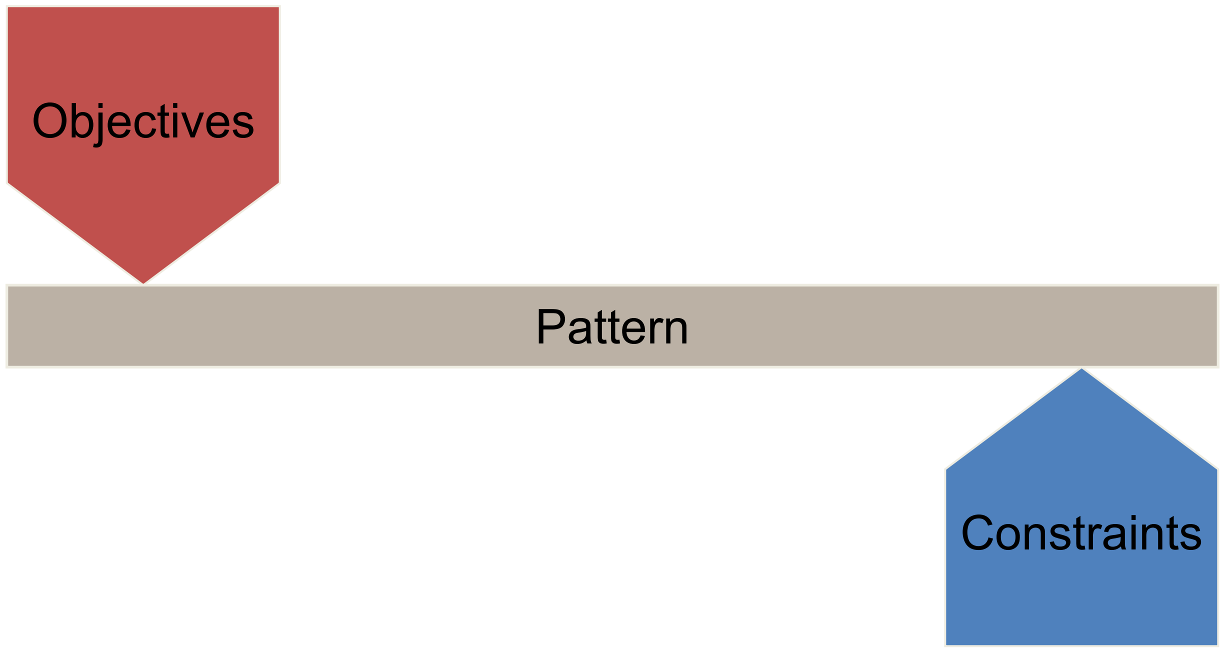
What is a Pattern?

A pattern is a solution to a problem in a context

- Context is the situation in which the pattern applies. The situation is recurring
- Problem refers to the goal we try to achieve in this context
  - It also refers to any constraints that occur in the context
- Solution is what we are after; a general design that anyone can apply which resolves the goal and set of constraints

Patterns – Balance Between Objectives and Constraints

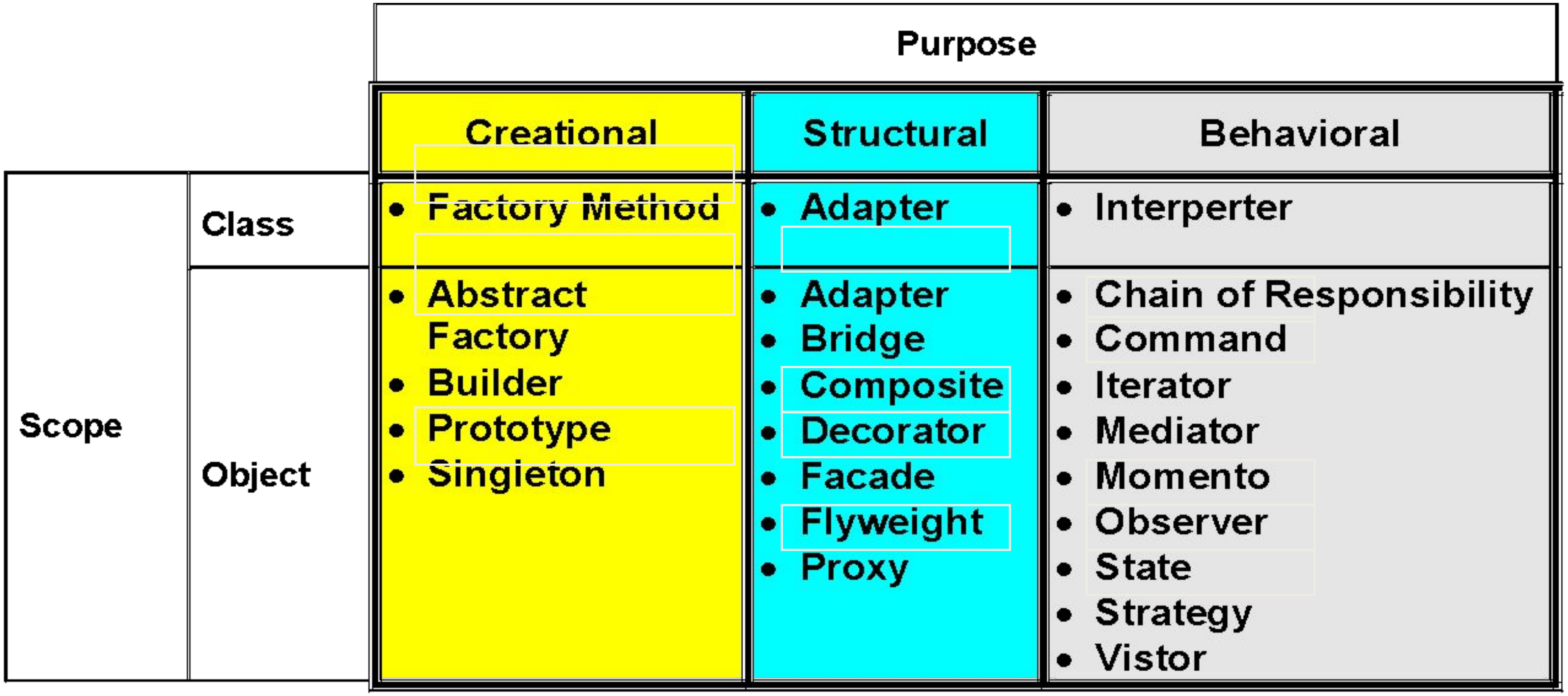


For an Industrial IT solution, constraints are generally more powerful than objectives!

Types of Design Patterns

- Creational Patterns
  - deal with initializing and configuring classes and objects
  - *how am I going to create my objects?*
- Structural Patterns
  - deal with decoupling the interface and implementation of classes and objects
  - *how classes and objects are composed to build larger structures*
- Behavioral Patterns
  - deal with dynamic interactions among societies of classes and objects
  - *how to manage complex control flows (communications)*

Design Pattern Classification - GoF

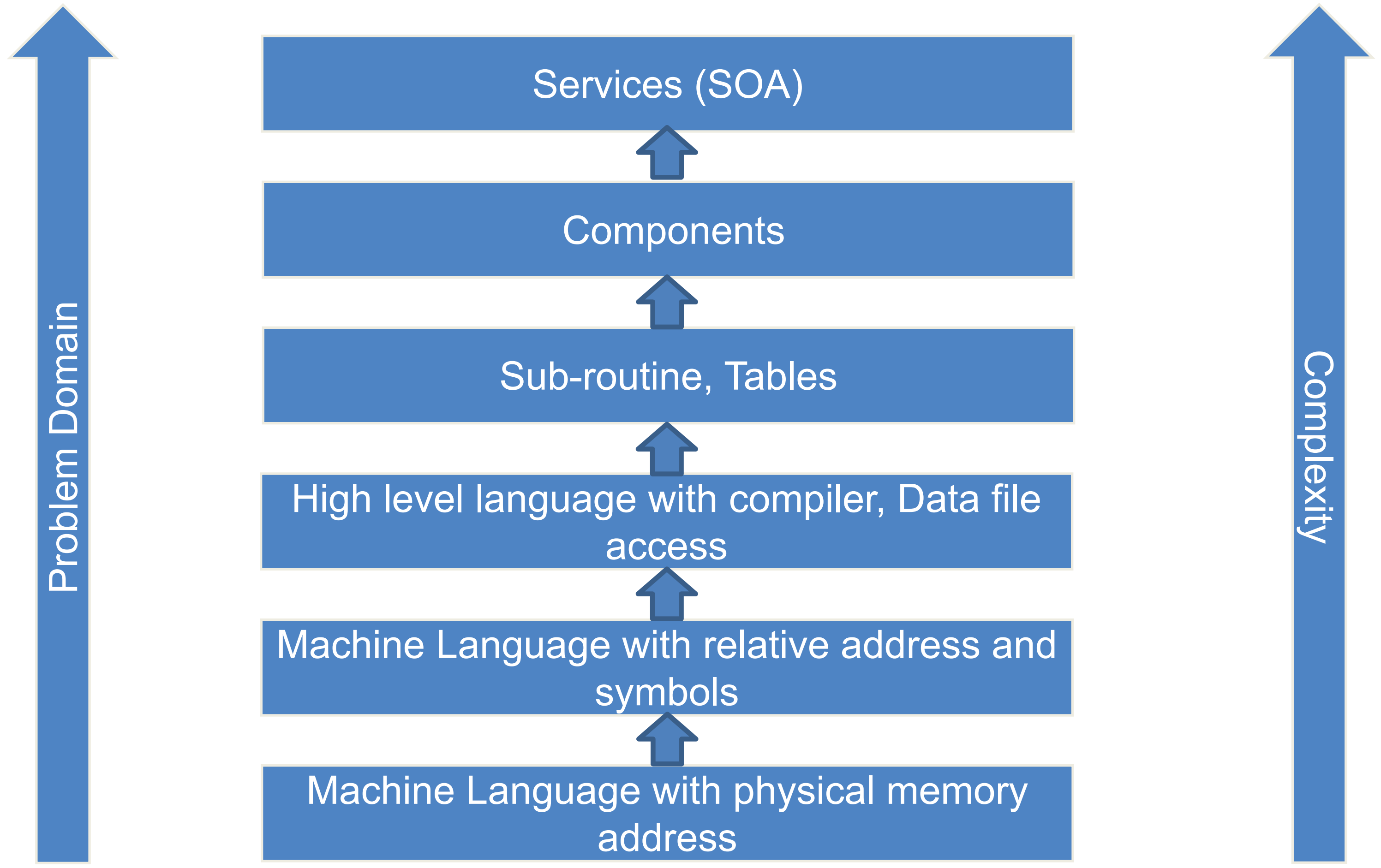


		Purpose		
		Creational	Structural	Behavioral
Scope	Class	• Factory Method	• Adapter	• Interperter
	Object	• Abstract Factory • Builder • Prototype • Singleton	• Adapter • Bridge • Composite • Decorator • Facade • Flyweight • Proxy	• Chain of Responsibility • Command • Iterator • Mediator • Momento • Observer • State • Strategy • Vistor

Evolution of Large Systems

“A complex system that works is invariably found to have evolved from a simple system that worked…A complex system designed from scratch never works and cannot be patched up to make it work. You have to start over, beginning with a working
simple system.” — John Gall in *Systematics: How Systems Really Work and How They Fail*

Level of Abstraction in IT Solution



Problem Domain
Services (SOA)
Components
Sub-routine, Tables
High level language with compiler, Data file access
Machine Language with relative address and symbols
Machine Language with physical memory address
Complexity

The entire history of software engineering is one of rising level of abstraction

Patterns are everywhere!!!

- Architecture
  - Seen in buildings, towns and cities
- Application
  - 3-tier architecture, client-server systems and the web
- Domain specific
  - Concurrent systems, real-time systems
- Business process
  - Interaction between businesses, customers and data
- Organizational
  - Structure and practices of human organizations
- User interface
  - Video game designers, GUI builders

Patterns are not alone about design or application development – patterns are everywhere!

As Level of Abstraction increased in Software, so is for Patterns

“Each pattern then depends both on the smaller patterns it contains, and on the larger patterns within which it is contained.” — Christopher Alexander in *The Timeless Way of Building*

The Timeless Way of Building is a 1979 book by Christopher Alexander that proposes a new theory of architecture (and design in general) that relies on the understanding and configuration of design patterns. It is essentially the introduction to A Pattern Language and The Oregon Experiment, providing the philosophical background to the Center for Environmental Structure series.

Pattern Language

- A **pattern language**, a term coined by architect Christopher Alexander, is a structured method of describing good design practices within a field of expertise.
- Advocates of this design approach claim that ordinary people of ordinary intelligence can use it to successfully solve very large, complex design problems.

Pattern Language

- Like all languages, a pattern language has vocabulary, syntax, and grammar.
- The odd part is that the language is applied to some complex activity other than communication.

History

- Christopher Alexander, M. Silverstein, S. Angel, S. Ishikawa, and D. Abrams. *The Oregon Experiment*, volume 3 of *Center for Environmental Structure Series*. Oxford University Press, New York, NY, 1975.
- Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Abstraction and Reuse in Object-Oriented Designs. In O. M. Nierstrasz, editor, *ECOOP'93: Object-Oriented Programming - Proc. of the 7th European Conference*, pages 406-431, Berlin, Heidelberg, 1993. Springer.

History

- Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley Publishing Company, New York, NY, 1995.

Design Patterns
Elements of Reusable Object Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

Four essential elements of a pattern

- **Pattern Name** - is a handle we can use to describe a design problem, its solutions, and consequences in a word or two.

- **Problem** - describes when to apply the pattern. It explains the problem and its context.
  - It might describe specific design problems such as how to represent algorithms as objects.
  - It might describe class or object structures that are symptomatic of an inflexible design.
  - Sometimes the problem will include a list of conditions that must be met before it makes sense to apply the pattern.

Four essential elements of a pattern

- **Solution** - describes the elements that make up the design, their relationships, responsibilities, and collaborations.
  - The solution doesn't describe a particular concrete design or implementation, because a pattern is like a template that can be applied in many different situations.
  - Instead, the pattern provides an abstract description of a design problem and how a general arrangement of elements (classes and objects in our case) solves it.

Four essential elements of a pattern

- **Consequences** - are the results and trade-offs of applying the pattern.
  - Though consequences are often unvoiced when we describe design decisions, they are critical for evaluating design alternatives and for understanding the costs and benefits of applying the pattern.
  - The consequences for software often concern space and time trade-offs.
  - They may address language and implementation issues as well.
  - Since reuse is often a factor in object-oriented design, the consequences of a pattern include its impact on a system's flexibility, extensibility, or portability. Listing these consequences explicitly helps you understand and evaluate them.

Describing Design Patterns

- **Pattern Name and Classification**

  The pattern's name conveys the essence of the pattern succinctly. A good name is vital, because it will become part of your design vocabulary.
- **Intent**

  A short statement that answers the following questions: What does the design pattern do? What is its rationale and intent? What particular design issue or problem does it address?
- **Also Known As**

  Other well-known names for the pattern, if any.

Describing Design Patterns

- **Motivation**

  A scenario that illustrates a design problem and how the class and object structures in the pattern solve the problem. The scenario will help you understand the more abstract description of the pattern that follows.

- A**pplicability**

  What are the situations in which the design pattern can be applied? What are examples of poor designs that the pattern can address? How can you recognize these situations?

- **Structure**

  A graphical representation of the classes in the pattern using a notation based on the Object Modeling Technique (OMT). We also use interaction diagrams to illustrate sequences of requests and collaborations between objects.

Describing Design Patterns

- **Participants**

  The classes and/or objects participating in the design pattern and their responsibilities.
- **Collaborations**

  How the participants collaborate to carry out their responsibilities.
- **Consequences**

  How does the pattern support its objectives? What are the trade-offs and results of using the pattern? What aspect of system structure does it let you vary independently?

Describing Design Patterns

- **Implementation**

  What pitfalls, hints, or techniques should you be aware of when implementing the pattern? Are there language-specific issues?
- **Sample Code**

  Code fragments that illustrate how you might implement the pattern in C++ or Smalltalk.
- **Known Uses**

  Examples of the pattern found in real systems. We include at least two examples from different domains.
- **Related Patterns**

  What design patterns are closely related to this one? What are the important differences? With which other patterns should this one be used?