The need for speed

Marcus Börger

The need for speed

☑ General aspects

- ➢ Communication
- ➢ Hardware
- ➢ Operating system

☑ How to use PHP

- ➢ As a web scripting language
- ➢ As a template system
- ➢ As a RAD tool
- ➢ The Rasmus way

☑ What to do and what not to do with PHP

Optimization?

Ef|fekt [lat.] der; -[e]s, -e: a) Wirkung, Erfolg; b) (meist Plural) auf Wirkung abzielendes Ausdrucks- u. Gestaltungsmittel; c) Ergebnis, sich aus etwas ergebender nutzen. […] **ef|fek|tiv** [lat.]: a) tatsächlich, wriklich; b) wirkungsvoll (im Verhältnis zu den aufgewendeten Mitteln); c) (ugs.) überhaupt, ganz u. gar, z.B. – nichts leisten; d) lohnend. […] **Ef|fek|ti|vität** die; Wirksamkeit, Durchschlagskraft, Leistungsfähigkeit, Wirkungskraft

Optimization?

ef|fi|zi|ent; -este [lat.] wirksam; wirtschaftlich; **Ef|fi|zi|enz**, die; -, -en Wirksamkeit

Effizienz (engl. efficiency): Ein Algorithmus heißt effizient, wenn er ein vorgegebenes Problem in möglichst kurzer Zeit und/oder mit möglichst geringem Aufwand an Betriebsmitteln löst. In der Praxis interessiert man sich meist für die benötigte Laufzeit (bzw. Für die Anzahl der auszuführenden Operationen), für die Größe des Speichers oder für die Zahl der Zugriffe auf Hintergrundspeicher. Die Komplexitätstheorie untersucht die Ordnung dieser Funktionen in Abhängigkeit von der Länge der Eingabe.

General aspects

☑ Do not loose your focus

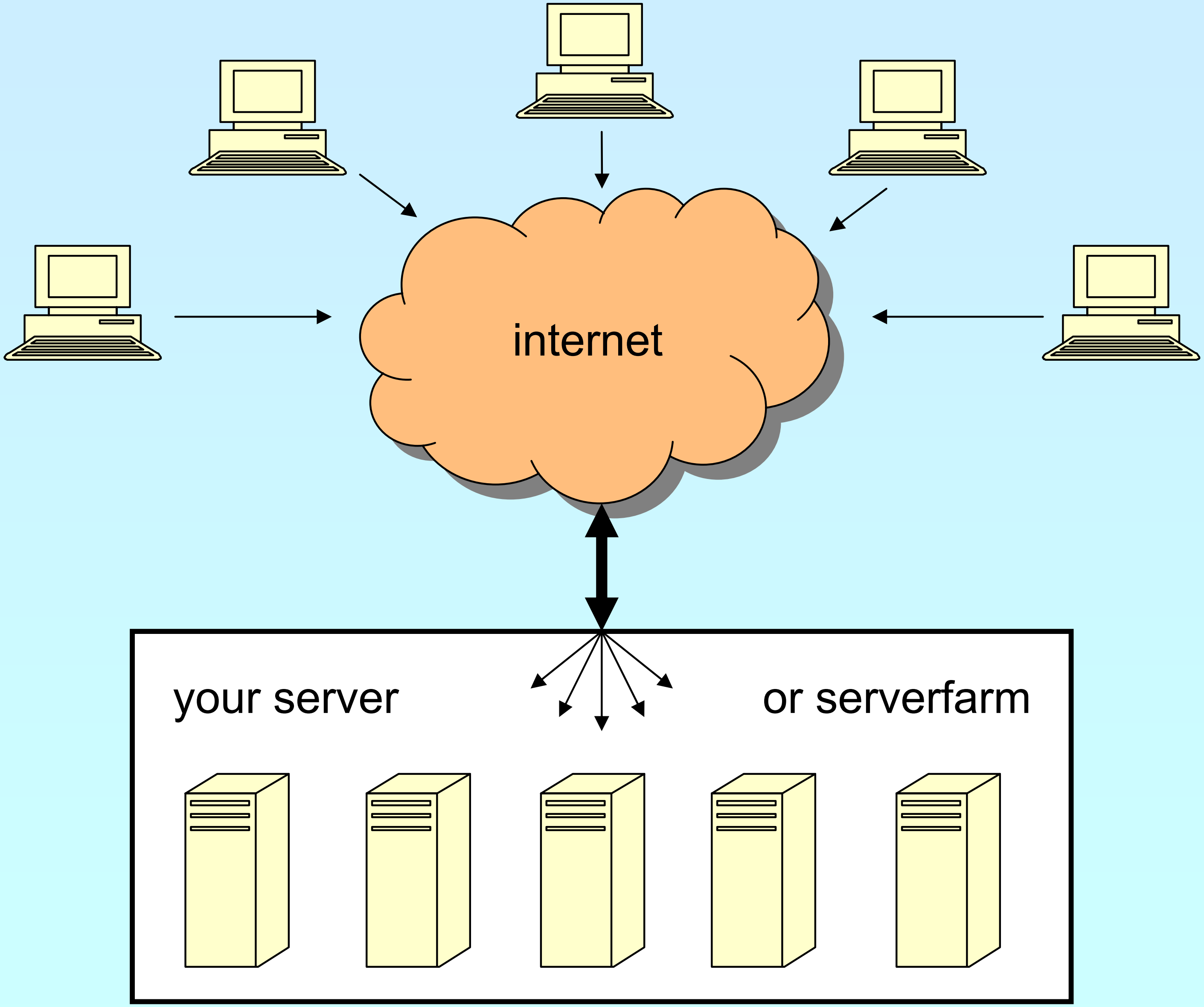
- Think before you do anything
- Always check you are still on track
- Estimate the time and money you (still) have
- Estimate the time and money you (still) need

- Are you using the right tools?
  - Is PHP the correct choice?
  - After all is a web application the right thing?

- Are you using the right algorithms?
  - Is there a better way?

- Know your environment
- Know your team

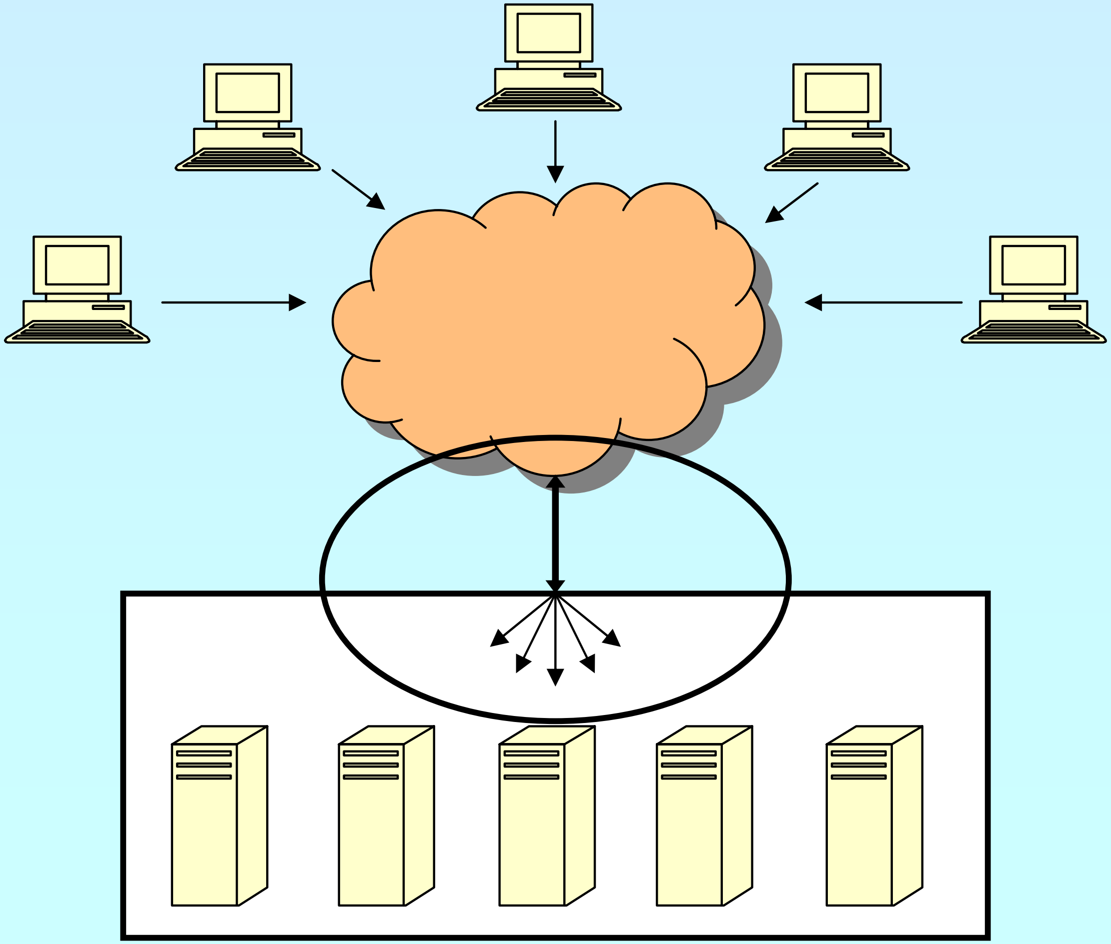
Communication



Communication

☑ The sum is smaller than the whole

- ➢ No need to apply more servers if no more bandwidth is available



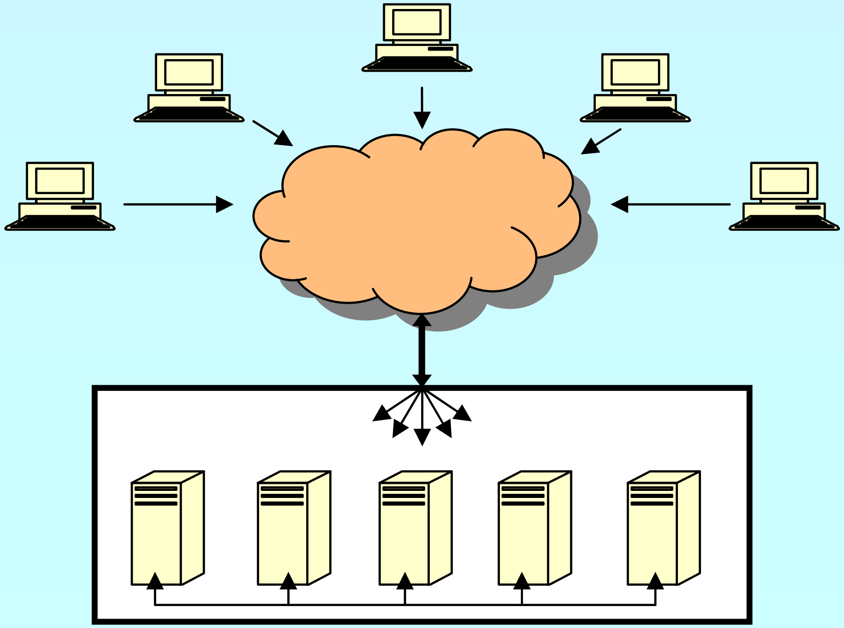
Communication

☑ The sum is smaller than the whole

- ➢ No need to apply more servers if no more bandwith is available

☑ A prepared DDoS can put down anything

☑ Applying more servers means they communicate



Communication

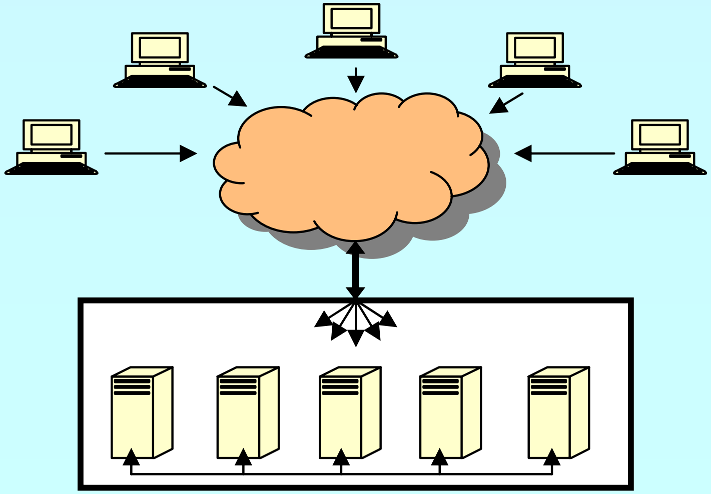
☑ The sum is smaller than the whole

- ➢ No need to apply more servers if no more bandwith is available

☑ A prepared DDoS can put down anything

☑ Applying more servers might help

- ➢ They will communicate
- ➢ You need more software
- ➢ You have more points of failure



Communication

☑ The sum is smaller than the whole

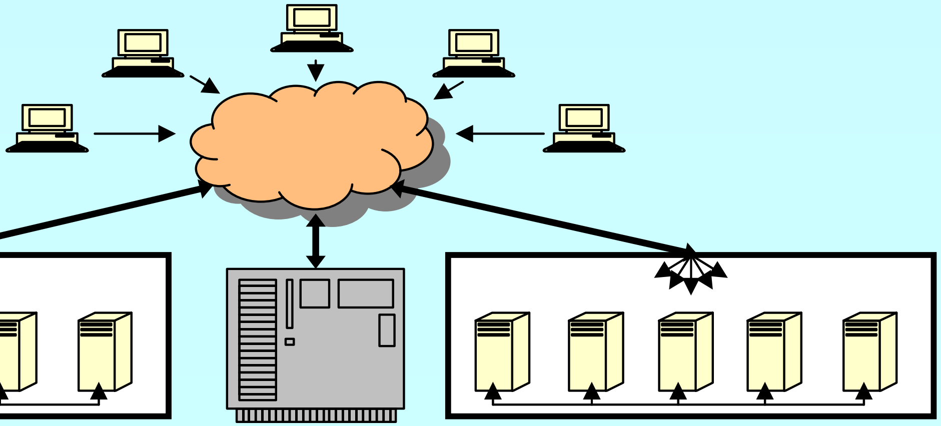
- ➢ No need to apply more servers if no more bandwith is available

☑ A prepared DDoS can put down anything

☑ Applying more servers might help

- ➢ They will communicate
- ➢ You need more software
- ➢ You have more points of failure

☑ New ideas can help



Hardware

☑ Every single hardware piece is a point of failure

- ☑ Avoid single point of failures
- ☑ Use the hardware as specified (speed, temperature)
- ☑ Don’t use it to emulate other hardware
- ☑ Don't use it to imitate other hardware

- ☑ If you don't have enough knowledge give it away

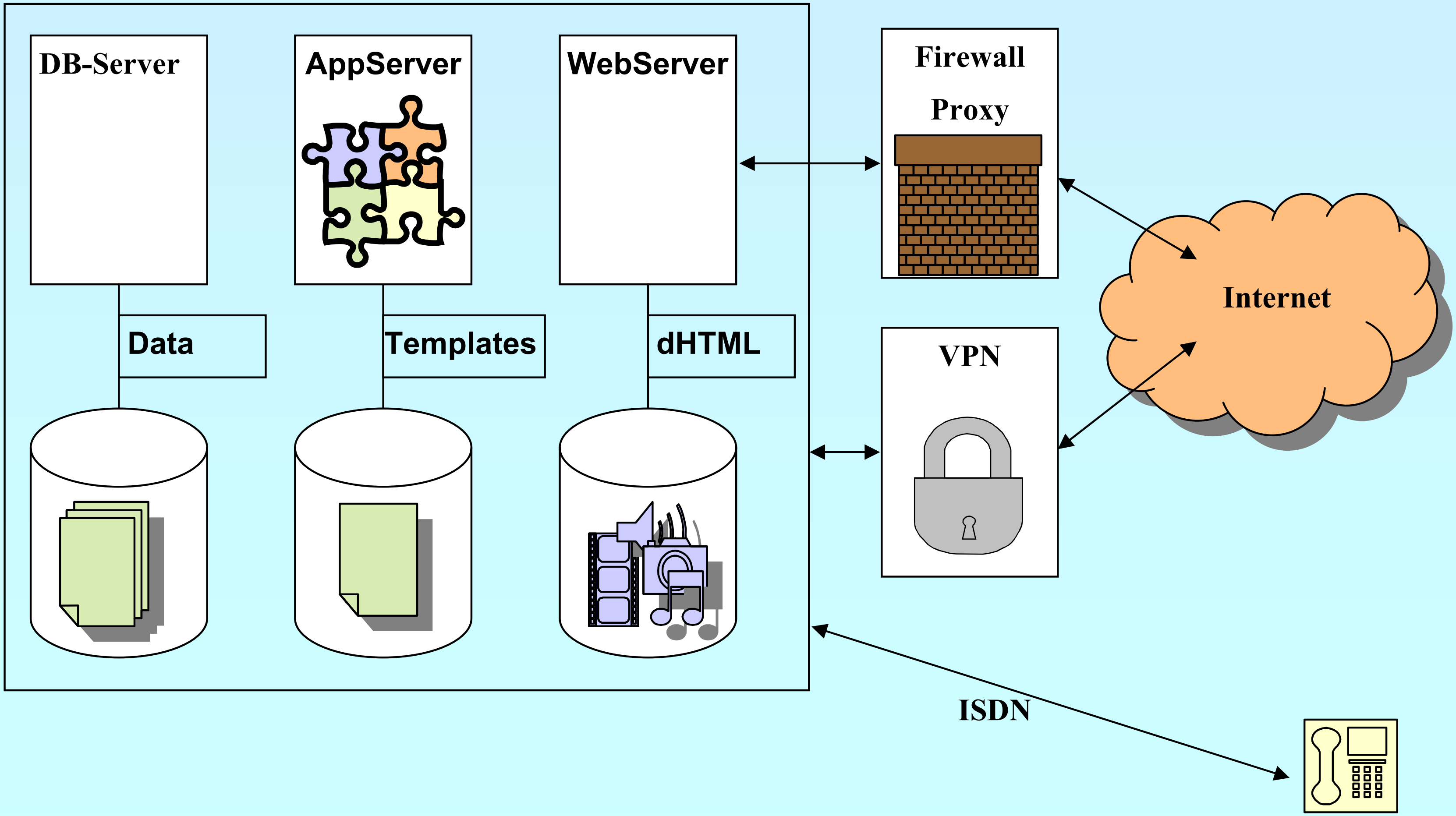
Operating system

☑ Choose the OS based on

- ☑ your hardware
- ☑ your software
- ☑ what you are going to do

Architecture

☑ Apply specialization



Database Server

☑ What kind of data

☑ What size does your data have

☑ Who is responsible for data integrity

☑ Who is responsible for security

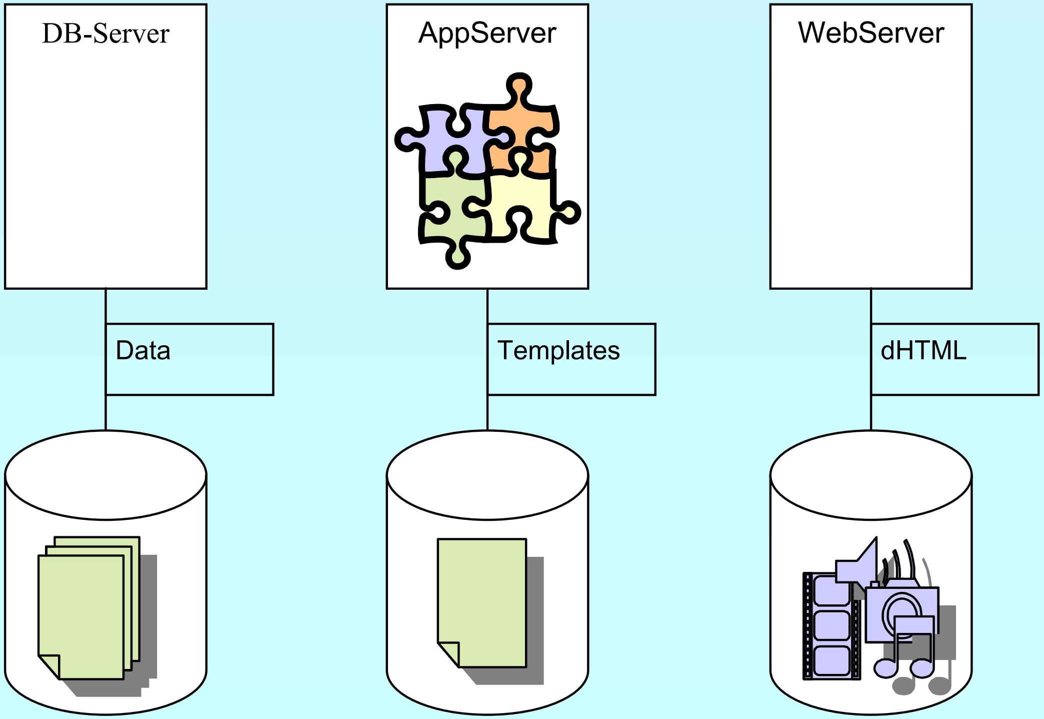
☑ Does the database need its own logic

Application Server

You want dependency injection?

You need inversion of control?

☞ PHP would need state first



Web server

☑ Apache

- ☑ Suitable for nearly all needs

☑ Microsoft IIS

- ☑ Perfect when the rest is also Microsoft
- ☑ Threadsafty issues
- ☑ Not the major/focused development platform

☑ Zeus

- ☑ Very fast

Web server

☑ TUX - kernel-based web server

- ☑ Virtual Host support.

☑ thttpd - tiny/turbo/throttling HTTP server

- ☑ Non-blocking I/O is good.
- ☑ Throttling capabilities.

☑ lighttpd

- ☑ On the fly compression.
- ☑ Excellent virtual host support.

Web server

Plenty of CPU power but limited bandwidth

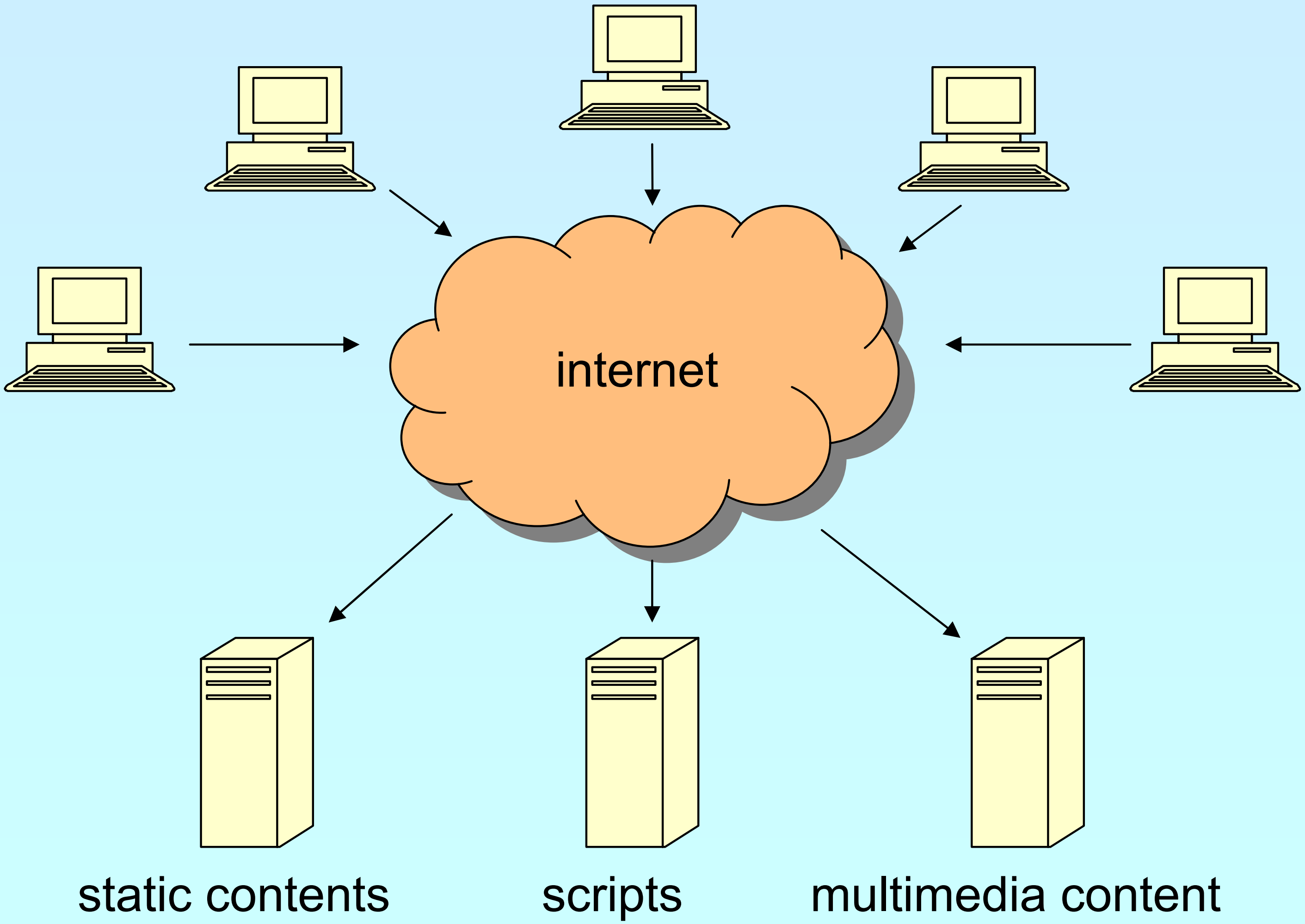
- ☞ Turn on output compression

Much bandwidth but limited CPU power

- ☞ Do not use output compression

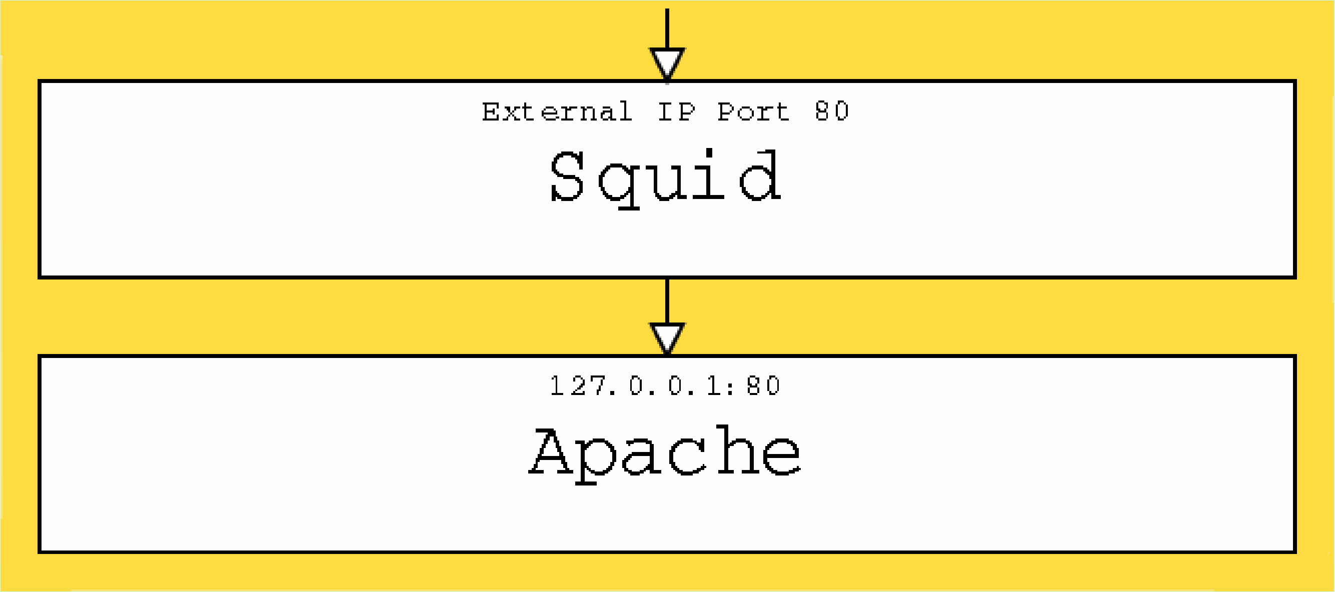
Web Server

☑ Use different web servers for different things



Reverse Proxy

☑ Cache static portions of your output



Other tricks

- ☑ Use a RAM disk where appropriate
- ☑ Use short paths and a flat layout

After all, Apache is slow?

- ☑ Compile your own apache
  - ☑ Build with static modules
  - ☑ Use –disable-all
  - ☑ Enable all compiler optimizations with -O3
  - ☑ Tell the compiler what CPU you use via -march -mcpu
  - ☑ Use CPU specific features -msse -mmmx -mfpmath=sse

After all, CGI is slow?

- ☑ Compile your own CGI
  - ☑ Build with static modules
  - ☑ Use –disable-all
  - ☑ Enable all compiler optimizations with -O3
  - ☑ Tell the compiler what CPU you use via -march -mcpu
  - ☑ Use CPU specific features -msse -mmmx -mfpmath=sse

  - ☑ Use strip to clean up your binaries
    - ☑ Saves loading time
    - ☑ Saves memory usage

Security

☑ Today security is the most important thing

☑ Many script kiddies will penetrate your application
☑ Without deep knowledge you cannot detect attacks
☑ Detecting attacks leads to protection
☑ Protection prevents misuse of your hard- and software
☑ Protection keeps your data safe

☠ Unsafe data or open systems lead directly to court

What is PHP

PHP is a scripting language specifically designed to help developers solve web problems, it works by embedding sections of code within HTML blocks.

PHP Advantages

- Easy to learn
- Targeted, built-in functions for web developers
- Good introduction to programming
- Configurable
- Simple extension API
- PEAR
- Runs britneyspears.com

PHP Disadvantages

- Focused on the Web environment
- Poor OO support until PHP 5
- Configurability Hurts Portability
- Easy for beginning users,
- Easy for beginning users to make mistakes

PHP - As web scripting language

☑ Every page is its own PHP script

- 👍 Flexible and easy
  - 👍 Independent scripts by independent programmers
- 👎 Hard to apply general tasks to all pages
  - ☞ Includes can help
  - ☞ CSS can help

PHP - As a template system

☑ PHP was developed as a template system

- ☑ PHP can be used as template system
- ☑ PHP can be the language to develop a template system

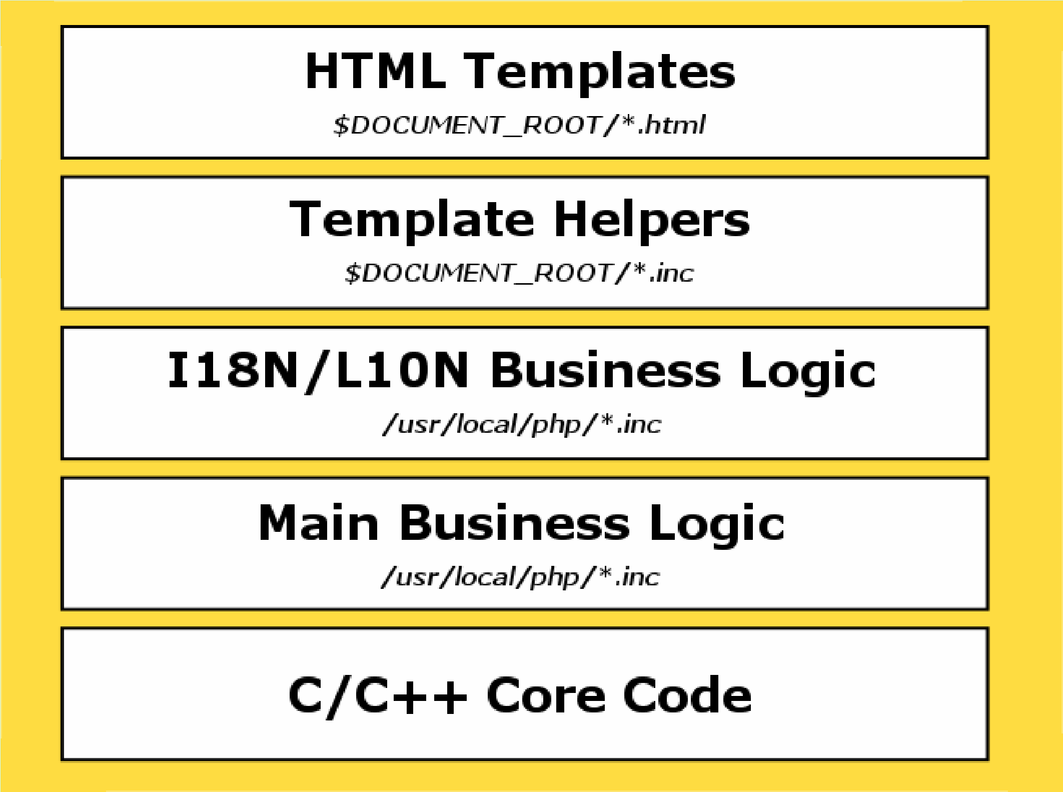
PHP - As a RAD tool

☑ No PHP in your real applications

- ☑ Test with PHP
- ☑ Implement in another language

PHP - The Rasmus way

- ☑ Small basic PHP scripts
- ☑ Small include files to solve general aspects
- ☑ Include files for the business logic
- ☑ Specialized extensions for the actual work



HTML Templates
$DOCUMENT_ROOT/*.html

Template Helpers
$DOCUMENT_ROOT/*.inc

I18N/L10N Business Logic
/usr/local/php/*.inc

Main Business Logic
/usr/local/php/*.inc

C/C++ Core Code

Optimize

☑ Everything has a cost

☑ Limit the number of includes per request

☑ Use the right tool for the right problem

☑ Use an opcode cache

☑ Use short and easy regular expressions

☑ Cache whatever you can

☑ Optimization is the root of all evil

  ☑ It steals all your time

  ☑ It makes everything complicated

  ☑ In rare cases it leads to less and easier code

Everything has a cost

☑ Do not use features you do not need

- ☑ CGI means module startup/shutdown for every page
- ☑ **`include_path`** means every possibility has to be tested
- ☑ **`open_basedir`** means every entry has to be checked
- ☑ **`variables_order`** lets you decide what you need
- ☑ **`magic_quotes_`*** means parsing/changing overhead
- ☑ **`register_argc_argv`** is only for CLI
- ☑ **`always_populate_raw_post_data`** only if necessary

Everything has a cost

☑ ext/tidy can beautify your output

you could do it before you send the data

☑ ext/tidy can strip out whitespace

reduces the bandwidth needed

takes CPU time

Everything has a cost

☑ PHP can dynamically resize images

- you could supply the resized images
- you could cache the resized images

Everything has a cost

☑ Function calls are expensive

☑ User Functions are more expensive

☑ Passing parameters takes time

☞ Learn about the PHP API

☞ Always have the manual at hand

☞ Do not write PHP functions when available by PHP

☞ Do not have long optional parameter lists

☞ Do not use functions for multiple purposes

Also: ☞ Do not write spaghetti code

☞ Document your code

Everything has a cost

☑ Copying a variable takes time

☞ Learn when PHP needs to copy

☞ Learn about references

Everything has a cost

- ☑ Close your sessions early
  - ☑ Use **`session_write_close()`**
  - ☑ An open session prevents others accessing the session

References

A famous PHP 4 rule:

If your code doesn't work spread some '&'s into it

If it still doesn't work use more '&'

☞ Understand references

References

- ☑ References are aliases
  - ☑ If you change one you change all others

```
<?php            // empty global table
$a = 25;         // creates a zval
$b = $a;         // creates a pointer to $a
$b = 42;         // makes $b a copy of $a and changes it
$c = $a;         // create another pointer to $a
$d = &$a;        // split/copy $a, creates $d as a reference to $a
$c = 43;         // change $c only
$d = 0;          // changes $d and hence $a
?>
```

References

☑ Variables are normally copied on function calls

```
<?php

function test($a)
{
}

$a = array(25);     // creates a global zval


test($a);           // creates a new symbol table, copies $a

?>
```

References

☑ Variables can be passed as references

```
<?php

function test(&$b)
{
   $b[] = 42;       // adds a new value to local $b = global $a
}

$a = array(25);  // creates a global zval


test($a);           // creates a new symbol table

?>
```

References

☑ Variables are normally copied on return

```
<?php

function test(&$b)
{
    return $b;
}

$a = array(25);

$b = test($a);      // $b is a new value, copied on return

?>
```

References

☑ Functions can return aliases

```
<?php

function &test(&$b)
{
    return $b;
}

$a = array(25);

$b = test($a);      // $b is a new value, copied after return

?>
```

References

- ☑ Functions can return aliases
- ☑ Explicit use of the returned reference is needed

```php
<?php

function &test(&$b)
{
   return $b;
}

$a = array(25);

$b = &test($a);    // $b is a reference to $a

?>
```

References

- ☑ Objects should always be references
  - ☑ In PHP 5 they are object-references

```
<?php
class test
{
    function factory() {
        return new test();
    }
}

$obj = test::factory();
?>
```

References

- ☑ Objects should always be references
  - ☑ In PHP 5 they are object-references
  - ☑ In PHP 3 and 4 you have to take care yourself

```
<?php
class test
{
    function &factory() {
        $a = &new test();
        return $a;
    }
}

$obj = &test::factory();
?>
```

References

☑ Most internal functions don't use references

- ☑ This is to allows you to pass arrays and strings without copying them into a variable first

```
<?php
$a = array_fill(0, $cnt, 'foo');
array_key_exists($i, $a);    // is_ref == 0, refcount == 1
$b = $a;
array_key_exists($i, $a);    // is_ref == 0, refcount == 2
array_key_exists($i, &$a);  // is_ref == 0, refcount == 2
unset($b);
$b =& $a;                    // making a reference, but not using it
array_key_exists($i, $b);    // is_ref == 1, refcount > 1 (pass as var)
array_key_exists($i, &$a);  // is_ref == 1, refcount > 1 (pass as ref)
unset($b);
array_key_exists($i, $a);    // is_ref == ?, refcount == 1
?>
```

Use the right tool
For the right problem

☑ Use OOP where appropriate not where nice

☑ Use layers not because it is easy or looks nice

☑ Use abstraction if derived or used often

☑ Use indirection if it is of any advantage

Profile your code

☑ Profile your code

- ☑ Do not use microtime() for performance measurements
- ☑ Use a profiler for your PHP script
  - ☑ APD
  - ☑ XDebug
  - ☑ ...
- ☑ Use a profiler for 'grown up' problems
  - ☑ Valgrind/calltree

The 80 / 20 rule

- ☑ 80% of your code takes less than 20% runtime
- ☑ You don't need to optimize anything in the 80%
- ☑ Find out which are the 20% to optimize

XDebug

- ☑ A tool to debug PHP
- ☑ Tracing function calls
- ☑ A profiler

Cache whatever you can

☑ Most dynamic data does not change

☑ At least not every time it is requested

- ☞ Use cache control header

Cache whatever you can

☑ Pre generation

- ☑ Generate your data once
- ☑ Server the generated data statically

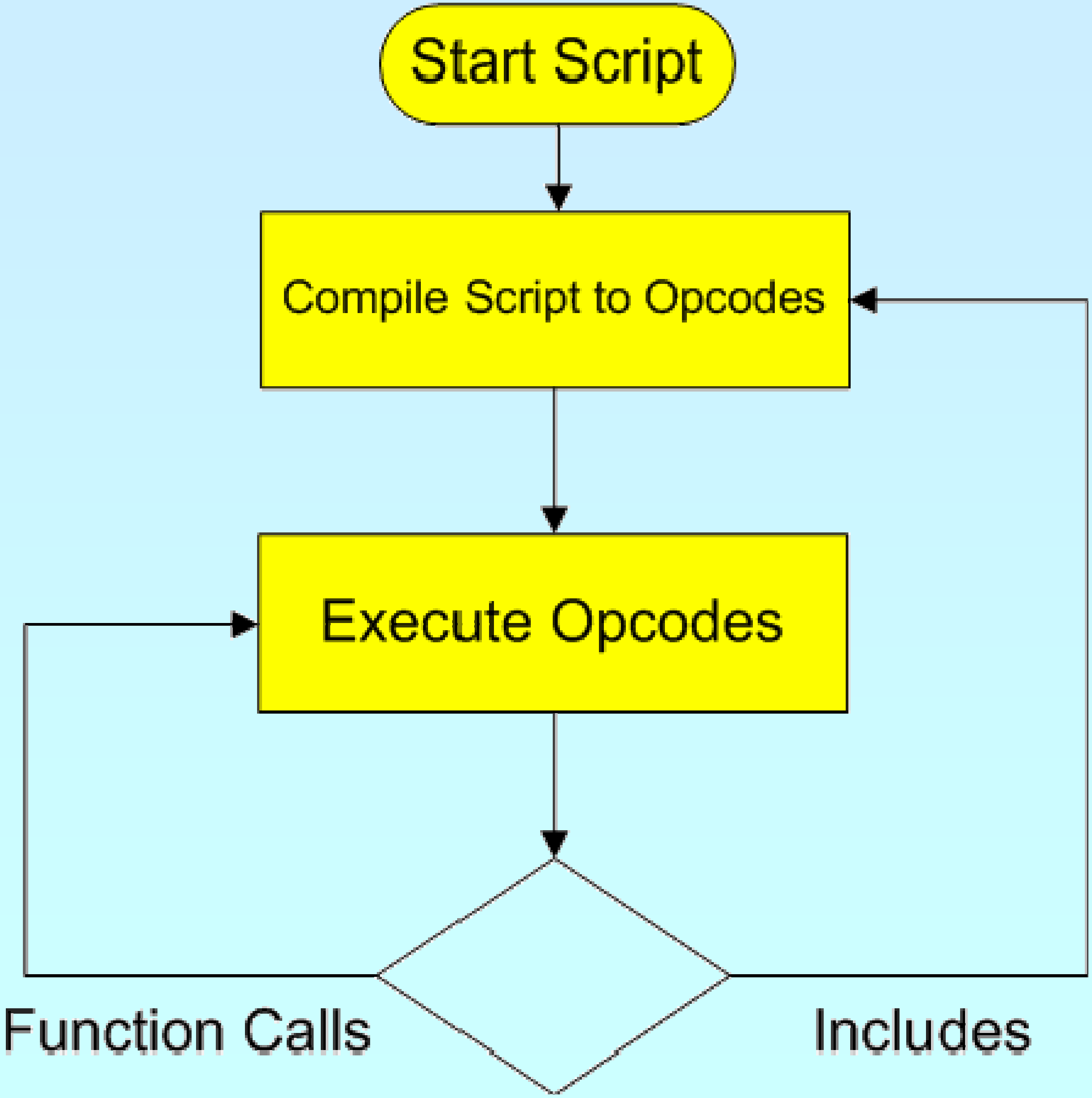
☑ On demand

- ☑ Generate when requested for the first time

☑ Dynamic caching

- ☑ Generate when necessary
- ☑ Serve generated data statically otherwise

Use an opcode cache



Start Script

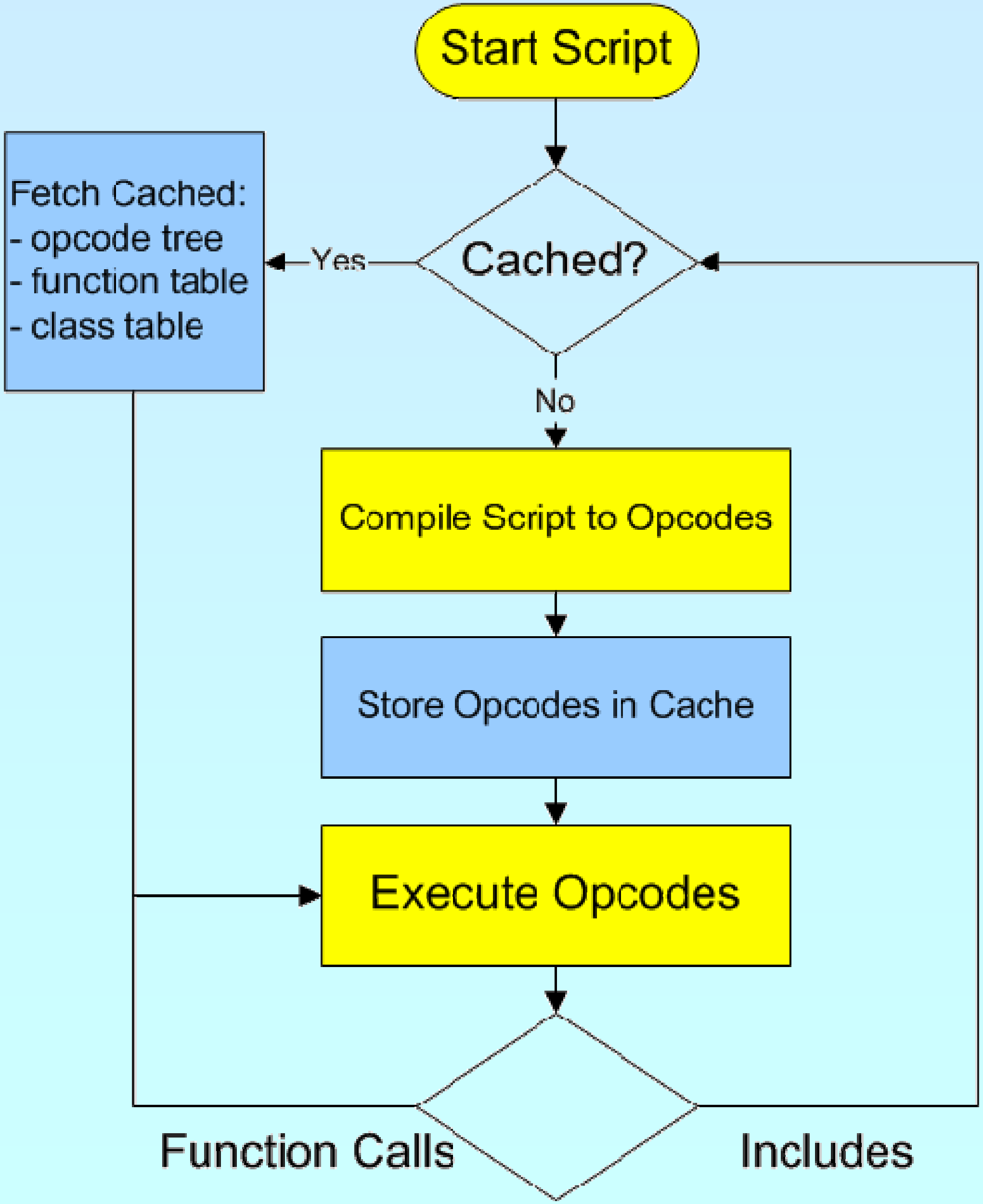
Compile Script to Opcodes

Execute Opcodes

Function Calls

Includes

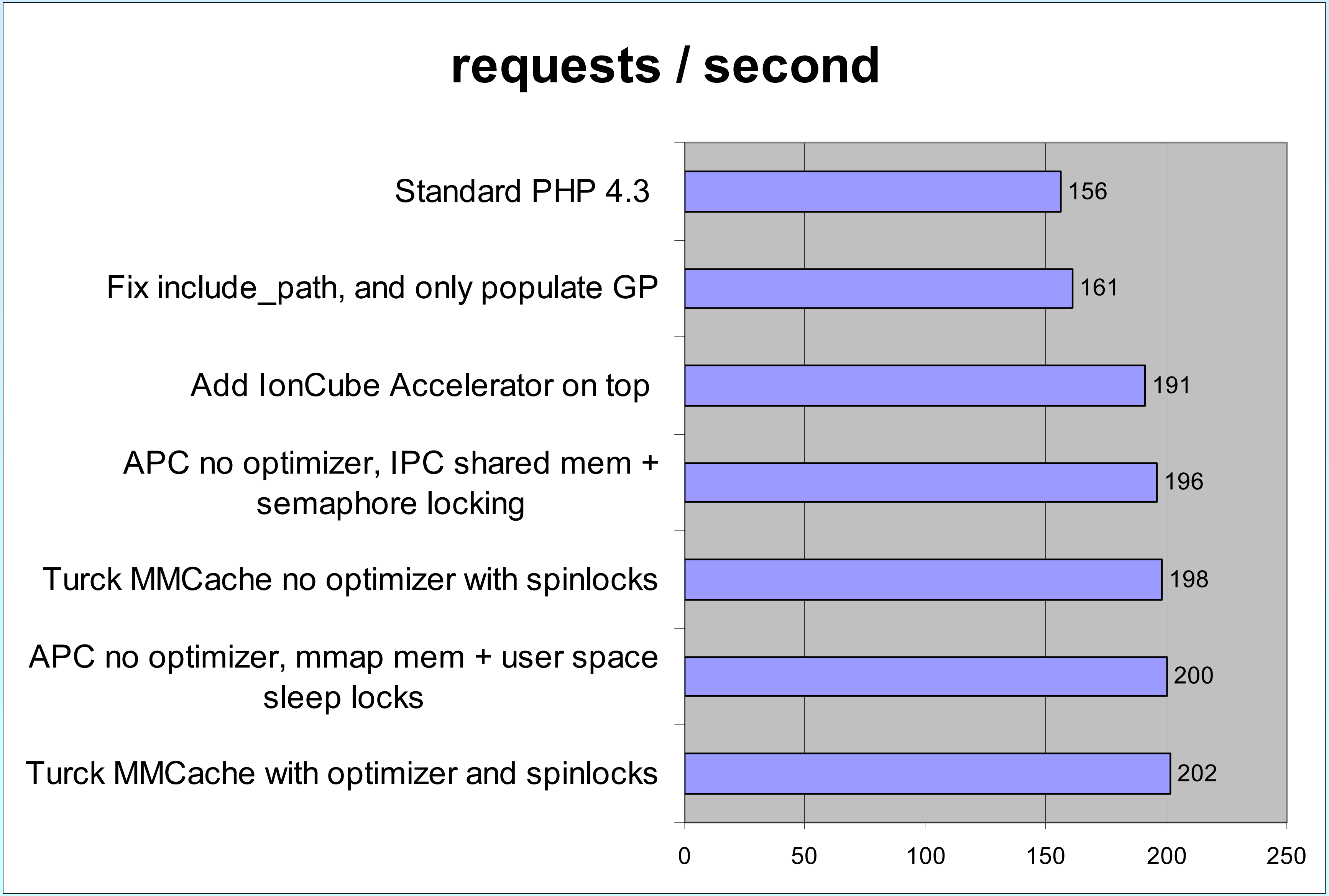
Use an opcode cache



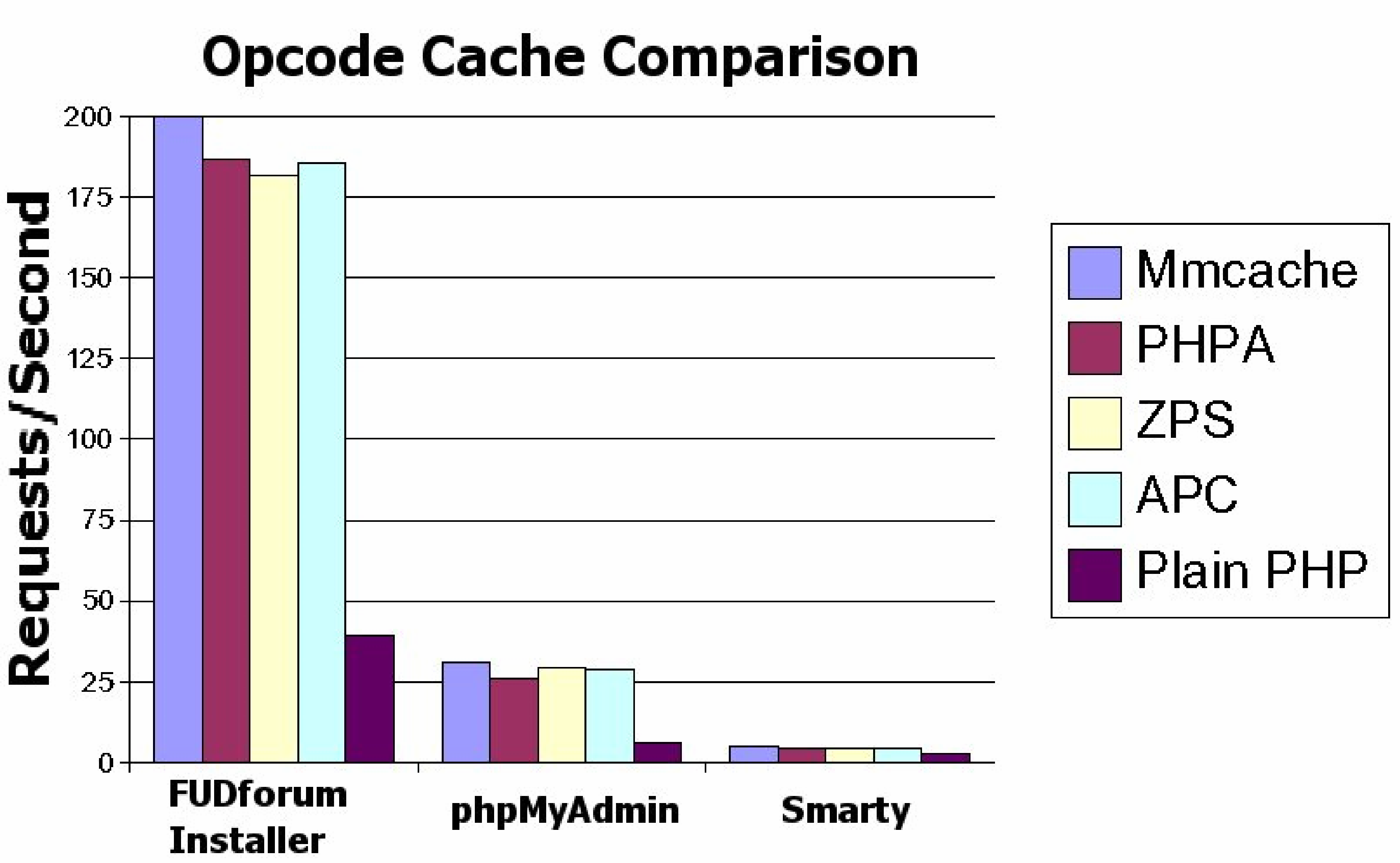
Use an opcode cache

- ☑ Turck MMCache (GPL)
  - ☑ Implements many features
  - ☑ Development halted
- ☑ APC (PHP)
  - ☑ Slow but development continues
  - ☑ Weak optimizer
- ☑ ionCube PHP Accelerator
  - ☑ It works
  - ☑ Development halted?
  - ☑ Free, but closed source
- ☑ Zend Cache (Proprietary)
  - ☑ Implements many features
  - ☑ Expensive

Use an opcode cache



Use an opcode cache



Stop

☑ Don’t get overexcited about optimization

☑ Sometimes it is cheaper and more efficient

  ☑ to buy another server

  ☑ to increase bandwidth

  ☑ To buy faster software

THANK YOU

http://somabo.de/talks/

http://talks.php.net