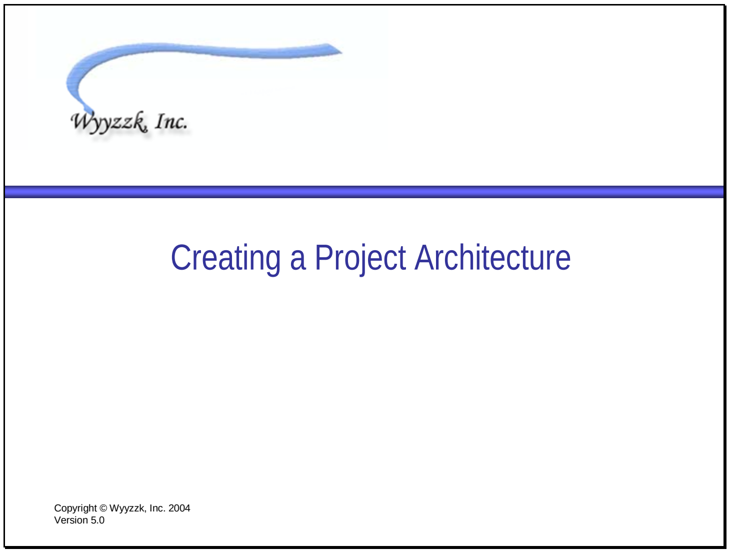
Slide 1

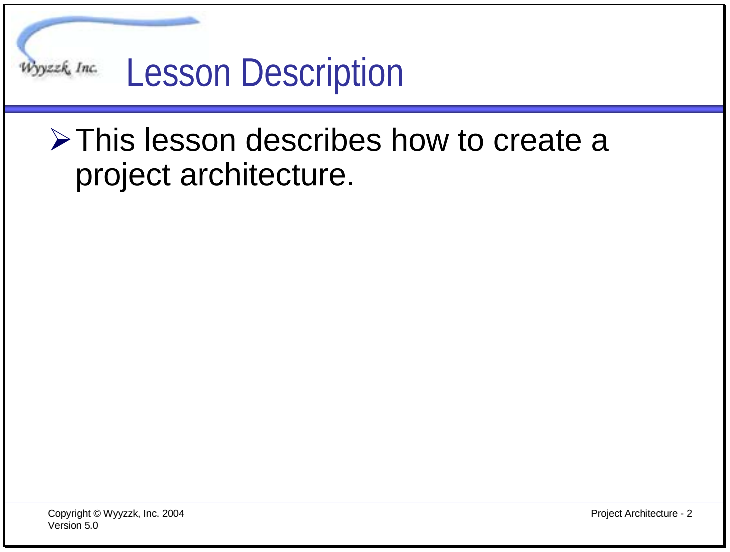


Wyyzzk, Inc.

Creating a Project Architecture

Version 5.0

Slide 2



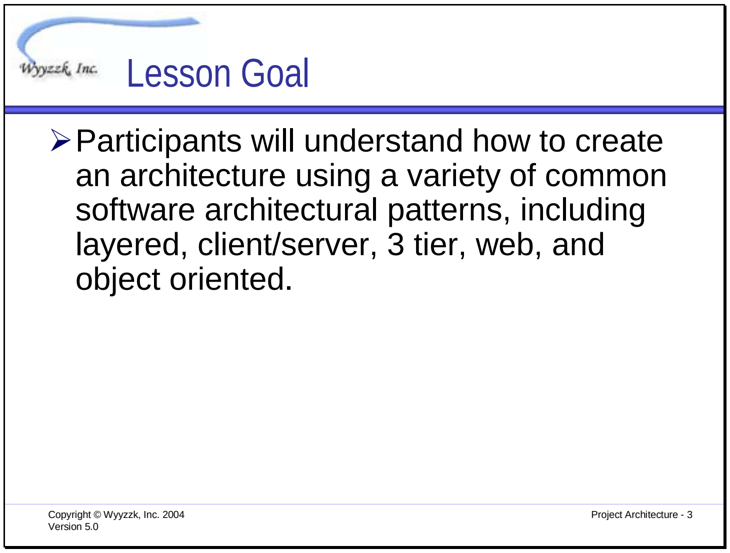
Wyyzzk, Inc.

Lesson Description

- This lesson describes how to create a project architecture.

Version 5.0

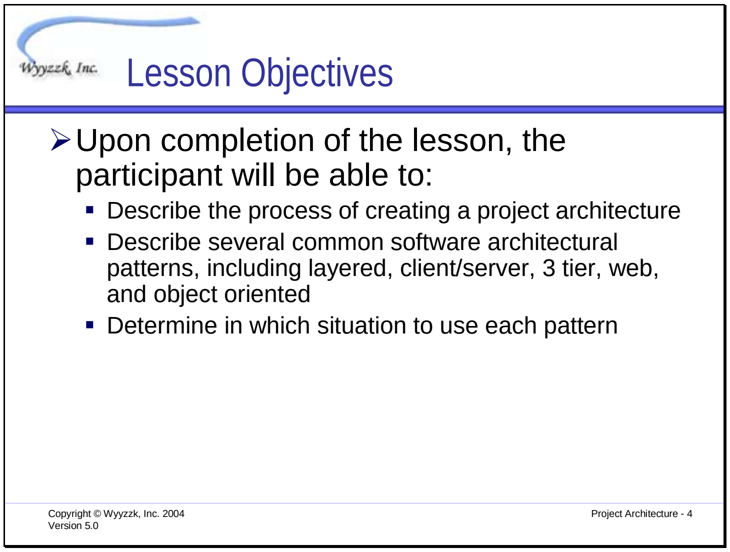
Slide 3



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.

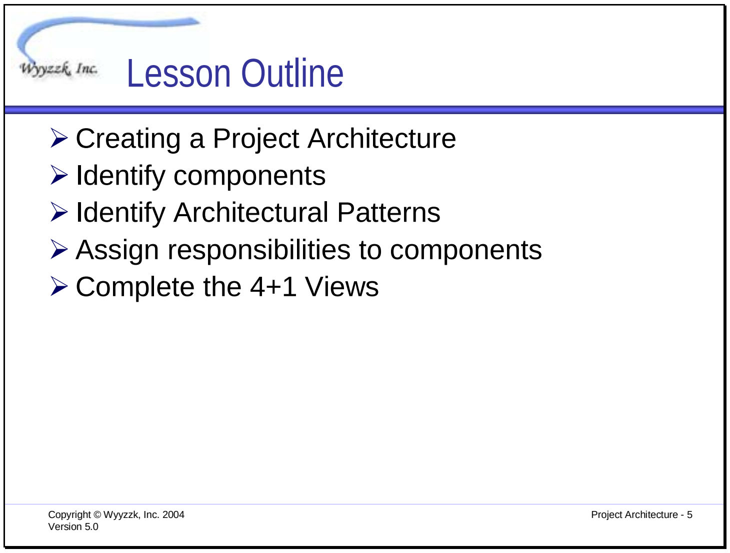
Slide 4



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

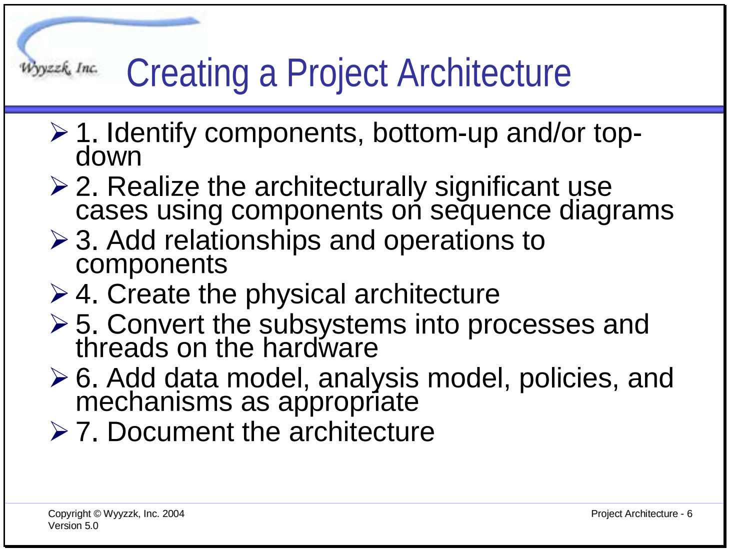
Slide 5



Lesson Outline

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

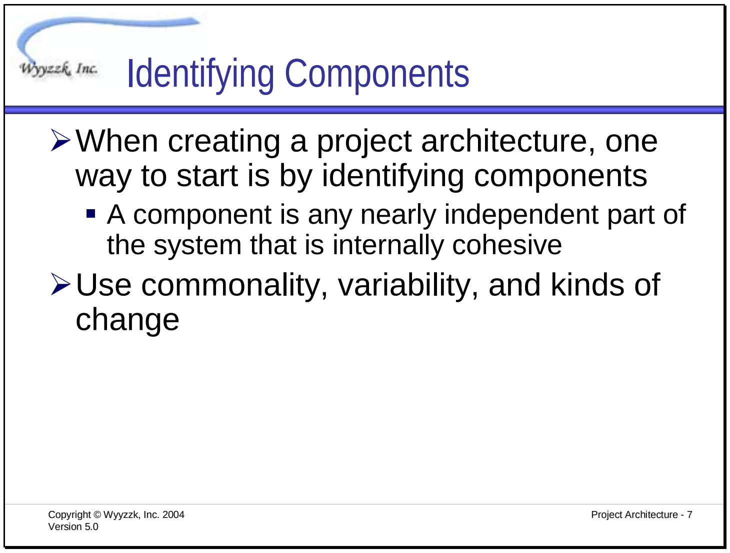
Slide 6



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

Slide 7



Identifying Components

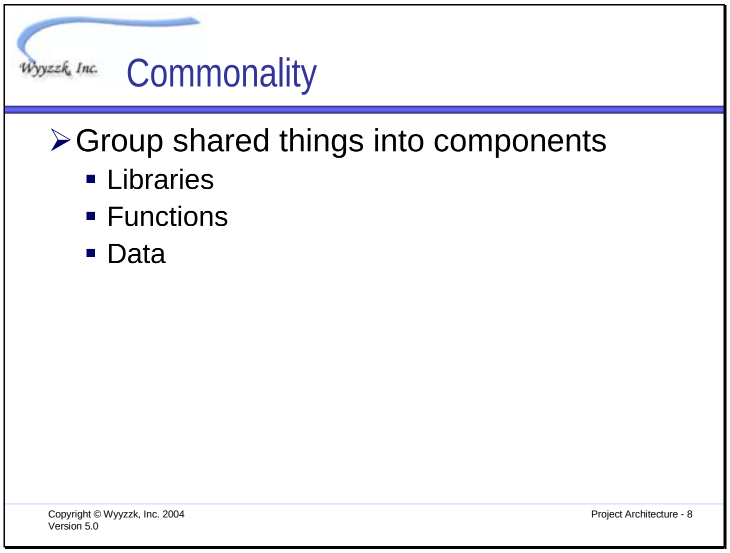
- When creating a project architecture, one way to start is by identifying components
  - A component is any nearly independent part of the system that is internally cohesive
- Use commonality, variability, and kinds of change

There are different approaches to creating a project architecture. In this section, we will look at a “bottom-up” approach and a “top-down” approach.

In a bottom-up approach, you start by identifying components, then determine how they will relate to each other. In a top-down approach, you start by identifying overall patterns for the architecture, then determine what components are needed to implement the patterns. Most architects do both at the same time, trying out different ideas until they are happy with the result.

When identifying components, consider commonality, variability, and kinds of change. Isolating areas that change is an important part of any architecture.

Slide 8



Wyyzzk, Inc.

Commonality

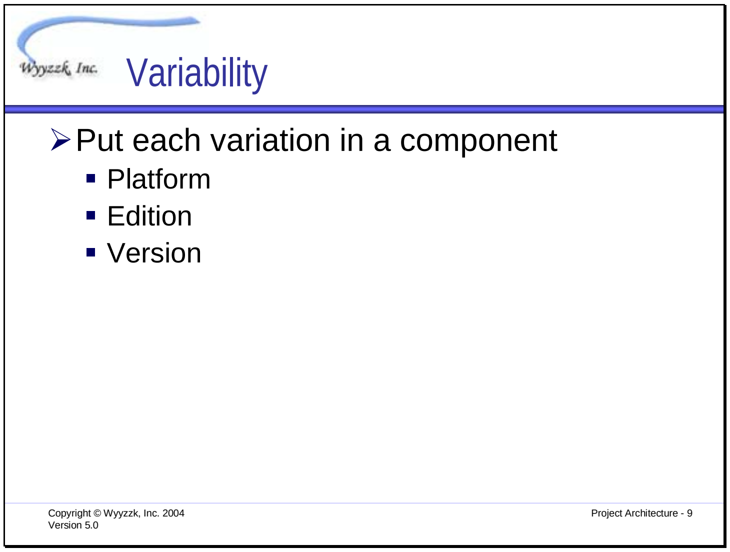
- Group shared things into components
  - Libraries
  - Functions
  - Data

Copyright © Wyyzzk, Inc. 2004
Version 5.0

Project Architecture - 8

Things that are shared in the application (libraries, functions, data) may be put into one or more components that are shared in the application. So you look for common things and create components to hold them. This gives you a single point of maintenance for things that are shared, which is very desirable. Look at the lesson on product line architectures for examples of this approach.

Slide 9



Wyyzzk, Inc.

Variability

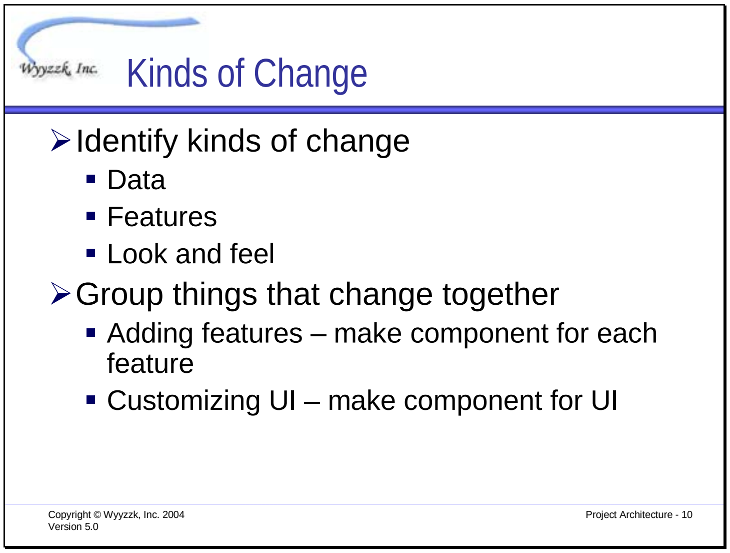
- Put each variation in a component
  - Platform
  - Edition
  - Version

Copyright © Wyyzzk, Inc. 2004
Version 5.0

Project Architecture - 9

Look at the application for areas that vary from a common base. Each variation can be made into a component that shares the common base functionality. Refer to the lesson on product line architectures for examples of this approach.

Slide 10



Kinds of Change

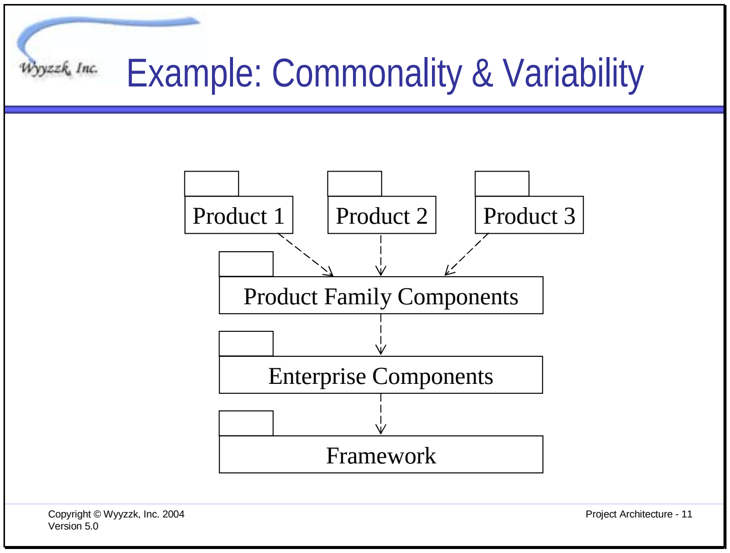
- ➢ Identify kinds of change
  - ▪ Data
  - ▪ Features
  - ▪ Look and feel
- ➢ Group things that change together
  - ▪ Adding features – make component for each feature
  - ▪ Customizing UI – make component for UI

Project Architecture - 10

A thing way to approach architecture is to consider what kind of changes might be required to the code in the future. In the best architecture, a particular kind of change should take place (as much as possible) within one component. Kinds of changes could be changes to the data (format change, change of database, addition or deletion of fields), changes to features (adding or deleting features, custom features, locking or unlocking features), or changes to the look and feel (change of UI colors, adding logos, internationalization).

So if the most common kind of change to your application is the addition of new features, then you might want a design with some basic service type components, then add a component for each feature. A feature component would include all the UI for that feature, the business rules, and the code to access the database (if not its own database). On the other hand, if your most common change is to the user interface, then you would put all of the user interface together in a component, which is used by other components of the system whenever interaction with the user is desired.

Slide 11



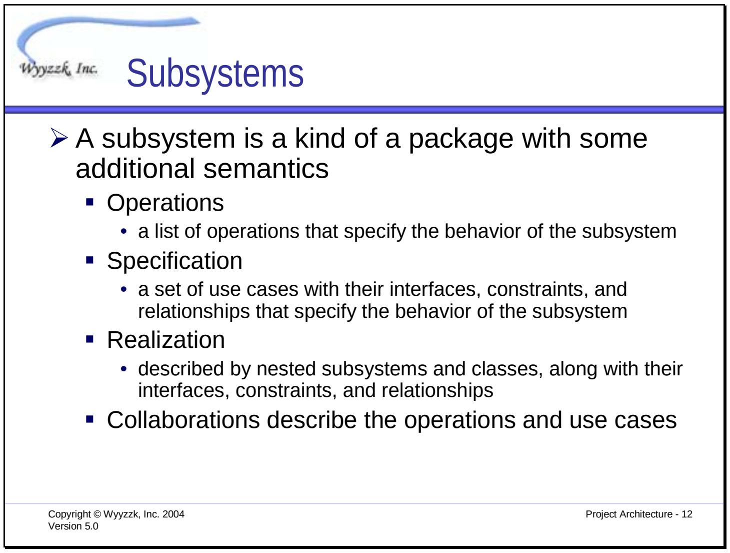
Example: Commonality & Variability

Copyright © Wyyzzk, Inc. 2004
Version 5.0

Project Architecture - 11

Having identified some components, we may later put those components into one or more of the common architectural patterns. If you start with an architectural pattern, then you have to identify the components of the pattern. In either approach, you end up with an overall pattern, and components that are meaningful to your application.

Slide 12



Subsystems

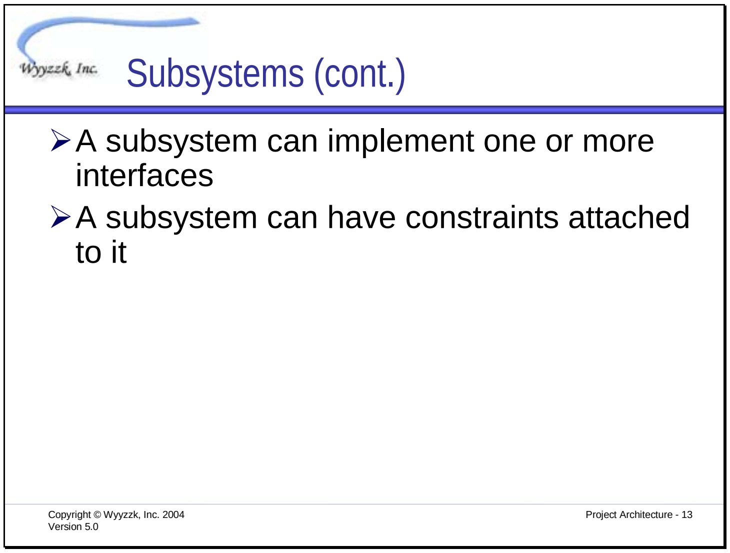
- A subsystem is a kind of a package with some additional semantics
  - Operations
    - a list of operations that specify the behavior of the subsystem
  - Specification
    - a set of use cases with their interfaces, constraints, and relationships that specify the behavior of the subsystem
  - Realization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

Copyright © Wyyzzk, Inc. 2004
Version 5.0

Project Architecture - 12

We start to document the components of the architecture using subsystems. At this point we are working at Bredemeyer's conceptual level. We show the component name, and assign responsibilities to the component.

slide 13



Wyyzzk, Inc.

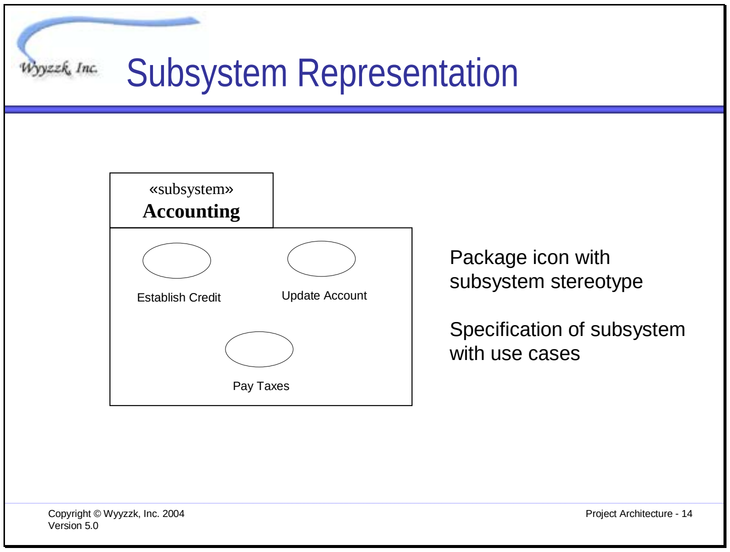
Subsystems (cont.)

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

Version 5.0

Project Architecture - 13

Slide 14



Wyyzzk, Inc.

Subsystem Representation

«subsystem»
Accounting

Establish Credit

Update Account

Pay Taxes

Package icon with subsystem stereotype

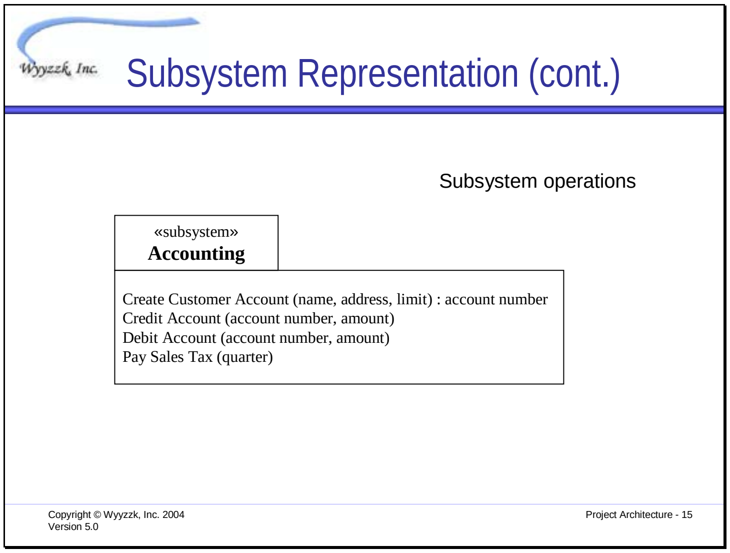
Specification of subsystem with use cases

Version 5.0

Project Architecture - 14

A subsystem is more than a name; it also has responsibilities. One way to indicate the responsibilities of a subsystem is by assigning use cases to the subsystem. This shows that the subsystem has to provide code to implement these use cases.

Slide 15



Subsystem Representation (cont.)

Subsystem operations

«subsystem»
Accounting

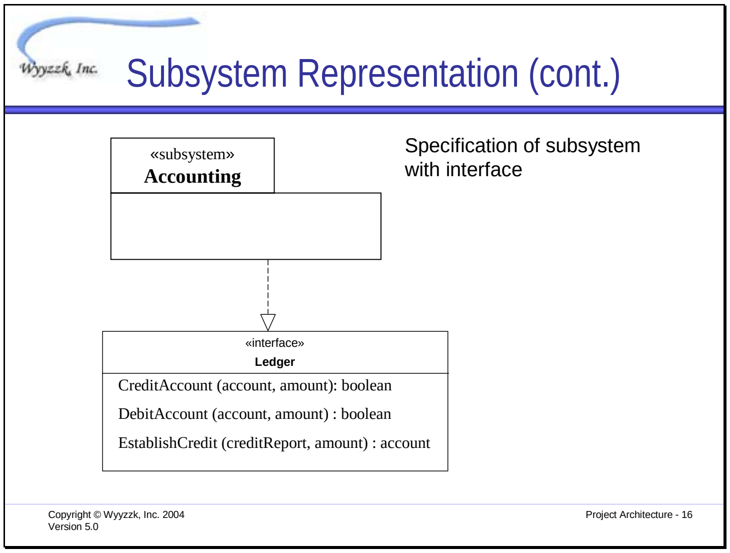
Create Customer Account (name, address, limit) : account number
Credit Account (account number, amount)
Debit Account (account number, amount)
Pay Sales Tax (quarter)

Copyright © Wyyzzk, Inc. 2004
Version 5.0

Project Architecture - 15

Another way to indicate the responsibilities of a subsystem is by assigning operations to the subsystem. This shows that the subsystem has to provide code to implement these operations.

Slide 16



Subsystem Representation (cont.)

Specification of subsystem with interface

«subsystem»
Accounting

«interface»
Ledger

CreditAccount (account, amount): boolean

DebitAccount (account, amount) : boolean

EstablishCredit (creditReport, amount) : account

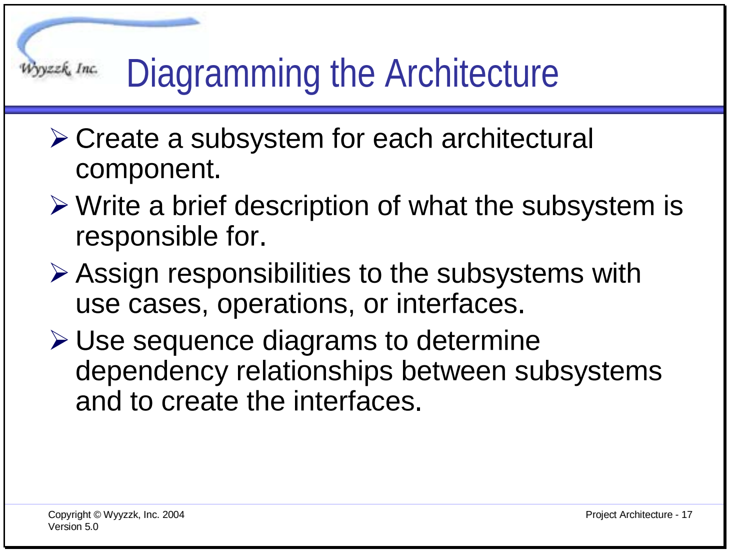
Copyright © Wyyzzk, Inc. 2004
Version 5.0

Project Architecture - 16

Another way to indicate the responsibilities of a subsystem is by showing that it implements one or more interfaces. This shows that the subsystem has to provide code to implement the operations in the interfaces.

There is a subtle difference between assigning operations to a component and having the component implement an interface. If operations are assigned to a component, the operations are part of the component. However, an interface is separate from the component, so you can plug any component into the interface as long as the component provides an implementation for the interface. This is one way to implement a plug-in architecture.

Slide 17



Diagramming the Architecture

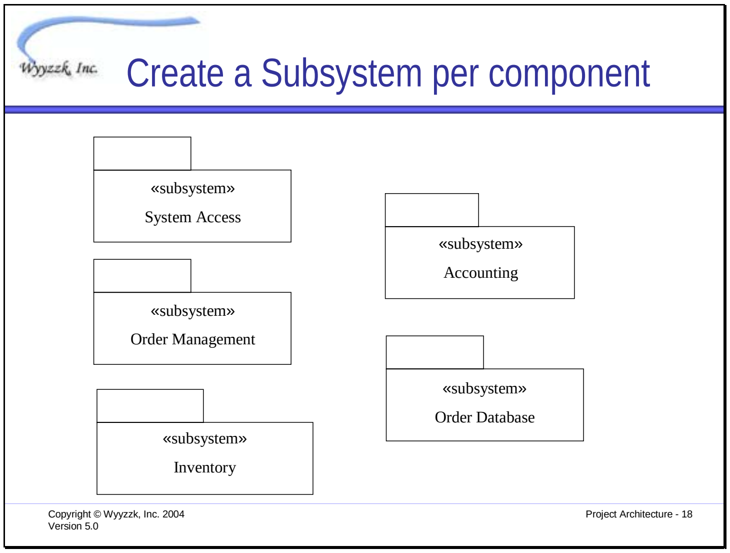
- Create a subsystem for each architectural component.
- Write a brief description of what the subsystem is responsible for.
- Assign responsibilities to the subsystems with use cases, operations, or interfaces.
- Use sequence diagrams to determine dependency relationships between subsystems and to create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

Version 5.0

Project Architecture - 17

We will look at assigning responsibilities, creating sequence diagrams, and creating relationships between components later in the lesson. For now, we just create a basic diagram with the components and write descriptions for each.

Slide 18



Create a Subsystem per component

«subsystem»
System Access

«subsystem»
Order Management

«subsystem»
Inventory

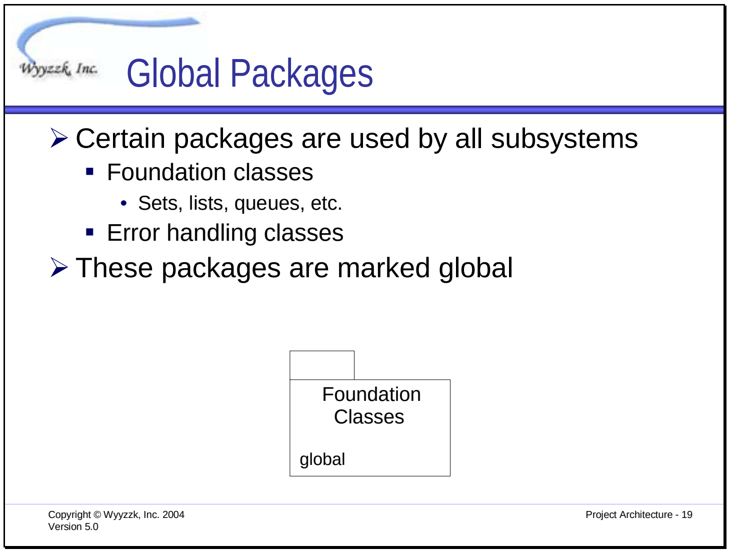
«subsystem»
Accounting

«subsystem»
Order Database

Version 5.0

Project Architecture - 18

Slide 19



Global Packages

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

Foundation Classes

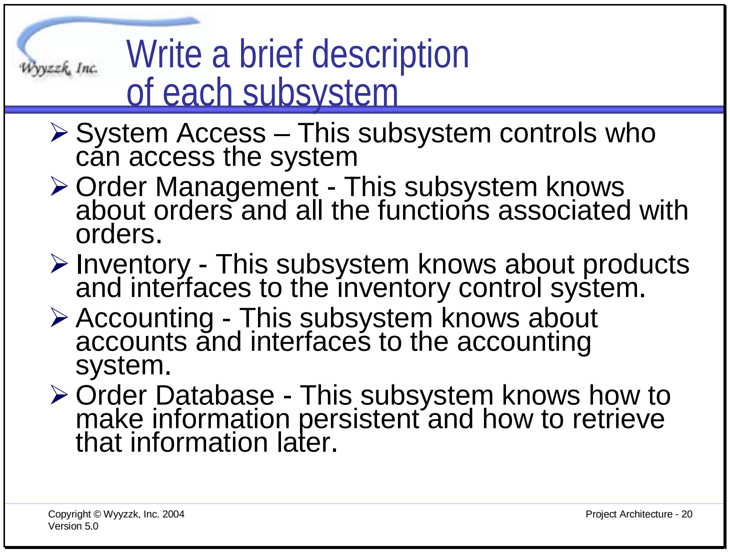
global

Copyright © Wyyzzk, Inc. 2004
Version 5.0

Project Architecture - 19

You do not have to show any relationships to global packages. Putting the notation “global” on the package indicates that all other subsystems (packages) can use this package.

Slide 20



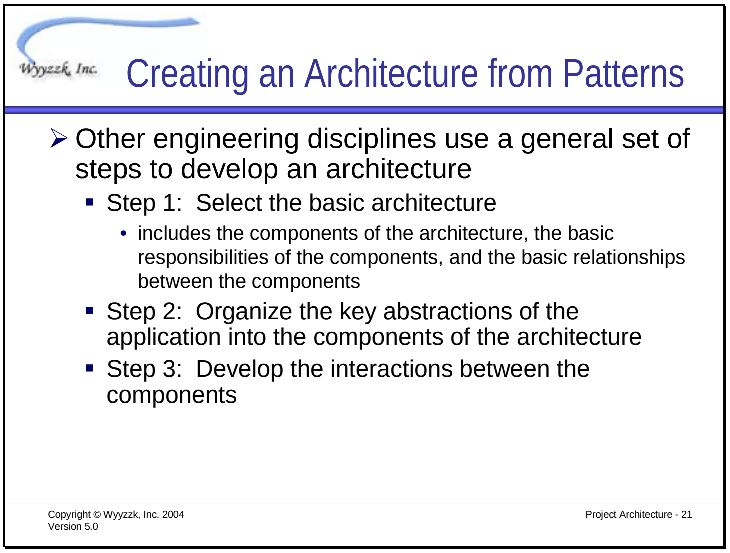
Write a brief description of each subsystem

- 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.

Copyright © Wyyzzk, Inc. 2004
Version 5.0

Project Architecture - 20

Slide 21



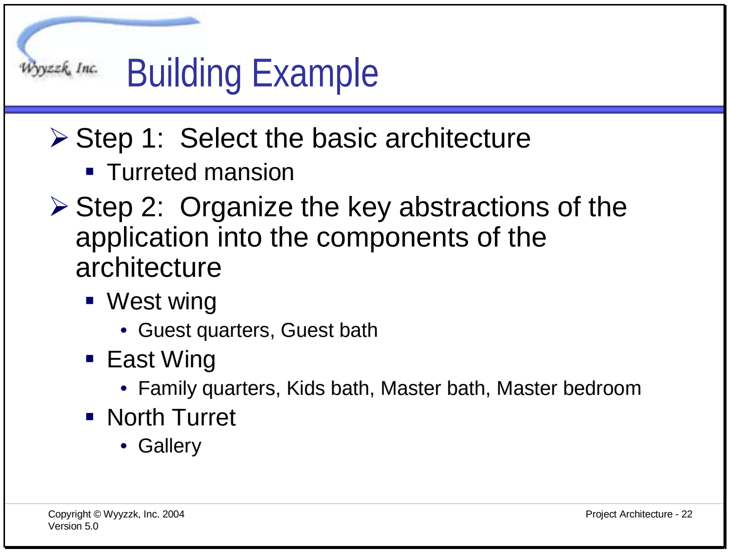
Creating an Architecture from Patterns

- 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

Project Architecture - 21

This process is based around the idea of starting with a pattern for the architecture and developing it further with specifics for your application. We are still working on getting a set of components for our architecture.

Slide 22

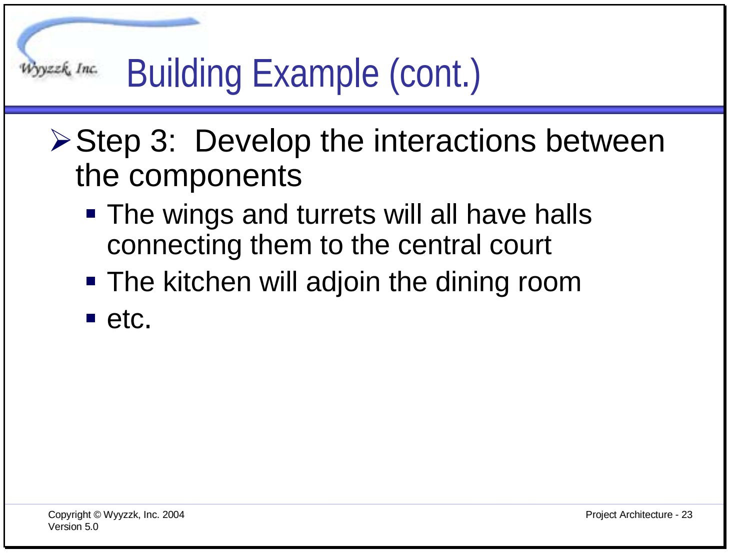


Building Example

- 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

Project Architecture - 22

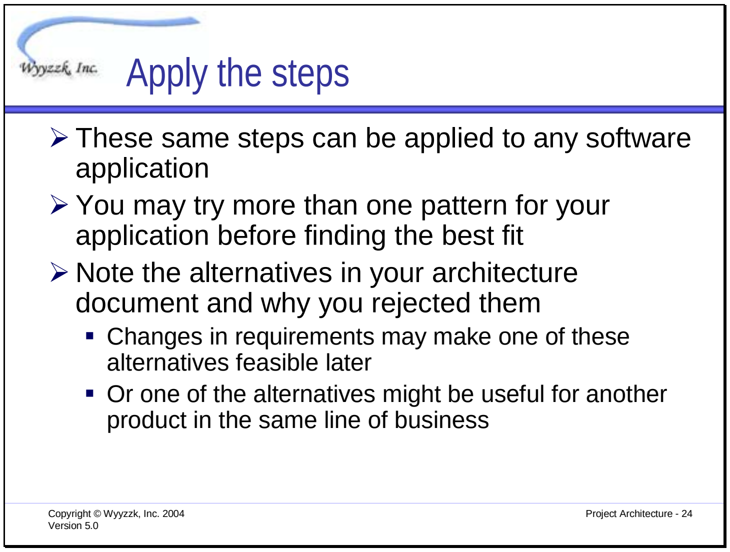
Slide 23



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.

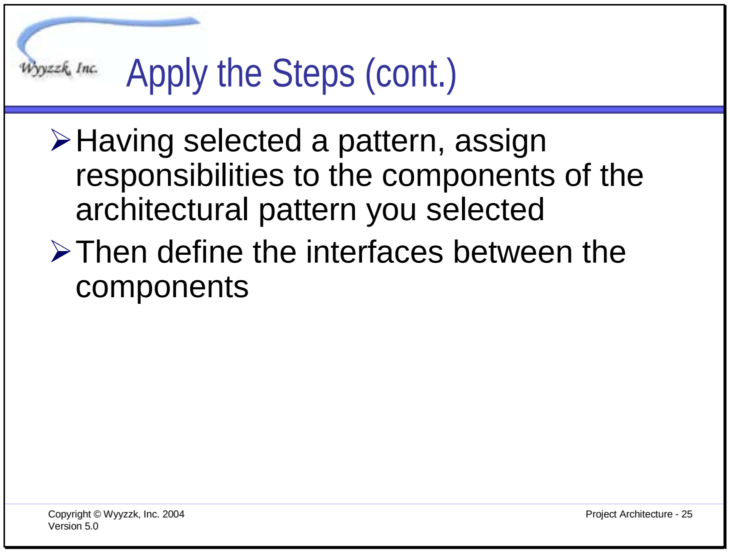
Slide 24



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

Slide 25



Apply the Steps (cont.)

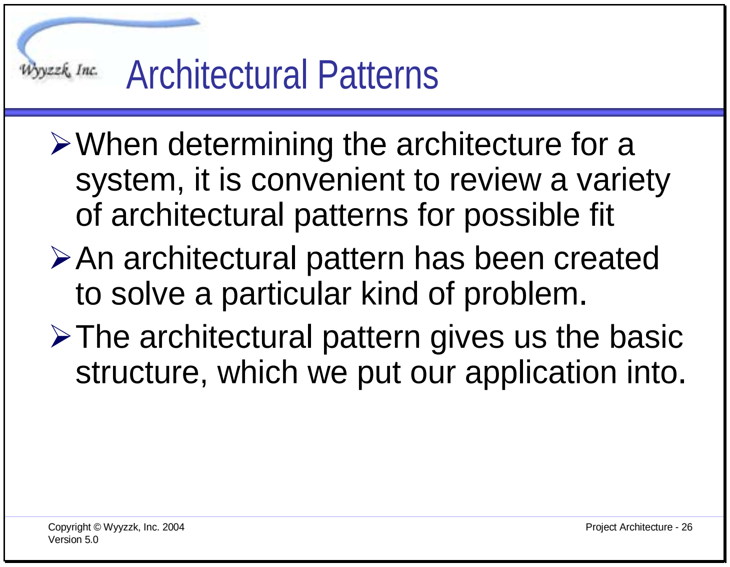
- Having selected a pattern, assign responsibilities to the components of the architectural pattern you selected
- Then define the interfaces between the components

Copyright © Wyyzzk, Inc. 2004
Version 5.0

Project Architecture - 25

We will go through the top-down approach to the same point as the bottom-up – identifying and describing components. Once we have reached that point, both methods continue the same way, by continuing to refine the components of the architecture. The difference in top-down and bottom-up is just how you go about selecting the components to begin with.

Slide 26



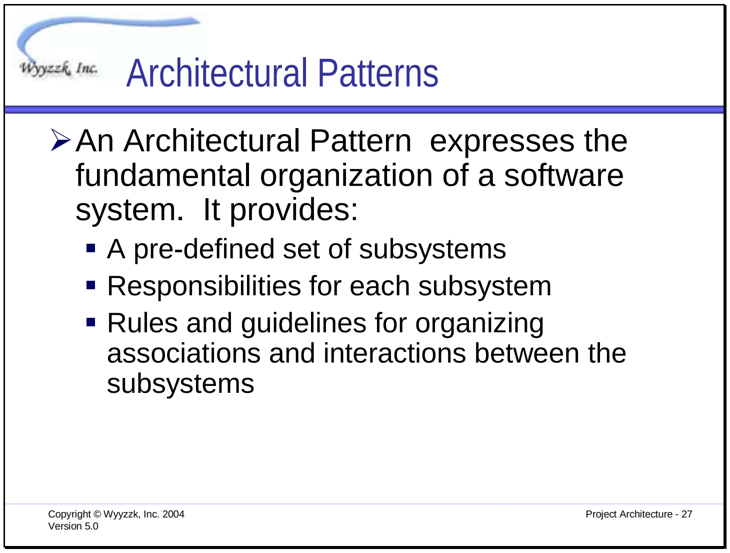
Architectural Patterns

- 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.

Copyright © Wyyzzk, Inc. 2004
Version 5.0

Project Architecture - 26

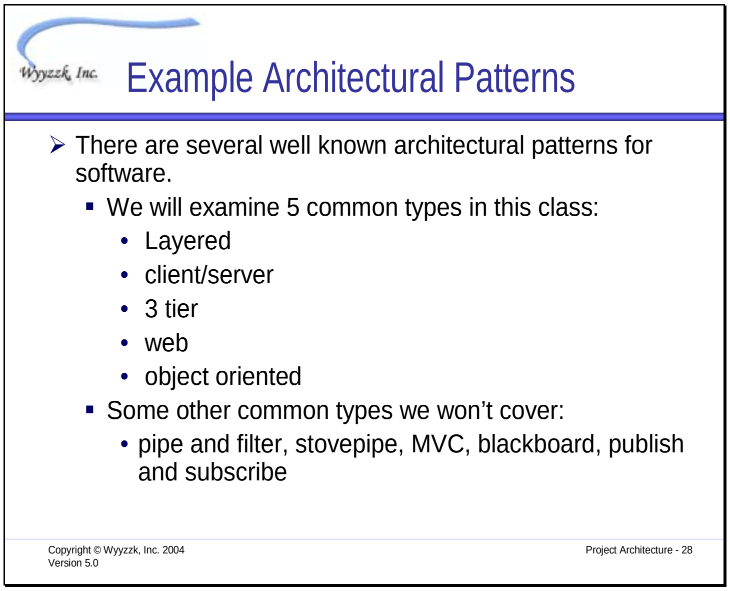
Slide 27



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

Slide 28

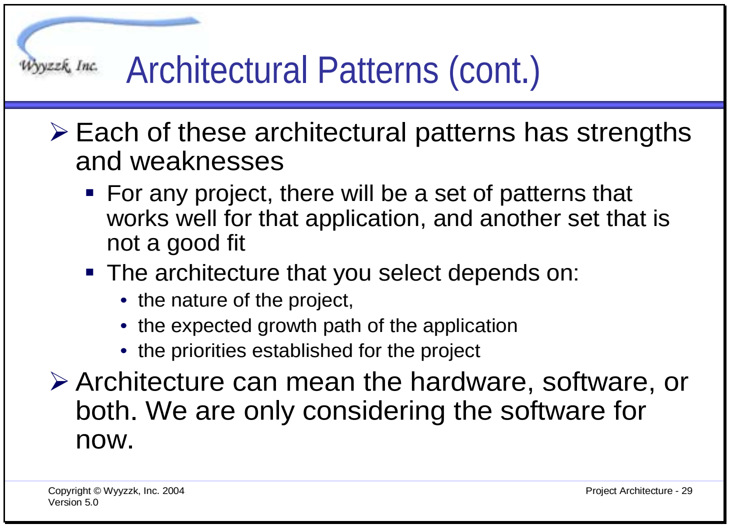


Example Architectural Patterns

- 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

There are quite a number of basic architectural patterns identified. See for example: Pattern-Oriented Software Architecture, Volume 1: A System of Patterns; Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal – and Software Architecture: Perspectives on an Emerging Discipline; Mary Shaw, David Garlan

Slide 29



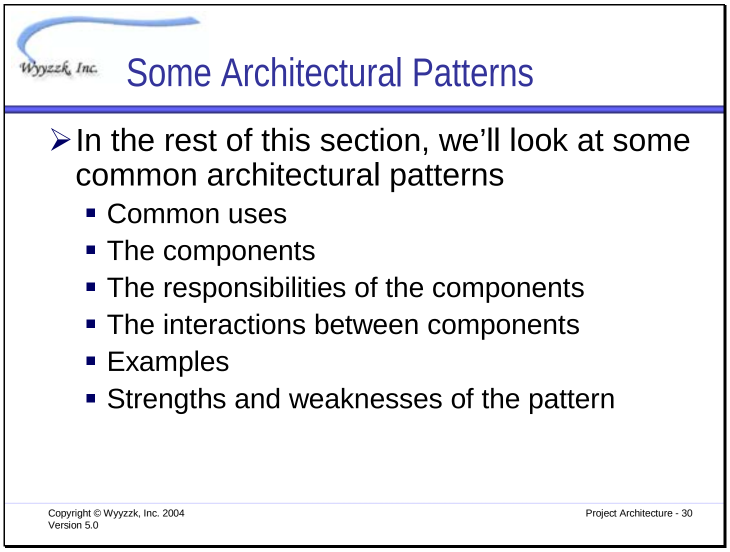
Architectural Patterns (cont.)

- 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.

Copyright © Wyyzzk, Inc. 2004
Version 5.0

Project Architecture - 29

Slide 30



Some Architectural Patterns

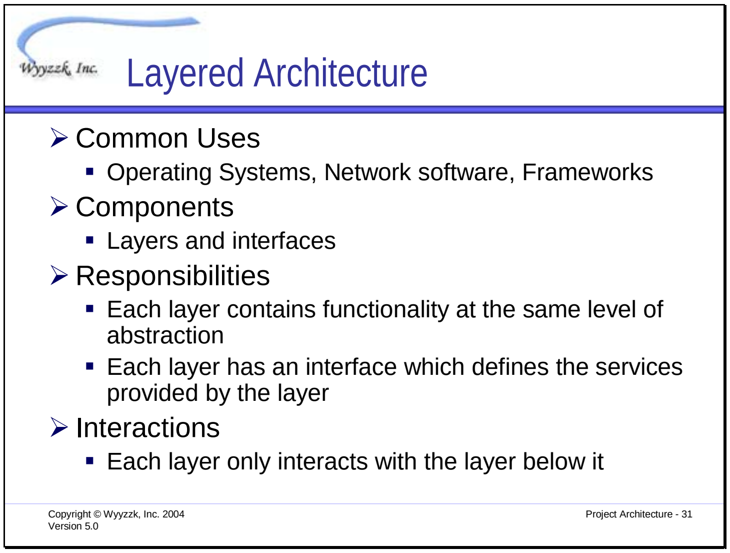
- In the rest of this section, we'll look at some common architectural patterns
  - Common uses
  - The components
  - The responsibilities of the components
  - The interactions between components
  - Examples
  - Strengths and weaknesses of the pattern

Copyright © Wyyzzk, Inc. 2004
Version 5.0

Project Architecture - 30

One way to pick an architectural pattern is to find one that is commonly used for your kind of application. For example, if you are working on writing a new operating system, you will certainly consider a layered architecture for the project, because layered architectures are commonly used for operating systems. You could invent some other architecture, but a layered architecture is known to work very well for this kind of application and you will have plenty of other problems to solve where you can apply your skill and creativity. Don't reinvent the wheel if you don't have to. By using what is known to work, you eliminate certain categories of failure points from your application. Each architectural pattern is good for certain things (it solves certain problems) and not so good for others.

Slide 31

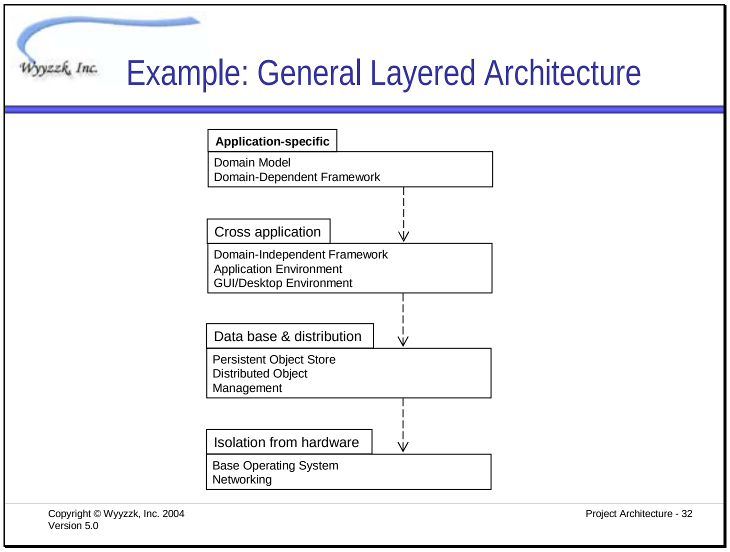


Layered Architecture

- 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

Version 5.0

Slide 32



Example: General Layered Architecture

Application-specific
Domain Model
Domain-Dependent Framework

Cross application
Domain-Independent Framework
Application Environment
GUI/Desktop Environment

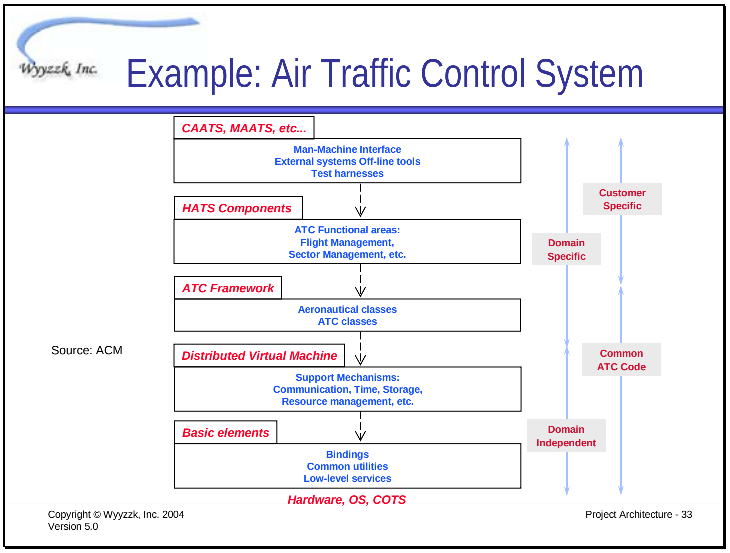
Data base & distribution
Persistent Object Store
Distributed Object
Management

Isolation from hardware
Base Operating System
Networking

Version 5.0

If the application specific layer needs something at the operating system level, the request has to go first to the cross-application layer, then to the database and distribution layer, then finally to the OS layer. The response has to return by the same path (but in reverse).

Slide 33



Wyyzzk, Inc.

Example: Air Traffic Control System

CAATS, MAATS, etc...

Man-Machine Interface
External systems Off-line tools
Test harnesses

HATS Components

ATC Functional areas:
Flight Management,
Sector Management, etc.

ATC Framework

Aeronautical classes
ATC classes

Distributed Virtual Machine

Support Mechanisms:
Communication, Time, Storage,
Resource management, etc.

Basic elements

Bindings
Common utilities
Low-level services

Hardware, OS, COTS

Customer Specific

Domain Specific

Common ATC Code

Domain Independent

Source: ACM

Copyright © Wyyzzk, Inc. 2004
Version 5.0

Project Architecture - 33

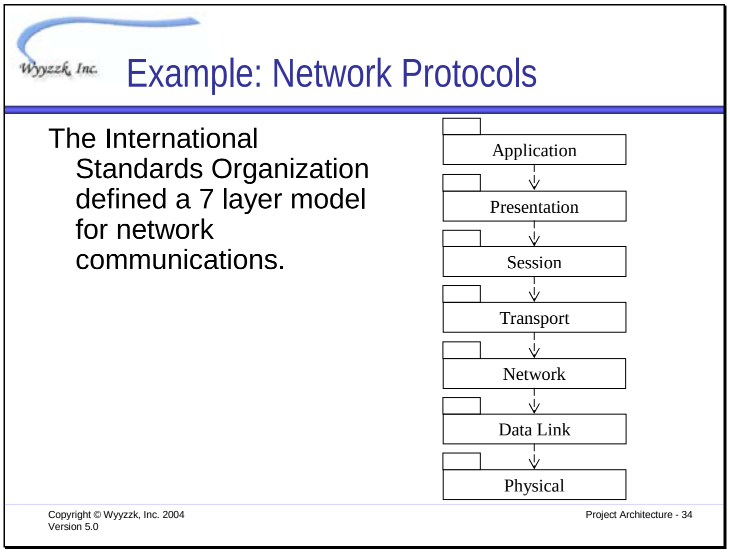
This example was shared with me by a good friend at Rational Software who worked on the system. It was also written up for the ACM.

CAATS = Canadian Air Air Traffic System
HATS = ? Air Traffic System

You see here that you can look at the layers a couple of different ways. One is to specify layers that are domain independent (not part of an air traffic control system) versus those that are specific to the domain. Another way of looking at it is to say that some layers are common across all air traffic control systems, and others are specific to a particular customer. Both viewpoints may be needed at various points in time.

ATC = Air Traffic Control
OS = Operating System
COTS = Commercial Off The Shelf
ACM = Association for Computing Machinery

Slide 34



Wyyzzk, Inc.

Example: Network Protocols

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

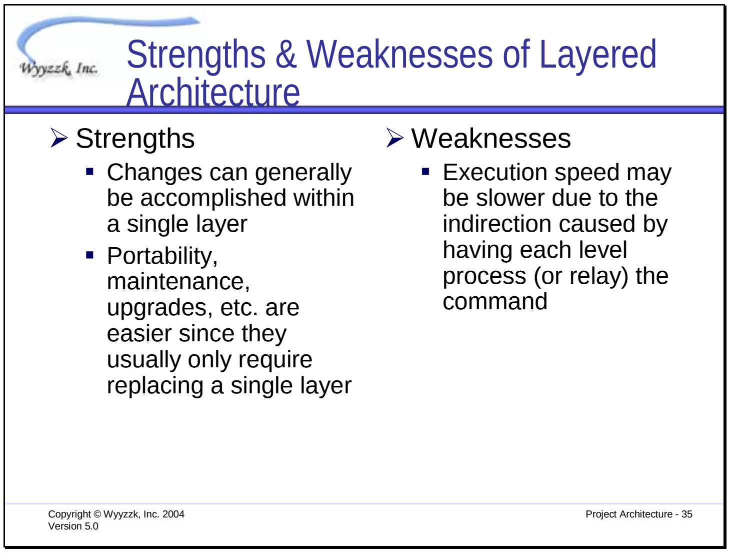
Application
Presentation
Session
Transport
Network
Data Link
Physical

Copyright © Wyyzzk, Inc. 2004
Version 5.0

Project Architecture - 34

Layered architectures are quite common, as you see from the several examples presented here. Note that the communication is always to one layer below. There is no communication that skips around a layer. The rule for layered architecture is that each layer only communicates with the layer directly below it.

Slide 35



Wyyzzk, Inc.

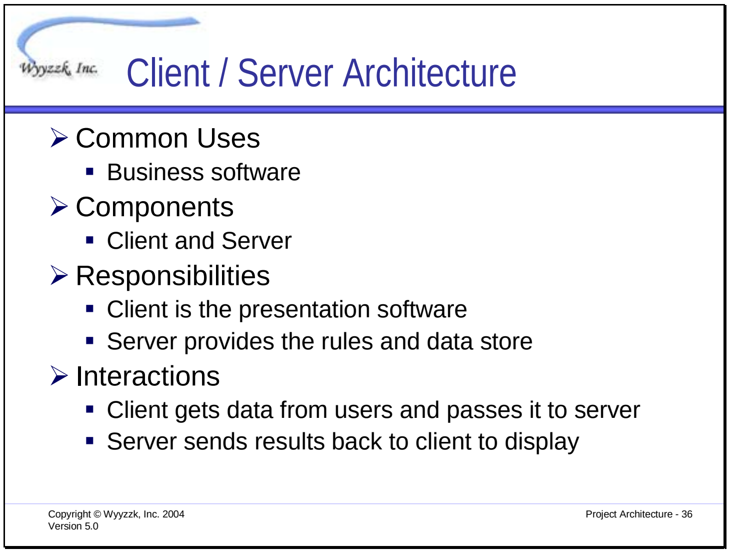
Strengths & Weaknesses of Layered Architecture

- 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

Copyright © Wyyzzk, Inc. 2004
Version 5.0

Project Architecture - 35

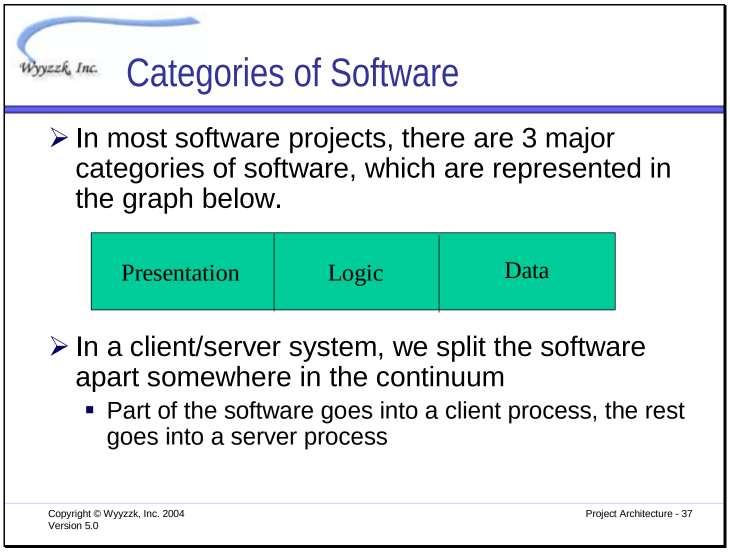
Slide 36



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

Slide 37



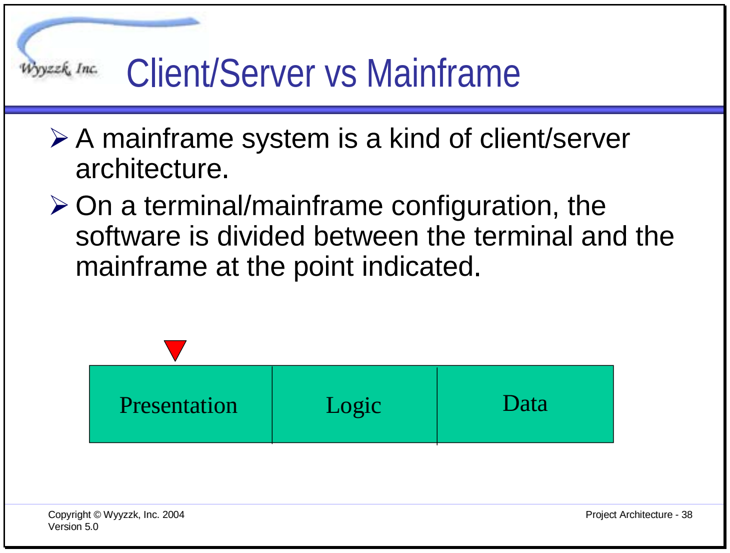
Categories of Software

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

Presentation	Logic	Data

- 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

Slide 38

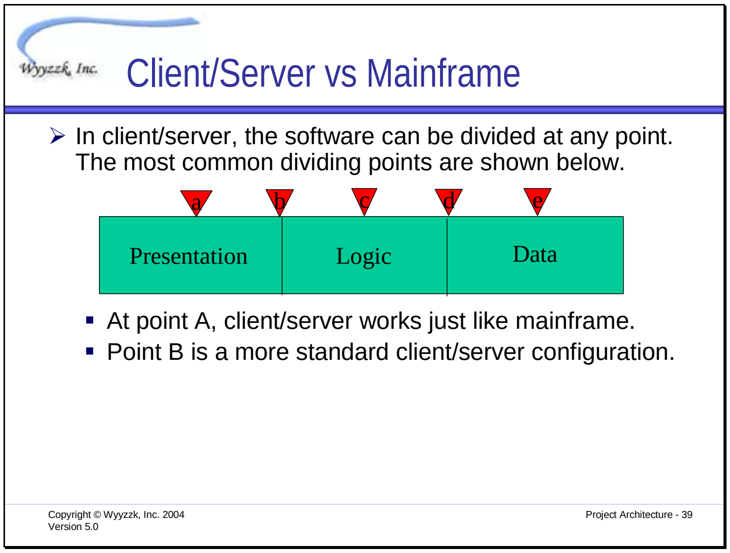


Client/Server vs Mainframe

- 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.

Presentation	Logic	Data

Slide 39



Wyyzzk, Inc.

Client/Server vs Mainframe

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

a
b
c
d
e
Presentation
Logic
Data

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

Version 5.0

Project Architecture - 39

At point A, you have a dumb terminal with all the processing done on the server

At point B, all of the presentation software is on the client

this enables sophisticated GUI's without interaction with the server

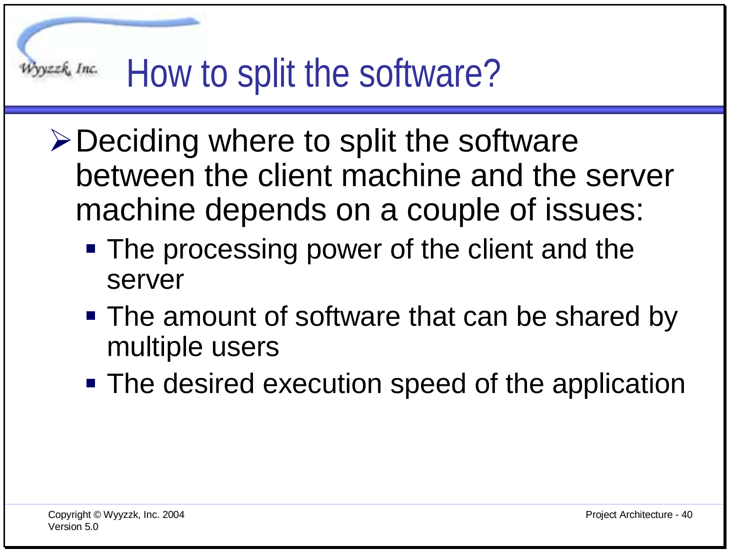
At point C, some application logic resides on the client

At point D, you have a thick client with a database server. The transaction monitor may move to the client, or all transactions may be handled inside the database.

At point E, some of the database moves to the client, usually in the form of client-side caching of data.

There is a great description of these architectures in the book Enterprise Computing with Objects; Shan, Earle.

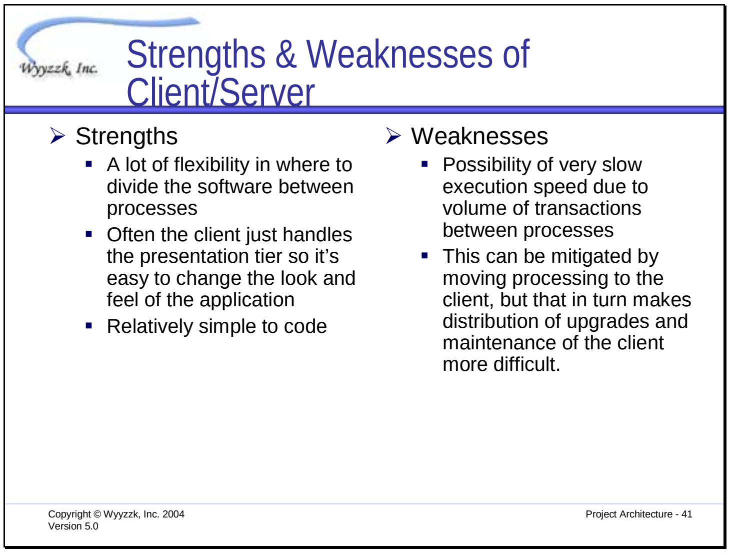
Slide 40



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

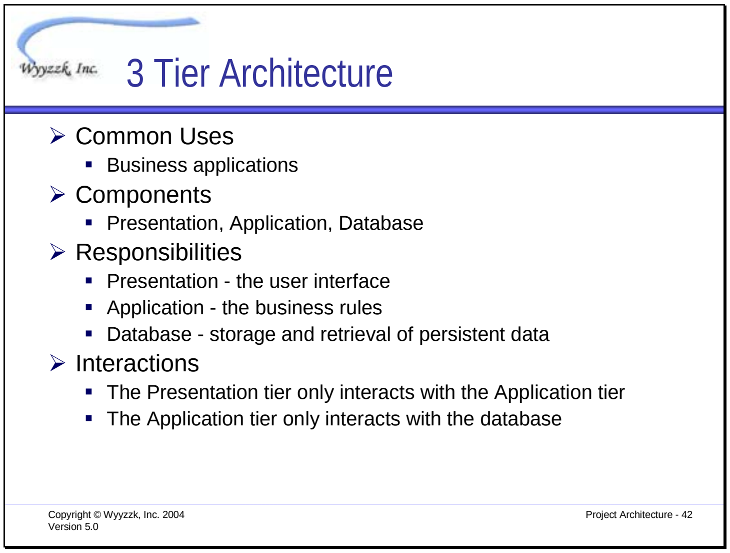
Slide 41



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.

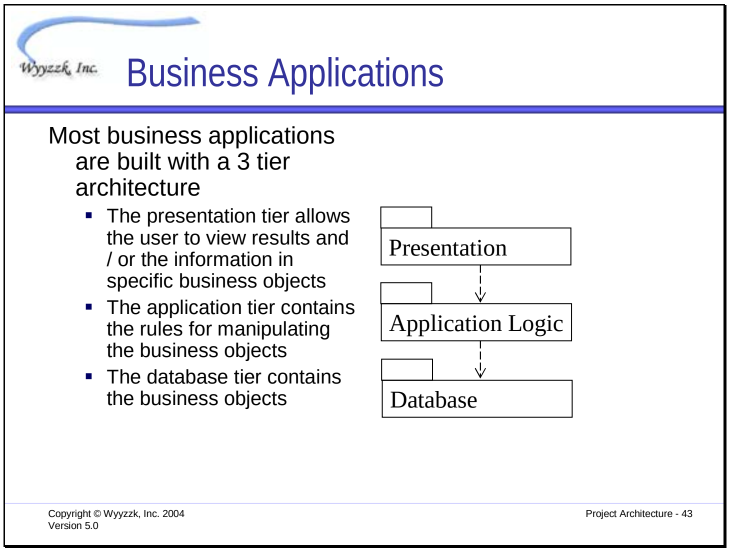
Slide 42



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

Slide 43



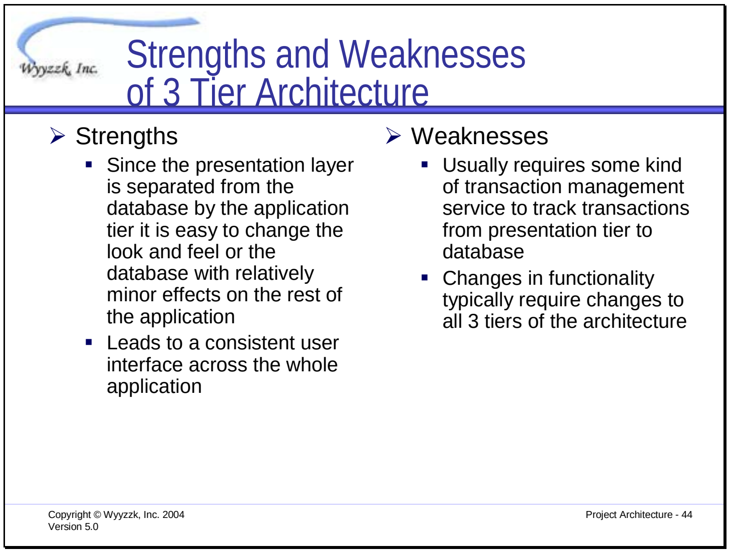
Business Applications

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

Presentation
Application Logic
Database

Slide 44

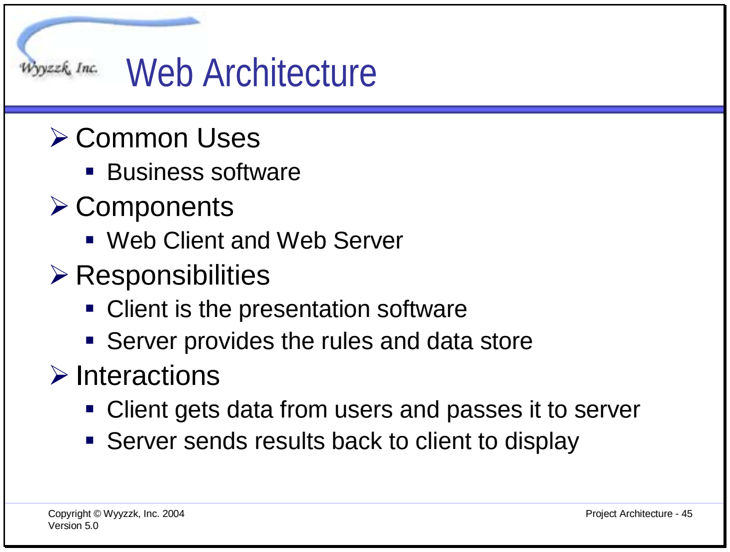


Strengths and Weaknesses of 3 Tier Architecture

- ➢ 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

Project Architecture - 44

Slide 45



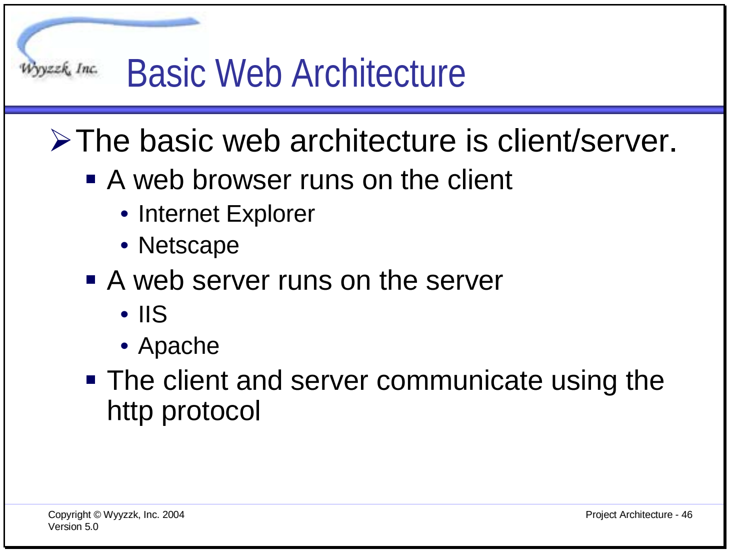
Web Architecture

- ➢ 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

Project Architecture - 45

This is just a quick introduction to web architecture. There is another whole lecture on just web architectures.

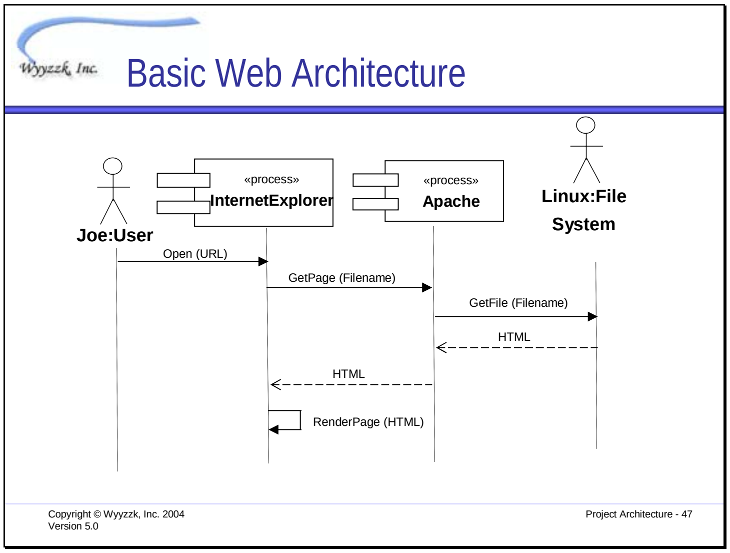
Slide 46



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

Slide 47



Basic Web Architecture

Joe:User

«process»
InternetExplorer

«process»
Apache

Linux:File System

Open (URL)

GetPage (Filename)

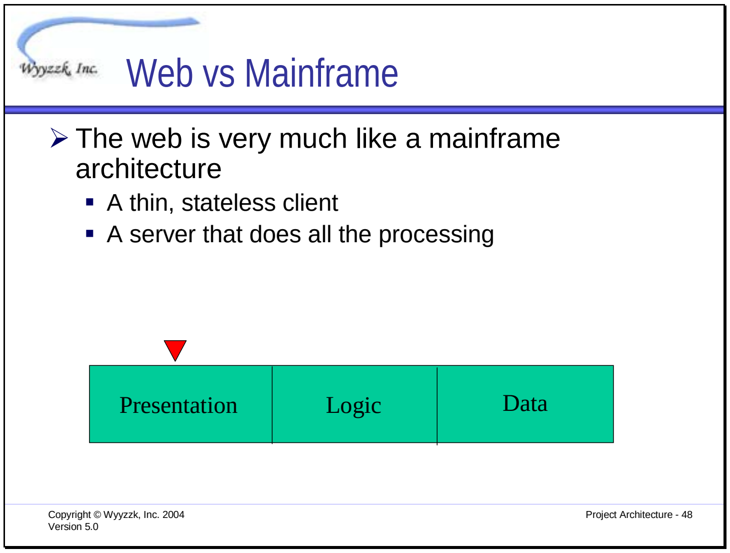
GetFile (Filename)

HTML

HTML

RenderPage (HTML)

Slide 48

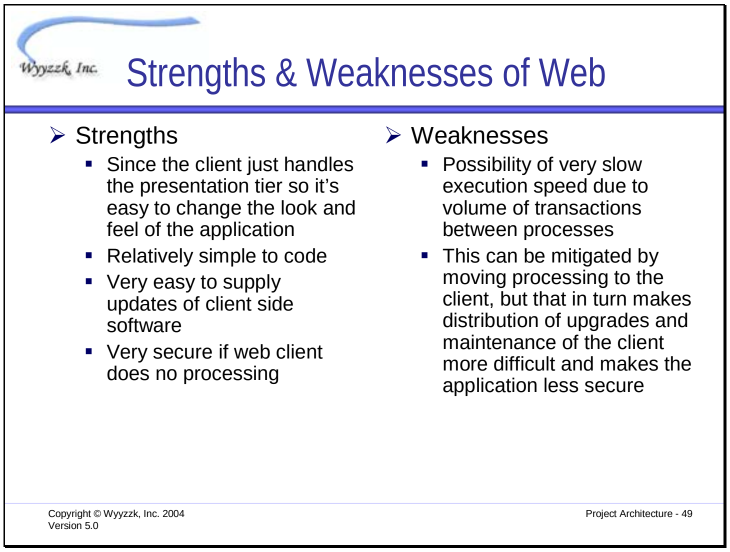


Web vs Mainframe

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

Presentation	Logic	Data

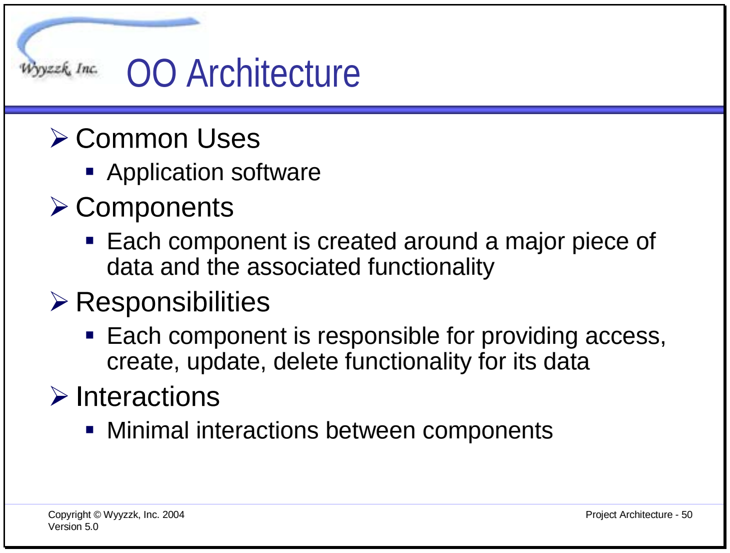
Slide 49



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

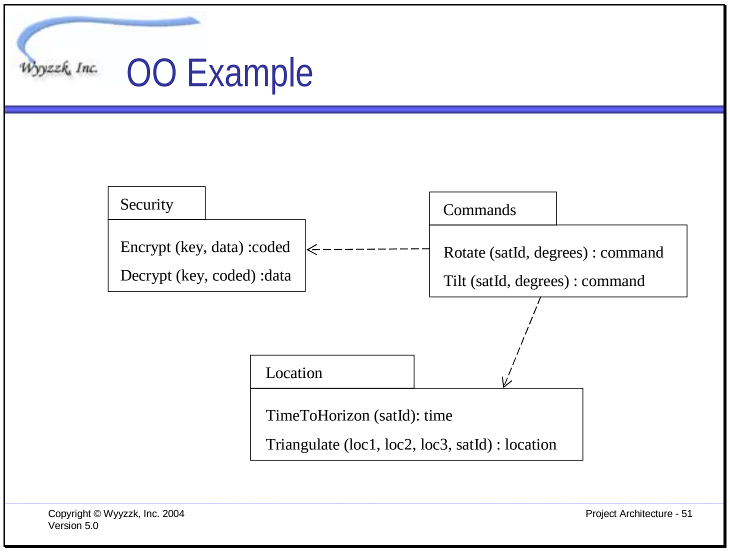
Slide 50



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

Slide 51



OO Example

Security

Encrypt (key, data) :coded

Decrypt (key, coded) :data

Commands

Rotate (satId, degrees) : command

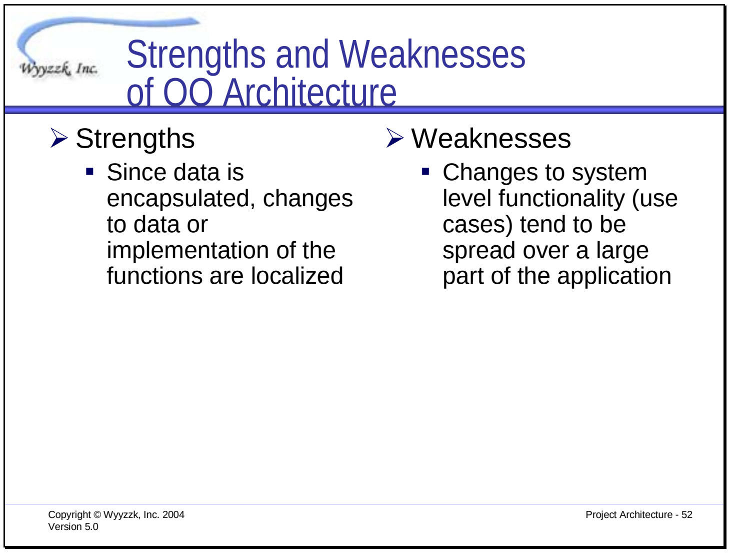
Tilt (satId, degrees) : command

Location

TimeToHorizon (satId): time

Triangulate (loc1, loc2, loc3, satId) : location

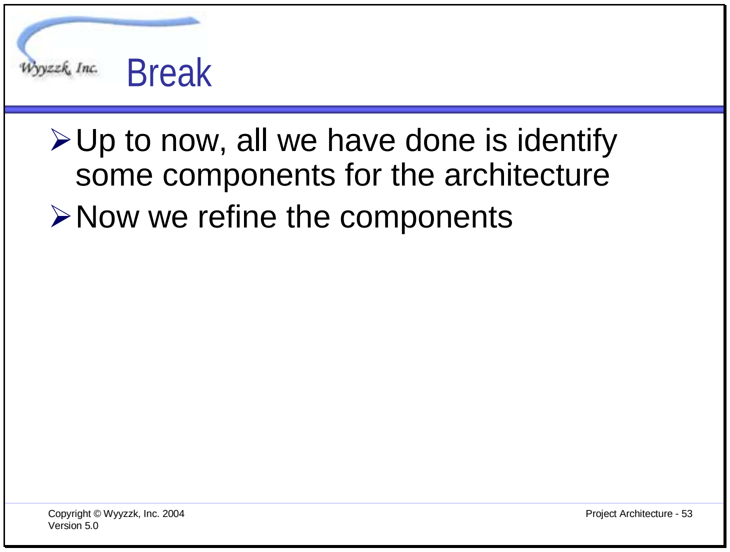
Slide 52



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

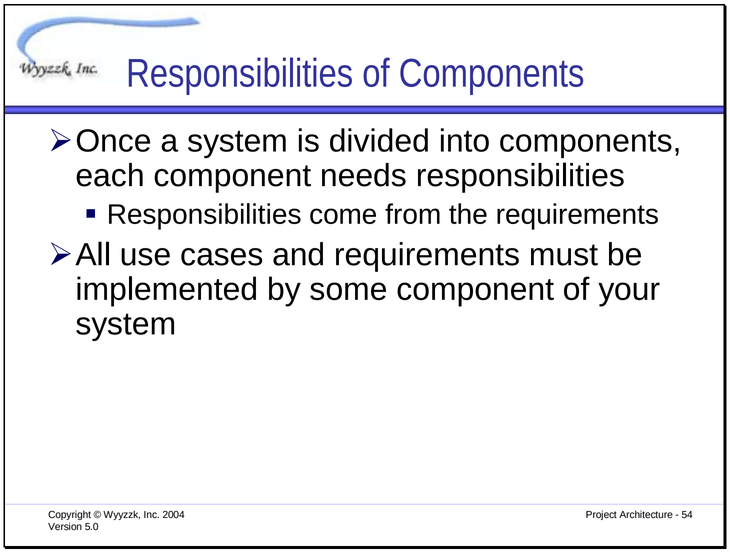
Slide 53



Break

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

Slide 54



Wyyzzk, Inc.

Responsibilities of Components

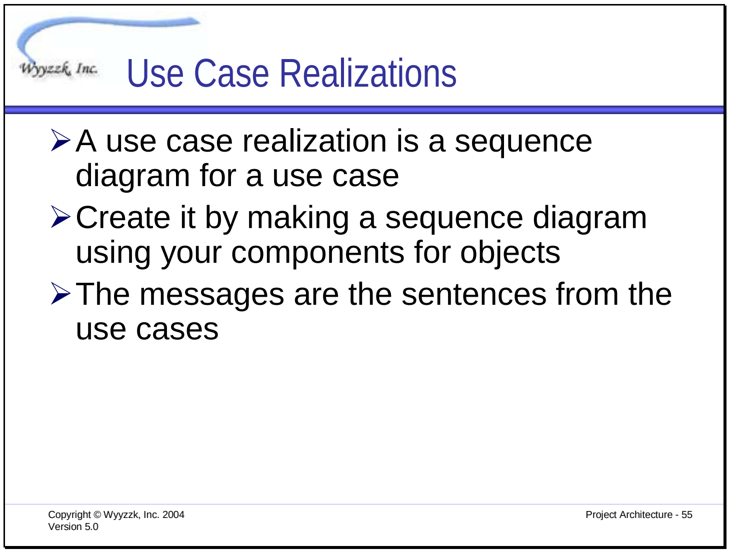
- Once a system is divided into components, each component needs responsibilities
  - Responsibilities come from the requirements
- All use cases and requirements must be implemented by some component of your system

Copyright © Wyyzzk, Inc. 2004
Version 5.0

Project Architecture - 54

Now that the components are identified, we add interfaces to the components. This moves us from Bredemeyer’s conceptual toward the logical architecture. Assigning standard non-functional (usability, reliability, performance, security) or “shall” requirements (the system shall do blah) is fairly easy because they are relatively small. Assigning use cases to components is harder, because often the use case is bigger than one component.

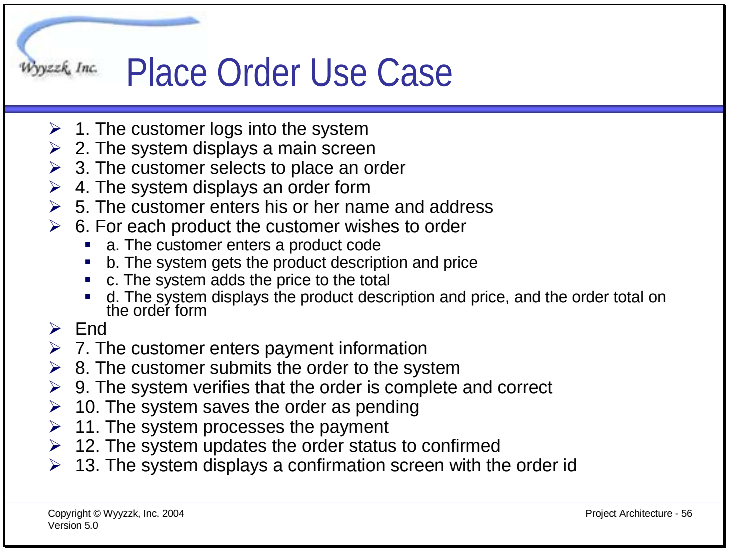
Slide 55



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

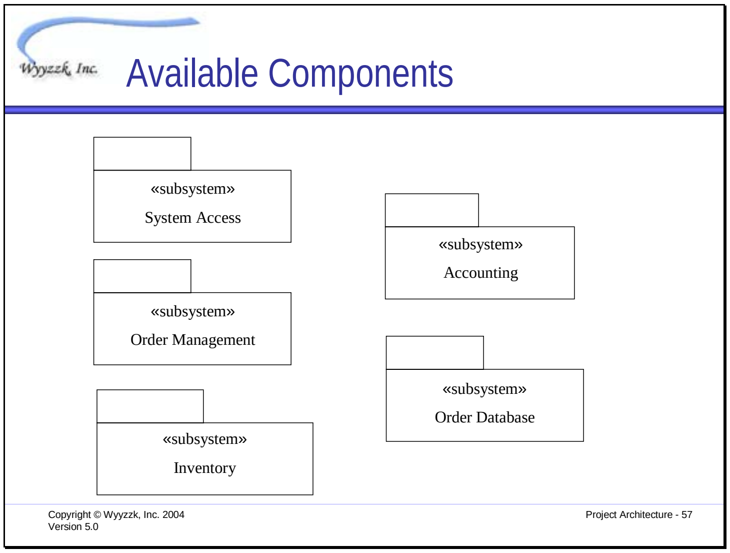
Slide 56



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

Slide 57



Wyyzzk, Inc.

Available Components

«subsystem»
System Access

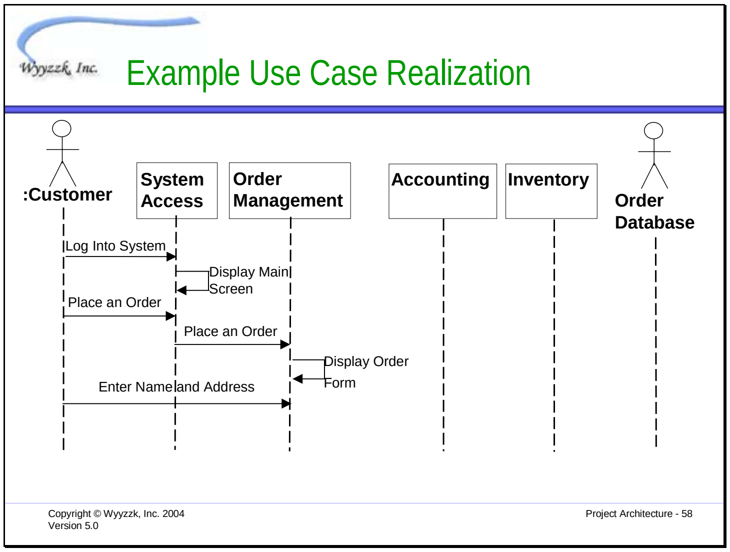
«subsystem»
Order Management

«subsystem»
Inventory

«subsystem»
Accounting

«subsystem»
Order Database

Slide 58



Wyyzzk, Inc.

Example Use Case Realization

:Customer

System Access

Order Management

Accounting

Inventory

Order Database

Log Into System

Display Main Screen

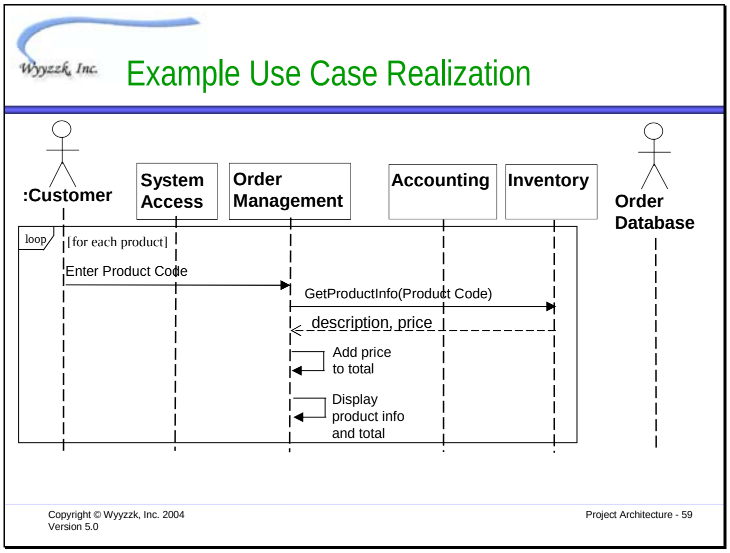
Place an Order

Place an Order

Display Order Form

Enter Name and Address

Slide 59



Example Use Case Realization

:Customer

System Access

Order Management

Accounting

Inventory

Order Database

loop [for each product]

Enter Product Code

GetProductInfo(Product Code)

description, price

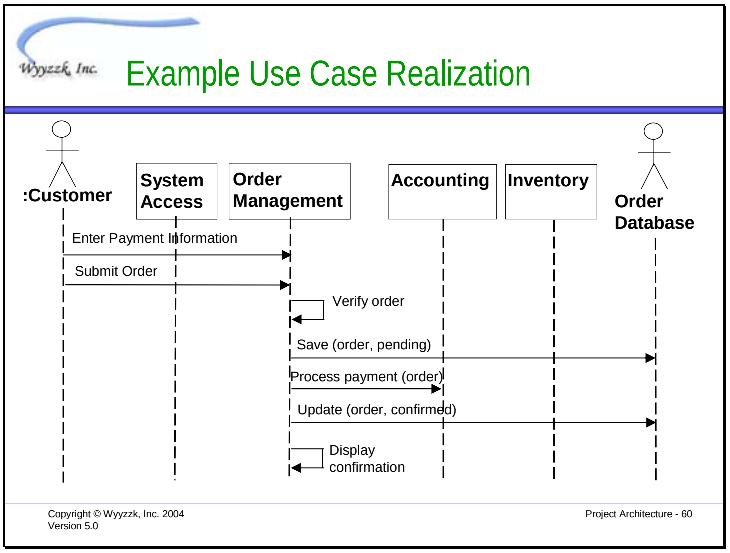
Add price to total

Display product info and total

Version 5.0

Project Architecture - 59

Slide 60



Example Use Case Realization

:Customer

System Access

Order Management

Accounting

Inventory

Order Database

Enter Payment Information

Submit Order

Verify order

Save (order, pending)

Process payment (order)

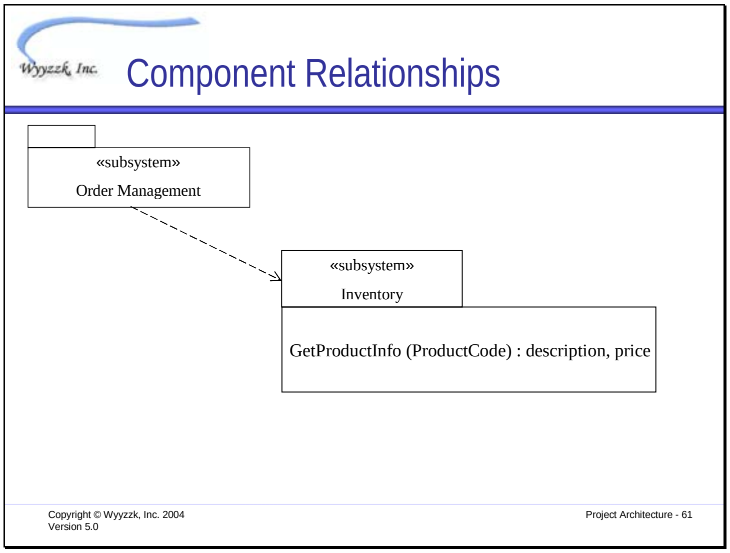
Update (order, confirmed)

Display confirmation

Version 5.0

Project Architecture - 60

Slide 61



Wyyzzk, Inc.

Component Relationships

«subsystem»

Order Management

«subsystem»

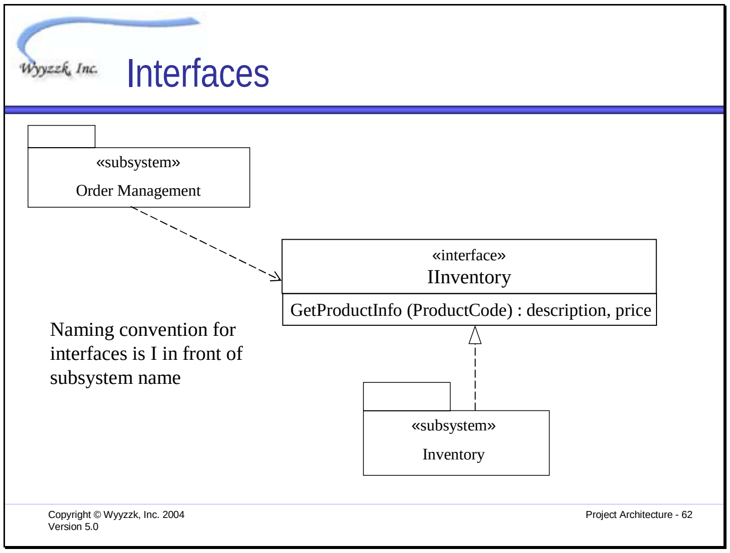
Inventory

GetProductInfo (ProductCode) : description, price

Project Architecture - 61

The sequence diagram shows you the operations for each component and the relationships between the components. If there is a message passed between two components, then you will have a dependency relationship between those components. The arrow on the dependency relationship points the same direction as the message. Notice that the dashed arrow on the sequence diagram shows the return of data. This is indicated in the text of the message shown above, where the returned data comes after the colon.

Slide 62



Wyyzzk, Inc.

Interfaces

«subsystem»
Order Management

«interface»
IInventory

GetProductInfo (ProductCode) : description, price

Naming convention for interfaces is I in front of subsystem name

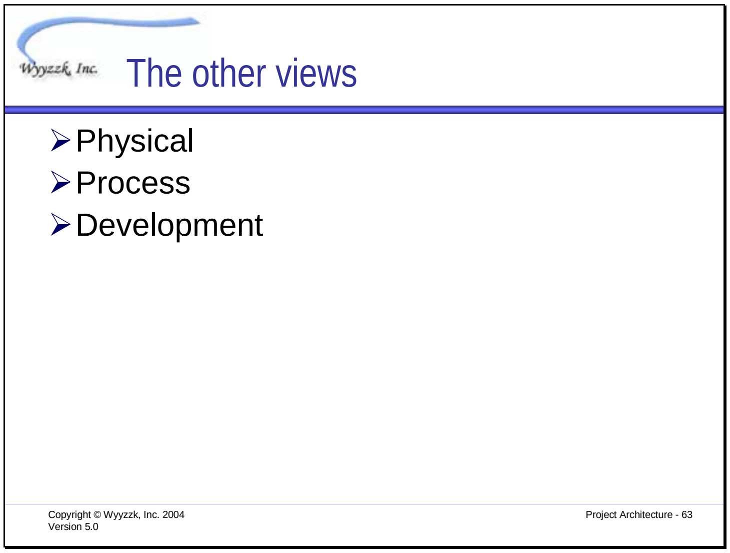
«subsystem»
Inventory

Version 5.0

An alternate way of showing component responsibilities is to use interfaces. Here the message is put in the interface rather than in the component itself. Most architects will show component responsibilities using interfaces rather than putting the operations directly in the component. Using interfaces allows you to plug in any component to the interface.

The dashed line with the triangle head is called realizes or implements. This tells me that the subsystem Inventory has to provide an implementation for all of the operations in the IInventory interface. The other dashed line with the stick arrowhead is dependency or uses. This tells me that the subsystem Order Management uses (makes calls to) the operations in the IInventory interface.

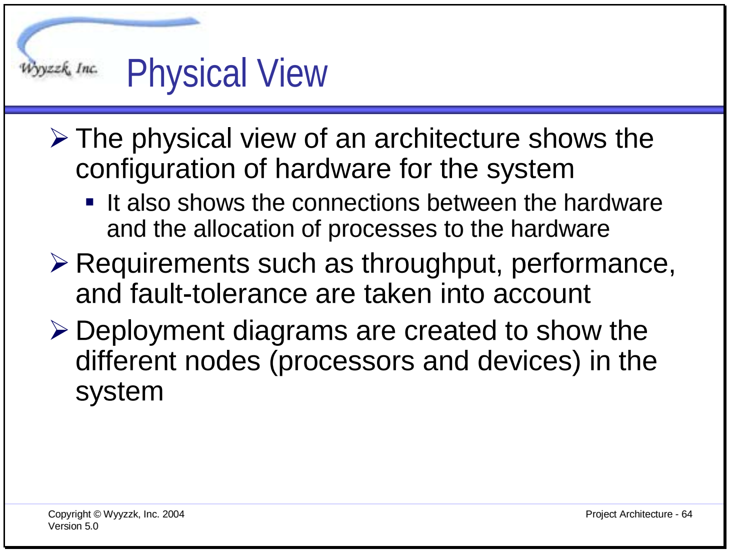
Slide 63



The other views

- Physical
- Process
- Development

Slide 64



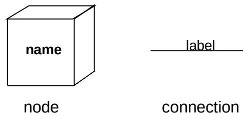
Physical View

- 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

Slide 65

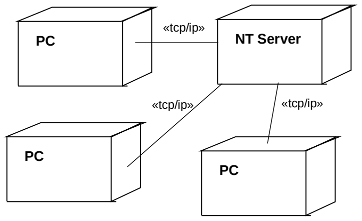
Notation For Deployment Diagrams

- 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

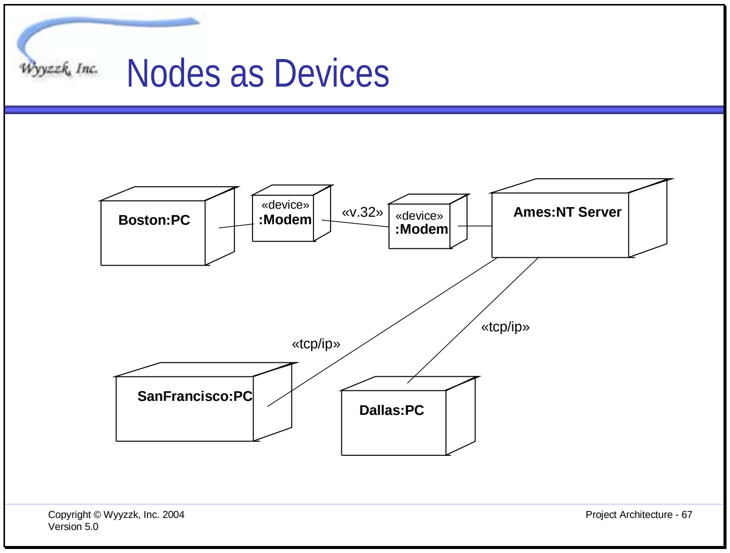


Slide 66

Nodes and Connectors



Slide 67



Nodes as Devices

Boston:PC

«device»
:Modem

«v.32»

«device»
:Modem

Ames:NT Server

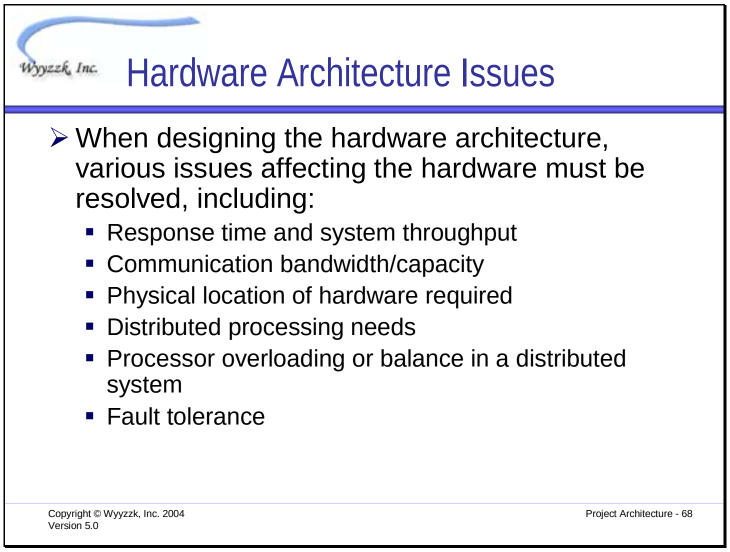
«tcp/ip»

«tcp/ip»

SanFrancisco:PC

Dallas:PC

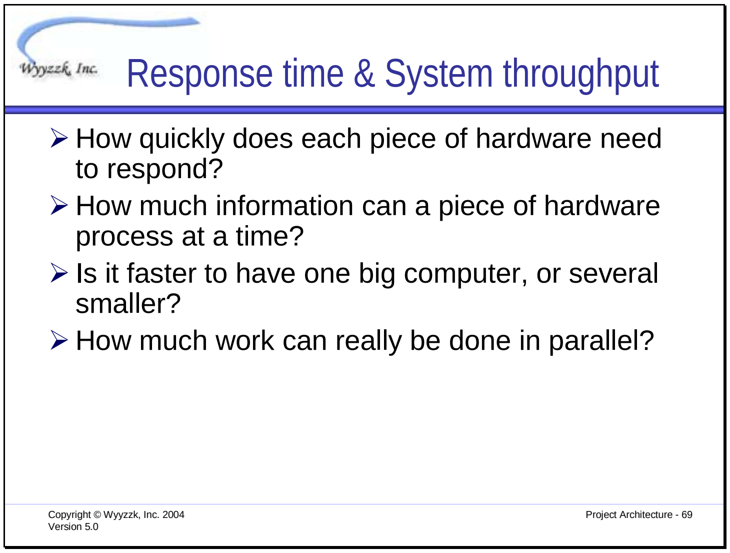
Slide 68



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

Slide 69

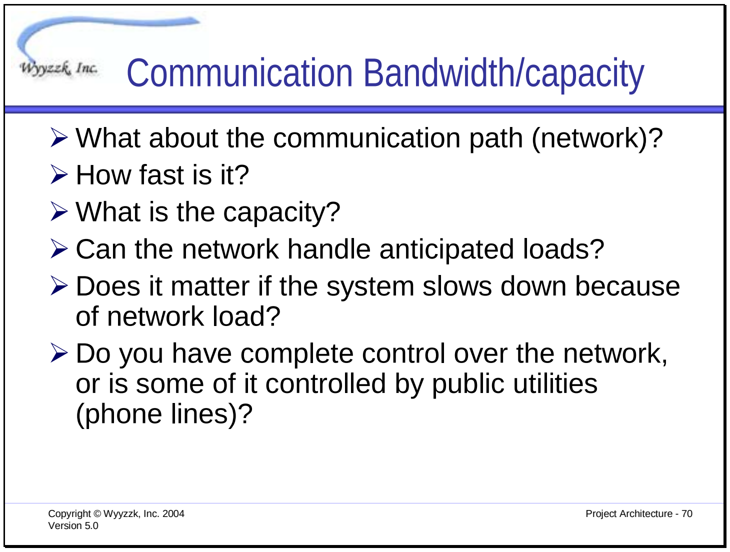


Response time & System throughput

- ➢ 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?

Version 5.0

Slide 70

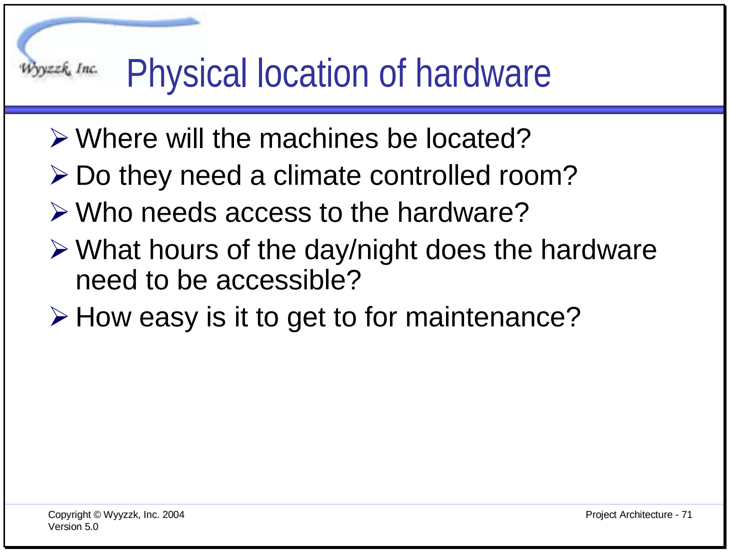


Communication Bandwidth/capacity

- ➢ 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)?

Version 5.0

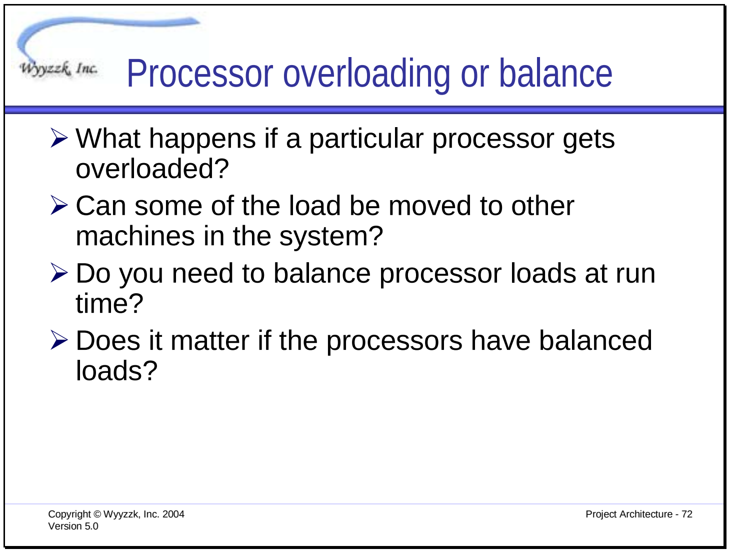
Slide 71



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?

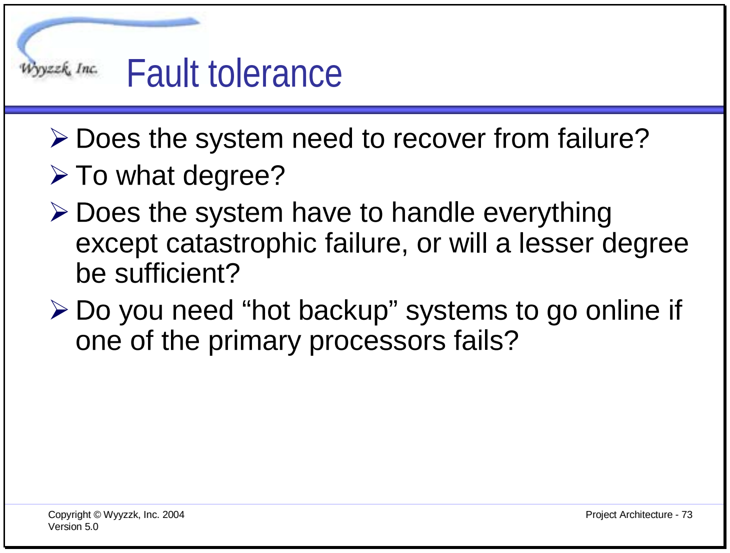
Slide 72



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?

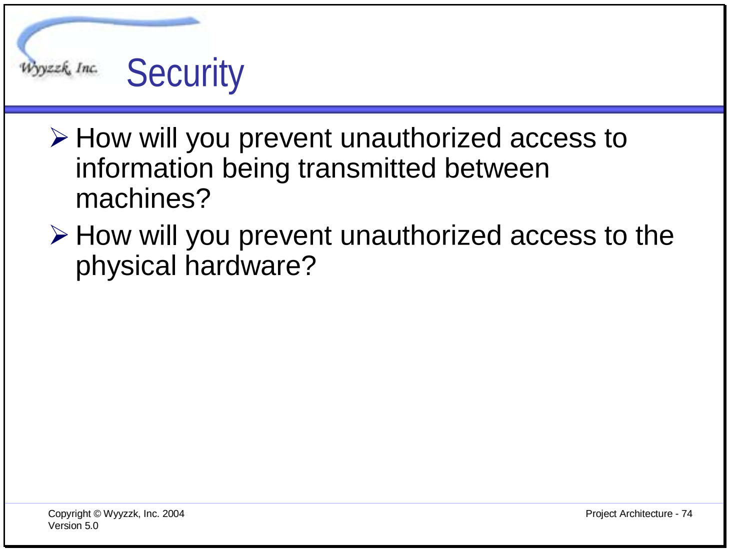
Slide 73



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?

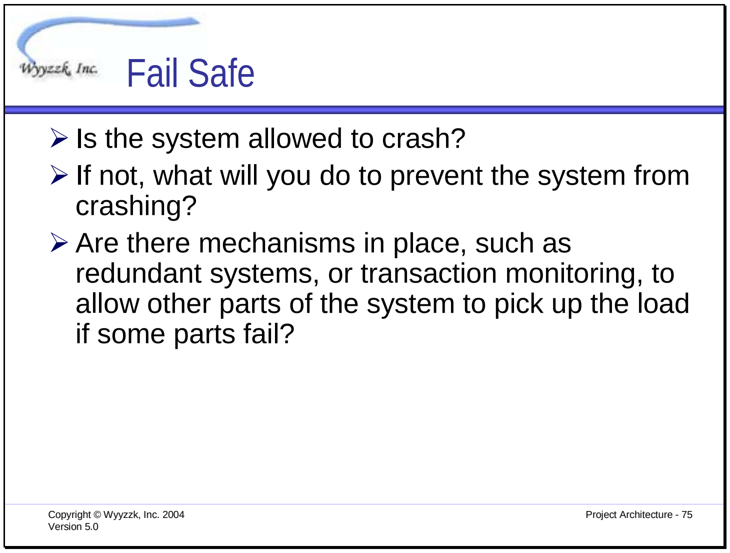
Slide 74



Security

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

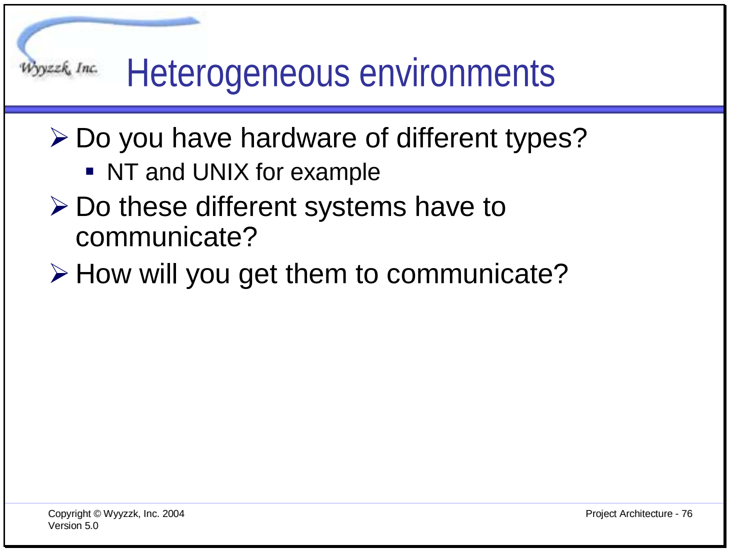
Slide 75



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?

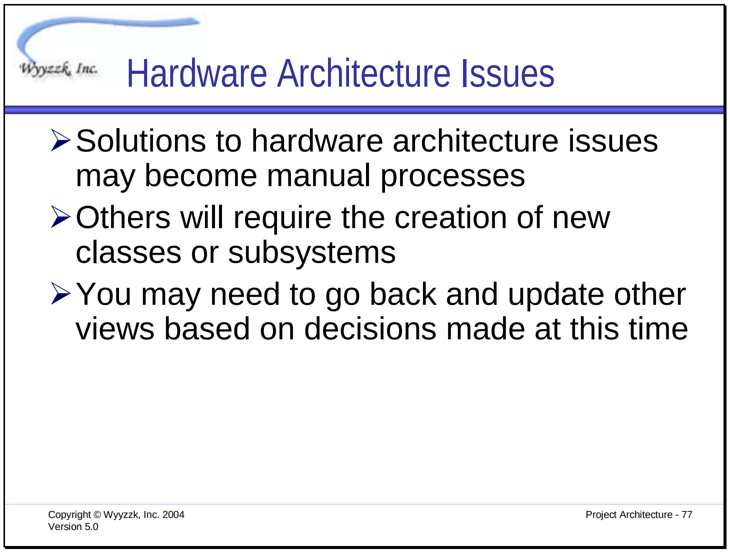
Slide 76



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?

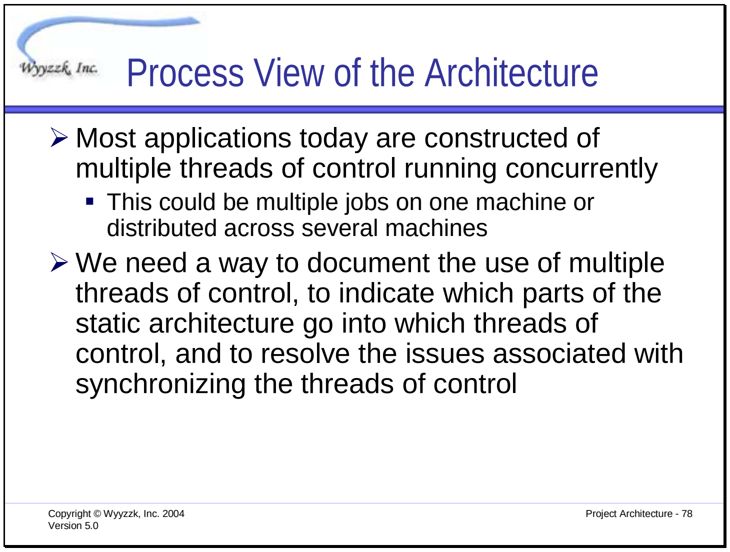
Slide 77



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

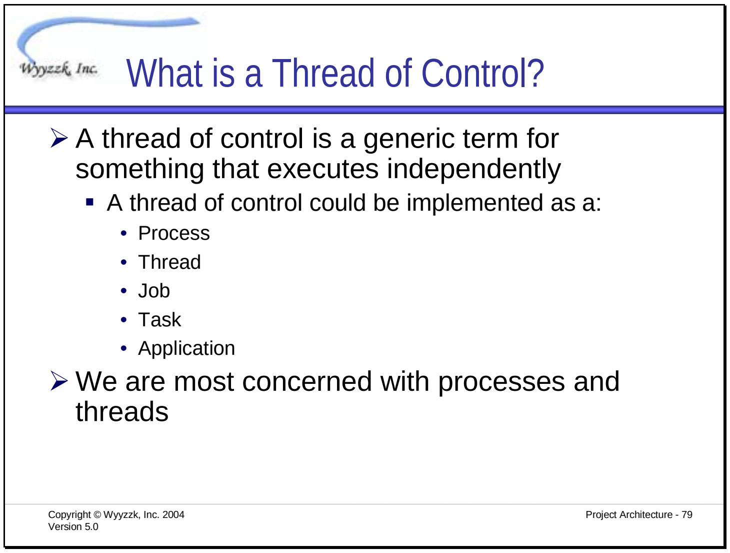
Slide 78



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

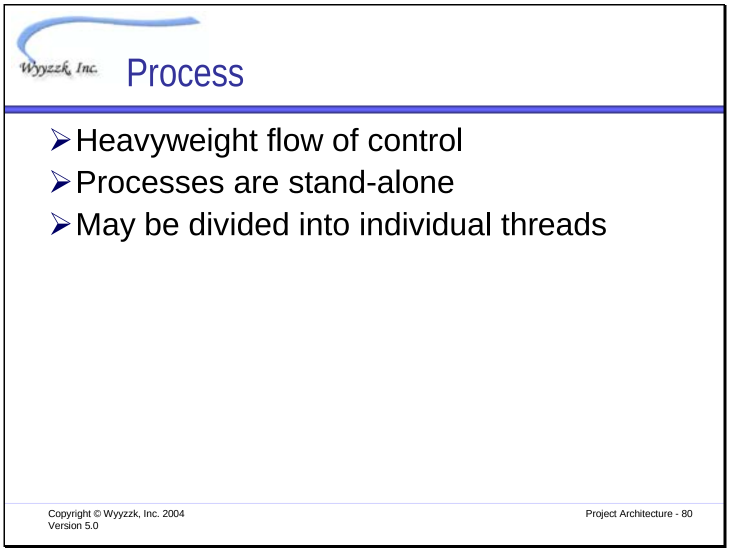
Slide 79



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

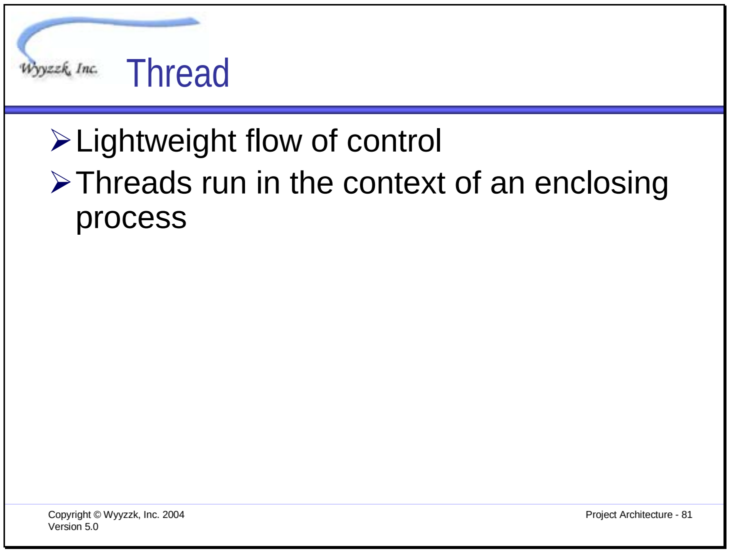
Slide 80



Process

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

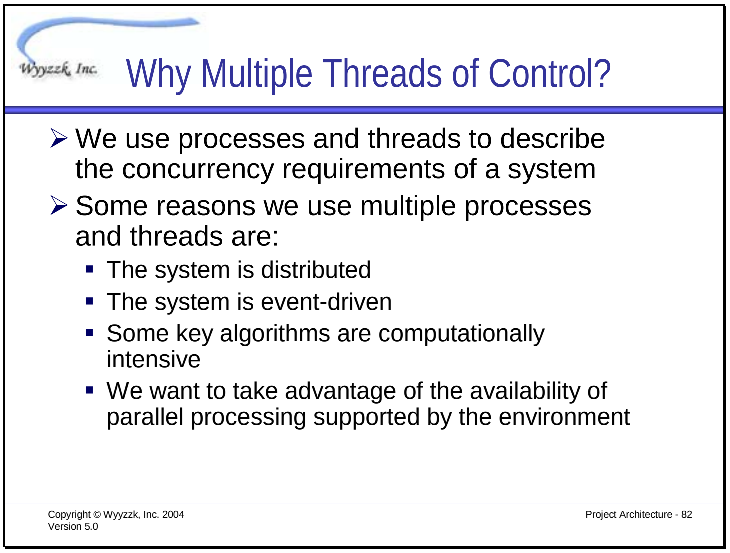
Slide 81



Thread

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

Slide 82



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

Copyright © Wyyzzk, Inc. 2004
Version 5.0

Project Architecture - 82

Concurrency requirements define the extent to which parallel execution of tasks is required for the system. These requirements help shape the architecture.

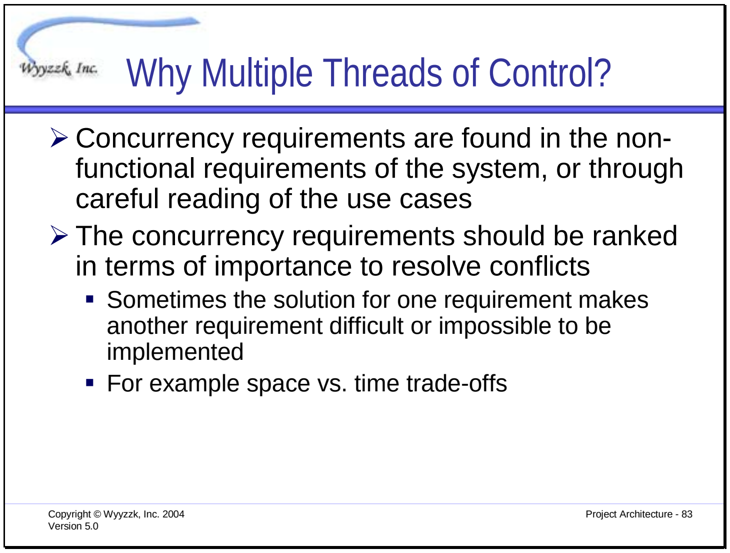
A system whose behavior must be distributed across processors or nodes virtually requires a multi-process architecture. A system which uses some sort of Database Management System or Transaction Manager also must consider the processes which those major subsystems introduce.

If dedicated processors are available to handle events, a multi-process architecture is probably best. On the other hand, to ensure events are handled, a uni-process architecture may be needed to circumvent the ‘fairness’ resource sharing algorithm of the operating system: it may be necessary for the application to monopolize resources by creating a single large process, using threads to control execution within the process.

In order to provide good response times, it may be necessary to place computationally intensive activities in a process or thread of their own so that the system is still able to respond to user inputs while computation takes place, albeit with fewer resources. If the operating system or environment does not support threads (lightweight processes) there is little point in considering their impact on the system architecture.

The above are mutually exclusive and may conflict with one another. Ranking requirements in terms of importance will help resolve the conflict.

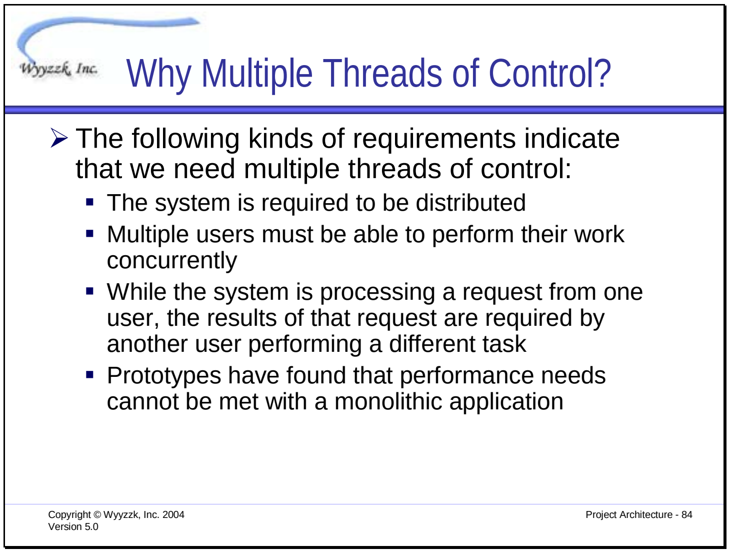
Slide 83



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

Slide 84



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

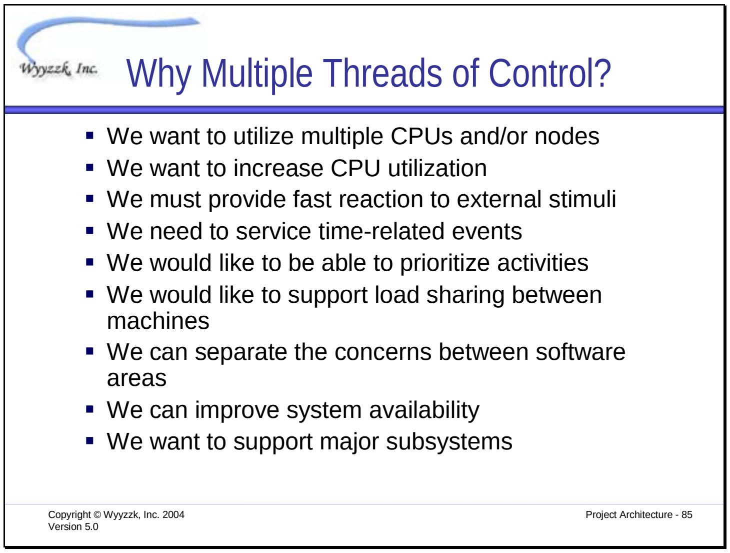
The above concurrency requirements were documented in the Course Registration System Supplemental Specification (see the Course Registration Requirements Document).

The first requirement is typical of any system, but the multi-tier aspects of our planned architecture will require some extra thought for this.

The second requirement demonstrates the need for a shared, independent process managing access to the course offerings.

The third issue will lead us to use some sort of mid-tier caching or preemptive retrieval strategy.

Slide 85



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

For each separate flow 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).

For example, we can say (not necessarily in order of importance) that separate threads of control may be needed to:

Use of multiple CPUs. There may be multiple CPUs in a node or multiple nodes in a distributed system

Increased CPU utilization. Processes can be used to increase CPU utilization by allocating cycles to other activities when a thread of control is suspended

Fast reaction to external stimuli

Service time-related events. Examples: timeouts, scheduled activities, periodic activities

Prioritize activities. Separate processes allows functionality in different processes to be prioritized individually

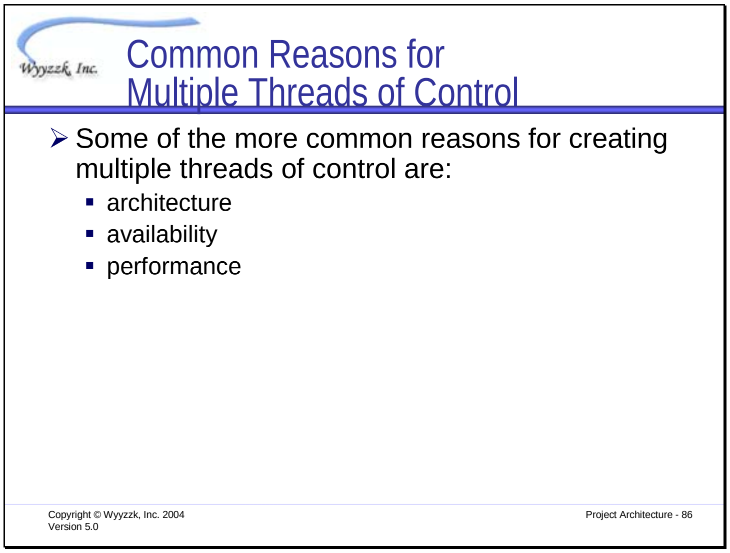
Scalability. Load sharing across several processes and processors

Separation of concerns. Separating concerns between different areas of the software, e.g., safety

Availability. Redundant processes. You can achieve a higher system availability by having backup processes

Support major subsystems. Some major subsystems may required separate processes (e.g., the DBMS, Transaction Manager, etc.)

Slide 86

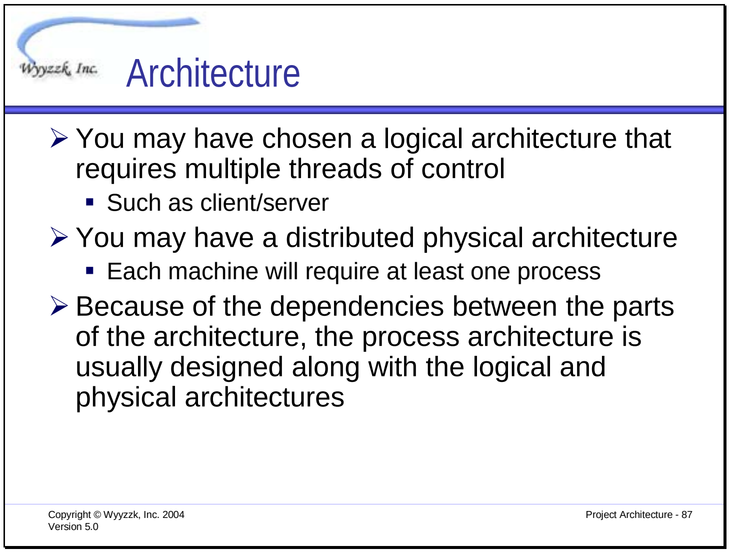


Common Reasons for Multiple Threads of Control

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

Project Architecture - 86

Slide 87

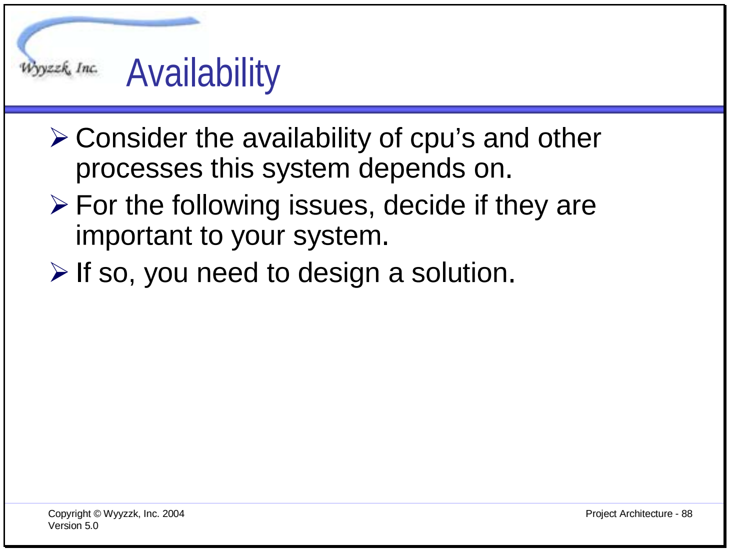


Wyyzzk, Inc.

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

Slide 88

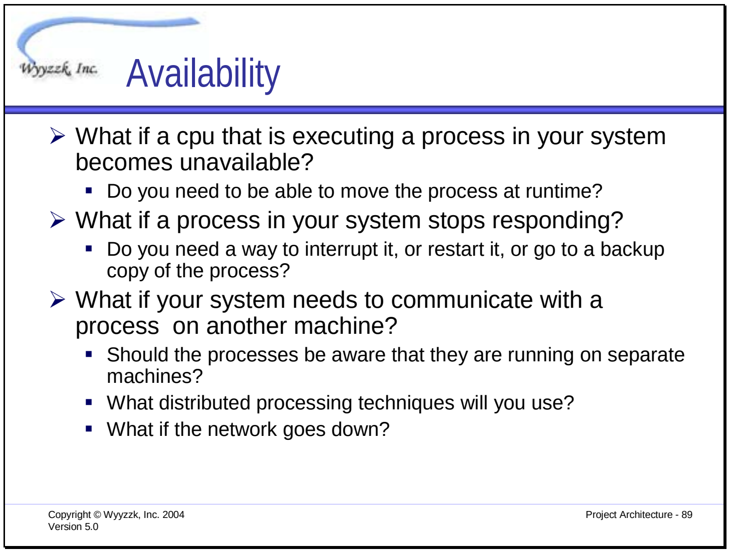


Wyyzzk, Inc.

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.

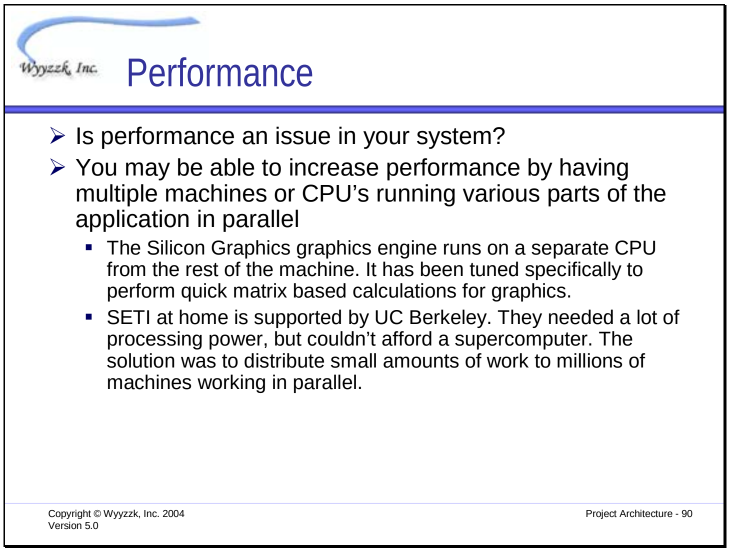
Slide 89



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?

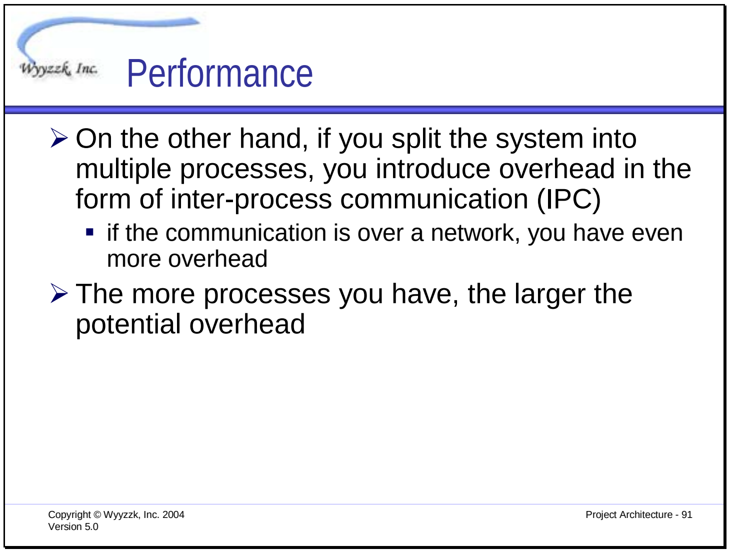
Slide 90



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.

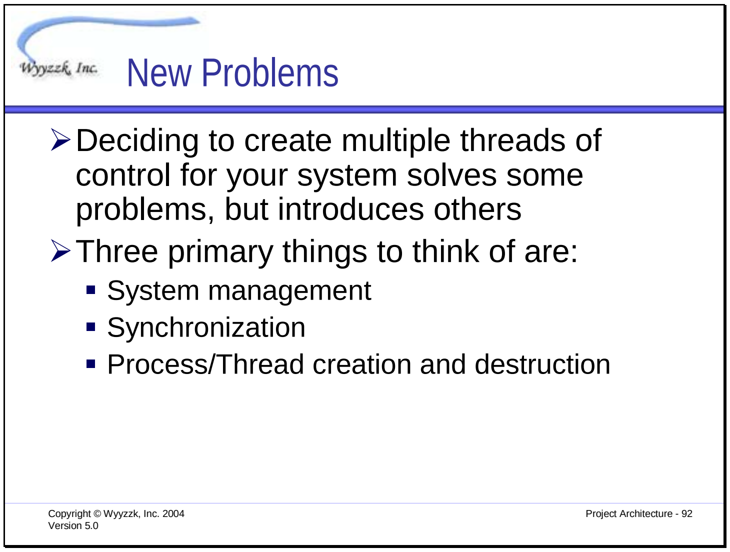
Slide 91



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

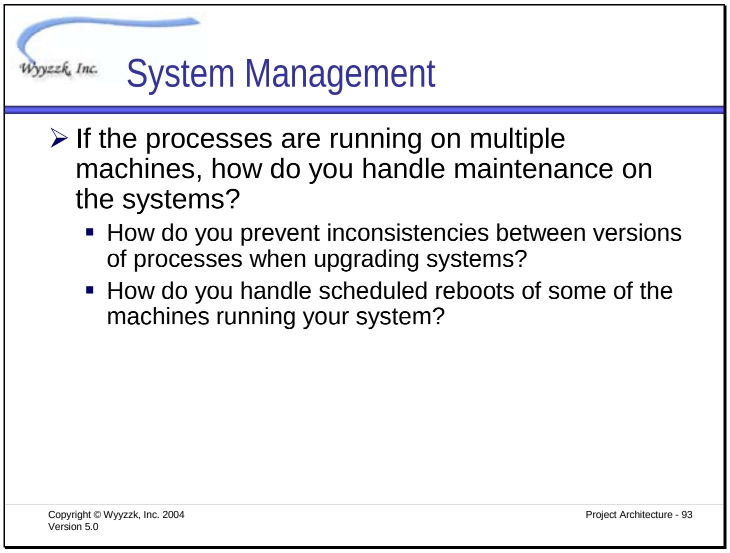
Slide 92



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

Slide 93

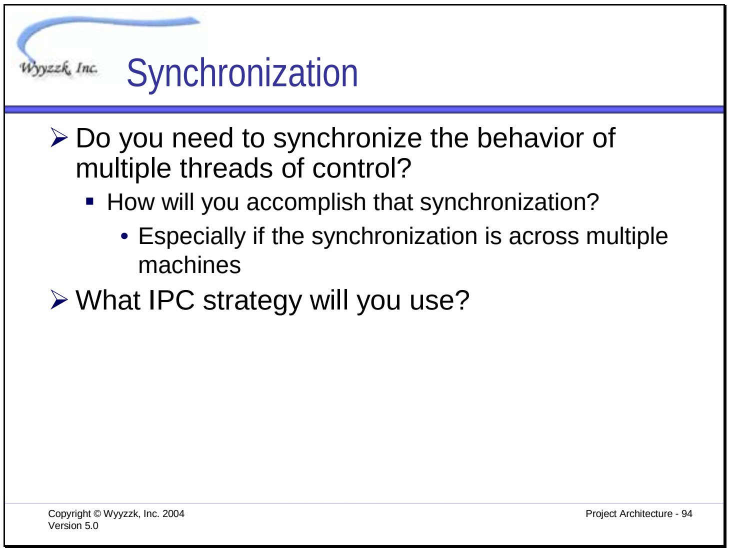


Wyyzzk, Inc.

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?

Slide 94

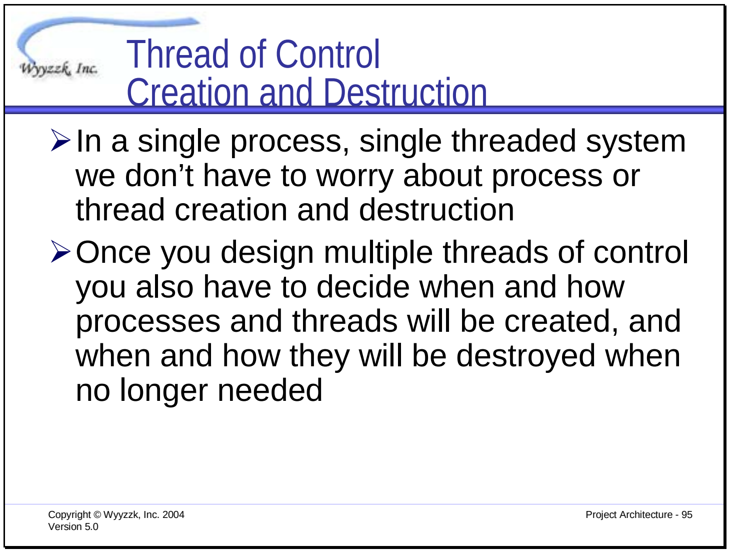


Wyyzzk, Inc.

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?

Slide 95



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

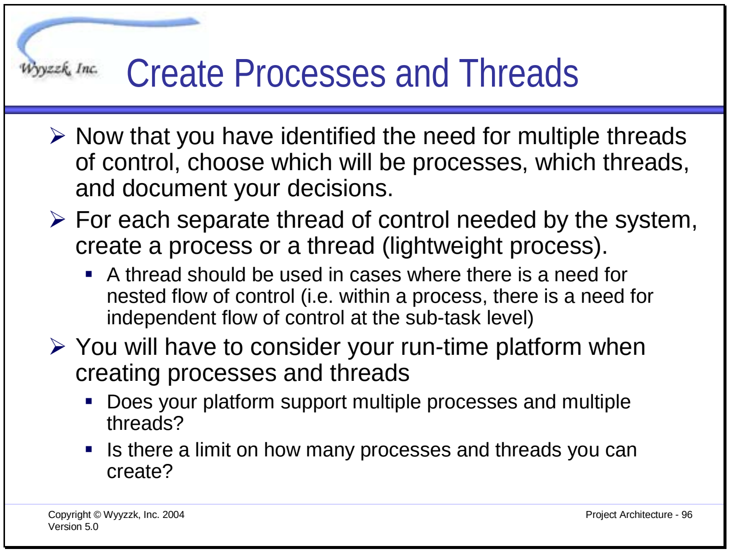
Version 5.0

Project Architecture - 95

Each process or thread of control must be created and destroyed. In a single-process architecture, process creation occurs when the application is started and process destruction occurs when the application ends. In multi-process architectures, new processes (or threads) are typically spawned or forked from the initial process created by the operating system when the application is started. These processes must be explicitly destroyed as well.

The sequence of events leading up to process creation and destruction must be determined and documented, as well as the mechanism for creation and deletion.

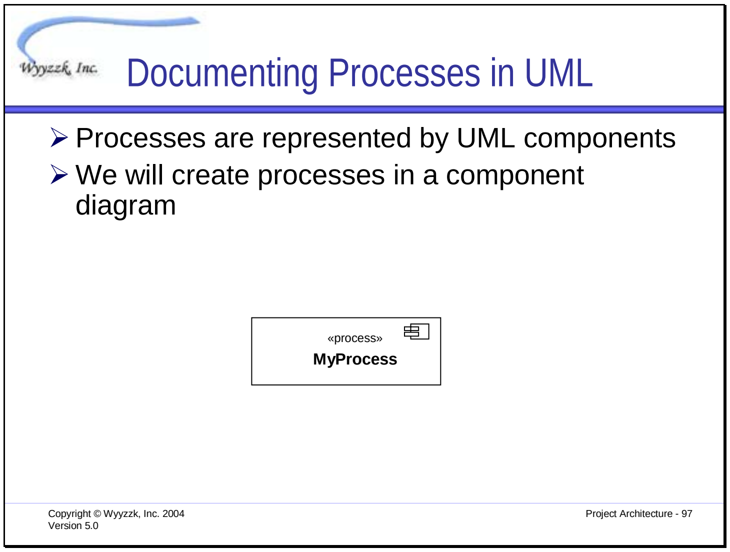
Slide 96



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?

Slide 97

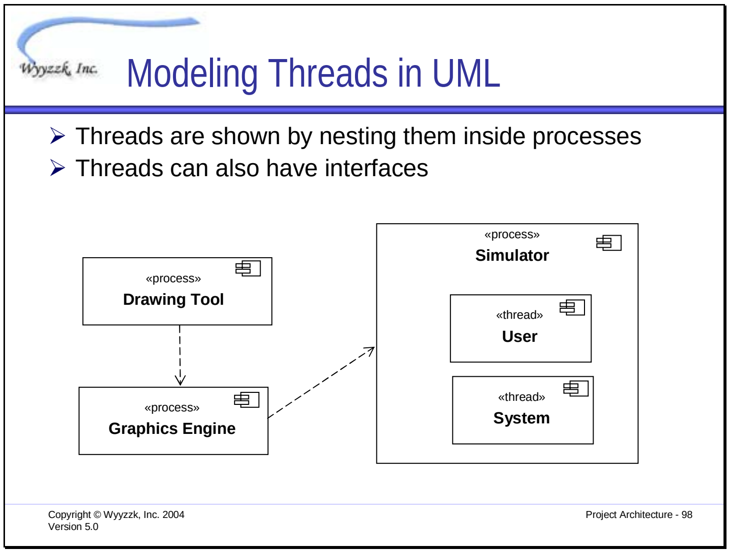


Documenting Processes in UML

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

«process»
MyProcess

Slide 98



Wyyzzk, Inc.

Modeling Threads in UML

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

«process»
Drawing Tool

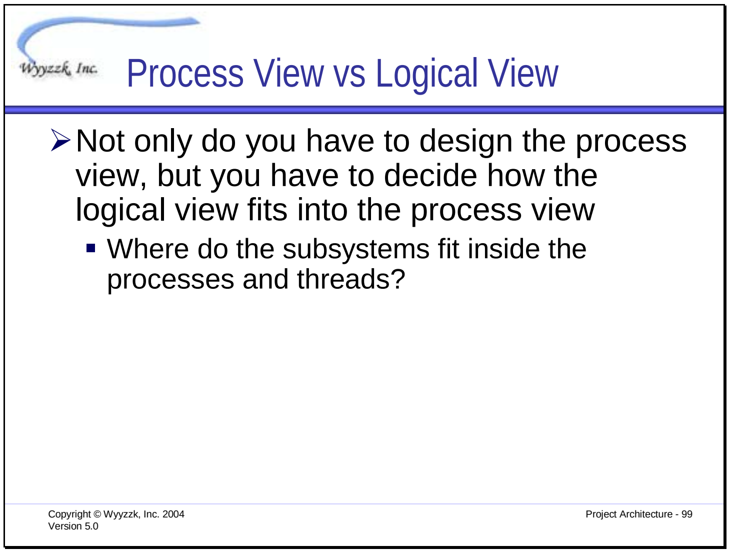
«process»
Graphics Engine

«process»
Simulator

«thread»
User

«thread»
System

Slide 99

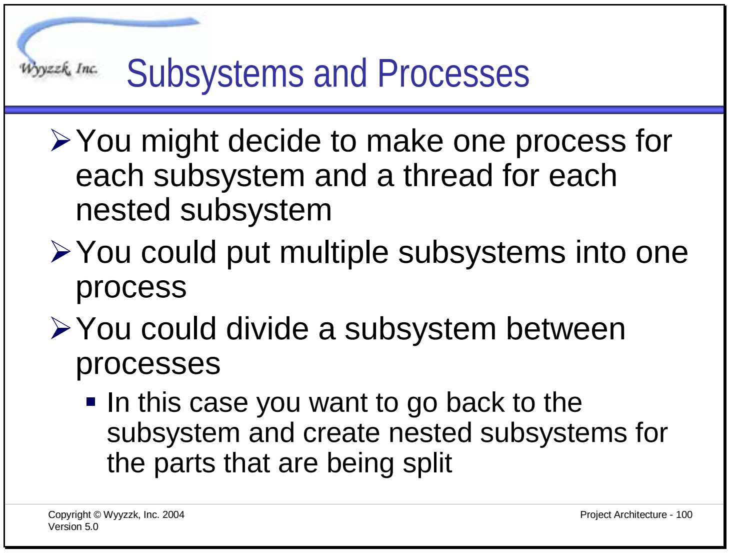


Wyyzzk, Inc.

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?

Slide 100

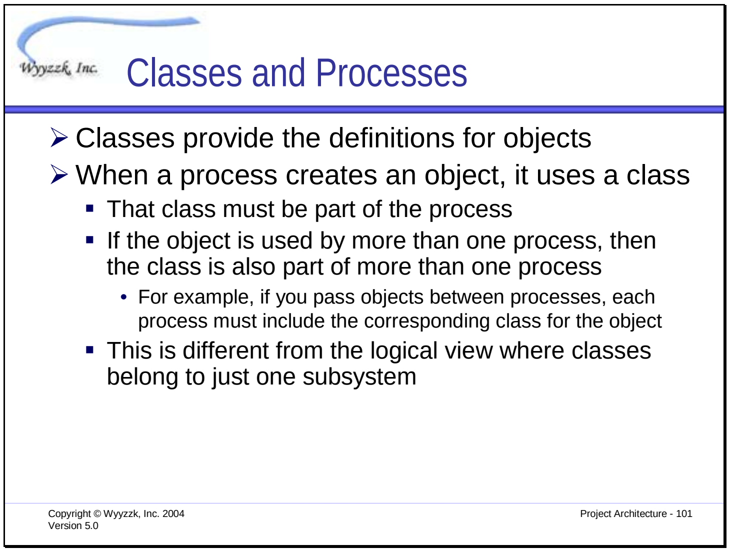


Subsystems and Processes

- 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

Version 5.0
Project Architecture - 100

Slide 101



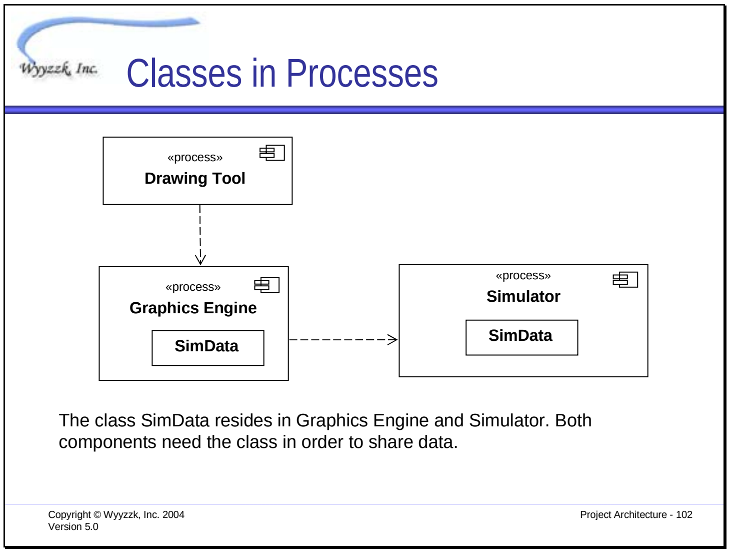
Classes and Processes

- 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

Version 5.0
Project Architecture - 101

When you pass data between processes, you put the data into an object which is defined by a class. That class is included in both processes, because it describes to the processes the data that they are sharing.

Slide 102



Wyyzzk, Inc.

Classes in Processes

«process»
Drawing Tool

«process»
Graphics Engine

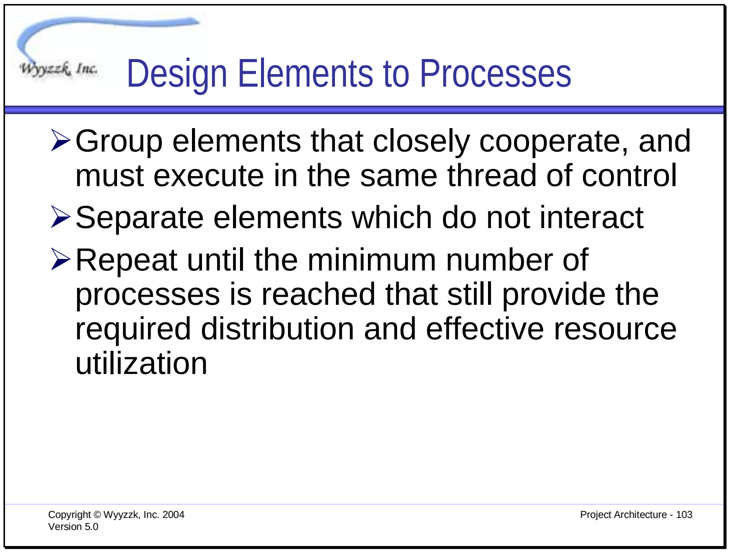
SimData

«process»
Simulator

SimData

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

Slide 103



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

Version 5.0

Project Architecture - 103

Classes and subsystems may be allocated to one or more processes and threads.

Inside-out

Group classes and subsystems together in sets of cooperating elements that (a) closely cooperate with one another and (b) need to execute in the same thread of control.

Consider the impact of introducing inter-process communication into the middle of a message sequence before separating elements into separate threads of control.

Conversely, separate classes and subsystems which do not interact at all, placing them in separate threads of control.

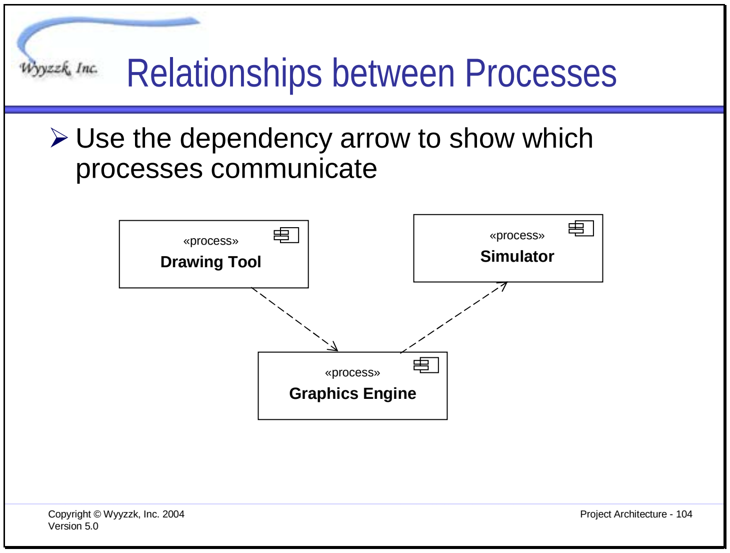
This clustering proceeds until the number of processes has been reduced to the smallest number that still allows distribution and use of the physical resources.

Outside-in

Identify external stimuli to which the system must respond. Define a separate thread of control to handle each stimuli and a separate server thread of control to provide each service.

Consider the data integrity and serialization constraints to reduce this initial set of threads of control to the number that can be supported by the execution environment.

Slide 104



Wyyzk, Inc.

Relationships between Processes

- Use the dependency arrow to show which processes communicate

«process»
Drawing Tool

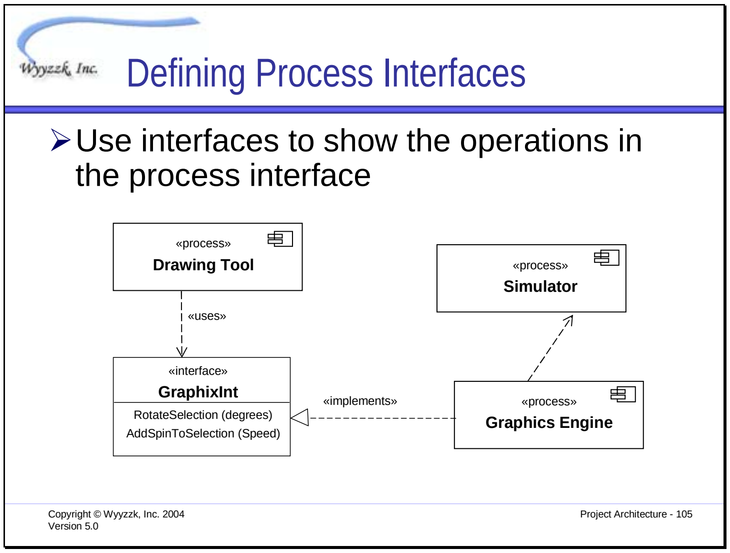
«process»
Simulator

«process»
Graphics Engine

Project Architecture - 104

If your subsystems have a relationship, then the processes that contain those subsystems also have a relationship.

Slide 105



Wyyzk, Inc.

Defining Process Interfaces

- Use interfaces to show the operations in the process interface

«process»
Drawing Tool

«uses»

«interface»
GraphixInt
RotateSelection (degrees)
AddSpinToSelection (Speed)

«implements»

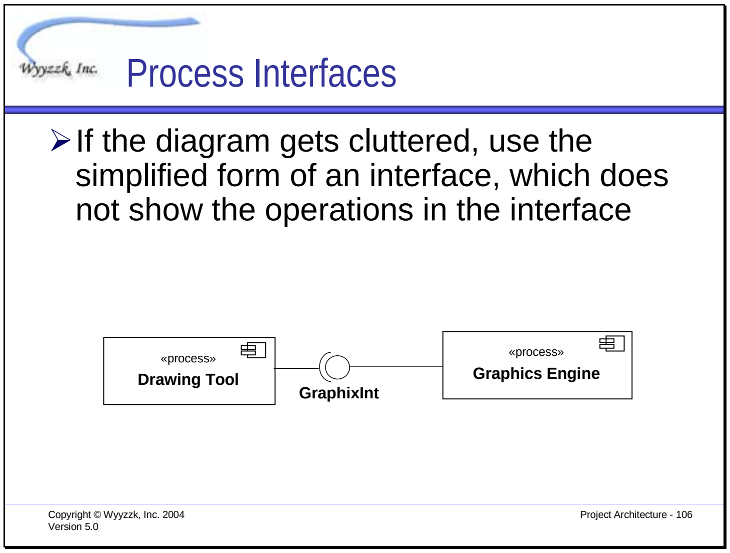
«process»
Simulator

«process»
Graphics Engine

Project Architecture - 105

The operations from a subsystem become the operations of the process that contains the subsystem.

Slide 106



Process Interfaces

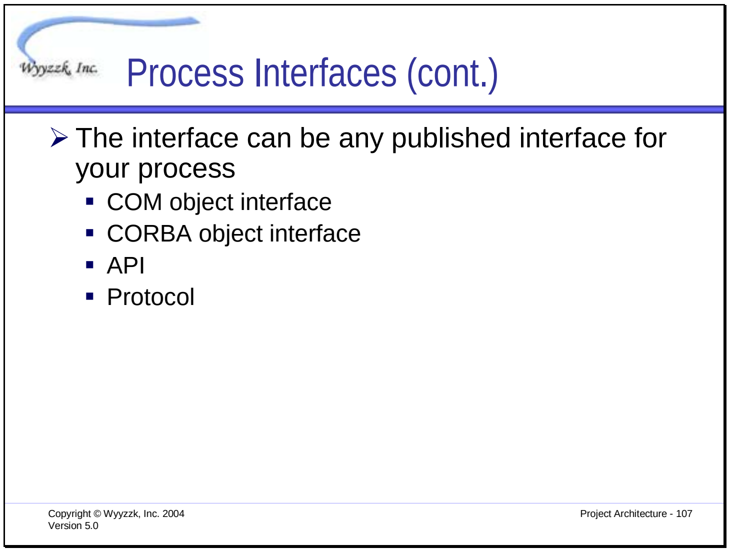
- If the diagram gets cluttered, use the simplified form of an interface, which does not show the operations in the interface

«process»
Drawing Tool

GraphixInt

«process»
Graphics Engine

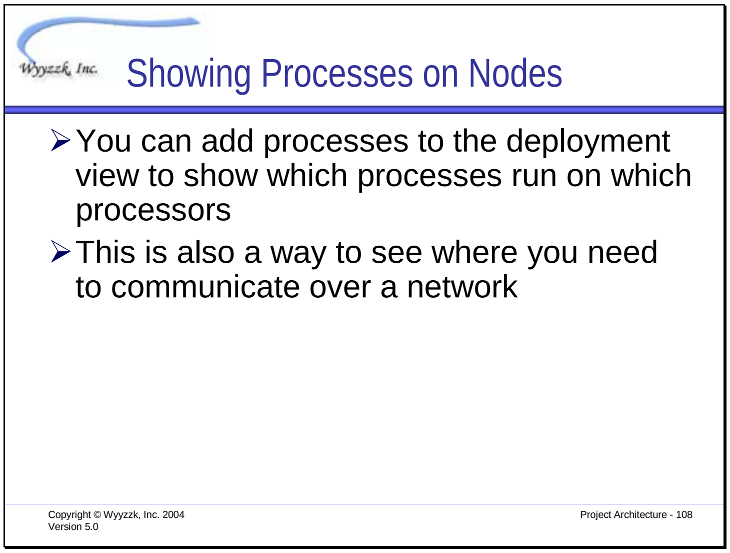
Slide 107



Process Interfaces (cont.)

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

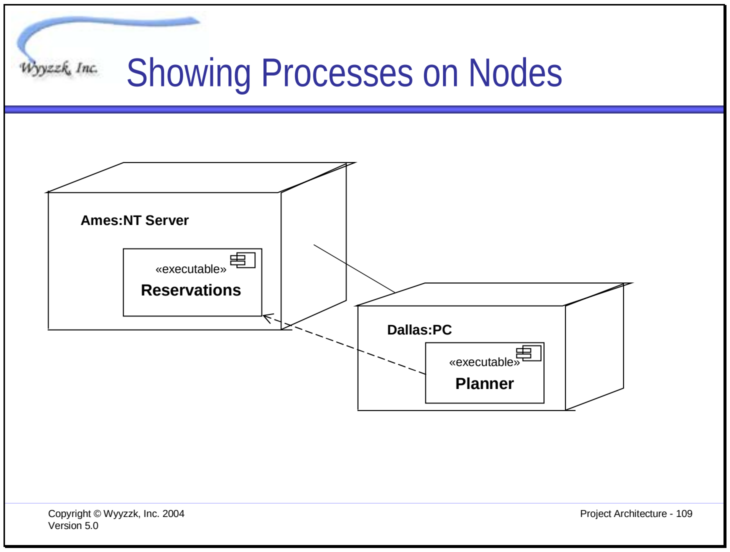
Slide 108



Showing Processes on Nodes

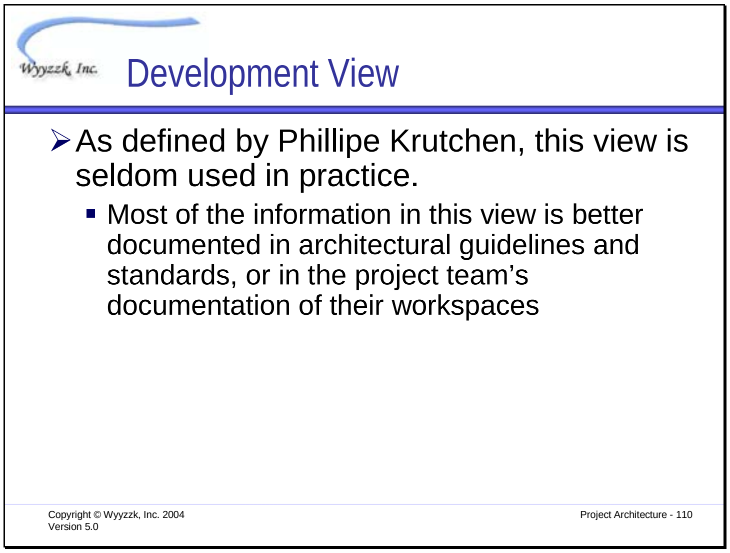
- You can add processes to the deployment view to show which processes run on which processors
- This is also a way to see where you need to communicate over a network

Slide 109



Showing Processes on Nodes

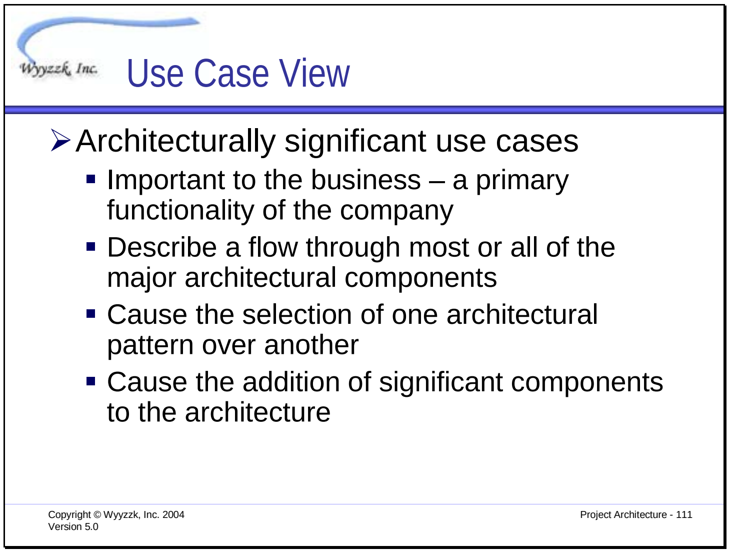
Slide 110



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

Slide 111



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

Use cases that are architecturally significant are described in the use case view. These are use cases that lead you to selecting one architecture over another.

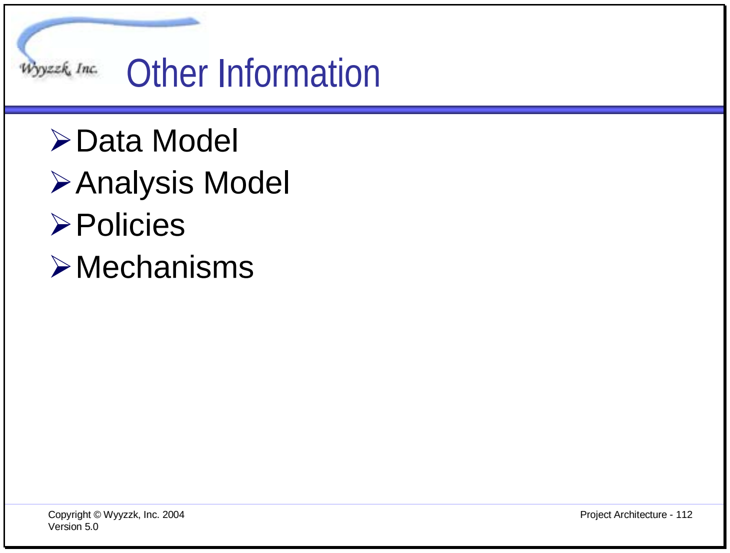
Important to the business – process orders for an online business

Describe a flow through most of the components – involve the use of user interfaces, business logic, and persistent data

Select an architectural pattern – use cases that describe batch processes would cause you to select a different architecture than use cases that describe human interactions with the system

Cause the addition of significant components – login, user tracking, logging can cause you to add a whole security component to your architecture.

Slide 112



Wyyzzk, Inc.

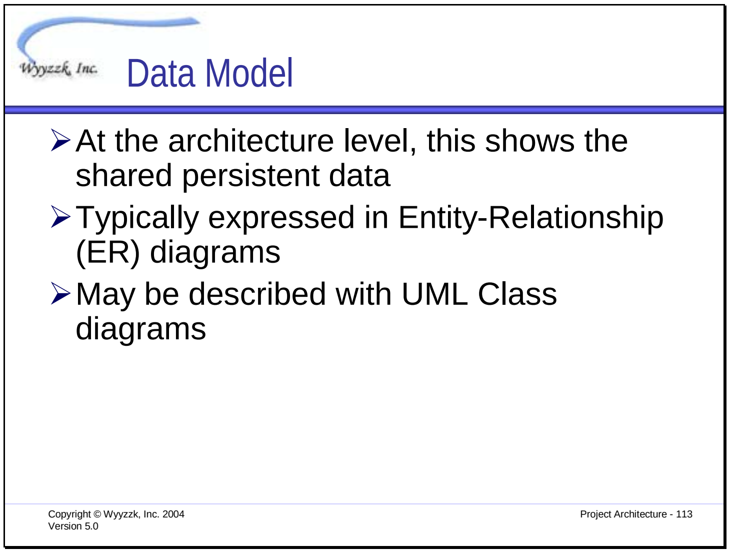
Other Information

- Data Model
- Analysis Model
- Policies
- Mechanisms

Copyright © Wyyzzk, Inc. 2004
Version 5.0

Project Architecture - 112

Slide 113



Wyyzzk, Inc.

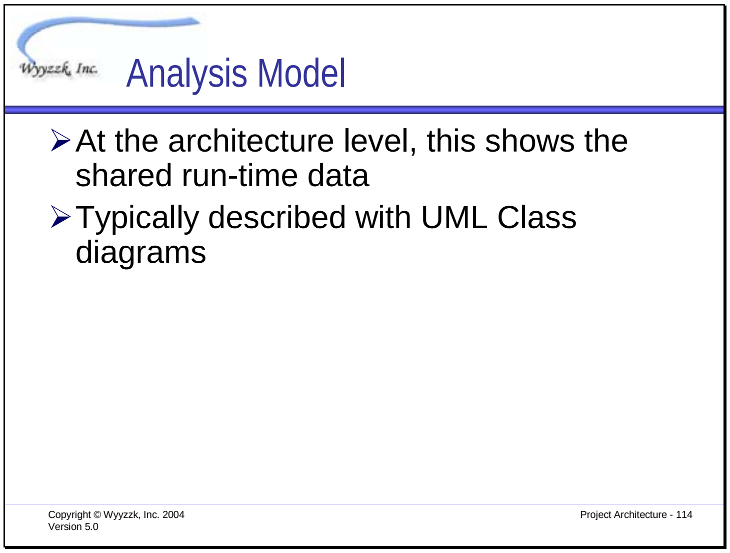
Data Model

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

Copyright © Wyyzzk, Inc. 2004
Version 5.0

Project Architecture - 113

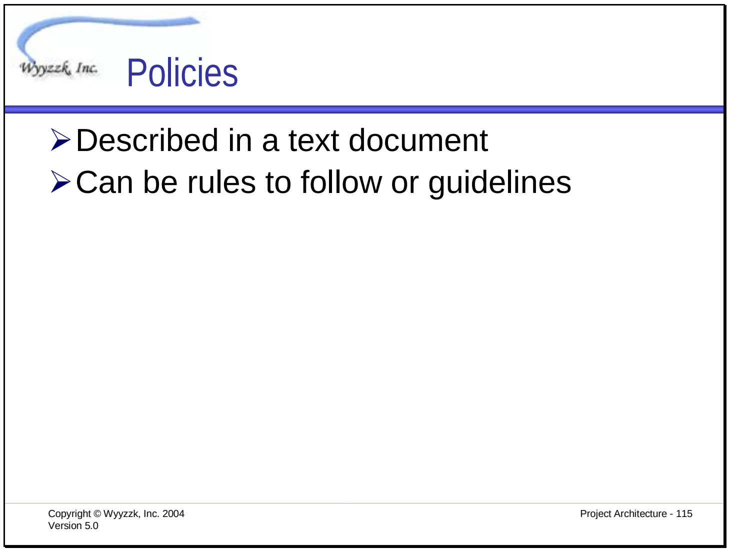
Slide 114



Analysis Model

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

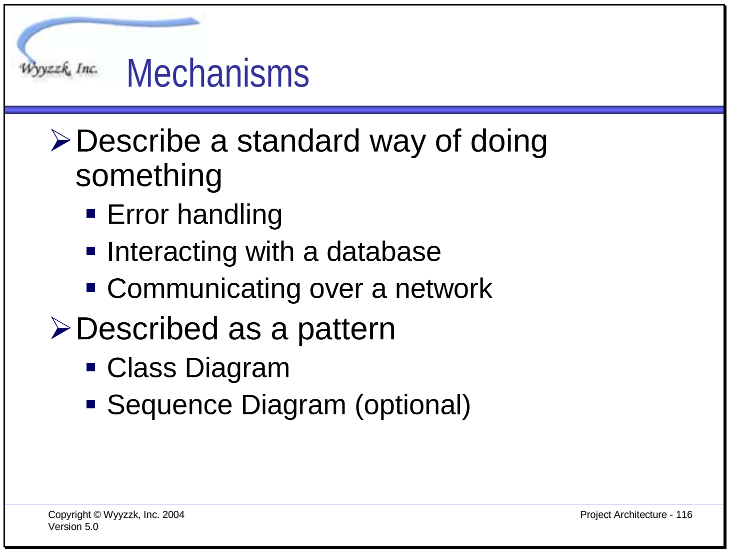
Slide 115



Policies

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

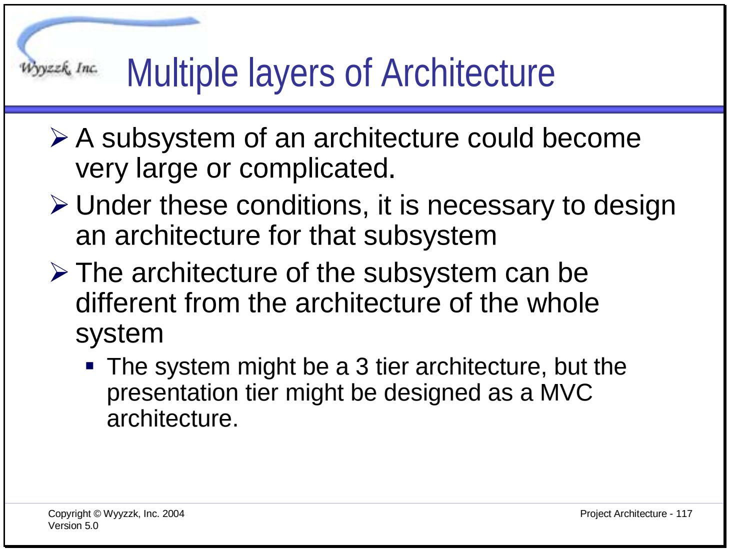
Slide 116



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)

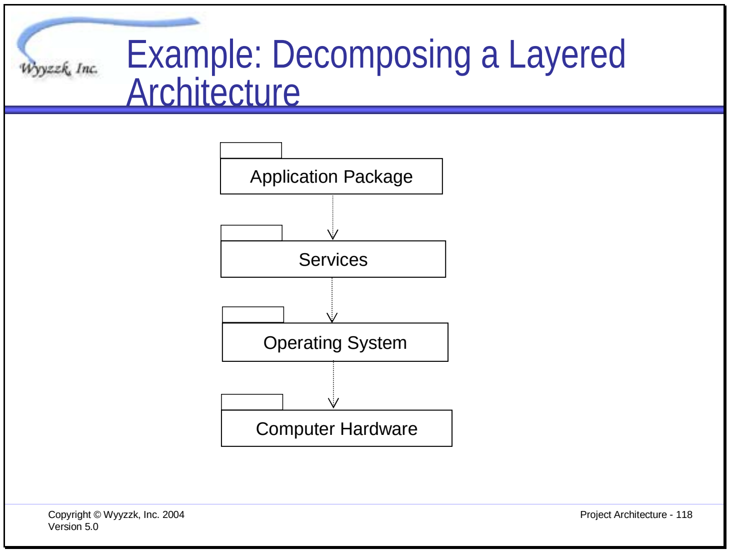
Slide 117



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.

Slide 118



Wyyzzk, Inc.

Example: Decomposing a Layered Architecture

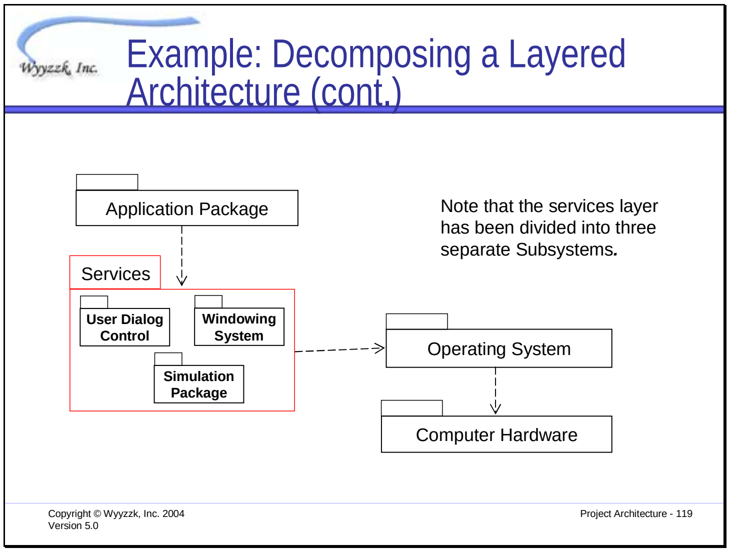
Application Package

Services

Operating System

Computer Hardware

Slide 119



Wyyzzk, Inc.

Example: Decomposing a Layered Architecture (cont.)

Application Package

Services

User Dialog Control

Windowing System

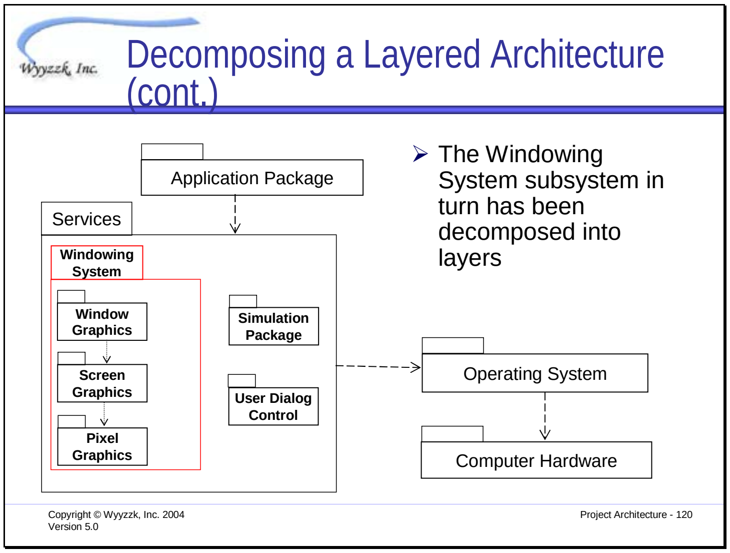
Simulation Package

Operating System

Computer Hardware

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

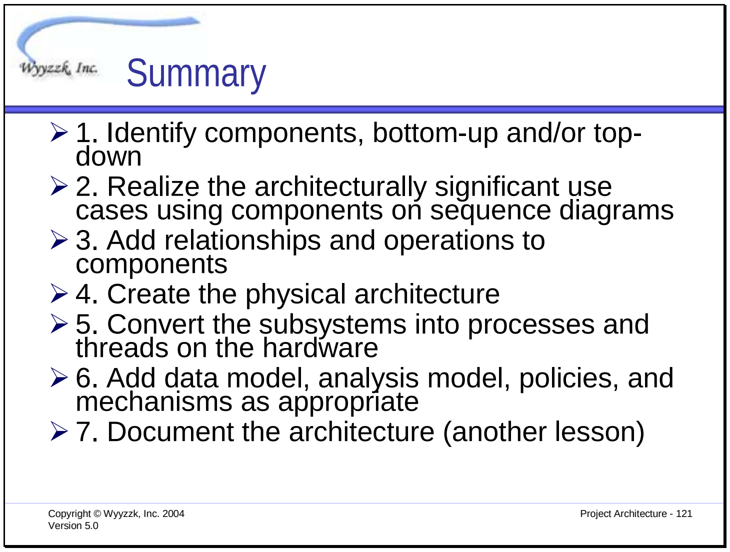
Slide 120



Decomposing a Layered Architecture (cont.)

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

Slide 121



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)