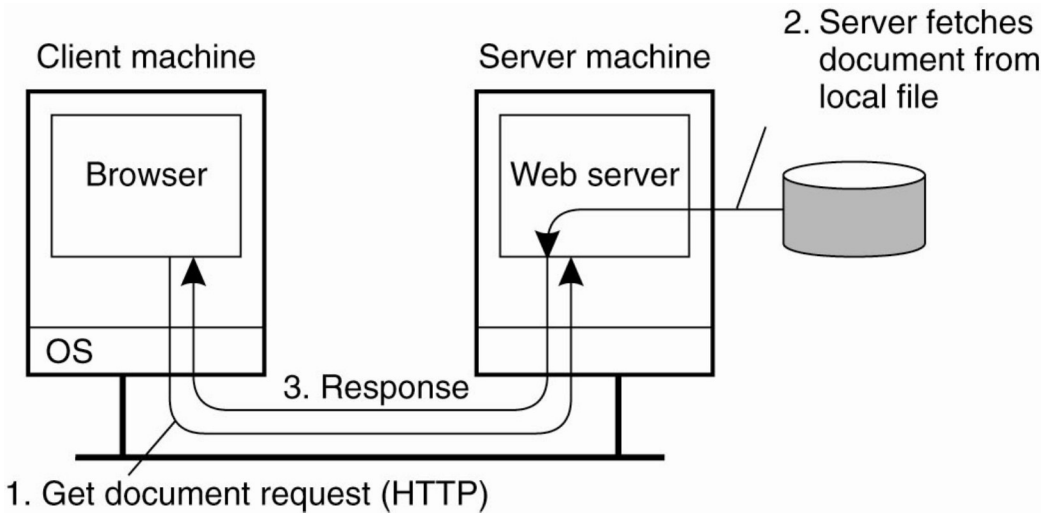
Today: World Wide Web

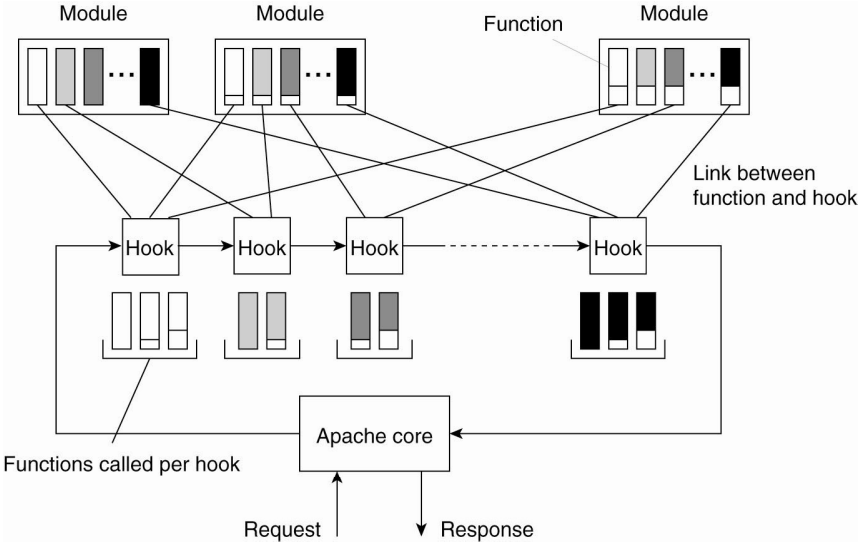
- WWW principles
- Case Study: web caching as an illustrative example
  - Invalidate versus updates
  - Push versus Pull
  - Cooperation between replicas

Traditional Web-Based Systems



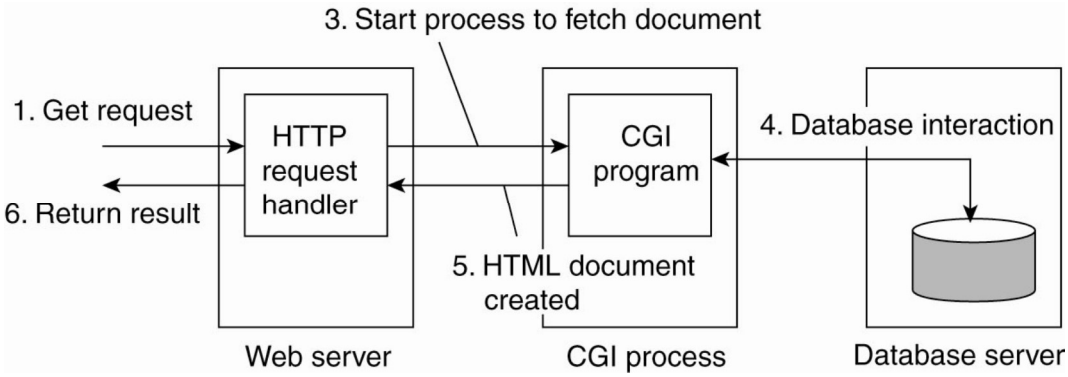
- The overall organization of a traditional Web site.

The Apache Web Server



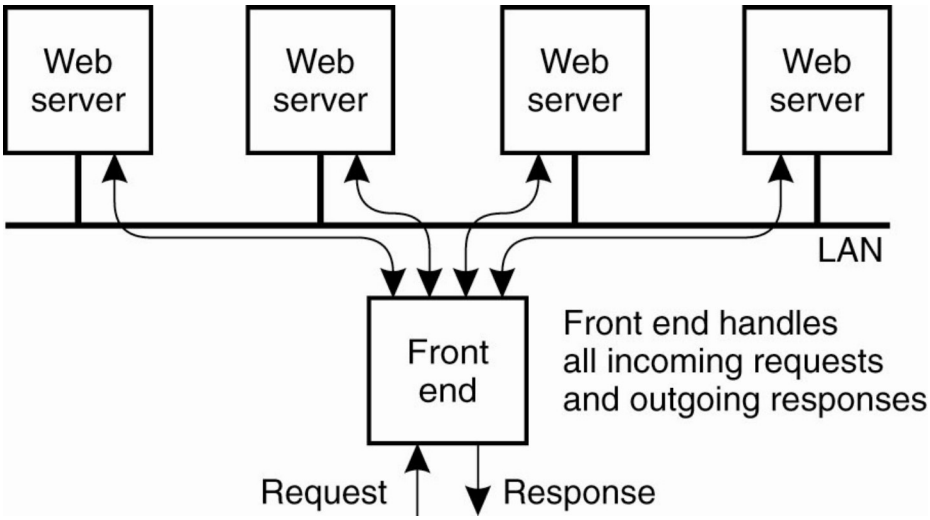
- The general organization of the Apache Web server.

Multitiered Architectures



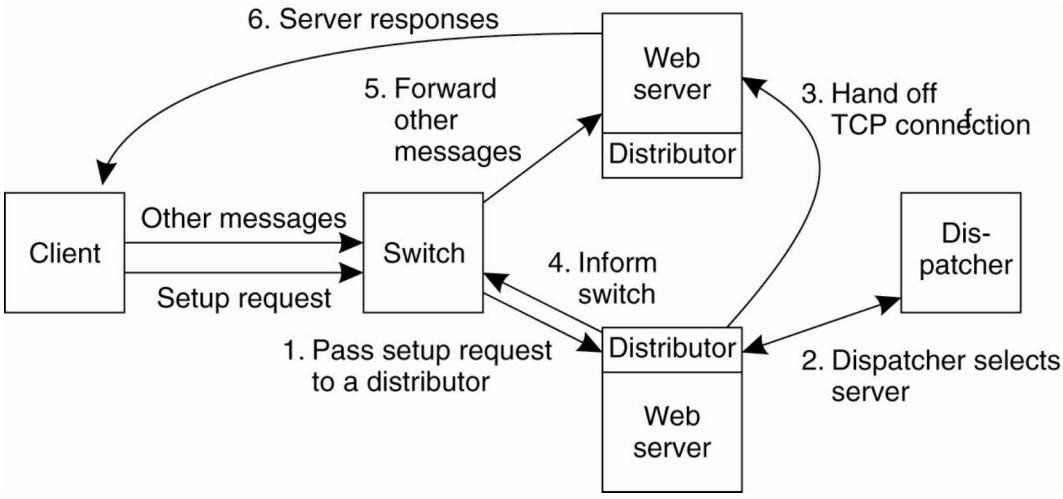
- The principle of using server-side CGI programs.

Web Server Clusters



- The principle of using a server cluster in combination with a front end to implement a Web service.

Web Server Clusters (2)



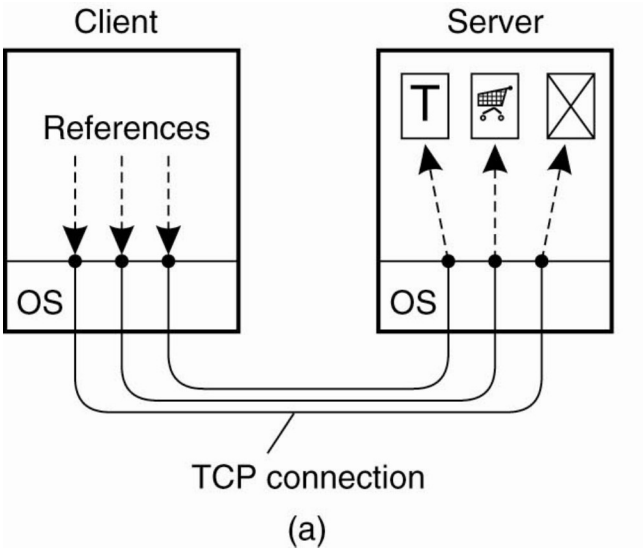
- A scalable content-aware cluster of Web servers.

HTML and Web Documents

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

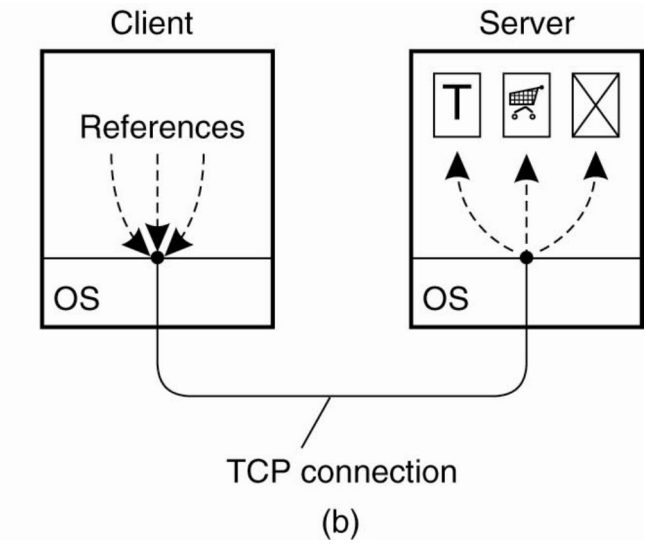
- Six top-level MIME types and some common subtypes.

HTTP Connections



- Using nonpersistent connections.

HTTP Connections



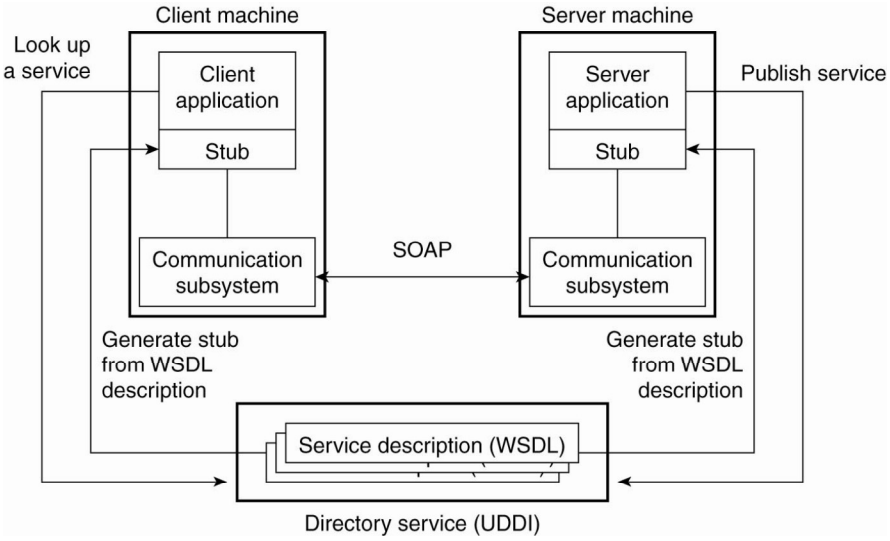
- (b) Using persistent connections.

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

- Operations supported by HTTP.

Web Services Fundamentals



- The principle of a Web service.

Simple Object Access Protocol

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
    <env:Header>
      <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
        <n:priority>1</n:priority>
        <n:expires>2001-06-22T14:00:00-05:00</n:expires>
      </n:alertcontrol>
    </env:Header>
    <env:Body>
      <m:alert xmlns:m="http://example.org/alert">
        <m:msg>Pick up Mary at school at 2pm</m:msg>
      </m:alert>
    </env:Body>
</env:Envelope>
```

- An example of an XML-based SOAP message.

RESTful Web Services

- SOAP heavyweight protocol for web-based distributed computing
  - RESTful web service: lightweight , point-to-point XML comm
- REST=representative state transfer
  - HTTP GET => read
  - HTTP POST => create, update, delete
  - HTTP PUT => create, update
  - HTTP DELETE => delete
- Simpler than RPC-sytle SOAP
  - closer to the web

RESTful Example

```
GET /StockPrice/IBM HTTP/1.1
Host: example.org
Accept: text/xml
Accept-Charset: utf-8
```

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<s:Quote xmlns:s="http://example.org/stock-service">
     <s:TickerSymbol>IBM</s:TickerSymbol>
     <s:StockPrice>45.25</s:StockPrice>
</s:Quote>
```

Corresponding SOAP Call

```
GET /StockPrice HTTP/1.1
Host: example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
   xmlns:s="http://www.example.org/stock-service">
   <env:Body>
     <s:GetStockQuote>
          <s:TickerSymbol>IBM</s:TickerSymbol>
     </s:GetStockQuote>
   </env:Body>
</env:Envelope>
```

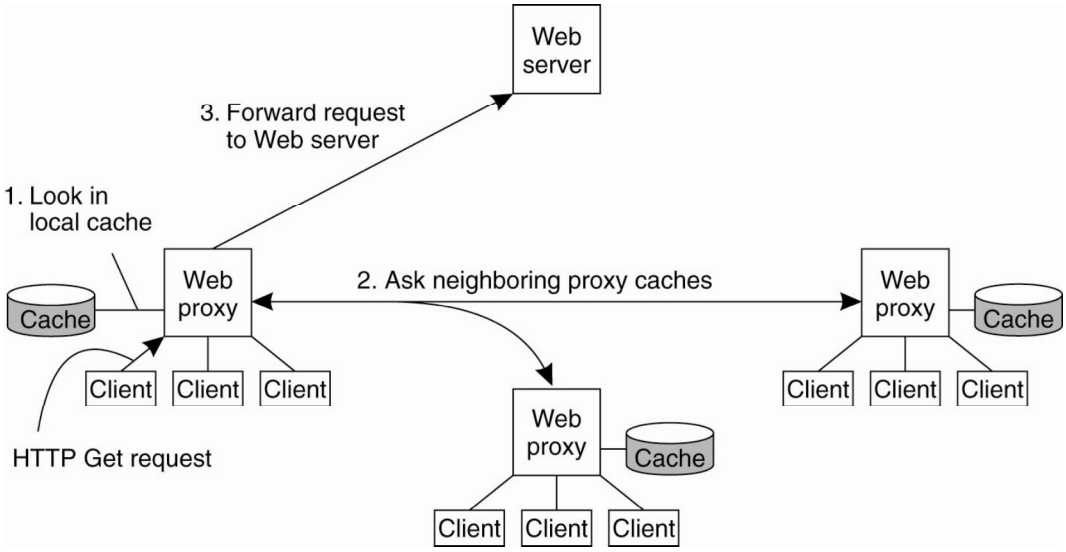
```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
   xmlns:s="http://www.example.org/stock-service">
   <env:Body>
     <s:GetStockQuoteResponse>
          <s:StockPrice>45.25</s:StockPrice>
     </s:GetStockQuoteResponse>
   </env:Body>
</env:Envelope>
```

SOAP vs RESTful WS

- Language, platform and transport agnostic
- Supports general distributed computing
- Standards based (WSDL, UDDI dir. service...)
- Builtin error handling
- Extensible
- More heavy-weight
- Harder to develop

- Language and platform agnostic
- Point-to-point only; no intermediaries
- Lack of standards support for security, reliability (“roll you own”
- Simpler, less learning curve, less reliance on tools
- Tied to HTTP transport layer
- More consise

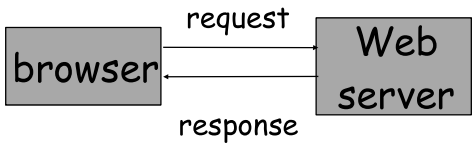
Web Proxy Caching

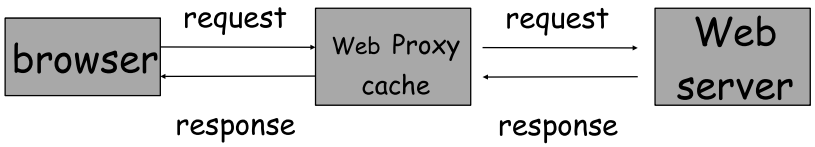


- The principle of cooperative caching.

Web Caching

- Example of the web to illustrate caching and replication issues
  - Simpler model: clients are read-only, only server updates data



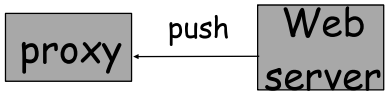


Consistency Issues

- Web pages tend to be updated over time
  - Some objects are static, others are dynamic
  - Different update frequencies (few minutes to few weeks)
- How can a proxy cache maintain consistency of cached data?
  - Send invalidate or update
  - Push versus pull

Push-based Approach

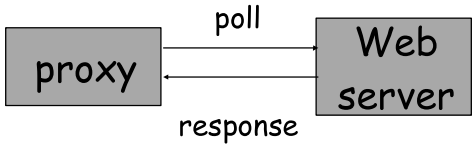
- Server tracks all proxies that have requested objects
- If a web page is modified, notify each proxy
- Notification types
  - Indicate object has changed [invalidate]
  - Send new version of object [update]
- How to decide between invalidate and updates?
  - Pros and cons?
  - One approach: send updates for more frequent objects, invalidate for rest



Push-based Approaches

- Advantages
  - Provide tight consistency [minimal stale data]
  - Proxies can be passive
- Disadvantages
  - Need to maintain state at the server
    - Recall that HTTP is stateless
    - Need mechanisms beyond HTTP
  - State may need to be maintained indefinitely
    - Not resilient to server crashes

Pull-based Approaches



- Proxy is entirely responsible for maintaining consistency
- Proxy periodically polls the server to see if object has changed
  - Use if-modified-since HTTP messages
- Key question: when should a proxy poll?
  - Server-assigned *Time-to-Live (TTL)* values
    - No guarantee if the object will change in the interim

Pull-based Approach: Intelligent Polling

- Proxy can dynamically determine the refresh interval
  - Compute based on past observations
    - Start with a conservative refresh interval
    - Increase interval if object has not changed between two successive polls
    - Decrease interval if object is updated between two polls
    - Adaptive: No prior knowledge of object characteristics needed

Pull-based Approach

- Advantages
  - Implementation using HTTP (If-modified-Since)
  - Server remains stateless
  - Resilient to both server and proxy failures
- Disadvantages
  - Weaker consistency guarantees (objects can change between two polls and proxy will contain stale data until next poll)
    - Strong consistency only if poll before every HTTP response
  - More sophisticated proxies required
  - High message overhead