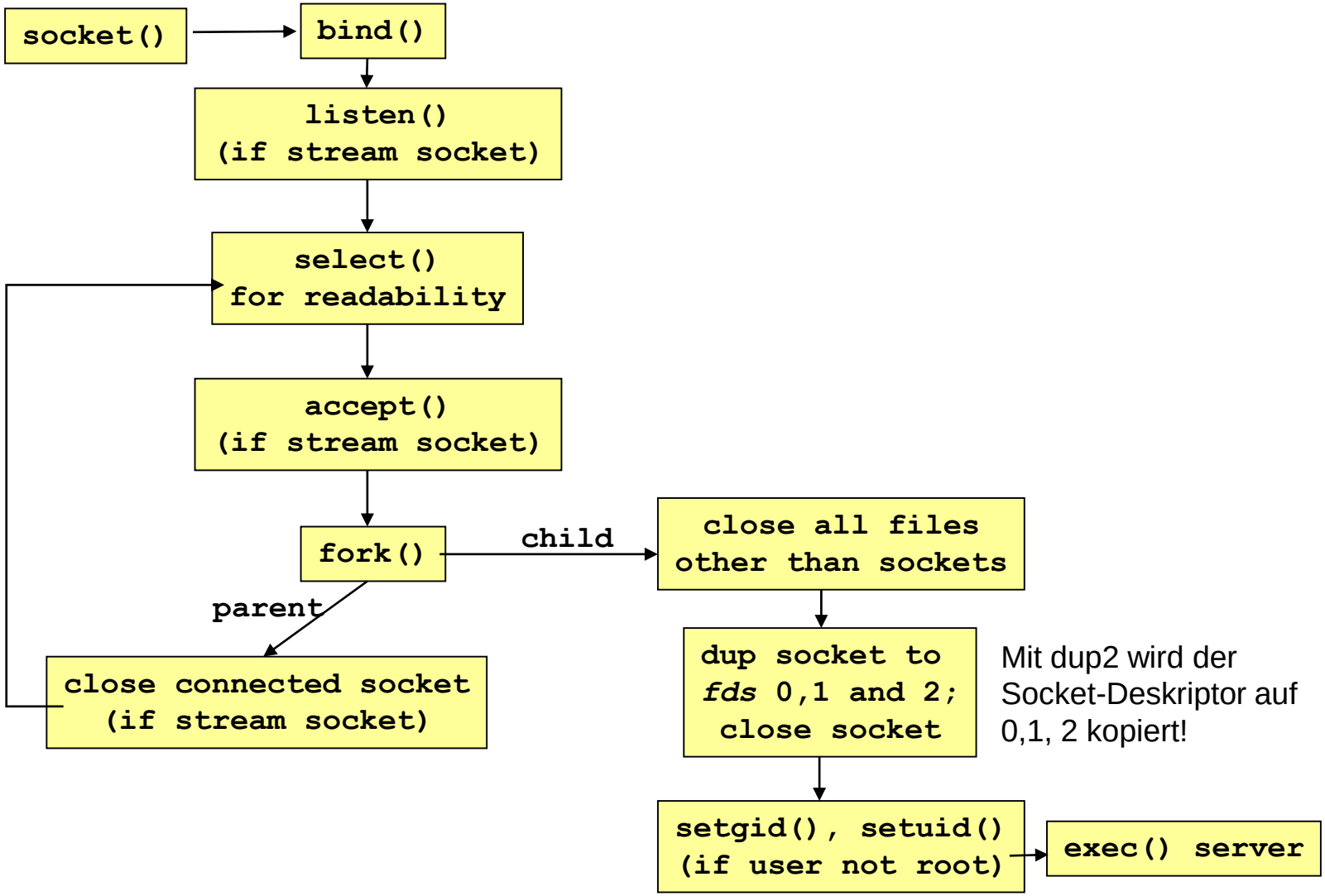
Webserver

Apache, nginx

Server
inetd-Modell



Server
inetd-Modell

Ablauf:

- Master Server wartet auf Anforderungen

- Anforderung erhalten
  - > Master Server akzeptiert Verbindung
  - > Master Server erzeugt Child Proess und übergibt Verbindung
  - > Child Process behandelt Anforderung

Ergebnis

- Pro Anforderung wird ein (Child) Server Prozess erzeugt
- Jeder Server muss seine Initialisierung selbst vornehmen

Weitere Informationen: http://www.fmc-modeling.org/category/projects/apache/amp/4_3Multitasking_server.html

Diskussion

- Diese Multiprocessing-Architektur ist sinnvoll, wenn die individuellen Aufgaben langfristiger Natur sind und der Client einen Status der Verbindung verwaltet

- HTTP ist jedoch zustandslos, wodurch der Client nicht über einen solchen Status verfügt

- Eine Server-Implementierung auf der Basis dieser Architektur wäre somit wenig effektiv
  - > Der Master-Server müsste für jede HTTP-Verbindung einen neuen Prozess kreieren, der nur diese eine Verbindung bedient. Aufwand!
  - > Das Erzeugen eines Kindprozesses blockiert die Verarbeitung des Master-Servers. Dieser kann währenddessen keine eingehenden Anfragen bearbeiten.

Server
Multi Processing Modules (MPM)

Wie werden Server-Anfragen abgearbeiten?

- Multi Processing Modules bestimmen wie es funktioniert
- Apache kommt standardmäßig mit zwei Modellen

Prefork

- Pro Anfrage ein Prozess (ähnlich inetd)
- Saubere Trennung der Anfragen
- Aber: RAM-intensiv

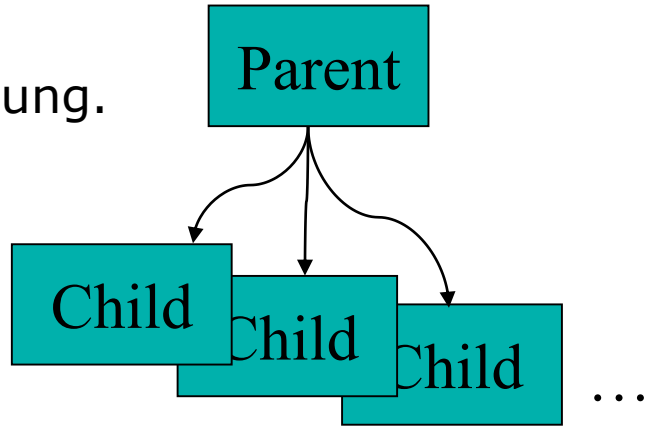
Worker

- Pro Anfrage ein Thread
- Aber: Verarbeitende Library muss multithreading-fähig sein

Server
Prefork-Modell

Grundsätzliche Idee

- Forking findet vor der eigentlichen Anfrage statt („Pre-Forking“)
- Prozesse „warten“ darauf genutzt zu werden
- Jeder Prozess behandelt jeweils eine Verbindung.
  - > Hoher Speicherbedarf
  - > “You’ll run out of memory before CPU”



Leader-Followers-Pattern

- Die Preforking-Architektur basiert auf einem Vorrat von Prozessen, die drei verschiedene Rollen haben:
  - > Listener: Warten auf Anfragen (Leader)
  - > Worker: Verarbeiten von Anfragen (untätige Worker = Follower)
  - > Idle-Worker: Einreihen in die Warteschlange um darauf zu warten, die Rolle des Listener zu übernehmen

Weitere Informationen: https://httpd.apache.org/docs/2.4/de/mod/prefork.html

Server
Prefork-Modell

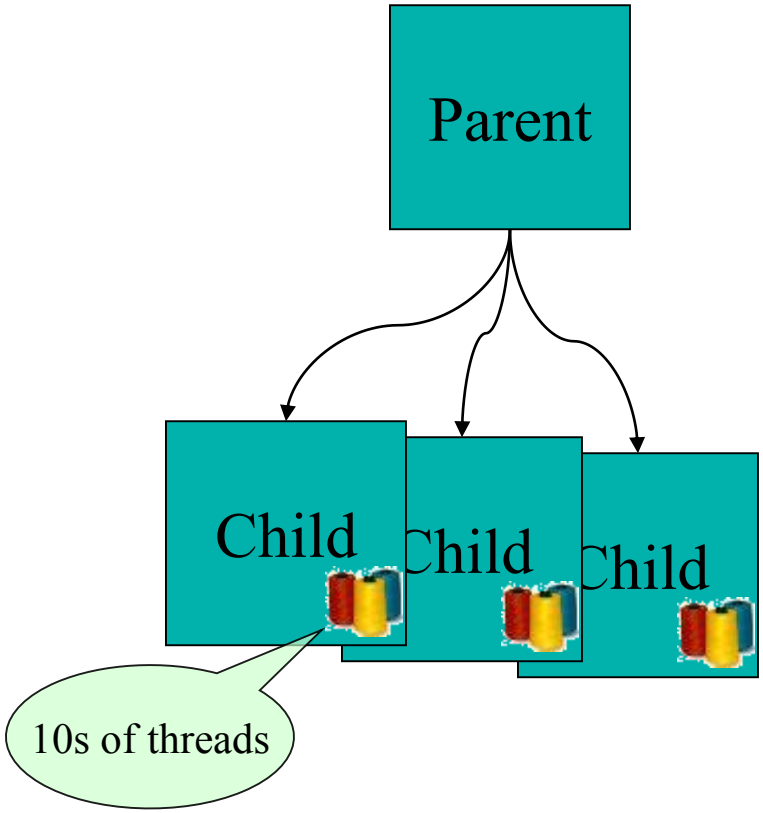
Direktiven

- StartServers
  > Anzahl der zu startenden Serverprozesse
- MinSpareServers
  > Mindestzahl der Serverprozesse, die ohne Last bleiben dürfen
- MaxSpareServers
  > Höchstzahl der Serverprozesse, die ohne Last bleiben können
- MaxClients
  > Höchstzahl der Serverprozesse, die gestartet werden dürfen
- MaxRequestsPerChild
  > Höchstzahl der Anfragen, auf die ein Serverprozess reagieren kann, bevor er beendet wird

Server
(Multithreaded-)Worker-Modell

Grundsätzliche Idee

- Nur wenige Child-Prozesse
- Jeder Child-Prozess behandelt viele Verbindungen gleichzeitig
  - > Ein Thred pro Verbindung



Server
(Multithreaded-)Worker-Modell

Direktiven

- MinSpareThreads
  - > Minimale Anzahl unbeschäftigter Threads
- MaxSpareThreads
  - > Maximale Anzahl unbeschäftigter Threads, die zur Bedienung von Anfragespitzen zur Verfügung stehen
- ThreadsPerChild
  - > Anzahl der Threads, die mit jedem Kindprozess gestartet werden
- MaxClients
  - > Maximale Anzahl der Kindprozesse, die zur Bedienung von Anfragen gestartet wird
- MaxRequestsPerChild
  - > Obergrenze für die Anzahl von Anfragen, die ein einzelner Kindprozess während seines Lebens bearbeitet

Server
Multi Processing Modules (MPM)

Prefork vs Worker

- Standard für Apache ist Prefork
- Worker-Modell wäre zu präferieren, da es weniger RAM benötigt
- Nur möglich, wenn das verwendete Modul auch thread-safe ist

Beispiel: mod_php (Apache PHP Module)

- Auszug aus der PHP Installation FAQ:

> ***Why shouldn't I use Apache2 with a threaded MPM in a production environment?***
>
> *PHP is glue. It is the glue used to build cool web applications by sticking dozens of 3rd-party libraries together and making it all appear as one coherent entity through an intuitive and easy to learn language interface. The flexibility and power of PHP relies on the stability and robustness of the underlying platform. […]*
>
> Quelle: http://php.net/manual/en/faq.installation.php#faq.installation.apache2

- Fazit: PHP sollte nur im Prefork-Modus betrieben werden

Apache HTTP Server

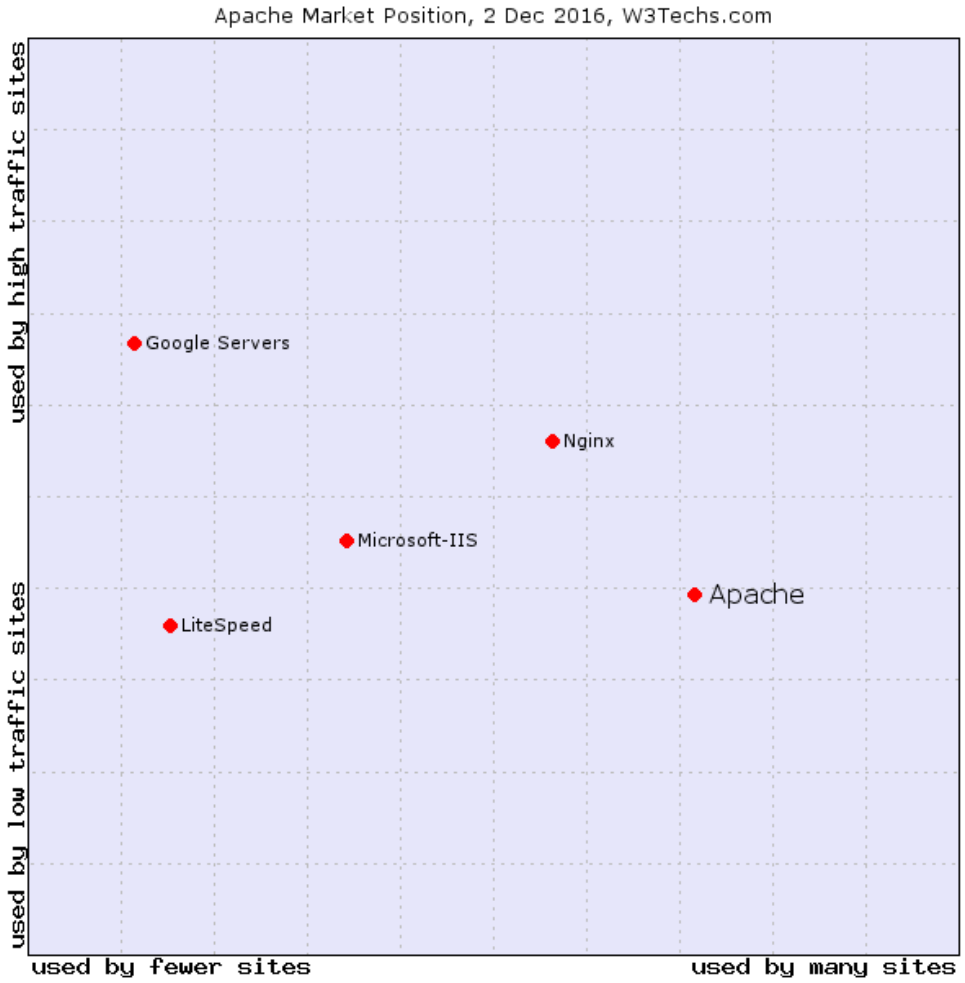
Das erklärte Ziel

- To provide an open-source, secure, efficient and extensible server that provides HTTP services in sync with non-proprietary World Wide Web standards

Apache Software Foundation (vorher: Apache Group)

- Non-Profit Organisation
- Entwicklung neuer Versionen und Fehlerbehebung
- Integration von Add-ons, die außerhalb der Kernentwicklergruppe implementiert wurden
- Release Tests
- Dokumentation
- apache.org

Apache
WebServer im Vergleich

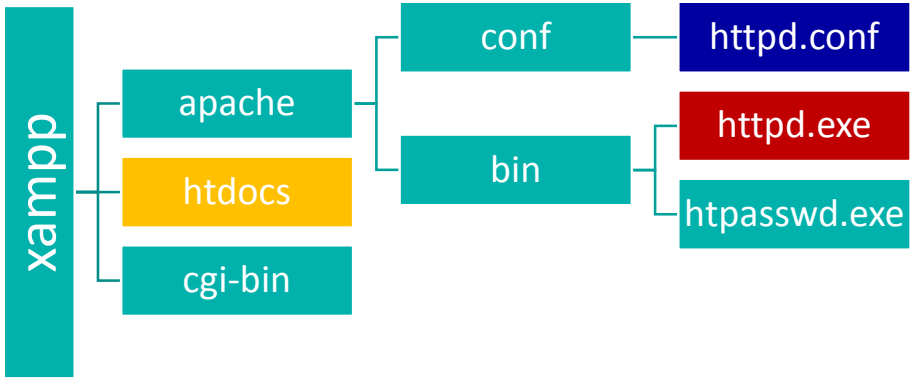


Quelle: https://w3techs.com/technologies/details/ws-apache/all/all

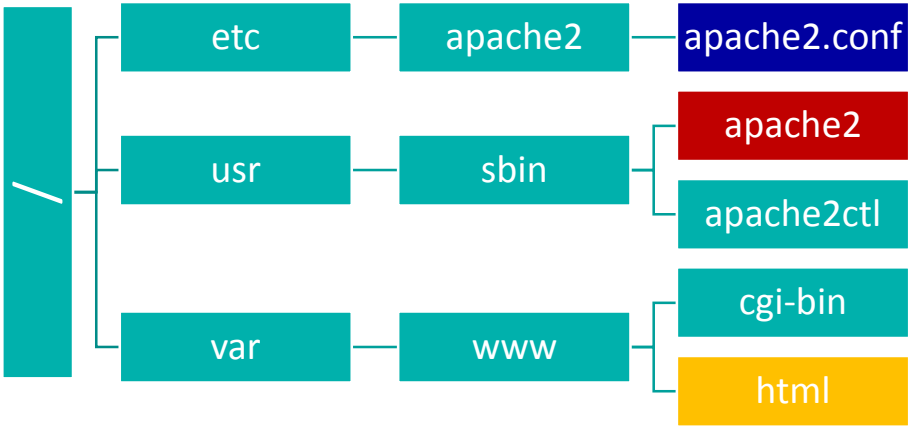
Apache
Verzeichnisstruktur

Verzeichnisstruktur ist abhängig von Betriebssystem und Distribution!

XAMPP Windows:



Apache Ubuntu:



Apache Executable

Hauptconfig

DocumentRoot

Einstiegsverzeichnis für den Webserver

Weitere Informationen: https://wiki.ubuntuusers.de/Apache/

Apache
Einsatz

XAMPP nur Entwicklungssystem!

- Viele Konfigurationen im Produktiveinsatz unsicher!
  - > PHP Errors werden dem Benutzer angezeigt
  - > Verzeichnisinhalte werden Besuchern angezeigt
- Eingeschränkte (Rolling-)Update- und Clustermöglichkeit

Produktivumgebungen meist Linux-Systeme

- Ubuntu
- Debian
- Red Hat
- Docker Container

Apache
Konfiguration

Hauptkonfigurationsdatei des Servers

- üblicherweise `httpd.conf`
- inkludiert weitere Konfigurationsdateien

Änderungen in den Konfigurationsdateien erfordern Neustart des Servers

- Zusätzlich kann in jedem Dokumentenverzeichnis die Datei .htaccess zur Konfiguration herangezogen werden (z.B. Zugriffskontrolle auf bestimmte Dateien)

Weitere Informationen: https://httpd.apache.org/docs/2.4/configuring.html

Apache
Konfiguration

Erweiterungen des Servers durch Module

- Beispiele: Authentisierung, PHP, Session-Management, …
- AddHandler/SetHandler
  - > Hinzufügen eines Handlers für Dateien
  - > Beispiel: `AddHandler cgi-script .cgi .pl .asp`

- LoadModule/LoadFile
  - > Aktivieren eines Modules
  - > Beispiel: `LoadModule userdir_module modules/mod_userdir.so`

- IfModule
  - > Für den Fall, dass eine Konfiguration nur in bestimmten Fällen aktiviert werden soll, sind auch If-Anweisungen möglich
  - > Beispiel:

```
<IfModule dir_module>
    DirectoryIndex index.php index.html index.htm
</IfModule>
```

Apache
Konfiguration

Die zentrale Konfigurationsdatei ist in drei Bereiche aufgeteilt:

- Globale Umgebung
  - > LoadModule
  - > Server-Root
  - > RLimitCPU, RLimitMem
- Hauptserver-Konfiguration
  - > Server-Name, Server-Admin, Server-Tokens
  - > Document-Root (Wo stehen die Dokumente?)
  - > Für Unix: UserDir (Wo stehen die Dokumente der Benutzer)
  - > DirectoryIndex (was wird gemacht, wenn ein Verzeichnis angesprochen wird?)
- Virtuelle Hosts
  - > VirtualHost (Verfügbarkeit mehrerer Webangebote auf einem Server)
  - > ServerAlias, ServerPath

Beispiel: httpd.conf

```
ServerRoot "C:/xampp/apache"
Listen 80

LoadModule alias_module modules/mod_alias.so
LoadModule autoindex_module modules/mod_autoindex.so
LoadModule auth_basic_module modules/mod_auth_basic.so
LoadModule dir_module modules/mod_dir.so
# rewrite, ssl, ...

ServerAdmin postmaster@localhost
ServerName localhost:80

<Directory />
    AllowOverride none
    Require all denied
</Directory>
```

Weitere Informationen: http://httpd.apache.org/docs/2.4/

Beispiel: httpd.conf (Forsetzung)

```
DocumentRoot "C:/xampp/htdocs"
<Directory "C:/xampp/htdocs">
    Options Indexes
    AllowOverride All
    Require all granted
</Directory>

<IfModule dir_module>
    DirectoryIndex index.php index.html index.htm
</IfModule>

<Files ".ht*">
    Require all denied
</Files>

ErrorLog "logs/error.log"
LogLevel warn

Include conf/extra/httpd-mpm.conf
Include conf/extra/httpd-vhosts.conf
Include conf/extra/httpd-xampp.conf
```

Beispiel: httpd-xampp.conf

```
LoadFile "C:/xampp/php/php7ts.dll"
LoadModule php7_module "C:/xampp/php/php7apache2_4.dll"

<FilesMatch "\.php$">
    SetHandler application/x-httpd-php
</FilesMatch>
<FilesMatch "\.phps$">
    SetHandler application/x-httpd-php-source
</FilesMatch>

<IfModule alias_module>
    Alias /phpmyadmin "C:/xampp/phpMyAdmin/"
    <Directory "C:/xampp/phpMyAdmin">
        AllowOverride AuthConfig
        Require local
        ErrorDocument 403 /error/XAMPP_FORBIDDEN.html.var
    </Directory>
</IfModule>
```

Beispiel: httpd-vhosts.conf

```
<VirtualHost *:80>
    DocumentRoot "C:/xampp/htdocs2"
    ServerName localhost2
    # Zugriffsrechte setzen, ...
</VirtualHost>


<VirtualHost *:80>
    DocumentRoot "C:/vhost/laravel/public"
    ServerName laravel.localhost
    ServerAlias www.laravel.example.com
    ErrorLog "logs/laravel-error.log"
    CustomLog "logs/laravel-access.log" common
</VirtualHost>
```

- Damit es auf einem lokalen Rechner funktioniert muss auch die etc/hosts-Datei geändert werden (Windows: `C:\Windows\System32\drivers\etc\hosts`)
- Mit Administrator-Rechten öffnen und eintragen:

```
127.0.0.1 localhost2
127.0.0.1 laravel.localhost
```

mod_userdir

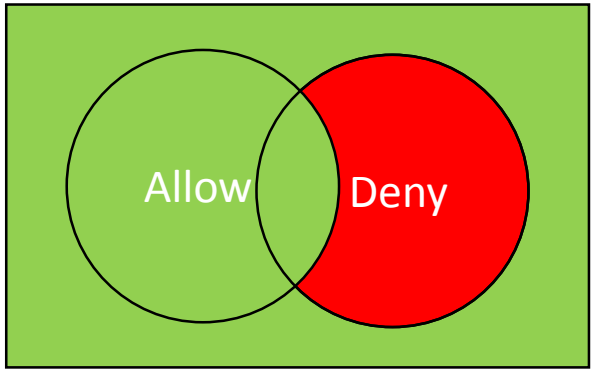
- Nutzer sollen ggf. die Möglichkeit erhalten, Web-Inhalte beizusteuern
- mod_userdir erlaubt es, jedem Benutzer ein spezifisches Verzeichnis für eigene Web-Inhalte bereitzustellen

- Beispiel-Direktive: `UserDir public_html`
  - > Jeder Systembenutzer kann nun die eigenen Dokumente in dem Verzeichnis *public_html* in seinem Heimatverzeichnis ablegen
  - > Auf `/home/thomas/public_html/file.html` kann nun per `http://example.com/~thomas/file.html` zugegriffen werden

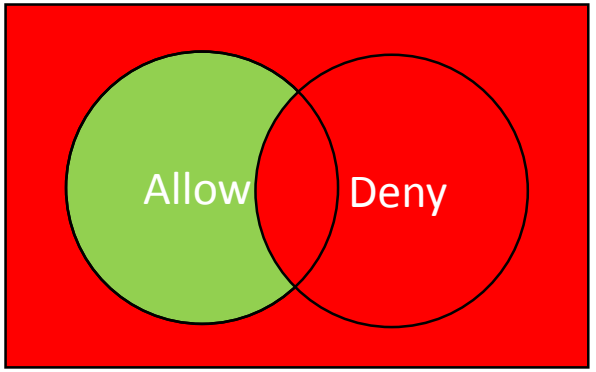
Zugriffskontrolle (Require seit Apache 2.4)

- Order Deny, Allow
  - > First, all *Deny* directives are evaluated; if any match, the request is denied **unless** it also matches an *Allow* directive. Any requests which do not match any *Allow* or *Deny* directives are permitted.
- Order Allow, Deny
  - > First, all *Allow* directives are evaluated; at least one must match, or the request is rejected. Next, all *Deny* directives are evaluated. If any matches, the request is rejected. Last, any requests which do not match an *Allow* or a *Deny* directive are denied by default.

Order Deny, Allow



Order Allow, Deny



Weitere Informationen (& Quelle der Texte): https://httpd.apache.org/docs/2.4/mod/mod_access_compat.html

Zugriffskontrolle

- Häufig „Deny,Allow“ zu finden mit explizieter „Deny from All“-Angabe
- Beispiele
  - > Alle aus einem bestimmten Subnetz haben Zugriff (+ eine bestimmte IP)

```
<Directory /home/apache/restricted>
    Order Deny,Allow
    Deny from All
    Allow from 138.5.201.99
    Allow from 140.221.0.0/16
</Directory>
```

  - > Auch Domänen können angegeben werden
    - Apache macht dann einen Reverse-DNS-Lookup

```
Order Allow,Deny
Allow from apache.org
Deny from foo.apache.org
```

.htaccess

- Erlaubt es Direktiven innerhalb von Verzeichnissen zu überschreiben, ohne das hierfür der Server neugestartet werden muss
- Häufig wird dies genutzt um z.B. den Zugriff zu beschränken (z.B. auf das Intranet oder bestimmte IP-Adressen)
- Es muss also nicht jegliche Konfiguration in Konfigurationsdateien geschrieben werden, sondern dies kann auch über .htaccess passieren

HTTP bietet kein eigenes Konzept zur Datensicherheit

- Absicherung der Übertragung der Daten zwischen Client und Server durch kryptographische Verschlüsselung
  - > HTTP über TLS/SSL (HTTPS)
  - > Dazu im Sicherheitsteil mehr (Zertifikate, Private/Public-Keys, …)

- Es werden allerdings Verfahren zur Einschränkung des Zugriffs durch Authentifizierung des Klienten bereitgestellt:
  - > Basic Authentication
  - > Digest Authentication

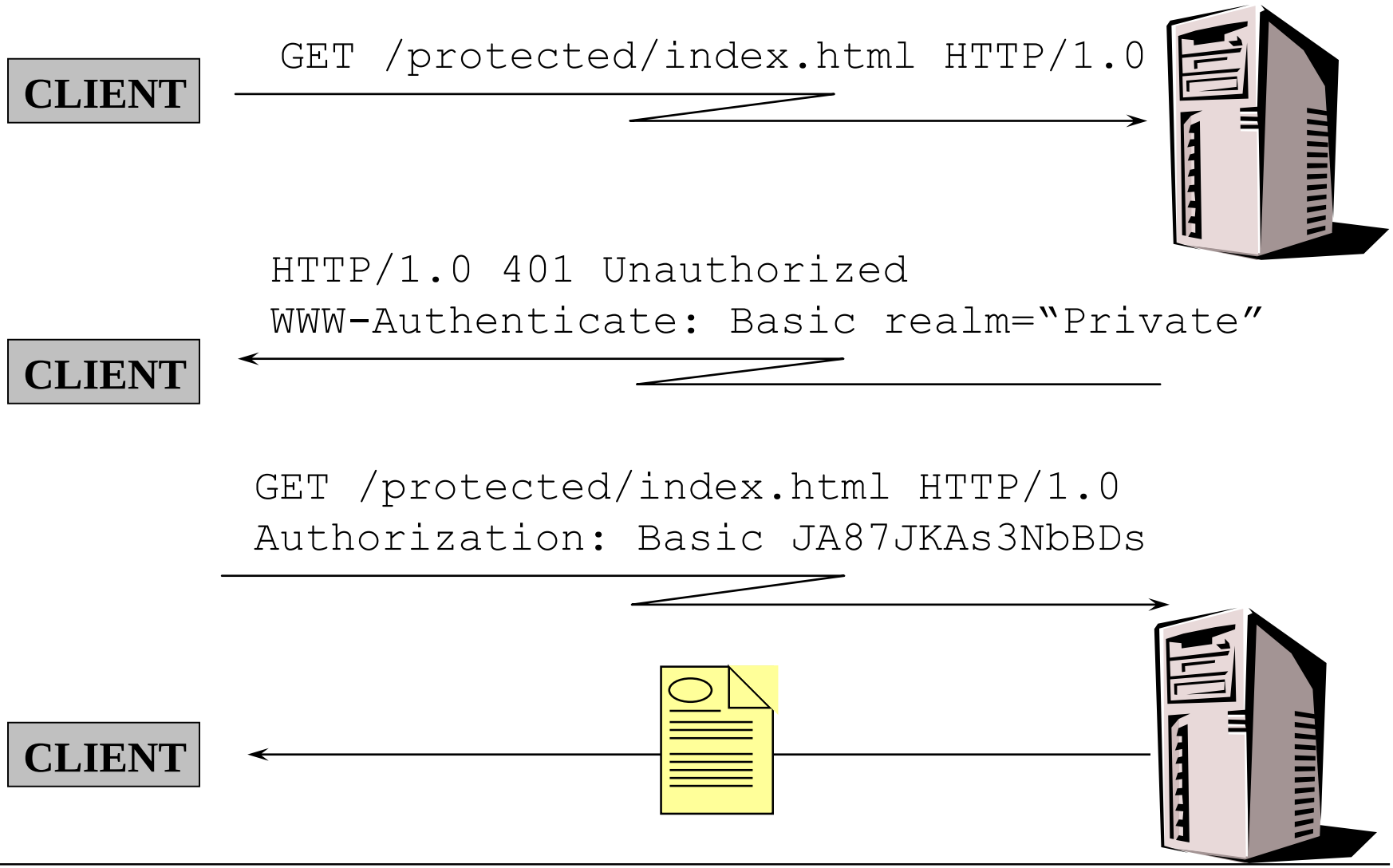
Basic Authentication Scheme

- Authentifizierungsmethode seit HTTP 1.0
- Übertragung von Benutzerkennung und Passwort
- Passwort kann sogar unverschlüsselt sein
- Autorisierung basiert auf passwd-ähnlicher Datei (crypt)

Digest Authentication Scheme

- Verbesserung in HTTP 1.1
- Server überträgt eine zufällig ausgewählte Zeichenkette (Challenge)
- Benutzer verwendet sein Kennwort um hierzu eine Signatur (Digest) zu erzeugen
- Server kennt das Passwort und prüft ob das verwendete Kennwort korrekt war

Server
Basic Authentication



Server
Basic Authentication

Beispiel

- (Annahme: AllowOverride-Direktive erlaubt den Einsatz von .htaccess)
- Anlegen einer .htaccess Datei im zu schützenden Verzeichnis

```
AuthUserFile c:\xampp\htdocs\protected\.htpasswd
AuthType Basic
AuthName "Der Zugriff auf die Kursseiten ist geschützt"
require valid-user
```

- Einfügen der Nutzer mit dem htpasswd-Kommando

```
htpasswd -d -c c:\xampp\htdocs\protected\.htpasswd user
```

AuthUserFile-Direktive

- Gibt Verzeichnis der Datei mit Benutzernamen und Passwörtern an
- Ein Datensatz pro Zeile:
  - > Benutzername:Passwort (ggf. Einwegfunktion)
- Beispiel: `AuthUserFile /usr/etc/.htpasswd`
- Pfadangabe am Anfang:
  - > / Pfad absolut interpretiert
  - > Ohne /Pfad relativ zu ServerRoot
  - > Windows: mit „X:“ auch andere Platte möglich

- Autorisierung von Gruppen:

```
AuthGroupFile /usr/etc/.htgroups
```

- Dateispezifische Regeln

```
<Files *.htm>
    require user sander hoffmann
    require group dozent
</Files>
```

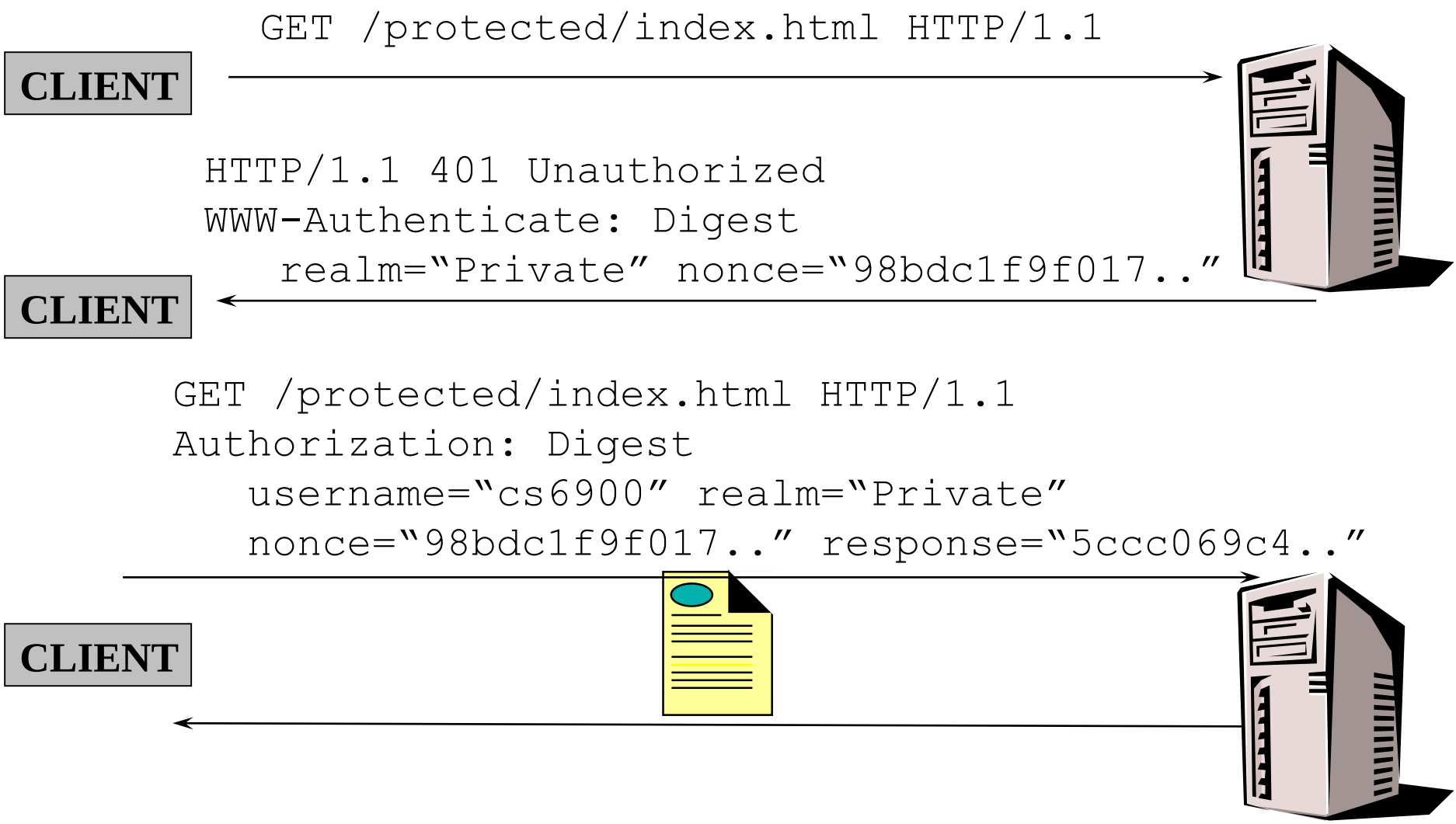
Probleme von Basic Authentication

- Passwörter werden im Klartext übermittelt
  > Leicht für Mittelsmänner abzufangen
- Keinerlei Authentisierung des Servers
  > Offen für Spoofing Attacken
- Keine Absicherung der Nachrichten
  > Man-in-the-Middle Attacken

- Lässt sich beheben, wenn HTTPS genutzt wird

Mit HTTP 1.1 eine weitere Alternative: Digest Authentication

Server
Digest Authentication



Server
Digest Authentication

Challenge (“nonce”): Frei wählbare Zeichenkette

- `MD5(IP address:timestamp:server secret)`

Response: Challenge signiert mit Benutzername & Passwort

- `MD5(MD5(name:realm:password):nonce:MD5(request))`

Server-spezifische Optionen

- Einmalige nonces
- Zeitstempel basierte nonces

URL + Realm definieren den geschützten Bereich

- Innerhalb dieses Bereichs kann mit den gleichen Credentials gearbeitet werden, solange sie gültig sind

Server
Digest Authentication

Nutzung

- Httpd.conf:

```
LoadModule auth_digest_module modules/mod_auth_digest.so
```

- Anlegen einer .htaccess Datei im zu schützenden Verzeichnis
  > (Annahme: AllowOverride-Direktive erlaubt den Einsatz von .htaccess)

```
AuthDigestFile c:\apachefriends\xampp\htdocs\.htpasswd.di
AuthType Digest
AuthName "Vorlesung"
require valid-user
```

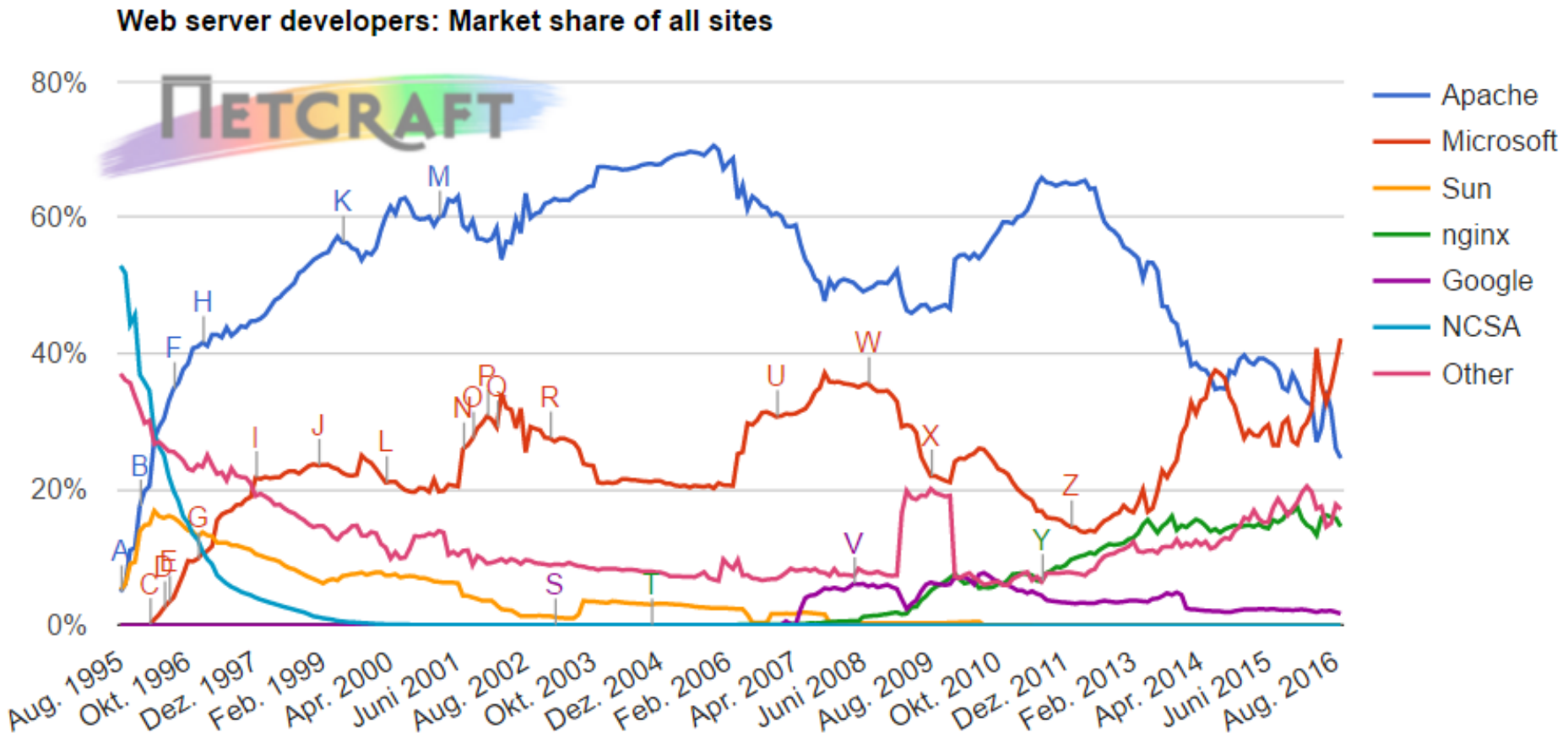
- Einfügen der Nutzer mit dem htdigest-Kommando

```
htdigest -c c:\apachefriends\xampp\htdocs\.htpasswd.di
  Vorlesung user
```

Vorteile gegenüber Basic Auth.

- Keine Passwörter im Klartext
- Keine Abspeicherung von Klartextpasswörtern beim Server
- Server wird authentisiert
- Ansonsten wird nicht viel gewonnen
  - > Man-in-the-middle-Attacken
  - > Sniffing-Attacken

Server
Marktanteile



Marktanteile Top-Server über alle Domains Aug. 1995 – Aug. 2016. Quelle: netcraft.com

nginx („engine-ex“)

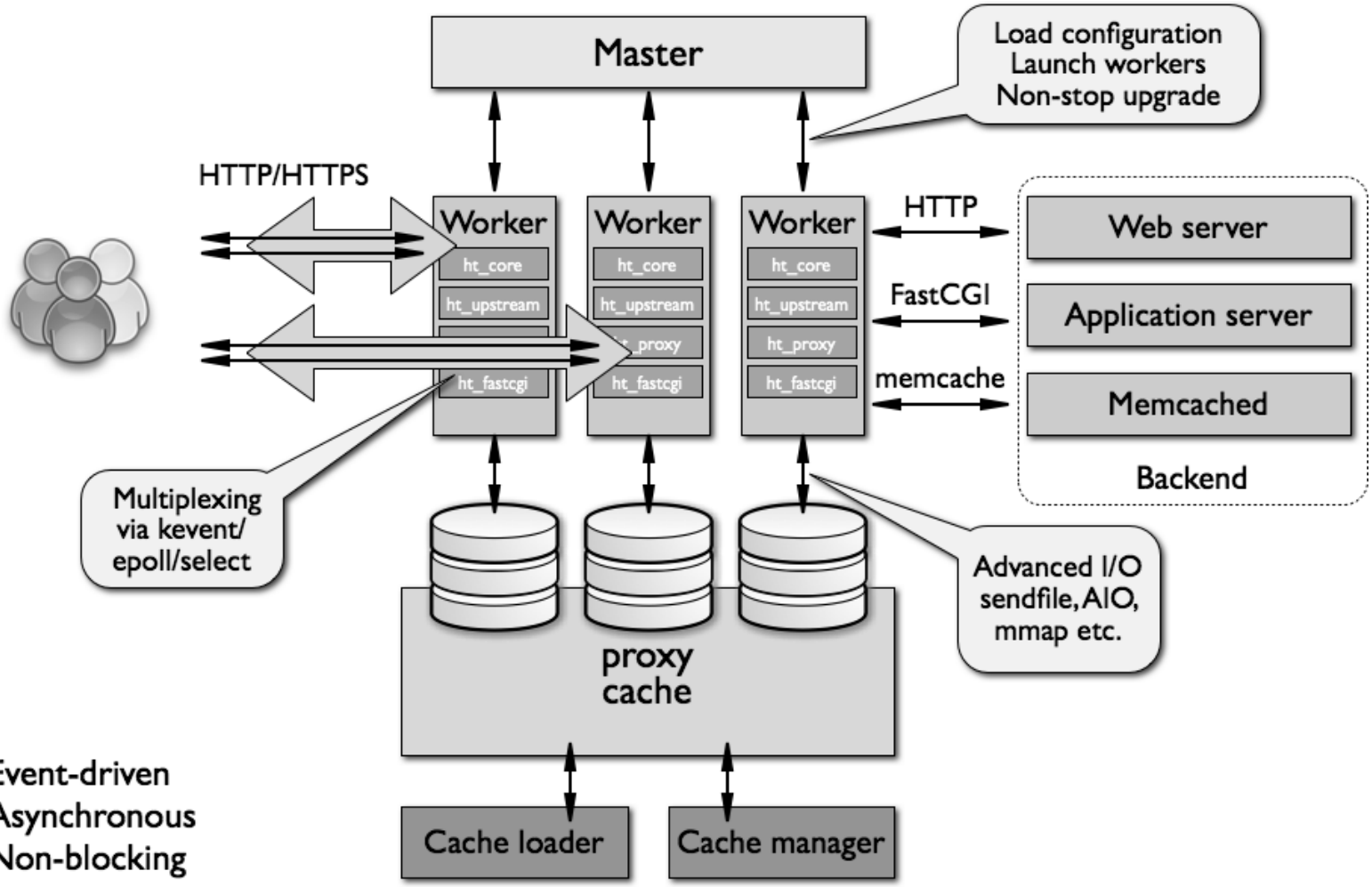
Der von Igor Sysoev ursprünglich für eine russische Suchmaschine entwickelte NGINX-Webserver besitzt besondere Eigenschaften:

- Hohe Performance
- Geringer Memory-Footprint
- Reverse Proxy
- E-Mail Proxy
- Erweiterbar durch Module

Für einfache Webserver, die nur selten Anpassungen erfordern und für Infrastruktur mit eingeschränkter Hardware (embedded Systeme) bietet NGINX große Vorteile.

Zusammen mit dem Memcached-Key-Value-Store-Module ergibt sich ein extrem leistungsfähiges System.

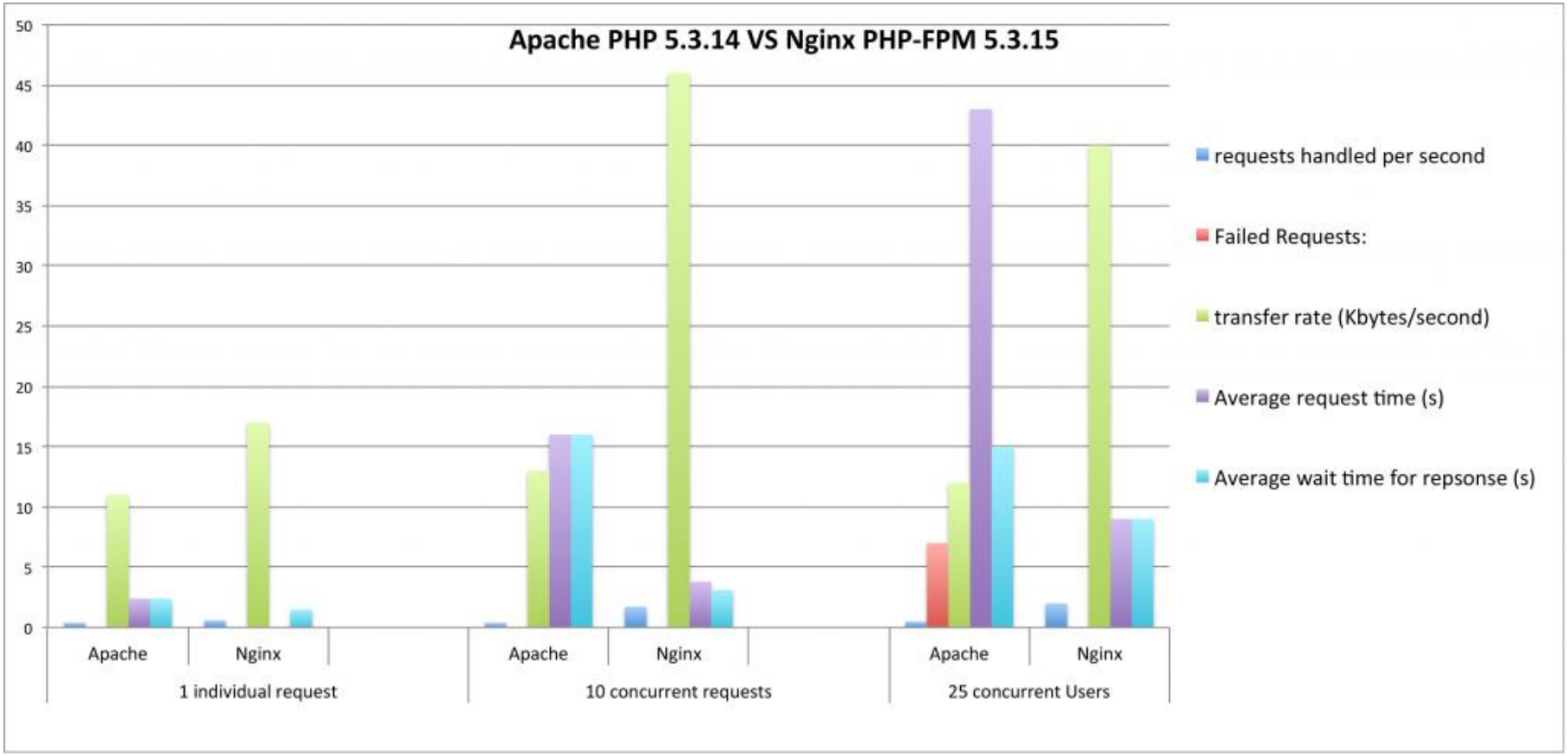
nginx



Quelle: http://www.aosabook.org/en/nginx.html

nginx vs. Apache

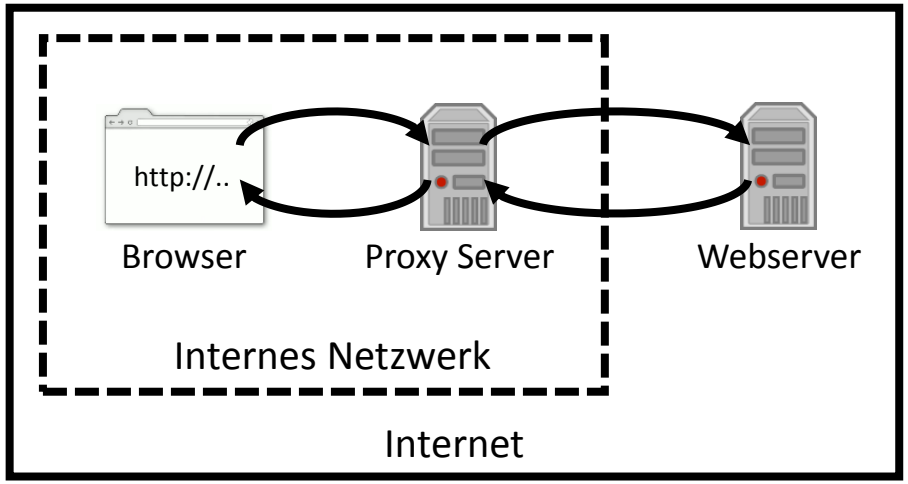
„Basically, it's 4.2 times as fast as Apache on these tests by average, and a lot more reliable"



Quelle: http://www.theorganicagency.com/blog/apache-vs-nginx-performance-comparison

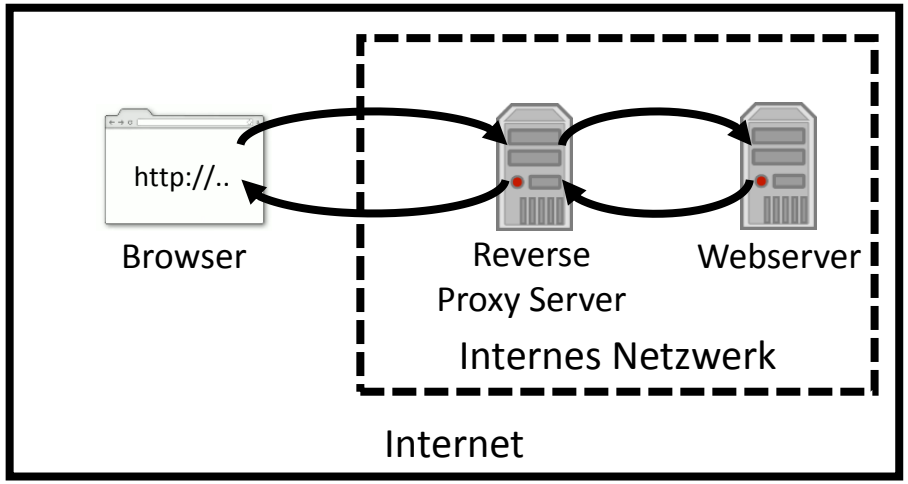
Exkurs: Normaler (Forward) Proxy

- Server, hinter dem sich der Client „versteckt“
- Aus Sicht des Webservers, ist der Proxy der Client
- Anwendungen:
  - > Anonymisierung
  - > Firewall
  - > TLS-Terminierung (keine Verschlüsselung im internet Netzwerk)
  - > Werbefilter
  - > Caching



Reverse Proxy

- Server, hinter dem sich ein anderer Server versteckt
- Der Client sieht nur den Reverse Proxy Server
- Anwendungen:
  - > Sicherheit/Firewall
  - > Verschlüsselung über den Reverse Proxy
  - > Load Balancing (hinter dem Proxy können mehrere andere Server liegen)
  - > Caching / Content Delivery
  - > Authentifizierung



Beispiel: nginx.conf

```
user nginx;
worker_processes 5;
error_log logs/error.log;

http {
  include    conf/mime.types;
  include    /etc/nginx/proxy.conf;
  include    /etc/nginx/fastcgi.conf;
  index      index.html index.htm index.php;

  server {  # simple static file server
    listen       80;
    server_name  static.example.com;
    root         /var/www/static;

    location / {
      autoindex on;
    }
  }

  server { # php with fastcgi
    listen       80;
    server_name  example.com www.example.com;
    root         html;

    location ~ \.php$ {
      fastcgi_pass   127.0.0.1:9000;
    }
  }
```

„Beginner‘s Guide“: http://nginx.org/en/docs/beginners_guide.html

Beispiel: nginx.conf (Fortsetzung)

```
  server { # reverse-proxy
    listen       80;
    server_name  www1.example.com www2.example.com;

    location ~ ^/(images|javascript|js|css|flash|media|static)/  { # serve static files
      root    /var/www/example.com/htdocs;
    }

    location / { # pass requests for dynamic content to another application
      proxy_pass      http://127.0.0.1:8080;
    }
  }

  # load balancing
  upstream big_server {
    server instance1.example.com weight=3;
    server instance2.example.com;
    server instance3.example.com;
  }
  server {
    listen          80;
    server_name     app.example.com;

    location / {
      proxy_pass      http://big_server;
    }
  }
}
```

Weitere Beispiele: https://www.nginx.com/resources/wiki/start/topics/examples/full/