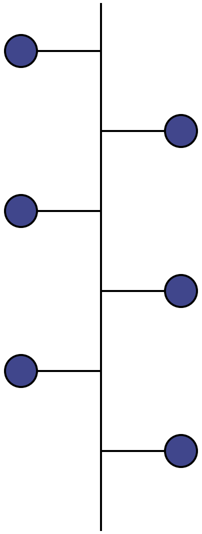
TechGI IV - ComDiS

ComDis

Introduction to
Communication Networks
and **Dis**tributed Systems



From WWW to Web Services

Prof. Dr.-Ing. Adam Wolisz

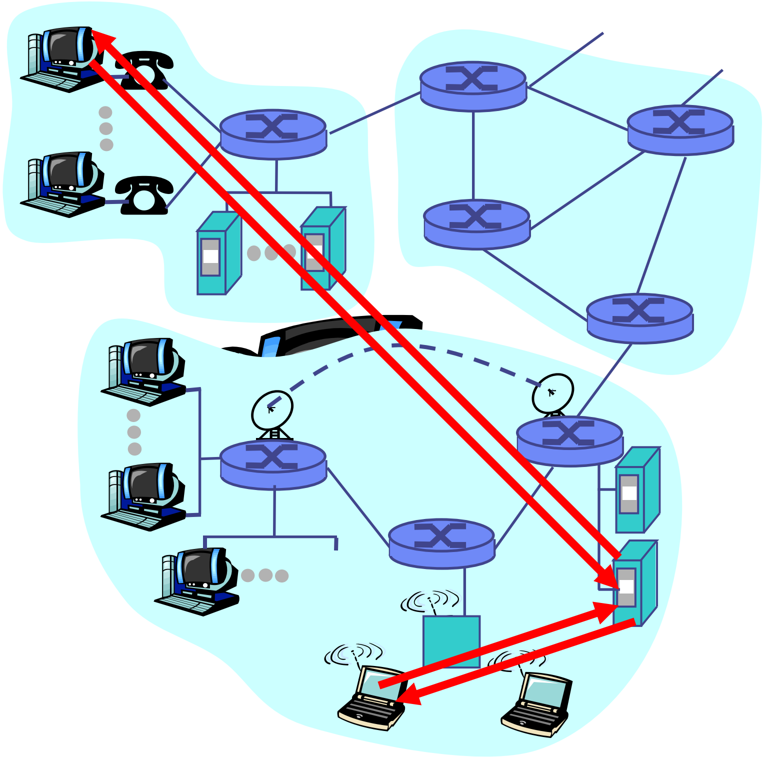
TKN Telecommunication Networks Group

Acknowledgements:

- We acknowledge the use of slides from: Prof. Holger Karl, Paderborn; Prof. Ion Stoica, Berkeley, Prof. Ashay Parekh; Berkeley; Prof Lauer WPI; Prof. Baker, ACET, as well as slides form books by Tannenbaum, Kurose and Ross, Colouris at al.

Client-Server Model

[Kurose, Ross]



server:

- always-on host
- permanent IP address
- server farms for scaling

clients:

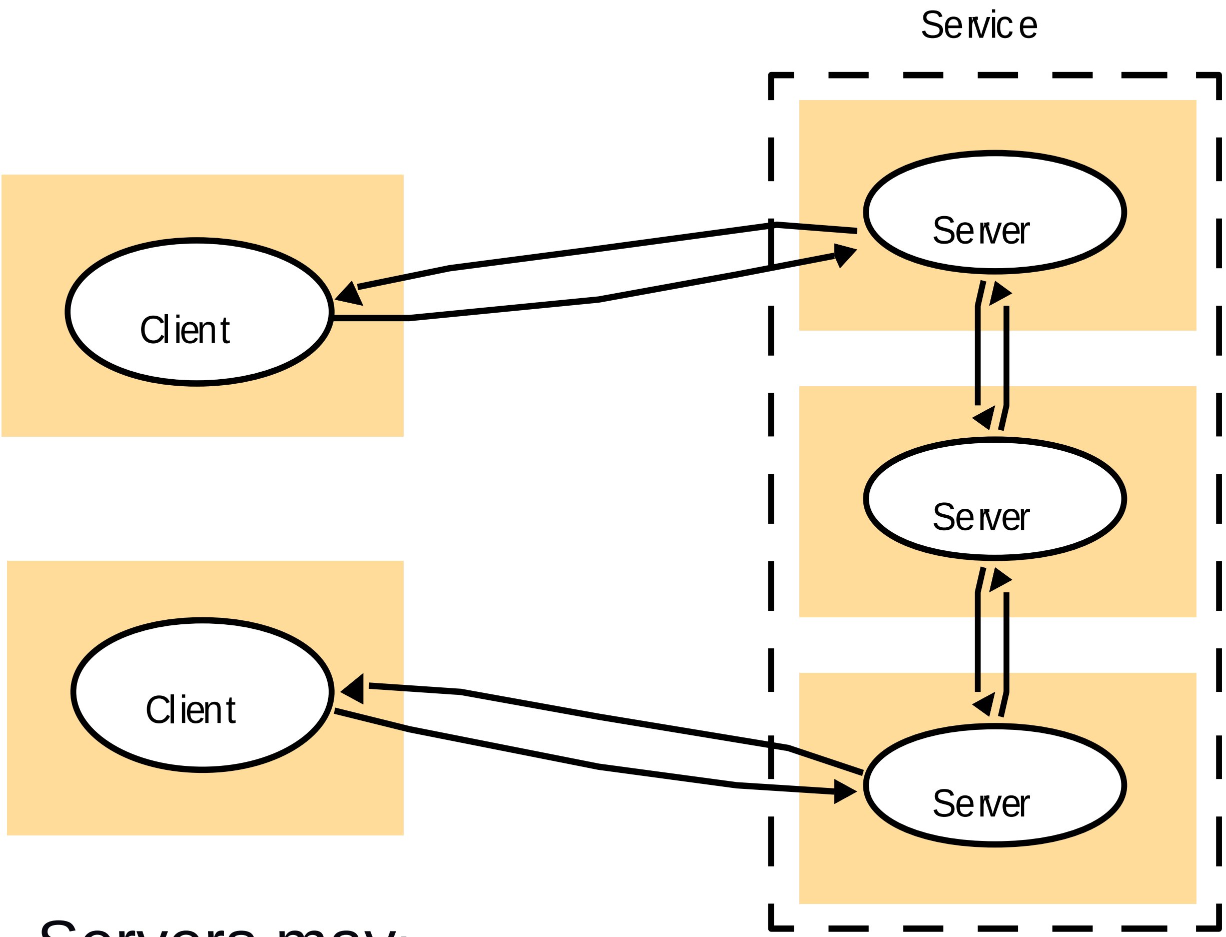
- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

Features of client-server systems

- Advantages
  - The work can be *distributed* among different machines
  - The clients can access the server's functionality from a *distance*
  - The client and server can be *designed separately*
  - Simple, well known programming paradigm: call a function!
  - The server can be accessed *simultaneously* by many clients
  - Modification of the server easy …
- Disadvantages
  - Server can become a bottleneck
  - Latency in communication might be significant
  - Resources (e.g. disc space) of clients might be wasted
  - How to find e server for a given service?
  - The service might be disables if a server fails.

Service Provided by Multiple Servers

[Colouris]



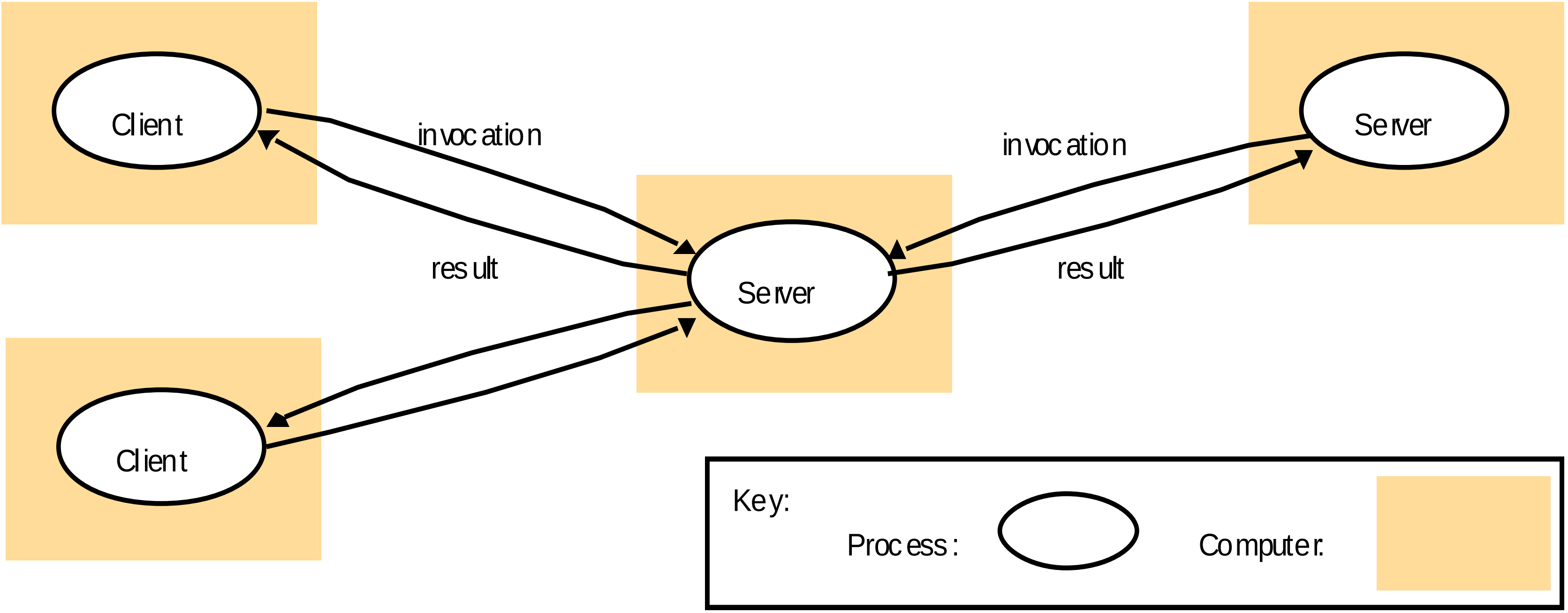
Servers may:

- partition the set of services
- maintain replicated copies (consistency of data???)

System Architecture: Client-Server Model

[Colouris]

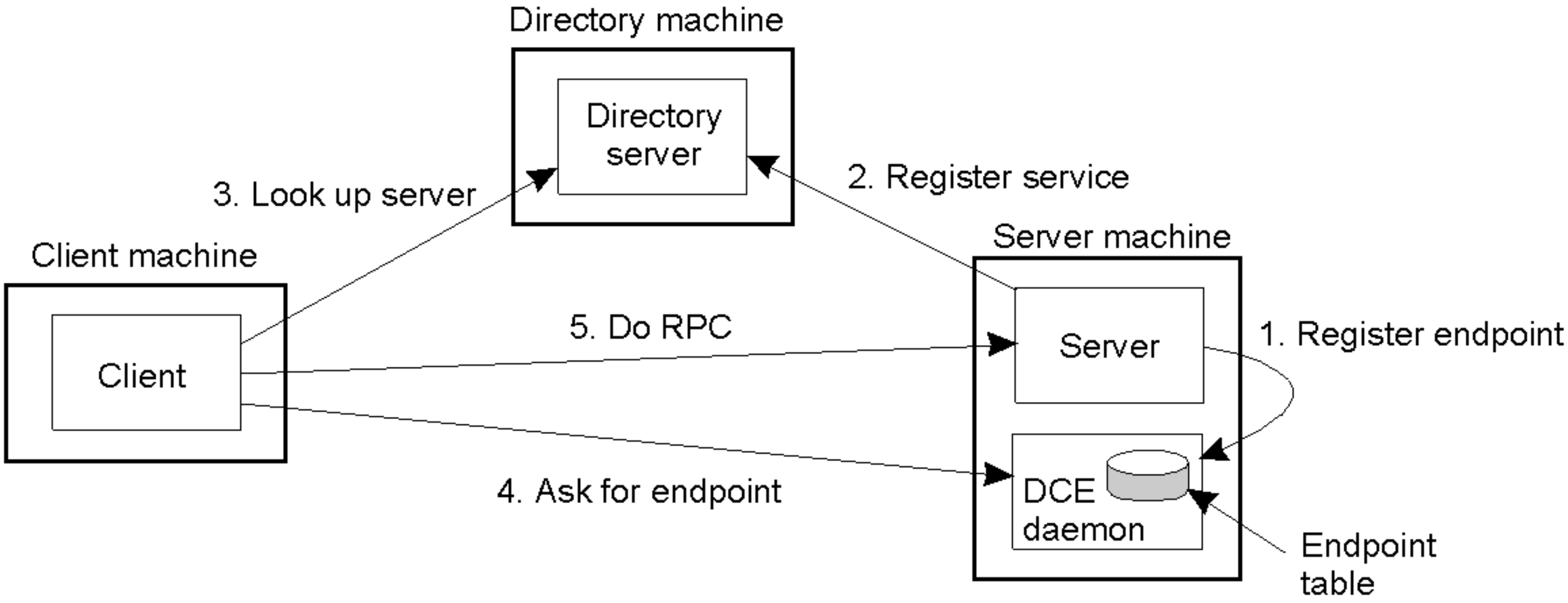
A server process can act as client for an other server



A service is not bound to a specific computer….
How to get to the right one?

Binding a Client to a Server

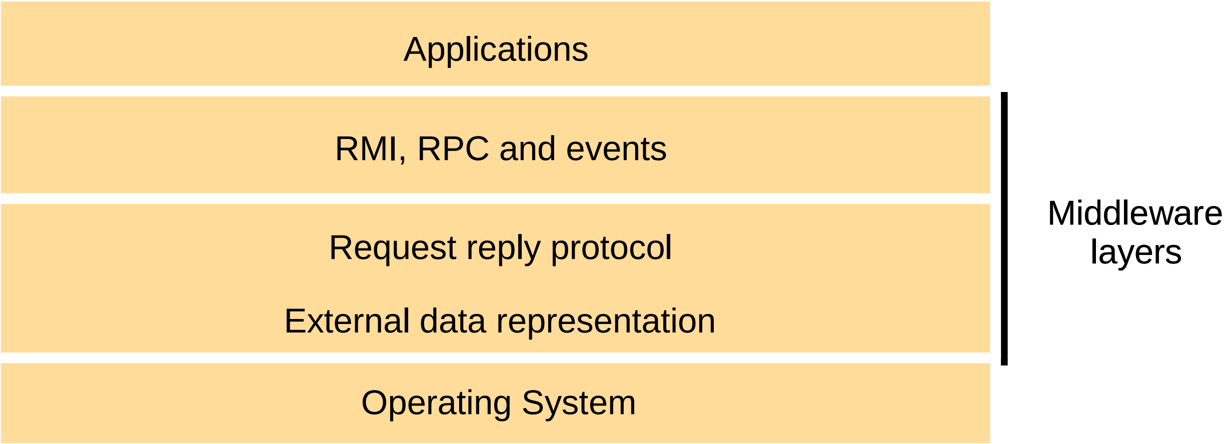
- Client-to-server binding in DCE.



RPC as middleware component

[karl,Paderborn]

- RPCs present an abstracter view of a distributed system than a request/reply protocol directly realized with sockets
  - New programming model!
- Collection of software realizing such a new programming model is a ***middleware***
  - Can achieve, e.g., transparency towards location, communication protocols, hardware, operating system, different programming languages, …



Applications

RMI, RPC and events

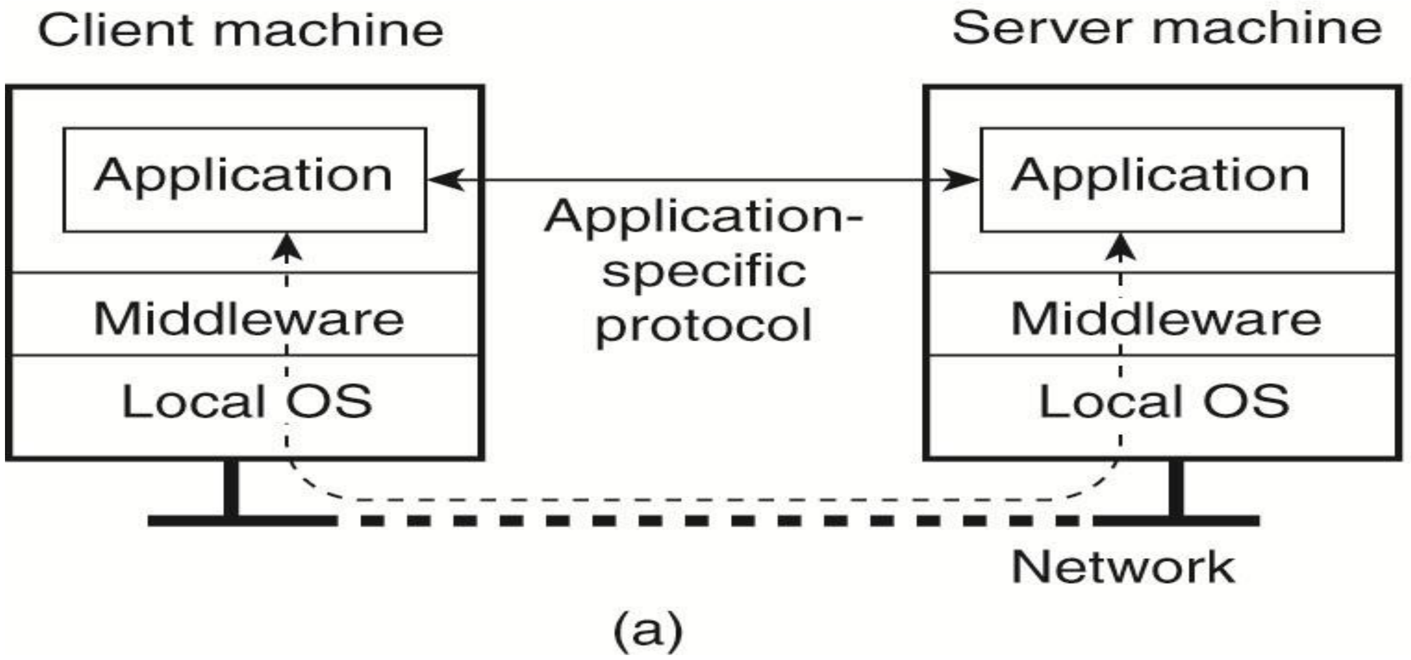
Request reply protocol

External data representation

Operating System

Middleware layers

Middleware placement.



Notions

- Names
  - Data types that refer to specific entities in a system

    Example: ***Jonathan M. Smith***
- Addresses
  - Identifiers of places to find the named entities

    Example: ***123 Park St., Andover, MA***
- Routes (also called *paths*)
  - steps to follow to get to named entities

    Example:

    - ***From Andover center, go west 1.2 miles***
    - ***Turn right, then take 3rd left***
    - ***He lives at the 2nd house on the righ***

Name Service

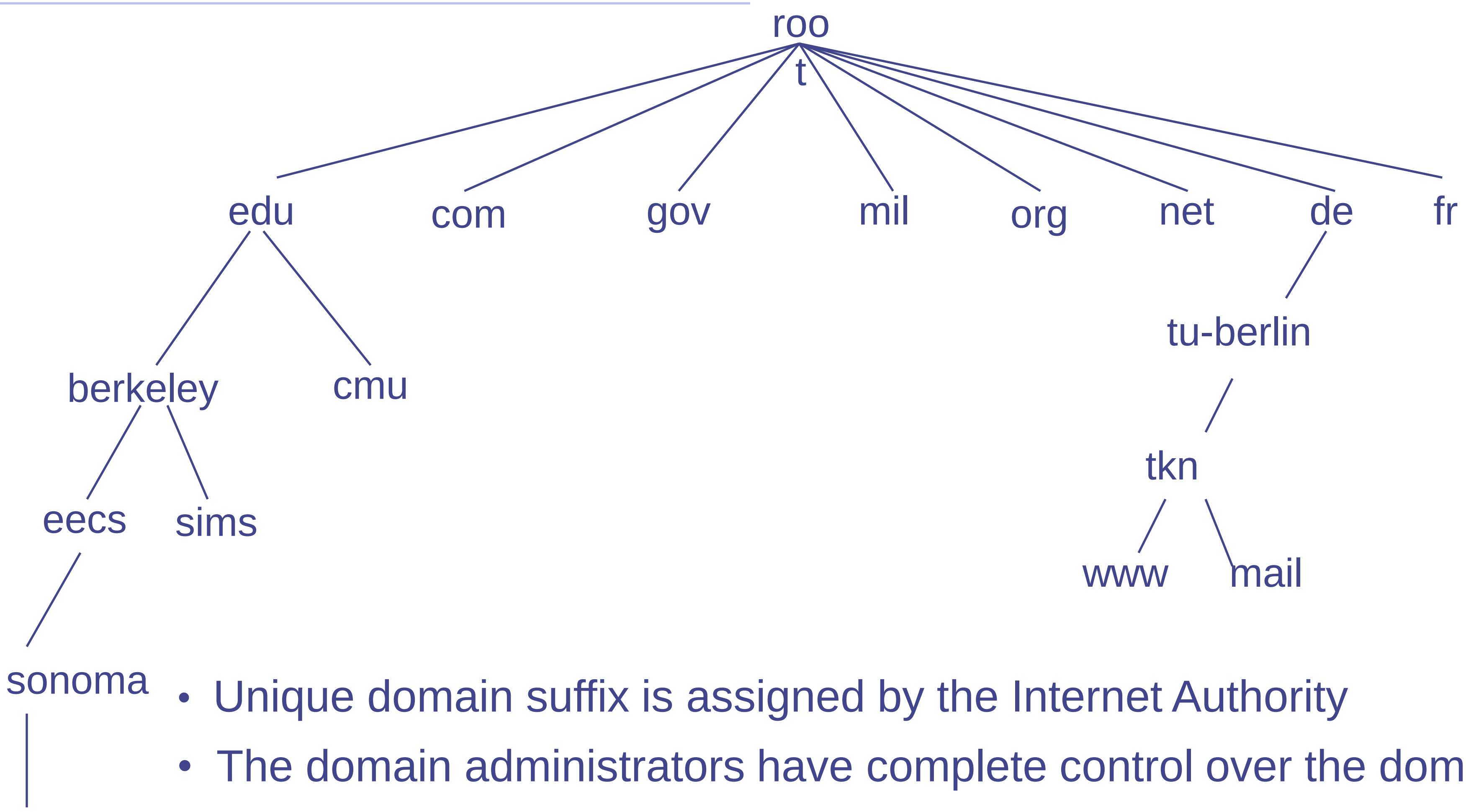
- Name space: set of possible names and their relationship

  (Names are usually chosen so as to be meaningful/easy to memorize)
- Bindings: the mapping between names and addresses
- Name resolution: procedure that, when invoked with a name, returns the corresponding value
- Name server (*directory*): specific implementation of a resolution mechanism that is available on the network and that can be queried by sending messages

Name Spaces

- Name space organization
  - Flat Name Space
    - All names are equivalent in name space
    - Must be globally unique
  - Hierarchical Name Space
    - Names (usually) have structure
    - Unique only within immediately containing level
- Internet names are structured (Scaling!!)
  - **`ccc3.wpi.edu`**
  - **`update.microsoft.com`**
- Names spaces organized into directed graph
  - Leaf nodes represent named entities
  - Interior nodes represent *directories*
- Everyone has to know *root node*

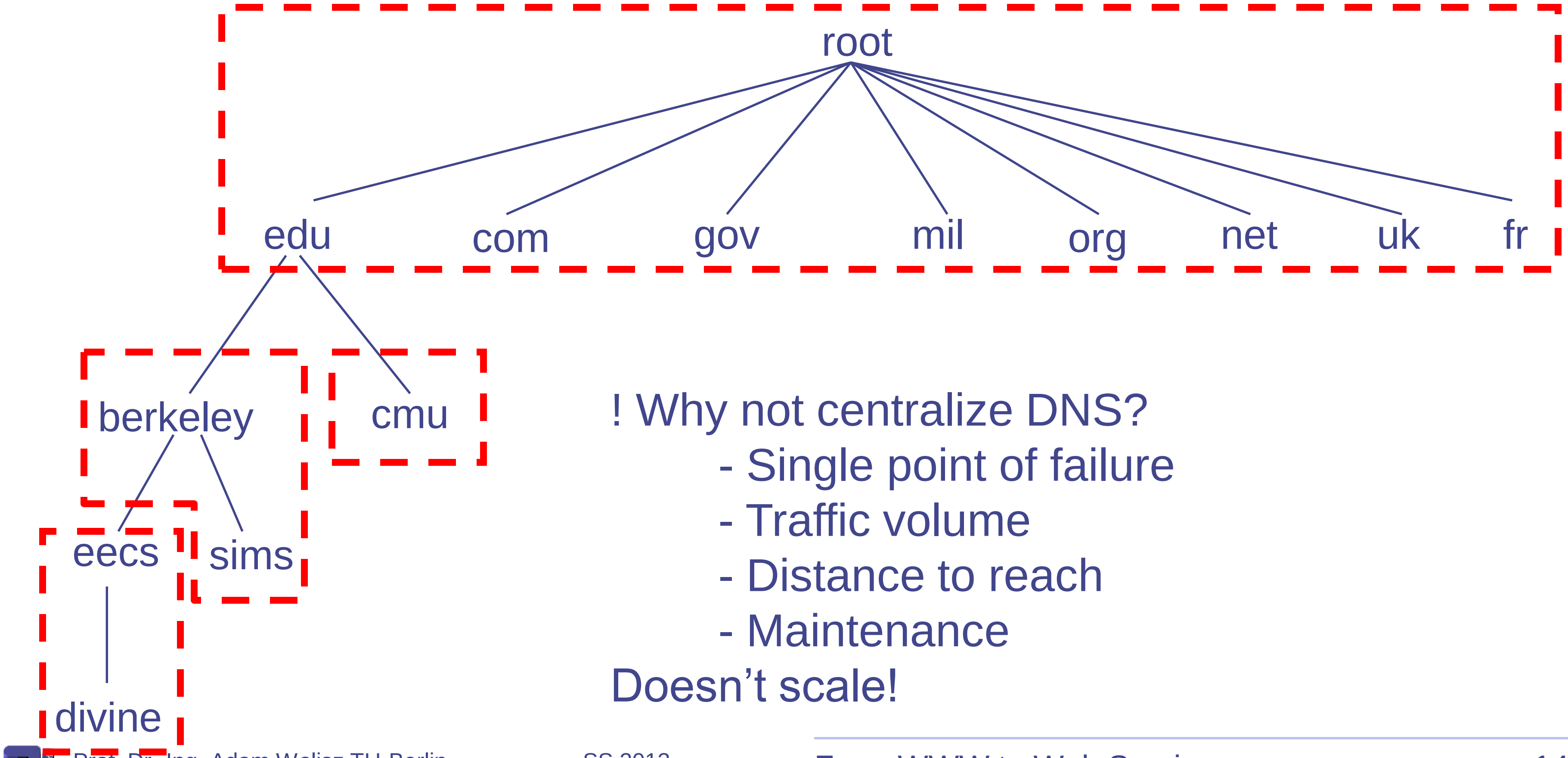
Domain Name System (DNS) Hierarchy



- Unique domain suffix is assigned by the Internet Authority
- The domain administrators have complete control over the domain
- No limit on the number of subdomains or number of levels
- Name space is not related with the physical interconnection
- ´ A name could be a domain or an individual object

Server Hierarchy: Zones

- A zone corresponds to an administrative authority that is responsible for that portion of the hierarchy



! Why not centralize DNS?

- Single point of failure
- Traffic volume
- Distance to reach
- Maintenance

Doesn’t scale!

Some details

- Each server has authority over the portion of the hierarchy
  - A server maintains only a subset of all names!
  - Each server contains all the names of the hosts in its zone
  - Back up-servers for redundancy

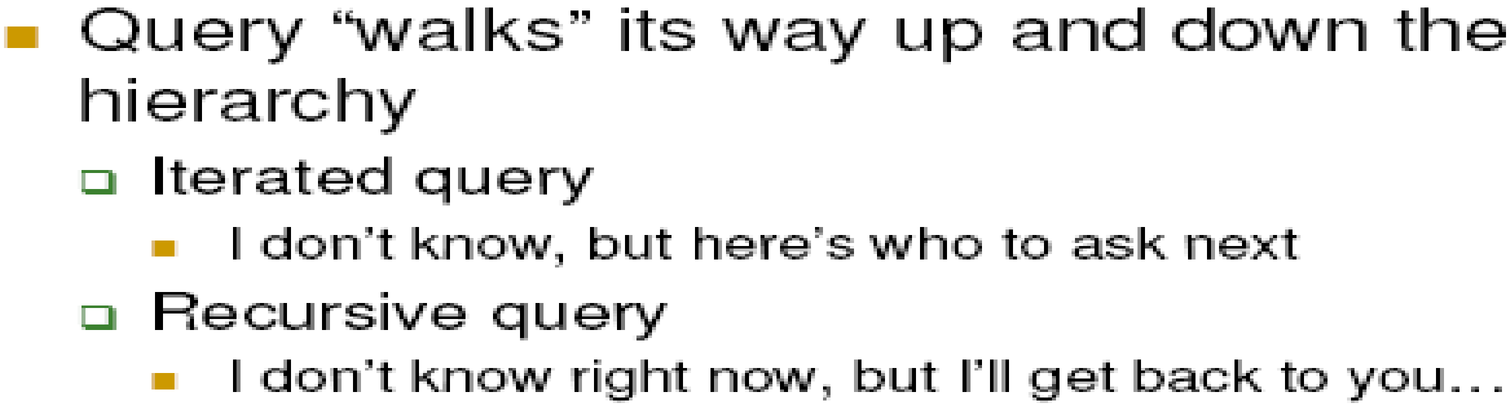
- Each server has to know servers responsible for other portions of the hierarchy
  - Every server knows its root
  - Root server knows all the top-level domains….

- Client server interaction on UDP Port 53

  - but can use TCP if desired!

Autoritative/Local Name server

- When a host makes a DNS Query, query is sent to ist LOCAL DNS server.
- If this server is not able to resolve, query starts walking up the hierarchy, until somebody can point out the Authoritative server for the queried domain…
- Afterwards the query „walks down“ until the proper sub-domain is found



- Query “walks” its way up and down the hierarchy
  - Iterated query
    - I don’t know, but here’s who to ask next
  - Recursive query
    - I don’t know right now, but I’ll get back to you…

Simple DNS example

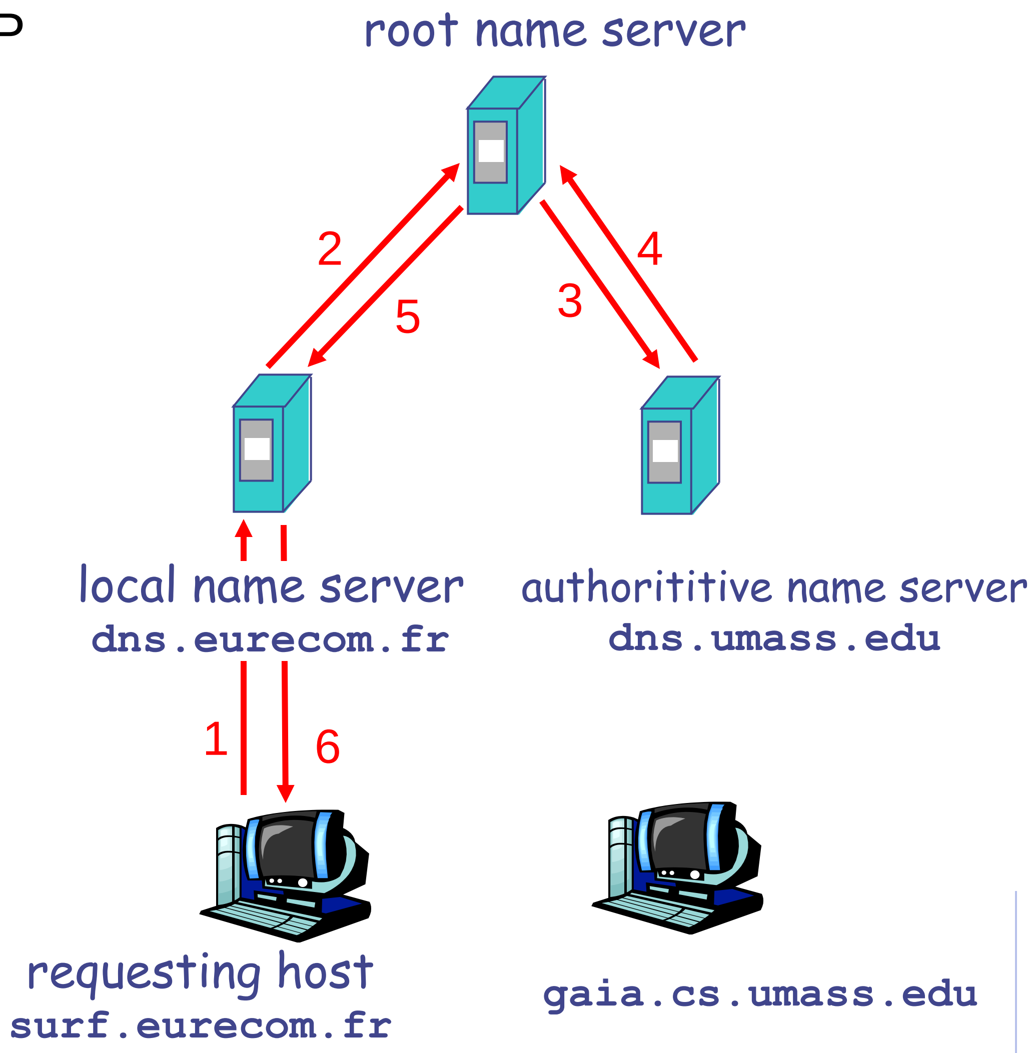
host **surf.eurecom.fr** wants IP address of **gaia.cs.umass.edu**

1. contacts its local DNS server, **dns.eurecom.fr**

2. **dns.eurecom.fr** contacts root name server, if necessary

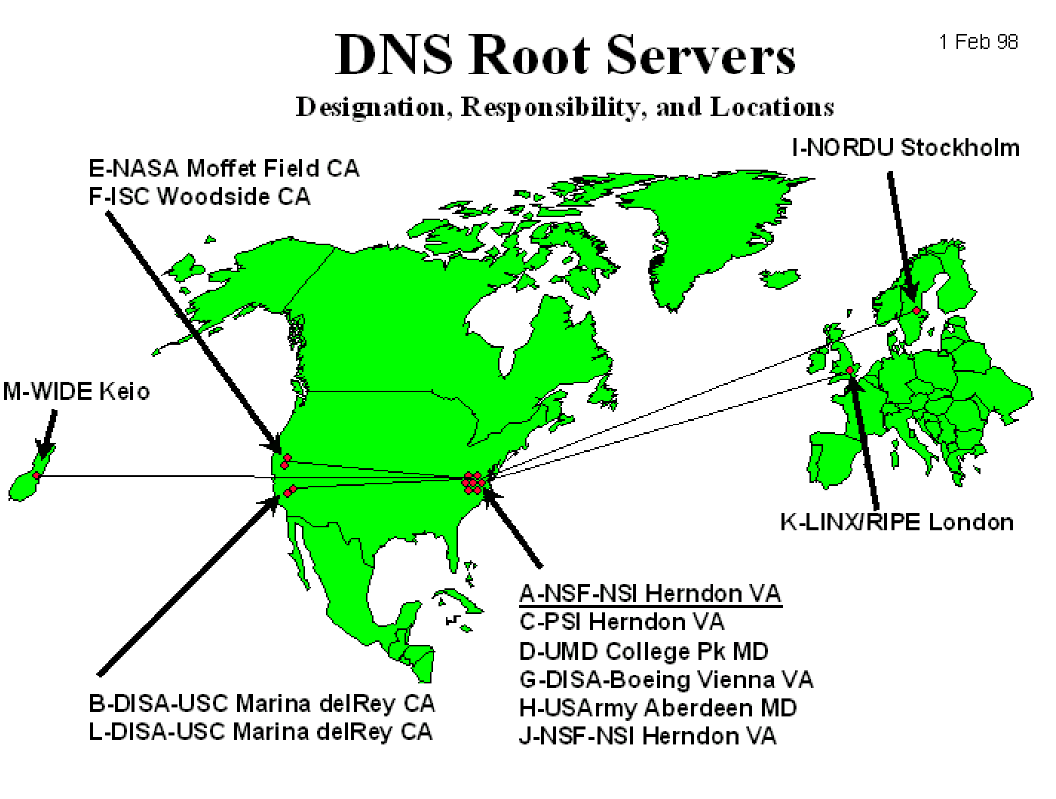
3. root name server contacts authoritative name server, **dns.umass.edu**, if necessary

***Comment**: the local name server can CASH the entry for later re-usage!!*



DNS: Root Name Servers

- Contacted by local name server that can not resolve name
- Root name server:
  - Contacts authoritative name server if name mapping not known
  - Gets mapping
  - Returns mapping to local name server
- ~ Dozen root name servers worldwide



DNS and Virtual IP addresses

DNS records don’t have to store the real IP address of the host.

- Multiple names can map onto the same address
  - Example: www.berkeley.edu and arachne.berkeley.edu maps to the same machine (i.e., the same IP address)
  - Example: All hosts in the acme.com may have the same IP address/port – in reality a gatekeeper
    - The Gatekeeper decides whether to “admit” a transport level connection to the host x.acme.com (the ***firewall*** functionality)
    - The Gatekeeper decides to forward the connection to one of several identical servers (A ***load balancer*** functionality)

Example: www.akamai.com

DNS records don't have to store the real IP address of the host.

- One name can map onto multiple addresses
  - Example: www.yahoo.com can be mapped to multiple machines
  - Example: www.akamai.com

    From Berkeley 64.164.108.148

    From the NY Area 63.240.15.146

    From the UK *194.82.174.224*

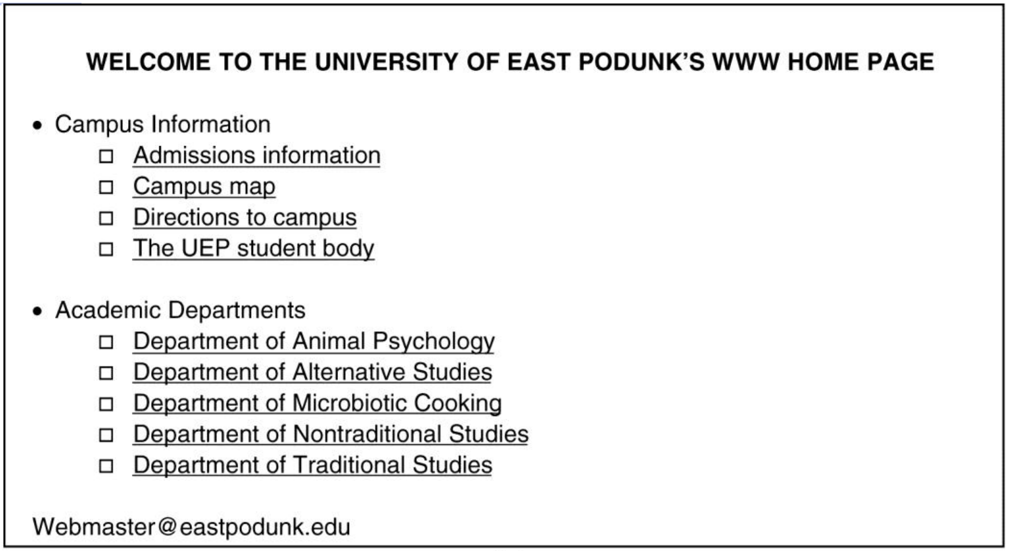
  What do we gain: shorter delay, load distribution

WWW

WWW
- we use it,
- do we understand it?

The notion of a hyperlink is – in fact - very old..

The technical implementation: information system instead of books- makes the difference!

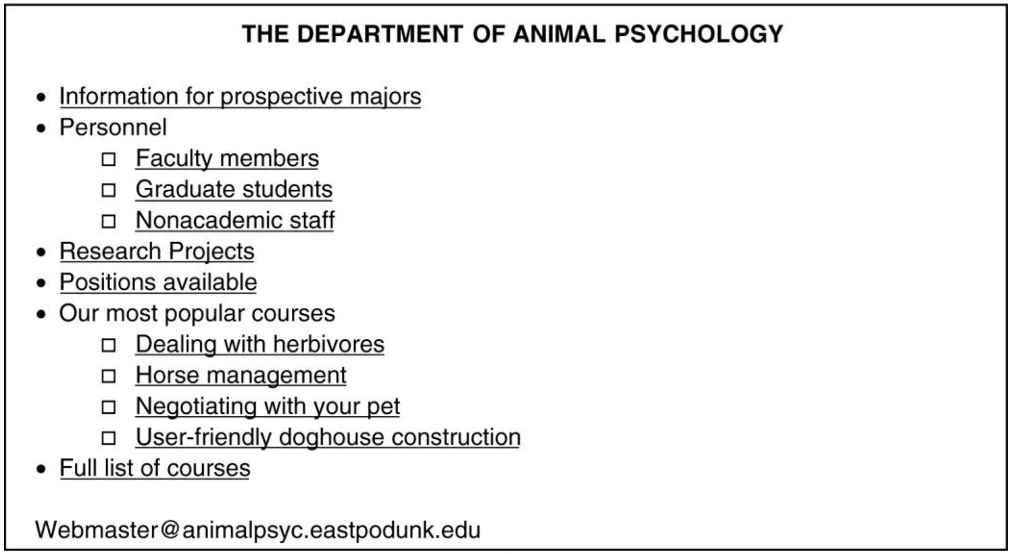


WELCOME TO THE UNIVERSITY OF EAST PODUNK'S WWW HOME PAGE

- Campus Information
  - □ Admissions information
  - □ Campus map
  - □ Directions to campus
  - □ The UEP student body
- Academic Departments
  - □ Department of Animal Psychology
  - □ Department of Alternative Studies
  - □ Department of Microbiotic Cooking
  - □ Department of Nontraditional Studies
  - □ Department of Traditional Studies

Webmaster@eastpodunk.edu

(a)



THE DEPARTMENT OF ANIMAL PSYCHOLOGY

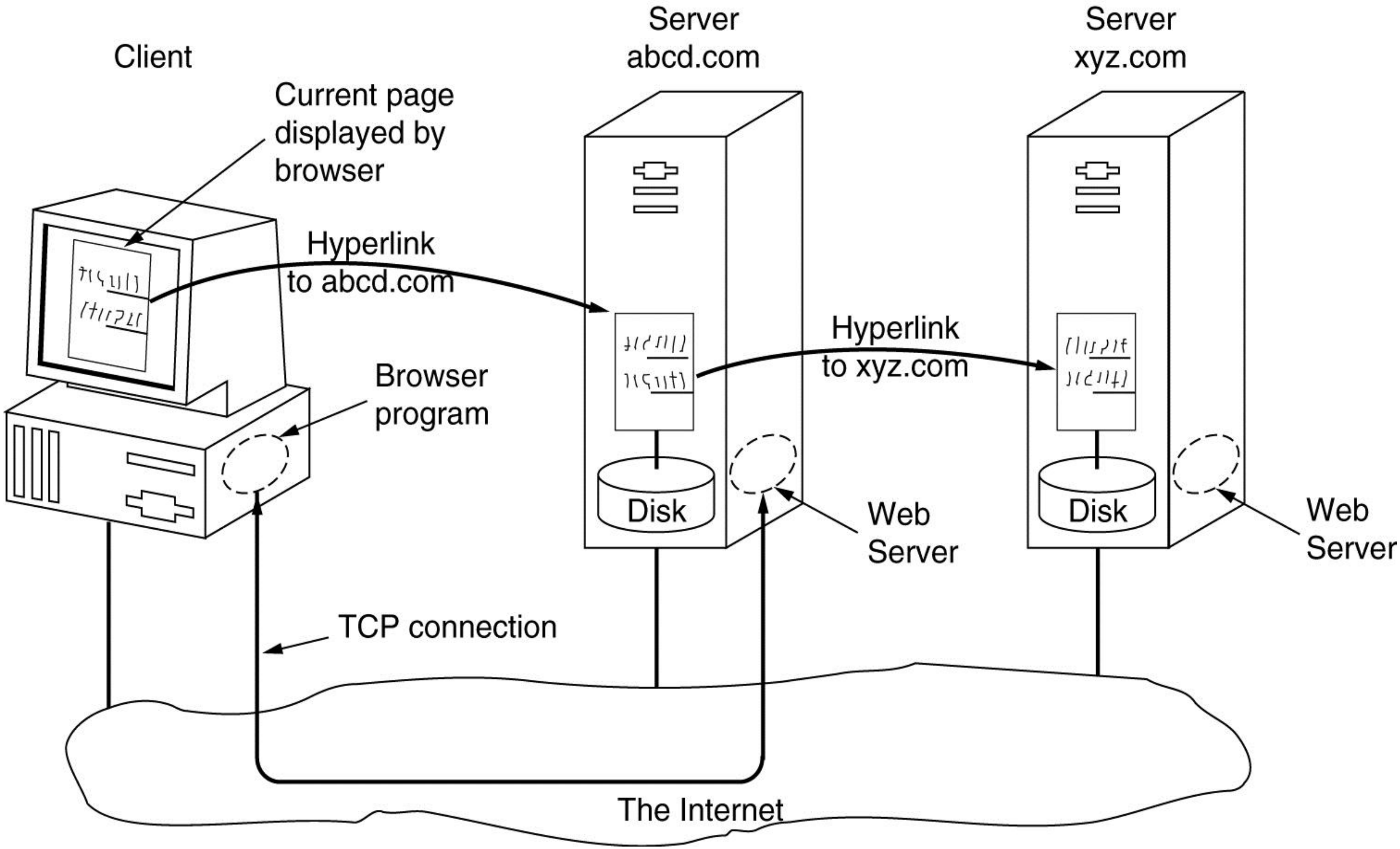
- Information for prospective majors
- Personnel
  - □ Faculty members
  - □ Graduate students
  - □ Nonacademic staff
- Research Projects
- Positions available
- Our most popular courses
  - □ Dealing with herbivores
  - □ Horse management
  - □ Negotiating with your pet
  - □ User-friendly doghouse construction
- Full list of courses

Webmaster@animalpsyc.eastpodunk.edu

(b)

Architectural Overview

[Tannenbaum]



The WWW

- A distributed database of URLs (*Uniform Resource Locators*)
- Core components:
  - Servers which store “web pages” and execute remote commands
  - Browsers retrieve and display “pages” of content linked by hypertext
    - Each link is a URL

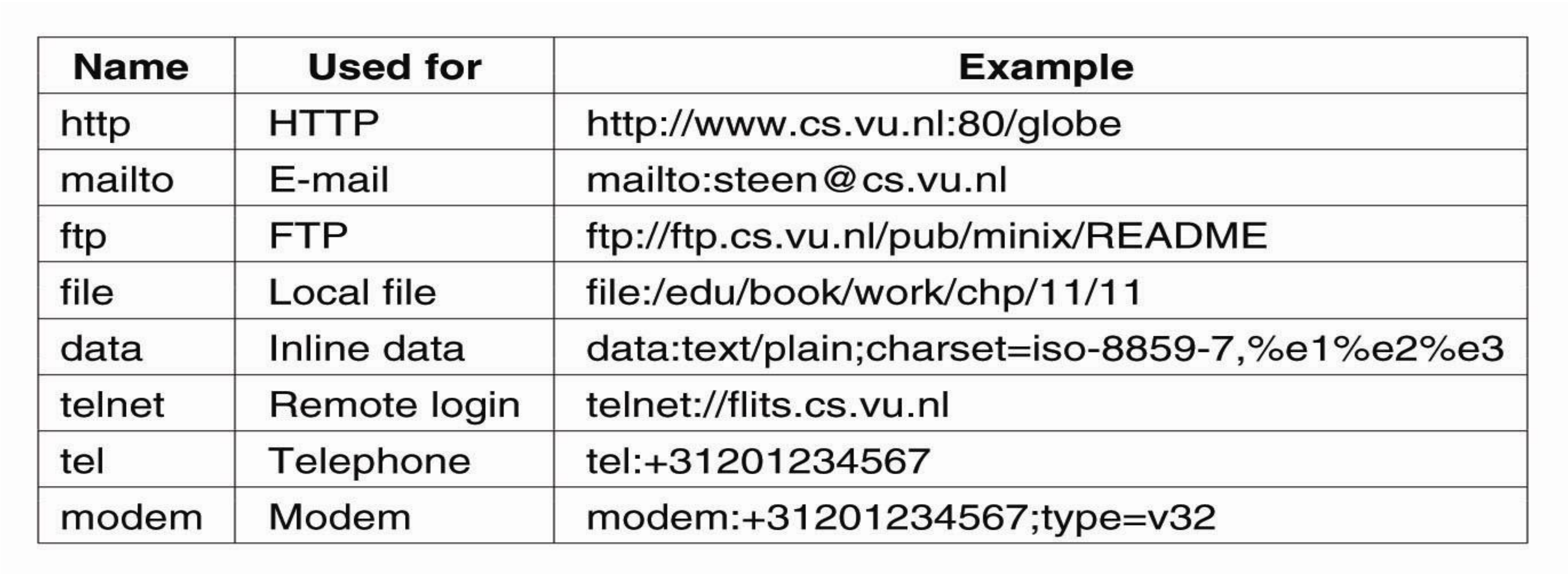
➔ Can build arbitrarily complex applications, all of which share a uniform client!

- Need a language to define the objects and the layout
  - HTML, XML
- Need a protocol to transfer information between clients and servers
  - HTTP

Uniform Record Locator

- protocol://host-name:port/directory-path/resource
- Extend the idea of hierarchical namespaces to include anything in a file system
  - – ftp://www.eecs.berkeley.edu/122/Lecture6/presentation.ppt

Examples of URIs



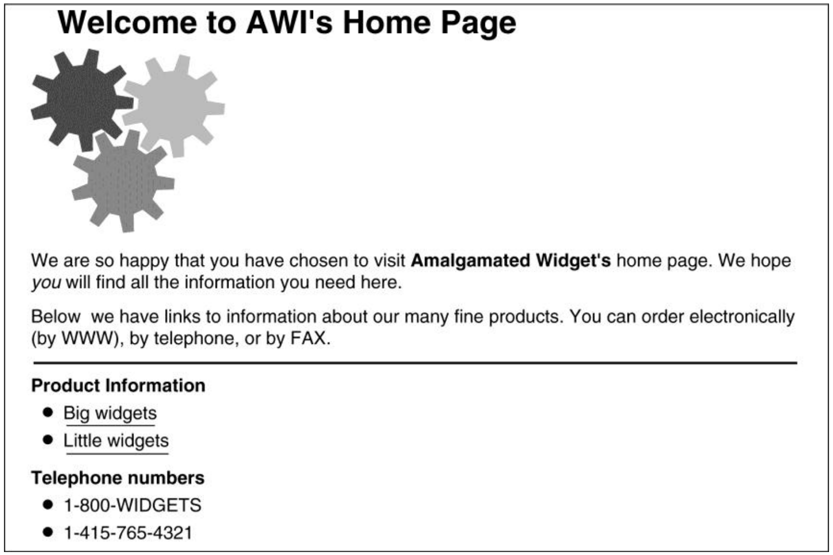
Name	Used for	Example
http	HTTP	http://www.cs.vu.nl:80/globe
mailto	E-mail	mailto:steen@cs.vu.nl
ftp	FTP	ftp://ftp.cs.vu.nl/pub/minix/README
file	Local file	file:/edu/book/work/chp/11/11
data	Inline data	data:text/plain;charset=iso-8859-7,%e1%e2%e3
telnet	Remote login	telnet://flits.cs.vu.nl
tel	Telephone	tel:+31201234567
modem	Modem	modem:+31201234567;type=v32

HTML – HyperText Markup Language

(a) The HTML for a sample Web page. (b) The formatted page.

```
<html>
<head><title> AMALGAMATED  WIDGET, INC. </title> </head>
<body> <h1> Welcome to AWI's Home Page</h1>
<img src="http://www.widget.com/images/logo.gif" ALT="AWI Logo"> <br>
We are so happy that you have chosen to visit <b> Amalgamated Widget's </b>
home page. We hope <i> you </i> will find all the information you need here.
<p>Below we have links to information about our many fine products.
You can order electronically (by WWW), by telephone, or by fax. </p>
<hr>
<h2> Product information </h2>
<ul>
   <li> <a href="http://widget.com/products/big"> Big widgets </a>
   <li> <a href="http://widget.com/products/little">  Little widgets </a>
</ul>
<h2> Telephone numbers</h2>
<ul>
   <li> By telephone: 1-800-WIDGETS
   <li> By fax: 1-415-765-4321
</ul>
</body>
</html>
```

(a)



Welcome to AWI's Home Page

We are so happy that you have chosen to visit **Amalgamated Widget's** home page. We hope *you* will find all the information you need here.

Below we have links to information about our many fine products. You can order electronically (by WWW), by telephone, or by FAX.

Product Information

- Big widgets
- Little widgets

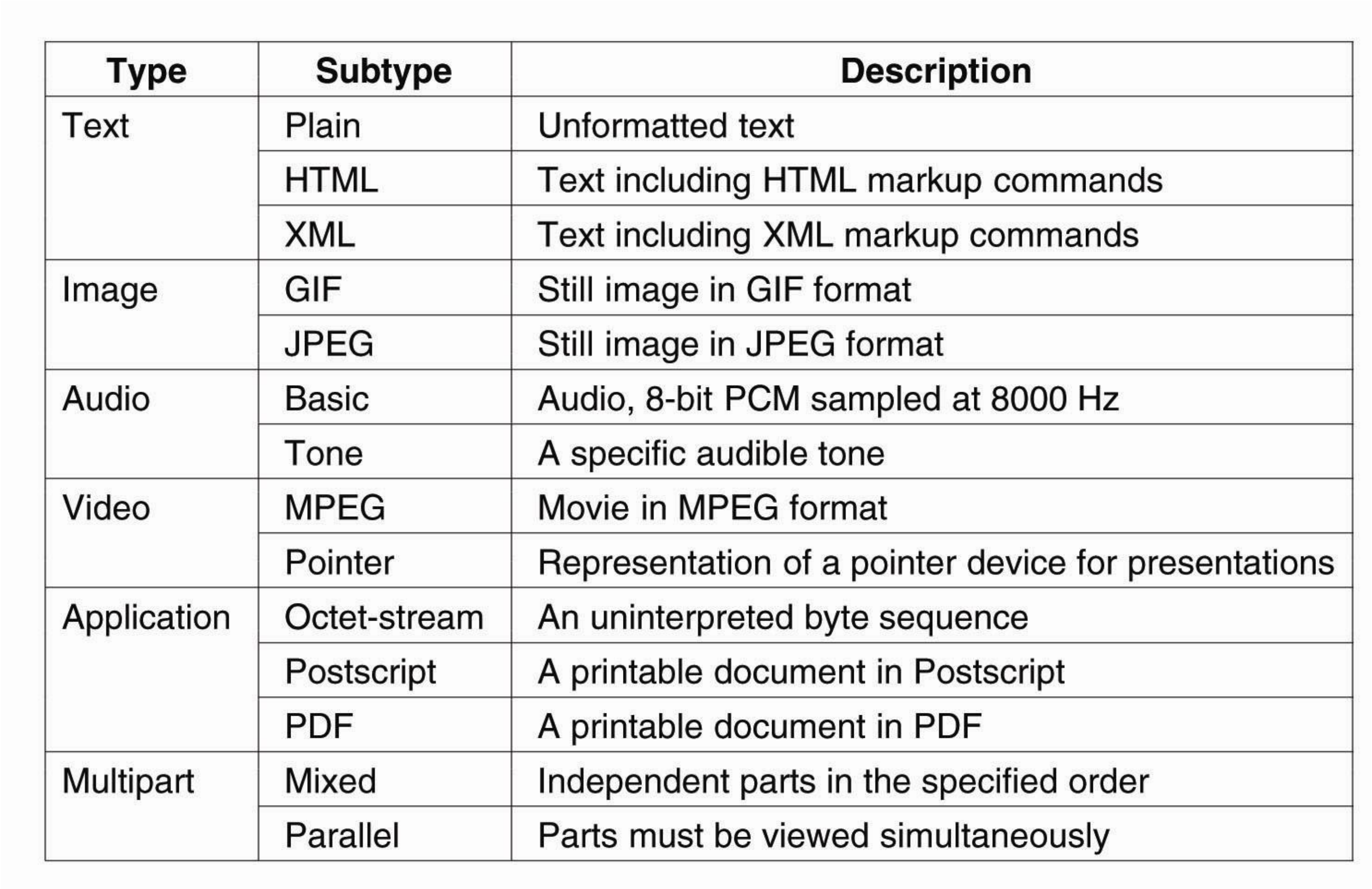
Telephone numbers

- 1-800-WIDGETS
- 1-415-765-4321

(b)

Web Documents

[Tanenbaum]



Type	Subtype	Description
Text	Plain	Unformatted text
	HTML	Text including HTML markup commands
	XML	Text including XML markup commands
Image	GIF	Still image in GIF format
	JPEG	Still image in JPEG format
Audio	Basic	Audio, 8-bit PCM sampled at 8000 Hz
	Tone	A specific audible tone
Video	MPEG	Movie in MPEG format
	Pointer	Representation of a pointer device for presentations
Application	Octet-stream	An uninterpreted byte sequence
	Postscript	A printable document in Postscript
	PDF	A printable document in PDF
Multipart	Mixed	Independent parts in the specified order
	Parallel	Parts must be viewed simultaneously

Six top-level MIME types and some common subtypes.

XML …

- XML is a text-based markup language that is fast becoming the standard for data interchange on the Web.
- XML has syntax analogous to HTML.
- Unlike HTML, XML tags tell you what the data *means*, rather than how to display it.
- Example:

```
<message>
        <to>you@yourAddress.com</to>
       <from>me@myAddress.com</from>
        <subject>XML Is Really Cool</subject>
        <text> How many ways is XML cool? Let me count the ways...
       </text>
 </message>
```

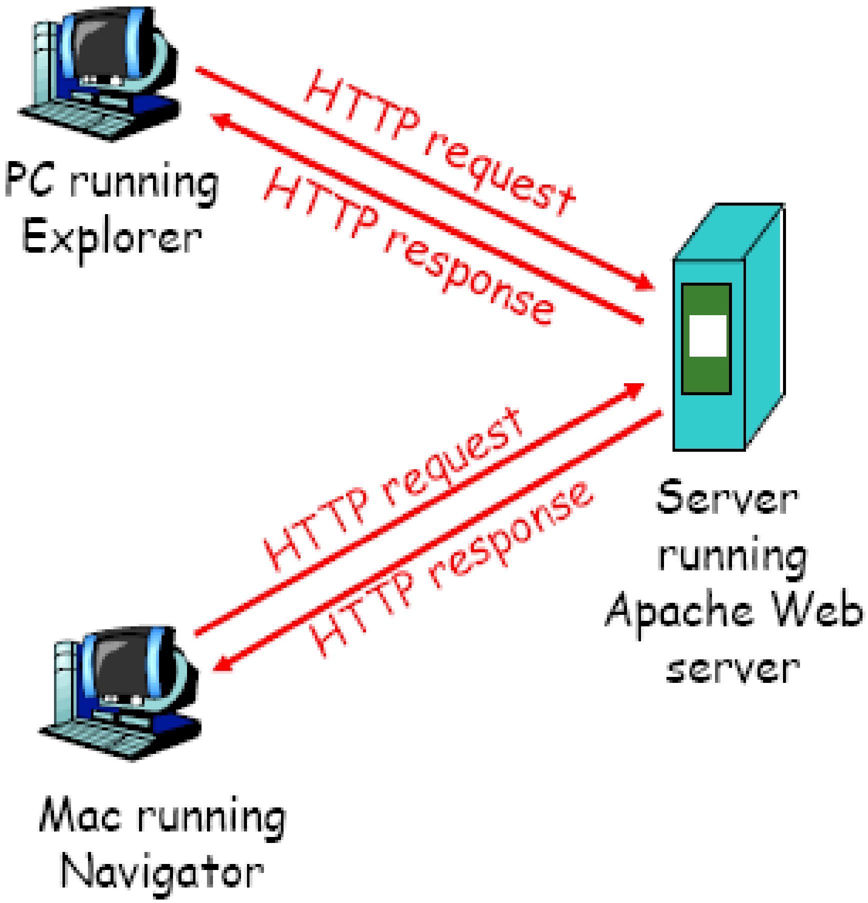
Taken from: http://java.sun.com/xml/docs/tutorial/overview/1_xml.html#intro

HTTP Overview

[this and following EECS122, Abhay Parekh]

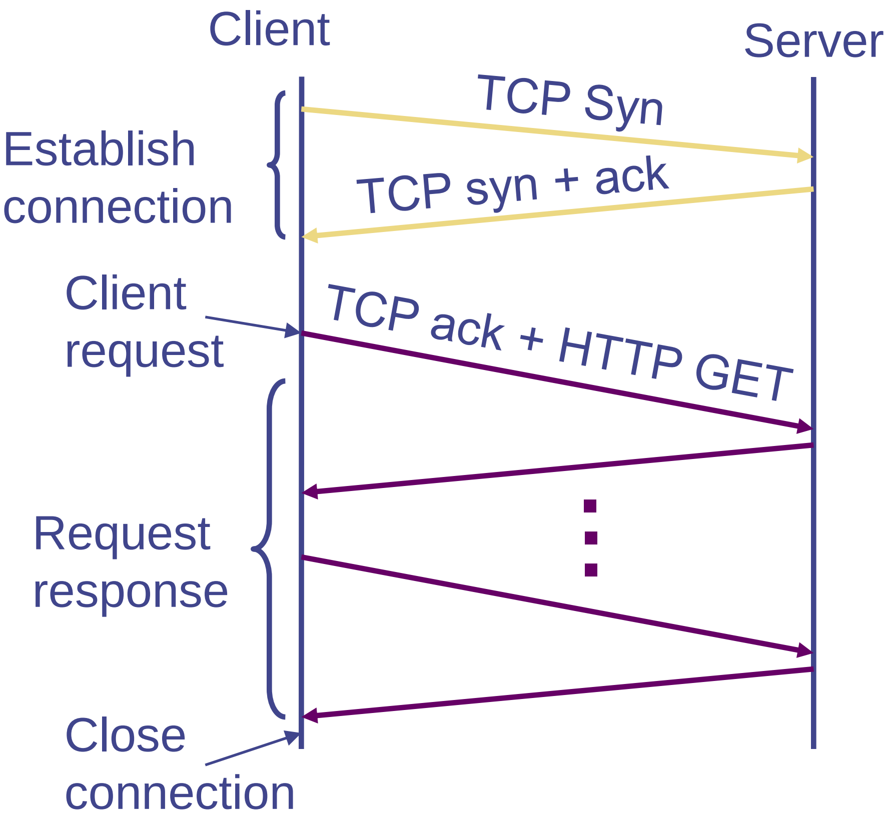
HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, "displays" Web objects
  - *server:* Web server sends objects in response to requests
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068

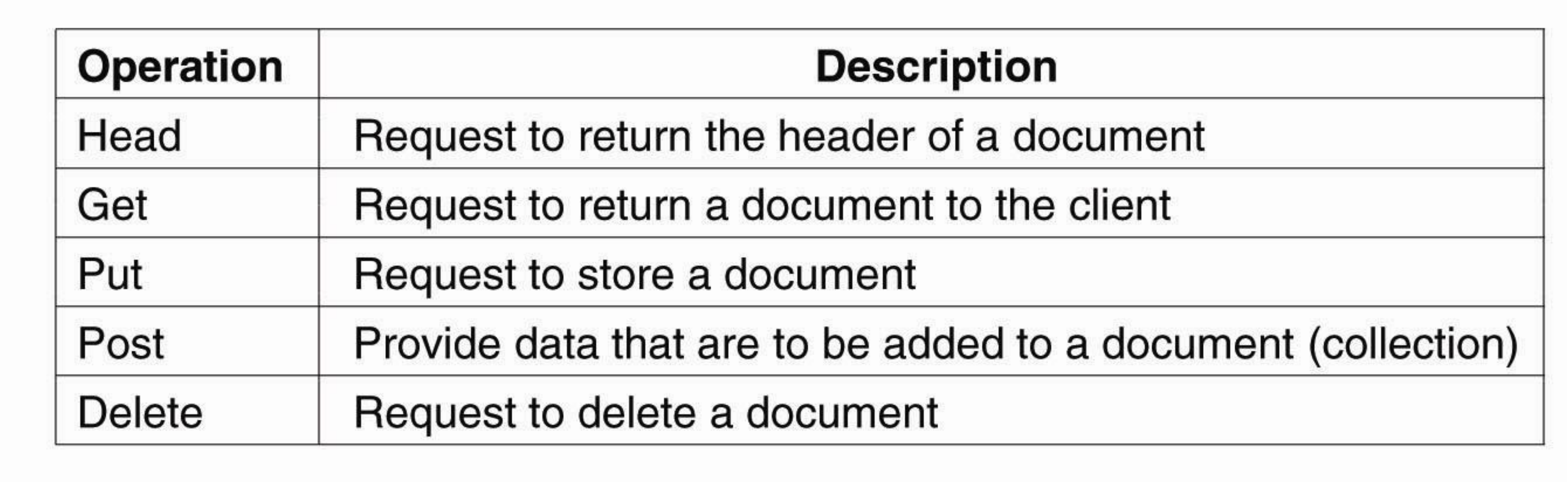


Note: The server is stateless, i.e. each request is self contained!

Big Picture - HTTP



HTTP Methods



Operation	Description
Head	Request to return the header of a document
Get	Request to return a document to the client
Put	Request to store a document
Post	Provide data that are to be added to a document (collection)
Delete	Request to delete a document

How does it work – Example

[Kurose and Ross]

- http://www.mylife.org/mypictures.htm
- After finding out the IP address of the host…(DNS!)

1. http client initiates a TCP connection on :80
2. Client sends the get request via socket established in 1
3. Server sends the html file, which is encapsulated in its response
4. http server tells tcp to terminate connection
5. http client receives the file and the browser parses it…contains ten jpeg images
6. Client repeats steps 1-4

Persistency of TCP usage

- A web page typically contains many objects
  - E.g. Images
  - Each object must be requested with a separate http "Get" command
  - Non Persistent Connection:
    - Different TCP connection for each object request.
    - HTTP 1.0
  - Persistent Connection
    - Reuse the same TCP connection for each object request
    - HTTP 1.1

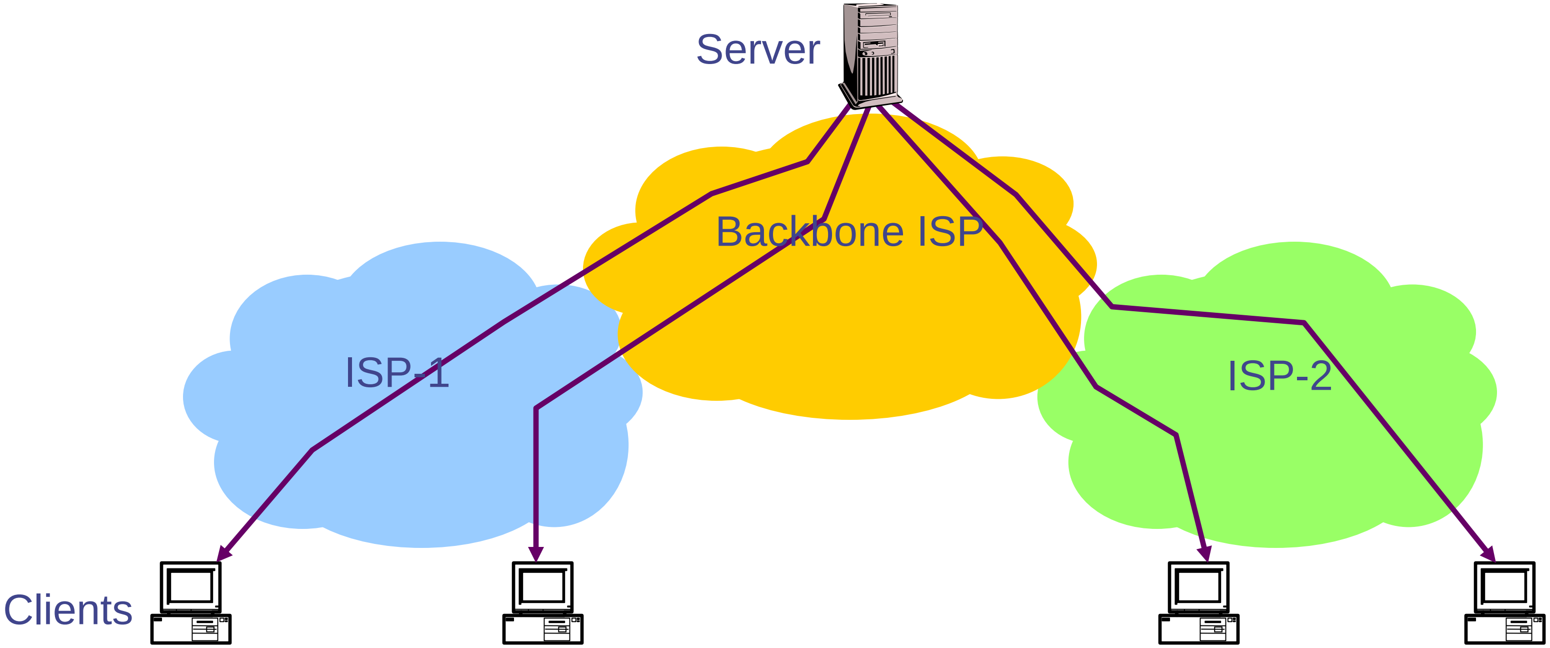
Performance and Reliability...

- Problem: You are a web content provider
  - How do you handle millions of web clients?
  - How do you ensure that all clients experience good performance?
  - How do you maintain availability in the presence of server and network failures?

- Solutions:
  - Add more servers at different locations → If you are CNN this might work!
  - Caching
  - Content Distribution Networks (Replication)

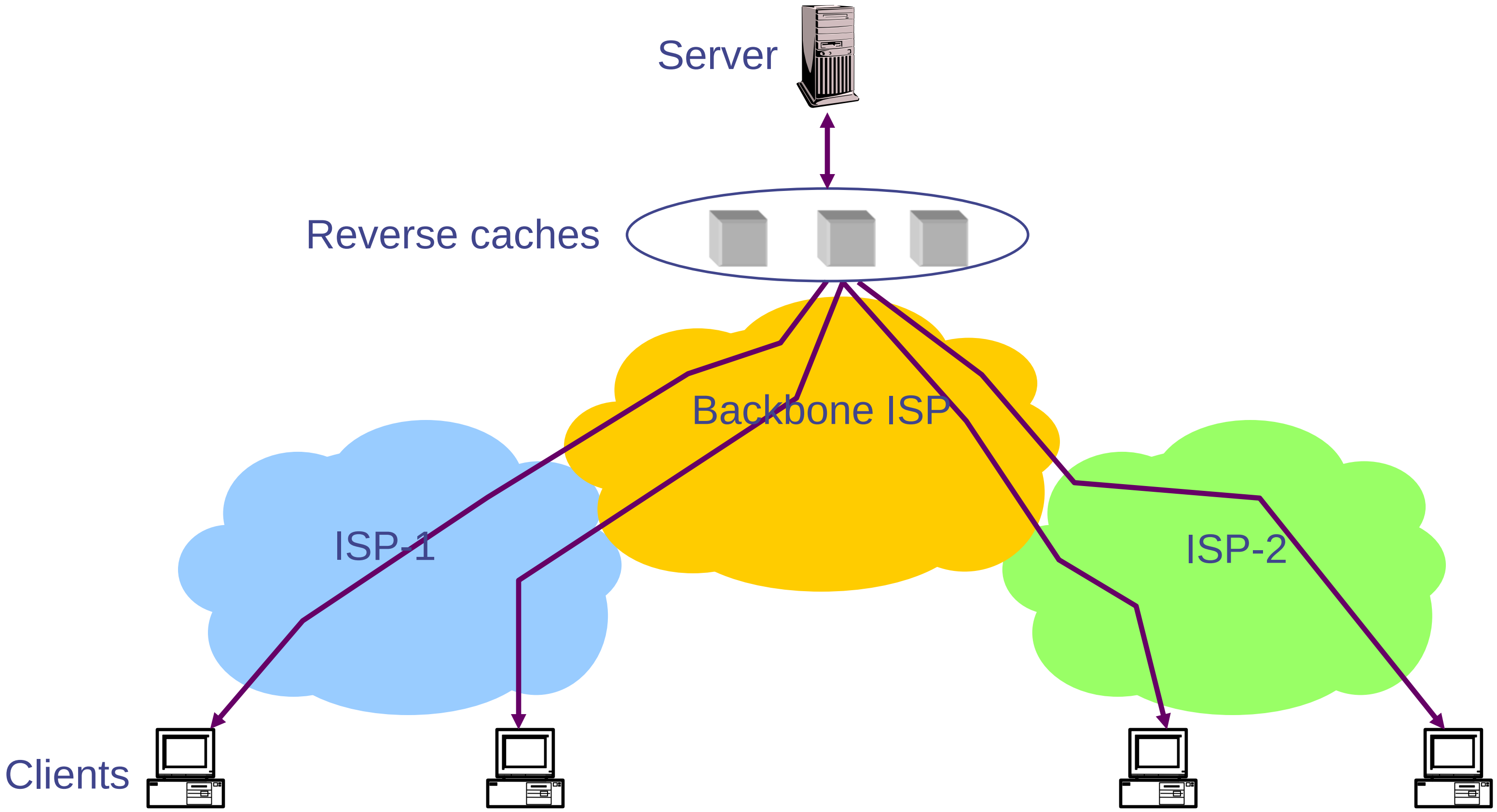
“Base-line”

- Many clients transfer same information
  - Generate unnecessary server and network load
  - Clients experience unnecessary latency



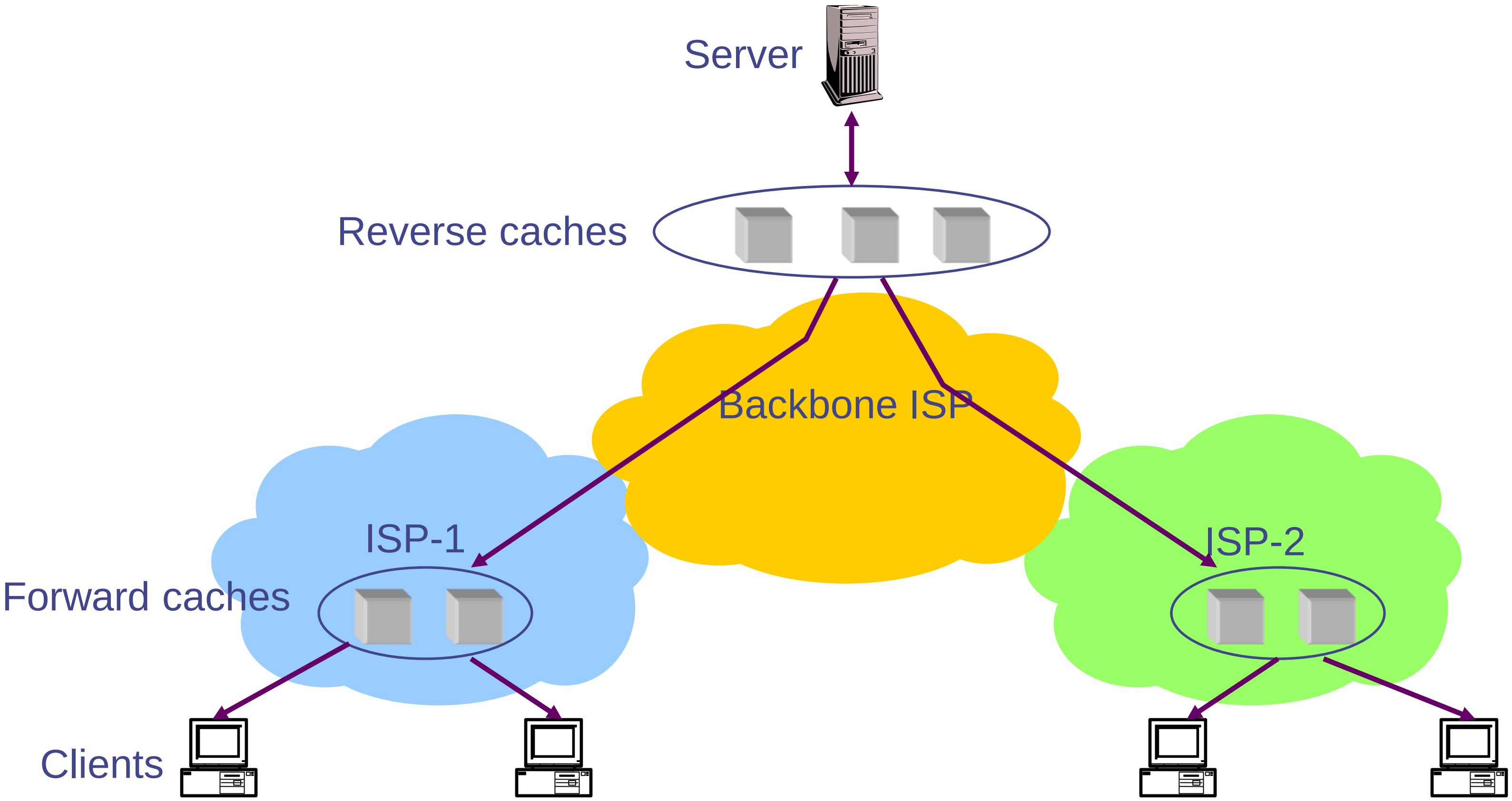
Reverse Caches

- Cache documents close to server → decrease server load
- Typically done by content providers



Forward Proxies

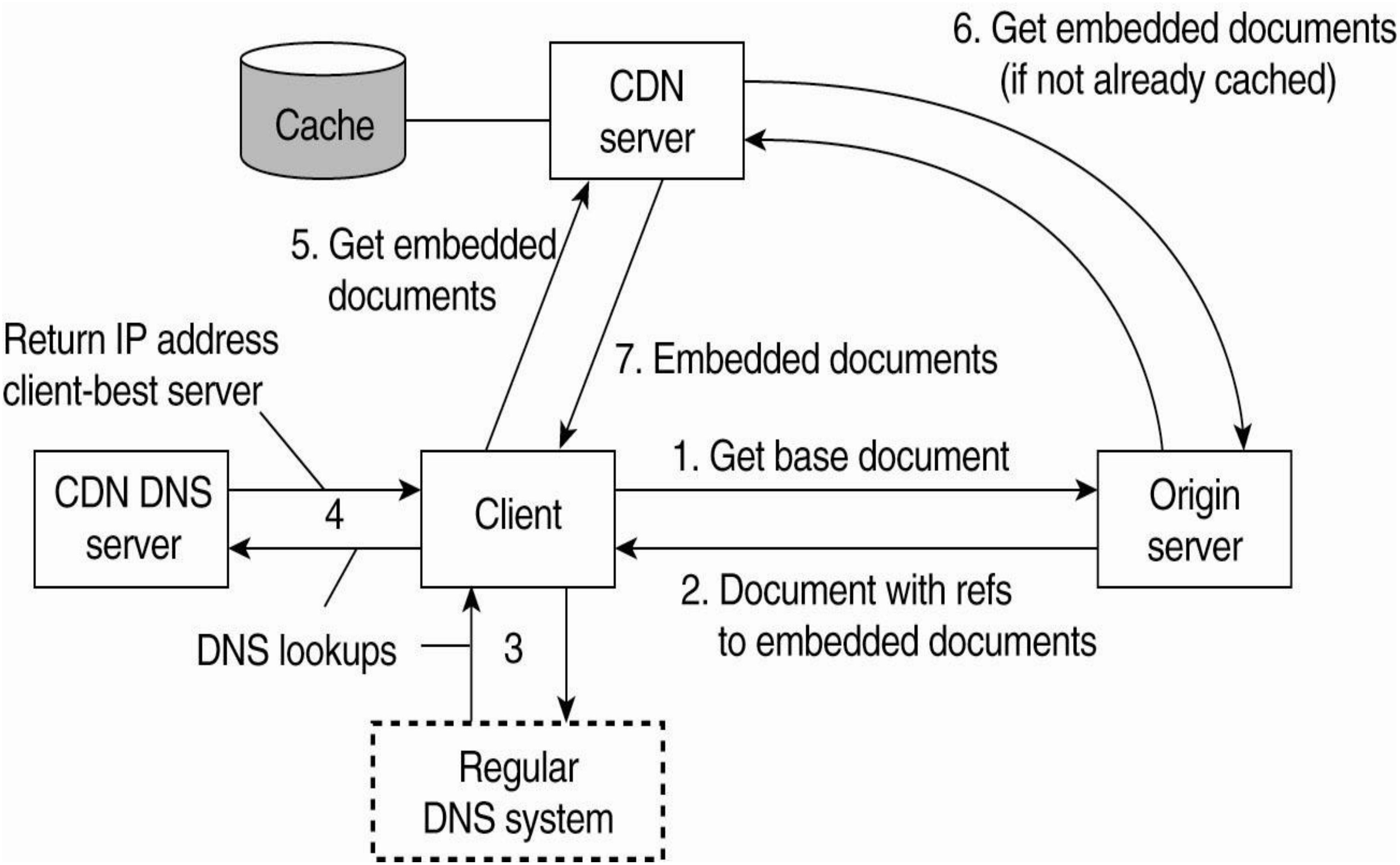
- Cache documents close to clients → reduce network traffic and decrease latency
- Typically done by ISPs or corporate LANs



Client Side - HTTP/1.0 Caching Support

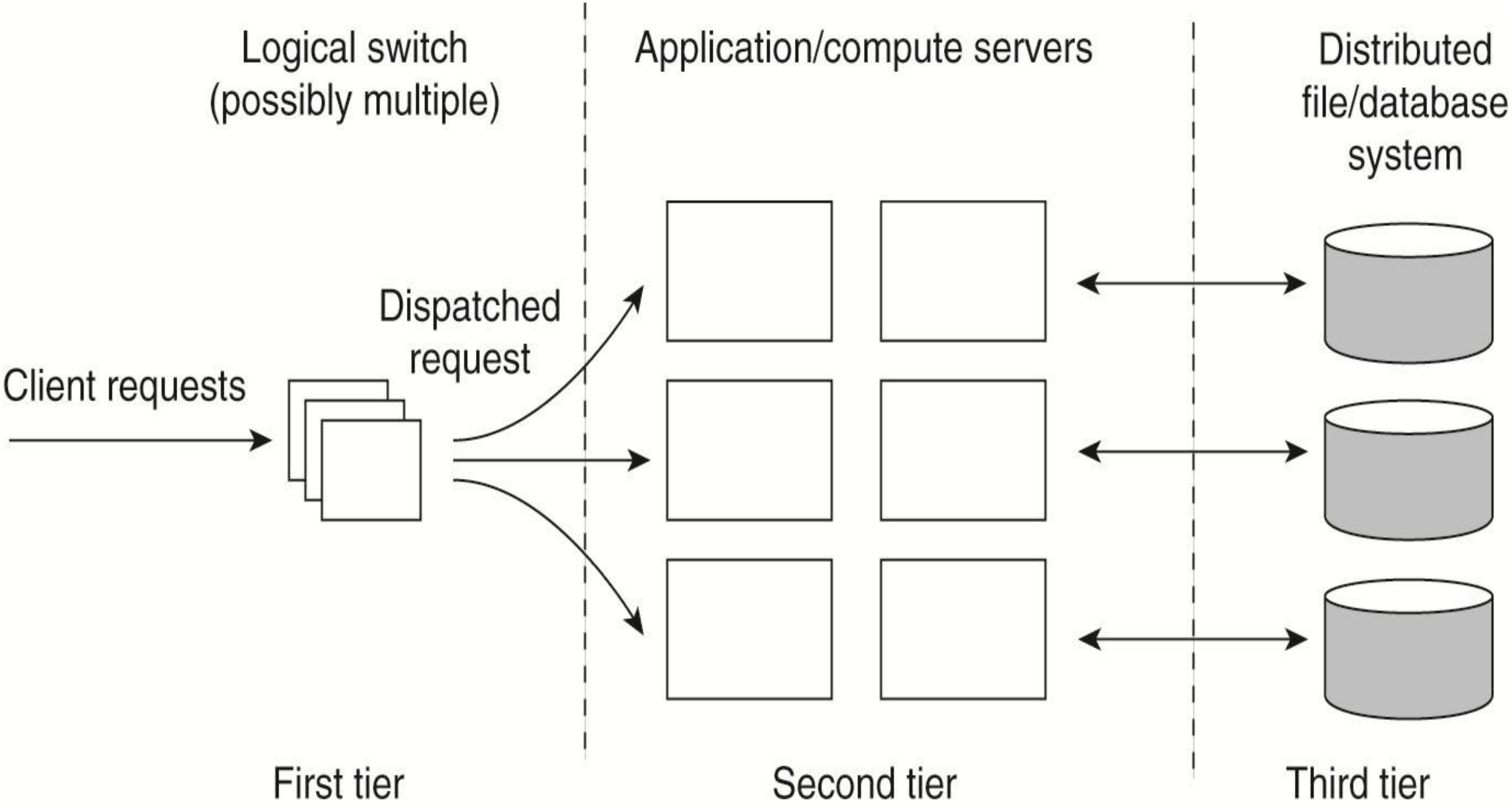
- Exploit locality of reference
- A modifier to the GET request:
  - If-modified-since – return a “not modified” response if resource was not modified since specified time
- A response header:
  - Expires – specify to the client for how long it is safe to cache the resource
- A request directive:
  - No-cache – ignore all caches and get resource directly from server
- These features can be best taken advantage of with HTTP proxies
  - Locality of reference increases if many clients share a proxy

Redirection: The Akamai Content Distribution Network (CDN)


6. Get embedded documents (if not already cached)
Cache
CDN server
5. Get embedded documents
Return IP address client-best server
7. Embedded documents
1. Get base document
CDN DNS server
4
Client
Origin server
2. Document with refs to embedded documents
DNS lookups
3
Regular DNS system

Operation on data basis: Three-tiered Server Clusters

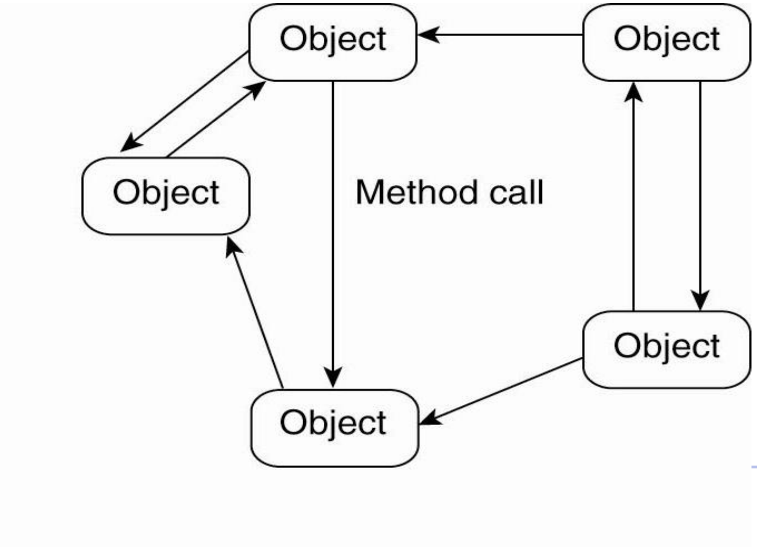
Web servers do frequently manipulate data bases (e.g. inventory, orders). Separating Web serving from data base helps keeping consistency…



A “normal” **object model**

[Karl, Paderborn – modified]

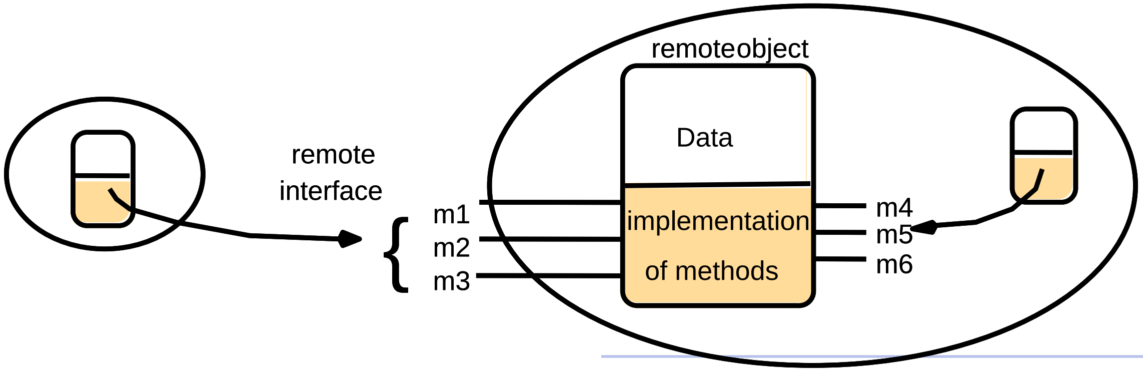
- Think: Java, C++, …)
- **Program**: Collection of objects, each with data & methods, interacting with each other by means of method invocations
- **Object references**: Necessary to access (identify) any object
- **Interfaces**: Defines signature of a set of methods,
- **Action**: Invoke a method; effects: state change, new instance created, further method invocations
- **Exceptions**
- **Garbage collection**



Distributed object model

[Karl, Paderborn]

- Interacting objects no longer are in a single process, but distributed onto several processes
- Requires appropriate notions of
  - **Remote object reference**: Unique identifier of an instance in the entire distributed system, not only within a single process
    - Might contain: IP address, process ID, object ID within this process
  - **Remote interface**: Contains methods that are remotely invocable
    - Typically specified in a programming-language-neutral form to allow invocations from other programming languages – an **interface definition language** (IDL)

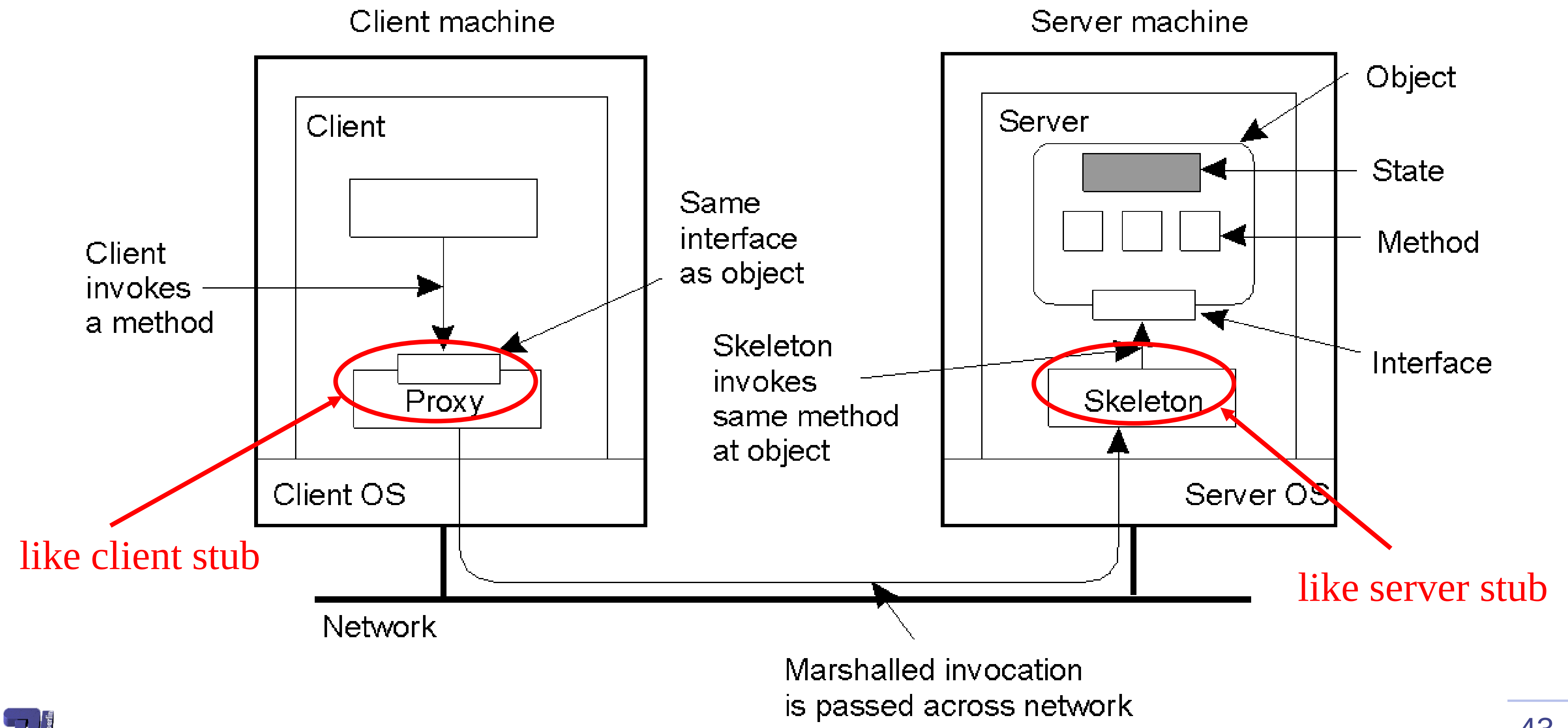


Distributed Objects - Similar to RPC…

In detail: Middleware needs to provide some extra functionality to deal with remote object references, details of object lifetime,…

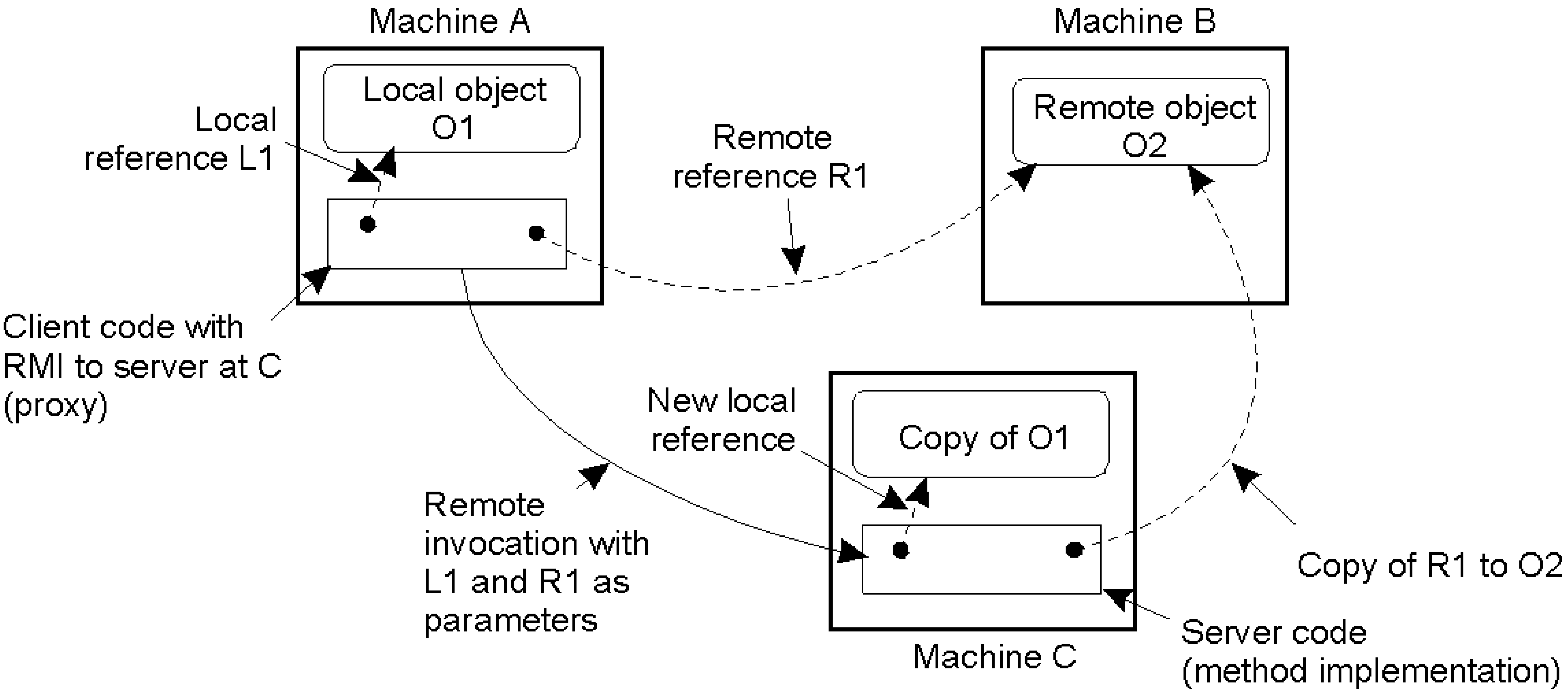
E.g., how to obtain a remote object reference in the first place?

A *bindery* service (e.g., RMIregistry)



Parameter Passing

- The situation when passing an object by reference or by value
  - Copy local object
  - Send only reference to remote object

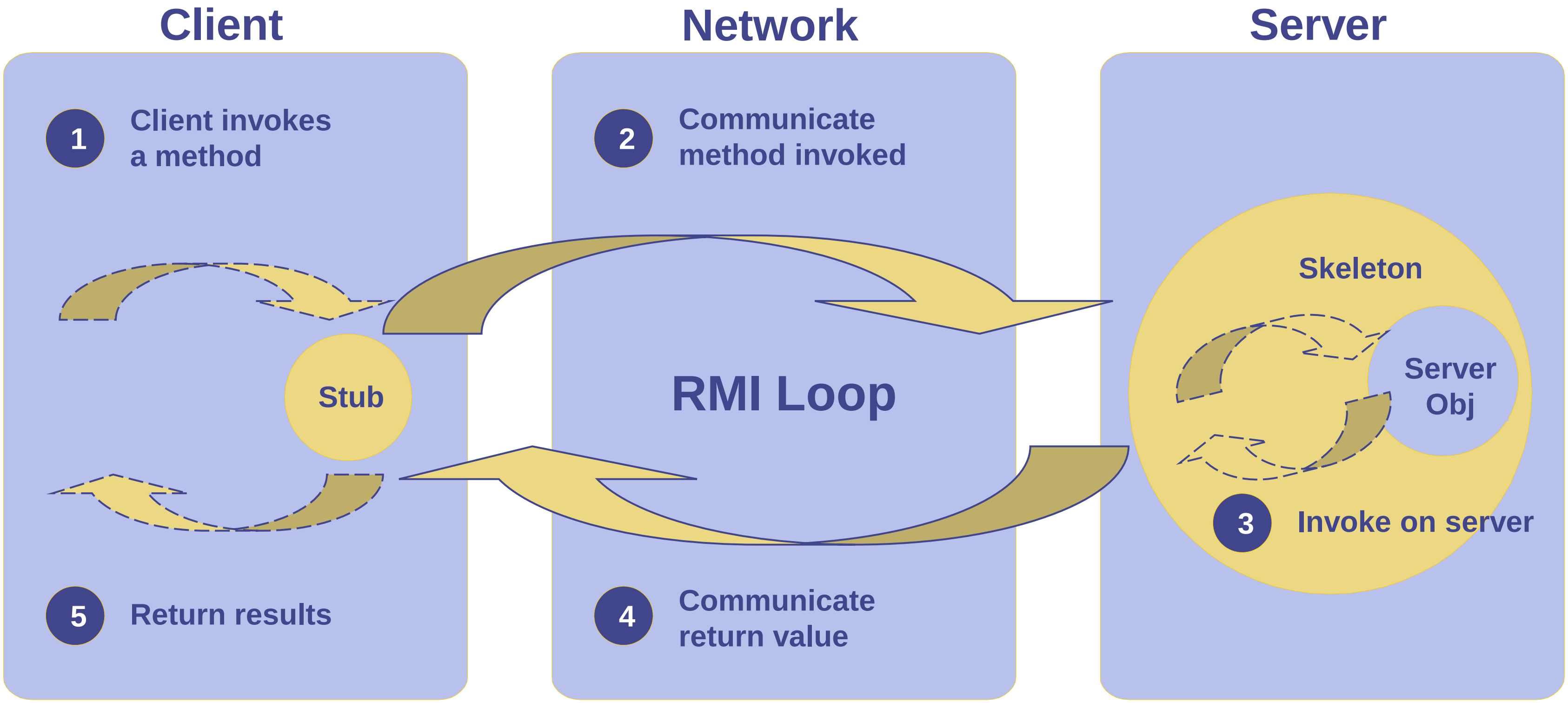


Example: Java RMI

[Lauer, WPI]

- Based on objects & methods
  - I.e., an object may be located remotely
  - Accessed via methods
  - RMI causes methods to be invoked by RPC
- Stubs may be local or remote
  - Subject to security policy
- Remote objects *must* be registered with RMI registry
  - May specify a *port*
  - Security policy compiled with object
- Client looks up object via *Naming* method
  - Specifies *host* (& *port*, if needed)
  - Remote stubs delivered with *lookup*, subject to security policy

RMI-Loop


Client
Network
Server
1
Client invokes a method
2
Communicate method invoked
Skeleton
Server Obj
Stub
RMI Loop
3
Invoke on server
5
Return results
4
Communicate return value

Java Remote Object details

[Lauer, WPI]

- Interface must extend `java.RMI.Remote`
  - Must be public
  - No new methods introduced by `java.RMI.Remote`
- Should also extend
  - `java.rmi.server.UnicastRemoteObject`
  - Replaces some Object class methods for proper operation in distributed environment
- Must throw `java.rmi.RemoteException`
  - May impact definition of existing interfaces
- Compiling object and stubs
  - Object compiled to bytecodes with `javac`
  - Stubs compiled with `rmic`
- RMI registry *must* be running before remote object can be launched
  - `rmiregistry` command (part of JDK distribution)
  - Multiple registries possible on different ports
- Remote object registers itself
  - `java.rmi.Naming` interface

Java Client details

[Lauer, WPI]

- Initialize RMI security manager
  - **`System.setSecurityManager`**(…)
  - If no security manager, only local stubs may be loaded
- Client must find and bind to remote object
  - **`lookup`**() method in **`java.RMI.Naming`**
  - Location of remote object specified in URL style
    - – E.g., “rmi://garden.wpi.edu:1099/ObjectName”
  - Remote stubs loaded by **`lookup`**
    - – Per security policy

Another example - CORBA

- Common Object Request Broker Architecture (**CORBA**) specification defines a framework for object-oriented distributed applications..
- It is an open standard for heterogeneous computing.
- Allows distributed programs in different languages and different platforms to interact as though they were in a single programming language on one computer
- Each CORBA object has a clearly defined interface specified in CORBA **interface definition language (IDL).**
- Distributed objects are identified by object references, which are typed by the IDL interfaces.
- The interface definition specifies the member functions available to the client without any assumption about the implementation of the object.

What are Web Services?

- WWW makes services available to humans…. – not easy to access web sites from a program for further processing
- A Web service is a collection of functions that are packaged as a single entity and published to the network for use by other programs.
- Web services are building blocks for creating open distributed systems, and allow companies and individuals to quickly and cheaply make their digital assets available worldwide.
- A Web service can aggregate other Web services to provide a higher-level set of features

Basic blocks of Web Services…

- **UDDI (universal description, discovery and integration)**

  Services have to be discovered - http://uddi.xml.org/

- **WSDL (web services description language)**

  Interfaces have to be described - http://www.w3.org/TR/wsdl

- **SOAP (simple object access protocol)**

  (remote) objects access - http://www.w3.org/TR/soap/

- **XML (Extensible Markup Language)**

  data description format - http://www.w3.org/XML/

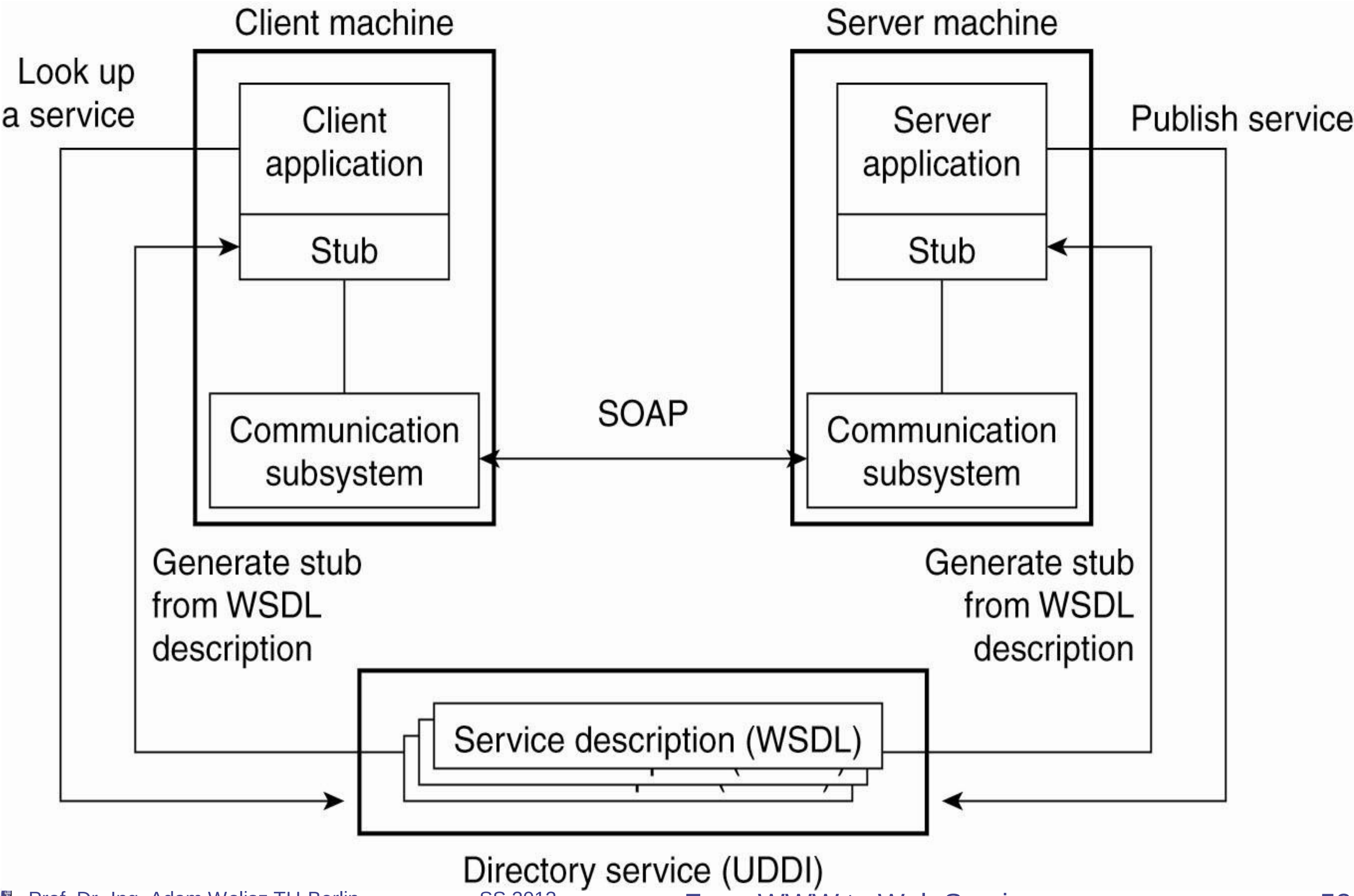
- **HTTP (Hyper Text Transfer Protocol)**

  communication layer - http://www.w3.org/Protocols/

see also: http://www.w3schools.com/default.asp

Web Services Fundamentals

[Tannenbaum]

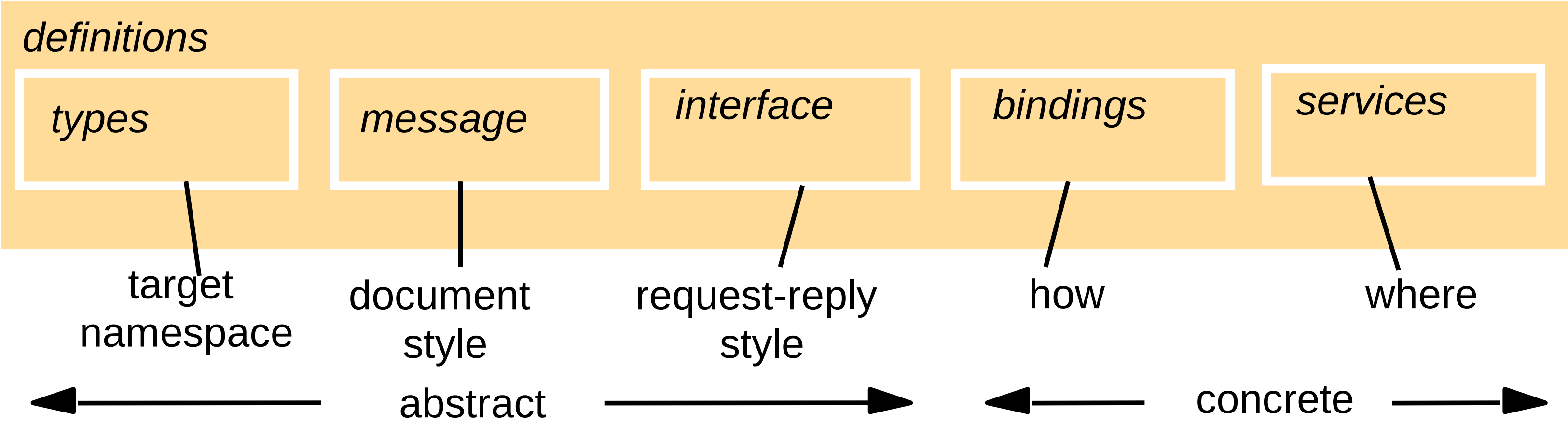


UDDI

- UDDI is used to register and look up services with a central registry
  - Service Providers advertise their business services
  - Service consumers can look up UDDI-entries
  - UDDI Parts:
    - White pages: Business information (Name, contact, description,...)
    - Yellow pages: Service information
    - Green pages: Technical information (Access point, WSDL reference)
- UDDI-registry:
  - Distributed system (!) of individual UDDI-Servers
  - XML-based; Stores descriptions, provides WSDL
- Initial vision: “[…] help companies conduct business with each other in an automated fashion [...]” [sys-con.com]
- Reality today: Human element stays important ->UDDI not very widespread
- *Followed by: Business Process Execution Language (BPEL)*
  - *Specification of business processes based on Web Services*

Web Service Description Language (WSDL)

- Interface specification for web services
  - Akin to interface definition languages for RPC, RMI
  - Written in XML to be programming-language-agnostic
  - Also includes how and where (URI) a service can be invoked
- Main elements of WSDL description
  - Abstract: which compound types are used, combined into which messages
  - Concrete: How and where is the service to be contacted?



WSDL Definitions

[Karl, Paderborn]

- *Types*: First define which data types are going to be exchanged between participants
  - Use existing XML-based type system
- *Message*: Define which kinds of messages can be sent between different entities
  - Which data types are included in which message
  - These are abstract messages, no reference to how these messages are represented on the wire

WSDL Definitions

[Karl, Paderborn]

- *Port types*: A set of supported operations form the type of a port
  - *Operation*: An operation is a specification which abstract message type is sent and which one is received
  - Four kinds of operations exist:
    - One-way: entity only receives a message
    - Request/response: entity receives a messages and answers with a message
    - Solicit/response: entity sends a message and receives an answer
    - Notification: entity only sends a message
- *Binding*: As a port type is still an abstract concept, a mapping to a single, real protocol has to be specified
  - Message format, protocol details
  - Typical bindings: SOAP, HTTP GET/POST
  - Bindings must not include address information

WSDL Definitions

[Karl, Paderborn]

- *Service*: Ports can be grouped into services
  - *Port*: A real port is then a binding with an address
    - Hence: an address where a number of operations can be invoked, along with the protocol information how to do so
  - Ports within a service do not communicate with each other
  - Service can contain several ports with the same port type, but different bindings -> alternative ways to access the same functionality using different protocols

Simple Object Access Protocol (SOAP)

- High-level communication protocol
  - Mostly: request/reply semantics (“RPC-style”)
  - But also: document exchange style
  - More precisely: SOAP defines message formats, not the protocol as such
  - Relies on the HTTP POST message for actually delivery
- Between applications
  - So far, we discussed RPC to achieve this
  - RPC “disadvantage”: compatibility problem, security (firewalls/proxy servers)
- Idea: Use RPC principles, but
  - Define a common representation of data
  - Use a generally available transport protocol: mostly HTTP
    - E.g., to traverse firewalls
    - Implementations using other protocols (e.g. SMTP) exist!
  - Use XML to represent data (plain text data representation)

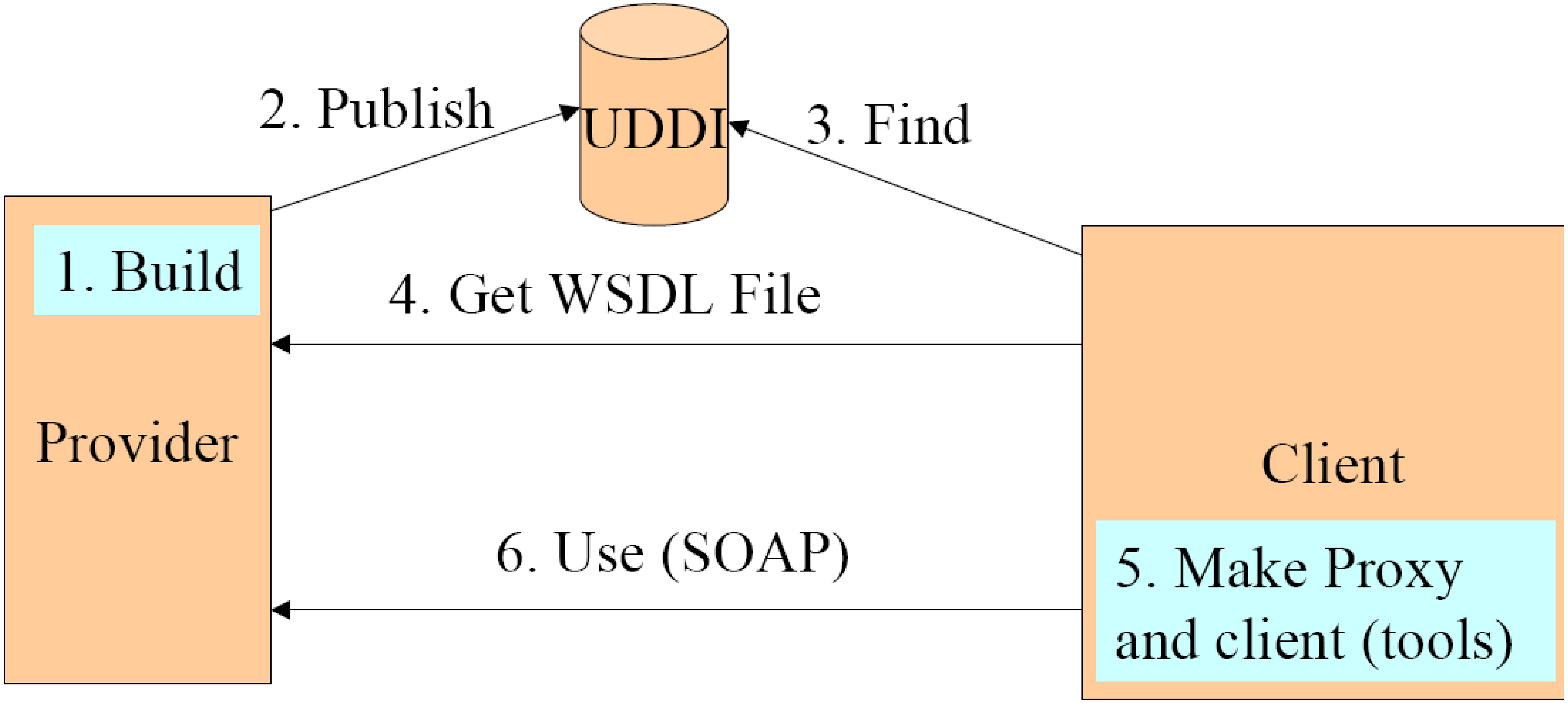
SOAP (cont.'d)

[Karl, Paderborn]

- Main point: The interface of the service to which the address is sent need not be known!
  - Restrictions can be expressed with attributes like mustUnderstand
- How the service is implemented is irrelevant – it only needs to be able to process HTTP and XML

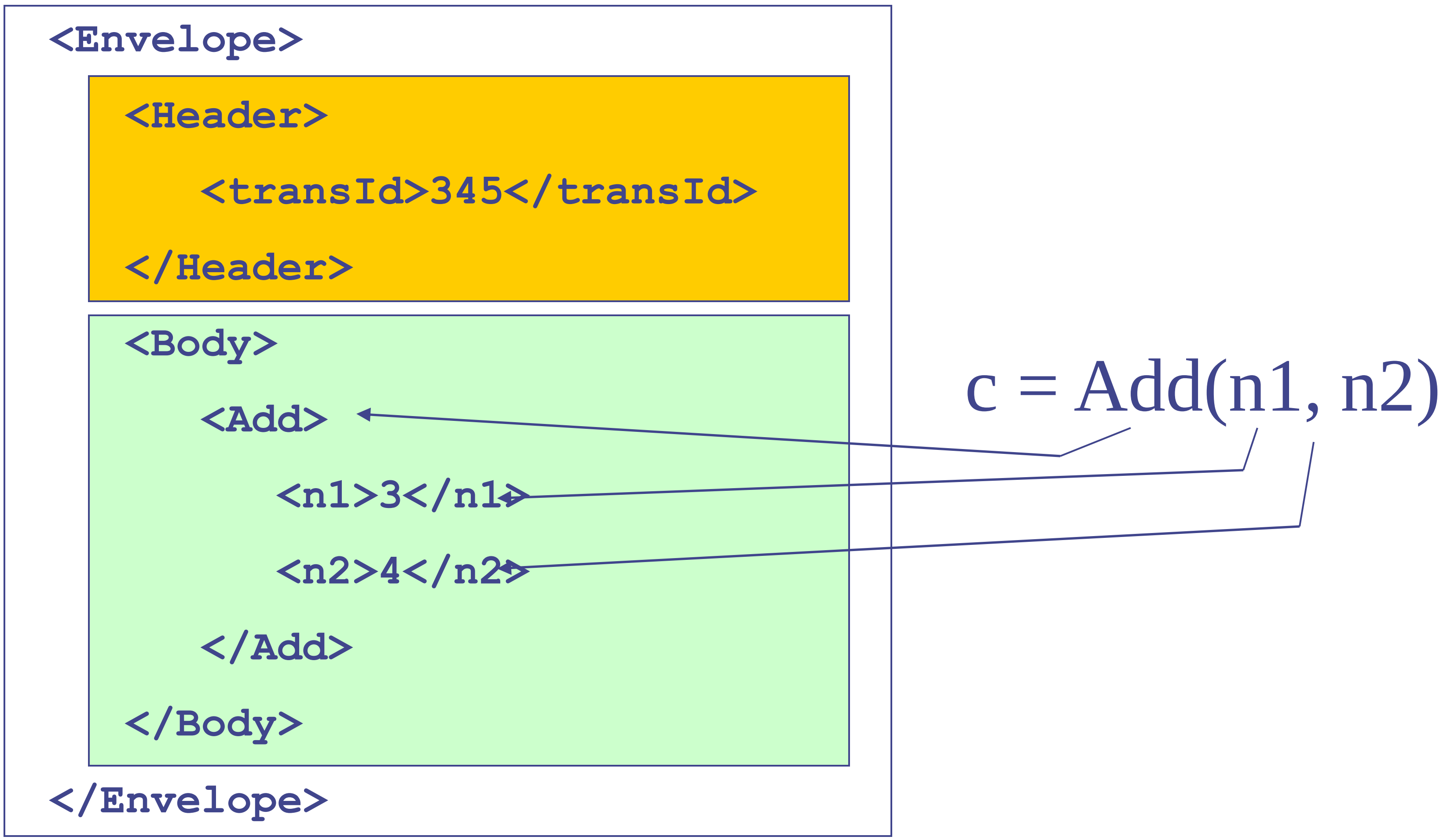
Interaction

[van Thanh]



Simple Example

[Shenker, Stoica]



```
<Envelope>
   <Header>
      <transId>345</transId>
   </Header>
   <Body>
      <Add>
         <n1>3</n1>
         <n2>4</n2>
      </Add>
   </Body>
</Envelope>
```

c = Add(n1, n2)

SOAP Request

[Shenker, Stoica]

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV=“http://schemas.xmlsoap.org/soap/envelope/”
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/”>
   <SOAP-ENV:Header>
      <t:transId xmlns:t=“http://a.com/trans”>345</t:transId>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <m:Add xmlns:m=“http://a.com/Calculator”>
         <n1>3</n1>
         <n2>4</n2>
      </m:Add>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Request

[Shenker, Stoica]

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV=“http://schemas.xmlsoap.org/soap/envelope/”
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/”>
   <SOAP-ENV:Header>
      <t:transId xmlns:t=“http://a
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <m:Add xmlns:m=“http://a
         <n1>3</n1>
         <n2>4</n2>
      </m:Add>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Scopes the message to the SOAP namespace describing the SOAP envelope

Establishes the type of encoding that is used within the message (different data types supported)

SOAP Request

[Shenker, Stoica]

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV=“http://schemas.xmlsoap.org/soap/envelope/”
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/”>
   <SOAP-ENV:Header>
      <t:transId xmlns:t=“http://a.com/trans”>345</t:transId>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <m:Add xmlns:m=“http://a.com/Calculator”>
         <n1>3</n1>
         <n2>4</n2>
      </m:Add>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Qualifies transaction Id

Defines the method

SOAP Response

[Shenker, Stoica]

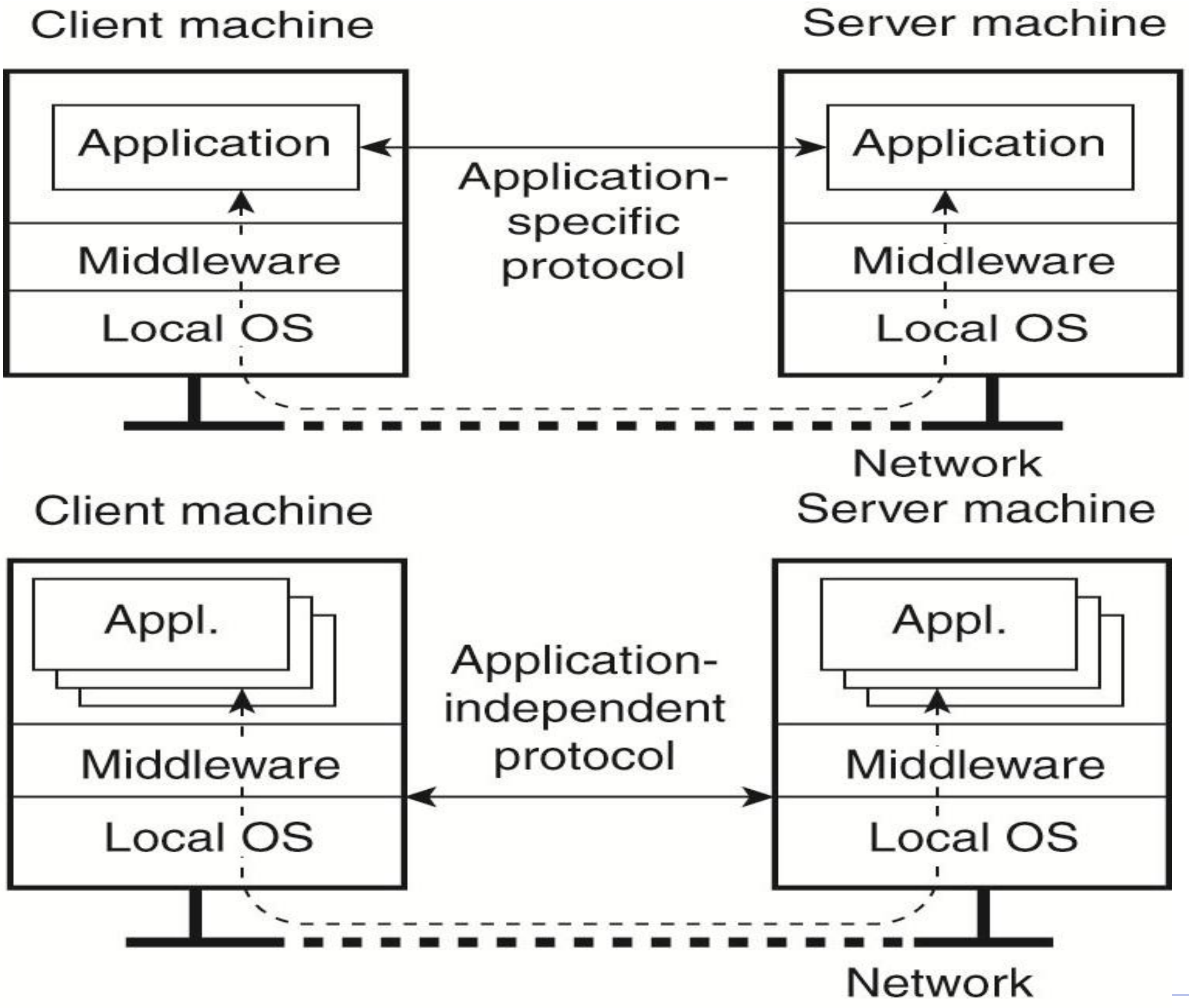
```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV=“http://schemas.xmlsoap.org/soap/envelope/”
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encodin
   <SOAP-ENV:Header>
      <t:transId xmlns:t=“http://a.com/trans”>345</t:transId>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <m:AddResponse xmlns:m=“http://a.com/Calculator”>
         <result>7</result>
      </m:AddResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response typically uses method name with “Response” appended

Examples

- Several popular sites provide Web services
  - Yahoo, google, ebay, Amazon, …
- Example: Access to Amazon from within a program
  - See http://aws.amazon.com for details
  - Wrappers for several programming languages available

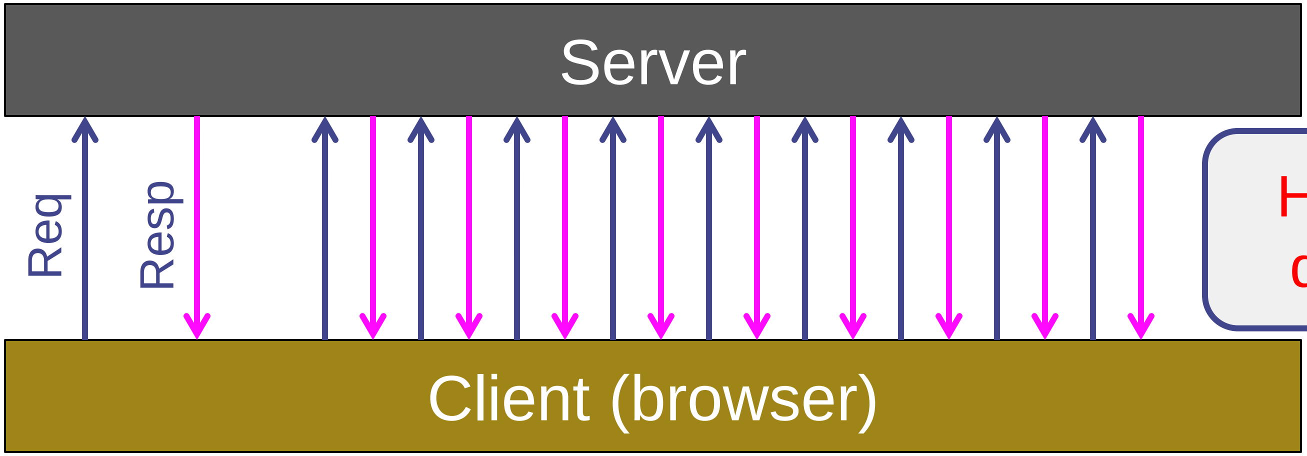
Special vs. general solutions



(b)

HTML 5 WebSocket API

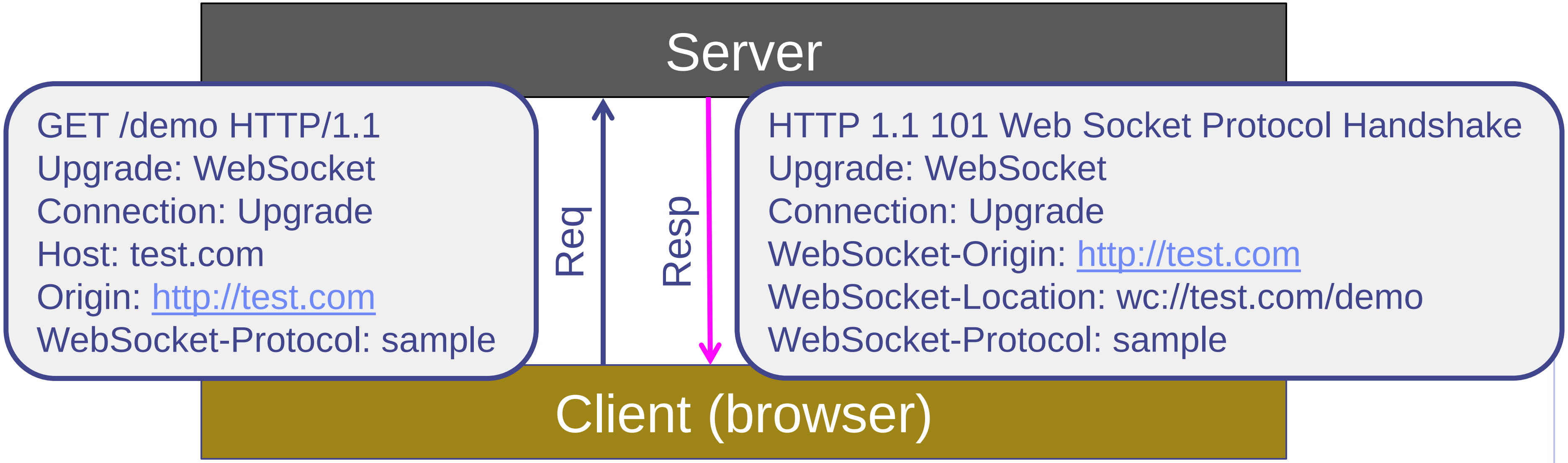
- Why do we need WebSockets?
  - Web apps demand real-time, event-driven communication with minimal latency
    - E.g. financial applications, online games, …
  - Problems with HTTP
    - HTTP is half-duplex (traffic flow in only one direction at a time)
    - HTTP adds latency
- Typical use case: polling (AJAX)
  - Poll server for updates, wait at client



High http meta-data overhead

What is a WebSocket (WS)?

- W3C/IETF standard
- Uses Websocket protocol instead of HTTP
  - ws:// and wss://
- True full-duplex communication channel
  - Strings + binary frames sent in any direction at the same time
- Uses port 80/443 (-> proxy/firewall)
- Connection established by „**upgrading**“ from HTTP to Websocket protocol



```
GET /demo HTTP/1.1
Upgrade: WebSocket
Connection: Upgrade
Host: test.com
Origin: http://test.com
WebSocket-Protocol: sample
```

```
HTTP 1.1 101 Web Socket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
WebSocket-Origin: http://test.com
WebSocket-Location: wc://test.com/demo
WebSocket-Protocol: sample
```

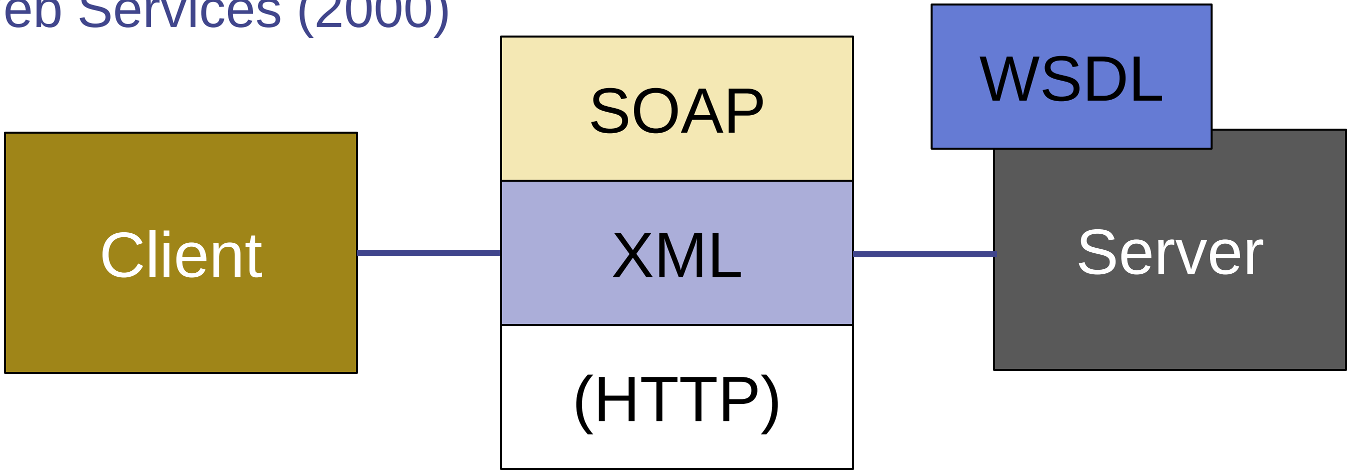
WS - How Network Traffic is Reduced

- Each message frame has only 2 Bytes of overhead
- No latency from establishing new TCP connection for each HTTP message
- No polling overhead – only send messages when there is something to send
- Usage: e.g. Javascript client

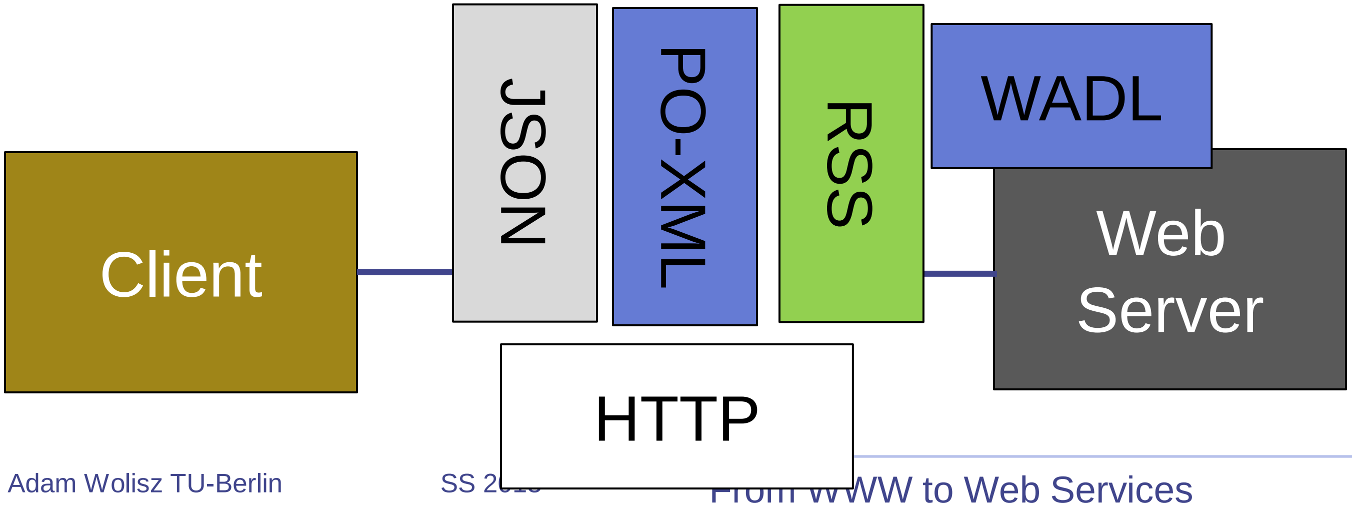
```
connect: function() {
        try {
                this.ws = new WebSocket('ws://www.test.com ');
                this.ws.onopen = function (event) { /* … */ };
                this.ws.onclose = function (event) { /* … */ };
                this.ws.onmessage = messageListener;
        } catch (exception) {}
},
messageListener: function(event) {
        alert('New message: ' + event.data);
},
send: function(message) {
        if (this.ws) {  this.ws.send(message);  }
},
```

RESTful - Representational State Transfer

- Comparing REST vs. SOAP / Web Services
  - Web Services (2000)



  - RESTful Web Services (2006)



RESTful [Baker]

- RESTful
  - REST is an architectural style for distributed systems.
- An architectural style is:
  - … an abstraction, a design pattern, a way of discussing an architecture without concern for its implementation.
- REST defines a series of constraints for distributed systems that together achieve the properties of:
  - Simplicity, Scalability, modifiable, performance, visibility (to monitoring), portability and reliability.
- A system that exhibits all defined constraints is RESTful!
- Protocol Layering
  - HTTP = Application-level Protocol (REST)
  - HTTP = Transport-level Protocol (WS-*)

Understanding REST [Baker]

- Representational State Transfer:
  - The Resource:
    - A resource is any information that can be named: documents, images, services, people, an collections.
  - Resources have state:
    - State may change over time.
  - Resources have identifiers:
    - A resource is anything important enough to be referenced.
  - Resources expose a uniform interface:
    - System architecture simplified, visibility improved,
    - Encourages independent evolution of implementations.

Understanding REST (2) [Baker]

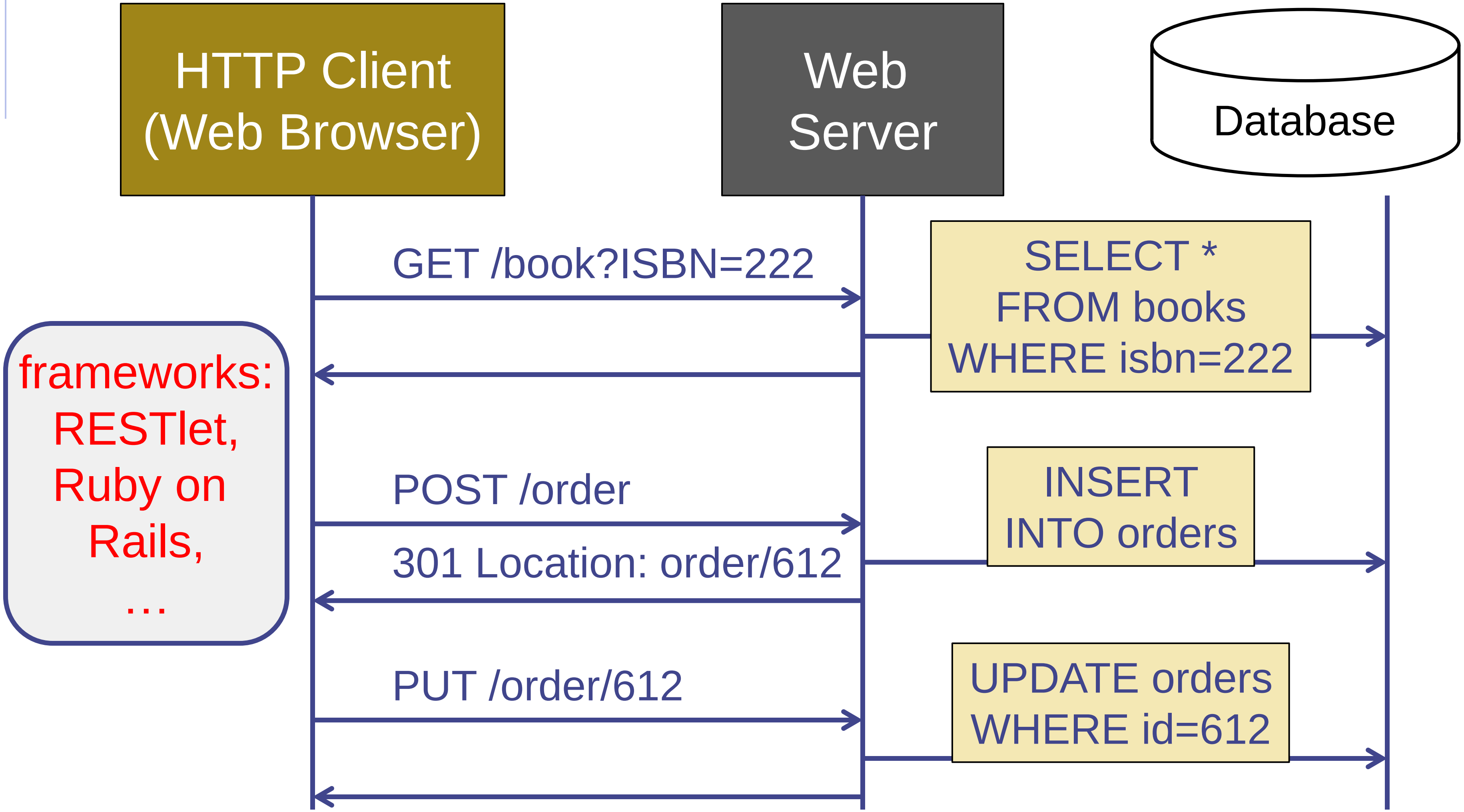
- Representational State Transfer:
  - On request, a resource may transfer a representation of its state to a client:
    - Necessitates a client-server architecture.
  - A client may transfer a proposed representation to a resource:
    - Manipulation of resources through representations.
  - Representations returned from the server should link to additional application state:
    - Clients may follow a proposed link.

Understanding REST (3) [Baker]

- Representational State Transfer:
  - Stateless interactions:
    - Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server.
  - Statelessness necessitates self-descriptive messages:
    - Standard media types,
    - Meta-data and control-data.
  - Uniform interface + Stateless + Self-descriptive = Cacheable

RESTful

- Example:



HTTP Client (Web Browser)

Web Server

Database

GET /book?ISBN=222

SELECT * FROM books WHERE isbn=222

POST /order

INSERT INTO orders

301 Location: order/612

PUT /order/612

UPDATE orders WHERE id=612

frameworks: RESTlet, Ruby on Rails, …