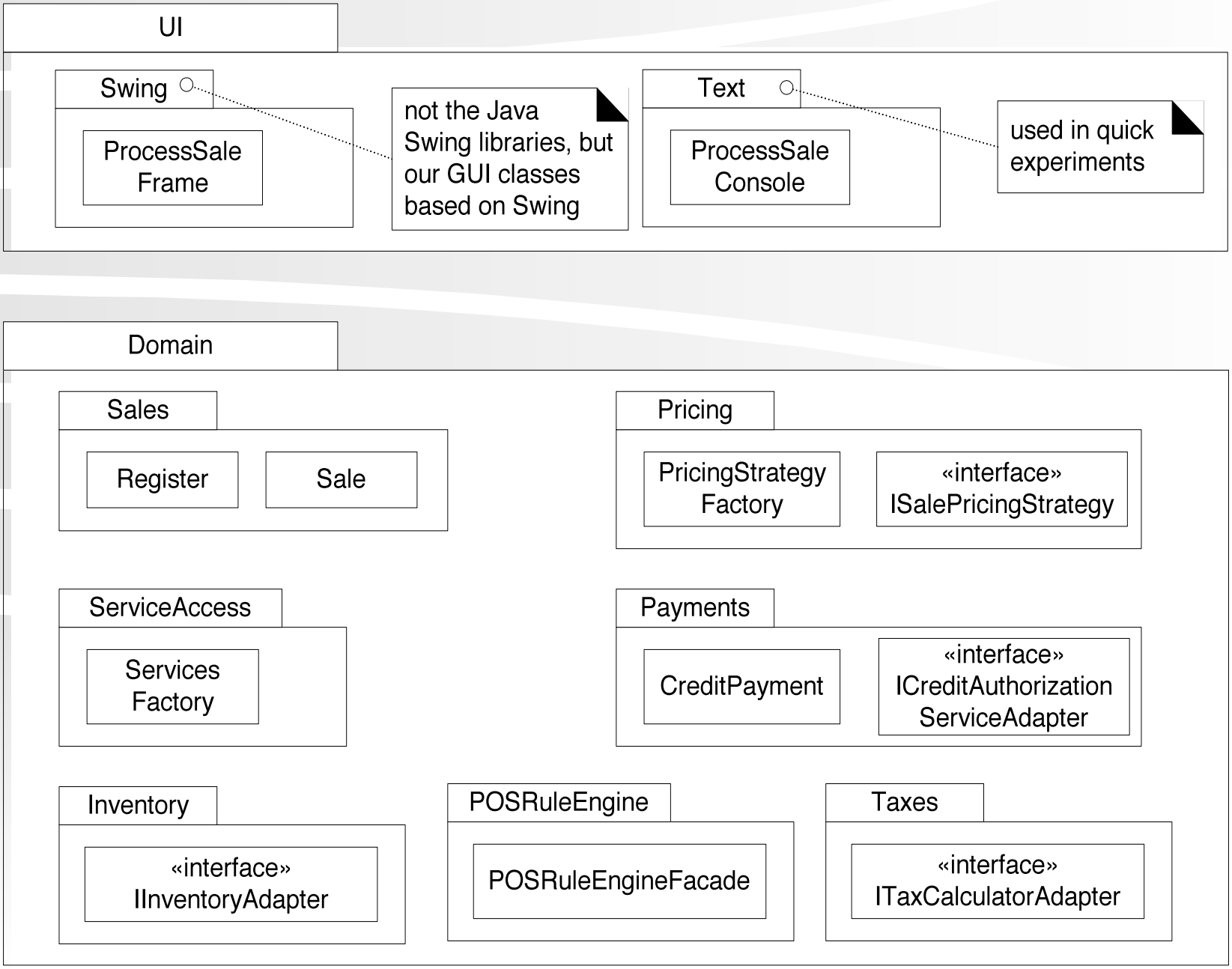
Logical Architecture Refinement and Package Design

Shawn Bohner
Office: Moench Room F212
Phone: (812) 877-8685
Email: bohner@rose-hulman.edu

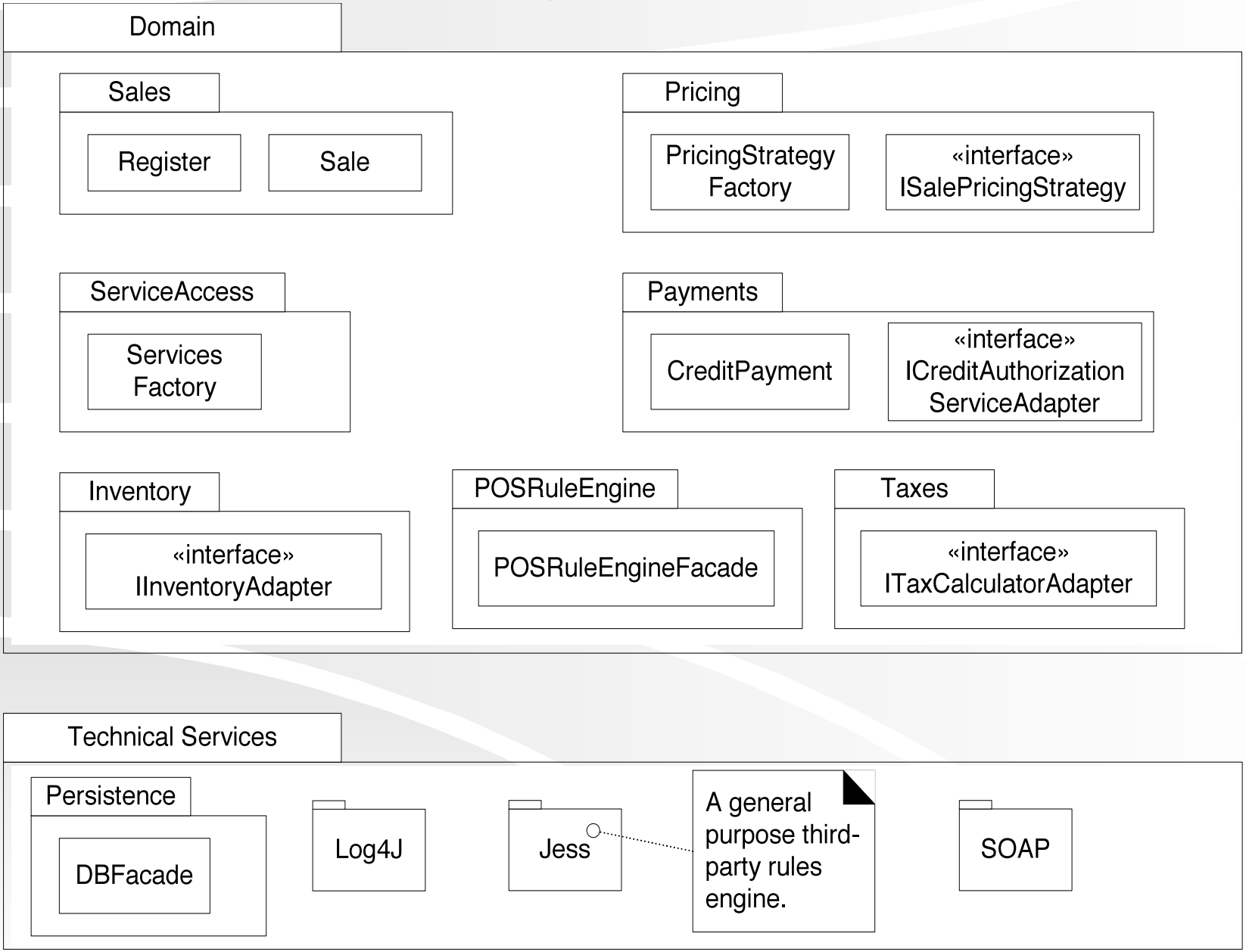
Today's Agenda

- Logical Architecture Refinements
- Package Design
- Design Studio:
  Team 16 – XNA Games

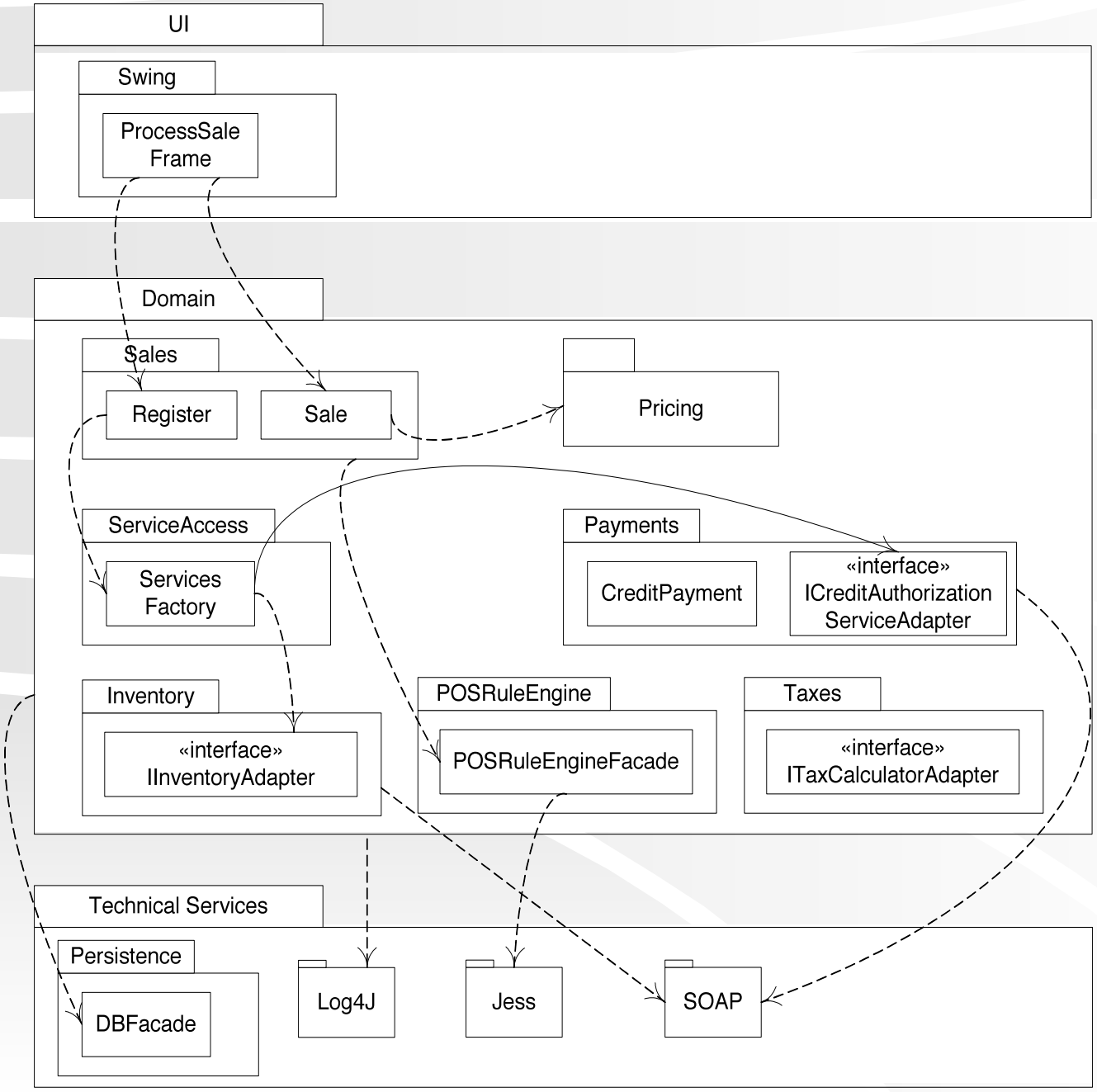
NextGen POS Logical Architecture (1 of 2)



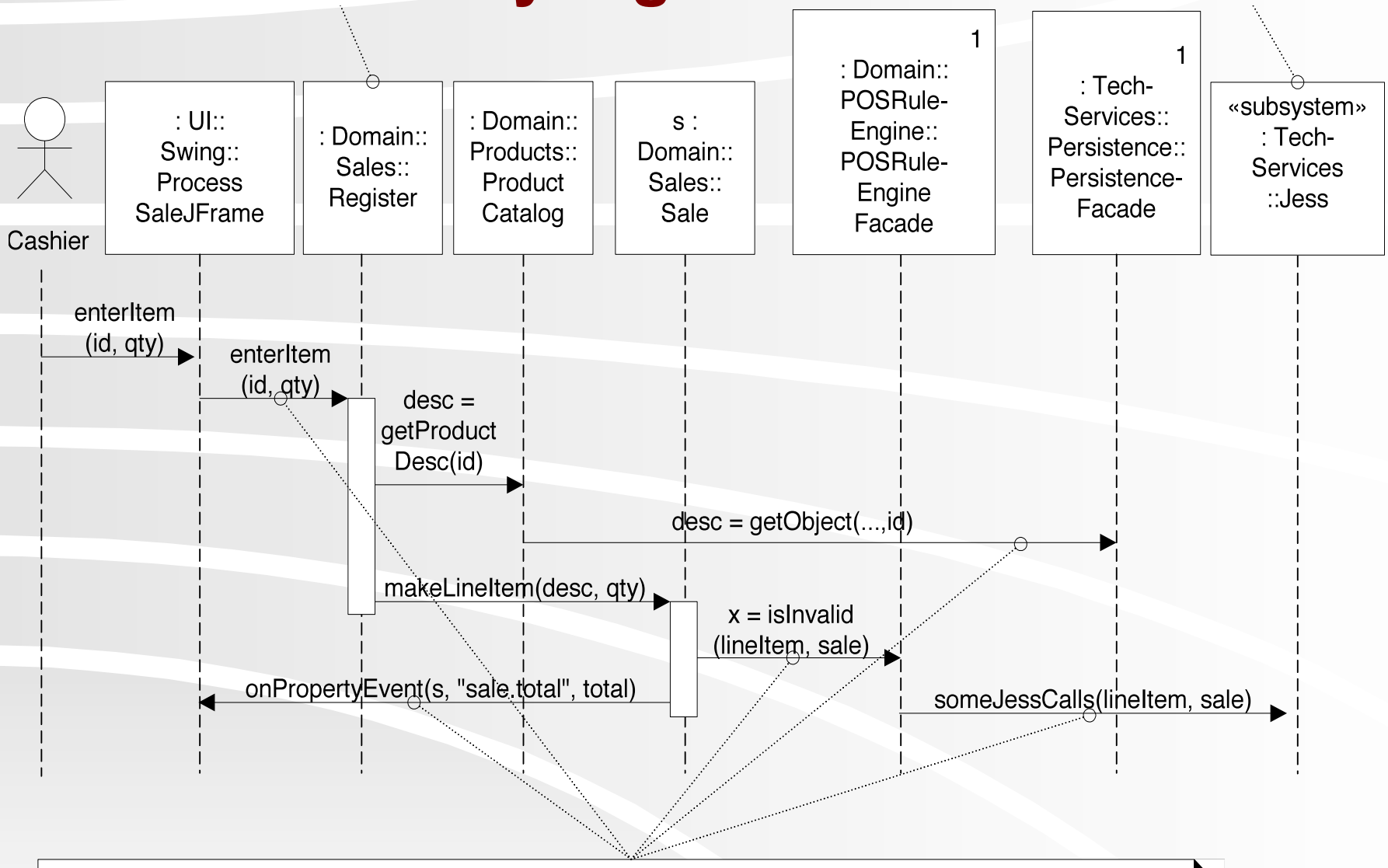
NextGen POS Logical Architecture (2 of 2)



Inter-Layer/Intra-Package Coupling



Architecturally Significant Scenarios



Points of crossing interesting boundaries or layers. These are especially noteworthy for people who need to understand the system, and thus are highlighted in this diagram. This diagram supports communicating the l*ogical view of the architecture*(a UP term) because it emphasizes architecturally signifi cant information.

Q1

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Architectural Level Design Decisions

- What are the big parts?
  - e.g., Layers and partitions
- How are they connected?
  - e.g., Façade, Controller, Observer

Recall: Common Layers

- User Interface
- Application
- Domain
- Business Infrastructure
- Technical Services
- Foundation

Systems will have many, but not necessarily all, of these

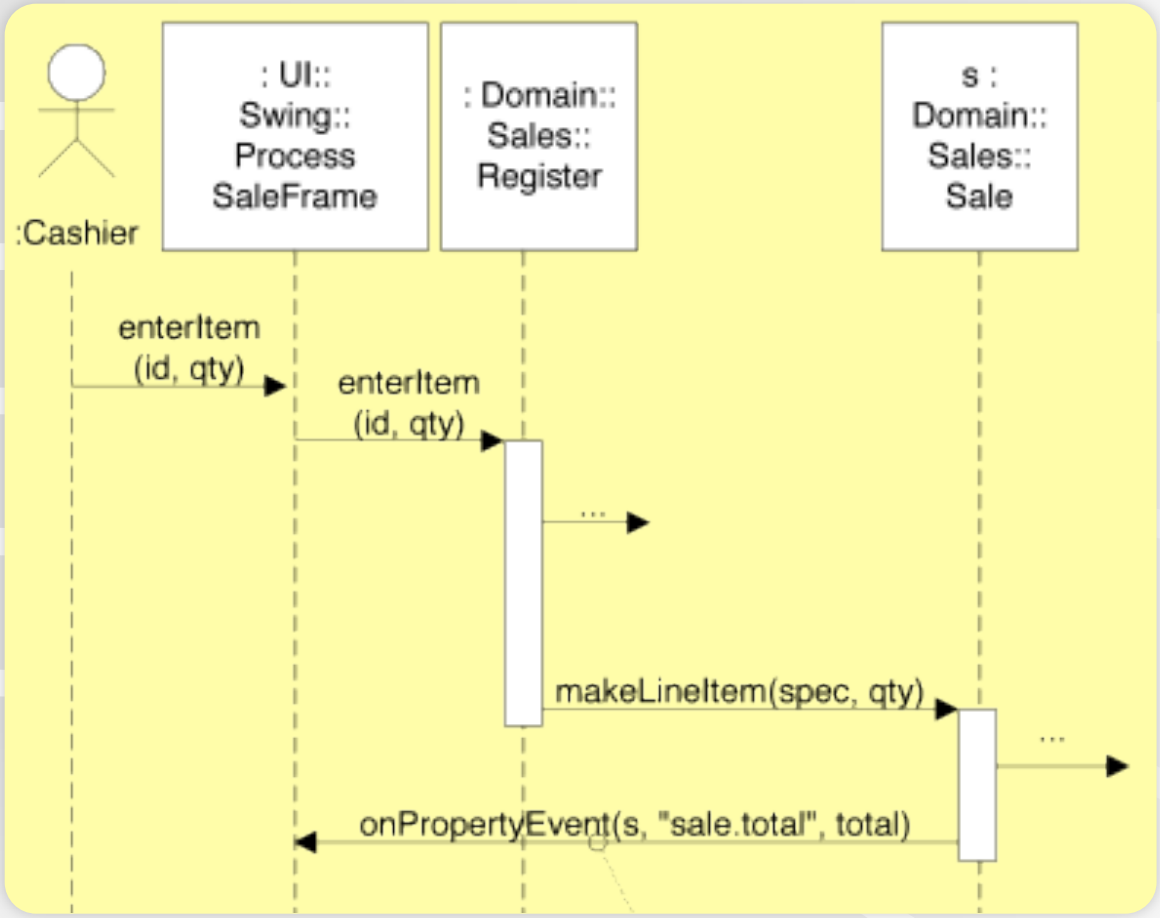
Simple Packages vs. Subsystems

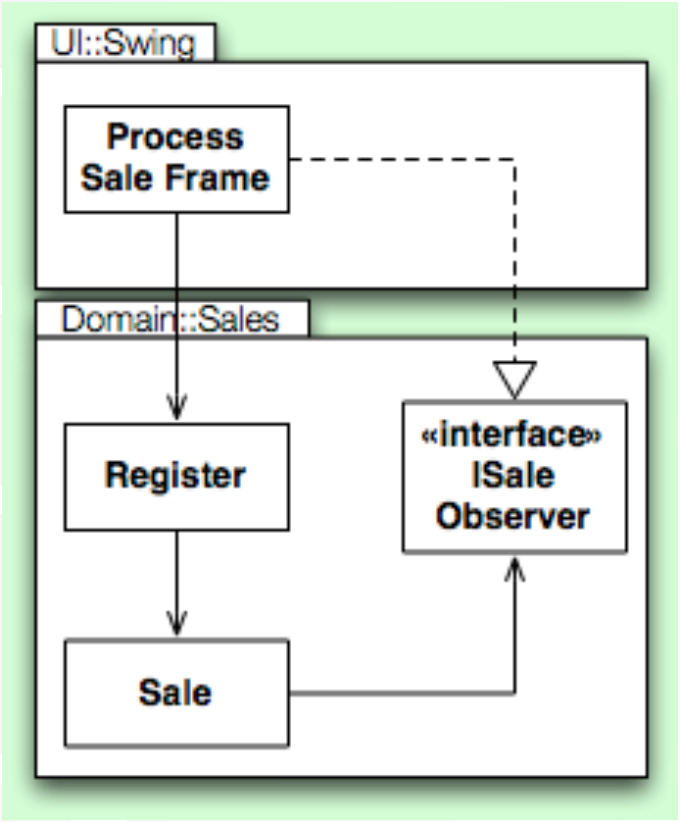
- *Simple package*: just groups classes
  - Pricing
  - Sales
- *Subsystem*: discrete, reusable "engine"
  - Persistence
  - POSRuleEngine

Subsystems Often Provide a Façade

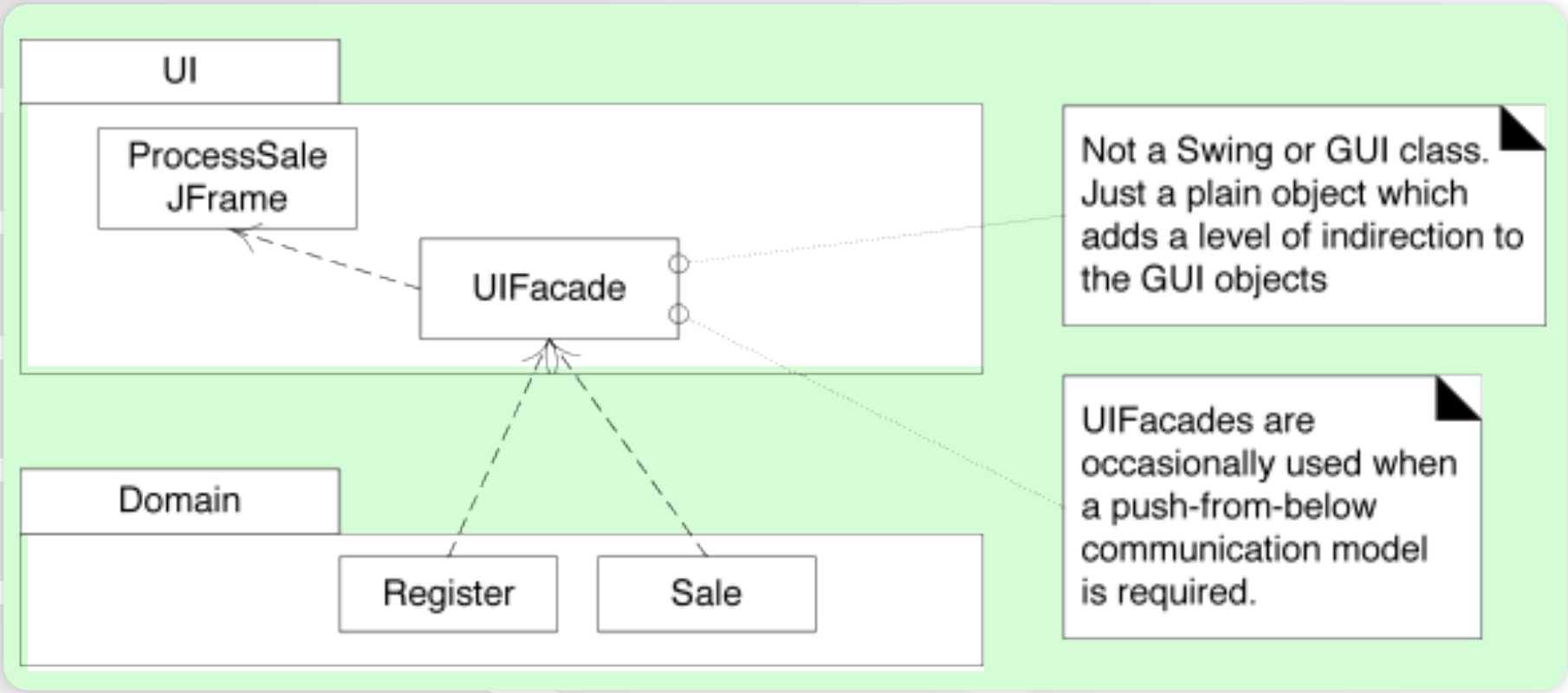
- Serves as a single variation point
- Defines the subsystems services
- Exposes just a few high-level operations
  - High cohesion
  - Allows different deployment architectures

Upward Collaboration with Observer





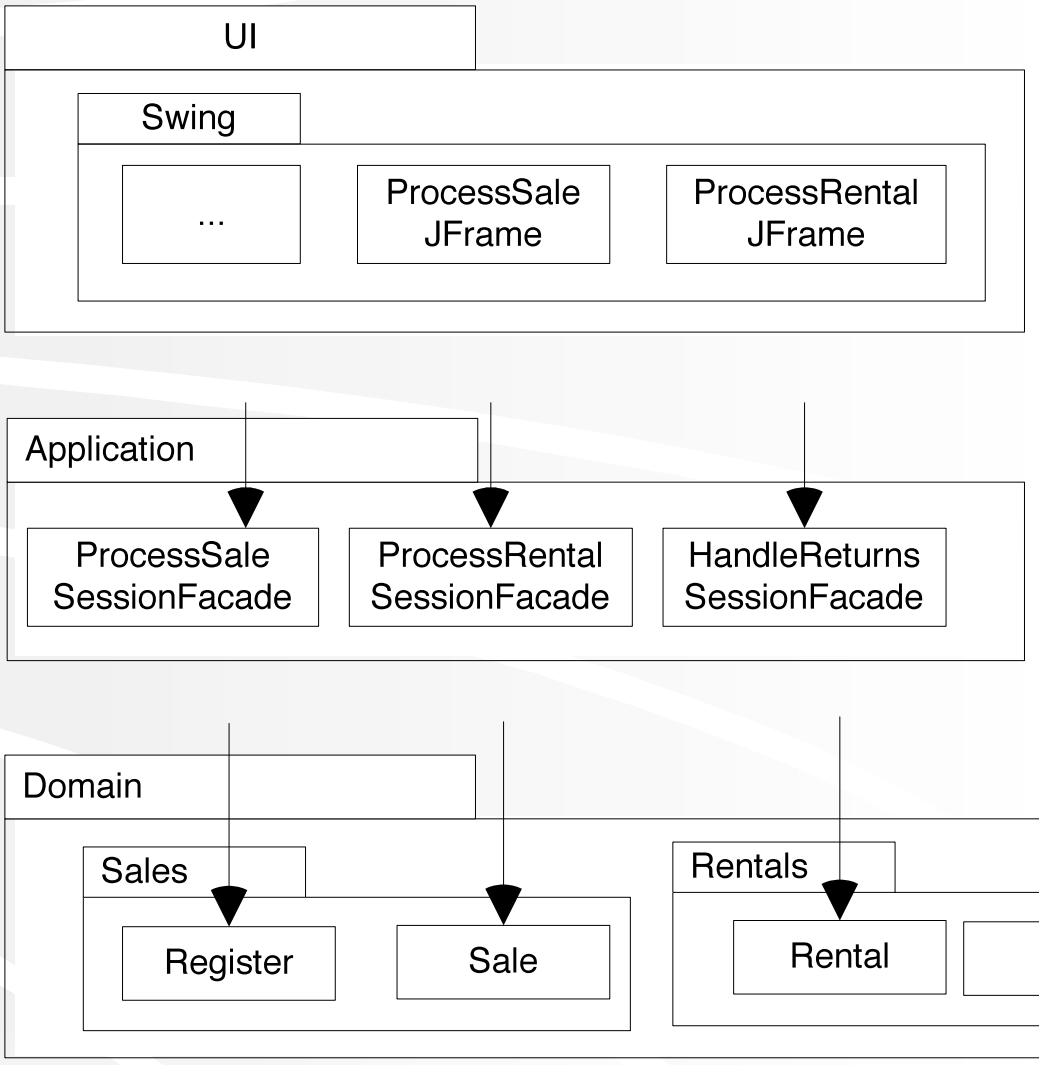
Alt: Upward Collaboration with UI Façade



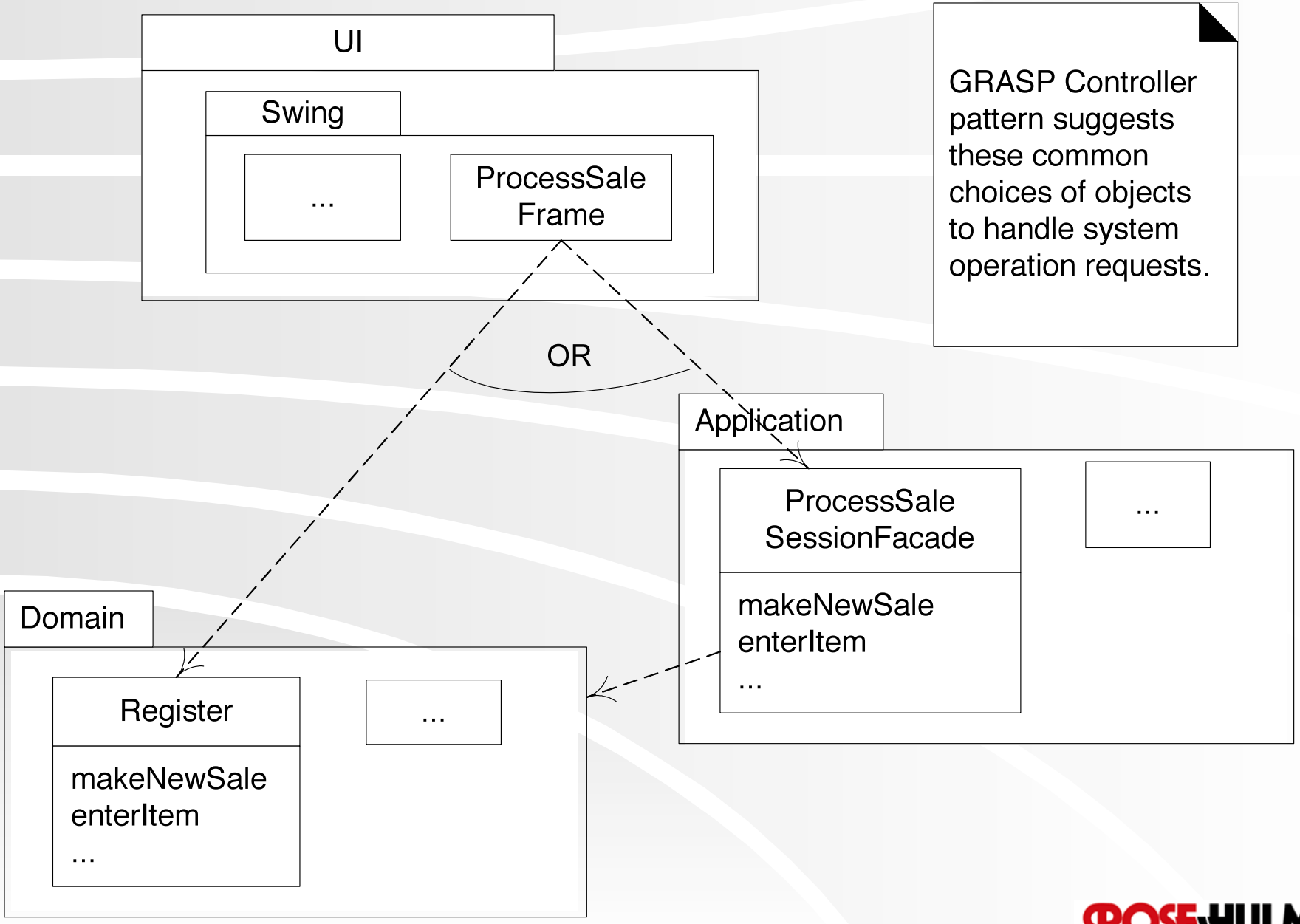
When might this be useful?

Application Layer

- Maintains session state
- Houses Controllers
- Enforces order of operations
- Useful when:
  - Multiple UIs
  - Distributed systems with UI and Domain separated
  - Insulating Domain from session state
  - Strict workflow



Controller Choices



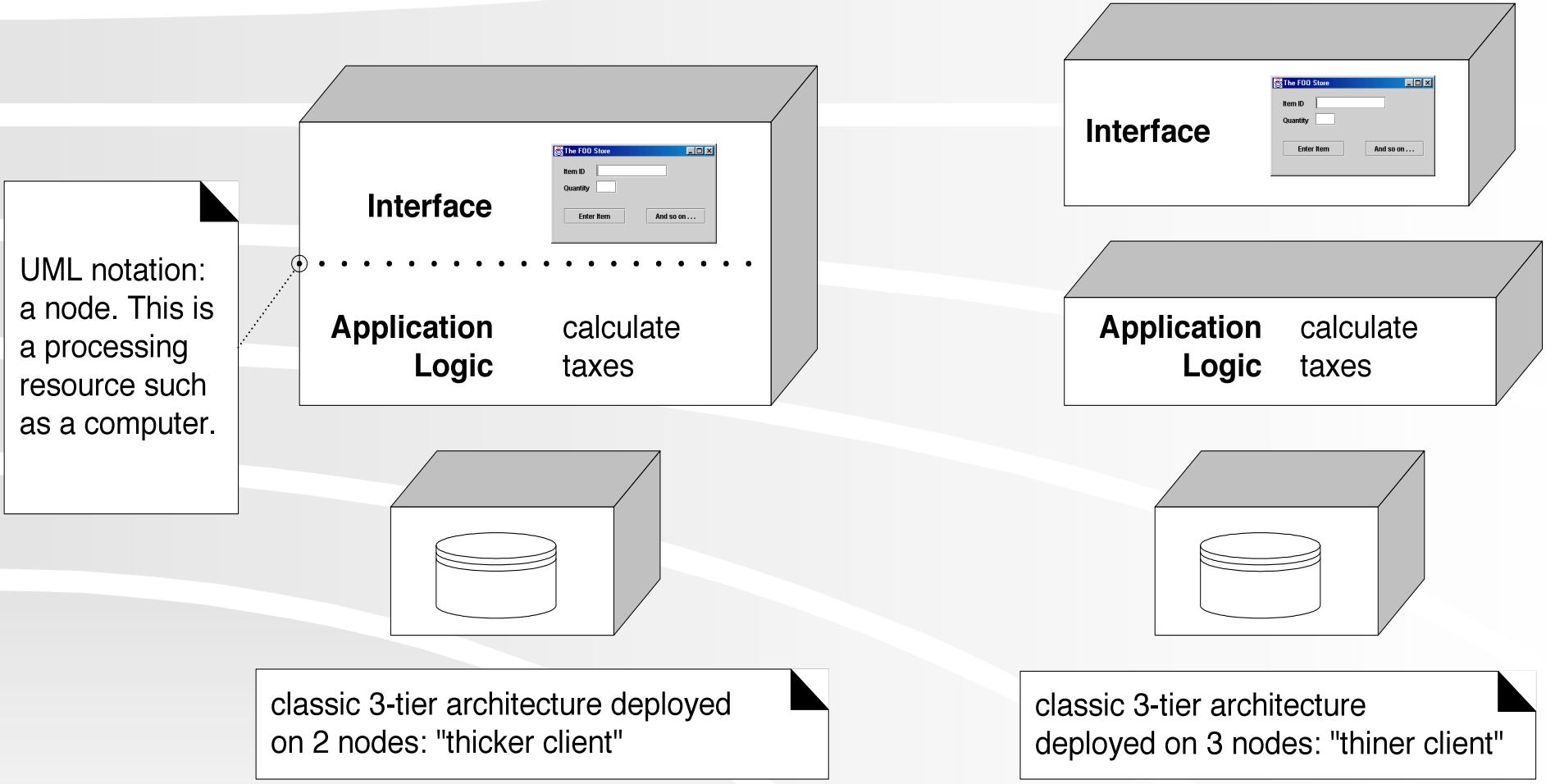
Typical Coupling Between Layers

- From higher layers to Technical Services and Foundation
- From Domain to Business Infrastructure
- From UI to Application and Application to Domain
- In desktop apps: UI uses Domain objects directly
  - E.g., Sales, Payment
- Distributed apps: UI gets data representation objects
  - E.g., SalesData, PaymentData

Liabilities with Layers

- Performance
  - E.g., game applications that need to directly communicate with graphics cards
- Poor architectural fit sometimes
  - Batch processing (use “Pipes and Filters”)
  - Expert systems (use “Blackboard”)

Classic 3-Tier Architecture on Two Nodes



Physical Package Design

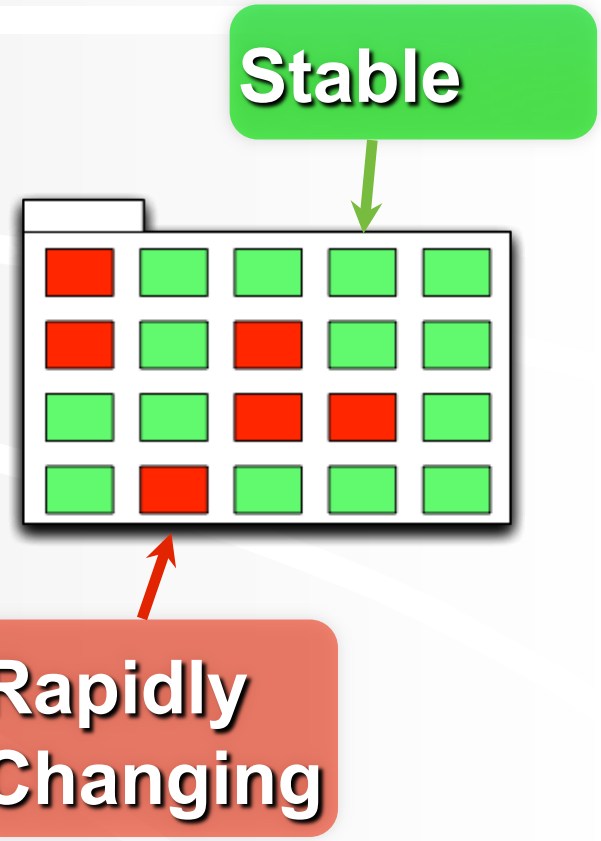
- Goal: define physical packages so they can be:
  - Developed independently
  - Deployed independently

Multiple logical packages might be developed together physically

- Packages should depend on other packages that are more stable than themselves
  - Avoids *version thrashing*

Package Organization Guidelines (1 of 3)

- Package functionally cohesive slices
  - Limit strong coupling within the package
  - Achieve weak coupling between packages
- Package a family of interfaces
  - Factor out independent types
- Package by clusters of unstable classes



Package Organization Guidelines (1 of 3)

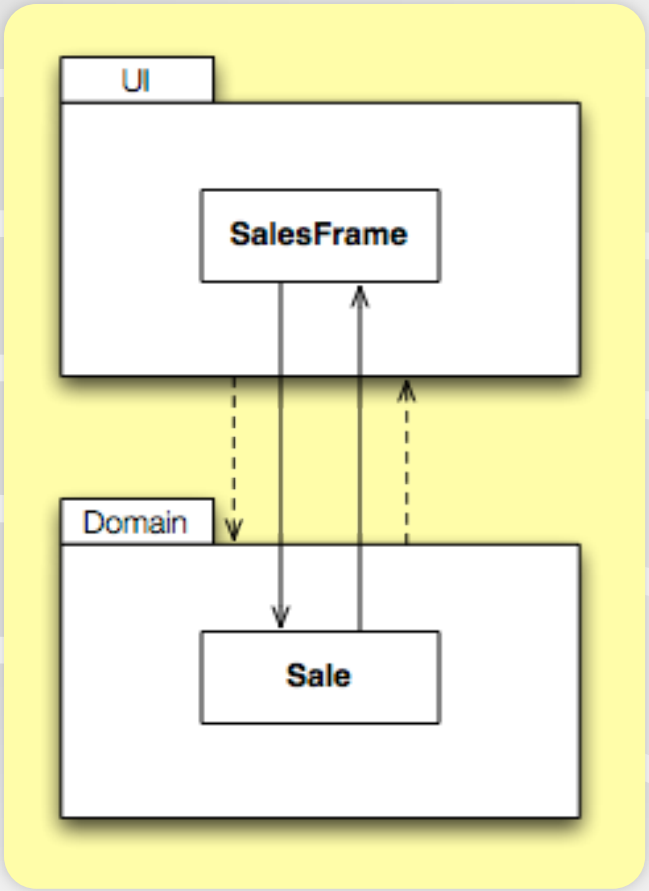
- Make the most depended-on packages the most stable
- Can increase stability by:
  - Using only or mostly interfaces and abstract classes
  - Not depending on other packages
  - Encapsulating dependencies (e.g., with Façade)
  - Heavy testing before first release
  - Fiat

Iron-fisted rule, not the Italian car brand

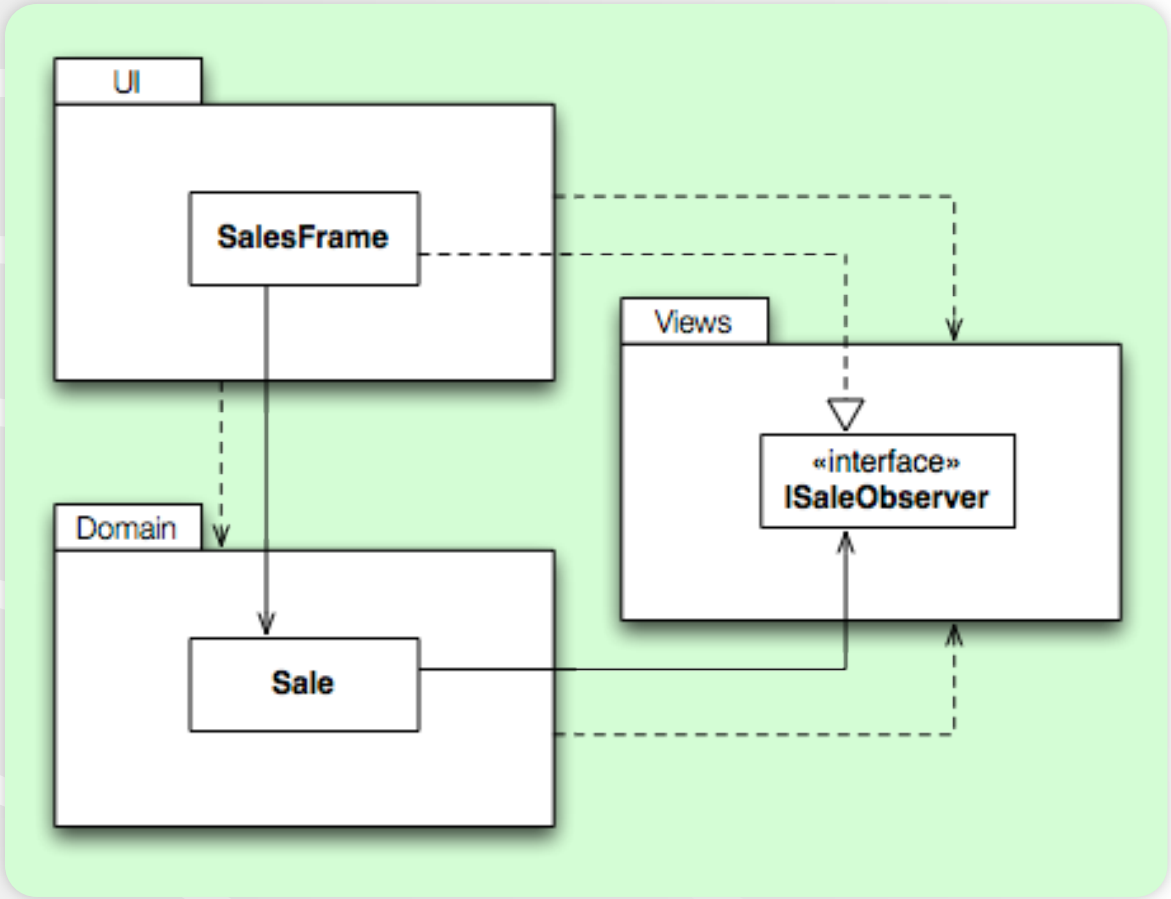
Package Organization Guidelines (1 of 3)

- Factor out the independent types
  - Grouping by common functionality may not provide right level of granularity in packages
  - e.g., Common Utilities
- Use factories to reduce dependencies on concrete packages
  - E.g., instead of exposing all the subtypes, expose an abstract superclass and a factory
- No cycles between packages
  - Cycles often force packages to be developed and released together

Breaking Dependency Cycles Between Packages



Cyclic Coupling



Better: Cycle Removed!

Design Studio:
Team 16: XNA Games

~5 minutes:
Team describes problem and current solution (if any)

~3 minutes:
Class thinks about questions, alternative approaches

~12 minutes:
On-board design with team modeling and instructor advising/facilitating

Homework and Milestone Reminders

- Read Chapter 36
- Milestone 5 – Iteration 3 Junior Project System with finalized Design Document
  - Preliminary Design Walkthrough on Friday, February 12th, 2010 during project meeting.
  - Final Project Due by 11:59pm Friday, February 19th, 2010.