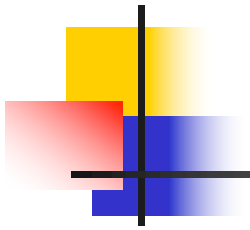
Introduction to Web Services



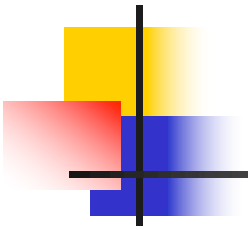
Outline

- A bit of history:
  - RPC,
  - CORBA, and
  - RMI
- Web Services:
  - SOAP (very brief)

My slides have been supplemented by material from various sources, including: W3C, Lillian Cassel (Villanova), Chiyoung Seo (USC), Alfred C. Weaver (UVa)

Communicating systems

- Researchers in the computing field have long worked on ways to send and receive data to between computers.
- ARPANET, a network connecting computers provided by the Advanced Research Project Agency (ARPA) has been introduced as one of the initial answers to this problem, in the 1960's.
- Its successor, the Internet, followed soon.

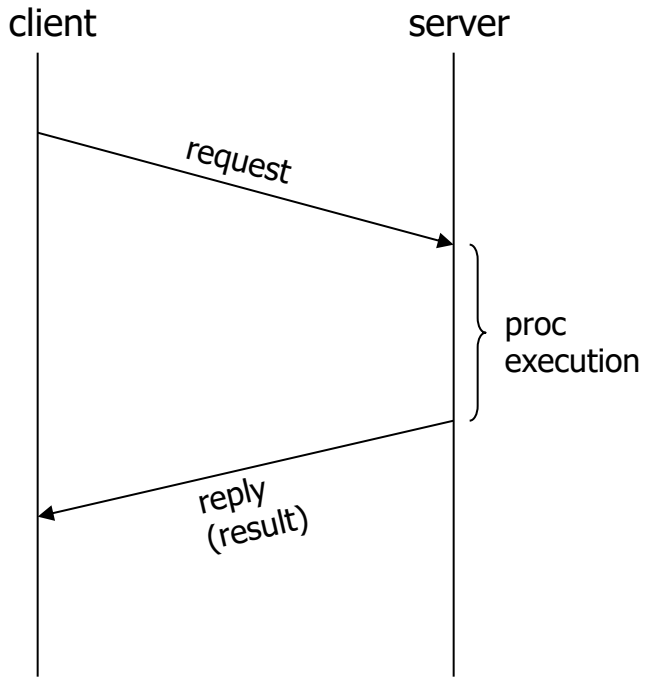


Communicating systems

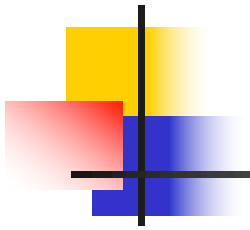
- Then the question was: how to create software systems that could execute programs remotely and receive their results?
- And a related question: how to make implementing this remote calls between computers easier.
- As an answer, people have first introduced
  - **RPC** (Remote Procedure Call)

RPC: Basic idea

- Call a procedure, but execute it at a remote computer `res = proc(args)`
- Get the results *locally*, as if the call was *local*



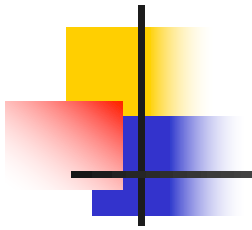
- Request / reply mechanism
- Procedure call, but in a *disjoint address space*, on a different host



RPC: basic idea

- Needed to formalize a separate, but general, *procedure call protocol*
- Idea proposed by J. E. White at SRI in 1976
- Fully described by Birrell and Nelson in the early 1980's
  - Based on the work done at Xerox PARC

Andrew D. Birrell and Bruce Jay Nelson. 1984. *Implementing remote procedure calls*. ACM Trans. Comput. Syst. 2, 1 (February 1984), 39-59.



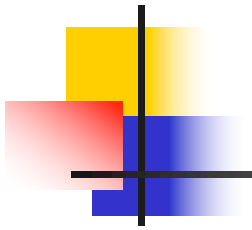
RPC: issues

- Issues to solve:
  - Binding (*connecting* the client and server)
  - Communication protocol
  - Dealing with failures – network/server crashes
  - Addressable arguments
  - Integration with existing systems
  - Data Integrity and security

RPC: principles

- A server defines its *protocol (interface)* using a Protocol Specification Language[1]
- IDL[2] specifies the names, parameters, and types for all client-callable server procedures
- It uses a specific syntax to define the above elements
- A **stub compiler (rpcgen)** reads the protocol spec file and produces two stub procedures for each server procedure (for both the client and server side).

1. http://tools.ietf.org/html/rfc1831.html
2. Now, it is referred to as Interface Definition Language (IDL)



RPC: principles

- The server programmer implements the server procedures and links them with the server-side stubs
- The client programmer implements the client program and links it with the client-side stubs
- The stubs are responsible for managing all details of the remote communication between client and server

RPC: Protocol Example

```
struct param_in {
   long factor;
};
struct result_out {
   long coverage;
};
program SMART_PROG {
   version SMART_VERS {
       result_out SMARTPROC(param_in) = 1; /* procedure # */
   } = 1;                  /* version # */
} = 0x31230000;     /* service id */
```

RPC: Protocol Example

```
struct param_in {
   long factor;
};
struct result_out {
   long coverage;
};
program SMART_PROG {
   version SMART_VERS {
      result_out SMARTPROC(param_in) = 1; /* procedure # */
   } = 1;                  /* version # */
} = 0x31230000;     /* service id */
```

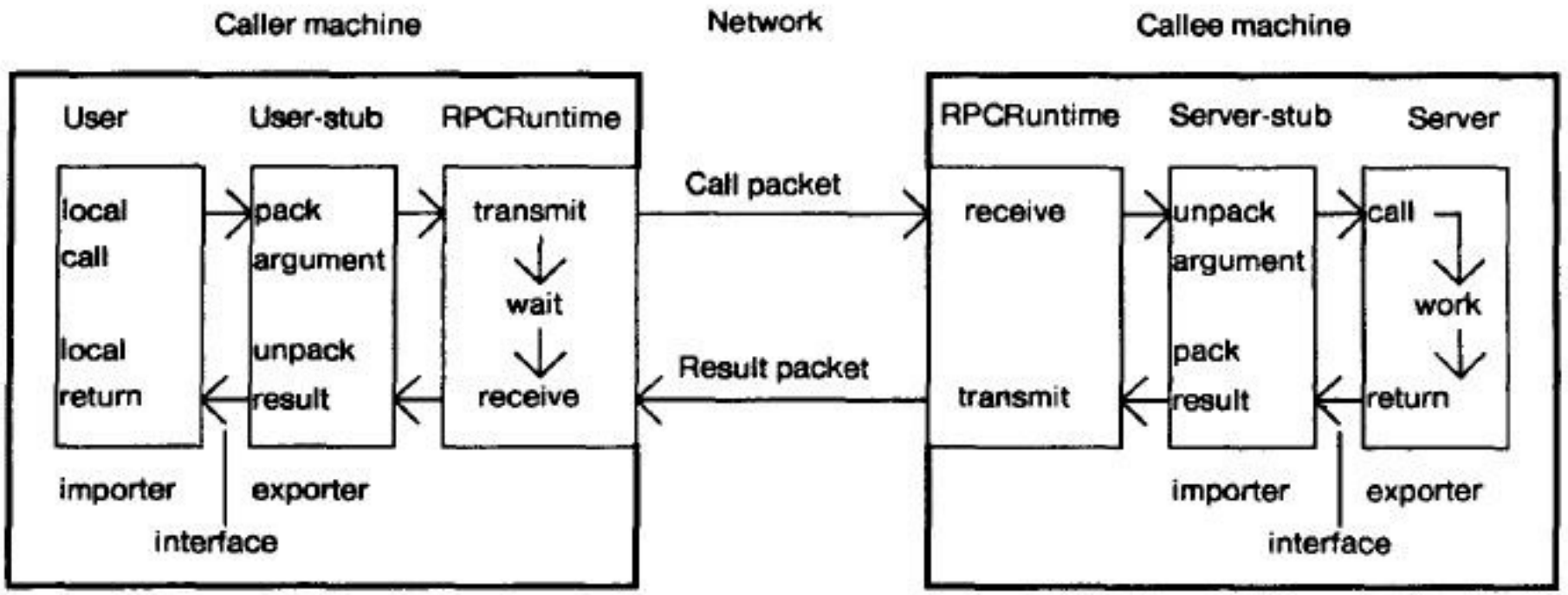
RPC: operation

- A *client-side stub* is a procedure that looks to the client as if it were a callable (server) procedure
- A *server-side stub* looks to the server as if a (regular) client called it
- The client program thinks it is calling a "normal" C function;

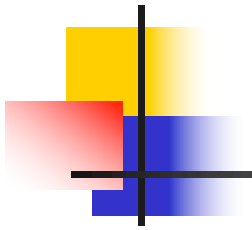
  in fact, it is calling the client stub
- The server program thinks it is called from a C function;

  in fact, it's called by the server stub
- The stubs send messages to each other to actually execute the call
- **RPC happens "transparently"**

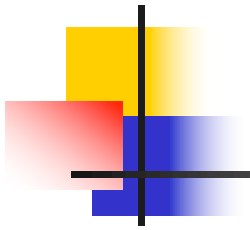
RPC: operation



RPC: operation

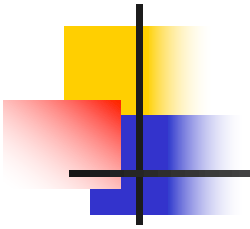


- **Marshalling** is the packing of procedure (function) parameters into a message packet
- Basically, *what to do* (procedure) *with what* (parameters)
- The packet must be representable as a sequence of bytes to be written/read over a network
- The RPC stubs call type-specific procedures to *marshal* (or *unmarshal*) the parameters to a call



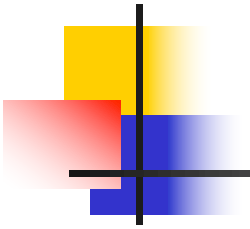
RPC: operation

- **Binding** is the process of connecting the client to the server
- The *server*, when it starts up, exports its interface
- Identifies itself to a network *name server*
- Tells RPC runtime it's alive and ready to accept calls
- Client uses the name server to find the location of a server and establish a connection
- Client is ready to invoke operations on the server



RPC: operation

- The client stub *marshals the operation and parameters* into a *message*
- The server stub *unmarshals operation and parameters* from the message and uses them to call the server operation (procedure) locally
- On return
  - The server stub *marshals the return value*
  - The client stub *unmarshals the return value* and returns it to the client program, as if it was produced locally

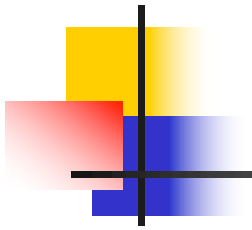


RPC: problems

- Issues with RPC
  - complexity
  - lack of language independent mechanisms (developed primarily for the C programming language)
- Performance overhead
  - often hand-tuned data transfer is more efficient
  - due to data encoding and to the overhead of authentication and fault tolerance

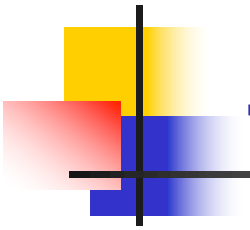
Distributed Objects

- Idea: extend the concepts of RPC to object-oriented style of programming
  - RPC for the "object crowd"
  - Remote method call
- Why Support Distributed Objects?
  - Object oriented paradigm
  - Hide implementation details under object interface
- Client code and object/class code potentially implemented in different OO languages



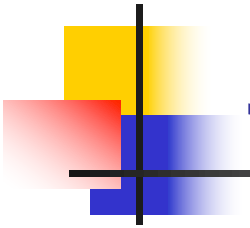
Distributed Objects

- CORBA (Common Object Request Broker Architect.)
  - Cross-lingual (primarily C++, Java)
  - Cross-platform
  - Many features
- DCOM (Distributed Component Object Model)
  - Microsoft's solution
  - Some cross-lingual support (within "Microsoft world")
  - Windows only
  - Built on DCE RPC and COM
- Java RMI
  - Single language, tightly integrated with Java



Java RMI

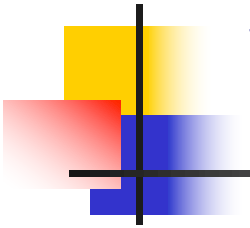
- Specification by Sun (now owned by Oracle) for making remote method invocations on Java objects from Java clients
- Specific to Java
- Interfaces and Classes for specification are
- Part of the core Java platform since 1.1 (1997)
- Java Standard Edition (J2SE) comes with the implementation of RMI



Java RMI: Marshalling

Serialization

- Conversion of a Java Object into a byte stream
- Any Java class instance can be *serialized*
- Involves writing and reading state of an object in the same order and using the same encoding
- Serialization can be
  - Provided automatically by the Java environment
  - Written explicitly by the developer writing the class
- Serialization is the “wire” protocol for Java RMI

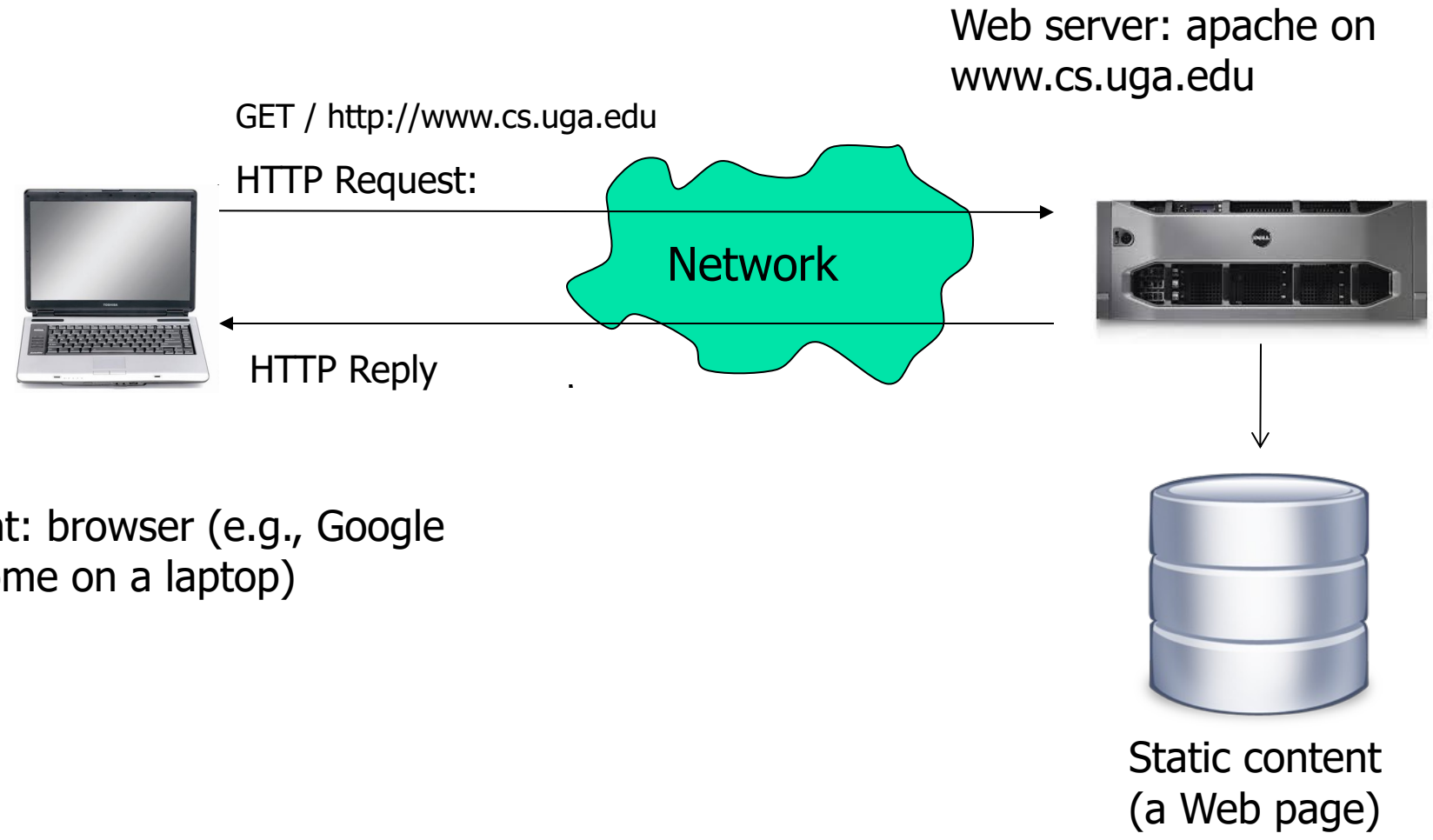


World Wide Web

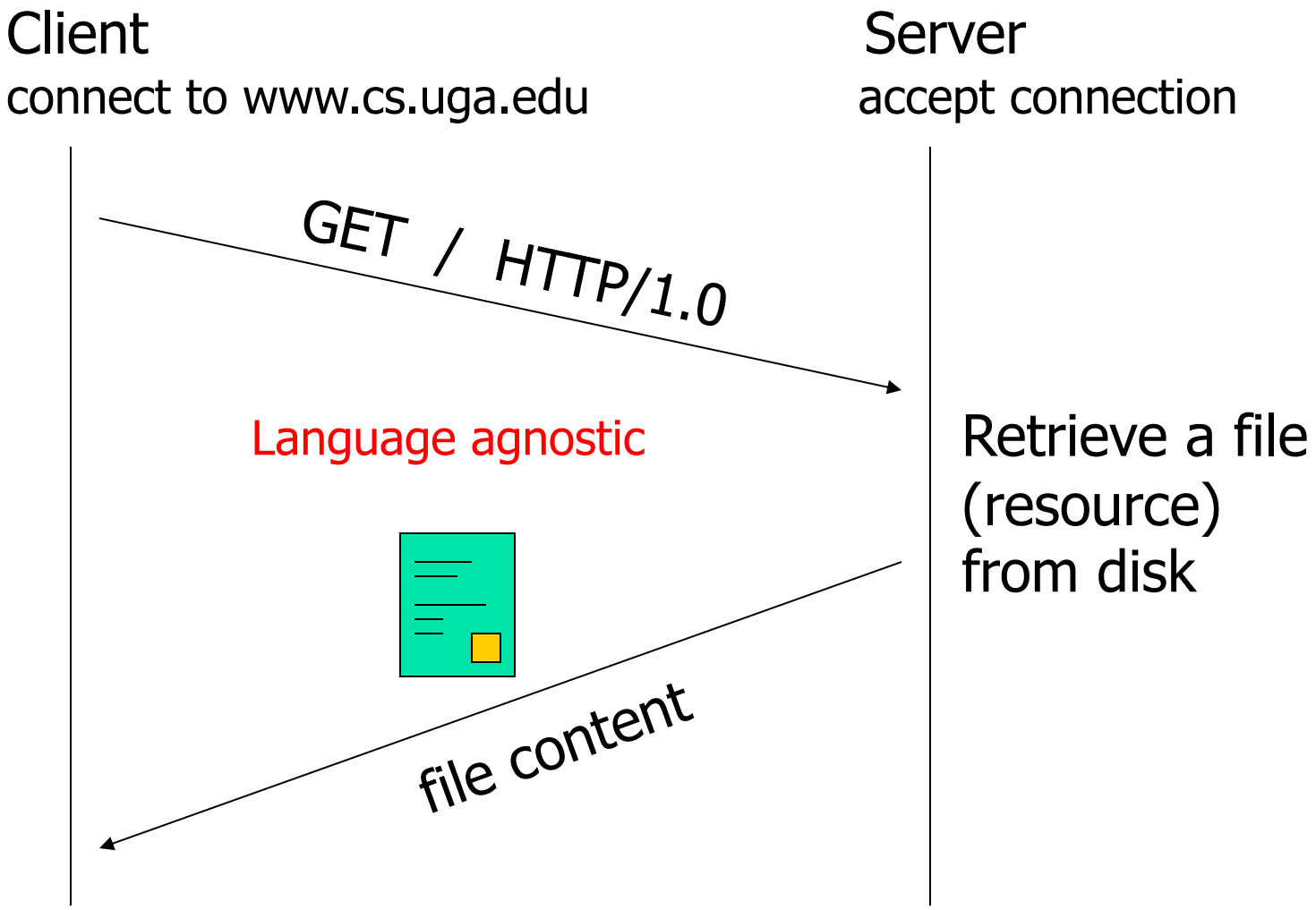
World Wide Web

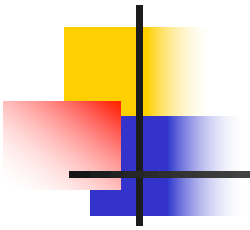
- Revolutionized remote access to data
- Data, referred to as ***resources***, can be remotely requested, downloaded/uploaded
- Resources are delivered by Web servers, or similar systems
- HTTP (Hypertext Transfer Protocol) is used to communicate between the client (requestor) and the server

WWW communication



Web communication





HTTP Request

- A request (HTTP 1.0) has the following form:

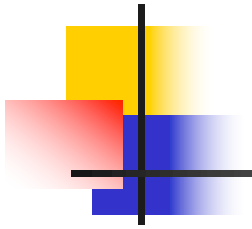
  Request-Method Request-URL HTTP-Version <CR><LF>
  (generic-header | request-header | entity-header) <CR><LF>
  <CR><LF>
  [message body]

- For example, on cobweb.cs.uga.edu (using HTTP 1.0):

  GET / HTTP/1.0

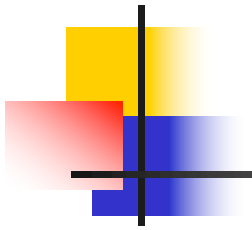
- A simple request to www.cs.uga.edu (using HTTP 1.1):

  GET / HTTP/1.1
  Host: www.cs.uga.edu



HTTP Request

- HTTP request methods:
  - **GET** – request whatever information is identified by the Request-URL
  - **POST** – request that server accepts the entity enclosed in the request (create a child resource)
  - **PUT** – request that the enclosed entity be stored under the Request-URL (replace an existing or create new)
  - **DELETE** – request that the server delete the resource identified by Request-URL
  - **HEAD** – identical to GET, but server must not return a message body in response
  - A few additional methods are available



Dynamic Content

- Responses (usually, Web pages) can be created dynamically, in response to requests
- Advantages
  - personalization (e.g., athena.uga.edu),
  - interaction with client input
  - interaction with back-end applications
- Disadvantages
  - performance penalty -- dynamic content must be created at request time, which takes time
- Generating dynamic content (CGI Scripts, Servlets)

Web communication

Web server: Apache on uml.cs.uga.edu

Get / http://uml.cs.uga.edu

HTTP Request:

Network

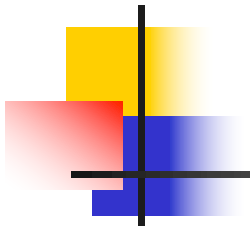
HTTP Reply

CGI

Client: browser (e.g., Google Chrome on a laptop)

Dynamic content (a CGI script)

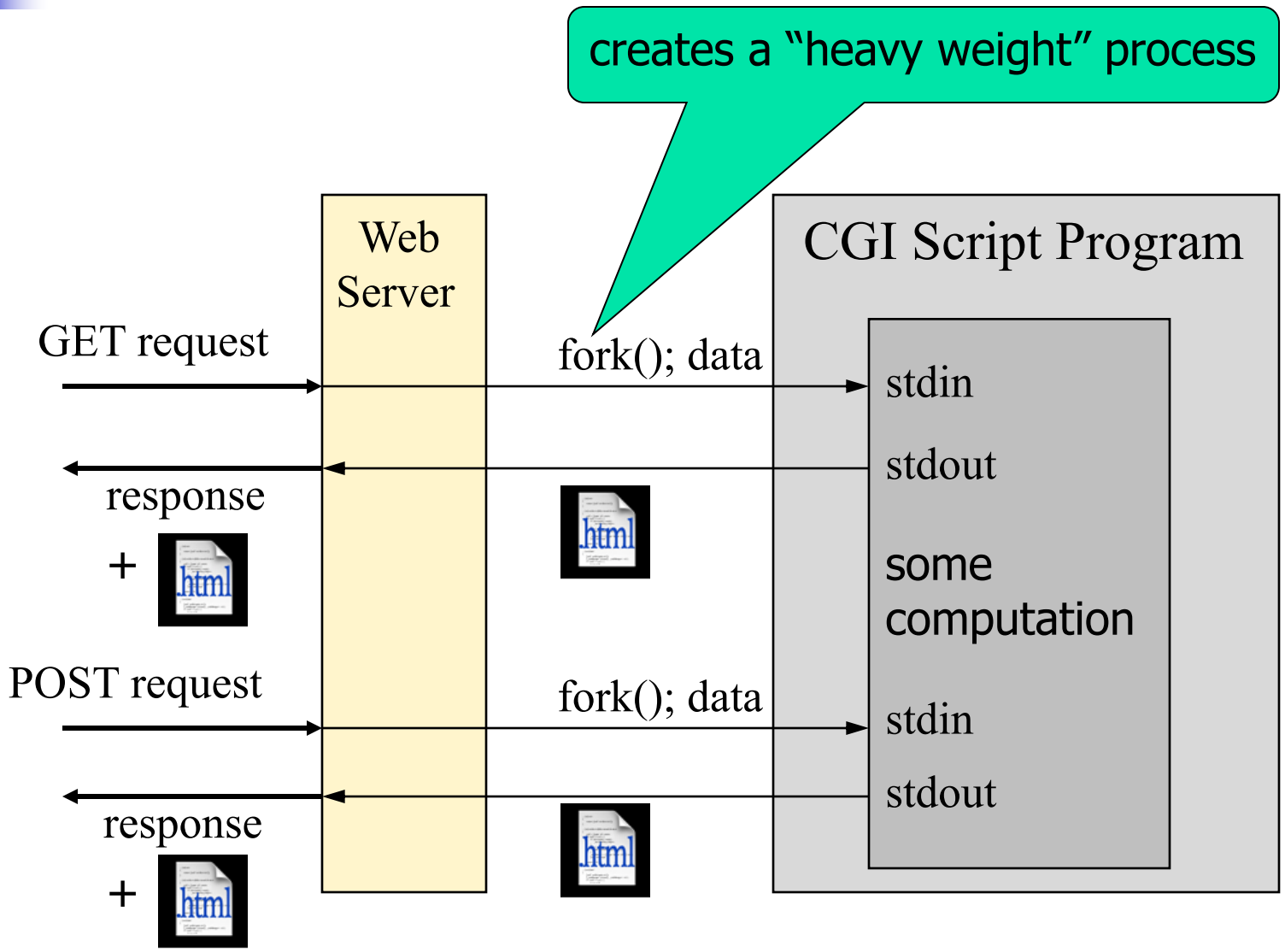
CGI is Common Gateway Interface

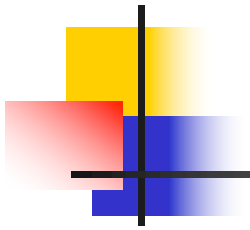


CGI Scripts

- CGI scripts were introduced to provide dynamic content by executing programs remotely
- The script is a program (e.g., C, PHP, Perl)
- A client requests a dynamic resource (URL)
- When the URL is requested, the server invokes the named script, passing to it client data (input parameters)
- The script executes and ultimately outputs an HTML page which is sent back to the client

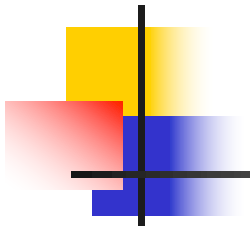
CGI Execution





Servlets

- When we run small Java programs within a browser these are referred to as *applets*
- Small Java programs running within a server are called *servlets*
- A *servlet* is a server-side program designed to process a client request (which requires interactivity)
- Servlets have become a popular mechanism for creating *interactive* servers

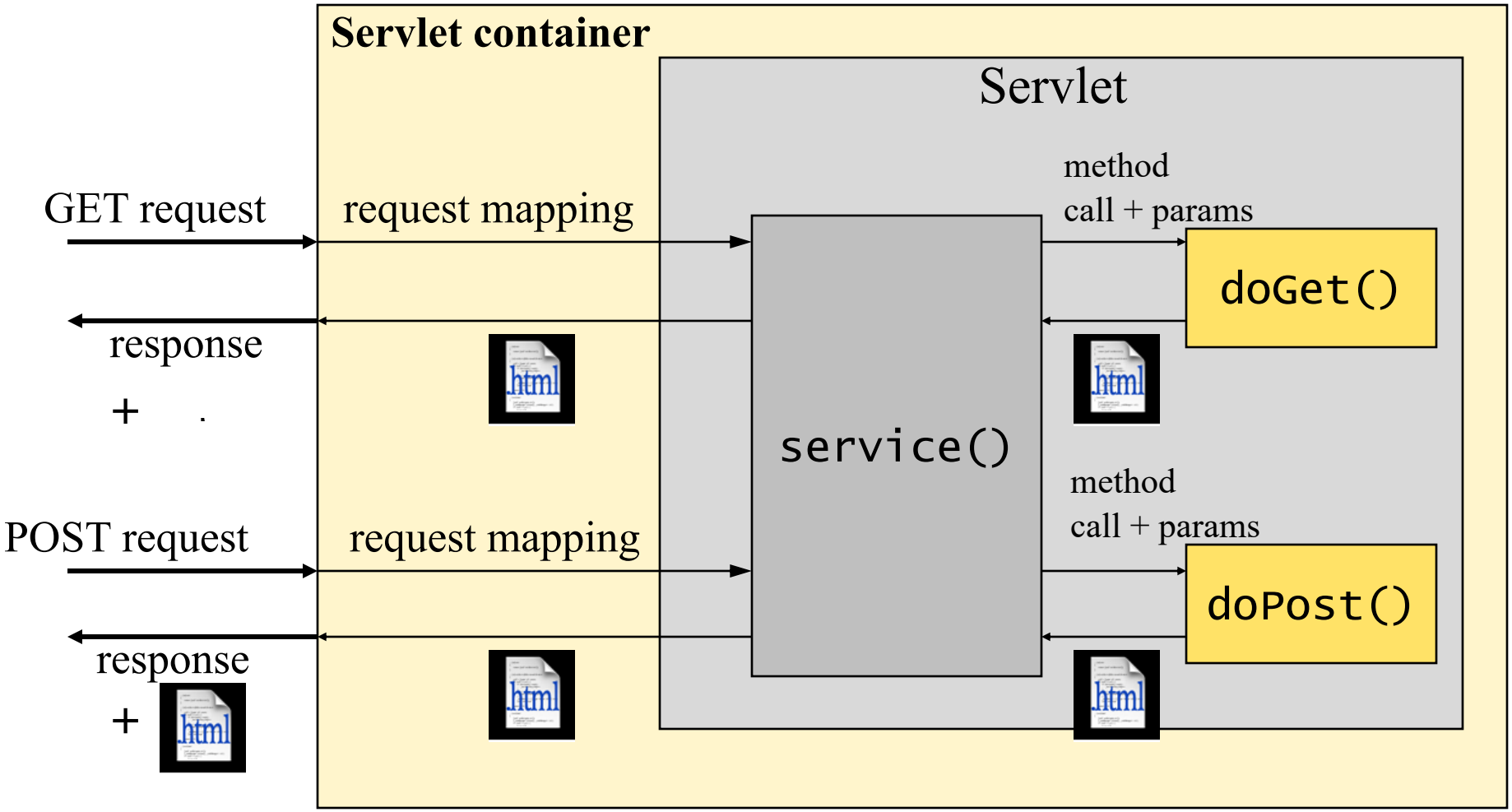


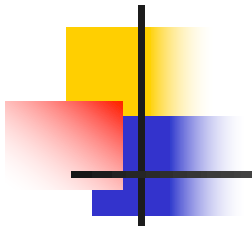
Servlets

- A servlet (an instance of a special Java class) is compiled and deployed to run within a special server
- Such a server is frequently called a servlet container
- A servlet container with greatly enhanced capabilities (messaging, administration, etc.) is referred to as an application server

Servlet invocation

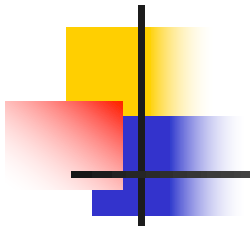
Once started, a servlet does not exit





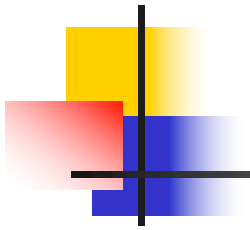
Web Services

- Consider a servlet container with many servlets, each for some type of a remote computation to produce desired data
- Collectively, these servlets can be thought of as a service providing some programs (services) to many clients
- Web services have been introduced as a generalization of server-side WWW processing



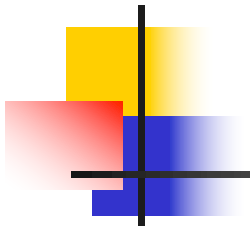
Web Services

- Web Services – the name
  - Hewlett-Packard's e-Speak in 1999, an enabler for e-services
  - Microsoft introduced the name “Web services" in June 2000
  - Now, Web services have become virtually ubiquitous



Web Services

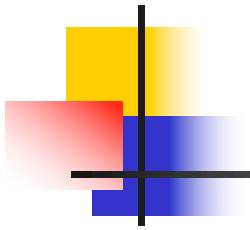
- Two basic types of Web Services:
  - SOAP Web Services
  - REST Web Services
- The same (old) problem: how to invoke a remote computation over the network and get the result?
- We will very briefly talk about SOAP and then focus on REST services



Web Services

SOAP (Simple Object Access Protocol) is Based on open, standard technologies

- XML – encoding data such that it can be exchanged between applications and platforms
- SOAP is a messaging protocol for transporting information and instructions between applications (uses XML)



Web Services

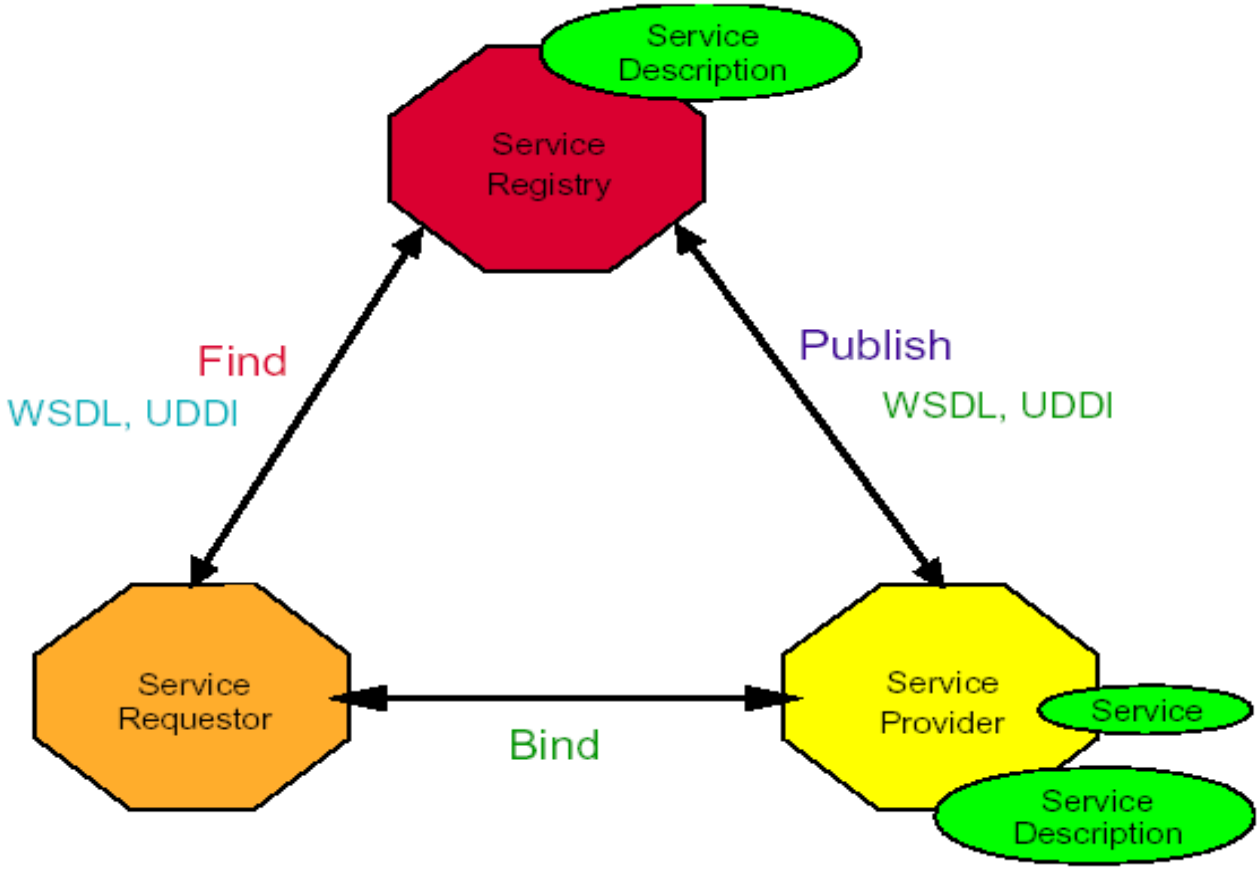
- WSDL (Web Services Description Language)

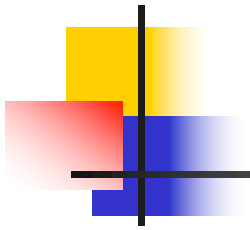
  a standard method of describing Web Services and their capabilities (in XML)

- UDDI (Universal Description, Discovery and Integration)

  defines XML-based rules for building directories in which service providers advertise their Web Services.

Web Services: Model

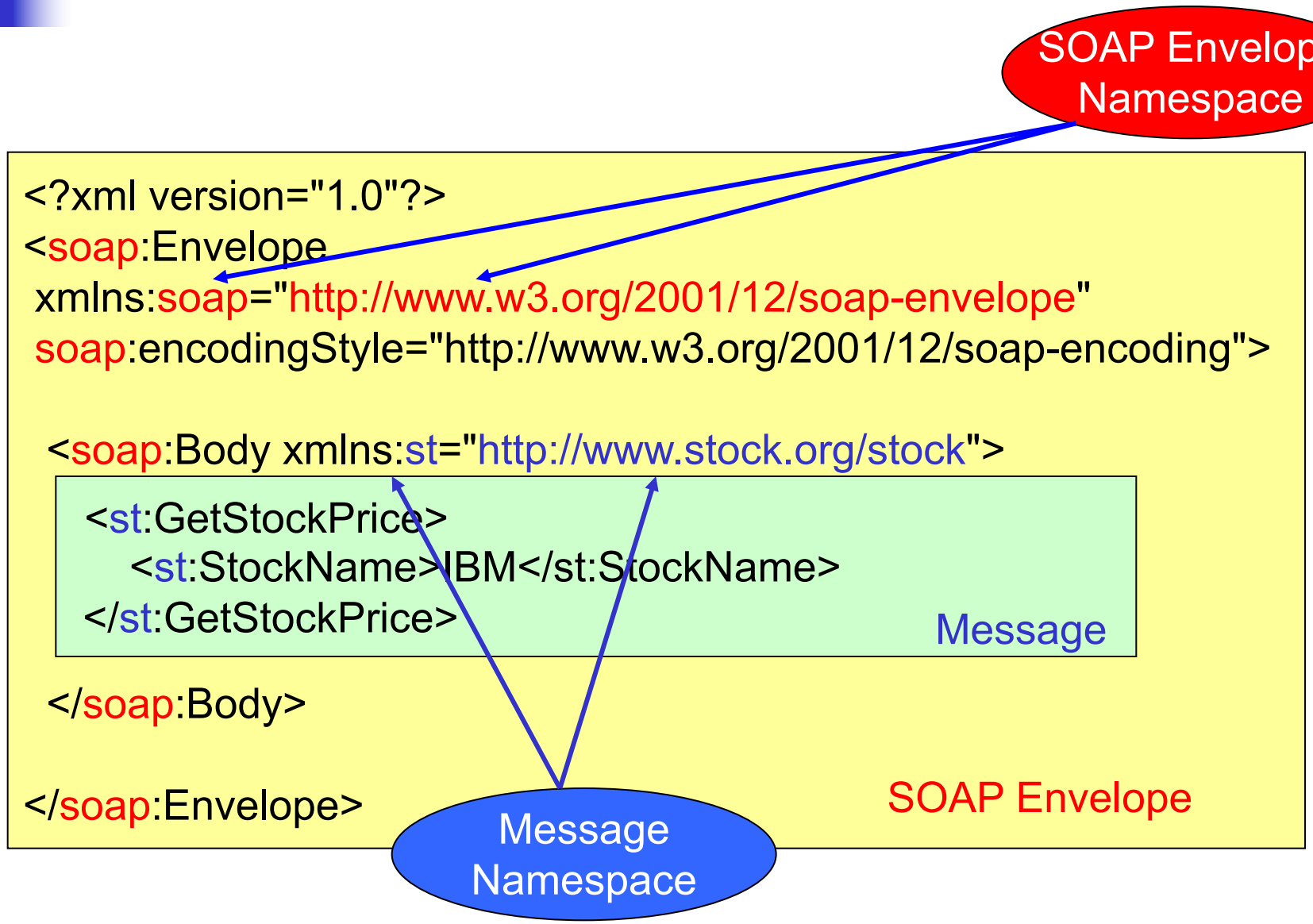




Web Services: SOAP

- Platform and language independent
- SOAP message has three parts
  - *envelope* – wraps entire message and contains header and body
  - *header* – optional element with additional info such as security or routing
  - *body* – application-specific data being communicated

SOAP Request Message



SOAP Response Message

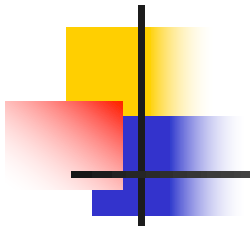
```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:st="http://www.stock.org/stock">

    <st:GetStockPriceResponse>
      <st:Price>34.5</st:Price>
    </st:GetStockPriceResponse>

  </soap:Body>
</soap:Envelope>
```

Message

SOAP Envelope

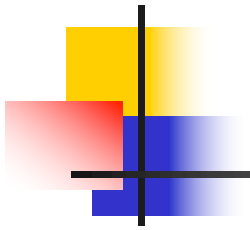
Result returned in Body

SOAP Services: Verb-centered



- Consider the remote calls considered so far:
  - RPC (Remote Procedure Call)
  - RMI (Remote Method Invocation)
  - Even a plain Java or C++ method call
- They all share a common aspect: their names correspond to actions, usually denoted as verbs or verb phrases:

  setText, compute, solve, draw, print



REST Services: Noun-centered

- We will transition to Web services where the focus is on resources (nouns), instead of actions (verbs)
- The actions are *fixed*
- Actions correspond to the main HTTP methods: GET, POST, PUT, and DELETE