# 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.
- **Semantic interoperability** is the ability to interpret the data being exchanged.

# 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 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:<br>• the request is (appropriately) rejected and appropriate entities (people or systems) are notified<br>• the request is (appropriately) accepted and information is exchanged successfully<br>• the request is logged by one or more of the involved systems |
| **Response Measure** | One or more of the following:<br>• percentage of information exchanges correctly processed<br>• 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%.

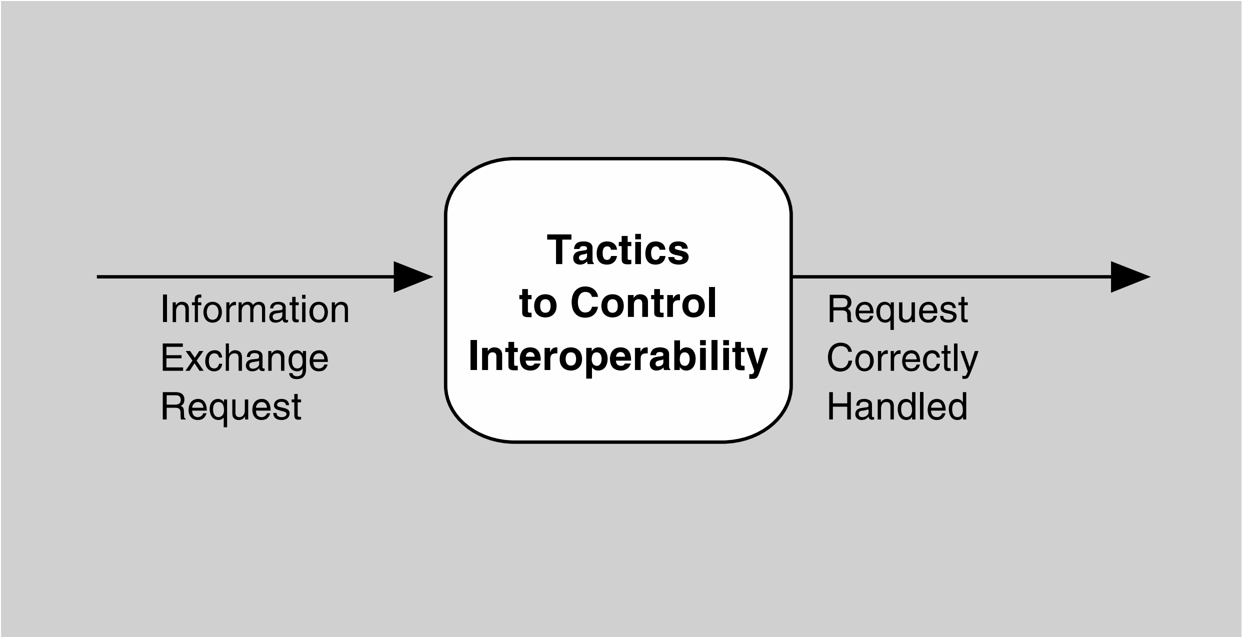
# SOAP v.s. REST

- 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

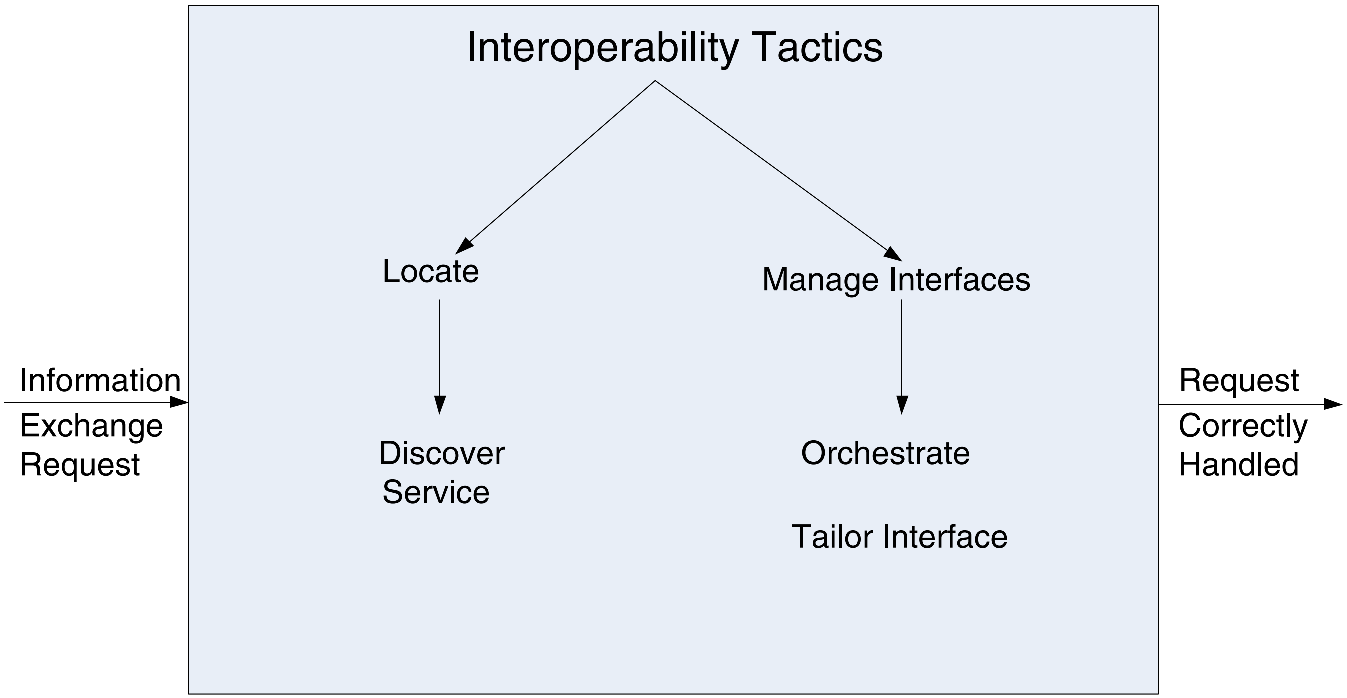
# Goal of Interoperability Tactics

- 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.

# Goal of Interoperability Tactics



# 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

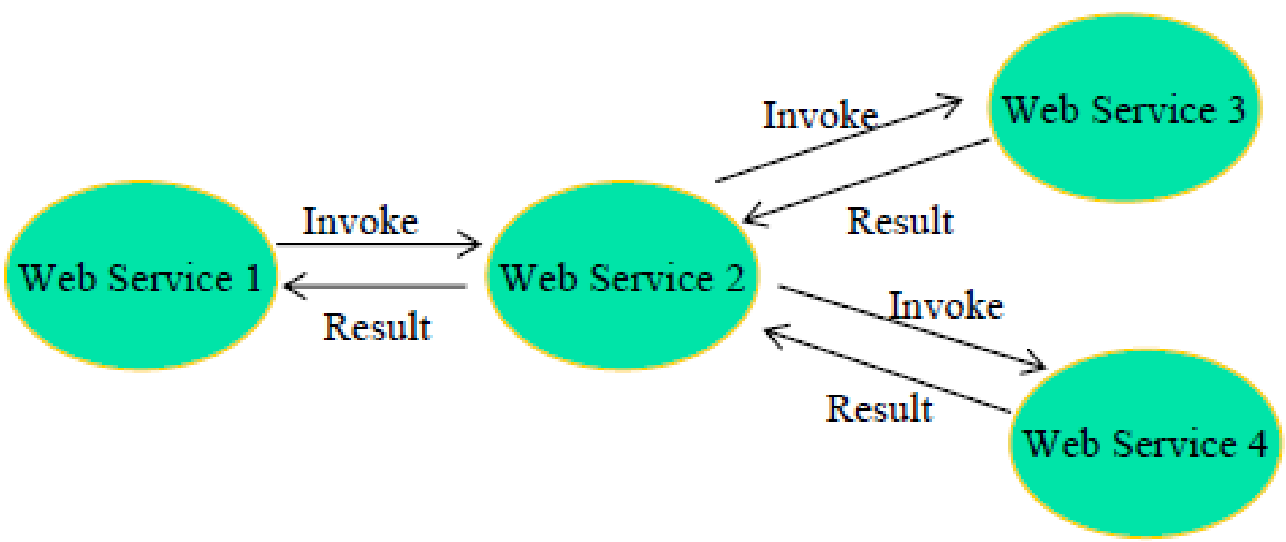
- Introduction of the “middlemen”, registry
  - The searched entity registers to a registry
  - The searcher can address to the registry to get information and find the searched entity
- Example
  - Service providers register their web services at UDDI registry which can be searched and found by Service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

# Searching Method – Registration

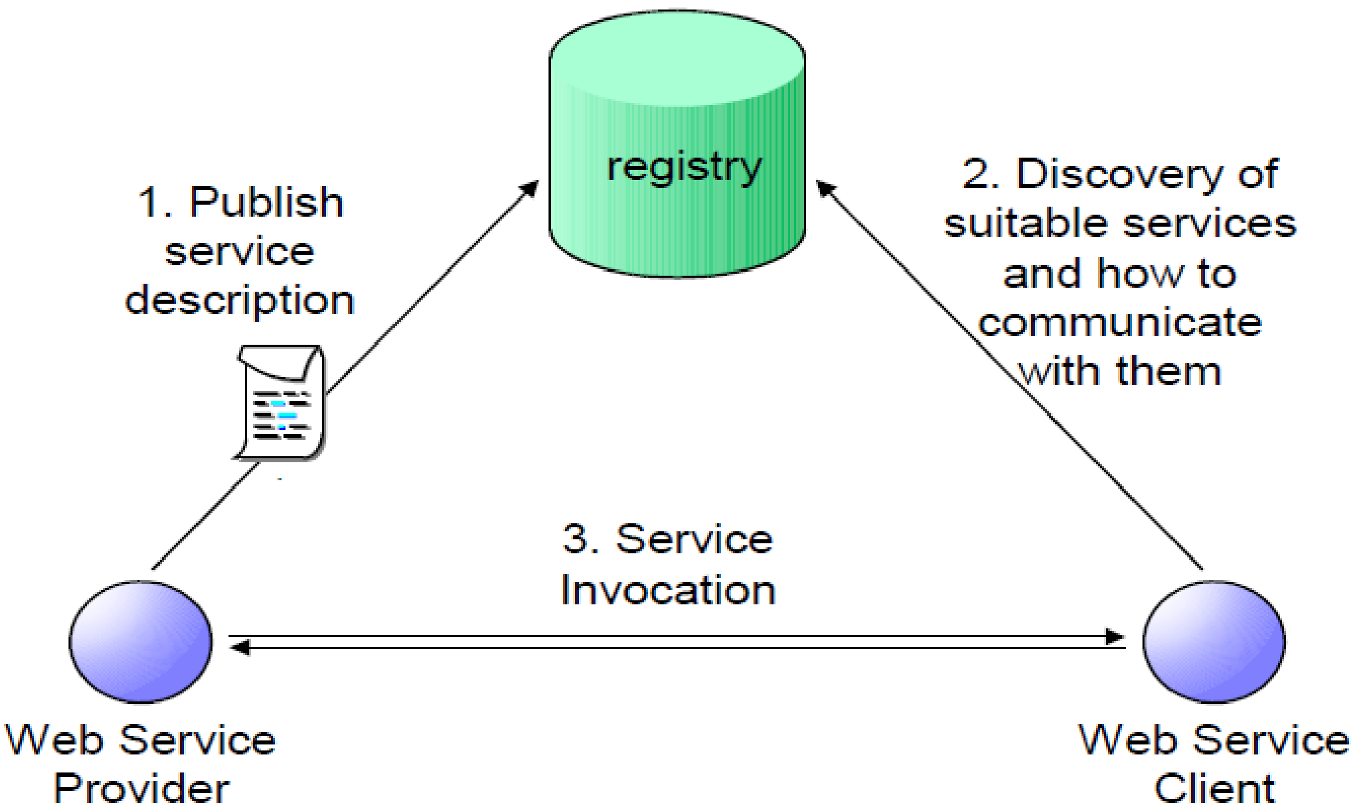

Less resource consumption on both searchers, searched, and less disturbing to environment, but the registry must be available, reliable, and correct

# 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

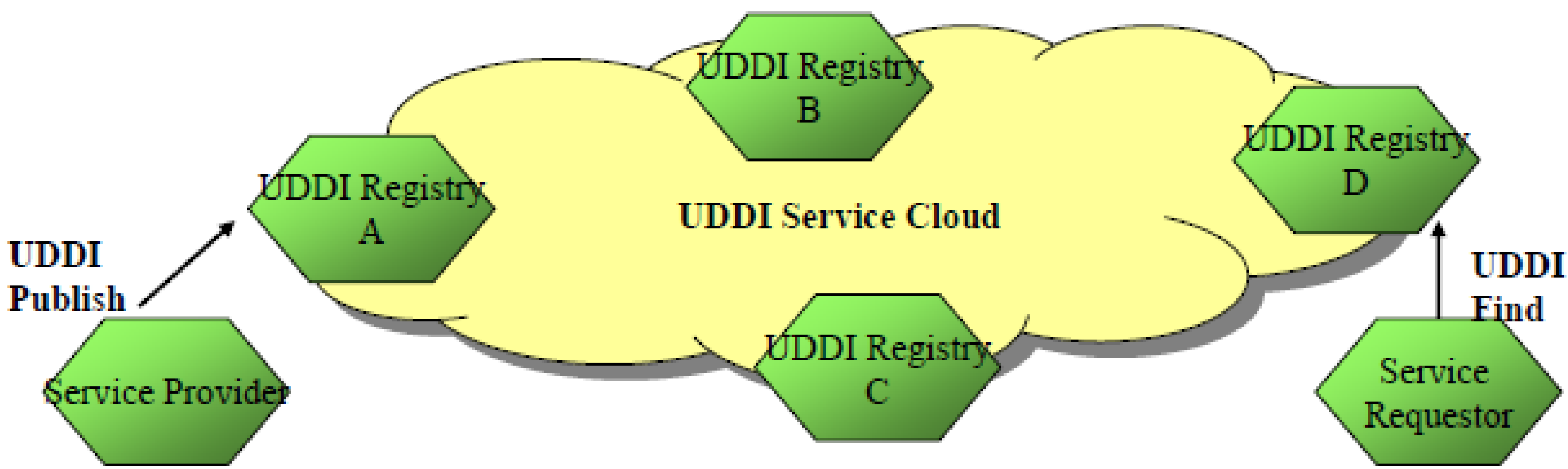


# Web Service Architecture

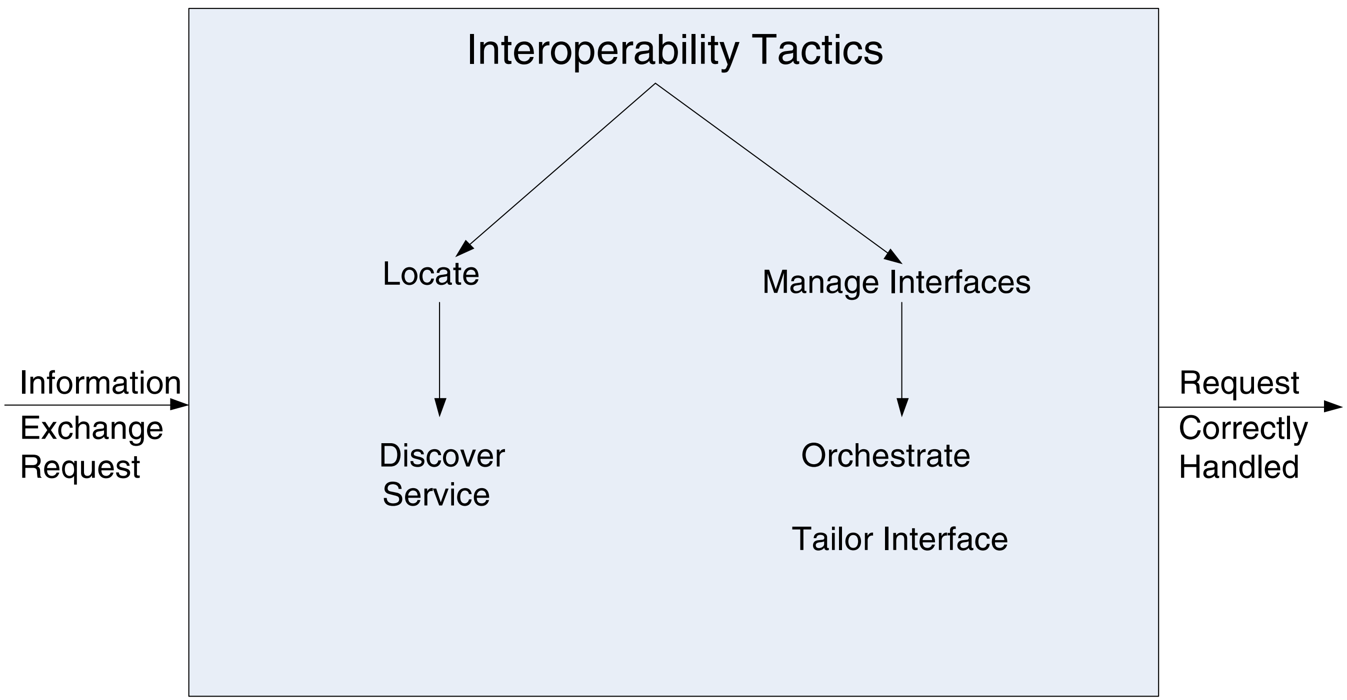
| |
|---|
| **Find**<br>Universal Description Discovery and Integration<br>UDDI |
| **Describe**<br>Web Service Description Language<br>WSDL |
| **Invoke**<br>Simple Object Access Protocol<br>SOAP |
| **Data format**<br>XML, XML Schema |
| **Transport**<br>HTTP, SMTP… |

# 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



# 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.

# Checklist for Interoperability

- Allocation of Responsibility
- Coordination Model
- Data Model
- Mapping among Architectural Elements
- Resource Management
- Binding Time

# Allocation of Responsibility

- Ensure that responsibility has been allocated
  - to detect a request to interoperate with external systems
  - to accept/reject the request
  - to exchange the information
  - to notify appropriate entities
  - log the request, which is essential for interoperability in an untrusted environment

# Coordination Model

- **Volume of traffic** on the network both created by the systems under your control and generated by system not under your control
- **Timeliness** of the messages being sent by your systems
- **Jitter** of the messages’ arrival times
- The system under your control makes assumptions about protocols and underlying networks that are **consistent** with systems not under your control

# Data Model

- Determine the syntax and semantics of the major data abstractions to be exchanged
- Ensure that these data abstractions are consistent with data from the interoperating systems
- If a system's data model is confidential, it is needed to transform to and from the data abstraction with which it interoperates

# Mapping among Architectural Elements

- For interoperability, the critical mapping is that of components to processors
- Make sure that components that communicate externally are hosted on processors that can reach the network

# Resource Management

- Interoperation with another system can never exhaust critical system resources
  - E.g., can a flood of requests cause service to be denied
- Resource load imposed by communications is acceptable
- If interoperation requires that resources be shared among the participating systems, an adequate policy is in place
  - E.g., bandwidth scheduling for video sharing

# Binding Time

- Determine when the systems become known to each other
- It has a policy for dealing with binding to both known and unknown external systems
- It has mechanisms to reject unacceptable bindings and to log such requests

# 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.

# Chapter 7: Modifiability

# 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 that the system computers
- 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
  - The cost of making the modification using the mechanisms
- Example
  - User interface builder

# Modifiability General Scenario

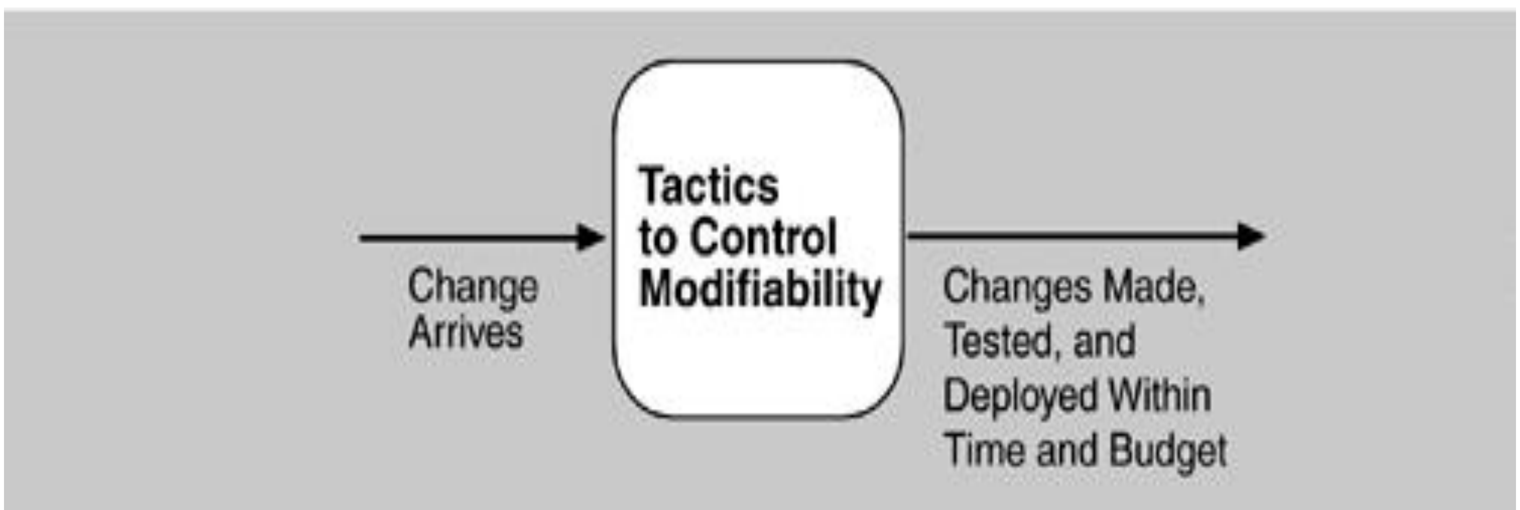
| Portion of Scenario | Possible Values |
|---|---|
| Source | End user, developer, system administrator |
| **Stimulus** | A directive to add/delete/modify functionality, or change a quality attribute, capacity, or technology |
| Artifacts | Code, data, interfaces, components, resources, configurations,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

# Sample Concrete Modifiability Scenario

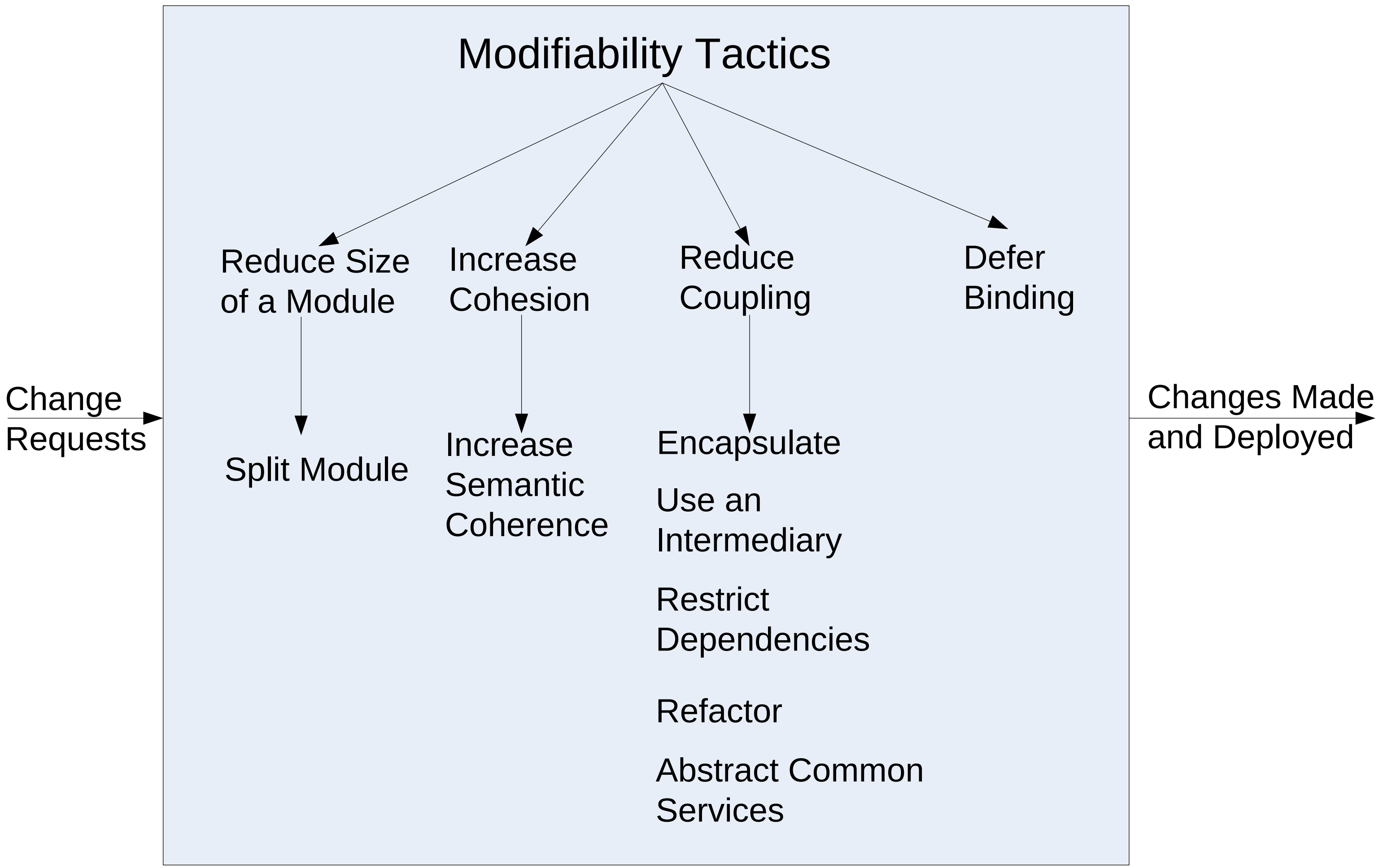
- The developer wishes to change the user interface by modifying the code at design time. The modifications are made with no side effects within three hours.
  - **Stimulus** – Wishes to change UI
  - **Artifact** – Code
  - **Environment**: Design time
  - **Response** – Change made
  - **Response measure**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

# 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
- High coupling is an enemy of modifiability

# Reduce Coupling

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

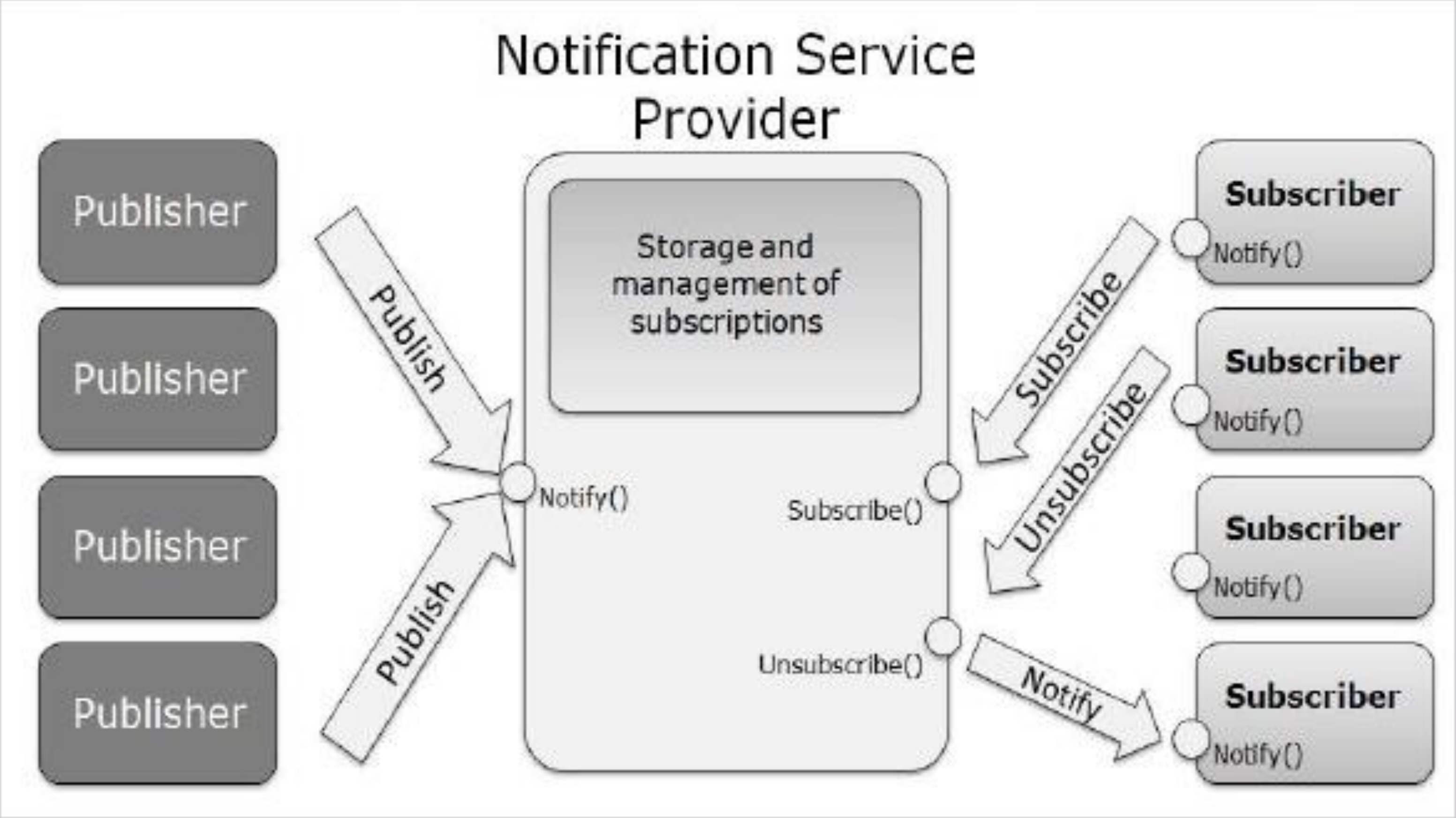
# What is a Publish/Subscribe System?

- **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.
- 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

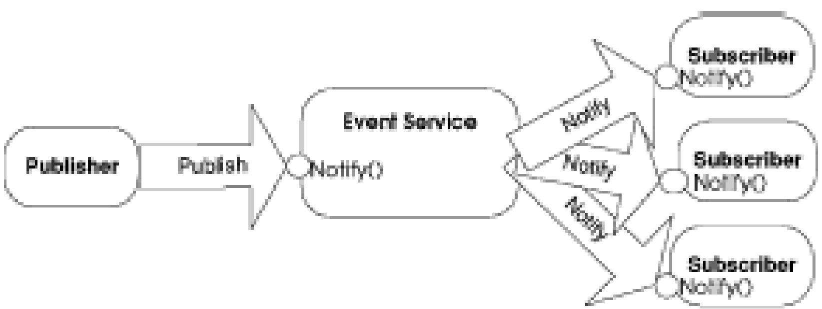
- **Publishers** : Publishers generate event data and publishes them.
- **Subscribers** : Subscribers submit their subscriptions and process the events received
- **P/S service**: It’s the mediator/broker that filters and routes events from publishers to interested subscribers.

# Publish-Subscribe Basic Model Overview

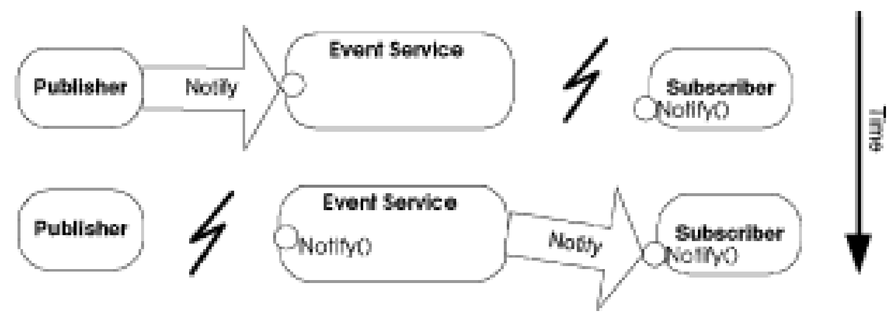


# Decoupling in time, space and synchronization

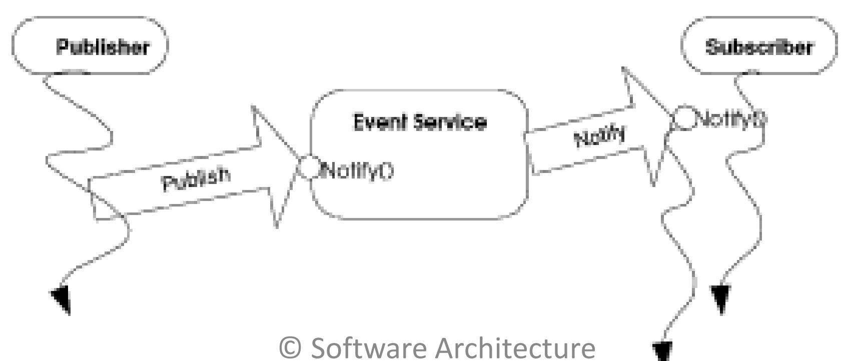
- Provides decoupling in time, space and synchronization.




Space decoupling




Time decoupling






Synchronization decoupling

# 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 event services, JMS, JEDI 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

# 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

- Simple process for matching an event to subscriptions. However, limited expressiveness.
- Event filtering is easy, event routing is difficult (Heavy load on the network). The challenge is to multicast event effectively to subscribers.

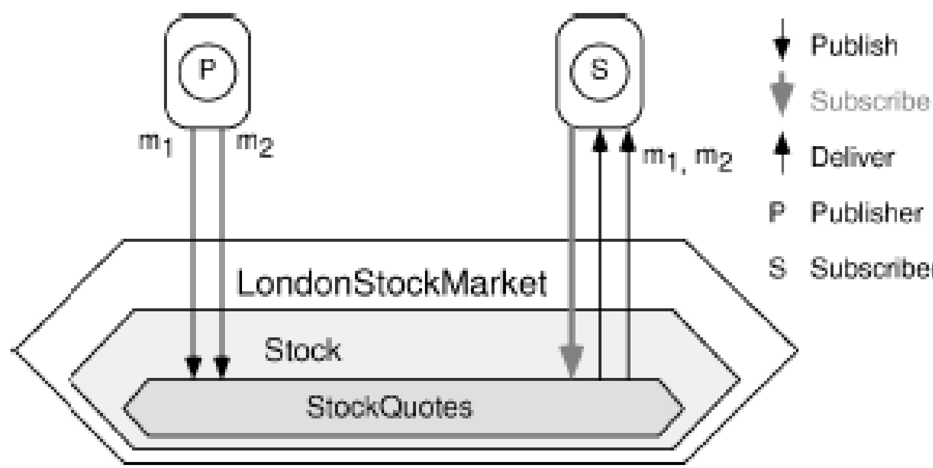



Fig. 12. Topic-based publish/subscribe interactions.

# 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.
  - E.g. Notify me of all stock quotes of IBM from New York stock exchange if the price is greater than 150

# Content-based Subscription

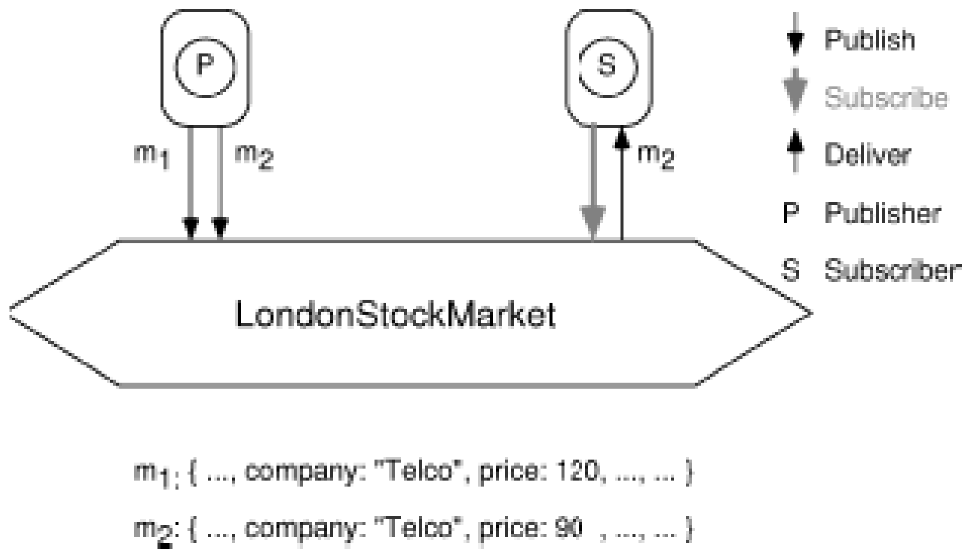



**Fig. 14.** Content-based publish/subscribe interactions.

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

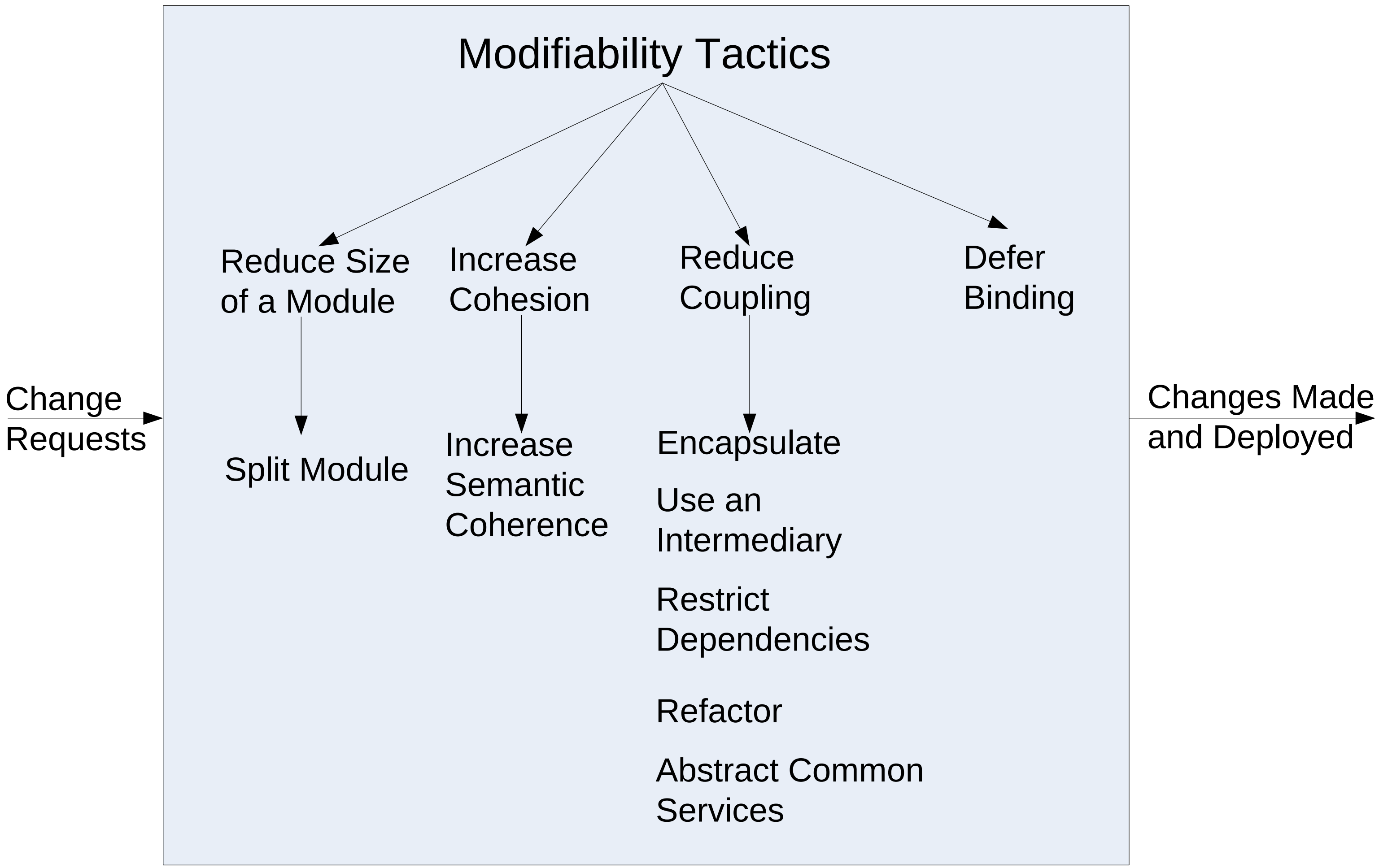
# Advantages of Pub/Sub

- 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

# 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.

# 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

# 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

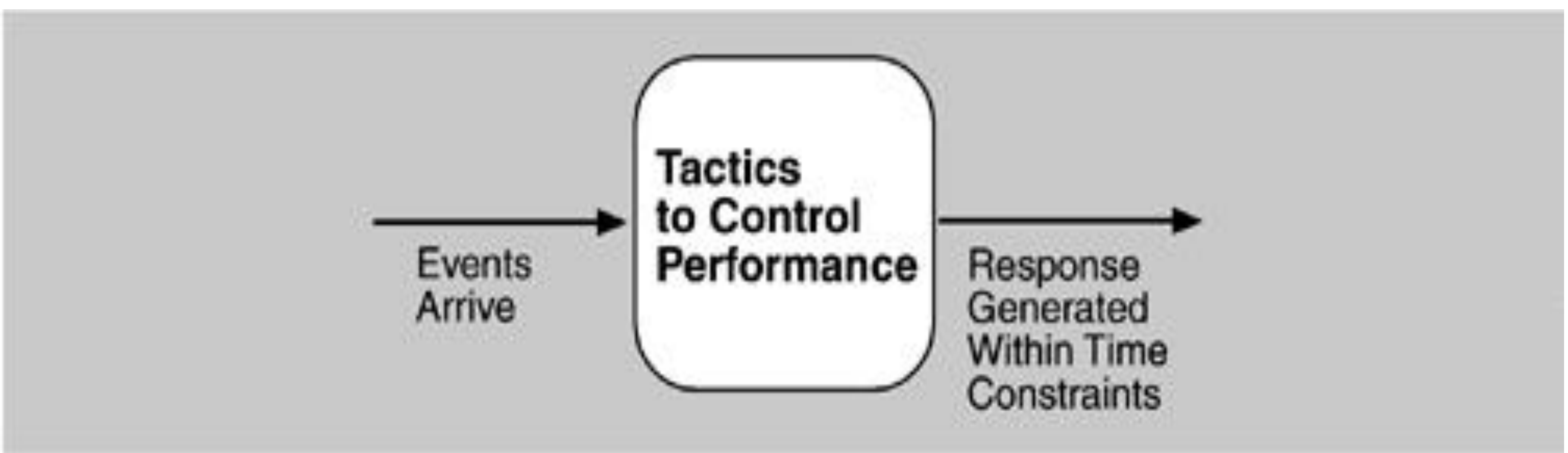
- Users initiate transactions under normal operations. **The system** processes the transactions with an average latency of two seconds.
  - Stimulus: transaction arrivals
  - Source: users
  - Artifact: **the system**
  - Response: process the transactions
  - Response measure: average latency of two 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
- Blocked time is caused by
  - Contention for resources
  - Availability of resources
  - Dependency on other computations

# Goal of Performance Tactics

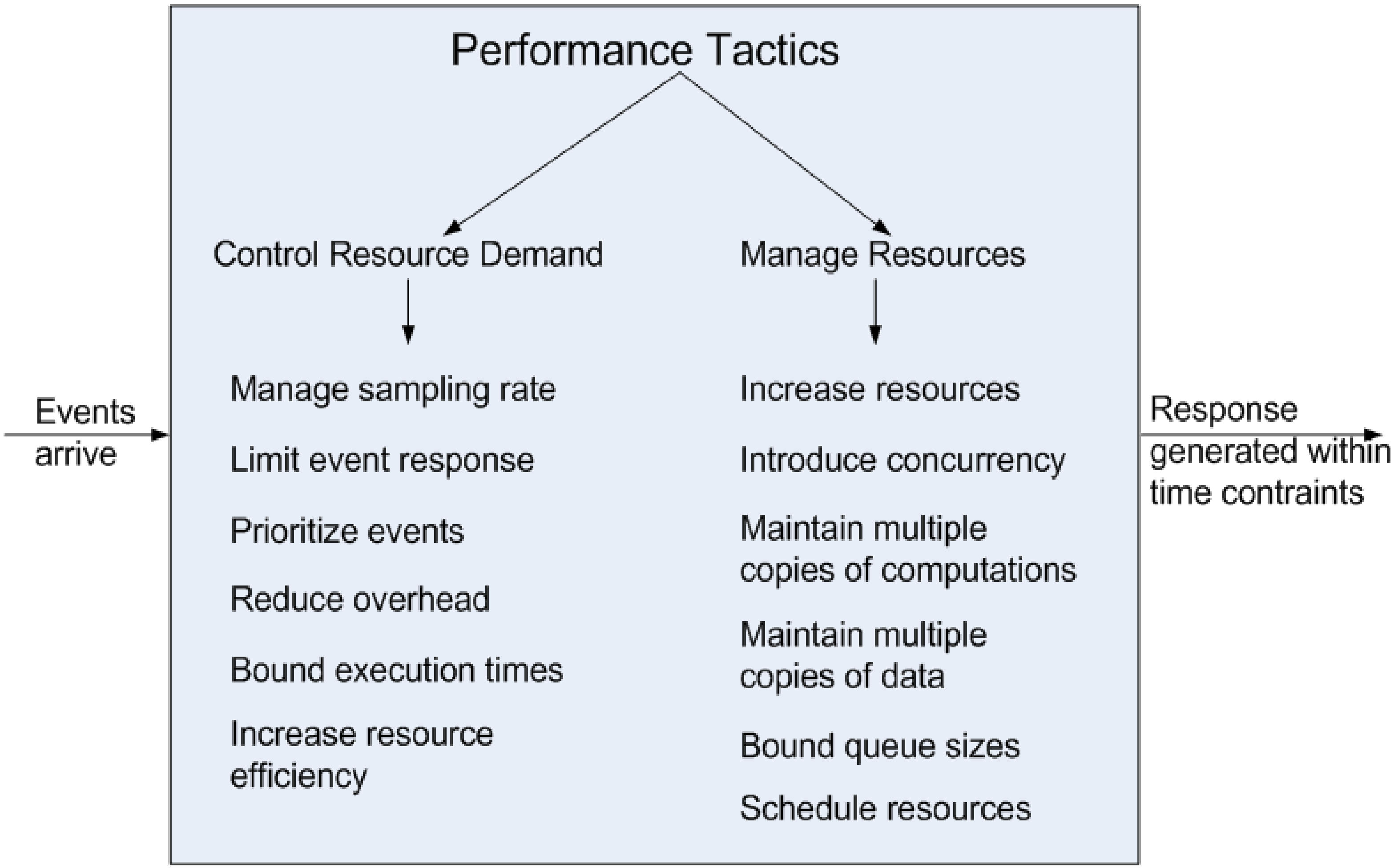
- To generate a response to an event arriving the system within some time-based constraint
- The event can be single or a stream, and is the trigger to perform computation



# 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
- **Prioritize Events**: to impose a priority scheme that ranks events according to the importance
  - Ignore low-priority events when resources are not enough

# Control Resource Demand

- **Reduce Overhead**: The use of intermediaries increases the resources consumed in processing an event stream; removing them improves latency.
  - Tradeoff between the modifiability and performance
- **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

# Control Resource Demand

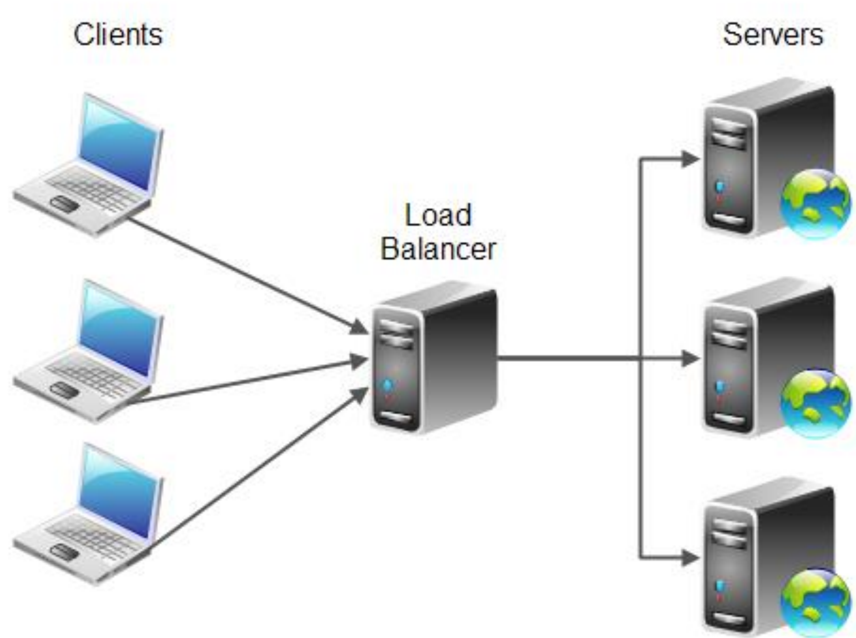
- **Increase Resource Efficiency**: Improving the algorithms used in critical areas will decrease latency.
- To reduce the complexity of the algorithm

# 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
- How to guarantee the consistency of multiple copies

# Scheduling

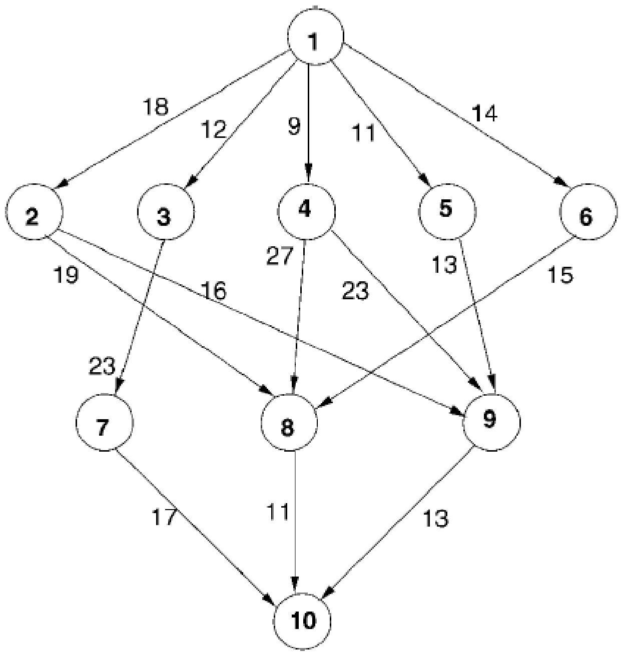
- When there is contention for a resource, the resource must be scheduled.
  - Processors needs to be scheduled
  - Buffers needs to be scheduled
  - Networks are scheduled

# 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
- Network model
  - Network topology
  - Bandwidths
  - Messages and energy consumption
  - E.g., sensor networks, data center networks, mobile cloud

# Objectives

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

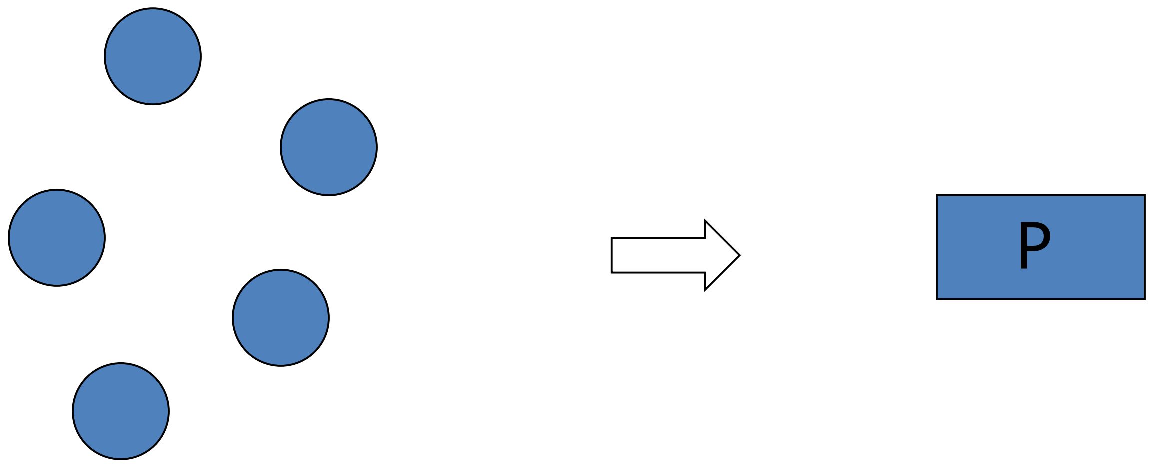
# Classification of Scheduling

- 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

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
5. Periodic tasks scheduling

# 1. Bag-of-Tasks on Single Processor

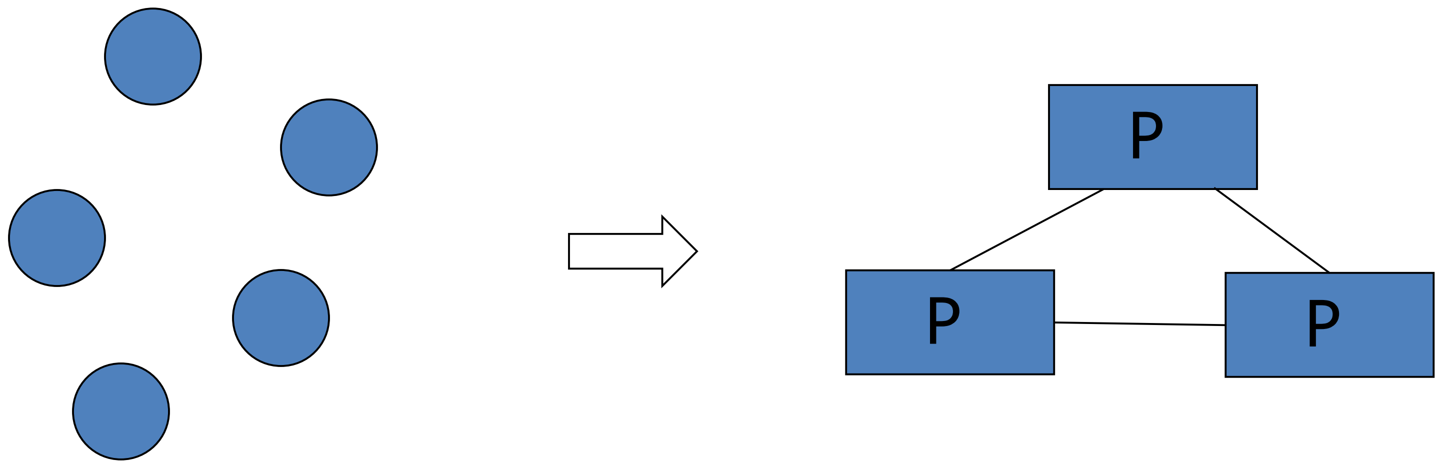



**Given**: *release time*, *workload of each task*, or *deadline*

To determine **when** each task is executed

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

# 2. Bag-of-Tasks on Multi-Processors

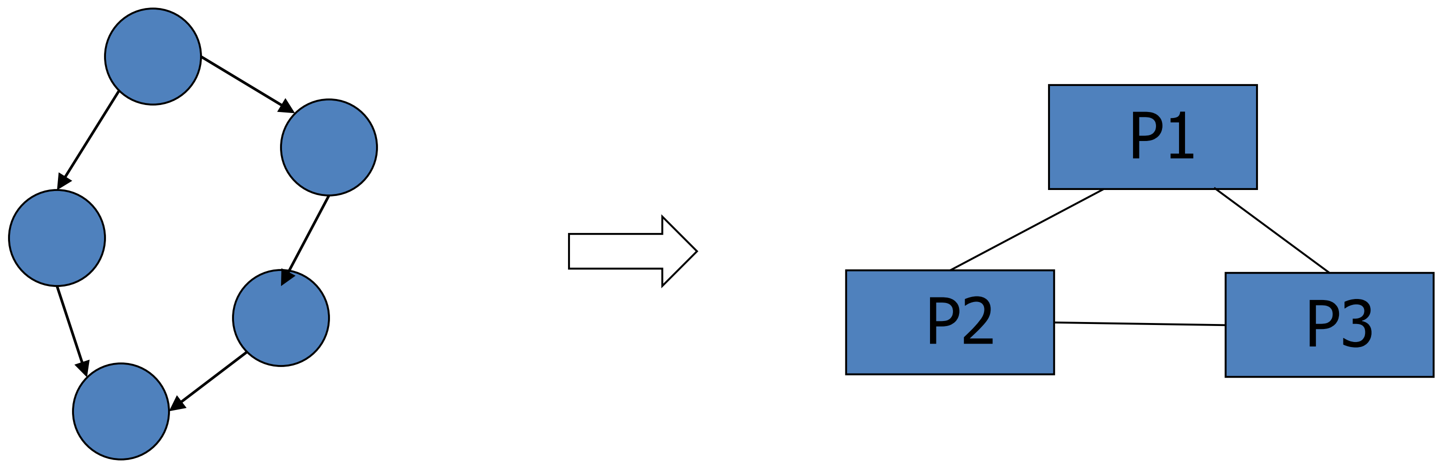



**Given**: *release time, workload of each task*

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

**Objectives**: make-span…

# 3. DAGs Scheduling on Heterogeneous Processors

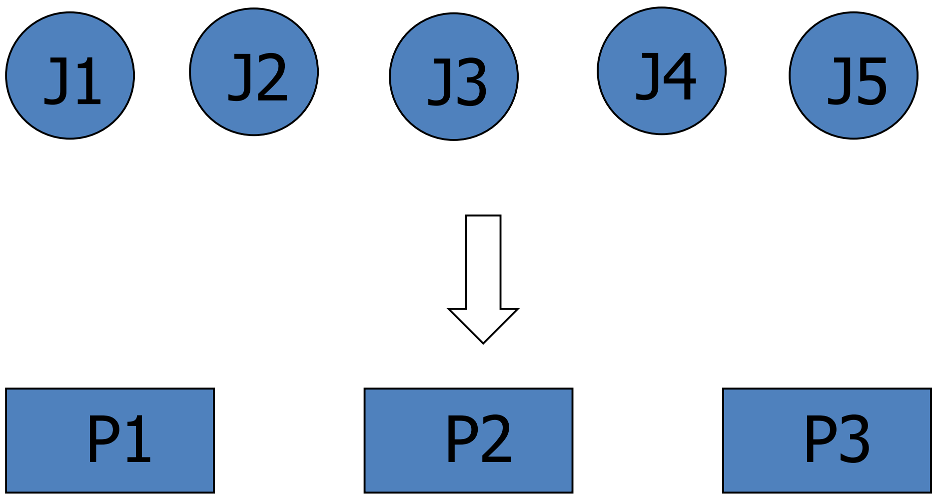



**Given**: *processing time* of every task on every processor, *communication time* on the edges

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

**Objective**: make-span …

# 4. Job Shop Problem (JSP)




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

**Constraint**: every job is executed by every processor exactly once

**Objective**: make-span …

# Similar Terminologies

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

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

# Online Methods

- Machine centric approach
  - Scheduling is triggered when a machine becomes idle
  - For each idle 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, …

# List Scheduling Method

- **Step 1: Task selection**

  Construct a ordered list of tasks by assigning priority to each task, and the select the task in the order of their priority.

- **Step 2: Processor selection**

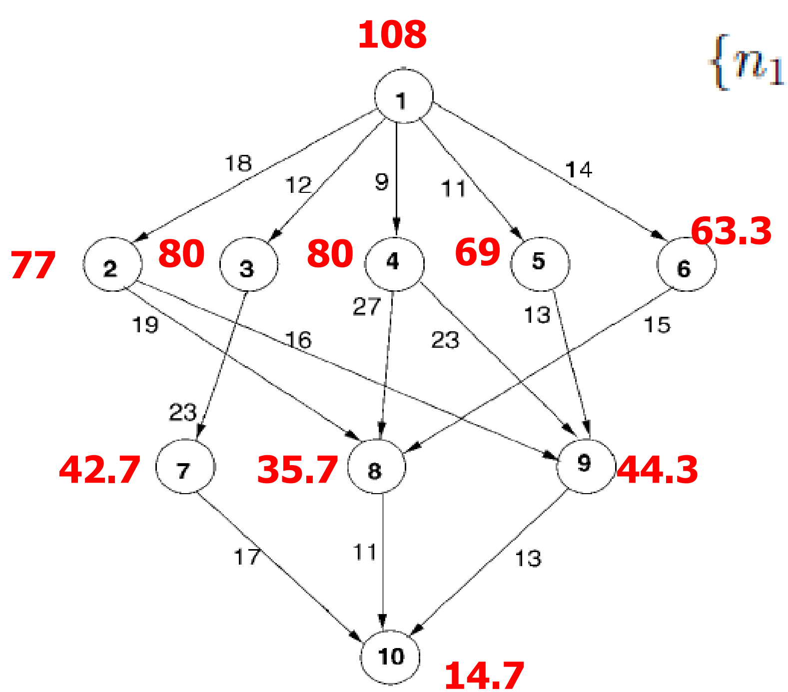
  Each selected task is scheduled to processor which minimizes a predefined cost function.

- **Repeat Step 1& Step 2 until all the tasks are scheduled**

# List Scheduling Method

- Step 1: Task selection – Upward rank

  Upward rank of node *i* is the length of the longest path from node *i* to the exit




$$\{n_1, n_3, n_4, n_2, n_5, n_6, n_9, n_7, n_8, n_{10}\}$$

**Computation Costs**

| Task | P1 | P2 | P3 |
|---|---|---|---|
| 1 | 14 | 16 | 9 |
| 2 | 13 | 19 | 18 |
| 3 | 11 | 13 | 19 |
| 4 | 13 | 8 | 17 |
| 5 | 12 | 13 | 10 |
| 6 | 13 | 16 | 9 |
| 7 | 7 | 15 | 11 |
| 8 | 5 | 11 | 14 |
| 9 | 18 | 12 | 20 |
| 10 | 21 | 7 | 16 |

Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. TPDS'02. (800+ Citations )

# List Scheduling Method

- Step 2: Processor Selection – Earlist Finish Time

$$EST(n_i, p_j) = \max\left\{avail[j], \max_{n_m \in pred(n_i)} \left(AFT(n_m) + c_{m,i}\right)\right\}$$

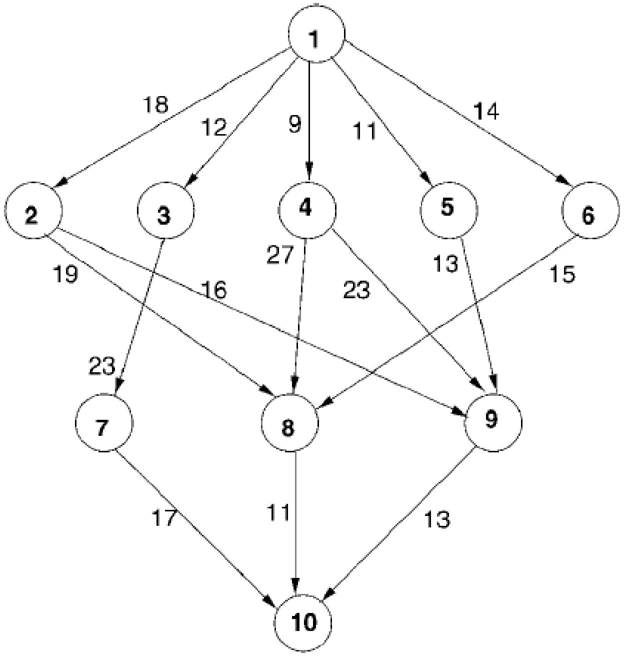
$$EFT(n_i, p_j) = w_{i,j} + EST(n_i, p_j)$$

$$EST(n_{entry}, p_j) = 0.$$

$$AFT(n_i) = \min_{\forall j} EFT(n_i, p_j)$$

**For each task, select the machine which can finish that task in an earliest time.**

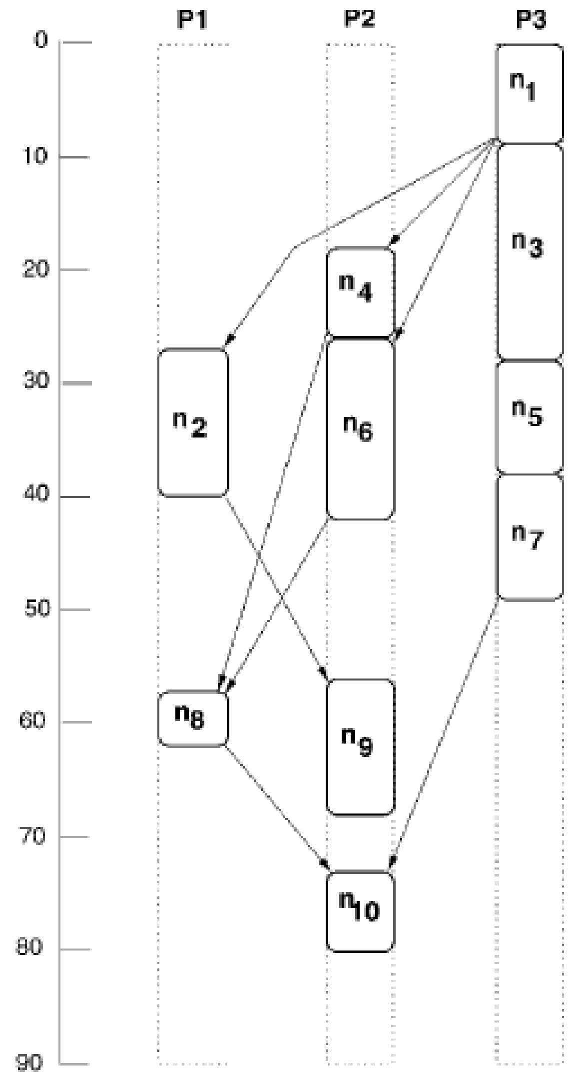
$$\{n_1, n_3, n_4, n_2, n_5, n_6, n_9, n_7, n_8, n_{10}\}$$




| Task | P1 | P2 | P3 |
|---|---|---|---|
| 1 | 14 | 16 | 9 |
| 2 | 13 | 19 | 18 |
| 3 | 11 | 13 | 19 |
| 4 | 13 | 8 | 17 |
| 5 | 12 | 13 | 10 |
| 6 | 13 | 16 | 9 |
| 7 | 7 | 15 | 11 |
| 8 | 5 | 11 | 14 |
| 9 | 18 | 12 | 20 |
| 10 | 21 | 7 | 16 |

# EFT Table

| Task | P1 | P2 | P3 |
|---|---|---|---|
| 1 | 14 | 16 | 9 |
| 3 | 32 | 34 | 28 |
| 4 | 31 | 26 | 45 |
| 2 | 40 | 43 | 46 |
| 5 | 52 | 39 | 38 |
| 6 | … | … | .. |
| … | | | |
| | | | |
| | | | |
| | | | |



# 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.