



IBM Tokyo Research Laboratory

A Study of Memory Management for Web-based Applications on Multicore Processors

Hiroshi Inoue, Hideaki Komatsu
and Toshio Nakatani
IBM Tokyo Research Laboratory

Goal

Our Goal

To develop an efficient memory management technique for Web-based applications to improve their performance on multicore processors.

Our main contributions include:

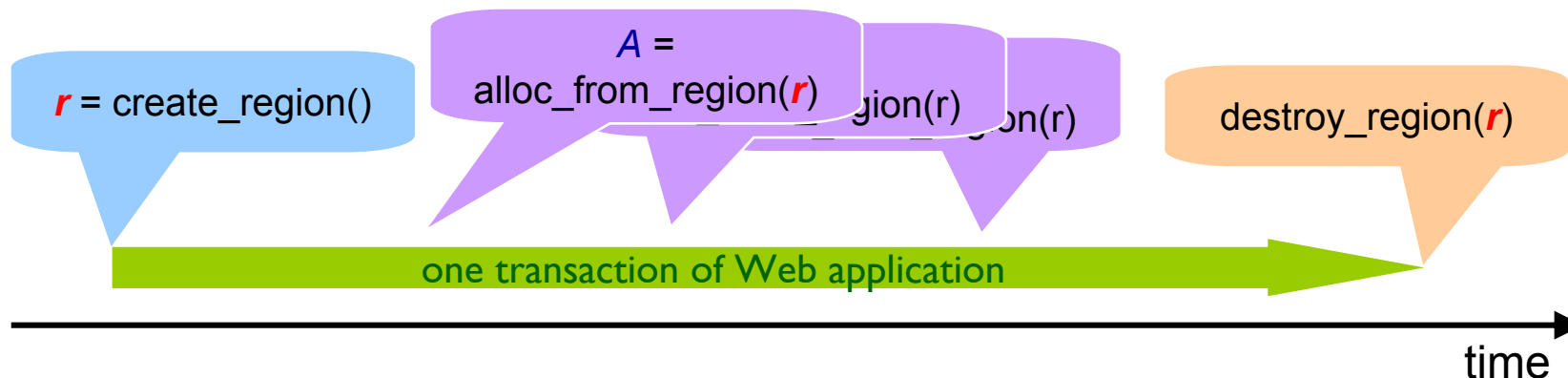
1. Showing that a good technique for a single core is not necessarily a good one on a multicore processor.
2. Proposing our new approach for multicore environments

Characteristics of Web-based applications

- In Web-based applications, most of the allocated memory blocks are *transaction scoped*, and live only during one transaction
- ➡ We exploit their characteristics to reduce the costs of memory management

An existing approach

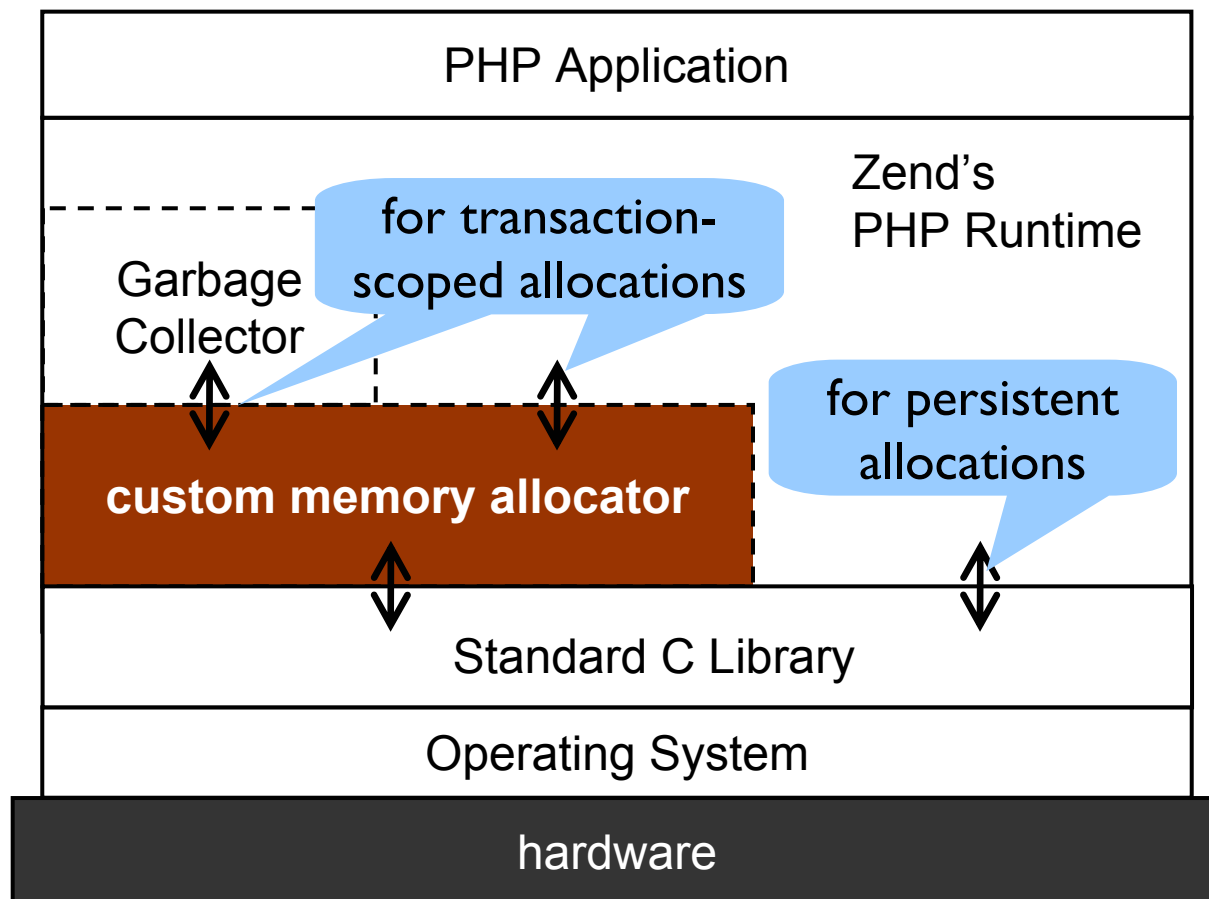
- Region-based memory management
 - reduces the cost of memory management by discarding all blocks in a region at once (instead of freeing individual blocks)
 - is widely used (e.g. Apache pool allocator)



➡ The region-based memory management looks ideal for managing transaction-scoped memory blocks efficiently

Experimental setup with the PHP runtime

Software stack of the Zend's PHP runtime



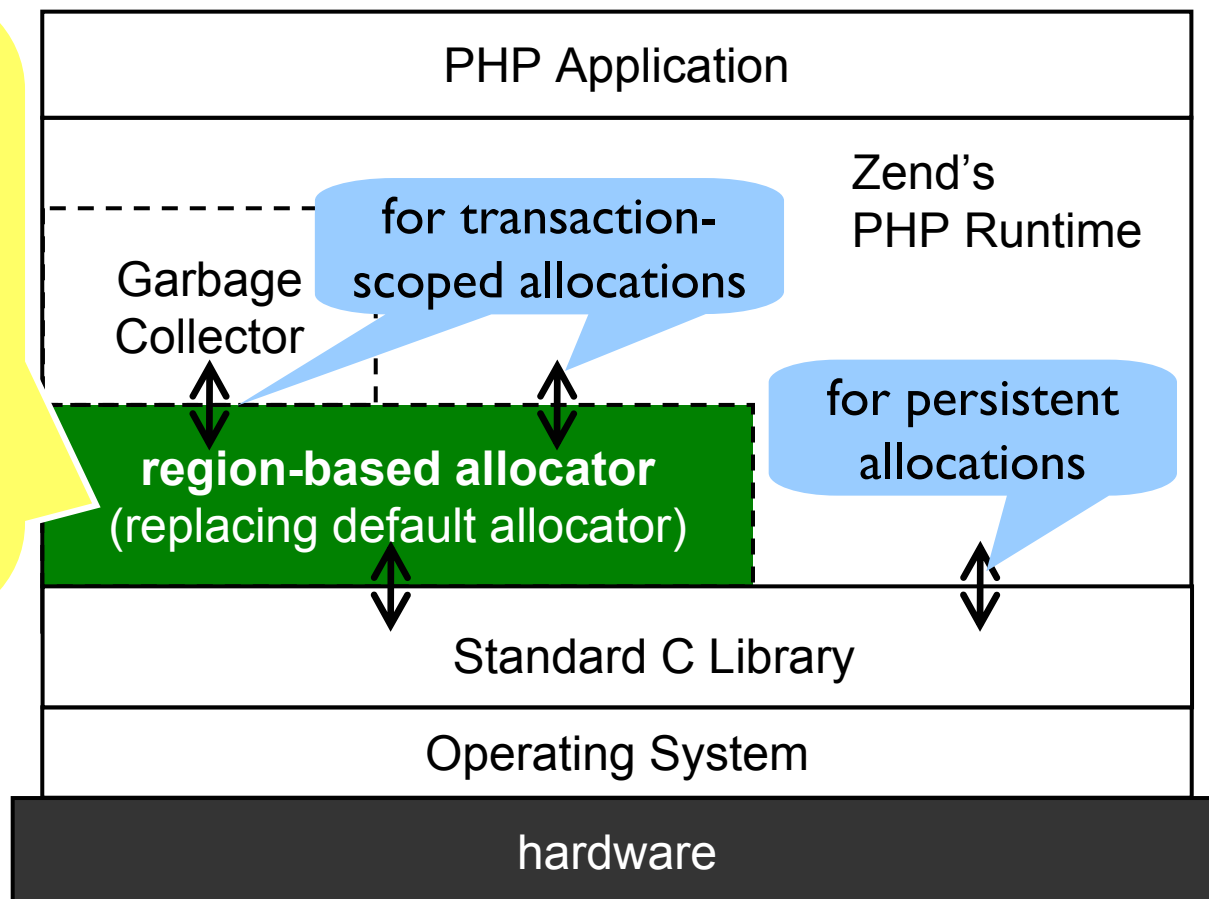
Experimental setup with the PHP runtime

Software stack of the Zend's PHP runtime

We replaced only the custom memory allocator of the PHP runtime

We did not modify

- ✓ PHP applications
- ✓ garbage collector
- ✓ memory allocator in libc



Experimental setup with the PHP runtime

System used

- Blade Center HS21
- 2x quad-core Xeon (Clovertown) 1.86 GHz
- 8 GB system memory
- Red Hat Enterprise Linux 5

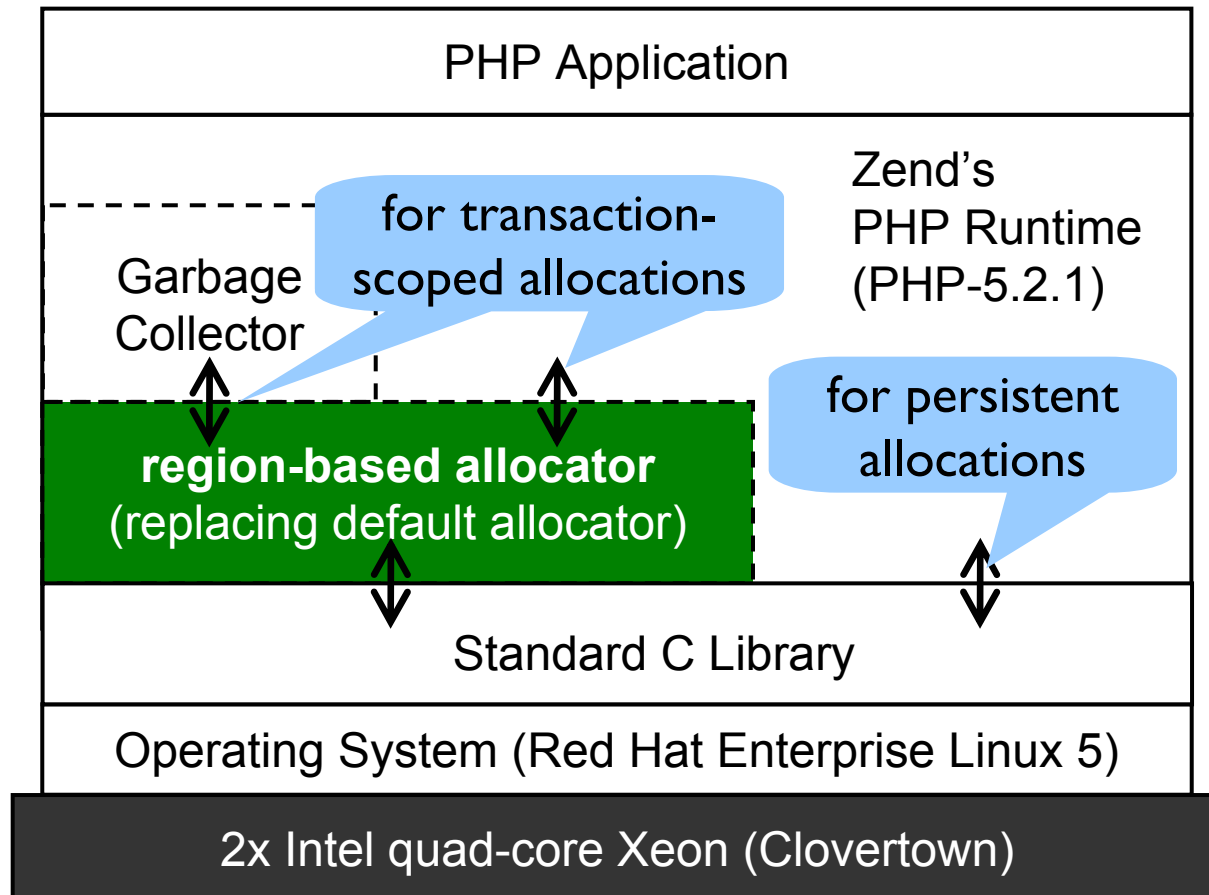
Software

- PHP-5.2.1 (w/ APC-3.0.14)
- lighttpd-1.4.13
- mysql-4.1.20

Benchmark

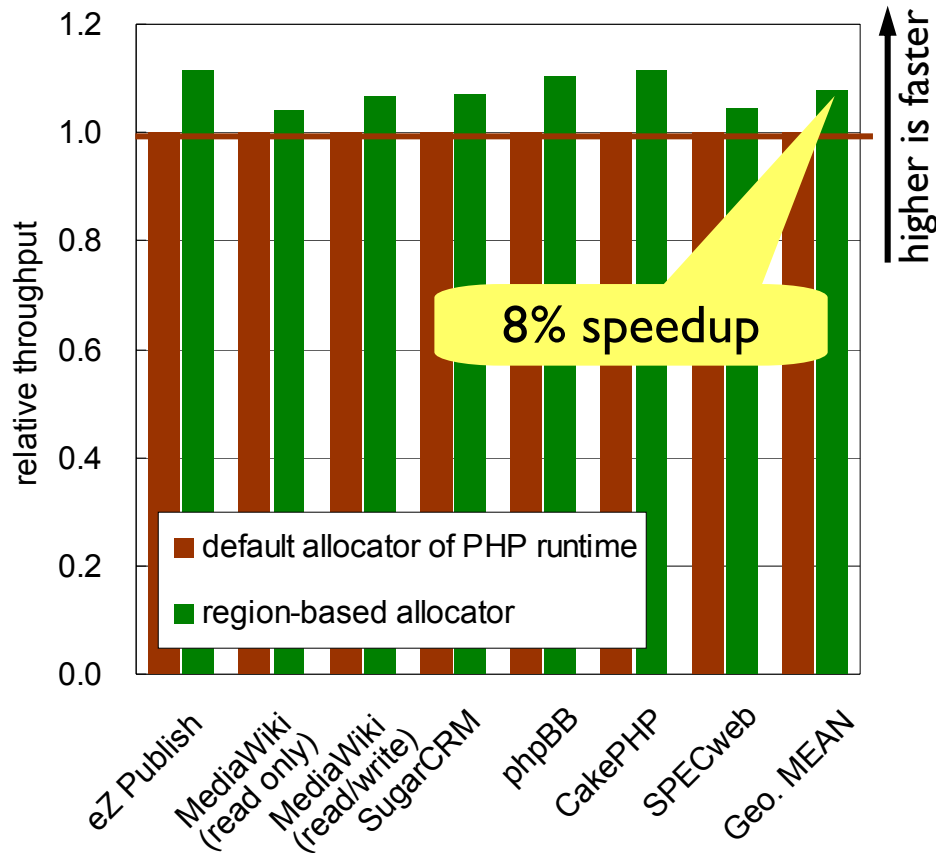
- six server applications (ezPublish, MediaWiki, SugarCRM, phpBB, CakePHP, SPECweb2005)

Software stack of the Zend's PHP runtime



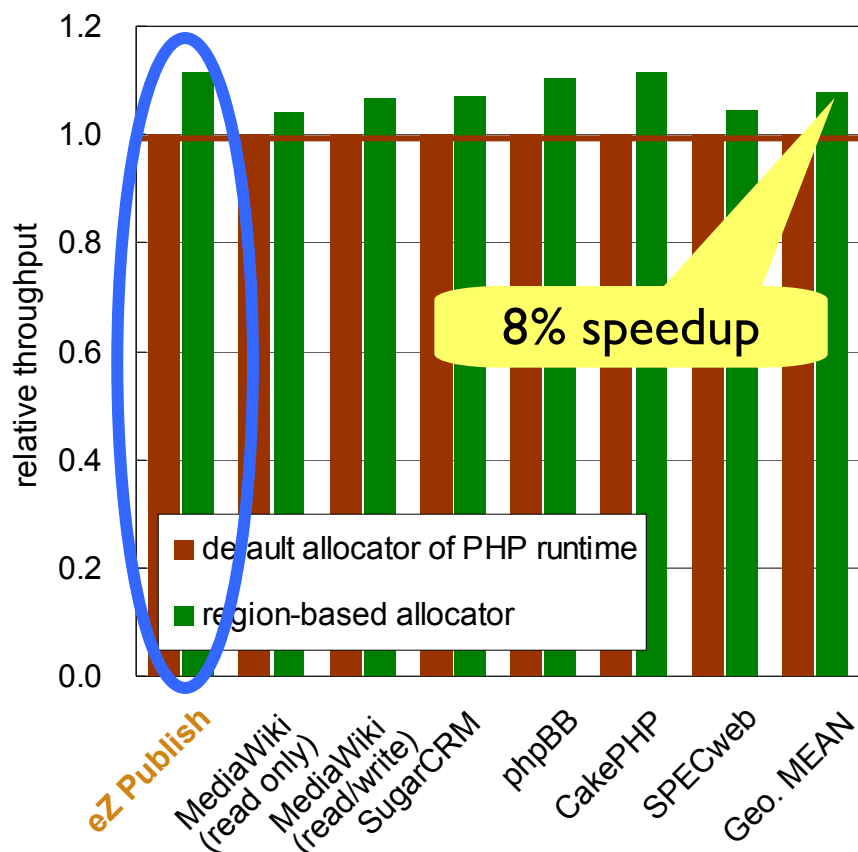
Performance of the region-based allocator: Throughput

on one core

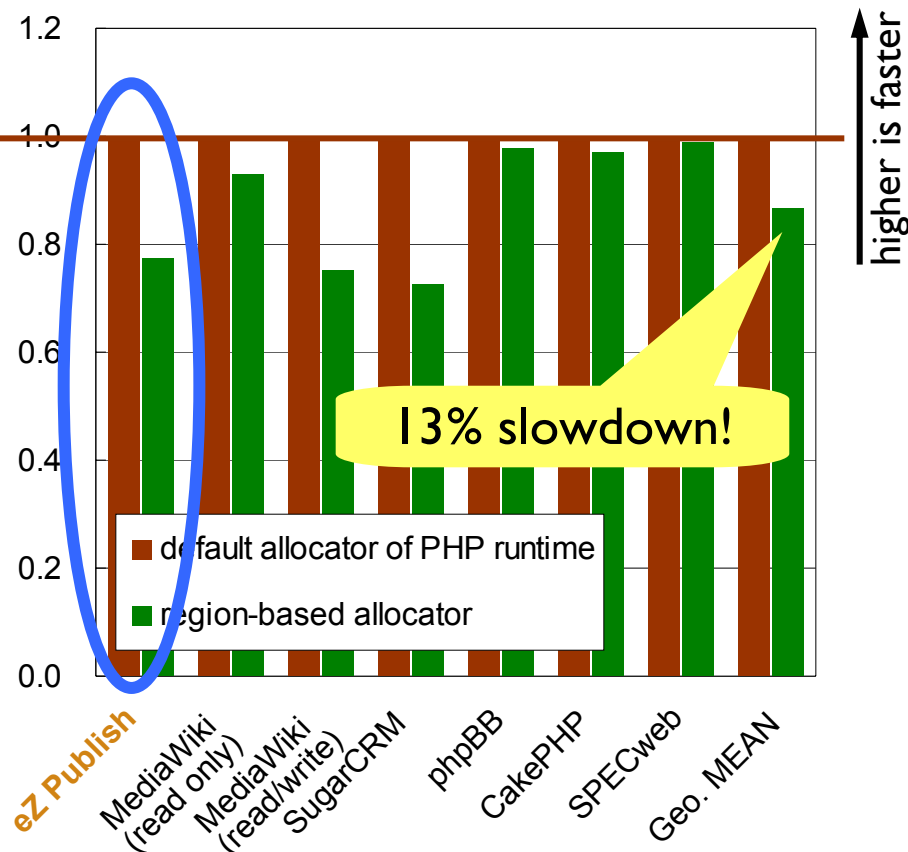


Performance of the region-based allocator: Throughput

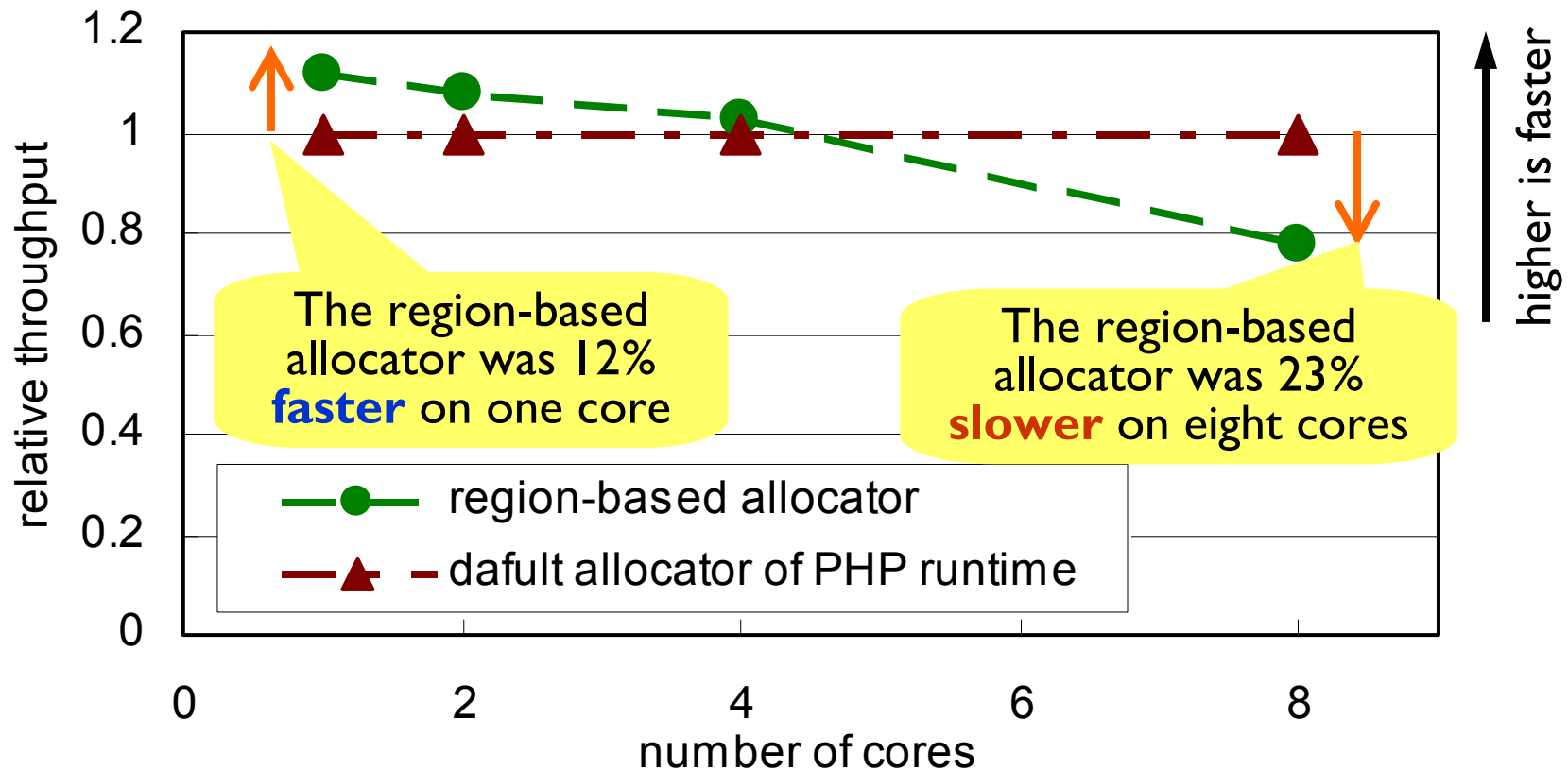
on one core



on eight cores

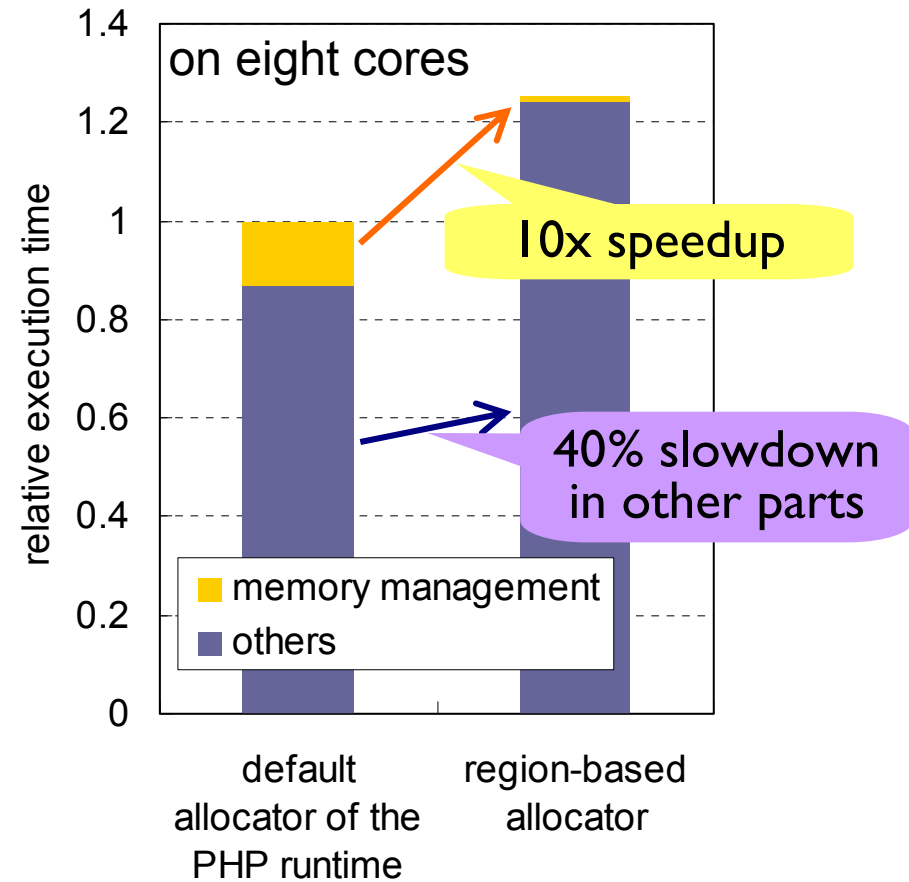
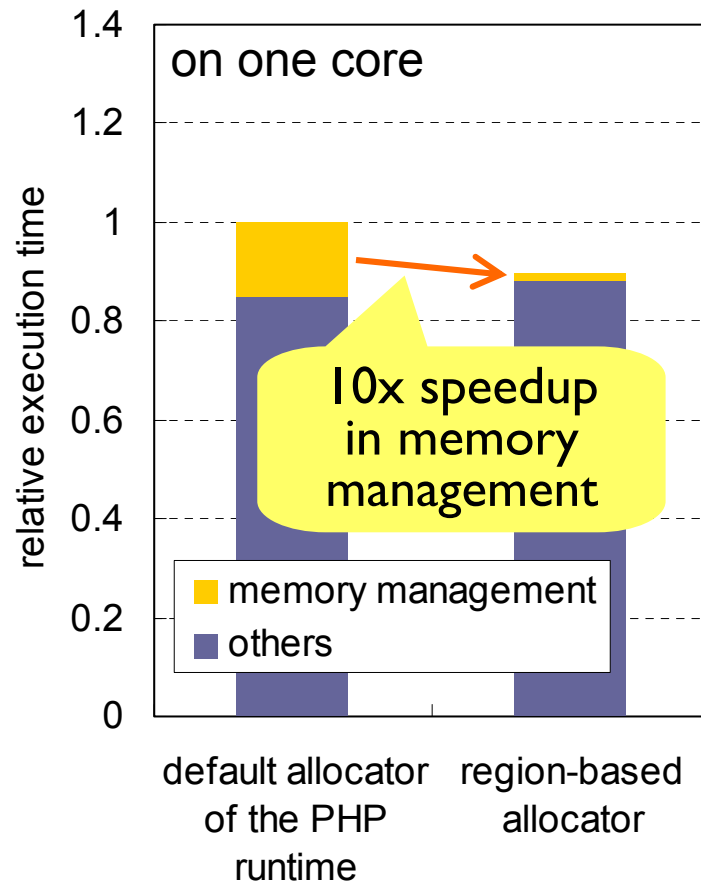


Performance of the region-based allocator: Scalability



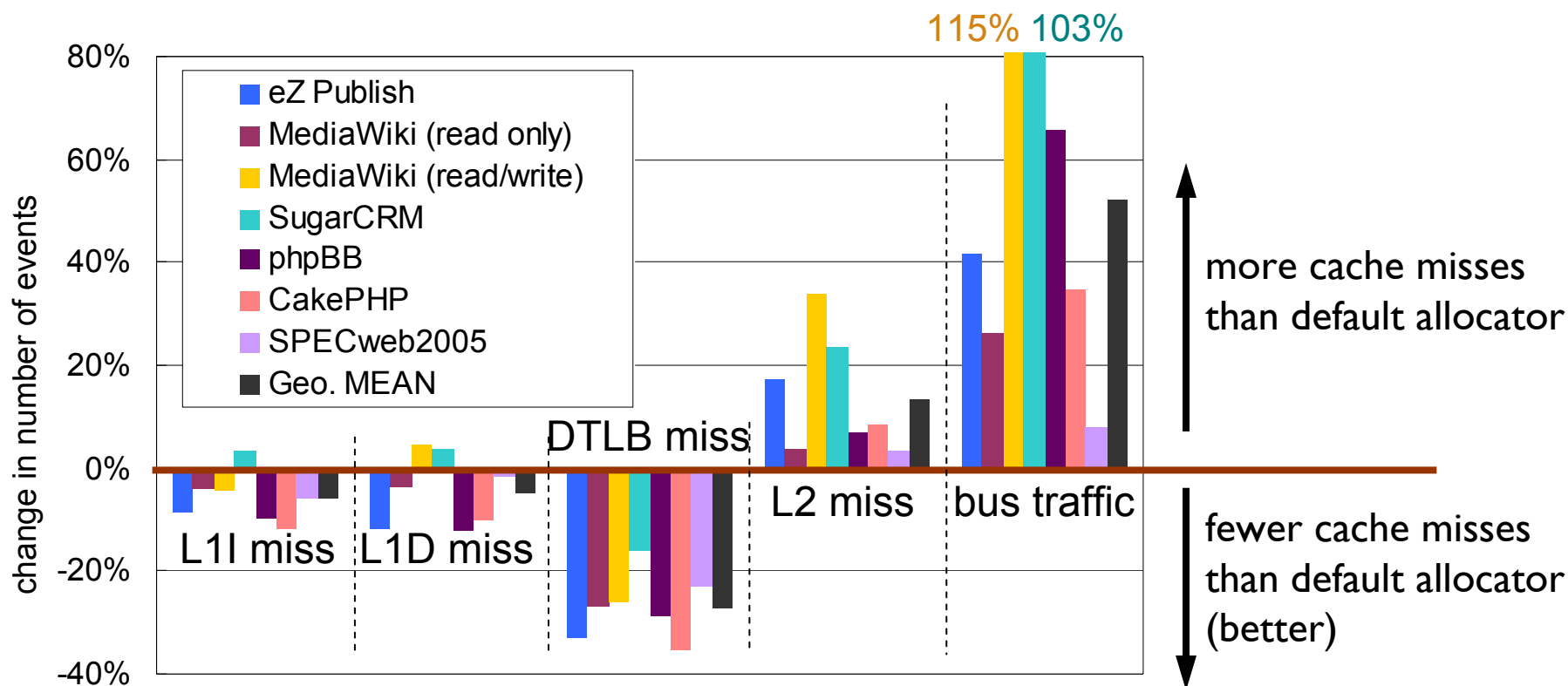
for ezPublish

Performance of the region-based allocator: Execution time breakdown



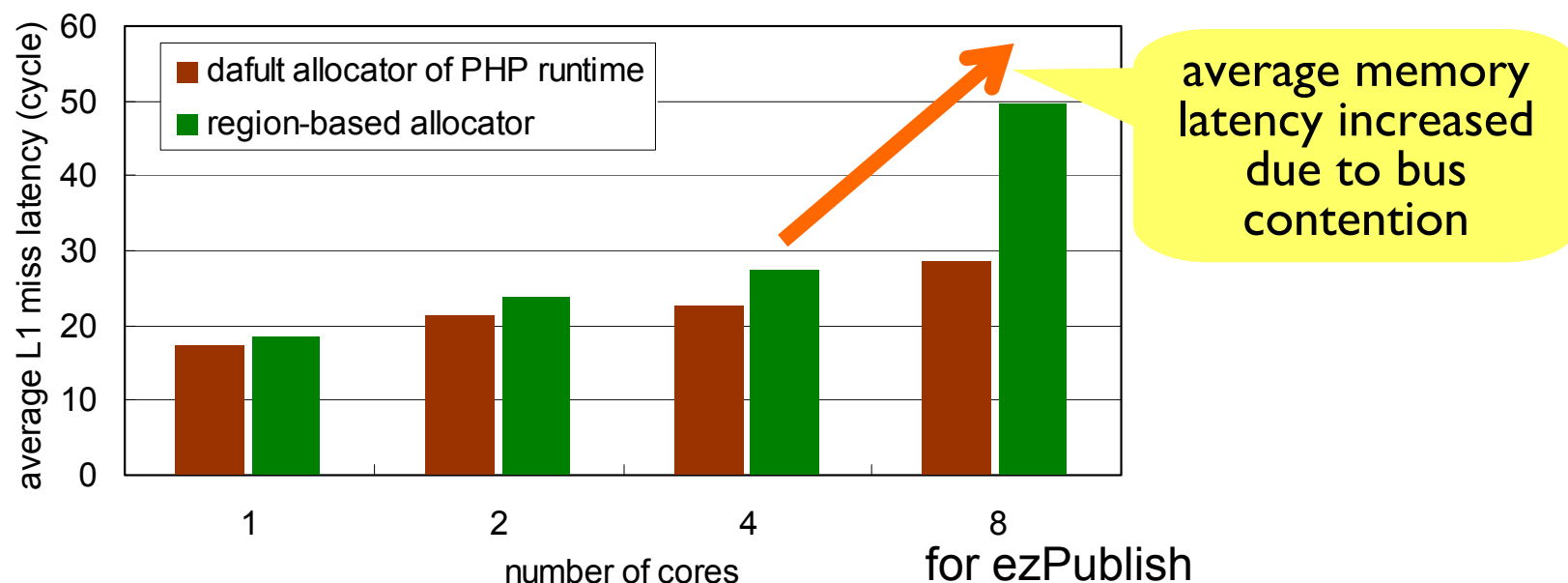
for ezPublish

Performance of the region-based allocator: Cache misses and bus traffic



on eight cores

Performance of the region-based allocator: Memory latency



on one or few cores

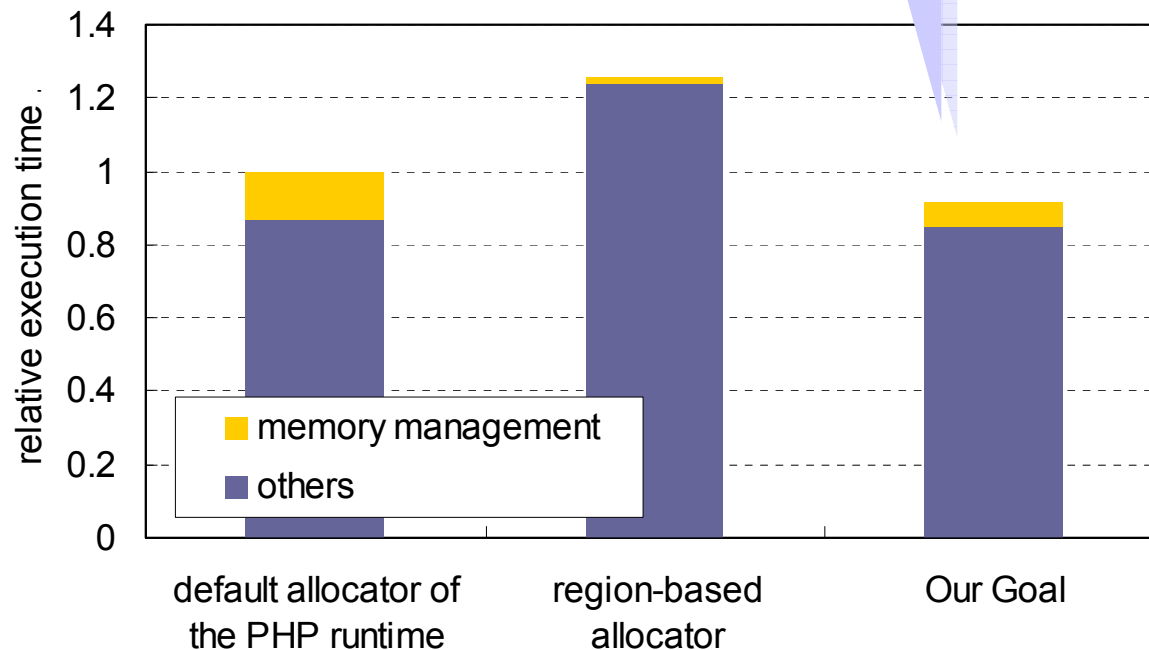
reduced cost of memory management > *increased bus traffic*

on more cores

reduced cost of memory management < *increased bus traffic*

Our Goal

- ✓ to reduce the cost of memory management
- ✓ without slowing down on multicore processors



Revisiting general-purpose memory allocators

- General-purpose memory allocators' tasks
 - malloc(): allocate memory block
 - free(): reclaim memory block and prepare for reuse in future allocations
 - minimize the heap fragmentation (*defragmentation*)

typically consumes large amounts of CPU time

- For example,
 - ✓ coalescing multiple small blocks into large blocks
 - ✓ splitting large blocks into small blocks
 - ✓ sorting unused blocks in the free lists

Our approach: *Defrag-Dodging*







Key observation

the transactions in Web-based applications are short enough to ignore heap fragmentation, and so the

cost of defragmentation > benefits

- Our new approach: *Defrag-Dodging*
 - reduces the memory management cost by avoiding defragmentation activities in malloc and free
 - unlike the region-based memory management, support a free() function to enable fine-grained memory reuse

Comparing three approaches

	general-purpose memory management	our Defrag-Dodging	region-based memory management
malloc() allocate new memory block			
free() reclaim and reuse memory block			-
defragmentation		-	-

Comparing three approaches

	general-purpose memory management	our Defrag-Dodging	region-based memory management
malloc() allocate new memory block	✓	✓	✓
free() reclaim and reuse memory block	✓	✓	-
defragmentation	✓	-	-

reduced memory management cost

- simpler allocator code
- simpler heap structure (e.g. no per-object metadata)

Comparing three approaches

	general-purpose memory management	our Defrag-Dodging	region-based memory management
malloc() allocate new memory block	✓	✓	✓
free() reclaim and reuse memory block	✓	✓	-
defragmentation	✓	-	-

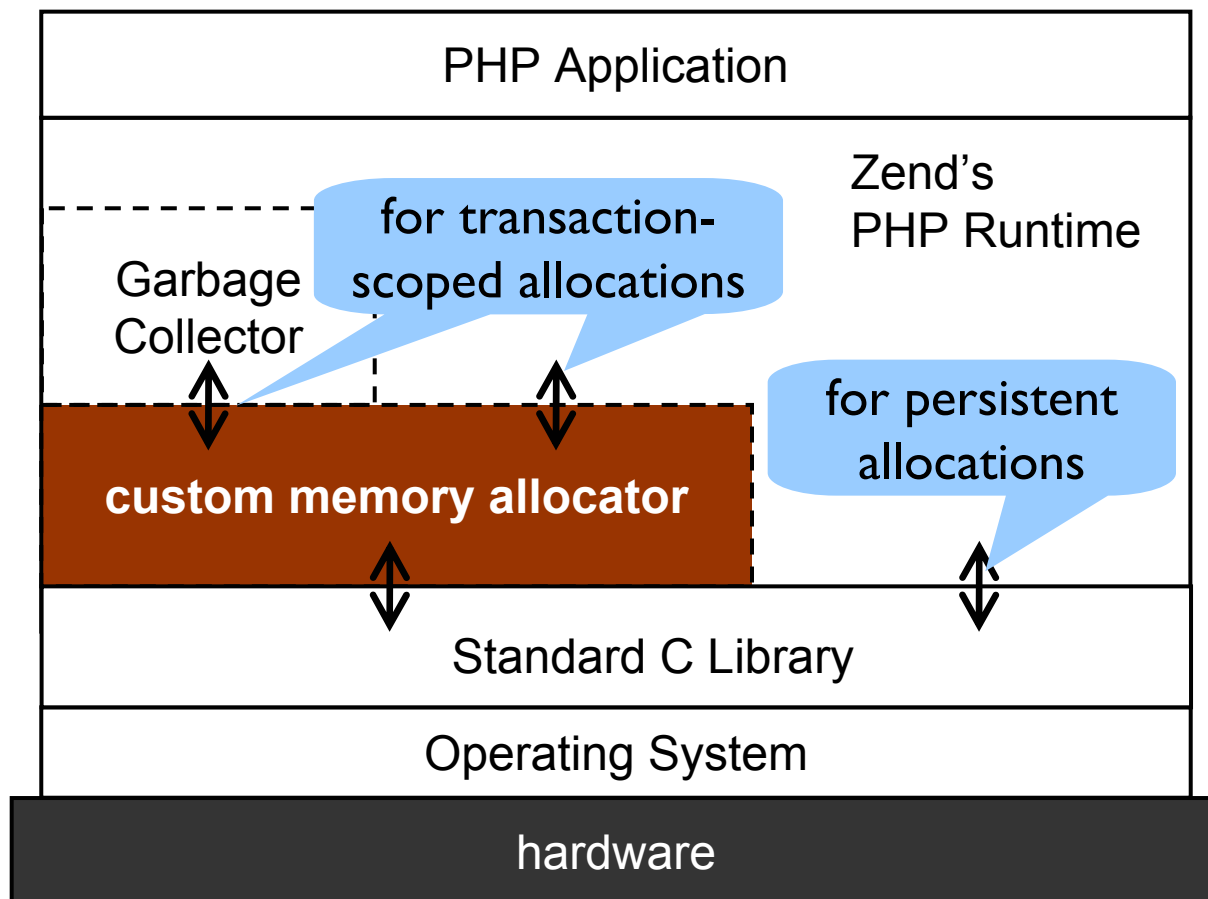
better scalability on multicore processors
by avoiding increases in bus traffic

DDmalloc: Implementation of Defrag-Dodging

- based on a segregated heap allocator
 - maintains free lists for each size of blocks to keep track of freed blocks and reuse them in future allocations
- reduced cost by keeping malloc and free as simple as possible
 - for example, free() function only chains the freed blocks to the corresponding free list (and does nothing else!)
 - the allocator code is less than 500 lines of C code
- clears all metadata to refresh the heap at the end of each transaction
- see the paper for the implementation details

Experimental setup with the PHP runtime

Software stack of the Zend's PHP runtime



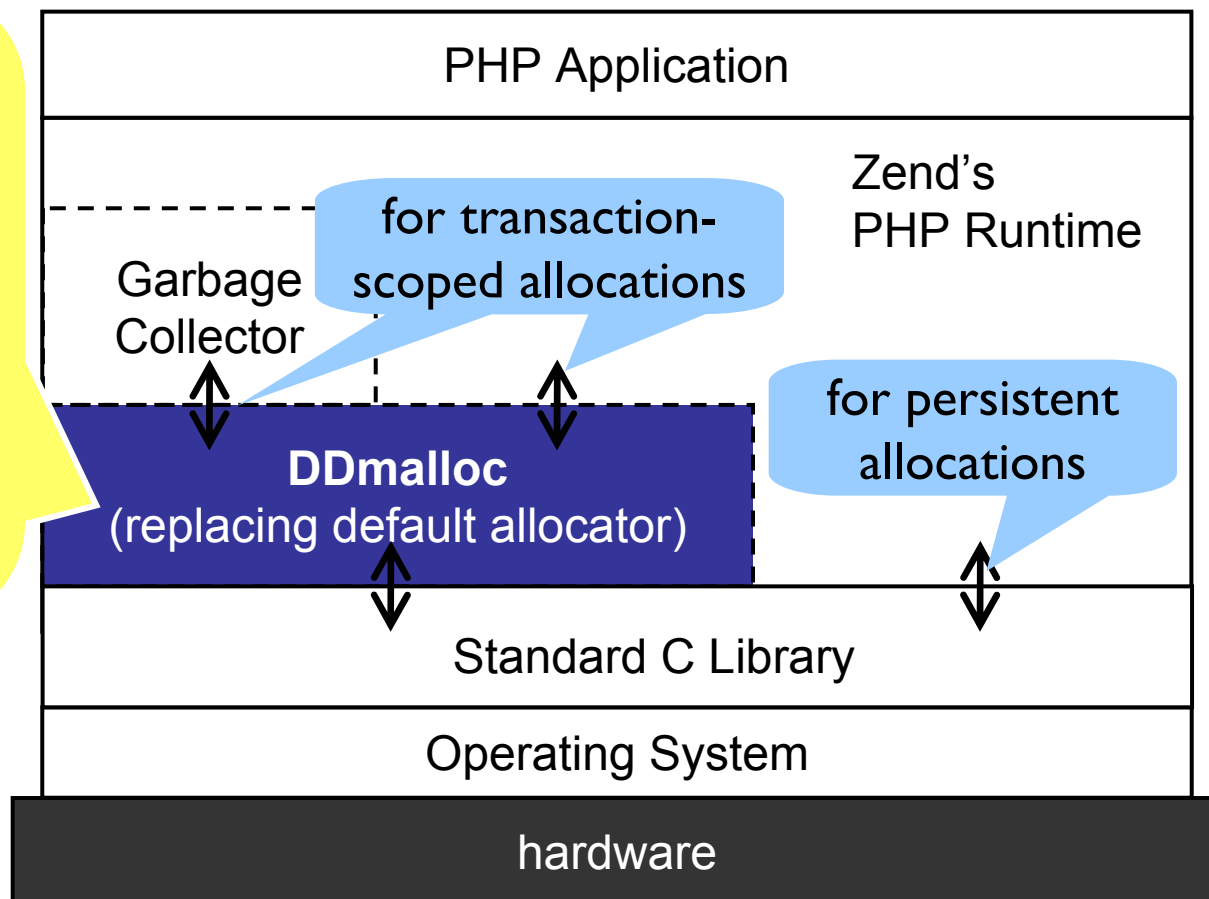
Experimental setup with the PHP runtime

Again, we replaced only the custom memory allocator with DDmalloc

We did not modify

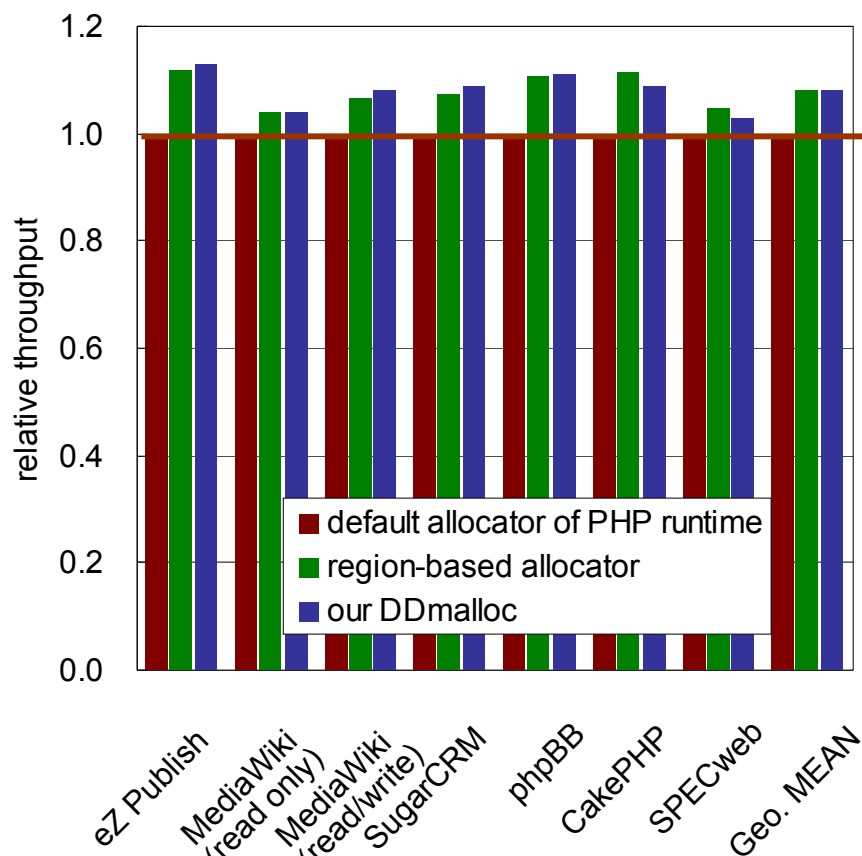
- ✓ PHP applications
- ✓ garbage collector
- ✓ memory allocator in libc

Software stack of the Zend's PHP runtime

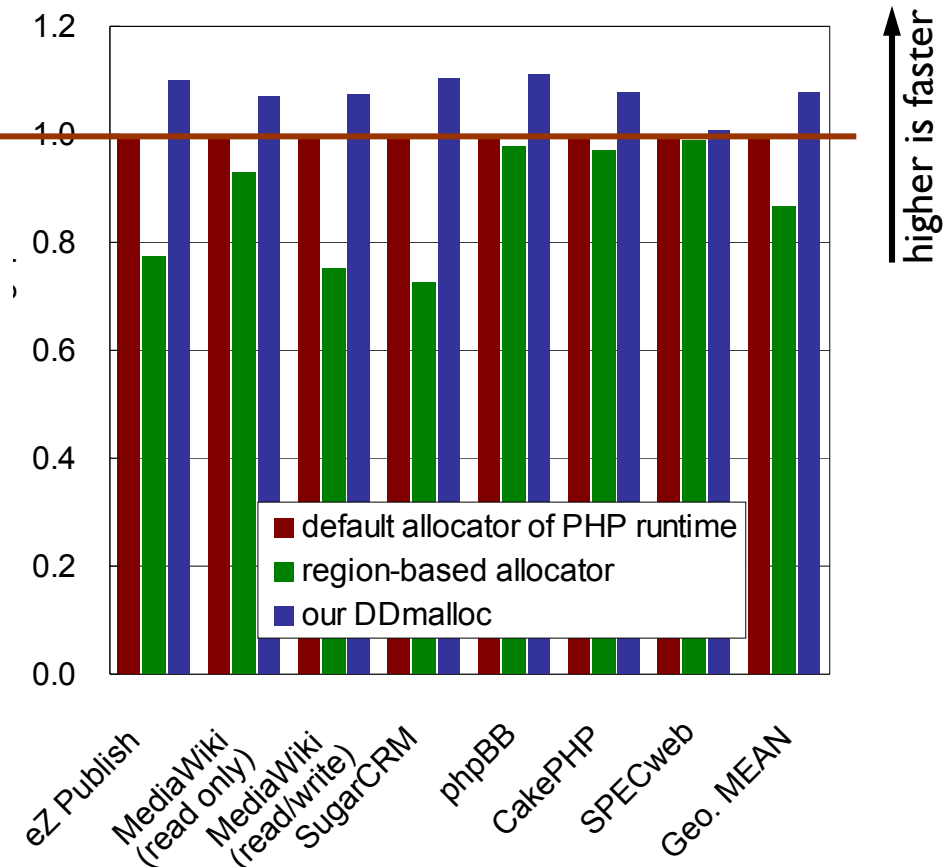


Performance of DDmalloc: Throughput

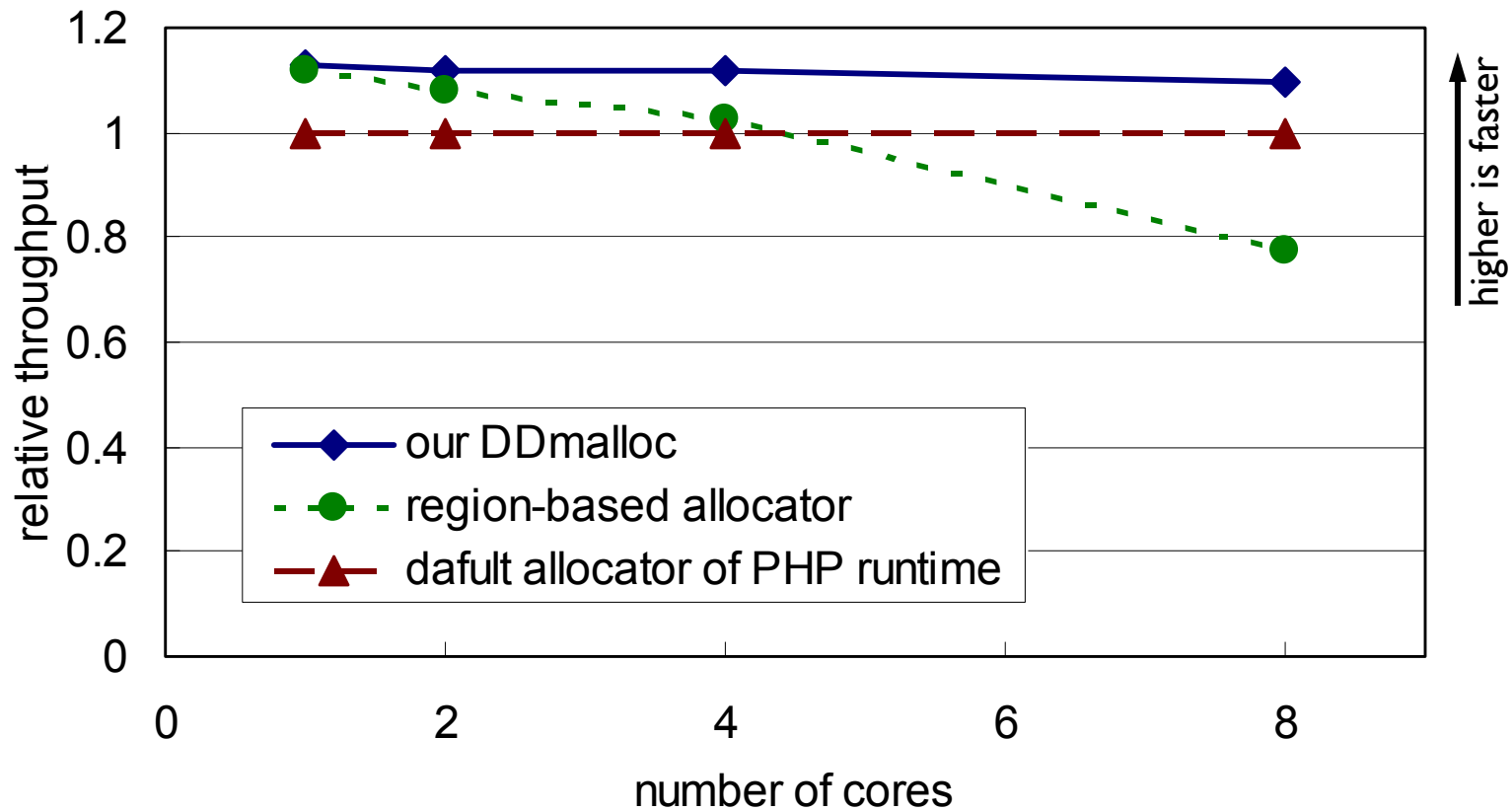
on one core



on eight cores

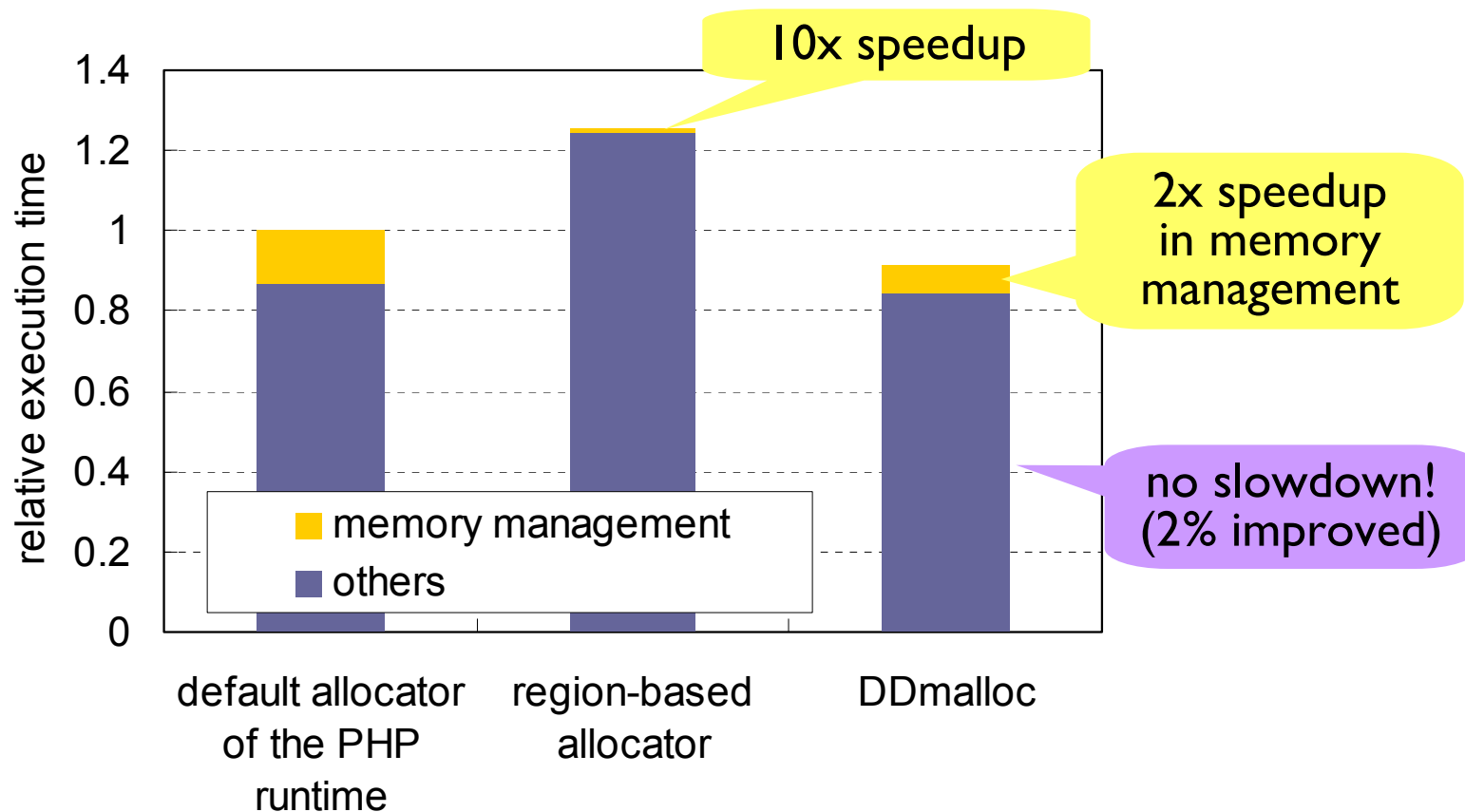


Performance of DDmalloc: Scalability



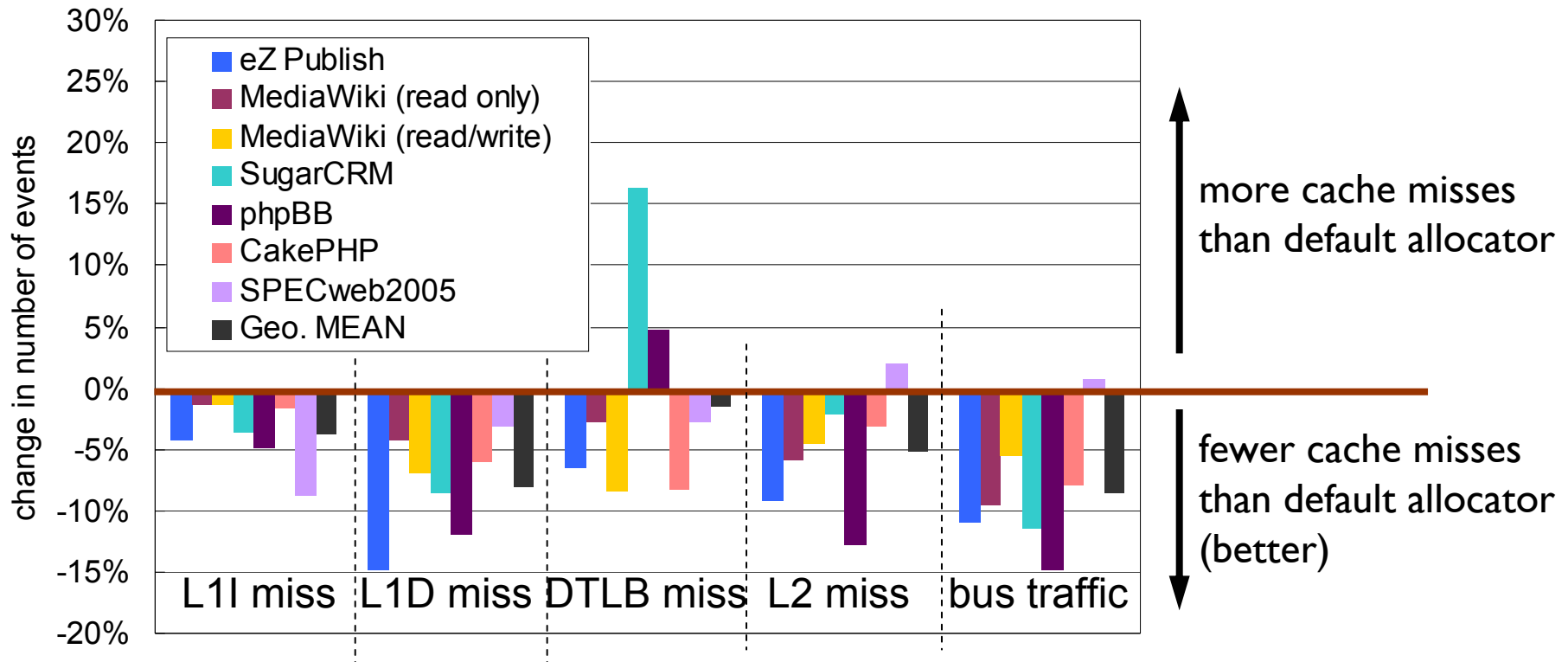
for ezPublish

Performance of DDmalloc: Execution time breakdown



for ezPublish

Performance of DDmalloc: Cache misses and bus traffic



on eight cores

Summary

- We studied the effects of memory management approaches on the performance of Web-based applications on multicore processors
 - region-based allocator: fast on a single core, but slow on multicore processors due to increased bus traffic
 - general-purpose allocator: not cost-effective in avoiding heap fragmentation
- We proposed the new approach of *Defrag-Dodging* to reduce memory management costs

More data on the paper

- ✓ evaluation on Niagara
- ✓ evaluation with Ruby runtime
- ✓ comparisons with TCmalloc, Hoard
- ✓ discussions on the GC-based languages