KING'S College LONDON

University of London

Software Architecture & Design

6CCS3SAD

Dr Kevin Lano

KING'S College LONDON

University of London

Session 4: More Styles, Complex Connectors

February 7th, 2020

Learning Outcomes

At the end of this session you should be able to:

- Explain different types of connections; and
- Explain and apply different basic architectural styles.

So far

- **Last week we began to look at architectural styles:**
  - Abstraction of recurring composition and interaction characteristics
  - Parts:
    - A vocabulary of design elements
    - A set of configuration rules, and
    - A semantic interpretation

Today

- **Continue through our (incomplete) list of domain-independent styles**
- **Next week: Move on to domain-specific styles**

Simple Architectural Styles

- **Domain/platform-independent styles**
  - Pipes and filters
  - 2-tiered
  - N-tiered
  - Layered
  - Blackboard
  - Model-View-Controller

Advanced Connectors

- **Some styles require different types of connectors**
  - Publish-Subscribe
  - Asynchronous communication

Advanced Connectors

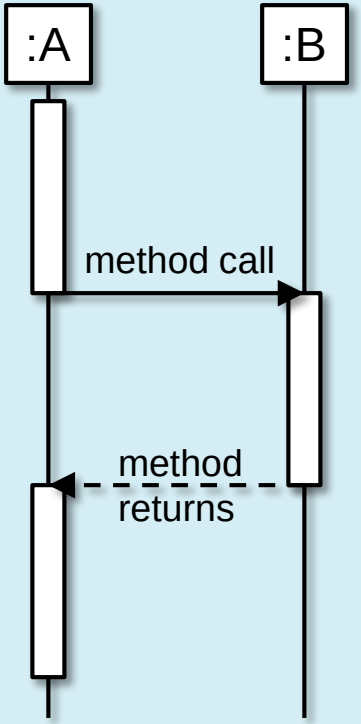
3 different message passing mechanisms

PART I

UML2 Connectors in Detail

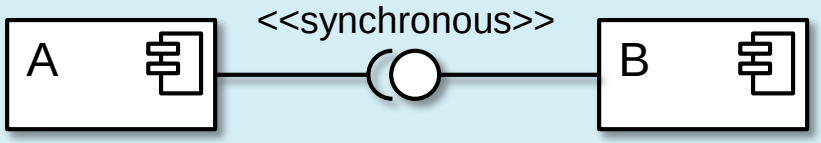
- **An architectural abstraction of usage between components**
  - Many possible implementations
  - Our simple Java implementation: a reference to one component class B within another A
    - Object implementing A communicates with the object for B by calling methods on B's interface
    - **Synchronous communication:** A must wait for method calls to finish before continuing

Synchronous Communication

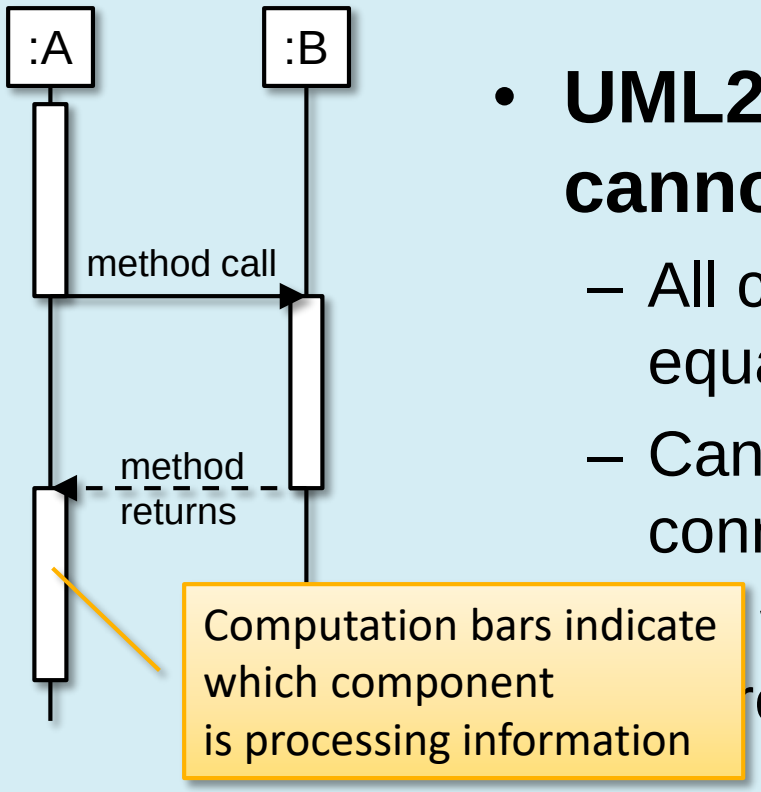


UML Sequence Diagram: shows object lifelines & messages

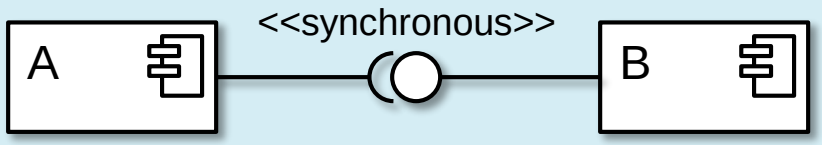
- **UML2 Component Diagrams cannot show this directly**
  - All connectors are considered equal by default
  - Can define stereotypes to adjust connector semantics
  - We will use <<synchronous>> stereotype



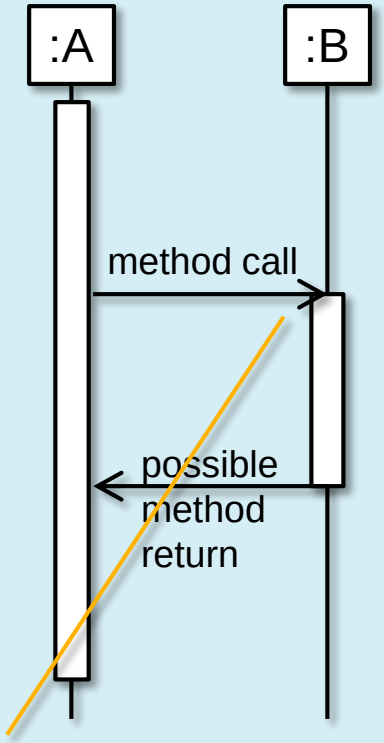
Synchronous Communication



- **UML2 Component Diagrams cannot show this directly**
  - All connectors are considered equal by default
  - Can define stereotypes to adjust connector semantics
  - will use <<synchronous>> reotype

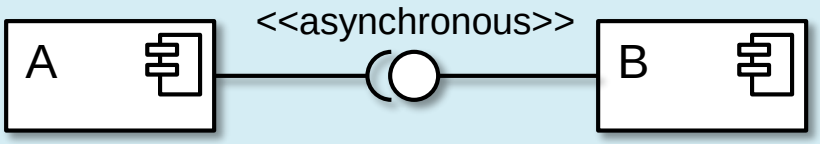


Asynchronous Communication



Notice the different arrow heads
This is how UML2 indicates asynchronous calls.

- **`A` can continue computation after sending call to `B`**
  - No need to wait
  - Good for decoupling components
  - Good for shielding against communication delays
- **Will use `<<asynchronous>>` stereotype**



Excursion: Implementing Asynchronous Calls

- **Asynchronous calls implemented with synchronous methods:**
  - Call immediately returns a ‘future’ result object
    - Implementation depends on details of message passing algorithm (e.g., networked message queue, ...)
  - Future object can be queried
    - Result available yet?
    - What is the result?
    - Block until a result is available

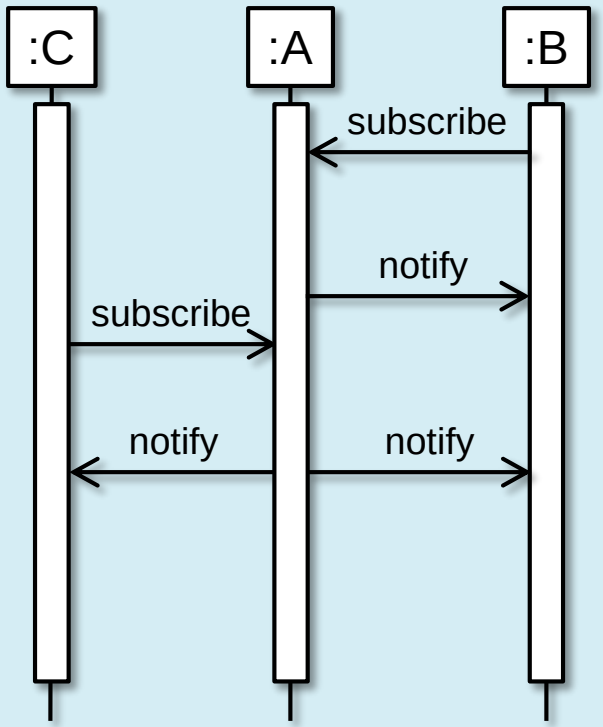
Excursion: Implementing Asynchronous Calls (2)

```
public class A {
  private IRequiredInterface b;
  public void doSomething() {
    // ...
    // asynchronous call
    Future<String> f = getExecutorService()
                       .submit (new Callable() {
                                  public String call() {
                                    return b.doSomethingB();
                                  }
                                });
    // ...
    // This statement will block until b.doSomething() finishes:
    String s = f.get();
    // ...
  }
}
```

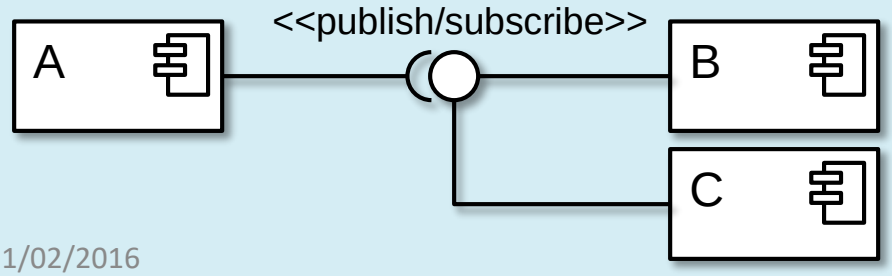
This will execute method **`call()`** in a separate thread and allow **`doSomething()`** to continue.

See: https://docs.oracle.com/javase/tutorial/essential/concurrency/exinter.html

Publish/Subscribe Connector



- **Components subscribe to events from publisher**
  - Publisher does not know subscribers
  - Sends event to all subscribers
- **We will use `<<publish/subscribe>>` stereotype**



More domain-independent styles

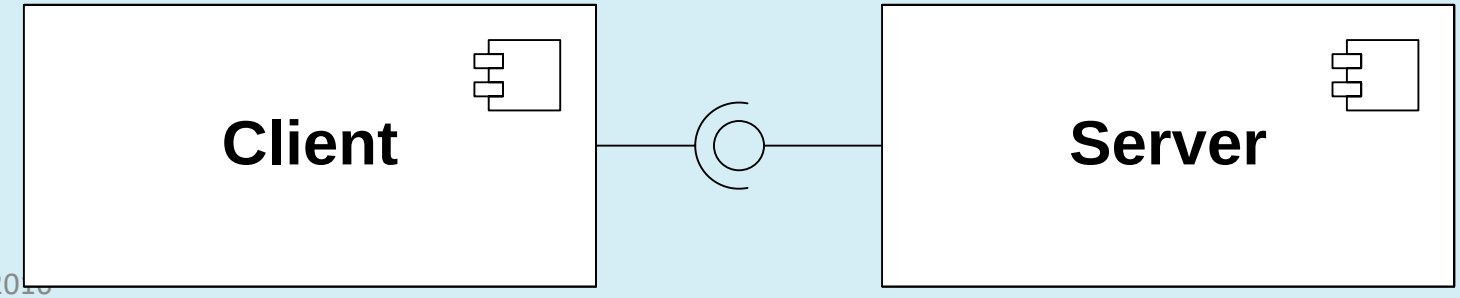
PART II

Simple Architectural Styles

- **Domain/platform-independent styles**
  - Pipes and filters
  - 2-tiered
  - N-tiered
  - Layered
  - Blackboard
  - Model-View-Controller

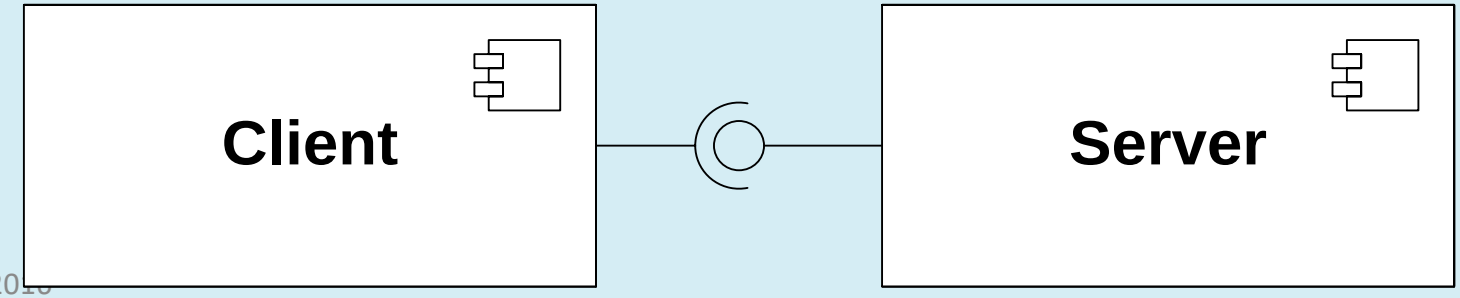
Style 2: 2-tiered (Client/Server)

- **One of most common styles in distributed systems**
- **Components are clients or servers**
  - Clients request services from servers
  - Clients and servers may be composite
- **Constraints**
  - Server provides services to the client, but does not require services from client
  - Clients require services of the server
  - A larger number of clients than servers



Style 2: 2-tiered (Client/Server) (2)

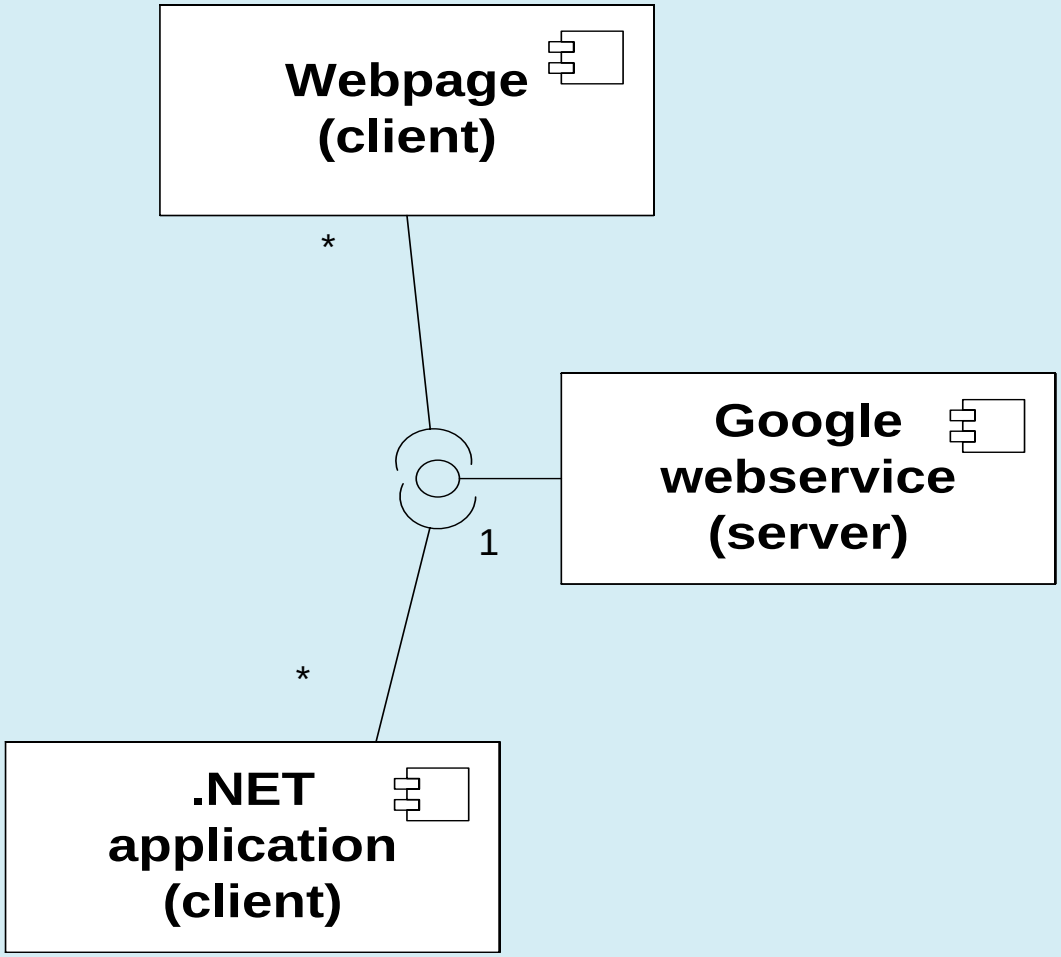
- **Semantics (meaning and implementation)**
  - Servers do not know the number of or identities of clients
  - Clients know server's identity
  - Clients and servers are often distributed (RPC used to implement connections)
- **Useful whenever there is a need to separate application logic into two tiers**



Style 2: Client/Server: Example

- **Google offers search webservice**
- **A range of clients use the webservice**
  - Webpage servlet clients use it as part of their pages
  - .NET based b-2-c application uses webservice to search for best buys of products
- **Different clients**
  - Very different purposes
  - But the same server functionality
- **Keep web searching and client functionality separate to**
  - Maximize possible uses of the server
  - Enable better maintenance

Style 2: Client/Server: Example



Simple Architectural Styles

- **Domain/platform-independent styles**
  - Pipes and filters
  - 2-tiered
  - N-tiered
  - Layered
  - Blackboard
  - Model-View-Controller

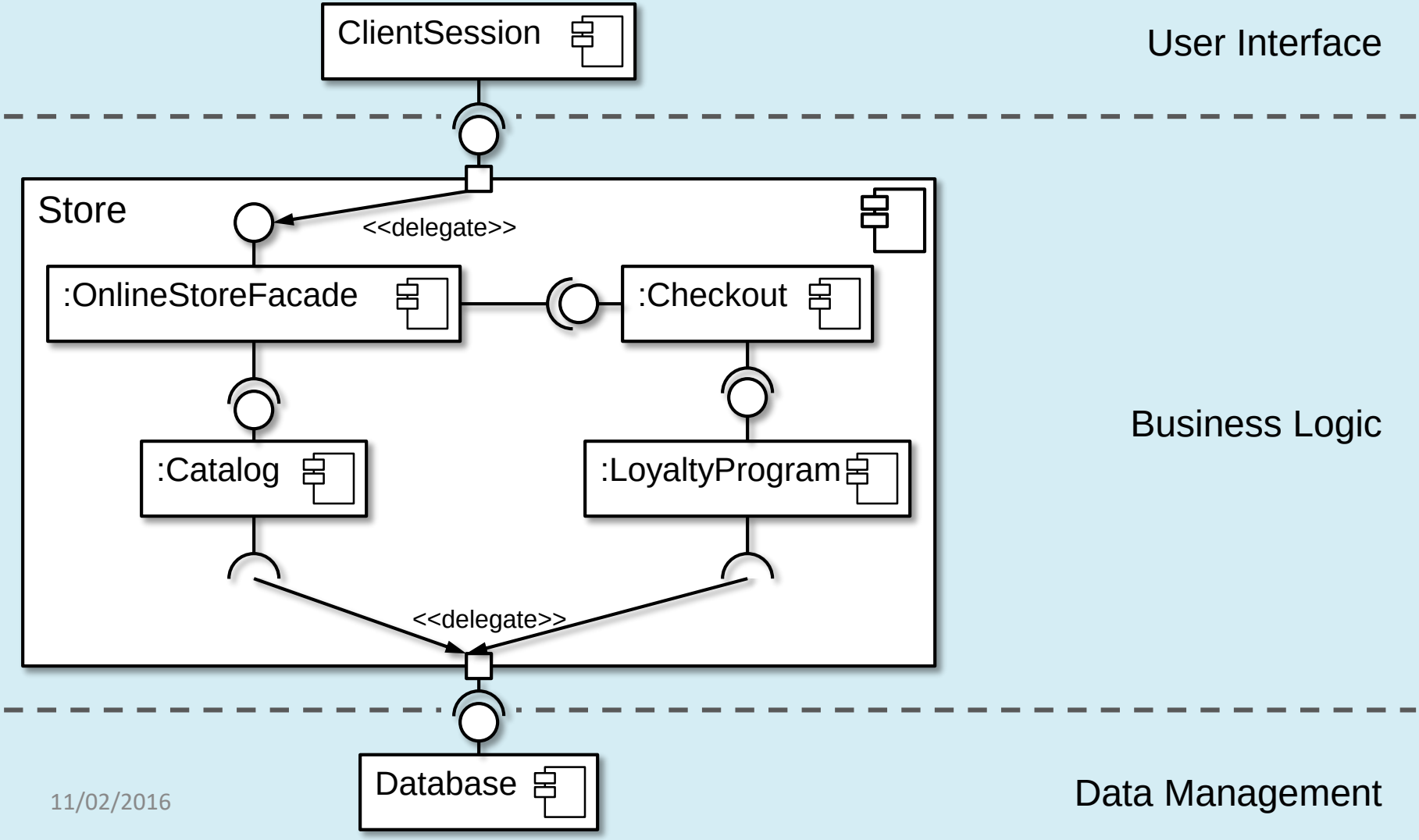
Style 3: N-tiered

- **Generalises client / server**
- **Servers can also be clients of other servers**
  - Application logic is divided between the chain of components
  - Clients and servers may be composite
- **Constraints**
  - Servers provide services to their clients, but do not require services from them
  - Clients require services of the server

Style 3: N-tiered

- **Semantics**
  - Servers do not know the number or identities of their clients
  - Clients know server's identity
  - Components are usually distributed (RPC used to implement connections)
- **Useful to separate application logic into software packages**

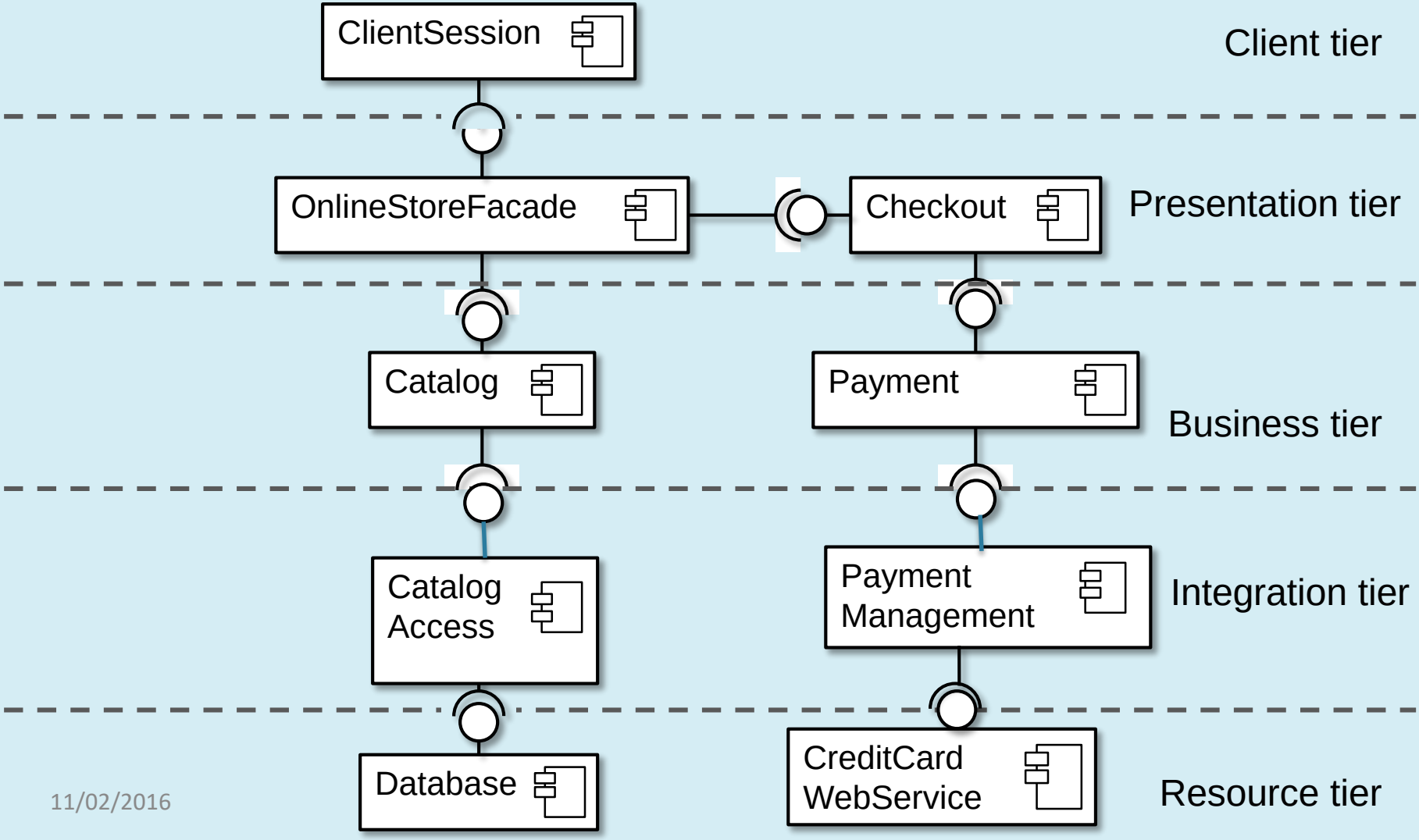
Example
A Typical 3-tiered Architecture



Enterprise System Architectures

- **Typically 5 tiers:**
  - *Client tier*: UI components for direct interaction with clients
  - *Presentation tier*: UI logic components & helpers
  - *Business tier*: business functions and logical representations of business data
  - *Integration tier*: components mediating between business tier & data sources
  - *Resource tier*: databases, web interfaces, etc

Example: EIS 5-tier architecture



Style 3: N-tiered

- **Advantages**
  - Separation of concerns, divide and conquer approach to solving large scale problems
  - Maintenance
- **Disadvantages**
  - Performance (especially if distributed components)
    - Need to introduce caching or otherwise limit network usage

BREAK

Simple Architectural Styles

- **Domain/platform-independent styles**
  - Pipes and filters
  - 2-tiered
  - N-tiered
  - Layered
  - Blackboard
  - Model-View-Controller

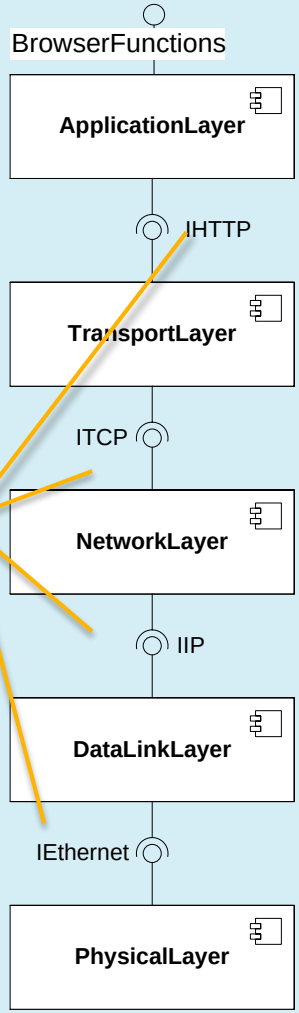
Style 4: Layered architectures

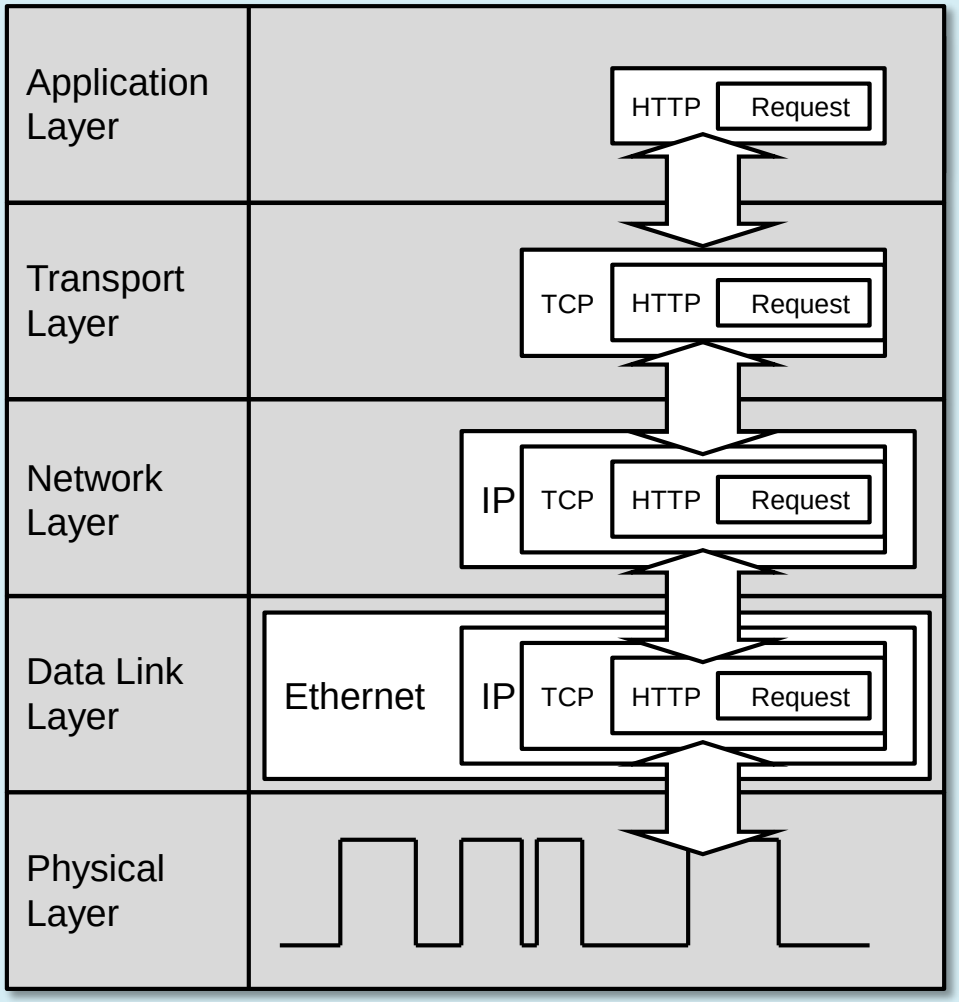
- **Hierarchical system organization**
  - "Multi-level client/server"
  - Components are classified by layers
  - Each layer exports an interface to be used by above layers
- **Constraints:**
  - 1-1 relation between each layer
  - Each layer communicates only with its neighbours below it
- **Connectors are protocols of layer interaction**
- **Examples**
  - Operating systems: Virtual machine style results from fully opaque layers
  - Network communications

Key difference to N-tiered style

Example: HTTP – Protocol stack in UML 2

Each layer communicates to the next via an interface, taking information according to a certain network protocol





Style 4: Layered Architectures

- **Advantages**
  - Increased abstraction
  - Maintainability
    - Each layer's role is defined
  - Reuse
  - Portability
    - Replacing layers to work within a different context
- **Disadvantages**
  - Not universally applicable
    - Some components may be spread around different layers
    - Layer bridging may be required – a layer might need to communicate directly with a non-neighbour
  - Performance
  - Determining the correct abstraction level

Simple Architectural Styles

- **Domain/platform-independent styles**
  - Pipes and filters
  - 2-tiered
  - N-tiered
  - Layered
  - Blackboard
  - Model-View-Controller

Style 5: Blackboard (Repository)

- **Two kinds of components**
  - Blackboard
    - Central data structure (sometimes called repository)
  - Clients
    - Operate on and receive information from the blackboard
- **System control driven by blackboard state**
- **Constraints**
  - 1 blackboard
  - 1..n clients

Style 5: Blackboard

- **Semantics**
  - Clients
    - Encapsulate individual pieces of application functionality
    - Interact only through blackboard
    - Make changes to the blackboard that lead incrementally to a desired outcome
  - Blackboard
    - Active component
    - State change triggers selection of clients to execute
  - Connectors
    - Publish/Subscribe for clients to be informed of blackboard changes
    - Separate (a)synchronous connectors for changing the blackboard state

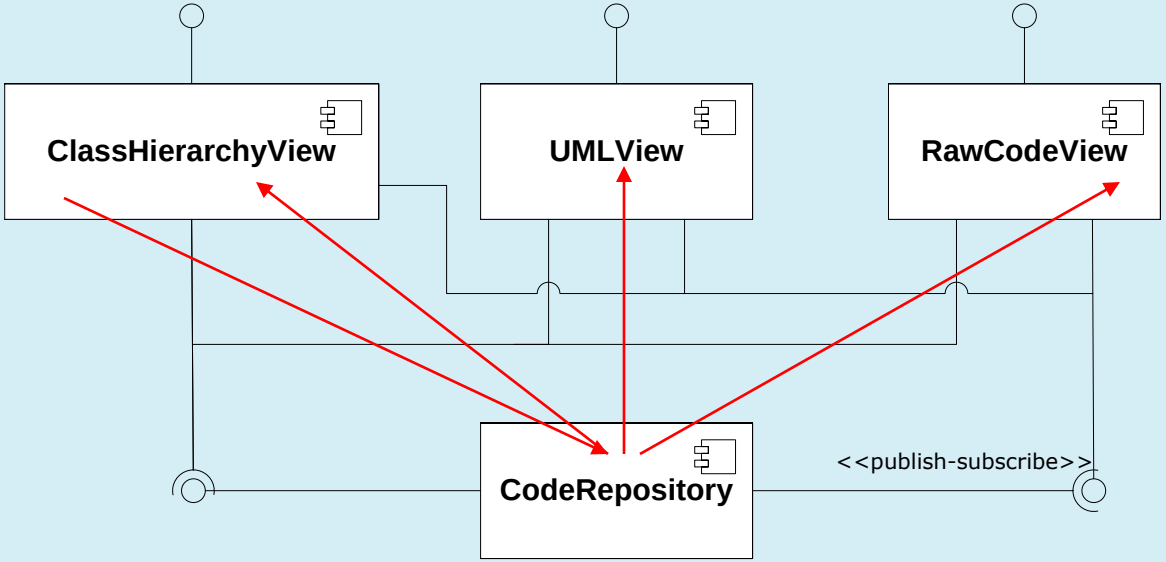
Style 5: Blackboard Examples

- **AI systems**
- **Integrated software developments environments (IDEs)**
  - E.g., Visual Studio, Borland Development Environment, Eclipse
- **Compiler architecture**
- **In e-commerce:**
  - Stateless components driven by stateful data components

Style 5: Blackboard 1st Example

- **Software development environment**
  - Stateful repository maintains software-project code
  - Three stateless components present views on the code
    - Change in one changes the code in the repository
    - This results in parallel changes to the other views

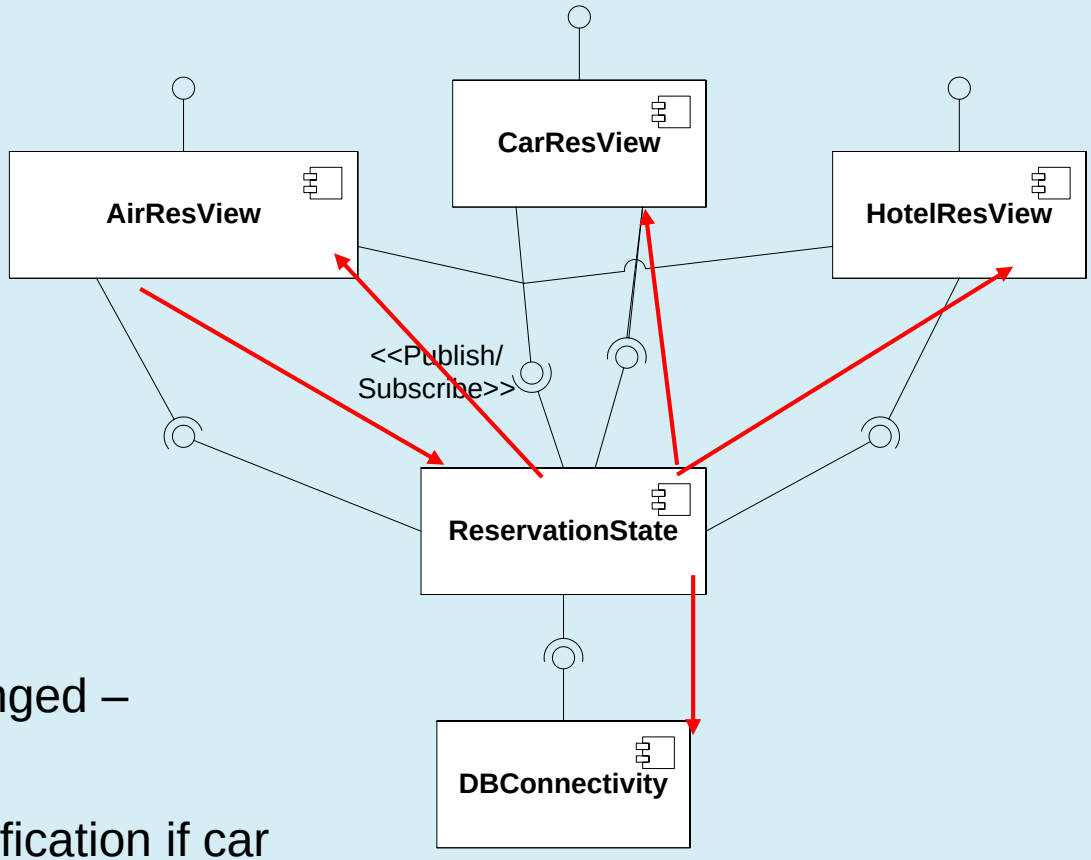
Changing class hierarchy (e.g., making one class inherit from another) results in change to code in repository and consequent updates to UML visual class structure and code view



Style 5: Blackboard 2nd Example

- **Holiday reservation system**
  - All reservations ultimately stored in database
  - Three components for different aspects of holiday reservation logic
  - Each component could connect to the database separately
    - Would tax performance
    - Especially as a change in one aspect (like date of air reservation) affects another aspect (like valid dates to book a hotel)
  - Solution: use an intermediate single blackboard component
    - Connects to database
    - Deals with all data of a user's reservation
  - Three reservation logic components can change the blackboard
  - Components can also subscribe to be notified of important changes to the blackboard data

Style 5: Blackboard 2nd Example



1. Dates of aeroplane ticket changed – ReservationState updated
2. ReservationState sends a notification if car reservation or hotel reservation dates do not lie within aeroplane dates
3. Only valid data is sent to database.

Style 5: Blackboard

- **Advantages**
  - Clients are relatively independent of each other
  - Data store is independent of clients
    - Scalable (new clients can be added easily)
    - Modifiable, maintainable
  - Data integration can be achieved through the blackboard

Simple Architectural Styles

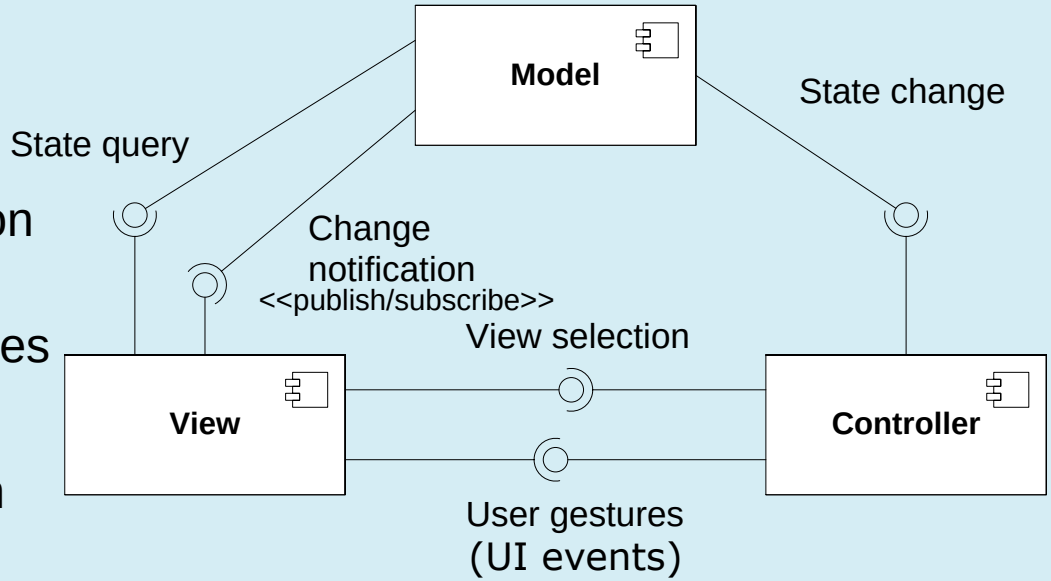
- **Domain/platform-independent styles**
  - Pipes and filters
  - 2-tiered
  - N-tiered
  - Layered
  - Blackboard
  - Model-View-Controller

Style 6: Model-View-Controller

- **Specifically for supporting multiple views and modifiers for shared data**
- **Three different kinds of components**
  - View
    - Responsible for displaying information to the user
  - Controller
    - Responsible for managing interaction with user
  - Model
    - Responsible for maintaining domain knowledge
    - Do not depend on any view or controller
    - Propagate changes in state to views via publish/subscribe

Style 6: Model-View-Controller

- **Semantics:**
  - Model
    - encapsulates application state and functionality
    - Notifies views of changes
  - View
    - Requests updates from models
    - Maps user gestures to controller actions
  - Controller
    - Maps user actions to model updates



Style 6: Model-View-Controller

- **Differences to Blackboard:**
  - Controller can implement coarse-grain operations in terms of fine-grain Model (blackboard) operations
  - In Blackboard, clients directly use the fine-grained blackboard operations
  - Controller has the responsibility to maintain integrity constraints for the Model data, in Blackboard this would need to be done within the blackboard or clients.
- **Generalises Observer:**
  - Views present up-to-date state of model: model notifies views when its state changes.
  - Same interaction as Observer pattern: View in the Observer role, Model in Subject role.

Heterogeneous Styles

PART III

Heterogeneous Styles

- **Combine different architectural styles**
  - Large architectures involve a combination of functionalities
  - Often entails a combination of styles in the best solution
- **Challenge:**
  - Ensure advantages of individual style choices are not negated by their combination

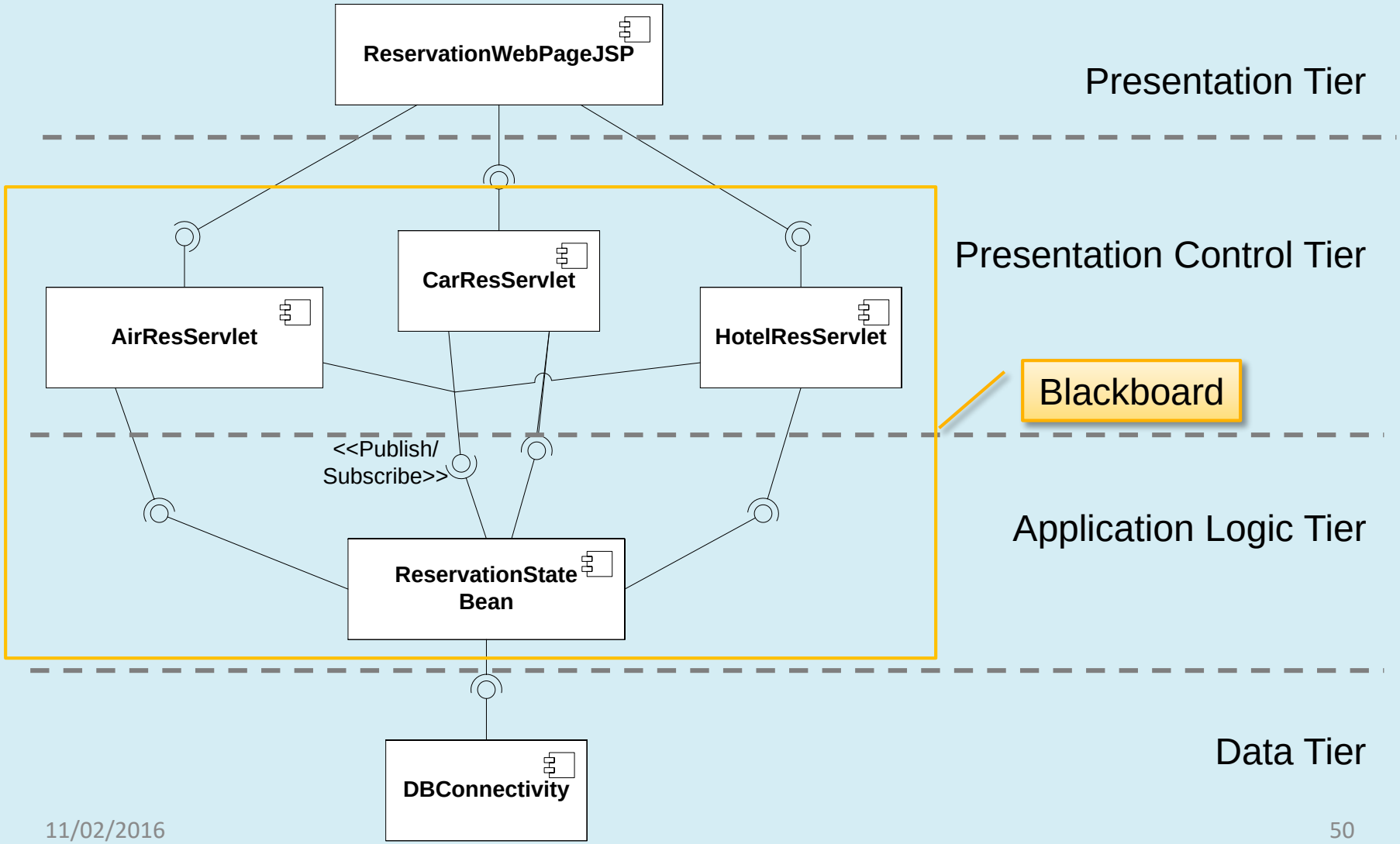
Heterogeneous Styles (2)

- **Good practice: Apply styles compositionally**
  - Decompose problem into subsystems
  - Apply one style per subsystem (e.g., in a composite component)
  - Helps avoid style benefits being in conflict

Heterogeneous styles

- **Safe practice: Mix styles addressing orthogonal issues**
  - Example:
    - N-tiered style is used to structure an overall architecture
    - Blackboard style is used for a specific kind of problem solving
    - Styles can be combined

Example: 4-tiered Blackboard



Coming up

- **.NET as a specific architecture implementation ("platform")**
- **Platform-specific styles – Enterprise Styles**