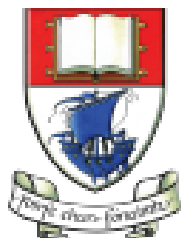


Software Paradigms

Produced by: Eamonn de Leastar (edeleestar@wit.ie)
Dr. Siobhan Drohan (sdrohan@wit.ie)



Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
<http://www.wit.ie/>

Literate Programmer

Good design and programming is not learned by generalities, but by seeing how significant programs can be made clean, easy to read, easy to maintain and modify, human-engineered, efficient, and reliable, by the application of good design and programming practices. Careful study and imitation of good designs and programs significantly improves development skills.

Kernighan & Plauger (1978)

A Paradigm is...

...“a framework containing the basic assumptions, ways of thinking, and methodology that are commonly accepted by members of a scientific community.”

Software Paradigms

- Paradigms are used as a means to classify programming languages.
- The rate of change in the software discipline has seen numerous paradigms.

Software Paradigms

There are LOTS of them!

https://en.wikipedia.org/wiki/Programming_paradigm

Some Popular Paradigm Examples

- **Imperative:** Control flow is an explicit sequence of commands e.g. C, COBOL.
- **Declarative:** Programs state the result you want, not how to get it e.g. Prolog, SQL.
- **Procedural:** Imperative programming with procedure calls e.g. Pascal, Modula-2.
- **Functional:** Computation proceeds by function calls that avoid any global state e.g. Haskell, Lambdas in Java.
- **Object-Oriented:** Computation is effected by sending messages to objects; objects have state and behaviour e.g. Java, C++
- **Event-Driven** — Control flow is determined by asynchronous actions (from humans or sensors) e.g. Visual Basic

Paradigms for This Module

- **Object Oriented Programming**

- OO Principles (particularly SOLID principles)
- Java Programming Language
- Xtend Programming Language

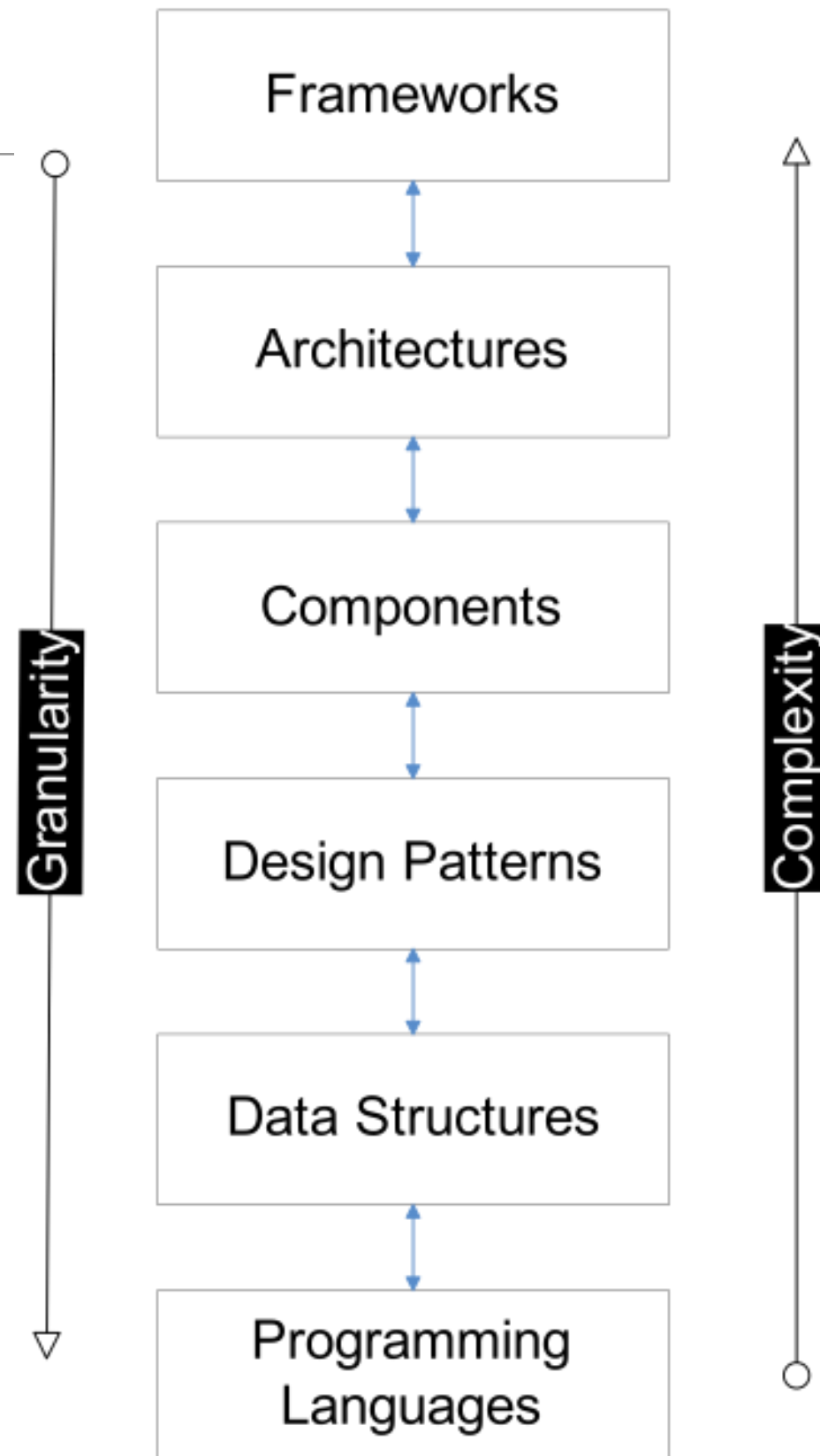
- **Agile Methods**

- Test Driven Development (TDD)
- Automated Build / Configuration Management

- **Network Programming**

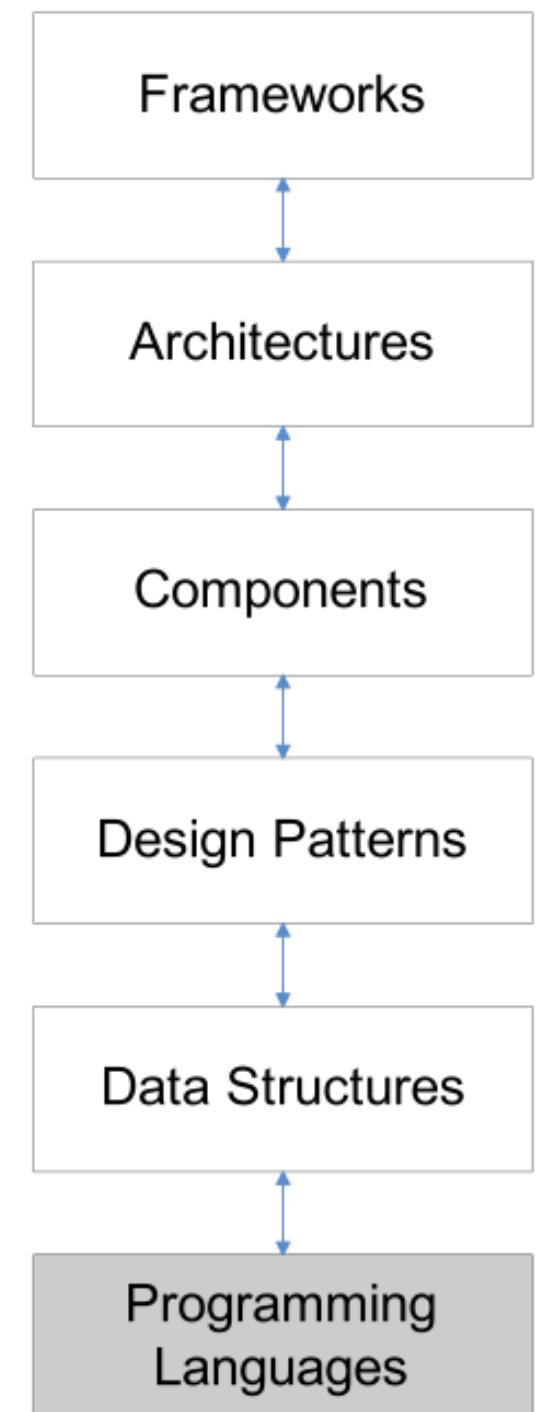
- HTTP/REST

Context



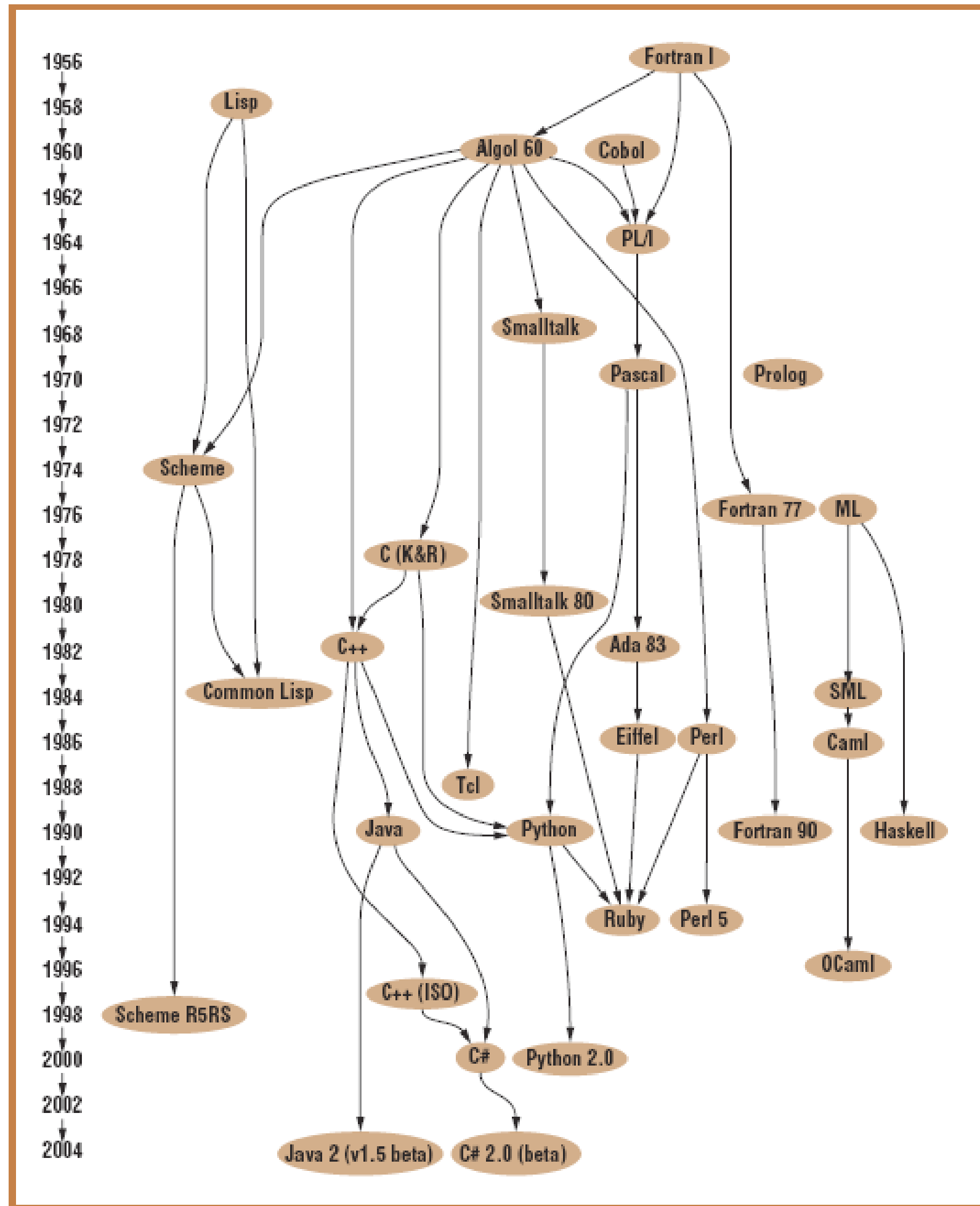
Programming Languages

- A programming language is a system of signs used to communicate a task/algorithm to a computer, causing the task to be performed.
- The task to be performed is called a computation, which follows absolutely precise and unambiguous rules.
- Three components:
 - The **syntax** of the language is a way of specifying what is legal in the phrase structure of the language; (analogous to knowing how to spell and form sentences English)
 - The second component is **semantics**, or meaning, of a program in that language.
 - Certain **idioms** that a programmer needs to know to use the language effectively - are usually acquired through practice and experience.

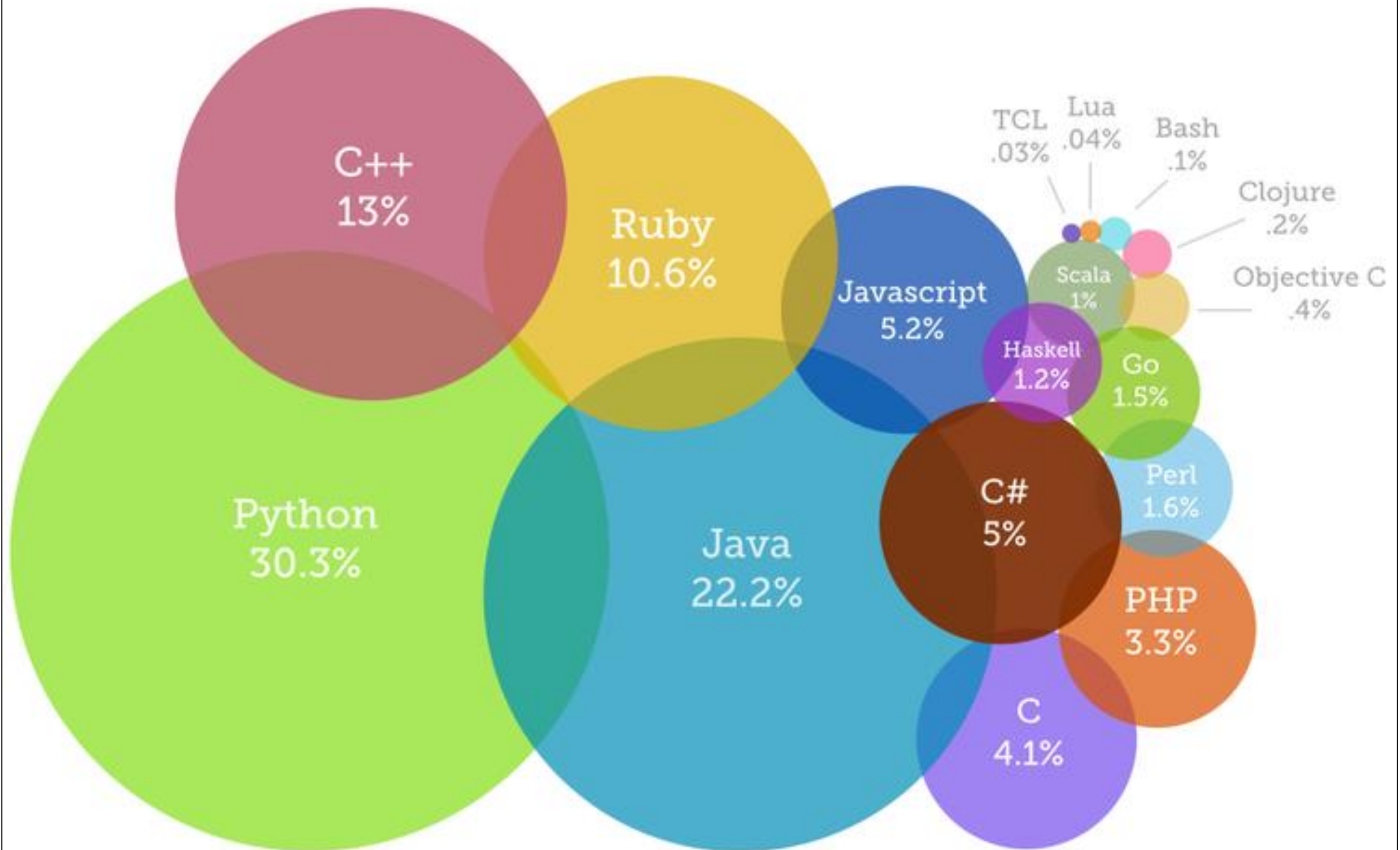


Family Tree

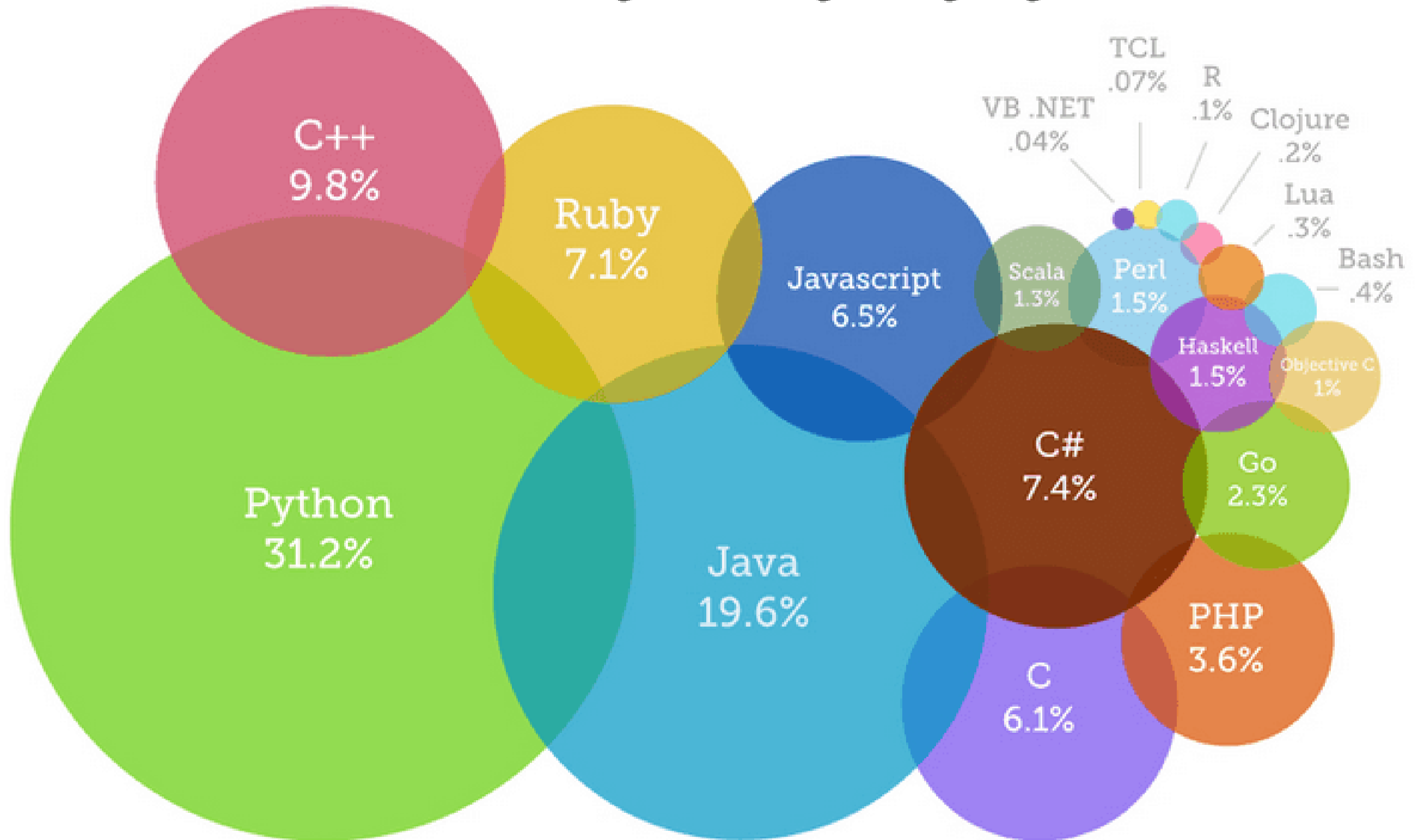
- Imperative languages: (Fortran, C, and Ada) enable programmers to express algorithms for solving problems.
- Declarative languages, (Lisp, Prolog, Haskell) allow the programmer to specify what has to be computed, but not how the computation is done.
- Object Oriented: can be viewed as a hybrid – of declarative (class structures) & imperative (methods) features.



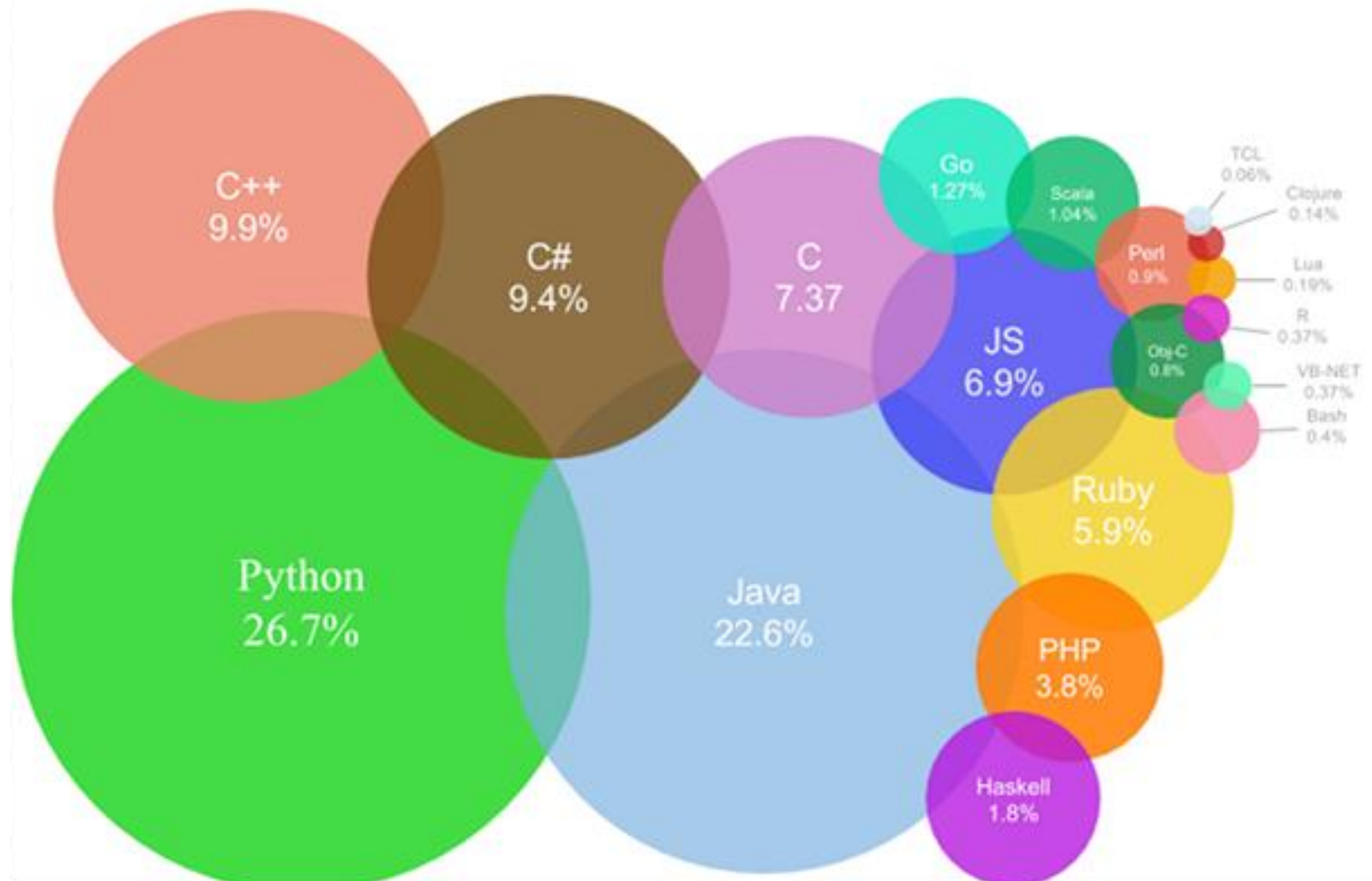
Most Popular Coding Languages of 2014



Most Popular Programming Languages of 2015



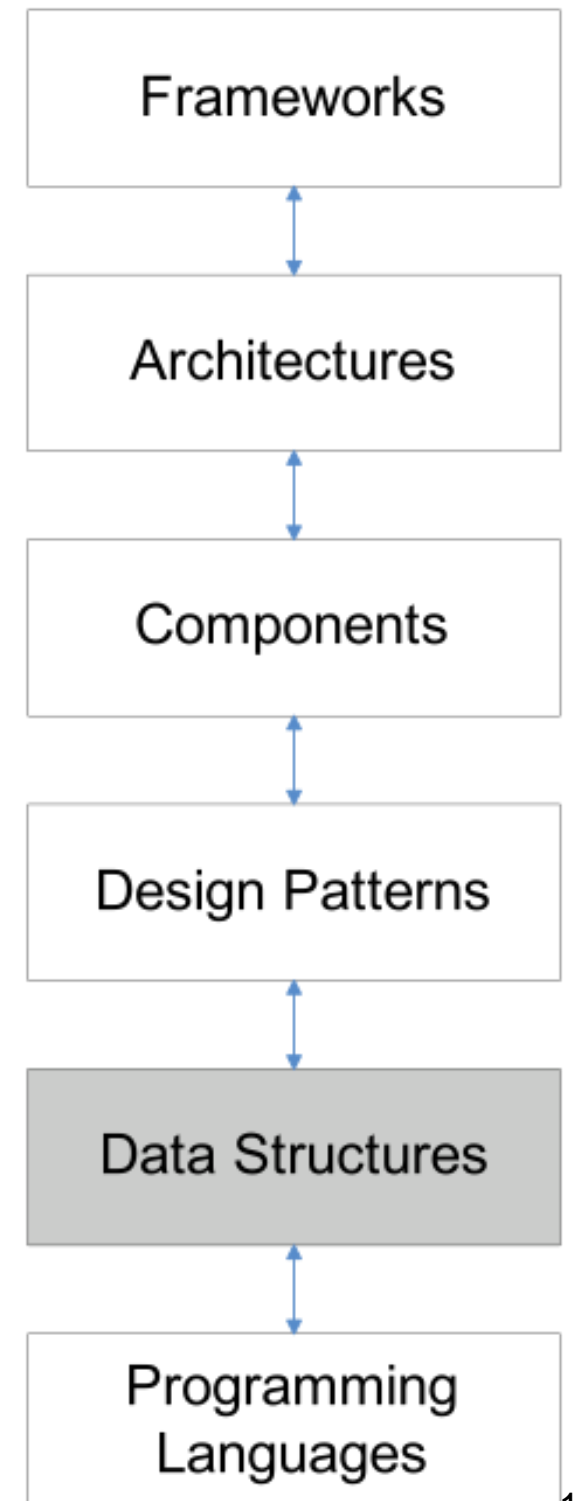
Most Popular Coding Languages of 2016



2015 Rank		2015	Change%	2014	Change%	2013	Change%
1	Python	26.67%	-14.64%	31.24%	3.10%	30.30%	5.21%
2	Java	22.58%	15.37%	19.57%	-11.85%	22.20%	-13.95%
3	C++	9.96%	1.76%	9.79%	-24.70%	13.00%	3.17%
4	C#	9.39%	27.37%	7.37%	47.37%	5.00%	100.00%
5	C	7.37%	21.37%	6.07%	48.14%	4.10%	-16.33%
6	JavaScript	6.88%	6.09%	6.48%	24.66%	5.20%	33.33%
7	Ruby	5.88%	-17.27%	7.11%	-32.90%	10.60%	10.42%
8	PHP	3.82%	5.45%	3.62%	9.84%	3.30%	-54.79%
9	Haskell	1.77%	17.24%	1.51%	25.83%	1.20%	
10	Go	1.27%	-44.00%	2.26%	50.67%	1.50%	-25.00%
11	Scala	1.04%	-17.80%	1.27%	27.00%	1.00%	66.67%
12	Perl	0.95%	-37.33%	1.52%	-6.17%	1.62%	
13	Objective-C	0.82%	-17.62%	1.00%	265.76%	0.27%	173.40%
14	Bash	0.46%	7.21%	0.43%	290.91%	0.11%	
15	R	0.37%	165.71%	0.14%	-30.00%	0.20%	
16	Visual Basic,NET	0.37%	825.50%	0.04%			
17	Lua	0.19%	-44.51%	0.35%	337.50%	0.08%	
18	Clojure	0.14%	-8.53%	0.15%	-48.28%	0.29%	-63.75%
19	Tcl	0.06%	-8.57%	0.07%	133.33%	0.03%	50.00%

Data Structures & Problems

- Typical Data Structures:
 - Lists, Maps, Stacks, Queues, Trees, etc.
 - Static and Dynamic implementations
- Typical Problem Categories:
 - Search
 - Sorting
 - Traversal
 - Inserting / Deleting
 - Merging
 - Clustering
 - Classification



Exploring a Data Structure

```
public class Contact
{
    private String name;

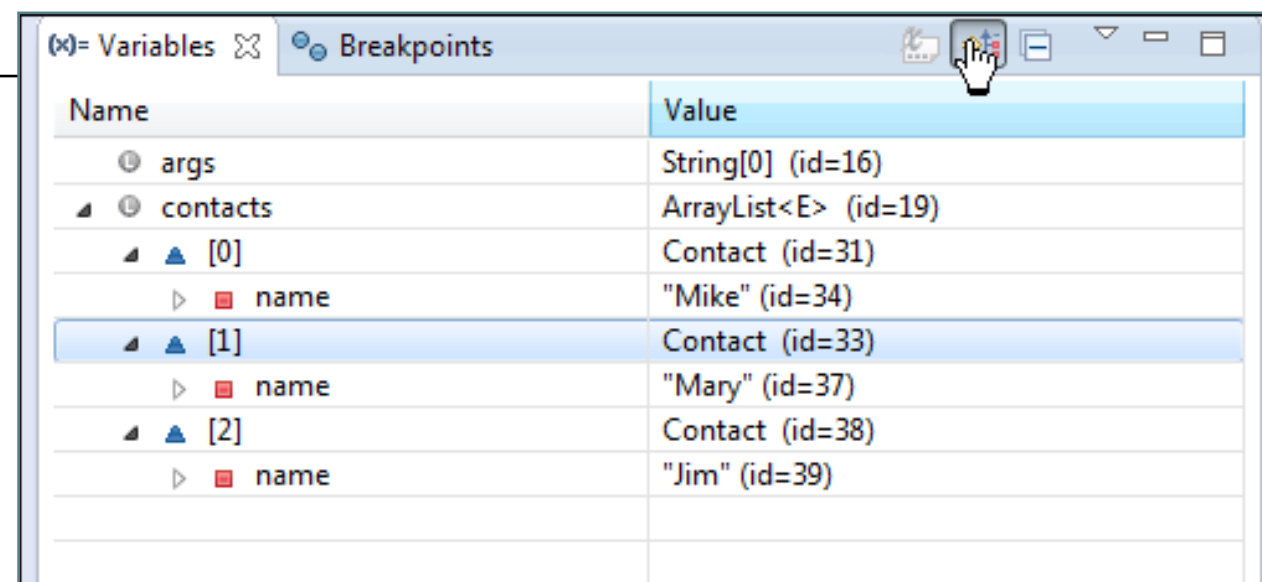
    public Contact(String name)
    {
        this.name = name;
    }

    public String toString()
    {
        return name;
    }
}
```

```
public class Main
{
    public static void main(String[] args)
    {
        List<Contact> contacts = new ArrayList<Contact>();

        contacts.add(new Contact("Mike"));
        contacts.add(new Contact("Mary"));
        contacts.add(new Contact("Jim"));

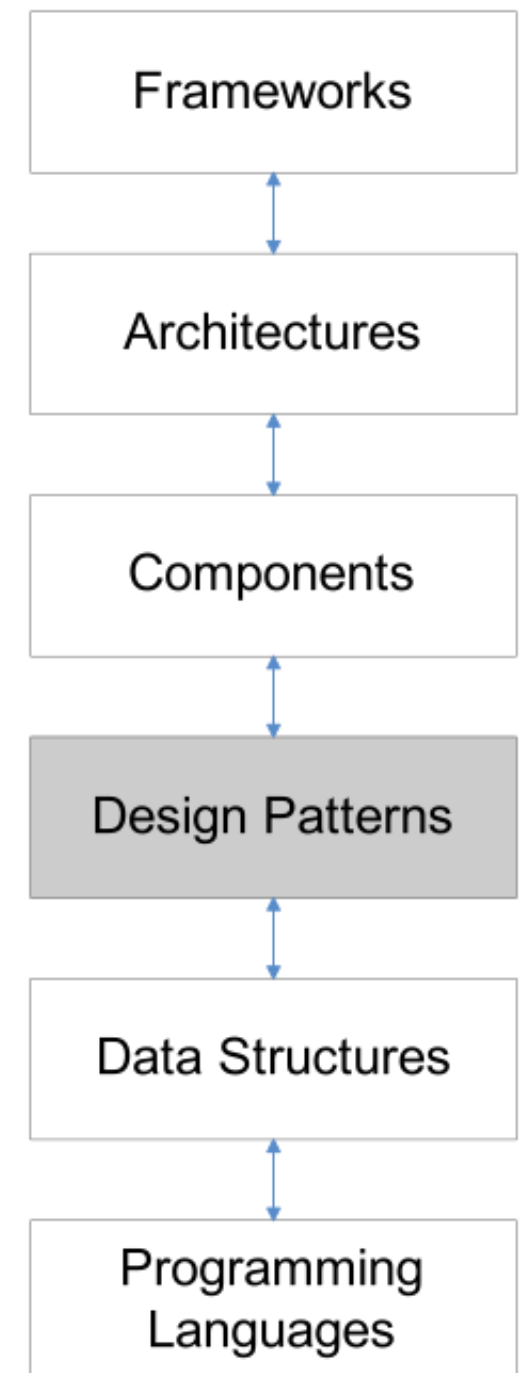
        System.out.println(contacts);
    }
}
```



Name	Value
args	String[0] (id=16)
contacts	ArrayList<E> (id=19)
[0]	Contact (id=31)
name	"Mike" (id=34)
[1]	Contact (id=33)
name	"Mary" (id=37)
[2]	Contact (id=38)
name	"Jim" (id=39)

Design Patterns

- A design pattern is a **proven** solution for a general design problem.
- It consists of communicating 'objects' that are customised to solve the problem in a particular context.
- Patterns have their origin in object-oriented programming; they are pre-packaged Object-oriented design knowledge that allows you to create more flexible and maintainable code.
- There isn't any fundamental relationship between patterns and objects; it just happens they began there.
- Patterns may have arisen because objects seem so elemental, but the problems we were trying to solve with them were so complex.



Why the need for Design Patterns?

- One word...CHANGE! Change is a constant in software design e.g. bugs, new features, changes to design, new regulations, etc. All software changes, so your designs should be ready for it.
- They allow you to typically anticipate common ways that systems grow and change over time.
- The primary goal of any design pattern is to help you structure your code so it is flexible and resilient.
- All patterns let some part of the code vary independently of the other parts.

Pattern Levels

Architectural Patterns:

- Expresses a fundamental structural organization or schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them.

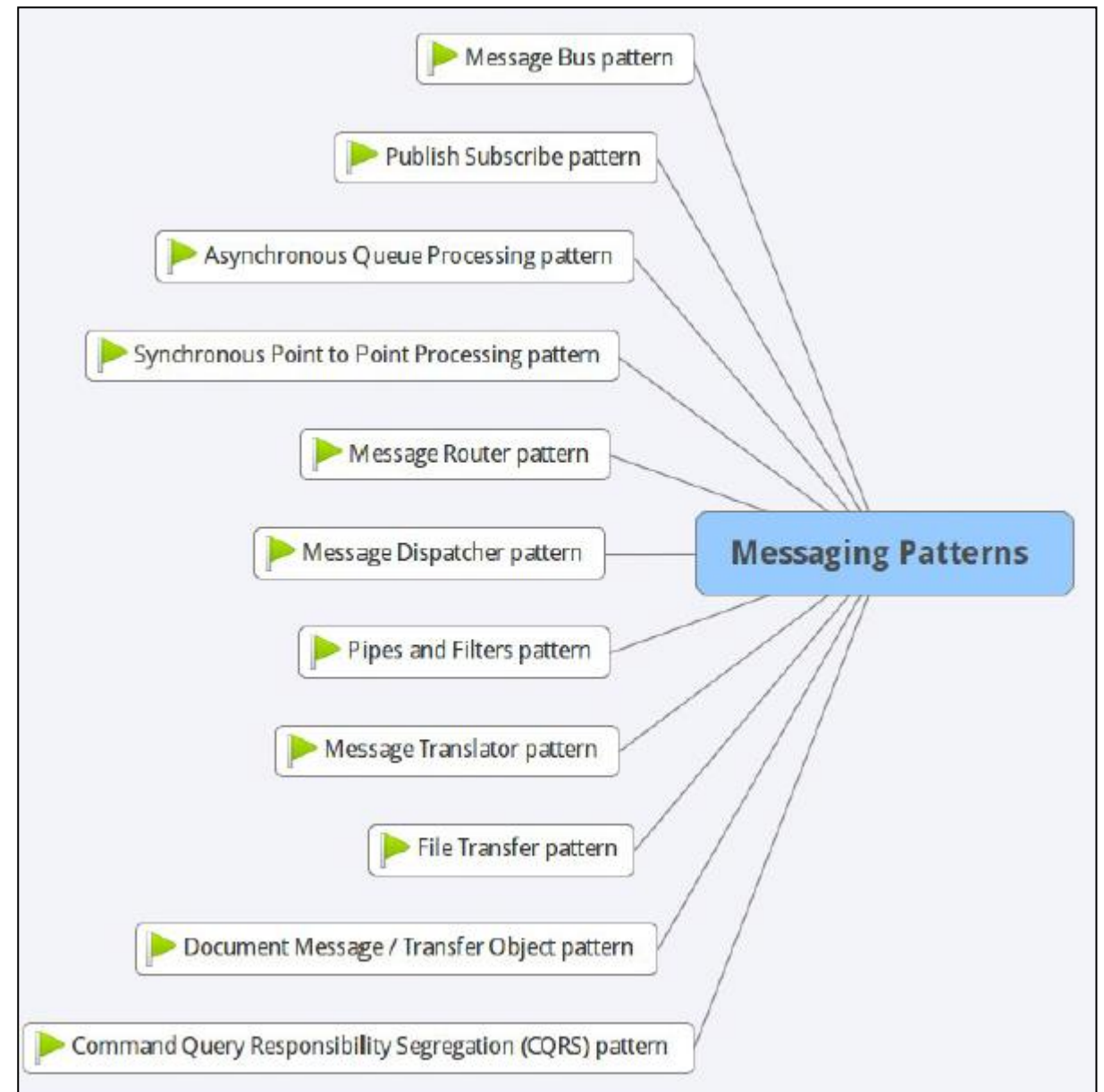
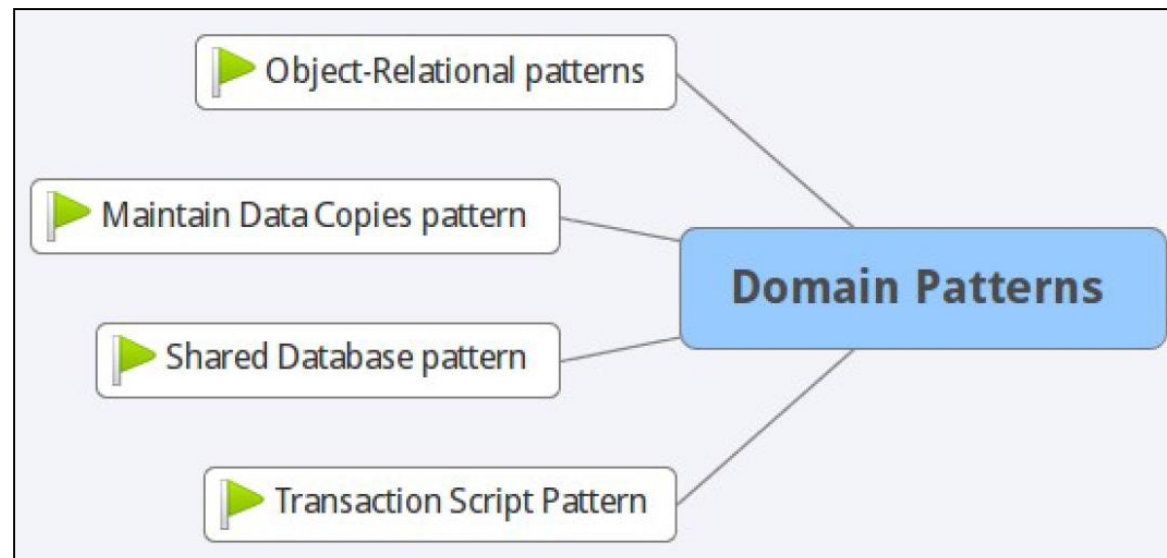
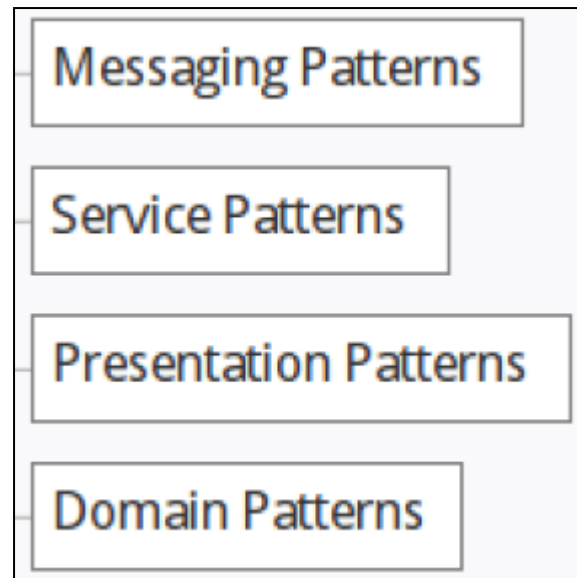
Design Patterns:

- Provides a scheme for refining the subsystems or components of a software system, or the relationships between them. It describes commonly recurring structure of communicating components that solves a general design problem within a particular context.

Idioms:




- A low-level pattern specific to a programming language. An idiom describes how to implement particular aspects of components or the relationships between them using the features of the given language.

Architectural Patterns Examples


























Note: this area is covered in detail in the Design Patterns Module.

Design Patterns Examples

-  **Creational Patterns:** Used to construct objects such that they can be decoupled from their implementing system.
-  **Structural Patterns:** Used to form large object structures between many disparate objects.
-  **Behavioral Patterns:** Used to manage algorithms, relationships, and responsibilities between objects.

Object Scope: Deals with object relationships that can be changed at runtime.

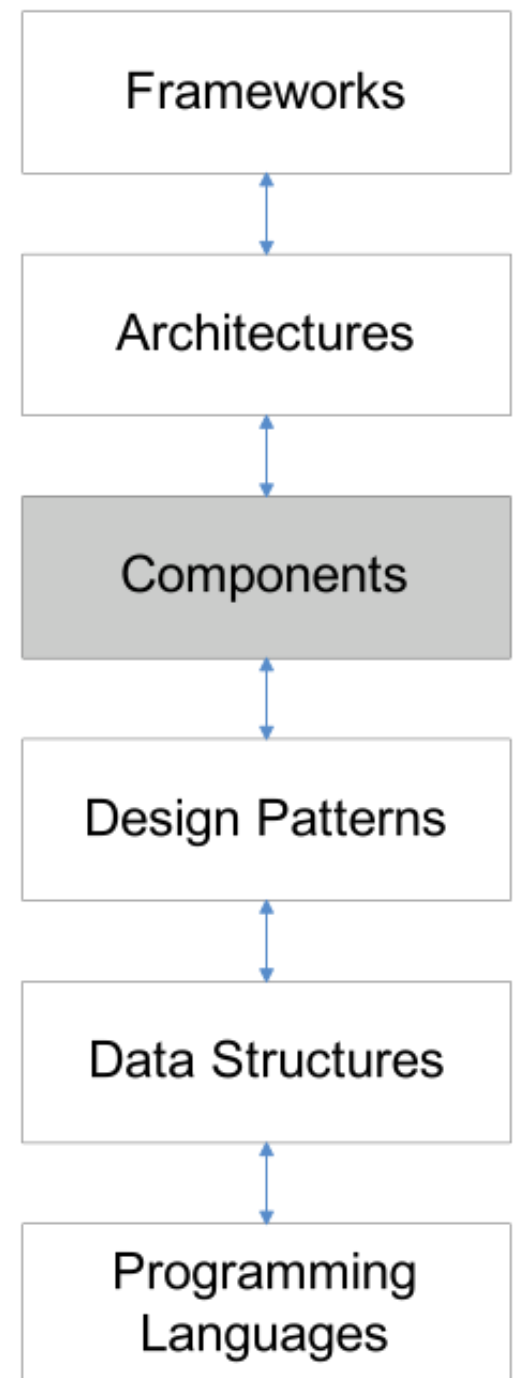
Class Scope: Deals with class relationships that can be changed at compile time.

 Abstract Factory	 Decorator	 Prototype
 Adapter	 Facade	 Proxy
 Bridge	 Factory Method	 Observer
 Builder	 Flyweight	 Singleton
 Chain of Responsibility	 Interpreter	 State
 Command	 Iterator	 Strategy
 Composite	 Mediator	 Template Method
	 Memento	 Visitor

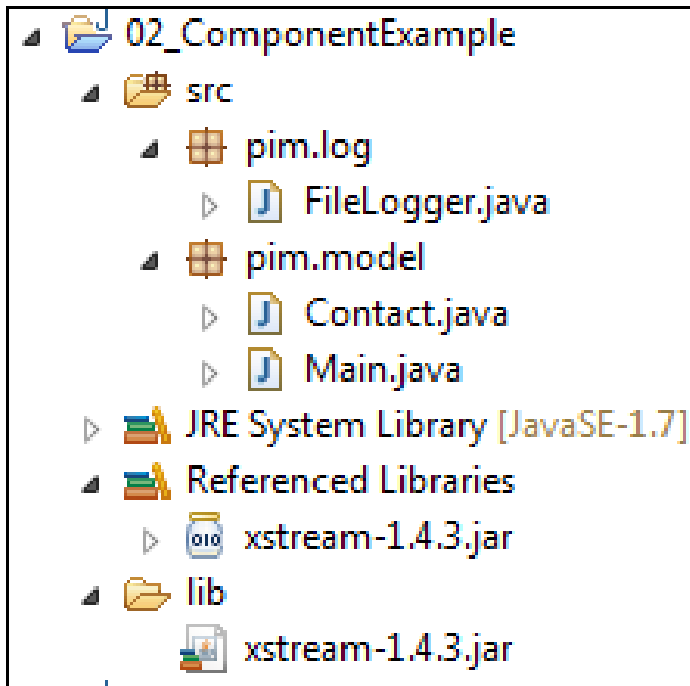
Note: this area is covered in detail in the Design Patterns Module.

Components

- Software components are binary units of:
 - independent production,
 - acquisition,
 - deployment
- that interact to form a functioning program.
(Szyperski, 1998)
- Emphasis on reusable units.
- A component must be compatible and interoperate with a whole range of other components.
- Two main issues arise with respect to interoperability information:
 - How to express interoperability information
 - How to publish this information



Exploring a Component (xstream-1-4-3.jar)



```
<object-stream>
<list>
  <pim.model.Contact>
    <name>Mike</name>
  </pim.model.Contact>
  <pim.model.Contact>
    <name>Mary</name>
  </pim.model.Contact>
  <pim.model.Contact>
    <name>Jim</name>
  </pim.model.Contact>
</list>
</object-stream>
```

```
public class Main{

    public static void main(String[] args) throws IOException{
        FileLogger logger = FileLogger.getLogger();

        logger.log("Creating contact list");

        List<Contact> contacts = new ArrayList<Contact>();
        logger.log("Adding contacts");
        contacts.add(new Contact("Mike"));
        contacts.add(new Contact("Mary"));
        contacts.add(new Contact("Jim"));
        System.out.println(contacts);

        logger.log("Serializing contacts to XML");
        XStream xstream = new XStream(new DomDriver());
        ObjectOutputStream out = xstream.createObjectOutputStream
            (new FileWriter("contacts.xml"));
        out.writeObject(contacts);
        out.close();

        logger.log("Finished - shutting down");
    }
}
```



```
public class FileLogger{

    private static FileLogger logger;

    private FileLogger(){
    }

    public static FileLogger getLogger(){
        if (logger == null){
            logger = new FileLogger();
        }
        return logger;
    }

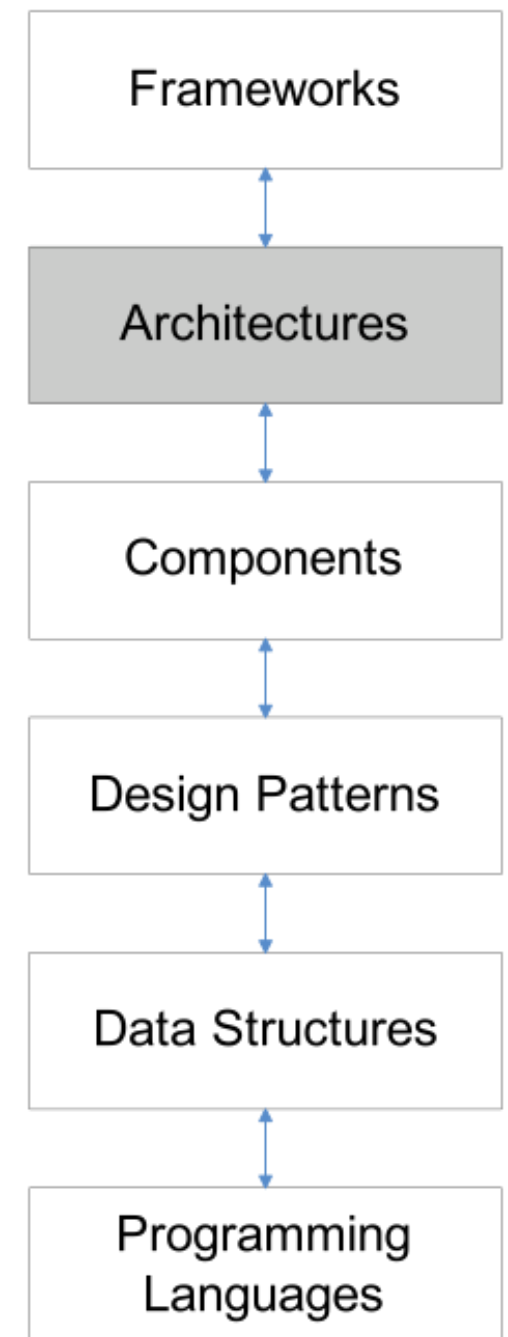
    public boolean log(String msg){
        try{
            PrintWriter writer = new PrintWriter(new FileWriter("log.txt", true));
            writer.println(msg);
            writer.close();
        }
        catch (FileNotFoundException ex){
            return (false);
        }
        catch (IOException ex){
            return (false);
        }
        return (true);
    }
}
```

More Component Definitions

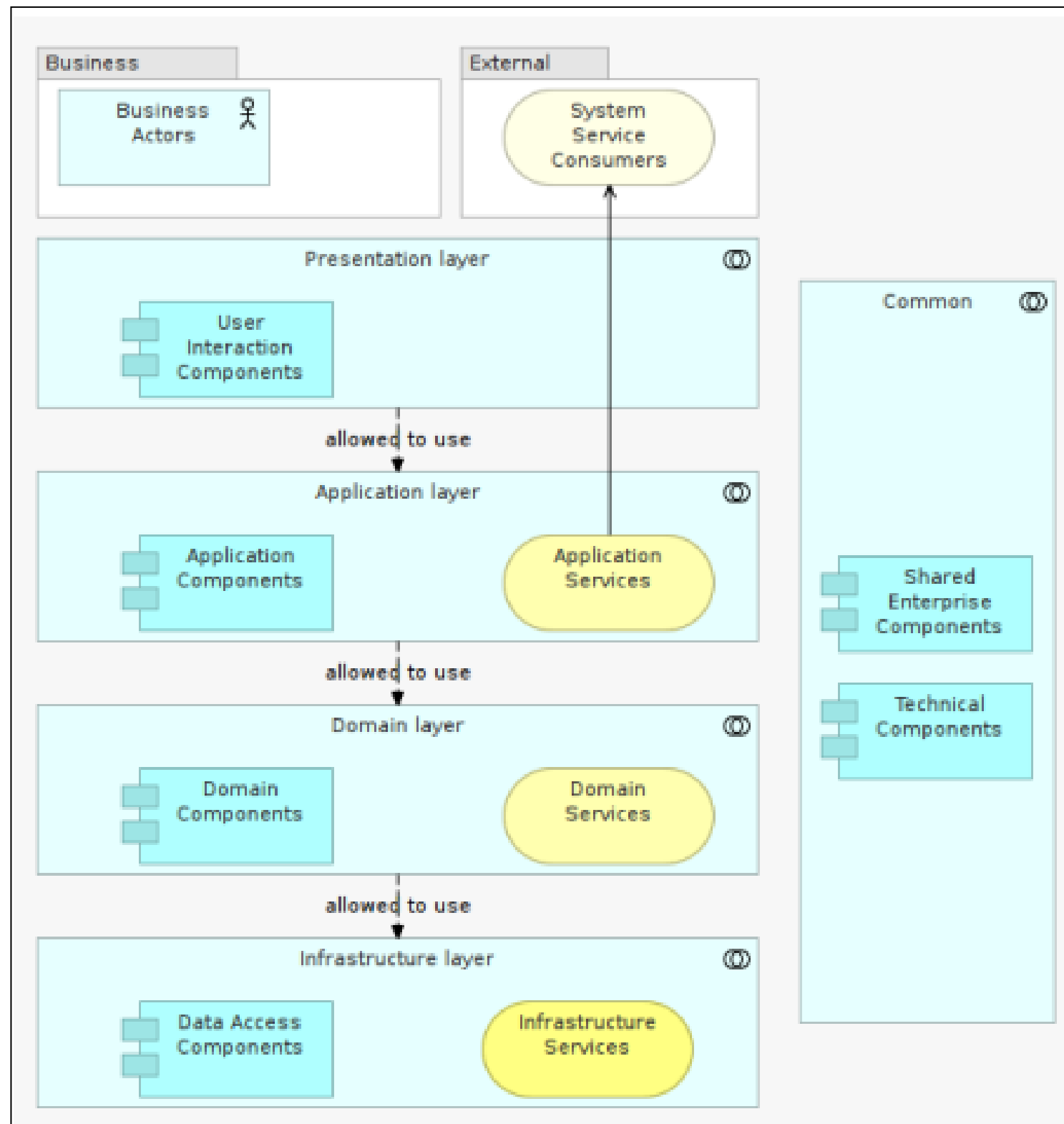
- "A component is a nontrivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture. A component conforms to and provides the physical realization of a set of interfaces." (Philippe Krutchen, Rational Software)
- "A runtime software component is a dynamically bindable package of one or more programs managed as a unit and accessed through documented interfaces that can be discovered at runtime." (Gartner Group)
- "A component is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces...typically represents the physical packaging of otherwise logical elements, such as classes, interfaces, and collaborations." (Grady Booch, Jim Rumbaugh, Ivar Jacobson, The UML User Guide, p. 343)

Architecture

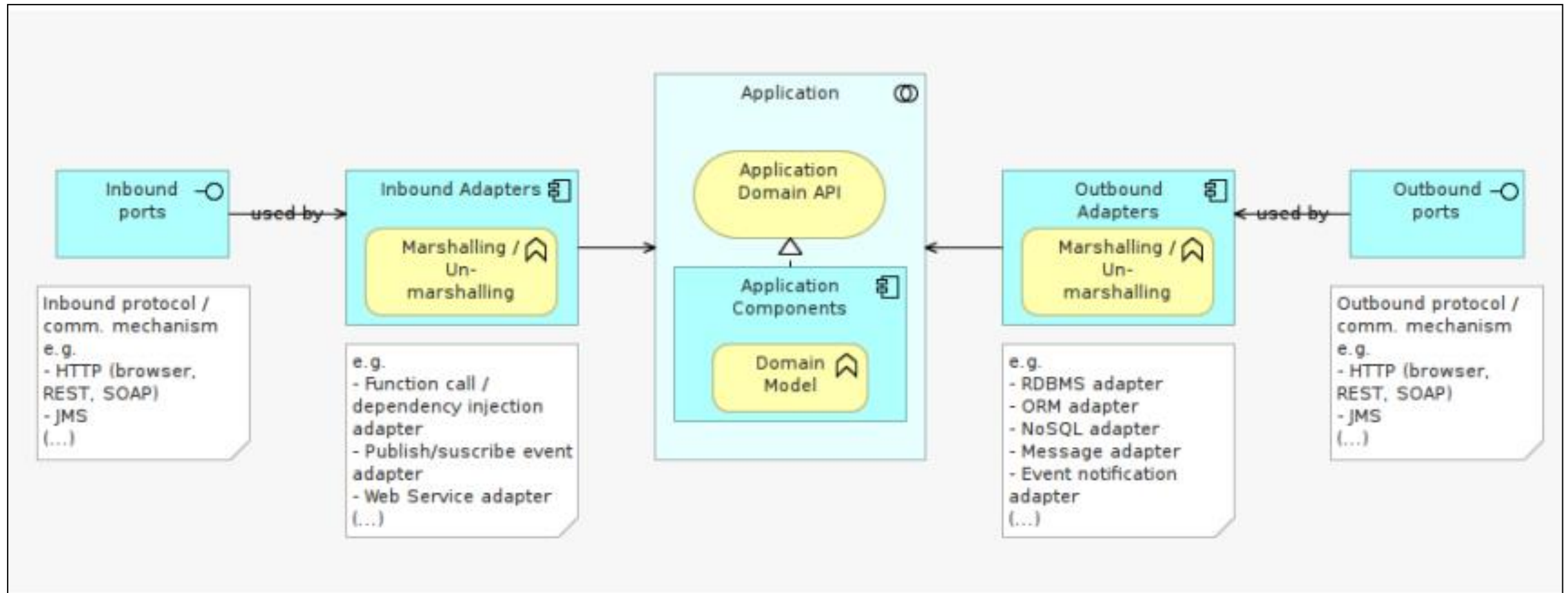
- The software architecture of a program or computing system is:
 - the structure or structures of the system, which comprise software components,
 - the externally visible properties of those components, and
 - the a set of rules that govern relationships among them.
- An architectural style is a family of software architectures, defining types of components and types of connections, and rules describing how to combine them.
- A software architecture is an instantiation of an architectural style for a certain system. The components and connections may be decomposed into architectures themselves.



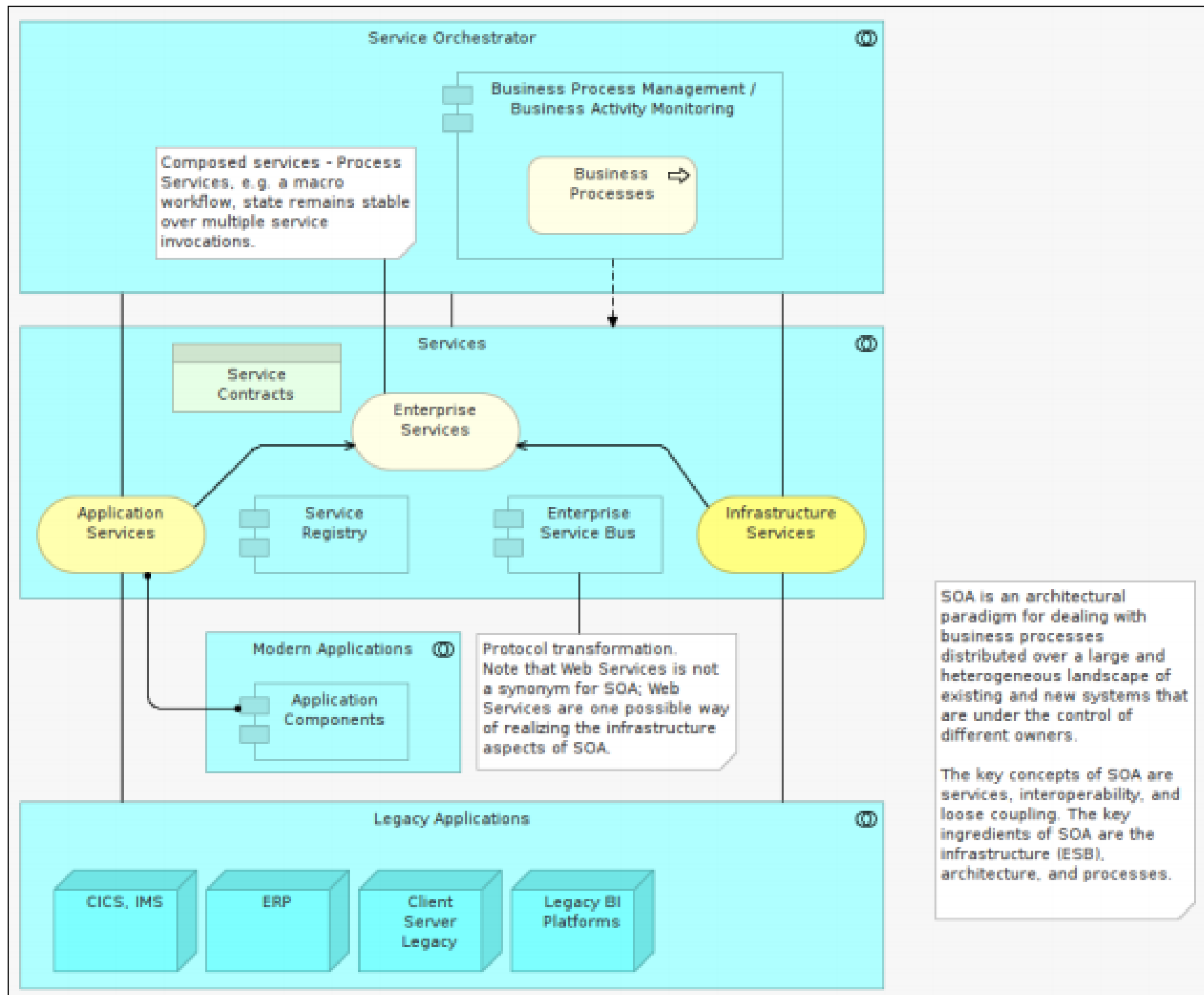
Architecture Example: Layered



Architecture Example: Ports and Adaptors

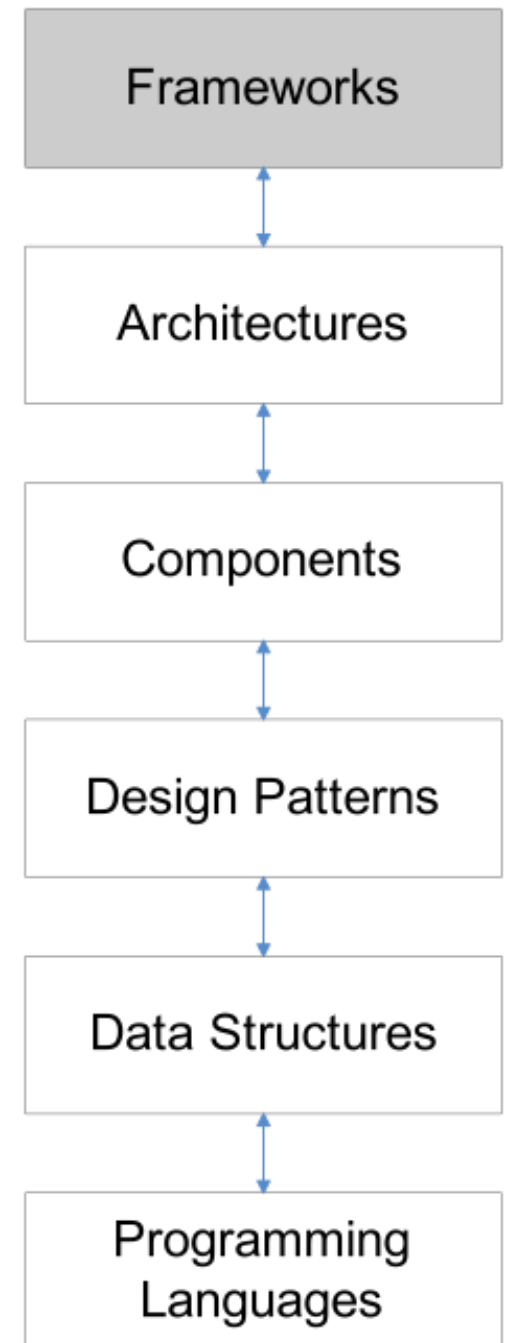


Architecture Example: Service Oriented



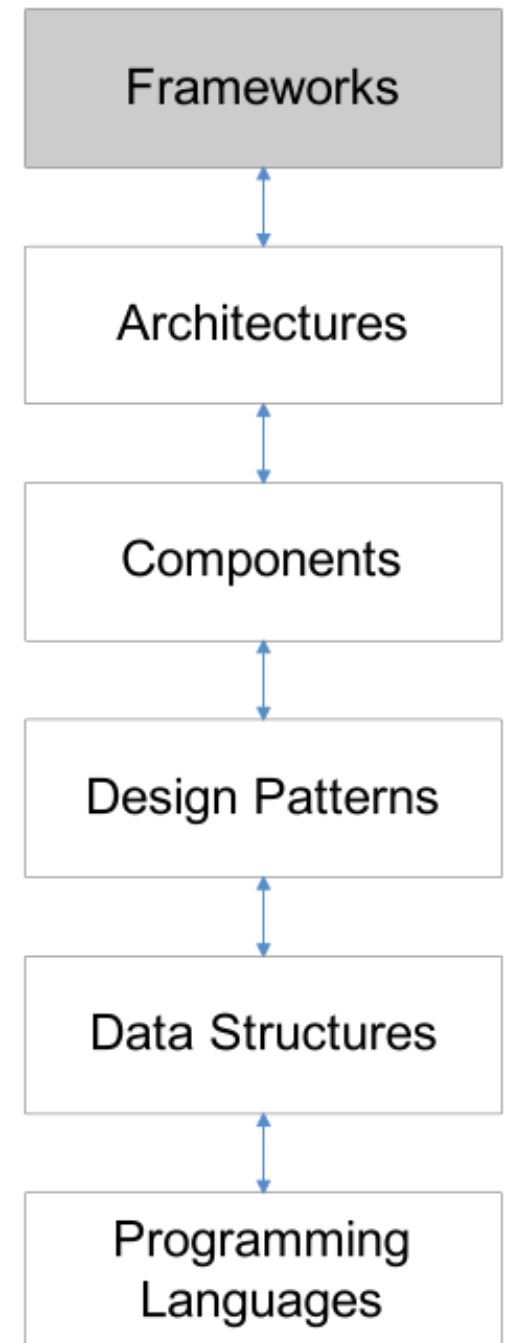
Frameworks

- A framework is a set of related components which you:
 - Specialize
 - Integrate and/or
 - Instantiate
- to implement an application or subsystem.
- A framework is usually, a semi complete application containing dynamic and static components that can be customized to produce applications.

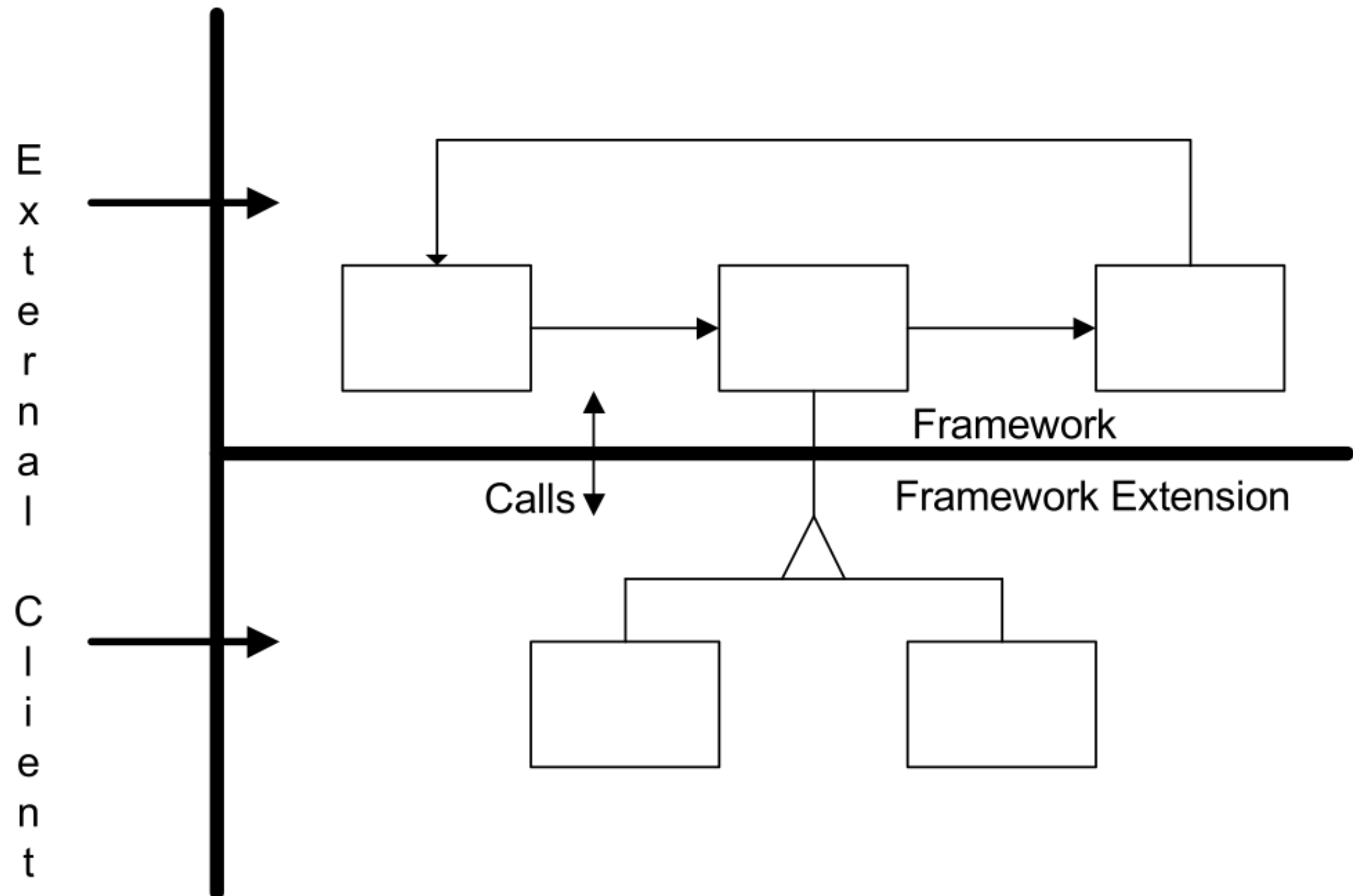


Frameworks

- Frameworks are targeted for a particular application domain & consists of a set of classes (abstract & concrete), whose instances:
 - collaborate
 - are intended to be extended, i.e. reused (abstract design)
 - do not have to address a complete application domain (allowing for composition of frameworks)
- Emphasize stable parts of the domain and their relationships and interactions.



Framework Structure



Framework Example

```
▼ > spacebook-p2 [spacebook-p2 master]
  ▼ app
    ▶ (default package)
    ▶ controllers
    ▶ models
    ▶ views
    ▶ views.nav
  ▼ test
    ▼ (default package)
      ▶ ApplicationTest.java
      ▶ FriendsTest.java
      ▶ IntegrationTest.java
      ▶ MessagesTest.java
      ▶ UsersTest.java
    ▶ Referenced Libraries
    ▶ JRE System Library [Java SE 7 [1.7.0_21]]
    ▶ > conf
    ▶ logs
    ▶ > project
    ▶ public
    ▶ target
    README
```

[Download](#)[Documentation](#)[Get Involved](#)





The High Velocity Web Framework For Java and Scala

Introduction to Play Framework for Java developers



19:28 HD vimeo

GET THE LATEST PACKAGE

[Download 2.1.4](#)

or [browse all versions](#)

GETTING STARTED WITH

[Java & Scala](#)

or [read full documentation](#)

Play Framework makes it easy to build web applications with Java & Scala.

Play is based on a lightweight, stateless, web-friendly architecture.

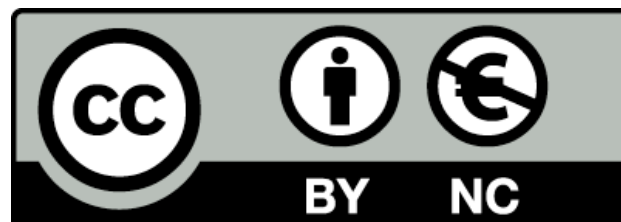
Built on Akka, Play provides predictable and minimal resource consumption (CPU, memory, threads) for highly-scalable applications.

Agile Software Development Module

- Assumptions:
 - General Programming Ability (not necessarily java)
- Focus for this course:
 - SOLID Principles within OO Programming
 - Test Driven Software Development in Java
 - Effective Build Processes
 - Network Programming
 - Future and Beyond Java

References

- Kernighan B. & Plauger P. (1978). The Elements of Programming Style, 2nd ed., New York: McGraw-Hill.
- Meyer, B. (1998). Object-Oriented Software Construction, Prentice-Hall, Englewood Cliffs, NJ.
- Roy, P.V. (2009). Programming Paradigms for Dummies: What Every Programmer Should Know. In G. Assayag and A. Gerzso (eds.), *New Computational Paradigms for Computer Music*, IRCAM/Delatour, France.
- Szyperski, C. (1998), Component Software: Beyond Object-Oriented Programming, Addison Wesley.



Except where otherwise noted, this content is licensed under a [Creative Commons Attribution-NonCommercial 3.0 License](http://creativecommons.org/licenses/by-nc/3.0/).

For more information, please see <http://creativecommons.org/licenses/by-nc/3.0/>

