Optimization Tricks & Mistakes to Avoid

Ilia Alshanetsky
@iliaa

http://joind.in/4026

Premature Optimization = $\sqrt{\text{evil}}$

Premature Optimization = $\sqrt{\text{evil}}$

Solve the business case, before optimizing the solution

Don't Over Engineer

- Understand your audience
- Estimate the scale and growth of your application (based on facts, not marketing fiction)
- Keep timelines in mind when setting the project scope

Simplify, Simplify & Simplify!

- Break complex tasks into simpler sub-components
- Don’t be afraid to modularize the code
- More code does not translate to slower code (common misconception)

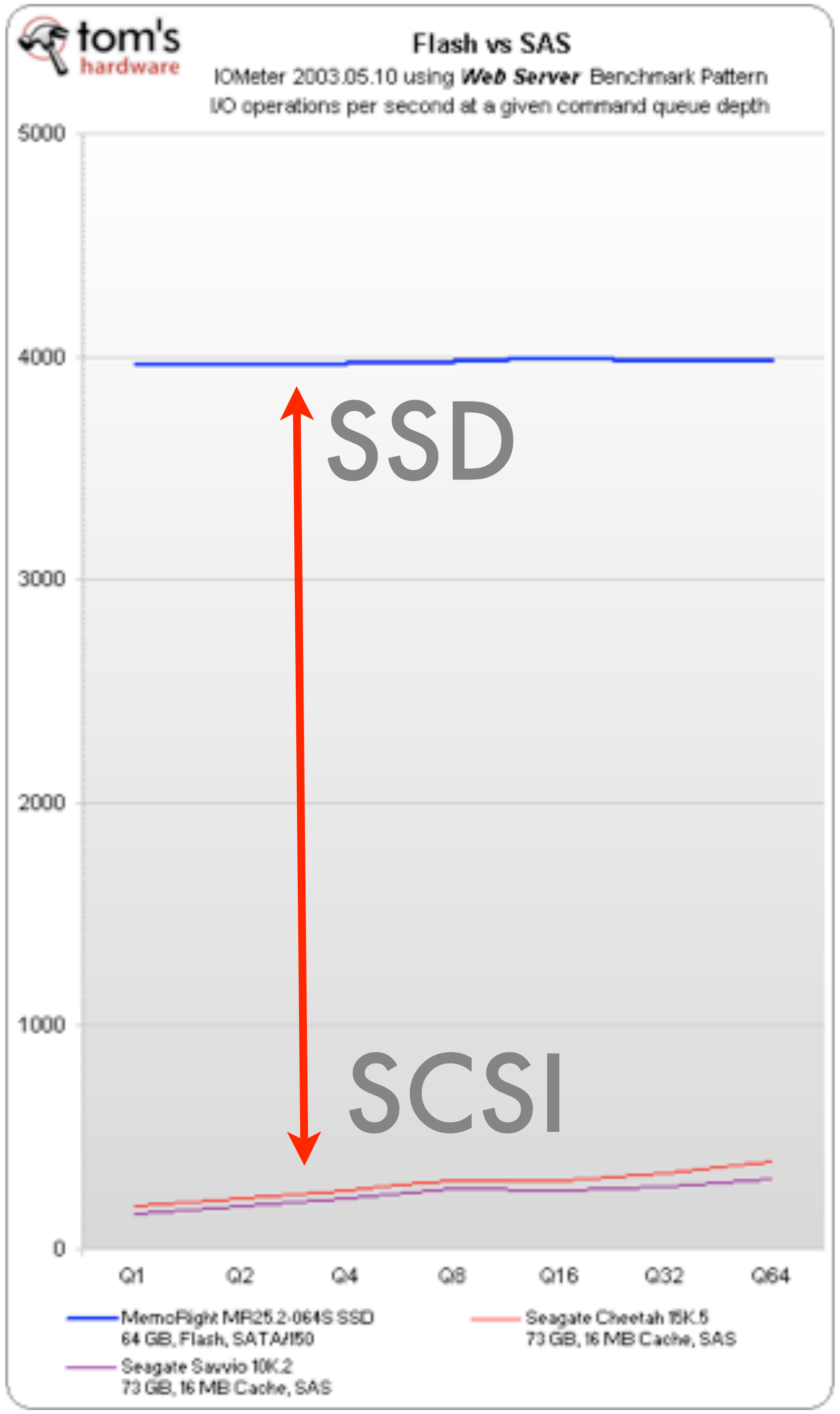
Hardware is Cheaper!

vs

In most cases applications can gain vast performance gains by improving hardware, quickly rather than through slow, error prone code optimization efforts.

Hardware

- In many application speed is limited by Disk IO. SSDs provide a rapid solution to that problem.

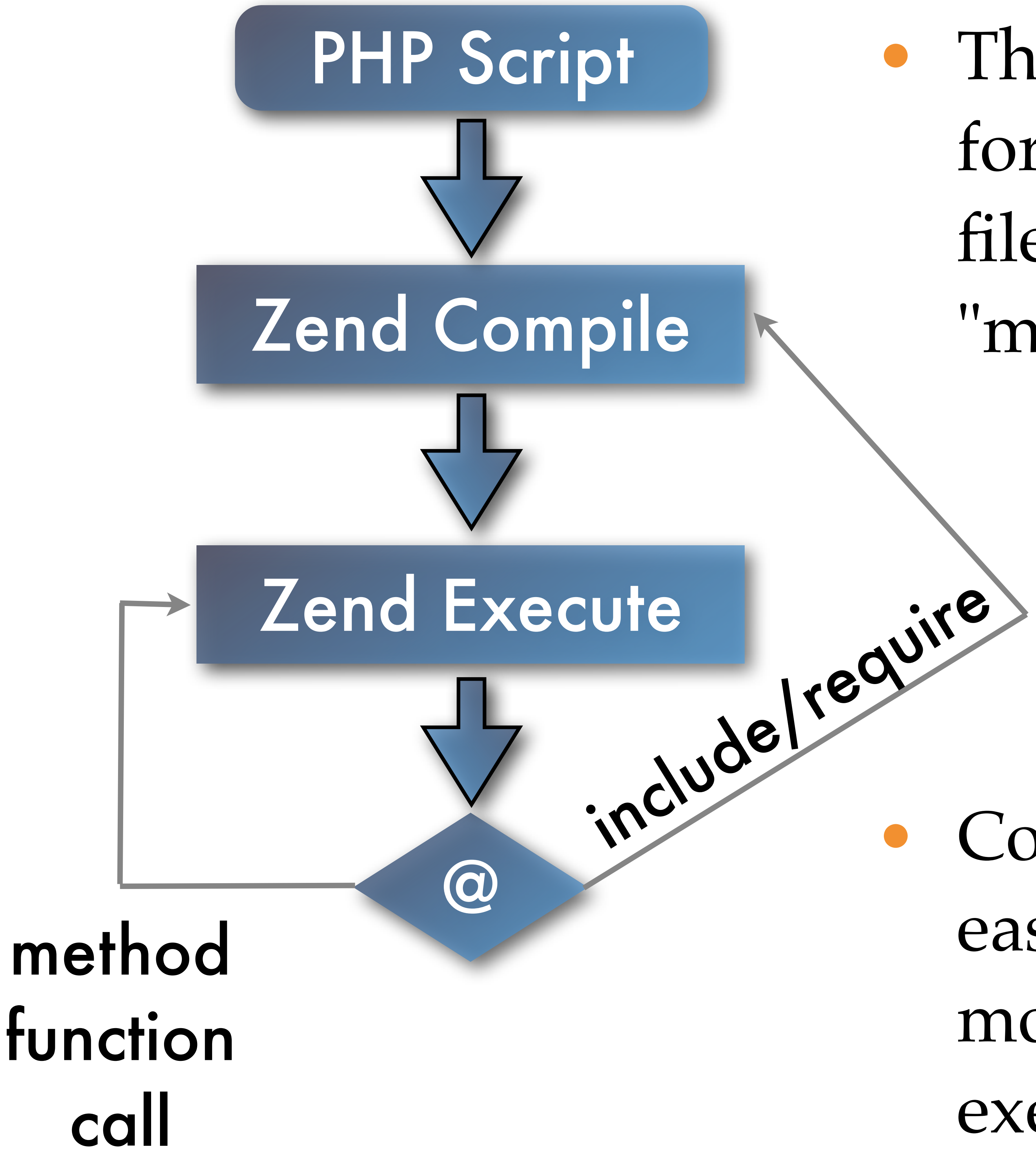


Hardware Caveat

- While quick to give results, in some situations it will not help for long:
  - Database saturation
  - Non-scalable code base
  - Network bound bottleneck
  - Extremely low number sessions per server

Optimize, but don't touch the code

How PHP works in 30 seconds



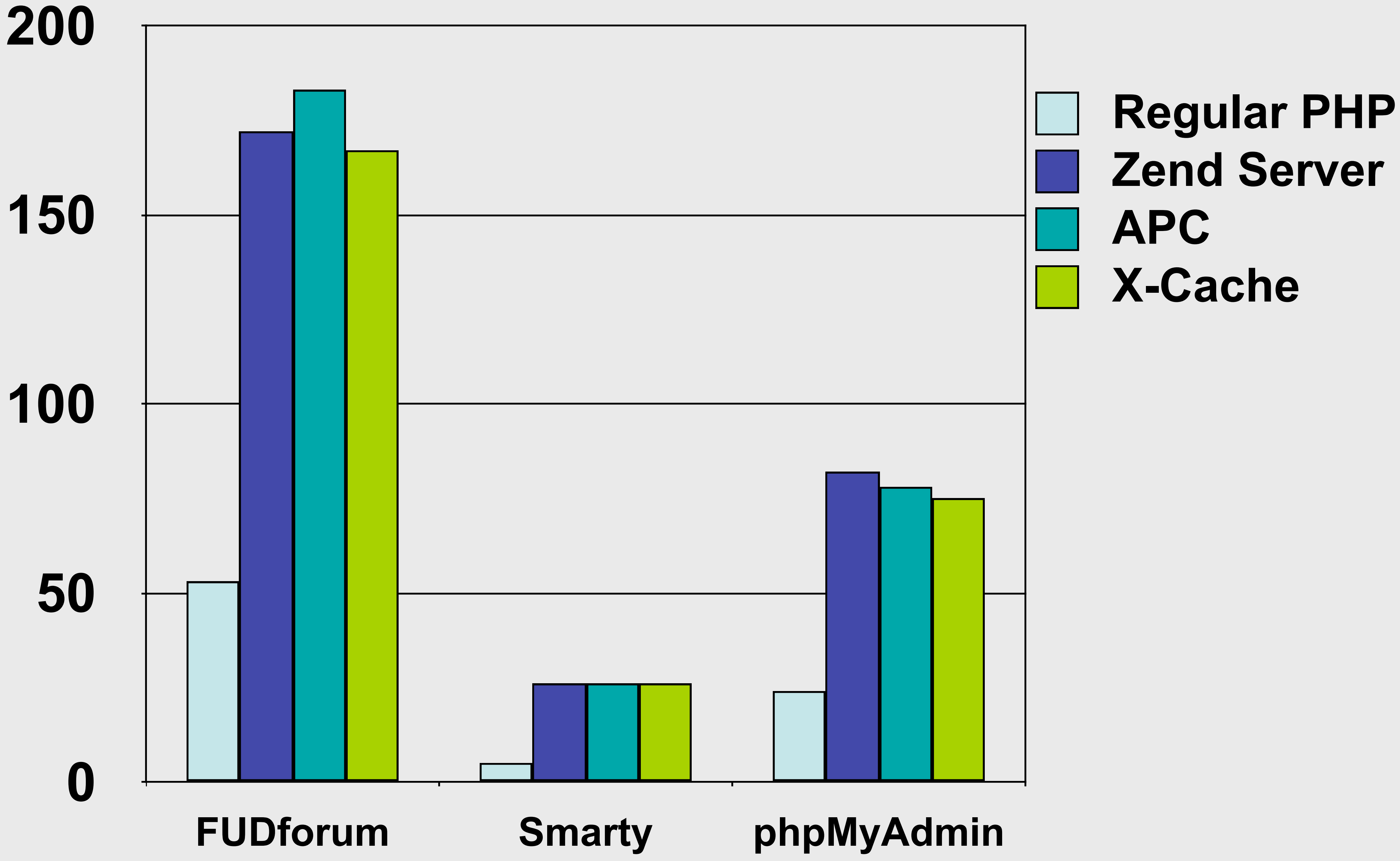
- This cycle happens for every include file, not just for the "main" script.
- Compilation can easily consume more time than execution.

Opcode Cache

- Each PHP script is interpreted only once for each revision
- Reduced File IO, opcodes are being read from memory instead of being parsed from disk
- Opcodes can optimized for faster execution
- Yields a minimum 20-30% speed improvement and often as much as 200-400%

http://pecl.php.net/apc

Quick Comparison



Use In-Memory Caches

- In-memory session storage is MUCH faster than disk or database equivalents

- Very simple via **Memcached** extension

```
session.save_path = “localhost:11211”
session.save_handler = “memcached”
```

https://github.com/php-memcached-dev

Everything has to be Real-time

Complete Page Caching

- Caching Proxy ala NginX
- Page pre-generation
- On-demand caching

Partial Cache - SQL

- In most applications the primary bottleneck can often be traced to “database work”

- Caching of SQL can drastically reduce the load caused by unavoidable, complex queries

SQL Caching Example

```
$key = md5(“some sort of sql query”);
if (!($result = cache_fetch($key))) {
  $result = $pdo->query($qry)->fetchAll();
  // cache query result for 1 hour
  cache_fetch($key, $result, NULL, 3600);
}
```

Partial Cache - Code

- Rather than optimizing complex PHP operations, it is often better to simply cache their output for a period of time
  - Faster payoff
  - Lower chance of breaking the code
  - Faster then any “code optimization”

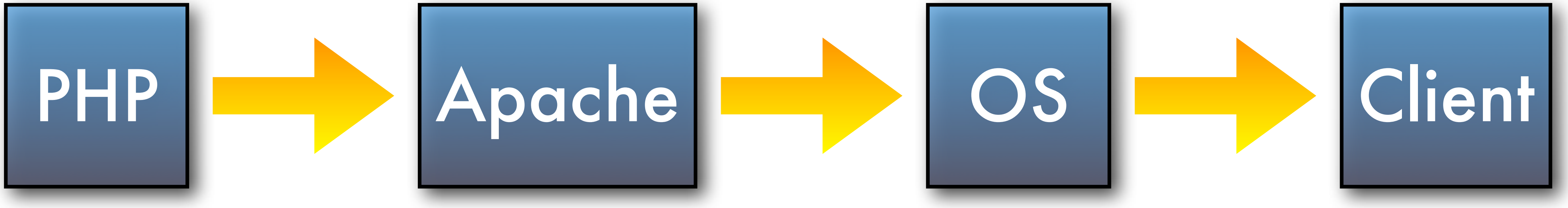
Code Caching Example

```
function myFunction($a, $b, $c) {
  $key = __FUNCTION__ . serialize(func_get_args());
  if (!($result = cache_get($key))) {
    $result = // function code
    // cache query result for 1 hour
    cache_set($key, $result, NULL, 3600);
  }
  return $result;
}
```

Output Buffering

Don’t fear output buffering because it uses RAM, RAM is cheap. IO, not so much.

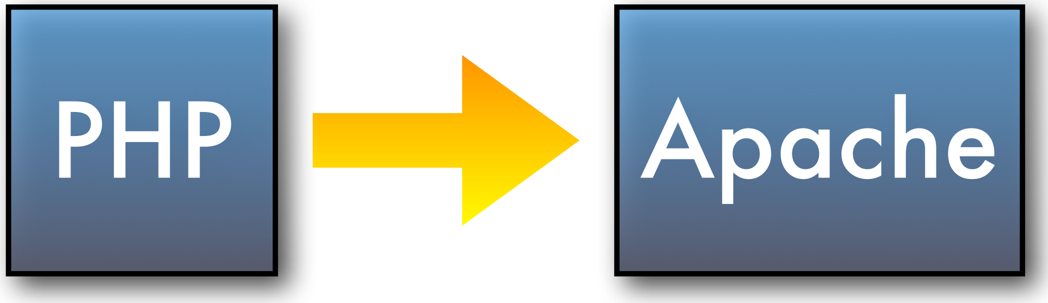
Matching Your IO Sizes



PHP → Apache → OS → Client

- The goal is to pass off as much work to the kernel as efficiently as possible.
- Optimizes PHP to OS Communication
- Reduces Number Of System Calls

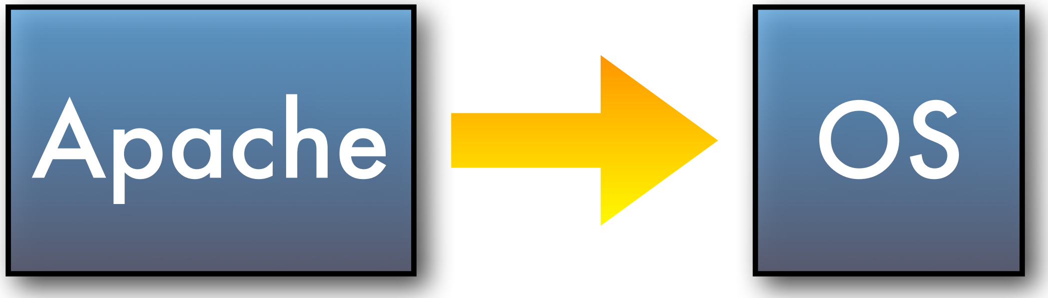
PHP: Output Control



- Efficient
- Flexible
- In your script, with `ob_start()`
- Everywhere, with `output_buffering` = Xkb
- Improves browser’s rendering speed

Apache: Output Control

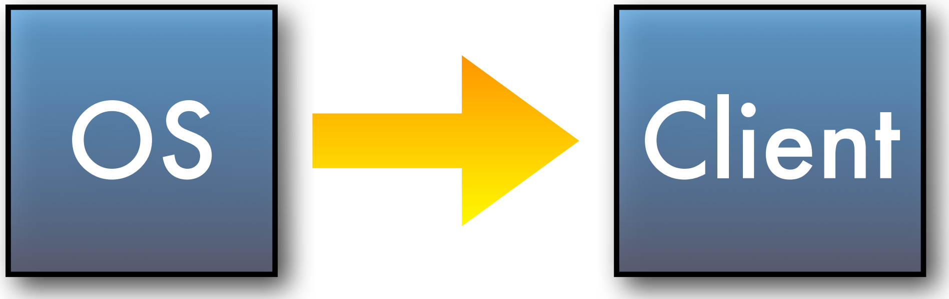
The idea is to hand off entire page to the kernel without blocking.



Set **SendBufferSize** = **PageSize**

OS: Output Control

OS (Linux)



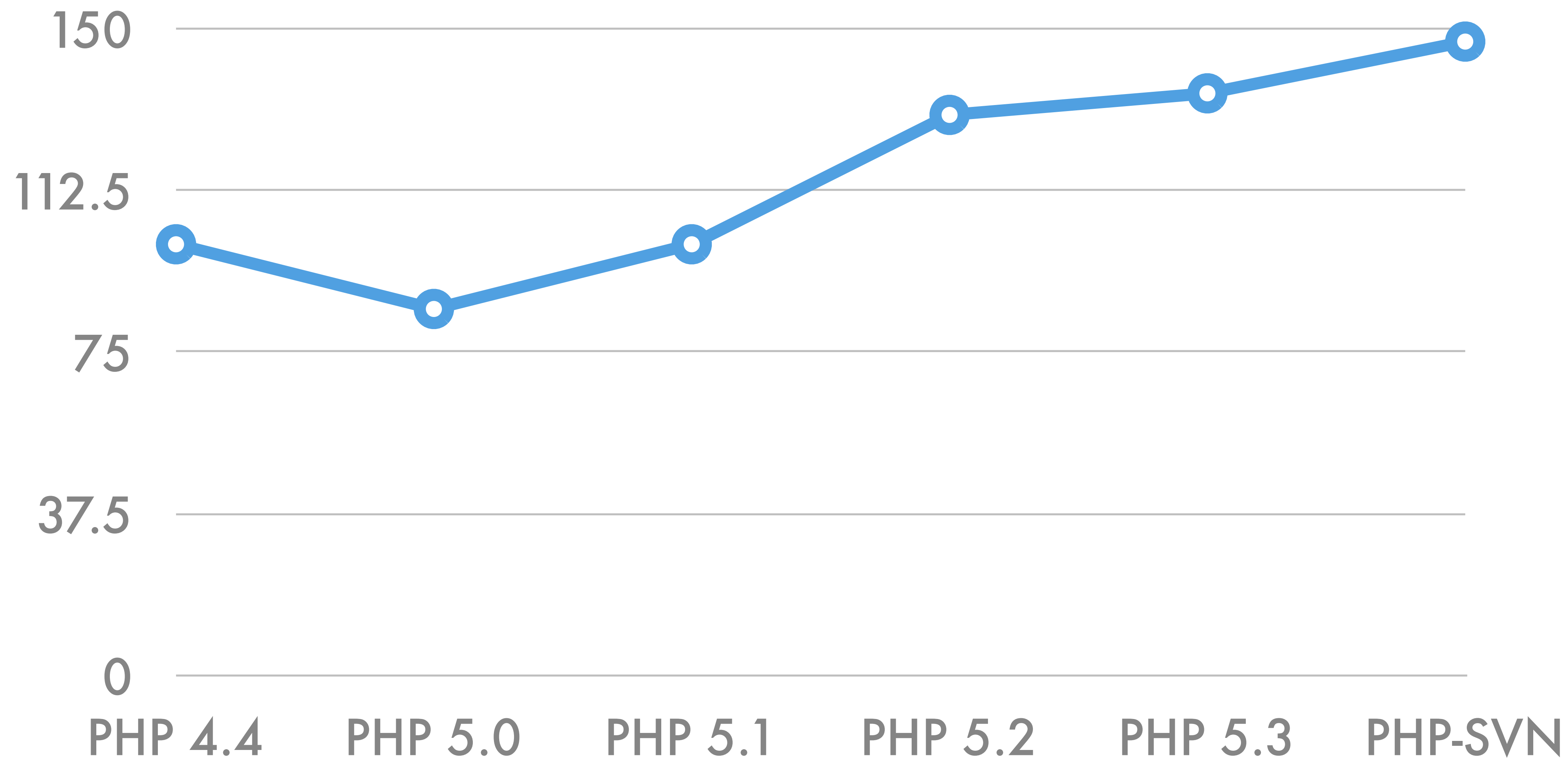
```
/proc/sys/net/ipv4/tcp_wmem
4096     16384    maxcontentsize
min          default        max
```

```
/proc/sys/net/ipv4/tcp_mem
(maxcontentsize * maxclients) / pagesize
```

- ✳ **Be careful on low memory systems!**

Upgrade Your PHP

Before “upgrading” your code, upgrade your PHP. Newer versions are typically faster!



Database before code

- One of the most common mistakes people make is optimizing code before even looking at the database
- Vast majority of applications have the bottleneck in the database not the code!

Watch Your Errors

- Excessive, non-critical errors, such as E_NOTICE or E_STRICT can only be detected via error-logging

- PHP code that generates any **errors** is going to **impact performance!**

Not Easily Detectable by Profilers

Micro Optimization

- Takes a long time
- Won’t solve your performance issues
- Almost guaranteed to break something
- Cost > Reward

Speed vs Scale

- If you are planning for growth, scale is far more important than speed!

- Focus on scalability rather than speed, you can always increase scalable app, by simply adding more hardware.

Don’t Re-invent the Wheel

Most attempts to make “faster” versions of native PHP functions using PHP code are silly exercises in futility.

Write Only Code

- Removing comments won't make code faster
- Neither will removal of whitespace
- Remember, you may need to debug that mess at some point ;-)
- Shorter code != Faster Code

Thank You!

Any Questions?

Slides @ www.ilia.ws

Comments @ joind.in/4026