Software Architecture

Architecture Patterns - 2

BITS Pilani

Vijayarajan

Contents

- Recap
- Architecture Patterns - 02

Assignment & Quiz 1

A1 - Build an Architecture for an App

- The App should at minimum include the technologies Web, Mobile, IOT, cloud and analytics.
- Each team will select an application and get the approval of the TA.
- Duplication May not be allowed …
- The submitted architecture will be evaluated by the TA
- The team will be evaluated for their developed architecture

A2 – Research paper on real life architecture / latest trends etc

- Each team has to select a topic and get the approval of the faculty.
- Duplication may not be allowed.
- Final Paper should be submitted as per the agreed upon template and schedule

Quiz 1

In the coming week. It will be available for 3 to 5 days. Wait for announcement in Taxila

Evolution of SW Architecture

We design and implement information systems to solve problems and process data.

As problems become larger and more complex and data becomes more voluminous, so do the associated information systems

- Structured programming, Data Structure, Higher Level languages, software engineering, Object Oriented etc

Computing become Distributed, on the cloud, Mobile as a front end

As the problem size and complexity increase, algorithms and data structures become less important than getting the right structure for the information system.

Specifying the right structure of the information system becomes a critical design problem itself

< Example from Construction Industry>

Importance of Quality attributes & Tactics

- Functional requirements help us to define the modules
- Quality attributes help us to **structure** the system

Availability

Modifiability

Performance

Security

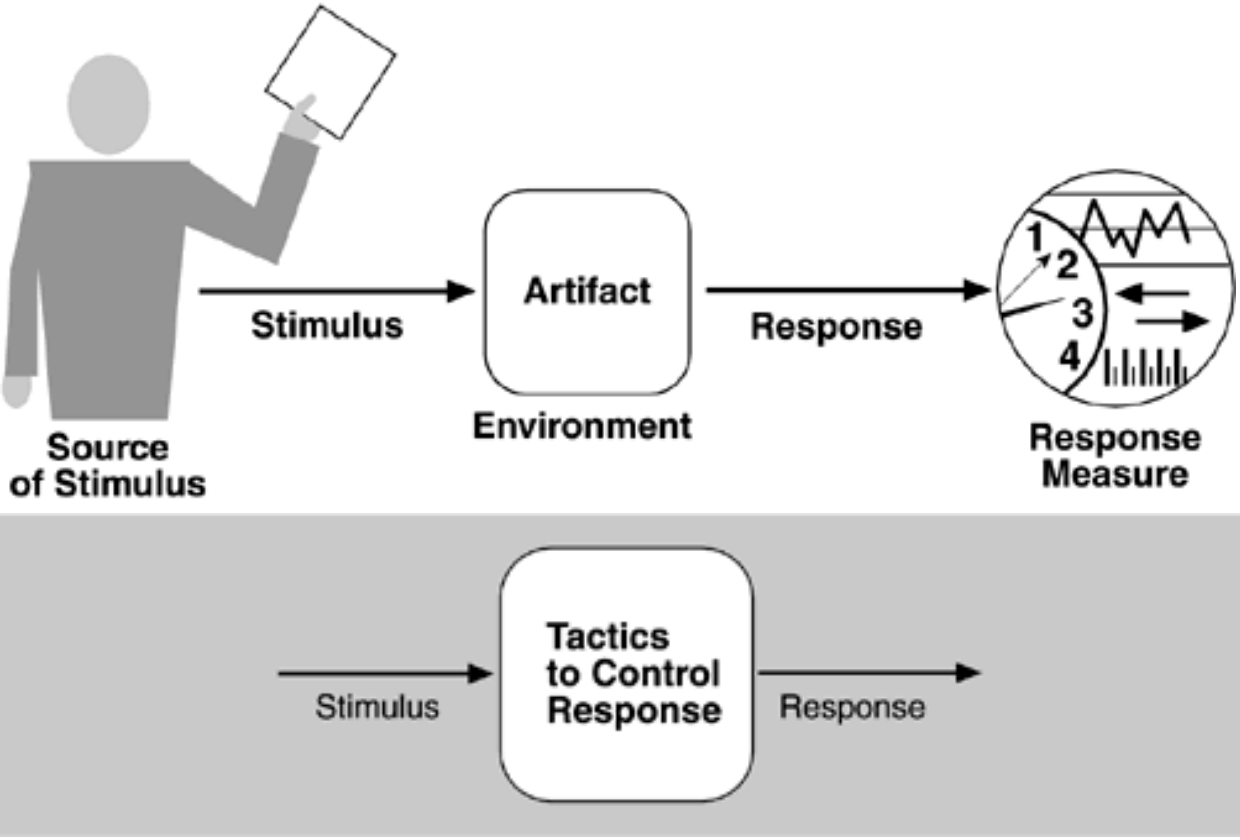
Usability

Interoperability

Scalability

Testability

Architecturally significant requirements



What is a Pattern

1. Addresses a recurring design problem
2. Documents existing, well proven design experience
3. Pattern identify and specify abstractions that are above the level of single classes and instances or of components

   Typically, a pattern describes several components, classes or objects, and details their **responsibilities** and relationships, as well as their **cooperation**.

   All components together solve the problem more effectively that the pattern addresses

Because patterns are (by definition) found repeatedly in practice, one does not invent them; one discovers them.

List of patterns

1. Layer
2. Pipe & Filter
3. MVC
4. Publish & Subscribe
5. Client & Server
6. P2P
7. Shared Data
8. Broker
9. Map-Reduce
10. Multi-tier
11. SOA

Service Oriented Architecture

Started as a means to integrate disparate applications of vendors & partners

SoA consists of:

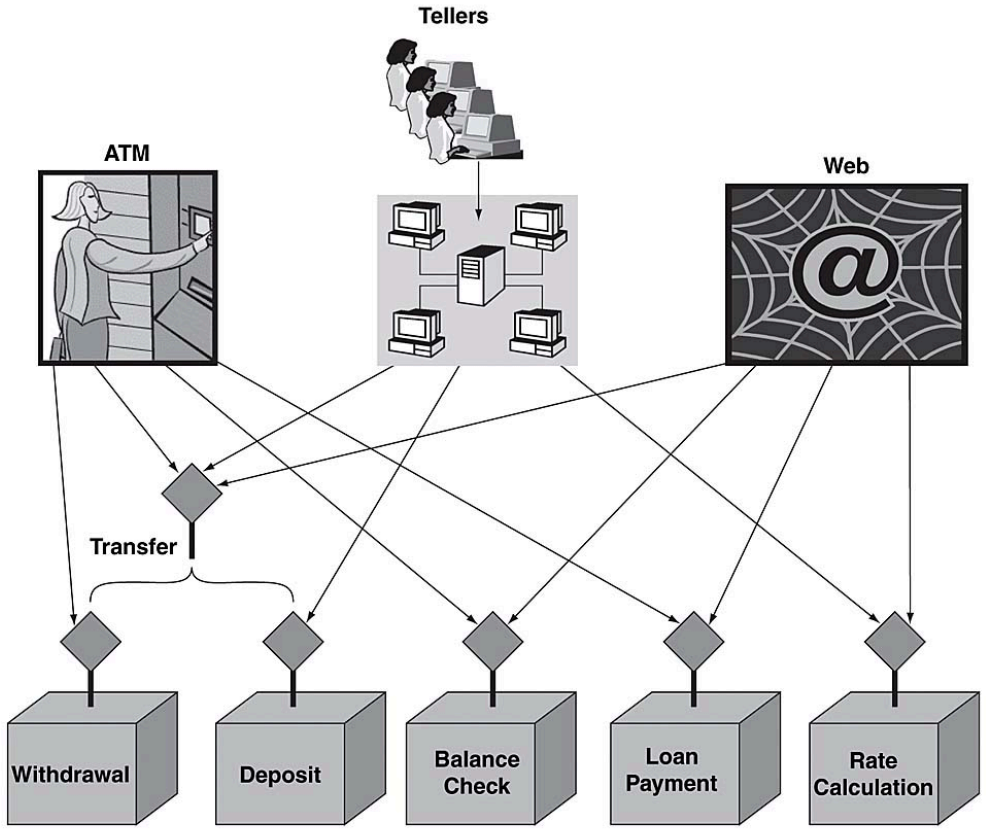
- Business services deployed in a network
- Application components that make use these services
- Infrastructure to define services and communicate with services

Introduction

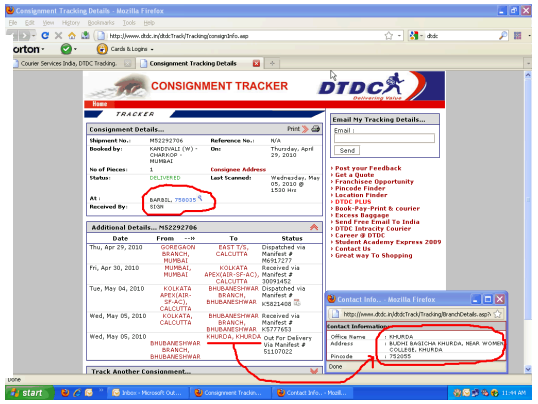
Systems that are built to change are more valuable than systems that are built to last.

In reality, systems that are built to change are the only ones that last.

Banking Example –Process & Tech View



Another Example



When a courier company makes the tracking of shipments visible to its customers, increasing customer satisfaction and reducing the costly overhead of status enquiries

Towards – SOA

If applications are built using reusable components as building blocks, those applications are likely to be more flexible to change and will last longer

Services as reusable components

- Companies implemented the service concept at the business level long before they used it in software.
- Many business functions in a modern enterprise are organized in the form of services

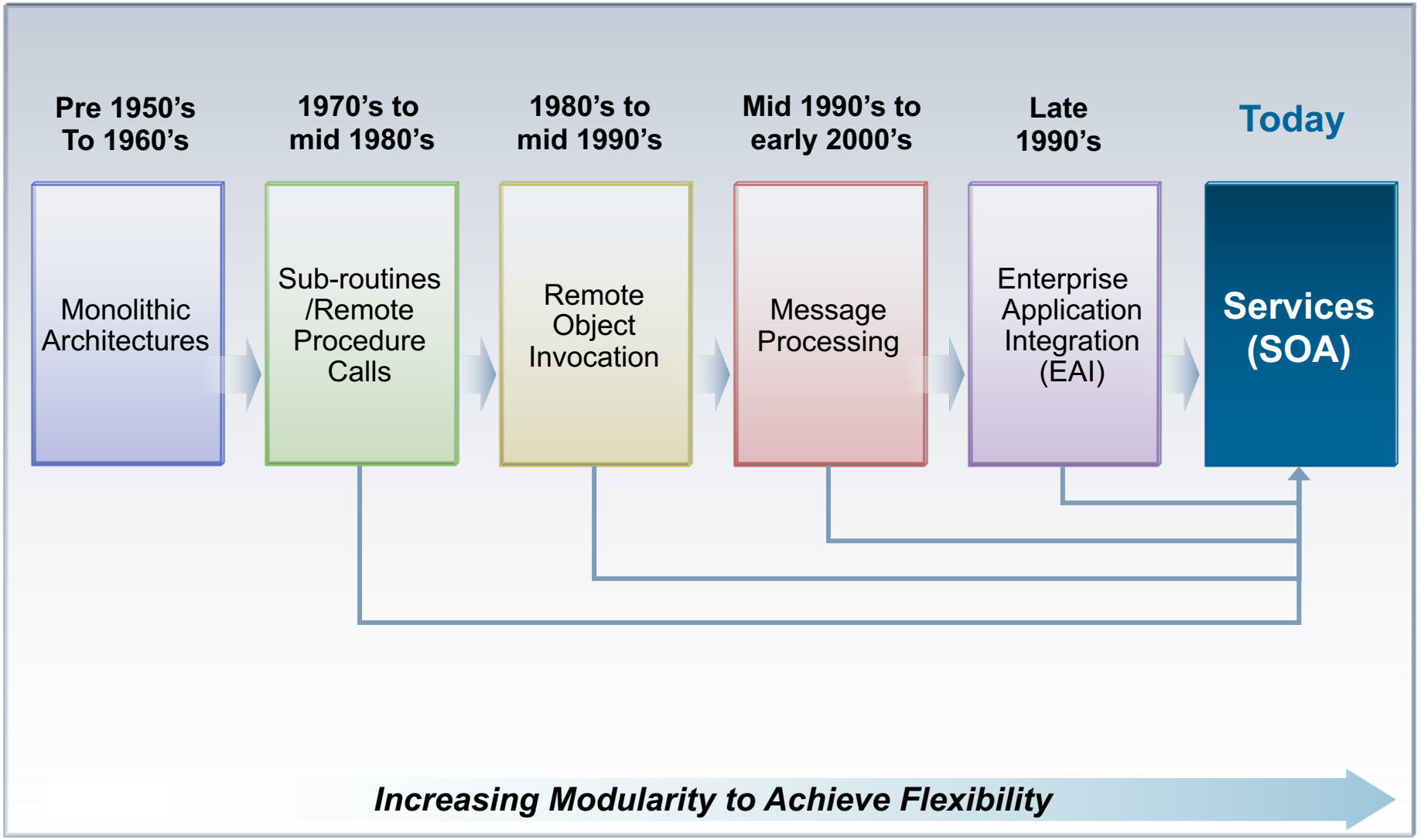
Services as reusable components Service oriented Architecture

Definition of Service Oriented Architecture

- Service-Oriented Architecture (SOA) is a way of designing, developing, deploying, and managing systems, in which
  - Services are *reusable components* that represent business or operational tasks,

- **Reusable** is a key element of this definition because it is what enables the creation of new business and operational processes based on these services

IT's Architectural Evolution: Making IT More Responsive



SOA - Introduction

Service-oriented architecture (SOA) is an architectural style for designing and developing distributed systems

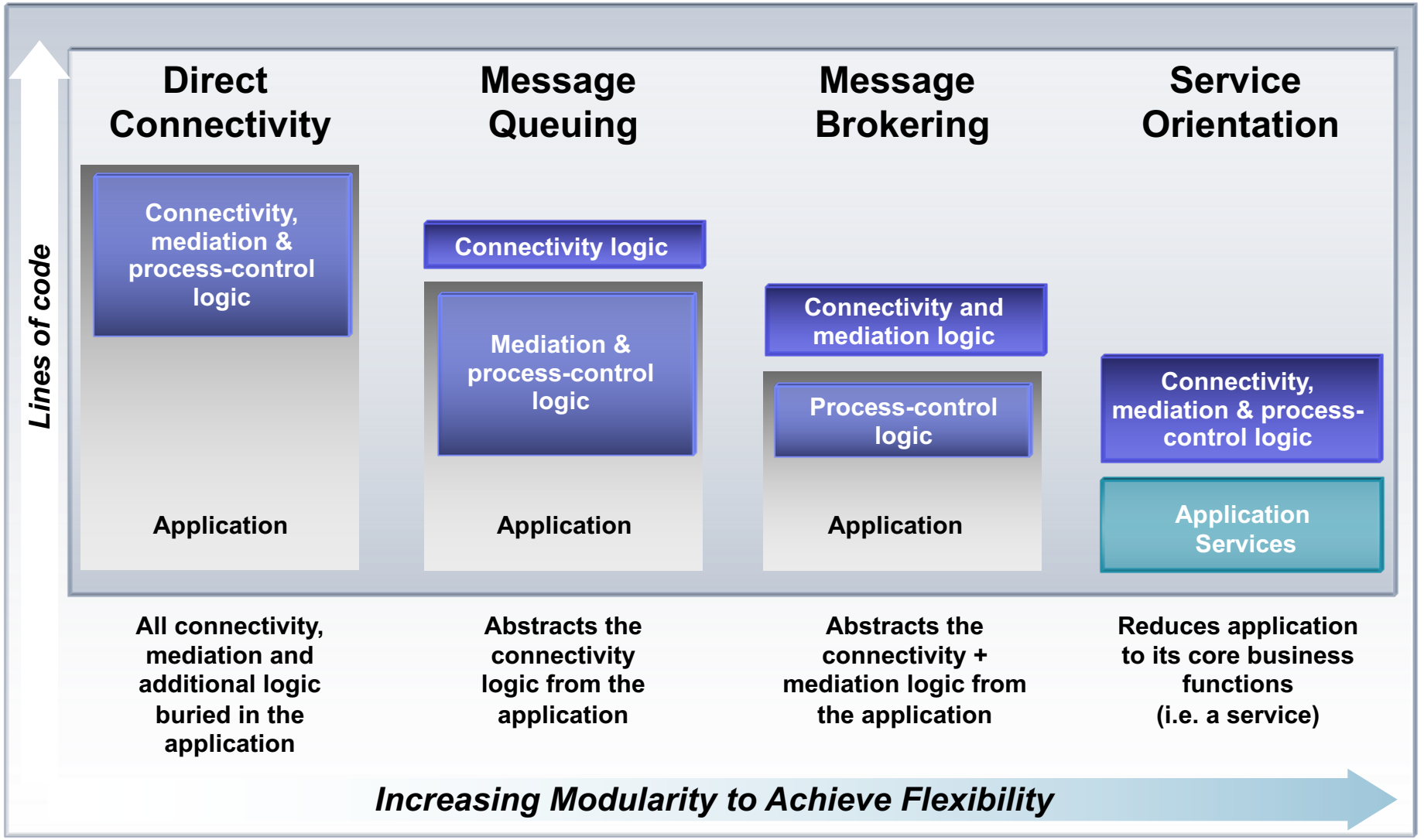
Business applications have evolved over a period of time from a relatively rigid monolithic architecture to an extremely flexible, distributed one

Data are Distributed

Computation is Distributed

Users are distributed

SOA: The Next Step on the Connectivity Evolution



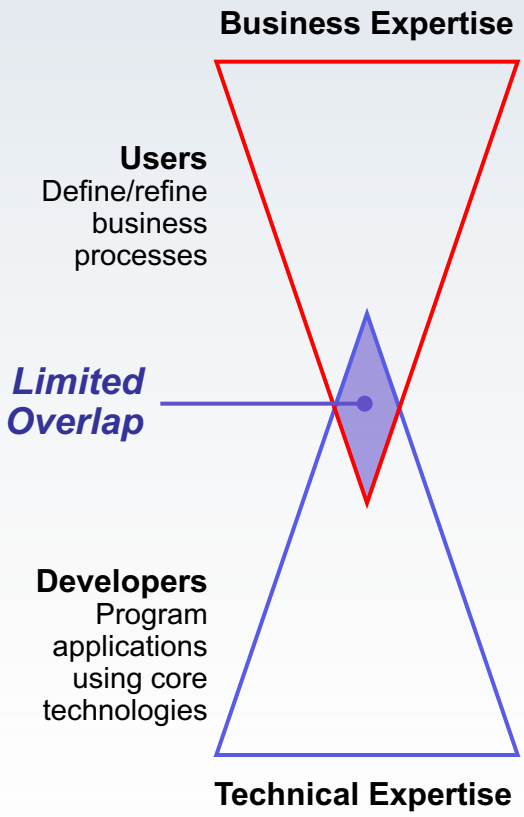
SOA

- A method of design, deployment, and management of applications where:
  - All software is organized into business services that are network accessible and executable.
  - Service interfaces are based on public standards for interoperability.
- Services become building blocks that form business flows
- Services can be reused by other applications

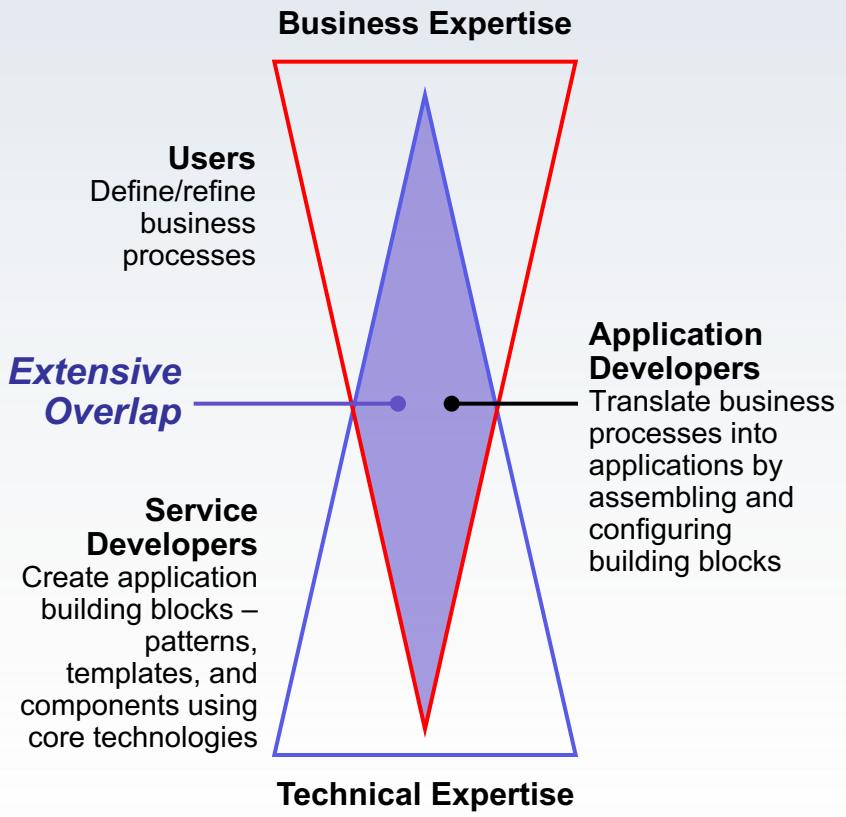
A New Programming Model

Supporting the SOA Abstraction Layering

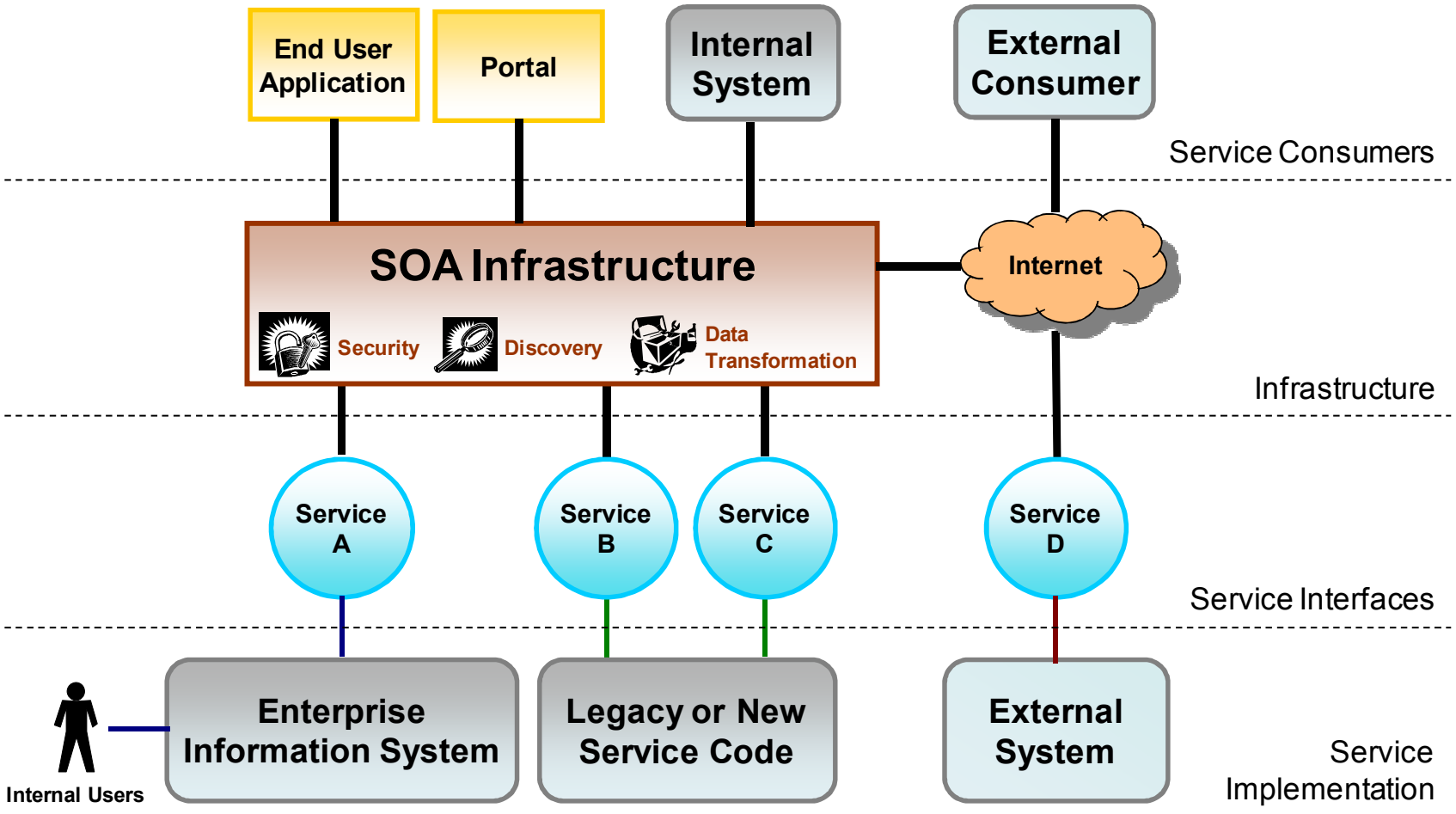
Traditional Software Development



Service-Oriented Development



Major elements of SOA



Service

- A service is a reusable component that can be used as a building block to form larger, more complex business-application functionality.
- A service may be as **simple** as “get me some person data,” or as **complex** as “process a disbursement.”
- A service provides a discrete business function that operates on data. Its job is to ensure that the business functionality is applied consistently, returns predictable results, and operates within the quality of service required.

WHAT CAN SERVICES DO?

- Perform business logic • Transform data
- Route messages
- Query databases
- Apply business policy
- Handle business exceptions
- Prepare information for use by a user interface
- Orchestrate conversations between multiple services

Service Implementation / service interface

- Service Implementation is the actual code that implements the capabilities
- ERP Systems, Custom developed code, Legacy systems
- Service interface: Services are accessed via standardized service interfaces that hide the implementation details of the service from consumers. The Service provider hosts a service implementation

Adapters

- Adapters make SOA possible.
- SOA is about being able to reuse the business applications that you already have. In order to do that, you need to add interfaces to these applications that allow you to directly invoke
- The SOA adapters provide these interfaces.

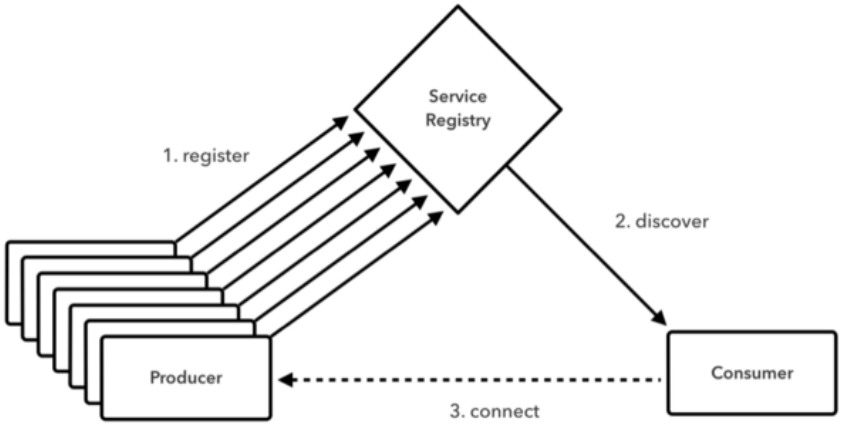
Service Consumers

- Service consumers are the clients for the functionality provided by the services. Examples of service consumers are end-users, internal systems, external systems, and composite services

Example:

- Tax Payers filing tax returns end users
- Stock analysis program accessing the “real time stock price” service
- Adding “cash on delivery service” to an existing e-commerce platform

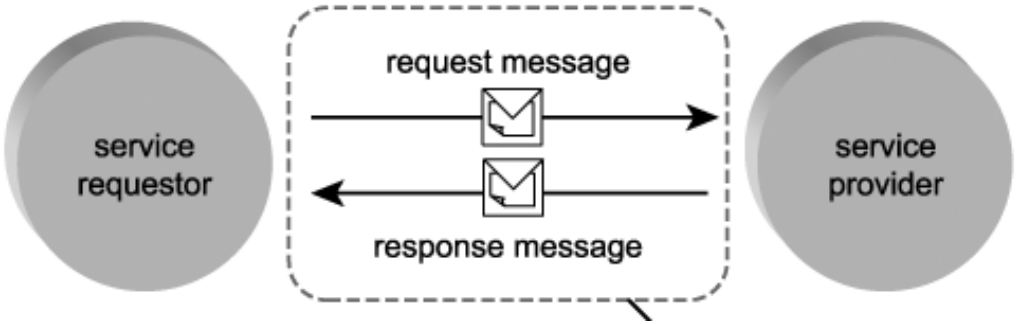
Service Registry, Discovery, Broker



Service discovery is the mechanism by which service consumers become aware of available services and their capabilities; Discovered at runtime
Service providers publish their services in some form of service registry
Service broker is the component that actually makes all the connections between components work.

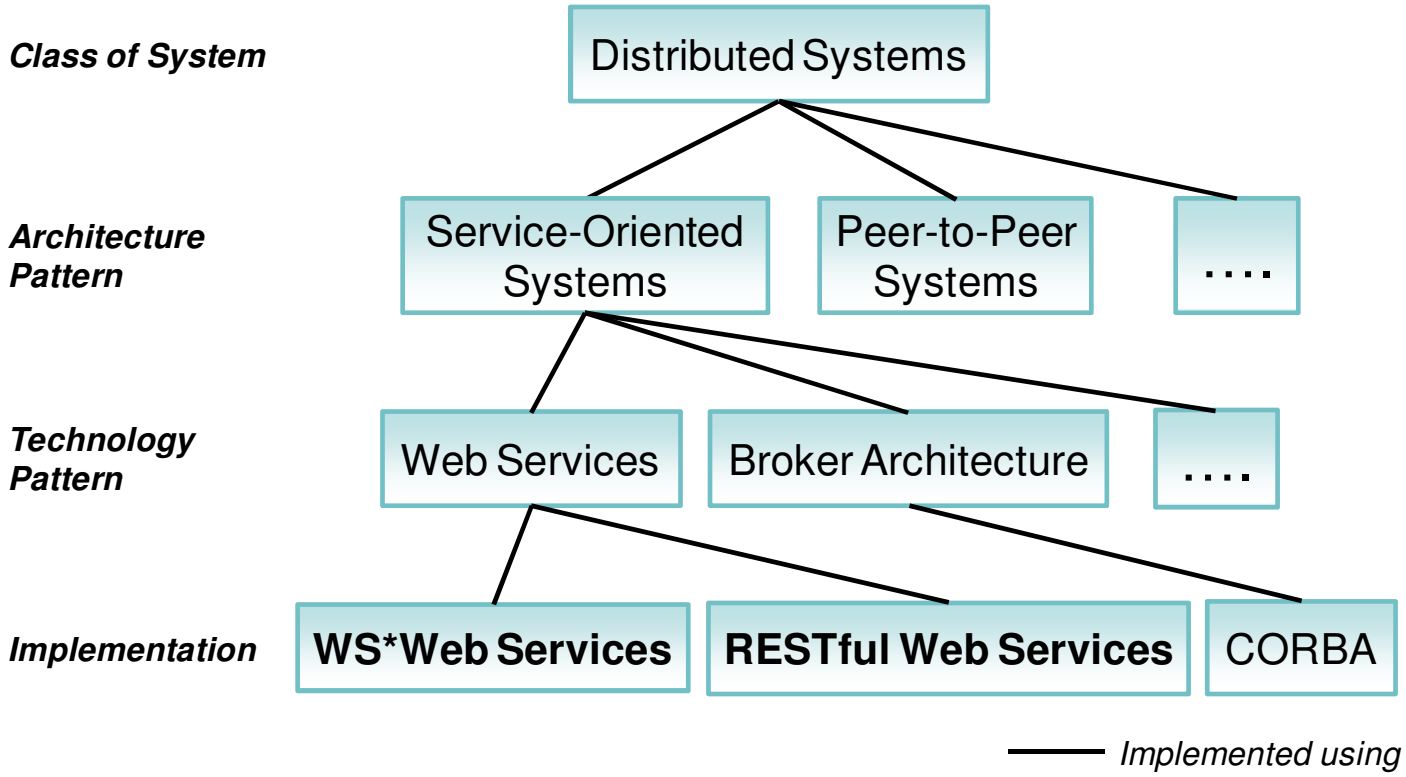
Benefits – choose the right service – functionality, performance, and cost
Easier introduction of system upgrades – an upgraded service can be made available for selection in parallel with the one that it replaces, which can then be withdrawn

Service Invocation



- Services normally invoke each other by exchanging messages, even where they are executing on the same processor
- enables loose coupling
  - services can be moved (execution location)
  - Services can be replaced by better one
  - Improves configuration flexibility
- Message Monitoring
- Message control
- Message transformation
- Message Security

Web Service



Web Services is one technology pattern for implementing service-oriented systems, Because it is the most common technology pattern, Web Services is often equated to SOA.

Few Technologies…

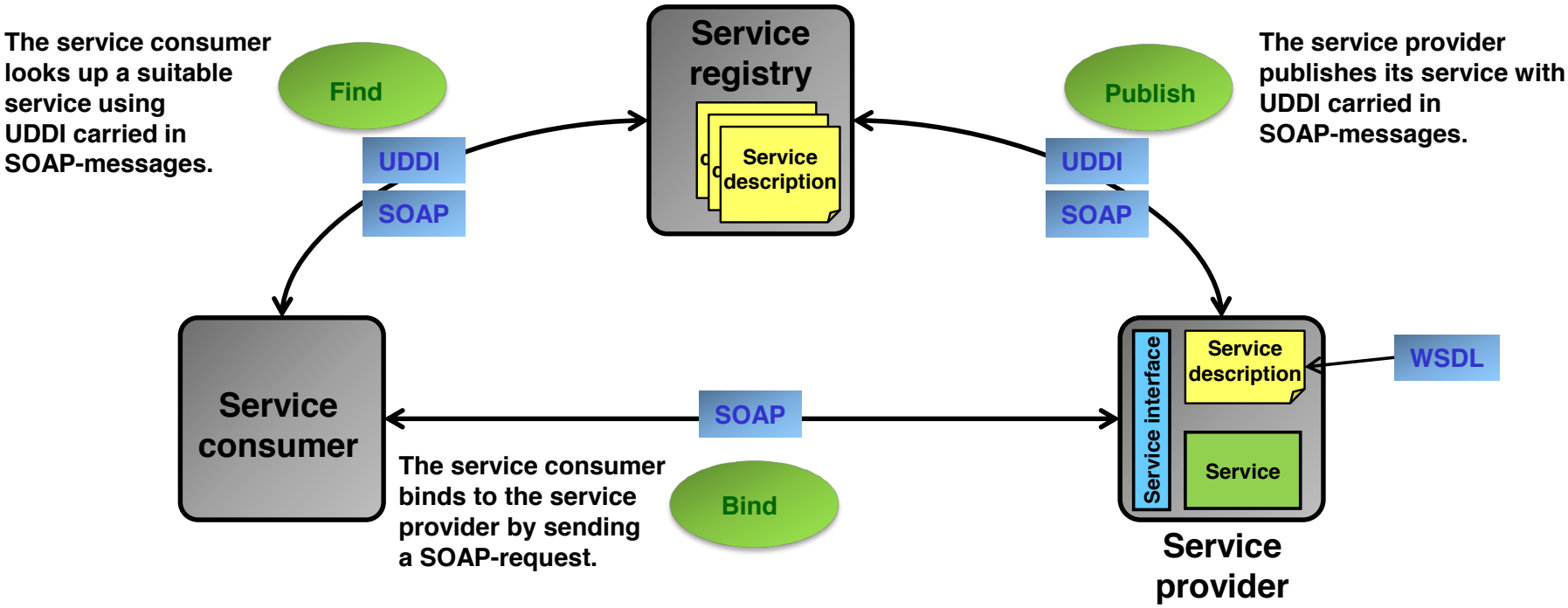
XML	eXtensible Markup Language	The definition language that accompanies information. It tells the software what that information actually is
SOAP	Simple Object Access Protocol	Uses XML to describe messages that are sent from one program / service to another.
WSDL	Web Services Description Language	XML document that describes a Web service - methods (services) available; The parameters and data types; the location of the Service and how to access it
UDDI	Universal Description, Discovery, and Integration	Registry of Web Services; (Yellow Pages, Searchable), Updated by Web Services providers; Information Stored as WSDL UDDI itself is a Service, Accessed using SOAP Consumer gets the Info on Services through UDDI

Data and Control Flow

2. Web service architecture

The combo SOAP+WSDL+UDDI defines a general model for a web service architecture.

SOAP:	**Simple Object Access Protocol**
WSDL:	**Web Service Description Language**
UDDI:	**Universal Description and Discovery Protocol**
Service consumer:	**User of a service**
Service provider:	**Entity that implements a service (=server)**
Service registry:	**Central place where available services are listed and advertised for lookup**



Service Oriented Architecture

Different Things to Different People

	Roles	
Capabilities that a business wants to expose as a ***set of services*** to clients and partner organizations	**Business**	
An ***architectural style*** that requires a service provider, requestor and a service description. It addresses characteristics such as loose coupling, reuse and simple and composite implementations	**Architecture**	
A ***programming model*** complete with standards, tools, methods and technologies such as Web services	**Implementation**	
A ***set of agreements*** among service requestors and service providers that specify the quality of service and identify key business and IT metrics	**Operations**	

Enterprise Service Bus

- An *enterprise service bus* (ESB) is the infrastructure that enables high interoperability between distributed systems for services. It makes it easier to distribute business processes over multiple systems using different platforms and technologies.

ESB - Services

- **Messaging** : provide intelligent content-based routing, and guarantee delivery.
- Management : Monitor their own performance, helping to
- enforce SLA (latency).
- **Interface** : Can validate messages against schema and provide application adapters
- **Mediation** : Transform messages between the formats
- **Metadata** : transform data from one format to another by using metadata definitions.
- Security : Encrypt messages where needed; authorize, authenticate, and audit all ESB activity.

No SOA

- In a solution that does not require the integration of components or systems running on different platforms, or implemented using different technologies, service orientation may be overkill because there is an overhead for the use of SOA technologies to provide interoperability across platforms.

No SOA

- Hard real-time systems are clearly not a good match for service orientation. Strict timeliness requirements conflict with several aspects of common technologies used in service-oriented systems (e.g., web services) that may introduce unbounded overhead in processing
- Embedded systems are not naturally fit to host service-oriented systems. Embedded platforms have limited computing power, memory, and disk resources. Many SOA technologies are heavyweight in terms of memory and CPU requirements. Thus, designing a SOA solution for the software on a washing machine or a video-game console can bring unneeded complication and overhead.

SOA - additional Points

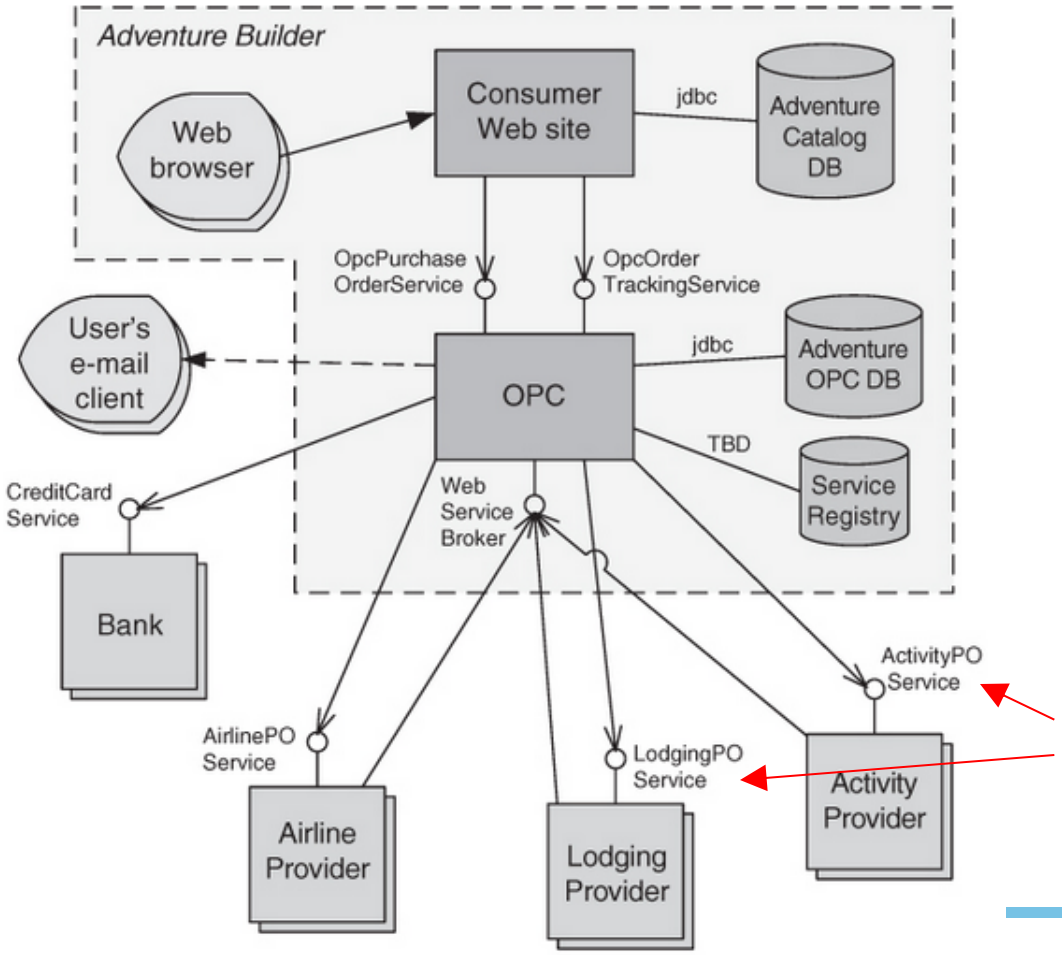
- Loose coupling: The consumer of the service is required to provide only the stated data on the interface definition, and to expect only the specified results on the interface definition. The service is capable of handling all processing (including exception processing).
- Stateless: The service does not maintain state between invocations. It takes the parameters provided, performs the defined function, and returns the expected result. If a transaction is involved, the transaction is committed, and the data is saved to the database.
- Location agnostic: Users of the service do not need to worry about the implementation details for accessing the service. The SOA infrastructure will provide standardized access mechanisms

SOA

- Service-oriented architecture (SOA) is an architectural style for designing and developing distributed systems.
- From a quality attribute point of view, the primary drivers for service orientation adoption are interoperability and modifiability
- the top drivers for SOA adoption were mainly **internally** focused: these top drivers generally included application integration, data integration, and internal process improvement.

SoA : Travel company

Travel company's application makes use of business services offered by partner systems such as Airline, Lodging, Bank, etc.

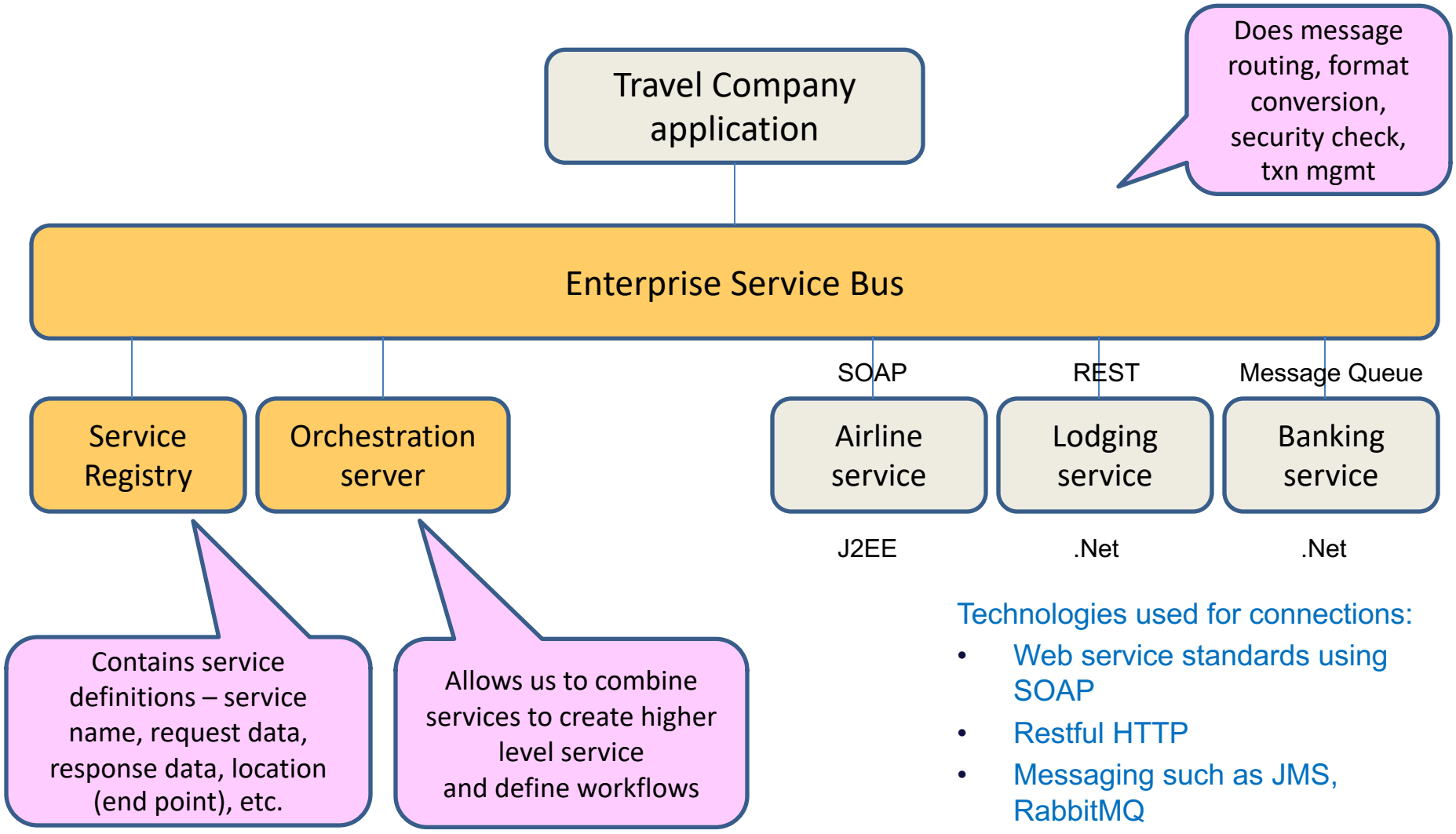


OPC: Order processing centre

Contains many application components

Business services

SoA: Travel company – SoA components



Technologies used for connections:

- Web service standards using SOAP
- Restful HTTP
- Messaging such as JMS, RabbitMQ

Map-Reduce

MAP REDUCE

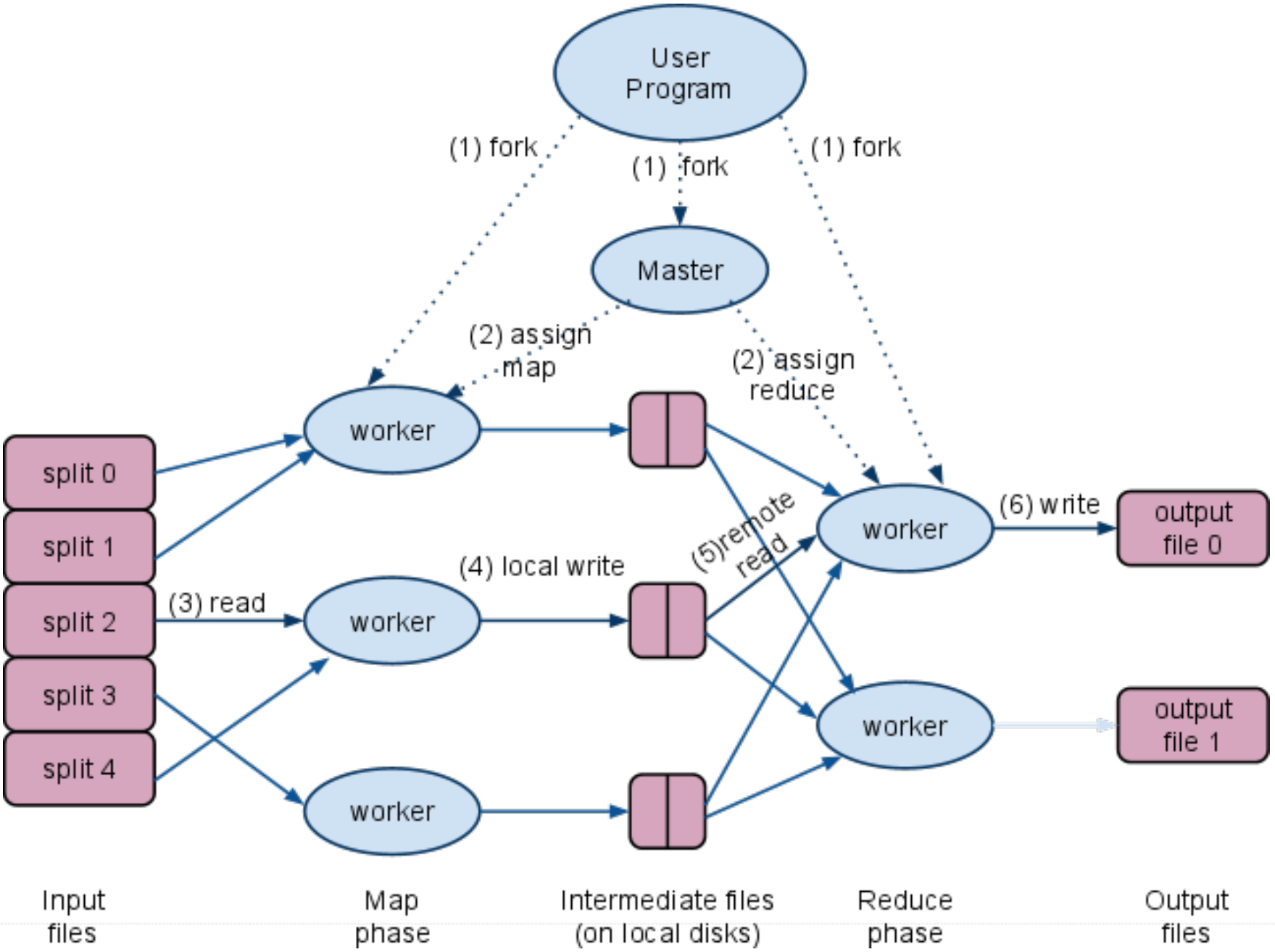
- Invented by Google.
- Is a programming model for processing large datasets
- distributed on a large cluster.
- Uses the concept of Divide and Conquer.
- Two methods: map() and Reduce() .
- Map() sorting and filtering.
- Reduce() counting and produce Result.

Map-Reduce Pattern

- **Context:** Businesses have a pressing need to quickly analyze enormous volumes of data they generate or access, at petabyte scale.
- **Problem:** The problem the map-reduce pattern solves is to efficiently perform a distributed and parallel sort of a large data set and provide a simple means for the programmer to specify the analysis to be done.

Map-Reduce Pattern

- **Solution:** The map-reduce pattern requires three parts:
  - A specialized **infrastructure** takes care of allocating software to the hardware nodes in a massively parallel computing environment and handles sorting the data as needed.
  - A programmer specified component called the **map** which filters the data to retrieve those items to be combined.
  - A programmer specified component called **reduce** which combines the results of the map


User Program
(1) fork
(1) fork
(1) fork
Master
(2) assign map
(2) assign reduce
worker
split 0
split 1
split 2
split 3
split 4
(3) read
worker
worker
(4) local write
(5)remote read
worker
worker
(6) write
output file 0
output file 1
Input files
Map phase
Intermediate files (on local disks)
Reduce phase
Output files

Map-Reduce Solution - 1

- Overview: The map-reduce pattern provides a framework for analyzing a large distributed set of data that will execute in **parallel**, on a **set of processors**. This parallelization allows for low latency and high availability. The map performs the extract and transform portions of the analysis and the reduce performs the loading of the results.

Map-Reduce Solution - 2

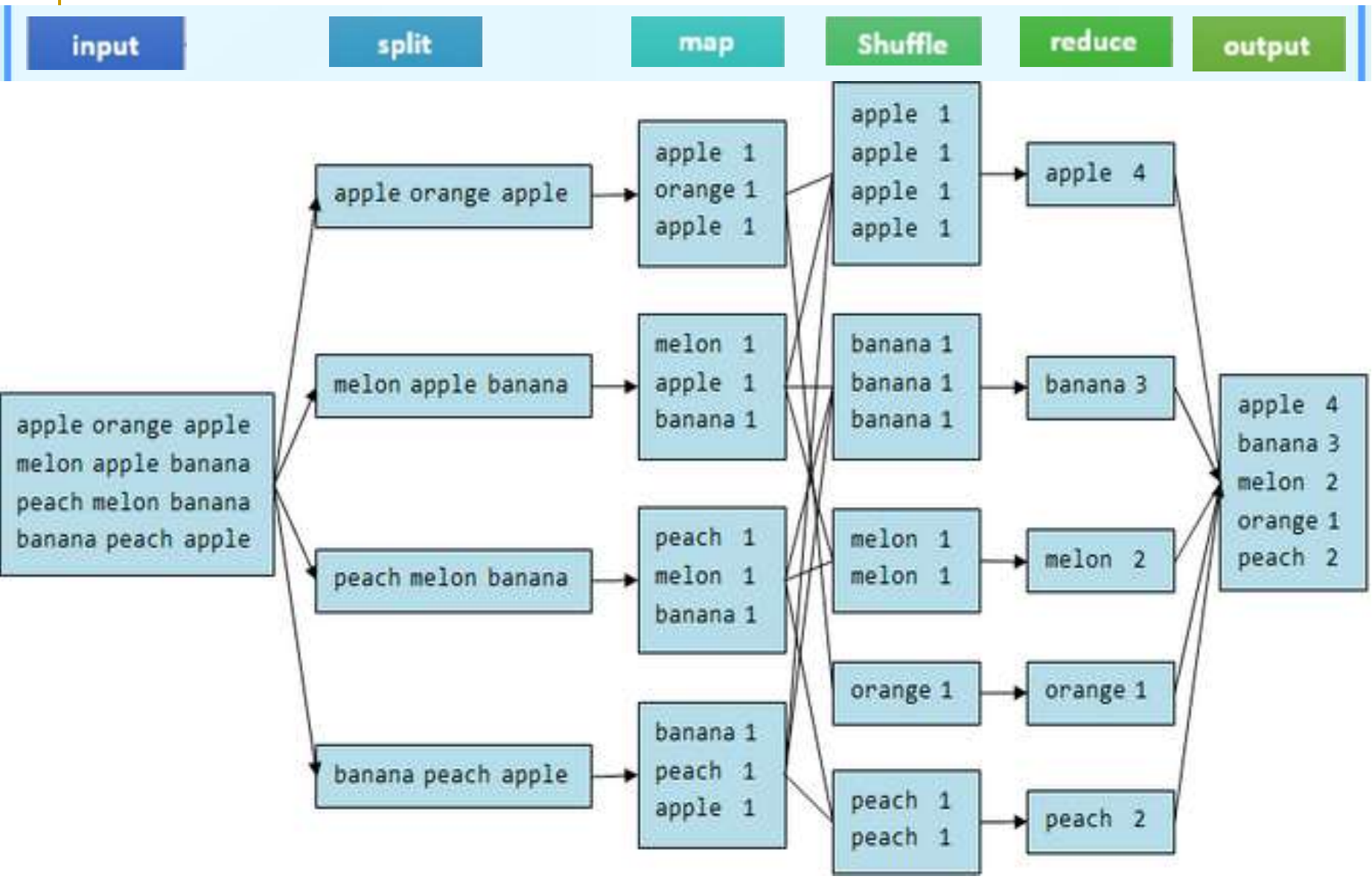
- Elements:
  - Map is a function with multiple instances deployed across multiple processors that performs the extract and transformation portions of the analysis.
  - Reduce is a function that may be deployed as a single instance or as multiple instances across processors to perform the load portion of extract-transform-load.
  - The infrastructure is the framework responsible for deploying map and reduce instances, shepherding the data between them, and detecting and recovering from failure.

Map-Reduce Solution - 3

- Relations:
  - **Deploy** on is the relation between an instance of a map or reduce function and the processor onto which it is installed.
  - **Instantiate, monitor, and control** is the relation between the infrastructure and the instances of map and reduce.

Map-Reduce Solution - 4

- Constraints:
  - The data to be analyzed must exist as a set of files.
  - Map functions are stateless and do not communicate with each other.
  - The only communication between map reduce instances is the data emitted from the map instances as <key, value> pairs.
- Weaknesses:
  - If you do not have large data sets, the overhead of map-reduce is not justified.
  - If you cannot divide your data set into similar sized subsets, the advantages of parallelism are lost.
  - Operations that require multiple reduces are complex to orchestrate.

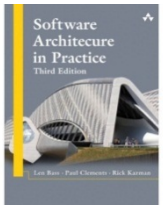

input
split
map
Shuffle
reduce
output
apple orange apple
melon apple banana
peach melon banana
banana peach apple
apple orange apple
melon apple banana
peach melon banana
banana peach apple
apple 1
orange 1
apple 1
melon 1
apple 1
banana 1
peach 1
melon 1
banana 1
banana 1
peach 1
apple 1
apple 1
apple 1
apple 1
apple 1
banana 1
banana 1
banana 1
melon 1
melon 1
orange 1
peach 1
peach 1
apple 4
banana 3
melon 2
orange 1
peach 2
apple 4
banana 3
melon 2
orange 1
peach 2

Map-Reduce – Generic Explanation

- The Map step would run first, do computations in the input key-value pair and generate a new output key-value.
- One must keep in mind that the format of the input key-value pairs does not need to necessarily match the output format pair.
- The Reduce step would assemble all values of the same key, performing other computations over it.
- As a result, this last step would output key-value pairs

Multi-tier pattern

- **Tier** is a physical unit, where the code / process runs. E.g.: client, application server, database server;
- **Layer** is a logical unit, how to organize the code. ... **Layers** are the logical groupings of the software components that make up the application or service.
- All layers may reside on one server or can be distributed across multiple servers
- Also all software components in a layer may run on server or can be distributed across multiple servers
- Tier makes it easier to ensure security & optimize performance

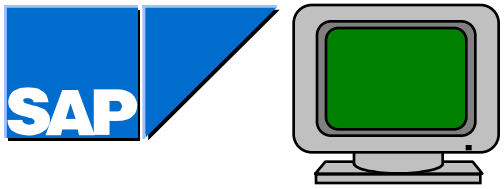


Multi-Tier Pattern

- **Context**: In a distributed deployment, there is often a need to distribute a system's infrastructure into distinct subsets.
- **Problem**: How can we split the system into a number of computationally independent execution structures—groups of software and hardware—connected by some communications media?
- **Solution**: The execution structures of many systems are organized as a set of logical groupings of components. Each grouping is termed a tier.

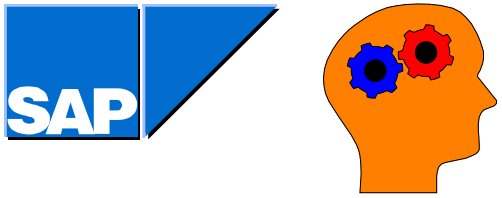
Logical Groupings or "Layers" of SAP R/3 Components

The Presentation Layer



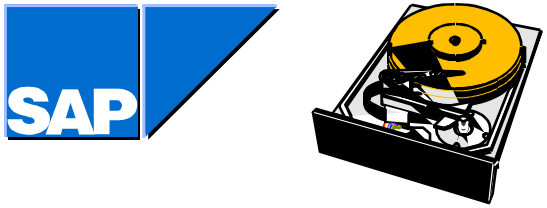
Those SAP R/3 software components that specialise in interacting with end-users form the Presentation Layer.

The Application Layer



Those SAP R/3 software components that specialise in processing business applications form the Application Layer.

The Database Layer

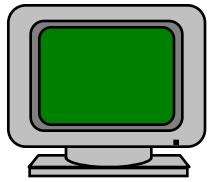


Those SAP R/3 software components that specialise in the management, storage and retrieval of data form the Database Layer

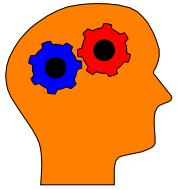
Three Tiered Client-Server Architecture “Logical Layers”

Communication

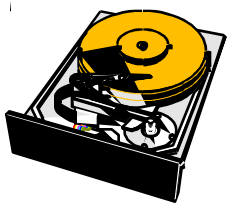
The Presentation Layer
collects user input and
creates process requests.



The Application Layer
uses the application logic of
SAP R/3 programs to collect
and process the process requests.



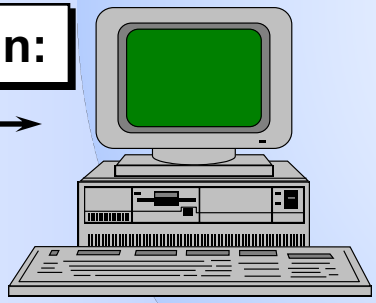
The Database Layer
stores and retrieves all data.



Physical Distribution of R/3'S Logical Layers

Presentation Layer components

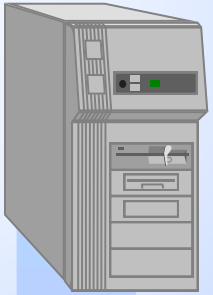
reside in:



Presentation servers:
Systems capable of providing a graphical interface.

Application Layer components

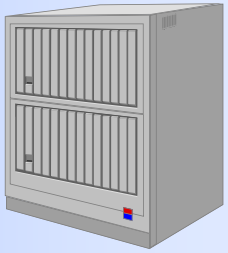
reside in:



Application servers:
Specialised systems multiple CPUs and vast amounts of RAM.

Database Layer components

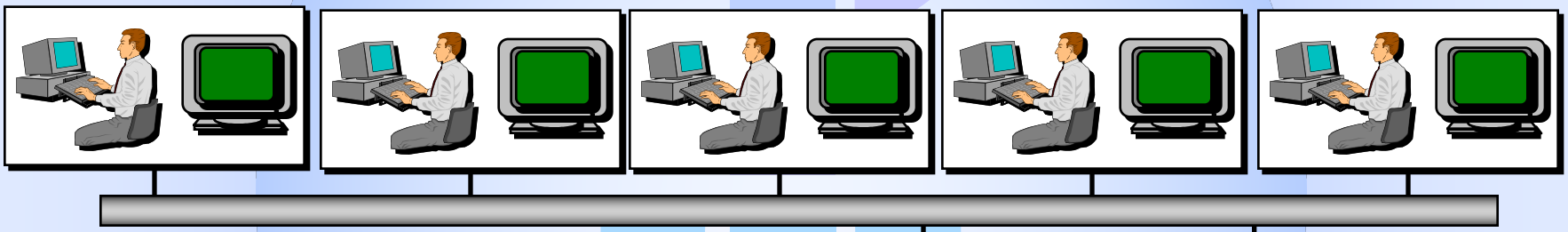
reside in:



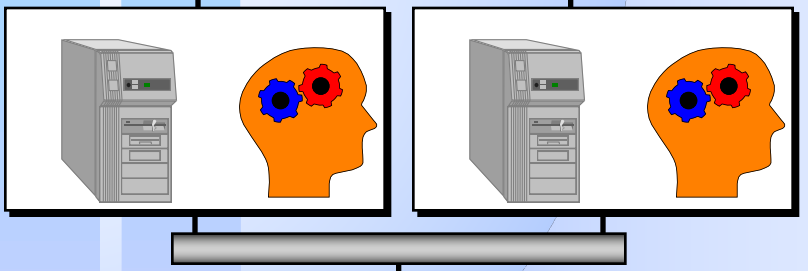
Database servers:
Specialised systems with fast and large hard drives.

Physical Distribution of R/3'S Three Layered Client-Server Architecture

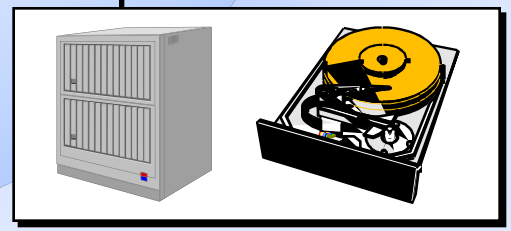
Presentation Layer components are installed across many PCs.

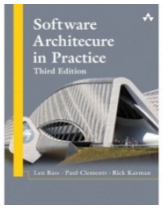


The Application Layer components are installed across one or more high-end servers.



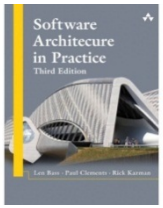
The Database Layer components are installed on one high-end database server.





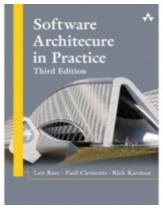
Relationships Between Tactics and Patterns

- Patterns are built from tactics; if a pattern is a molecule, a tactic is an atom.
- MVC, for example utilizes the tactics:
  - Increase semantic coherence
  - Encapsulation
  - Use an intermediary
  - Use run time binding



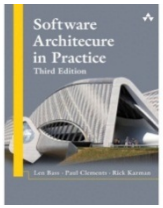
Tactics Augment Patterns

- Patterns solve a specific problem but are neutral or have weaknesses with respect to other qualities.
- Consider the broker pattern
  - May have performance bottlenecks
  - May have a single point of failure
- Using tactics such as
  - Increase resources will help performance
  - Maintain multiple copies will help availability



Tactics and Interactions

- Each tactic has pluses (its reason for being) and minuses – side effects.
- Use of tactics can help alleviate the minuses.
- But nothing is free…

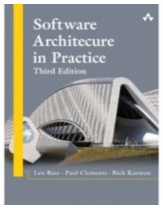


Tactics and Interactions - 2

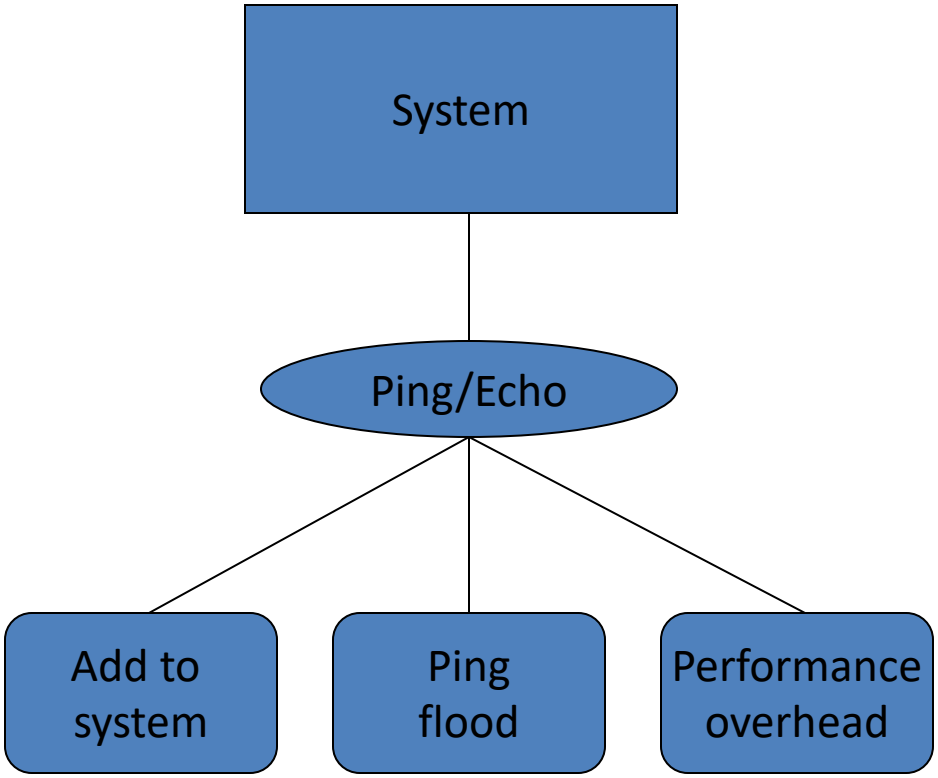
A common tactic for detecting faults is Ping/Echo.

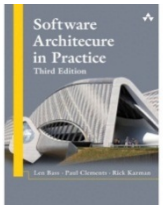
Common side-effects of Ping/Echo are:

- security: how to prevent a ping flood attack?
- performance: how to ensure that the performance overhead of ping/echo is small?
- modifiability: how to add ping/echo to the existing architecture?



Tactics and Interactions - 3



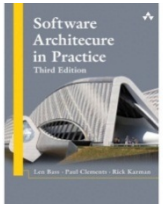


Tactics and Interactions - 4

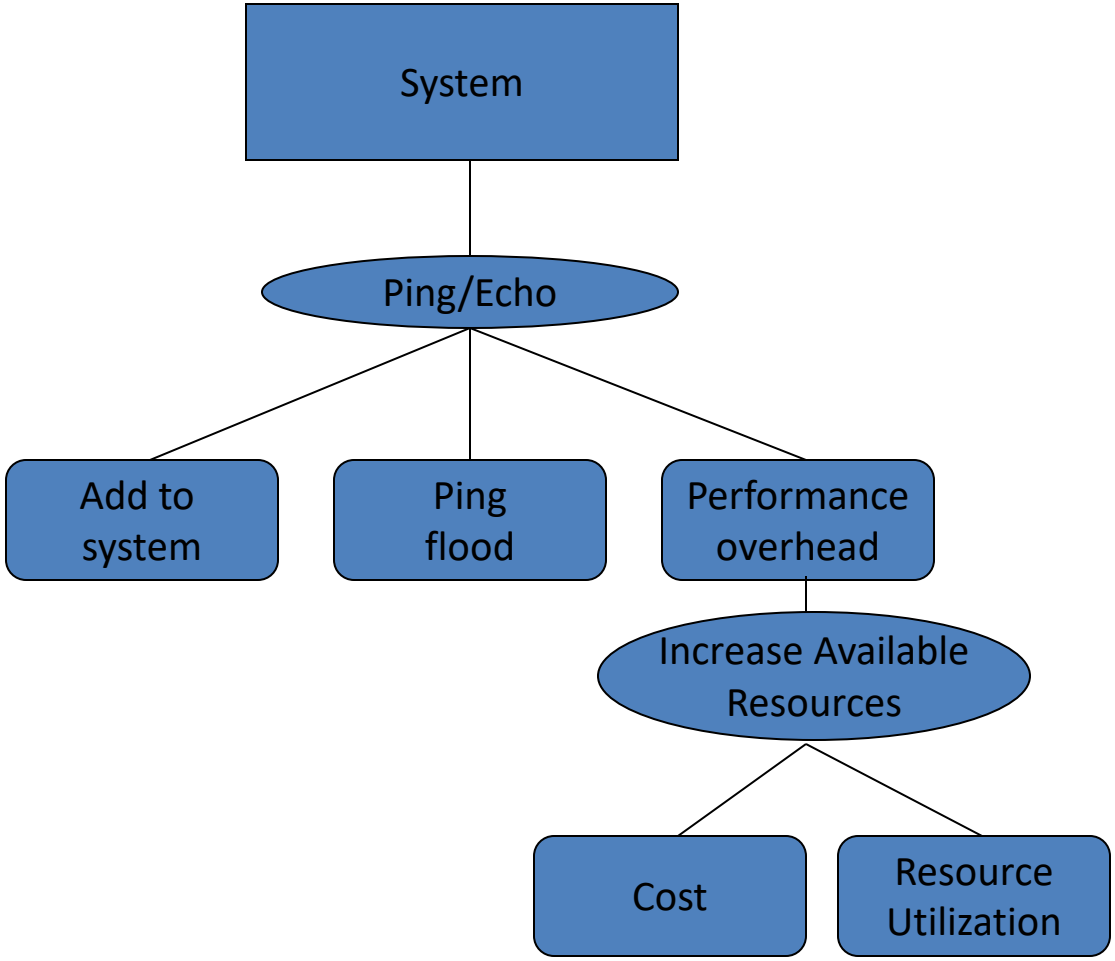
A tactic to address the performance side-effect is “Increase Available Resources”.

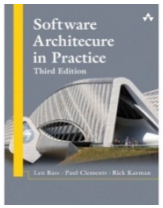
Common side effects of Increase Available Resources are:

- cost: increased resources cost more
- performance: how to utilize the increase resources efficiently?



Tactics and Interactions - 5



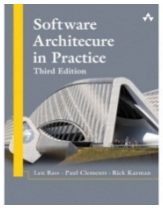


Tactics and Interactions - 6

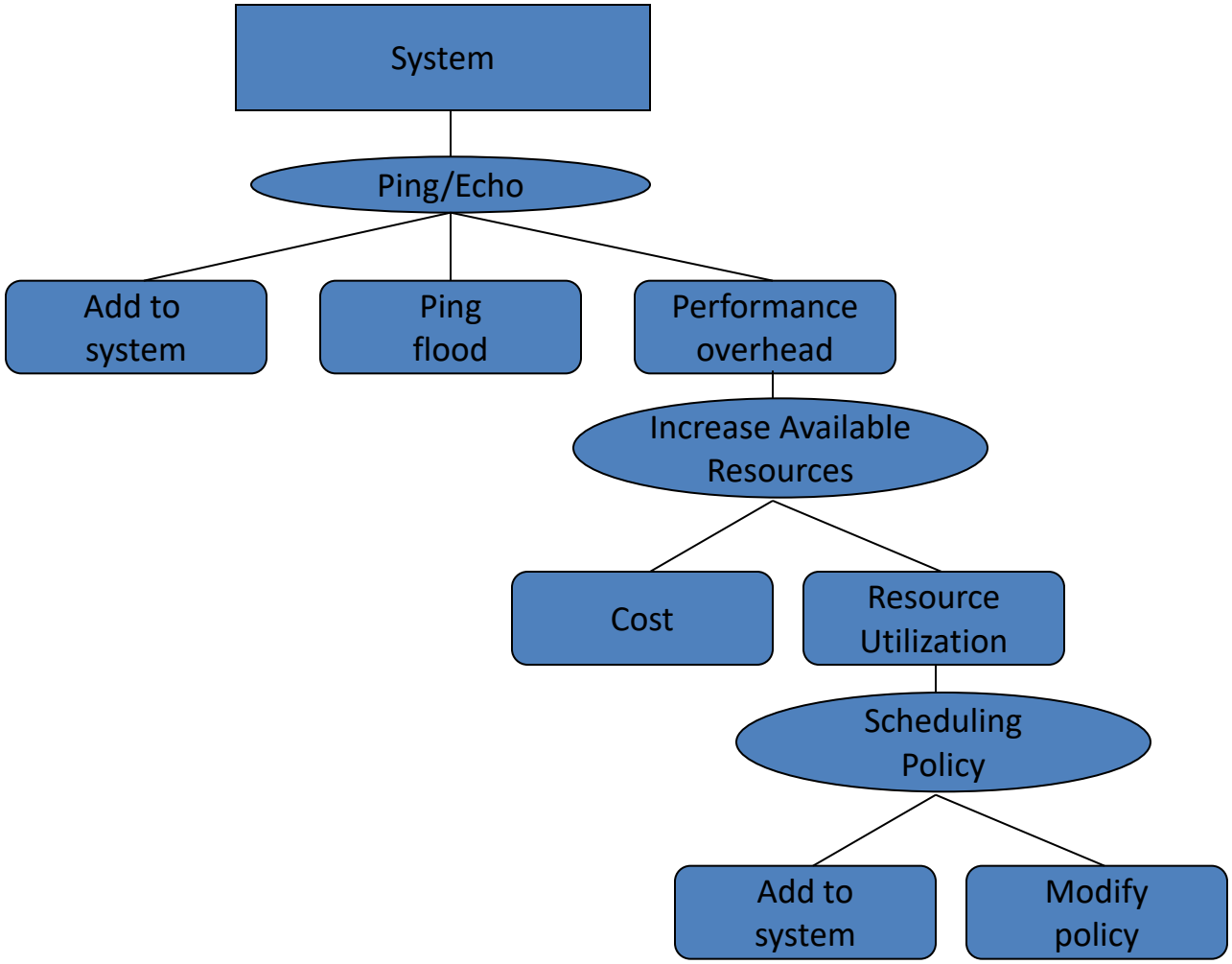
A tactic to address the efficient use of resources side-effect is “Scheduling Policy”.

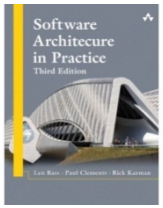
Common side effects of Scheduling Policy are:

- modifiability: how to add the scheduling policy to the existing architecture
- modifiability: how to change the scheduling policy in the future?



Tactics and Interactions - 7



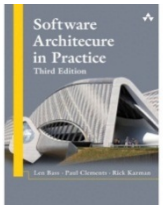


Tactics and Interactions - 8

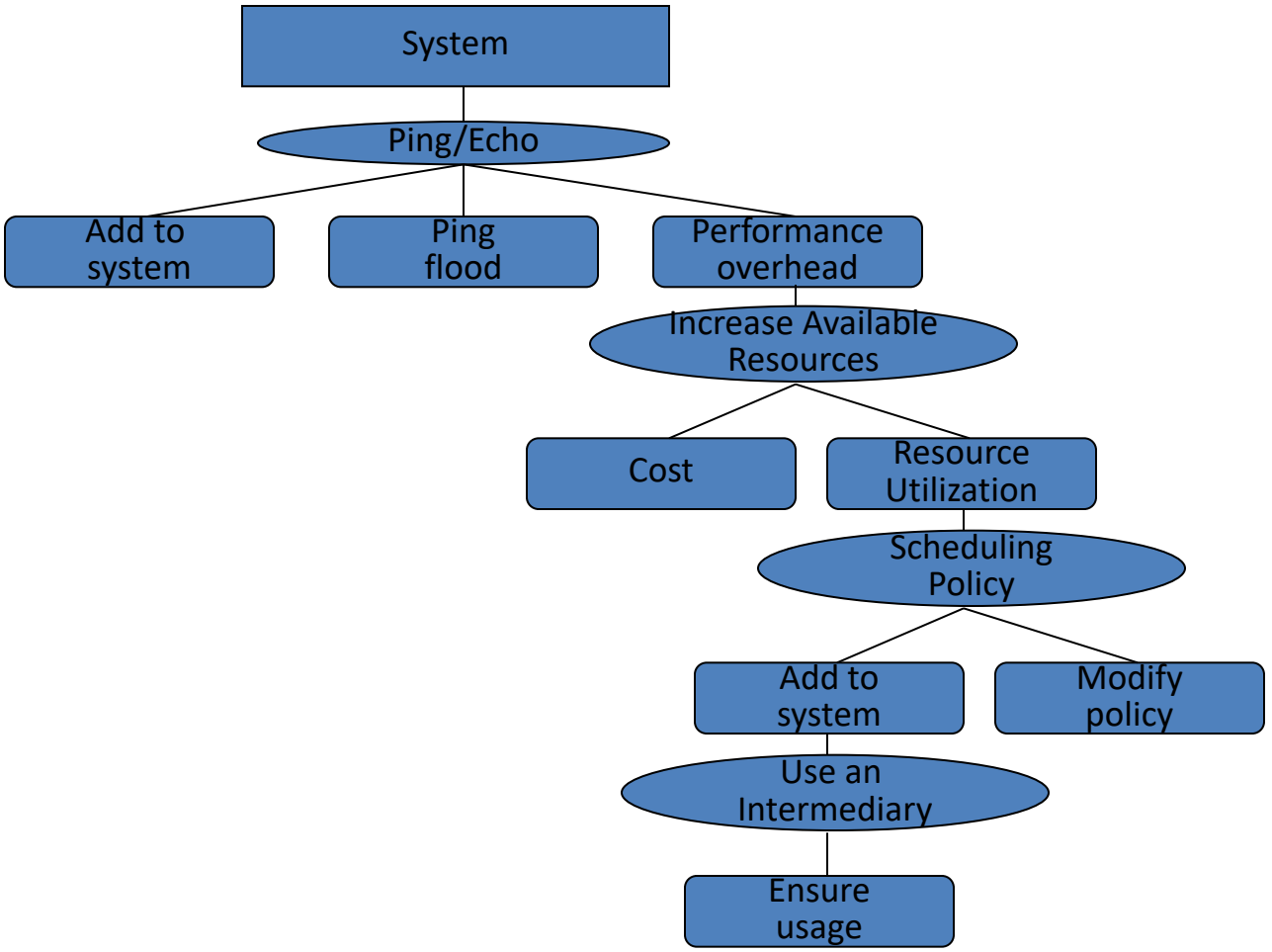
A tactic to address the addition of the scheduler to the system is “Use an Intermediary”.

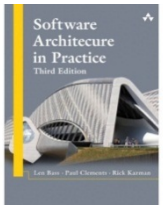
Common side effects of Use an Intermediary are:

- modifiability: how to ensure that all communication passes through the intermediary?



Tactics and Interactions - 9





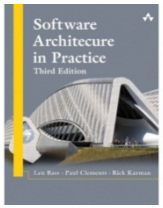
Tactics and Interactions – 10.

A tactic to address the concern that all communication passes through the intermediary is “Restrict Communication Paths”.

Common side effects of Restrict Communication Paths are:

- performance: how to ensure that the performance overhead of the intermediary are not excessive?

Note: this design problem has now become recursive!



How Does This Process End?

- Each use of tactic introduces new concerns.
- Each new concern causes new tactics to be added.
- Are we in an infinite progression?
- No. Eventually the side-effects of each tactic become small enough to ignore.

Broker pattern

Broker is an intermediary software that provides certain services

Examples of Brokers:

- Load balancer
- EJB server

Broker pattern: Example: Load balancer

Load balancer
- Distributes requests to different servers in a server farm
- Takes care of non-availability issues by switching to another server

Some algorithms used to distribute load are:

•Round robin
•Least connections

Load balancer vendors
- F5
- Citrix

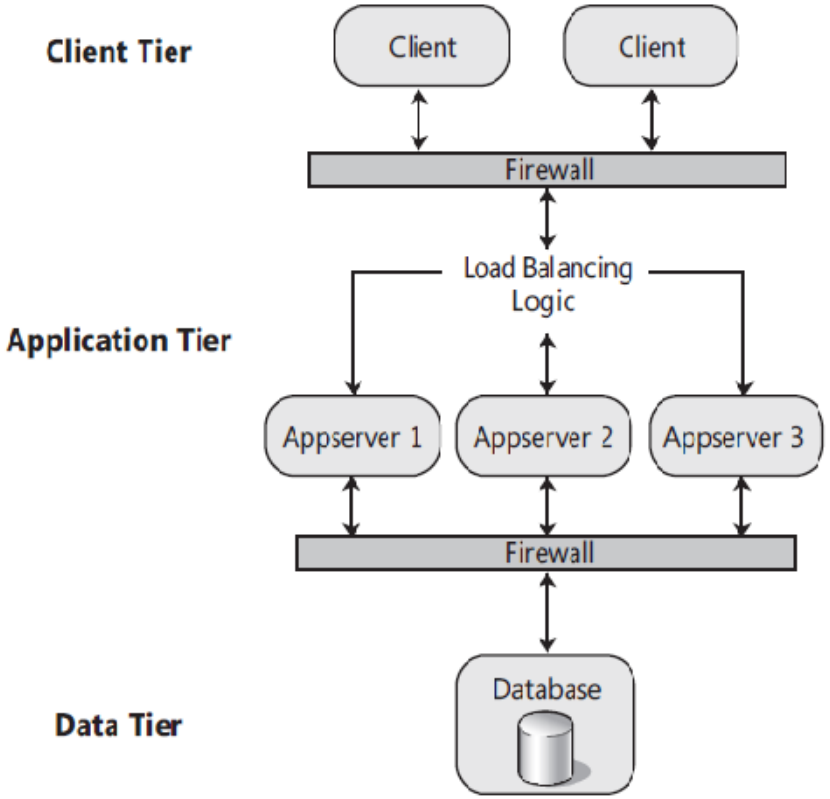


Figure 9
A load-balanced cluster

Broker pattern

- Mediates between client and server to provide different kinds of services

- It adds a layer of indirection that may cause performance issues
- A broker can be a single point of failure and hence needs to be designed for fault tolerance
- It can also be a target for security attacks