UCSPI-XXL

DAS UNIX CLIENT/SERVER PROGRAM

INTERFACE FÜR IPV6 UND TLS

UCSPI-xxl:

- Das Unix Client/Server Program Interface UCSPI ist historisch als Alternative zum Unix **inetd** Dienstes bzw. bei Solaris für **mconnect** gedacht und von Dan Bernstein (djb) um 1995 als **ucspi-tls** entwickelt worden. Es steht heute in der *Public Domain*.

- Zentraler Baustein ist der **tcpserver**, der sowohl ein *Socket-Interface* für die Netzkommunikation bereitstellt, als auch *Filedeskriptoren* für die Anwendung. Über eine *CDB* (*constant data base*) kann **tcpserver** wie ein *Paketfilter* konfiguriert werden.

UCSPI-xxl - die Probleme:

- **ucspi-tcp** kennt nur IPv4 Sockets. IPv4-Adressen können nur im Standardformat (‚dotted-decimal' Notation) genutzt werden; keine *CIDR* Unterstützung.

- **ucspi-tcp** steht nur für IPv4 zur Verfügung.
  - Patches existieren (Felix von Leitner/fefe).

- **ucspi-tcp** kennt keine verschlüsselten Verbindungen via TLS/SSL.
  - Von *Superscript* existiert aber das Pendant **ucspi-ssl**.

- Keine AMD64-Unterstützung, kein Clang-Support …

UCSPI-xxl - die Lösung:

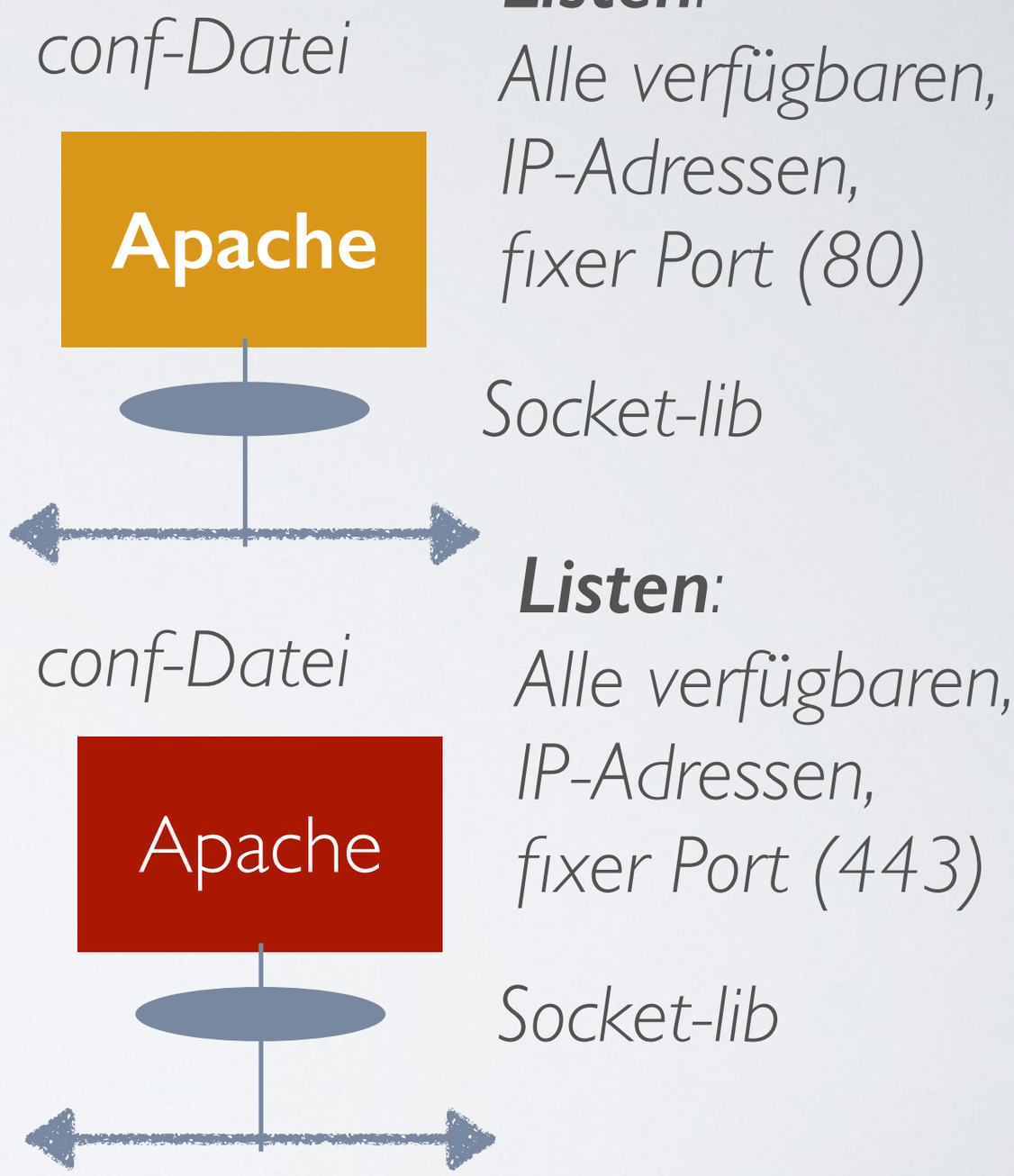
- **ucspi-tcp** wird ums die fehlenden Funktionen erweitert,
  - in das */slashpacket* Format überführt und
  - als **ucspi-tcp6** veröffentlicht.
- Zugleich wird **ucspi-ssl** um IPv6-Funktionen
  - auf gleicher Code-Basis weiterentwickelt.

☞ Mit **ucspi-tcp6** 1.0 und **ucspi-ssl** 0.94 stehen aktuelle Pakete zur Verfügung.

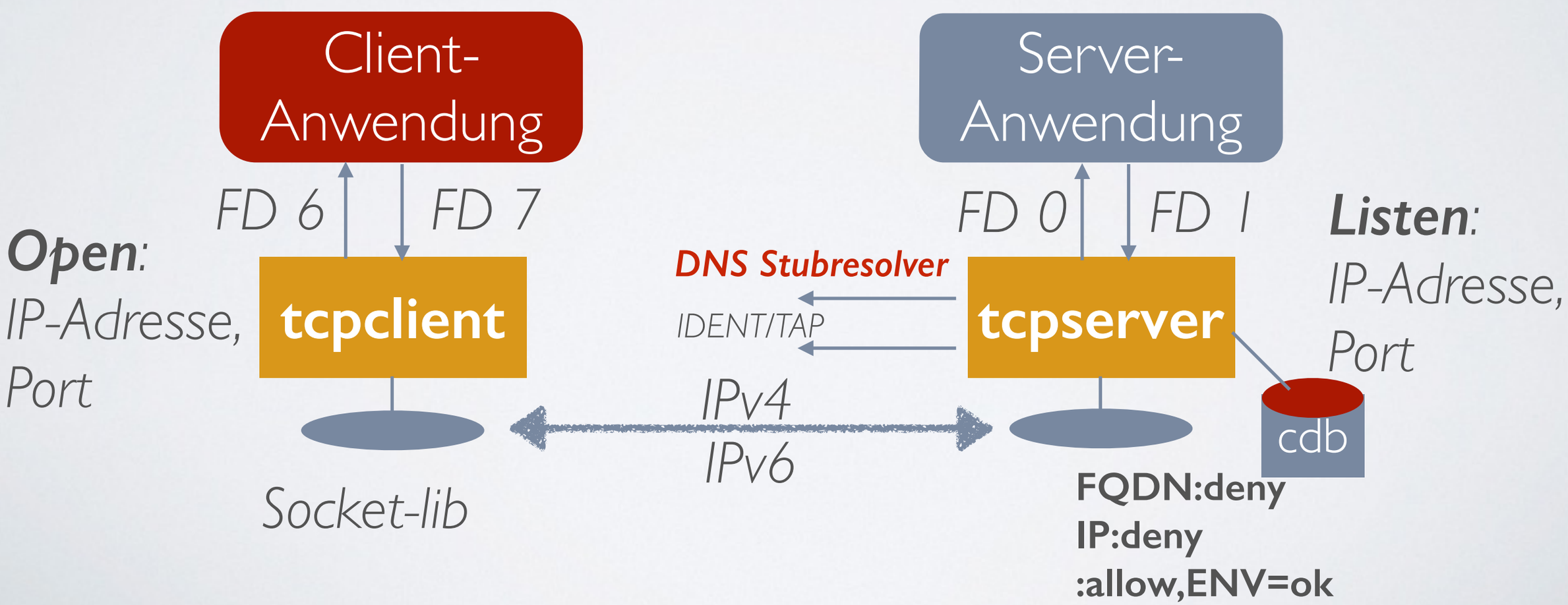
Überblick:

- Wie sieht das Konzept von ucspi-xxl aus ?
- Wie funktioniert **tcpserver**/**sslserver** ?
- Wie kann **tcpserver**/**sslserver** bei IPv6 eingesetzt werden?
- Was ist das */slashpacket* Format ?
- Hands on! Aufsetzen eines HTTP-Servers !

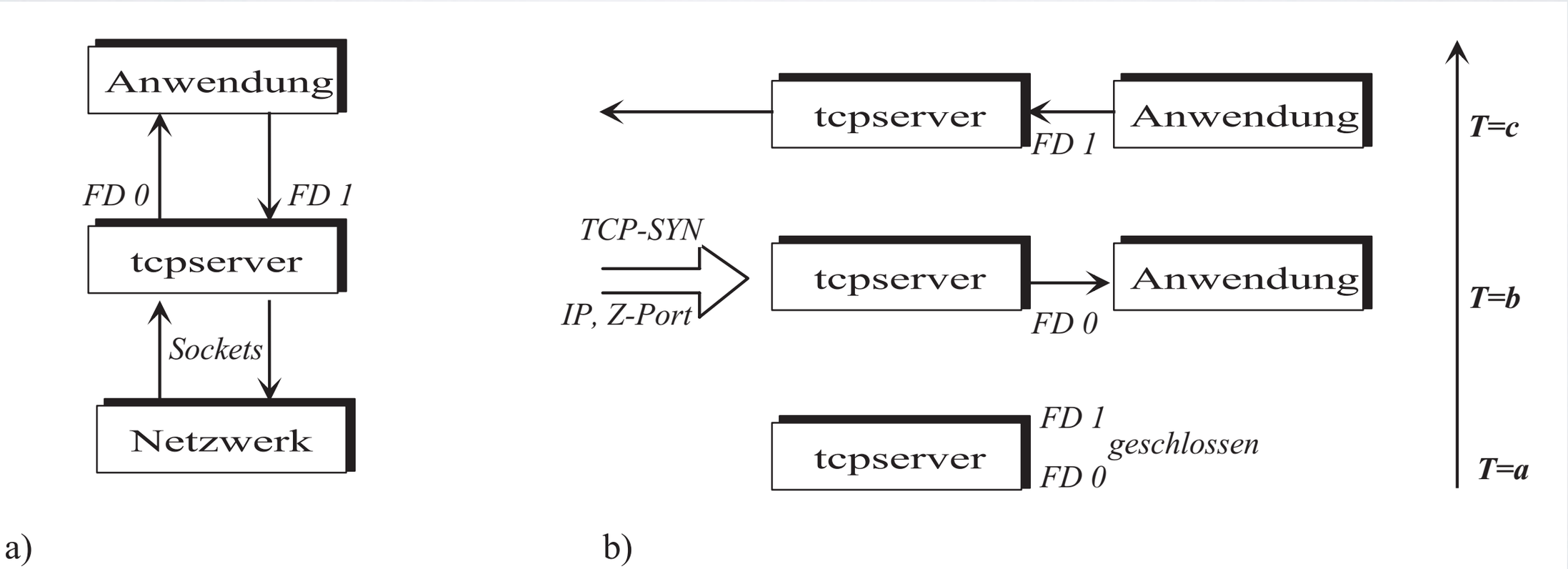
- Typische Daemons unter Unix wie HTTP sind gelinkt mit
  - der *Socket-Lib*,
  - nutzen *DNS-Stub-Resolver*,
  - greifen zurück auf *tcp-wrapper* (etc/hosts.allow),
  - sind mit *SSL-Libs* gebunden,
  - brauchen ggf. *root*-Rechte (und droppen diese),
  - legen sich selbst in den Hintergrund (*Daemons*),
  - besitzen eigenes *Logging*,
  - nutzen *SASL-Lib* für *Authentisierung*,
  - bilden ggf. *Prozessgruppe*.



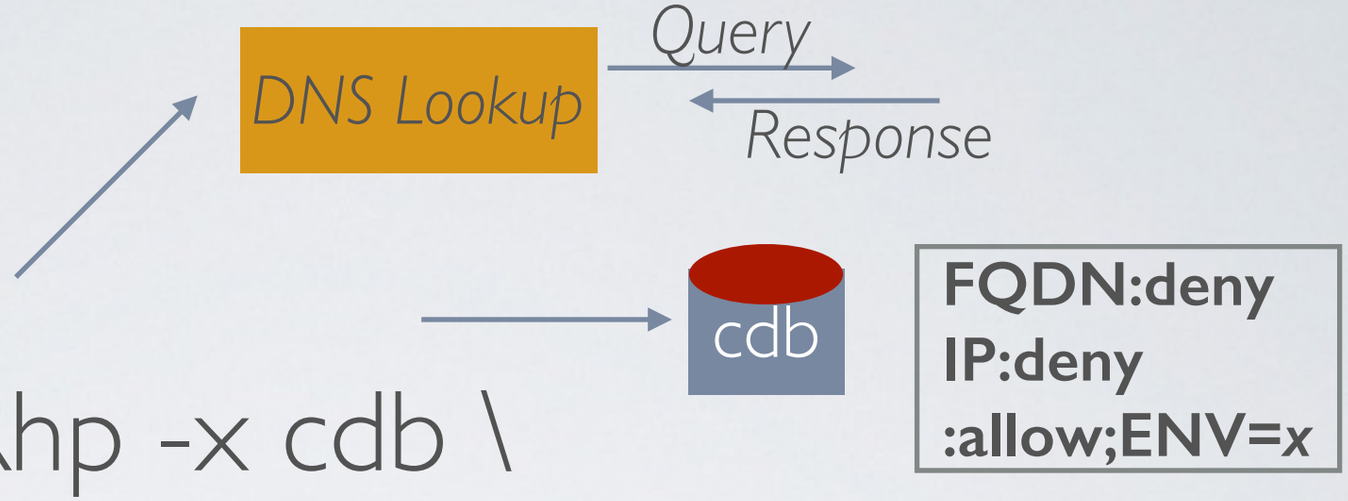
- Mittels **tcpserver/tcpclient** lassen sich Client/Server-Anwendungen realisieren, ohne dass die Anwendung ein Socket-IF besitzen muss.
  - TCP-Verbindungen können auf der **tcpserver**-Seite per IP-Adresse oder per FQDN kontrolliert werden.
  - Applikation läuft in einer *chroot*-Umgebung.



- Für jede Verbindung wird **tcpserver** bei einlaufendem <SYN> auf der konfigurieren IP Adresse geforked.
  - Die max. Anzahl der **tcpserver**-Instanzen kann vorgeben werden.



- Aufruf:

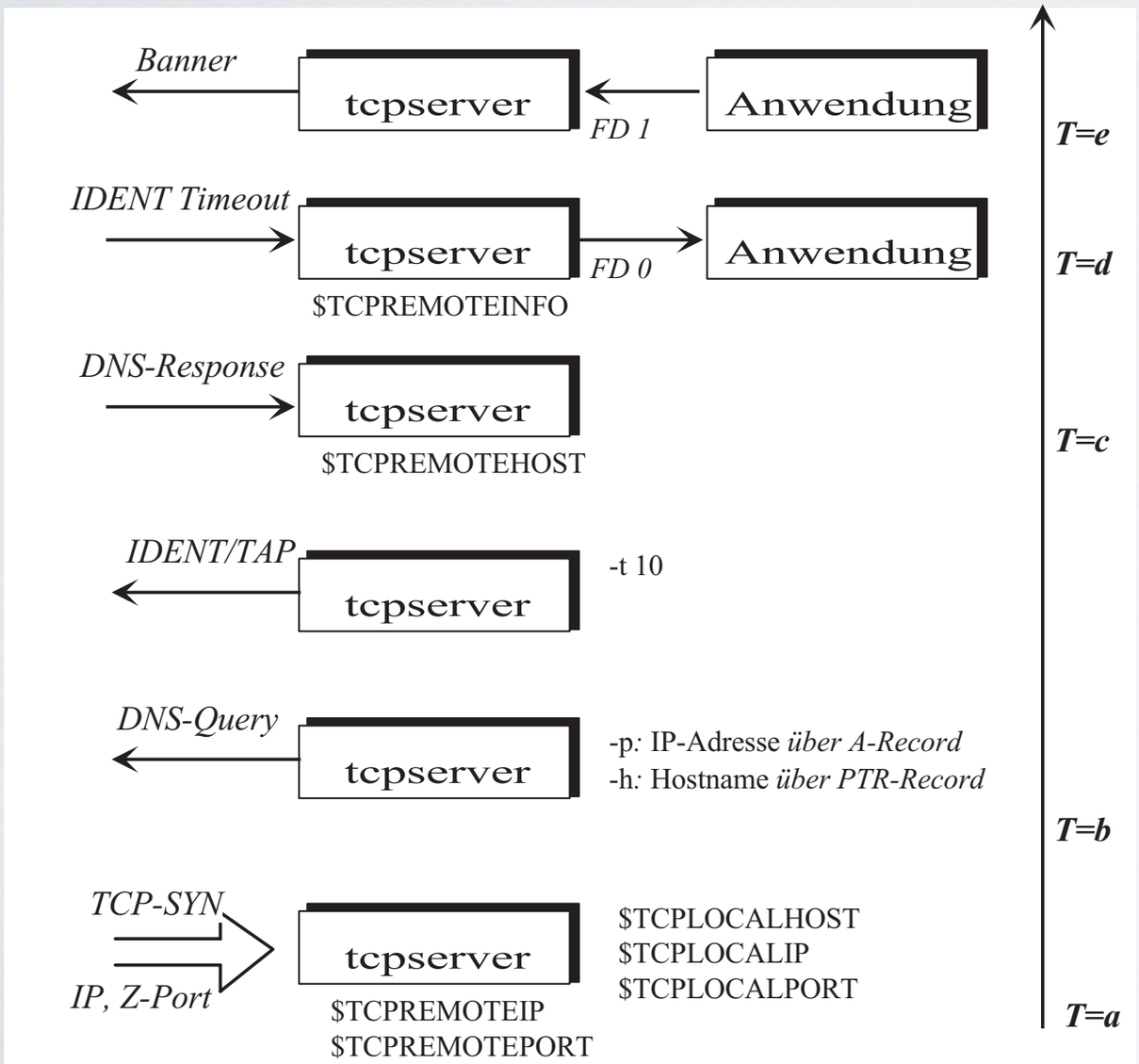


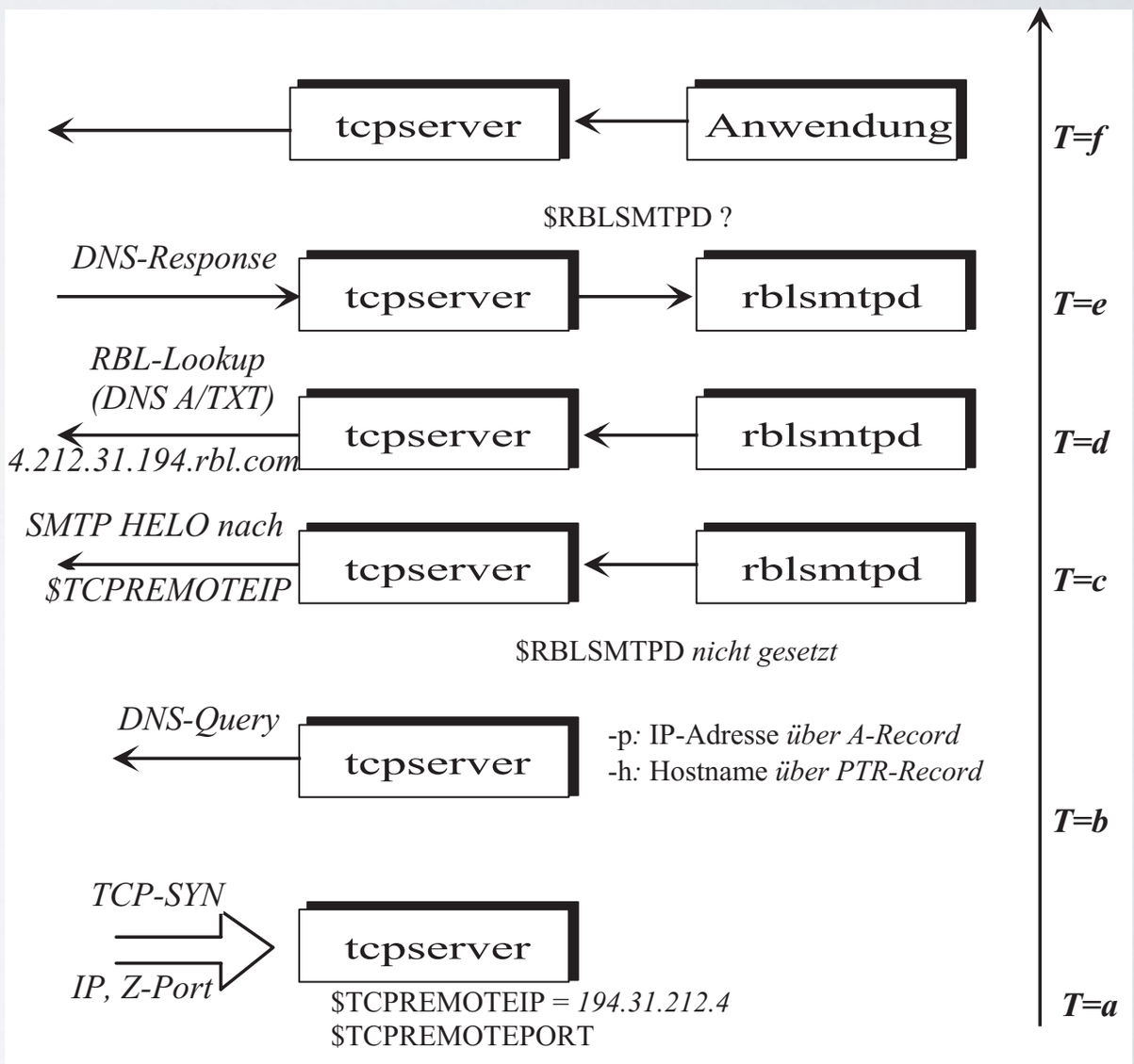
  - **tcpserver** -v -Rhp -x cdb \
    -c connections \
    -u *user* -g *group* \
    -l *localhost* \
    IP-Adresse Port \
    progX *args* **progX server** *args*

Statische + verbindungsabhängige Environment-Variablen

- **tcpserver** wird pro Applikation immer explizit auf eine IP-Adresse gebunden; durch Angabe von ‚0' auf alle verfügbaren.

- Standard-Aufruf von **tcpserver** und gesetzte Environment-Variablen.





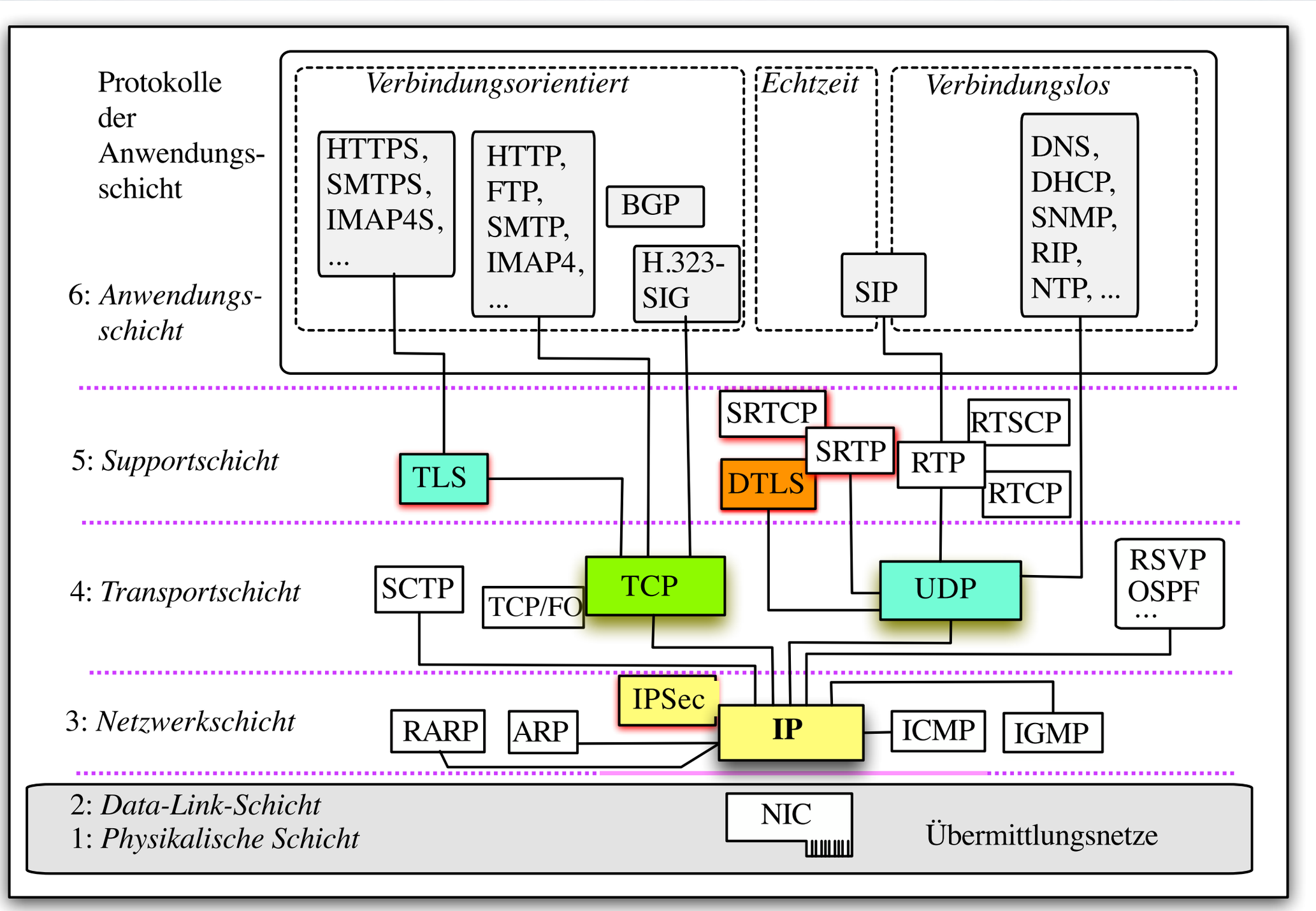
- IPv6 Erweiterungen für **tcpserver**:
  - **tcpserver** *-4 | -6 -I ifname* -Rhp -x cdb \
    -c connections \
    -u *user* -g *group* \
    -l *localhost* \
    IP-Adresse Port \
    progX *args* **progX server** *args*

Default ist immer IPv6 !

- CIDR Unterstützung:
  - `192.168/16:allow;RELAYCLIENT=„“`
  - `fe80::/10:allow;RELAYCLIENT=„“`
  - `fe80::1:deny`

tcpserver versteht ‚kompaktifizierte‘ IPv6-Adressen!

- *Return of the* … OSI-Model ….



- **sslserver**:
  - **sslserver** *-4 | -6 -l ifname* -Rhp -x cdb \
    -c connections \
    -u *user* -g *group* \
    -l *localhost* \
    IP-Adresse Port \

Default ist immer IPv6 !

progX *args* **progX server** *args*

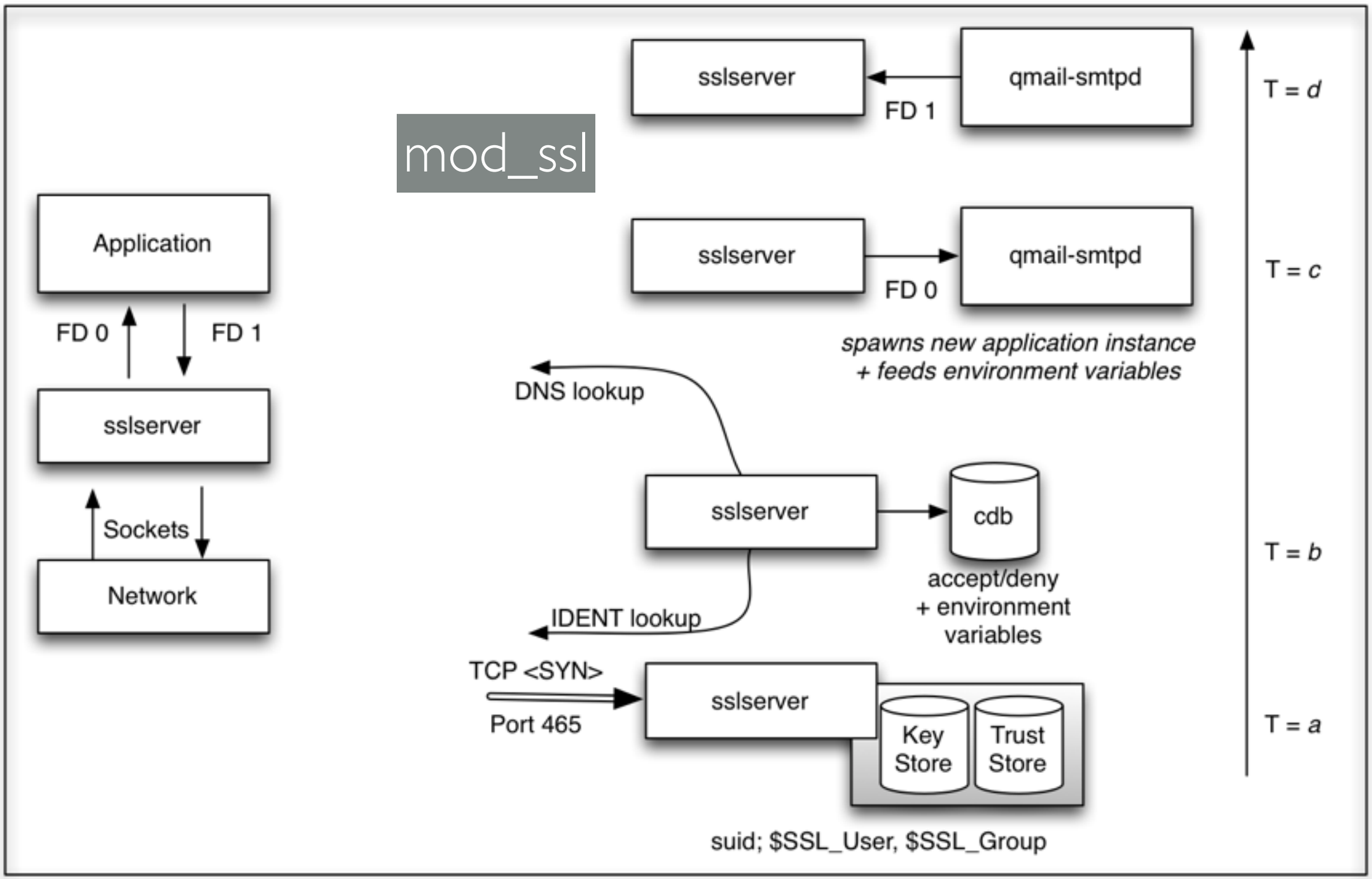
DHPARM= *CIPHERS=*

export

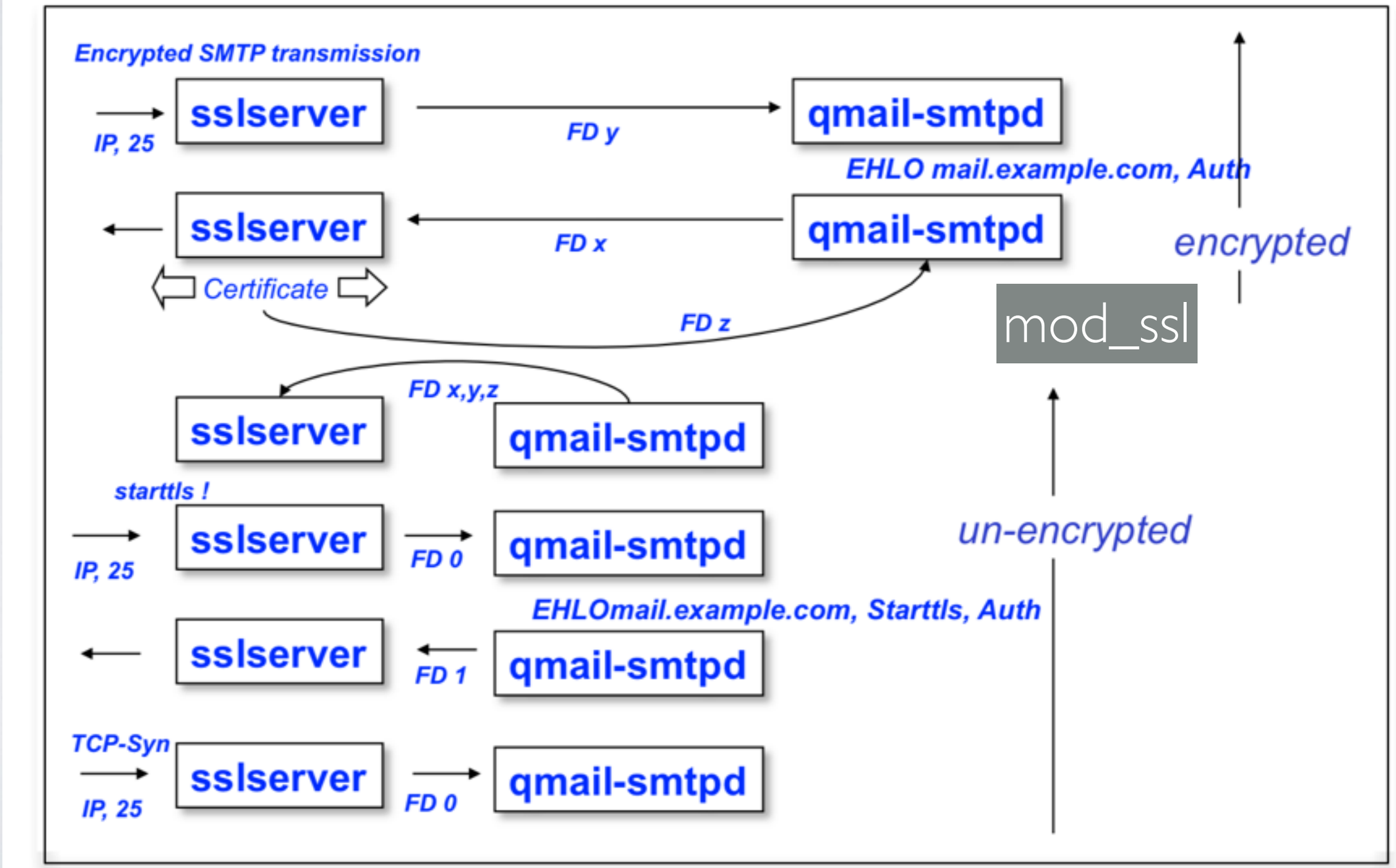
KEYFILE= *CAFILE=*
CADIR=

Keystore und Truststore können pro Verbindung gewählt werden!

- **sslserver**: Architektur für TLS



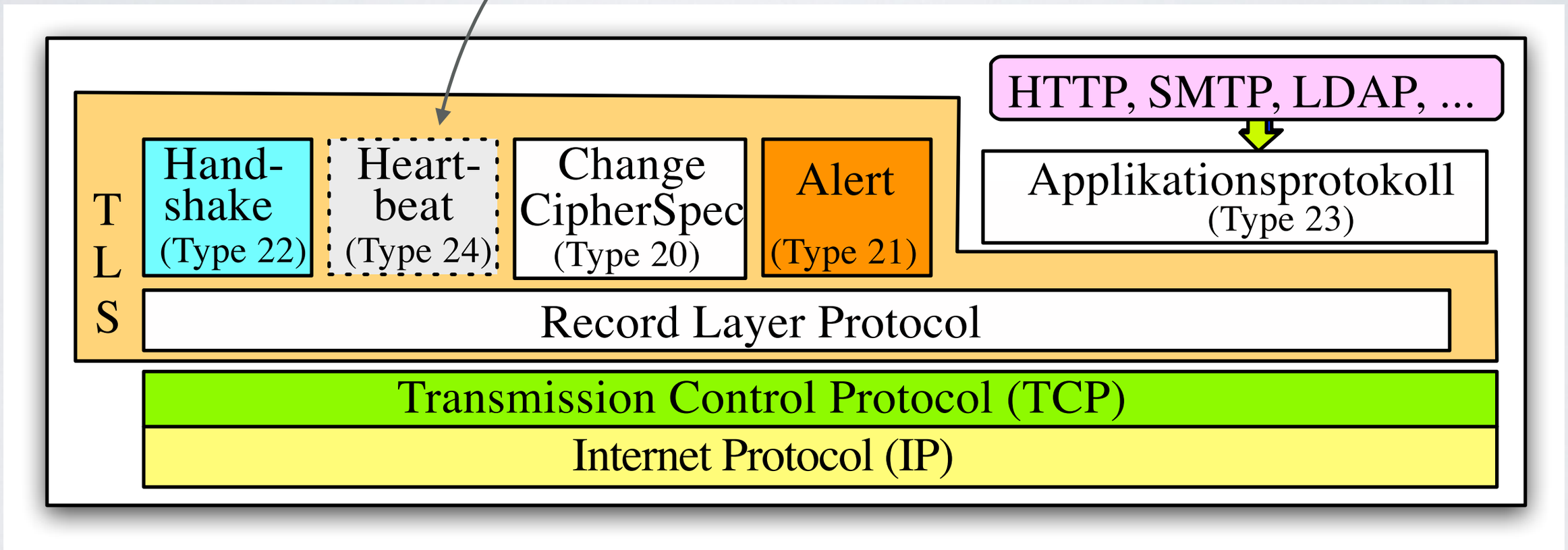
• **sslserver**: STARTTLS Unterstützung



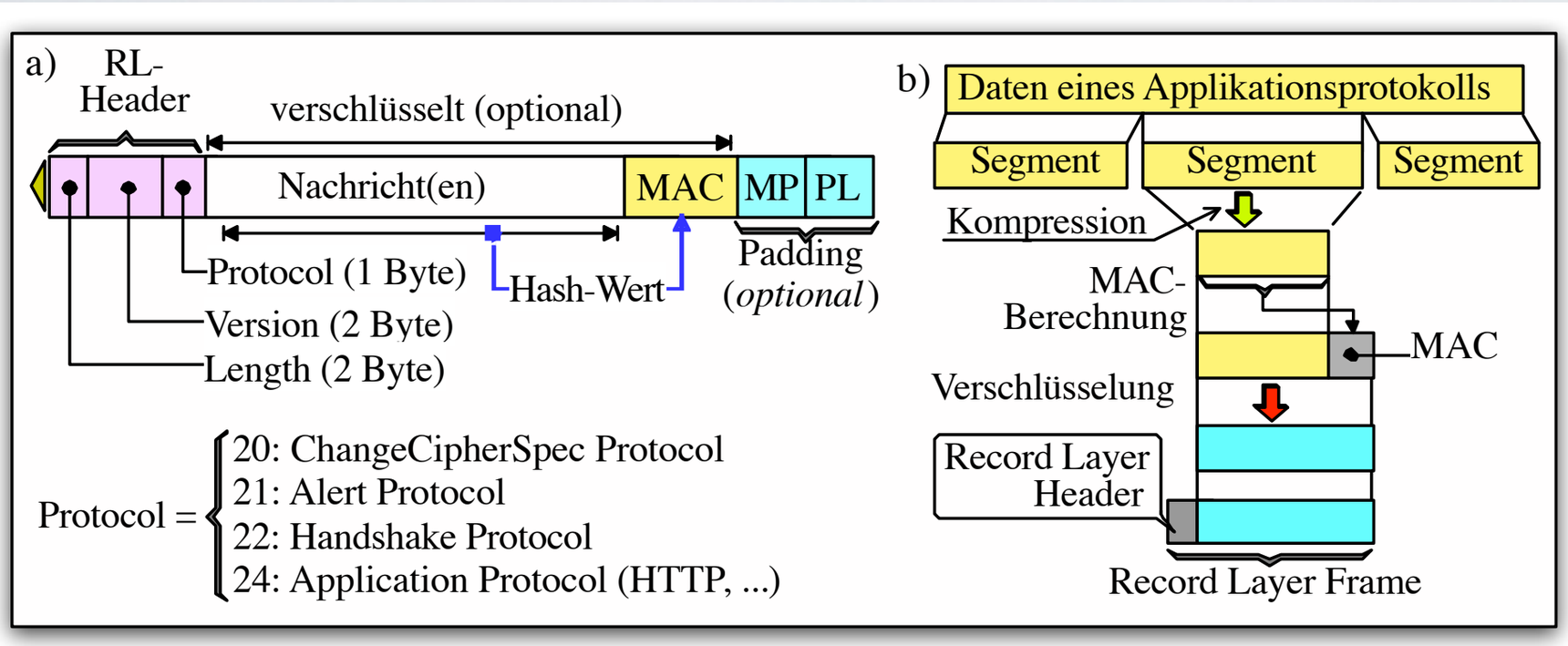
- **Aufbau von TLS:**

RFC 6520

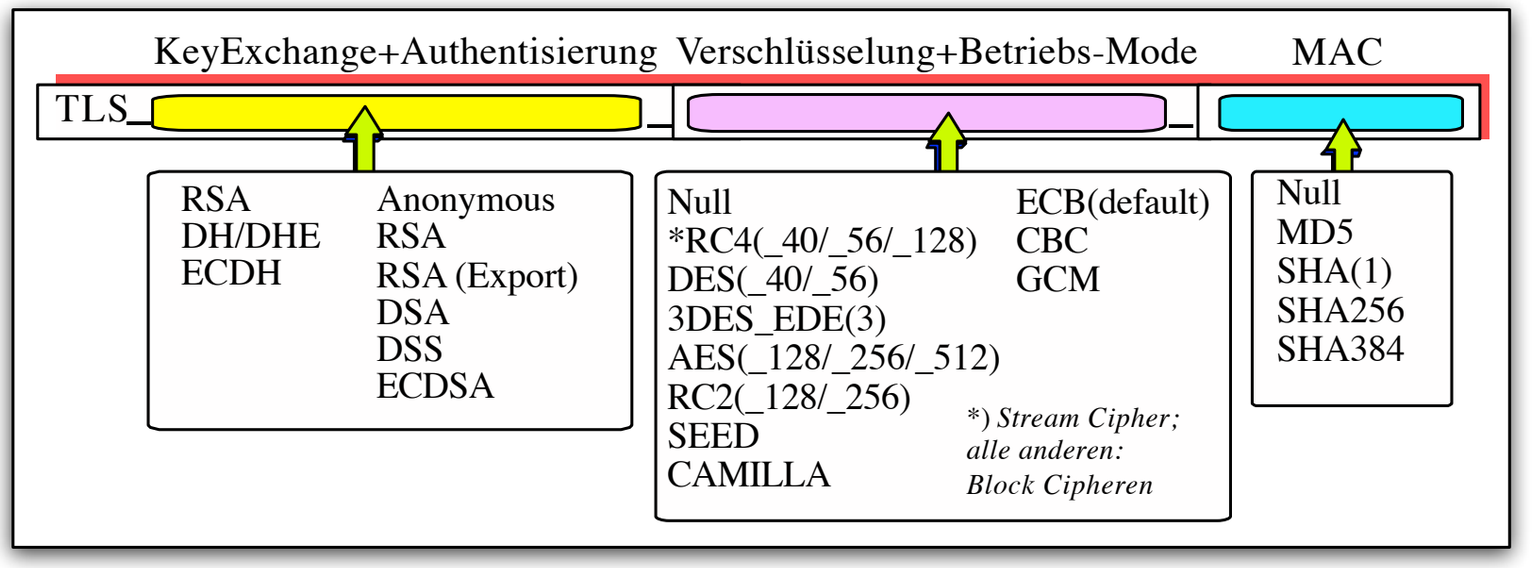
Juni 2012 - OpenSSL 1.0.1



• Aufbau von TLS Nachrichten / Cipher Suite:



Cipher-Suite:



Heartbleed Bug?

- **sslserver** hängt von OpenSSL ab.

- Daher ist **sslserver** auch vom Heartbleed Bug in OpenSSL 1.0.1 betroffen … aber

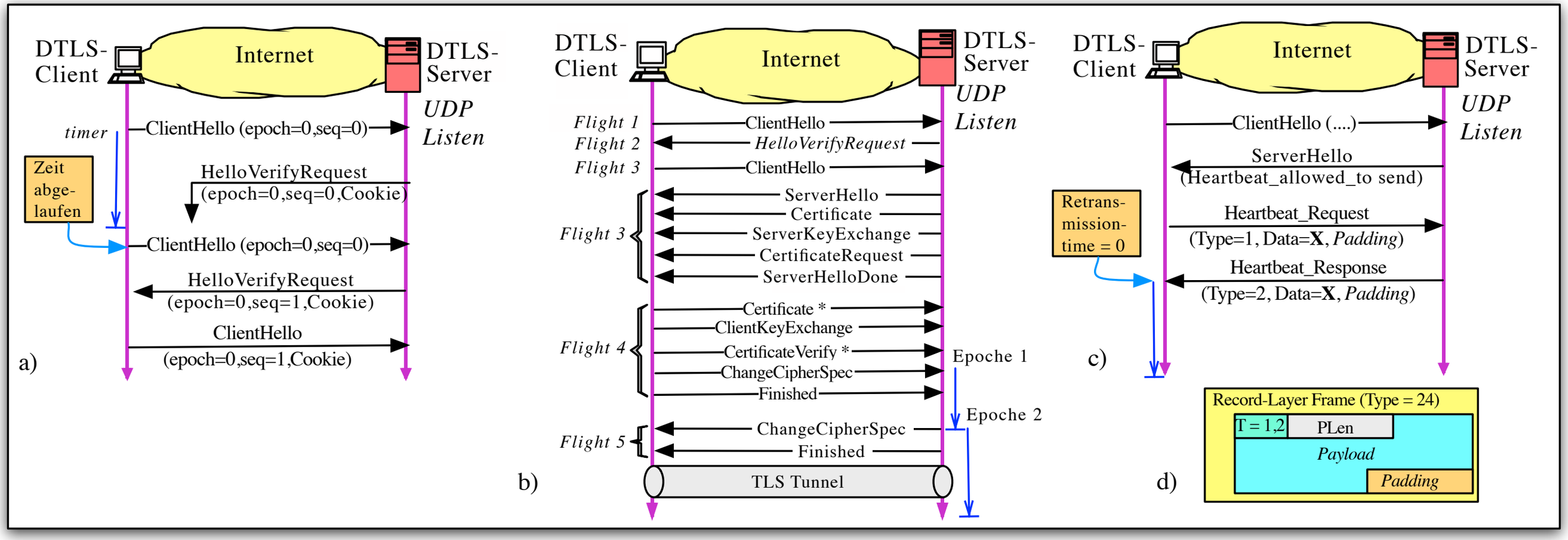
- er kann nicht ausgenutzt werden, da pro Client-IP eine **sslserver**-Instanz geöffnet wird (neuer Speicherbereich) und dieser auch nur für diesem Client.

- Nach dem Lesen der CA Certs und des Keyfiles erfolgt die eigentliche Verbindungs-ver/entschlüsselung in einer *chroot*-Umgebung statt.

- Diese werden für die aktuelle Verschlüsselung auch gar nicht benötigt; ausser zum Schlüsseltausch bei RSA (keine *Perfect Forward Secrecy* PFS).

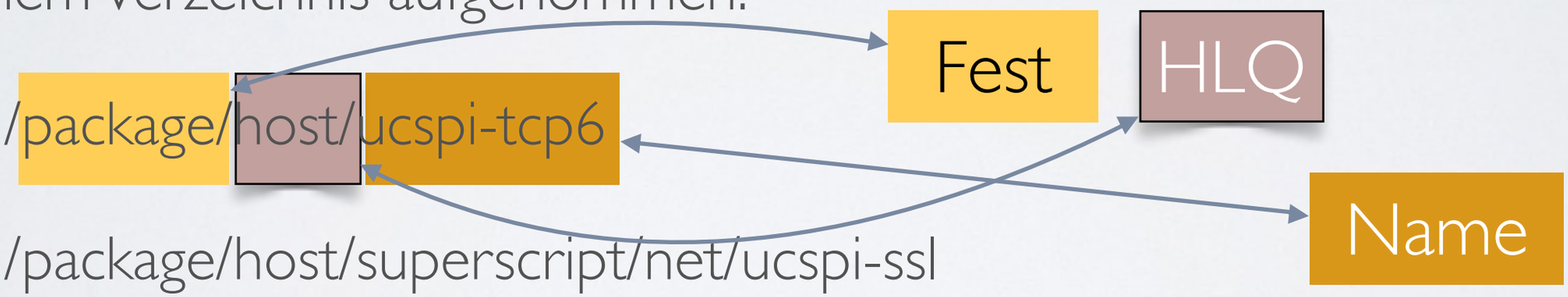
- Allerdings stehen die Certs im ***Kontext*** der SSL-Verbindung.

Datagram-TLS (DTLS): Der Heartbeat



Packetierung von ucspi-tcp6 und ucspi-ssl:

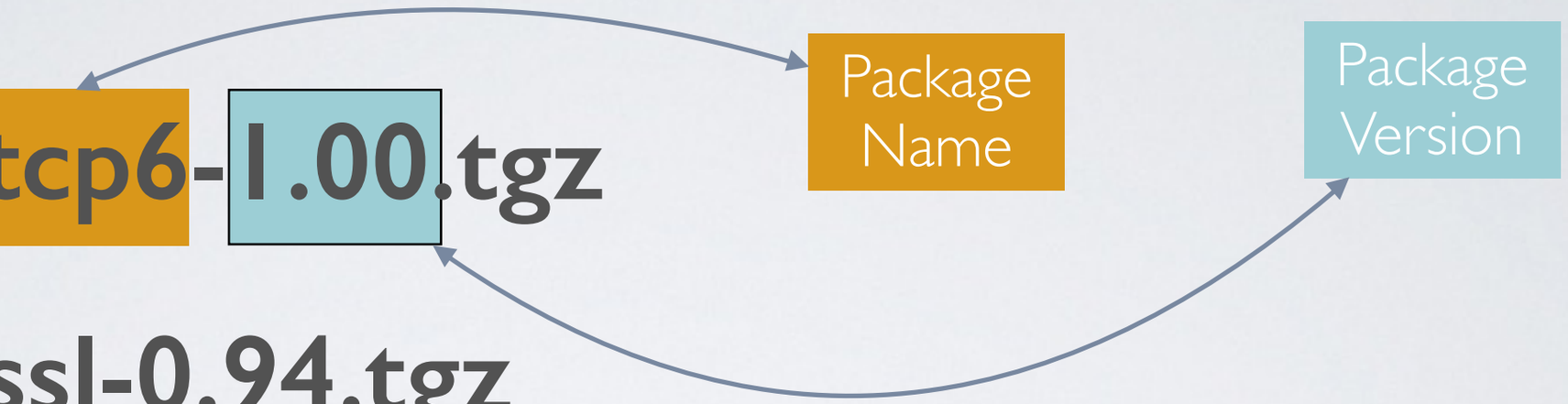
- **ucspi-tcp6** und **ucspi-ssl** nutzen die **/slashpacket** Konvention von djb:
  - Installationsverzeichnis ist **/package**
  - Jedes Package wird mit einem *eindeutigen Namen* (und Position) hierin in einem Verzeichnis aufgenommen:
    - /package/host/ucspi-tcp6
    - /package/host/superscript/net/ucspi-ssl



  - djb nimmt die Registrierung der Packages vor und reserviert den Namespace.

Installation der Packages:

- Packages kommen als tar-Archiv:
  - **ucspi-tcp6-1.00.tgz**
  - **ucspi-ssl-0.94.tgz**



Wir nutzen das Semantic Versioning.

- Packages werden unter **/package** von *root* entpackt:
  - `mkdir /package; cd /package`
  - `tar -xzf path/ucspi-ssl-0.94.tgz`

Compilieren der Packages:

- Das Package entpackt sich am vorgegebenen Ort im Verzeichnis mit Versions-Kennung:
- **ucspi-ssl-0.94.tgz ⇨../ucspi-ssl-0.94**
- Dann geht alles ganz einfach:
  - `cd /package/…/ucspi-ssl-0.94`
  - `package/install base`

Verlinkung des Packages:

- Die aktuellen Binaries werden immer unter
  - packagename/**command** abgelegt
  - Hierin findet sich also immer die aktuelle Version:

ucspi-ssl-0.94.tgz ➪../ucspi-ssl/command

- Von **ucspi-ssl/command** werden Symlinks nach**/usr/local/bin** gelegt:
  - `/usr/local/bin/sslserver -> /package/host/superscript.com/net/ucspi-ssl/command/sslserver`

Vorteile von Packages:

- Der Entwickler muss sich Überlegungen stellen, dass SW auf OS installiert werden kann (Compiler-Flags etc.)
- Kein ‚**configure**‘.
- Automatische Versionierung.
- Update im laufenden Betrieb:
  - Unix **install** blockiert, falls Dienst aktiv ist.

Packages unter der Haube:

- Verzeichnisstruktur:
  - ./src (Quellcode)
  - ./package (Installationsskripte)
  - ./doc (bei mir)
  - ./man (bei mir)
- Nach Kompilierung:
  - ./compile (gelinkte Quellen aus ./src + O'files)
  - ./command (ausführbare Dateien)

Packages mit Optionen:

- Partielles Compilieren:
  - **package/install** *base* | *perl*
  - **package/man** (Nachträgliche Installation)
- Tests:
  - **package/rts** (Entwickler erstellt Test-Output)
  - **package/rts** *base*
- Erfolgsreport:
  - **package/report** (email mit OS-Parms an *feh@fehcom.de*)

Build der Packages:

- Keine Dokumentation
  - Reverse-Engineering von Packages
  - Eigenes, generisches Build-Skript
- Testen der Packages problematisch
  - Verschiedenes Laufzeit-Verhalten …
  - Firewall aktiv ….
  - DNS Libraries …

Hands-On:

- Installation von ucspi-tcp6
- HTTP-Server:

```
#tcpserver -v -rh ::1 9000 ./hello
tcpserver -v -rh -x deny.cdb 2001:4dd0:ff00:8d3b::30 9000 ./
hello
```

- Hello: HTML Content:

```
#!/bin/sh
echo "HTTP/1.1 200 OK
Content-Type: text/html

<html>
<head>
<title>My title</title>
</head>

<body>
<h1>Hello in big font</h1>
</body>
</html>“
```

- Mit Web-Browser hierauf verbinden!

- **Fragen** ?
- *Antworten* !
  - http://cr.yp.to/ucspi-tcp.html
  - http://cr.yp.to/slashpackage.html
  - http://www.fehcom.de/qmail/qmailbook.html
  - http://www.fehcom.de/qmail/smtptls.html
  - http://www.fehcom.de/ipnet/ucspi-ssl.html
  - http://www.fehcom.de/ipnet/ucspi-tcp6.html
  - http://www.superscript.com/ucspi-ssl/index.html