Modular Monoliths Architecture

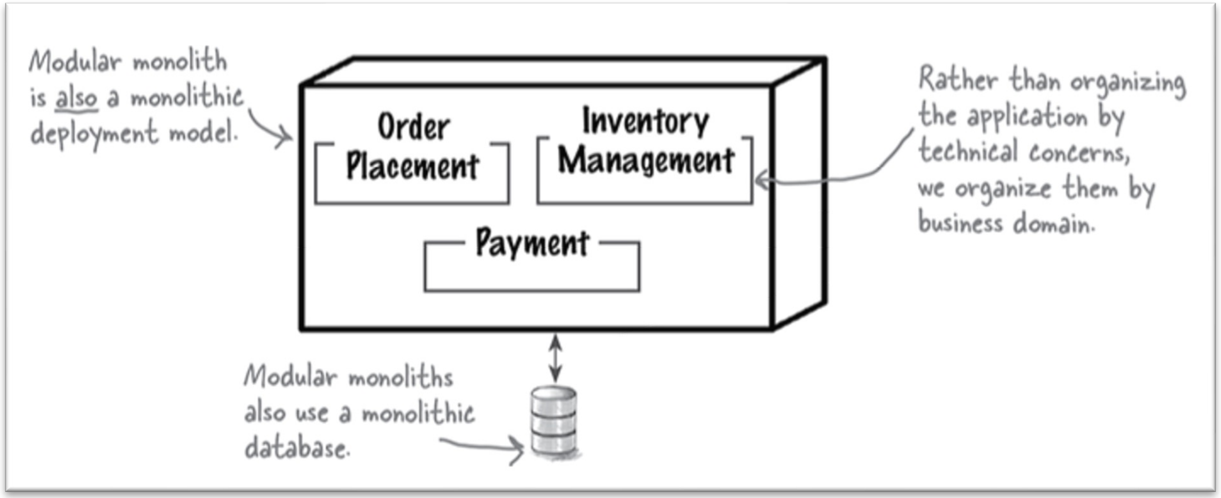
COMP2511, CSE, UNSW

These lecture slides are from the books:

- “*Head First Software Architecture*”, by Raju Gandhi, Mark Richards, Neal Ford, O'Reilly Media, Inc., March 2024
- “*Fundamentals of Software Architecture*”, 2nd Edition, by Mark Richards, Neal Ford

Introduction to Modular Monoliths

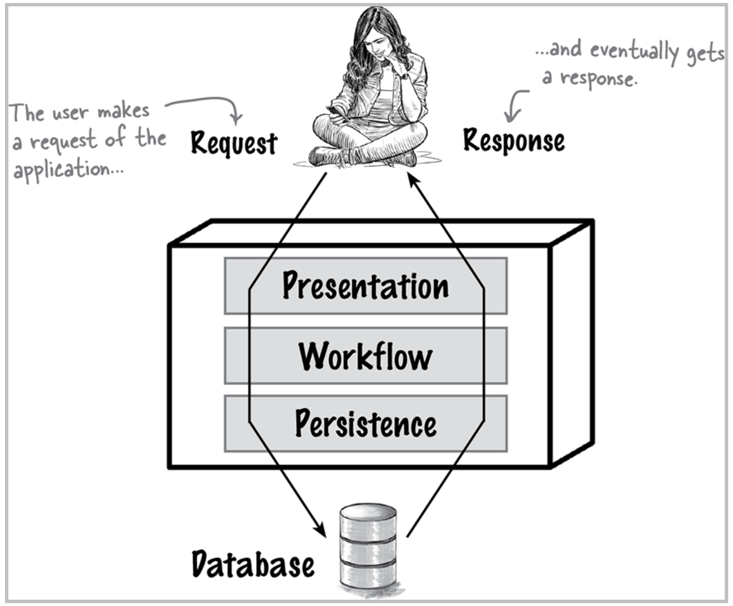
- Definition: A monolithic architecture organized by domain, not technical layers.
- Goal: Align code and teams around business capabilities.
- Key Trait: Deployed as a single unit, with domain-based modular structure

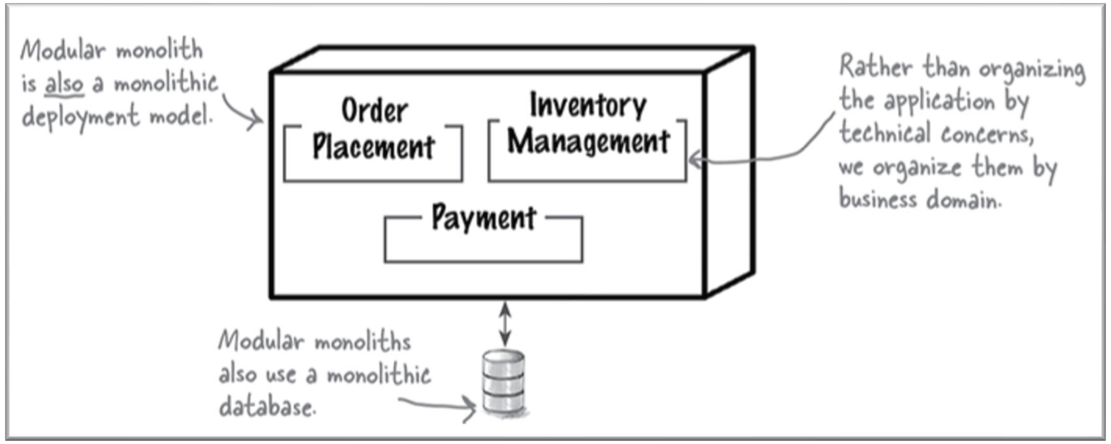

Modular monolith is also a monolithic deployment model.
Order Placement
Inventory Management
Payment
Rather than organizing the application by technical concerns, we organize them by business domain.
Modular monoliths also use a monolithic database.

Layered vs. Modular Monolith

- Layered: Organized by technical concerns (UI, services, DB).
- Modular: Organized by domain (Order, Payment, Inventory).

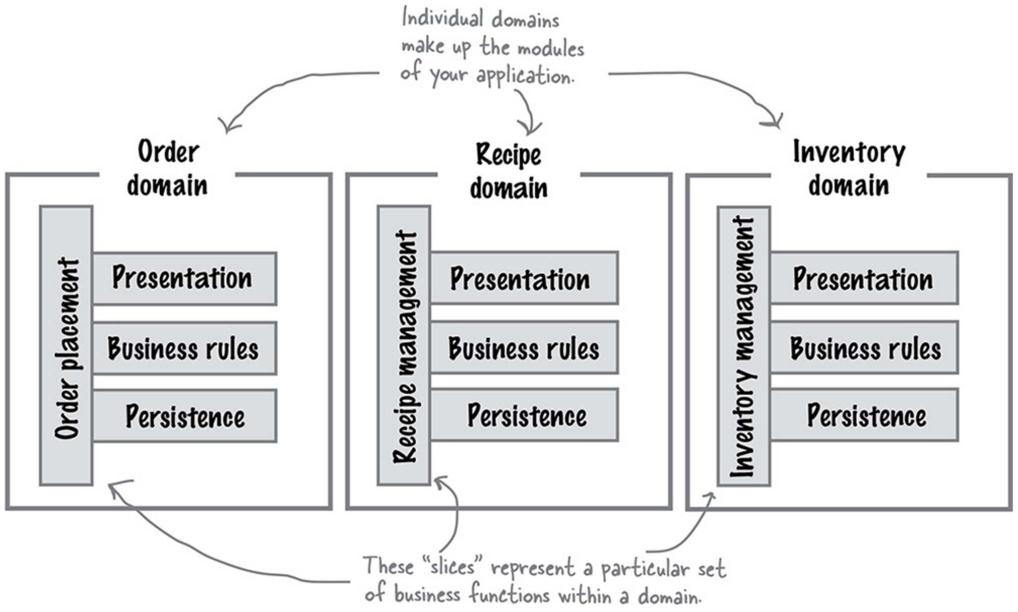
- Problem with Layered: Changes often touch many teams.
- Benefit of Modular: Changes are isolated within a domain.


The user makes a request of the application...
Request
...and eventually gets a response.
Response
Presentation
Workflow
Persistence
Database


Modular monolith is also a monolithic deployment model.
Order Placement
Inventory Management
Payment
Rather than organizing the application by technical concerns, we organize them by business domain.
Modular monoliths also use a monolithic database.

What Is a Module?

- Independent unit within a domain.
- Contains all business logic for its domain.
- Examples:
  - *OrderPlacement* module handles order lifecycle
  - *Recipe* module contains ingredients and cooking steps
  - *Inventory* module tracks stock levels and alerts
  - *UserManagement* module handles user accounts and roles



Why Choose a Modular Monolith?

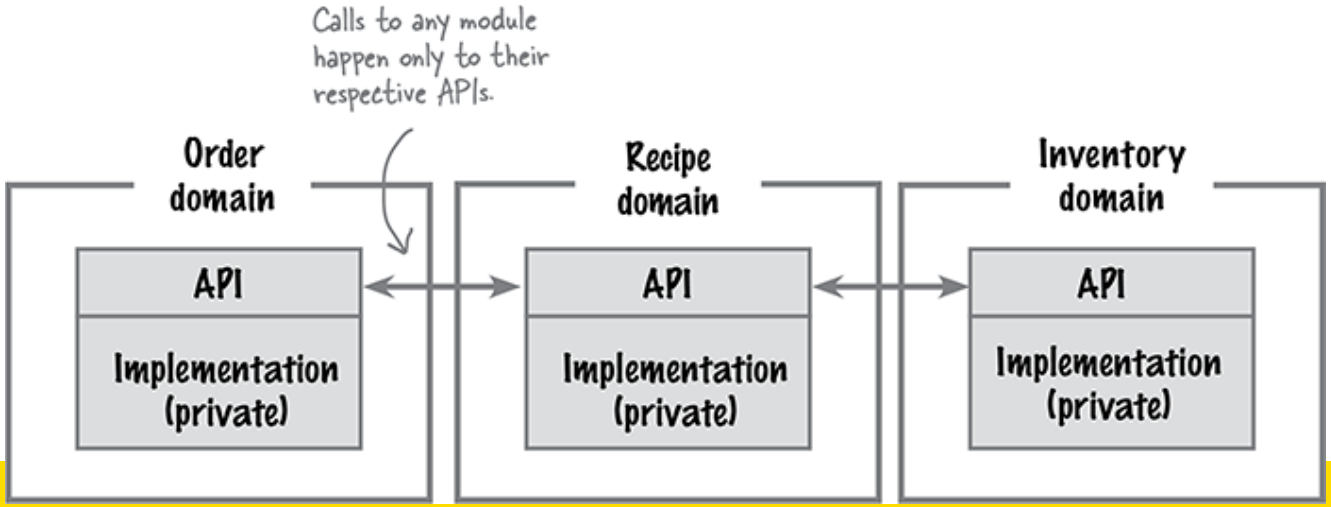
- Business alignment: Modules map to subdomains
- Team ownership: Cross-functional teams per domain
- Faster changes: Changes isolated to one module
- High performance: No inter-service network latency
- Easier testing: Scoped test suites per module

Code Organization in a Modular Monolith

- Single deployment
- Separate namespaces/packages for each module
- Each module has:
  - Public API
  - Private internals
- Example (namespace):
  - com.naanpop.order
  - com.naanpop.inventory
  - com.naanpop.reports

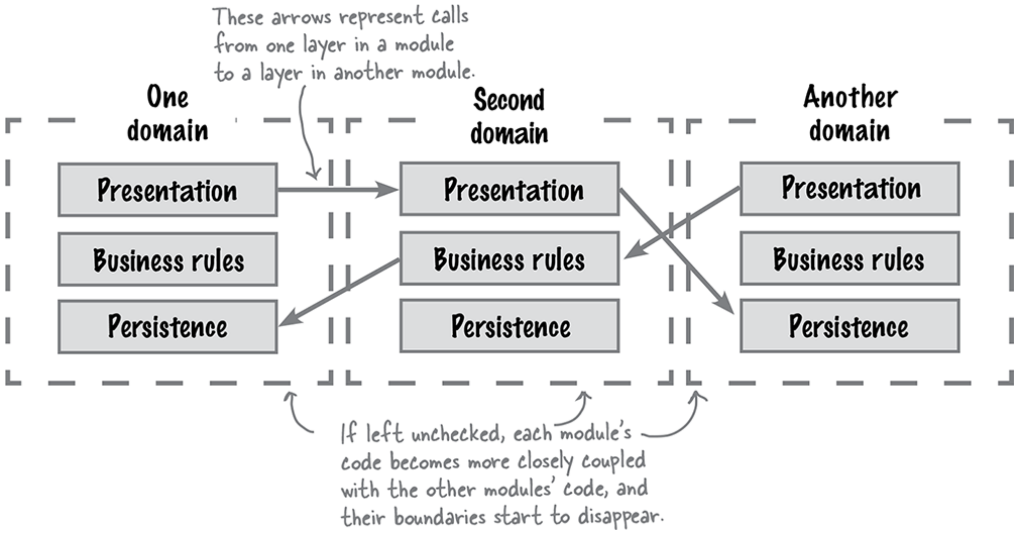
Managing Inter-Module Communication

- Don't: Direct calls between modules (tight coupling)
- Do: Use public APIs
- Risk: Big ball of mud from uncontrolled access
- Solution: Interface-based interaction only



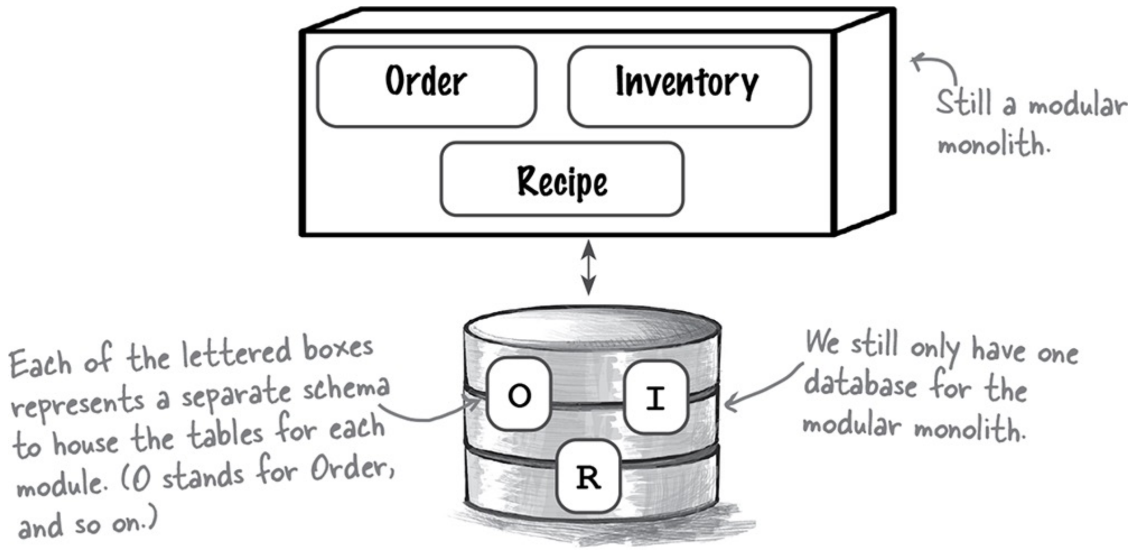
Keeping Modules Modular

- IDE features (e.g. auto-import) can break boundaries
- Separate folders/repositories
- Use build tools (e.g., Gradle subprojects)
- Use language features:
  - Java: JPMS
  - .NET: internal keyword


These arrows represent calls from one layer in a module to a layer in another module.
One domain
Second domain
Another domain
Presentation
Business rules
Persistence
Presentation
Business rules
Persistence
Presentation
Business rules
Persistence
If left unchecked, each module's code becomes more closely coupled with the other modules' code, and their boundaries start to disappear.

Modularizing the Database

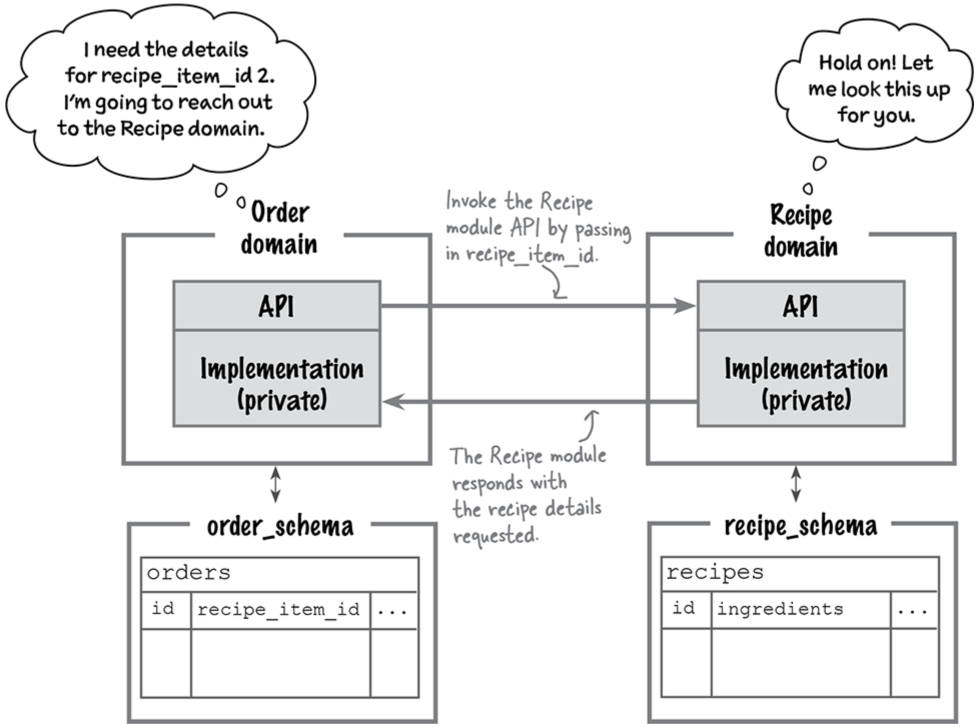
- One DB per monolith, but partitioned by schema
- **Rule**: *Each module accesses only its own tables*
- No foreign keys between modules
- Use ID references and API calls



Order
Inventory
Recipe
Still a modular monolith.
O
I
R
Each of the lettered boxes represents a separate schema to house the tables for each module. (O stands for Order, and so on.)
We still only have one database for the modular monolith.

Avoiding Coupling in Data Access

- Risk: JOINs across module tables reintroduce coupling
- Solution:
  - Store IDs, not foreign keys
  - Retrieve info via module API
- Example:
  - Order module stores *RecipeItemID*
  - Calls Recipe API when needed



I need the details for recipe_item_id 2. I'm going to reach out to the Recipe domain.
Hold on! Let me look this up for you.
Order domain
Recipe domain
Invoke the Recipe module API by passing in recipe_item_id.
API
Implementation (private)
API
Implementation (private)
The Recipe module responds with the recipe details requested.
order_schema
orders
id
recipe_item_id
...
recipe_schema
recipes
id
ingredients
...

Extending Modularity to Teams

- Align teams with subdomains (modular ownership)
- Foster domain expertise and autonomy
- Minimize coordination overhead
- Example: Inventory team owns inventory module and tests

Example – Expense Tracking App

- Requirements:
  - Users add expenses
  - Auditors review reports
  - Audit trail for traceability

- Modules:
  - ExpenseEntry
  - AuditReview
  - UserManagement

Example – Educational LMS

- Requirements:
  - Instructors upload courses
  - Students enroll and complete assessments
  - Admins manage roles and reports

- Modules:
  - CourseContent
  - Enrollment
  - AssessmentEngine
  - UserAdministration

Benefits of Modular Monoliths

- Domain Partitioning: Better team alignment
- Performance: No inter-service latency
- Maintainability: Domain-local changes
- Testability: Scoped, isolated testing
- Deployability: Single unit, easier CI/CD

Limitations of Modular Monoliths

- Reuse: Harder to share utilities
- One set of characteristics: No per-module customization
- Fragile modularity: Easy to break boundaries
- Operational limits: Harder to scale or isolate faults

Governance and Discipline

- Modular monoliths require:
  - Discipline in access control
  - Codebase enforcement (tools, practices)
  - Database discipline (modular schemas)

- Governance tools help but don't eliminate the need for vigilance

When to Use Modular Monoliths

- Teams aligned to business domains
- Applications that must remain performant
- Systems needing easy testability and deployment

Transition Path – Layered to Modular

- Start with layered → modularize by domain over time
- Introduce governance and APIs gradually
- Split database logically first, physically later

Modular Monolith Advantages

- Better domain alignment than layered monoliths
- Single deployment with domain modularity
- Enables domain-oriented teams
- Maintains runtime performance of monoliths
- Fewer operational headaches than microservices

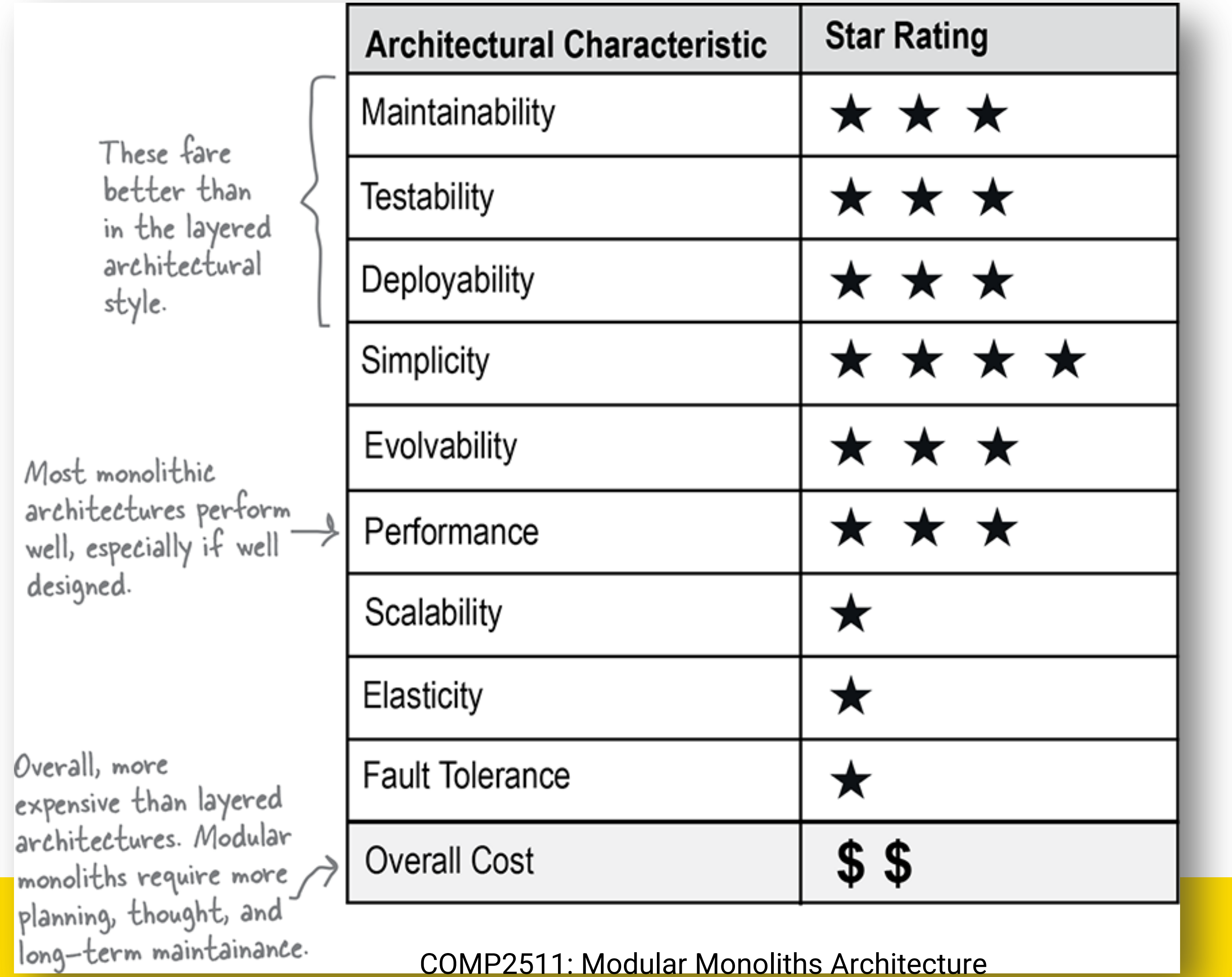
Common Pitfalls in Modular Monoliths

- Bypassing module APIs (direct access)
- Database JOINs across modules
- Overusing shared libraries (tight coupling)
- Lack of observability into module interactions

Techniques for Success

- Define strong module boundaries
- Maintain minimal public API surface
- Invest in automated testing and monitoring
- Review architecture regularly for erosion

Modular Monolith Star Ratings



Architectural Characteristic	Star Rating
Maintainability	★ ★ ★
Testability	★ ★ ★
Deployability	★ ★ ★
Simplicity	★ ★ ★ ★
Evolvability	★ ★ ★
Performance	★ ★ ★
Scalability	★
Elasticity	★
Fault Tolerance	★
Overall Cost	$ $

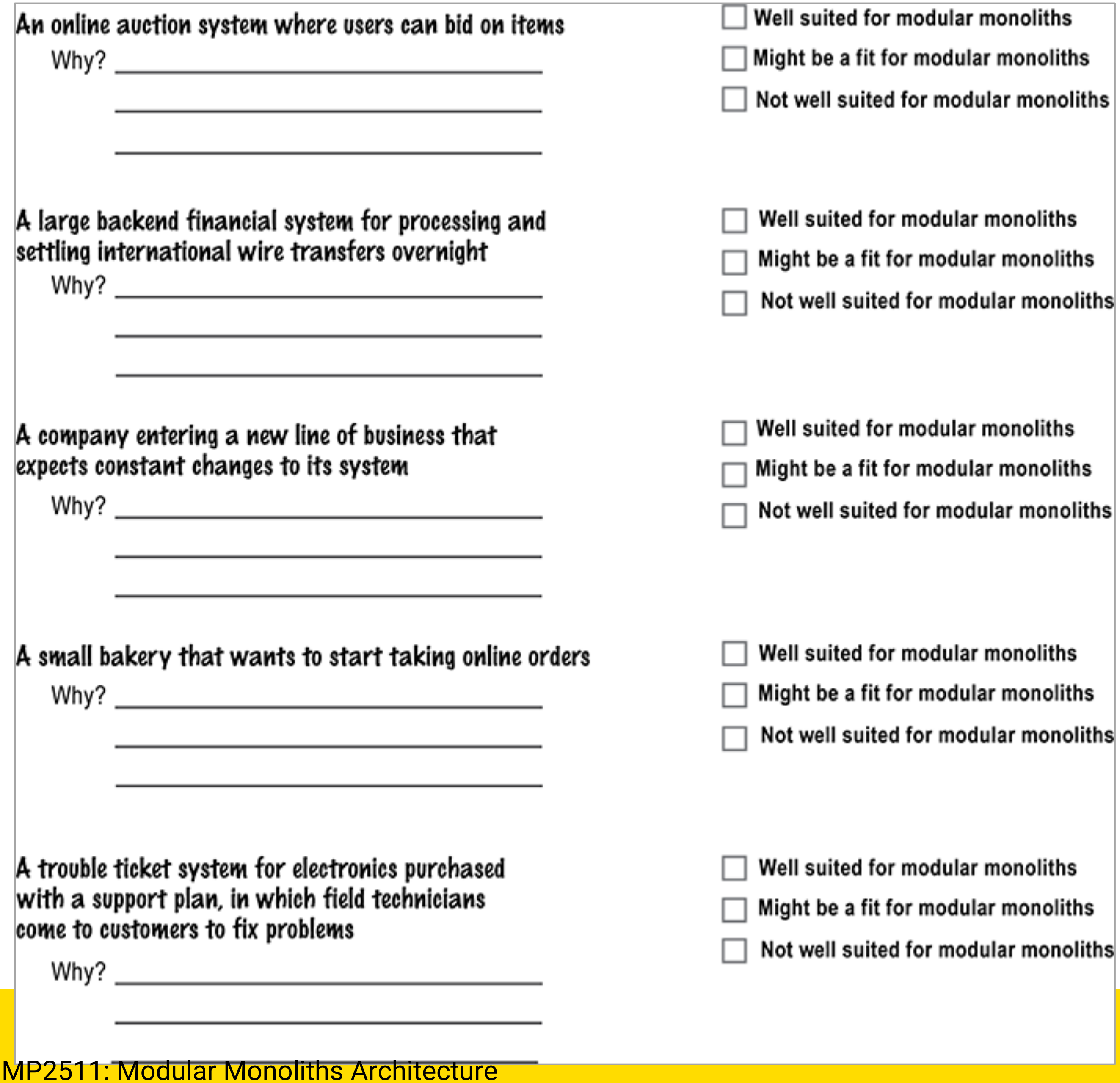
These fare better than in the layered architectural style.

Most monolithic architectures perform well, especially if well designed.

Overall, more expensive than layered architectures. Modular monoliths require more planning, thought, and long-term maintainance.

Exercise

Which of the following systems might be well suited for the modular monolith architectural style, and why?



An online auction system where users can bid on items

Why? ______________________________

____

____

□ Well suited for modular monoliths
□ Might be a fit for modular monoliths
□ Not well suited for modular monoliths

A large backend financial system for processing and settling international wire transfers overnight

Why? ______________________________

____

____

□ Well suited for modular monoliths
□ Might be a fit for modular monoliths
□ Not well suited for modular monoliths

A company entering a new line of business that expects constant changes to its system

Why? ______________________________

____

____

□ Well suited for modular monoliths
□ Might be a fit for modular monoliths
□ Not well suited for modular monoliths

A small bakery that wants to start taking online orders

Why? ______________________________

____

____

□ Well suited for modular monoliths
□ Might be a fit for modular monoliths
□ Not well suited for modular monoliths

A trouble ticket system for electronics purchased with a support plan, in which field technicians come to customers to fix problems

Why? ______________________________

____

____

□ Well suited for modular monoliths
□ Might be a fit for modular monoliths
□ Not well suited for modular monoliths