Two Memory Allocators that Use Hints to Improve Locality

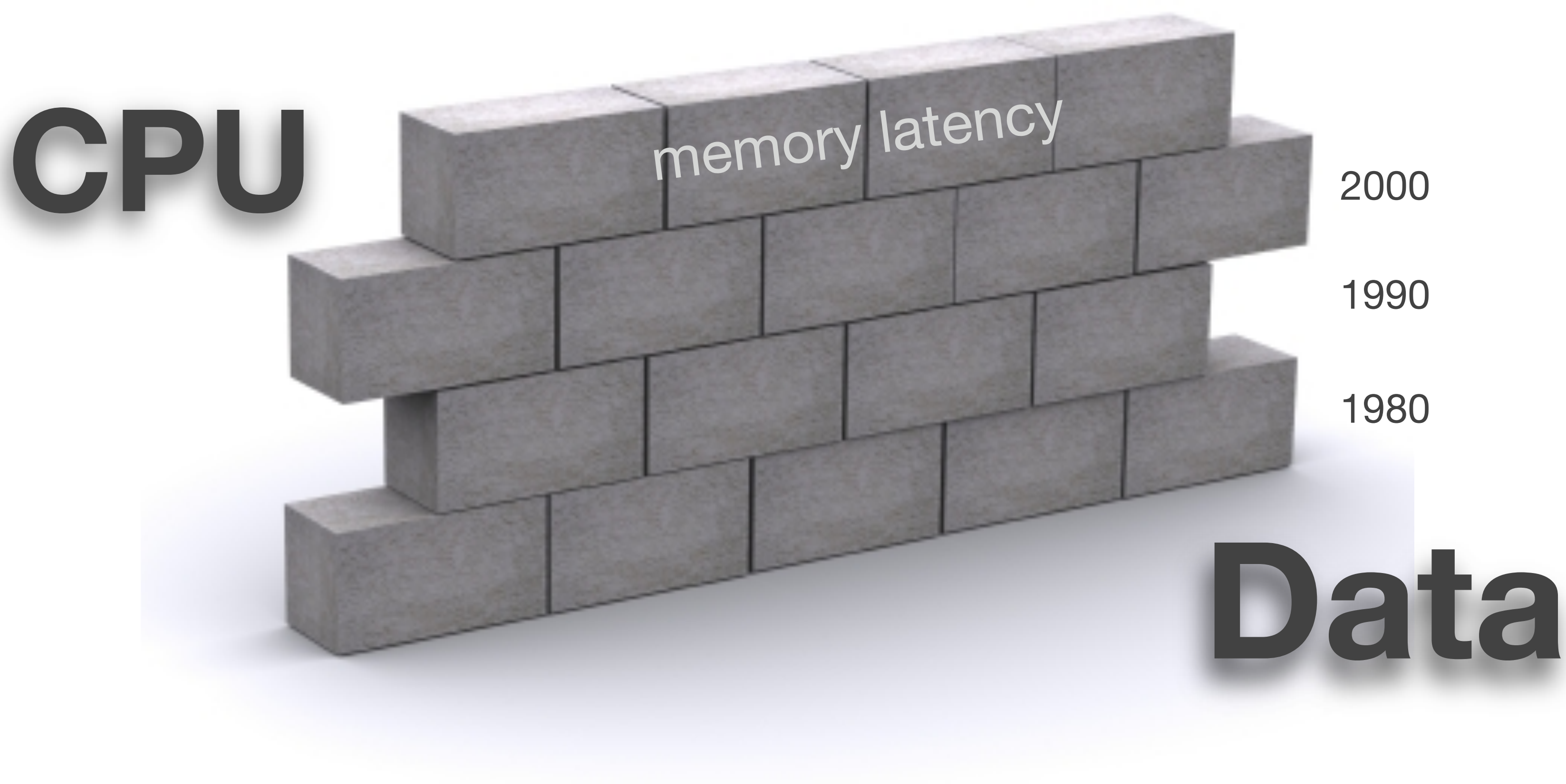
Fast Allocation Speed, Low Memory Fragmentation, and High Spatial Locality: Pick Two.

Lawrence Rauchwerger
Texas A&M University
College Station, TX, USA

Alin Jula
SAP Labs
Palo Alto, CA, USA

June 20, ISMM 2009, Dublin

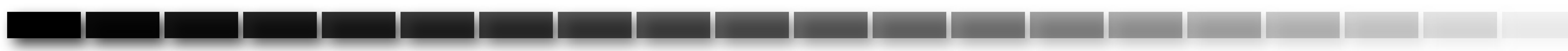
Locality - Performance Bottleneck



Memory management could reduce latency by better data layout at runtime

The Locality Problem: a Bird's Eye View

Allocator



Application

The Locality Problem: a Bird's Eye View

Allocator

allocation

Application

The Locality Problem: a Bird's Eye View

Allocator

deallocation

Application

The Locality Problem: a Bird's Eye View

Memory chunks are now disordered !

Allocator

deallocation

Application

Problem = next allocation will exhibit poor locality

Approaches to Improving Locality

- Explicit memory allocation, e.g. C/C++
  - Locality of reference: FreeBSD,Linux, Vam
  - Using allocation hints: ccmalloc, ialloc
  - Compiler
  - Software pre-fetching
  - Profiling (use context)
- Implicit - garbage collection, e.g. Java, M/L (region-based)

Approaches to Improving Locality

- Explicit memory allocation, e.g. C/C++
  - Locality of reference: FreeBSD,Linux, Vam
  - Using allocation hints: ccmalloc, ialloc
  - Compiler
  - Software pre-fetching
  - Profiling (use context)
- Implicit - garbage collection, e.g. Java, M/L (region-based)

What is the Challenge?

- Fragmentation = extra memory used by allocator
  - Theoretical lower bound 50% rule, Observed bound 20%
- Allocation speed = instructions spent in allocator
  - Participates directly in an application's performance
- Locality = data layout of allocated memory
  - Online algorithm (hard) - at runtime, with no chance to rearrange memory

What is the Challenge?

Fragmentation + Speed + Locality = Hard

- Fragmentation = extra memory used by allocator
  - Theoretical lower bound 50% rule, Observed bound 20%
- Allocation speed = instructions spent in allocator
  - Participates directly in an application’s performance
- Locality = data layout of allocated memory
  - Online algorithm (hard) - at runtime, with no chance to rearrange memory

Locality Improving for C/C++

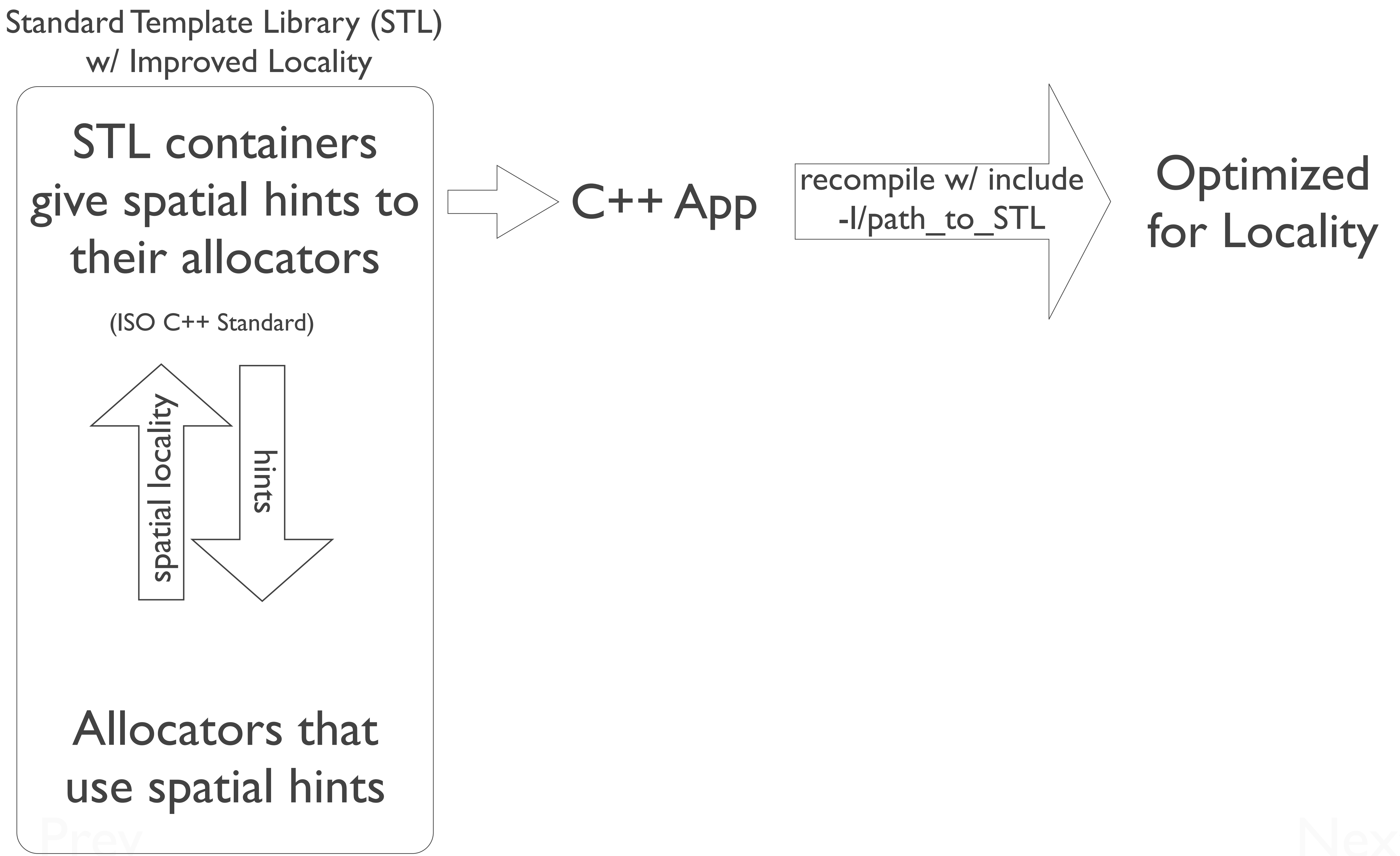
1. Temporal: Locality of Reference
   - Allocates consecutive requests in same vicinity, e.g. FreeBSD, Vam
   - ✓ Easy to use
   - ✗ No application feedback
2. Spatial: Use Allocation Hints
   - Cache line collocation and static tree organization, e.g. ccmalloc
   - Instance interleaving, e.g. ialloc
   - ✓ Tailored for each application
   - ✗ Code modifications, manual hints impractical, cache only

Locality Improving for C/C++

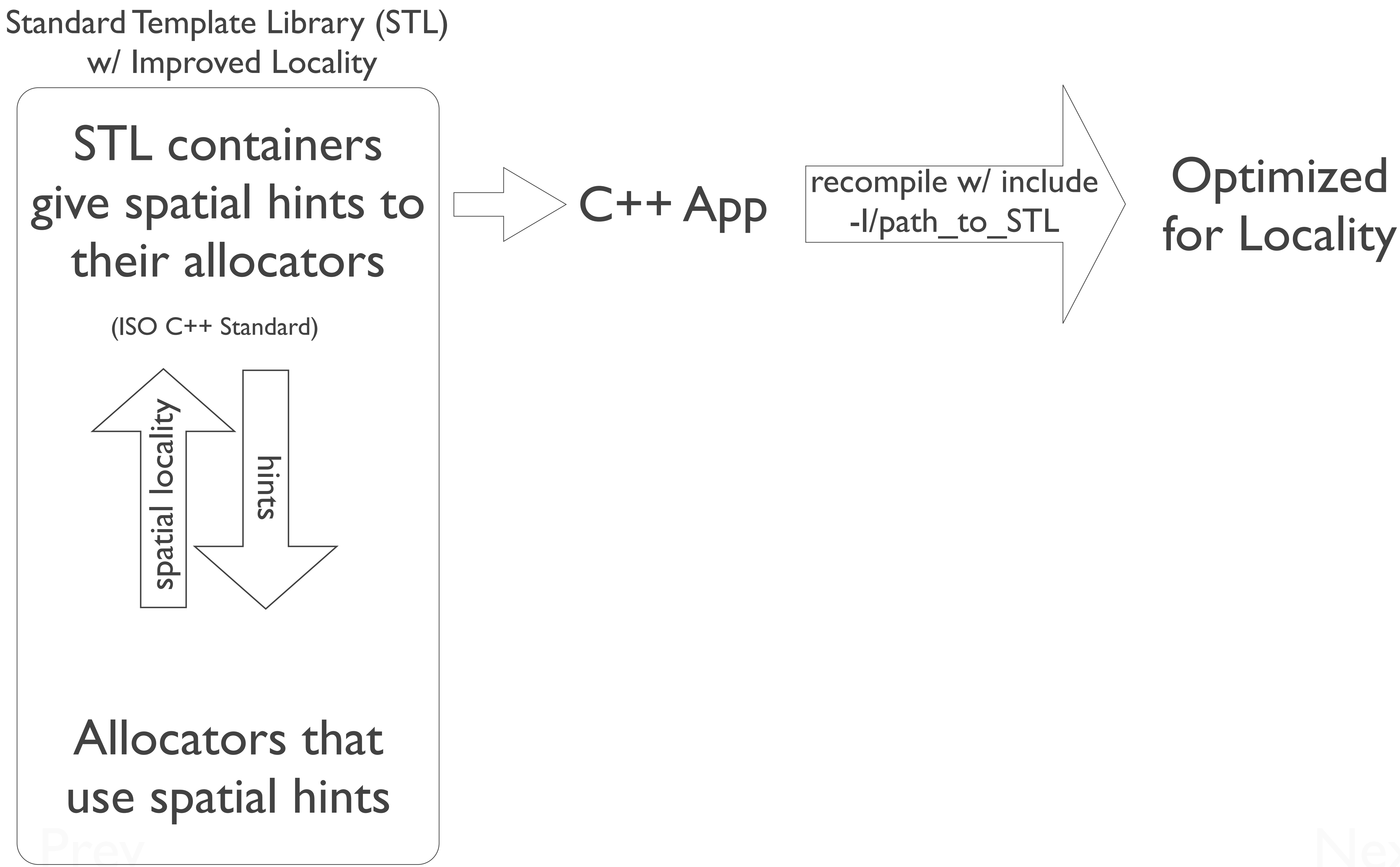
Our Work - Both Temporal and Spatial

1. Temporal: Locality of Reference
   - Allocates consecutive requests in same vicinity, e.g. FreeBSD, Vam
   - ✓ Easy to use
   - ✗ No application feedback
2. Spatial: Use Allocation Hints
   - Cache line collocation and static tree organization, e.g. ccmalloc
   - Instance interleaving, e.g. ialloc
   - ✓ Tailored for each application
   - ✗ Code modifications, manual hints impractical, cache only

The Idea:


Standard Template Library (STL)
w/ Improved Locality
STL containers give spatial hints to their allocators
(ISO C++ Standard)
spatial locality
hints
Allocators that use spatial hints
C++ App
recompile w/ include
-I/path_to_STL
Optimized for Locality

The Idea: *No costs! Just recompile the application.*



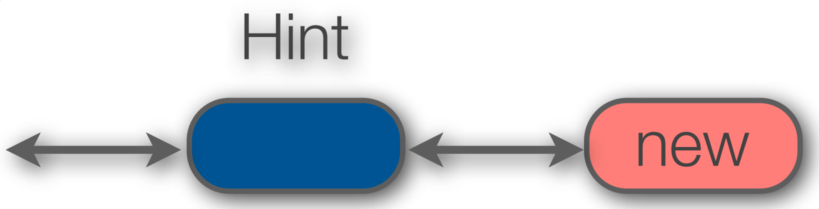
1) How to select hints?

2) How does an allocator efficiently use the hints?

Next

How to Select Hints?

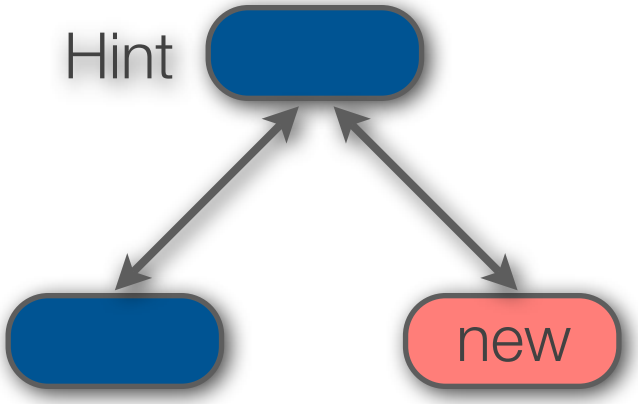
List



Linear traversal

Hint = address of previous element

Tree



DFS traversal (set, map)

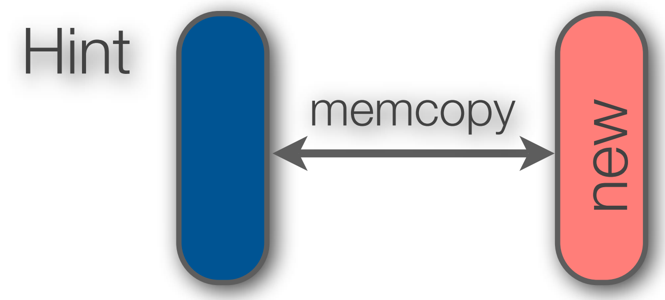
Hint = parent’s address (sibling’s -BFS)

Deque

Linear traversal

Hint = neighboring array’s address

Vector



`vector()` and `operator=`

Hint = source’s address

How Does an Allocator Use Hints

1. **Interface** - ISO C++ Standard
   - void* allocator<T>::allocate(size_t size,**void* address**)

2. **Management Strategy**
   - Index free blocks on ‘address’, too

3. **Allocation Strategy**
   1. Try to find a block in the hint’s vicinity (fail fast)
   2. Otherwise, abandon locality and allocate based on size

Managing Address for Locality

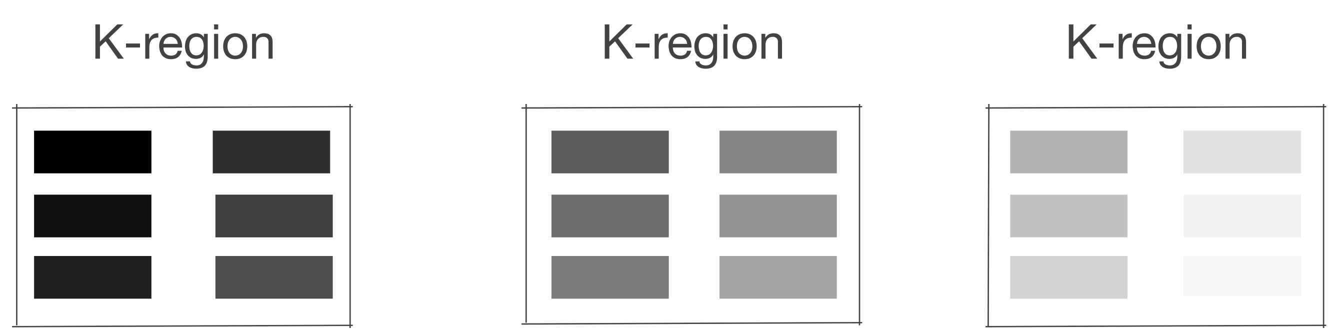
- The whole memory is organized in **K-regions**
- K-region = a contiguous block of size $2^{64\text{-}K}$ aligned to its size
  - X, Y ∈ same K-region iff they share the 64-K most significant bits
  - Examples
    - 0x**0000**FFFF and 0x**0000**CCCC in same 16-region
    - A 64-byte cache line is a 6-region
    - A 4 KB virtual page is a 12-region
- K ∈ [0,64) **adjustable**

Managing Address for Locality

Idea: Approximate address search within a K-region

- The whole memory is organized in **K-regions**
- K-region = a contiguous block of size 2^{64-K} aligned to its size
  - X, Y ∈ same K-region iff they share the 64-K most significant bits
  - Examples
    - 0x**0000**FFFF and 0x**0000**CCCC in same 16-region
    - A 64-byte cache line is a 6-region
    - A 4 KB virtual page is a 12-region
- K ∈ [0,64) **adjustable**

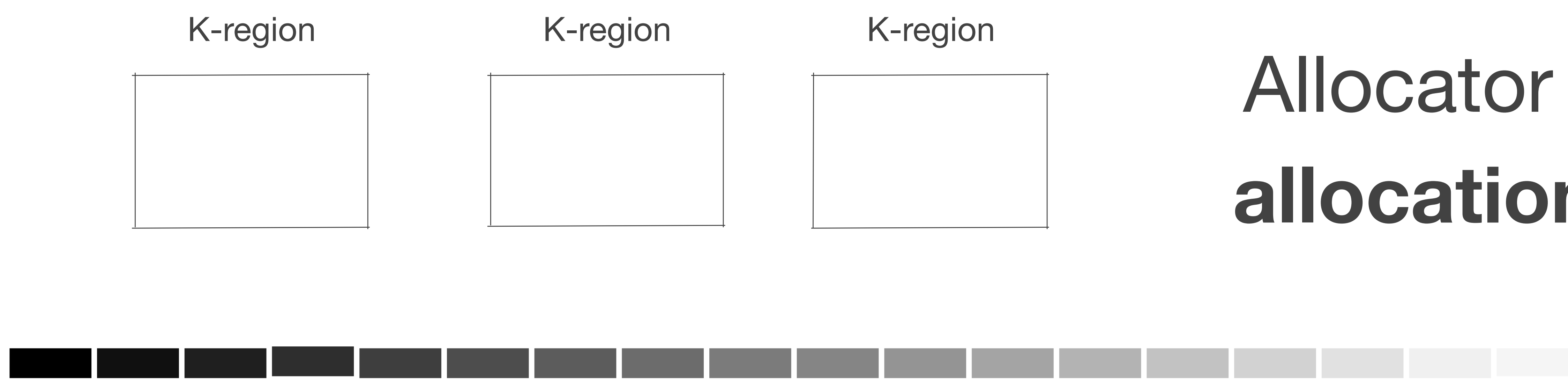
Locality Improved: a Bird's Eye View



Allocator

Application

Locality Improved: a Bird's Eye View

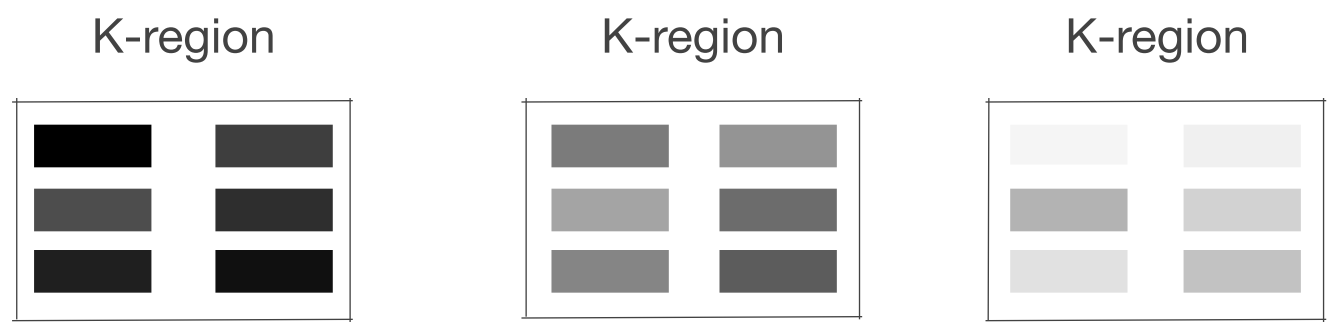


Allocator

allocation

Application

Locality Improved: a Bird's Eye View

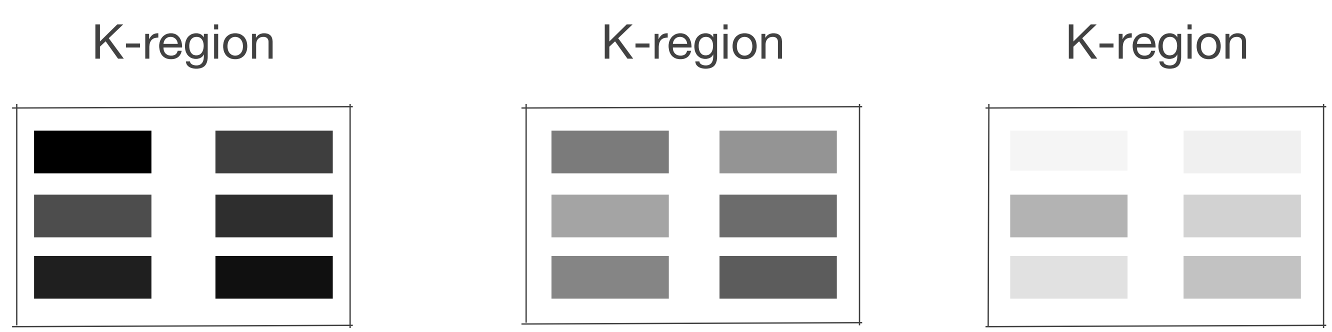


Allocator
deallocation

Application

Locality Improved: a Bird's Eye View

Memory chunks are now partially ordered



Allocator

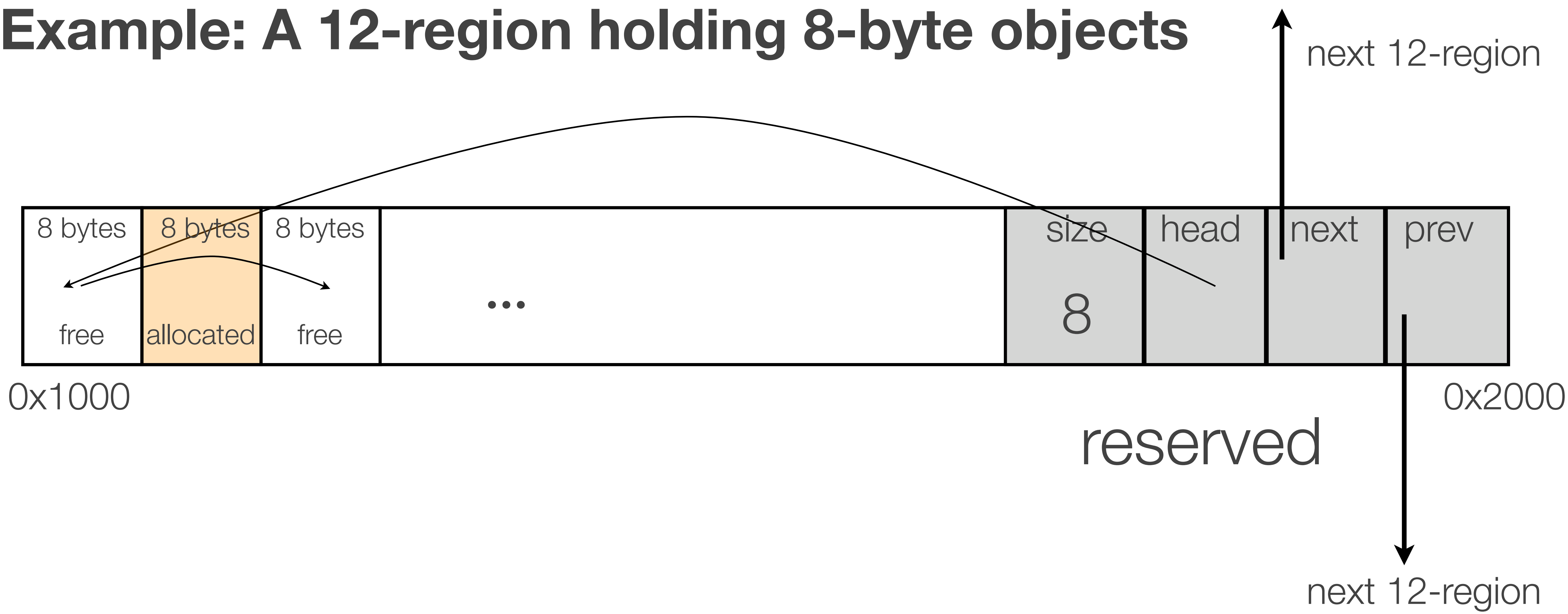
deallocation

Application

Next allocation will exhibit better locality

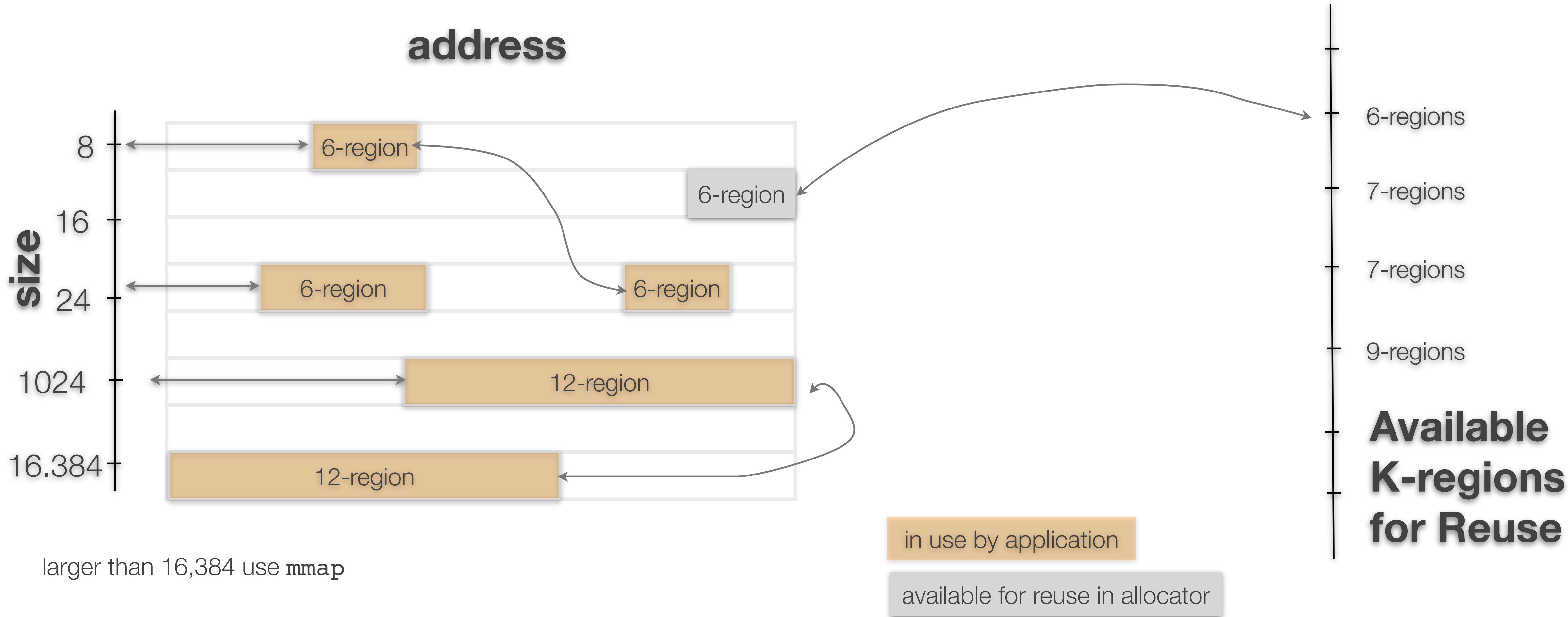
Anatomy of a K-region

Example: A 12-region holding 8-byte objects



- Holds objects of **same** size in a linked list
- Topmost 16 bytes reserved
- Software alignment

Two Partitions (TP) Allocator



- **Allocation Strategy: Address then Size. O(1)**
  - If hint's K-region is not empty then select first block
  - Else select first block on size
- **Deallocation: O(1)**
  - Return block to its K-region

TP's Size Segregation and Its Progressive K-Region Selection

Size Classes		K-region
0 - 512 B	: 64 classes, 8 bytes apart	12-region (4 KB)
512 - 1KB	: 8 classes, 64 bytes apart	13-region (8 KB)
1 KB - 2 KB	: 8 classes, 128 bytes apart	14-region (16 KB)
2 KB - 4 KB	: 8 classes, 256 bytes apart	15-region (32 KB)
4 KB - 8 KB	: 8 classes, 512 bytes apart	16-region (64 KB)
8 KB - 16 KB	: 8 classes, 1024 bytes apart	17-region (128 KB)

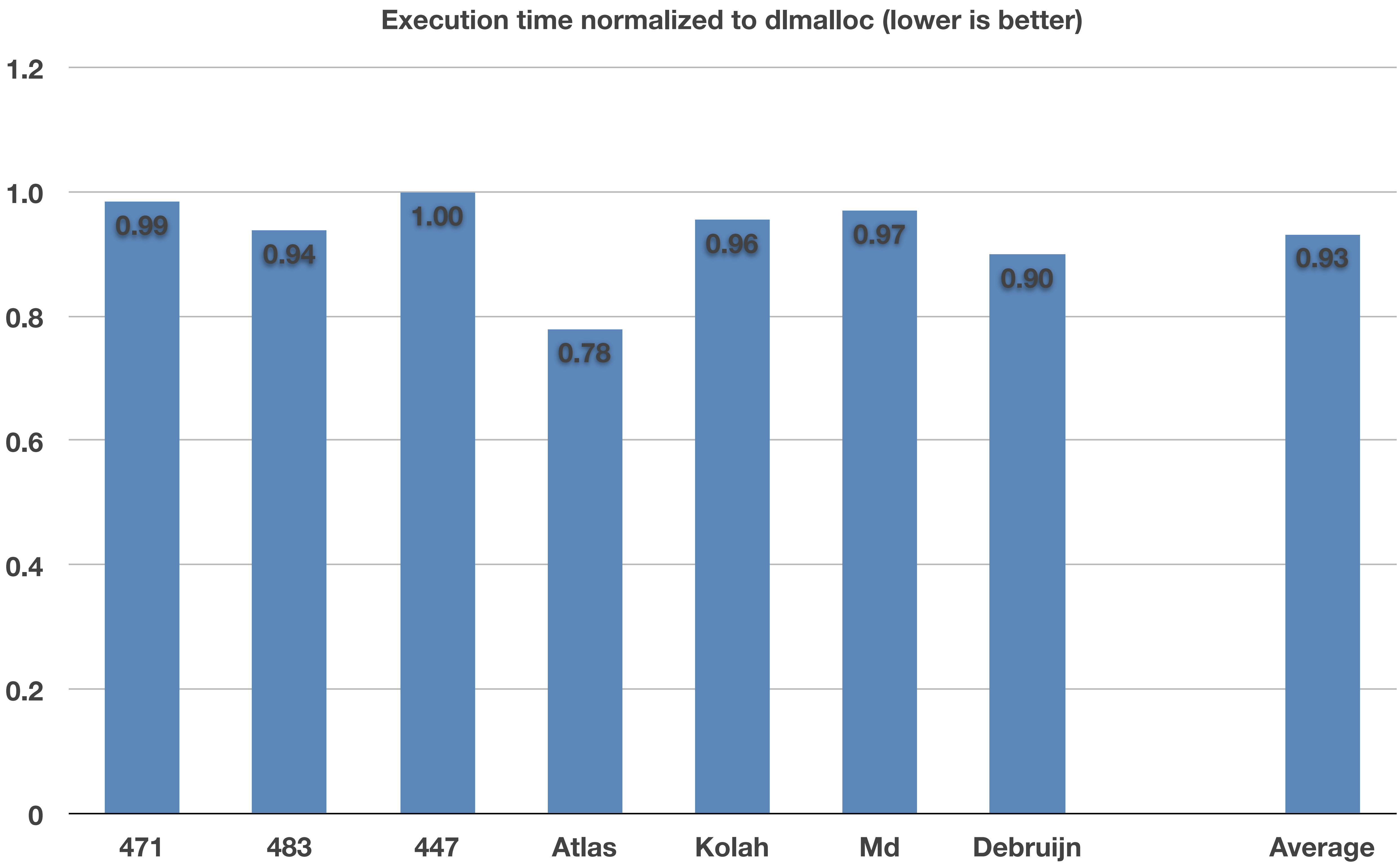
Larger than 16 KB managed by `mmap` and `munmapped`

C++ STL Benchmarks Suite

- Seven **large** and **real world** applications

	Lines of code	Files	Execution time (sec)
447.dealII - adaptive finite elements (Spec 2006)	198,649	452	1,040
471.omnetpp - even simulation (Spec 2006)	47,910	155	1,152
483.xalancbmk - XML processor (Spec 2006)	553,643	1,773	841
Atlas- mathematical application (Atlas Lie Group)	53,869	230	557
Kolah - mesh transport (Lawrence National Labs)	52,921	227	70
Md -Molecular dynamics (Sandia National Labs)	1,259	30	64
Debruijn - network simulation (Texas A&M)	499	3	15

TP vs. Dlmalloc



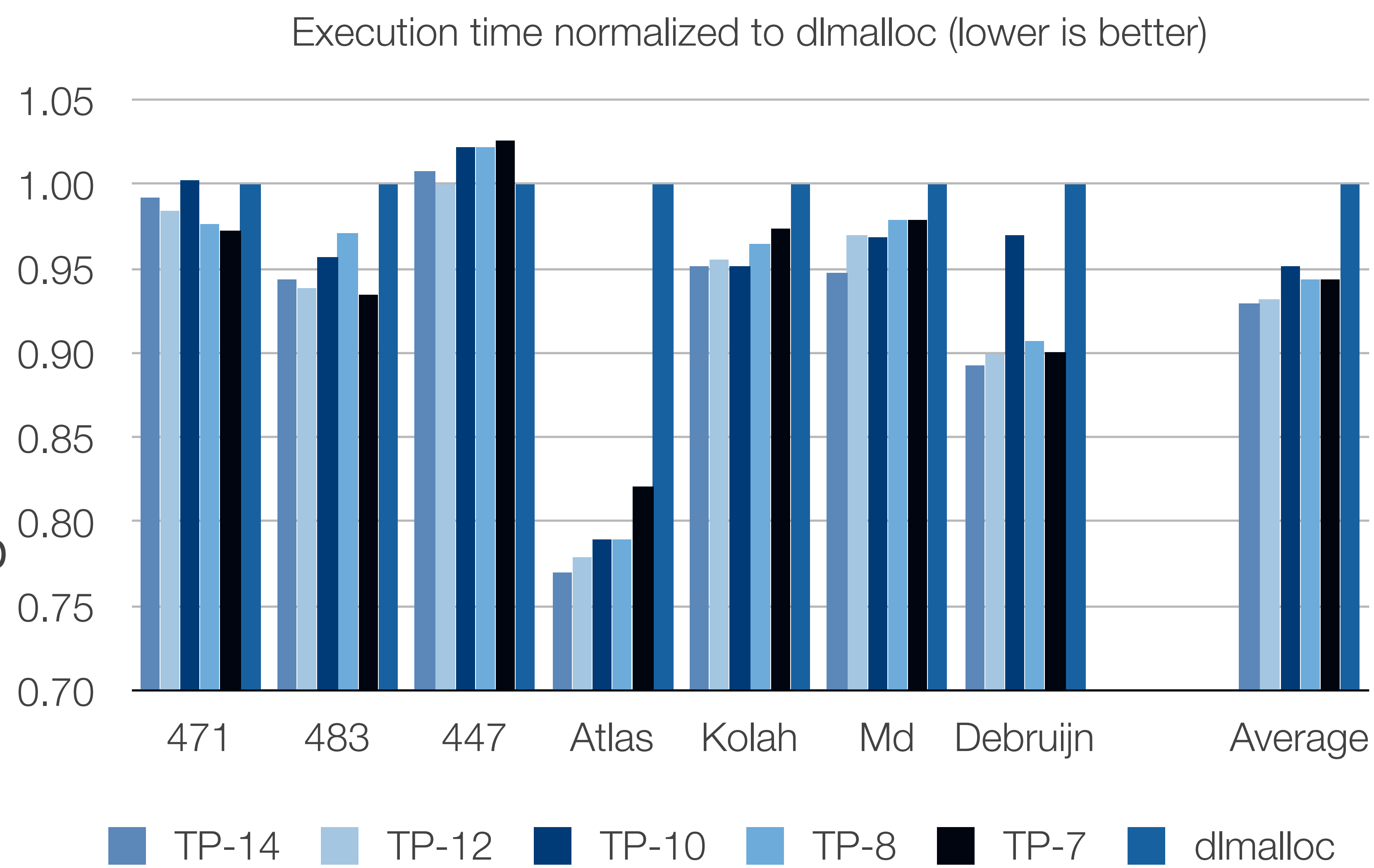
Locality Precision Study

- We varied K between 7 and 14
  - Only for objects < 512 bytes (≈ 95%)
  - Notation: TP-7 uses 7-regions for small objects
- Why K-regions other than cache lines or virtual pages could improve locality?
  - They increase the **probability** of allocating in the same cache line or virtual page

TP's Performance Improvement

TP vs dlmalloc

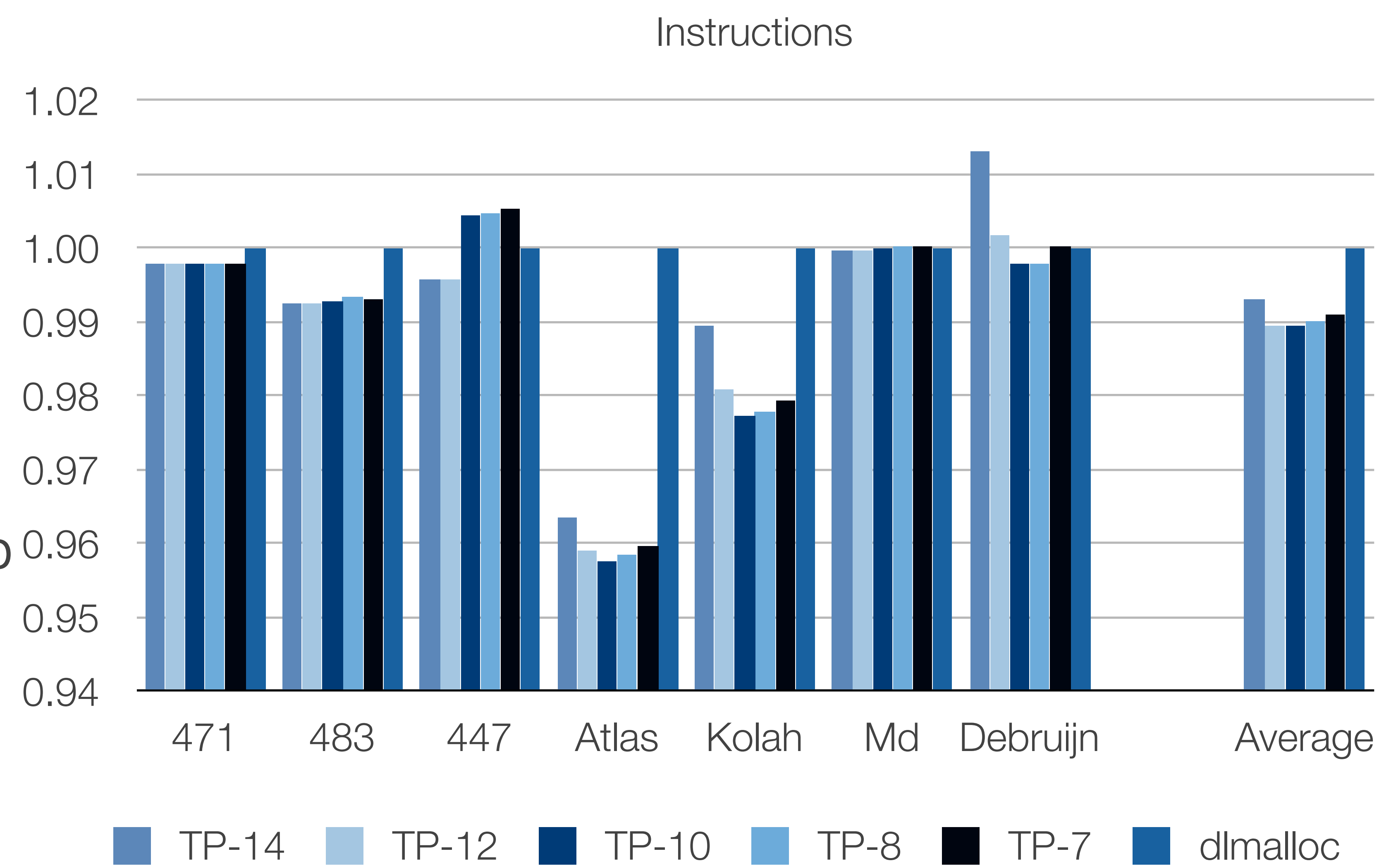
- Time: 5-7%
- Instructions: 1%
- TLB Misses: 13-18%
- L1 Misses: 1-5%
- L2 Misses: 10-12%
- Default K's value 12



TP's Performance Improvement

TP vs dlmalloc

- Time: 5-7%
- Instructions: 1%
- TLB Misses: 13-18%
- L1 Misses: 1-5%
- L2 Misses: 10-12%
- Default K's value 12



TP's Performance Improvement

TP vs dlmalloc

- Time: 5-7%
- Instructions: 1%
- TLB Misses: 13-18%
- L1 Misses: 1-5%
- L2 Misses: 10-12%
- Default K's value 12

Two pitfalls

1. Whole K-region aquisition
2. Hint from allocated memory

TP's Performance Improvement

TP vs dlmalloc

- Time: 5-7%
- Instructions: 1%
- TLB Misses: 13-18%
- L1 Misses: 1-5%
- L2 Misses: 10-12%
- Default K's value 12

Two pitfalls

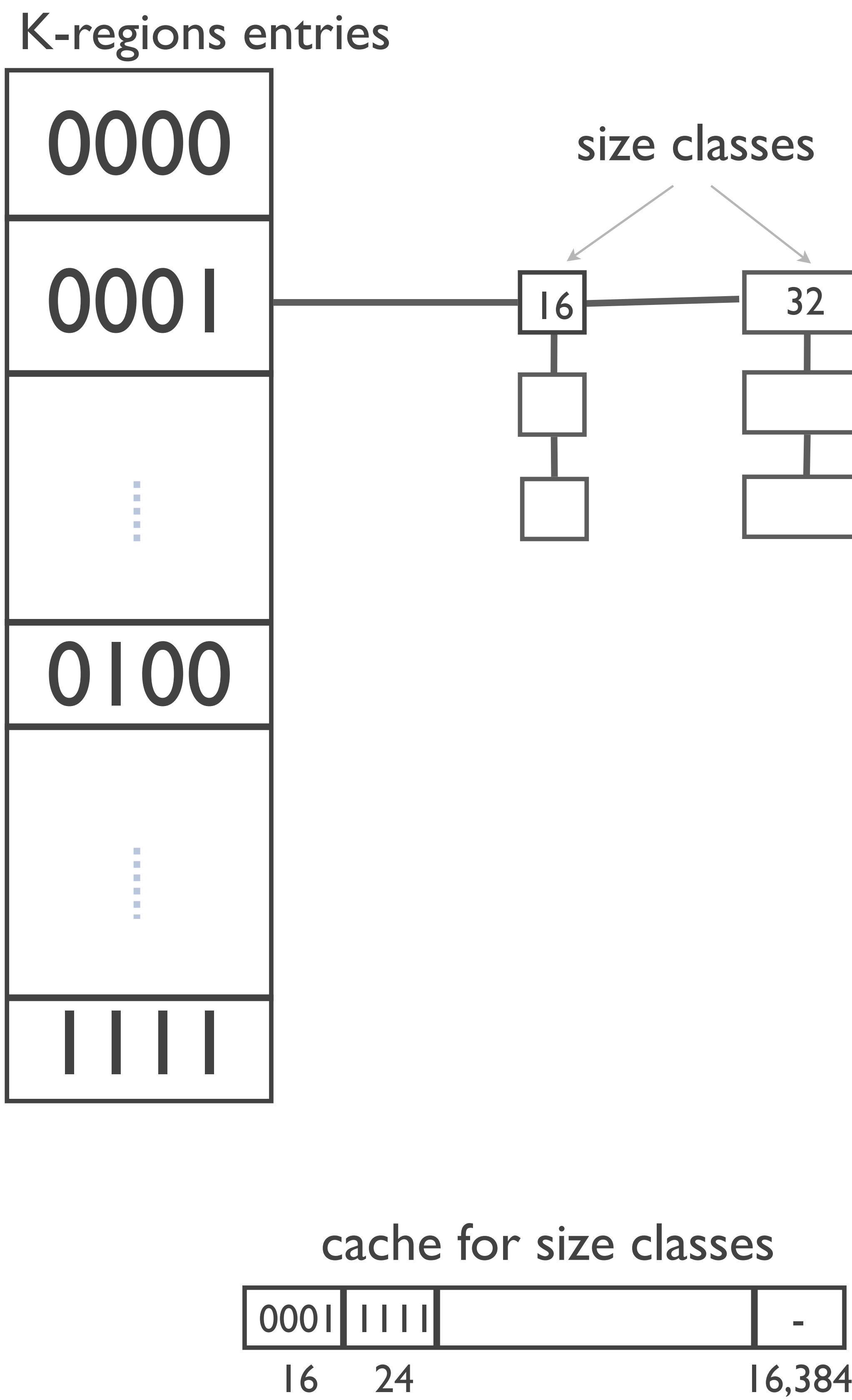
1. Whole K-region aquisition
2. Hint from allocated memory

Can we do better?

Medius - Another Approach

- Addresses TP's pitfalls
  - no alignment, hint need not be from allocated memory
- Uses same K-regions
  - stores them in an array 2^{32-K} size
- K-region = list of lists
  - size classes store blocks of same size in a list
  - K-regions store size classes in a list
- Prioritizes address over size
- O(1) access to hint's K-region

Medius allocator



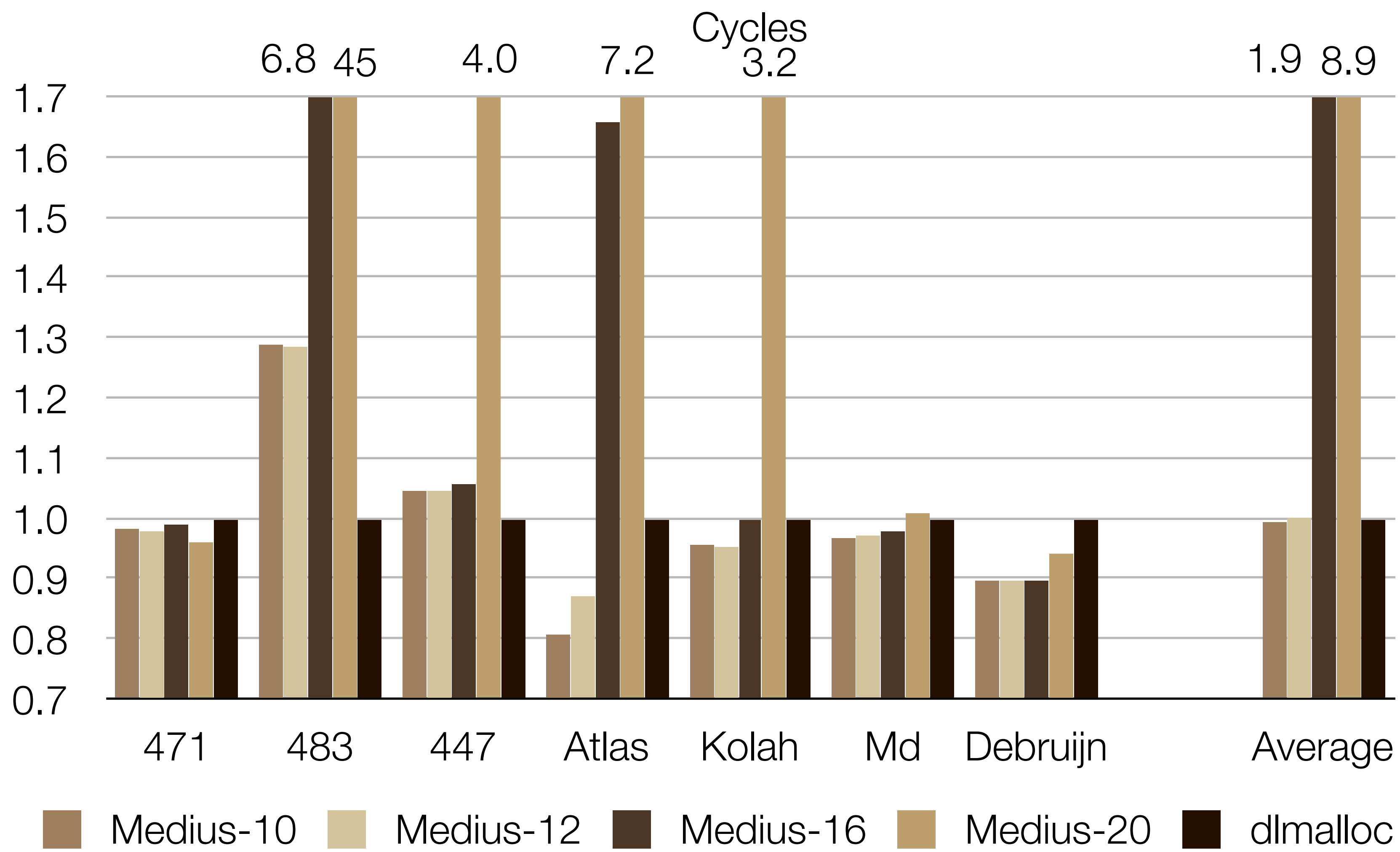
- **Allocation - O(s) on average**
  - If hint's K-region is not empty select size class and return
  - Else select first K-region with size class (from cache)
- **Deallocation - O(s)**
  - return block to its K-region

- **Strategy: Address and then size**

Medius' Performance Improvement

Medius vs dlmalloc

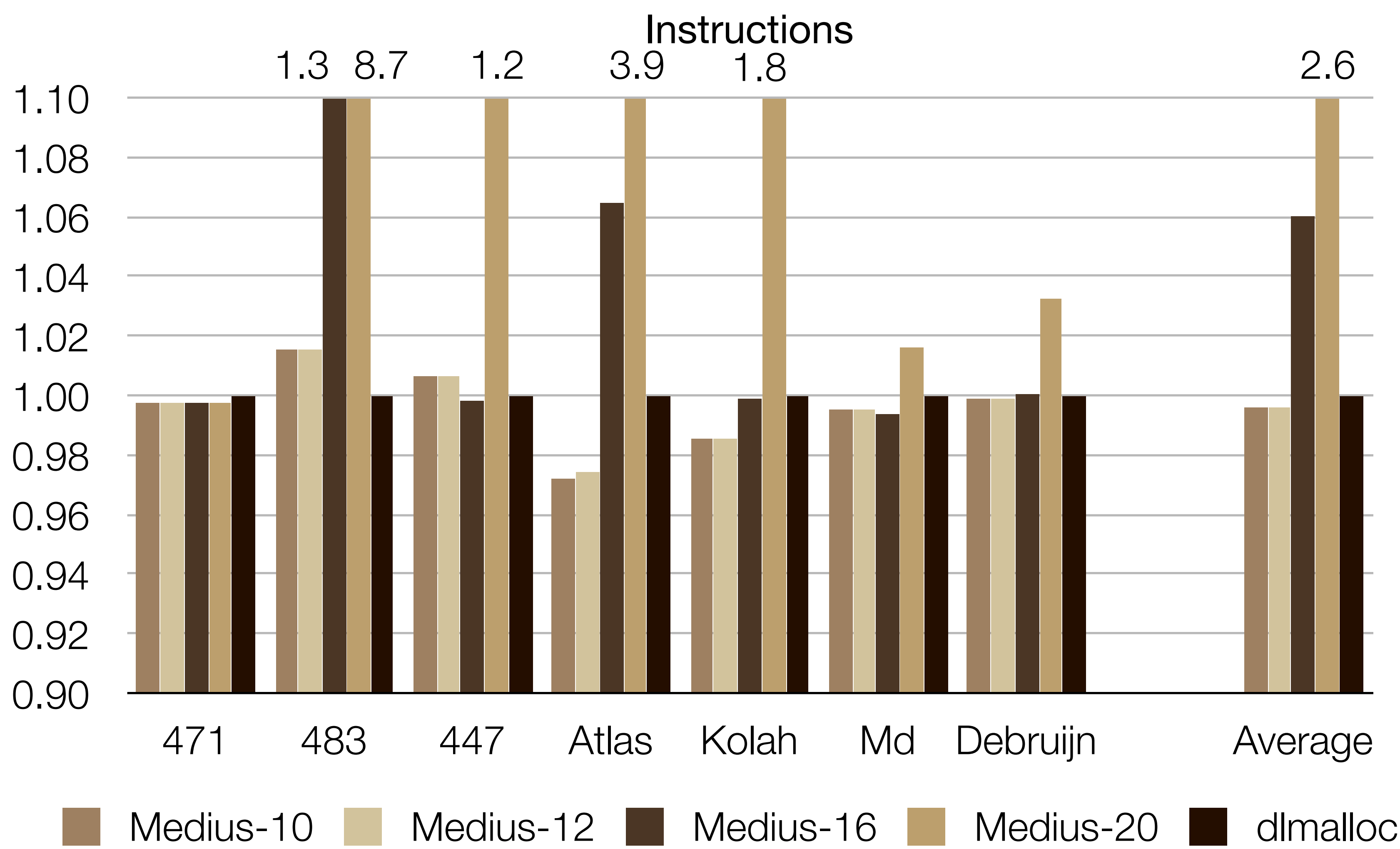
- Time: 0%
- Instructions: 0.36%
- TLB Misses: -13%
- L1 Misses: -3%
- L2 Misses: -3%
- Default K's value 12



Medius' Performance Improvement

Medius vs dlmalloc

- Time: 0%
- Instructions: 0.36%
- TLB Misses: -13%
- L1 Misses: -3%
- L2 Misses: -3%
- Default K's value 12



Medius' Performance Improvement

Medius vs dlmalloc

- Time: 0%
- Instructions: 0.36%
- TLB Misses: -13%
- L1 Misses: -3%
- L2 Misses: -3%
- Default K's value 12

Two pitfalls

1. Slower
2. No memory reuse

Medius' Performance Improvement

Medius vs dlmalloc

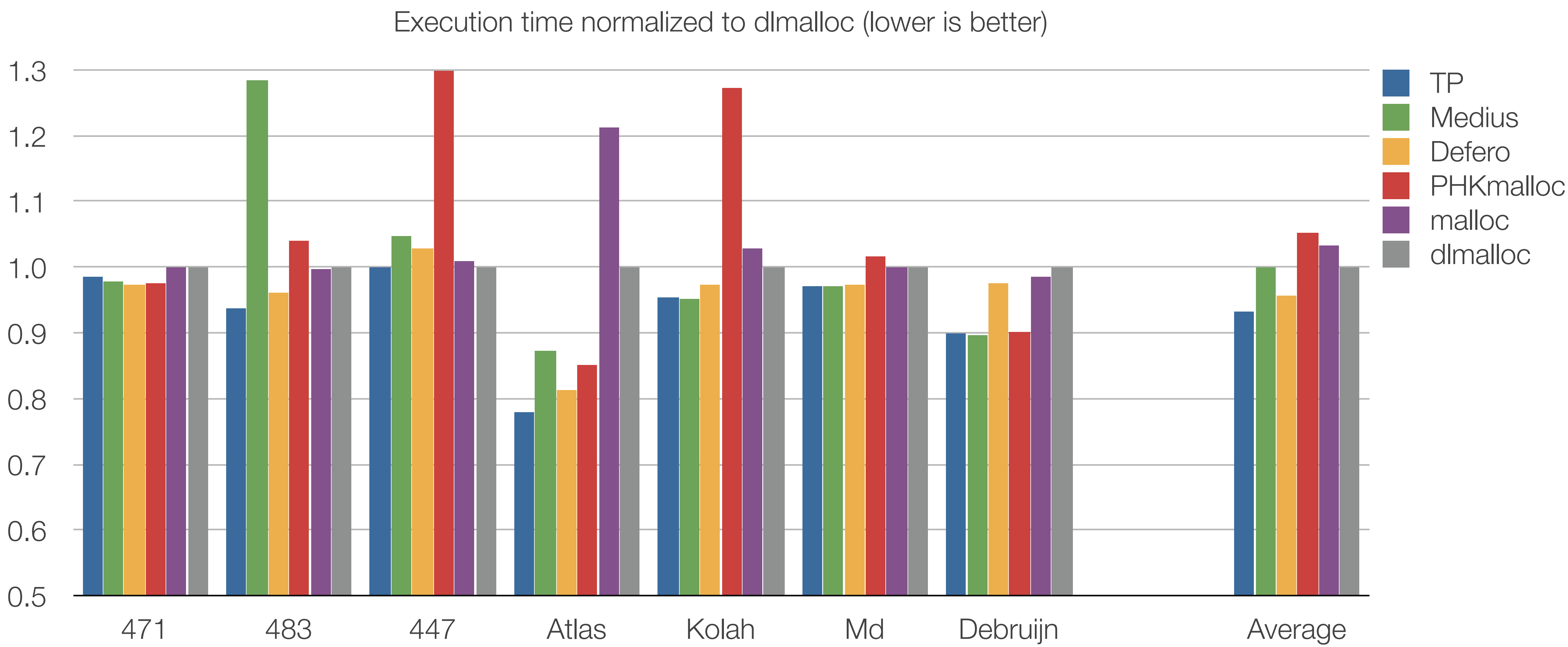
- Time: 0%
- Instructions: 0.36%
- TLB Misses: -13%
- L1 Misses: -3%
- L2 Misses: -3%
- Default K's value 12

Two pitfalls

1. Slower
2. No memory reuse

We did worse !

Comparison of Allocators



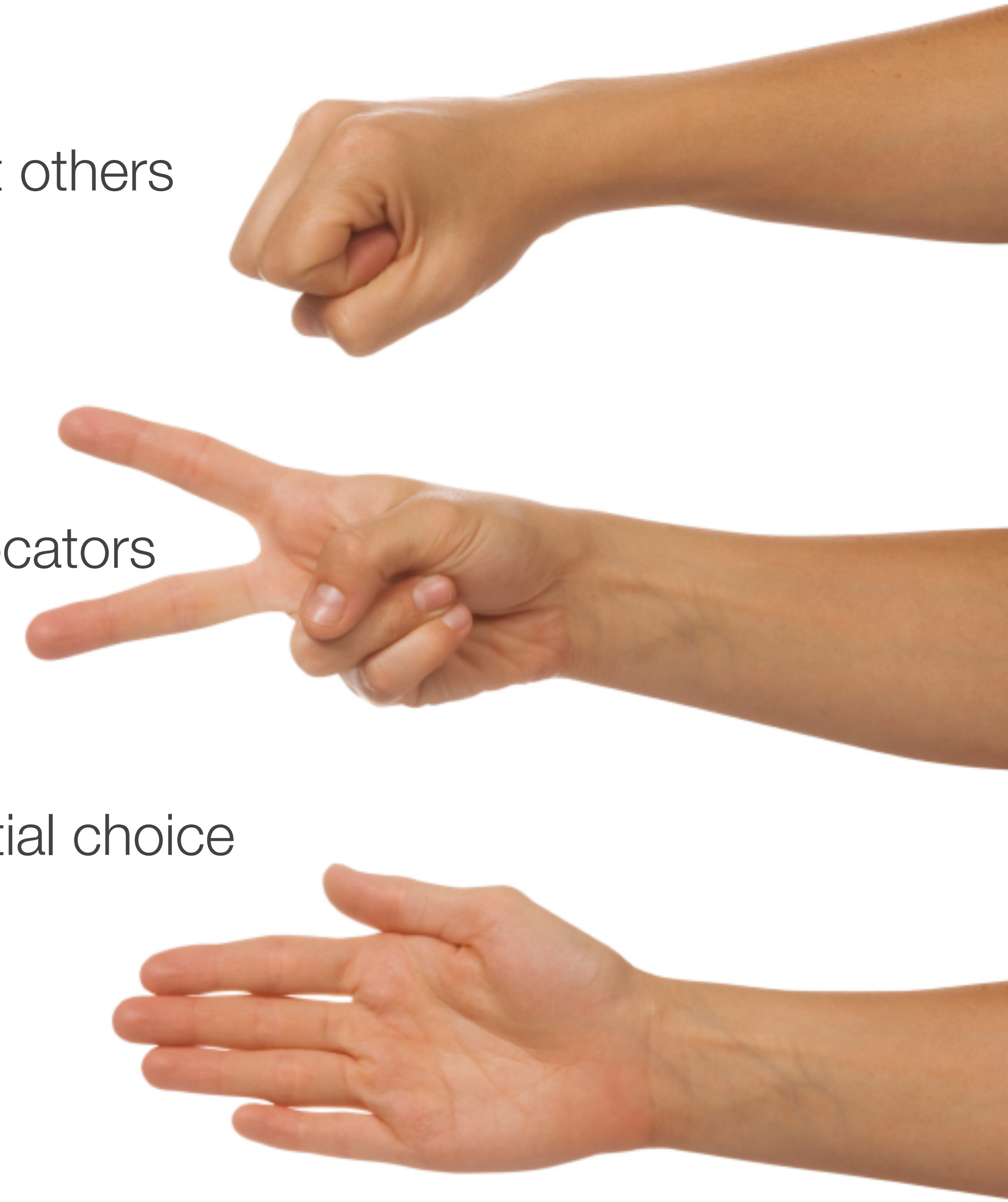
- FreeBSD’s allocator - PHKmalloc
- Defero - locality improving allocator that uses hints
- Vam - locality of reference
- native 4.1.2 gnu malloc (older version of dlmalloc)

Fragmentation

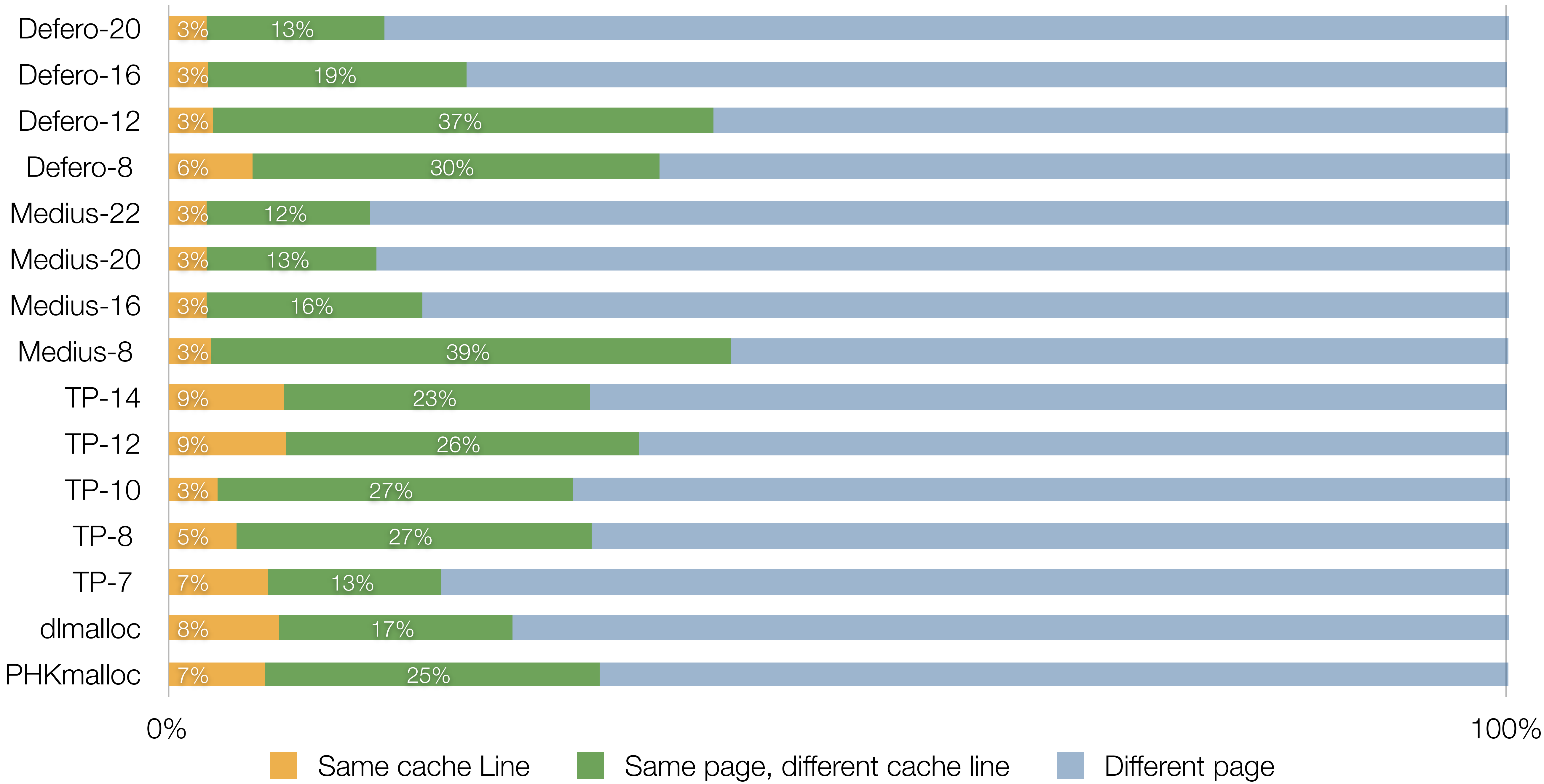
Benchmarks	TP	Medius	Defero	PHK	dlmalloc
471.omnetpp	0.6	3.2	1.5	58.0	23.5
483.xalancbmk	75.6	37.2	28.4	30.3	1.3
447.dealII	21.2	25.3	24.7	57.0	18.2
Atlas	10.8	12.9	7.2	23.4	2.7
Kolah	1.1	0.9	0.9	5.4	12.3
Md	21.4	20.15	22.14	20.7	10.7
Debruijn	6.6	44.7	44.7	1.3	6.8
Geometric Mean	***7.7***	***11.7***	***9.5***	***18.1***	***7.2***

Speed, Fragmentation and Locality

- Rock, paper, scissors
  - Efforts to increase one trait hurt others
  - Intra and inter allocators
- Speed v. Fragmentation
  - Studied relation: the fastest allocators have high fragmentation
- Locality v. Fragmentation
  - Less searching space limits spatial choice
- Locality v. Speed
  - Searching takes additional time



Locality Accuracy - Benefits w/o Costs



Fast Allocation Speed,
Low Memory Fragmentation,
and High Spatial Locality:
Pick Two.

Allocation Speed (instr)

1. TP-12 (98.96)
2. TP-10 (98.96)
3. TP-8 (99.00)
4. TP-7 (99.09)
5. Defero-20 (99.18)
6. TP-14 (99.31)
7. Defero-16 (99.55)
8. Medius-22 (99.61)
9. Medius-20 (99.64)
10. dlmalloc (100.00)
11. Defero-12 (100.08)
12. Defero-8 (100.90)
13. PHKmalloc (119.31)
14. Medius-16 (106.08)
15. Medius-12 (268.37)

Fast Allocation Speed,
Low Memory Fragmentation,
and High Spatial Locality:
Pick Two.

Allocation Speed (instr)

1.TP-12 (98.96)
2.TP-10 (98.96)
3.TP-8 (99.00)
4.TP-7 (99.09)
5.Defero-20 (99.18)
6.TP-14 (99.31)
7.Defero-16 (99.55)
8.Medius-22 (99.61)
9.Medius-20 (99.64)
10.dlmalloc (100.00)
11.Defero-12 (100.08)
12.Defero-8 (100.90)
13.PHKmalloc (119.31)
14.Medius-16 (106.08)
15.Medius-12 (268.37)

Fast Allocation Speed,
Low Memory Fragmentation,
and High Spatial Locality:
Pick Two.

Allocation Speed (instr)

1. TP-12 (98.96)
2. TP-10 (98.96)
3. TP-8 (99.00)
4. TP-7 (99.09)
5. Defero-20 (99.18)
6. TP-14 (99.31)
7. Defero-16 (99.55)
8. Medius-22 (99.61)
9. Medius-20 (99.64)
10. dlmalloc (100.00)
11. Defero-12 (100.08)
12. Defero-8 (100.90)
13. PHKmalloc (119.31)
14. Medius-16 (106.08)
15. Medius-12 (268.37)

Locality (page accuracy)

1. Medius-12 (42%)
2. Defero-12 (40%)
3. Defero-8 (36%)
4. TP-12 (35%)
5. TP-14 (32%)
6. TP-8 (32%)
7. PHKmalloc (32%)
8. TP-10 (30%)
9. dlmalloc (25%)
10. Defero-16 (22%)
11. TP-7 (20%)
12. Medius-16 (19%)
13. Medius-20 (16%)
14. Defero-20 (16%)
15. Medius-22 (15%)

Fast Allocation Speed,
Low Memory Fragmentation,
and High Spatial Locality:
Pick Two.

Allocation Speed (instr)

1.TP-12 (98.96)
2.TP-10 (98.96)
3.TP-8 (99.00)
4.TP-7 (99.09)
5.Defero-20 (99.18)
6.TP-14 (99.31)
7.Defero-16 (99.55)
8.Medius-22 (99.61)
9.Medius-20 (99.64)
10.dlmalloc (100.00)
11.Defero-12 (100.08)
12.Defero-8 (100.90)
13.PHKmalloc (119.31)
14.Medius-16 (106.08)
15.Medius-12 (268.37)

Locality (page accuracy)

1. Medius-12 (42%)
2. Defero-12 (40%)
3. Defero-8 (36%)
4. TP-12 (35%)
5. TP-14 (32%)
6. TP-8 (32%)
7. PHKmalloc (32%)
8. TP-10 (30%)
9. dlmalloc (25%)
10.Defero-16 (22%)
11.TP-7 (20%)
12.Medius-16 (19%)
13.Medius-20 (16%)
14.Defero-20 (16%)
15.Medius-22 (15%)

Fragmentation

1. dlmalloc (7.2%)
2.TP-12 (7.7%)
3.Defero-20 (9.5%)
4.Defero-16 (9.5%)
5.Defero-12 (9.5%)
6.Defero-8 (9.5%)
7.Medius-22 (11.0%)
8.Medius-20 (11.3%)
9.Medius-16 (16.7%)
10.PHKmalloc (18.1%)
11.TP-10 (18.5%)
12.TP-8 (20.3%)
13.Medius-12 (27.3%)
14.TP-7 (29.2%)
15.TP-14 (39.5%)

Fast Allocation Speed, Low Memory Fragmentation, and High Spatial Locality: Pick Two.

Allocation Speed (instr)

1.TP-12 (98.96)
2.TP-10 (98.96)
3.TP-8 (99.00)
4.TP-7 (99.09)
5.Defero-20 (99.18)
6.TP-14 (99.31)
7.Defero-16 (99.55)
8.Medius-22 (99.61)
9.Medius-20 (99.64)
10.dlmalloc (100.00)
11.Defero-12 (100.08)
12.Defero-8 (100.90)
13.PHKmalloc (119.31)
14.Medius-16 (106.08)
15.Medius-12 (268.37)

Locality (page accuracy)

1. Medius-12 (42%)
2. Defero-12 (40%)
3. Defero-8 (36%)
4. TP-12 (35%)
5. TP-14 (32%)
6. TP-8 (32%)
7. PHKmalloc (32%)
8. TP-10 (30%)
9. dlmalloc (25%)
10.Defero-16 (22%)
11.TP-7 (20%)
12.Medius-16 (19%)
13.Medius-20 (16%)
14.Defero-20 (16%)
15.Medius-22 (15%)

Fragmentation

1. dlmalloc (7.2%)
2.TP-12 (7.7%)
3.Defero-20 (9.5%)
4.Defero-16 (9.5%)
5.Defero-12 (9.5%)
6.Defero-8 (9.5%)
7.Medius-22 (11.0%)
8.Medius-20 (11.3%)
9.Medius-16 (16.7%)
10.PHKmalloc (18.1%)
11.TP-10 (18.5%)
12.TP-8 (20.3%)
13.Medius-12 (27.3%)
14.TP-7 (29.2%)
15.TP-14 (39.5%)

Execution Time

1. TP-14 (93%)
2. TP-12 (93%)
3. TP-8 (94%)
4. TP-7 (94%)
5. TP-10 (95%)
6. Defero-16 (95%)
7. Defero-20 (95%)
8. Defero-12 (97%)
9. Defero-8 (97%)
10.Medius-22 (99%)
11.dlmalloc (100%)
12.Medius-20 (100)
13.PHKmalloc (119)
14.Medius-16 (190)
15.Medius-12 (890)

Fast Allocation Speed, Low Memory Fragmentation, and High Spatial Locality: Pick Two.

Allocation Speed (instr)

1. TP-12
2. TP-10
3. TP-8
4. TP-7
5. Defero-20
6. TP-14
7. Defero-16
8. Medius-22
9. Medius-20
10. dlmalloc
11. Defero-12
12. Defero-8
13. PHKmalloc
14. Medius-16
15. Medius-12

Locality (page accuracy)

1. Medius-12
2. Defero-12
3. Defero-8
4. TP-12
5. TP-14
6. TP-8
7. PHKmalloc
8. TP-10
9. dlmalloc
10. Defero-16
11. TP-7
12. Medius-16
13. Medius-20
14. Defero-20
15. Medius-22

Fragmentation

1. dlmalloc
2. TP-12
3. Defero-20
4. Defero-16
5. Defero-12
6. Defero-8
7. Medius-22
8. Medius-20
9. Medius-16
10. PHKmalloc
11. TP-10
12. TP-8
13. Medius-12
14. TP-7
15. TP-14

Execution Time

1. TP-14
2. TP-12
3. TP-8
4. TP-7
5. TP-10
6. Defero-16
7. Defero-20
8. Defero-12
9. Defero-8
10. Medius-22
11. dlmalloc
12. Medius-20
13. PHKmalloc
14. Medius-16
15. Medius-12

**Fast Allocation Speed,
Low Memory Fragmentation,
and High Spatial Locality:
Pick Two.**

Allocation Speed (instr)

1. TP-12
2. TP-10
3. TP-8
4. TP-7
5. Defero-20
6. TP-14
7. Defero-16
8. Medius-22
9. Medius-20
10. dlmalloc
11. Defero-12
12. Defero-8
13. PHKmalloc
14. Medius-16
15. Medius-12

Locality (page accuracy)

1. Medius-12
2. Defero-12
3. Defero-8
4. TP-12
5. TP-14
6. TP-8
7. PHKmalloc
8. TP-10
9. dlmalloc
10. Defero-16
11. TP-7
12. Medius-16
13. Medius-20
14. Defero-20
15. Medius-22

Fragmentation

1. dlmalloc
2. TP-12
3. Defero-20
4. Defero-16
5. Defero-12
6. Defero-8
7. Medius-22
8. Medius-20
9. Medius-16
10. PHKmalloc
11. TP-10
12. TP-8
13. Medius-12
14. TP-7
15. TP-14

Execution Time

1. TP-14
2. TP-12
3. TP-8
4. TP-7
5. TP-10
6. Defero-16
7. Defero-20
8. Defero-12
9. Defero-8
10. Medius-22
11. dlmalloc
12. Medius-20
13. PHKmalloc
14. Medius-16
15. Medius-12

**Fast Allocation Speed,
Low Memory Fragmentation,
and High Spatial Locality:
Pick Two.**

Allocation Speed (instr)

1. TP-12
2. TP-10
3. TP-8
4. TP-7
5. Defero-20
6. TP-14
7. Defero-16
8. Medius-22
9. Medius-20
10. dlmalloc
11. Defero-12
12. Defero-8
13. PHKmalloc
14. Medius-16
15. Medius-12

Locality (page accuracy)

1. Medius-12
2. Defero-12
3. Defero-8
4. TP-12
5. TP-14
6. TP-8
7. PHKmalloc
8. TP-10
9. dlmalloc
10. Defero-16
11. TP-7
12. Medius-16
13. Medius-20
14. Defero-20
15. Medius-22

Fragmentation

1. dlmalloc
2. TP-12
3. Defero-20
4. Defero-16
5. Defero-12
6. Defero-8
7. Medius-22
8. Medius-20
9. Medius-16
10. PHKmalloc
11. TP-10
12. TP-8
13. Medius-12
14. TP-7
15. TP-14

Execution Time

1. TP-14
2. TP-12
3. TP-8
4. TP-7
5. TP-10
6. Defero-16
7. Defero-20
8. Defero-12
9. Defero-8
10. Medius-22
11. dlmalloc
12. Medius-20
13. PHKmalloc
14. Medius-16
15. Medius-12

Fast Allocation Speed,
Low Memory Fragmentation,
and High Spatial Locality:
Pick Two.

Speed	Locality (page accuracy)	Fragmentation	Execution Time
1. TP-12	1. Medius-12	1. dlmalloc	1. TP-14
2. TP-10	2. Defero-12	2.TP-12	2. TP-12
3.TP-8	3. Defero-8	3.Defero-20	3. TP-8
4.TP-7	4. TP-12	4.Defero-16	4. TP-7
5.Defero-20	5. TP-14	5.Defero-12	5. TP-10
6. TP-14	6. TP-8	6.Defero-8	6. Defero-16
7.Defero-16	7. PHKmalloc	7.Medius-22	7. Defero-20
8.Medius-22	8. TP-10	8.Medius-20	8. Defero-12
9.Medius-20	9. dlmalloc	9.Medius-16	9. Defero-8
10.dlmalloc	10.Defero-16	10.PHKmalloc	10.Medius-22
11.Defero-12	11.TP-7	11.TP-10	11.dlmalloc
12.Defero-8	12.Medius-16	12.TP-8	12.Medius-20
13.PHKmalloc	13.Medius-20	13.Medius-12	13.PHKmalloc
14.Medius-16	14.Defero-20	14.TP-7	14.Medius-16
15.Medius-12	15.Medius-22	15.TP-14	15.Medius-12

Fast Allocation Speed,
Low Memory Fragmentation,
and High Spatial Locality:
Pick Two.

Allocation Speed (instr)	Locality (page accuracy)	Fragmentation	Execution Time
1.TP-12	1. Medius-12	1. dlmalloc	1. TP-14
2.TP-10	2. Defero-12	2.TP-12	2. TP-12
3.TP-8	3. Defero-8	3.Defero-20	3. TP-8
4.TP-7	4. TP-12	4.Defero-16	4. TP-7
5.Defero-20	5. TP-14	5.Defero-12	5. TP-10
6.TP-14	6. TP-8	6.Defero-8	6. Defero-16
7.Defero-16	7. PHKmalloc	7.Medius-22	7. Defero-20
8.Medius-22	8. TP-10	8.Medius-20	8. Defero-12
9.Medius-20	9. dlmalloc	9.Medius-16	9. Defero-8
10.dlmalloc	10.Defero-16	10.PHKmalloc	10.Medius-22
11.Defero-12	11.TP-7	11.TP-10	11.dlmalloc
12.Defero-8	12.Medius-16	12.TP-8	12.Medius-20
13.PHKmalloc	13.Medius-20	13.Medius-12	13.PHKmalloc
14.Medius-16	14.Defero-20	14.TP-7	14.Medius-16
15.Medius-12	15.Medius-22	15.TP-14	15.Medius-12

**Fast Allocation Speed,
Low Memory Fragmentation,
and High Spatial Locality:
Pick Two.**

Conclusions

- TP is 7% better than state-of-the-art
  - Benefits both temporal and spatial
- Fragmentation, allocation speed, and locality are antagonistic

Thank you