LECTURE 2

Tiered Web Architecture

How to organize an application's code and computers — from one machine to many.

CSC336 Web Technologies

OVERVIEW

In This Lecture

1. Distinguish between tiers (physical deployment) and layers (logical code organization)
2. Learn the common layers inside a typical application, and how they combine with tiers
3. Understand one-tier (standalone) architecture and when it's used
4. Understand two-tier (client-server) architecture, its advantages and limitations
5. Understand three-tier and N-tier architecture
6. Learn how to choose an architecture based on scalability, maintainability, and cost

Tiers vs. Layers: Two Different Ideas

Both aim at separation of concerns — but tiers separate physically, layers separate logically.

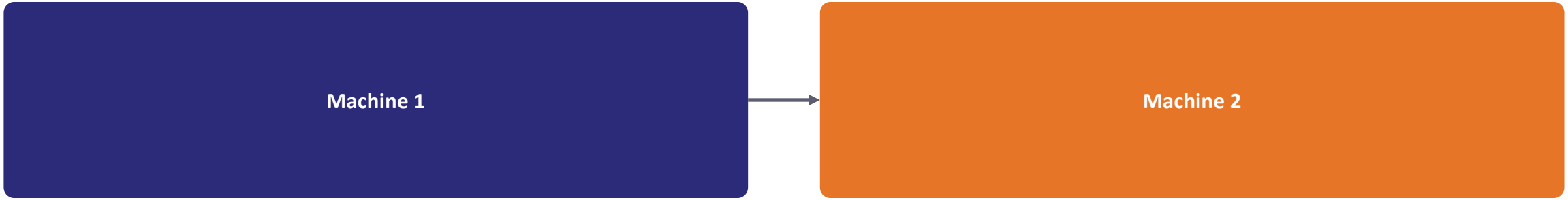
TIER

- A physical (deployment) boundary
- Describes WHERE a piece of the app runs — which machine, process, or container
- Two pieces running on different computers are in different tiers

LAYER

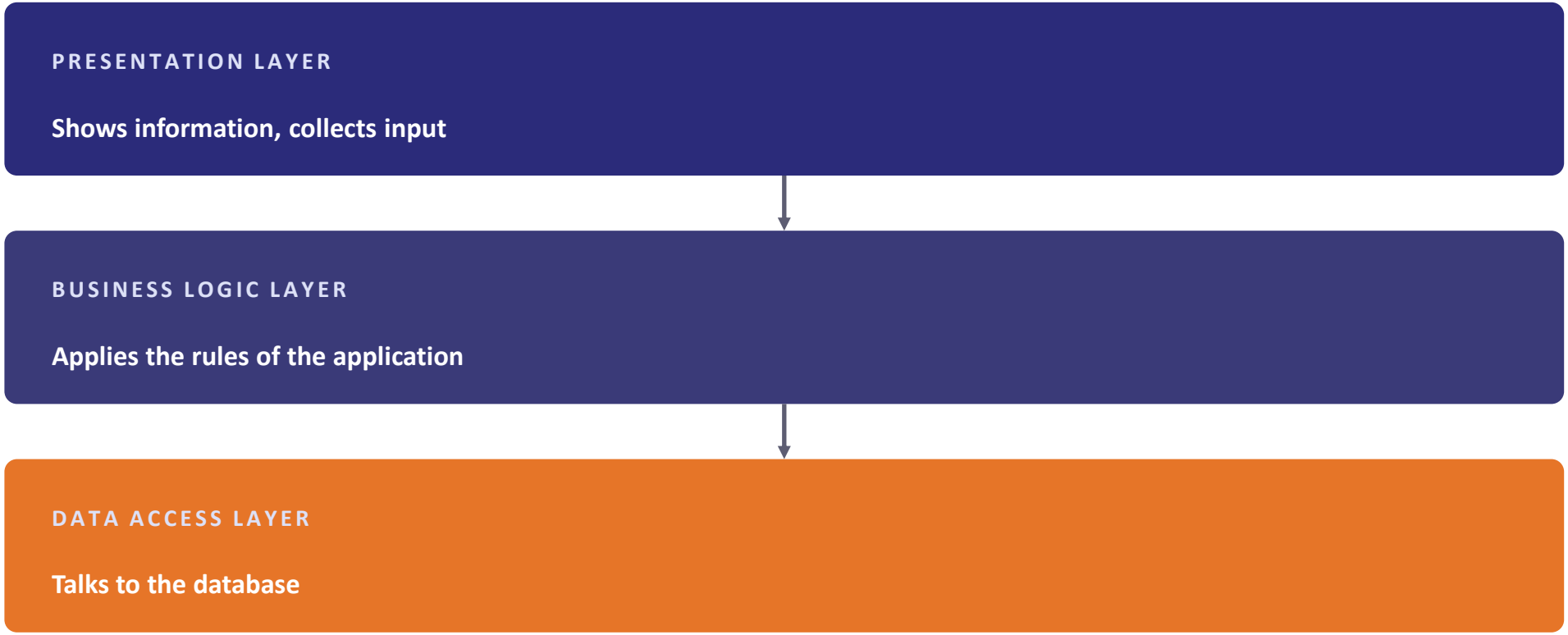
- A logical boundary inside your code
- Describes HOW responsibilities are organized within a program
- Layers can all live on the very same machine, in the very same process

Tiers = Physical Deployment



Two pieces of an app communicating over a network are in two different tiers, no matter how similar their code looks.

Layers = Logical Code Organization



A Layered App Can Still Be One Tier

NOTE

You could write a program with clean layers — a data layer, a business-logic layer, a presentation layer — and still run the entire thing on a single laptop as one process. That is one tier (everything deployed together) but multiple layers (well-organized code). Tiers and layers are independent design decisions.

SIMPLEST CASE

One-Tier (Standalone) Architecture

- The entire application — UI, business logic, and data storage — runs on a single machine, in a single process
- No network communication between tiers, because there is only one tier
- Example: a desktop calculator, a local script, or a static HTML file opened via file:///

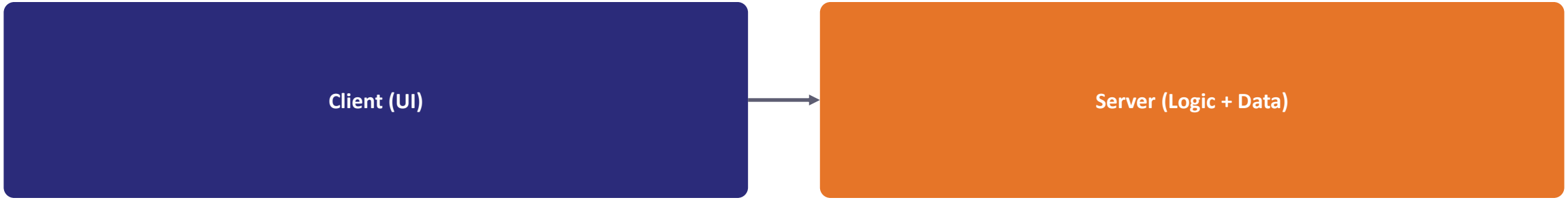
SIMPLEST CASE

One-Tier, Visualized

```
[ Your Computer ]
   ├── User Interface
   ├── Application Logic
   └── Data (local file / in-memory)
```

One-tier applications are the easiest to build and deploy — no network, no server to configure. But they cannot be shared over the Internet as-is; they only work for a single user on a single machine.

Two-Tier (Client–Server) Architecture



The client and server communicate over a network — the same client-server model from Lecture 1, now viewed as an architecture pattern.

TRADE-OFFS

Two-Tier: Advantages and Limitations

ADVANTAGES

- Simple to build and reason about — only two moving parts
- Centralized data — all clients see the same up-to-date data
- Lower initial cost — fewer servers to set up and pay for

LIMITATIONS

- Scalability bottleneck — one server handles UI, logic, and data all at once
- Tight coupling between business logic and data access
- Single point of failure — if the server goes down, so does the app

Three-Tier Architecture

This is exactly the shape of the MERN stack you'll build later: React (presentation), Express/Node.js (application), MongoDB (data) — each capable of running on its own machine.

N-Tier: As Many Tiers as You Need

N-tier generalizes three-tier: split into any number of tiers when it's useful.

- A caching tier (e.g., Redis) to speed up repeated requests
- A load balancer tier in front of multiple application servers
- A separate authentication tier dedicated to verifying user identity

TRADE-OFFS

Three-Tier / N-Tier: Advantages and Limitations

ADVANTAGES

- Independent scaling — add app servers without touching the database
- Each tier can be developed and deployed by different teams
- Reusability — one API tier can serve web, mobile, and third parties
- Better fault isolation

LIMITATIONS

- More complexity — more configuration, more network calls
- Higher cost — more servers or cloud services to pay for
- Network latency between tiers, slower than in-process calls

The Three Common Layers

1. Presentation Layer — shows information and captures input; no business rules here
2. Business Logic Layer — the actual rules: validation, calculations, decisions
3. Data Access Layer — talks to the database; knows nothing about business rules or UI

A Request Flowing Through the Layers

```
project/
├── routes/       <- Presentation: reads the request, sends response
├── services/     <- Business logic: validation, calculations
├── models/       <- Data access: talks to the database
└── app.js
```

A request flows down through the layers, and the response flows back up — each layer only ever talks to its direct neighbor, never skipping ahead. This is exactly the structure you'll build starting in Unit 5.

DECISION TIME

Choosing an Architecture

Factor	Question to Ask
Scalability	Will the number of users grow? Do parts need to grow at different rates?
Maintainability	Will multiple teams work on this? Update one part without breaking another?
Cost	What's the hosting budget? Can you afford the added operational complexity?

One-tier for personal tools and prototypes. Two-tier for small, stable apps. Three-tier/N-tier for apps expected to grow or be maintained by a team long-term.

A WORD OF CAUTION

More Tiers Is Not Automatically Better

WARNING

A three-tier setup for a small class project adds deployment overhead (multiple servers, network configuration) that isn't justified if the project will never need to scale. Match the architecture to the actual requirements, not to what sounds most impressive.

KEY TAKEAWAYS

Lecture 2 in Six Points

1. Tiers are physical/deployment boundaries; layers are logical boundaries in code — both aim at separation of concerns, independently.
2. One-tier runs everything on one machine; two-tier splits client from server but has a single point of failure.
3. Three-tier separates presentation, application, and data — enabling independent scaling at the cost of complexity.
4. N-tier generalizes further, adding tiers like caching or load balancing as needed.
5. Most apps organize code into three layers: presentation, business logic, and data access — each talks only to its neighbor.
6. Choosing an architecture is a trade-off between scalability, maintainability, and cost.