HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Internet Content Distribution

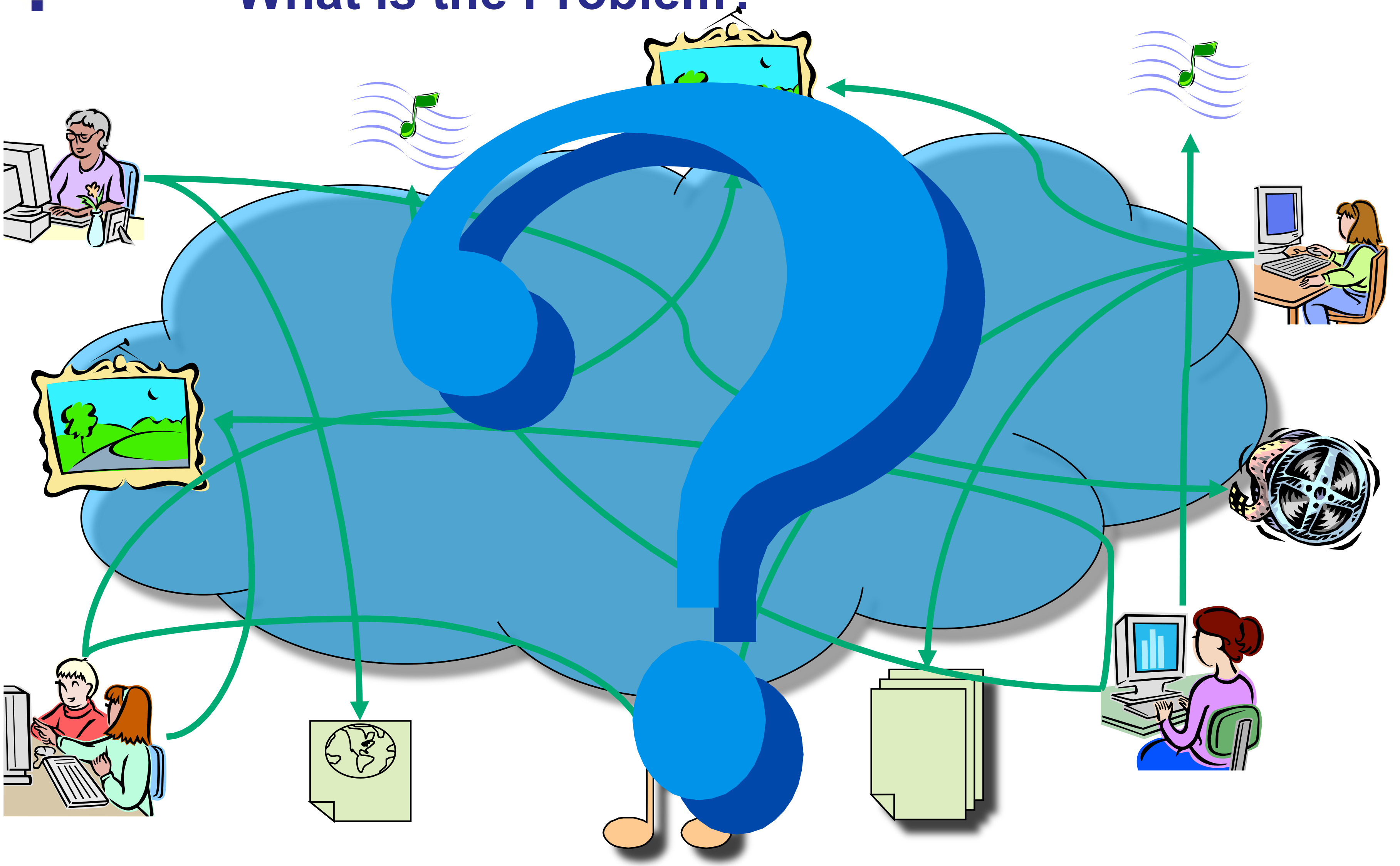
Chapter 1: Introduction

Jussi Kangasharju

Chapter Outline

- Introduction into content distribution
- Basic concepts
  - TCP
  - DNS
  - HTTP
- Outline of the rest of the course

What is the Problem?

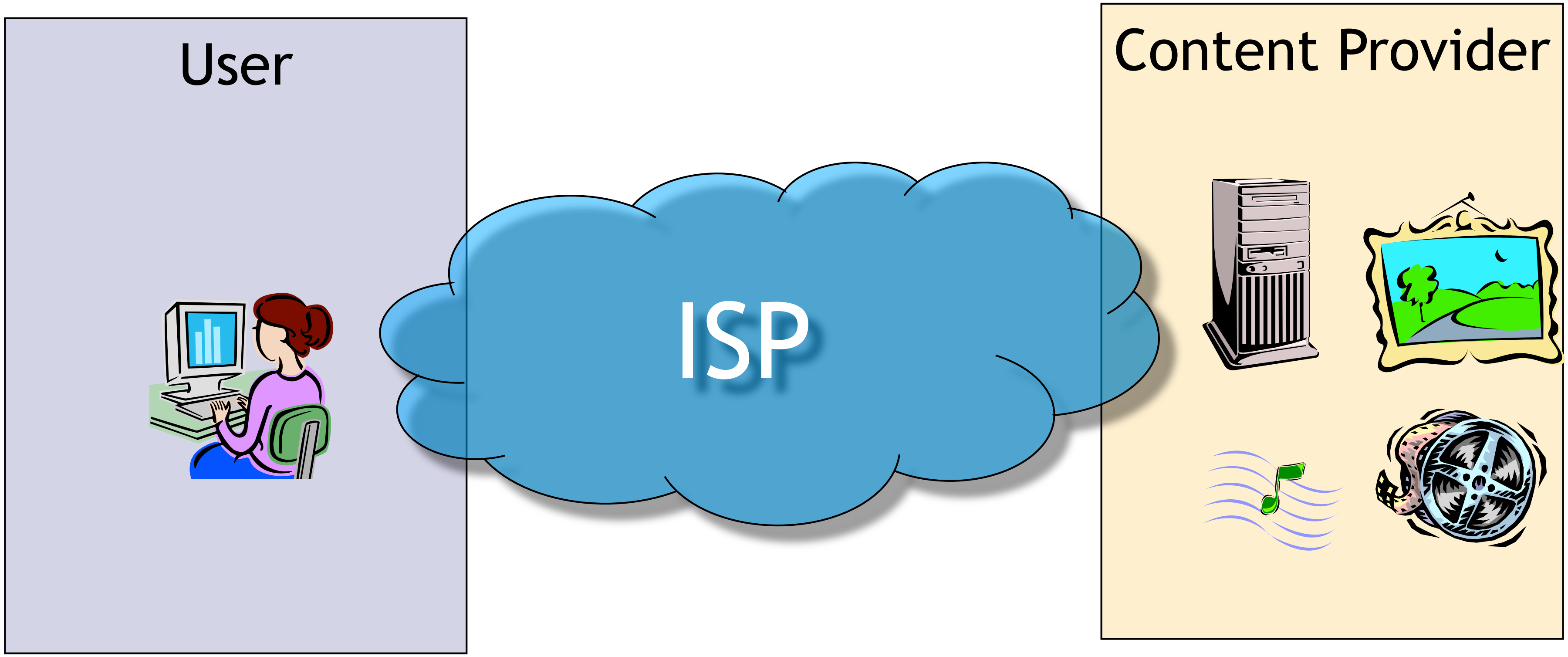


Problem Definition

- How to distribute (popular) content efficiently on Internet to a large number of geographically distributed people?
- …while keeping your costs manageable! :-)
- Who is “you”?
- So, we need to answer the following questions…

1. Who are the players?
2. What is the content?
3. How are costs determined?
4. How is performance measured?

The Players



- ISP represents the network path between the user and the content provider
- Typically, such a path contains several ISPs

Roles of the Players

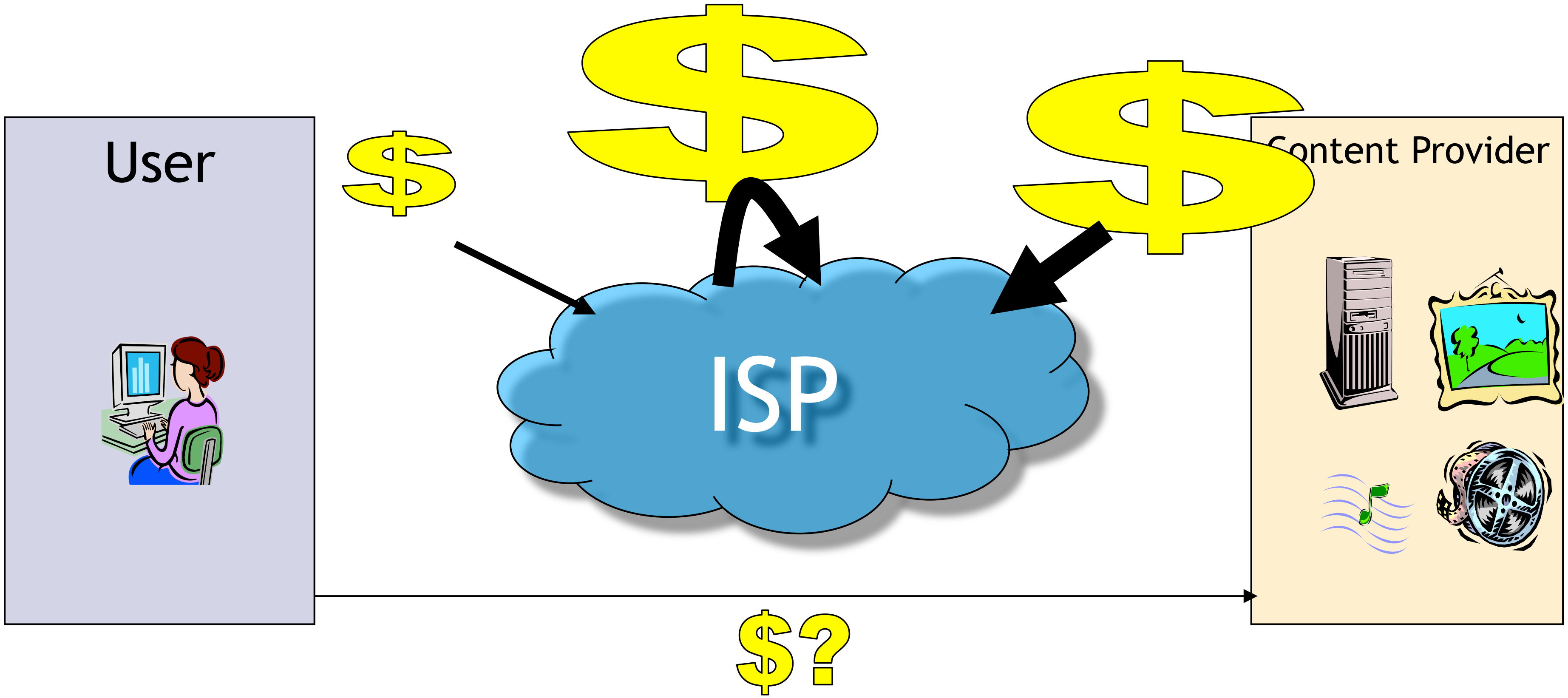
- User
  - Wants to access content provided by content provider
  - Buys access to network from ISP
  - Possibly pays content provider for content
- Content provider
  - Produces content
  - Runs a server (or outsources this)
  - Buys access from an ISP
- ISP
  - Provides the network connection for the others

What Is Content?

- Content is any digital content the users want
- Examples:
  - Web pages
  - Music files
  - Videos
  - Streaming media
  - and so on…
- In this course we will focus mainly on web content
- This means:
  - Mostly small files (a few hundred KB at most, often < 10 KB)
  - Relatively fast delivery (user is waiting to see the page)

Follow the Money

- Where does the money come from?



Business Relationships

- User
  - Pays money to ISP for connectivity
  - These days often flat-rate, can be per-byte
  - Might rarely pay content provider
- Content provider
  - Pays ISP for connectivity
  - Pays for running a server
  - Needs to get money from somewhere
  - *May or may not need to make a profit from content*
- ISP
  - Gets money from others
  - Uses money to run network
  - *Wants to make a profit*

Performance Targets

- User
  - Wants content fast
  - Does not want to pay (too much)
- Content provider
  - Wants to get as many users as possible
  - While maintaining costs as low as possible
- ISP
  - Wants to make as much money as possible
- Which goal is the most important?

- In real world, user needs often not "relevant"
  - Users do not contribute enough money
  - Users might not have a choice, e.g., only 1 ISP possible
- ISP is the most important in many cases
- Content providers have some say in some cases
  - Compare proxy caching and CDNs

Our Focus

- So, what is our focus then?
- We look at the problem from all sides
  - User (client)
  - Content provider (server)
  - ISP (network)
- Different solutions for different parts
- Can use any of the solutions individually or several simultaneously in combination

Constraints

- Where are we allowed to touch things?
- Two main constraints:
  1. No modifications to user software allowed
  2. No modifications to the network allowed
- Above constraints can be relaxed a bit:
  - Can make easy software updates to clients
  - Each ISP can configure his own network
- What remains: Any solution must be backwards compatible
- Practical constraint: *Users do not need to do anything*
  - Too complicated for most users to configure even simple things

What Does That Leave Us?

- So, what can we do?
- Install several servers
  - But how to get users to use other servers?
- Install caches on client side
  - But how to get users to use them?
- Replicate content in the network
  - But how to get users to find the content?
- And so on…
- Core problem: How to get users to use our solution?

- Before we get to that, let's recap the basics…

Internet Basics

- Three main components that are relevant to us

1. TCP
   - All transfers happen over TCP
2. DNS
   - Mapping between server names and IP addresses
3. HTTP
   - Since our focus is on web content, we have HTTP running on top of TCP

- Below a brief recap of the most relevant features of the three
  - And an in-depth look at HTTP performance

TCP

- TCP is a reliable byte-stream transport protocol
- Responsibilities:
  - Reliable end-to-end transport
  - Flow control
  - Congestion control
- Main problem for us is TCP congestion control
- Recall:
  - Three-way handshake
  - Slow-start
  - Congestion avoidance
  - Fast retransmit
  - Fast recovery

Three-Way Handshake

- In order to open a TCP connection, we need to send three packets before any actual application data can be sent

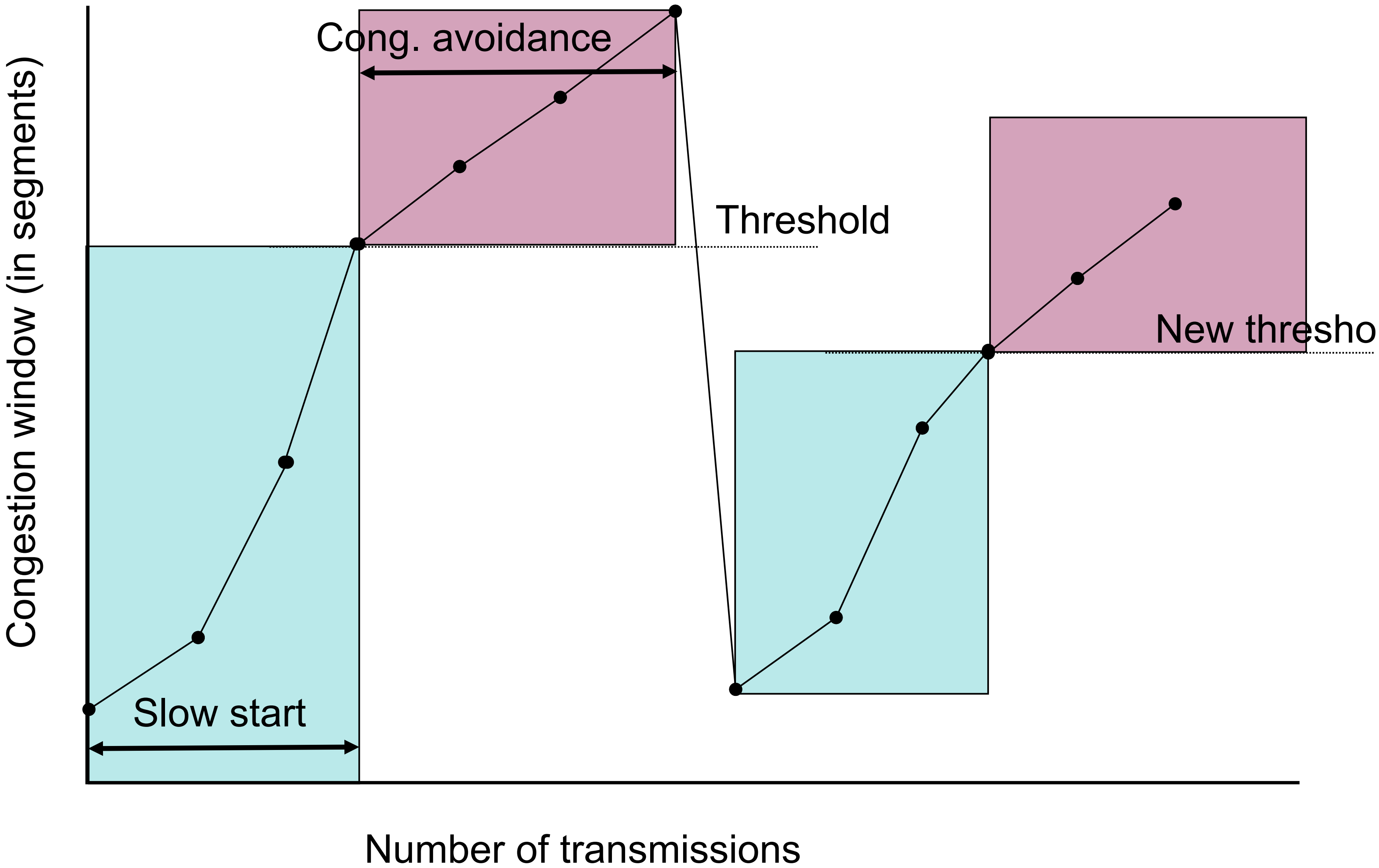
1. Client sends SYN packet to server
2. Server replies with SYN ACK
3. Client acknowledges this with an ACK

- In other words, one RTT delay
  - Actually, a bit more than one
- Nothing we can do about this
- Some extensions have proposed faster transactions (T/TCP), but none have been widely implemented

TCP Basics:
Slow Start and Congestion Avoidance

- Sender calculates a congestion window CW for a receiver
- Start: CW :== 1 segment
- ACKs arrive (no congestion and no other errors): CW increases
  - Exponential (*2) up to congestion threshold, then linear (+1)
  - Congestion threshold dynamically determined
- ACK misses:
  - Congestion threshold :== ½ current CW
  - CW :== 1 segment, ... i.e., slow start again
- TCP is part of a class of protocol called AIMD
  - Additive Increase, Multiplicative Decrease

TCP Congestion Control, SS/CA

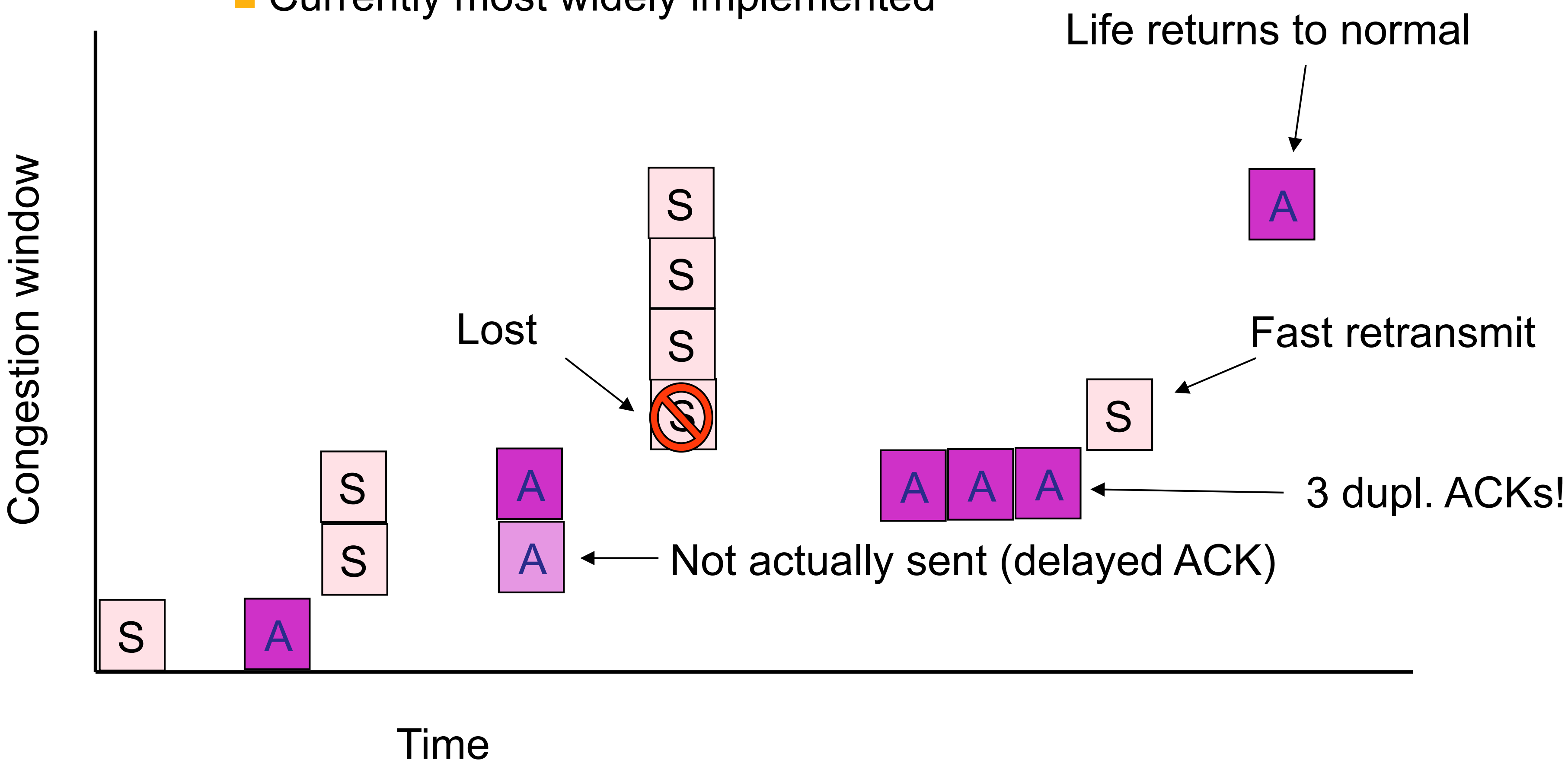


TCP Basics: Fast Retransmit/Recovery

- TCP sends an acknowledgement only after receiving a packet (modulo some enhancements)
- If a sender receives several acknowledgements for the same packet, this is due to a gap in received packets at the receiver
  - Recall: TCP has cumulative ACKs
- However, the receiver got all packets up to the gap and is actually receiving packets
  - Means, (probably) only 1 packet was lost
- Just send that missing packet and hope it fixes the problem
- If congestion short-lived, continue with current congestion window (do not use slow-start)

Fast Retransmit/Recovery

- If congestion short-lived and only 1 packet lost
  - Cutting cwnd to 1 and new slow start too drastic
  - Currently most widely implemented



TCP and Us

- Recall two assumptions
1. Content transferred over TCP
2. Most content is small files
   - But usually not small enough to fit in one packet!

- Result: TCP often never makes it out of slow-start
- This means we cannot really use all the bandwidth in the network that is available to us

- The above is the main effect of TCP on us!
  - Keep this in mind!

Solutions for TCP's Problem

1. Live with the problem
   - A.k.a, the Ostrich approach
2. Try to figure out a way to make connections longer-lived
   - Tricky, but currently the most wide-spread solution
3. Develop a completely new transport protocol
   - Not feasible for web, maybe for new types of content
   - Also, what is the effect of the new protocol on TCP?

- Note that solutions 2 and 3 do require modifications on client side
- The reason why 2 works is that it was implemented in web browsers and people update the browsers
  - Note: Updates are sometimes forced on people, no need to act
  - Still, it took a long time to be widely supported!

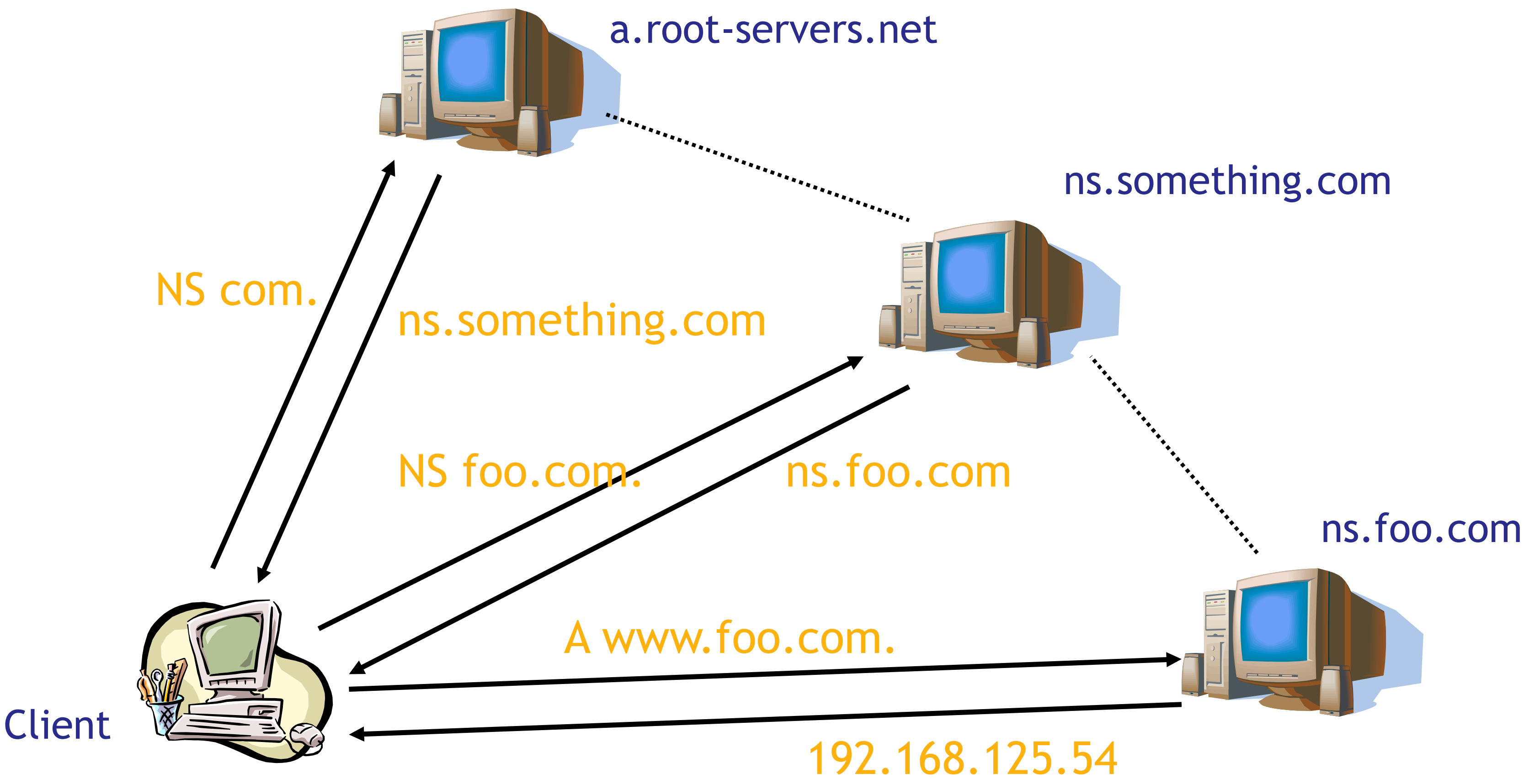
Domain Name System (DNS)

- DNS is a directory service for Internet
- Most commonly used to map hostnames to IP-addresses
- But DNS can do a lot more
  - Although it is not really used for much else
- DNS is a critical service, without it the whole Internet “breaks”
- DNS is a hierarchical system
- Root consists of 13 well-known name servers (called root servers)
- All queries start from the root and work their way down until they find the information they seek

DNS: Overview

- DNS organized in zones (≈ domain)
  - Actual data in resource records (RR)
  - Several types of RRs: A, PTR, NS, MX, CNAME, …
- Administrator of zone responsible for setting up a server for that zone (+ redundant servers at other domains)
- Owner of a zone is responsible for serving zone's data
- RRs can be cached on client side
  - Up to a period determined by zone's administrator

DNS: Example



- Client wants to resolve www.foo.com
- Replies to queries have additional information (IP address + name)
- Queries can be iterative (here) or recursive

DNS Queries

- As shown above, DNS queries typically iterative
- Exception: Client (= user's PC) sends recursive query to its local name server which then proceeds iteratively
- Answers to queries may also be different for different clients
- Why is that an important feature?
- Short answer: We can do all kinds of nice tricks
- Long answer: DNS redirection, see CDN chapter :-)

DNS and Content Distribution

- All clients must use DNS to resolve IP-addresses
  - Mandatory step, implemented everywhere
  - Because of its ubiquity, DNS is a viable way to improve content distribution
- DNS used in content distribution mainly in two ways:

1. Load balancing on server side
   - See Chapter 2
2. DNS redirection on client side
   - See Chapter 4

- Besides those, DNS is simply a black box for us

HTTP

- HyperText Transfer Protocol (HTTP) is The Protocol™ for delivering Web content
- For a while in late 90's, HTTP was the cause for the largest share of Internet traffic
  - Before it, the "big fish" was email, afterwards P2P
  - HTTP-share getting bigger again (YouTube and others)
- HTTP standardized by the World Wide Web Consortium (W3C) and by IETF
- Two versions have been standardized:
  - HTTP/1.0: Earlier version, simple protocol
  - HTTP/1.1: Latest version, much more complicated
- Currently, HTTP/1.1 already widely implemented
  - Spread through browser updates :-)

HTTP Details

- HTTP is a (simple) client-server protocol
- Clients request files, one file at a time
- Files identified by Uniform Resource Locators (URL)
- In web context:
  - One HTML page
  - Several images
  - → Need to request several files to show one page to user
- Images are referred to on HTML page
  - References can be relative or absolute
- Relative reference: URL path is the same as HTML
- Absolute reference: Absolute URL for image
- Keep this distinction in mind!

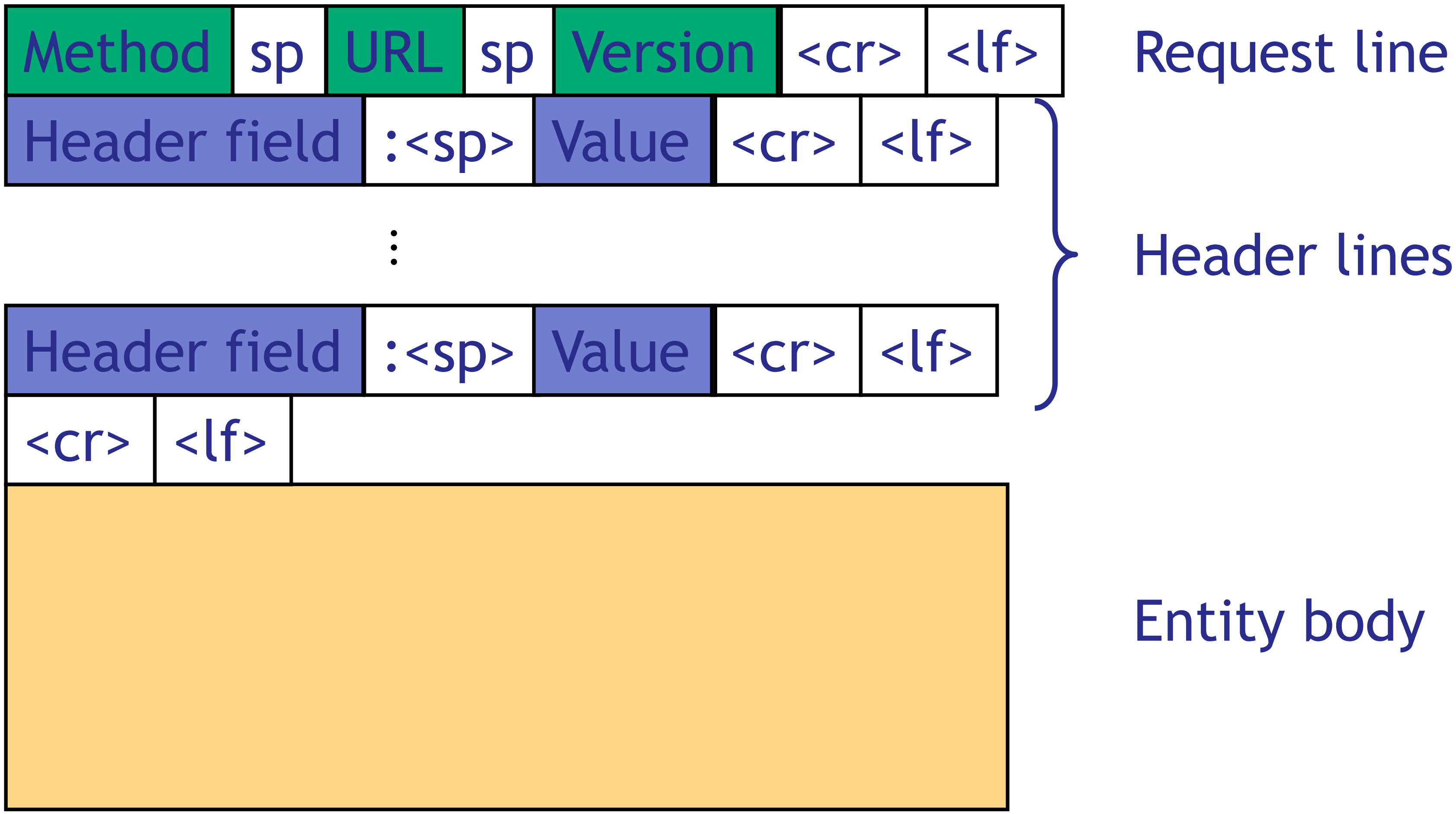
HTTP Requests

- Example request:

```
GET /index.html HTTP/1.1
Host: www.google.com
Connection: close
User-Agent: Mozilla 1.6
<CR><LF>
```

- Request line, header lines, possible body

HTTP Request Format



HTTP Requests

- Method: GET, POST, HEAD, ... (see RFC 2616)
- Normal requests GET
  - Retrieving web pages, images, etc.
  - Simple forms also processed with GET
- Entity body in POST
  - POST used for complicated forms (lot of info to handle)
  - Contents of form in entity body
- Headers give more information about request or modify it in some way
- We will see more HTTP headers in client-side techniques chapter

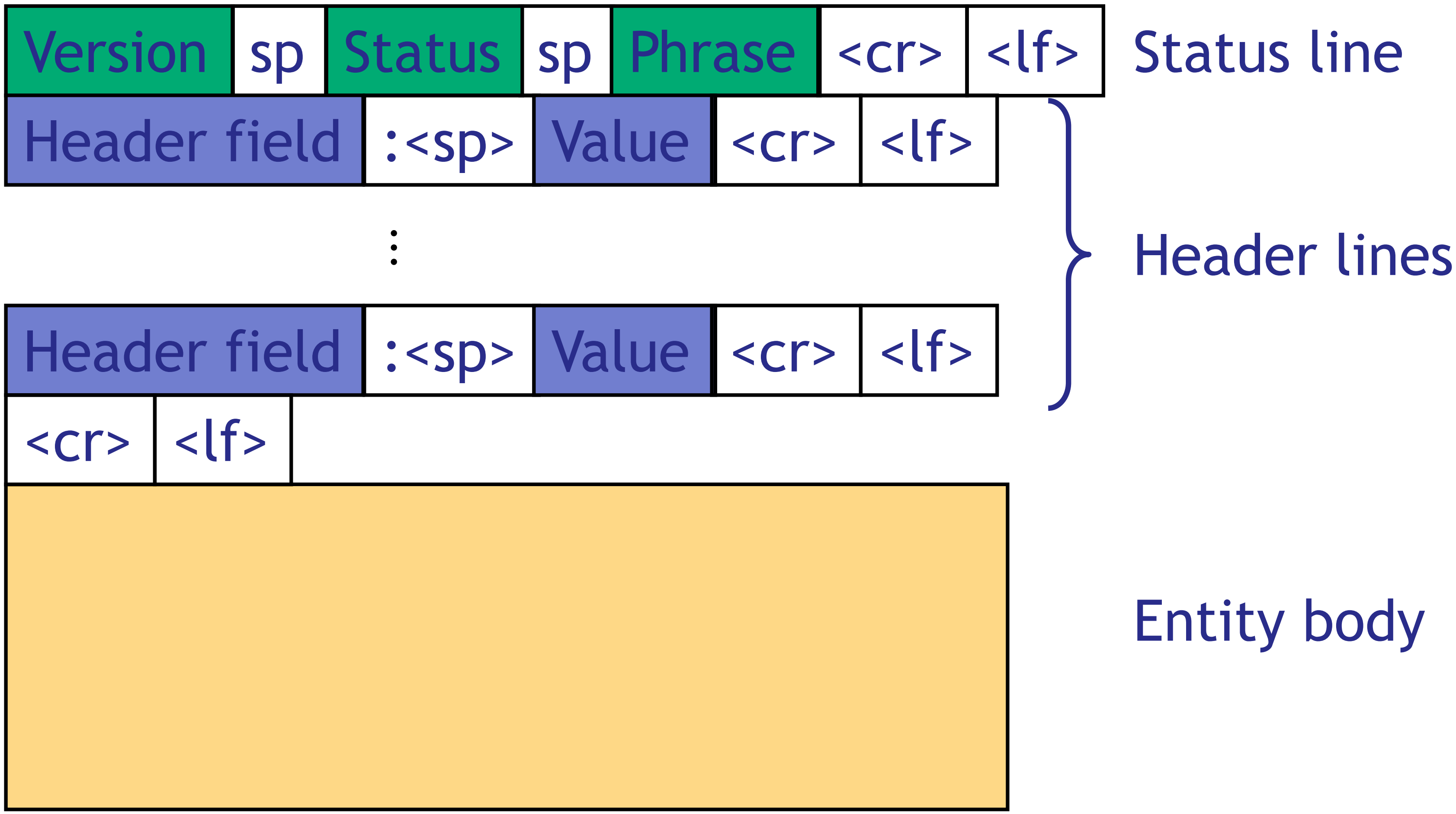
HTTP Responses

- Example response:

```
HTTP/1.1 200 OK
Date: Thu, 06 Aug 1998 12:00:15 GMT
Connection: close
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 09:23:24 GMT
Content-Length: 6821
Content-Type: text/html
<CR><LF>
(data data data data data)
```

- Status line, header lines, requested document

HTTP Response Format



HTTP Responses

- Status code gives result
- Phrase for humans, only code is important!
- Typical status codes with standard phrases:
  - 200 OK: Everything went fine, information follows
  - 301 Moved Permanently: Document moved, new location in Location-header in response
  - 400 Bad Request: Error in request
  - 404 Not Found: Document does not exist on server
  - 505 HTTP Version Not Supported: Requested protocol version not supported by server
- Headers give more information, like with requests

HTTP Connections

- Basic HTTP interaction: Client sends a separate HTTP request for each object
  - Read: Client opens a new TCP connection for each object
- These are so-called non-persistent connections
- Implemented in early versions of HTTP
  - And still implemented for backwards compatibility
- Consider web page with 10 images
- 1 HTML page + 10 images = 11 files = 11 TCP connections
- Each connection has to do 3-way handshake
- Each connection has to go through slow-start
- Very inefficient!

HTTP Persistent Connections

- Solution to above problem: Persistent connections
- Client keeps connection to server open and sends several requests over the same connection
- Two advantages:
  - 3-way handshake only in the beginning
  - Slow-start only once
- When download of a file finishes, client can send new request
- This means 1 RTT delay between objects
  - Compare to 2 RTT for non-persistent connections
  - Better, but still not stellar…

HTTP Pipelining

- Pipelining remedies the 1 RTT delay in persistent connections
- Idea: Client sends all requests it has at once and server processes them one after the other
- Benefits:
  - No need to wait between objects
  - Continuous download (good for TCP congestion control)
- Disadvantage:
  - Not widely implemented nor supported

HTTP in Practice

- Browsers typically open several connections in parallel
  - In particular for non-persistent connections
- Typically, 2-6 connections are used simultaneously
- HTTP/1.0 defined keep-alives
- HTTP/1.1 defined persistent connections
  - Both are functionally equivalent

- What are the real advantages of persistent connections and pipelining?
- Results from Nielsen et al. Network Performance Effects of HTTP/1.1, CSS1, and PNG, published in SIGCOMM '97

Test Setup

- Synthetic website, similar to popular websites
  - Similarity was true in 1997, but still close to reality today
- 1 HTML page, size 42 KB
- 42 inlined images, total size 125 KB
  - Inlined images 70B -- 40KB
  - 19 images < 1KB, 7 images between 1KB and 2KB, and 6 images between 2KB and 3KB
- Three kinds of network conditions:
  - High bandwidth, low latency (LAN)
  - High bandwidth, high latency (wide-area)
  - Low bandwidth, high latency (modem)
- Computers running mostly variants of Unix

Test Clients

- HTTP/1.0, 4 parallel non-persistent connections (HTTP/1.0)
- HTTP/1.1, persistent connections (HTTP/1.1)
- HTTP/1.1, persistent connections with pipelining (Pipeline)

	HTTP/1.0	HTTP/1.1	Pipeline
Max simultaneous sockets	6	1	1
Total sockets	40	1	1
Packets client->server	226	70	25
Packets server->client	271	153	58
Total packets	497	223	83
Elapsed time	1.85s	4.13s	3.02s

Analysis of Results

- Both persistent connections send less packets
- Persistent connections with pipelining are much slower than non-persistent connections
- How is this possible?!?
- How can persistent connections be slower on a lightly loaded LAN?!?
- Short answer: There is no reason for that
- Long answer: You need to tune things correctly
  - Flushing buffers, Nagle's algorithm, connection management, …
- In other words, initial comparison was not fair

Let's Try Again… :-)

- Results for high bandwidth, high latency
  - Average over 5 runs

	Packets	Bytes	Seconds	TCP overhead
HTTP/1.0	565.8	251913	4.17	8.2%
HTTP/1.1	304	193595	6.64	5.9%
Pipeline	214.2	193887	2.33	4.2%
Pipeline w/ compr.	183.2	161698	2.09	4.3%

What Do We Learn?

- Best performance is with persistent connections and pipelining together
  - 50% gain in retrieval time
- Non-persistent but parallel connections always beat persistent connections without pipelining in terms of elapsed time
- Getting the implementation details right is hard
- Lots of dependencies between operating system, network stack, and application
- Bottom line: Done correctly, persistent connections with pipelining is the best solution
- Currently, pipelining not widely implemented, but persistent connections are
  - And parallel connections… ☺

Chapter Summary

- Introduction to content distribution problem
- Basic technologies:
  - TCP
  - DNS
  - HTTP
- We will revisit each of them later, when they are needed

Outline of Future Chapters

- Chapter 2: Server-side techniques
- Chapter 3: Client-side techniques
- Chapter 4: Content Distribution Networks (CDN)