3. Enabling Web and Software Technologies

3.1 Client / Server Architectures and the Internet

3.2 Internet Protocols and Internet Infrastructure

HTTP, Security for HTTP, Web Servers

3.3 Multi-Tier Architectures

3.4 Platform Choices and Connectivity Options

Web Technologies (1)

HTTP: Hypertext Transfer Protocol. Purpose: Accessing resources on the internet (web documents). Clients (browsers) issue requests for resources to a server, the server sends the requested document back to the client as a response. Current version is HTTP/1.1.

HTTP Requirements:

1. **Infrastructure**

Proxies, Gateways, Tunnels, Mirrors, Firewalls: Important additional client- and server-side resources on the web used to enhance performance, availability, accessibility and to protect servers, etc.

2. **Target Identification**

URL: Uniform Resource Locator. Defines the location of a resource on the internet. Example: http://www.sts.tu-harburg.de/teaching/ is a URL.

3. **Service Messaging**

Requests, Responses, Headers, Extensions, Negotiation, etc. regulate the format of messages which communicate service details

Uniform Resource Locator (URL) (1)

A **uniform resource locator** (URL) defines the location of a resource on the Internet in a constructive way. A resource can be something as simple as a file or a directory, or it can be a reference to a more complicated object, such as a query to a database or to a search engine.

A URL comprises several parts: *<Protocol>*:// *<Host>* :*<Port>* / *<Path>*

Example: `http://www.sts.tu-harburg.de:80/teaching/entry.html`

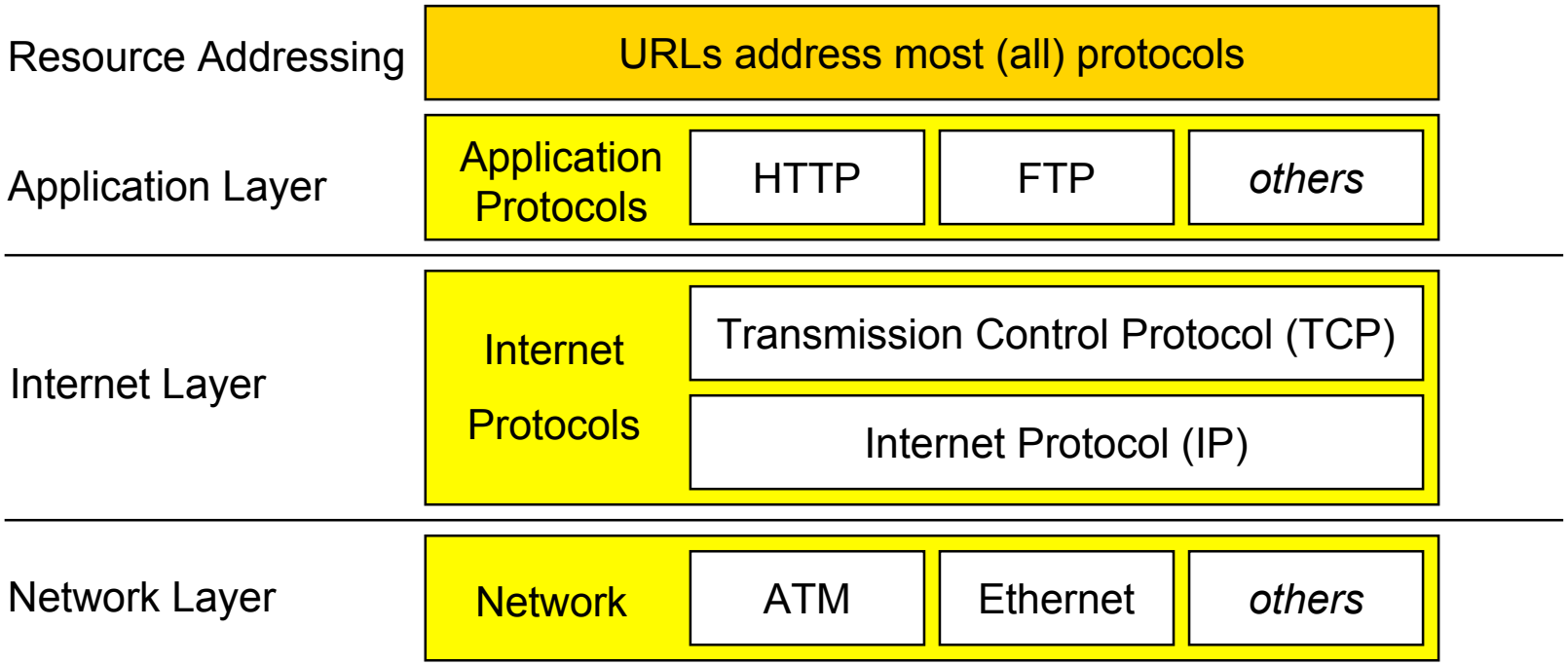
- The *Protocol:* specifies which protocol to use to access the resource. Examples: http, https (secure http), ftp, ldap, rmi, ...
- The *Host:* specifies the server on which the resource is stored. Usually, the host can be specified either as host name (www.sts.tu-harburg.de) or as host IP address (134.28.70.1).
- The *Port:* defines the port to which to connect to on the resource server. For standard protocols, this is determined by the protocol (http uses port 80 or 81, https uses port 343, etc.).
- The *Path:* specifies the name of the resource on the host. The exact meaning of this name on the host machine is both protocol dependent and host dependent. The target information (content) normally resides in a file, but it could be generated on the fly (example: virtual directories).

Uniform Resource Locator (URL) (2)

Example

http	://	www.sts.tu-harburg.de	/teaching/entry.html
Protocol		*Host*	*Path*

In the example the port is omitted; it can be deduced from the protocol.



URL Extensions

These extensions are commonly used, but are not part of a URL by definition:

- *Fragment*

  A fragment indicates that after the specified resource is retrieved, the application is specifically interested in a part of the resource that is marked-up (tagged) by the fragment‘s name. The meaning of a fragment is resource-specific.

  Example:

  http://www.sts.tu-harburg.de/teaching/entry.html**#top**

- *Parameters*

  Parameters are used to transmit – a limited amount (max. 4 KBytes) of - information from the client to the server.

  Example:

  http://www.register.com**?firstname=patrick&lastname=hupe**

Comparison: URI - URL - URN

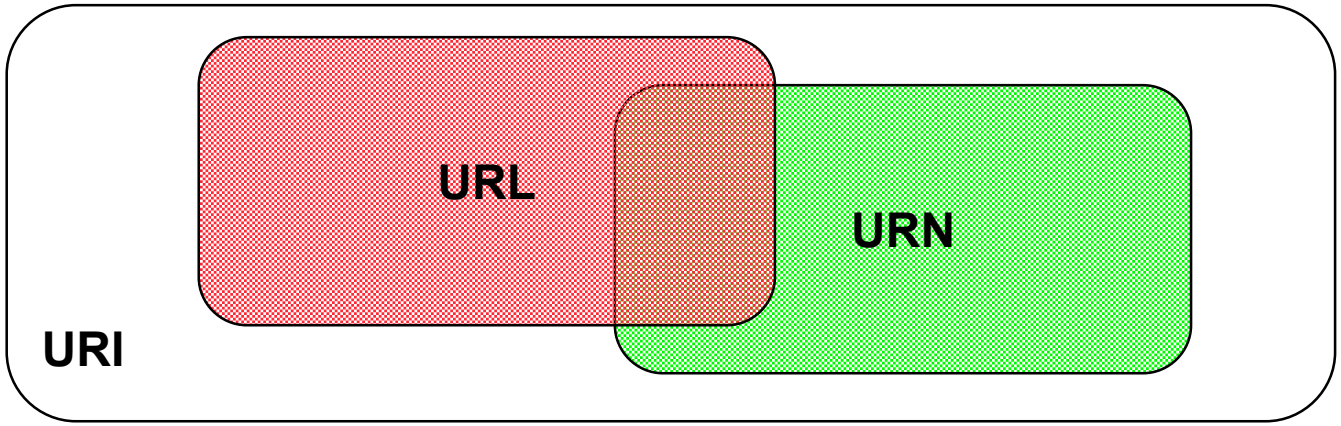
URLs are used to identify resources on the Internet. Still, a generalization of resource identifiers exists:

Definitions

A **Uniform Resource Identifier (URI)** is a name or address that identifies a resource on the Internet. URIs comprise both URLs and URNs.

While a URL only identifies a resource on the Internet, but persistence and availability of the resource are not guaranteed, a **Uniform Resource Name (URN)** is a persistent name for a resource. The name – usually - follows a particular scheme (urn:...) and is intended to serve as a persistent, location-independent, resource identifier.

NOTE: URNs can - again - be URLs (see e.g., Persistent URLs, http://purl.oclc.org/).



HTTP Messages

HTTP requests and response follow the HTTP message format.

Generic HTTP message format:

generic-message ::= *start-line* (*message-header*)* <CR/LF> (*message-body*)?
start-line ::= *request-line* | *status-line*
message-header ::= *general-header* | *entity-header* | *request-header* | *response-header*

- Entity-headers contain meta-information describing the entity (content).
- Request-headers may only appear in requests.
- Response-headers may only appear in responses.

For examples of HTTP Messages (GET and POST requests) see slides 3.51 and 3.53, 3.54.

*Notation: * = repetition (0..n) ? = option (0 or 1) | = choice (a or b)*

HTTP Request

request ::= *request-line* (*general-header* | *entity-header* | *request-header*)*
<CR/LF> (*message-body*)?

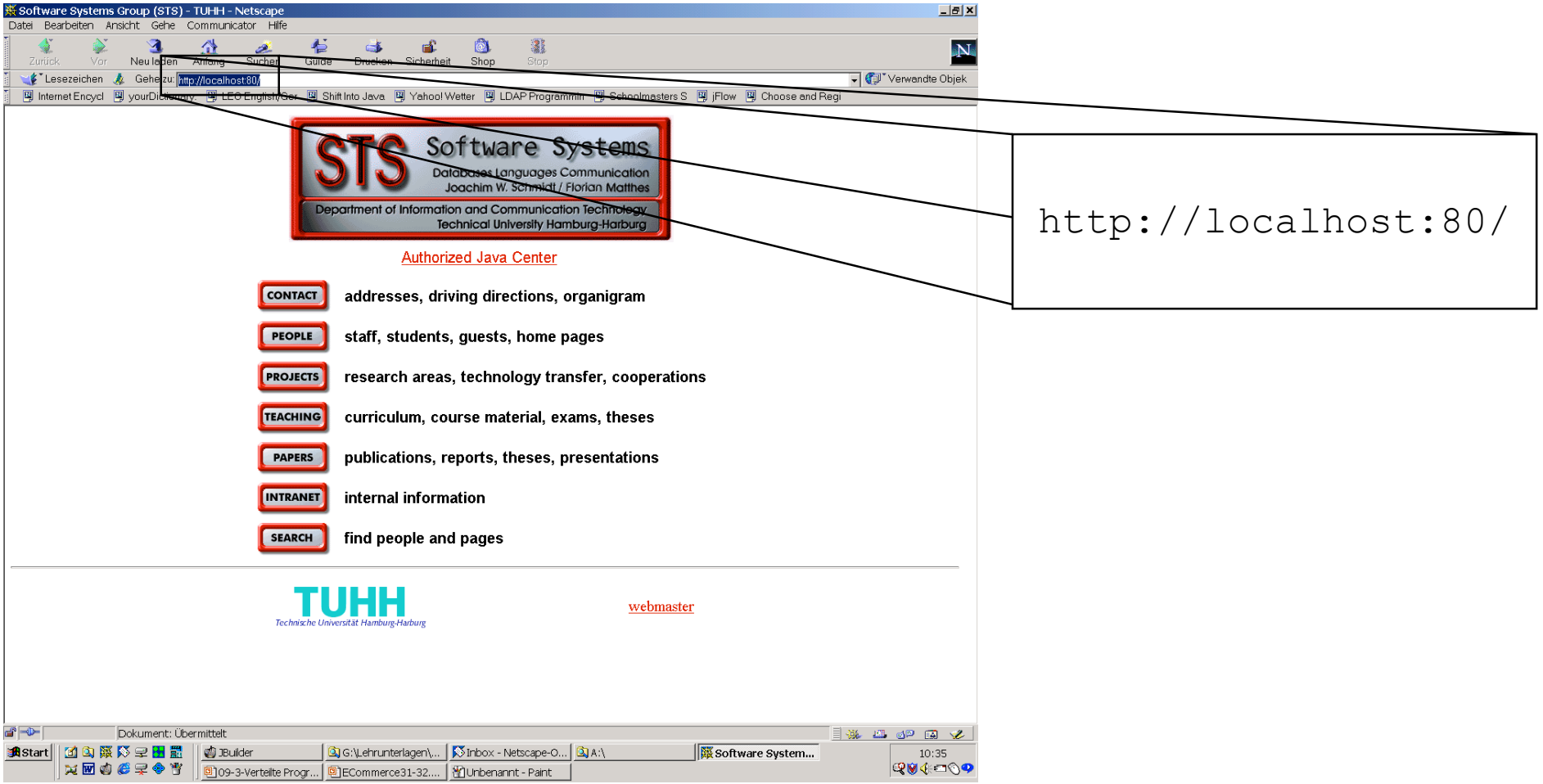
request-line ::= *request-method* <SPACE> URI <SPACE> http-version <CR/LF>

HTTP request methods:

- **GET:** Used by client to send a retrieve request for a document on a server.
- **HEAD:** Used by client to retrieve only the headers (meta-information), but not the document itself. As the headers contain an entry on the document‘s length, this method is useful for quality-based transfer decisions.
- **POST:** Used by client to send information to the server (usually information provided in a form). An alternative for sending information from client to server is using the GET method with URL rewriting.

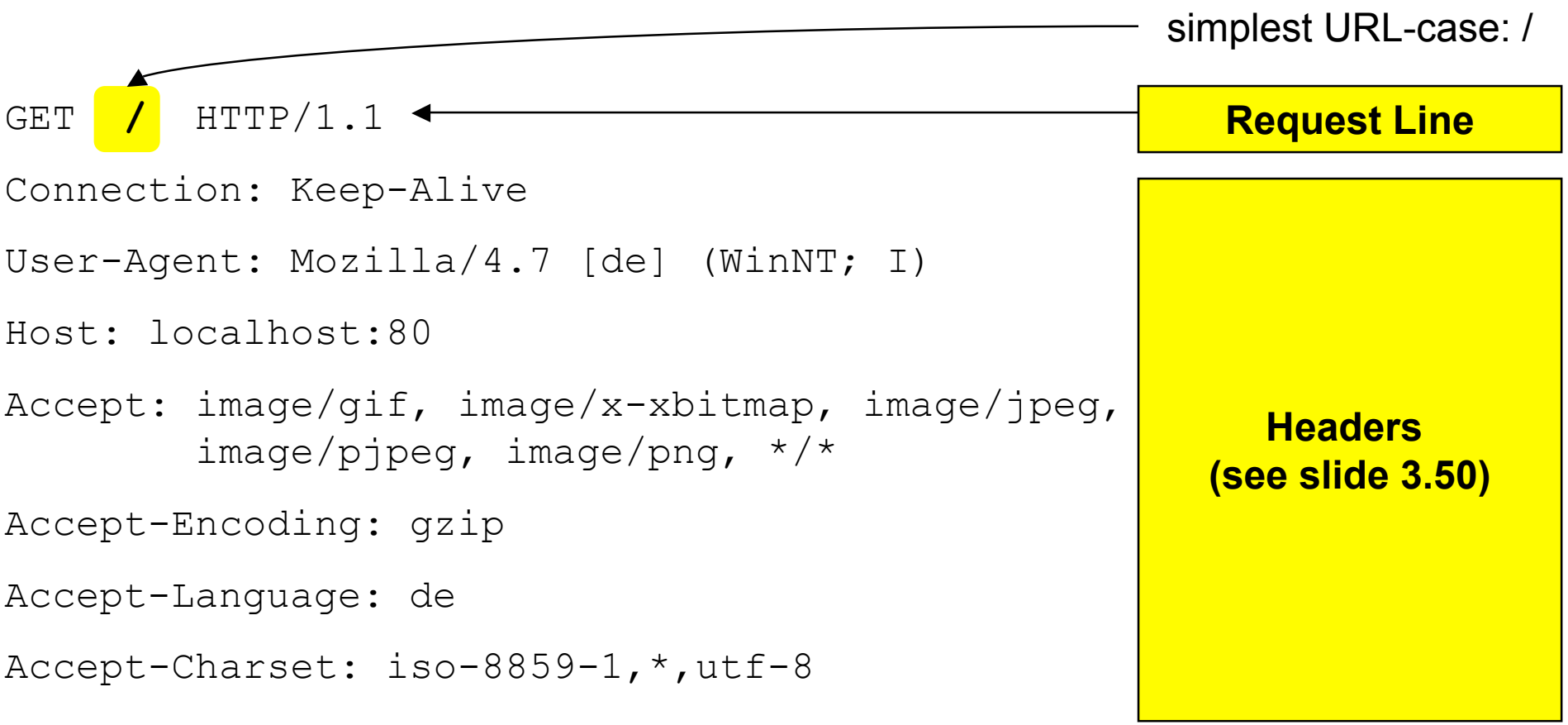
Note: TRACE, PUT, DELETE, and OPTIONS are not commonly used in ECommerce applications and are therefore skipped. Further reading [Wilde99].

GET Request: Application View

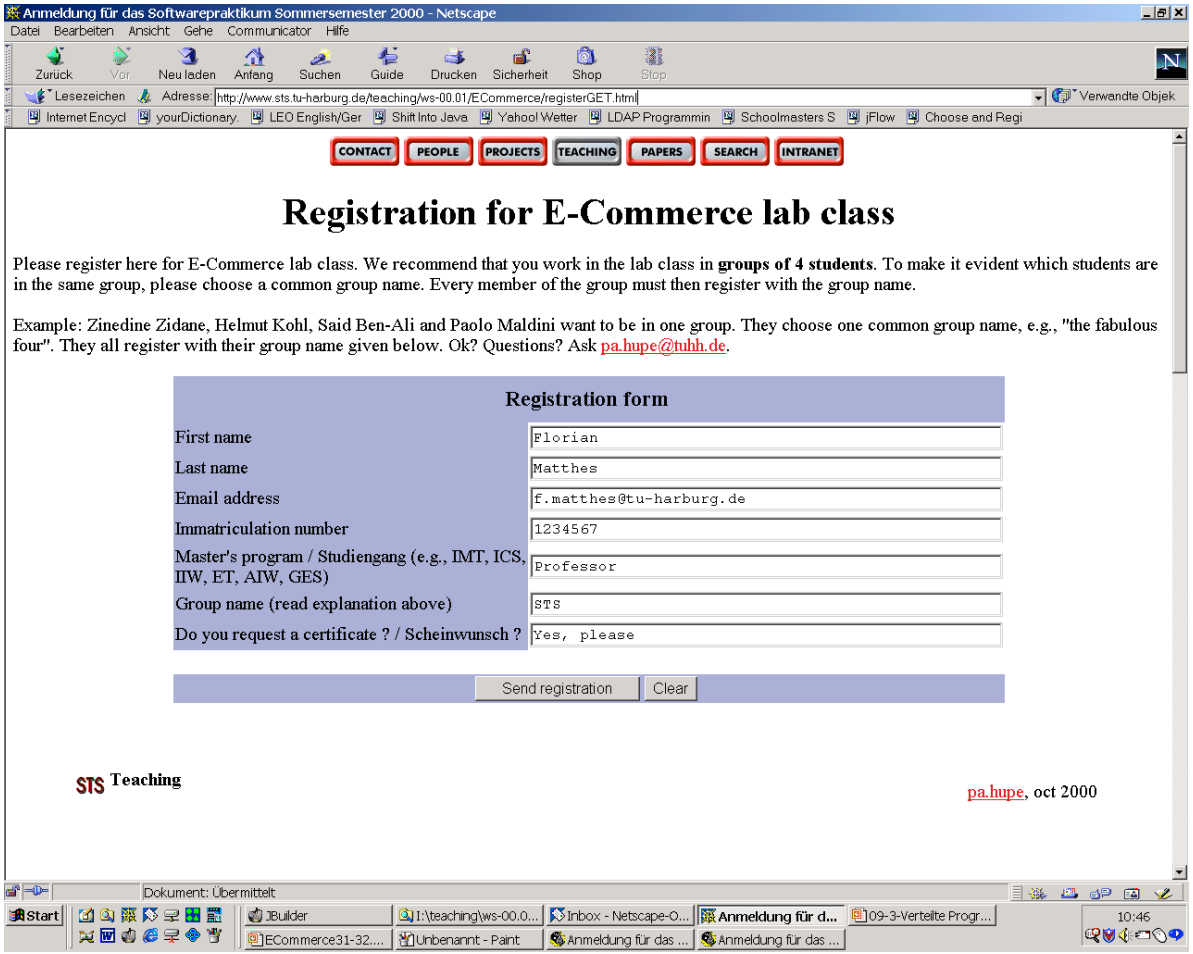


GET Request: Protocol View

Transferred request information



POST Request: Application View



POST Request: Protocol View: Header

```
POST / HTTP/1.1

Referer: http://www.sts.tu-harburg.de/
teaching/ws-02.03/ECommerce/
registerGET.html#danke

Connection: Keep-Alive

User-Agent: Mozilla/4.7 [de] (WinNT; I)

Host: localhost:80

Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg, image/png, */*

Accept-Encoding: gzip

Accept-Language: de

Accept-Charset: iso-8859-1,*,utf-8

Content-type: text/plain

Content-Disposition: inline; form-data
```

empty line here (CR/LF)

Request Line

Headers
(see slide 3.50)

POST Request: Protocol View: Message Body

```
nummer=EC02
vorname=Patrick
nachname=Hupe
email=pa.hupe@tu-harburg.de
matrikelnr=1234567
fach=Tutor
gruppe=STS
schein=Yes
```

Message-Body:
Name / Value Pairs

(see slide 3.50)

HTTP Response

response ::= *status-line* (*general-header* | *entity-header* | *response-header*)*

<CR/LF> (*message-body*)?

status-line ::= http-version <SPACE> status-code <SPACE> reason-phrase <CR/LF>

Purpose: Status report on protocol execution

The status-code is a three-digit number giving information about the operation (success, failure and reason).

The reason-phrase is a short phrase describing the status-code (typically in English).

Status-codes classes:

- 1xx Informational
- 2xx Successful
- 3xx Redirection
- 4xx Client Error
- 5xx Server Error

HTTP Useful Header Fields

Header fields useful in EC applications:

General header fields:

- *Via*: Helps trace-route messages through the internet.
- *Transfer-Encoding*: Used for compressing large message contents (gzip).

Entity header fields:

- *Content-Encoding*: See Transfer-Encoding. Defines that content is stored encoded at the server.
- *Content-Type*: Defines the content type (see MIME).
- *Expires*: Tells proxies and gateways when to clear cached copies.

Request header field:

- *Referer*: Tells the server the URL of the page that contained a link the client followed. Used for profiling users and accounting revenues in the *Affiliate Business Model*.

Response header field:

- *WWW-Authenticate*: Requests authentication from the client for a given realm.

header field definition: see slide 3.49

MIME (Multipurpose Internet Mail Extensions)

Allow email content to contain characters other than US-ASCII:

- Text written in languages with extended character set (german umlauts; japanese katakana, hiragana and kanji; chinese word symbols; ...)
- Binary attachments (graphics: GIF, JPEG; program files downloaded: EXE; archives: ZIP, TAR)
- Multi-part messages, e.g., used for attachments (images, documents) in HTTP POST requests

Part of HTTP since HTTP/1.0:

- specified by entity header field *ContentType:*

  Examples: *text/plain, image/gif, text/html, application/msword, text/wml.*

HTTP Content Negotiation (1)

Purpose: A resource can be available at a server in several different *variants*. Client and server should negotiate on which variant to retrieve.

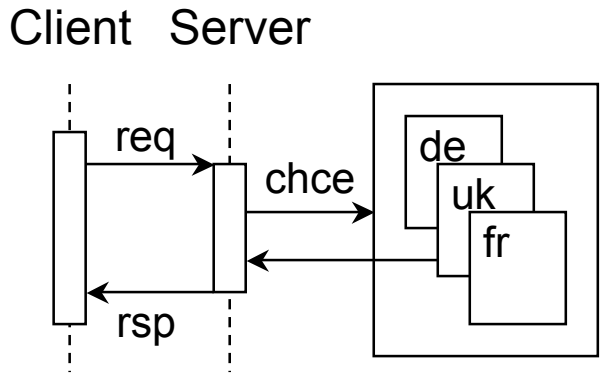
Variants follow three categories:

- Language (German, English, Korean, Bantu, ...)
- Quality (lo-res vs. hi-res, b/w vs. color images, abstract text vs. full paper)
- Encoding (character-set: US-ASCII, Unicode, ...)

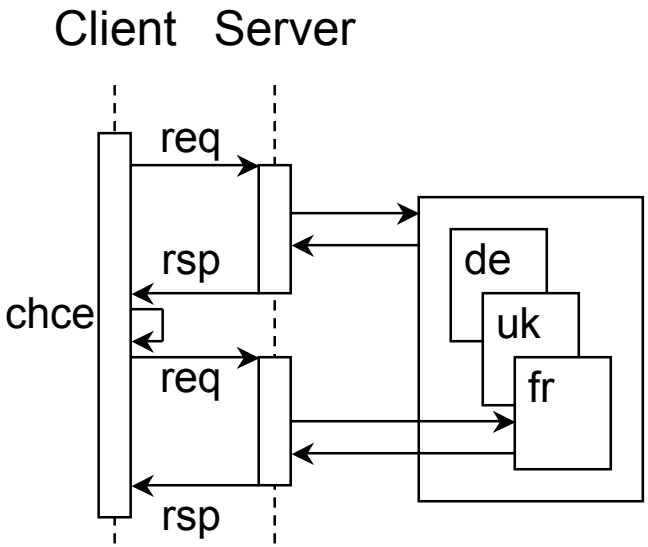
Clients and servers can negotiate these parameters following three strategies:

- Server-driven negotiation (The server „knows“ what is good for the client)
- Agent-driven negotiation (Client choice)
- Transparent negotiation (A proxy negotiates what is best for the client)

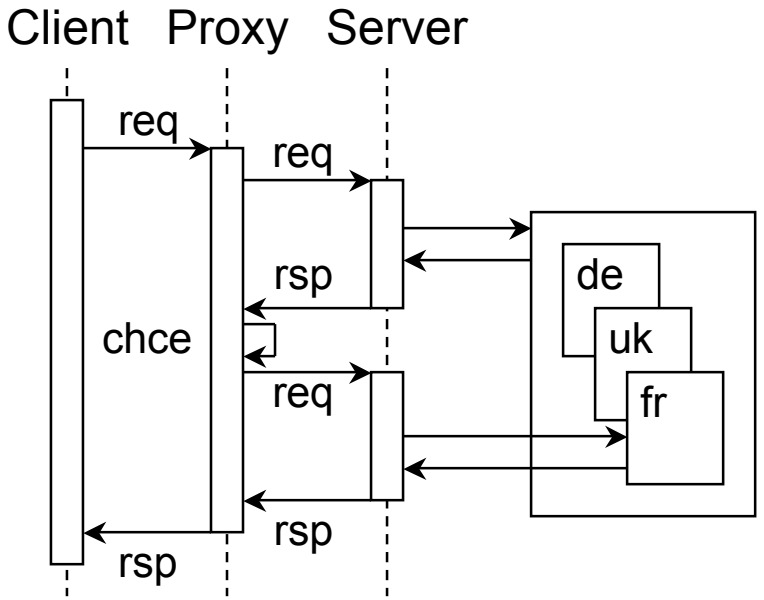
HTTP Content Negotiation (2)



a) server-driven negotiation



b) agent-driven negotiation



c) transparent negotiation

HTTP/0.9 and HTTP/1.0

HTTP/0.9:

+ First HTTP version
- Only GET request method supported
- No media types other than US-ASCII text allowed

HTTP/1.0:

+ Integration of MIME for media types other than US-ASCII text
+ POST request method added (clients can send data to the server; useful for forms)
+ Responses can contain status codes
+ Simple user authentification
- Virtual, non-IP based server hosts not supported (see end of chapter)

HTTP/1.1 (1)

Main design goal: Gaining performance with a better request/response-Interaction Model.

Problem: The TCP connection is closed after the server‘s response. Performance gains possible from TCP connections that remain open for several request/response cycles.

Choices:

- Persistent HTTP (P-HTTP): TCP connections are kept over (alive) between client and server.

  `Connection: Keep-Alive` ← Request header field

- HTTP over Transaction TCP (T/TCP): TCP connections are closed after each cycle, but overhead for reopening is reduced significantly.

Chosen Solution:

+ P-HTTP integrated into HTTP/1.1.

HTTP/1.1 (2)

Further improvements:

+ Virtual hosts supported via new header field HOST: Several servers can be made available through a single TCP address (IP-address:port combination). See chapter 3.6).

```
Host: eurift.sts.tu-harburg.de:80
Host: wips.sts.tu-harburg.de:80
```

Identical IP address (134.28.70.3) and port distinguished by HOST field

+ New request methods PUT, DELETE, TRACE, OPTIONS
+ Partial transmissions of resource entities (documents)
+ Content Negotiation (negotiation of visualization, language, quality, encoding)

```
Accept: image/gif, image/x-xbitmap, image/jpeg,
        image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: de
Accept-Charset: iso-8859-1,*,utf-8
```

+ Improved Authentication

HTTP Extensions

- Session Management: Adding session management to the stateless HTTP request/response protocol
- User identification and authentication
- Security: Adding layers for securing HTTP: S/HTTP, HTTPS, IPSEC
- Refresh / Redirect: Trigger actions at the client browser
- more...

HTTP Sessions Management

Problem:

- HTTP is a stateless protocol. How can a server track a session, i.e., relate several HTTP request to a single client and manage session state?

Solutions:

- **URL rewriting**: All otherwise unrelated HTTP request are assigned a unique identification number (ID), e.g., 000313. Before sending a web page to a client, all links within the page that do not point to a different server are extended by an ID as parameter. When the client uses this link, the ID will be sent back as a parameter to the server.

  Example:

  Link www.sts.tu-harburg.de/teaching is changed to www.sts.tu-harburg.de/teaching?id=000313.

- **Cookies**: Server stores ID information (cookie) on the client's computer. Every time the client connects to the server, the client adds this information to the request.

Identification, Authentication and Authorization

Assure a subject is who he/she/it claims to be and yield him/her/it rights accordingly.

- **Identification:** Identify a subject (a human, a user agent) by an identification (usually a user name). Whenever the identification is used, assume it represents the subject.
- **Authentication:** Make sure the subject is who he/she claims to be, usually by requesting a password. Alternatives are: smartcard, biometrics (iris-scan, fingerprint, voiceprint, etc.).
- **Authorization:** Grant an identified subject rights (accessing, altering, deleting documents).

Further reading: [Schm98]

HTTP Authentication (1)

Basic Authentication (since HTTP/1.0):

- Identification and password are transmitted as plain text. **THIS IS INSECURE!**

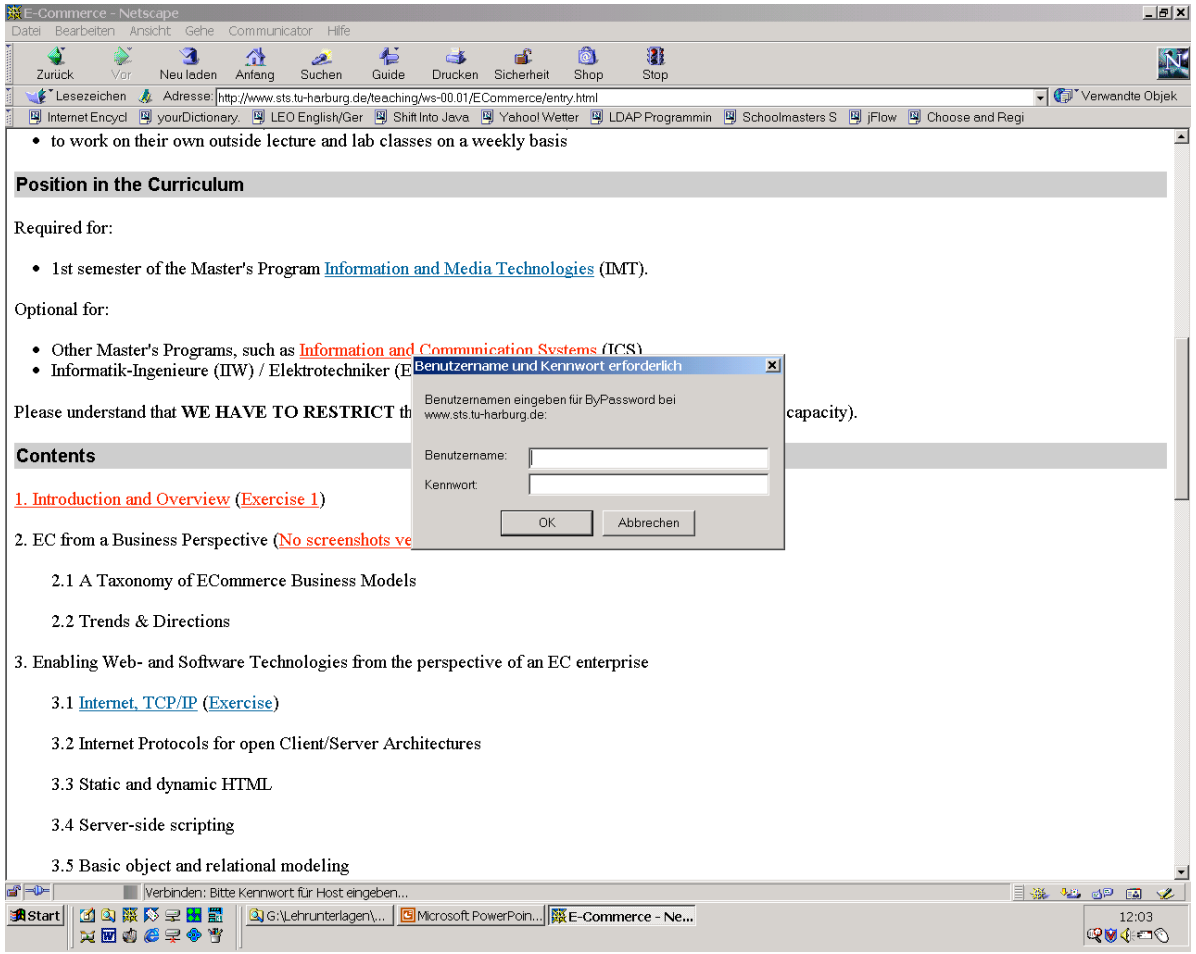
Digest Access Authentication (since HTTP/1.1):

- Client encrypts identification and password using a one-way function and sends this *digest* to the server. Server performs same computation and compares results.

HTTP Authentication (2)

Drawbacks:

- Does not fit into page design
- Cannot be visualized following the company's corporate design
- Language cannot be selected



HTTP Authentication (3)

Security problem not yet solved:

- Replay attack: “Man in the middle-attack”: Attacker copies authentication message and replays it to the server, this will authenticate him.

Possible solution: Authentication message is only valid one time. For this, encrypt the following items (and combinations thereof) into the client request:

- **Server-generated *nonce (nonce = “number, generated once”)***
- Client IP-address
- Timestamp
- Identification
- Password
- Request method
- Requested URI
- ...

HTTP Security

Secure data transmission is not only relevant to authentication, but also to data transmission.

Three standards exist in different network layers (see slides 3.71, 3.74, 3.75):

- HTTPS (Hypertext Transport Protocol over Secure Socket Layer, SSL)
- SHTTP (Secure HTTP)
- IPSEC (Secure IP protocol)

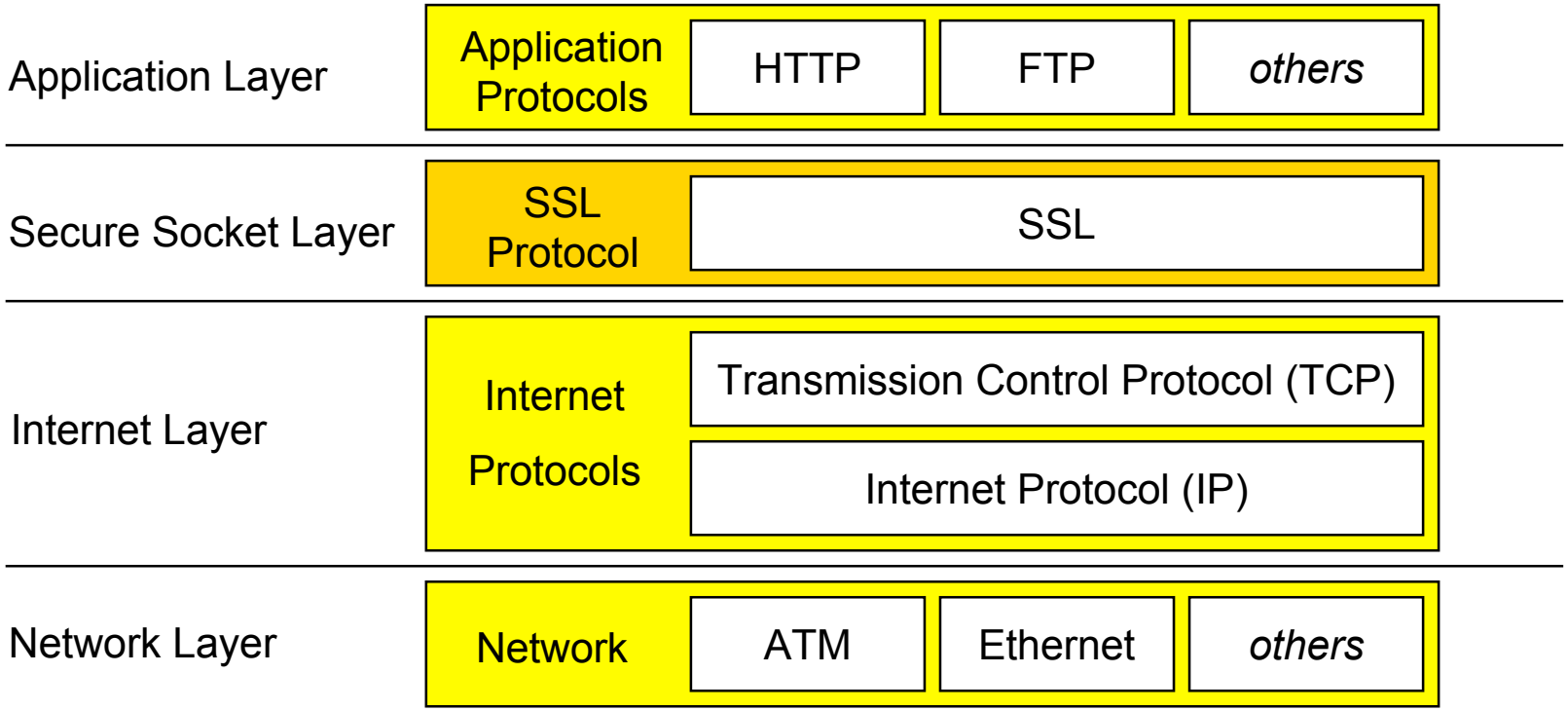
HTTPS is the standard used on the web.

SHTTP offers more functionality, but is not widely supported.

As with IPv6, IPSEC will be included. It could then replace HTTPS.

HTTPS (HTTP over SSL) (1)

Secure Socket Layer (SSL) uses the insecure Internet layer to create a secure transport layer for applications. It is the leading security protocol on the Internet. When an SSL session is started, the browser sends its public key to the server so that the server can securely send a secret key to the browser. The browser and server exchange data via secret key encryption during that session (see chapter 5).



HTTPS (HTTP over SSL) (2)

SSL design goals (ordered by importance):

- Crypto security
- Interoperability
- Extensibility
- Relative usability (session caching avoids recomputation of keys)

SSL properties:

- Secure and reliable connection (at different security levels)
- Optional authentication

Problem:

- SSL works only with TCP (e.g., not with the UDP protocol). Application services that use UDP (e.g., telnet, ...) cannot use SSL.

HTTPS uses *https:* for protocol prefix. Example: https://www.sts.tu-harburg.de/

HTTPS (HTTP over SSL) (3)

Implications for web clients (browsers): SSL must be integrated into browser. All major browsers support SSL v2 / v3:

- Netscape Navigator,
- Internet Explorer,
- Opera
- ...

Implications for web servers:

- SSL module must be integrated into web server
- Must support SSL v2 and SSL v3

Key length for encryption can be 40, 56, 128 bit

- Defined by server side (Example: bank: strong encryption)
- Restricted by client browser (versions which support only 40 / 56 bit encryption)

For connection between key length and encryption strength, see chapter 5.

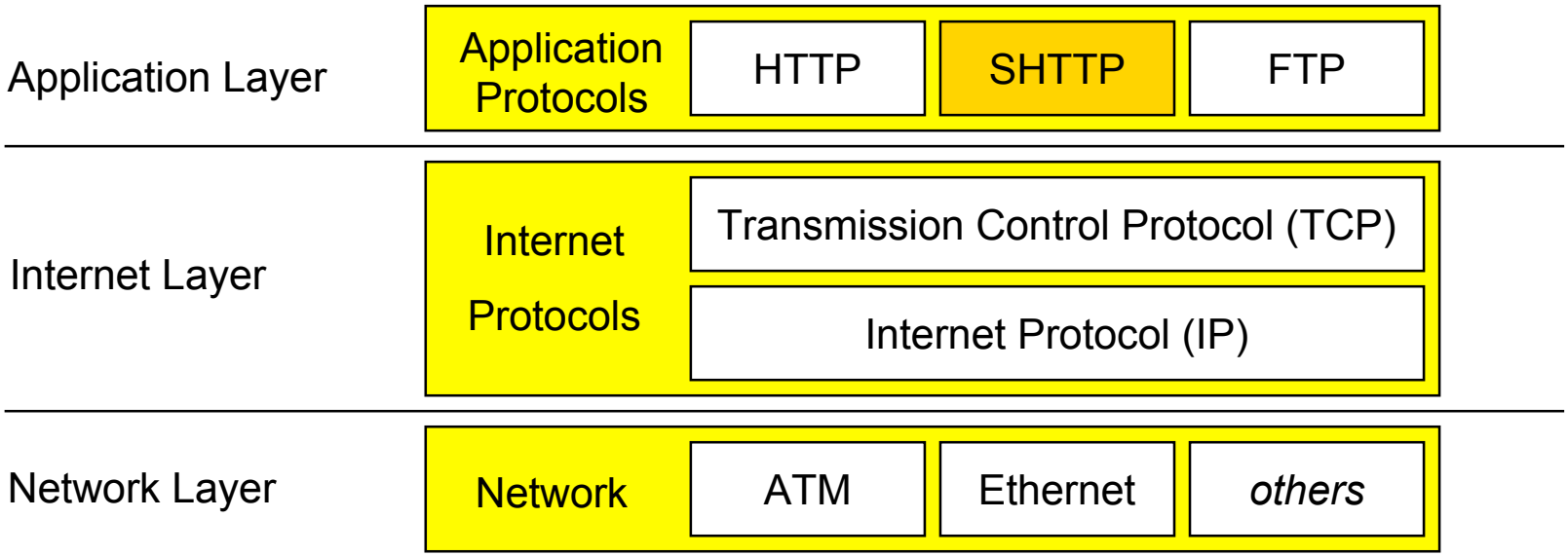
SHTTP (Secure HTTP)

SHTTP is an alternative to HTTPS. Endorsed by National Center for Supercomputing Applications (NCSA) and a variety of organizations.

Primary design goal: Secure encapsulation of HTTP messages.

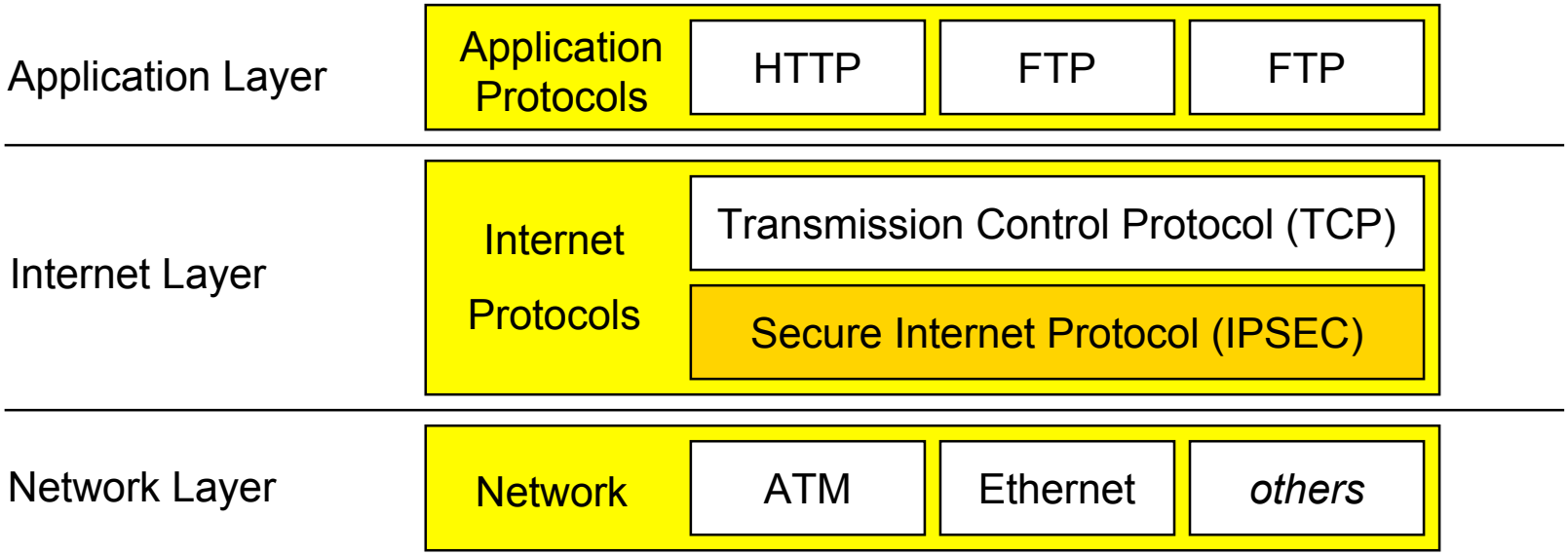
Problem:

- SSL works with adapted HTTP (not with TCP or UDP). Other application services cannot use this.



IPSEC (Secure IP)

Purpose: Allows data encryption at the IP level. All application services can use IPSEC for encryption. Therefore, IPSEC is well-suited for building Virtual Private Networks (VPN). For VPN, see chapter 3.7.



Additional HTTP Extensions

Header fields that are not part of the HTTP/1.1 standard:

- **Refresh** (reload web page) and

  Redirect (switch to different web page after a certain time interval):

```
<META http-equiv="refresh" content="30"
      URL="http://www.sts.tu-harburg.de/">
```

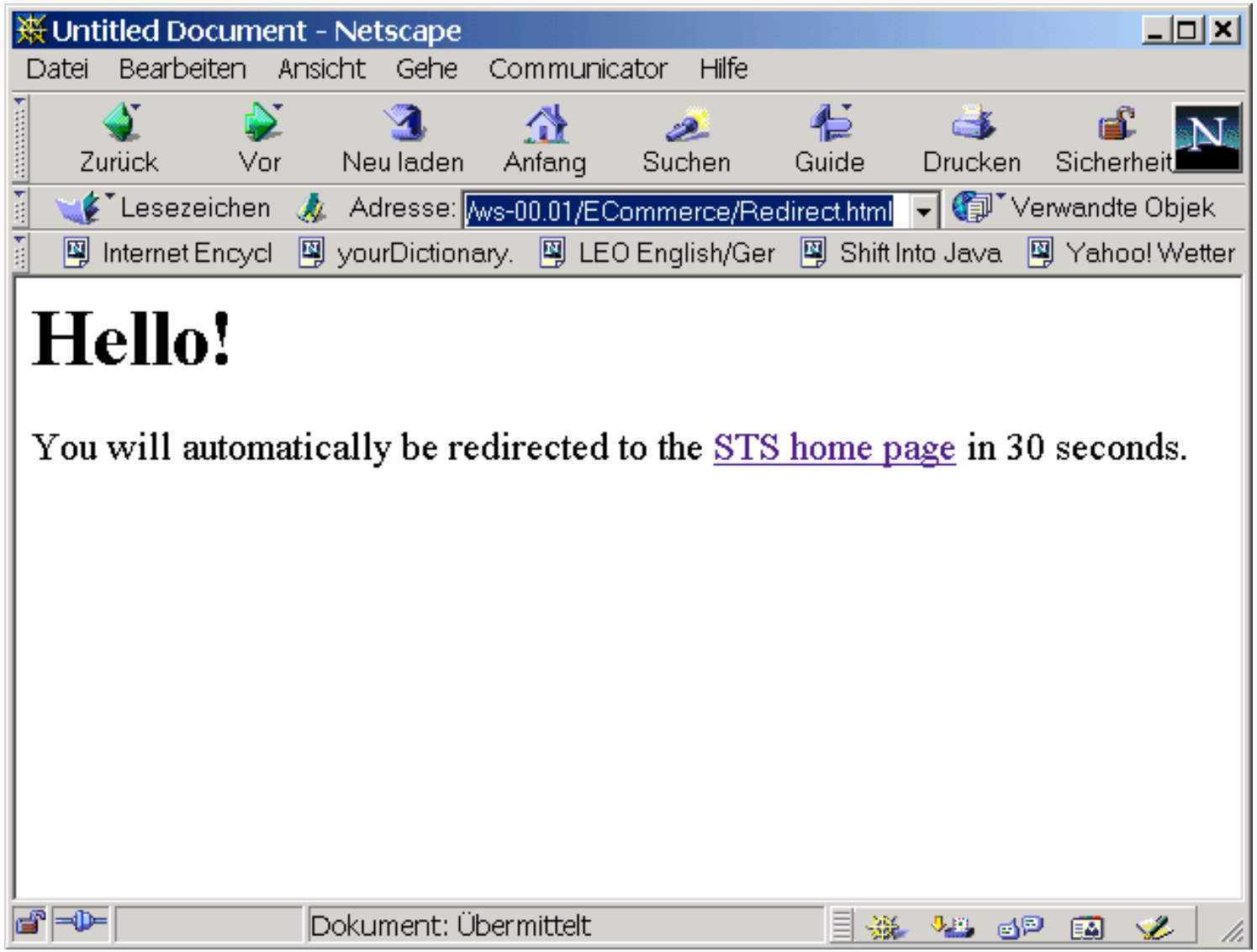
- Visualizations of web page transitions (Microsoft-specific)

  Page enter / Page exit / Site enter / Site exit

Redirection

Redirection example.

Usually, the page also contains a link to the page for browsers that do not support the *redirect* meta tag.

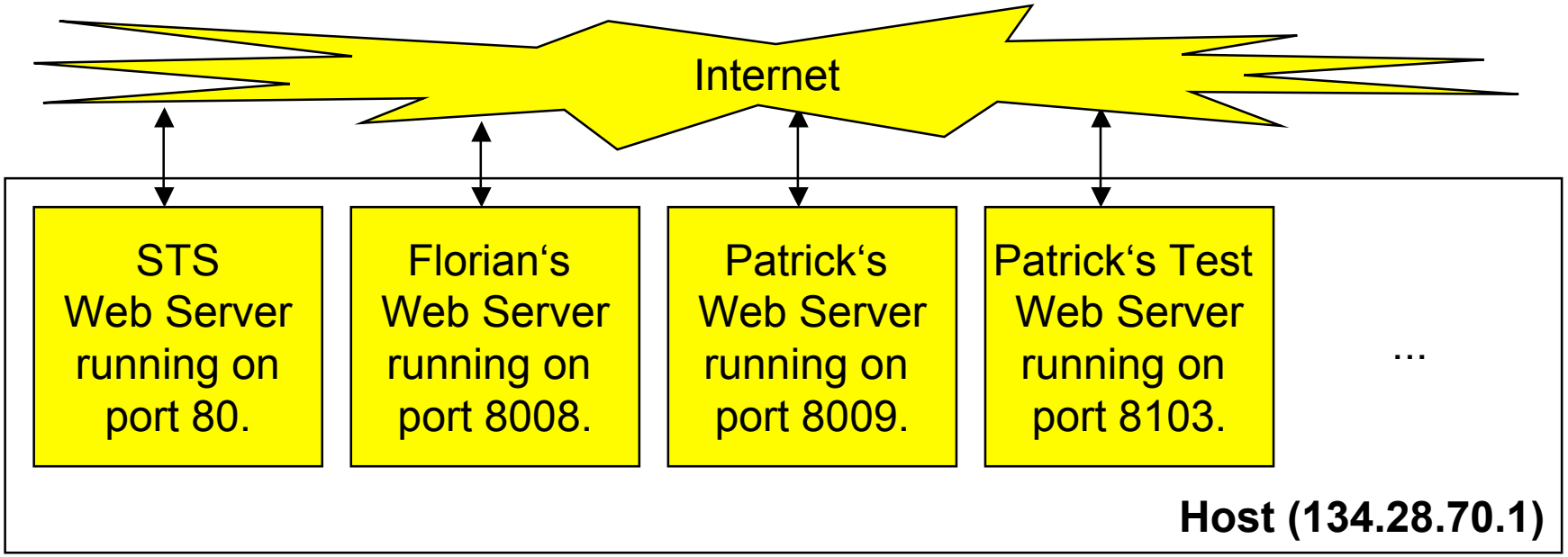


Specific Internet Services: Web Server

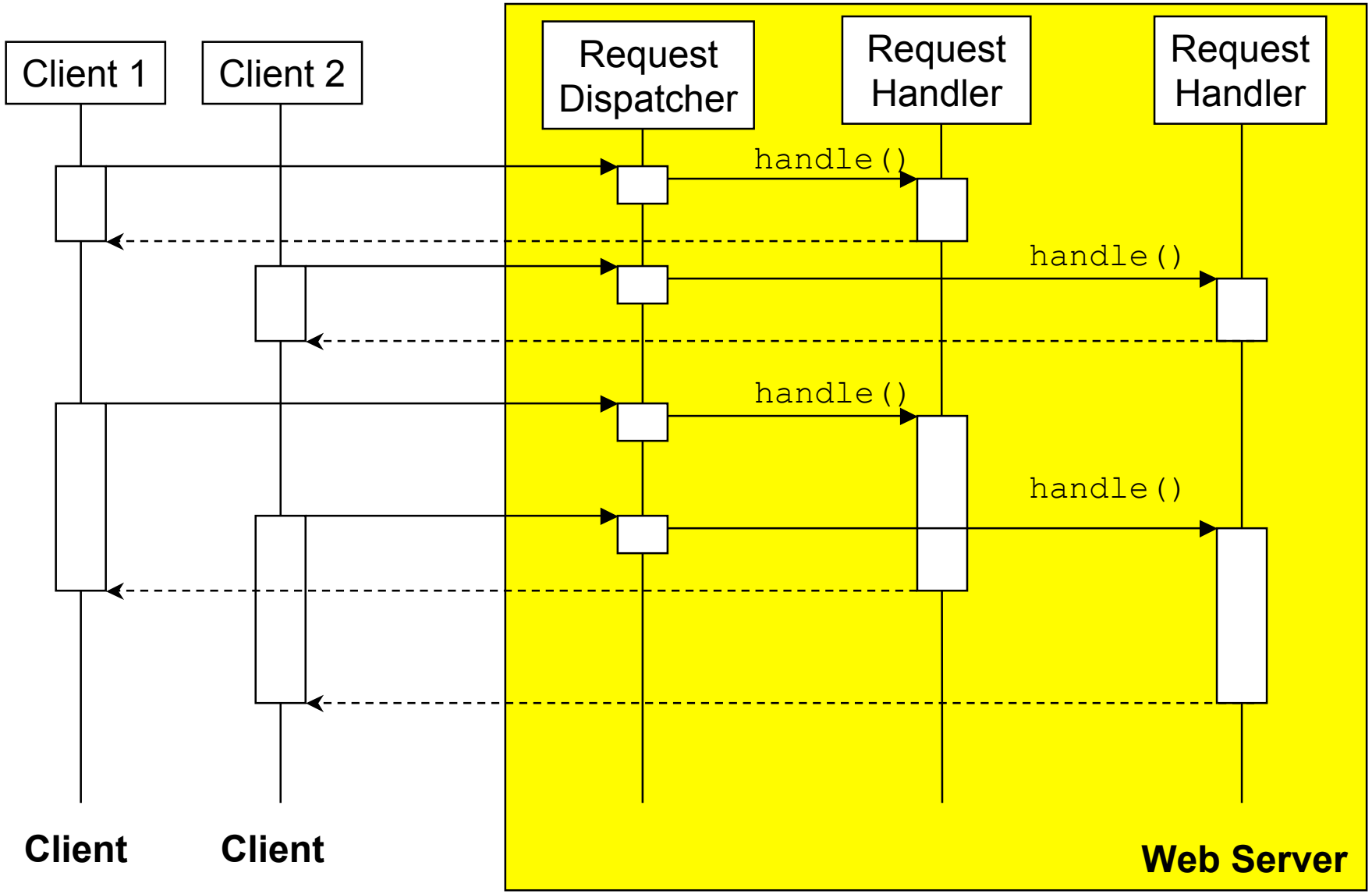
Definition

A **Web server** is a server that provides access to resources via the HTTP protocol.

A web server is *implemented* as a *service running on the host machine*. As a web server is only a single process, several web servers can be run on one computer (using different ports).



Web Server: Dispatching



Web Server: Request Handling

The request handler performs only some very simple actions:

```
handle() {
        /* retrieve the path from the request */
      path = retrieveResourcePath();
        /* read the content and write back to the client */
      content = readContent( path );
      writeToSocket( content );
        /* close the socket */
      closeSocket();
}
```

Pseudo code, not Java!