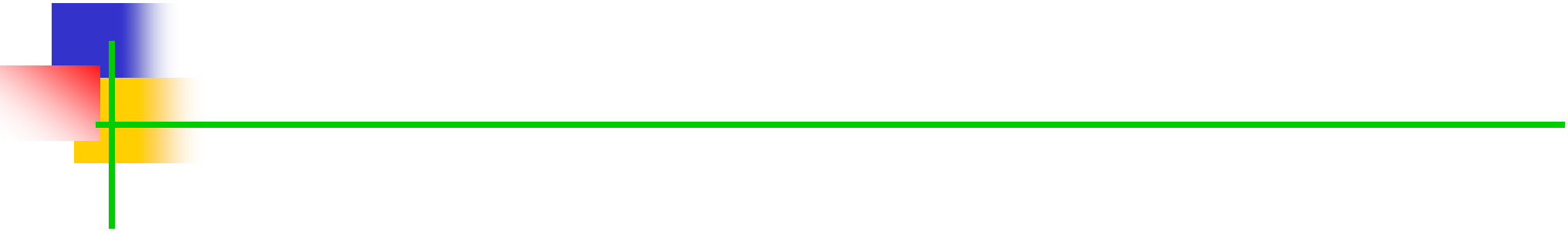
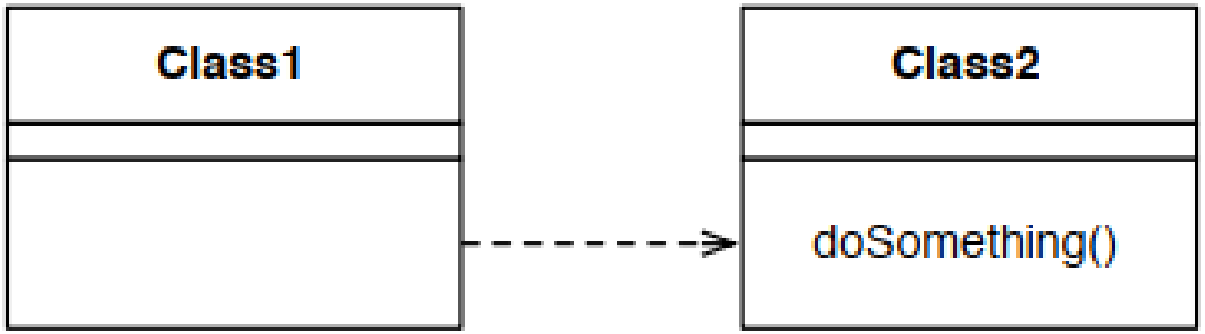
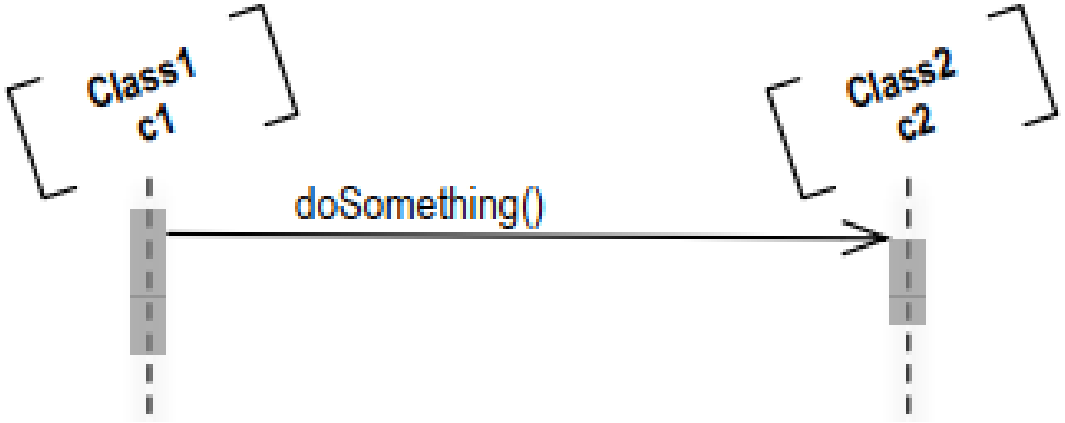
Service Oriented Architecture & Web services



Web Services?

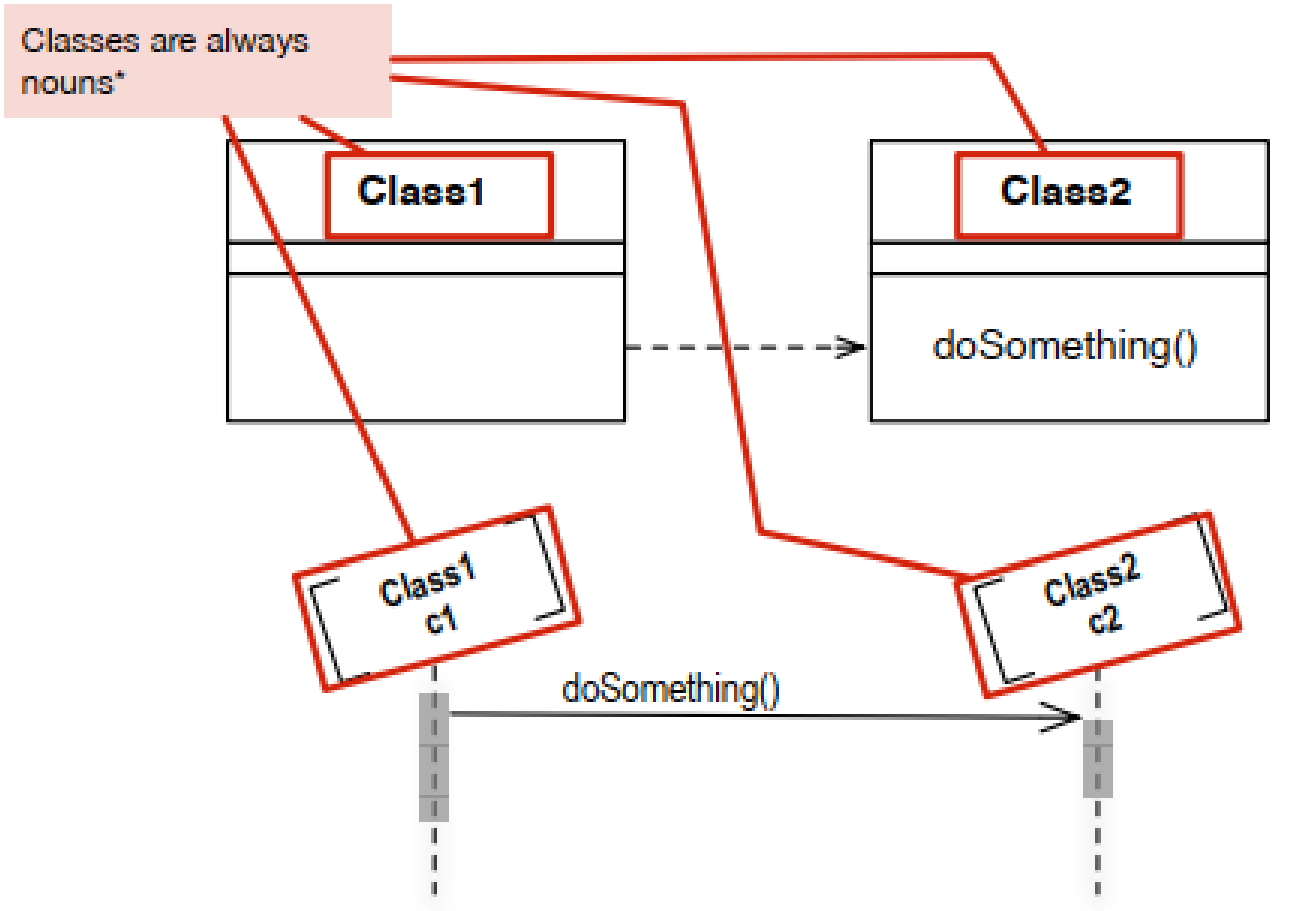
A quick look at Classes, and how they "talk" to each other





These slides are taken from CPSC 310 2015 course offering at UBC

A quick look at Classes, and how they “talk” to each other

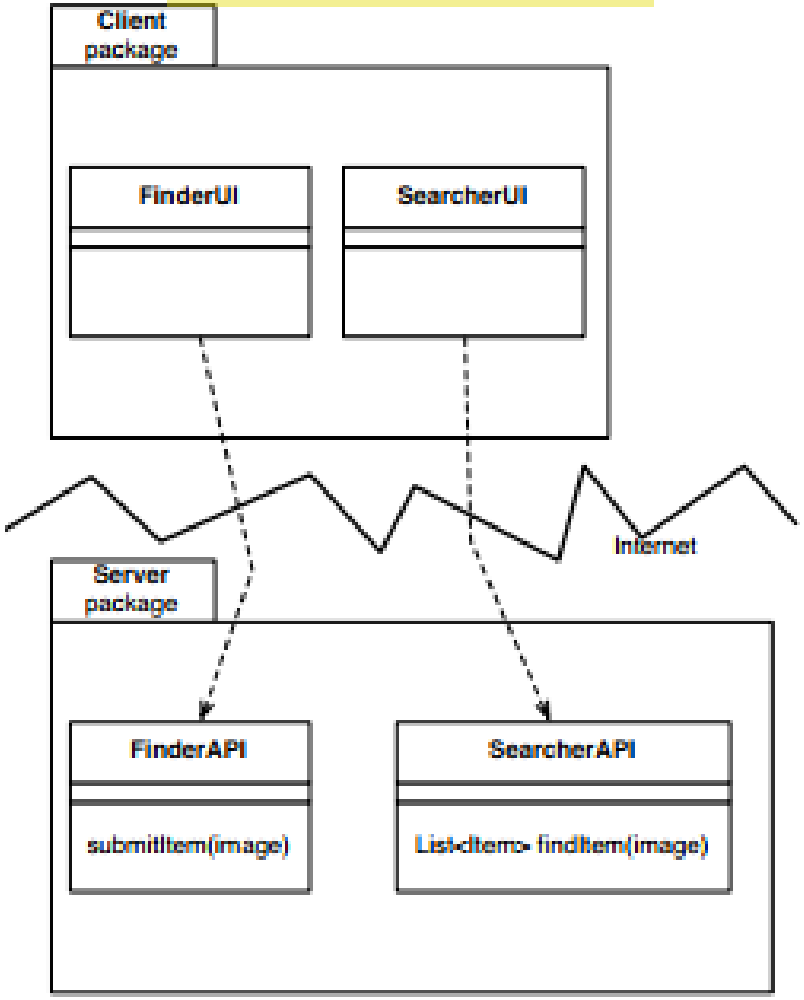


"note - they're often also "doers", or "responsibilities" but we'll think of them this way for now

These slides are taken from CPSC 310 2015 course offering at UBC

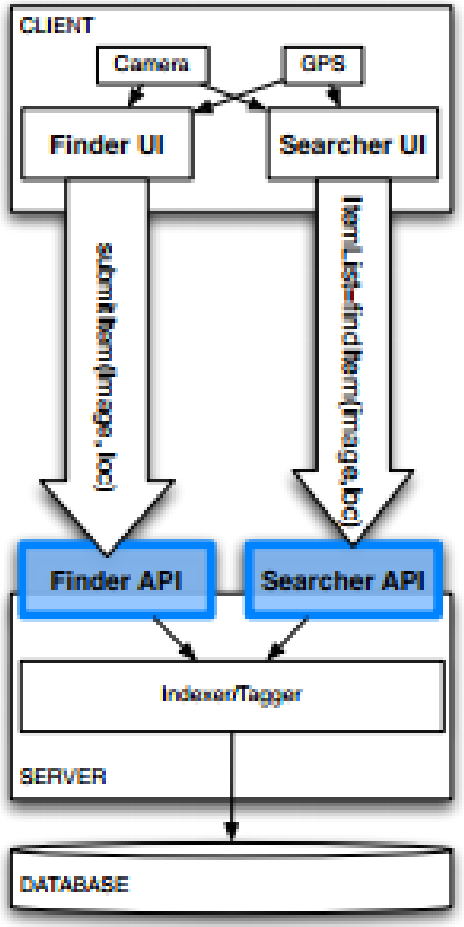
This design arrangements holds for client-server behaviour in an **RPC** architecture

Object-Oriented view



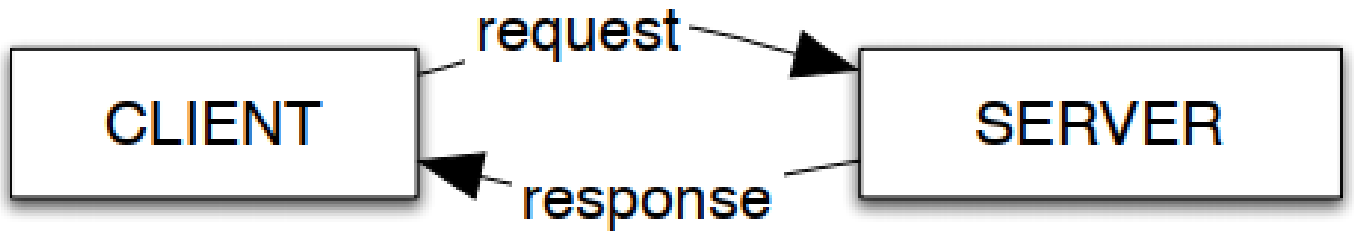
- This approach involves Remote Procedure Calls (**RPC**)
- Note that the **calls** are specialised **verbs** (submitImage( ), find( ))
- The data must be pre-defined, meaning we both the client and the server must agree on data types.

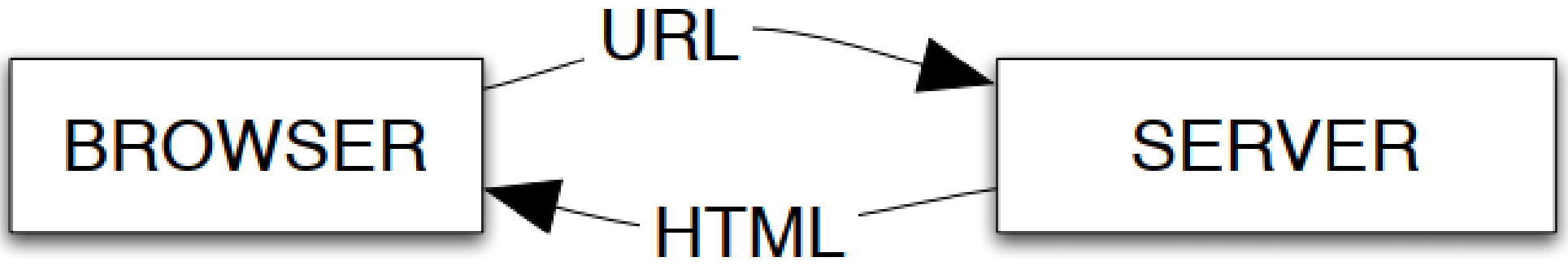
Architectural view



These slides are taken from CPSC 310 2015 course offering at UBC

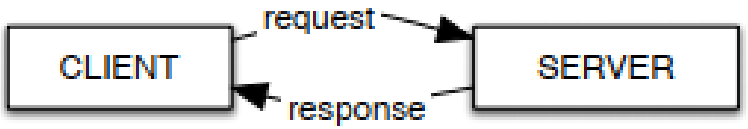
If your Client is a browser

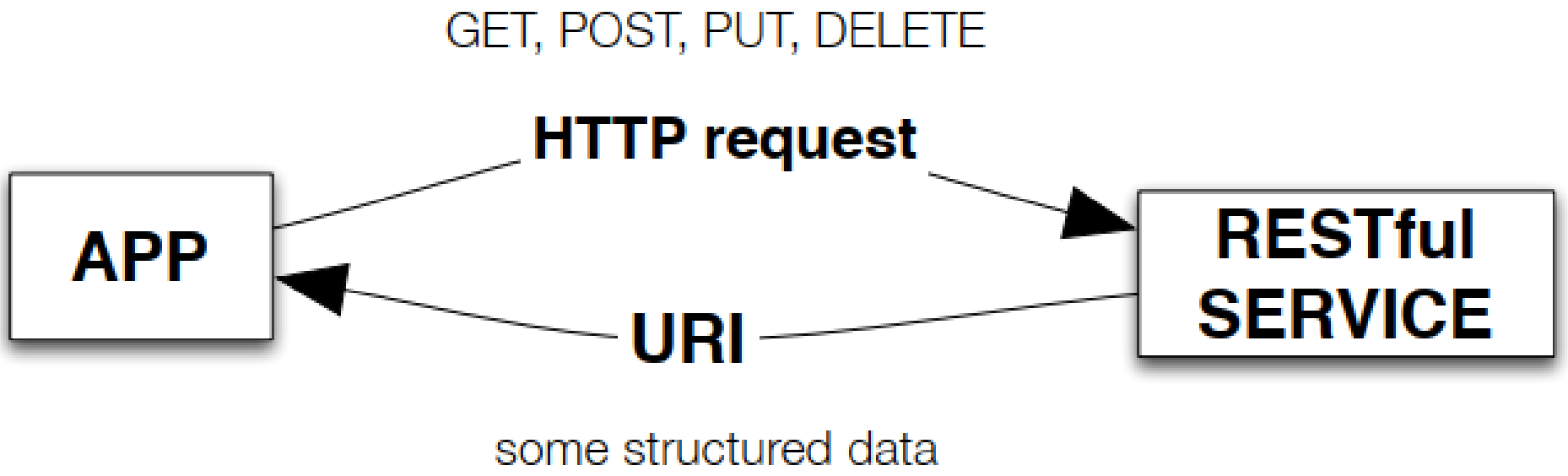




These slides are taken from CPSC 310 2015 course offering at UBC

If your Client is an Application



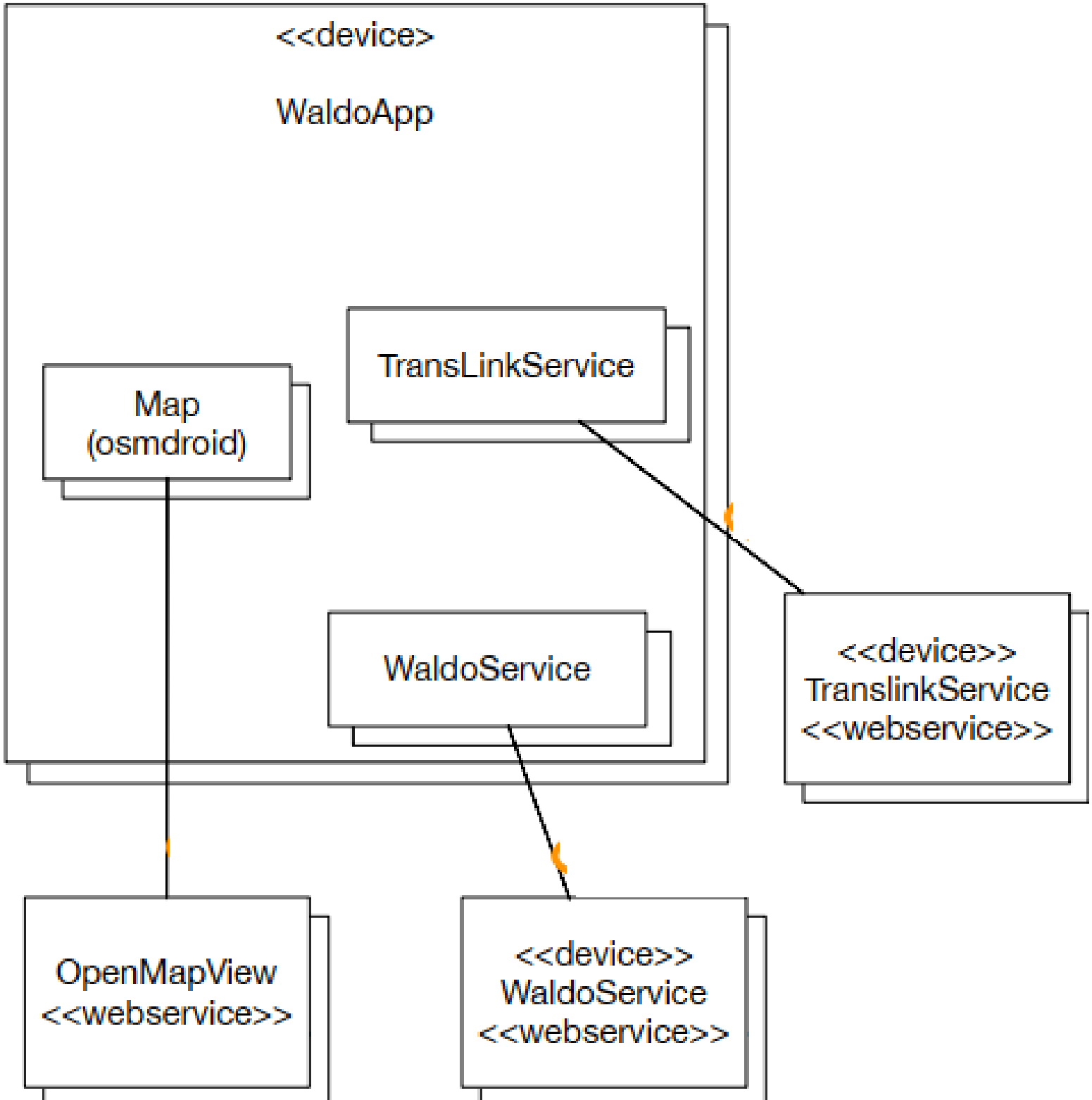


These slides are taken from CPSC 310 2015 course offering at UBC

Web Service Example: Waldo

- Consider an Android application called Waldo.
- The Waldo application lets you:
  - (1) track other users of the Waldo application within a certain geographic area
  - (2) plots your location and the locations of other users on a map
  - (3) uses live bus information from Translink to determine the best bus to use to reach a selected user's location from your current location.

These slides are taken from CPSC 210 2013 course offering at UBC



These slides are taken from CPSC 410 2016 course offering at UBC

Waldo Web Service

- The Waldo web service will let you query to:
  - find Waldos to plot on your map and for which you can search for routes to, and
  - receive messages sent to you from Waldos

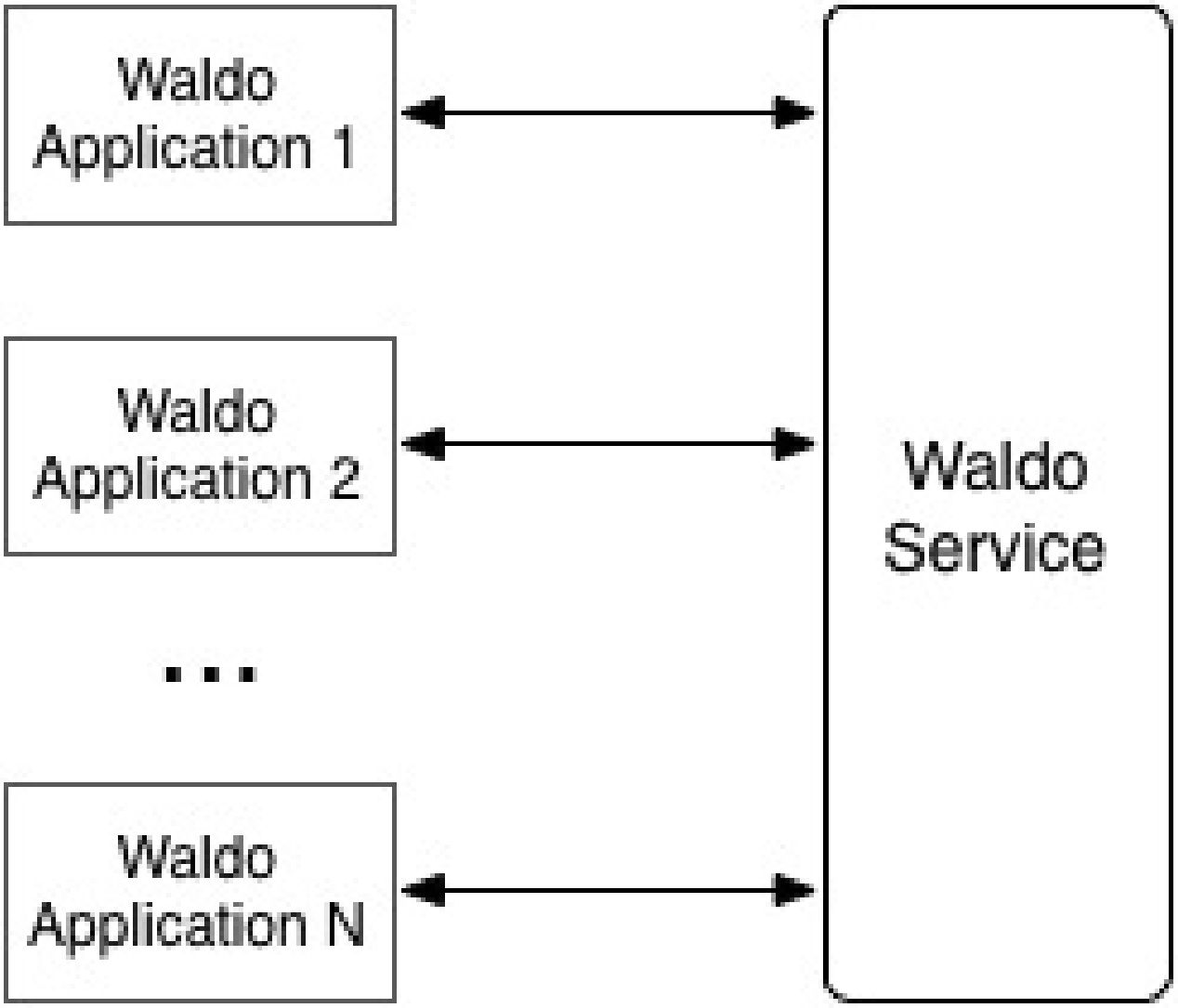
These slides are taken from CPSC 210 2013 course offering at UBC

Waldo Web Service

- The waldo service is a web-service that is deployed on server. It:
  - tracks the names of users, or waldos, who are using the service
  - allows waldos to notify the service of their current location
  - permits waldos to retrieve the locations of other waldos
  - permits waldos to exchange messages in a limited fashion

These slides are taken from CPSC 210 2013 course offering at UBC

Waldo Web Service Architecture



These slides are taken from CPSC 210 2013 course offering at UBC

What API is provided by Waldo Service?

- Waldo service API
  - 1 initSession
  - 2 getWaldos
  - 3 getWaldoByName
  - 4 postLocation
  - 5 sendMsg
  - 6 getMsgs
  - 7 Error codes
  - 8 Notes

These slides are taken from CPSC 210 2013 course offering at UBC

What API is provided by Waldo Service?

- Waldo service API
  - **getWaldos**
- Description:
  - Retrieve the location records for num number of waldos who have most recently updated their locations.
- Signature:
  - {ErrorNumber,ErrorString} | [{Name,Loc}, ... , {Name,Loc}] = getWaldos(key, num)

These slides are taken from CPSC 210 2013 course offering at UBC

What API is provided by Waldo Service?

- Waldo service API
  - **getWaldos**
- Example URL:

http://kramer.nss.cs.ubc.ca:8080/getwaldos/CWejewid/5

key = "CWejewid"

num = 5

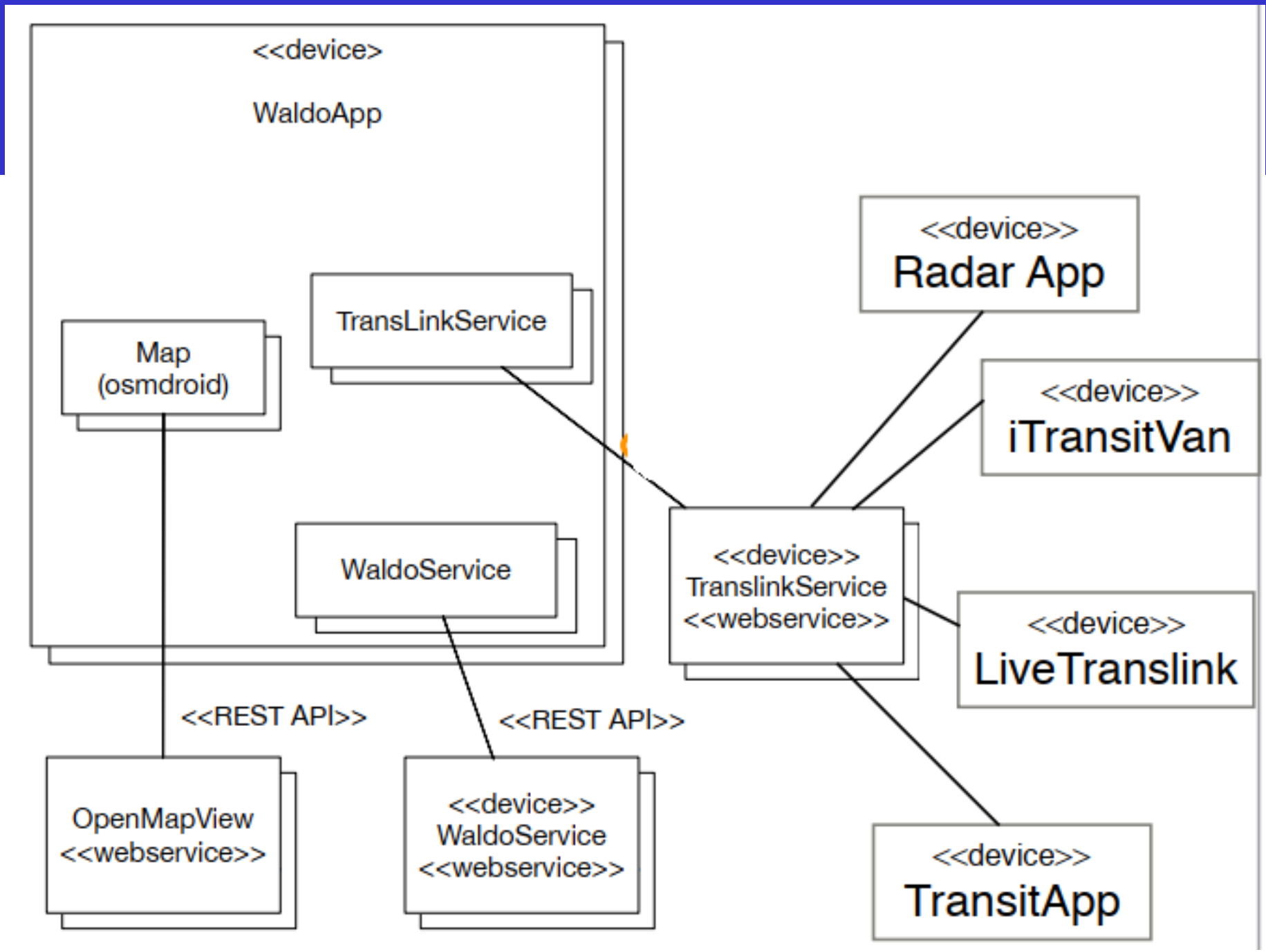
These slides are taken from CPSC 210 2013 course offering at UBC

What API is provided by Waldo Service?

- Waldo service API
  - **getWaldos**
- Example successful return value:

[{"Name":"StationaryEchoBot",
"Loc":{"Lat":49.26612,
"Long":-123.24703,
"Tstamp":1383530259}}
....]

These slides are taken from CPSC 210 2013 course offering at UBC



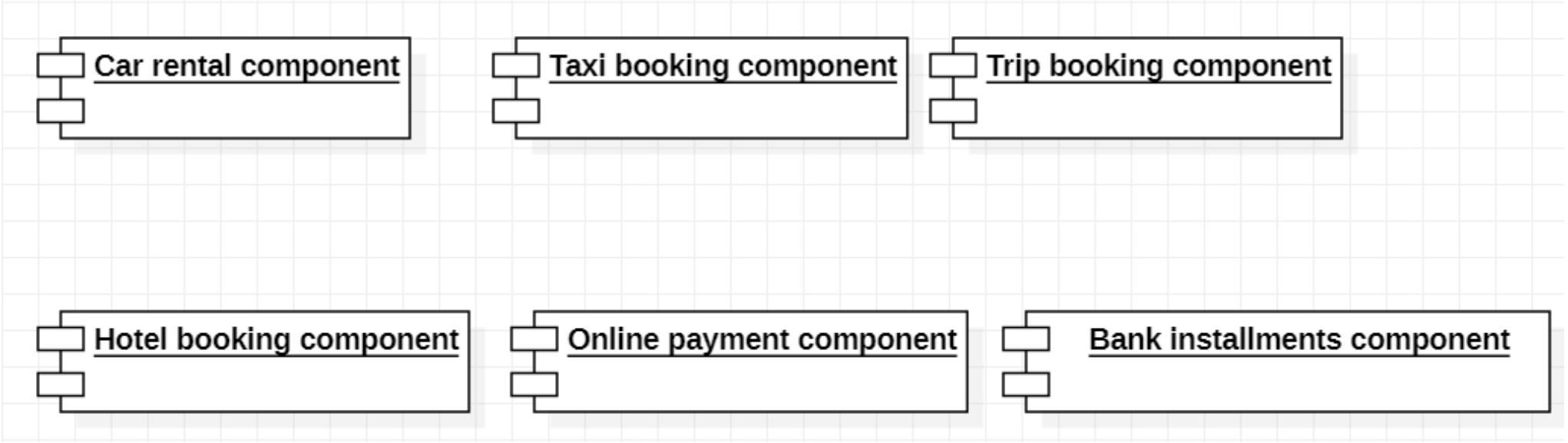
These slides are taken from CPSC 410 2016 course offering at UBC

Service-oriented Architecture

- A service is some functionality that is exposed for use by other processes.
- Such exposure is achieved through some interface that can be used by any service requestor.
- A web service is functionality that is exposed and would be accessed through web technologies.
- Service-oriented architecture examines how to build, use, and combine services.
- Instead of creating a large software suite that does everything, we can build and use services, and design an architecture that supports using such services.
- Note: Such definitions are retrieved from UoFA course on service-oriented architecture from Coursera.

Service-oriented Architecture

- How could the situation be prior to web services?



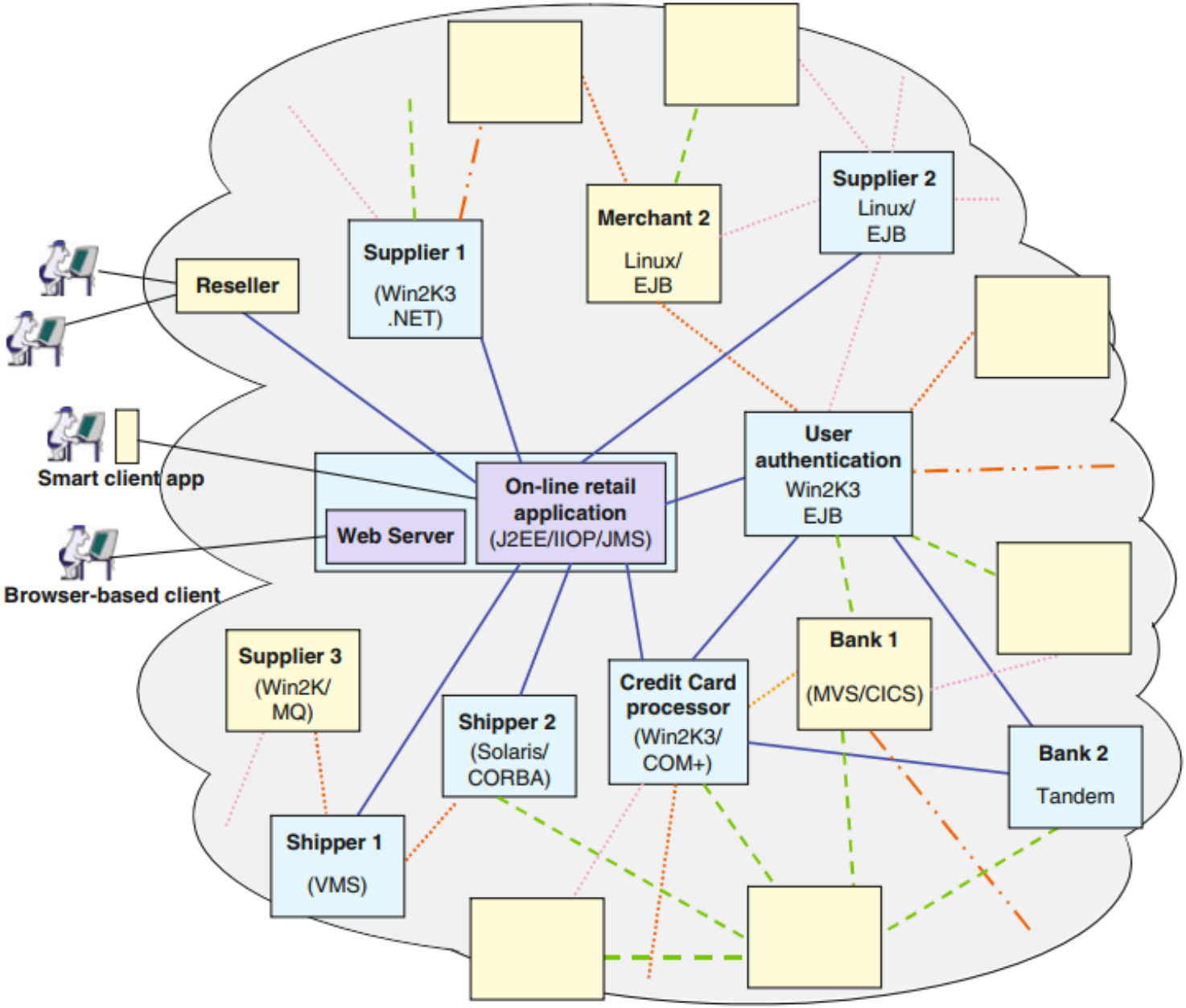
Service-oriented Architecture

- Instead, a web service defines such communication through a set of communication protocols and standard data formats.
- Hence, a web application for travelling (e.g., travgo) may take information from services that obtain flight prices, services that obtain hotel prices, car rental services.
- However, some non-functional requirements become very important, like:
  - Response time
  - Availability

Adopting Service Oriented Architecture

- Moving to a completely service-oriented architecture could be difficult to support despite its benefits.
- Instead, services to the most crucial, cross-departmental features are introduced first.
- Services provide: modularity, extensibility, and code reuse.

SOA Example



Customers can access this application using Web browsers or through smarter client applications that make calls directly into the back-end services provided by the retailer's core application.

SOA Principles

- Boundaries are explicit
  - Services are independent applications, not some code that is bound to your program that can be called at no cost.
  - Accessing a service requires crossing over boundaries that separate processes, traversing networks, and user authentication.
  - Every boundary (process, machine, trust) that has to be crossed reduces performance, adds complexity, and increases the chances of failure.
  - Good services should have simple interfaces and share as few abstractions as possible and assumptions with their users (why?)

SOA Principles

- Services are autonomous
  - Services are autonomous independent applications, not classes or components that are tightly bound into client applications
  - Services have control over the logic they encapsulate, from a Design-time and a Run-time perspective.
  - Design-time autonomy (Example?)
  - Run-time autonomy (Example?)
  - Source: https://en.wikipedia.org/wiki/Service_autonomy_principle

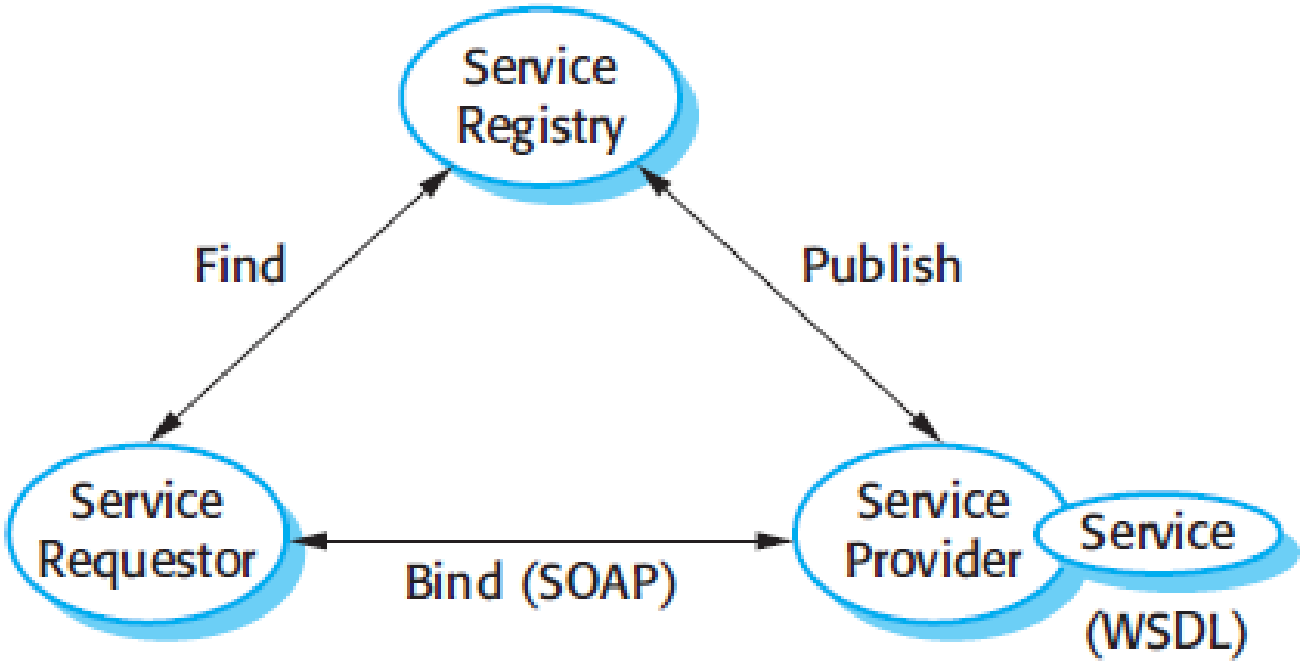
SOA Principles

- Share schemas and contracts, not implementations
  - Services are just applications that receive and send messages. Clients and services share nothing other than the definitions of these messages and certainly don’t share method code or complex run-time environments.
- Service compatibility is based on policy
  - Clients have to be completely compatible with the services they want to use (E.g., Expected communication protocol)

Web Services Types

- The two big web service standards are **SOAP** and **REST**.
- With SOAP, messages are defined using the Web Services Description Language (WSDL) and the message is encoded in a well-defined XML format called a SOAP message.

Roles of SOA Building Blocks



by Dirk Krafzig, Karl Banke, and Dirk Slama

26

SOAP Web Services

- Simple Object Access Protocol (SOAP) is a protocol for exchanging structured information in the implementation of web services.
- SOAP uses XML for its message format, and relies on HTTP or SMTP for message negotiation and transmission.
- The Web Services Description Language (WSDL) is an XML-based interface definition language (IDL) that is used for describing the functionality offered by a web service.

SOAP Web Services

- Invoking/Binding (SOAP): is the act of generating the necessary code to interact with a service so that a service requester can begin invoking it.

28

WSDL Example

```
<service name="StockQuoteService">
  <documentation>Stock quote service</documentation>
  <port name="StockQuotePort"
        binding="tns:StockQuoteBinding">
     <soap:address location=
              "http://myCompany.com/stockServices"/>
  </port>
</service>
</definitions>
```

```
<message name="GetLastTradePrice">
    <part name="body" type="xsd:string"/>
</message>
<message name="LastTradePrice">
    <part name="body" type="xsd:float "/>
</message>
```

Invocation (SOAP) Example

```
<soap:Header>
<wsa:Action>
    http://myCompany.com/getLastTradePrice</wsa:Action>
      <wsa:MessageID>uuid:4ec3a973-a86d-4fc9-bbc4-ade31d0370dc
      </wsa:MessageID>
      <wsse:Security soap:mustUnderstand="1"
            <wsse:UsernameToken>
                  <wsse:Username>NNK</wsse:Username>
                  <wsse:PasswordType="http://docs.oasis-
                  open.org/wss/2004/01/oasis-200401-wss-username
                        -token-profile-1.0#PasswordDigest">
                  weYI3nXd8LjMNVksCKFV8t3rgHh3Rw==
                  </wsse:Password>
                  <wsse:Nonce>WScqanjCEAC4mQoBE07sAQ==</wsse:Nonce>
                  <wsu:Created>2003-07-16T01:24:32Z</wsu:Created>
            </wsse:UsernameToken>
      </wsse:Security>
</soap:Header>
```

Invocation (SOAP) Example – Cont'd

```
<soap:Body>
        <m:GetLastTradePrice
        xmlns:m="http://myCompany.com/stockServices">
                <symbol>DIS</symbol>
        </m:GetLastTradePrice>
</soap:Body>
```

RESTful Web Services

- RESTful Web services rely on HTTP as a sufficiently rich protocol to completely meet the needs of Web services applications.
- In the REST model, each web service is mapped to one of the HTTP methods: GET, POST, PUT, and DELETE.
- Documents are “representations” of “resources” that are identified by normal Web URIs (Uniform Resource Identifiers)

RESTful Web Services (Example)

- The following example shows how a simple customer database Web service could be implemented using a RESTful approach.
- The URI http://example.com/customers identifies the customer database resource.
  - GET requests sent to this URI return the set of all customers as a single XML document containing a list of URIs that point to the individual customer resources.

RESTful Web Services (Example - 2)

- The URI for each customer in the database is formed by appending the customer's unique ID to the customer set URI. For example, http://example.com/customers/1 identifies the resource corresponding to the customer with ID 1
- A GET request sent to one of these unique customer URIs retrieves an XML document that contains a representation of the current state of the corresponding customer.
- Existing customer resources can be updated by PUTting an XML document containing a representation of the desired new state of the customer to the appropriate customer URI.

RESTful Web Services (Example - 3)

- New customers can be added to the database by POSTing XML documents containing the representation of the new resource to the collection URI.
- The URIs for the new customer resources are returned using the HTTP location header in the server's responses.
- Customer entries can be deleted by sending a DELETE request to the customer URI.

Required Reading

- Chapter 5, from Ian Gorton's: Essential Software Architecture, Springer Verlag Second Edition, 2011.

References

- Distributed Systems: Concepts and Design, 5th edition. Coulouris, et. Al
- http://en.wikipedia.org/wiki/Service-oriented_architecture
- http://en.wikipedia.org/wiki/Web_service
- Software Engineering (9th Edition), Ian Sommerville
- Cook, William R., and Janel Barfield. "Web services versus distributed objects: A case study of performance and interface design." Web Services, 2006. ICWS'06. International Conference on. IEEE, 2006.