# WEB AS DISTRIBUTED COMPUTING PLATFORM

Jonathan R. Engelsma, Ph.D.

## TOPICS

- Overview
- Service Oriented Architectures
- W3C Web Services
- RESTful Web APIs
- Contrasting W3C / REST

As the Internet becomes increasingly pervasive (e.g. what doesn't have a network interface these days?), what is the best way to build distributed client/server software systems?

## IMPLEMENT ONE-OFF SOLUTIONS

- Custom software for each system/application.
  - Highly optimized solutions
  - Labor intensive / Costly
  - Nightmare to manage.
  - Poor support for future evolution

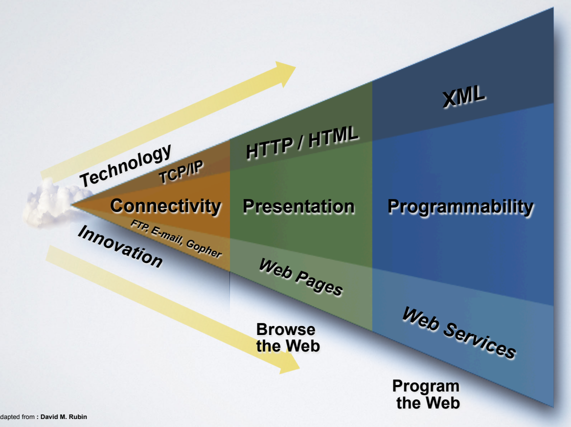
## USE TRADITIONAL MIDDLEWARE SOLUTIONS

- RPC, CORBA, DCOM, Java RMI, etc.
- Often vendor centric, and not very interoperable
- Don't play well across the public Internet
- Can have scaling issues.

## WEB AS PROGRAMMABLE PLATFORM

- Use the web as a programmable platform.
- Vendor / Platform agnostic
- Plays well with existing web infrastructure
- Can support massive scaling… if done right.

## THE EVOLUTION OF THE WEB

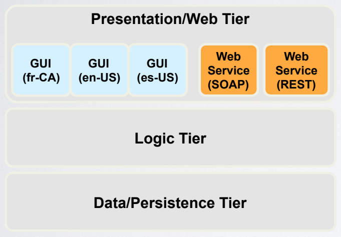


Adapted from : David M. Rubin

## TWO SCHOOL OF THOUGHT

- Approaches to "Web as programmable platform"
  - W3C Web Services
    - aka "big" web services or "SOAP" web services.
  - RESTful Web Services
    - aka "web APIs".

## THREE TIER ARCHITECTURES



- A web service is any web application whose output is meant to be consumed by a program instead of a human.
- Web services are just presentations, and so live in the presentation tier.
- Should expose the full power of the web to programs as well as to humans.

## SERVICE ORIENTED ARCHITECTURES (SOA)

- provides a set of principles of governing concepts used during phases of systems development and integration.
- packages functionality as interoperable services: software modules provided as a service can be integrated or used by several organizations, even if their respective client systems are substantially different.
- requires loose coupling of services with operating systems, and other technologies that underlie applications.

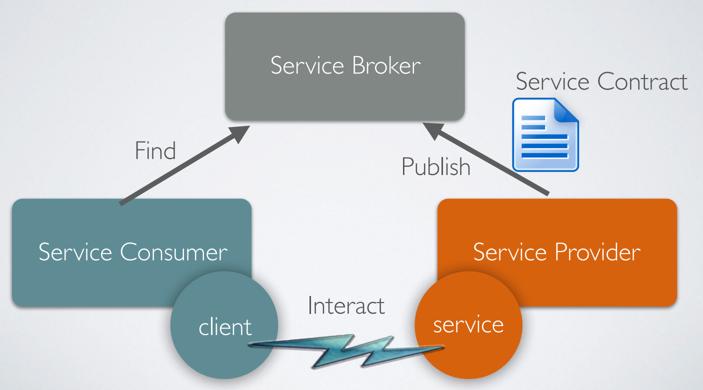
## INVOCATION DYNAMICS

- real time: when an immediate response is required.
    - e.g. In banking example, user expects an immediate response.
- deferrable: when an immediate response is not required.
    - e.g., in a flight reservation system, frequent flyer info needs to be posted to appropriate system before next statement is issued.

## TYPICAL CHARACTERISTICS OF A SOA

- Requester independence (built as a "black box")
- Has a verifiable identity and a defined set of interfaces.
- Versioned
- Accessible via some sort of service directory
- Invoked by requesters and can invoke other services.
- Support error recovery mechanisms.

## ABSTRACT ARCHITECTURE



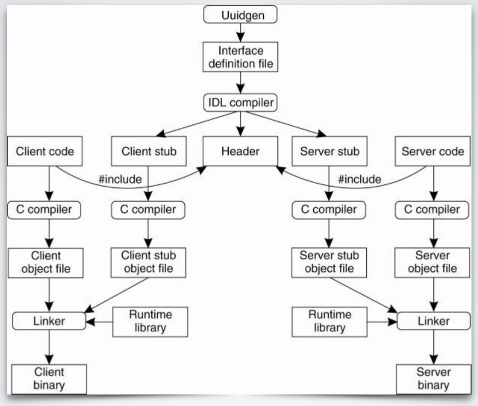
## W3C WEB SERVICES

- W3C def: "A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."
- http://www.w3.org/TR/ws-arch/

## HISTORICAL BACKGROUND

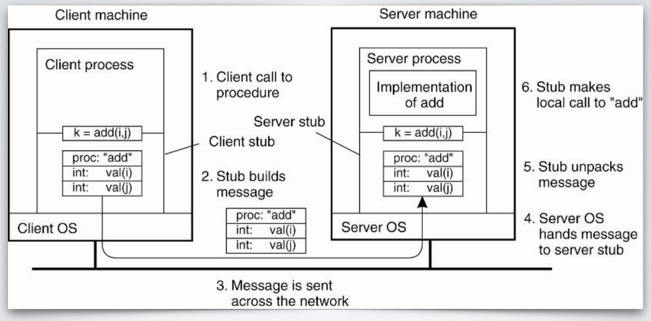
- Successor to traditional distributed object middleware (XML-RPC, CORBA, DCOM, Java RMI). Huge improvement!!
- Unlike CORBA and DCOM messaging, layering SOAP on HTTP makes it play well with proxies and firewalls. XML messages are another big win.
- Created by Microsoft and DevelopMentor in 1999, when they submitted their SOAP spec to the W3C for standardization.
- Allied standards include UDDI for discovering web services and WSDL for describing web service APIs, and many more.

## RPC PROGRAMMING MODEL



Source: Distributed Systems, Tanenbaum & Van Steen

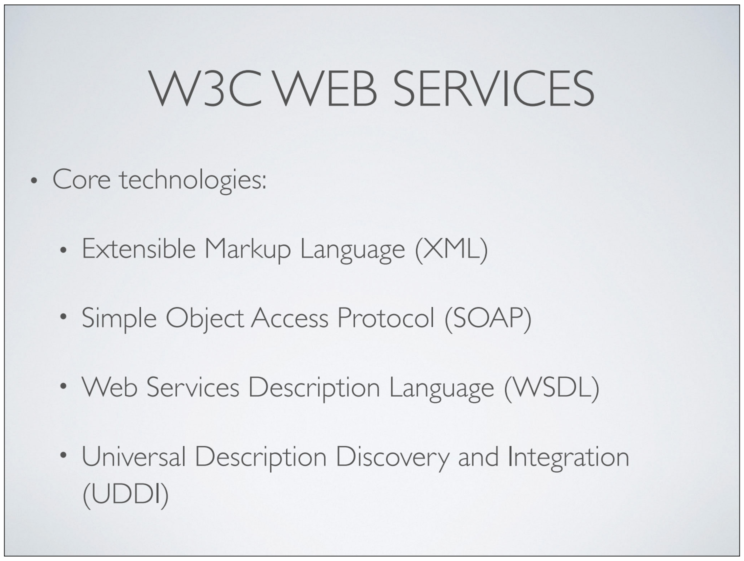
## RPC PROGRAMMING MODEL



Source: Distributed Systems, Tanenbaum & Van Steen

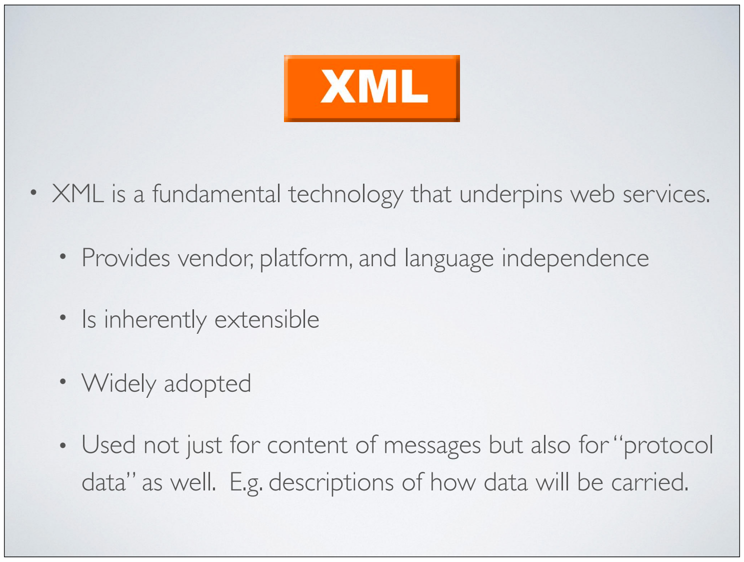
## WHAT DO W3C WEB SERVICES PROVIDE?

- message structure and infrastructure
- service description mechanism
- message routing
- message exchange patterns (choreography, workflow)
- service discovery



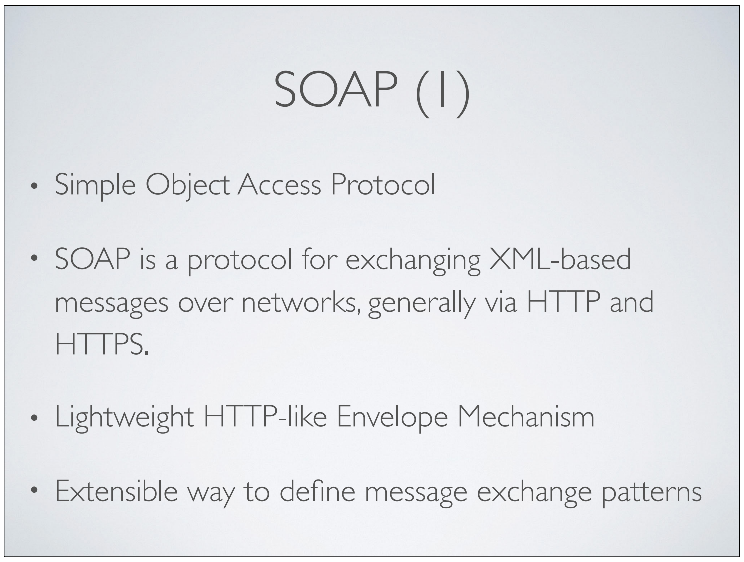
## W3C WEB SERVICES

- Core technologies:
  - Extensible Markup Language (XML)
  - Simple Object Access Protocol (SOAP)
  - Web Services Description Language (WSDL)
  - Universal Description Discovery and Integration (UDDI)



## XML

- XML is a fundamental technology that underpins web services.
  - Provides vendor, platform, and language independence
  - Is inherently extensible
  - Widely adopted
  - Used not just for content of messages but also for "protocol data" as well. E.g. descriptions of how data will be carried.



## SOAP (1)

- Simple Object Access Protocol
- SOAP is a protocol for exchanging XML-based messages over networks, generally via HTTP and HTTPS.
- Lightweight HTTP-like Envelope Mechanism
- Extensible way to define message exchange patterns

## SOAP (2)

- Fault System
- Bindings to support transport protocols (HTTP, SMTP, etc).
- In practice, only HTTP is supported.

## SOAP VS. HTTP

HTTP Example

```
POST /orders HTTP/1.1
Host: themobilemontage.com
Content-Type: application/vnd.jre+xml
Content-Length: 1002

<order xmlns="http://..." ..../>
```

SOAP Example

```
<soap:Envelope xmlns:soap="http://...">
    <soap:Headers>
        <wsa:To xmlns:wsa="http://...">http://themobilemontage.com/orders</wsa:To>
    </soap>:Headers>
    <soap:Body>
        <order xmlns="http://..." ..../>
    </soap:Body>
</soap:Envelope>
```

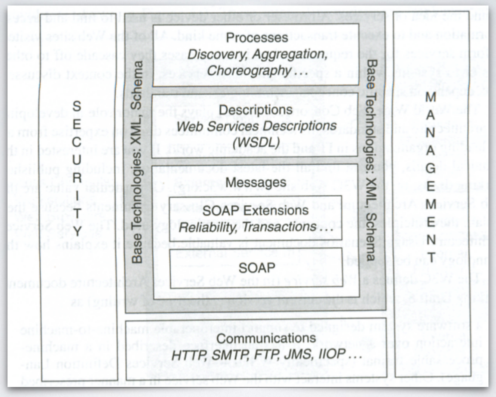
## WSDL

- Web Services Description Language
- Pronounced as "whiz dull"
- XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information.
- Abstract description that is concretely bound a network protocol and message format.
- http://www.w3.org/TR/wsdl

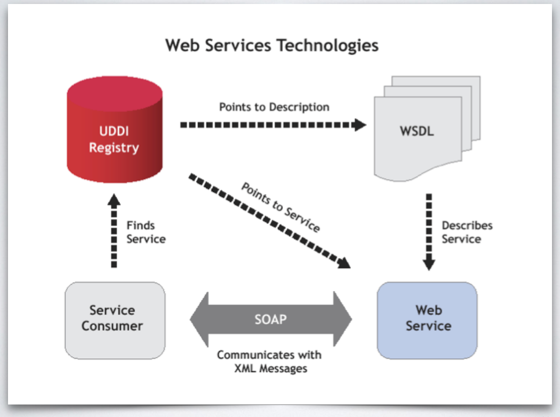
## UDDI

- Universal description, discovery and integration
- registry system
- The UDDI Business Registry is intended to serve as a global, all-inclusive listing of businesses and their services.
- private repositories possible
- UDDI began as ad hoc consortium; now housed at OASIS.

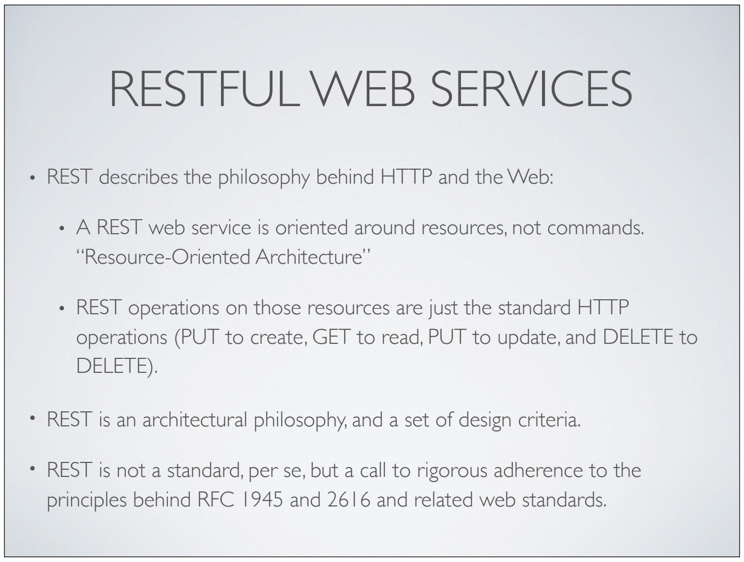
## W3C WS ARCH


Processes
Discovery, Aggregation, Choreography . . .
Descriptions
Web Services Descriptions (WSDL)
Messages
SOAP Extensions
Reliability, Transactions . . .
SOAP
Communications
HTTP, SMTP, FTP, JMS, IIOP . . .
SECURITY
MANAGEMENT
Base Technologies: XML, Schema
Base Technologies: XML, Schema

## PUTTING IT ALTOGETHER

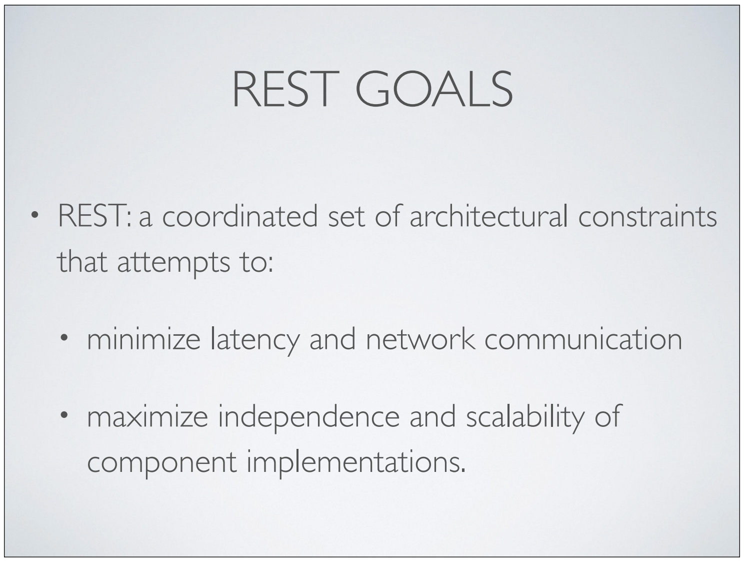


Adapted from : **David M. Rubin**



## RESTFUL WEB SERVICES

- REST describes the philosophy behind HTTP and the Web:
  - A REST web service is oriented around resources, not commands. "Resource-Oriented Architecture"
  - REST operations on those resources are just the standard HTTP operations (PUT to create, GET to read, PUT to update, and DELETE to DELETE).
- REST is an architectural philosophy, and a set of design criteria.
- REST is not a standard, per se, but a call to rigorous adherence to the principles behind RFC 1945 and 2616 and related web standards.



## REST GOALS

- REST: a coordinated set of architectural constraints that attempts to:
  - minimize latency and network communication
  - maximize independence and scalability of component implementations.

## HISTORICAL BACKGROUND

- REST: **Re**presentational **S**tate **T**ransfer
- The term "REST" was first introduced in Roy Fielding's PhD dissertation at USC.
- Fielding is a co-editor of IETF's HTTP standard (RFC 1945, RFC 2616)

**REST's key insight:** the human web is extremely powerful, and this power should be fully given to programs.

?

Just what is it that makes the web so powerful to humans? How did it improve pre-web information systems, or even non-digital conventional ways to organize info?

## WEB REQUIREMENTS (1)

- Low Barrier to Entry to users, authors, developers (participation is voluntary)
- Extensibility: assume change will happen.
- Distributed Hypermedia - presence of app control info embedded within the presentation of info.

## WEB REQUIREMENTS (2)

- Internet Scale
  - "anarchic" scalability
  - independent deployment - multiple organization, old versions coexist with new, etc.

## REST: HYBRID ARCHITECTURE

- R. T. Fielding and R. N. Taylor

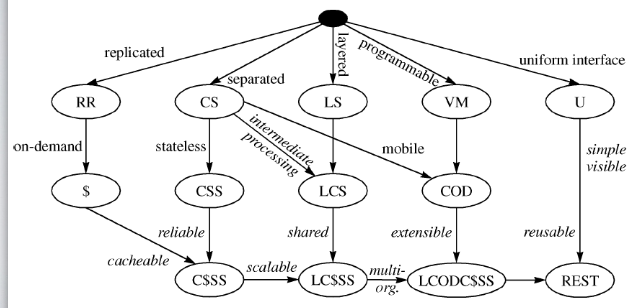
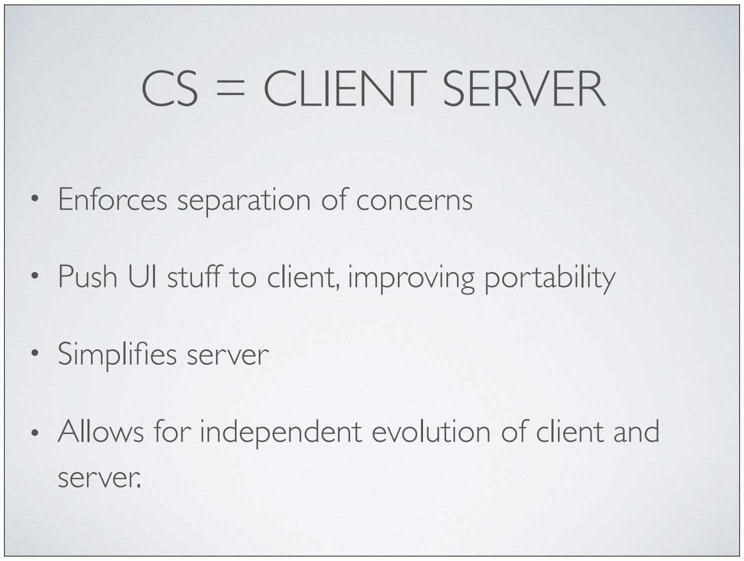
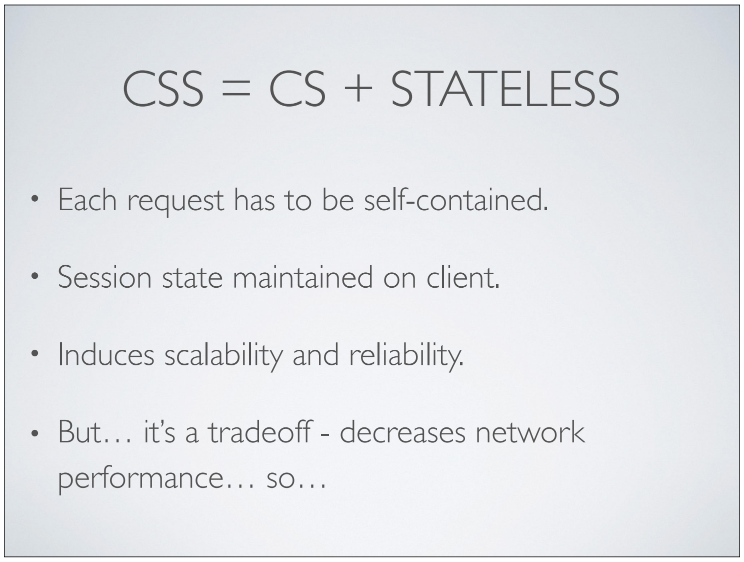


Fig. 1. REST derivation by style constraints.



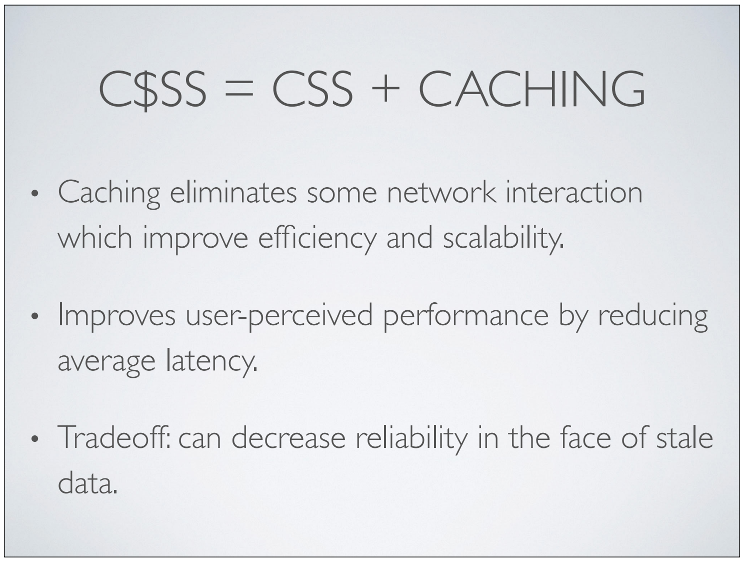
## CS = CLIENT SERVER

- Enforces separation of concerns
- Push UI stuff to client, improving portability
- Simplifies server
- Allows for independent evolution of client and server.



## CSS = CS + STATELESS

- Each request has to be self-contained.
- Session state maintained on client.
- Induces scalability and reliability.
- But… it's a tradeoff - decreases network performance… so…



## C$SS = CSS + CACHING

- Caching eliminates some network interaction which improve efficiency and scalability.
- Improves user-perceived performance by reducing average latency.
- Tradeoff: can decrease reliability in the face of stale data.

## U = UNIFORM INTERFACE

- Effectively decouples implementations from the services they provide
- Encourages independent evolution.

## REST INTERFACE CONSTRAINTS

- Identification of resources
- Manipulation of resources through representations
- Self-descriptive messages.
- Hypermedia as the *engine of application state.*

## LS: LAYERED STYLE

- Layering bounds overall complexity (can't see beyond immediate layer)
- Component simplification by moving common functionality to a shared intermediary.
- Tradeoff: Overhead and latency, hence caching intermediaries

## COD: CODE ON DEMAND

- e.g. Java Applets, Javascript, etc.
- Simplified clients by reducing the number of features needing pre-implementation.
- Improves extensibility.

## IMPLEMENTING DISTRIBUTED HYPERMEDIA

- 3 options:
  - render that data on server and send "fixed image" to client.
  - encapsulate rendering engine with data and send to client.
  - send raw data to client along with metadata that describes it. Client can choose its own rendering engine.

## REST ADOPTS A HYBRID APPROACH

- Shared understanding of data types with metadata.
- Limiting scope of what is revealed via a standard interface.
- Optionally supports COD where needed.

## RESOURCES

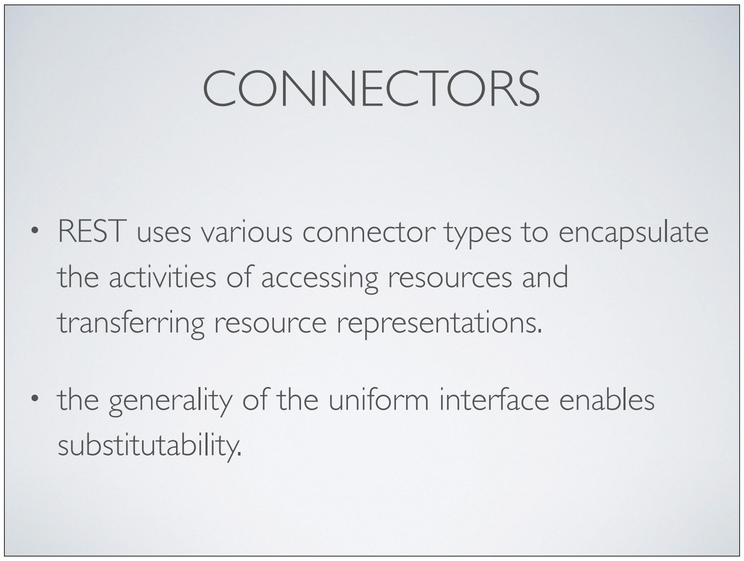
- A resource R is a temporally varying membership function $M_R(t)$, which for time $t$ maps to a set of entities or values.
- values in the set may be resource representations or resource identifier.
- R can map to the empty set.

## RESOURCE IDENTIFIERS

- identifiers are universal and unique.
- REST provides a generic interface for accessing or manipulating the value set of a resource.
- identifier remains the same, even when the info represented changes.
- Authors choose the identifier.

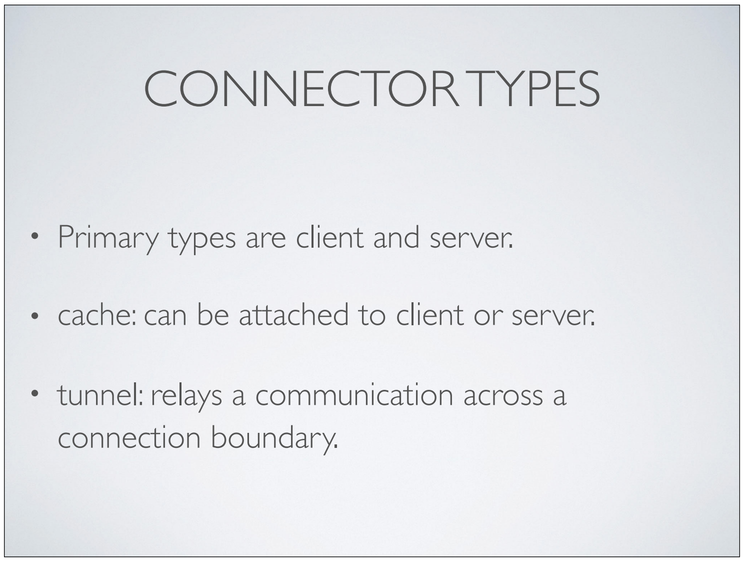
## REPRESENTATIONS

- A representation is a sequence of bytes, plus representation metadata to describe those bytes.
- Resources are maintained by the server (details of resource implementation hidden from clients).
- Clients perform actions on a resource by using a representation to capture the current (or intended) state and transferring that representation from (to) the server.



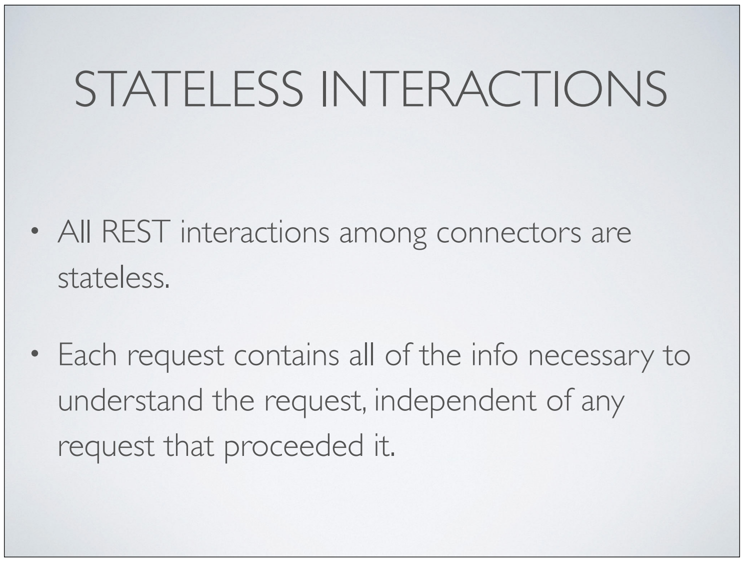
## CONNECTORS

- REST uses various connector types to encapsulate the activities of accessing resources and transferring resource representations.
- the generality of the uniform interface enables substitutability.



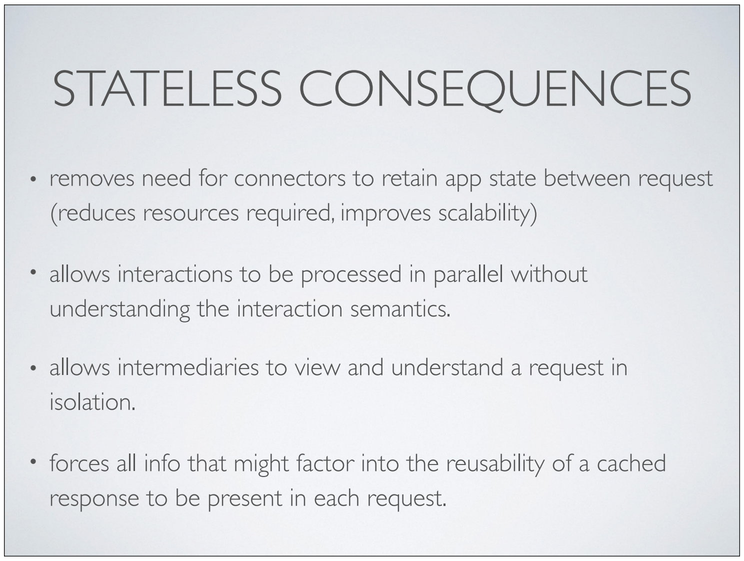
## CONNECTOR TYPES

- Primary types are client and server.
- cache: can be attached to client or server.
- tunnel: relays a communication across a connection boundary.



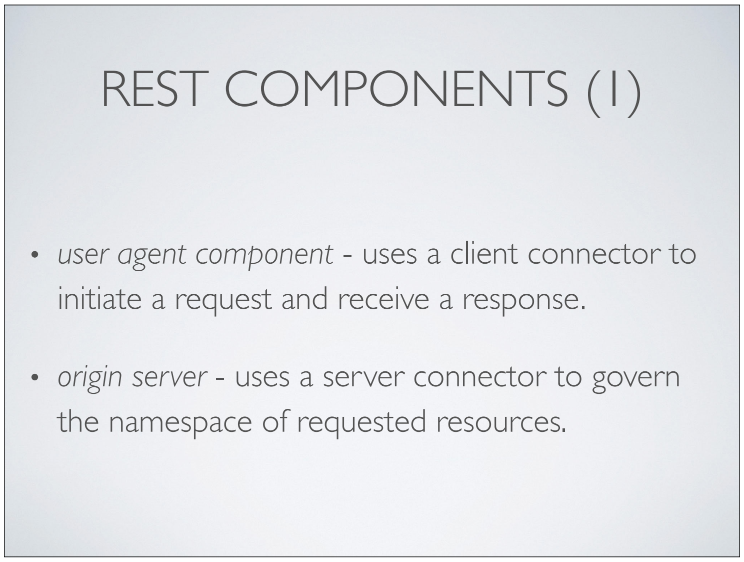
## STATELESS INTERACTIONS

- All REST interactions among connectors are stateless.
- Each request contains all of the info necessary to understand the request, independent of any request that proceeded it.



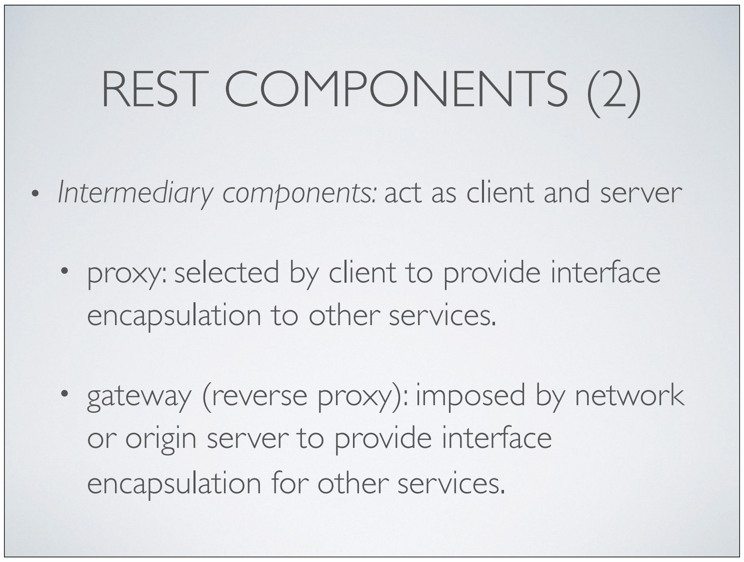
## STATELESS CONSEQUENCES

- removes need for connectors to retain app state between request (reduces resources required, improves scalability)
- allows interactions to be processed in parallel without understanding the interaction semantics.
- allows intermediaries to view and understand a request in isolation.
- forces all info that might factor into the reusability of a cached response to be present in each request.



## REST COMPONENTS (1)

- *user agent component* - uses a client connector to initiate a request and receive a response.
- *origin server* - uses a server connector to govern the namespace of requested resources.



## REST COMPONENTS (2)

- *Intermediary components:* act as client and server
  - proxy: selected by client to provide interface encapsulation to other services.
  - gateway (reverse proxy): imposed by network or origin server to provide interface encapsulation for other services.

## CHARACTERIZING REST (1)

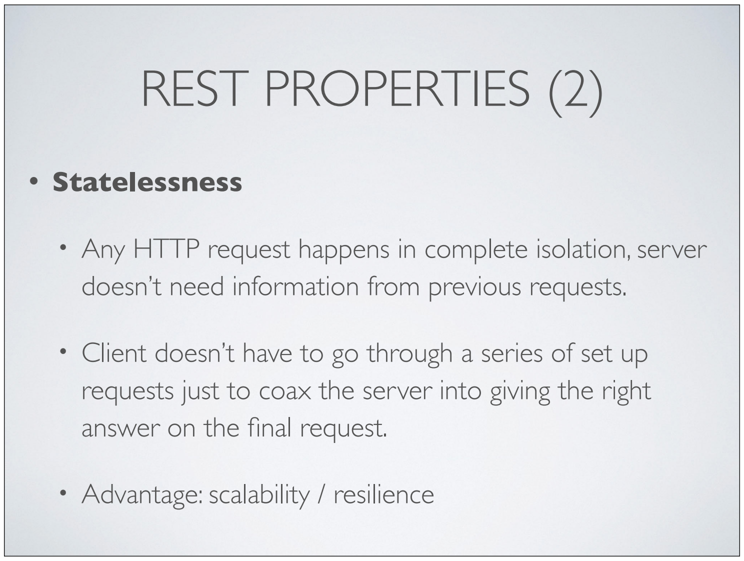
- State and functionality is divided into *resources*.
- A universal syntax is used to *name* resources (URI)
- Each resource has a constrained set of of typed *representations*.

## CHARACTERIZING REST (2)

- Each resource is *linked* to other resources.
- There is a highly constrained set of *operations* to access/manipulate resources.
- There is a protocol that is:
  - client/server
  - stateless
  - supports  transmission or representations via a chain of intermediaries.

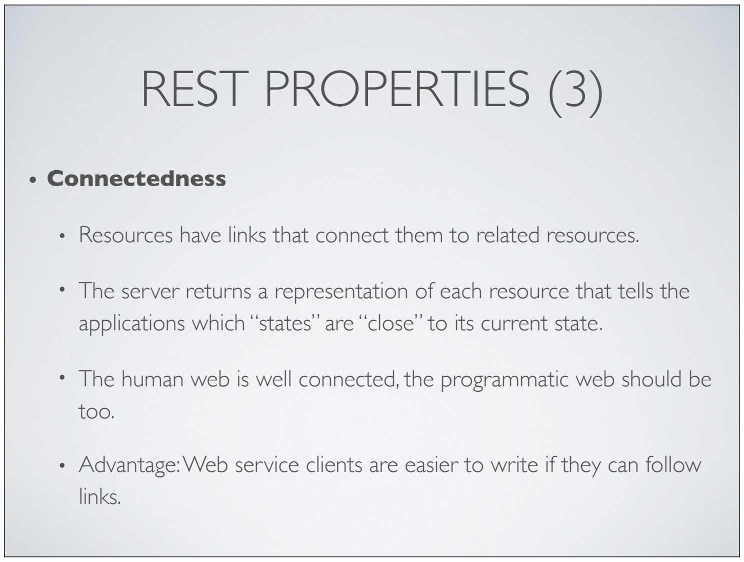
## REST PROPERTIES (1)

- **Addressability**
  - Any piece of information of conceivable interest to clients is exposed as a resource.
  - Each resource has a name, with synonyms limited as much as possible.
  - Advantages:
    - Can print the name in a book.
    - Can bookmark the name.
    - Supports browser history, RSS feeds, "back" button, ...
    - The name is a handle for caching representations of the resource.



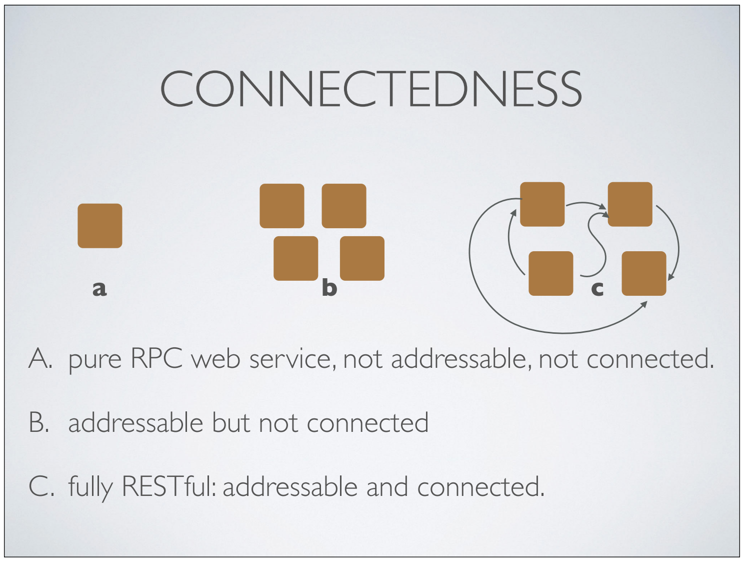
## REST PROPERTIES (2)

- **Statelessness**
  - Any HTTP request happens in complete isolation, server doesn't need information from previous requests.
  - Client doesn't have to go through a series of set up requests just to coax the server into giving the right answer on the final request.
  - Advantage: scalability / resilience

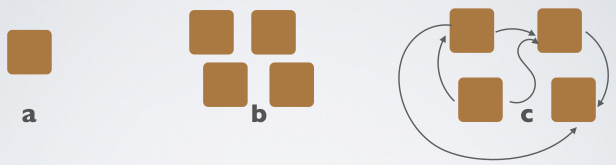


## REST PROPERTIES (3)

- **Connectedness**
  - Resources have links that connect them to related resources.
  - The server returns a representation of each resource that tells the applications which "states" are "close" to its current state.
  - The human web is well connected, the programmatic web should be too.
  - Advantage: Web service clients are easier to write if they can follow links.



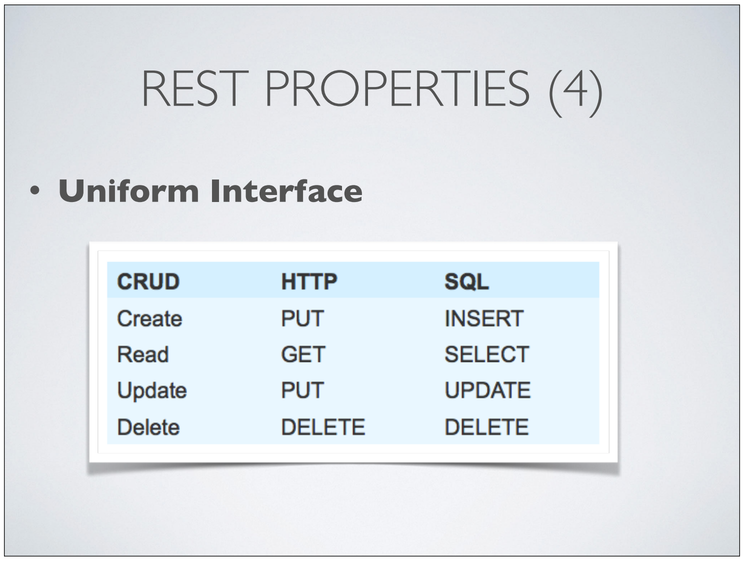
## CONNECTEDNESS



A. pure RPC web service, not addressable, not connected.

B. addressable but not connected

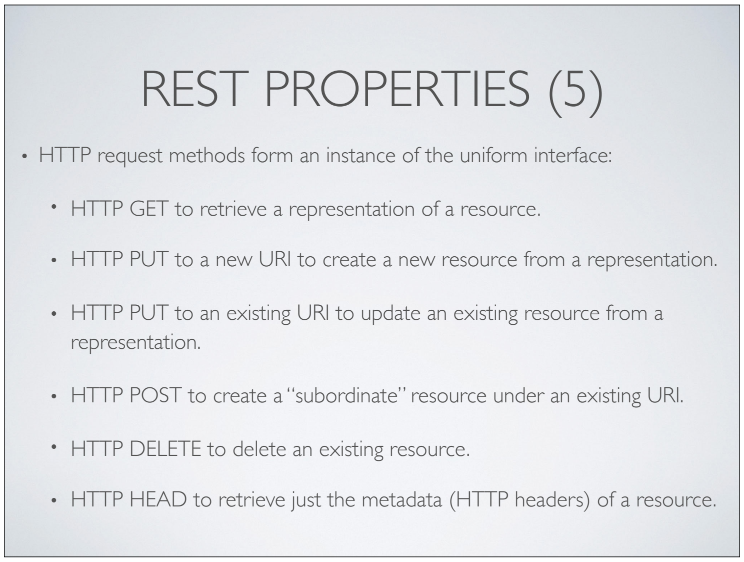
C. fully RESTful: addressable and connected.



## REST PROPERTIES (4)

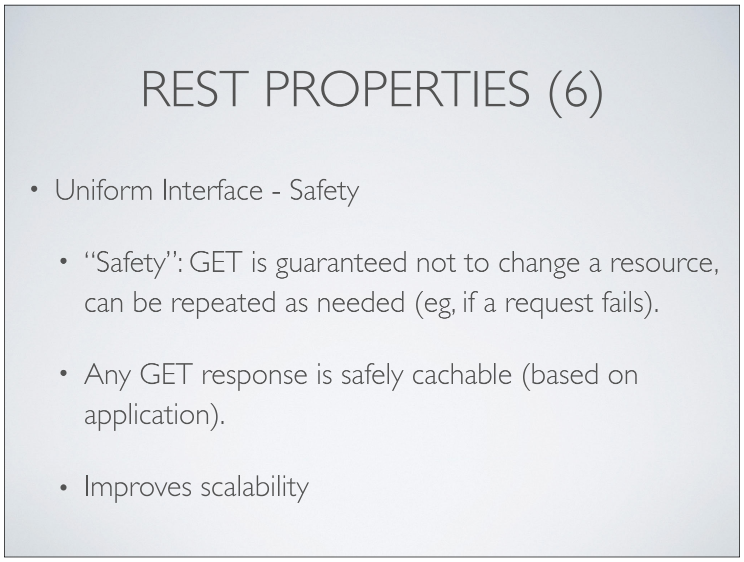
- **Uniform Interface**

| CRUD | HTTP | SQL |
|---|---|---|
| Create | PUT | INSERT |
| Read | GET | SELECT |
| Update | PUT | UPDATE |
| Delete | DELETE | DELETE |



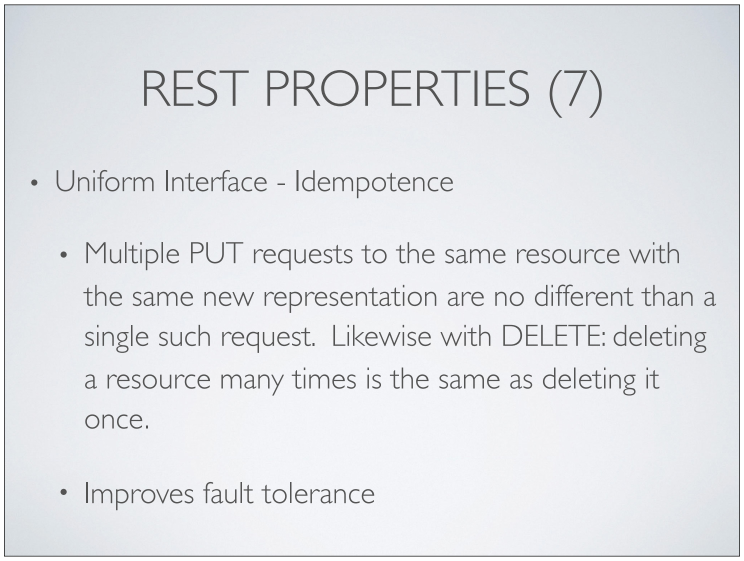
## REST PROPERTIES (5)

- HTTP request methods form an instance of the uniform interface:
  - HTTP GET to retrieve a representation of a resource.
  - HTTP PUT to a new URI to create a new resource from a representation.
  - HTTP PUT to an existing URI to update an existing resource from a representation.
  - HTTP POST to create a "subordinate" resource under an existing URI.
  - HTTP DELETE to delete an existing resource.
  - HTTP HEAD to retrieve just the metadata (HTTP headers) of a resource.



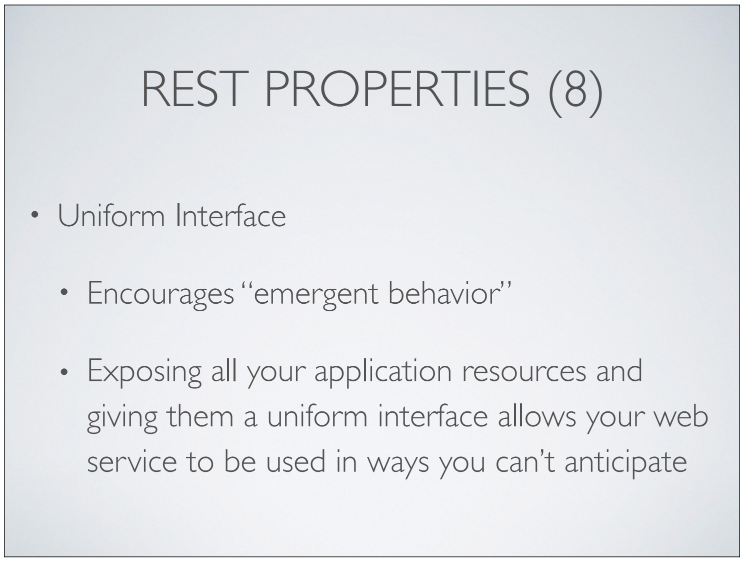
## REST PROPERTIES (6)

- Uniform Interface - Safety
  - "Safety": GET is guaranteed not to change a resource, can be repeated as needed (eg, if a request fails).
  - Any GET response is safely cachable (based on application).
  - Improves scalability



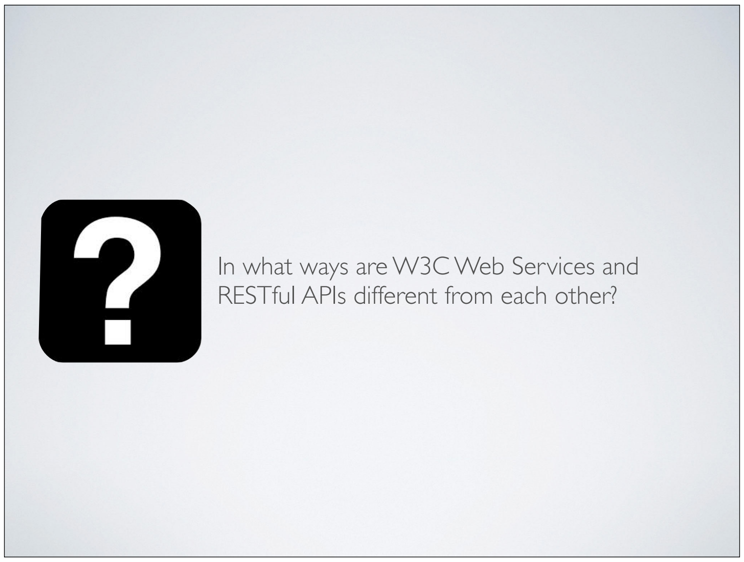
## REST PROPERTIES (7)

- Uniform Interface - Idempotence
  - Multiple PUT requests to the same resource with the same new representation are no different than a single such request. Likewise with DELETE: deleting a resource many times is the same as deleting it once.
  - Improves fault tolerance



## REST PROPERTIES (8)

- Uniform Interface
  - Encourages "emergent behavior"
  - Exposing all your application resources and giving them a uniform interface allows your web service to be used in ways you can't anticipate

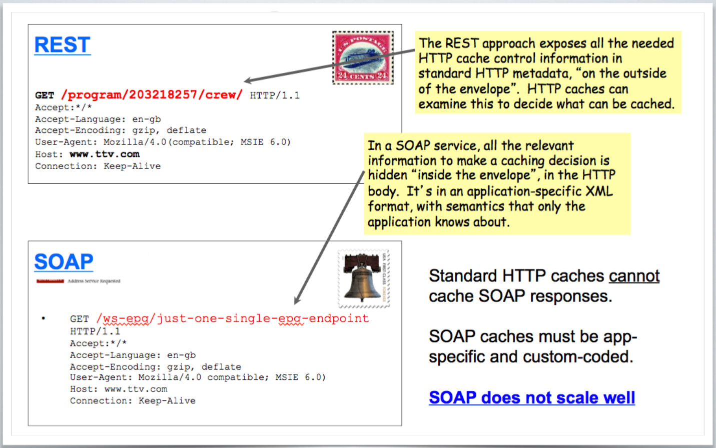


In what ways are W3C Web Services and RESTful APIs different from each other?

## SURFACE LEVEL DIFFS

- W3C Web Services are a full set of standards, based in practice on the RPC paradigm.
- REST is a set of criteria for judging distributed architectures, based on the successful architectural principles behind the web.
- SOAP, UDDI, and WSDL do more than vanilla REST: service discovery, service description, security, transactions.
- REST implementations have a smaller footprint than W3C Web Services.
- W3C Web Services don't leverage the web model and standard web infrastructure.

## HTTP CACHING



**REST**

```
GET /program/203218257/crew/ HTTP/1.1
Accept:*/*
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0(compatible; MSIE 6.0)
Host: www.ttv.com
Connection: Keep-Alive
```

The REST approach exposes all the needed HTTP cache control information in standard HTTP metadata, "on the outside of the envelope". HTTP caches can examine this to decide what can be cached.

In a SOAP service, all the relevant information to make a caching decision is hidden "inside the envelope", in the HTTP body. It's in an application-specific XML format, with semantics that only the application knows about.

**SOAP**

Address Service Requested

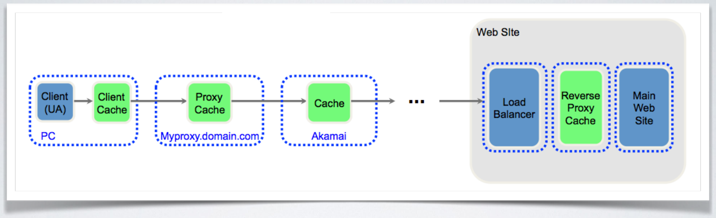
```
•   GET /ws-epg/just-one-single-epg-endpoint
    HTTP/1.1
    Accept:*/*
    Accept-Language: en-gb
    Accept-Encoding: gzip, deflate
    User-Agent: Mozilla/4.0 compatible; MSIE 6.0)
    Host: www.ttv.com
    Connection: Keep-Alive
```

Standard HTTP caches <u>cannot</u> cache SOAP responses.

SOAP caches must be app-specific and custom-coded.

**SOAP does not scale well**

## HTTP CACHING OPPORTUNITIES



- HTTP caches exist in the client ("user agent"), the proxy server, at intermediate points in the path to the web site, and at the border of your web site.
- Caching in W3C web services are application specific.

## THE "SOFT UNDERBELLY"

- WSDL is problematic because:
  - tight coupling of service interface to underlying implementation
  - tooling makes it easy to promote anything as a service.
  - makes for "chatty" interfaces.
  - does not lend itself to "future proof" systems.

## READING ASSIGNMENT

- see Course schedule online for complete list of required readings.