Software Performance Analysis

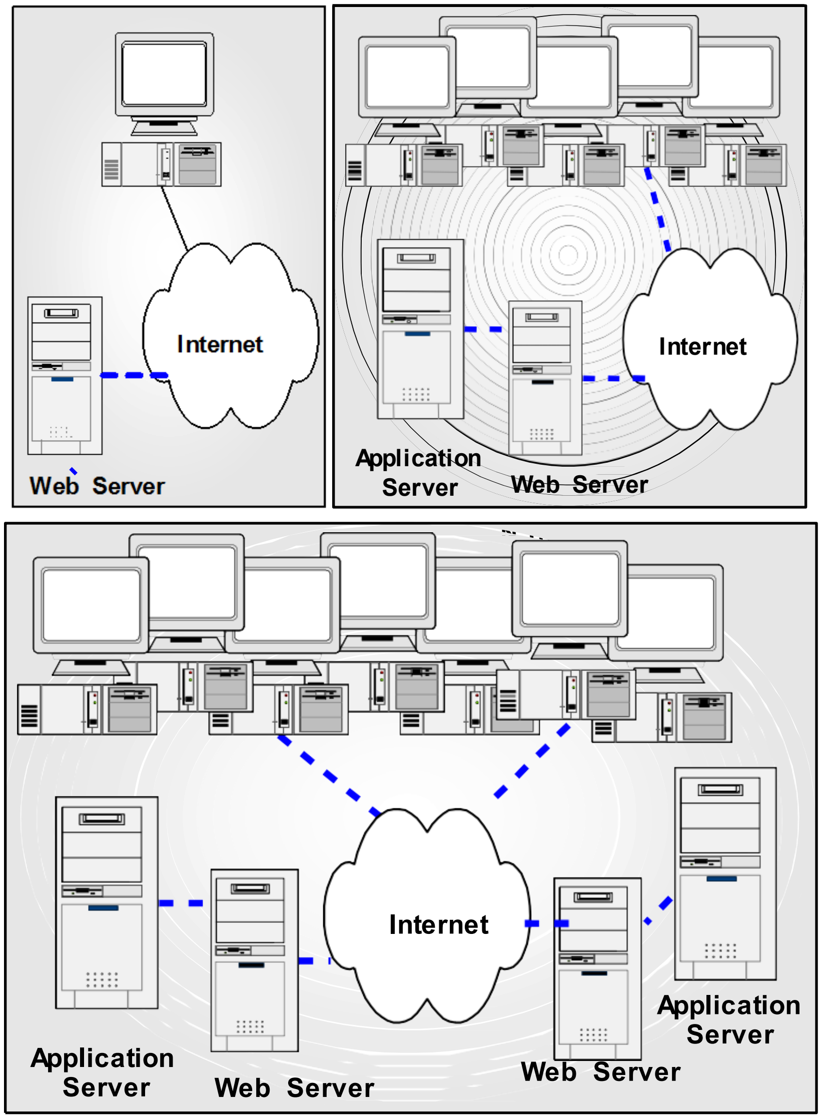
Ch 5:Web Applications and Other Distributed Systems

Outline

- Overview of Web applications, distributed object technologies, and the important considerations for SPE
- Techniques to represent and analyze interactions among multiple systems
- Extensions to software execution model for distributed systems
- Analysis of contention effects

Web Application Architecture

- Web applications adopt **client/server** architecture or **multi-tier** architecture
- The Web is a collection of Internet servers
  - **Web server** stores electronic files
  - **Application server** houses business logic



Observations

- Component based development technologies are largely adopted in the development of Web applications
- The development environments require little training of distributed object technologies or even in computer science
- Many Web applications are developed without the benefit of serious analysis, design or SPE
- The development process takes “**fix-it-later**” approach
- Both responsiveness and scalability are important for Web Applications

Class Diagram for a Web Application

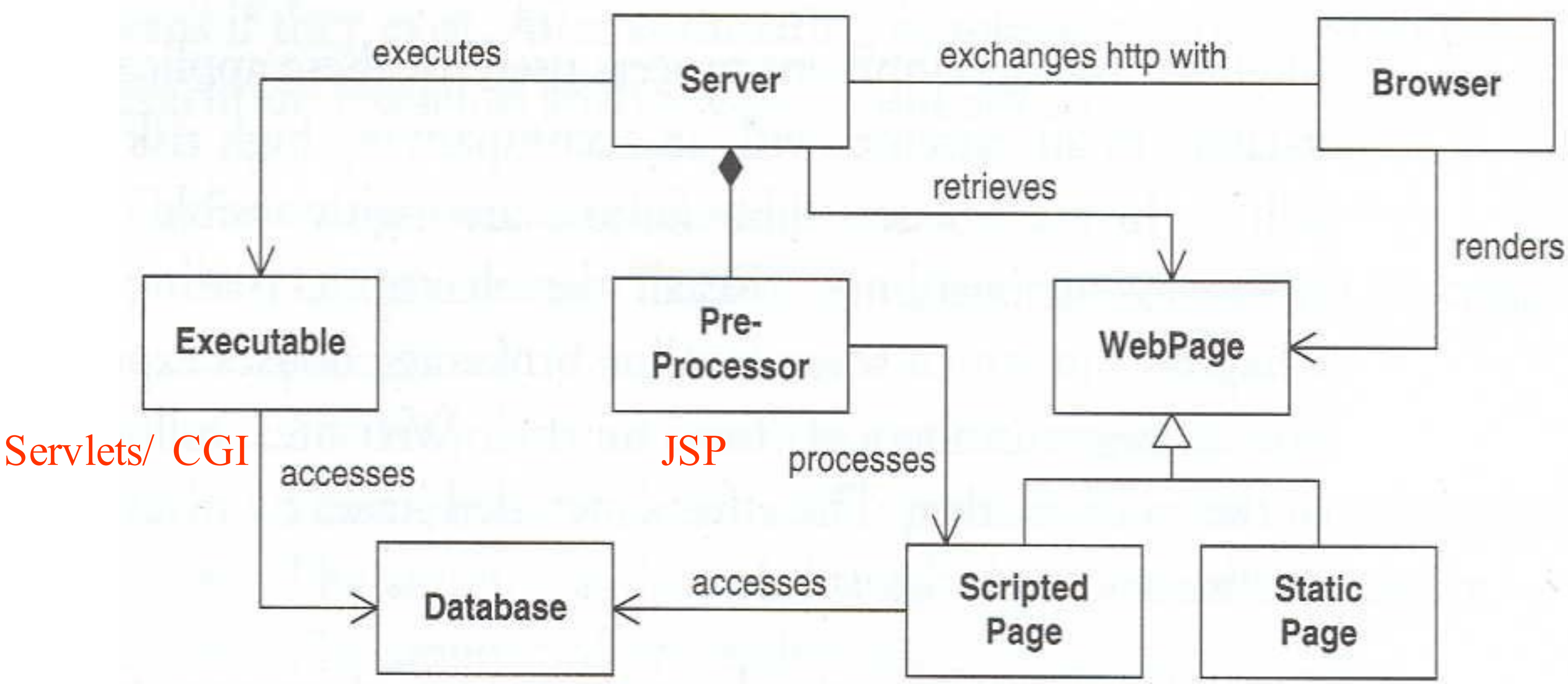
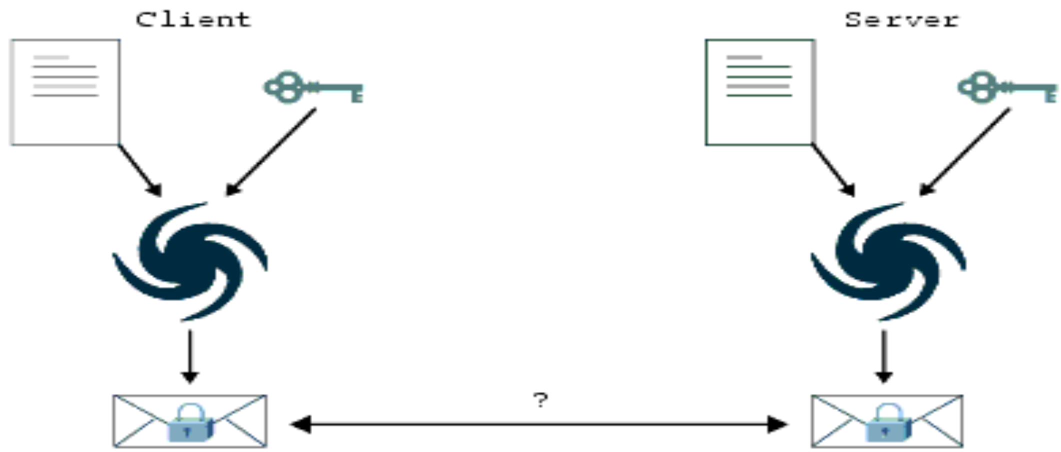


Figure 5-1: Generic Web Application

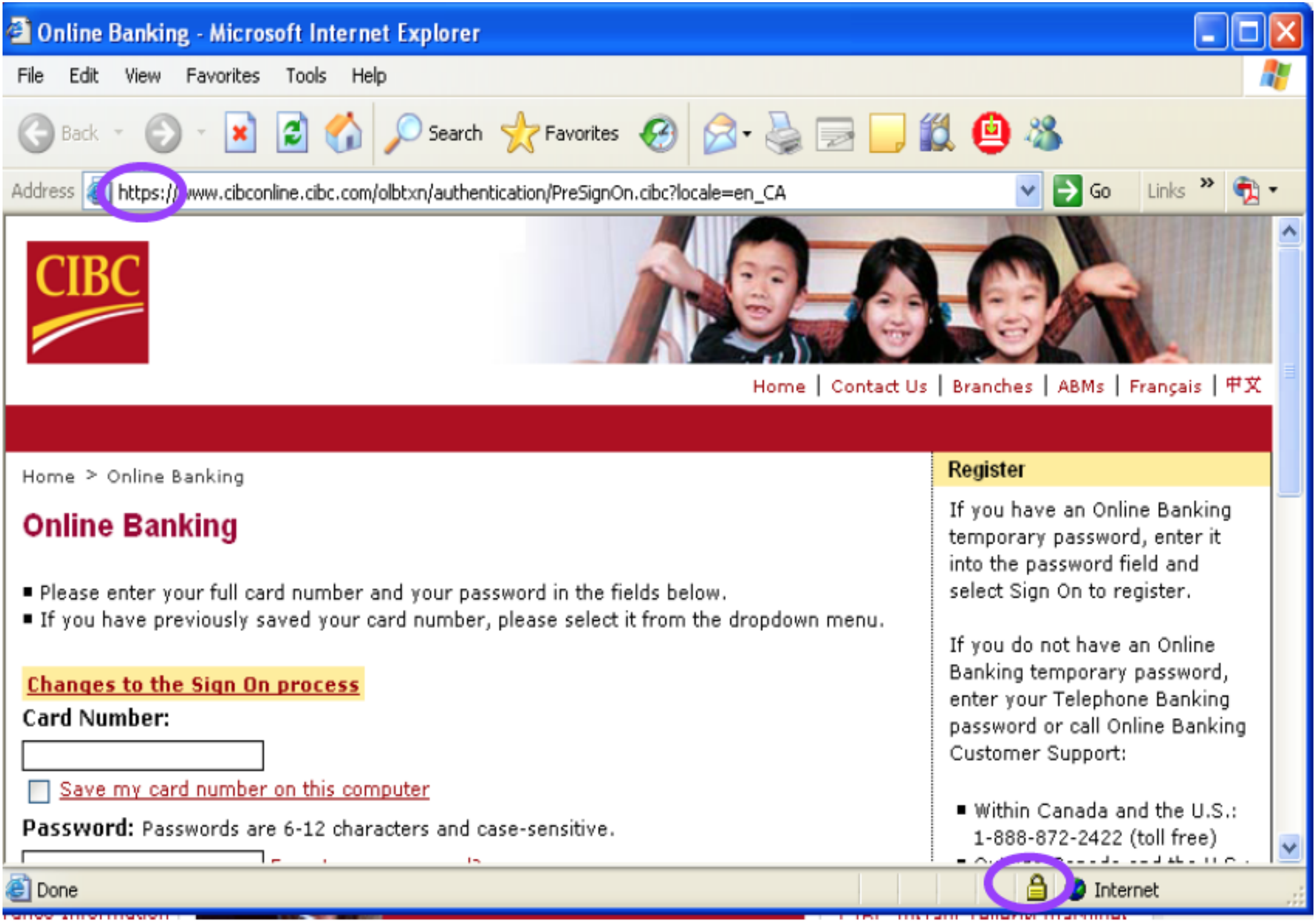
Performance Concerns

- Forecasting the volume of activity
- Choice of mechanism for executables
- Allocation of executables to processing nodes
- Mechanisms for accessing the database
- Mechanisms for handling persistent data
- Mechanisms for providing access and/or data security
- Instrumentation for characterizing Web application usage
- Location of the database
- Interfaces to legacy systems
- Impact of downloading applets or other code

Performance Problems with Good Security



- SSL (Secure Sockets Layer) is commonly adopted for secured data over the Internet
- Every time a browser makes an https request for a page
  - the server that the user is connecting to generates a digital key.
  - Generating this key is a computationally intensive operation.
- Some solutions:
  - Reuse of a key for the same session
  - Keep webpage short

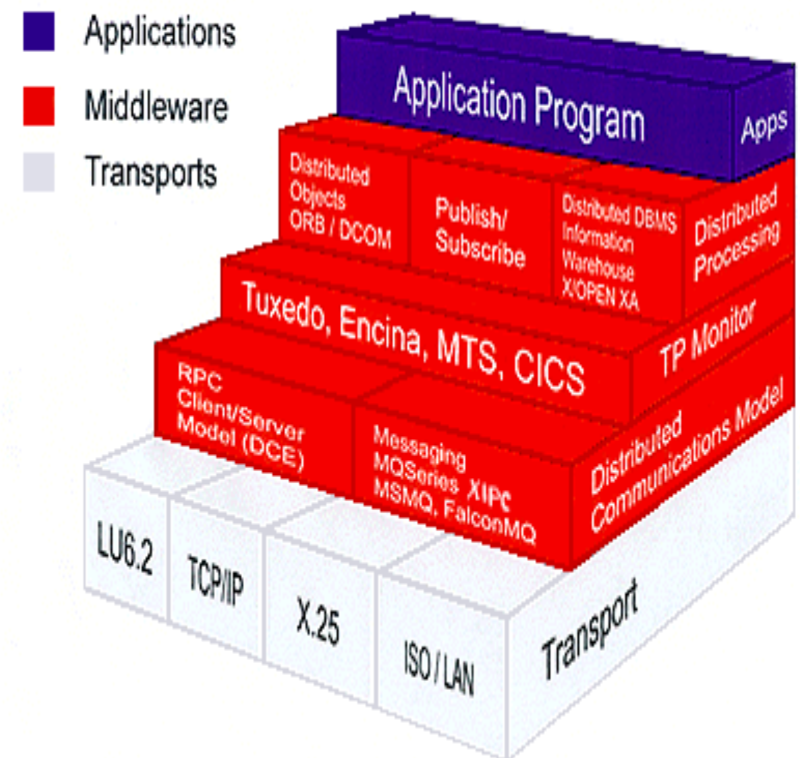

Online Banking - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address https://www.cibconline.cibc.com/olbtxn/authentication/PreSignOn.cibc?locale=en_CA
Go
Links
CIBC
Home | Contact Us | Branches | ABMs | Français | 中文
Home > Online Banking
Online Banking
Please enter your full card number and your password in the fields below.
If you have previously saved your card number, please select it from the dropdown menu.
Changes to the Sign On process
Card Number:
Save my card number on this computer
Password: Passwords are 6-12 characters and case-sensitive.
Register
If you have an Online Banking temporary password, enter it into the password field and select Sign On to register.
If you do not have an Online Banking temporary password, enter your Telephone Banking password or call Online Banking Customer Support:
Within Canada and the U.S.: 1-888-872-2422 (toll free)
Done
Internet

Distributed Object Technology (DOT)

- DOT is the merging of object oriented technology with distributed systems technology
- The object is the unit of computation and distribution in a heterogeneous multi-platform computing environment
  - Encapsulation of data and operations
  - Separate their interface from their implementation

Middleware

- A region in between client and server
- Provides an isolation layer, dealing with the different protocols and interfaces amassed from heterogeneous environment
- Presents its own API



Major Middleware Technologies

- Common Object Request Architecture (CORBA), from OMG
- Distributed Common Object Model (DCOM), from Microsoft
- J2EE from Sun →now Oracle
- Common Object Model Plus (COM+), from Microsoft

Performance Aspects Related to DOT

Latency

- Latency is the difference in response time between a local and a remote operation invocation
- Sources of latency include
  – network speed
  – middleware overhead
  – communication overhead due to objects in difference address space
- Performance in a distributed system is governed by communication overhead rather than component implementation

Memory Access

- Objects in the same memory space can be accessed efficiently using pointers
- Object in different address spaces can be accessed using less efficient object references
- The differences in the way local versus remote objects are accessed requires that either
  - The programmer must be aware of the ultimate location of the object, or
  - The execution environment must provide a uniform mechanism for accessing objects that hides their location

Partial Failure

- Failure occurs in a component, network, a given processor while others continue to operate and communicate normally
- It is **difficult** to restore consistency following a failure
- Two alternatives for coping with failures
  – Treat all objects as if they were local
    - Leads to nondeterministic behavior in the case of partial failure
  – Treat all objects as if they were remote
    - Adds additional latency for accessing local objects

Concurrency

- Methods of same objects could be invoked concurrently
- To prevent inconsistencies or data corruption, distributed objects must define and maintain critical sections to manage concurrent access to their data
- Approaches to treating all objects uniformly
  - Ignore the concurrency
  - Make all objects single thread
  - Include concurrency semantics in all objects, regardless of their location

Effective Development with DOT

- Object locations and concurrency semantics should be fixed early in the development process
  - For example, if an object will be remote, minimize the number of calls to the object, and maximizes the value of the results obtained
- From a performance perspective, the only rational way to make trade-offs is to base decisions on quantitative information

Modeling Distributed System Interactions

- The essence of the distributed system software architecture is ***the placement of objects*** and the ***communication*** and ***synchronization among objects***
- Communication and synchronization overhead have significant impact on performance
- We adapt "keep-it-simple" approach
  - Represent synchronization as a ***delay*** for a client to receive results from a server process in the stage of modeling software execution models
  - Add specific synchronization notation to the sequence diagrams and software execution models, later

Modeling Distributed System Interactions (con't)

– Use ***software model approximation techniques*** to solve synchronization extended software execution models

The Essence of Distributed Systems

Object Placement

Communication

Synchronization

Types of System Interactions

- Four types of system interactions are typically supported in middleware
  - Synchronous
  - Asynchronous
  - Deferred synchronous, and
  - Asynchronous callback communication
- Severs provide services to clients
- Clients request services on servers
- The roles of servers and clients refer to a particular interaction, and may be reversed in subsequent interactions

Synchronous Communication

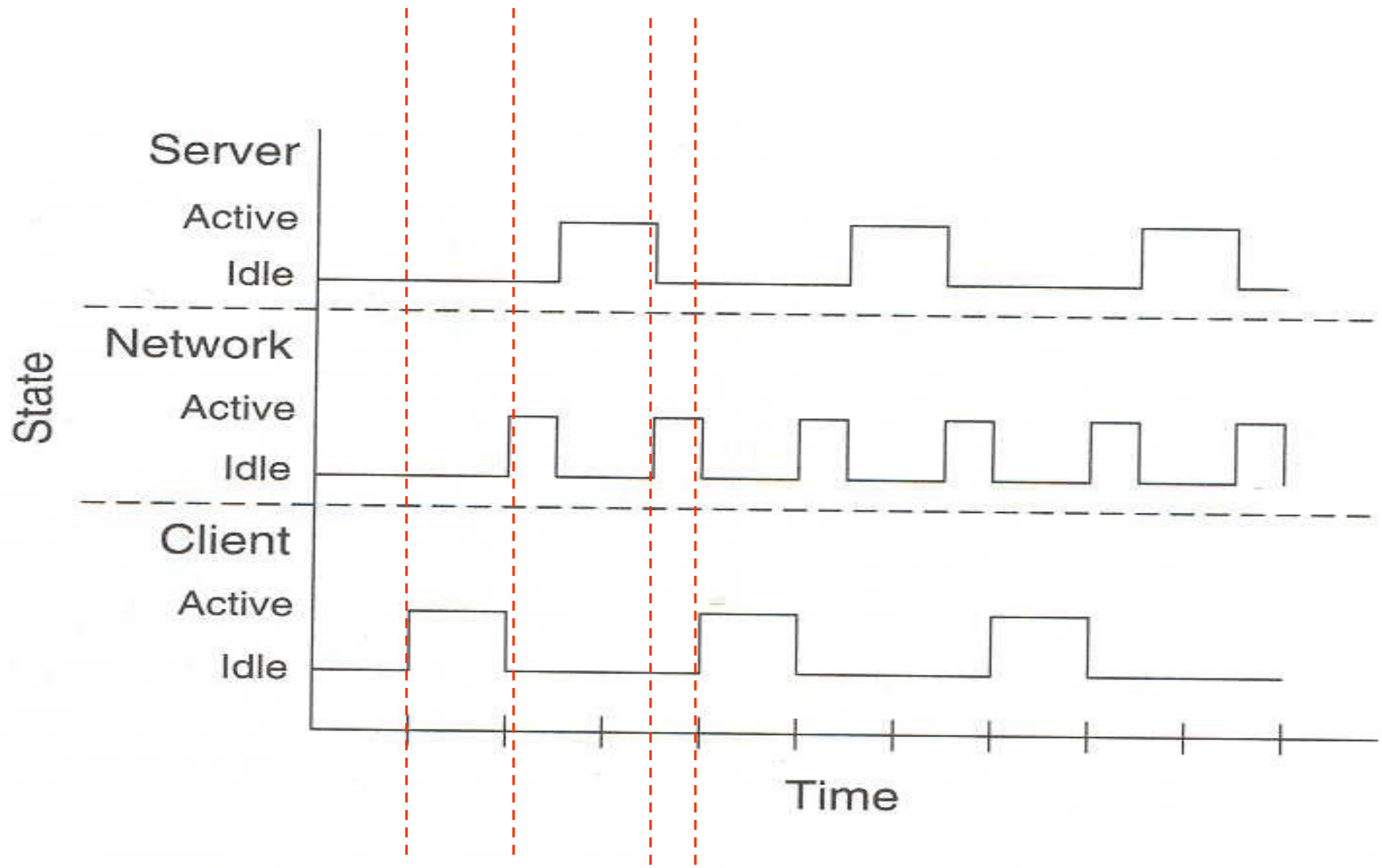


Figure 5-2: Synchronous Communication

Asynchronous Communication

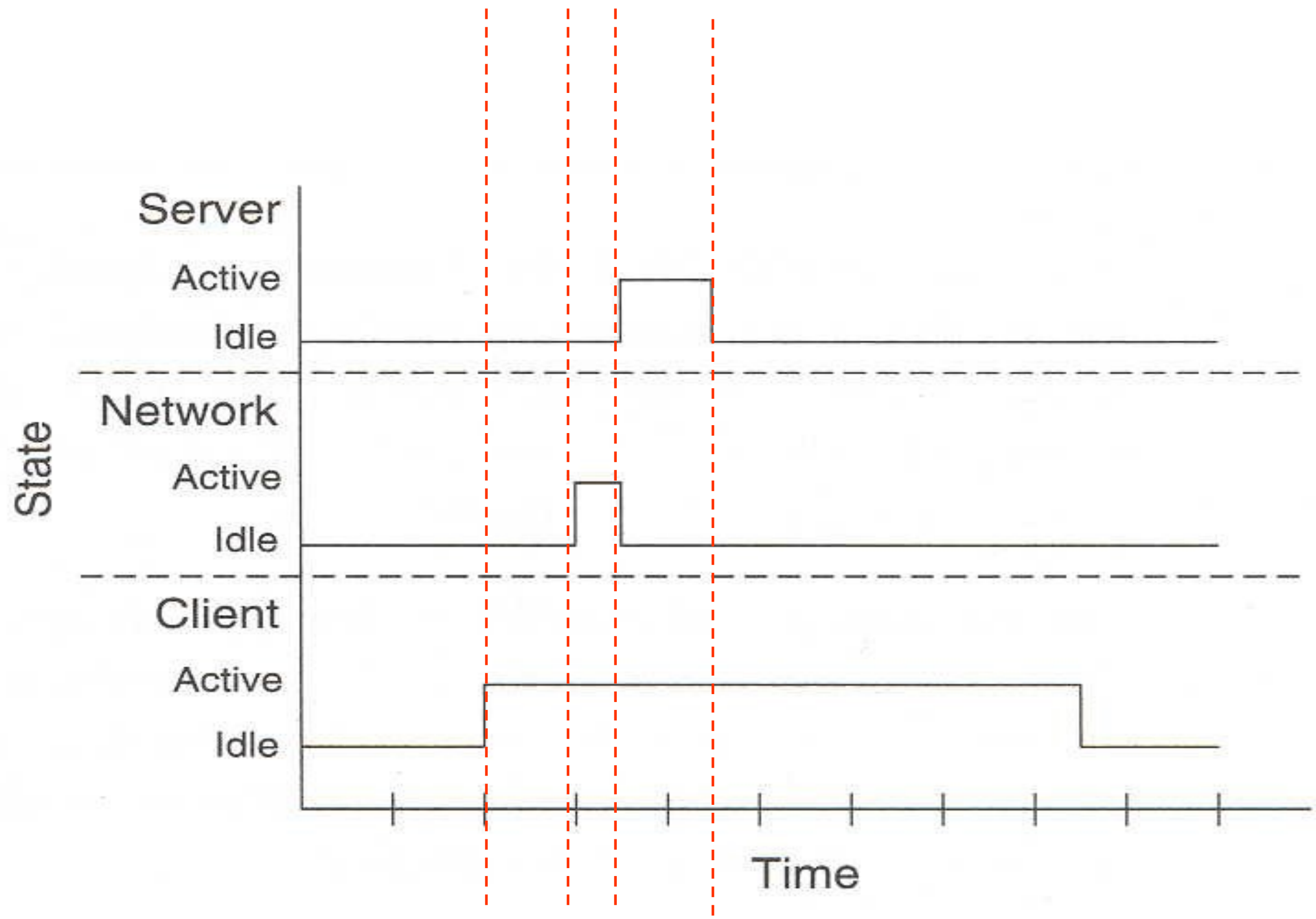


Figure 5-3: Asynchronous Communication

Deferred Synchronous Communication

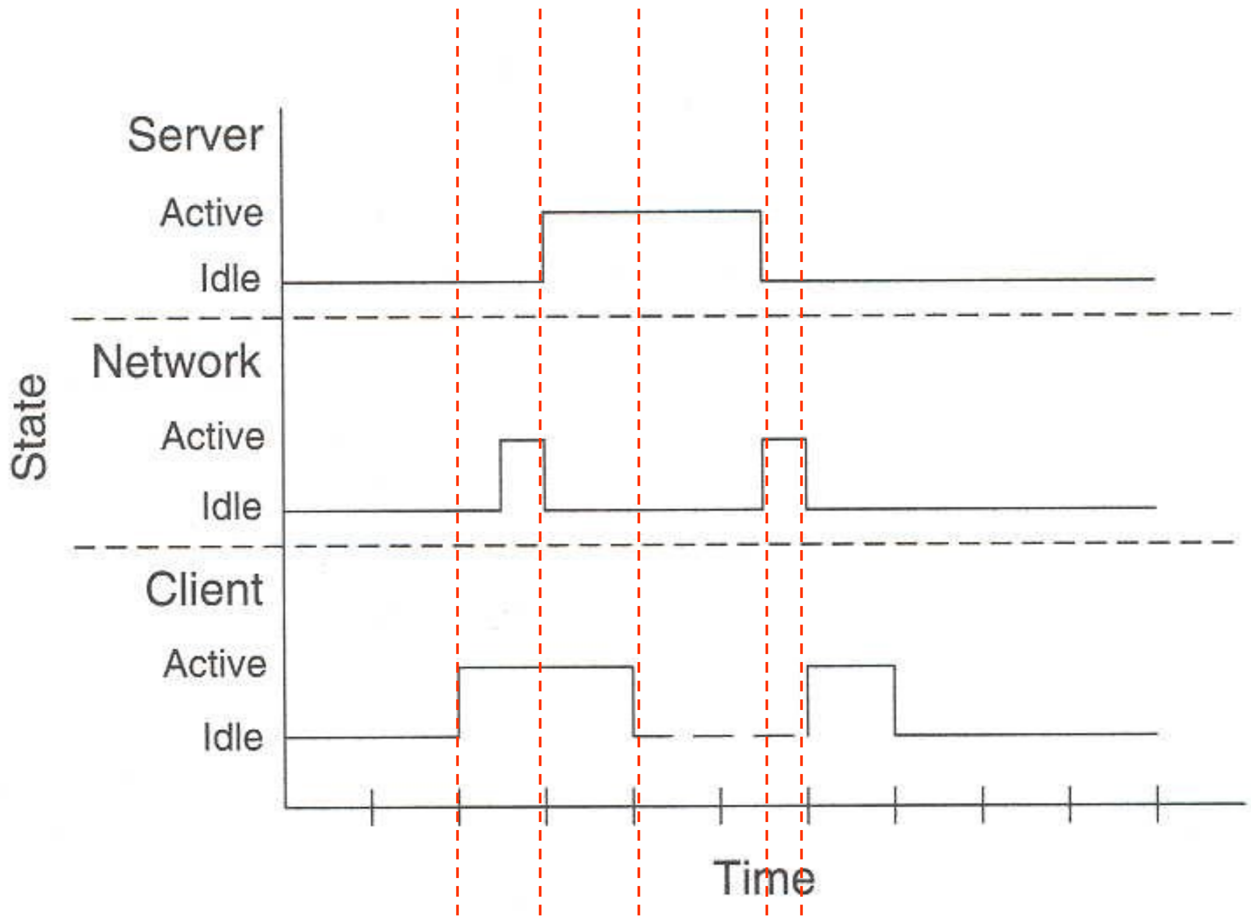


Figure 5-4: Deferred Synchronous Communication

Asynchronous Callbacks

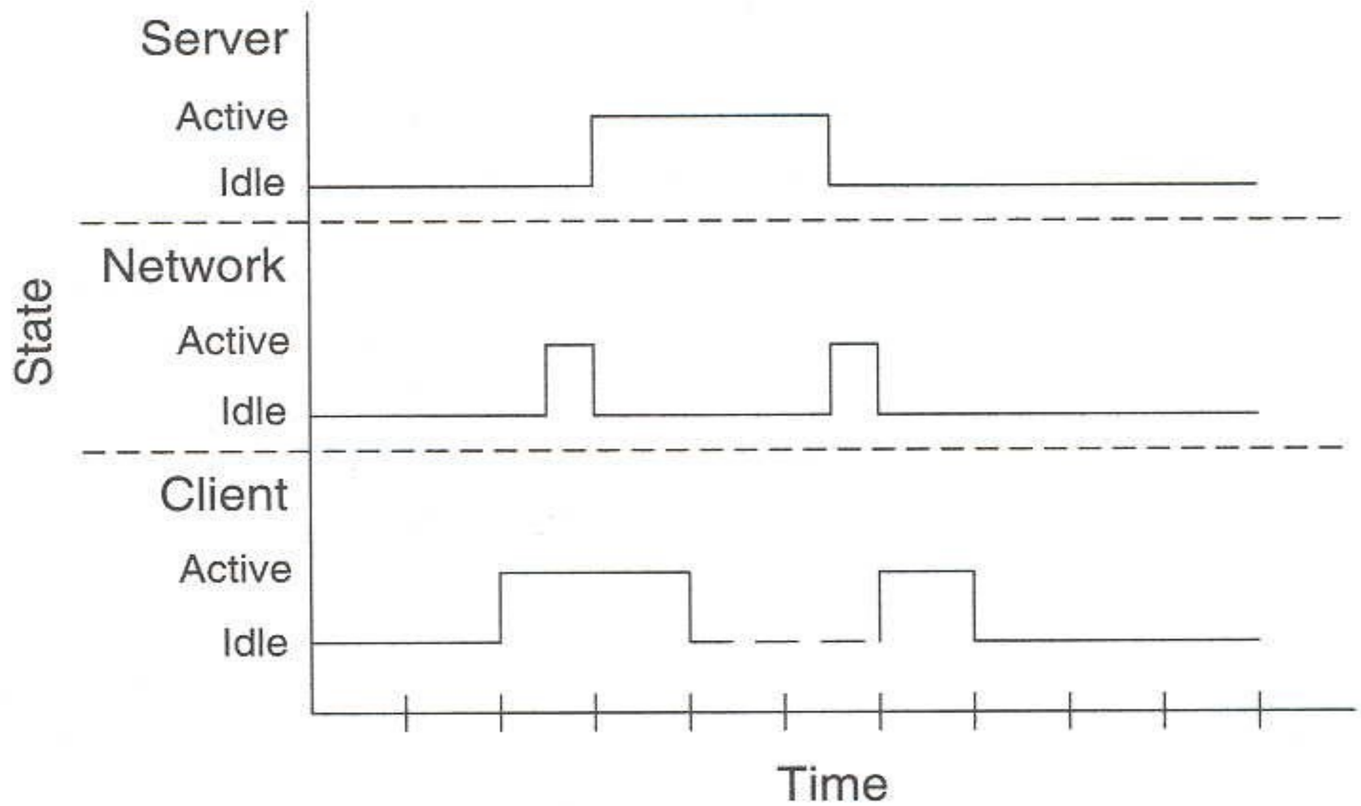


Figure 5-4: Deferred Synchronous Communication

RECALL

Synchronization

- UML provides different types of arrowheads to represent communications among objects

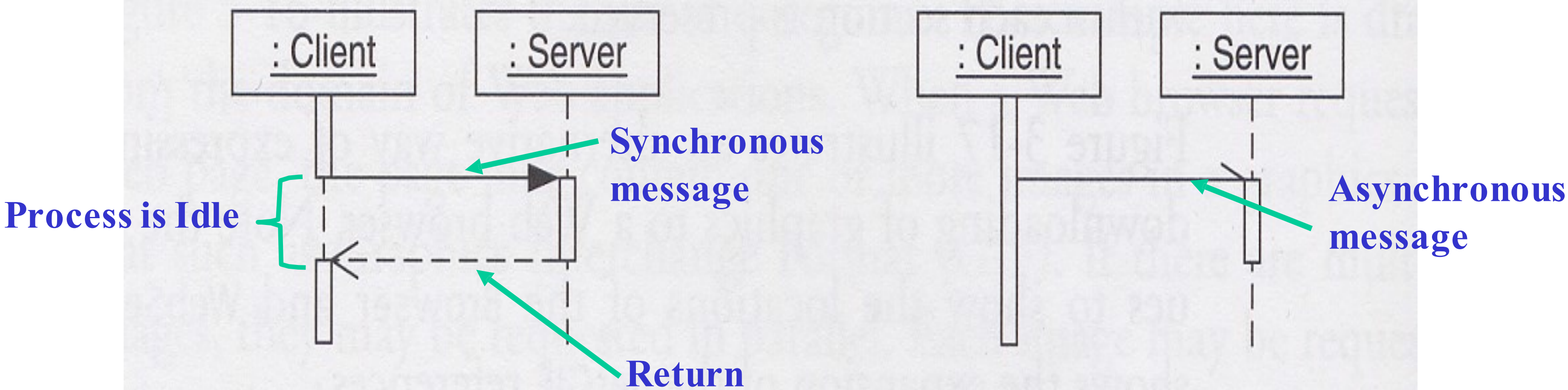


Figure 3-19: Synchronous Communication

Figure 3-20: Asynchronous Communication

RECALL

Synchronization (con’t)

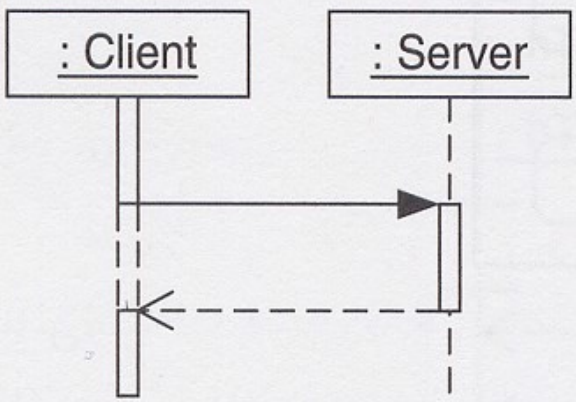


Figure 3-21: Deferred Synchronous Communication

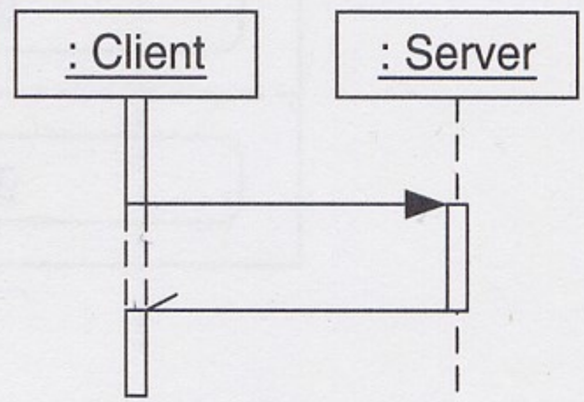


Figure 3-22: Asynchronous Callback

Software Execution Model Representation

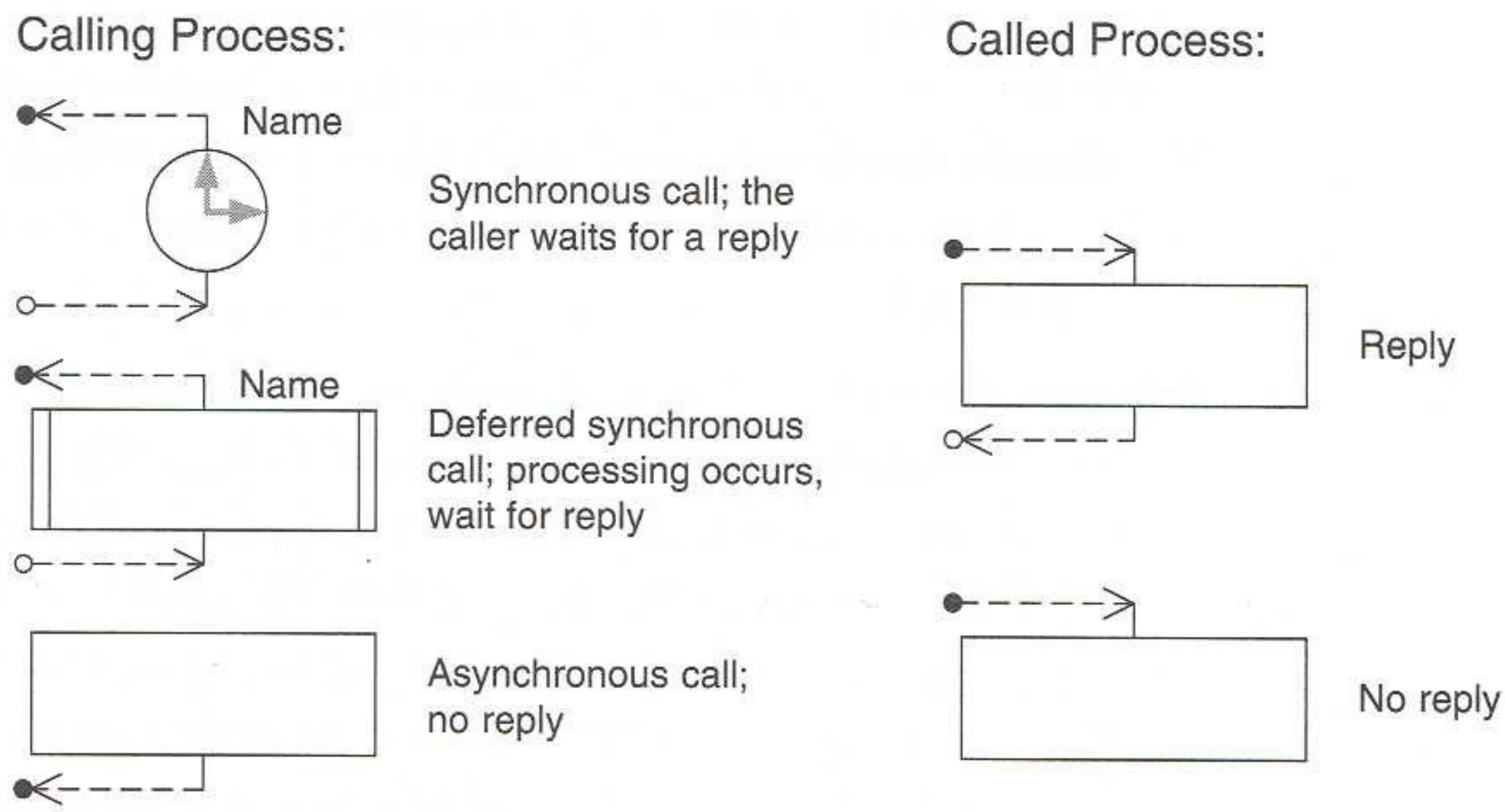


Figure 5-5: Execution Graph Nodes for Software Synchronization

Software Model Solution Approximations

- Use approximate solutions of software models to easily identify serious problems and to permit quick analysis of many architectural and design alternatives
- Begin by creating separate performance scenarios for each key facility
- Specify resource requirements for active regions and estimate delays between active regions
- Later, if necessary, model individual processes on each facility

Solving Software Execution Models for Distributed Systems

- Insert synchronization nodes at appropriate points in the processing steps
- To solve the models, you specify
  - Resource requirements for the processing nodes
  - The estimated delay for synchronization nodes
  - The number of messages sent via the network
  - The processing overhead for middleware that handles remote calls

Example: Web e-Commerce Application

- We consider a simple e-commerce application in which users may purchase items via the Web
- With the browser, the user ***selects items to purchase***
- Once all selections have been made, the user "***checks out***", and the order is processed
- When order processing is complete, ***the user receives an acknowledgement***

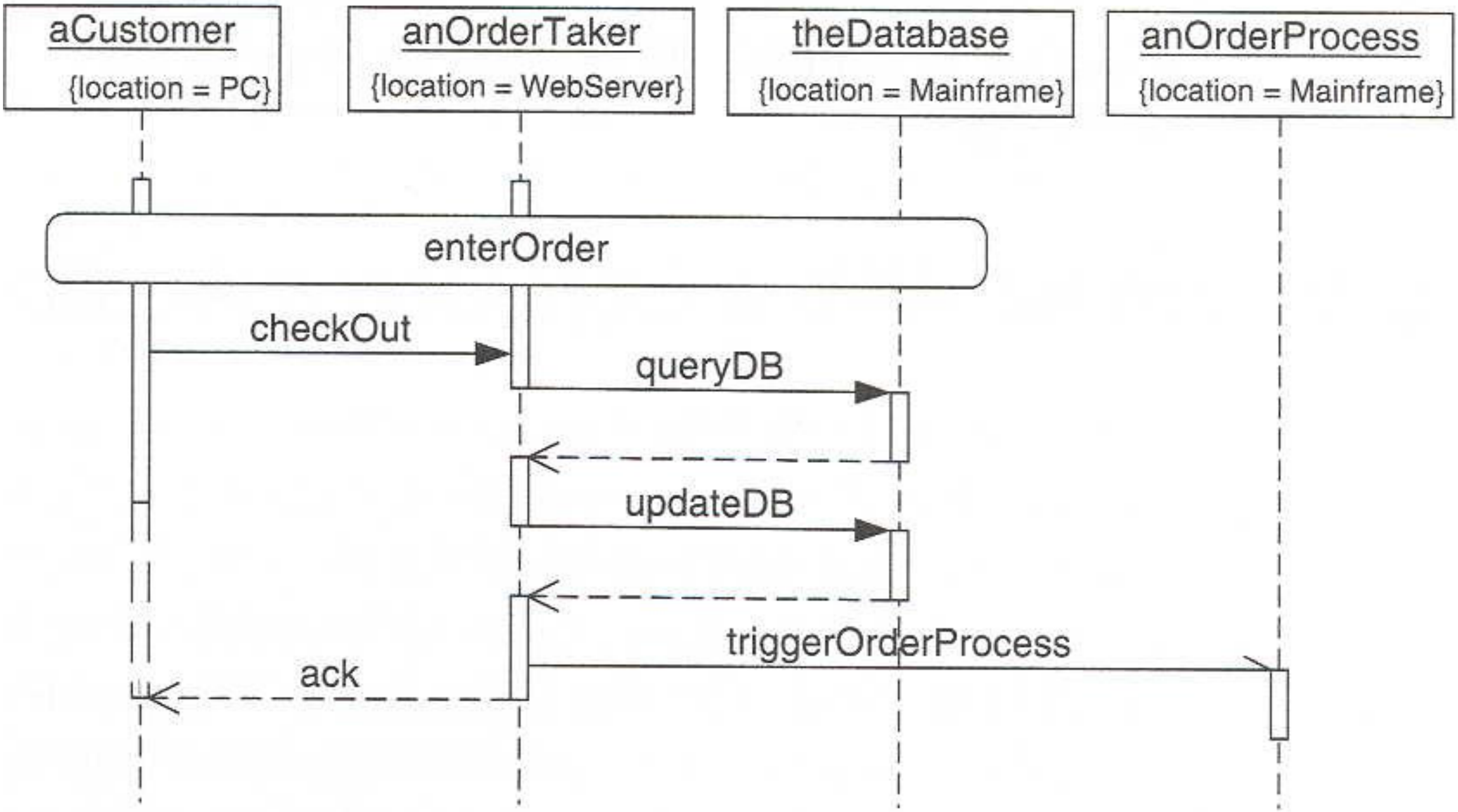


Figure 5-6: checkOut Scenario

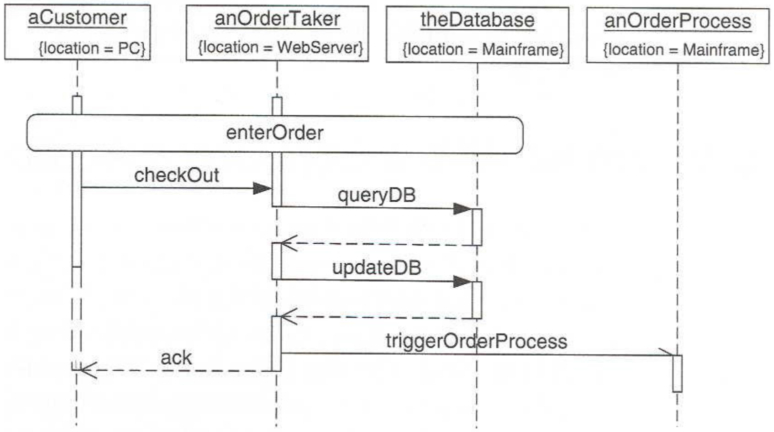


Figure 5-6: checkOut Scenario

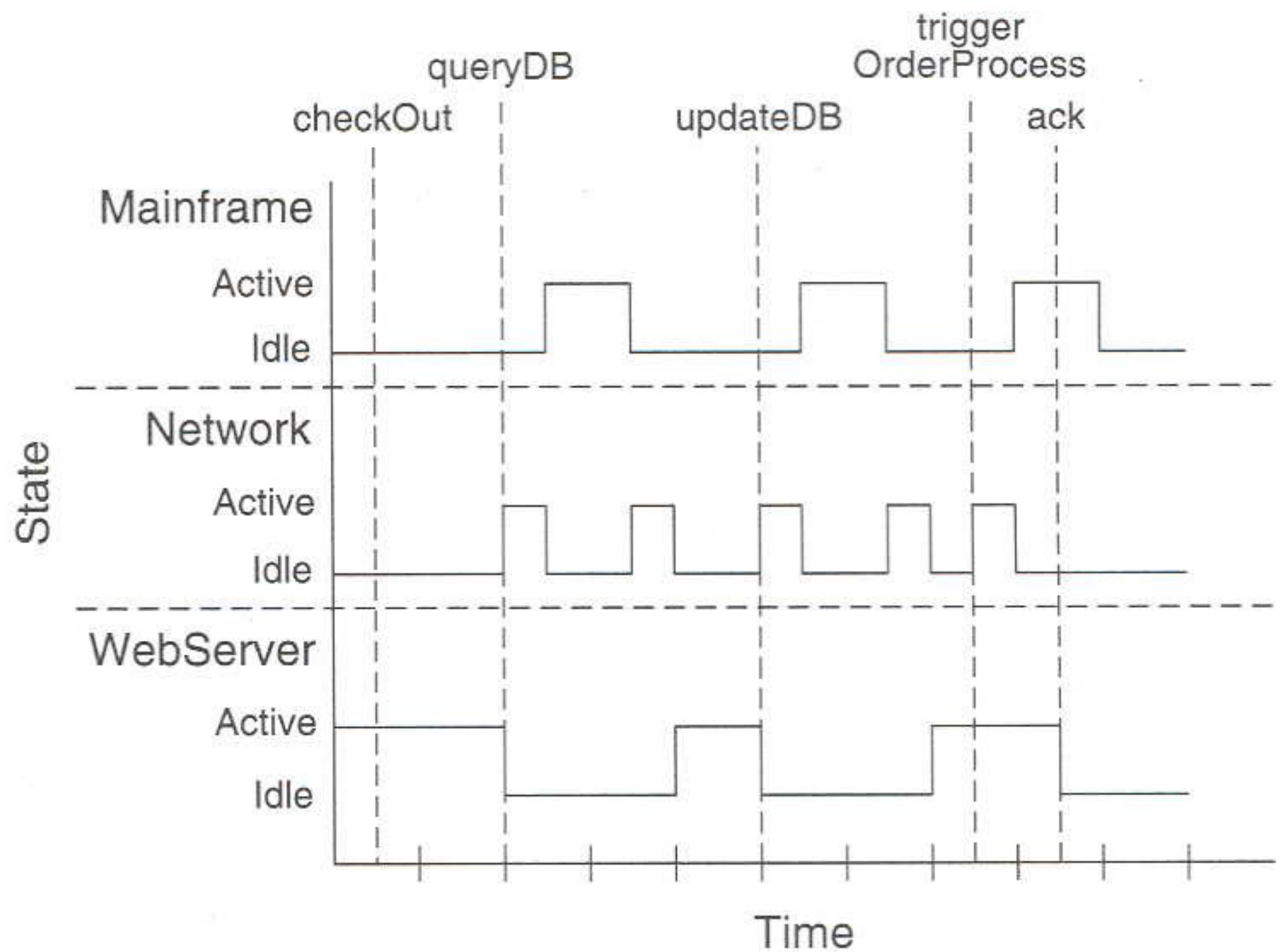


Figure 5-7: Timing Diagram for newOrder Scenario

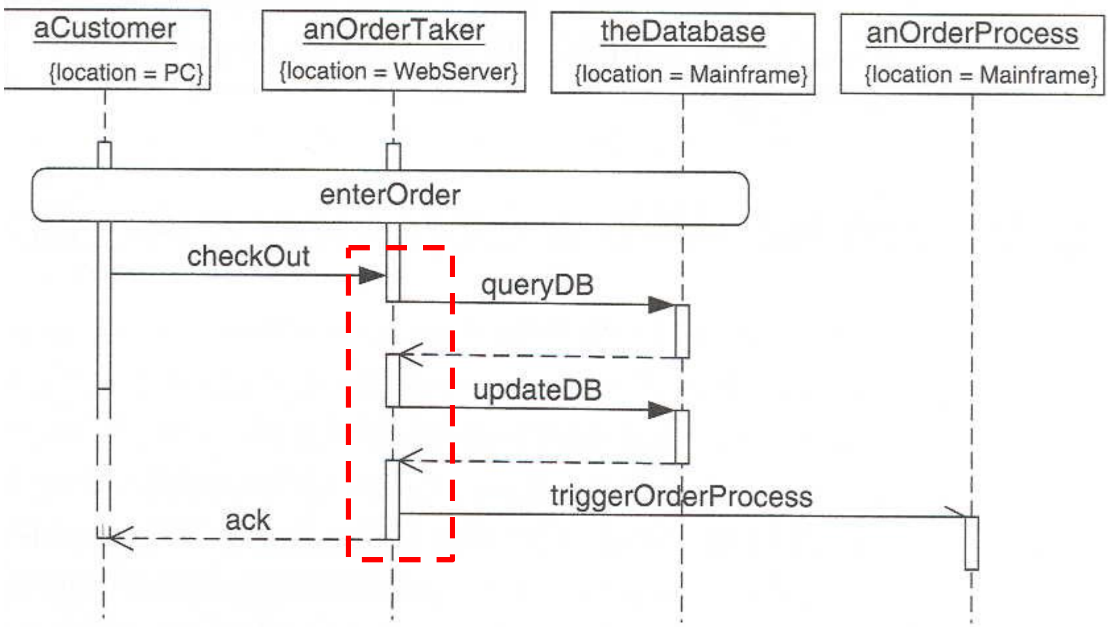


Figure 5-6: checkOut Scenario

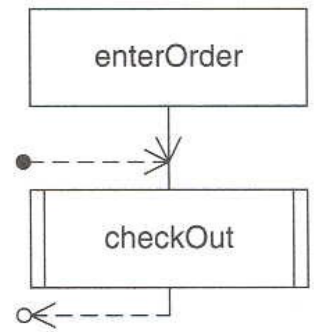


Figure 5-8: checkOut Execution Graph

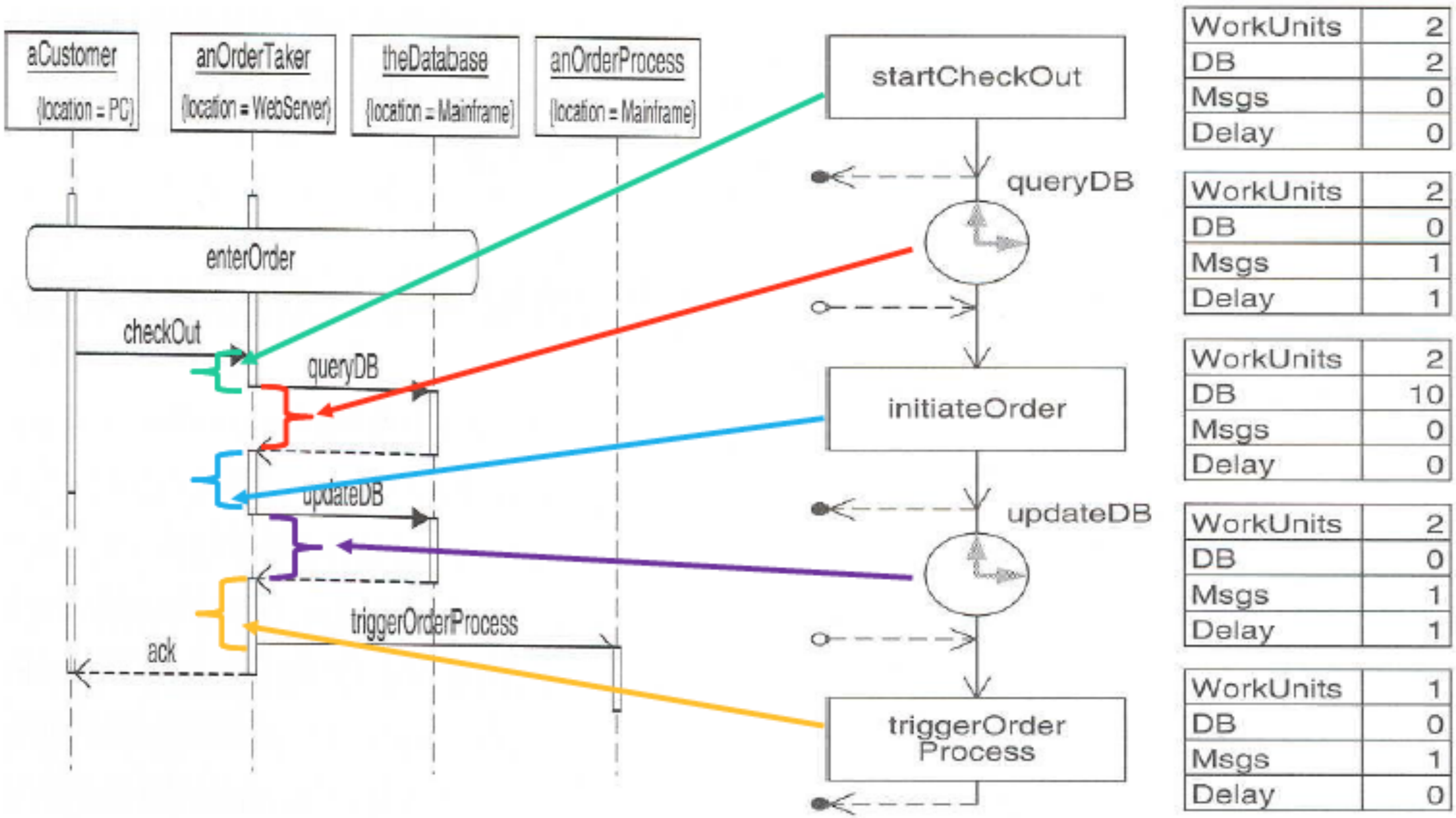


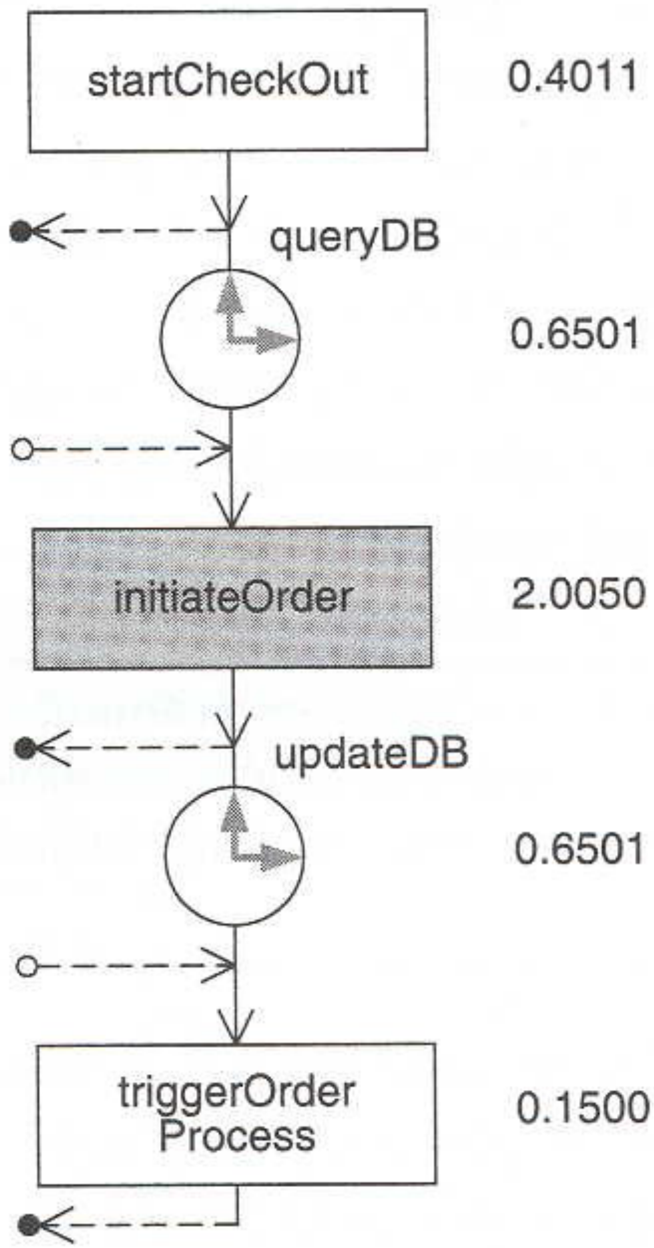
Figure 5-6: checkOut Scenario

Figure 5-9: Expansion of checkOut

Table 5-1: Web Server Computer Resource Requirements

Devices	CPU	Disk	Delay	GINet
Quantity	1	1	1	1
Service Units	K Instr.	I/Os	Visits	Msgs.
WorkUnits	25			
DB	500	4		
Msgs	25	1		1
Delay			1	
Service Time (sec.)	.000001	0.05	0.5	0.1

Time, no contention: 3.8563



Total Resource Usage

CPU	0.0063
Disk	2.5500
Delay	1.0000
GINet	0.3000

Figure 5-10: checkOut Software Model Results

Database Scenario

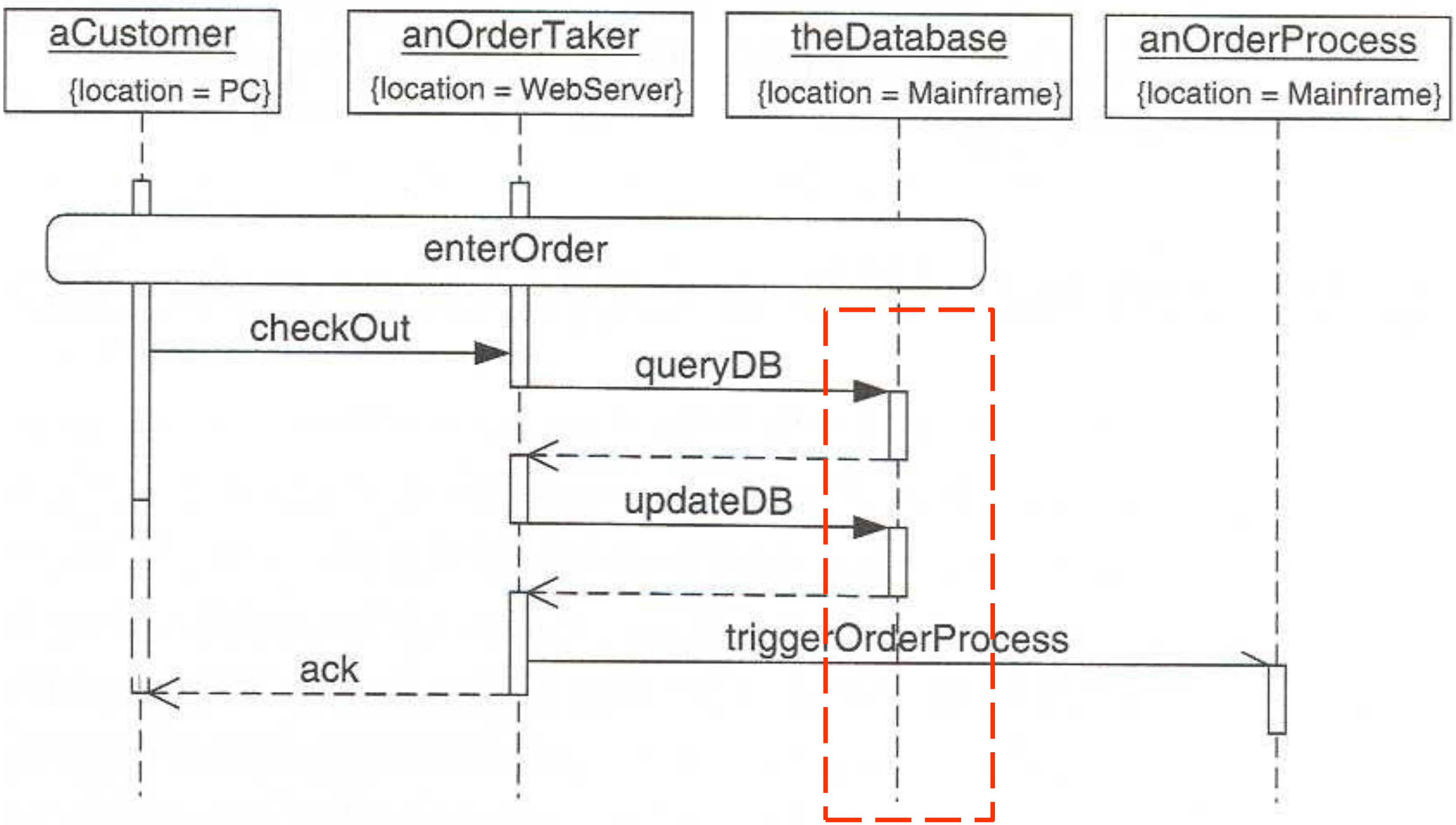


Figure 5-6: checkOut Scenario

Database Scenario

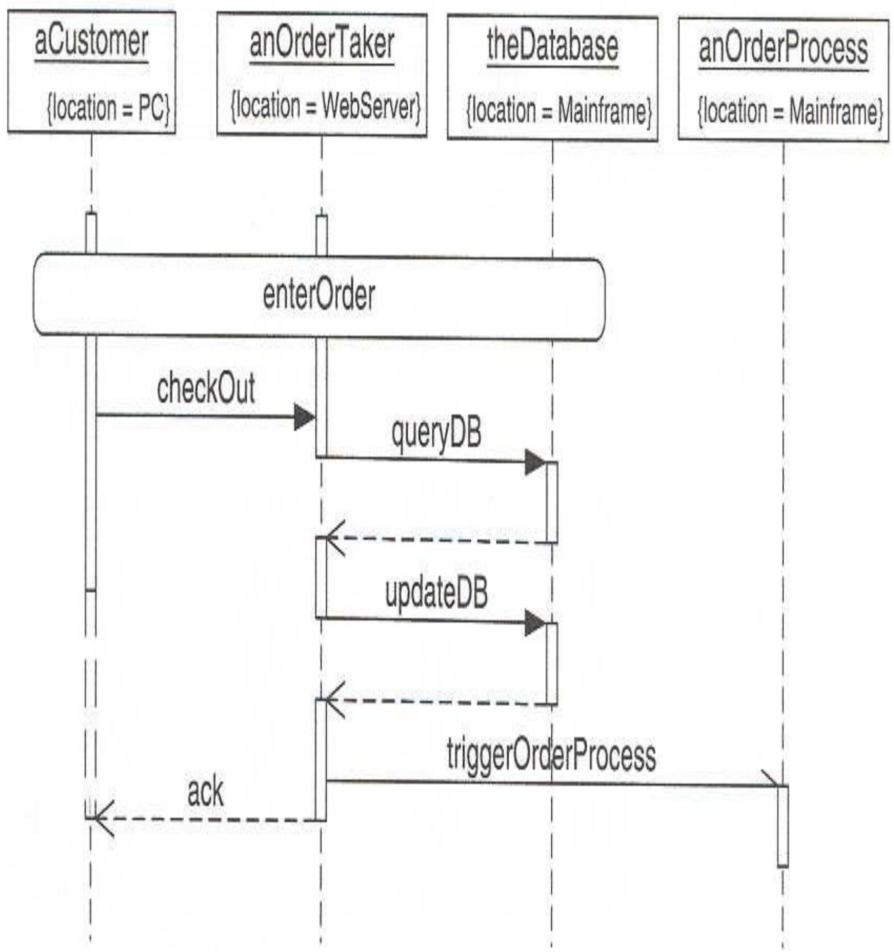


Figure 5-6: checkOut Scenario

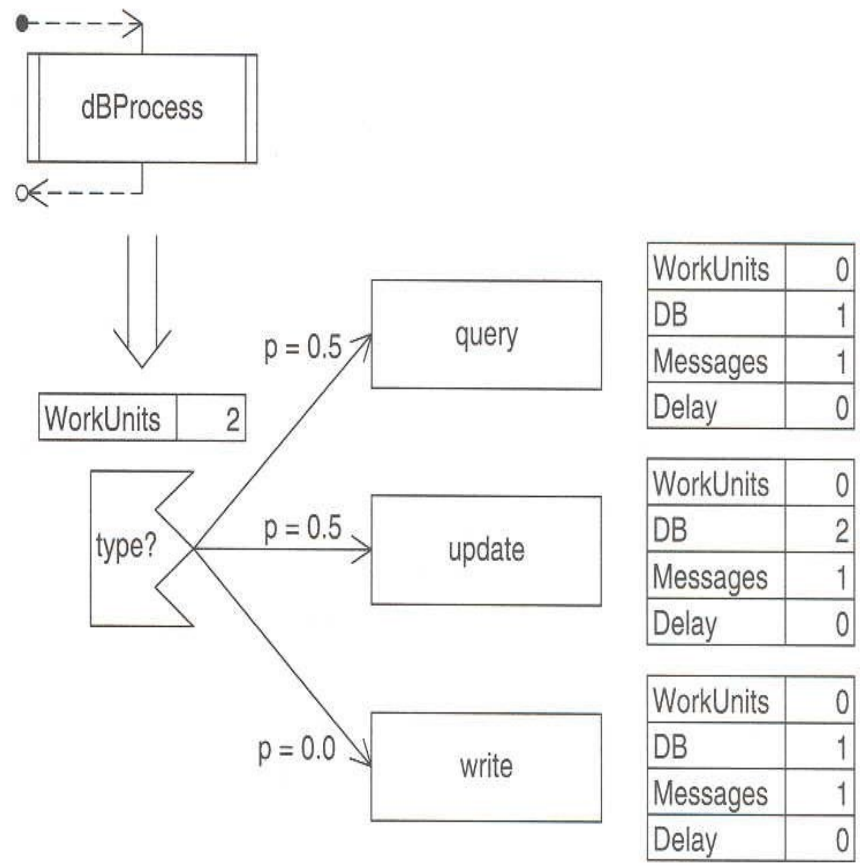


Figure 5-11: Server Processing Steps

Order Process Scenario

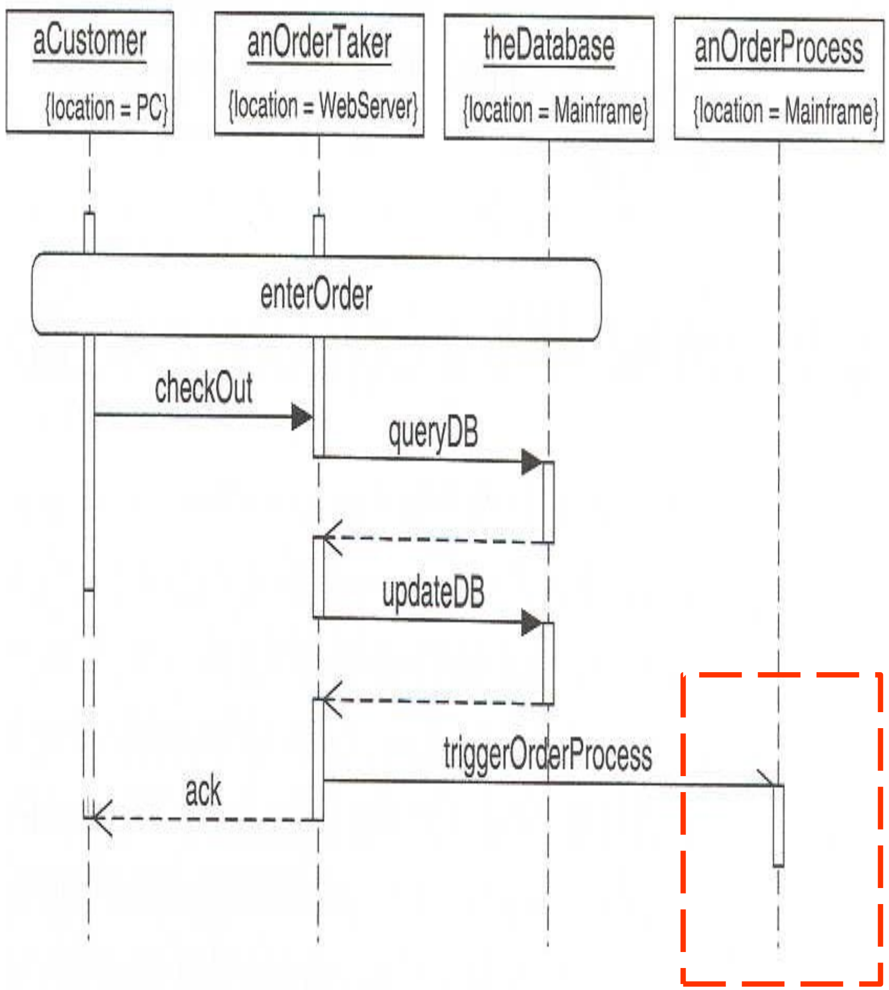
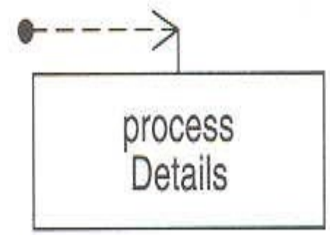


Figure 5-6: checkOut Scenario



WorkUnits	200
DB	20
Messages	0
Delay	0

Figure 5-12: processOrder Scenario

Example Summary

- This example has illustrated the construction and solution of software execution models for distributed systems
- We model scenarios individually, using estimates for delays introduced by communication and synchronization with objects on other processors
- The models can be solved iteratively to refine these estimates, if necessary