



message
systems™



Micro Optimize This!

Brandon Savage



**What comes to your
mind when you think
“Micro Optimization”?**



Single quotes versus
double quotes

Print() vs. Echo()

Which loop is fastest

include() vs
include_once()

Static methods vs
object instantiation

Removing whitespace

It's Silliness.

```
/* Removing  
Comments */
```



**So how can we
optimize,
intelligently?**



The 80/20 Rule



**Truth: When people
talk about
“optimization” they
usually mean
“boosting
performance.”**



Micro* optimizations that DO work.

* Where "micro" refers to the amount of effort required



Hierarchy of Performance:

#1 Memory (RAM)

#2 Network I/O

#3 Disk I/O

(Note: This is a generalization)



**Faster request
handling = more
requests in the same
time period**



Standard Disclaimer Time!

- These ideas may not solve YOUR problem.
- The only way to know for sure is if YOU profile YOUR application.
- Premature optimization is the ROOT of all EVIL.



The good news.

And the bad news.



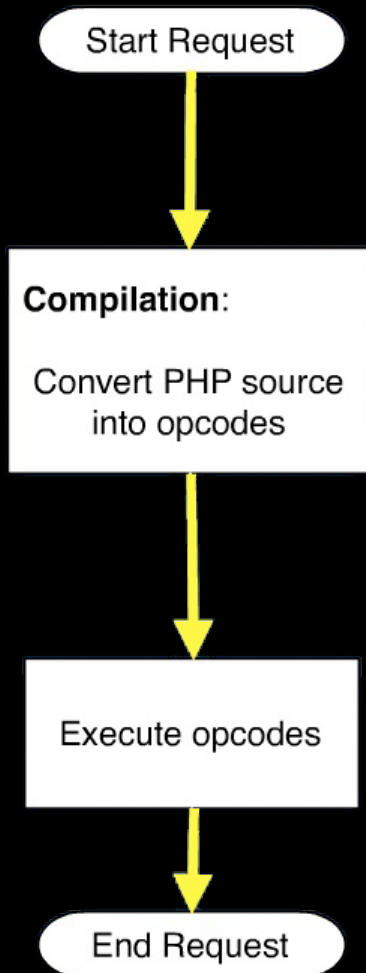
The Alternative PHP Cache (APC)



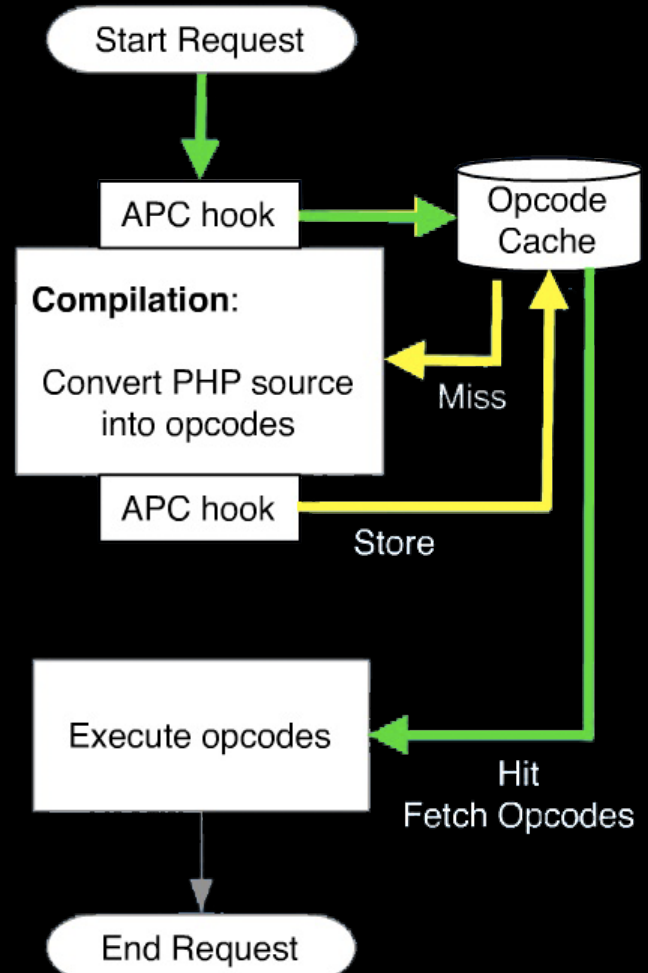
APC

- PHP is compiled at runtime.
- Each request compiles the file, then executes the opcodes to generate a response.
- Compilation is expensive.

PHP



APC



Borrowed from APC@Facebook



Making APC Even Faster

- By default, APC checks the status of a file to see if it has been updated since the last time it compiled the file.
- Most of the time, this won't be the case. So we can set `apc.stat = 0` in `php.ini`.



**WARNING: Disabling
this has
consequences!
Understand what they
are!**



APC.stat Consequences

- When updates are made, the cache must be cleared manually.
- Since the whole cache is invalidated at once, developers should consider `apc.slam_defense`.
- On low-memory machines, APC may dump the cached files anyway, rendering this setting pointless.



**Absolute paths are
faster than relative
paths for APC.**



Output Buffering



print() is expensive.

So is echo().



Without Output Buffering

- Every call to `echo()` or `print()` makes a system call to `write(2)`.
- This increases system load and network utilization.



Output Buffering

- Our goal: be as efficient as possible.
- To accomplish this, we want to pass as much work to the OS as we can at one time.
- Output buffering passes large chunks, reducing the overhead associated with system calls and may reduce total packets sent.



Enabling Output Buffering

- Use `ob_start()` to enable it in a single script.
- Use `output_buffering = On` for all of PHP.



WARNING: Be careful
with unlimited output
buffering on low
memory systems!



Memory Based Caches



Memory-Based Caches

- Since memory is faster than disk I/O, using a memory-based cache is a good alternative.
- APC has a user space cache for storing on the local machine.
- Memcache is the accepted standard for distributed memory-based caches.



Memory-Based Caches

- Consider using memcache to store sessions (PHP supports this by default)
- Consider using memory caches to store complex operations.



Caching Tradeoffs

- Caching can introduce strange, unpredictable behavior at the application level.
- Invalid cached data can wreak havoc over the user experience.
- Once memcache is distributed, it becomes a network-based cache, subject to latency.

Any sufficiently complex caching infrastructure is indistinguishable from black magic

7:46 AM Apr 9th via web



lukewelling
Luke Welling



Database Bottlenecks



In data-driven applications, the database is often the bottleneck.



**Assuming the
database is the
problem without
profiling is bad karma.**



**The database
combines two I/O
problems: Network
AND Disk.**



Caching A SQL Query

```
public function selectRows()  
{  
    $sql = 'SELECT * FROM table';  
    $key = md5($sql);  
    $data = $this->_cache->fetch($key);  
    if($data) {  
        return $data;  
    }  
  
    $data = $this->_executeQuery($sql);  
    $this->_cache->store($key, $data);  
    return $data;  
}
```



Everything has to be real time!





Log Only Important Errors



Which errors are important?

- These are production values!
- PHP 5.1, 5.2: `error_reporting = E_ALL`
- PHP 5.3: `error_reporting = E_ALL & ~E_DEPRECATED`



Reducing Errors

- In order to properly debug production problems, all errors must be logged.
- Each error being logged is added disk IO.
- Solution: reduce the number of errors being logged.



Reducing Errors

- Code with `error_reporting = E_ALL & E_STRICT`.
- When you see `E_NOTICE` and `E_WARNING`, **FIX THEM!**



Reducing Error Logging Load

- All services will have periods of high errors or component failure.
- Consider placing error logging on another set of disks to avoid torturing core servers during high error periods.



Don't Forget Webserver Errors (404, 500, etc.)

They're Expensive, Too!



Consider an autoloader



Autoloaders

- There's much debate as to the efficacy of autoloading for reducing execution time.
- After seeing benchmarks on autoloading Zend Framework, I believe they help.
- Profile your own apps and decide for yourself.



Autoloaders

- Standard practice: include all the files you need; many times, include all the files in one giant list, so they're available globally.
- Problem: each included file must be parsed, compiled and executed!
- Autoloading only includes the files you need, when they are needed.



Autoloader Example

```
function __autoload($class_name)
{
    require '/www/includes/' . $class_name . '.php';
}
```



Autoloaders: Drawbacks

- Autoloaders only really work with object-oriented code.
- Autoloaders can be difficult to configure or debug.
- If you use `__autoload()` instead of `spl_autoload_register()`, you may find conflicts in other libraries/packages.



Invest in hardware



Hardware is cheap.

Disk I/O, not so much.



**This is great, but I
already do all that.**

**I'm facing buying 10
new boxes. What can
I do?**



**Assume Nothing.
Profile Everything.**



Profiling

- Without profiling, it's impossible to know what the actual bottlenecks are.
- Take a good look under the hood, both at what your code is doing AND what PHP is doing.



Profiling

- Cachegrind is a great tool for interpreting profiler results.
- Xdebug is a great tool for producing profiling results.



Some Micro Optimizations That DO Work

- Calling the same functions repeatedly is expensive. Use data already obtained.
- Some open source libraries may be designed for many use cases; consider writing your own.
- PHP frameworks and open source tools may also be slow and not optimized.



**It bears repeating:
PROFILE
EVERYTHING!**



**Looking for
challenging work?**

**Message Systems is
hiring!**



Contacting Me

- I'm available on IRC, Twitter, through Email...
- Email: brandon@brandonsavage.net
- Twitter: [brandonsavage](https://twitter.com/brandonsavage)
- Freenode: You'll find me in #zftalk frequently



Questions?