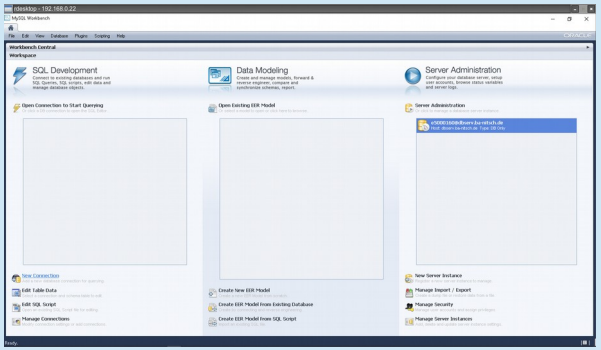
Datenbank - API

Datenbank – API
Bisher

Datenbank-
Client



Anforderung

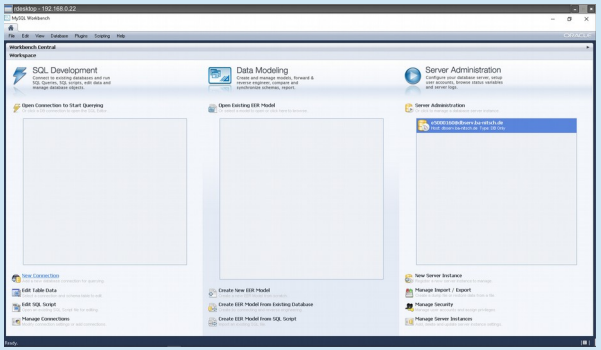
Proprietäres Protokoll

Antwort

Datenbank-
Server

Datenbank – API
Bisher

Datenbank-Client



Anforderung

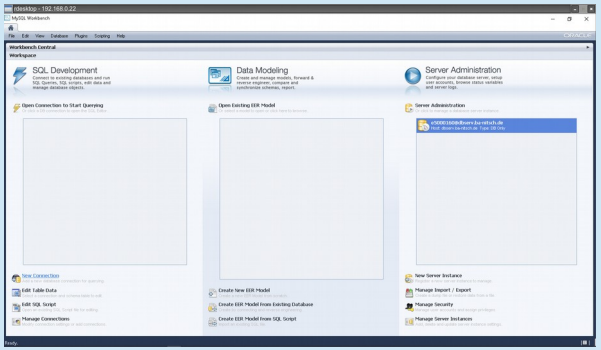
Nutzen von fertigen API

Antwort

Datenbank-Server

Datenbank – API
Bisher

Datenbank-Client



Nutzen von fertigen API

Aber: Jede Zieldatenbank hat eigenens API

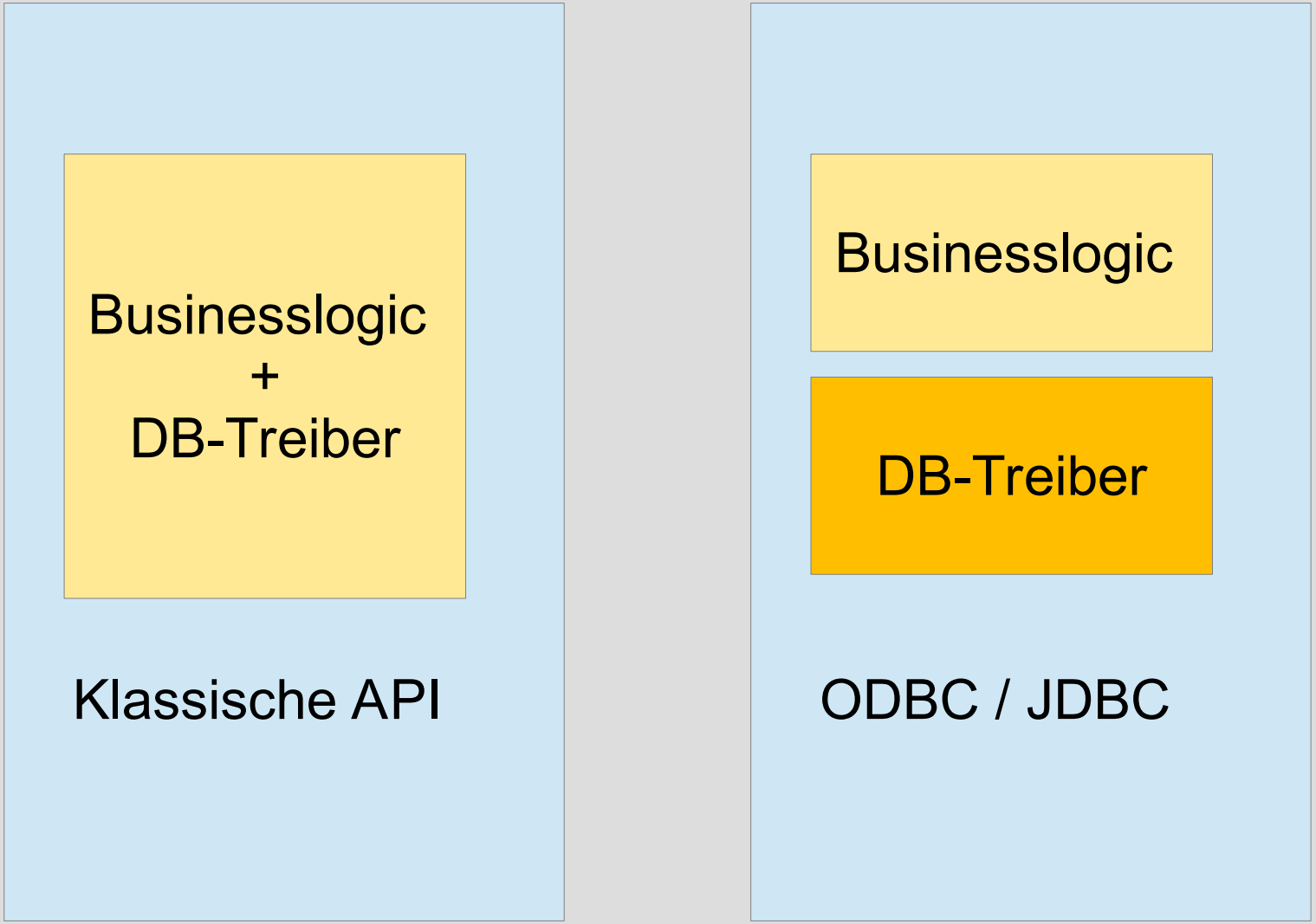
Ausweg:
Abstrahierende API

Programmiersprachen-abhängig: z.B/PDO bei PHP oder DPD bei Perl

Programmiersprachen-unabhängig: ODBC/JDBC

Datenbank-Server

Datenbank – API
Bisher

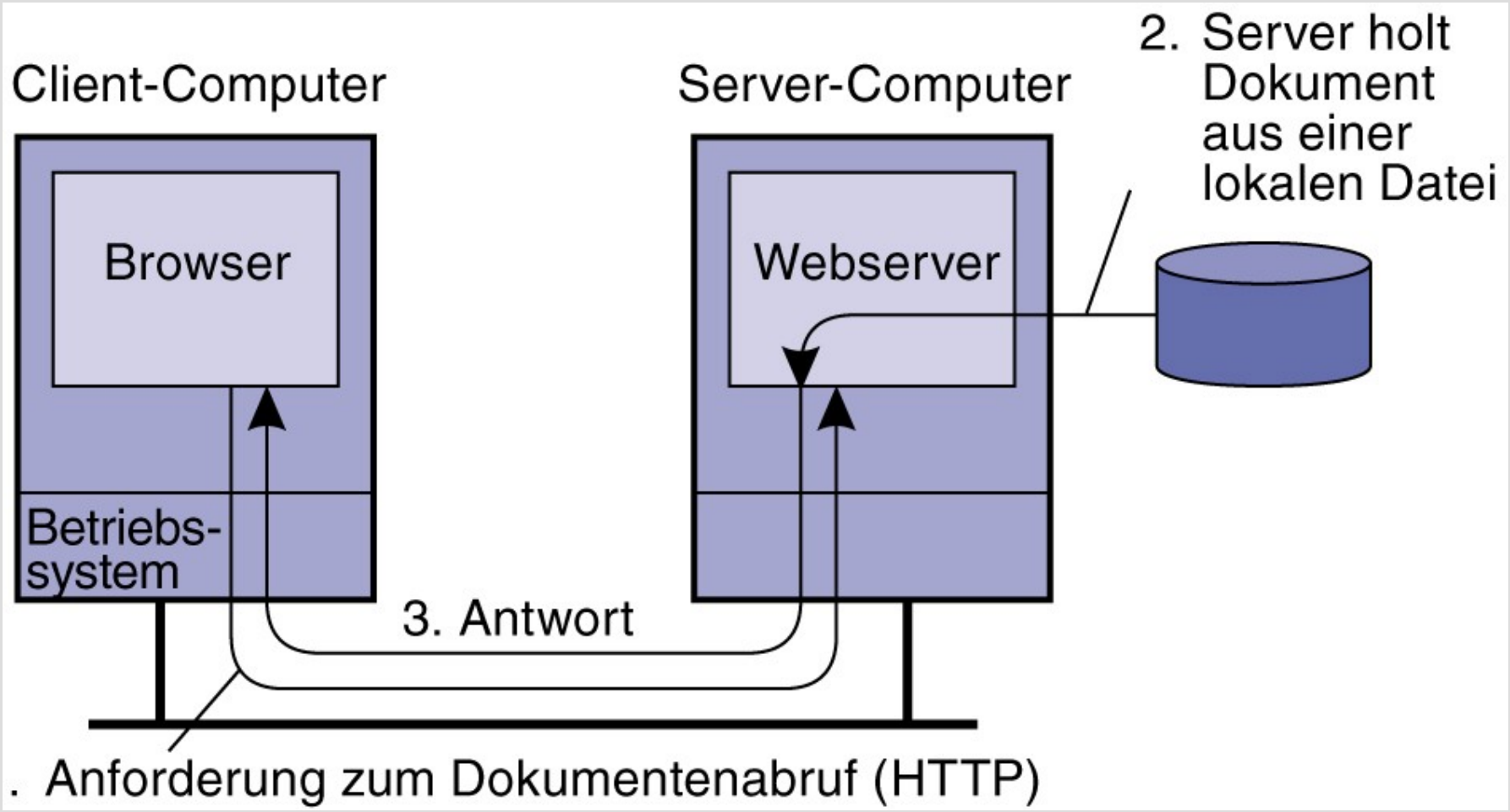


Datenbank - API

Bis hier aber immer Ausgabe auf der Konsole.

Wie auf ein browserbasiertes Websystem erweitern?

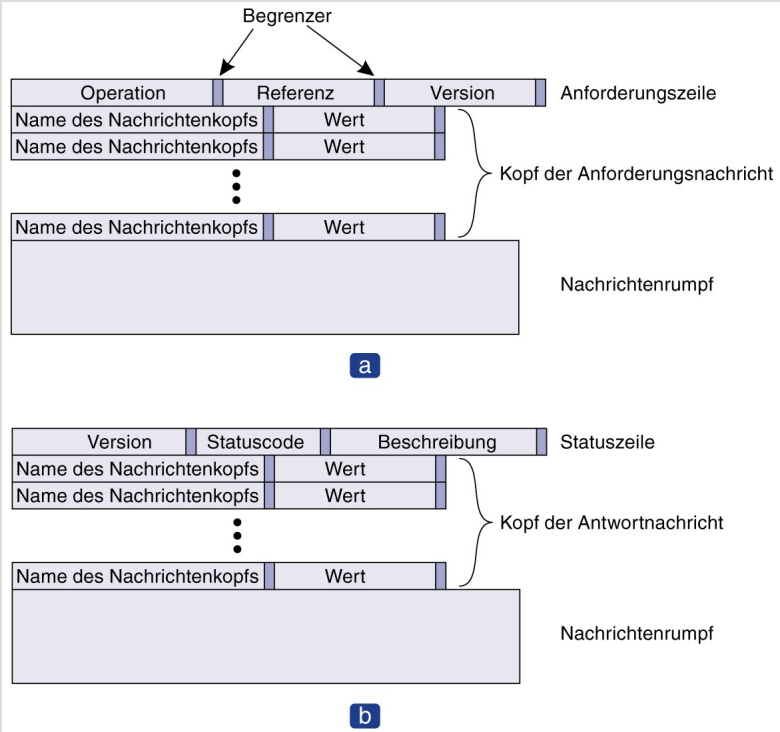
Website

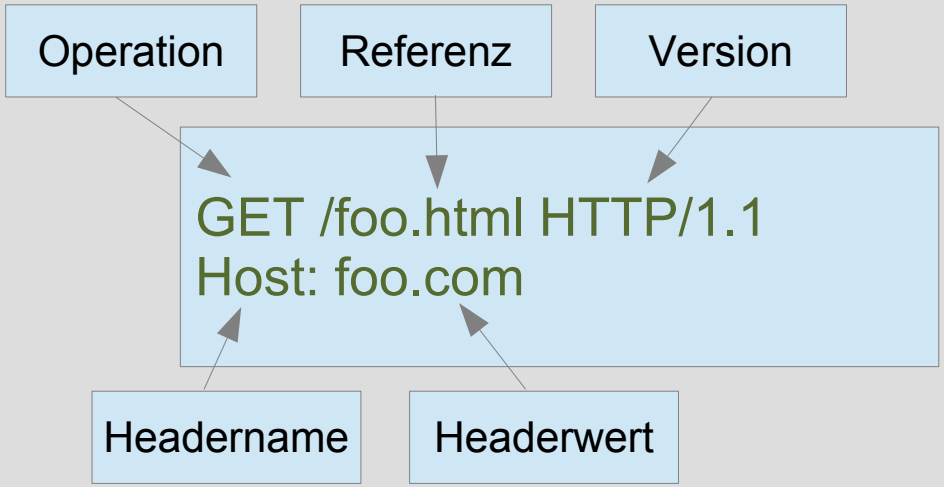


Tanenbaum, A. S.; Steen, M.v.: Verteilte Systeme. Pearson Studium, München, 2003

Bsp: GET /foo.html HTTP/1.0

HTTP-Protokoll
Request vs Response

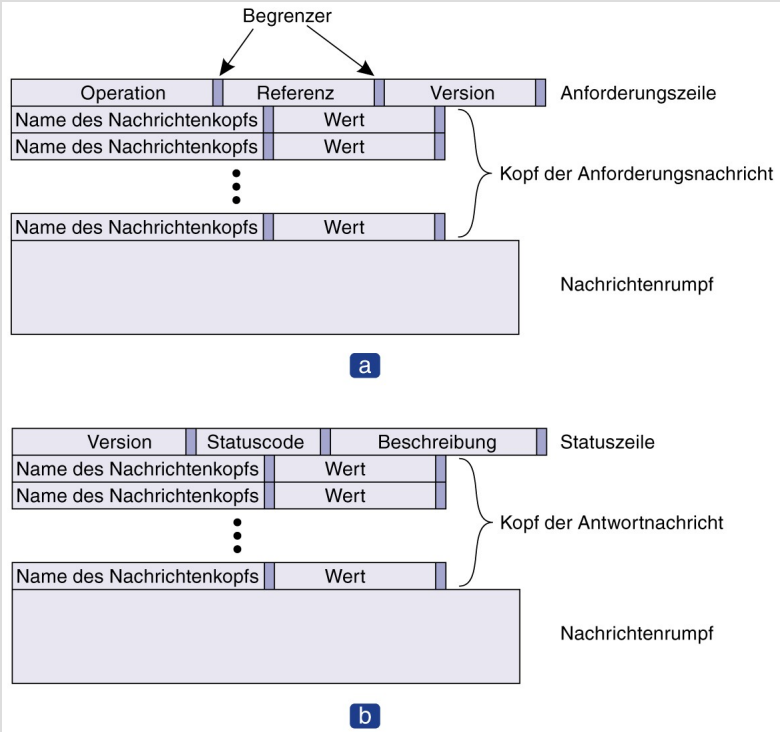


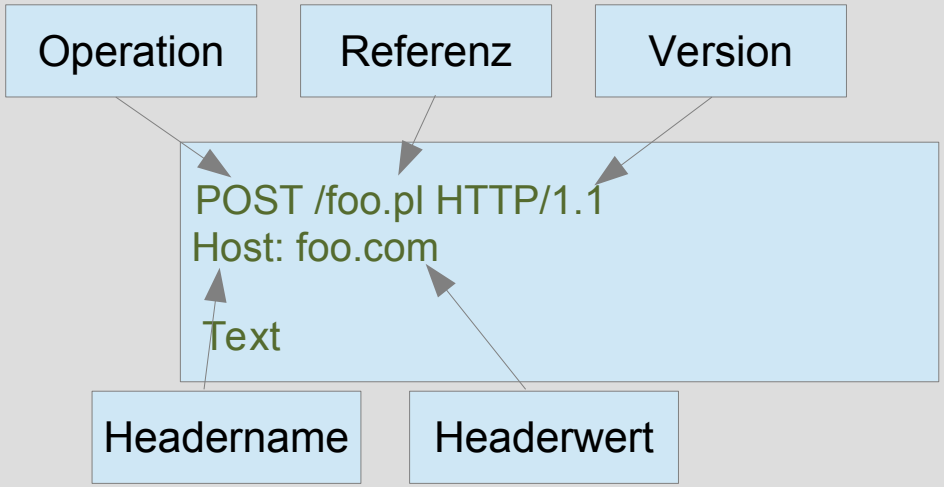



```
HTTP/1.1 200 OK
Server: Apache/2.4.10
Content-Length: 429
Content-Type: text/html

<!doctype html>
<html>..</html>
```

Tanenbaum, A. S.; Steen, M.v.: Verteilte Systeme. Pearson Studium, München, 2003

HTTP-Protokoll
Request vs Response





Tanenbaum, A. S.; Steen, M.v.: Verteilte Systeme. Pearson Studium, München, 2003

HTTP

Das war der „Trivialfall“:

→ Statische Dokumente abrufen
→ Hochladen ginge auch mittels PUT

Interessanter: Dynamisches HTTP ergo Einbinden von Programmen, und damit unseren Skripten mit Datenbank-API.

HTTP wird zum Transportprotokoll jenseits statischer Webseiten.

Datenbank – API
Weg hin zum Websystem

Browser

HTTP

Webserver

CGI

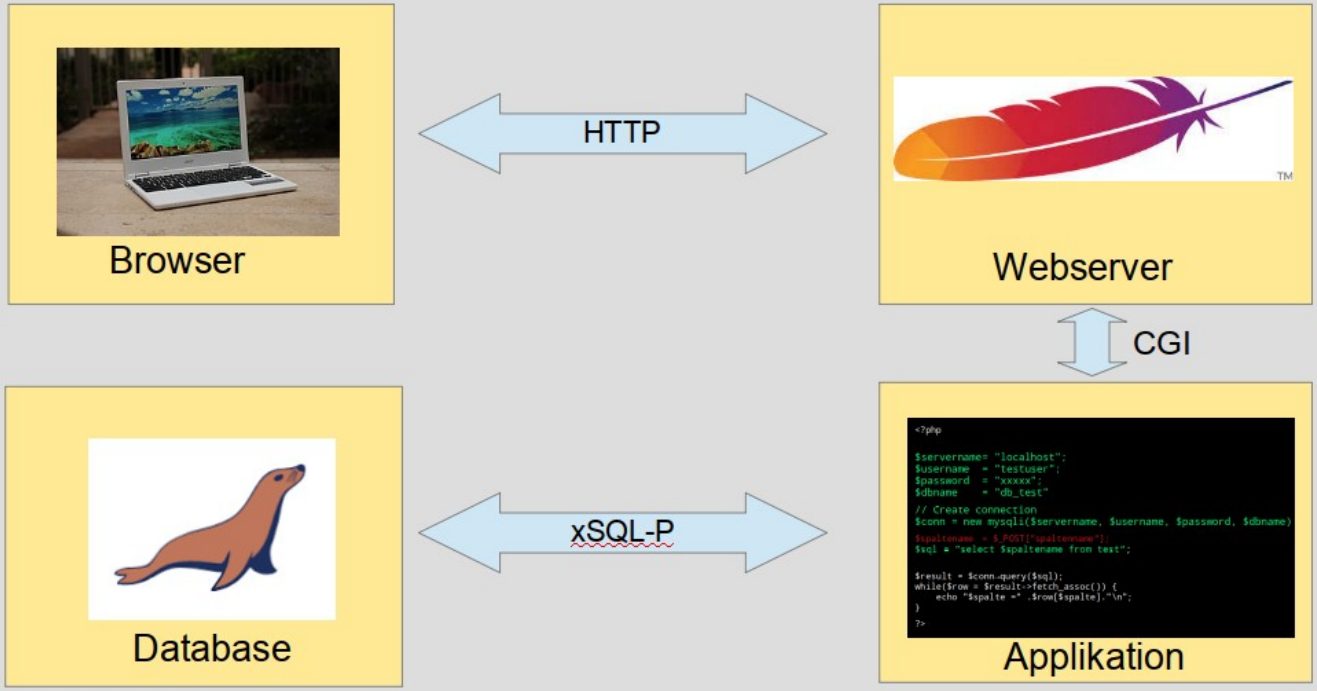
Database

xSQL-P

```php
<?php

$servername= "localhost";
$username  = "testuser";
$password  = "xxxxx";
$dbname    = "db_test"

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname)

$spaltename  = $_POST["spaltennname"];
$sql = "select $spaltename from test";


$result = $conn->query($sql);
while($row = $result->fetch_assoc()) {
    echo "$spalte =" .$row[$spalte]."\n";
}
?>
```

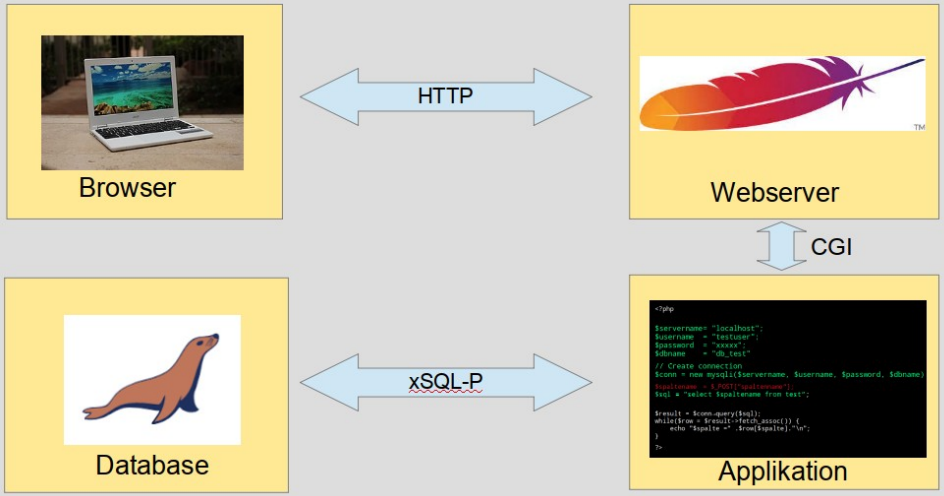
Applikation

CGI-Programme



GET /cgi-bin/foo.sh HTTP/1.1

CGI-Programme



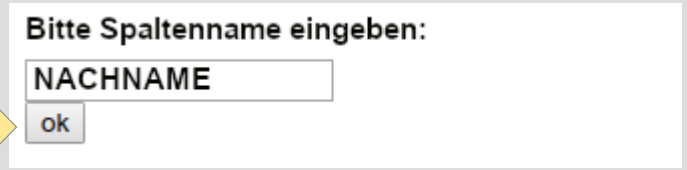
GET /cgi-bin/foo.sh HTTP/1.1

foo.sh:

```
#!/bin/sh
echo content-type: text/html
echo
echo FOO
```

Stdout wird umgeleitet
und im Browser angezeigt.

CGI
Argumente übertragen: Basics



Ansicht

```
<form method="post" action="/cgi-bin/foo.php">
<input type="text" name="spaltenname" value=""
<input type="submit" value="ok
</form>
```

HTML

```
POST / HTTP/1.1
Host: xxx.tld
Content-Length: 9
Content-Type: application/x-www-urlencoded

spaltenname=NACHNAME
```

HTTP – Post

(Aufruf durch Browser nach Click auf OK)

Datenbank - API:
CGI - Parameter mit PHP erfassen

```php
<?php

$servername= "localhost";
$username  = "testuser";
$password  = "xxxxx";
$dbname    = "db_test"

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

$spaltename  = $_POST[“spaltenname“];
$sql = "select $spaltename from test“;


$result = $conn→query($sql);
while($row = $result->fetch_assoc()) {
    echo "$spalte =“ .$row[$spalte]."\n";
}

?>
```

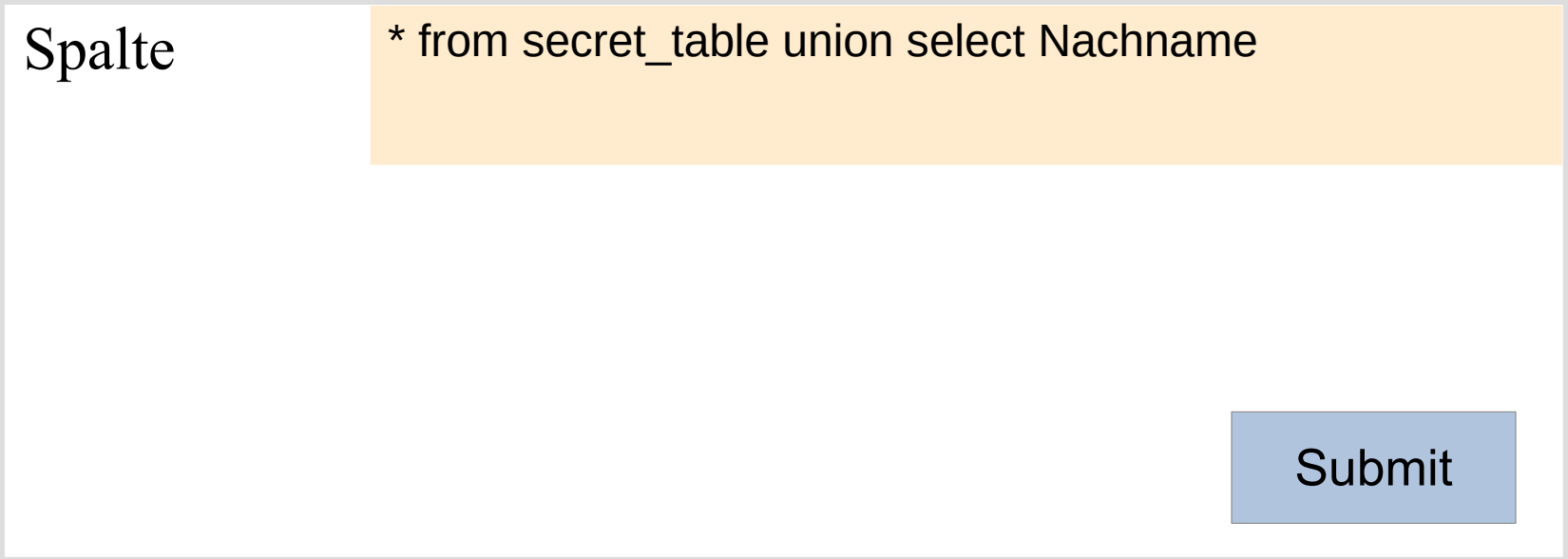
Datenbank - API

Das funktioniert im Prinzip,

Aber…

Gefahr von SQL-Injections

Datenbank - API



Das gleiche Webformular nochmal. Aber diesmal gibt der User statt eines Spaltennamens das hier ins Formularfeld ein:

* from secret_table union select Nachname

Was soll das?

Datenbank - API:PHP
Die Abfrage

```
<?php



...

$sql = "select $spaltenname from test“;
...
?>
```

Zumindest die Wirkung sieht man hier. Nun wird dieser SQL-String gebaut:

select * from secret_table union select Nachname from test

Und der sorgt dafür, das viel mehr angezeigt wird als angedacht.

Datenbank - API

Gefahr von SQL-Injections:

Nutzereingaben immer prüfen