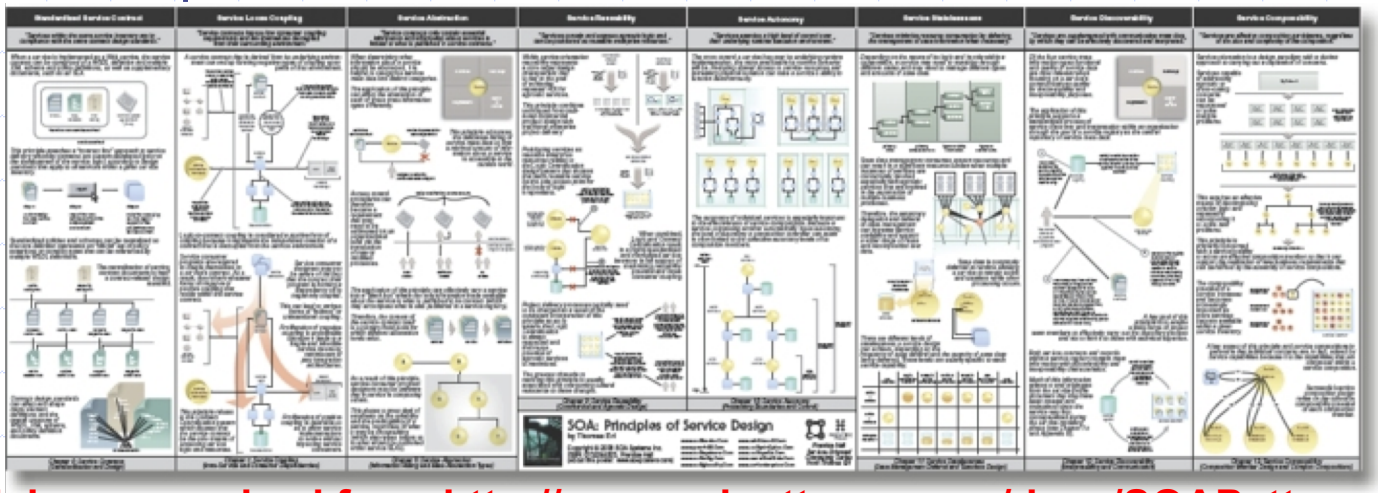
A Service Oriented Architecture (SOA) Primer

SOA as an Architectural Style

- **SOA = Service Oriented Architectures:** Why the "A" in SOA represents an Architecture Style versus an Architecture Pattern or something else
- Architecture Style = components and connectors that are used together with a collection of constraints
- Architecture style provides the foundation for a collection of solution patterns



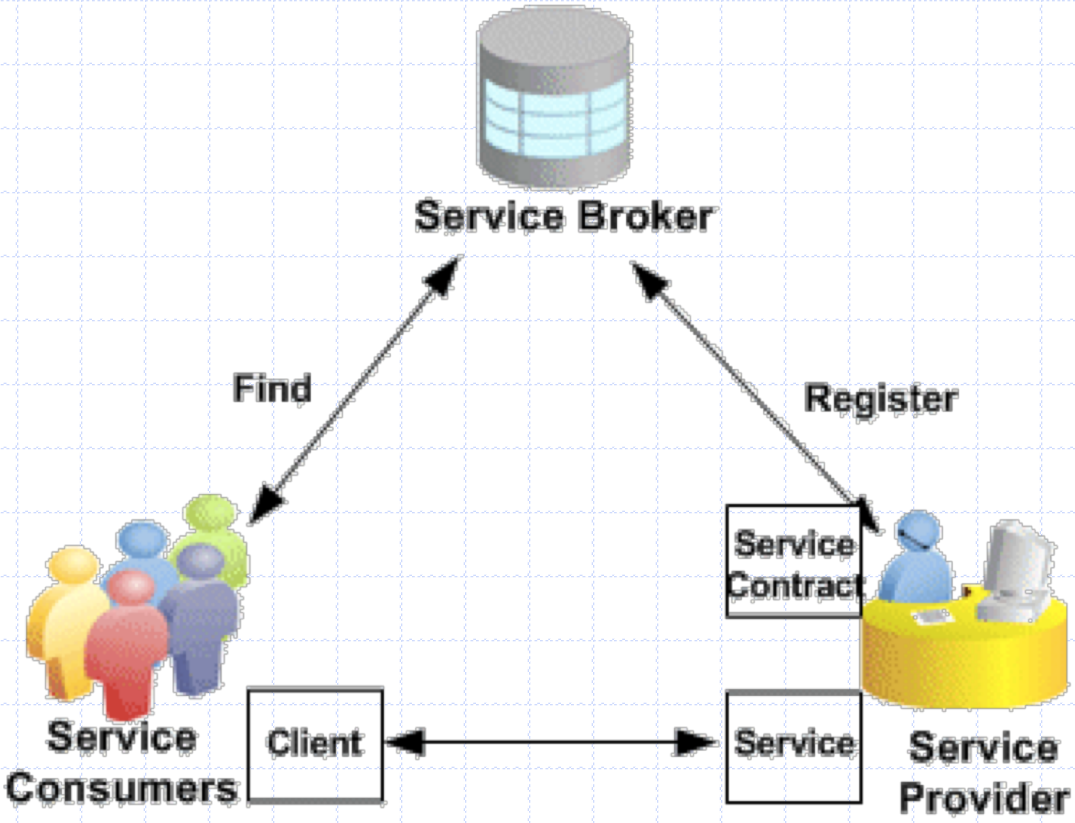
Materials summarized from http://www.eaipatterns.com/docs/SOAPatterns.pdf

When do we use a SOA Style?

- **Context:** A number of services are offered (and described) by service providers and consumed by service consumers. Service consumers need to be able to understand and use these services without any detailed knowledge of their implementation.
- **Problem**: How can we support interoperability of distributed components running on different platforms and written in different implementation languages, provided by different organizations, and distributed across the Internet?
- **Solution**: The service-oriented architecture (SOA) pattern describes a collection of distributed components that provide and/or consume services.

From "Software Architecture In Practice" Len Bass, Paul Clements, Rick Kazman

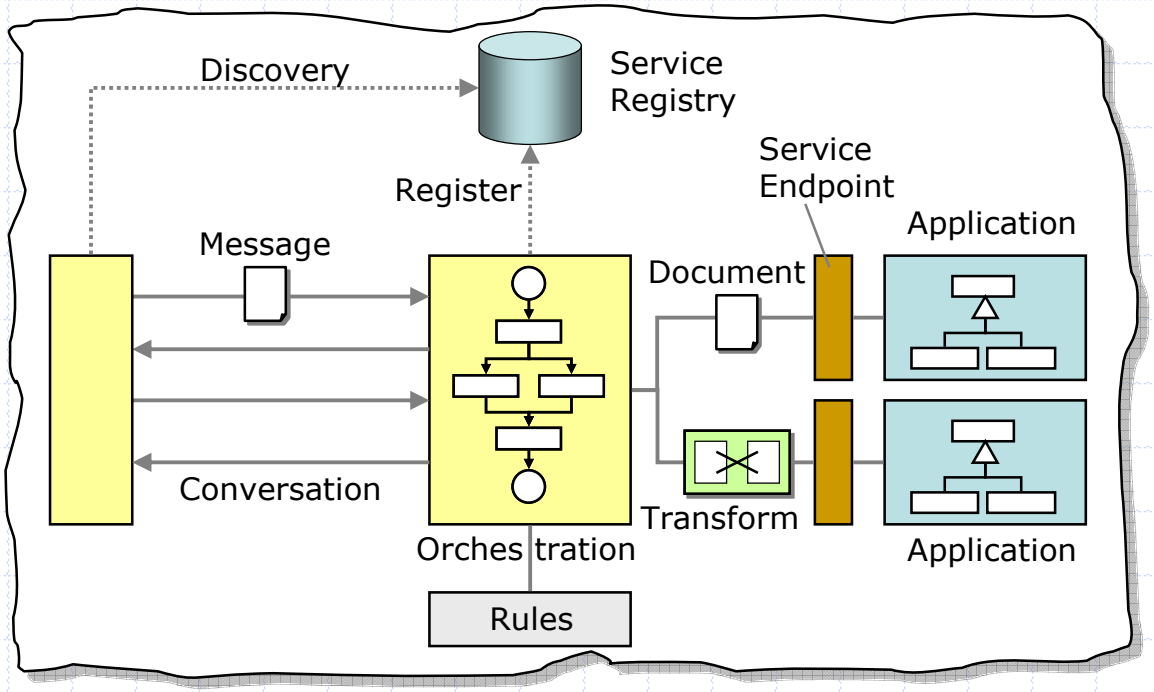
What does a "traditional" service oriented architecture look like?



Why SOA?

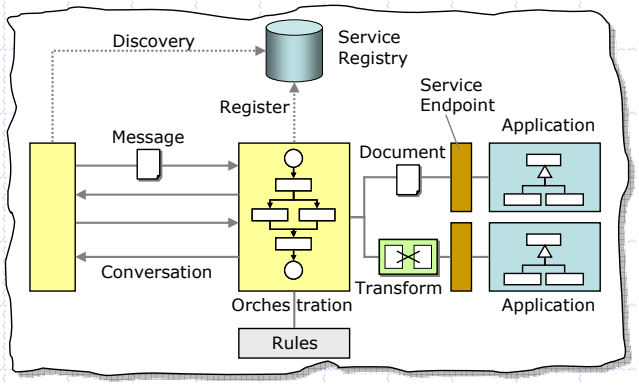
- Modern application architectures are almost **always distributed**, we desire an architecture approach to construct applications using a collection of autonomous services.
- Focus of design is on integration – individual services could be developed and deployed using different technology stacks and platforms. Integration patterns are selected that use messaging
- Desire to reuse well known OO principles such as encapsulation, abstraction, and well-defined interfaces in creating new architectures based on SOA
- Availability and stability of services influence design decisions – separate core aspects of building the service (not likely to change) from the service configuration (likely to change)
- The desire to easily integrate capabilities built and hosted by others into our own applications

What does a typical SOA inspired solution look like?



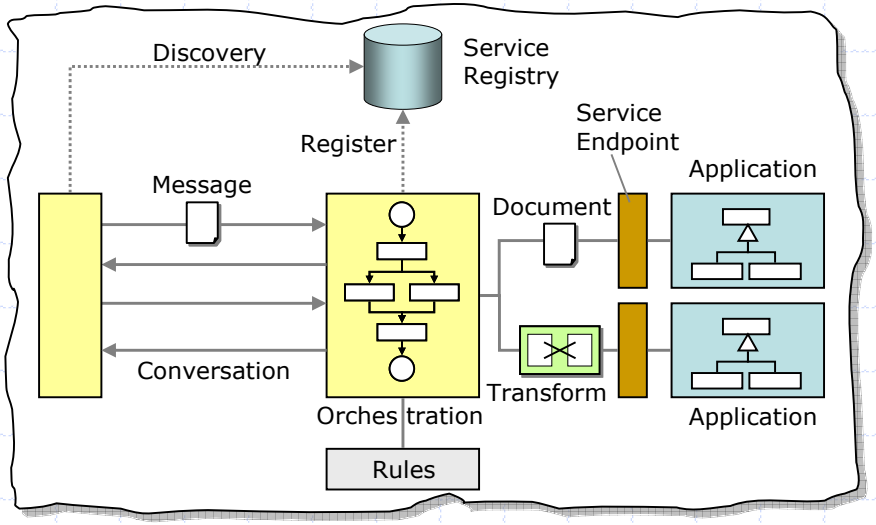
What design models does a SOA promote?

- Composition
  - Robust structures are built up from granular components
- Process-Orchestration
  - Orchestration logic is used to coordinate the interaction between multiple services



- Declarative
  - Express the logic of computation without hard-coding the control flow
  - Focus is on designing "what" should be done within the realm of the problem domain, versus "how" to go about accomplishing it
- Event-Driven
  - Message-based communication is used to trigger actions based on interesting events
  - SOA based systems trigger and react to events versus hard coding what happens and in what order it happens

Almost too good to be true

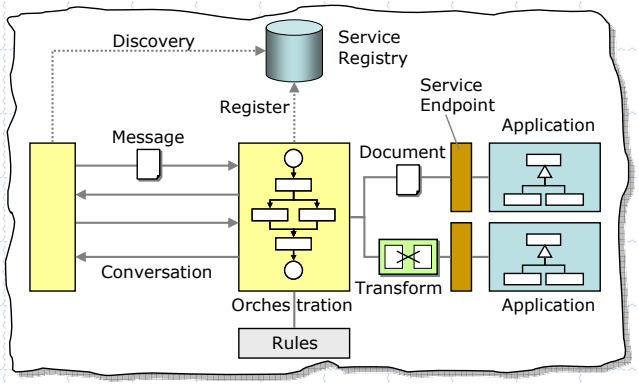


- Doesn't account for latest thinking around Smart endpoints and dumb pipes versus "smart pipes"
  - https://medium.com/@nathankpeck/microservice-principles-smart-endpoints-and-dumb-pipes-5691d410700f
  - https://martinfowler.com/articles/microservices.html
- Hard to scale
- Hard to make resilient
- Hard to make performant
- Hard to evolve
- Fragile to change

The good and bad parts of traditional SOA

The good parts

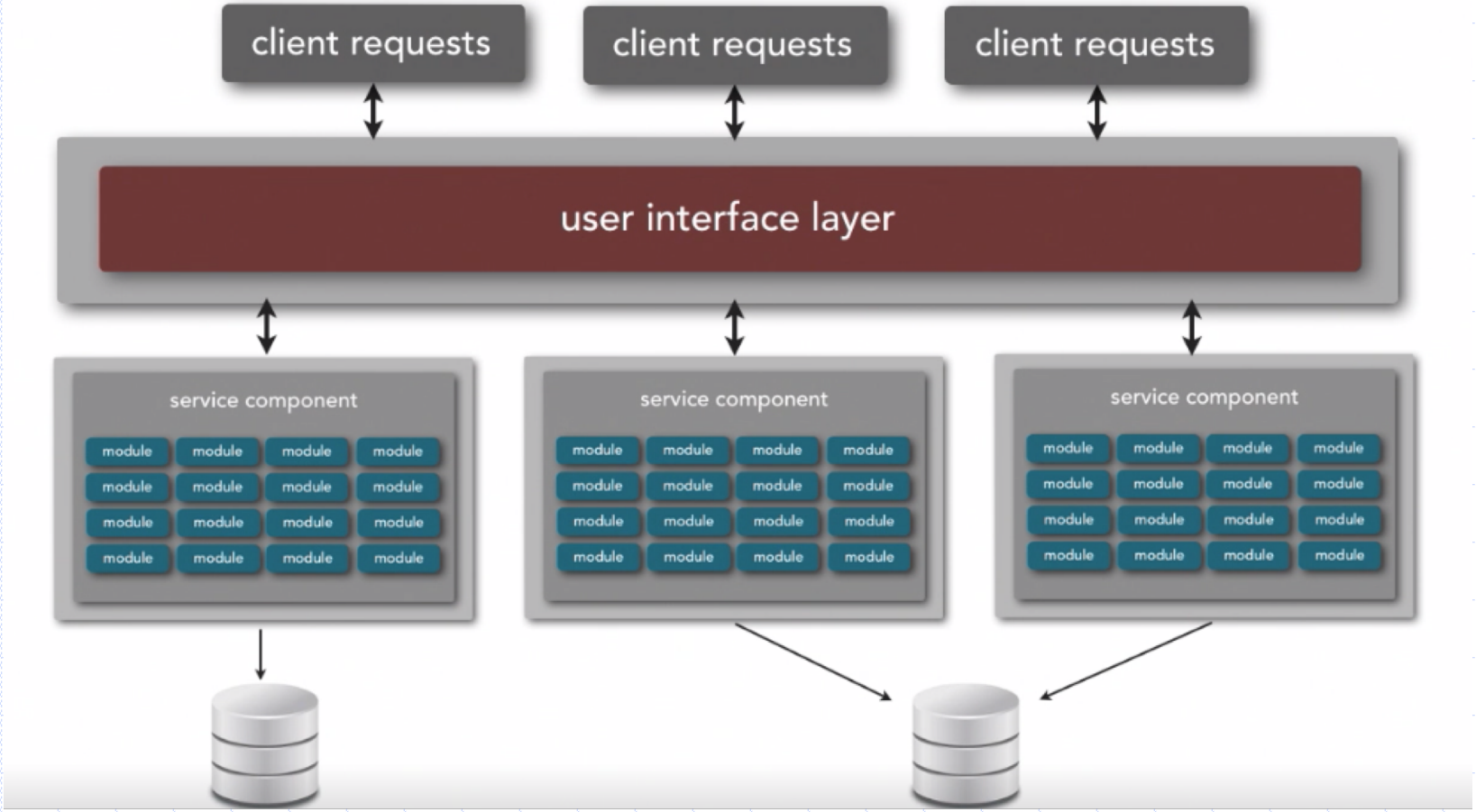
- **Composition**
  - Robust structures are built up from granular components
- **Process-Orchestration**
  - Orchestration logic is used to coordinate the interaction between multiple services
- **Declarative**
  - Express the logic of computation without hard-coding the control flow
  - Focus is on designing "what" should be done within the realm of the problem domain, versus "how" to go about accomplishing it
- **Event-Driven**
  - Message-based communication is used to trigger actions based on interesting events
  - SOA based systems trigger and react to events versus hard coding what happens and in what order it happens



The not so good parts

Service Based Style

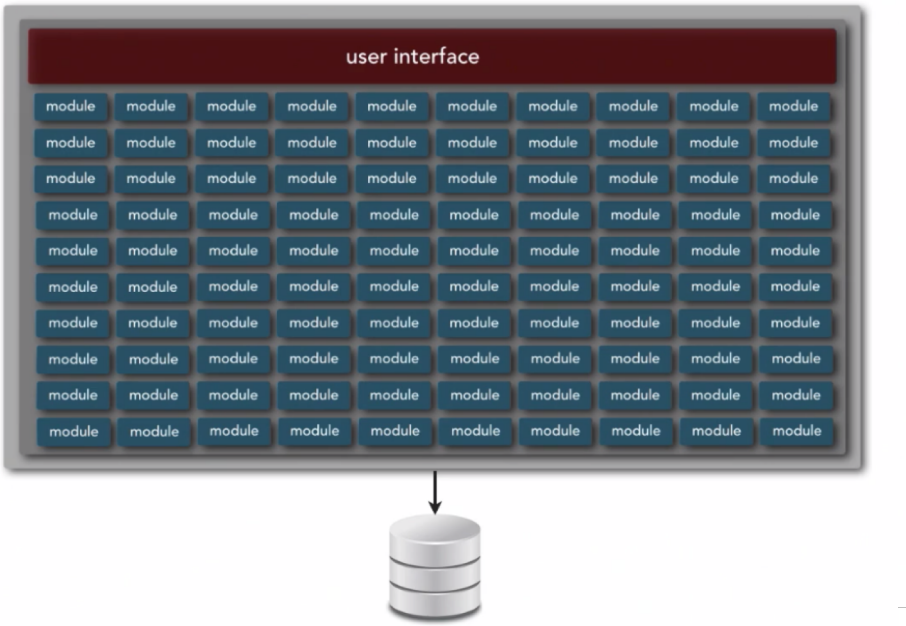
Ref: Neil Ford & Mark Richards
Software Architecture Fundamentals

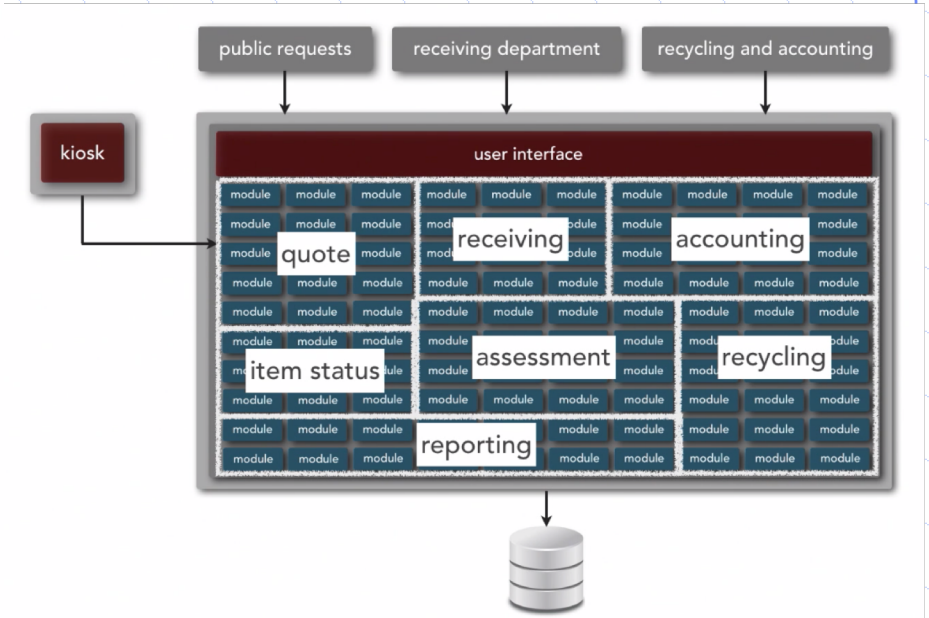


Think of SOA, "the good parts"

Service Based Style - Example

Ref: Neil Ford & Mark Richards
Software Architecture Fundamentals





Generally Good To Decompose a Monolithic Application

Tenets of SOA – Boundaries are Explicit

- A boundary represents the border between the services interface and its internal private representation
- Interaction via messages is used to cross well-defined boundaries
- A boundary is used to abstract certain factors such as geography, trust, or execution complexity
- A SOA-based design needs to account for the opaque aspects of crossing boundaries
  - The physical location of the service is unknown and should not matter
  - Security models and policies are enforced as boundaries are crossed
  - Marshalling and casting data structures from the services public interface to private internal representations might require additional resources
  - The service consumer is likely to have no knowledge of the internal service implementation and therefore has no control over certain service execution aspects such as performance
  - The distributed nature of crossing service boundaries introduces multiple opportunities for errors, thus error processing must be robust

Tenets of SOA – Services are Autonomous

- Services are entities that can be independent deployed, versioned and managed
- Services are dynamically addressable through a well known URI
- The physical location of the service can change with the expectation of not impacting the consumer
- If the service relies on another service, it should not use the service client to bridge this dependency
- Both service consumers and service providers should take a pessimistic view of performance and the behavior of the other party
  - Service consumers should be able to time-out and appropriately deal with a slow response times
  - Service providers should assume that service consumers might misuse the service (accidently or maliciously) and be hardened to deal with these issues

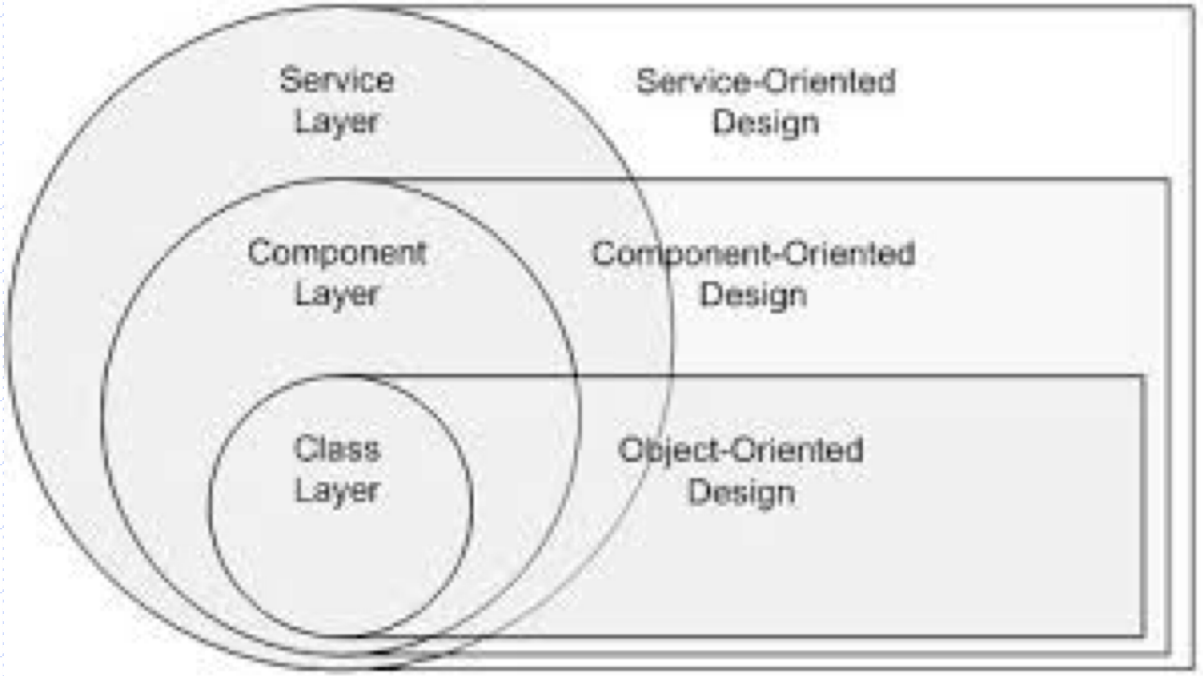
Tenets of SOA – Services Share Schema and Contract, not Class

- Services interaction should be based solely on the services interface contract (versions, discovery, policies, message-formats/schemas, behaviors)
- Implementation details of a service (the class structure used to construct the service) are independent from the interface used to invoke the service
- The key to designing good services, is designing good service interfaces
  - They are hard to change – may break existing service consumers
  - They can introduce performance challenges – transforming between the service interface and native interface can be expensive
  - Consumers must not only know the structural aspects of the interface but also the policies that govern the service implementation

Tenets of SOA – Service Compatibility is based on Policy

- Service interfaces define the structural contract for interfacing with a service, policies define the semantic aspects of interacting with a service
- Policies govern the behavior and expectations of a particular service, example:
  - Before a new user is created, the ID must be unique
  - Services supporting external users should be given more priority than services supporting internal applications
  - A secure service must adhere to a set of required security policies (e.g., OAuth, mutual auth, etc)
  - Messages received by a service must be encrypted in a certain way
  - Messages can be accepted over a queue or HTTP

Designing for SOA – build upon previous knowledge in OOD and COD



Characteristics of the SOA Style

Characteristic	Traditional Architectures	SOA Architectures
Design & Implementation	• Function Oriented • Designed to Last • Long development cycles • Integration with other applications difficult	• Coordination Oriented • Built for Change • Build and deployed incrementally • Ease of integration with other applications
Physical System	• Application boundaries intrinsic to system • Tightly coupled components • Object- or Component-oriented interactions	• Enterprise solutions • Loosely coupled • Semantic message-oriented interactions

SOA-based architectures have been around for a long time and have matured over the years. While once an approach to manage design and integration complexity - SOA-based architectures have matured to the point where many companies are starting to offer service-only solutions (APIs) leaving the build of interesting applications to others. See: https://apigee.com/console/others

Ingredients needed to create applications using the SOA style

- **A set of services** that is thought to be of value to customers, partners, or other areas of an organization
- **An architectural style** that requires a service provider, mediation, and service requestor with a service description
- **A set of architectural principles, patterns and criteria** that address characteristics such as modularity, encapsulation, loose coupling, separation of concerns, reuse and composability
- **A programming model** complete with standards, tools and technologies that supports web services, REST services or other kinds of services
- **A middleware solution** optimized for service assembly, orchestration, monitoring, and management

Adopted from: http://www-01.ibm.com/software/solutions/soa/

Service Oriented Architecture is an Example of an Architectural Style

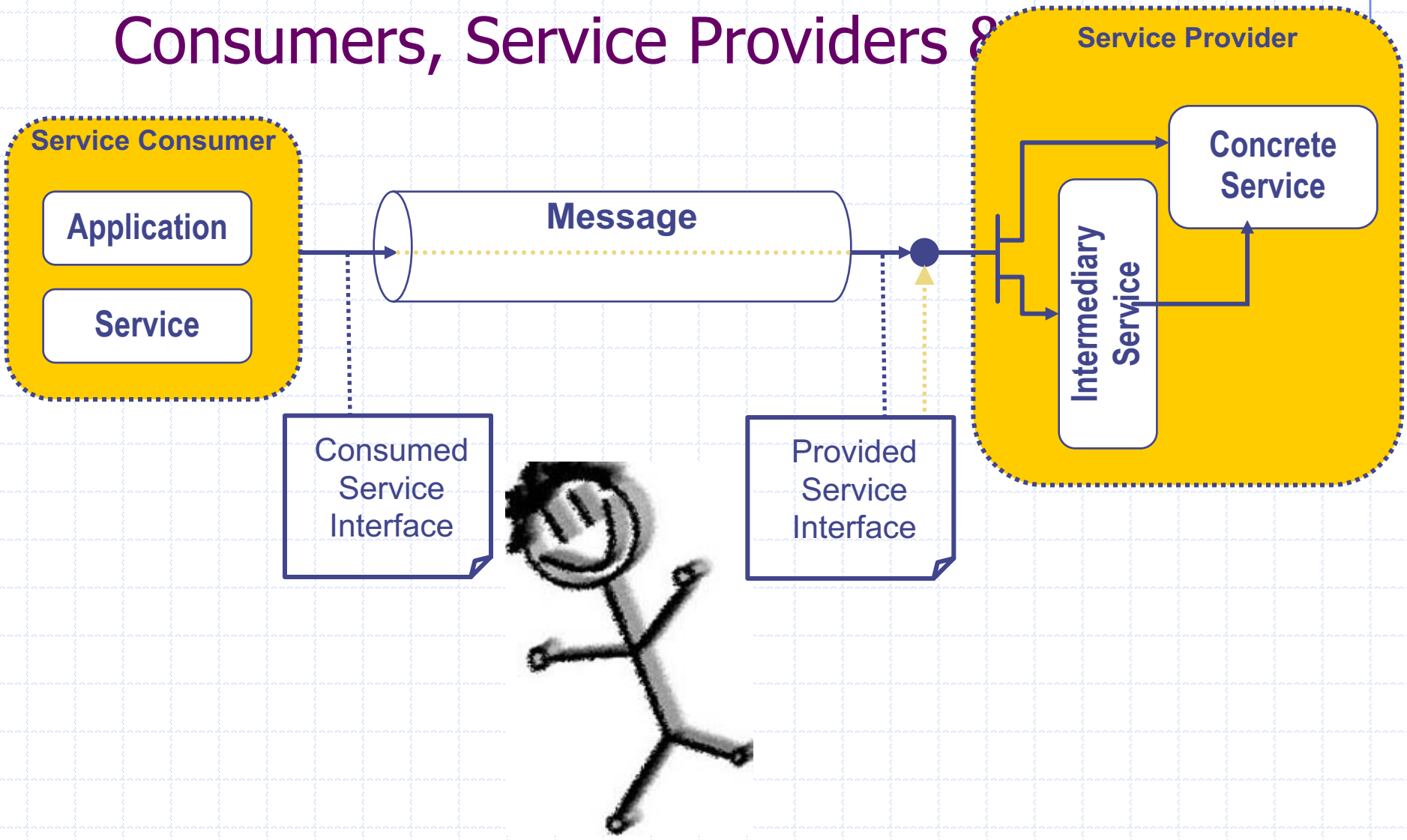
- An **Architectural Style** defines a family of systems in terms of a pattern of structural organization.
  - What are the architectural components?
  - What are the architectural connectors?
  - What patterns guide the design of the components and connectors?
  - How are faults and unexpected events handled?
  - Clear definition of the set of constraints on the architectural components and the relationships that are allowed between them

Because SOA is an Architectural Style a Reference Architecture can be constructed to govern common aspects of all applications built in accordance with this style… see: http://www.opengroup.org/soa/source-book/soa_refarch/intro.htm

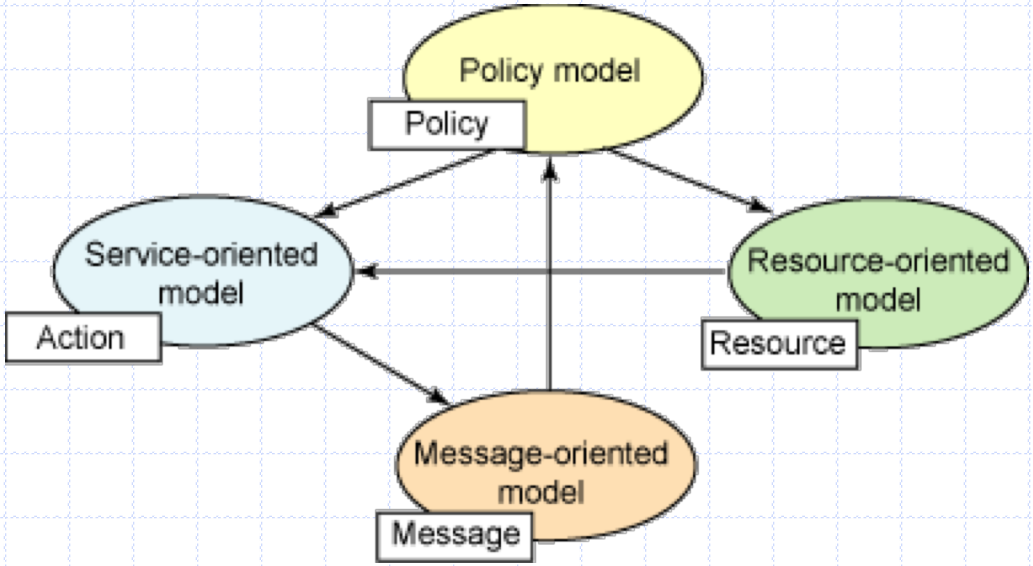
Service Oriented Architecture is an Example of an Architectural Style

- SOA as an Architectural Style:
  - What are the architectural components?
    - Services
  - What are the architectural connectors?
    - Messages
  - What patterns govern the design of the components and connectors?
    - Data Services, Business Services, Composite Services
  - How are faults and unexpected events handled?
    - Language specific exception handling mapped to service faults (SOAP)
    - HTTP Error Codes – 400 Series (REST)
  - Clear definition of the set of constraints on the architectural components and the relationships that are allowed between them
    - Services are network addressable
    - Services are language and platform independent
    - Services have flexible instantiation capabilities
    - Services are stateless
    - Messages are formally defined by a service contract
    - …

The "Actors" in an SOA – Service Consumers, Service Providers &



The SOA Model from the W3C



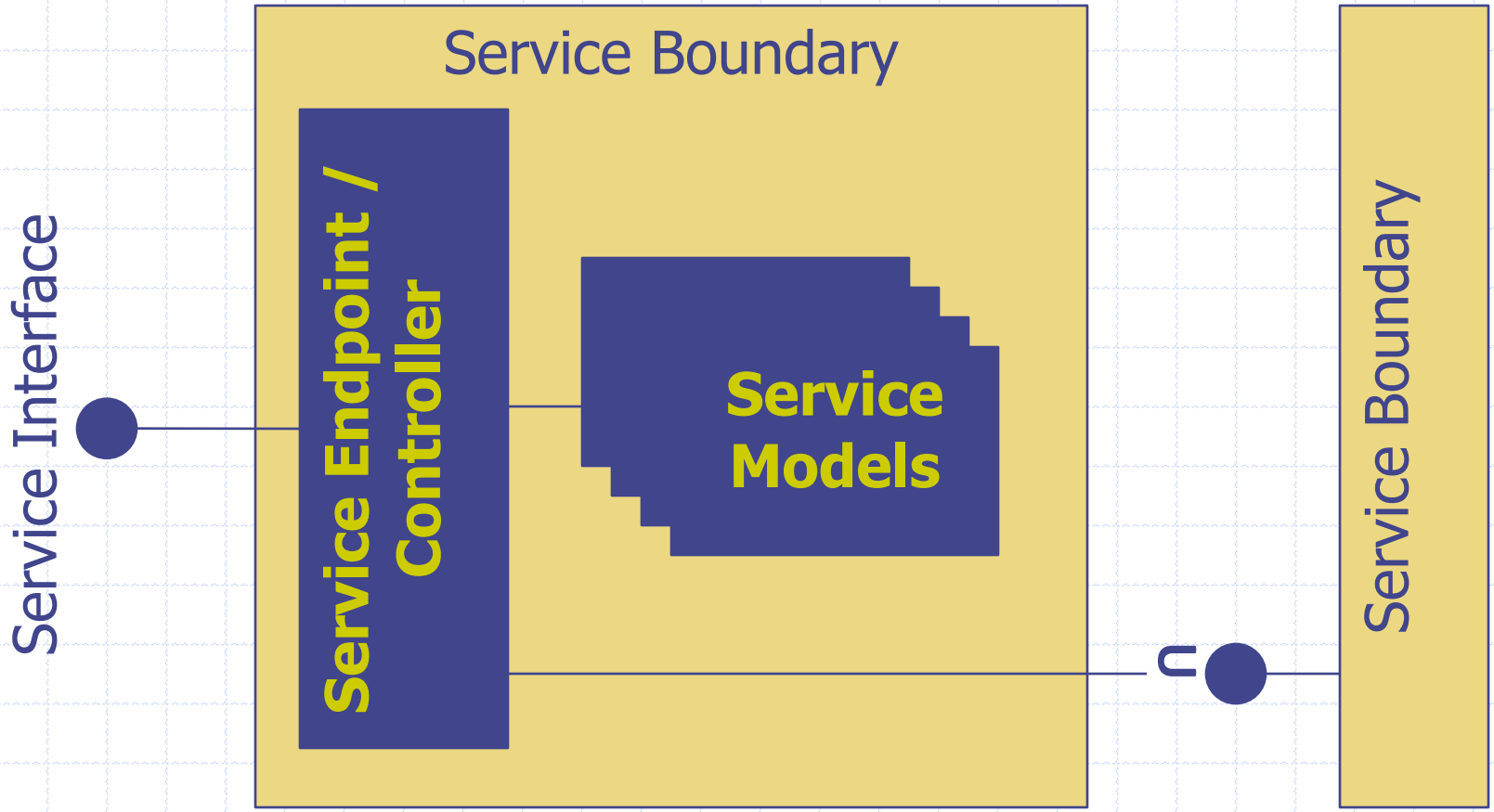
The **Service Model** is dependent on other models. The **Message Model** defines the messages that a service can accept (content, body, protocol, etc); the **Resource Model** defines the resources used to fulfill the service contract and are hidden from outside consumers of the service; the **Policy Model** defines constraints on allowable actions or states and is the enforcement point for security.

http://www.ibm.com/developerworks/architecture/library/ar-soastyle/

The Architectural Components - Services

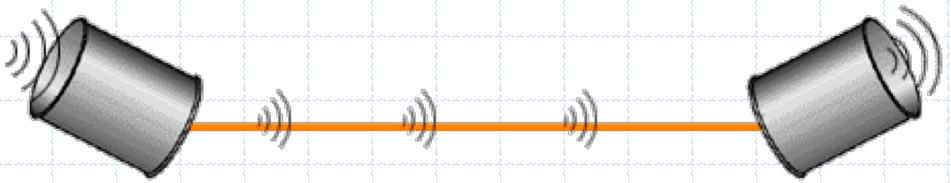
- Services are a conceptual **design component**, and can be implemented using a variety of different technologies
  - Java (via frameworks – Play, Spring, Restlet, etc)
  - Scala (via frameworks – Play, Spray, etc)
  - .Net
  - Javascript frameworks such as Node.js
  - Even Python, COBOL or just about any other language these days
  - Third party tools: Siebel, Salesforce
- Services are designed to have flexible interfaces and are evolved easily
  - Services separate the concerns of the service consumer from the service provider
- Services can be instantiated in a variety of different ways
  - Local components, Web Services, Sync-/Asynchronous Messages
- Services are lifecycle managed by an application server container of some sort
  - CICS, .Net Framework, Java (WebSphere, Tomcat), Node.js, Vert.x, etc.

The Architectural Components - Services



The Architectural Connectors - Messages

- Services interoperate with each other using “messages” capable of verifying and certifying their own syntax and semantics
- Architecture Requirements for Messages
  - Messages do not assume any sort of delivery technology
  - Messages support intermediaries and can be transformed between the service consumer and the service provider without either party being aware of the transformation process
  - Messages can be secured end-to-end
  - Messages can be deserialized into language-specific components
  - Language specific components can serialize themselves into a valid message that adheres to both the syntactic and semantic requirements of the message

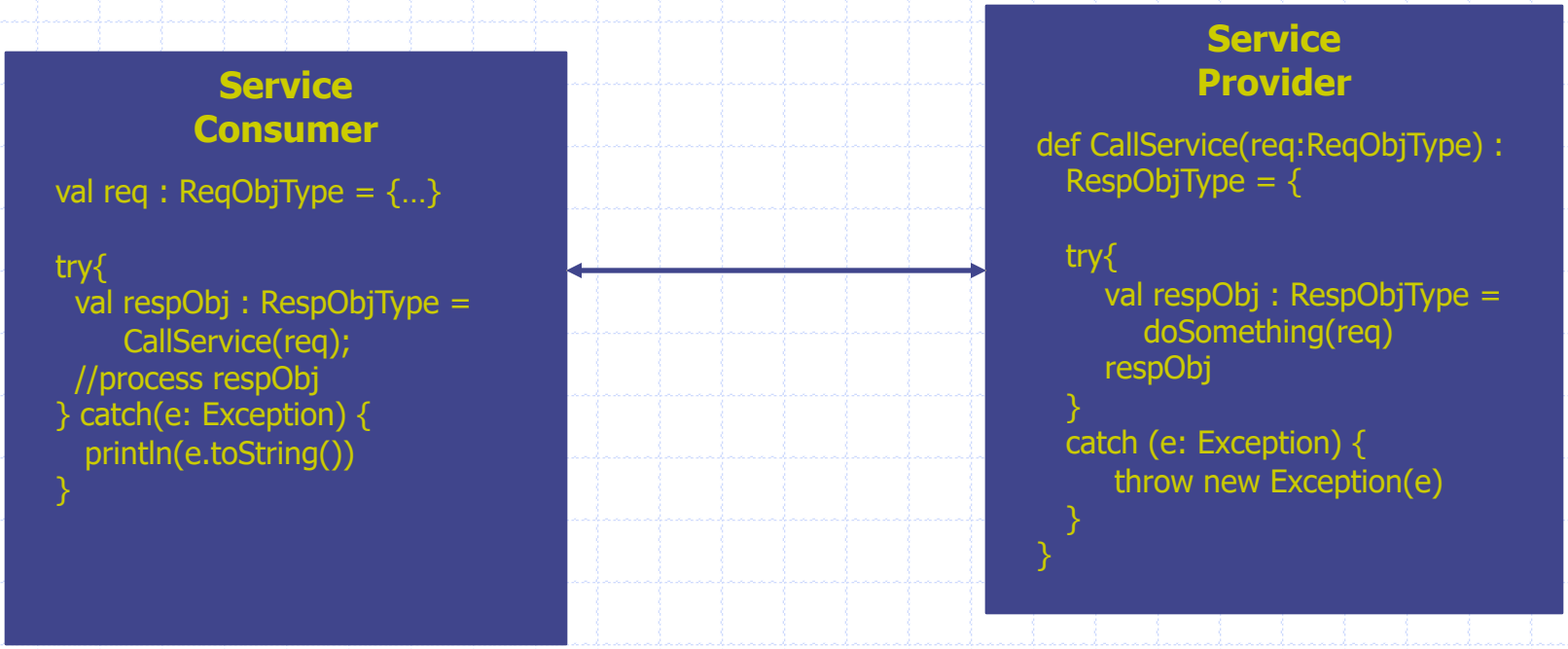


Service Integration Styles

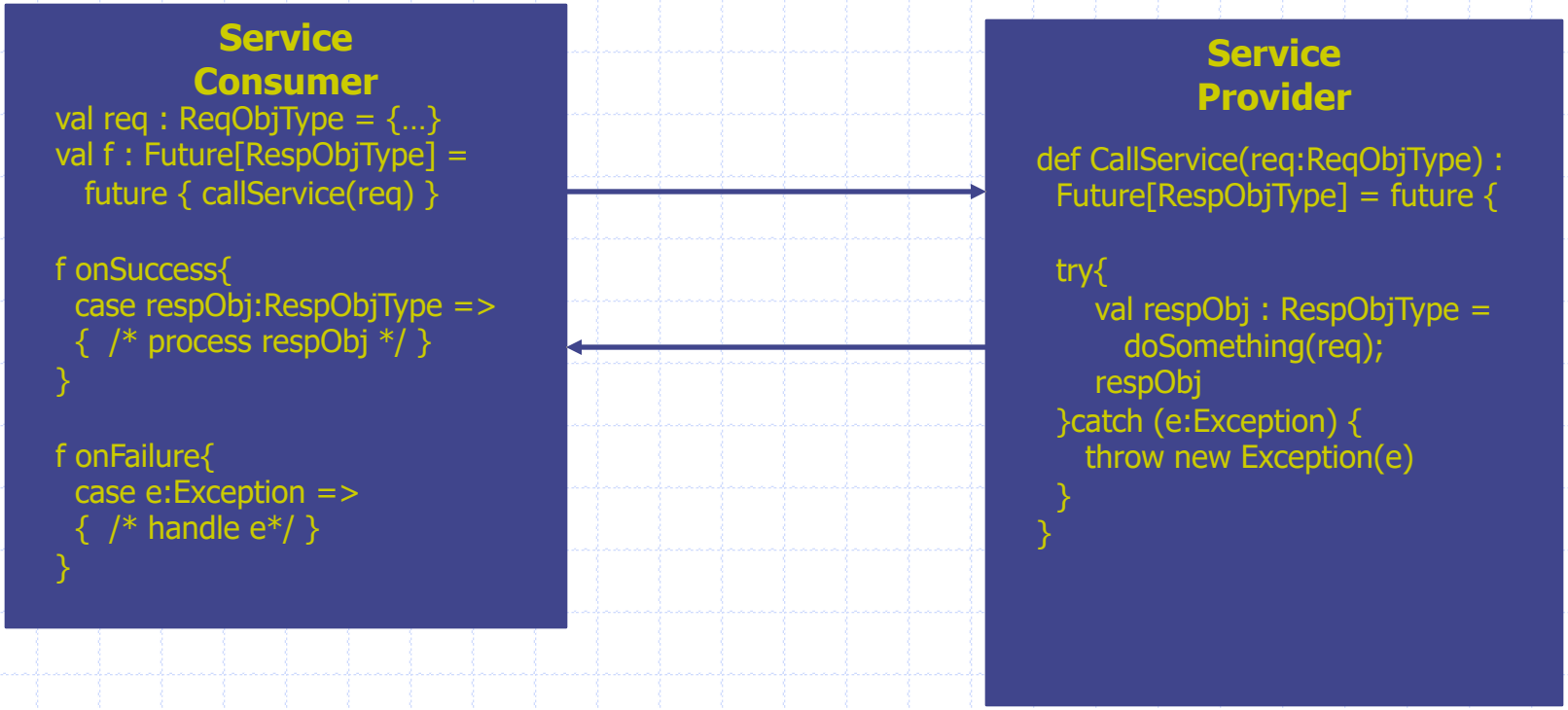
- Service consumers can be wired to service providers using both synchronous and asynchronous messaging
- Patterns can be request-reply or subscription based

	One	**Many**
Synchronous	Try[T]	Iterable[T]
Asynchronous	Future[T]	Observable[T]

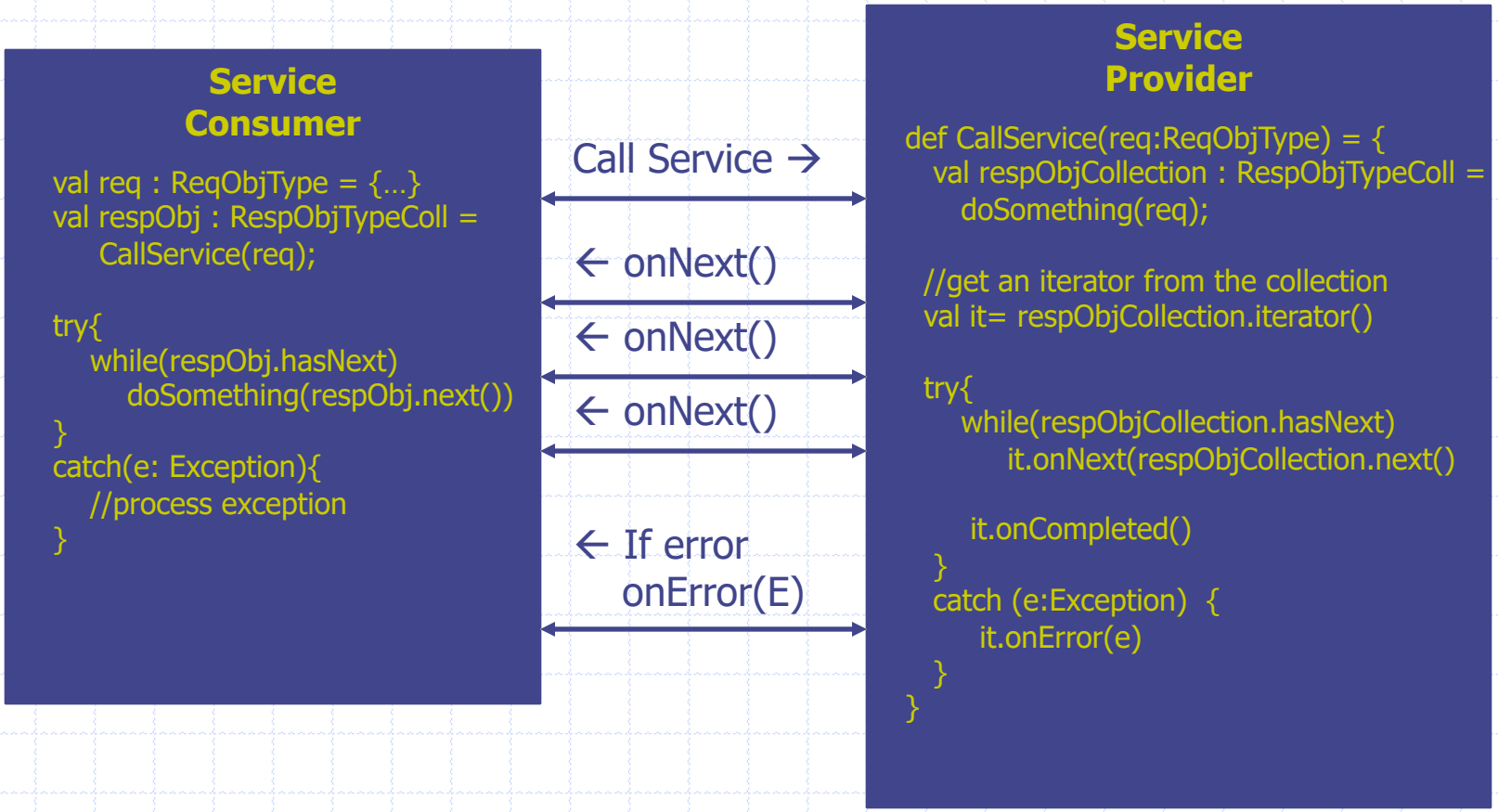
Example of a Synchronous-One Pattern



Example of an Asynchronous One Pattern

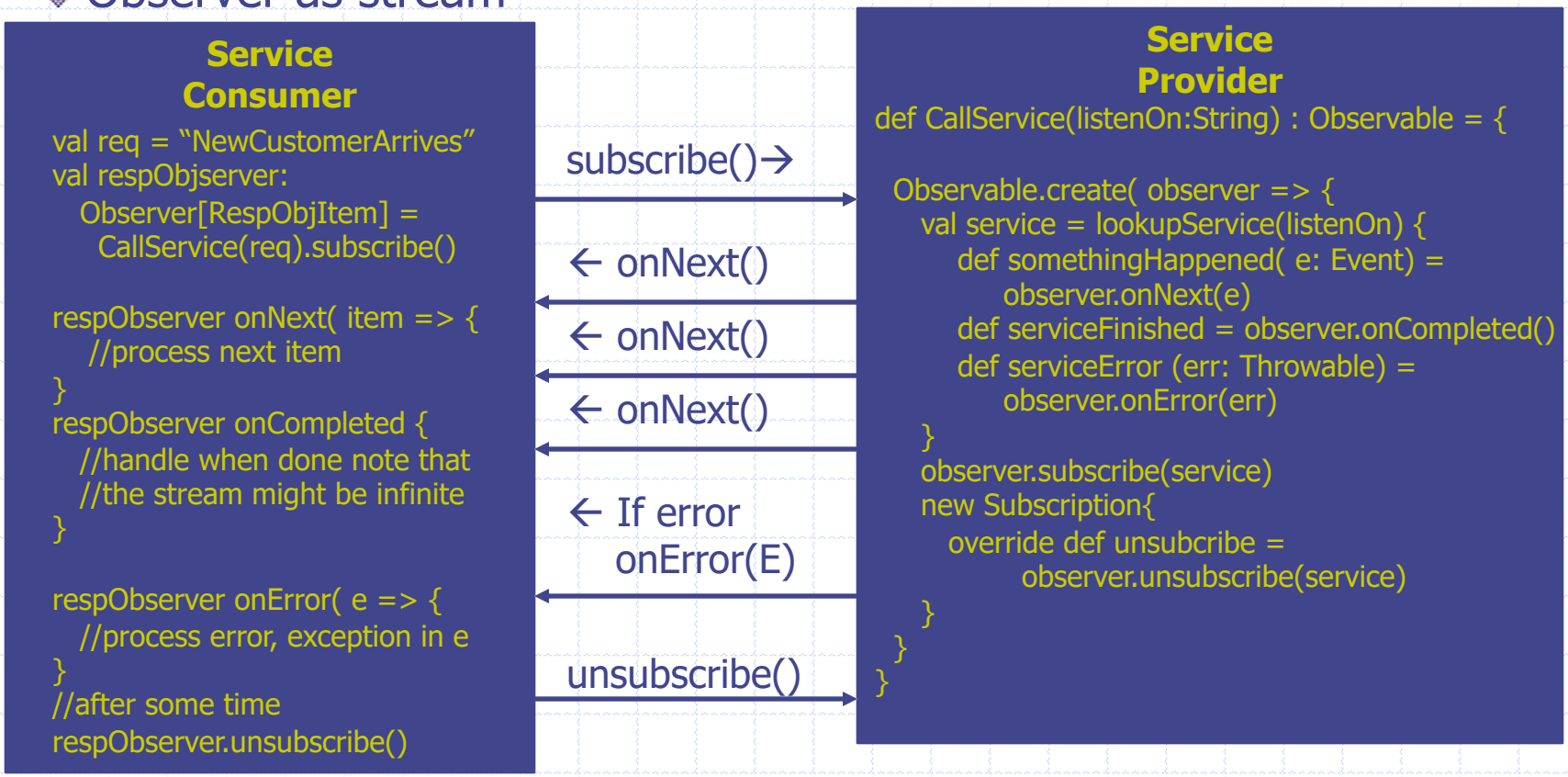


Example of the Synchronous Many (Iterable) Pattern



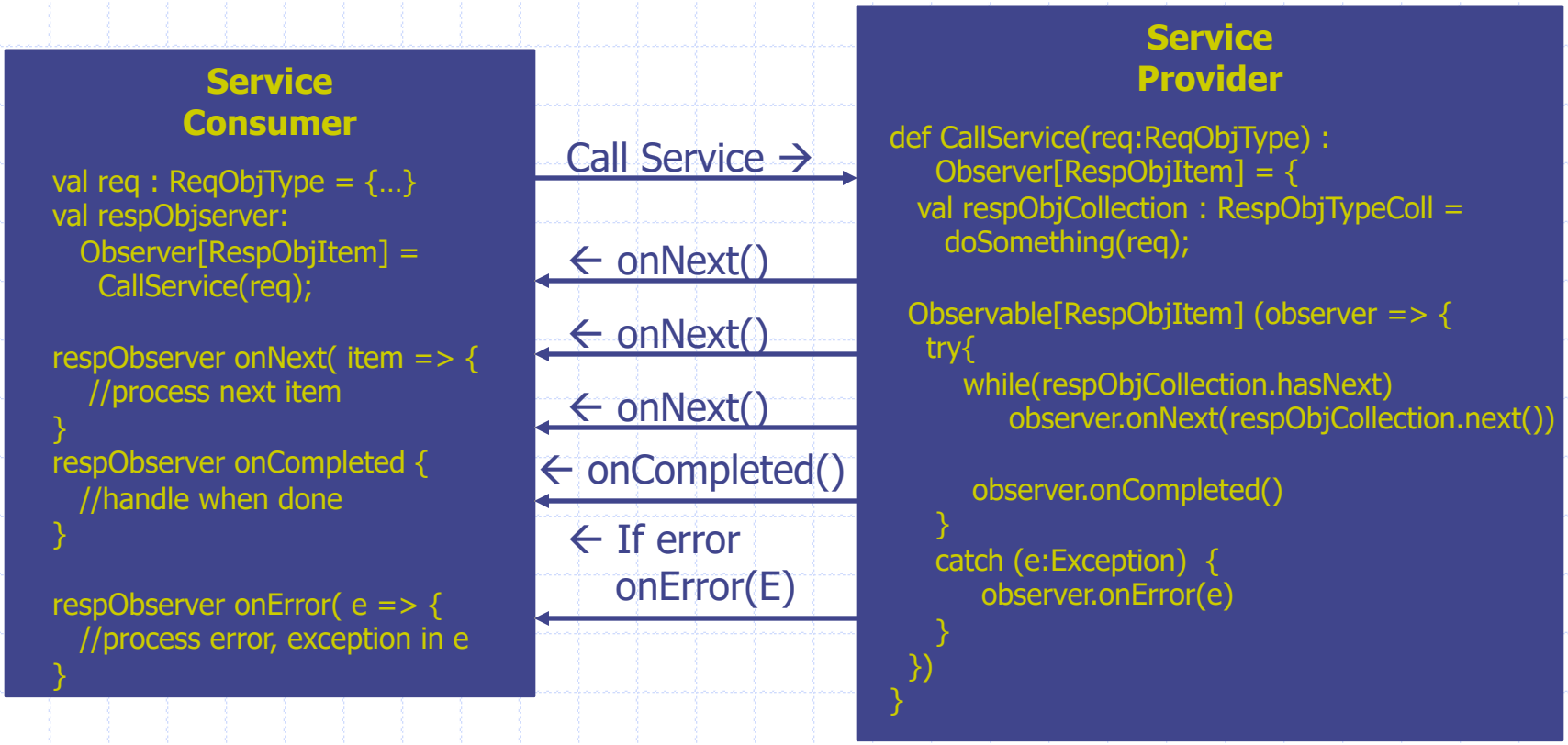
Example of the Asynchronous Many Pattern (Streams)

- Observer as stream



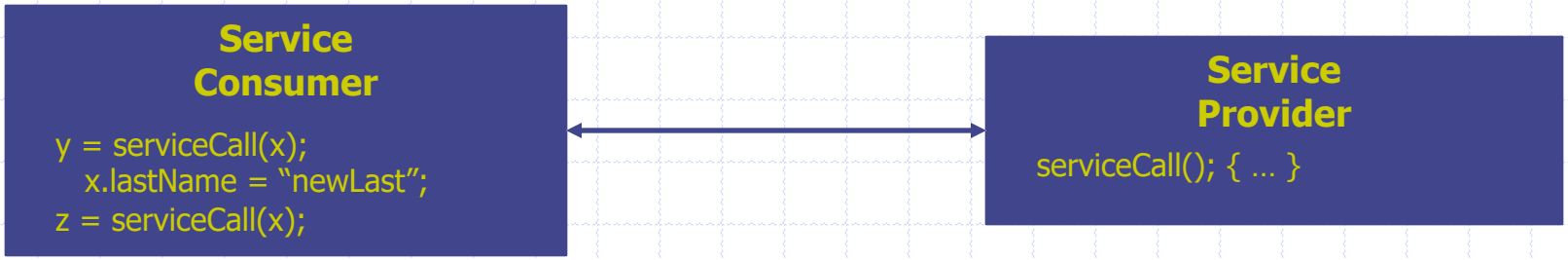
Note that this pattern can also apply to subscribing to server streams that are unbounded in length

Example showing how an Interable can be converted to an Observable

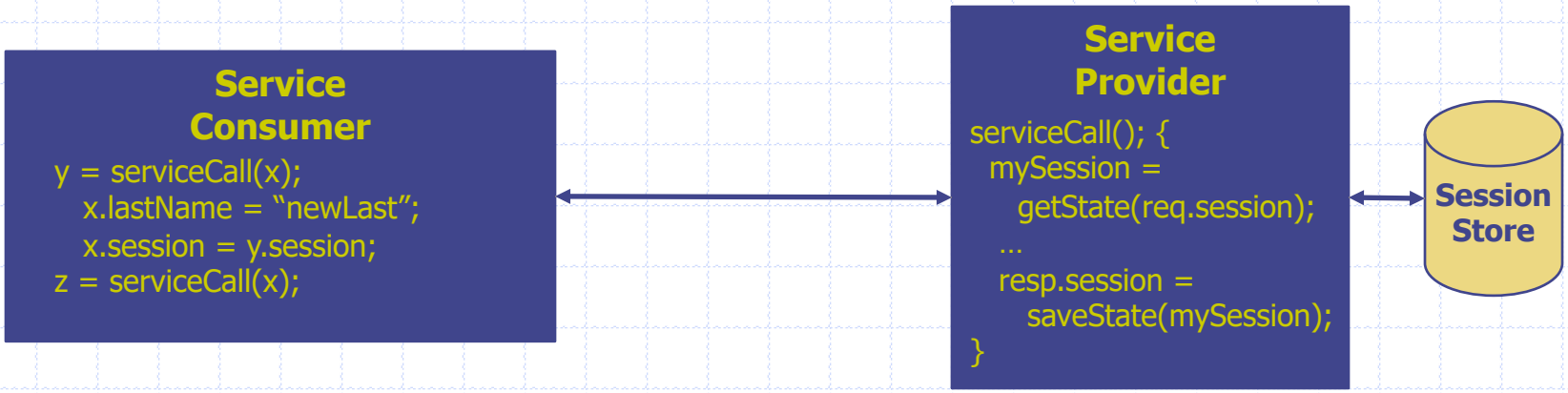


Service Types – Stateful and Stateless

- Stateless – every call is independent

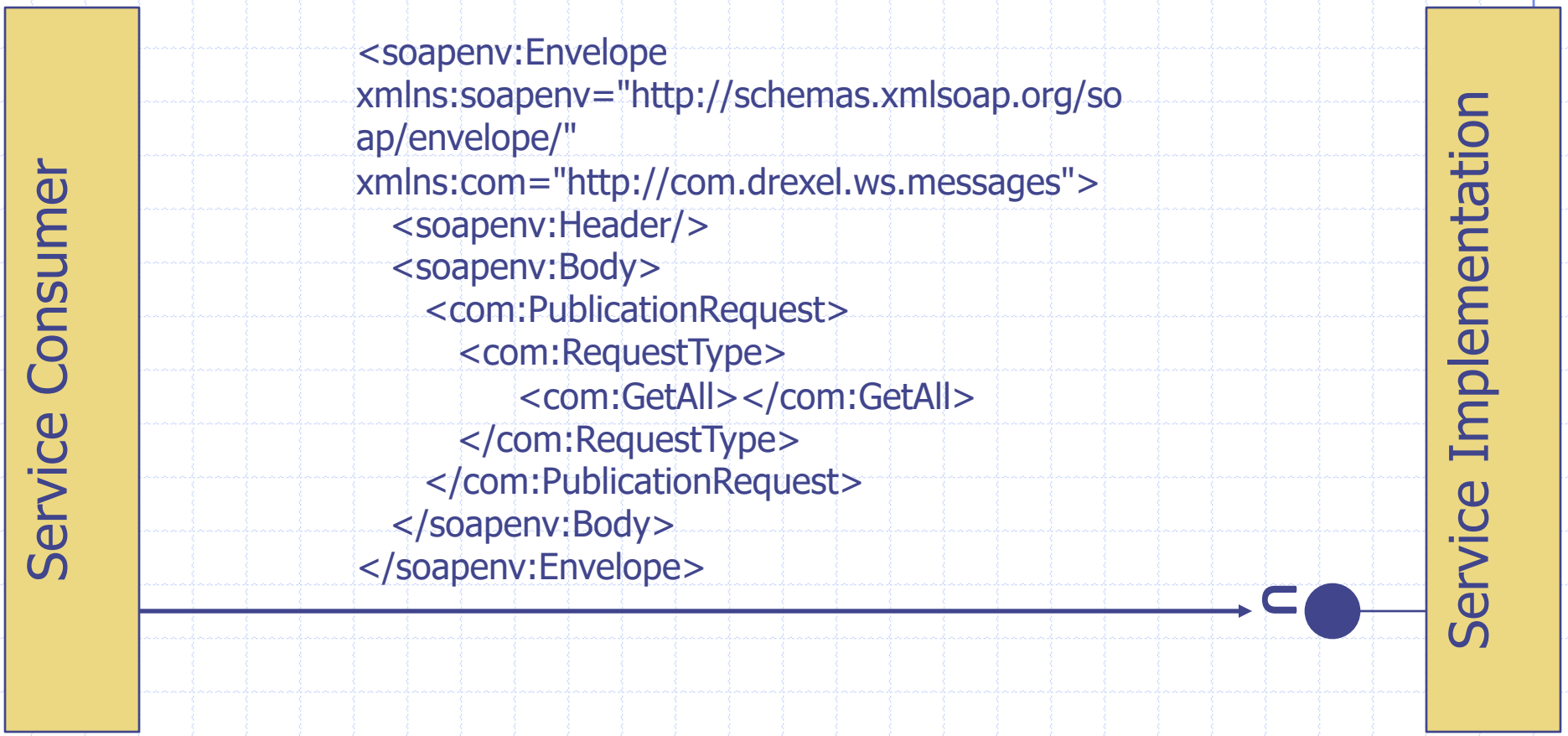


- Statefull – current call depends on state of previous call(s)



To improve scalability stateless is preferred over stateful services

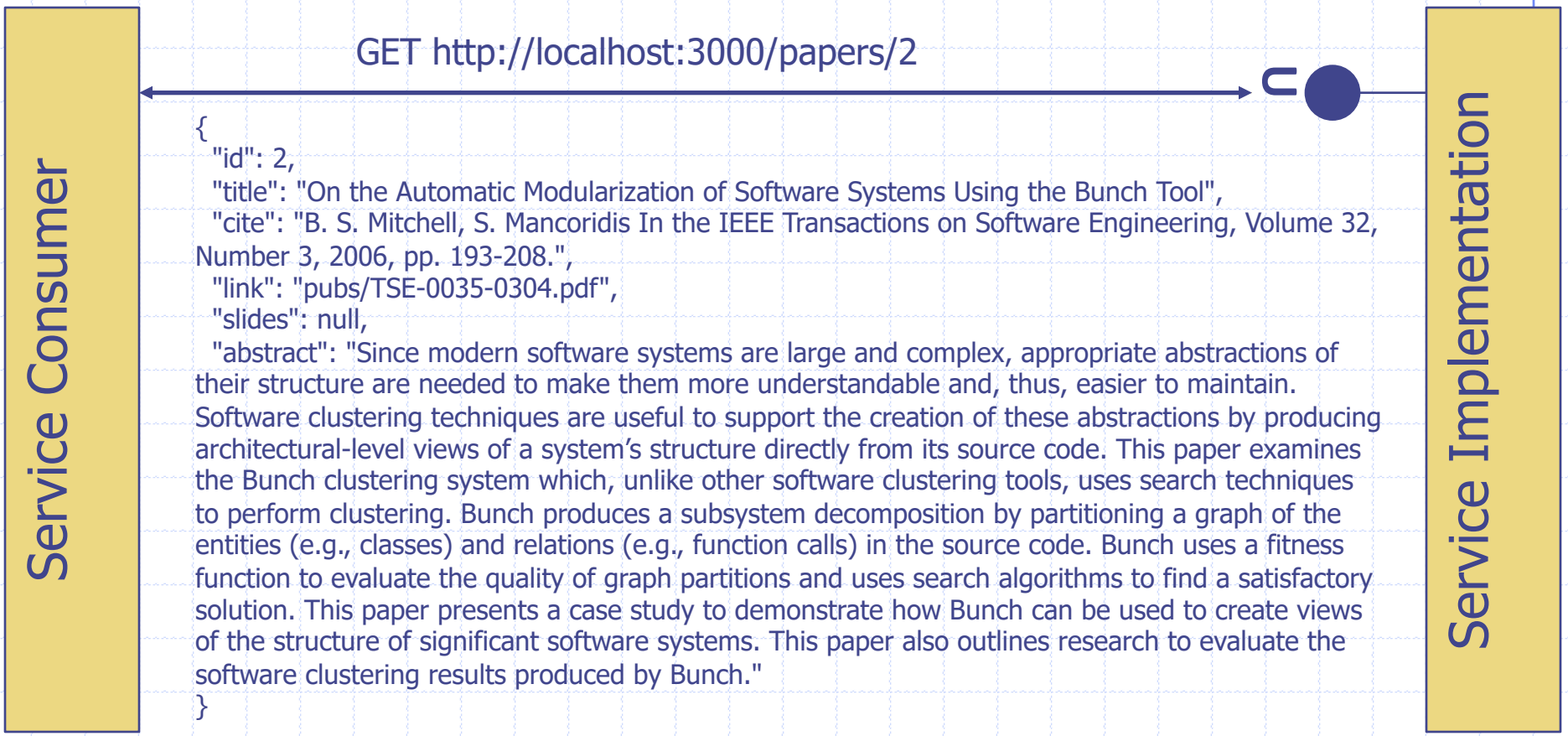
The Architectural Components – Messages - Request



The Architectural Components – Messages - Response


```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns2:PublicationResponse xmlns:ns2="http://com.drexel.ws.messages">
      <ns2:Article id="1">
        <ns2:Title>title</ns2:Title>
        <ns2:AuthorList>
          <ns2:Author>Brian</ns2:Author>
          <ns2:Author>Ben</ns2:Author>
        </ns2:AuthorList>
        <ns2:Cite>by xxxx</ns2:Cite>
        <ns2:PubDate>10/10/2010</ns2:PubDate>
        <ns2:Abstract>abstract</ns2:Abstract>
        <ns2:PubLink>http://xxx.yy.com/xxx.pdf</ns2:PubLink>
      </ns2:Article>
    </ns2:PublicationResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The Architectural Components – Messages – Request/Response REST



Approaches to Identify Services in a SOA Design

- 1. Business Process Decomposition
  - Businesses can be viewed in terms of their primary processes, these processes can be further subdivided into sub-processes, and then decomposed again into very granular processes. These are typically called the L1, L2, and L3 processes
  - The L3 processes are "logical units of work" that support the business
  - Logical units of work have a clear input, output and rules that transform inputs into outputs – these can be services
  - Example: EnrollCustomerInLoyaltyProgram
- 2. Business Functions
  - Similar to business processes, many organizations operate using a collection of business functions
  - Designing around business functions are less likely to be biased by the way the business processes were implemented.
  - A function takes the form of y = f(x) where inputs "x" are transformed into a well defined output(s).
  - Example: ProcessCustomerPayment(x) applies x dollars against the customers account

Approaches to Identify Services in a SOA Design

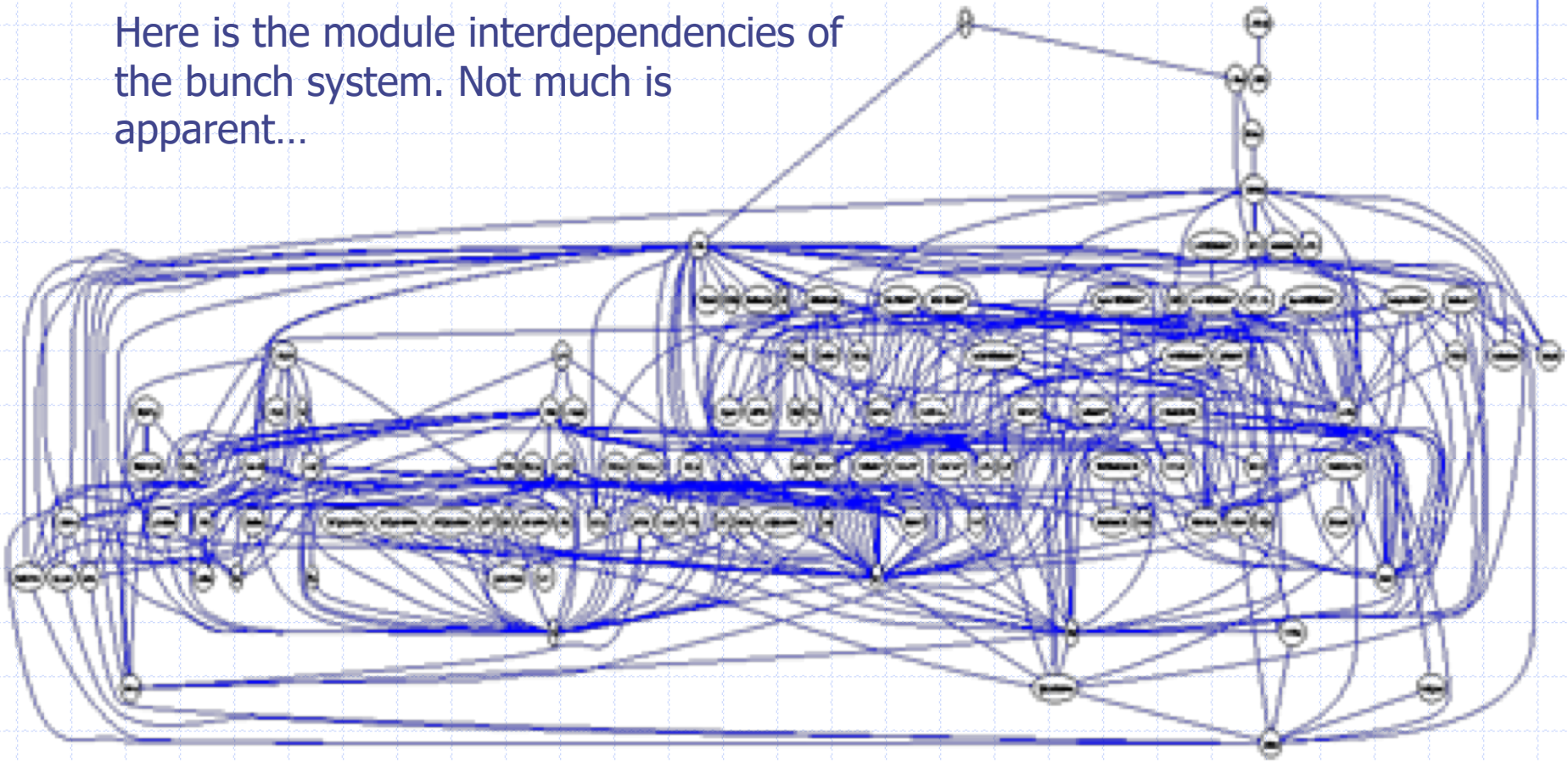
- 3. Look for "Business Entity" Objects in the problem domain
  - Most SOA solutions operate on data. Objects in an OOD work at the table level
  - SOA components operate at the business object level
  - Look for key business entities in the application domain from the perspective of (C)reate, (R)ead, (U)pdate, or (D)elete operations.
  - Examples: Customer, Payments, Order, etc
- 4. Look at "Ownership and Responsibility" to leverage reusable services built by others (where it makes sense)
  - This SOA design attribute is less about identifying a service, and more about identifying who builds, owns and maintains the service.
  - Many SOA applications are "mash ups", combining services from different parties.
  - When identifying a service, look for opportunities to reuse services that are offered by others when they are not key to differentiating you application.
  - Example: If you need a service for mapping or geo-coding would you build this yourself, or would you integrate a service offered by google?

Approaches to Identify Services in a SOA Design

- 5. Goal-Driven Service identification
  - Identify goals of your business or application that can be realized via automated support.
  - Example: Goal: "Improve customer transparency into service pricing"; Service: create an "Estimator" service that allows customer to estimate costs before they make a purchase decision
- 6. Component-Based
  - Use a traditional software engineering approach to identify services
    - Create a conceptual architecture view of the target application
    - Look for natural boundaries that adhere to the principles of maximizing cohesion and minimizing coupling.
    - These identify candidate components
  - For each candidate component see if they well defined responsibility, clear ownership, and if they can be distributed to run in different address spaces
  - Example: The bunch software clustering tool offers a clustering service where the modularization quality calculation is externalized as a service.

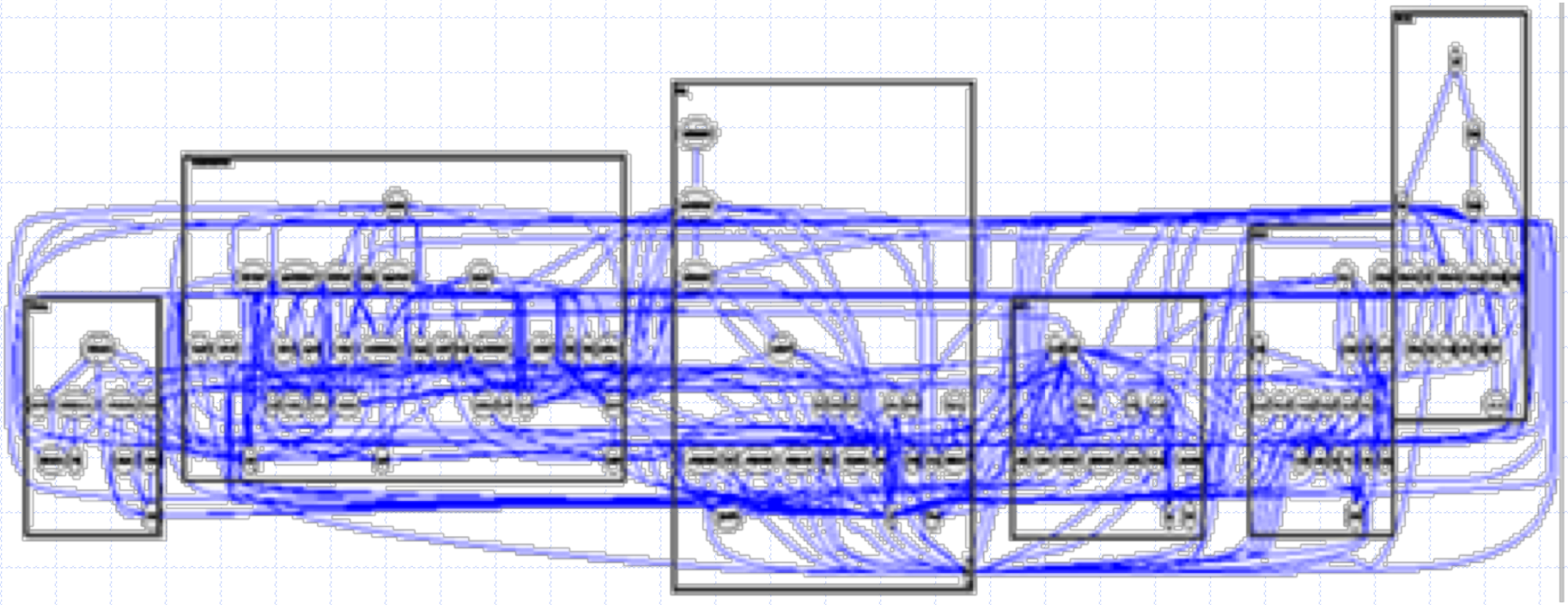
Component-Based Decomposition Example – The bunch system

Here is the module interdependencies of the bunch system. Not much is apparent…



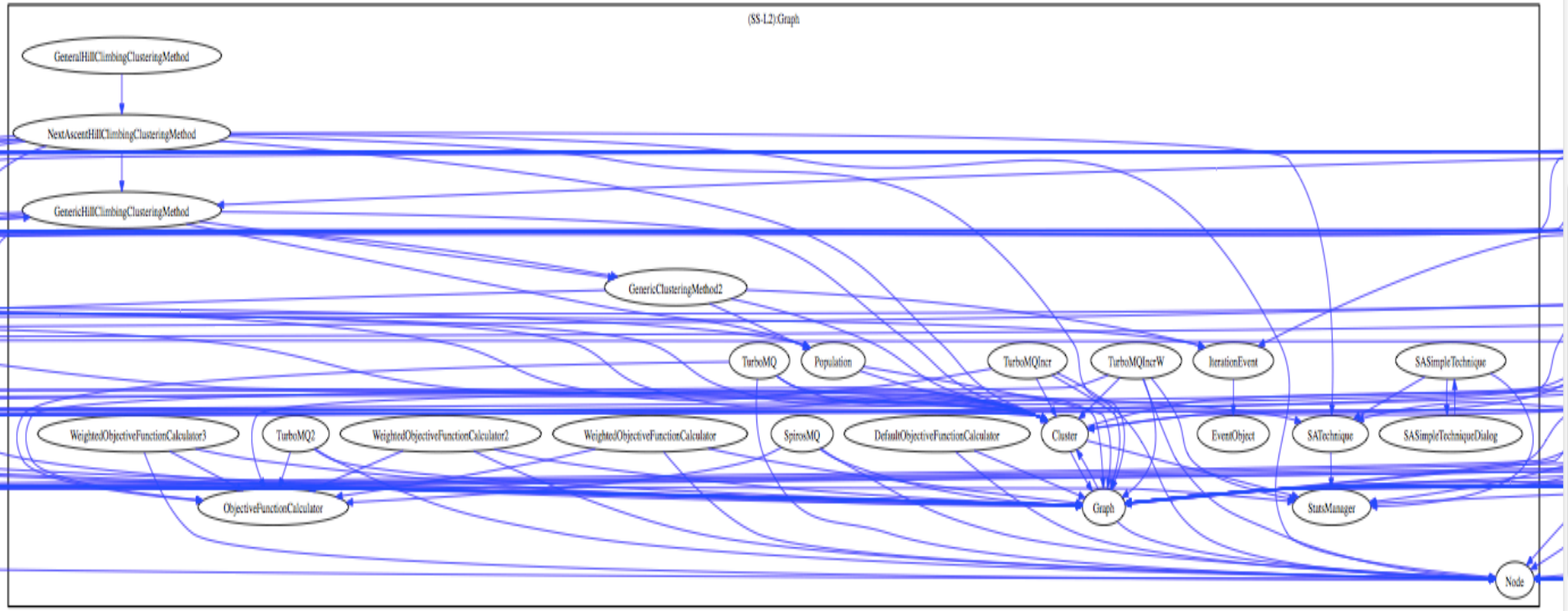
Component-Based Decomposition Example – The bunch system

Here is the module interdependencies of the bunch system. Lets run bunch itself to cluster the system to look for some candidate subsystems…

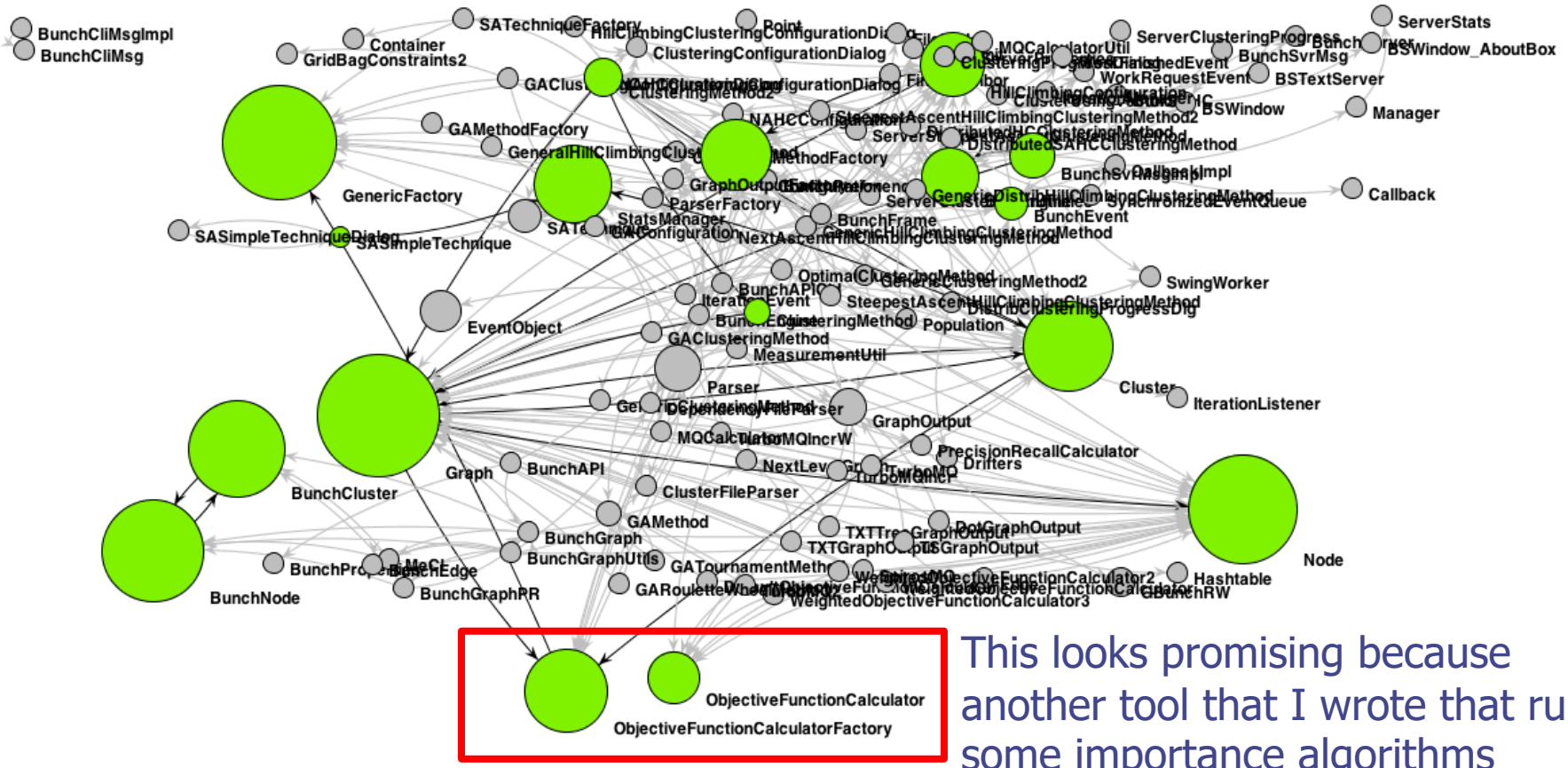


Component-Based Decomposition Example – The bunch system

This one looks promising – it has a lot of functionality related to evaluating the modularization quality function. Domain knowledge also tells us that its compute intensive so it might make for a good service…



Component-Based Decomposition Example – The bunch system



This looks promising because another tool that I wrote that runs some importance algorithms identifies some of these classes as important.

We refactored bunch to support distributed computation and describe it here…

An Architecture for Distributing the Computation of Software Clustering Algorithms

Brian Mitchell, Martin Traverso, Spiros Mancoridis
Department of Mathematics & Computer Science
Drexel University, Philadelphia, PA, USA
{bmitchel, umtraver, smancori}@mcs.drexel.edu

Abstract

Collections of general purpose networked workstations offer processing capability that often rivals or exceeds supercomputers. Since networked workstations are readily available in most organizations, they provide an economic and scalable alternative to parallel machines. In this paper we discuss how individual nodes in a computer network can be used as a collection of connected processing elements to improve the performance of a software engineering tool that we developed.

Our tool, called Bunch, automatically clusters the structure of software systems into a hierarchy of subsystems. Clustering helps developers understand complex systems by providing them with high-level abstract (clustered) views of the software structure. The algorithms used by Bunch are computationally intensive and, hence, we would like to improve our tool's performance in order to cluster very large systems. This paper describes how we designed and implemented a distributed version of Bunch, which is useful for clustering large systems.

handling applications that have significant interprocess synchronization and communication requirements. However, computationally intensive applications that can partition their workload into small, relatively independent subproblems, are good candidates to be implemented as a distributed system. One such application is a tool that we developed to recover the high-level subsystem structure of a software system directly from its source code.

More than ever, software maintainers are looking for tools to help them understand the structure of large and complex systems. One of the reasons that the structure of these systems is difficult to understand is the overwhelming number of modules, classes and interrelationships that exist between them.

Ideally, system maintainers have access to accurate requirements and design documentation that describes the software structure, the architecture and the design rationale. Unfortunately, developers often find that no accurate design documentation exists, and that the original developers of the system are not available for consultation. Without automated assistance, developers often modify the source code without a thorough understanding of how their modifications affect

Approaches to Identify Services in a SOA Design

- 7. Existing Supply – Refactoring into a SOA
  - Look at existing inventory of applications and how they work together
  - Identify existing integration interfaces – API calls, Database Transactions, Database Queries
  - Select integration points that can be migrated away from using existing integration patterns into a SOA pattern
  - Example: Refactor an application that supports customer service, identify all ODBC and SQL calls that related to customer management, refactor by creating a SOA service for "Customer"
- 8. Front Office Application Usage Analysis
  - Most enterprise have multiple applications that have some underlying redundant functionality – typically this functionality is implemented differently in each application.
  - Identify these redundant functions and see if they can be encapsulated into a common service.
  - Example: Drexel grading data is used by students, facility and the administration – it can be accessed via the web, but we would like to extend to mobile. Solution is to create a common grading service that can be used by multiple applications. Imagine a service like "HoldGrades()" that is called from the billing system

Approaches to Identify Services in a SOA Design

- 9. Infrastructure
  - Although not common, there are cases when services are useful to take advantage of sharing infrastructure-specific capabilities.
  - Consider that you have a special piece of hardware that performs a very specific function and you want to share that capability with a lot of consuming applications.
  - Consider when you need to isolate a specific function to a certain piece of hardware or network zone for compliance purposes
  - Create a service that hides the specifics of the special infrastructure and expose it to other applications. Example: Use PayPal's payment services instead of doing your own credit card processing – eliminates the need to deal with PCI compliance.
- 10. Look at non-functional requirements.
  - Examine the set of non-functional requirements and look for opportunities where centralizing the implementation of certain capabilities helps enable the non-functional requirements.
  - Common examples are from security and performance set of non-functionals.
  - Example: Create a service to encapsulate making authorization decisions abstracting the complex dataflow of the oAuth protocol; another example is the distribution of the MQ calculation in the Bunch tool (from Method #6)

Reference: "Ten Ways to Identify Services", http://searchsoa.techtarget.com/tip/Ten-ways-to-identify-services

Lets Try To Design-
Example: Online "Secure" Wallet App

- Problem Statement: The goal of the application is to design a secure online wallet application. The application should:
  - Allow a user to login, create and manage a profile
  - Allow a user to manage various security and financial instruments:
    - Bank and investment account information (account numbers, account types, security credentials)
    - Credit Card Information (accounts, numbers, expiration dates) to support automated payments
    - Website credentials (website address, user ids, passwords)
    - Loyalty card programs (card types, account numbers, etc)
    - Frequent flyer account information
  - All information at rest and transmitted should be encrypted
  - Access to the application is via a single strong password
  - Secondary passwords can be used to secure additional account information access
  - Application should be designed so that it can be accessed over the web and via mobile devices
- Questions: Why is this application a good candidate to be implemented using a SOA pattern?
- What are the key components and connectors?

Specialization of SOA Models

- Contract/Operation Based (SOAP)
  - Service is defined in terms of operations that the service can perform, the messaging frameworks that the service supports, and a strict definition of the data structures supported by the service operations (request and response)
  - The service contract is specified in a special XML document, called a WSDL document
  - The WSDL document can also specify policies associated with using the service.
  - Contract can be loosely- or strictly- enforced
  - Lots of extensions, defined as WS-* standards
  - Messaging typically in the form of XML documents
- Resource Based (REST),
  - First introduced in 2000 by Roy Fielding's Ph.D. thesis
  - Service is defined in terms of resources (generally nouns)
  - Service operations are mapped to HTTP verbs
  - Service runtime utilizes web runtime, in fact they cant be distinguished
  - Messaging typically in the form of JSON documents, although XML is also used
  - Example-based versus contract-based specification. Some work on this with the WADL specifications.

Up until a few years ago SOAP was the more popular approach; however, now RESTful services are dominating SOA-based architecture…

We will also look at very recent advancement of SOA patterns later in the course – Reactive, Graph Query, etc.

Specialization of SOA Models

- GraphQL
  - Open-sourced by Facebook
  - Strongly typed schemas
  - A JSON based query language

https://graphql.org/

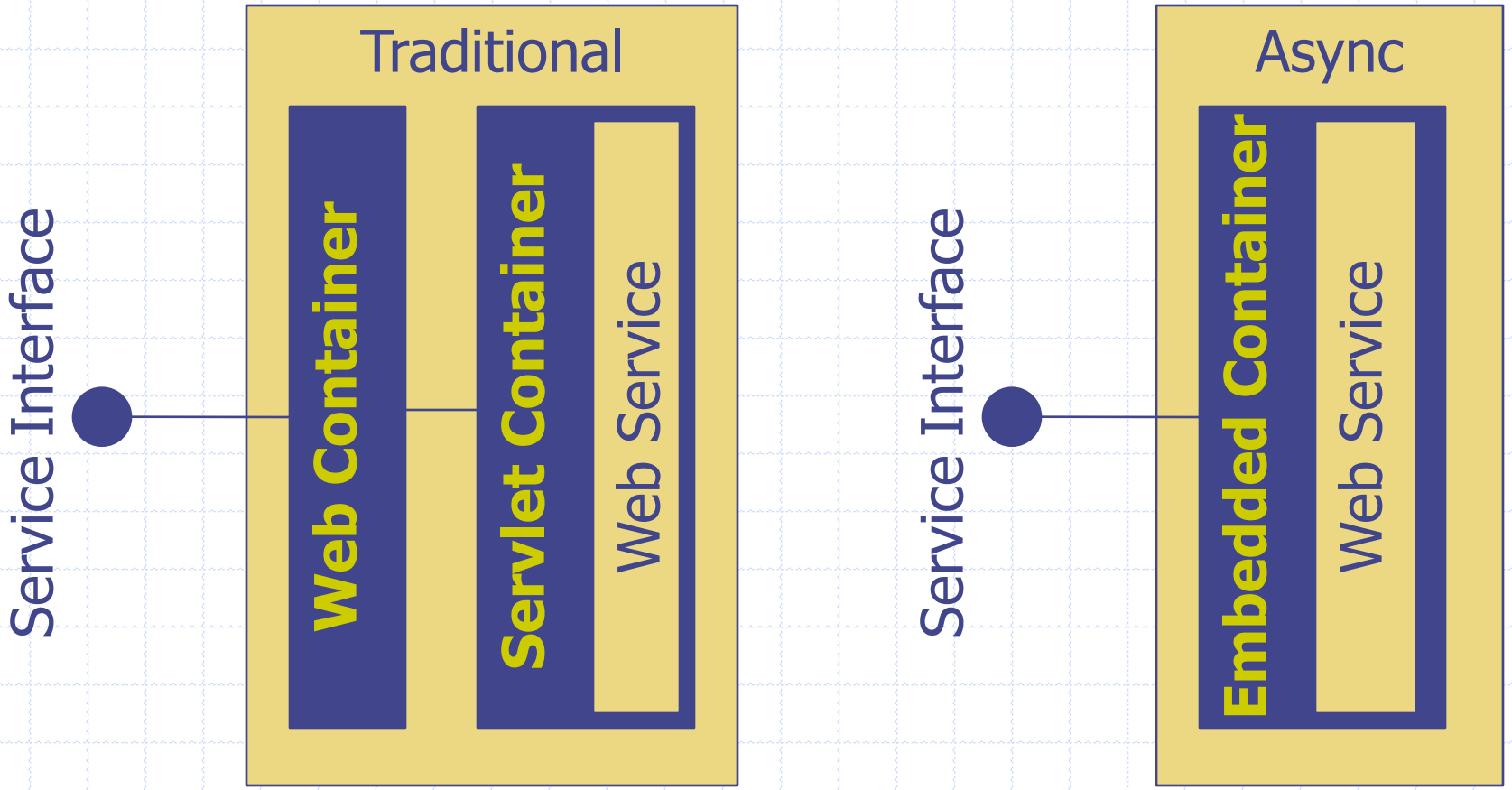
- gRPC
  - Open Sourced by google
  - Based on Protocol Buffers
  - Binary and very fast

https://grpc.io/

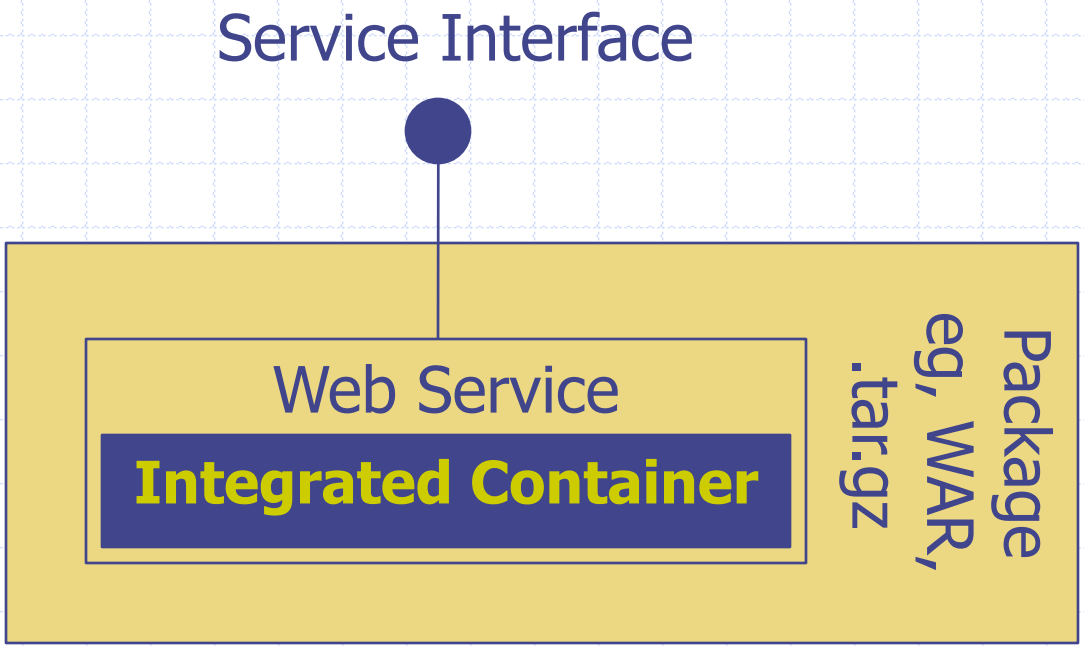
SOA Requirements around the Runtime Stack

- SOA services generally run within a framework to provide a number of useful capabilities:
  - Network – TCP/IP, HTTP/S, HTTP2
  - Serialization – objects to/from XML or JSON (or Binary)
  - Thread Management – allocating threads to services or providing an async framework (remember the reactor pattern).
  - Security – best practice is to externalize security from the service implementation. This includes user management (authentication/authorization) as well as defending against DoS and injection attacks

The runtime containers for web services come in two flavors



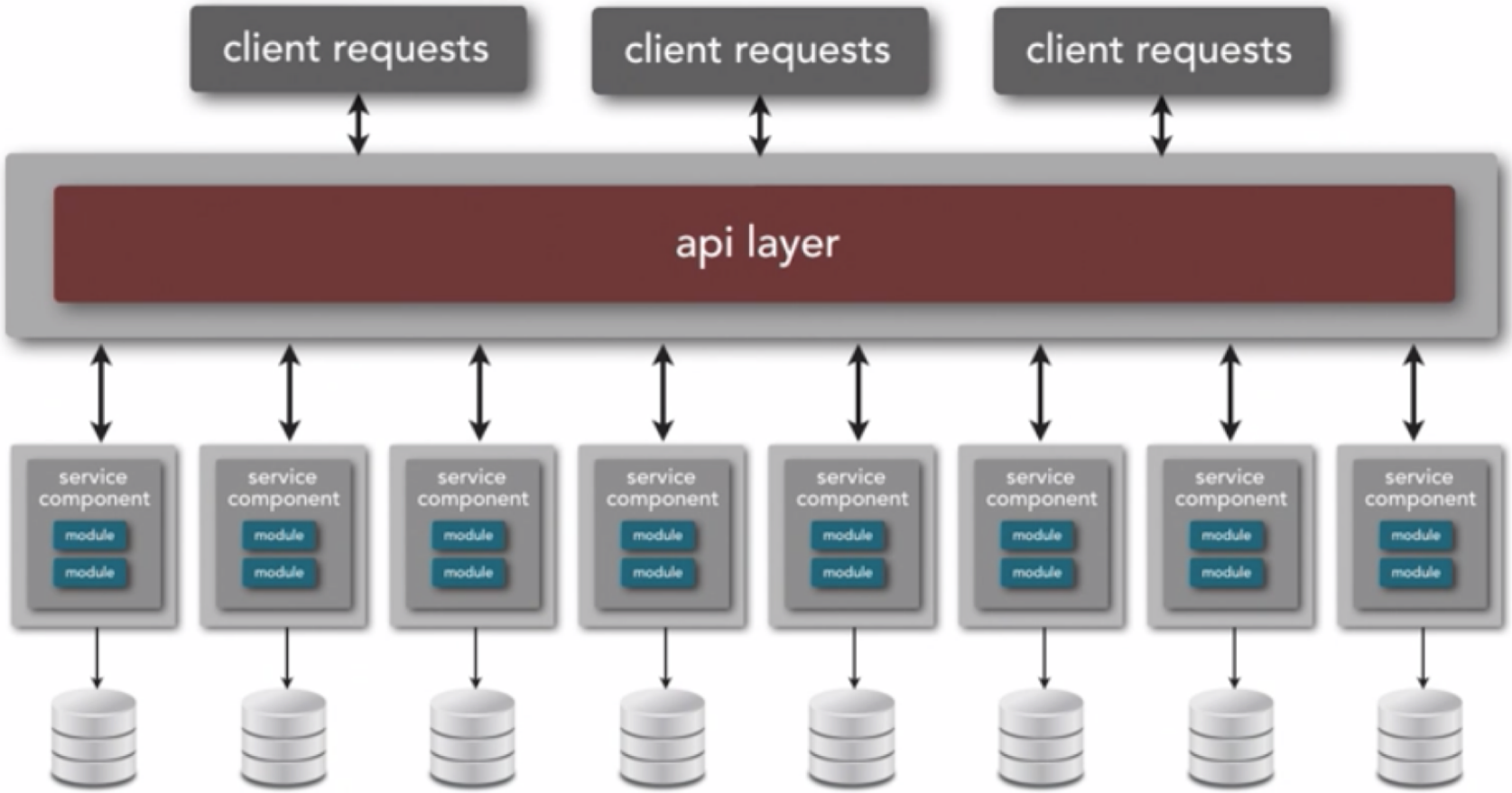
And most recently a third option is arising Microservices



Additional information on Microservices can be found on Martin Fowlers website or looking at reference implementations like DropWizzard, Sprint Boot, or Docker

Microservice Architecture

Ref: Neil Ford & Mark Richards
Software Architecture Fundamentals



Basic Idea – SHARE NOTHING

References

- SOA Patterns, http://www.eaipatterns.com/SoaPatterns.pdf
- "From Objects to Services: A Journey in Search of Component Reuse Nirvana" by M. Dodani, Journal of Object Technology vol3, no. 8, http://www.jot.fm/issues/issue_2004_09/column5.
- "Service-Oriented Architecture : A Field Guide to Integrating XML and Web Services" by T. Erl, Prentice Hall , ISBN: 0131428985
- "What Is An Enterprise Service Bus?", by M. Gilpin, Forrester Research, http://www.forrester.com/go?docid=35193.
- "Ten Ways to Identify Services", http://searchsoa.techtarget.com/tip/Ten-ways-to-identify-services
- Microservices, http://martinfowler.com/articles/microservices.html