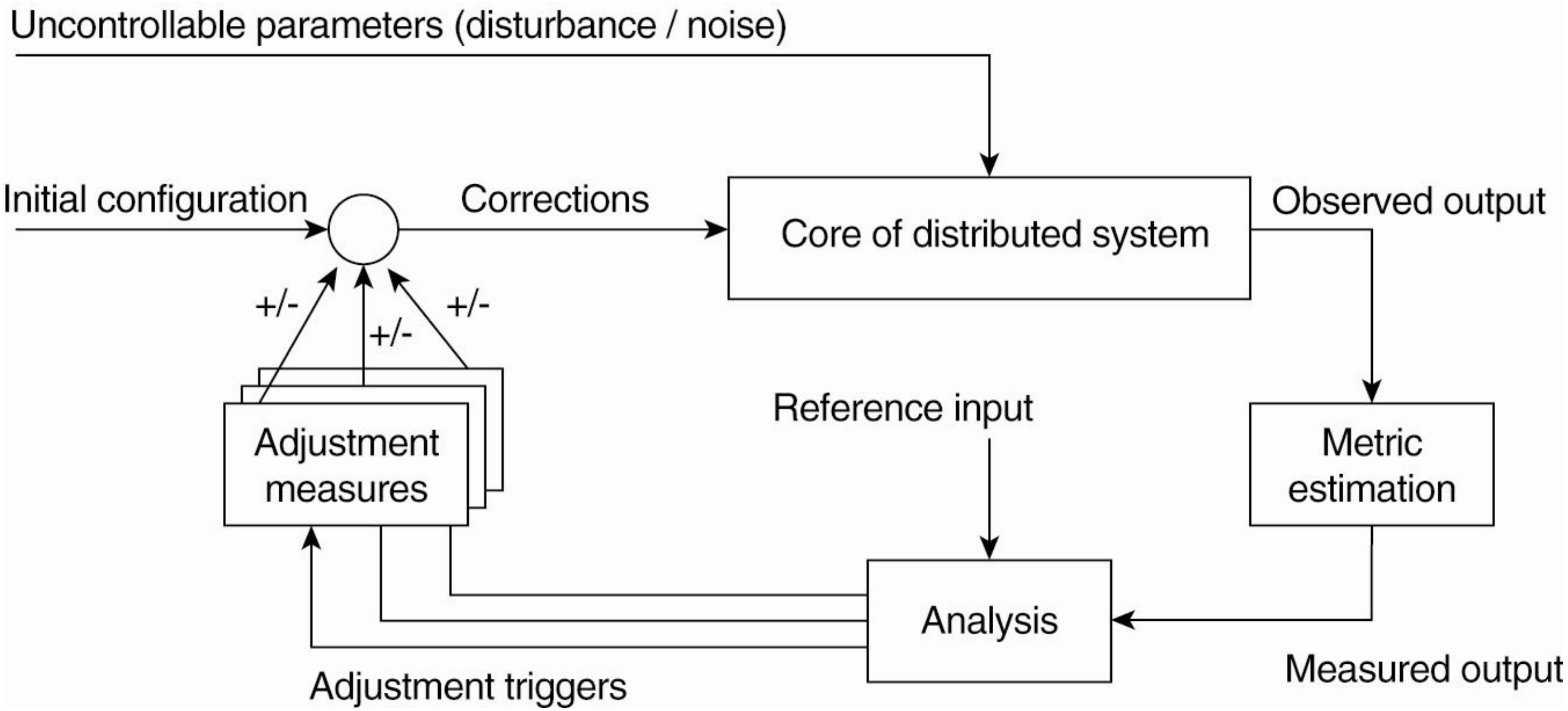
- Hybrid P2P:
  - The architecture can use *some* central servers to help peers locate each other
    - A combination of P2P and master-slave models
  - It offers a trade-off between the *centralized functionality* provided by the master-slave model and the *node equality* afforded by the *pure* P2P model
    - In other words, it combines the advantages of the master-slave and P2P models and precludes their disadvantages

Self-Managing Systems

- System is adaptive
  - Monitors itself and takes action autonomously when needed
    - Autonomic computing, self-managing systems
- Self-*: self-managing, self-healing
- Example: automatic capacity provisioning
  - Vary capacity of a web server based on demand



Feedback Control Model



- Use feedback and control theory to design a self-managing system

A To-Do List

- Read Chapters 2, Architectures
- Project teams