Service Oriented Architectures (SOA):

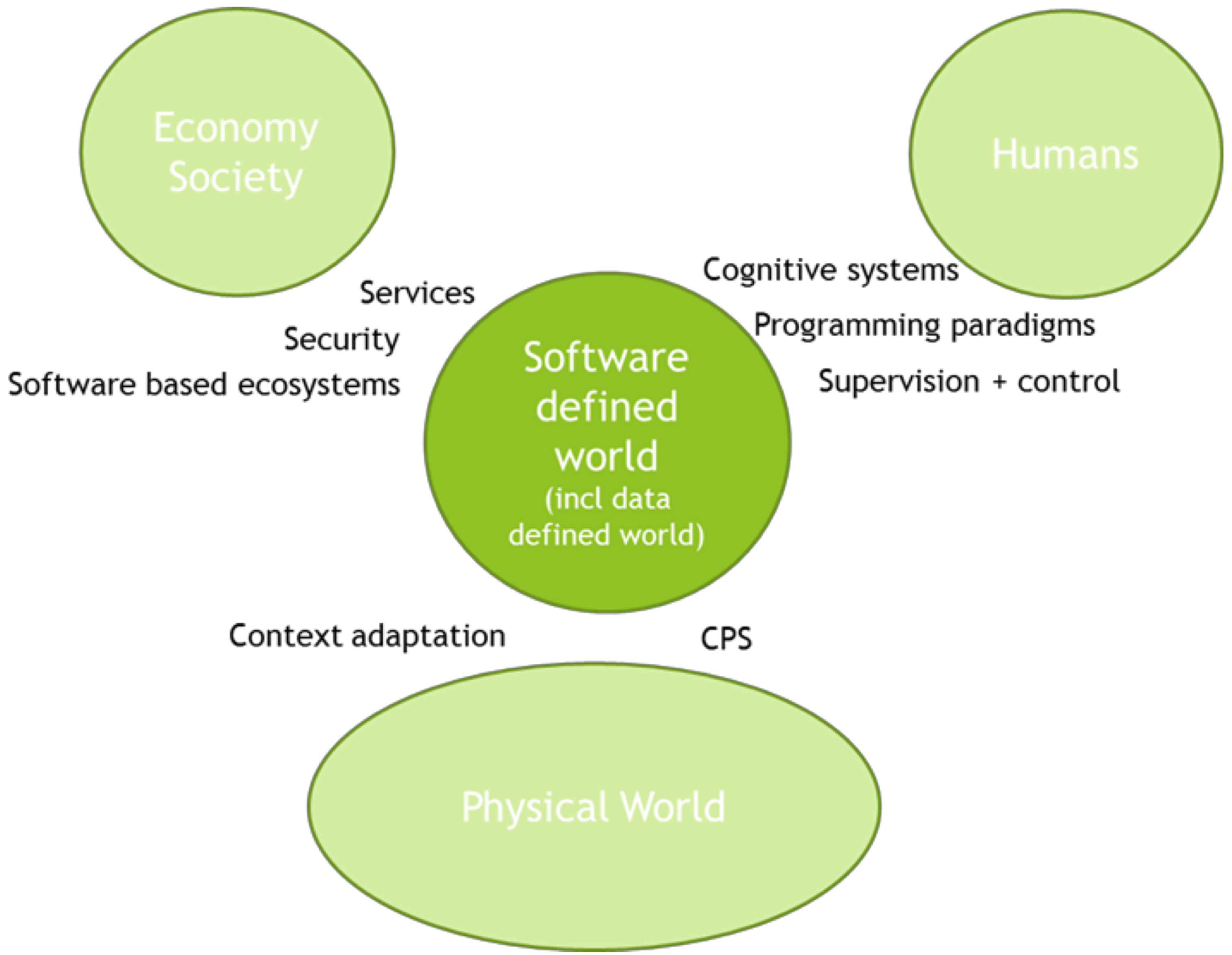
Architectural styles and patterns for services and microservices

Prof. Paolo Ciancarini
Software Architecture
CdL M Informatica
Università di Bologna

Agenda

- Software as a Service (SaaS)
- The SOA style
- The REST style
- Microservices

Software is "eating the world"



From products to services

- Our society is shaped by the forces of
  - Specialization
  - Standardization
  - Scalability
- It is currently very “service” oriented
  - Transportation
  - Telecommunication
  - Retail
  - Healthcare
  - Financial services
  - Education and training
  - …

The service economy

- Modern economies rely upon *services*
- This means that some companies **offer support** for activities in the primary (agricolture) or secondary (industry of goods) sectors, or to other companies offering services themselves
- A service company is usually independent (**not owned**) from other companies and their services; however in several cases they should cooperate or at least be **coordinated** in some way

The API economy

UBER

No car

Alibaba Group 阿里巴巴集团

No storage

airbnb

No hotel

What is the API economy?

Photo service increases revenues **x6** wrt shops

Sensors and API's open for all trash containers to share info of when to pick up trash

90% revenues from API

60% revenues from API

Attributes of physical services

- A service is not owned by its user (compare with *product*)
- It has a well defined, easy-to-use, standardized interface
  - New services can be offered by combining existing services
- (almost) always available but idle until requests come
- "Provision-able" (used by someone only when necessary, then reassigned to someone else)
- A service should be easily accessible and usable readily
  - no "integration" required
  - Self-contained: no visible dependencies to other services
- A service is usually coarse grained
- It is independent from the consumer's context
  - but a service can have a context
- It should have a quantifiable quality of service
  - Services do not compete on "What" but on "How"

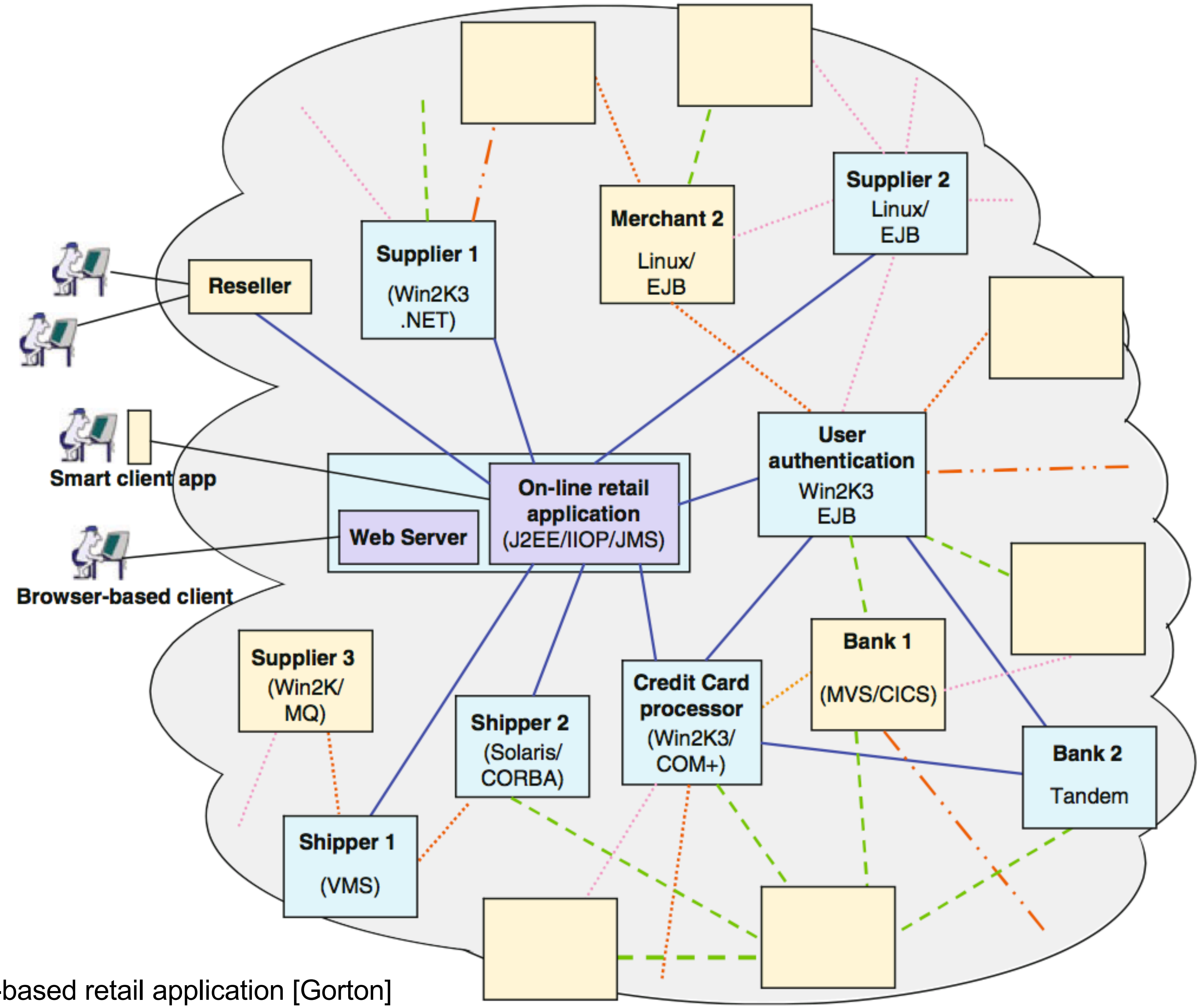
Example: mail

- Customers use mail everywhere
- Interfaces: PostOffice, Mail Box, stamp, postman
- Apparently no relationship with other services (eg. transportation, payments)
- Quantifiable quality of service: price, delivery time, lost messages

What is meant by *service*?

- In economics and marketing, a **service is the intangible equivalent of an economic good**. Service provision is an economic activity where the buyer does not generally, except by exclusive contract, obtain exclusive ownership of the thing purchased (Wikipedia, 2010)
- A service is a "*provider to client*" **interaction** that creates and captures value while sharing the risks of the interactions
- A service is the application of specialized competences (skills and knowledge) for the benefit of someone
- A service is a value that can be **rented** (in the broad sense) by some process that the renter (client) participates in. This contrasts with goods, whose value (once purchased) is owned by the customer (Lovelock & Gummesson, 2004)

http://sdlogic.net/foundations.html



Service-based retail application [Gorton]

Software-as-a-Service (SaaS)

- 1990: Web 0.9 (Tim Berners-Lee)
- 1995: Web 1.0 (some dynamic content, Netscape)
- 1997: Services (Google, e-commerce...)
- 1999: LoudCloud (first infrastructure for SaaS)
- 2000: Web 2.0 (rich UI's, social computing)
- 2001: Autonomic computing (IBM)
- 2004: SaaS & SOA (Google Maps, Amazon S3...)
- 2006: Amazon Web Services
- 2008: Cloud Computing (pay as you go)
- 2011: Microservices

Software-as-a-Service (SaaS)

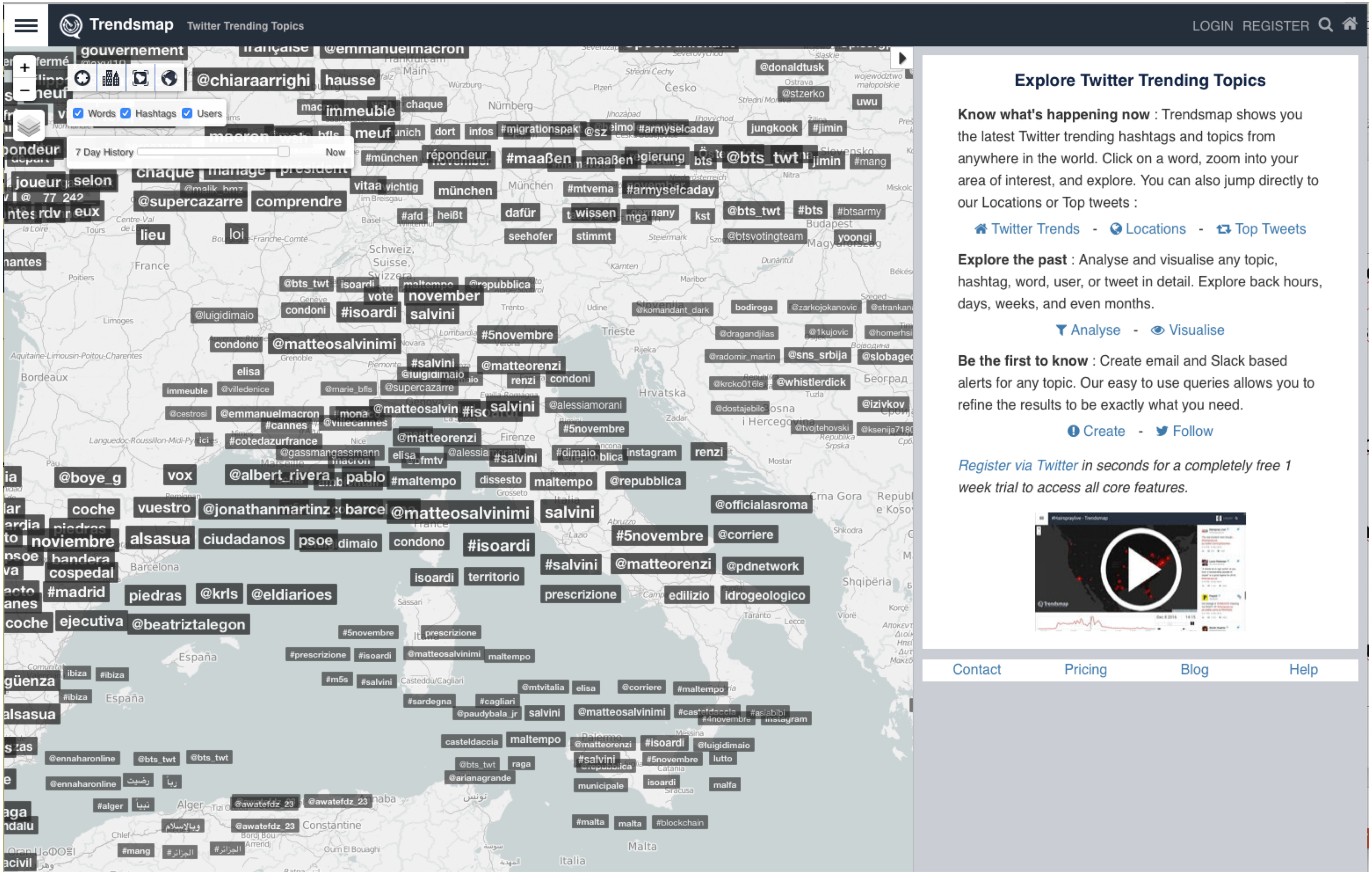
- Definition of SaaS: Internet-based deployment (for access and management) of commercially available software
- Service managed from “central” locations enabling customers to access applications remotely via web
- Delivery: one-to-many model (single instance, multi-tenant architecture) including architecture, pricing, partnering, and management characteristics
- Centralized updating, which obviates the need for end-users to download patches and upgrades
- Frequent integration into a larger network of communicating software—either as part of a *mashup* or a plugin to a platform as a *service*

(Web) Mashup

- A mashup is a web application that uses content from more than one source to create a single new service displayed in a single interface
- For example, a user could combine the addresses and photographs of their library branches with a Google map to create a map mashup.
- The term implies easy, fast integration, frequently using open application programming interfaces (open API) and data sources to produce enriched results that were not necessarily the original reason for producing the raw source data.

- The main characteristics of a mashup are combination, visualization, and aggregation.
- It is important to make existing data more useful, for personal and professional use.
- To be able to permanently access the data of other services, mashups are generally client applications or hosted online.

Mashup example: trendsmap



Explore Twitter Trending Topics

Know what's happening now : Trendsmap shows you the latest Twitter trending hashtags and topics from anywhere in the world. Click on a word, zoom into your area of interest, and explore. You can also jump directly to our Locations or Top tweets :

Twitter Trends - Locations - Top Tweets

Explore the past : Analyse and visualise any topic, hashtag, word, user, or tweet in detail. Explore back hours, days, weeks, and even months.

Analyse - Visualise

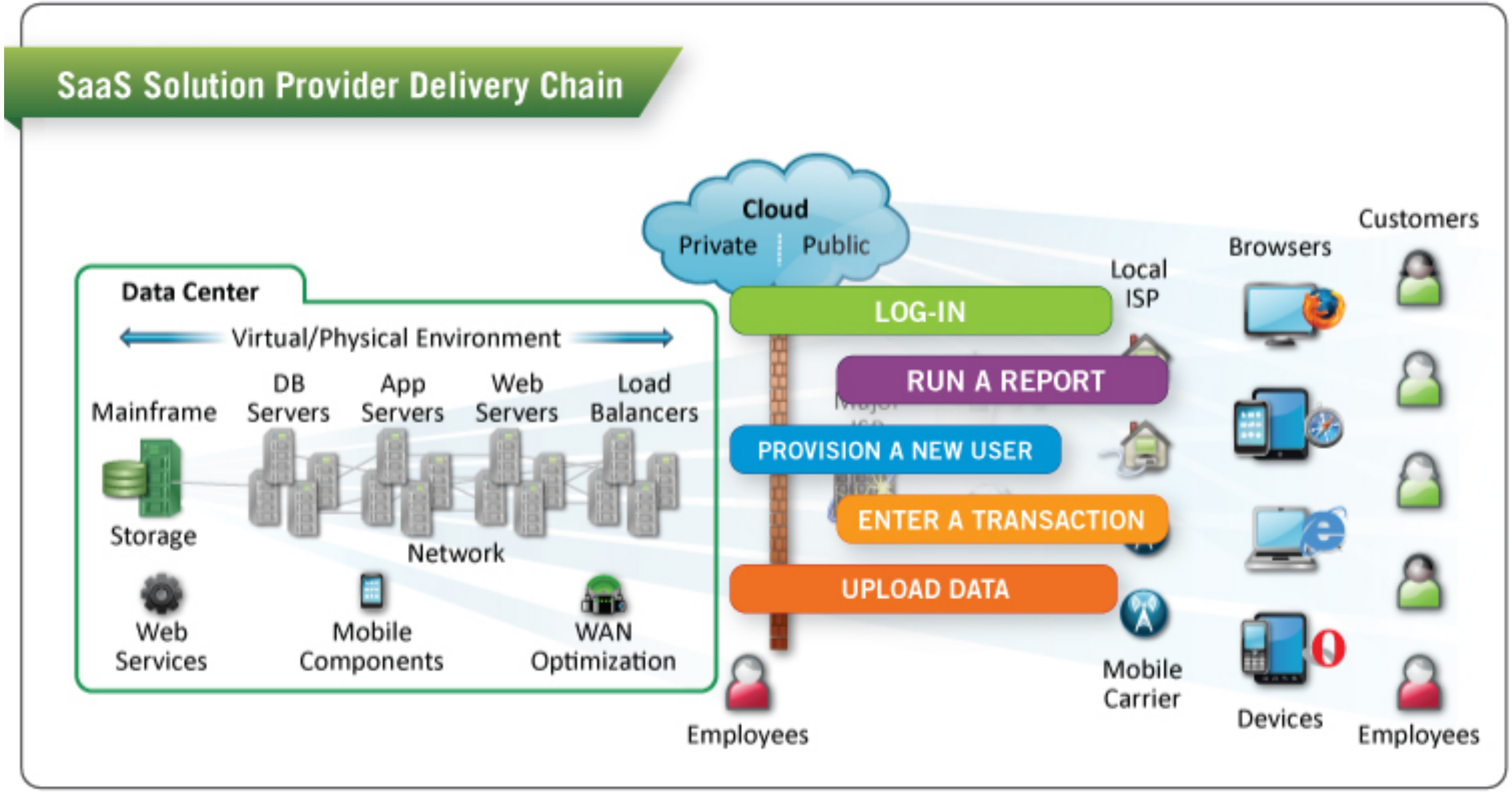
Be the first to know : Create email and Slack based alerts for any topic. Our easy to use queries allows you to refine the results to be exactly what you need.

Create - Follow

Register via Twitter in seconds for a completely free 1 week trial to access all core features.

Contact Pricing Blog Help

SaaS delivery chain



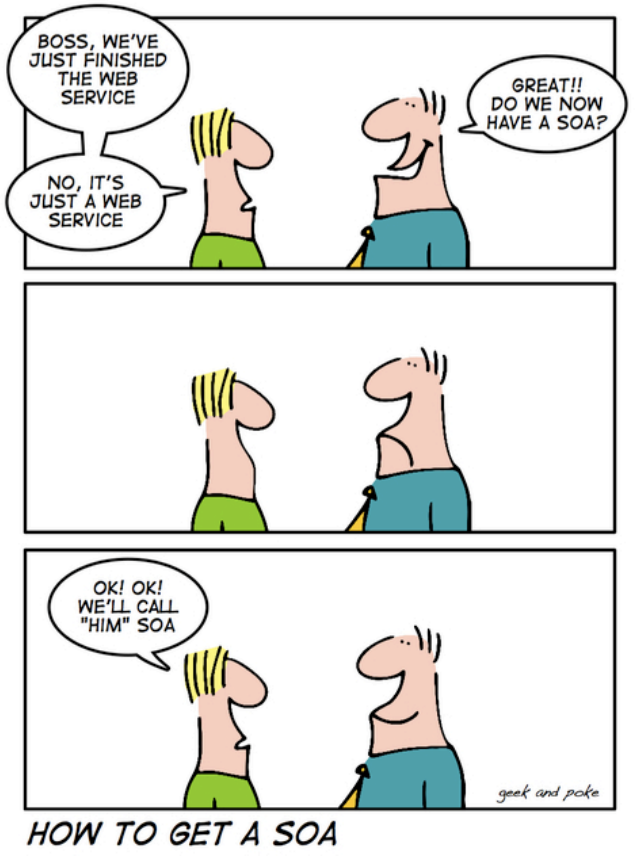
Source: Compuware

Software as a service

- An operating system service?
  - Program execution, file management

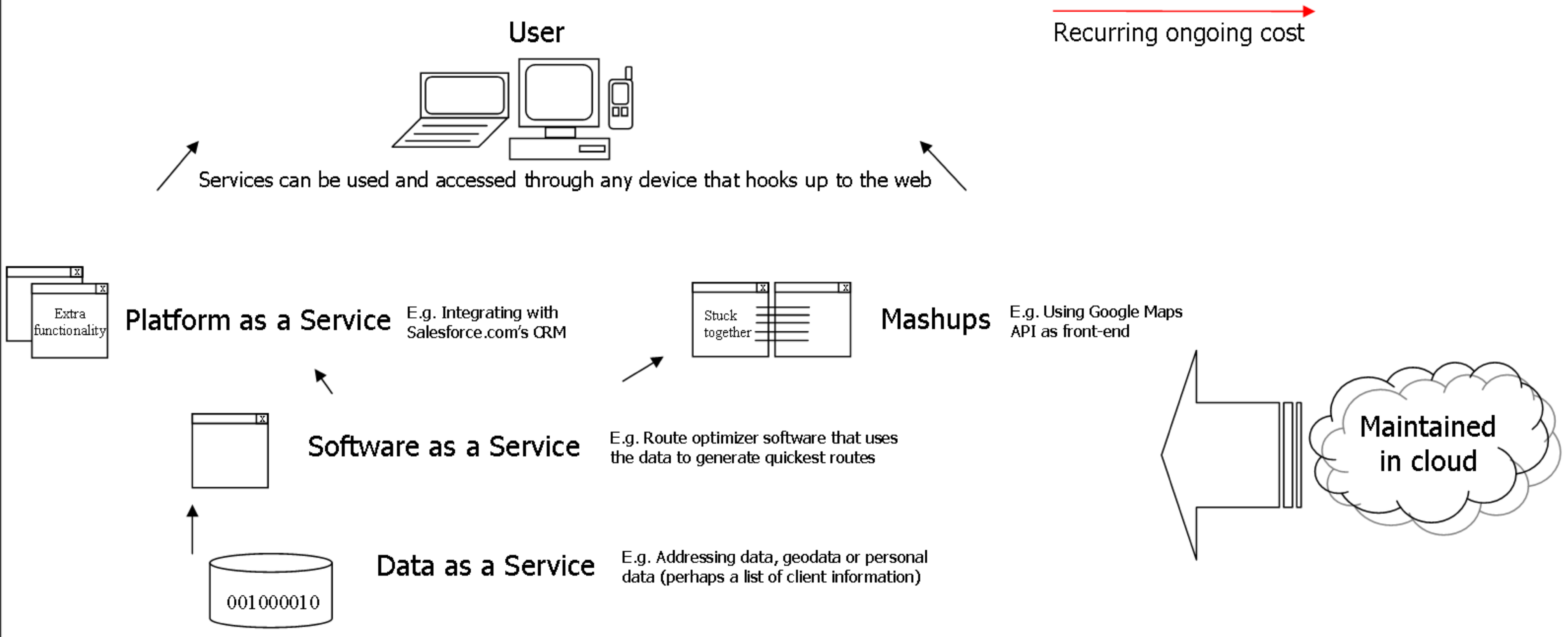
- A Software Service?
  - Compiler Service
  - Search engine service
  - Deployment service

- A Business Service?
  - Customer service
  - Bidding service


BOSS, WE'VE JUST FINISHED THE WEB SERVICE
GREAT!! DO WE NOW HAVE A SOA?
NO, IT'S JUST A WEB SERVICE
OK! OK! WE'LL CALL "HIM" SOA
geek and poke
HOW TO GET A SOA

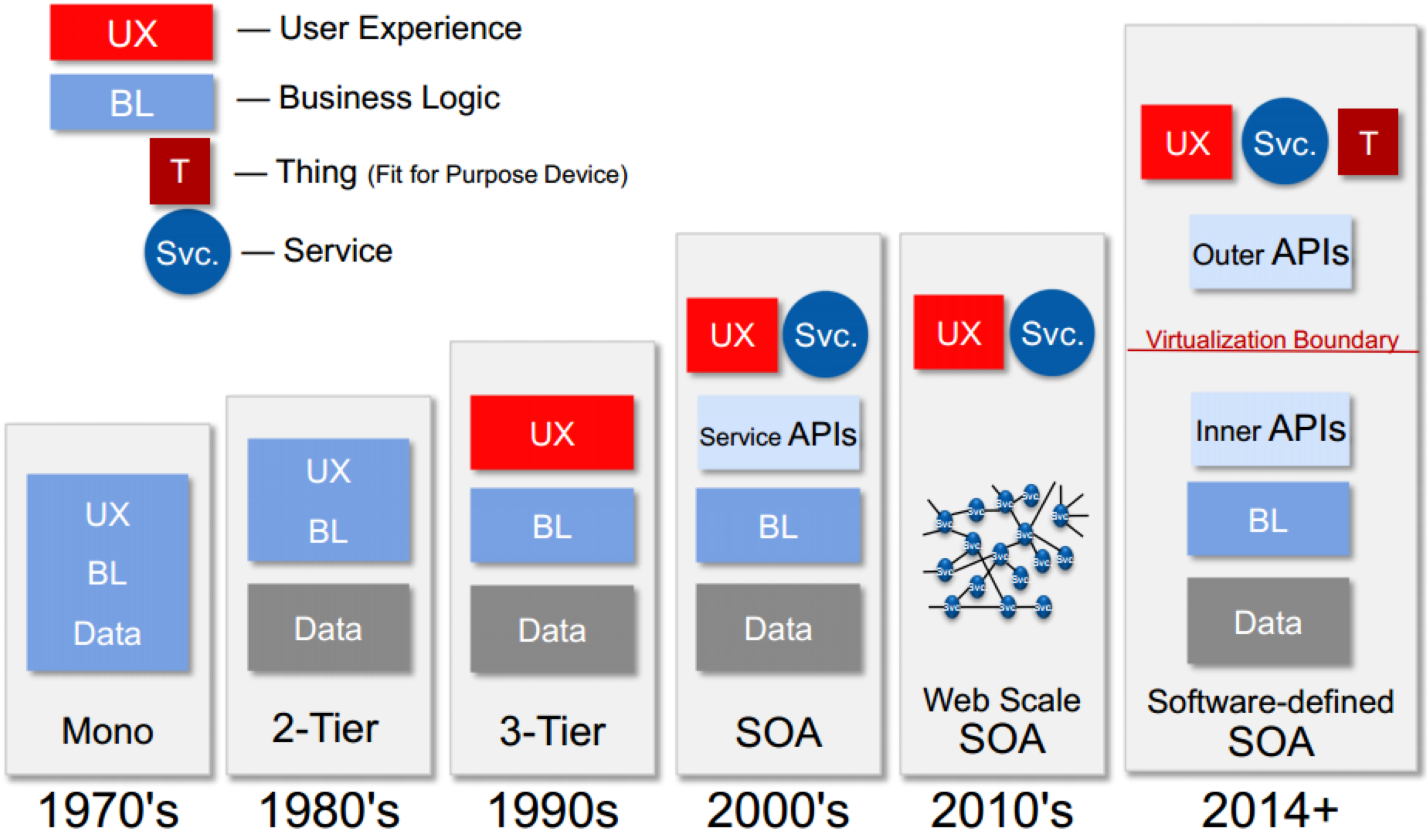
Service-Oriented Architecture

A completely service-oriented model



User
Recurring ongoing cost
Services can be used and accessed through any device that hooks up to the web
Extra functionality
Platform as a Service
E.g. Integrating with Salesforce.com's CRM
Stuck together
Mashups
E.g. Using Google Maps API as front-end
Software as a Service
E.g. Route optimizer software that uses the data to generate quickest routes
Maintained in cloud
001000010
Data as a Service
E.g. Addressing data, geodata or personal data (perhaps a list of client information)

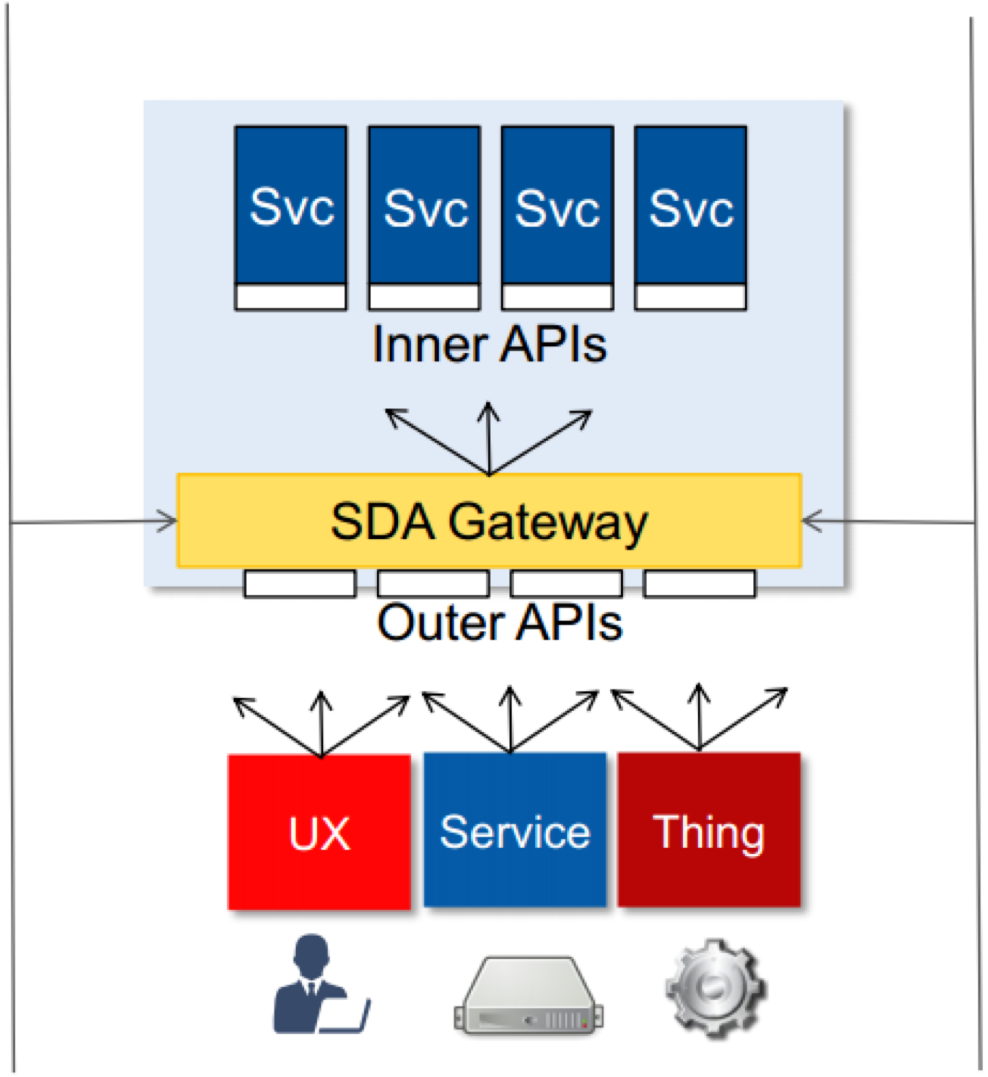
Software-defined Applications on the Application Architecture Road Map



Gartner

Step Up to Digital Business: 2. Software-defined Architecture for Applications

- Virtualization
- API Translation
- Protocol Translation
- Model Translation
- Ad-hoc APIs
- Security
- Monitoring
- Optimization
- Routing
- Load Balancing
- Integration
- Composition
- Orchestration
- Context Injection
- Scheduling



- API Manager
  - Apigee,
  - Mashery
- API Gateway
  - Layer7,
  - Vordel
- ESB suite
  - Tibco,
  - Software AG
- iPaaS
  - IBM CastIron,
  - Dell Boomi

Based in part on “Magic Quadrant For Application Services Governance”

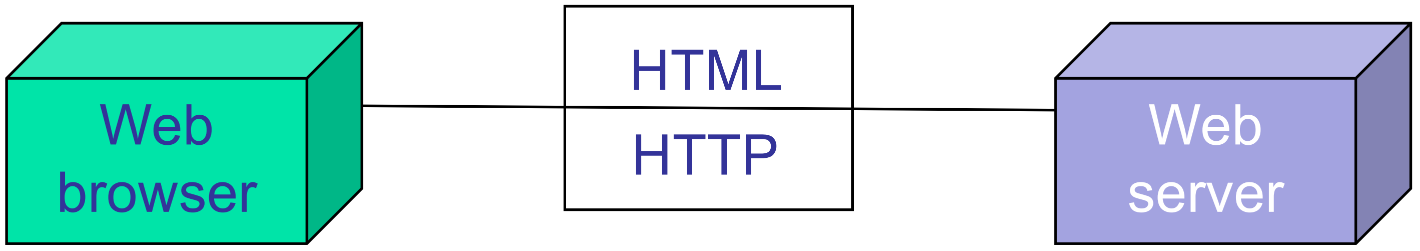
Gartner.

SaaS vs SOA

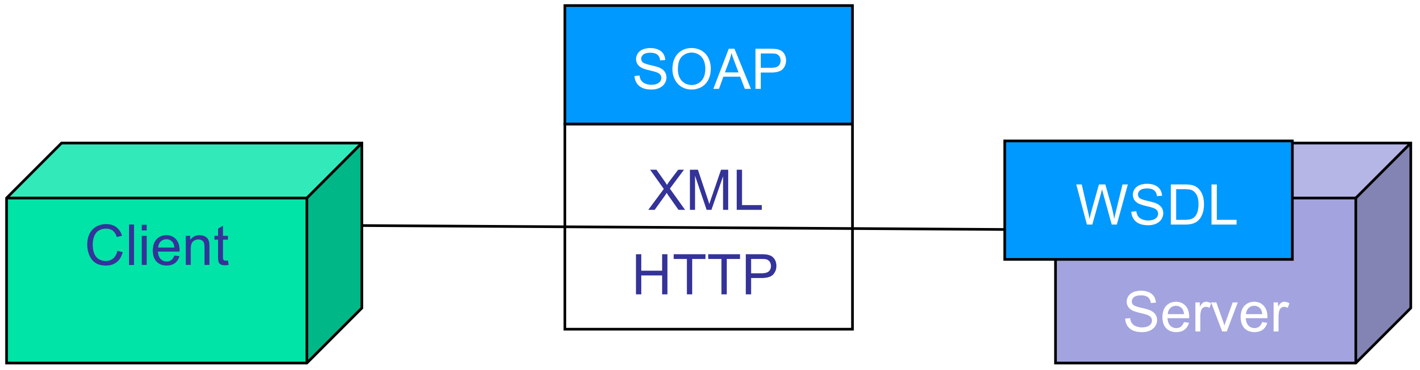
- SOA is an architecture, whereas SaaS is hosting a set of software services over the Web.
- SaaS focuses on Software Hosted As A Service,
- SOA focuses on Software Designed As A Service.
- SaaS may be considered as a consumption model in which a user is involved; SOA is a design model in which there is no restriction on who the consumer is
- all SaaS implementations follow the SOA concept. SaaS relies upon the Web, whereas SOA does not restrict its use on the web only
- SaaS means using software as a service over the Web using some protocol, which is used to communicate between the client side application and the server side software service.
- Traditionally, SaaS services use REST but SOAP (as discussed later) is also used. SaaS services are also hosted on the cloud just like Web services, but a SaaS application usually calls the services using RESTful services, where as web services make calls using RPC (Remote Procedure Call)

SOA

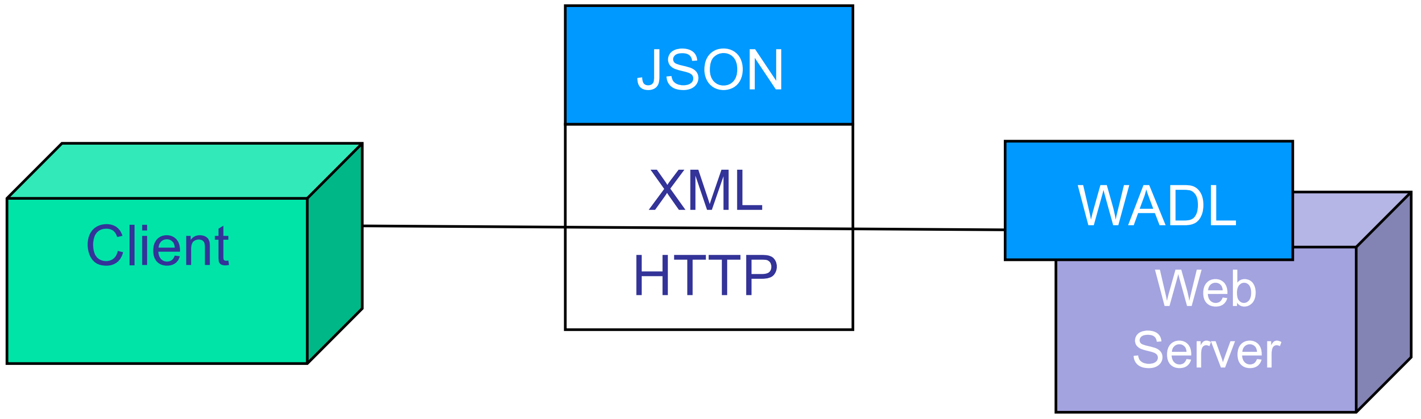
- SOA = Service Oriented Architecture
- SOA is not a reference architecture, it is a technology stack for application integration
- The essence of a SOA lies in **independent** services interconnected by **messages**
- On the Web, a specific SOA technology stack is Web Services (W3C)



Web

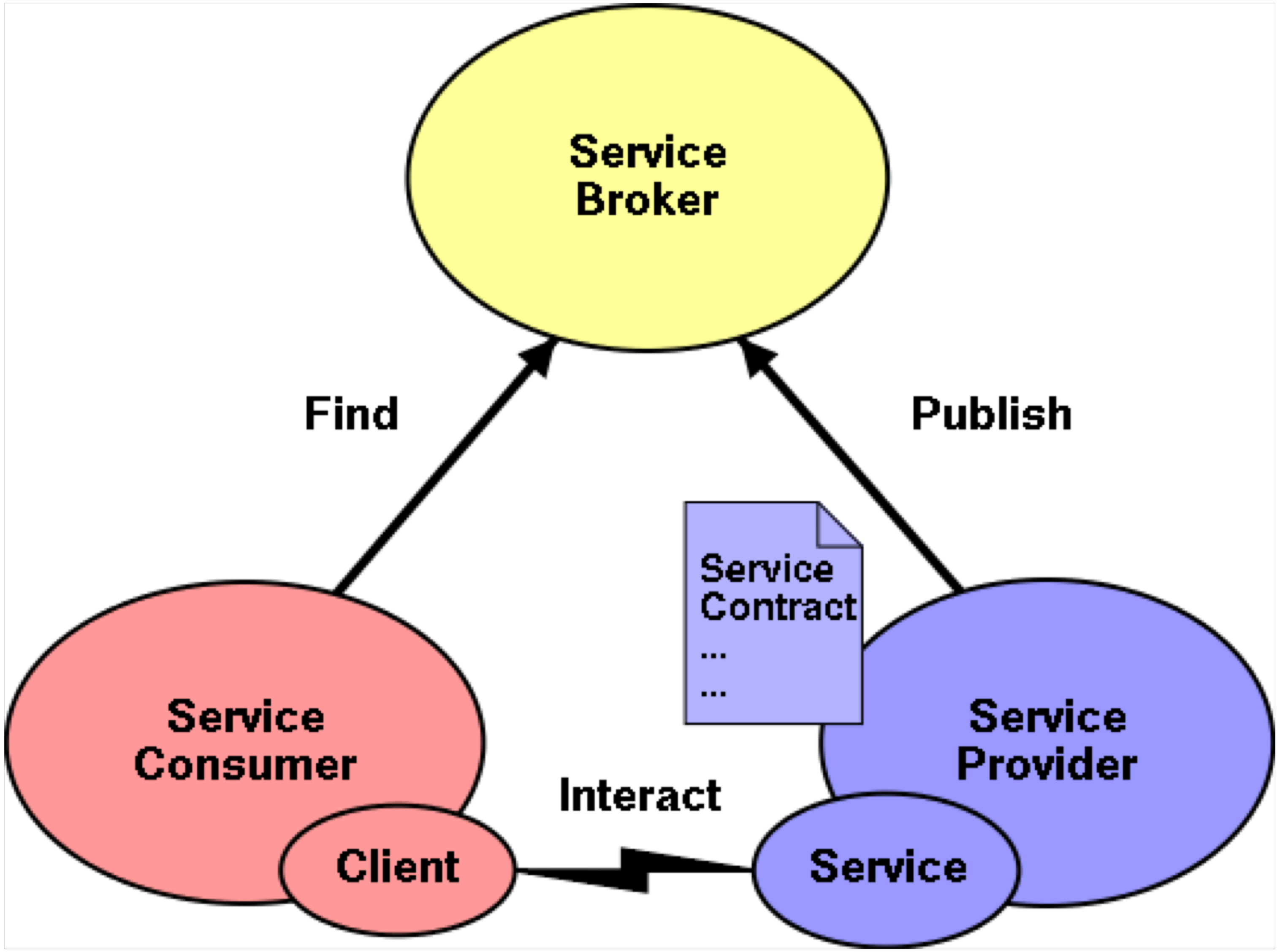


Web Services

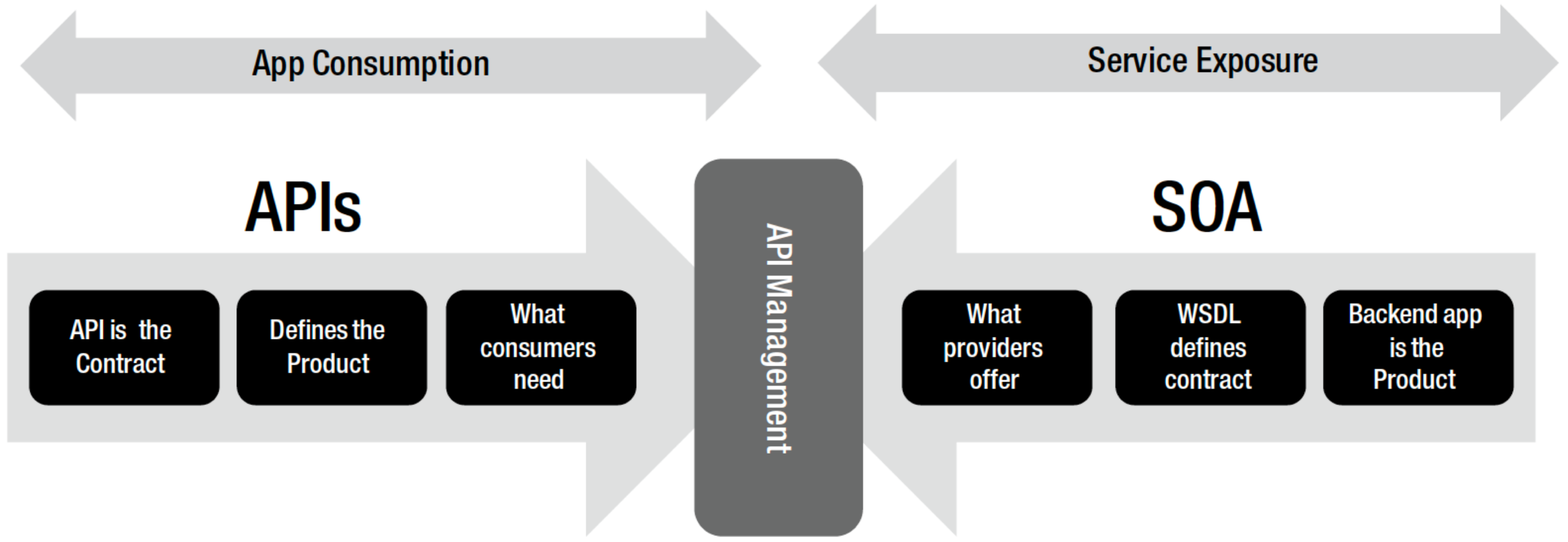


RESTful WS

The SOA style

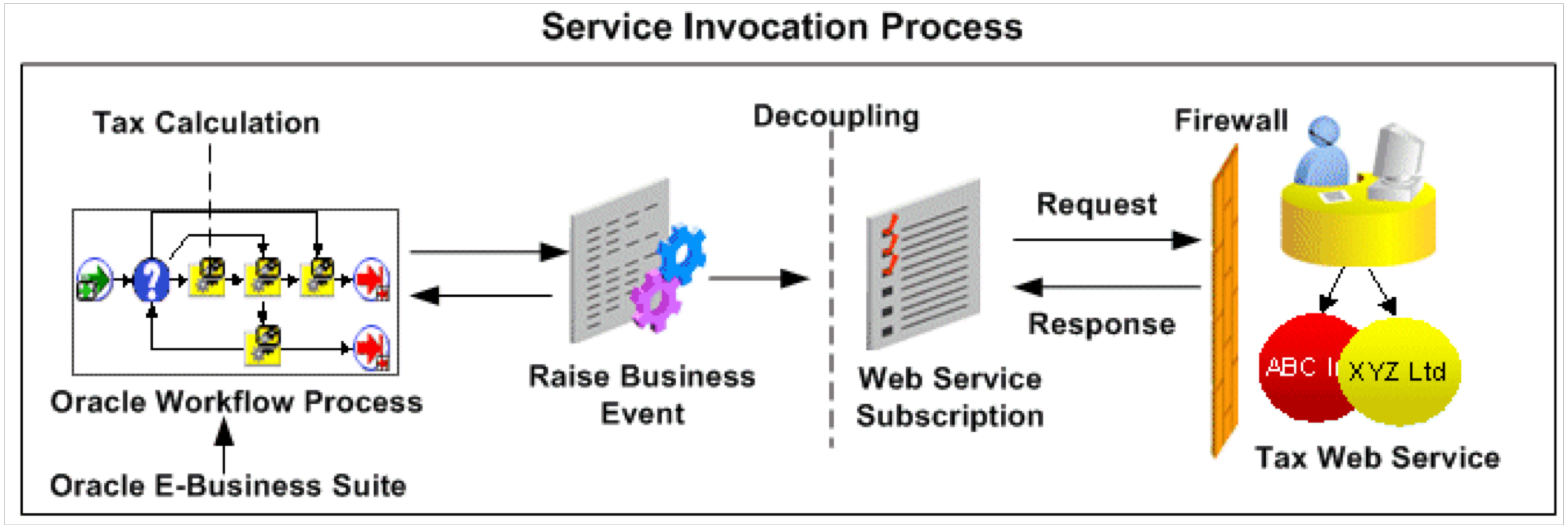


API vs SOA

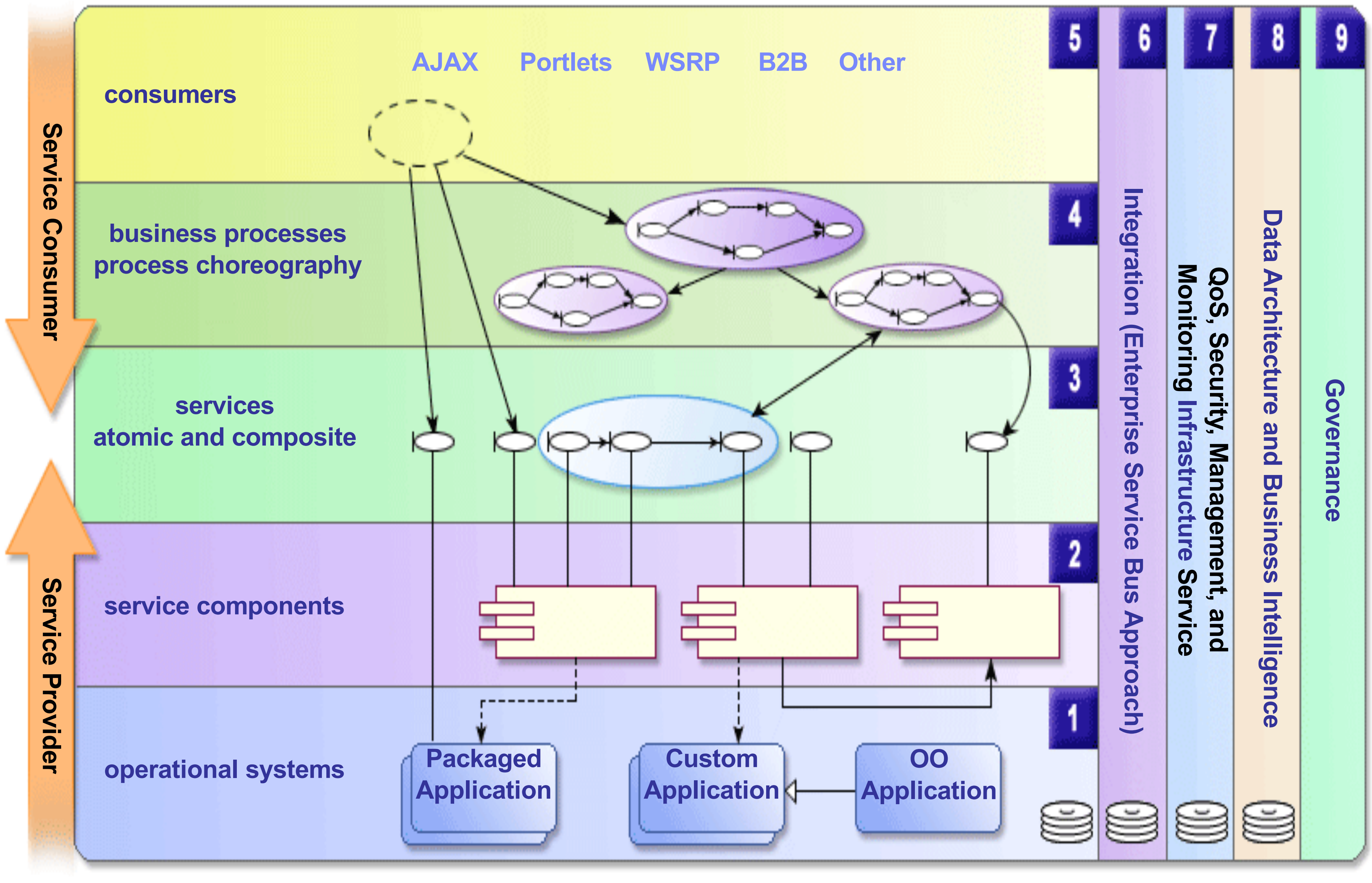


- API technology focuses on the consumption of the back-end services created using SOA principles.
- APIs can be thought of as an evolution of SOA: creating and exposing reusable services.
- The main difference between them is that APIs are focused more on making consumption easier, whereas SOA is focused on control and has an extensive and well-defined description language

SOA: example


Service Invocation Process
Tax Calculation
Decoupling
Firewall
Request
Response
Raise Business Event
Web Service Subscription
Oracle Workflow Process
Oracle E-Business Suite
ABC I
XYZ Ltd
Tax Web Service

SOA technology stack



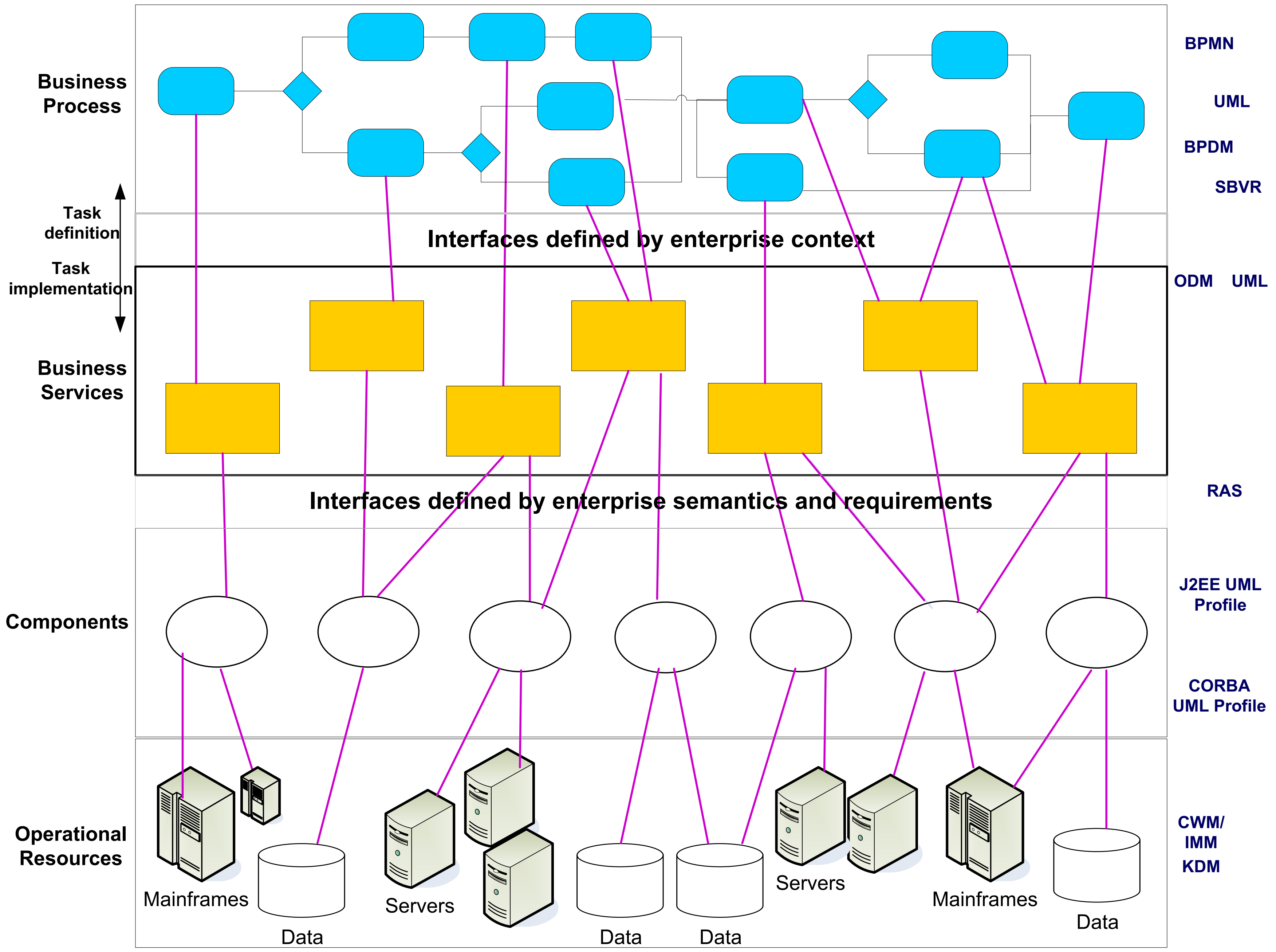
Main SOA Standards (the SOA soup)

- **XML** (Extensible Markup Language): a data markup language for Web services
- **SOAP** (Simple Object Access Protocol): a W3C-approved standard for exchanging information among applications
- **WSDL** (Web Service Description Language): a W3C-approved standard for using XML to define Web services
- **WADL** (Web ApplicationDescription Language): is the REST equivalent of WSDL, WADL is based on XML and models the resources provided by a service and the relationships between them
- **UDDI** (Universal Description, Discovery, and Integration): an OASIS-approved standard specification for defining Web service registries
- **WS-Reliability** (Web Services Reliability): a SOAP-based protocol for exchanging SOAP messages, with delivery and message-ordering guarantees
- **WS-Security** (Web Services Security): a SOAP-based protocol that addresses data integrity, confidentiality, and authentication in Web services
- **JEE**: the Java Platform, Enterprise Edition, with APIs for deploying and managing Web services
- **WSIF** (Web Services Invocation Framework): an open source standard for specifying, in WSDL, EJB implementations for the Web server
- **WSRP** (Web Services for Remote Portlets): an OASIS standard for integrating remote Web services into portals
- **BPEL** (Business Process Execution Language): a standard for assembling sets of discrete services into an end-to-end business process

Standardizing bodies for SOAs

- **W3C** (established 1994)
- **OASIS** (1993), consortium of former GML providers, deals with applications using XML
- **OMG** (1989)
- **WS-I** (2002), promotes interoperability among the stack of Web Services specifications

OMG Standards for SOA



Typical issues in SOA

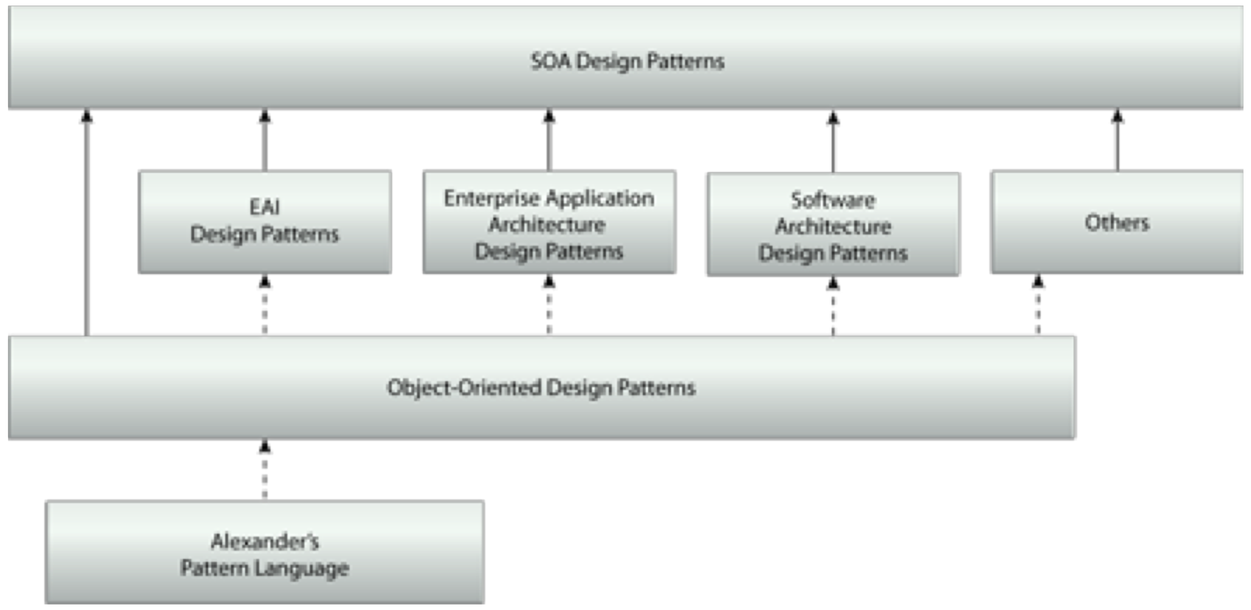
- Model, design, and implement a SOA
- Automate business processes by mapping them to the architectural model
- Orchestrate services and execute processes with the Business Process Execution Language (BPEL)
- Choreography describes a global protocol governing how individual participants interact with one another
- Achieve interoperability within a SOA using proven standards and best practices
- Secure and govern an enterprise SOA

Elements of SOA Design

- Business Modeling
- Service Oriented Architectural Modeling and Design
- Model Driven assumptions (loose coupling)
- Distributed objects and MOM (Message Oriented Middleware) for component-based sw systems

SOA design patterns http://soapatterns.org

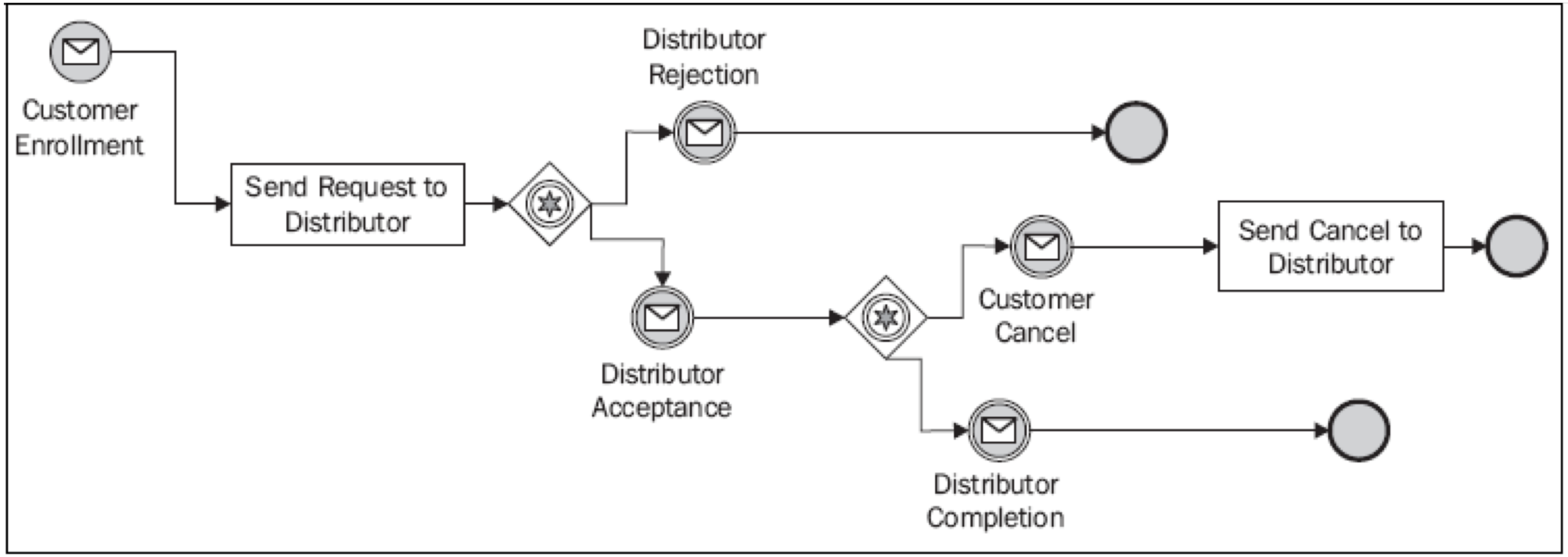
- Service-orientation has deep roots in distributed computing platforms,
- Many SOA design patterns can be traced back to established design concepts, approaches, and previously published design pattern catalogs


SOA Design Patterns
EAI Design Patterns
Enterprise Application Architecture Design Patterns
Software Architecture Design Patterns
Others
Object-Oriented Design Patterns
Alexander's Pattern Language

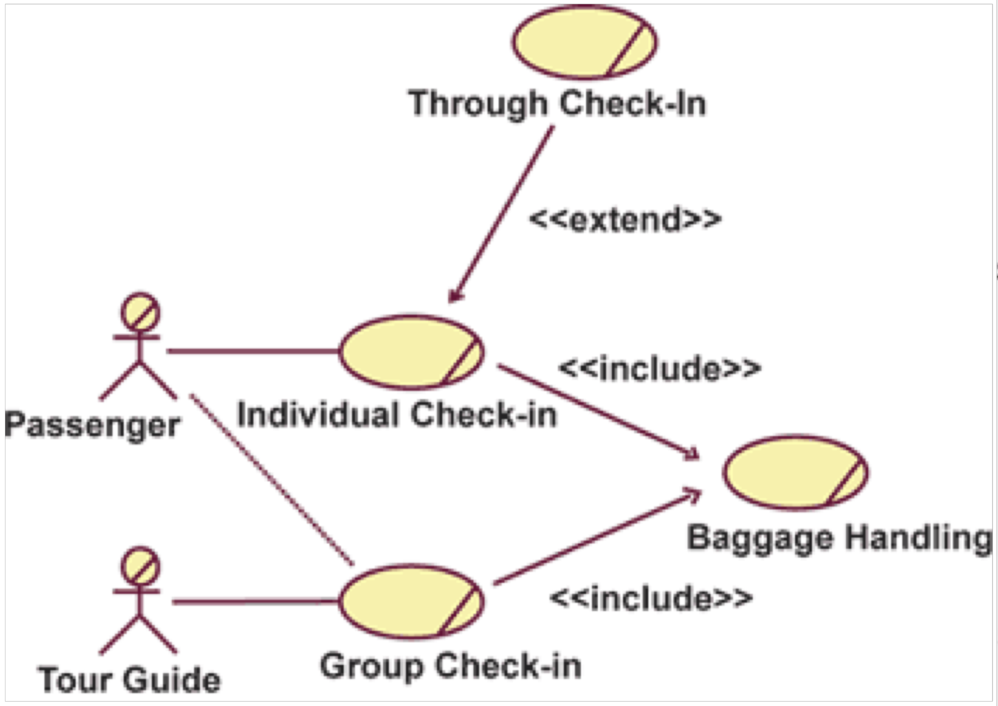
Historical influences on SOA patterns

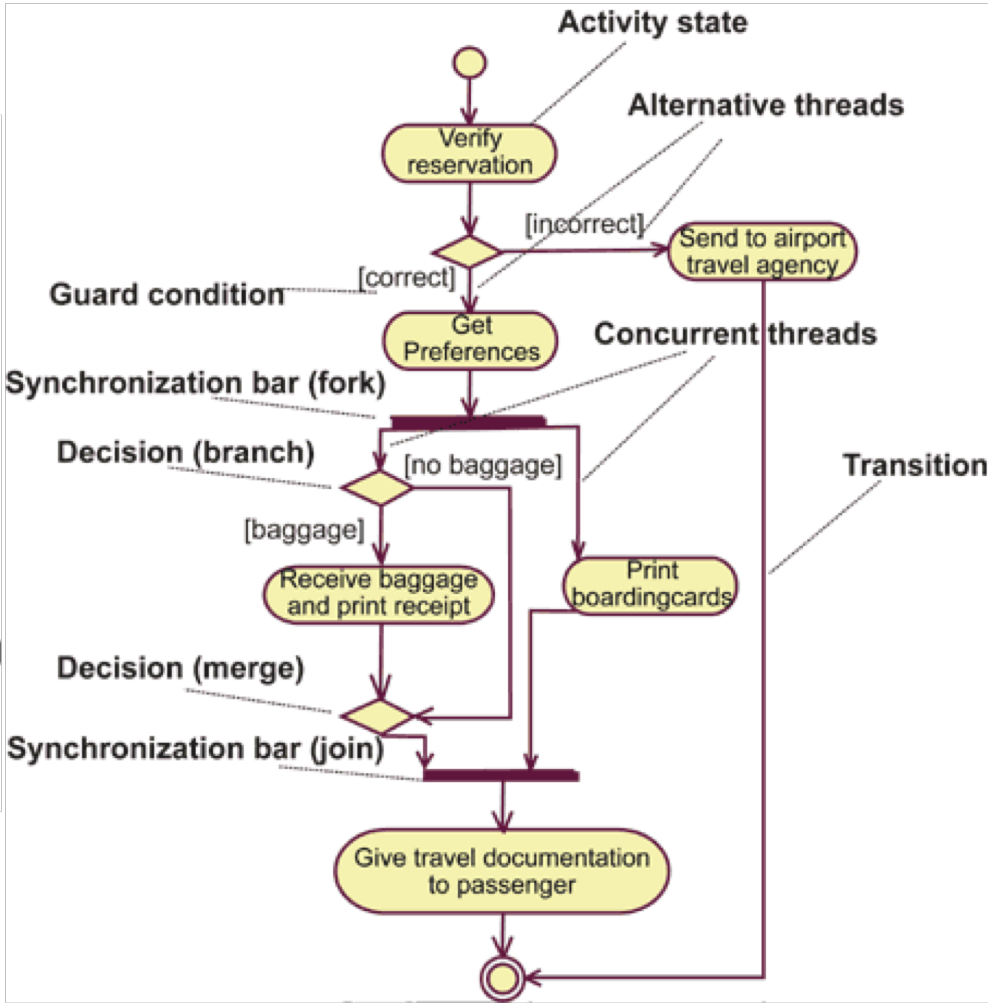
Orchestration and choreography

- An orchestration process has public and private activities
- The public activities are those required by the choreography
- Private activities are there to meet internal requirements, but are not visible to partners
- The figure shows the public activities of the orchestration process for an energy retailer



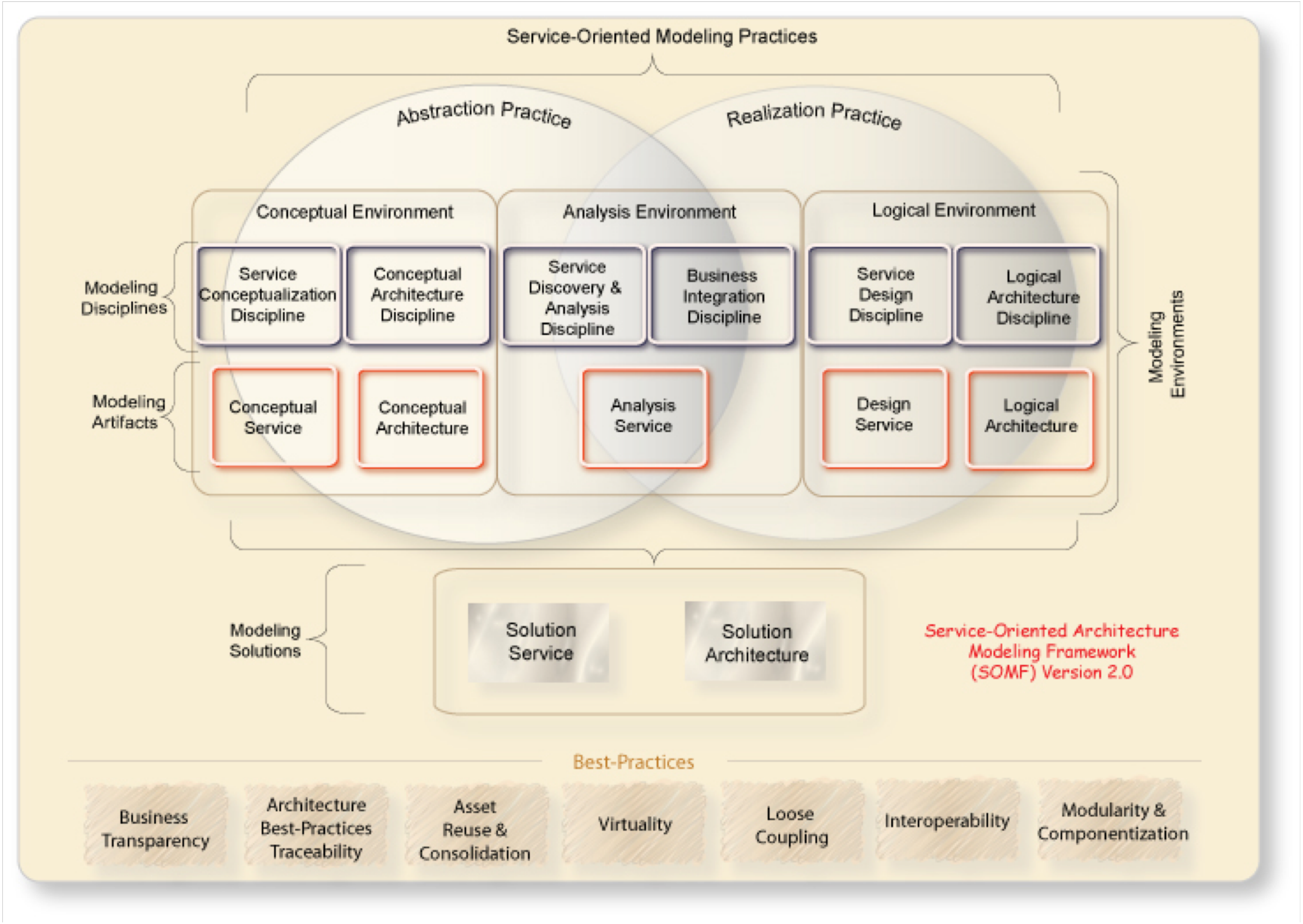
Business modeling



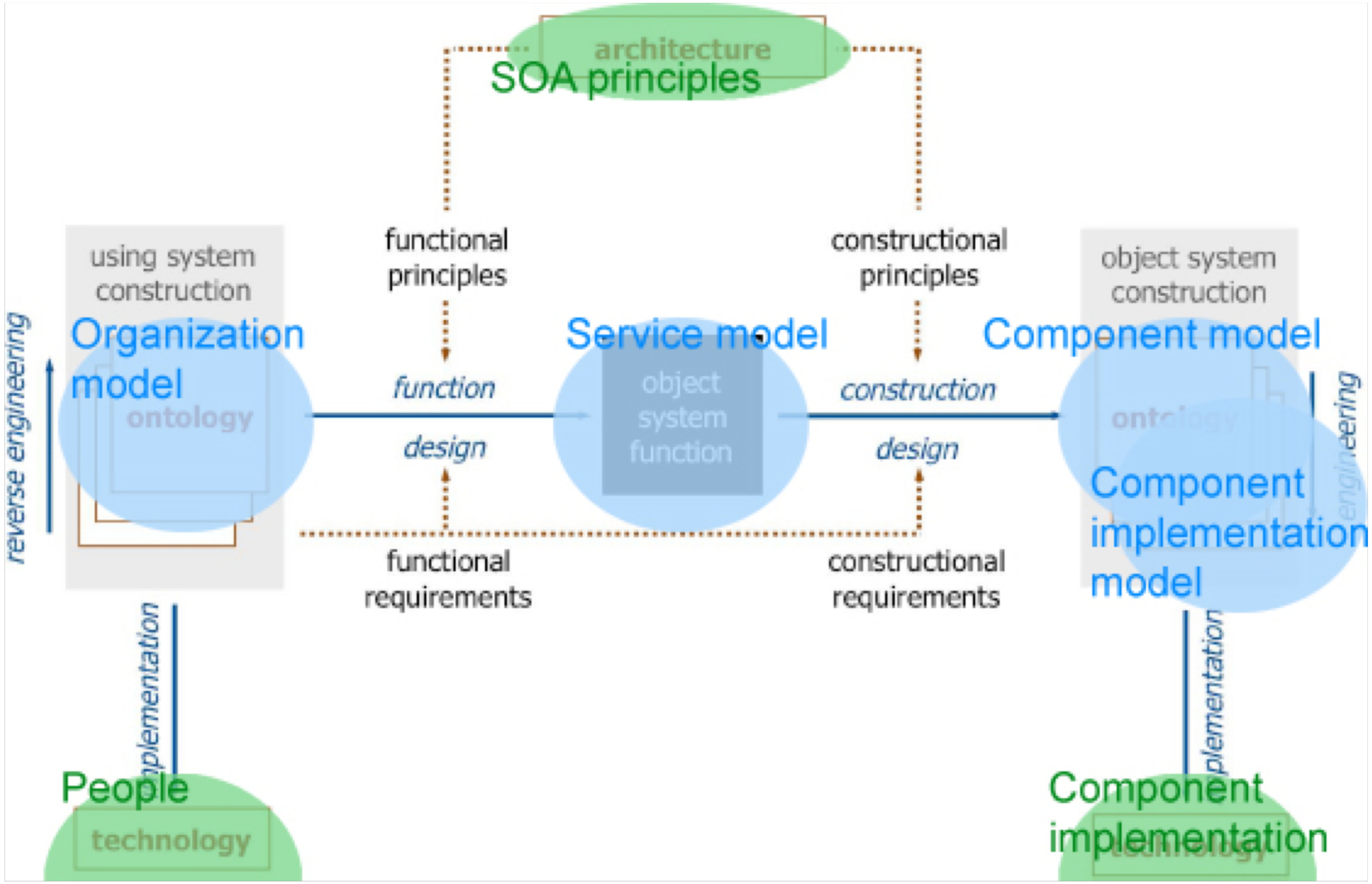


http://www.ibm.com/developerworks/rational/library/360.html

Service oriented modeling and architecting


Service-Oriented Modeling Practices
Abstraction Practice
Realization Practice
Conceptual Environment
Analysis Environment
Logical Environment
Modeling Disciplines
Modeling Artifacts
Modeling Environments
Service Conceptualization Discipline
Conceptual Architecture Discipline
Service Discovery & Analysis Discipline
Business Integration Discipline
Service Design Discipline
Logical Architecture Discipline
Conceptual Service
Conceptual Architecture
Analysis Service
Design Service
Logical Architecture
Modeling Solutions
Solution Service
Solution Architecture
Service-Oriented Architecture Modeling Framework (SOMF) Version 2.0
Best-Practices
Business Transparency
Architecture Best-Practices Traceability
Asset Reuse & Consolidation
Virtuality
Loose Coupling
Interoperability
Modularity & Componentization

Model driven SOA design

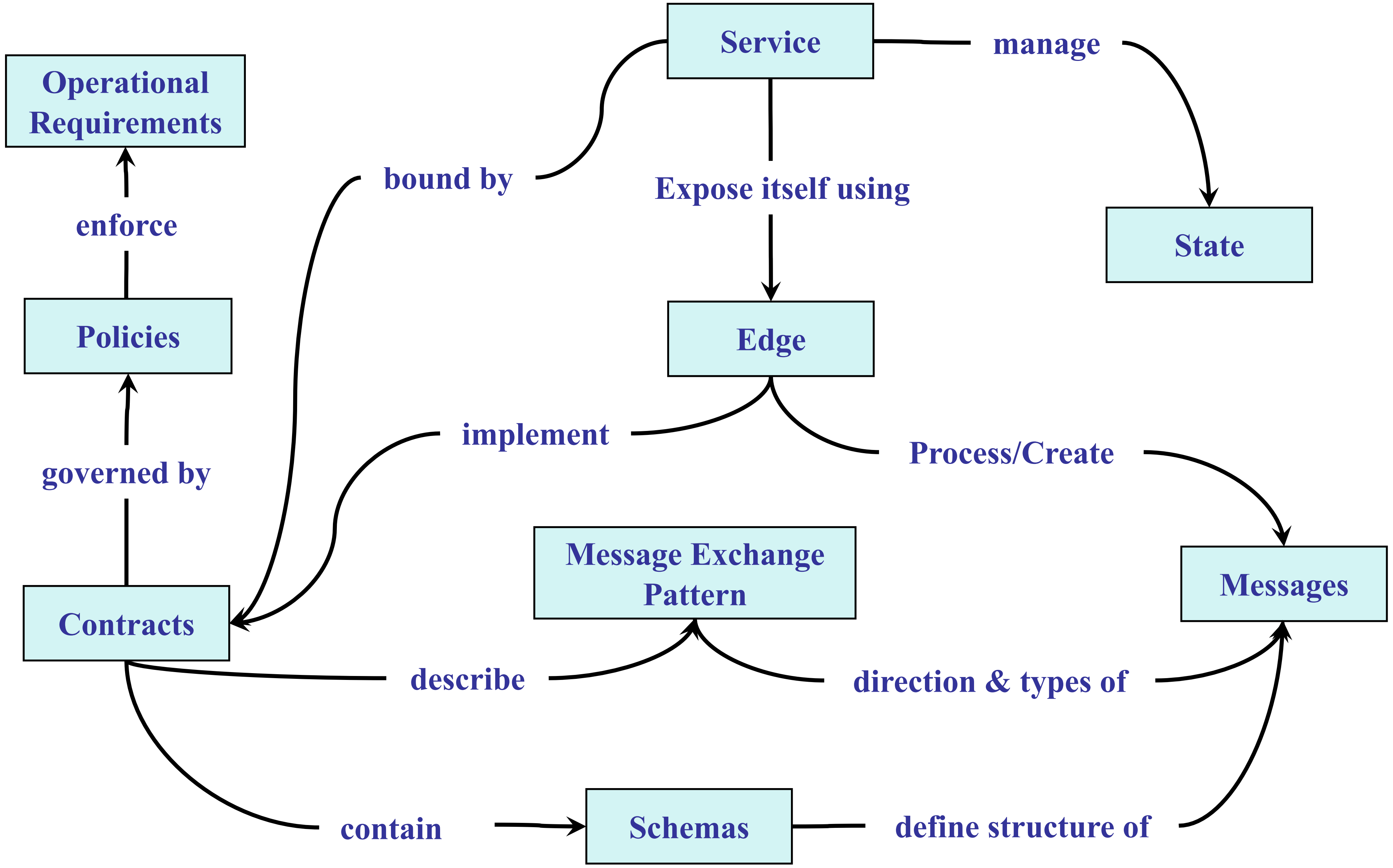


www.theenterprisearchitect.eu/archive/2009/06/03/a-framework-for-model-driven-soa

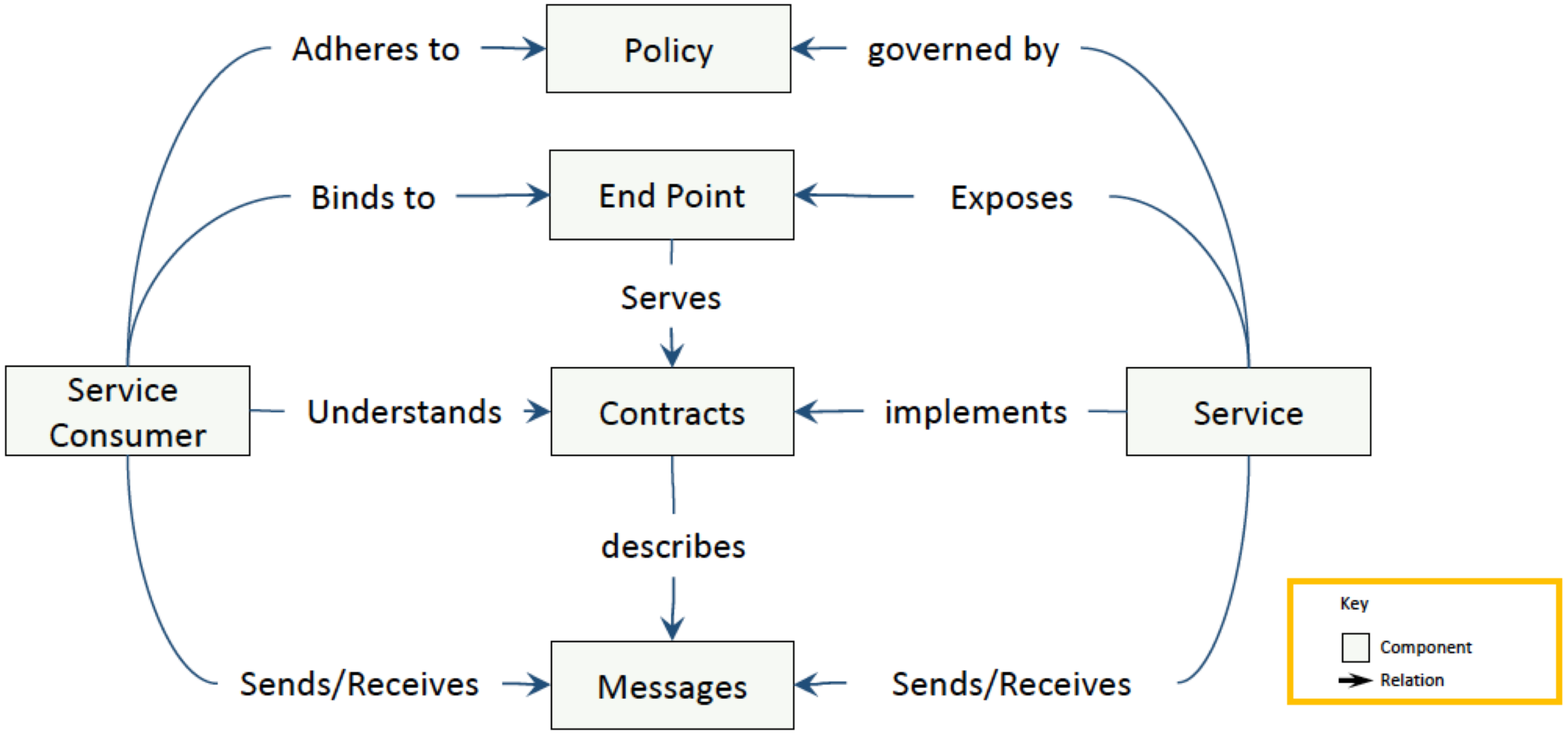
Services and SOAs

- A **service** is a program interacting via message exchanges
  - Using Web Services all messages and service descriptions are written in XML
- A SOA is a set of **deployed** services cooperating in a given task
  - Adapt to new services after deployment

SOA Concepts



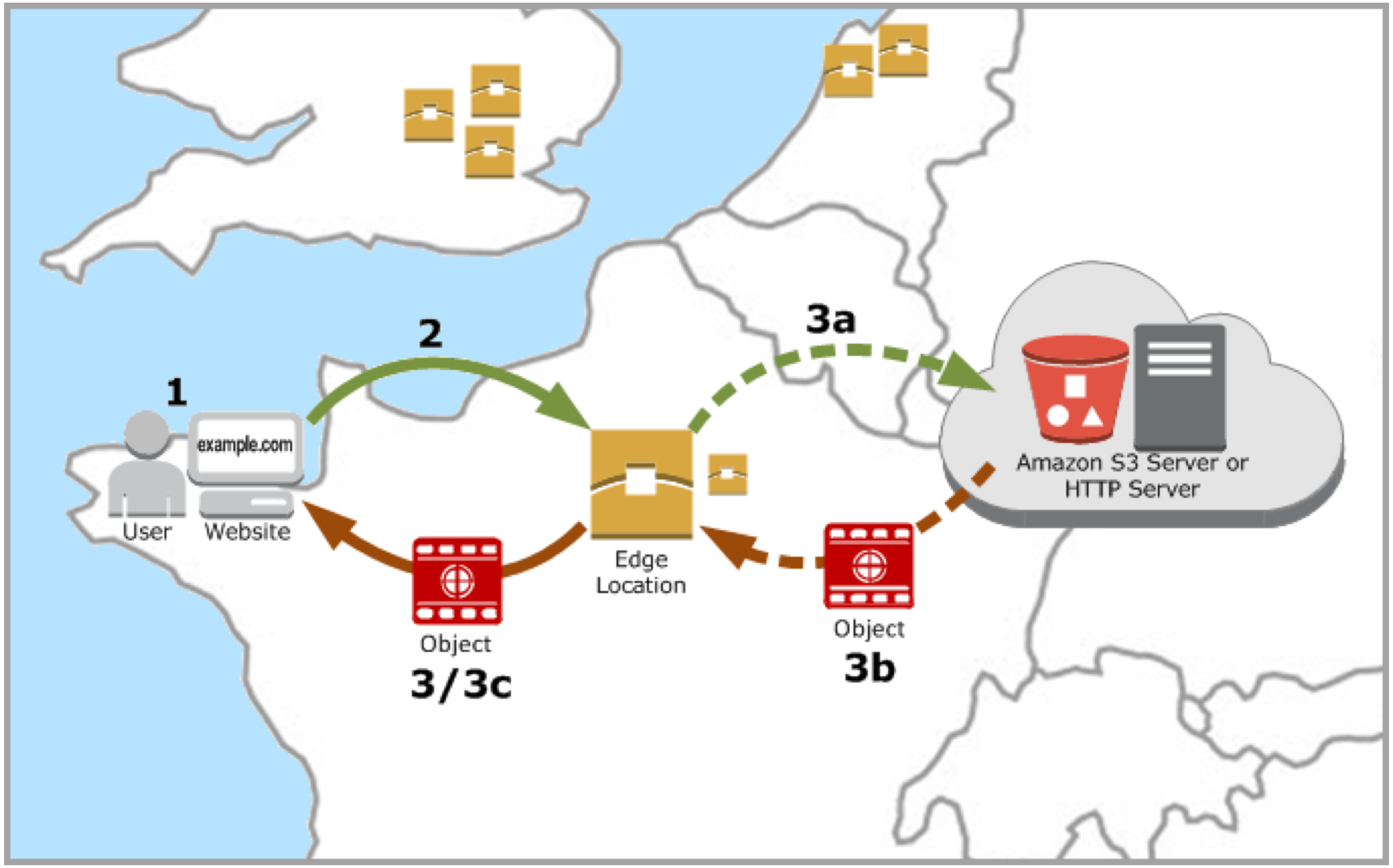
SOA: main components and relations



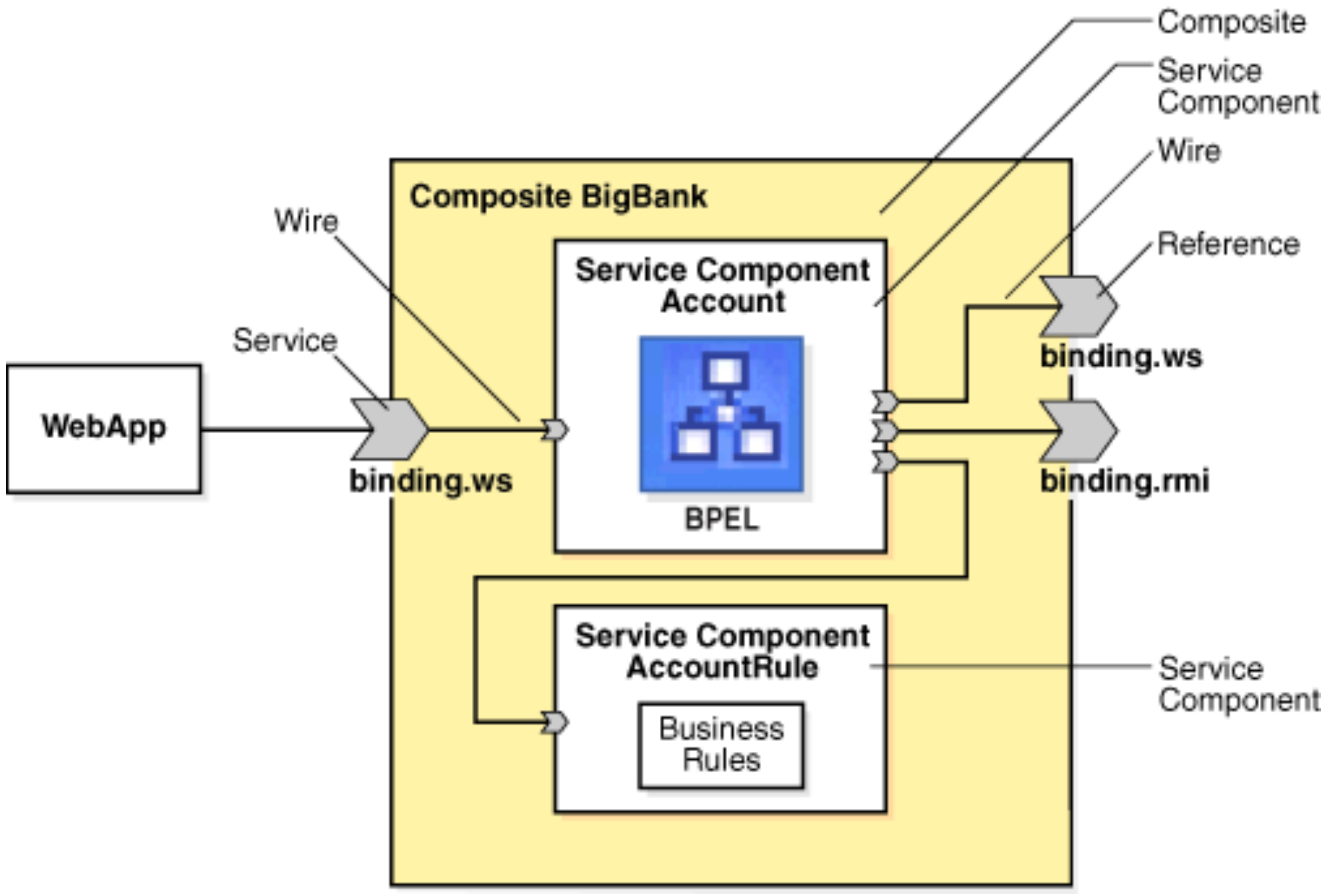
The essence of SOA

- **Use other services as RPC servers for your app**
- Web 1.0: large sites organized this way *internally*
  - Yahoo!, Amazon, Google, …: *external* “Services” available, but complex: Doubleclick ads, Akamai
- XML based Web Services msgs and descriptions
- Web 2.0: public service API’s
  - Services: Google API, Amazon CloudFront...
  - Platforms: Facebook, Google Maps, ...
  - Mashups, e.g. housingmaps.com
  - User-composable services, e.g. Yahoo Pipes

Example: Amazon Cloudfront



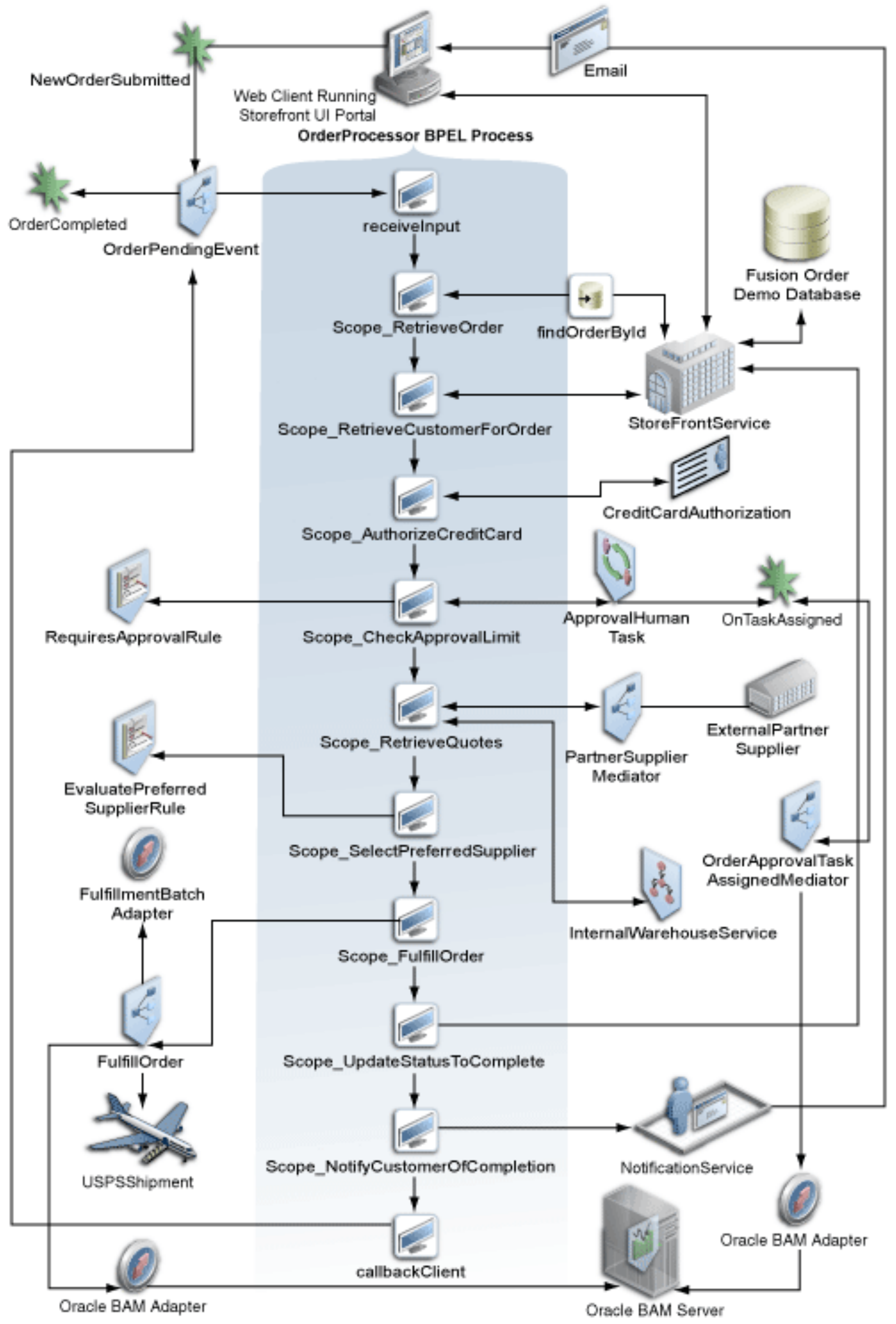
Simple SOA architecture



http://docs.oracle.com/cd/E21764_01/integration.1111/e10224/fod_intro1.htm#CHDEBGCJ

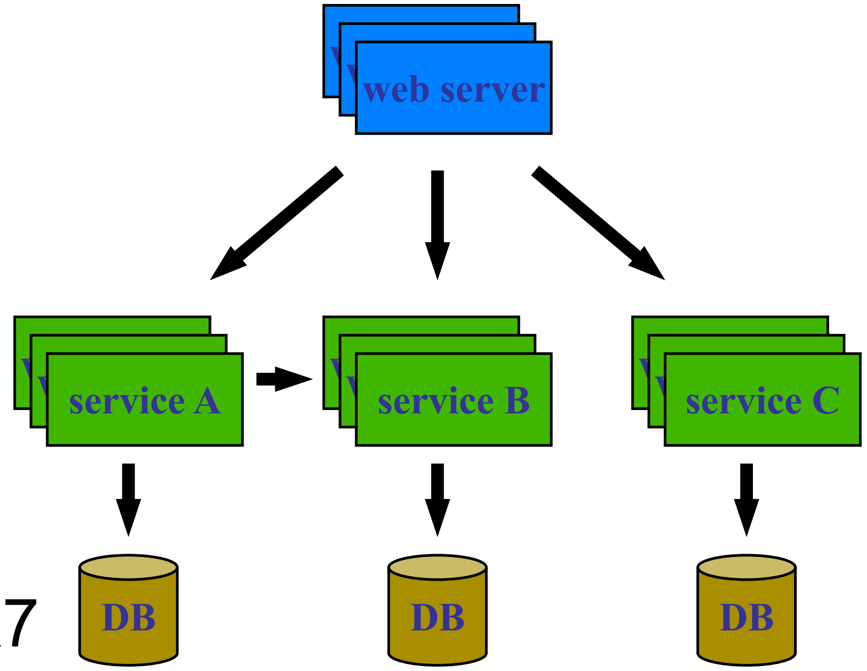
A workflow based on a SOA

- This workflow shows how services, both internal to an enterprise, and external at other sites, can be integrated using a SOA architecture to create an ordering system.
- The BPEL process orchestrates all the services in the enterprise for order fulfillment with the right warehouse, based on the business rules in the process



Example Web 1.0 SOA: amazon.com

- ~50 “two-pizza” teams of “developer/operators”
- ~10 operators
  - monitor the whole site
  - page the resolvers on alarm
- ~1000 resolvers
  - 10-15 per team, 1 on-call 24x7
  - monitor own service, fix problems
- Over 140 code change commits/month
- SOA (like Yahoo, Google, others)



P. Bodík et al., *Advanced Tools for Operators at Amazon.com,* Proc. ICAC 2005

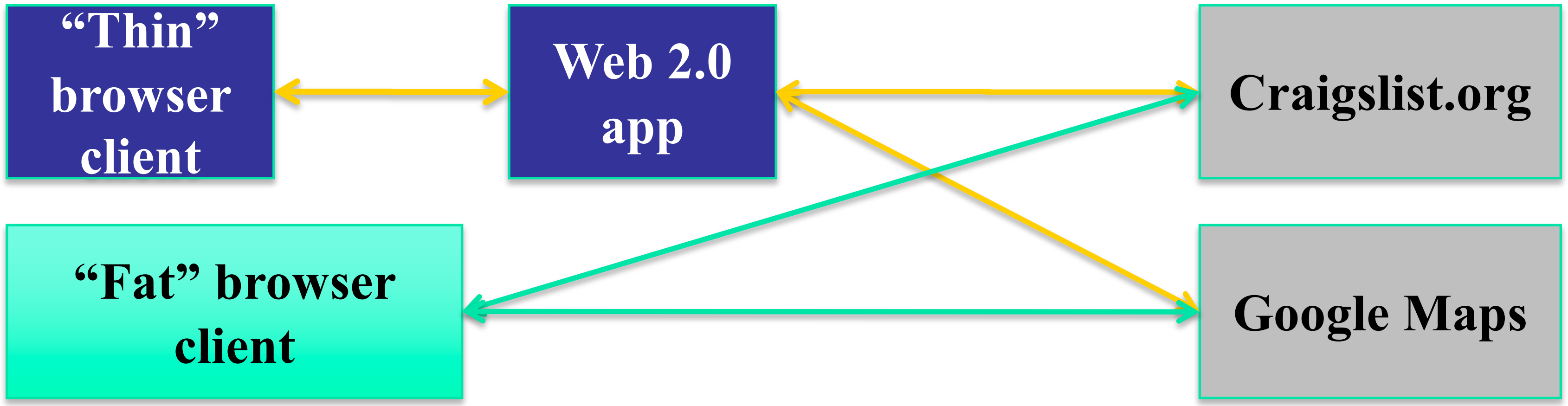
SOA based on RPC

- Transport: HTTP(S)
- Data interchange:
  - XML DTD (e.g., RSS)
  - JSON (Javascript Object Notation)
- Request protocol:
  - SOAP (Simple Object Access Protocol)
  - JSON-RPC
- See also WebHooks (HTTP POST callback, for "push") `www.webhooks.org`

AJAX vs SOA

- AJAX: client ⇔ server
  - A client makes async requests to a HTTP server
  - client-side JavaScript upcall receives reply and decides what to do
  - response includes XHTML/XML to update page, or JavaScript to execute
- SOA: server ⇔ server or client ⇔ server
  - An initiator makes (sync or async) requests to an HTTP server
  - In the past, initiator was a server running some app
  - today, JavaScript clients can exploit this approach

Two ways to do it: thin or fat clients



+ Client portability

+/– Client performance (both app download & JavaScript execution)

+ Availability of utility libraries for app development

– Privacy/trustworthiness of aggregator app

– Caching

REST (Representational State Transfer) style

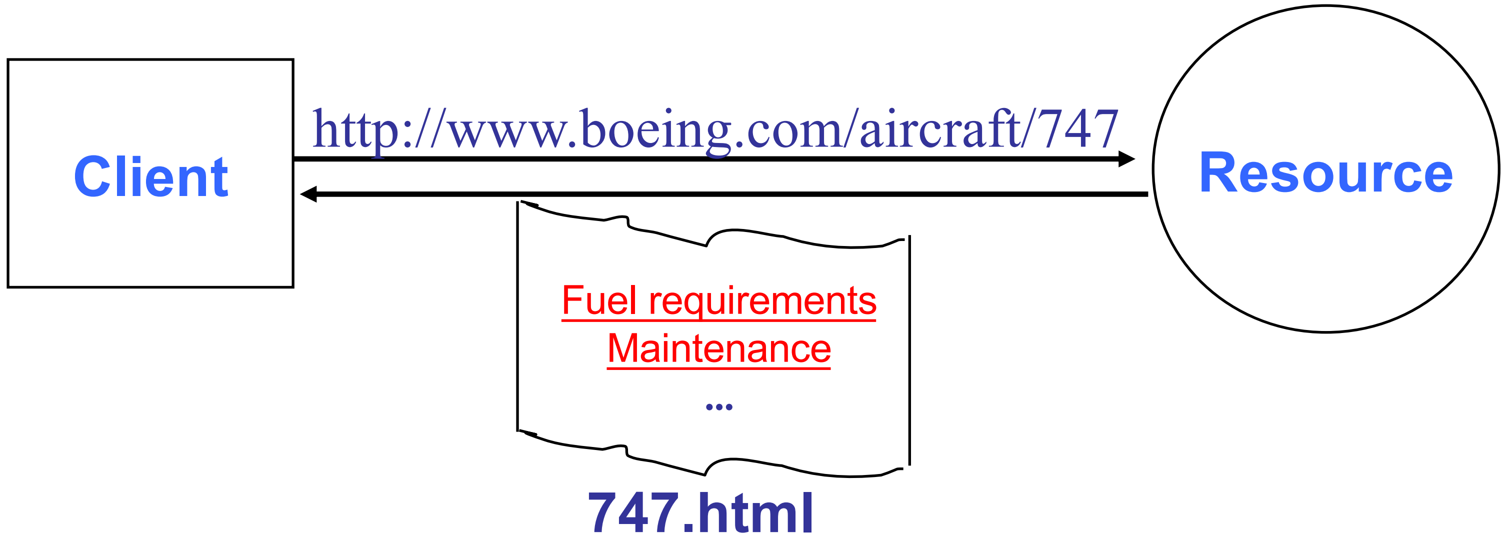
- Architectural style:
  - Client-server, stateless, cache
  - description of properties that made Web 1.0 successful by constraining SOA interactions
- In context of SOA for Web 2.0
  - HTTP is transport; HTTP methods (get, put, etc.) are the only commands
  - URI names are a resource
  - Client has resource ⇔ has enough info to request *modification* of the resource on server
  - A cookie can encode part of transferred state
- If an app is RESTful, it is easy to "SOA"-ify

REST style

- Representation State Transfer (REST) was introduced by R.Fielding to describe an **architectural style** of networked software systems
- REST prescribes how a well-designed Web application behaves: a net of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for use."

http://www.ebuilt.com/fielding/pubs/dissertation/top.htm

Why is it called "Representation State Transfer"?



The Client references a Web resource using a URL. A representation of the resource is returned (in this case as an HTML document).
The representation (e.g., Boeing747.html) places the client application in a state.
The result of the client traversing a hyperlink 747.html is another resource is accessed.
The new representation places the client application into yet another state.
Thus, the client application changes (transfers)
state with each resource representation --> Representation State Transfer!

Motivation for REST

"The motivation for developing REST was to create
an architectural model for how the Web should work,
such that it could serve as a framework for the Web protocol standards.

REST has been applied to describe the desired Web architecture,
help identify existing problems, compare alternative solutions,
and ensure that protocol extensions
would not violate the core constraints that make the Web successful."

- Roy Fielding

REST with HTTP examples

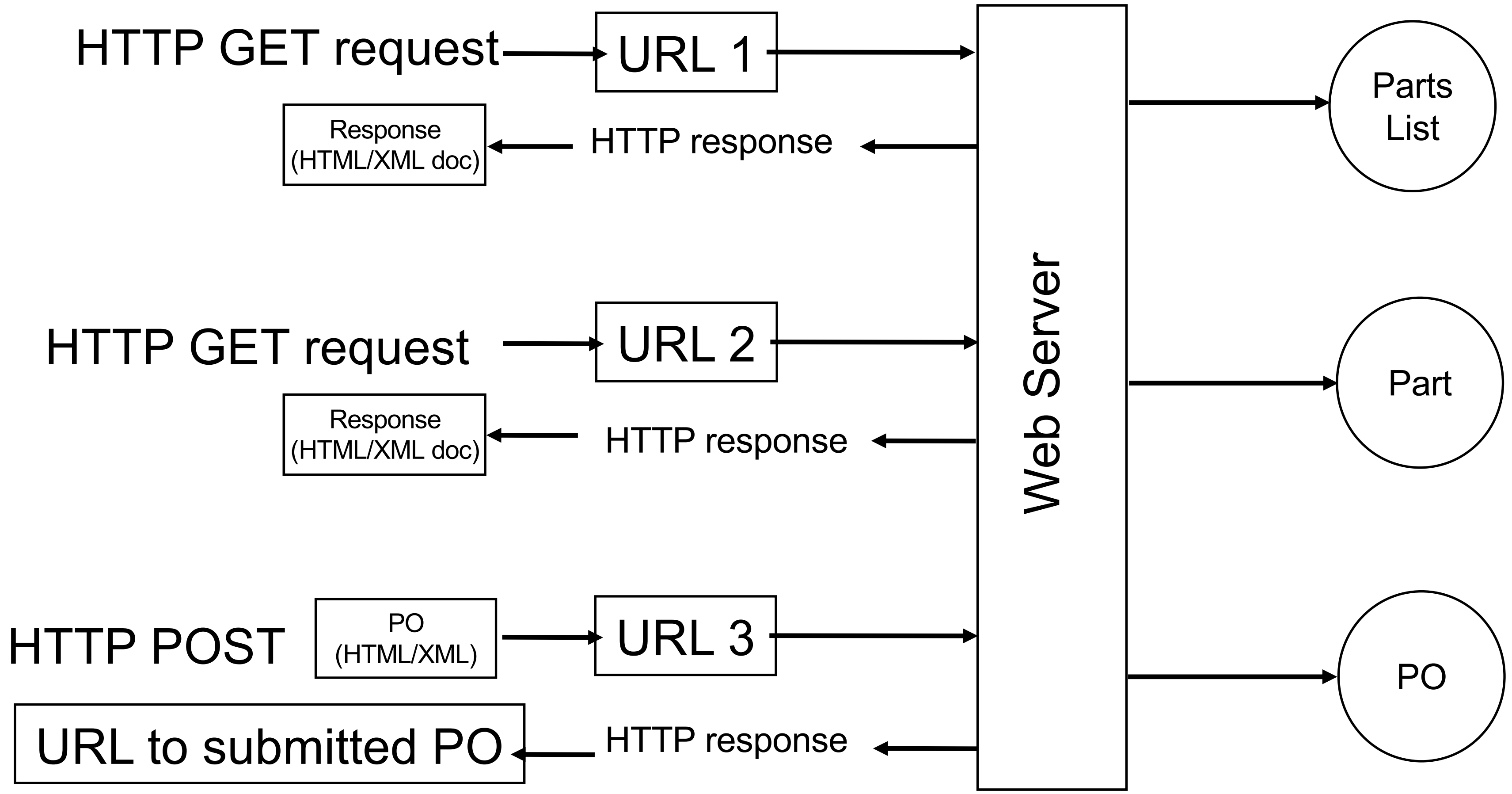
HTTP GET	HTTP PUT	HTTP POST	HTTP DELETE
Collection URI, such as `http://example.com/customers/257/orders`			
List the members of the collection, complete with their member URIs for further navigation	Replace the entire collection with another collection	Create a new entry in the collection. The ID created is usually included as part of the data returned by this operation.	delete the entire collection
HTTP GET	**HTTP PUT**	**HTTP POST**	**HTTP DELETE**
Element URI, such as `http://example.com/resources/7HOU57Y`			
Retrieve a representation of the addressed member of the collection in an appropriate MIME type	Update (or create) the addressed member of the collection	Treats the addressed member as a collection in its own right and creates a new subordinate of it.	Delete the addressed member of the collection.

REST vs SOAP: example

- A company deploying 3 Web services to enable its customers to:
  - get a list of parts
  - get detailed information about a particular part
  - submit a Purchase Order (PO)

- the REST solution first, then the SOAP solution

The REST way of Implementing the Web Services



Implementing a Web Service using SOAP

- Service: Get detailed information about a particular part
  - The client creates a SOAP document that specifies the procedure desired, along with the part-id parameter.

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>

      <p:getPart xmlns:p="http://www.parts-depot.com">
           <part-id>00345</part-id>
      </p:getPart>

   </soap:Body>
</soap:Envelope>
```

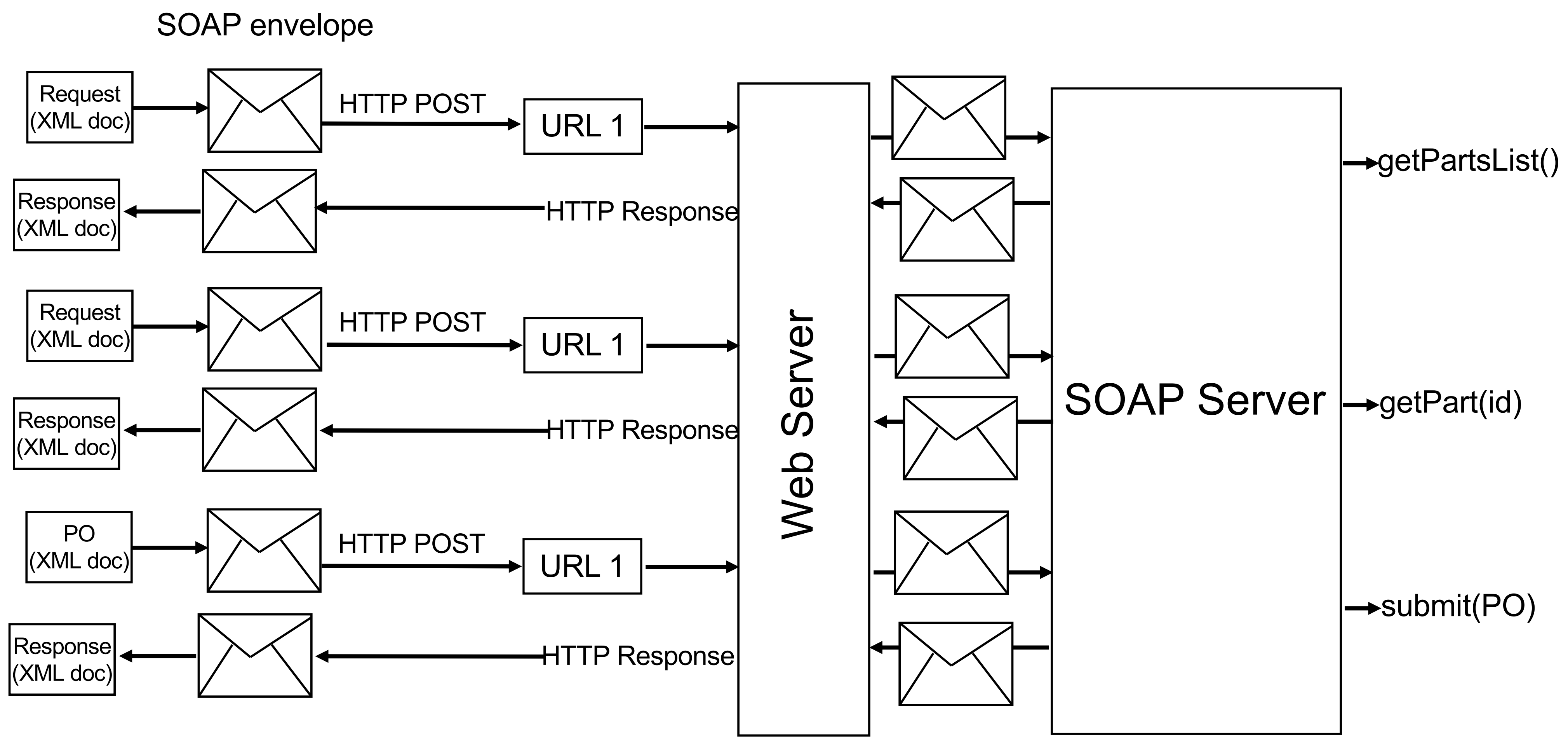
the client will HTTP POST this document to the SOAP server at:

`http://www.parts-depot.com/soap/servlet/messagerouter`

Note that this is the same URL as was used when requesting the parts list.

The SOAP server peeks into this document to determine what procedure to invoke.

Implementing the Web Services using SOAP



Note the use of the same URL (URL 1) for all transactions.
The SOAP Server parses the SOAP message to determine which method to invoke.
All SOAP messages are sent using an HTTP POST.

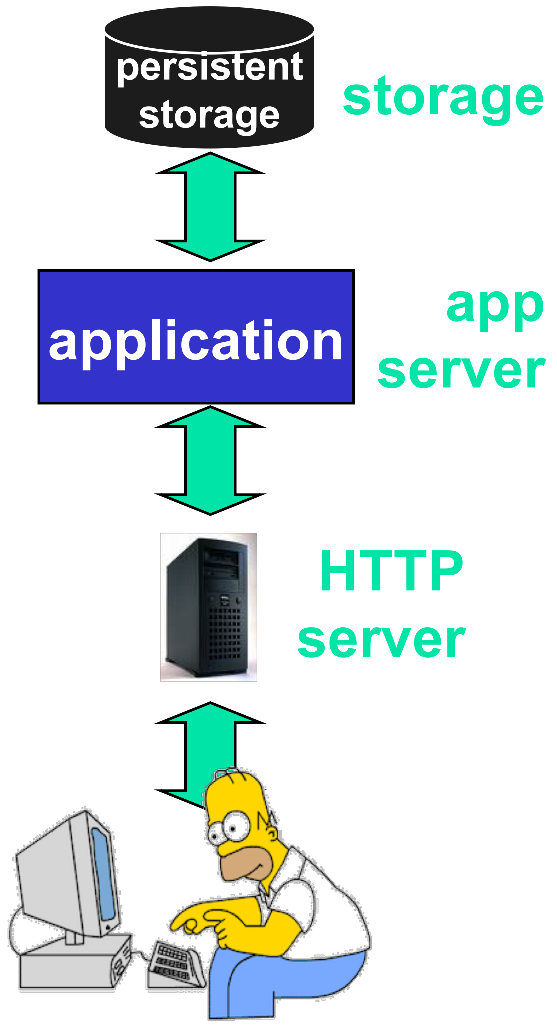
Dynamic content generation

- Most Web 1.0 (e-commerce) sites actually *run a program* that generates the output
- Originally: templates with embedded code “snippets”
- Eventually, embedded code became “tail that wagged the dog” and moved out of the Web server
- Languages/frameworks evolved to capture common tasks
  - Perl, PHP, Python, Ruby on Rails, ASP, ASP.NET, Java Servlet Pages, Java Beans/J2EE, ...

SaaS 3-tiers architecture

- *Common gateway interface* (cgi): allows a Web server to run a program
  - Server maps some URI's to application names
  - When the app is run, it gets the complete HTTP request including headers
- "Arguments" embedded in URL with "&" syntax or sent as request body (with POST)

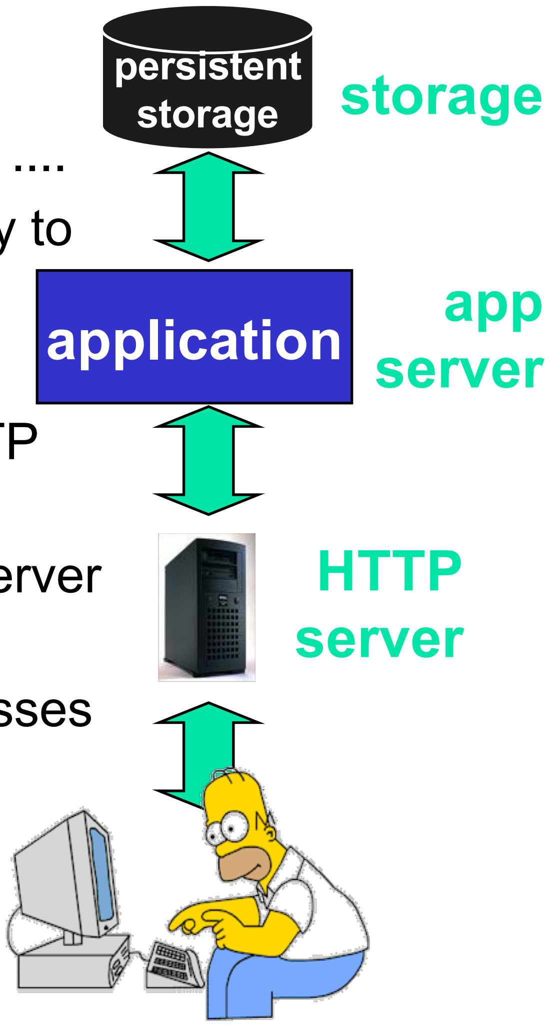
  http://www.foo.com/search?term=white%20rabbit&show=10&page=1
- App generates the entire response
  - content (HTML? an image? some javascript?)
  - HTTP headers & response code
- Plug-in *modules* for Web servers allow long-running CGI programs & link to language interpreters



- **Various *frameworks* have evolved to capture this structure**

SaaS 3-tiers deployment

- **HTTP server** ("web server")
  - "fat" (e.g. Apache): support virtual hosts, plugins for multiple languages, URL rewriting, reverse proxying, ...
  - "thin" (*nginx, thin,* Tomcat, ...): bare-bones machinery to support *one* language/framework; no frills
- **Application server**
  1. separate server process, front-ended by a "thin" HTTP server
  2. **or** linked to an Apache worker via FastCGI or web server plug-in: mod_perl, mod_php, mod_rails, ...
  - Apache can spawn/quiesce/reap independent processes
- **Persistent storage**
  - Common RDBMS (MySQL, PostgreSQL, etc.)
  - communicate w/app via proprietary or standardized database "connector" (ODBC, JDBC, ...)
- Hence **LAMP**: Linux, Apache, MySQL, PHP/Perl



Social Computing

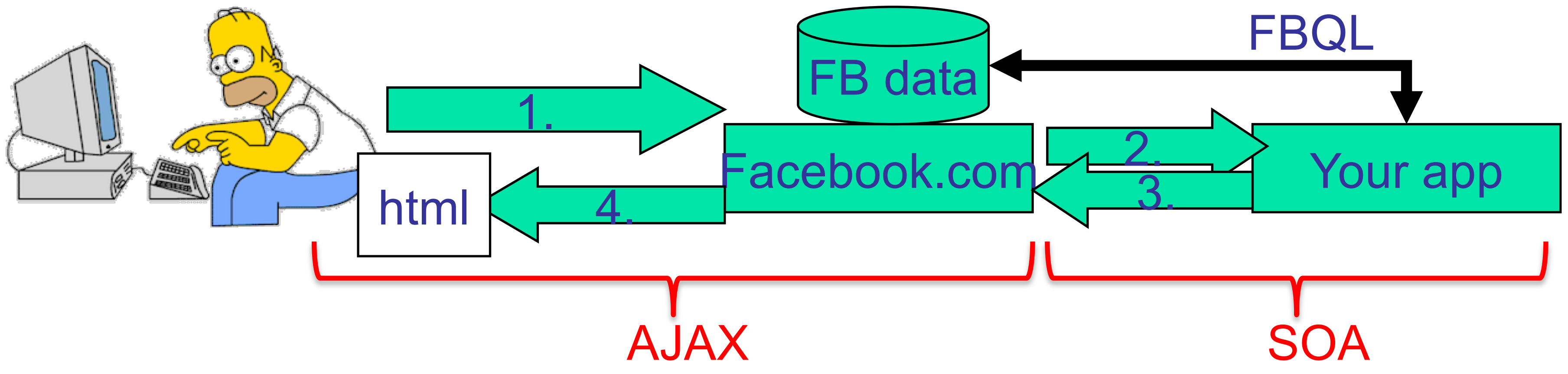
- Web 1.0: add value via mass customization
  - select content for you based on best guesses about your interests
  - resource: demographic/analytic data about users
- Web 2.0: add value via connecting to social network
  - vendor: your friends' interests are a good indicator of your interests
  - user: value added to existing content == how *your friends* interact with it
  - resource: social networks
- From social networking *site* to social network as a way of structuring applications

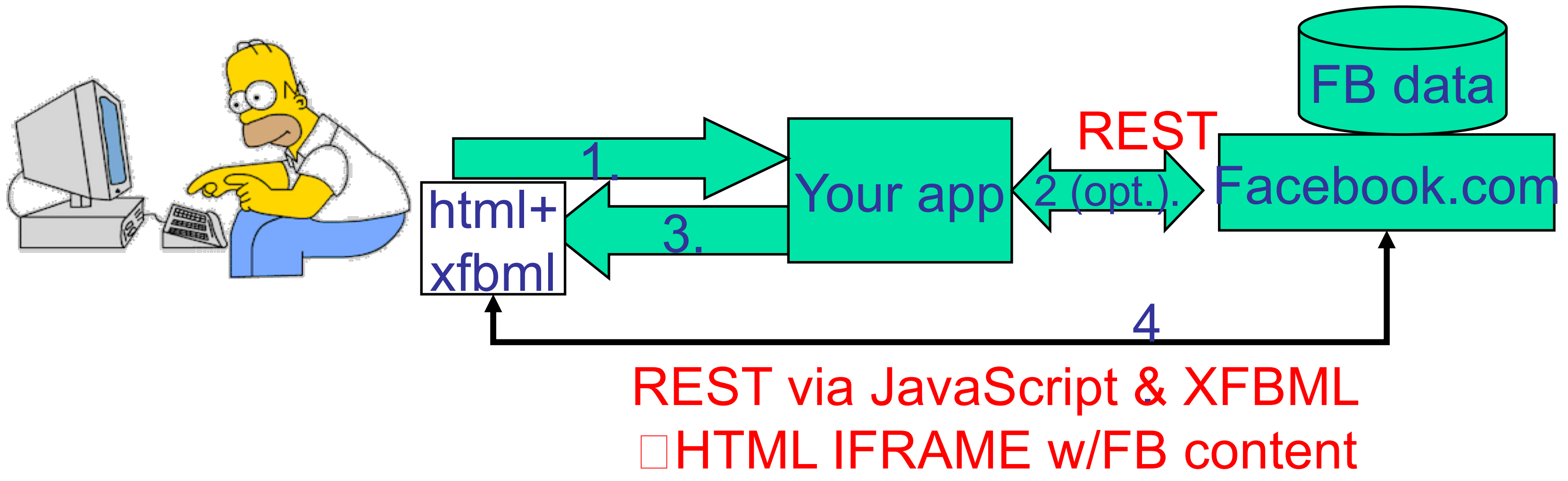
Social Computing

- Amount of content “created” by each user small
  - e.g., tweets, rate video, play a Facebook game
- but still creates lots of short random writes
  - consider “Like” feature on Facebook
  - social graphs hard to partition
- current developers should not ask *whether* social computing is part of their app, but *how*

Facebook “connect”

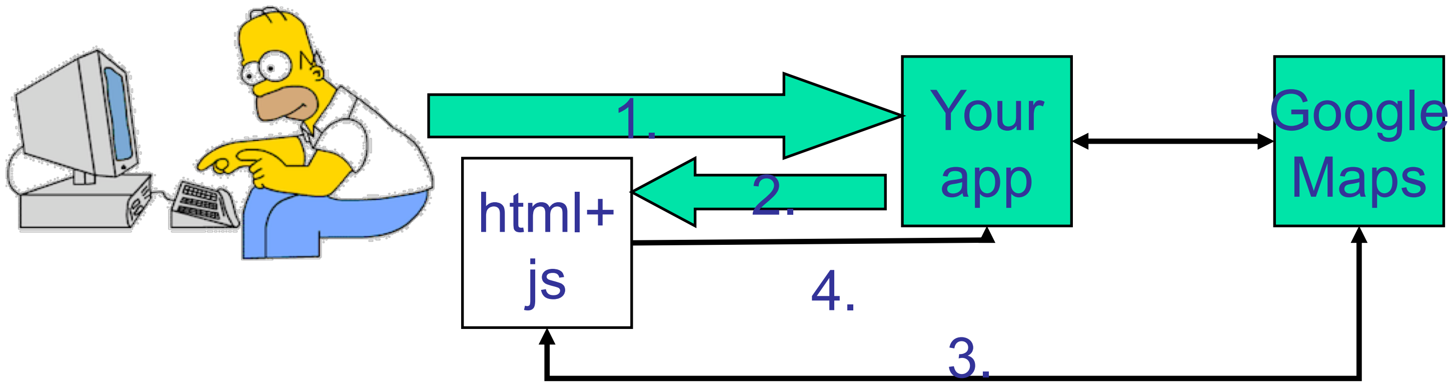
- Facebook plug-in apps



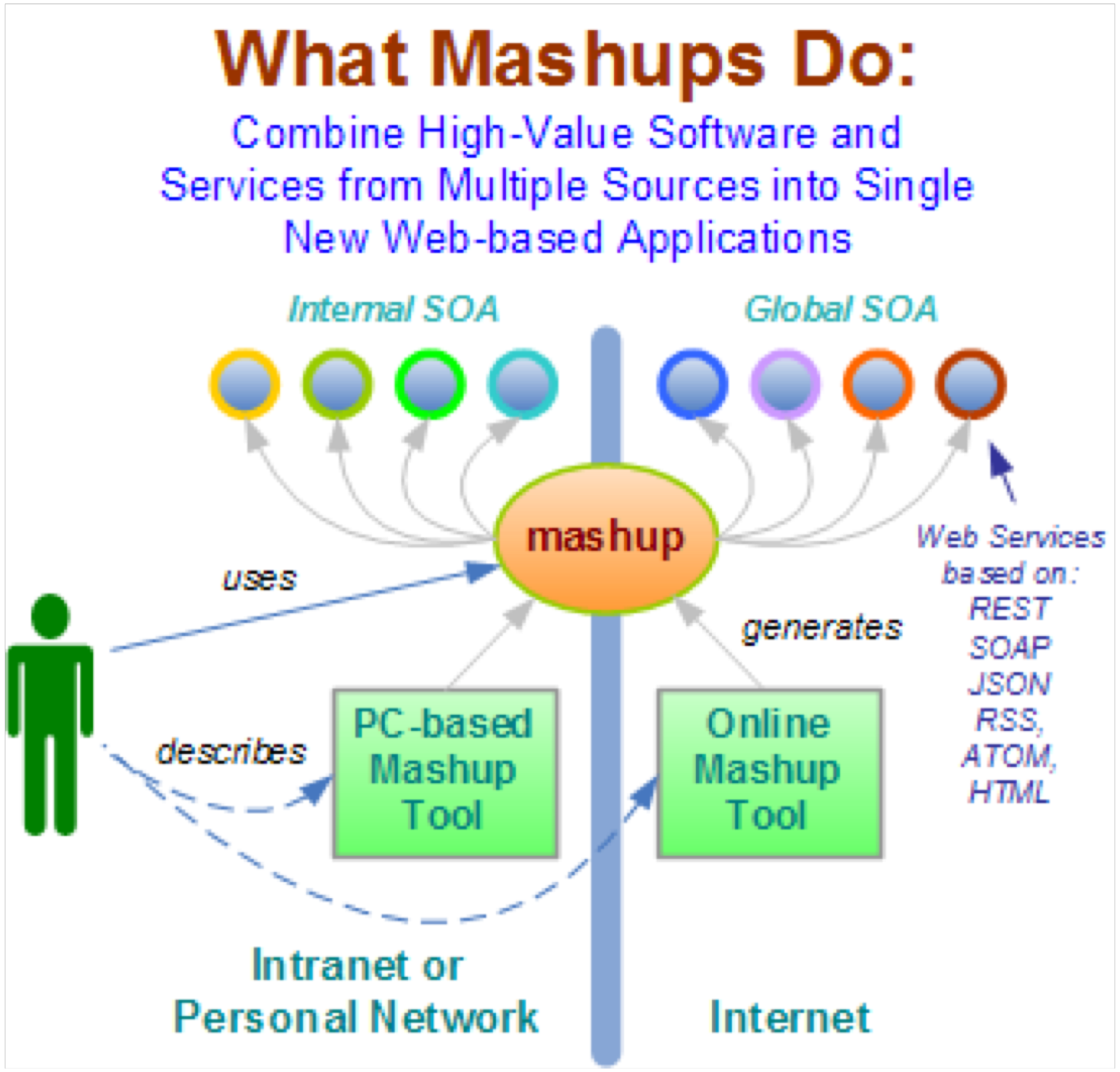


Google Maps

- App embeds Javascript client code (provided by Google)
  - client-side functionality: clear/draw overlays, etc.
  - server-side functionality: fetch new map, rescale, geocoding
- Attach callbacks (handled by your app) to UI actions
- Result of callback can trigger additional calls to Google Maps code, which in turn contact GMaps servers

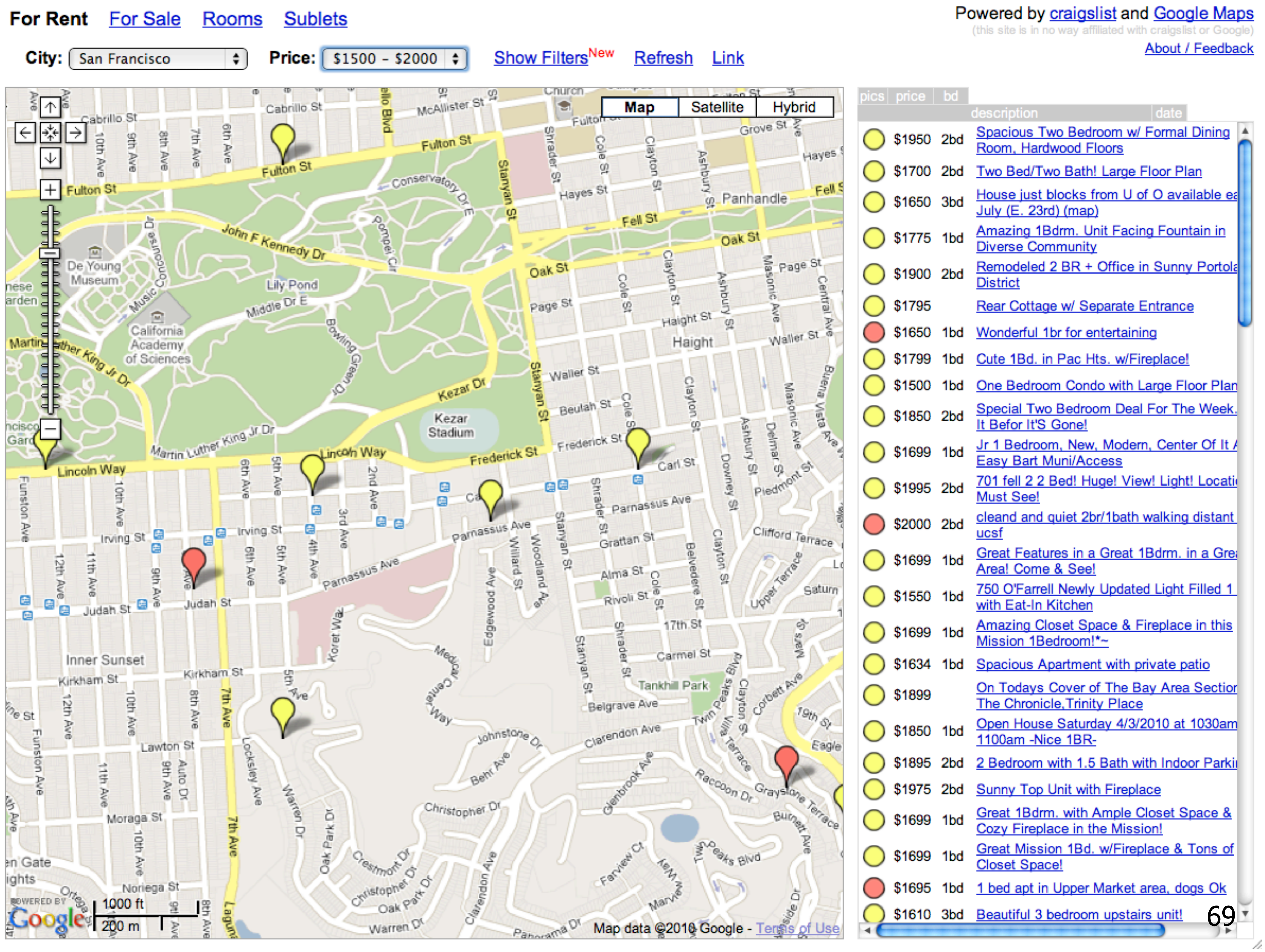


Mashup


What Mashups Do:
Combine High-Value Software and Services from Multiple Sources into Single New Web-based Applications
Internal SOA
Global SOA
mashup
uses
describes
generates
PC-based Mashup Tool
Online Mashup Tool
Web Services based on: REST SOAP JSON RSS, ATOM, HTML
Intranet or Personal Network
Internet

http://www.programmableweb.com

Mashups: housingmaps.com



For Rent For Sale Rooms Sublets

Powered by craigslist and Google Maps
(this site is in no way affiliated with craigslist or Google)
About / Feedback

City: San Francisco Price: $1500 - $2000 Show Filters New Refresh Link

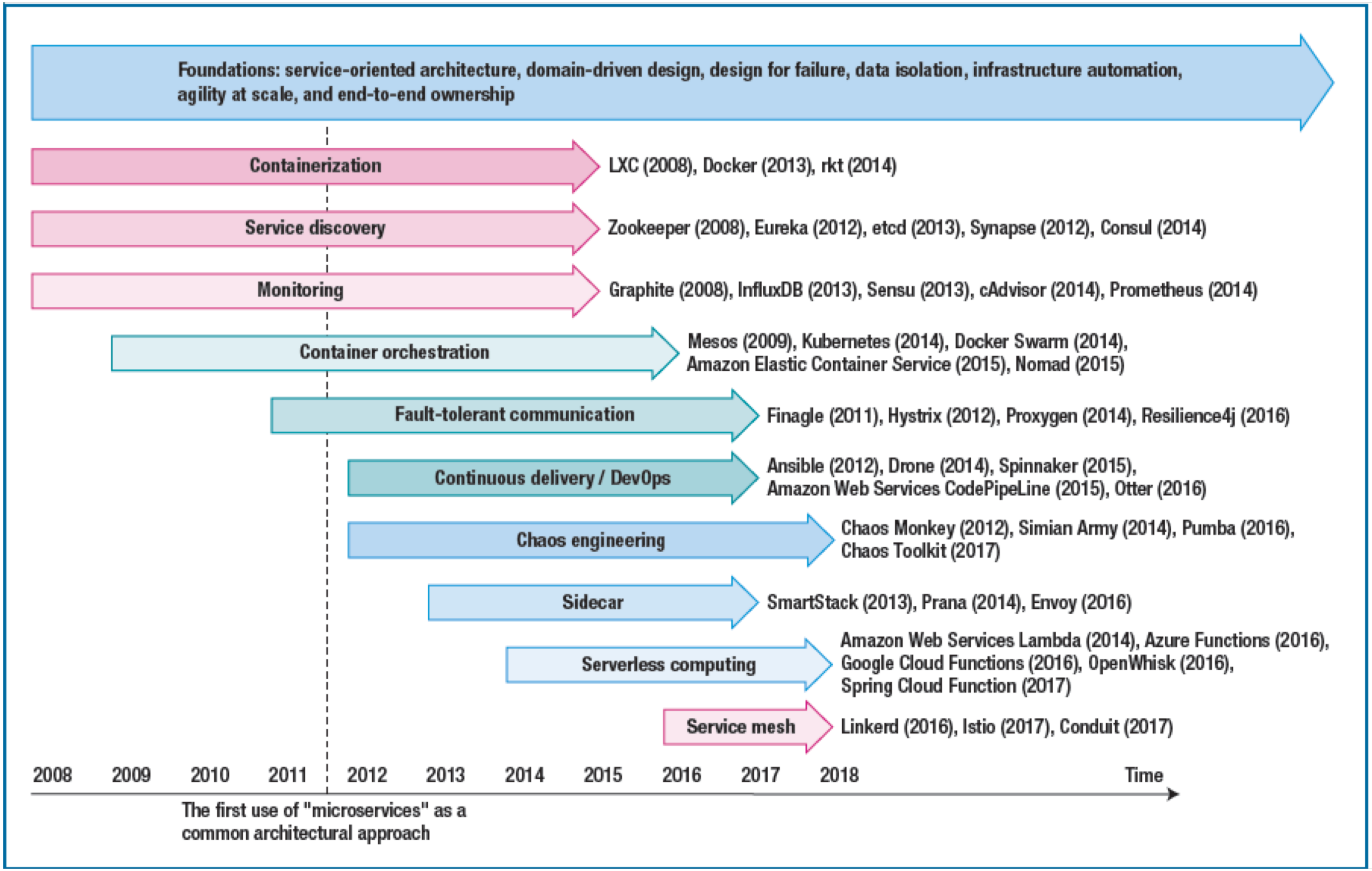
pics	price	bd	description
🟡	$1950	2bd	Spacious Two Bedroom w/ Formal Dining Room, Hardwood Floors
🟡	$1700	2bd	Two Bed/Two Bath! Large Floor Plan
🟡	$1650	3bd	House just blocks from U of O available ea July (E. 23rd) (map)
🟡	$1775	1bd	Amazing 1Bdrm. Unit Facing Fountain in Diverse Community
🟡	$1900	2bd	Remodeled 2 BR + Office in Sunny Portola District
🟡	$1795		Rear Cottage w/ Separate Entrance
🔴	$1650	1bd	Wonderful 1br for entertaining
🟡	$1799	1bd	Cute 1Bd. in Pac Hts. w/Fireplace!
🟡	$1500	1bd	One Bedroom Condo with Large Floor Plan
🟡	$1850	2bd	Special Two Bedroom Deal For The Week. It Befor It'S Gone!
🟡	$1699	1bd	Jr 1 Bedroom, New, Modern, Center Of It A Easy Bart Muni/Access
🟡	$1995	2bd	701 fell 2 2 Bed! Huge! View! Light! Locati Must See!
🔴	$2000	2bd	cleand and quiet 2br/1bath walking distant ucsf
🟡	$1699	1bd	Great Features in a Great 1Bdrm. in a Gre Area! Come & See!
🟡	$1550	1bd	750 O'Farrell Newly Updated Light Filled 1 with Eat-In Kitchen
🟡	$1699	1bd	Amazing Closet Space & Fireplace in this Mission 1Bedroom!*~
🟡	$1634	1bd	Spacious Apartment with private patio
🟡	$1899		On Todays Cover of The Bay Area Sectior The Chronicle,Trinity Place
🟡	$1850	1bd	Open House Saturday 4/3/2010 at 1030am 1100am -Nice 1BR-
🟡	$1895	2bd	2 Bedroom with 1.5 Bath with Indoor Parki
🟡	$1975	2bd	Sunny Top Unit with Fireplace
🟡	$1699	1bd	Great 1Bdrm. with Ample Closet Space & Cozy Fireplace in the Mission!
🟡	$1699	1bd	Great Mission 1Bd. w/Fireplace & Tons of Closet Space!
🔴	$1695	1bd	1 bed apt in Upper Market area, dogs Ok
🟡	$1610	3bd	Beautiful 3 bedroom upstairs unit!

Microservices

- *"Microservice Architecture" describes a particular way of designing software applications as suites of independently deployable services.*
- *While there is no precise definition of this architectural style, there are certain common characteristics around organization, business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data*

http://martinfowler.com/articles/microservices.html

Microservice technologies timeline


Foundations: service-oriented architecture, domain-driven design, design for failure, data isolation, infrastructure automation, agility at scale, and end-to-end ownership
Containerization
LXC (2008), Docker (2013), rkt (2014)
Service discovery
Zookeeper (2008), Eureka (2012), etcd (2013), Synapse (2012), Consul (2014)
Monitoring
Graphite (2008), InfluxDB (2013), Sensu (2013), cAdvisor (2014), Prometheus (2014)
Container orchestration
Mesos (2009), Kubernetes (2014), Docker Swarm (2014), Amazon Elastic Container Service (2015), Nomad (2015)
Fault-tolerant communication
Finagle (2011), Hystrix (2012), Proxygen (2014), Resilience4j (2016)
Continuous delivery / DevOps
Ansible (2012), Drone (2014), Spinnaker (2015), Amazon Web Services CodePipeLine (2015), Otter (2016)
Chaos engineering
Chaos Monkey (2012), Simian Army (2014), Pumba (2016), Chaos Toolkit (2017)
Sidecar
SmartStack (2013), Prana (2014), Envoy (2016)
Serverless computing
Amazon Web Services Lambda (2014), Azure Functions (2016), Google Cloud Functions (2016), OpenWhisk (2016), Spring Cloud Function (2017)
Service mesh
Linkerd (2016), Istio (2017), Conduit (2017)
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
Time
The first use of "microservices" as a common architectural approach

Summary: SOA principles

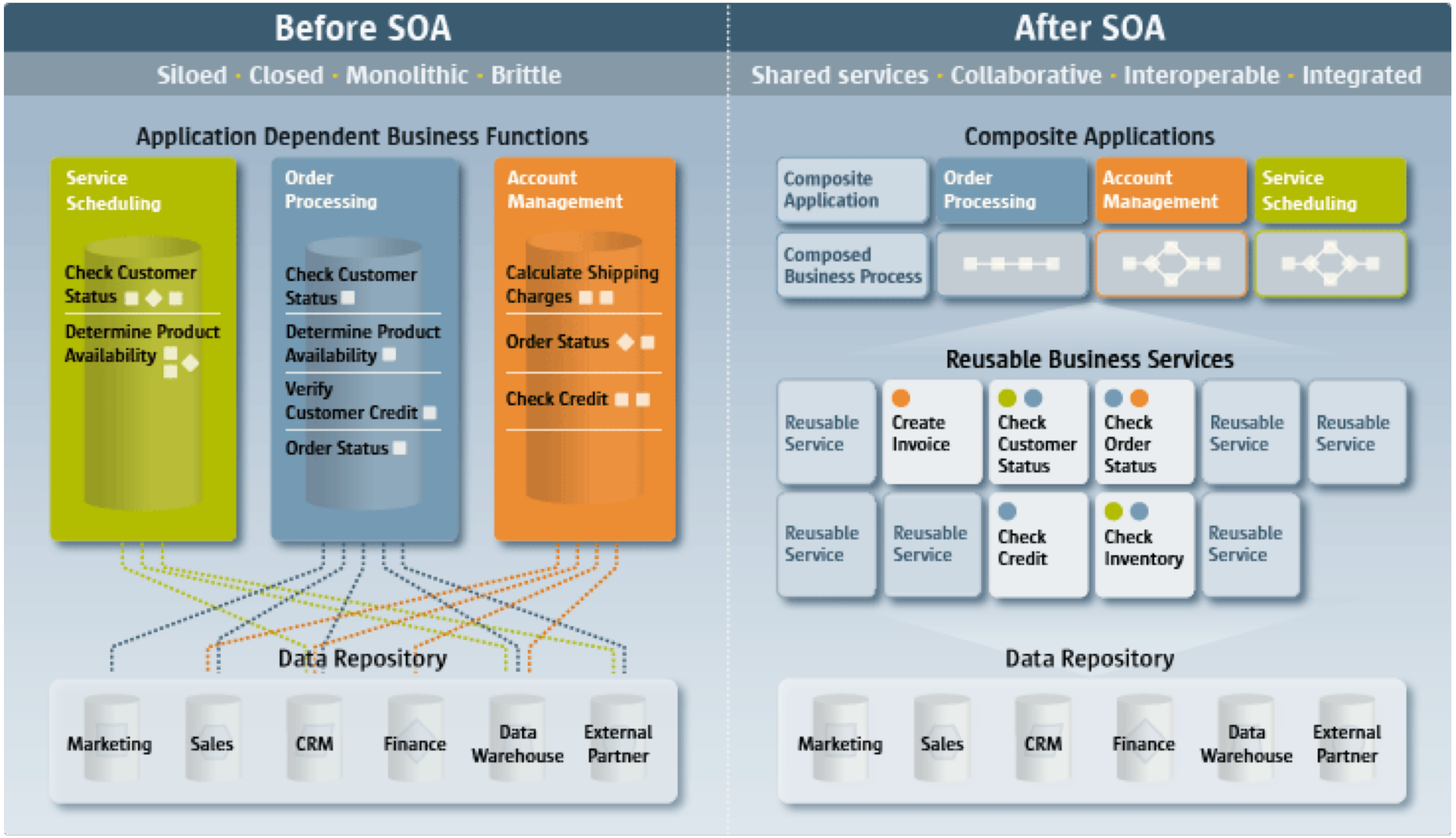
- **Services are autonomous**
- **Services are distributable**
- **Services are loosely coupled**
- **Services share schema and contract, not class**
- **Compatibility is based on policy**

Common examples of service-oriented applications include sharing information, handling multistep processes such as reservation systems and online stores, exposing specific data or services over an extranet, and creating mashups that combine information from multiple sources

SOA benefits: summary

- **Domain alignment**. Reuse of services with standard interfaces increases business and technology opportunities and reduces costs
- **Abstraction**. Services are autonomous and accessed through a formal contract, which provides loose coupling and abstraction
- **Discoverability**. Services expose descriptions that allow other services to locate them and automatically determine the interface
- **Interoperability**. Because the protocols and data formats are based on industry standards, the provider and consumer of the service can be built and deployed on different platforms
- **Rationalization**. Services can be granular in order to provide specific functionality, rather than duplicating the functionality in number of applications, which removes duplication

Before and after SOA



https://herbertograca.com/2017/11/09/service-oriented-architecture-soa/

Self test

- Which are the consequences of defining software “a *service*” (instead of “a *good*”)?
- What are the main features of a SOA architectural style?
- What are the main features of the REST architectural style?
- Discuss the difference between REST and SOAP-based architectures.
- How can we see that a site is RESTful?
- Which architectural issues and patterns are typical of systems based on SOA technologies?

References

- Erl, *SOA design patterns*, Prentice Hall, 2008
- RotemGalOz, *SOA patterns*, Manning, 2012
- Richards, *Microservices vs SOA*, O’Reilly, 2015
- Wolff, *Microservices*, AW, 2016
- Daniel & Matera, *Mashups*, Springer, 2014

Relevant sites

- www.soapatterns.org
- www.omg.org/technology/readingroom/SOA.htm
- martinfowler.com/articles/microservices.html
- www.ibm.com/developerworks/architecture/library/ar-logsoa
- aws.typepad.com/
- kasunpanorama.blogspot.it/2015/11/microservices-in-practice.html
- www.packtpub.com/article/modeling-orchestration-and-choreography-in-service-oriented-architecture
- www.ibm.com/developerworks/rational/library/09/modelingwithsoaml-1/index.htm

Journals and conferences

- IEEE Transactions on Network and Service management
- Future Generation Computing Systems
- IEEE Int. Conf. on Web Services

Questions?