8. Distributed Web-based Systems

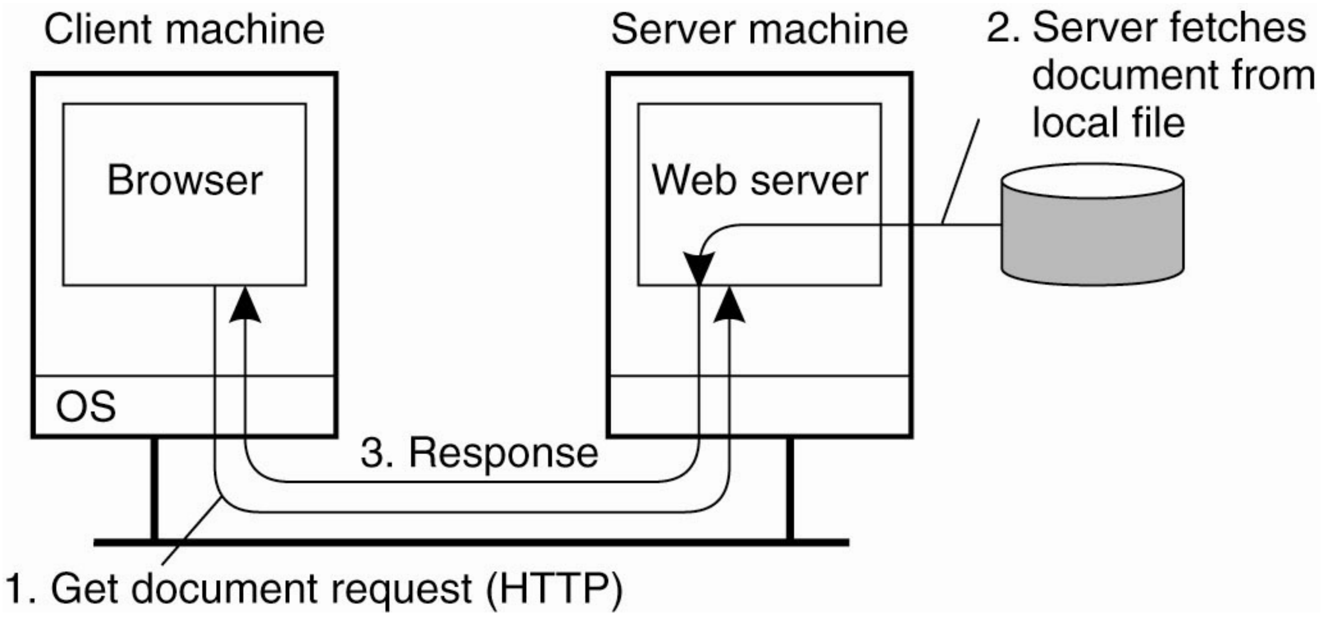
Sistemes Distribuïts en Xarxa (SDX)
Facultat d’Informàtica de Barcelona (FIB)
Universitat Politècnica de Catalunya (UPC)
2024/2025 Q2

Contents

- **Architecture**
- Communication
- Naming
- Synchronization
- Consistency & replication
- Fault tolerance

Traditional Web-based systems

- The WWW is a huge client-server distributed system with millions of clients and servers
  - Server: host hyperlinked docs, run commands
  - Browser: retrieve and display docs

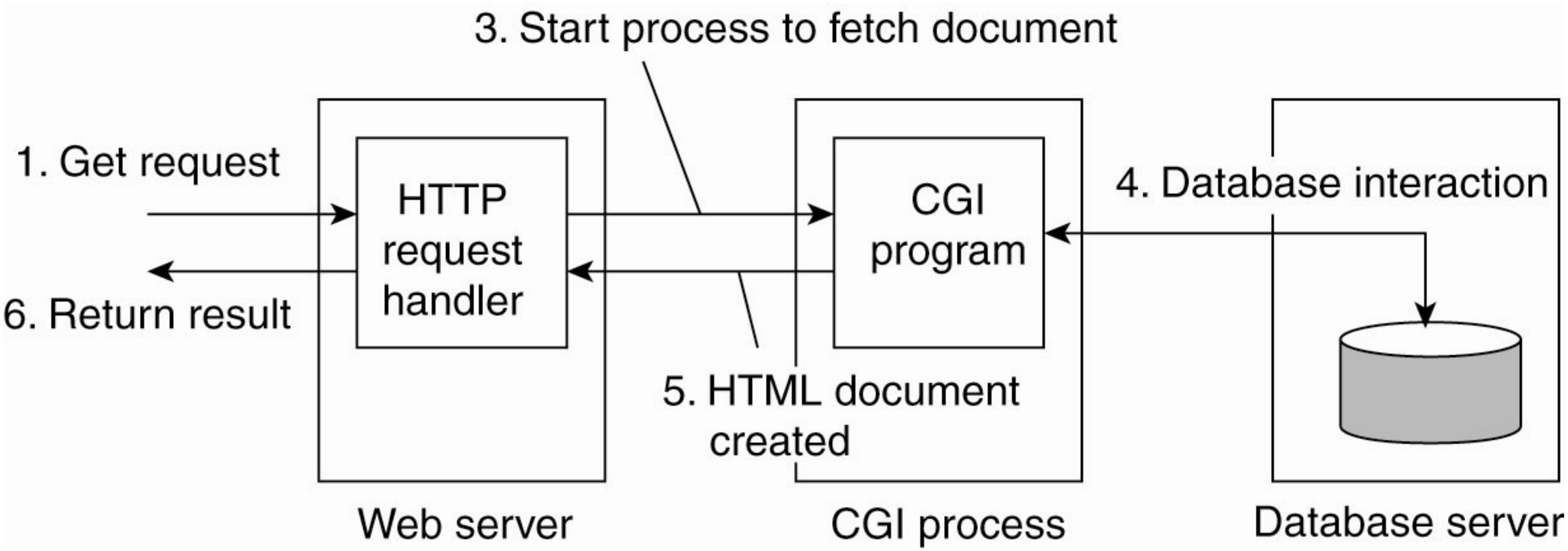


Traditional Web-based systems

- Documents can be (among others):
  - Plain text
  - HTML: HyperText Markup Language
    - Most common
    - Includes instructions expressing how the content should be displayed
    - Allows embedding of links to other documents
  - Images, Audio, Video
  - Applications (e.g. PDF, PS)
- Can include <u>scripts</u> that execute in the client
  - e.g. Javascript

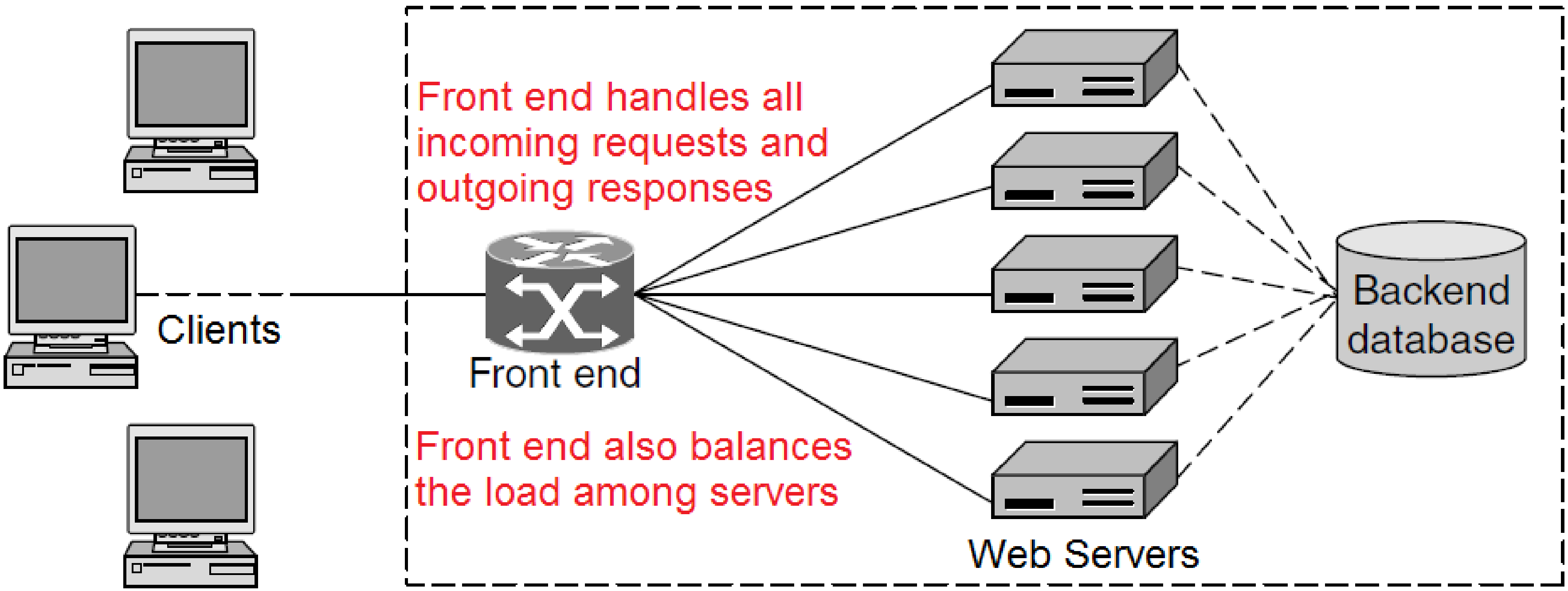
Multitiered architectures

- Common Gateway Interface (CGI)
  - Server runs a program taking user data as input
  - CGIs evolved to **servlets**, **JSP**, and **PHP**
- This has lead to <u>three-tiered architectures</u>



Web server clusters

- Web servers can be clustered <u>transparently</u> to clients, improving performance & availability



Web server clusters

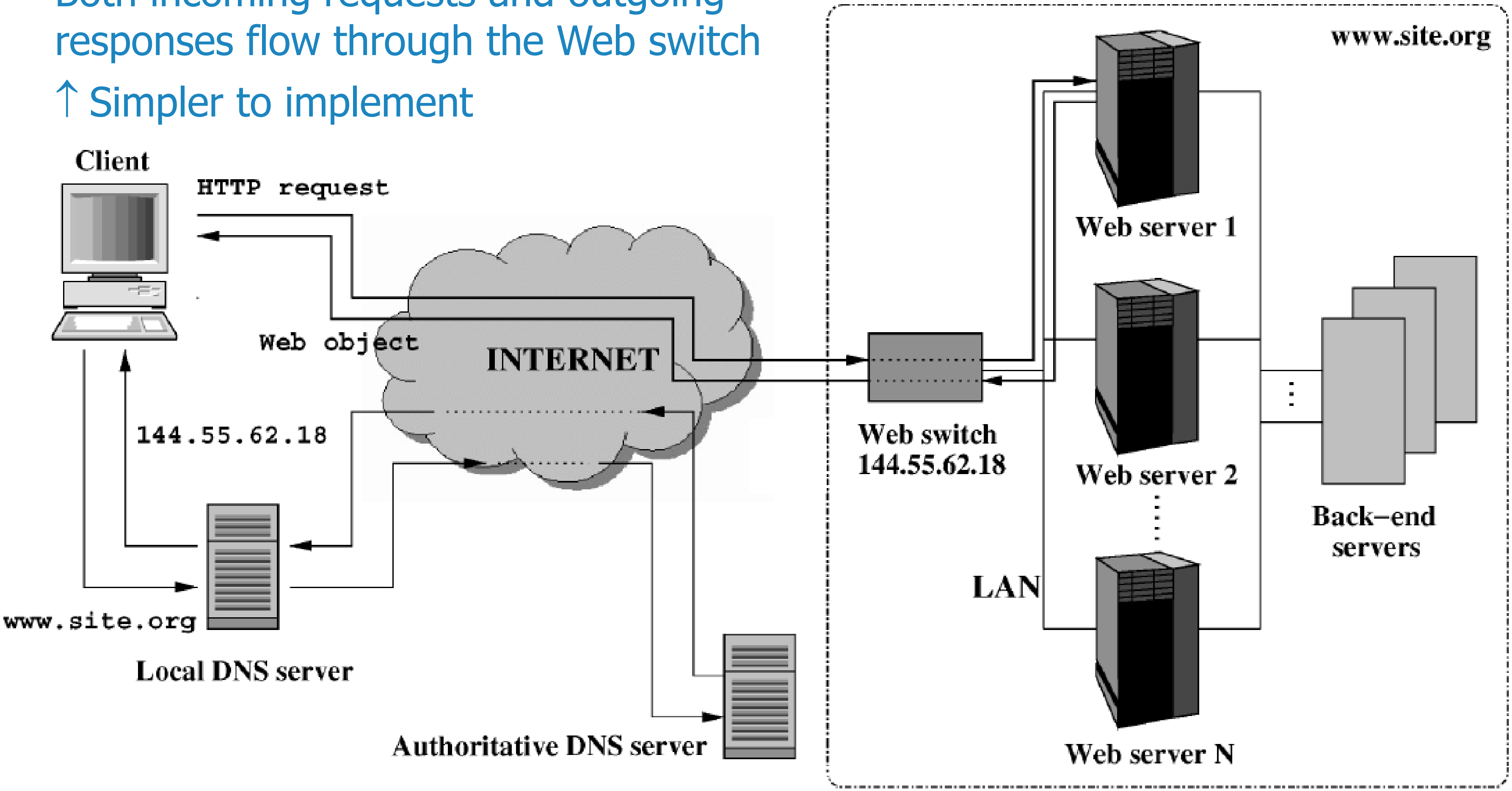
- The front end (a.k.a. Web switch) may get overloaded, so it must be carefully designed

1. **Request routing**
   – Mechanisms to route incoming client requests to their selected target Web server
   – One- or two-way architectures depending on the data flow between the client and the target server
2. **Request dispatching**
   – Policies to select the Web server that is considered best suited to serve each client’s request

Routing: Two-way architecture

Both incoming requests and outgoing responses flow through the Web switch

↑ Simpler to implement

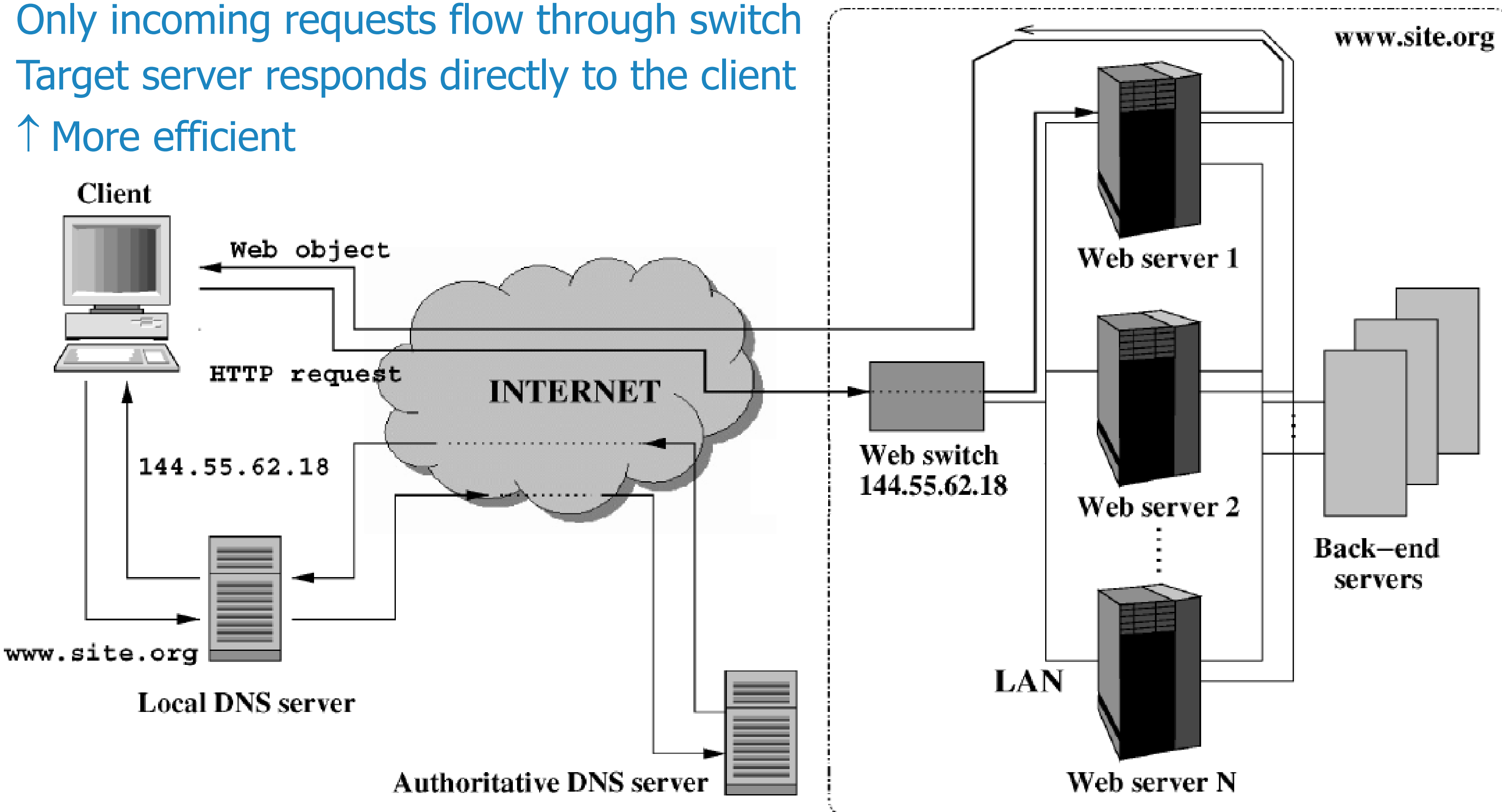


Routing: One-way architecture

Only incoming requests flow through switch

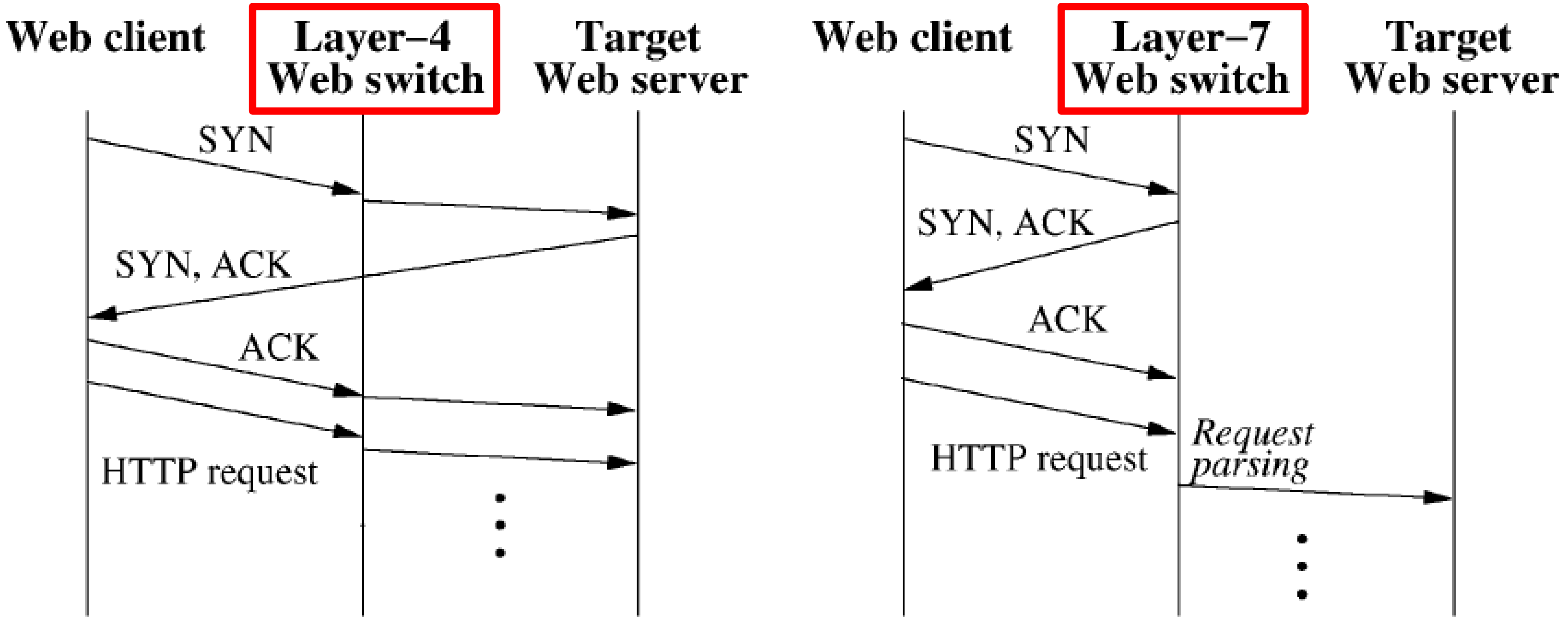
Target server responds directly to the client

↑ More efficient



Request routing

- Two types of switches depending on the OSI protocol stack layer at which the Web switch routes inbound packets to the target server



Layer-4 request routing

a) <u>Packet rewriting</u>
   - Switch (or target server) rewrites the destination / source addresses of inbound / outbound packets with the IP of the target server / web switch

b) <u>Packet tunneling</u>
   - Switch tunnels inbound packets to target server by encapsulating them within other IP packets

c) <u>Packet forwarding</u>
   - Switch forwards inbound packets to the target server through its MAC address (all nodes share now the same IP address)

Layer-7 request routing

a) TCP gateway
   - Switch runs a proxy at the application layer that mediates between the client and the target server

b) TCP splicing
   - Switch runs a proxy at the network layer that splices together the client–switch and the switch–server TCP connections

c) TCP hand-off
   - Switch hands off its TCP connection with the client to the target server, which uses it to communicate directly with the client

Request dispatching

- Request routing has also a big impact on dispatching policies due to the kind of information available at the Web switch

A. <u>Content-blind dispatching</u> (Layer-4 routing)
   - ↑ Efficient
   - ↓ Simple policies

B. <u>Content-aware dispatching</u> (Layer-7 routing)
   - ↑ Sophisticated policies: better profit of caching, better load sharing
   - ↓ Less efficient: higher demand on the front end

Content-blind request dispatching

A. Static algorithms
   – Do not consider any state information
      • Efficient & easy to implement (↑) but naïve decisions (↓)
   ⇒ e.g. Random and Round-Robin algorithms

B. Dynamic algorithms
   – Consider client and/or server state information
      • Better decisions (↑) but overhead to collect state (↓)
   ⇒ e.g. Least Loaded: assign request to the server with the fewest active connections
   ⇒ e.g. Client Affinity: assign consecutive requests from the same client to the same server

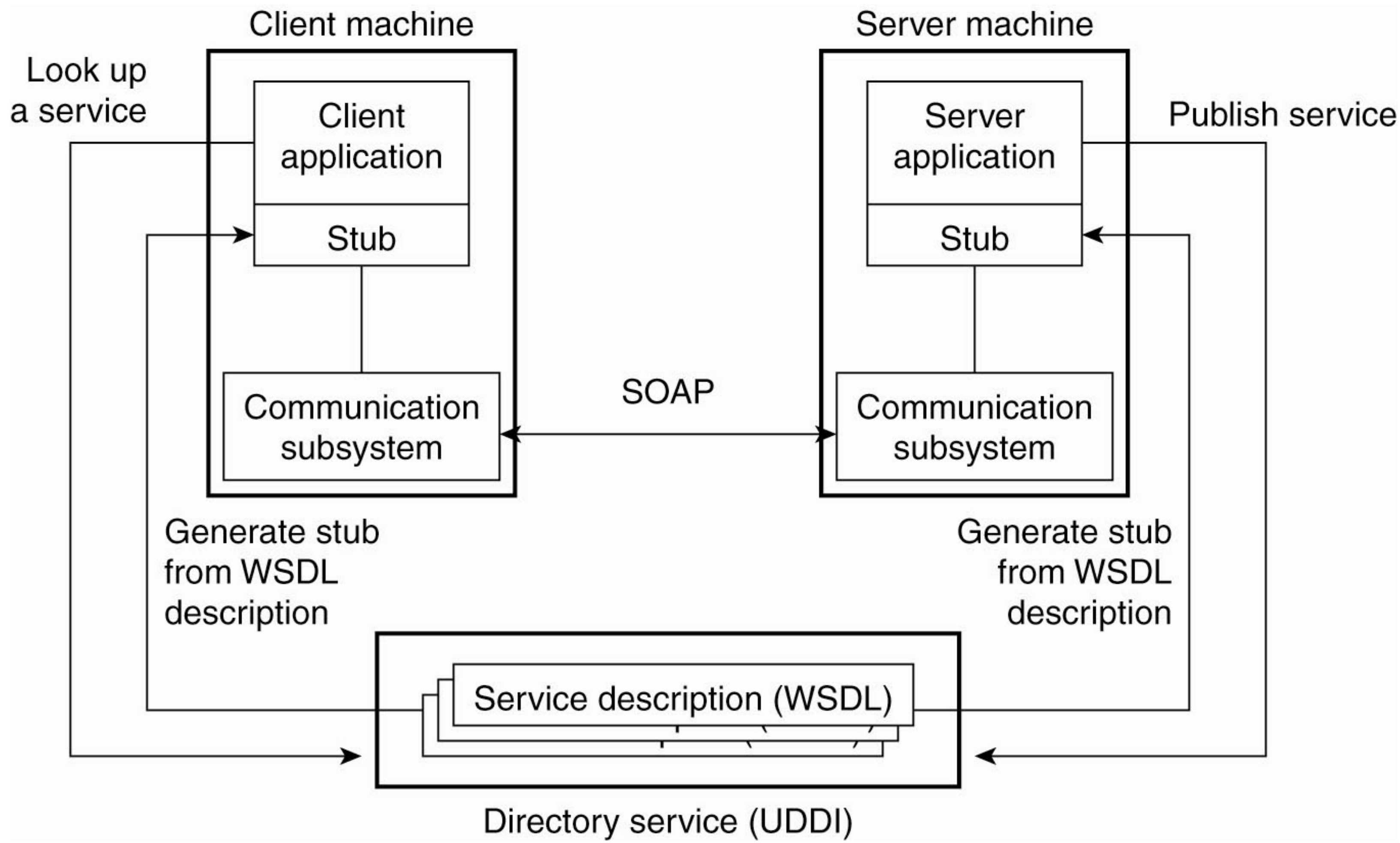
Content-aware request dispatching

A. Dynamic algorithms
   - Consider client and/or server state information
   - ⇒ e.g. Cache Affinity: partition data among servers and assign clients based on the data they access
   - ⇒ e.g. Load Sharing: assign clients to balance load among servers (based on the size of requested file or the expected impact on the server resources)
   - ⇒ e.g. employ specialized servers for certain type of requests (e.g. multimedia, streaming)
   - ⇒ e.g. Client Affinity: avoid the limitations of the IP address identification by using individual client identifiers, such as cookies and SSL identifiers

Web Services

- Go beyond simple user-site interaction and offer **services** to remote applications
  - Communication using Internet standards
- W3C Web Services components
  - A standard way for communication (SOAP)
  - A uniform data representation and exchange mechanism (XML)
  - A standard meta language to describe the services offered (WSDL)
  - A mechanism to register and locate WS-based applications (UDDI)

W3C Web Services

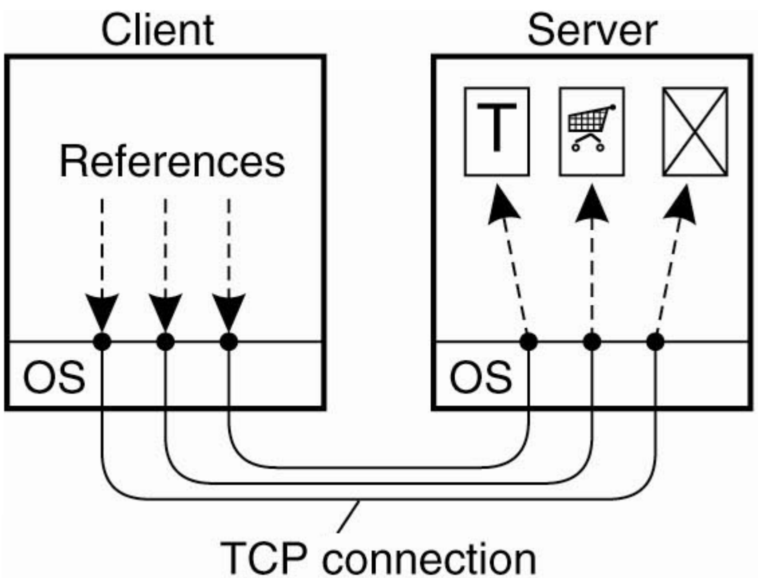


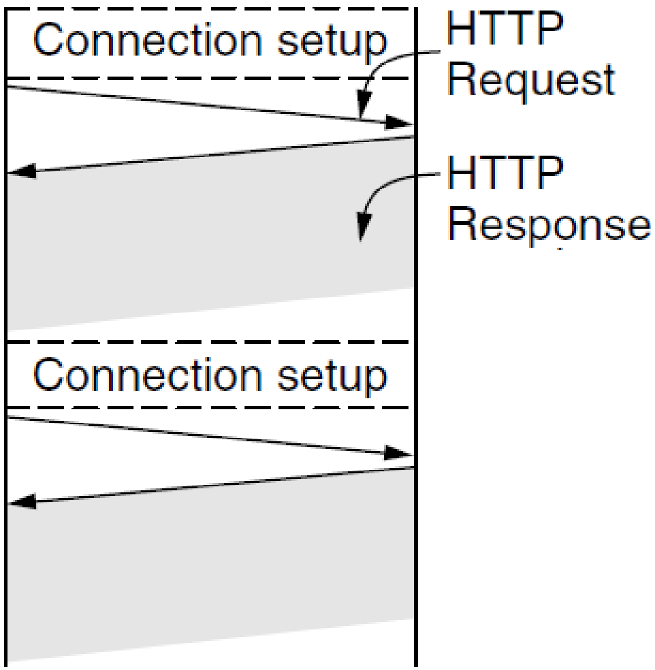
Contents

- Architecture
- **Communication**
- Naming
- Synchronization
- Consistency & replication
- Fault tolerance

HyperText Transfer Protocol (HTTP)

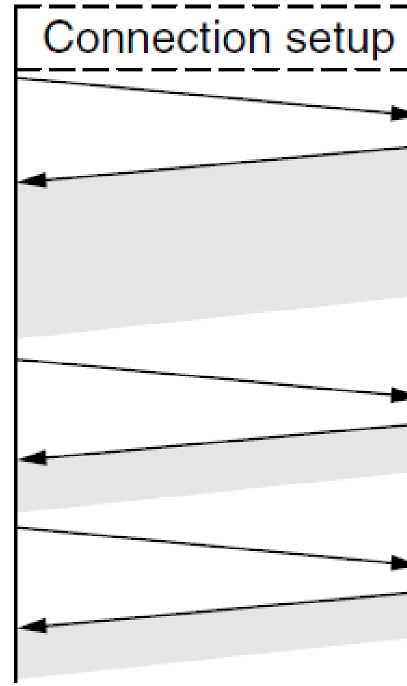
- Simple request-reply protocol running on TCP
- HTTP v1.0 uses non-persistent connections
  - Each request sets up a separate TCP connection

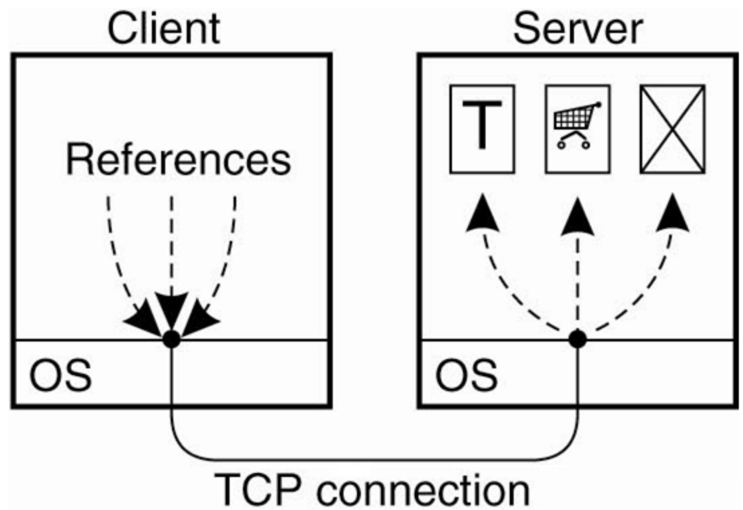


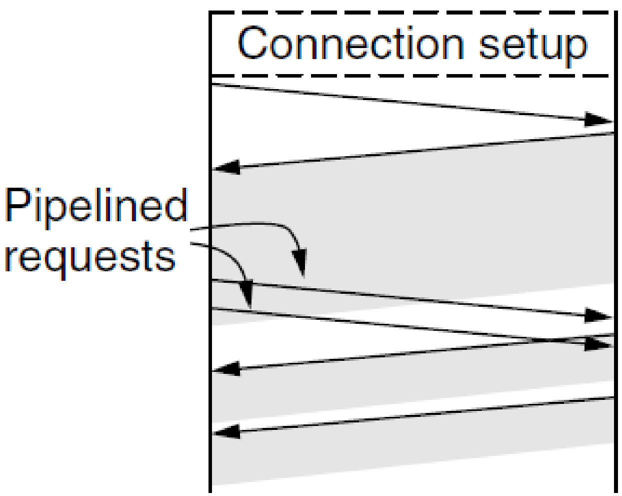


HyperText Transfer Protocol (HTTP)

- HTTP v1.1 uses persistent connections
  - Use same TCP connection to issue several requests







- Pipelining: Send several requests in a row before the response for the first one arrives

HyperText Transfer Protocol (HTTP)

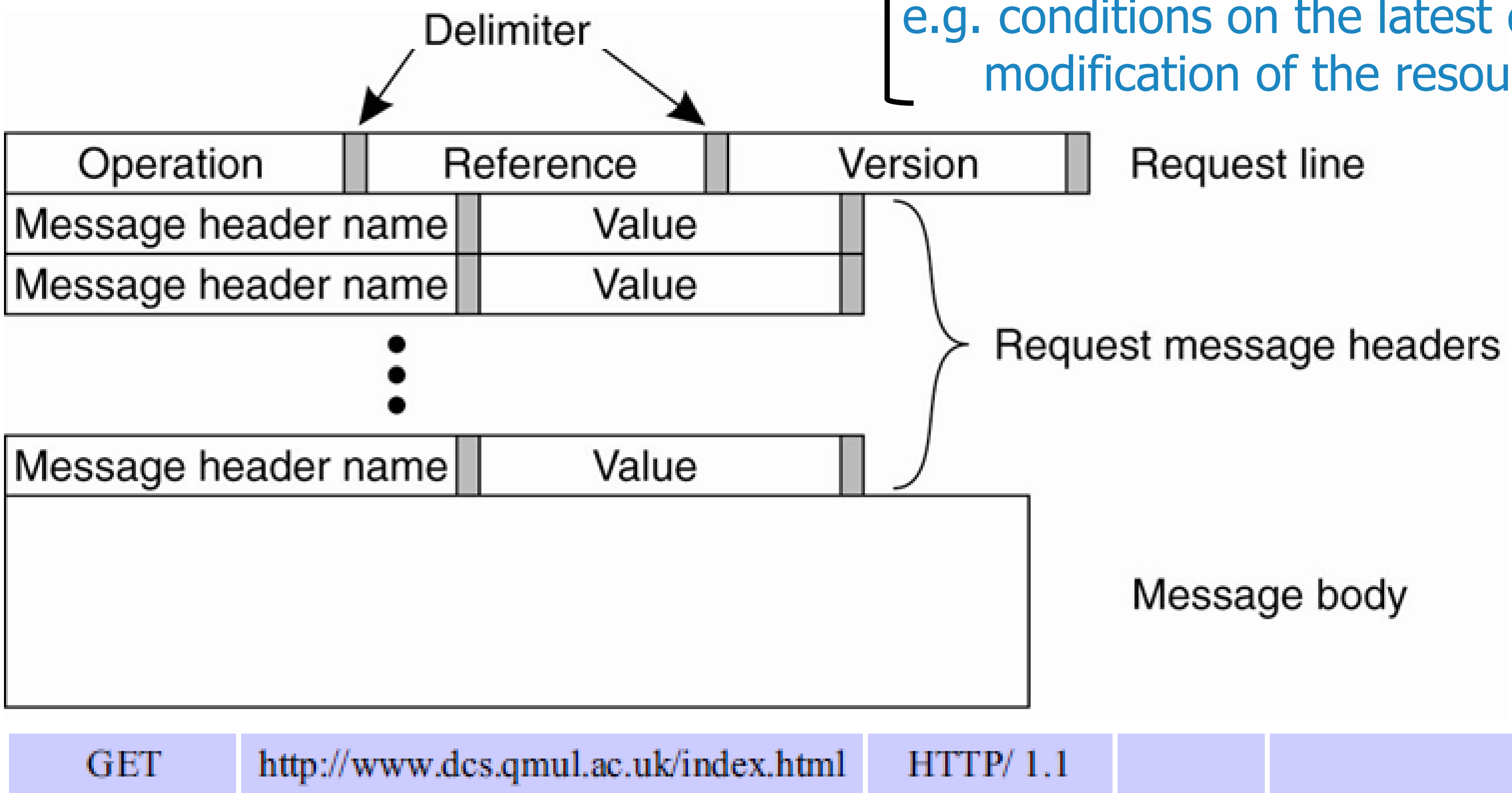
- Operations supported by HTTP

Operation	**Description**
Head	Request to return the header of a resource
Get	Request to return a resource to the client. Can be a document or the output of a program execution
Put	Request to store data as a resource
Post	Provide data that are to be handled by a resource
Delete	Request to delete a resource

HyperText Transfer Protocol (HTTP)

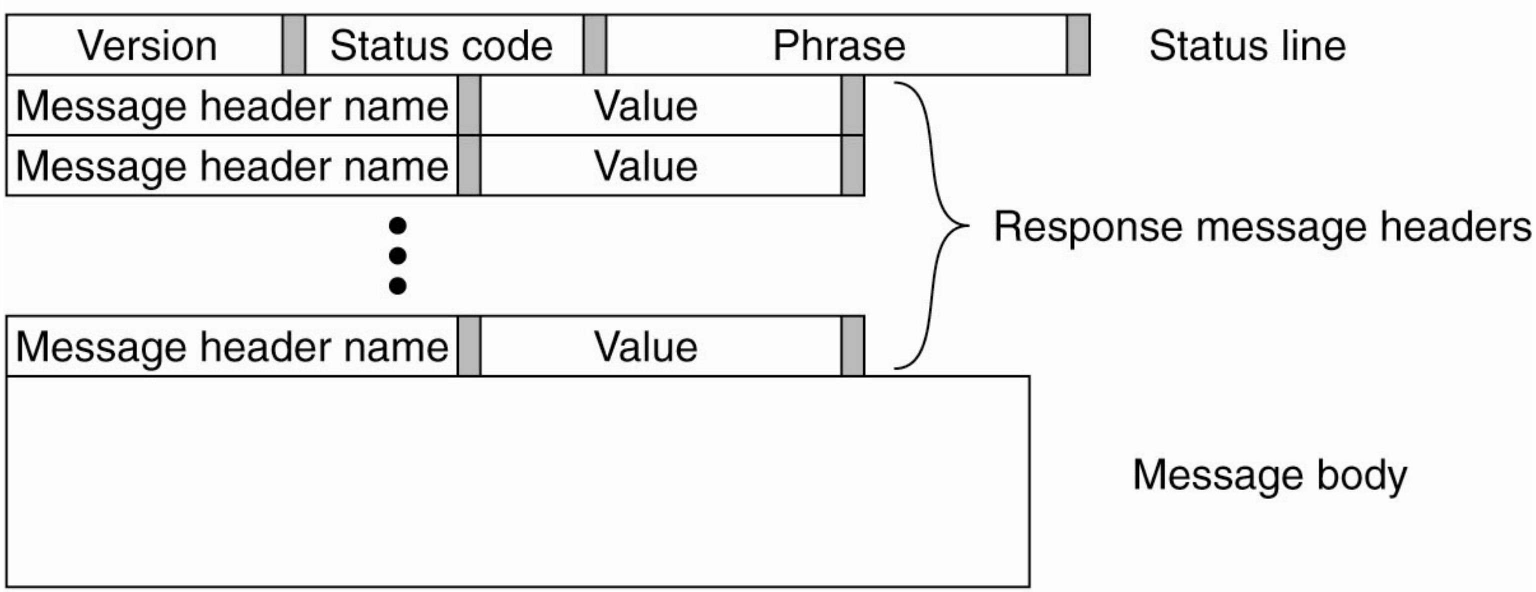
- HTTP request message
  - Request message headers

e.g. acceptable content types
e.g. acceptable document encoding
e.g. list of client’s credentials
e.g. conditions on the latest date of modification of the resource



HyperText Transfer Protocol (HTTP)

- HTTP response message
  - Response message headers
    - e.g. time of last modification
    - e.g. response's expiration time
    - e.g. URL to redirect request



HTTP/1.1	200	OK		resource data

Simple Object Access Protocol

- SOAP is the standard protocol for communication between W3C Web services
- Based on XML (extends XML-RPC)
  - ↑ XML allows self-describing data (portable)
  - ↓ Parsing overhead
  - ↓ Not meant to be read by human beings
- SOAP is platform independent, but bound to an underlying transfer protocol (a.k.a. carrier)
  - Currently HTTP, SMTP
  - Transfer protocol specifies the recipient's address

Simple Object Access Protocol

- SOAP message consists of two elements which are jointly put inside an Envelope
    1. Header (optional) contains information relevant for nodes along the path from sender to receiver
        - e.g. routing, authentication, transactions
    2. Body (mandatory) contains the actual message
- SOAP offers 2 different styles of interactions
    1. Document-style: Conversational mode of XML message exchange (placed directly in the body)
    2. RPC-style: XML representation of a method invocation-response

SOAP example

REQUEST

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

XML schema for SOAP envelopes

XML schema for the service description

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

RESPONSE

```
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://example.org/stock-service">
  <env:Body>
    <s:GetStockQuoteResponse>
      <s:StockPrice>45.25</s:StockPrice>
    </s:GetStockQuoteResponse>
  </env:Body>
</env:Envelope>
```

Representative State Transfer

- RESTful Web Services
  - Simpler approach where clients use URLs and HTTP operations (GET, PUT, DELETE, POST) to manipulate resources represented in XML/JSON

REQUEST

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

RESPONSE

```
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>
```

Web Services Description Language

- WSDL is a formal language for describing precisely the service provided by a W3C WS
  a) Interfaces of operations, data types, message exchange patterns
  b) Binding: choice of protocols: e.g. SOAP/HTTP
  c) Endpoint of the service: e.g. URI
- Based on XML
- Can be automatically translated to client and server stubs
- Analogous to IDL in RPCs

Contents

- Architecture
- Communication
- **Naming**
- Synchronization
- Consistency & replication
- Fault tolerance

Naming

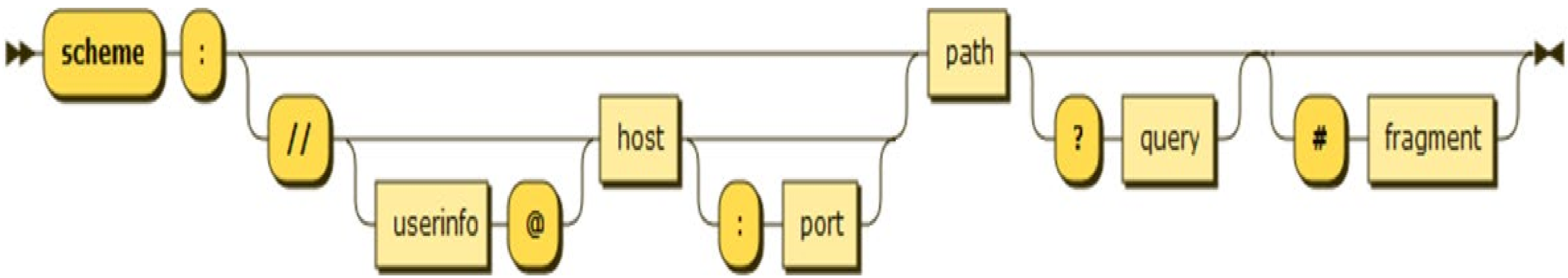
- Resources referred by URIs
- Two forms of Uniform Resource Identifiers:

a) URN: Uniform Resource Name
   - Globally unique, location independent, and persistent reference to a resource
   - e.g. urn:ietf:rfc:3187

b) URL: Uniform Resource Locator
   - Includes information on how to locate and access the resource
   - e.g. http://tools.ietf.org/html/rfc3187.html

URI syntax

- scheme: Application-level protocol for transferring the resource (e.g. http, ftp, mailto, file, data)
- authority: [userinfo (username-password)@] + DNS name/IP address of host + [:port]
- path: Pathname of the resource in host file system
- query: Input query to the resource
- fragment: Direction to a specific element of resource



URI examples

```
          userinfo       host          port
          ┌───┴───┐ ┌──────┴──────┐ ┌┴┐
https://john.doe@www.example.com:123/forum/questions/?tag=networking&order=newest#top
└─┬─┘   └───────────┬────────────┘└───────┬───────┘ └───────────┬────────────┘ └┬┘
scheme          authority                path                 query          fragment

ldap://[2001:db8::7]/c=GB?objectClass?one
└┬─┘   └─────┬─────┘└─┬─┘ └──────┬──────┘
scheme   authority   path      query

mailto:John.Doe@example.com
└─┬──┘ └────────┬─────────┘
scheme         path

news:comp.infosystems.www.servers.unix
└┬─┘ └───────────────┬───────────────┘
scheme              path

tel:+1-816-555-1212
└┬┘ └──────┬──────┘
scheme    path

telnet://192.0.2.16:80/
└─┬──┘   └─────┬─────┘│
scheme     authority  path
```

Examples of URIs with various schemes

Universal Description Discovery Interface

- UDDI: name & directory service for W3C WS
- Stores service descriptions in the form of WSDL documents
- Clients can look up services <u>by name</u> (name service) or <u>by attribute</u> (directory service)
- Service descriptions can be <u>replicated</u> across several servers
- Any server may respond to queries without any interaction with rest
  - Unlike LDAP, which partitions data among servers

Universal Description Discovery Interface

- Changes to a service description must be performed at the owner server (i.e. primary)
  - When a client publishes a service at a server, this server becomes its owner
  - Ownership can be passed on to another server
- Changes are propagated periodically by organizing servers in a logical ring
  - Server S_1 advertises its changes to its successor S_2
  - S_2 requests (pulls) its missing changes from S_1
  - S_2 forwards its own advertisement along the ring

Contents

Synchronization

- Not an issue in traditional web-based systems
  - Nothing to synchronize since servers did not exchange information with other servers
  - WWW was a read-mostly system. Updates were done by a single entity
- But today distributed authoring of documents is emerging
  - Synchronization is needed
  - Let's see how this is accomplished in collaborative editing of documents: Google Docs
    - Based on **Operational Transformation (OT)**

Google Docs collaboration

- A document is stored in the server as a list of chronological changes: revision log
  - 3 basic types of changes: inserting text, deleting text, and applying styles to a range of text
    - e.g. {InsertText ‘SDX' @10}, {DeleteText @9-11}, {ApplyStyle bold @10-20}
  - Append each change to the end of the revision log
- Collaborative protocol to sync changes
  1. Each editor sends changes to the server and waits for acknowledgement
     - Changes during this period are keep in a pending list (never send more than one change at a time)

Google Docs collaboration

2. For each change, server updates revision log, acknowledges a new revision to the sender and sends change to the other editors
3. Each editor transforms incoming changes against its pending changes so that they make sense relative to the local version of the document
   - Operational Transformation to define the different ways that InsertText, DeleteText, and ApplyStyle changes can be paired and transformed against each other
     - OT: https://drive.googleblog.com/2010/09/whats-different-about-new-google-docs_22.html

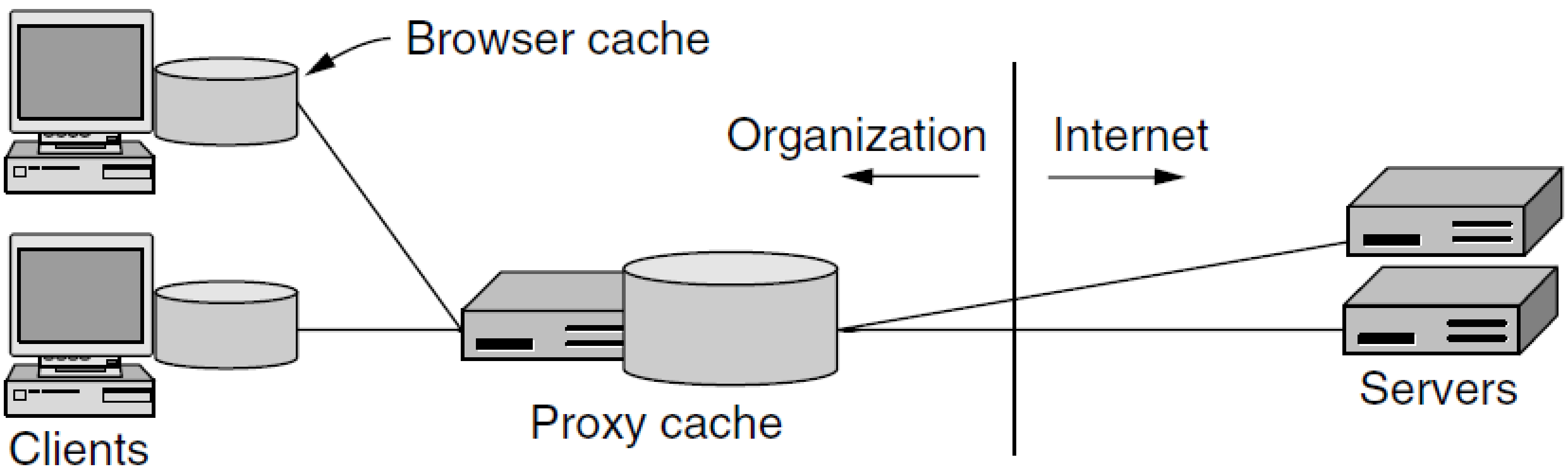
– Sync example: https://drive.googleblog.com/2010/09/whats-different-about-new-google-docs.html

Contents

- Architecture
- Communication
- Naming
- Synchronization
- **Consistency & replication**
- Fault tolerance

Client-side caching

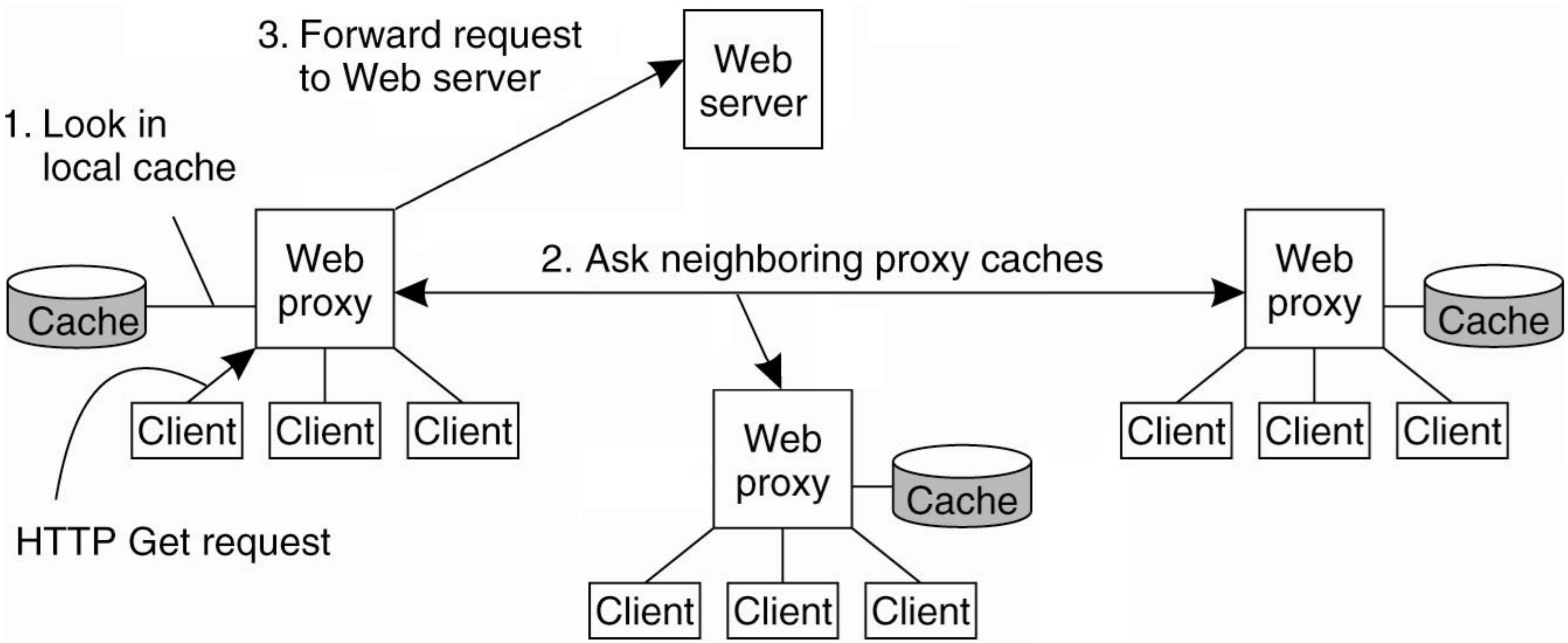
1. Browser's cache
2. Web proxy caching
   - Site installs a separate **proxy server** that passes all requests from local clients to the Web servers
   - Proxy subsequently caches incoming documents



Web proxy caching

a) Hierarchical caches (institution, region, nation): On cache miss, request is moved upward

b) Cooperative caching: On cache miss, the Web proxy checks the neighboring proxies



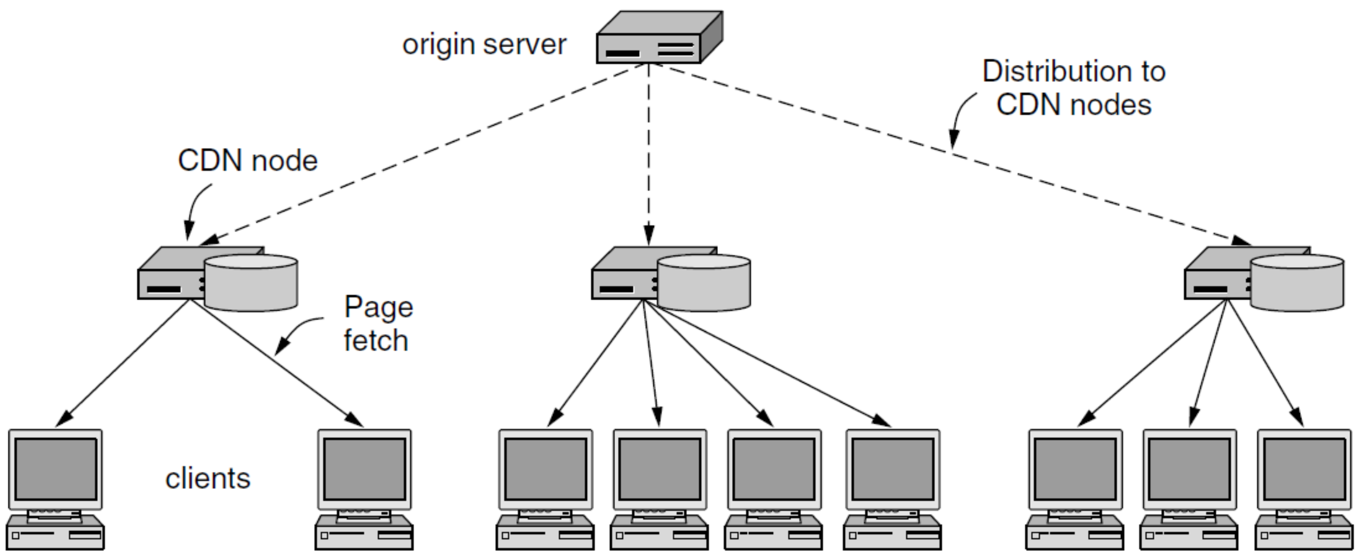
Cache-consistency protocols

1. Always verify validity by contacting server
   - Use `If-Modified-Since` HTTP request header to download the document only if it has changed
   - Strong consistency (↑) but server must be contacted for each request (↓)
2. Age-based consistency (e.g. Squid proxy)
   - Assign an expiration time to each document
     - Depends on the last modification time of a document and the time when it was cached
     - Document is considered valid until this time
   - Better performance (↑) but weak consistency (↓)

Contents

Content Delivery Networks

- Idea: distributed Web server
  - CDN nodes **replicate** contents across the Web and deliver them to clients on behalf of origin site
  - ↑ Reduce the load on origin server
  - ↑ Bring content closer to clients (reduce latency)



Content Delivery Networks

- What content to replicate into CDN servers?
  - Full-site: the entire origin site is replicated
  - Partial-site: only embedded objects are replicated
  - Caching results of previous queries is also possible
- How to choose the best CDN server to serve the client's request?
  - See 'Request dispatching policies' (slides 13-15)
  - Typical metrics considered in CDNs: distance to client, client's perceived latency, load of servers

Content Delivery Networks

- How to enforce consistency between servers?
  - Primary-based approach: updates are carried out at the origin server
  - Typically, CDN servers use a ‘pull’ approach to retrieve updated content from the origin server
    - A ‘push’ approach could be used for specific contents if the expense of bandwidth is worth
  - Origin server can instruct CDN servers about how long the content is to be considered fresh
    - Optionally, CDN servers can cooperate with each other in case of cache misses (cooperative caching)

Content Delivery Networks

- How to route client requests to CDN servers?
  ⇒ <u>redirection schemes</u>

A. HTTP redirection

- – The `Location` HTTP header in the response tells the client to resubmit its request to a CDN server
- ↑ Medium-grain granularity: individual Web pages
- ↑ Allows content-aware dispatching
- ↓ Lack of transparency
- ↓ Overhead: adds an address resolution and an extra message round-trip time for every request

Content Delivery Networks

B. URL rewriting

- Rewrite links for the embedded objects within the returned page to redirect client to CDN servers
- e.g. http://www.foo.com/images/logo.gif ⟹

http://a9.g.akamai.net/7/9/21/aaa7a80f016a2c/www.foo.com/images/logo.gif

↑ Fine-grain granularity: embedded objects

↑ Allows content-aware dispatching

↓ Additional load on the origin server to dynamically generate every Web page

↓ DNS overhead: an additional address resolution is needed for each embedded object

Content Delivery Networks

C. DNS redirection

– CDN takes control of the DNS zone of origin site

– Modify DNS resolution so that each request is redirected to a different CDN server

↑ Transparent: seamless integration with DNS

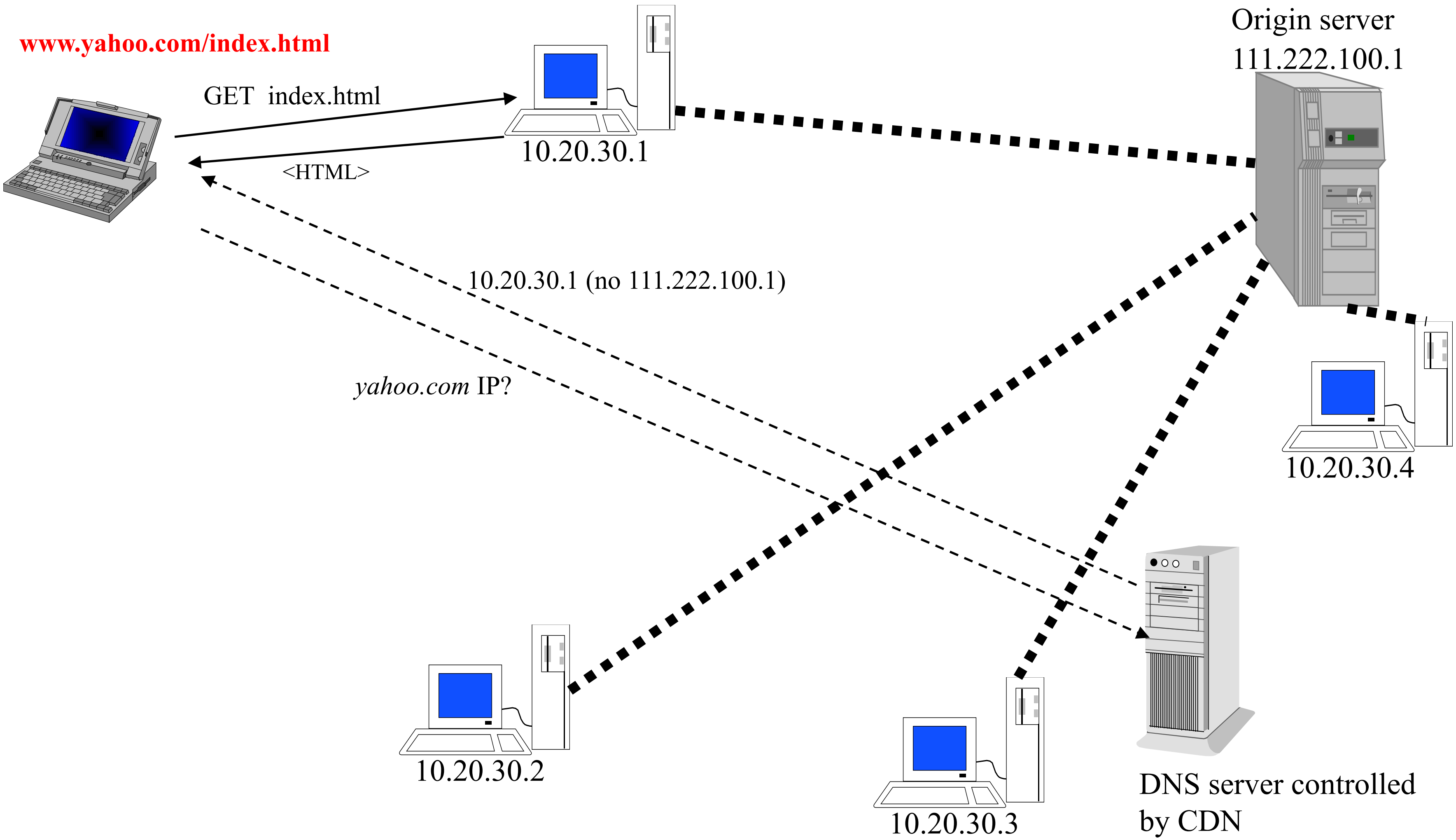
↑ No redirection overhead

↓ Only content-blind dispatching

↓ Caching at the browser and intermediate servers prevents many requests to contact the origin site

↓ Coarse-grain granularity: all client's requests in a session will reach the same server

DNS redirection example



Contents

Fault tolerance

- Mainly achieved through client-side caching and server replication

Summary

- WWW as a distributed system
  - From traditional client-server architectures for fetching hyperlinked documents to Web Services
  - Use of standardized protocols
    - Communication: HTTP, SOAP
    - Naming: URI, UDDI
  - Caching and replication for improving performance and fault tolerance (e.g. web proxy caching, CDN)
- Further details:
  - [Tanenbaum, 2nd Ed.]: chapter 12
  - [Coulouris]: chapters 1.6 and 9