Chapter 6: Interoperability

What is Interoperability?

- **Interoperability** is about the degree to which two or more systems can usefully exchange meaningful information via interfaces in a particular context.
- Any discussion of **a system's interoperability** needs to identify with whom, and under what circumstance.

What is Interoperability?

- **Syntactic interoperability** is the ability to exchange data. (语法互操作性)
- **Semantic interoperability** is the ability to interpret the data being exchanged. (语义互操作性)

What is Interoperability?

- Two perspectives for achieving interoperability
  - With the knowledge about the interfaces of external systems, design that knowledge into the system
  - Without the knowledge about other systems, design the system to interoperate in a **more general** fashion

Motivation

- The system provides a service to be used by a collection of unknown systems, eg., GoogleMaps
- The system is constructed from multiple existing systems, for example
  - Producing a representation of what was sensed
  - Interpreting the data
  - Processing the raw data
  - Sensing the environment

Two Important Aspects of Interoperability

- **Discovery**. The consumer of a service must discover the location, identity, and interface of service
- **Handling the response**. Three possibilities:
  - The service reports back to the requester
  - The service sends its response on to another system
  - The service broadcasts its response to any interested parties

Interoperability General Scenario

Portion of Scenario	Possible Values
Source	A system
Stimulus	A request to exchange information among system(s).
Artifact	The systems that wish to interoperate
Environment	System(s) wishing to interoperate are discovered at run time or known prior to run time.
Response	One or more of the following: • the request is (appropriately) rejected and appropriate entities (people or systems) are notified • the request is (appropriately) accepted and information is exchanged successfully • the request is logged by one or more of the involved systems
Response Measure	One or more of the following: • percentage of information exchanges correctly processed • percentage of information exchanges rejected

Sample Concrete Interoperability Scenario

- Our vehicle information system sends our current location to the traffic monitoring system.
- The traffic monitoring system combines our location with other information, overlays this information on a Google Map, and ***broadcasts*** it.
- Our location information is correctly included with a probability of 99.9%.

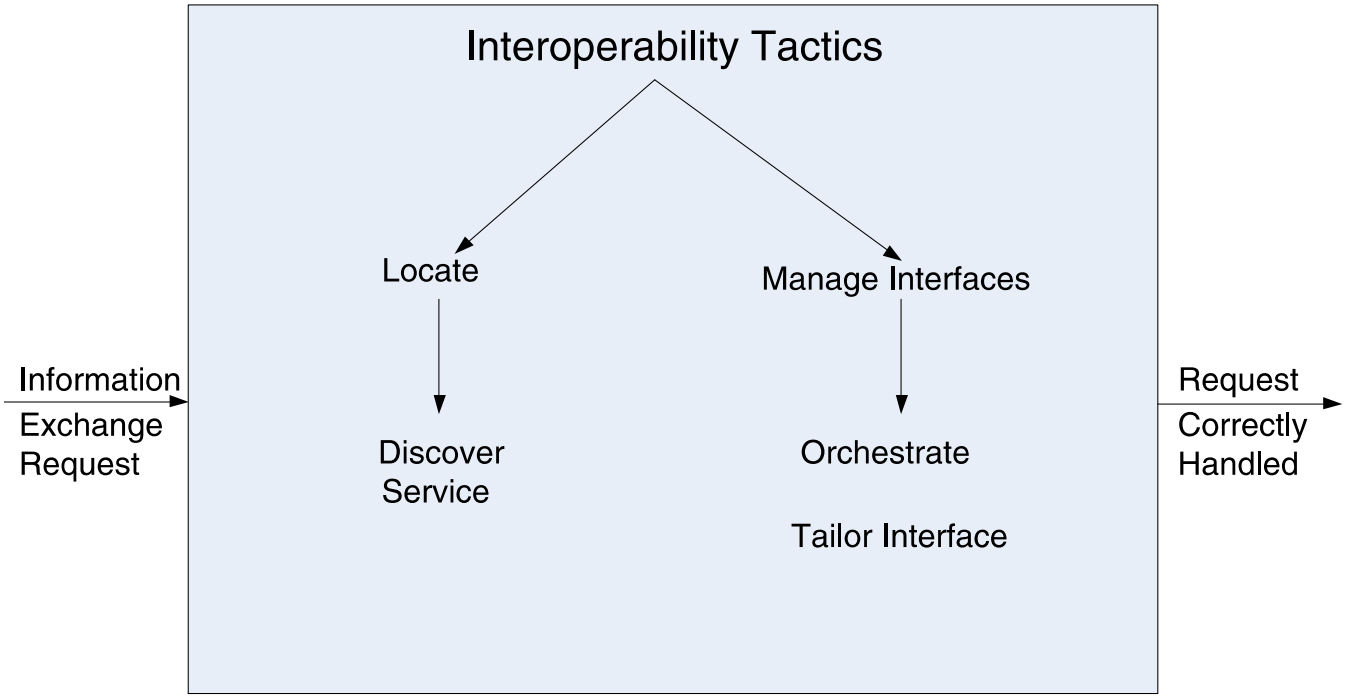
SOAP v.s. REST

- Two technology options to allow the web-based application to interoperate
- SOAP is used in SOA systems along with a set of protocols
  - Service description& discovery, e.g., WSDL, UDDI
  - Service composition, e.g., BPEL
- SOAP is more complex and used for exchange messages with structured data, while REST is simple and used for small messages

Goal of Interoperability Tactics

- For two or more systems to usefully exchange information they must
  - Know about each other. That is the purpose behind the locate tactics.
  - Exchange information in a semantically meaningful fashion. That is the purpose behind the manage interfaces tactics. Two aspects of the exchange are
    - Provide services in the correct sequence
    - Modify information produced by one actor to a form acceptable to the second actor.

Interoperability Tactics



Locate

- **Service Discovery** : Locate a service through searching
- There are many **service discovery** mechanisms:
  - UDDI for Webservices
  - Jini for Jave objects
  - Simple Service Discovery Protocol (SSDP) as used in Universal plug-and-play (UPnP)
  - DNS Service Discovery (DNS-SD)
  - Bluetooth Service Discovery Protocol (SDP)

Service Discovery – Necessary conditions

- The searcher wants to find the searched entity and the searched entity **wants to be found**
- The searched entity must have **identifiers**
- The searcher must acquire **sufficient identifiers** to identify the searched entity

Searching Method – Searcher's initiative

- Flood/Broadcast request
  - Ask every entity and wait for answer
- Examples
  - Paging in the location area to find the mobile terminal
  - DHCP discover: the client broadcasts on the local subnet to find available servers to ask for IP address
- Efficient and less resource consuming for the searcher
- Low resource consuming for the searched
- But disturbing and resource consuming for the environment

Searching Method – Searcher's initiative

- Successive request:
  - Ask one entity at the time and perform matching
  - If no match, continue with next until finding a match
- Less efficient and high resource consuming for the searcher
- But less disturbing and less resource consuming for the environment

Searching Method – Searched's initiative

- Continuous/periodical advertisement:
  - Continuously or periodically publish advertisement such that every searcher can notice and respond
- Efficient but high resource consuming for the searched
- Low resource demanding for the searcher
- Disturbing and resource consuming for the environment

Searching Method – Searched's initiative

- Advertisement upon arrival of new entity
  - E.g., present himself when a new person enters the lobby
- Require detection mechanism upon new entity arrival
- Less resource consuming for the searched
- Low resource demanding for the searcher
- Less disturbing and resource consuming for the environment

Searching Method – Registration

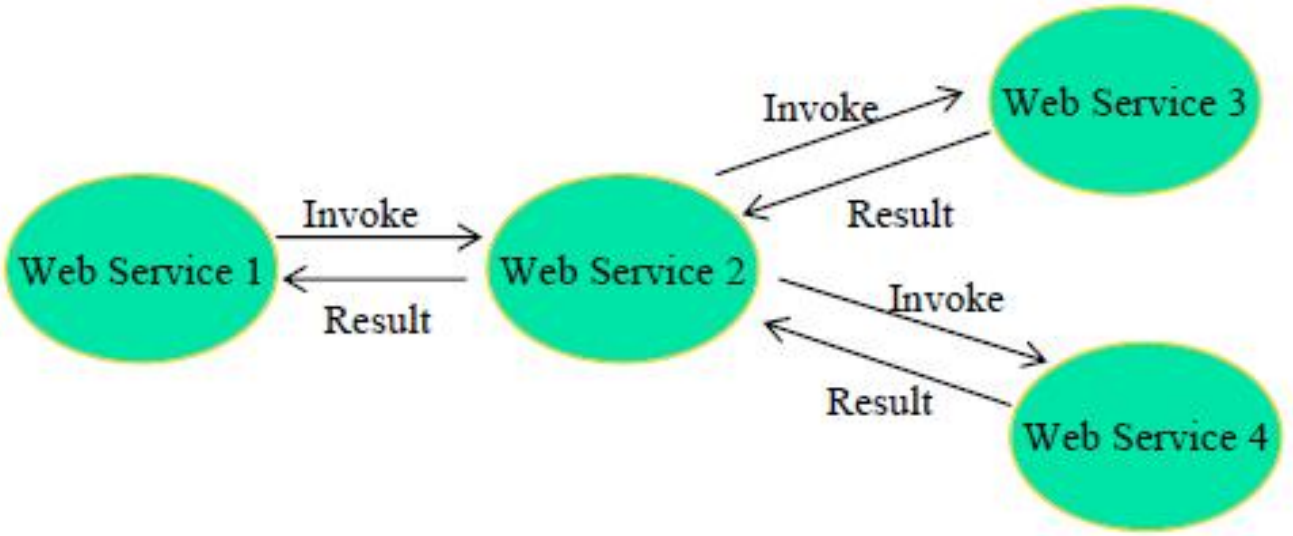
- Introduction of the “middlemen”, registry
  - The searched entity registers to a registry
  - The searcher can address to the registry to get information and find the searched entity
- Example
  - Service providers register their web services at UDDI registry which can be searched and found by Service Requestors

Searching Method – Registration

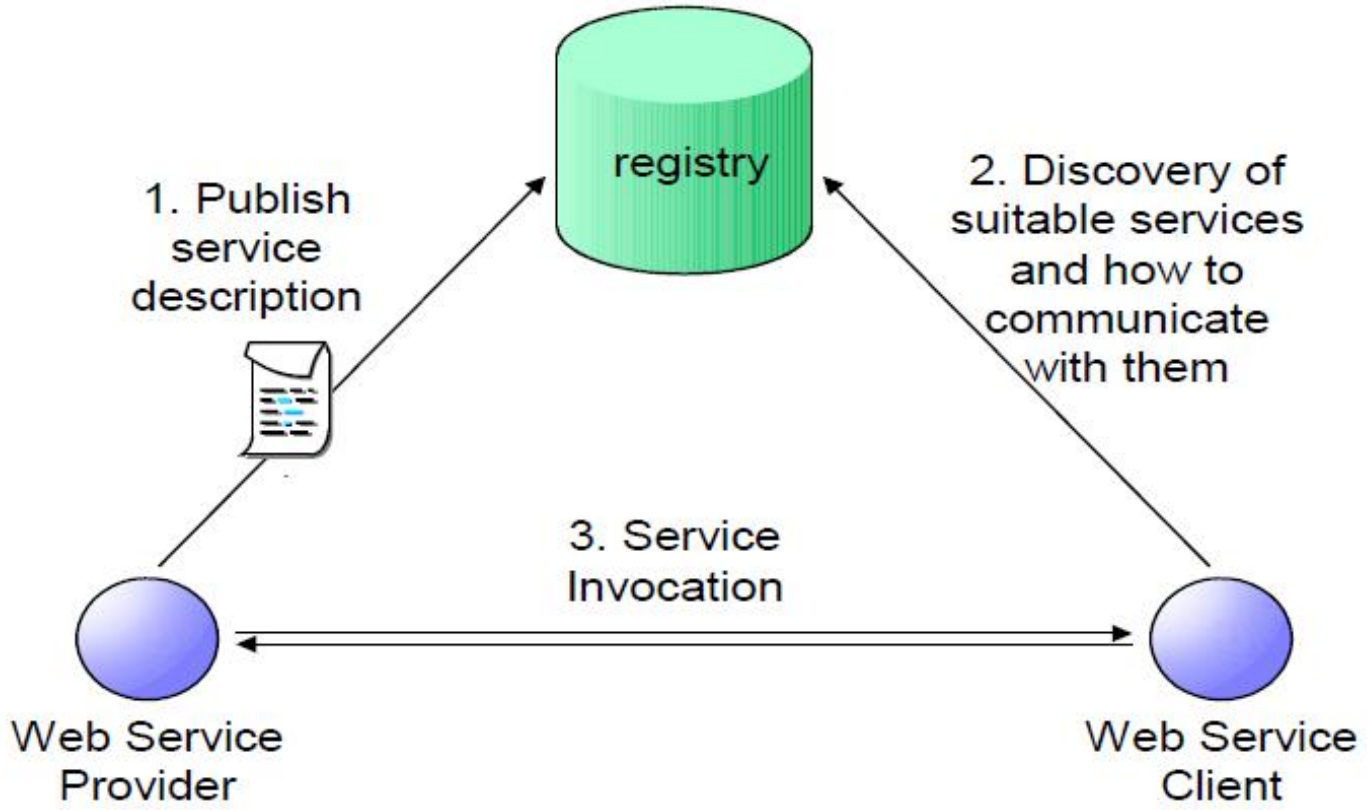
Less resource consumption on both searchers, searched, and less disturbing to environment, but the registry must be available, reliable, and correct

Web Service

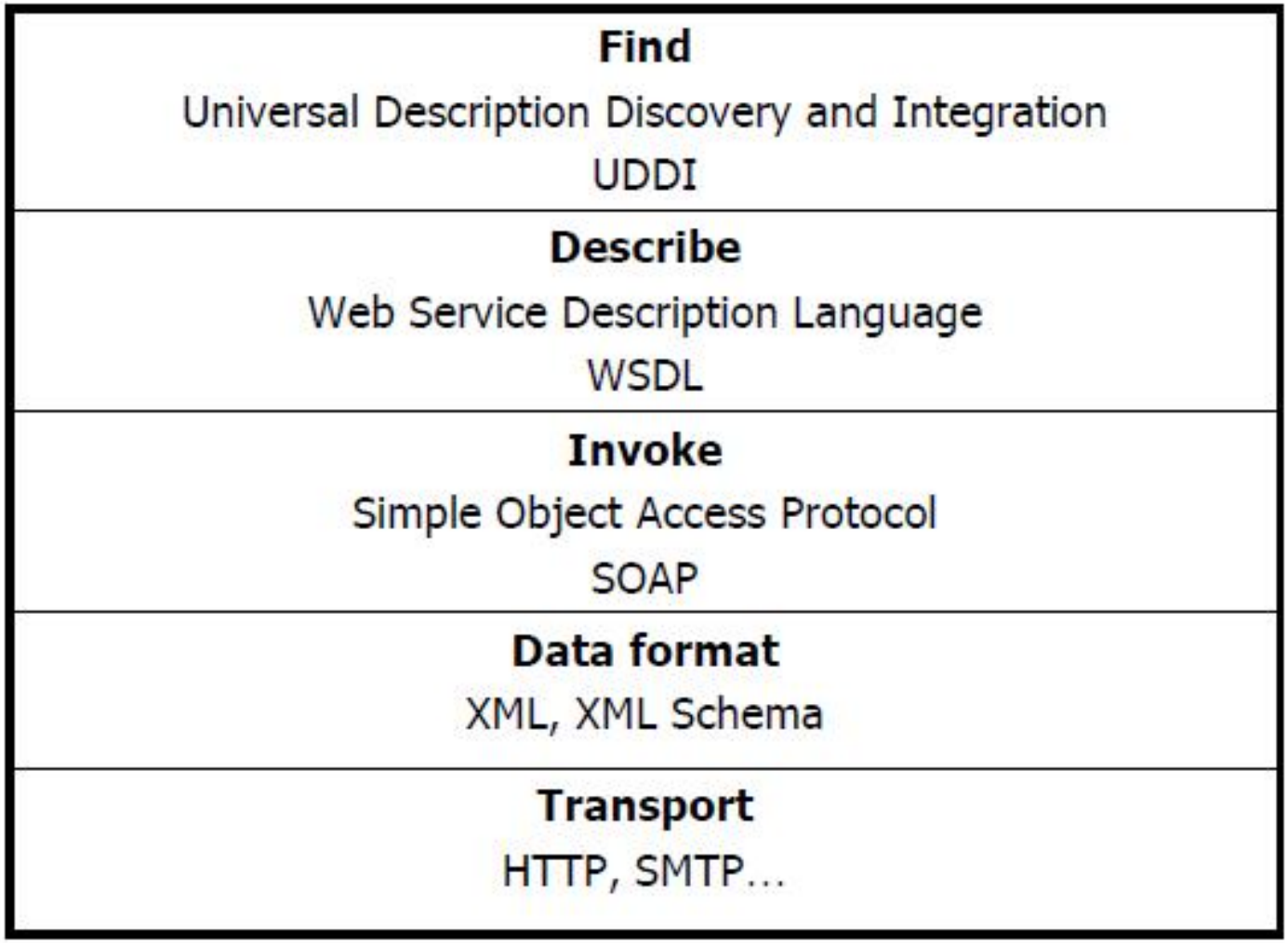
- Describes any computational functionality that can be found and invoked over any network (e.g. the Internet)
- Represents a self-describing, self-contained application
- Designed to be used by other programs or applications rather than humans



Web Service Architecture

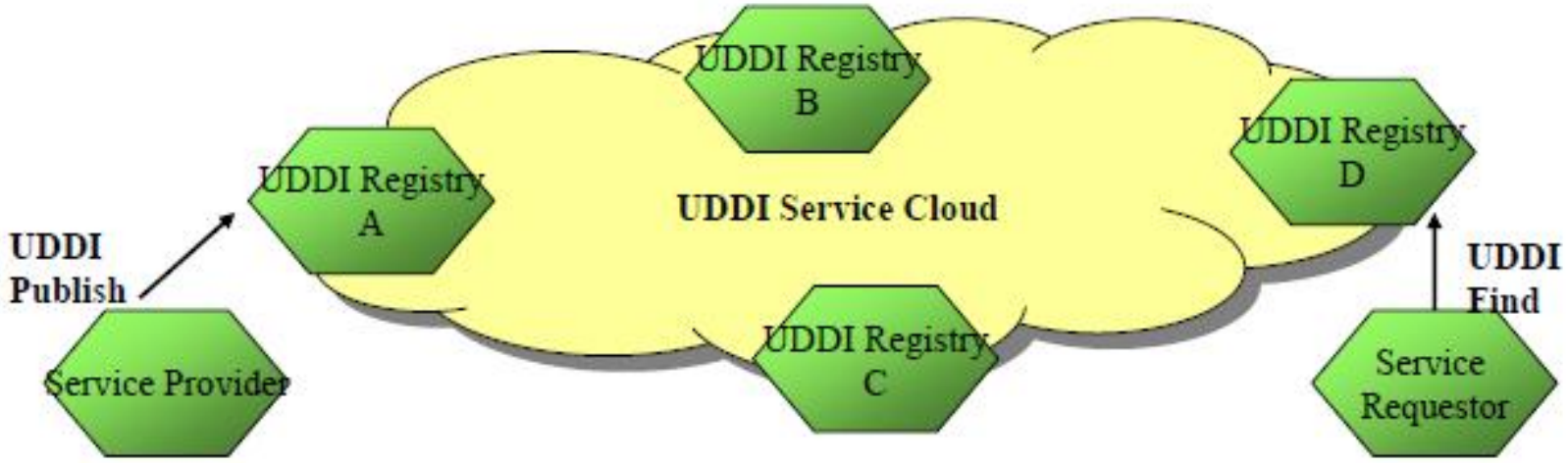


Web Service Architecture



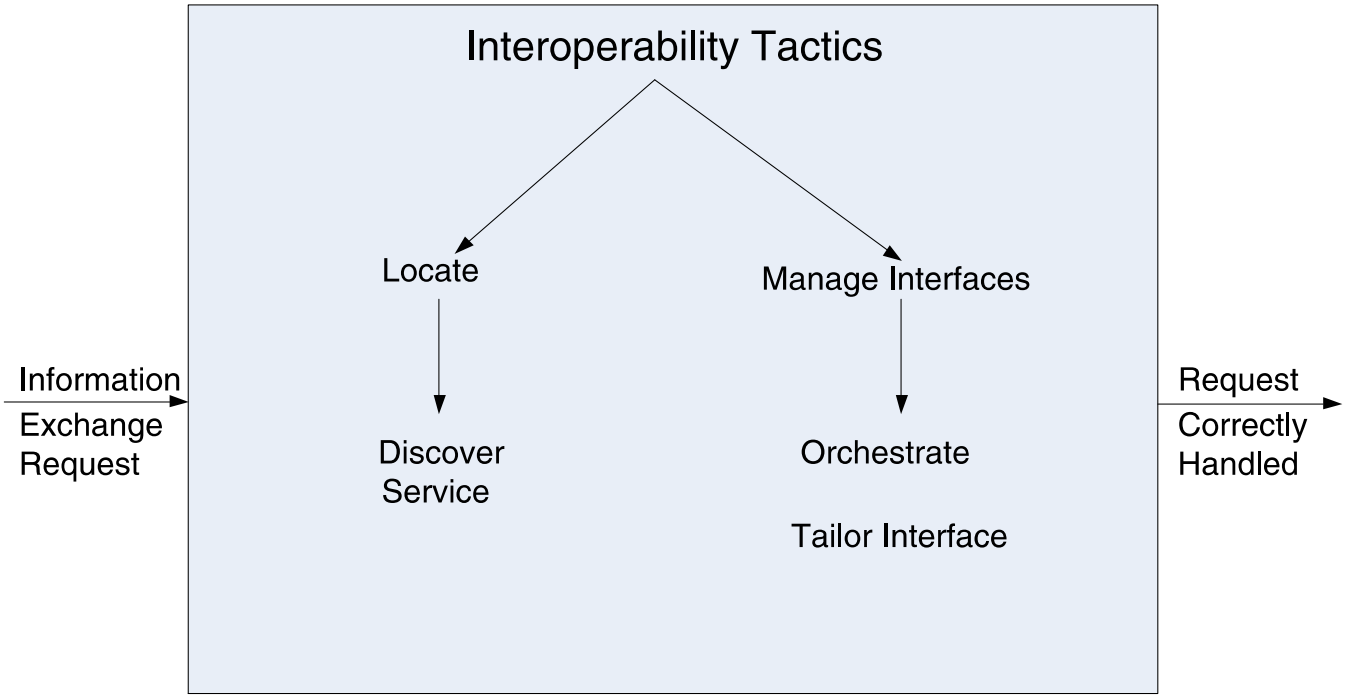
UDDI Registries

- A network of UDDI registries resembling the Domain Name System (DNS)
- All UDDI registers exchange information
- Accessing one registry provides all information contained in all registries



- Universal Description, Discovery, and Integration (UDDI) was touted as a discovery service, but commercial support for UDDI is being withdrawn.
- Why do you suppose this is? Does it have anything to do with the quality attributes delivered or not delivered by UDDI solutions?
  - 单中心
  - 单点失败
  - 瓶颈

Interoperability Tactics



Manage Interfaces

- **Orchestrate**: uses a control mechanism to coordinate, manage and sequence the invocation of services.
- Orchestration is used when systems must interact in a complex fashion to accomplish a complex task.
- **Tailor Interface**: add or remove capabilities to an interface such as translation, buffering, or data-smoothing.

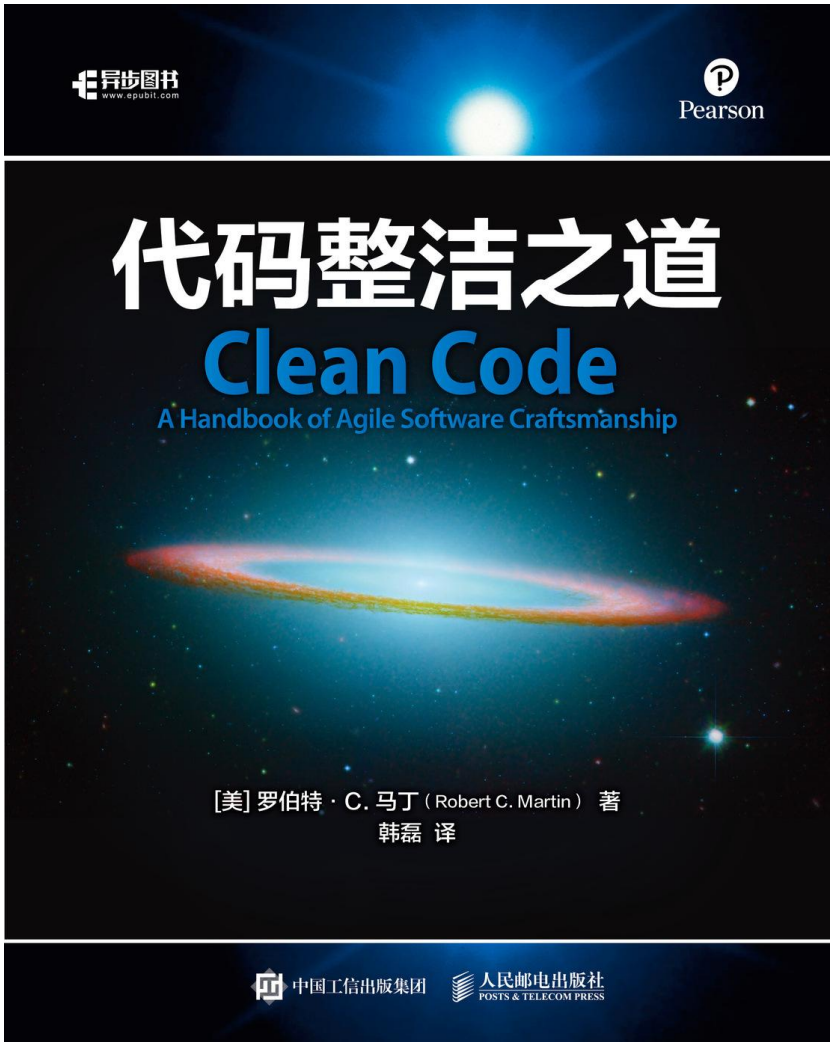
Summary

- Interoperability refers to the ability of systems to usefully exchange information.
- Achieving interoperability involves the relevant systems locating each other and then managing the interfaces so that they can exchange information.

软件体系结构 10.17

微信扫码签到

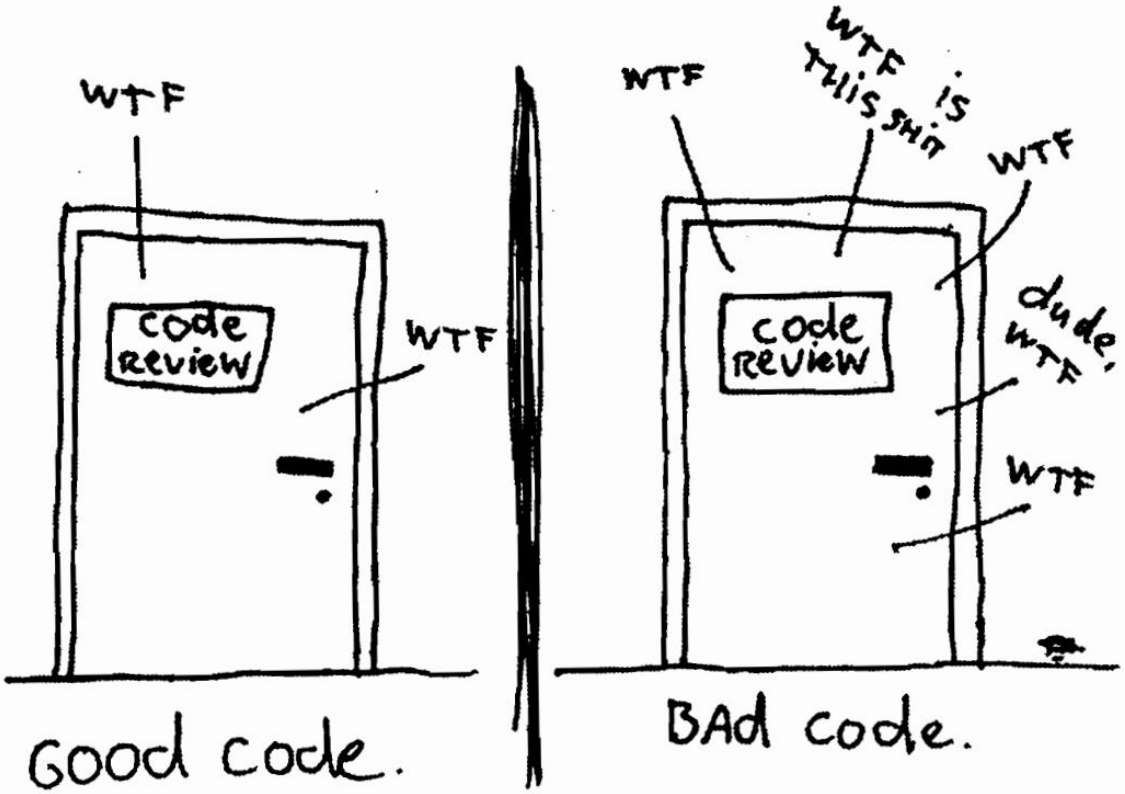
Chapter 7: Modifiability

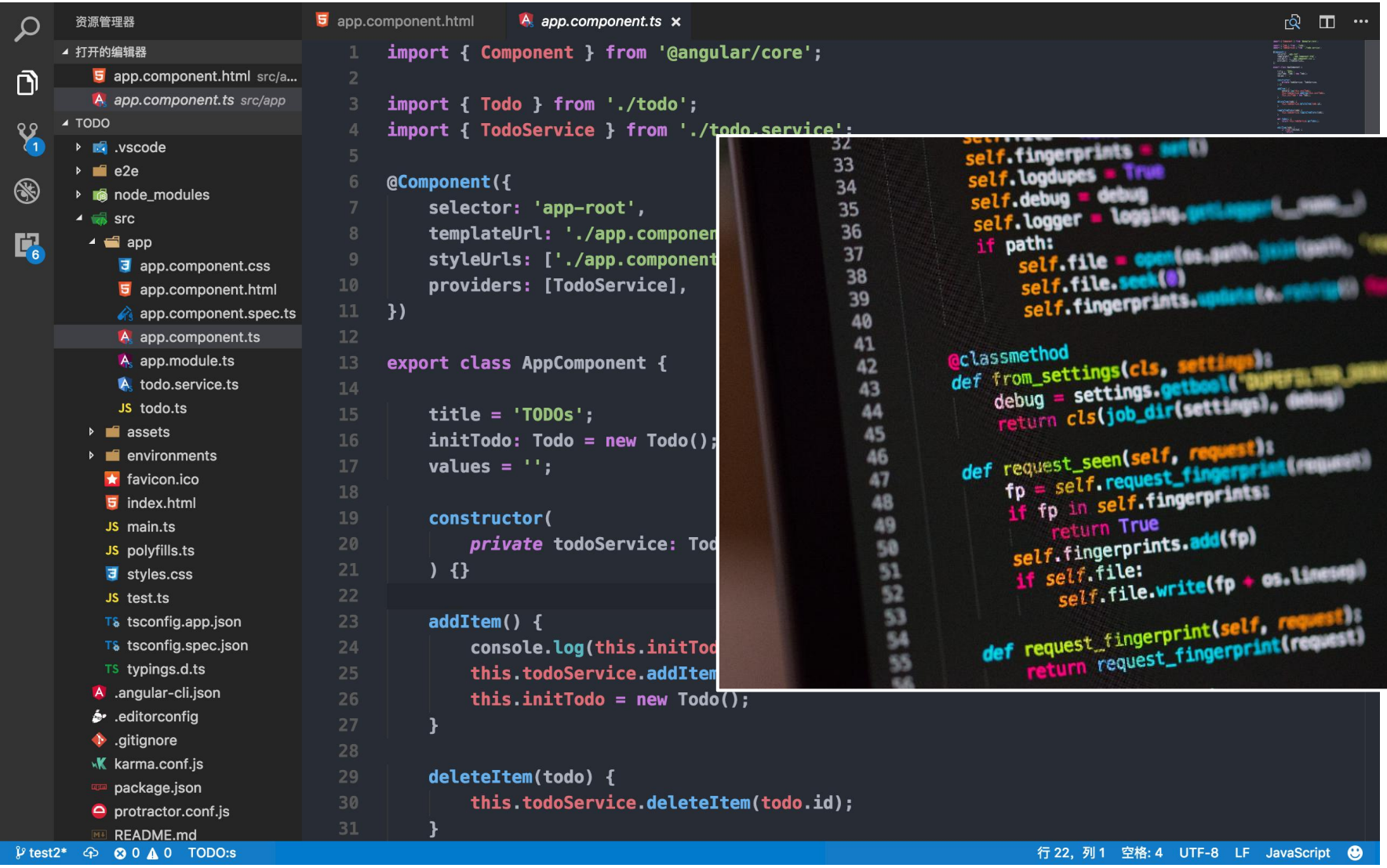

异步图书
www.epubit.com
Pearson
代码整洁之道
Clean Code
A Handbook of Agile Software Craftsmanship
[美] 罗伯特·C. 马丁（Robert C. Martin） 著
韩磊 译
中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

写代码要像写诗一样

The ONLY VALID MEASUREMENT OF Code QUALITY: WTFs/minute

衡量代码质量的唯一有效标准：WTF/min




资源管理器
打开的编辑器
app.component.html src/a...
app.component.ts src/app
TODO
.vscode
e2e
node_modules
src
app
app.component.css
app.component.html
app.component.spec.ts
app.component.ts
app.module.ts
todo.service.ts
todo.ts
assets
environments
favicon.ico
index.html
main.ts
polyfills.ts
styles.css
test.ts
tsconfig.app.json
tsconfig.spec.json
typings.d.ts
.angular-cli.json
.editorconfig
.gitignore
karma.conf.js
package.json
protractor.conf.js
README.md
app.component.html
app.component.ts
import { Component } from '@angular/core';
import { Todo } from './todo';
import { TodoService } from './todo.service';
@Component({
selector: 'app-root',
templateUrl: './app.componen
styleUrls: ['./app.componen
providers: [TodoService],
})
export class AppComponent {
title = 'TODOs';
initTodo: Todo = new Todo();
values = '';
constructor(
private todoService: Toc
) {}
addItem() {
console.log(this.initTo
this.todoService.addIte
this.initTodo = new Todo();
}
deleteItem(todo) {
this.todoService.deleteItem(todo.id);
}
test2*
0 0
TODO:s
行 22，列 1
空格: 4
UTF-8
LF
JavaScript

```
        int n = s.length();
        while(i<n){
            if(s.charAt(i)>='0'&&s.charAt(i)<='9'){
                numSign = 1;
                sign = -1;
            }else if(s.charAt(i)=='+'||s.charAt(i)=='-'){
                if(sign>0){
                    sign = -sign;
                }else{
                    return false;
                }
                if(i>0&&s.charAt(i-1)=='.'){
                    return false;
                }
            }else if(s.charAt(i)=='.'){
                //numSign = -1;

                if(pointSign>0){
                    pointSign = -pointSign;
                }else{
                    return false;
                }
                if(i>0&&(s.charAt(i-1)=='e'||s.charAt(i-1)=='E')){
                    return false;
                }
            }else if(s.charAt(i)=='e'||s.charAt(i)=='E'){
                if(eSign<0||numSign<0){
                    return false;
                }
                eSign = -1;
                sign = 1;
                numSign = -1;
                pointSign = -1;
            }else{
                return false;
            }
            i++;
        }
        return numSign>0;
    }
}
```

nunit.js (grunt-nunit-runner) - Sublime Text (UNREGISTERED)

Goto Tools Project Preferences Help

nunit.js

```
var fs = require('fs'),
    path = require('path'),
    _ = require('underscore'),
    msbuild = require('./msbuild.js'),
    sax = require('sax');

exports.findTestAssemblies = function(files, options) {
    var assemblies = [];
    var projects = [];
    files.forEach(function(file) {
        switch(path.extname(file)) {
            case '.sln': projects = projects.concat(msbuild.getSolutionProjectInfo(file)
                ); break;
            case '.csproj': projects.push(msbuild.getProjectInfo(file)); break;
            default: {
                if (!fs.existsSync(file)) throw new Error('Assembly not found: ' + file)
                    ;
                assemblies.push(path.normalize(file));
            }
        }
    });
    projects.
        filter(function(project) { return _.contains(project.references, '
            nunit.framework'); }).
        forEach(function(project) {
            var outputs = project.output.filter(function(output) { return fs.existsSync(
                output); });
            if (outputs.length === 0) throw new Error('No assemblies exist for project:
                ' + project.path);

            if (options && options.config) {
                outputs = outputs.filter(function(output) {
                    return output.toLowerCase().indexOf(options.config.toLowerCase()) >
                        -1;
                });
            }

            if (outputs.length === 0) throw new Error('No assemblies exist for project
                matching config parameter: ' + project.path);
            assemblies.push(path.normalize(outputs[0]));
```

Spaces: 4 JavaScript

What is Modifiability?

- **Modifiability** is about change and our interest in it is in the cost and risk of making changes.
- To plan for modifiability, an architect has to consider four questions:
  - What can change?
  - What is the likelihood of the change?
  - When is the change made and who makes it?
  - What is the cost of the change?

What can change?

- The functions of the system
- The platforms, i.e., the hardware, operating system, middleware
- The environment in which the system operates
  - The systems with which it must interoperate
  - The protocols it use to communicate
- The capacity
  - Number of users supported
  - Number of simultaneous operations

When is the change made and who makes it?

- Changes can be made during
  - **implementation** by modifying the source code
  - **build** by choice of libraries
  - **execution** by parameter setting, plugins, etc
- Changes can also be made by
  - a developer
  - an end user
  - a system administrator

What is the cost of the change?

- Involving two types of cost
  - The cost of introducing the mechanisms to make the system more modifiable
    - e.g. 最简单的变更机制是等待变更请求进来，然后变更源代码以适应该请求, cost = 0
  - The cost of making the modification using the mechanisms
    - e.g. cost -> max

It might be cheaper in the long run to build a sophisticated change-handling mechanism

Modifiability General Scenario

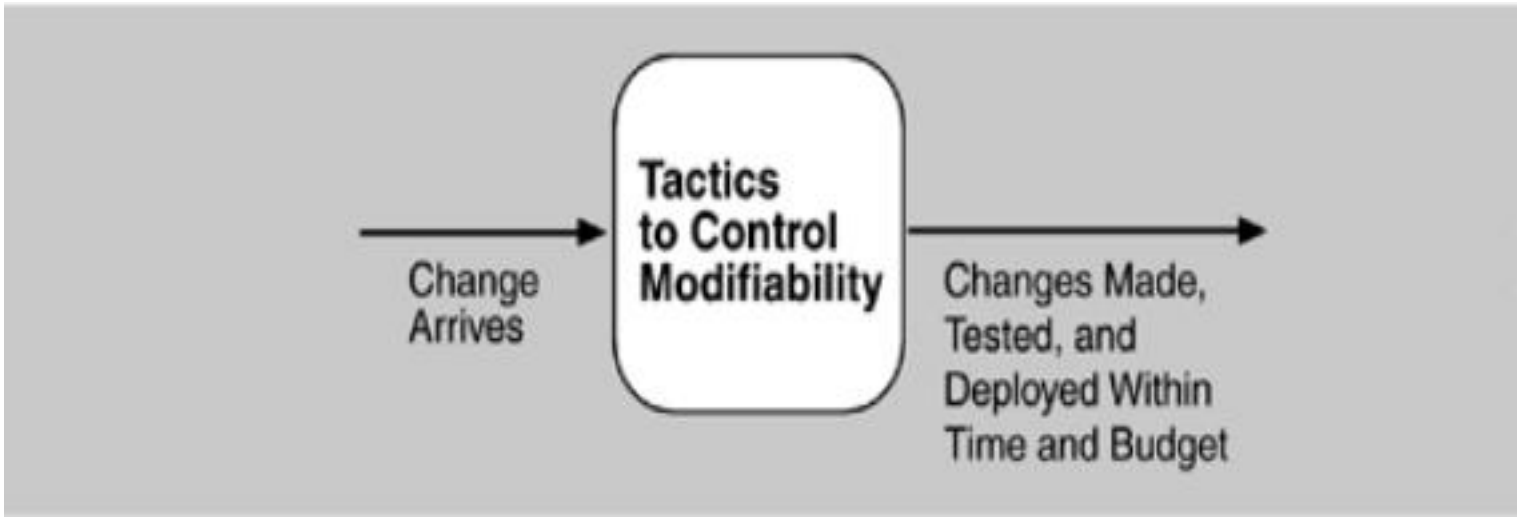
Portion of Scenario	Possible Values
Source	End user, developer, system administrator
Stimulus	A directive to add/delete/modify functionality, or change a quality attribute, capacity, or technology
Artifacts	Code, data, interfaces, components, resources, configurations, …
Environment	Runtime, compile time, build time, initiation time, design time
Response	One or more of the following: • make modification • test modification • deploy modification
Response Measure	Cost in terms of: • number, size, complexity of affected artifacts • effort • calendar time • money (direct outlay or opportunity cost) • extent to which this modification affects other functions or quality attributes • new defects introduced

Sample Concrete Modifiability Scenario

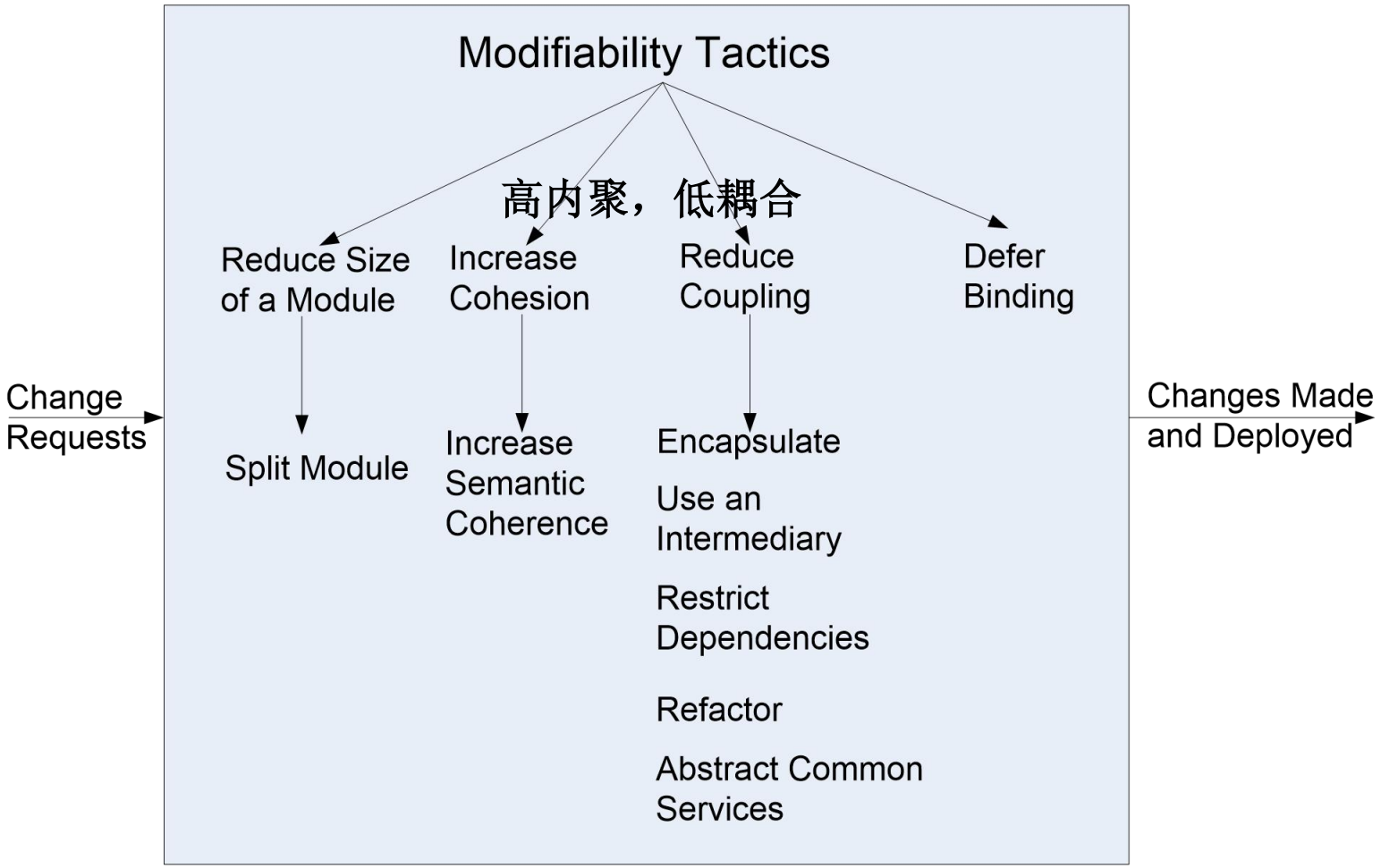
- The developer wishes to change the user interface by modifying the code at design time. The modifications are made with no side effects within three hours.
  - **Stimulus** – Wishes to change UI
  - **Artifact** – Code
  - **Environment**: Design time
  - **Response** – Change made
  - **Response measure** – No side effects in three hours
  - **Source** - Developer

Goal of Modifiability Tactics

- Goal of modifiability
  - controlling the complexity of making changes,
  - controlling the time and cost to make changes.



Modifiability Tactics



Reduce Size of a Module

- **Split Module**: If the module being modified includes a great deal of capability, the modification costs will likely be high.
- Refining the module into several smaller modules should reduce the average cost of future changes.

Increase Cohesion

- **Increase Semantic Coherence**: If the responsibilities A and B in a module do not serve the same purpose, they should be placed in different modules.
- This may involve creating a new module or it may involve moving a responsibility to an existing module.

- 内聚(Cohesion)用于度量一个模块的职责之间的关系有多紧密。非正式地，它衡量模块的“目标一致性”。

Reducing Coupling

- What is coupling?
- If two modules' responsibilities overlap, a single change may affect them both
- **Coupling** is measured by this overlap, i.e., by the probability that a modification to one module will propagate to the other
  - 如果两个模块的职责以某种方式重叠，那么对其中一个进行变更可能会影响到另一个
- High coupling is an enemy of modifiability

Reduce Coupling

- **Encapsulate（封装）**: Encapsulation introduces an explicit interface to a module. This interface includes an API and its associated responsibilities

```
class Book:
    def __init__(self, title, author):
        self.__title = title  # 私有属性，无法直接访问
        self.__author = author  # 私有属性

    # 提供公共方法访问书名
    def get_title(self):
        return self.__title
```

- **Use an Intermediary**: Given a dependency between responsibility A and responsibility B (for example, carrying out A first requires carrying out B), the dependency can be broken by using an intermediary.

Publish/Subscribe System

Introduction: Motivations for Pub/Sub model

- Traditional Client/Server communication model

  (Employs RPC, message queue etc..)

  - Synchronous, tightly-coupled request invocations.
    - 两个都得一起改
  - Very restrictive for distributed applications, especially for WAN and mobile environments.
  - When nodes/links fail, system is affected. Fault Tolerance must be built in to support this.

- Require a more flexible and **de-coupled** communication style that offers asynchronous mechanisms.

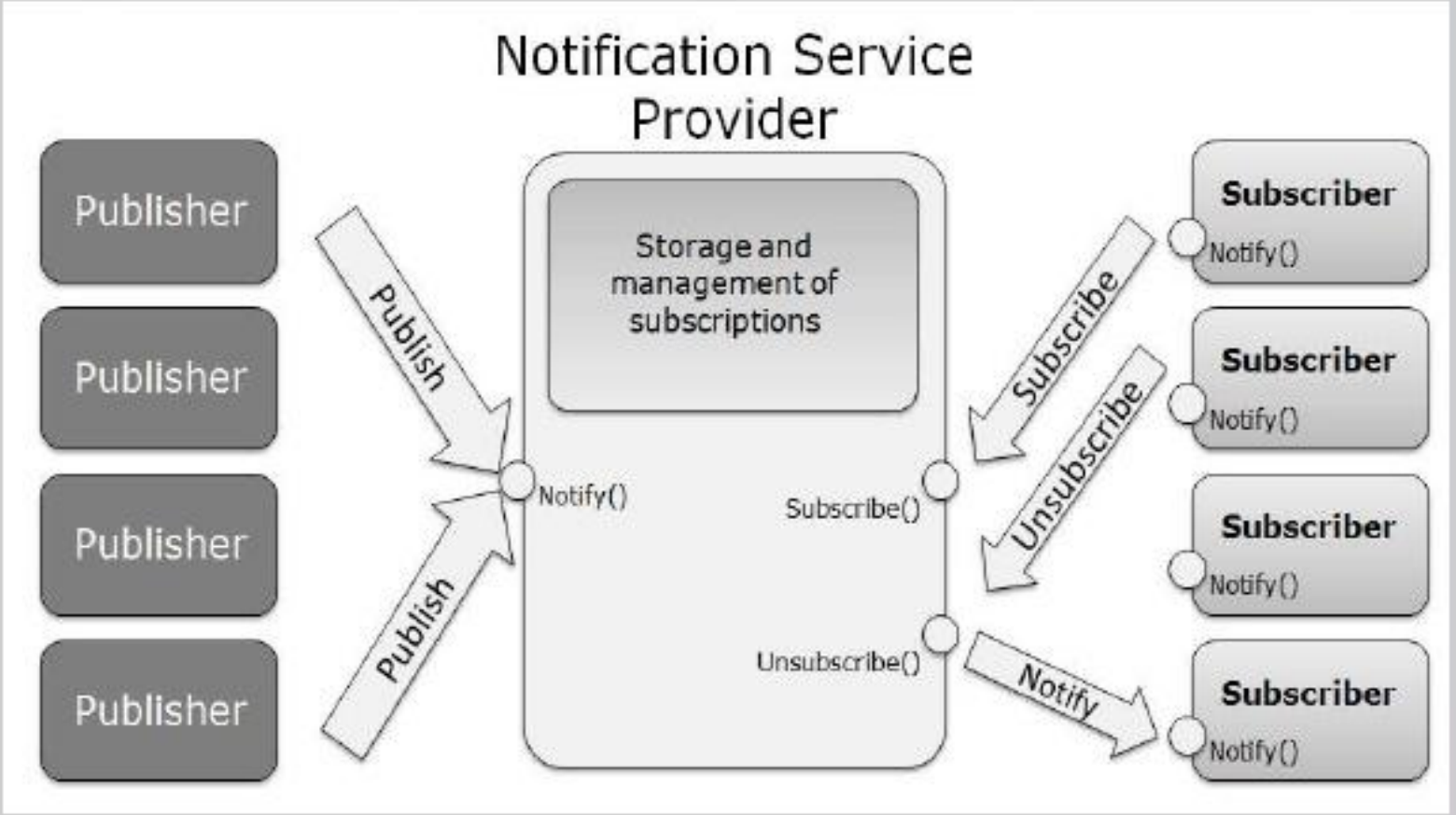
What is a Publish/Subscribe System?

- **Pub/Sub System** is a communication paradigm that allows freedom in the (distributed) system by the decoupling of communication entities in terms of time, space and synchronization.
  - 增删订阅者
  - 加删新功能
  - 发布者/订阅者只关注消息的生成/处理
  - e.g. 社交媒体/公众号/天气通知
- An event service system that is asynchronous, anonymous and loosely-coupled.
- Ability to quickly adapt in a dynamic environment.

Key components of Pub/Sub System

- **Publishers** : Publishers generate event data and publishes them.
- **Subscribers** : Subscribers submit their subscriptions and process the events received
- **P/S service**: It's the mediator/broker that filters and routes events from publishers to interested subscribers.

Publish-Subscribe Basic Model Overview



Example: Weather (Traditional Client/Server Model)

How It Works:

- The user’s device (client) sends a request to the weather service’ s server, usually using RPC (Remote Procedure Call).
- The server processes the request and sends back the current weather information.

Characteristics:

- Tight Coupling: The client and server are directly dependent on each other. The client needs to know the server's interface to request weather data correctly.
- Synchronous Requests: Every time the user wants an update, the client must manually request it from the server and wait for a response. If the server is slow, the user experience suffers.

Problems:

- Delayed Information: Users have to keep asking for updates, which might lead them to miss important information (like sudden weather changes).
- Load Issues: During high demand, the server may face many requests, causing performance problems.
- Poor Scalability: Adding new features (like personalized weather alerts) might require changes to both the client and server code, increasing maintenance costs.

Example: Weather (Publish/Subscribe)

How It Works:

- Users subscribe to weather updates for specific areas. When the weather changes, the service publishes the information to all subscribers.
- Users receive new weather information without having to request it manually.

Characteristics:

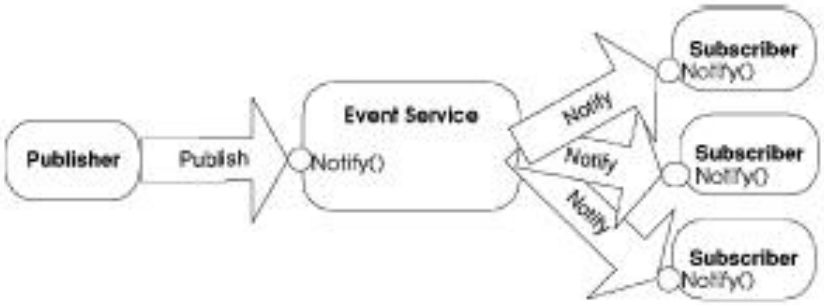
- Loose Coupling: The publisher (weather service) and subscribers (user devices) are not directly dependent on each other. Users just need to subscribe to the information they want, and the weather service doesn’t need to know the details of each user.
- Asynchronous Notifications: The weather service actively pushes notifications when information changes, allowing users to receive updates in real-time without waiting.

Advantages:

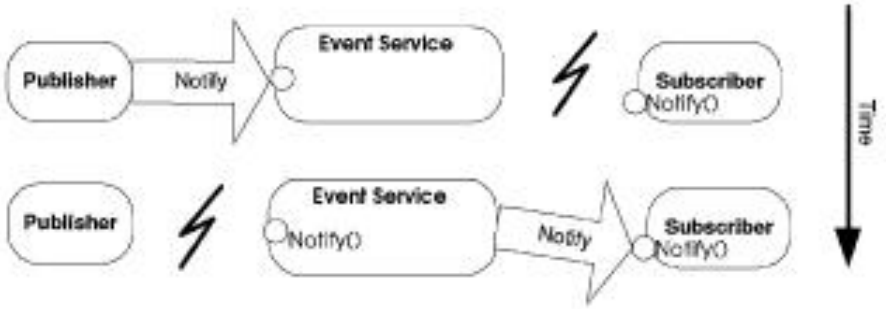
- Instant Updates: Users get timely alerts for sudden weather changes, improving their experience.
- Balanced System Load: The server sends out information during weather changes, reducing the request burden on users.
- Easy to Expand: New features, like multi-area subscriptions or personalized alerts, can be added easily without major changes to existing code.

Decoupling in time, space and synchronization

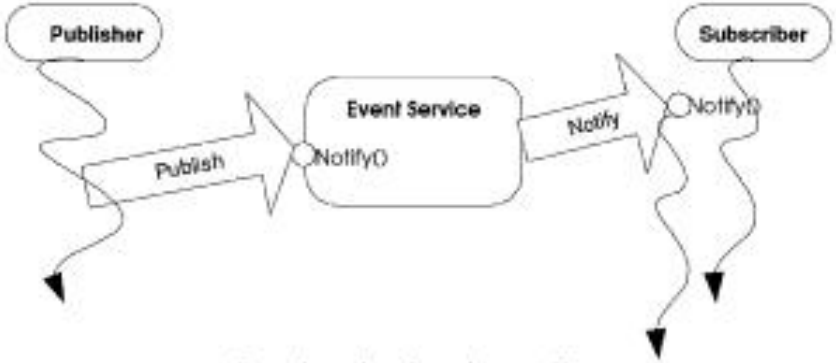
- Provides decoupling in time, space and synchronization.



Space decoupling



Time decoupling



Synchronization decoupling

Decoupling in time, space and synchronization

Spatial Decoupling

- Spatial decoupling means that publishers and subscribers do not need to know about each other. An intermediary service manages the communication between them, reducing direct dependencies.

Temporal Decoupling

- Temporal decoupling allows publishers and subscribers to not be active at the same time. A publisher can stop running after sending a message, and the subscriber will still receive it, as long as the intermediary has received and forwarded the message. This increases flexibility in how components operate.

Synchronization Decoupling

- Synchronization decoupling ensures that publishers are not blocked while producing events, and subscribers can be notified of events asynchronously while performing other tasks. This enhances the system's concurrency and responsiveness.

Classification of Pub/Sub Architectures

- **Centralized Broker model**
  - Consists of multiple publishers and multiple subscribers and centralized broker/brokers (an overlay network of brokers interacting with each other).
  - Subscribers/Publishers will contact 1 broker, and does not need to have knowledge about others.
  - E.g. CORBA (Common Object Request Broker Architecture) event services, JMS (Java Message Service) etc…

Classification of Pub/Sub Architectures

- **Peer-to-Peer model**
  - Each node can be publisher, subscriber or broker.
  - Subscribers subscribe to publishers directly and publishers notify subscribers directly. Therefore they must maintain knowledge of each other.
  - Complex in nature, mechanisms such as DHT and CHORD are employed to locate nodes in the network.
  - E.g. Java distributed event service
  - E.g. BitTorrent: In file-sharing applications, peers share pieces of files with each other, acting as both publishers (uploading pieces) and subscribers (downloading pieces).

Key Functions Implemented by P/S Middleware Service

- Event filtering (event selection)
  - The process of selecting the set of subscribers that have shown interest in a given event.
  - Subscriptions are stored in memory and searched when a publisher publishes a new event.
- Event routing (event delivery)
  - The process of routing the published events to all interested subscribers

Event Filtering (Subscription Model) Topic based VS Content based

- Topic based
  - Generally also known as topic based, group based or channel based event filtering.
  - Each event is published to one of these channels by its publisher.
  - Subscribers subscribes to a particular channel and will receive ALL events published to the subscribed channel.

Topic-based subscription

- Event filtering is easy. Simple process for matching an event to subscriptions.
- Limited expressiveness

Event Filtering- Subscription Model Topic based VS Content based

- Content based
  - More flexibility and power to subscribers, by allowing more expression in arbitrary/customized query over the contents of the event.
  - Event publication by a key/value attribute pair, and subscriptions specify filters using a explicit subscription language.

Content-based Subscription

- Added complexity in matching an event to subscriptions.
- However, more precision is provided and event routing is easier

Advantages of Pub/Sub

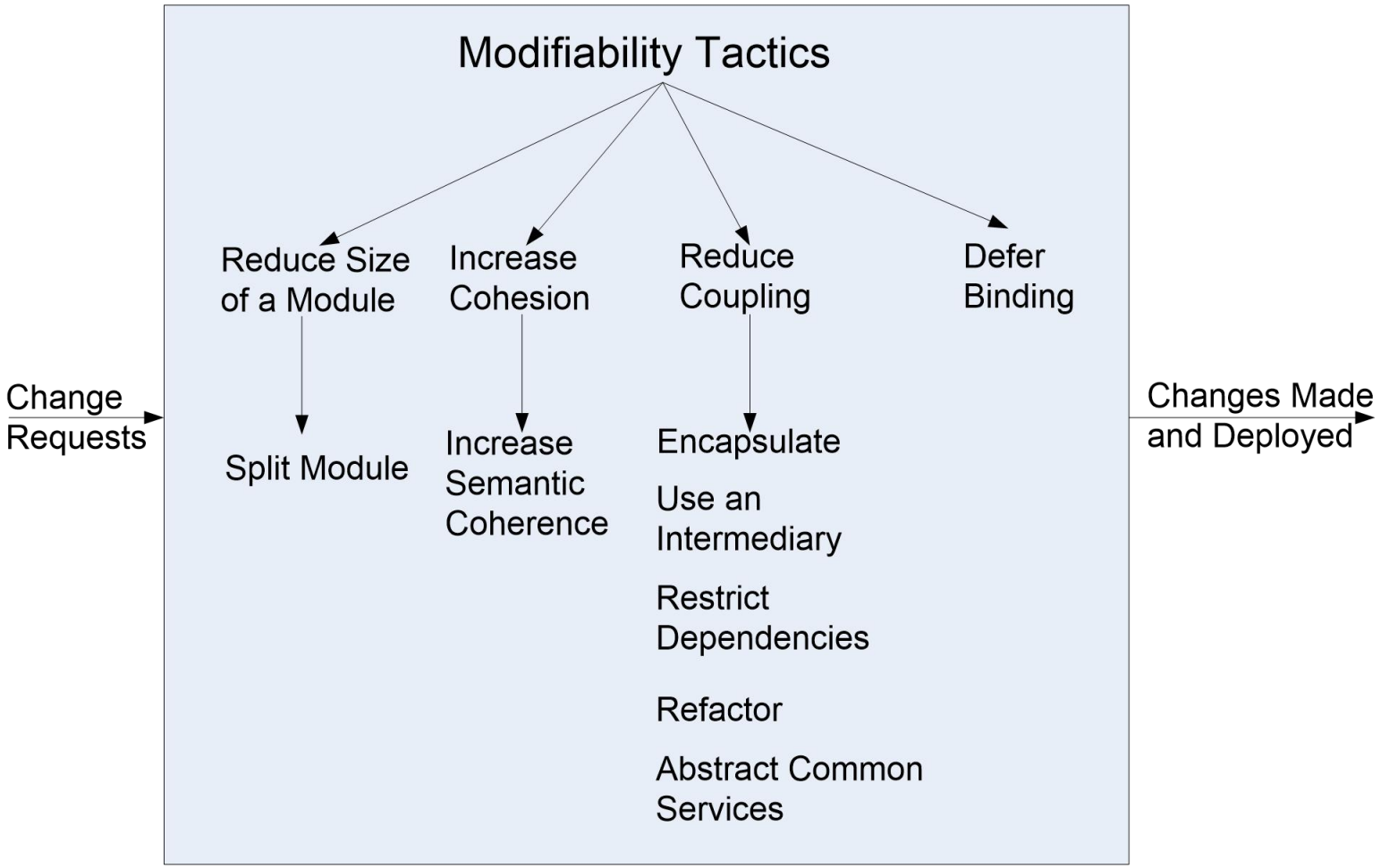
- Highly suited for mobile applications, ubiquitous computing and distributed embedded systems
- Robust – Failure of publishers or subscribers does not bring down the entire system
- Scalability- Suited to build distributed applications consisting a large number of entities
- Adaptability- can be varied to suit different environments (mobile, internet game, embedded systems etc…)

Disadvantages of Pub/Sub

- Reliability – no strong guarantee on broker to deliver content to subscriber. After a publisher publishes the event, it assumes that all corresponding subscribers would receive it.

- Potential bottleneck in brokers when subscribers and publishers overload them. (Solve by load balancing techniques)

Modifiability Tactics



Reduce Coupling

- **Restrict Dependencies**: restricts the modules which a given module interacts with or depends on.
- By restricting a module's visibility and by authorization
- For example,
  - a layer is allowed to see the modules in its bottom layer (layered module)

Reduce Coupling

- **Abstract Common Services**: where two modules provide not-quite-the-same but similar services, it may be cost-effective to implement the services just once in a more general (abstract) form.

Defer Binding

- In general, the later in the life cycle we can bind values, the better.
- If we design artifacts with built-in flexibility, then exercising that flexibility is usually cheaper than hand-coding a specific change.
- However, putting the mechanisms in place to facilitate that late binding tends to be more expensive.

Summary

- **Modifiability** deals with change and the cost in time or money of making a change, including the extent to which this modification affects other functions or quality attributes.
- Tactics to reduce the cost of making a change include making modules smaller, increasing cohesion, and reducing coupling.

Chapter 8: Performance

What is Performance?

- **It is about time**
- Performance is about time and the software system's ability to meet timing requirements
- When events occur, the system must respond to them in time
  - Events include interrupts, messages, requests from users or other systems, or clock events marking the passage of time

Why is Performance Important?

- There are some important problems that we know how to solve with computers, but we can't solve them because we can't do them fast enough.
- All systems have performance requirements, even if they are not stated explicitly.
  - A typing processor may not have any explicit performance requirements, but waiting an hour (or a minute or a second) before seeing typed characters appear on the screen is unacceptable.

Why is Performance Important?

- Typically, once a system is built and performance is found to be insufficient, performance improvements are required.
  - If performance is considered **in advance** when the system is designed, the size of the resource pool can be simply increased. Otherwise, your options are very limited.

How to measure?

- A system timing log can help you determine where time is wasted, so that you can focus on improving the performance of the key parts of the system.
  - It is not worthwhile to spend a lot of time optimizing for only a small performance gain.

Trade-Off

- It has frequently compromised the achievement of all other qualities.
  - Scalability, e.g., increasing your system's capacity for work, while still performing well
  - Modifiablity
  - Security
  - Interoperability

Performance General Scenario

Portion of Scenario	Possible Values
Source	Internal or external to the system
Stimulus	Arrival of a periodic, sporadic, or stochastic event
Artifact	System or one or more components in the system.
Environment	Operational mode: normal, emergency, peak load, overload.
Response	Process events, change level of service
Response Measure	Latency, deadline, throughput, jitter, miss rate

Sample Concrete Performance Scenario

- Users initiate transactions under normal operations. **The system** processes the transactions with an average latency of two seconds.
  - Stimulus: transaction arrivals
  - Source: users
  - Artifact: **the system**
  - Response: process the transactions
  - Response measure: average latency of two seconds
  - Environment: under normal operation

Performance Modeling

- Two basic contributors to the response time
- **Processing time** is the time that the system is working to respond
- **Blocked time** is the time that the system is unable to respond

Processing time

Suppose a message is generated by one component.

- It might be placed on the network, after which it arrives at another component. It is then placed in a buffer; transformed in some fashion; processed according to some algorithm; transformed for output; placed in an output buffer; and sent onward to another component, another system, or some actor.

Each of these steps consumes resources and time and contributes to the overall latency of the processing of that event.

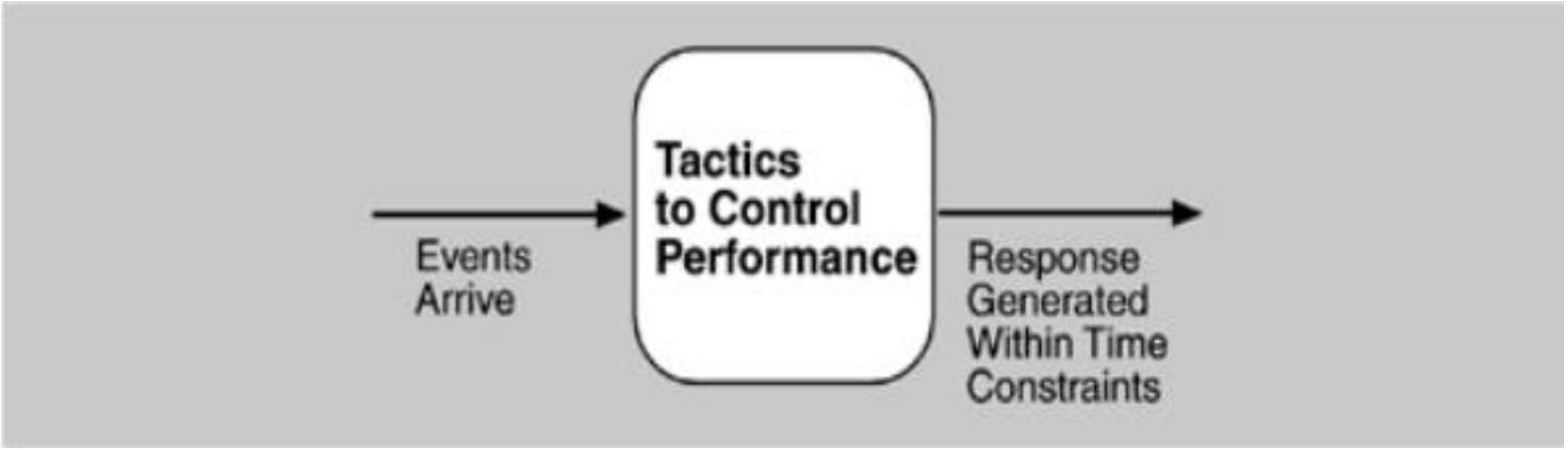
Blocked time

Blocked time is caused by

- **Contention for resources**: Many resources can only be used by a single client at a time. This means that other clients must wait for access to those resources.
- **Availability of resources**: Even in the absence of contention, computation cannot proceed if a resource is unavailable.
- **Dependency on other computations**: it must synchronize with the results of another computation or because it is waiting for the results of a computation that it initiated

Goal of Performance Tactics

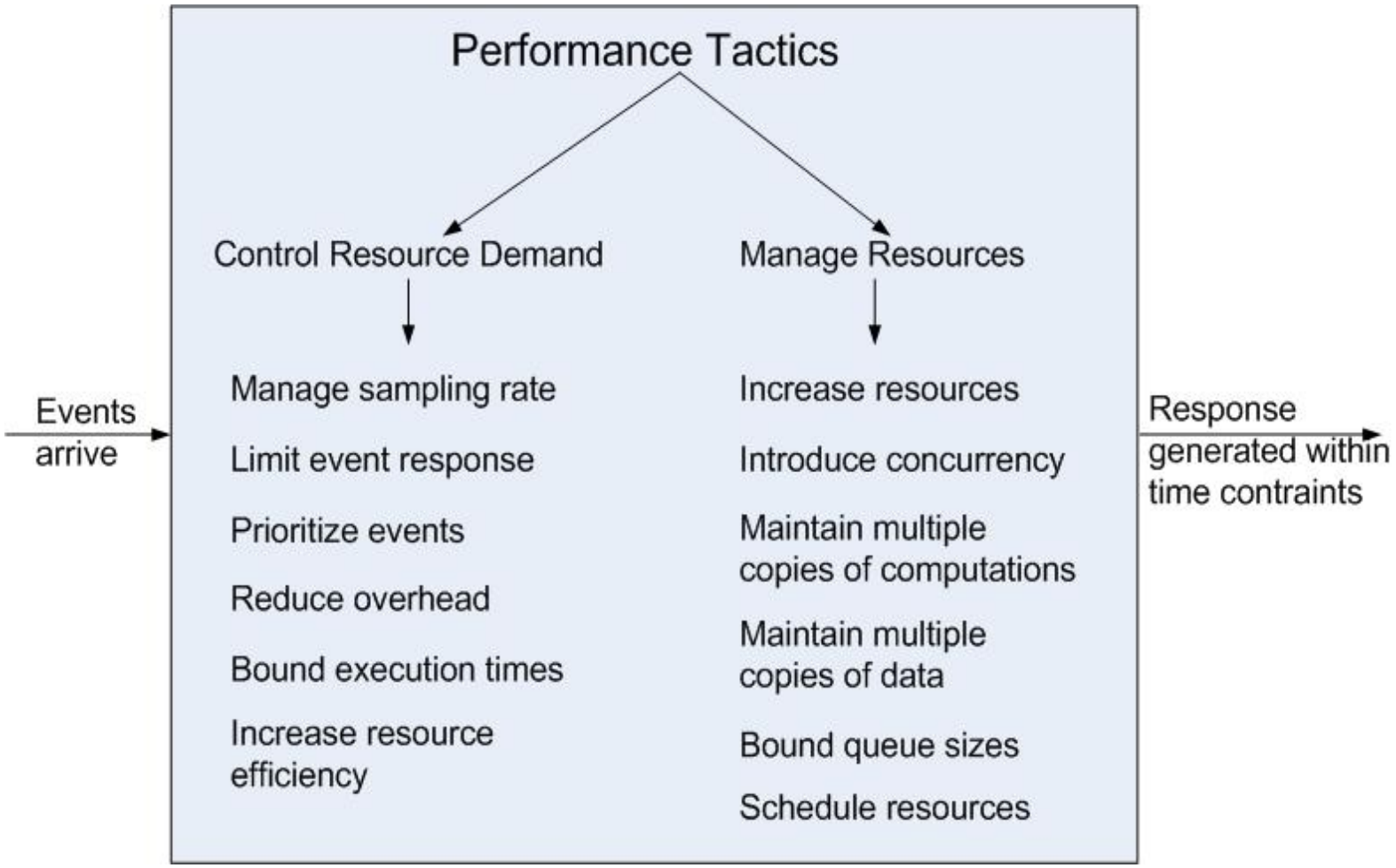
- To generate a response to an event arriving the system within some time-based constraint
- The event can be single or a stream, and is the trigger to perform computation



Two Tactic Categories

- Control resource demand
  - To produce smaller demand on the resources
  - Operate on the demand side
- Manage resources
  - To make the resources at hand work more effectively in handling the demands
  - Operate on the response side
- Resources
  - Hardware resources, e.g., CPU, data stores, network bandwidth, and memory
  - Software resources, e.g., buffers, or critical sections

Performance Tactics



Control Resource Demand

- **Manage Sampling Rate**: to reduce the sampling frequency at which a stream of data is captured
  - For example, signal processing systems; trade-off between accuracy and latency
- **Limit event response:** When discrete events arrive at the system (or element) too rapidly to be processed, then the events must be queued until they can be processed.

Control Resource Demand

- **Prioritize Events**: to impose a priority scheme that ranks events according to the importance
  - Ignore low-priority events when resources are not enough
  - For example, a building management system may raise a variety of alarms. Life-threatening alarms such as a fire alarm should be given higher priority than informational alarms such as a room is too cold.

Control Resource Demand

- **Bound Execution Times**: Place a limit on how much execution time is used to respond to an event.
  - In algorithm design, limiting the number of iterations is a method for bounding exec. time
  - Trade-off between the performance and accuracy
    - progressive, approximate

Control Resource Demand

- **Reduce Overhead**: The use of intermediaries increases the resources consumed in processing an event stream; removing them improves latency.
  - The use of intermediaries increases the resources consumed in processing an event stream, and so removing them improves latency (Tradeoff between the modifiability and performance)
  - A strategy for reducing computational overhead is to co-locate resources. Co-location may mean hosting cooperating components on the same processor to avoid the time delay of network communication

Control Resource Demand

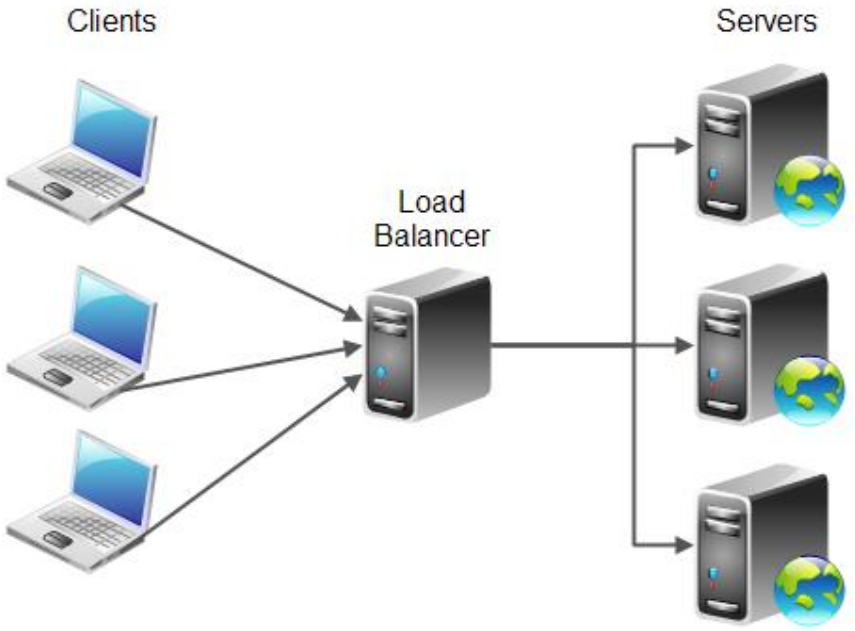
- **Increase Resource Efficiency**: Improving the algorithms used in critical areas will decrease latency.
- To reduce the complexity of the algorithm
- Index

Manage Resources

- **Increase Resources**: Faster processors, additional processors, additional memory, and faster networks all have the potential for reducing latency.
- **Increase Concurrency**: If requests can be processed in parallel, the blocked time can be reduced.
- Concurrency can be introduced by processing different streams of events on different threads

Maintain Multiple Copies of Computations

- The purpose of replicas is to reduce the resource contention on a single server
- Load balancer assigns new work to one of the duplicate server



Maintain Multiple Copies of Data

- **Data caching** is to keep copies of data on storage with different access speeds.
  - E.g., memory access v.s. disk access
  - Local access v.s. remote access via networks
- **Data replication** is to keep separate copies of data to reduce the contention from multiple simultaneous accesses
- How to choose the data to be cached/replicated
  - frequently accessed
- How to guarantee the consistency of multiple copies
  - Strong Consistency/Eventual Consistency

Scheduling

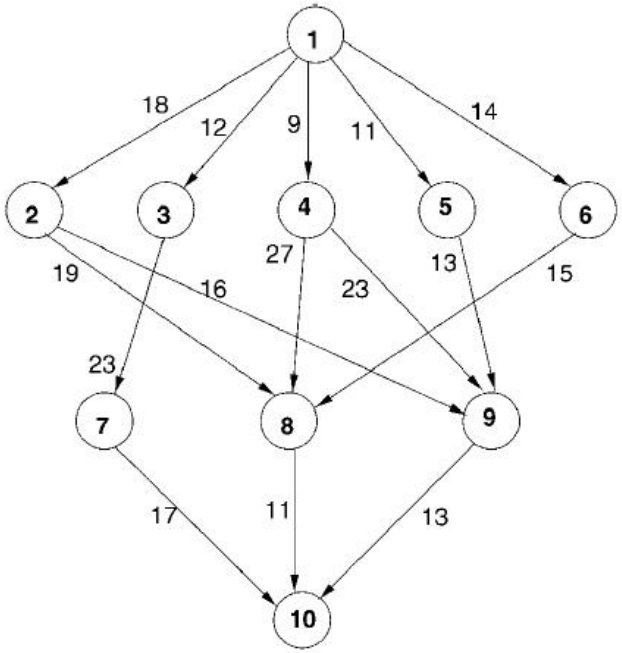
- When there is contention for a resource, the resource must be scheduled.
  - Processors needs to be scheduled
  - Buffers needs to be scheduled
  - Networks are scheduled

3-Dimension Framework for Scheduling Problem

1. Tasks
2. Resources
3. Objectives

Task Model

- Bag of tasks
- Directed Acyclic Graph (DAG)
- Periodic/cyclic tasks
- Task properties
  - Execution cost
  - Transmission cost
  - Arrival time
  - Deadline
  - Preemptive or non-preemptive …



Resource Model

- The resources include a set of machines/processors which are connected by networks
- Machine/processor model
  - Processing capability/speed, energy consumption
    - 通常涉及任务调度、负载均衡和资源分配等，提高计算效率和降低能耗
- Network model
  - Network topology
  - Bandwidths
  - Messages and energy consumption
  - E.g., sensor networks, data center networks, mobile cloud
    - 涉及数据传输、通信效率和网络可靠性等，目的是优化数据流动和减少延迟

Objectives

- Minimize completion time
- Meeting deadline
- Maximize throughput
- Minimize data transmission/messages
- Minimize energy consumption
- ...

Classification of Scheduling

- Real time scheduling v.s. non-real time scheduling
- Static scheduling v.s. dynamic scheduling
- Offline scheduling v.s. online scheduling
- Determinist scheduling v.s. Stochastic scheduling

Task Scheduling Problems

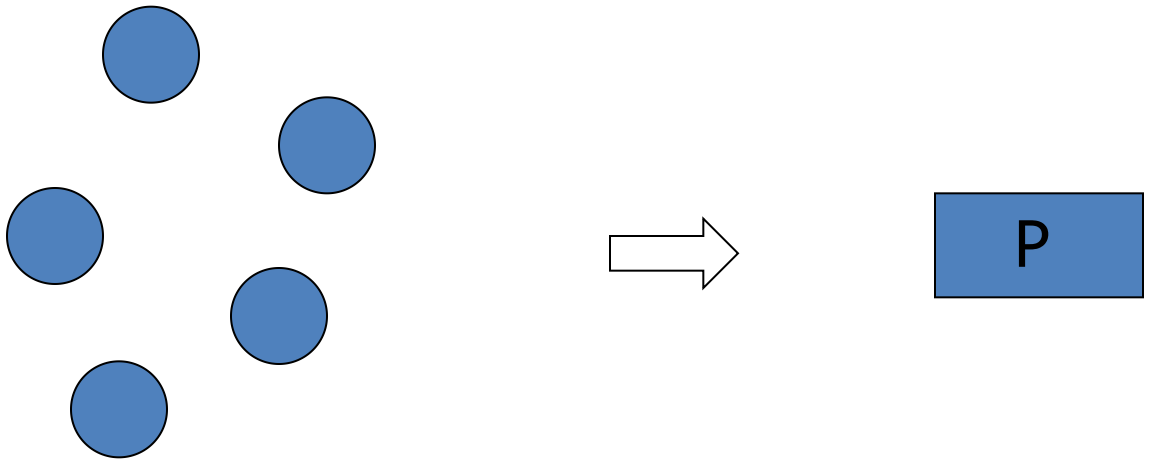
FIFO/FCFS

Fixed-priority scheduling

Dynamic-priority scheduling

1. Bag-of-Tasks scheduling on single processor
2. Bag-of-Tasks scheduling on multiple processors
3. DAGs scheduling on heterogeneous processors
4. Job shop scheduling

1. Bag-of-Tasks on Single Processor



Given: *release time*, *workload of each task*, or *deadline*

To determine **when** each task is executed

Objectives: *average completion time of the tasks*, or *meeting deadlines*

1. Bag-of-Tasks on Single Processor

The execution order of tasks is important for several reasons:

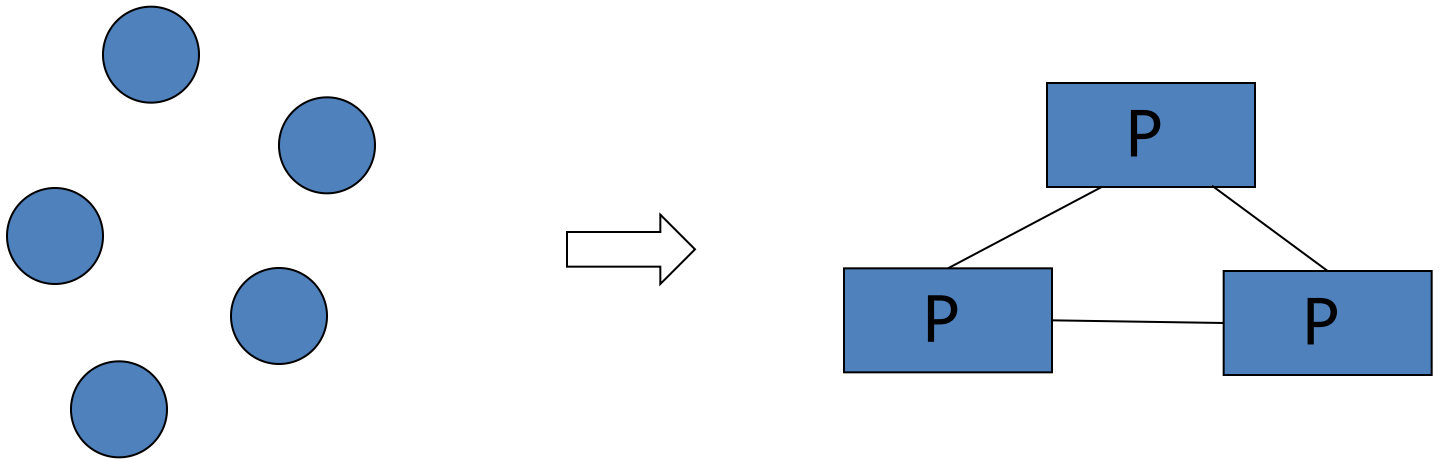
1. Waiting Time

- FCFS (First-Come, First-Served): Tasks are executed in the order they arrive, which can lead to longer average waiting times, especially if the first task is lengthy.
- SJF (Shortest Job First): This strategy prioritizes shorter tasks, which can reduce the waiting time for subsequent tasks, generally resulting in a lower average waiting time.

2. Response Time

- In scenarios requiring quick feedback, such as user interactions, choosing shorter tasks (SJF) can provide results faster, improving user experience.

2. Bag-of-Tasks on Multi-Processors

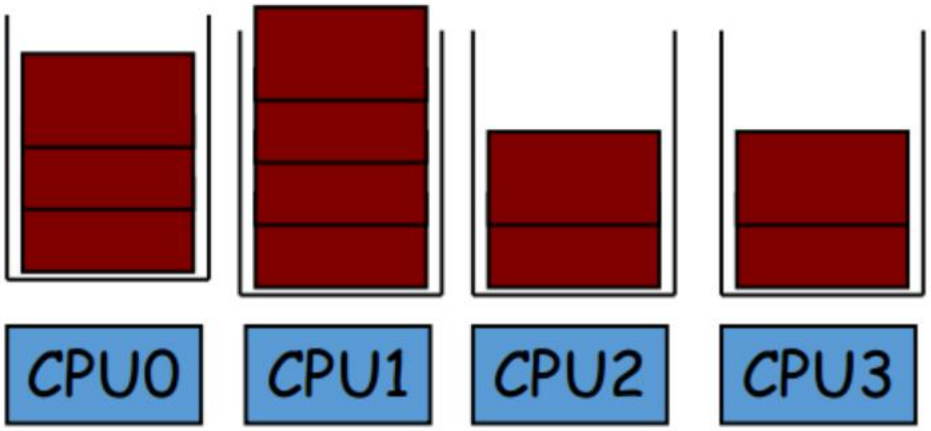


Given: *release time, workload of each task*

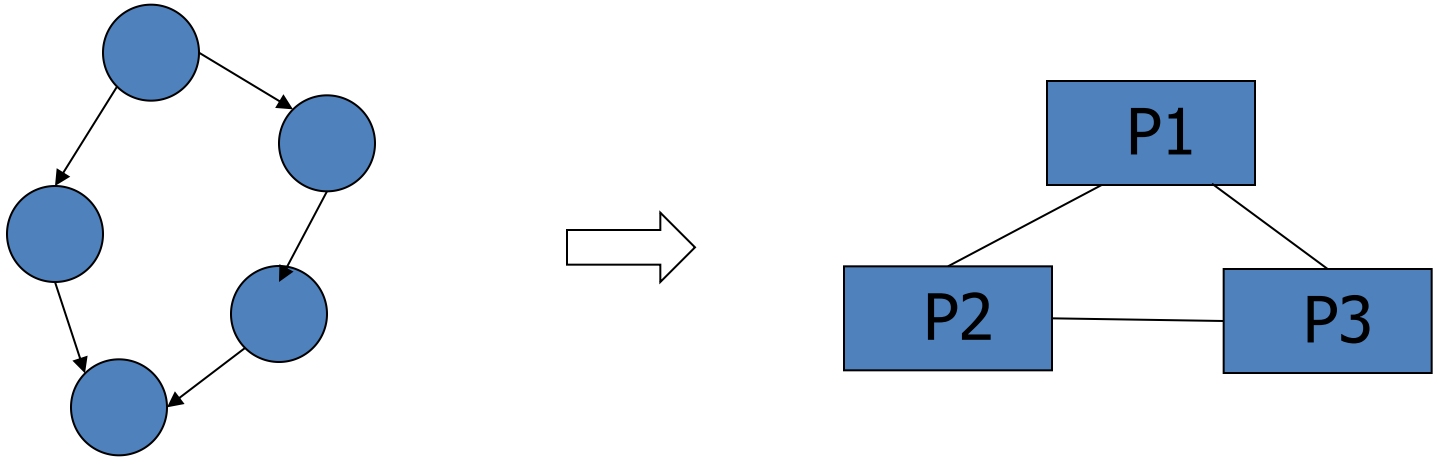
To determine **where and when** each task is executed

Objectives: make-span…

2. Bag-of-Tasks on Multi-Processors



3. DAGs Scheduling on Heterogeneous Processors

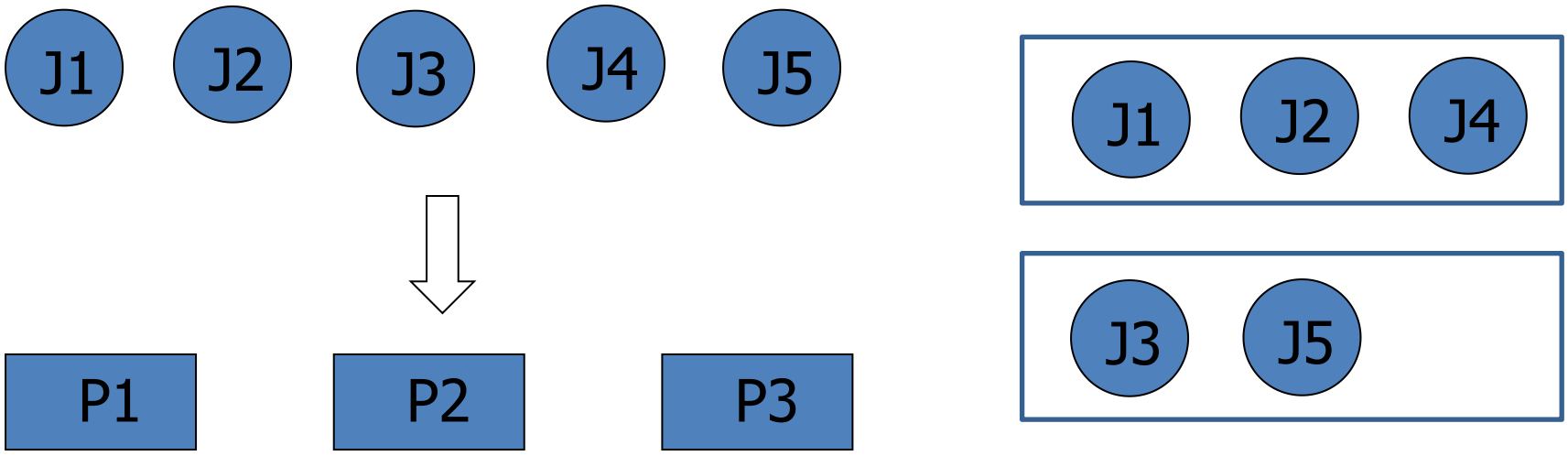


Given: *processing time* of every task on every processor, *communication time* on the edges

To determine **where and when** each task is executed

Objective: make-span …

4. Job Shop Problem (JSP) (np-hard)

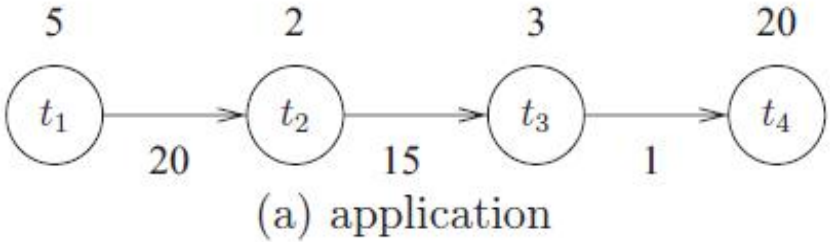


Given: *processing time* of every job on every processor

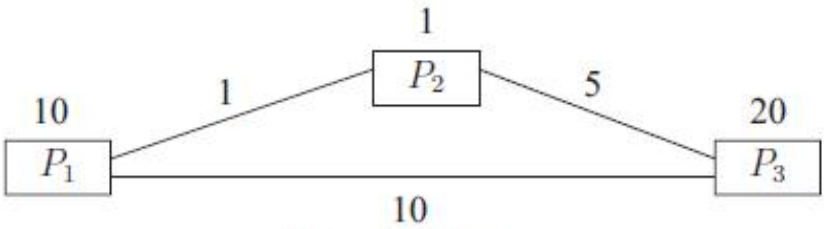
Constraint: every job is executed by every processor exactly once

Objective: make-span …

5. Periodic tasks scheduling

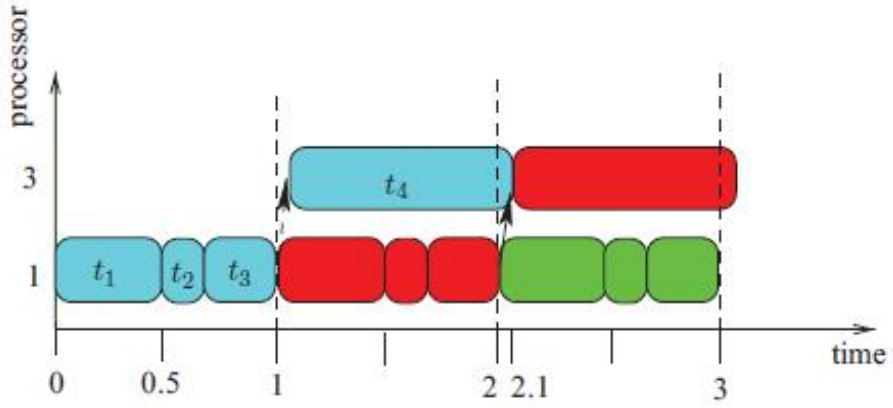


(a) application



(b) platform

The tasks are released periodically and executed in pipeline



Make-span: 2.1
Throughput: 1

Similar Terminologies

- Resource allocation
- Resource scheduling
- Task assignment
- Task placement
- Task allocation

The problems above are considered as the special cases/instances of the scheduling problem.

Categories of Scheduling Algorithms

Algorithms to solve optimization problems

1. Mathematical programing, i.e., integer programming, linear programing
2. Evolutionary algorithms, i.e, GA, PSO, Ant colony
3. Simple and competitive heuristics, i.e., list scheduling
4. Machine learning based algorithm, i.e., deep reinforcement learning

Scheduling Method: System Considerations

- Machine centric approach
  - Scheduling is triggered when a machine becomes idle
  - For each idle (空闲) machine, select the task according to some policies, e.g.,
    - First-Come-First-Serve (FCFS),
    - Shortest Job First (SJF),
    - Earliest Deadline First (EDF)
    - Job with the longest waiting time first, …
- Task centric approach
  - Scheduling is triggered done when a new task arrives
  - For each scheduled task, select the machine according to some policies, e.g., earliest finished time, …

Performance Tactics on the Road

- **Manage event rate**. **Lights** on highway entrance ramps let cars onto the highway only at set intervals, and cars must wait (queue) on the ramp for their turn.
- **Prioritize events**. **Ambulances and police**, with their lights and sirens going, have higher priority than ordinary citizens; some highways have high-occupancy vehicle (HOV) lanes, giving priority to vehicles with two or more occupants.
- **Maintain multiple copies**. Add **traffic lanes** to existing roads, or build parallel routes. In addition, there are some tricks that users of the system can employ:
- **Increase resources**. Buy a **Ferrari**, for example. All other things being equal, the fastest car with a competent driver on an open road will get you to your destination more quickly.
- **Increase efficiency**. Find a new route that is quicker and/or **shorter** than your current route.
- **Reduce computational overhead**. You can **drive closer** to the car in front of you, or you can **load more people** into the same vehicle (that is, carpooling).

Summary

- Performance is about the management of system resources in the face of particular types of demand to achieve acceptable timing behavior.
- Performance can be measured in terms of throughput and latency for both interactive and embedded real time systems.
- Performance can be improved by reducing demand or by managing resources more appropriately.