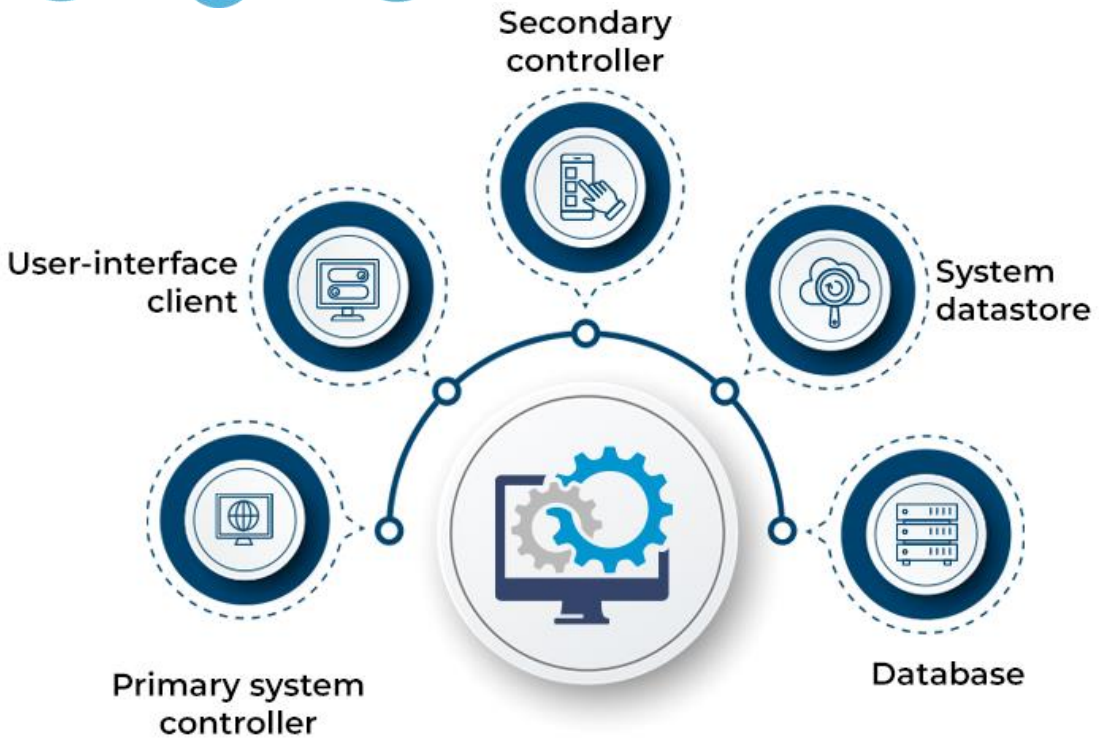
Sistem Terdistribusi

IF2222



03: Arsitektur

2022-23 Ganjil

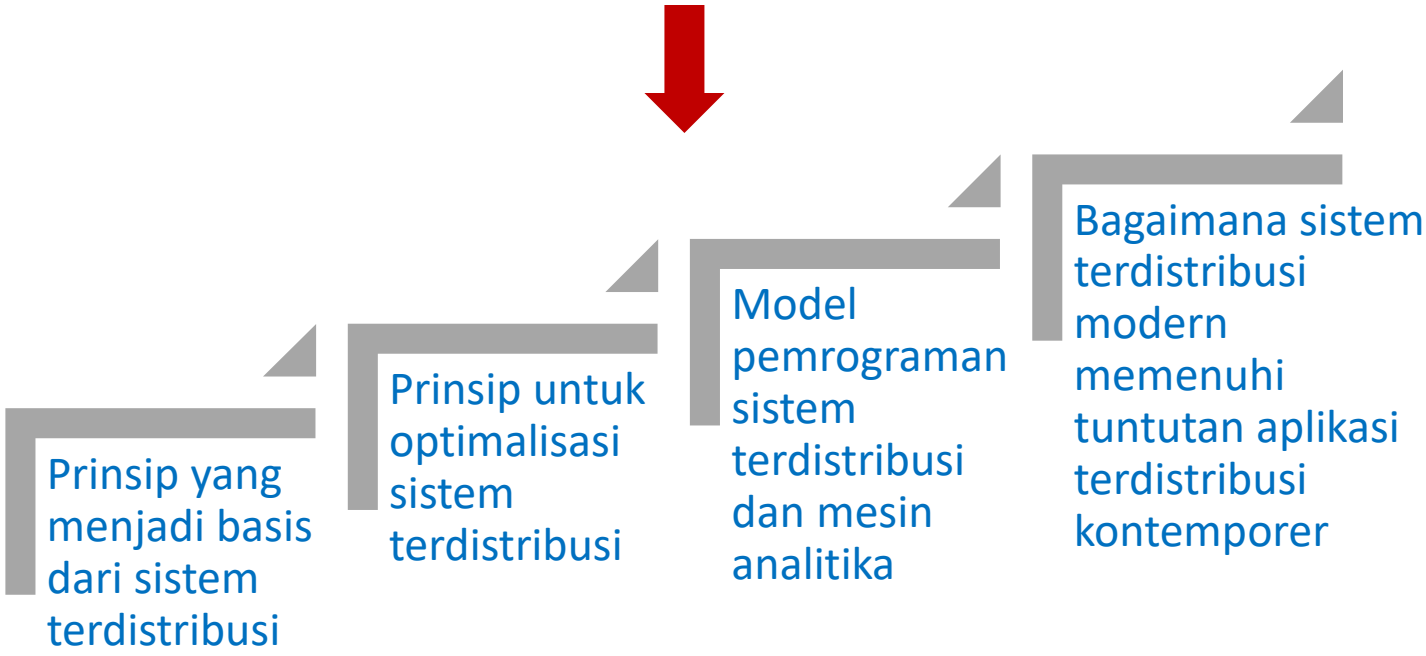
Teknik Informatika
Universitas Trunojoyo Madura

Sistem Terdistribusi 2022

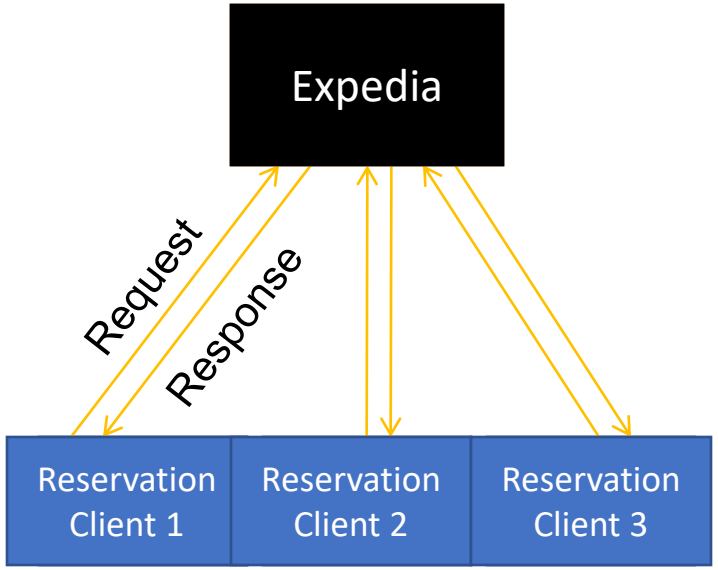
1. Mengenal Sistem Terdistribusi
2. Review Jaringan Komputer (layer 2, 3, dan 4)
3. **Arsitektur Sistem Terdistribusi**
4. *Remote Procedure Calls* (RPC)
5. Layanan Penamaan
6. Sinkronisasi Data (2 pekan)
7. *Message Passing Interface* (MPI)
8. Contoh Arsitektur: Hadoop, Pregel, Blockchain
9. Teknik *Caching*
10. Teknik Replikasi Data (2 pekan)
11. Basis Data Terdistribusi
12. Toleransi Kegagalan

Capaian Pembelajaran

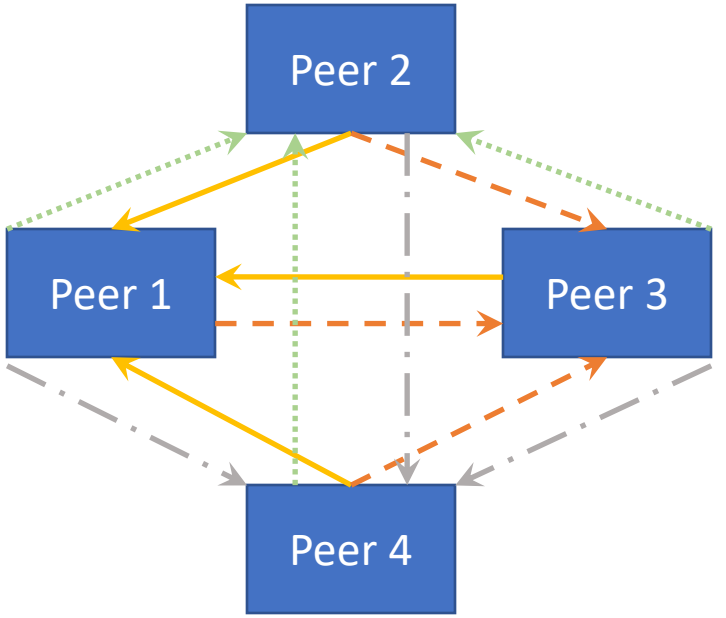
Kuliah ini bertujuan memberikan pemahaman mendalam dan pengalaman langsung tentang:



Bird's Eye View of Some Distributed Systems



Google Search
Airline Booking



Bit-torrent
BlockChain/BitCoin

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– we will see study more paradigms later)

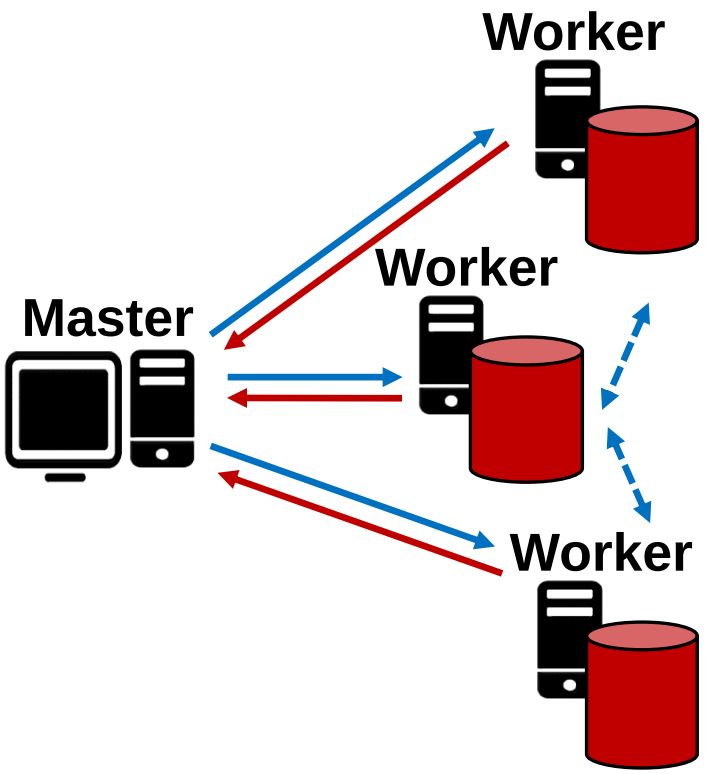
- What roles and responsibilities do the entities have?
  c) This could lead to different organizations (referred, henceforth, to as *architectures*)

Architectures

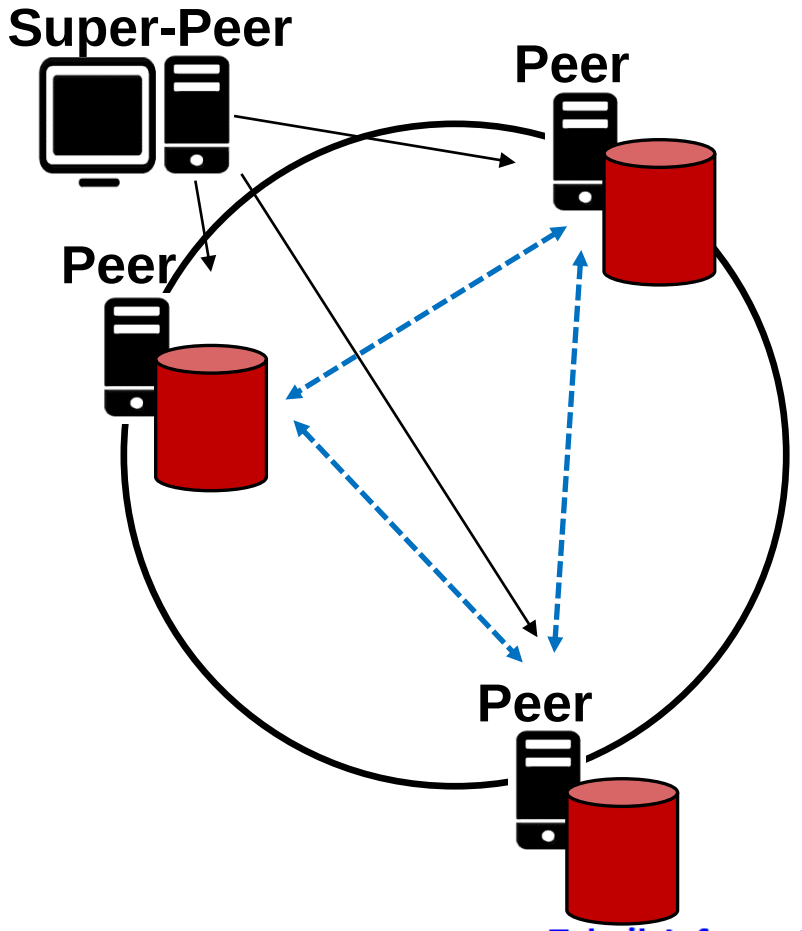
- Two main architectures:
  - Master-Slave architecture
    - Roles of entities are *asymmetric*
  - Peer-to-Peer architecture
    - Roles of entities are *symmetric*

Architectures

Master-Slave



Peer-to-Peer

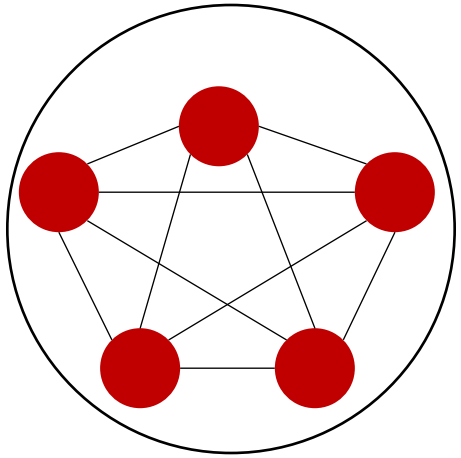


Master-Slave Architecture

- A master-slave architecture can be characterized as follows:
  1) Nodes are *unequal* (there is a hierarchy)
     - Vulnerable to *Single-Point-of-Failure* (SPOF)
  2) The master acts as a *central coordinator*
     - Decision making becomes easy
  3) The underlying system *cannot scale out* indefinitely
     - The master can render a *performance bottleneck* as the number of workers is increased

Peer-to-Peer Architecture

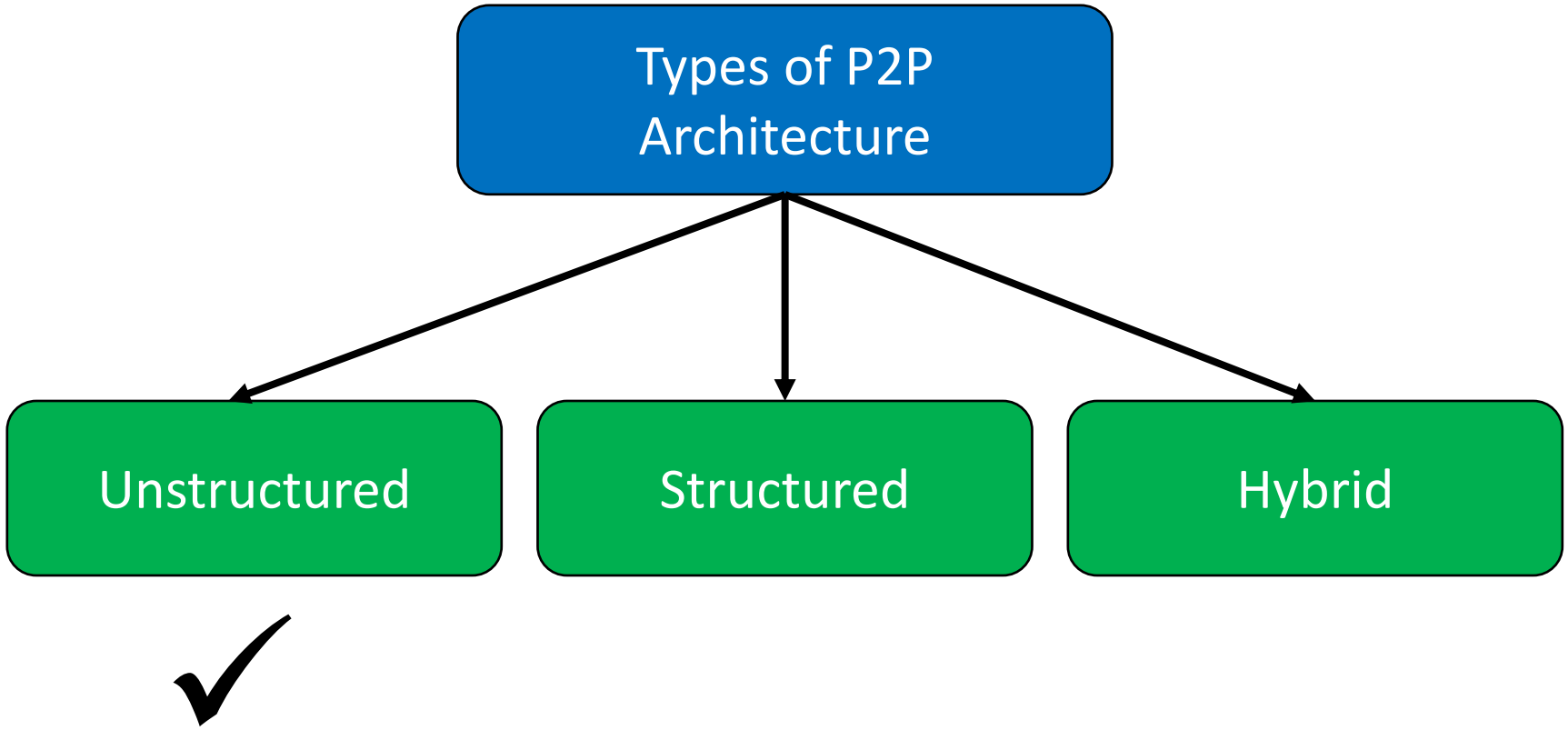
- A peer-to-peer (P2P) architecture can be characterized as follows:
  1) All nodes are equal (no hierarchy)
     - No Single-Point-of-Failure (SPOF)
  2) A central coordinator is not needed
     - But, decision making becomes harder
  3) The underlying system can scale out indefinitely
     - In principle,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



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

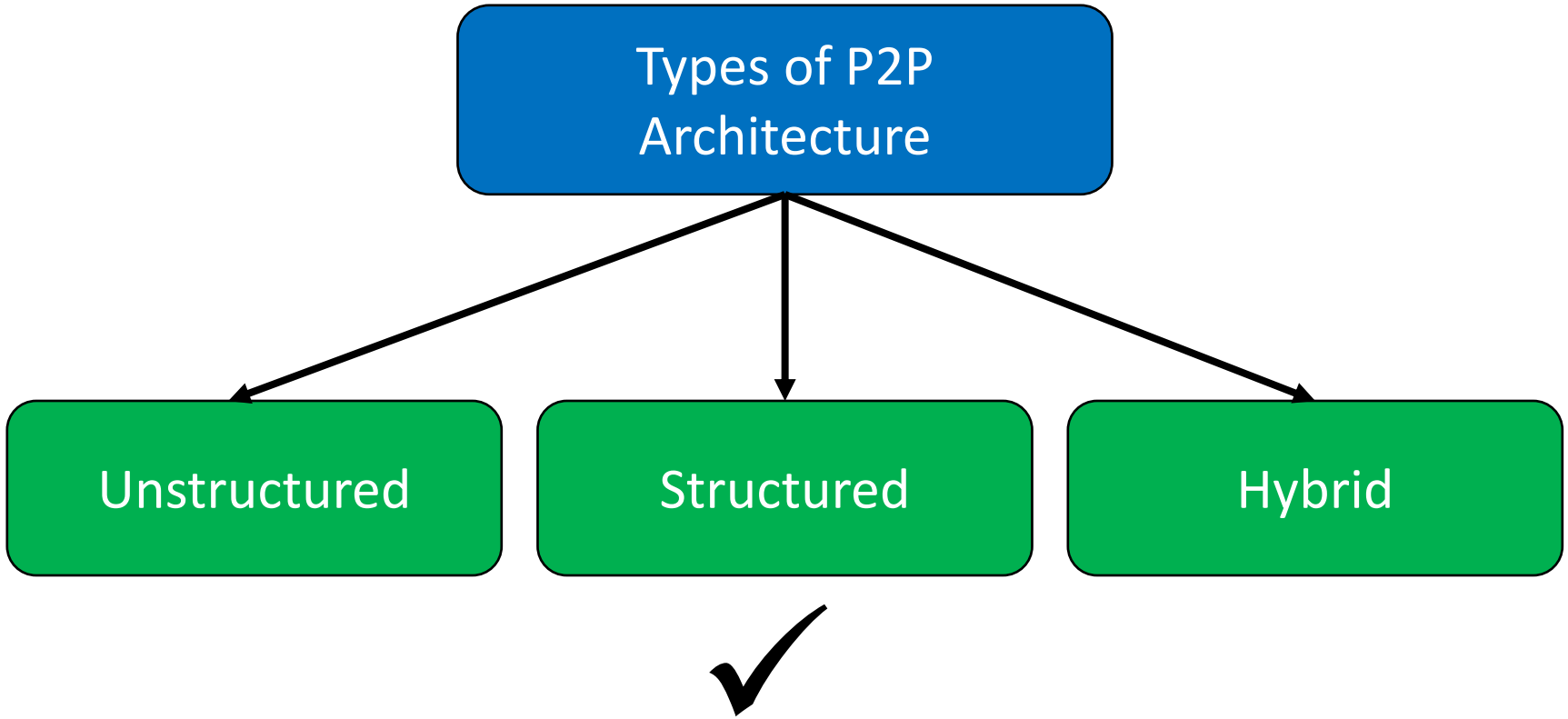
P2P Types


Types of P2P Architecture
Unstructured
Structured
Hybrid

P2P Types

- Unstructured P2P:
  - The architecture does not impose any particular structure on the overlay network
  - Advantages:
    - Easy to build
    - Highly robust against high rates of *churn* (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

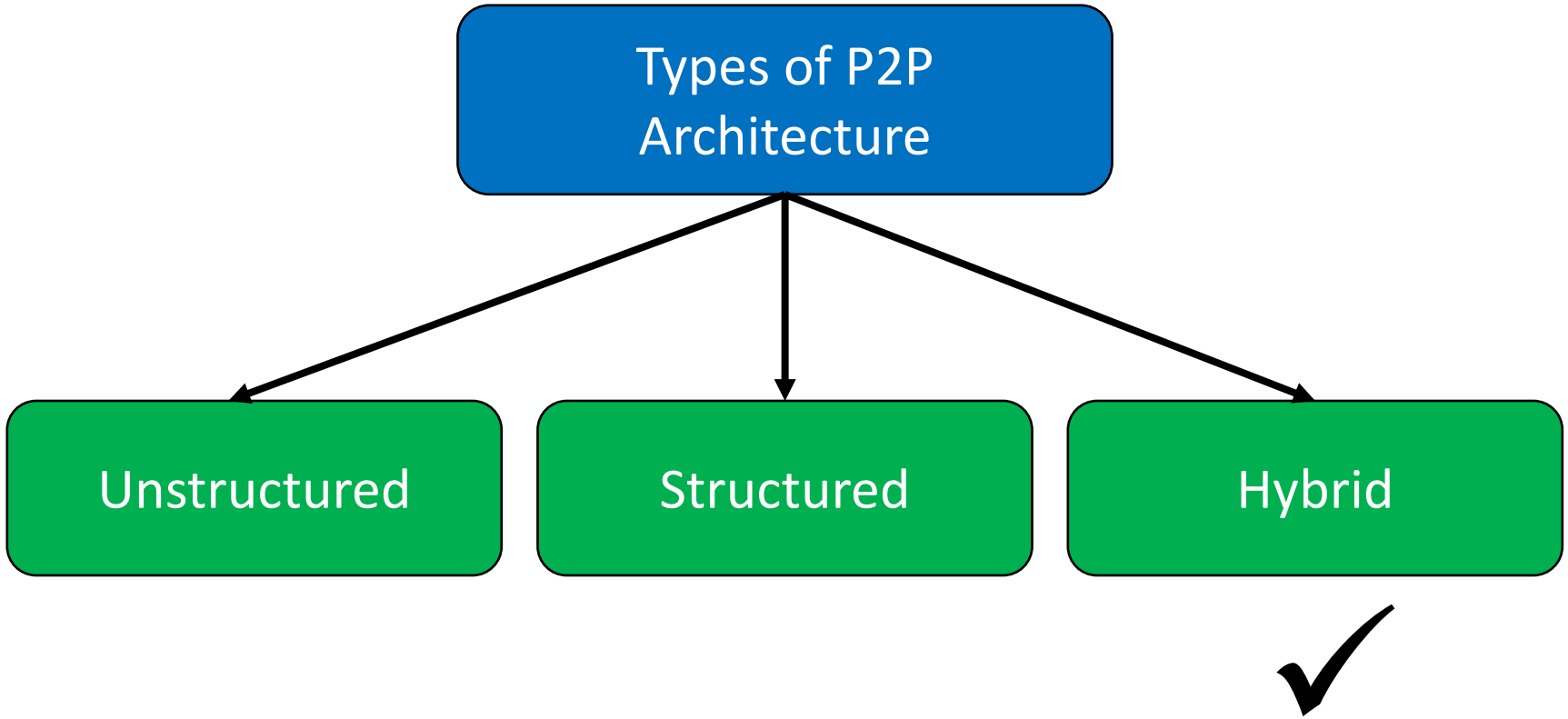
P2P Types


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

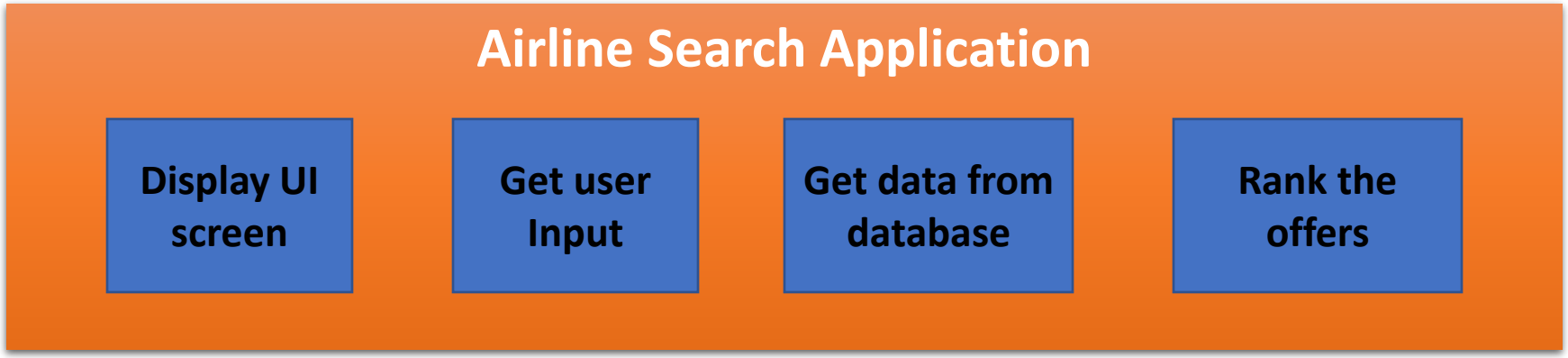
Architectural Patterns

- Aside from *architectures*, primitive architectural elements can be combined to form various *patterns* via:
  - Tiering
  - Layering

- Tiering and layering are complementary
  - Tiering = horizontal splitting of services
  - Layering = vertical organization of services

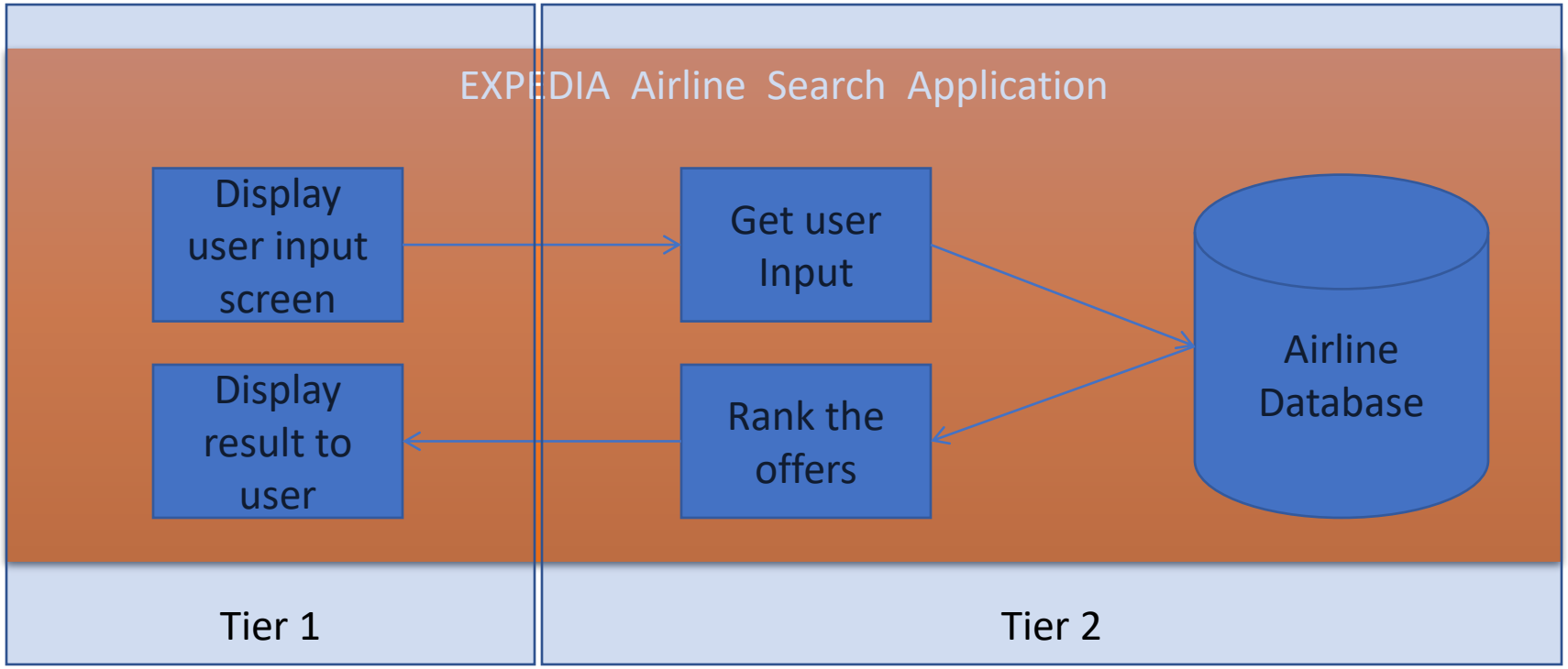
Tiering

- Tiering is a technique to:
  1. Organize the functionality of a service,
  2. and place the functionality into appropriate servers



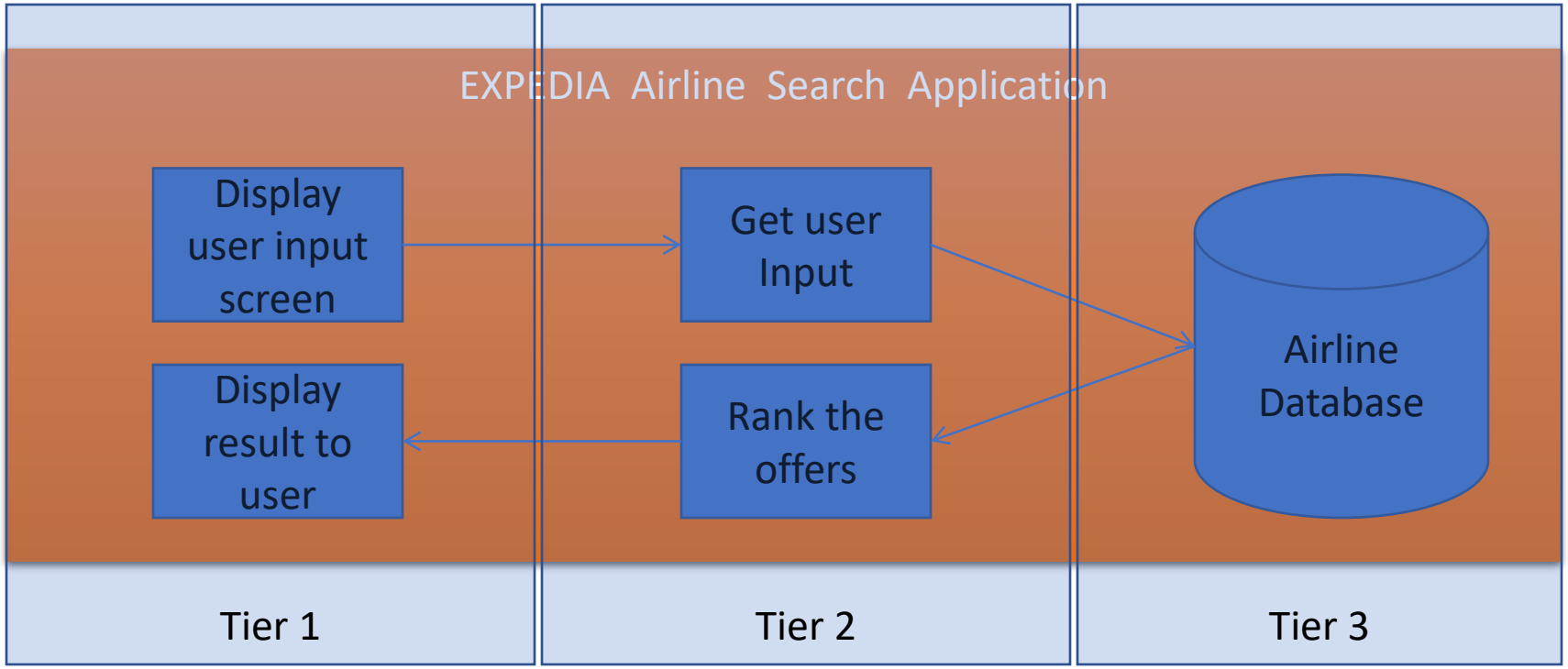
A Two-Tiered Architecture

- How would you design an airline search application?

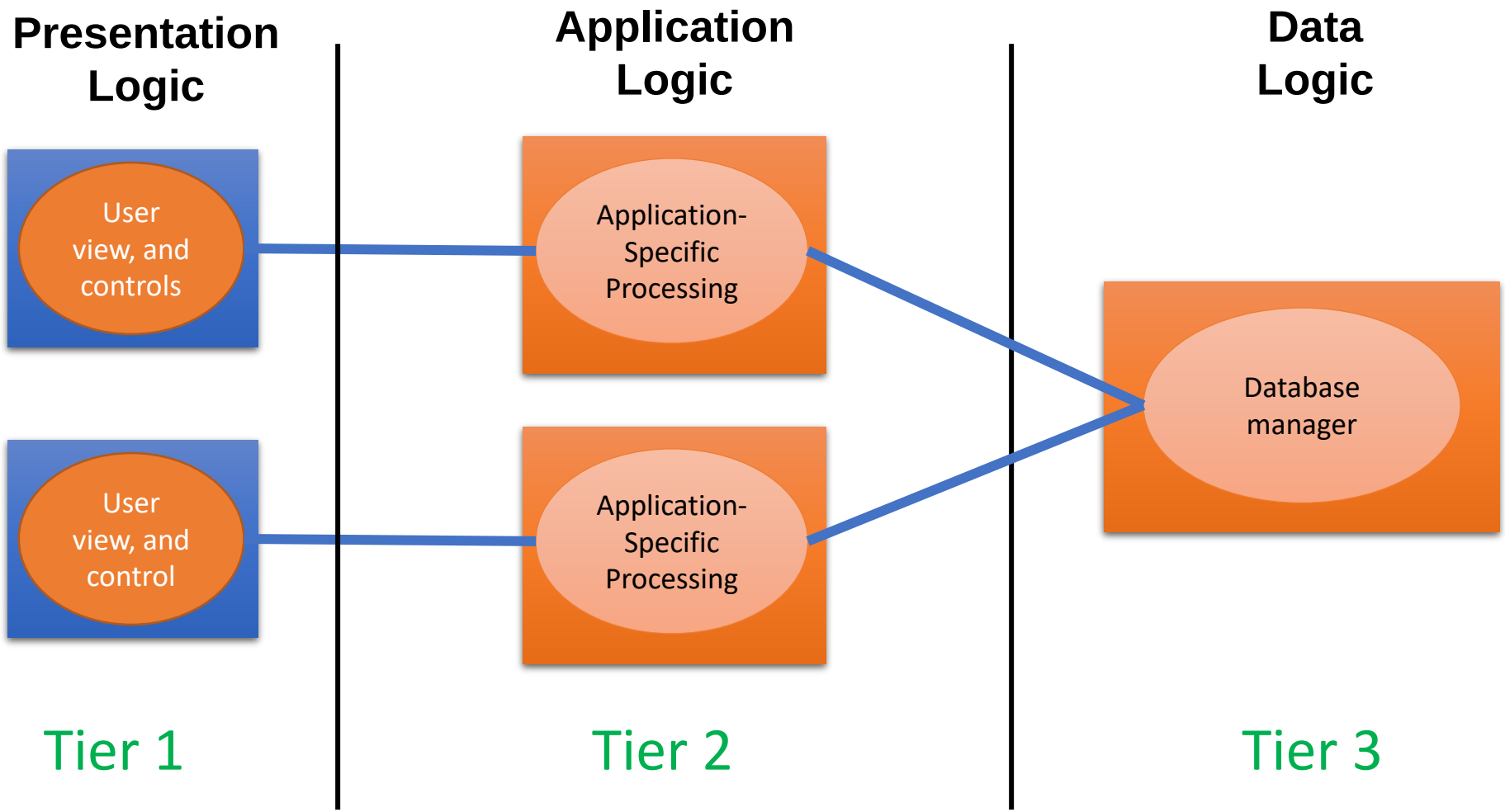


A Three-Tiered Architecture

- How would you design an airline search application?



A Three-Tiered Architecture



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

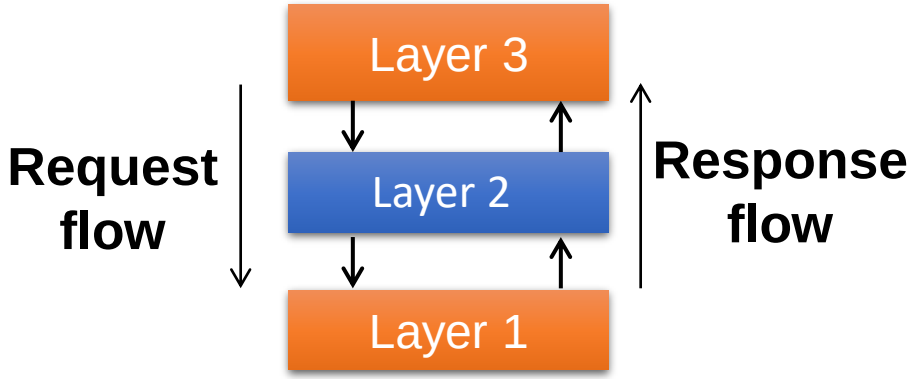
- Disadvantages:
  - Added complexity due to managing multiple servers
  - Added network traffic
  - Added latency

Layering

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

- 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:

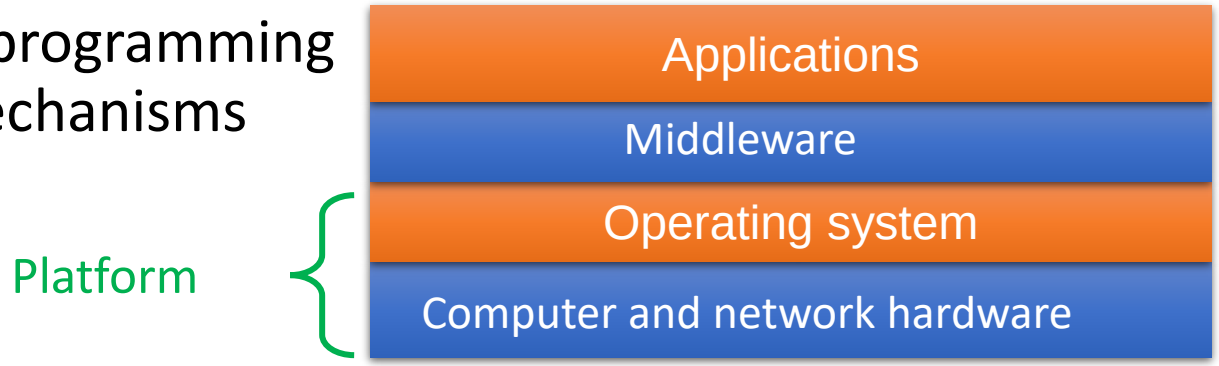
1.Platform

- 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



Kuliah Berikutnya

- *Remote Procedure Calls* (RPC)

Pertanyaan?