Module Five: Service Oriented Architecture: architectural patterns and modelling methods of SOA, designing a distributed service system with SOA

Our materials for this module:

- **Thomas Erl, Service-Oriented Architecture, Concepts, Technology, and Design, ISBN：9787030336422，**
  - Chapter 3
  - Chapter 8
- **Online resources:**
  https://www.guru99.com/soa-principles.html

Module 5 Learning Outcomes

- Analyze common characteristics of SOA
- List common benefits of using SOA
- Assess common pitfalls of adopting SOA
- Understand the service orientation in the enterprise
- Be familiar with anatomy of a service oriented architecture
- Be able to explain the common principles of service orientation
- Understand how service orientation principles inter-relate
- Be able to evaluate Web service support for service-orientation principles

Guided questions for Module 5

- What does the “service oriented” concept mean?
- What is a service oriented architecture?
- What are services within SOA?
- What is a business process?
- How is a service related to a business process?
- How services relate to each other within SOA?
- How services communicate?
- How services are designed?

Guided questions for Module 5

- What are the principles of service-orientation?
- What are characteristics of contemporary SOA?
- What are the benefits of SOA?
- What are the pitfalls of adopting SOA?
- How service orientation applies to the enterprise
- How service orientation principles inter-relate?
- How Web services support service orientation principles?

Service Oriented Architecture

Architecture

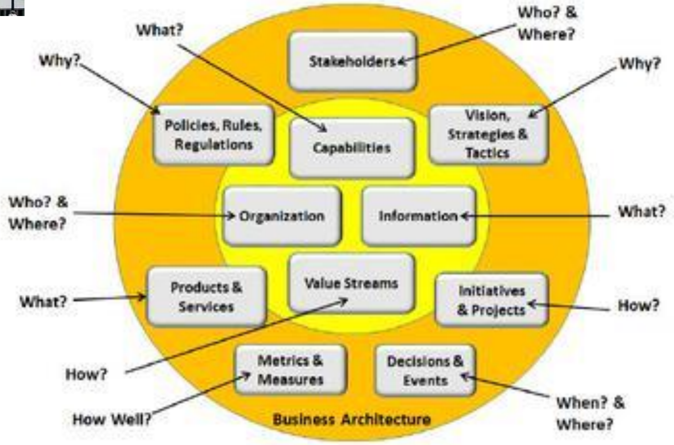
- The process of planning, designing and constructing structures

Architecture

- The process of planning, designing and constructing structures
  - Building

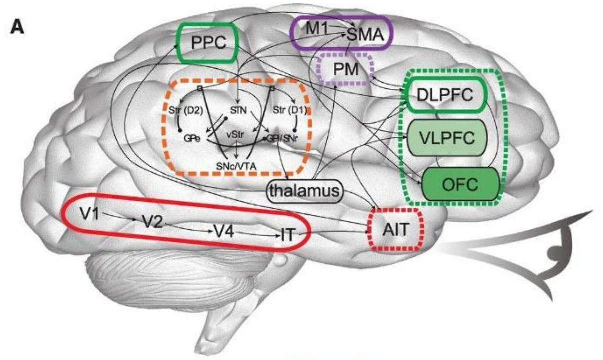
Architecture

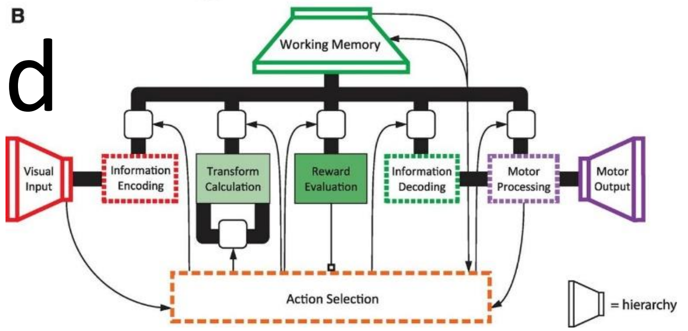
- The process of planning, designing and constructing structures
  - Building
  - Business

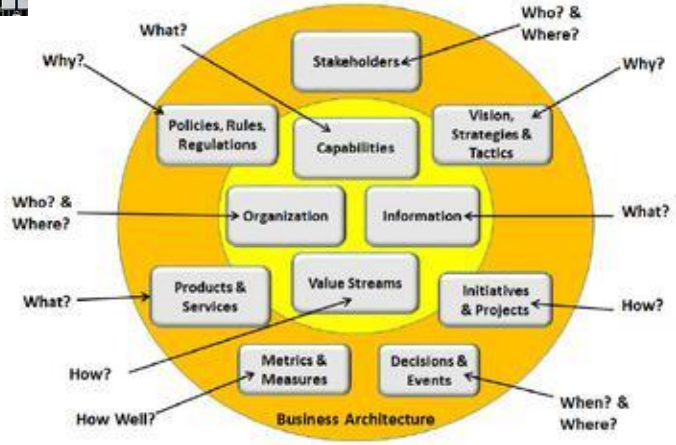


Architecture

- The process of planning, designing and constructing structures
    - Building
    - Business
    - Cognitive

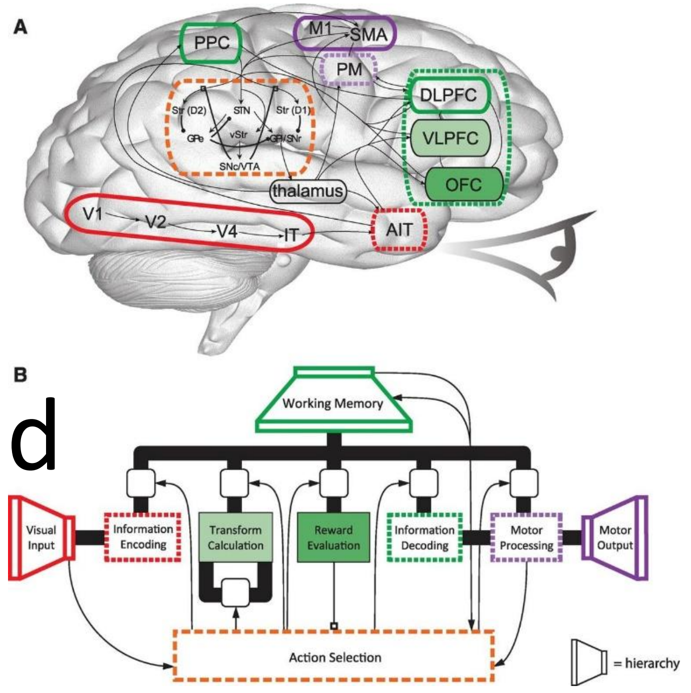


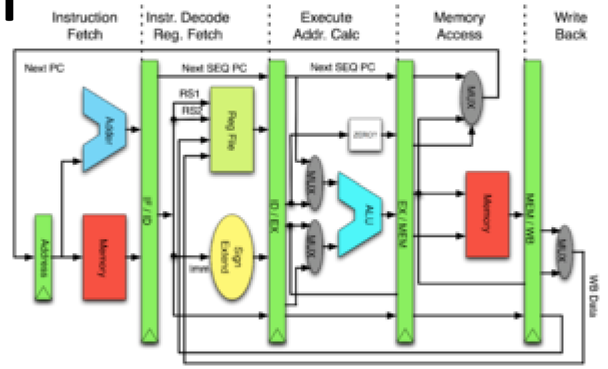


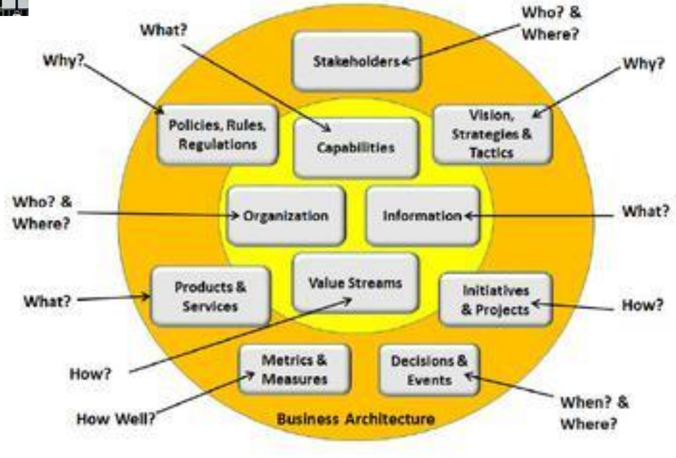


Architecture

- The process of planning, designing and constructing structures
  - Building
  - Business
  - Cognitive
  - Computer

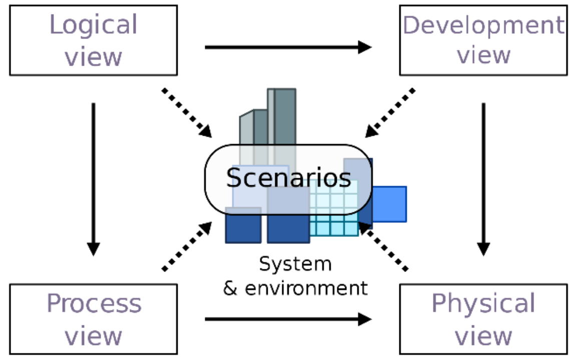


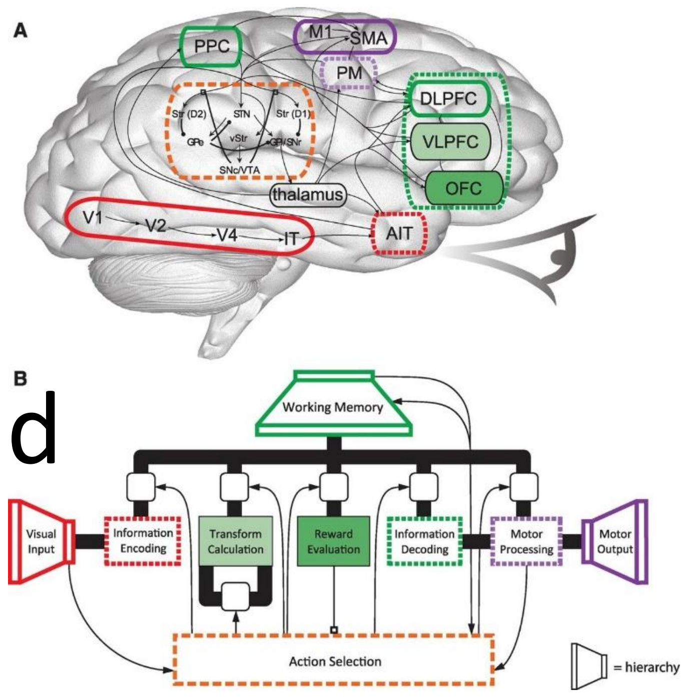


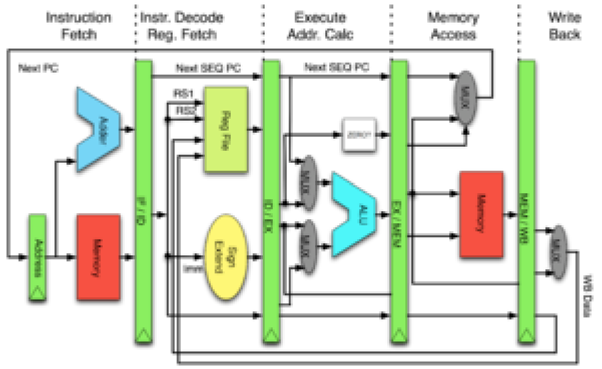


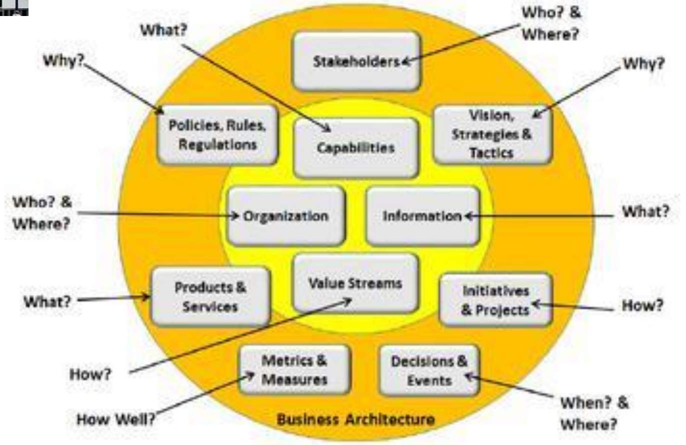
Architecture

- The process of planning, designing and constructing structures
  - Building
  - Business
  - Cognitive
  - Computer
  - Software
  - …..









Software Architecture

- Structures of a software system and the discipline of creating such structures and systems

Software Architecture

- Structures of a software system and the discipline of creating such structures and systems
- Each structure compromises
  - Software elements
  - Relations among the elements
  - Properties of elements
  - Properties of relations

Software Architecture

- Structures of a software system and the discipline of creating such structures and systems
- Each structure compromises
  - Software elements
  - Relations among the elements
  - Properties of elements
  - Properties of relations
- Blueprint for the system and the developing project, laying out the task necessary to be executed by the design teams

Software Architecture

- The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

Service Oriented Architecture

- A style of software design where services are provided to the other companies by application components, through a communication protocol over a network

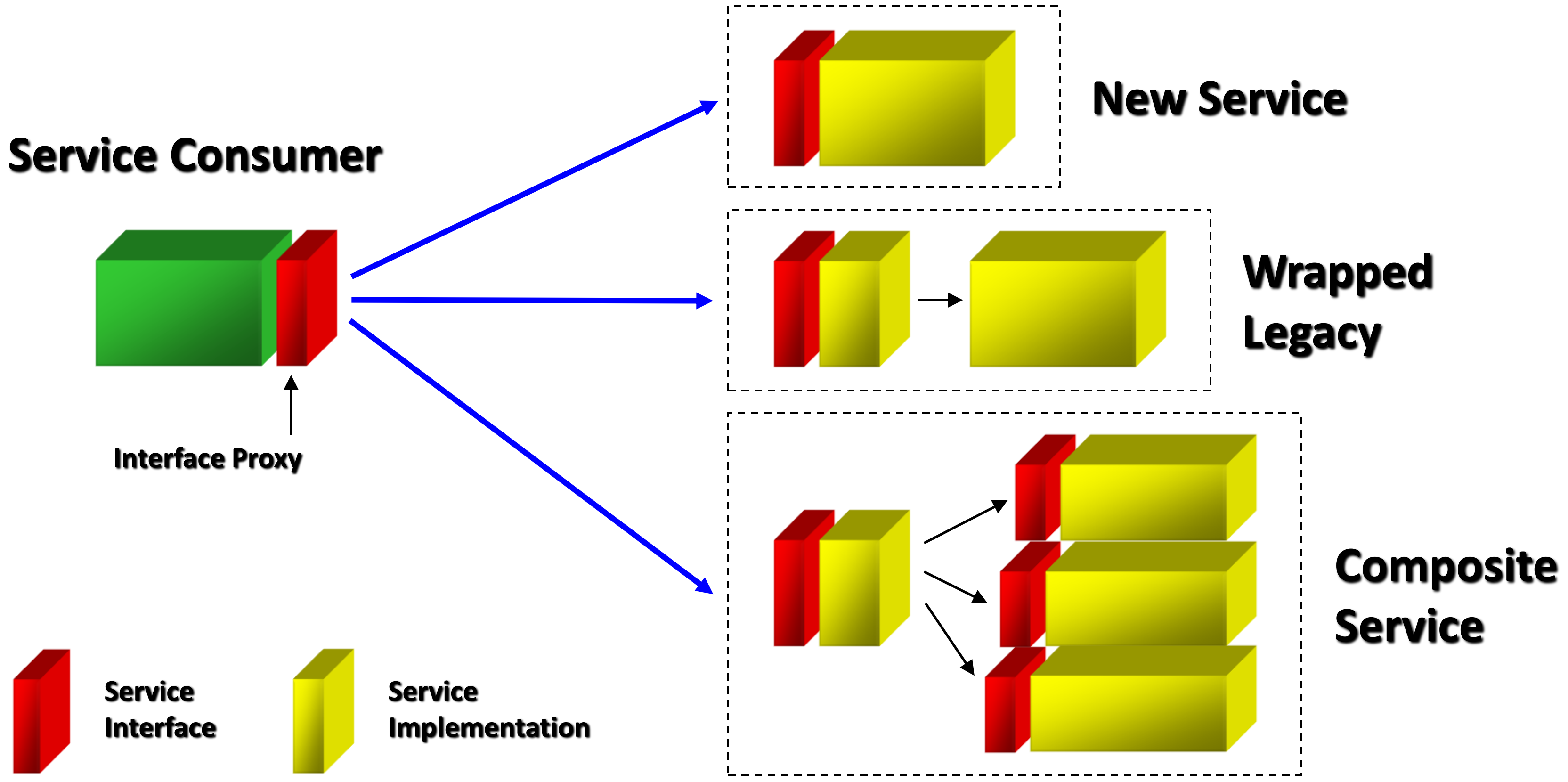
What is SOA?

- Service-Oriented Architecture (SOA) is a set of principles and methodologies for designing and developing software in the form of interoperable services.
- These services are well-defined business functionalities that are built as software components that can be reused for different purposes.
- SOA design principles are used during the phases of systems development and integration.

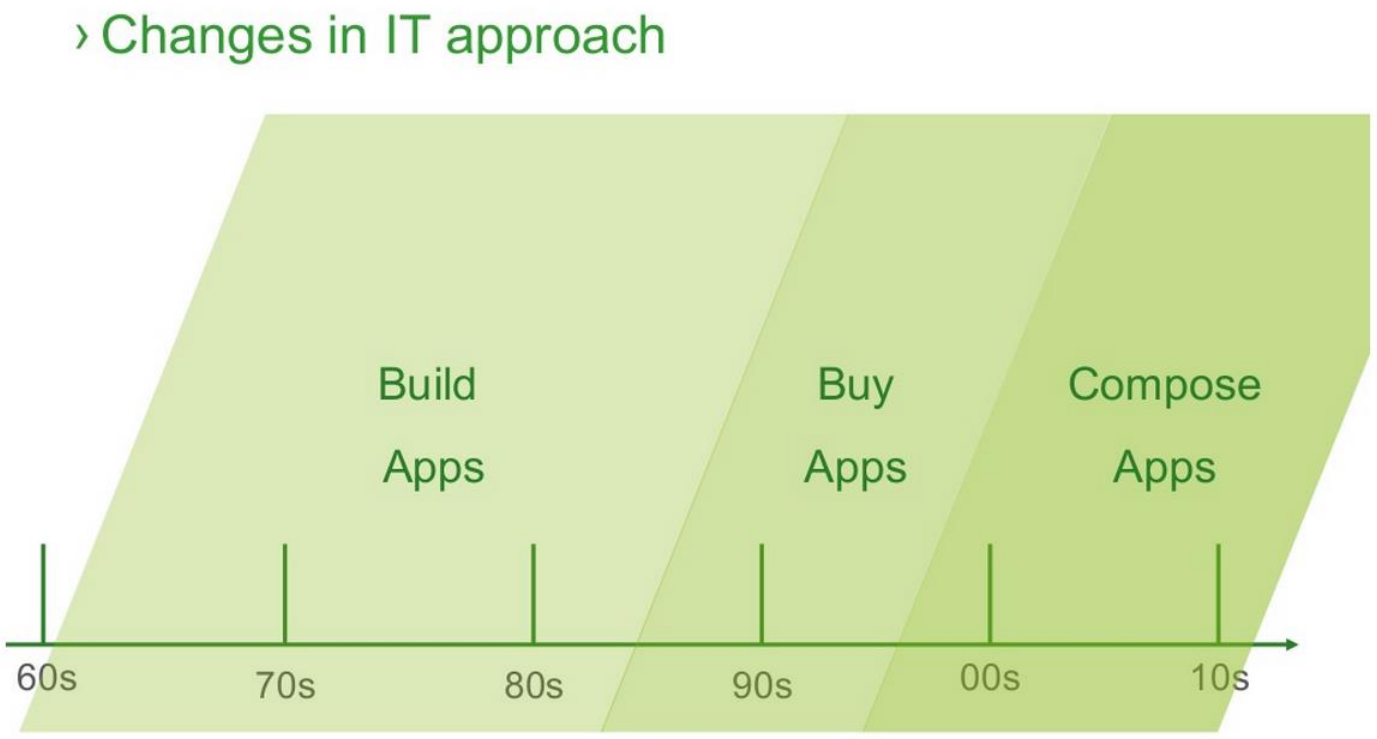
SOA Service

- A discrete unit of functionality that can be accessed remotely and acted upon and updated independently
- A repeatable task, for example:
  - Open an account
  - Perform a credit check
  - retrieving a credit card statement online

Anatomy of a Service

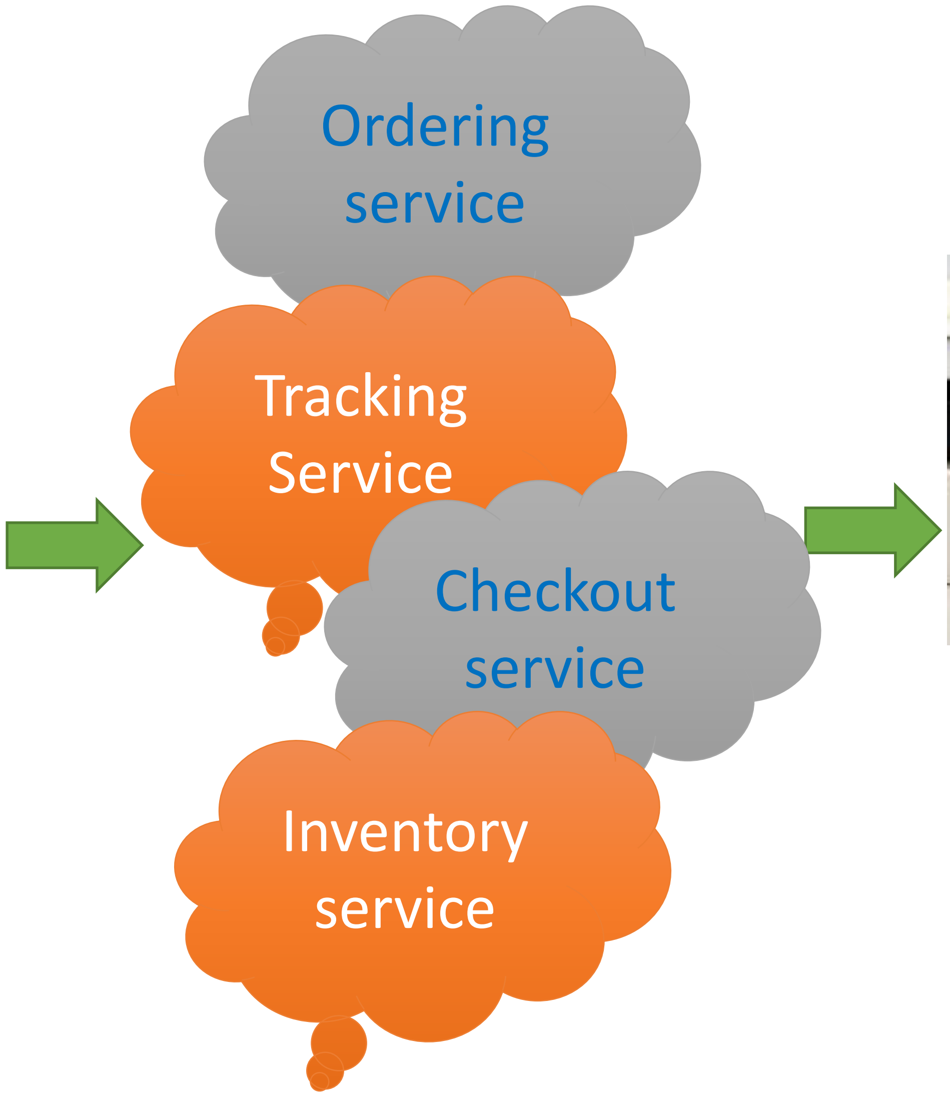


How did it come to SOA?



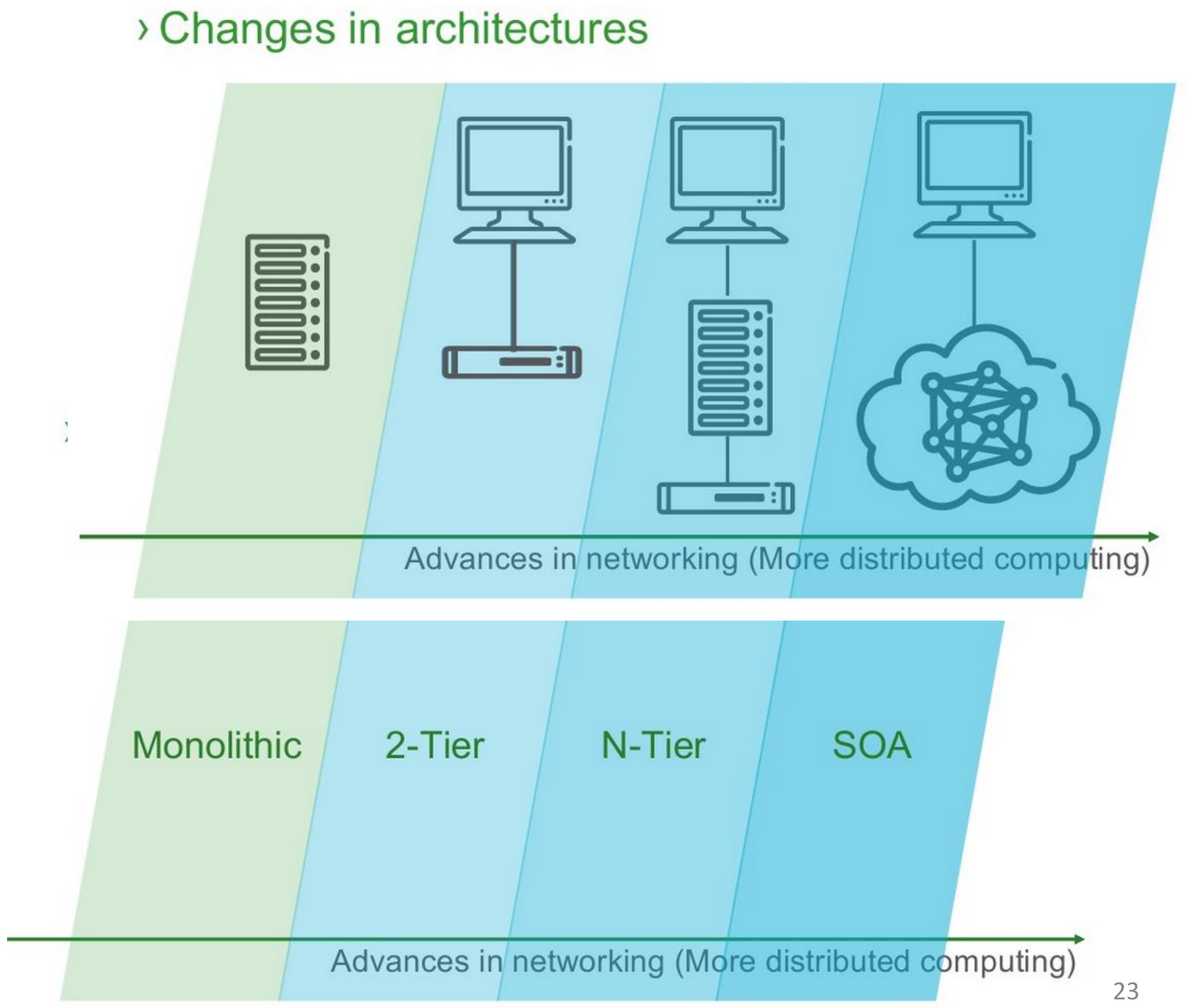
How did it come to SOA?

A website selling gadgets

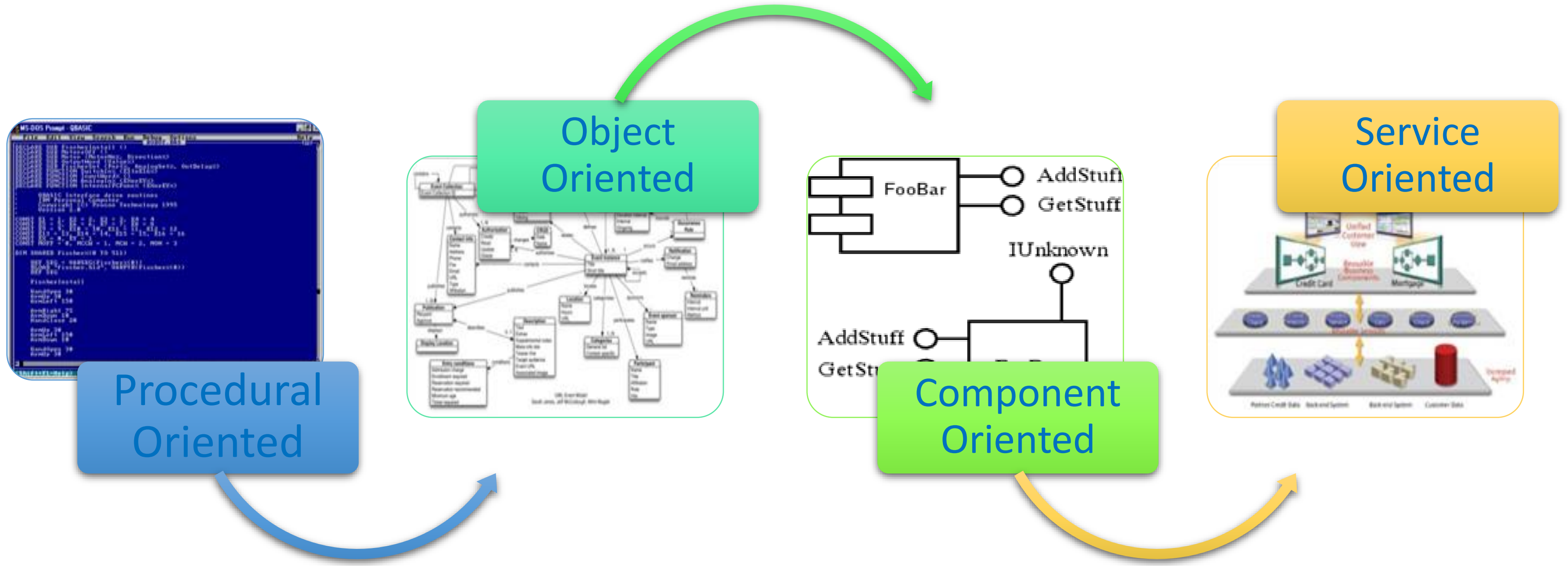


Serving its customers

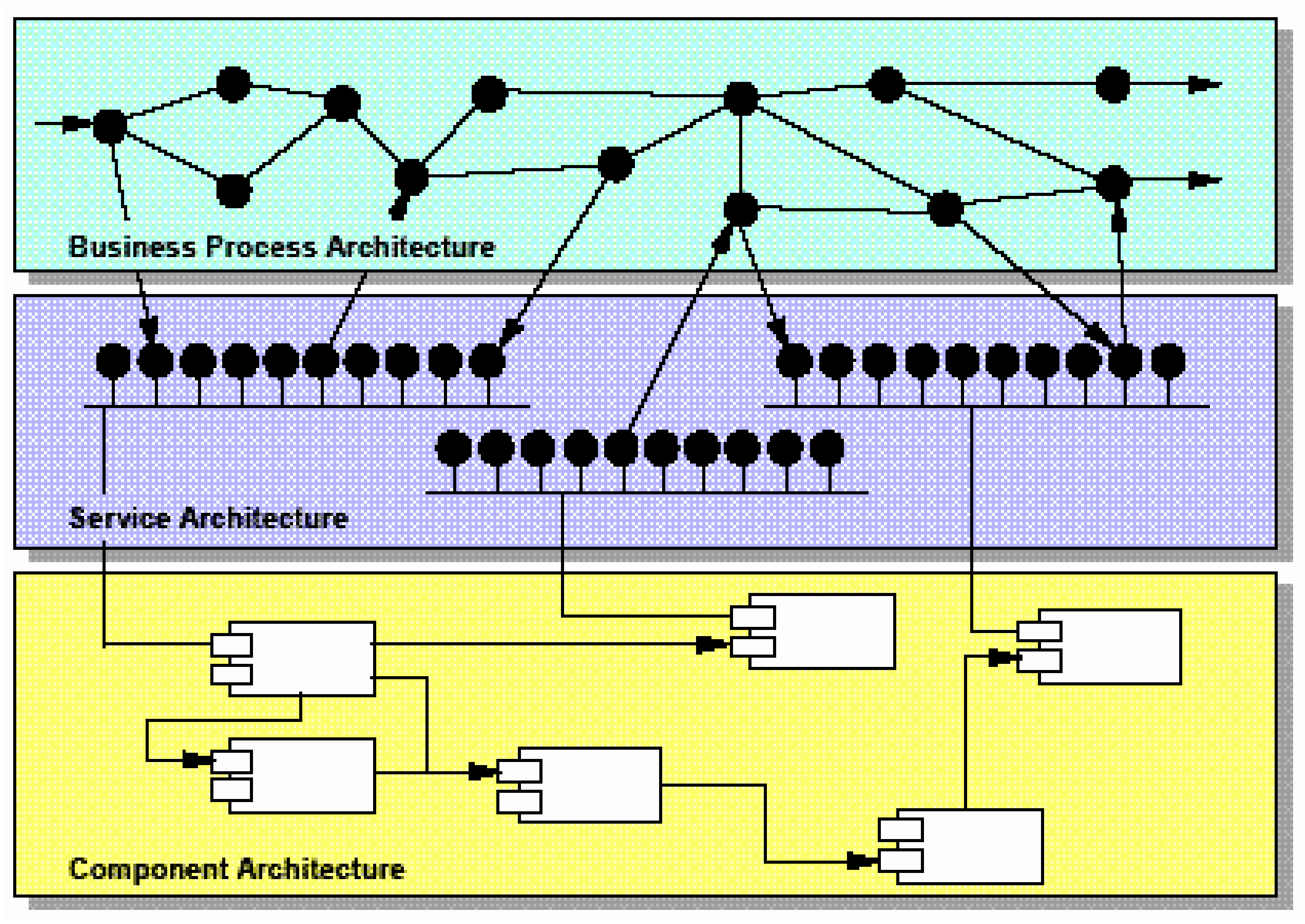
How did it come to SOA?


› Changes in architectures
Advances in networking (More distributed computing)
Monolithic
2-Tier
N-Tier
SOA
Advances in networking (More distributed computing)

How did it come to SOA?



Isolating the Business Process from the Implementation



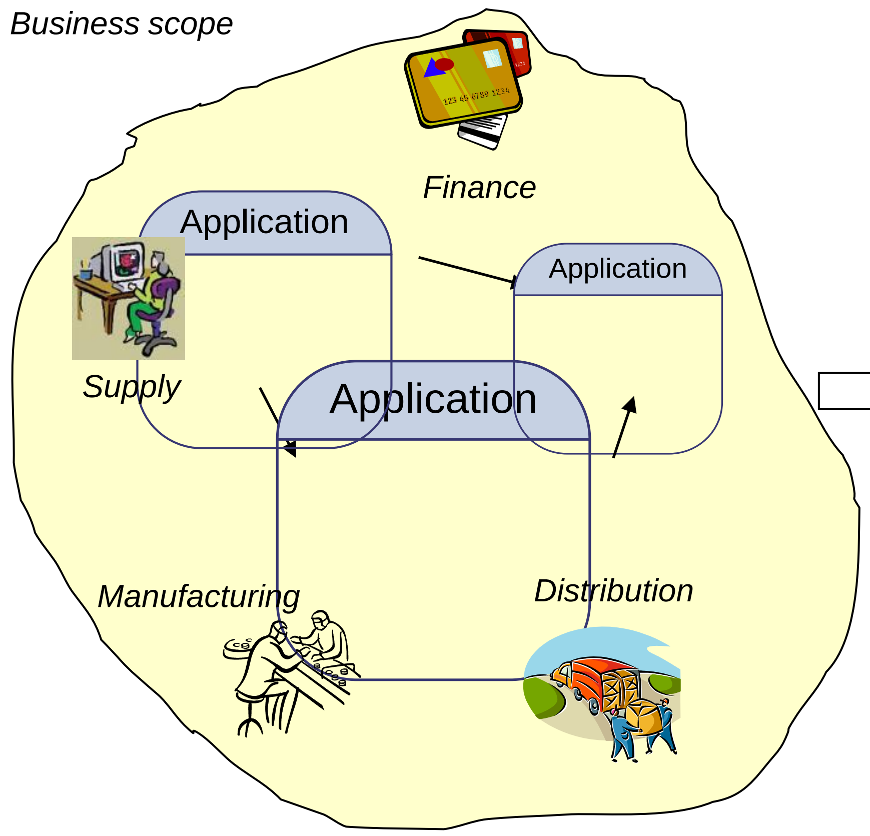
Why SOA in Enterprise?

- Why to introduce SOA in an organization? What are the benefits for enterprises?

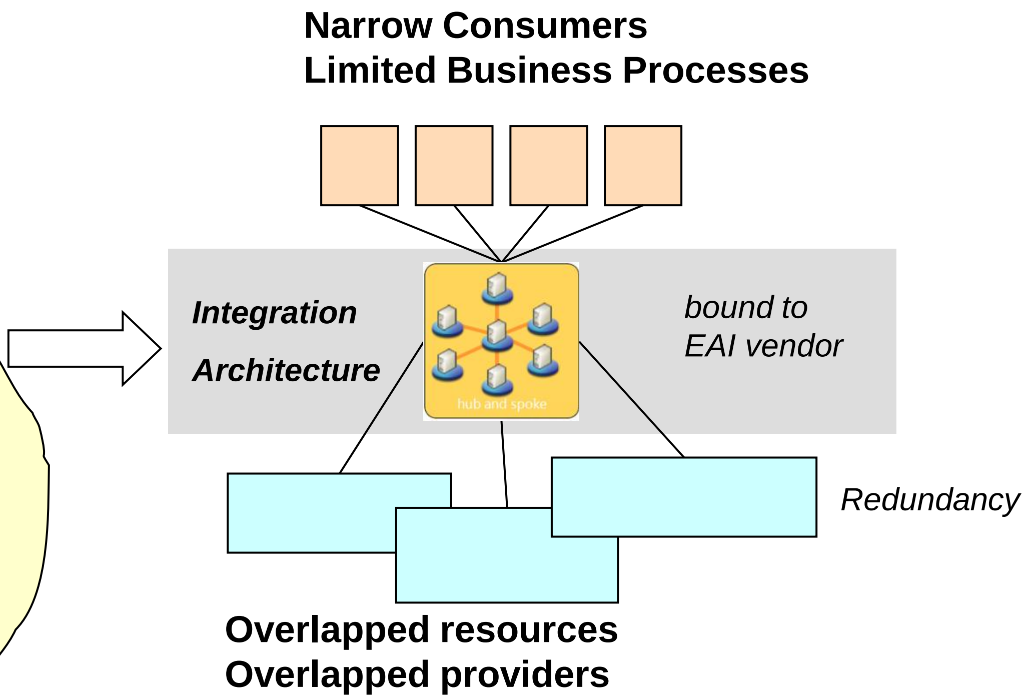
Why SOA?

- From application centric to service centric environment

Application Centric

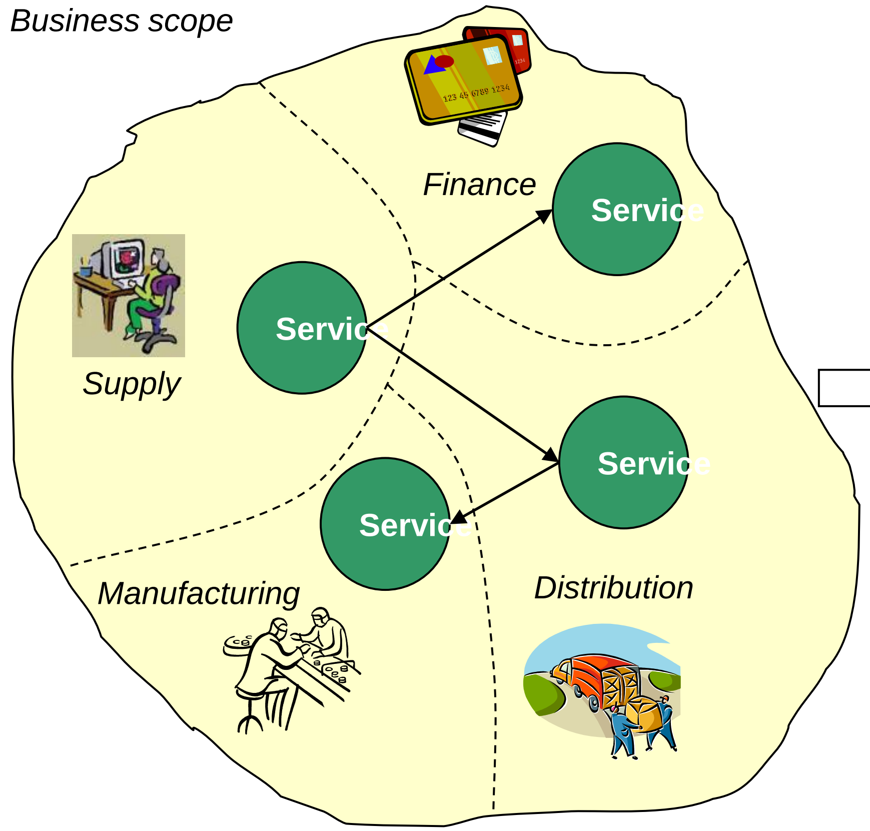


Business functionality is duplicated in each application that requires it.

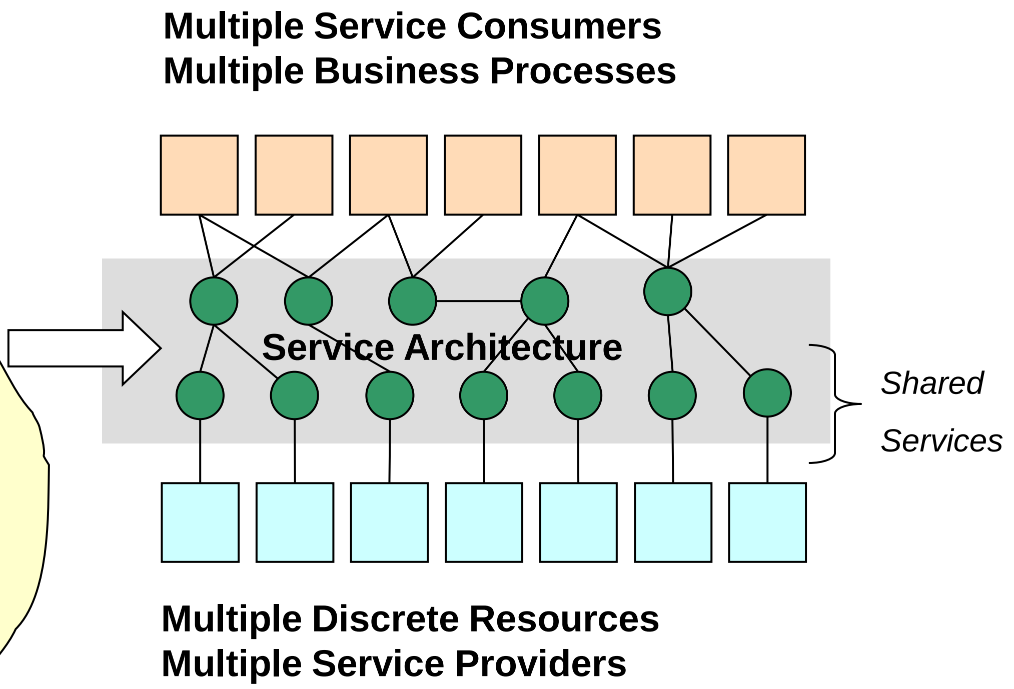


EAI 'leverage' application silos with the drawback of data and function redundancy.

Service Centric



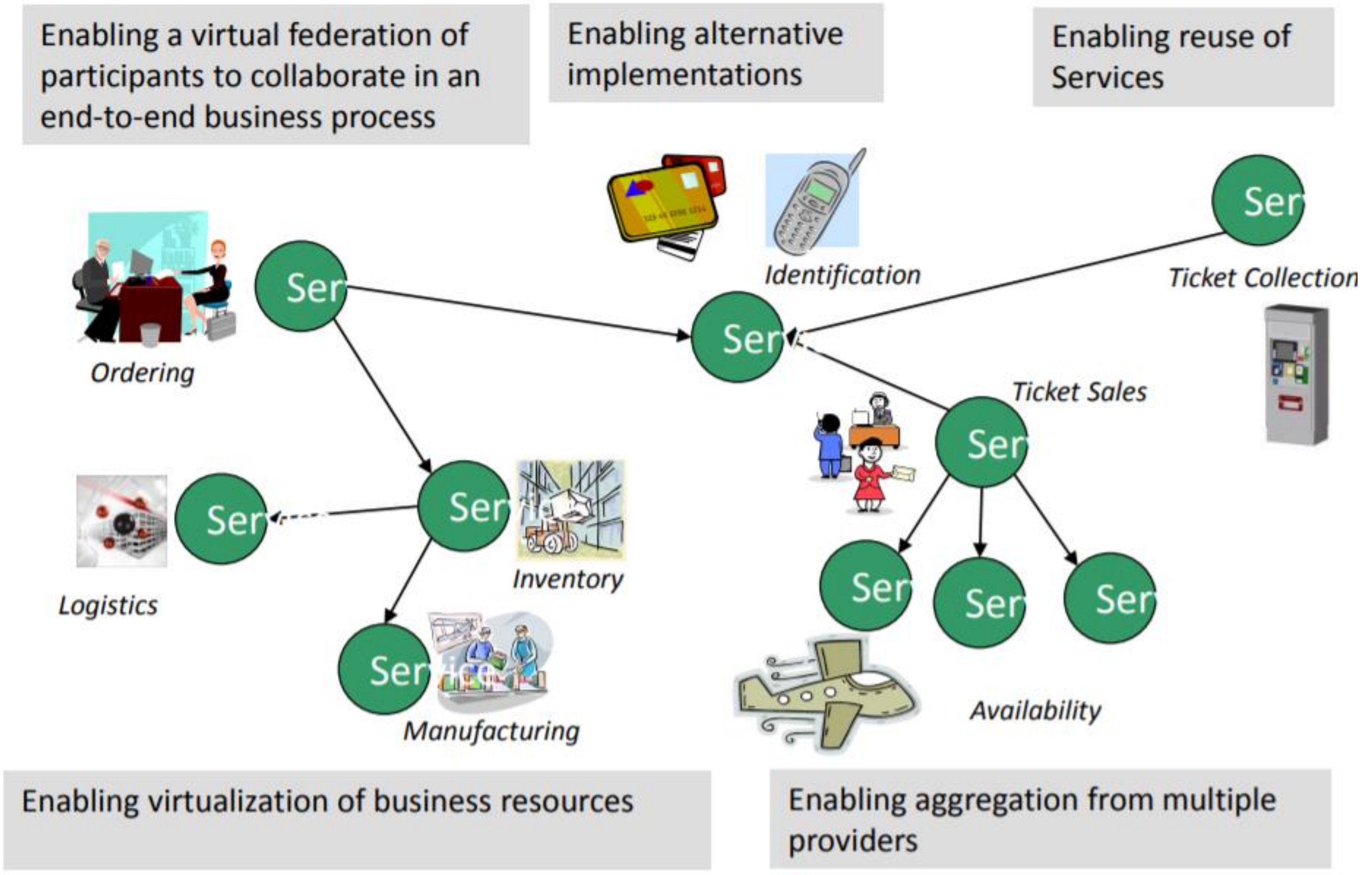
SOA structures the business and its systems as a set of capabilities that are offered as Services, organized into a Service Architecture


Multiple Service Consumers
Multiple Business Processes
Service Architecture
Shared
Services
Multiple Discrete Resources
Multiple Service Providers

Service virtualizes how that capability is performed, and where and by whom the resources are provided, enabling multiple providers and consumers to participate together in shared business activities.

Why SOA in Enterprises? ENABLE FLEXIBLE, FEDERATED BUSINESS PROCESSES

- Enable flexible, federated business processes


Enabling a virtual federation of participants to collaborate in an end-to-end business process
Enabling alternative implementations
Enabling reuse of Services
Ser
Ticket Collection
Identification
Ser
Ordering
Ser
Ticket Sales
Ser
Ser
Logistics
Ser
Inventory
Ser
Ser
Ser
Ser
Manufacturing
Availability
Enabling virtualization of business resources
Enabling aggregation from multiple providers

Business benefits – decreased cost

- Decreased cost:
  - Add value to core investments by leveraging existing assets
  - New systems can be built faster for less money
    - Reducing integration expense
    - Built for flexibility
    - Long term value of interoperability

Business benefits – increased productivity

- Increased employee productivity:
  - Built on existing skills
  - Consolidate duplicate functionality

Business benefits - partnership

- Built for partnerships:
  - Standards based
  - Business relationships expressed via service interactions
  - Integration is driven by what is needed, not what is technically possible

Business benefits – agility

- Agility - Built for change
  - Helps applications evolve over time and last
  - Abstract the backend and replace over time
  - Focusing on core-competencies
  - Incremental implementation approach is supported
  - Service Outsourcing – new business model!

Technical Benefits

- Services Scale
  - Build scalable, evolvable systems
  - Scale down to mobile devices
  - Scale up to for large systems or across organizations

Technical Benefits

- Services Scale
  - Build scalable, evolvable systems
  - Scale down to mobile devices
  - Scale up to for large systems or across organizations
- Manage complex systems
  - Does not require centralized services
  - Empowers users with high end communication

Technical Benefits

- Services Scale
  - Build scalable, evolvable systems
  - Scale down to mobile devices
  - Scale up to for large systems or across organizations
- Manage complex systems
  - Does not require centralized services
  - Empowers users with high end communication
- Platform independent use

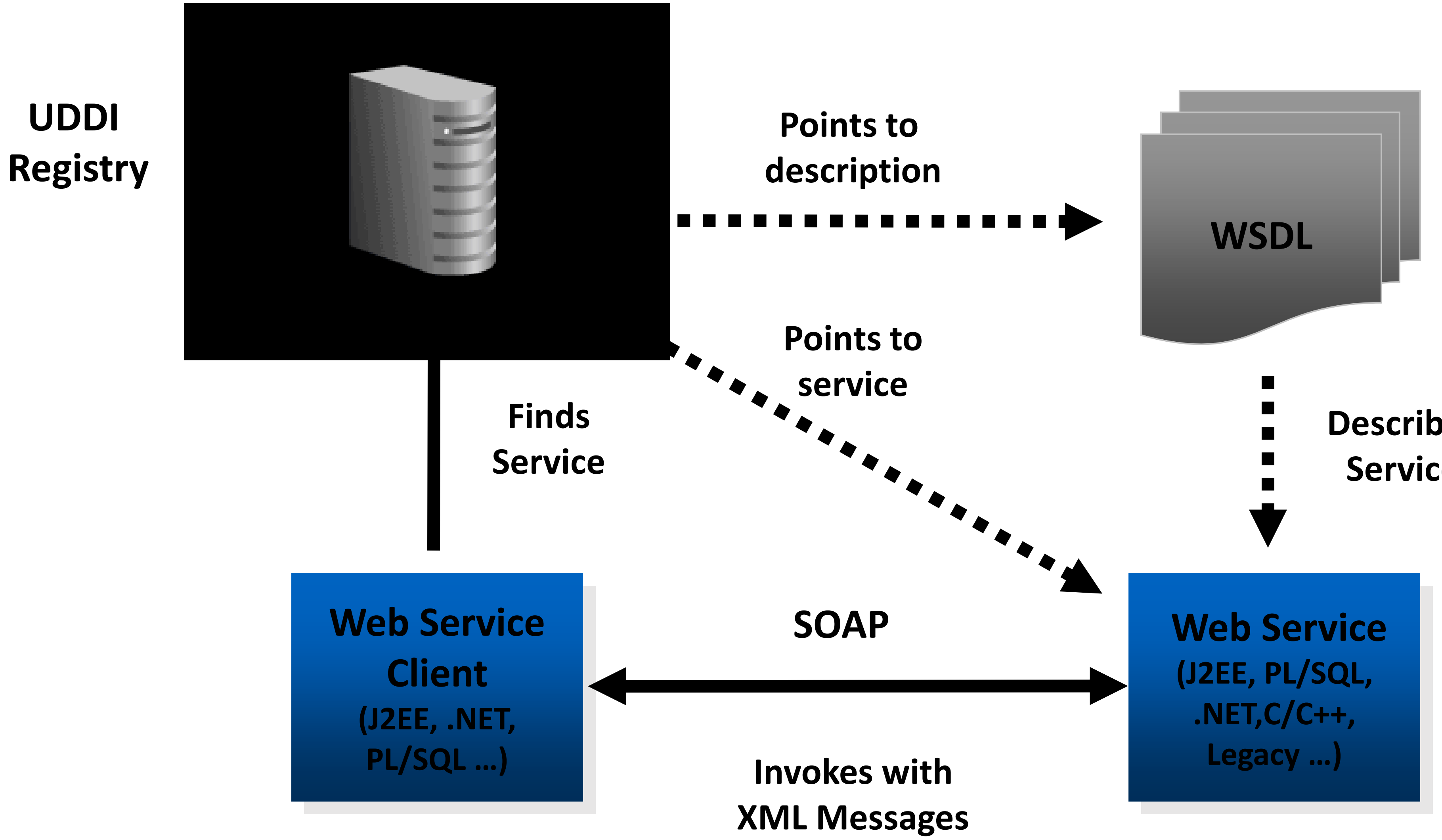
Technical Benefits

- Services Scale
  - Build scalable, evolvable systems
  - Scale down to mobile devices
  - Scale up to for large systems or across organizations
- Manage complex systems
  - Does not require centralized services
  - Empowers users with high end communication
- Platform independent use
- Loose Coupling allows flexibility

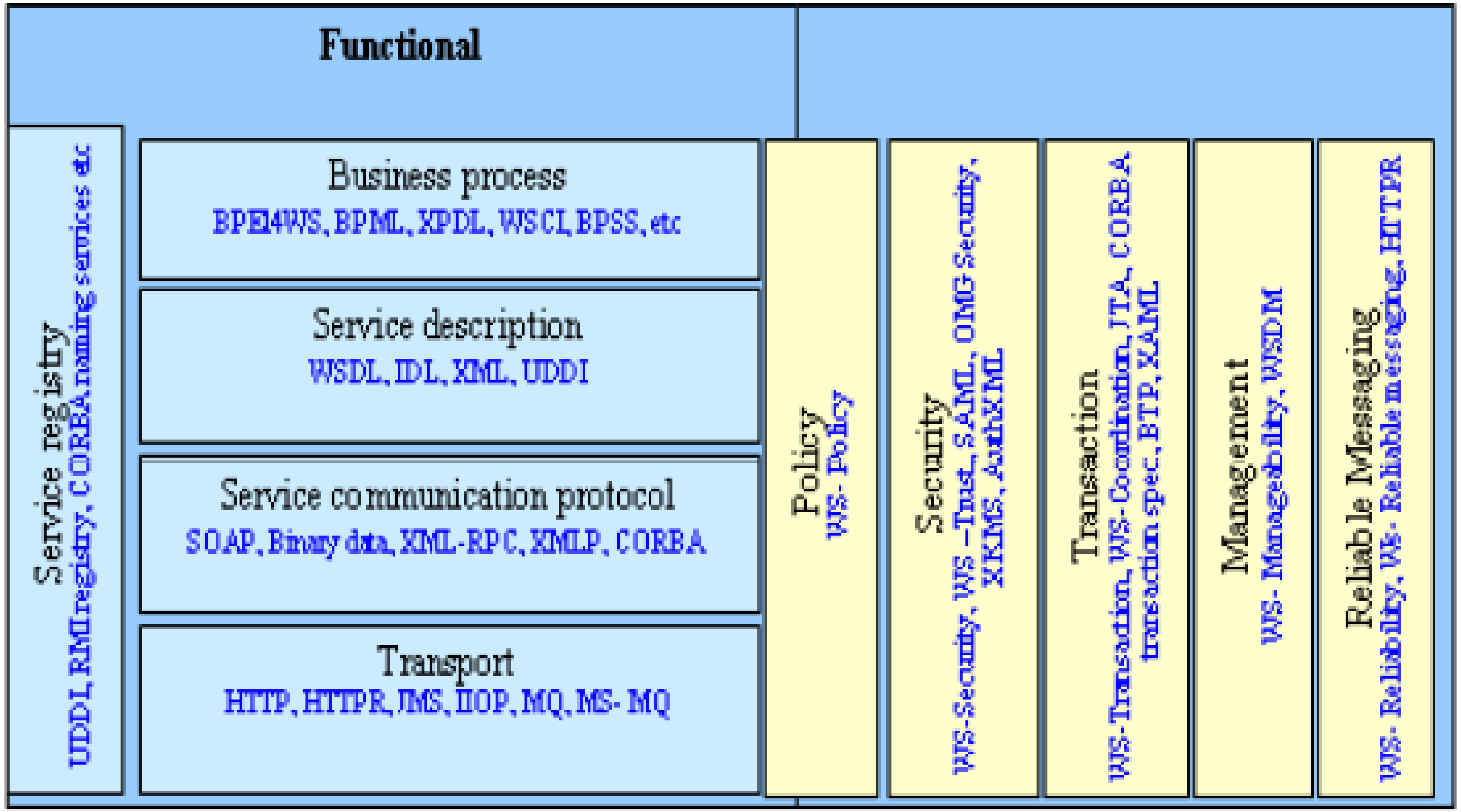
Web service implementation of SOA

- Service-oriented architecture can be implemented with web services.
- Functional building blocks are accessible over standard internet protocols
  - Independent of platforms and programming languages
- Web services can represent
  - New applications
  - Wrappers around existing legacy systems
- SOA can be implemented using other technologies

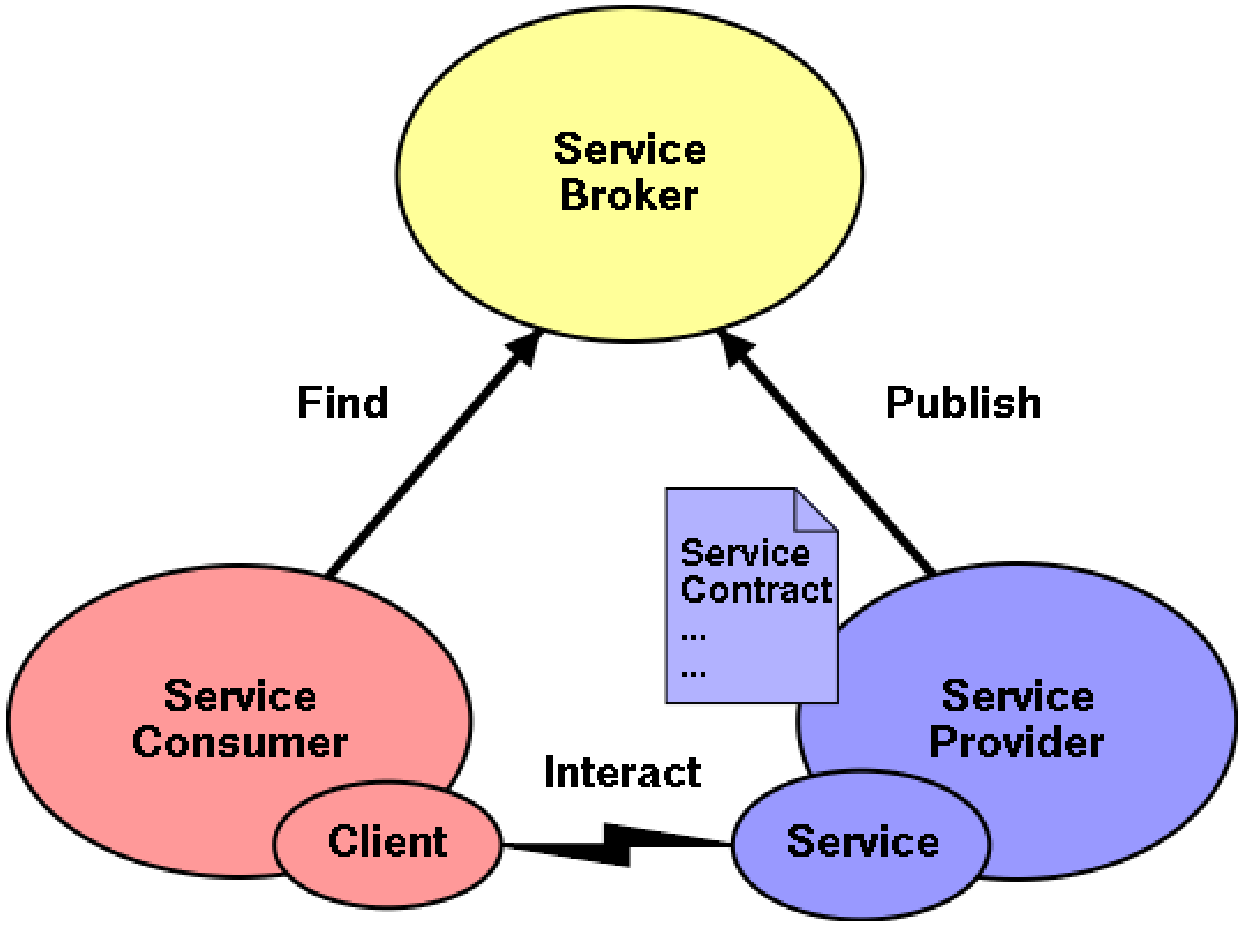
Basic Web Services


UDDI Registry
Points to description
WSDL
Points to service
Finds Service
Describes Service
Web Service Client (J2EE, .NET, PL/SQL …)
SOAP
Web Service (J2EE, PL/SQL, .NET,C/C++, Legacy …)
Invokes with XML Messages

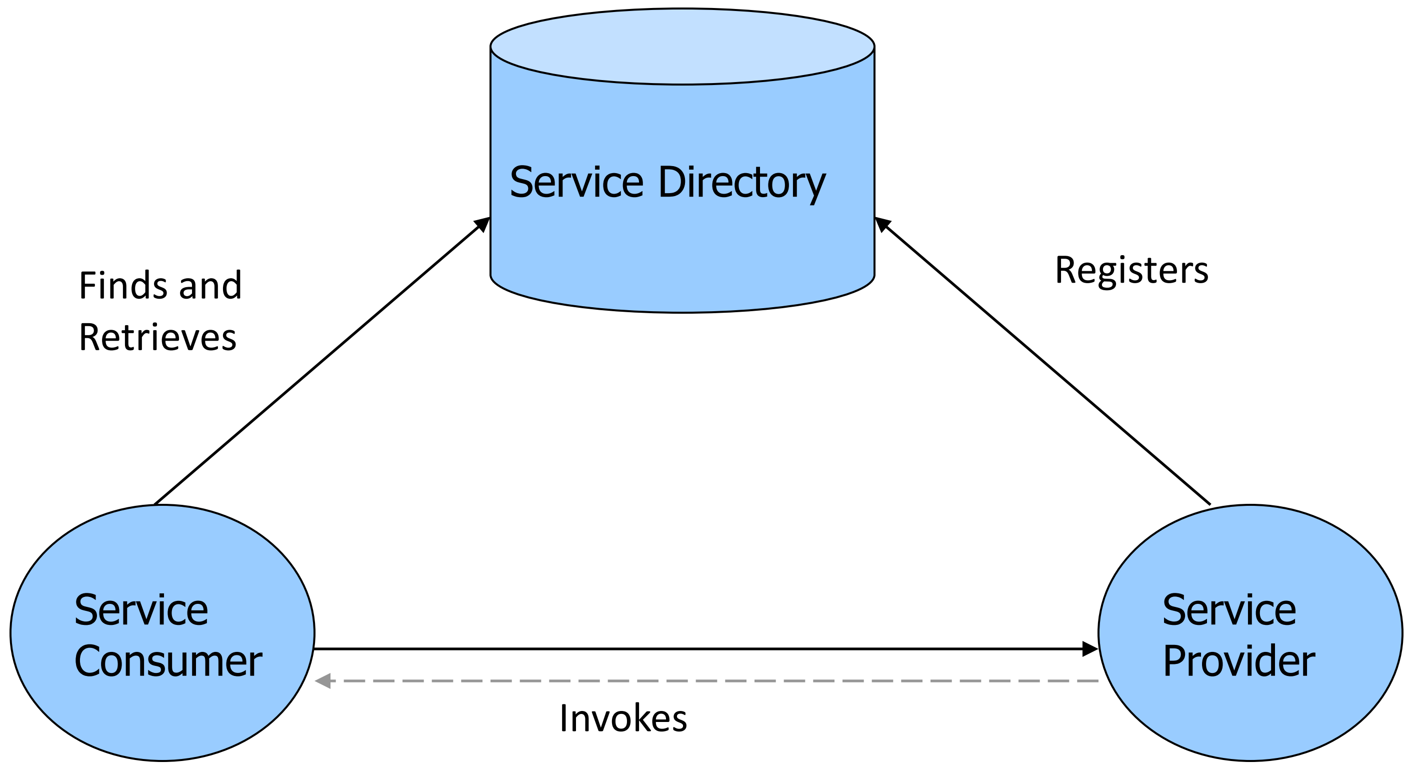
SOA Standards Stack


Functional
Service registry
UDDI, RMI registry, CORBA naming services etc
Business process
BPEL4WS, BPML, XPDL, WSCI, BPSS, etc
Service description
WSDL, IDL, XML, UDDI
Service communication protocol
SOAP, Binary data, XML-RPC, XMLP, CORBA
Transport
HTTP, HTTPR, JMS, IIOP, MQ, MS- MQ
Policy
WS- Policy
Security
WS-Security, WS –Trust, SAML, OMG Security, XKMS, AuthXML
Transaction
WS-Transaction, WS-Coordination, JTA, CORBA transaction spec., BTP, XAML
Management
WS- Manageability, WSDM
Reliable Messaging
WS- Reliability, WS- Reliable messaging, HTTPR

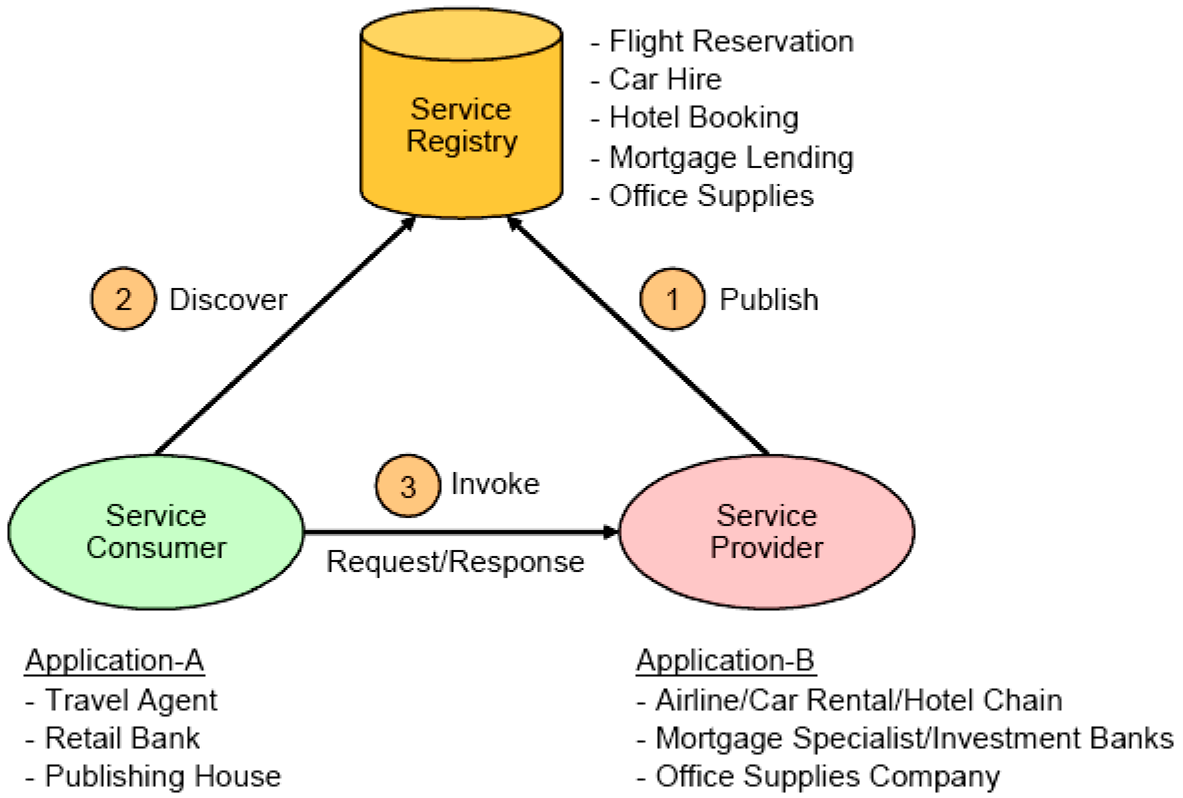
SOA Architecture



SOA Architecture



SOA Components and Operations



Basic Components of an SOA

- SOA consists of the following three components:
  - Service provider
  - Service consumer
  - Service registry
- Each component can also act as one of the two other components.
  - For instance, if a service provider needs additional information that it can only acquire from another service, it acts as a service consumer.

Characteristics of service-oriented architectures

- Any function can be a service but need not be. The criteria for whether a function needs to be a service is based on its reuse potential
  - Business process services
    - createStockOrder, reconcileAccount, renewPolicy
  - Business transaction services
    - checkOrderAvailability, createBillingRecord
  - Business function services
    - calculateDollarValueFromYen, getStockPrice
  - Technical function services
    - auditEvent, checkUserPassword,checkUserAuthorisation

Service Design Principles

- The principles of service-orientation provide a means of supporting and achieving a foundation paradigm based upon building of SOA characteristics.

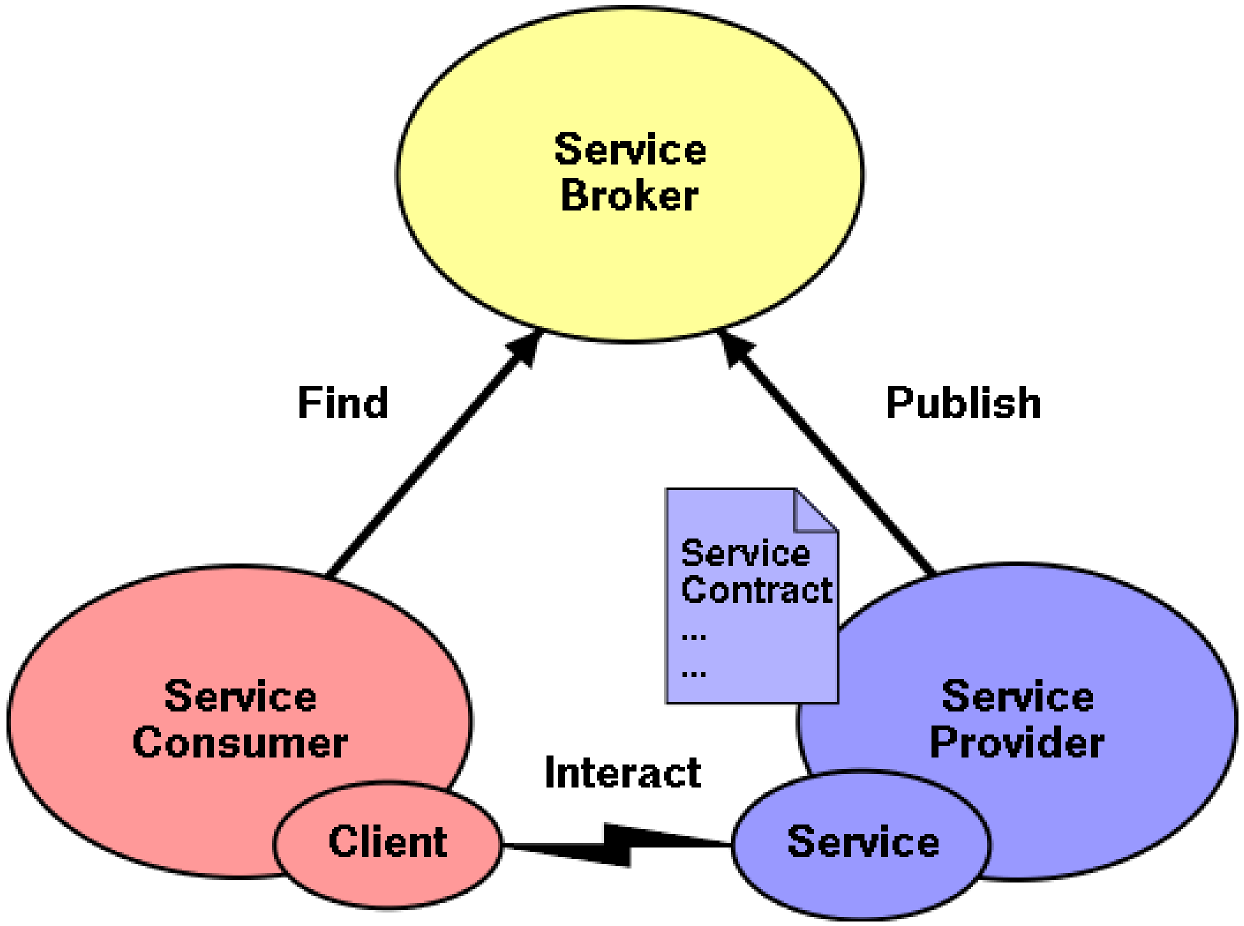
Common Service Design Principles

- Standardized Service Contracts
- Loose Coupling
- Abstraction
- Reusability
- Autonomy
- Statelessness
- Discoverability
- Composability

Standardized Service Contracts

- Services share a formal contract
- Services adhere to a communications agreement as defined collectively by one or more service description documents
- For services to interact, they need not share anything but a formal contract that describes each service and defines the terms of information exchange.

SOA Architecture



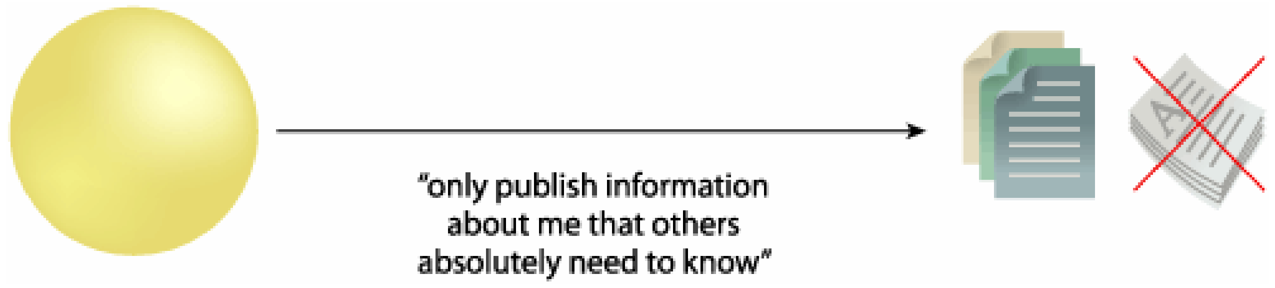
Loose Coupling

- Services are loosely coupled
- Services maintain a relationship that minimizes dependencies and only maintain an awareness of each other
- Services must be designed to interact without the need for tight, cross-service dependencies
- A service is defined solely by an implementation-independent interface
- Services should be able to change their implementation without impacting service consumers

What is loose coupling?

- SOA is an architectural style with characteristics such as *loose coupling, reuse, and simple and composite implementations*
- Loosely coupled services, even if they use incompatible system technologies, can be joined together dynamically to create composite services or disassembled just as easily into their functional components
- Service requesters depend on the interface and not on the service provider’s implementation
- Various aspects of service interactions such as time (availability), protocol, message format, language, platform, or location are specified in the service interface separate from the service implementation

Abstraction

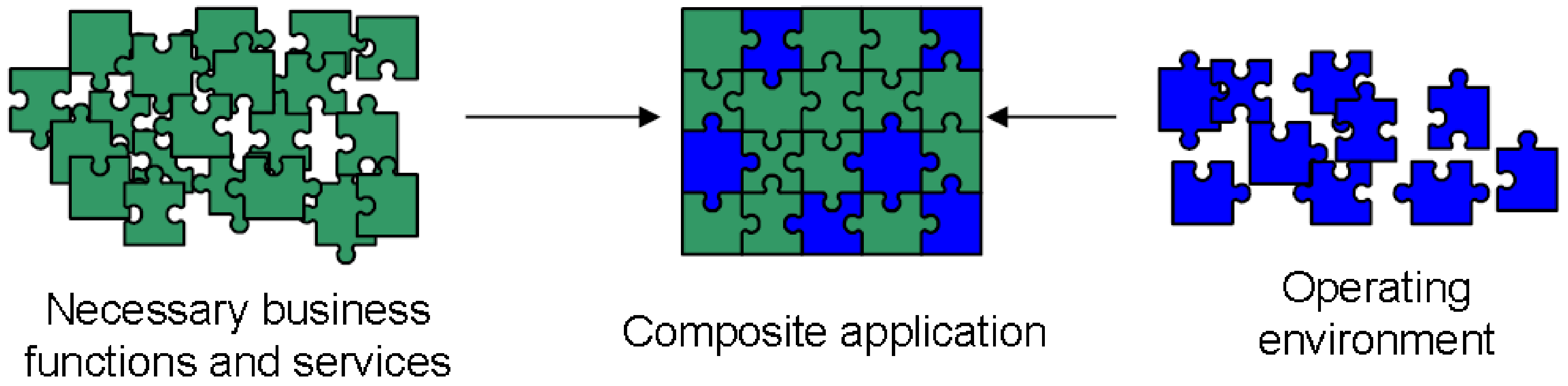

"only publish information about me that others absolutely need to know"

- Services abstract underlying logic
- Beyond what is described in the service contract, services hide logic from the outside world
- The only part of a service that is visible to the outside world is what is exposed via the service contract. Underlying logic, beyond what is expressed in the descriptions that comprise the contract, is invisible and irrelevant to service requestors.

Reusability

- Services are reusable
- Logic is divided into services with the intention of promoting reuse
- Regardless of whether immediate reuse opportunities exist; services are designed to support potential reuse

Service design principles: Reuse



- Concept
  - A service interface should be designed with reuse in mind
  - Anticipate reuse scenarios

Reuse

- Consequences
  - Well factored service interfaces:
    - Anticipate usage scenarios and consequently facilitate reuse
  - Poorly factored service interfaces:
    - Hinder reuse and encourage functional duplication, which can result in architectural decay (loss of architectural integrity over time)

Autonomy

- Services are autonomous
- Services have control over the logic they encapsulate
- The logic governed by a service resides within an explicit boundary. The service has control within this boundary and is not dependent on other services for it to execute its governance.

Statelessness

- Services are stateless
- Services should not be required to manage state information, as that can impede their ability to remain loosely coupled. Services should be designed to maximize statelessness even if that means deferring state management elsewhere.
- Service implementations should not hold conversational state across multiple requests.
  - Communicate complete information at each request.
- Each operation should be functionally isolated (separate, independent).

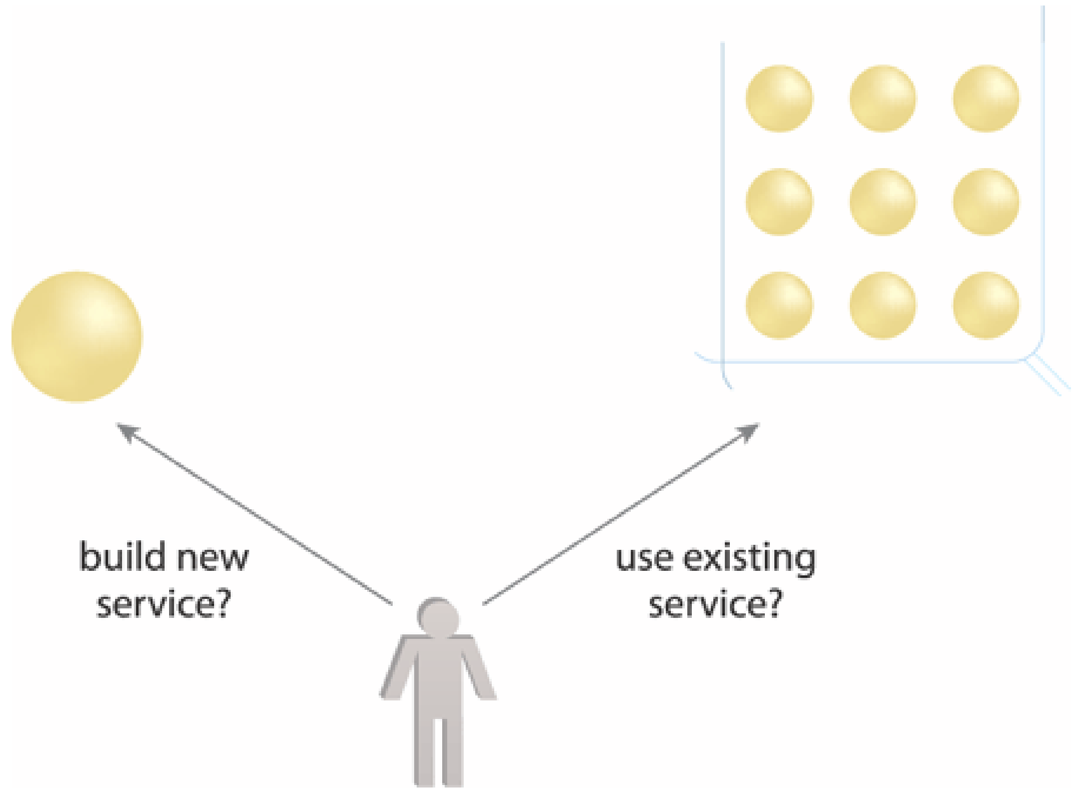
Statelessness--Consequences

- Stateless/connectionless services:
  - Benefit adaptability owing to the independence that exists between successive service requests of a client and the service instance that fulfils each request.
    - This is an enabler for improved runtime qualities (for example, service request throughput or concurrent service requests) using pooling and sharing of service instances (client-service independence).
- Stateful services:
  - Hinder adaptability as a consequence of tight dependency (coupling) between a clients successive service requests
    - There is then a need for a specific service instance to fulfil a particular request (client-service affinity).

Discoverability

- Services are discoverable
- Services are designed to be outwardly descriptive so that they can be found and assessed via available discovery mechanisms
- Services should allow their descriptions to be discovered and understood by humans and service requestors that may be able to make use of their logic.

Discoverability



- *"Services are supplemented with communicative meta data by which they can be effectively discovered and interpreted."*
- Service contracts contain appropriate meta data for discovery which also communicates purpose and capabilities to humans
- Store meta data in a service registry or profile documents

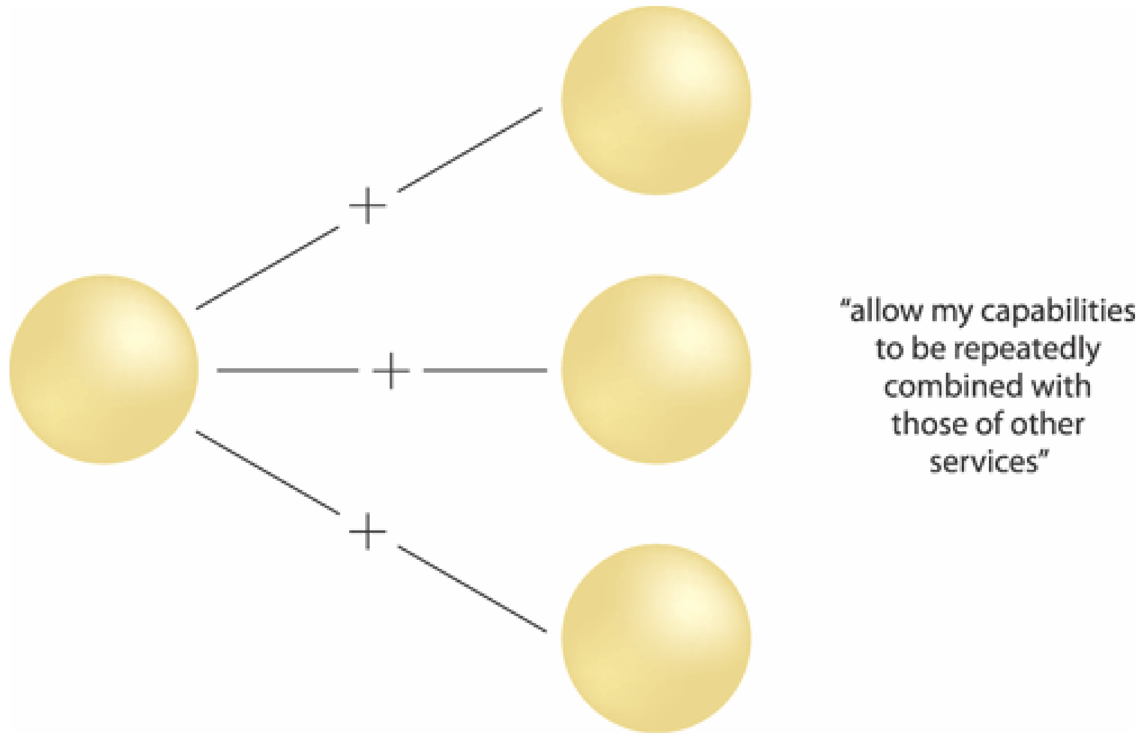
Source: Thomas Erl

Composability

- Services are composable
- Collections of services can be coordinated and assembled to form composite services
- Services may compose other services. This allows logic to be represented at different levels of granularity and promotes reusability and the creation of abstraction layers.

Composability

- *"Services are effective composition participants, regardless of the size and complexity of the composition."*
- Ensures services are able to participate in multiple compositions to solve multiple larger problems
- Related to Reusability principle
- Service execution should be efficient in that individual processing should be highly tuned
- Flexible service contracts to allow different types of data exchange requirements for similar functions


"allow my capabilities to be repeatedly combined with those of other services"

Source: Thomas Erl

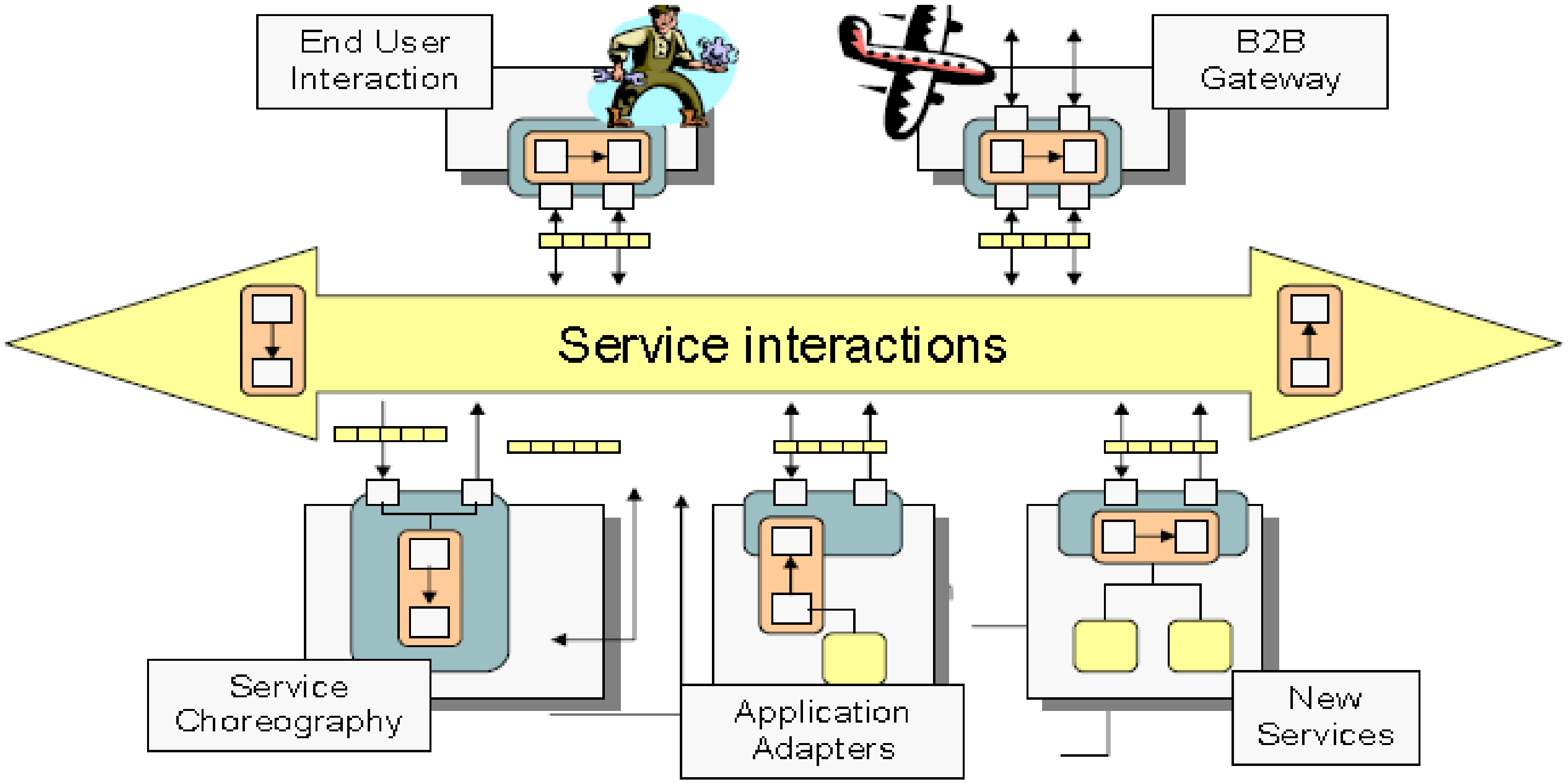
How do you apply these principles?

You can work towards this goal in a number of ways:

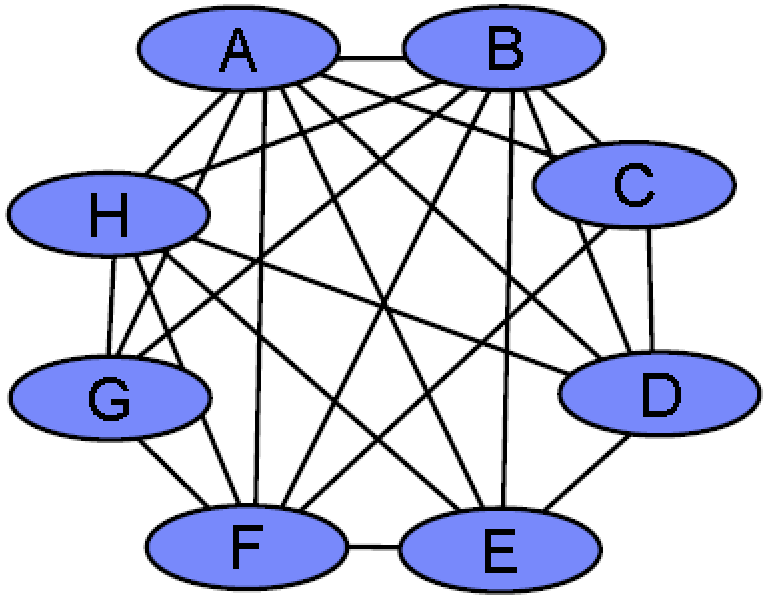
- Declarative techniques
  - Provide separation of concerns between the application logic and the configuration of middleware
- Code generation
  - WSDL code generation can hide the complexities of SOAP, HTTP and JMS from the developer
- Tooling
  - Can help reduce complexity for the developer
- Model-driven development
  - Exploiting both tooling and code generation capabilities to simplify development

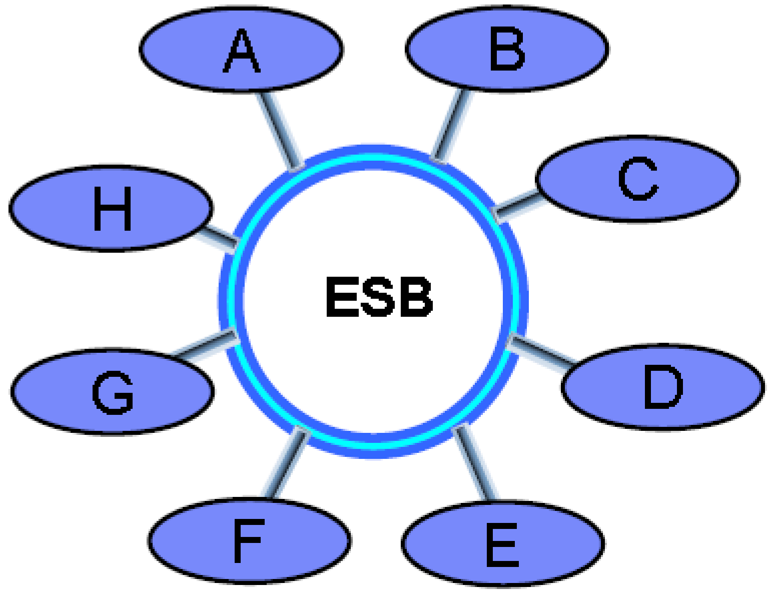
Loose coupling is enabled by an ESB

- An ESB (enterprise service bus) is an intermediary-oriented approach



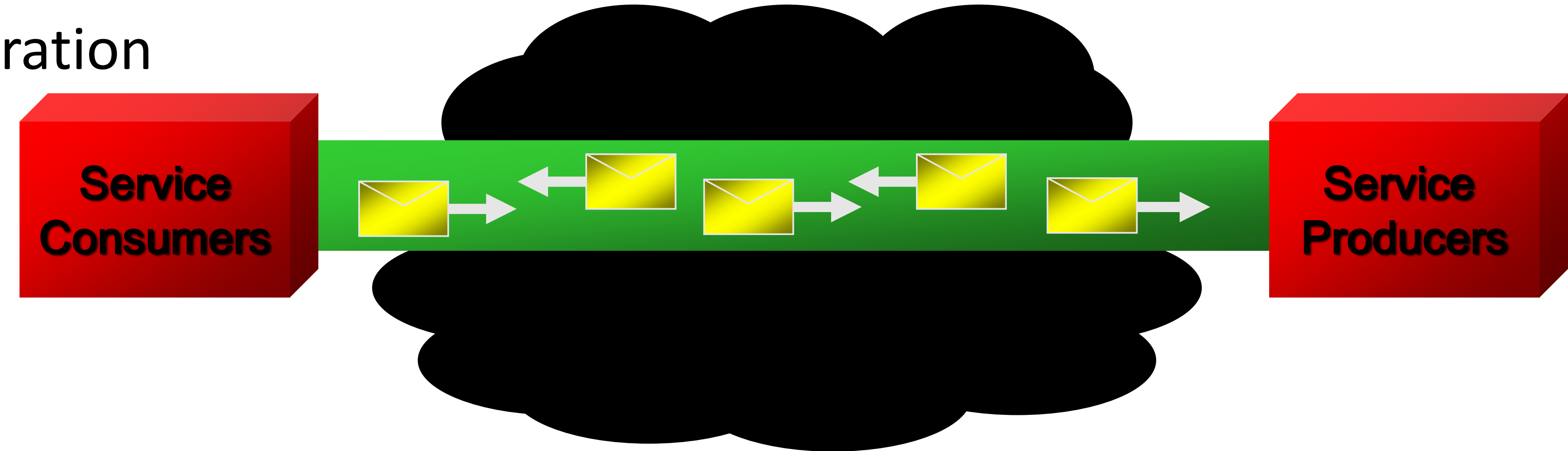
Point-to-point service interactions





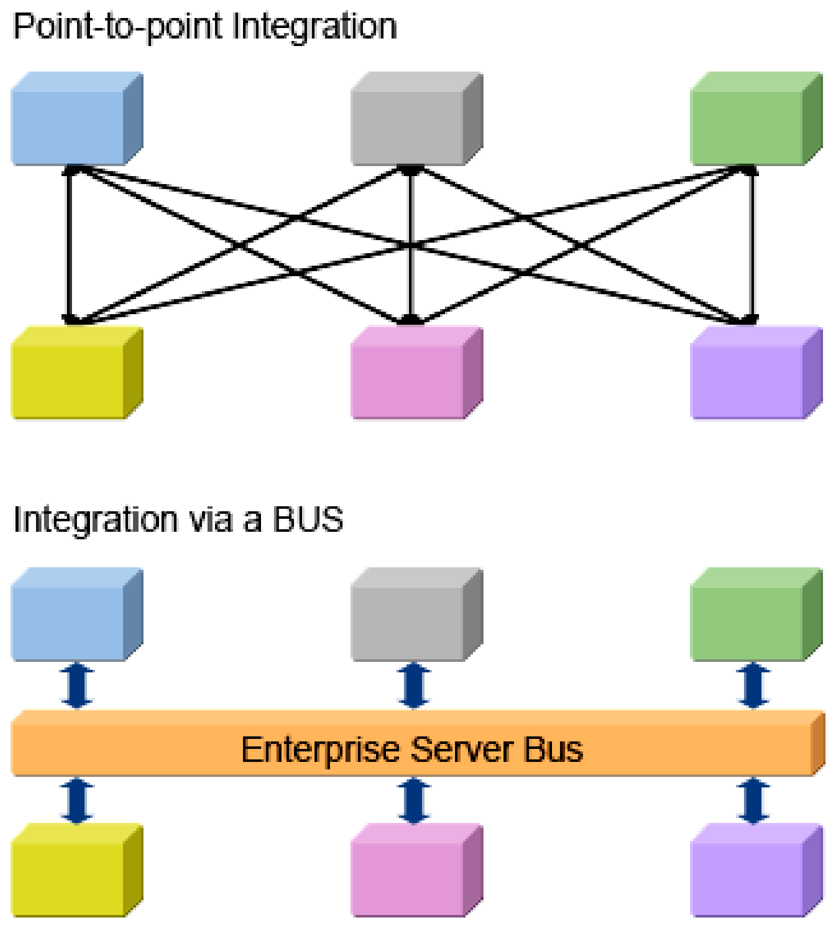
Service Communication

- Communicate with messages
- No knowledge about partner
- Likely heterogeneous
- Providing reliability and security to messages
- Sending messages across consumers and producers
- Service Orchestration

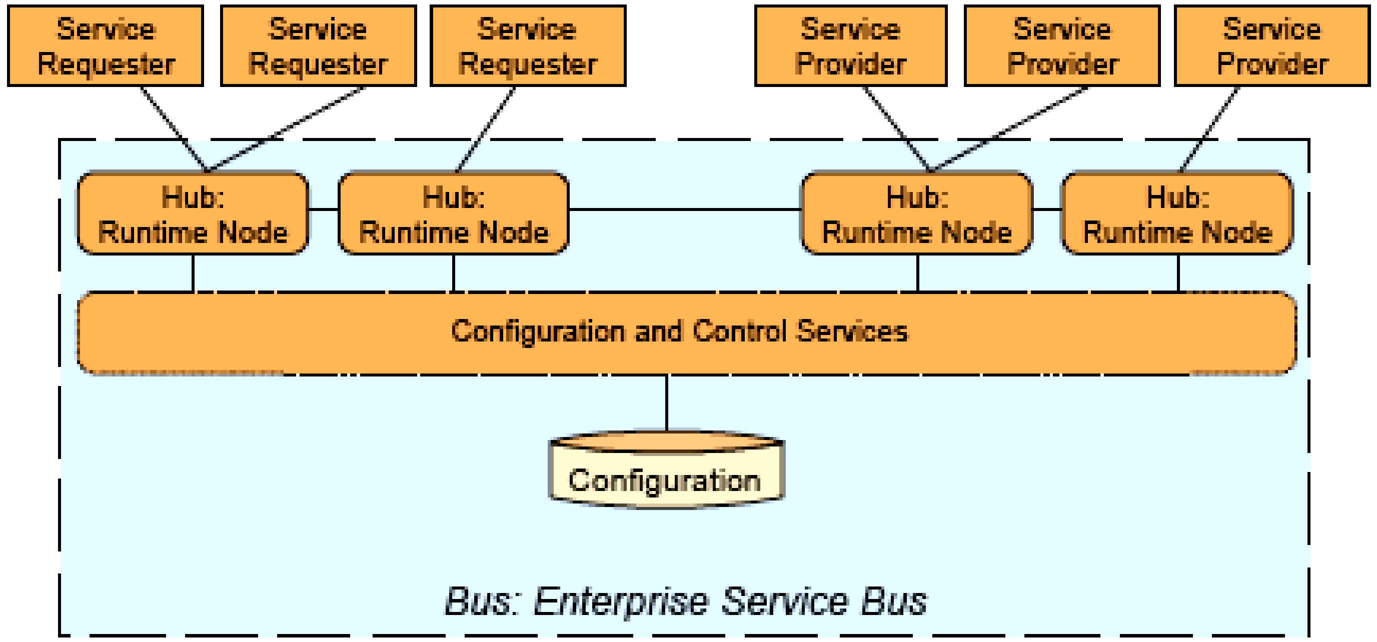


Enterprise Service Bus (ESB)

Overcoming the Problems of Point-to-point Integration with ESB

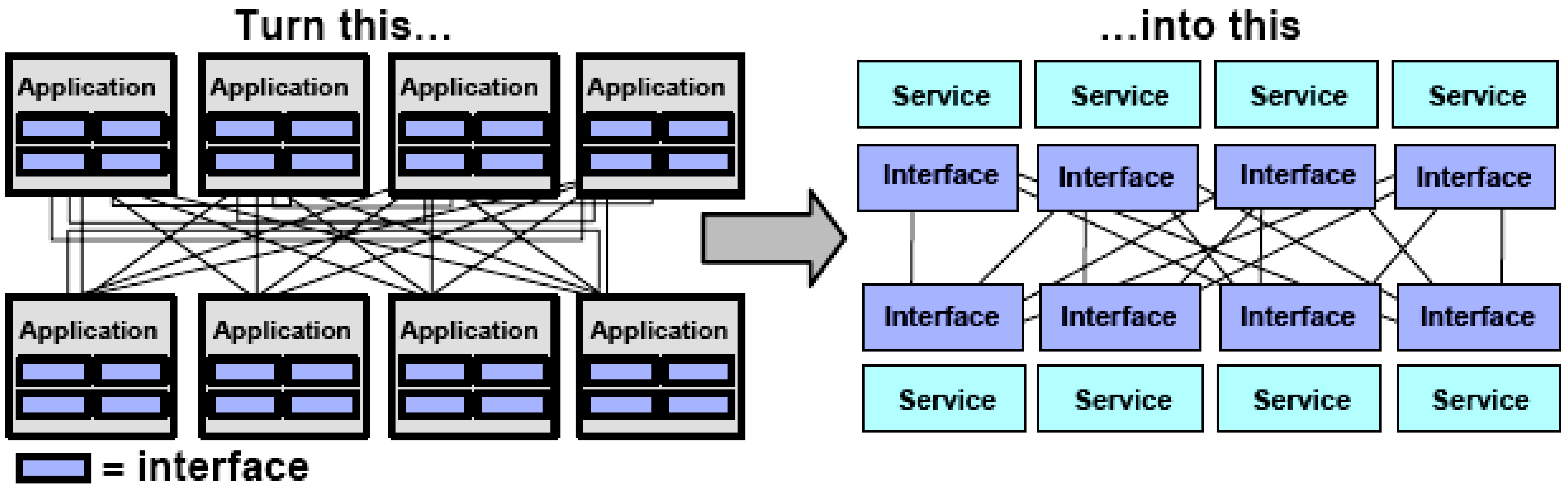


ESB as a Distributed Infrastructure with Centralized Control



SOA without ESB

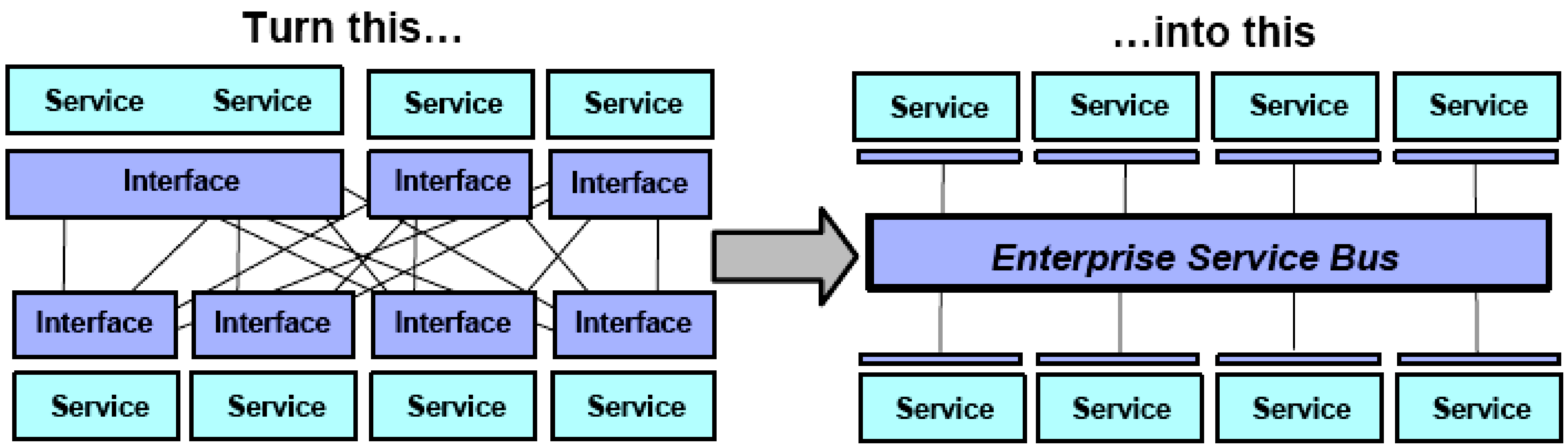
- *Decouples interfaces from applications*
- Separate connection points still leave bloated interfaces



- ✓ Rich business abstractions describe the application interface
- ✓ Decouples the interfaces from the business applications
- ✓ The number and complexity of the interfaces is reduced
- ✓ Business applications and their interfaces become reusable

SOA with ESB

- Shrinks the interfaces further



- ✓ Decouples the point-to-point connections from the interfaces
- ✓ Allows for dynamic selection, substitution, and matching
- ✓ Enables more flexible coupling and decoupling of the applications
- ✓ Enables you to find both the applications and the interfaces for re-use

The ESB Architecture

- The purpose of the ESB is so that common specifications, policies, etc can be made at the bus level, rather than for each individual service.

Module 5 Summary

- Architectural patterns of SOA
- Designing distributed systems with SOA