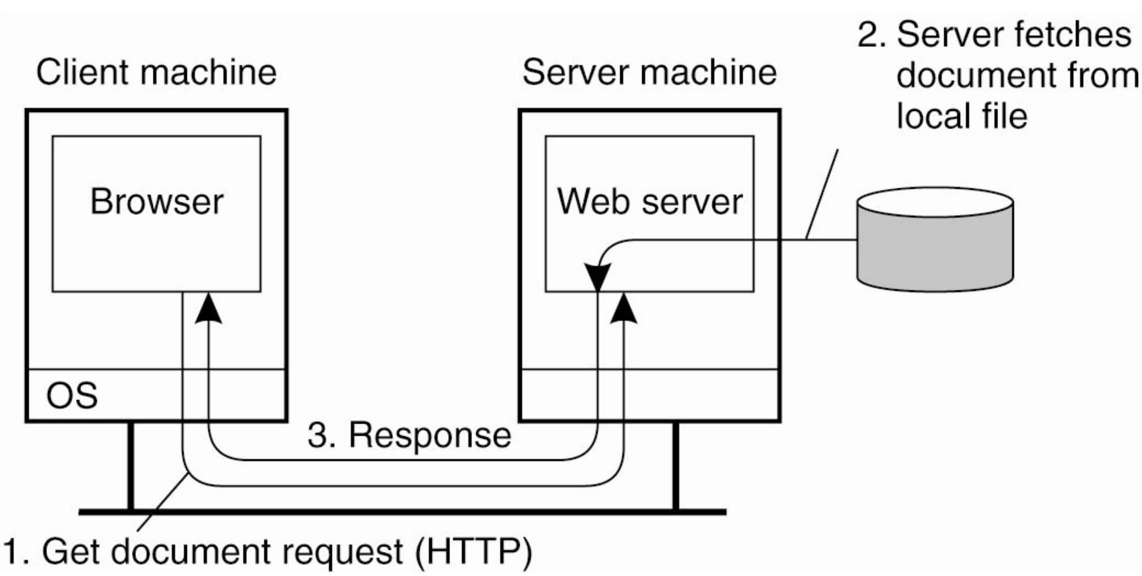
# Distributed Web Applications

- WWW principles

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

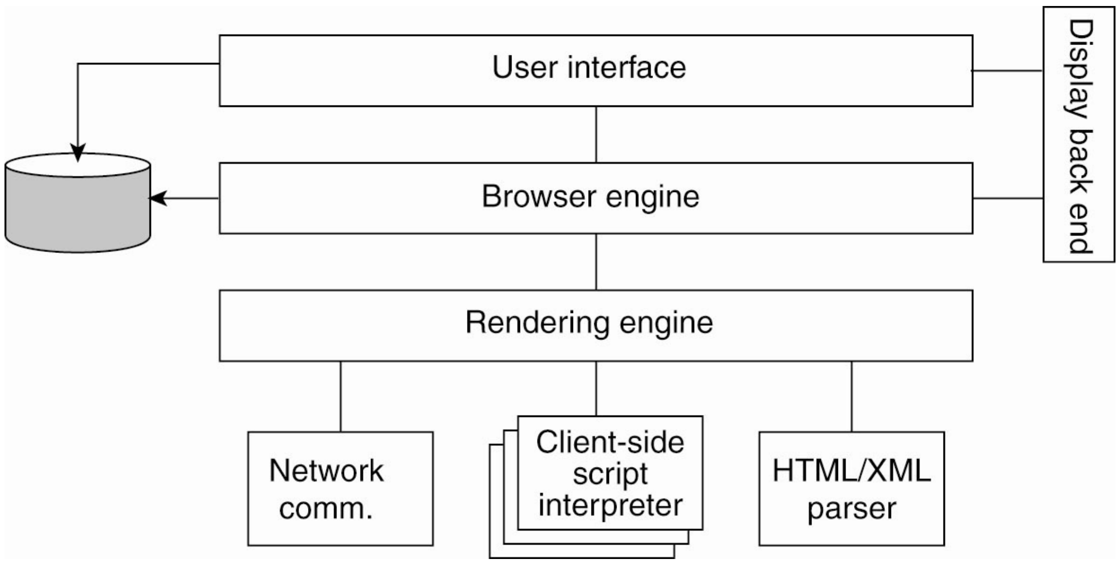


# Traditional Web-Based Systems




- Client-server web applications

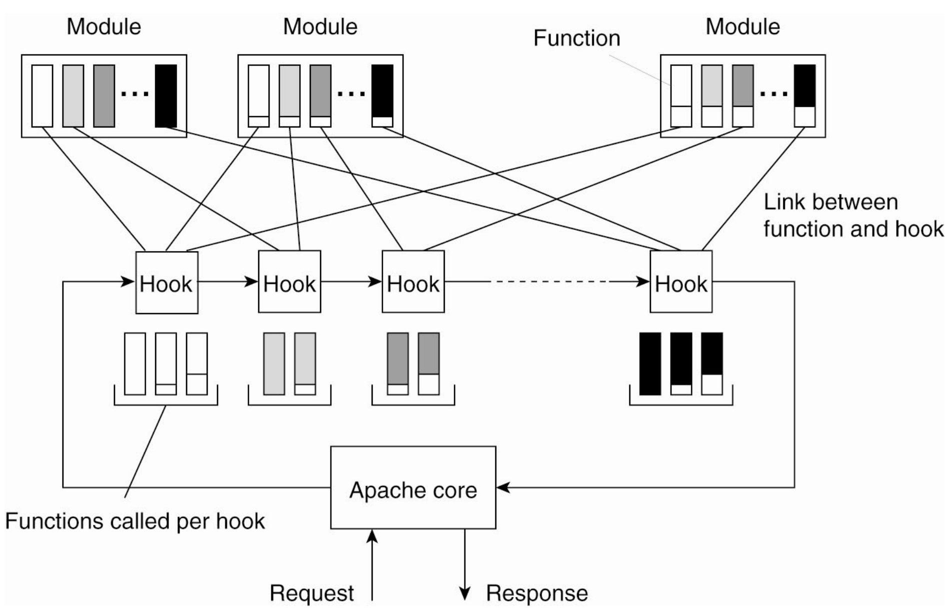
# Web Browser Clients




- The logical components of a Web browser.



# The Apache Web Server



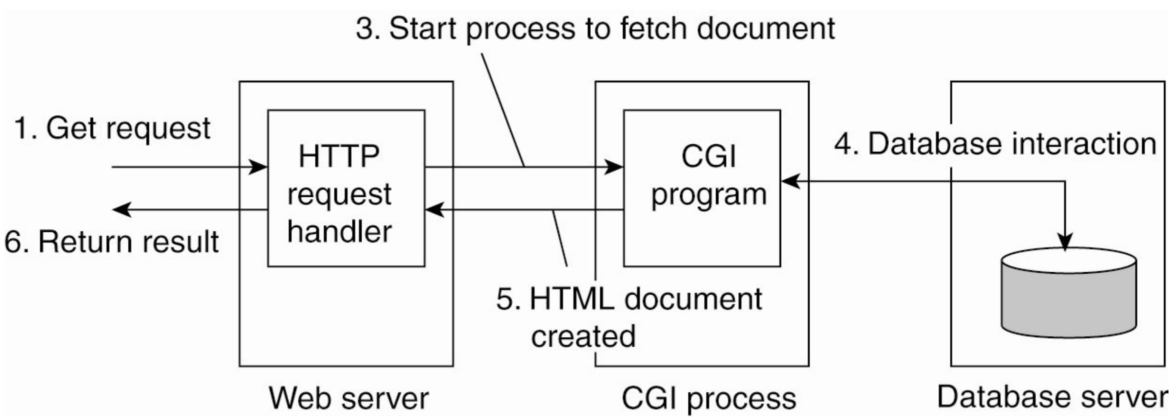

- The general organization of the Apache Web server.

# Proxy Servers


- Using a Web proxy when the browser does not speak FTP (or for caching and offloading)

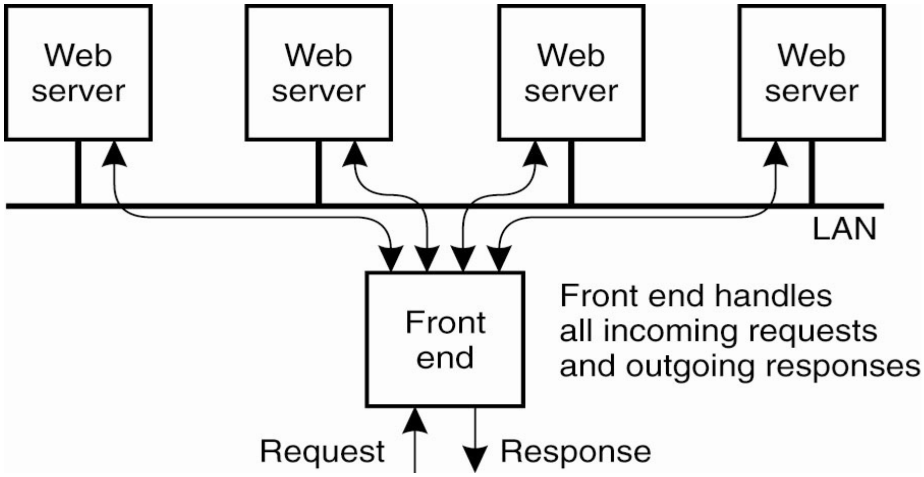


# Multitiered Architectures




- Three tiers: HTTP, application, and database tier

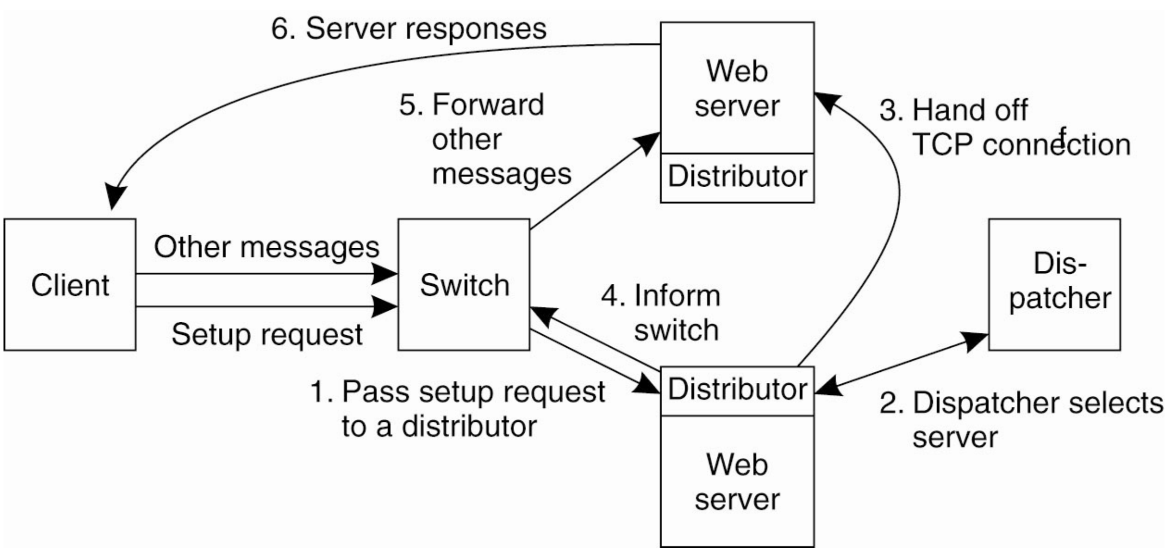
# Web Server Clusters




- Clients connect to front-end dispatcher, which forwards requests to a replica (recall discussion from Cluster scheduling)
- Each replica can be a tiered system
  - For consistency, database can be a common/non-replicated



# Web Server Clusters (2)




- A scalable content-aware cluster of Web servers.

# Web Clusters

- Request-based scheduling
  - Forward each request to a replica based on a policy
- Session-based scheduling
  - Forward each session to a replica based on a policy
- Scheduling policy: round-robin, least loaded
- HTTP redirect vs TCP splicing vs TCP handoff



# Elastic Scaling

- Web workloads: temporal time of day, seasonal variations
  - Flash crowds: black friday, sports events, news events
- Overloads can occur even with clustering and replication
- Elastic scaling: dynamically vary application capacity based on workload (aka auto-scaling, dynamic provisioning)
- Two approaches:
  - Horizontal scaling: increase or decrease # of replicas based on load
  - Vertical scaling: increase or decrease size of replica (e.g., # of cores allocated to container or VM) based on load
- Proactive versus reactive scaling
  - Proactive: predict future load and scale in advance
  - Reactive: scale based on observed workload
- Common in large cloud-based web applications

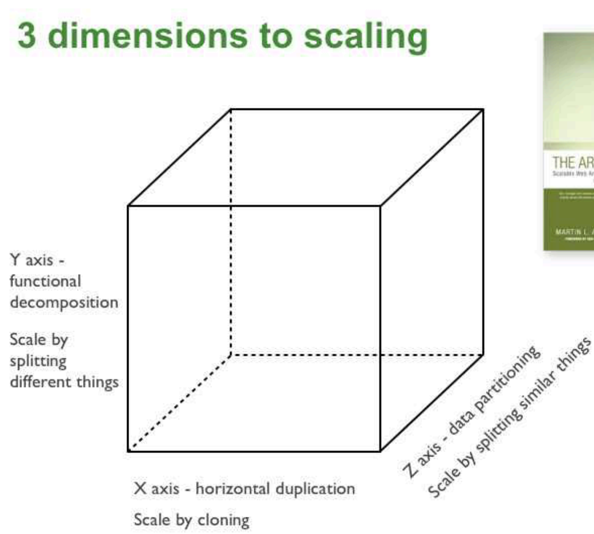
# Micro-services Architecture

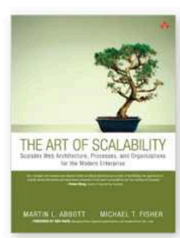
- Micro-services: application is a collection of smaller services
  - Example of service-oriented architecture
  - Modular approach to overcome “monolith hell”
- Each microservice is small and can be maintained independently of others
- Each is independently deployable
- Clustering and auto-scaling can be performed independently



# Scaling Web applications

- Three approaches for scaling




https://microservices.io/articles/scalecube.html

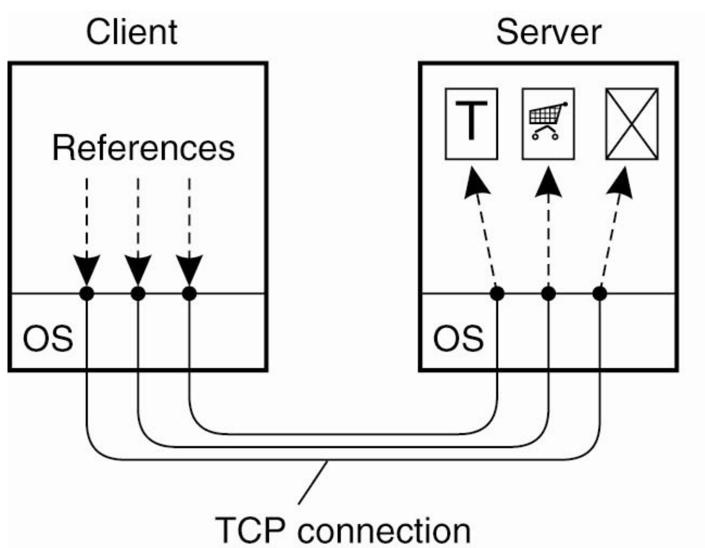
# Web Documents

| 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.



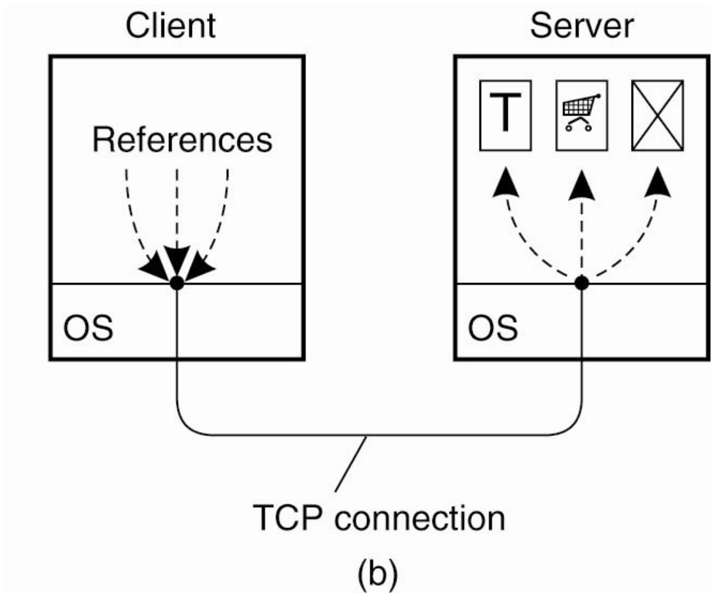
# HTTP Connections




(a)

- Using nonpersistent connections.

# HTTP 1.1 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.

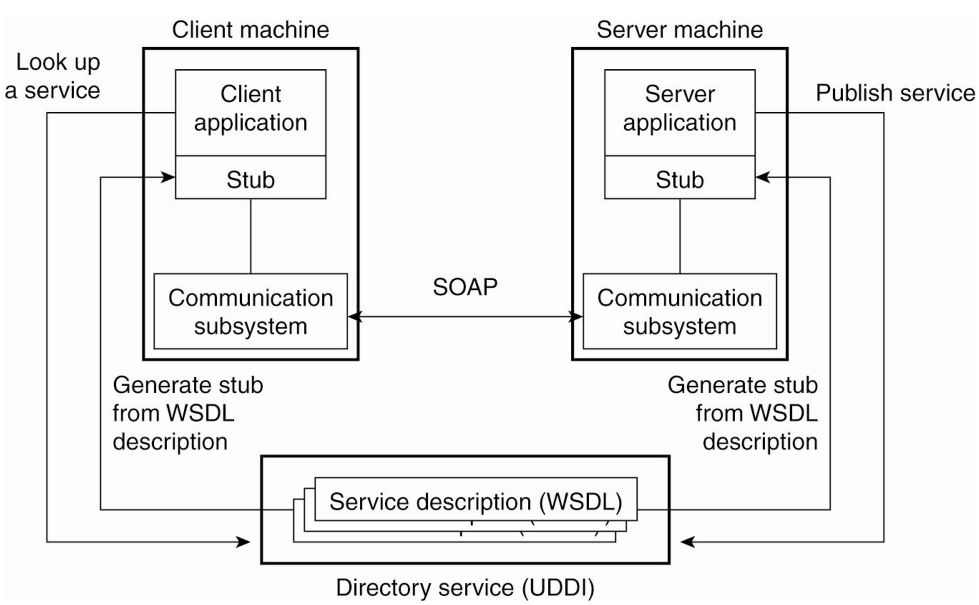
# HTTP 2.0

- Http 1.1 allows pipelining over same connection
  - Most browsers do not use this feature
- HTTP v2: Designed to reduce message latency
  - No new message or response types
- Key features
  - Binary headers (over text headers of http 1.1)
  - Uses compression of headers and messages
  - Multiplex concurrent connection over same TCP connection
    - each connection has multiple "streams", each carrying a request and response
      - No blocking caused by pipelining in http 1.1

See https://developers.google.com/web/fundamentals/performance/http2/



# 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 heavy-weight 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 concise