Introduction: Overview of Patterns & Frameworks (Part 1)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

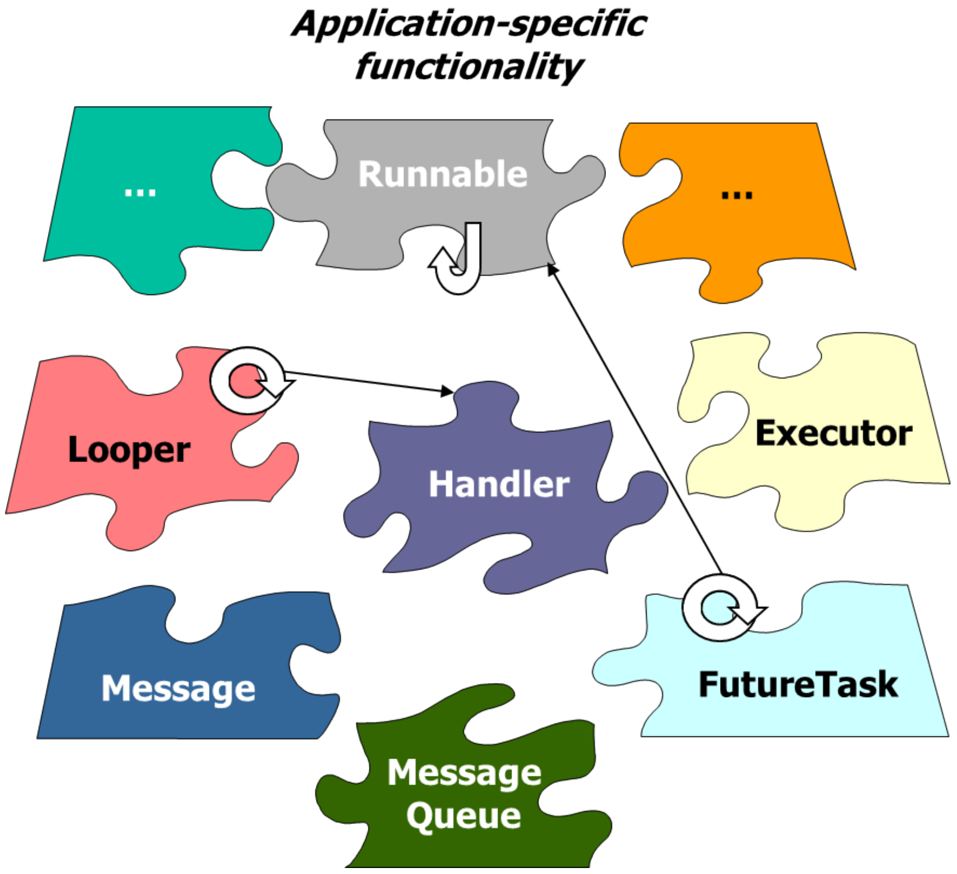
Professor of Computer Science

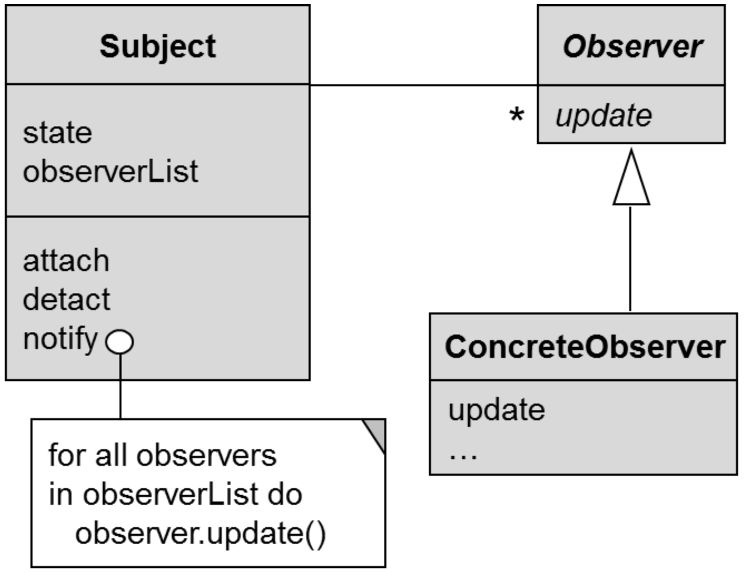
Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Module

- Understand what *patterns* & *frameworks* are & why they are essential for programming mobile device software

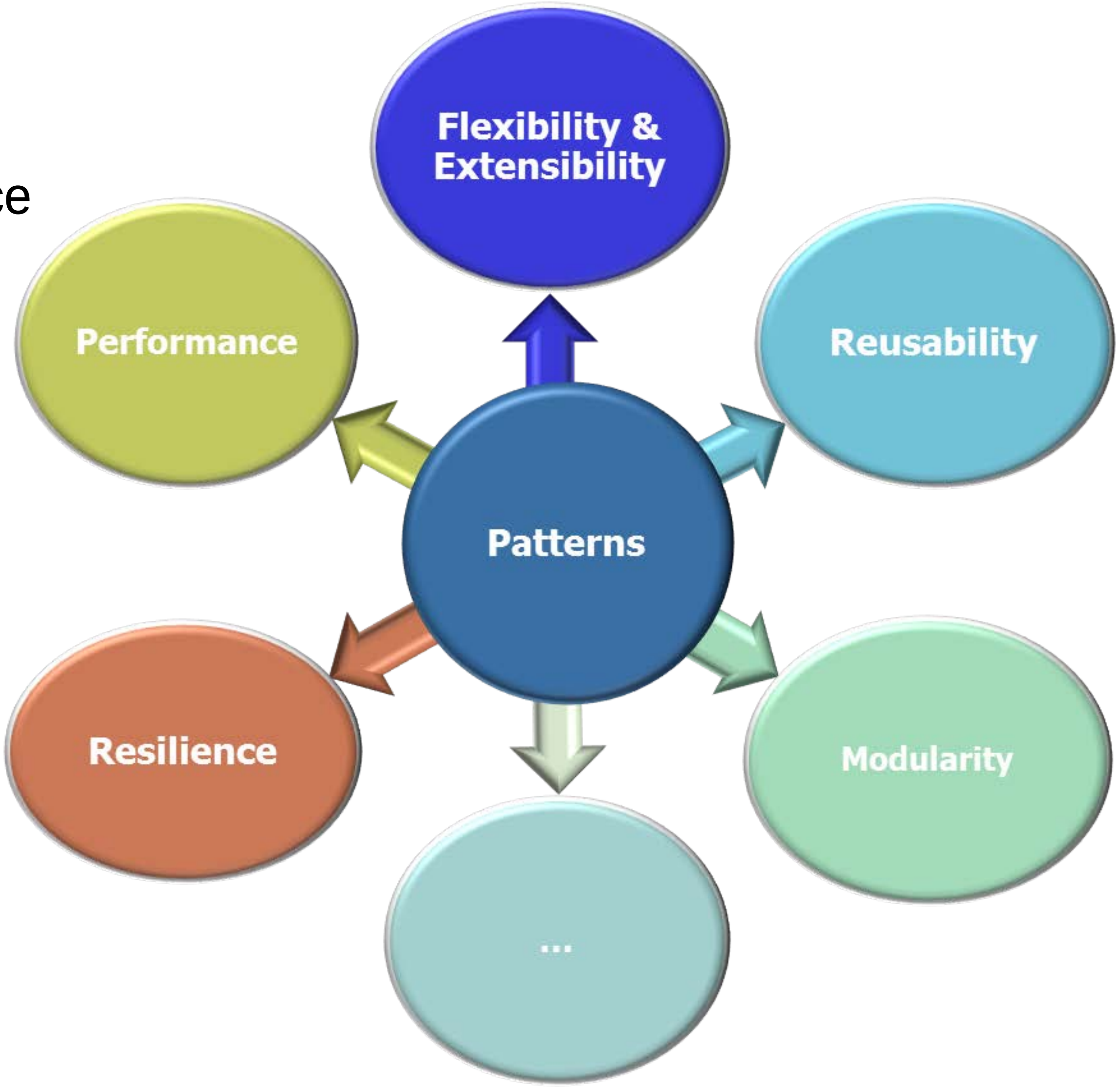




The *Observer* pattern

Motivation for Software Patterns & Frameworks

- Patterns & frameworks leverage proven design & implementation experience to enhance key software quality attributes



www.sei.cmu.edu/architecture/start/reasoning.cfm has more on quality attributes

Leveraging Experience in Other Domains

- Experts perform differently than beginners

Leveraging Experience in Other Domains

- Experts perform differently than beginners
  - Unlike novices, professional athletes, musicians, & dancers move fluidly & effortlessly, without focusing on each individual movement

Leveraging Experience in Other Domains

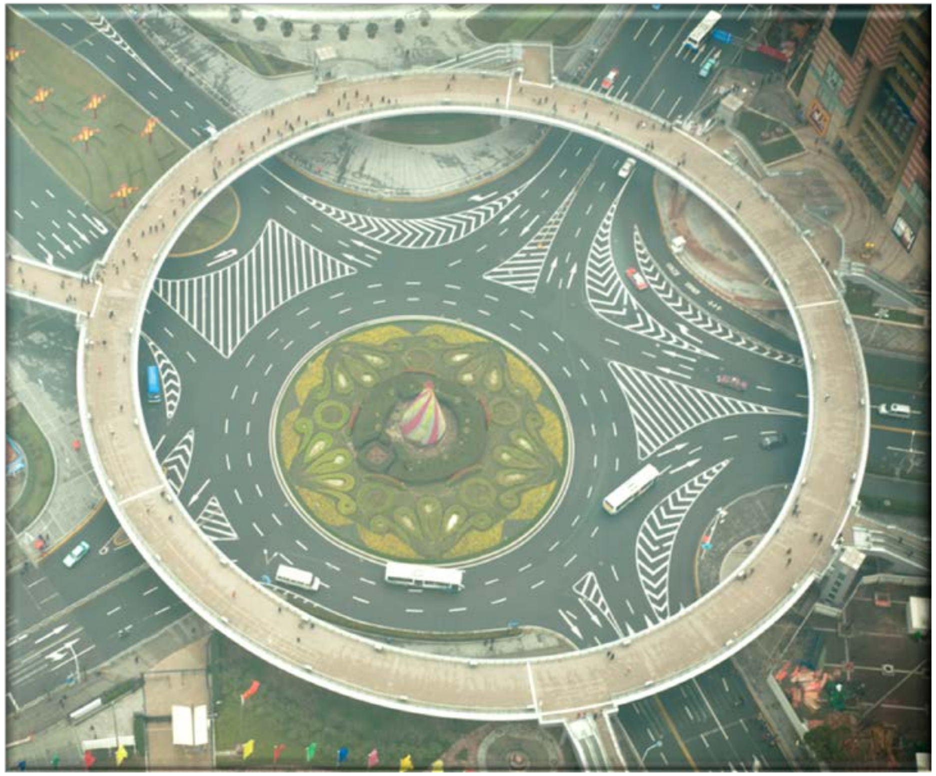
- Experts perform differently than beginners
- When watching experts perform it’s easy to forget how much effort they’ve put into reaching high levels of achievement

Leveraging Experience in Other Domains

- Experts perform differently than beginners
- When watching experts perform it's easy to forget how much effort they've put into reaching high levels of achievement
- Continuous repetition, practice, & mentoring from other experts are crucial to their success

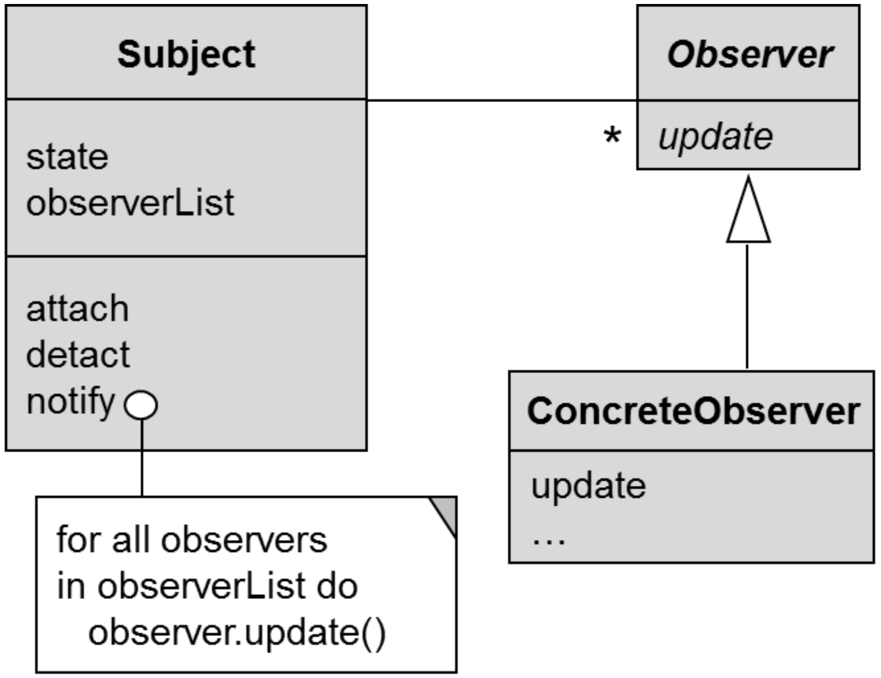
Leveraging Software Experience via Patterns

- Patterns provide reusable solutions to common problems arising within a context



Leveraging Software Experience via Patterns

- Patterns provide reusable solutions to common problems arising within a context

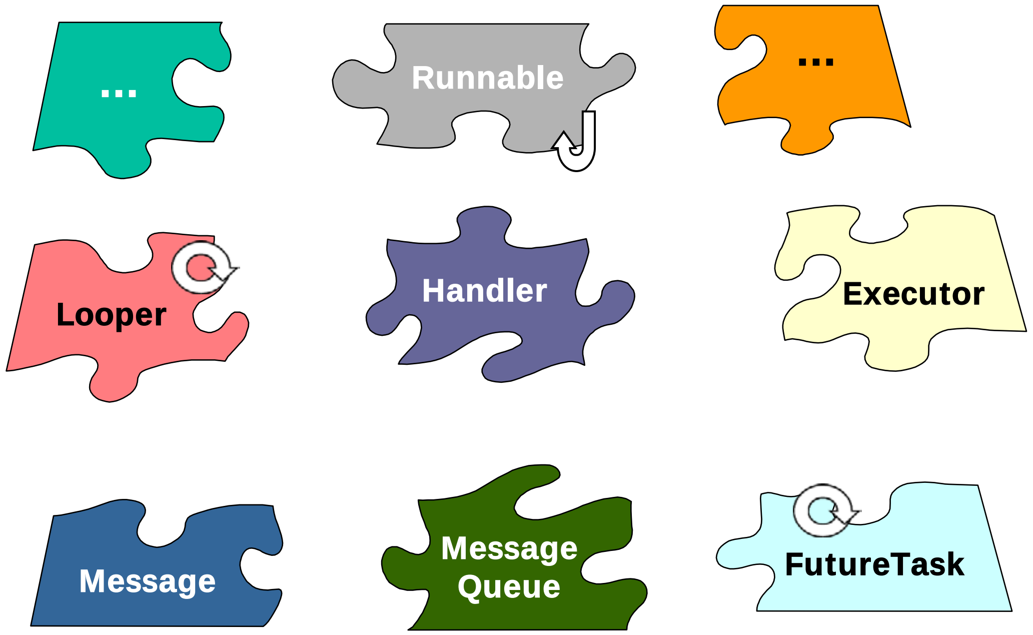


The *Observer* pattern

Patterns provide *design* & *architecture* guidance

Leveraging Software Experience via Frameworks

- A frameworks is an integrated set of components that collaborate to provide a reusable architecture for a family of related applications or services



Frameworks provide *implementation* guidance

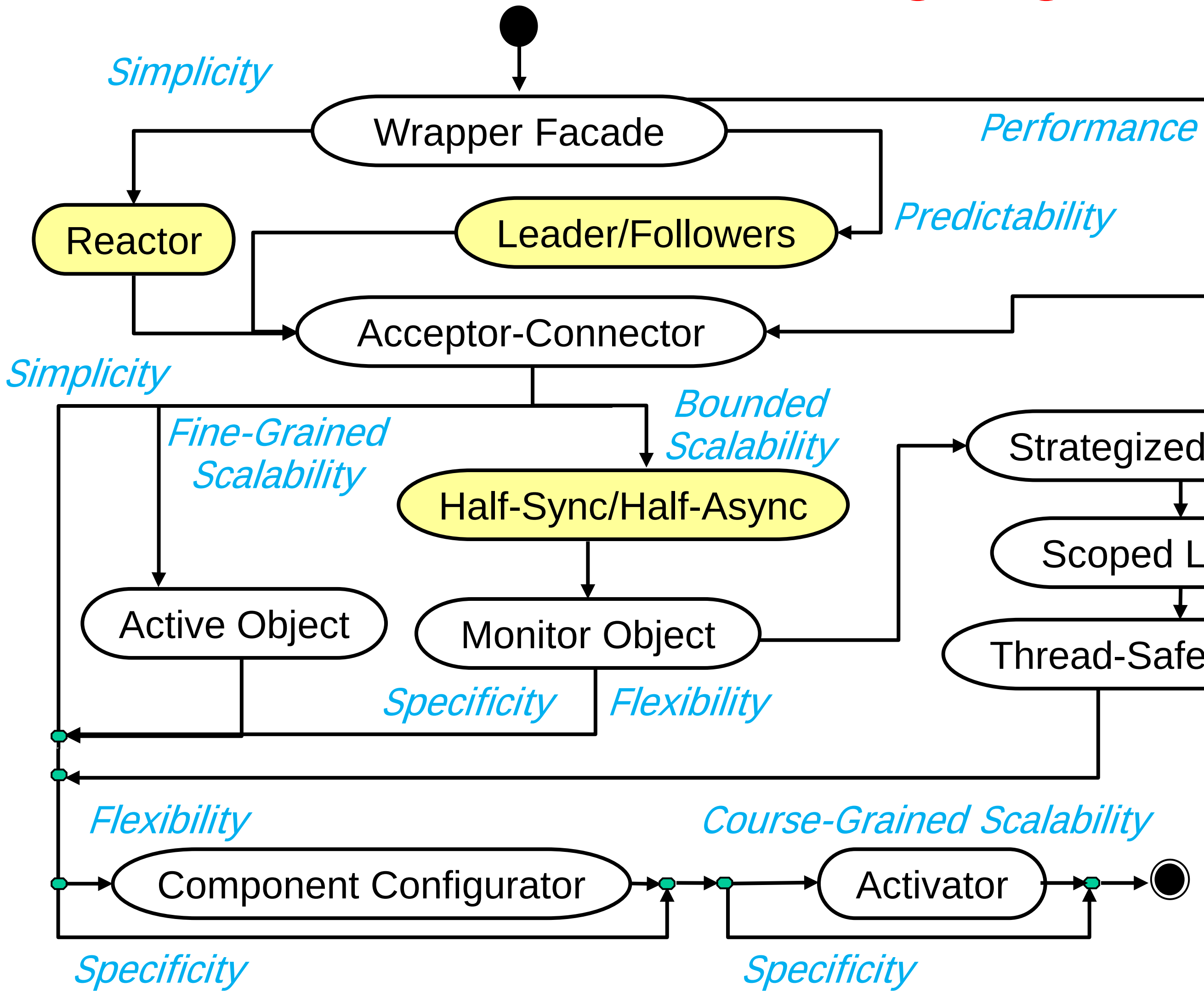
Frameworks & Patterns are Synergistic

- Frameworks are concrete realizations of patterns that facilitate direct reuse of detail design & source code

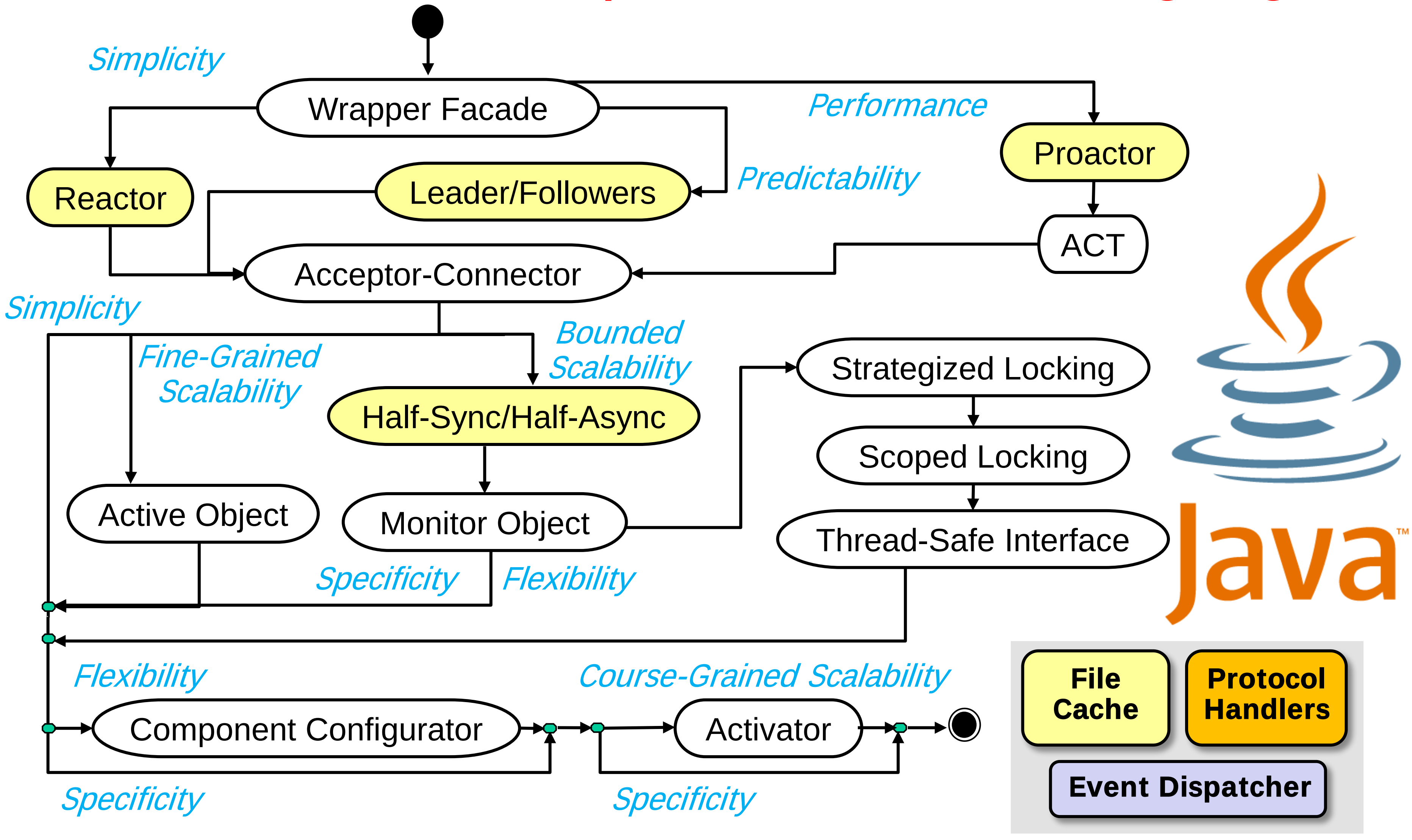
Frameworks & Patterns are Synergistic

- Frameworks are concrete realizations of patterns that facilitate direct reuse of detail design & source code
- Patterns are abstract descriptions of frameworks that facilitate reuse of software architecture & design knowledge

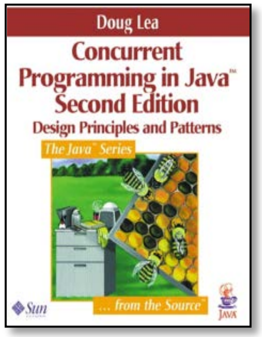
Patterns are Often Language Independent

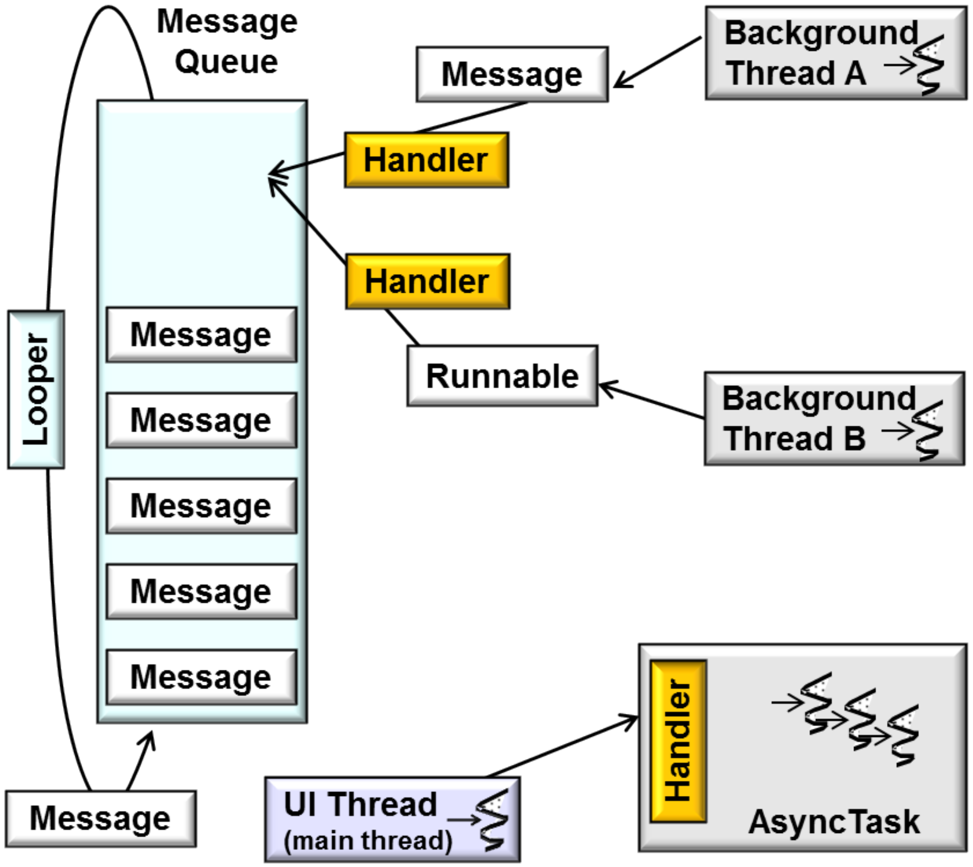


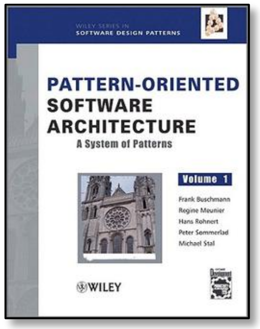
Frameworks are Implemented in a Language



Android Uses Patterns & Frameworks Heavily


Doug Lea
Concurrent Programming in Java Second Edition
Design Principles and Patterns
The Java Series
... from the Source





PATTERN-ORIENTED SOFTWARE ARCHITECTURE
Volume 2
Patterns for Concurrent and Networked Objects
WILEY

Overview of Patterns

Overview of Patterns

- Present solutions to common problems arising within a context

Electronic Trading

Aerospace & Avionics

Mobile Devices

e-commerce

Automotive

Overview of Patterns

- Present solutions to common problems arising within a context

Electronic Trading

Aerospace & Avionics

Mobile Devices

Automotive

e-commerce

Overview of Patterns

- Present solutions to common problems arising within a context

Electronic Trading

Aerospace & Avionics

Mobile Devices

Automotive

e-commerce

Overview of Patterns

- Present solutions to common problems arising within a context

Electronic Trading

Aerospace & Avionics

Mobile Devices

Automotive

e-commerce

Overview of Patterns

- Present solutions to common problems arising within a context

Electronic Trading

Aerospace & Avionics

Mobile Devices

e-commerce

Automotive

Overview of Patterns

- Present solutions to common problems arising within a context

Electronic Trading

Aerospace & Avionics

Mobile Devices

Automotive

e-commerce

Overview of Patterns

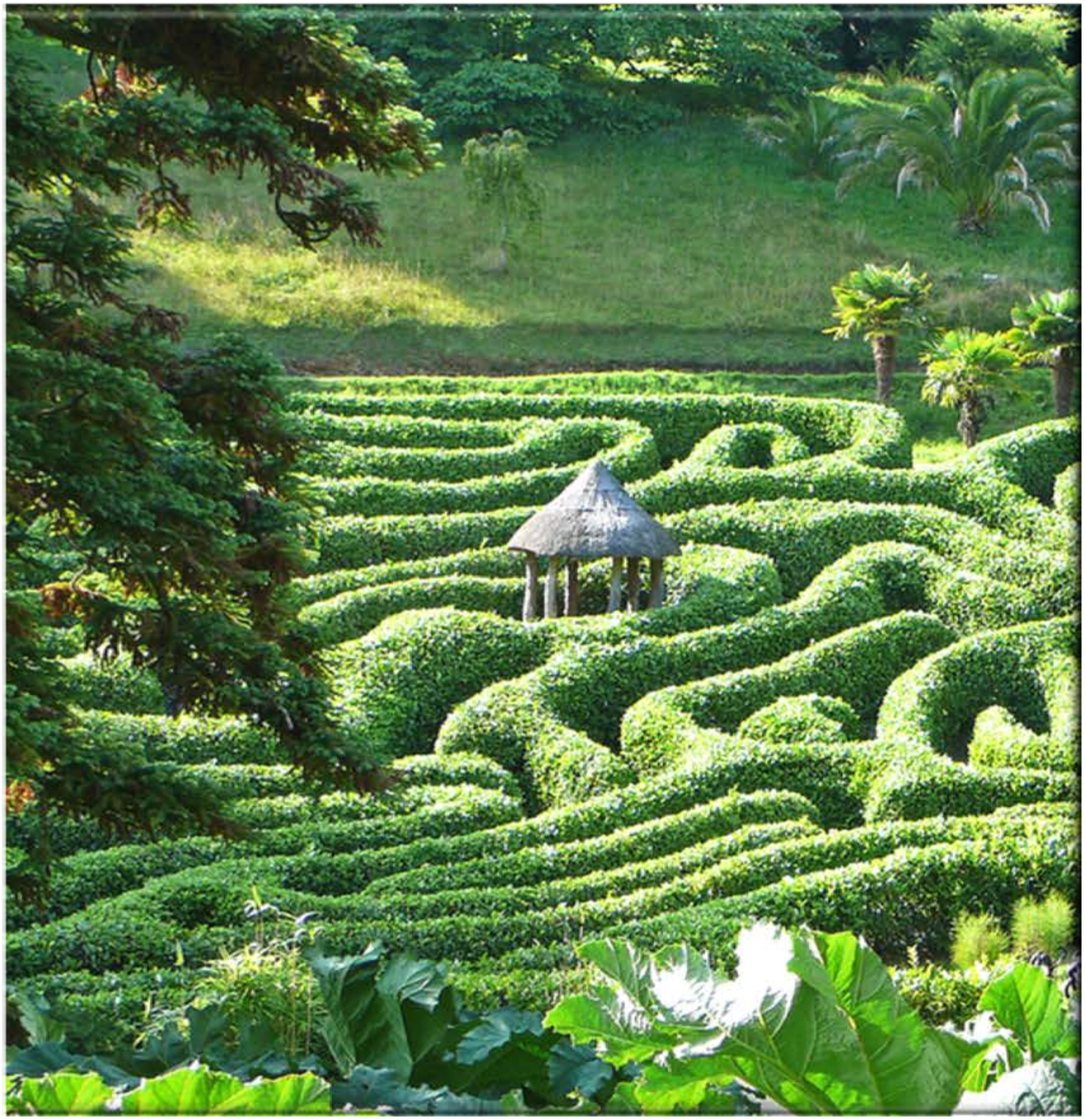
- Present solutions to common problems arising within a context
- Help resolve key software quality attribute forces

Quality of Service
Quality of Design

Software quality attributes must be balanced & traded-off against various forces

Overview of Patterns

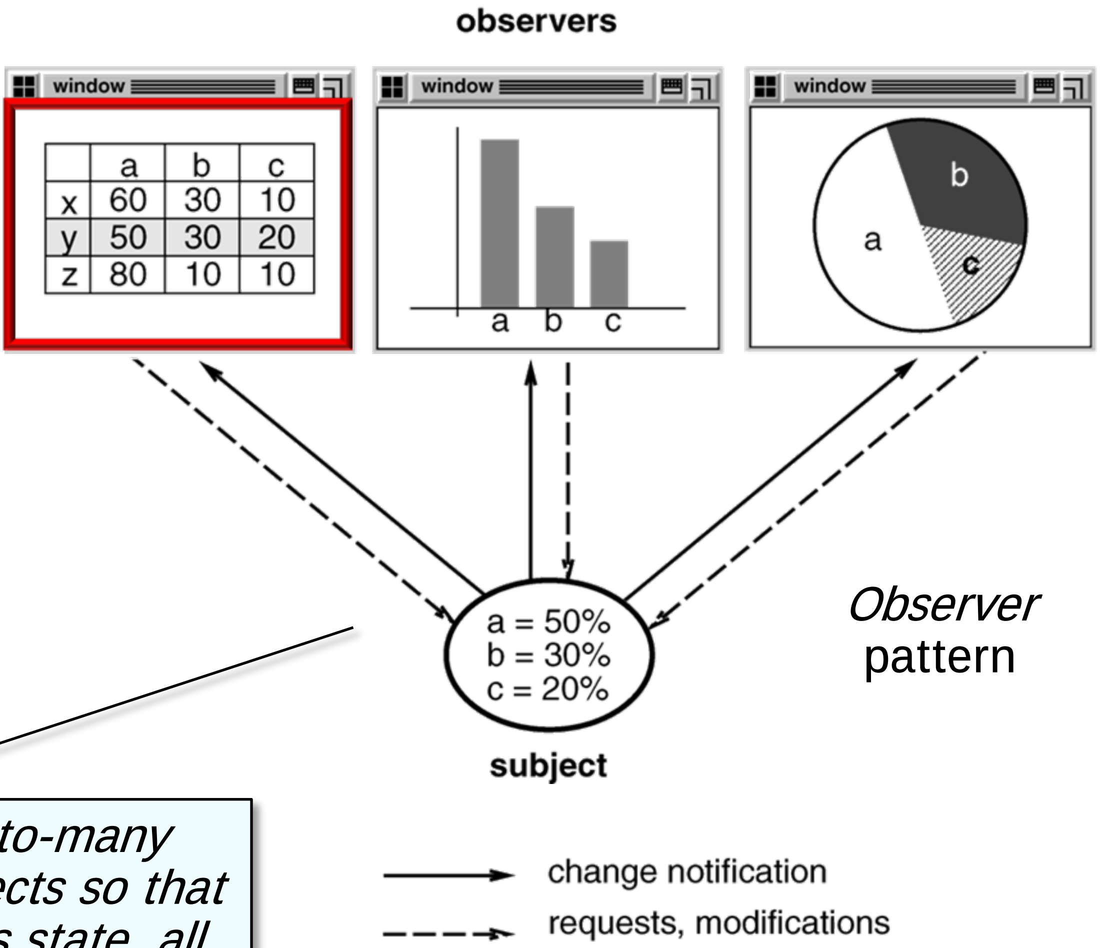
- Present solutions to common problems arising within a context
- Help resolve key software quality attribute forces



Patterns help developers navigate through trade-offs in domains & design spaces

Overview of Patterns

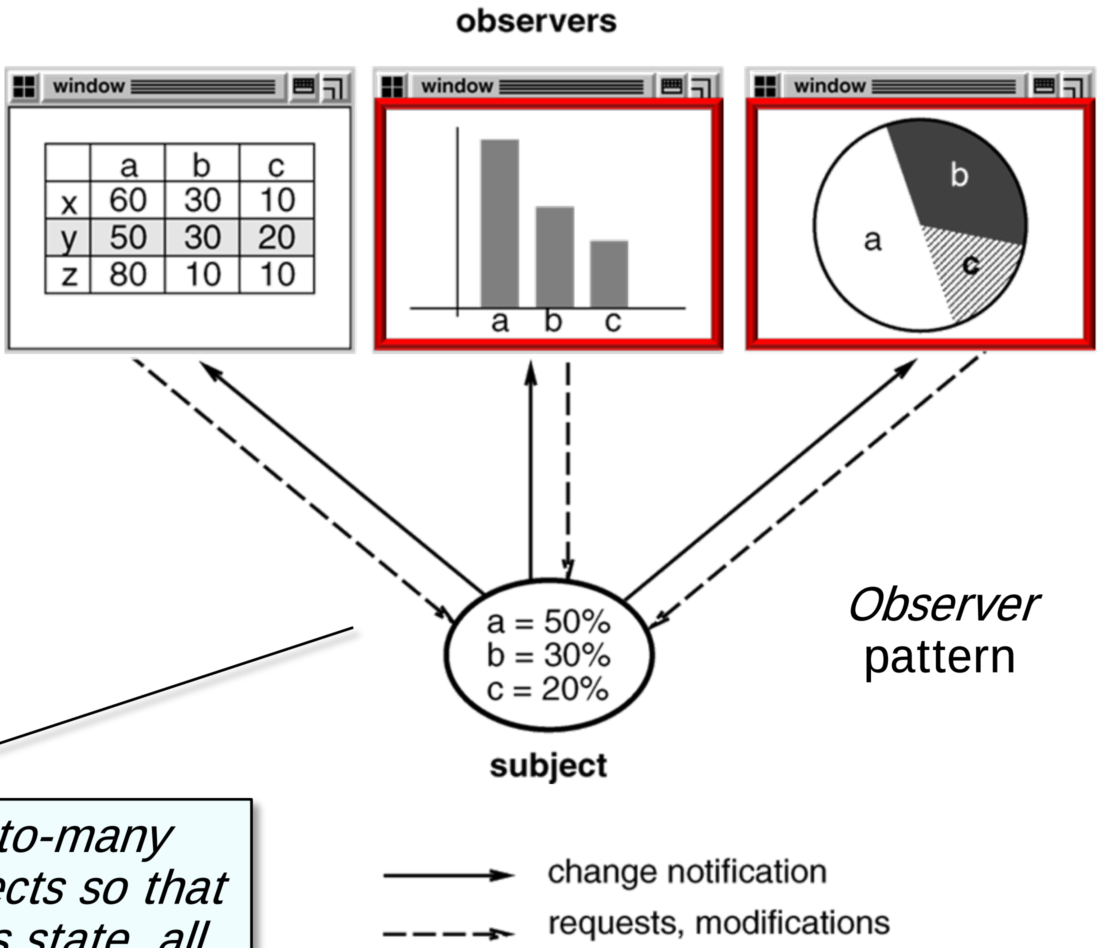
- Present solutions to common problems arising within a context
- Help resolve key software quality attribute forces
- Capture recurring structures & dynamics among software elements to facilitate reuse of successful designs



***Intent**: "Define a one-to-many dependency between objects so that when one object changes state, all dependents are notified & updated"*

Overview of Patterns

- Present solutions to common problems arising within a context
- Help resolve key software quality attribute forces
- Capture recurring structures & dynamics among software elements to facilitate reuse of successful designs

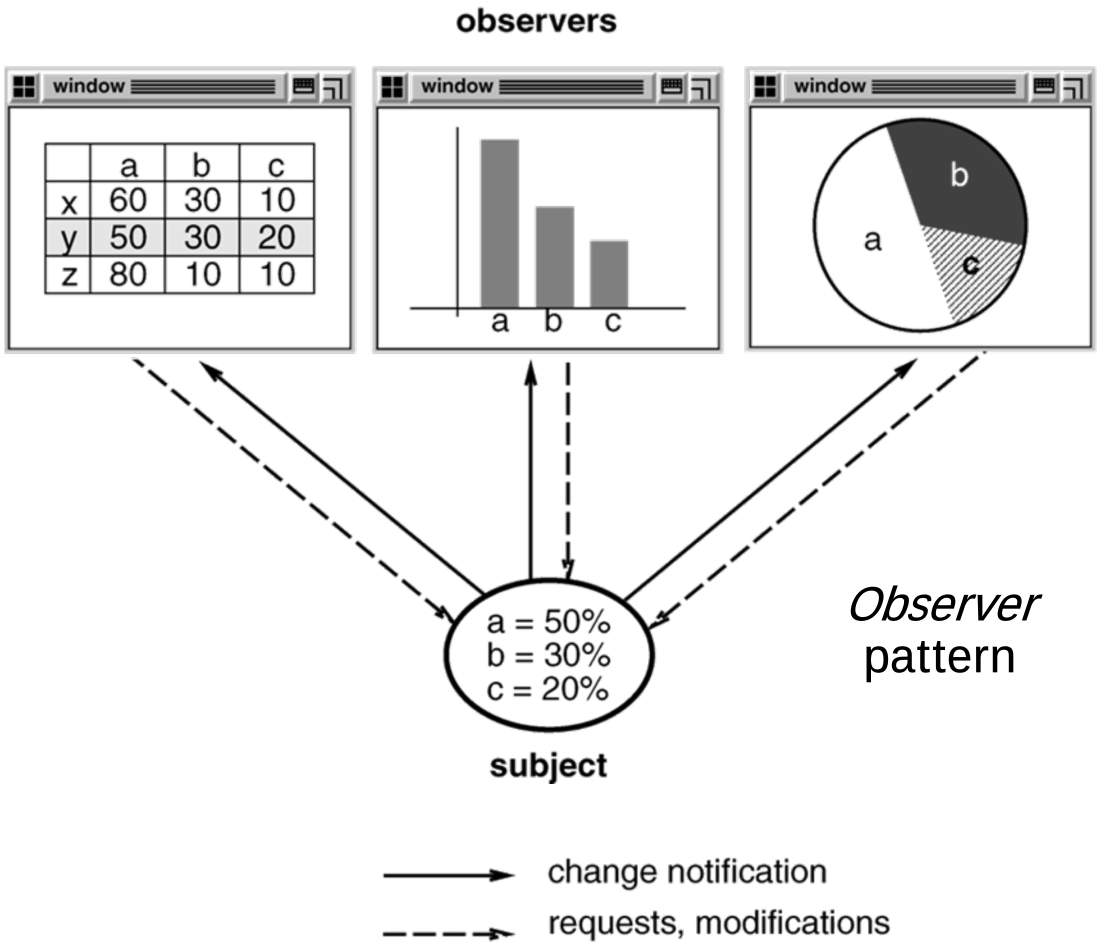


Observer pattern

Intent: *"Define a one-to-many dependency between objects so that when one object changes state, all dependents are notified & updated"*

Overview of Patterns

- Present solutions to common problems arising within a context
- Help resolve key software quality attribute forces
- Capture recurring structures & dynamics among software elements to facilitate reuse of successful designs

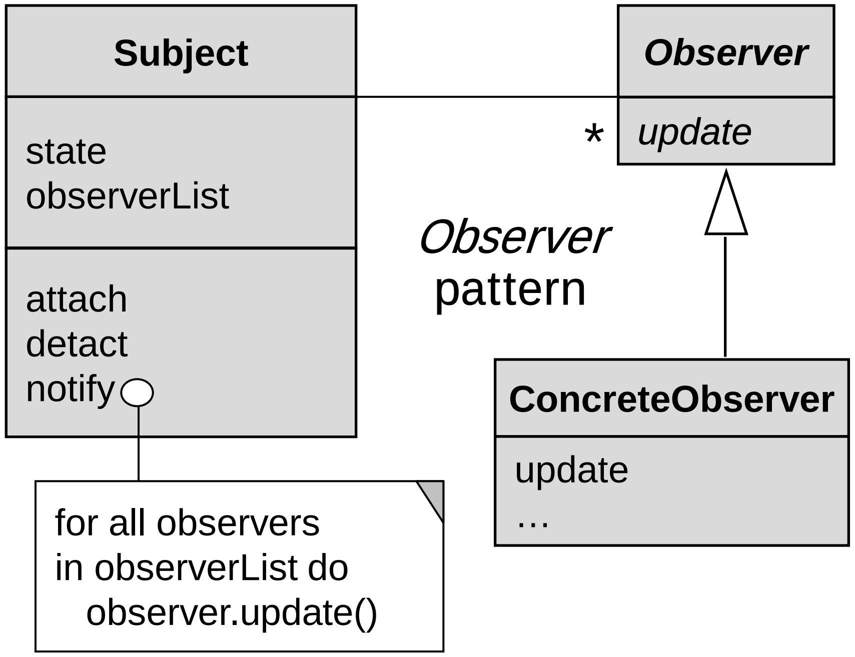


Observer pattern

en.wikipedia.org/wiki/Observer_pattern has more on *Observer*

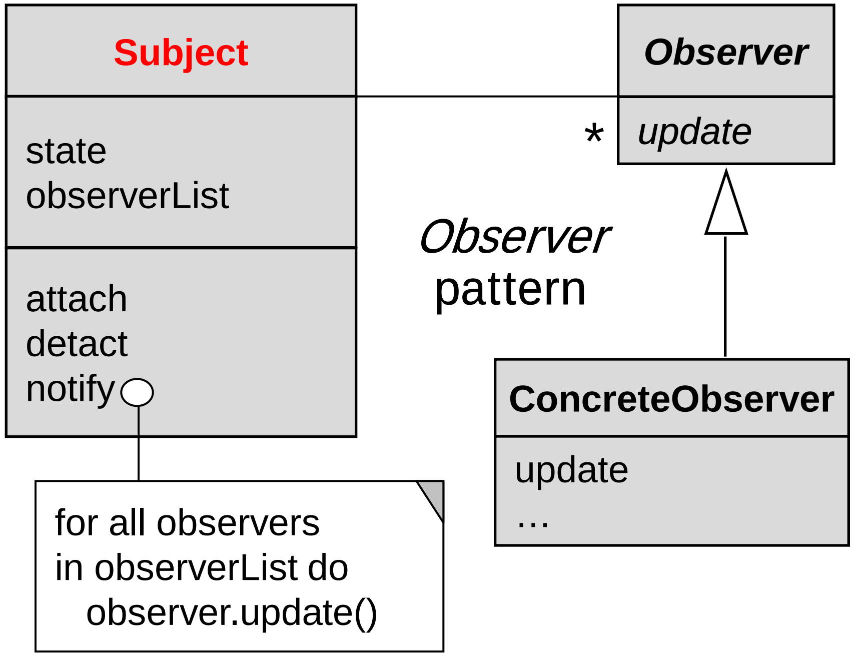
Overview of Patterns

- Present solutions to common problems arising within a context
- Help resolve key software quality attribute forces
- Capture recurring structures & dynamics among software elements to facilitate reuse of successful designs



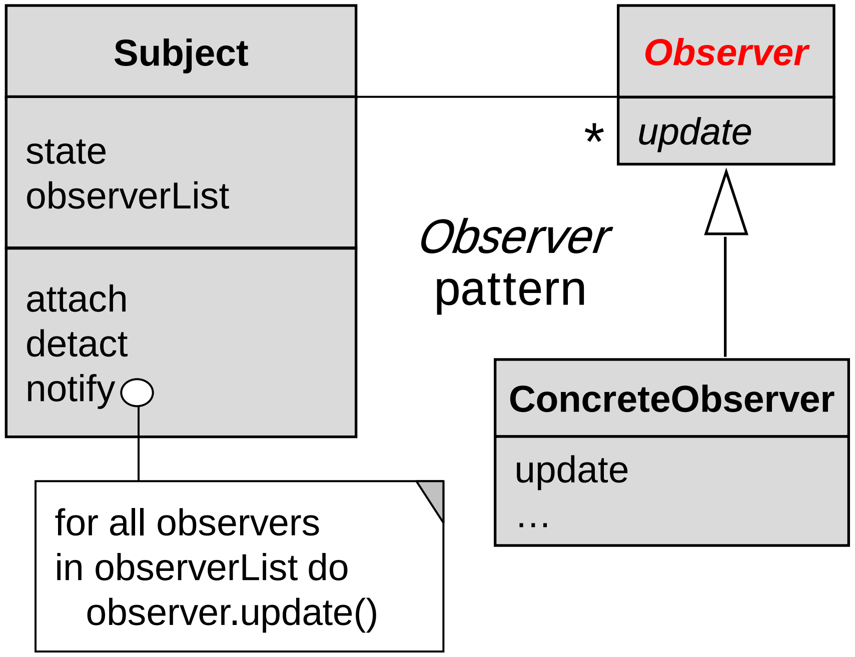
Overview of Patterns

- Present solutions to common problems arising within a context
- Help resolve key software quality attribute forces
- Capture recurring structures & dynamics among software elements to facilitate reuse of successful designs



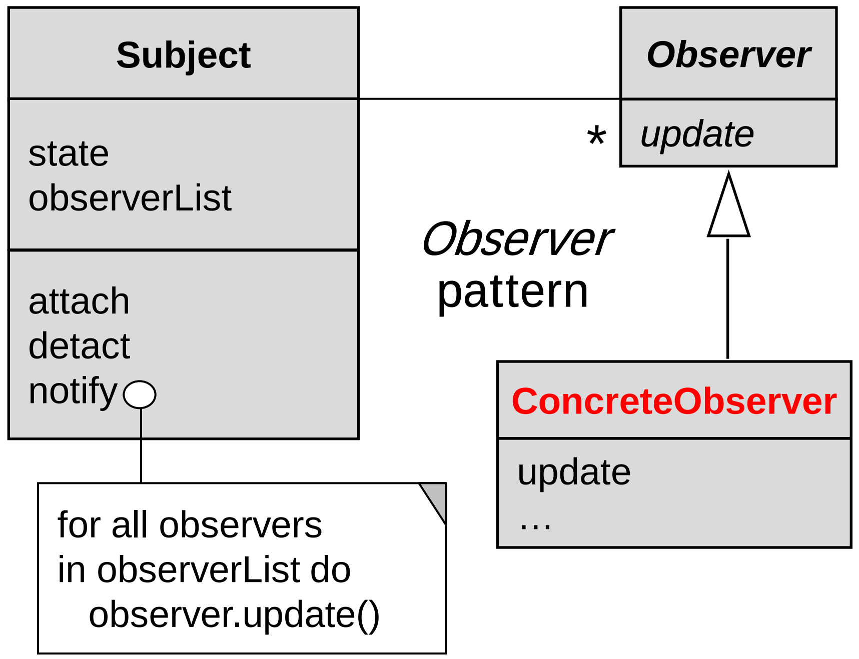
Overview of Patterns

- Present solutions to common problems arising within a context
- Help resolve key software quality attribute forces
- Capture recurring structures & dynamics among software elements to facilitate reuse of successful designs



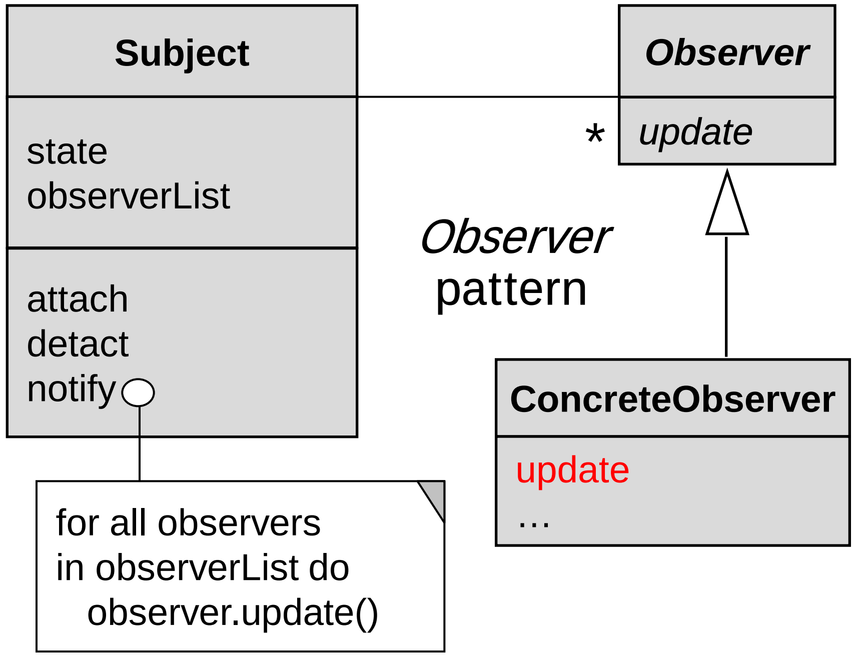
Overview of Patterns

- Present solutions to common problems arising within a context
- Help resolve key software quality attribute forces
- Capture recurring structures & dynamics among software elements to facilitate reuse of successful designs



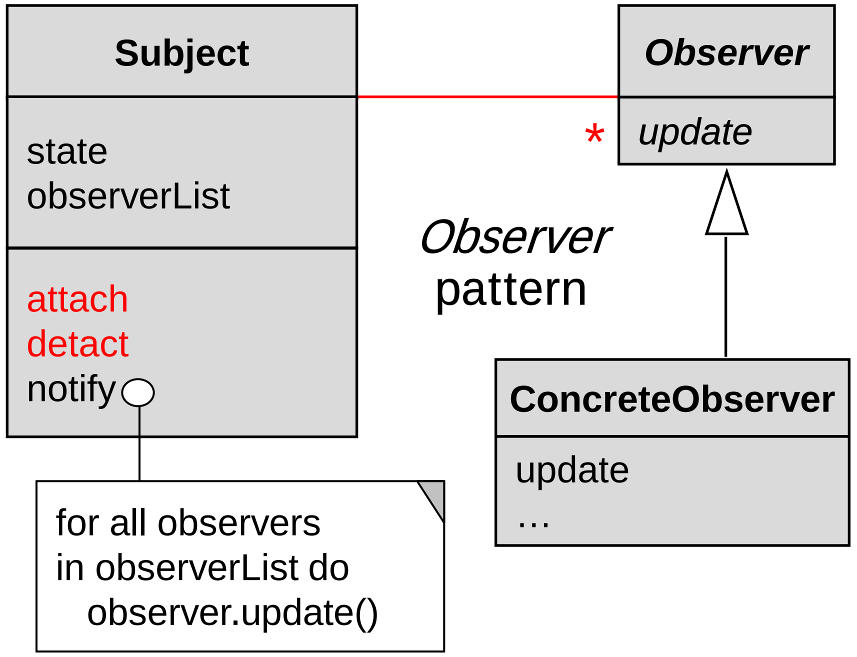
Overview of Patterns

- Present solutions to common problems arising within a context
- Help resolve key software quality attribute forces
- Capture recurring structures & dynamics among software elements to facilitate reuse of successful designs



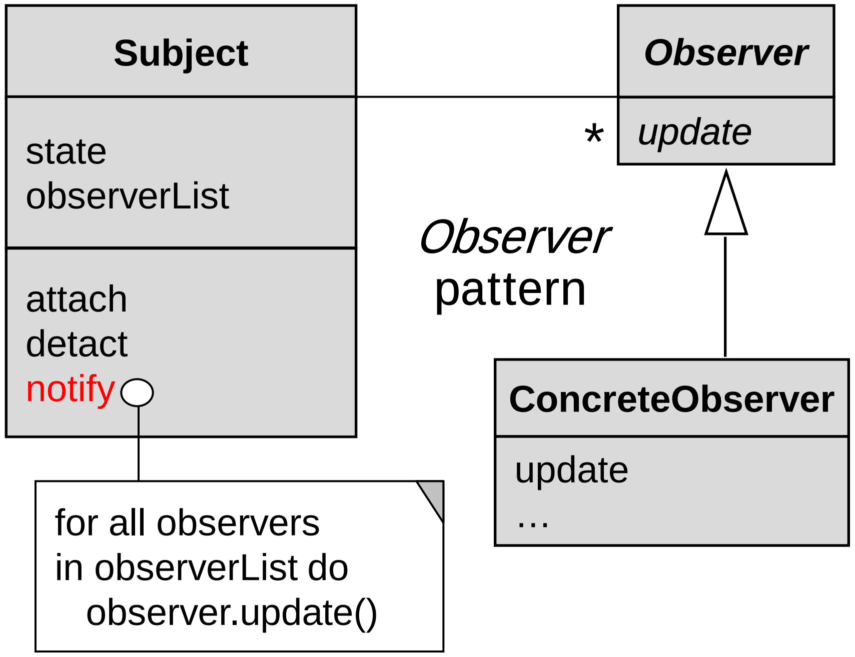
Overview of Patterns

- Present solutions to common problems arising within a context
- Help resolve key software quality attribute forces
- Capture recurring structures & dynamics among software elements to facilitate reuse of successful designs



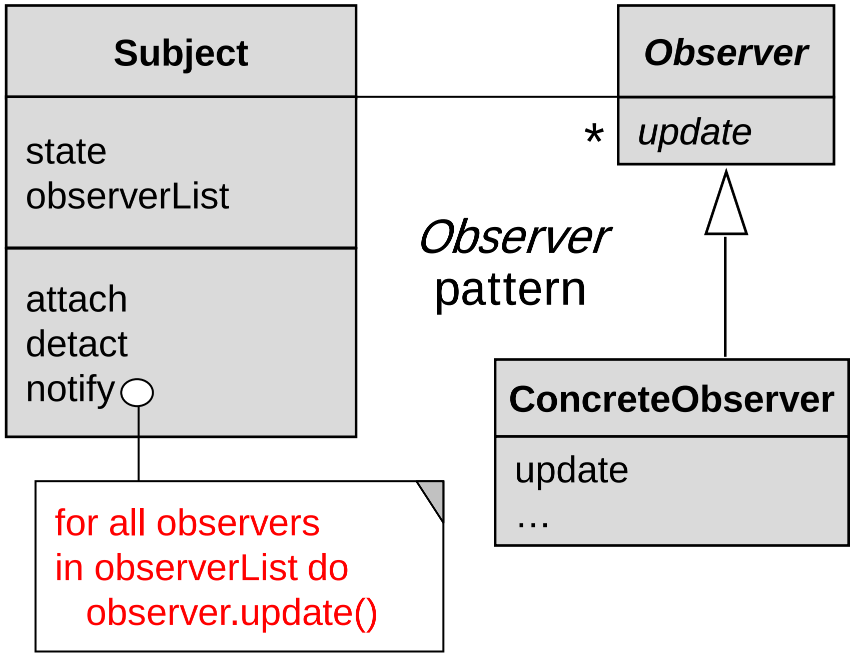
Overview of Patterns

- Present solutions to common problems arising within a context
- Help resolve key software quality attribute forces
- Capture recurring structures & dynamics among software elements to facilitate reuse of successful designs



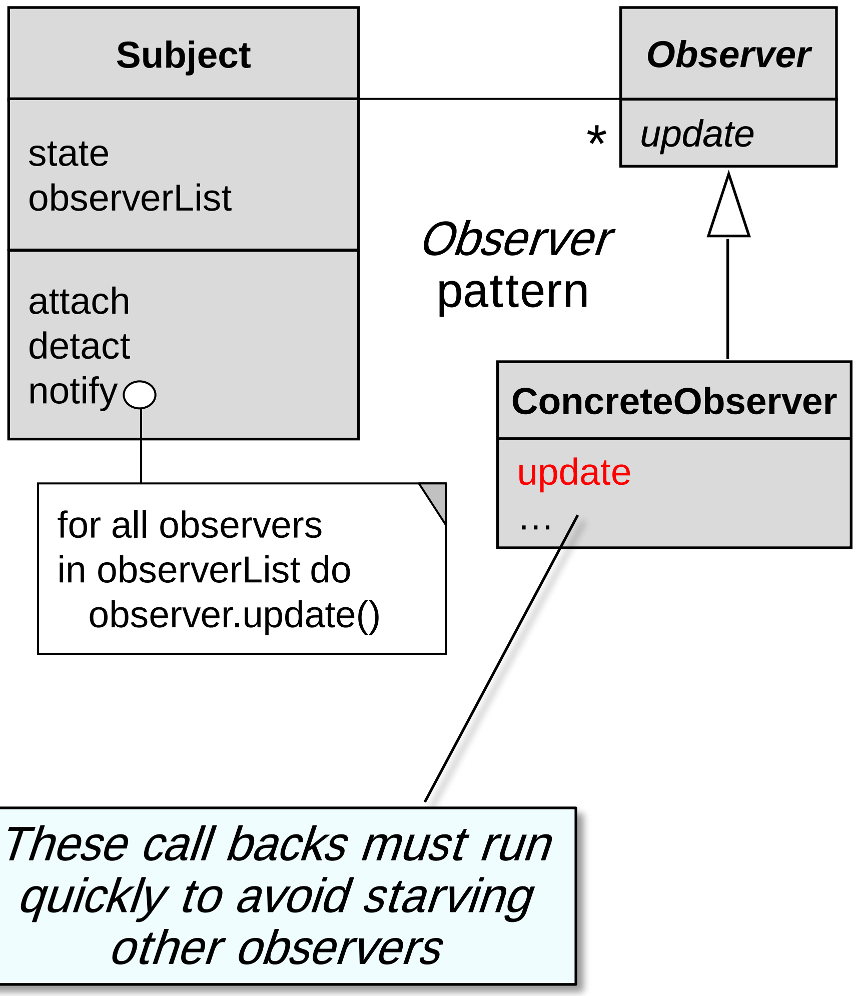
Overview of Patterns

- Present solutions to common problems arising within a context
- Help resolve key software quality attribute forces
- Capture recurring structures & dynamics among software elements to facilitate reuse of successful designs



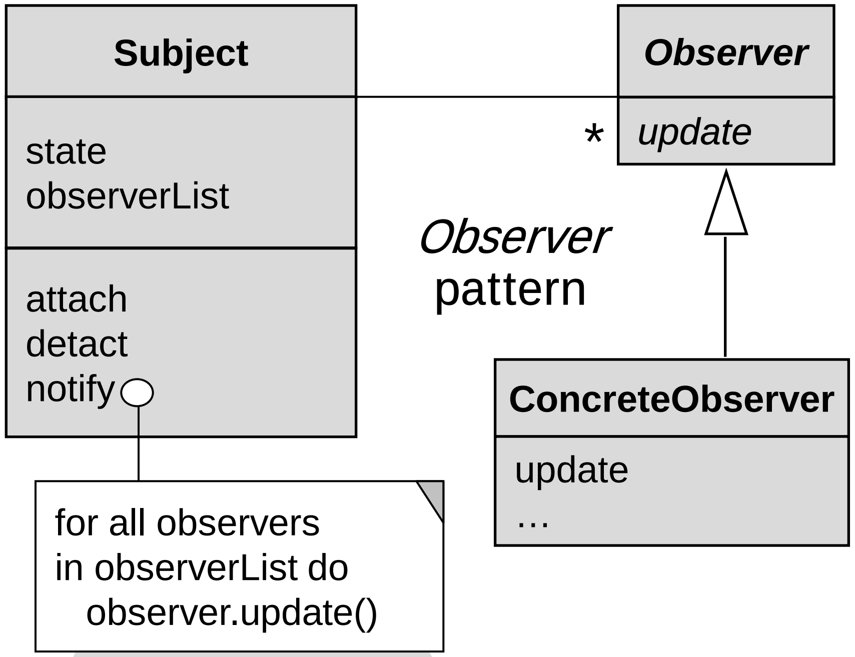
Overview of Patterns

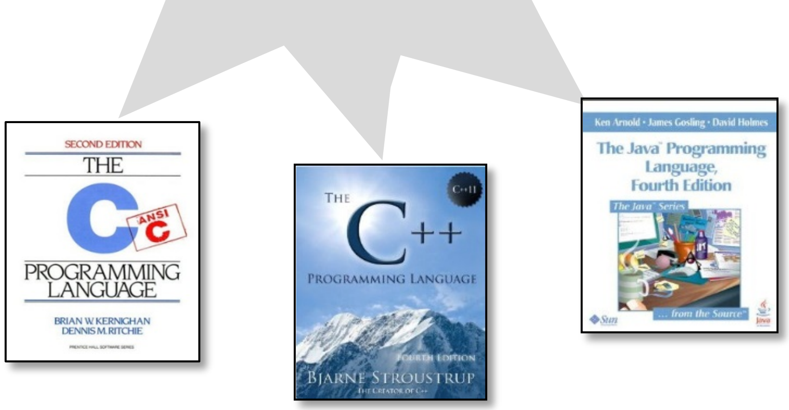
- Present solutions to common problems arising within a context
- Help resolve key software quality attribute forces
- Capture recurring structures & dynamics among software elements to facilitate reuse of successful designs



Overview of Patterns

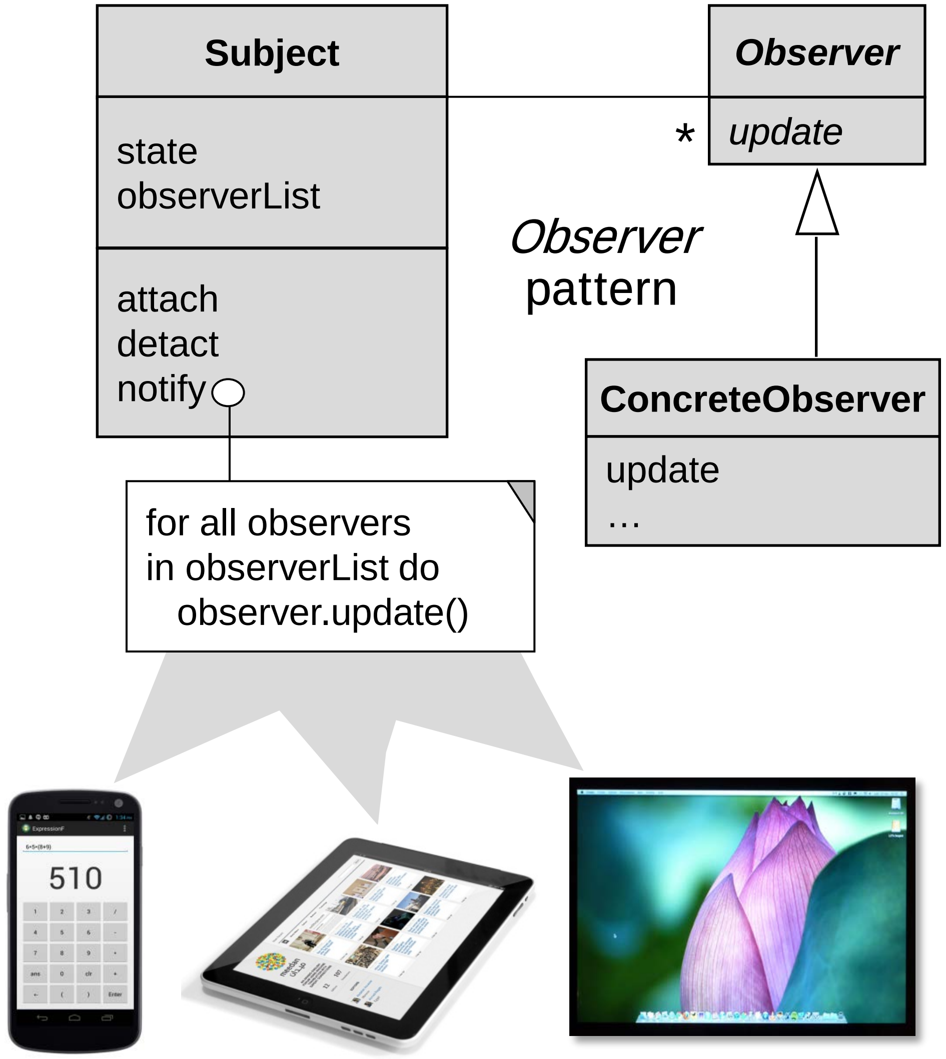
- Present solutions to common problems arising within a context
- Help resolve key software quality attribute forces
- Capture recurring structures & dynamics among software elements to facilitate reuse of successful designs




THE C PROGRAMMING LANGUAGE
THE C++ PROGRAMMING LANGUAGE
The Java Programming Language, Fourth Edition

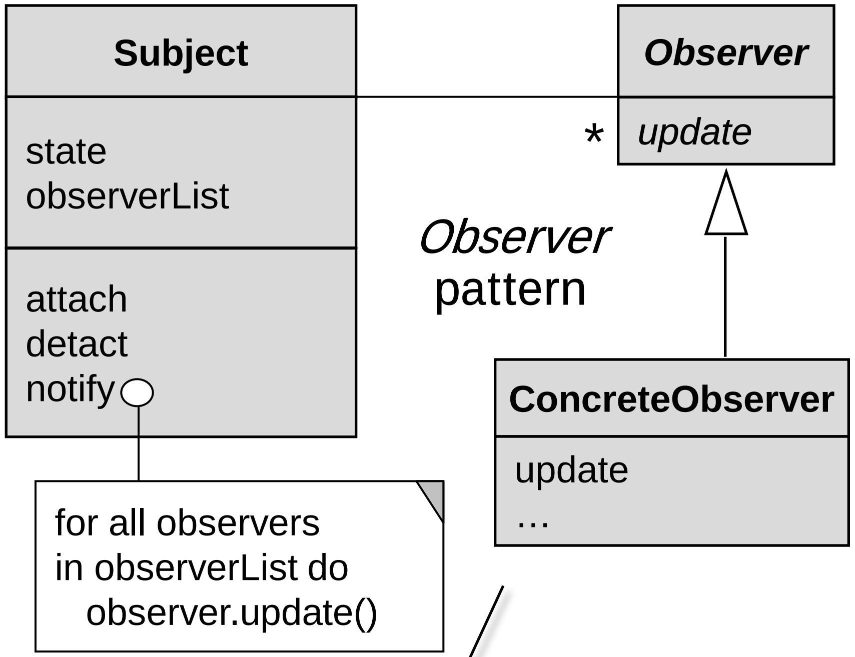
Overview of Patterns

- Present solutions to common problems arising within a context
- Help resolve key software quality attribute forces
- Capture recurring structures & dynamics among software elements to facilitate reuse of successful designs



Overview of Patterns

- Present solutions to common problems arising within a context
- Help resolve key software quality attribute forces
- Capture recurring structures & dynamics among software elements to facilitate reuse of successful designs



The roles & responsibilities of patterns are largely the same across implementations

Patterns codify & allow reuse of valuable design knowledge

Overview of Patterns

- Present solutions to common problems arising within a context
- Help resolve key software quality attribute forces
- Capture recurring structures & dynamics among software elements to facilitate reuse of successful designs
- Help augment & “de-risk” mentoring relationships between experts & apprentices

en.wikipedia.org/wiki/Bus_factor has more on getting hit by a bus ;-)

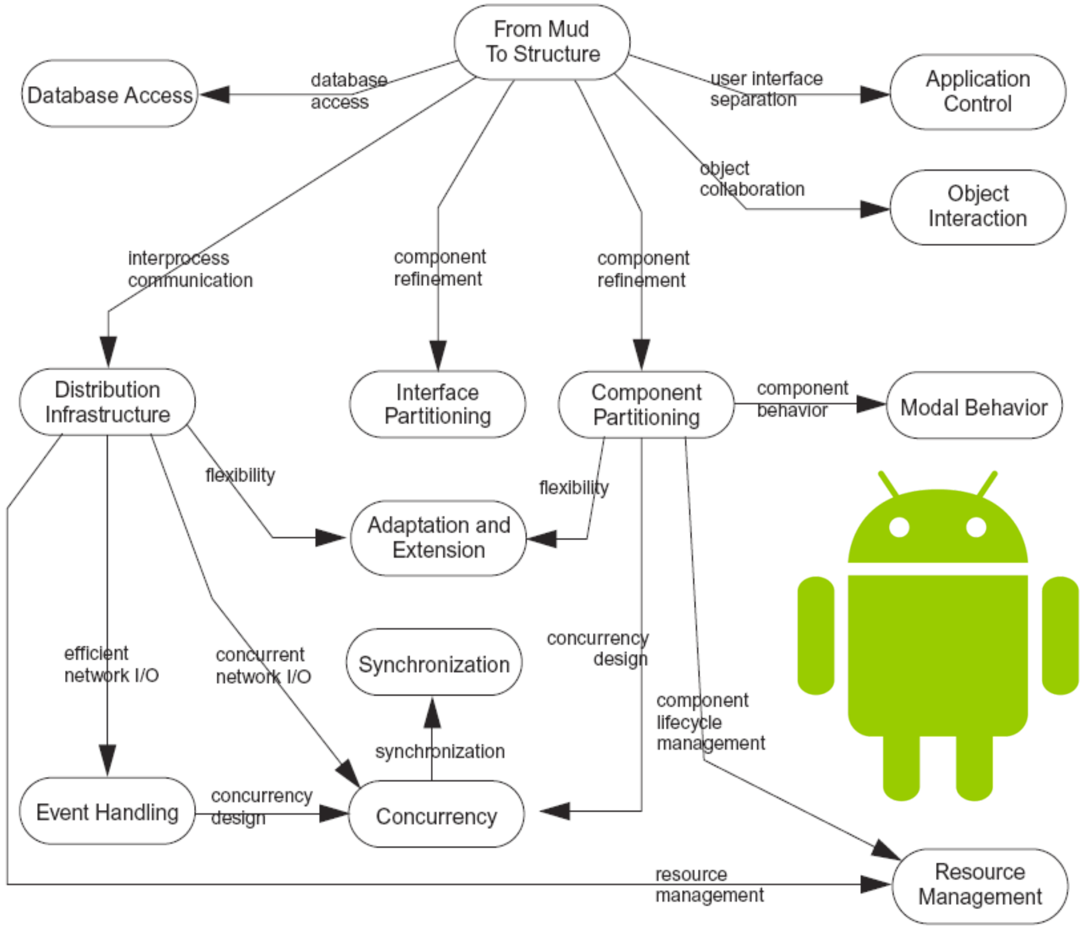
Overview of Patterns

- Present solutions to common problems arising within a context
- Help resolve key software quality attribute forces
- Capture recurring structures & dynamics among software elements to facilitate reuse of successful designs
- Help augment & "de-risk" mentoring relationships between experts & apprentices

Enterprise Integration Patterns
Designing, Building, and Deploying Messaging Solutions
Gregor Hohpe
Bobby Woolf
Pattern-Oriented Software Architecture
Volume 3 Patterns for Resource Management
Small Memory Software
Patterns for systems with limited memory
James Noble
Charles Weir
Security Patterns
Integrating Security and Systems Engineering
Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
Server Component Patterns
Component Infrastructures Illustrated with EJB
Pattern-Oriented Software Architecture
A System of Patterns
Volume 1
Frank Buschmann
Regine Meunier
Hans Rohnert
Peter Sommerlad
Michael Stal
WILEY
Pattern-Oriented Software Architecture
On Patterns and Pattern Languages
Volume 5
Pattern-Oriented Software Architecture
Volume 2 Patterns for Concurrent and Networked Objects
Patterns for Parallel Software Design
Doug Lea
Concurrent Programming in Java
Second Edition
Design Principles and Patterns
Pattern-Oriented Software Architecture
A Pattern Language for Distributed Object Computing
Volume 4

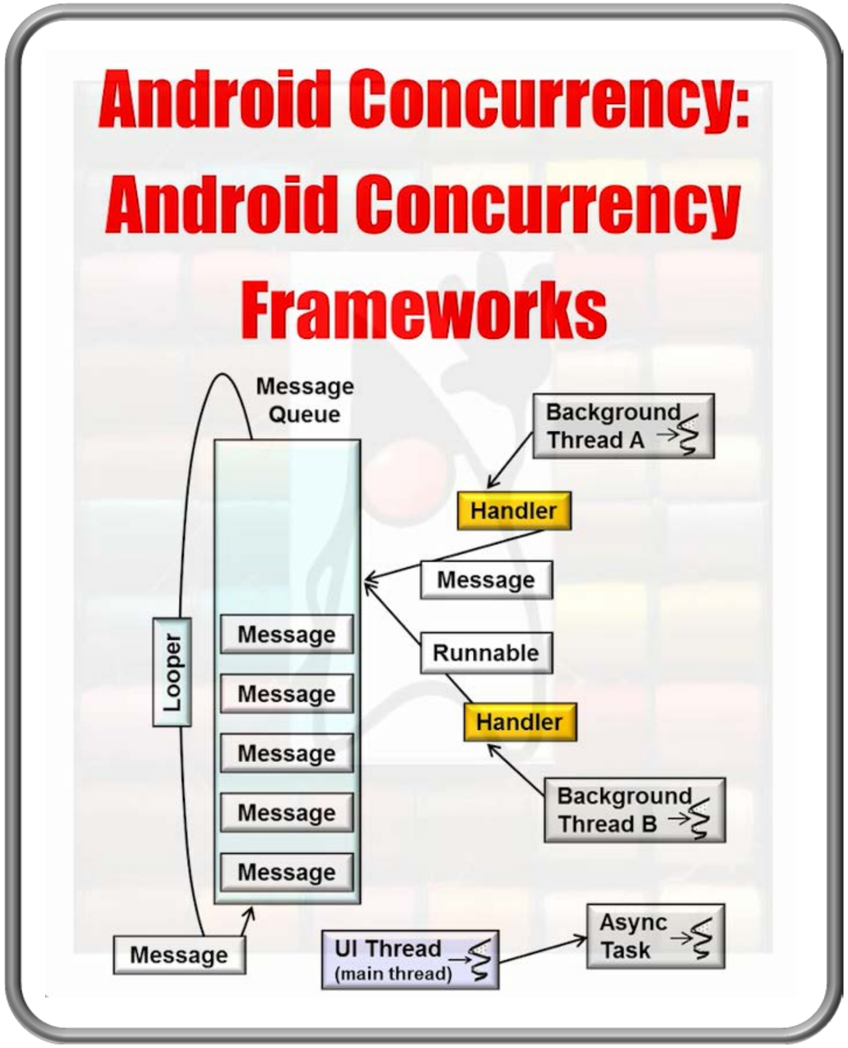
Overview of Patterns

- Present solutions to common problems arising within a context
- Help resolve key software quality attribute forces
- Capture recurring structures & dynamics among software elements to facilitate reuse of successful designs
- Help augment & "de-risk" mentoring relationships between experts & apprentices
- Many parts of Android are guided by patterns



Overview of Patterns

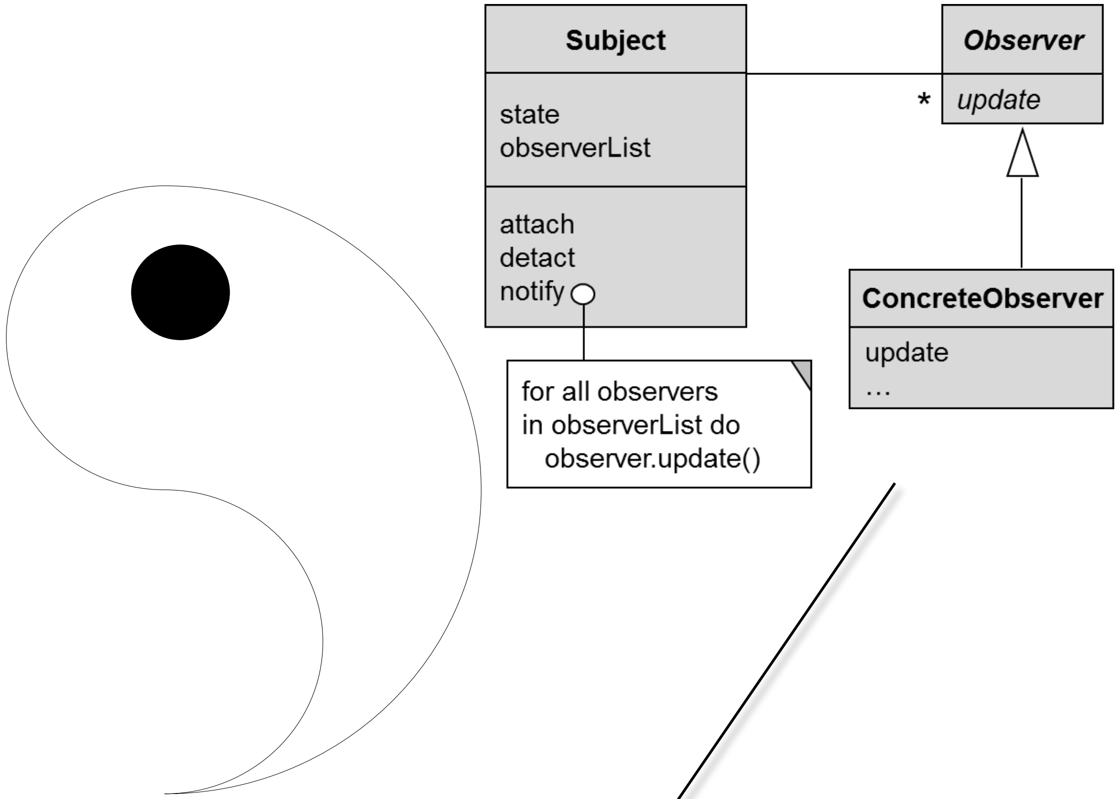
- Present solutions to common problems arising within a context
- Help resolve key software quality attribute forces
- Capture recurring structures & dynamics among software elements to facilitate reuse of successful designs
- Help augment & "de-risk" mentoring relationships between experts & apprentices
- Many parts of Android are guided by patterns



See item #17 at class.coursera.org/posa-002/wiki/FrequentlyAskedQuestions

Overview of Frameworks

Motivation for Frameworks

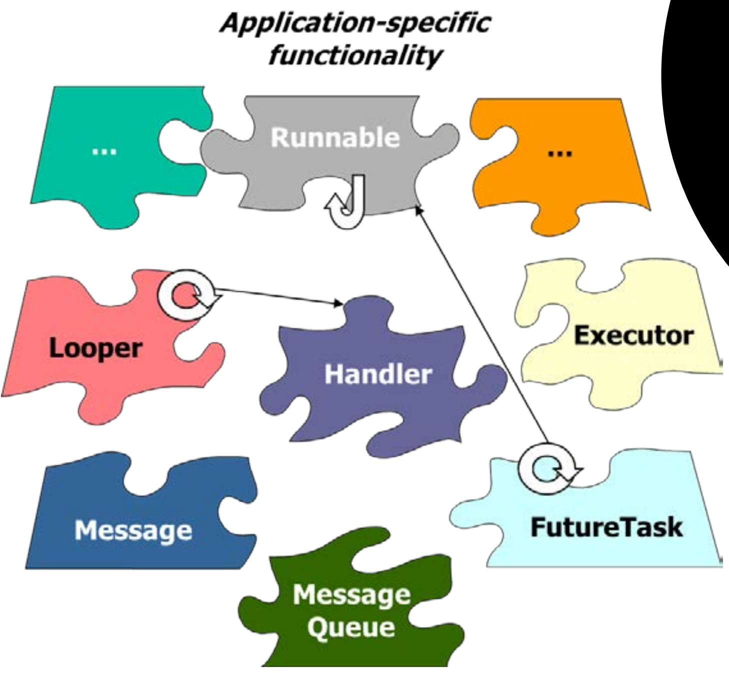


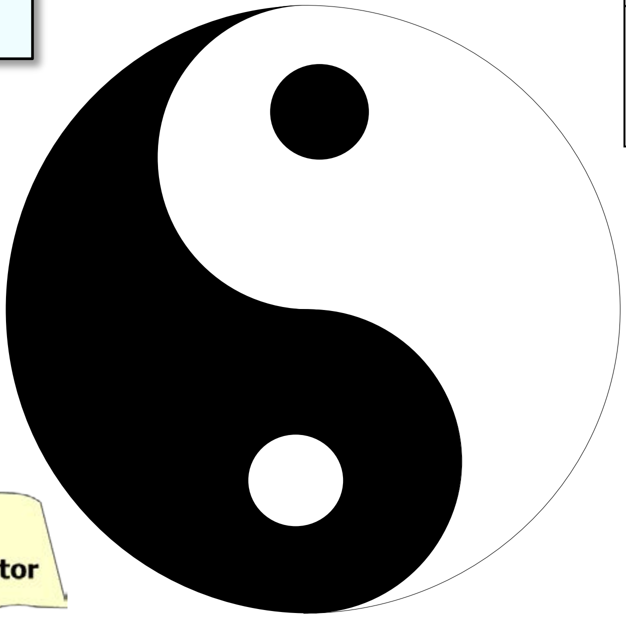
Patterns codify experience & guide design & architecture decisions

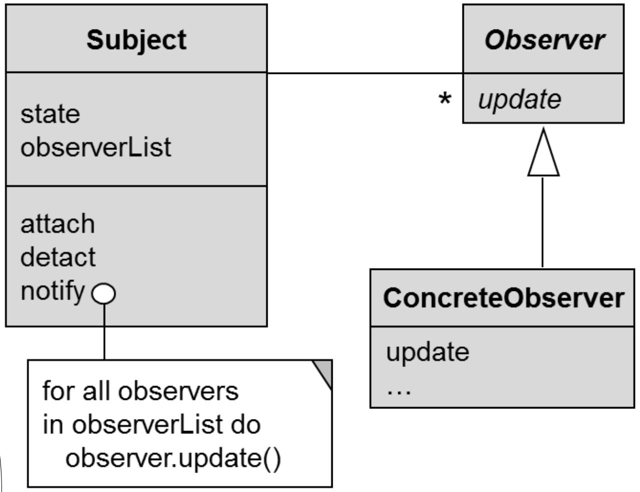
Software professionals also need implementation guidance

Motivation for Frameworks

Frameworks reify software experience & patterns in tangible code artifacts



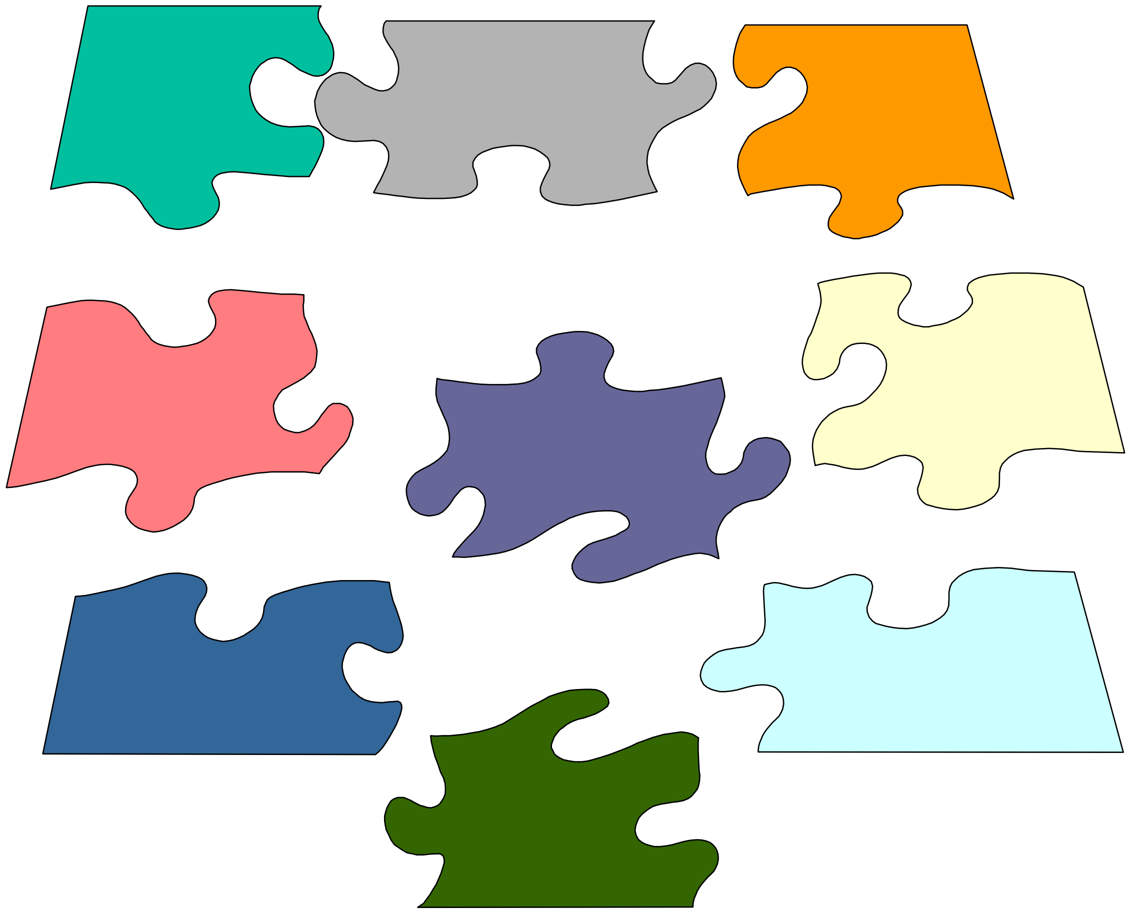




See en.wikipedia.org/wiki/Reification for more on what “reify” means

Overview of Frameworks

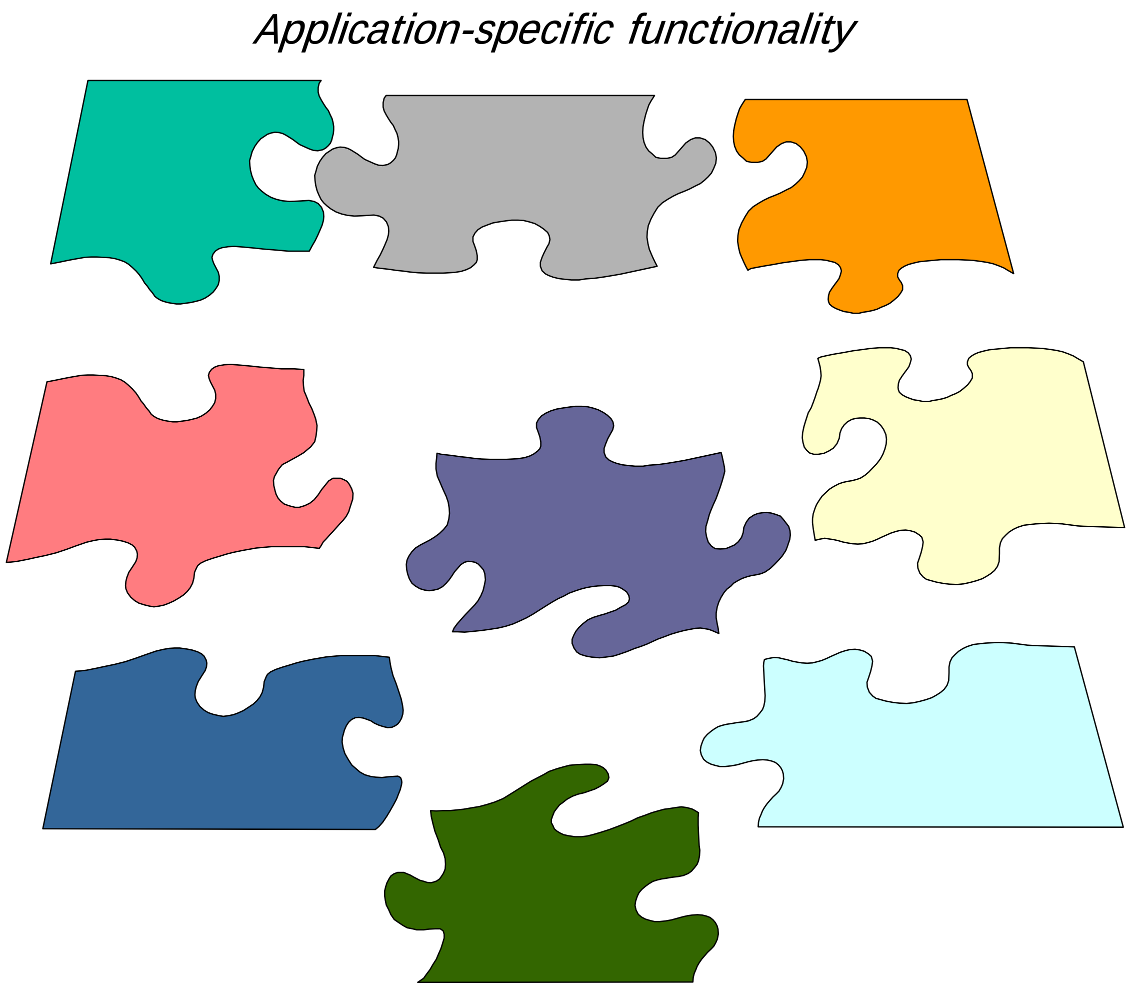
A framework is an integrated set of software components that collaborate to provide a reusable architecture for a family of related applications



See www.dre.vanderbilt.edu/~schmidt/CACM-frameworks.html for more info

Overview of Frameworks

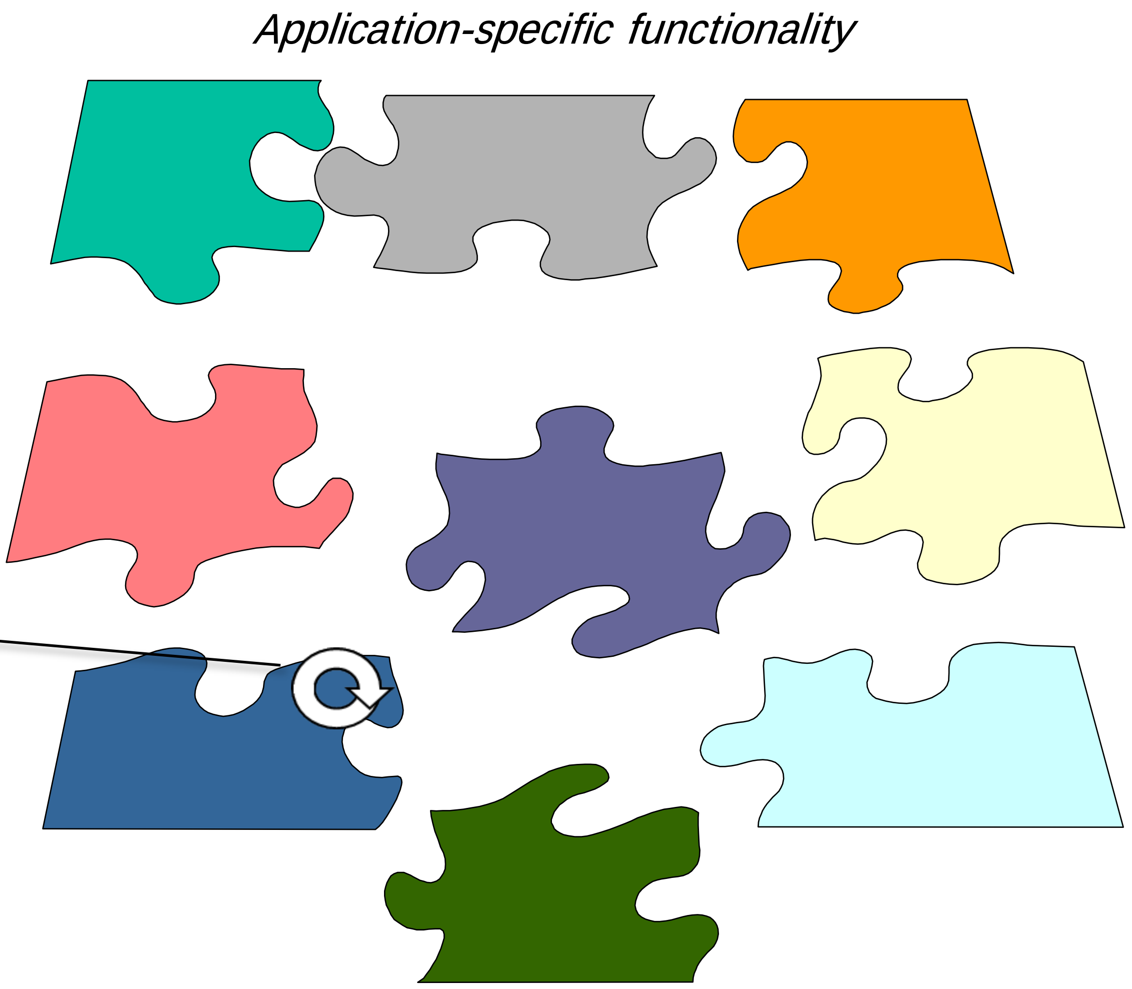
- They exhibit “inversion of control” via callbacks



See www.dre.vanderbilt.edu/~schmidt/Coursera/articles/hollywood-principle.txt

Overview of Frameworks

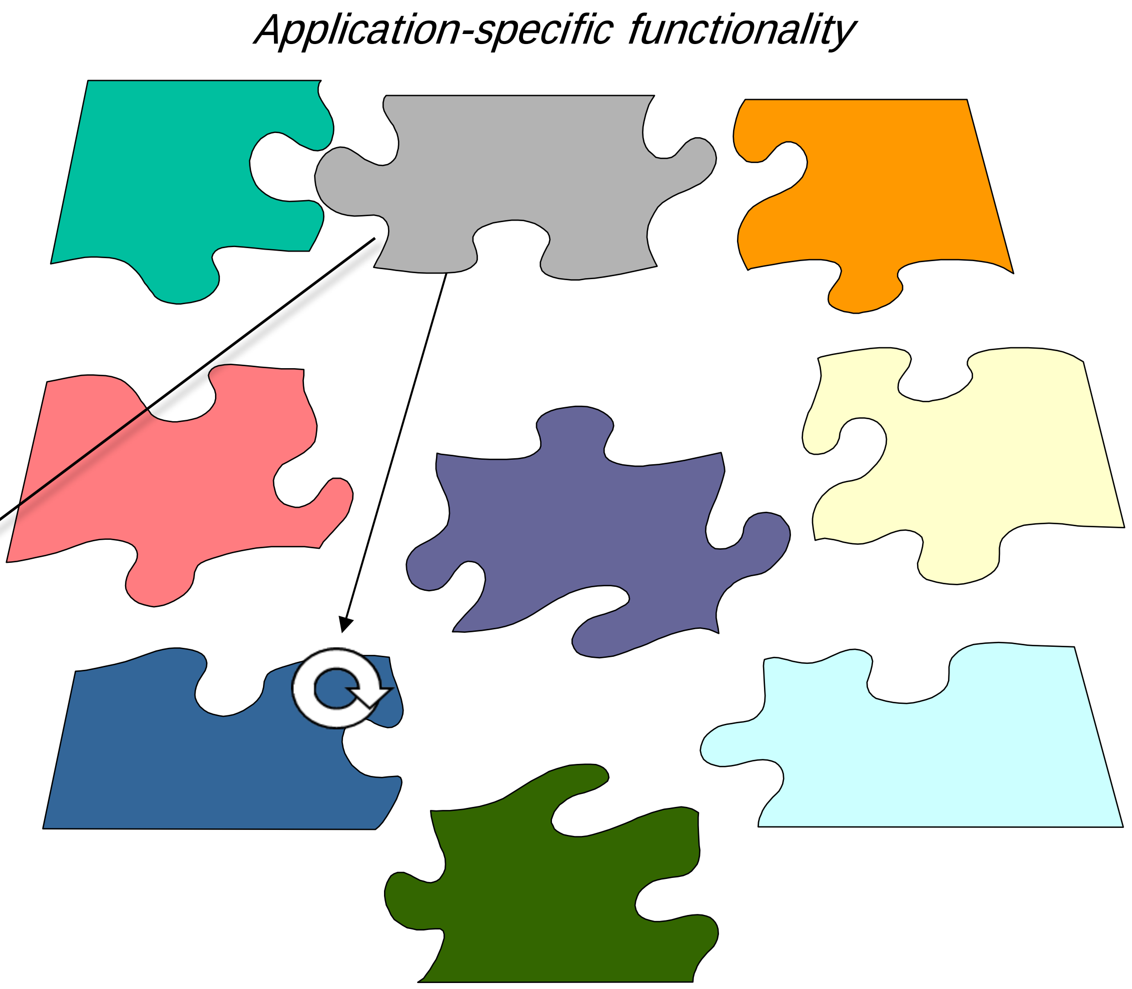
- They exhibit "inversion of control" via callbacks



The framework owns the application event loop

Overview of Frameworks

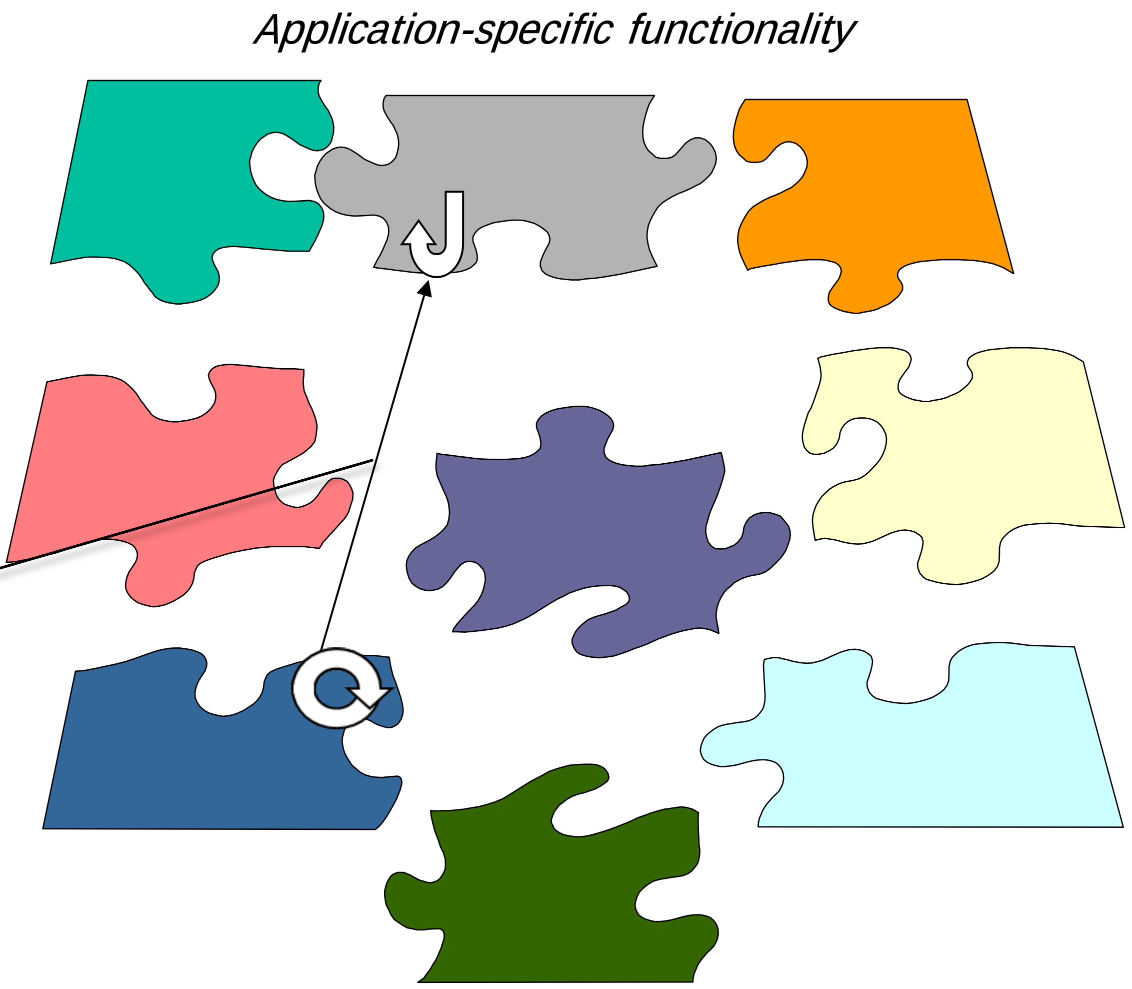
- They exhibit "inversion of control" via callbacks



Applications register event handler objects with the framework

Overview of Frameworks

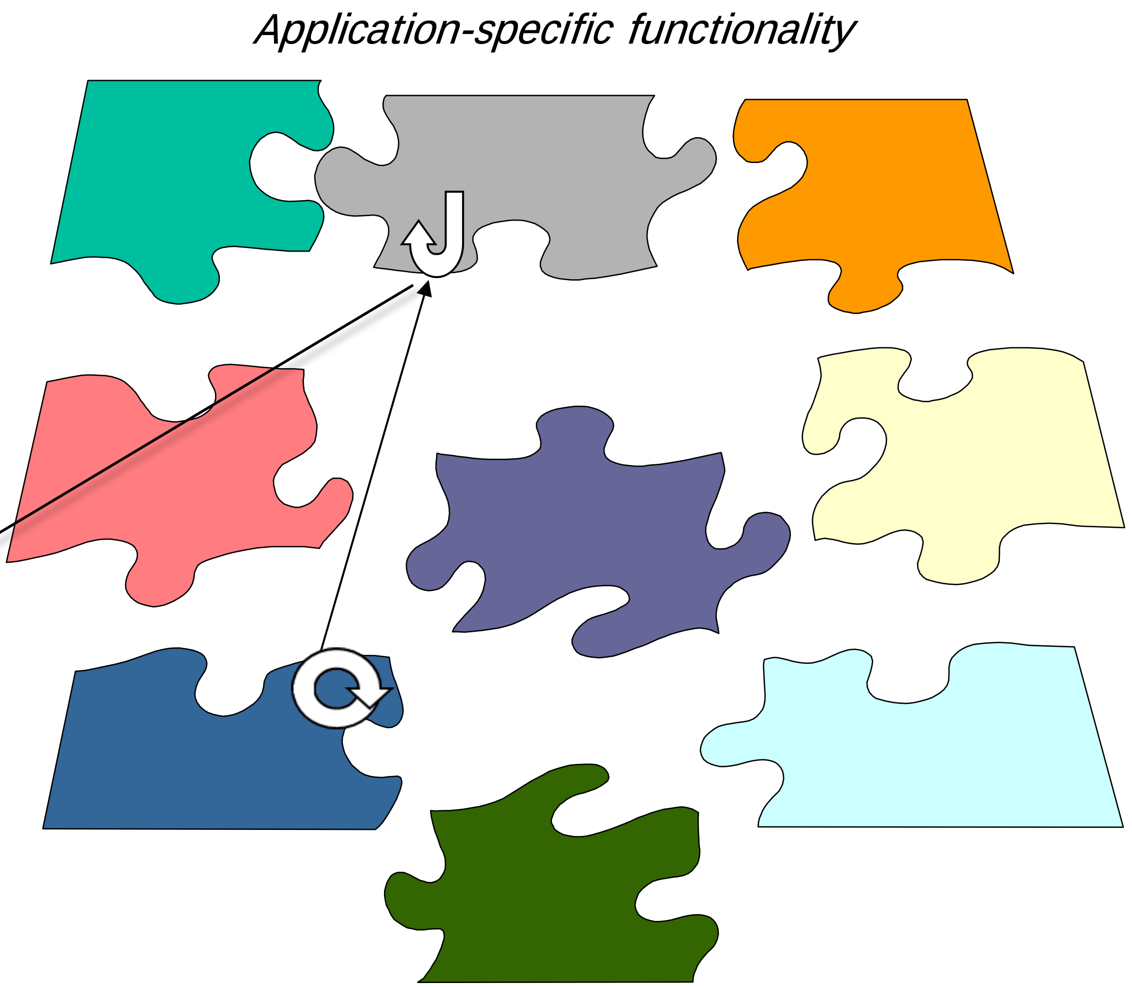
- They exhibit "inversion of control" via callbacks



The framework dispatches events to hook methods defined in event handlers

Overview of Frameworks

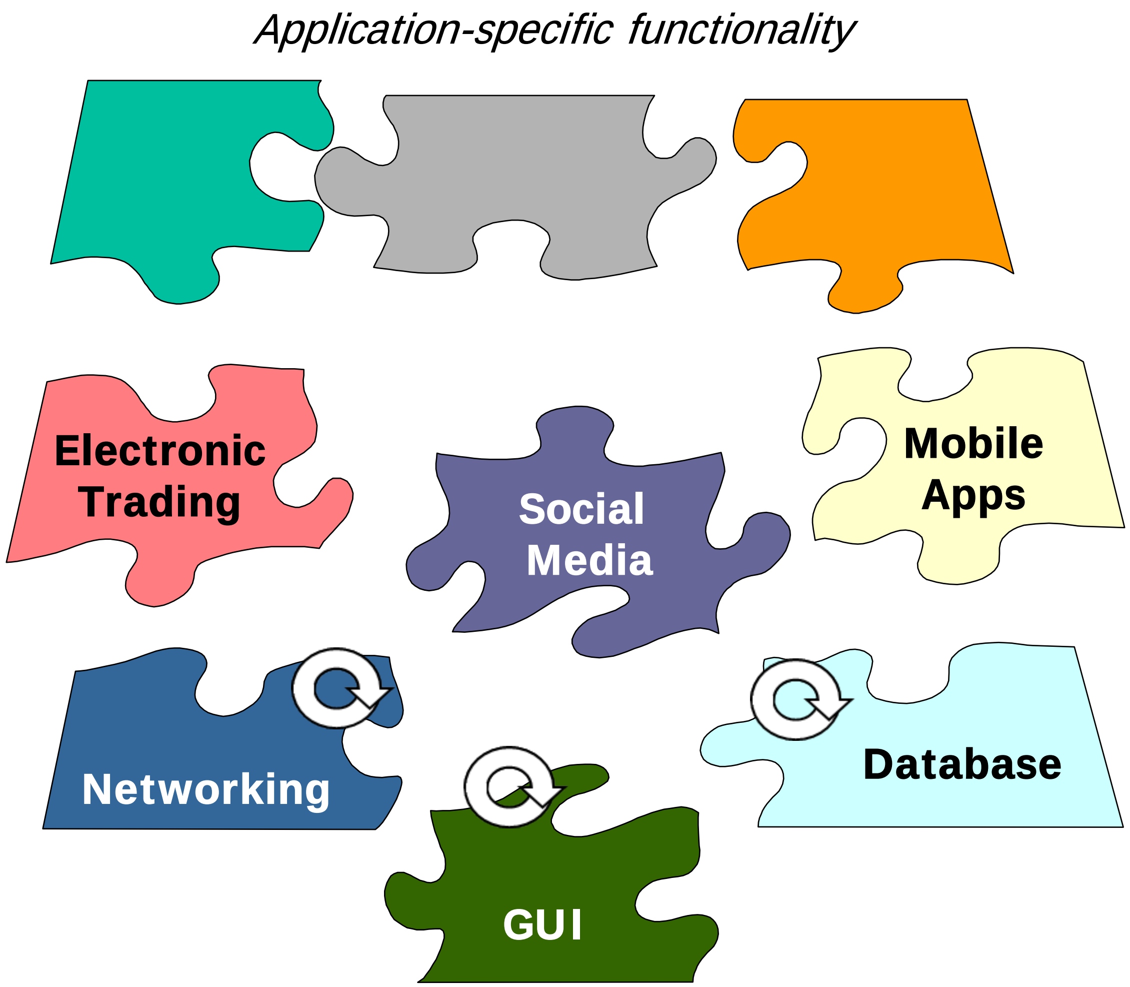
- They exhibit "inversion of control" via callbacks



Hook methods customize generic framework components to run application-specific logic

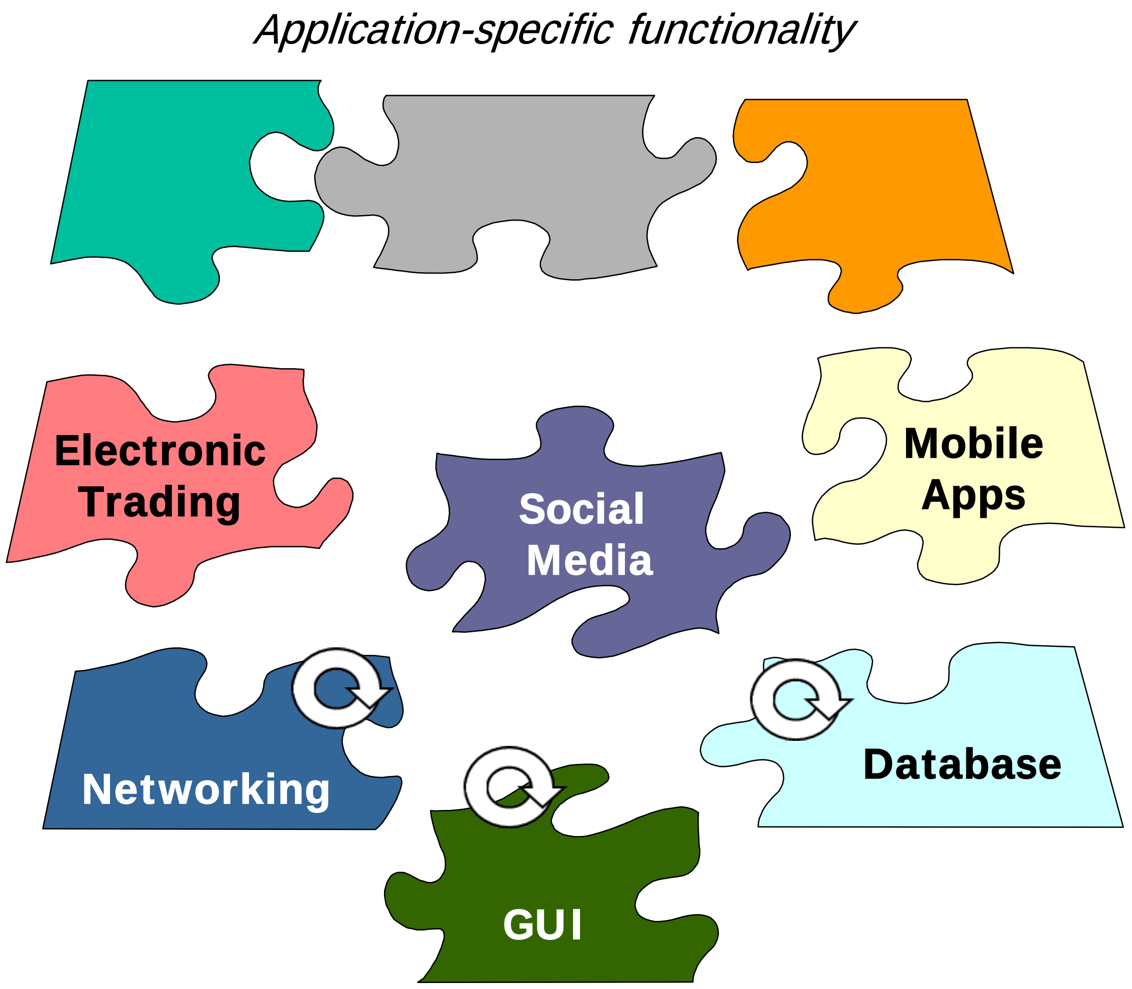
Overview of Frameworks

- They exhibit "inversion of control" via callbacks
- They provide integrated domain-specific structures & functionality



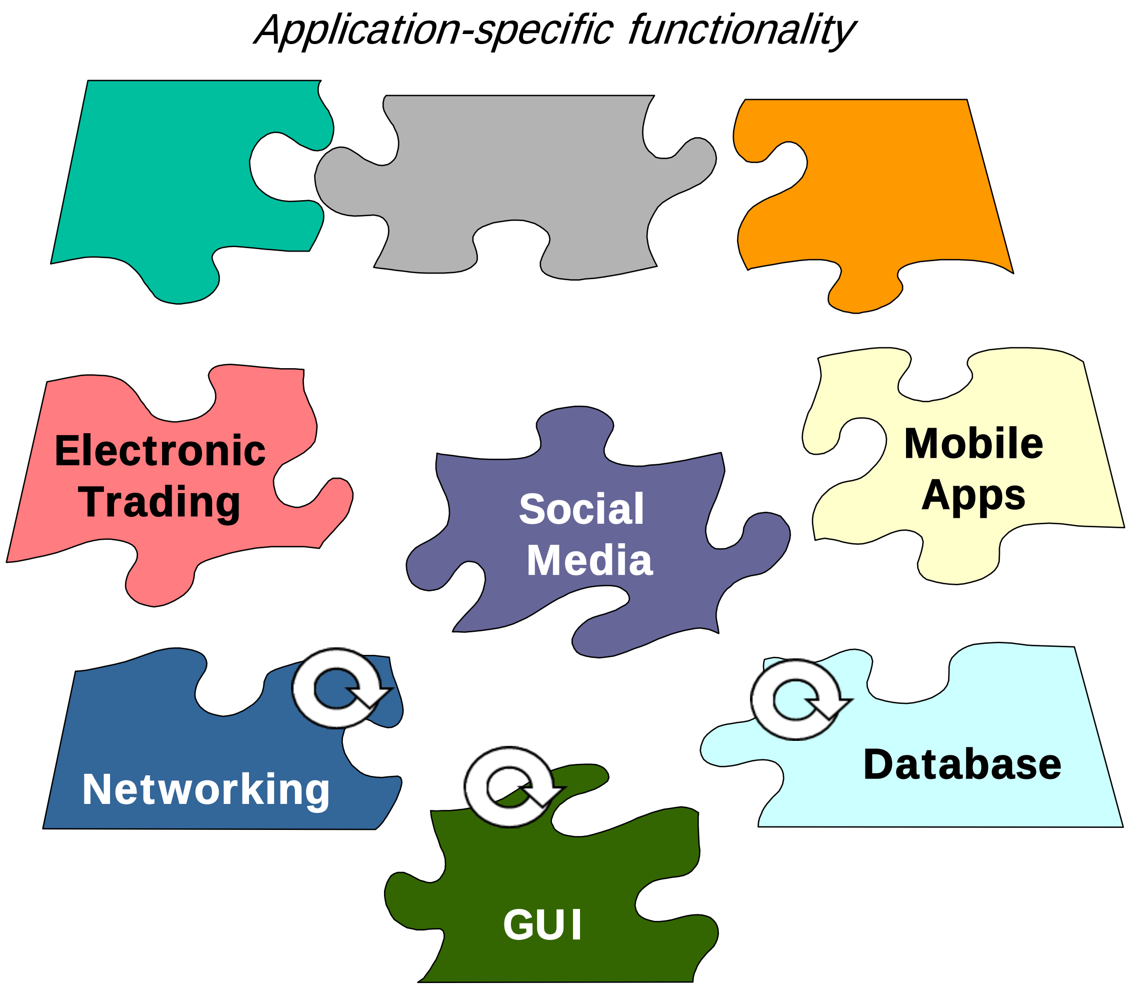
Overview of Frameworks

- They exhibit "inversion of control" via callbacks
- They provide integrated domain-specific structures & functionality, e.g.
  - Mediate common interactions between abstract & concrete classes



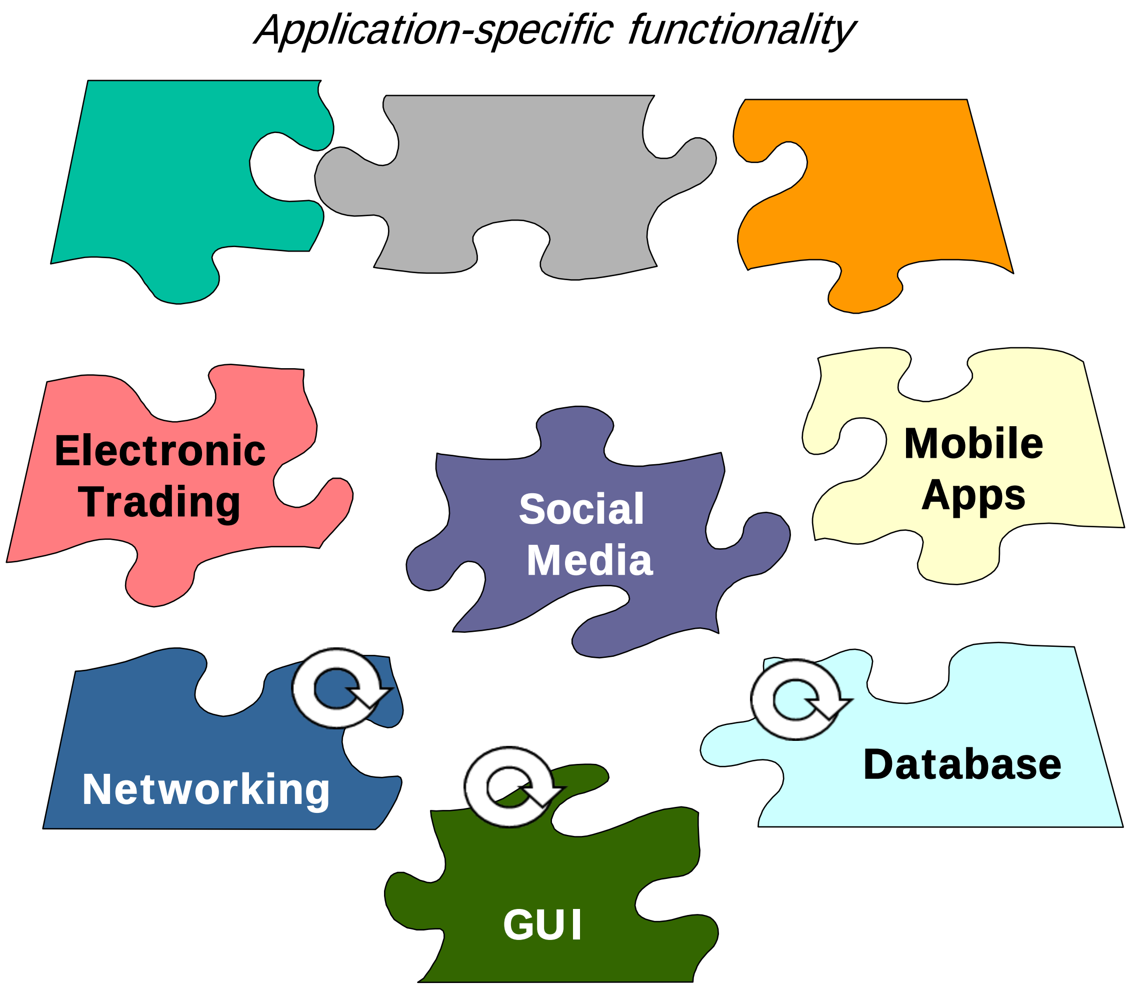
Overview of Frameworks

- They exhibit "inversion of control" via callbacks
- They provide integrated domain-specific structures & functionality, e.g.
  - Mediate common interactions between abstract & concrete classes
  - Guide canonical flow of control



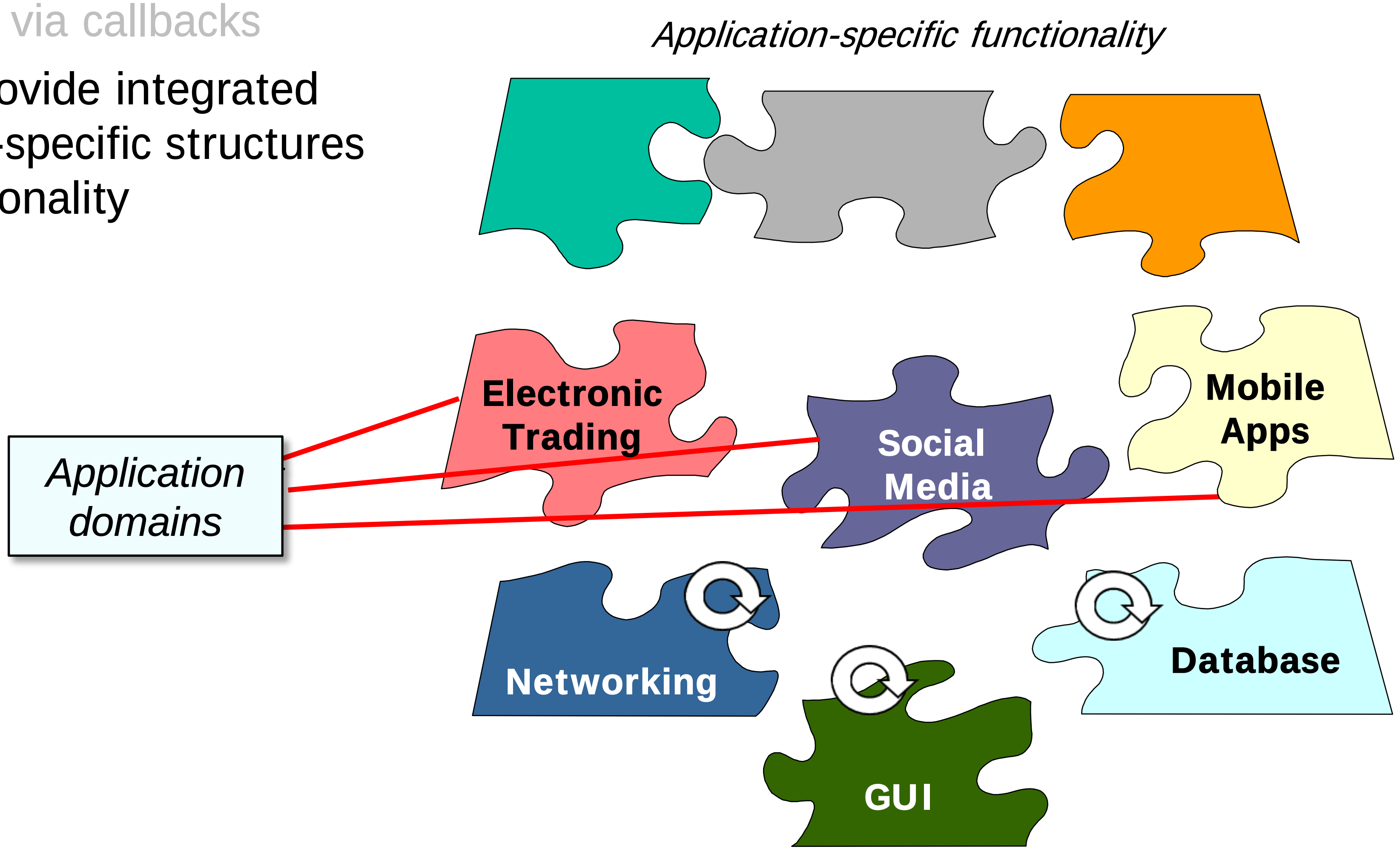
Overview of Frameworks

- They exhibit "inversion of control" via callbacks
- They provide integrated domain-specific structures & functionality, e.g.
    - Mediate common interactions between abstract & concrete classes
    - Guide canonical flow of control
    - Enforce architectural constraints



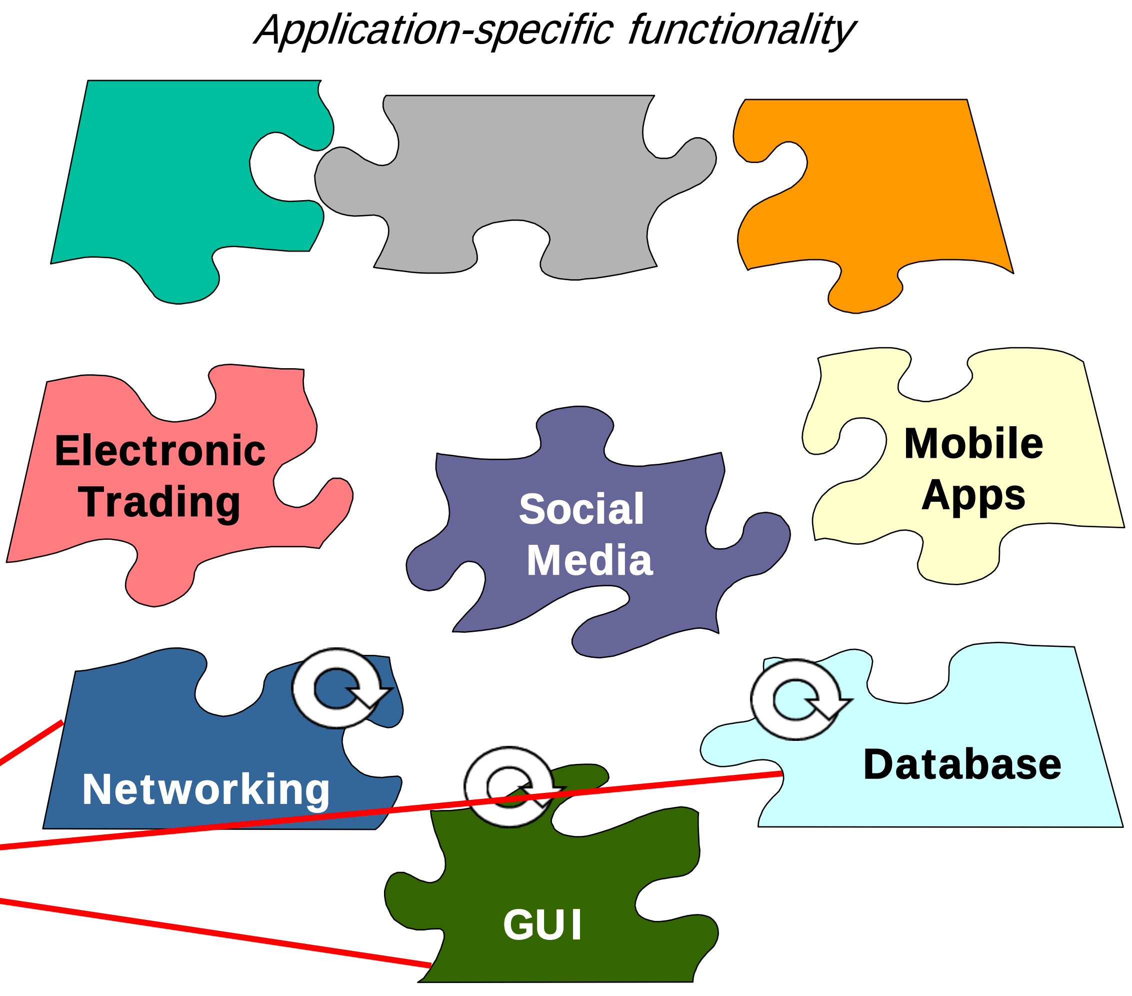
Overview of Frameworks

- They exhibit "inversion of control" via callbacks
- They provide integrated domain-specific structures & functionality



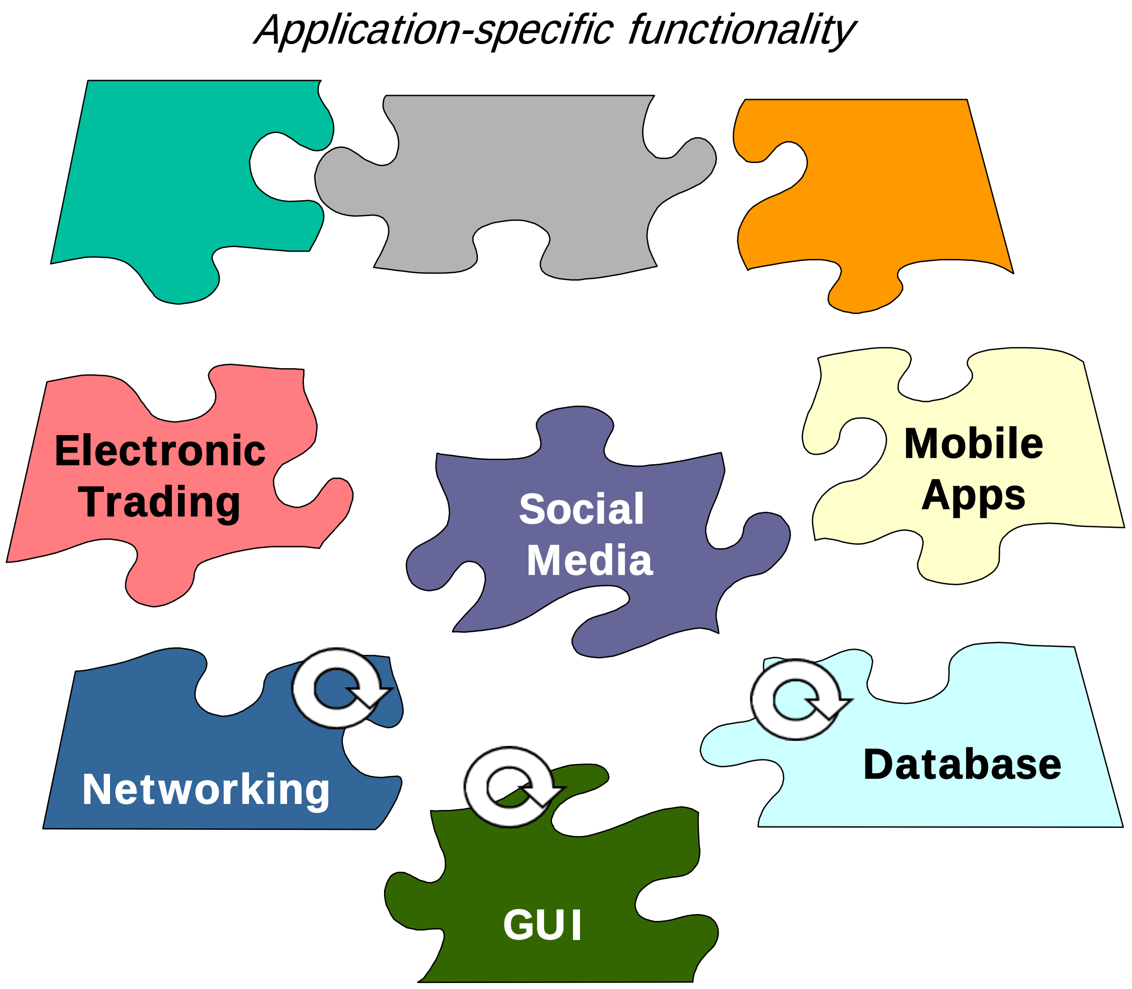
Overview of Frameworks

- They exhibit "inversion of control" via callbacks
- They provide integrated domain-specific structures & functionality



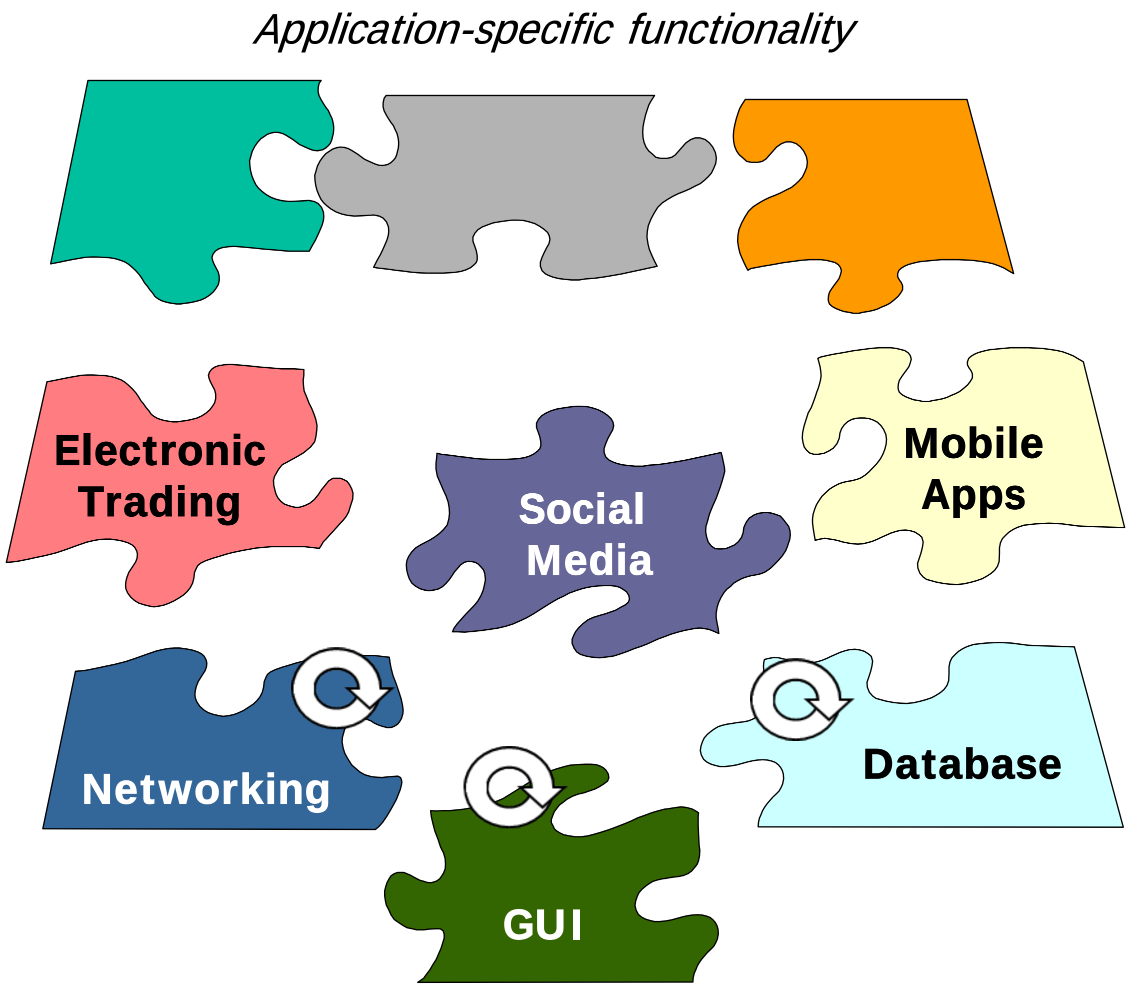
Overview of Frameworks

- They exhibit "inversion of control" via callbacks
- They provide integrated domain-specific structures & functionality
- They are "semi-complete" applications



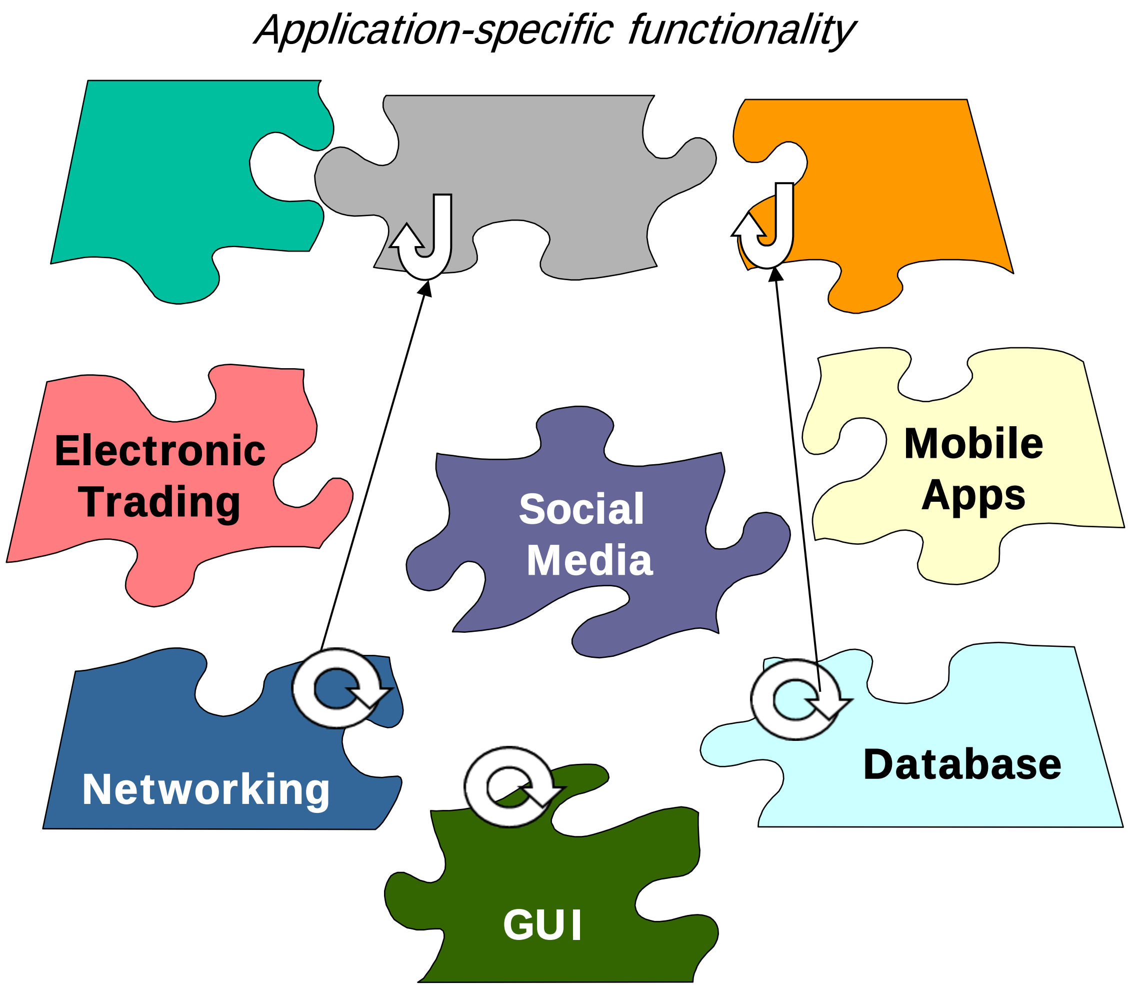
Overview of Frameworks

- They exhibit "inversion of control" via callbacks
- They provide integrated domain-specific structures & functionality
- They are "semi-complete" applications
  - Framework provides reusable interfaces & classes



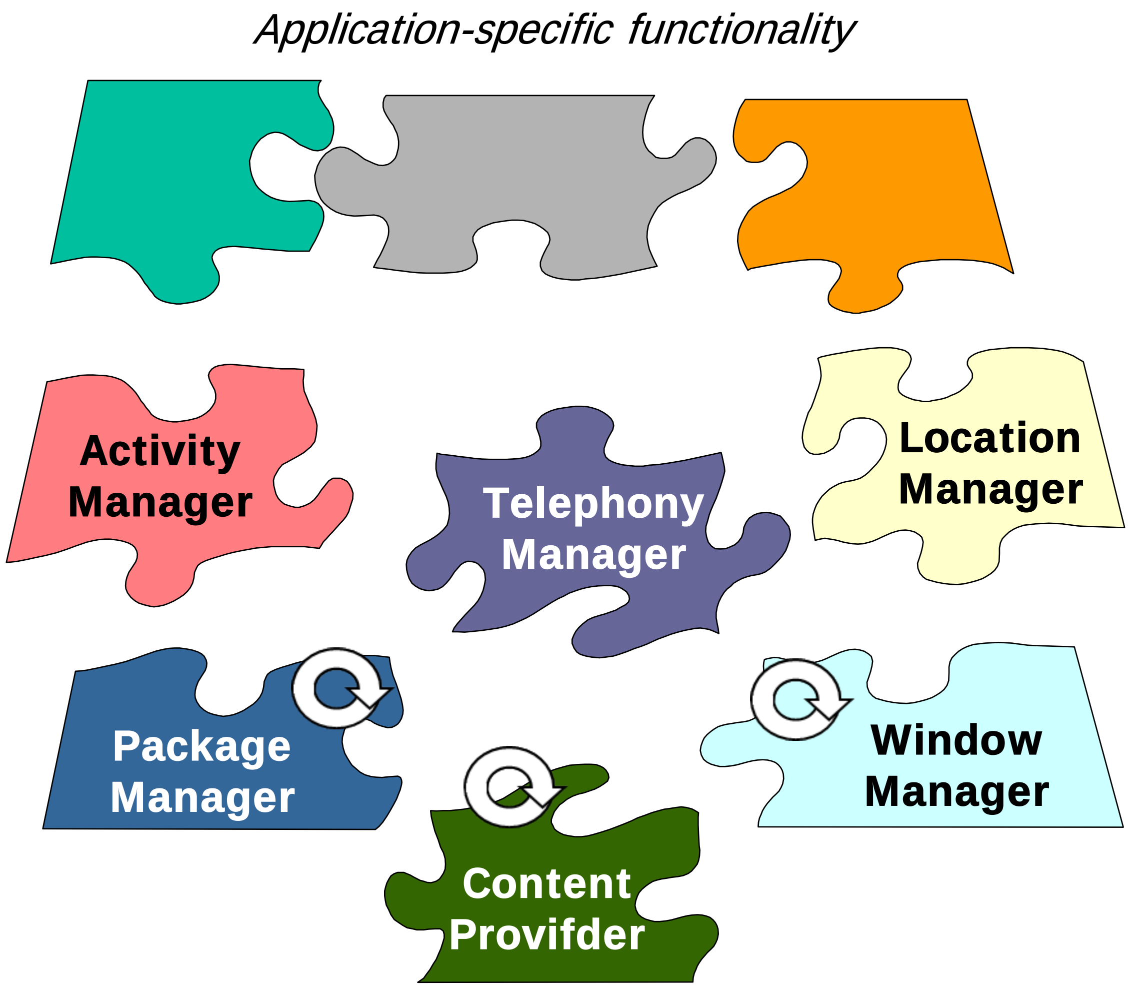
Overview of Frameworks

- They exhibit "inversion of control" via callbacks
- They provide integrated domain-specific structures & functionality
- They are "semi-complete" applications
  - Framework provides reusable interfaces & classes
  - Applications customize frameworks via subclassing & overriding hook methods



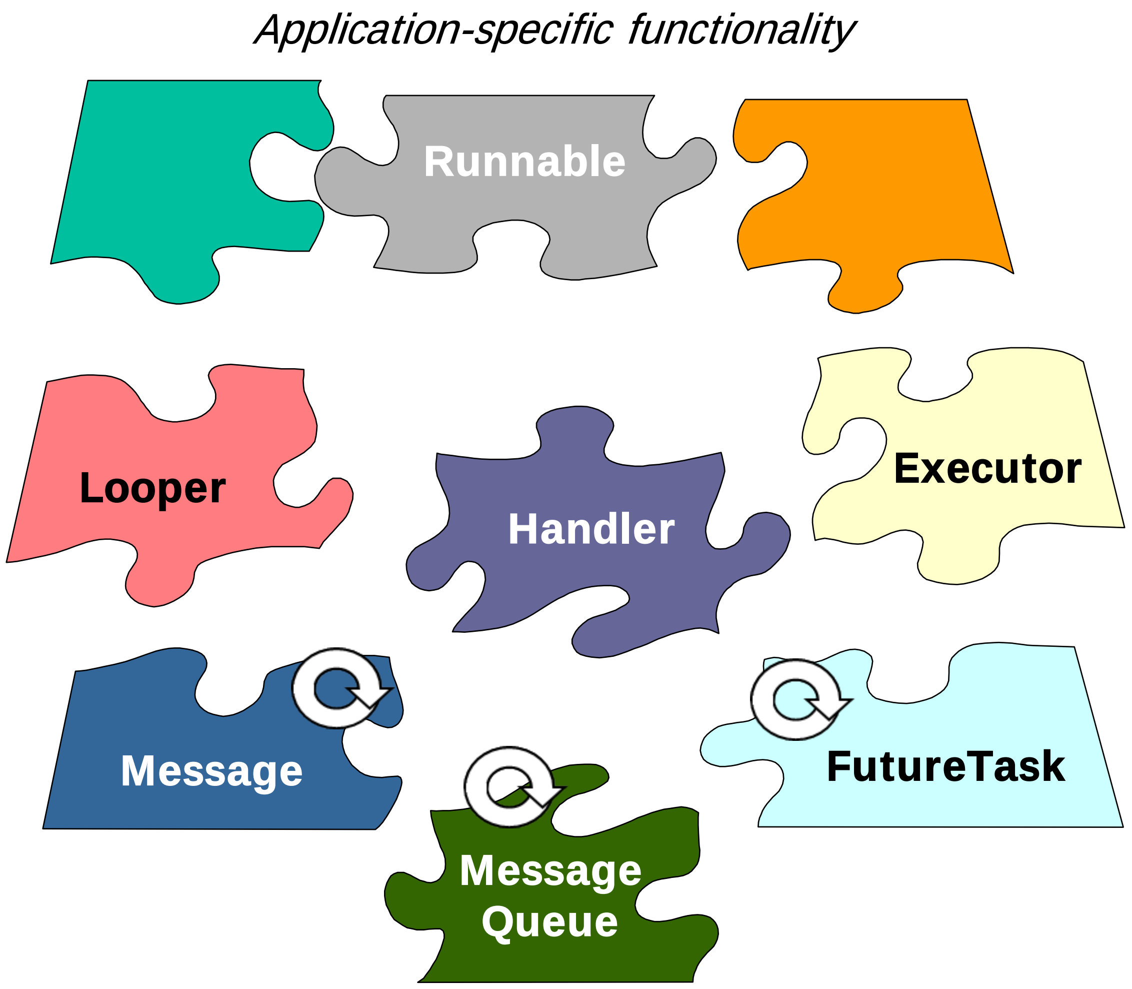
Overview of Frameworks

- They exhibit "inversion of control" via callbacks
- They provide integrated domain-specific structures & functionality
- They are "semi-complete" applications
- Android provides many frameworks



Overview of Frameworks

- They exhibit "inversion of control" via callbacks
- They provide integrated domain-specific structures & functionality
- They are "semi-complete" applications
- Android provides many frameworks
    - We focus on several Android concurrency & communication frameworks

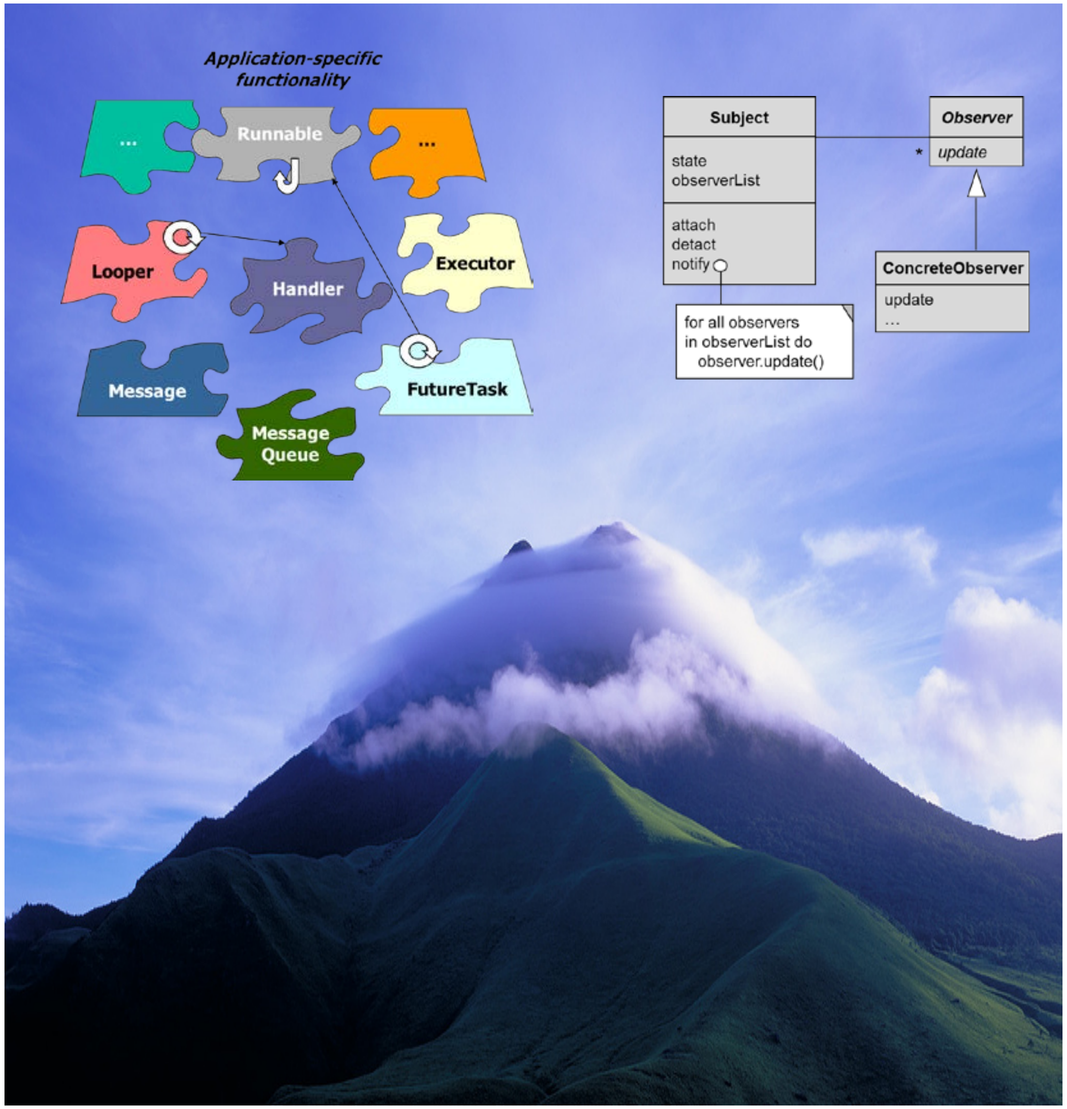


See item #17 at class.coursera.org/posa-002/wiki/FrequentlyAskedQuestions

Applying the Observer Pattern in Android Frameworks

Applying Observer in Android Frameworks

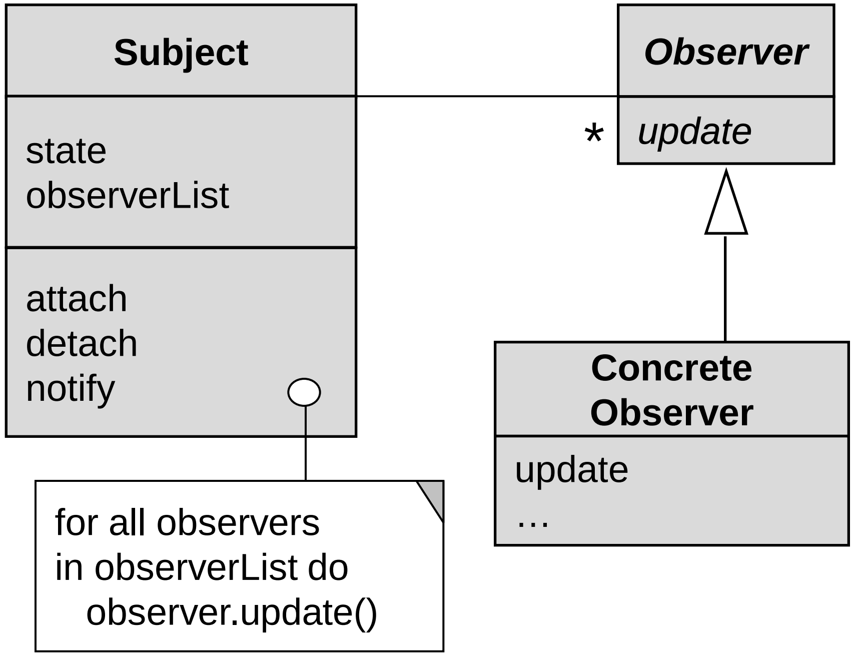
- The earlier overview of patterns & frameworks was intentionally high level



Applying Observer in Android Frameworks

- The earlier overview of patterns & frameworks was intentionally high level
- We now show examples of applying the *Observer* pattern to several Android frameworks

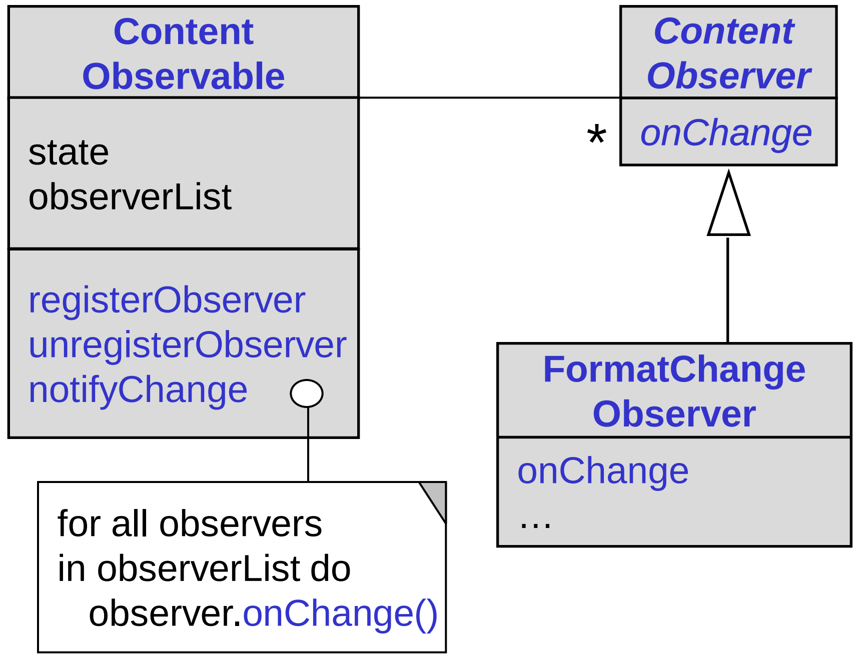
The Observer pattern



Android applies the *Observer* pattern extensive throughout its frameworks

Applying Observer in Android Frameworks

- The earlier overview of patterns & frameworks was intentionally high level
- We now show examples of applying the *Observer* pattern to several Android frameworks
  - The Content Provider & Resolver framework uses it to notify when rows in an SQLite database change

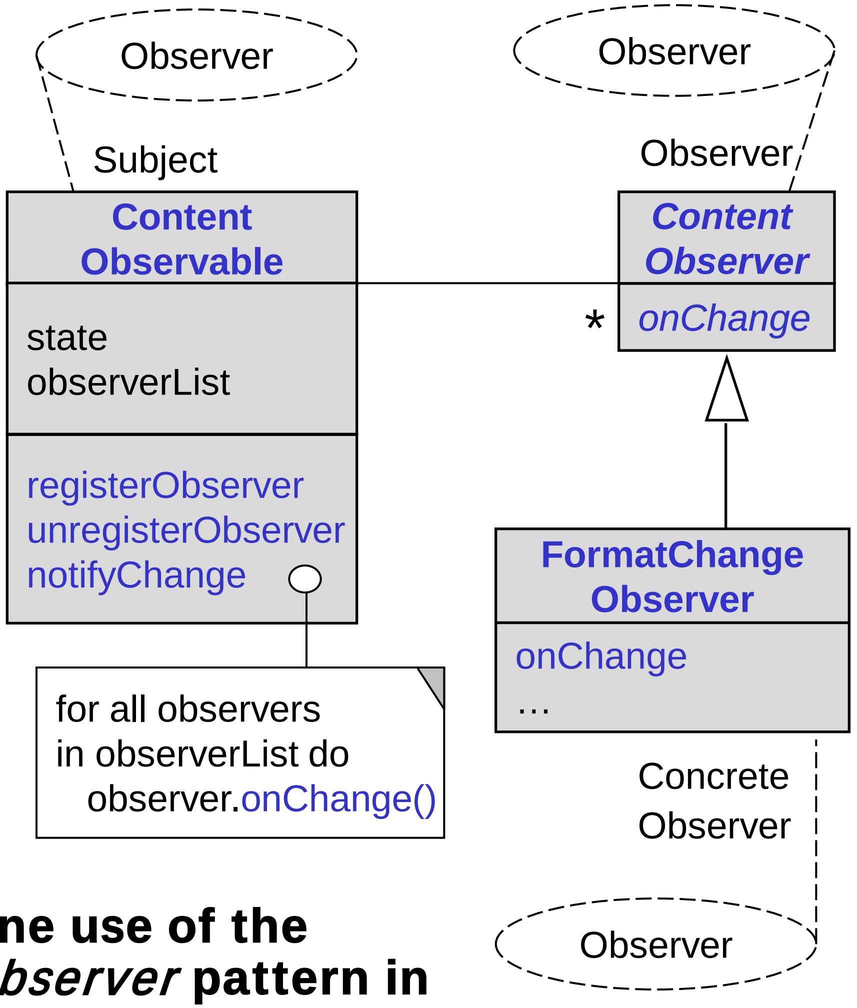


One use of the *Observer* pattern in Android

See www.grokkingandroid.com/use-contentobserver-to-listen-to-changes

Applying Observer in Android Frameworks

- The earlier overview of patterns & frameworks was intentionally high level
- We now show examples of applying the *Observer* pattern to several Android frameworks

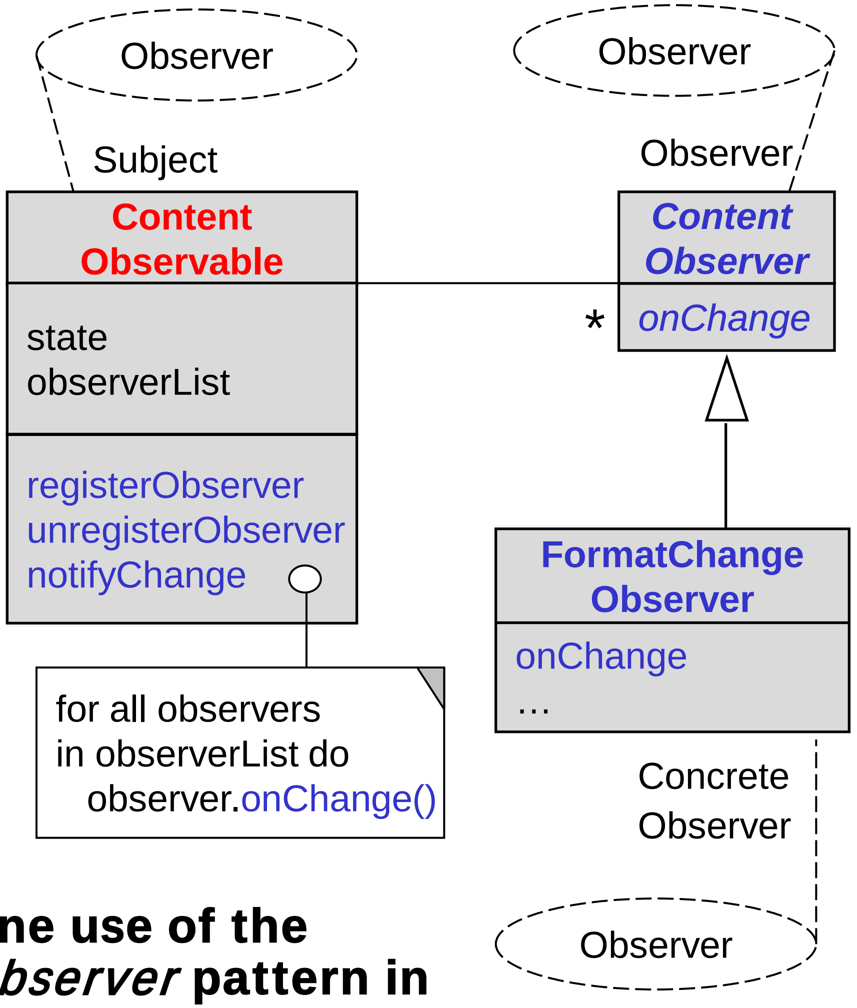


One use of the *Observer* pattern in Android

See packages/apps/DeskClock/src/com/android/deskclock/DigitalClock.java

Applying Observer in Android Frameworks

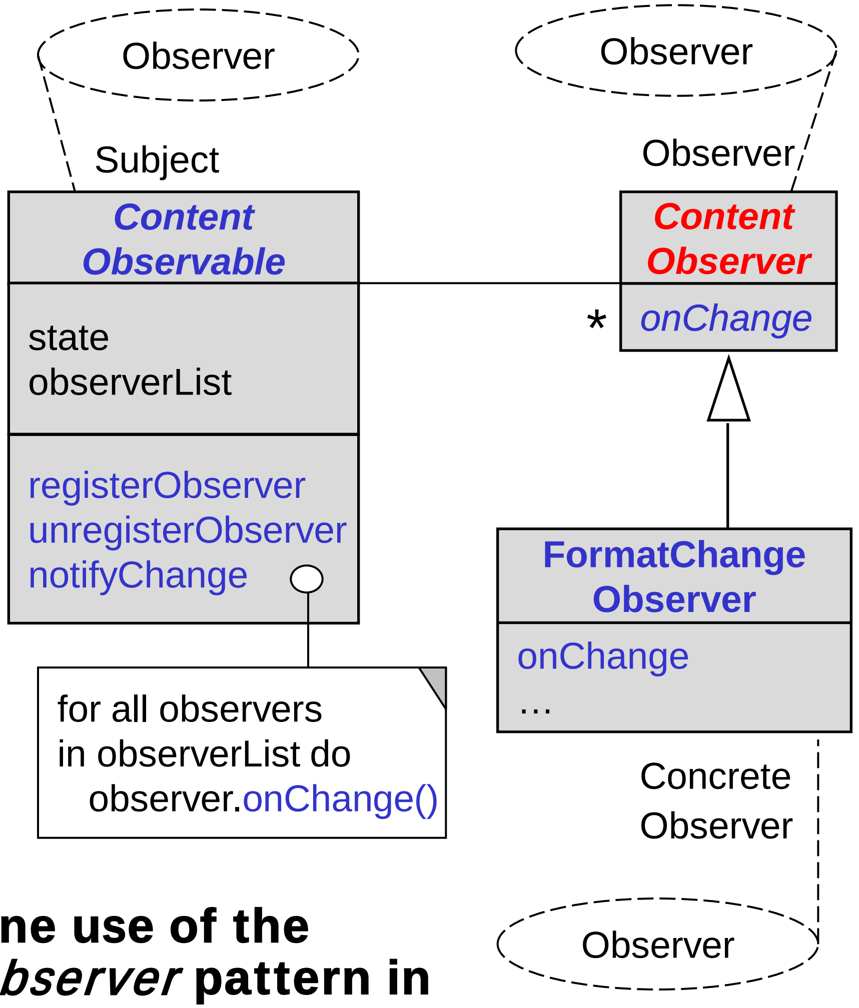
- The earlier overview of patterns & frameworks was intentionally high level
- We now show examples of applying the *Observer* pattern to several Android frameworks



One use of the *Observer* pattern in Android

Applying Observer in Android Frameworks

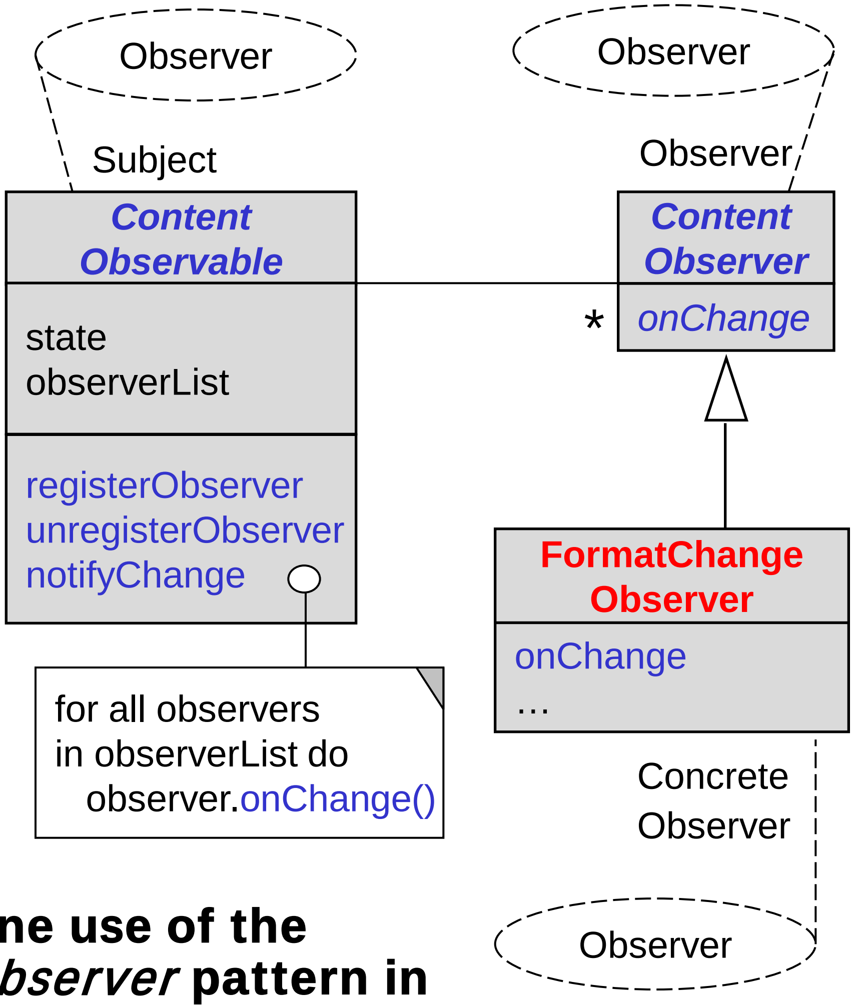
- The earlier overview of patterns & frameworks was intentionally high level
- We now show examples of applying the *Observer* pattern to several Android frameworks



One use of the *Observer* pattern in Android

Applying Observer in Android Frameworks

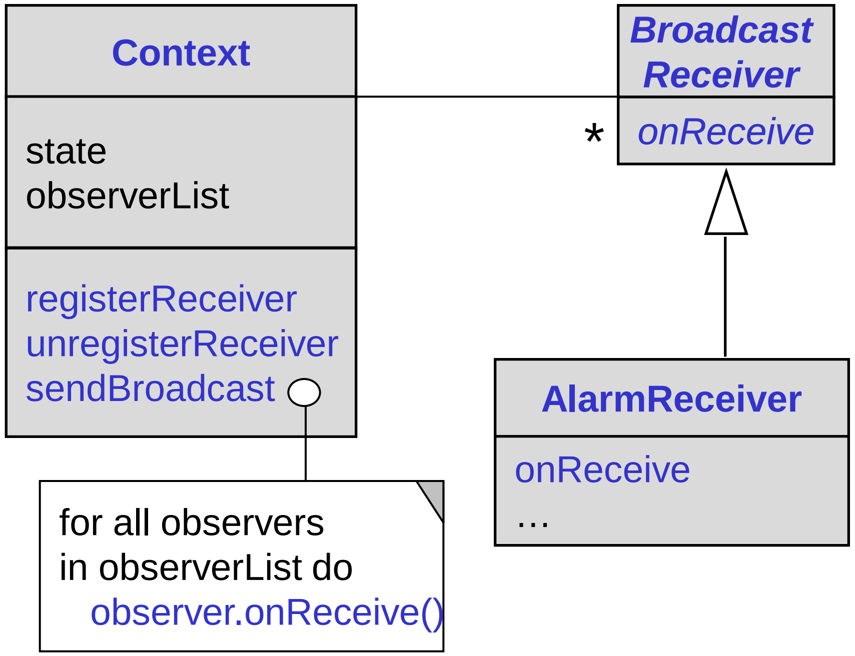
- The earlier overview of patterns & frameworks was intentionally high level
- We now show examples of applying the *Observer* pattern to several Android frameworks



One use of the *Observer* pattern in Android

Overview of Patterns

- The earlier overview of patterns & frameworks was intentionally high level
- We now show examples of applying the *Observer* pattern to several Android frameworks
  - The Content Provider & Resolver framework uses it to notify when rows in an SQLite database change
  - Its Intents framework uses it to notify Broadcast Receivers that events of interest have occurred

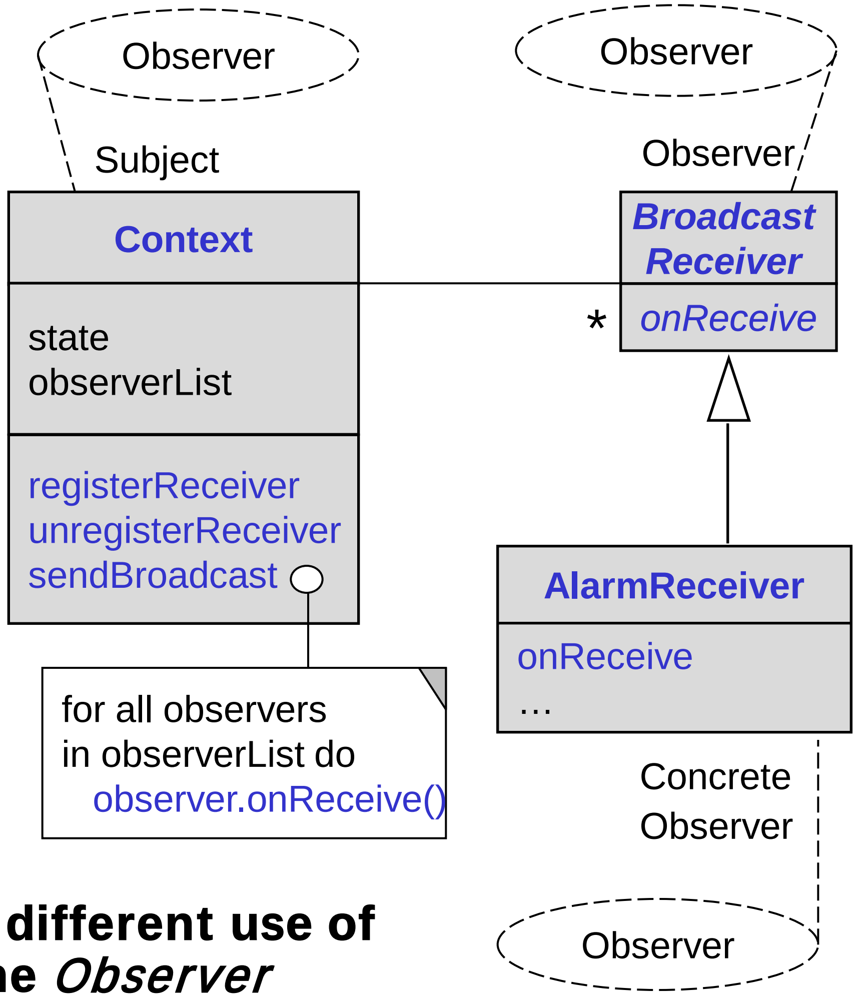


A different use of the *Observer* pattern in Android

See developer.android.com/guide/components/intents-filters.html

Overview of Patterns

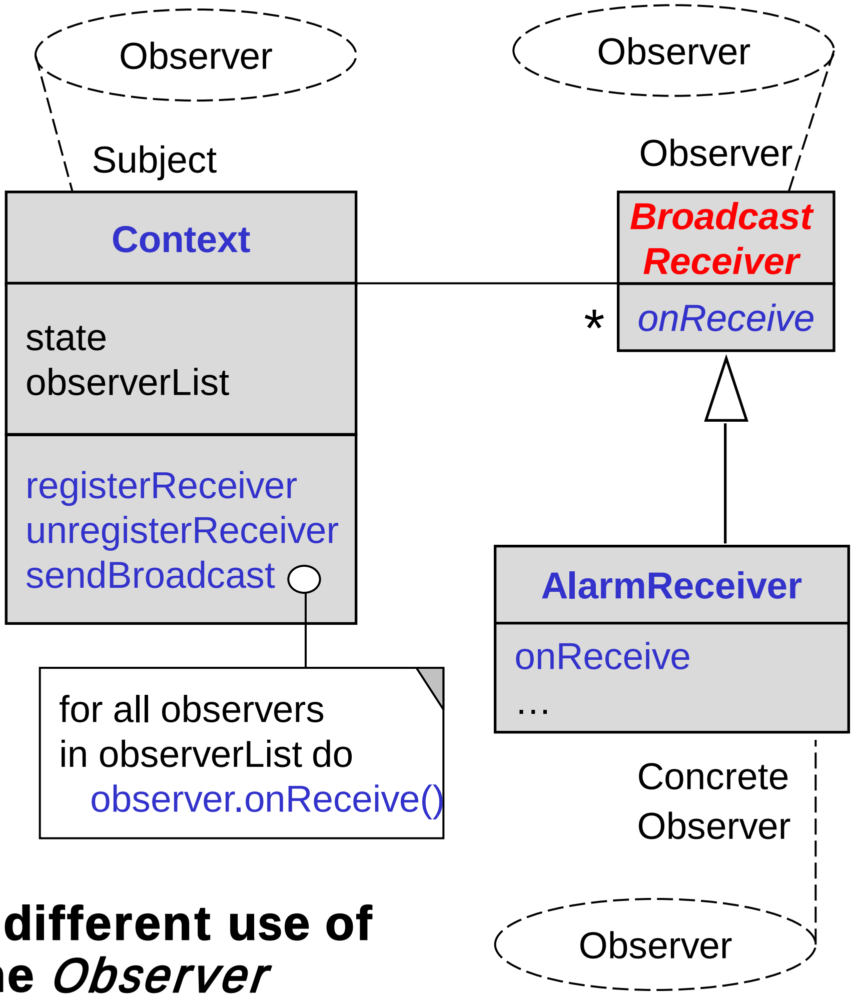
- The earlier overview of patterns & frameworks was intentionally high level
- We now show examples of applying the *Observer* pattern to several Android frameworks



A different use of the *Observer* pattern in Android

Overview of Patterns

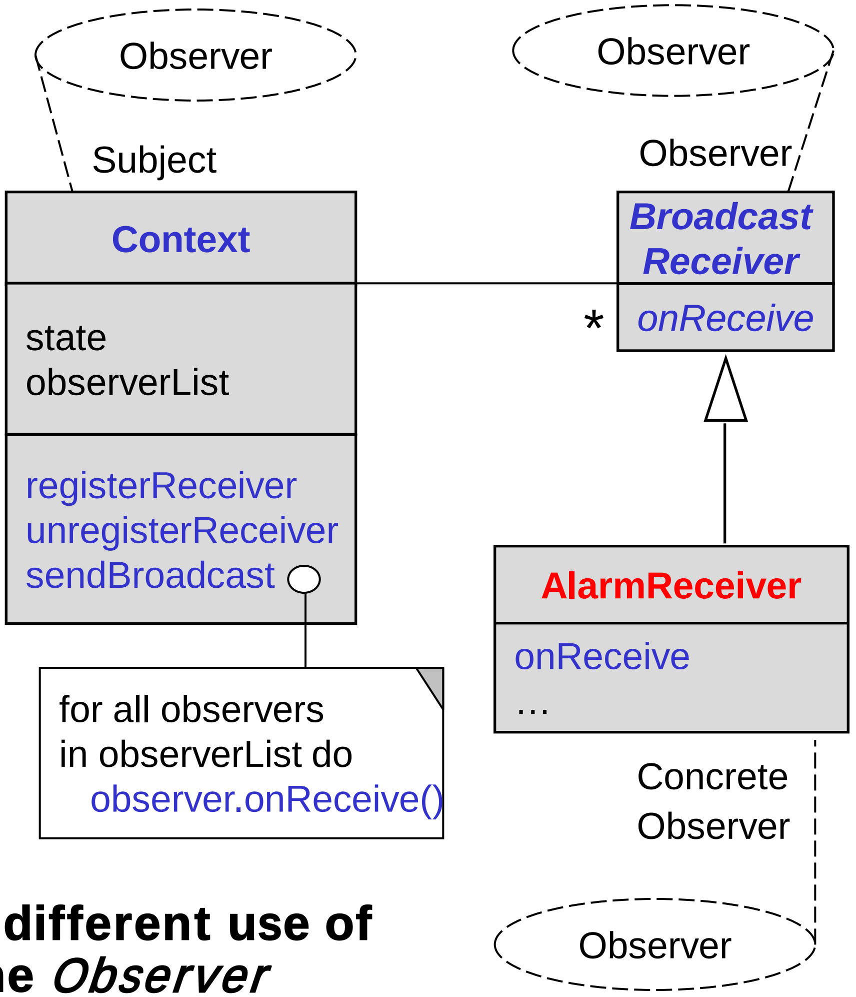
- The earlier overview of patterns & frameworks was intentionally high level
- We now show examples of applying the *Observer* pattern to several Android frameworks



A different use of the *Observer* pattern in Android

Overview of Patterns

- The earlier overview of patterns & frameworks was intentionally high level
- We now show examples of applying the *Observer* pattern to several Android frameworks

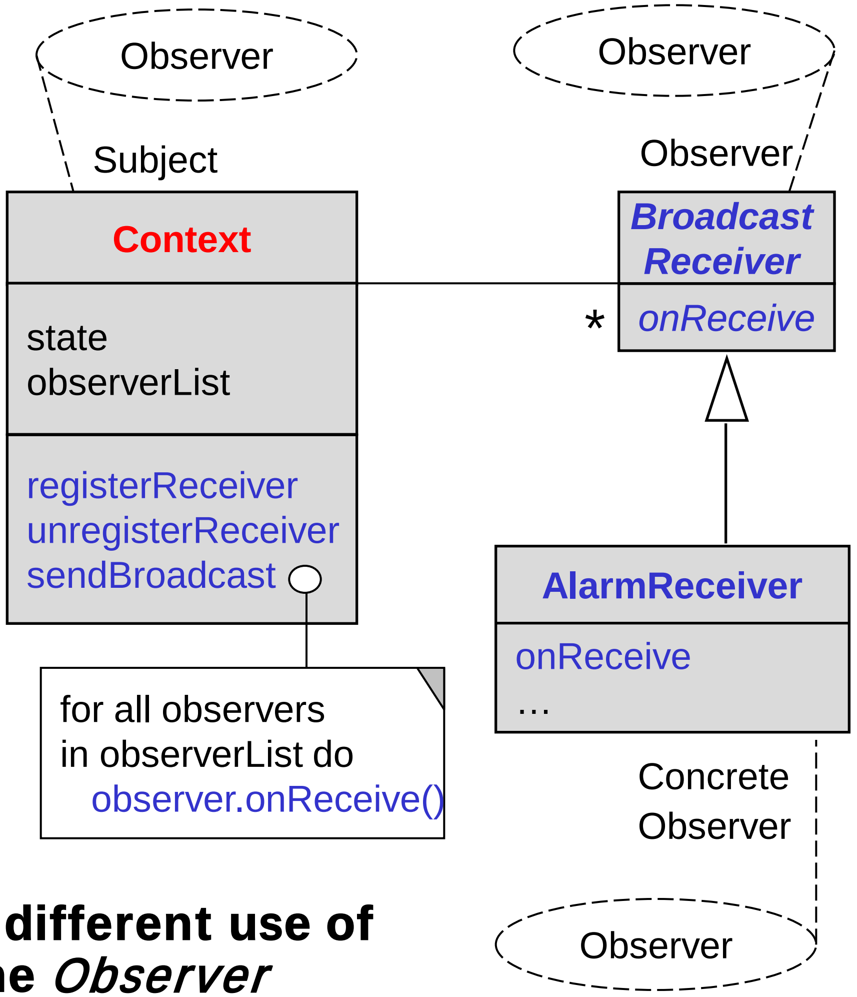


A different use of the *Observer* pattern in Android

See packages/apps/DeskClock/src/com/android/deskclock/AlarmReceiver.java

Overview of Patterns

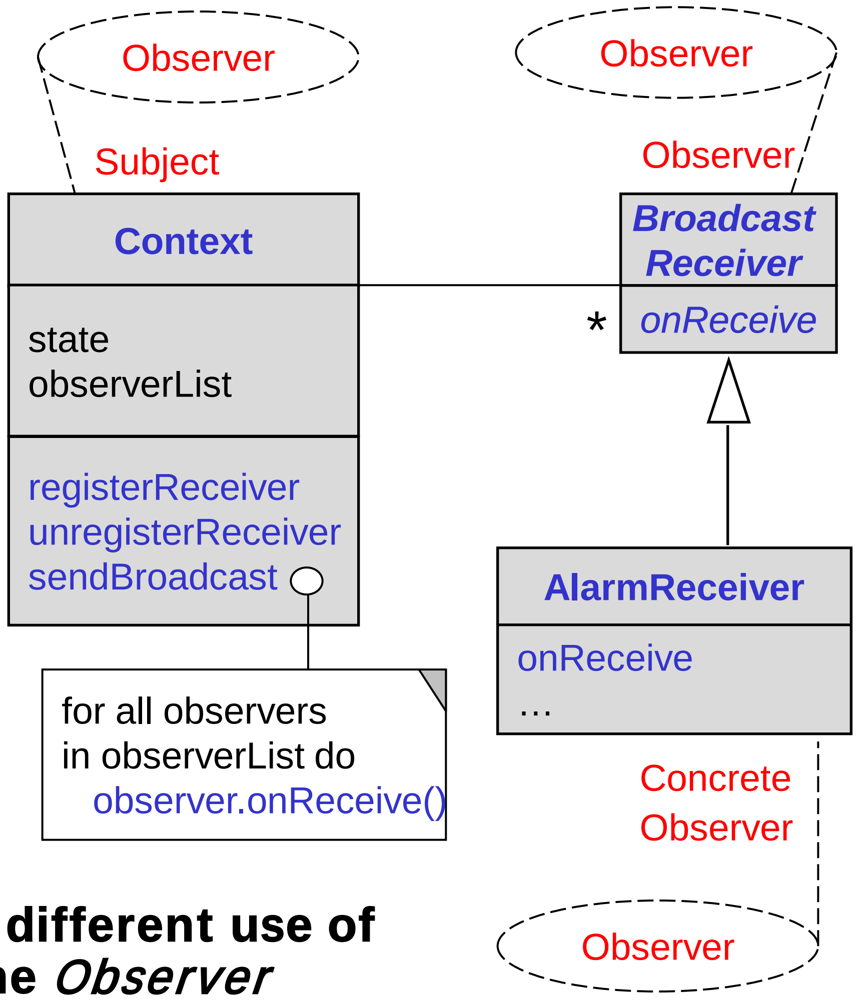
- The earlier overview of patterns & frameworks was intentionally high level
- We now show examples of applying the *Observer* pattern to several Android frameworks



A different use of the *Observer* pattern in Android

Overview of Patterns

- The earlier overview of patterns & frameworks was intentionally high level
- We now show examples of applying the *Observer* pattern to several Android frameworks

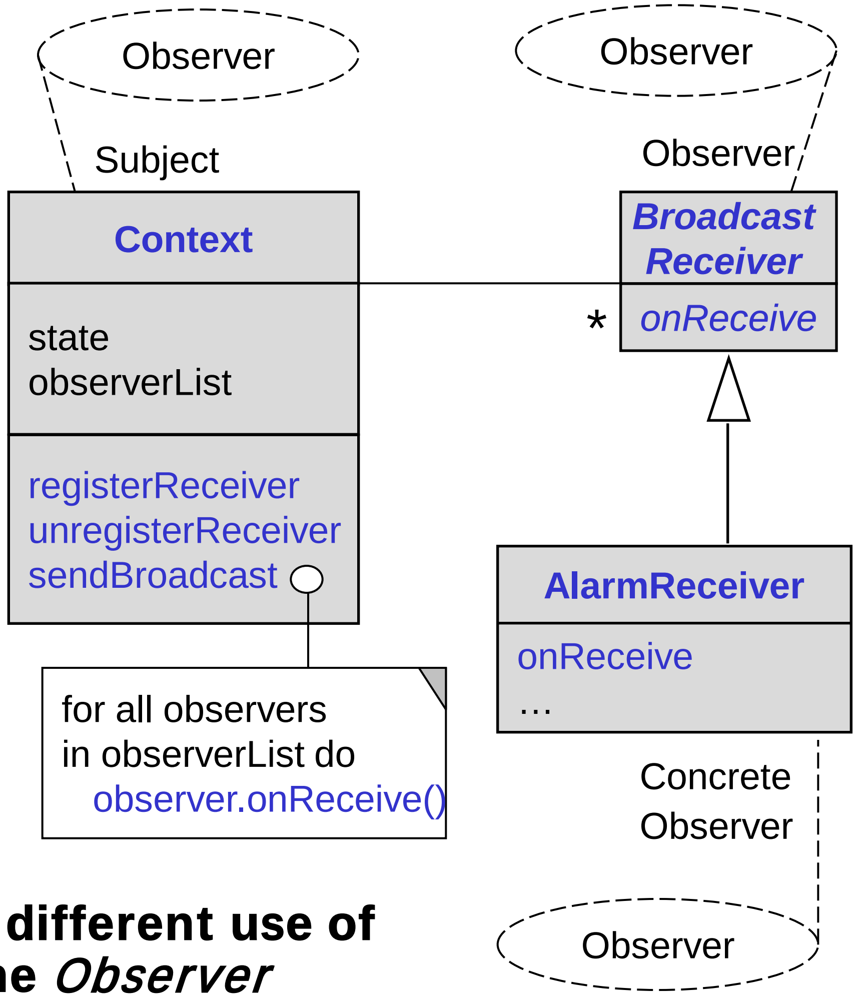


A different use of the *Observer* pattern in Android

Knowledge of *Observer* pattern reduces “surface area” of the design space

Overview of Patterns

- The earlier overview of patterns & frameworks was intentionally high level
- We now show examples of applying the *Observer* pattern to several Android frameworks



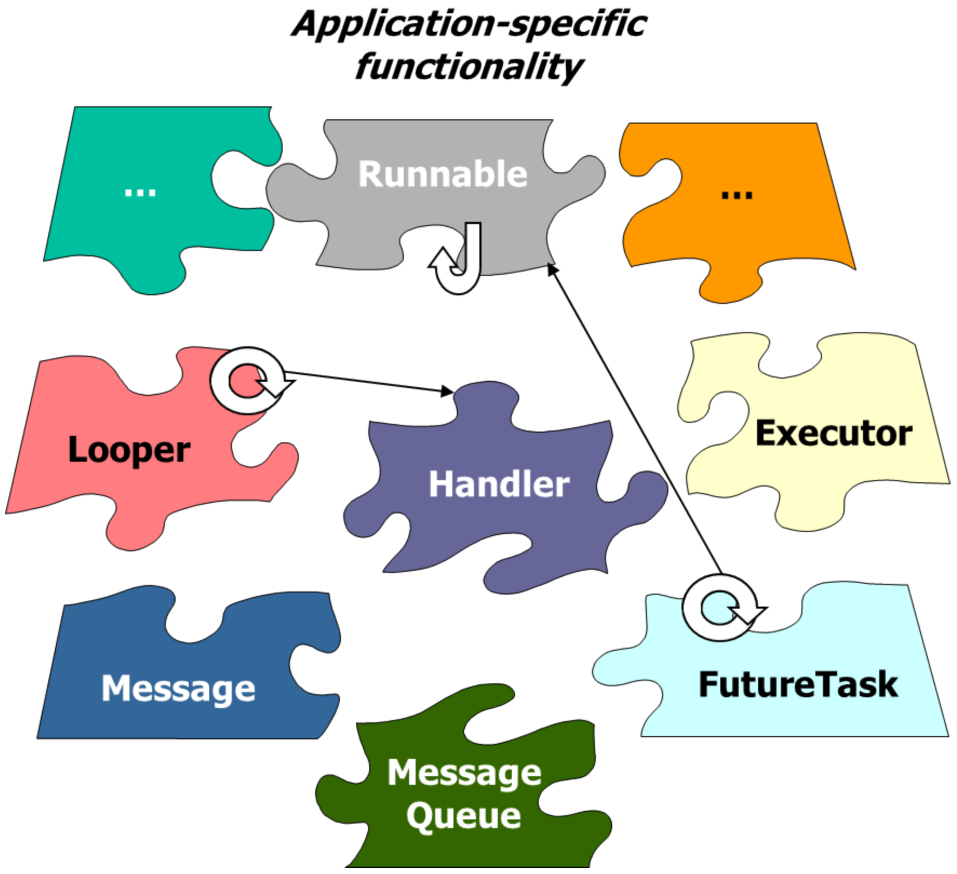
A different use of the *Observer* pattern in Android

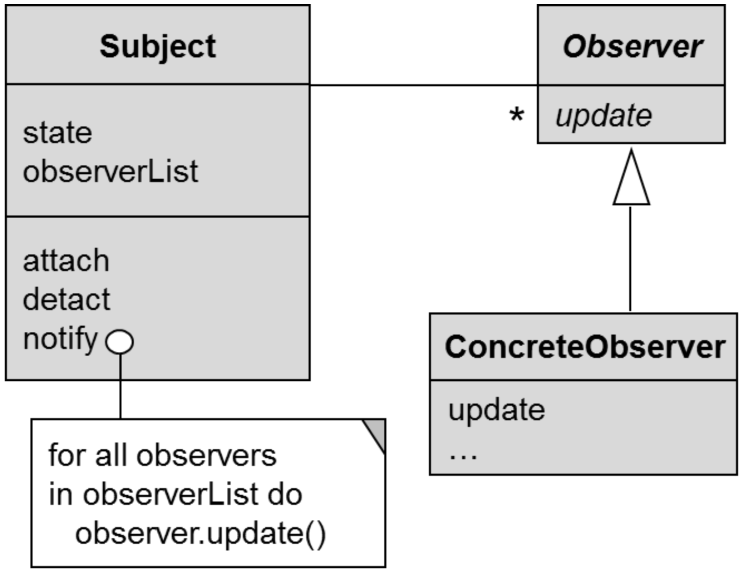
These *Observer* implementations also demonstrate framework characteristics

Summary

Summary

- Patterns & frameworks enhance *systematic reuse* of software





The *Observer* pattern

Summary

- Patterns & frameworks enhance *systematic reuse* of software
  - Create or acquire reusable assets & then consistently use & evolve them

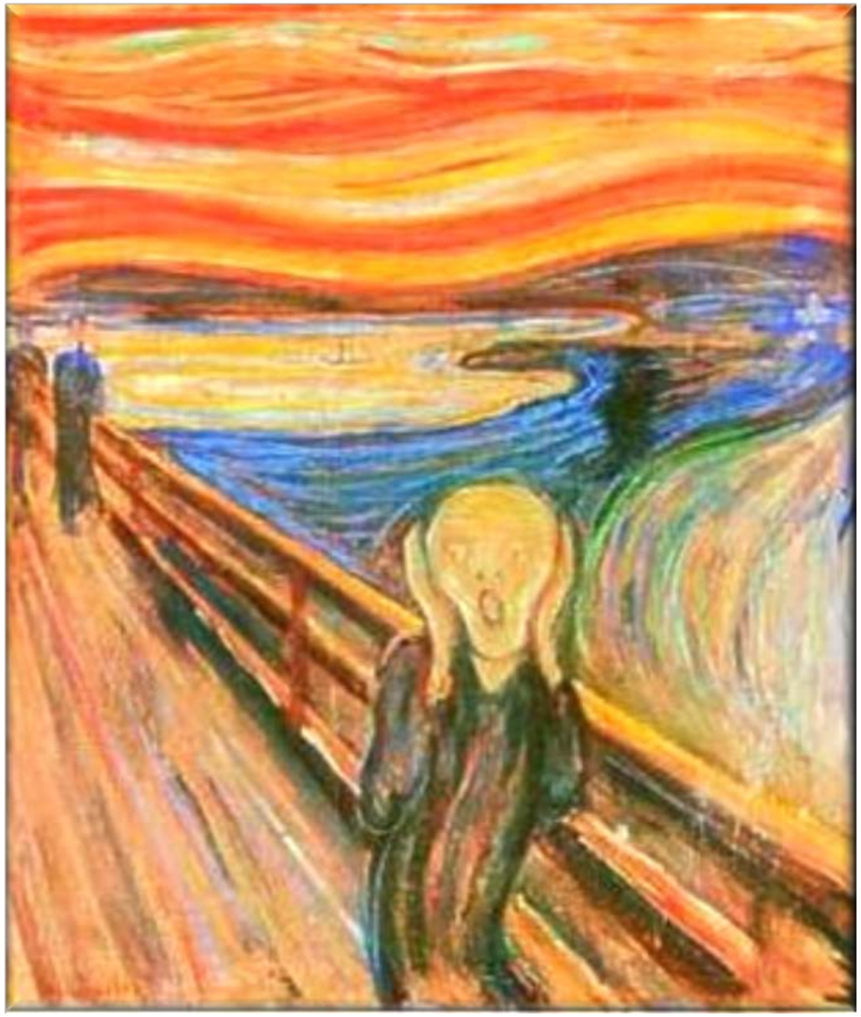
martin.griss.com/pubs/fusion1.htm has more on systematic reuse

Summary

- Patterns & frameworks enhance *systematic reuse* of software
- In contrast to opportunistic reuse, which involves "cutting & pasting"

Summary

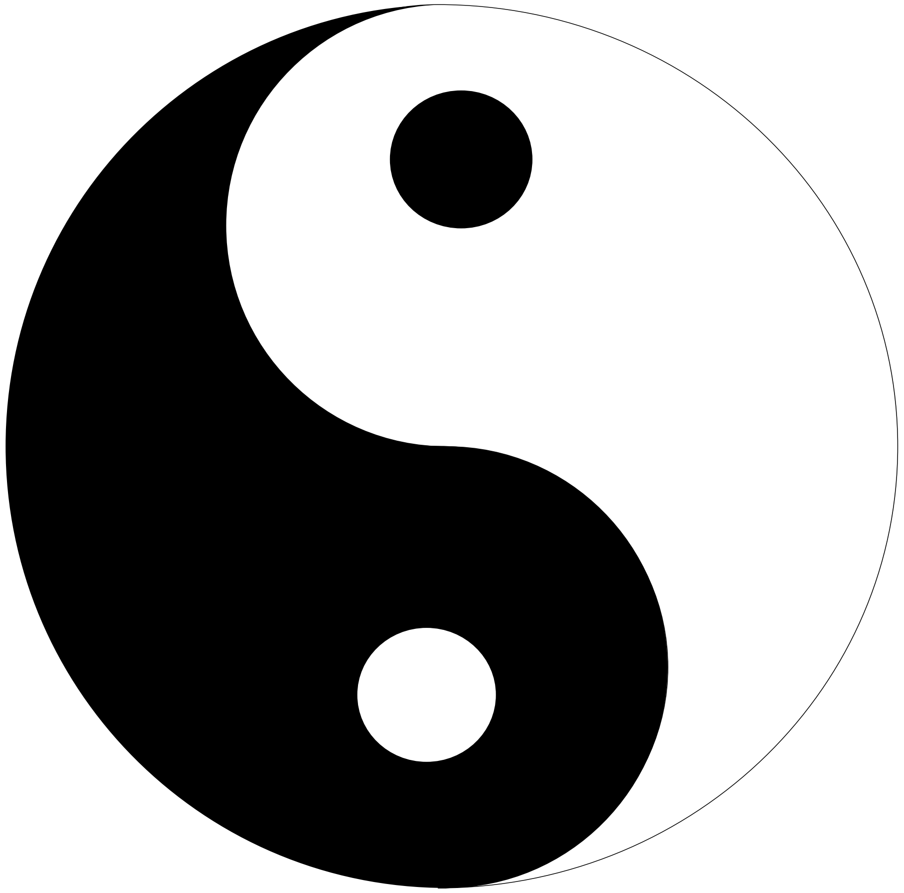
- Patterns & frameworks enhance *systematic reuse* of software
- In contrast to opportunistic reuse, which involves "cutting & pasting"
  - Yields many code forks that are hard to evolve & sustain over time



www.peachpit.com/articles/article.aspx?p=29753&seqNum=4 has more

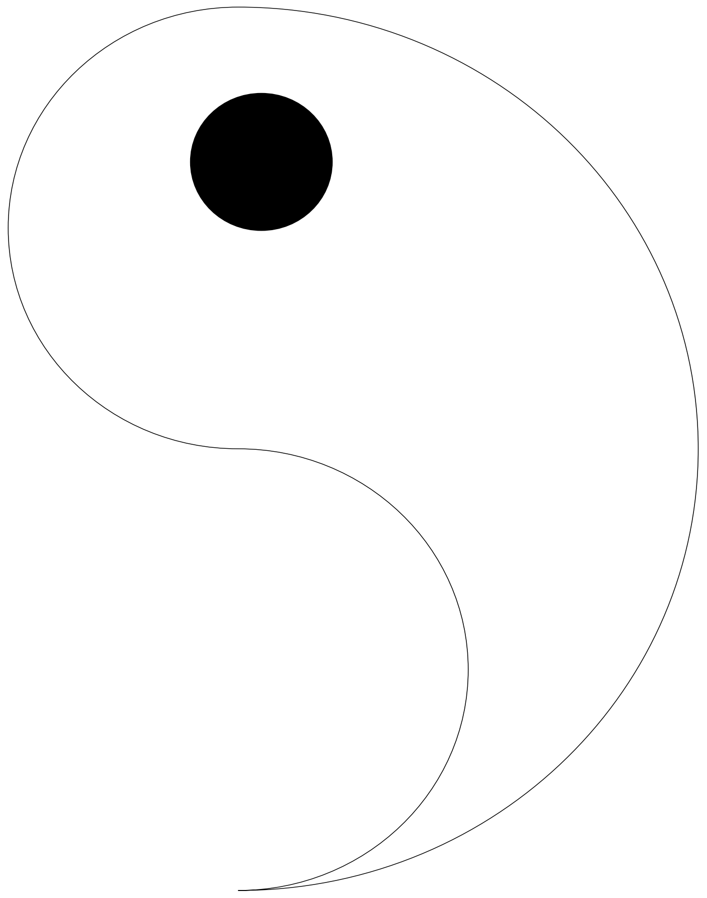
Summary

- Patterns & frameworks enhance *systematic reuse* of software
- In contrast to opportunistic reuse, which involves "cutting & pasting"
- Systematic reuse involves both design ***and*** code reuse



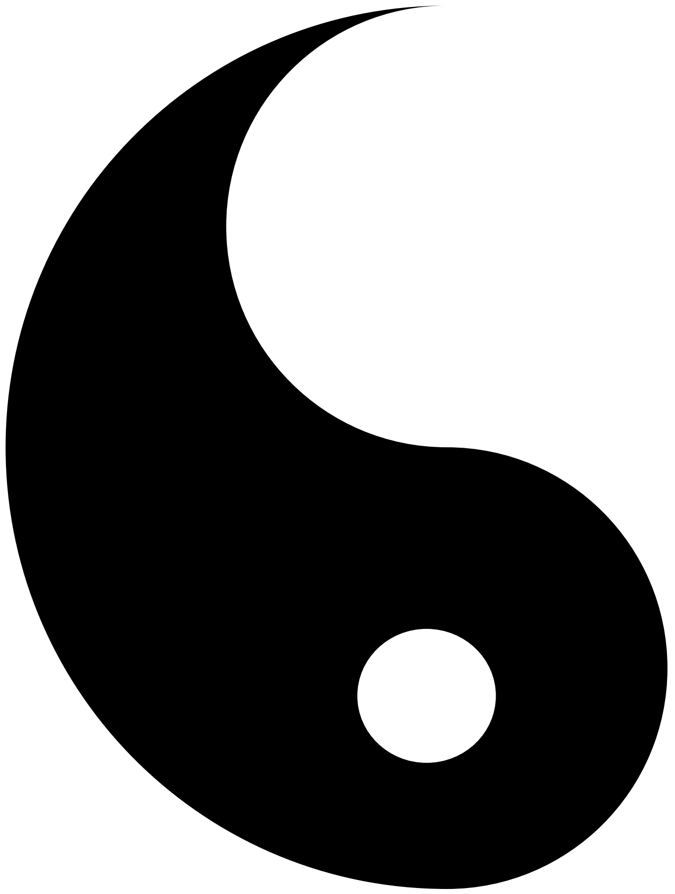
Summary

- Patterns & frameworks enhance *systematic reuse* of software
- In contrast to opportunistic reuse, which involves "cutting & pasting"
- Systematic reuse involves both design ***and*** code reuse
  - **Design reuse** – Match problems to relevant structures/dynamics & patterns in a domain



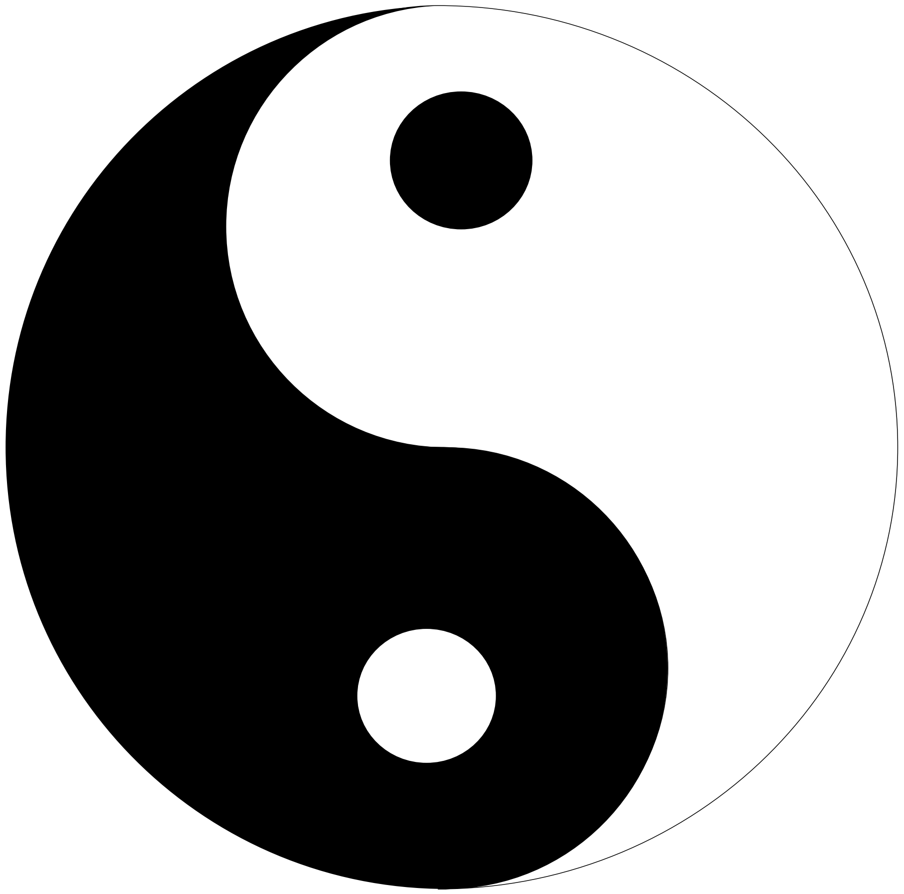
Summary

- Patterns & frameworks enhance *systematic reuse* of software
- In contrast to opportunistic reuse, which involves "cutting & pasting"
- Systematic reuse involves both design ***and*** code reuse
  - **Design reuse** – Match problems to relevant structures/dynamics & patterns in a domain
  - **Code reuse** – Reify proven designs within particular development environments & domains



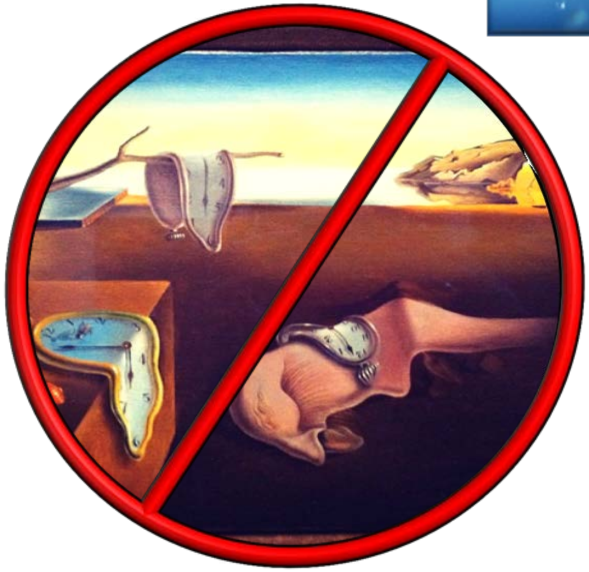
Summary

- Patterns & frameworks enhance *systematic reuse* of software
- In contrast to opportunistic reuse, which involves "cutting & pasting"
- Systematic reuse involves both design *and* code reuse
- Together, patterns & frameworks avoid rediscovering & reinventing solutions



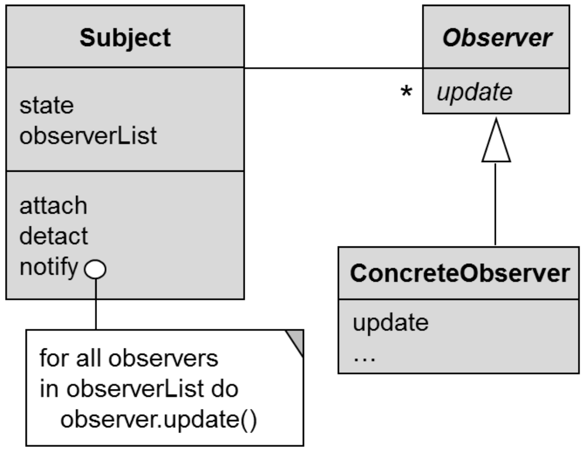
Summary

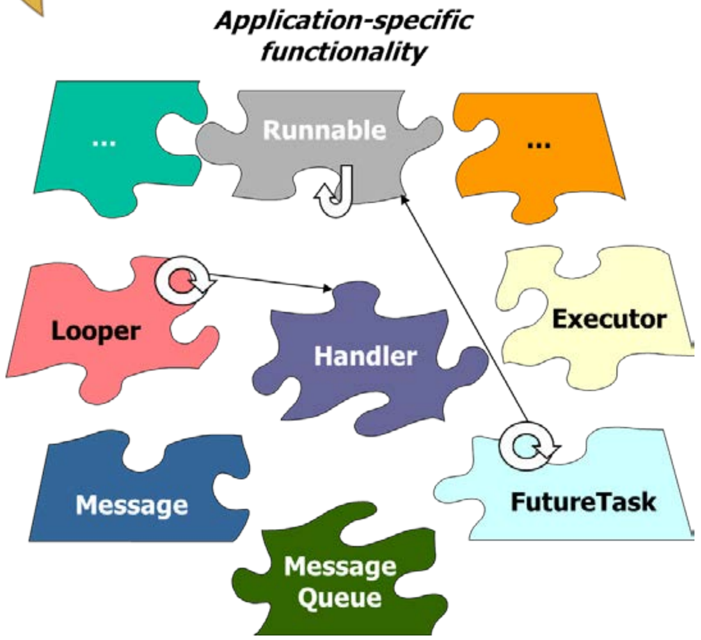
- Patterns & frameworks enhance *systematic reuse* of software
- In contrast to opportunistic reuse, which involves "cutting & pasting"
- Systematic reuse involves both design *and* code reuse
- Together, patterns & frameworks avoid rediscovering & reinventing solutions
  - Helps save time & money



Summary

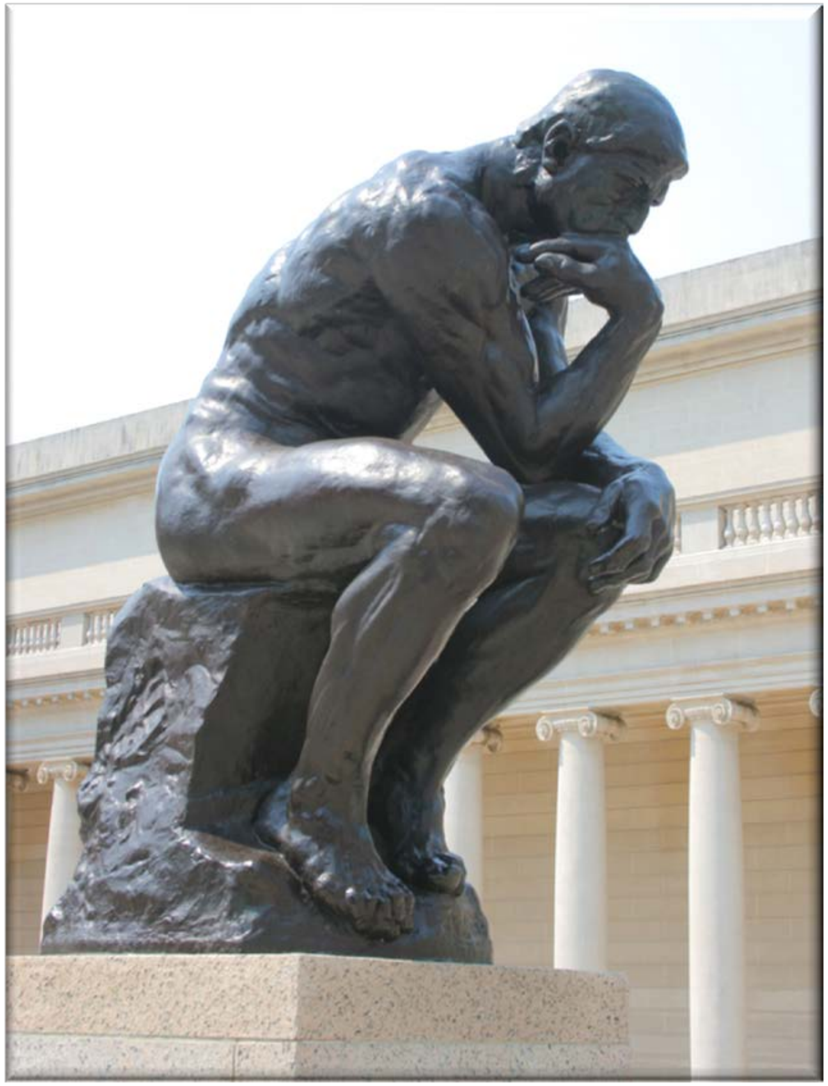
- Patterns & frameworks enhance *systematic reuse* of software
- In contrast to opportunistic reuse, which involves "cutting & pasting"
- Systematic reuse involves both design *and* code reuse
- Together, patterns & frameworks avoid rediscovering & reinventing solutions
  - Helps save time & money
  - Improves quality over the lifecycle





Summary

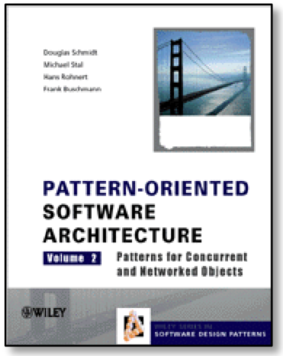
- Patterns & frameworks enhance *systematic reuse* of software
- In contrast to opportunistic reuse, which involves “cutting & pasting”
- Systematic reuse involves both design *and* code reuse
- Together, patterns & frameworks avoid rediscovering & reinventing solutions
- Patterns & frameworks are deep technical topics

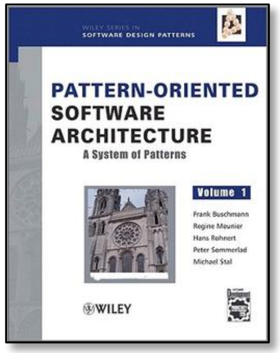


Summary

- Patterns & frameworks enhance *systematic reuse* of software
- In contrast to opportunistic reuse, which involves "cutting & pasting"
- Systematic reuse involves both design *and* code reuse
- Together, patterns & frameworks avoid rediscovering & reinventing solutions
- Patterns & frameworks are deep technical topics
- We just cover patterns & frameworks that pertain to Android concurrency & communication mechanisms

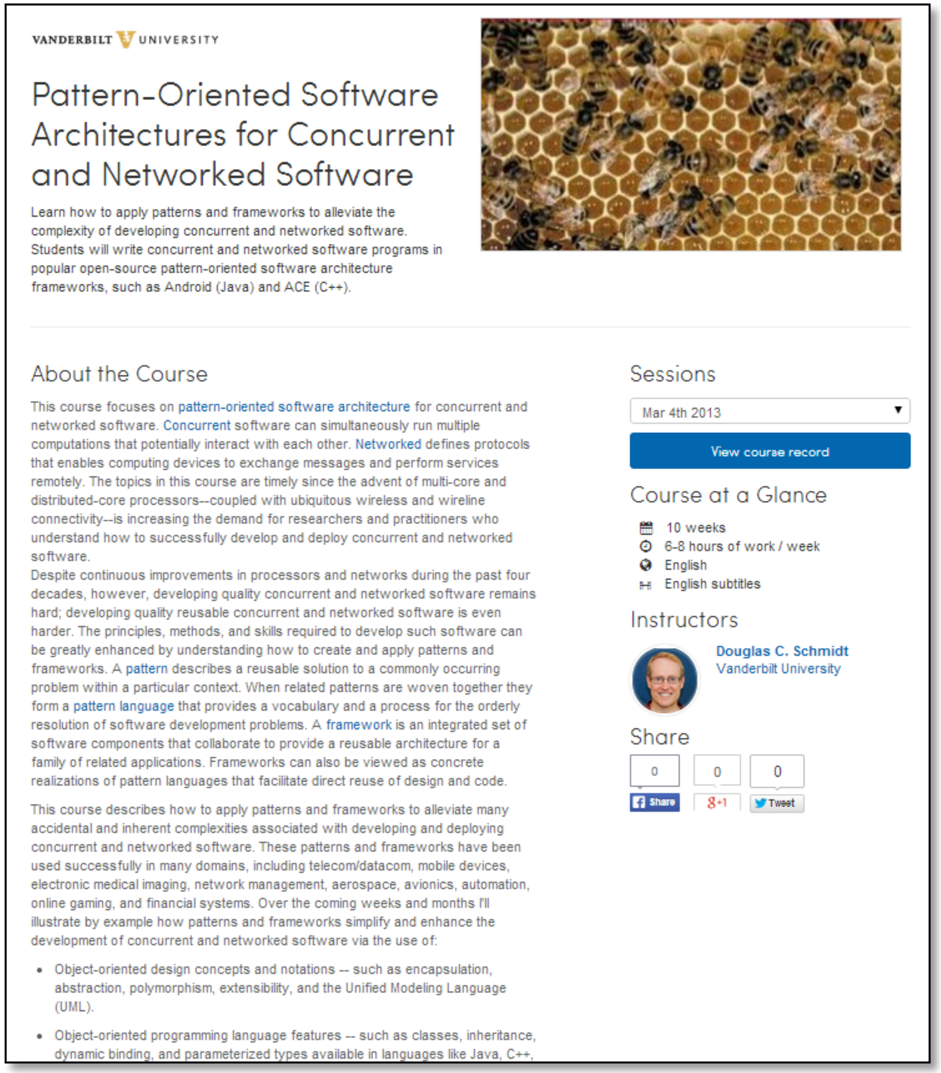
Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES


Douglas Schmidt
Michael Stal
Hans Rohnert
Frank Buschmann
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
Volume 2
Patterns for Concurrent and Networked Objects
WILEY
WILEY SERIES IN SOFTWARE DESIGN PATTERNS



Summary

- Patterns & frameworks enhance *systematic reuse* of software
- In contrast to opportunistic reuse, which involves "cutting & pasting"
- Systematic reuse involves both design *and* code reuse
- Together, patterns & frameworks avoid rediscovering & reinventing solutions
- Patterns & frameworks are deep technical topics
- We just cover patterns & frameworks that pertain to Android concurrency & communication mechanisms
- More available in 2013 POSA MOOC



VANDERBILT UNIVERSITY

Pattern-Oriented Software Architectures for Concurrent and Networked Software

Learn how to apply patterns and frameworks to alleviate the complexity of developing concurrent and networked software. Students will write concurrent and networked software programs in popular open-source pattern-oriented software architecture frameworks, such as Android (Java) and ACE (C++).

About the Course

This course focuses on pattern-oriented software architecture for concurrent and networked software. Concurrent software can simultaneously run multiple computations that potentially interact with each other. Networked defines protocols that enables computing devices to exchange messages and perform services remotely. The topics in this course are timely since the advent of multi-core and distributed-core processors--coupled with ubiquitous wireless and wireline connectivity--is increasing the demand for researchers and practitioners who understand how to successfully develop and deploy concurrent and networked software.
Despite continuous improvements in processors and networks during the past four decades, however, developing quality concurrent and networked software remains hard; developing quality reusable concurrent and networked software is even harder. The principles, methods, and skills required to develop such software can be greatly enhanced by understanding how to create and apply patterns and frameworks. A pattern describes a reusable solution to a commonly occurring problem within a particular context. When related patterns are woven together they form a pattern language that provides a vocabulary and a process for the orderly resolution of software development problems. A framework is an integrated set of software components that collaborate to provide a reusable architecture for a family of related applications. Frameworks can also be viewed as concrete realizations of pattern languages that facilitate direct reuse of design and code.

This course describes how to apply patterns and frameworks to alleviate many accidental and inherent complexities associated with developing and deploying concurrent and networked software. These patterns and frameworks have been used successfully in many domains, including telecom/datacom, mobile devices, electronic medical imaging, network management, aerospace, avionics, automation, online gaming, and financial systems. Over the coming weeks and months I'll illustrate by example how patterns and frameworks simplify and enhance the development of concurrent and networked software via the use of:

- Object-oriented design concepts and notations -- such as encapsulation, abstraction, polymorphism, extensibility, and the Unified Modeling Language (UML).
- Object-oriented programming language features -- such as classes, inheritance, dynamic binding, and parameterized types available in languages like Java, C++,

Sessions

Mar 4th 2013 ▾

View course record

Course at a Glance

- 10 weeks
- 6-8 hours of work / week
- English
- English subtitles

Instructors

Douglas C. Schmidt
Vanderbilt University

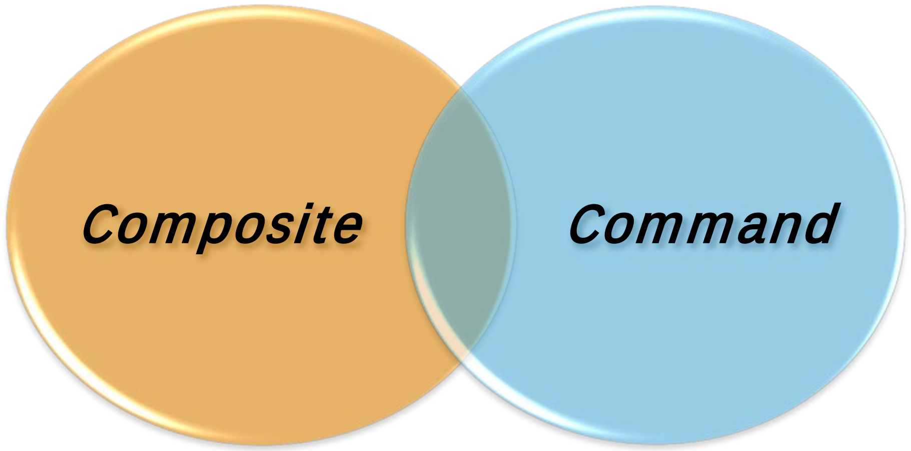
Share

0 Share 0 +1 0 Tweet

See item #22 at class.coursera.org/posa-002/wiki/FrequentlyAskedQuestions

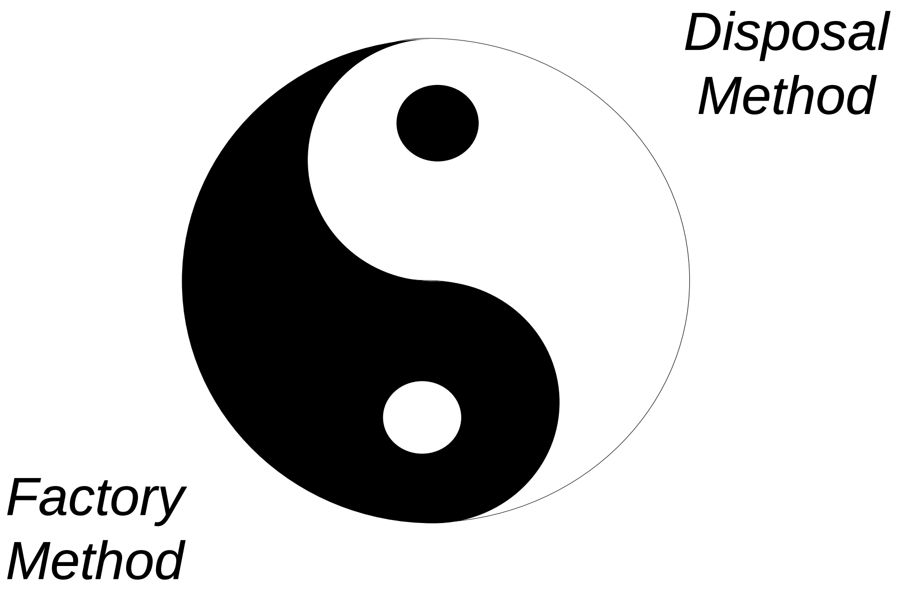
Summary

- Patterns & frameworks enhance *systematic reuse* of software
- In contrast to opportunistic reuse, which involves "cutting & pasting"
- Systematic reuse involves both design ***and*** code reuse
- Together, patterns & frameworks avoid rediscovering & reinventing solutions
- Patterns & frameworks are deep technical topics
- We just cover patterns & frameworks that pertain to Android concurrency & communication mechanisms
- More available in 2013 POSA MOOC



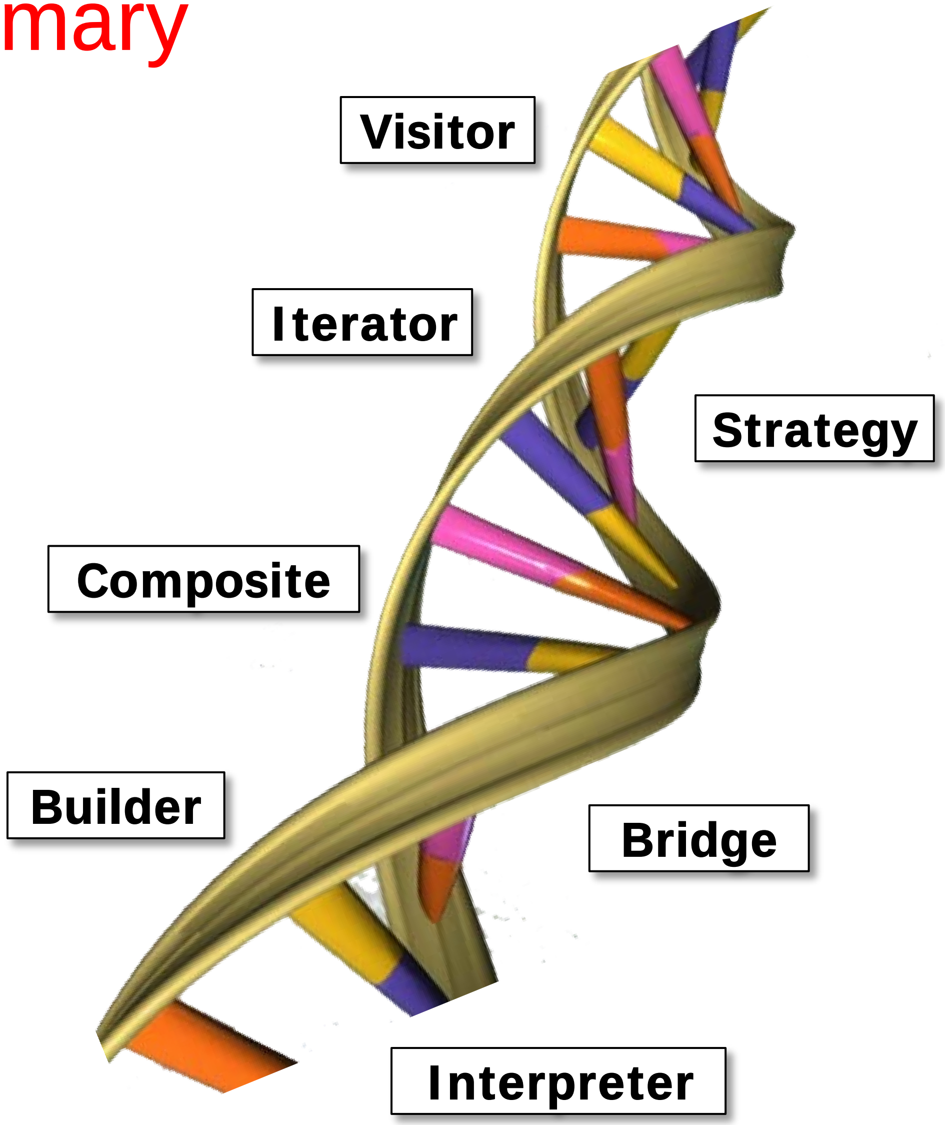
Summary

- Patterns & frameworks enhance *systematic reuse* of software
- In contrast to opportunistic reuse, which involves "cutting & pasting"
- Systematic reuse involves both design ***and*** code reuse
- Together, patterns & frameworks avoid rediscovering & reinventing solutions
- Patterns & frameworks are deep technical topics
- We just cover patterns & frameworks that pertain to Android concurrency & communication mechanisms
- More available in 2013 POSA MOOC



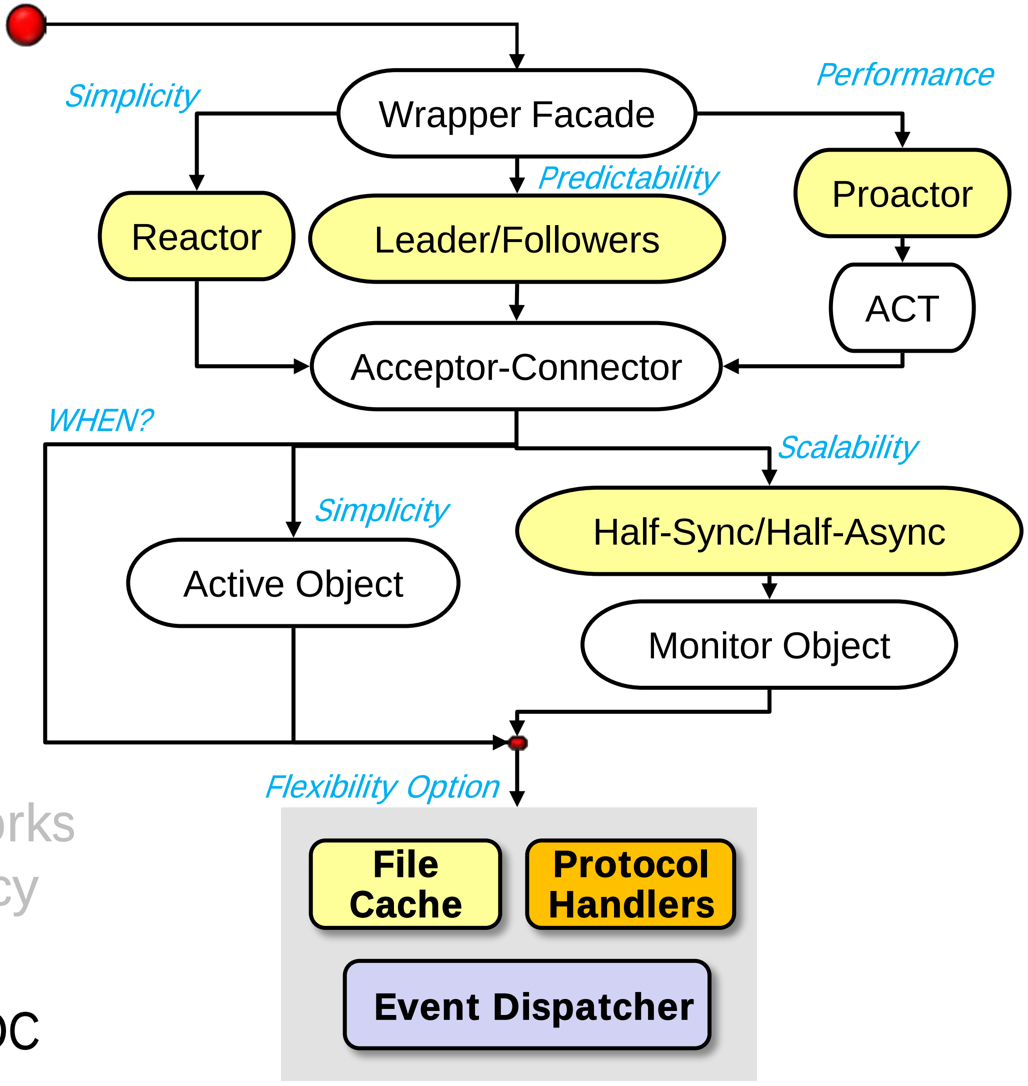
Summary

- Patterns & frameworks enhance *systematic reuse* of software
- In contrast to opportunistic reuse, which involves “cutting & pasting”
- Systematic reuse involves both design *and* code reuse
- Together, patterns & frameworks avoid rediscovering & reinventing solutions
- Patterns & frameworks are deep technical topics
- We just cover patterns & frameworks that pertain to Android concurrency & communication mechanisms
- More available in 2013 POSA MOOC



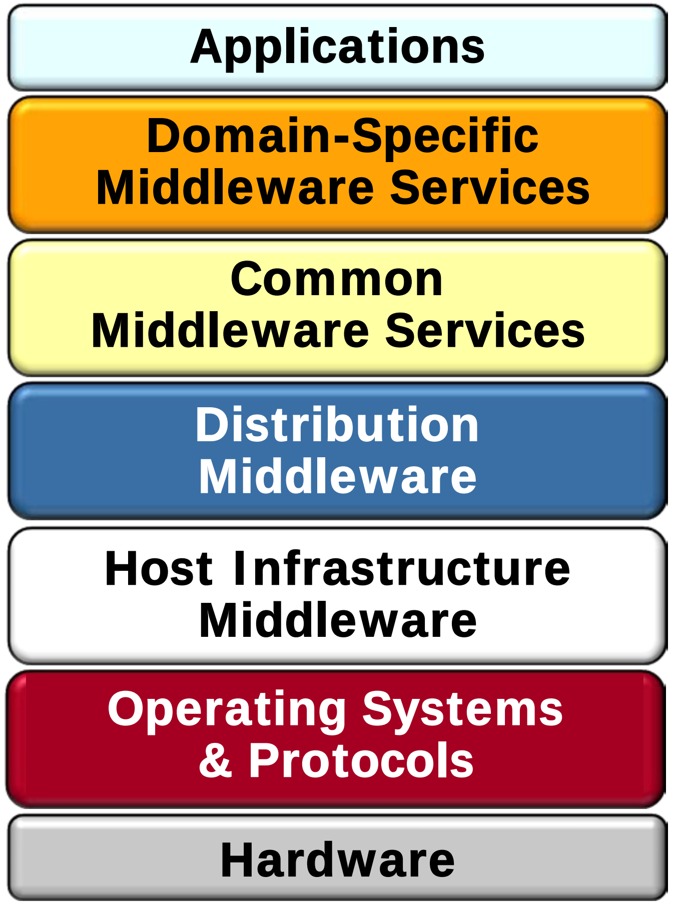
Summary

- Patterns & frameworks enhance *systematic reuse* of software
- In contrast to opportunistic reuse, which involves "cutting & pasting"
- Systematic reuse involves both design *and* code reuse
- Together, patterns & frameworks avoid rediscovering & reinventing solutions
- Patterns & frameworks are deep technical topics
- We just cover patterns & frameworks that pertain to Android concurrency & communication mechanisms
- More available in 2013 POSA MOOC



Summary

- Patterns & frameworks enhance *systematic reuse* of software
- In contrast to opportunistic reuse, which involves "cutting & pasting"
- Systematic reuse involves both design *and* code reuse
- Together, patterns & frameworks avoid rediscovering & reinventing solutions
- Patterns & frameworks are deep technical topics
- We just cover patterns & frameworks that pertain to Android concurrency & communication mechanisms
- More available in 2013 POSA MOOC


Applications
Domain-Specific Middleware Services
Common Middleware Services
Distribution Middleware
Host Infrastructure Middleware
Operating Systems & Protocols
Hardware