Web Services
and
The REST Architectural Style

© Riccardo Sisto, Politecnico di Torino

Reference for study: L. Richardson, S. Ruby - "RESTful Web Services", O'Reilly, 2007

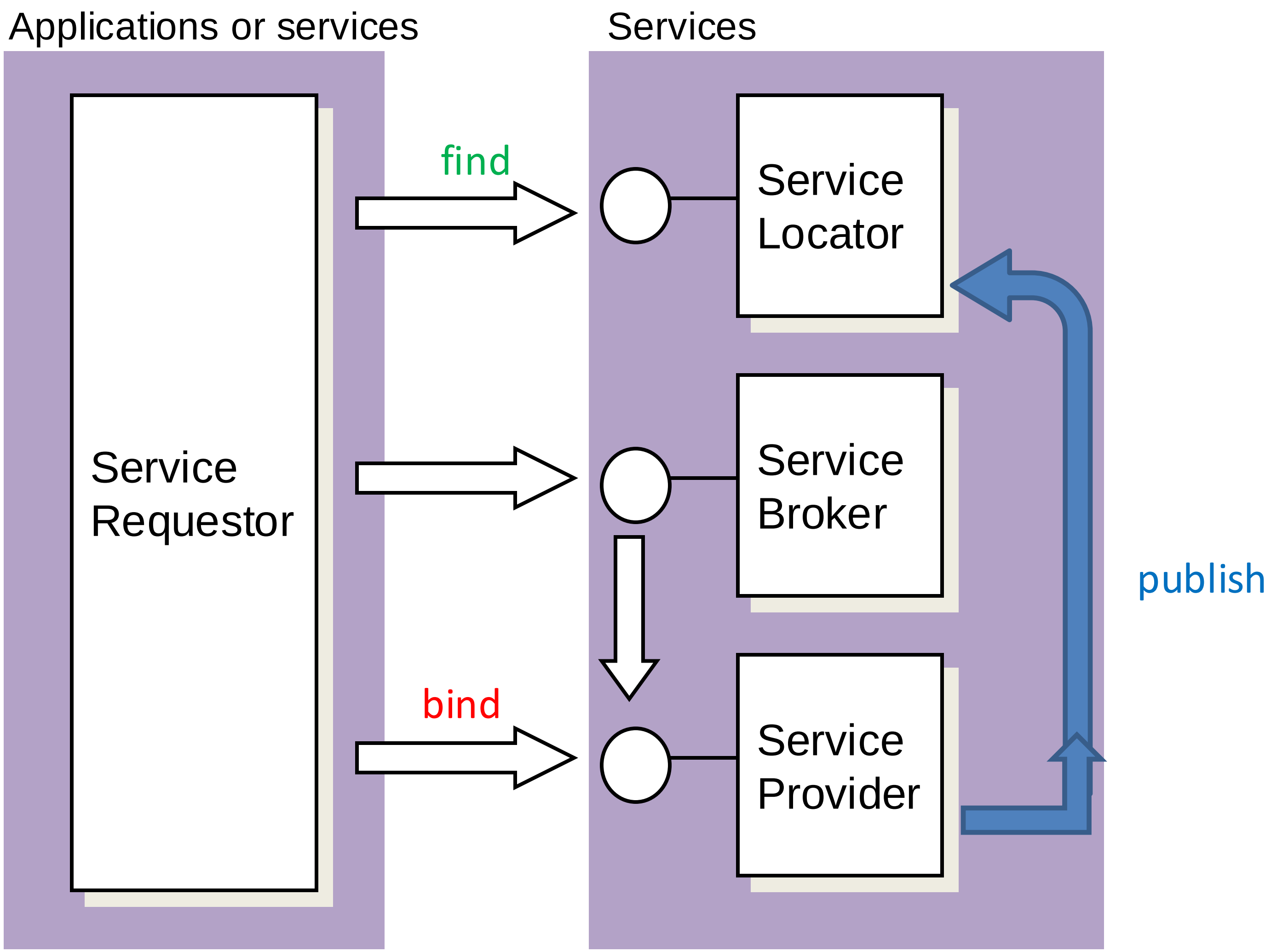
Distributed Services

- Abstraction similar to distributed objects
- Main differences:
  - Services have **larger grain** than objects
  - Services are **autonomous** and **long-living** entities
  - Services may be made available for use by different clients from **different organizations**
  - Services enable **service composition**

Service Oriented Architecture (SOA)

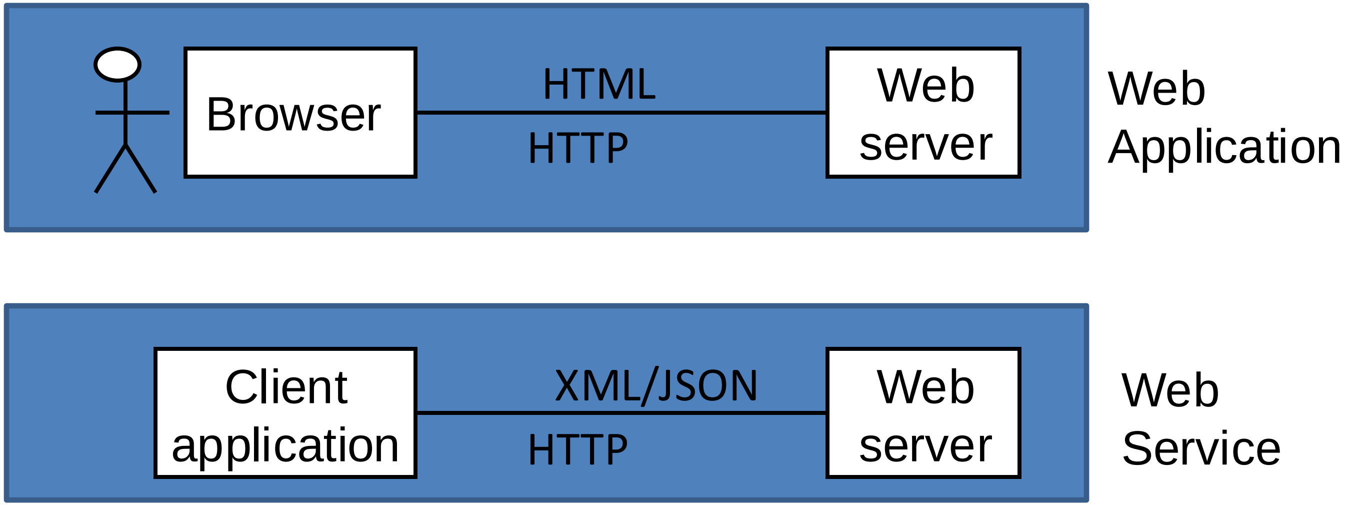
- Software organized as a set of **services**
- Services provided through **interfaces** that are
  - Published and Automatically discoverable
  - Machine readable
- **Service**:
  - described by a contract, made of one or more interfaces
  - implemented by a **single instance,** always available
  - **coarse grained**
  - **loosely coupled**
    - interactions by (typically asynchronous) message exchanges

SOA


Applications or services
Services
find
Service Locator
Service Requestor
Service Broker
publish
bind
Service Provider

Web Services (Web APIs)

- Distributed Services based on the web (HTTP)



- Different flavors:
  - SOA-based (standard or SOAP) web services
  - RESTful web services

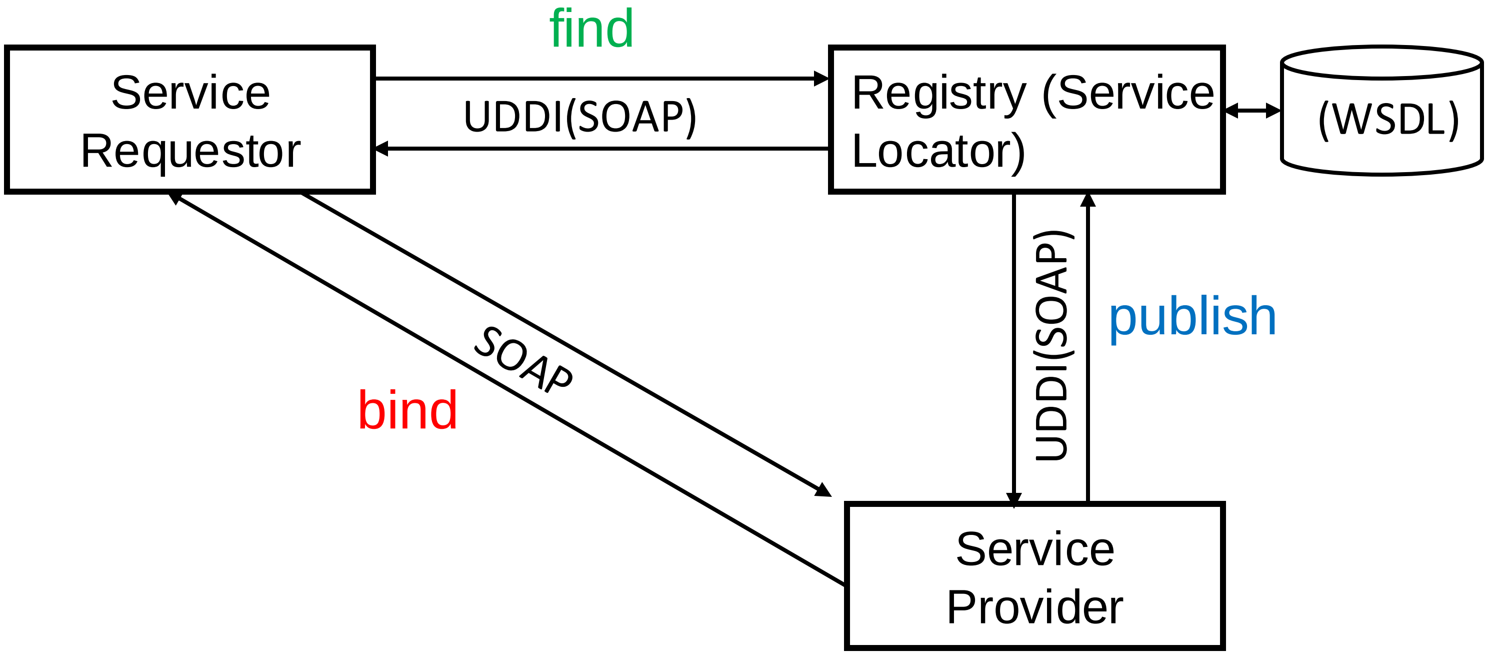
Web Service Scenarios

- Software components, made available through the network *as web services* (SaaS)
  - A software vendor sells the **usage** of software components and manages internally its maintenance
- Whole applications (for example a whole information system process) made available as web services
- Integration of services in order to create added-value new services
  - e.g.: a travel booking service based on hotel booking services, flight and train booking services, etc.
- Web services marketplace

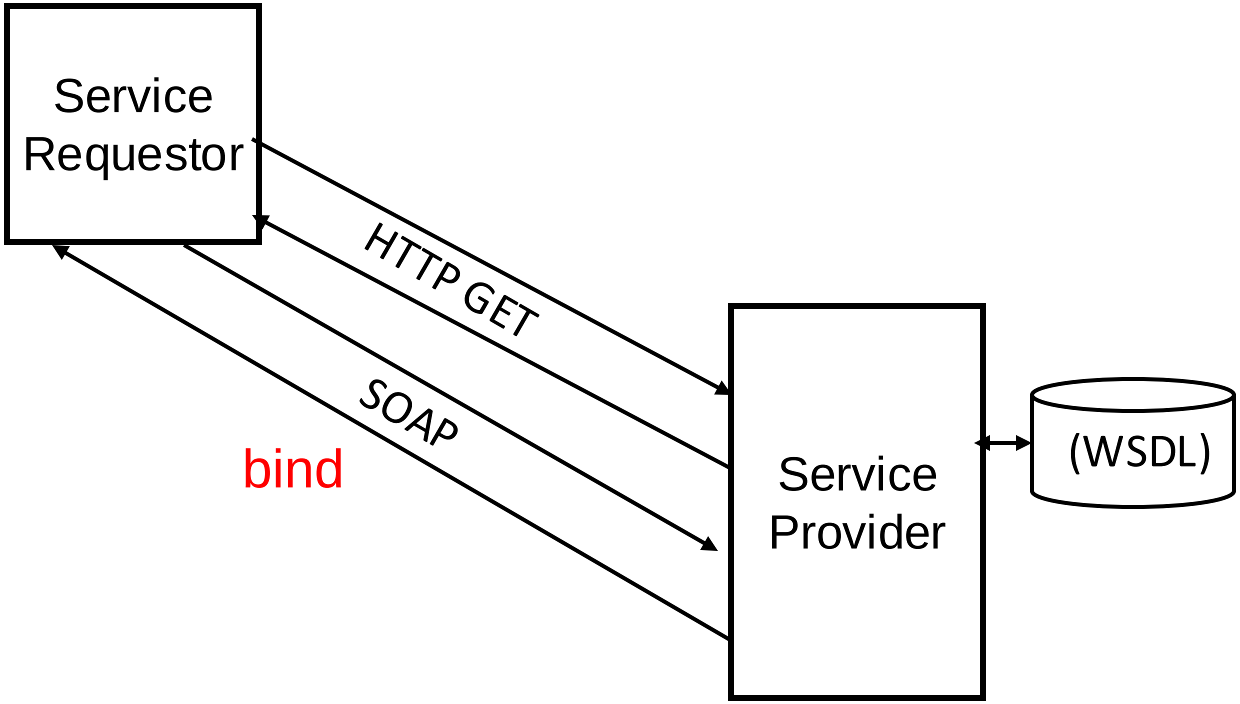
WS Life Cycle

- Build (service is designed and implemented)
  - Design and specification of services and interfaces
  - Service implementation (or simply linkage to existing implementations)
- Deployment (service is installed)
  - Installation in the target run-time environment
  - Publishing
- Management (service is maintained)
- Decommissioning (service is dismissed)

SOA-based Web Services (SOAP/Standard Web Services)

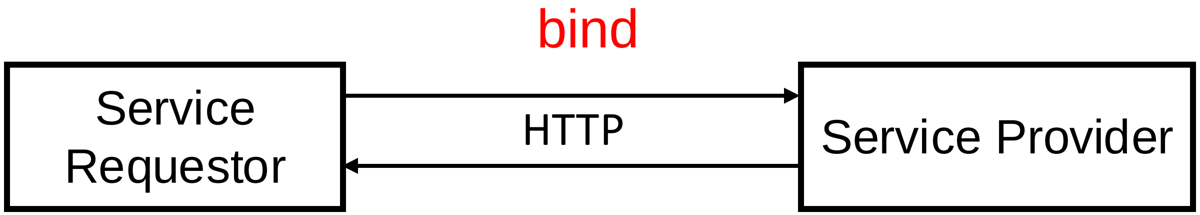


SOAP/Standard Web Services Actual Simplified Model



RESTful Web Services

- Web Services based on the REST Architectural Style



bind

Service Requestor

HTTP

Service Provider

The REST Architectural Style

REpresentational State Transfer

- REST is a way for architecting distributed systems (architectural style: set of constraints)
  - Introduced by Roy Fielding (the main designer of HTTP 1.1)
  - Implemented by the HTTP protocol and the web (the concrete REST architecture)

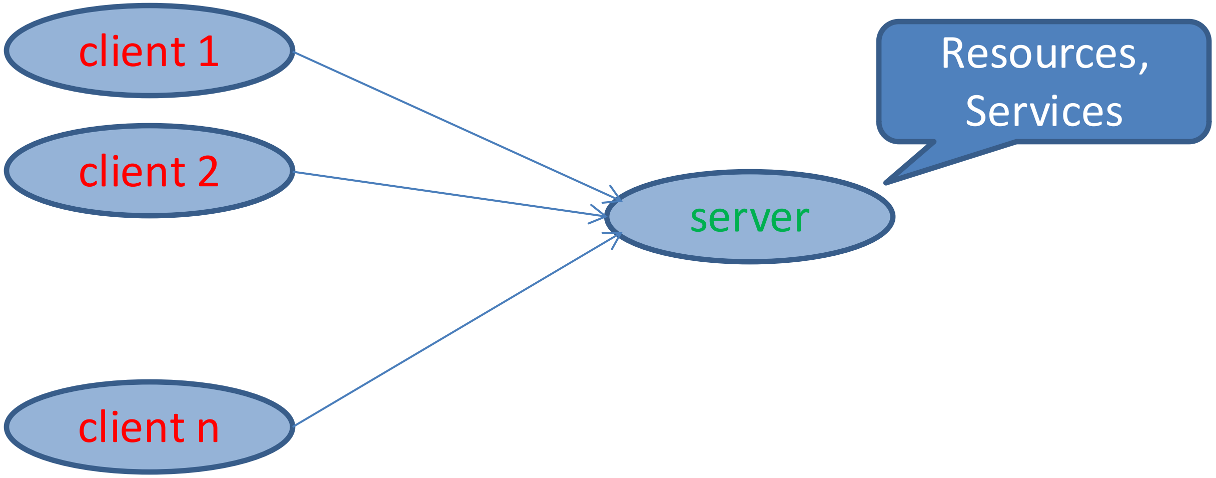
- Documentation:
  - R. Fielding, Architectural Styles and the Design of Network-based Software Architectures, PhD Thesis, 2000, Chapter 5
    https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
  - HTTP Semantics, RFC 9110, 2022
    https://tools.ietf.org/html/rfc9110
  - L. Richardson, S. Ruby "RESTful Web Services", O'Reilly, 2007

The REST Constraints

- REST is based on a typed request/response messaging protocol (like HTTP) that is:
  - Client/server
  - Stateless
  - Cacheable
  - Layered
  - With a fixed (uniform) interface

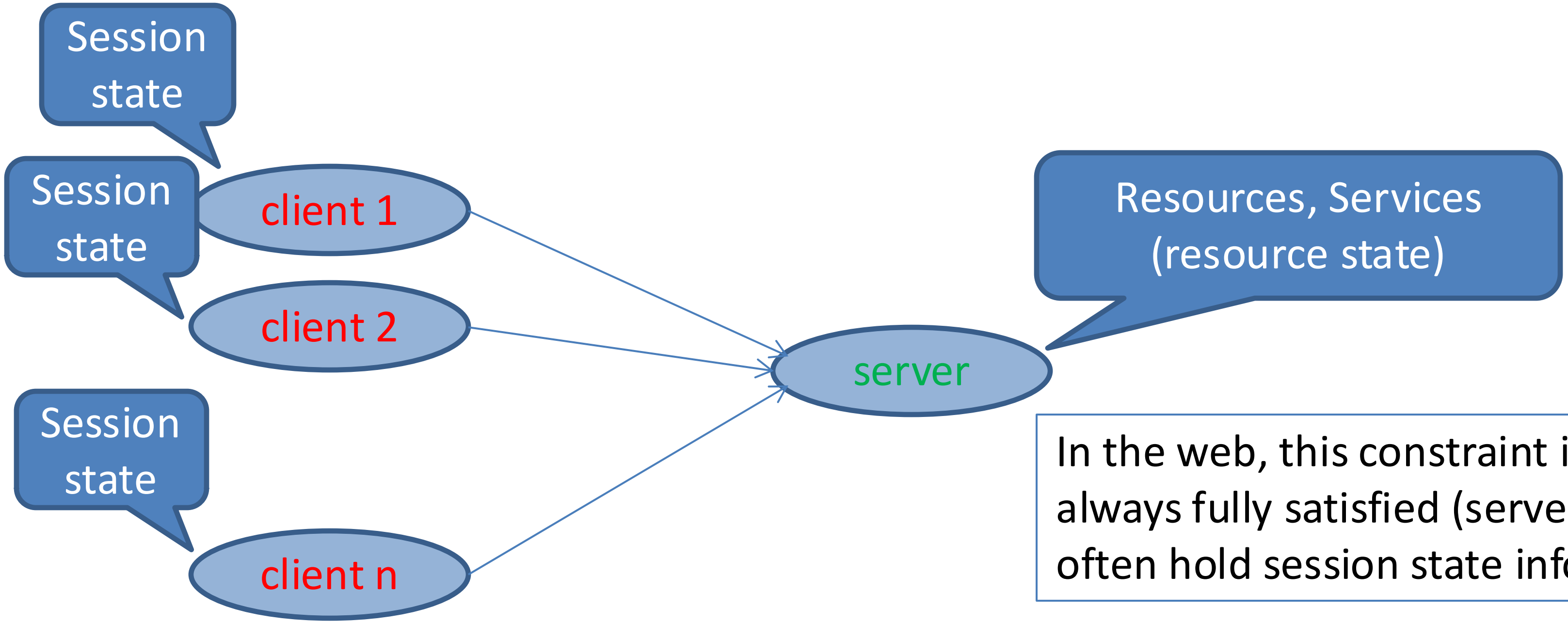
Client-Server

- Separation of roles
  - Simplicity
  - Helps for scalability and security
    - Servers and clients can be kept simple and evolve independently (even in different organizational domains)



Stateless Interaction Protocol

- Session state entirely on the client
  - Improves visibility and monitoring
  - Improves reliability (simplifies recovery from partial failures)
  - Helps with scalability (servers don't need to store session info)



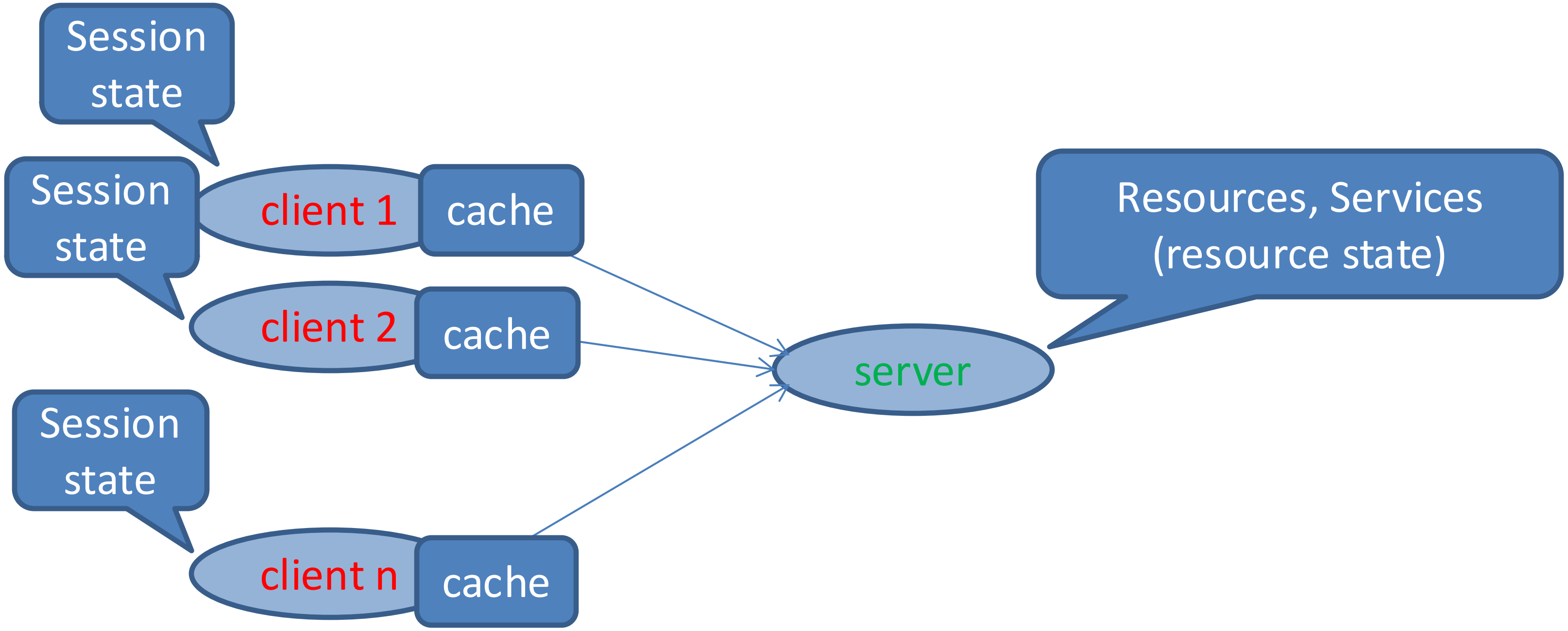
In the web, this constraint is not always fully satisfied (servers often hold session state info)

Protocol Enables Caching

- Responses can be labeled as cacheable or non-cacheable
- Cacheable responses can be cached by clients

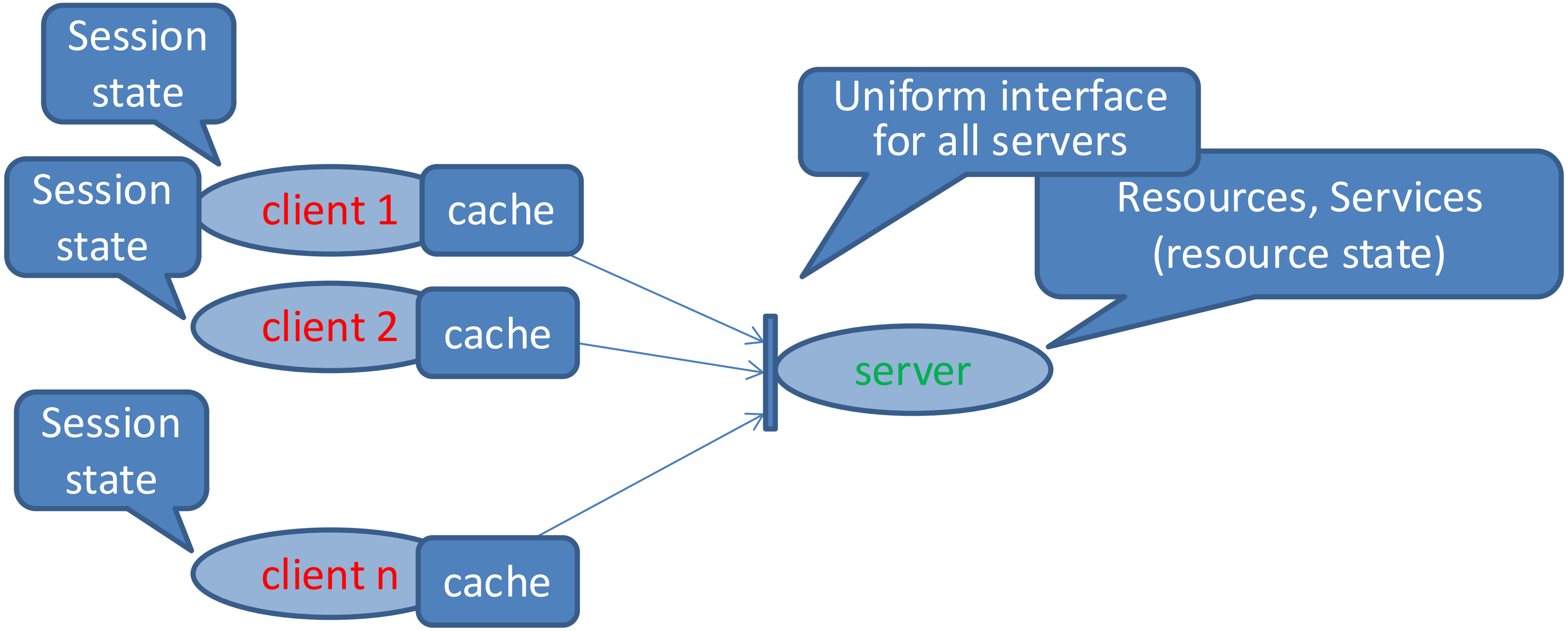
=> Avoids some interactions

- Improves network efficiency, scalability, user-perceived performance (latency)



Fixed (Uniform) Interface

- Obtained by setting a number of additional constraints on how the interface must be organized (more on this later on)
- Leads to improved simplicity and observability/openness

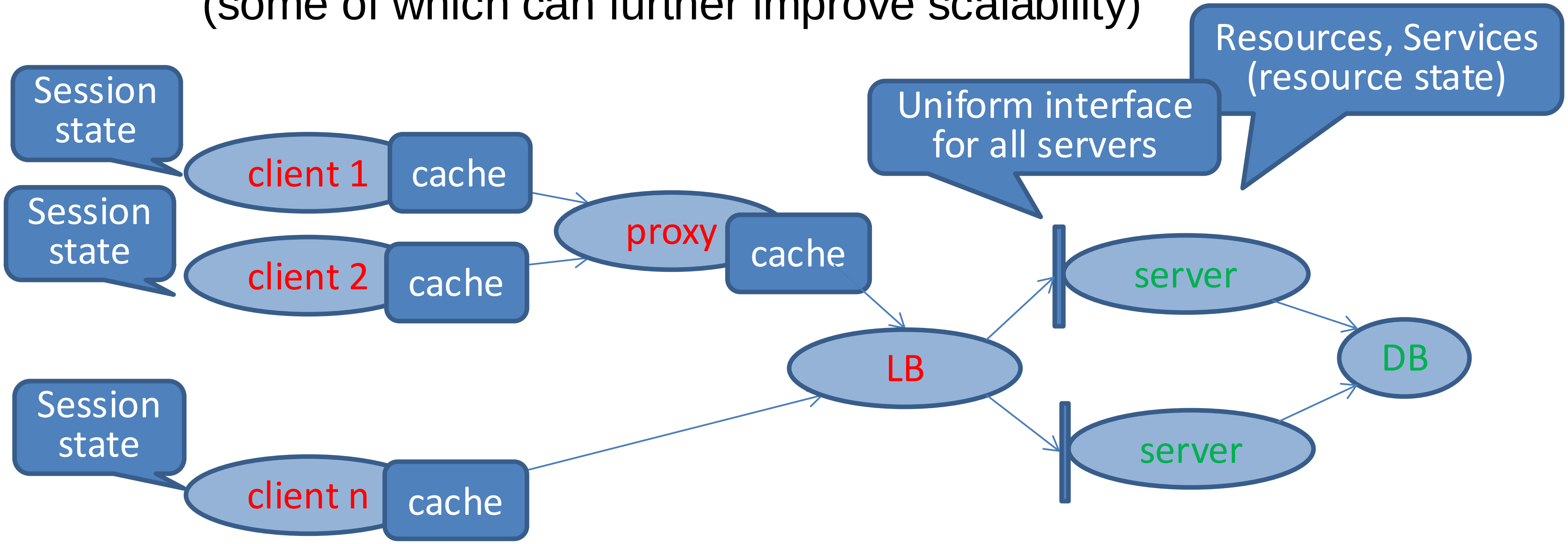


Layered Topology

- Corresponds to enabling multi-tier architectures
  - vertical layering (each layer interacts only with adjacent layers)

  => Improved simplicity

  => Possibility to dynamically and simply add functions (some of which can further improve scalability)

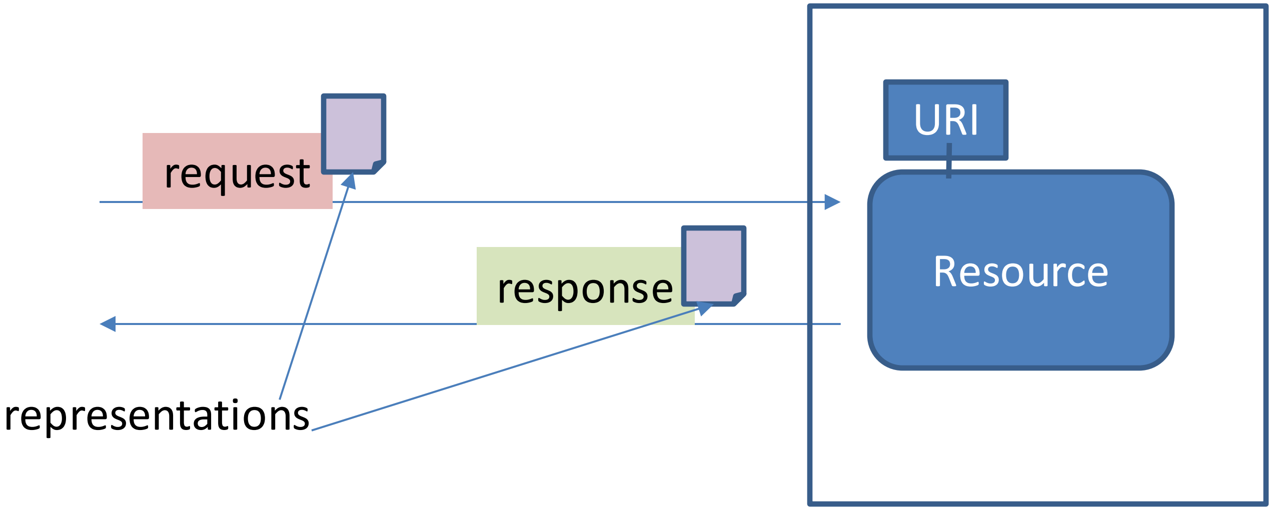


Uniform Interface: Details Resources

- Server interfaces are resource-oriented
  - Key abstraction: resource
    - A concept
    - Information item with time-varying state
    - Identified (URI)
    - Zero or more representations (of different media types)
    - Fixed set of possible operations (resource methods)
  - Protocol enables transfer of representations with metadata
  - Resource example: the Turin Polytechnic, URI http://www.polito.it
    - Possible representations:
      - The Turin Polytechnic html home page (content-type: text/html)
      - a jpeg picture of the Turin Polytechnic (content-type: image/jpeg)

How do Resources work?

- When a client requests an operation on a resource, a *representation of the resource* (current/past/future) may be transferred
- Resources themselves are never transferred. They stay always in the server side, where their URI points to.



Operations on Resources

- The protocol (HTTP) offers a fixed (uniform) interface for acting on resources
- The semantics of each requested operation is pre-defined by the protocol. With HTTP, it depends on
  - the requested method
  - the request body
  - the request headers (control metadata)
  - the target URL (including any query string)
- The result of the operation is communicated in the response (via status code, response body and other response headers)

Operations on Resources

- The fixed interface offered by HTTP basically provides the CRUD (Create, Read, Update, Delete) operations (methods) defined by REST:

CRUD Operation	HTTP request	HTTP response (if no error)
Create a new resource under *res* (or send data to *res* for resource-specific processing)	POST *res* (resource or data representation in the body)	URL of new resource created and/or result of processing (in body)
Read resource *res*	GET *res* (no body)	representation of current state of *res* (in the body)
Update resource *res* by replacing its state with a new one (*res* can be created)	PUT *res* (new resource representation in the body)	description of executed operation (200 or 201 or 204 status code)
Delete resource *res*	DELETE *res*	

Operation parameters and return values

- POST and PUT requests carry parameters in their body
- GET and DELETE requests have no body (only headers) => they carry parameters in their query string
- Responses carry return value in their body
- HTTP status codes report exit information
  - Examples:
    - 200 OK
    - 201 Created
    - 304 Not modified
    - 400 Bad request
    - 500 Internal server error

For full description of allowed parameters and responses see the HTTP standard

Other Operations

Operation	HTTP request	HTTP response (if no error)
Read information about resource *res* without transferring its representation	HEAD *res* (no body)	same as for GET but without body (only headers sent)
Read supported methods for a resource	OPTIONS *res* (no body)	list of supported methods (in the Allow header and possibly in the body)
Test the connection to resource *res* (loopback testing similar to echo)	TRACE *res*	the received request (in the body)
Partially Update the resource *res* by applying a patch	PATCH *res* (patch to be applied to *res* in the body)	description of executed operation (200 or 201 or 204 status code)

Implied Features of HTTP methods

HTTP method	CRUD operations	idempotent	Safe	Request body	Cacheable response
POST	C	no	no	yes	(no)
GET	R	yes	yes	no	yes
PUT	U, C	yes	no	yes	no
DELETE	D	yes	no	no	no
HEAD	R	yes	yes	no	yes
OPTIONS	R	yes	yes	no	no
TRACE	-	yes	yes	yes	no
PATCH	U, C	no	no	yes	no

- For a full description of HTTP methods semantics, see the HTTP standard (https://tools.ietf.org/html/rfc9110) and the PATCH RFC (https://tools.ietf.org/html/rfc5789)

Example: A Collection of Items

- courses A resource that represents a collection of courses
- courses/xx A resource that represents a single course

Method URI pair	Meaning
POST courses	Create a new course with given representation, and add it to the collection as a new resource under courses. Return its URI
GET courses	Read the whole collection of courses
GET courses/xx	Read only course xx in the collection
GET courses?year=1	Read only the courses of the first year in the collection
PUT courses	Update the whole collection with the given representation
PUT courses/xx	Update only course xx in the collection with the given representation
DELETE courses	Delete the whole collection of courses
DELETE courses/xx	Delete only course xx in the collection

Expressiveness

- The REST interface is powerful enough to represent any intended set of operations
- => RESTful web services are semantically as expressive as SOAP web services
- Style is opposite
  - REST: few fixed operations, many resources (constrained)
  - SOAP: few fixed resources, many operations

Example: A Distributed Counter as a RESTful Web Service

- A single fixed resource *count*
- Admitted requests:

Request	Meaning
GET *count*	Read the value of the counter (value operation)
PUT *count* with 0 as body	Reset the counter to 0 (zero operation)
POST *count* with value as body	Increment the counter by value (increment operation)

Other Example: The Pizza Shop Service

- Customers can read the menu (pizza types and available toppings)
- Customers can submit pizza orders
  - pizza types, quantities, toppings,...
- Pizza orders are processed by pizza makers

The Pizza Shop Service

- Resources:

– menu	the menu
– submittedOrders	the orders that have been submitted but not yet consumed
– consumedOrders	the already consumed orders

- Admitted Requests:

Request	Meaning
GET *menu*	Read the menu(getMenu operation)
POST *submittedOrders* with Order element as body	Submit an order (submit operation) Side effect: order is added to submittedOrders
POST *consumedOrders* with empty body (response includes Order)	Consume next order (getNextOrder operation) Side effect: oldest order is moved from submittedOrders to consumedOrders

Machine-Readable Description of RESTful web services

- No globally accepted standard yet.
- Main Initiatives:
  - WADL (Web Application Description Language)
    - Description of resources
    - For each resource: admitted methods with related parameters
  - WSDL 2.0 (with HTTP binding)
    - Standard but more limited (e.g. no description of resources)
    - Not fully REST-compliant
  - SWAGGER (OpenAPI Specification)
    - Complete description (resources, operations, parameters, types)
    - Supported by all major SW vendors, is becoming a de-facto standard
  - Google API Discovery Documents

W3C

HATEOAS (Hypermedia As The Engine Of Application State)

- Ideally, a RESTful web service should operate like a traditional web application
  - stateless interactions (no client-related state stored in the server, only resource states)
  - state information represented by hyperlinks (conveyed in requests and responses) and stored in the client
  - state changes occur by following hyperlinks
    - clients never "build" hyperlinks, they just follow the ones included in responses
    - a client is expected to have just the URL of the main resource
  - applications are self-describing
    - however, the difference is that for web services the description must be machine-intelligible

Richardson Maturity Model

- Ranking web services about how they are compliant to the REST constraints
  - Level 0 No REST
    - single resource, HTTP as transport ("tunneling")
  - Level 1 Resources
    - multiple resources corresponding to different application entities
  - Level 2 HTTP verbs
    - operations represented only by means of HTTP verbs
  - Level 3 Hypermedia controls
    - hyperlinks included in responses and self-describing features (e.g. options)