Profiling with XHProf

Ilia Alshanetsky

@iliaa

Me, Myself and I

PHP Core Developer

Author of
Guide to PHP Security

CIO at Centah Inc.

Why are we here?

Speed Matters!

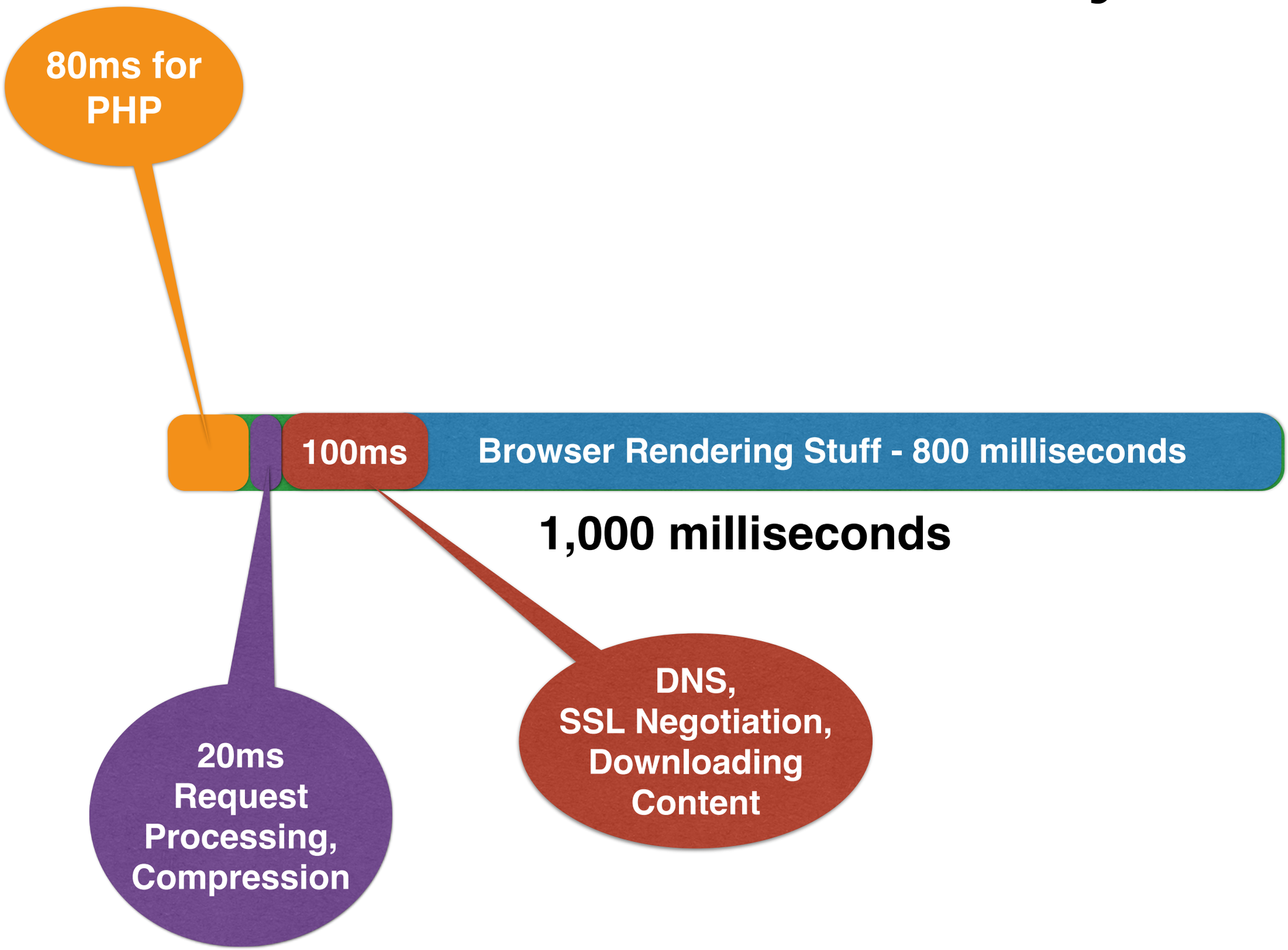
http://www.phpied.com/the-performance-business-pitch/

Speed

Greater User Engagement

Speed is all about perception!

The Web Story



Making Things Faster

=

Eliminating Bottlenecks

You find bottlenecks
by ~~Benchmarking~~!
Profiling!

Benchmarking Sucks!

Profiling Can Suck Too!

Profiling in Production

- Needs to be fast
- Simple to Implement
- Not introduce “breakage”
- Must be aggregatable

This is where

XHProf Excels!

Installation

- **Two Part Process**
- **First install the PHP Extension**
- **Then install the storage engine & results interface.**

PHP Extension

If you are brave, you can compile from source

- **Via PECL**

  pecl install xhprof

- **Via Package Manager**

  apt-get install php5-xhprof

.....

- **Enable inside php.ini**

  extension=xhprof.so

XHProf Gui

- **For MySQL backend**
  - **https://github.com/preinheimer/xhprof**
- **For MongoDB backend**
  - **https://github.com/perftools/xhgui**

The One “File” Trick

php_auto_prepend=“/path/to/magic.php”

```
class MagicProfiler {
    static public $profiling = 0;

    public function __construct() {
        self::$profiling = !(mt_rand() % 9);
        if (self::$profiling) {
            xhprof_enable(XHPROF_FLAGS_CPU | XHPROF_FLAGS_MEMORY);
        }
    }

    public function __destruct() {
        if (self::$profiling) {
            $data = xhprof_disable();
            include '/path/to/xhprof/xhprof_lib/config.php';
            $GLOBALS['_xhprof'] = $_xhprof;
            include '/path/to/xhprof/xhprof_lib/utils/xhprof_lib.php';
            include '/path/to/xhprof/xhprof_lib/utils/xhprof_runs.php';
            $x = new XHProfRuns_Default();
            $x->save_run($data, 'App Name', null, $_xhprof);
        }
    }
}
$me = new MagicProfiler();
```

Configuration File

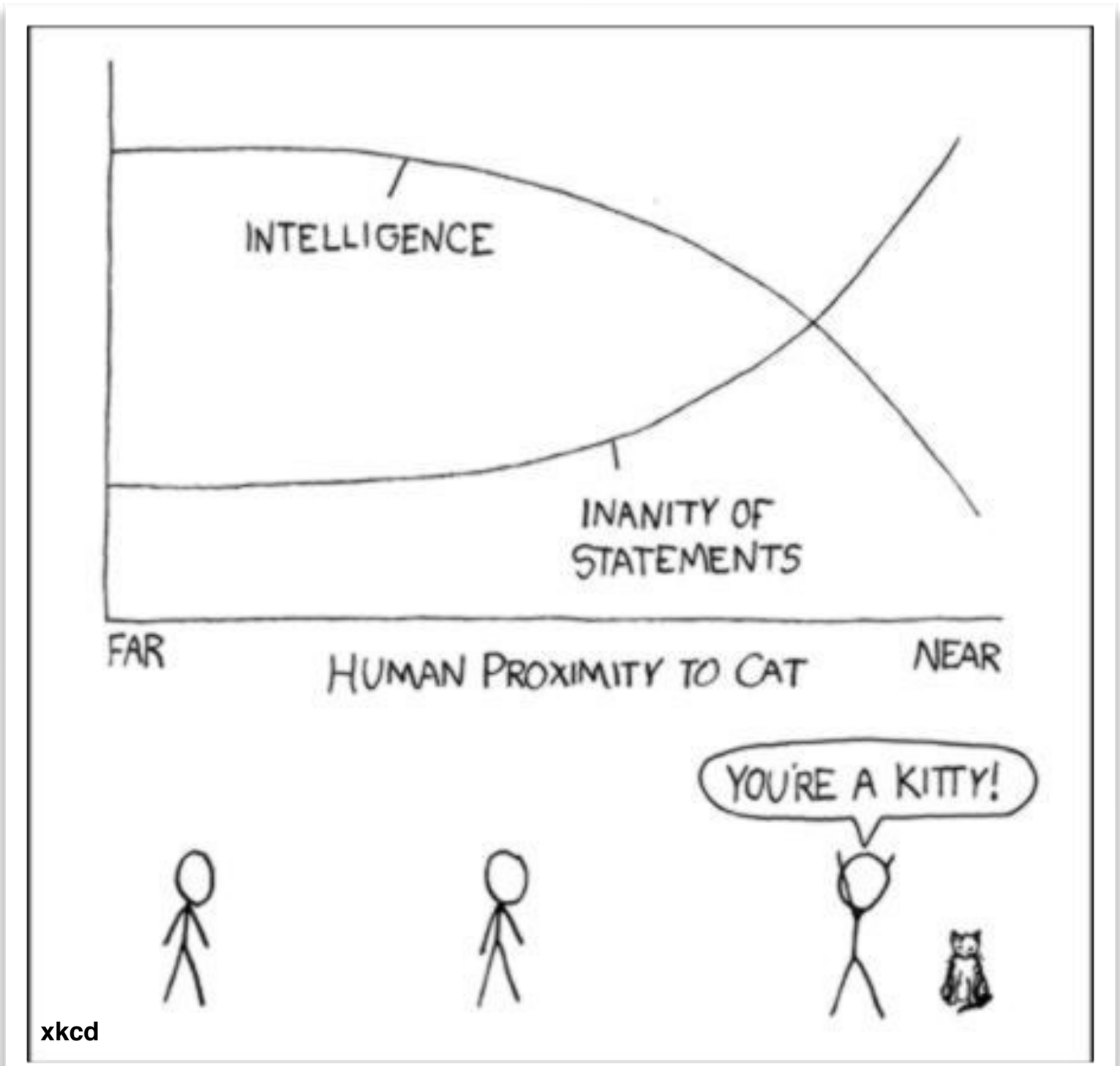
```
$_xhprof['dbtype'] = 'mysql'; // Only relevant for PDO
$_xhprof['dbhost'] = 'localhost';
$_xhprof['dbuser'] = 'root';
$_xhprof['dbpass'] = 'root';
$_xhprof['dbname'] = 'profile';
$_xhprof['dbadapter'] = 'Pdo'; // Or mysql or mysqli
$_xhprof['namespace'] = 'myapp'; // could be different per-app
```

```
// I recommend using igbinary
$_xhprof['serializer'] = 'php';

// Ignore URLs containing the following keywords
$ignoreURLs = array('/images/');

// Ignore requests for the domains with following keywords
$ignoreDomains = array('static.');

// Don't log POST input for URLs containing the following
$exceptionPostURLs = array('login','payment');
```

Configuration File Part 2

```
$controlIPs = false; // Disabled by default
// this means everyone's got access

// Otherwise list IPs that can manually
// trigger profiling via GET _profile=1
$controlIPs = array();
// both IPv4 and IPv6 values are allowed
$controlIPs[] = '127.0.0.1';
```

Interpreting the Results



xkcd

General Summaries

Timestamp	Cpu	Wall Time	Peak Memory Usage	URL	Simplified URL
Sep 17 11:41:25	6381030	8627592	62294608	/?_%5Bpage%5D=scheduler&_	/?_%5Bpage%5D=scheduler&_%5Bstate_id%5D=5_53.
Sep 22 16:27:54	4883258	5233343	74901608	/?_%5Bpage%5D=scheduler&_	/?_%5Bpage%5D=scheduler&_%5Bstate_id%5D=0_21.
Sep 17 13:58:13	4112375	4338331	71964272	/?_%5Bpage%5D=scheduler&_	/?_%5Bpage%5D=scheduler&_%5Bstate_id%5D=4_59.
Sep 22 16:29:17	3720435	3914633	42557944	/?_%5Bpage%5D=scheduler&_	/?_%5Bpage%5D=scheduler&_%5Bstate_id%5D=6_13.
Sep 17 15:35:52	3572456	5574733	182382752	/?_[event][][back]=1&_[page]=show_	/?_[event][][back]=1&_[page]=show_order_audit&_[st
Sep 18 14:55:24	3341492	4823630	181026920	/?id=671566&_[component]	/?id=66&_[component]=545&_[state_id]=8_2

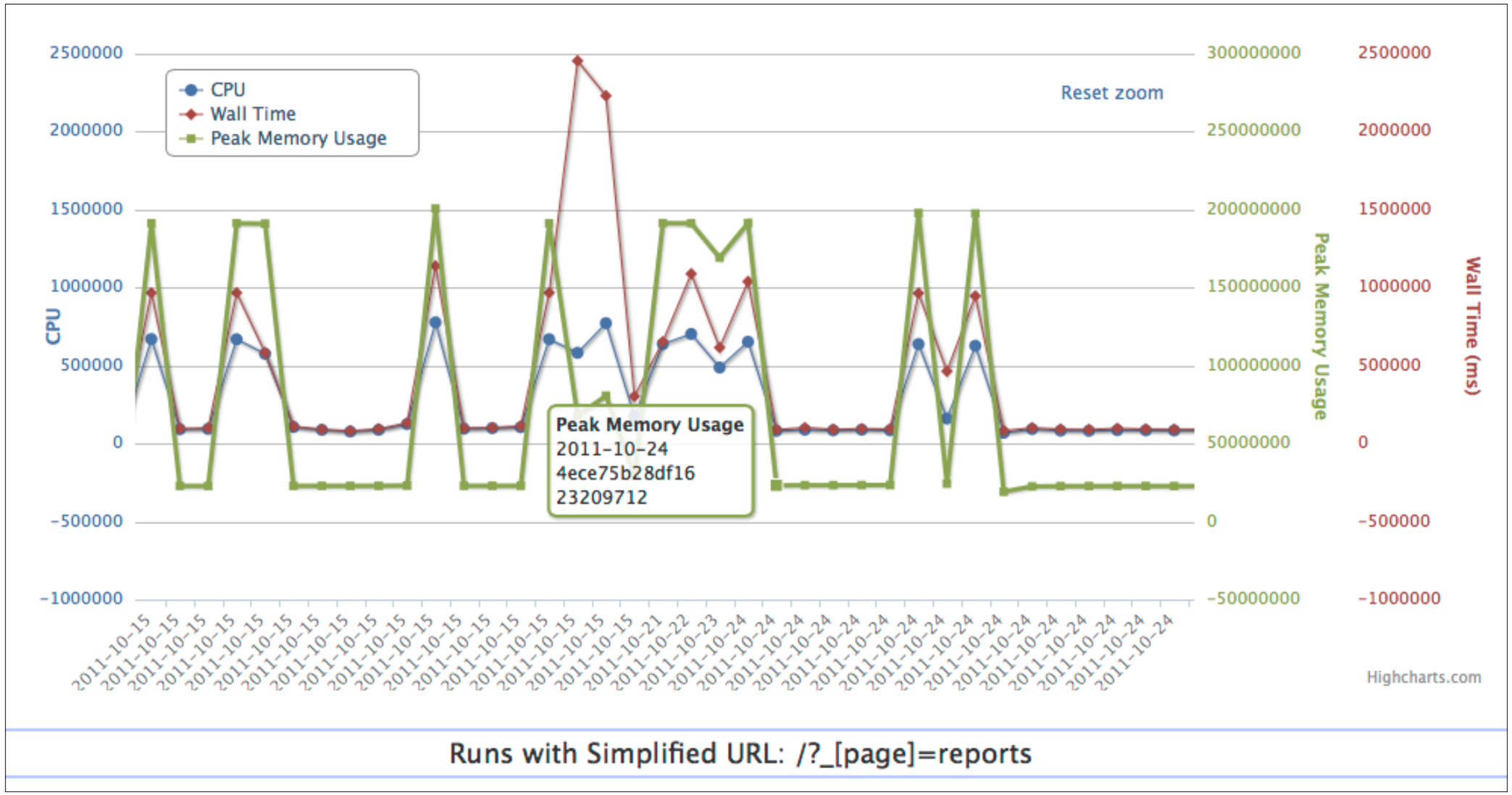
- **Hardest Hit**
- **Slowest**
- **Biggest memory hogs**
- **Longest Running**

Today or Week to Date

Trending Baseline

Stat	Exact URL	Similar URLs
Count	5596	5596
Min Wall Time	20.1520 ms	20.1520 ms
Max Wall Time	2.4764 minutes!	2.4764 minutes!
Avg Wall Time	686.5729 ms	686.5729 ms
95% Wall Time	2.0456 s	2.0456 s
Display run Incl. Wall Time (microsec)	**1,006,332 microsecs**	
Min CPU Ticks	16.9970 ms	16.9970 ms
Max CPU Ticks	2.4066 s	2.4066 s
Avg CPU Ticks	385.5804 ms	385.5804 ms
95% CPU Ticks	1.0678 s	1.0678 s
Display run Incl. CPU (microsecs)	**923,859 microsecs**	
Min Peak Memory Usage	6,006,952 bytes	6,006,952 bytes
Max Peak Memory Usage	309,707,216 bytes	309,707,216 bytes
Avg Peak Memory Usage	72,154,616 bytes	72,154,616 bytes
95% Peak Memory Usage	208,371,968 bytes	208,371,968 bytes
Display run Incl. PeakMemUse (bytes)	**168,900,960 bytes**	
Number of Function Calls:	**10,123**	

Trending



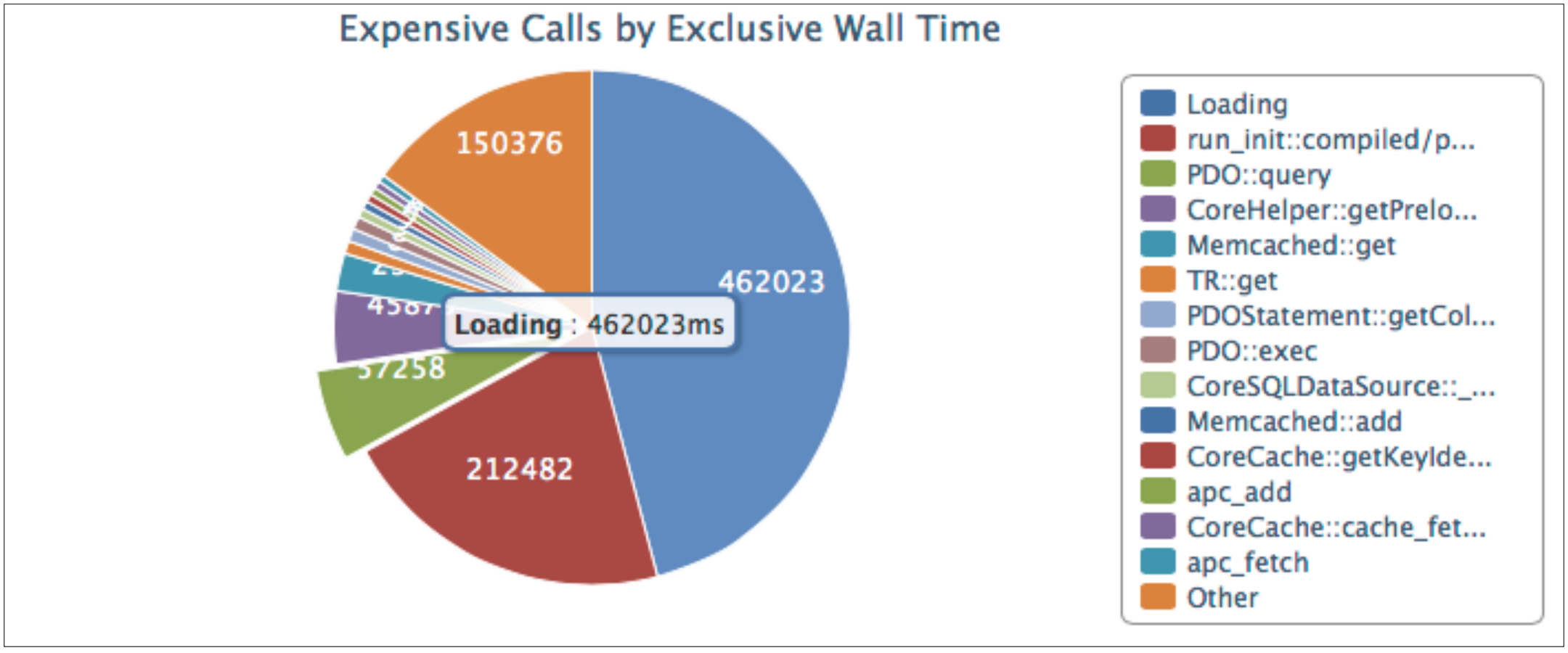
Runs with Simplified URL: /?_[page]=reports

Specific Run Comparison

http://xhprof/?run1=RUN_ID&run2=RUN_ID

	Run One ID: 5419abae5c36b	Run Two ID: 5420864fef1ba	Diff	Diff%
Number of Function Calls	1,334,557	170,517	-1,164,040	-87.2%
Incl. Wall Time (microsec)	8,627,592	5,233,343	-3,394,249	-39.3%
Incl. CPU (microsecs)	6,381,030	4,883,258	-1,497,772	-23.5%
Incl. MemUse (bytes)	53,870,696	67,092,616	13,221,920	24.5%
Incl. PeakMemUse (bytes)	62,294,608	74,901,608	12,607,000	20.2%

In-Depth Specifics



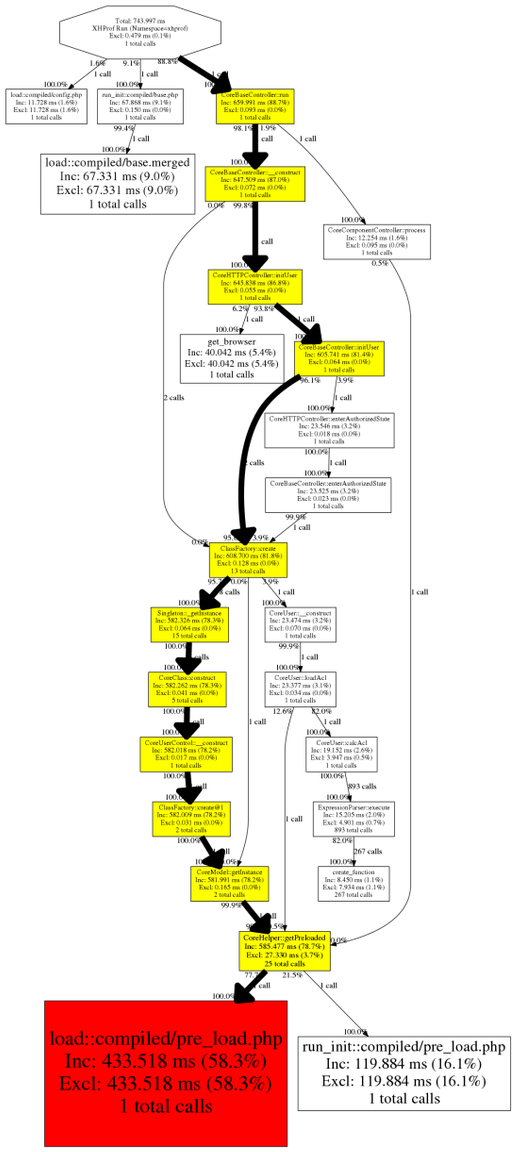
Really In-Depth

Function	Call Count	Wall Time	CPU	Memory Usage	Peak Memory Usage	Exclusive Wall Time	Exclusive CPU	Exclusive Memory Usage	Exclusive Peak Memory Usage
main()	1	8627592	6381030	53870696	62294608	668	1000	-98416	552
CoreBaseController::run	1	8528193	6282045	38455144	46846056	85	0	3040	0
CoreComponentController::proce	1	8522065	6277046	36747664	44988216	158	0	6840	0
CoreComponent::dispatchEvent	3	8202059	6096073	17628888	26802216	54	0	2848	0
CoreEventDispatcher::sendEvent	2	8201893	6096073	17621056	26802216	27	0	-2008	0
CoreEventDispatcher::sendEvent	1	8201852	6096073	17621336	26802216	22	0	1592	0
CoreComponentFactory::dispatch	1	8201740	6096073	17613584	26802216	27	0	1728	0
SchedulerPageComponent::onSubm	1	8201644	6096073	17607464	26802216	23298	15995	-842344	0
CoreModel::insert	4	8102264	6019087	14610480	22661424	28299	14999	-15023888	192
CoreModel::createPlan	3137	4859440	3539457	22594160	22167056	1493159	863857	-22767328	7288
CoreModel::executePlan	3137	3169555	2422637	3377328	0	104107	27996	-13932208	0

Graphical Trace

Requires Graphiz package

```
apt-get install graphic
```



Profiling Tricks

Only Examine User Functions

```
xhprof_enable(
  XHPROF_FLAGS_NO_BUILTINS |
  XHPROF_FLAGS_CPU |
  XHPROF_FLAGS_MEMORY
);
```

Ignore Functions

```
xhprof_enable(
    // usual flags,
    array('ignored_functions' => array(
        'exec',
        'ClassName::__destruct'
    ))
);
```

Lightweight Profiling

```
// use
xhprof_sample_enable();
   // instead of xhprof_enable();
   // termination of profiling is done via
   xhprof_sample_disable();
```

** Does not support flags or function ignore lists

User Tracing

```
session_start();

if (empty($_SESSION['user_id'])) {
     $_SESSION['user_id'] = init_user();

        if (!($_SESSION['user_id'] % 10)) {
            $_SESSION['profile'] = 1;
        }
 }
```

** Avoid using Cookies or GET/POST params for triggering profiling

Database Maintenance

Profiling DBs grow quickly, so clean-up data frequently if you don't have a lot of storage place…

42k profiled **pages**, take **500+ MB** in MySQL

https://github.com/gajus/xhprof.io

Alternative UI to XHGUI

A nicer UI, simpler code, but
appears to be abandoned
for >1 year now :-(

Commercial Solution also available via Tideways profiler**

Honorary Mentions ;-)

BlackFire from Sensio Labs

AppDynamics

New Relic

THANK YOU FOR LISTENING

Slides: http://ilia.ws

@iliaa