Università della Svizzera italiana

Facoltà di scienze informatiche

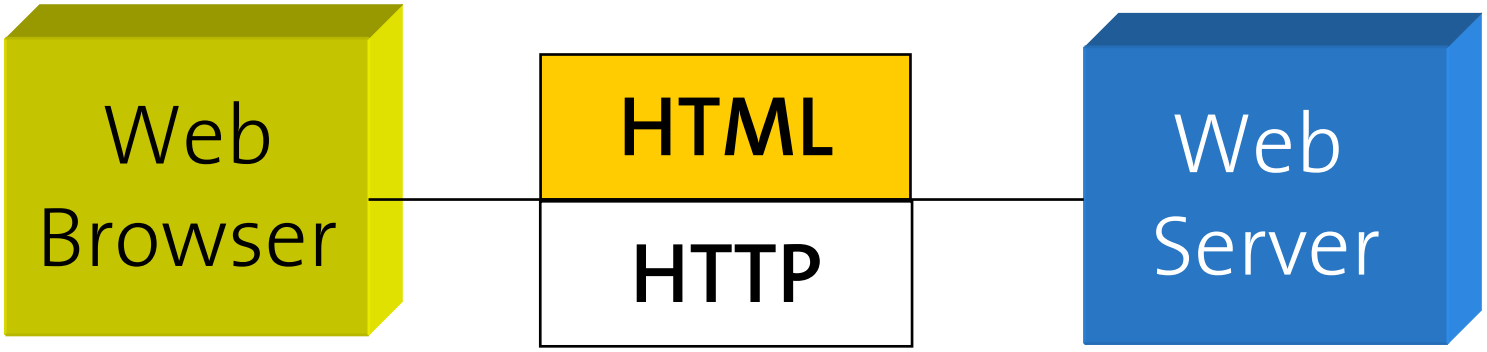
UC Berkeley School of Information

4 REST vs WS-* Comparison

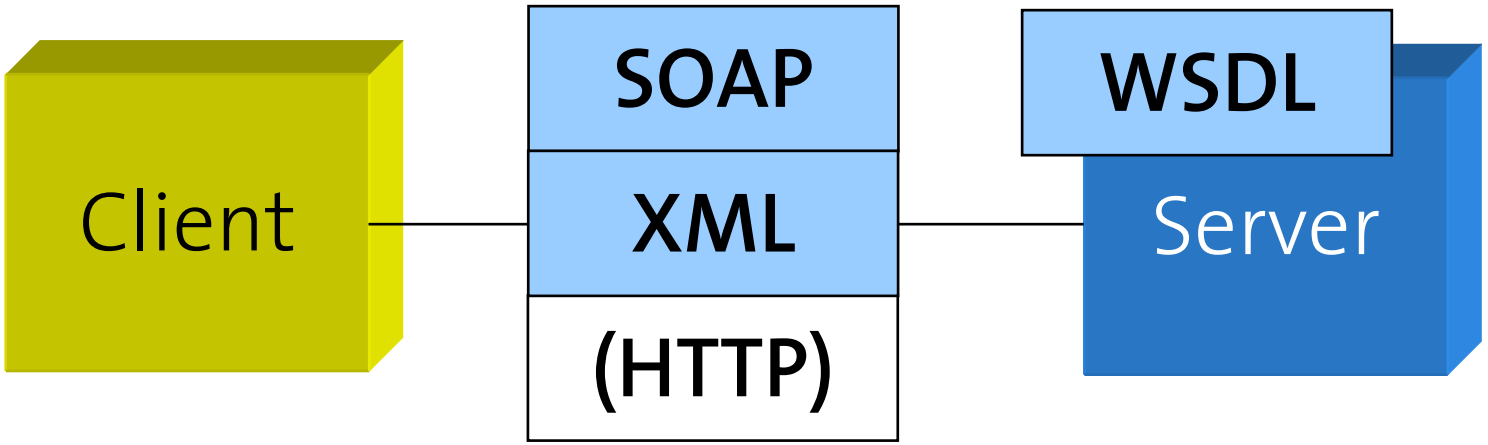
Cesare Pautasso
Faculty of Informatics
University of Lugano, Switzerland

c.pautasso@ieee.org
http://www.pautasso.info

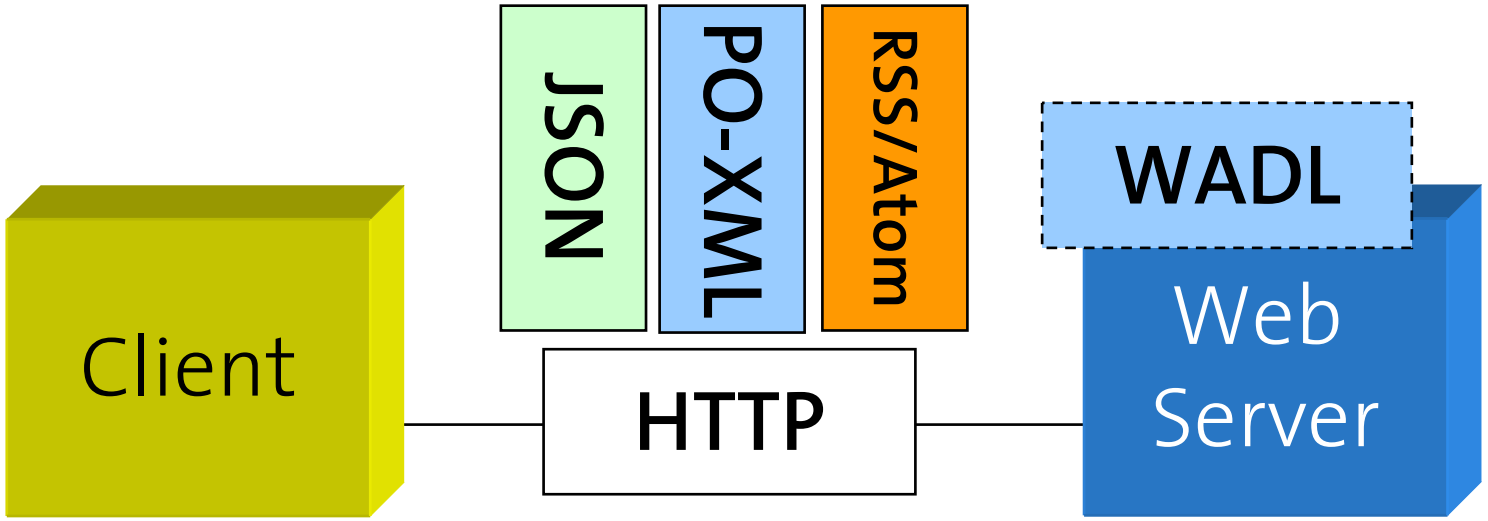
Web Sites (1992)



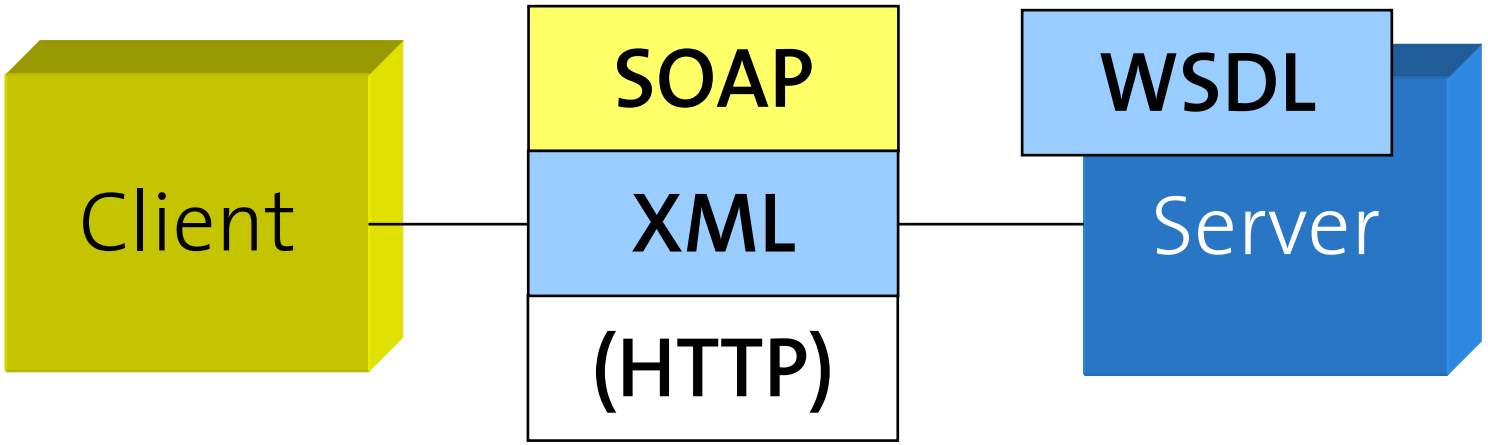
WS-* Web Services (2000)



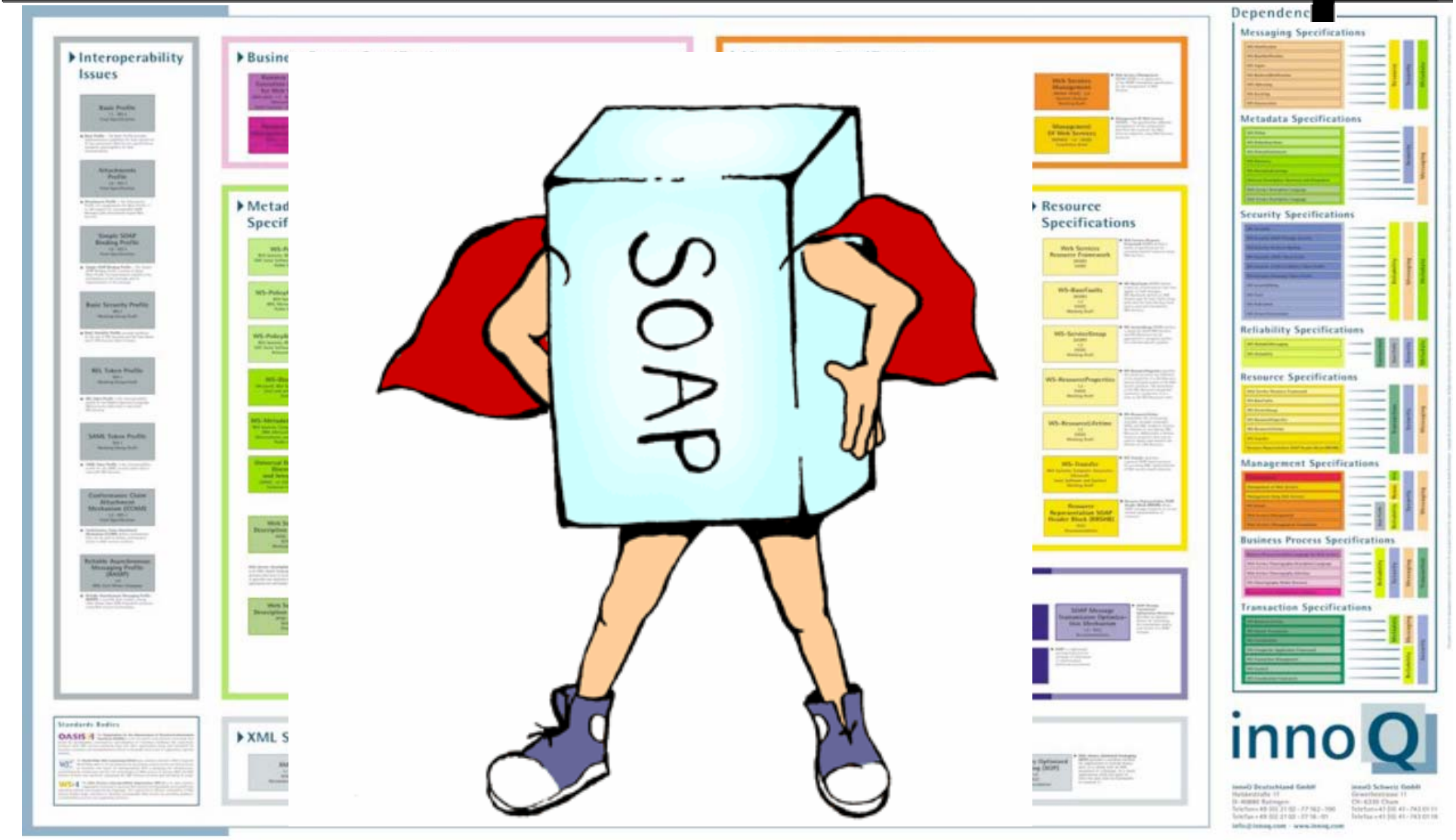
RESTful Web Services (2007)



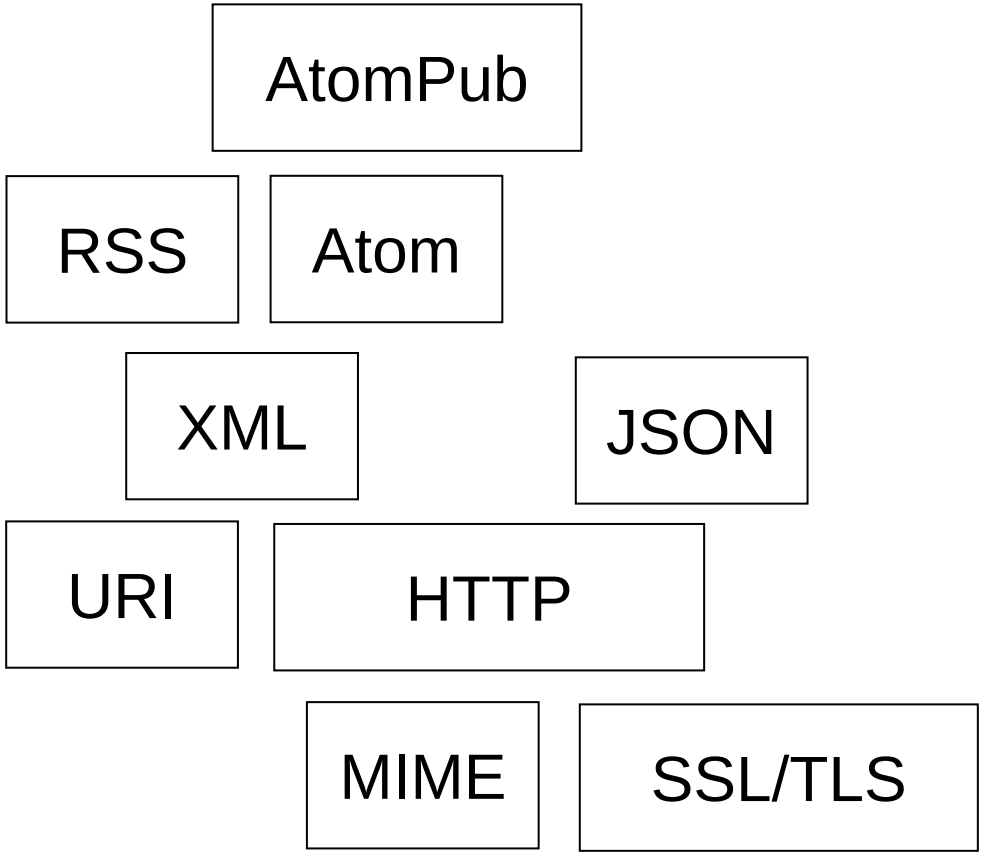
WS-* Web Services (2000)



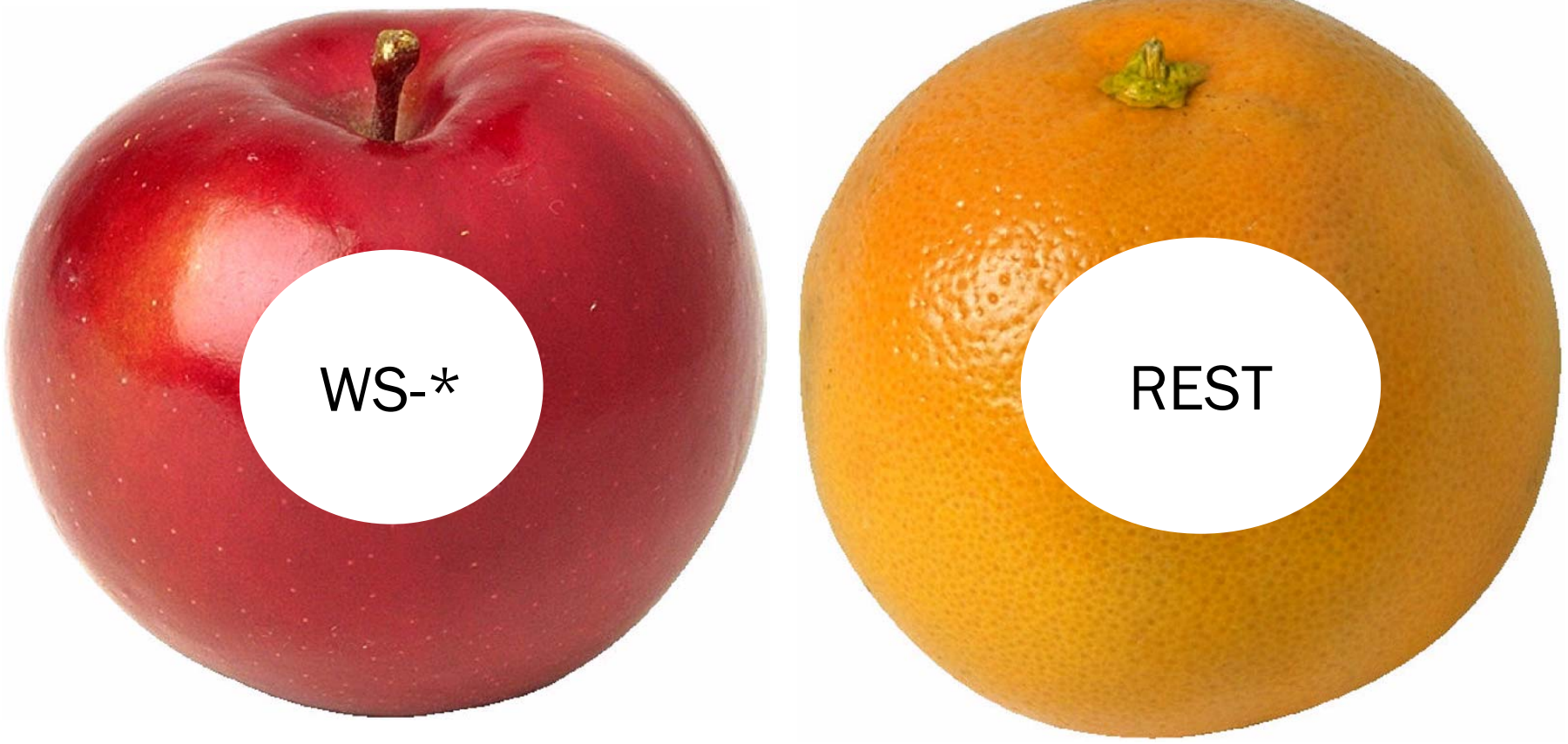
WS-* Standards Stack



RESTful Web Services Standards Stack

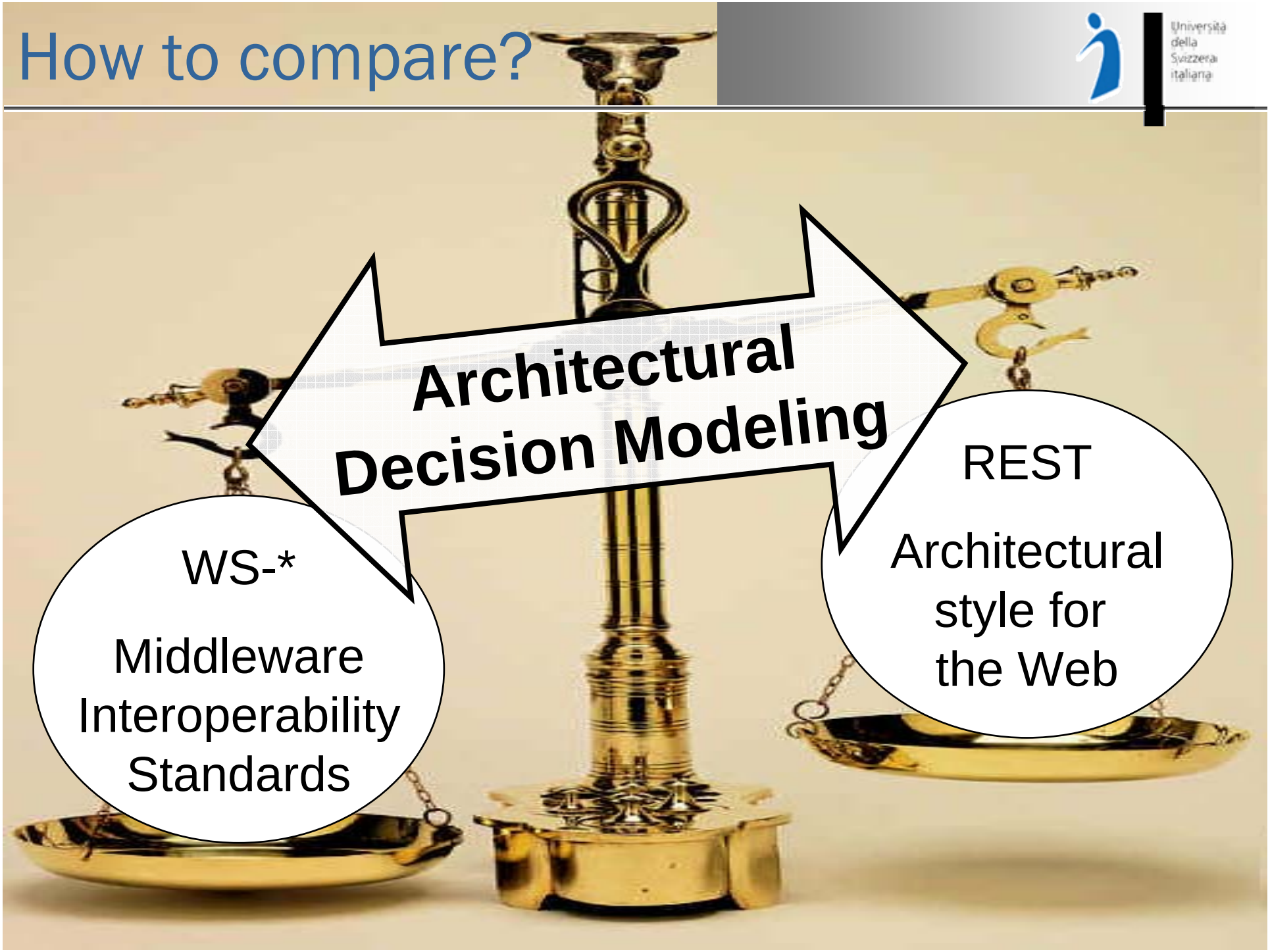


Can we really compare WS-* vs. REST?



Can we really compare WS-* vs. REST?

How to compare?



Architectural Decisions

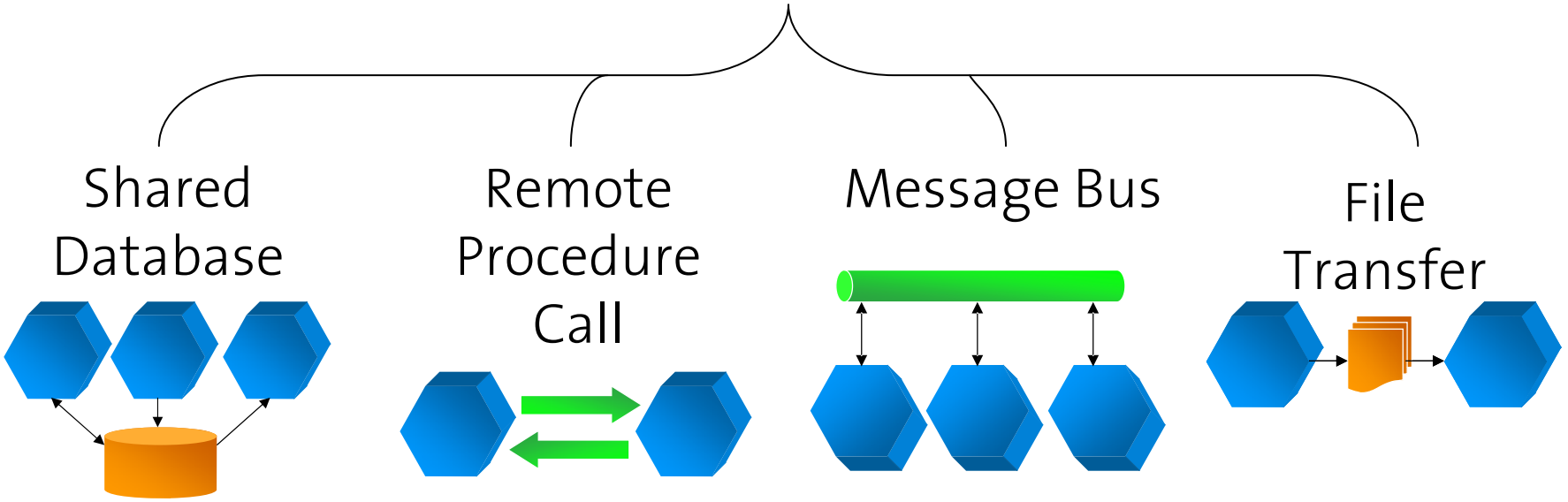
- Architectural decisions capture the main design issues and the rationale behind a chosen technical solution
- The choice between REST vs. WS-* is an important architectural decision for Web service design
- Architectural decisions affect one another

Architectural Decision:
Programming Language

Architecture Alternatives:
1. **Java**
2. **C#**
3. **C++**
4. **C**
5. **Eiffel**
6. **Ruby**
7. **…**

Rationale

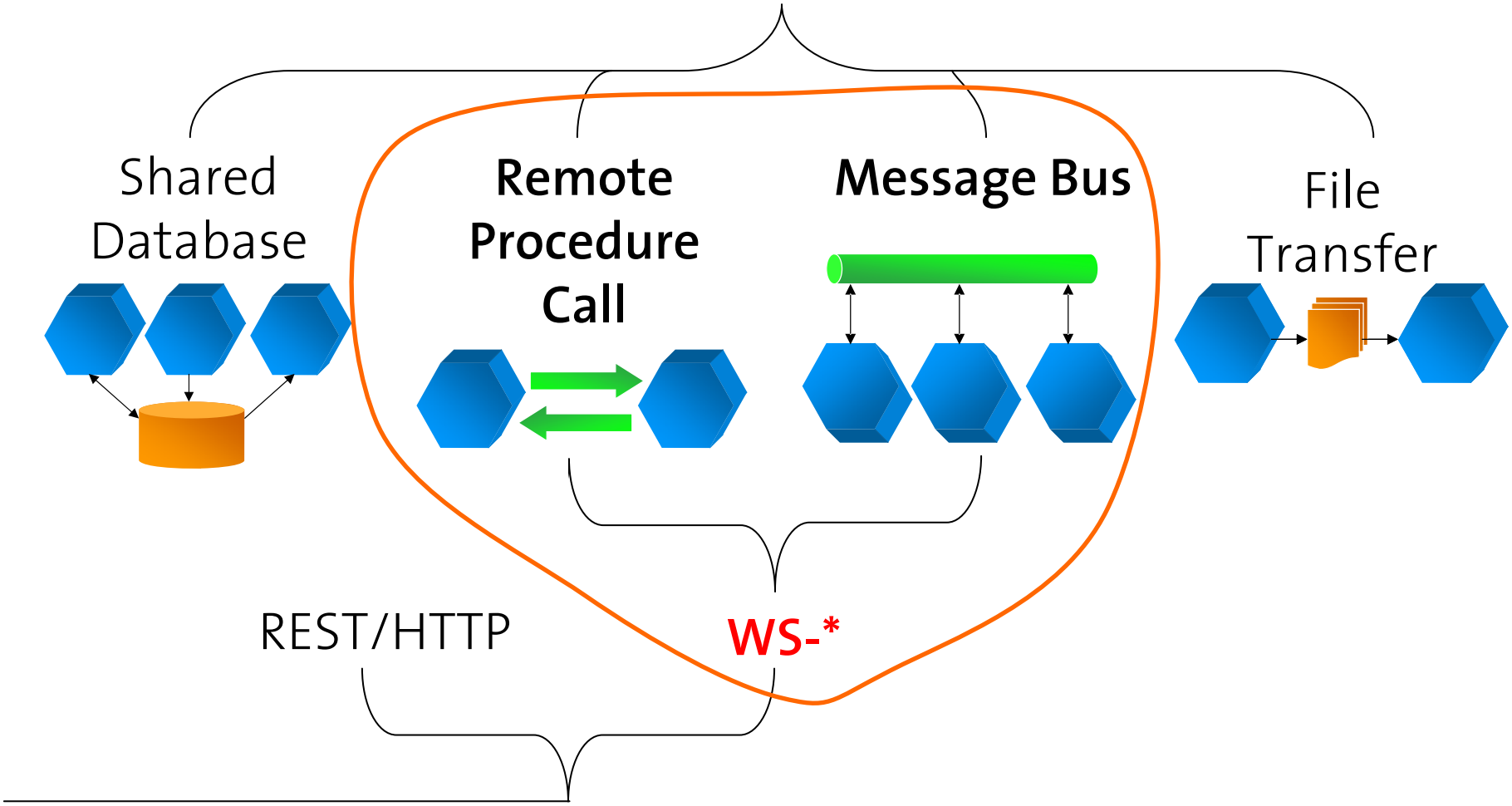
Application Integration Styles



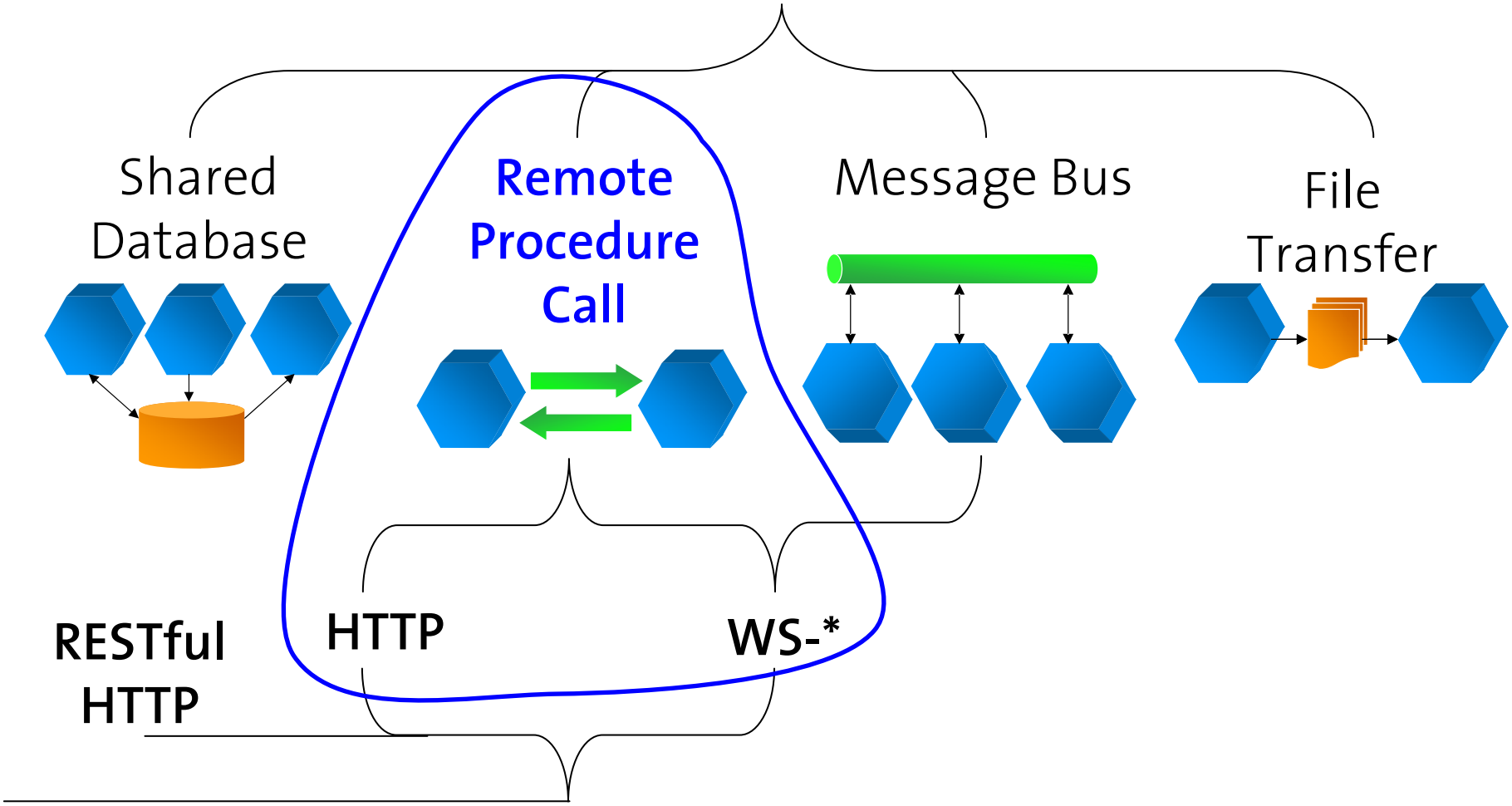
REST/HTTP WS-*

Technology Platform

Related Decisions (WS-*)



Related Decisions (RPC)



Decision Space Overview

Architectural Decision and AAs	REST	WS-*
Integration Style	1 AA	2 AAs
Shared Database		
File Transfer		
Remote Procedure Call	✓	✓
Messaging		✓
Contract Design	1 AA	2 AAs
Contract-first		✓
Contract-last		✓
Contract-less	✓	
Resource Identification	1 AA	n/a
Do-it-yourself	✓	
URI Design	2 AA	n/a
"Nice" URI scheme	✓	
No URI scheme	✓	
Resource Interaction Semantics	2 AAs	n/a
Lo-REST (`POST`, `GET` only)	✓	
Hi-REST (4 verbs)	✓	
Resource Relationships	1 AA	n/a
Do-it-yourself	✓	
Data Representation/Modeling	1 AA	1 AA
XML Schema	(✓)[a]	✓
Do-it-yourself	✓	
Message Exchange Patterns	1 AA	2 AAs
Request-Response	✓	✓
One-Way		✓
Service Operations Enumeration	n/a	≥3 AAs
By functional domain		✓
By non-functional properties and QoS		✓
By organizational criterion (versioning)		✓
Total Number of Decisions, AAs	**8**, 10	**5**, ≥10

[a] Optional

Table 2: Conceptual Comparison Summary

Architectural Decision and AAs	REST	WS-*
Transport Protocol	1 AA	≥7 AAs
HTTP	✓	✓[a]
waka [13]	(✓)[b]	
TCP		✓
SMTP		✓
JMS		✓
MQ		✓
BEEP		✓
IIOP		✓
Payload Format	≥6 AAs	1 AA
XML (SOAP)	✓	✓
XML (POX)	✓	
XML (RSS)	✓	
JSON [10]	✓	
YAML	✓	
MIME	✓	
Service Identification	1 AA	2 AA
URI	✓	✓
WS-Addressing		✓
Service Description	3 AAs	2 AAs
Textual Documentation	✓	
XML Schema	(✓)[c]	✓
WSDL	✓[d]	✓
WADL [18]	✓	
Reliability	1 AA	4 AAs
HTTPR [38][e]	(✓)	(✓)
WS-Reliability		✓
WS-ReliableMessaging		✓
Native		✓
Do-it-yourself	✓	✓
Security	1 AA	2 AAs
HTTPS	✓	✓
WS-Security		✓

Transactions	1 AA	3 AAs
WS-AT, WS-BA		✓
WS-CAF		✓
Do-it-yourself	✓	✓
Service Composition	2 AAs	2 AAs
WS-BPEL		✓
Mashups	✓	
Do-it-yourself	✓	✓
Service Discovery	1 AAs	2 AAs
UDDI		✓
Do-it-yourself	✓	✓
Implementation Technology	many	many
...	✓	✓
Total Number of Decisions, AAs	**10**, ≥17	**10**, ≥25

[a] Limited to only the verb `POST`
[b] Still under development
[c] Optional
[d] WSDL 2.0
[e] Not standard

Table 3: Technology Comparison Summary

Architectural Principle and Aspects	REST	WS-*
Protocol Layering	yes	yes
HTTP as application-level protocol	✓	
HTTP as transport-level protocol		✓
Dealing with Heterogeneity	yes	yes
Browser Wars	✓	
Enterprise Computing Middleware		✓
Loose Coupling, aspects covered	yes, 2	yes, 3
Time/Availability		✓
Location (Dynamic Late Binding)	(✓)	✓
Service Evolution:		
Uniform Interface	✓	
XML Extensibility	✓	✓
Total Principles Supported	**3**	**3**

Table 1: Principles Comparison Summary

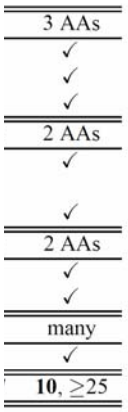
Decision Space Overview

21 Decisions and 64 alternatives

Classified by level of abstraction:

- 3 Architectural **Principles**
- 9 **Conceptual** Decisions
- 9 **Technology**-level Decisions

Decisions help us to **measure the complexity** implied by the choice of REST or WS-*



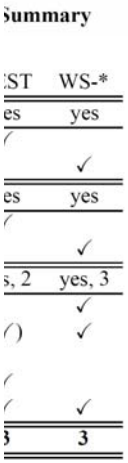


Table 2: Conceptual Comparison Summary

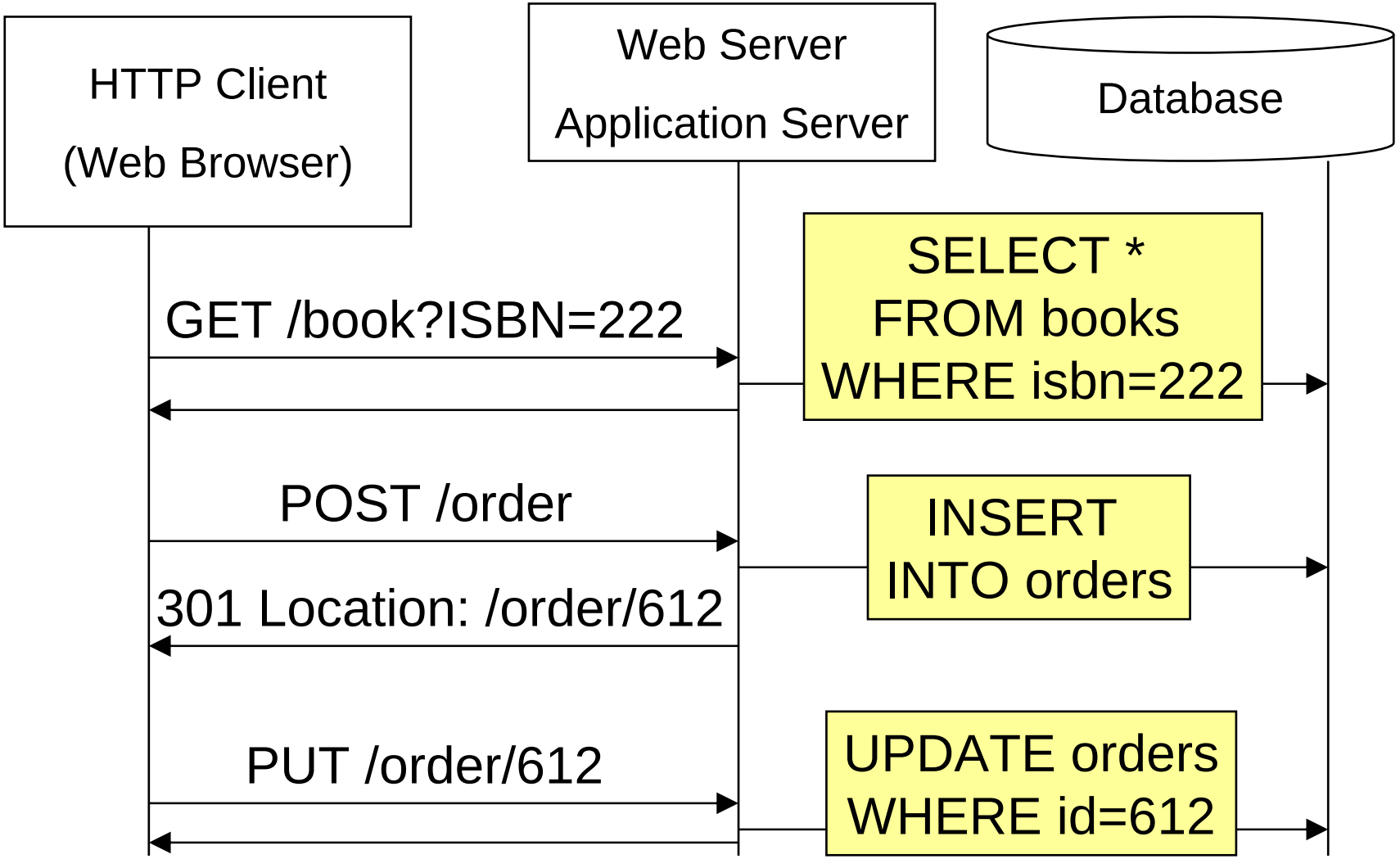
Table 1: Principles Comparison Summary

Architectural Principles

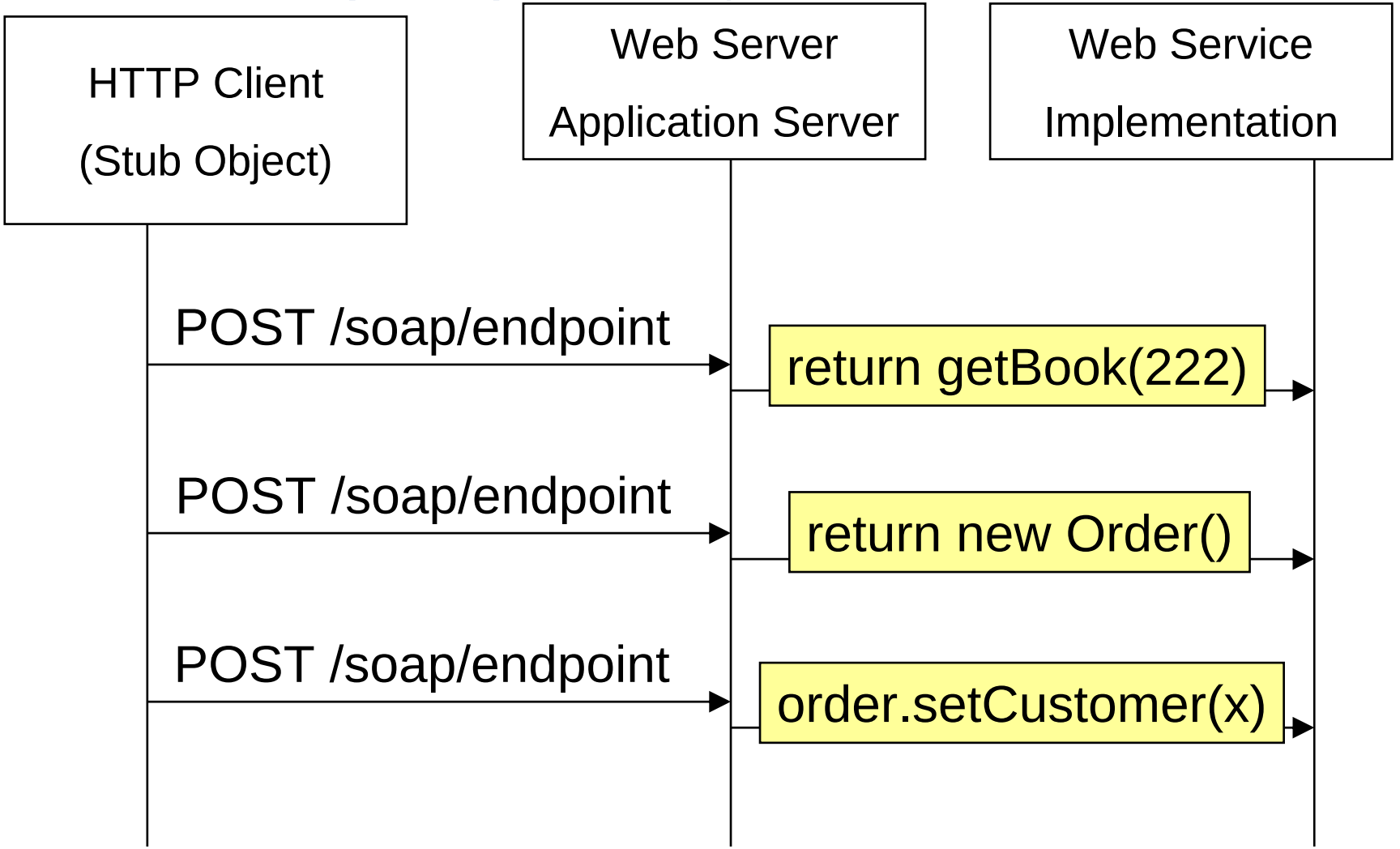
1. Protocol Layering
   - HTTP = Application-level Protocol (REST)
   - HTTP = Transport-level Protocol (WS-*)
2. Dealing with Heterogeneity
3. Loose Coupling

Cesare Pautasso and Erik Wilde. *Why is the Web Loosely Coupled? A Multi-Faceted Metric for Service Design*, WWW2009 (Wednesday 16:30)

RESTful Web Service Example



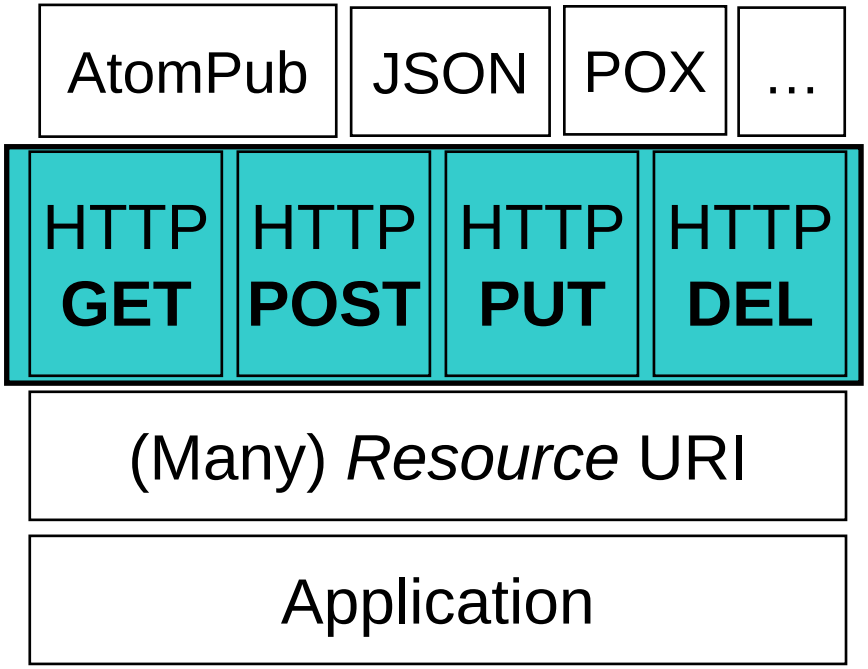
WS-* Service Example (from REST perspective)



Protocol Layering

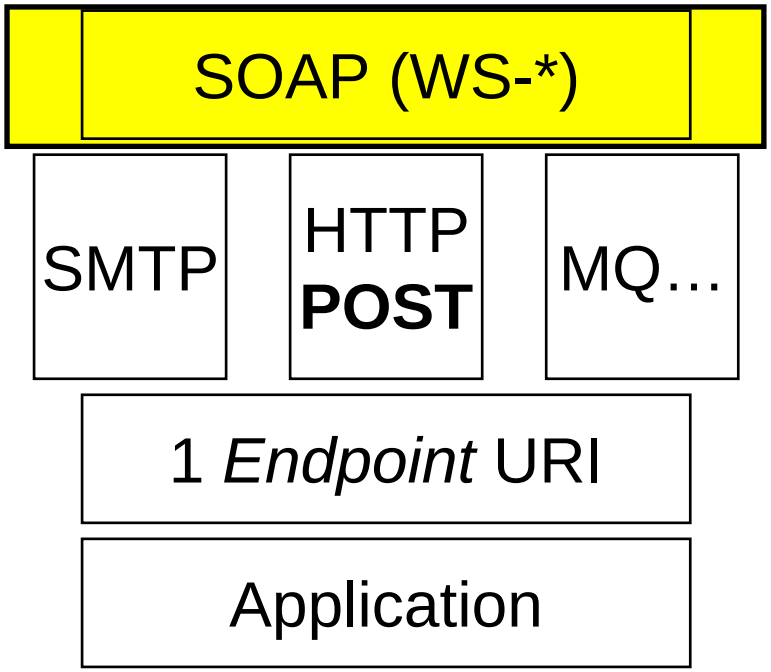
"The Web is the universe of globally accessible information" (Tim Berners Lee)

- Applications should publish their data on the Web (through URI)



"The Web is the universal (tunneling) transport for messages"

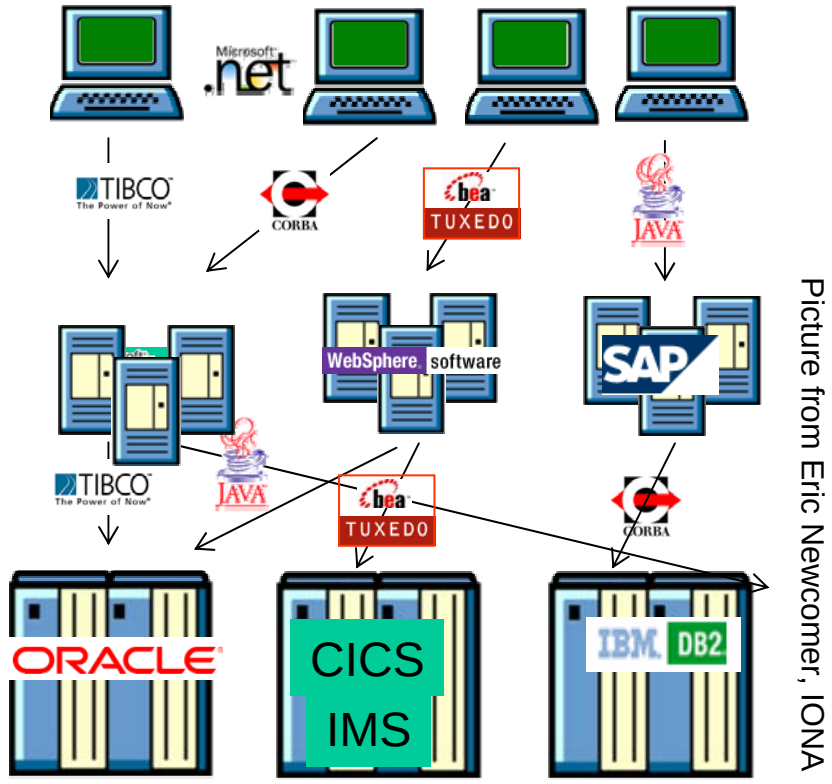
- Applications get a chance to interact but they remain "outside of the Web"



Dealing with Heterogeneity

- Enable Cooperation
- Web Applications

- Enable Integration
- Enterprise Computing



Picture from Eric Newcomer, IONA

Managing State

- REST provides explicit state transitions
  - Communication is stateless*
  - Resources contain data **and hyperlinks** representing valid state transitions
  - Clients maintain application state correctly by navigating hyperlinks
- Techniques for adding session to HTTP:
  - Cookies (HTTP Headers)
  - URI Re-writing
  - Hidden Form Fields

- SOAP services have implicit state transitions
  - Servers may maintain conversation state across multiple message exchanges
  - Messages contain only data (but do not include information about valid state transitions)
  - Clients maintain state by guessing the state machine of the service
- Techniques for adding session to SOAP:
  - Session Headers (non standard)
  - WS-Resource Framework (HTTP on top of SOAP on top of HTTP)

(*) Each client request to the server must contain all information needed to understand the request, without referring to any stored context on the server. Of course the server stores the state of its resources, shared by all clients.

What about service description?

- REST relies on human readable documentation that defines requests URIs and responses (XML, JSON)
- Interacting with the service means hours of testing and debugging URIs manually built as parameter combinations. (Is is it really that simpler building URIs by hand?)
- Why do we need strongly typed SOAP messages if both sides already agree on the content?
- WADL proposed Nov. 2006
- XML Forms enough?

- Client stubs can be built from WSDL descriptions in most programming languages
- Strong typing
- Each service publishes its own interface with different semantics
- WSDL 1.1 (entire port type can be bound to HTTP GET or HTTP POST or SOAP/HTTP POST or other protocols)
- WSDL 2.0 (more flexible, each operation can choose whether to use GET or POST)

What about security?

- REST security is all about HTTPS (HTTP + SSL/TLS)
- Proven track record (SSL1.0 from 1994)
- HTTP Basic Authentication (RFC 2617, 1999 RFC 1945, 1996)

- SOAP security extensions defined by WS-Security (from 2004)
- XML Encryption (2002)
- XML Signature (2001)
- Implementations are starting to appear now
  - Full interoperability moot
  - Performance?

- Secure, point to point communication (Authentication, Integrity and Encryption)

- Secure, end-to-end communication – Self-protecting SOAP messages (does not require HTTPS)

What about asynchronous reliable messaging?

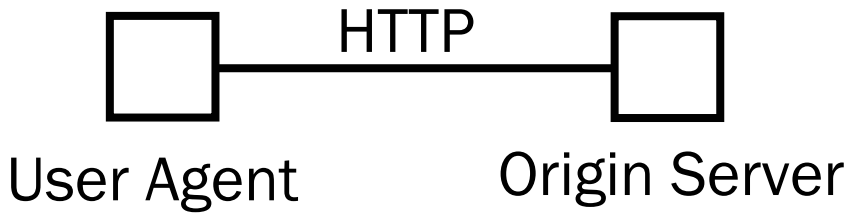
- Although HTTP is a synchronous protocol, it can be used to "simulate" a message queue.

```
POST /queue

202 Accepted
Location:
  /queue/message/1230213
-----------------------
GET /queue/message/1230213

DELETE /queue/message/1230213
```

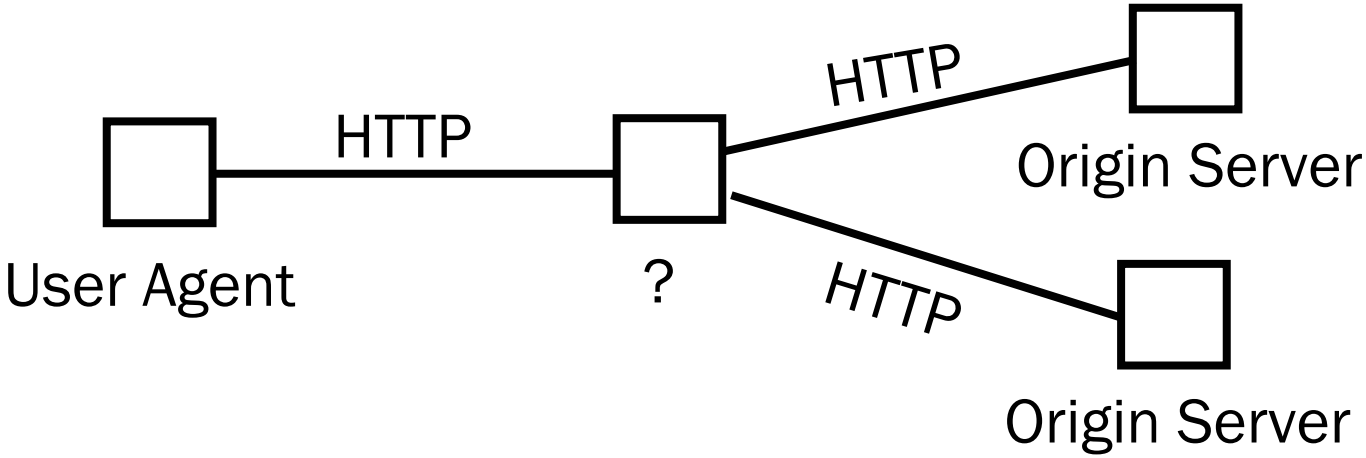
- SOAP messages can be transferred using asynchronous transport protocols and APIs (like JMS, MQ, ...)
- WS-Addressing can be used to define transport-independent endpoint references
- WS-ReliableExchange defines a protocol for reliable message delivery based on SOAP headers for message identification and acknowledgement

What about composition?

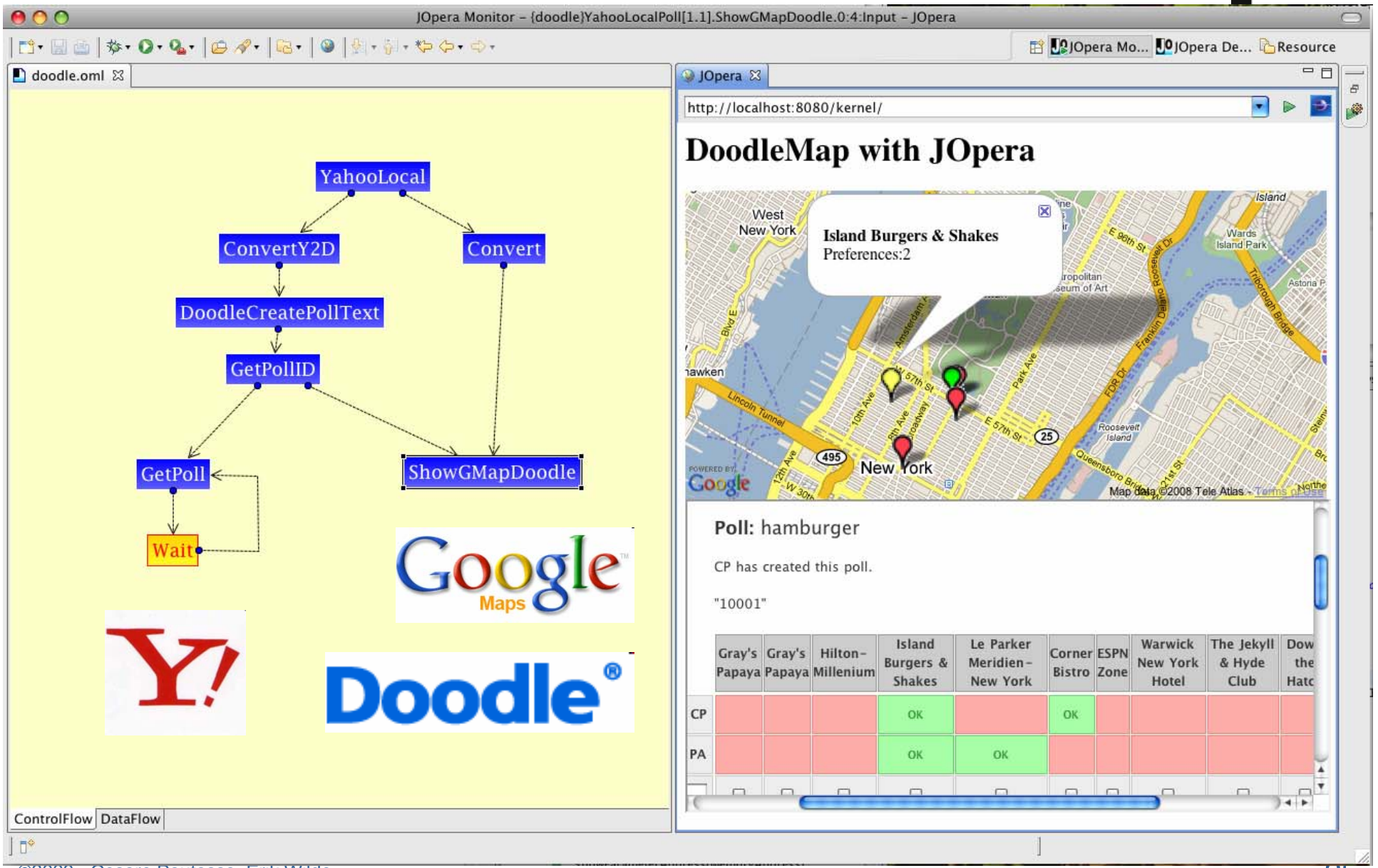
- The basic REST design elements do not take composition into account



- WS-BPEL is the standard Web service composition language. Business process models are used to specify how a collection of services is orchestrated into a composite service
- Can we apply WS-BPEL to RESTful services?



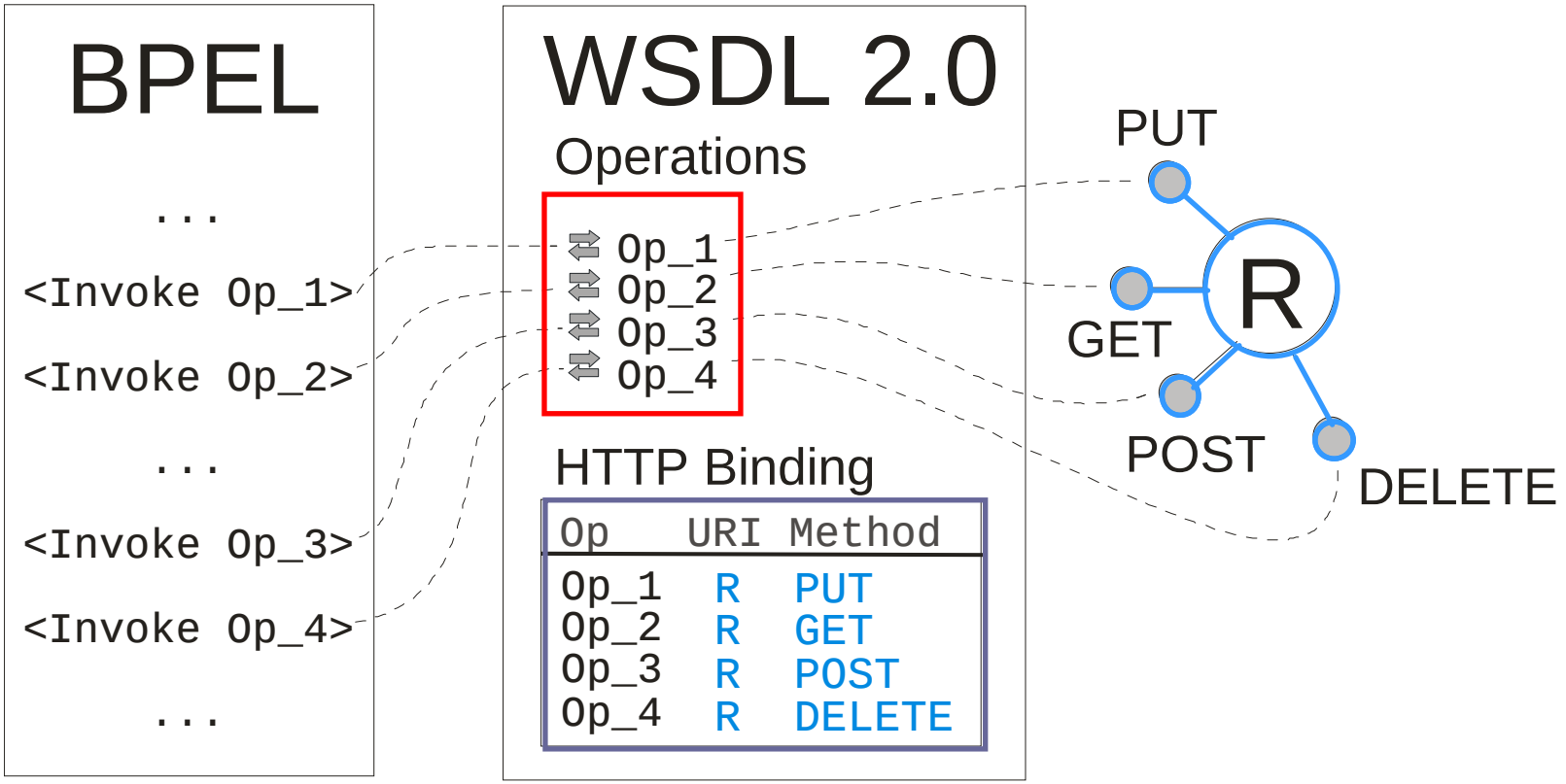
RESTful Composition Example



BPEL and WSDL 2.0

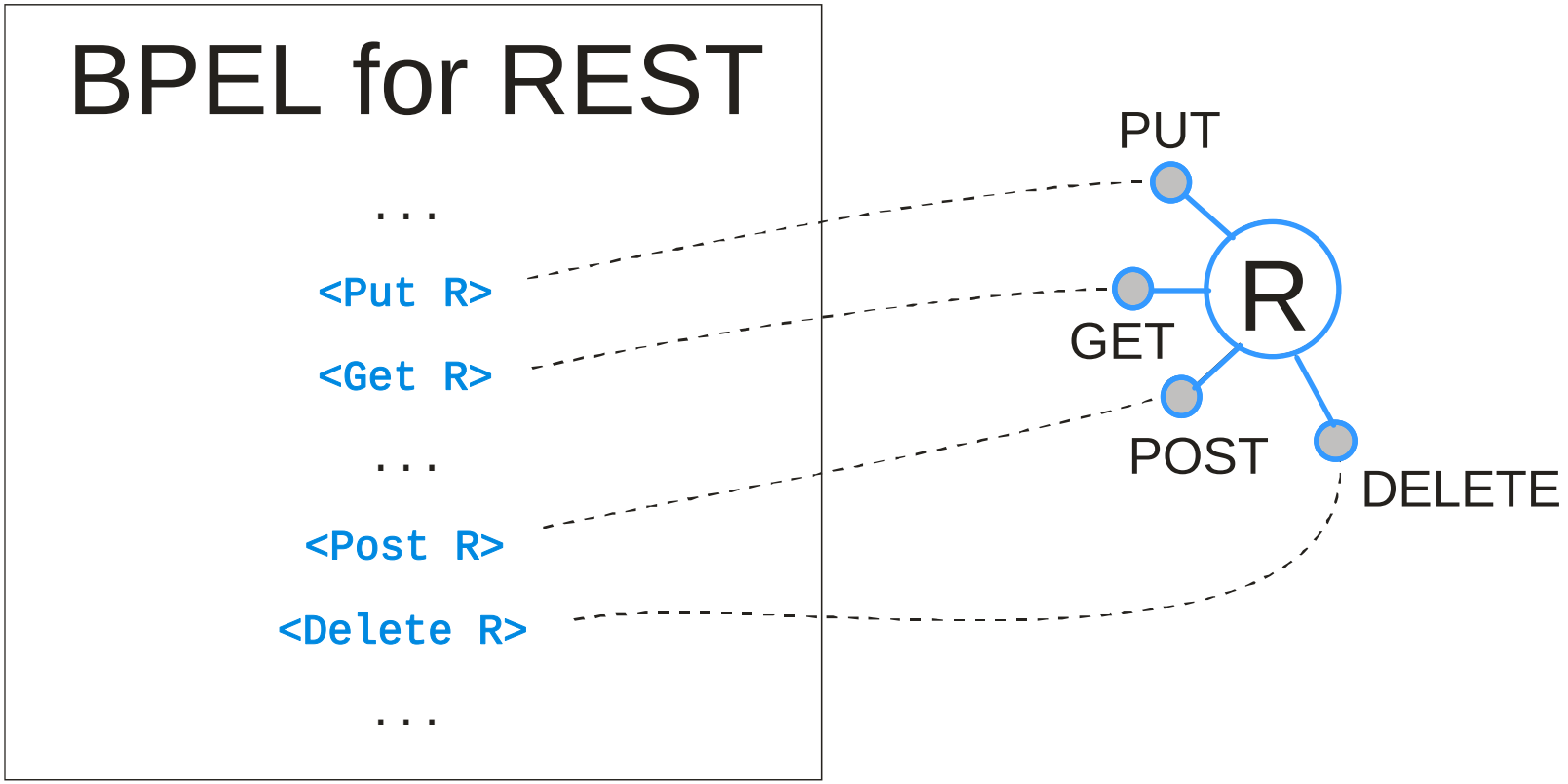
WSDL 2.0 HTTP Binding can wrap RESTful Web Services

(WS-BPEL 2.0 does not support WSDL 2.0)



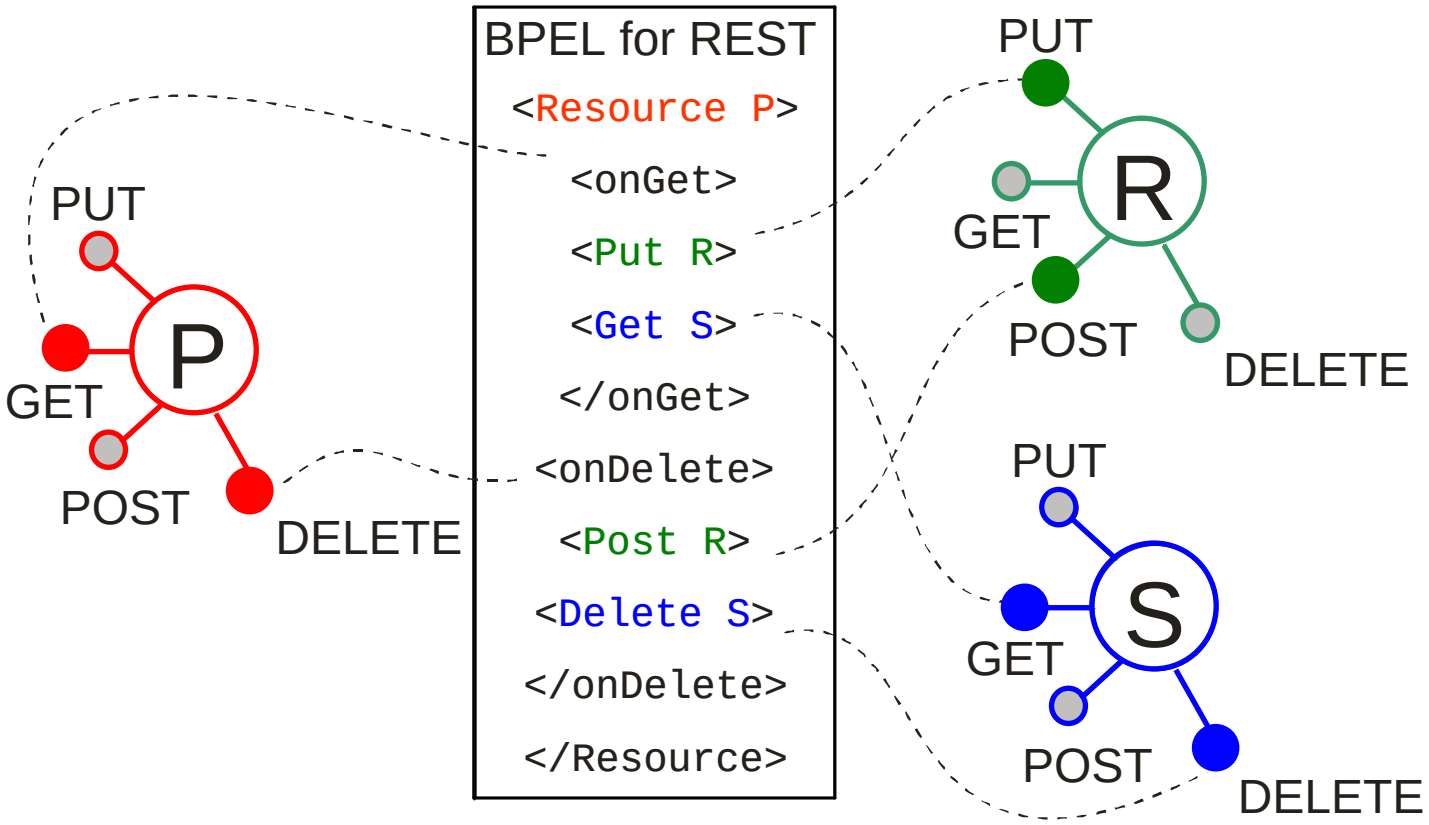
BPEL for REST

Make REST interaction primitives first-class language constructs of BPEL



BPEL for REST – Resource Block

- Dynamically publish resources from BPEL processes and handle client requests



Measuring Complexity

- Why is REST perceived to be simpler?
- Architectural Decisions give a **quantitative measure** of the complexity of an architectural design space:
  - Total number of decisions
  - For each decision, number of alternative options
  - For each alternative option, estimate the effort

	REST	WS-*
Decisions	17	14
Alternatives	27	35

Decisions with *1 or more* alternative options

Measuring Complexity

	REST	WS-*
Decisions	5	12
Alternatives	16	32

Decisions with *more than 1* alternative options

	REST	WS-*
Decisions	17	14
Alternatives	27	35

Decisions with *1 or more* alternative options

Measuring Complexity

	REST	WS-*
Decisions	5	12
Alternatives	16	32

Decisions with *more than 1* alternative options

- URI Design
- Resource Interaction Semantics
- Payload Format
- Service Description
- Service Composition

Measuring Complexity

	REST	WS-*
Decisions	5	12
Alternatives	16	32

Decisions with *more than 1* alternative options

	REST	WS-*
Decisions	12	2

Decisions with *only 1* alternative option

Measuring Complexity

- Payload Format
- Data Representation Modeling

	REST	WS-*
Decisions	12	2

Decisions with *only 1* alternative option

Measuring Effort

	REST	WS-*
Do-it-yourself Alternatives	5	0

Decisions with **only** *do-it-yourself* alternatives

	REST	WS-*
Decisions	12	2

Decisions with *only 1* alternative option

Measuring Effort

	REST	WS-*
Do-it-yourself Alternatives	5	0

Decisions with **only** *do-it-yourself* alternatives

- Resource Identification
- Resource Relationship

- Reliability
- Transactions
- Service Discovery

Freedom of Choice

Freedom from Choice

Architectural Decision and AAs	REST	WS-*
Integration Style	1 AA	2 AAs
Shared Database		
File Transfer		
Remote Procedure Call	✓	✓
Messaging		✓
Contract Design	1 AA	2 AAs
Contract-first		✓
Contract-last		✓
Contract-less	✓	
Resource Identification	1 AA	n/a
Do-it-yourself	✓	
URI Design	2 AA	n/a
"Nice" URI scheme	✓	
No URI scheme	✓	
Resource Interaction Semantics	2 AAs	n/a
Lo-REST (`POST`, `GET` only)	✓	
Hi-REST (4 verbs)	✓	
Resource Relationships	1 AA	n/a
Do-it-yourself	✓	
Data Representation/Modeling	1 AA	1 AA
XML Schema	(✓)[a]	✓
Do-it-yourself	✓	
Message Exchange Patterns	1 AA	2 AAs
Request-Response	✓	✓
One-Way		✓
Service Operations Enumeration	n/a	≥3 AAs
By functional domain		✓
By non-functional properties and QoS		✓
By organizational criterion (versioning)		✓
Total Number of Decisions, AAs	**8**, 10	**5**, ≥10

[a] Optional

Table 2: Conceptual Comparison Summary

Architectural Decision and AAs	REST	WS-*
Transport Protocol	1 AA	≥7 AAs
HTTP	✓	✓[a]
waka [13]	(✓)[b]	
TCP		✓
SMTP		✓
JMS		✓
MQ		✓
BEEP		✓
IIOP		✓
Payload Format	≥6 AAs	1 AA
XML (SOAP)	✓	✓
XML (POX)	✓	
XML (RSS)	✓	
JSON [10]	✓	
YAML	✓	
MIME	✓	
Service Identification	1 AA	2 AA
URI	✓	✓
WS-Addressing		✓
Service Description	3 AAs	2 AAs
Textual Documentation	✓	
XML Schema	(✓)[c]	✓
WSDL	✓[d]	✓
WADL [18]	✓	
Reliability	1 AA	4 AAs
HTTPR [38][e]	(✓)	(✓)
WS-Reliability		✓
WS-ReliableMessaging		✓
Native		✓
Do-it-yourself	✓	✓
Security	1 AA	2 AAs
HTTPS	✓	✓
WS-Security		✓

Transactions	1 AA	3 AAs
WS-AT, WS-BA		✓
WS-CAF		✓
Do-it-yourself	✓	✓
Service Composition	2 AAs	2 AAs
WS-BPEL		✓
Mashups	✓	
Do-it-yourself	✓	✓
Service Discovery	1 AAs	2 AAs
UDDI		✓
Do-it-yourself	✓	✓
Implementation Technology	many	many
...	✓	✓
Total Number of Decisions, AAs	**10**, ≥17	**10**, ≥25

[a] Limited to only the verb `POST`
[b] Still under development
[c] Optional
[d] WSDL 2.0
[e] Not standard

Table 3: Technology Comparison Summary

Architectural Principle and Aspects	REST	WS-*
Protocol Layering	yes	yes
HTTP as application-level protocol	✓	
HTTP as transport-level protocol		✓
Dealing with Heterogeneity	yes	yes
Browser Wars	✓	
Enterprise Computing Middleware		✓
Loose Coupling, aspects covered	yes, 2	yes, 3
Time/Availability		✓
Location (Dynamic Late Binding)	(✓)	✓
Service Evolution:		
Uniform Interface	✓	
XML Extensibility	✓	✓
Total Principles Supported	**3**	**3**

Table 1: Principles Comparison Summary

Comparison Summary

- Architectural Decisions measure complexity implied by alternative technologies

- **REST simplicity** = freedom from choice
  - 5 decisions require to choose among 16 alternatives
  - 12 decisions are already taken (but 5 are *do-it-yourself*)

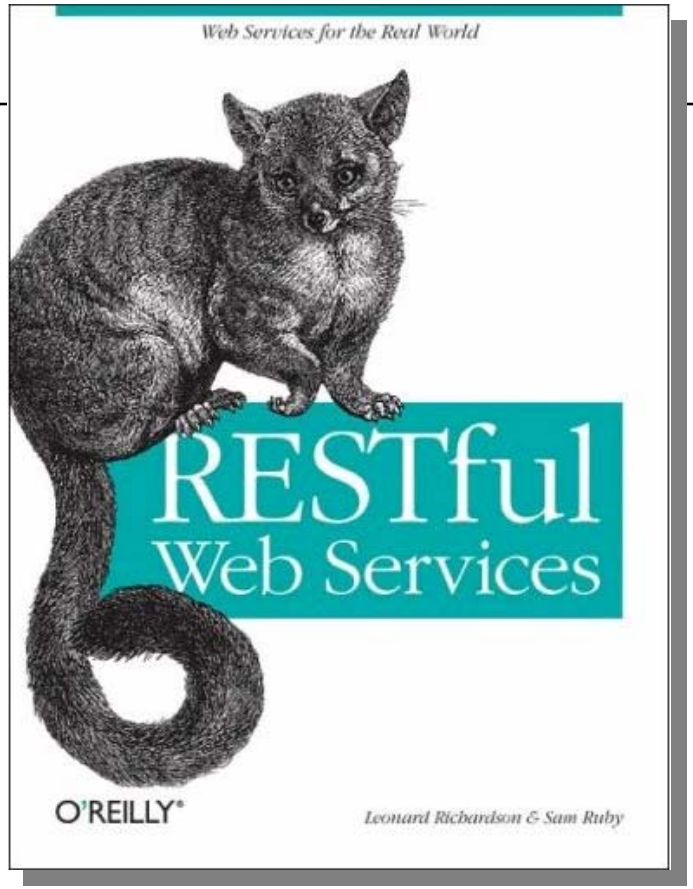
- **WS-* complexity** = freedom of choice
  - 12 decisions require to choose among 32 alternatives
  - 2 decisions are already taken (SOAP, WSDL+XSD)

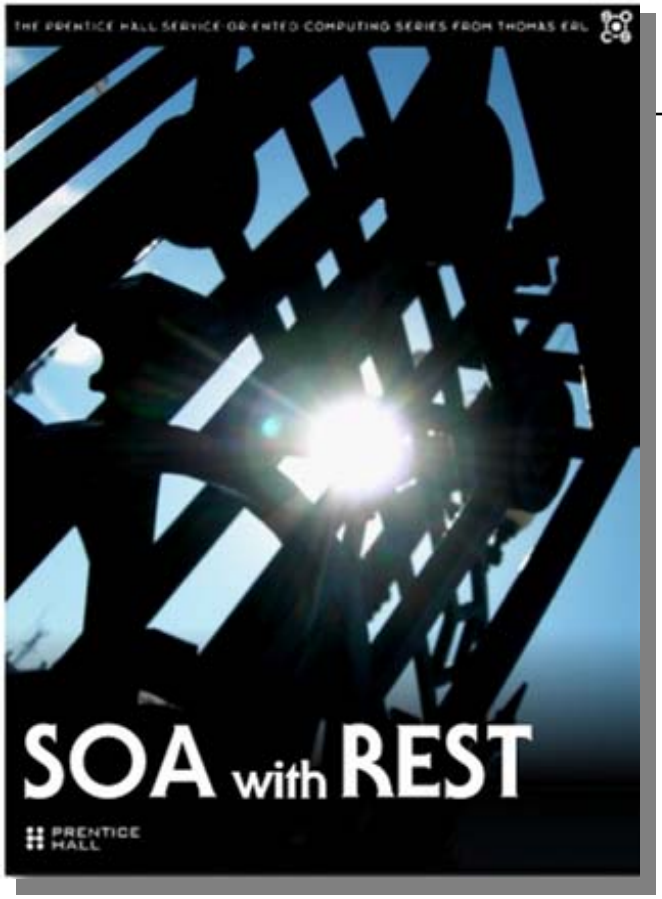
Comparison Conclusion

- You should focus on whatever solution gets the job done and try to **avoid being religious** about any specific architectures or technologies.
- WS-* has strengths and weaknesses and will be highly suitable to some applications and positively terrible for others.
- Likewise with REST.
- The decision of which to use depends entirely on the application requirements and constraints.
- We hope this comparison will help you make the right choice.

References

- Cesare Pautasso, Olaf Zimmermann, Frank Leymann, RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision, Proc. of the 17th International World Wide Web Conference (WWW2008), Bejing, China, April 2008.
- Cesare Pautasso and Erik Wilde. Why is the Web Loosely Coupled? A Multi-Faceted Metric for Service Design, Proc of the 18th International World Wide Web Conference (WWW2009), Madrid, Spain, April 2009.
- Cesare Pautasso, BPEL for REST, Proc. of the 6th International Conference on Business Process Management (BPM 2008), Milan, Italy, September 2008.
- Cesare Pautasso, Gustavo Alonso: **From Web Service Composition to Megaprogramming** In: Proceedings of the 5th VLDB Workshop on Technologies for E-Services (TES-04), Toronto, Canada, August 2004.





Leonard Richardson, Sam Ruby,
RESTful Web Services,
O'Reilly, May 2007

Raj Balasubramanians, Benjamin Carlyle,Thomas Erl, Cesare Pautasso,
SOA with REST,
Prentice Hall, End of 2009