

Component Architecture for the Internet

Barcelona, July 20th, 2001



Contents

Part 1: Motivation

- Efficient Development
- Specific against General
- Library Weight

Part 2: Theory

- Concepts & Philosophies
- Inheritance & Instantiation
- Aggregation, Forwarding & Delegation
- Interface Syntax: Signatures
- Interface Semantics: Contracts
- Patterns
- Frameworks
- Components

Part 3: Case Studies

- Java Beans
- Corba
- Visual Basic and COM/DCOM
- Competitiveness Marketplace
- Unix Tools

Part 4: Exercises

- Exercises
- Component Library Guidelines

Part 1: Motivation

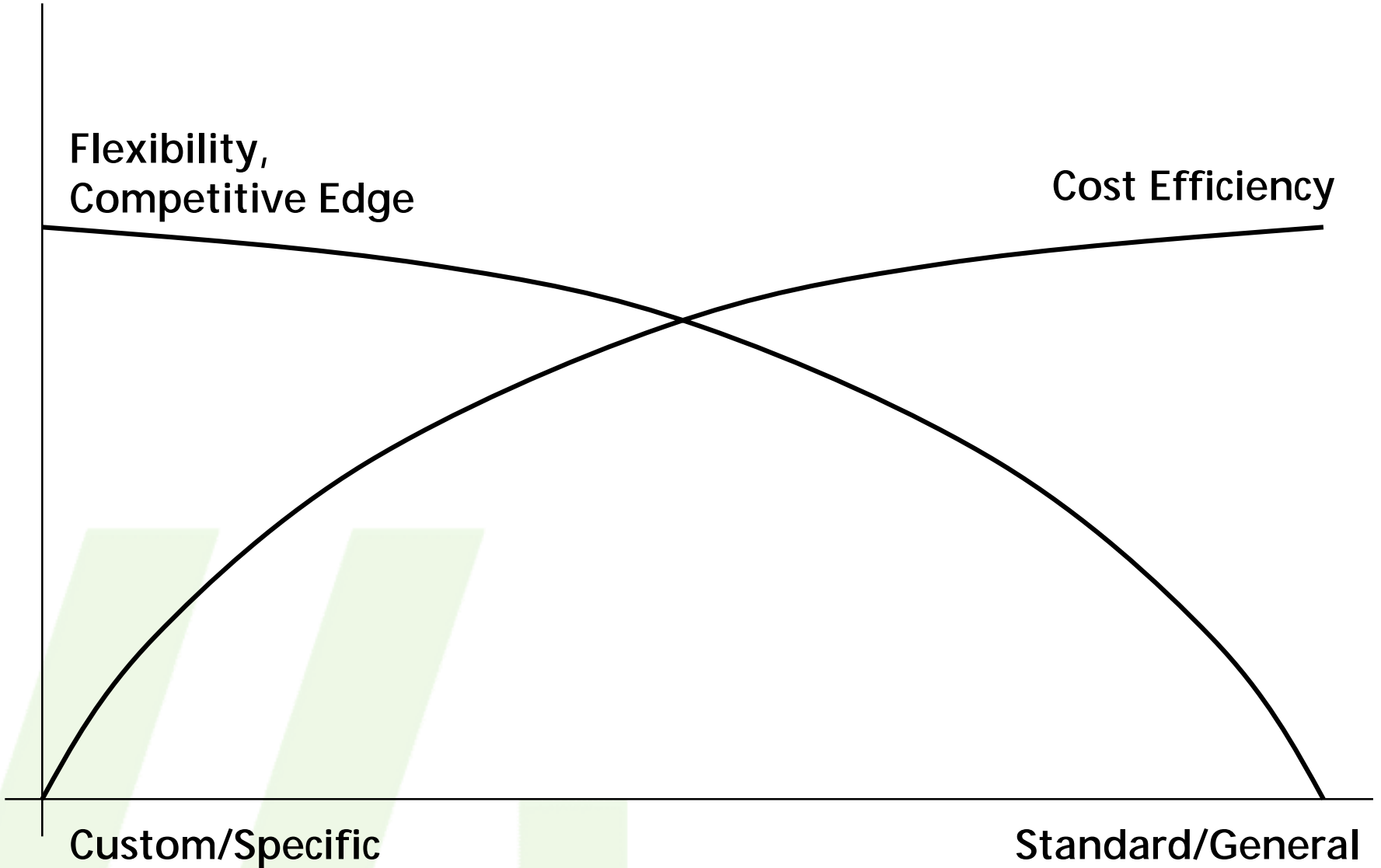


Efficient Development

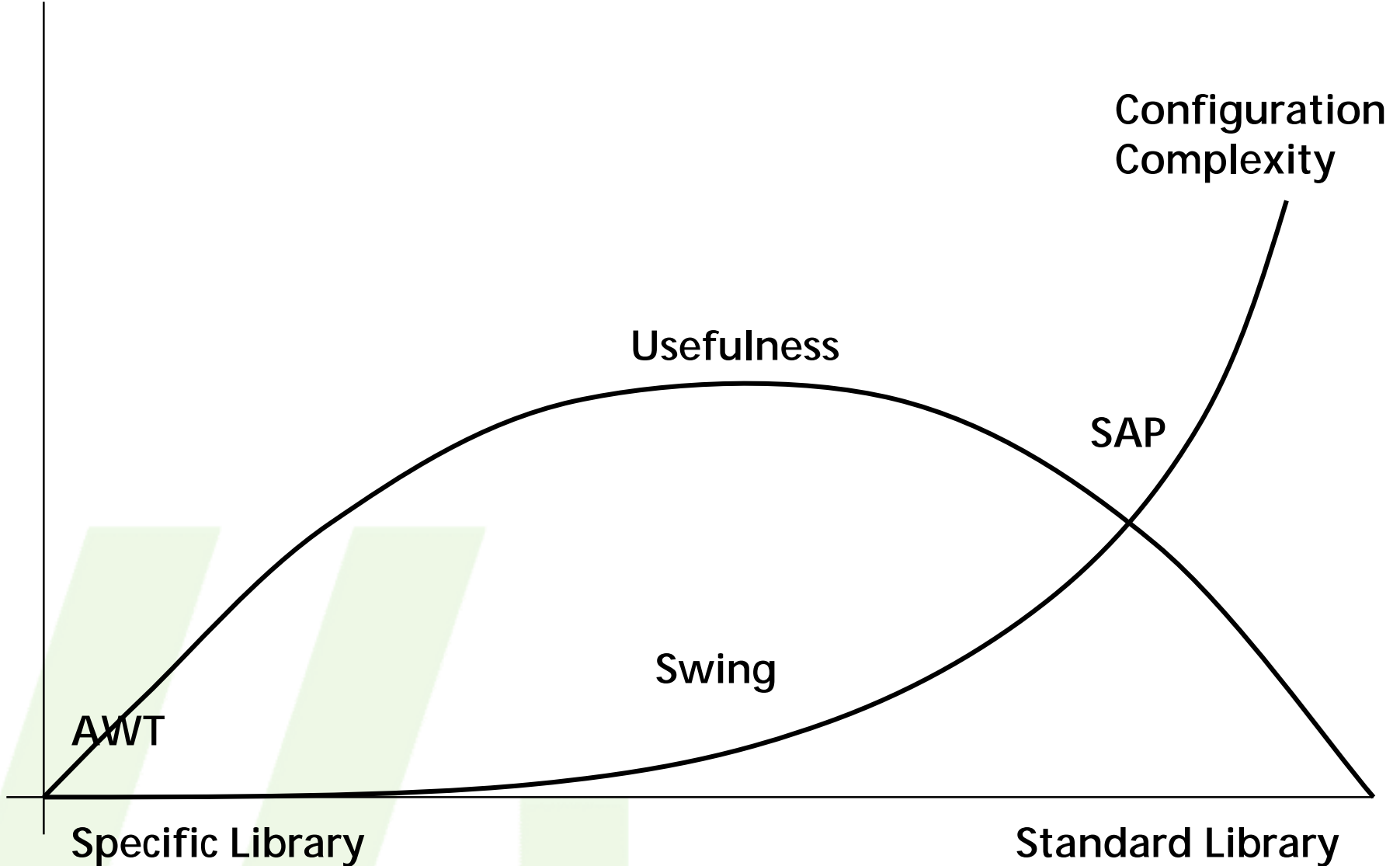
$$\text{Time} = \frac{\text{Functionality} - \text{Library}}{\text{People} * \text{Efficiency}}$$



Specific against General



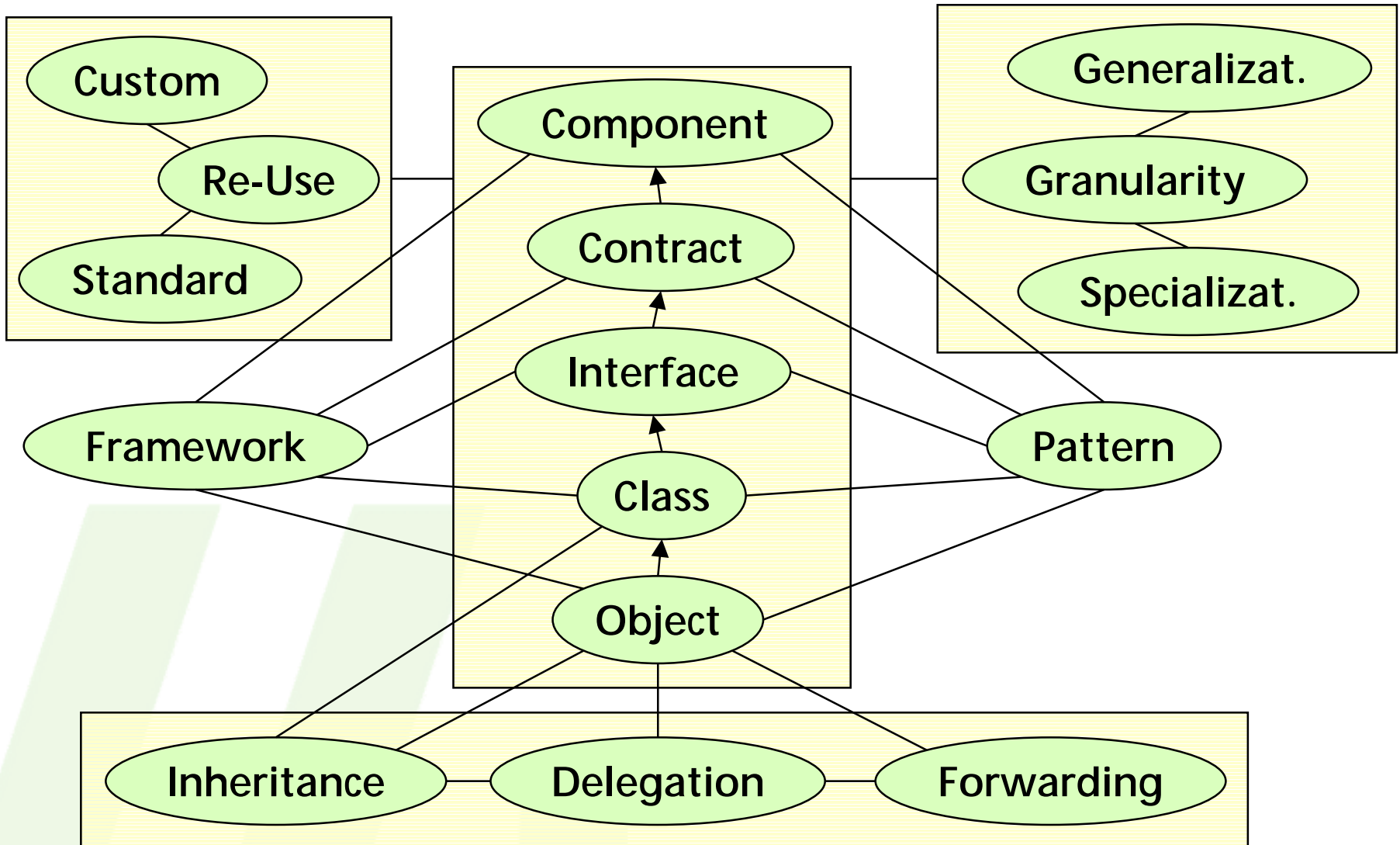
Library Weight



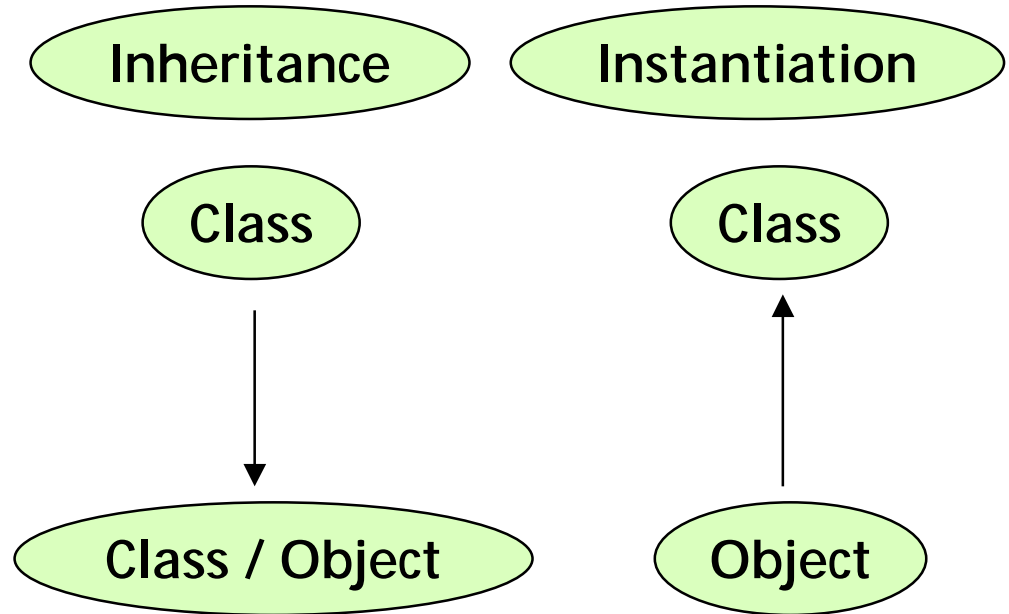
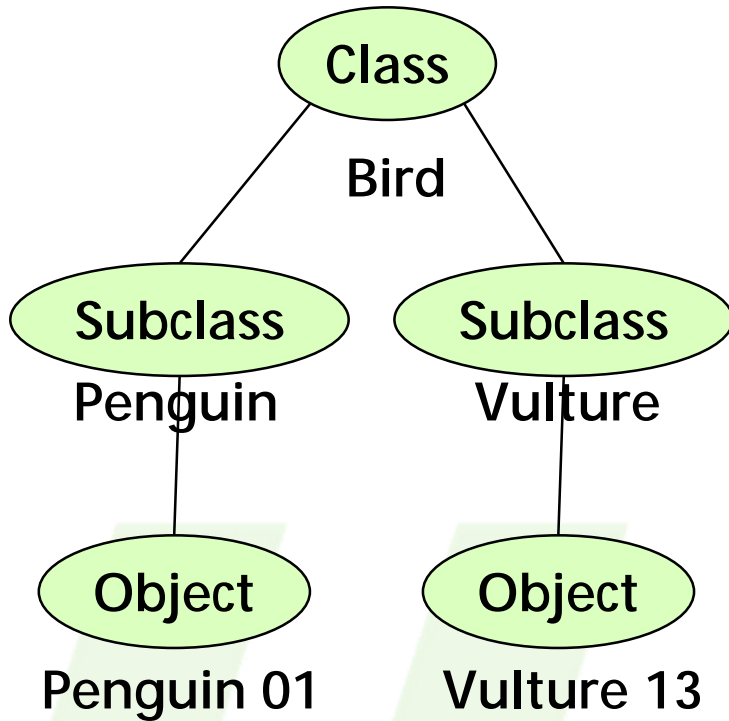
Part 2: Theory



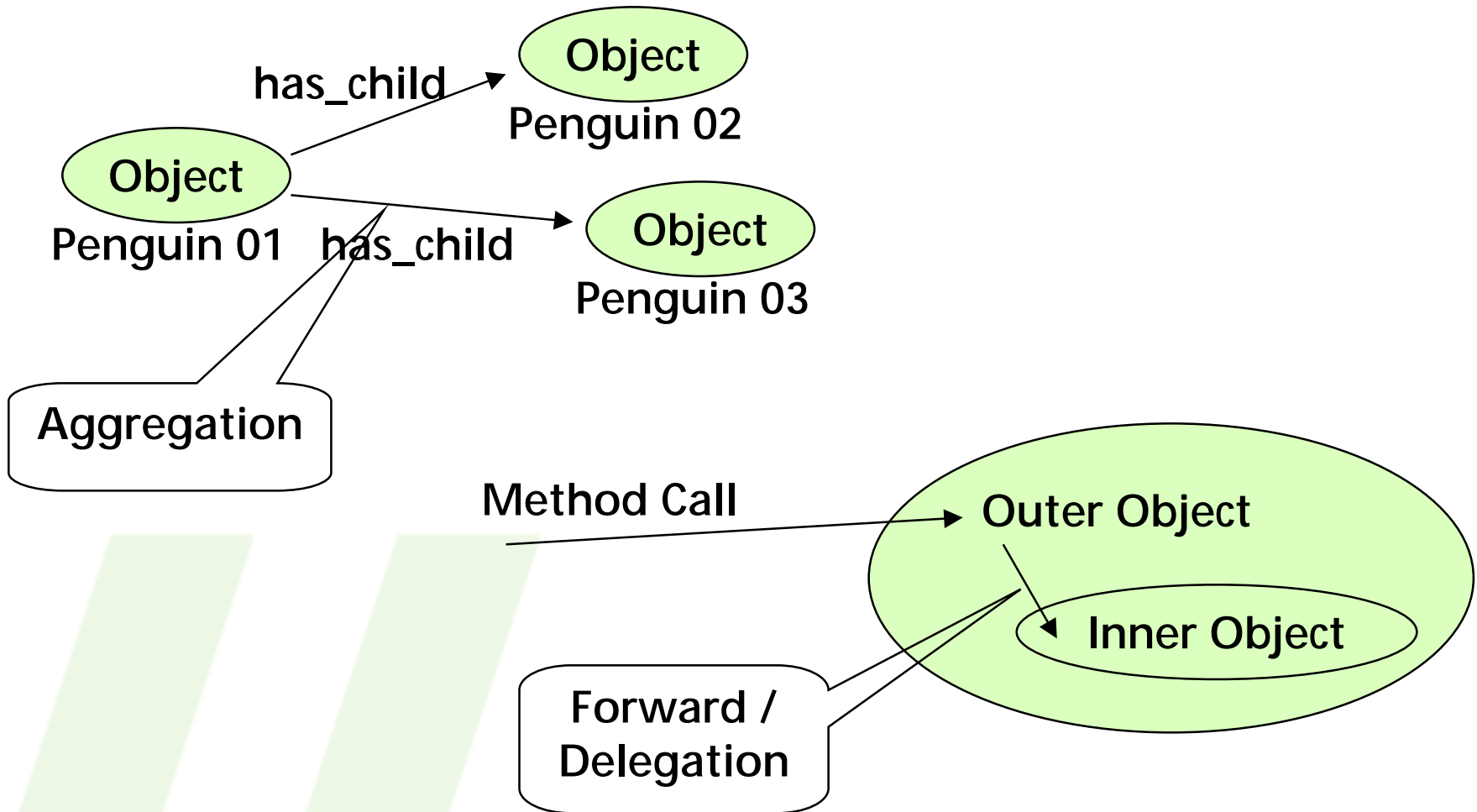
Concepts & Philosophies



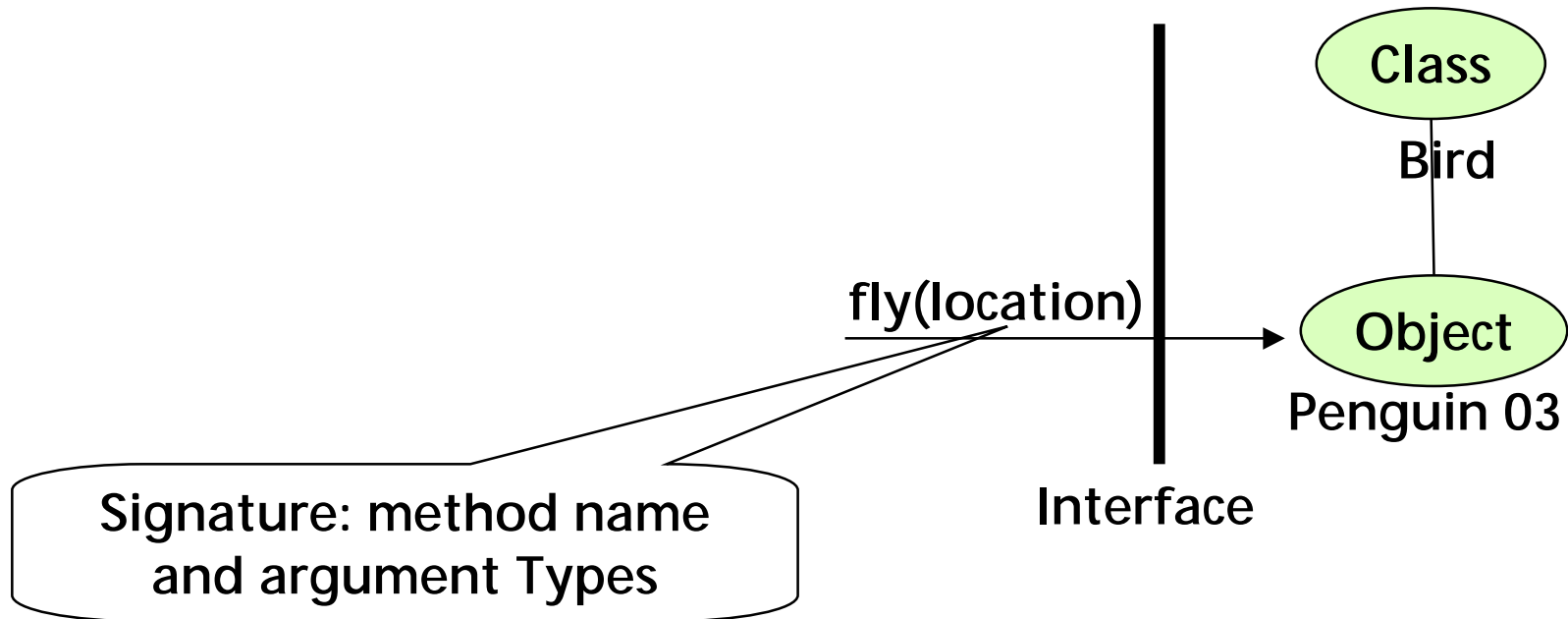
Inheritance & Instantiation



Aggregation, Forwarding & Delegation



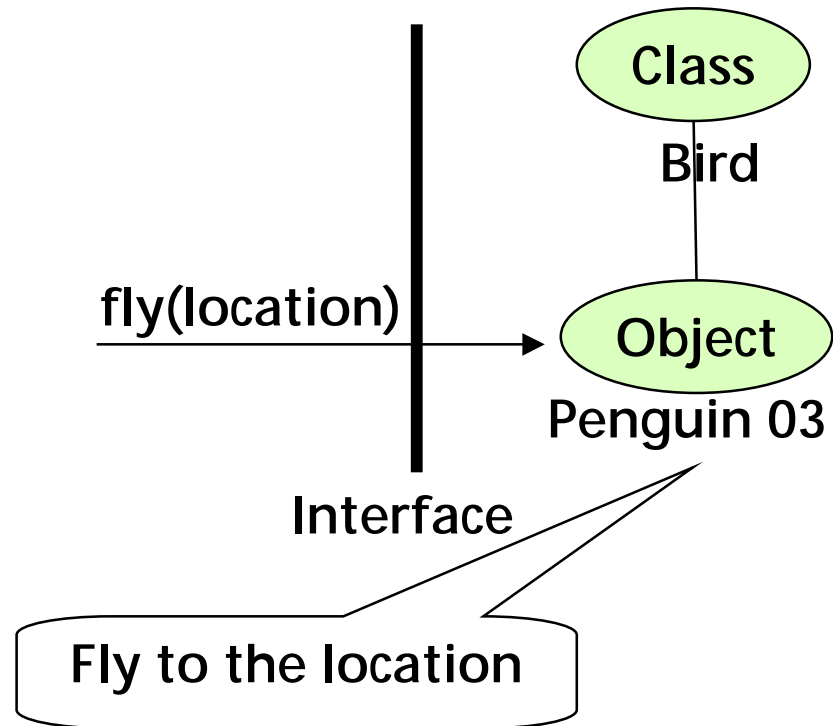
Interface Syntax: Signatures



Interface Semantics: Contracts

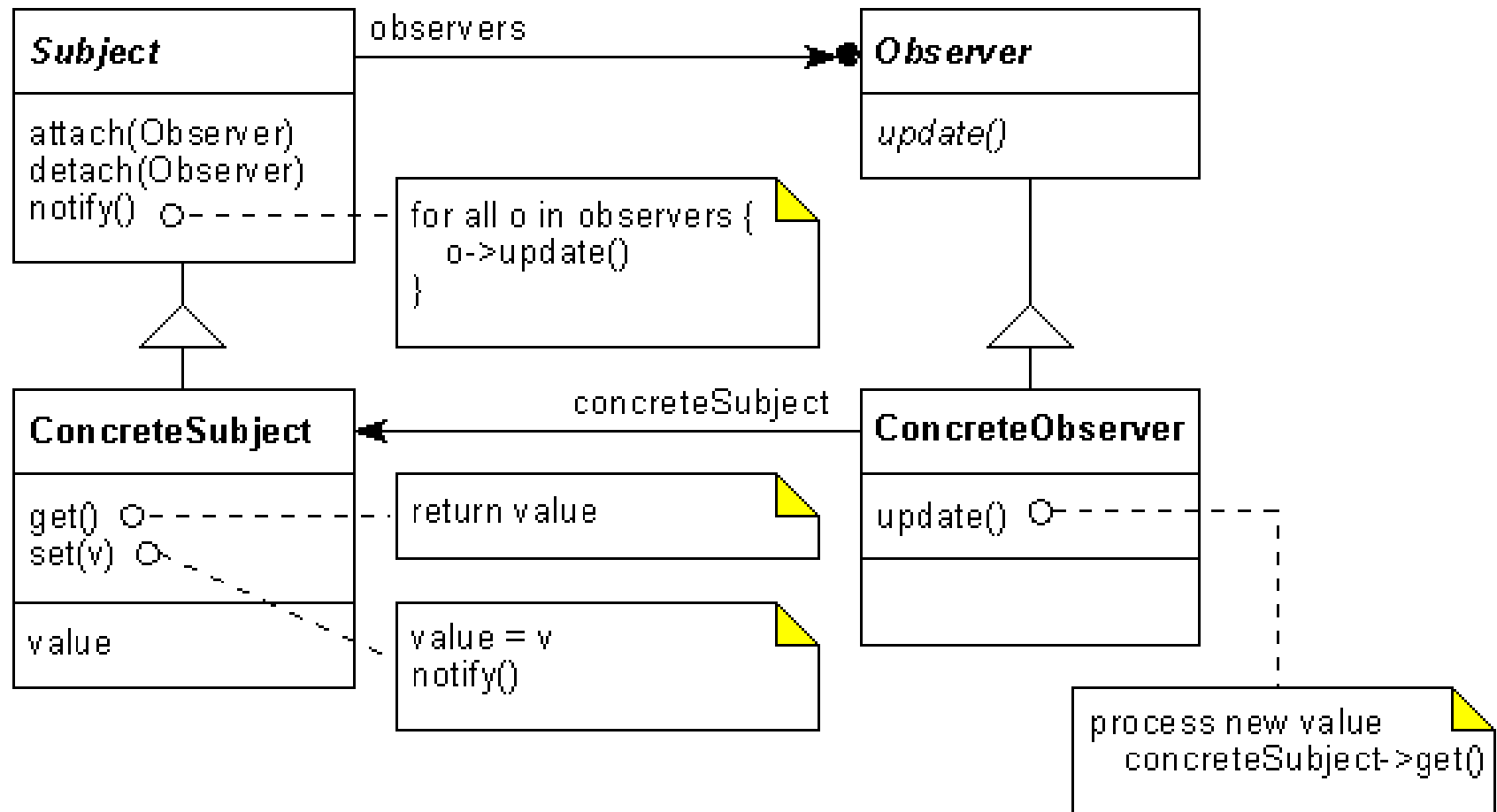
Defined using:

- Precondition
- Postcondition

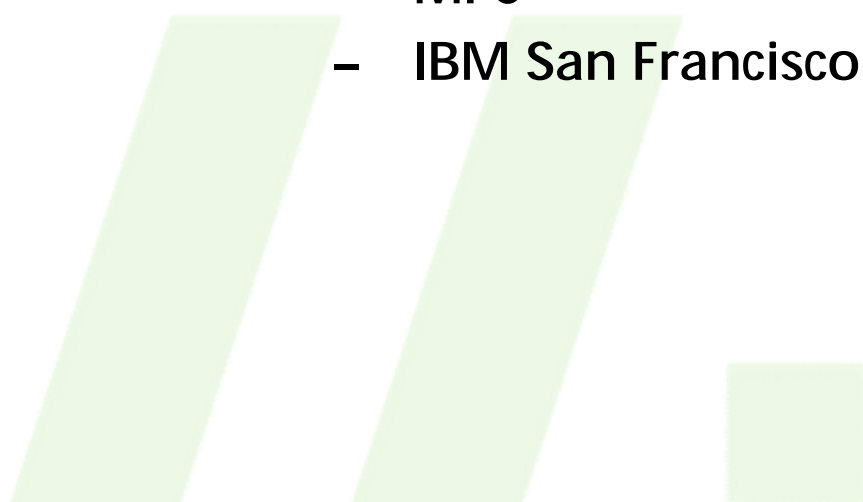


Patterns

The "Observer" pattern



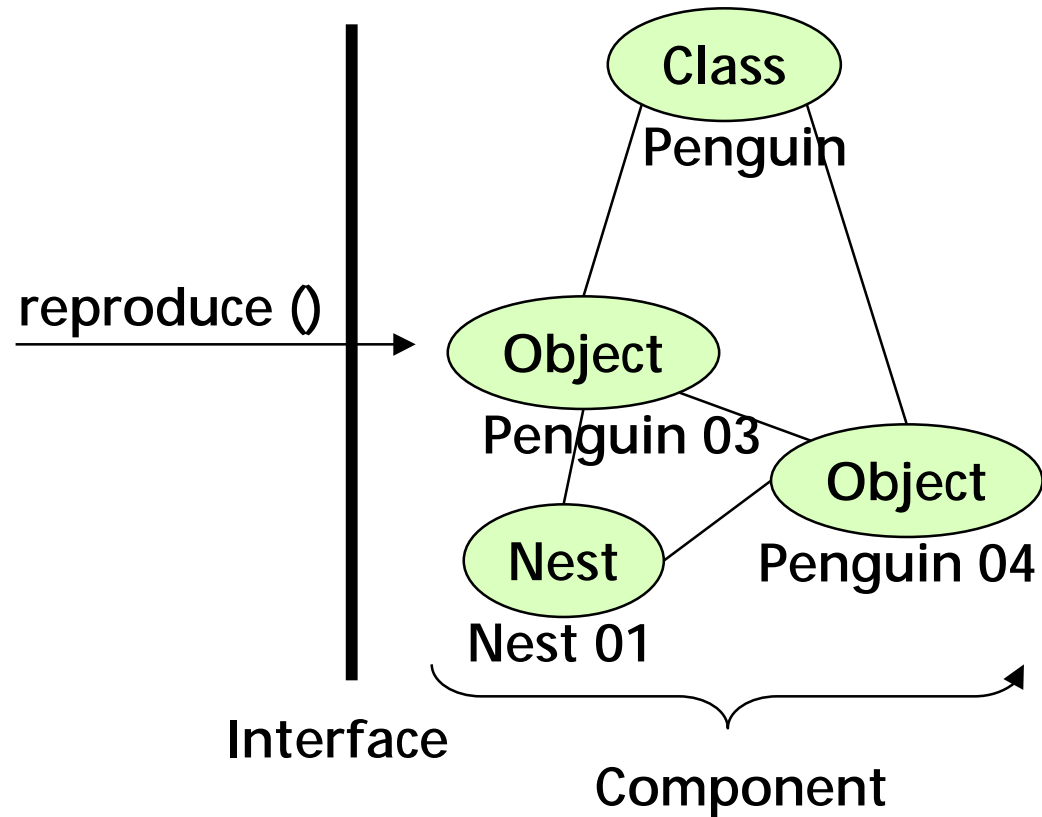
Frameworks

- A library of classes and preinstantiated objects
 - Typically requires you to subclass the framework classes
 - Examples:
 - Java Swing
 - MFC
 - IBM San Francisco
- 

Components

Defined using:

- Interface
- Unit of deployment



Part 3: Case Studies



Java Beans

- Component Technology: Java
- Glue Code: Java
- Linking:
 - Procedure calls for initialization and setup
 - Events to communicate state changes to observers



Corba

- **Component Technology: Any language**
- **Glue Code: ORB facilitated remote procedure calls**
- **Linking:**
 - RPC, carrying procedure calls and events



Visual Basic and COM/DCOM

- Component Technology: C++
- Glue Code: Visual Basic
- Linking:
 - Procedure calls



Competitiveness Marketplace

- Component Technology: Tcl
- Glue Code: HTML (.adp) or TCL (.tcl)
- Linking:
 - HTML or Tcl



Unix Tools

- Component Technology: C
- Glue Code: Bash, Perl, ...
- Linking:
 - Program execution
 - TCP or Unix pipes



Part 4: Exercises



Exercises

Use the guidelines on the next slide to develop component architectures for the following applications:

- The Competitiveness.com “Capacity Exchange” application
- The CarConfigurator:
http://www.fraber.de/projects/car_config/
- The Microsoft office family (Word, Excel, PowerPoint)

Component Library Guidelines

- Application scenarios: Decide on the specificity/generality of your library. Decide for your type of target application. Try to stay more on the "specific" side, because specific components tend to be faster to develop.
- Decide about the granularity of your component library. Make sure that all components in your library are of the same level. Group your components into several libraries if you identify several levels. Define the following:
 - Unit of abstraction
 - Unit of accounting
 - Unit of analysis
 - Unit of compilation
 - Unit of delivery
 - Unit of dispute
 - Unit of extension
 - Unit of fault containment
 - Unit of instantiation
 - Unit of loading
 - Unit of locality
 - Unit of maintenance
 - Unit of system management
- Are there already component libraries available in your application scenario? Name them and fill out the following questions for each for them:
 - What is the scenario/granularity of the library?
 - What technology are they based on?
 - Why are these libraries not applicable in your context?
 - Analyze their design and identify their characteristics.
 - What can you learn from their design for your library?
 - How could you otherwise take advantage of them?
- How should the contracts for your component look like? Identify common/repeated parameters/objects in the contract specifications. Write some sample contracts and specify the interfaces for them.
- Identify the language/formalism to implement your components and your "Glue Code".
 - What is component code and what is glue code?
 - How are you going to specify the component interfaces?
 - How are you going to deal with version changes in components?
- Persistence and data storage: Do your components have to deal with persistent storage? How are you going to interface with the storage? Are there problems with transactions/concurrency or atomicity?
- Inheritance: How are you going to deal with inheritance? How are you going to enforce a shallow inheritance hierarchy?