J2EE Patterns

A Pattern-Based Approach to Effective J2EE Application Design

Andy Longshaw, andy@blueskyline.com

Who am I?

- Independent consultant, writer, trainer, mentor
- Covering
  - .NET, J2EE, platforms, enterprise development
  - Design and architecture
  - Components
  - Web technologies, XML, B2B
- Why am I stood here?
  - Architecture work
  - Training and mentoring is part of what I do

Some important questions

- Who am I? ✓
- Who are you?
  - Developers, designers and technical architects who are interested in using J2EE patterns
  - Basic understanding of J2EE and a grounding in application development
- Why are you here?
  - Explore the context in which we are working
  - Examine some of the main J2EE patterns
  - Using J2EE patterns as part of application design

Expectations

- What this session *is not*
  - X In-depth masterclass for EJB, JSP, servlets, etc.
  - X Tricks and tips for particular J2EE containers
  - X Exhaustive analysis of each pattern
- What this *is*
  - √ Reduction in kludges by getting design right
  - √ When, where and why to apply the patterns
  - √ Application of patterns that work “with the flow”

Process

- Timing
  - Part 1 09.00 – 10.30
  - Part 2 10.40 – 12.00
  - Please come back promptly!
- Questions
  - Please ask as we go
  - A little time at the end

Agenda

- J2EE Patterns in Context
- Distribution and Web patterns
- Business and persistence patterns
- Pattern directions in J2EE

Agenda

- J2EE Patterns in Context
- Distribution and Web patterns
- Business and persistence patterns
- Pattern directions in J2EE

J2EE technologies

Distribution technologies
RMI, CORBA/IIOP, HTTP, JNDI

Alphabet soup

Enterprise beans
Stateless session, stateful session, entities, MDB, BMP, CMP

Web technologies
servlets, servlet filters, JSP, tag libraries

EAI connectivity
JMS, Connectors

Data and transactions
JDBC, RowSets, JDBC extensions and pooling, JTA + COS transactions

XML and web services
JAXP, JAX-RPC, SAAJ, JAXM, JAXR

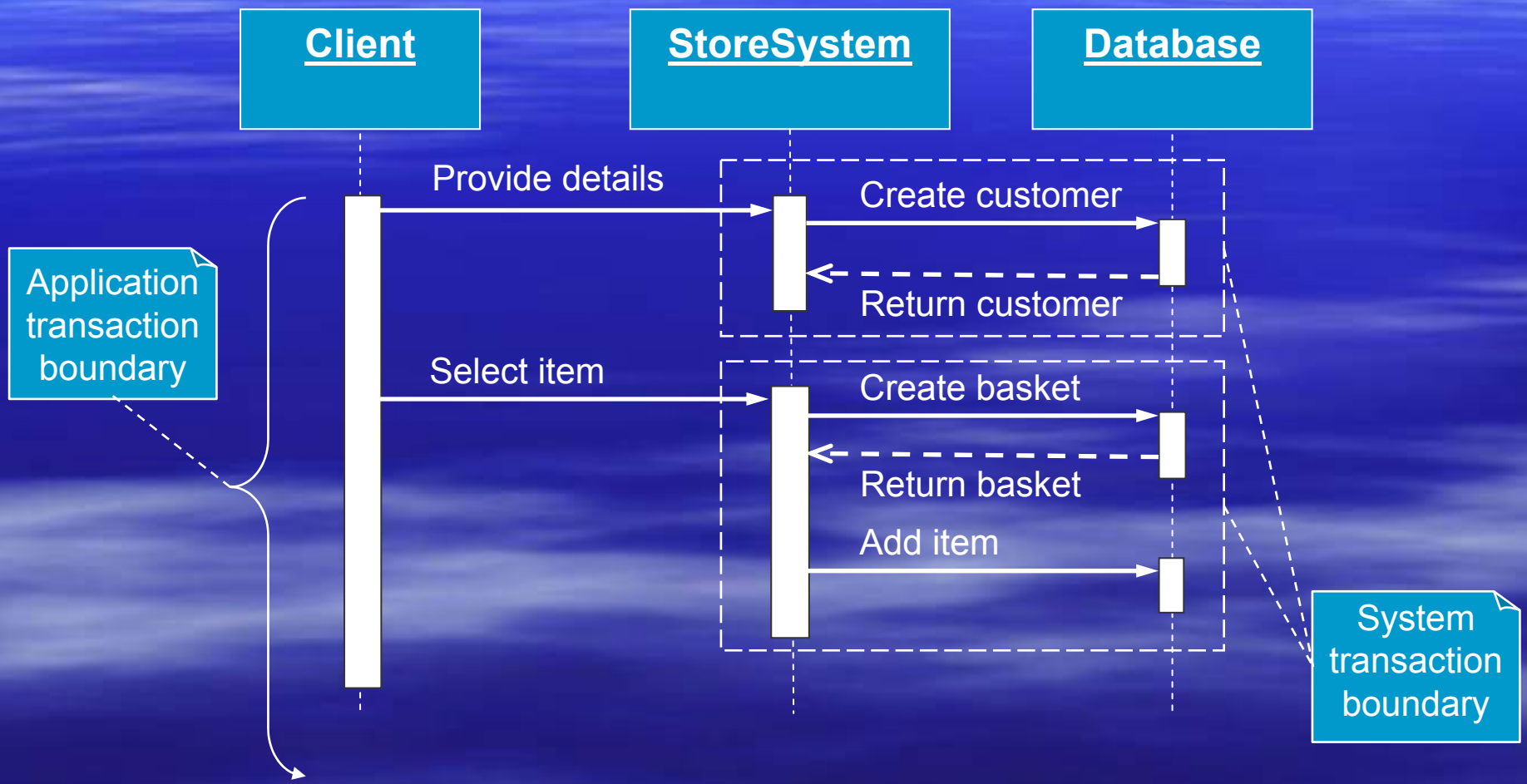
What are we trying to do?

- J2EE provides various choices and options
  - Lots of nice, CV-friendly technology
  - Product vendors deliver blueprints and prescriptive architecture guidance
  - Which bits do you really need to use?
- Let's stop and think
  - What is the point of the application?
  - What does it consist of?
  - How do we structure it?

Terminology: logic and data

Term	Example
Domain object	Customer
Domain logic	Must have an address
Business logic	Create a customer
Application logic	Place an order

Terminology: transactions



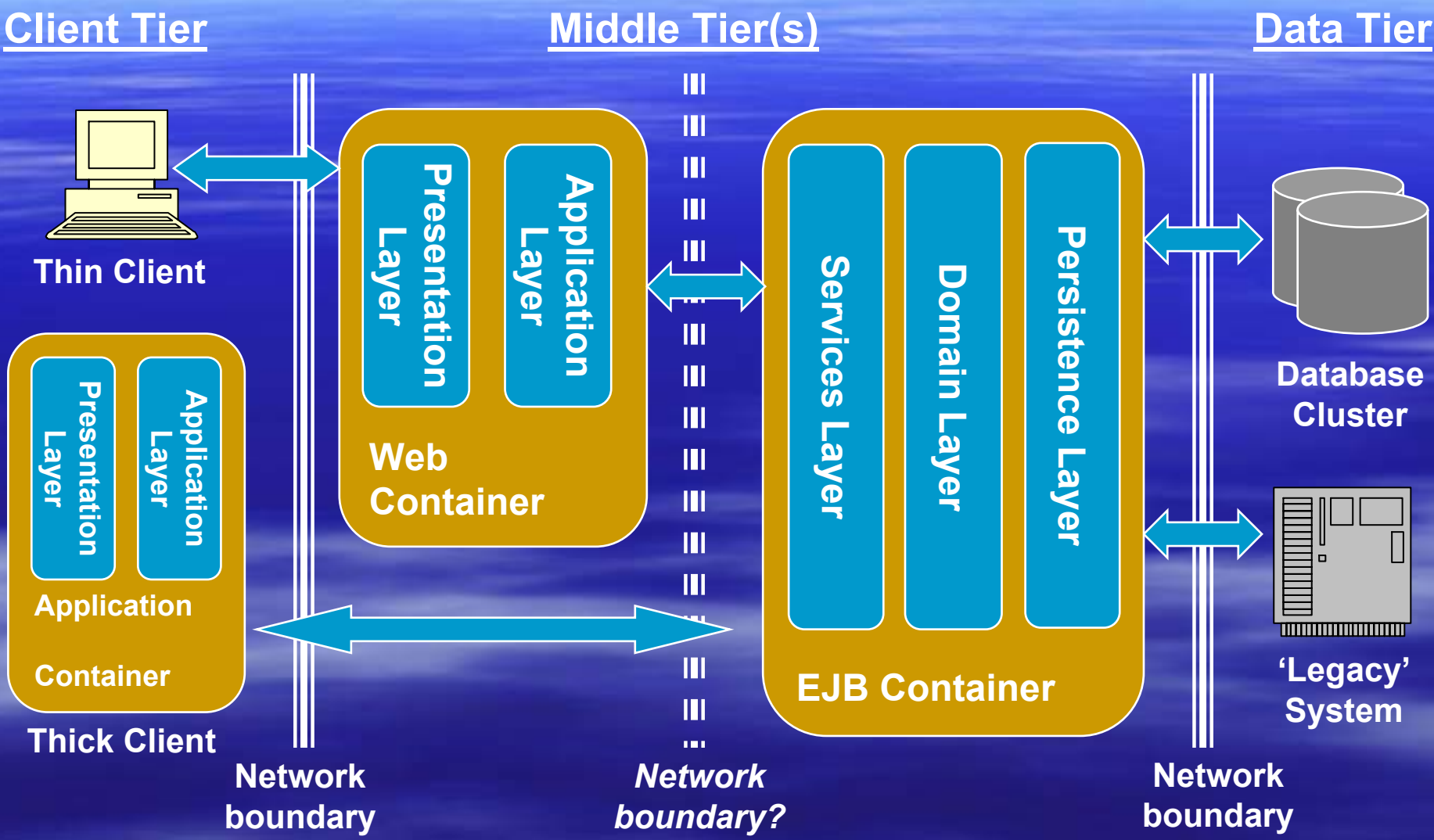
J2EE Application Architecture

Model based on "EJB Design Patterns" [*EJBD*]

Layer Name	Responsibilities
Presentation	UI rendering, B2B data delivery
Application	Application logic and transactions
Services	Business logic and system transactions
Domain	Domain objects and domain logic
Persistence	Store and retrieve domain data

- Reduce dependencies to create coherent layers
- Components within layers: EJBs, Servlets/JSPs, POJOs

Layers and Tiers



Tiers and Distribution

- There are definite network boundaries in the architecture
  - Client to application server
  - Application server to database
- Maybe another one between containers
  - For sharing functionality or due to security concerns
- Component communication
  - Same VM, you can call direct
  - Different VM, you need remote call semantics
  - Therefore cross-VM calls always have remote semantics
- RMI creates a pseudo-tier due to remote interface semantics
- Application designers and architects must deal with distribution issues

Pattern forms and discovery

- Patterns are identified, not invented!
- Pattern forms
  - Must have: problem, context, solution
  - Usually has: forces, example, resulting context
  - Can have others: related patterns
- No overall body that judges patterns
- Patterns vs. idioms vs. frameworks
- Implementation and pattern instantiation

Pattern sources

- Books
  - General: Alexander, PLoP series
  - Software design: GOF, POSA, POSA2, PEAA
  - Platform-specific: Core J2EE Patterns, EJB Design Patterns
- Web Sites
  - General: Hillside, Portland pattern repository
  - Software design: IBM e-commerce patterns
  - Platform-specific: theserverside.com, blueprints patterns

What are the benefits of patterns?

- A shared language between designers
  - You say Singleton, I say Singleton, let's not call the whole thing off…
- Reusable design concepts
- Solutions you didn't think of
- Distilled experience

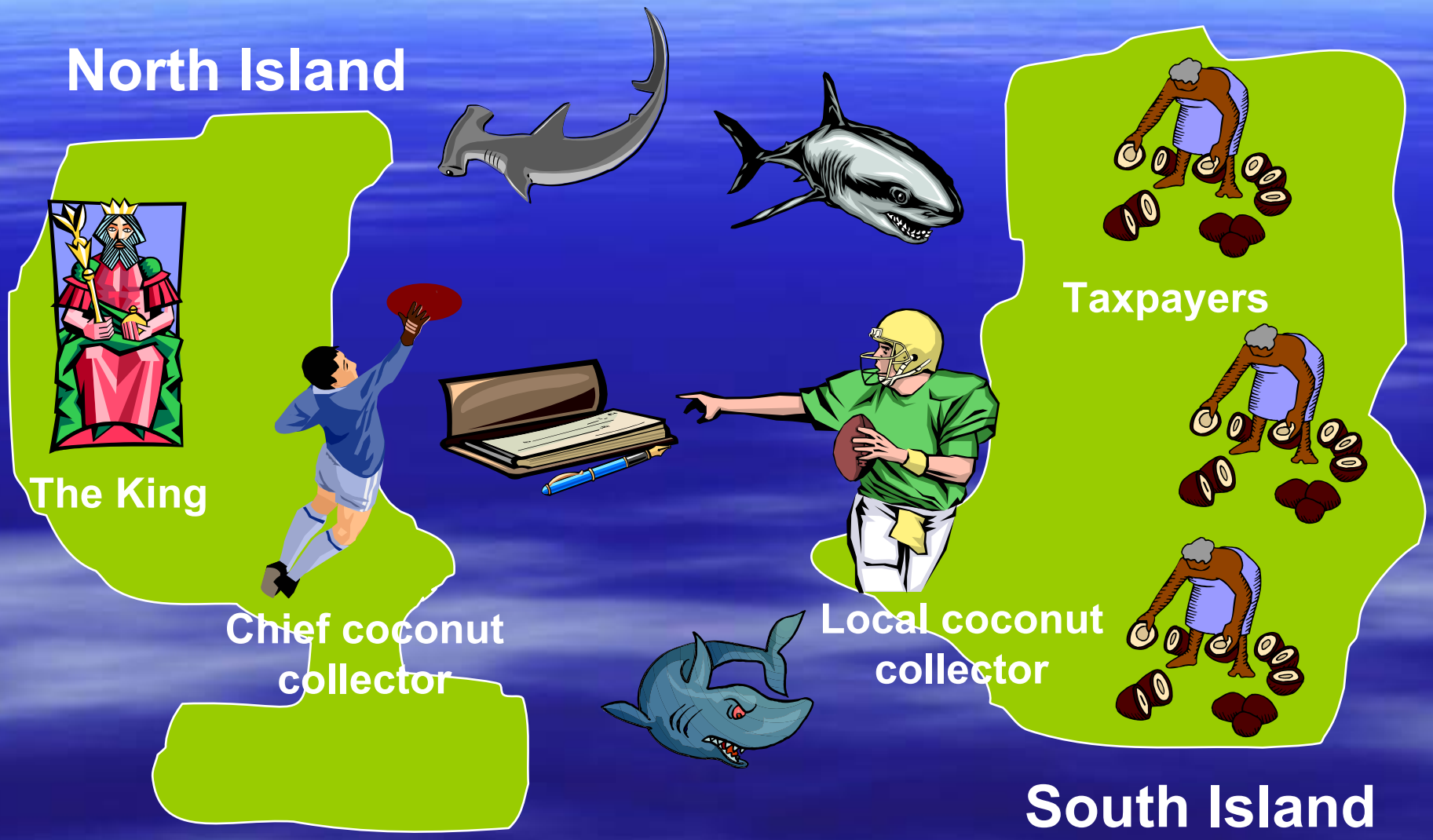
J2EE Patterns

- Somewhere between patterns and platform-specific idioms

Software design patterns for enterprise systems with guidance on how to implement them in J2EE

- Still guidelines and templates, not code
- Not a coherent pattern language due to differing sources
  - Sun Java Centre, Blueprints, theserverside, and books
  - Worth reading similar patterns
- Patterns change over time as J2EE platform changes
  - At least 3 early patterns are now directly implemented in J2EE

Collecting taxes can be tricky…



The related patterns

- Local coconut collector
  - J2EE Pattern: **Session Façade**
  - Hides pesky local details
- Chief coconut collector
  - J2EE Pattern: **Business Delegate**
  - Removes concerns about going off-island
- The cheque
  - J2EE Pattern: **Data Transfer Object**
  - Easier than passing a bunch of coconuts!

What's not a J2EE pattern?

- J2EE patterns are design guides for J2EE applications designers
- The following things do not count as J2EE patterns:
  - Most pattern implementations in the J2EE APIs
  - Anything dependent on a particular implementation
  - Frameworks

Agenda

- J2EE Patterns in Context
- Distribution and Web patterns
- Business and persistence patterns
- Pattern directions in J2EE

Distributed systems are different

"There are fundamental differences between the interactions of distributed objects and the interactions of non-distributed objects. Further, work in distributed object-oriented systems that is based on a model that ignores or denies these differences is doomed to failure, and could easily lead to an industry-wide rejection of the notion of distributed object-based systems."

A Note on Distributed Computing

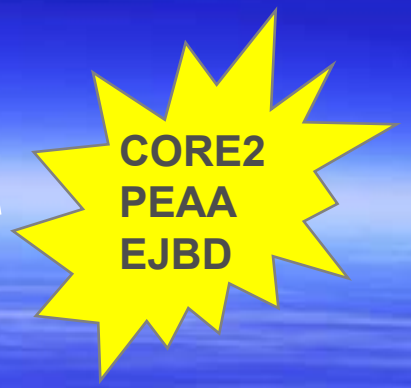
- The myth of transparent remoting
  - Implicit concurrency
  - Partial failure
  - Distributed failure modes
- Ignoring distribution is ignoring application context

Fowler's First Law of Distributed Computing

Distributed design

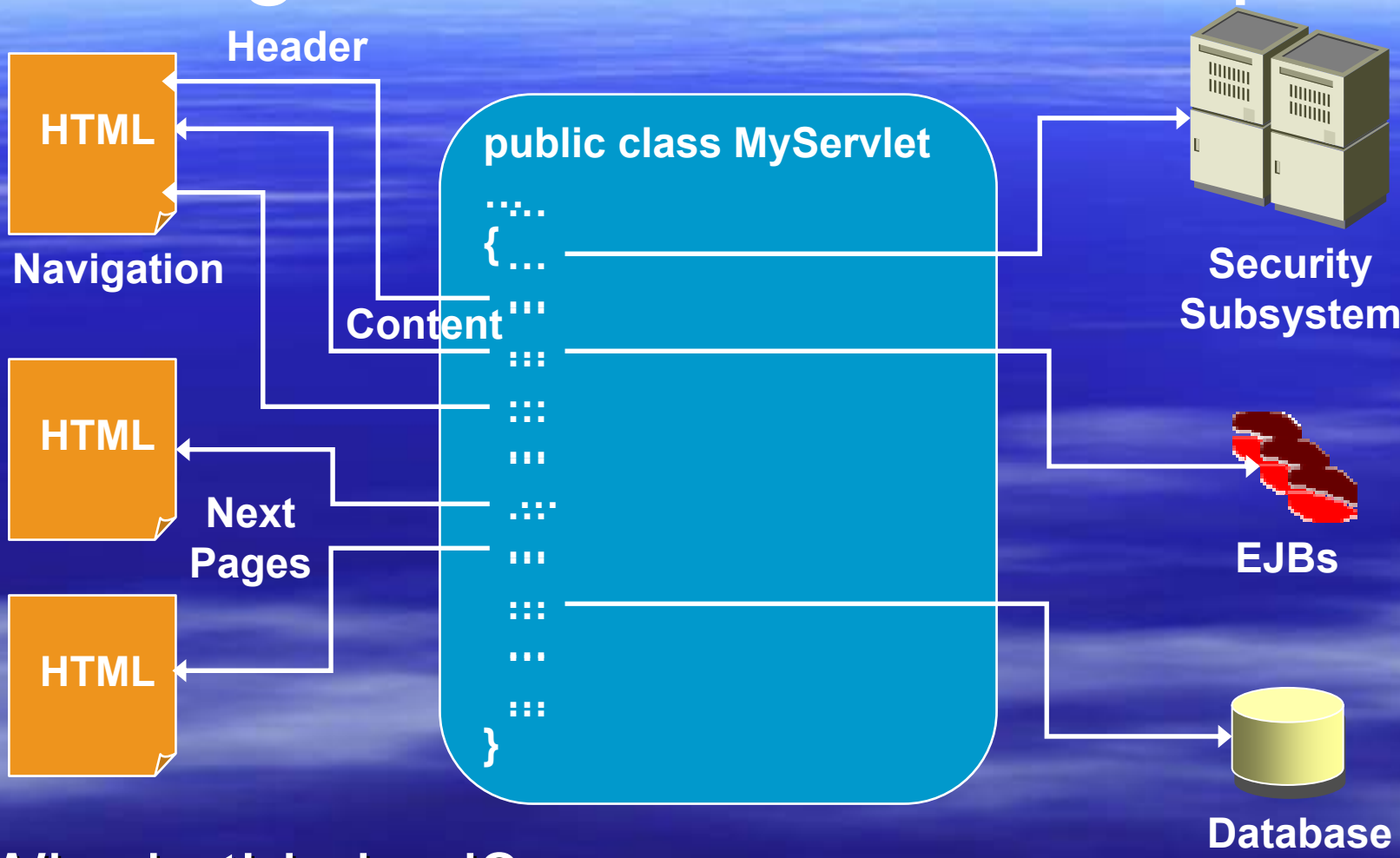
- Principles
  - Minimize number of calls
  - Reduce amount of data passed
- Strategies
  - Do it at the client: caching and sharing
  - Do it at the server: batching
- Try to avoid warping the domain model
  - Proxies and facades

Data Transfer Object



- Solution
  - Create a coarse-grained representation of data to be exchanged (DTO)
  - Pass DTO in single method call and access properties locally
  - Reduces number of method calls
- Variants
  - Exact replica of domain data (**Domain DTO**)
  - Tailored for the job (**Custom DTO**)
  - Generic map (**Data Transfer HashMap**)
  - XML document
- Benefited NFCs
  - Performance, efficiency, maintainability

Setting a bad Web UI example

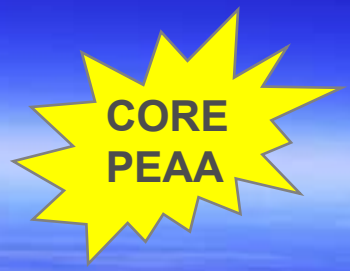


- Why is this bad?

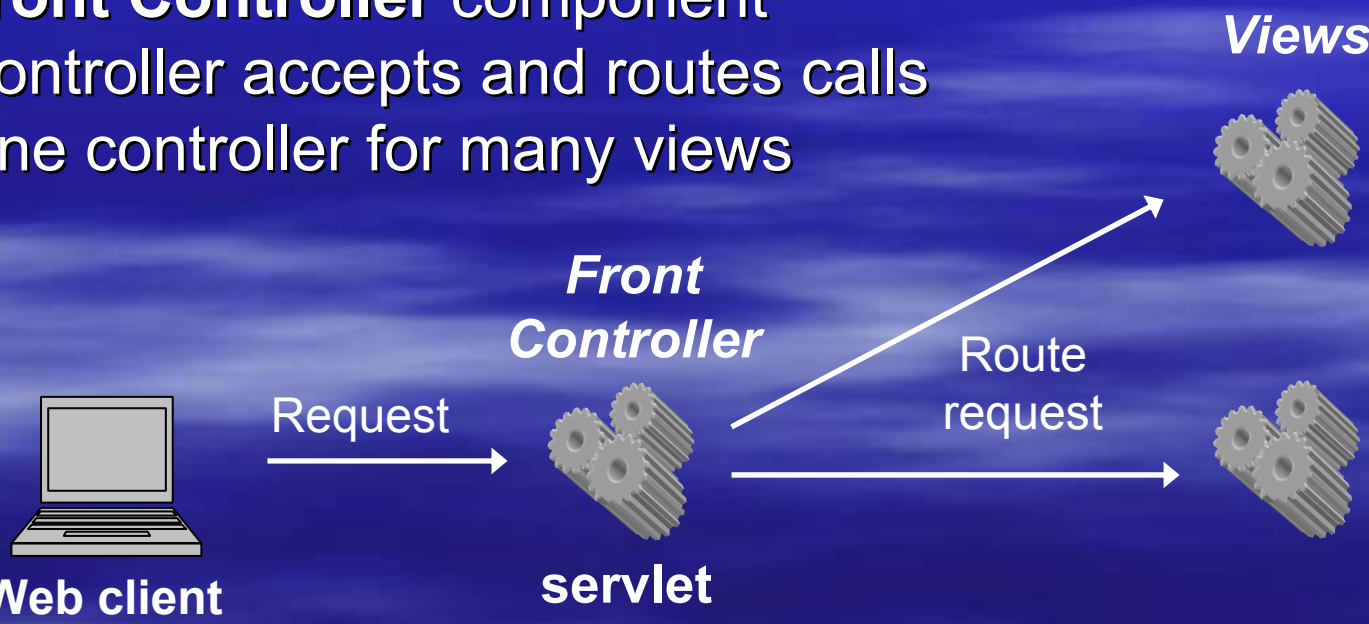
Non-Functional Characteristics

- Non-functional requirements on the system
  - Scalability, availability, performance, security, manageability, flexibility, internationalization, efficiency, maintainability/simplicity, portability
  - The system must display the appropriate non-functional characteristics (or systemic qualities)
  - A major concern for application architects
- Which qualities should be optimized for Web UI?
  - Maintainability
    - Lots of chance for repeated code
    - UI is particularly hard to test
  - Efficiency
  - Flexibility/evolvability (UI changes more rapidly)

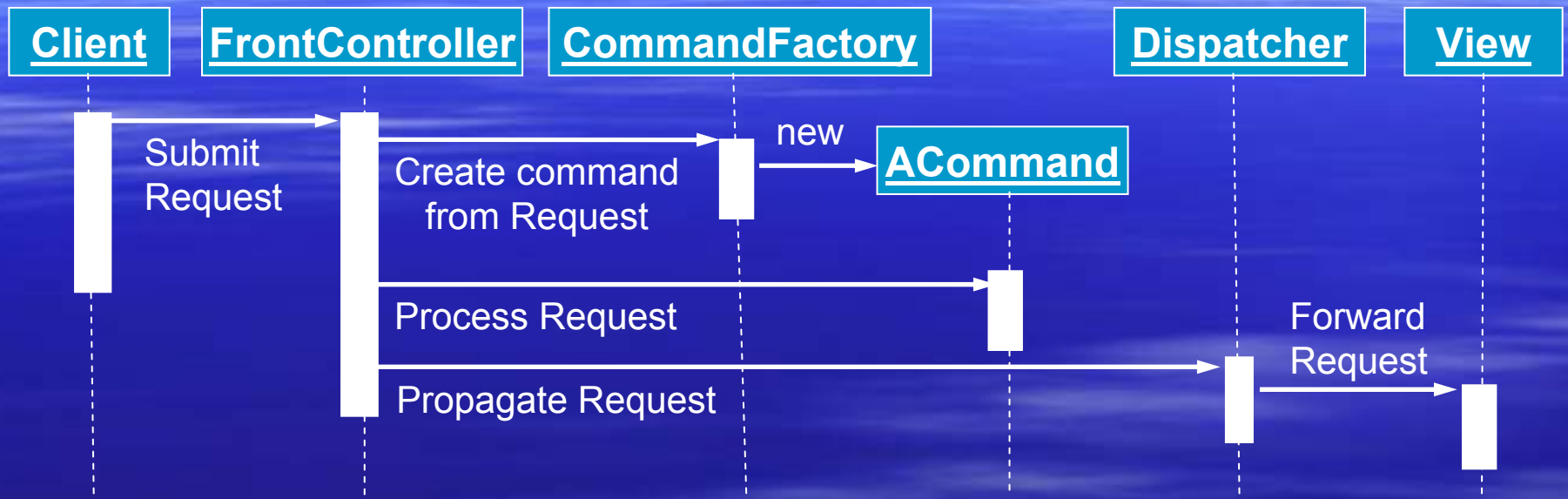
Front Controller



- Problem
  - Control logic meshed with display code
- Solution
  - Factor out decision making in **Front Controller** component
  - Controller accepts and routes calls
  - One controller for many views



Front Controller sequence

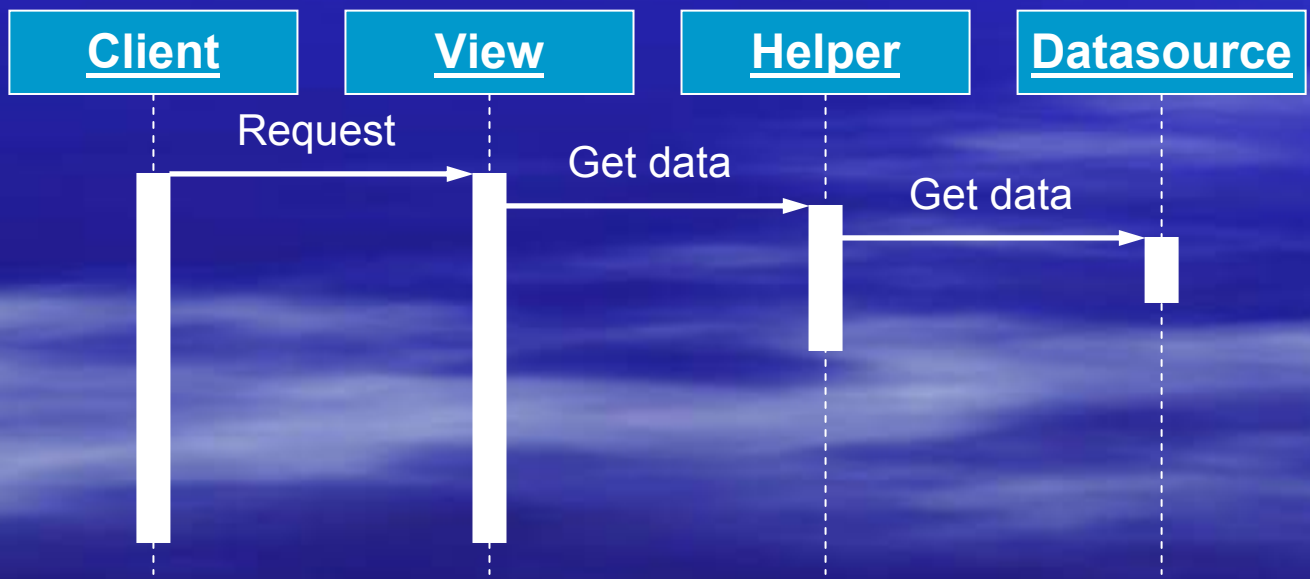


- Common to delegate to command and dispatcher
  - More flexible and simplifies controller implementation
  - Type of command based on request information
- Can use controller as hook for other functionality
- Controller and view strategies include servlet and JSP

View Helper

CORE

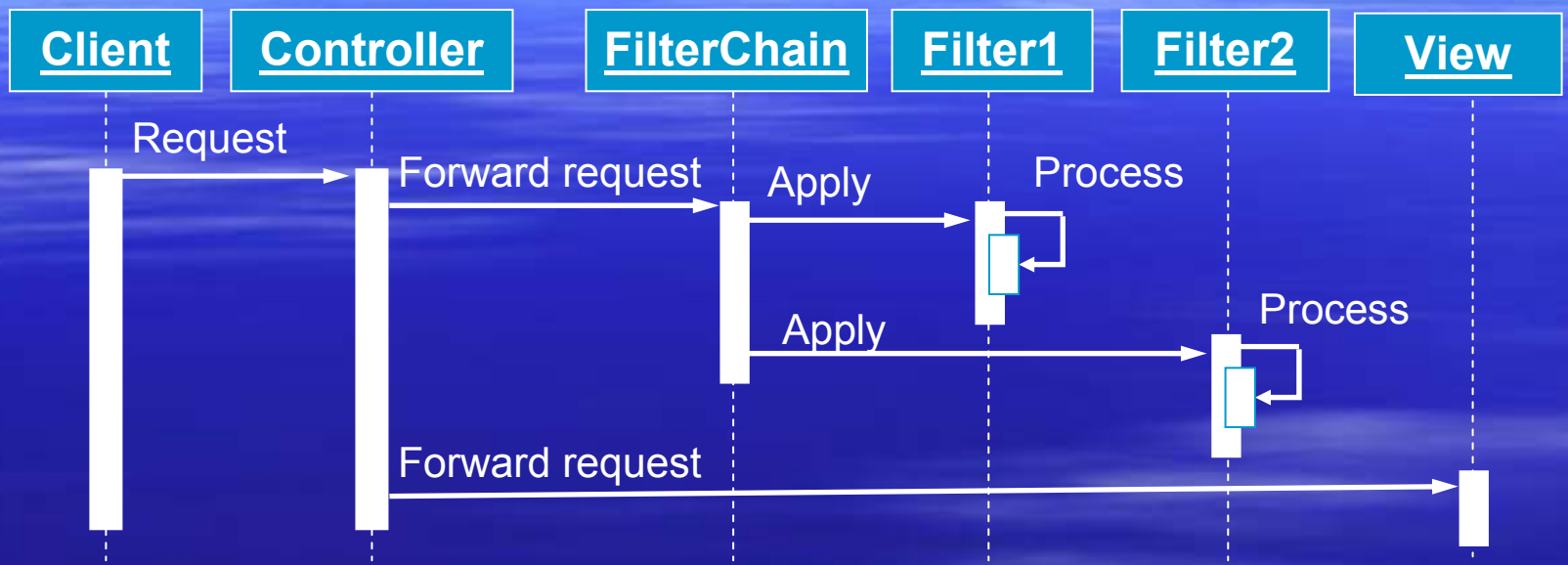
- Formalization of use of helper in a view
  - Choice of JSP tag, JavaBean or POJO
  - Provides data and/or functionality for view



- Service to Worker and Dispatcher View variants

Intercepting Filter

CORE



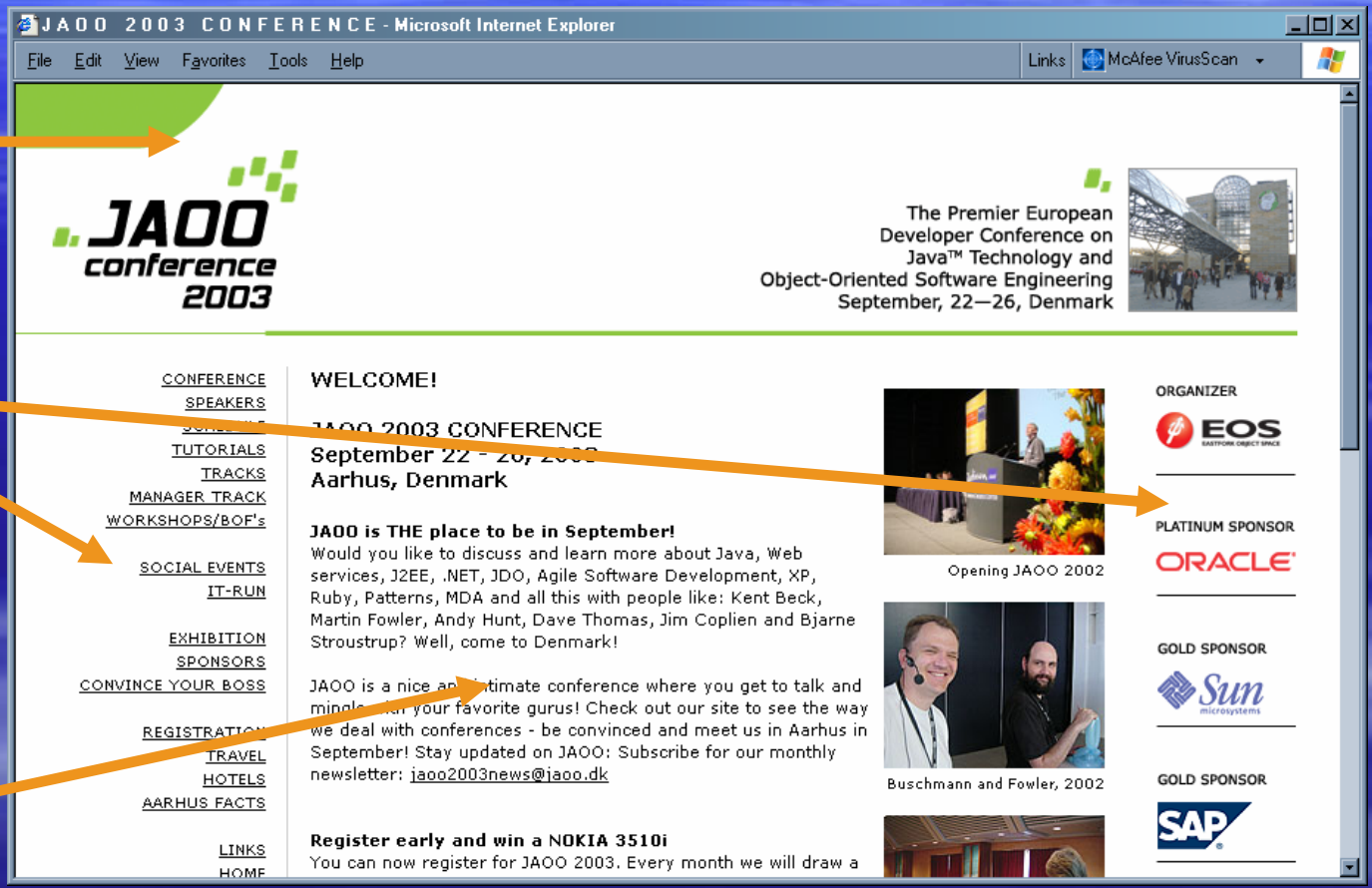
- Pluggable/configurable chain of filters
  - A form of **Pipes and Filters**
- Triggered through a controller
- Useful for pre-processing of request
  - Security, internationalization, etc.

Building the output

Header view

Navigation views

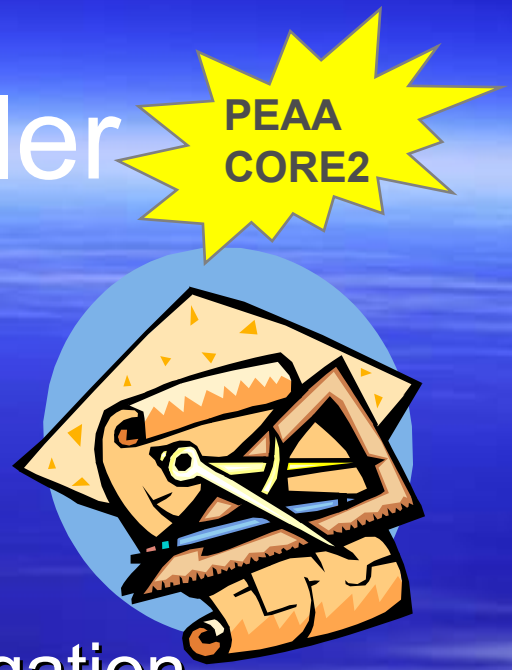
Main body view



Composite View

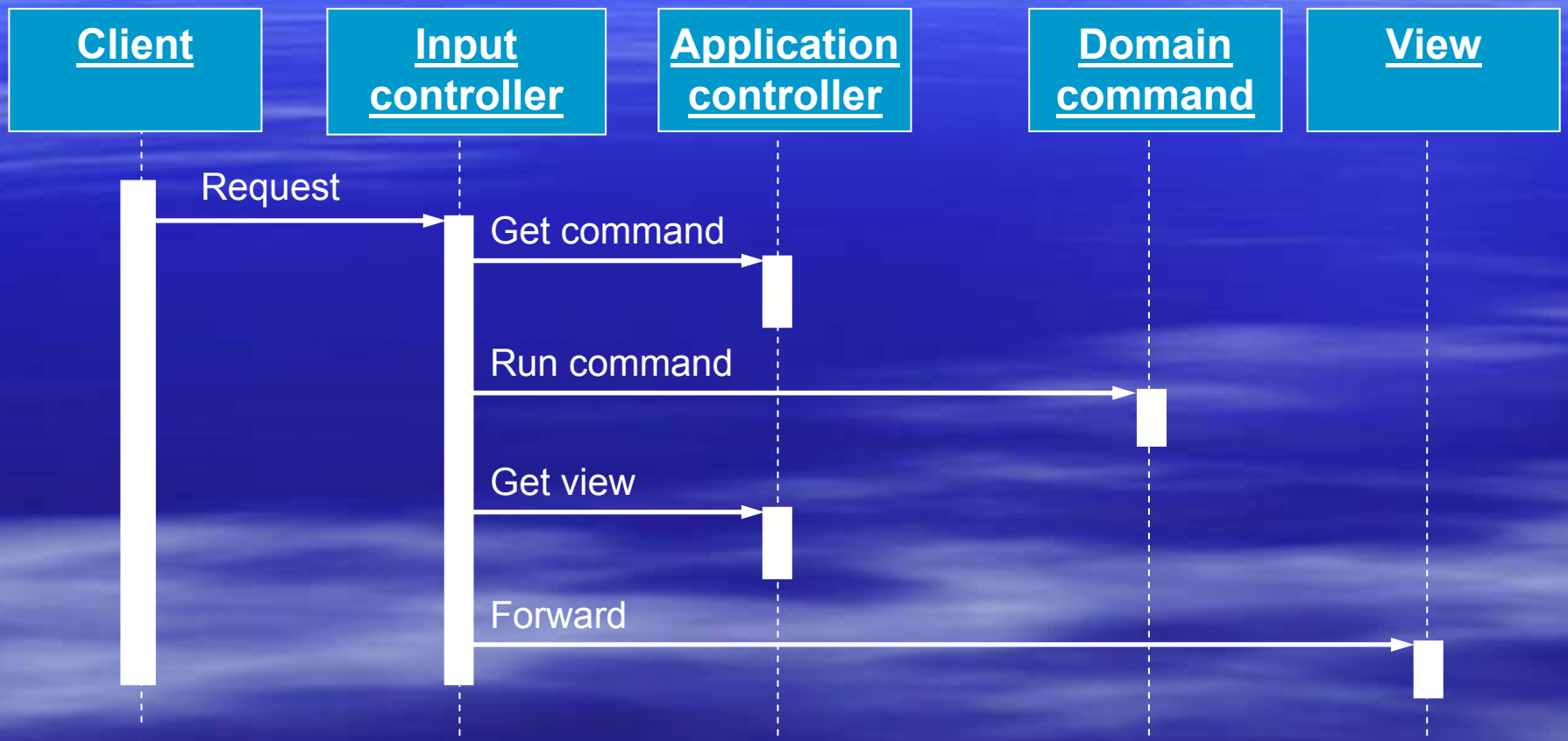
- Problem
  - Common look, feel and navigation calls for repeated functionality
  - Avoid that duplicate code smell again
- Solution
  - Compose view from multiple sub-views
  - Common way of building web content using any technology
  - Basic strategy uses <jsp:include>
  - More powerful inclusion strategies with JavaBeans or tags allow conditional behaviour
  - Translation time vs. runtime inclusion

Application Controller

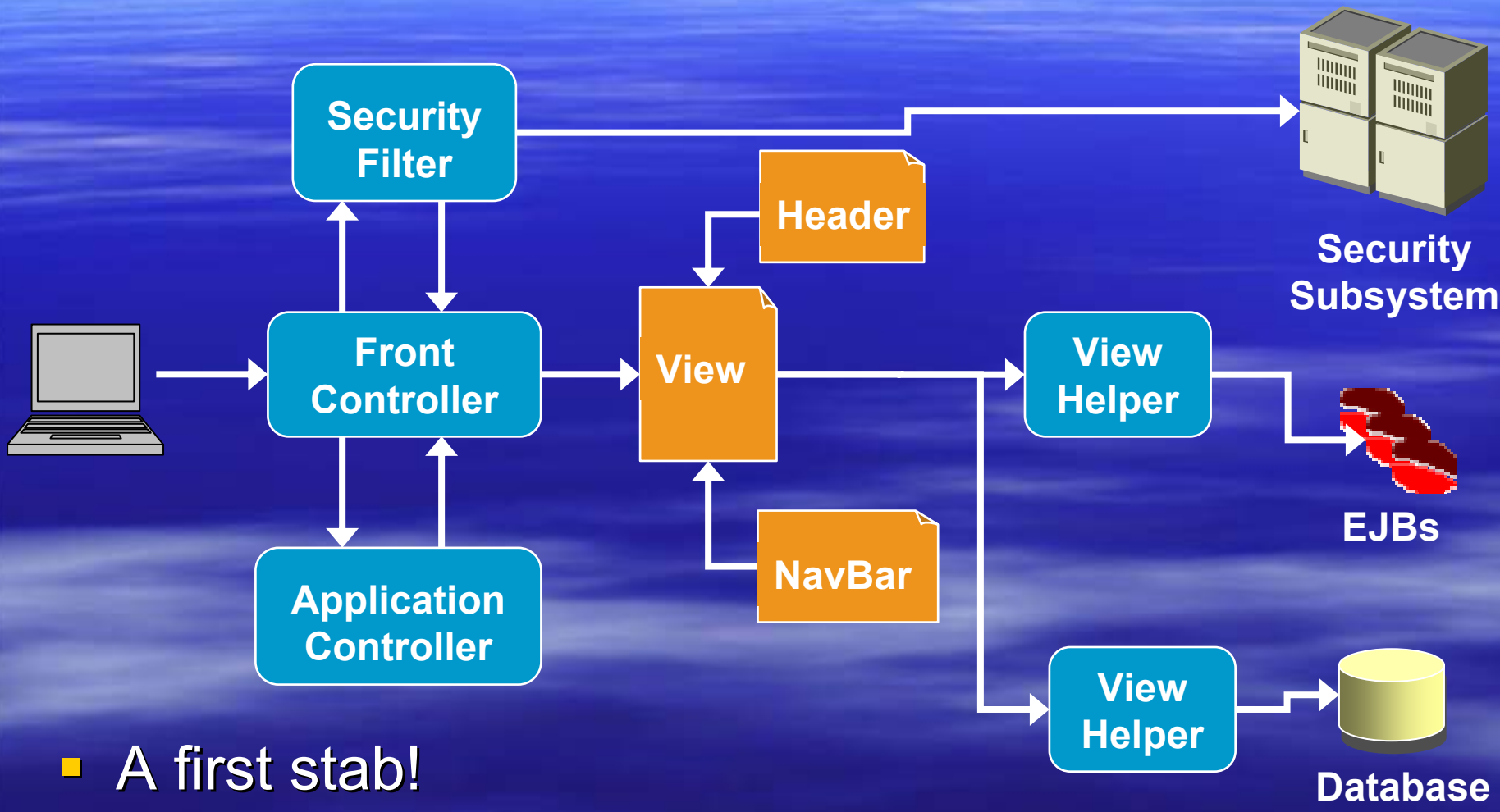


- Problem
  - Navigation control gets complex
- Solution
  - Create components to control navigation
  - These form the application layer
  - Input controller references **Application Controller** for application logic and navigation
  - Initialize from database or file (XML)
  - Keep **Application Controller** separate from UI

Application Controller sequence



Bad example refactored

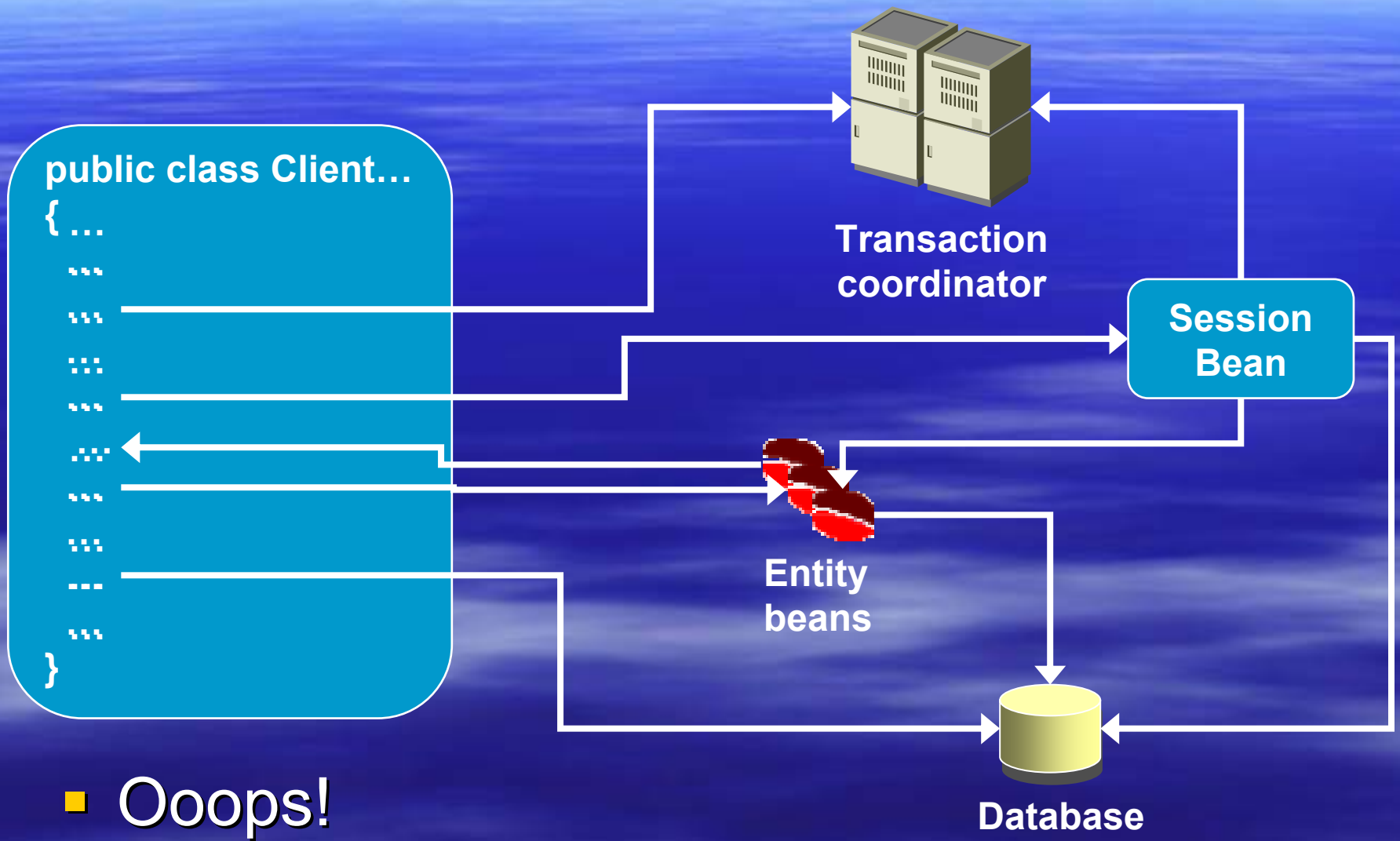


- A first stab!

Agenda

- J2EE Patterns in Context
- Distribution and Web patterns
- Business and persistence patterns
- Pattern directions in J2EE

Setting another bad example



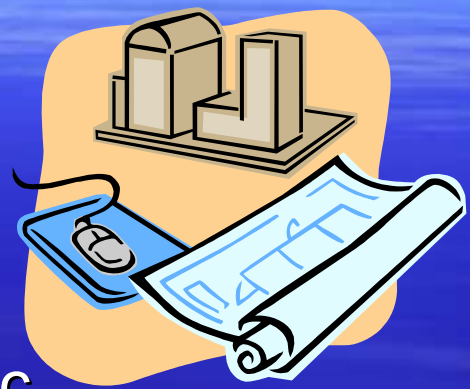
- Ooops!

Domain logic vs. business logic

- Need to split up business logic
  - How complex is it?
  - How is your data represented?
- Different approaches
  - **Transaction Script**
  - **Domain Model**
  - **Service Layer**
  - **Table Module**
- **Table Module** not common in J2EE

Domain Model

PEAA



- Problem
  - Complicated business logic
- Solution
  - Split out rules from other business logic
  - Encapsulate data with rules
  - Rules become domain logic, e.g. data validation
  - Simple domain model maps 1:1 with database which maps to J2EE entity beans
  - Richer domain model harder to map
  - Decouple from surrounding layers

CORE2 Business Object?

Layers and tiers revisited

- Don’t mix infrastructure with business logic
  - Can you take away the infrastructure and run domain logic layers in a single (servlet) container?
  - Layers are logic, tiers are infrastructure
- EJB RMI interface provides pseudo-tier
  - Separate the logic from the EJB with POJOs
- Still need architectural prototype
- Types and tiers
  - Use the same model types on the client side?
  - Depends on where your “application” starts and finishes

Why use EJBs?

- Why, precisely, do you need to use EJBs?
  - Thick client and sharing business logic
  - Reuse of EJB-based **Service Layer**
  - Declarative transactions and security for easier coding
  - “They deliver a high level of scalability”
  - Your tools push you that way
  - Everyone else is using them
- EJBs are a powerful framework
  - Need to understand the consequences
  - Need to ask some questions, such as
- Do your EJBs need remote interfaces?
  - Do you only need local interfaces?

Decoupling the layers

- Logical decoupling
  - Vary layers independently using facades
  - **Façade** amalgamates underlying functionality
- Distribution decoupling
  - Tiers or pseudo-tiers
  - Hiding distribution, e.g. **Service Locator**
  - Asynchronous interaction
  - EJB facades
  - Client-side delegates
  - Command-based services
- Interface of **Façade** differs from business objects
  - Operation signatures, Data types, Errors

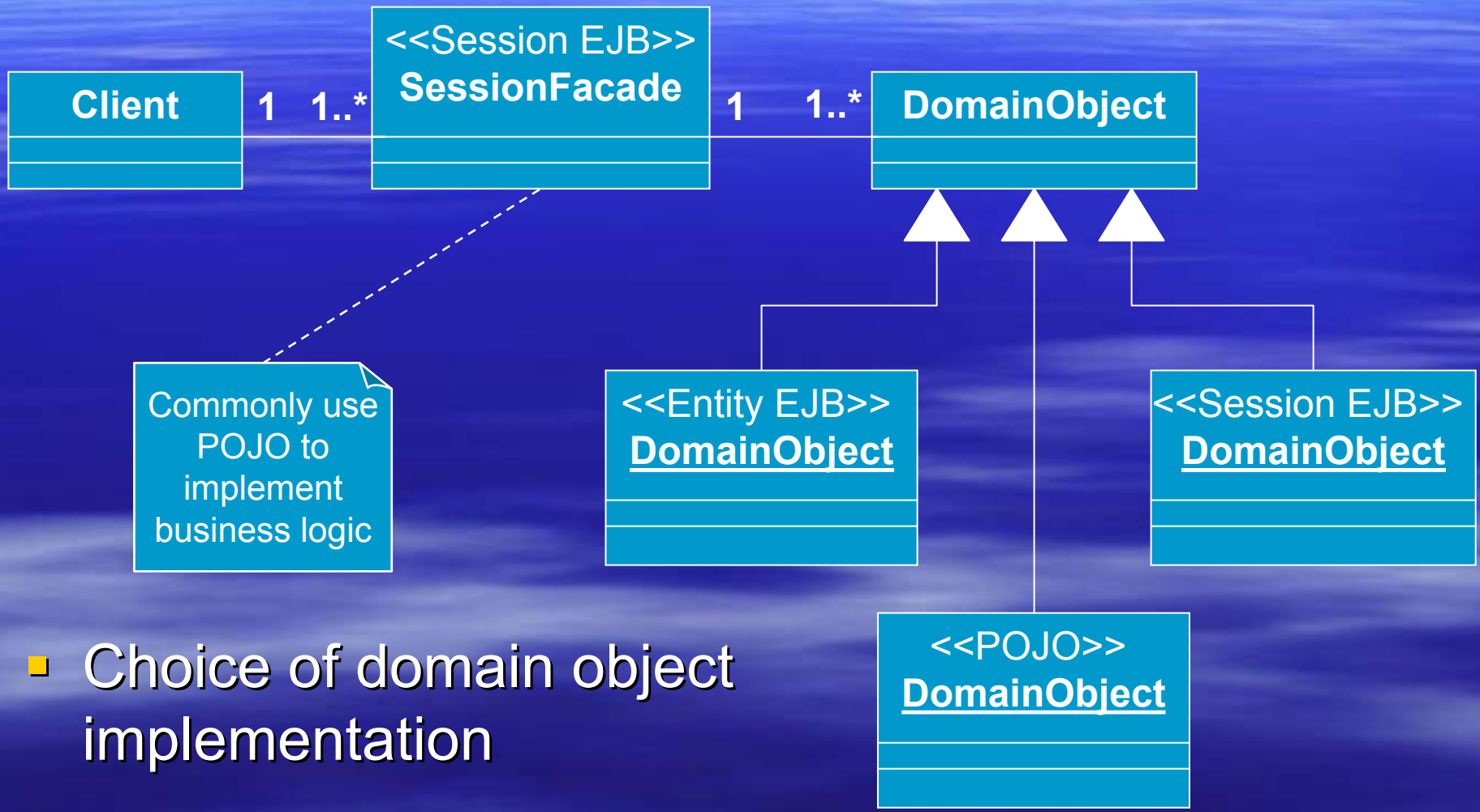
Remote Façade

- Solution
  - Create a server-side **Façade** offering a more distribution-friendly interface for business logic
  - Contains no domain or application logic
  - Domain and application classes remain unchanged
- **Remote Façade** will
  - Convert between DTOs and domain data
  - Handle exceptions crossing the distribution boundary
  - Perform other duties such as security checking
- Benefited NFCs
  - Maintainability, flexibility

Session Façade

- Problem
  - Pseudo-tier boundary with entity **Domain Model**
  - Where does business logic go?
- Solution
  - Use session bean as a **Remote Facade**
  - Can combine logical POJO with **Session Façade**
  - Associate with POJO using inheritance or delegation
  - Session bean is (part of) your component interface
  - **Layer Supertype** for **Session Facades**

Session Façade classes



- Choice of domain object implementation

Distribution, EJBs and transactions

- Big principles of distributed computing #4:

  Avoid distributed transactions wherever possible

- Just because you can do it doesn't mean that you should…
- Keep system transactions within a tier
- Start system transactions in the same layer
- **Session Façade** is a good place to demarcate system transactions
  - EJB declarative transactions
  - System transactions parallel **Transaction Scripts**
- Combine system transactions at the **Session Façade** level

Service Activator

- Problem
  - Messaging can provide many NFC benefits
    - Performance, availability, scalability
  - Most business logic organized in non-message forms
- Solution
  - Provide an Adaptor between messaging and business logic
  - Invoke business functionality in response to asynchronous message
  - Strategy for EJB 2.0 is MDB
  - Can use JMS application to trigger non-MDBs, pre-EJB 2.0 beans, or POJOs
- Complexity comes from
  - Coordination when responses are required
  - Handling errors

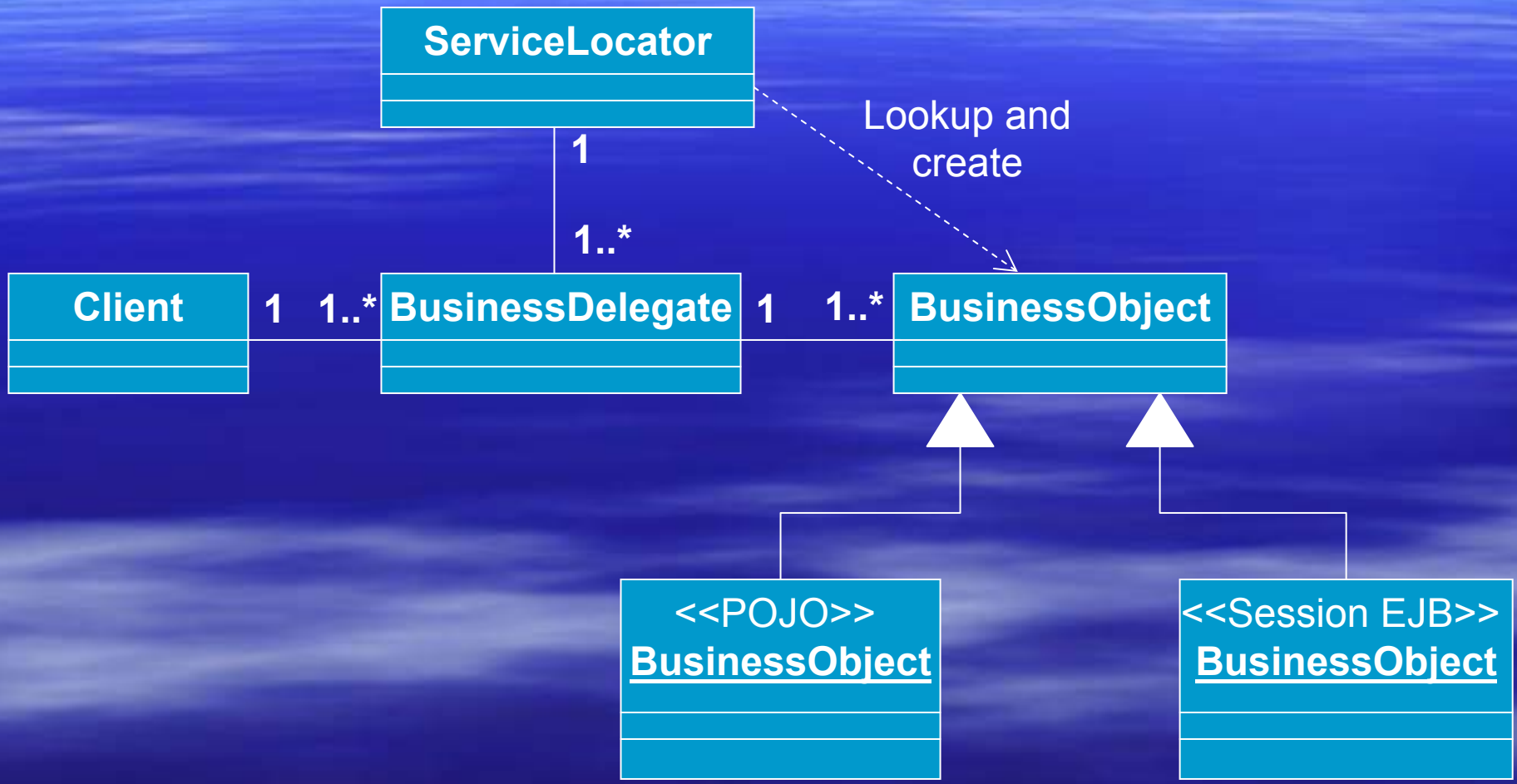
Message Façade

- Problem
  - Message client needs to invoke multiple business functions in a single operation
  - Desired asynchronous functionality is on entity bean
- Solution
  - Wrap desired asynchronous functionality in a stateless **Session Facade**
  - Expose session bean as an MDB
  - Single **Transaction Script** per-bean
  - Façade layered on **Service Activator**
  - Asynchronous adaptor for non-stateless session beans
- Same cons as **Service Activator**

Business Delegate

- Problem
  - Business interface may not suit client
  - Clients exposed to distribution mechanism
- Solution
  - Use a smart proxy to reduce coupling
  - Hides distribution issues, e.g. RemoteException
  - Combine with **Service Locator**
  - Common pairing with **Session Facade**
  - Beware of lifecycle issues such as servlet serialization

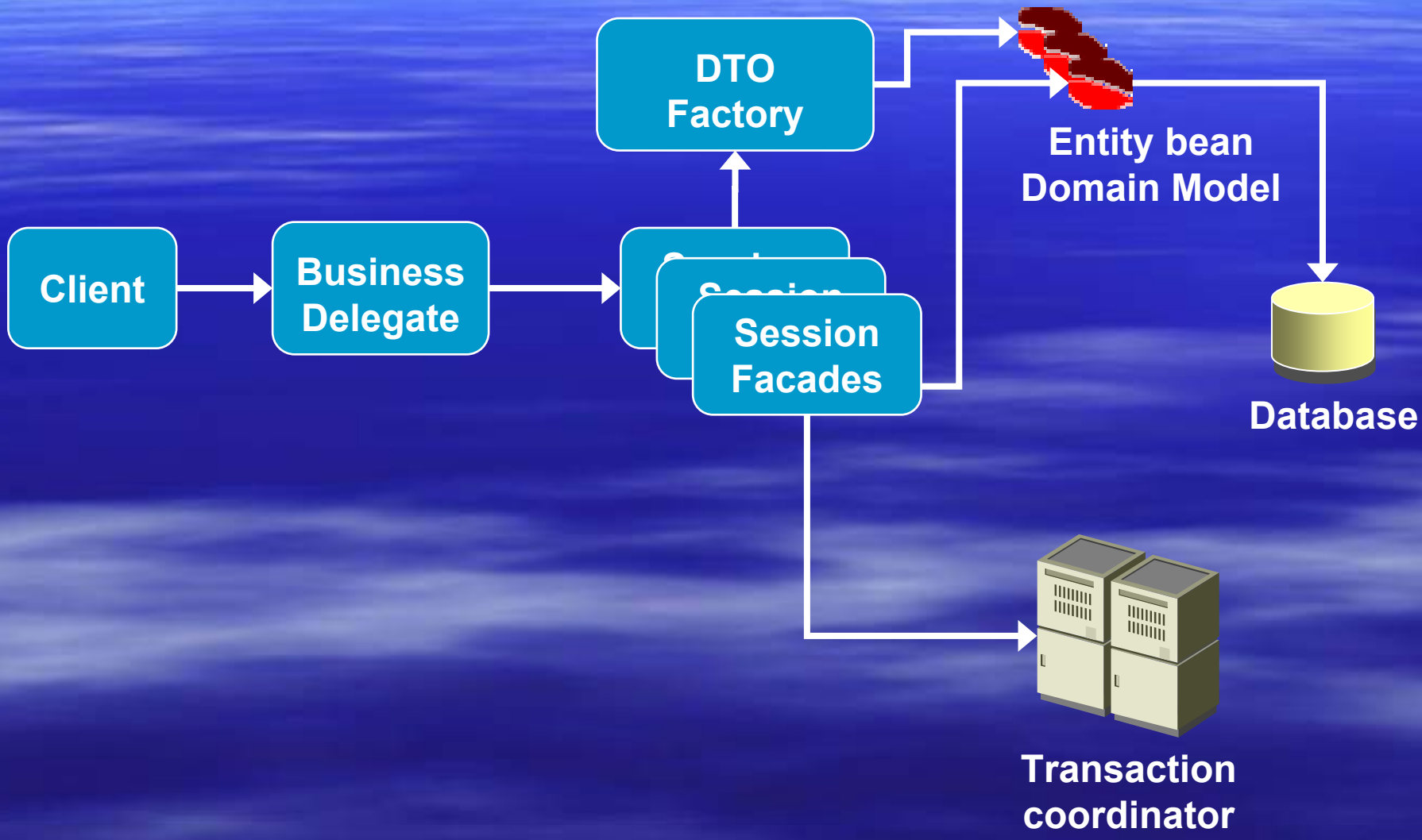
Business Delegate classes



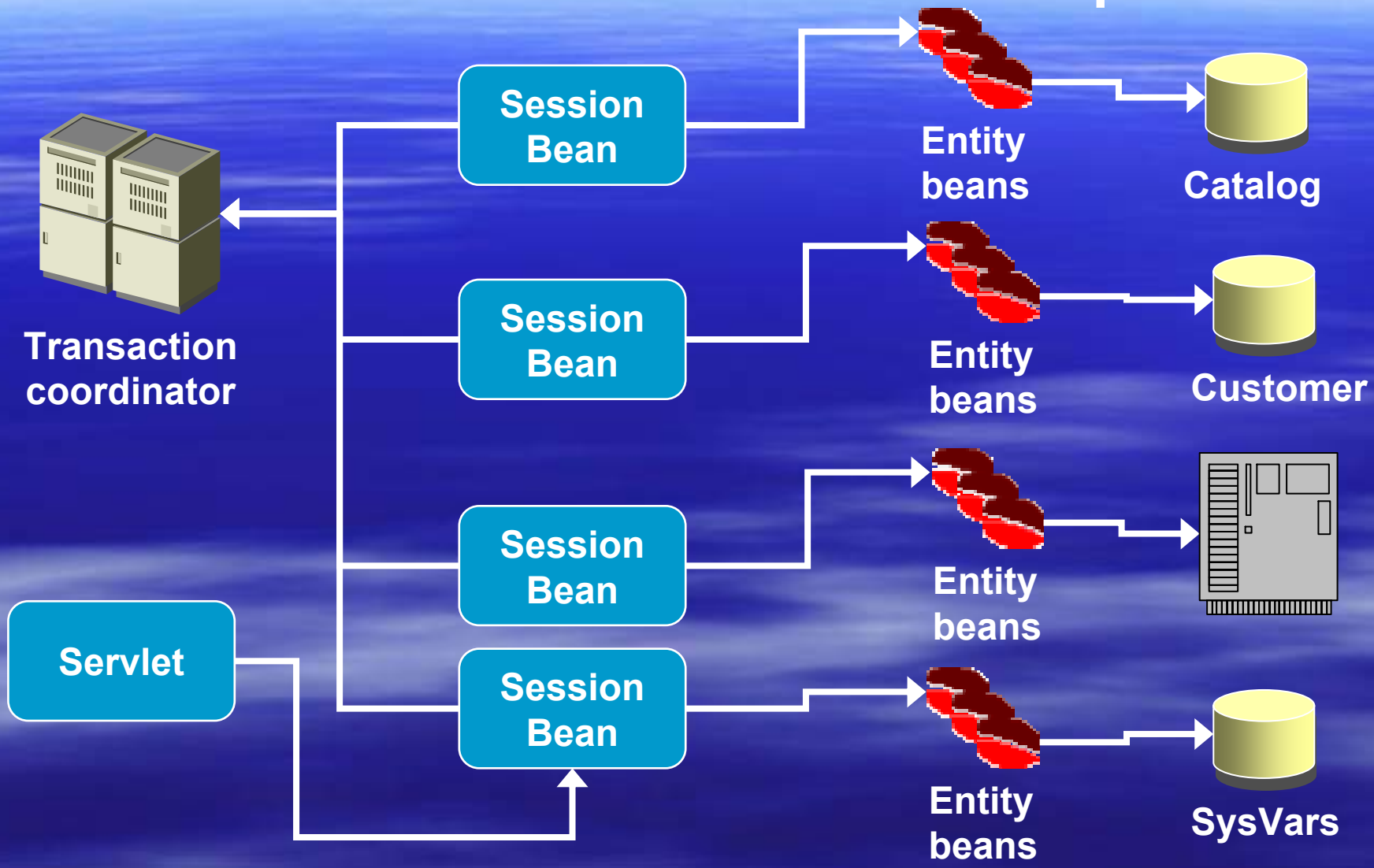
Data Transfer Object Factory

- Problem
  - Reduce dependencies between **Domain Model**, **Session Facades** and **DTO**s
  - Reduce duplicated code ‘smell’
- Solution
  - Use a factory to create and consume the **DTO**s
  - **Session Façade** delegates to factory
  - Single point of change for conversion logic
  - Applies to entity beans or any other Domain Object type
  - Can implement as POJO or session bean
  - Same as CORE **VO/DTO Assembler**

Bad example refactored



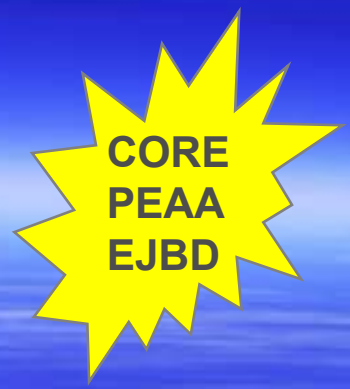
Yet another bad example



What are we trying to do here?

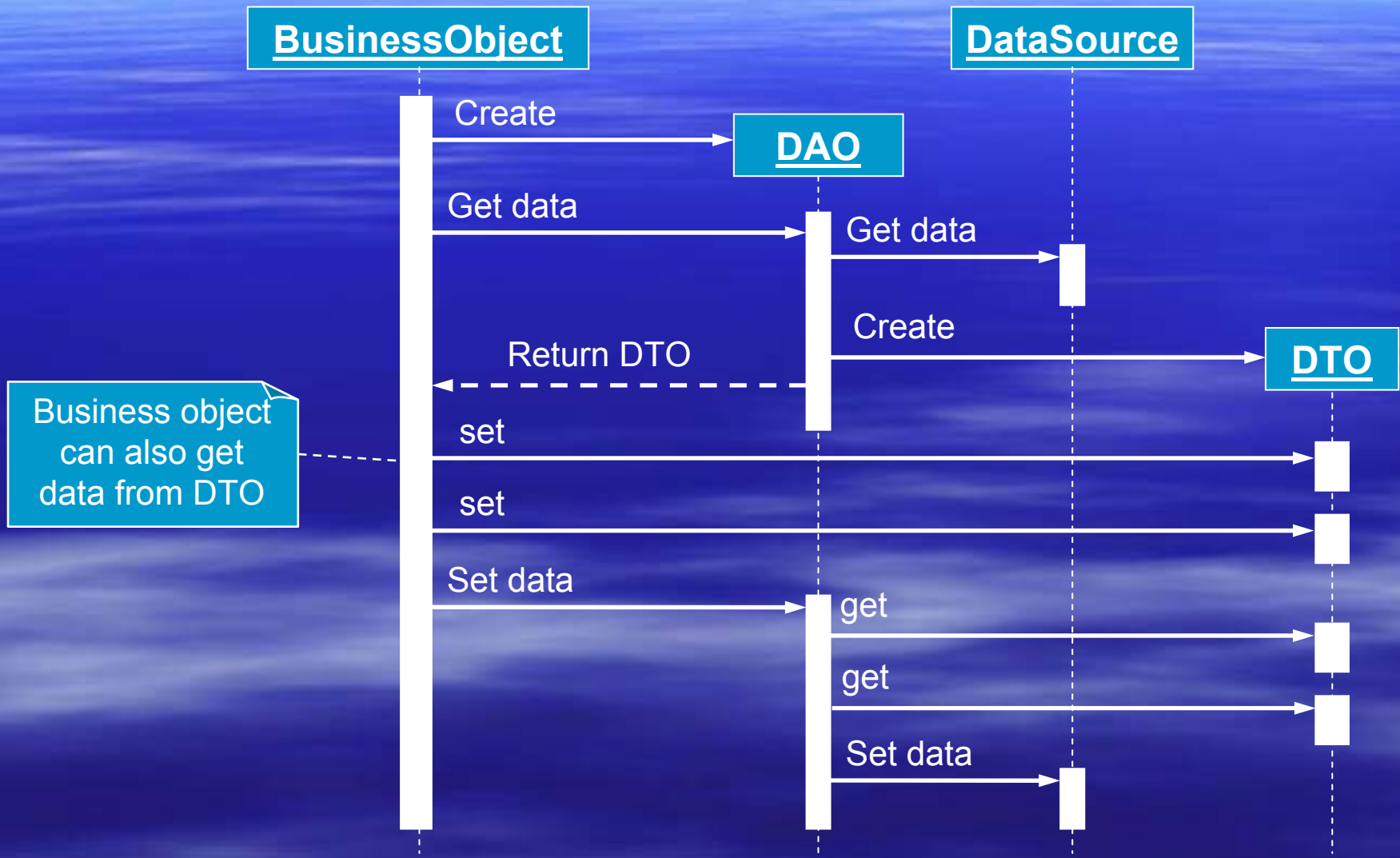
- Data held in many places outside J2EE container
  - Databases, mainframes, XML, etc.
- **Gateway** pattern for access to external data from an OO language
- Consider this in two parts
  - Data from relation databases
  - Data and functionality from elsewhere
- J2EE data representation options
  - Entity bean shown in many places
  - Can equally use POJOs and JDBC

Data Access Object



- Problem
  - Need to decouple data access from data source
- Solution
  - Create **Data Access Object** (**DAO**) to encapsulate and abstract data access (CRUDL)
  - Return **Data Transfer Object**s and collections
  - Decouples business code from database
  - Pass **DTO**s back in to update
  - **DAO** manages connection
  - Can use as part of EJB BMP, or…
  - …CMP entity EJB implementation can be your DAO
  - Can apply a **Factory** for **DAO** creation
  - Aka **Table Data Gateway** in PEAA

DAO sequence

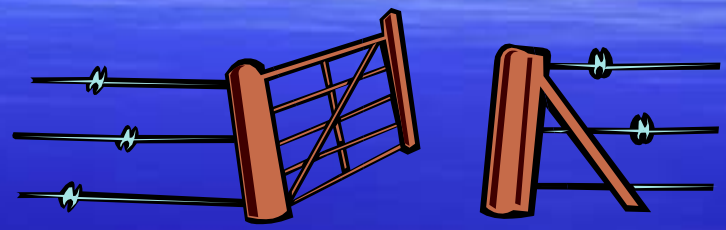


‘Do it in the database’

- A sliding scale
  - Do it all in Java
  - Do it all in stored procedures
  - Performance vs. portability/flexibility
- Best to strike a balance
  - Beware lack of data separation
  - Do your developers all write good SQL?

Row Data Gateway

- Problem
  - Need persistence for object model
- Solution
  - Create an object representing a row in the database
  - Add code to save, load and insert itself into the database
  - Strongly coupled to database schema
  - Use a finder class to get hold of instances
  - Very much in the style of an entity bean with no domain logic
  - Can implement as an entity or as a POJO
  - Use with **Transaction Scripts** (like entity and **Session Façade**)…
  - … or **Domain Model** (delegate database access)
  - If duplicate code smell appears migrate to **Active Record**



Active Record

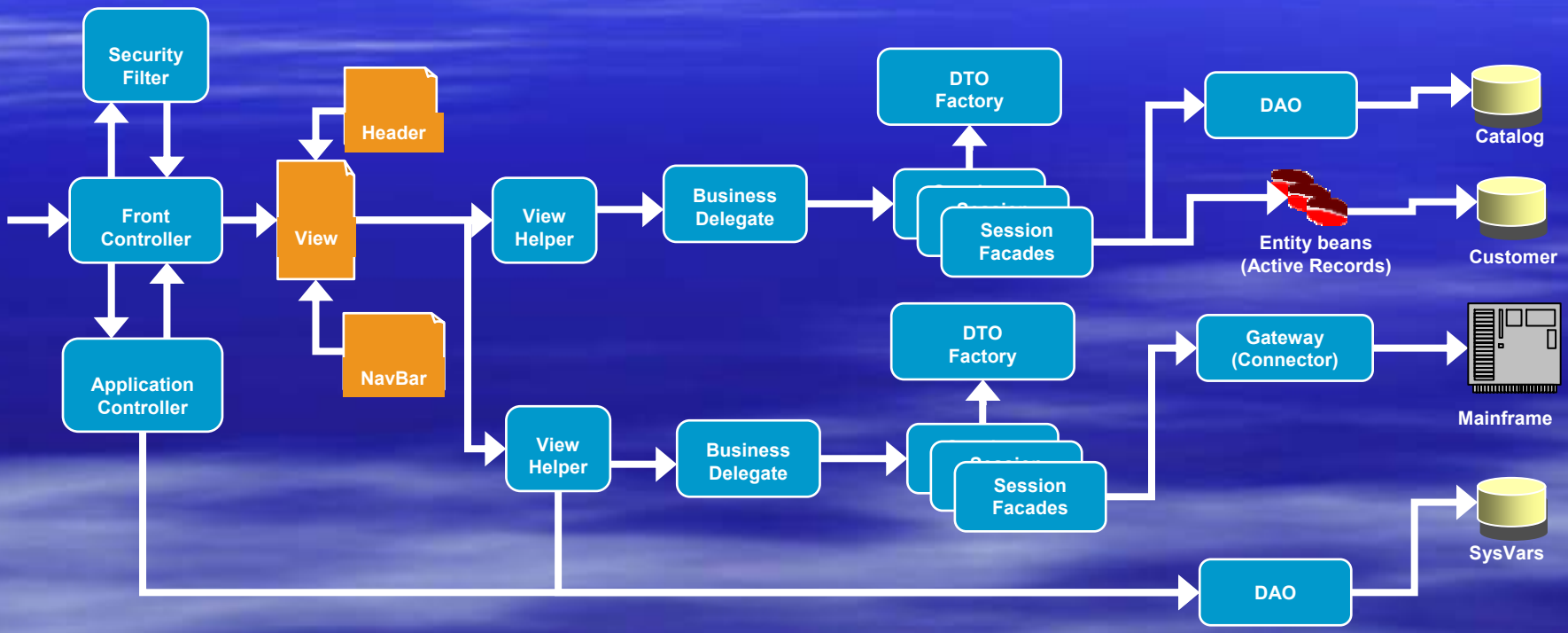
- Problem
  - Need persistence for object model with associated domain logic
- Solution
  - Like a **Row Data Gateway** with added domain logic
  - Strongly coupled to database schema
  - Use a finder class or static method to get hold of instances
  - Maps straight onto an entity bean
  - Can also implement as a POJO
  - Use with **Transaction Scripts** (like entity and **Session Façade**)…
  - … or as a **Domain Model**
  - Good if domain model matches data and domain logic is relatively simple
  - If domain logic is not simple, try a **Data Mapper**

Data Mapper

PEAA

- Problem
  - Provide persistence for divergent **Domain Model** and data source
- Solution
  - Create a **Data Mapper** to encapsulate access to external data and create or persist objects based on this
  - Provides persistence for a **Domain Model**
  - Allows divergent **Domain Model** and database schema
  - Better than other patterns for handling complex relationships between **Domain Model** and database data
  - On the way to full O-R mapping
  - Optimize the **Data Mapper** with caching and lazy reads/writes

Our refactored application



Agenda

- J2EE Patterns in Context
- Distribution and Web patterns
- Business and persistence patterns
- Pattern directions in J2EE

What else can we learn from?

- Other types of application
- Pre-provided applications
- J2EE Blueprints
  - Code + running applications
  - Books
  - Web site
  - FAQs

A balanced view

- The .NET Pet Shop

¼ of the code and 10 x faster

"Performance, performance, performance, that's all you ever think about"

With apologies to Monty Python

- .NET version of the Blueprints application
  - Discussion and examples at gotdotnet.com
  - Comparison and flame war at The Middleware Company
  - Rickard Oberg's deconstruction of TBC's benchmark
- What are the differences?
  - Implementation platform
  - Mindset about design
  - Different dogmas (Entity beans vs. DataSets)
- What can we learn from this?

Effect of Web Services on patterns

- Service-oriented patterns are favoured
- State management changes
- Data transfer is important
- Loose coupling favours messaging patterns
- Exceptions and errors are different

Where next?

- Lots of material in References slides
  - Read things that look interesting
  - Don't just read J2EE material – design crosses technology boundaries
- J2EEPATTERNS-INTEREST email list
  - Bit noisy now but good discussion in the archive
- The ServerSide
  - Mixed level of debate
- Find good people to talk to

In conclusion

- Designing systems is not easy
- Many forces to consider
- Forces change as platforms and technologies evolve
- Patterns are a useful tool
- You never stop learning about design

References 1

- Core J2EE Patterns (CORE)
  - Alur, Crupi, Malks; Prentice Hall
- Core J2EE Patterns, 2nd Edition (CORE2)
  - Alur, Crupi, Malks; Prentice Hall
- Patterns of Enterprise Application Architecture (PEAA)
  - Fowler; Addison Wesley
- EJB Design Patterns (EJBD)
  - Marinescu; Wiley
- J2EE Blueprints Patterns Catalog
  - http://java.sun.com/blueprints/patterns/catalog.html
- The Server Side
  - http://www.theserverside.com

References 2

- J2EE Design and Development
  - Johnson; Wrox
- Designing Enterprise Applications with the Java 2 Platform Enterprise Edition
  - Kassem at al.; Addison Wesley
- Pattern-Oriented Software Architecture
  - Buschmann, Meunier, Rohnert, Sommerlad, Stal; Wiley
- Pattern-Oriented Software Architecture – Volume 2, Patterns for Concurrent and Networked Objects
  - Schmidt, Stal, Rohnert, Buschmann; Wiley
- UML Components
  - Cheeseman, Daniels; Addison-Wesley