Wyyzzk, Inc.

Creating a Project Architecture

Version 5.0

Wyyzzk, Inc.

Lesson Description

- This lesson describes how to create a project architecture.

Wyyzzk, Inc.

Lesson Goal

- Participants will understand how to create an architecture using a variety of common software architectural patterns, including layered, client/server, 3 tier, web, and object oriented.

Wyyzzk, Inc.

Lesson Objectives

- Upon completion of the lesson, the participant will be able to:
  - Describe the process of creating a project architecture
  - Describe several common software architectural patterns, including layered, client/server, 3 tier, web, and object oriented
  - Determine in which situation to use each pattern

Wyyzzk, Inc.

Lesson Outline

- Creating a Project Architecture
- Identify components
- Identify Architectural Patterns
- Assign responsibilities to components
- Complete the 4+1 Views

Wyyzzk, Inc.

Creating a Project Architecture

- 1. Identify components, bottom-up and/or top-down
- 2. Realize the architecturally significant use cases using components on sequence diagrams
- 3. Add relationships and operations to components
- 4. Create the physical architecture
- 5. Convert the subsystems into processes and threads on the hardware
- 6. Add data model, analysis model, policies, and mechanisms as appropriate
- 7. Document the architecture

Wyyzzk, Inc.

Identifying Components

- When creating a project architecture, one way to start is by identifying components
  - A component is any nearly independent part of the system that is internally cohesive
- Use commonality, variability, and kinds of change

Wyyzzk, Inc.

Commonality

- Group shared things into components
  - Libraries
  - Functions
  - Data

Wyyzzk, Inc.

Variability

- Put each variation in a component
  - Platform
  - Edition
  - Version

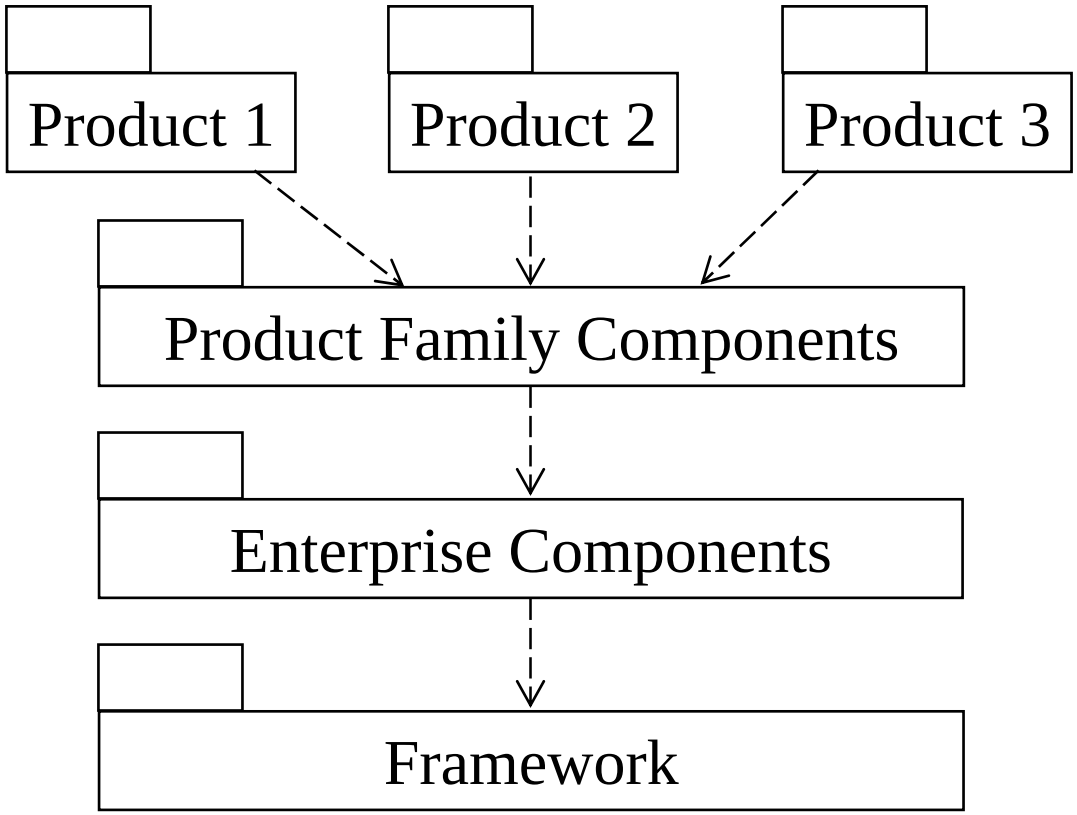
Wyyzzk, Inc.

Kinds of Change

- Identify kinds of change
  - Data
  - Features
  - Look and feel
- Group things that change together
  - Adding features – make component for each feature
  - Customizing UI – make component for UI

Wyyzzk, Inc.

Example: Commonality & Variability



Wyyzzk, Inc.

Subsystems

- A subsystem is a kind of a package with some additional semantics
  - Operations
    - a list of operations that specify the behavior of the subsystem
  - Specification
    - a set of use cases with their interfaces, constraints, and relationships that specify the behavior of the subsystem
  - Realization
    - described by nested subsystems and classes, along with their interfaces, constraints, and relationships
  - Collaborations describe the operations and use cases

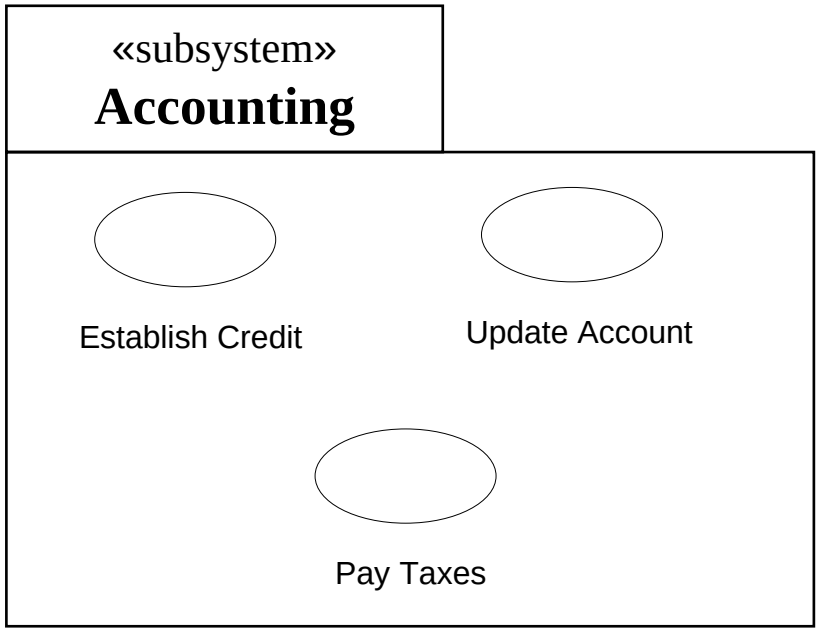
Wyyzzk, Inc.

Subsystems (cont.)

- A subsystem can implement one or more interfaces
- A subsystem can have constraints attached to it

Wyyzzk, Inc.

Subsystem Representation



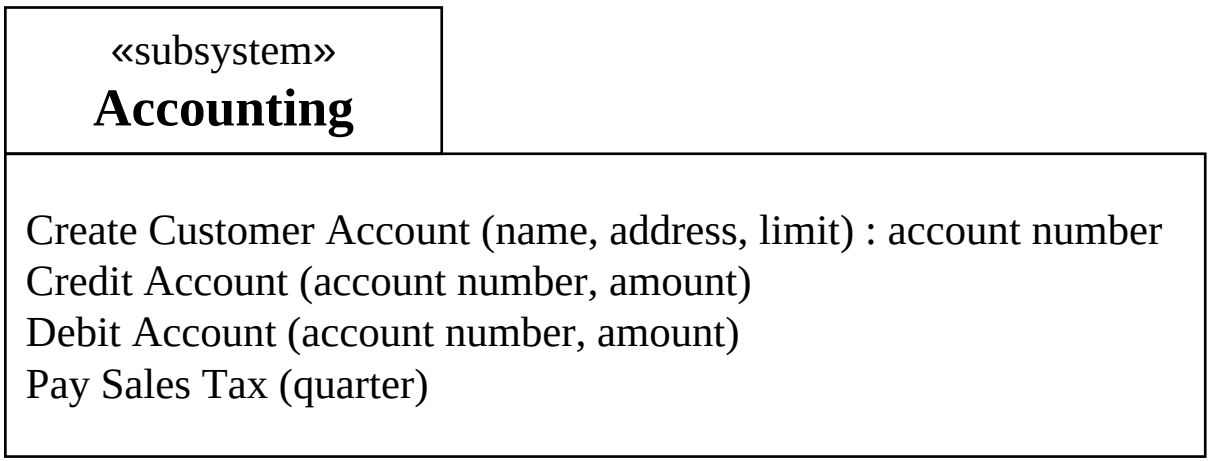
Package icon with subsystem stereotype

Specification of subsystem with use cases

Wyyzzk, Inc.

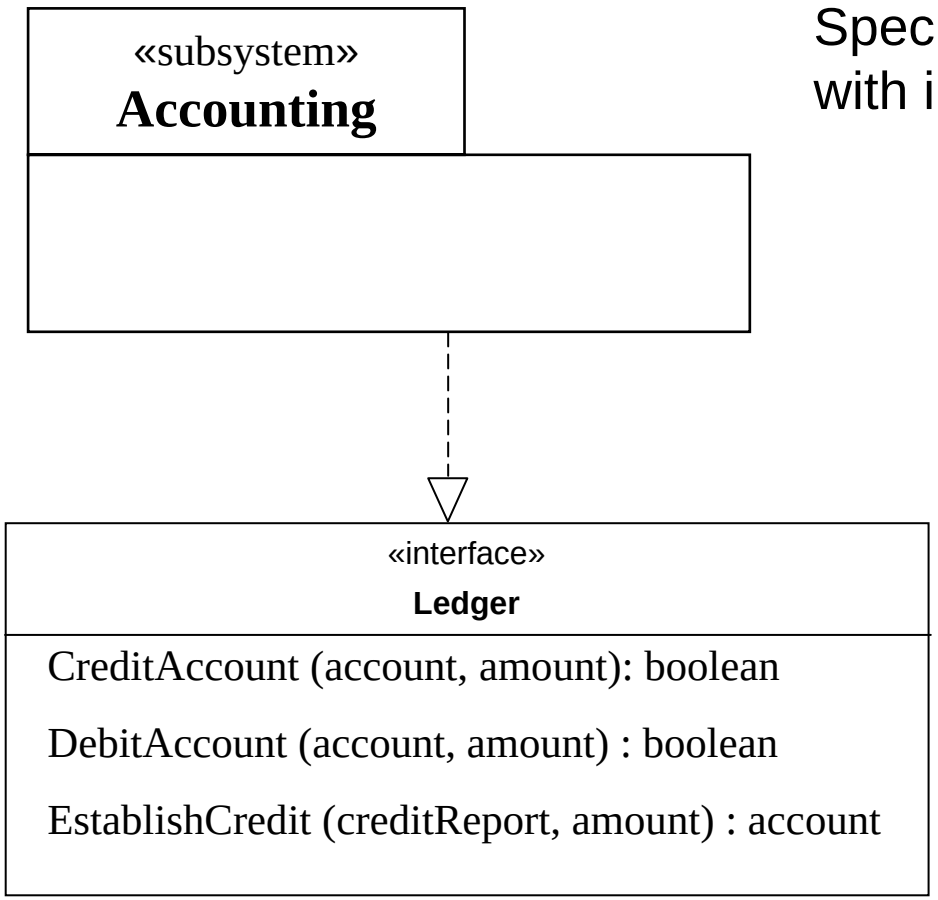
Subsystem Representation (cont.)

Subsystem operations



Wyyzzk, Inc.

Subsystem Representation (cont.)



Specification of subsystem with interface

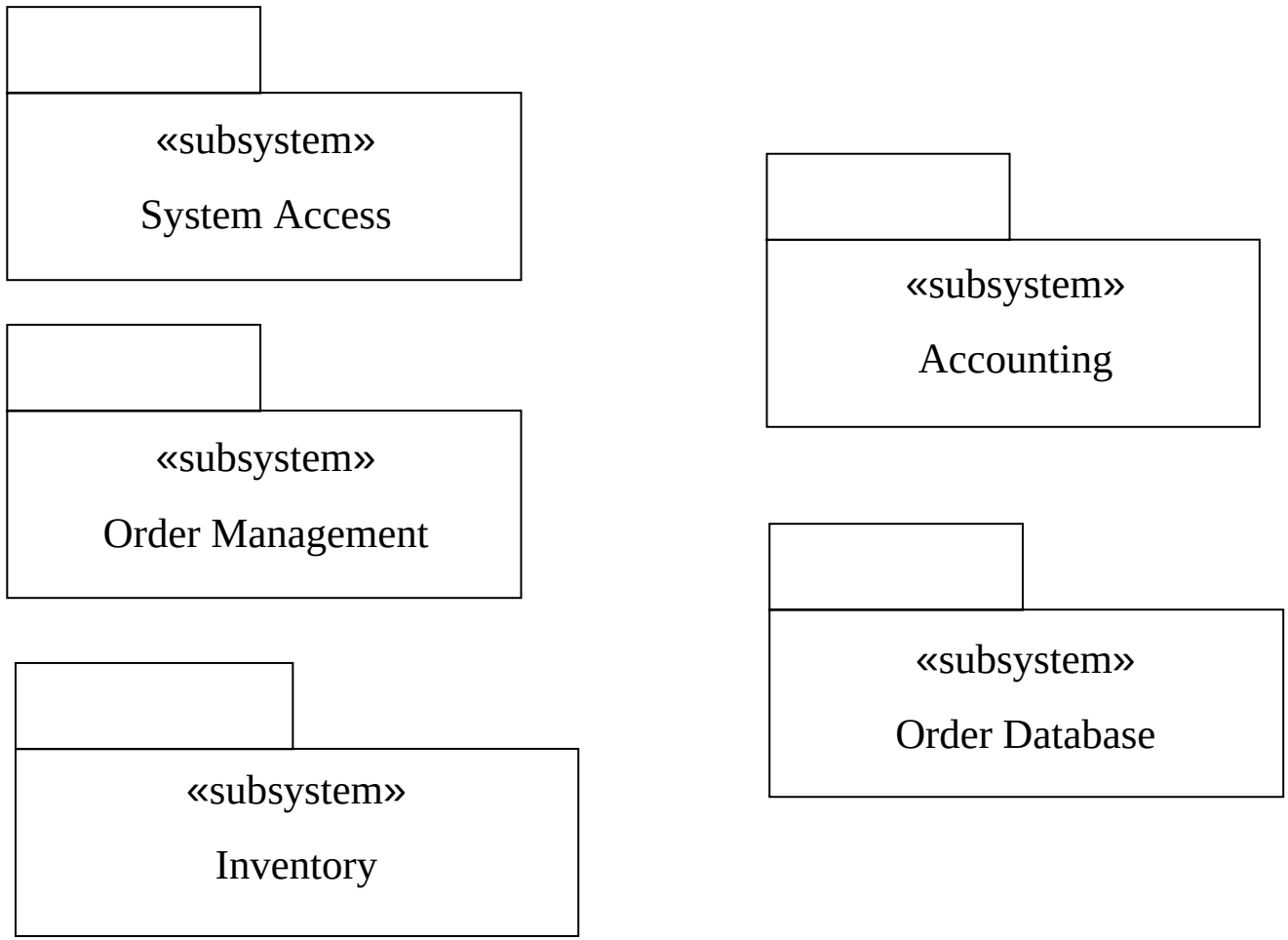
Wyyzzk, Inc.

Diagramming the Architecture

- Create a subsystem for each architectural component.
- Write a brief description of what the subsystem is responsible for.
- Assign responsibilities to the subsystems with use cases, operations, or interfaces.
- Use sequence diagrams to determine dependency relationships between subsystems and to create the interfaces.

Wyyzzk, Inc.

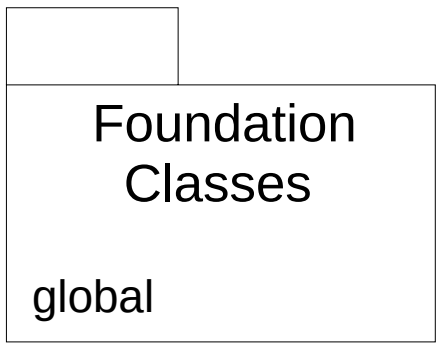
Create a Subsystem per component



Wyyzzk, Inc.

Global Packages

- Certain packages are used by all subsystems
  - Foundation classes
    - Sets, lists, queues, etc.
  - Error handling classes
- These packages are marked global



Wyyzzk, Inc.

Write a brief description of each subsystem

- System Access – This subsystem controls who can access the system
- Order Management - This subsystem knows about orders and all the functions associated with orders.
- Inventory - This subsystem knows about products and interfaces to the inventory control system.
- Accounting - This subsystem knows about accounts and interfaces to the accounting system.
- Order Database - This subsystem knows how to make information persistent and how to retrieve that information later.

Wyyzzk, Inc.

Creating an Architecture from Patterns

- Other engineering disciplines use a general set of steps to develop an architecture
  - Step 1: Select the basic architecture
    - includes the components of the architecture, the basic responsibilities of the components, and the basic relationships between the components
  - Step 2: Organize the key abstractions of the application into the components of the architecture
  - Step 3: Develop the interactions between the components

Wyyzzk, Inc.

Building Example

- Step 1: Select the basic architecture
  - Turreted mansion
- Step 2: Organize the key abstractions of the application into the components of the architecture
  - West wing
    - Guest quarters, Guest bath
  - East Wing
    - Family quarters, Kids bath, Master bath, Master bedroom
  - North Turret
    - Gallery

Building Example (cont.)

- Step 3: Develop the interactions between the components
  - The wings and turrets will all have halls connecting them to the central court
  - The kitchen will adjoin the dining room
  - etc.

Wyyzzk, Inc.

Apply the steps

- These same steps can be applied to any software application
- You may try more than one pattern for your application before finding the best fit
- Note the alternatives in your architecture document and why you rejected them
  - Changes in requirements may make one of these alternatives feasible later
  - Or one of the alternatives might be useful for another product in the same line of business

Apply the Steps (cont.)

- Having selected a pattern, assign responsibilities to the components of the architectural pattern you selected
- Then define the interfaces between the components

Wyyzzk, Inc.

Architectural Patterns

- When determining the architecture for a system, it is convenient to review a variety of architectural patterns for possible fit
- An architectural pattern has been created to solve a particular kind of problem.
- The architectural pattern gives us the basic structure, which we put our application into.

Wyyzzk, Inc.

Architectural Patterns

- An Architectural Pattern expresses the fundamental organization of a software system. It provides:
  - A pre-defined set of subsystems
  - Responsibilities for each subsystem
  - Rules and guidelines for organizing associations and interactions between the subsystems

Wyyzzk, Inc.

Example Architectural Patterns

- There are several well known architectural patterns for software.
  - We will examine 5 common types in this class:
    - Layered
    - client/server
    - 3 tier
    - web
    - object oriented
  - Some other common types we won't cover:
    - pipe and filter, stovepipe, MVC, blackboard, publish and subscribe

Architectural Patterns (cont.)

- Each of these architectural patterns has strengths and weaknesses
  - For any project, there will be a set of patterns that works well for that application, and another set that is not a good fit
  - The architecture that you select depends on:
    - the nature of the project,
    - the expected growth path of the application
    - the priorities established for the project
- Architecture can mean the hardware, software, or both. We are only considering the software for now.

Wyyzzk, Inc.

Some Architectural Patterns

- In the rest of this section, we'll look at some common architectural patterns
  - Common uses
  - The components
  - The responsibilities of the components
  - The interactions between components
  - Examples
  - Strengths and weaknesses of the pattern

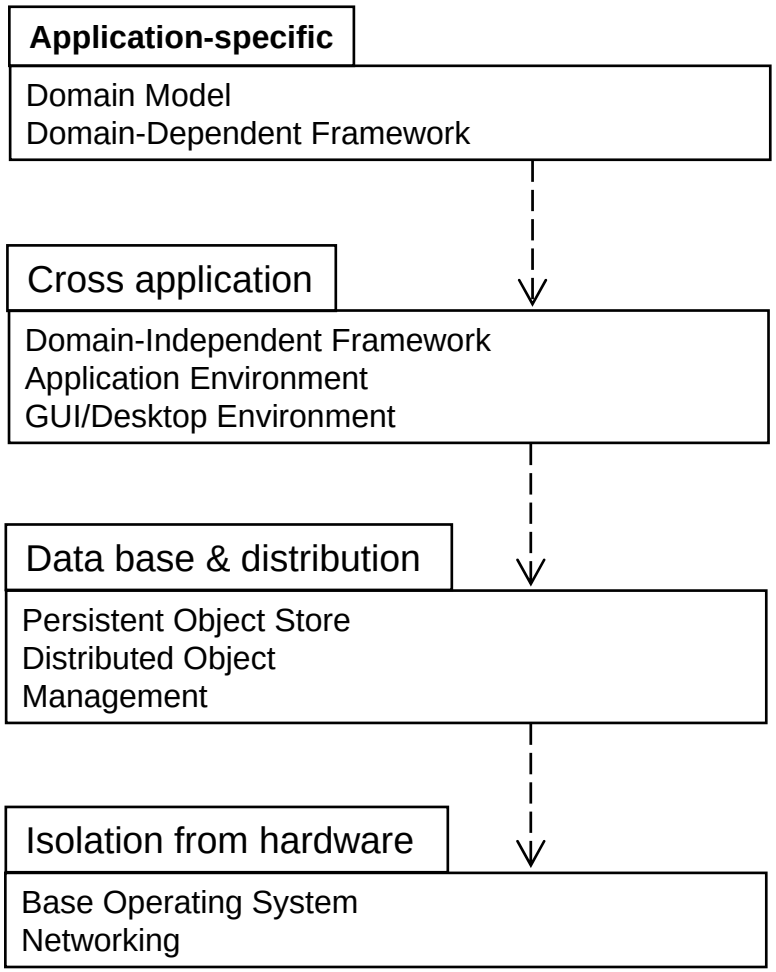
Wyyzzk, Inc.

Layered Architecture

- Common Uses
  - Operating Systems, Network software, Frameworks
- Components
  - Layers and interfaces
- Responsibilities
  - Each layer contains functionality at the same level of abstraction
  - Each layer has an interface which defines the services provided by the layer
- Interactions
  - Each layer only interacts with the layer below it

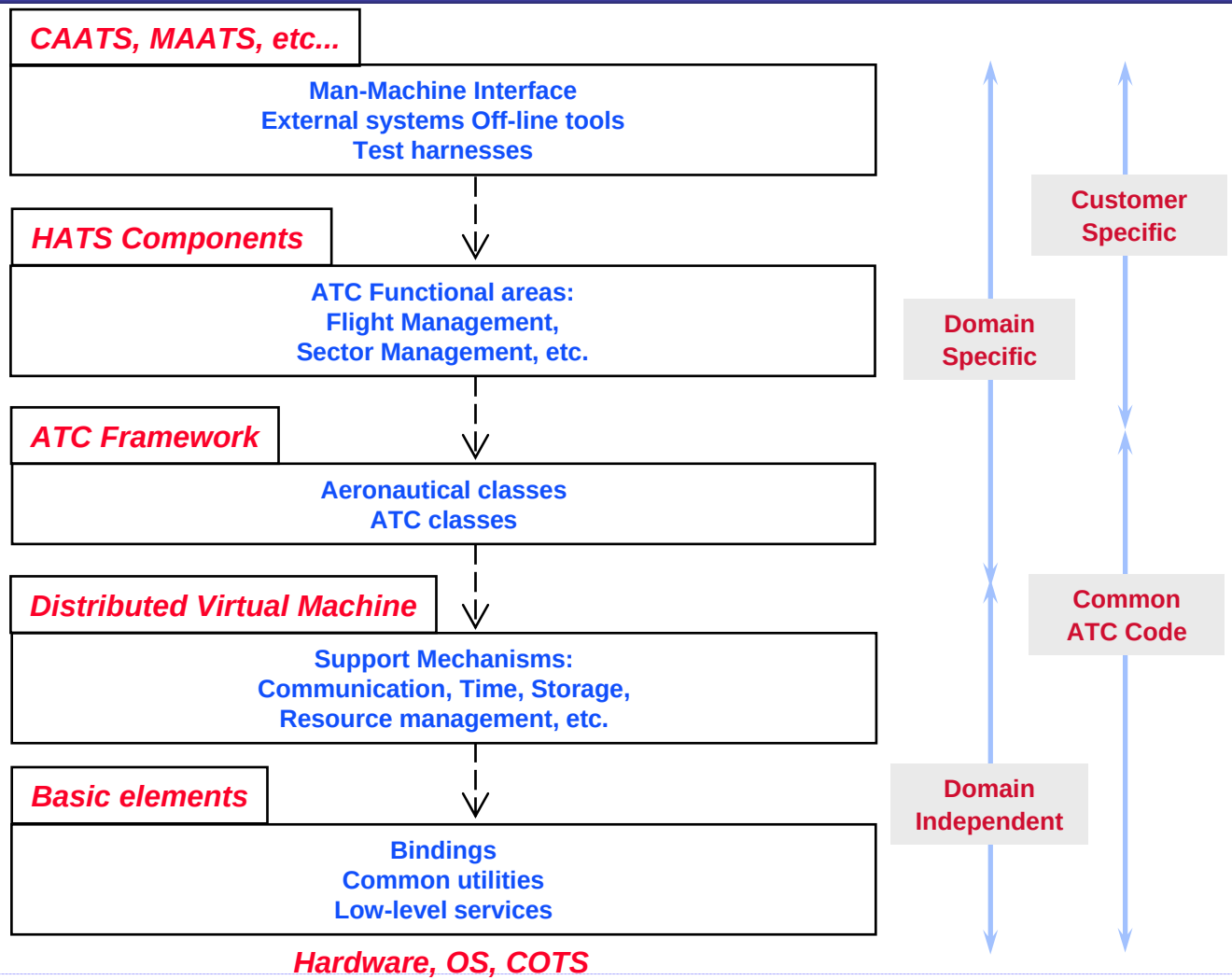
Wyyzzk, Inc.

Example: General Layered Architecture



Wyyzzk, Inc.

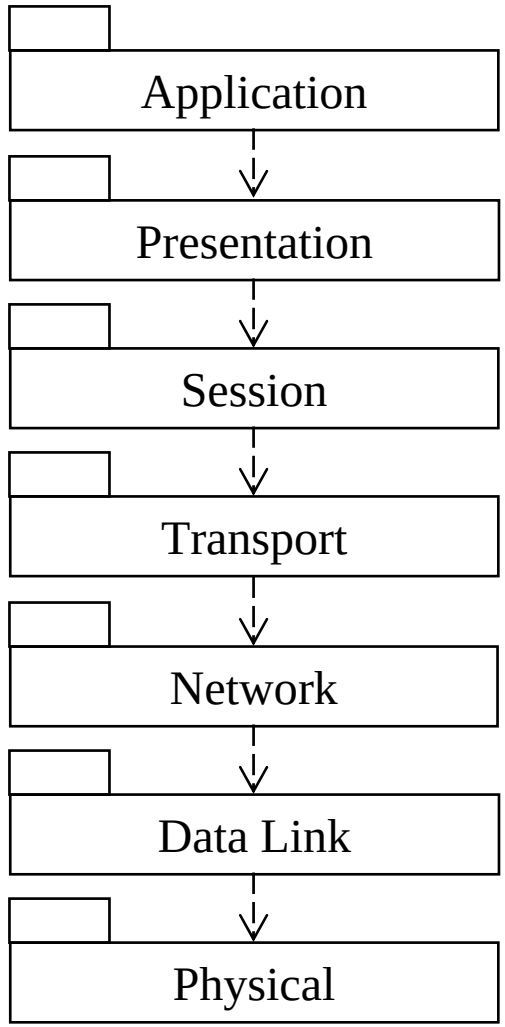
Example: Air Traffic Control System



Source: ACM

Example: Network Protocols

The International Standards Organization defined a 7 layer model for network communications.



Wyyzzk, Inc.

Strengths & Weaknesses of Layered Architecture

- Strengths
  - Changes can generally be accomplished within a single layer
  - Portability, maintenance, upgrades, etc. are easier since they usually only require replacing a single layer
- Weaknesses
  - Execution speed may be slower due to the indirection caused by having each level process (or relay) the command

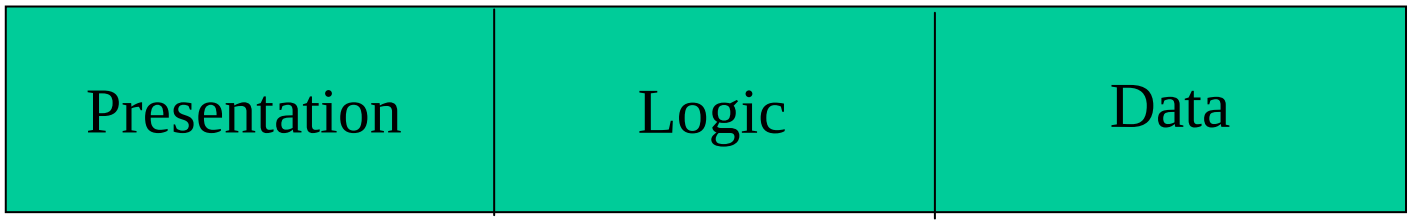
Wyyzzk, Inc.

Client / Server Architecture

- Common Uses
  - Business software
- Components
  - Client and Server
- Responsibilities
  - Client is the presentation software
  - Server provides the rules and data store
- Interactions
  - Client gets data from users and passes it to server
  - Server sends results back to client to display

Categories of Software

- In most software projects, there are 3 major categories of software, which are represented in the graph below.

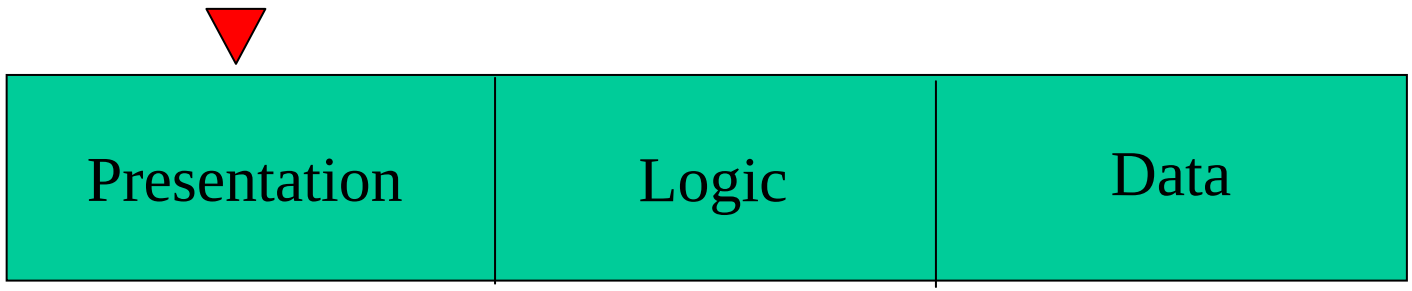


- In a client/server system, we split the software apart somewhere in the continuum
  - Part of the software goes into a client process, the rest goes into a server process

Wyyzzk, Inc.

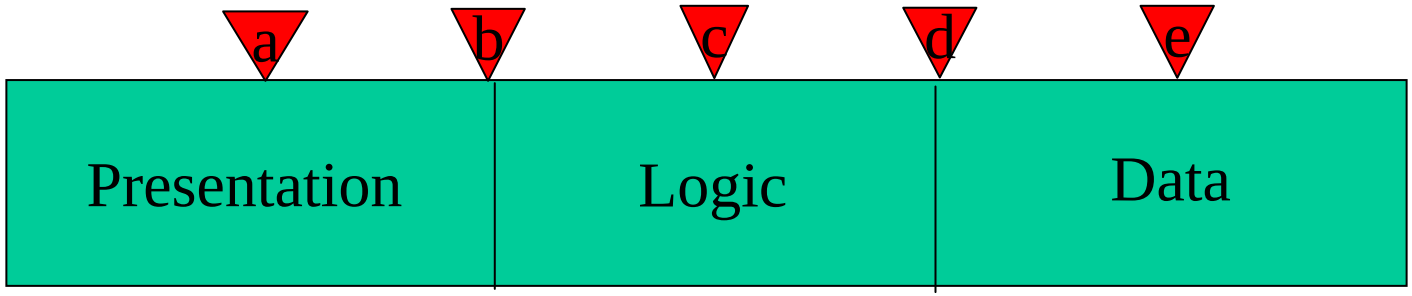
Client/Server vs Mainframe

- A mainframe system is a kind of client/server architecture.
- On a terminal/mainframe configuration, the software is divided between the terminal and the mainframe at the point indicated.



Client/Server vs Mainframe

- In client/server, the software can be divided at any point. The most common dividing points are shown below.



  - At point A, client/server works just like mainframe.
  - Point B is a more standard client/server configuration.

Wyyzzk, Inc.

How to split the software?

- Deciding where to split the software between the client machine and the server machine depends on a couple of issues:
  - The processing power of the client and the server
  - The amount of software that can be shared by multiple users
  - The desired execution speed of the application

Wyyzzk, Inc.

Strengths & Weaknesses of Client/Server

- Strengths
  - A lot of flexibility in where to divide the software between processes
  - Often the client just handles the presentation tier so it's easy to change the look and feel of the application
  - Relatively simple to code
- Weaknesses
  - Possibility of very slow execution speed due to volume of transactions between processes
  - This can be mitigated by moving processing to the client, but that in turn makes distribution of upgrades and maintenance of the client more difficult.

Wyyzzk, Inc.

3 Tier Architecture

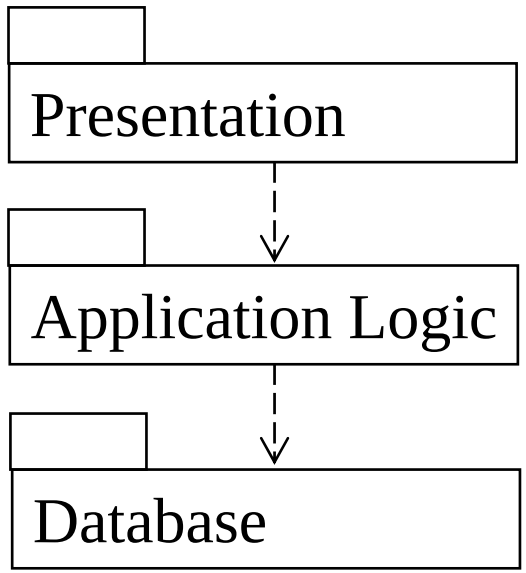
- Common Uses
  - Business applications
- Components
  - Presentation, Application, Database
- Responsibilities
  - Presentation - the user interface
  - Application - the business rules
  - Database - storage and retrieval of persistent data
- Interactions
  - The Presentation tier only interacts with the Application tier
  - The Application tier only interacts with the database

Wyyzzk, Inc.

Business Applications

Most business applications are built with a 3 tier architecture

- The presentation tier allows the user to view results and / or the information in specific business objects
- The application tier contains the rules for manipulating the business objects
- The database tier contains the business objects



Wyyzzk, Inc.

Strengths and Weaknesses of 3 Tier Architecture

- Strengths
  - Since the presentation layer is separated from the database by the application tier it is easy to change the look and feel or the database with relatively minor effects on the rest of the application
  - Leads to a consistent user interface across the whole application
- Weaknesses
  - Usually requires some kind of transaction management service to track transactions from presentation tier to database
  - Changes in functionality typically require changes to all 3 tiers of the architecture

Wyyzzk, Inc.

Web Architecture

- Common Uses
  - Business software
- Components
  - Web Client and Web Server
- Responsibilities
  - Client is the presentation software
  - Server provides the rules and data store
- Interactions
  - Client gets data from users and passes it to server
  - Server sends results back to client to display

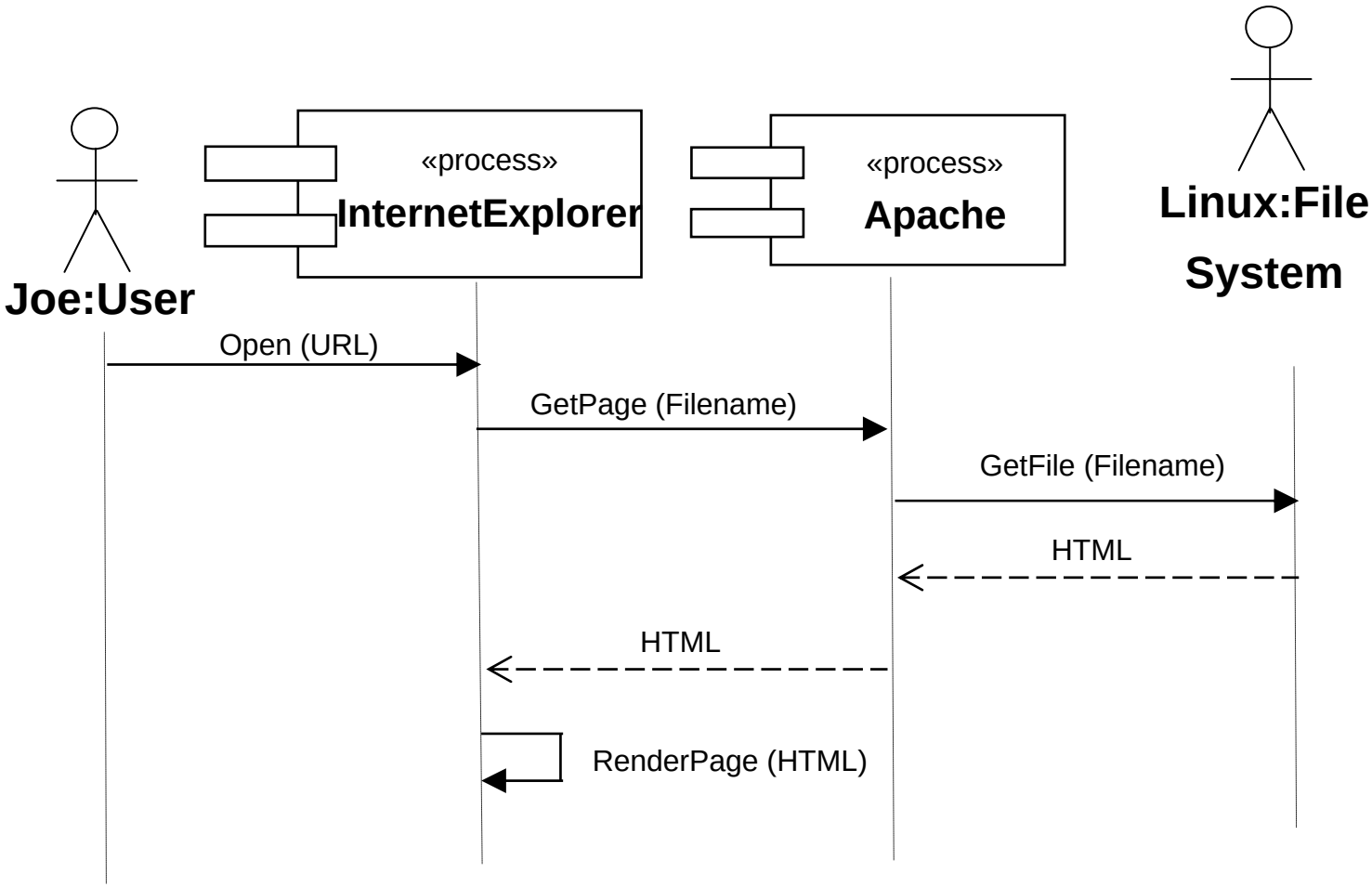
Wyyzzk, Inc.

Basic Web Architecture

- The basic web architecture is client/server.
  - A web browser runs on the client
    - Internet Explorer
    - Netscape
  - A web server runs on the server
    - IIS
    - Apache
  - The client and server communicate using the http protocol

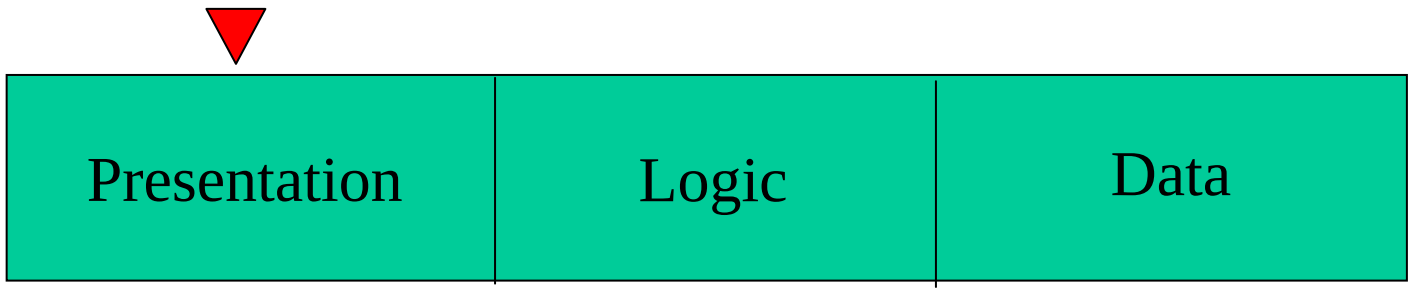
Wyyzzk, Inc.

Basic Web Architecture



Web vs Mainframe

- The web is very much like a mainframe architecture
  - A thin, stateless client
  - A server that does all the processing



Wyyzzk, Inc.

Strengths & Weaknesses of Web

- Strengths
  - Since the client just handles the presentation tier so it's easy to change the look and feel of the application
  - Relatively simple to code
  - Very easy to supply updates of client side software
  - Very secure if web client does no processing
- Weaknesses
  - Possibility of very slow execution speed due to volume of transactions between processes
  - This can be mitigated by moving processing to the client, but that in turn makes distribution of upgrades and maintenance of the client more difficult and makes the application less secure

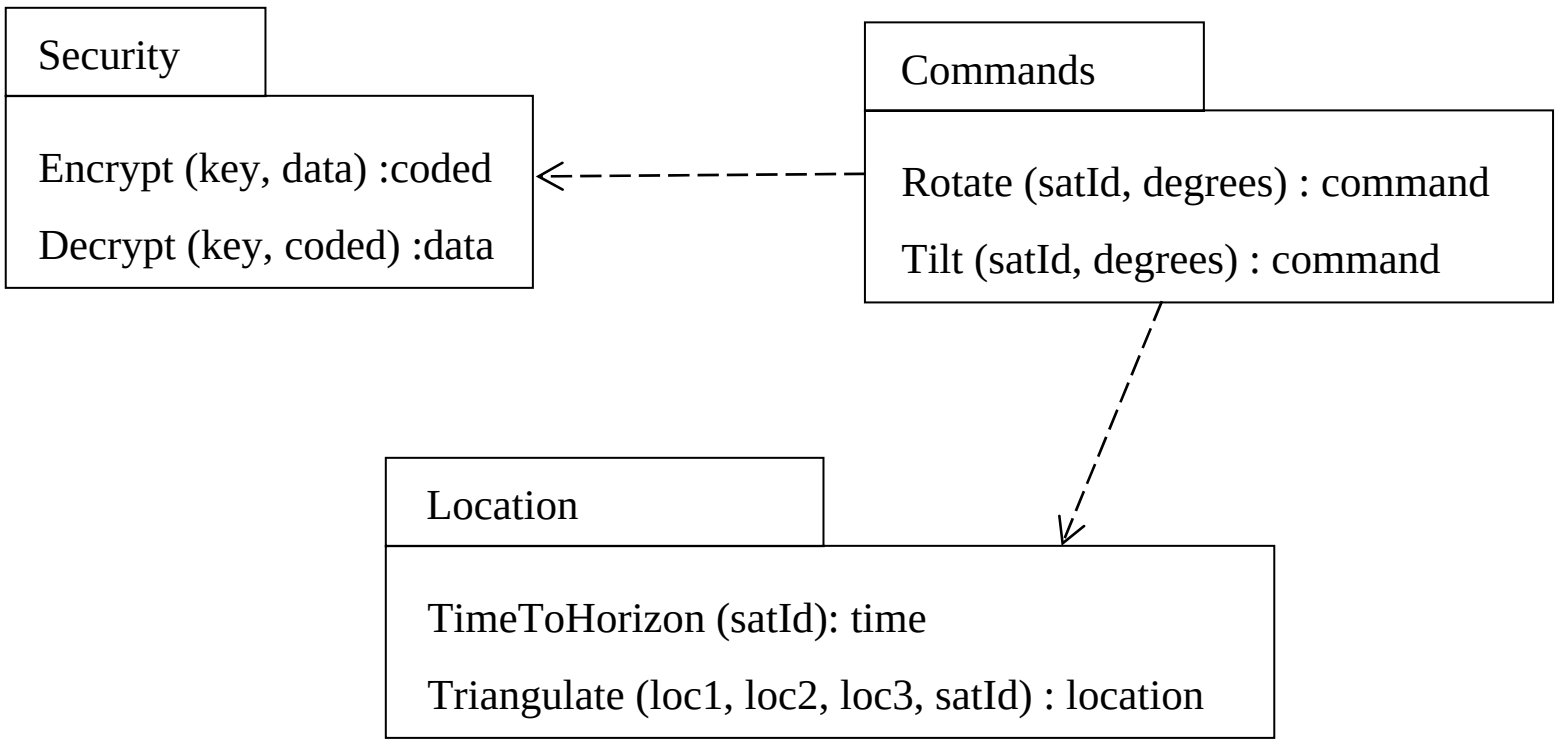
Wyyzzk, Inc.

OO Architecture

- Common Uses
  - Application software
- Components
  - Each component is created around a major piece of data and the associated functionality
- Responsibilities
  - Each component is responsible for providing access, create, update, delete functionality for its data
- Interactions
  - Minimal interactions between components

Wyyzzk, Inc.

OO Example



Wyyzzk, Inc.

Strengths and Weaknesses of OO Architecture

- Strengths
  - Since data is encapsulated, changes to data or implementation of the functions are localized
- Weaknesses
  - Changes to system level functionality (use cases) tend to be spread over a large part of the application

Wyyzzk, Inc.

Break

- Up to now, all we have done is identify some components for the architecture
- Now we refine the components

Wyyzzk, Inc.

Responsibilities of Components

- Once a system is divided into components, each component needs responsibilities
  - Responsibilities come from the requirements
- All use cases and requirements must be implemented by some component of your system

Wyyzzk, Inc.

Use Case Realizations

- A use case realization is a sequence diagram for a use case
- Create it by making a sequence diagram using your components for objects
- The messages are the sentences from the use cases

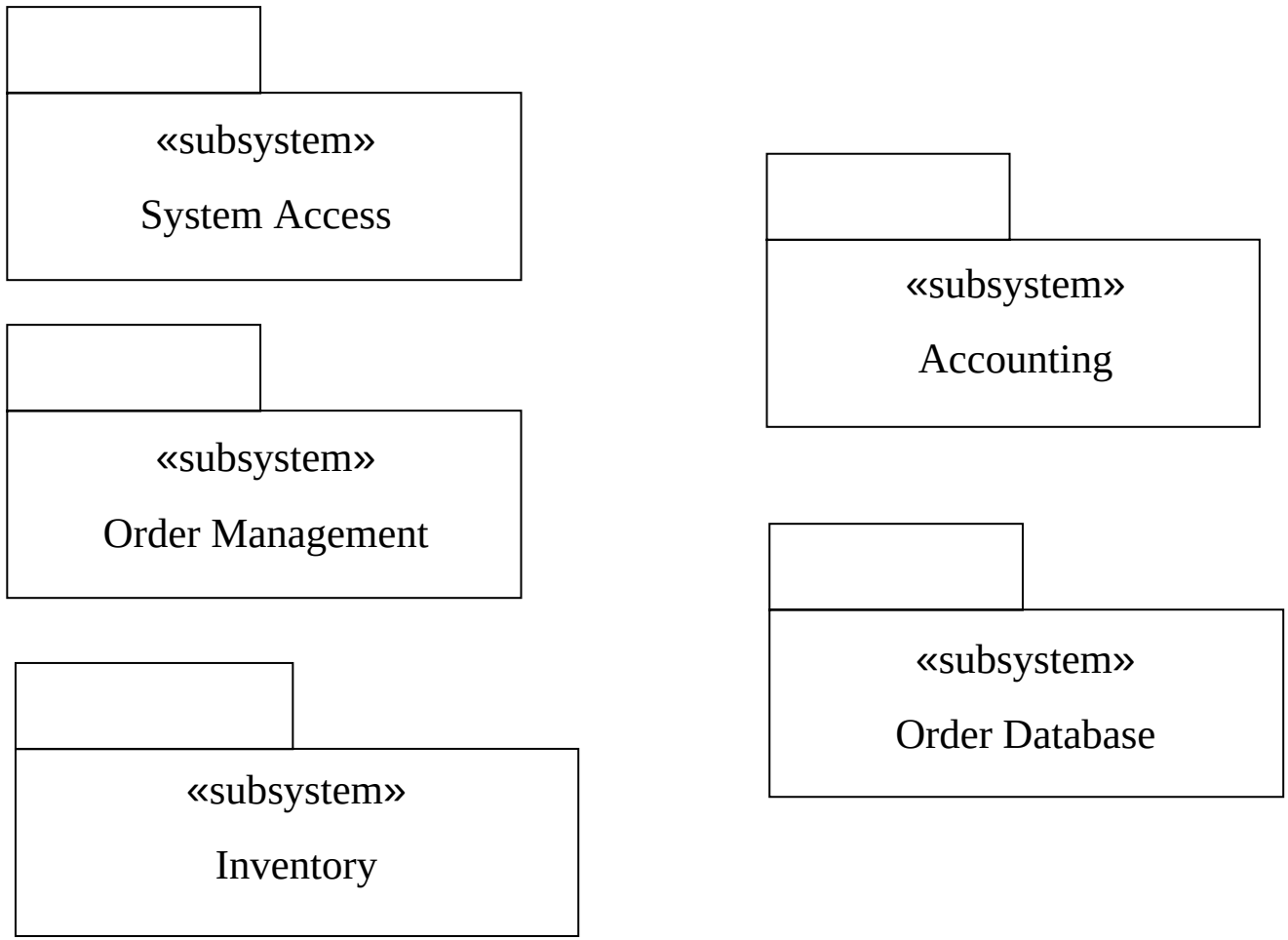
Wyyzzk, Inc.

Place Order Use Case

- 1. The customer logs into the system
- 2. The system displays a main screen
- 3. The customer selects to place an order
- 4. The system displays an order form
- 5. The customer enters his or her name and address
- 6. For each product the customer wishes to order
  - a. The customer enters a product code
  - b. The system gets the product description and price
  - c. The system adds the price to the total
  - d. The system displays the product description and price, and the order total on the order form
- End
- 7. The customer enters payment information
- 8. The customer submits the order to the system
- 9. The system verifies that the order is complete and correct
- 10. The system saves the order as pending
- 11. The system processes the payment
- 12. The system updates the order status to confirmed
- 13. The system displays a confirmation screen with the order id

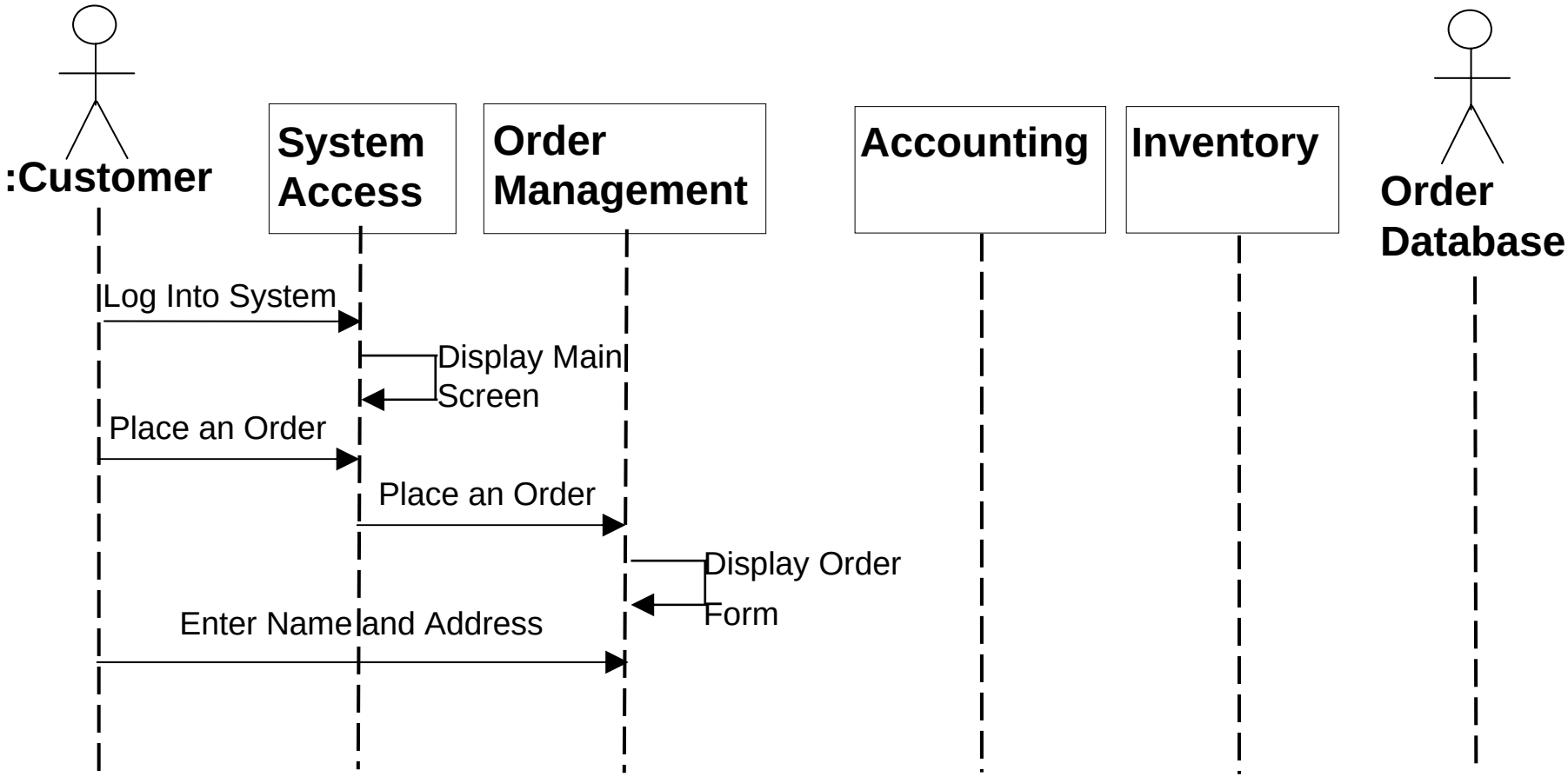
Wyyzzk, Inc.

Available Components



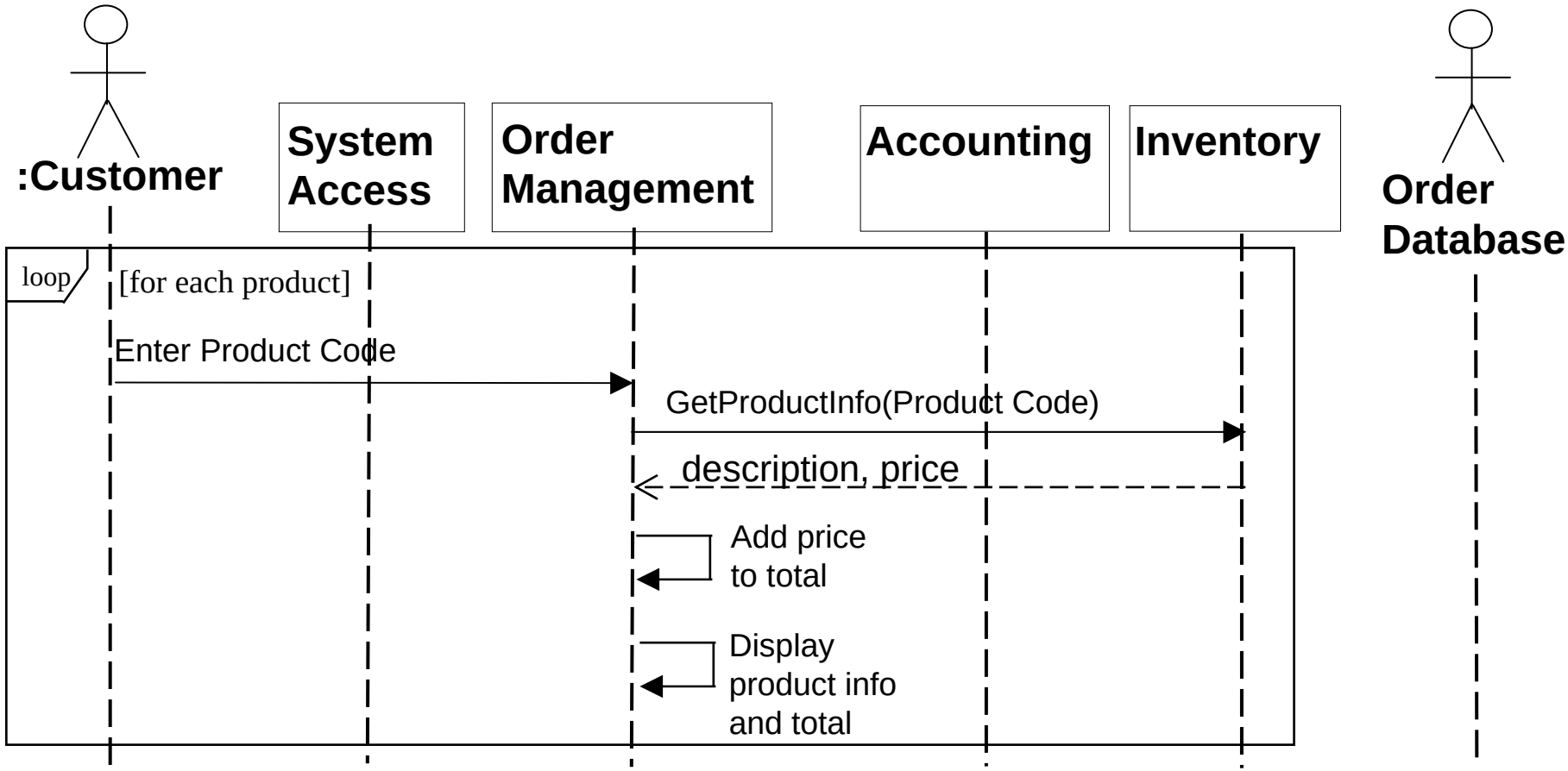
Wyyzzk, Inc.

Example Use Case Realization



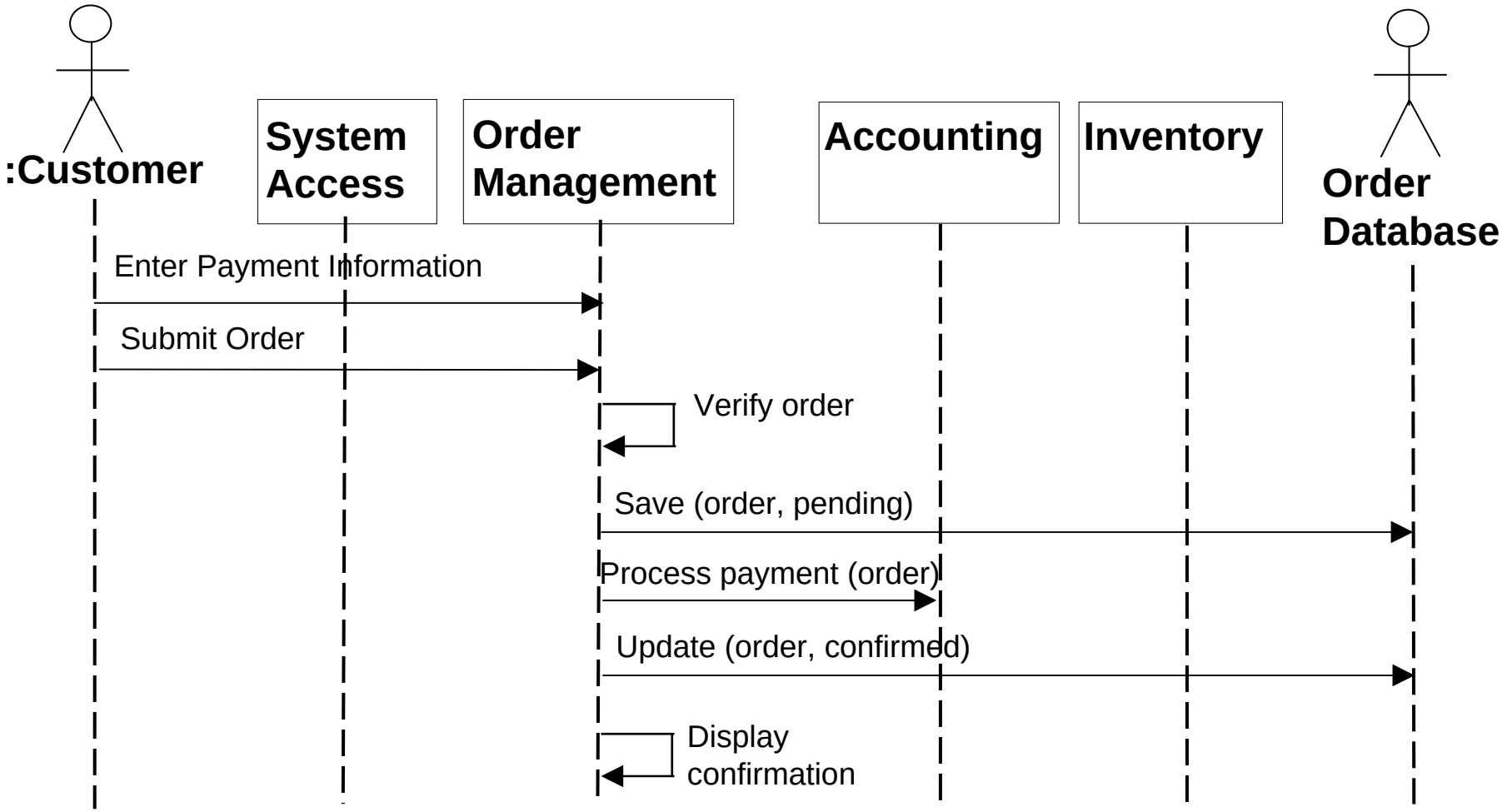
Wyyzzk, Inc.

Example Use Case Realization



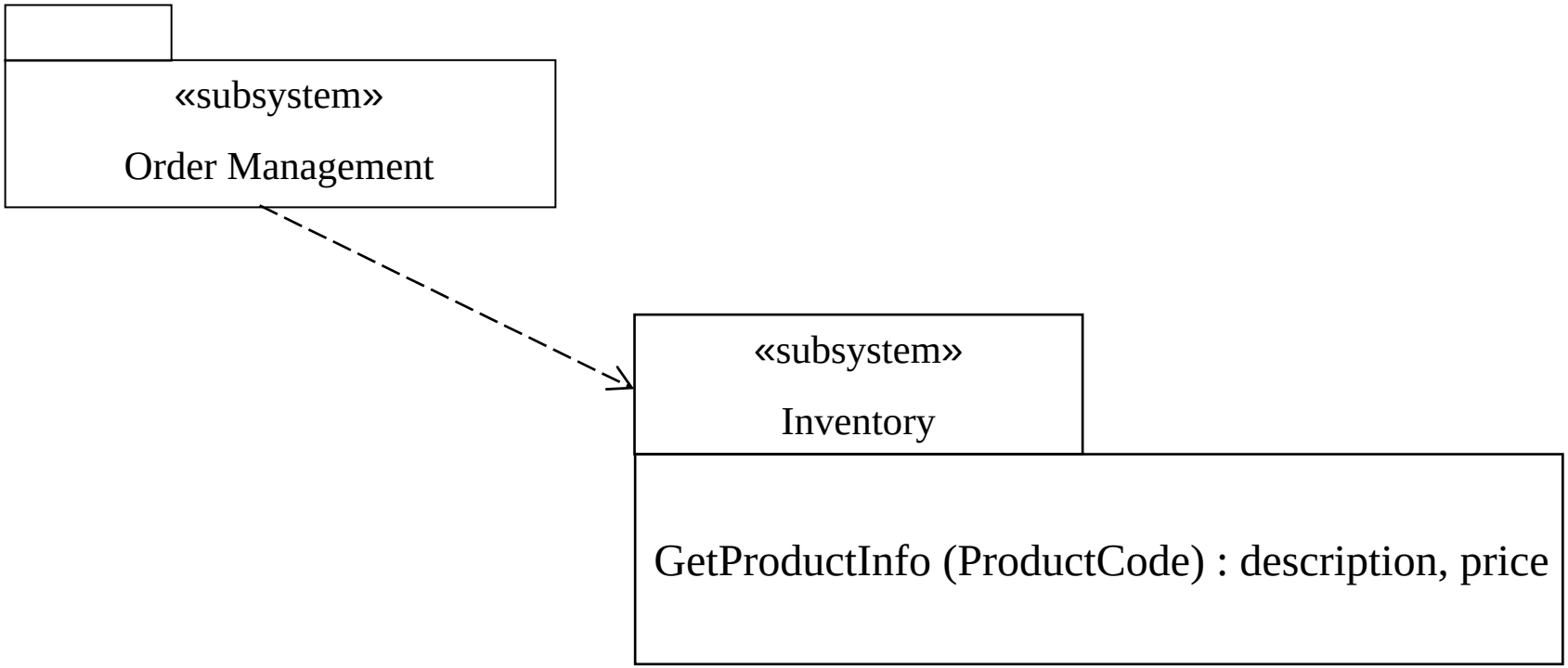
Wyyzzk, Inc.

Example Use Case Realization



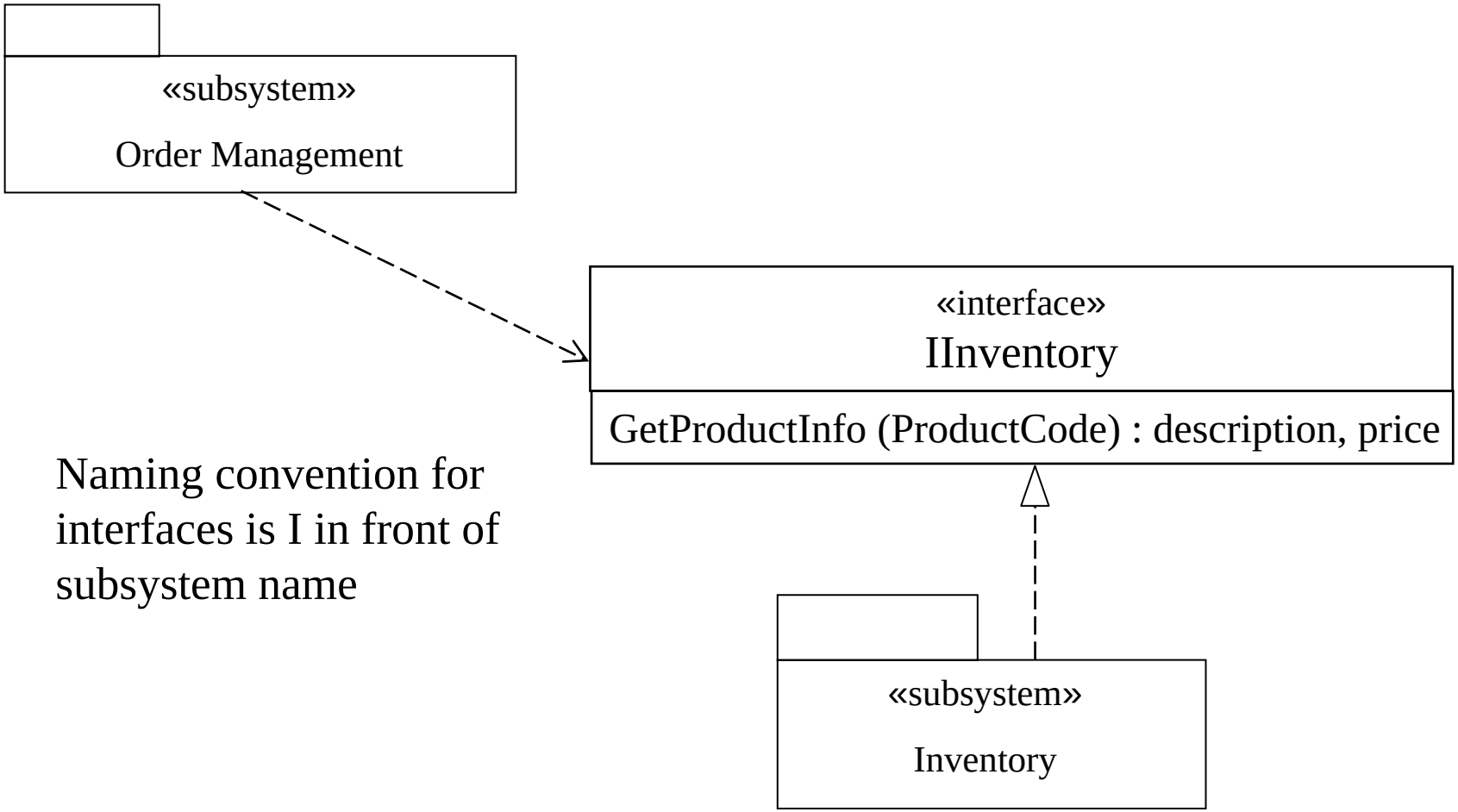
Wyyzzk, Inc.

Component Relationships



Wyyzzk, Inc.

Interfaces



Naming convention for interfaces is I in front of subsystem name

Wyyzzk, Inc.

The other views

- Physical
- Process
- Development

Wyyzzk, Inc.

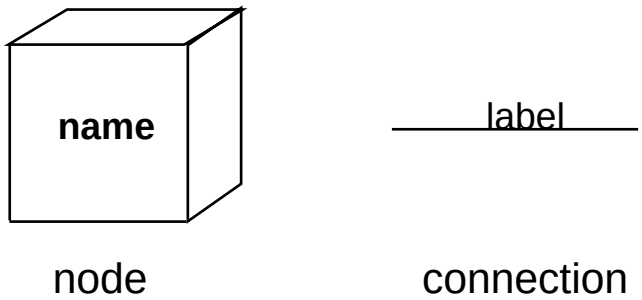
Physical View

- The physical view of an architecture shows the configuration of hardware for the system
  - It also shows the connections between the hardware and the allocation of processes to the hardware
- Requirements such as throughput, performance, and fault-tolerance are taken into account
- Deployment diagrams are created to show the different nodes (processors and devices) in the system

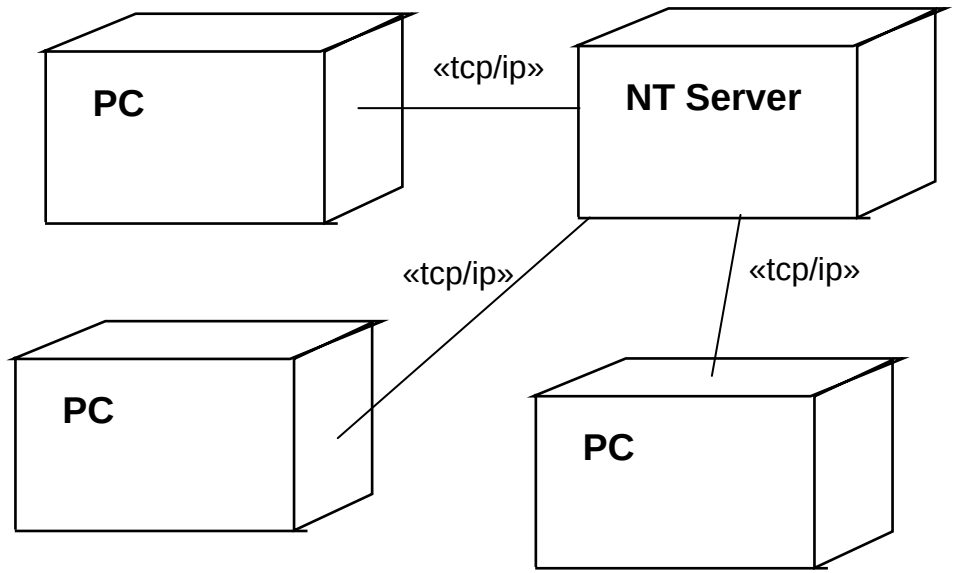
Wyyzzk, Inc.

Notation For Deployment Diagrams

- A node is a run-time physical object representing computational resources
- A connection indicates communication
  - the connection can be stereotyped with the communication protocol
- Nodes can be stereotyped as devices

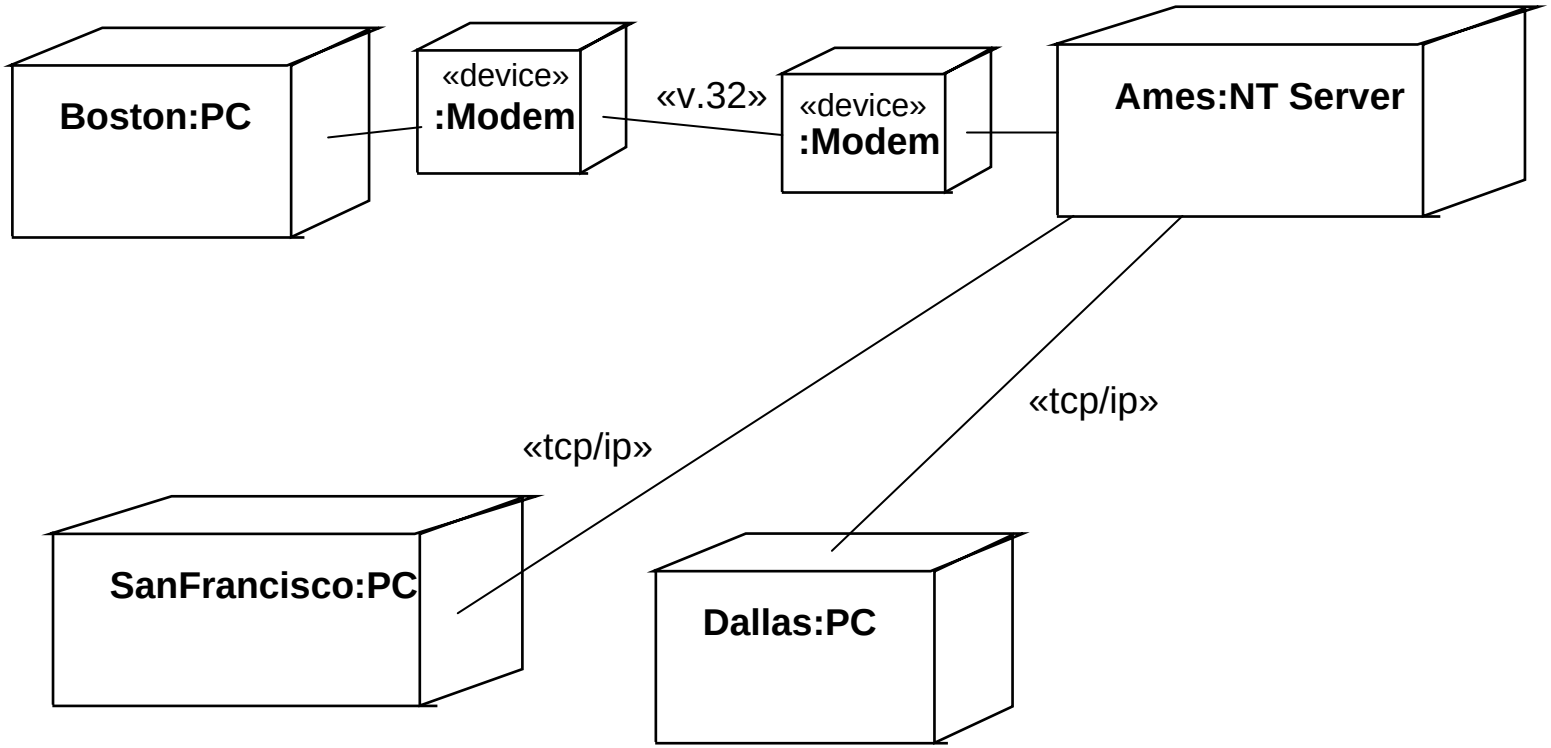


Nodes and Connectors



Wyyzzk, Inc.

Nodes as Devices



Hardware Architecture Issues

- When designing the hardware architecture, various issues affecting the hardware must be resolved, including:
  - Response time and system throughput
  - Communication bandwidth/capacity
  - Physical location of hardware required
  - Distributed processing needs
  - Processor overloading or balance in a distributed system
  - Fault tolerance

Wyyzzk, Inc.

Response time & System throughput

- How quickly does each piece of hardware need to respond?
- How much information can a piece of hardware process at a time?
- Is it faster to have one big computer, or several smaller?
- How much work can really be done in parallel?

Wyyzzk, Inc.

Communication Bandwidth/capacity

- What about the communication path (network)?
- How fast is it?
- What is the capacity?
- Can the network handle anticipated loads?
- Does it matter if the system slows down because of network load?
- Do you have complete control over the network, or is some of it controlled by public utilities (phone lines)?

Wyyzzk, Inc.

Physical location of hardware

- Where will the machines be located?
- Do they need a climate controlled room?
- Who needs access to the hardware?
- What hours of the day/night does the hardware need to be accessible?
- How easy is it to get to for maintenance?

Wyyzzk, Inc.

Processor overloading or balance

- What happens if a particular processor gets overloaded?
- Can some of the load be moved to other machines in the system?
- Do you need to balance processor loads at run time?
- Does it matter if the processors have balanced loads?

Wyyzzk, Inc.

Fault tolerance

- Does the system need to recover from failure?
- To what degree?
- Does the system have to handle everything except catastrophic failure, or will a lesser degree be sufficient?
- Do you need “hot backup” systems to go online if one of the primary processors fails?

Wyyzzk, Inc.

Security

- How will you prevent unauthorized access to information being transmitted between machines?
- How will you prevent unauthorized access to the physical hardware?

Wyyzzk, Inc.

Fail Safe

- Is the system allowed to crash?
- If not, what will you do to prevent the system from crashing?
- Are there mechanisms in place, such as redundant systems, or transaction monitoring, to allow other parts of the system to pick up the load if some parts fail?

Wyyzzk, Inc.

Heterogeneous environments

- Do you have hardware of different types?
  - NT and UNIX for example
- Do these different systems have to communicate?
- How will you get them to communicate?

Wyyzzk, Inc.

Hardware Architecture Issues

- Solutions to hardware architecture issues may become manual processes
- Others will require the creation of new classes or subsystems
- You may need to go back and update other views based on decisions made at this time

Process View of the Architecture

- Most applications today are constructed of multiple threads of control running concurrently
  - This could be multiple jobs on one machine or distributed across several machines
- We need a way to document the use of multiple threads of control, to indicate which parts of the static architecture go into which threads of control, and to resolve the issues associated with synchronizing the threads of control

Wyyzzk, Inc.

What is a Thread of Control?

- A thread of control is a generic term for something that executes independently
  - A thread of control could be implemented as a:
    - Process
    - Thread
    - Job
    - Task
    - Application
- We are most concerned with processes and threads

Process

- Heavyweight flow of control
- Processes are stand-alone
- May be divided into individual threads

Wyyzzk, Inc.

Thread

- Lightweight flow of control
- Threads run in the context of an enclosing process

Wyyzzk, Inc.

Why Multiple Threads of Control?

- We use processes and threads to describe the concurrency requirements of a system
- Some reasons we use multiple processes and threads are:
  - The system is distributed
  - The system is event-driven
  - Some key algorithms are computationally intensive
  - We want to take advantage of the availability of parallel processing supported by the environment

Why Multiple Threads of Control?

- Concurrency requirements are found in the non-functional requirements of the system, or through careful reading of the use cases
- The concurrency requirements should be ranked in terms of importance to resolve conflicts
  - Sometimes the solution for one requirement makes another requirement difficult or impossible to be implemented
  - For example space vs. time trade-offs

Wyyzzk, Inc.

Why Multiple Threads of Control?

- The following kinds of requirements indicate that we need multiple threads of control:
  - The system is required to be distributed
  - Multiple users must be able to perform their work concurrently
  - While the system is processing a request from one user, the results of that request are required by another user performing a different task
  - Prototypes have found that performance needs cannot be met with a monolithic application

Why Multiple Threads of Control?

- We want to utilize multiple CPUs and/or nodes
- We want to increase CPU utilization
- We must provide fast reaction to external stimuli
- We need to service time-related events
- We would like to be able to prioritize activities
- We would like to support load sharing between machines
- We can separate the concerns between software areas
- We can improve system availability
- We want to support major subsystems

Wyyzzk, Inc.

Common Reasons for Multiple Threads of Control

- Some of the more common reasons for creating multiple threads of control are:
  - architecture
  - availability
  - performance

Wyyzzk, Inc.

Architecture

- You may have chosen a logical architecture that requires multiple threads of control
  - Such as client/server
- You may have a distributed physical architecture
  - Each machine will require at least one process
- Because of the dependencies between the parts of the architecture, the process architecture is usually designed along with the logical and physical architectures

Wyyzzk, Inc.

Availability

- Consider the availability of cpu's and other processes this system depends on.
- For the following issues, decide if they are important to your system.
- If so, you need to design a solution.

Wyyzzk, Inc.

Availability

- What if a cpu that is executing a process in your system becomes unavailable?
  - Do you need to be able to move the process at runtime?
- What if a process in your system stops responding?
  - Do you need a way to interrupt it, or restart it, or go to a backup copy of the process?
- What if your system needs to communicate with a process on another machine?
  - Should the processes be aware that they are running on separate machines?
  - What distributed processing techniques will you use?
  - What if the network goes down?

Performance

- Is performance an issue in your system?
- You may be able to increase performance by having multiple machines or CPU's running various parts of the application in parallel
  - The Silicon Graphics graphics engine runs on a separate CPU from the rest of the machine. It has been tuned specifically to perform quick matrix based calculations for graphics.
  - SETI at home is supported by UC Berkeley. They needed a lot of processing power, but couldn't afford a supercomputer. The solution was to distribute small amounts of work to millions of machines working in parallel.

Performance

- On the other hand, if you split the system into multiple processes, you introduce overhead in the form of inter-process communication (IPC)
  - if the communication is over a network, you have even more overhead
- The more processes you have, the larger the potential overhead

New Problems

- Deciding to create multiple threads of control for your system solves some problems, but introduces others
- Three primary things to think of are:
  - System management
  - Synchronization
  - Process/Thread creation and destruction

Wyyzzk, Inc.

System Management

- If the processes are running on multiple machines, how do you handle maintenance on the systems?
  - How do you prevent inconsistencies between versions of processes when upgrading systems?
  - How do you handle scheduled reboots of some of the machines running your system?

Wyyzzk, Inc.

Synchronization

- Do you need to synchronize the behavior of multiple threads of control?
  - How will you accomplish that synchronization?
    - Especially if the synchronization is across multiple machines
- What IPC strategy will you use?

Wyyzzk, Inc.

Thread of Control Creation and Destruction

- In a single process, single threaded system we don't have to worry about process or thread creation and destruction
- Once you design multiple threads of control you also have to decide when and how processes and threads will be created, and when and how they will be destroyed when no longer needed

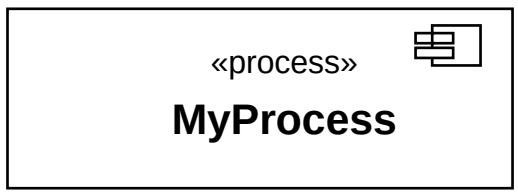
Create Processes and Threads

- Now that you have identified the need for multiple threads of control, choose which will be processes, which threads, and document your decisions.
- For each separate thread of control needed by the system, create a process or a thread (lightweight process).
  - A thread should be used in cases where there is a need for nested flow of control (i.e. within a process, there is a need for independent flow of control at the sub-task level)
- You will have to consider your run-time platform when creating processes and threads
  - Does your platform support multiple processes and multiple threads?
  - Is there a limit on how many processes and threads you can create?

Wyyzzk, Inc.

Documenting Processes in UML

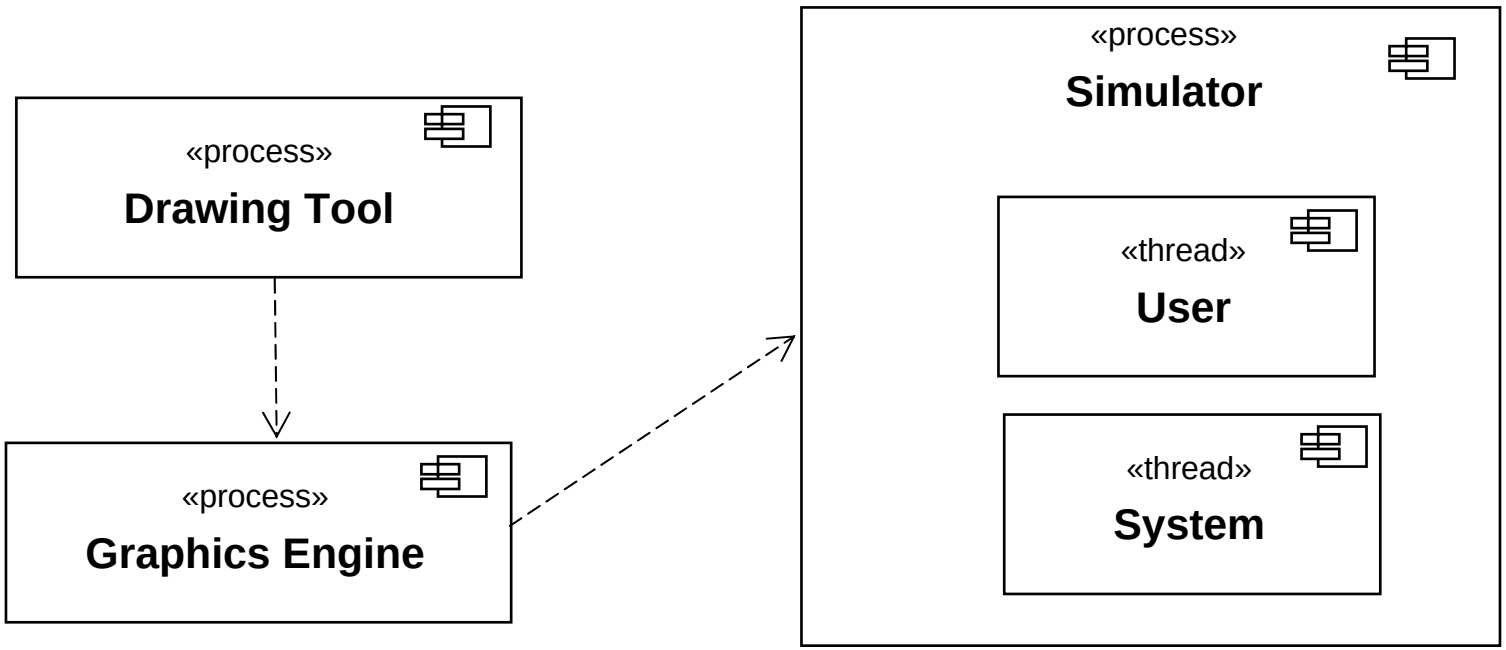
- Processes are represented by UML components
- We will create processes in a component diagram



Wyyzzk, Inc.

Modeling Threads in UML

- Threads are shown by nesting them inside processes
- Threads can also have interfaces



Wyyzzk, Inc.

Process View vs Logical View

- Not only do you have to design the process view, but you have to decide how the logical view fits into the process view
  - Where do the subsystems fit inside the processes and threads?

Wyyzzk, Inc.

Subsystems and Processes

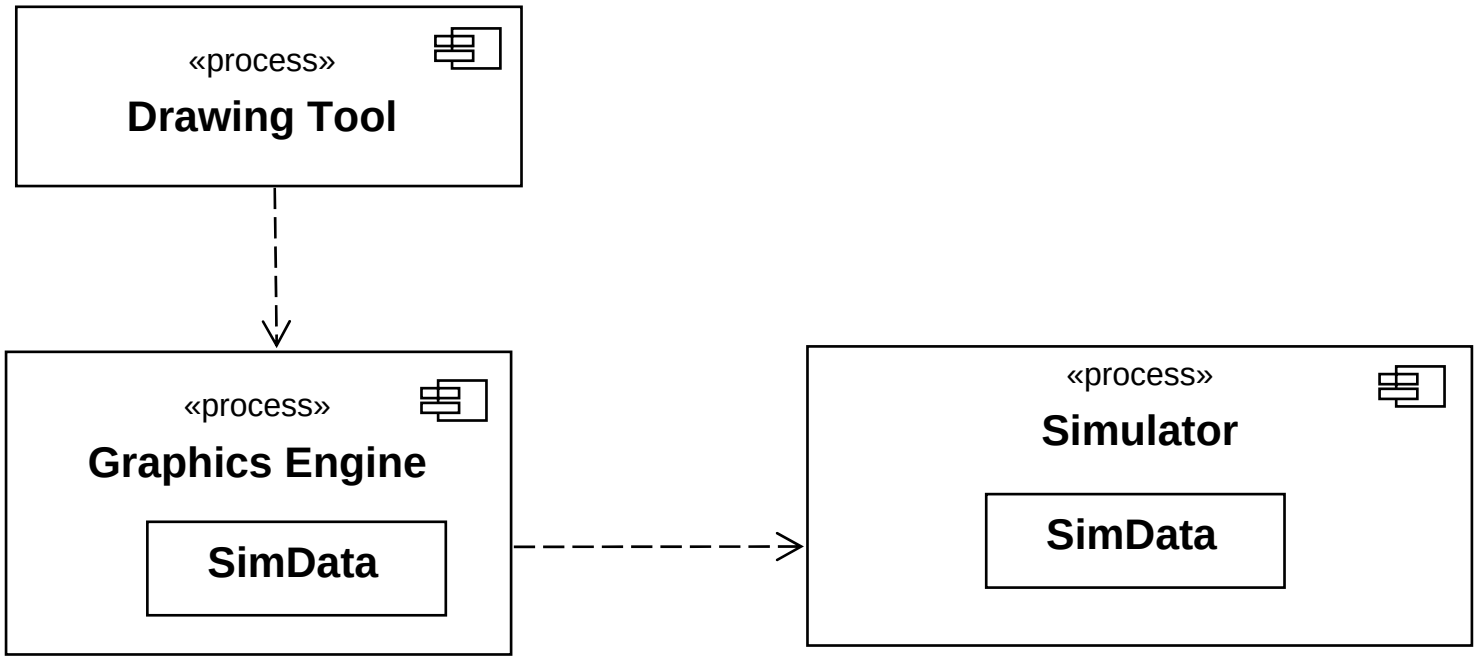
- You might decide to make one process for each subsystem and a thread for each nested subsystem
- You could put multiple subsystems into one process
- You could divide a subsystem between processes
  - In this case you want to go back to the subsystem and create nested subsystems for the parts that are being split

Wyyzzk, Inc.

Classes and Processes

- Classes provide the definitions for objects
- When a process creates an object, it uses a class
  - That class must be part of the process
  - If the object is used by more than one process, then the class is also part of more than one process
    - For example, if you pass objects between processes, each process must include the corresponding class for the object
  - This is different from the logical view where classes belong to just one subsystem

Classes in Processes



The class SimData resides in Graphics Engine and Simulator. Both components need the class in order to share data.

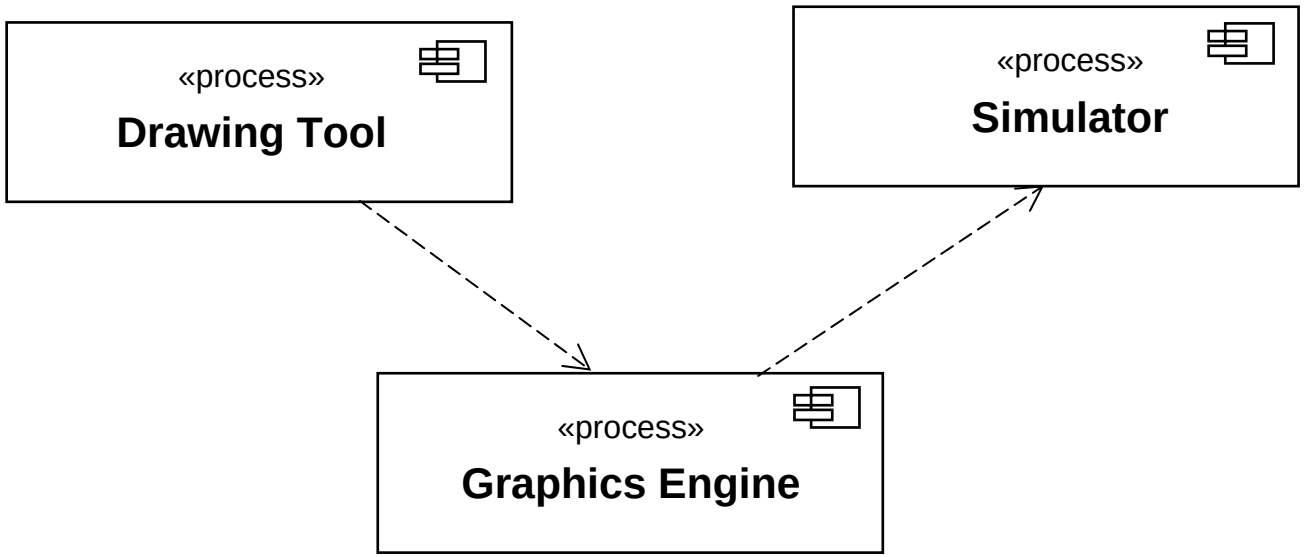
Wyyzzk, Inc.

Design Elements to Processes

- Group elements that closely cooperate, and must execute in the same thread of control
- Separate elements which do not interact
- Repeat until the minimum number of processes is reached that still provide the required distribution and effective resource utilization

Relationships between Processes

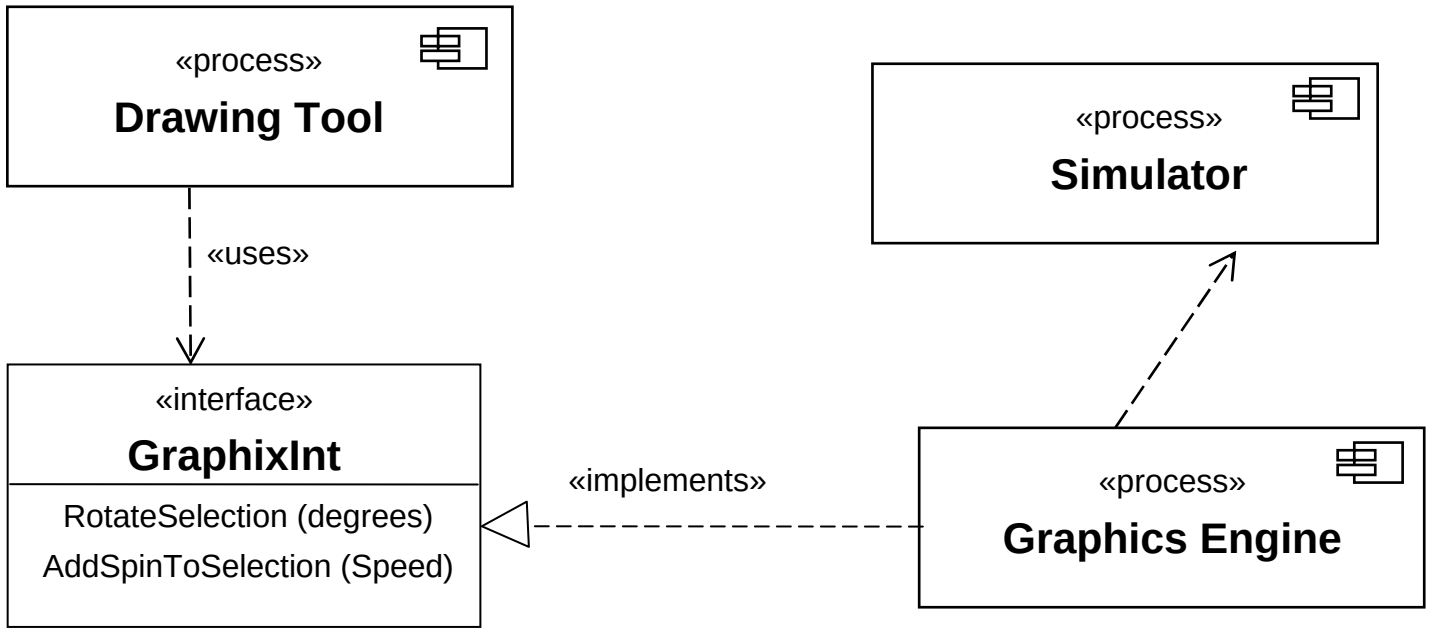
- Use the dependency arrow to show which processes communicate



Wyyzzk, Inc.

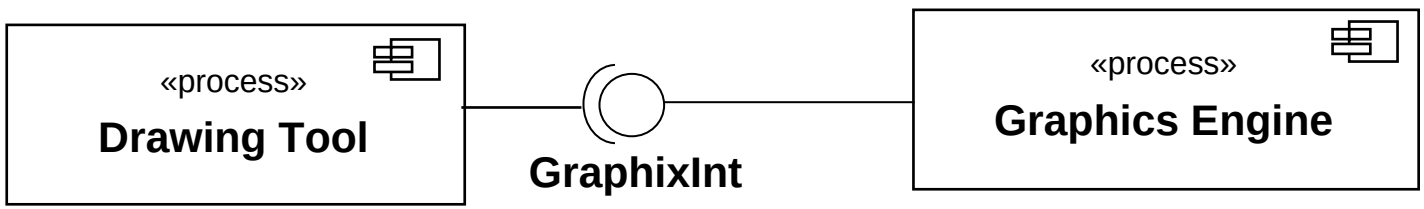
Defining Process Interfaces

- Use interfaces to show the operations in the process interface



Process Interfaces

- If the diagram gets cluttered, use the simplified form of an interface, which does not show the operations in the interface



Process Interfaces (cont.)

- The interface can be any published interface for your process
  - COM object interface
  - CORBA object interface
  - API
  - Protocol

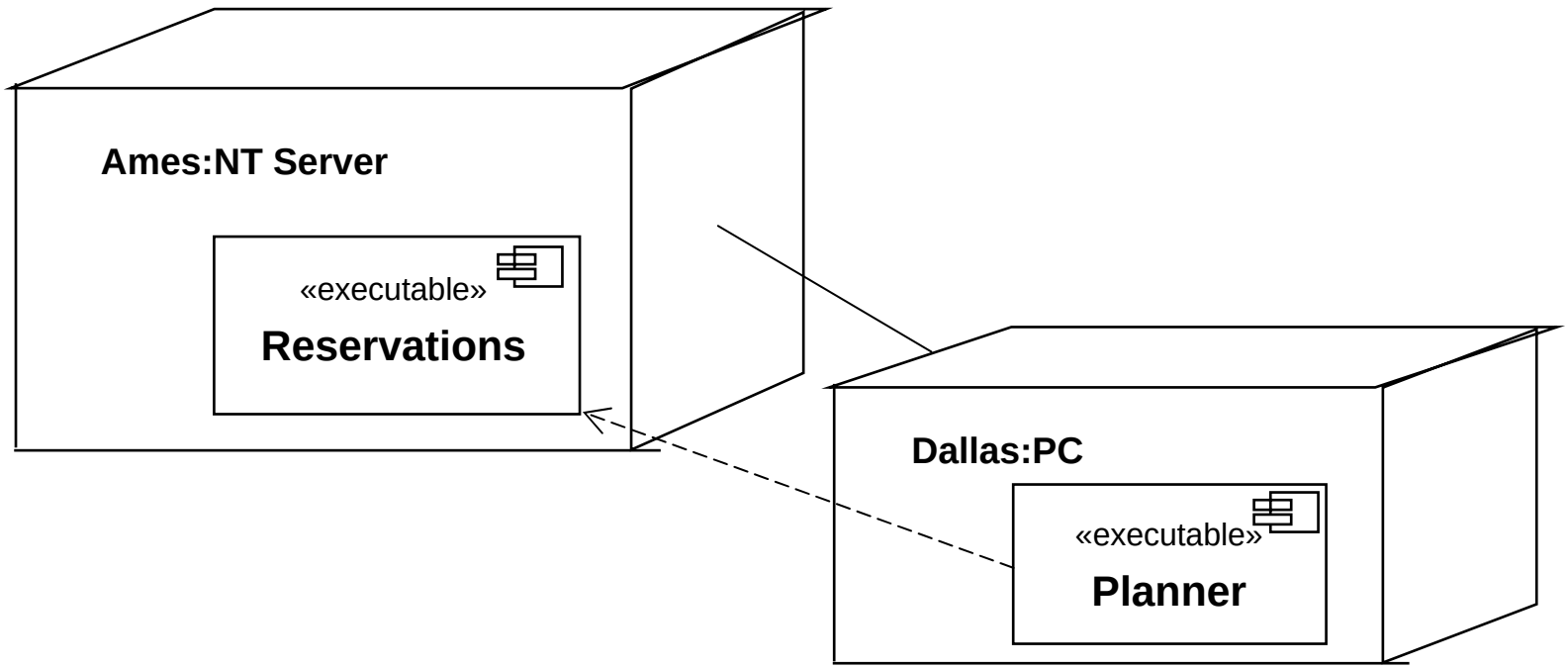
Wyyzzk, Inc.

Showing Processes on Nodes

- You can add processes to the deployment view to show which processes run on which processors
- This is also a way to see where you need to communicate over a network

Wyyzzk, Inc.

Showing Processes on Nodes



Wyyzzk, Inc.

Development View

- As defined by Phillipe Krutchen, this view is seldom used in practice.
  - Most of the information in this view is better documented in architectural guidelines and standards, or in the project team's documentation of their workspaces

Wyyzzk, Inc.

Use Case View

- Architecturally significant use cases
  - Important to the business – a primary functionality of the company
  - Describe a flow through most or all of the major architectural components
  - Cause the selection of one architectural pattern over another
  - Cause the addition of significant components to the architecture

Wyyzzk, Inc.

Other Information

- Data Model
- Analysis Model
- Policies
- Mechanisms

Wyyzzk, Inc.

Data Model

- At the architecture level, this shows the shared persistent data
- Typically expressed in Entity-Relationship (ER) diagrams
- May be described with UML Class diagrams

Wyyzzk, Inc.

Analysis Model

- At the architecture level, this shows the shared run-time data
- Typically described with UML Class diagrams

Wyyzzk, Inc.

Policies

- Described in a text document
- Can be rules to follow or guidelines

Wyyzzk, Inc.

Mechanisms

- Describe a standard way of doing something
  - Error handling
  - Interacting with a database
  - Communicating over a network
- Described as a pattern
  - Class Diagram
  - Sequence Diagram (optional)

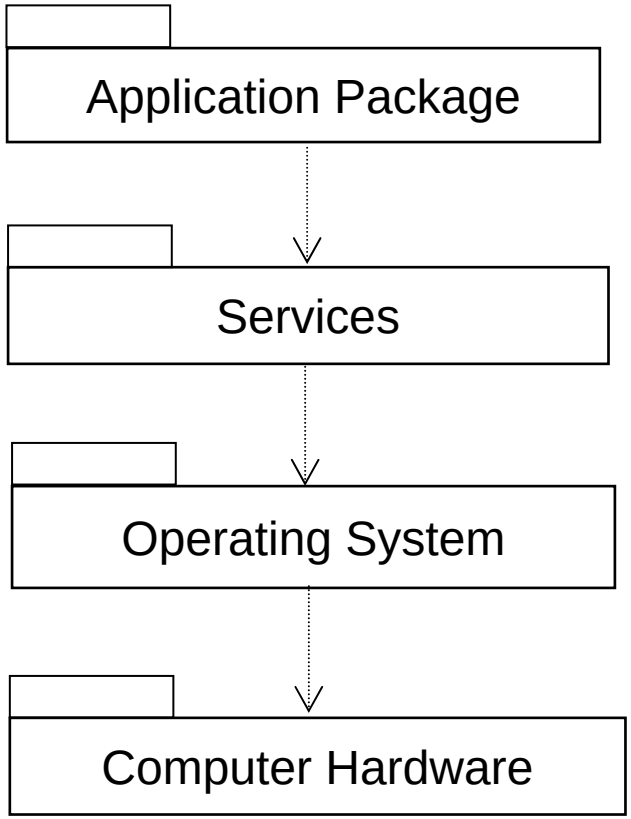
Wyyzzk, Inc.

Multiple layers of Architecture

- A subsystem of an architecture could become very large or complicated.
- Under these conditions, it is necessary to design an architecture for that subsystem
- The architecture of the subsystem can be different from the architecture of the whole system
  - The system might be a 3 tier architecture, but the presentation tier might be designed as a MVC architecture.

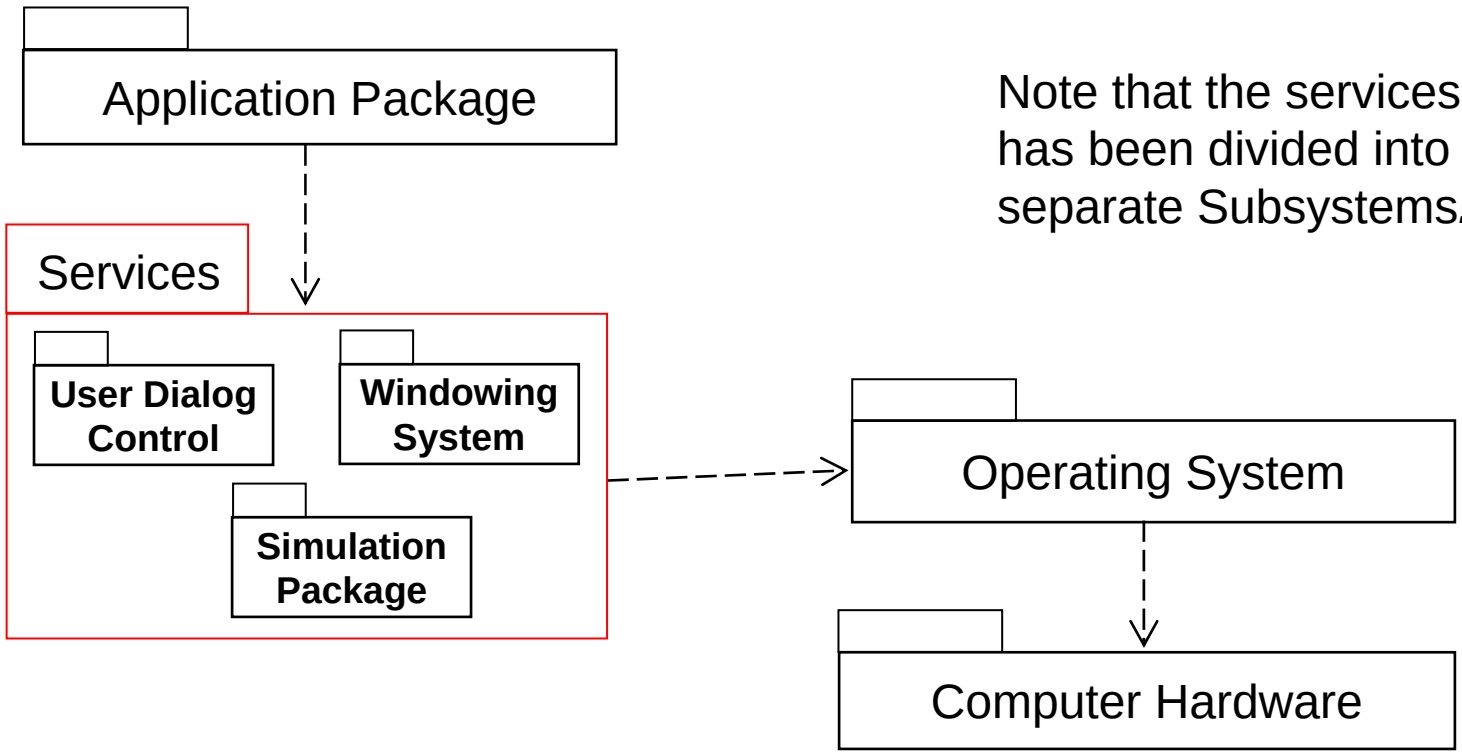
Wyyzzk, Inc.

Example: Decomposing a Layered Architecture



Wyyzzk, Inc.

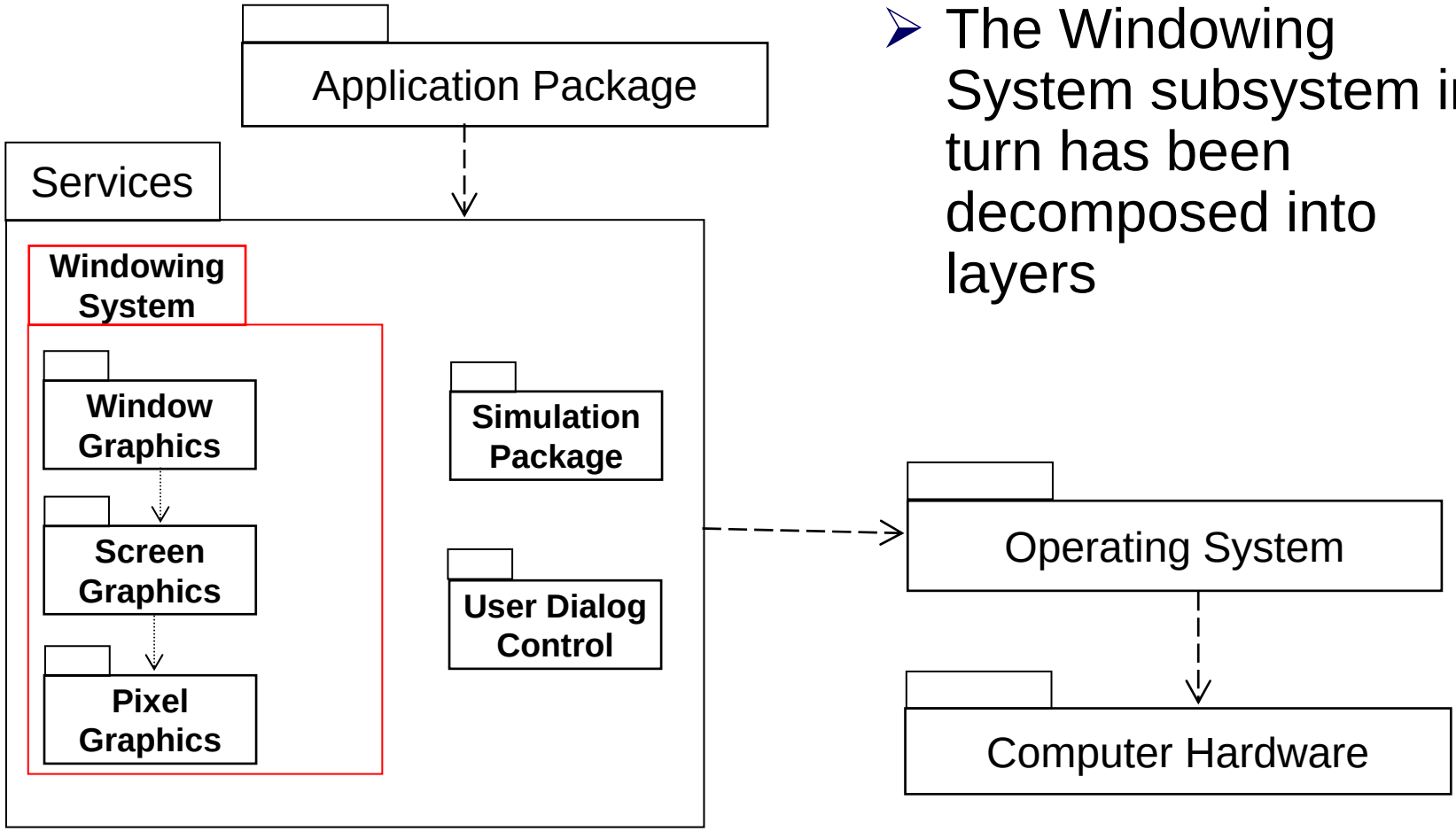
Example: Decomposing a Layered Architecture (cont.)



Note that the services layer has been divided into three separate Subsystems.

Wyyzzk, Inc.

Decomposing a Layered Architecture (cont.)



- The Windowing System subsystem in turn has been decomposed into layers

Wyyzzk, Inc.

Summary

- 1. Identify components, bottom-up and/or top-down
- 2. Realize the architecturally significant use cases using components on sequence diagrams
- 3. Add relationships and operations to components
- 4. Create the physical architecture
- 5. Convert the subsystems into processes and threads on the hardware
- 6. Add data model, analysis model, policies, and mechanisms as appropriate
- 7. Document the architecture (another lesson)