# Architecture Patterns and Tactics

Lotfi ben Othmane

# Definitions

- Architectural styles define types of components and connectors in specified topology that are useful for structuring an application logically or physically.
- Architectural/design patterns are conceptual solutions for recurring problems
- Tactics are design decisions that influence the control of a quality attribute response

- Architecture styles are often mixed up with architecture patterns – they often refer to the same thing

# Practice

- Which architecture style addresses the need of data sharing?
- Which architecture style addresses the need of interoperability of OSs?
- What style addresses the need of deployability? What about software with rich GUI?

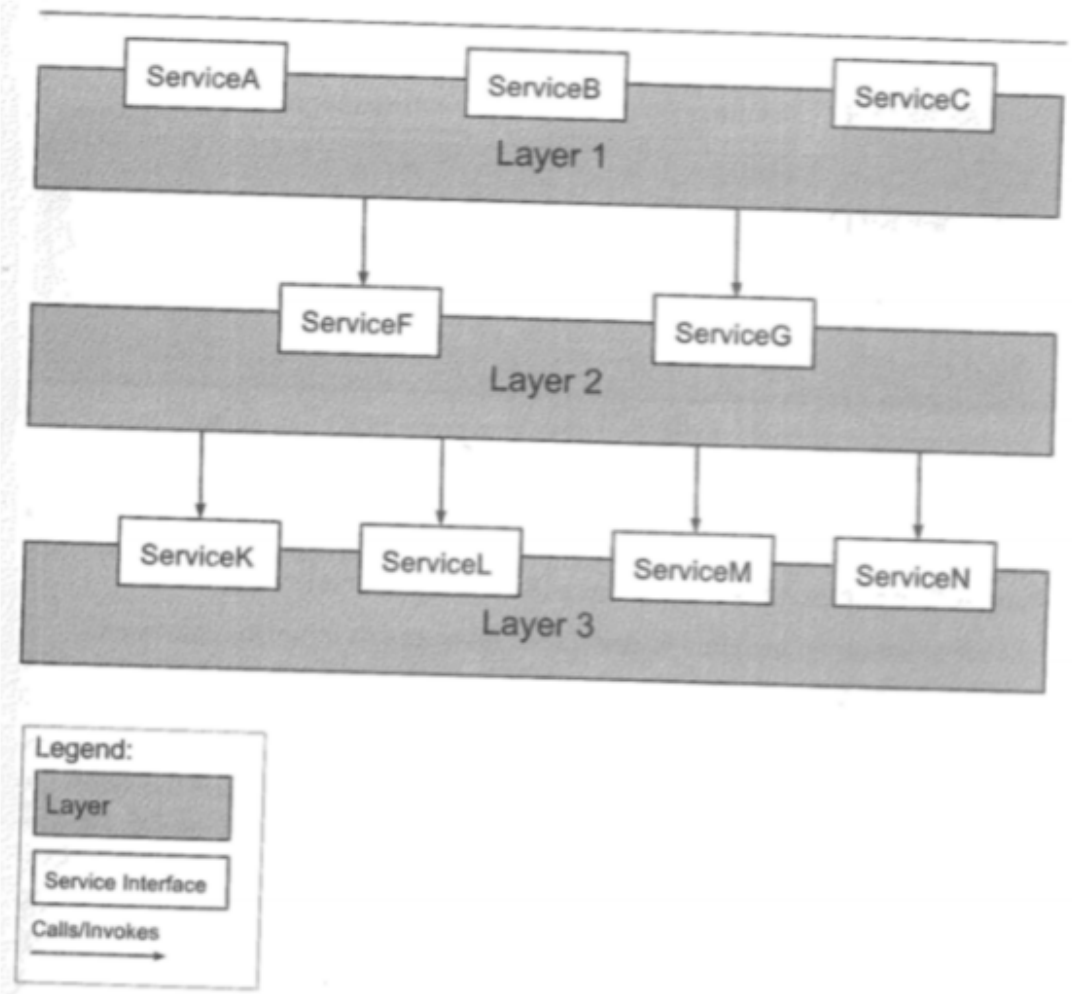
# Example of Modifiability Drivers

- Team members are specialized in backend, frontend, and data processing.
- Backend does not depend on frontend.
- There could be many frontends.
- The code should be easy to modify.

# Layered Architecture

- An architectural style that involves grouping functionalities into layers that only communicate in a singular direction – upper layers send commands to lower layers.
- Functionalities of each layer are related by a common role or set of responsibilities.
- Layers are often related to technologies, e.g. databases, business, presentations.

# Layered Architecture



# Layered Architecture

Benefits:

- Abstraction – support changes of layers without impacting the upper layers
- Isolation – isolate technology upgrades
- Manageability – knowledge of the dependencies helps manage the sections
- Possible reuse of layers
- Test layers separately

# Layered Architecture

You should consider this style to:

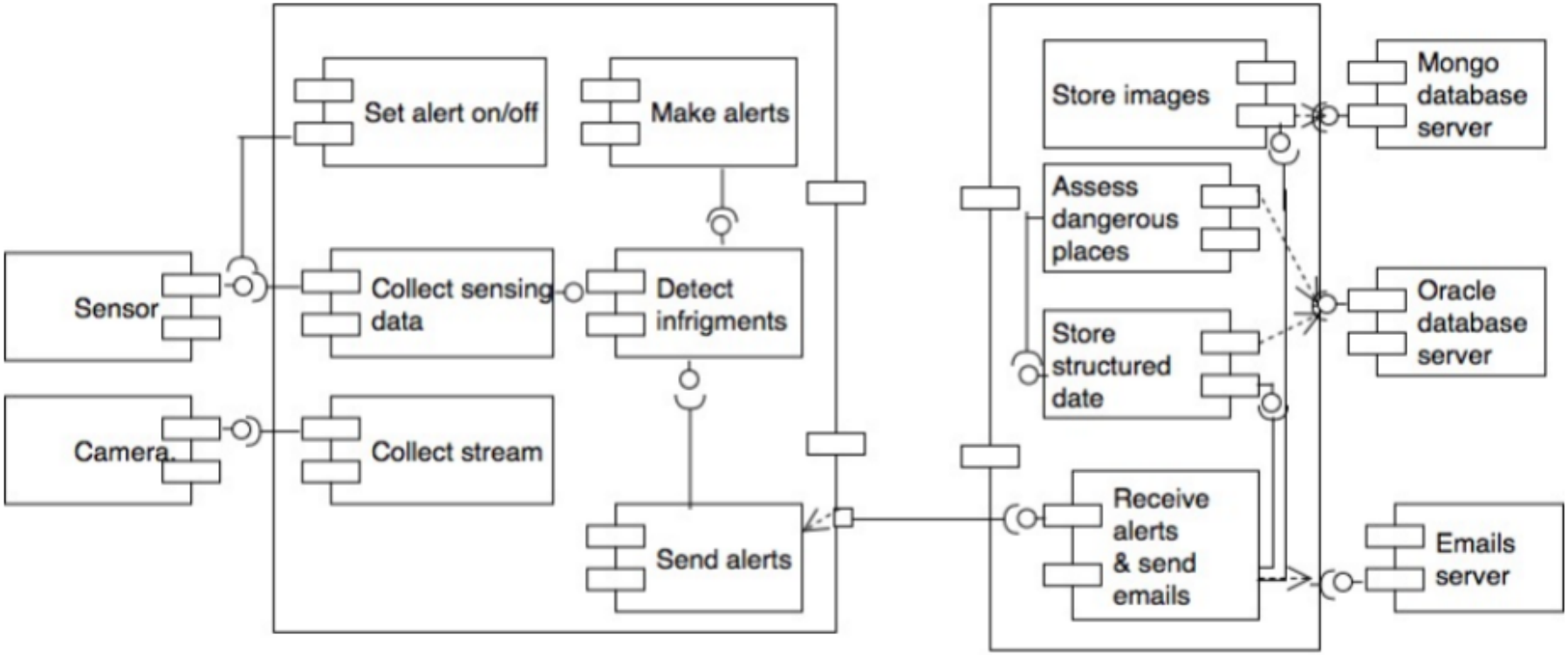
- Use existing layered implementations that you can extend.
- Adapt to team members skills with respect to technology.
- Support different client types.

# Extensibility Drivers

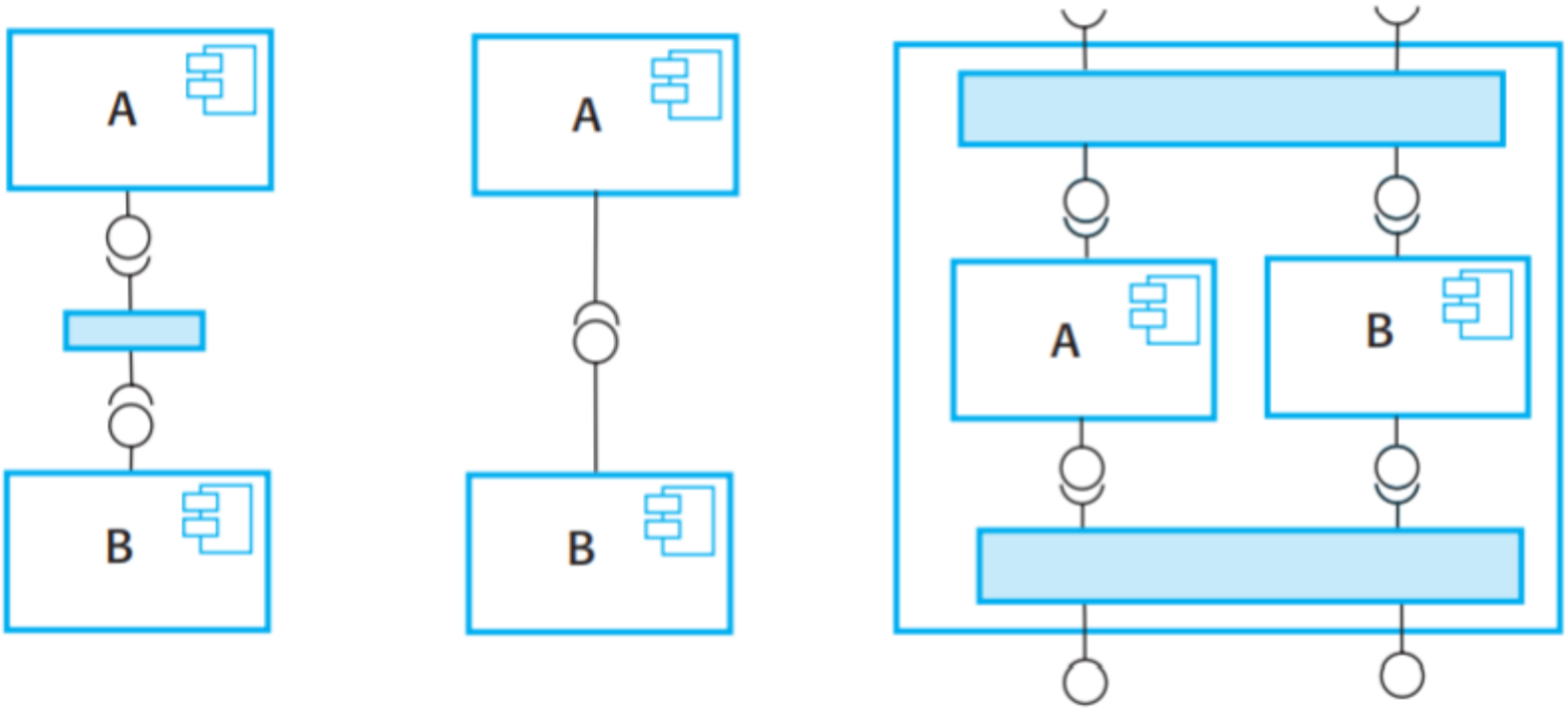
- Support adding new types of hardware that have unknown interfaces
- Support adding new components to the software and use them without the need to recompile the system

# Extensibility of a Software

How would you support different types of sensors and different types of cameras?



# Types of Composition




(1) Sequential

(2) Hierarchical

(3) Additive

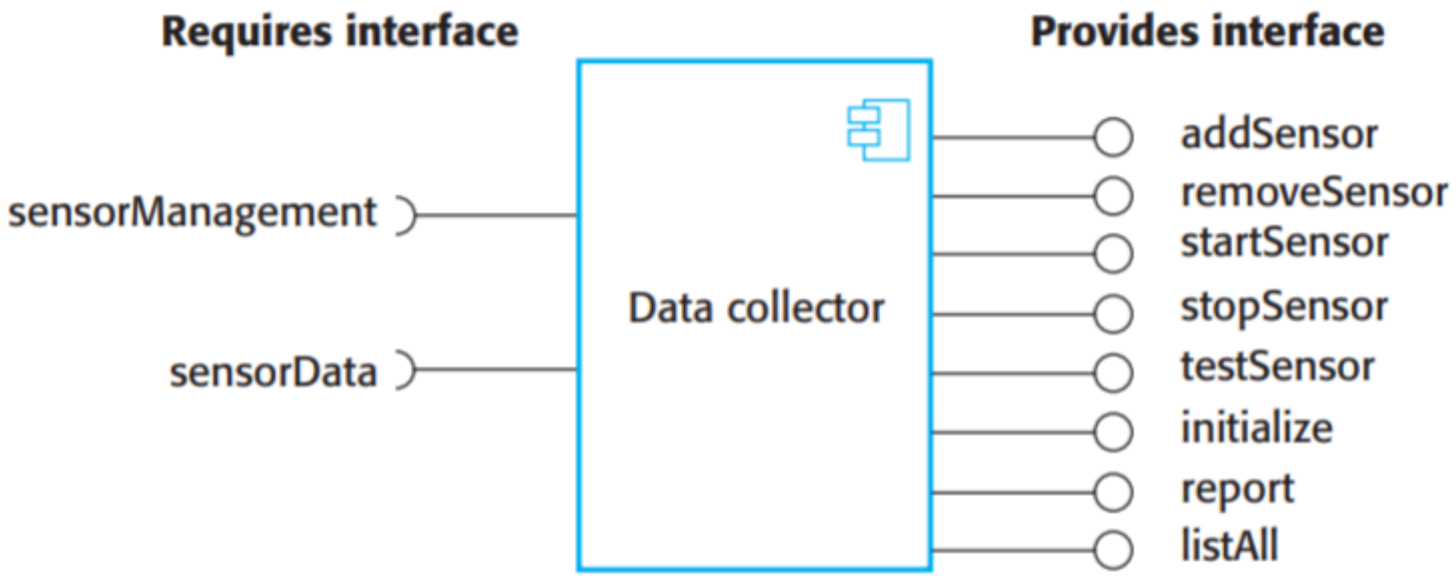
# Type of Composition

Sequential composition – the composed components are executed in sequence.

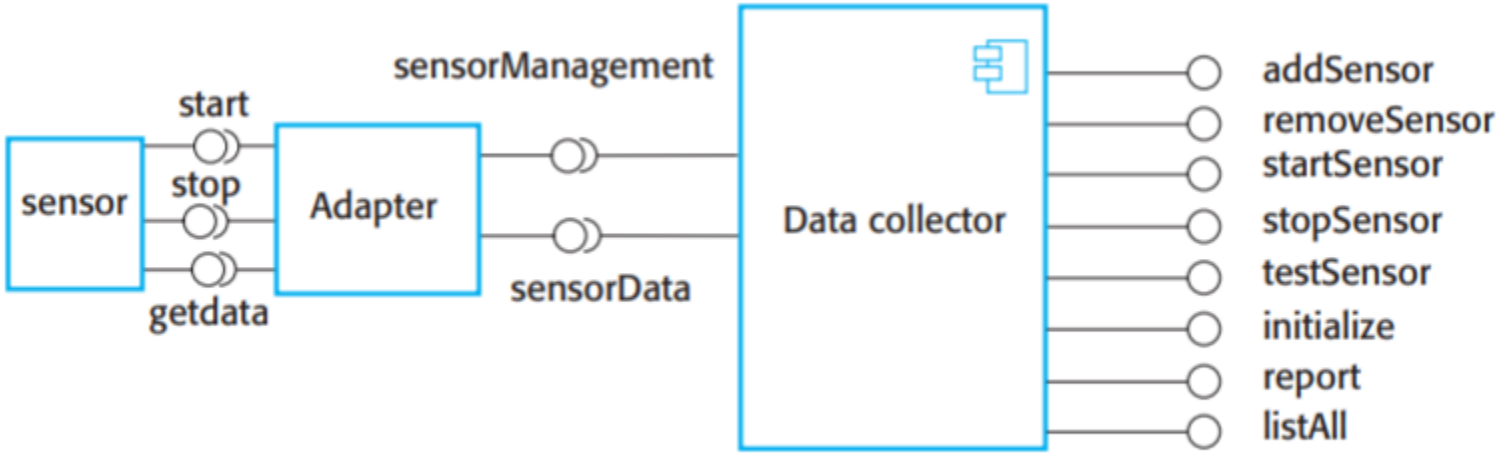
Hierarchical composition – one component calls on the services of another.

Additive composition – the interfaces of two components are put together to create a new component.

# Components Interfaces



# Adapter Linking a Data Collector and Sensor



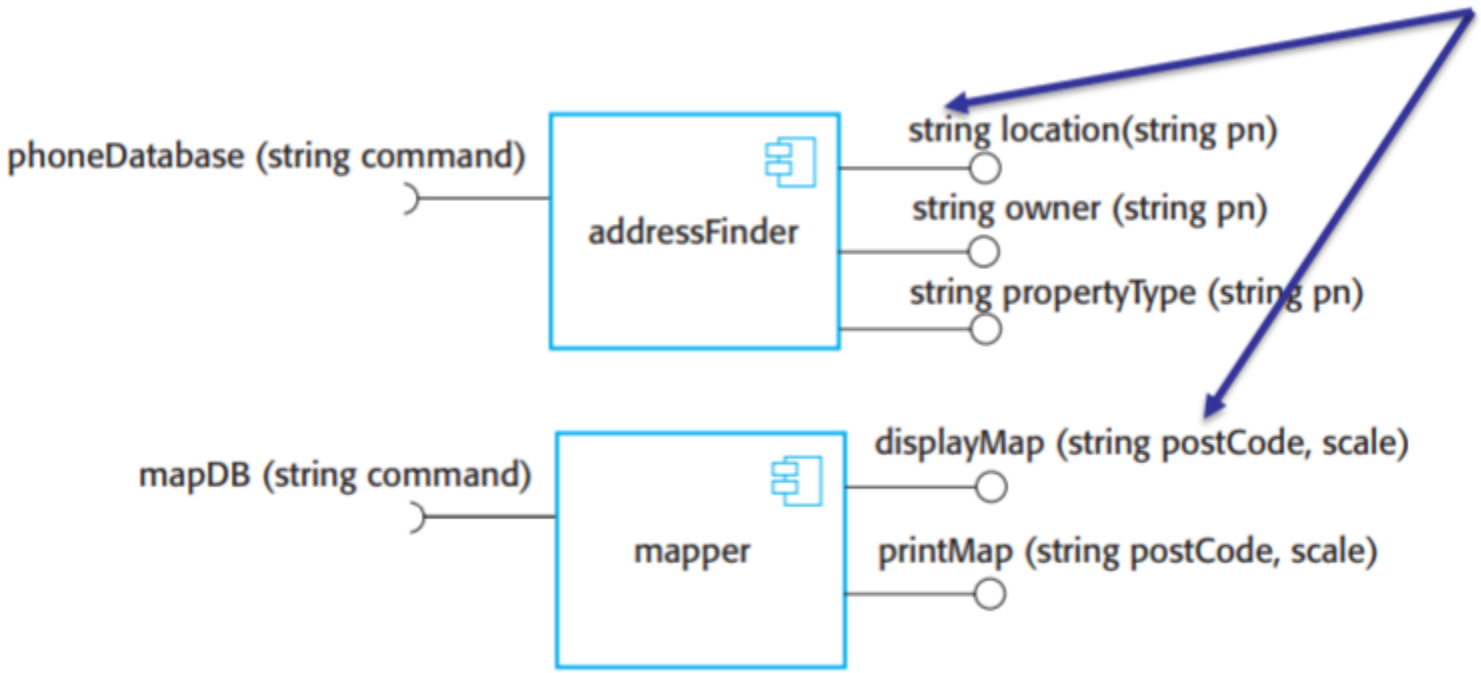
# Interface Incompatibility

Parameter Incompatibility – where operations have the same name but are of different types.

Operation Incompatibility – where the names of operations in the composed interfaces are different.

Operation Incompleteness – where the provided interface of one component is a subset of the required interface of another.

# Components with Incompatible Interfaces



# Adapter Pattern

Address the problem of component incompatibility by reconciling the interfaces of the used components.

An *addressFinder* and a mapper component may be composed through and adapter that strips the postal code from an address and passes this to the mapper component.

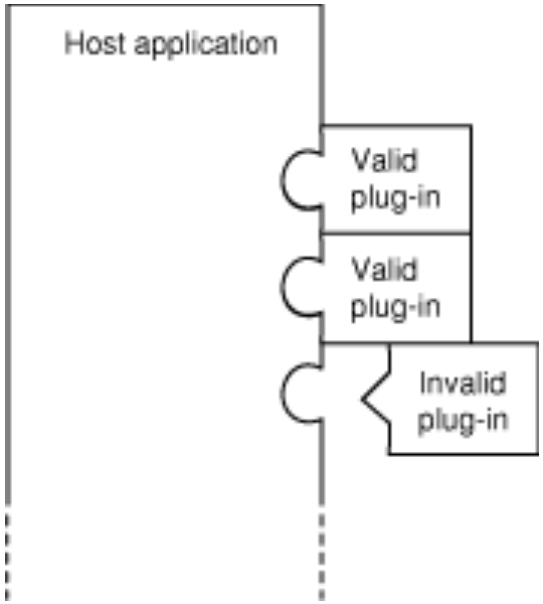
```
address = addressFinder.location (phonenumber) ;
postCode = postCodeStripper.getPostCode (address) ;
mapper.displayMap(postCode, 10000)
```

# Plugins Architecture

- Plugins architecture allows third parties to quickly add features to an application without access to source code

- The software is structured as a well-designed host framework and a set of plug-ins

- A plug-in is a bundle that adds functionality to a host framework through some well-defined architecture for extensibility.



# Plugins Architecture

Safari File Edit View History Bookmarks Window Help

developer.apple.com

Documentation Archive

Code Loading Programming Topics

Table of Contents

- Introduction
- About Loadable Bundles
- Loadable Bundles in Cocoa
- CFBundle and NSBundle
- Multi-Bundle Applications
- Plug-in Architectures
- Loading Bundles
- Creating Loadable Bundles
- Building Applications with Multiple Bundles
- Creating Plug-in Architectures
- Preventing Name Conflicts
- Loading Objective-C Libraries From Java
- Revision History

## Plug-in Architectures

This section describes how to architect an application for extensibility through plug-ins. If you want to make your application modular, cu
should read this section to learn about the different ways to build a plug-in architecture.

## About Plug-in Architectures

Plug-in architectures are an attractive solution for developers seeking to build applications that are modular, customizable, and easily ext
allow third parties to add features to an application without access to source code has, for many developers, evolved into a full-blown me

Structuring an application as a well-designed host framework and a set of plug-ins gives you many benefits as an application developer:

chromium.org

The Chromium Projects

For Developers > Design Documents >

## Plugin Architecture

### Background

Before reading this document, you should be familiar with Chromium's multi-process architecture.

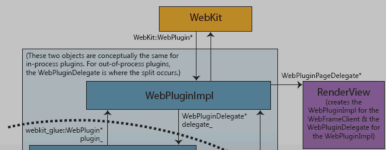
### Overview

Plugins are a major source of browser instability. Plugins also make sandboxing the process where the renderer runs impractical, as plugins are written by third-part
operating system. The solution is to run plugins in their own separate process.

### Detailed design

#### In-process plugins

Chromium has the ability to run plugins in process (this is handy for testing) as well as out of process. Both start at our non-multi-process-aware WebKit embedding
the `WebKit::WebPlugin` interface. This is implemented by `WebPluginImpl`. The `WebPluginImpl` talks "up" the chain to a `WebPluginDelegate` interface, wh
`WebPluginDelegateImpl`. This in turn talks to our NPAPI wrapper layer.




Not Secure — eclipse.org

lipse Corner Article

## Notes on the Eclipse Plug-in Architecture

**Summary**
Eclipse plug-ins embody an architectural pattern for building an application from constituent parts. This article presents an in-depth view of the participant roles and co
instance of the Eclipse workbench. The goal is to provide an understanding of plug-ins, and of how plug-in extensions are defined and processed, independently of the

**Azad Bolour, Bolour Computing**
July 3, 2003

**Table of Contents**

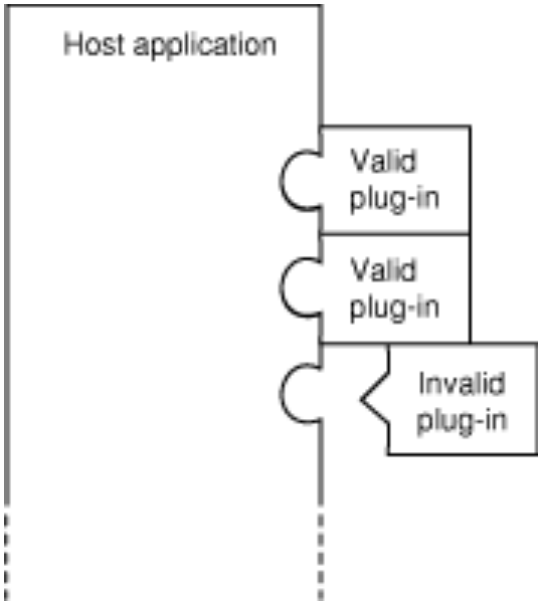
# Plugins Architecture

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
   id="com.bolour.sample.eclipse.demo"
   name="Extension Processing Demo"
   version="1.0.0">
   <runtime>
      <library name="demo.jar"/>
   </runtime>
   <requires>
      <import plugin="org.eclipse.ui"/>
   </requires>
</plugin>
```

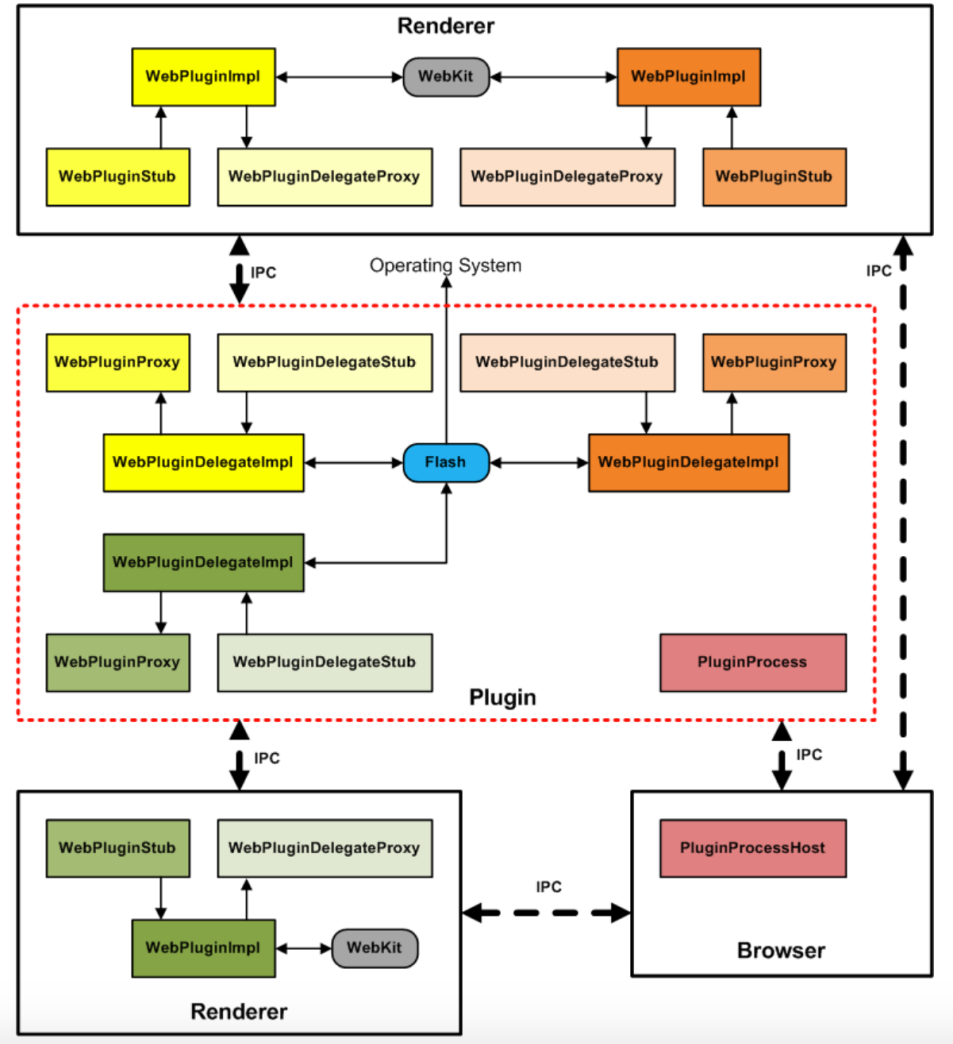
Listing 2.2. Specifying Plug-in Dependencies.

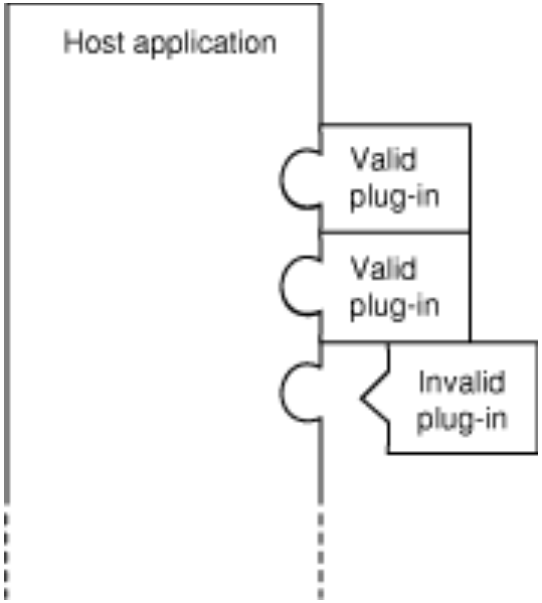
```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
1  id="org.eclipse.ui"
   name="Eclipse UI"
   version="2.1.0"
   provider-name="Eclipse.org"
   class="org.eclipse.ui.internal.UIPlugin">

2  <extension-point id="actionSets" name="Action Sets"
     schema="schema/actionSets.exsd"/>
   <!-- Other specifications omitted. -->
</plugin>
```

Listing 2.3. Declaring an Extension-Point.



# Plugins Architecture



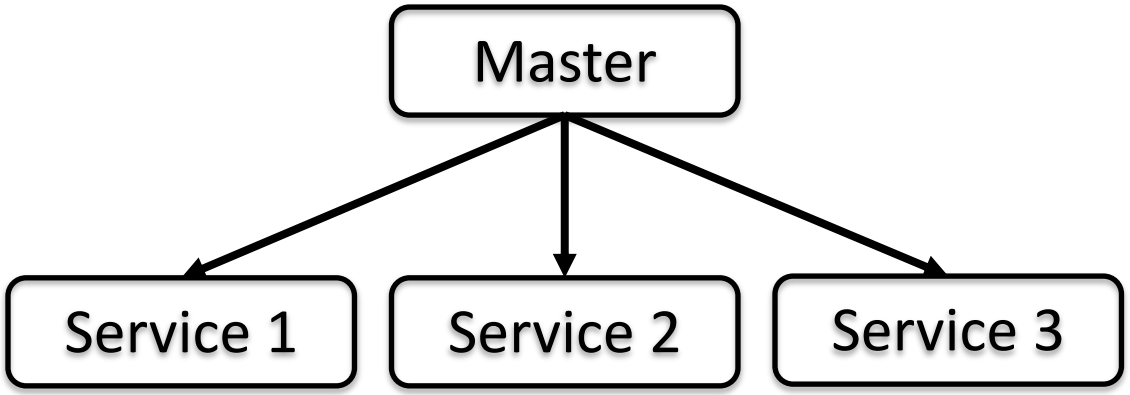
# Performance and Reliability Patterns

How to improve the reliability and/or performance of a given system?

# Master-Slave Pattern

Problem: Sporadic heavy load of a service can cause performance issue.

Solution: Distribute the load among a set of service instances

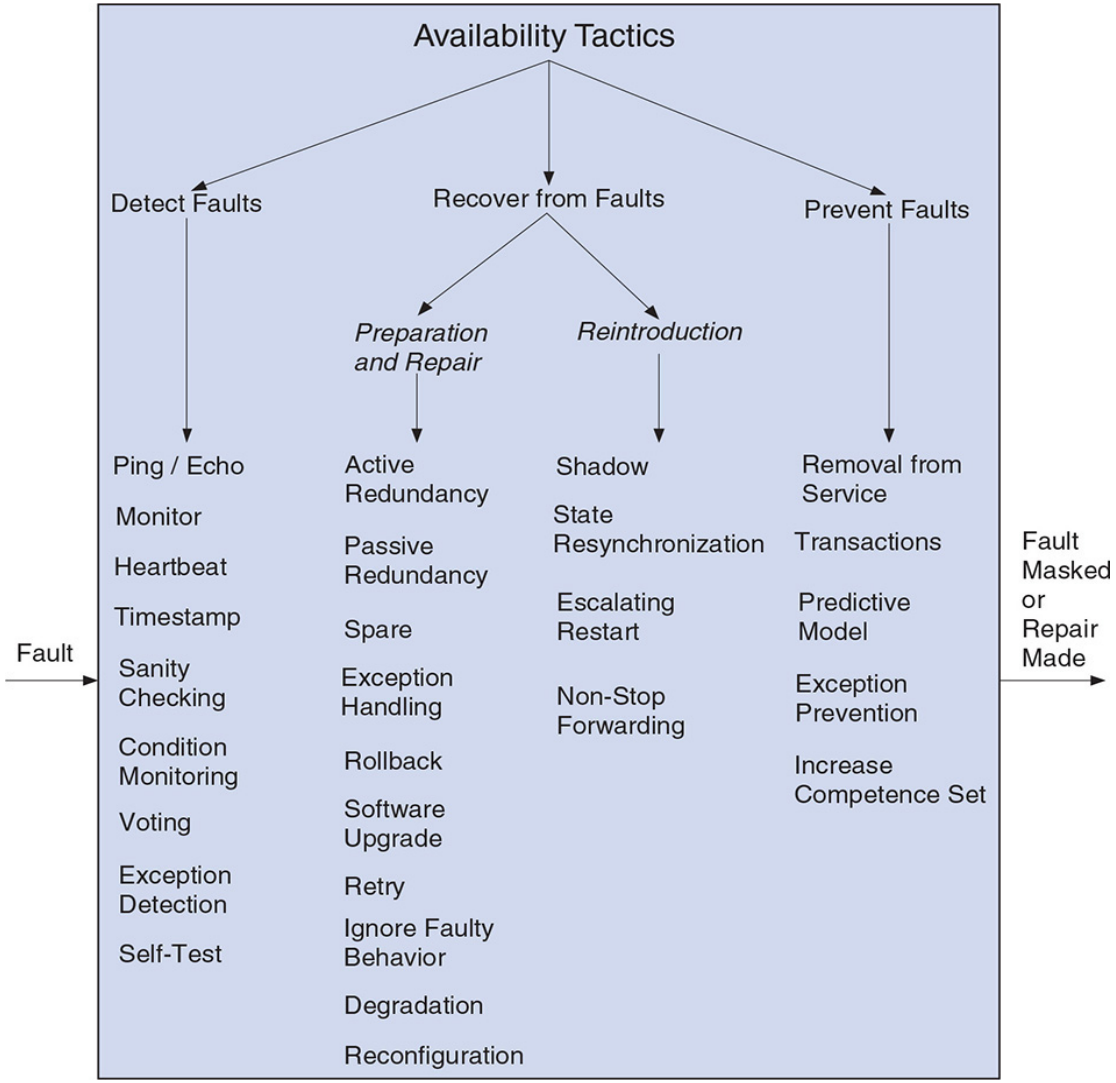


# Definitions

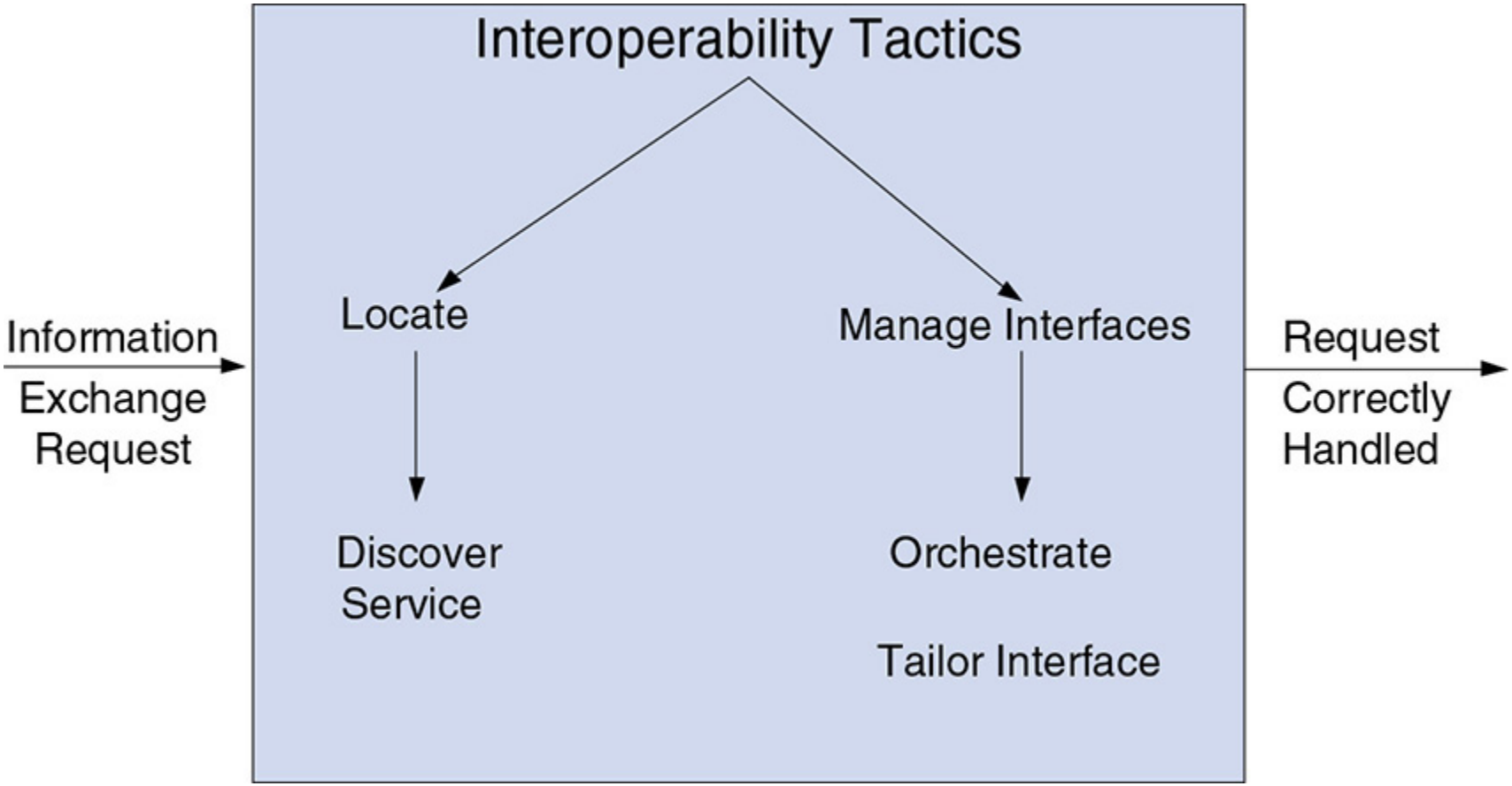
- Architectural styles define types of components and connectors in specified topology that are useful for structuring an application logically or physically.
- Architectural/design patterns are conceptual solutions for recurring problems
- Deployment patterns provide models to physically structure software
- Tactics are design decisions that influence the control of a quality attribute response

- Architecture styles are often mixed up with architecture patterns – they often refer to the same thing

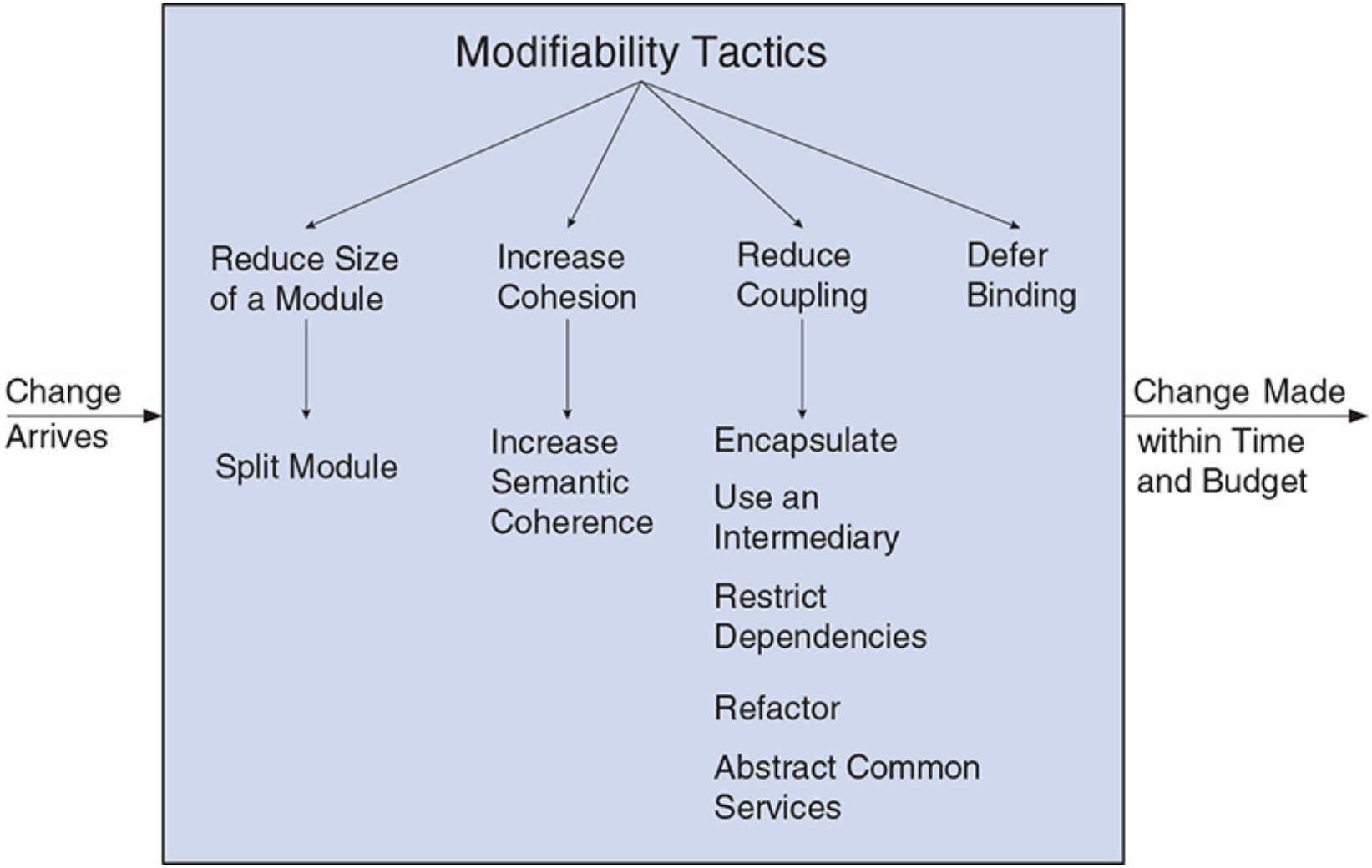
# Availability Tactics



# Interoperability Tactics



# Modifiability Tactics

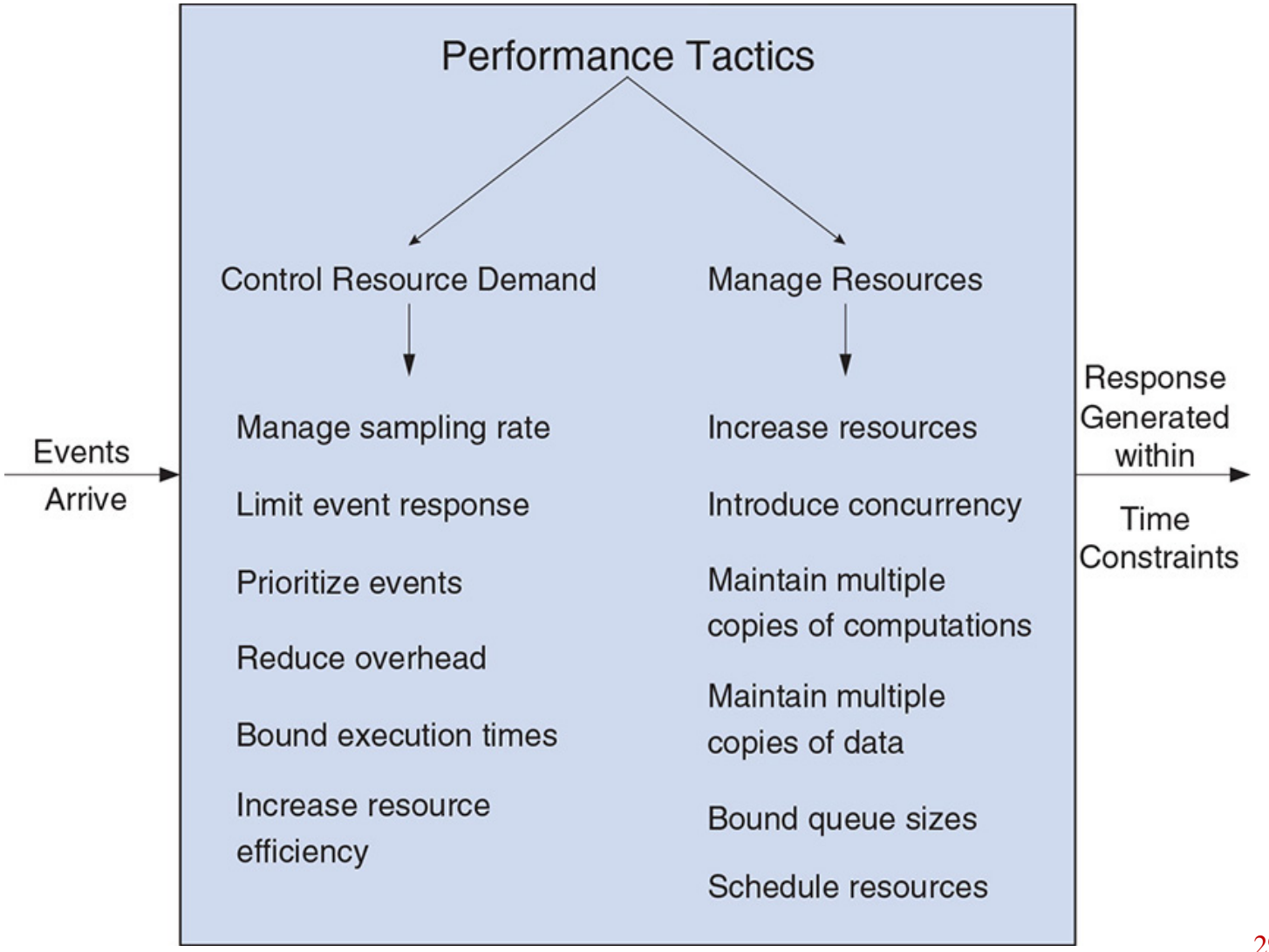


# Practice 7

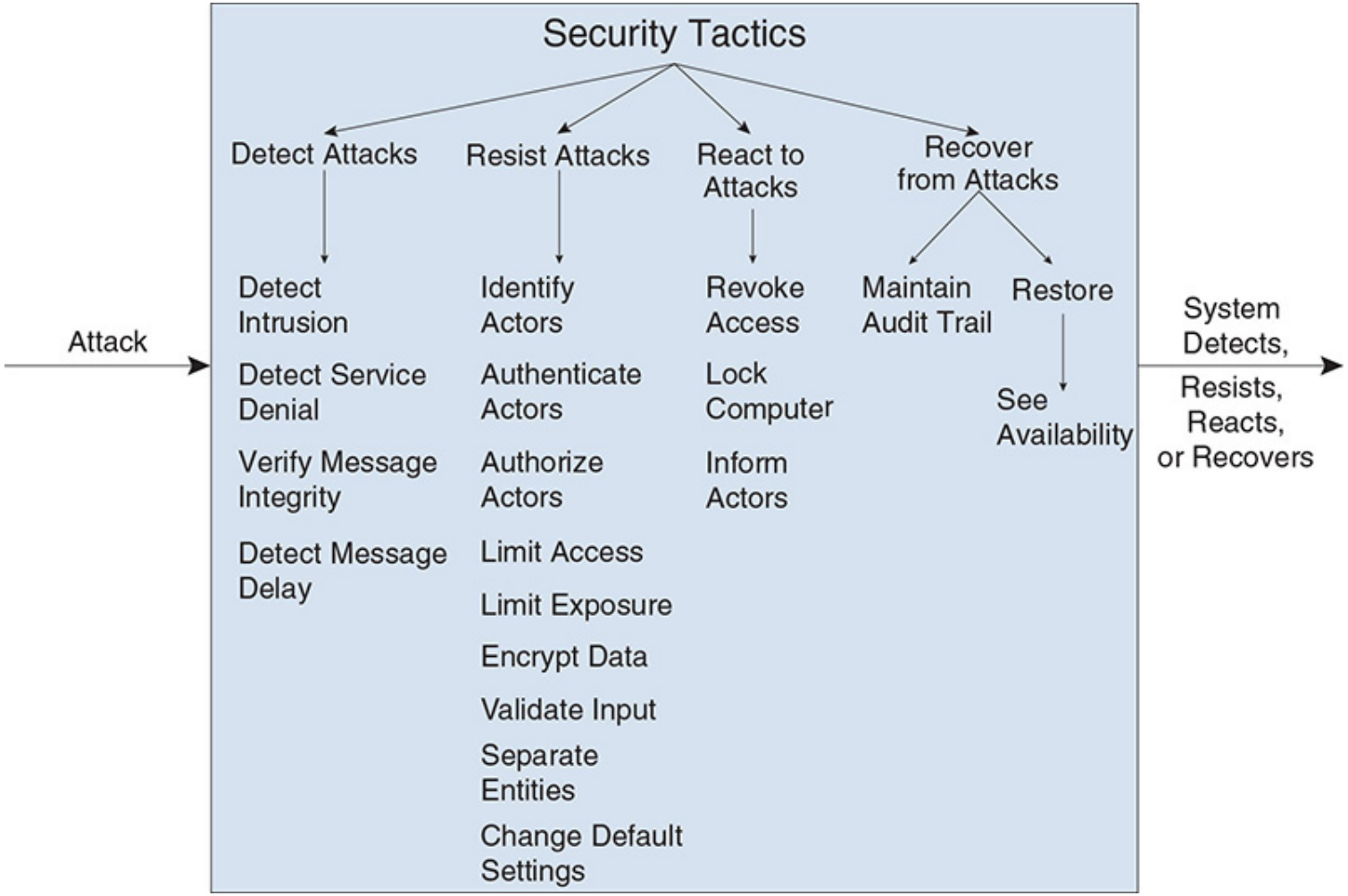
You are working on the life-style project as an initiative with few of your friends in the context of an open source project. You planned to produce releases every 3 months. What tactics would you use to achieve modifiability?

1. Reduce size of modules
2. Increase cohesion
3. Reduce coupling
4. Defer binding
5. I need further information (specify your question)

# Performance Tactics



# Security Tactics



# Practice 7

What tactics would you use for your smart home project to ensure that only you can access your devices

1. Detect intrusion
2. Verify message integrity
3. Limit access
4. Limit exposure
5. Maintain audit log
6. Encrypt data
7. Validate input
8. I know a better tactic

# Self-check

1. What problem does the layered-architecture pattern address?
2. How should you address the problem of incompatible interfaces?
3. What style should we use to increase modifiability of the system?
4. What pattern should we use to address the performance of the system?

Thank you

Questions?