university of koblenz
Computer Science

4. Communication (Part I)

Engineering Web and Data-intensive Systems

Dr. Volker Riediger - Winter Term 2022/23

ist *Institute for Software Technology*

Communication (Part I)

- Network basics
- HTTP
- Sessions
- Scaling and Load Balancing

Image: colourbox.de

Chapter 4. Communication

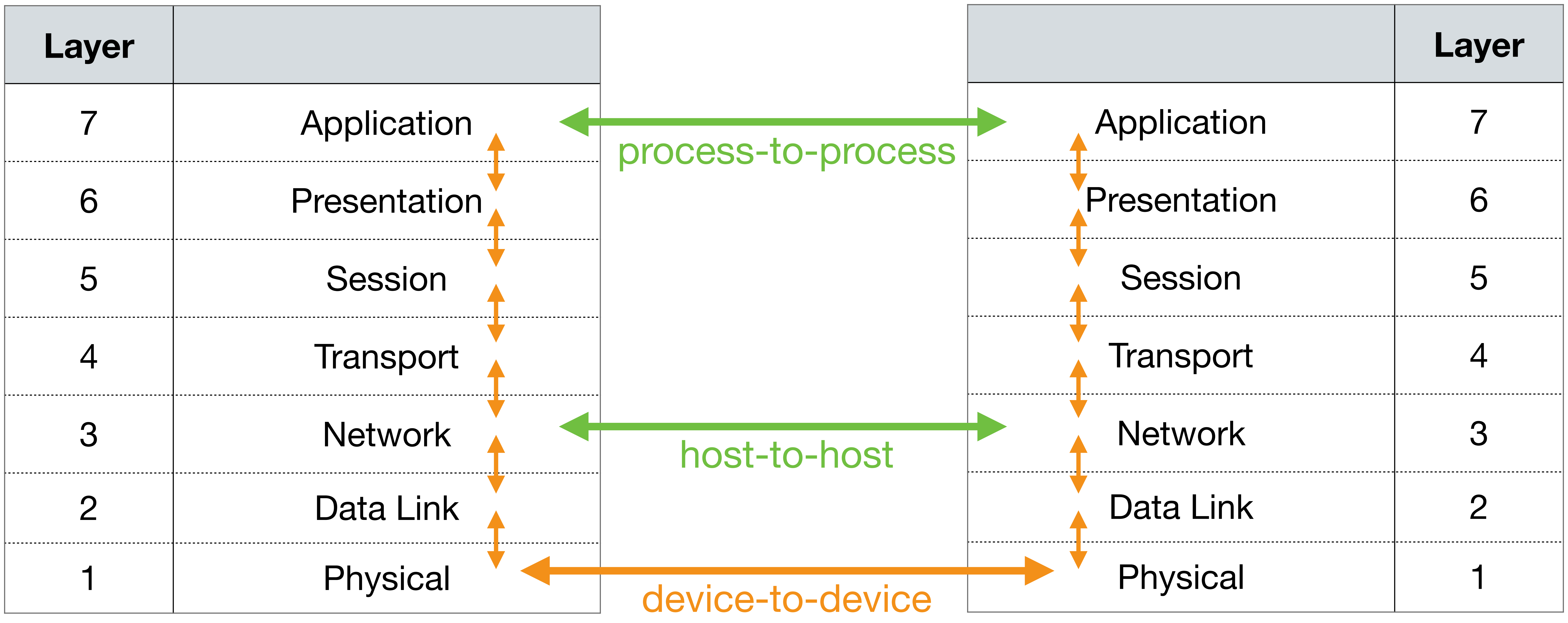
4.1 Network Communication

Network Communication

- To enable communication between network nodes, models, architecture, rules, logical, and physical properties have to be agreed upon by the communicating parties
- Reference architecture defined in the Open Systems Interconnection (OSI) project conducted by the International Standard Organization (ISO)
- Many implementations emerged for different communication scenarios, e.g.
- TCP/IP - the internet protocol, transmission control protocol
- IPX/SPX - Internetwork/Sequenced Packet Exchange
- SNA - IBM Systems Network Architecture
- UMTS - Universal Mobile Telecommunications System
- Various different physical transfer technologies exist

➡ Need to hide the underlying complexity from the communicating parties

OSI Reference Model



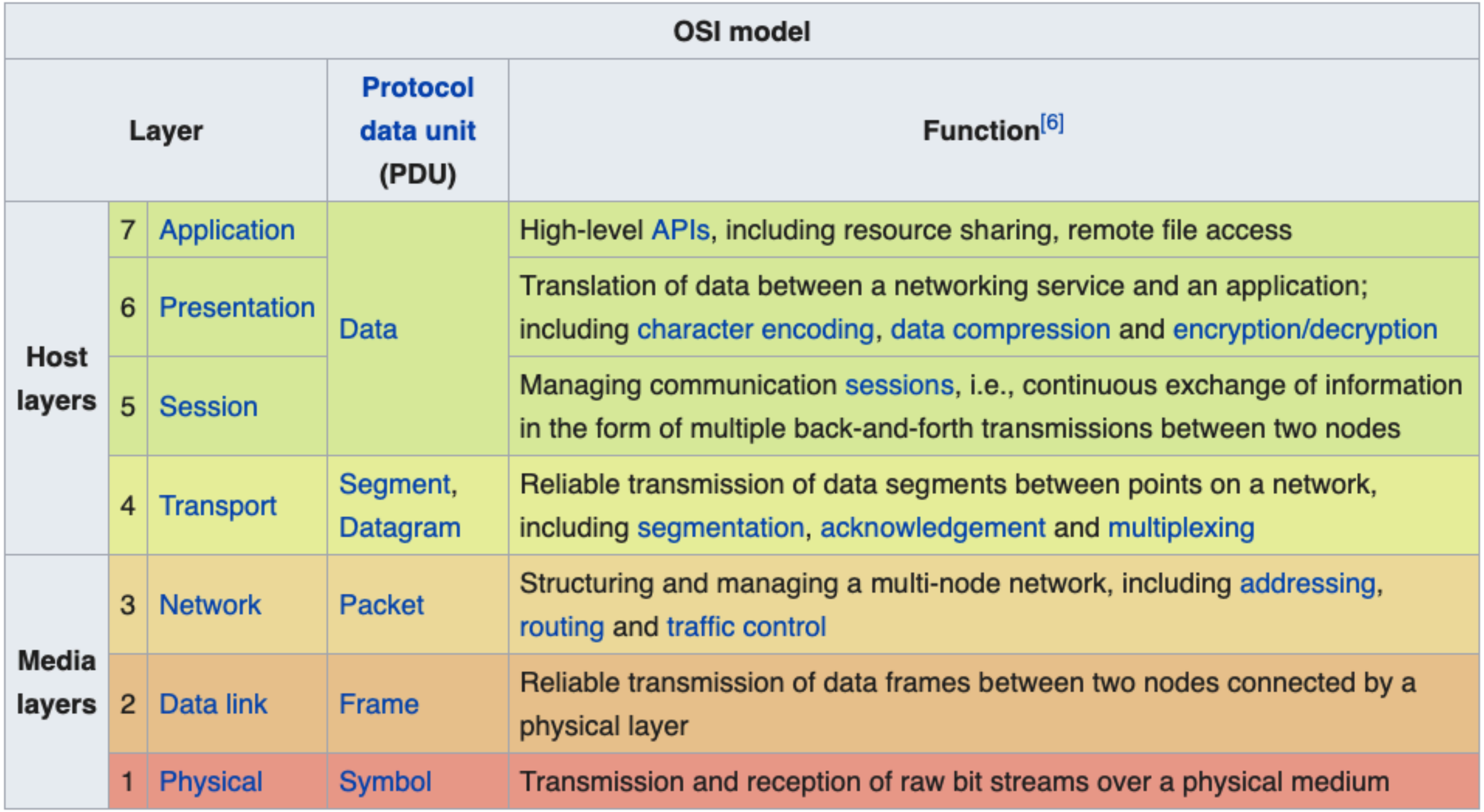
Layer	
7	Application
6	Presentation
5	Session
4	Transport
3	Network
2	Data Link
1	Physical

	Layer
Application	7
Presentation	6
Session	5
Transport	4
Network	3
Data Link	2
Physical	1

↔ logical communication

↔ physical communication

OSI Layers

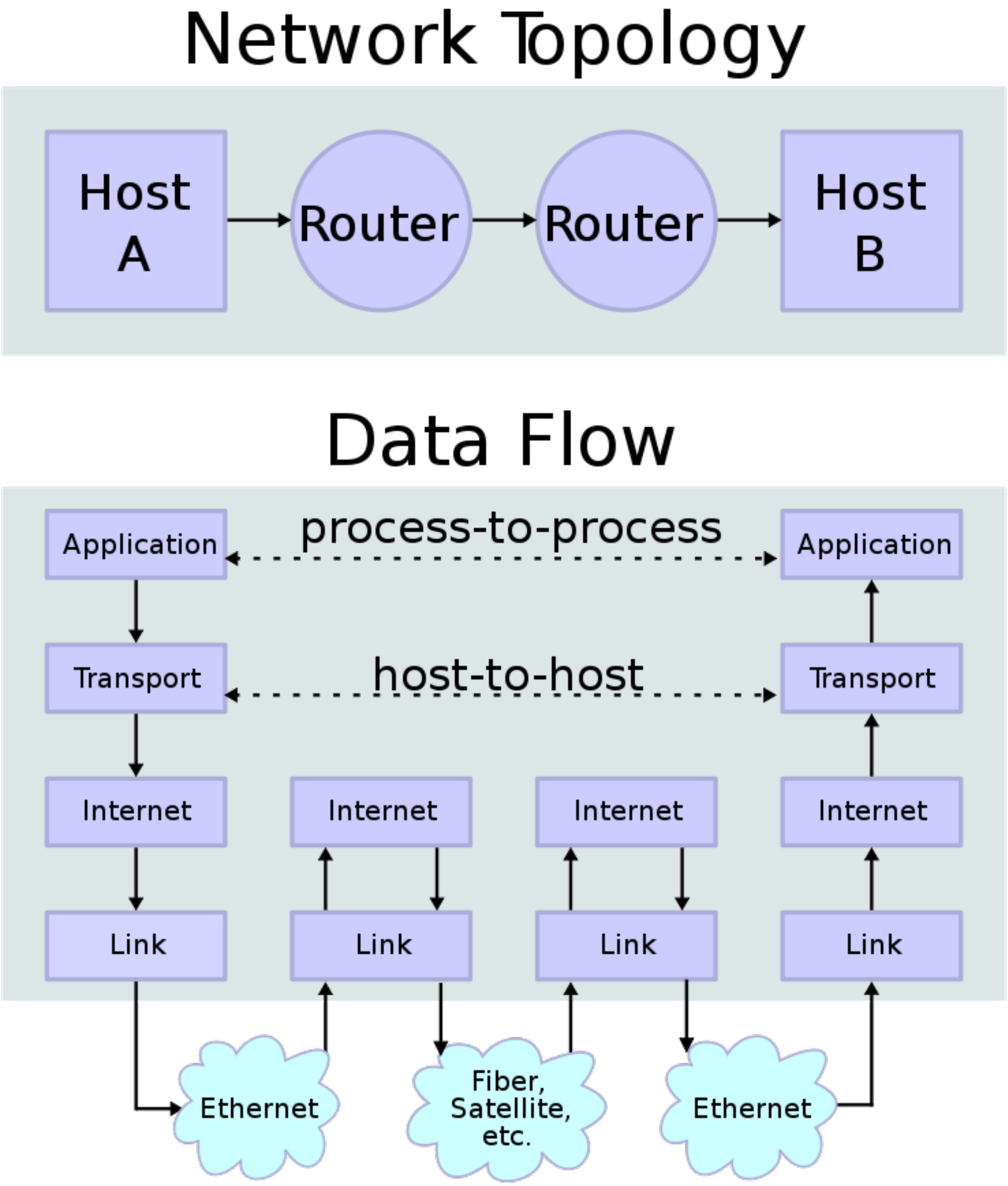


OSI model				
Layer			Protocol data unit (PDU)	Function[6]
Host layers	7	Application	Data	High-level APIs, including resource sharing, remote file access
	6	Presentation		Translation of data between a networking service and an application; including character encoding, data compression and encryption/decryption
	5	Session		Managing communication sessions, i.e., continuous exchange of information in the form of multiple back-and-forth transmissions between two nodes
	4	Transport	Segment, Datagram	Reliable transmission of data segments between points on a network, including segmentation, acknowledgement and multiplexing
Media layers	3	Network	Packet	Structuring and managing a multi-node network, including addressing, routing and traffic control
	2	Data link	Frame	Reliable transmission of data frames between two nodes connected by a physical layer
	1	Physical	Symbol	Transmission and reception of raw bit streams over a physical medium

from [https://en.wikipedia.org/wiki/OSI_model]

TCP/IP

- Layered Network Protocol Scheme
- Layers relate to, but not exactly match, the OSI layers
- Offers 2 Protocols:
  - UDP User Datagram Protocol connectionless best effort transfer
  - TCP Connection-based bi-directional reliable transfer
- IP = Internet Protocol
  - Packet transfer between nodes
  - Address schemes
  - Routing services



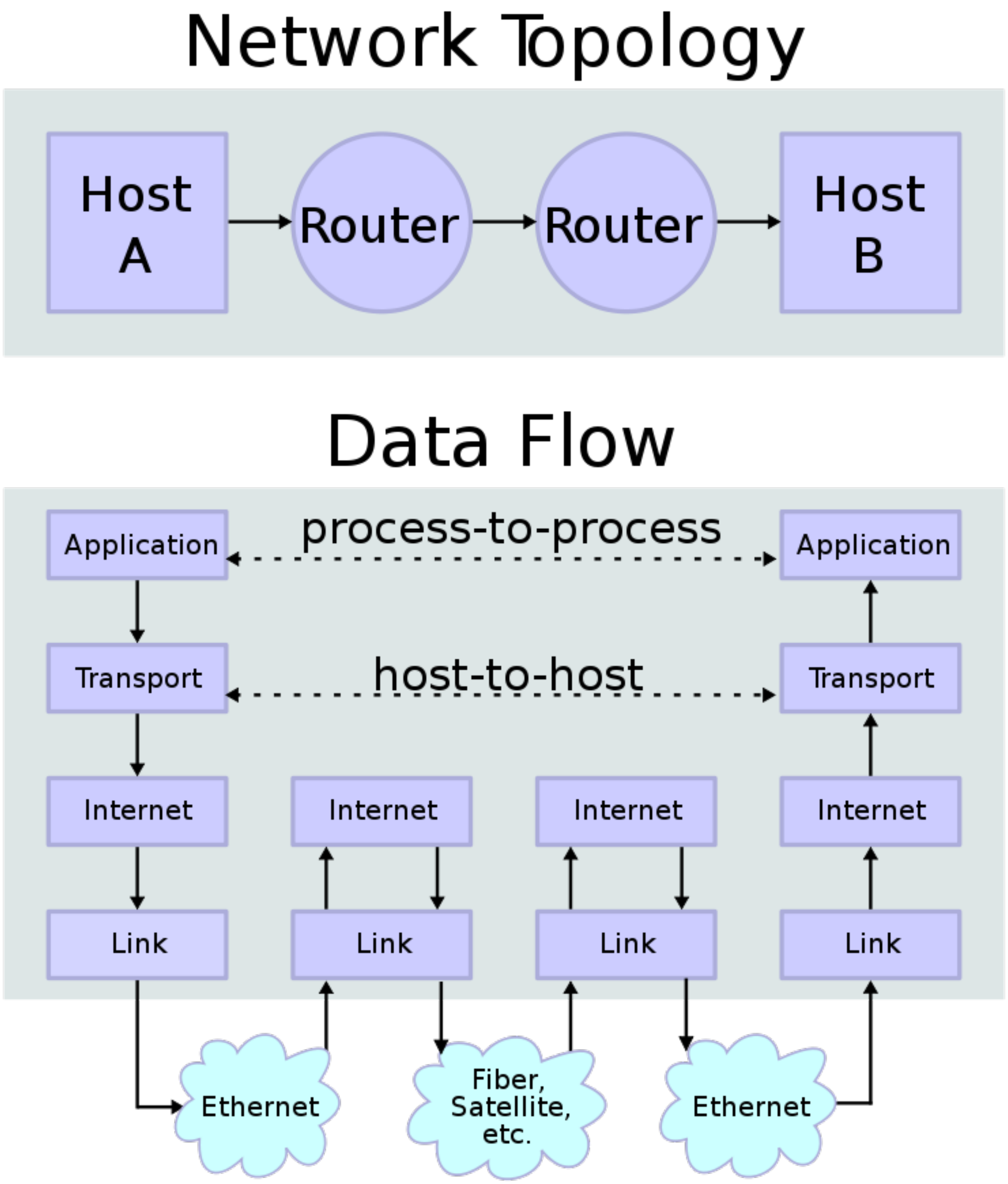
[https://en.wikipedia.org/wiki/Internet_protocol_suite]

TCP/IP

- On top of TCP, many application protocols are defined, e.g.

HTTP	80 / 443
SMTP	25 / 465
POP	110 / 965
IMAP	143 / 993
FTP	20,22 / 989,990
SSH	22
…	

- Protocols define well-known ports (part of UDP and TCP) to identify endpoints
- Registry managed by IANA (Internet Assigned Numbers Authority)



[https://en.wikipedia.org/wiki/Internet_protocol_suite]

4.2 HTTP - Hyper Text Transfer Protocol

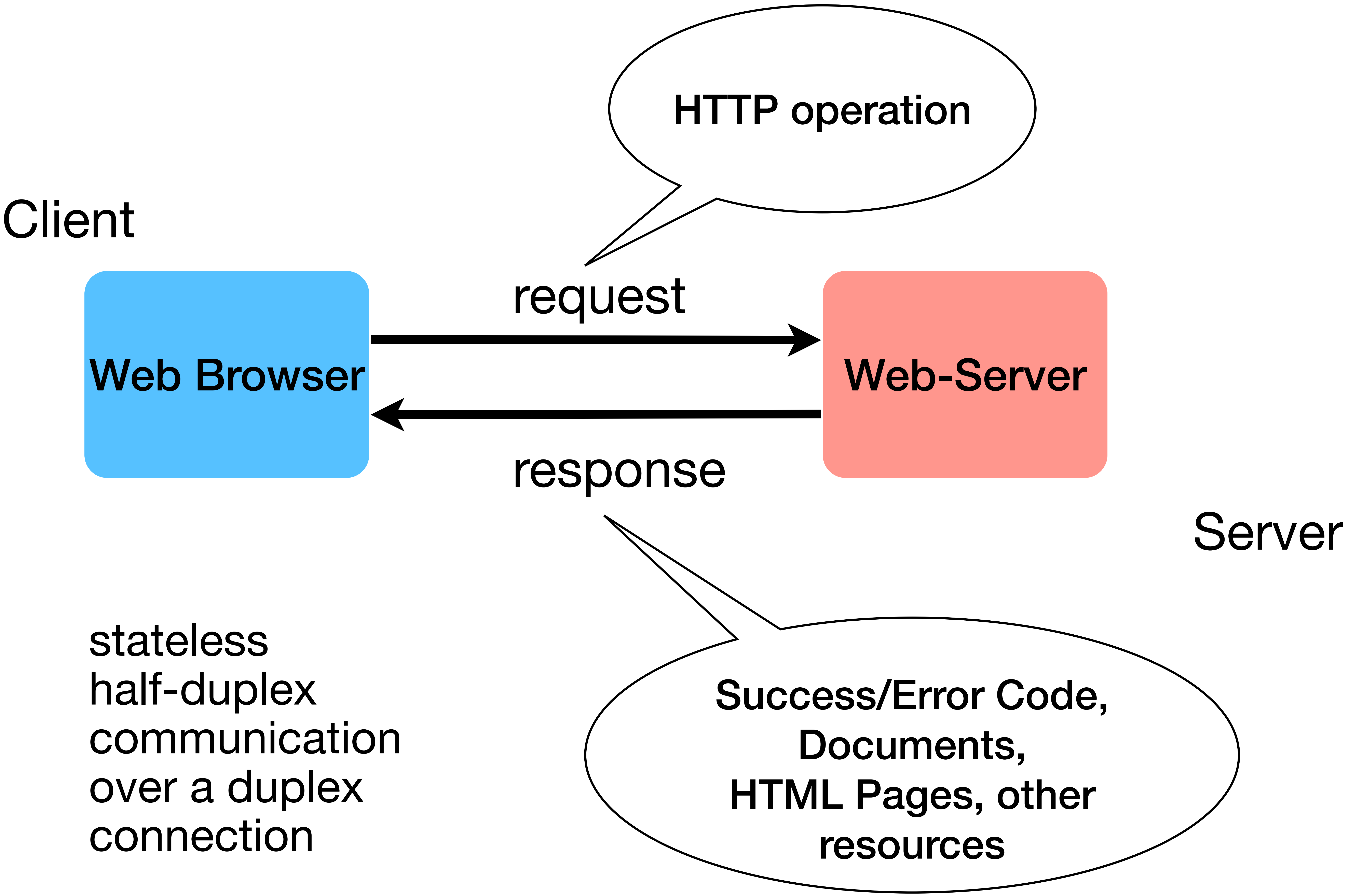
Definitions

A **Web Server** is a program that delivers web pages using the Web using the HTTP protocol.

A **Web Browser** is a program that supports rendering of, and interaction with web pages written in HTML.

A **protocol** is a formal description of message formats and rules for exchanging those messages.

Web Application: Client and Server



stateless
half-duplex
communication
over a duplex
connection

HTTP

- Communication in the Web is done using the Hypertext Transfer Protocol (HTTP), also developed by Tim Berners-Lee.

- HTTP is a simple protocol on the application layer of the Internet, supporting the exchange of documents between the client and the server.

- HTTP is a coordinated result of IETF and W3C.

HTTP in Web Applications

- transfer static content
  - text
  - stylesheets
  - pictures
  - …
- transfer dynamic content depending on system state, e.g.
  - database content
  - time
  - user identity
  - …
- provide web services
  - call business functions
  - modify data
  - …

HTTP Versions

Year	Version	Standard	Features
1991	0.9	-	early WWW prototype
1996	1.0	RFC 1945	additional methods and header fields
1997	1.1	RFC 7231	Multiple subsequent requests, CONNECT method
2015	2.0	RFC 7540	request multiplexing i.e. multiple parallel requests, servers may push resources
2018	3.0	upcoming, draft status as of Dec 2019	support UDP Transport

HTTP operations (methods)

- Method and its parameters are specified in the request header
- `GET` loads a resource from a web server
- `HEAD` loads meta information for a resource
- `OPTIONS` list of allowed methods for a resource
- `TRACE` echoes the command

these so-called **safe methods** are assumed to be read-only and not to modify content

- `POST` resource specific processing of the payload, can also create
- `PUT` replaces/creates a resource on the web server
- `DELETE` removes a resource from the server
- `CONNECT` (1.1) builds a tunnel when using proxies
- `PATCH` (2.0) modifies (parts of) a resource
- …and many more, see also: List of HTTP methods

Form based Web Applications

BROWSER

WEB SERVER

1. **Request** of a page (GET)

2. **Delivery** of HTML document

3. **Request** for further resources (GET)
   e.g. images, scripts, styles…

4. **Delivery** of requested data
   steps 3. and 4. possibly repeated many times

5. **Presentation** of the complete web page
   (rendering)

6. **Send** form data (POST)

7. **Processing** of form data, then delivery of
   response document
   (continue at step 2)

repeated for each subsequent action

Please install curl for your operating system!
Conduct your own experiments with curl.
You'll have to use curl in the persistence assignments!

Examples using curl and Browser Tools

Single Page Web Applications

WEB BROWSER

1. **Request** of a page (GET)

WEB SERVER

2. **Delivery** of HTML document

3. **Request** for further resources (GET)
   e.g. images, scripts, styles…

4. **Delivery** of requested data
   steps 3. and 4. possibly repeated many times

5. **Presentation** of the complete web page
   (rendering)

6. **Transmission** of a so-called
   „AJAX-Request“ (GET/POST/PUT/DELETE)

7. **Processing** of request,
   response as XML or JSON document, „pure data“

8. Processing of response,
   modification of the page,
   continue at step 6.

repeated for each subsequent acion

Example: RESToku

HTTP Headers

- Request and response start with the HTTP protocol header
- **Request header:**
  - Method, URL path, and protocol version
  - Hostname (of the server)
  - optional information, e.g. content length, content encoding, cookies, accepted content type
  - a blank line
  - (after that, content may be transferred)

Example requesting https://www.uni-koblenz,de
(The > and < marks are not part of the protocol but added by the cURL program)

```
> GET / HTTP/1.1
> Host: www.uni-koblenz.de
> User-Agent: curl/7.64.1
> Accept: */*
>
```

In this case, the server responds with a so-called redirect message, a web browser would then request the document indicated by the Location header. The response additionally contains human-readable HTML for browsers that don’t automatically redirect.

```
< HTTP/1.1 301 Moved Permanently
< Date: Thu, 16 Jan 2020 12:06:13 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Location: https://www.uni-koblenz-landau.de/de/koblenz/
< Content-Length: 338
< Content-Type: text/html; charset=iso-8859-1
<
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
<p>The document has moved <a href="https://www.uni-koblenz-
landau.de/de/koblenz/">here</a>.</p>
<hr>
<address>Apache/2.4.18 (Ubuntu) Server at www.uni-koblenz.de
Port 443</address>
</body></html>
```

HTTP Headers

- Request and response start with a HTTP protocol header

- **Response header:**
  - Protocol version, status code, status description
  - Timestamp
  - status code specific information
  - content length, content type, cookies, …
  - a blank line
  - (after that, content may be transferred)

Example requesting https://www.uni-koblenz,de
(The > and < marks are not part of the protocol but added by the cURL program)

```
> GET / HTTP/1.1
> Host: www.uni-koblenz.de
> User-Agent: curl/7.64.1
> Accept: */*
>
```

In this case, the server responds with a so-called redirect message, a web browser would then request the document indicated by the Location header. The response additionally contains human-readable HTML for browsers that don't automatically redirect.

```
< HTTP/1.1 301 Moved Permanently
< Date: Thu, 16 Jan 2020 12:06:13 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Location: https://www.uni-koblenz-landau.de/de/koblenz/
< Content-Length: 338
< Content-Type: text/html; charset=iso-8859-1
<
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
<p>The document has moved <a href="https://www.uni-koblenz-
landau.de/de/koblenz/">here</a>.</p>
<hr>
<address>Apache/2.4.18 (Ubuntu) Server at www.uni-koblenz.de
Port 443</address>
</body></html>
```

Protocols “on top of HTTP”

- Private and commercial networks usually block most services from the internet, e.g. by placing firewalls (packet filters) between the external and the internal networks
- Many well-known services use privileged port numbers ≤1023

- Only a few services are exposed
  - Packets are filtered by firewalls based on source or destination IP addresses and/or protocol port numbers
    (and other criteria)
  - However, connections via HTTP(S) ports 80/443 are permitted in most networks

Protocols “on top of HTTP”

- Hence, many application protocols were built “on top” of HTTP, thereby using the same port numbers and circumventing filter rules
- Commonly, at least the initiating communication is done via HTTP ports before using application specific ports
- The HTTP protocol has an `UPGRADE` method to switch to different protocols once the connection is established

- **WebDAV**
  HTTP extension for Distributed Authoring and Versioning (RFC 4918)
- **WebSockets**
  HTTP extension (via `UPGRADE`) for bi-directional communication and push notifications (RFC 6455)
- **Messaging**
  e.g. Skype, iMessage, WhatsApp, and others
- …

4.3 Sessions

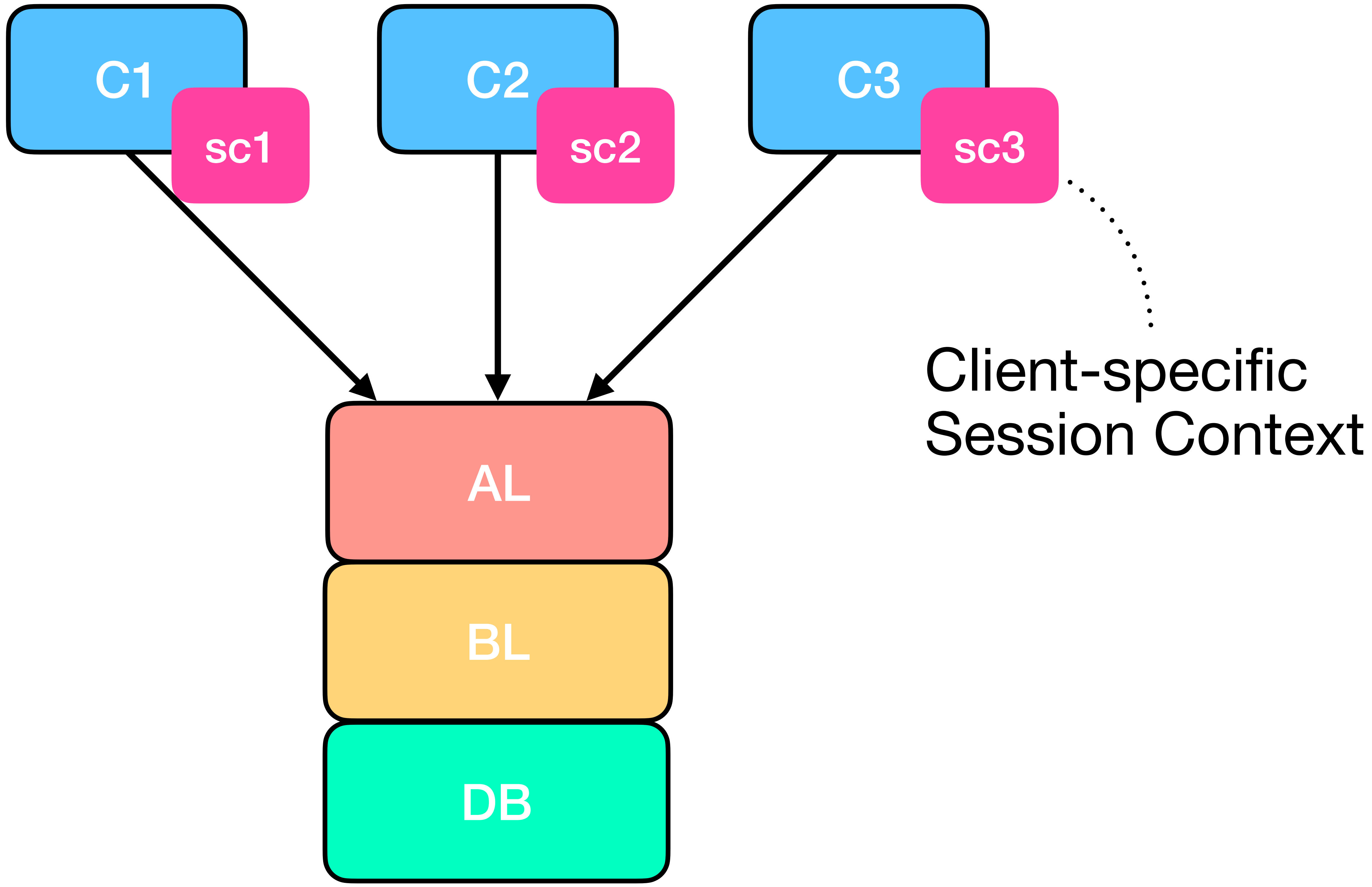
Sessions

- Problem: HTTP is stateless by definition
- Subsequent requests appear to a server as if they were unrelated, and HTTP servers *have to* handle requests as if they were unrelated.
- Web Applications require a user session, e.g. log-in, interact with the application, log-out
- Such a session has to "survive" many basic HTTP requests

- This requires that some **state data associated with the current client connection**, the so-called session context, has to be stored
- E.g. "current customer and associated shopping cart" in web shop systems
- Question: Where to store the session context?
    - Client side
    - Server side

Client-side session context

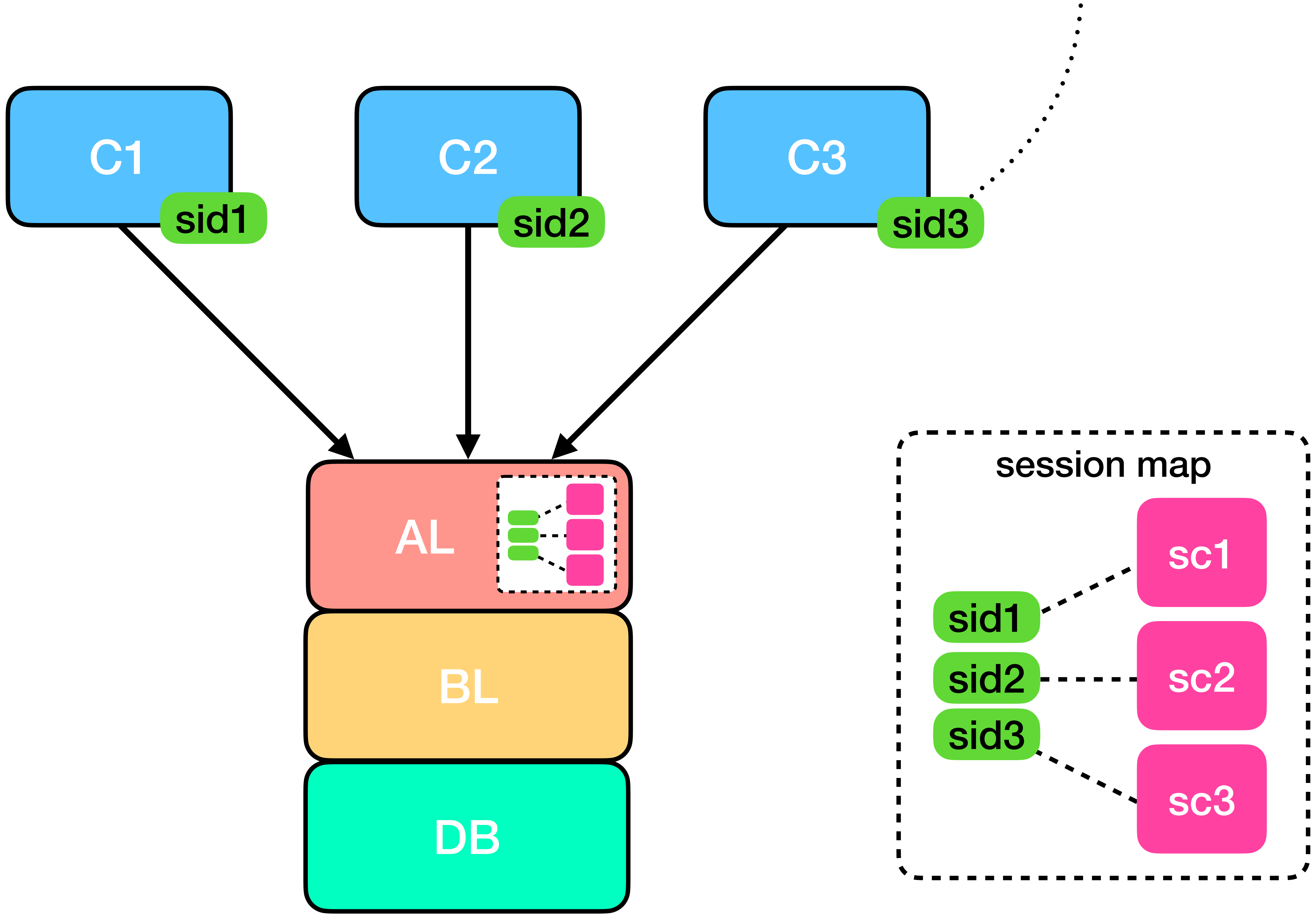
- Session context in browser memory (e.g., in cookies or local storage)
- **Advantages**
  - no memory consumption on server side, even if client breaks connection
  - server can be changed during session (→ load balancing later in this lesson)
  - session hijacking attacks can be harder
- **Disadvantages**
  - higher data transfer volume - complete context has to be transmitted
  - context may not survive client restarts



Server-side session context

- Clients only store small keys (session IDs)
- Server holds session map sid→context, *usually* in the AL/Web Server component
- **Advantages**
  - very little extra information transferred
  - session can survive restarts of client
- **Disadvantages**
  - memory consumption on the server side
  - switching servers during session requires special attendance
  - server has to monitor / evict inactive sessions
  - leaking session IDs can lead to hijacking

Client-specific Session ID

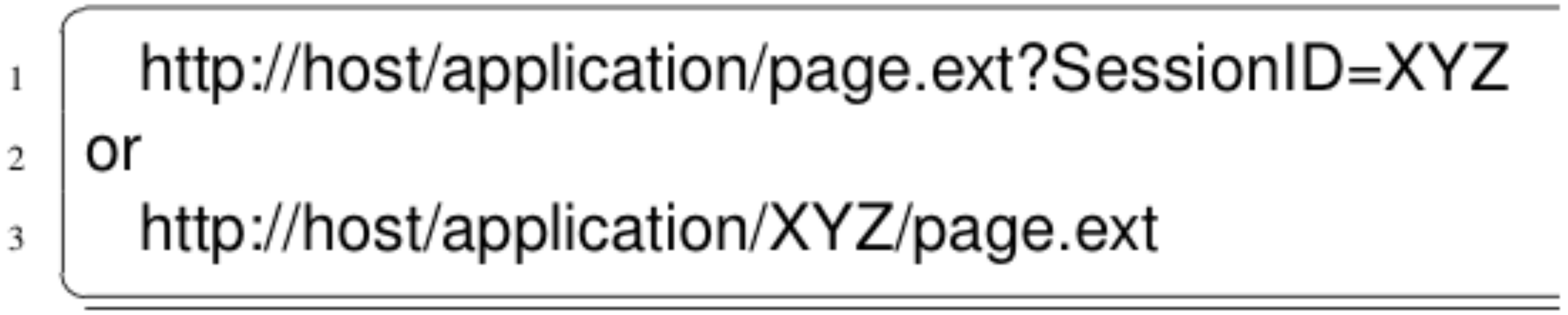


Realizing Sessions

- In most systems, server-side storage of the context is used
- Identification of simultaneous sessions realized via session IDs
- Transfer of the session IDs with each HTTP request
    - as request parameter in `GET` or `POST` requests
    - as part of the URL (requires URL rewriting)
    - as session cookie in HTTP headers
- The session context binds server resources during the lifetime of a session
- Servers usually define a session timeout to be able to remove inactive sessions to reclaim and reuse those resources

URL rewriting

- URL rewriting transmits all session-dependent data as parameters in the URL (XYZ should be random and hard to guess):



```
  http://host/application/page.ext?SessionID=XYZ
or
  http://host/application/XYZ/page.ext
```

- Requires inspection and rewriting of URLs in links inside page content

Drawbacks of URL Rewriting

- the URL can become messy and error-prone
- bookmarking becomes impossible
- the approach can be unusable due to length-limits of URLs on some systems
- the URLs inside the documents have to be adapted dynamically to include the session ID
- it is inherently insecure (unless using https, or unless no content at web-space should ever be protected).

Cookies

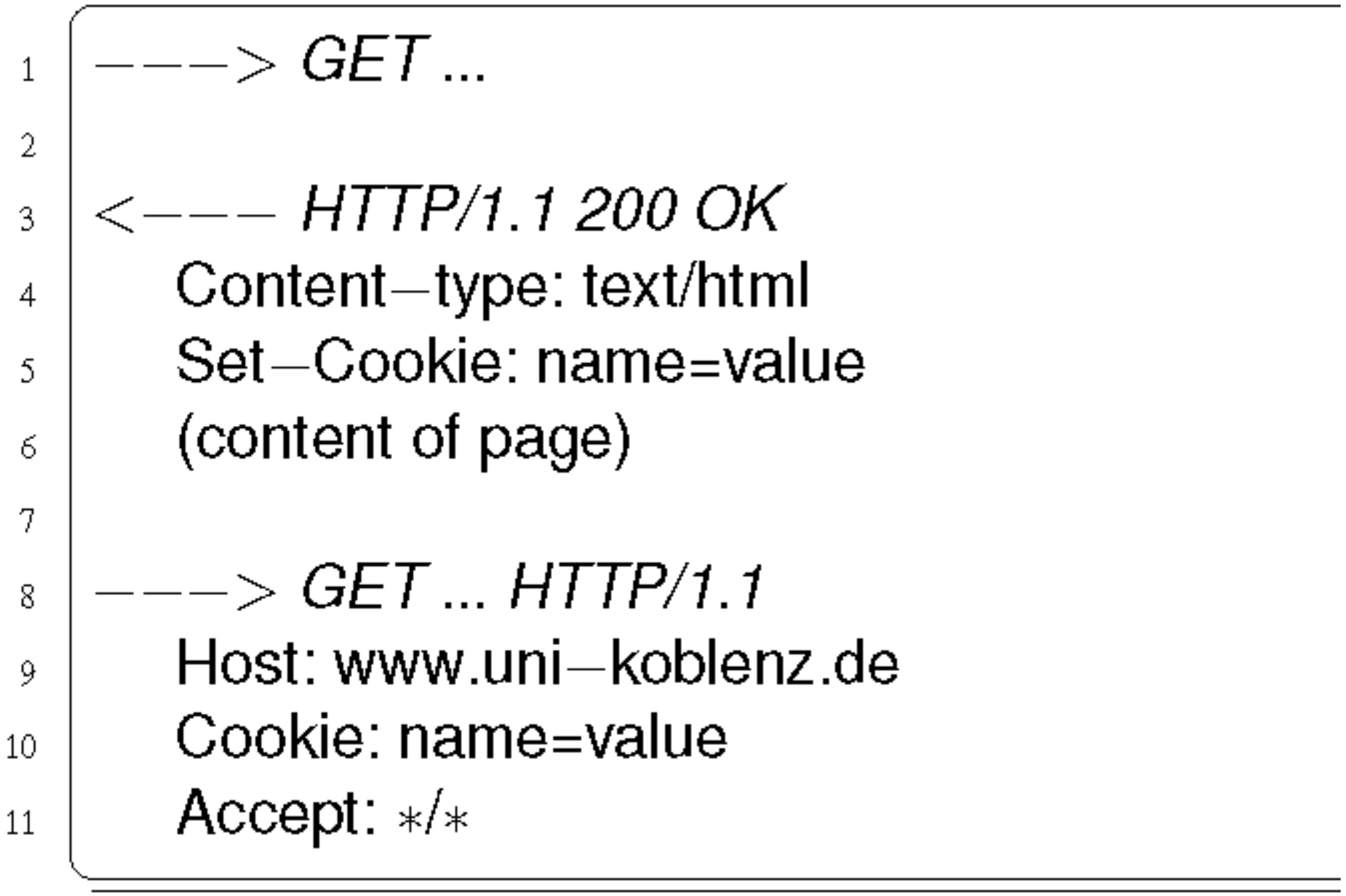
- Cookies are small text files used to store server information (e.g., a session ID) on the client node as name-value pairs.
- Cookies allow to make the session context at the server's side accessible by an ID via the session map.

Cookies

- Web server transmits cookies to the browser in the HTTP response header
- Browser re-transmits cookies to the respective server with each subsequent request in the request header
- **Session cookies**
  - kept in browser memory
  - deleted as soon as the browser terminates
- **Permanent cookies**
  - stored persistently on disk
  - for a certain lifetime or unlimited

- Cookies represent sensitive information
  - identification of a user during a session
  - countermeasures against session hijacking attacks have to be taken
  - tracking of activities possible
  - …

Cookie Headers



```
--->  GET ...

<---  HTTP/1.1 200 OK
      Content-type: text/html
      Set-Cookie: name=value
      (content of page)

--->  GET ... HTTP/1.1
      Host: www.uni-koblenz.de
      Cookie: name=value
      Accept: */*
```

Examples of session cookies (using browser tools)

4.4 Vertical and Horizontal Scaling, Load Balancing

Scalability…

- Scalability is a quality measure for **how easy a system can be adapted to varying loads**
- Adaption means to **react** on **higher** loads by scaling up, and to react on **lower** loads by scaling down
- Ideally, a system can be scaled without interruption of service
- In software systems, architectural design plays a **central role to achieve scalability**

- Dealing with web applications, we assume distributed systems consisting of web browsers, web servers, application servers, database servers, (any possibly many more components)
- Systems with layered architectures generally are more flexible since scaling can be applied on the bottlenecks only

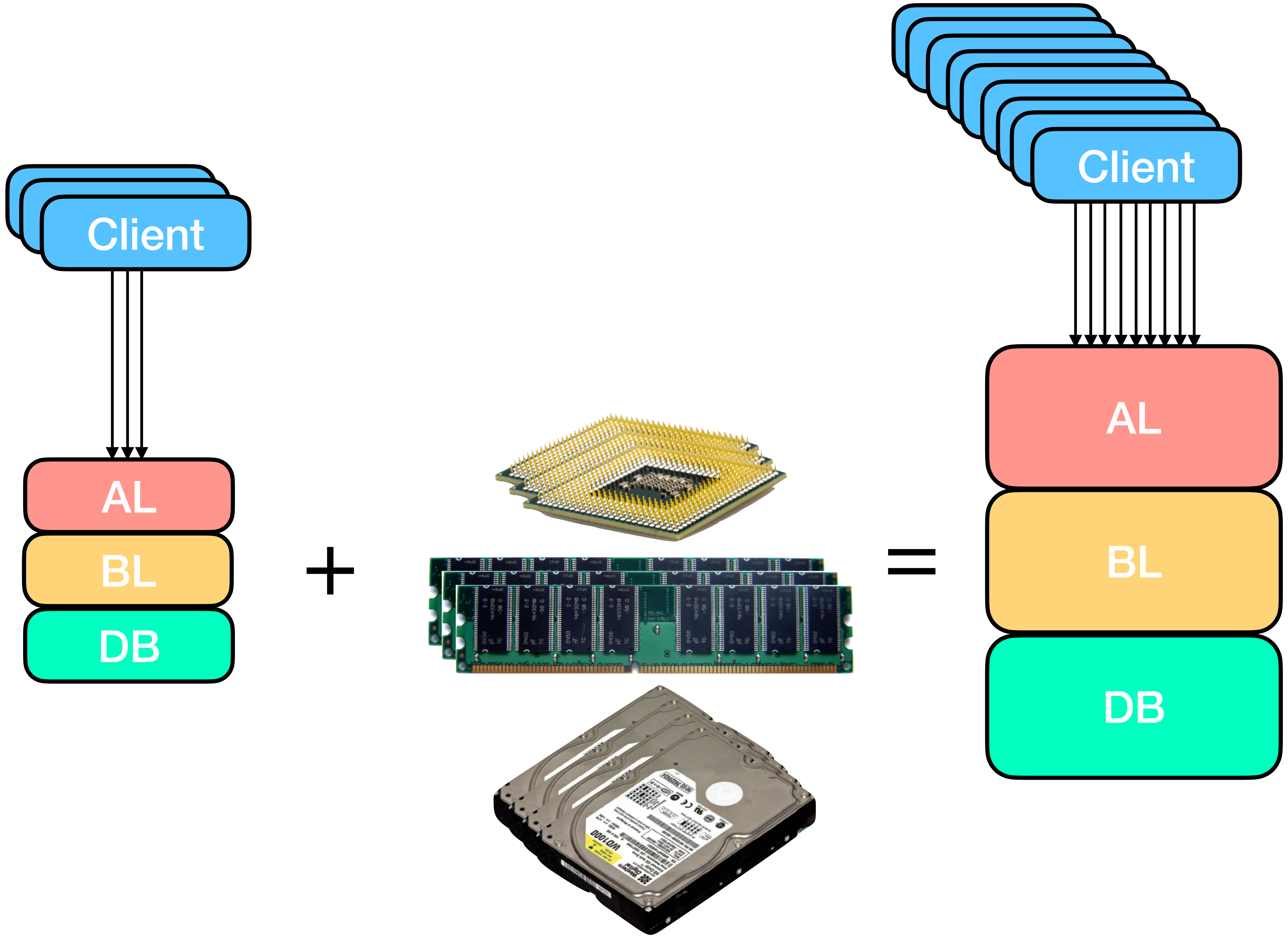
Resource Congestion

- Network and server resources shared by many clients
- Limitations due to
  - Network bandwidth
  - Number of concurrent connections
  - Memory consumption
  - CPU consumption
  - Storage space
  - Storage bandwidth
  - …and other factors
- Increased number of concurrent clients and/or requests consume resources
- High load results in slow processing, or even in denial of service

Vertical Scaling

Put more power to a machine

- Usable in situations of
  - Memory congestion
  - CPU congestion
  - Storage space congestion
  - Storage bandwidth congestion
- Not applicable to deal with limits in network bandwidth or concurrent connections

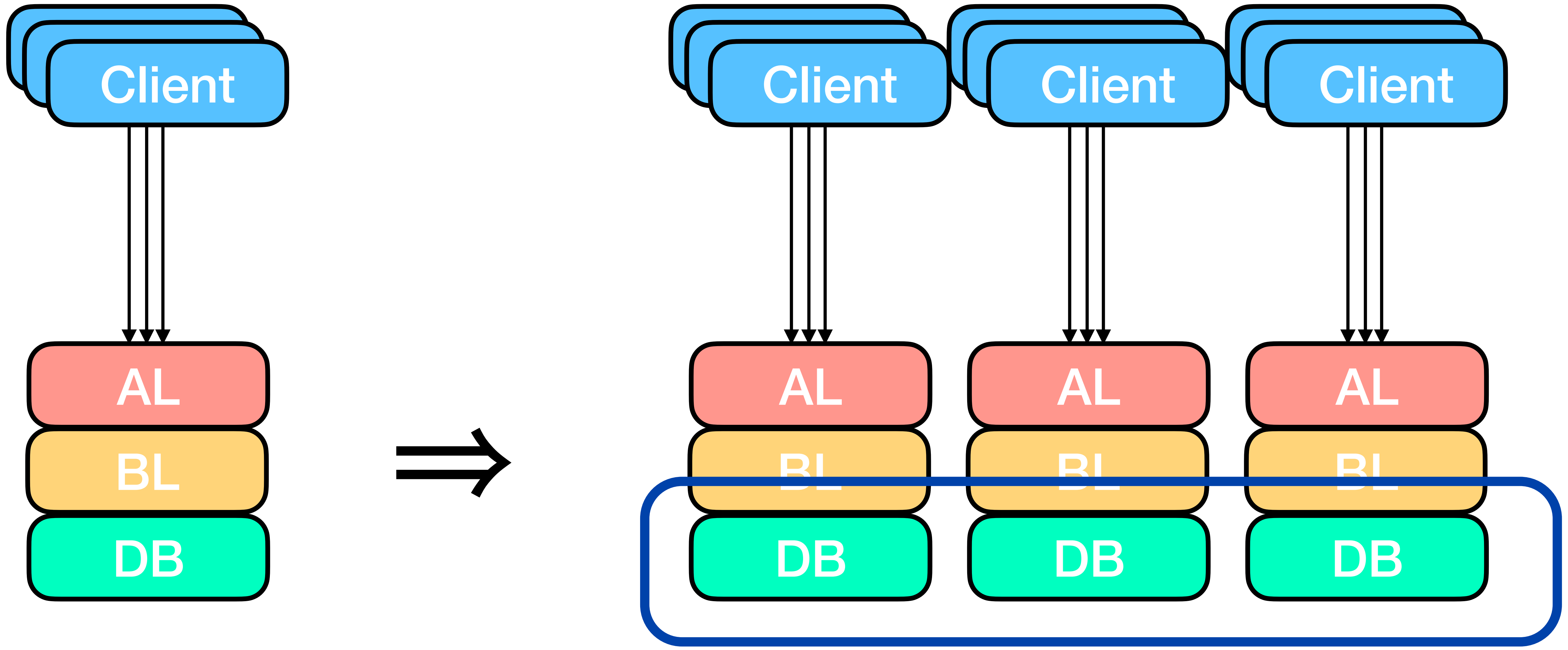


Horizontal Scaling

Increase number of machines

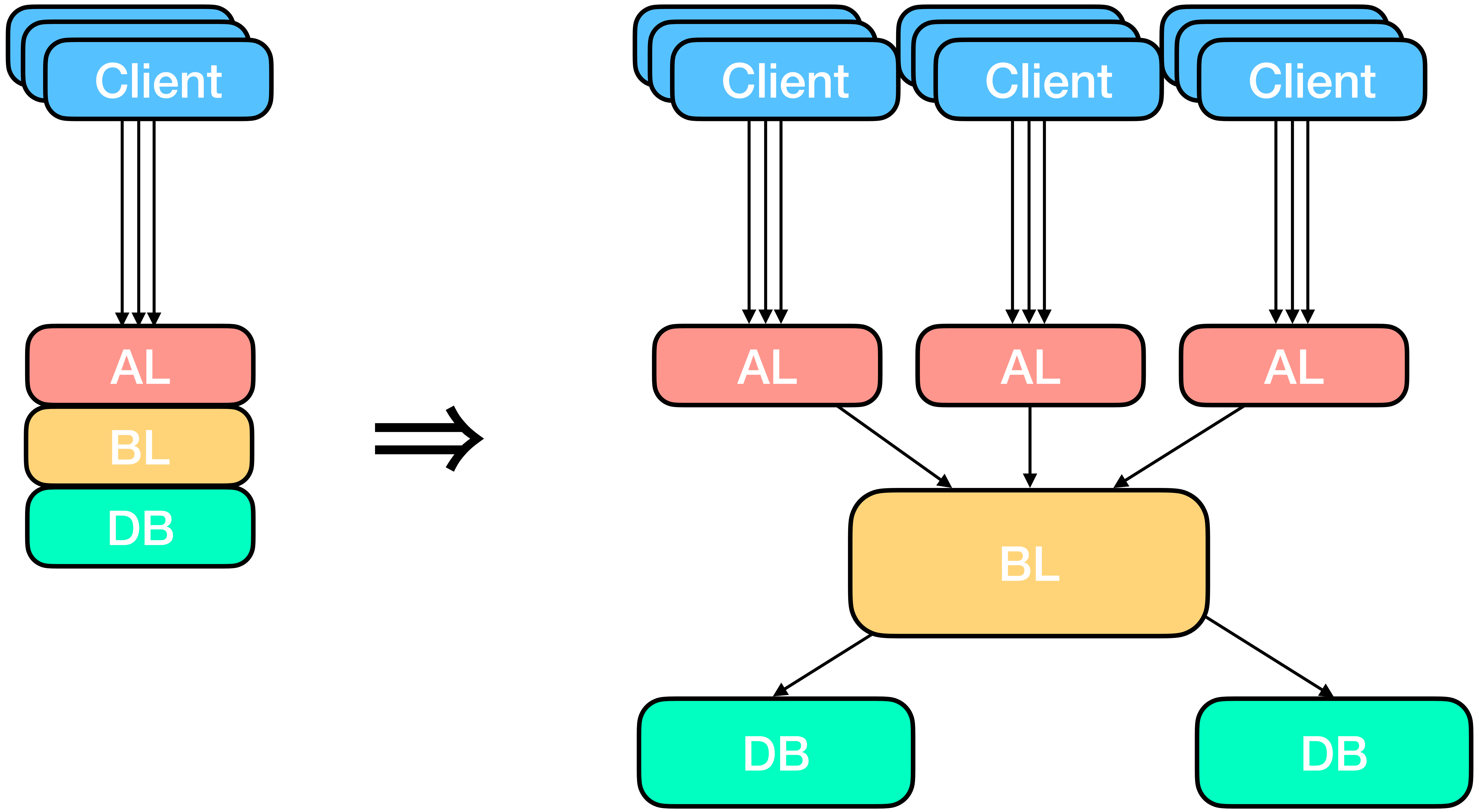
- Usable in situations of
  - Network bandwidth congestion
  - Exceeded number of concurrent connections
  - Memory congestion
  - CPU congestion
  - Storage space congestion
  - Storage bandwidth congestion

Load balancing required to distribute clients/ requests to multiple servers



Horizontal DB scaling requires separation of and synchronization between data partitions

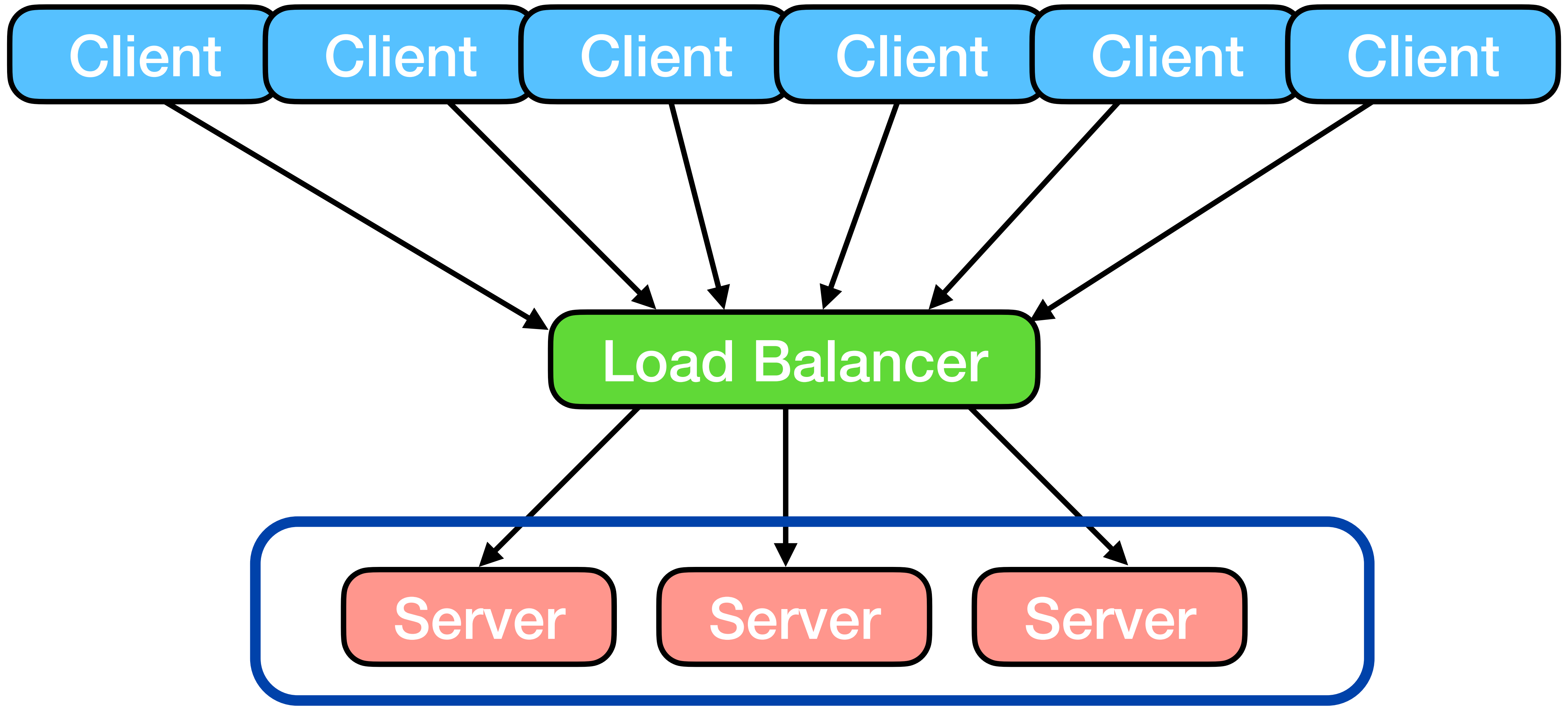
Combined Vertical and Horizontal Scaling



Layered architectures allow to scale each layer individually depending on the individual congestion problem

Load Balancing

- Transparent to the clients - only a single server from a client's perspective
- Load balancing can work on various layers of the network stack (see page #7)
- Static and dynamic variants
  - Dynamic variants consider system state to take decision, static variant doesn't
- Methods include
  - DNS round robin (simplest approach)
  - NAT based load balancing (local, network layer)
  - Anycast load balancing (global, network layer)
  - Flat based balancing (local, link layer)
  - Application layer balancing



- Load balancer can become new bottleneck
- Data inspection can be infeasible due to end-to-end encryption

Load Balancing

- DNS round robin
  - DNS = Domain Name System
  - Multiple servers with a single name
  - Name-to-IP-Address resolution results in different IP on each request
  - Easy implementation, on on-premise devices
  - Difficult to realize server-based session stores
  - Client “sees” multiple servers

- NAT based load balancing
  - NAT = Network Address Translation
  - Multiple servers on internal network with different addresses
  - Load balancer/router rewrites network packets from clients to reach different internal servers
  - All traffic needs to go through load balancer
  - Server-based session stores require “server affinity”, i.e., the same server has to be assigned to a client during a session
  - Servers appear as single node clients

Load Balancing

- Anycast load balancing
  - Multiple servers with same IP address in spatially separated networks
  - Global routing strategy (BGP, Border Gateway Protocol) ensures that only one server responds
  - Server with shortest network route (number of “hops”) is selected
  - Usually, picks a server geographically close to the client
  - Server-based sessions require relatively “stable” network
  - Servers appear as single node to clients

- Flat based balancing
  - Multiple servers with different addresses on local network
  - Also known as “MAT” - MAC Address Translation
    (MAC = Media Access Control address, physical local network address of a node)
  - Load balancer rewrites incoming packages to different server MACs, outgoing traffic from server to client bypasses load balancer
  - Beneficial because in general, incoming data volume is way smaller than outgoing; prevents overload of the load balancer
  - Affinity required for server-bases sessions
  - Servers appear as single node to clients

Load Balancing vs. Sessions

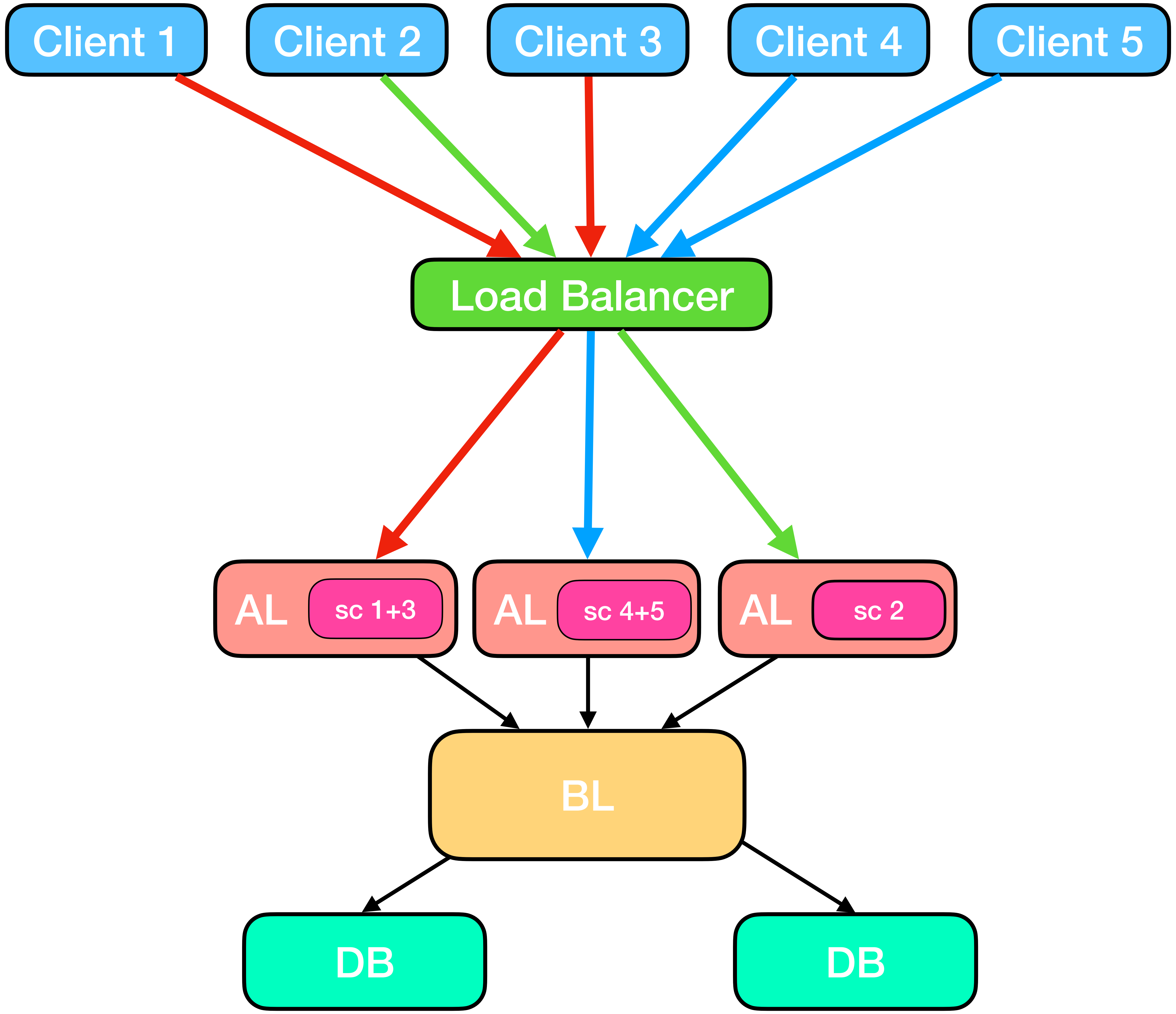
- Presence of server-side session context requires “server affinity”, i.e. the **same server** has to be used for **subsequent requests** of a client

- Depending on the scaling strategy, session maps have to be moved to BL layer and/or stored in the DB layer

  - Increased memory load to lower layers, results in slower session access

  - More traffic between lower layers

  - Possibly need to synchronize distributed databases and/or implement “affinity” on lower layers



Stateless Interface

- Horizontal scaling is easier with stateless interfaces
- Stateless means that a server instance doesn’t store any state information
- Hence, each server instance is equivalent to all others
- → Facade pattern in the architecture chapter
- Load balancer may select any server for each request, no affinity required
- Stateless AL and BL layers can be achieved by client-side session context, or by storing the session context in the DB layer
- As mentioned before, data transfer volume increases while memory load in AL and/or BL decreases
- Careful investigation of performance bottlenecks required to achieve an appropriate solution design in such tradeoff situations

What we have learned…

Communication (Part I)

✓ Network basics

✓ HTTP

✓ Sessions

✓ Scaling and Load Balancing

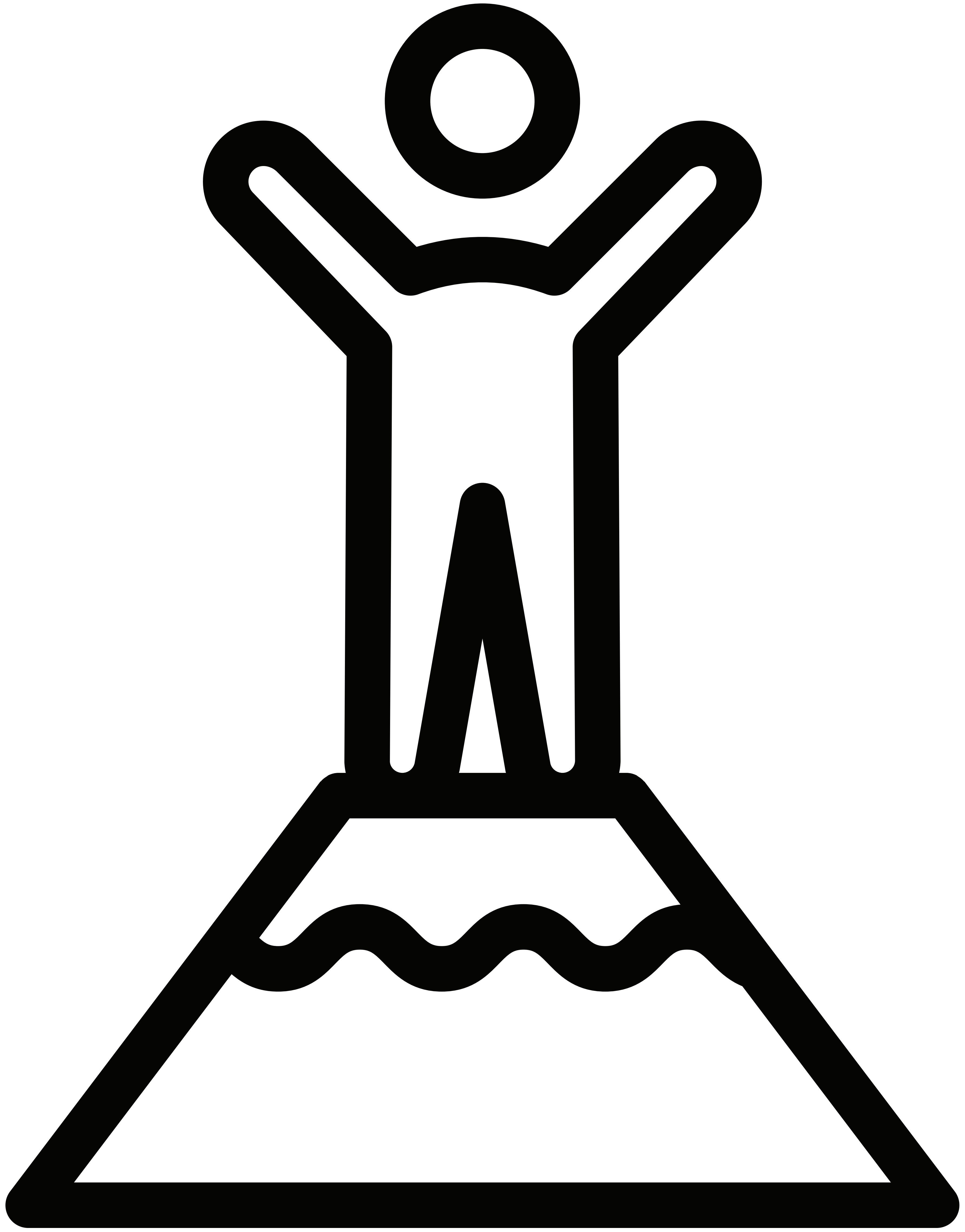


Image: colourbox.de