Architectural views

Dr. C. Constantinides, P.Eng.
Computer Science and Software Engineering
Concordia University

Motivation for architectural views

- Software architecture is commonly organized in *views*, (or *viewpoints*) which are analogous to the different types of blueprints made in building architecture.

- Many large system specifications are complex and extensive.
  - No single individual can fully comprehend all aspects of the specifications.
  - In order to fully capture a software system's architecture multiple views are needed.

Motivation for architectural views /cont.

- Stakeholders would have different interests in a given system and different reasons for examining the system's specifications.
- A business executive will ask different questions about a system than would a system developer.
- A *view* is a representation of a whole system from the perspective of a related set of concerns.

Views

- Some possible views are:
  - Functional/logical view.
  - Code/module view.
  - Development/structural view.
  - Concurrency/process/thread view.
  - Physical/deployment view.
  - User action/feedback view.
  - Data view.

Architectural description languages

- No consensus has yet been reached on which language (“architecture description language”) should be adopted for describing software architectures.
- The UML was established as a standard *"to model systems (and not just software),"* and thus applies to views about software architecture.

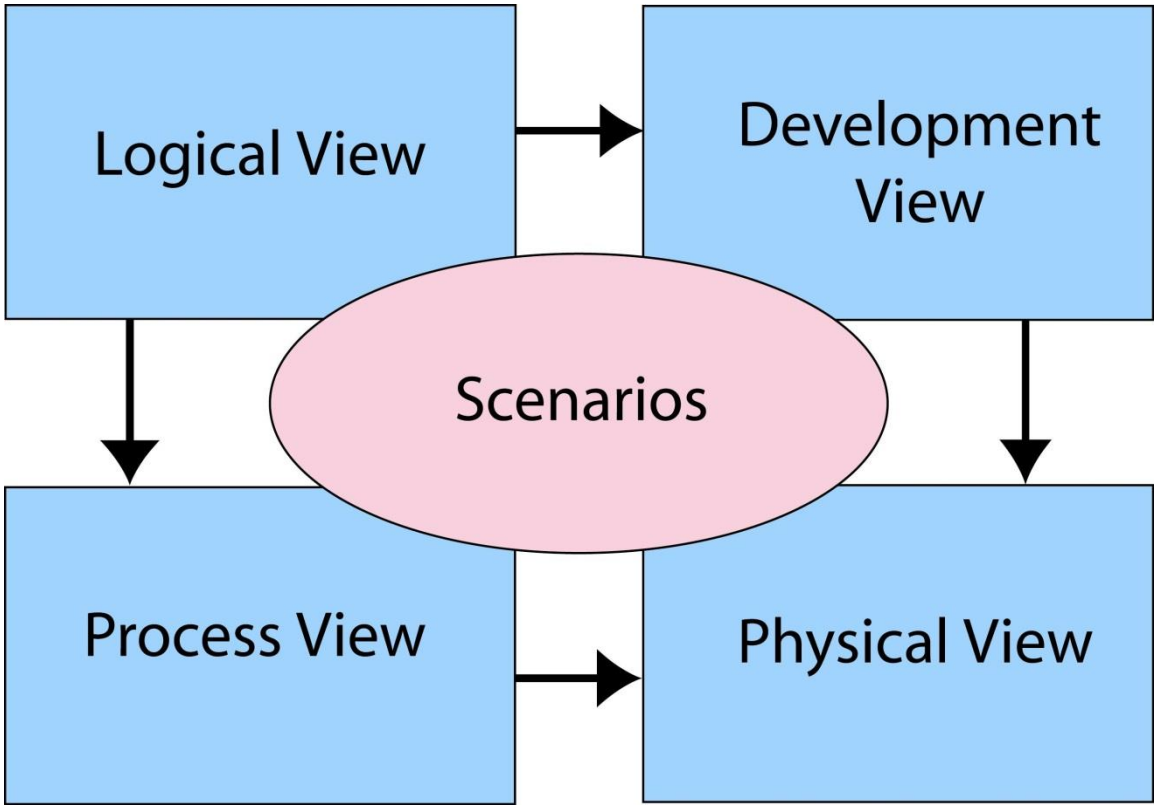
The 4+1 view model

- Adopt several concurrent *views* or *perspectives*, with different notations, each one addressing one specific set for concerns.

- The “4+1” is a view model designed by Philippe Kruchten for *"describing the architecture of software-intensive systems, based on the use of multiple, concurrent views.“*

- The four views of the model are *logical*, *development*, *process* and *physical view*. In addition the *use case model* serves as the 'plus one' view.

The 4+1 view model /cont.



Adapted from “Architectural Blueprints – the 4+1 view model of software architecture”, Kruchten, P. (1997)

Use-case view

4+1: Use case view (scenarios)

- The view of a system’s architecture that encompasses the use cases that describe the behavior of the system as seen by its end users (actors) and other external stakeholders.

4+1: Use case view (scenarios) /cont.

- The scenarios describe sequences of interactions between actors and the system, or between systems.
- UML Diagrams used to represent the scenario view include the *use case diagram*.

Logical view

4+1: Logical view

- The logical view is concerned with the functionality that the system provides to end-users.
- Illustrates the collaborations that realize the system's use cases, the subsystems that provide the central layering and decomposition of the system, and the interfaces that are exposed by those subsystems and the system as a whole.

4+1: Logical view

- Focuses on
  - Functionality.
  - Key Abstractions.
  - Mechanisms.
  - Separation of concerns and distribution of responsibilities.

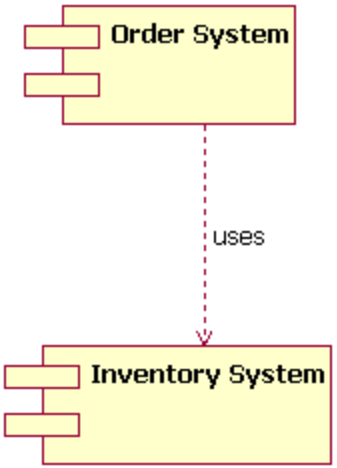
- UML Diagrams used to represent the logical view include the **class diagram**, and **interaction diagrams** (communication and sequence diagrams).

Development view

4+1: Development (or implementation) view

- The view of a system's architecture that encompasses the components used to assemble the physical system.
- UML Diagrams used to represent the development view include the **package diagrams**, and **component diagrams**.

UML Component diagrams



- A *component* is a class that represents a replaceable unit in the system, the parts of which are encapsulated.

- A component provides a public functionality to the outside through a well defined interface.

Process view

4+1: Process view

- Illustrates the system processes and how they communicate, and focuses on the runtime behavior of the system.

- The process view addresses concurrency and synchronization mechanisms.

4+1: Process view /cont.

- Focuses on
  - Performance.
  - Scalability.
  - Throughput (number/rate of completed tasks).

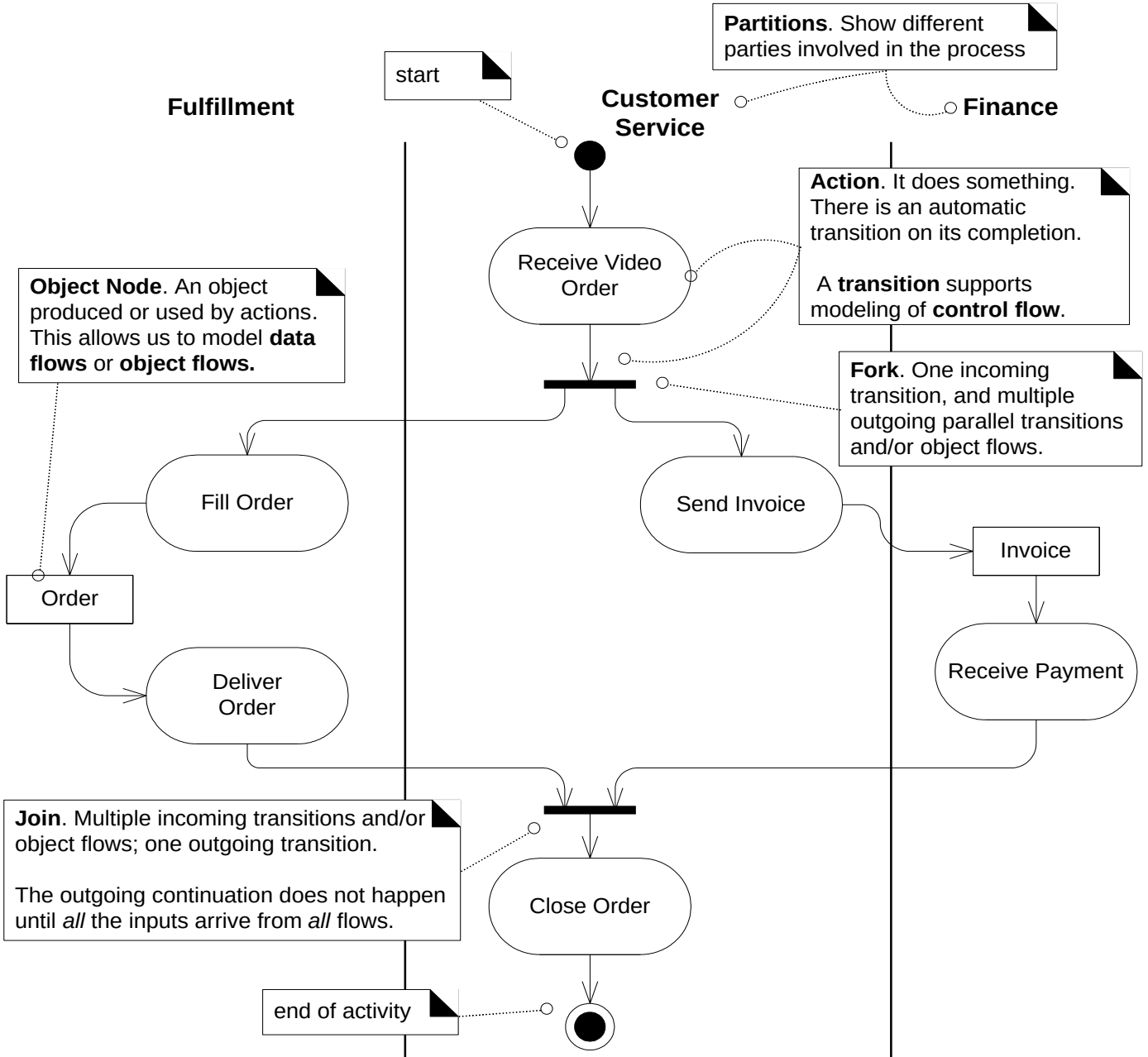
- UML Diagrams to represent process view include the **activity diagram**.

UML Activity diagrams

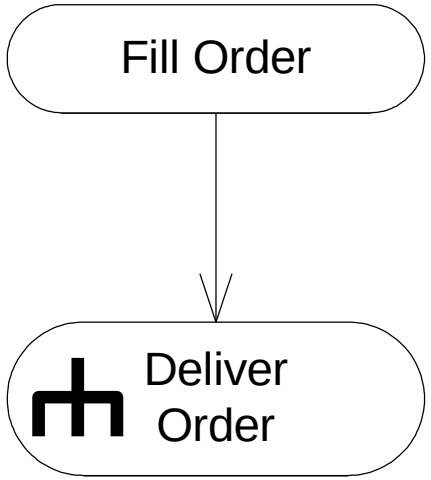
- A UML activity diagram shows sequential and parallel activities in a process.

- Most notation is self-explanatory. Two points:
  1. Once an action is completed there is an automatic outgoing transition.
  2. The diagram can illustrate both control flow and data flow.

- Basic notation includes action, partition, fork, join and object node.

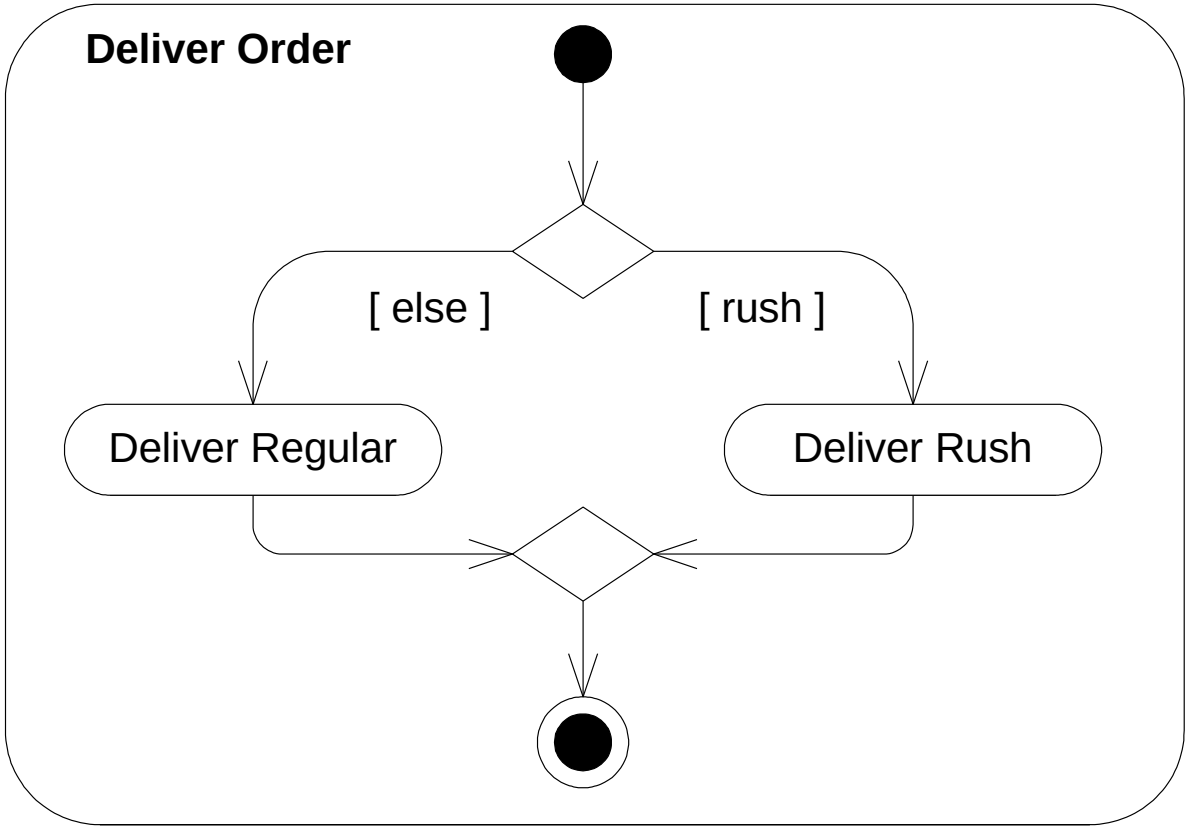

Partitions. Show different parties involved in the process
start
Fulfillment
Customer Service
Finance
Action. It does something. There is an automatic transition on its completion.
A transition supports modeling of control flow.
Receive Video Order
Object Node. An object produced or used by actions. This allows us to model data flows or object flows.
Fork. One incoming transition, and multiple outgoing parallel transitions and/or object flows.
Fill Order
Send Invoice
Invoice
Order
Receive Payment
Deliver Order
Join. Multiple incoming transitions and/or object flows; one outgoing transition.
The outgoing continuation does not happen until all the inputs arrive from all flows.
Close Order
end of activity

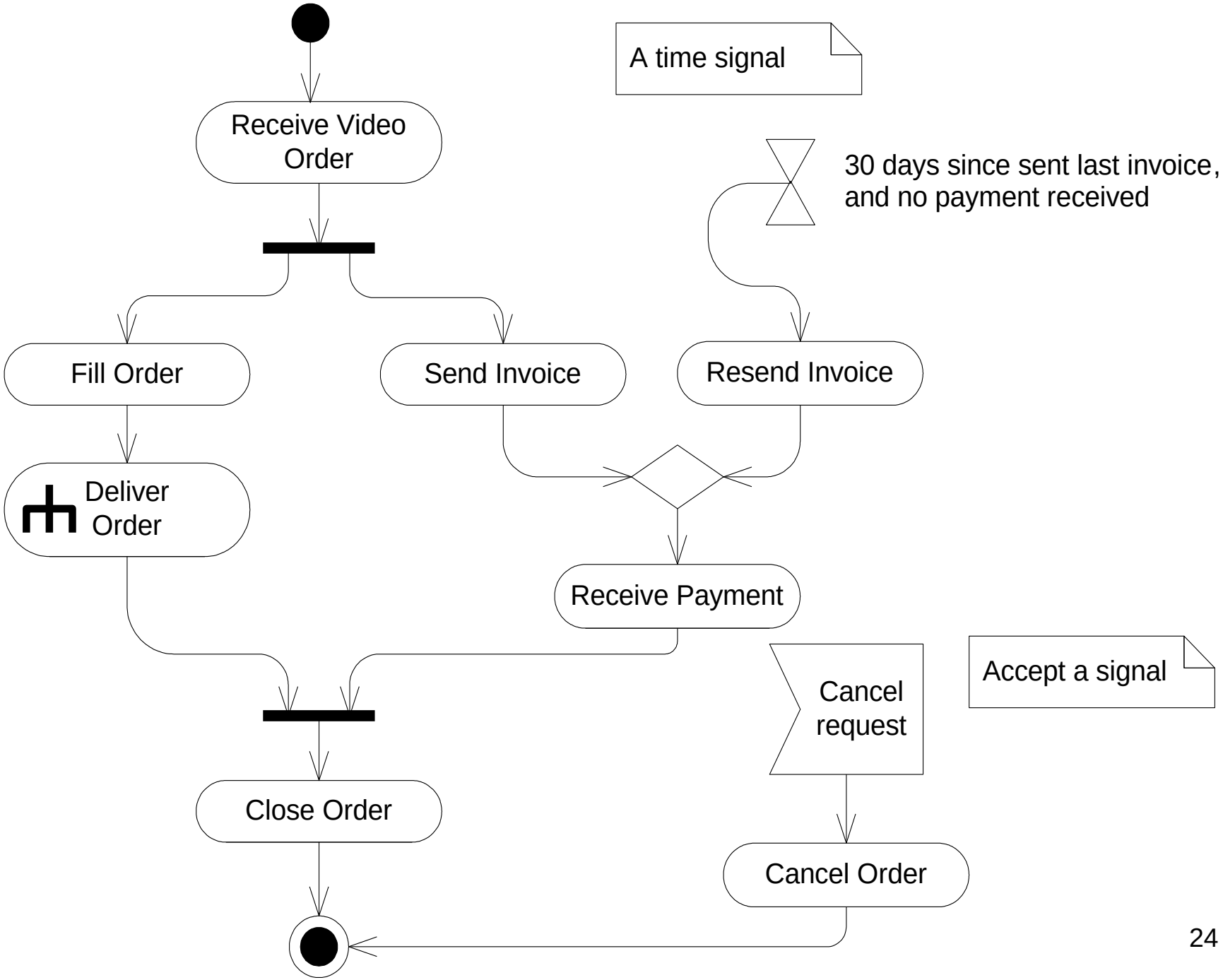
the “rake” symbol (which represents a hierarchy) indicates this activity is expanded in a sub-activity diagram



Decision: Any branch happens. Mutual exclusion

Merge: *Any* input leads to continuation. This is in contrast to a *join*, in which case *all* the inputs have to arrive before it continues.




A time signal
Receive Video Order
30 days since sent last invoice, and no payment received
Fill Order
Send Invoice
Resend Invoice
Deliver Order
Receive Payment
Accept a signal
Cancel request
Close Order
Cancel Order

Physical view

4+1: Physical (or deployment) view

- It is concerned with the topology of software components on the physical layer (represented as nodes forming the system's hardware and software), as well as communication between these components.

- Focuses on
  - Distribution.
  - Communication.

- UML Diagrams used to represent physical view include the **deployment diagram**.

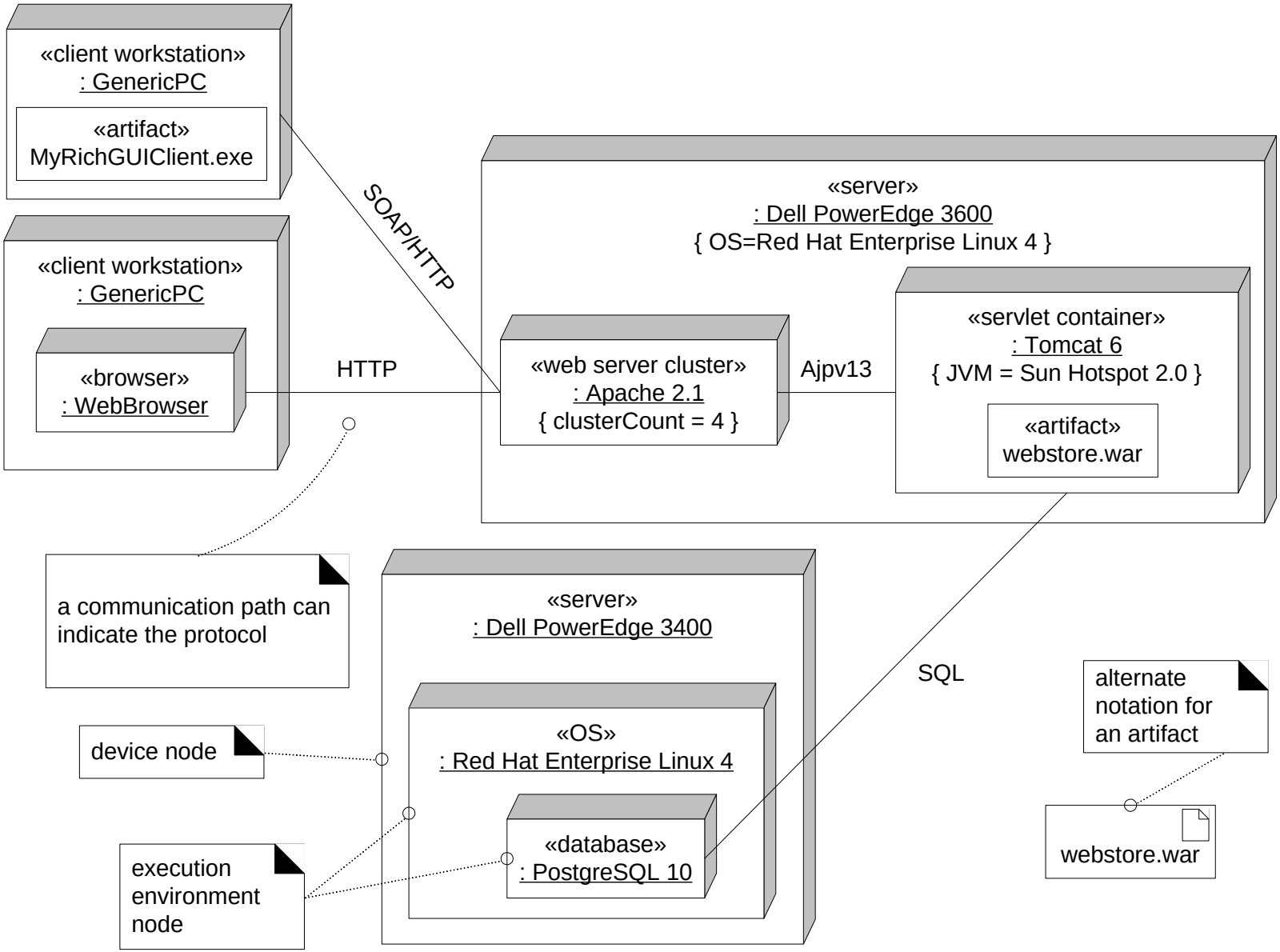
UML Deployment diagrams

- A node can be of two types:

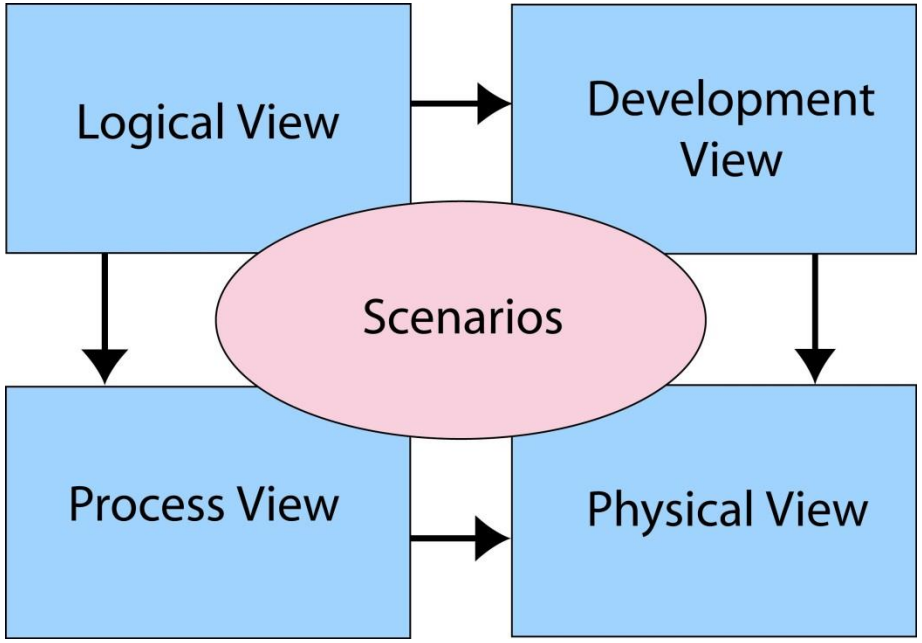
1. *Device node* (or *device*):
   - A physical resource with processing and memory services to execute software (e.g. computer system, mobile phone).

2. *Execution environment node*:
   - A software computing resource that runs within an outer node (e.g. a computer) and which itself provides a service to host other executable software elements.
   - e.g. operating system, virtual machine, database engine.

Example of physical view



4+1: Correspondence between views



- Views are interconnected.
- Start with Logical view and Move to Development or Process view and then finally to Physical view.

Adapting views

- Not all systems require all views
  - Single process (ignore process view)
  - Small program (ignore implementation view)
  - Single processor (ignore deployment view)
- Some systems require additional views
  - Data view.
  - Security view.
  - Other aspects.

Case study on architectural views: Point of sale

- For this system we will adopt the layered architectural style.

- Additionally, and in order to manage the size and the complexity of the system as well as to allow parallel development, each subsystem can be (logically and physically) placed in a package diagram.

Designing the logical architecture /cont.

- A typical (large) system is composed of many logical packages where each one groups a set of cohesive responsibilities.
  - E.g. user interface package, database access, etc.

- An architecture is the organization and the structure of a system.

- The logical architecture describes the system in terms of its conceptual organization in layers, packages, classes, interfaces and subsystems.

The layered architectural style

- Many different architectural styles exist. It is also common to adopt one style for the top-level system while we adopt different styles for subsystems.

- What is important is to carefully select a style which can best capture the requirements of the system. In this case study we will adopt the layered style.

- The large-scale logical structure of a system is organized into discrete layers of distinct, related responsibilities with a clean and cohesive separation of concerns.

- A layer is a large-scale element normally composed of several packages or subsystems and has a cohesive responsibility for a major aspect of the system.

The layered architectural style

- Layers are organized such as “higher” layers (e.g. UI layer) call upon services of “lower” layers (but not normally vice versa).
  - Collaboration and coupling is from higher to lower layers.
- The “lower” layers are low-level and provide general services and the “higher” layers are more application specific.

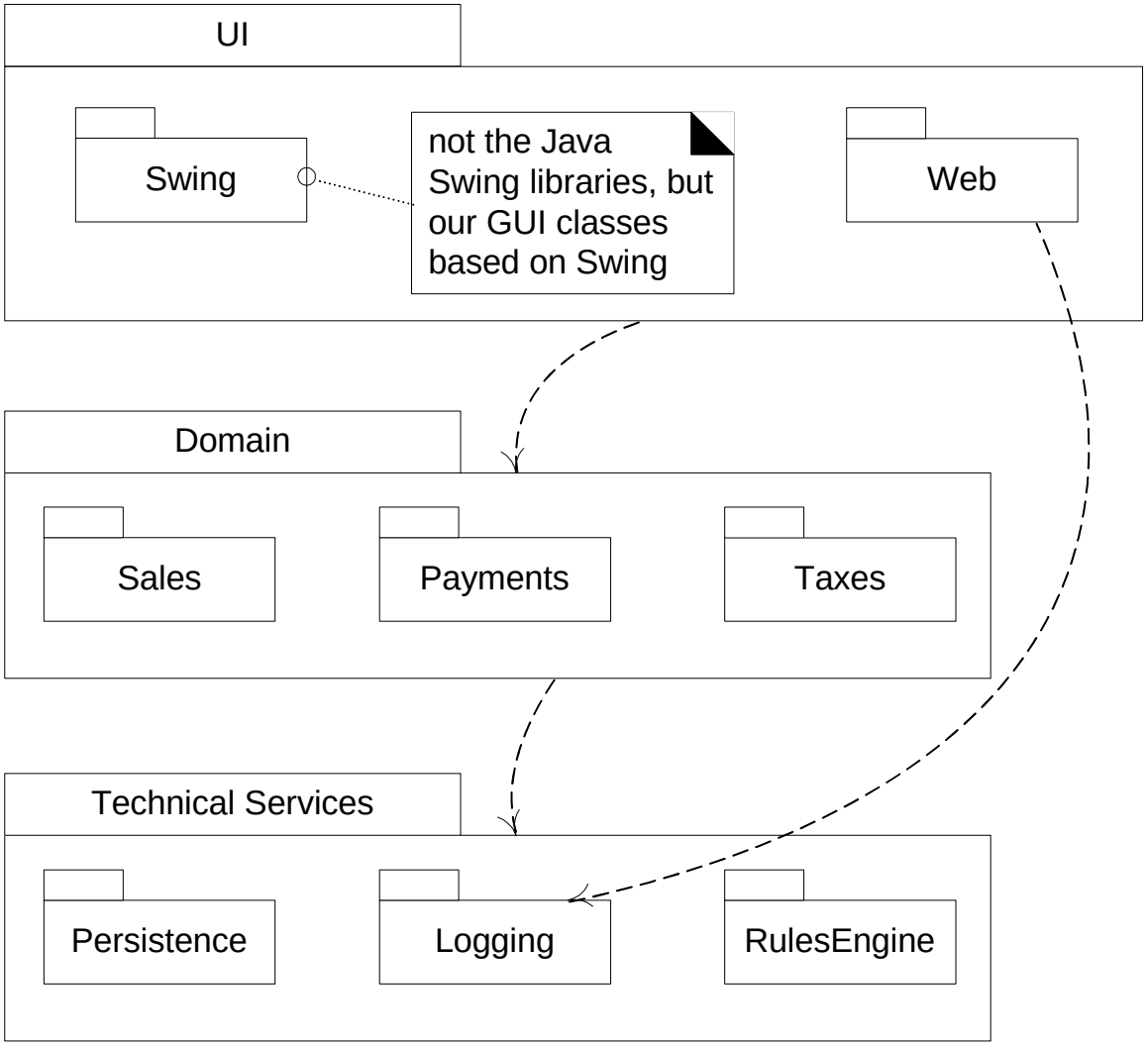
The layered architectural style /cont.

- Typically, layers in an OO system include:
- User interface
- Application logic and domain objects
  - Software objects representing domain concepts (e.g. class Sale).
- Technical services
  - General purpose objects and subsystems that provide supporting technical services (e.g. interfacing with a DBMS)
  - Usually application independent and reusable across several systems.

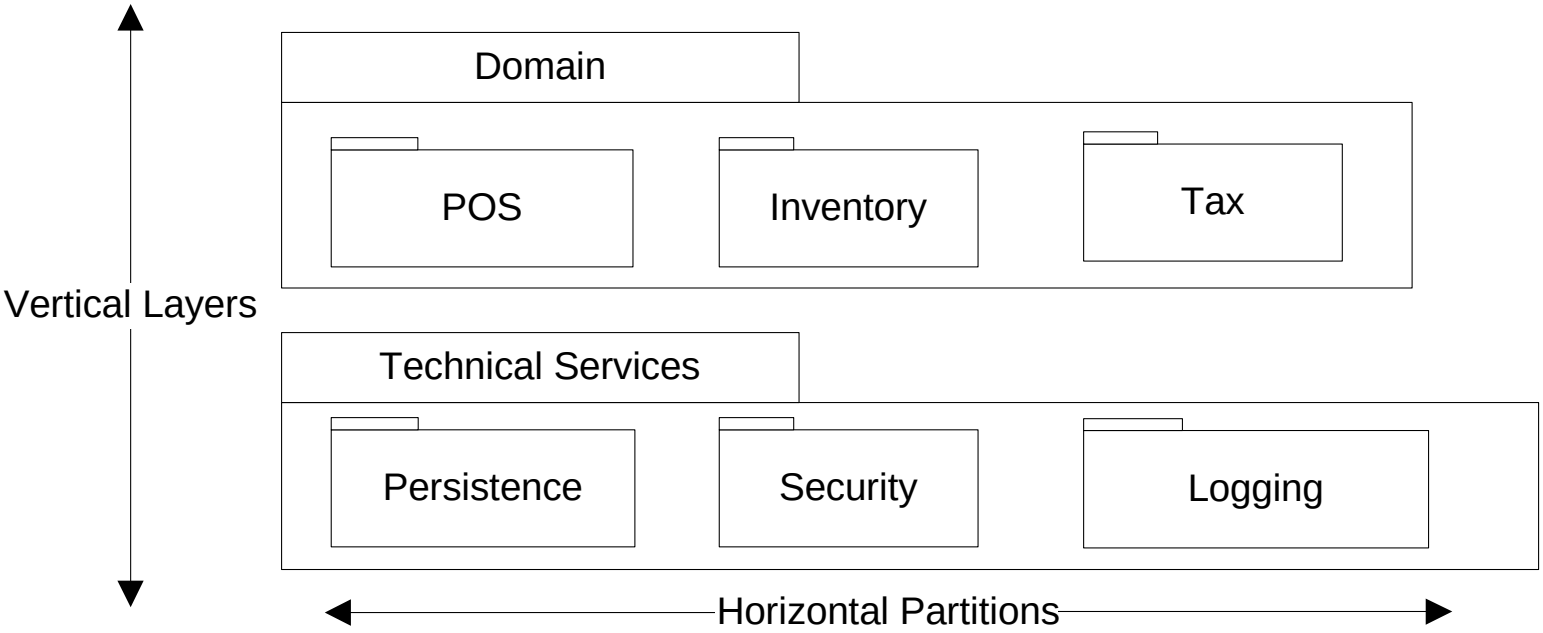
The layered architectural style /cont.

- Many parts of the system are highly coupled; Changes to code must ripple throughout the system.
- Application (business) logic is tangled with the user interface. Business logic cannot be reused with a different interface.

Logical view



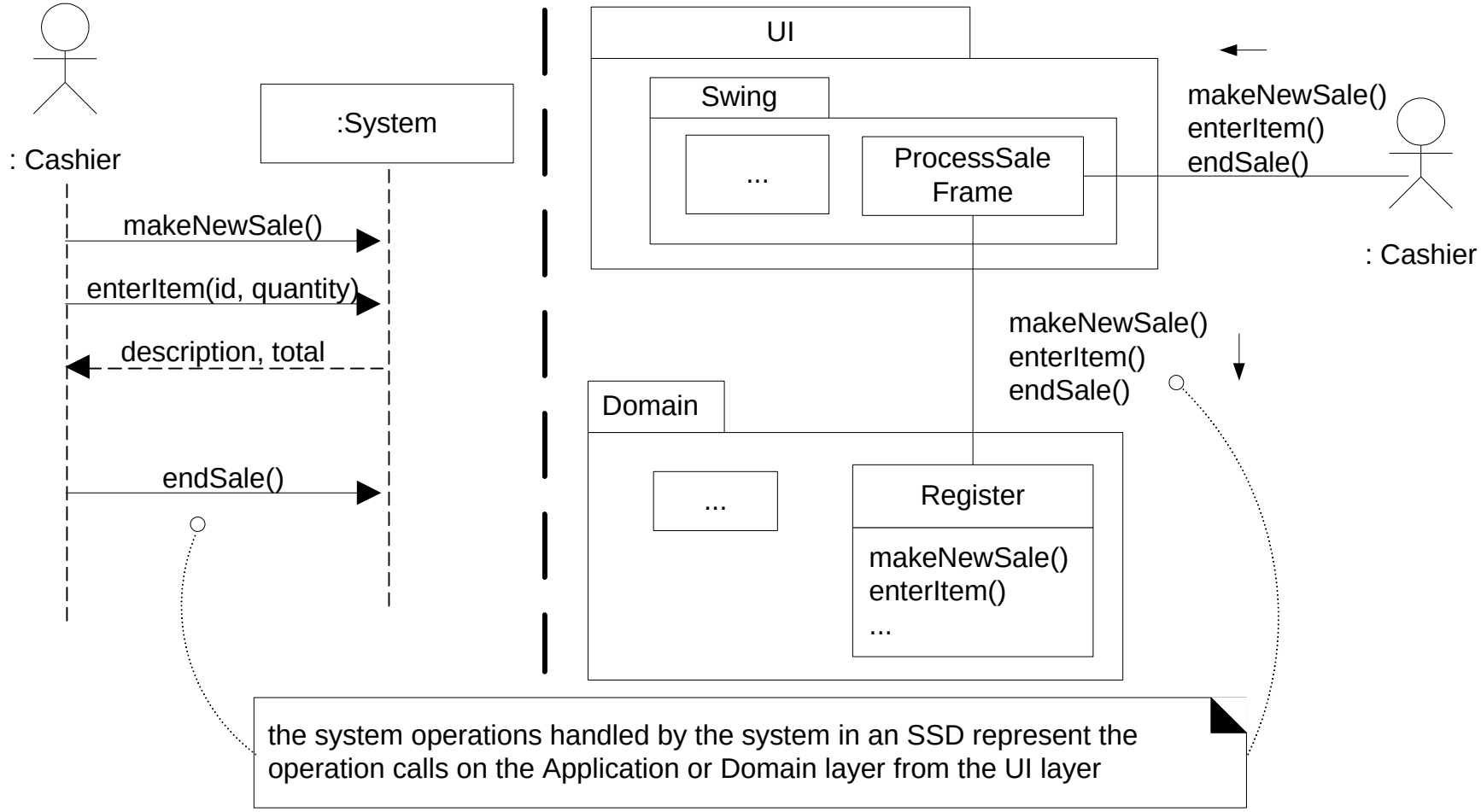
Vertical and horizontal partitions



- A package allows subdivisions.

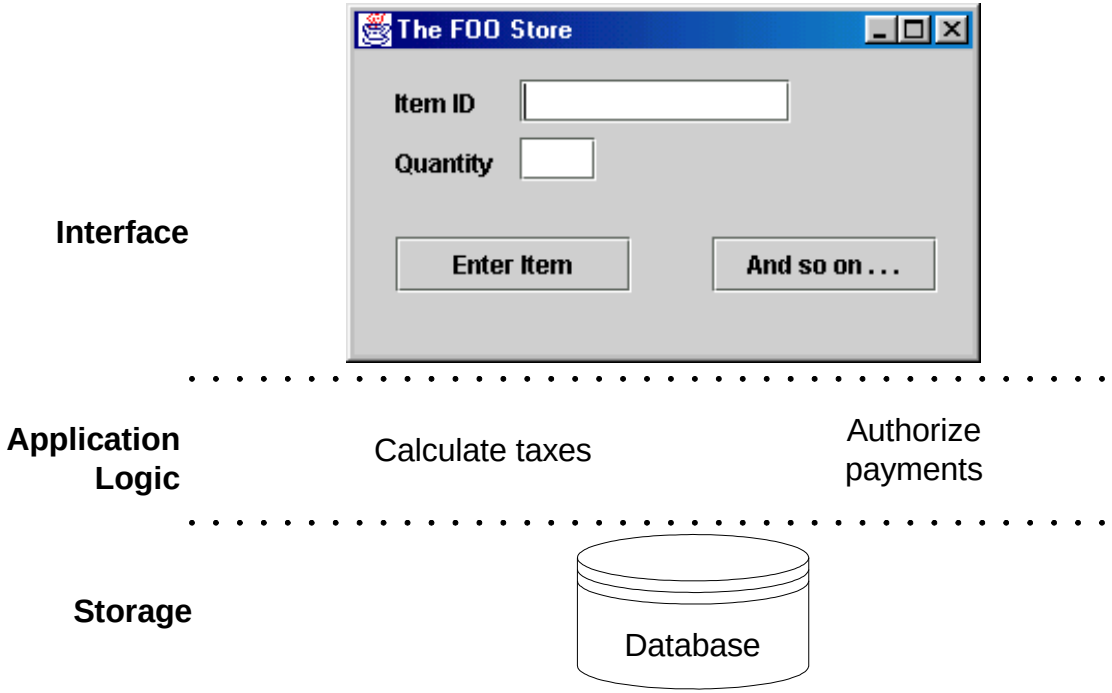
- Large and complex packages may be further divided into horizontal partitions.

Presentation layer and SSDs

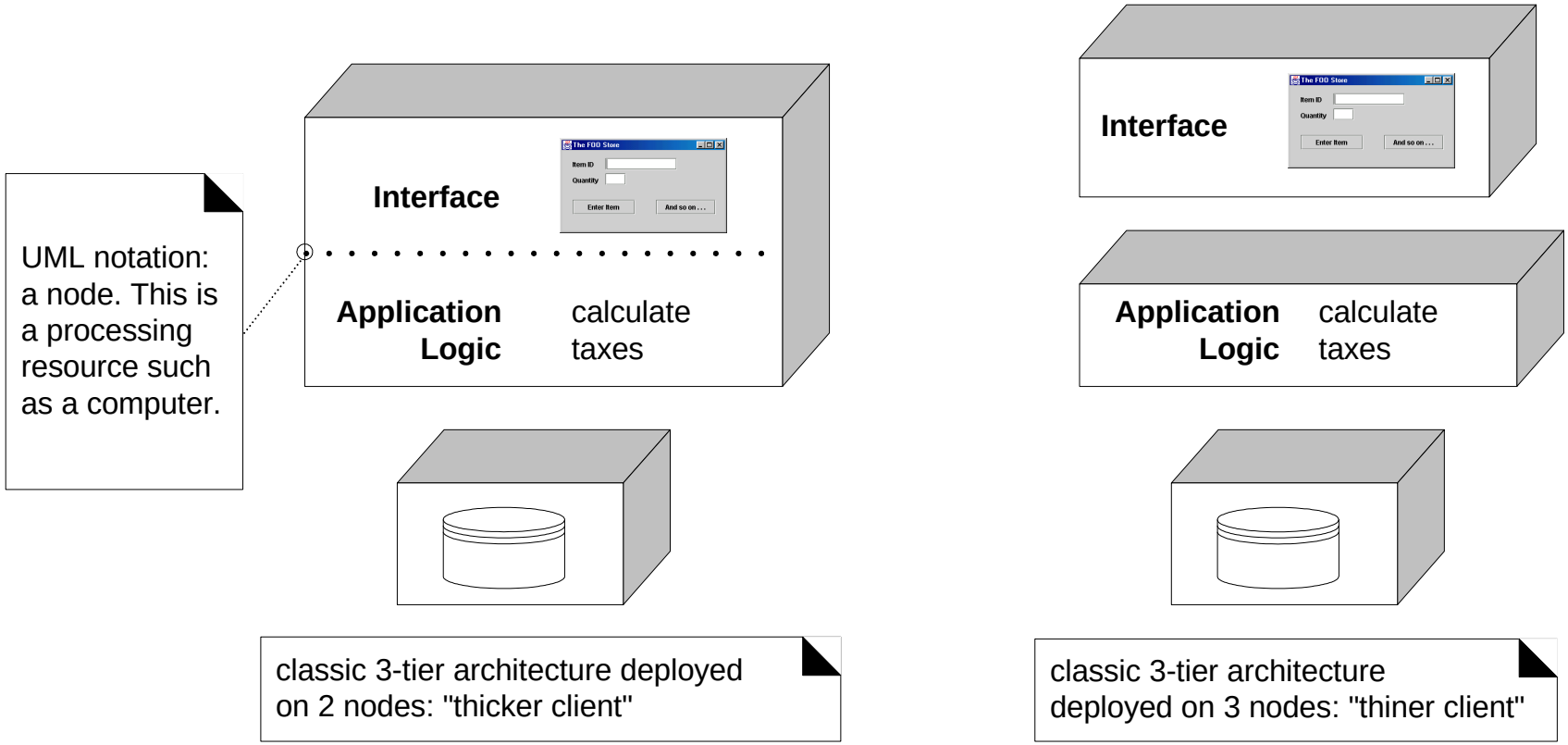


The 3-tier architectural style

- Layers define logical concepts whereas tiers define physical entities.
- The three layers are Interface, application logic (tasks and rules that govern the process) and storage (persistence).



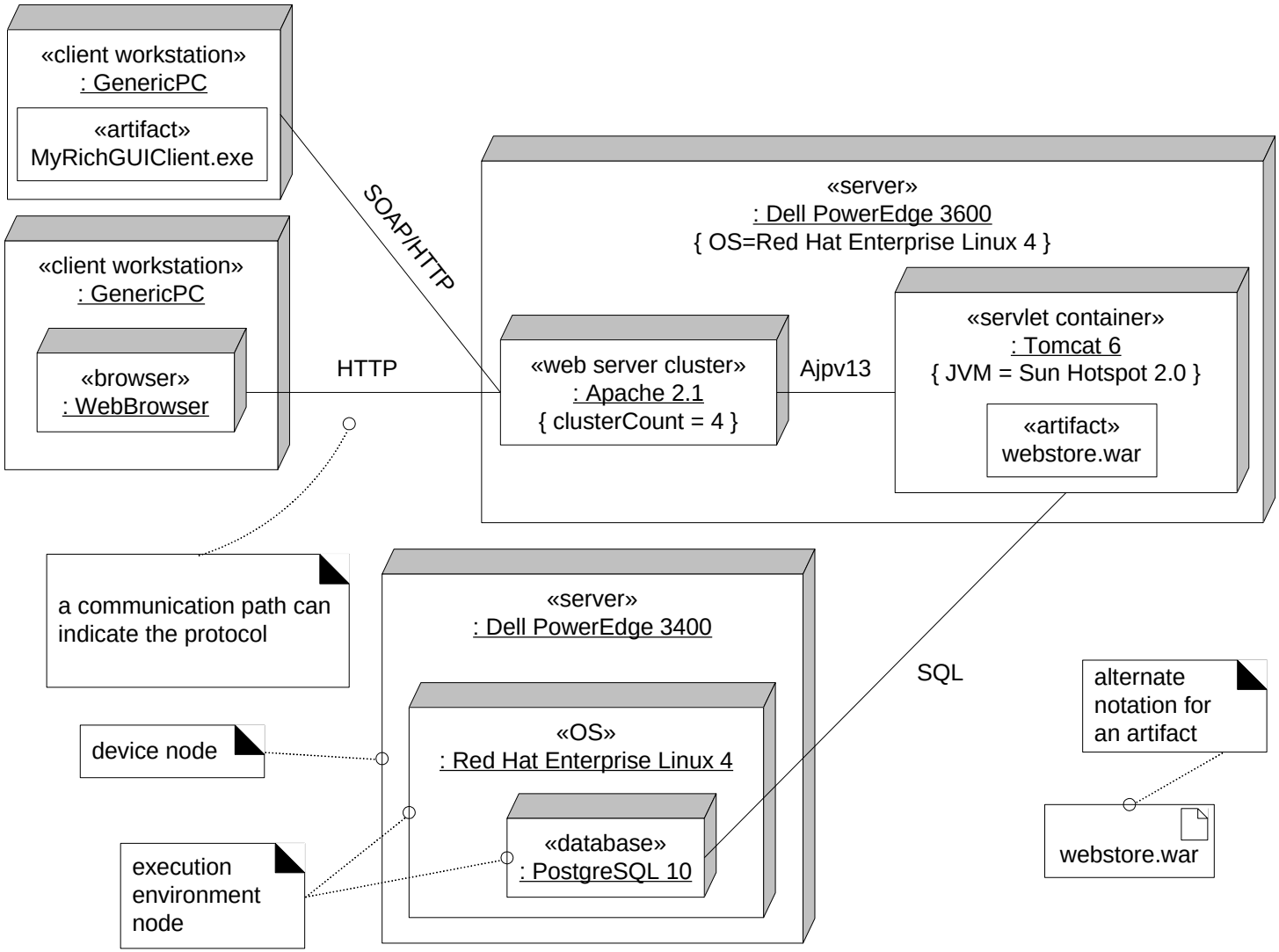
Physical (deployment) view of tiers captured by UML deployment diagrams



Here, the 3-layered architecture is developed over 3 tiers and 2 nodes.

Here, the 3-layered architecture is developed over 3 tiers and 3 nodes.

Physical (deployment) view captured by UML deployment diagrams /cont.



References

- Kruchten, Philippe, *”Architectural Blueprints — The “4+1” View Model of Software Architecture”,* IEEE Software 12 (6), pp. 42-50, November 1995.