

**fraber-**  
**consulting**

---

*Barcelona, February 21<sup>th</sup>, 2002*

# **Component Architecture**

## Part 1: Motivation & Intro

- Richness and Reach
  - Applications
  - Libraries

## Part 2: Libraries

- What is a Library?
- Organization Issues
  - Library Weight
  - Inheritance against Aggregation
  - Small Interfaces

## Part 3: Case Studies

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

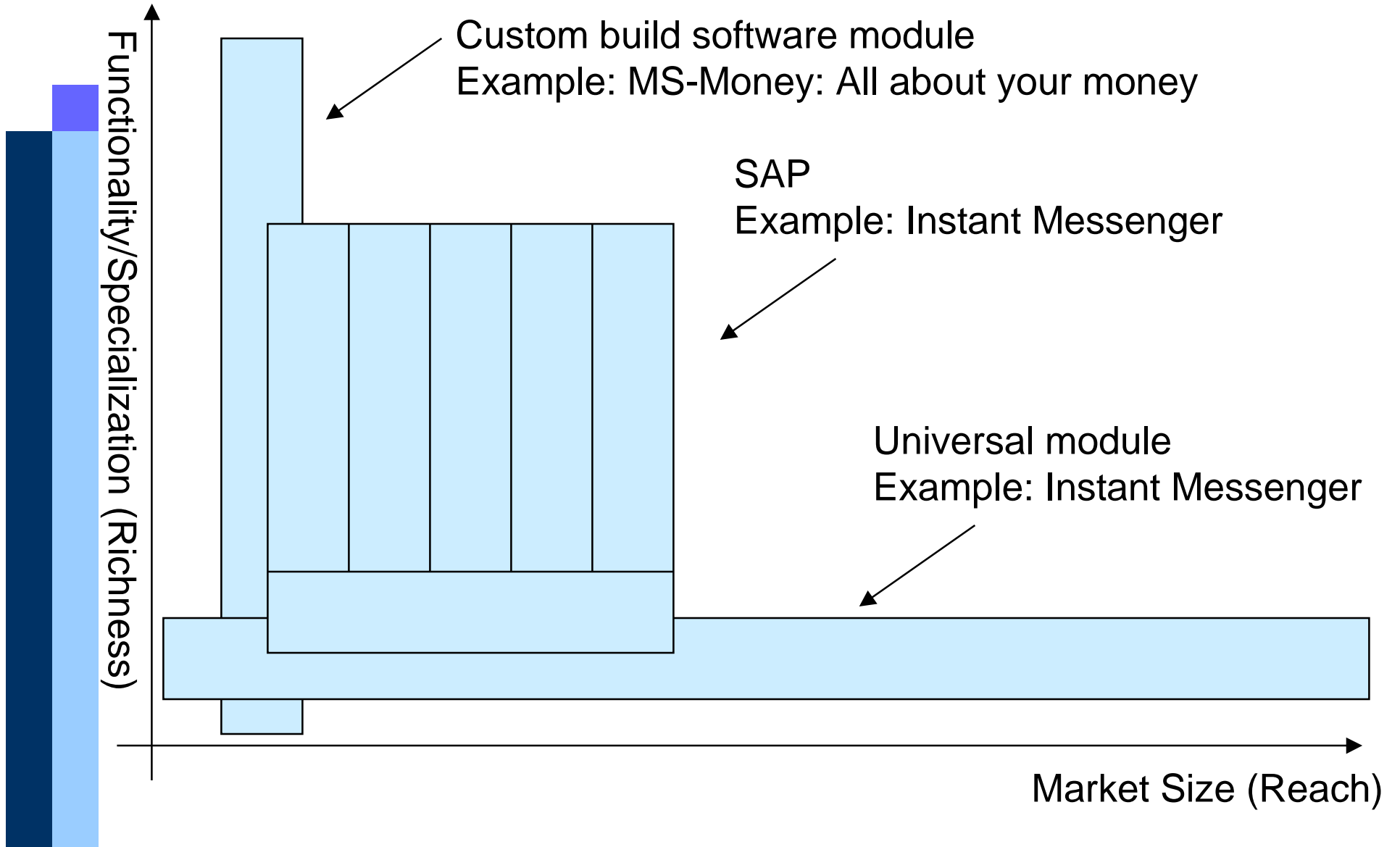
## Part 4: Exercises

- Exercises
- Component Library Guidelines



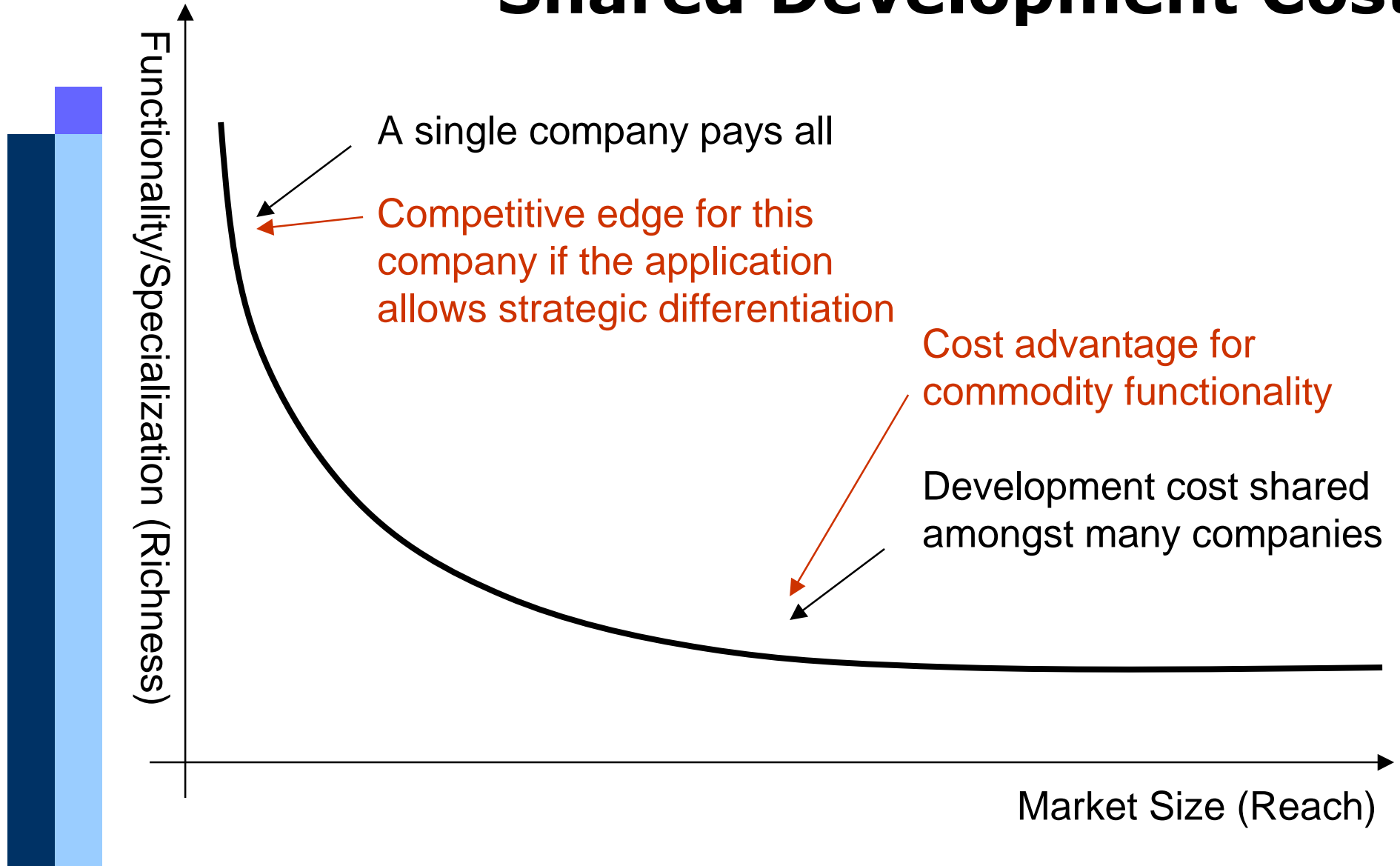
# Part 1: Motivation & Intro

# Richness and Reach



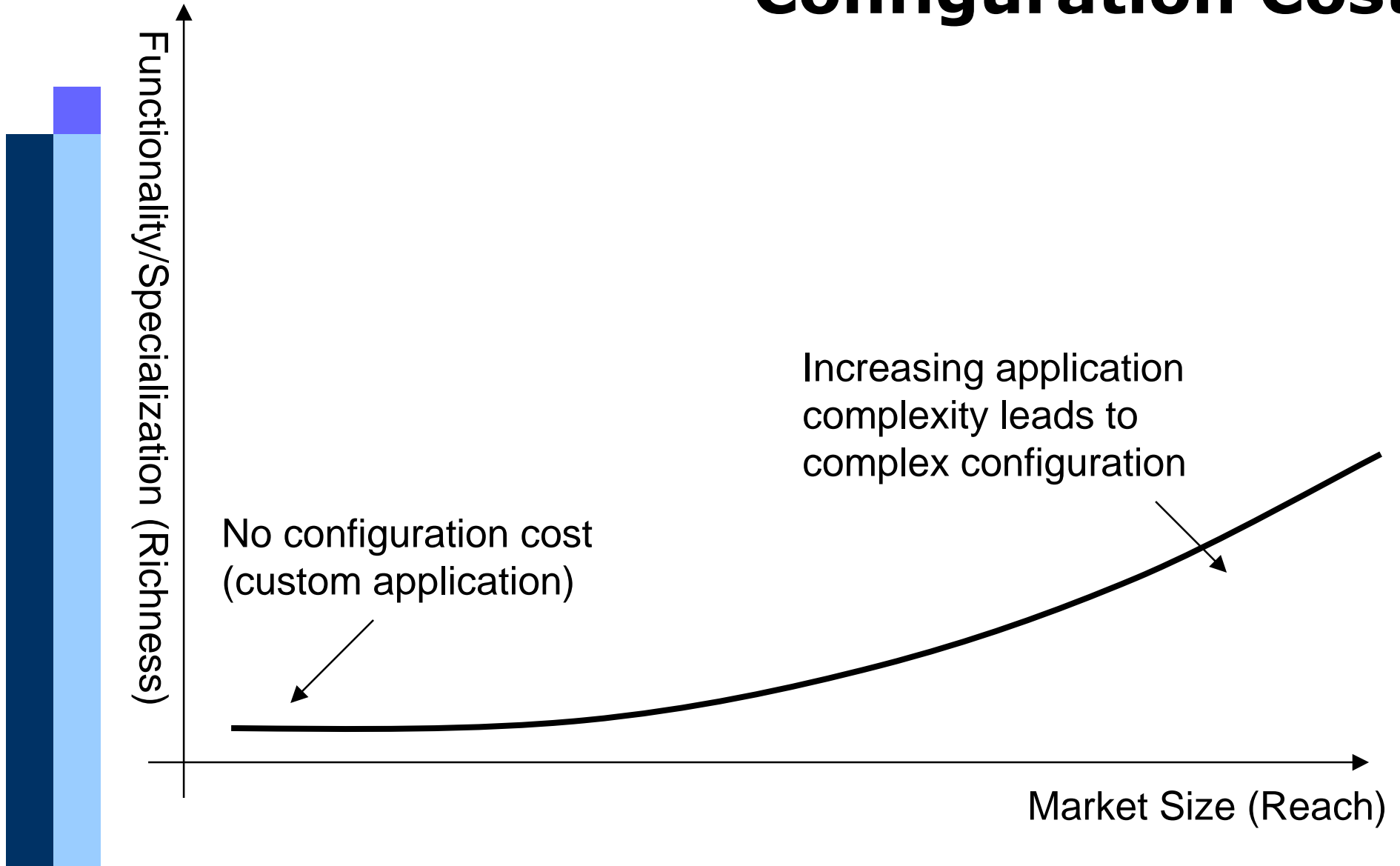
**fraber-**  
**consulting**

# Rich/Reach: Shared Development Cost



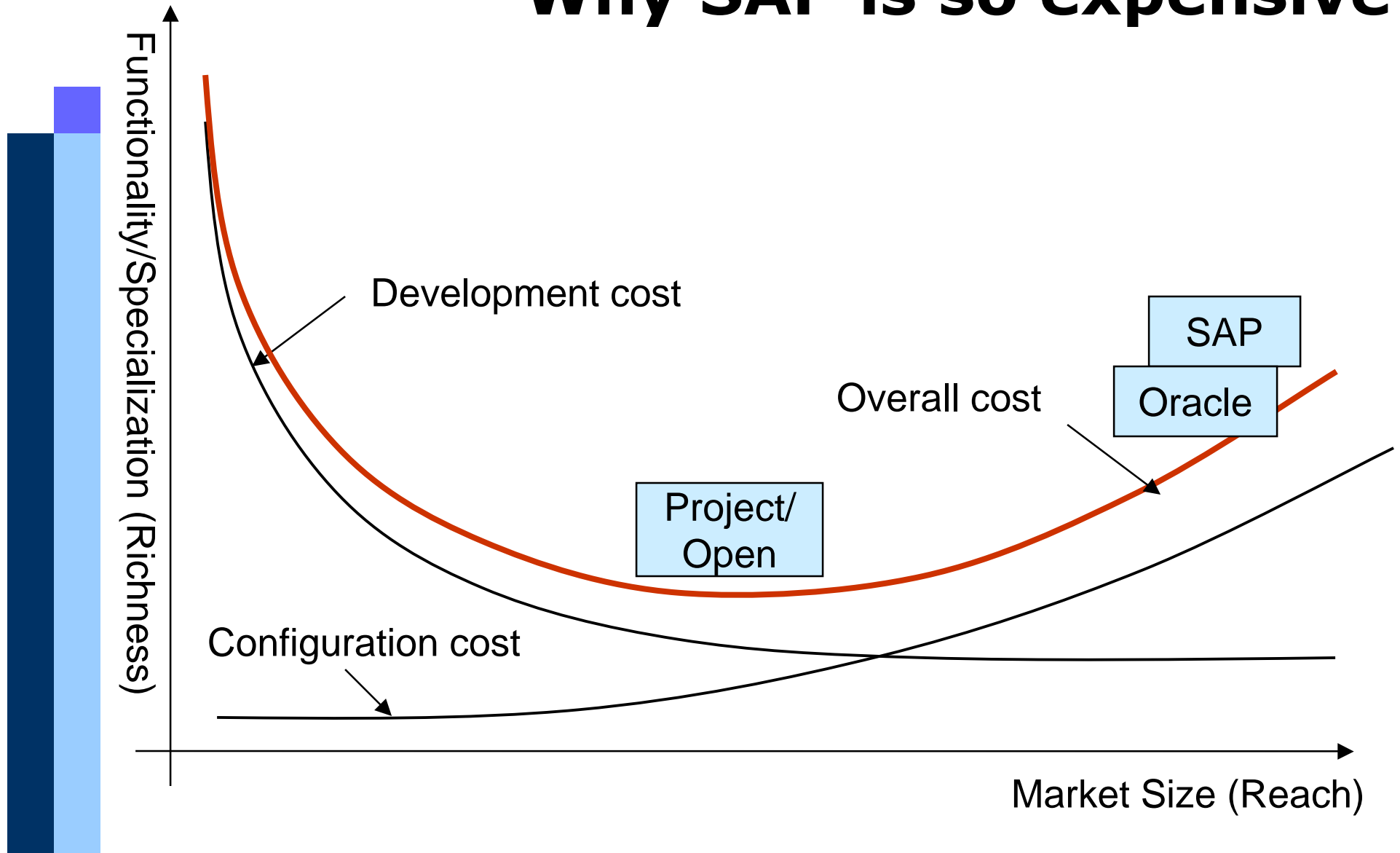
**fraber-**  
**consulting**

# Rich/Reach: Configuration Cost

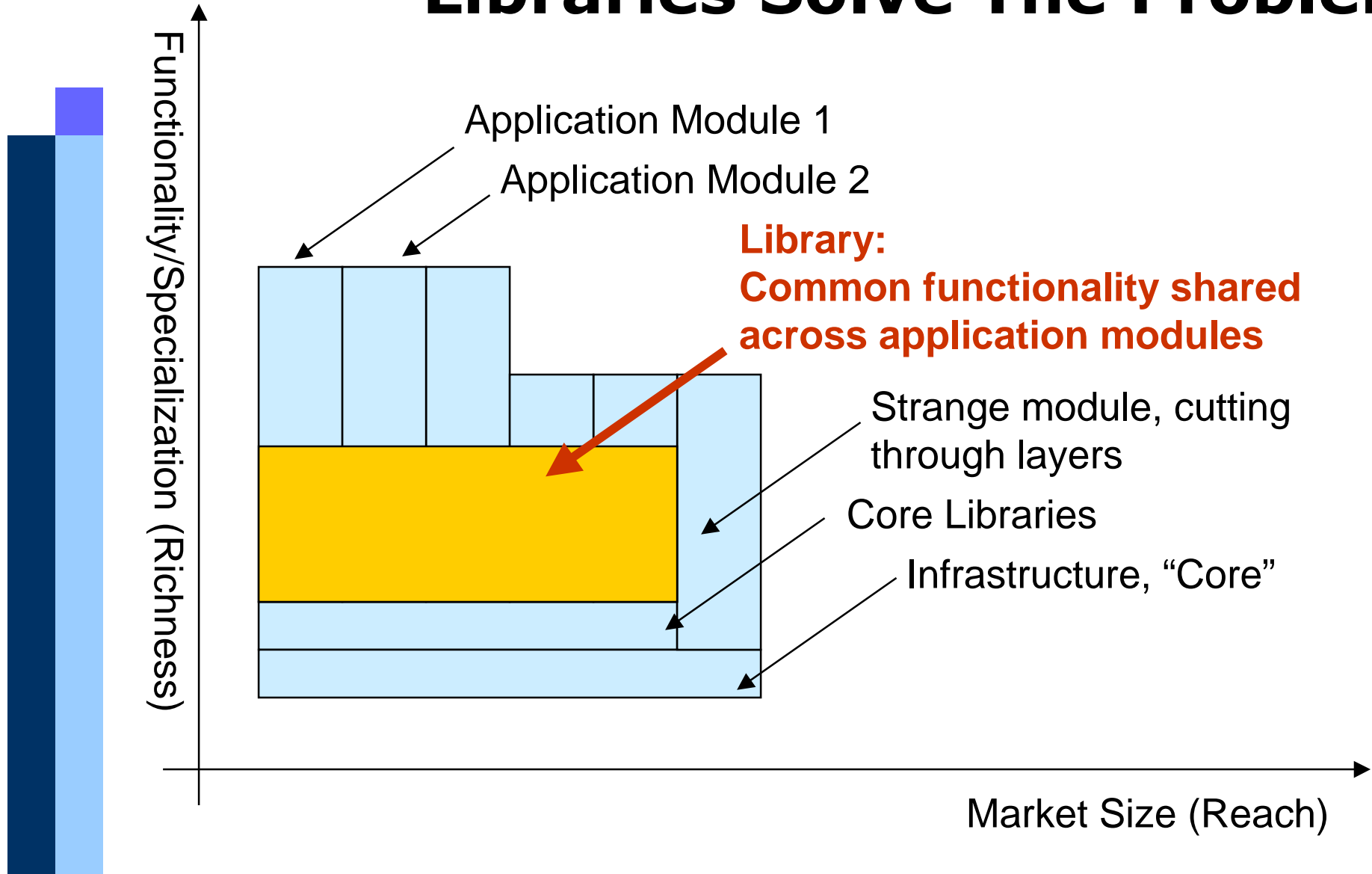


**fraber-consulting**

# Rich/Reach: Why SAP is so expensive



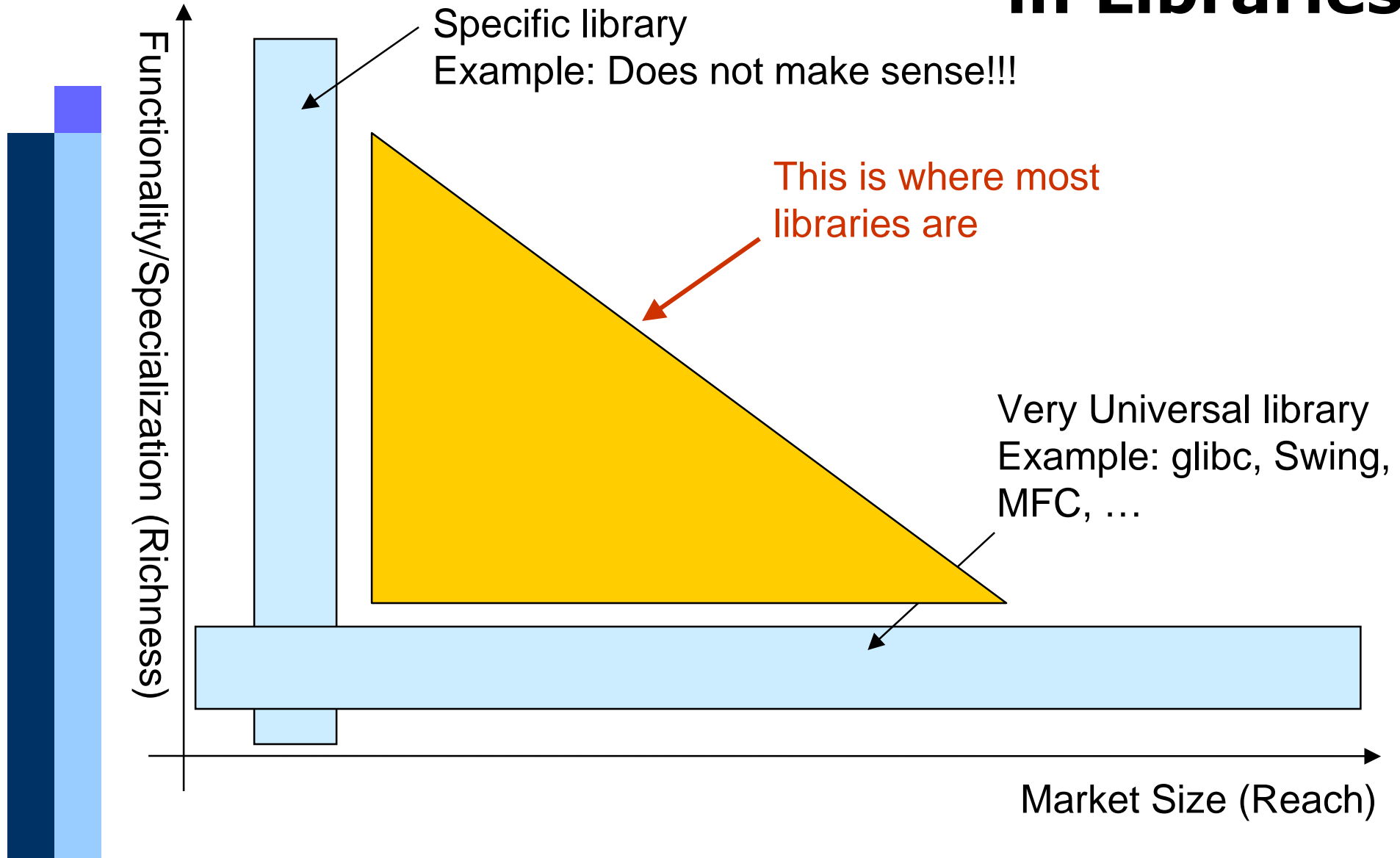
# Rich/Reach: Libraries Solve The Problem

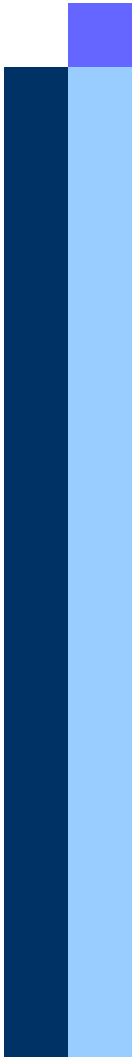




# fraber-consulting


# Rich/Reach in Libraries





# Part 2: Libraries

## What is a Library?

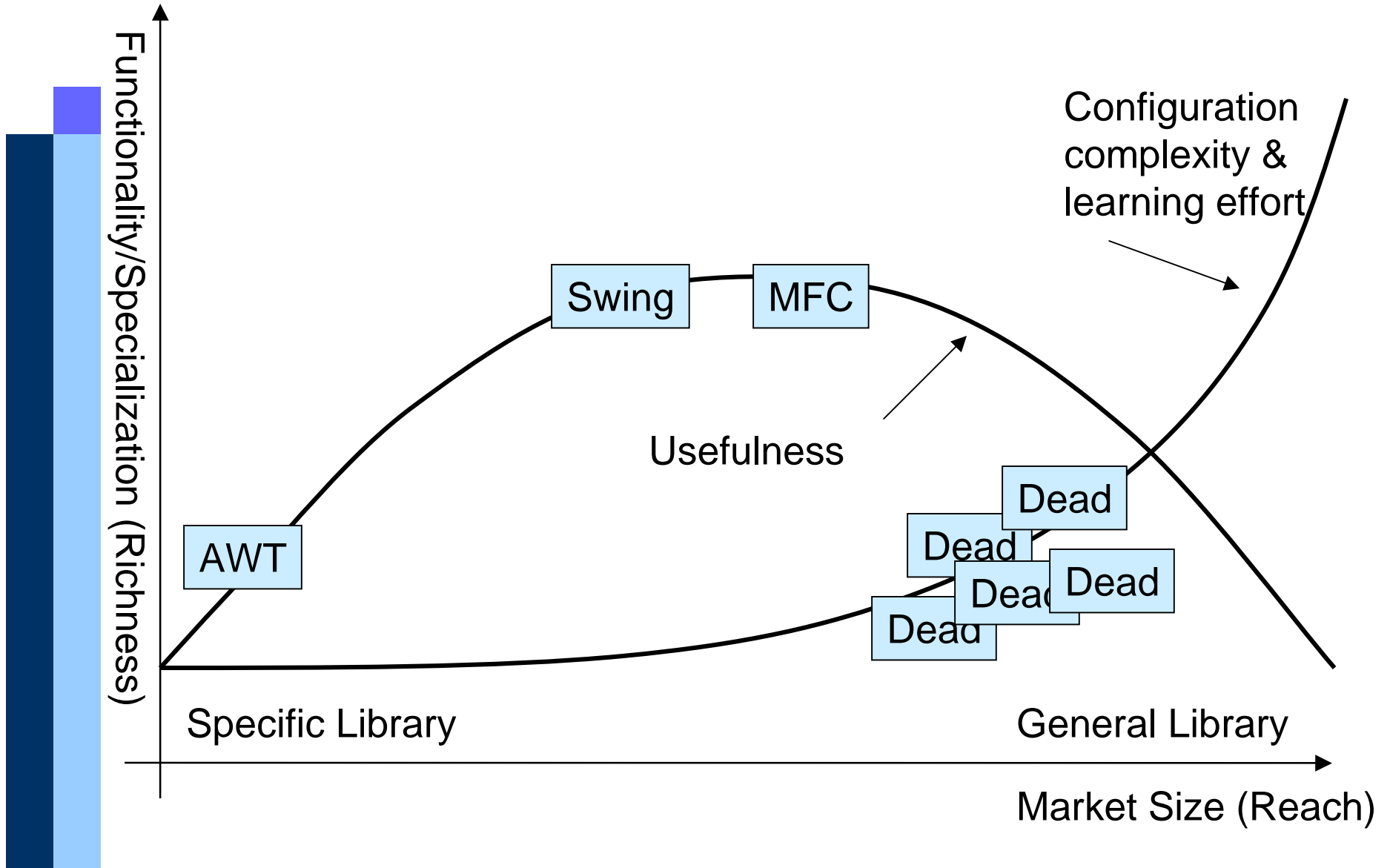
- 
- A collection of functionality shared across several modules
  - A collection of reusable objects/components
  - **Procedural**
    - Write a text on the screen
    - Determine the URL of a HTML page
  - **Object/Components**
    - Drop-down list
    - Marketplace Logo
    - Menu

# Library Organization Issues

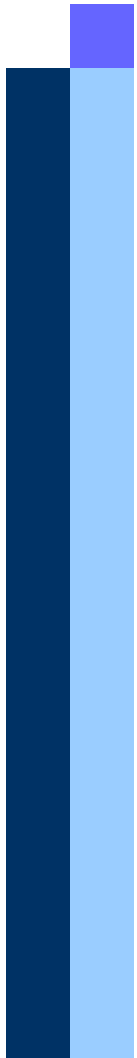
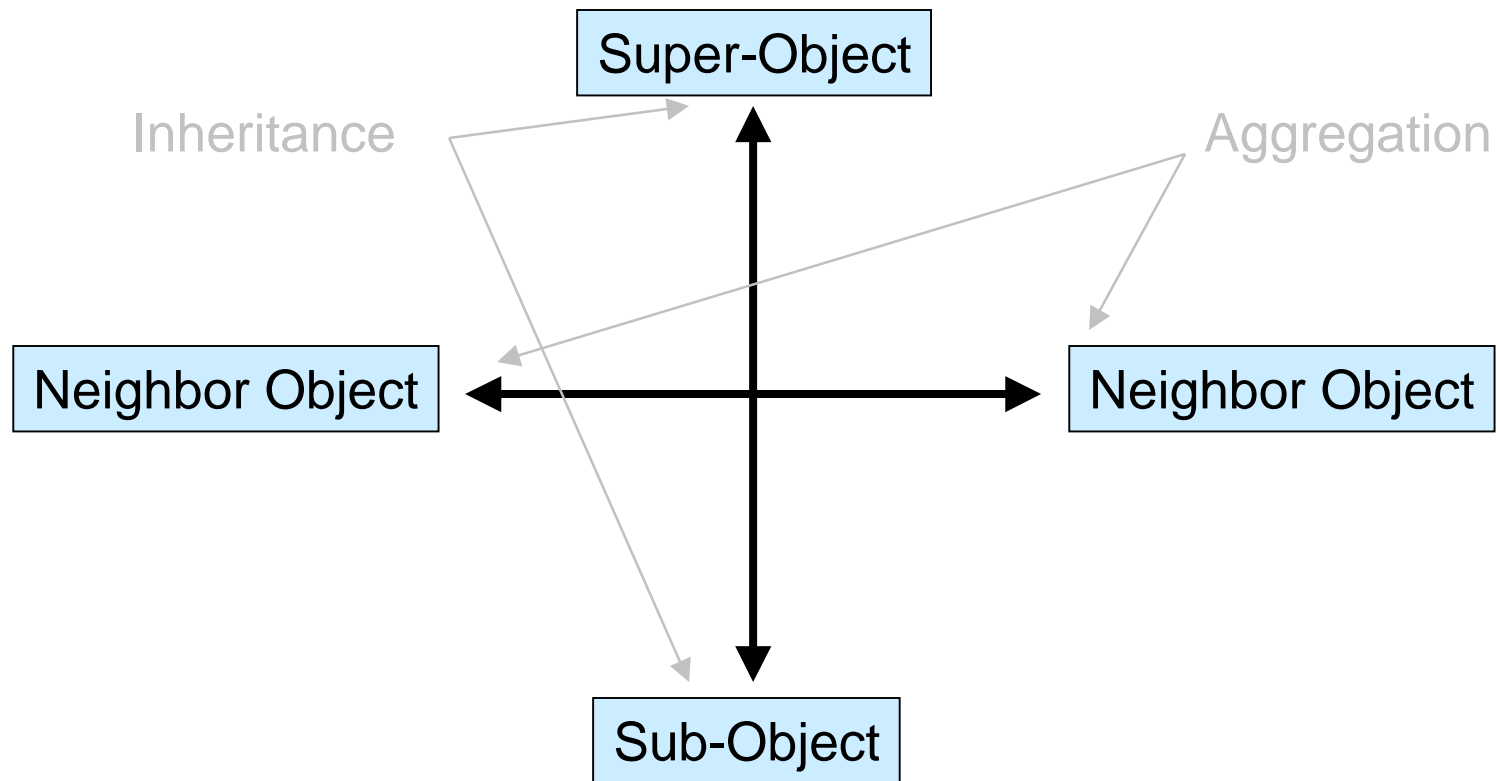


## Three Main Issues:

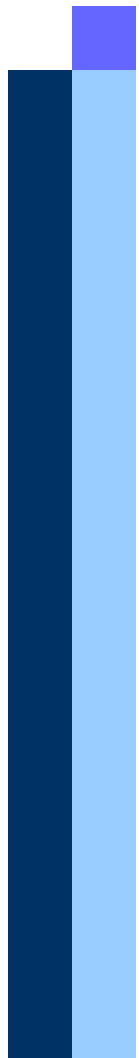
- Library Weight
- Aggregation against Inheritance
- Small Interface



# Inheritance & Aggregation



# Inheritance: "Is-a" Relationship



Inheritance

HTML Image Component

Marketplace Logo Component

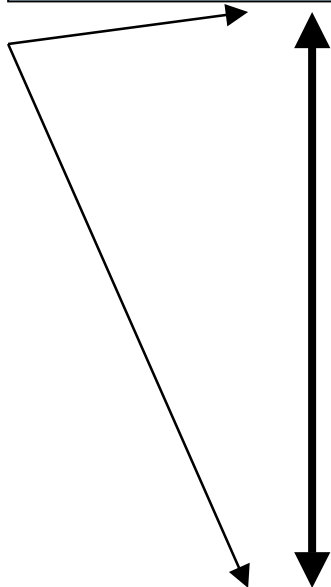
Registrese

//-AHORA

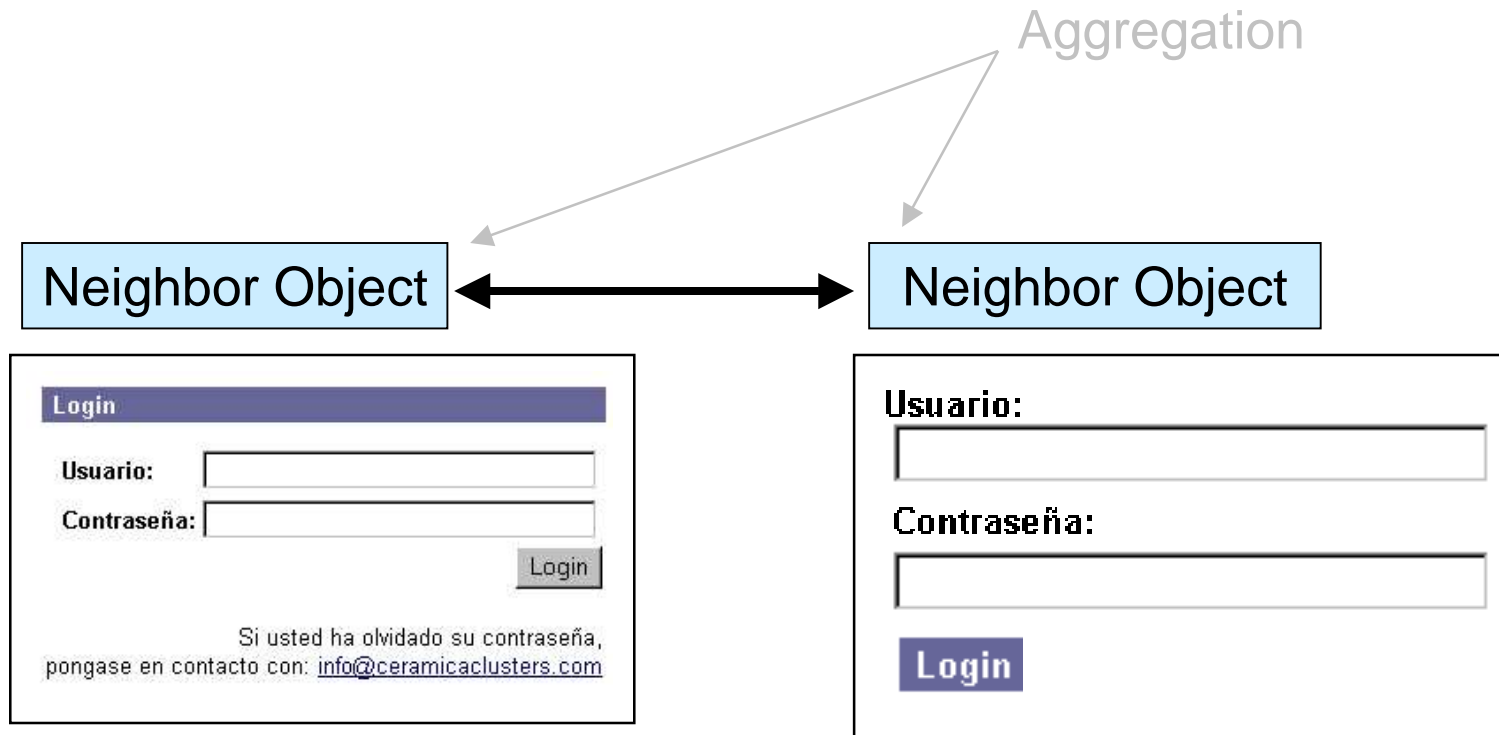
Plain GIF

//-Ceramica *CLUSTERS*

Changes with the URL




# Aggregation: "Has-a" Relationship



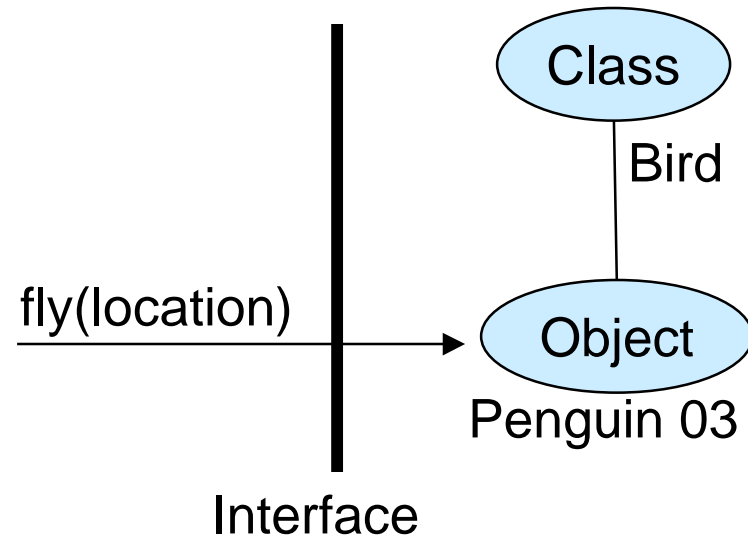


# **Inheritance & Aggregation**

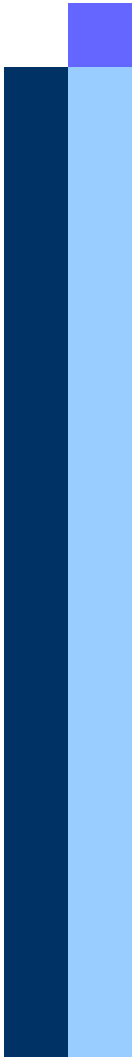
- 
- The decision is not always clear
  - General rule: Avoid deep inheritance trees
  - Aggregation is preferred today


## Small Interfaces


- Design your libraries/ components with a minimum number of external access points
- It is not always clear how to do that.
- “Beauty” or “elegancy” of a library captures a design with a small interface




# Part 3: Case Studies





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

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

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

# fraber- consulting **Visual Basic and COM/DCOM**

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

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






# Part 4: Exercises

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

- The Project/Open system
- A Java ERP
- A CarConfigurator (Product Choicboard)  
[http://www.fraber.de/projects/car\\_config/](http://www.fraber.de/projects/car_config/)
- The Microsoft office family (Word, Excel, PowerPoint)

## Component Library Guidelines

- 
- Application scenarios: Decide on the specificity/generalality 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?