CptS 323 Software Design

Lecture #23

System Design – Part 3

Some of the slides have been adopted from the supplementary notes provided for Object-Oriented Software Engineering textbook

Instructor: Bolong Zeng

WASHINGTON STATE UNIVERSITY
World Class. Face to Face.

System Design

✓1. Design Goals

Definition
Trade-offs

✓2. Subsystem Decomposition

Design Principles
Coherence/Coupling
Architectural Patterns

3. Hardware/ Software Mapping

Special Purpose
Buy vs Build
Allocation of Resources
Connectivity

4. Persistent Data Management

Persistent Objects
File system vs Database

5. Access Control

Global Access Table vs Access Control List vs Capabilities
Security

6. Global Control Flow

Procedure-Driven
Event-Driven
Threads

7. Services

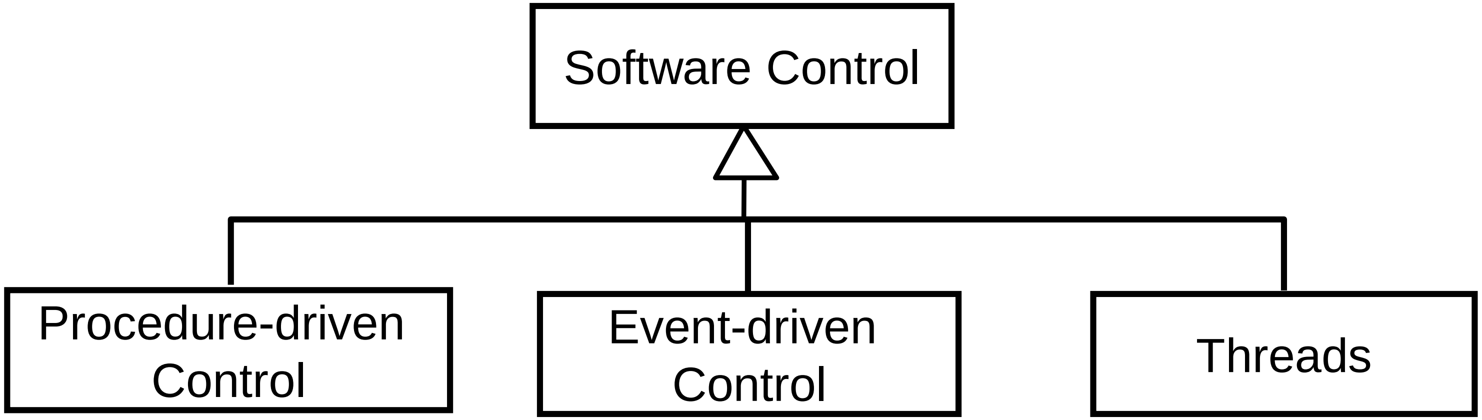
Procedure-Driven
Event-Driven
Threads

8. Boundary Conditions

Initialization
Termination
Failure

6. Designing the Global Control Flow

- Control flow is the sequencing of actions in a system.
- Three major design choices for control flow:
  1. Procedure driven control
  2. Event driven control
  3. Threads



Procedure-Driven Control

- Control resides within program code
- Operations wait for input whenever they need data from an actor
- Used in legacy systems and systems written in procedural languages
- Introduces difficulties when used with object-oriented languages

```
    Stream in, out;
    String userid, passwd;
/* Initialization omitted */
    out.println("Login:");
    in.readln(userid);
    out.println("Password:");
    in.readln(passwd);
    if (!security.check(userid, passwd)) {
        out.println("Login failed.");
        system.exit(-1);
    }
/* ...*/
```

Event-Driven Control

- Control resides within a dispatcher calling functions via callbacks
- Whenever an event becomes available, it is sent to the appropriate object
- Enables a simpler structure and centralizing all input in the main loop
- Harder to implement.

```
Iterator subscribers, eventStream;
Subscriber subscriber;
Event event;
EventStream eventStream;
/* ... */
while (eventStream.hasNext()) {
    event = eventStream.next();
    subscribers = dispatchInfo.getSubscribers(event);
    while (subscribers.hasNext()) {
        subscriber = subscribers.next()) {
        subscriber.process(event);
    }
}
/* ... */
```

Threaded Control

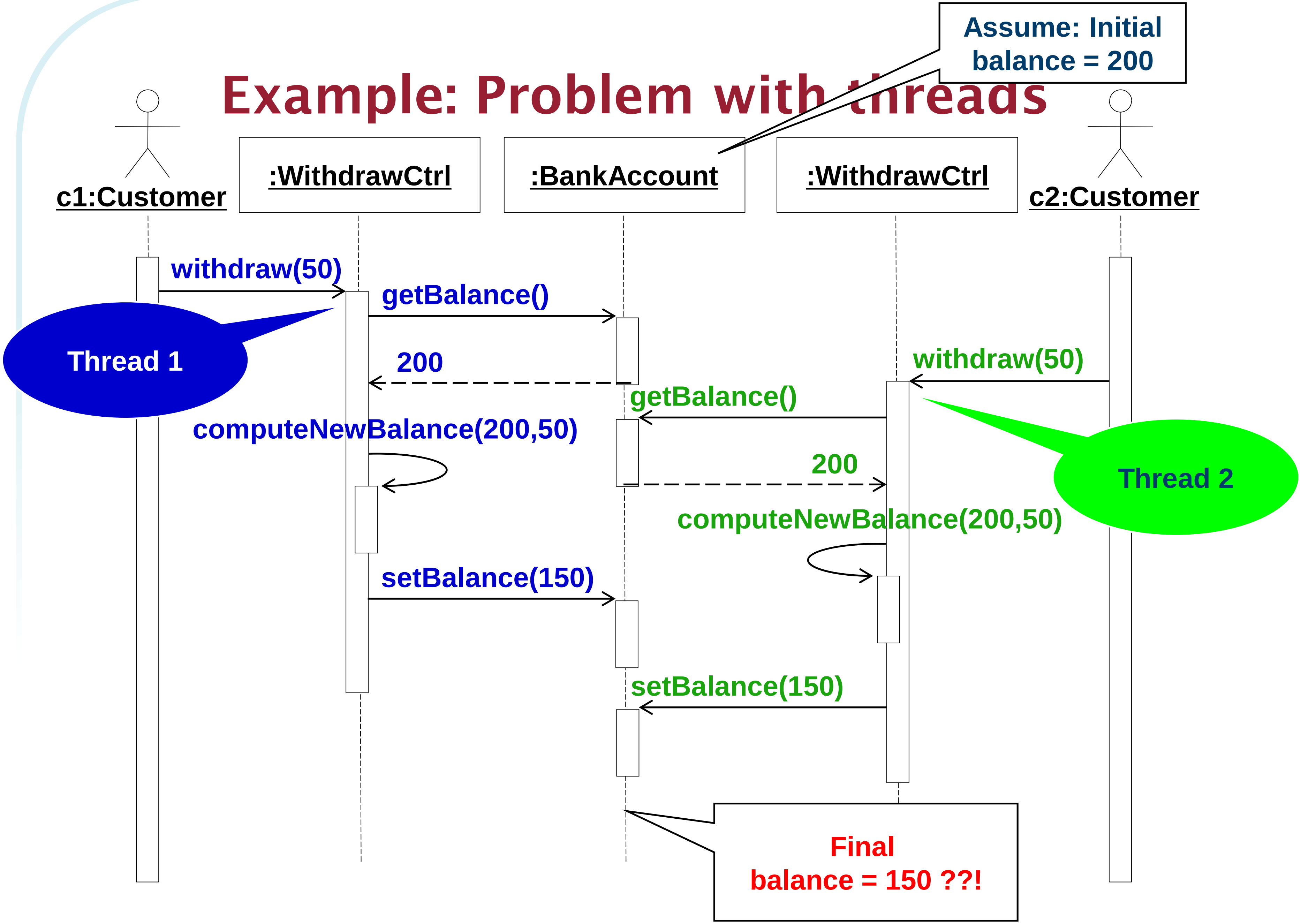
- Threads are the concurrent variation of procedure driven control, where each execute a certain task.
- The system can create arbitrary number of threads each corresponding to a different event
- Suits well to object-oriented languages
- Threaded control flow is the most intuitive, but hard to test.

```
Thread thread;
Event event;
EventHandler eventHandler;
boolean done;
/* ...*/
while (!done) {
    event = eventStream.getNextEvent();
    eventHandler = new EventHandler(event)
    thread = new Thread(eventHandler);
    thread.start();
}
/* ...*/
```

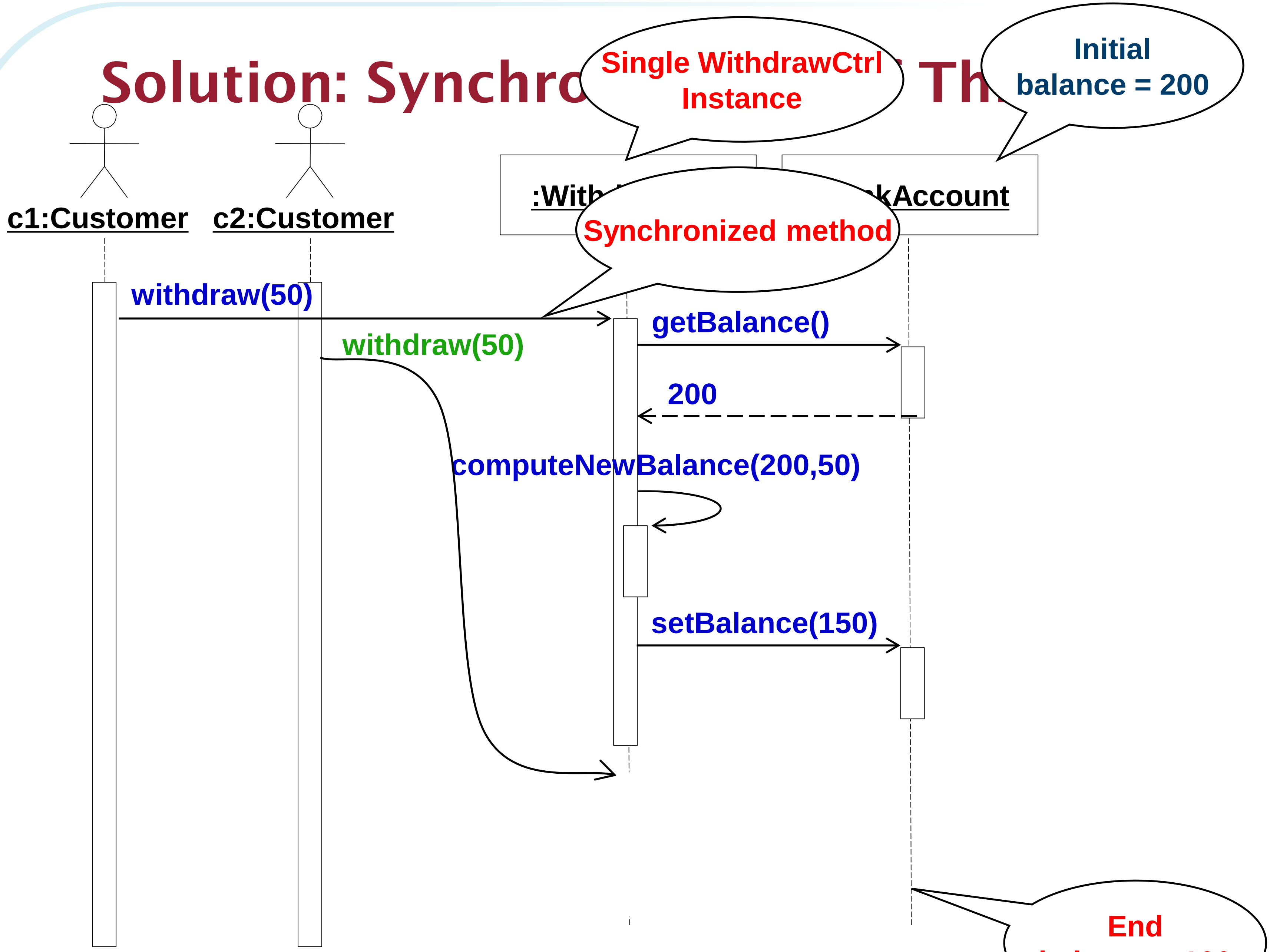
Concurrency

- Non-functional requirements to be addressed: Performance, response time, latency, availability.
- Two objects are inherently concurrent if they can receive events at the same time without interacting
- Inherently concurrent objects can be assigned to different threads of control
- Objects with mutual exclusive activity could be folded into a single thread of control

Example: Problem with threads



Solution: Synchro... Th...



Concurrency Questions

- To identify threads for concurrency we ask the following questions:
  - Does the system provide access to multiple users?
  - Which entity objects of the object model can be executed independently from each other?
  - What kinds of control objects are identifiable, can they run concurrently?
  - Can a single request to the system be decomposed into multiple requests? Can these requests and handled in parallel? (Example: a distributed query)

Implementing Concurrency

- Concurrent systems can be implemented on any system that provides
  - Physical concurrency: Threads are provided by hardware

  or

  - Logical concurrency: Threads are provided by software
- Physical concurrency is provided by multiprocessors and computer networks
- Logical concurrency is provided by threads packages.
  - Example: Java has a thread abstraction

Implementing Concurrency – Scheduling

- Issues: starvation, deadlocks, fairness
  -> Topic for researchers in operating systems

- Today's operating systems provide a variety of scheduling mechanisms:
  - Round robin, time slicing, collaborating processes, interrupt handling

- Sometimes we have to solve the scheduling problem ourselves
  - Which thread runs when?
  - Topic addressed by software control

System Design

✓1. Design Goals
- Definition
- Trade-offs

✓2. Subsystem Decomposition
- Design Principles
  - Coherence/Coupling
- Architectural Patterns

3. Hardware/ Software Mapping
- Special Purpose
- Buy vs Build
- Allocation of Resources
- Connectivity

4. Persistent Data Management
- Persistent Objects
- File system vs Database

5. Access Control
- Global Access Table vs Access Control List vs Capabilities
- Security

6. Global Control Flow
- Procedure-Driven
- Event-Driven
- Threads

7. Services
- Procedure-Driven
- Event-Driven
- Threads

8. Boundary Conditions
- Initialization
- Termination
- Failure

7. Identifying Services

- Review each dependency between subsystems
- Refine the subsystem responsibilities
- Identify the services provided by each subsystem
- Identify the services required by each subsystem
- Define an interface for each service identified

- Rationale: By focusing on services at the architectural abstraction level, we can reassign responsibilities between subsystems when needed, without changing many modeling elements.
  - Each service will be precisely specified in terms of attributes, operations and constraints during Object Design

System Design

✓1. Design Goals
- Definition
- Trade-offs

✓2. Subsystem Decomposition
- Design Principles
  - Coherence/Coupling
- Architectural Patterns

3. Hardware/ Software Mapping
- Special Purpose
- Buy vs Build
- Allocation of Resources
- Connectivity

4. Persistent Data Management
- Persistent Objects
- File system vs Database

5. Access Control
- Global Access Table vs Access Control List vs Capabilities
- Security

6. Global Control Flow
- Procedure-Driven
- Event-Driven
- Threads

7. Services
- Procedure-Driven
- Event-Driven
- Threads

8. Boundary Conditions
- Initialization
- Termination
- Failure

8. Identifying Boundary Conditions

- Initialization
  - The system is brought from a non-initialized state to steady-state
- Termination
  - Resources are cleaned up and other systems are notified upon termination
- Failure
  - Possible failures: Bugs, errors, external problems
- Good system design foresees fatal failures and provides mechanisms to deal with them.

Boundary Condition Questions

- Configuration
  - In which use cases are the persistent objects created?
- Initialization
  - What data need to be accessed at startup time?
  - What services have to registered?
  - What does the user interface do at start up time?
- Termination
  - Are single subsystems allowed to terminate?
  - Are subsystems notified if a single subsystem terminates?
  - How are updates communicated to the database?
- Failure
  - How does the system behave when a node or communication link fails?
  - How does the system recover from failure?.

Modeling Boundary Conditions

- Boundary conditions are best modeled as use cases with actors and objects
- We call them boundary use cases or administrative use cases
- Actor: often the system administrator
- Interesting use cases:
  - Start up of a subsystem
  - Start up of the full system
  - Termination of a subsystem
  - Error in a subsystem or component, failure of a subsystem or component.

Exception Handling

- Defines how the system would react to component failures
- Exceptions are caused by three different sources:
  - Hardware failure
  - Changes in the operating environment
  - Software fault
- Handling exceptions:
  - Design components to tolarate the failure
    - Might result in changes in the system decomposition
  - Write boundary use cases to specify how the user will experince failure

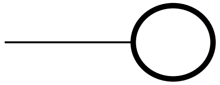
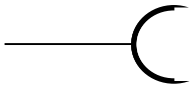
Summary

- System design activities:

Hardware/Software Mapping
  - Persistent Data Management
  - Providing Access Control
  - Designing the Global Control Flow
  - Identifying Services
  - Identifying Boundary Conditions
- Each of these activities may affect the subsystem decomposition
- Two new UML Notations
  - UML Component Diagram: Showing compile time and runtime dependencies between subsystems
  - UML Deployment Diagram: Drawing the runtime configuration of the system.

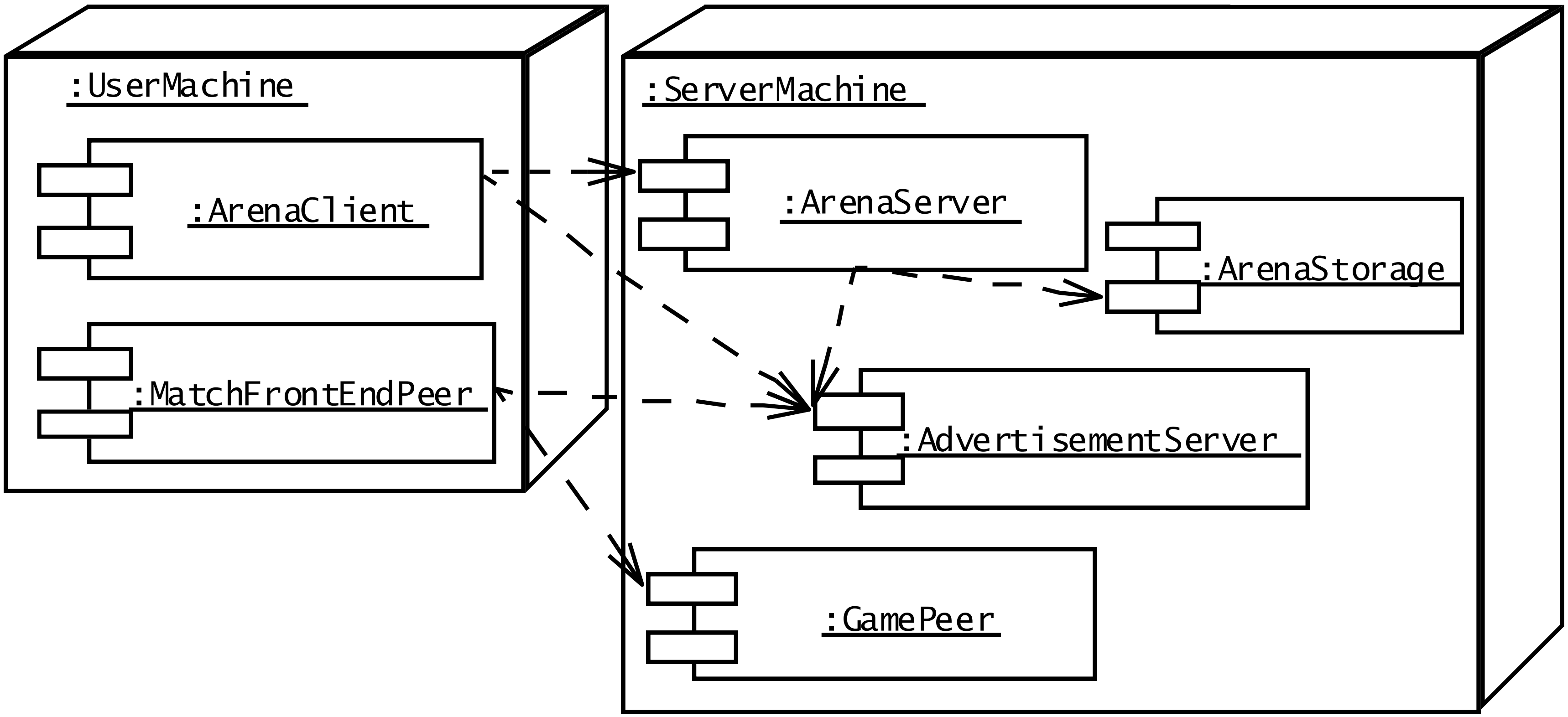
Additional Slides

UML Interfaces: Lollipops and Sockets

- A UML interface describes a group of operations used or created by UML components.
  - There are two types of interfaces: provided and required interfaces.
    - A provided interface is modeled using the lollipop notation 
    - A required interface is modeled using the socket notation. 

- A port specifies a distinct interaction point between the component and its environment.
  - Ports are depicted as small squares on the sides of classifiers.

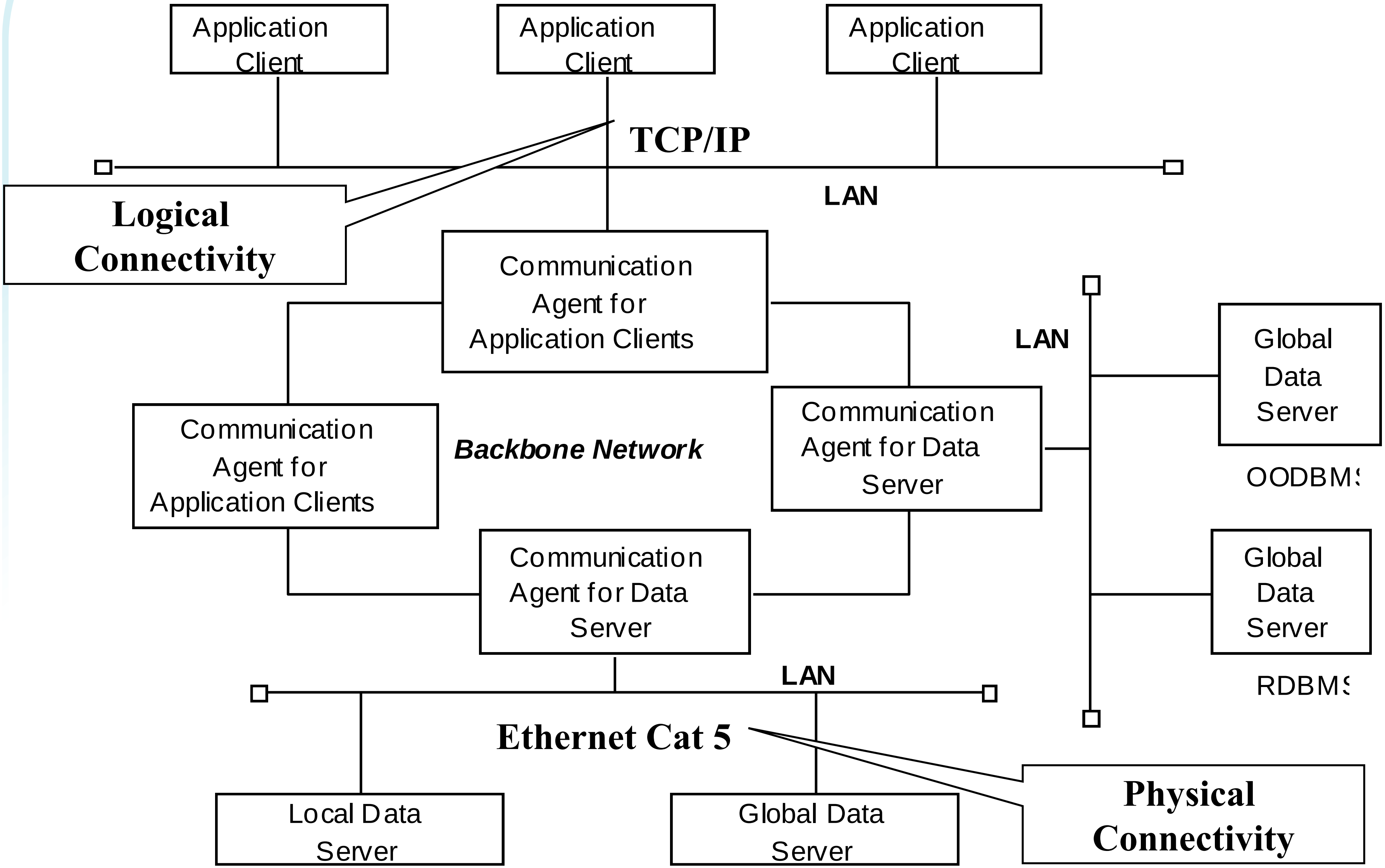
ARENA Deployment Diagram (UML 1.0 Notation)



Mapping the Associations: Connectivity

- Describe the physical connectivity
  - ("Physical layer in the OSI reference model")
    - Describes which associations in the object model are mapped to physical connections
- Describe the logical connectivity (subsystem associations)
  - Associations that do not directly map into physical connections
  - In which layer should these associations be implemented?
- Informal connectivity drawings often contain both types of connectivity

Example: Informal Connectivity Drawing



Logical vs Physical Connectivity and the Relationship to Subsystem Layering