Services

Distribution paradigm between hype and revolution

Walter Kriha

jasongorman
@jasongorman

Follow

1998: "You should build systems out of single-purpose loosely-coupled components

2008: "You should build systems out of single-purpose loosely-coupled components"

2018: "You should build systems out of single-purpose loosely-coupled components"

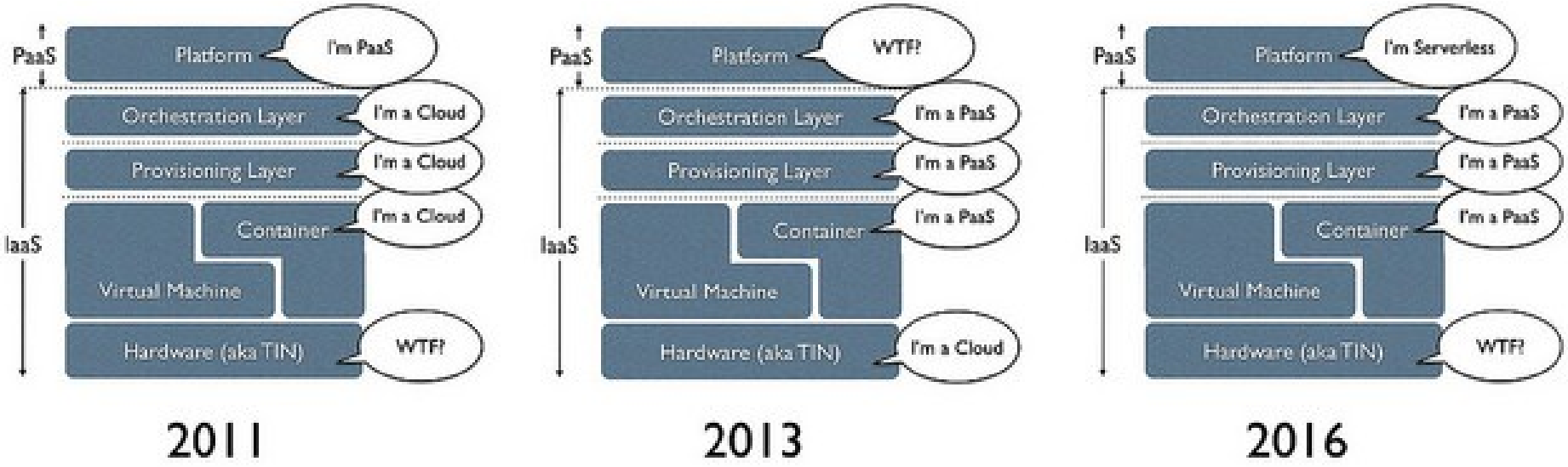
Technology changes so fast!

1:57 AM - 19 Nov 2018

1,726 Retweets **4,390** Likes

66 1.7K 4.4K

Sometimes, the best thing you can do in technology is change your name



https://twitter.com/swardley/status/952180307540824065

No kidding: think about model-driven architecture becoming no-code/low-code...

Overview

- Standing on the shoulders of giants
  - CORBA
  - WebServices
  - SOA
- REST
- MicroServices and Conway's Law
- NanoServices (aka Serverless Computing)

Goals

Understand the “service idea” in distributed systems and its history

See how organization and technology need to be aligned

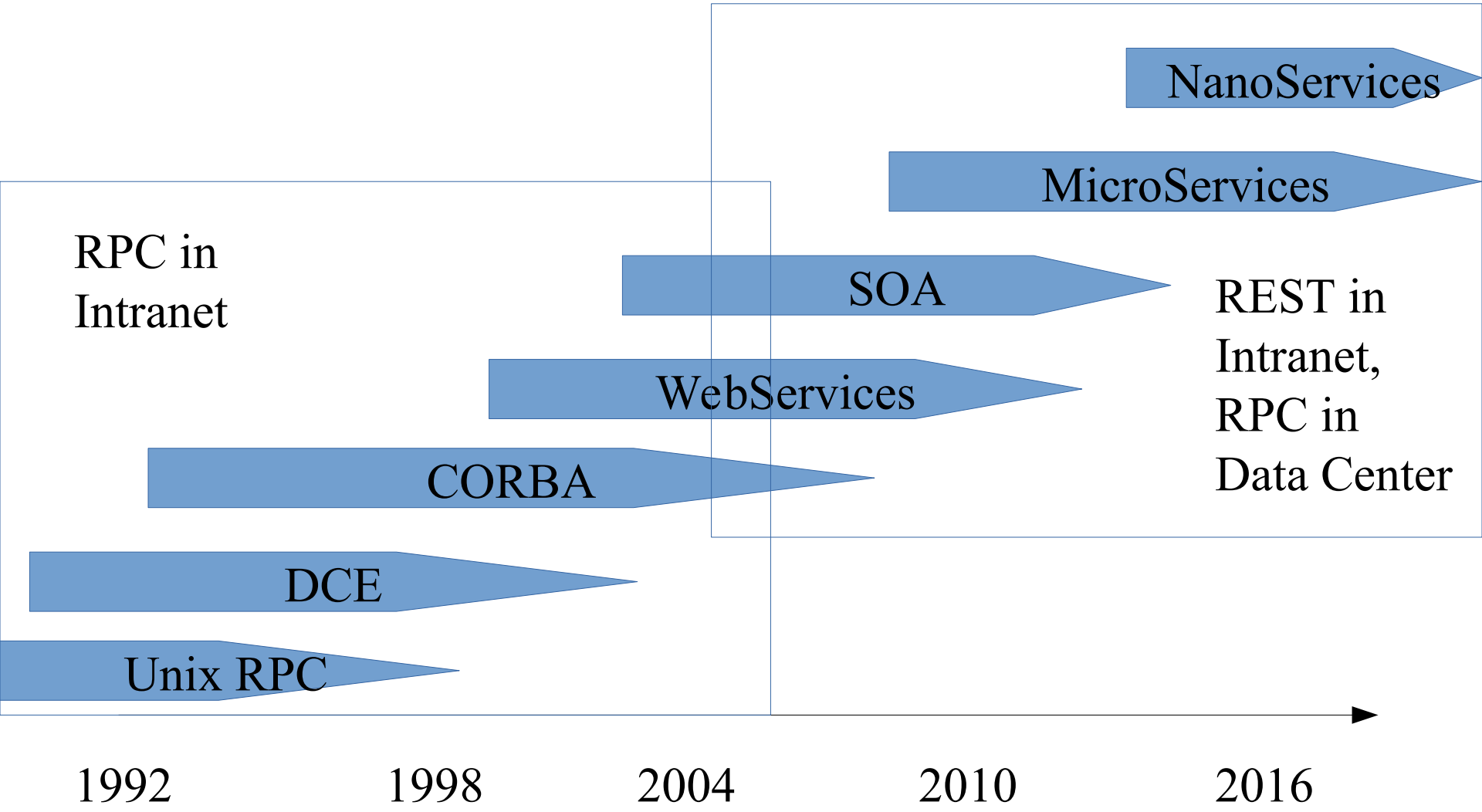
See how different approaches deal with cross-cutting concerns like transactions, security and delivery guarantees

Tr to understand how “services” and the “layer” architectural design pattern mix.

Try to understand the conflicting goals behind SOA: loose coupling, re-use, short round-trip times, general services vs. Special needs of applications, transactions, different granularity, workflow composition from services and the problems of context and concerns.

Right now it looks like the service idea has “won”. Heavyweight containers like JEE/.NET demonstrated scalability problems.

Timeline of Distributed Service Architectures



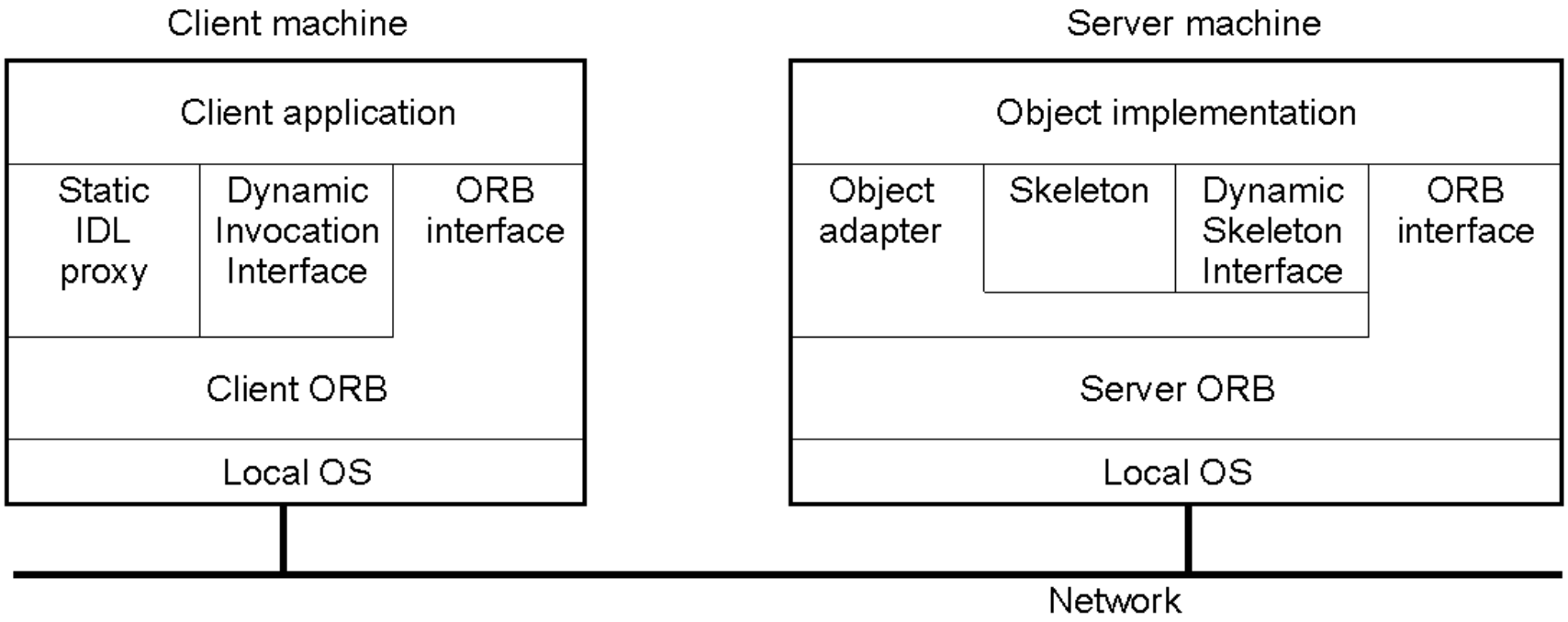
Common Request Broker Architecture (CORBA)

CORBA Services

SPECIFICATION	acronym	topical area / domain	Document #
Additional Structuring Mechanisms for the OTS	OTS	transaction management, middleware	formal/2005-01-01
Collection Service	COLL	collection management, middleware	formal/2002-08-03
Concurrency Service	CONC	object consistency, middleware	formal/2000-06-14
Enhanced View of Time	EVoT	time management, middleware	formal/2008-08-01
Event Service	EVNT	event management, middleware	formal/2004-10-02
Externalization Service	EXT	object state management, middleware	formal/2000-06-16
Licensing Service	LIC	software licensing, middleware	formal/2000-06-17
Life Cycle Service	LFCYC	object life cycle management, middleware	formal/2002-09-01
Lightweight Services	LtSVC	resources constraints	formal/2004-10-01
Management of Event Domains	MED	event management, middleware	formal/2001-06-03
Naming Service	NAM	object location management, middleware	formal/2004-10-03
Notification Service	NOT	event management, middleware	formal/2004-10-11
Notification / JMS Interworking	NOTJMS	event management, middleware	formal/2004-10-09
Persistent State Service	PSS	object persistence, middleware	formal/2002-09-06
Property Service	PROP	object properties, middleware	formal/2000-06-22
Query Service	QUER	collection management, middleware	formal/2000-06-23
Relationship Service	REL	object relationships, middleware	formal/2000-06-24
Security Service	SEC	security, middleware	formal/2002-03-11
Telecoms Log Service	TLOG	event management, middleware	formal/2003-07-01
Time Service	TIME	time management, middleware	formal/2002-05-06
Trading Object Service	TRADE	object location management, middleware	formal/2000-06-27
Transaction Service	TRANS	transaction management, middleware	formal/2003-09-02

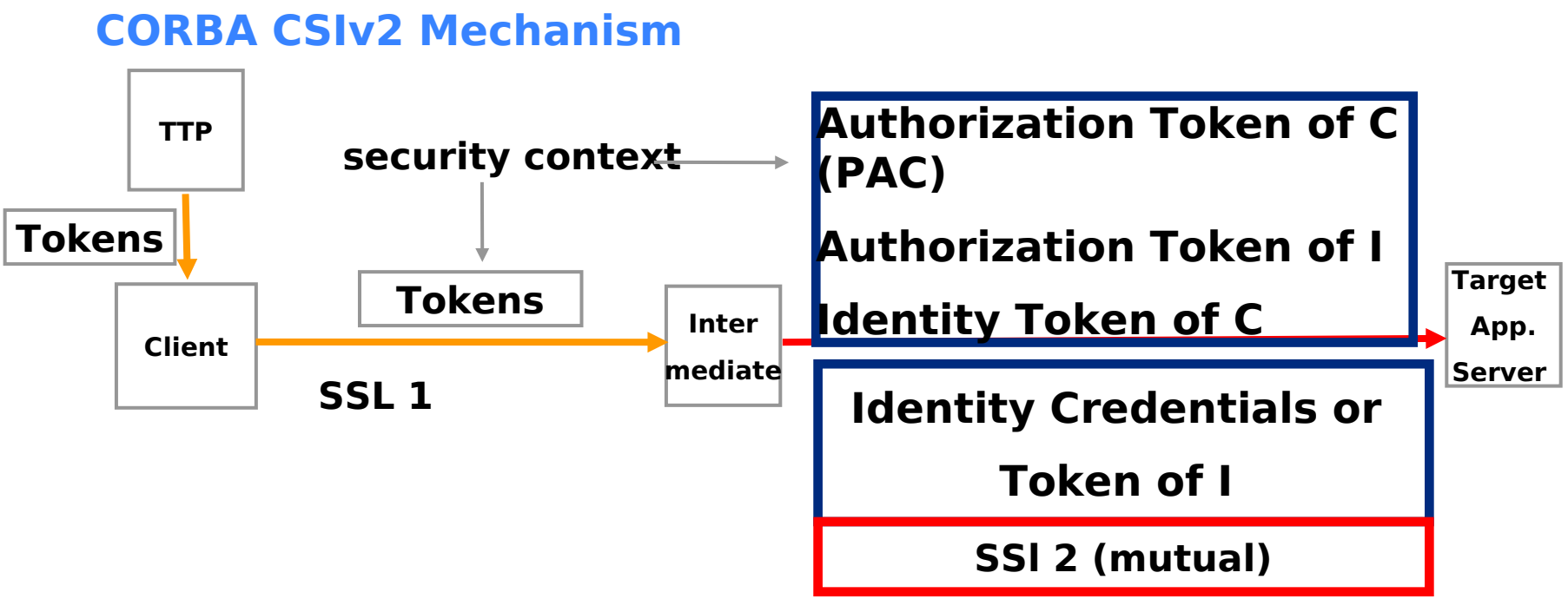
http://www.omg.org/spec/index.htm

Object Request Broker Architecture



from van Steen, Tanenbaum, Distributed Systems. For protocols etc.: Common Object Request Broker Architecture: Core Specification, http://www.omg.org/cgi-bin/doc?formal/04-03-12.pdf

Security: Secure Delegation Concept



Every system involved authenticates itself against other tiers and flows client tokens. No secrets are shared. Defined routes prevent token abuse. Later tiers can verify original requestor and route.

Transaction Service

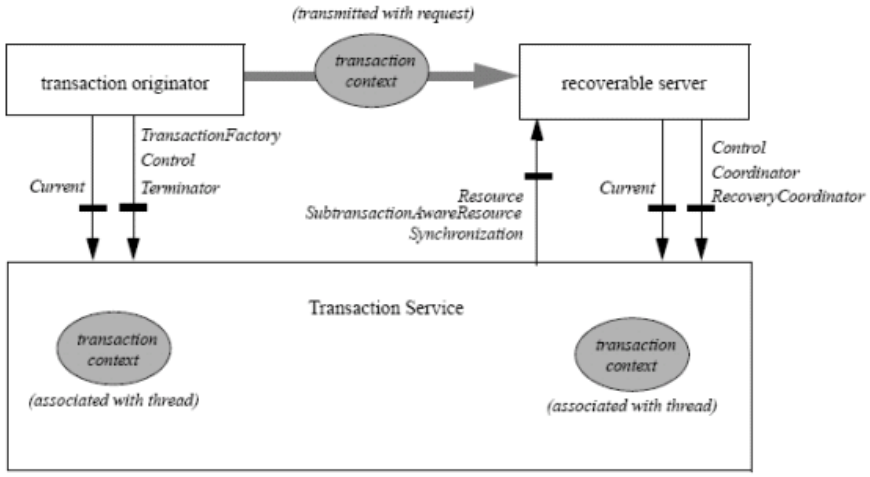


Figure 1-2 Major Components and Interfaces of the Transaction Service

Object Transaction Service from CORBA

```
interface TransactionFactory {
  Control create(in unsigned long time_out);
  ...
};
interface Control {
  Terminator  get_terminator();
  Coordinator get_coordinator();
};
interface Terminator {
  void commit(...);
  void rollback();
};
interface Coordinator {
  RecoveryCoordinator register_resource(in Resource r);
  ...
};
interface RecoveryCoordinator {
  Status replay_completion(in Resource r);
};
interface Resource {
  Vote prepare();
  void rollback();
  void commit();
  ...
};
local interface Current : CORBA::Current {
  void begin();
  void commit();
  void rollback();
  void set_timeout(in unsigned long seconds);
  unsigned long get_timeout();
  Control get_control();
  Control suspend();
  void resume(in Control which);
};
```

CORBA Example

Hello.idl

```
module HelloApp
{
  interface Hello
  {
  string sayHello();
  oneway void shutdown();
  };
};
```

```
// HelloServer.java
import org.omg.*;

class HelloImpl extends HelloPOA {
  private ORB orb;
  public void setORB(ORB orb_val)
{
    orb = orb_val;
  }

  // implement sayHello() method
  public String sayHello() {
    return "\nHello world !!\n";
  }

  // implement shutdown() method
  public void shutdown() {
    orb.shutdown(false);
  }}
```

```
public class HelloClient {
  static Hello helloImpl;
  public static void main(String args[])
    {   // create and initialize the ORB
        ORB orb = ORB.init(args, null);
        // get the root naming context
        org.omg.CORBA.Object objRef =
            orb.resolve_initial_references("NameService");
        // Use NamingContextExt instead of NamingContext. This is
        // part of the Interoperable naming Service.
        NamingContextExt ncRef =
NamingContextExtHelper.narrow(objRef);

        // resolve the Object Reference in Naming
        String name = "Hello";
        helloImpl = HelloHelper.narrow(ncRef.resolve_str(name));
        System.out.println("Obtained a handle on server object: "
+ helloImpl);
        System.out.println(helloImpl.sayHello());
        helloImpl.shutdown();

    }}
```

```
public class HelloServer {
  public static void main(String args[]) {
      // create and initialize the ORB
      ORB orb = ORB.init(args, null);
      // get reference to rootpoa & activate the
POAManager
      POA rootpoa =
POAHelper.narrow(orb.resolve_initial_references("RootPOA")
);
      rootpoa.the_POAManager().activate();
      // create servant and register it with the ORB
      HelloImpl helloImpl = new HelloImpl();
      helloImpl.setORB(orb);
      // get object reference from the servant
      org.omg.CORBA.Object ref =
rootpoa.servant_to_reference(helloImpl);
      Hello href = HelloHelper.narrow(ref);

      // get the root naming context
      // NameService invokes the name service
      org.omg.CORBA.Object objRef =
          orb.resolve_initial_references("NameService");
      // Use NamingContextExt which is part of the
Interoperable
      // Naming Service (INS) specification.
      NamingContextExt ncRef =
NamingContextExtHelper.narrow(objRef);
      // bind the Object Reference in Naming
      String name = "Hello";
      NameComponent path[] = ncRef.to_name( name );
      ncRef.rebind(path, href);
      System.out.println("HelloServer ready and
waiting ...");
      // wait for invocations from clients
      orb.run();
    } }}
```

(Modified) from:
http://docs.oracle.com/javase/7/docs/technotes/guides/idl/jidlExample.html

CORBA Core Properties

- Clearly an INTRANET technology
- Language independent with a focus on interface definitions
- Base protocol defined for interoperability and cross-cutting concerns (IIOP)
- Delivery guarantees provided by base protocol
- Mostly used to connect heterogeneous (legacy) software in large corporations
- Difficult and tedious standardization process
- Lots of "boilerplate code" leading to extensive code generation and model-driven development
- Java 9 will no longer include CORBA in the default classpath

Michi Henning, The Rise and Fall of CORBA, ACM
http://cacm.acm.org/magazines/2008/8/5336-the-rise-and-fall-of-corba/fulltext

WebServices

The following is only a small part of a much larger course on webservices, ws-security, canonical XML, encrypted XML etc.

What are Web Services?

„A Web service is a software component that represents a business function (or a business service) and can be accessed by another application (a client, a server or another Web service) over public networks using generally available ubiquitous protocols and transports (i.e. SOAP over http)“. (http://www3.gartner.com/Init by M.Pezzini, April 2001)

WWW: from GUI driven to B2B

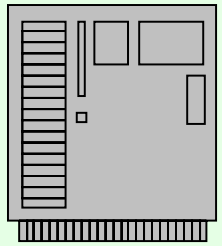
```
<FORM action="http://stockservice.com/getquote" method="post">
    <P><LABEL for=valor">valor: </LABEL>
        <INPUT type="text" id=„valor"><BR>
        <INPUT type="submit" value="Send"> </FORM>
```

stockservice: valor=IBM

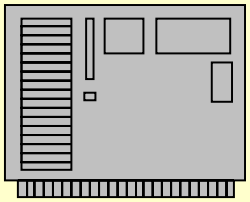
→

html document with IBM=44.56

←



stock server



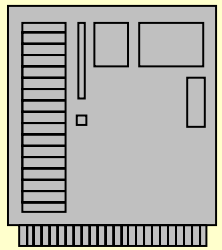
myYahoo

```
<xml-rpc><service>stockservice</
service><request>getquote><parameter><name>valor</
name><value>IBM</value></parameter></request></xml-rpc>
```

→

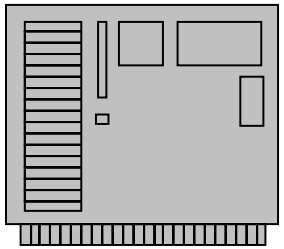
```
<xml-rpc><service>stockservice</
service><response>getquote><parameter><name>IBM</
name><value>44.56</value></parameter></response></xml-rpc>
```

←

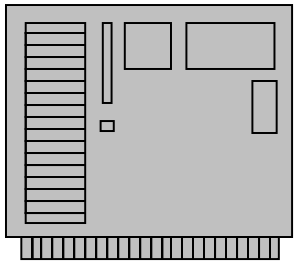


The concept of a web service is extremely simple: use XML to create requests and responses and send them using http. This allows machines to communicate with each others, e.g. to perform supply chain management or other business to business processing. XML-RPC by David Winer (userland.com) was one of the earliest standard proposals. Companies have used this technology internally for quite a while.

Web Services Components



single sign on services: Hailstorm/liberty alliance



global registry (UDDI)

Digital Signatures	Transactions	Metering

Universal Description, Discovery, Integration UDDI (XML)

Web Service Description Language WSDL (XML)

Request format: SOAP (XML)

Transport layer: http(s), smtp, httpr

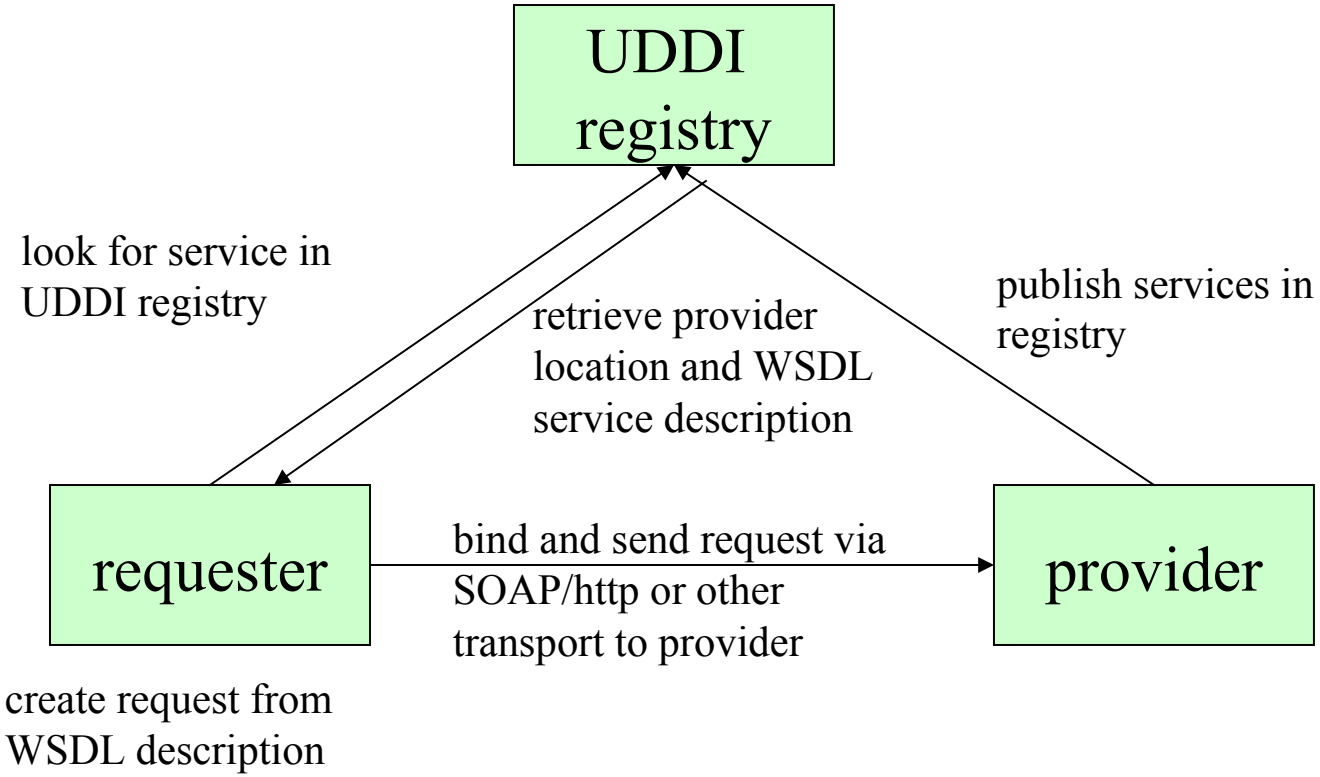
XML is the standard format used in Web Services. On top of standard transport mechanisms are requests formatted using the SOAP XML schema. Clients learn about service providers by browsing the UDDI registry. Services are described in a special description language, again a XML schema.

Web Services Core Properties

- „simple“ requests
- Over public networks/Internet
- Using http transport for firewall reasons (Delivery guarantees?)
- XML message format (language independent)
- Added features for reliability, security and transactions
- In many cases a re-write of CORBA interfaces with XML syntax
- Expressing a business function

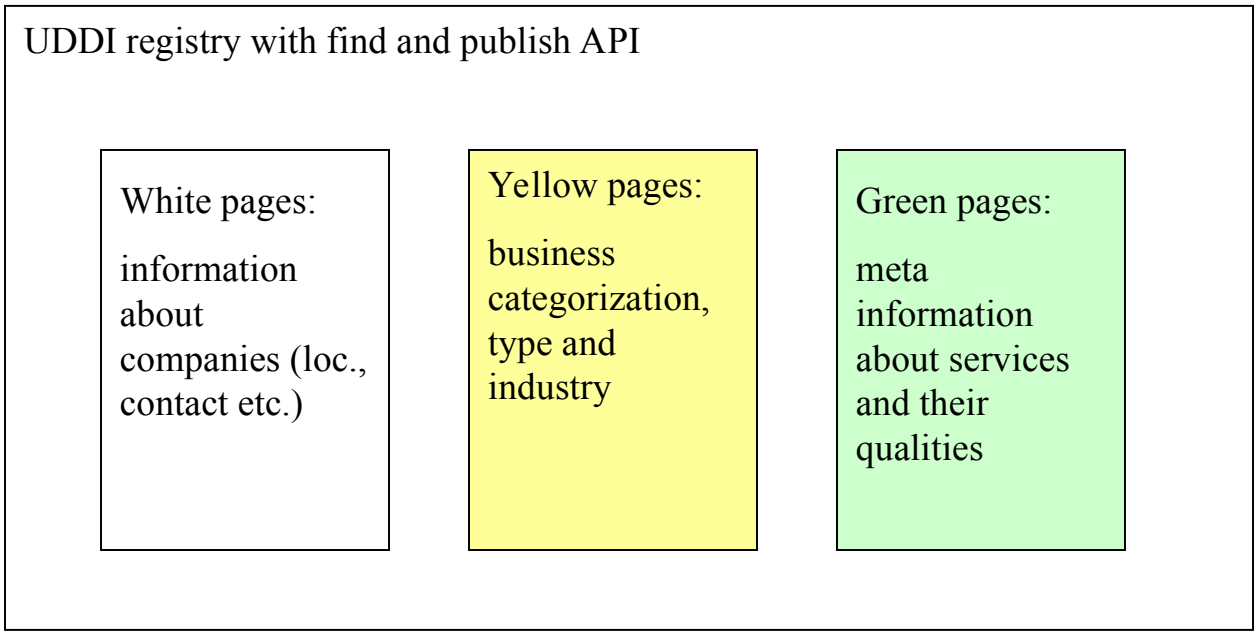
Massively overhyped WebServices postulated automatic interoperability based on self-describing services and ontologies. The technical base was provided by forms of XML-RPC. SOAP had nothing to do with distributed objects in spite of the name!

„Service Oriented“ Architecture



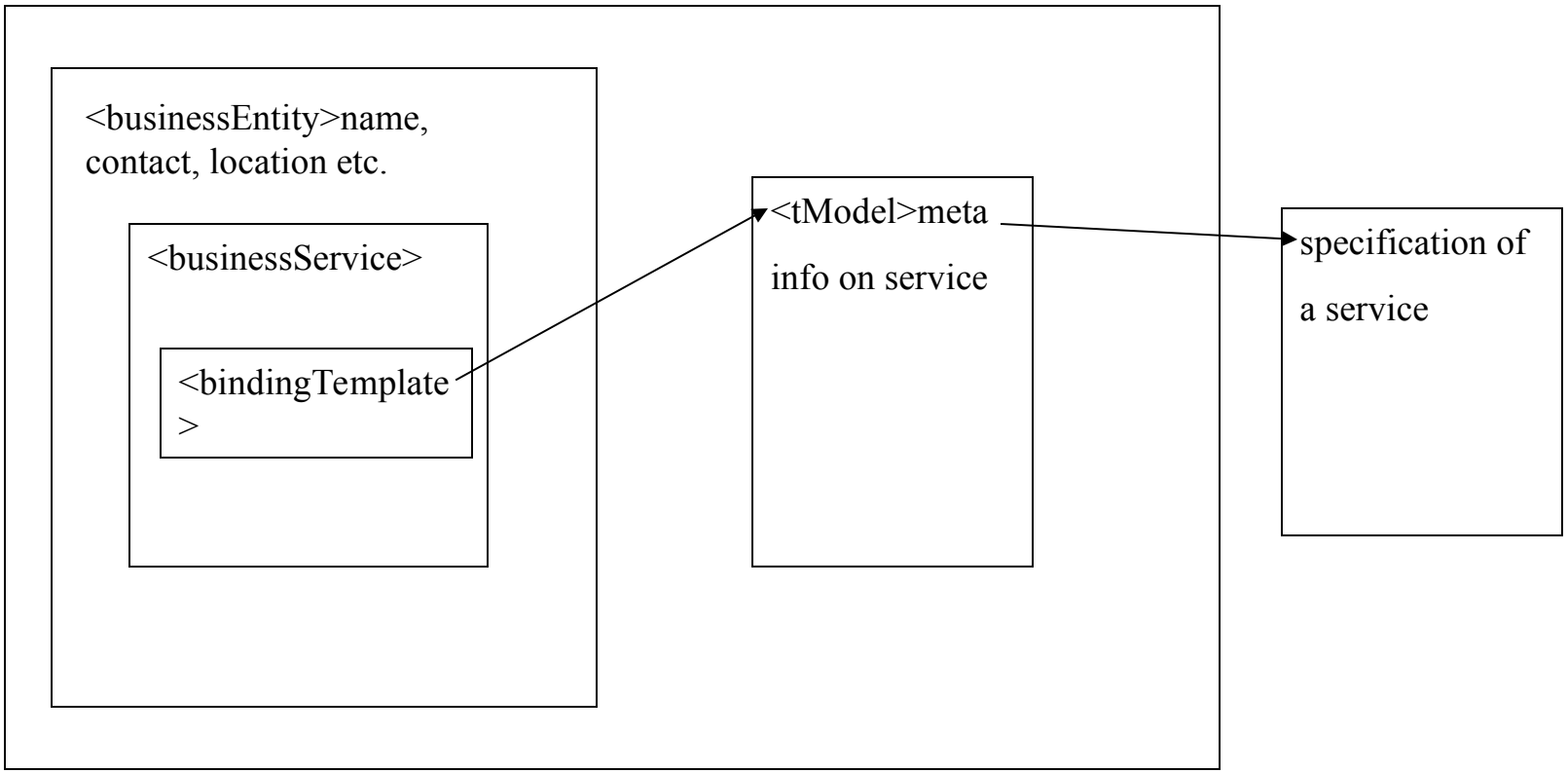
This type of architecture is called „service-oriented“. It uses a broker for service advertisement and lookup. Requester and provider bind dynamically with respect to transport (http, smtp etc.) (Raghavan N. Srinivas, Web services hits the Java scene part 1, http://www.javaworld.com)

Service Discovery: UDDI



most distributed services use some kind of central registry for service lookup. The Universal Description, Discovery and Integration registry plays this role in web services. Especially the green pages property has led some people to proclaim automatic service matching by service requesters browsing the meta-information contained there. For the difficulties behind ontologies and automated discovery see: Steve Vinoski, Web Services and Dynamic Discovery on webservices.org

Service Discovery (2): UDDI content



All content in UDDI is expressed in XML. Besides the obvious elements for companies and services a number of meta-information elements like tModel exist. A core feature of UDDI is the expectation that requester and provider do a dynamic bind where they agree on service and transport characteristics. A local registry can be downloaded from www.alphaworks.ibm.com

WSDL: The IDL of Web Services

```
<?xml version="1.0"?>   <definitions name="StockQuote"
        <schena targetNamespace=http://example.com/stockquote.wsdl [...]
  <types><schema targetNamespace="http://example.com/stockquote.xsd" [...]
            <element name="TradePriceRequest">
              <complexType>  <all>   <element name="tickerSymbol" type="string"/>  </all> </complexType>
            </element></schema> </types>
   <message name="GetLastTradePriceInput">
            <part name="body" element="xsd1:TradePriceRequest"/></message>
  <portType name="StockQuotePortType">
            <operation name="GetLastTradePrice">
              <input message="tns:GetLastTradePriceInput"/>
              <output message="tns:GetLastTradePriceOutput"/>  </operation> </portType>
 <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
            [..]   <operation name="GetLastTradePrice"> [..] </binding>
 <service name="StockQuoteService">
            <documentation>My first service</documentation>
            <port name="StockQuotePort" binding="tns:StockQuoteBinding">
              <soap:address location="http://example.com/stockquote"/> </port> </service>  </definitions>
```

Web Services Description Language (WSDL) is the metadata language of Web Services. It defines how service providers and requesters understand Web Services. When exposing back-ends as Web Services, WSDL defines and exposes components and lists all the data types, operations, and parameters used by that service. WSDL provides all the information that a client application needs to build a valid SOAP invocation that in turn is mapped by the Web Services platform onto back-end enterprise logic. (after P.J.Murray, Web Services and CORBA, CapeClear)

WSDL Elements

- Types– a container for data type definitions using some type system (such as XSD).
- Message– an abstract, typed definition of the data being communicated.
- Operation– an abstract description of an action supported by the service.
- Port Type–an abstract set of operations supported by one or more endpoints.
- Binding– a concrete protocol and data format specification for a particular port type.
- Port– a single endpoint defined as a combination of a binding and a network address.
- Service– a collection of related endpoints.

A WSDL document defines services as collections of network endpoints, or ports. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: messages, which are abstract descriptions of the data being exchanged, and port types which are abstract collections of operations. The concrete protocol and data format specifications for a particular port type constitutes a reusable binding.

Request Format of Web Services: SOAP

hello-request

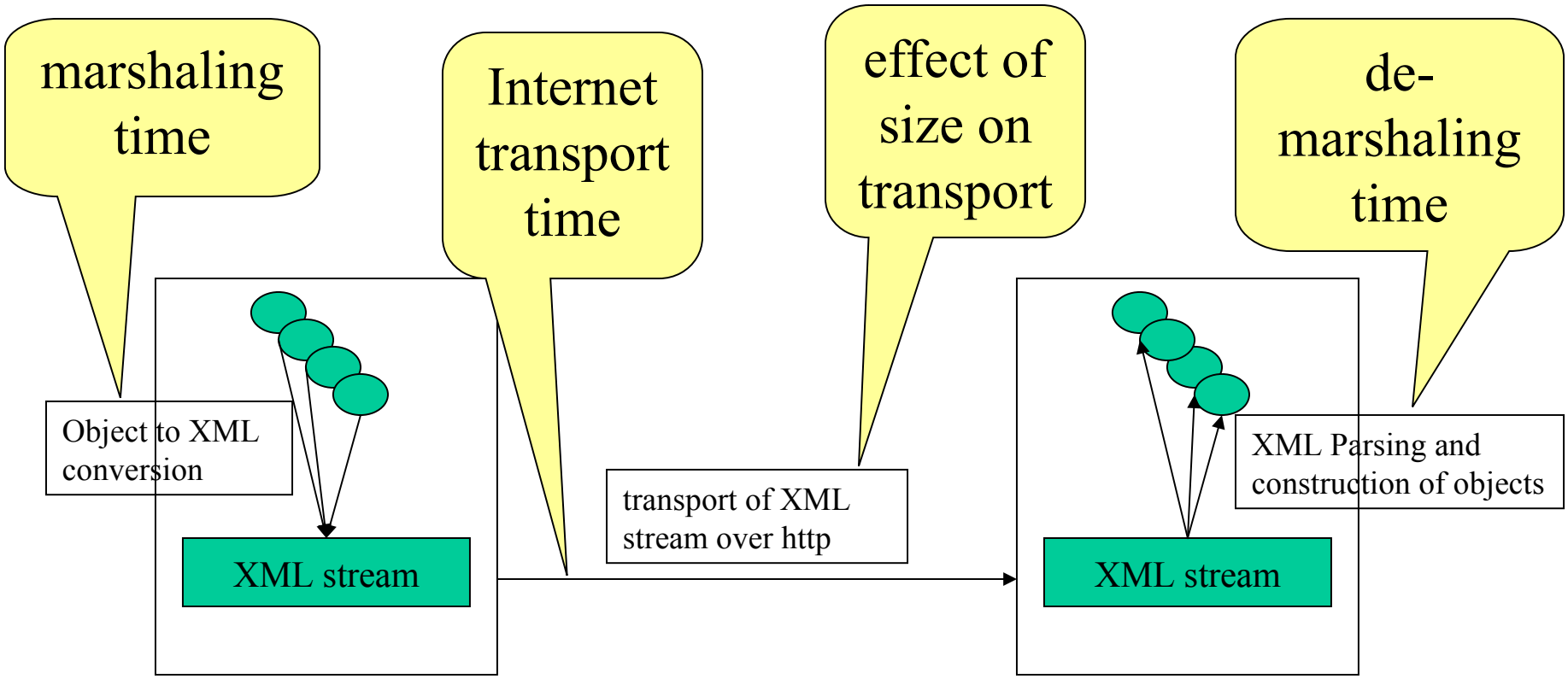
```
<s:Envelope
xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
    <s:Body>
      <m:sayHello xmlns:m='urn:Example1'>
        <name xsi:type='xsd:string'>James</name>
      </m:sayHello>
    </s:Body>
</s:Envelope>
```

hello-response

```
<s:Envelope
xmlns:s="http://www.w3.org/2001/06/soap-envelope"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-
instance"xmlns:xsd="http://www.w3.org/1999/XMLSchema">
    <s:Body>
      <n:sayHelloResponse xmlns:n="urn:Example1">
        <return xsi:type="xsd:string">
          Hello James
        </return>
      </n:sayHelloResponse>
    </s:Body>
</s:Envelope>
```

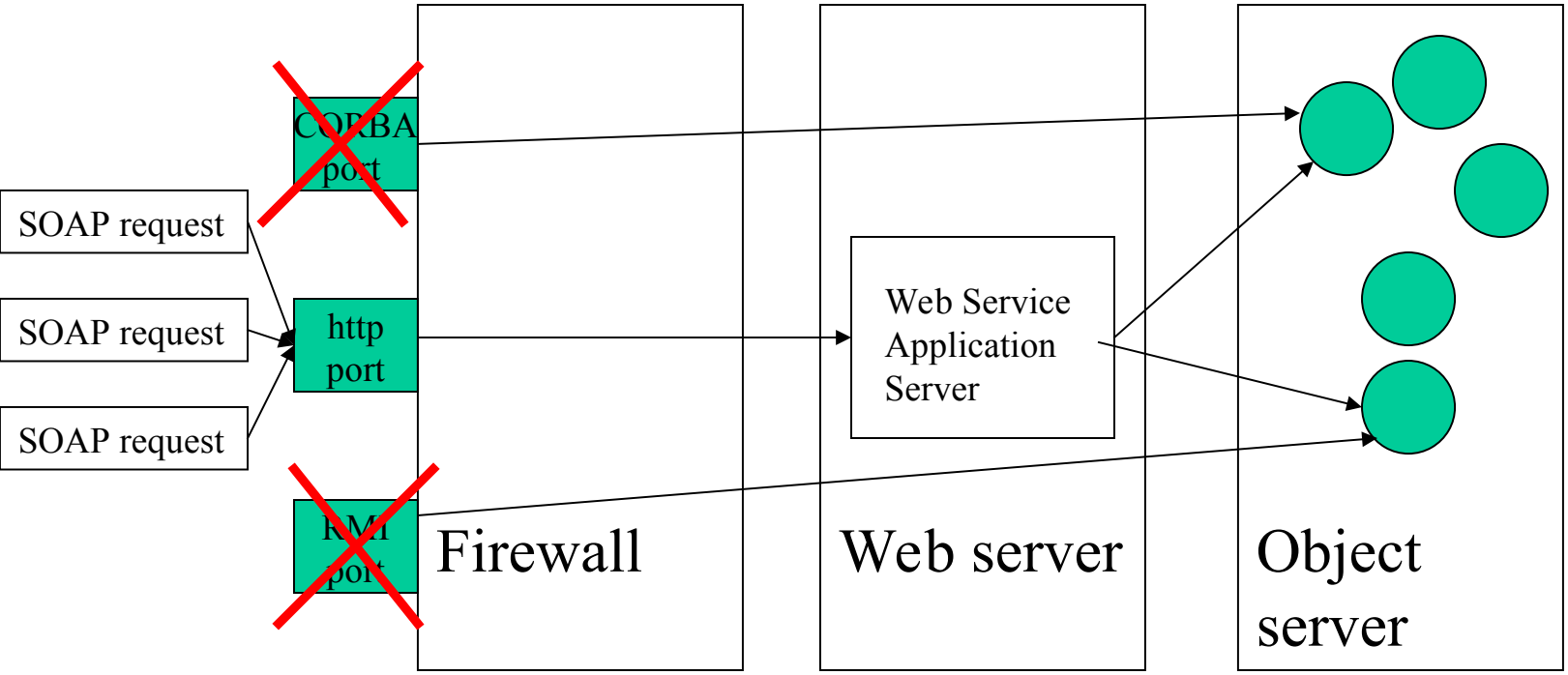
SOAP is essentially an RPC protocol with XML. It provides elements for type marshalling and RPC semantics. A header element contains meta-information but is optional. See Snell et.al. Programming Web Services... for details. Find a complete SOAP implementation at apache.org

SOAP: performance aspects



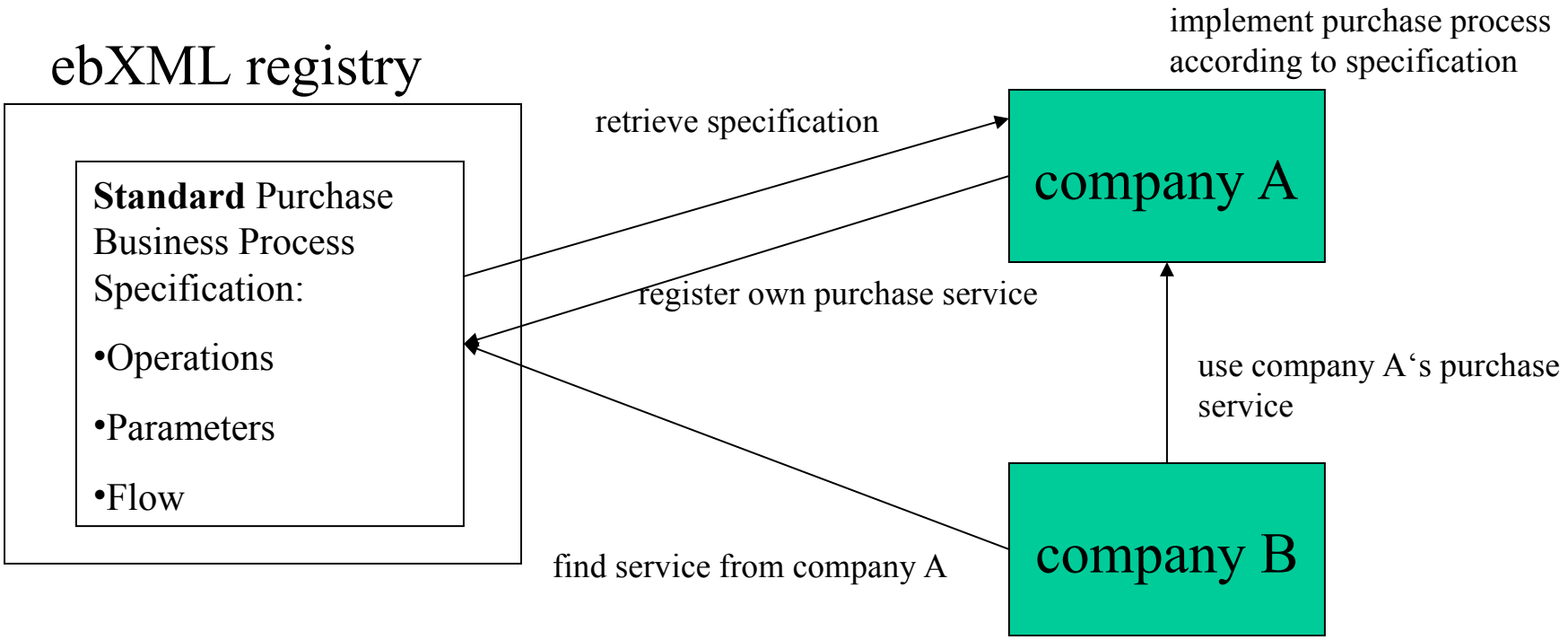
The only way to find an answer on possible performance problems is to measure the effect of individual processing steps or transport times on the overall request time. It became clear that the internet transport time with lacking QOS has far greater effects on overall request time than the size and interpretation effort of a textual format. In other words: it is NOT the XML that is problematic, it is the public network (Internet) that takes a toll on request/response protocols. (watch Amdahls law in action)

Web Services and Firewalls



The firewall „friendliness“ of Web Services has been touted all along. But firewalls were introduced for a reason: to block protocolls that cannot be tracked and filtered properly – perhaps because the necessary infrastructure was never developed – perhaps because the protocols were not intended for the Internet like CORBA and RMI. But Web Services make no sense without such an infrastructure.

Common Business Processes: ebXML



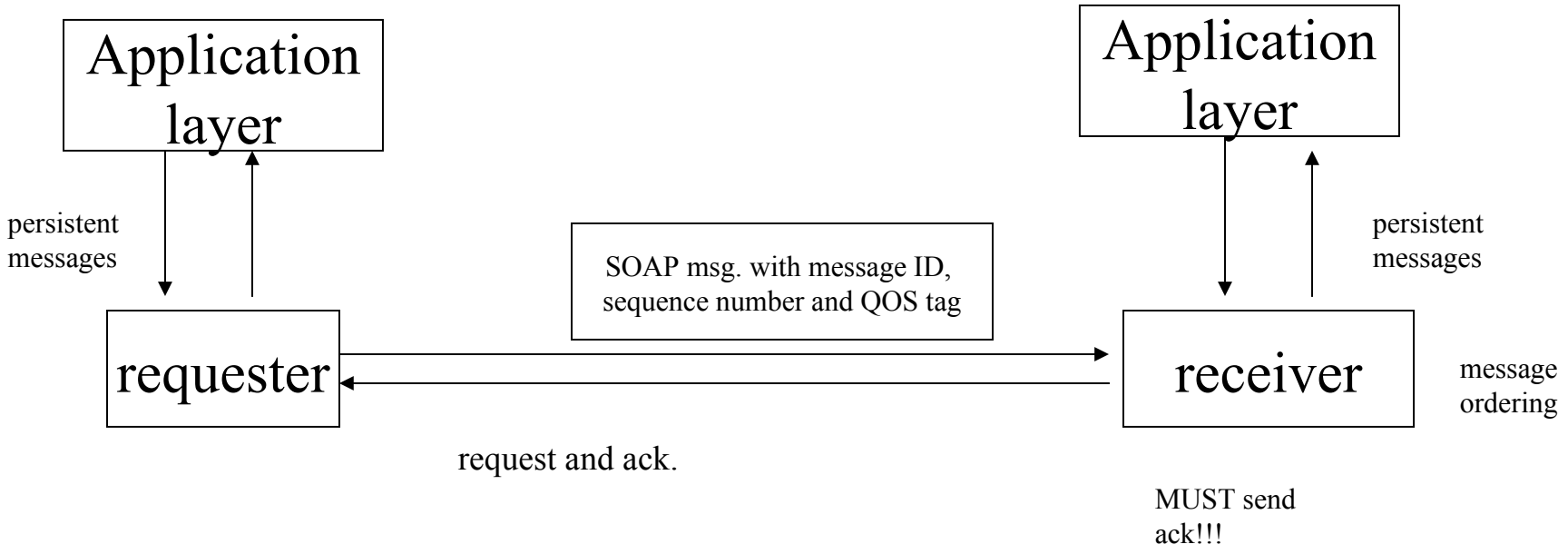
Without standard schemas for services every company will implement their business processes differently. Clients will have to deal with many different interfaces for the same type of service. ebXML is a global electronic business standard and defines a framework for defining, finding and using standard business process services. see www.oasis-open.org

Web Services Security Standards and Technologies

- SOAP, WSDL, UDDI: Message Envelope, Interfaces Definition and Registry
- WS-Security: Secure Messaging Definitions
- WS-Trust: How to get Security Tokens (issuing, validation etc.)
- WS-Federation (How to make security interoperable between trust domains)
- WS-Policy (How to express security requirements)
- Secure AssociationsMarkup Language (a language to express security related statements)
- WS-Reli (Rights management)
- WS-Util (Helper elements)
- WS-Authorization (express access rights)

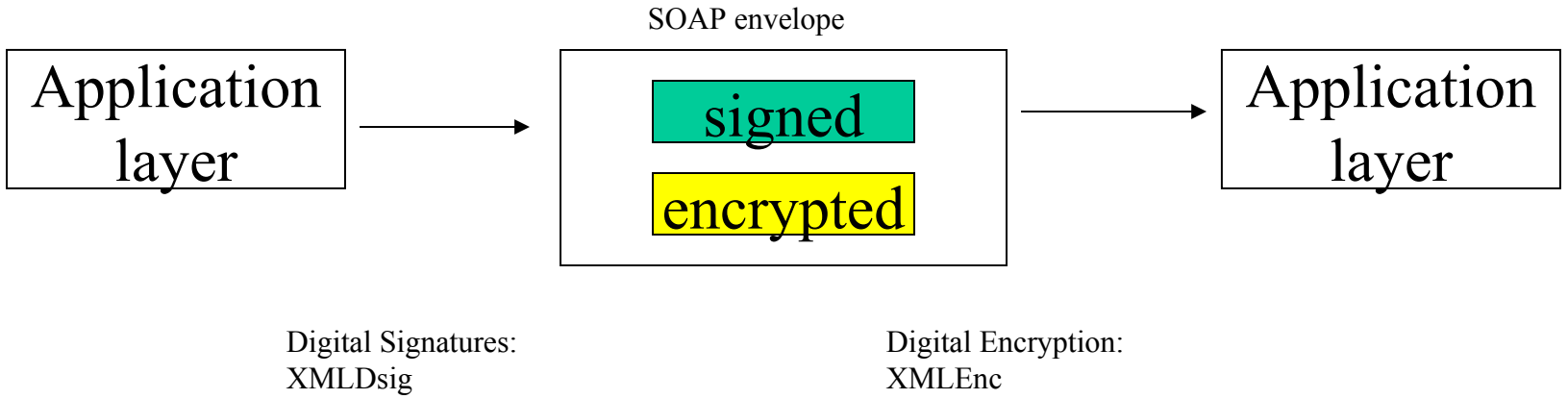
WS-Security is the foundation of Web Services Security. We will take a close look at this specification and WS-Trust. Soap and WSDL are basic Web Services standards for messaging. If you are not familiar with them, you can find an introduction here: http://www.kriha.de/krihaorg/docs/lectures/distributedsystems/webservices/webservices.html. For a list of all Web Services standards go to: www.webservicessummit.com) Please NOTE: few of those standards are available in implementations. E.g. Web Services Enhancement 2.0 from Microsoft covers only basic Web Services security features.

Reliable Messaging



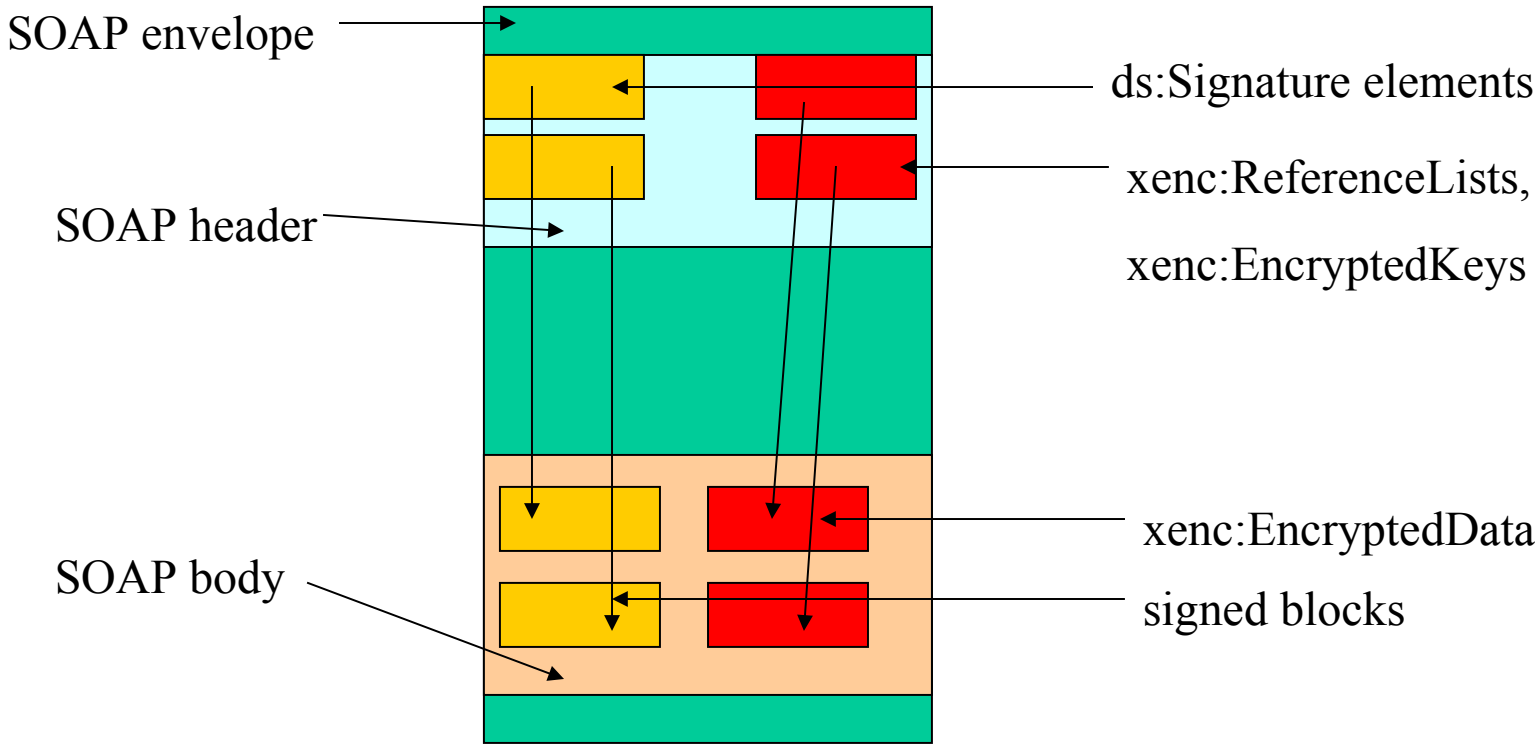
Reliable B2B messages need guaranteed delivery (ack enforced), duplicate removal (message ID) and message ordering (sequence numbers). SOAP and http do NOT provide those qualities. Further QOS extensions could be: time to hold messages, number of retries etc. Proxies are considered transparent.

Secure Messages



WS-Security goes from channel based security to message (object) based security. Individual messages can be signed and encrypted. WSDL can advertise the QOS expected/provided by a receiver. End-to-end security is possible across intermediates. See my internet security lecture for details on WS-Security, security policies and expressions (SAML, WS-Policy), WS-SecureConversation and WS-Trust. The idea of the „Virtual Organization“ – overlay structures over existing real organizations is one of the driving factors here. Today, federation is more important (see OAuth2) which is expressed in WS-Federation standard.

Using XML DSIG and XML XENC in SOAP



To make DSIG and XENC compatible with SOAP ws-security defines a number of rules, most of them having to do with the fact that Web Services are explicitely designed for use with intermediates. Those intermediates can add signatures or encryption to the SOAP envelop, e.g. to create a chain of trust. The rule here is that new signatures or encryption information is always PREPENDED to already existing information. No encryption of envelope, header or body tag is allowed. Signatures need to respect the right of intermediates to change the envelope or some header information. Again, these restrictions are the results of SOAP processing by intermediates.

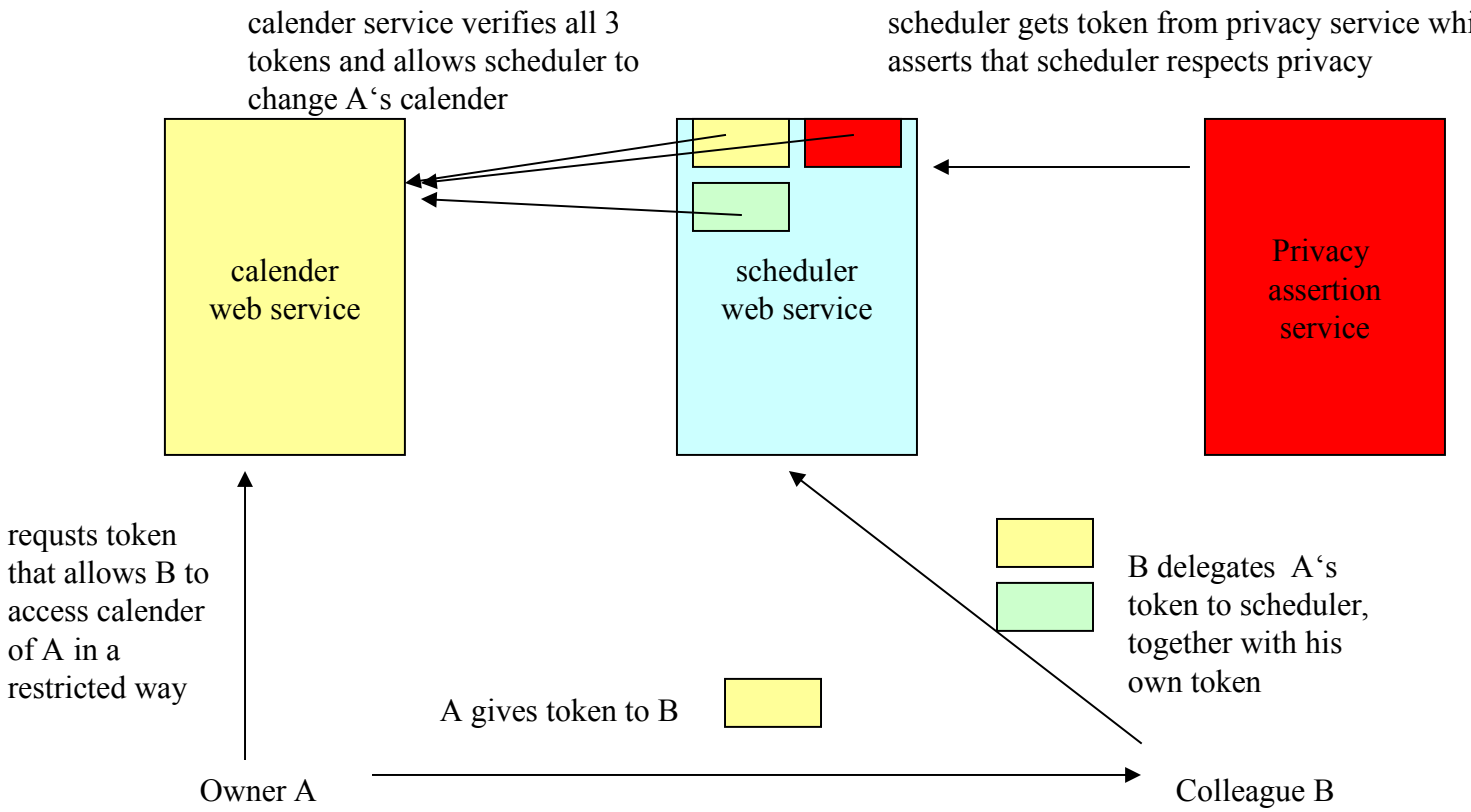
Encrypted Keys in WS-Security

```
<wsse:Security>
  <xenc:ReferenceList>
  <xenc:DataReference URI=„#foo“/>
  </xenc:ReferenceList>
<wsse:Security>
.....
<s:Body>
<xenc:EncryptedData Id=„foo“>
 <ds:KeyInfo>
 <ds:KeyName>CN=Walter Kriha, C=DE</ds:KeyName>
 </ds:KeyInfo>
 <xenc:CipherData>
<xenc:CipherValue>a5e349cddb1243....</xenc:CipherValue>
<xenc:CipherData>
</xenc:EncryptedData></s:Body>
```

```
<wsse:Security>  <xenc:EncryptedKey>
<ds:KeyInfo>.....
<xenc:CipherData>
<xenc:CipherValue>78ef34abc3412....</xenc:CipherValue>
<xenc:CipherData>
  <xenc:ReferenceList>
  <xenc:DataReference URI=„#foo“/>
  </xenc:ReferenceList> <wsse:Security>
.....
<s:Body> <xenc:EncryptedData Id=„foo“>
 <ds:KeyInfo>
 <ds:KeyName>CN=Walter Kriha, C=DE</ds:KeyName>
 </ds:KeyInfo>
 <xenc:CipherData>
<xenc:CipherValue>a5e349cddb1243....</xenc:CipherValue>
<xenc:CipherData>
</xenc:EncryptedData></s:Body>
```

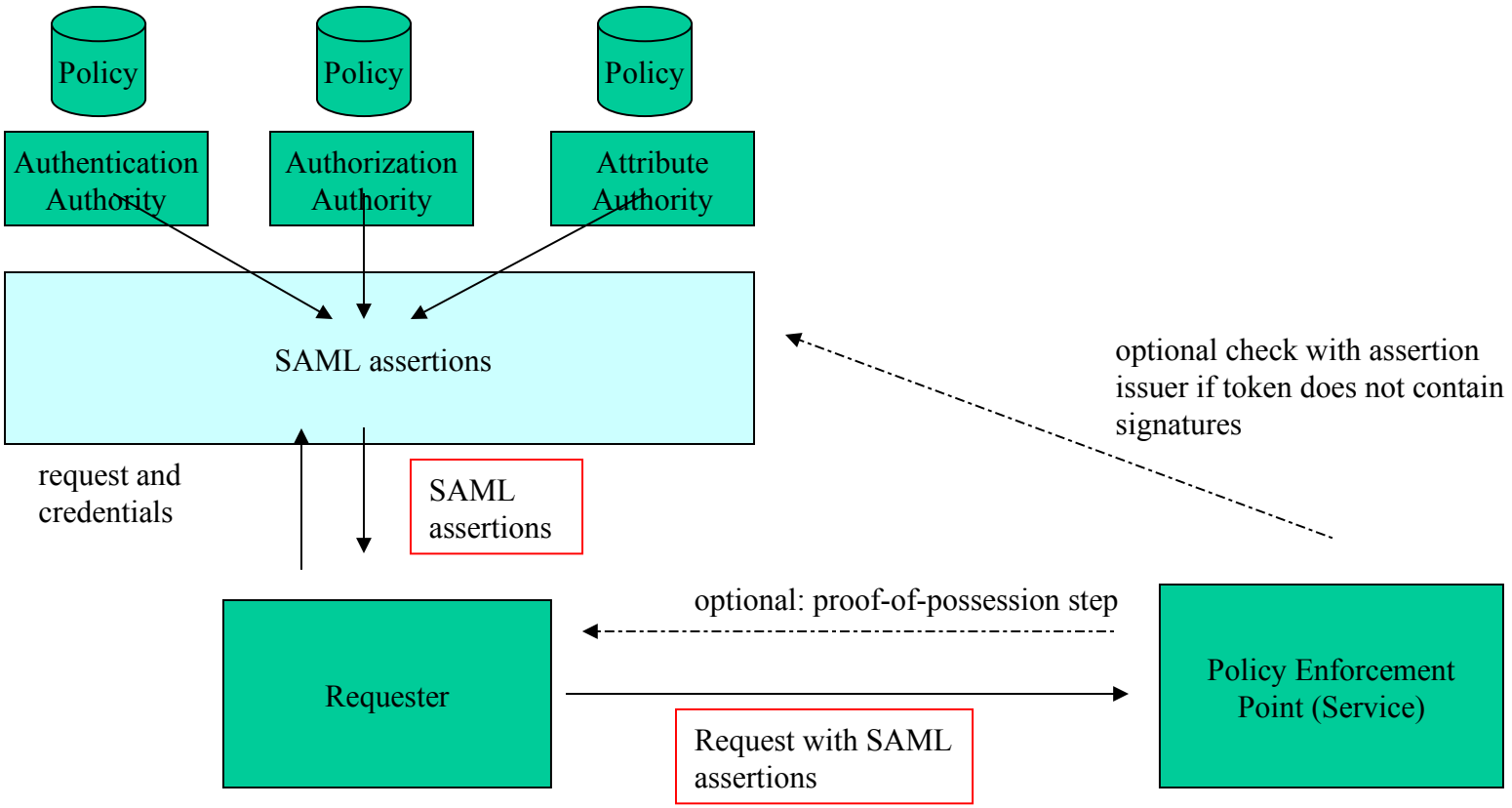
The message on the left side assumes a shared symmetric key between receiver and sender. Therefore no key is embedded or referenced. Only the key names are associated with the encrypted parts. The right side embeds an encrypted key in the message - probably encrypted using the receivers public key. The key points to the encrypted part.

Token based delegated authorization



This is an example from the roadmap for ws-security. Please note that it depends on expiration data in the tokens how often A needs to re-issue an access token for B. If B needs to access the calender frequently it might be better to use endpoint access control to restrict and control B‘s access. See next page. Just extending the expiration dates causes problems with revocation.

Secure Association Markup Language (SAML)



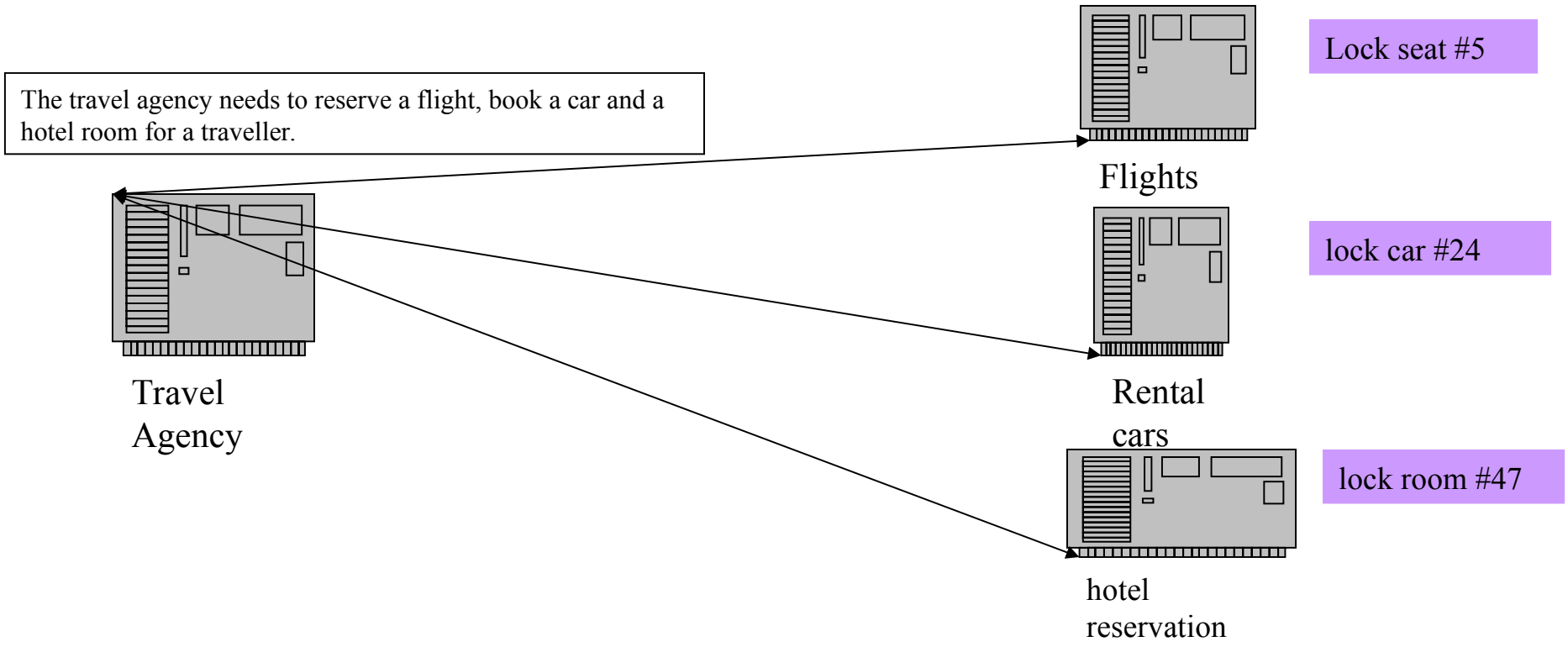
SAML allows to EXTERNALIZE all policies and mechanisms with respect to authentication, authorization and attribute assertion. The access control point needs to check only the assertions but does not have to implement all these mechanisms. On top of this, SAML makes all these statements interchangeable between different services because the format of the assertions is fixed.

Coordination and Transactions

- A generic coordination service providing coordination tpyes, context and protocol (e.g. atomic transactions, business activities
- A transaction service which covers traditional TA‘s and conversational Business Activities.

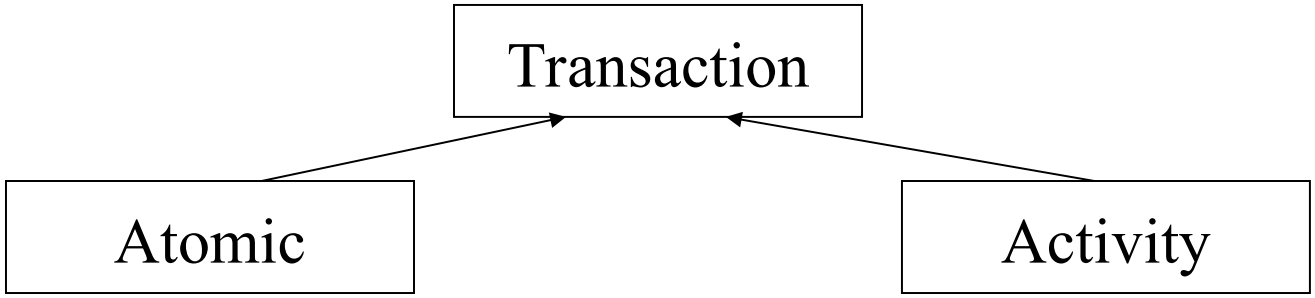
Loosely coupled services which last a long time cannot use regular locking or do a complete rollback if something fails. Business Activities comprise several low-leval atomic TA‘s but make progress even in case of individual task failures.

Transactional Web Applications



The 2-phase commit protocol does distributed transactions. But it does not really fit to the web world because it requires resource managers to lock resources. On the un-reliable medium internet this should be avoided. SOAP does not yet specify a transactional service. OASIS is working on „Business Transaction Processing“ to support Web Services Transactions. IBM is proposing a model using „tentative“ reservation to overcome the locking problem.

Transaction Models

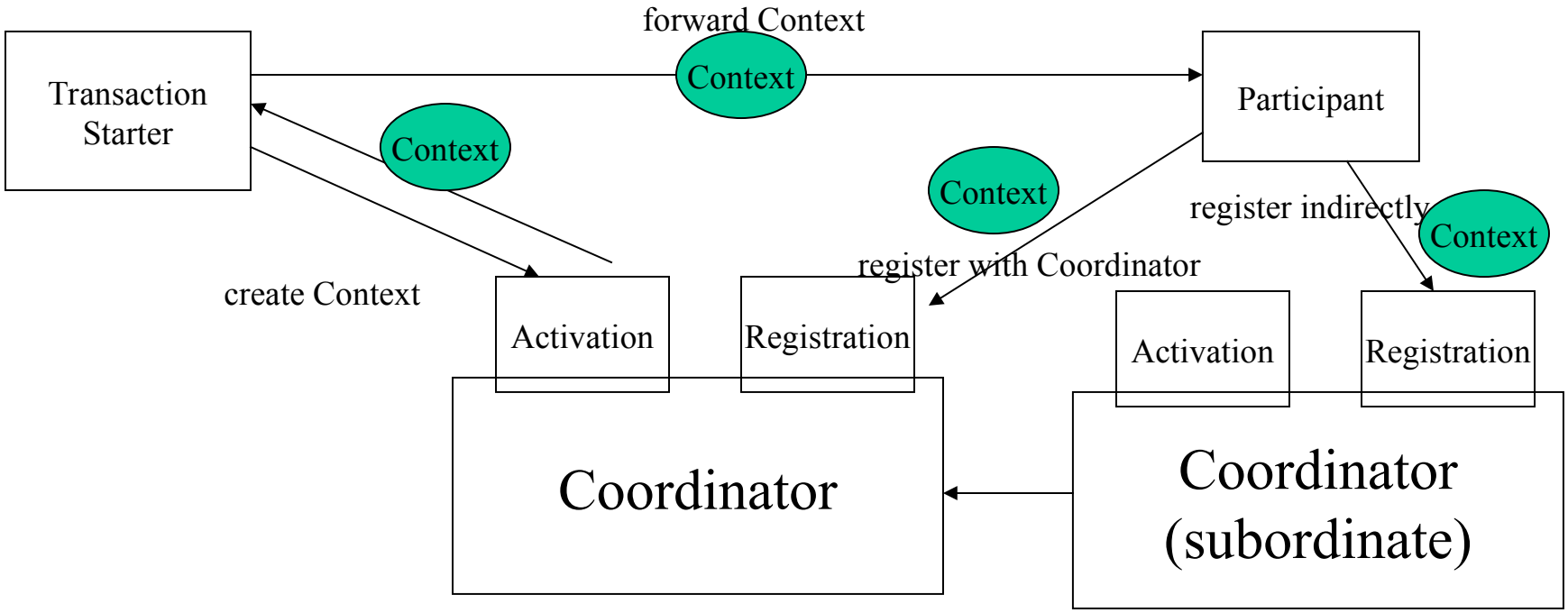


- not nested
- short
- tightly coupled business task
- rollback in case of error
- errors: system crash

- nested tasks
- long running
- loosely coupled business activity
- compensating tasks and activities
- errors: order cancellation etc.

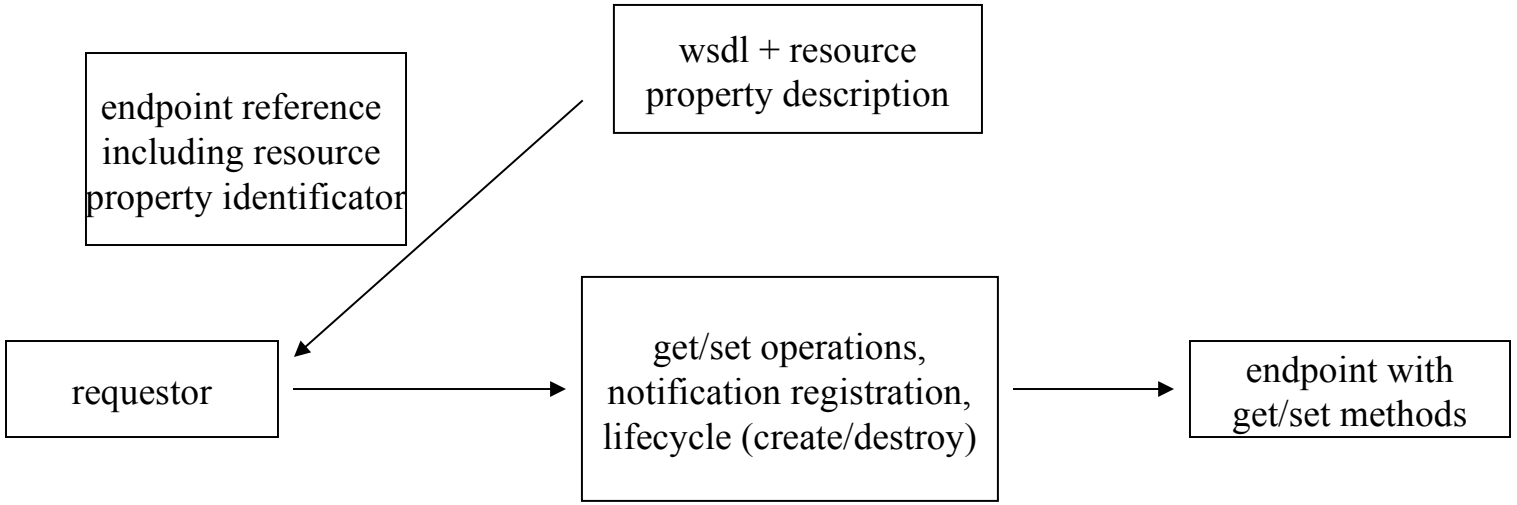
Complete rollback is too expensive for long running business activities. Intermediate results must be visible early – compensating acting try to undo tasks in case of business errors. Internet site do not like the locking of resources by external callers...

A Coordinator



Activation works like a factory method to create a new Coordination Context. This context is forwarded to participants which register through it either directly with the first coordinator or with their own coordinator which registers itself as a sub-coordinator. Protocol and type of coordination are contained in context.

Stateful Web Services



Stateful architectures like computational grids need the concept of a resource. WS-Resource adds this via meta-data descriptions contained in the WSDL and WS-Adressing schemas. An identifier is used to communicate state information between requestors and endpoints. On top of WS-Resource advanced notification requests can be built. See: The WS-Resource Framework (Czaikowski, Ferguson et.al.)

Best Practice for Promoting Scalable Web Services

1. Stay away from using XML messaging to do fine-grained RPC. For example, stay away from a service which returns the square root of a number. Stay away from a service that returns a stock quote (this is the classic-cited example of a Web service).

2. Conversely, use course-grained RPC. XML web services usually have to be defined at a coarser granularity than ordinary software objects. That is, use Web services that "do a lot of work, and return a lot of information".

3. When the transport may be slow and/or unreliable, or the processing is complex and/or long-running, consider an asynchronous messaging model.

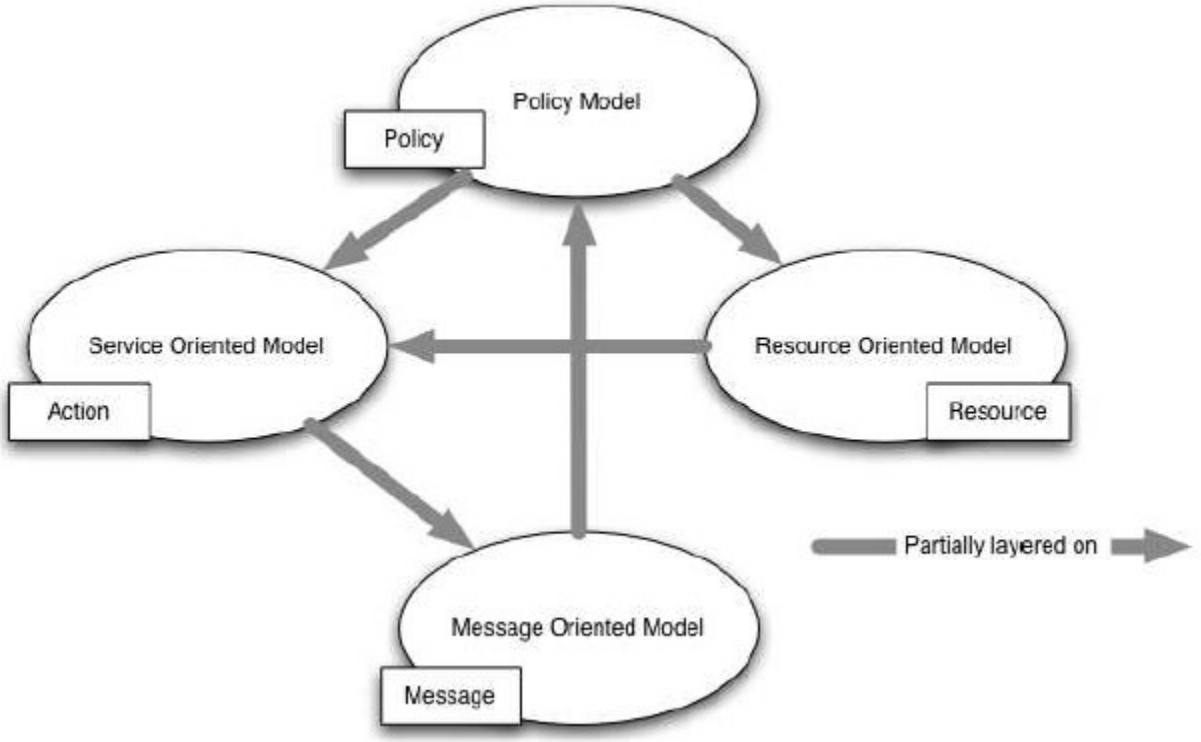
4. Always take the overall system performance into account. Don't optimize until you know where the bottlenecks are, i.e., don't assume that XML's "bloat" or HTTP's limitations are a problem until they are demonstrated in your application.

5. Take the frequency of the messaging into account. A high rate of requests might suggest that you load (replicate) some of the data and processing back to the client. The client occassionally connects to synch-up with the server, and get the lastest data.

6. Aggregation using replication. There will be Web services which serve to aggregate data from other Web services. One approach is to perform the aggregation on demand - the services which supply the data are invoked in real time, the data is aggregated, and returned to a requesting client. Alternatively, all the data from the supplier services may be retrieved during off-hours in one large, course-grained transaction. Thus, the aggregation is performed in real-time (rather than trying to retrieve the supplier data in real-time). The later is recommended whereever possible.

this is the result of an interesting discussion at xml-dev. Do you agree?

Other Web Services Architectural Models



Representational State Transfer Architecture (REST), SOA and Policies are models beyond mere messaging. Diagram taken from „Web Services Architecture (w3c)

Service Oriented Architecture (SOA)

Beyond WebServices

Example

Let's look at my effort on monday to get Wilco's cover of Blue Oyster Cult's Don't Fear The Reaper into my iPod and posted on my blog.
This effort required to integration of about eight web services, most of which were supplied by individuals, not businesses.
Web Service #1 - Wilcoworld webcasts the Fillmore Show live over the internet
Web Service #2 - Somebody records the internet stream using Total Recorder
Web Service #3 - HappyKev uploades the Bittorrent of the show into etree
Web Service #4 - Wilcobase posts the setlist from the Fillmore show
Web Service #5 - Bloglines shows me the setlist via RSS
Web Service #6 - I find the torrent on etree and download it using Azureus
Web Service #7 - I convert the files to MP3 using dbPowerAmp
Web Service #8 - I blog it using Typepad

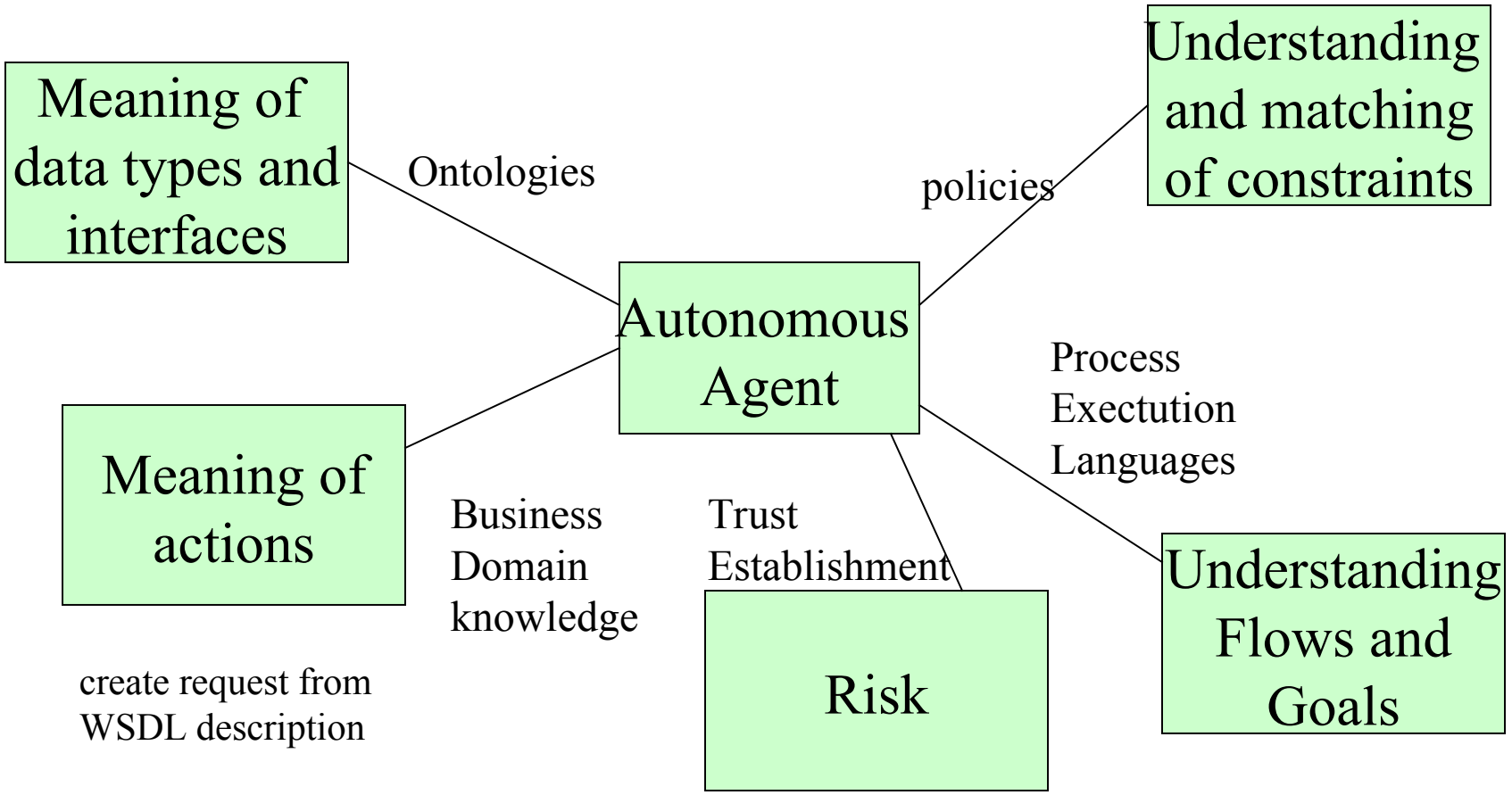
Taken from http://avc.blogs.com/a_vc/2004/11/the_architectur.html to show you web services that work. Most public WebServic es were much simpler than the interfaces defined in the WS-specifications and used a REST API on top of this!

Why UDDI could not work

- central registries of service descriptions
- independent automatic agents searching for services
- machines understanding service descriptions
- machines deciding on service use
- machines being able to use a service properly
- machines being able to construct advanced workflows from different services

A hype must be pretty big to make people forget about the problems behind above assumptions. But it gets worse: even if you replace machines with human beings (e.g. for the service decision) UDDI does not work: Too much in services is ambiguous, undefined or expressed in different and incompatible terms – not to forget that the service interface use (order of calls, meaning of datatypes etc.) is undefined as well.

Missing Technology behind UDDI



Not to forget things like business languages which standardize business terms like contract, sale, customer etc. Generally speaking a ton of meta-data where missing. Webservices (WSDL, SOAP) merely covered the mechanics of message exchange.

Lessons Learned from WebServices and CORBA

- Webservices are a low-level concept which need more semantics
- Workflow has not really been covered by WS and CORBA
- SOA is not only about interfaces and interface design. In the first place it is about HOSTING SERVICES. An ounce of a really available service on the web is worth more than a ton of specifications and interfaces

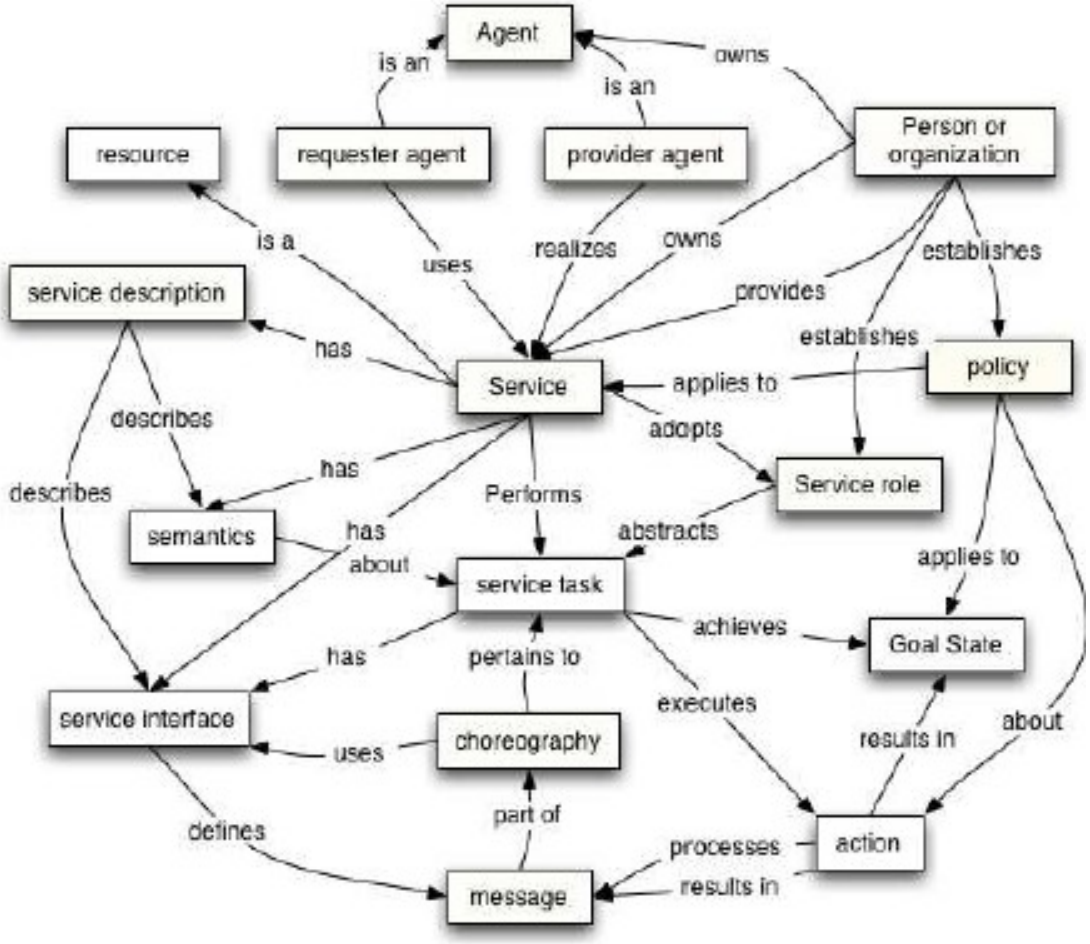
Many of the concepts now sold under „SOA“ have been expressed in the 90‘s e.g. in the „Business System Application Architecture“ of the OMG (www.omg.org)

SOA Core Properties

- Services offer high-level, business type interfaces
- Service Choreography (aka workflow) is performed outside services which allows the combination of services into larger business processes
- A set of semantic standards and technologies allows agents to understand services and their interfaces (OWL, SAML, Semantic Web etc.)
- Legacy applications will be wrapped through a service interface and become available to other companies
- SOA will use Web Service technology at its base

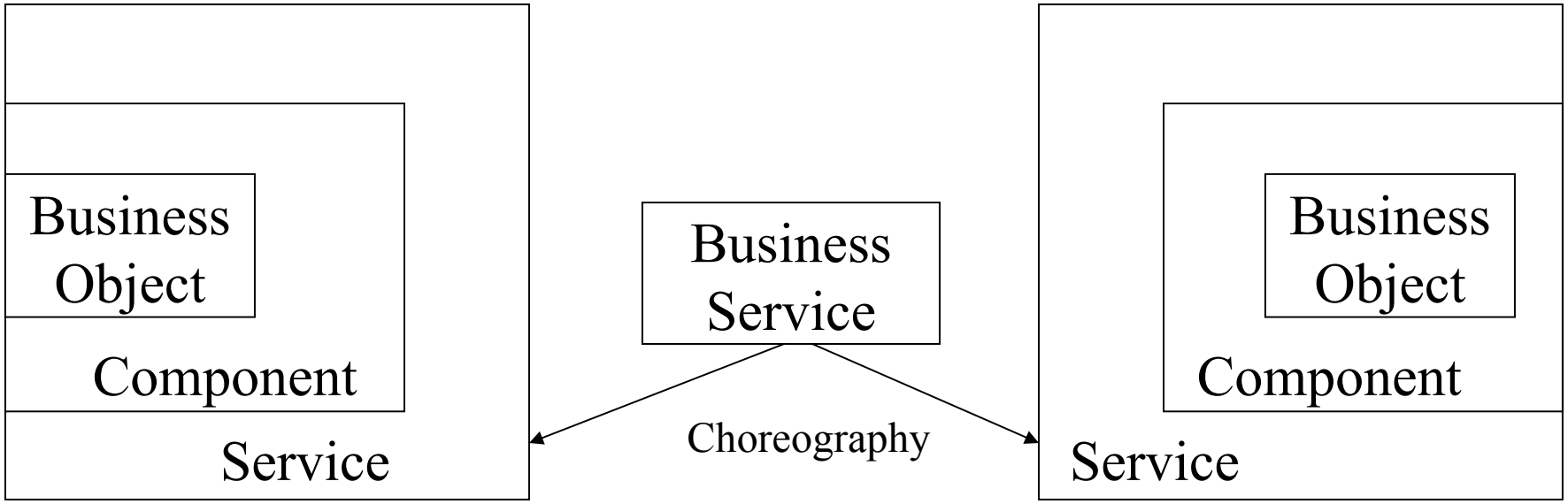
It is interesting to see the how the industry seems to shy away from the term „workflow“ in this context. Too many projects gone south and too many unfulfilled promises?

SOA Architectural Model



This diagram from „Web Services Architecture“ (see resources) shows internal and external elements of the SOA architecture. Action e.g. is not an externally visible element. Note the important roles of „policy“ and „semantics“

SOA Design



This diagram is modelled after O.Zimmermann et.al. „Elements of a Service-Oriented Analysis and Design“ (see resources). The paper also shows nicely how flow oriented a SOA really is and that a class diagram does not catch the essence of SOA. A state-diagram performs much better. The authors also note that SOA is process and not use-case driven design.

Interface Design

Object interface: can be conversational, accepts transactions, fast, Object references

Component interface: value objects, transaction border, relatively fast. Mostly stateless.

Service interface: long running transactions with state in DB. Compensation Functions. Short process time, long business TA time. Isolated and independent. Composable to larger services (choreography) or composed of smaller services (orchestration). Stateless.

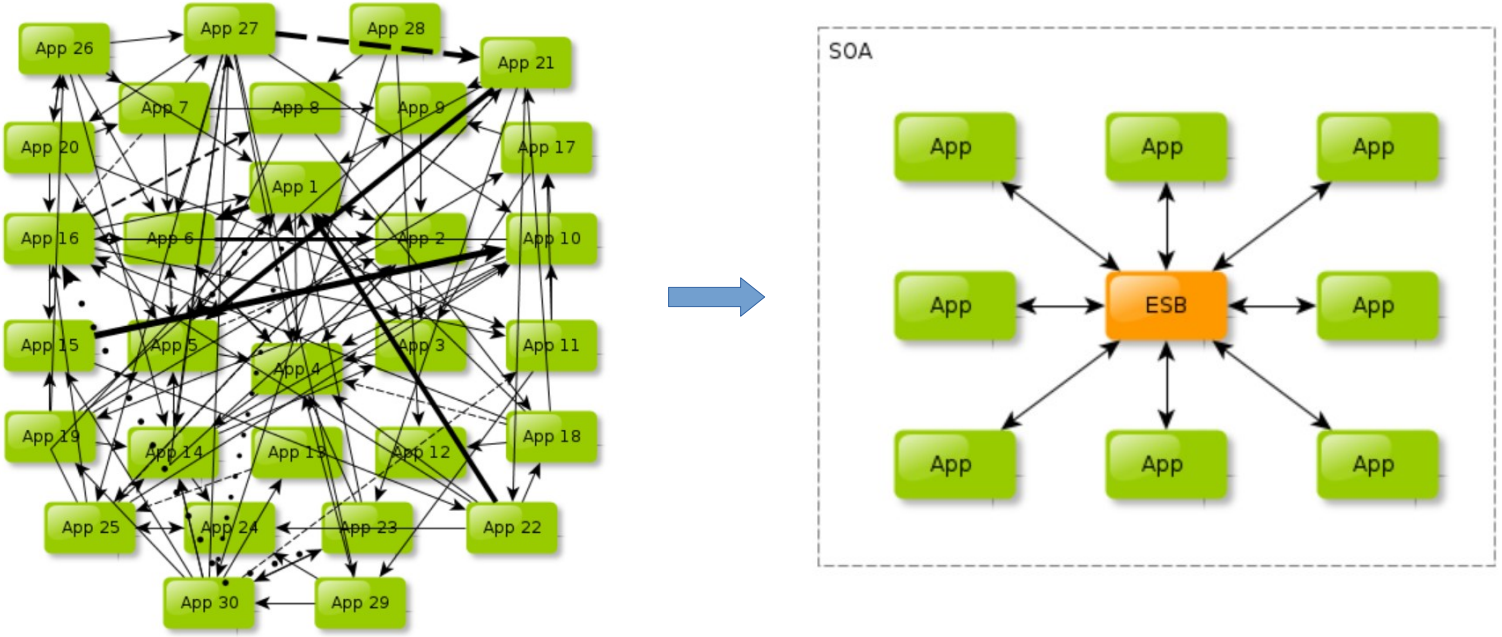
Only objects (classes) are programming language constructs. But a detailed look at the interfaces reveals that component and service type interfaces are just a different type of interface model.

SOA Blueprint Service Types

- Component Service: atomic operation on a simple object (e.g. DB-access)
- Composite Service: atomic, uses several simple services (orchestration), stateless for caller.
- Workflow Service: Stateful, defined state changes (state kept in persistent store)
- Data Service: Information integration via message based request/response mechanism.
- Pub/Sub Service: typical event service with callbacks and registration.
- Service Broker: Intermediate, rule based message manipulation and forwarding
- Compensation Service: revert actions (not rollback like)

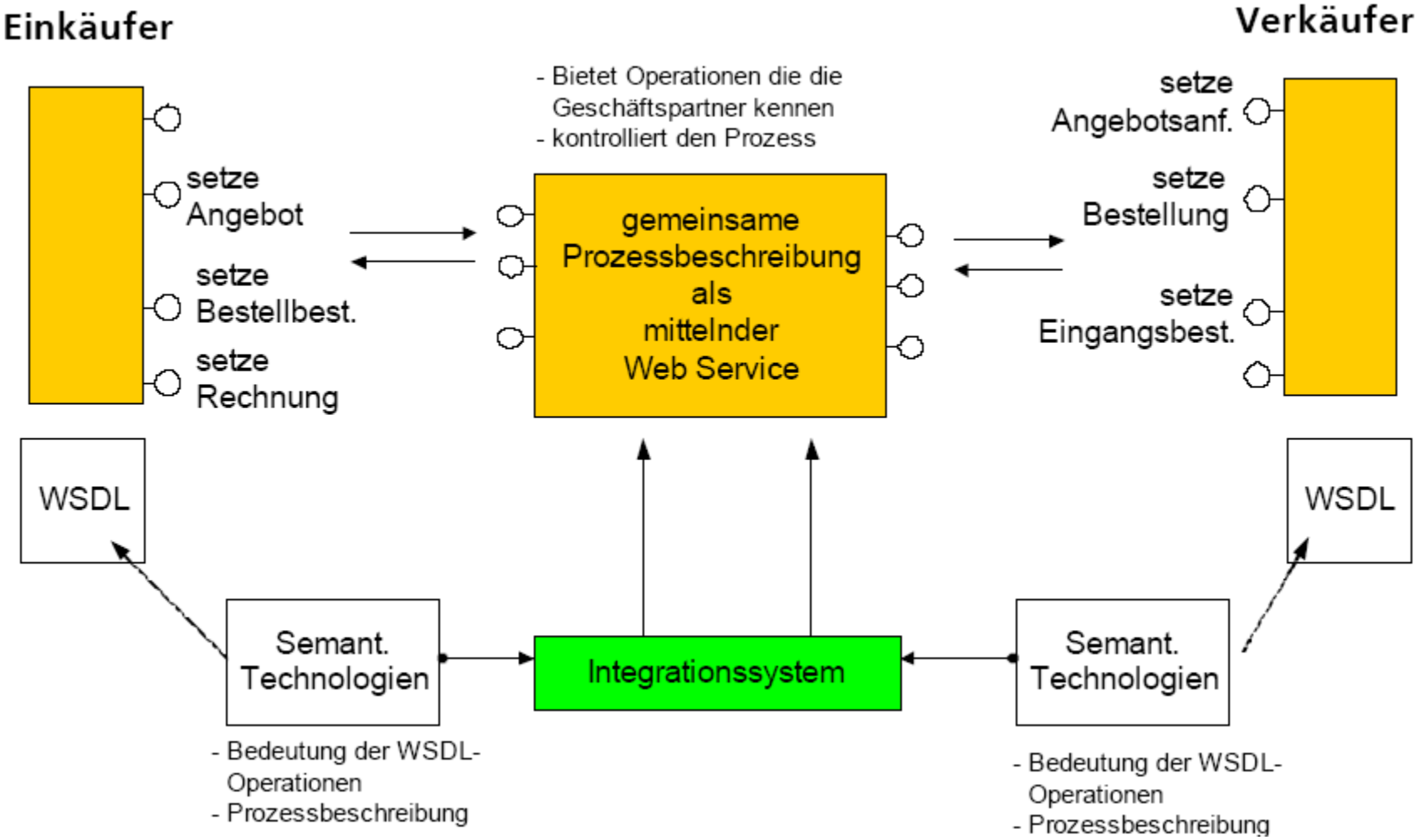
From the SOA Blueprint of The Middleware Company (TMC) found in: Soa auf dem Prüfstand (see resources)

Event-Driven SOA: ESB



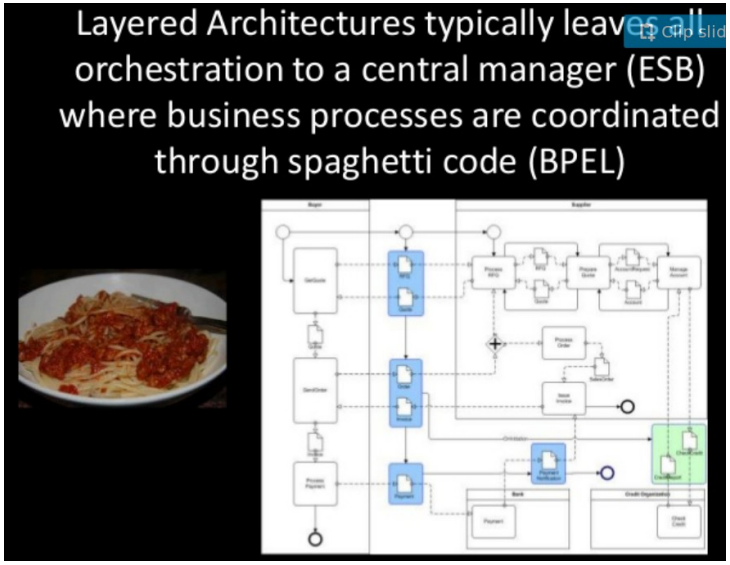
The concept of an Enterprise Service Bus played a major role in SOA: The bus was supposed to allow loose coupling between apps, format conversions, notifications and pub-sub features.
(from:https://zato.io/docs/intro/esb-soa.html

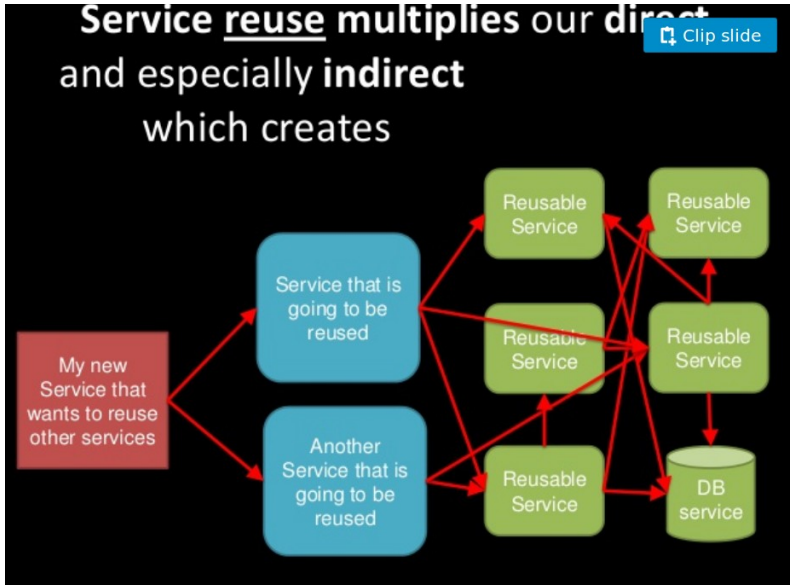
Meta1:Automatic Service Composition



An e-procurement example using semantic and generative technologies (Christoph Diefenthal/Fraunhofer IAO)

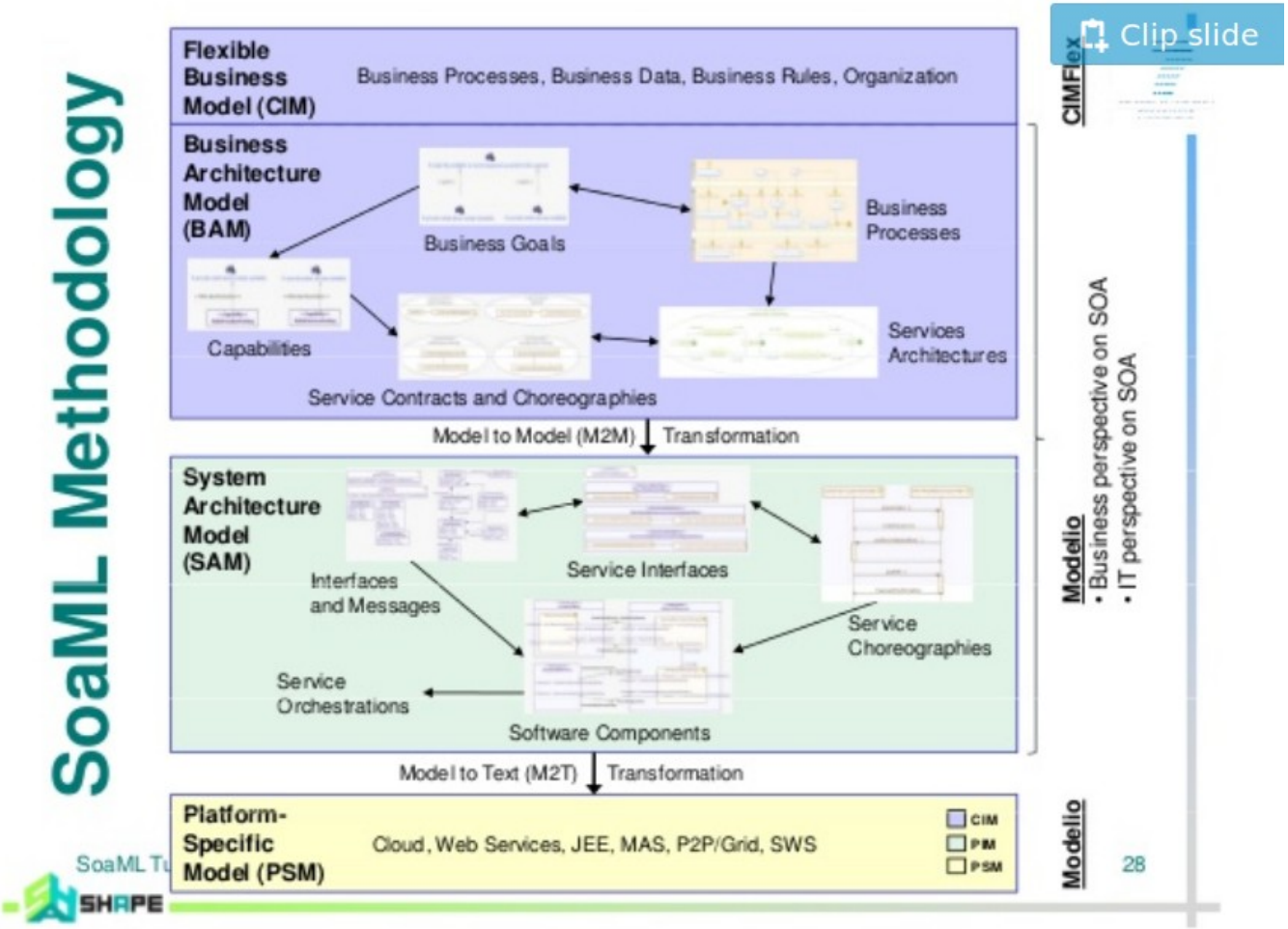
SOA: Critical Points





Jeppe Cramon, SOA and Event Driven Architecture (SOA 2.0)
http://www.slideshare.net/jeppec/soa-and-event-driven-architecture-soa-20

Meta to the Rescue...



B. Elvesæter, "Service Modelling with SoaML", tutorial at the SSAIE 2010
http://www.slideshare.net/elvesater/be-ssaie-2010soamlpresentation

SOA vs. Microservices: Service Taxnonomy

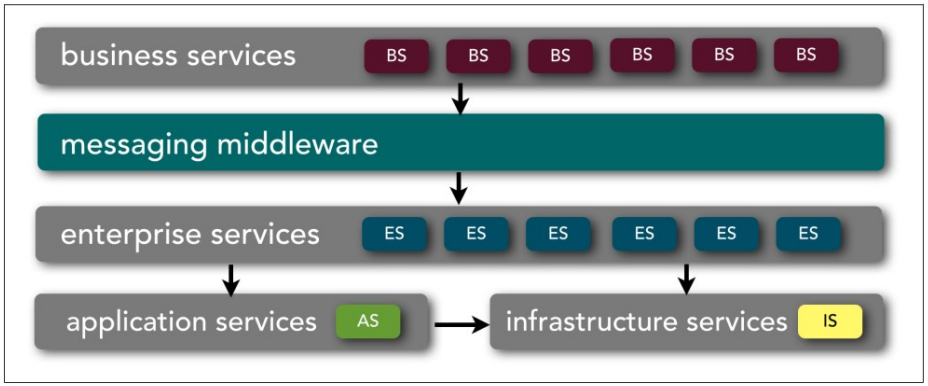


Figure 2-2. SOA taxonomy

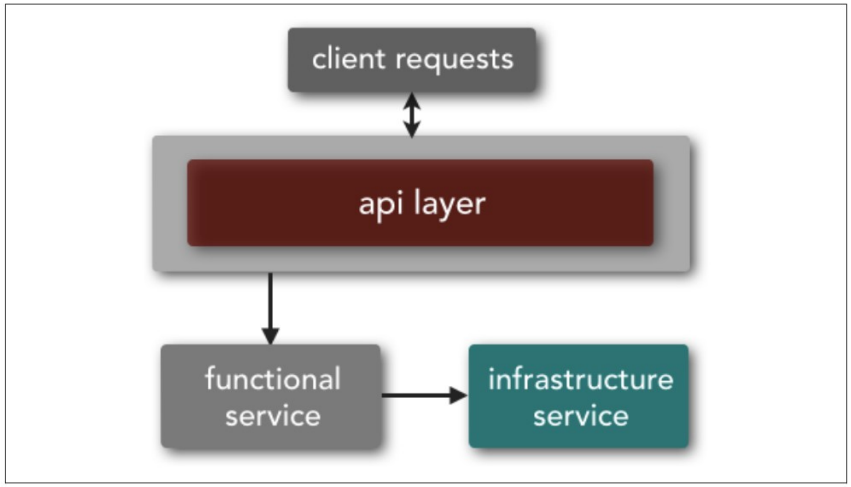
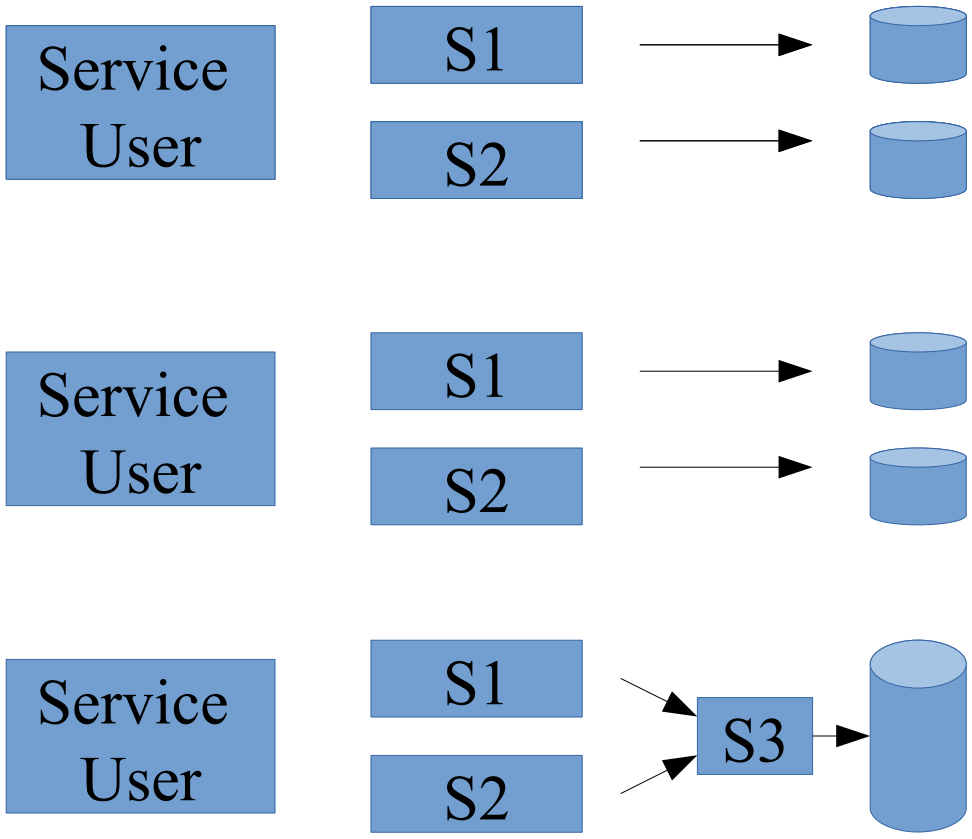


Figure 2-1. Microservice service taxonomy

SOA services are much more detailed and owned by different groups. See: Mark Richard, Microservices vs. SOA (Oreilly, sponsored by NginX)

SOA vs. Microservices: Service Granularity



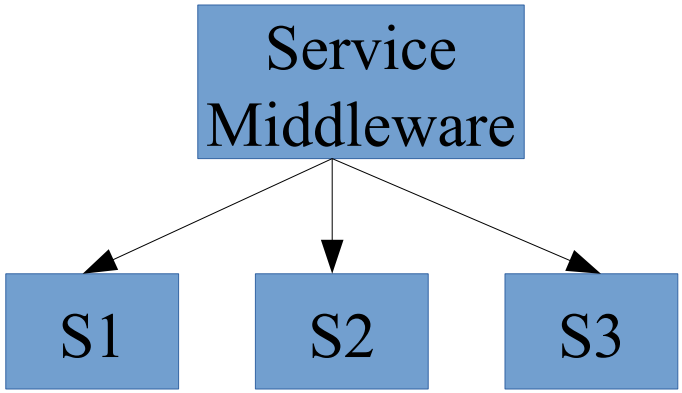
No transactional bracket, different DBs

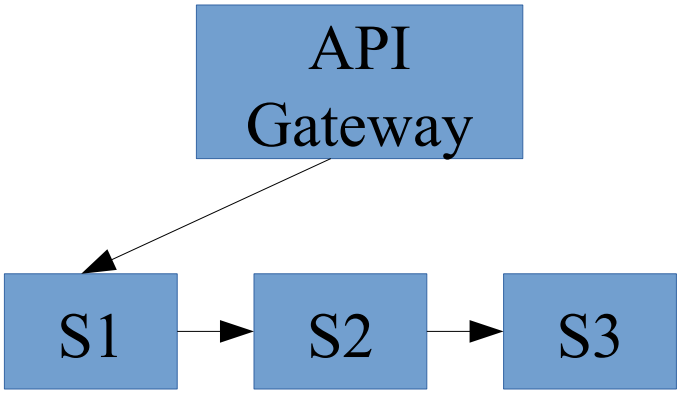
Distributed Transaction (costly, fragile)

Shared enterprise service (transactional)

SOA services are more often transactional and therefore larger than MS. MS use eventual (BASE) technologies (eventual consistent stores, event-sourcing approaches) which are NOT ACID!

Service Arc.: Orchestration vs. Choreography





SOA prefers orchestration due to higher level middleware components

Microservices prefer choreography which can lead to highly connected and dependent systems

Mark Richard, Microservices vs. SOA (Oreilly, sponsored by NginX)

SOA vs. Microservices

- much more enterprise architecture with middleware (messaging) , service layers and ownership concepts
- more contract decoupling using meta-data and messaging middleware
- “share as much as possible” approach
- big services are transactional

- simple API gateway, teams own infrastructure and business services
- no contract decoupling
- “share as little as possible” approach
- services offer only BASE consistency (eventual)

Representational State Transfer (REST)

RESTful Web – against the RPC Model

-The WEB is based on representation of resources using URIs, Web Services create private, non-standard ways of information access

-The envelope paradigm does not add any value over the generic http get/put/post

-RPC mechanisms are not suitable for the WEB. Some extensions to get/put/post might be necessary though (going in the direction of tuple-space systems)

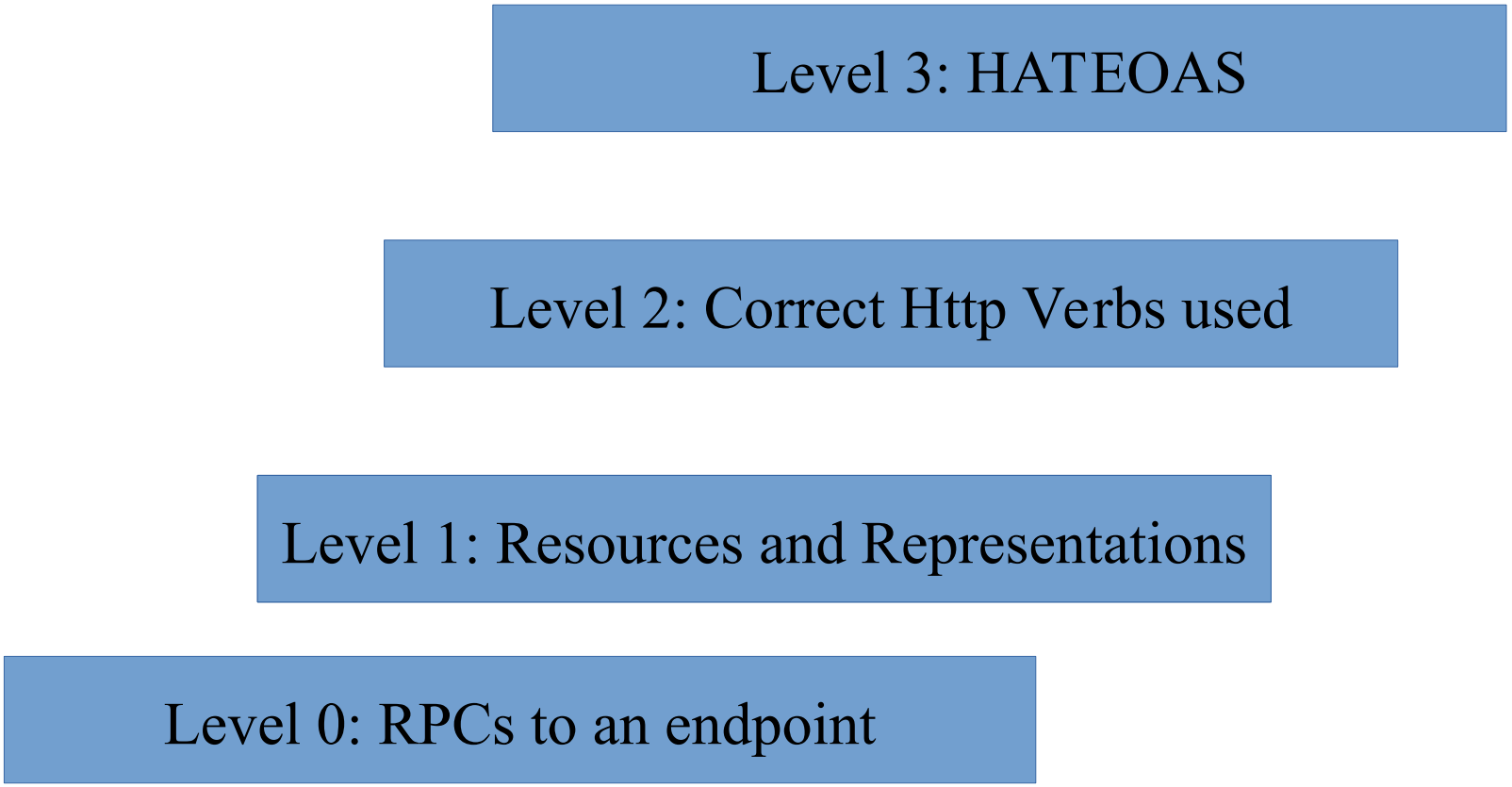
This is a hot topic currently: ask yourself whenever you think about building a web services: could it be done with just an http get or post? REST btw. stands for Representational State Transer Architecture, a term coined by Roy Fielding, the father of http. see resources on REST. But in later versions Web Services have been extended through a document centric model as well.

The Web's Architectural Style

- Client-server
- Uniform interface
  - Identification of resources (URI)
  - Manipulation of resources through representations
  - Self-descriptive messages (Meta-data, header, convers.)
  - Hypermedia as the engine of application state (HATEOAS): Response contains actionable links
- Layered system (Intermediaries for caching, security, LB)
- Cache (Declare the cacheability of a response)
- Stateless (Clients need to provide context/state)
- Code-on-demand (Server can send scripts, flash, applets etc.)

from: M.Masse, REST API Desing Book

REST Maturity Model



See resources: Richardson, Fowler

REST Level 0: RPC

Client

Service

POST /appointmentService
<openSlotRequest doc = “Webster”, date=“12_12_2020” />

<openSlotList doc=”Webster”, time=”15.00-16.00” date=” 12_12_2020”>

POST /appointmentService
<appointmentRequest PatientID=”WSmith”,doc=”Webster”, time=”15.00-16.00”
“date=12_12_2020“ />

OK: <appointment doc=”Webster”, time=”15.00-16.00” date=” 12_12_2020”>
OR: <appointmentRequestFailure doc=”Webster”, time=”15.00-16.00” date=”
12_12_2020”>

After (modified): Fowler, RMM. This looks like regular RPC to one endpoint. The appointment made does not show up as a resource and is not accessible without a new RPC function

REST Level 1: Resources

Client

Service

POST /doctors/webster/slots
<openSlotRequest date=”12_12_2020”/>

<openSlotList appointment= “14” doc=”Webster”, time=”15.00-16.00” date=” 12_12_2020”/>

POST /doctors/webster/slots/14
<appointmentRequest patient=”WSmith” />

OK: <appointment doc=”Webster”, time=”15.00-16.00” date=” 12_12_2020”>
OR: <appointmentRequestFailure doc=”Webster”, time=”15.00-16.00” date=” 12_12_2020”>

After (modified): Fowler, RMM. Function names and parameters have been turned into resources. Appointments do have an identity now and anybody can get/post something to an existing appointment

Rest Resource Archetypes

- Document: Fields and links (base resource, noun, create:POST)
- Collections: Containers maintained by server with URI generation (noun, POST)
- Stores: Container elements maintained by client with “put” and without URI generation on server side (noun, insert/update:PUT)
- Controllers: Procedures (verb, POST)
- URI Path Design: Reflects resource model
- Variable path segments with query terms

After: M.Masse, REST API Design Book

REST Level 2: Http Verbs

Client

Service

GET /doctors/webster/slots?date=12_12_2020&status=open

<openSlotList appointment= “14” doc=”Webster”, time=”15.00-16.00” date=”12_12_2020”/>

POST /doctors/webster/slots/14
<appointmentRequest patient=”WSmith” />

201 created <appointment doc=”Webster”, time=”15.00-16.00” date=” 12_12_2020”>
OR: 409 conflict <openSlotList appointment= “14” doc=”Webster”, time=”16.00-17.00” date=” 12_12_2020”/>

After (modified): Fowler, RMM. It is now crucial to use the correct verb. GET is idempotent and creates cachable resources. The response codes have to be used correctly, in this case to indicate a new resource or a conflict. Do not use OK codes and report an error in the body.

REST Level 3: HATEOAS

Client | Service

GET /doctors/webster/slots?date=12_12_2020&status=open

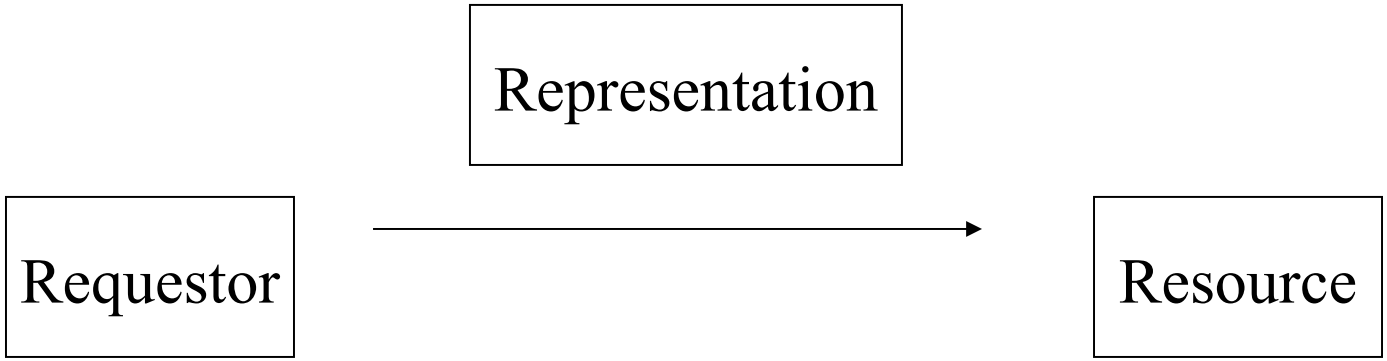
```
<openSlotList appointment= “14” doc=”Webster”, time=”15.00-16.00” date=”20201212”>
        <link rel=”/linkrels/slot/book” uri = “/slots/14”/>
```

POST /doctors/webster/slots/14
<appointmentRequest patient=”WSmith” />

```
201 created
<appointment doc=”Webster”, time=”15.00-16.00” date=”20201212”>
<slot id = "14" >
<patient id = "wsmith"/>
        <link rel = "/linkrels/appointment/cancel"                  uri = "/slots/14/appointment"/>
        <link rel = "/linkrels/appointment/addTest"                 uri = "/slots/14/appointment/tests"/>
        <link rel = "self"                                          uri = "/slots/14/appointment"/>
        <link rel = "/linkrels/appointment/changeTime"              uri = "/doctors/webster/slots?date=20201212@status=open"/>
        <link rel = "/linkrels/appointment/updateContactInfo"       uri = "/patients/wsmith/contactInfo"/>
        <link rel = "/linkrels/help"                                uri = "/help/appointment"/>
</appointment>
```

After (modified): Fowler, RMM. Responses now encode optional actions which can be invoked by the client. Services can change URIs without breaking clients. The rel attibute describes the semantics behind the URI link.

RESTful Web: CRUD like Message Semantics



GET -> Read (idempotent, does not change server state)
POST –> Create resource on the server
PUT -> Update Resource on the server (??)
DELETE -> Delete Resource on server

Is this separation of updates and reads something new? Not by far. Bertrand Meyer of OO fame calls this a core principle of sound software design and made it a requirement for his Eiffel programming language. He calls it “”command-query separation principle”:
“Commands do not return a result; queries may not change the state – in other words they satisfy referential transparency” B. Meyer, Software Architecture: Object Oriented Versus Functional [Meyer]

RESTful Web Features

Four strands that make a servive RESTful:

- explicit use of http protocol in a CRUD like manner
- stateless design between client and server
- meaningful URIs which represent objects and their relationships in the form of directory entries (mostly parent/child or general/specific entity relations)
- use of XML or JSON as a transfer format and use of content negotiation with mime types

All state change is reflected by a change in representation. Resources are manipulated through a very simple and uniform interface (CRUD like) and through the exchange of representations. This is how the WWW works. A subset of Web Services are REST-compliant. From A.Rodriguez, see resources.

REST: critical points

- At-least-once delivery of requests?
- At-most-once delivery of requests?
- Transactions (optimistic, ETAG)?
- (Federated) Security with bearer tokens?
- Secure delegation and backend security?
- Performance over http?
- Too many round-trips? (Orchestration API)

When new requirements come along, developers face a choice: Should we create a new endpoint and have our clients make another request for that data? Or should we overload an existing endpoint with more data?Developers often choose the 2nd option because it's easier to add another field and prevent a client-to-server round trip. Over time, this causes your API to become heavy, kludgy, and serve more than a single responsibility

From: https://medium.com/paypal-engineering/graphql-a-success-story-for-paypal-checkout-3482f724fb53

There are solutions for those problems, but for inter service calls (see microservices later) it is often more convenient to use an RPC protocol (thrift, protocol buffers etc.)

Post REST?

• Messaging and Eventing · This approach is all over, and I mean all over, the cloud infrastructure that I work on. The idea is you get a request, you validate it, maybe you do some computation on it, then you drop it on a queue (or bus, or stream, or whatever you want to call it) and forget about it, it's not your problem any more.

¶

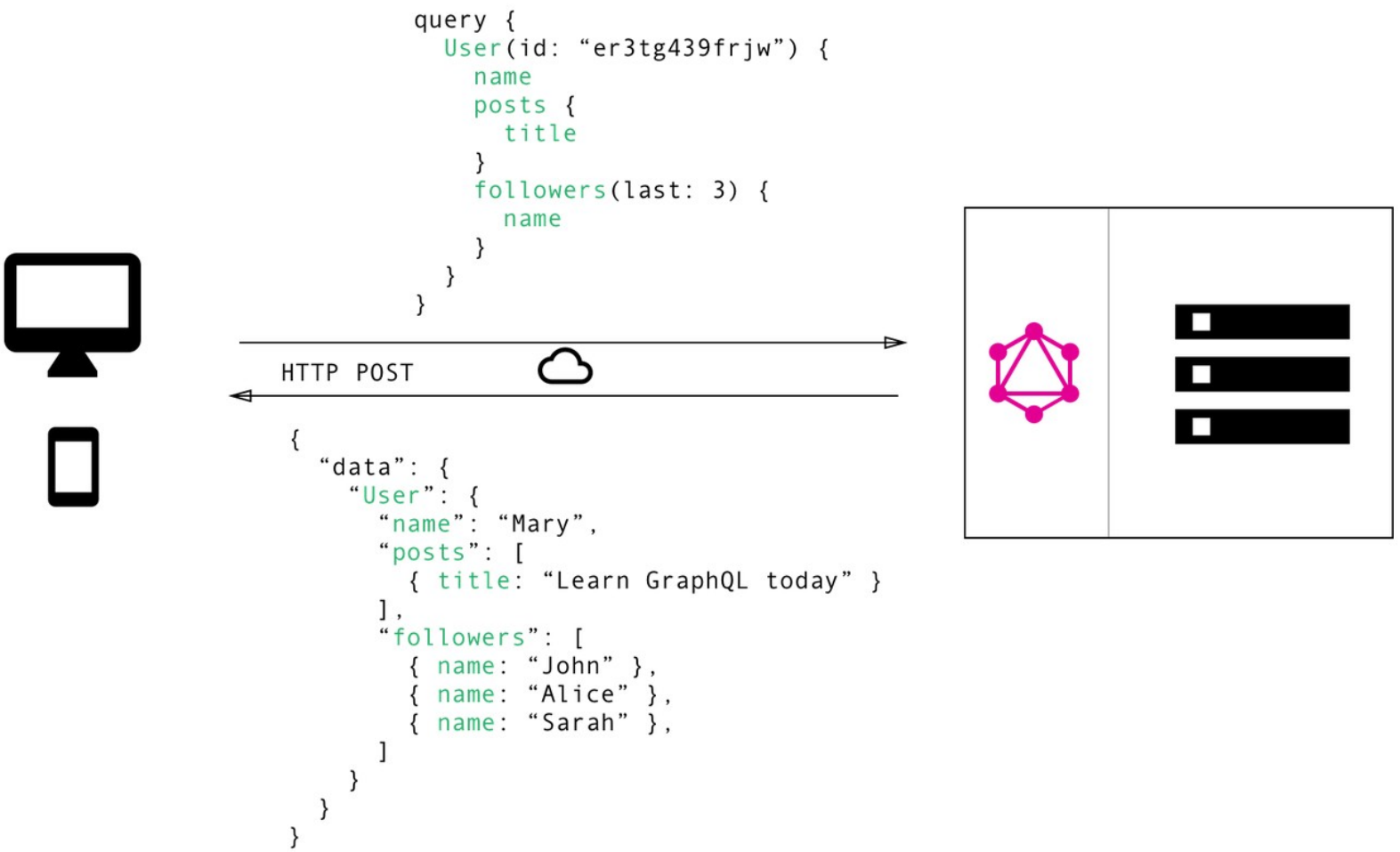
• Orchestration · This gets into workflow territory, something I've been working on a lot recently. Where by "workflow" I mean a service tracking the state of computations that have multiple steps, any one of which can take an arbitrarily long time period, can fail, can need to be retried, and whose behav ior and output affect the choice of subsequent output steps and their behavior.

• Persistent connections · Back a few paragraphs I talked about how MQ message brokers work, main taining a bunch of nailed-up network connections, and pumping bytes back and forth across them. It's not hard to believe that there are lots of scenarios where this is a good fit for the way data and execution want to flow.

• GraphQL: control plane in REST, data plane in other technologies

https://www.tbray.org/ongoing/When/201x/2018/11/18/Post-REST

GraphQL: a Query-API



Using GraphQL, the client can specify exactly the data it needs in a *query*. Notice that the *structure* of the server's response follows precisely the nested structure defined in the query.

Diag from: https://www.howtographql.com/

GraphQL Properties

- No over/underfetching
- Fewer requests
- One endpoint with resolvers
- Data and query syntax identical
- Typed to avoid mistakes
- Federated servers possible

- Danger: huge queries possible (see: https://blog.acolyer.org/2018/05/21/semantics-and-complexity-of-graphql/ for an polynomial time algorithm)

MicroServices

Technology and Eco-System

Forces/Context

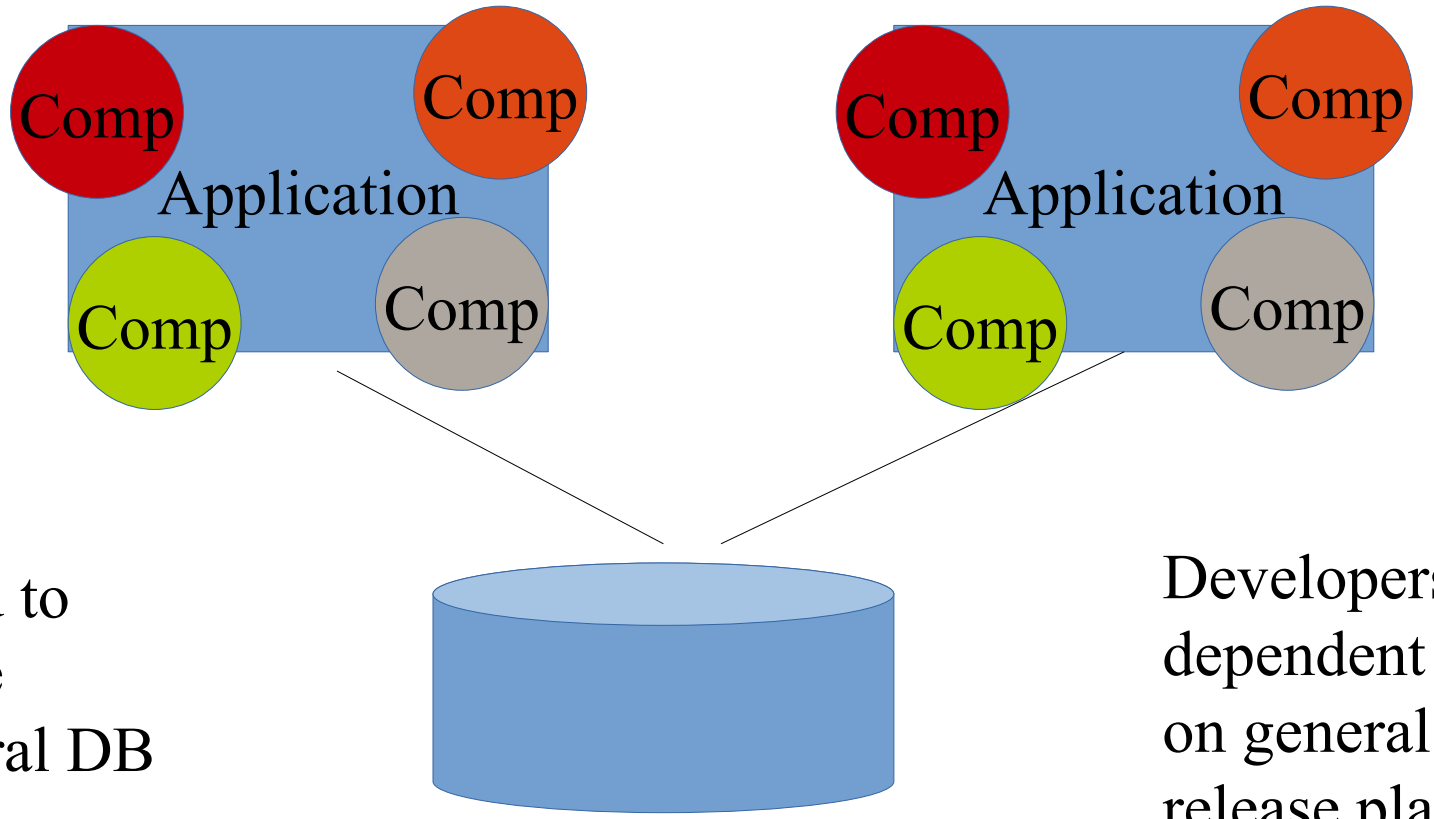
- Ultra large-scale sites require efficient horizontal scaling
- Unicorn companies need to develop new features extremely fast with independent teams
- Unicorn companies need to deploy new features extremely fast (competition, experiments)
- Unicorn companies need to offer an API for network effects

Fat applications running on application servers do not fit into this context!

Scalability Problems of Monolithic Applications

Deployment as a whole only

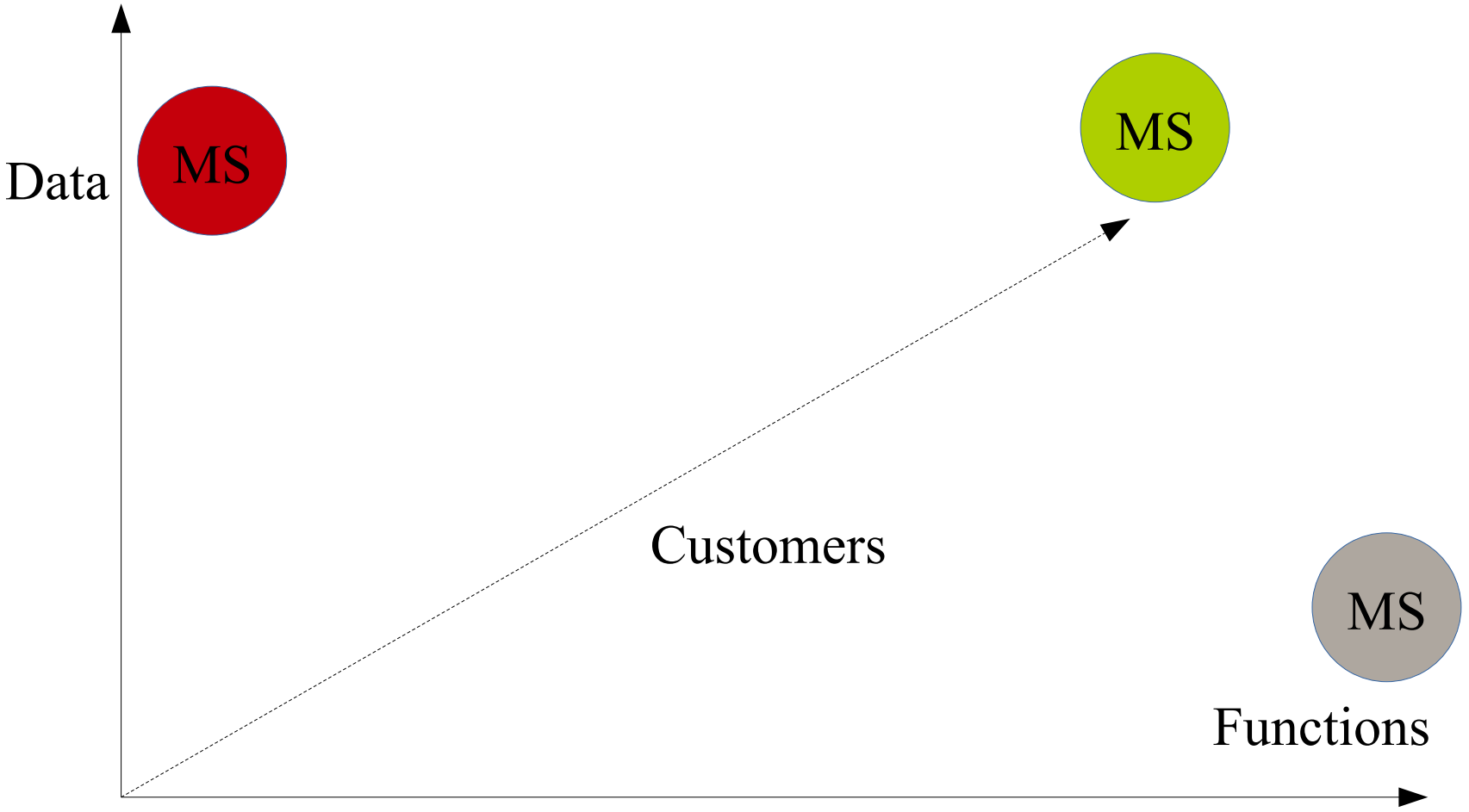
Hard to scale API



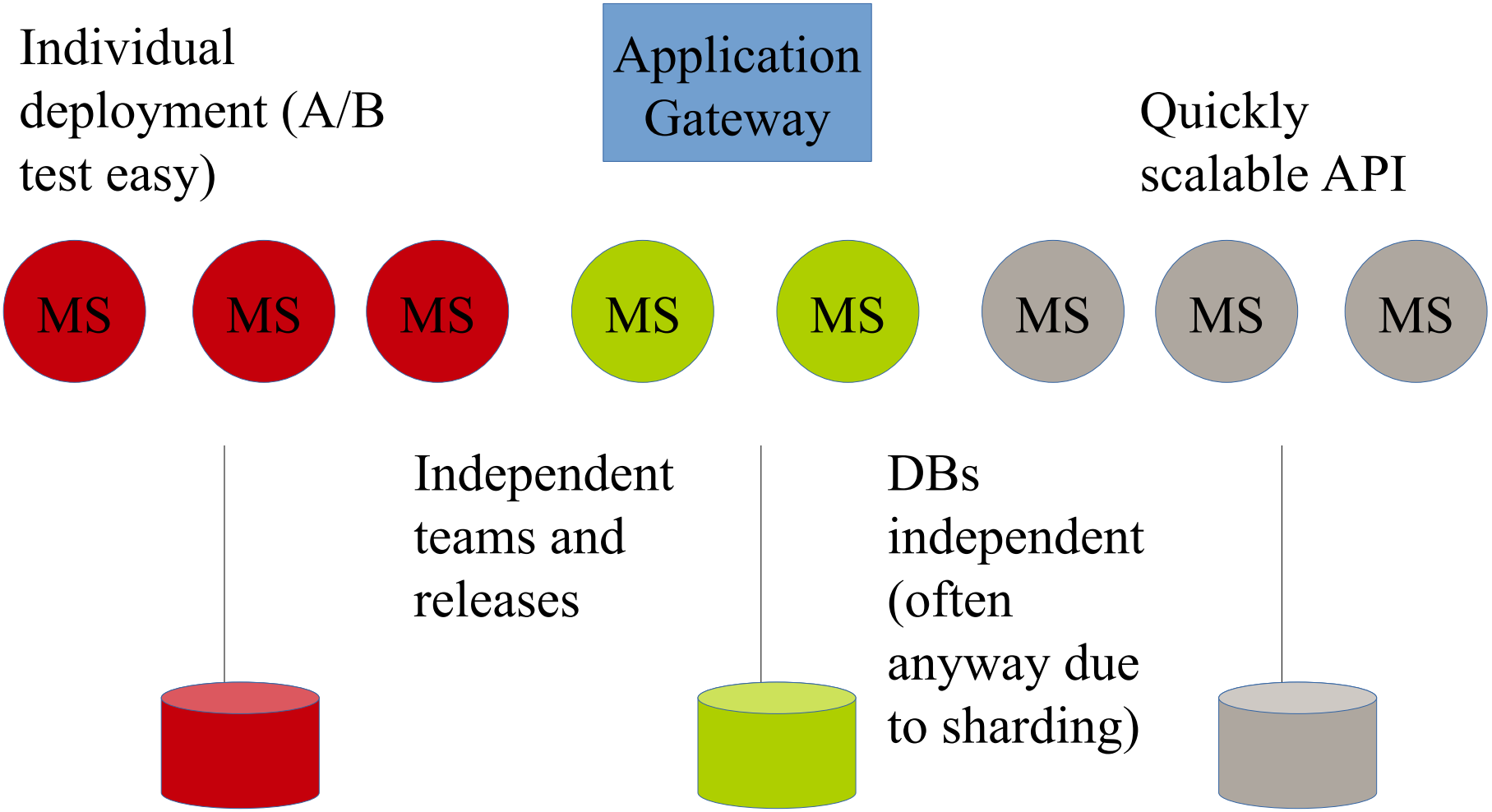
Hard to scale central DB

Developers dependent on general release plan

Solution: Partitioning/De-Composition



MicroServices: Vertically Partitioned Functions



Eco-System: Vertically Partitioned Teams

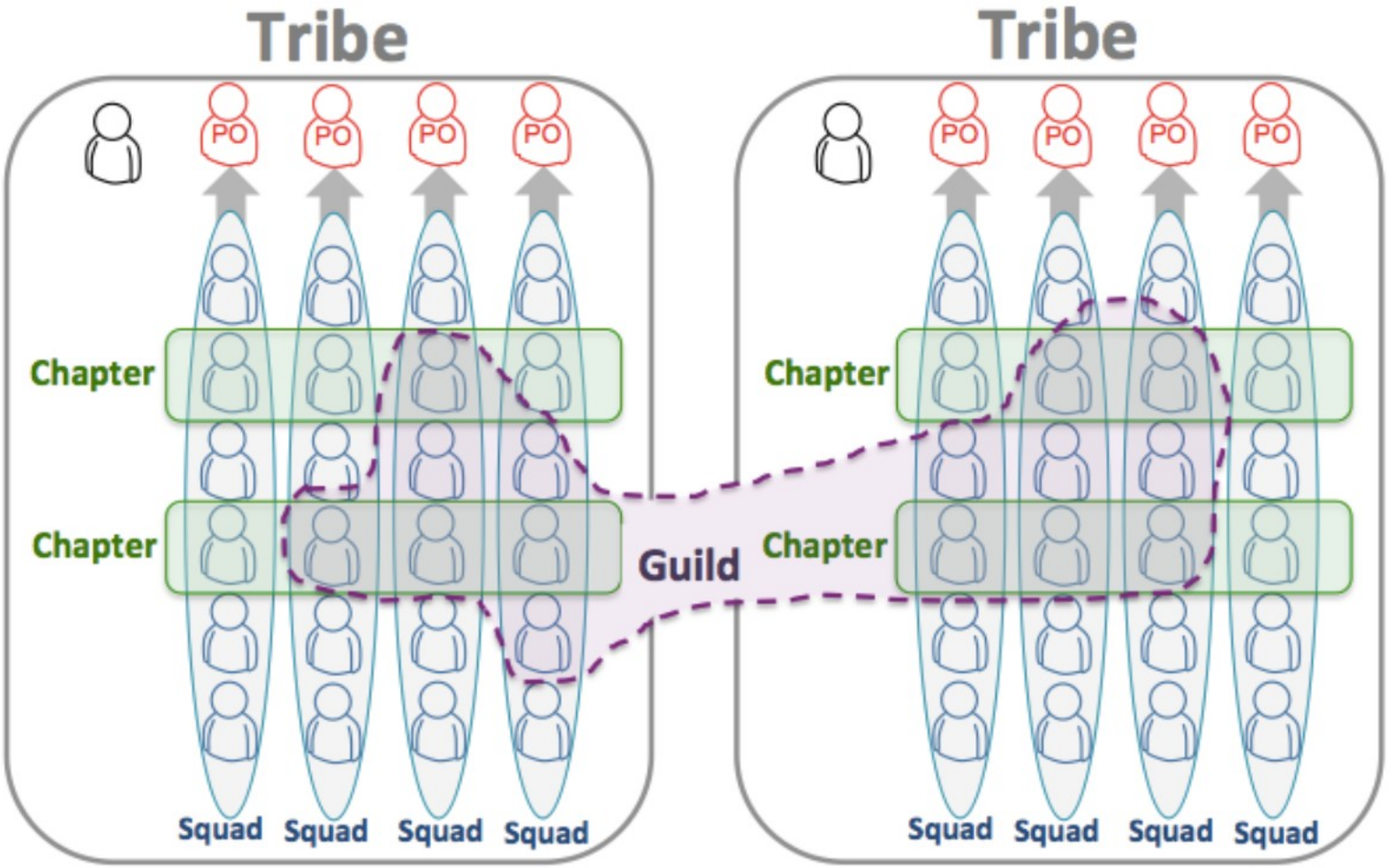
REQ

GUI

BUS

DB

OPS

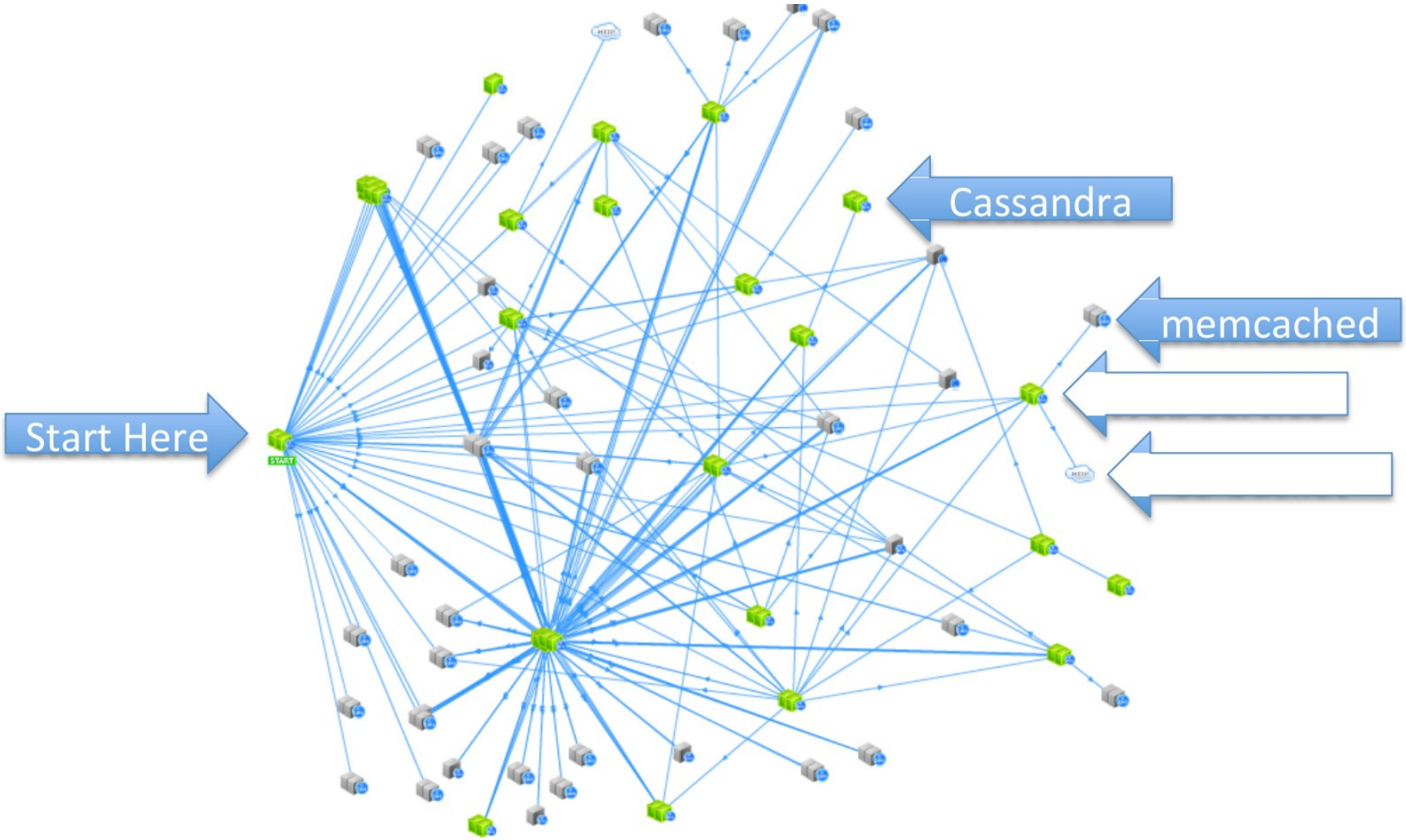


After: Henrik Kniberg & Anders Ivarsson, Scaling Spotify

Eco-System: Technologies and Processes

- Continuous integration/deployment (experiments)
- Fully automated build and deploy
- Free choice of languages
- Continuous monitoring (ELK etc.)
- REST APIs plus RPC tools
- Containers over VM: small MS waste VM instances
- DevOps: teams responsible for operations
- Site-Reliability Engineers (SRE)
- No distributed transactions
- Federated Security
- Fault-tolerance patterns

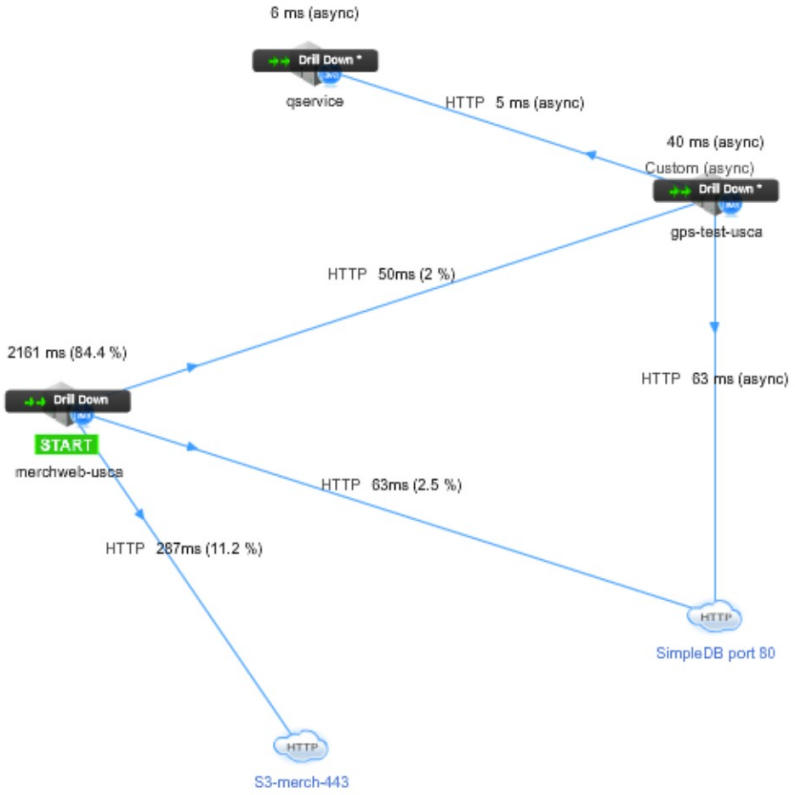
MS Death Star



From:Adrian Cockcroft, Globally Distributed Cloud Applications at Netlix (Uber: 2000 MS in 1.5 years!)

NETFLIX

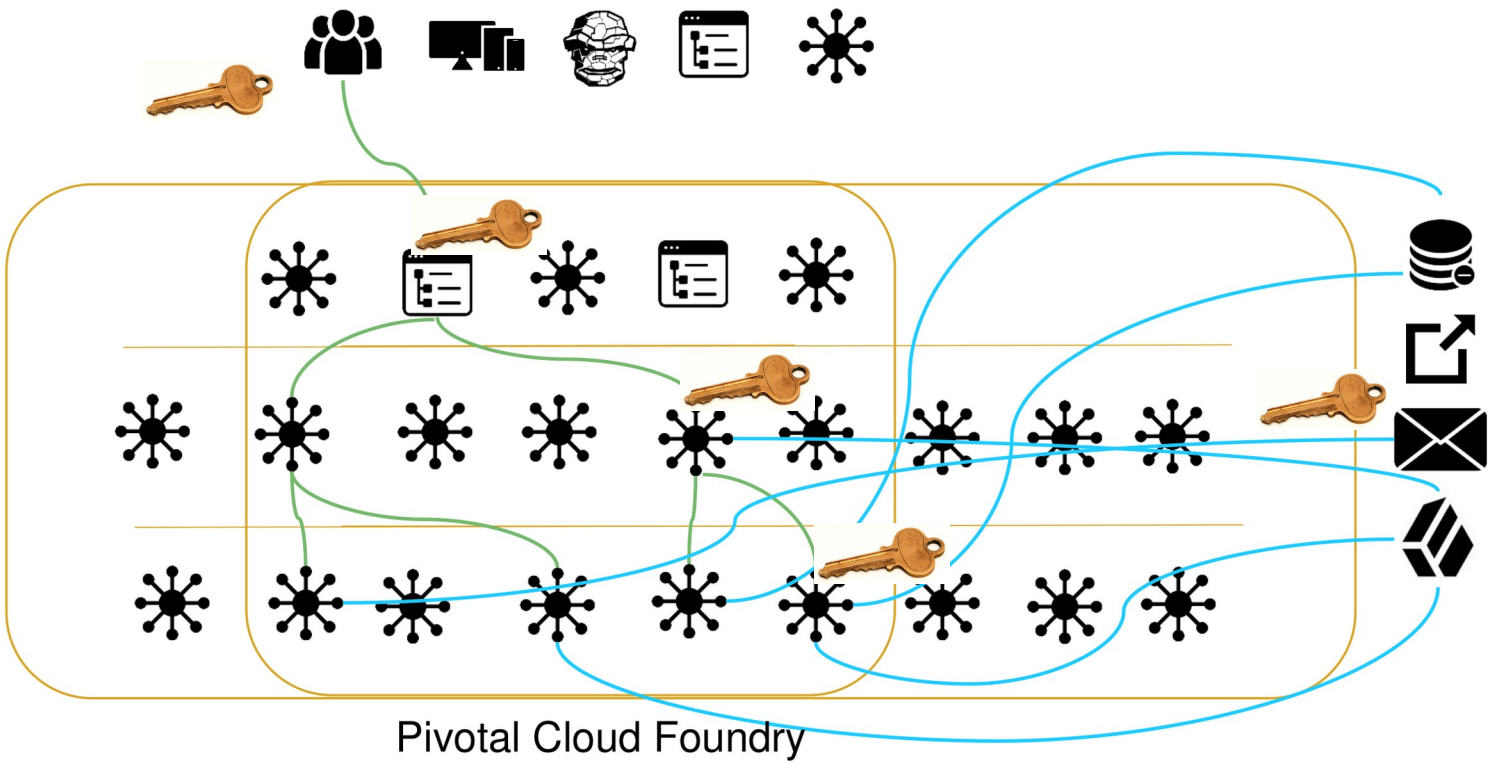
Monitoring/Tracing



NETFLIX

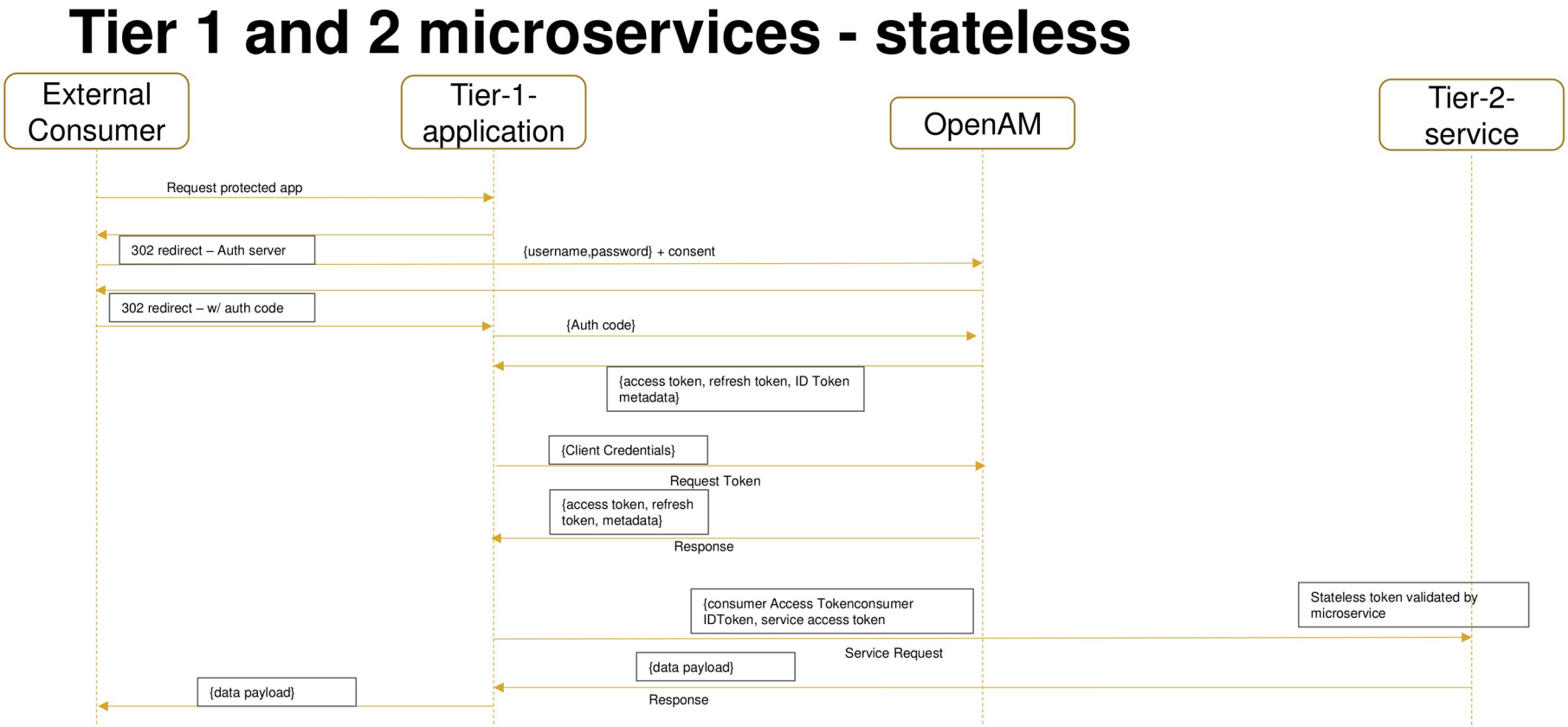
From:Adrian Cockcroft, Globally Distributed Cloud Applications at Netlix. Needs fully automated everything

MS Security: Bearer Tokens



Backend security and secure delegation are needed!

MS Security: OAuth2



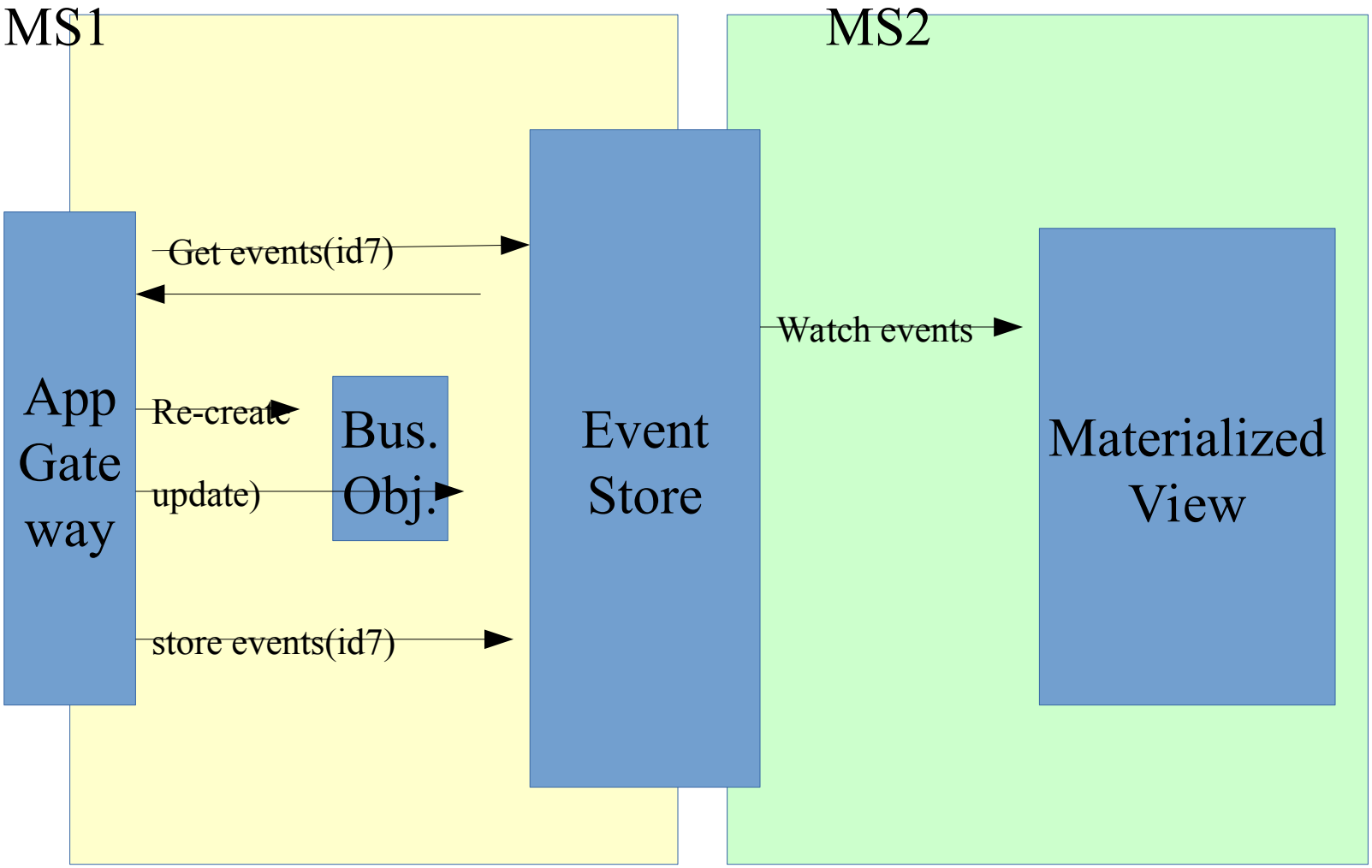
From: D.Ferriera, Authentication and Authorization Architecture for Microservices, Qcon 2016

Main MicroServices Patterns

Microservices architecture: loosely coupled services and teams
API gateway: Facade to fine-granular services
Client-side discovery: provided by MS chassis (e.g. spring boot)
Server-side discovery
Service registry:services register themselves during startup
Self registration: by chassis
Service instance per Container: scales better than VM per service
Serverless deployment: see nanoservices below
Database per Service: no touching other MS database..
Event-driven architecture: programming without a stack...
Event sourcing: record change events in event store
CQRS: separate update and idempotent reads (eventual...)
Transaction log tailing:follow transaction log for changes
Database triggers: put events in events table after changes
Application events

After: C.Richardson,
http://microservices.io/patterns/

Event Sourcing/CQRS



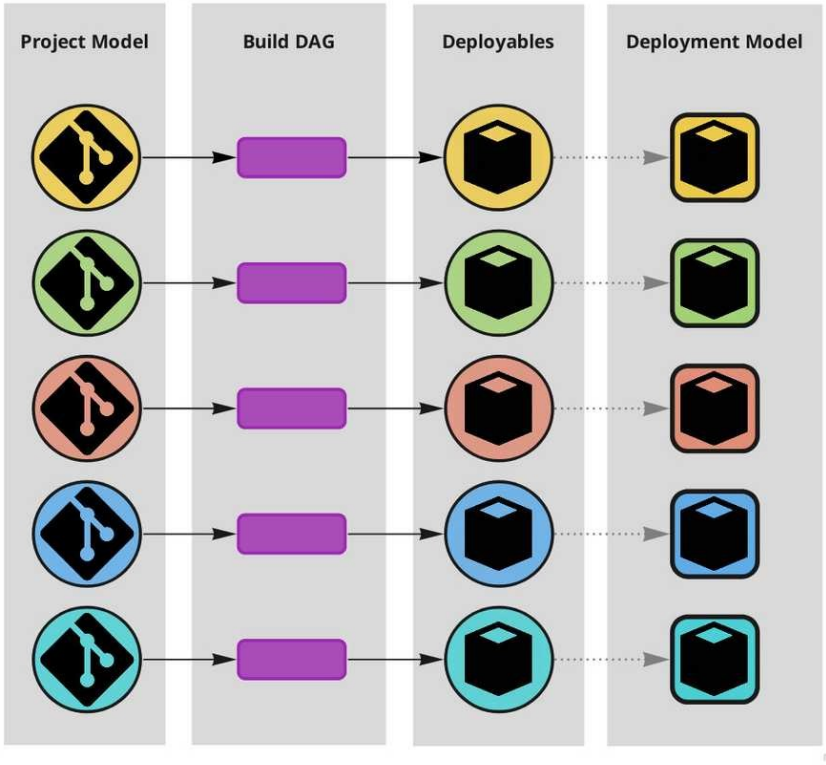
After: Building and deploying microservices with event sourcing, CQRS and Docker (QCONSF 2014) from Chris Richardson

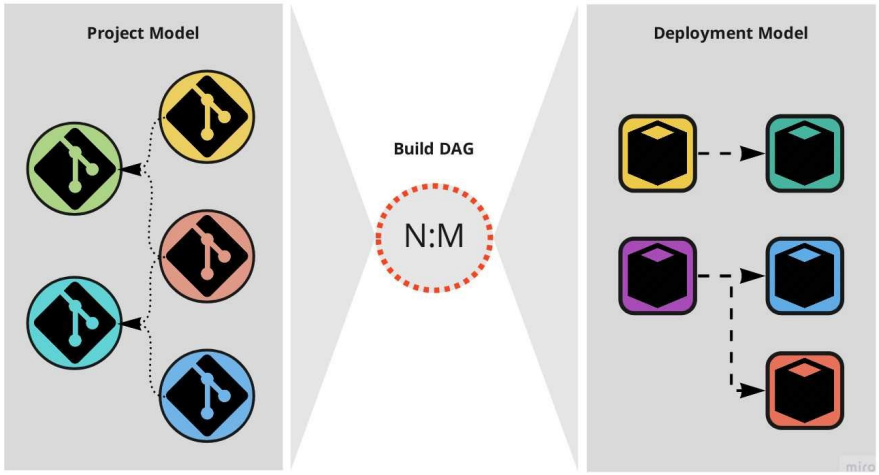
Caveats

There’s no such thing as a microservices architecture, there is software architecture. The size and granularity of services are a dimension of the solution, not a given of the problem. To dictate it from the start is reductive and only shifts complexity elsewhere

https://vlfig.me/posts/microservices

Project-Build-Deployment Models





Define dependencies on several layers. Compile time dependencies are safer than runtime dependencies.

https://vlfig.me/posts/microservices. Also: https://randalldavis.github.io/microservice/testing/2017/06/05/microservice-edges.html

Eventual Consistent Data

Scenario:
1. User created
2. Shopping Cart created

------ User does not yet have the cart ----------
What should clients do in that case? Repeat request?
Which error status should be used?

---

3. Shopping Cart added in User

After: Building and deploying microservices with event sourcing, CQRS and Docker (QCONSF 2014) from Chris Richardson

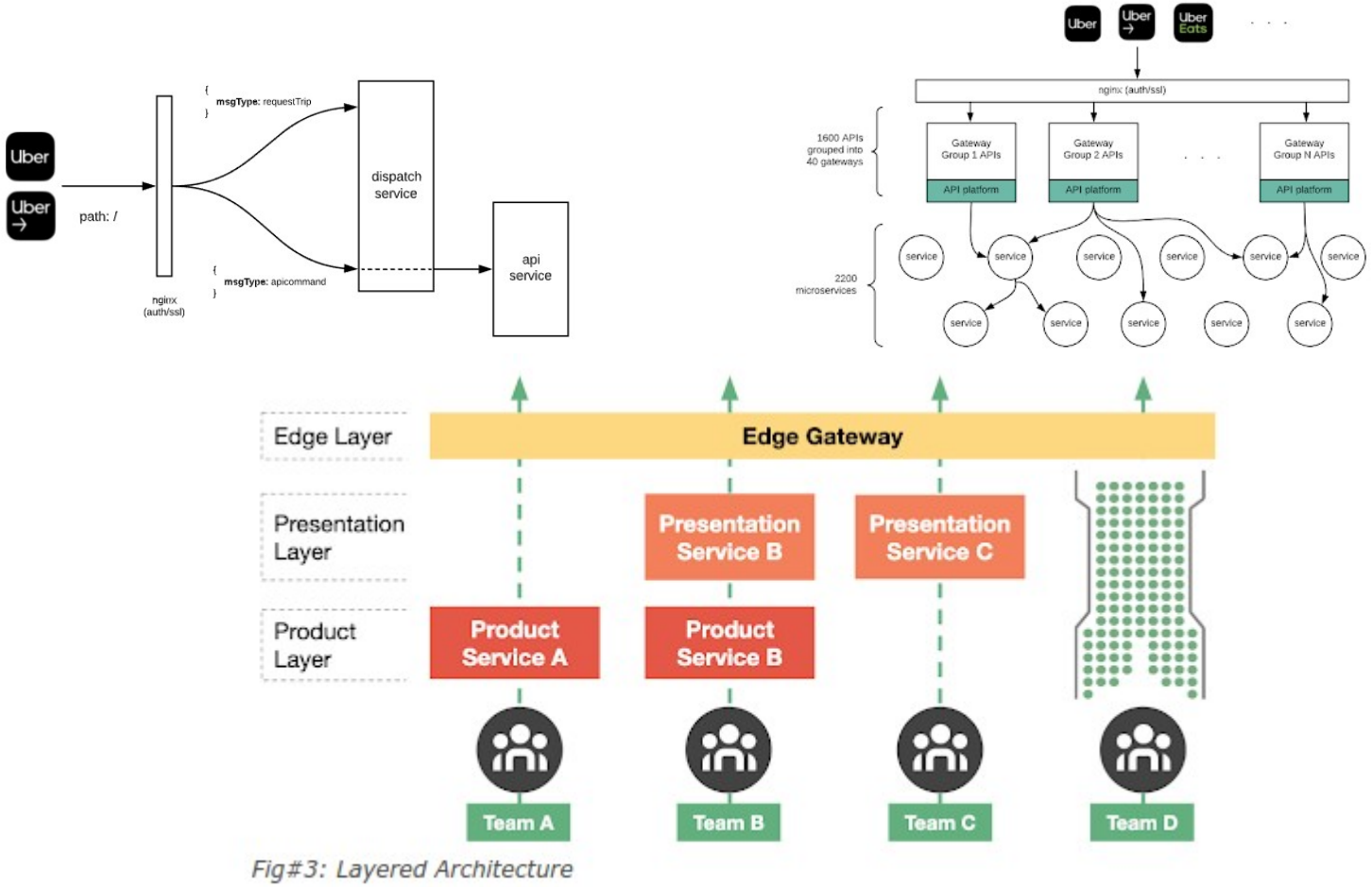
Critical Points

- Cross-Concerns: Transactions, Security, Performance, Scalability: SRE's to the rescue?
- VMs too big for small services with low throughput
- Maintenance of large numbers of services (Netflix death star, Uber:2000 MS in 1.5 years)
- Monitoring complex due to large number of independent services (correlation?)
- Different languages and technologies used
- Danger of new central bottlenecks (eventstore?)
- REST often not the best API model and transport
- Distributed commits are "eventual consistent" and not allowed to get lost!

Photo Site Example

https://mariadb.com/resources/blog/how-create-microservices-and-set-microservice-architecture-mariadb-docker-and-go

Refactoring MS-Architectures



Fig#3: Layered Architecture

Backend-for-frontend, reducing round trip times, avoiding coupling, better languages and protocols, gRPC, domain services etc.
https://eng.uber.com/gatewayuberapi/

Serverless Computing

Serverless Computing (Aka FaaS, NanoService etc.)

Serverless will fundamentally change how we build business around technology and how you code; Containers are important but ultimately invisible subsystems and this is not where you should be focused.

Todd Hoff, highscalability.com

What Is Serverless?

A cloud-native platform for short-running, stateless computation
And event-driven applications which scales up and down instantly and automatically and charges for actual usage at a millisecond granularity (Stephen Fink, IBM defines serverless at @ServerlessConf London, 28th Oktober 2016)

- Decoupling of computation and storage; they scale separately and are priced independently.
- The abstraction of executing a piece of code instead of allocating resources on which to execute that code.
- Paying for the code execution instead of paying for resources you have allocated to executing the code.

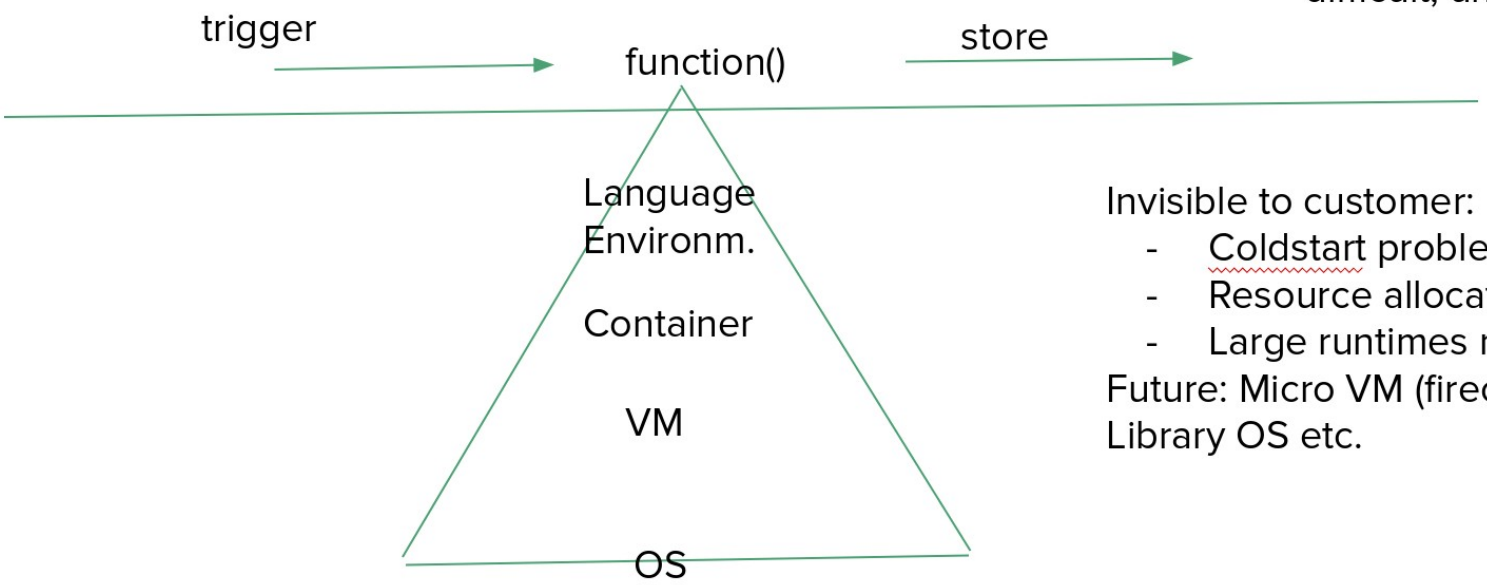
(Cloud Programming Simplified: A Berkeley View on Serverless Computing, Eric Jonas et.al.)

Stateless

- Where is config information? Dummy class file in classpath?
- No memory in function
- “mostly stateless Java VM” (static class initializers…)
- needs stuff in persistent storage (like s3): change mgt.
- hypercomposition is hard to do

Serverless Platform

Customer visible: RAM/CPU paid, autoscaling, debugging difficult, unreliable latency

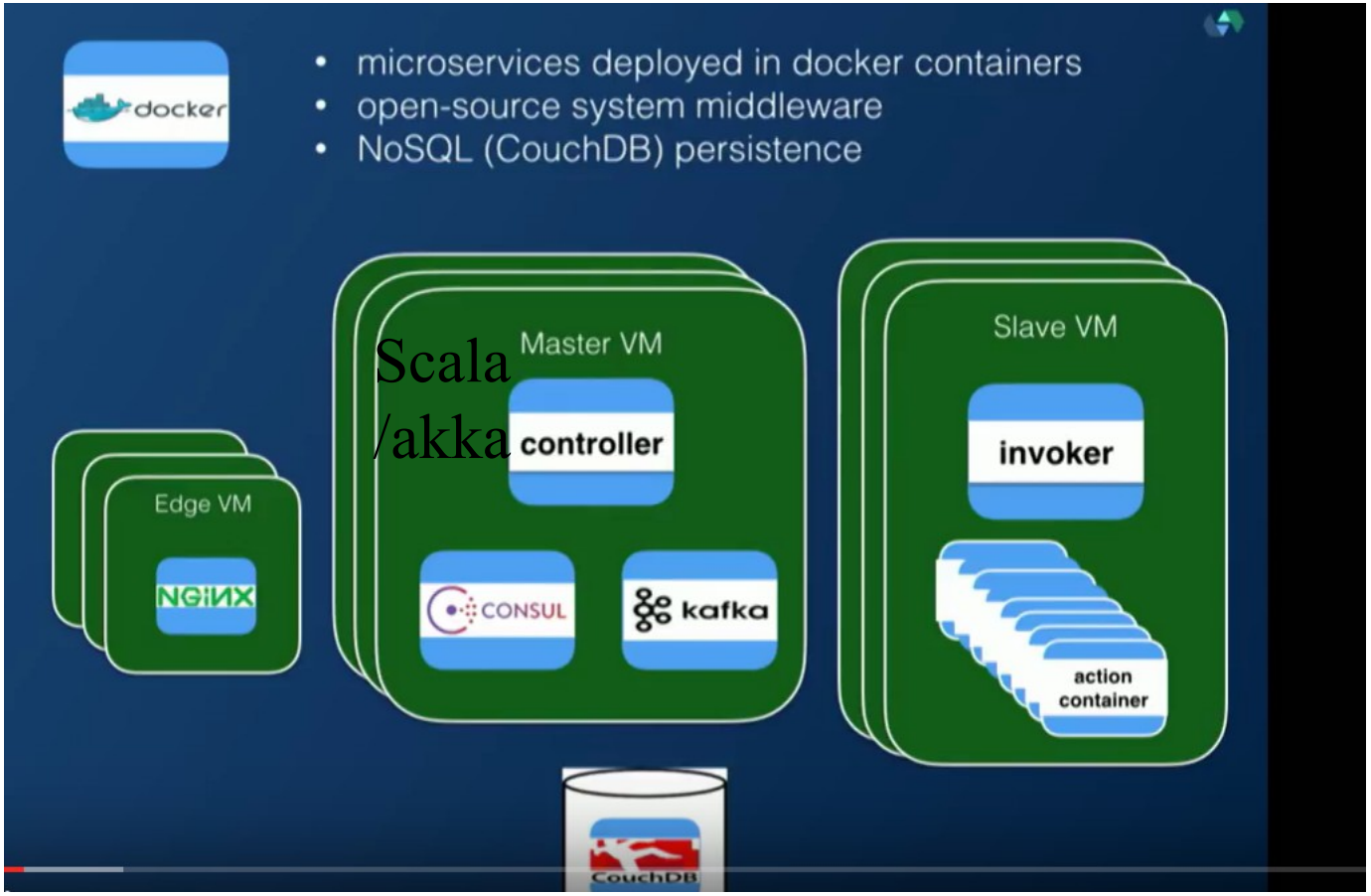


Invisible to customer:

- Coldstart problems
- Resource allocation
- Large runtimes needed

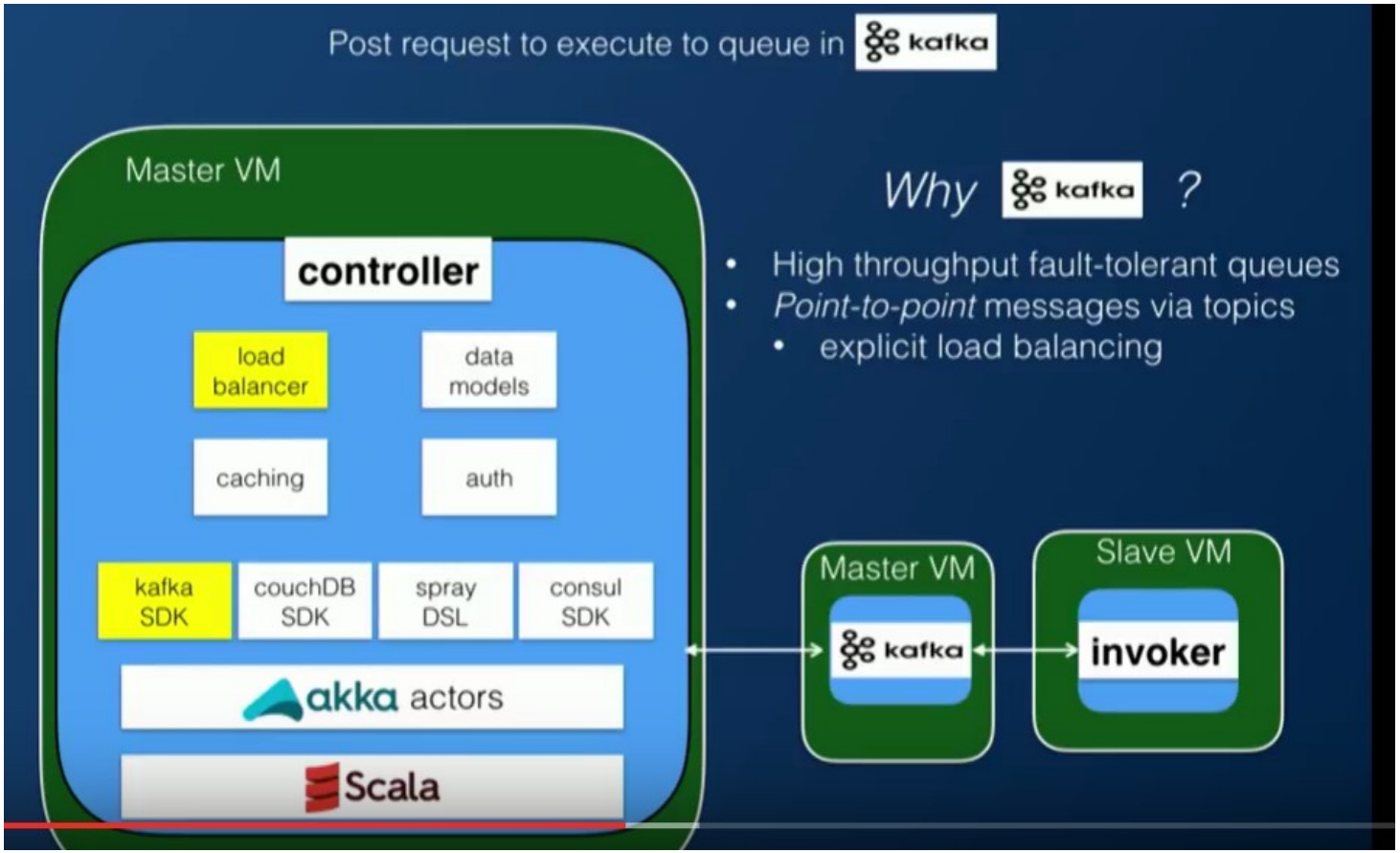
Future: Micro VM (firecracker) Library OS etc.

OpenWhisk Runtime



Stephen Fink, OpenWhisk talk at @ServerlessConf London, 28th Oktober 2016

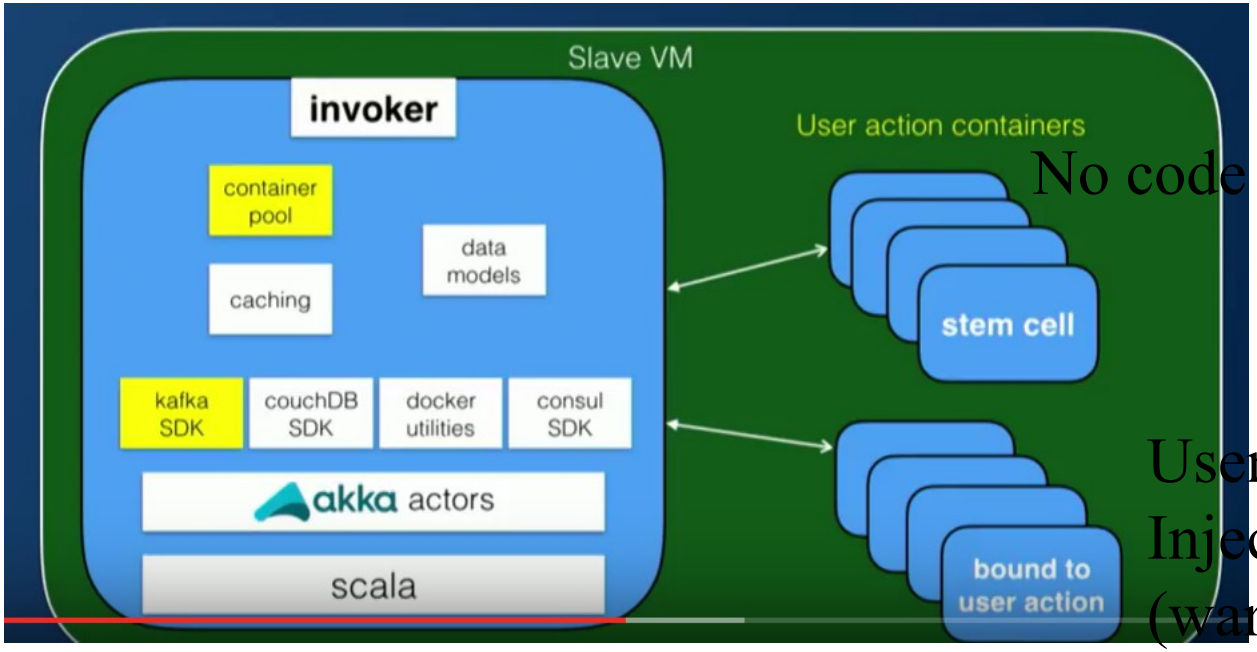
OpenWhisk Controller



Stephen Fink, OpenWhisk talk at @ServerlessConf London, 28th Oktober 2016

OpenWhisk Invoker

Container manager

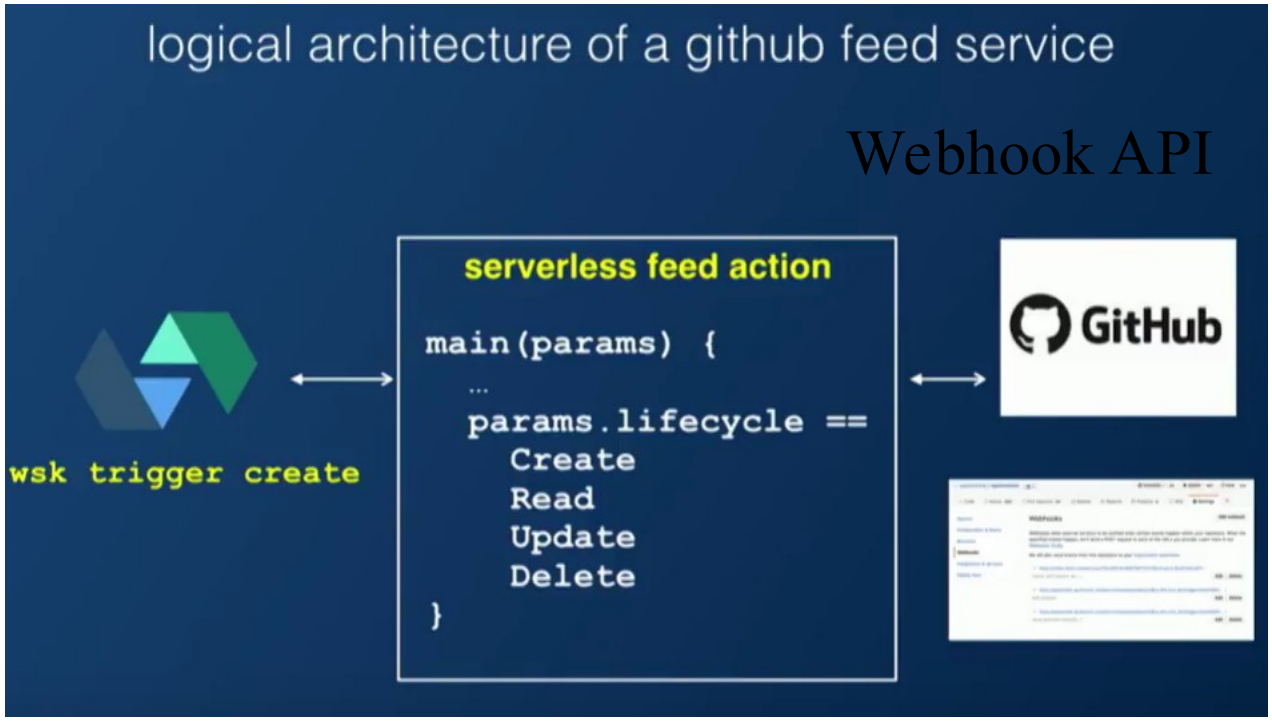


No code yet

User code Injected (warm)

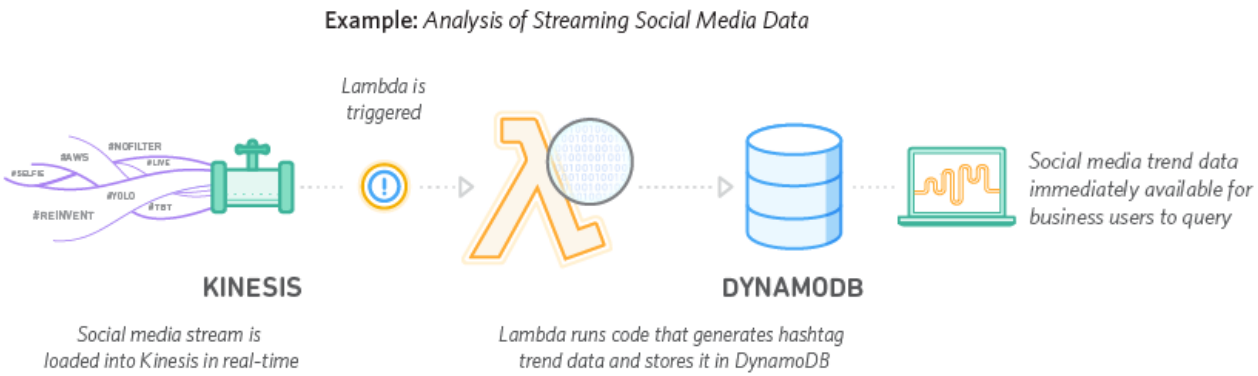
Stephen Fink, OpenWhisk talk at @ServerlessConf London, 28th Oktober 2016

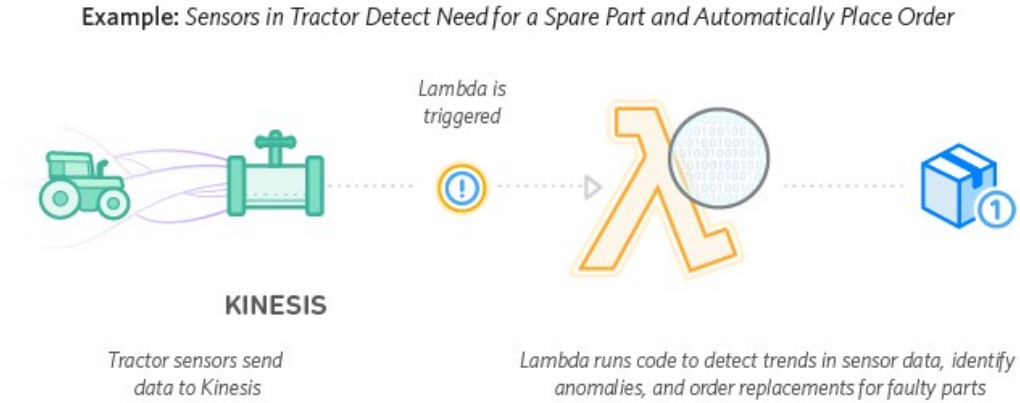
OpenWhisk Serverless Feed Action



Stephen Fink, OpenWhisk talk at @ServerlessConf London, 28th Oktober 2016

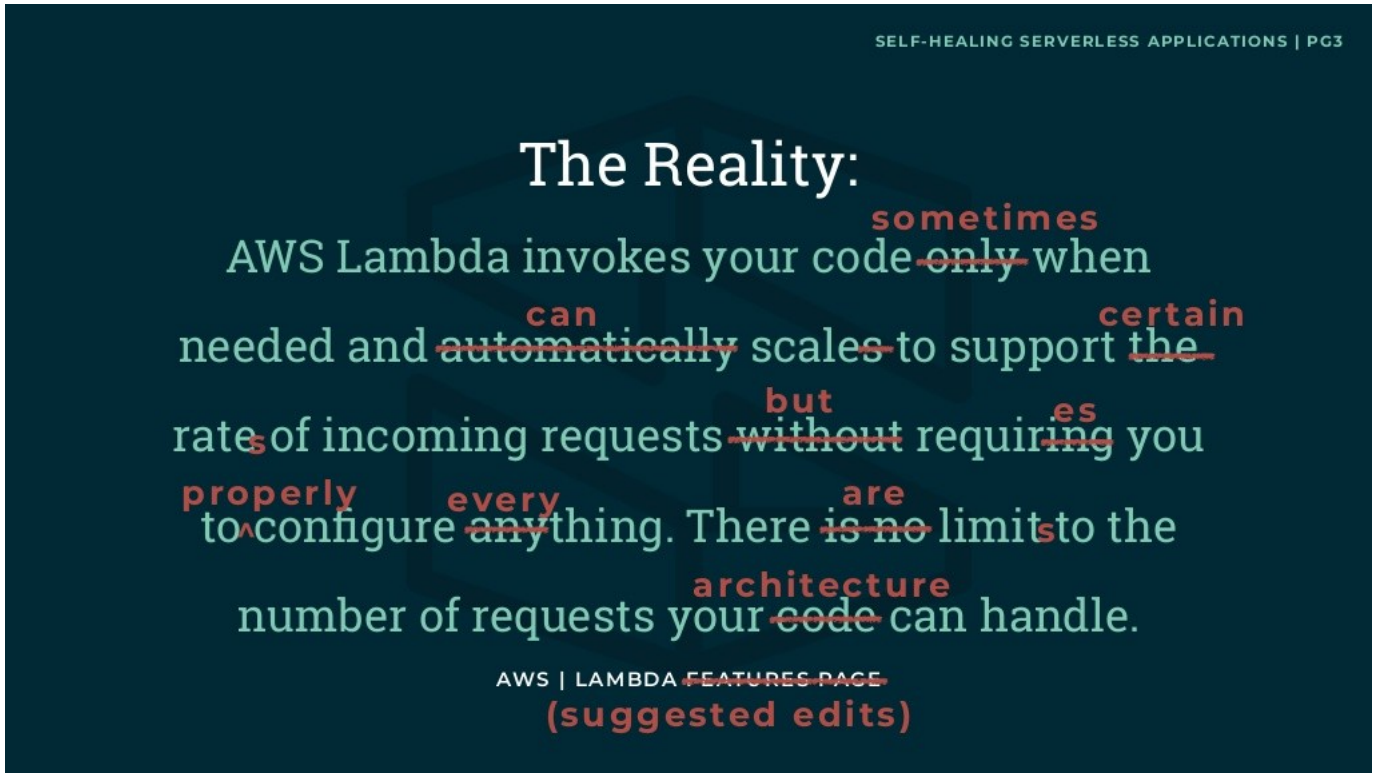
Amazon AWS Examples





https://www.youtube.com/watch?v=eOBq__h4OJ4&feature=youtu.be

The Serverless Bubble?



https://www.jeremydaly.com/takeaways-from-serverlessnyc-2018/

Serverless Issues

- countless small IAM rules
- coupling with less scaleable components
- coldstart
- Unclear bug handling (dlq, rquest order, wrapper)
- Stateful functions
- Limits everywhere
- New testing concepts needed
- high costs when waiting for something

https://www.jeremydaly.com/takeaways-from-serverlessnyc-2018/

The State of Serverless

Application	*Description*	*Challenges*	*Workarounds*	*Cost-performance*
Real-time video compression (ExCamera)	On-the-fly video encoding	Object store too slow to support fine-grained communication; functions too coarse grained for tasks.	Function-to-function communication to avoid object store; a function executes more than one task.	60x faster, 6x cheaper versus VM instances.
MapReduce	Big data processing (Sort 100TB)	Shuffle doesn't scale due to object stores latency and IOPS limits	Small storage with low-latency, high IOPS to speed-up shuffle.	Sorted 100 TB 1% faster than VM instances, costs 15% more.
Linear algebra (Numpy-wren)	Large scale linear algebra	Need large problem size to overcome storage (S3) latency, hard to implement efficient broadcast.	Storage with low-latency high-throughput to handle smaller problem sizes.	Up to 3x slower completion time. 1.26x to 2.5x lower in CPU resource consumption.
ML pipelines (Cirrus)	ML training at scale	Lack of fast storage to implement parameter server; hard to implement efficient broadcast, aggregation.	Storage with low-latency, high IOPS to implement parameter server.	3x-5x faster than VM instances, up to 7x higher total cost.
Databases (Serverless SQLite)	Primary state for applications (OLTP)	Lack of shared memory, object store has high latency, lack of support for inbound connectivity.	Shared file system can work if write needs are low.	3x higher cost per transaction than published TPC-C benchmarks. Reads scale to match but writes do not.

- Unreliable latency
- Functions not directly addressable
- Poor support for standard com. Patterns (e.g. batching)
- 1 CPU/function
- Limited lifetime
- Debugging/tracing/monitoring hard/impossible (dead letter box)
- Autoscaling dangers (cost, time)
- Storage systems not good for small objects/calls (expensive/slow)
- No fine-grained coordination
- See the Berkeley Serverless Report (Literature)

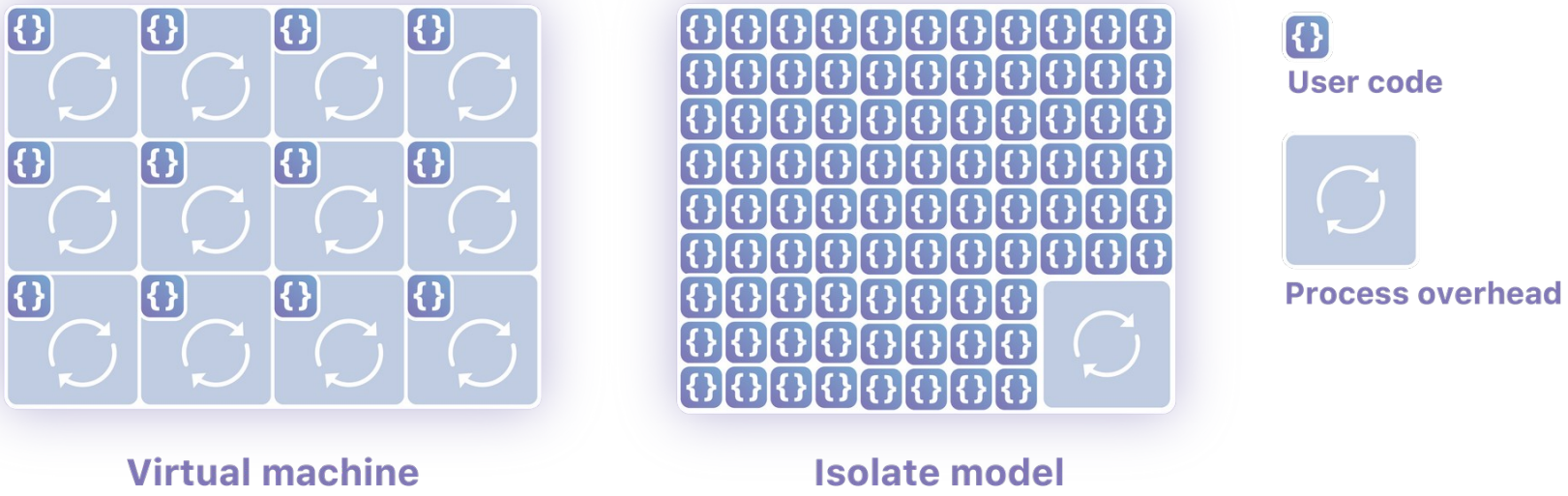
Serverless Microservice Patterns

- Scalable Webhook
- Gatekeeper
- Internal API
- Internal Handoff
- Aggregator
- Notifier
- FIFOer
- Streamer
- Router
- State Machine
- etc.

Watch out for prices, latencies, failure handling (DLQ)

https://www.jeremydaly.com/serverless-microservice-patterns-for-aws/

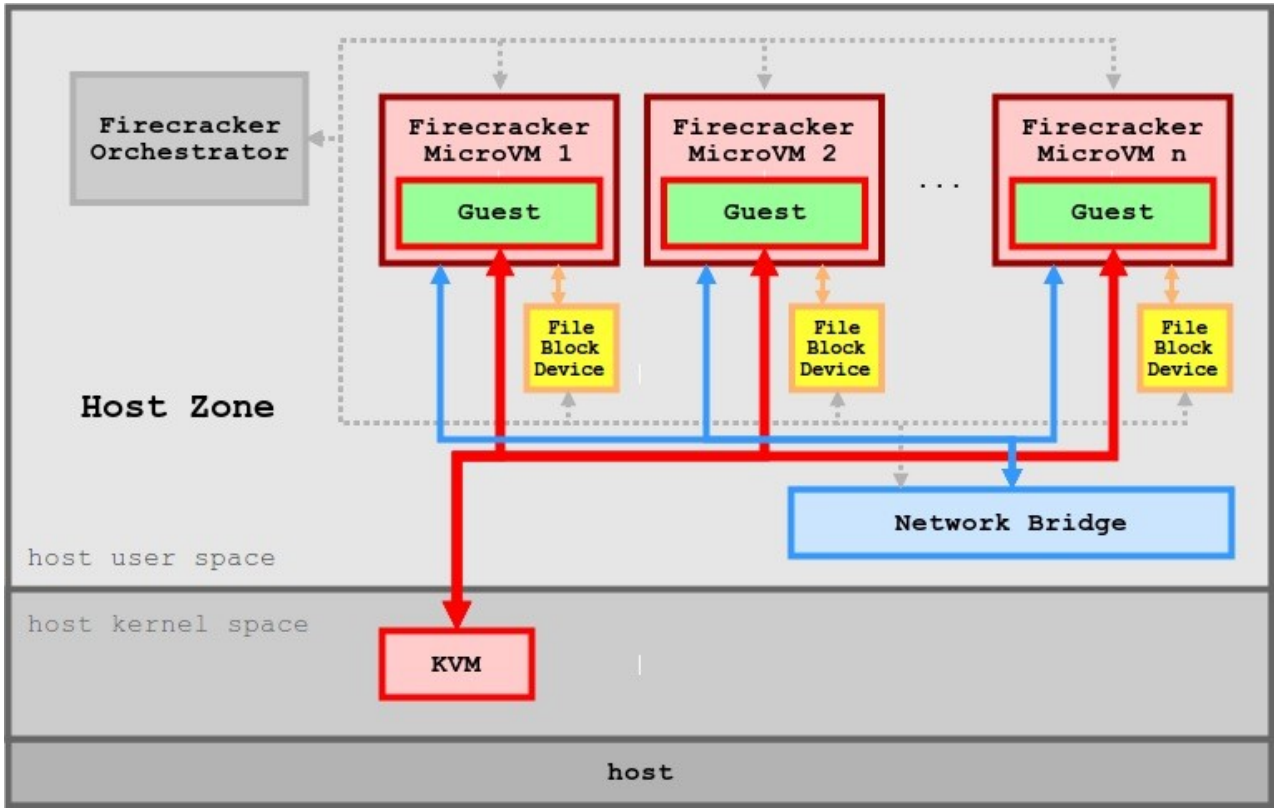
Even smaller: Isolates



This goes in the same direction as library operating systems. The isolation concept is by controlling references in the compile process (object capabilities? Like Singularity?). Multi-tenancy!!

https://blog.cloudflare.com/cloud-computing-without-containers/

New MicroVM for Functions



Size and startup time are essential for serverless computing. Firecracker, a new – Rust-based – MicroVM (fork of CromiumOs VM).

Diag. By David Bryant, https://www.infoq.com/news/2018/12/aws-firecracker

A Function Market?

The future of software development will be lots of lambda functions consumed from a marketplace, stitched together with some new capability. Waste will be reduced, bias (i.e. custom building something which is already well crafted) will start to disappear and we get all that other good "financial development" stuff the last post covered. Hurrah!

So when I look at my trading system, then as time goes on then not only will more and more of the components be provided by the AWS marketplace but if AWS is playing an ILC game then many will become industrialised components provided by AWS itself. The marketplace will just be future potential AWS components and on top of this, all the novel exciting stuff (which is directly giving early warning to AWS through consumption data) is just future market components. I've shown an example of this in the map below.

The benefits to consumers i.e. those trying to build stuff will be overwhelming. Amazon will continue to accelerate in efficiency, customer focus and apparent innovation despite the occasional gnashing of teeth as they chew up bits of the software industry. Have no doubt, you can use this model to chew up the entire software industry (or the duplicated mess of bias which calls itself a software industry) and push people to providing either components sold through the marketplace or building actually novel stuff.

Now most executives especially in the software industry will react just as they did with cloud in 2006/07 by trotting out the usual layers of inertia to this idea. It'll never happen! This is not how software works! It's a relationship business! Security! Prior investment! Our business model is successful!

Simon Wardley, blog.gardeviance.org/2016/11/amazon-is-eating-software-which-is.html.

Resources (1)

xml-dev: mailing list for XML developers. High traffic site. Had a good discussion on XML-RPC performance lately

- Security for Web services, Raghavan Srinivas, http://www.sun.com/developers/evangcentral/totallytech/Warsaw/SecurityWarsaw.pdf

Resources (2)

- Programming Web Services with SOAP, J.Snell et.al., O´Reilly 2002.
- www.oasis-open.org , Portal for ebXML and other XML schema definitions. Work on business transactions over web-services.
- Global XML Web Services Architecture, Microsoft paper October 2001, www.gotdotnet.com (.net portal for web services)
- Michael Stal, Web Services im Überblick, Objectspectrum 7/8 2001
- www.uddi.org, portal for UDDI.

Resources (3)

- The IBM UDDI registry: http://www.ibm.com/services/uddi
- Microsoft's UDDI registry: http://uddi.microsoft.com
- Andre Tost, UDDI4J lets Java do the walking. Good introduction to the concepts behind UDDI
- Steve Vinoski, Web Services and Dynamic Discovery, Article on webservices.org about the real difficulties with ontologies and automatic understanding. Yes, Steve is one of the fathers of CORBA and IONA´s chief architect.
- P.J.Murray, Web Services and CORBA, CapeClear. Good explanation of the mapping problems when exposing CORBA services via Web Services.

Resources (4)

- Dave Winer et.al., A busy developers guide to SOAP1.1, from www.soapware.org, bare bone explanation of the most important features. Does not cover SOAP with attachements etc.
- Web Services for Remote Portals (WSRP), http://www.oasis-open.org/committees/wsrp/ , a new approach to re-use services WITH their GUI. Headed by Thomas Schaeck, IBM Böblingen
- the RESTwiki on http://conveyor.com/RESTwiki/moin.cgi
- Principled Design of a modern Web Architecture, R. Fielding, http://www.cs.virginia.edu/~cs650/assignments/papers/p407-fielding.pdf
- Alex Rodriguez , RESTful Web services: The basics, IBM , 06 Nov 2008 http://www.ibm.com/developerworks/webservices/library/ws-restful/index.html?S_TACT=105AGX54&S_CMP=B1113&ca=dnw-945

Resources (5)

- James McCarthy, Reap the benefits of document style Web services http://www-106.ibm.com/developerworks/library/ws-docstyle.html?n-ws-6202 . A nice explanation of document style web services and when to use them. E.g. if there is NO pre-existing rpc-service you might be better of designing your communication in document style right away. Better for asynchronous processing as well. And coarse grained which is better in many cases of dist-sys as we have learned.
- The WS-Resource Framework V1.0, Czaikowski, Ferguson et.al. describes the addition of statful resources to web services by using meta-data and identifiers. Read the grid papers to understand the need for it.
- Security for Grid Services, Von Welch et.al. Describes the security needs of virtual organizations.
- Martin Brown, Building a grid using Web Services standards Part1-6. www.ibm.com/developerworks Shows a distributed movie serving application built with web services. Looks a bit like napsters design. Shows how similar p2p and grid applications really are.
- REST in Rails: http://www.b-simple.de/documents

SOA Resources (1)

- Olaf Zimmermann et.al., Elements of Service-oriented Analysis and Design, 6/2004, www.ibm.com/developerworks
- Ali Arsanjani, Service-oriented modeling and architecture, 11/2004 www.ibm.com/developerworks
- Guido Laures et.al., SOA auf dem Prüfstand, ObjectSpektrum 01/2005. Covers the new benchmark by The Middleware Company for SOA implementations
- David Booth et.al, Web Services Architecture – W3C Working Group Note 2/2004. A very good introduction which explains the architectural models nicely. Covers messaging, resources, SOA and policies. Lots of interesting links to additional standards.

SOA Resources (2)

- WS-Policies, Standard for matching and merging of service policies. Note that the standard is conservative and does not require advanced merging when basic types differ
- Christoph Diefenthal, Automatic composition of business processes between companies - using semantic technologies and SOA. (Thesis work at the HDM and Fraunhofer IAO). Excellent work showing web intermediates integrating business services automatically.
- http://www.oreillynet.com/pub/wlg/3017, Tim O‘Reilly on what makes open source different and empowering. Very good.
- http://www.onjava.com/pub/a/onjava/2005/01/26/soa-intro.html shows how JINI‘s dynamic service lookup and call features are offered by SOA in a language independent way

REST Resources

- Martin Fowler, Richardson Maturity Model
- Martin Fowler, Enterprise Integration using Rest
- Richardson, Leonard, and Sam Ruby. RESTful Web Services. Sebastopol: O'Reilly Media, Inc., 2007.
- Mark Masse, REST API Design Book, O'Reilly Media
-

Serverless Computing Resources

- Swardley on Why the fuss about serverless? https://medium.com/@swardley/why-the-fuss-about-serverless-4370b1596da0
- Videos from ServerlessConf, London: https://www.youtube.com/channel/UCqlcVgk8SkUmve4Kw4xSlgw
- Old Programmers and New Programmers Can Learn New Tricks - Donald Ferguson https://youtu.be/vWyeS_aAZmo
- Amazon AWS Lambda: https://www.youtube.com/watch?v=eOBq__h4OJ4&feature=youtu.be
- Guide to serverless technologies, https://thenewstack.io/ebooks/serverless/guide-to-serverless-technologies

Serverless Computing Resources

- Swardley on Why the fuss about serverless? https://medium.com/@swardley/why-the-fuss-about-serverless-4370b1596da0
- Videos from ServerlessConf, London: https://www.youtube.com/channel/UCqlcVgk8SkUmve4Kw4xSlgw
- Old Programmers and New Programmers Can Learn New Tricks - Donald Ferguson https://youtu.be/vWyeS_aAZmo
- Amazon AWS Lambda: https://www.youtube.com/watch?v=eOBq__h4OJ4&feature=youtu.be

MicroServices Resources

- Life Beyond Distributed Transactions, http://www.ics.uci.edu/~cs223/papers/cidr07p15.pdf
- MicroServices Patterns: microservices.io (Chris Richardson)
- Graham Lea, Distributed Transactions: The Icebergs of Microservices, Posted on August 30, 2016, http://www.grahamlea.com/2016/08/distributed-transactions-microservices-icebergs/
-