SoftServe

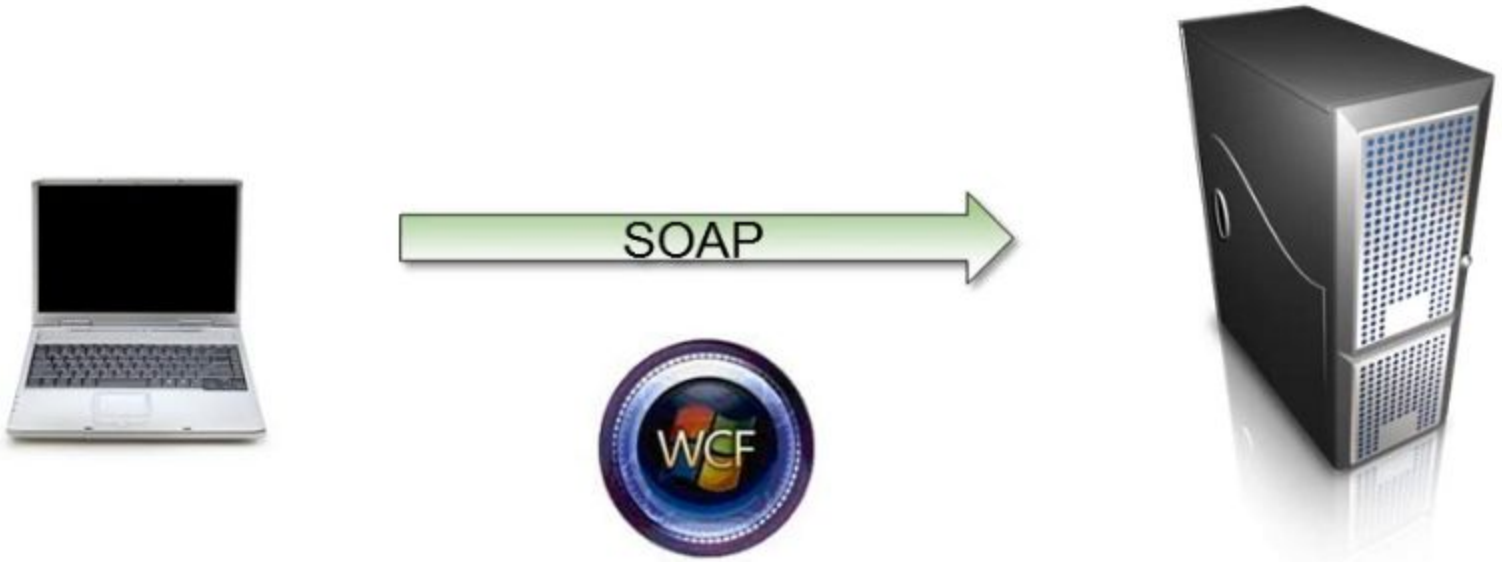
Empowering your Business
through Software Development

RestFul Services. Part 2.

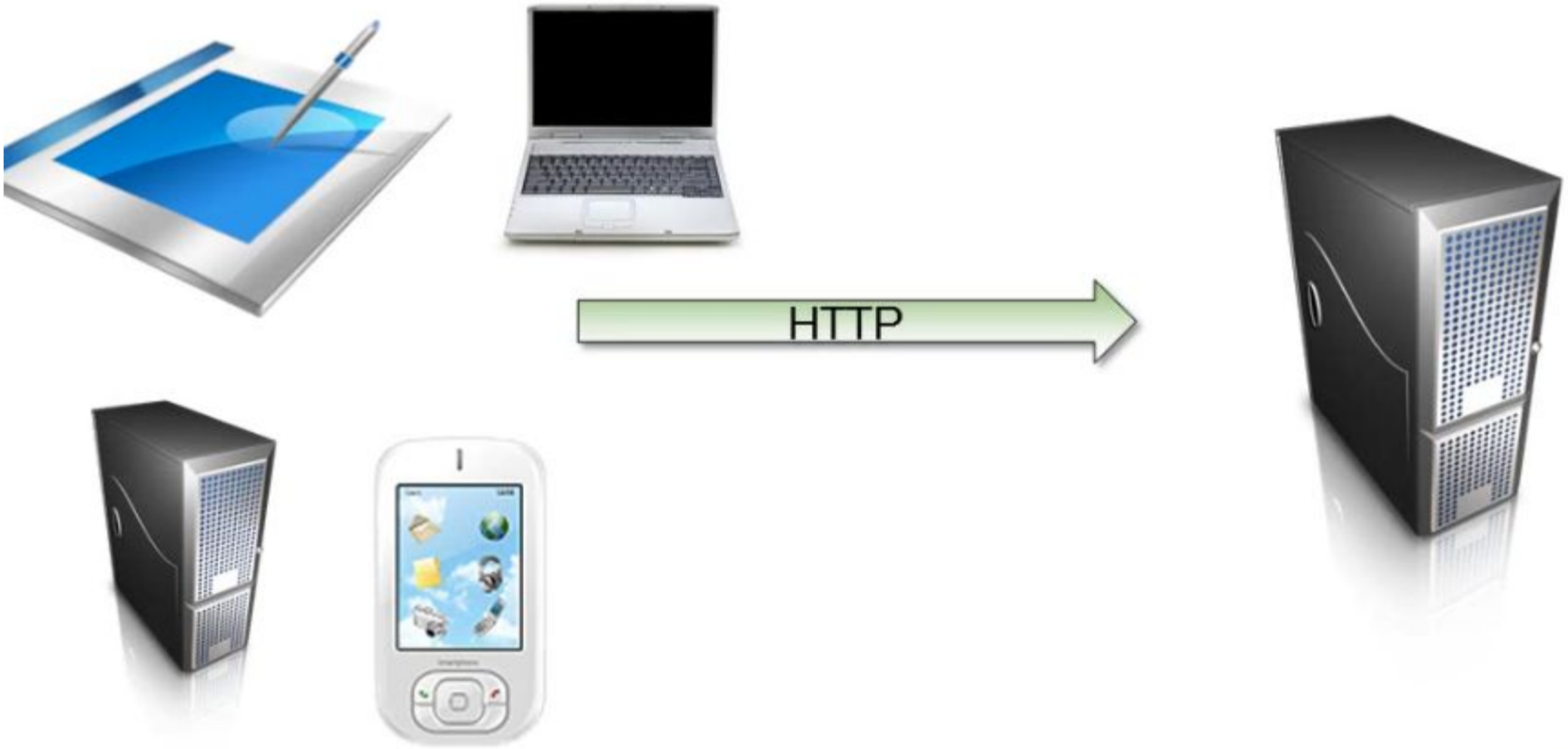
Agenda

- What is Rest? Conceptual overview
- Restful Web services
- SOAP vs REST
- High-level example: hotel booking
- Technologies
- Using HTTP to build REST Applications

Web Services



HTTP Web Services



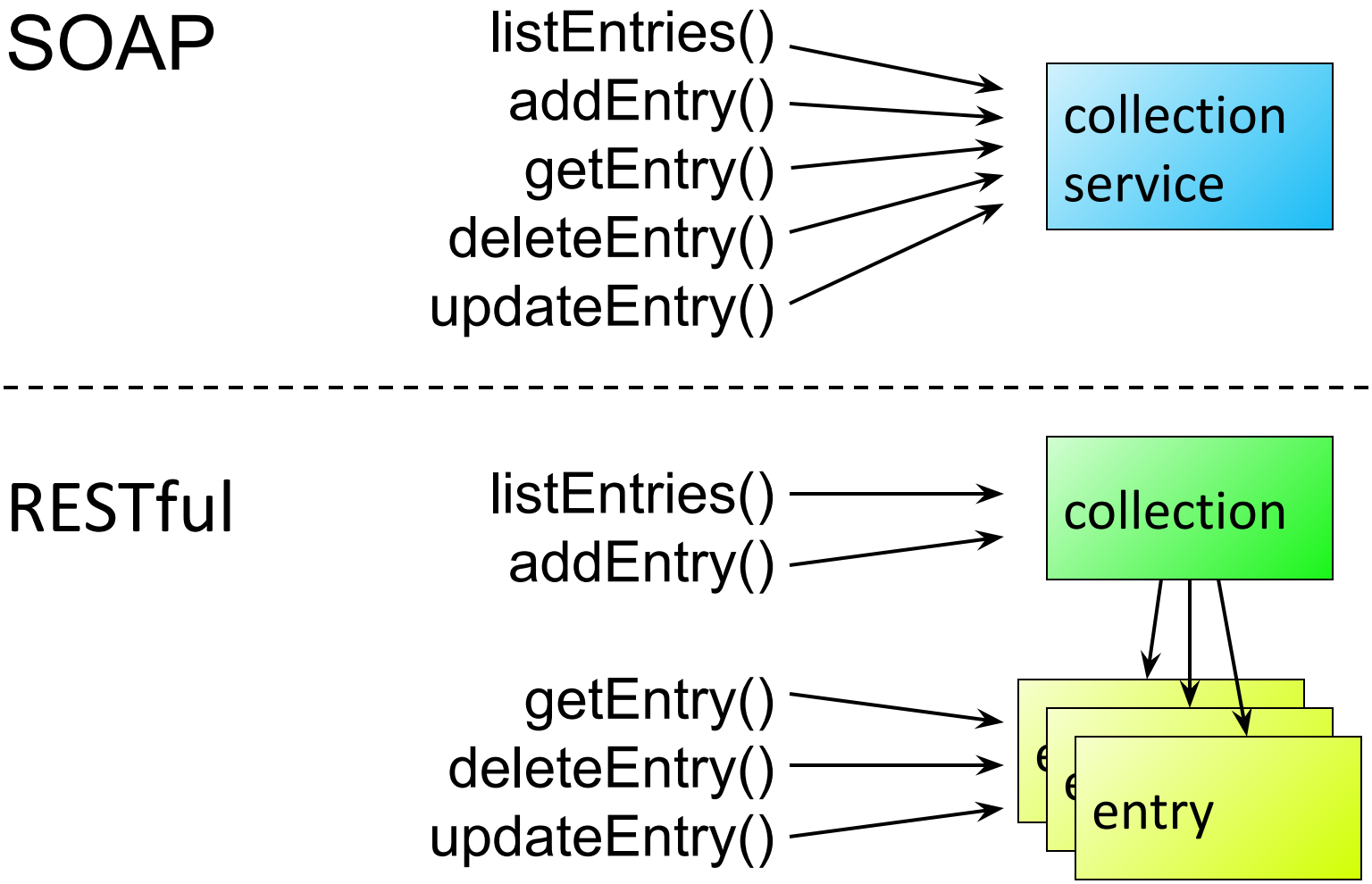
Conceptual Overview
Representational State Transfer (REST)

- **Representational State Transfer (REST)**
  - A style of software architecture for distributed hypermedia systems such as the World Wide Web.
- **REST is basically client/server architectural style**
  - Requests and responses are built around the transfer of "representations" of "resources".
- **REST is centered around two basic principles:**
  - ***Resources* as URLs**. A resource is something like a "business entity" in modelling lingo. It's an entity you wish to expose as part of an API. Almost always, the entity is a noun, e.g. a person, a car, or a football match. Each resource is represented as a unique URL.
  - ***Operations* as HTTP methods.** REST leverages the existing HTTP methods, particularly GET, POST, PUT, and DELETE.

RESTful Web Service definition

- **A RESTful Web service is:**
  - A set of Web resources.
  - Interlinked.
  - Data-centric, not functionality-centric.
  - Machine-oriented.
- **Like Web applications, but for machines.**
- **Like SOAP, but with more Web resources.**

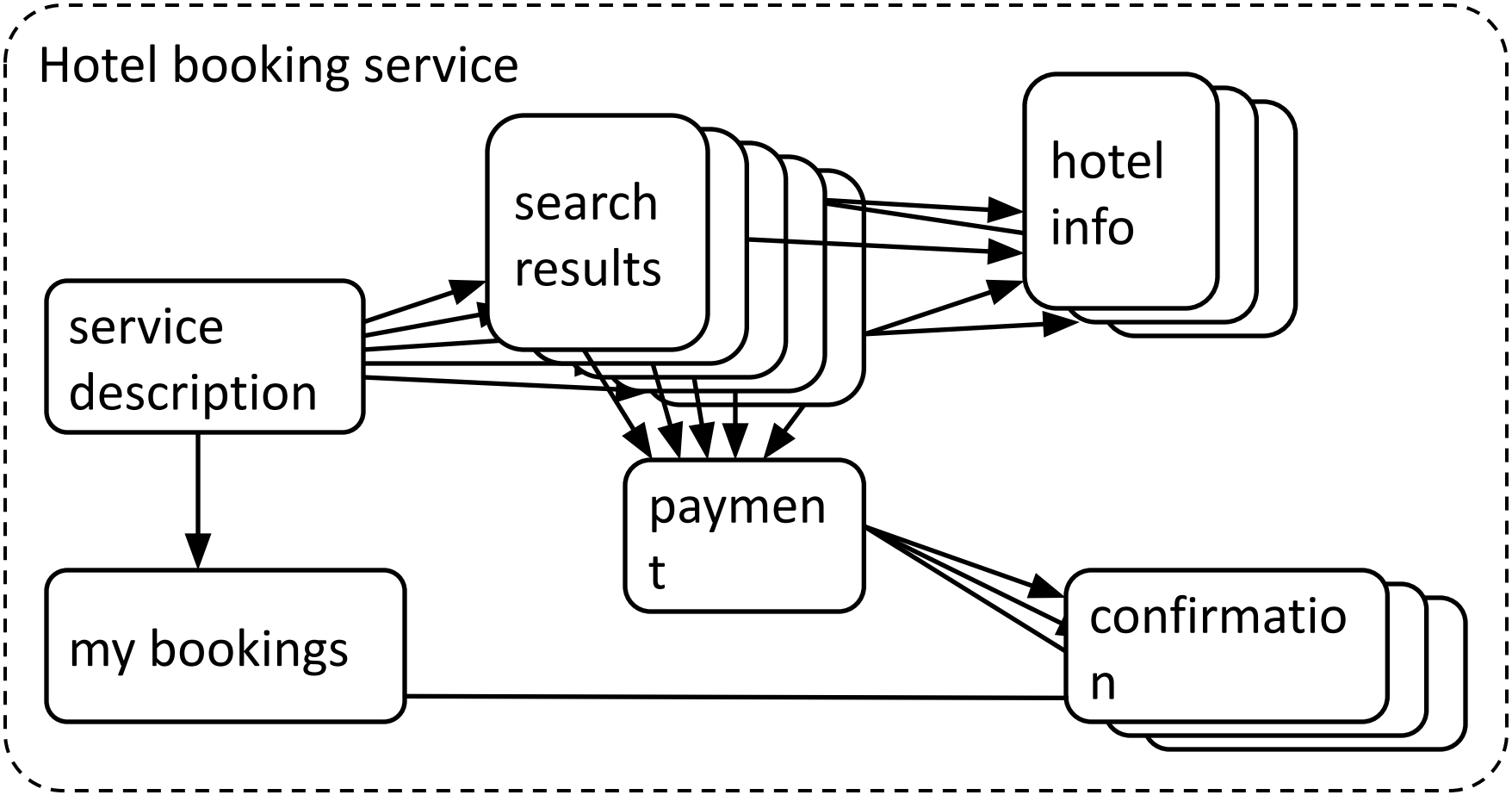
SOAP vs REST: A quick comparison



SOAP vs REST: A quick comparison

- A SOAP service has a single endpoint that handles all the operations – therefore it has to have an application-specific interface.
- A RESTful service has a number of resources (the collection, each entry), so the operations can be distributed onto the resources and mapped to a small uniform set of operations.

High-level example: hotel booking



High-level example: hotel booking

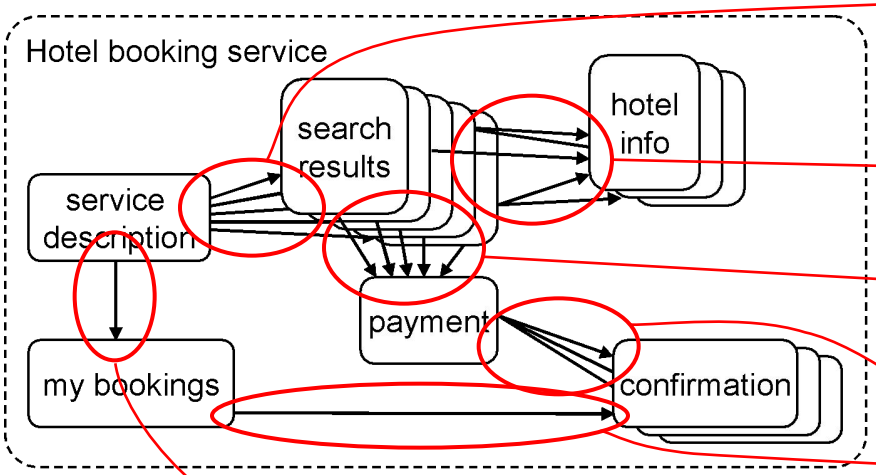
Hotel booking workflow

1. Retrieve service description
2. Submit search criteria according to description
3. Retrieve linked details of interesting hotels
4. Submit payment details according to selected rate description
5. Retrieve confirmation of booking

2b. Retrieve list of user's bookings

High-level example: hotel booking

hypermedia -> operations



search(date, city)

- list of hotels & rates

getHotelDetails(hotel)

- hotel details

reserve(rate, creditCard)

- confirmationID

getConfirmationDetails(confID)

- confirmation details

listMyBookings()

- list of confirmationIDs

nouns vs. verbs

RestFull Services. Technologies

- Todays's set of technologies, protocols and languages used to apply RESTful paradigm:
  - HTTP as the basis
  - XML and JSON for data exchange
  - AJAX for client-side programming (e.g. browser)

Using HTTP to build REST Applications

- The REST Recipe:
  - Find all the nouns,
  - Define the formats,
  - Pick the operations,
  - Highlight exceptional status codes.

Using HTTP to build REST Applications

- Find all the nouns:
  - Everything in a RESTful system is a resource – a noun.
  - Every resource has a URI,
  - URIs should be descriptive:
    - **http://example.com/expenses;pending**
    - Not a REST principle, but a good idea!
  - URIs should be opaque:
    - automated (non-human) clients should not infer meaning from a URI.

Using HTTP to build REST Applications

- Find all the nouns:
  - Use path variables to encode hierarchy:
    - **/expenses/123**
  - Use other punctuation to avoid implying hierarchy:
    - **/expenses/Q107;Q307**
    - **/expenses/lacey,peter**
  - Use query variables to imply inputs into an algorithm:
    - **/search?approved=false**
      - Caches tend to (wrongly) ignore URIs with query variables!
  - URI space is infinite (but URI length is not ~ 4K).

Using HTTP to build REST Applications

Resource	URI
Bill's expense reports	/users/bill/expenses
Expense report #123	/users/bill/expenses/123
All expense reports (you're allowed to see)	/expenses/
All pending (new, etc.) expense reports	/expenses;pending (new, etc.)
Bill's pending expense reports	/users/bill/expenses;pending
Expense 123's line items	/users/bill/expenses/123/line_items
Line item 2 of Expense 123	/users/bill/expenses/123/line_items/2
2006 expenses	/2006/expenses/
2006 open expenses	/2006/expenses;submitted

Using HTTP to build REST Applications

- Define the formats:
  - Neither HTTP nor REST mandate a single representation for data.
  - A resource may have multiple representations:
    - XML, JSON, binary (e.g., jpeg), name/value pairs
  - Schema languages are not required (if even possible).
  - Representations should be well-known media types (MIME types).
  - Try and use “up-stack” media types:
    - Makes your resources maximally accessible,
    - XHTML or Atom instead of vanilla XML.

Using HTTP to build REST Applications

- Pick the operations:
  - HTTP has a constrained user interface (set of verbs/operations/methods):
    - GET,
    - POST,
    - PUT,
    - DELETE,
    - HEAD,
    - OPTIONS (not widely supported),
    - TRACE (not significant),
    - CONNECT (not significant).
  - All of our resources will support GET!

Using HTTP to build REST Applications

- GET returns a representation of the current state of a resource.
- GET is “safe”:
  - Does not change resource state,
  - May trivially change server side state, e.g. log files,
- GET is “idempotent”:
  - Multiple requests are the same as a single request,
  - Might be impacted by concurrent operations.
- NEVER violate these rule.

Using HTTP to build REST Applications

- Highlight exceptional status codes:
  - HTTP has more response codes than 200 and 404 – learn them:
    - Information: 1xx, Success 2xx, Redirection 3xx, Client Error 4xx, Server Error 5xx;
  - For GETs we will need:
    - 200 OK,
    - 204 No Content,
    - 404 Not Found,
    - We'll need more later.

Using HTTP to build REST Applications

- REST Frameworks:
  - It is possible and legitimate to build REST systems with any HTTP-enabled application environment.
  - Few frameworks now, but more everyday:
    - RESTlet (Java, open source)
    - Ruby on Rails (Ruby, open source)
    - Django (Python, open source)
    - CherryPy (Python, open source)
    - JSR 311/JAX-RS (Java)
    - ASP.NET Web API (.NET)
    - Project Zero (Groovy/PHP, IBM incubator project)
    - Tycho (Reading system).