PHP ON THE METAL

Keith Adams
kma@fb.com

THE HIPHOP VIRTUAL MACHINE

- HHVM is the world's fastest PHP engine
- https://github.com/facebook/hiphop-php
- JIT compiler for development and production
- Nickel tour of the JIT
- Perf-oriented perspective on its development
- A new approach to cache profiling
- Lessons learned

MOTIVATION

BACKGROUND: PHP

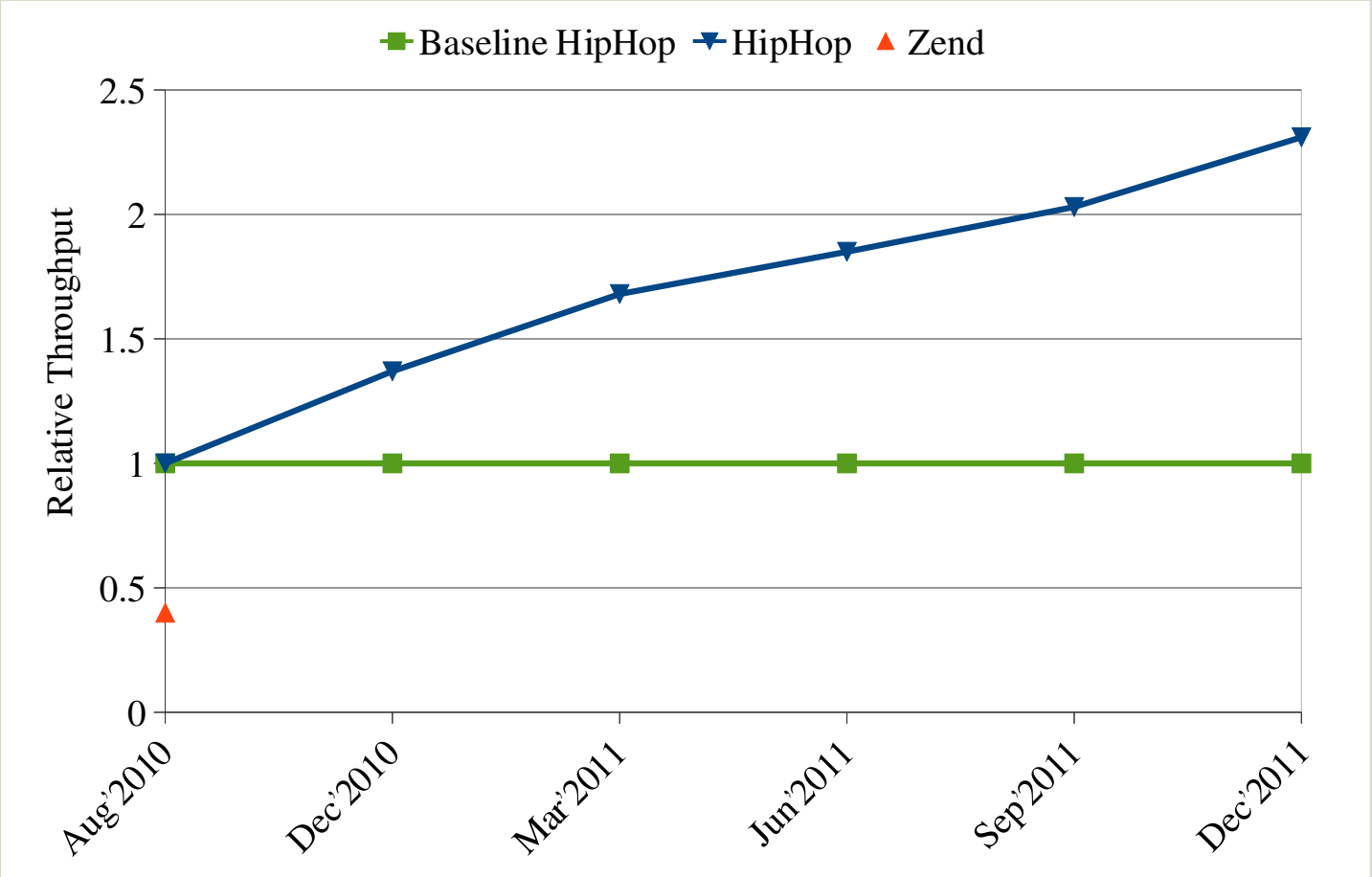
- Your average “developer productivity” language
- Dynamic bindings for everything
- Variables are untyped

```
<?php
function max($a, $b) {
  return $a > $b ? $a : $b;
}
echo max(1, 2);
echo max(“abe”, “zebra”);
```

BACKGROUND: HIPHOP

- Interpreter, debugger, profiler, AoT compiler
- AoT offers 2-7x win over interpreted PHP
- Paper in OOPSLA ’12
- Crucial optimization: *type inference*

PRODUCTION THROUGHPUT



From "The HipHop Compiler for PHP," Zhao et al., OOPSLA 2012

HARD EXPRESSIONS FOR HPHP

```
goldbach_conjecture() ? 3.14159 : “string”
```

```
mysql_fetch_row($result)[0]
```

```
123.2 / $divisor
```

HHVM: THEORY

- HHVM vision
  - Incremental compilation
  - Same engine in dev and prod
  - Optimize in response to program behavior
  - Type *every datum* in the system!
- Higher performance, more cohesion, faster dev environment
  - Win/win/win!

HHVM CORE DESIGN

- PHP programs are represented in bytecode (HHBC)
- JIT Goal: Never operate on generic data
- Compilation unit: the *Tracelet*
  - *Basic block, with concrete input types*
  - Use the concrete input types to guard tracelet entry
  - Inside the tracelet, exploit type information
  - If type inference fails, break the Tracelet and reguard

HHBC

```
function mymax($a, $b) {
  return $a > $b ? $a : $b;
}
```

```
    PushL 1
    PushL 0
    Gt
    JmpZ 1f
    PushL 0
    Jmp 7 2f
1:  PushL 1
2:  RetC
```

TRACELET CONSTRUCTION: MACHINE CODE

- mymax(10, 333);

```
Local0 :: Int
Local1 :: Int
PushL 1
PushL 0
Gt
JmpZ X
```

```
cmpl   $0x3,-0x4(%rbp)
jne    <retranslate>
cmpl   $0x3,-0x14(%rbp)
jne    <retranslate>

mov    -0x20(%rbp),%rax
mov    -0x10(%rbp),%r13
mov    %r13,%rcx
cmp    %rax,%rcx
jle    <translateSuccessor0>
jmpq   <translateSuccessor1
```

HHVM: PROTOTYPE

- 6-month, 3-man effort
  - Drew Paroski, Jason Evans, Keith Adams
- PHP subset
- Showed real promise
  - microbenches
  - kernel extracted from Facebook's production code
- We decide to move forward...

FROM PROTOTYPE TO PRODUCTION

- PHP: a *big* language
  - Lots of non-orthogonal features
  - Doesn't boil down to a few key primitives
  - Corner cases
- Facebook's codebase: ~20 MLOC
  - Exercises *all* of PHP
  - ...and some new parts we invented

HHVM: PRACTICE

- 12 months later: Facebook runs in HHVM
- ~13% of the compiler's performance
- 7x slower

LOW-HANGING FRUIT

- Profiling found hot spots
- We optimized them…
- and things got a lot better!

watermelons by matneym
flickr creative commons

…BUT NOT GOOD ENOUGH

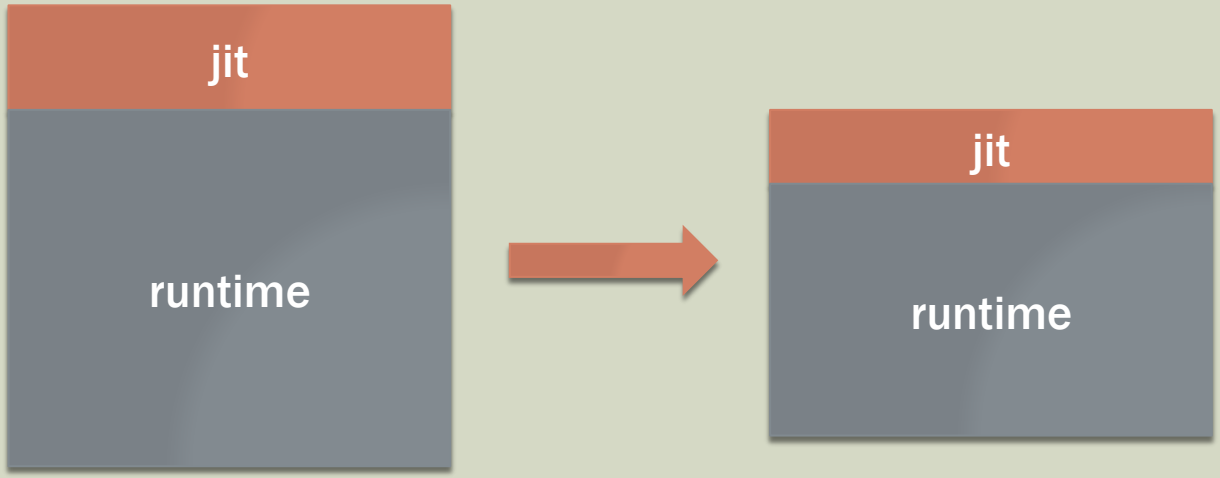
- April 2012: performance stagnates
- ~50%, 2x slower
- Flat CPU profile
  - ~18% of time spent in JIT output
  - Long tail of runtime functions
  - memory allocation
- Diminishing returns to “measure and tune” methodology

SOME SCARY QUESTIONS

- Was there something *fundamentally wrong* with our design?
- Was the system *not working as designed*?

A CLUE

- Jordan DeLong changed our strategy for chaining tracelets together
- Got a *14%* win!
- Only *18%* of time spent in JIT output, both before and after
- Somehow, improving the JIT made *all the other code* faster, too



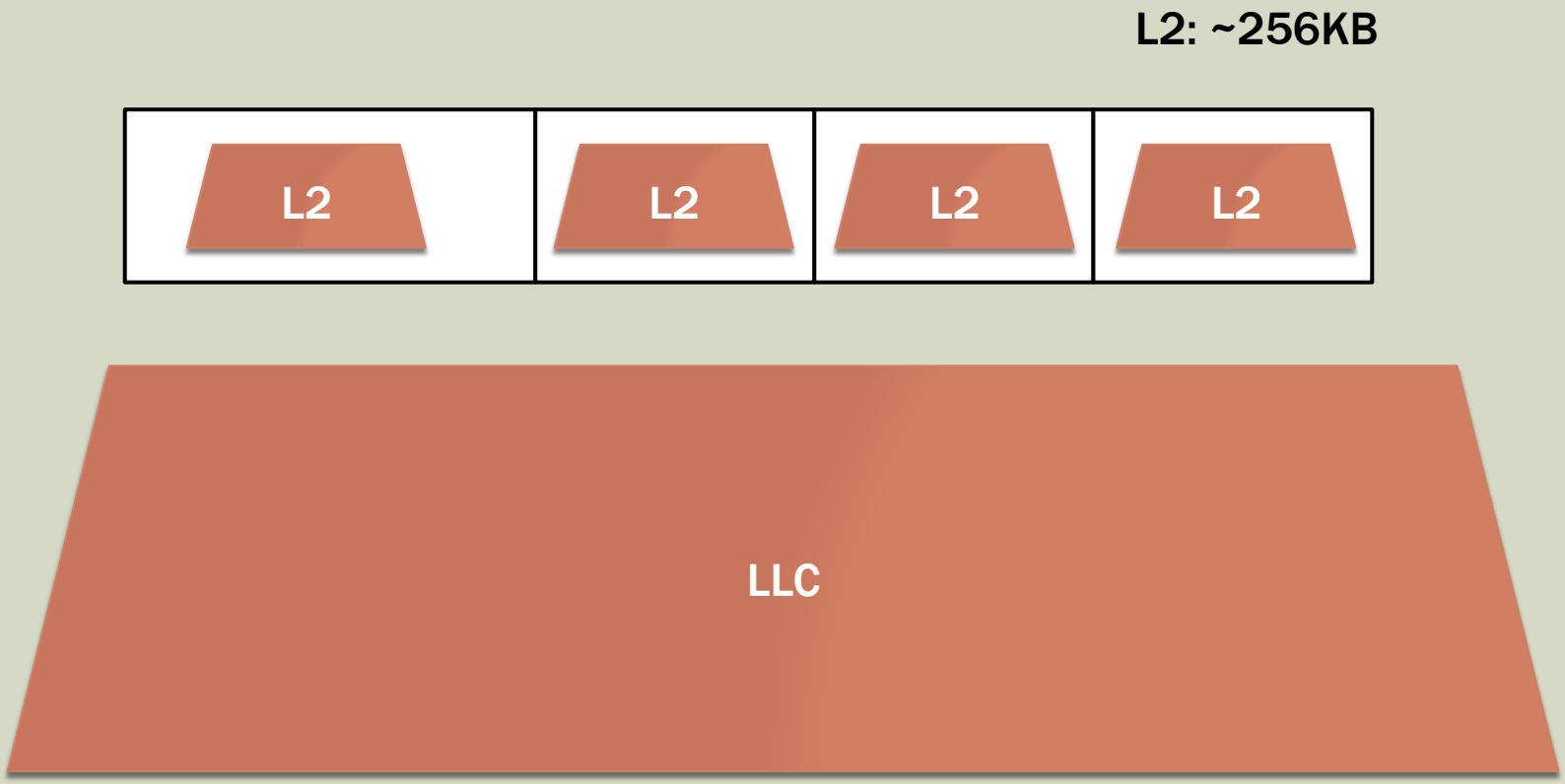
SPOOKY ACTION-AT-A-DISTANCE

- When code makes unrelated code faster or slower, *suspect caching*.
- Cache is a shared, stateful resource
- Medium for performance teleportation

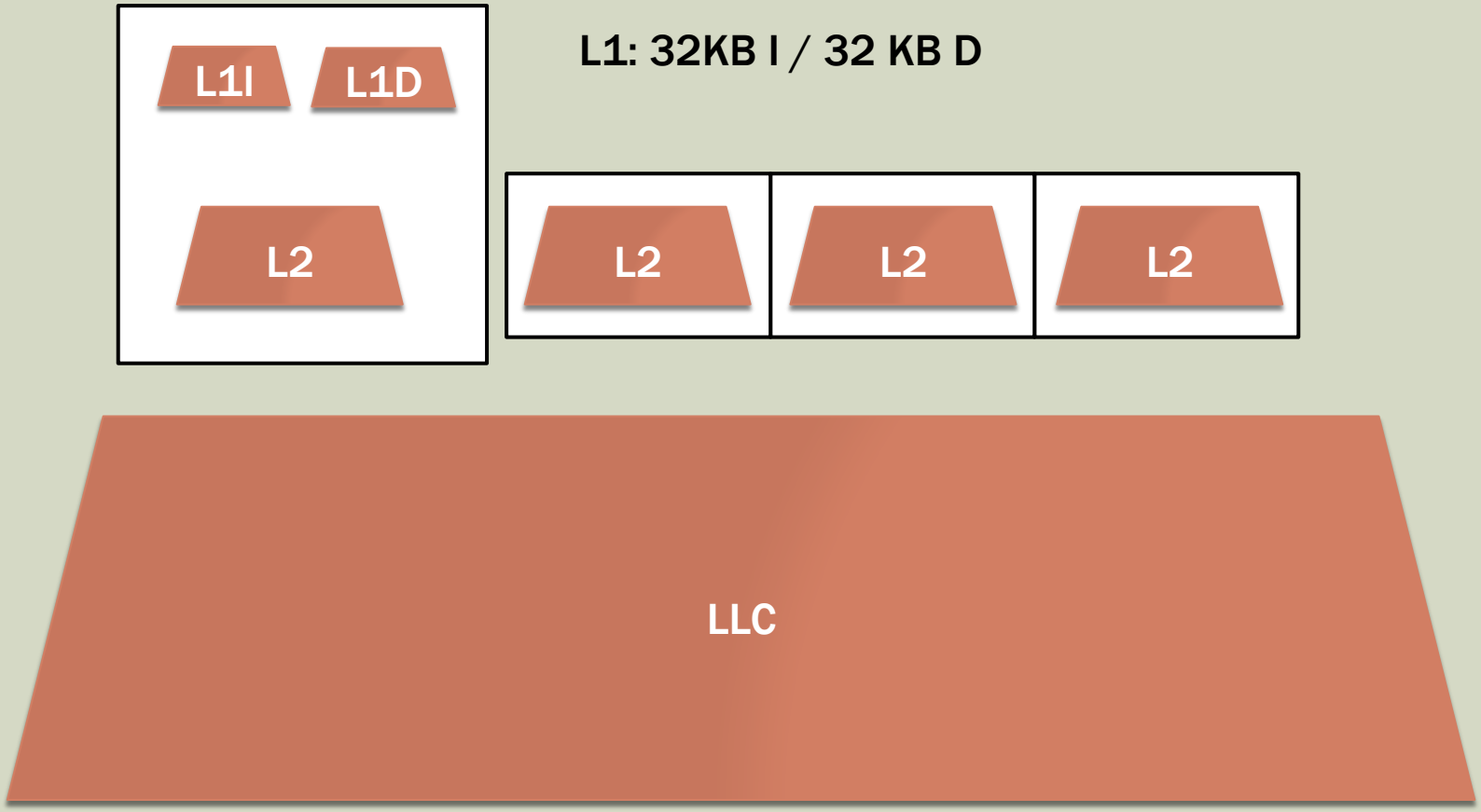
MEMORY HIERARCHY

LLC: ~16MB

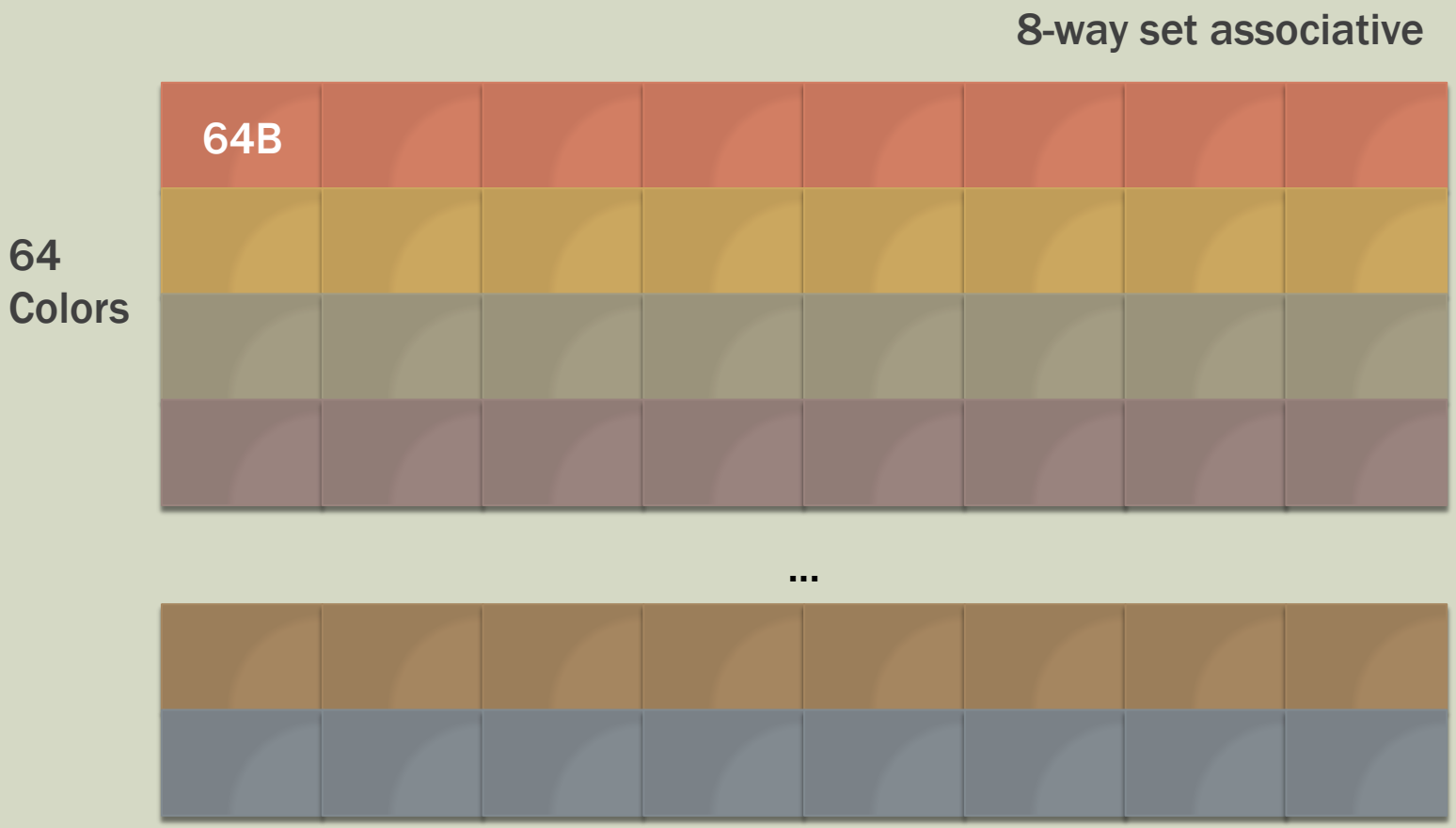
MEMORY HIERARCHY



MEMORY HIERARCHY



OUR CACHES, OURSELVES



Sandy Bridge L1 icache: total 32KB

CACHE SIZE TREND

Date	CPU	L1 dcache capacity
1992	Sun SuperSPARC	16KB
1996	DEC Alpha 21264	64 KB
1999	Intel Pentium III	16 KB
2003	AMD Opteron	64 KB
2004	IBM POWER5	32KB
2007	ARM A8 Cortex	16KB
2012	Intel Sandy Bridge	32 KB

32KB

- ~8,000 instructions
- ~1000-2000 lines of C

- This is all the code or data a core can see at a time

PROFILING FAILS FOR CACHE MISSES

- Histograms of misses lead to bogus conclusions
- Tells you *what* is not in cache
- *Cannot* tell you *why* it is not in cache
  - It used to be
  - What pushed it out?

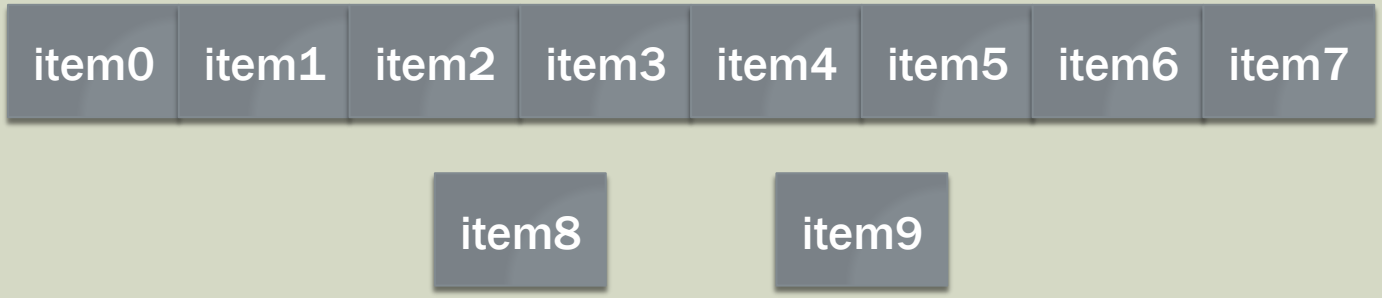
EXAMPLE

```
for i = 0 to M
        touch item0, item1, .. item8
        for j = 0 to N
                touch item9
```

- 10 items sharing a way
- Loop takes 10M cache misses
- Get rid of one: 9M
- Get rid of any *two*: 0

- Cache miss profiles show 10 separate, equally important problems, when there is only one problem

EXAMPLE

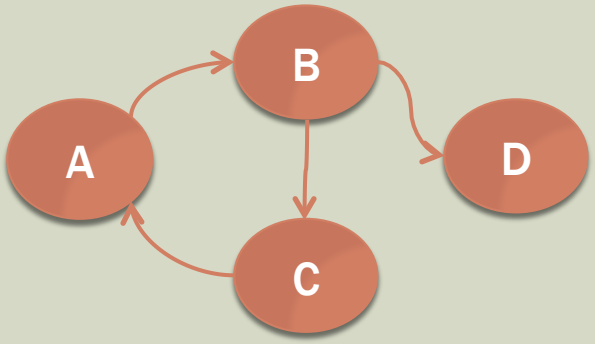


- In a complex profile, it's unclear what is interfering with what
- Every miss is also an eviction, but hardware tells you what missed, not what was evicted
- We want to ask "what if" questions: if I get rid of these misses, what happens?

ABSTRACTION: INTERFERENCE GRAPH

- The edge A->B means “A evicted B”
- Edge weighted by frequency of eviction

- Heuristic: Focus optimization effort on high-weight cycles in this graph



TRACE-BASED CACHE PROFILING

- Step 1: Pin-based instruction trace generator
  - Instruments *every single instruction*
  - Dumps 1 million out of every billion

```
0x1bfcd61
0x1bfcd64
0x1bfcd65
0x1bfcd68
0x1bfcd6c
0x1bfc8a0
0x1bfc8a1
0x1bfc8a4
0x1bfc8a7
0x1bfc8ab
0x1bfc8ae
0x1bfc8b1
0x1bfc8b3
0x1bfc8b6
0x1bfc8bc
0x1bfc8be
0x1bfc8c1
0x1bfc8c4
```

TRACE-BASED CACHE PROFILING

- Step 2: Build a simple cache simulator
  - https://github.com/kmafb/cachesim
- Dumps contents of cache at every eviction
- Entries that evict one another frequently are *interfering*

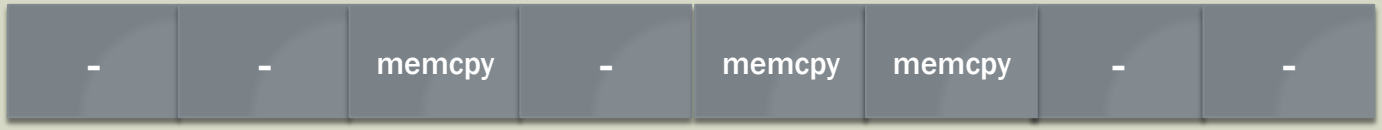
```
evict 0x250bb1bc0 0x3807ff38ac01bc1 newer 0x2501660bc0
0x2407ff38c17dbc0 0x240bb1bc0 0x2401c6fbc0
0x2507ff38c17bbc0 0x2501be9bc0 0x2407ff38c17bbc0
miss     950875 0x3807ff38ac01bc1
evict 0x2507ff38c17bc00 0x3807ff38ac01c08 newer
0x2401e1ec00 0x2407ff38c17dc00 0x2401c71c00 0x2401c6fc00
0x240bb1c00 0x2501660c00 0x2407ff38c17bc00
miss     950881 0x3807ff38ac01c08
evict 0x2501fd4680 0x3807ff38ac04680 newer 0x2401c02680
0x2401c70680 0x2401656680 0x250ba6680 0x2501656680
0x2401655680 0x3807ff38aec2680
miss     951104 0x3807ff38ac04680
```

HHVM ICACHE TRACE RESULTS

- An offender in lots of high-weight cycles: memcpy
- memcpy hopes
  - super small
  - super hot
  - how can it miss in cache?

ICACHE AND MEMCPY

- Our system's memcpy: 11KB!
- Specialized for size, source/dest overlap, CPU, alignment, etc.
- *Awesome* in memcpy microbenchmarks
- *Fragile* in the cache



FBMEMCPY

- Solution: “worse” memcpy
- Good for about 1%
- Nice! But no miracle

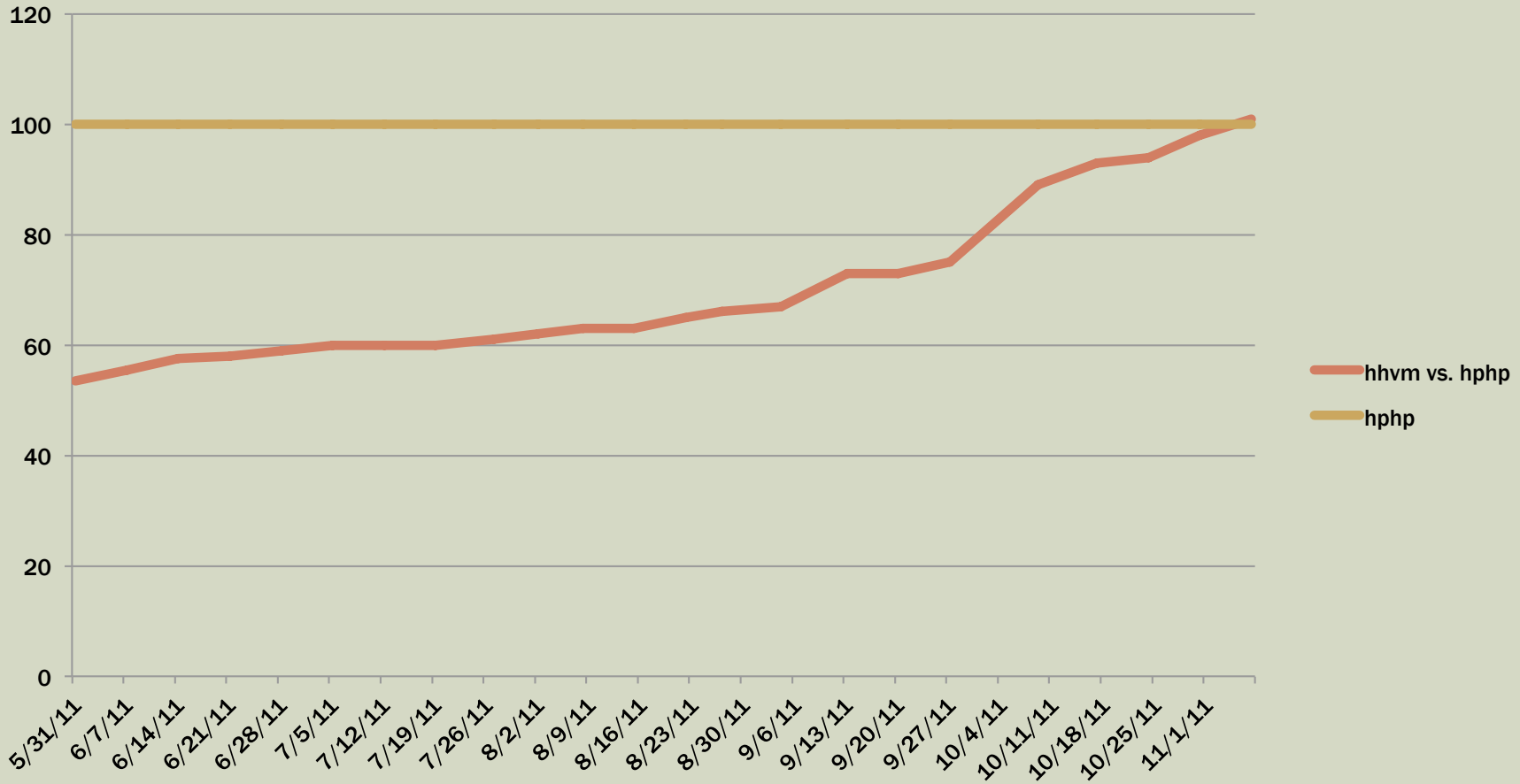
```
extern "C" {

HOT_FUNC
void*
memcpy(void* vdest, const void* vsrc, size_t len) {
  auto src  = (const char*)vsrc;
  auto dest = (char*) vdest;
   ...
  // Do the bulk with fat loads/stores.
  ASSERT((len & 0x3f) == 0);
  while (len) {
    auto dqdest = (__m128i*)dest;
    auto dqsrc = (__m128i*)src;
    __m128i xmm0 = _mm_loadu_si128(dqsrc + 0);
    __m128i xmm1 = _mm_loadu_si128(dqsrc + 1);
    __m128i xmm2 = _mm_loadu_si128(dqsrc + 2);
    __m128i xmm3 = _mm_loadu_si128(dqsrc + 3);
    len -= 64;
    dest += 64;
    src += 64;
    _mm_storeu_si128(dqdest + 0, xmm0);
    _mm_storeu_si128(dqdest + 1, xmm1);
    _mm_storeu_si128(dqdest + 2, xmm2);
    _mm_storeu_si128(dqdest + 3, xmm3);
  }
  return vdest;
}
```

NO MIRACLES

- How did we get twice as fast?
- By getting 1% faster *over and over*

HHVM PERF


120
100
80
60
40
20
0
5/31/11
6/7/11
6/14/11
6/21/11
6/28/11
7/5/11
7/12/11
7/19/11
7/26/11
8/2/11
8/9/11
8/16/11
8/23/11
8/30/11
9/6/11
9/13/11
9/20/11
9/27/11
10/4/11
10/11/11
10/18/11
10/25/11
11/1/11
hhvm vs. hphp
hphp


Happy
1.00

SCARY QUESTIONS ANSWERED

- Basic design was sound
- ...and the system was working as designed

- Initial performance gap due to Unreasonable Effectiveness of Tuning

TACTICAL LESSONS

- When the profiler works, use it
- Your CPU is still a *microcomputer*
  - Can only see 16-64KB of code, data at a time
- Spooky action-at-a-distance is caused by cache interference
- Count-based cache profiles can hide opportunities
- Trace-based cache profiles rock, but tools are non-existent

STRATEGIC LESSONS

- Replacing a working, tuned system will take longer than you think
- Big, sweeping changes were a mirage
- Sometimes seeing a fundamentally sound system through requires, well, *faith*
  - or at least, tolerance of existential doubt

TEAM HHVM











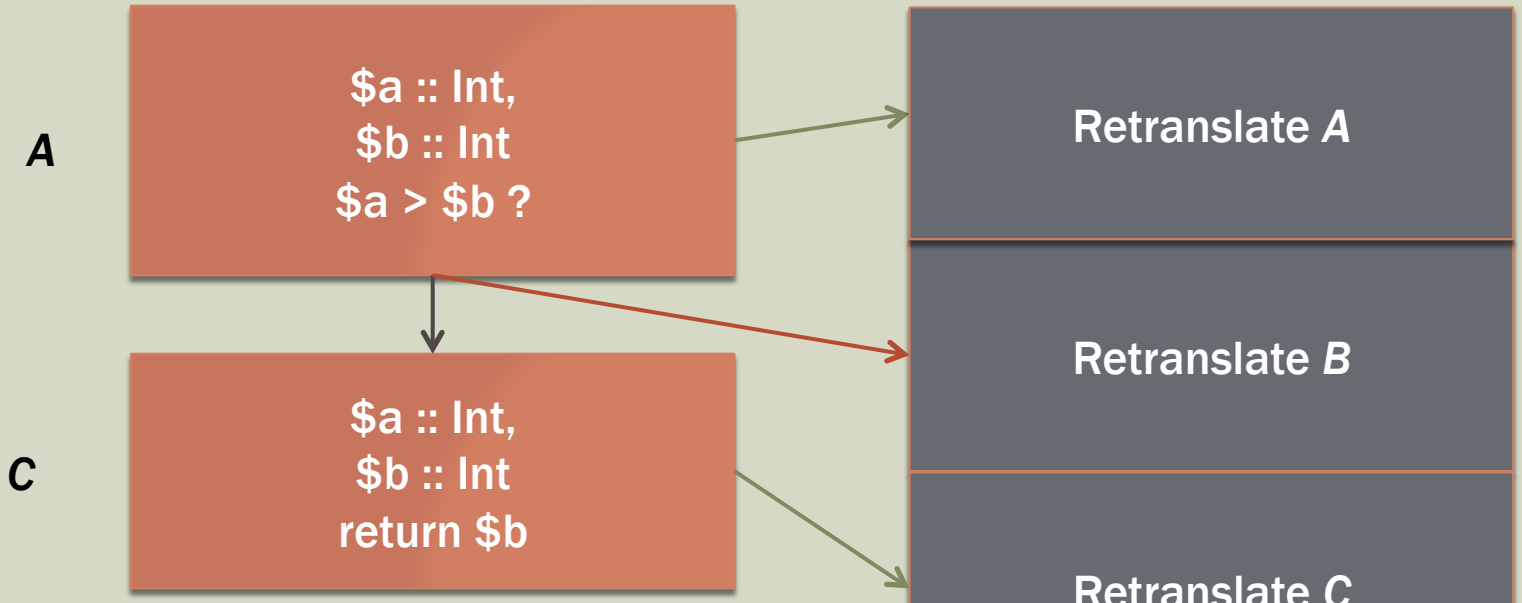


THANKS

- https://github.com/facebook/hiphop-php/
- Questions?

BACKUP

LOGICAL VIEW OF CODE CACHE

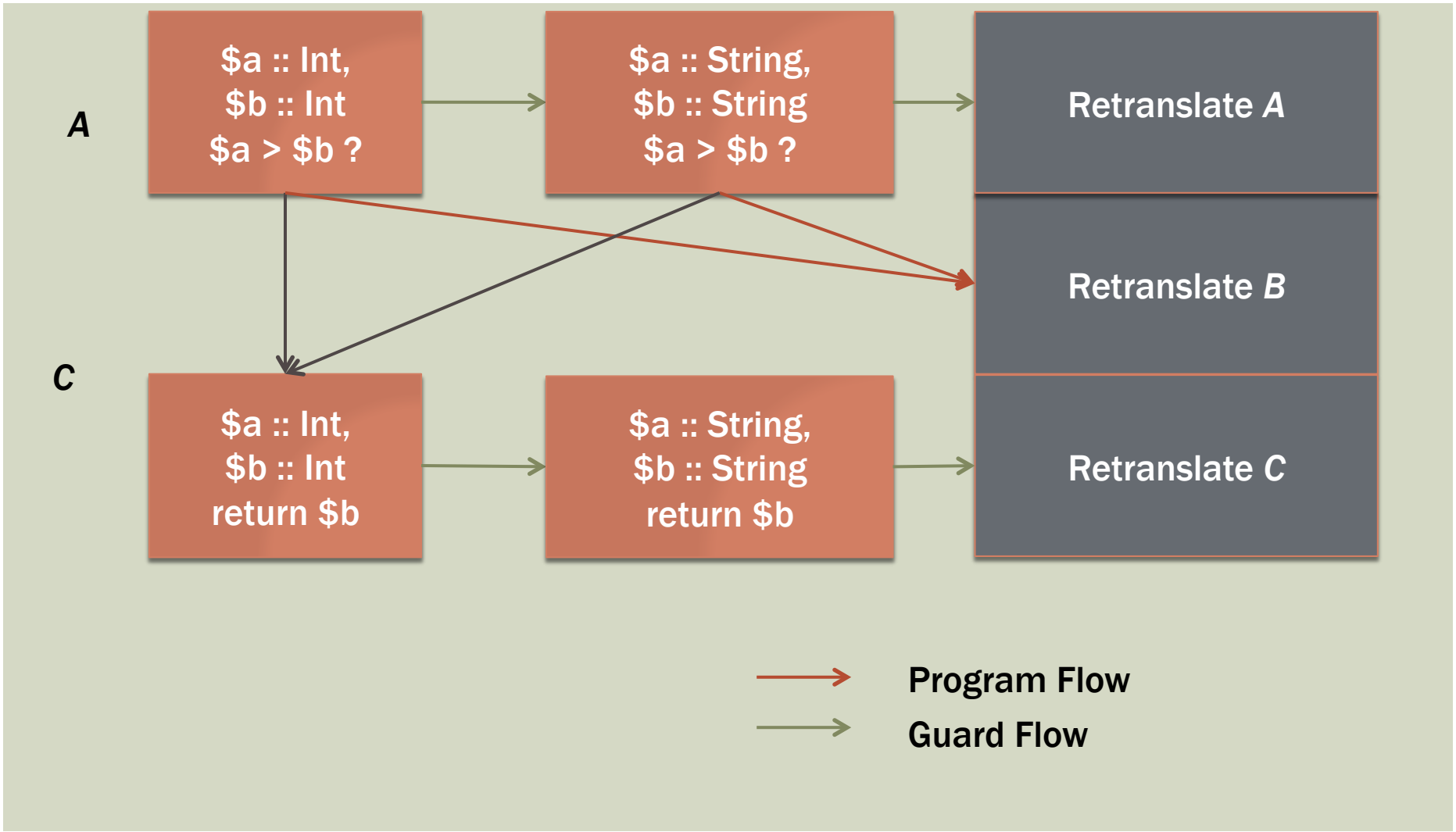



```
A: PushL 1
   PushL 0
   Gt
   JmpZ 1f
B: PushL 0
   Jmp 7 2f
C: 1: PushL 1
2: RetC
```

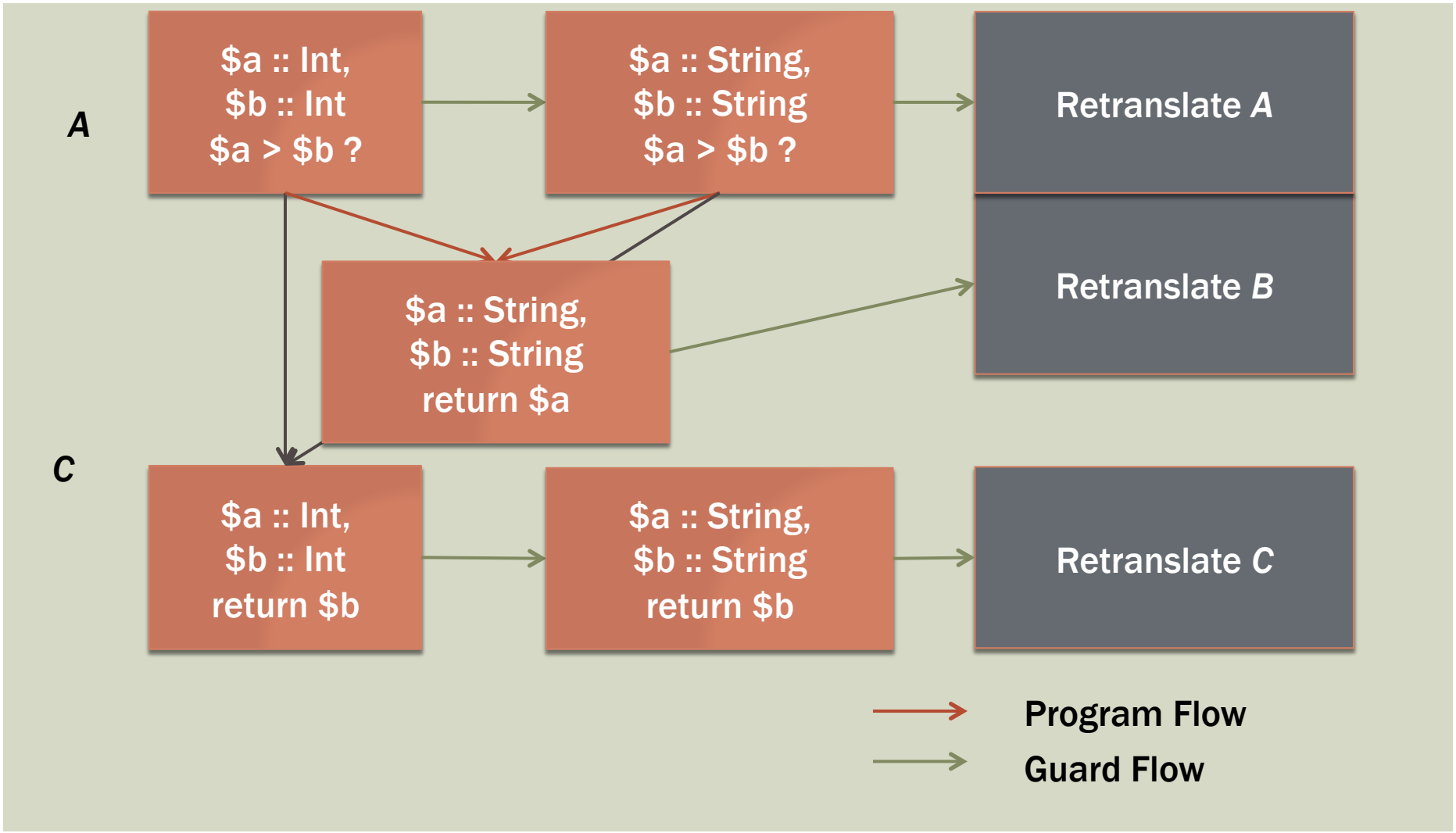
Program Flow

Guard Flow

CALL MYMAX("A", "Z")



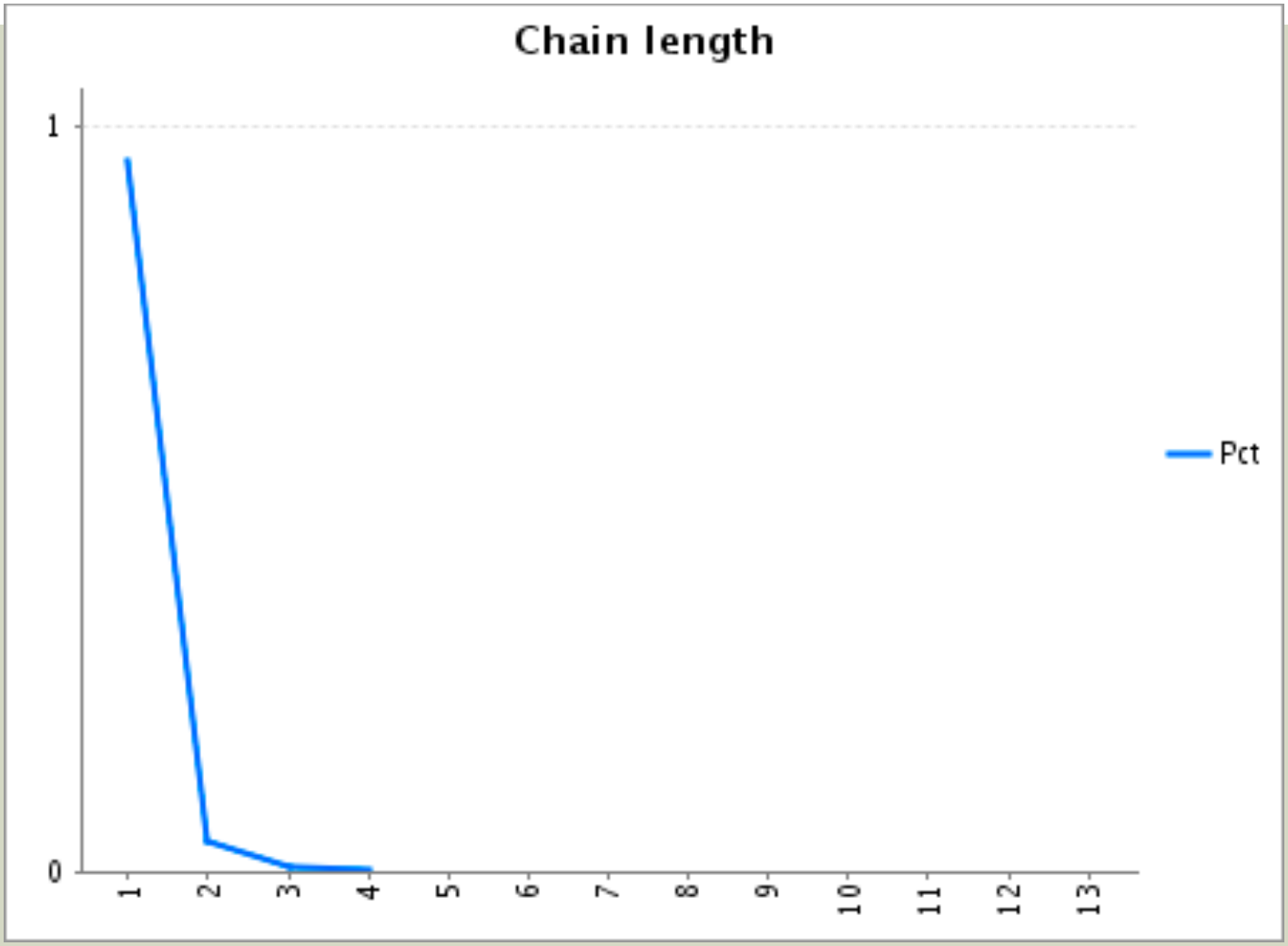
CALL MYMAX("Z", "A")



RISK: CODE EXPLOSION

- *N* inputs, each takes on *t* types
  - will yield t^N separate translations!
- Solution: truncate tracelet chain at 12 items
- Fall back to interpreter.
- Applies to 0.0066% of chains

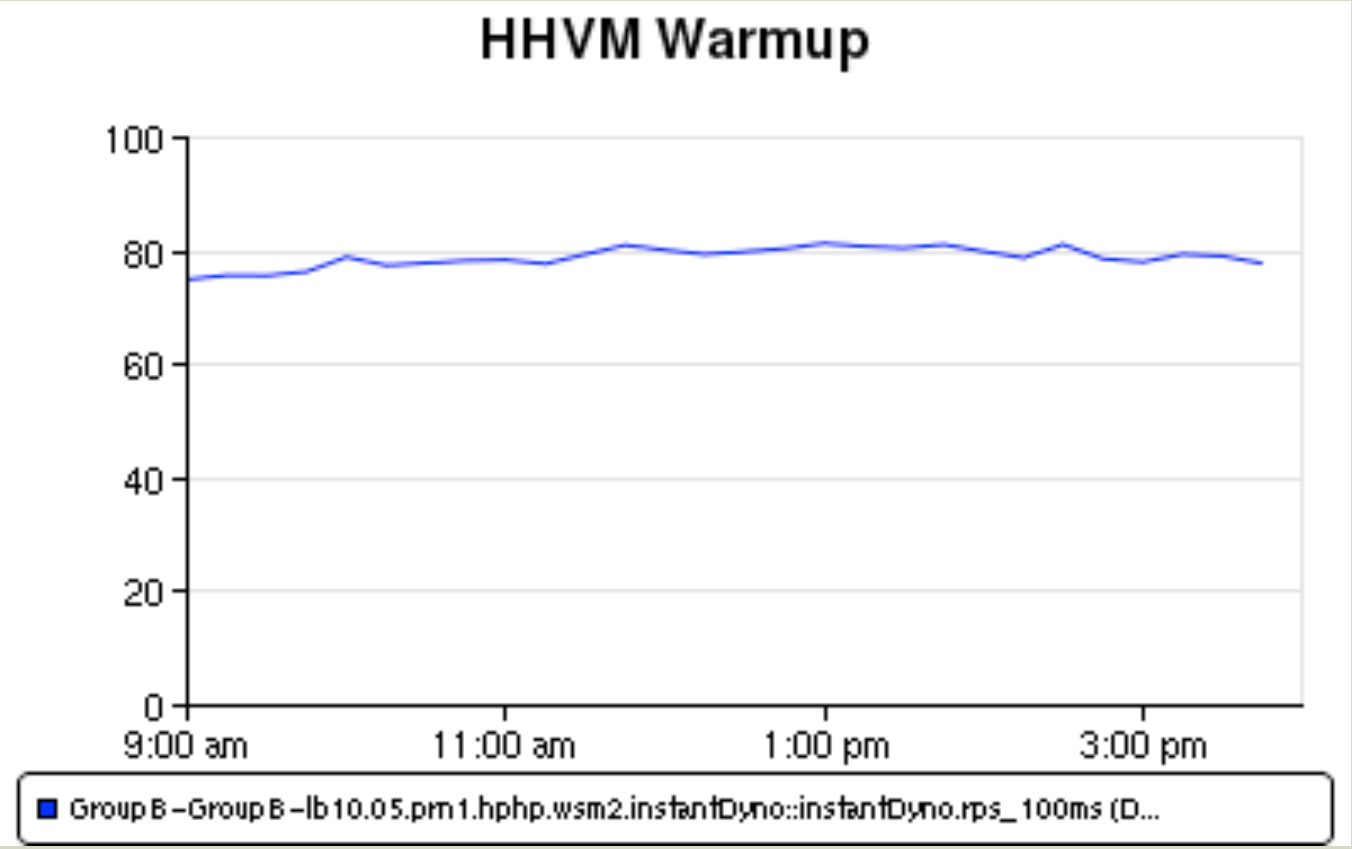
PROD: TRACELET CHAIN LENGTH



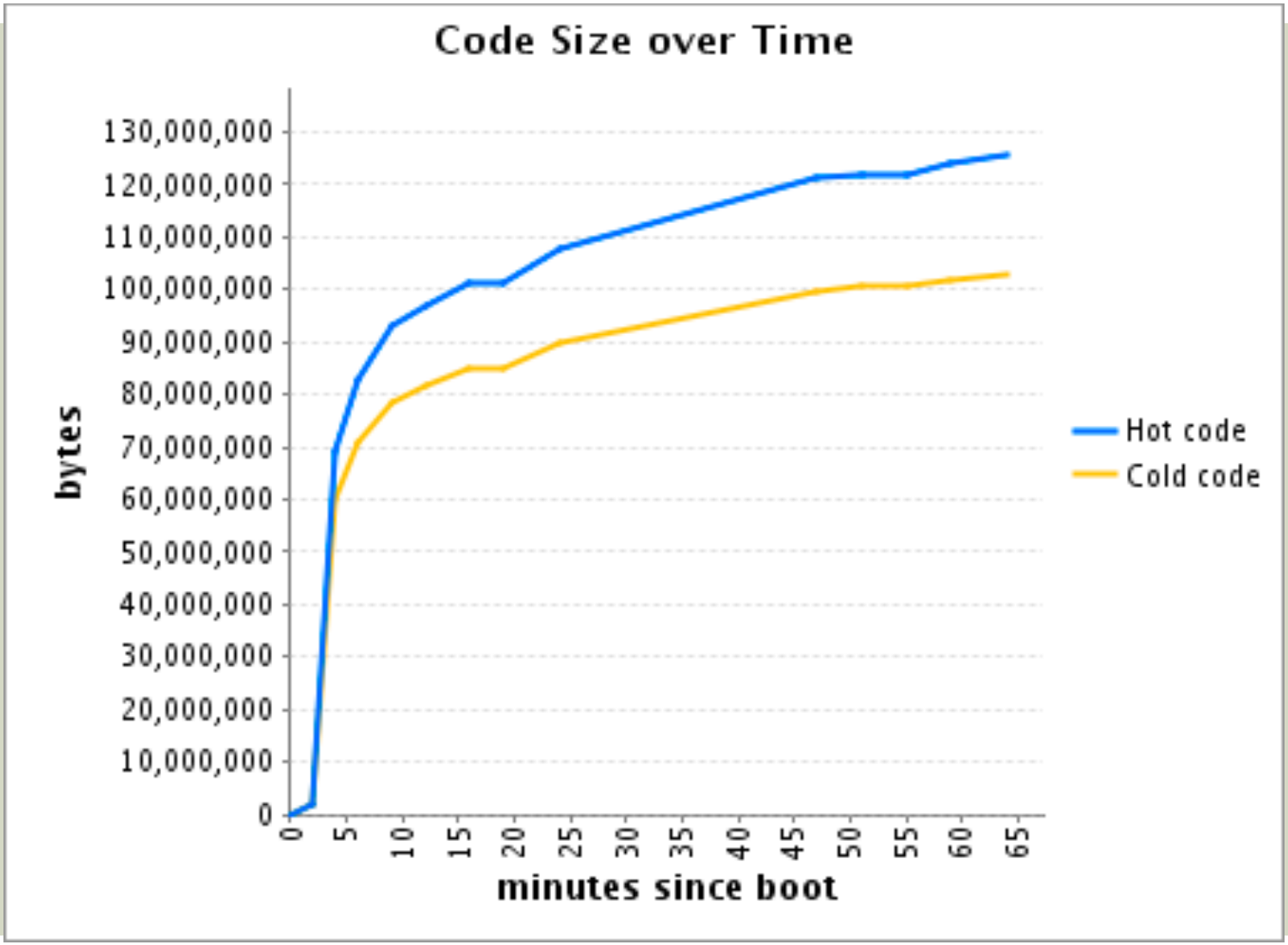
RISK: WARMUP

- Possible weak point of JIT vs. AoT: warmup latency
- We start with an empty code cache
- Goal: reach steady state *quickly*

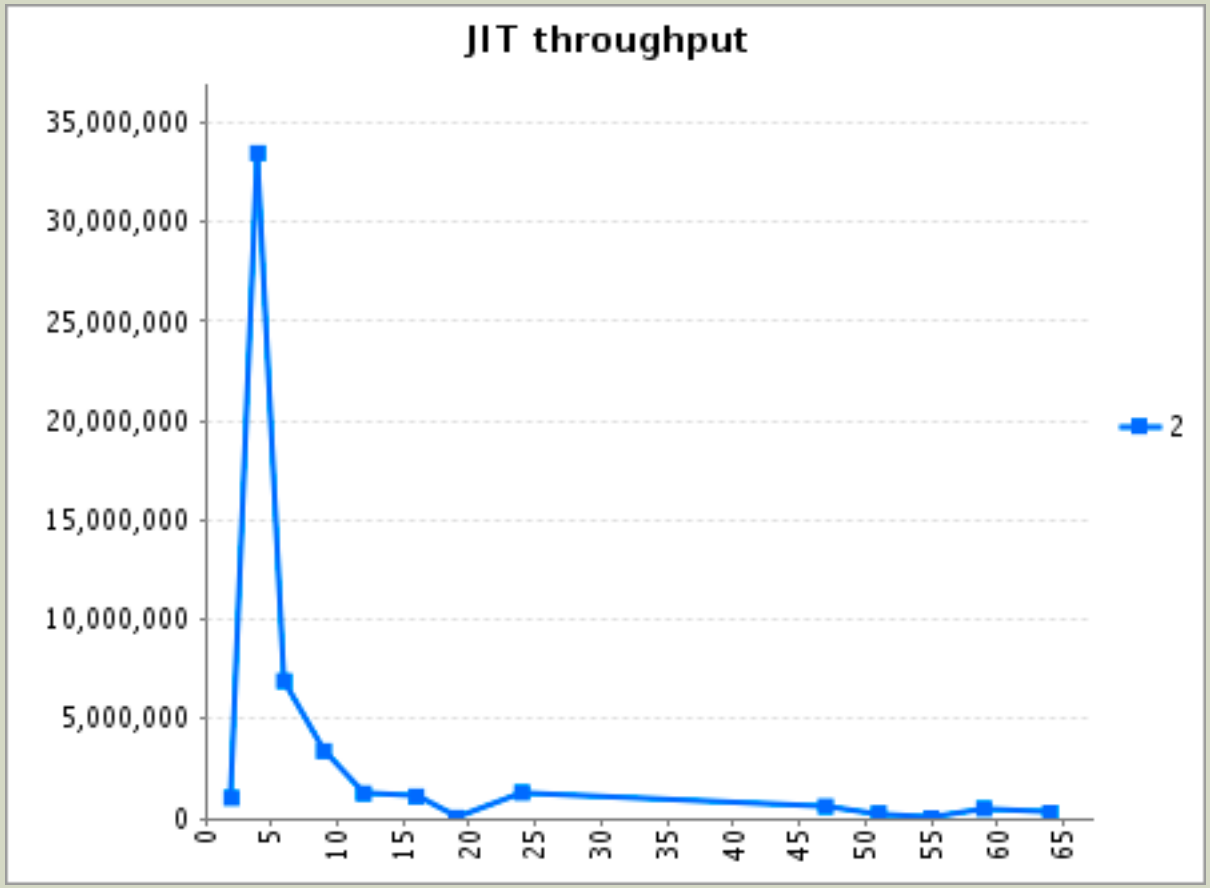
WARMUP: PRODUCTION REQUESTS/SECOND



CODE SIZE OVER TIME



JIT THROUGHPUT / TIME



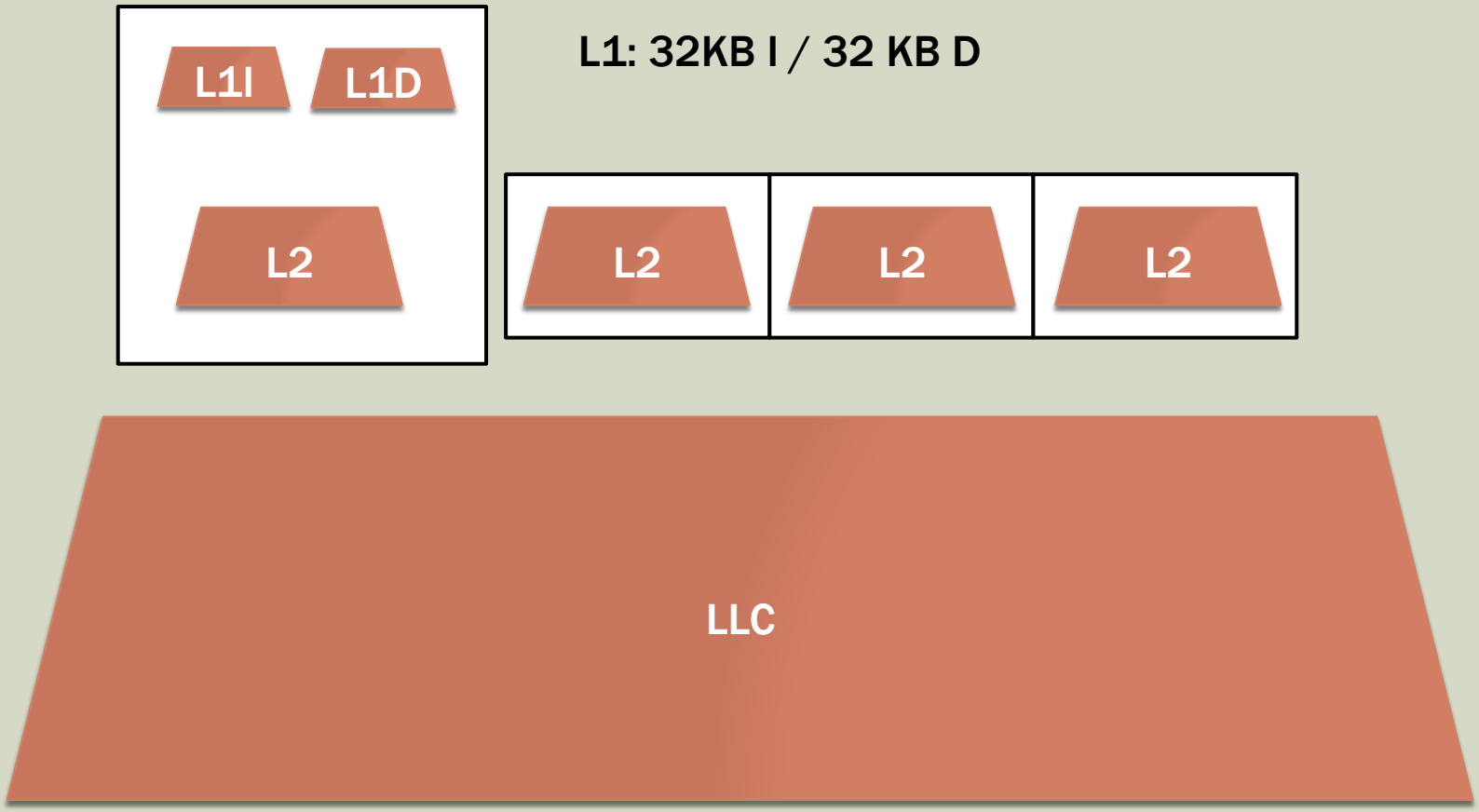
PERF TOOL

- When investigating cache effects, you're blind without hardware performance counters
- Use the Linux kernel *perf* tool
- Whole-system sampling for hardware performance counters.
- When a sample fires, records the instruction, and optionally the stack trace where the event occured.

PERF OUTPUT

```
Events: 782K L1-dcache-load-misses
+  13.12%            hhvm  perf-31695.map                [.] 0x0000000006c9dc61
+   4.65%            hhvm  hhvm                          [.] void HPHP::VM::Unit::mergeImpl<false>(voi
+   3.09%            hhvm  libc-2.13.so                  [.] __memcpy_ssse3
-   2.07%            hhvm  hhvm                          [.] HPHP::HphpArray::find(HPHP::StringData co
   - HPHP::HphpArray::find(HPHP::StringData const*, int) const
      + 31.09% HPHP::HphpArray::nvGet(HPHP::StringData const*) const
      + 17.21% HPHP::HphpArray::get(HPHP::Variant const&, bool) const
      + 15.87% _ZN4HPHP2VML15array_getm_implEPNS_9ArrayDataEPNS_10StringDataEiPNS_10TypedValueE
      + 12.68% HPHP::VM::array_issetm_s0(void const*, HPHP::StringData*)
      + 12.32% HPHP::HphpArray::exists(HPHP::String const&) const
      + 2.86% HPHP::VM::array_issetm_s0_fast(void const*, HPHP::StringData*)
      + 2.60% HPHP::HphpArray::exists(HPHP::Variant const&) const
      + 1.30% HPHP::VM::array_issetm_s(void const*, HPHP::StringData*)
      + 0.80% HPHP::HphpArray::get(HPHP::String const&, bool) const
+   1.51%            hhvm  hhvm                          [.] HPHP::tvDecRefHelper(HPHP::DataType, unsi
+   1.50%            hhvm  hhvm                          [.] HPHP::HphpArray::~HphpArray()
+   1.49%            hhvm  hhvm                          [.] HPHP::VM::Transl::TargetCache::Cache<HPHP
+   1.45%            hhvm  hhvm                          [.] free
+   1.34%            hhvm  hhvm                          [.] HPHP::VM::Transl::newInstanceHelperCached
+   1.25%            hhvm  hhvm                          [.] HPHP::SmartAllocatorImpl::alloc(unsigned
+   1.19%            hhvm  libc-2.13.so                  [.] __memset_sse2
+   1.18%            hhvm  hhvm                          [.] longest_match
+   1.11%            hhvm  hhvm                          [.] HPHP::VM::Class::classof(HPHP::VM::PreCla
+   1.09%            hhvm  hhvm                          [.] malloc
Press '?' for help on key bindings
```

INCLUSIVE CACHES



SYSTEM SIZE

- Source tree contains 262864 semi-colons
- PHP runtime (including 169 extensions): 132092
- Excluding extensions: 72729
- Jit: 17582

ICACHE AND MEMCPY

- When investigating our high rate of instruction cache misses, perf led to an unusual culprit: memcpy
- Shouldn't memcpy be in cache all the time?

```
Events: 280K L1-icache-load-misses
    32.51%             hhvm  perf-23494.map                [.] 0x0000000006ed0308
+    1.81%             hhvm  libc-2.13.so                  [.] __memcpy_ssse3
+    1.60%             hhvm  hhvm                          [.] HPHP::VM::Transl::TargetCache::Cache<HPHP::VM
+    1.22%             hhvm  hhvm                          [.] HPHP::HphpArray::find(HPHP::StringData const*
+    1.10%             hhvm  hhvm                          [.] HPHP::SmartAllocatorImpl::alloc(unsigned long
+    0.85%             hhvm  hhvm                          [.] malloc
+    0.71%             hhvm  hhvm                          [.] _ZN4HPHP2VM6TranslL19VerifyParamTypeSlowEPKNS
+    0.69%             hhvm  hhvm                          [.] HPHP::VM::Transl::newInstanceHelperCached(HPH
+    0.67%             hhvm  hhvm                          [.] _ZN4HPHP2VML4ElemILb1ELNS_8DataTypeEn1EEEPNS_
+    0.66%             hhvm  hhvm                          [.] free
+    0.58%             hhvm  hhvm                          [.] HPHP::VM::Class::getSProp(HPHP::VM::Class*, H
+    0.58%             hhvm  hhvm                          [.] HPHP::VM::FixedStringMap<unsigned int, true>:
+    0.57%             hhvm  hhvm                          [.] HPHP::VM::iter_value_cell_local_array(HPHP::V
+    0.56%             hhvm  hhvm                          [.] HPHP::tvDecRefHelper(HPHP::DataType, unsigned
+    0.54%             hhvm  libc-2.13.so                  [.] __memcmp_sse4_1
+    0.52%             hhvm  libc-2.13.so                  [.] __memset_sse2
+    0.51%             hhvm  hhvm                          [.] HPHP::HphpArray::update(HPHP::StringData*, HP
+    0.50%             hhvm  hhvm                          [.] HPHP::Variant::toKey() const
```