UCL Université catholique de Louvain

ip networking lab
ucl, louvain-la-neuve, belgium

Computer Networking : Principles, Protocols and Practice

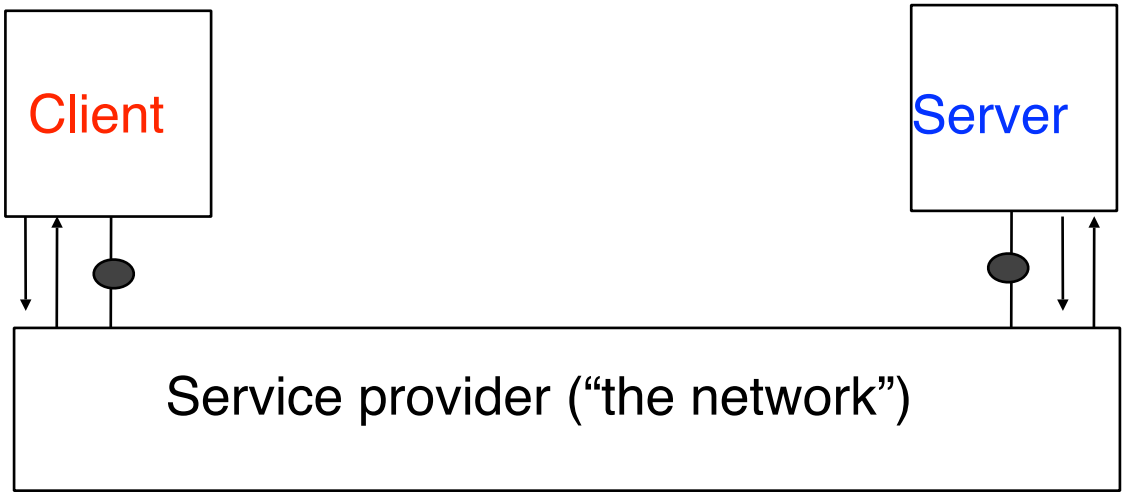
Part 2 : Applications

Olivier Bonaventure
http://inl.info.ucl.ac.be/

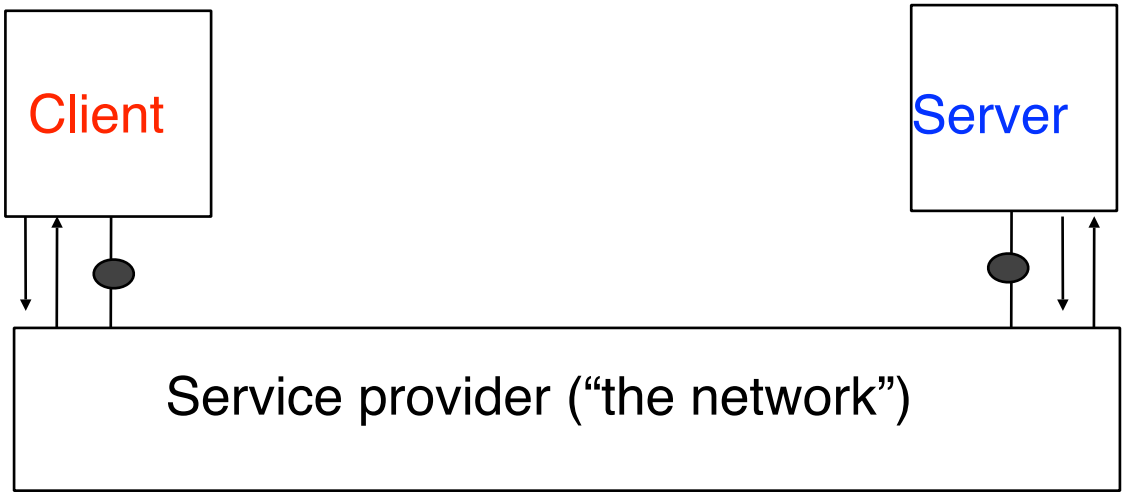
The Application Layer

- Contents
  - The client-server model
  - Name to address resolution
  - email
  - world wide web
  - peer-to-peer applications

The client server model

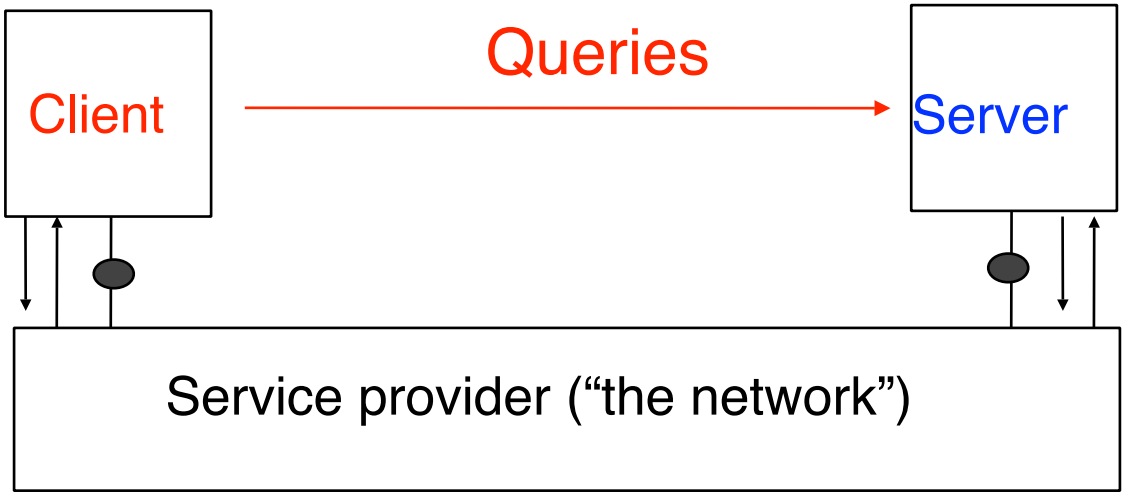


The client server model



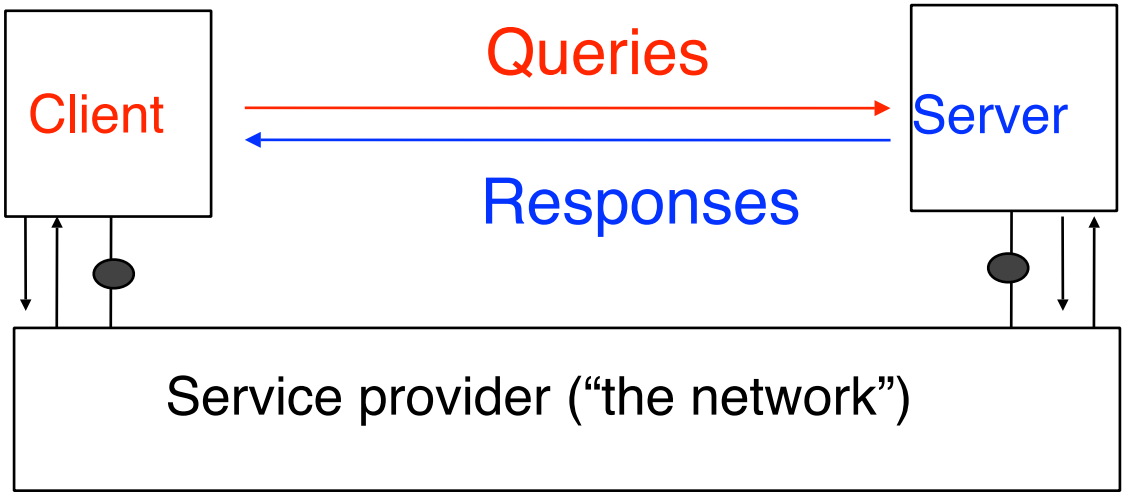
- Client
  - interacts with server through transport layer
  - sends queries or commands

The client server model



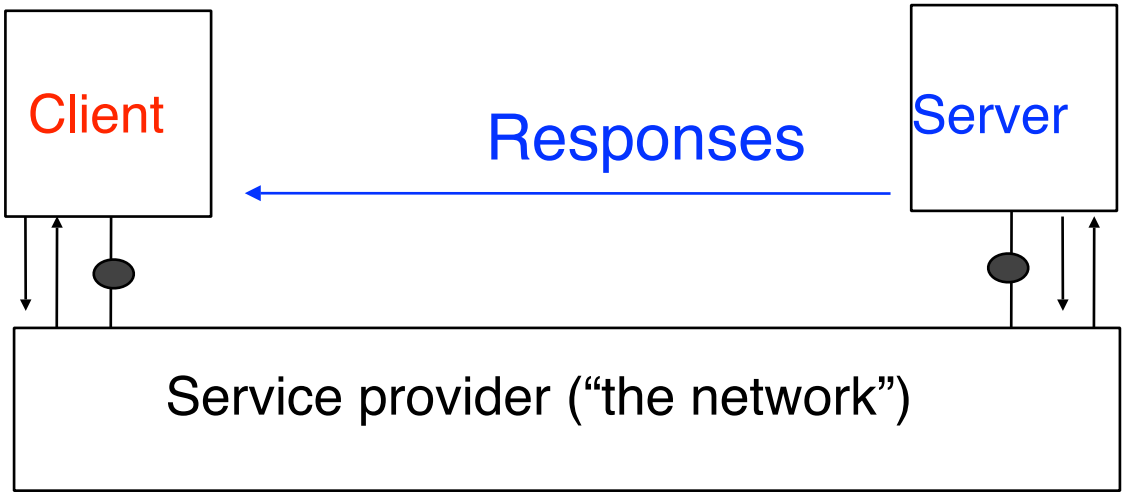
- Client
  - interacts with server through transport layer
  - sends queries or commands

The client server model



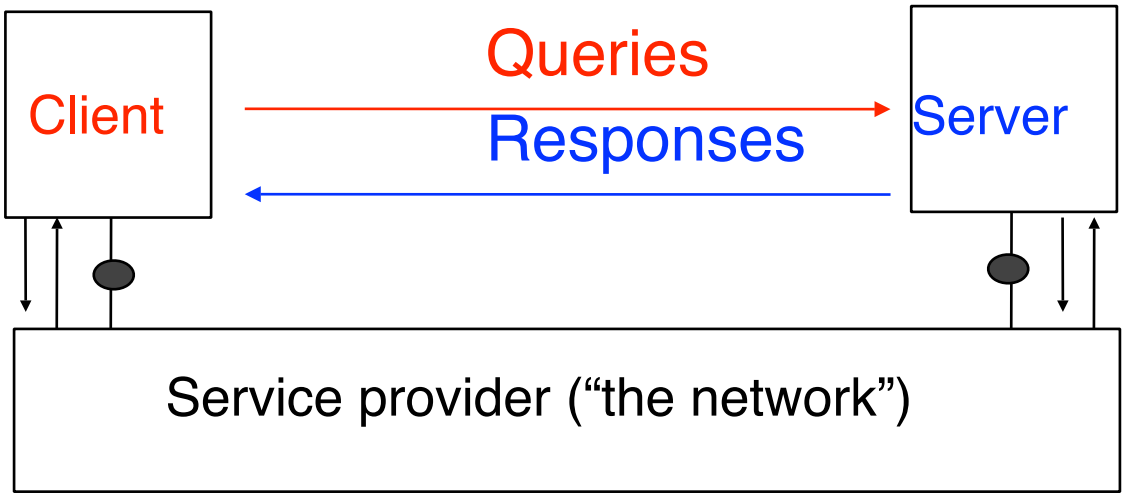
- Client
  - interacts with server through transport layer
  - sends queries or commands
- Server
  - Answers the queries received from clients
  - Executes the commands from clients
  - Many clients can use the same server
- Example : email, www, ...

The client server model (2)



- Client and servers interact with service provider
- Both the client and the server must speak the same language
  - Application-level protocol : set of syntactical and semantical rules that define the messages exchanged between the client and the server and their ordering

The client server model (2)

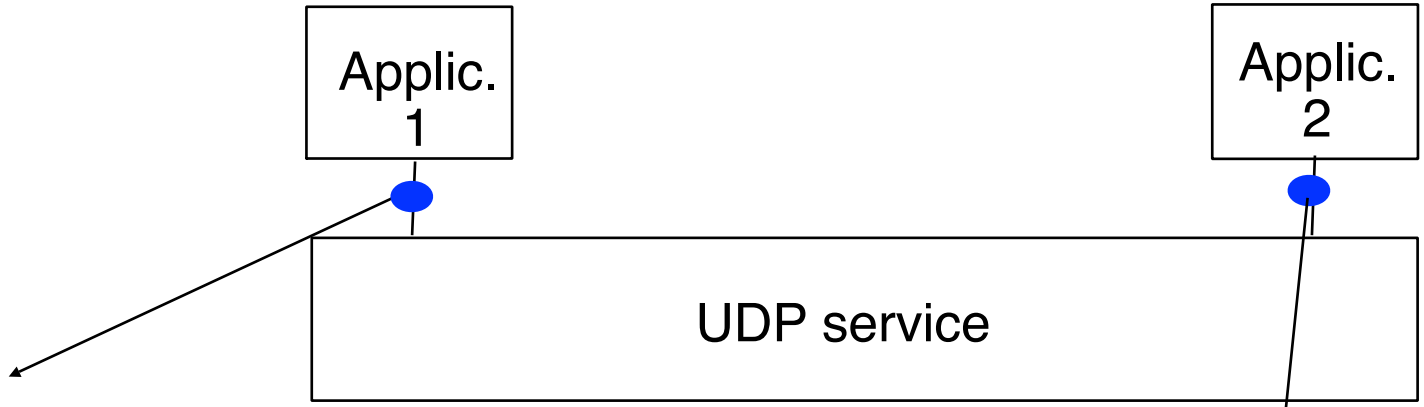


- Client and servers interact with service provider
- Both the client and the server must speak the same language
  - Application-level protocol : set of syntactical and semantical rules that define the messages exchanged between the client and the server and their ordering

Transport service on the Internet

- On the Internet, applications can use two different transport services
  - The service provided by the User Datagram Protocol (UDP)
    - unreliable connectionless service with error detection
  - The service provided by the Transmission Control Protocol (TCP)
    - reliable bytestream connection-oriented service

UDP service

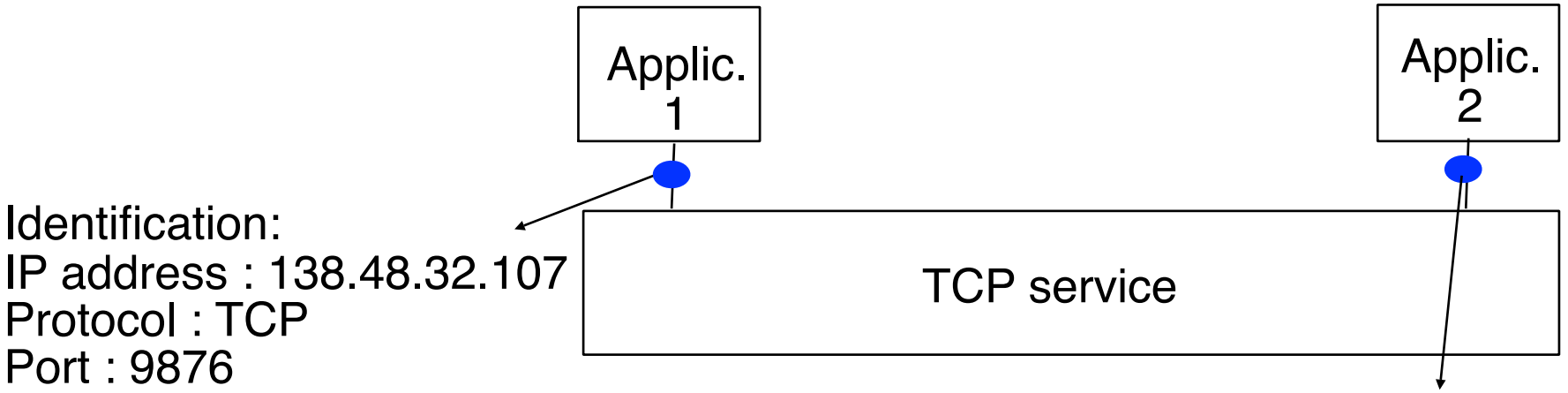


Identification:
IP address : 2001:6a8:3080:2:217:f2ff:fed6:65c0
Protocol : UDP
Port : 1234

Identification
IP address : 2001:4860:a005::68
Protocol : UDP
Port : 53

- Identification of an application
  - IP address + UDP + port number
- Characteristics of UDP service
  - connectionless
  - unreliable
    - messages can be lost
    - transmission errors can be detected but not recovered
    - sequence is not preserved
  - Maximum size of messages : almost 64 Kbytes

TCP service



- Identification of an application
  - IP address + UDP + port number
- Characteristics of TCP service
  - connection-oriented
  - bidirectional
  - reliable
  - byte stream
  - connection release
    - abrupt if initiated by service provider
    - graceful or abrupt if initiated by user

Internet applications

- Contents
  - The client-server model
  - Name to address resolution
  - email
  - world wide web
  - peer-to-peer applications

Names and addresses

Names and addresses

- Address of a server
  - IP Address of the host on which the server is running port number (TCP or UDP)
    - usually well known port number

Names and addresses

- Address of a server
  - IP Address of the host on which the server is running port number (TCP or UDP)
    - usually well known port number
- Drawback
  - Difficult to remember an IP address for a human

Names and addresses

- Address of a server
  - IP Address of the host on which the server is running port number (TCP or UDP)
    - usually well known port number
- Drawback
  - Difficult to remember an IP address for a human
- Idea
  - Replace IP address by a hostname
  - Easier for humans
    - but IP address is necessary to contact server
  - How to translate a hostname in an IP address ?

Names and addresses (2)

- `hosts.txt` file
  - contains the name-address table
  - must be updated regularly

```
#
# Internet host table
#
127.0.0.1         localhost
138.48.32.99      babbage
138.48.32.100     leibniz
138.48.32.1       routeur
138.48.32.92      corneille
138.48.32.107     backus
138.48.20.152     arzach
138.48.32.137     almin01
138.48.32.170     duke
```

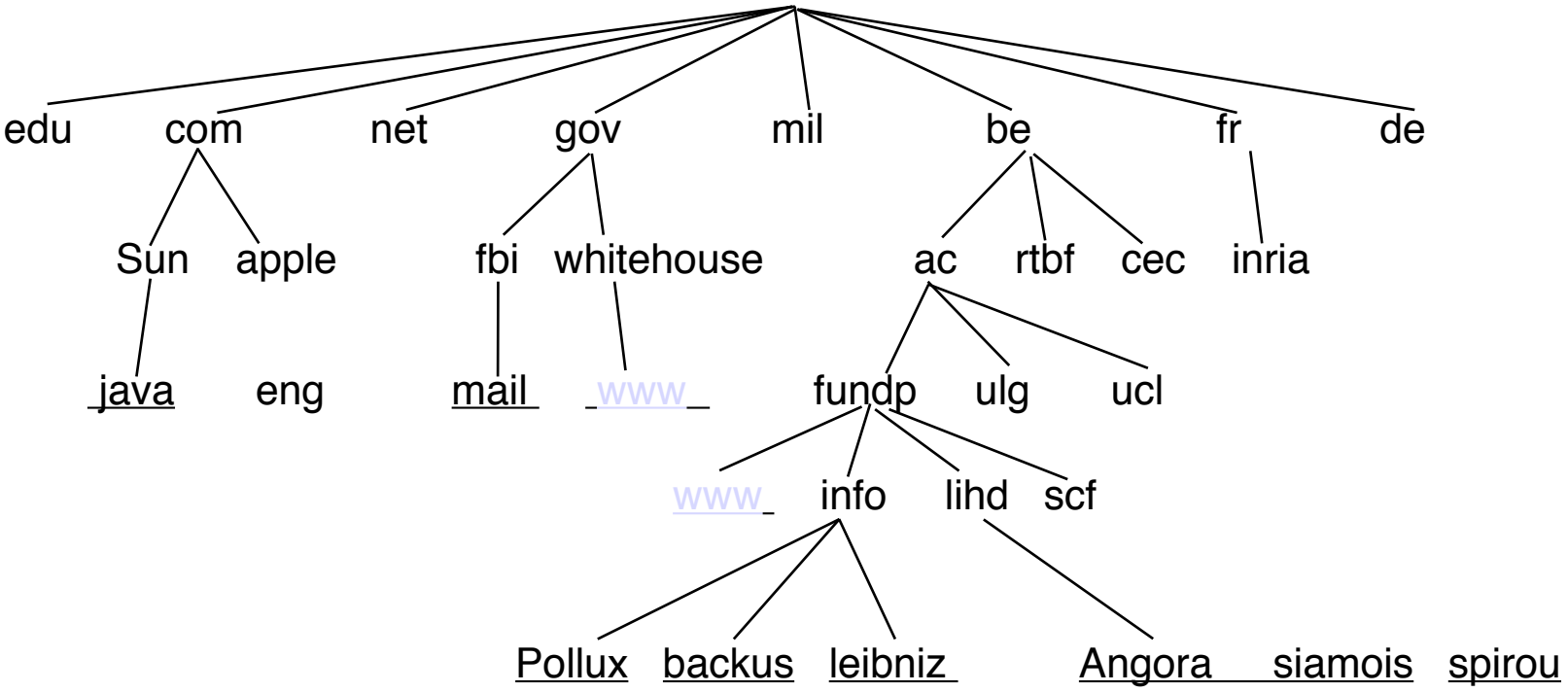
  - cannot be used in a large network

Hostnames

- Requirement
  - Host names should be unique
- How to achieve this in a scalable manner ?
  - Introduce hierarchy
  - Each hostname is composed of two parts
    - domain name (globally unique)
    - hostname (unique within a given domain)
  - How to uniquely distribute domain names ?
    - Introduce hierarchy
    - A small number of top-level domain names
    - Inside each top-level domain, allocate uniquely second level domain names
    - Inside each seconde-level domain, allocate uniquely either third-level domain names or host names,
    - ...

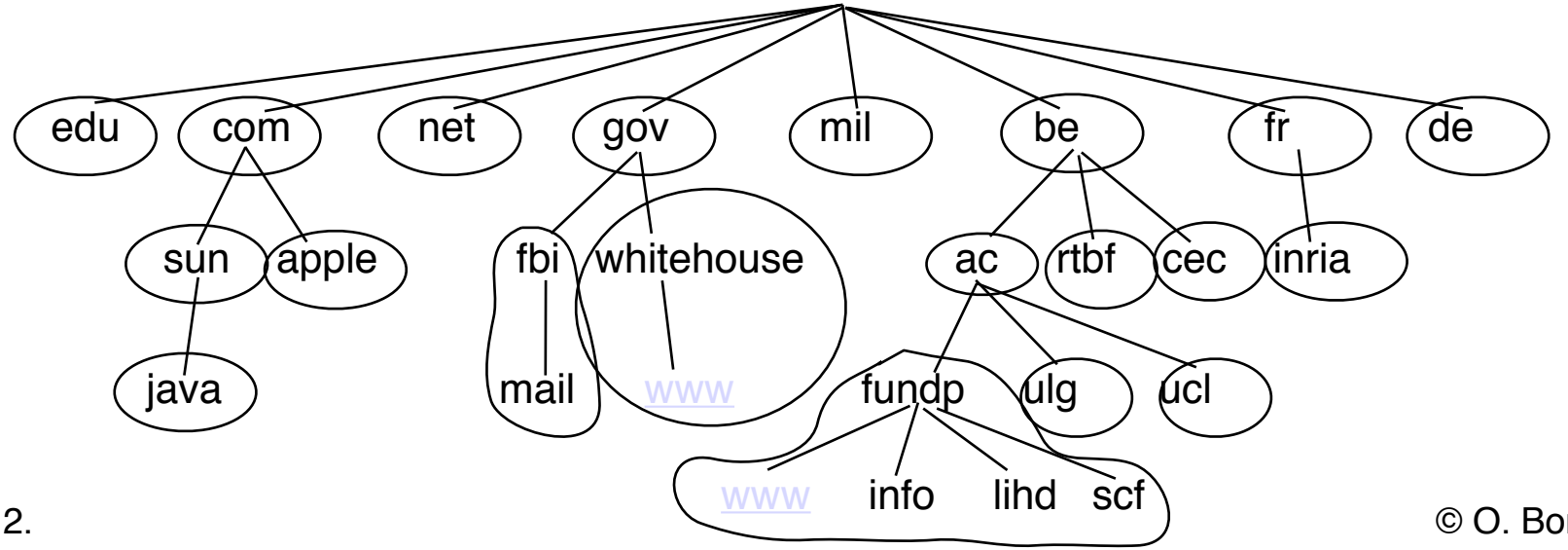
Host names and domain names

- Tree of all host names



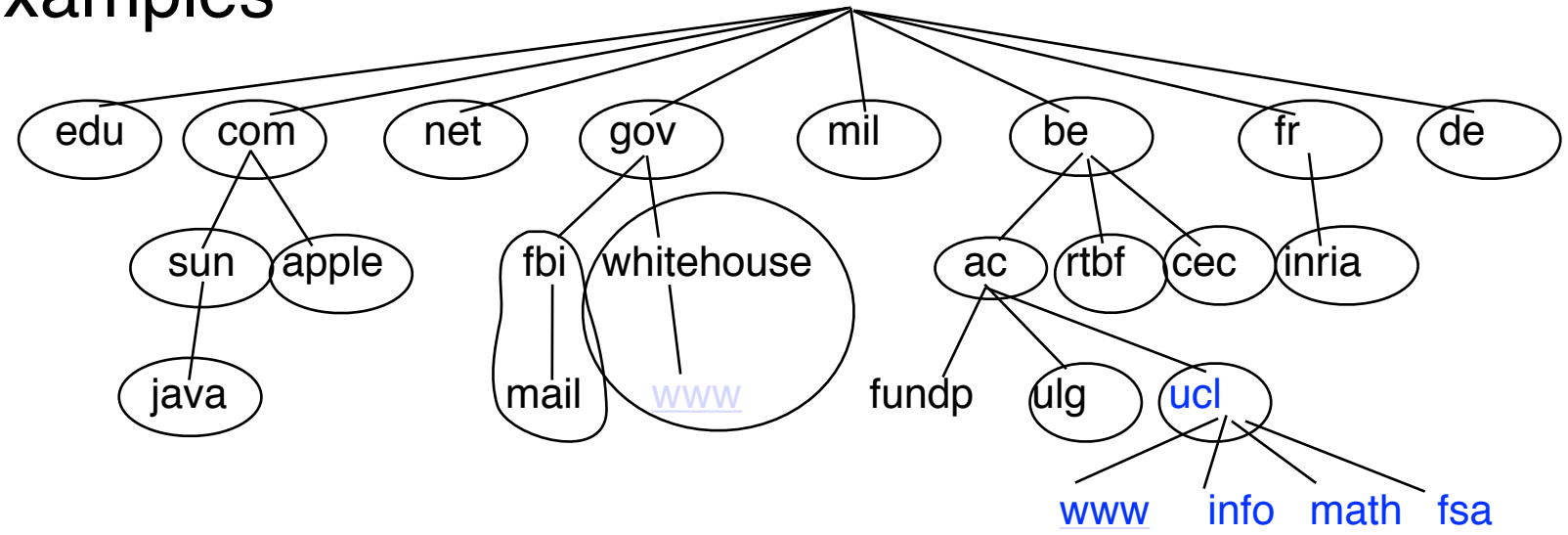
How to translate names into addresses ?

- How to efficiently translate a host name ?
  - By using a centralised database
    - there are more than 1 billion host names today
  - By using a distributed database
    - DNS : Domain Name System
    - relies on the hierarchy of domain names
    - there is one server responsible for each domain and this server must be queried to translate host names inside this domain



How to translate names into addresses ?

- Domain Name Service (DNS)
  - Each DNS server is responsible for a domain and knows
    - The IP addresses of all host names in this domain
    - The IP addresses of the DNS servers responsible for subdomains
  - Examples



    - java.sun.com
    - www.ucl.ac.be

DNS resolver

- To be able to translate name to addresses, a DNS implementation needs
  - to know actual list of IP addresses of root servers
  - to implement the DNS protocol and traverse the domain names hierarchy
  - Difficult to do this on all endhosts

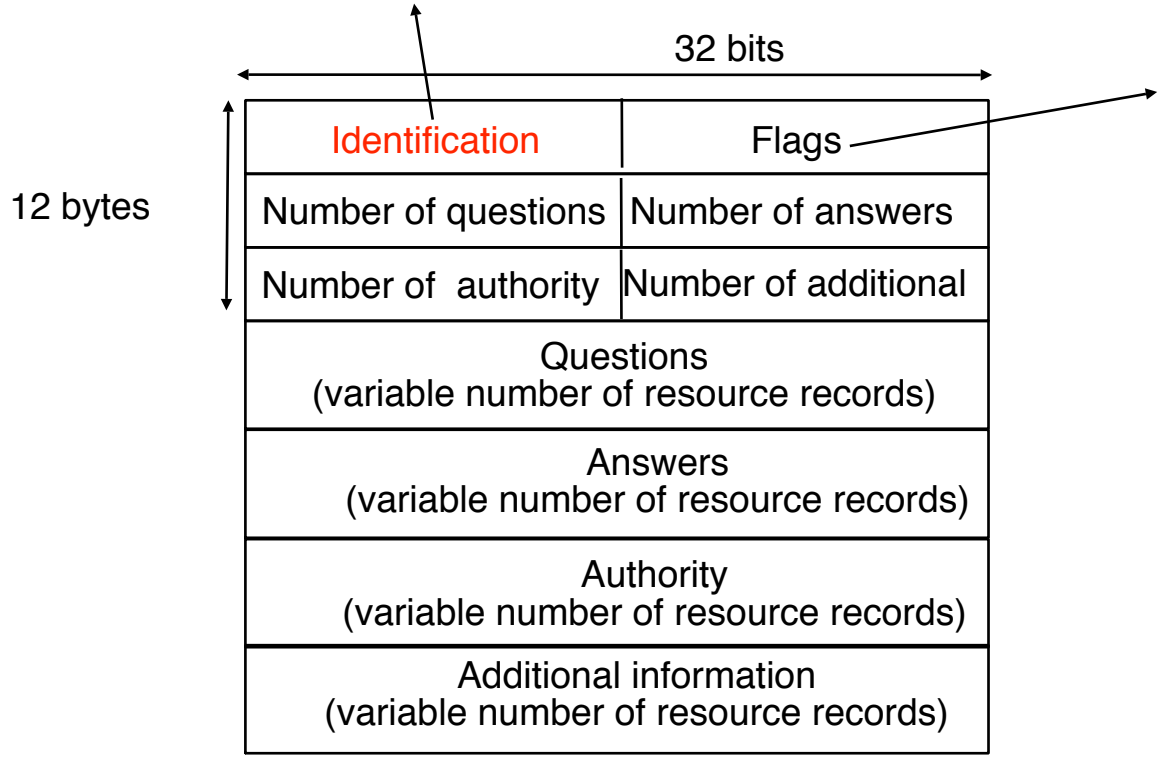
- Solution
  - DNS resolver
    - one resolver for a set of endhosts
    - maintains up-to-date list of IP addresses of root servers
    - implements DNS protocol
  - endhosts
    - only need to be able to send DNS requests to resolver
    - must know IP address of closest DNS resolvers

DNS : optimisations

- Reduce risk of failures
  - several root-servers
  - server DNS servers authoritative for each domain
  - each endhost can send queries to multiple resolvers
- Improved performance
  - avoid sending several times the same query
  - cache memory on DNS resolvers containing
    - recent name-addresses translations
    - addresses of DNS servers recently contacted
- DNS protocol
  - usually runs over UDP
  - sometimes is also used over TCP

DNS : message format

Each DNS request contains a number that will be returned in the response by the server to allow the client to match the request.



- Question/Response
- Recursive question or not
- Authoritative answer or not
- Possible error

DNS : resource records

- Each DNS messages is composed of resource records (RR) encoded as TLV
  - < Name, Value, Type, TTL>
  - Types de RR
    - A (Address)
      - Name is a hostname and Value an **IPv4 address**
    - AAAA (Address)
      - Name is a hostname and Value an **IPv6 address**
    - NS (NameServer)
      - Name is a domain name and Value is the hostname of the DNS server responsible for this domain
    - MX (Mail Exchange)
      - Name is a domain name and Value is the name of the SMTP server that must be contacted to send emails to this domain
    - Type CNAME
      - Alias

Lifetime of the RR in server's cache

Internet applications

- Contents
  - The client-server model
  - Name to address resolution
  - email
  - world wide web
  - peer-to-peer applications

Email

- Simplified model
  - Alice sends an email to Bob

Alice@a.net

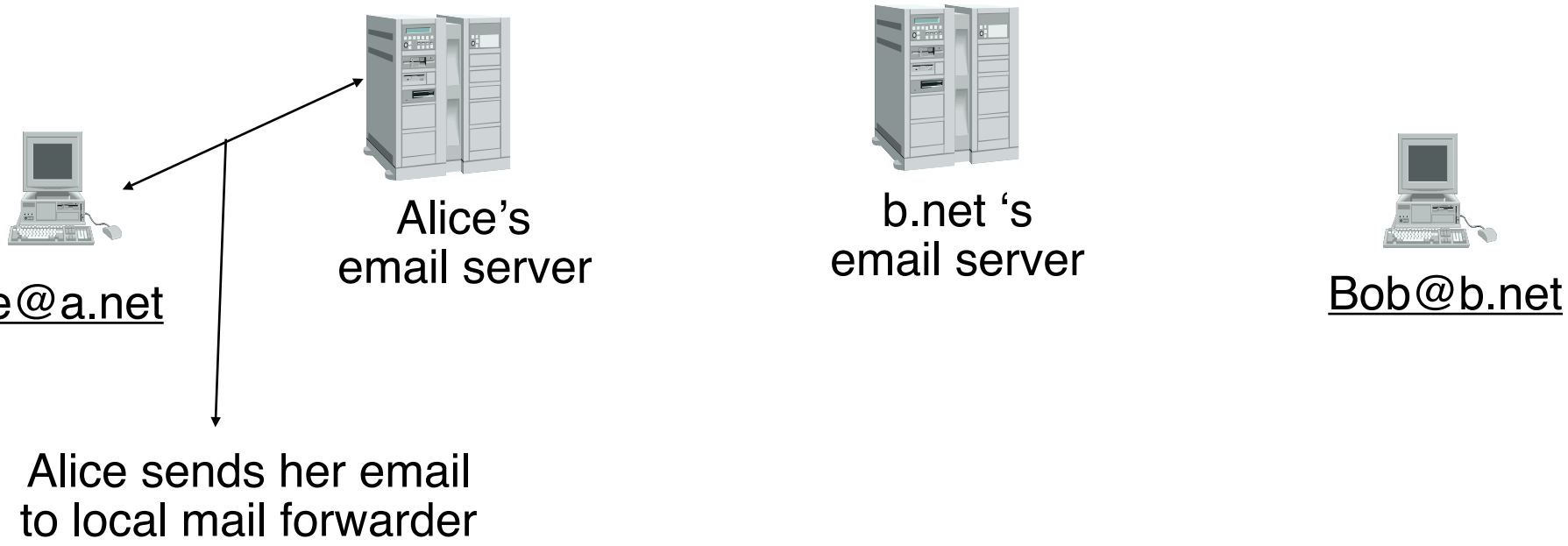
Alice's
email server

b.net 's
email server

Bob@b.net

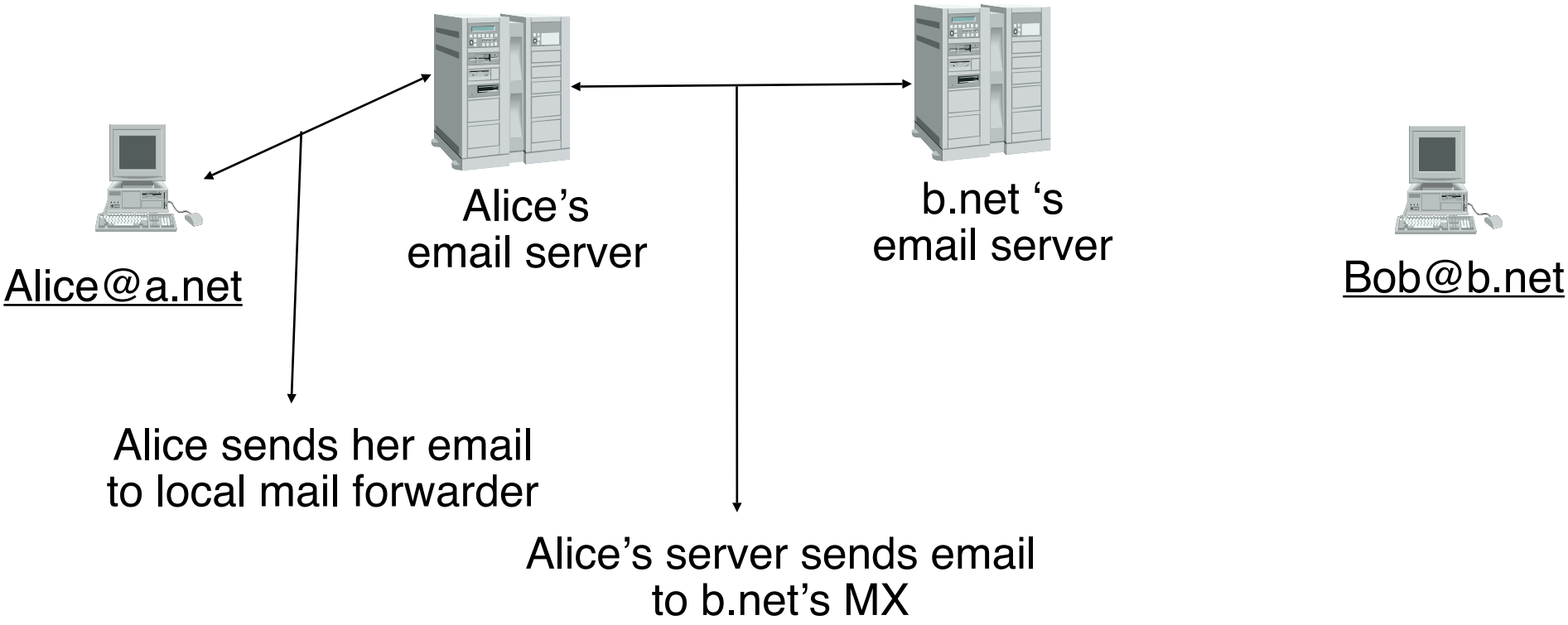
Email

- Simplified model
  - Alice sends an email to Bob



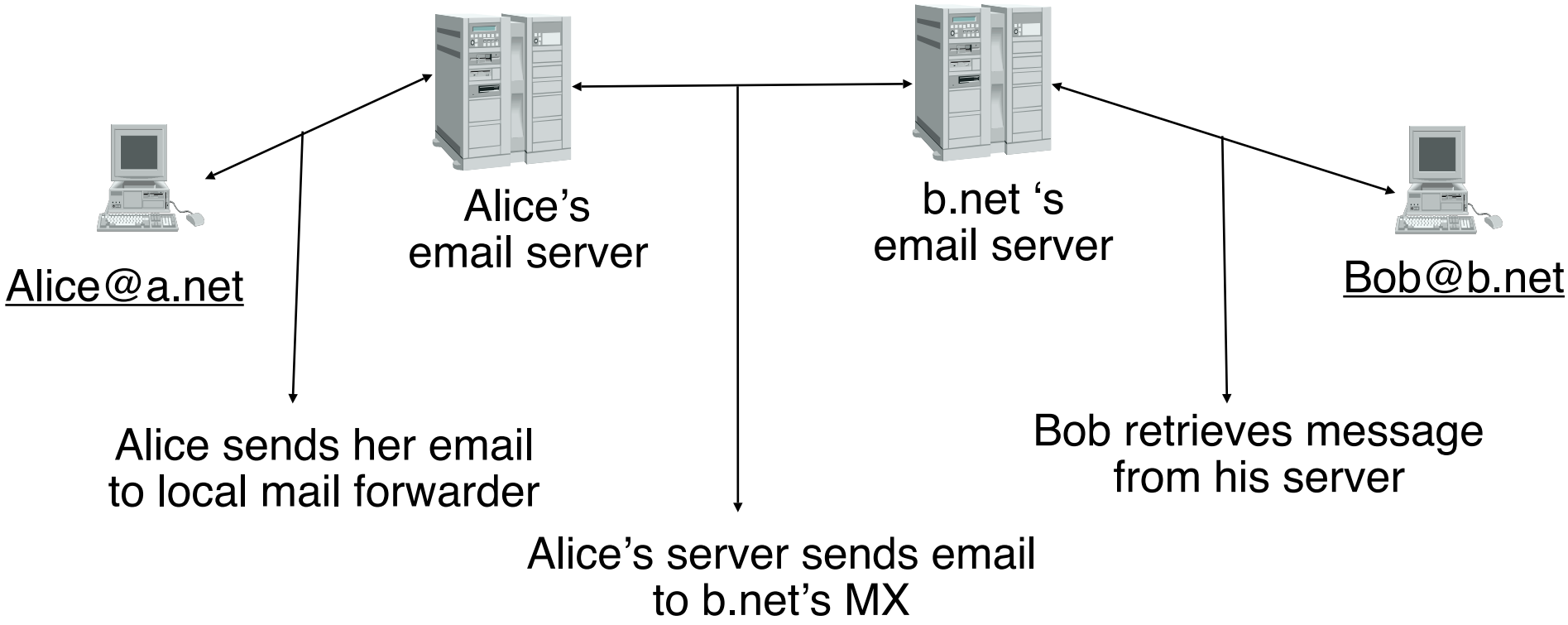
Email

- Simplified model
  - Alice sends an email to Bob

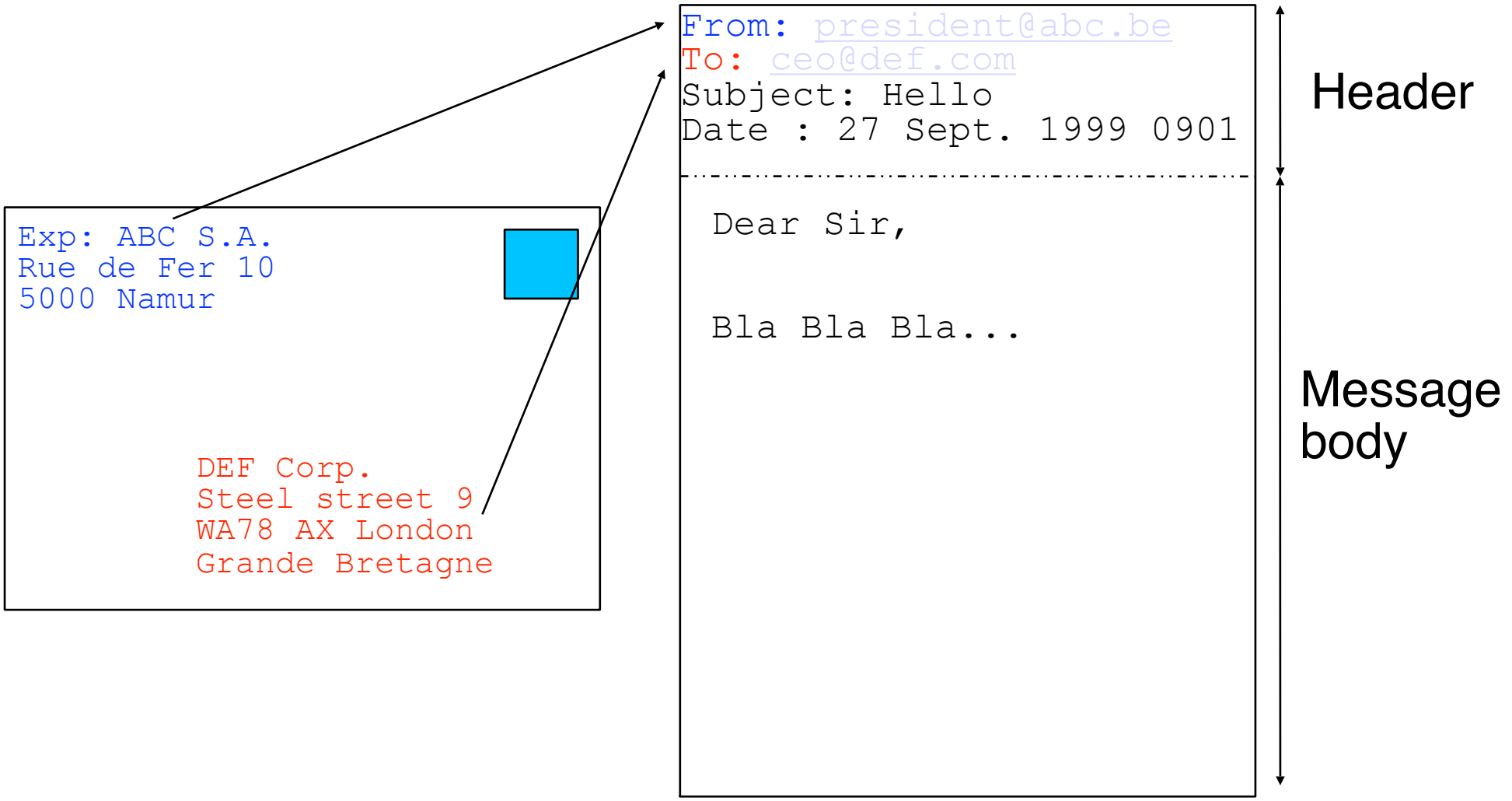


Email

- Simplified model
  - Alice sends an email to Bob



Email message format



Message format (2)

- Header format
  - Contains only US-ASCII (7bits) characters
  - At least three lines that end with <CRLF>
    - From: sender@domain
    - To:recipient@domain
    - Date: <creation date of message>
      - example : 26 Aug 199 1445 EDT
  - Optional fields
    - Subject: subject of message
    - cc: copy@domain
    - Message-ID: <number@domain>
    - Received: information on path followed by message
    - In-Reply-To: <message-ID>
  - Header ends with empty line (<CRLF>)

MIME

MIME

- Internet email was designed for US-ASCII
  - How to transmit more complex messages ?

MIME

- Internet email was designed for US-ASCII
  - How to transmit more complex messages ?
- Multipurpose Internet Mail Extensions
  - Improved email message format
  - Constraints
    - must remain compatible with old email servers
      - most of them only support US-ASCII and short lines
    - must support non-English text
      - character set must be beyond 7bits US-ASCII
    - must support various formats in a single message
      - message body, attachments, ...
    - must allow to transmit audio, video, ...
      - need to identify the type of content

MIME

- Internet email was designed for US-ASCII
  - How to transmit more complex messages ?
- Multipurpose Internet Mail Extensions
  - Improved email message format
  - Constraints
    - must remain compatible with old email servers
      - most of them only support US-ASCII and short lines
    - must support non-English text
      - character set must be beyond 7bits US-ASCII
    - must support various formats in a single message
      - message body, attachments, ...
    - must allow to transmit audio, video, ...
      - need to identify the type of content
- Solution
  - add new optional fields in header
  - add optional fields inside message body when

MIME (2)

- New header fields
  - MIME-Version:
    - version of MIME used to encode message
    - current version : 1.0
  - Content-Description:
    - comment describing the content of the message
  - Content-Type:
    - type of information inside message
  - Content-Transfer-Encoding:
    - how the message has been encoded
  - Content-Id:
    - unique identifier for the content

MIME: Content-Type

- Content-Type : type/encoding
  - type of content
    - text, image, video, application
    - multipart
  - encoding of content
    - text/plain , text/html
    - image/gif, image/jpeg
    - audio/basic
    - video/mpeg, video/quicktime
    - application/octet-stream, application/postscript
    - multipart/alternative
      - message contains several times the same information with different encodings
    - multipart/mixed
      - message contains several information of different types
        - example : text of message body and attachment

Character sets and content encoding

- How to support rich character sets ?
  - Content-Type: text/plain; charset=us-ascii
    - ASCII 7bits, default
  - Content-Type: text/plain; charset=iso-8859-1
    - Character set suitable for Western European languages, defined by ISO, 8 bits per character
    - Content-Type: text/plain; charset=unicode
    - Universal character set, defined by ISO, 16 bits per character
- How to encode non-text data ?
  - data must be encoded in US-ASCII 7 bits characters
  - Base64
  - uses ASCII characteres A...Z,a...z,0...9, "+" et "/"
    - A=0, B=1, C=2, ... +=62 et /=63
  - Each character is used to encode 6 bits
    - 24 bits from initial message -> 4 ASCII characters
  - Special character “=” used for padding

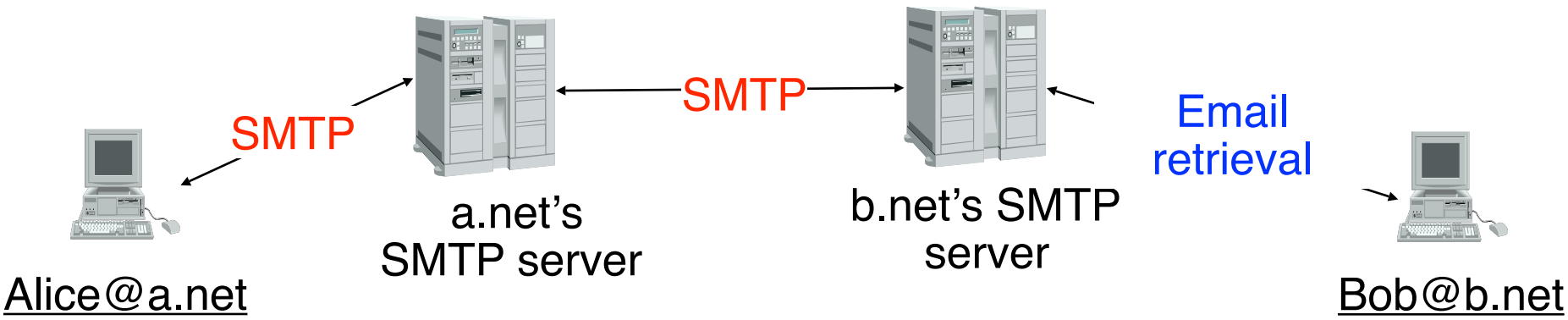
Multipart/mixed

- How to place different contents and encoding in a single message ?
  - We need a delimiter between the different content types placed inside message body

```
Date: Mon, 20 Sep 1999 16:33:16 +0200
From: Nathaniel Borenstein <nsb@bellcore.com>
To: Ned Freed <ned@innosoft.com>
Subject: Test
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="simple boundary"

preamble, to be ignored

--simple boundary
Content-Type: text/plain; charset=us-ascii

partie 1

--simple boundary
Content-Type: text/plain; charset=us-ascii

partie 2
--simple boundary
```

Email transmission

- SMTP : Simple Mail Transfer protocol
  - uses TCP service
  - Address of SMTP server
    - IP address of server + TCP + port number: 25
      - RR of type MX can be used to find the SMTP server responsible for a given domain



SMTP

- Client-server model
  - Server waits for email messages to relay/deliver
  - Client sends email messages through server
- Application-level protocol
  - client opens TCP connection
  - Client sends commands composed of
    - `command parameter <CRLF>`
      - HELO
      - MAIL FROM:
      - RCPT TO:
      - DATA
      - QUIT
  - Server answers with one-line replies
    - `numeric_code comment (text) <CRLF>`
      - 250 OK
      - 221 closing

SMTP (2)

- Three phases of SMTP
  1. Establishment of an SMTP association
     - TCP connection established upon request from client
     - Server greetings
     - HELO command from client
  2. Message transfer
     - MAIL FROM: <user@domaine>
     - RCPT TO: <user@domaine>
     - DATA
       - transmission of entire message including headers
       - one line containing only the dot "." characters marks end of message
     - Other subsequent messages can be transmitted after
  3. Release of the SMTP association
     - QUIT
     - Closing message from server
     - TCP connection is closed

Retrieval of email messages

- In the old days
  1. Destination is always connected to the Internet
     - email addresses are username@hostname
     - When an email arrives, it is stored in a file that belongs to the user, e.g. `/var/mail` on Unix

- Today
  - Most networks have one or a few SMTP servers used to receive emails, but also detect spam, viruses, ...
  - Endusers retrieve their emails from this server
    - Post Office Protocol (POP)
    - Internet Mail Access Protocol (IMAP)
    - Webmail

POP

- Goal
  - Allow authenticated users to retrieve email messages from server
- Operation
  - POP uses TCP service
  - Address of POP server
    - Host address + TCP + port number : 110
  - Client send commands
    - command : one ASCII line ending with <CRLF>
    - `USER, PASS, STAT, RETR, DELE, QUIT`
  - server replies with
    - `+OK` if command was successful
      - email messages follow some +OK replies
    - `-ERR` in case of errors

POP (2)

- Three phases of the protocol
  1. Authorisation : checking the user credentials
     - USER <username>
     - PASS <password>
  2. Transaction
     - retrieval and removal of messages
     - STAT
       - list headers of stored messages
     - RETR <n>
       - retrieval of the nth message
     - DELE <n>
       - the nth message is marked for deletion
  3. Update
     - End of the retrieval phase
       - Messages marked for deletion are removed from server
       - TCP connection is closed

Internet applications

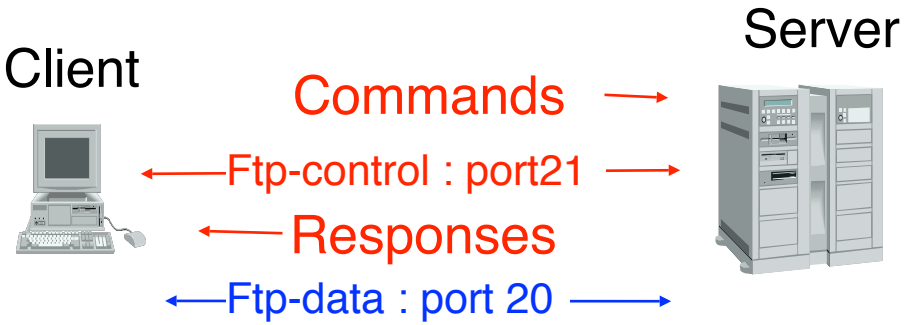
- Contents
  - The client-server model
  - Name to address resolution
  - email
  - world wide web
  - peer-to-peer applications

FTP : File Transfer Protocol

- Protocol from the old days
  - allows a client to send/retrieve files from a server
- Problems solved by FTP
  - User authentication
    - username, password
  - Filesystem traversal
    - browse directories on server and locate files
  - file transfer
    - to or from the server

FTP : File Transfer Protocol

- Protocol from the old days
  - allows a client to send/retrieve files from a server
- Problems solved by FTP
  - User authentication
    - username, password
  - Filesystem traversal
    - browse directories on server and locate files
  - file transfer
    - to or from the server

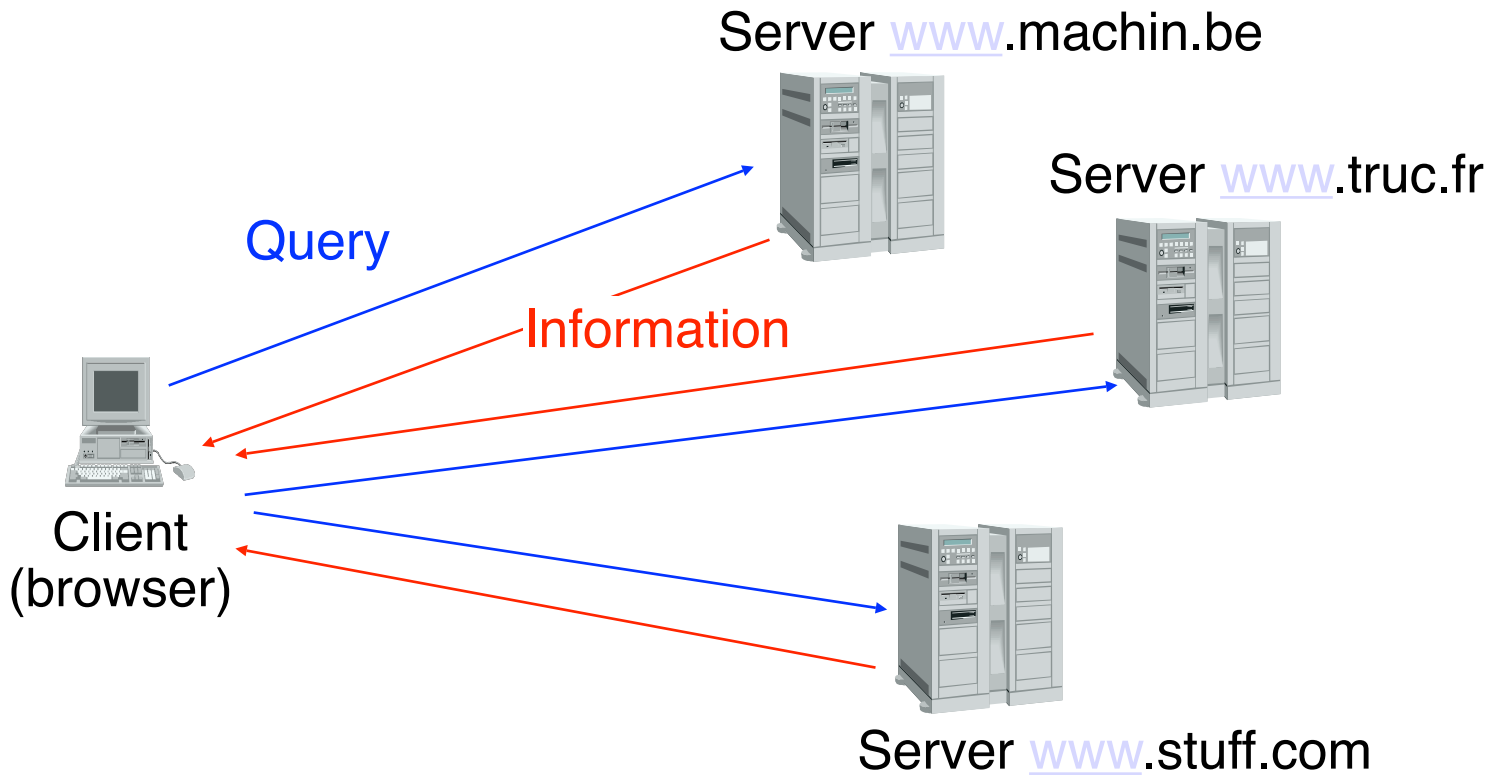


FTP : main commands

- Main commands
  - USER <user>
    - username, ftp for anonymous access
  - PASS <pass>
    - allows user to send password associated to username
  - SYST
    - information about type of server
  - CWD <path>
    - directory traversal
  - STOR <file>
    - save file in the current directory on server
  - RETR <file>
    - retrieve file from current directory on server
  - PORT <B1,B2,B3,B4,B5,B6>
    - use TCP connection on port B5*256+B6 on B1.B2.B3.B4

World Wide Web

- Goals
  - Allow browsers to browse hypertext documents stored on multiple servers



World Wide Web (2)

- The five key elements of www

1. An addressing scheme that allows to identify any document stored on a server
   - URL : Uniform Resource Locator
2. An hypertext language that allows to easily write documents with hypertext links
   - HTML : HyperText Markup Language
3. An efficient and lightweight application-level protocol to exchange documents
   - HTTP : HyperText Transfer Protocol
4. Servers
5. Clients (browsers)

Uniform Resource Locator (URL)

Uniform Resource Locator (URL)

- generic syntax : <protocol>://<document>
  - protocol used to retrieve document from server
    - http is the most common one but others are frequently used
  - document indicates the server and the location of the document
  - *<user>:<password>*@<server>*:<port>*/<path>
    - <user> : optional username
    - <password> : optional password
    - <machine> : hostname or IP address of the server that hosts the document
    - <port> : optional port number
    - <path> : document location on server
  - examples
    - http://www.info.ucl.ac.be
    - http://alice:secret@inl.info.ucl.ac.be:80/index.html

HTML

- HyperText Markup Langage
  - Language used to encode documents on the web
  - Keywords
    - <HTML>...</HTML>
    - <HEAD>...</HEAD>
    - <BODY>...</BODY>
    - <TITLE>..</TITLE>
    - <B>...</B>
    - <I>...</I>
    - <H1>...</H1>
    - <P>
    - <HR>
    - <UL>...</UL>
    - <OL>...</OL>
    - <IMG SRC="URL">
    - <A HREF="URL">*text anchor*</A>

HTML (2)

- Example

```
<HTML>

<HEAD>
<TITLE>HTML test page</TITLE>
</HEAD>

<BODY>
<IMG SRC="http://www.images.be/logo.gif">
<H1>Web servers from UCL UCL<P></H1>
<HR>
<UL>
 <LI><A HREF="http://www.uclouvain.be">UCL</A>
 <LI><A HREF="http://www.info.ucl.ac.be">CSE Dept.</A>
 <LI><A HREF="http://www.math.ucl.ac.be">Math</A>
</UL>
</BODY>

</HTML>
```

Header

Body

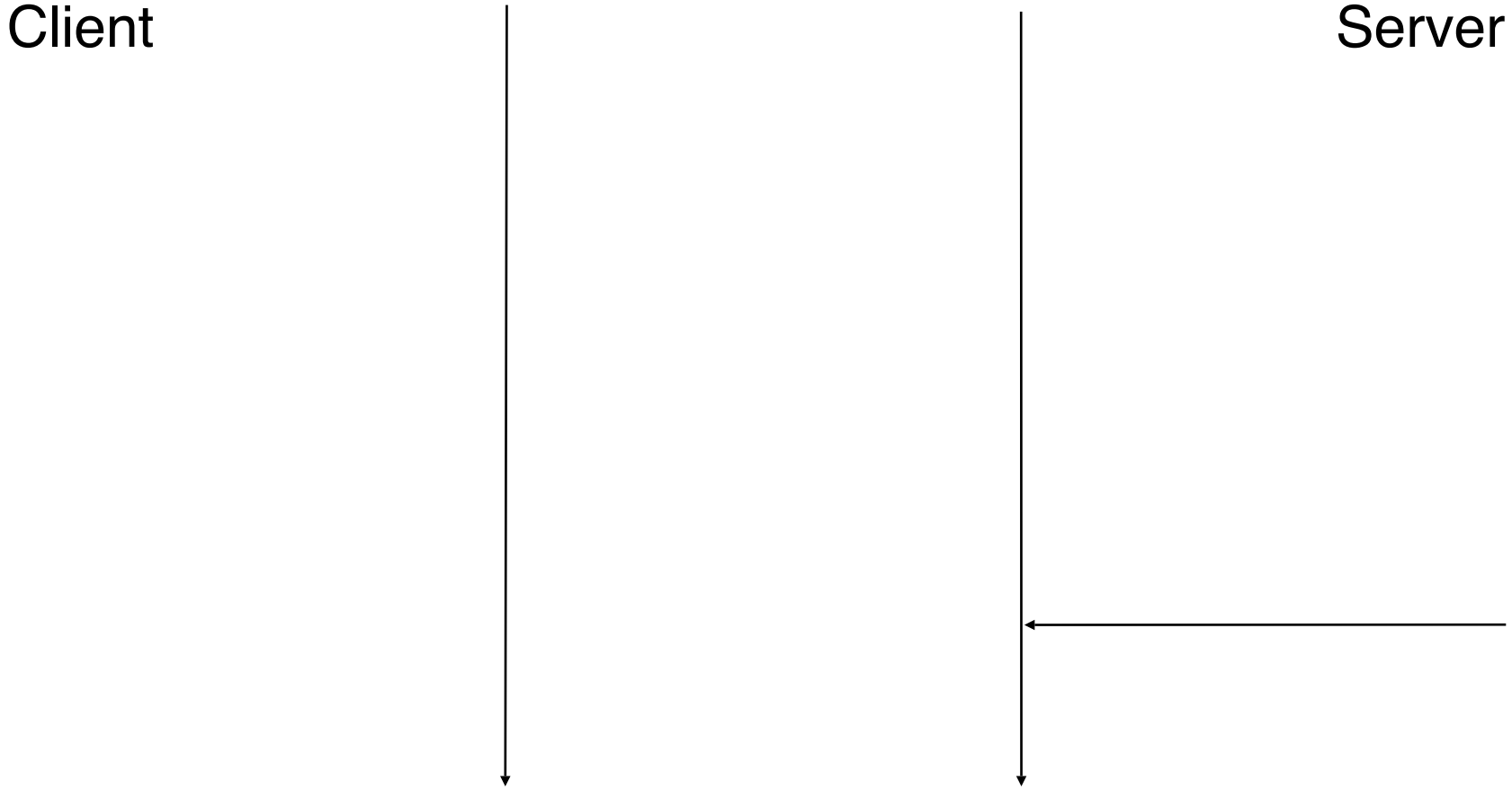
Image on remote server

First level title

External hypertext link

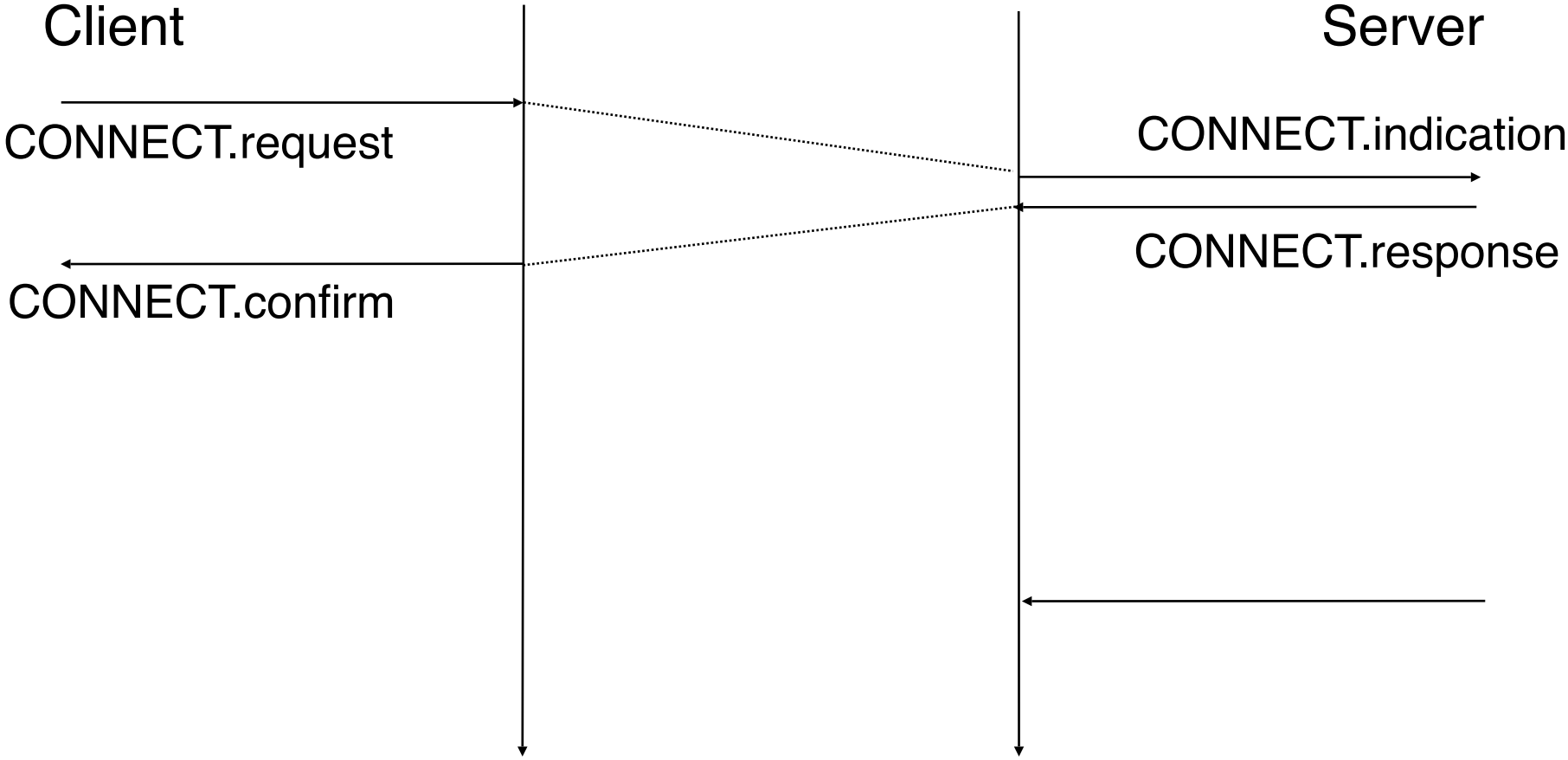
Information transfer www

- HTTP 1.0 - non-persistent connection
  - Principle
    - relies on TPC service (default port : 80)
    - Client sends request, server sends reply



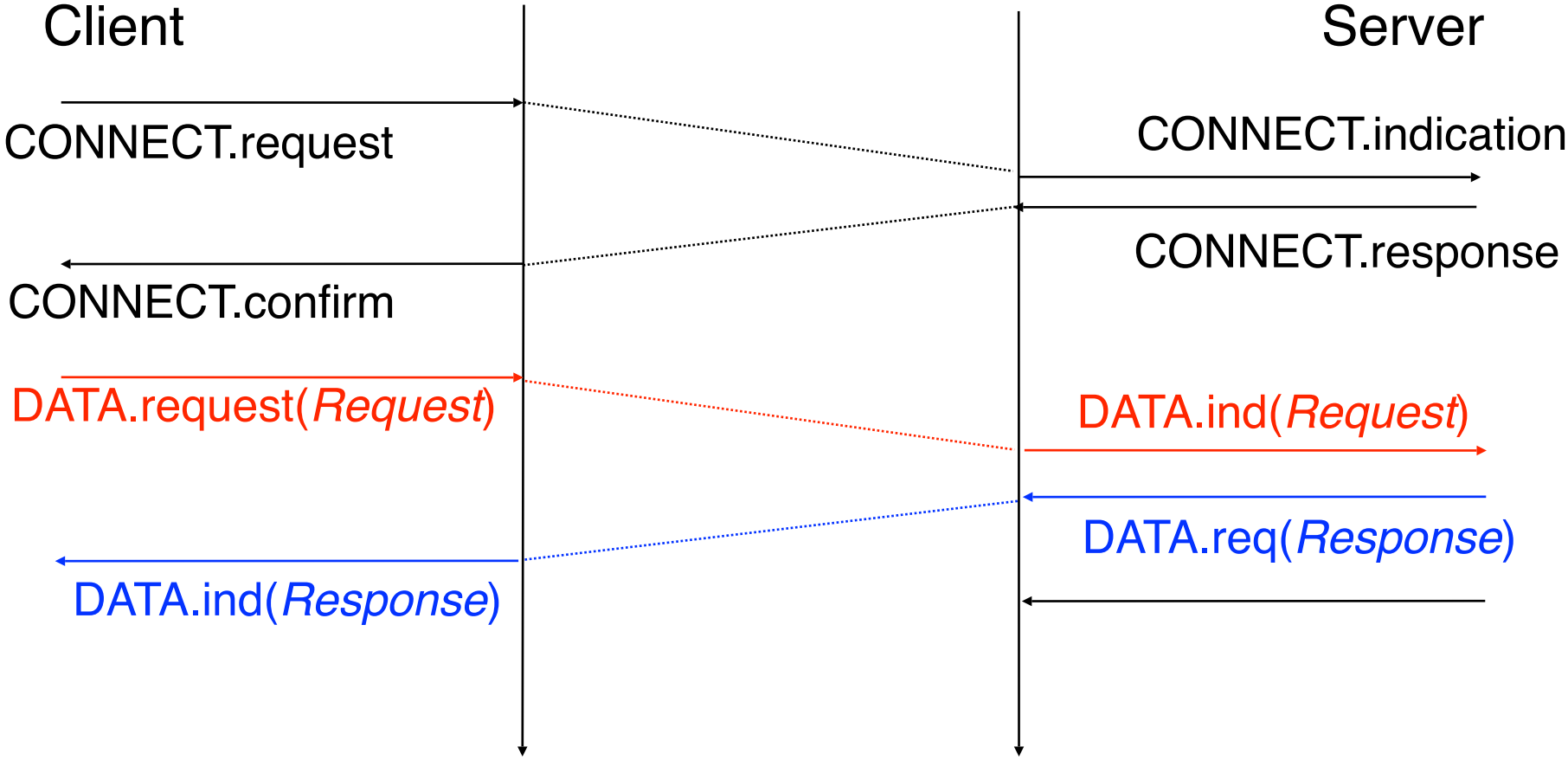
Information transfer www

- HTTP 1.0 - non-persistent connection
  - Principle
    - relies on TPC service (default port : 80)
    - Client sends request, server sends reply



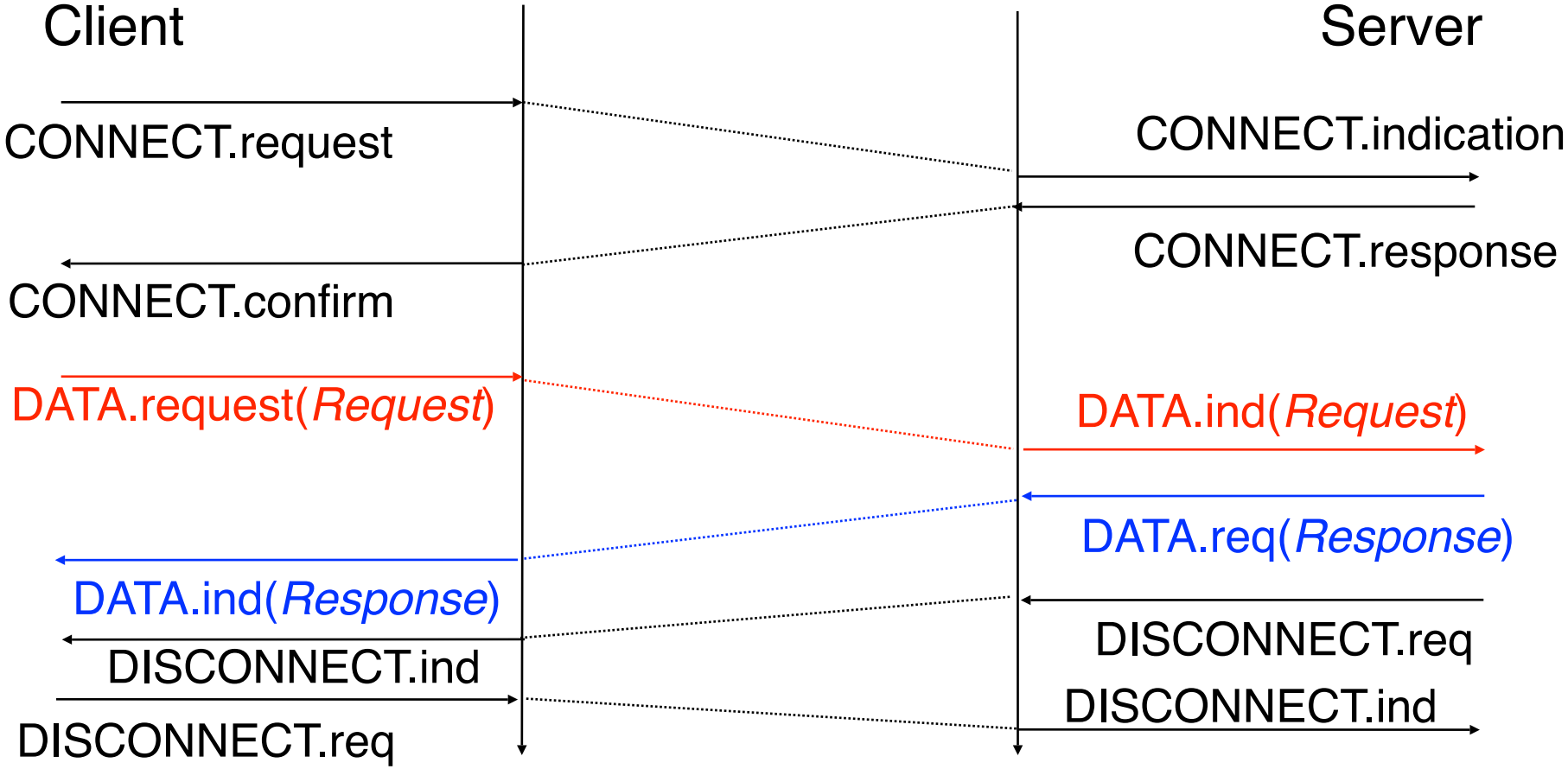
Information transfer www

- HTTP 1.0 - non-persistent connection
  - Principle
    - relies on TPC service (default port : 80)
    - Client sends request, server sends reply



Information transfer www

- HTTP 1.0 - non-persistent connection
  - Principle
    - relies on TPC service (default port : 80)
    - Client sends request, server sends reply



HTTP

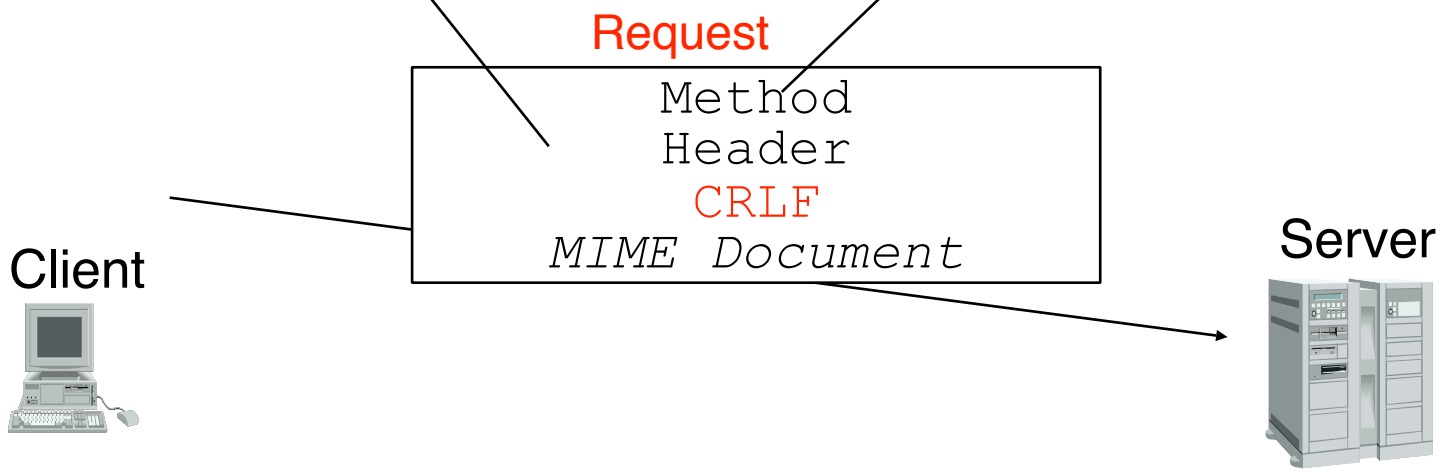
Client

Server

HTTP

Header contains additional information about request sent by client

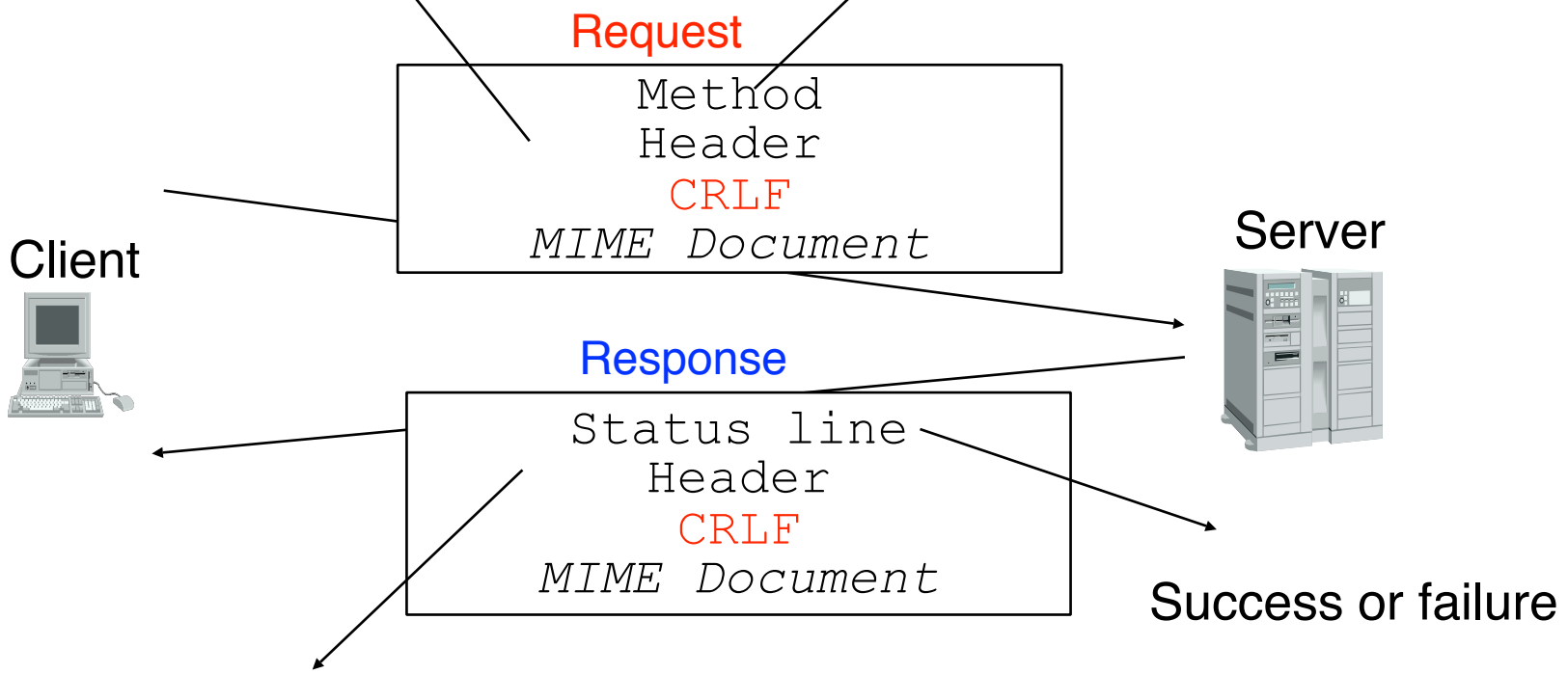
Method
GET
POST
...



HTTP

Header contains additional information about request sent by client

Method
GET
POST
...

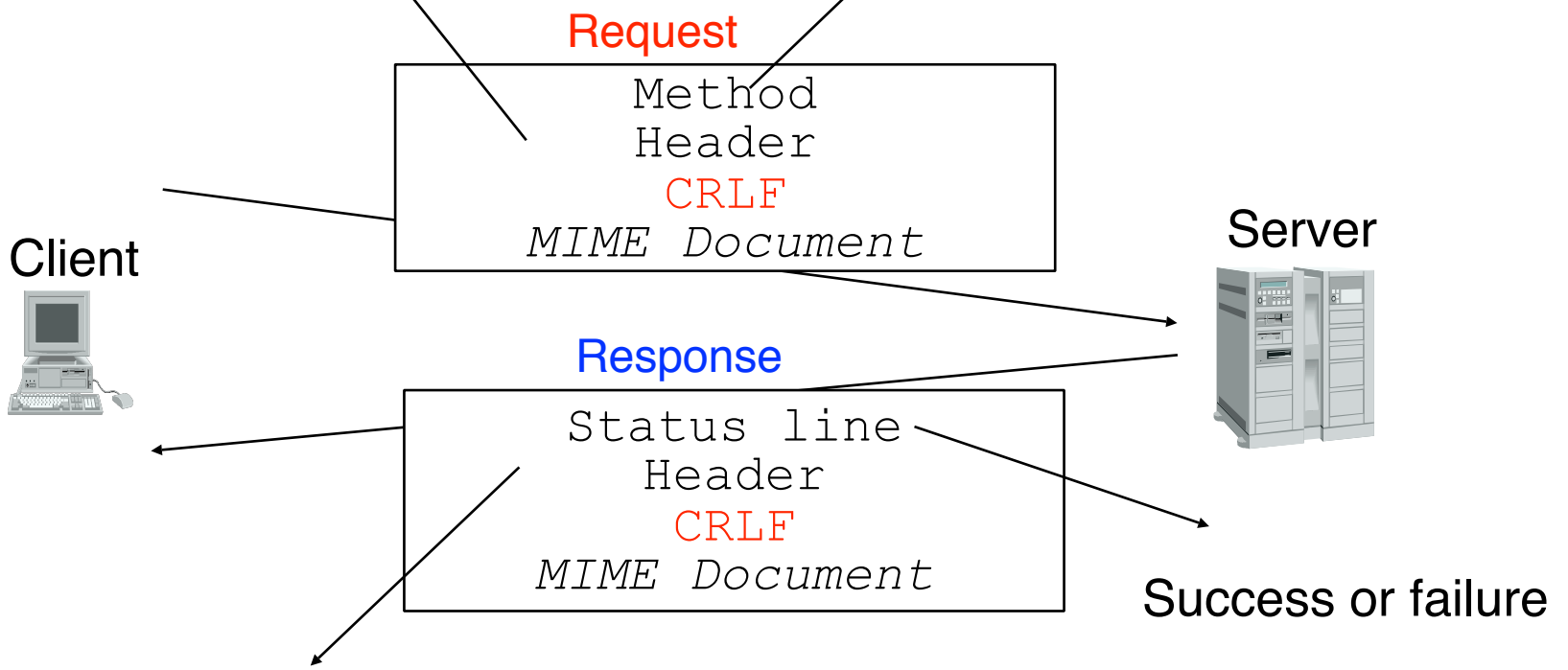


Header contains information about server and optional parameters specific to response

HTTP

Header contains additional information about request sent by client

Method
GET
POST
...



Header contains information about server and optional parameters specific to response

HTTP is a stateless protocol, server does not maintain any state from one request to another

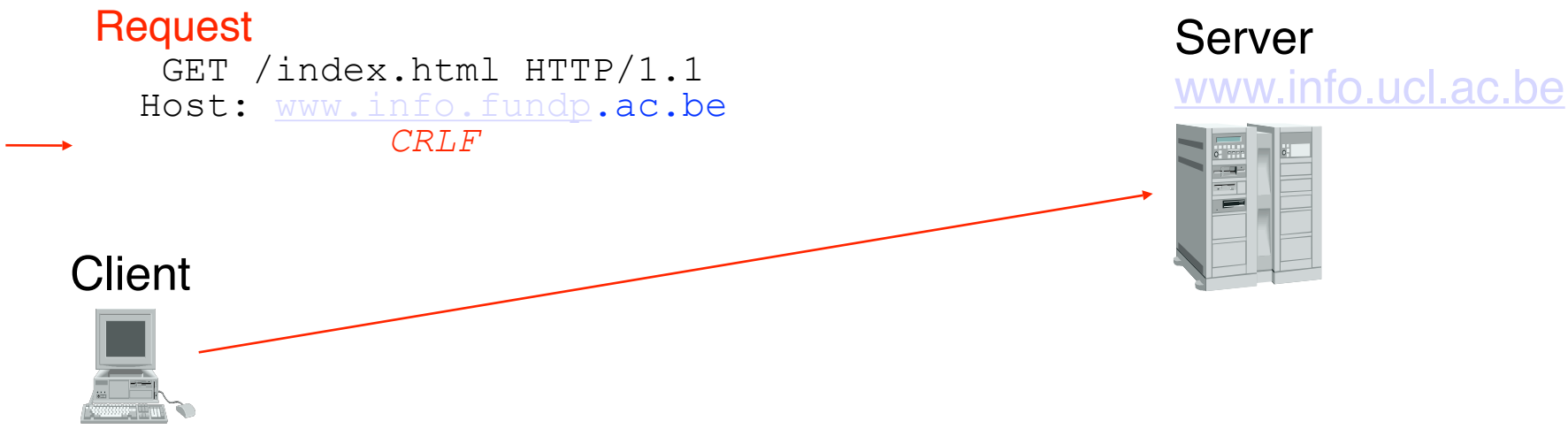
- POP, FTP, SMTP are examples of stateful protocols in contrast

HTTP : Example

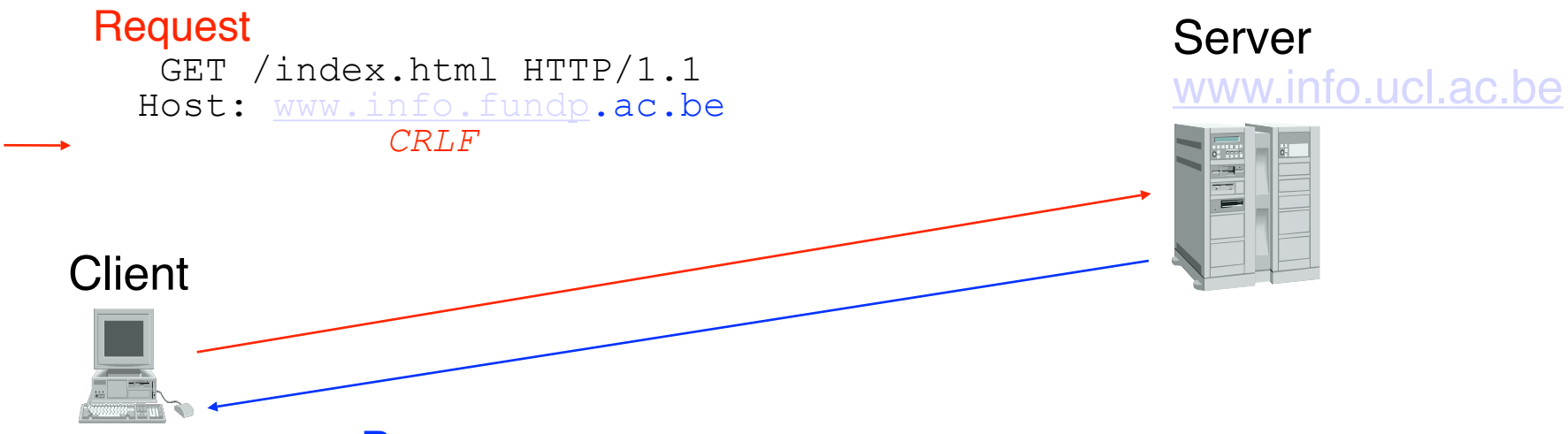
Server
www.info.ucl.ac.be

Client

HTTP : Example



HTTP : Example



HTTP : Methods

- Methods
  - GET
    - method used to request a "document" stored on server
      - GET <document> HTTP/1.0
    - example
      - `GET /index.html HTTP/1.0`
  - POST
    - method used to send a "document" to a server
      - document is part of the request and encoded as a MIME document

HTTP : Request headers

- Request headers
  - Allow to add information about the client or the request
  - `Host: <name>`
    - Name of the server where the document is stored
  - `Authorization`
    - allows to perform access control
  - `If-Modified-Since: <date>`
    - server will only send the requested document if the document is more recent than date
  - `Referer: <url>`
    - Information, indicates the URL visited by the client before this request
  - `User-Agent: <agent>`
    - information, indicates the browser used on the client

HTTP : Status line

- Status liine
  - Format : `Version_HTTP Code Comment`
  - Success/Failure
    - `1xx` : For information (unused)
    - `2xx` : Success
      - Example : `HTTP/1.0 200 OK`
    - `3xx` : Redirection
      - Request could not be handled on local server and should be sent to another server
      - Example :
        - `HTTP/1.0 301 Moved permanently`
        - attached MIME document will contain URL of document
    - `4xx` : Client-side error
      - examples
        - syntax error, unreachable URL, unauthorised, ...
    - `5xx` : Server-side error
      - examples :
        - internal error, method not implemented on server, ...

HTTP : Response headers

- Header
  - Optional information about the server, the response or the document attached to the response
  - Date
    - date of the document attached to response
    - example : `Date: Wed, 05 Sep 2001 13:27:34 GMT`
  - Server
    - Name and version of http server used
    - example :
      `Server: Apache/1.3.20 (Unix)ApacheJServ/1.1.2 PHP/4.0.6`
  - Content-*
    - MIME header of the attached document
    - example :

```
Content-Length: 5891
Content-Type: text/html
```

HTTP 1.1

HTTP 1.1

- HTTP 1.0
  - a single TCP connection used to transmit a single document (html file, image,...)
    - the establishment and release of the TCP connection induce a significant overhead, in particular for small pages

HTTP 1.1

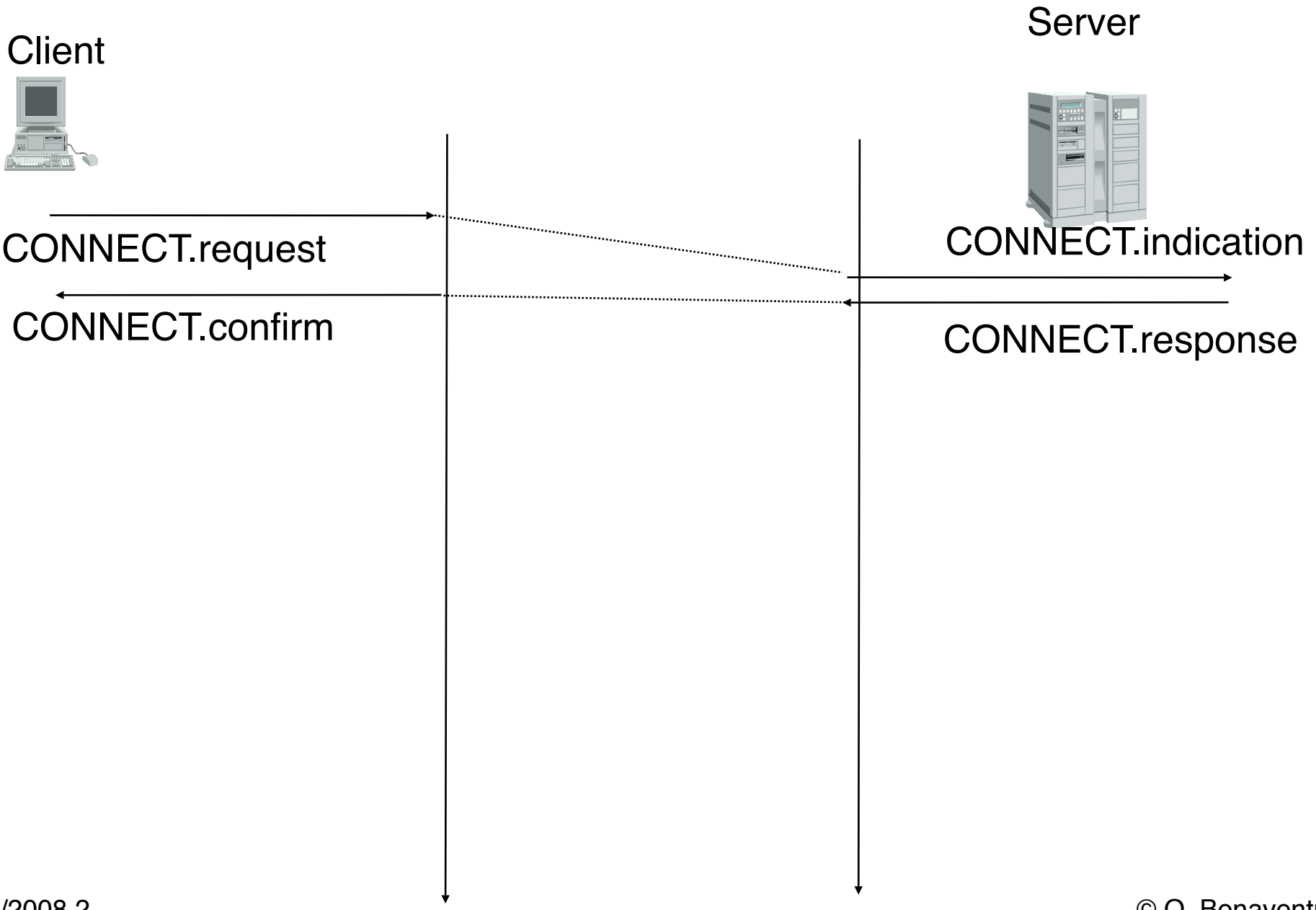
- HTTP 1.0
  - a single TCP connection used to transmit a single document (html file, image,...)
    - the establishment and release of the TCP connection induce a significant overhead, in particular for small pages
- HTTP 1.1
  - uses a single persistent TCP connection
    - This TPC connection can be used for several requests and the corresponding responses
    - the cost of establishing and releasing the TCP connection is amortised over multiple requests
    - Although HTTP 1.1 uses a single TCP connection for multiple requests, HTTP 1.1 remains stateless

HTTP 1.1 : Persistent connection

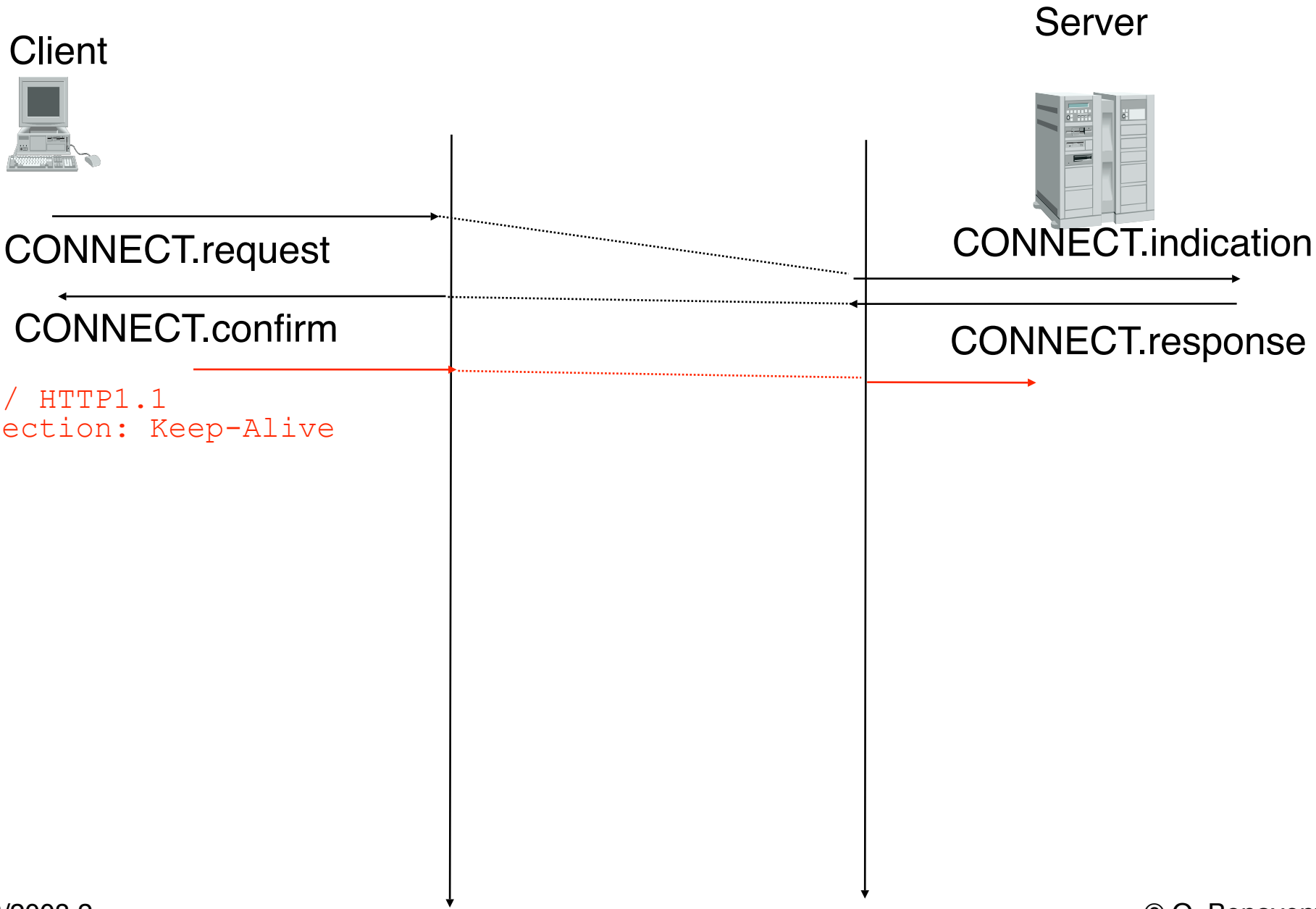
Client

Server

HTTP 1.1 : Persistent connection

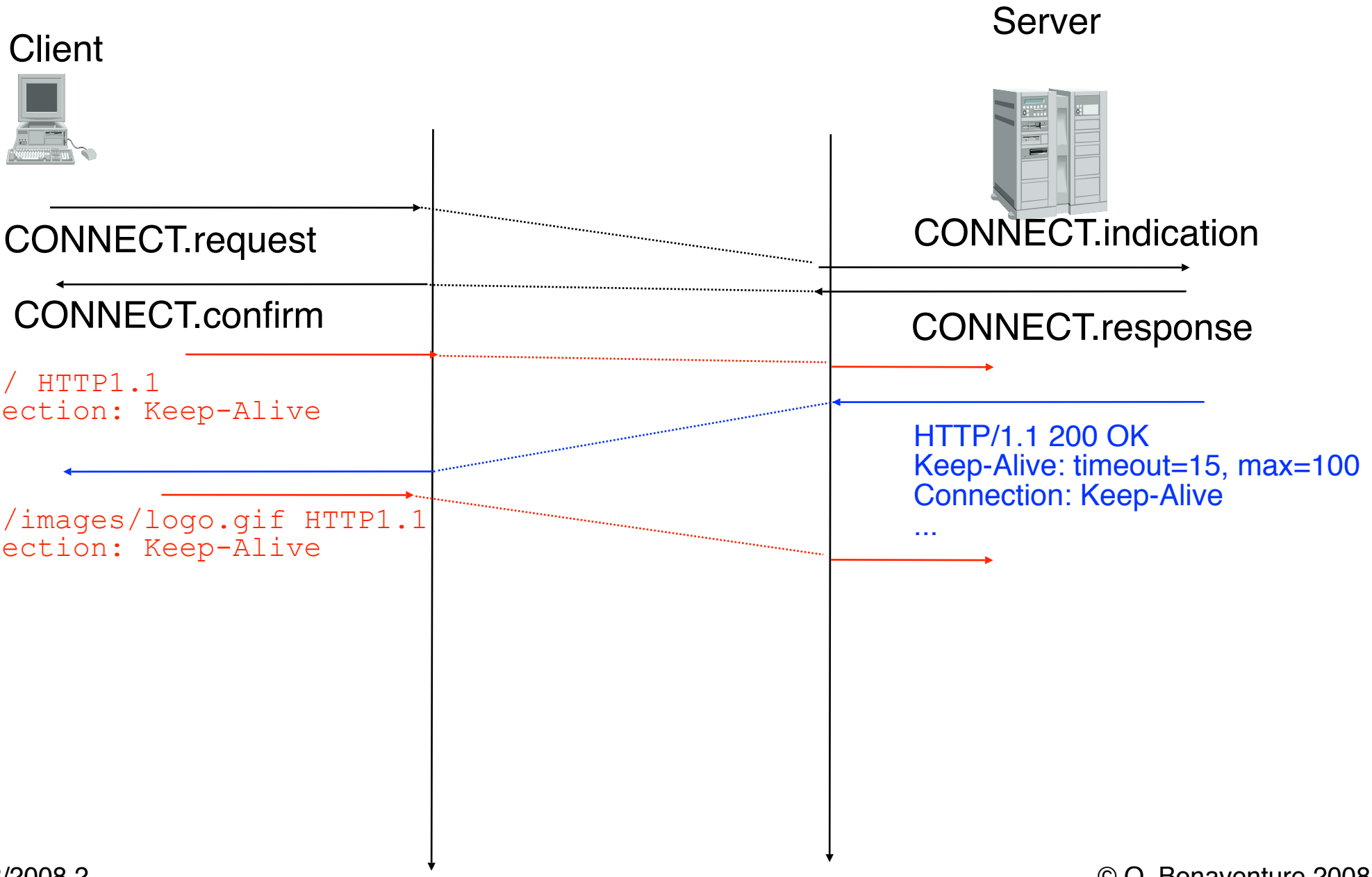


HTTP 1.1 : Persistent connection



HTTP 1.1 : Persistent connection

HTTP 1.1 : Persistent connection



HTTP 1.1 : Persistent connection

Client

Server

CONNECT.request

CONNECT.indication

CONNECT.confirm

CONNECT.response

```
GET / HTTP1.1
Connection: Keep-Alive
...
```

HTTP/1.1 200 OK
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
...

```
GET /images/logo.gif HTTP1.1
Connection: Keep-Alive
...
```

HTTP/1.1 200 OK
Keep-Alive: timeout=15, max=99
Connection: Keep-Alive
...

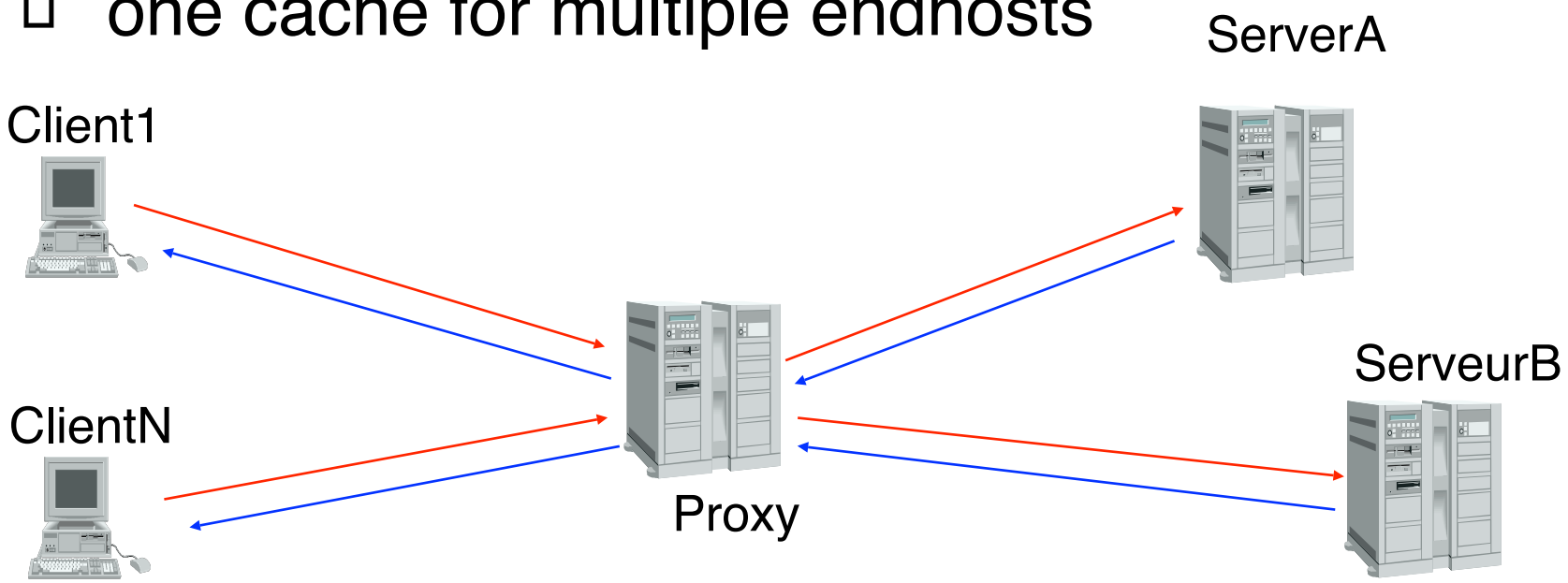
HTTP 1.1 : Persistent connection

Improving performance

- Observation
  - Many pages are requested multiple times or from close endhosts
- Solution
  - local cache on each client
    - `if-modified-since` header helps
  - one cache for multiple endhosts

Improving performance

- Observation
  - Many pages are requested multiple times or from close endhosts
- Solution
  - local cache on each client
    - `if-modified-since` header helps
  - one cache for multiple endhosts

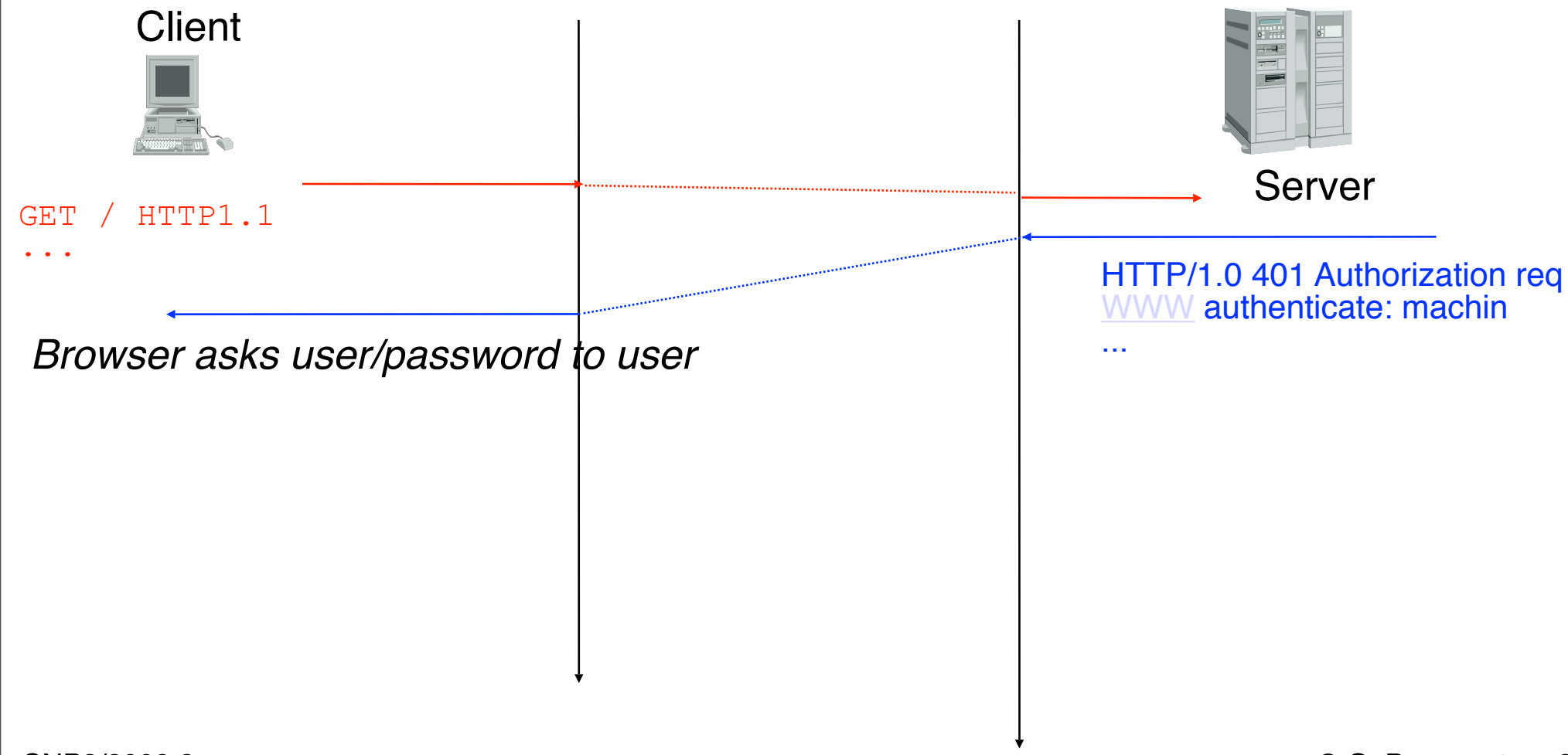


HTTP Authentication

- Example

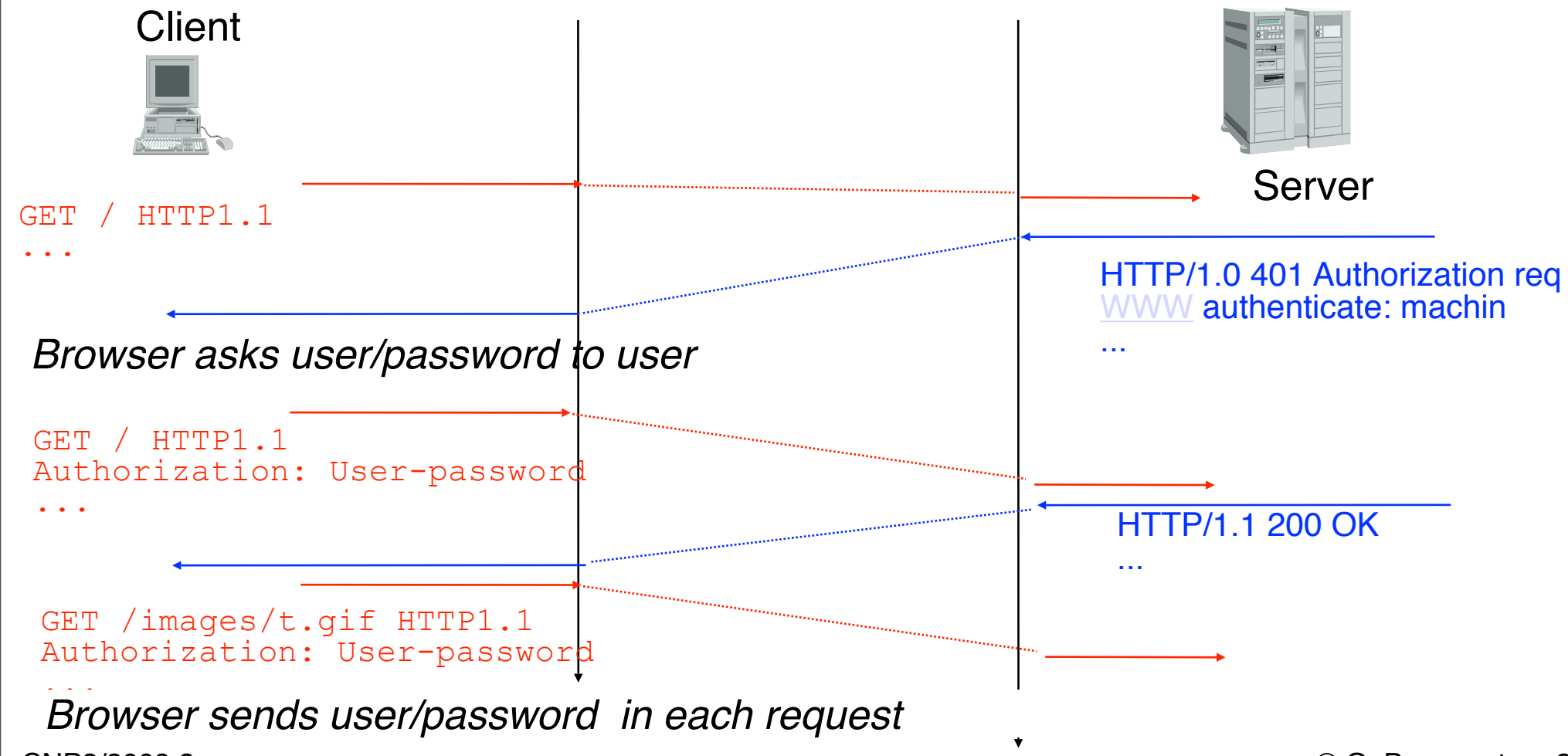
HTTP Authentication

- Example



HTTP Authentication

- Example



HTTP Cookies

- Example

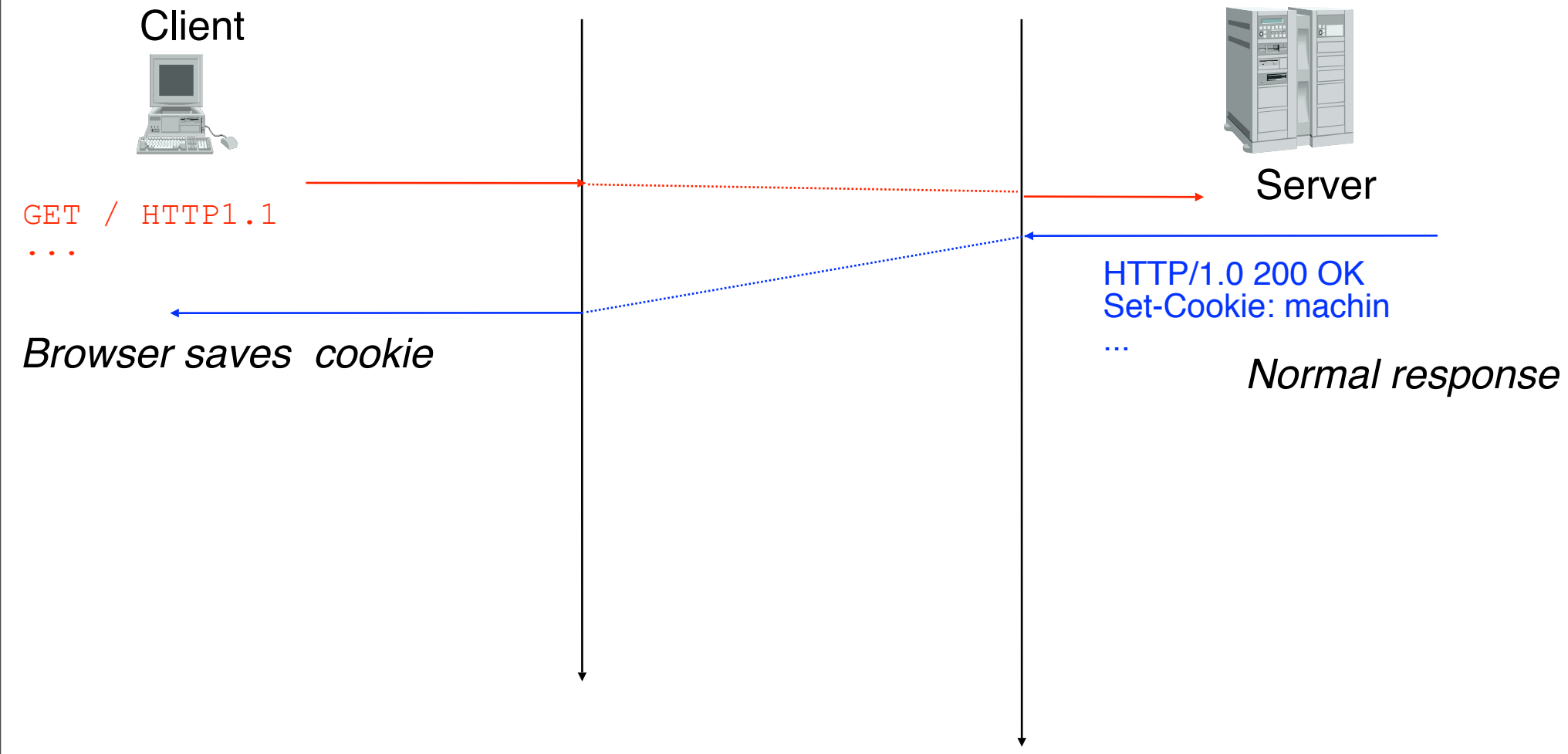
Client

Server

Normal response

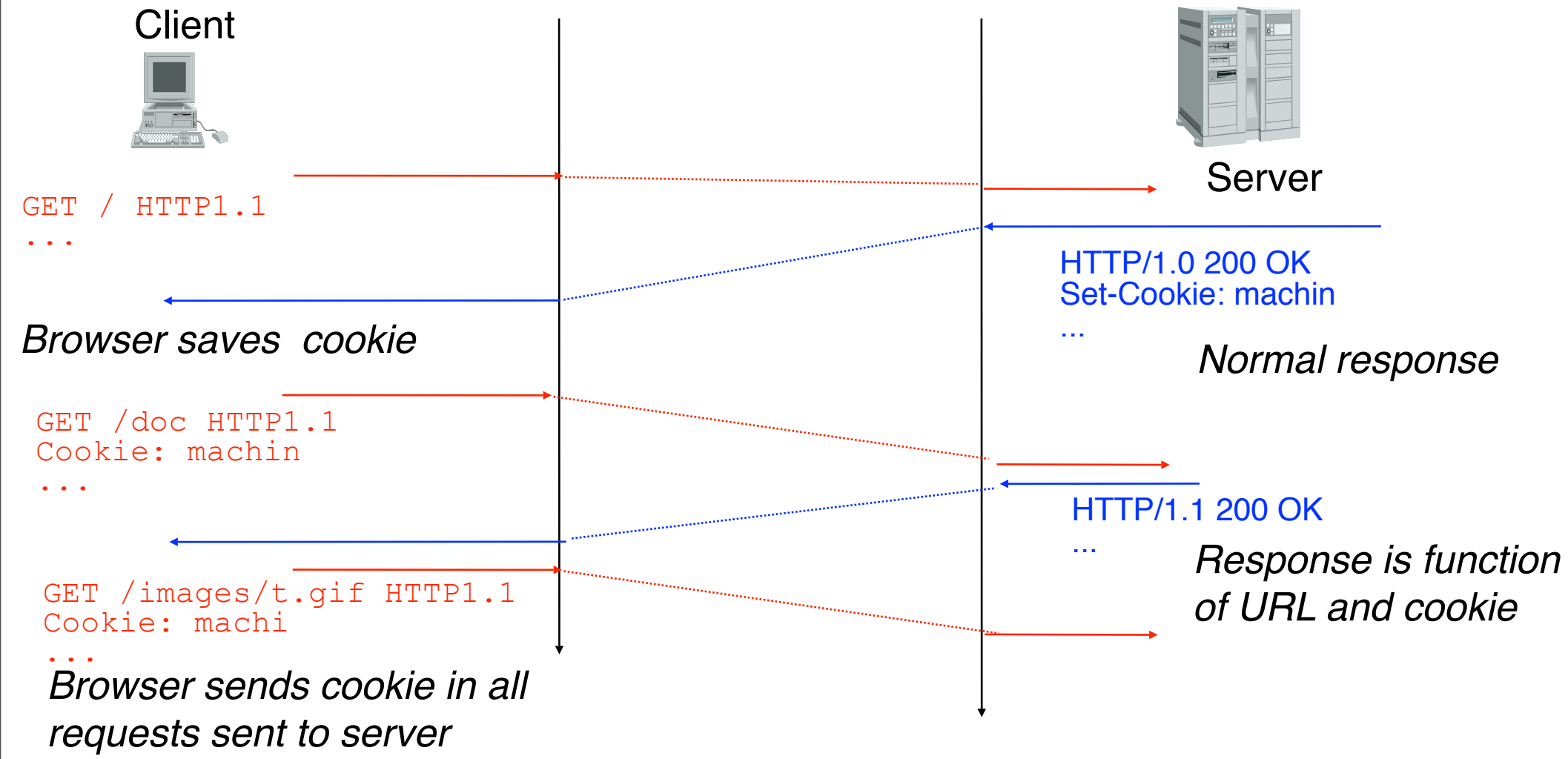
HTTP Cookies

- Example



HTTP Cookies

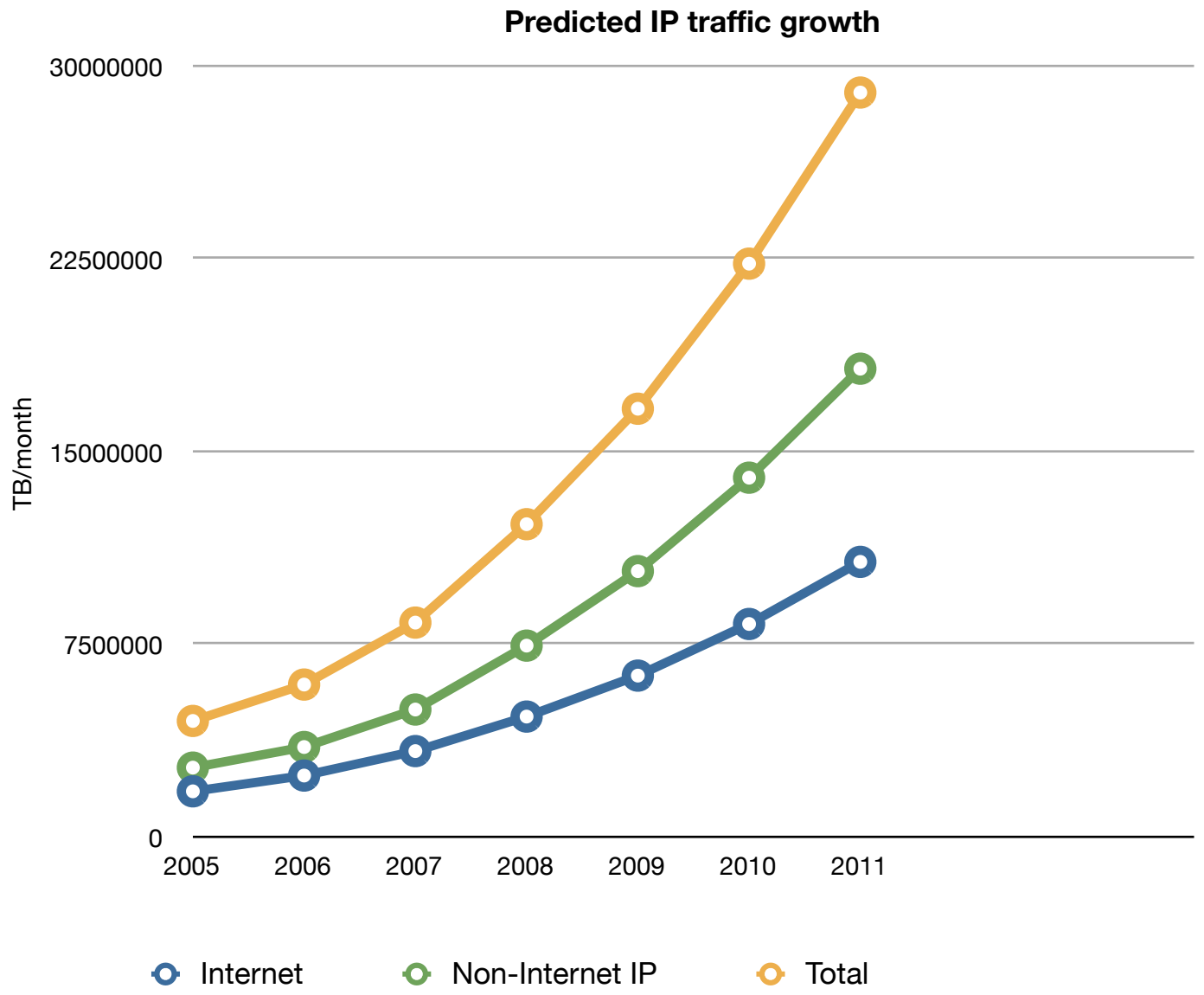
- Example



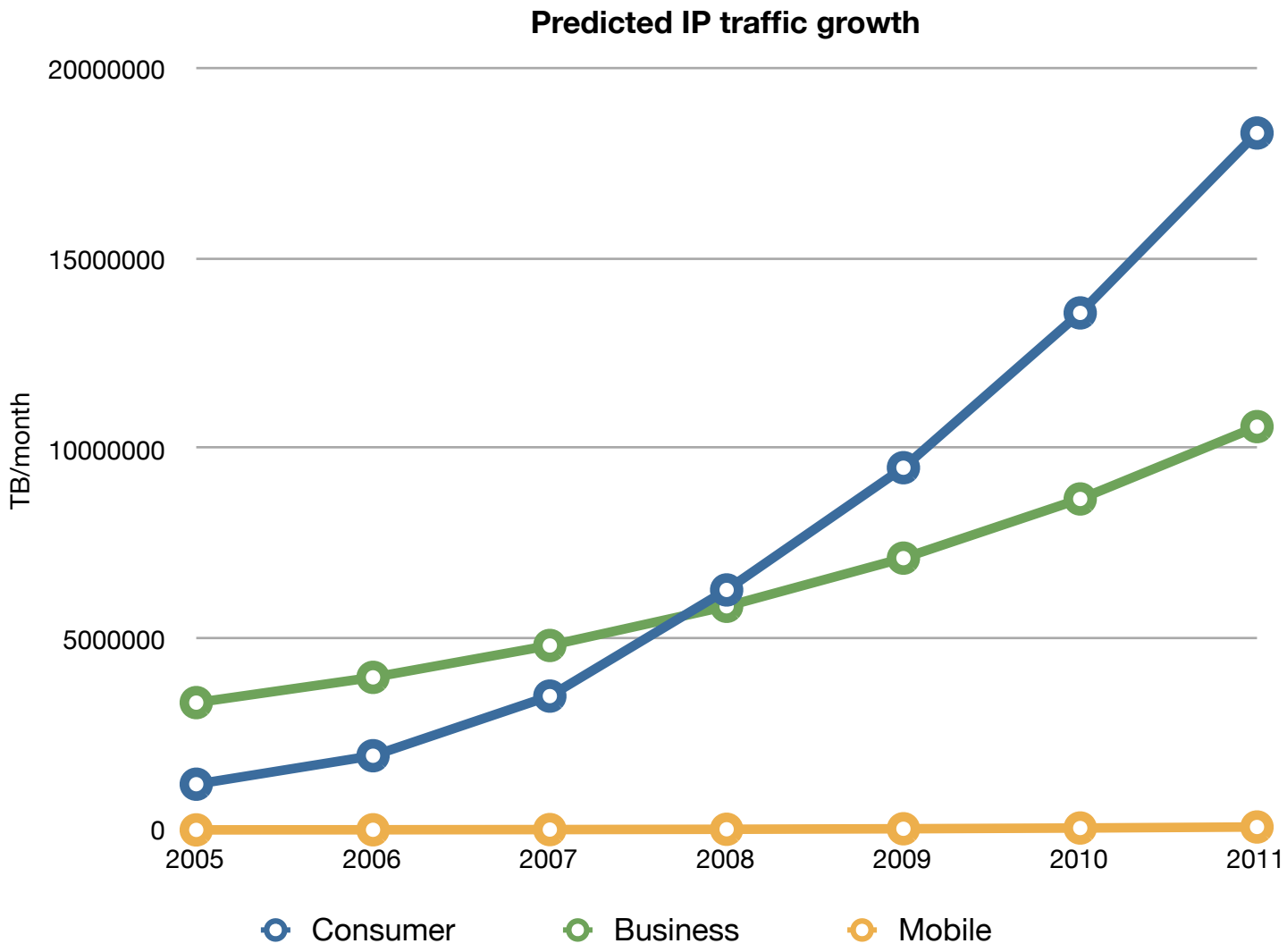
Internet applications

- Contents
  - The client-server model
  - Name to address resolution
  - email
  - world wide web
  - peer-to-peer applications

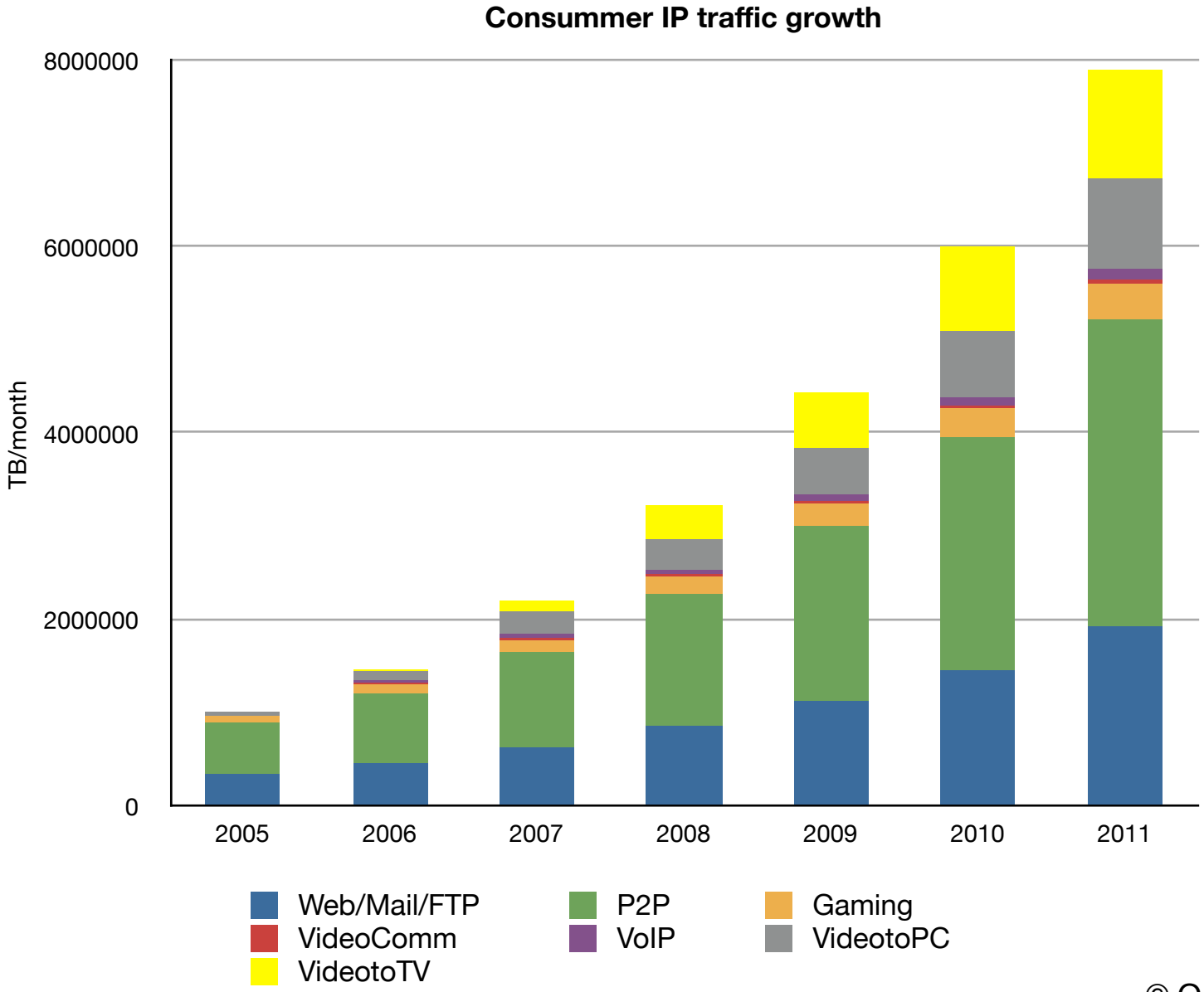
Predicted IP traffic growth



Predicted IP traffic growth



Predicted IP traffic growth (3)



Peer-to-peer file sharing

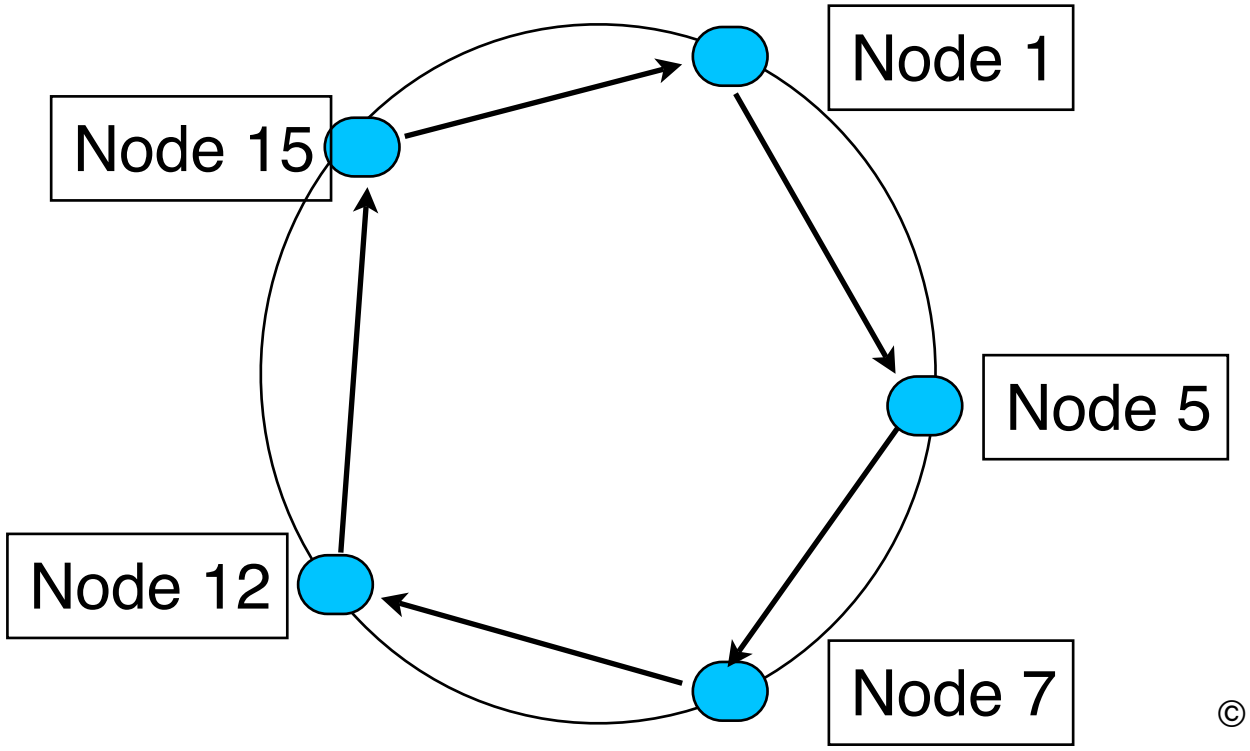
- Evolution of file sharing on the Internet
- Servers using `ftp` protocol
  - A single server that serves all files on disk
  - A set of mirror serves serving the same content
- The innovation introduced by Napster
  - How to distribute many files from many nodes ?
    - Keep the files on their source nodes
    - Central Napster server stores description and URL of each shared file
    - Users willing to obtain a file consult central server to obtain file URL and then download file from their respective source nodes
      - server remains simple and can index large number of files
      - server does not directly participate in file transfer

Peer-to-peer file sharing (2)

- Limitations of the Napster approach
  - a single server indexes all files
  - if a source node fails, then the ongoing file transfers must be restarted
    - completely or partially depending on the file transfer protocol begin used for the transfer
  - performance of file transfer is function of performance of the corresponding source node
    - if source node is connected via ADSL, performance will be severely limited
- How to improve ?
  - Divide the file in blocks
  - Each block can be served by multiple nodes
    - provides redundancy
  - Download from several nodes at the same time
    - one TCP connection may be slow and others faster

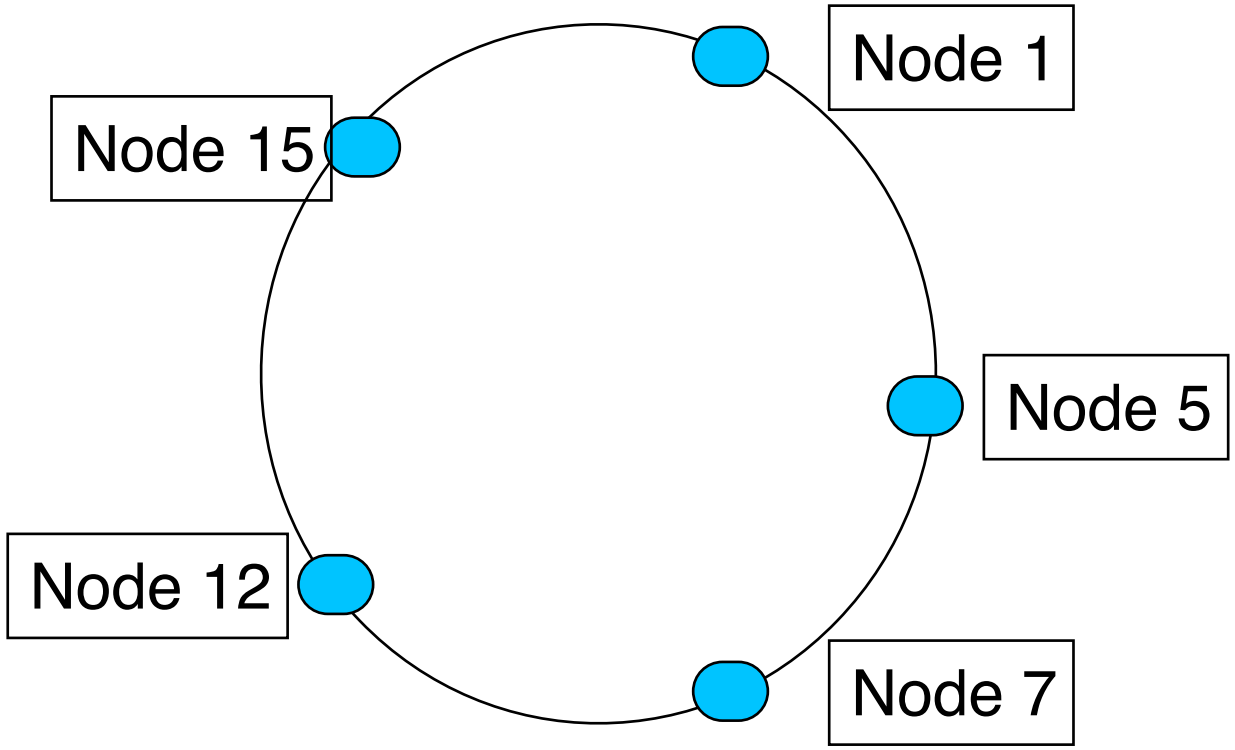
Distributed Hash Table based P2P

- How to scale file sharing to a very large number *n* of nodes ?
- Principle of the solution
  - Use a hash function such as SHA-1
  - Each node has one identifier, id=hash(IPaddress) and a pointer to its successor on the Chord ring



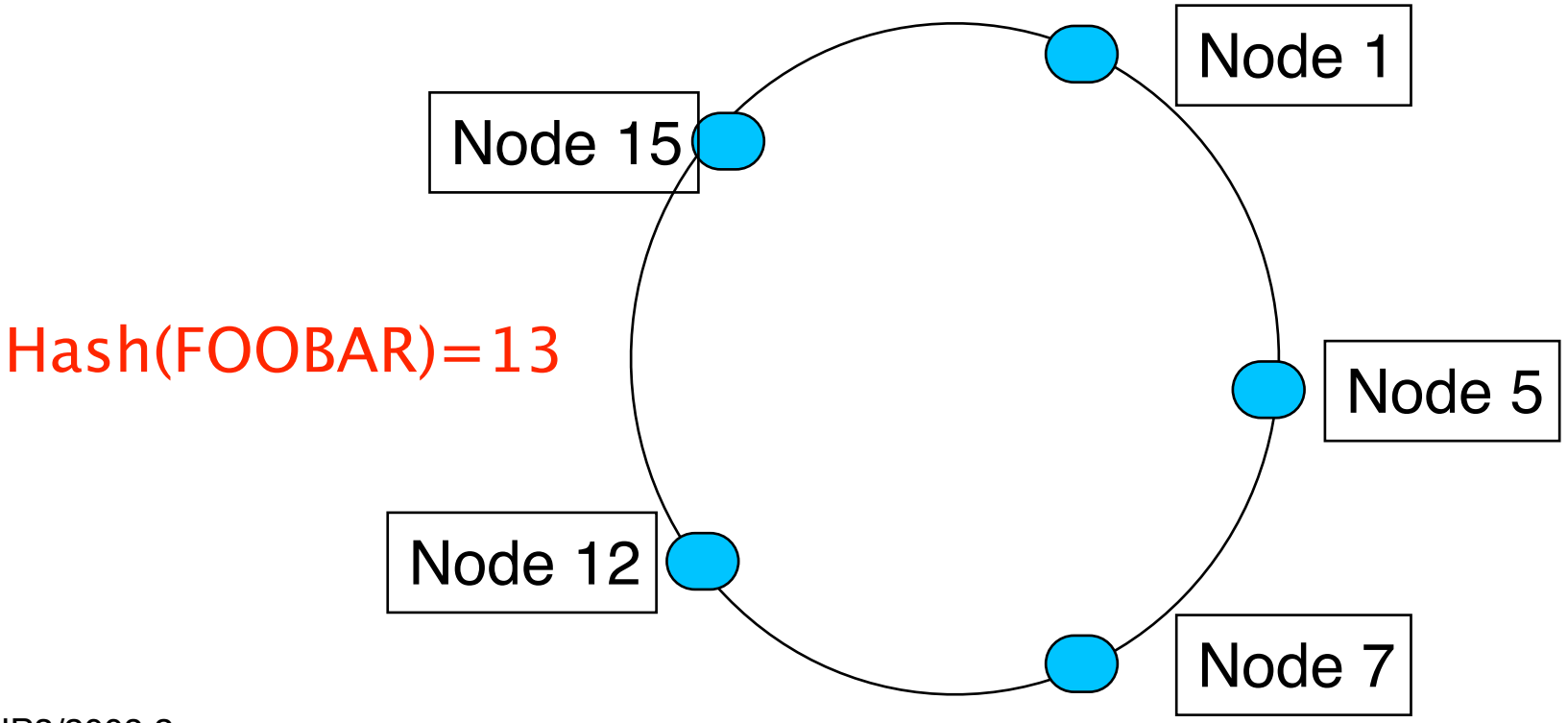
How to store files ?

- Principle
  - File FOOBAR is stored on the node whose id is the successor of hash("FOOBAR") on the ring
  - A node on the ring uses its successors to find the responsible node for a given file
    - Examples



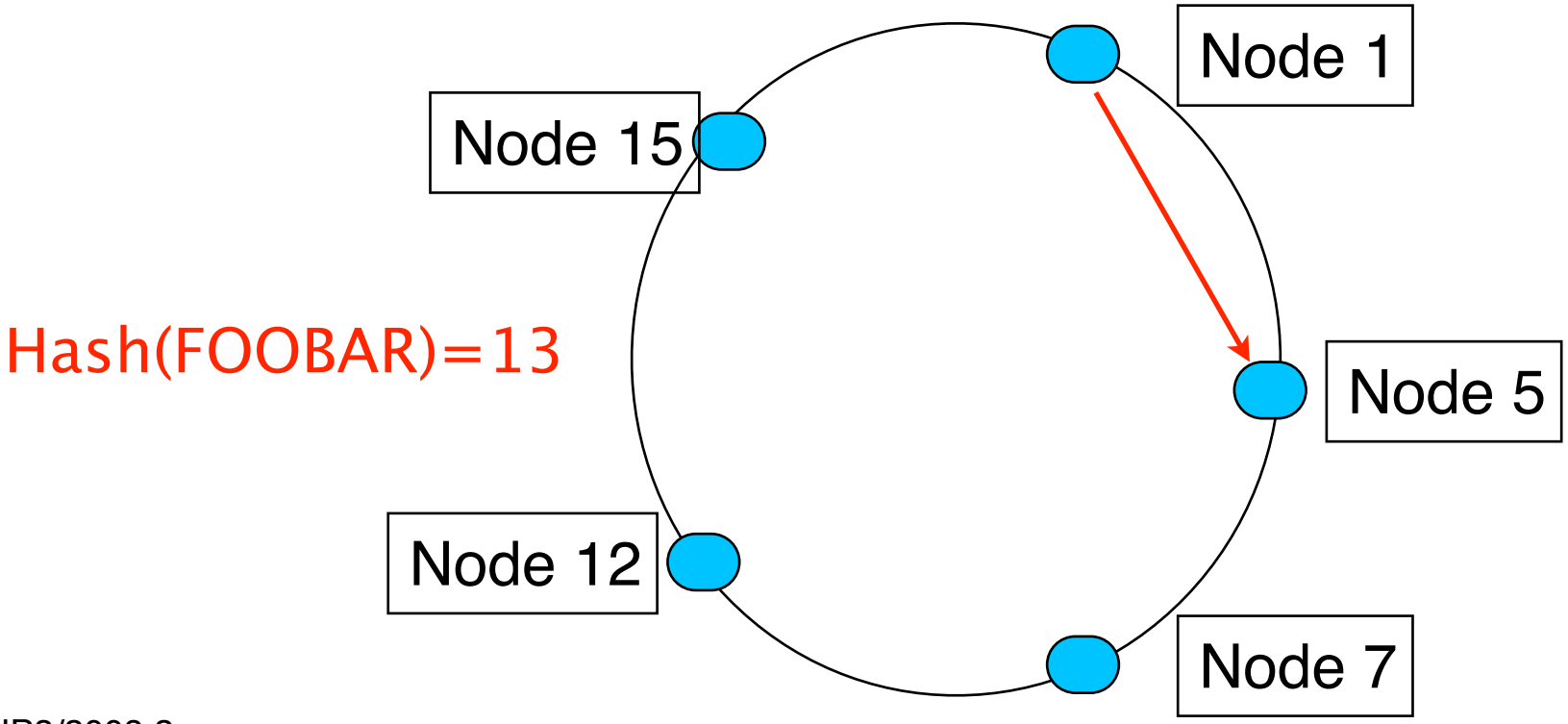
How to store files ?

- Principle
  - File FOOBAR is stored on the node whose id is the successor of hash("FOOBAR") on the ring
  - A node on the ring uses its successors to find the responsible node for a given file
    - Examples



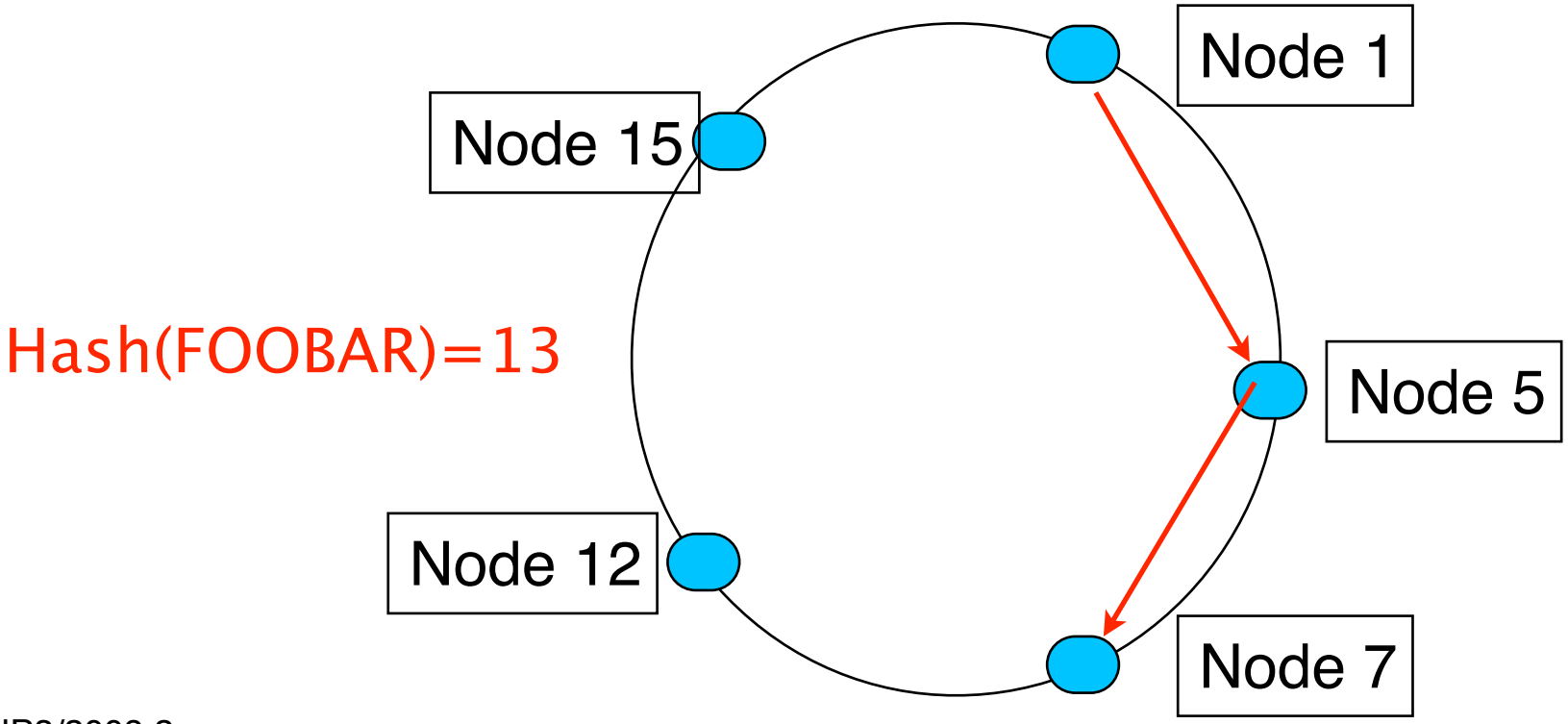
How to store files ?

- Principle
  - File FOOBAR is stored on the node whose id is the successor of hash("FOOBAR") on the ring
  - A node on the ring uses its successors to find the responsible node for a given file
    - Examples



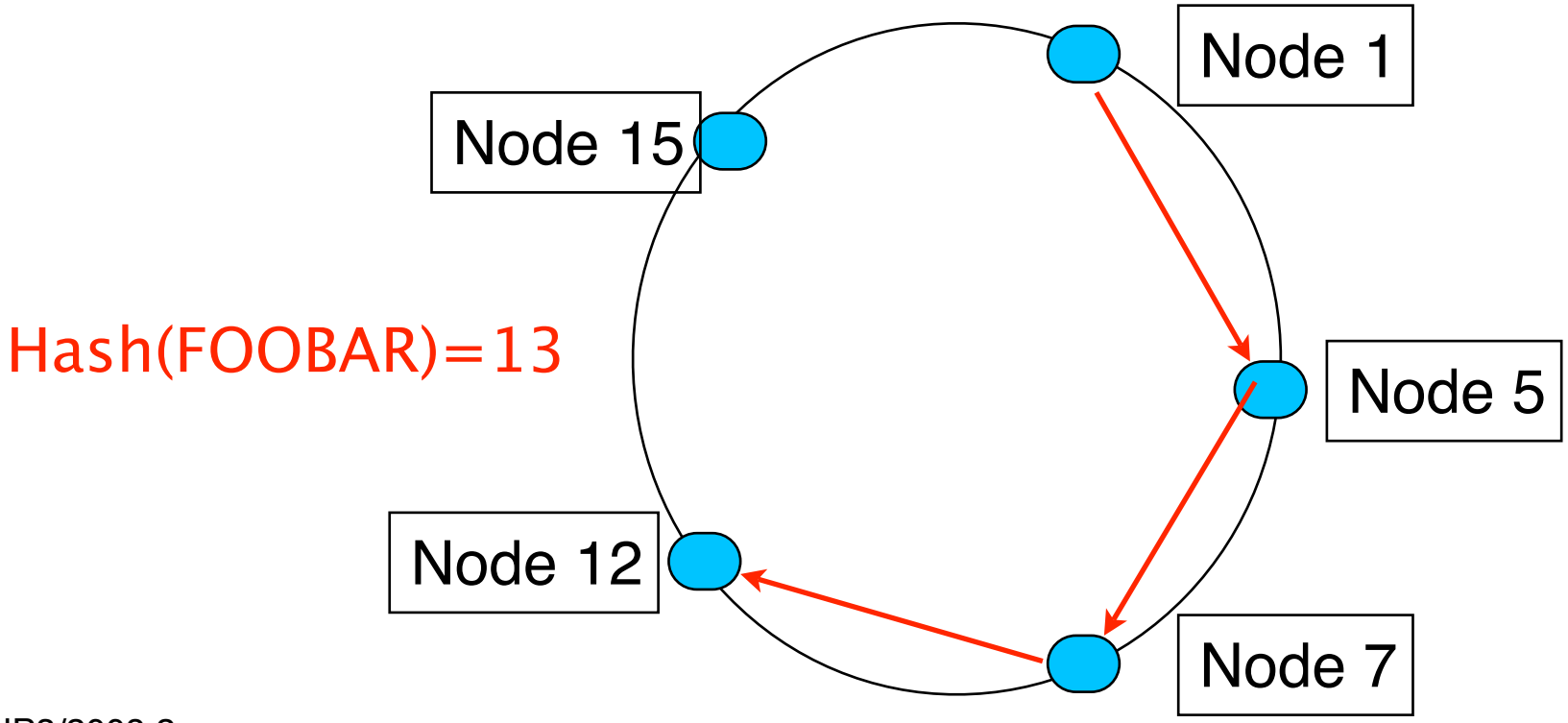
How to store files ?

- Principle
  - File FOOBAR is stored on the node whose id is the successor of hash("FOOBAR") on the ring
  - A node on the ring uses its successors to find the responsible node for a given file
    - Examples



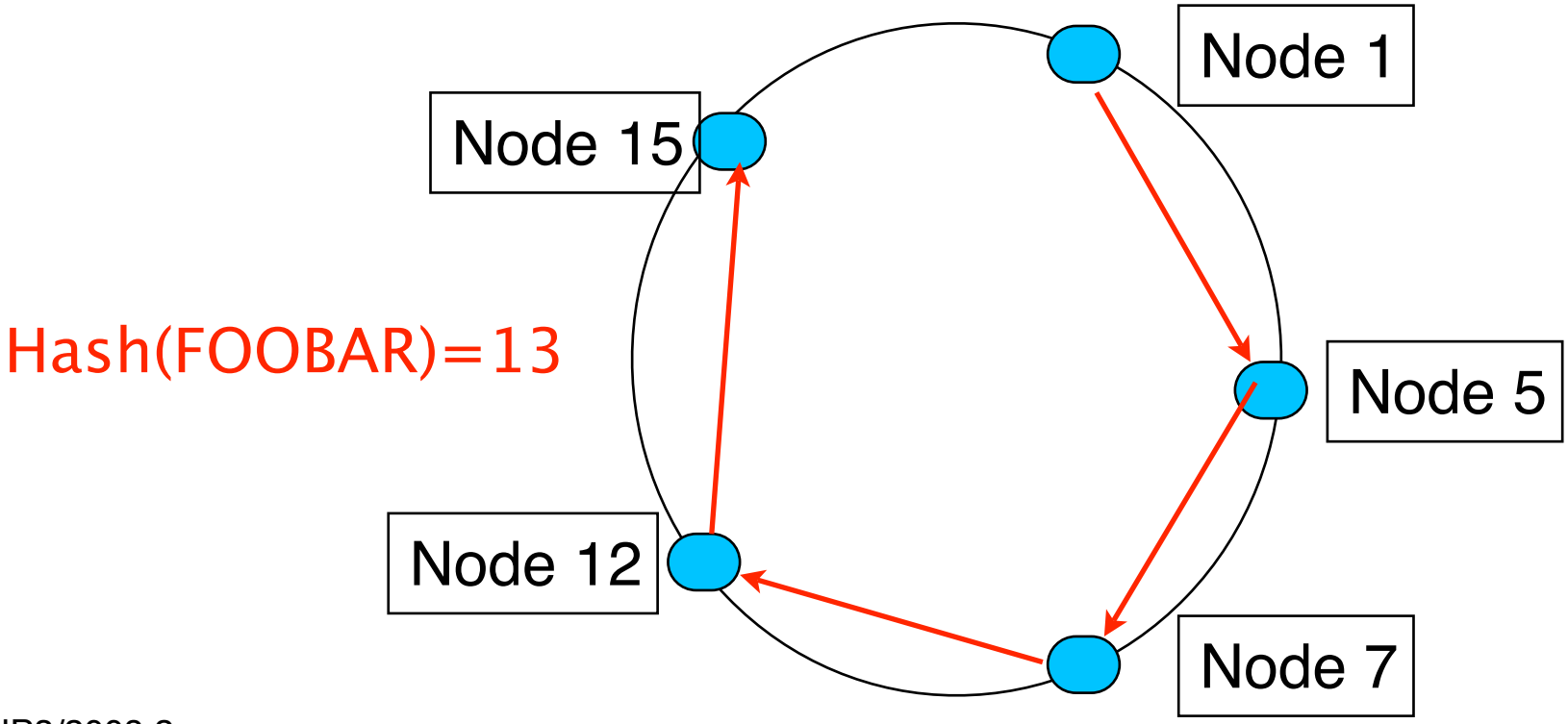
How to store files ?

- Principle
  - File FOOBAR is stored on the node whose id is the successor of hash("FOOBAR") on the ring
  - A node on the ring uses its successors to find the responsible node for a given file
    - Examples



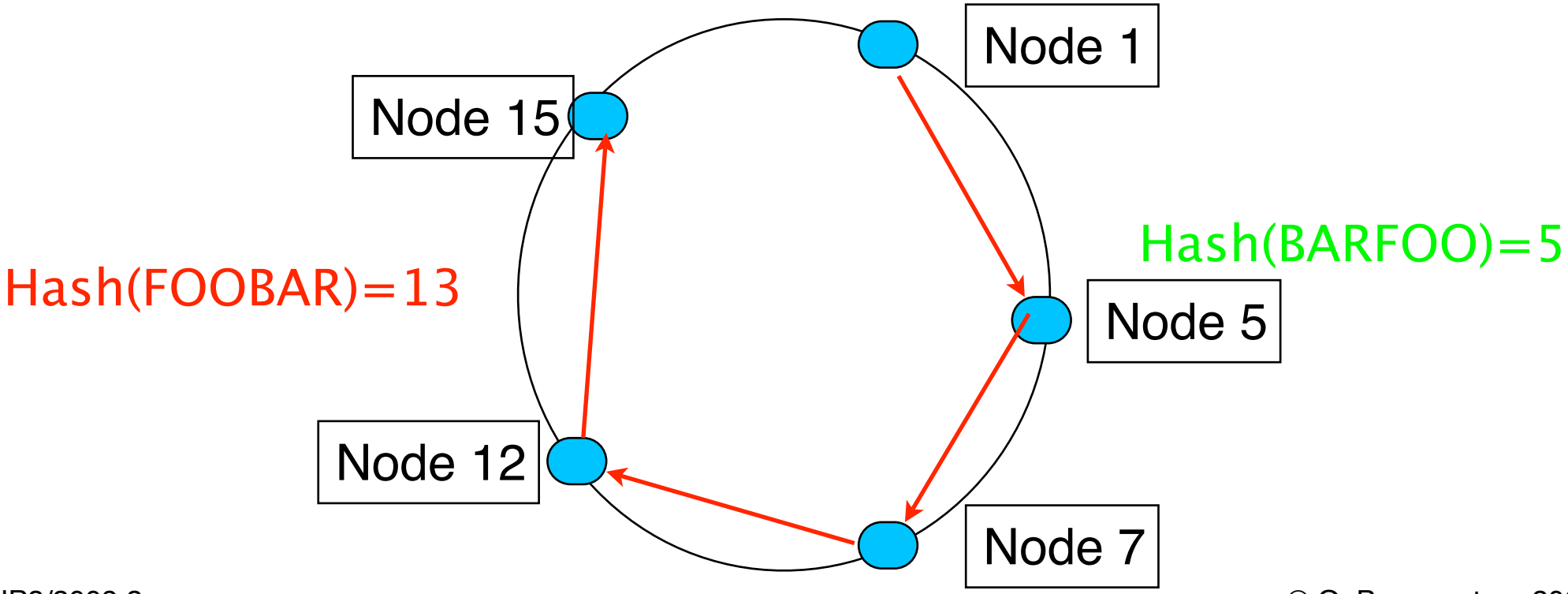
How to store files ?

- Principle
  - File FOOBAR is stored on the node whose id is the successor of hash("FOOBAR") on the ring
  - A node on the ring uses its successors to find the responsible node for a given file
    - Examples



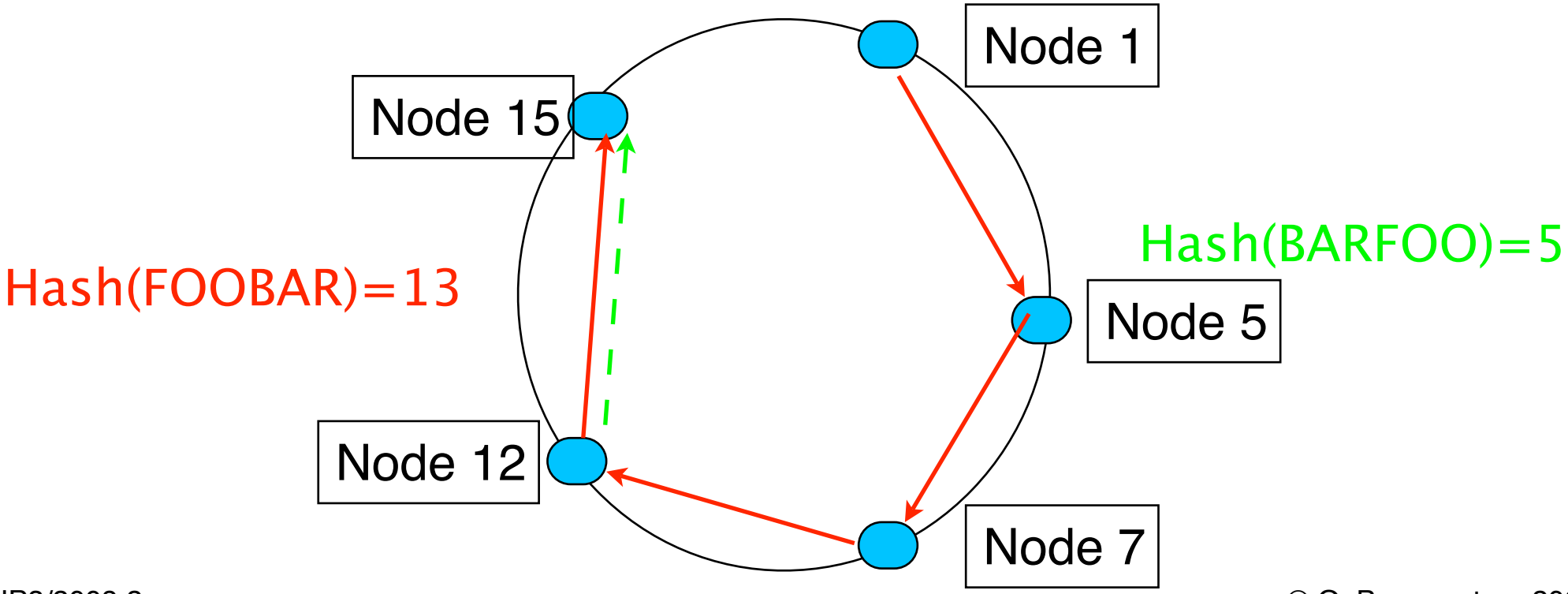
How to store files ?

- Principle
  - File FOOBAR is stored on the node whose id is the successor of hash("FOOBAR") on the ring
  - A node on the ring uses its successors to find the responsible node for a given file
    - Examples



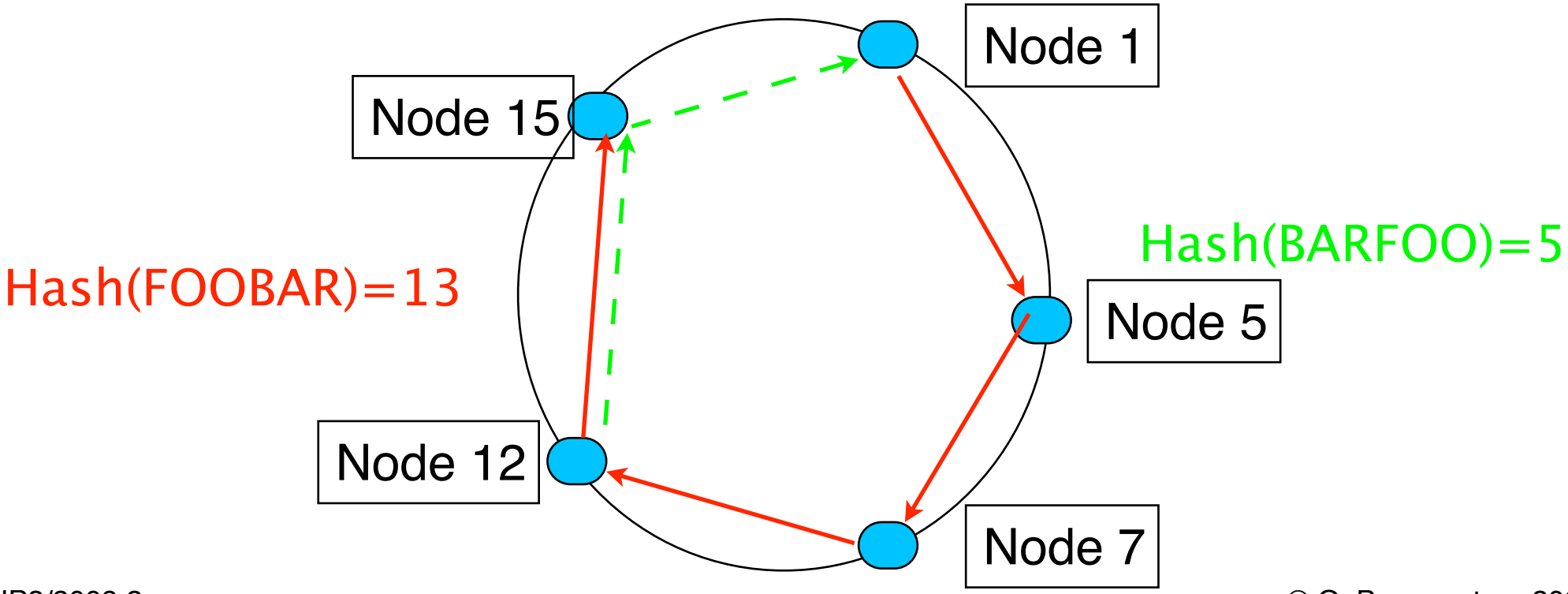
How to store files ?

- Principle
  - File FOOBAR is stored on the node whose id is the successor of hash("FOOBAR") on the ring
  - A node on the ring uses its successors to find the responsible node for a given file
    - Examples



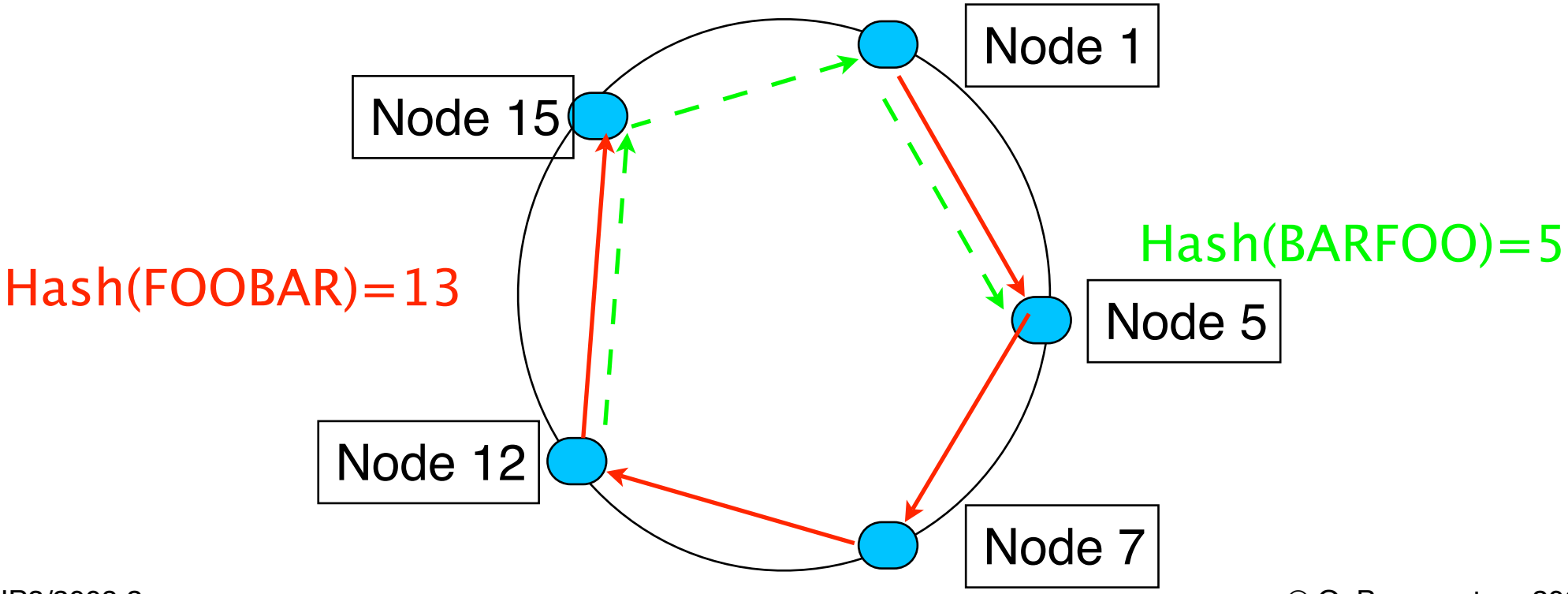
How to store files ?

- Principle
  - File FOOBAR is stored on the node whose id is the successor of hash("FOOBAR") on the ring
  - A node on the ring uses its successors to find the responsible node for a given file
    - Examples



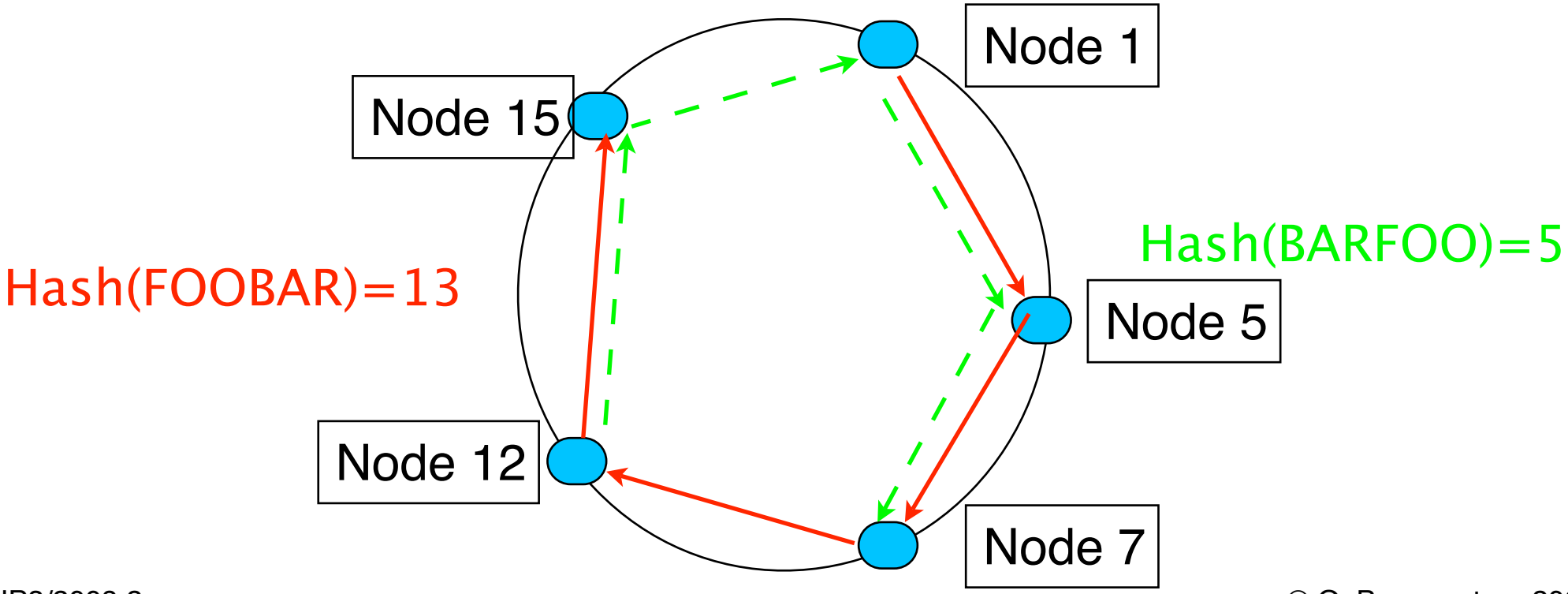
How to store files ?

- Principle
  - File FOOBAR is stored on the node whose id is the successor of hash("FOOBAR") on the ring
  - A node on the ring uses its successors to find the responsible node for a given file
    - Examples



How to store files ?

- Principle
  - File FOOBAR is stored on the node whose id is the successor of hash("FOOBAR") on the ring
  - A node on the ring uses its successors to find the responsible node for a given file
    - Examples



How to find files faster ?

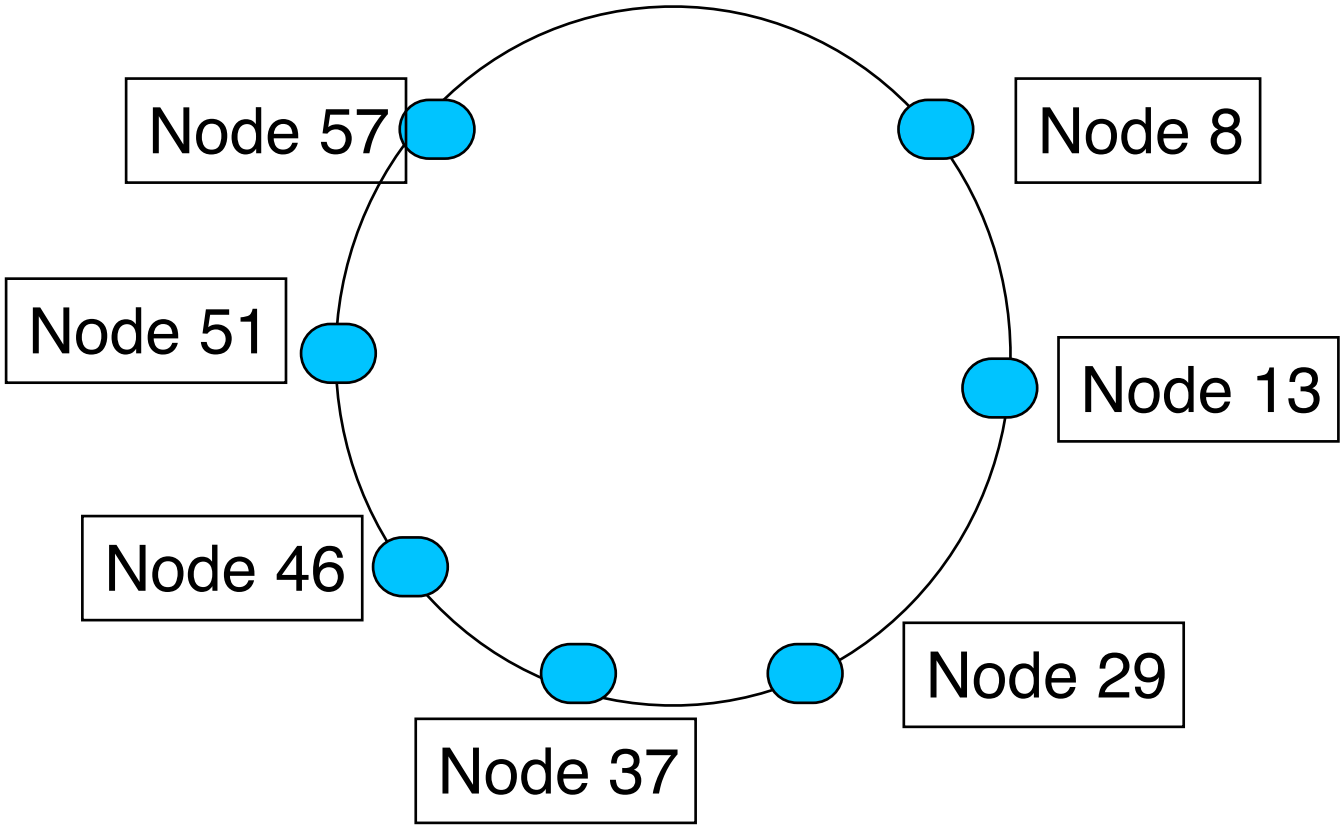
How to find files faster ?

- Performance of file lookup
  - O(n)
    - worst case is to follow linked list of n nodes

How to find files faster ?

- Performance of file lookup
  - O(n)
    - worst case is to follow linked list of n nodes
- How to improve ?
  - Allow nodes to know addition pointers to other nodes on the Chord ring to speedup lookup
    - m = number of bits in the key/node identifiers
    - Each node maintains routing table of m entries
    - The ith entry in the table at node n contains the identity of the first node, s, that succeeds n by at least 2^i–1 on the identifier circle, i.e., s = successor(n + 2^i–1)
      - arithmetic modulo 2^m is used
    - A finger table entry includes both the Chord identifier and the IP address (and port number) of relevant node.

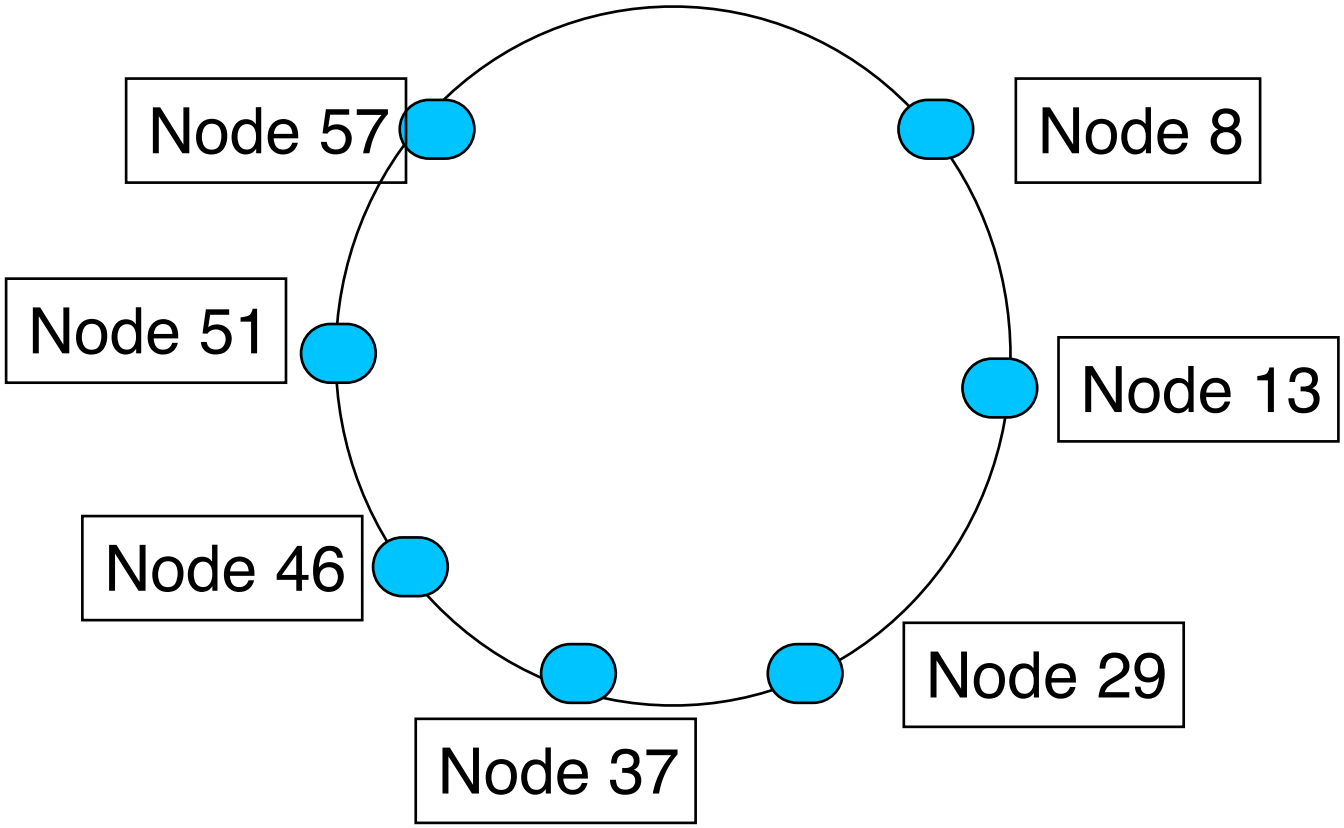
Scalable lookup with Chord



N8+1	N13
N8+2	N13
N8+4	N13
N8+8	N29
N8+16	N29
N8+32	N46

Scalable lookup with Chord

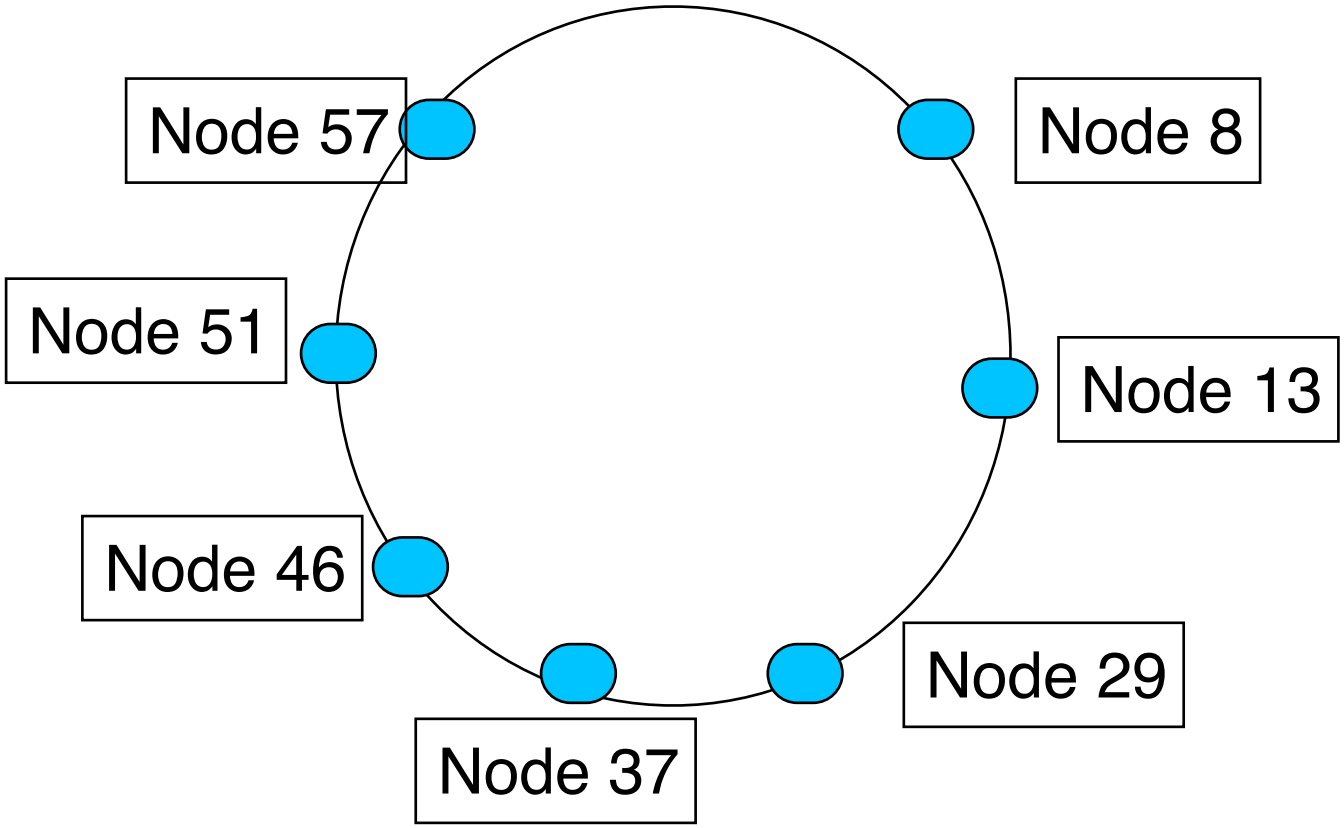
- Example
  - m=8



N8+1	N13
N8+2	N13
N8+4	N13
N8+8	N29
N8+16	N29
N8+32	N46

Scalable lookup with Chord

- Example
  - m=8

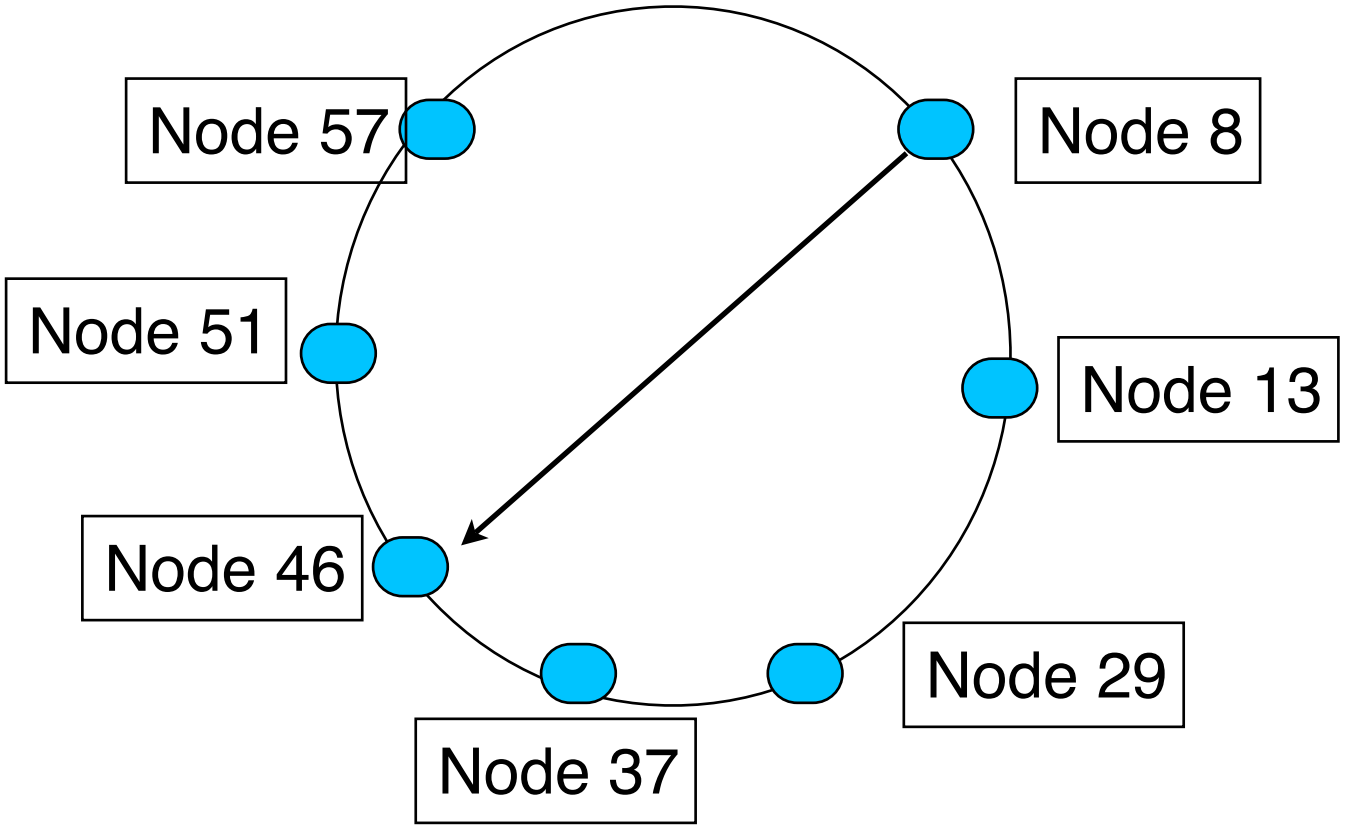


N8+1	N13
N8+2	N13
N8+4	N13
N8+8	N29
N8+16	N29
N8+32	N46

- How to find key 53 from node 8 ?

Scalable lookup with Chord

- Example
  - m=8



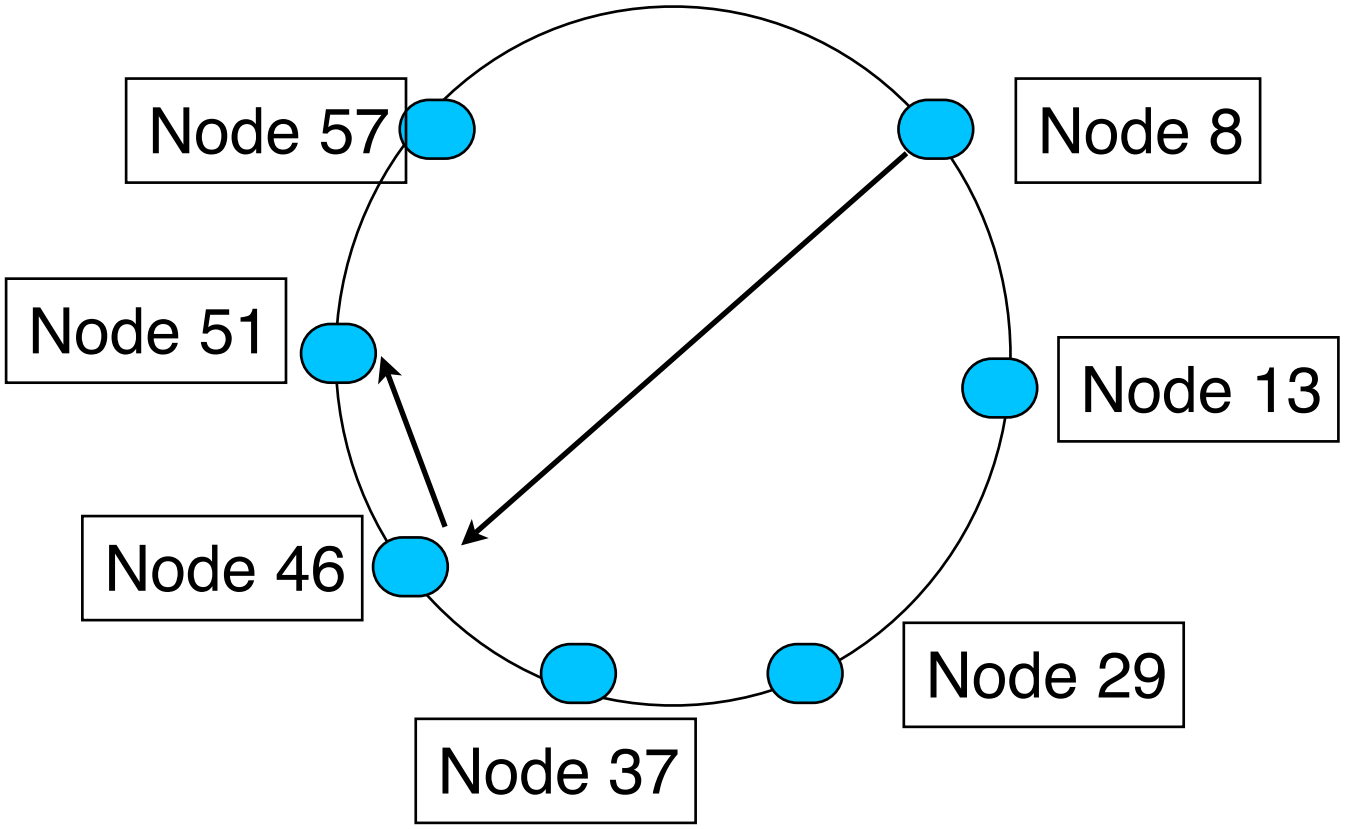
N8+1	N13
N8+2	N13
N8+4	N13
N8+8	N29
N8+16	N29
N8+32	N46

N46+1	N51
N46+2	N51
N46+4	N51
N46+8	N57
N46+16	N57
N46+32	N8

- How to find key 53 from node 8 ?

Scalable lookup with Chord

- Example
  - m=8



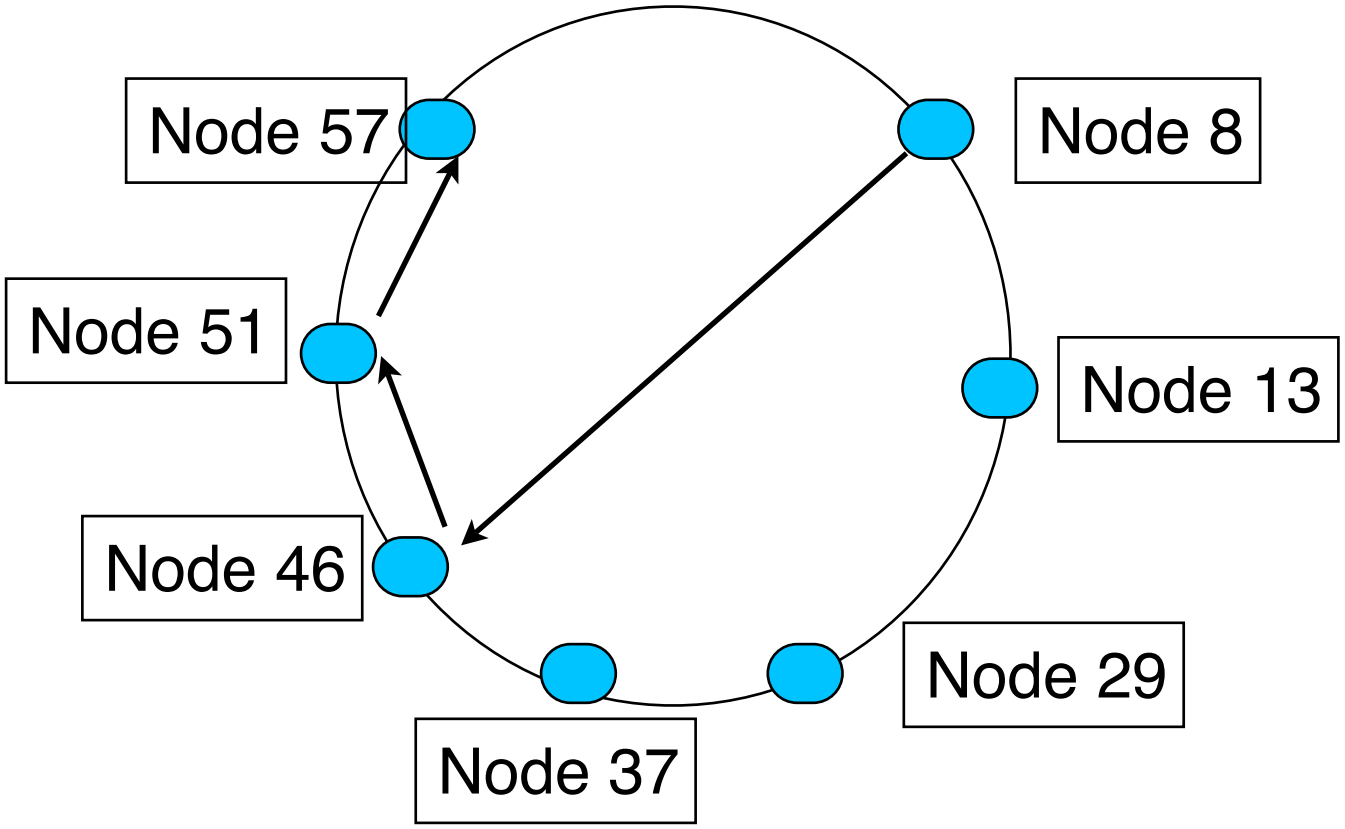
N8+1	N13
N8+2	N13
N8+4	N13
N8+8	N29
N8+16	N29
N8+32	N46

N46+1	N51
N46+2	N51
N46+4	N51
N46+8	N57
N46+16	N57
N46+32	N8

- How to find key 53 from node 8 ?

Scalable lookup with Chord

- Example
  - m=8



N8+1	N13
N8+2	N13
N8+4	N13
N8+8	N29
N8+16	N29
N8+32	N46

N46+1	N51
N46+2	N51
N46+4	N51
N46+8	N57
N46+16	N57
N46+32	N8

- How to find key 53 from node 8 ?

Summary

- Client-server model
- UDP and TCP services
- Application-level protocols
  - DNS
    - relies on UDP, stateless
  - SMTP, POP, FTP
    - rely on TCP, statefull
  - HTTP
    - relies on TCP, stateless
- Peer-to-peer applications