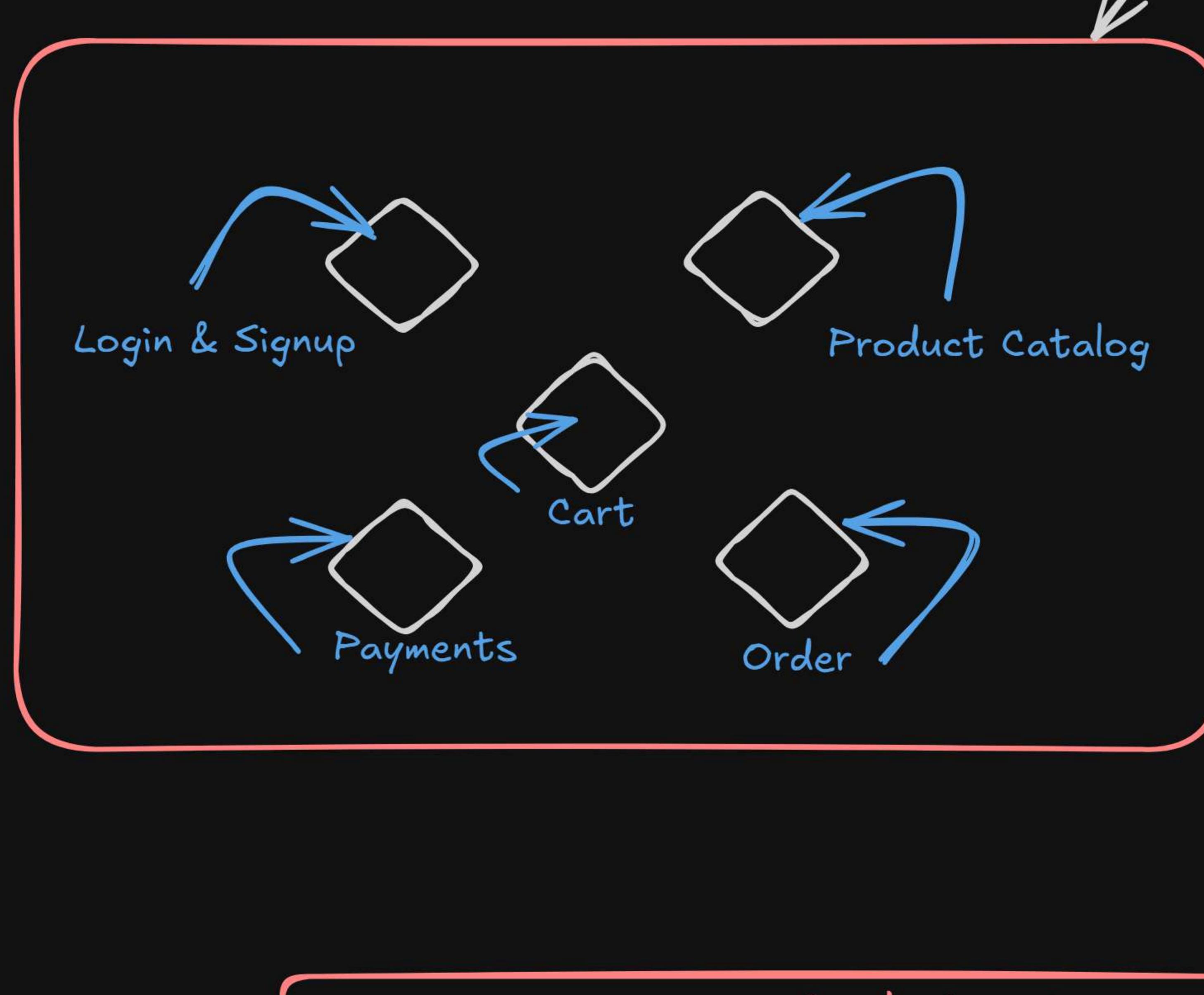
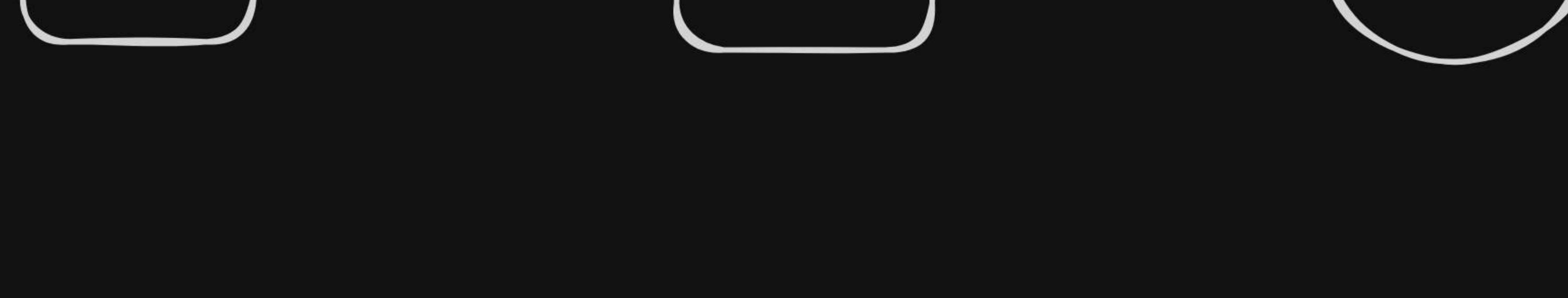


Monolithic Architecture

- all the code is bundled together in a single unit → one deployable artifact (like a .war, .jar, .exe, or Docker image)
- all authentication, payments, search, notifications etc in a single unit



Pros:

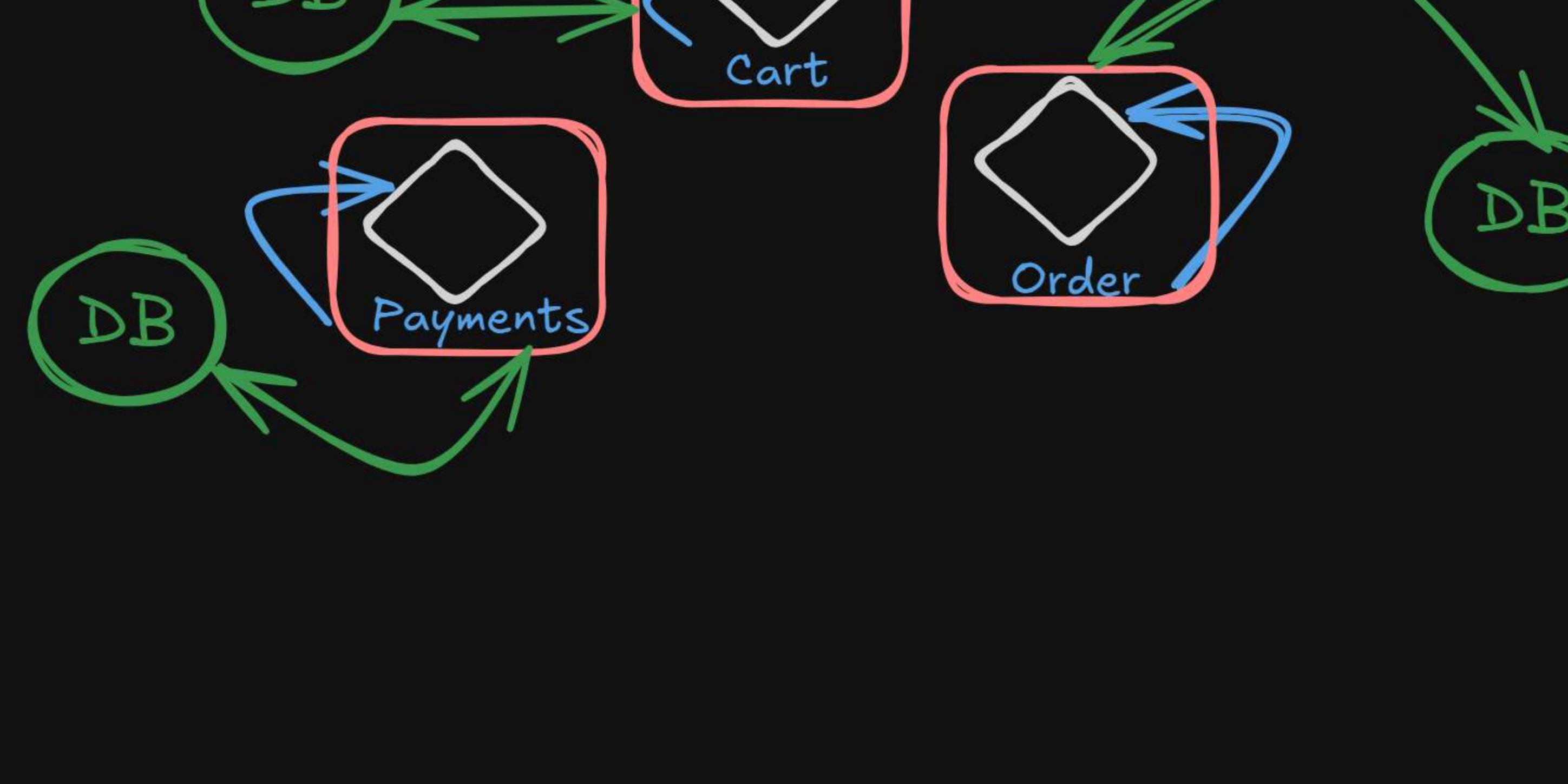
- single unit → easy for a small team to manage
- Easy Deployment: no complex pipelines
Example: Just copy the .war file to your Tomcat server and you're done.
- Fast Performance: Function calls instead of network calls

Cons:

- Scaling Problems
- Tightly Coupled:
Everything depends on everything.
- Slow Development with Big Teams:
Multiple teams editing the same codebase = merge conflicts, coordination overhead.
- Hard to Adapt New Tech
- Deployment Risk

Microservices Architecture

- the application is split into multiple small, independent services.
- Each service can be developed, deployed, and scaled independently.



Pros

- Scalability
- Independent Deployment
- If one service fails, others still work
- Flexibility in tech

Cons

- Complex communication between many services
- DevOps Overhead: Need Kubernetes/Docker, CI/CD, monitoring, logging, service discovery.
- Network Latency
- Each service may have its own DB, keeping data consistent
- Debugging

How to break a monolith into microservices

1. Don't split randomly – use Domain-Driven Design

Break by business capability, not by technical layers.

technical:

Frontend Service

Backend Service

Database Service

business

User Service (authentication, profiles)

Order Service (cart, checkout, order history)

Payment Service (transactions, refunds)

Inventory Service (stock, availability)

Notification Service (emails, SMS, push)

2. one service at a time

3. Expose APIs

4. database

5. API Gateway

6. Repeat for others

- If services share the same DB, they're still coupled at the data layer
- Choice of DB
- Fault isolation

Problems with Separate Databases

- Multiple databases = more infra to manage, more monitoring, more backups

- Complex Queries
Get all orders with user details:

Hybrid Approach (Common in Real Life)

