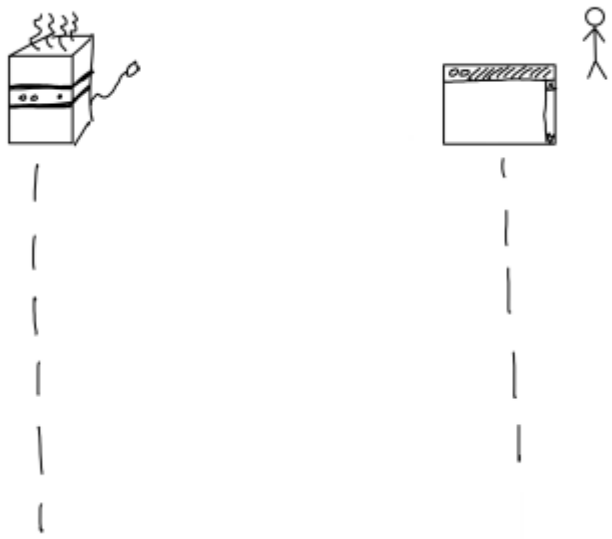
HTTP

Nachdem jetzt klar ist, wie grundsätzlich Inhalte zwischen zwei Rechnern *gesprochen* wird, wird es etwas konkreter: Der Server ist jetzt der Web-Server und der Client ist der Web-Browser. Dafür müssen sie sich auf eine gemeinsame Sprache verabreden, das ist das



Hyper**t**ext **T**ransfer **P**rotocol

Der Name scheint aus einer anderen Zeit zu stammen. Tatsächlich ist HTTP gar nicht so alt wie das Internet:

- RFC 1945 HTTP/1.0 (1996)
- RFC 2616 HTTP/1.1 (1999)
- RFC 7540 HTTP/2 (2015)

Zu den einzelnen Bestandteilen des Worts:

Was macht einen Text zu einem Hypertext?

Es geht nicht nur um Text im landläufigen Sinn. Der Text enthält selbst noch Metadaten, die Formatierung und Textsatz ermöglichen und (ganz wichtig) es sind Elemente mit Interaktion möglich. Zuerst nur *Links*, die zu einem anderen Dokument verweisen, aber auch Formularelemente o.ä..

Der Name steht für eine Gattung *moderner* Dokumente. Das prominenteste Format eines solchen Dokumententyps ist **HTML**:

Hyper**t**ext **M**arkup **L**anguage

Merke: Das ist kein Protokoll, das ist ein Dokumentenformat!

Das H in HTTP macht klar, dass das Protokoll unter anderem für den Zugriff auf Dokumente diesen Typs gebaut wurde. Es war und ist aber nicht die einzige Aufgabe geblieben.

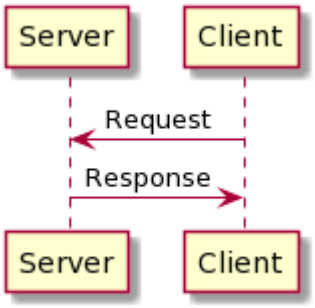
Damit ist aber auch verständlich, wofür der Begriff *Transfer* steht: Der Zugriff und die Übertragung

einzelner Dokumente, z.B. HTML Dokumente, die in sich ja Verweise (Links) auf andere Dokumente mittels HTTP Zugriff enthalten können.

Was macht es dann zu einem Protokoll?

Das Zusammenspiel zwischen dem Browser und dem Server folgt dem allereinfachsten Muster bidirektionaler Kommunikation:

Frage / Antwort, das heißt im Fachjargon Request / Response



1. Der Browser (ein TCP-Client) stellt dem Server eine wohldefinierte Anfrage.
2. Der Server antwortet auf demselben TCP-Kommuniationskanal exakt auf diese Anfrage mit einem Status-Code (OK oder irgend ein Fehler) und in einem und in einem Rutsch mit dem angefragten Inhalt.
3. Danach wird der Kommunikationskanal wieder abgebaut (so war das zumindest ganz zu Beginn, mittlerweile wartet der Server auf der einmal aufgebauten TCP-Verbindung, ob der Browser nicht direkt mehrere Requests loswerden möchte, das ist effizienter)

> **Merke:** Die ganze Logik steckt eigentlich im Browser, der Server reagiert nur und gibt stumpf zu jeder einzelnen Anfrage das eine passende Dokument zurück (oder reagiert mit einer Fehlermeldung).

Ganz so einfach ist das heutzutage alles auch nicht mehr. Aber ein minimalistischer Webserver könnte genau so funktionieren. So ein Server ist beispielsweise ein sogenannter „statischer“ Webserver, der Anfragen auf fest vorgegebene Inhalte oder an fixen Orten abgelegte Dokumente abbildet und mit diesen Inhalten antwortet. Ein solches Verhältnis zwischen Client und Server nennt man auch *zustandslos*, weil der *Response* völlig unabhängig von der Vorgeschichte der Kommunikation ausschließlich vom Inhalt des *Requests* abhängt: Gleiche Frage, gleiche Antwort!

Aber wie sieht ein wohldefinierter Request aus?

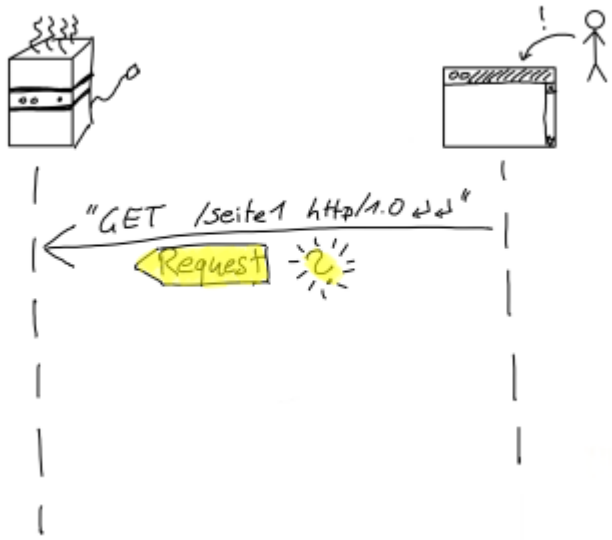
HTTP nutzt eine besonders einfache Art und Weise: Menschenlesbarer Text!

Das hat den Vorteil, dass es sich leicht realisieren lässt und weitgehend unabhängig von der zugrundeliegenden Technik ist.

Aber man muss sich an eine strikte Konvention halten, damit der Text der Anfrage/des Requests auch von jeder Maschine einfach interpretiert werden kann und nicht kompliziert analysiert werden muss. Folgende Regeln muss der Browser beim Zusammenbau des Request einhalten:

Der Text beginnt mit der ersten Zeile und dem ersten Zeichen, Leerzeichen sind Trenner zwischen

den einzelnen Bestandteilen. Das erste Wort ist ein Kommando, das beiden Seiten bekannt sein muss und im jeweils aktuellen RFC definiert wird: GET, POST, HEAD usw. Danach folgt als zweites Wort die sogenannte *Ressource*, also das Dokument, dass auf diesem Server angefragt wird. Das ist der Mindestsatz an Informationen. Die Anfrage wird mit einer leeren Zeile beendet. Heutige Browser sind äußerst geschwätzig und übermitteln darüber hinaus (also vor der Leerzeile) noch jede Menge Zusatzinformation, die der Server bei der Auswahl und Zusammenstellung des Dokuments hinzuziehen kann, praktisch alles davon ist aber optional bei der einfachsten Request-Form, dem GET. Beispiele:



- HTTP Version (z.B. „http/1.1“)
- Browser-Typ (z.B.„
- Landessprache
- unterstützte Dokumenttypen (außer HTML)
- Cookie-Werte von vorherigen Besuchen dieses Servers

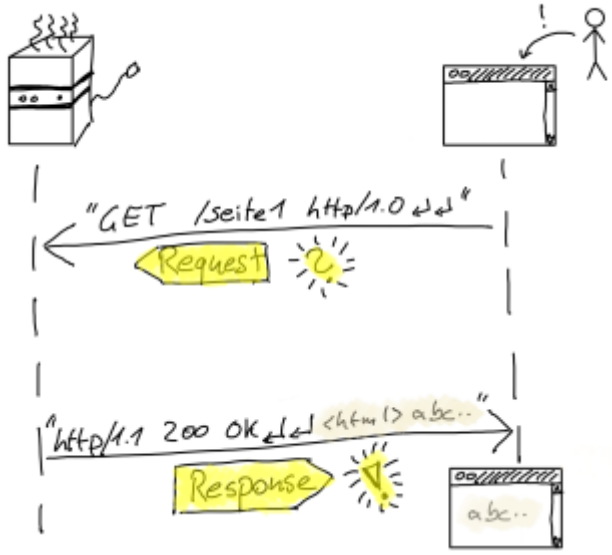
Ab HTTP 1.1 folgt nach der Zeile mit dem Kommando (also z.B. „GET /index.html“) immer eine Zeile, aus der der Name des Servers, so wie der Browser ihn nennt, ablesbar ist:

```
Server: www.qgelm.de
```

zum Beispiel. Das hat erst mal nichts mit der IP-Adresse des Servers zu tun. Aber es ist durchaus üblich, dass ein Server mit einer IP-Adresse verschieden Namen haben kann und verschieden reagiert. Zum Zusammenhang des *Namens* mit der *IP-Adresse* gibt es hier einen kleinen Exkurs.

Der Server schaut dann, wie er den Inhalt der angeforderten Ressource bekommt und packt das wie

folgt in seinen *Response*:



- Er nennt das verwendete Protokoll mit Version also z.B. `http/1.1`).
- Wenn alle glatt lief folgt dann der Statuscode mit Zahl und Kurzbeschreibung `200 OK`.
- Bis hier war alles in einer Zeile. Die nächsten Zeilen können dann weitere sogenannte *HTTP-Header* enthalten, sind aber optional.
- Die eigentlich übermittelte Ressource folgt dann nach einer Leerzeile und wird einfach angehängt.

Kommen wir nach der Vorrede zu einem einfachen Beispiel eines statischen HTTP-Servers.