DynaSOAr: A Parallel Memory Allocator for Object-oriented Programming on GPUs with Efficient Memory Access

Matthias Springer, Hidehiko Masuhara
Tokyo Institute of Technology

ECOOP 2019

Introduction / Motivation

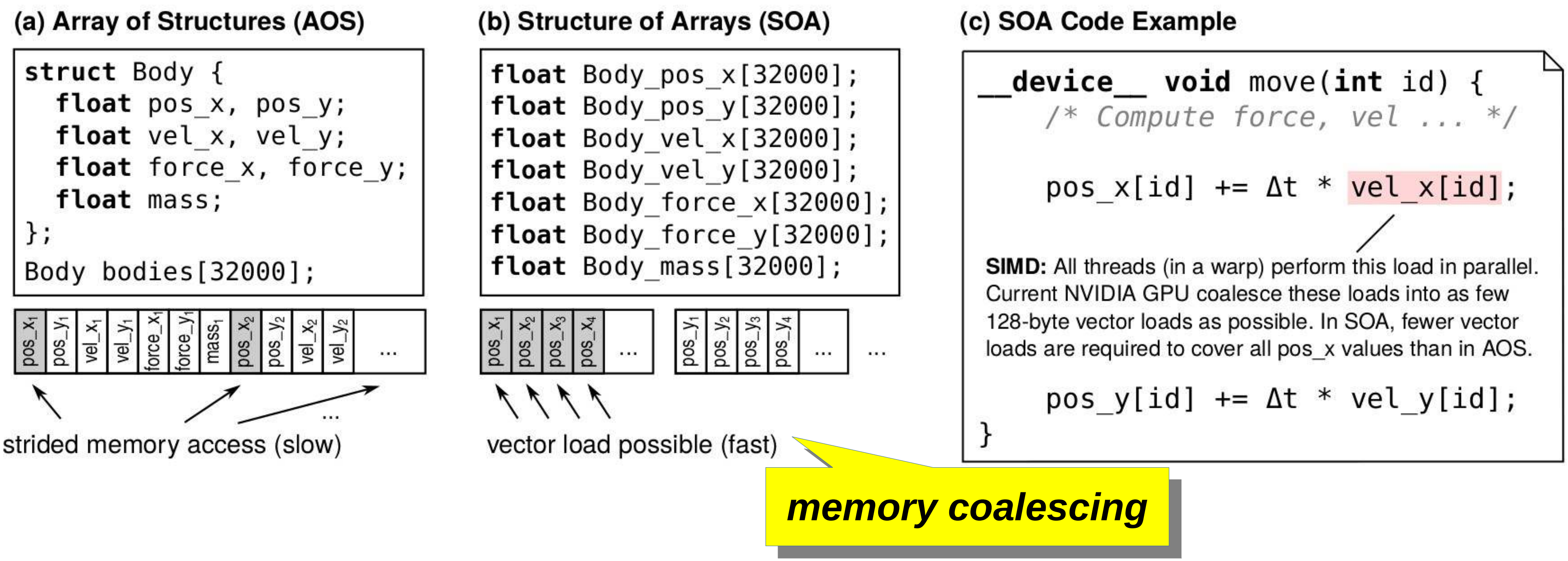
東京工業大学
Tokyo Institute of Technology

- *Goal:* Make GPU programming **easier to use**.
- *Focus:* **Object-oriented programming** (OOP) on GPUs/CUDA.
  - Many OOP applications in high-performance computing (HPC).
  - **Dynamic memory allocation** is highly useful in OOP.
- *This work:* **DynaSOAr**, a lock-free dynamic memory allocator for `struct`-ured data, based on hierarchical bitmaps.

東京工業大学
Tokyo Institute of Technology

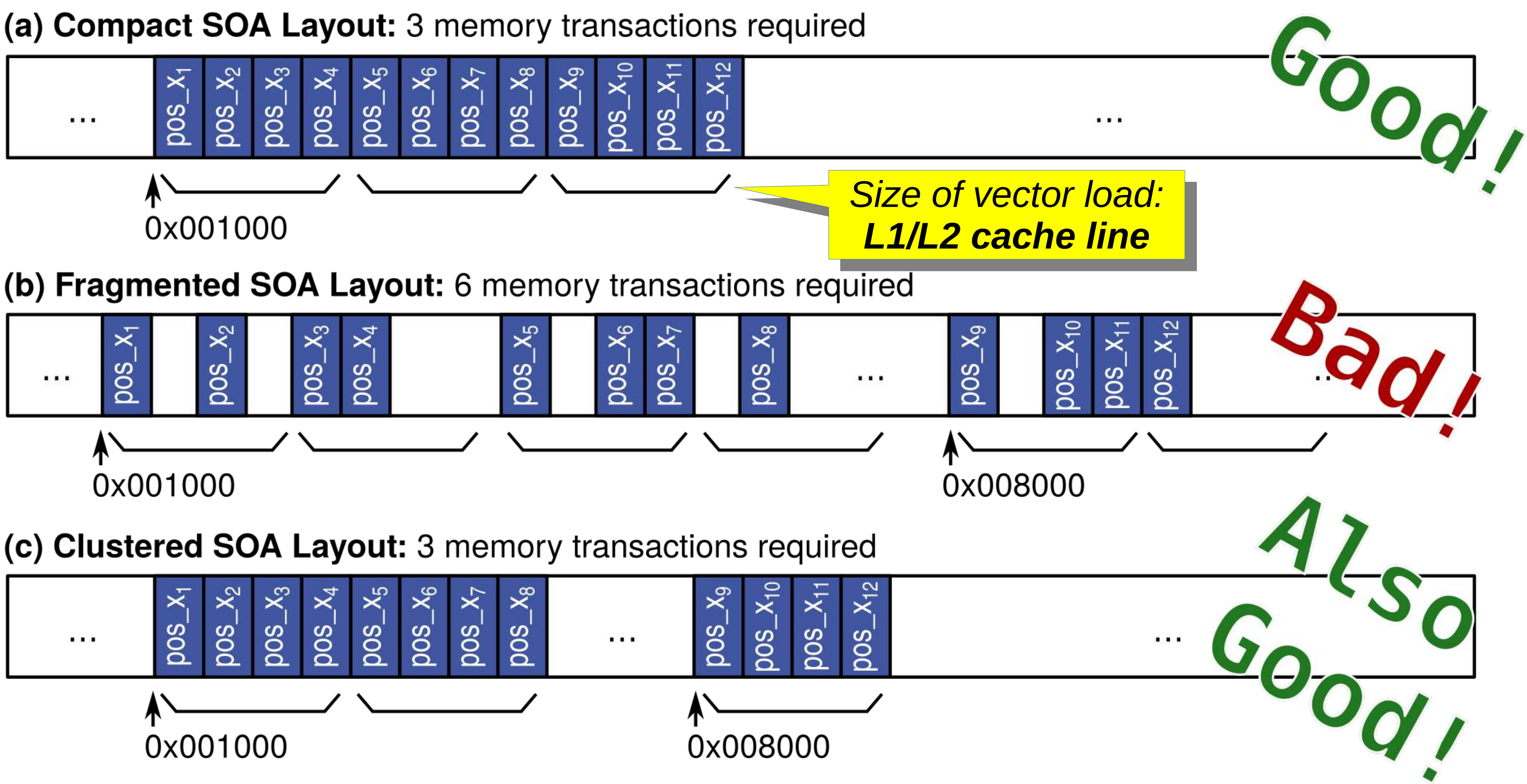
Background / Design Requirements

Memory Access Performance



- **SOA:** Structure of Arrays data layout.
- A **best practice** for SIMD/GPU programmers.
- The main optimization of DynaSOAr.

Fragmentation



For illustration purposes: Vector length 32 byte (4 scalars) instead of 128 byte (32 scalars). N-body sim.

Requirements

東京工業大学
Tokyo Institute of Technology

- Allocations in **SOA data layout**.
- **Low fragmentation** (high frag. makes SOA less efficient).
  - Other allocators [1, 2] use **hashing**. This leads to high fragmentation / less dense allocations.
  - Without hashing, *raw allocation* in DynaSOAr will surely be **slower than** in **other allocators**. But we can make up for it with better memory access performance.
- Efficient heap usage: **Low allocator overhead**.
- **No locking:** Can easily deadlock on GPUs.

[1] M. Steinberger, et. al. **ScatterAlloc: Massively Parallel Dynamic Memory Allocation for the GPU**. In: InPar 2012.
[2] A. Adinetz, D. Pleiter. **Halloc: A High-Throughput Dynamic Memory Allocator for GPGPU Architectures**. In: GTC 2014.

東京工業大学
Tokyo Institute of Technology

Overview of DynaSOAr

DynaSOAr Components

東京工業大学
Tokyo Institute of Technology

- DynaSOAr achieves superior memory access performance by controlling both **memory layout** and **memory access patterns**.
- Traditional memory allocators control only memory layout.

Memory Allocator	SOA Data Layout DSL [3]	Parallel Do-all Operation
• Memory allocator for **structured data**. • Allocates data in **SOA** data layout. • **Lock-free** design.	• **Language-level** component. • Hides custom data layout from programmers. • Could be also be implemented at the compiler level.	• SOA by itself is not sufficient. • SOA improves performance only with certain (**coalesceable**) memory access patterns. • One such pattern: **SMMO**

[3] M. Springer, H. Masuhara. **Ikra-Cpp: A C++/CUDA DSL for Object-Oriented Programming with Structure-of-Arrays Layout**. In: WPMVP 2018.

東京工業大学
Tokyo Institute of Technology

DynaSOAr Components

- DynaSOAr achieves superior memory access performance by controlling both **memory layout** and **memory access patterns**.
- Traditional memory allocators control only memory layout.

Memory Allocator

- Memory allocator for **structured data**.
- Allocates data in **SOA** data layout.
- **Lock-free** design.

SOA Data Layout DSL [3]

- **Language-level** component.
- H
la
p
- C
in
compiler level.

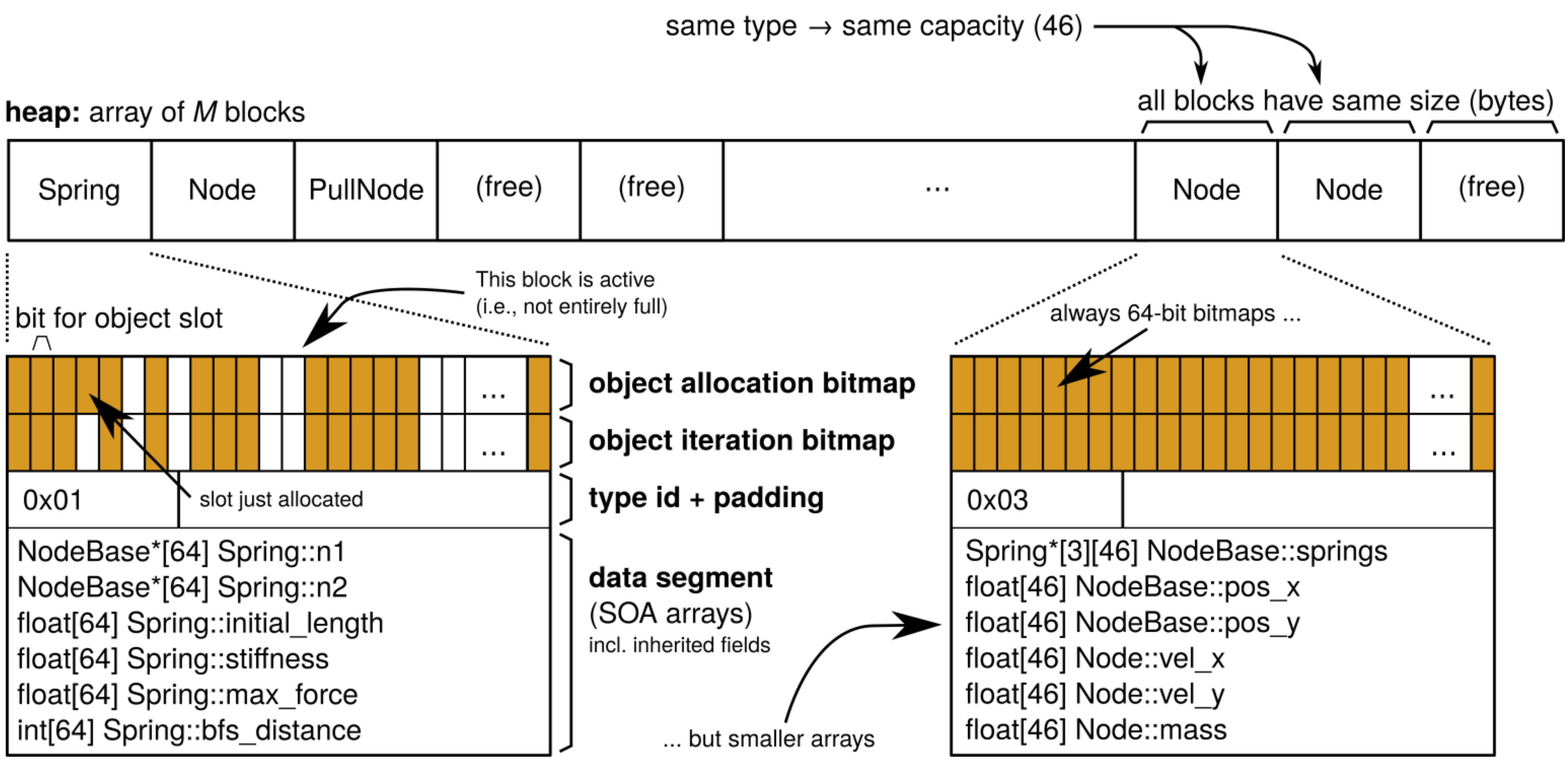
Parallel Do-all Operation

- SOA by itself is not sufficient.
- proves
ance only with
coalesceable)
access
.
- e such pattern: **SMMO**

Single-Method Multiple-Objects: *Run same method for all objects of a type. Let DynaSOAr take care of object-to-thread assignment.*

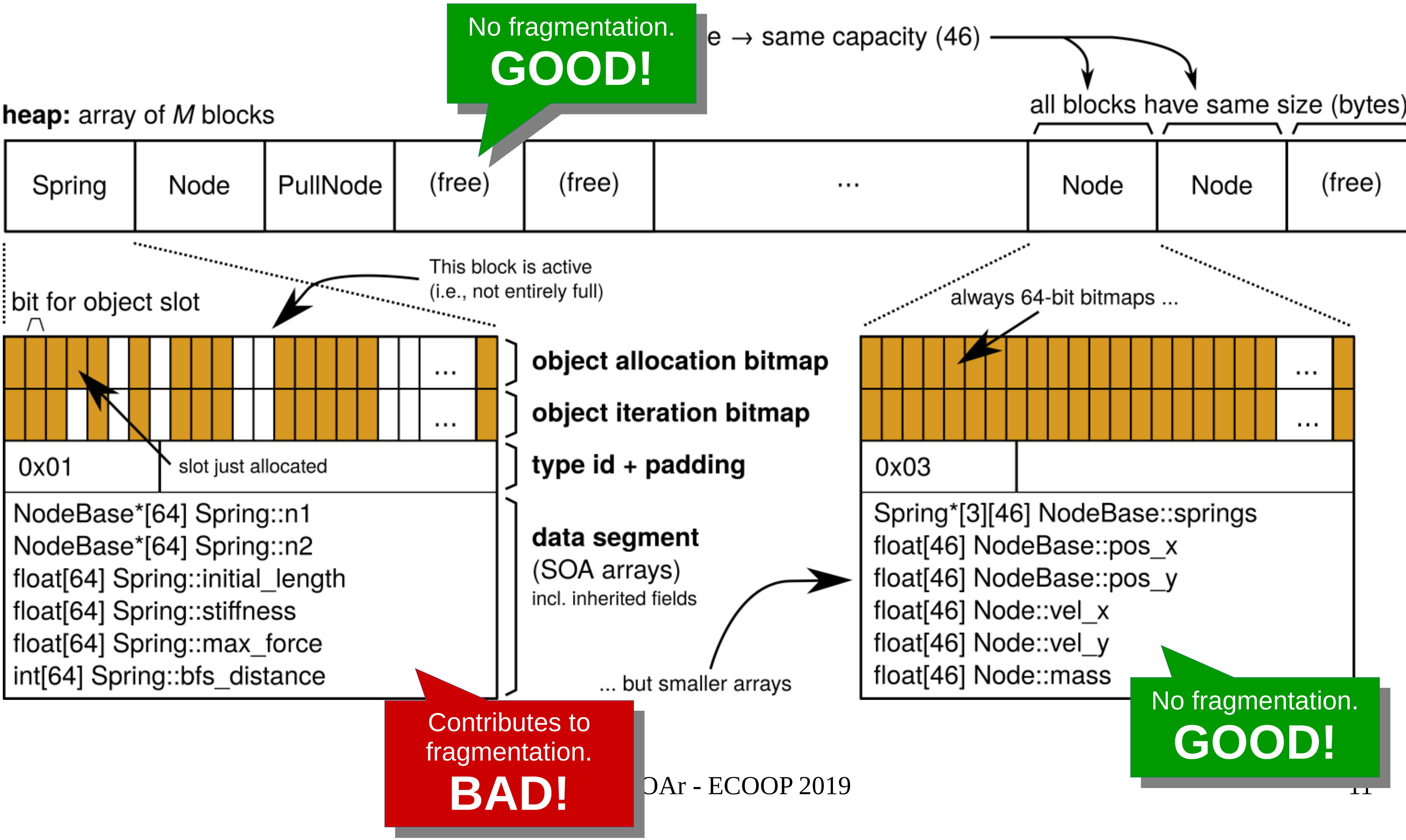
[3] M. Springer, H. Masuhara. **Ikra-Cpp: A C++/CUDA DSL for Object-Oriented Programming with Structure-of-Arrays Layout**. In: WPMVP 2018.

DynaSOAr Heap Layout

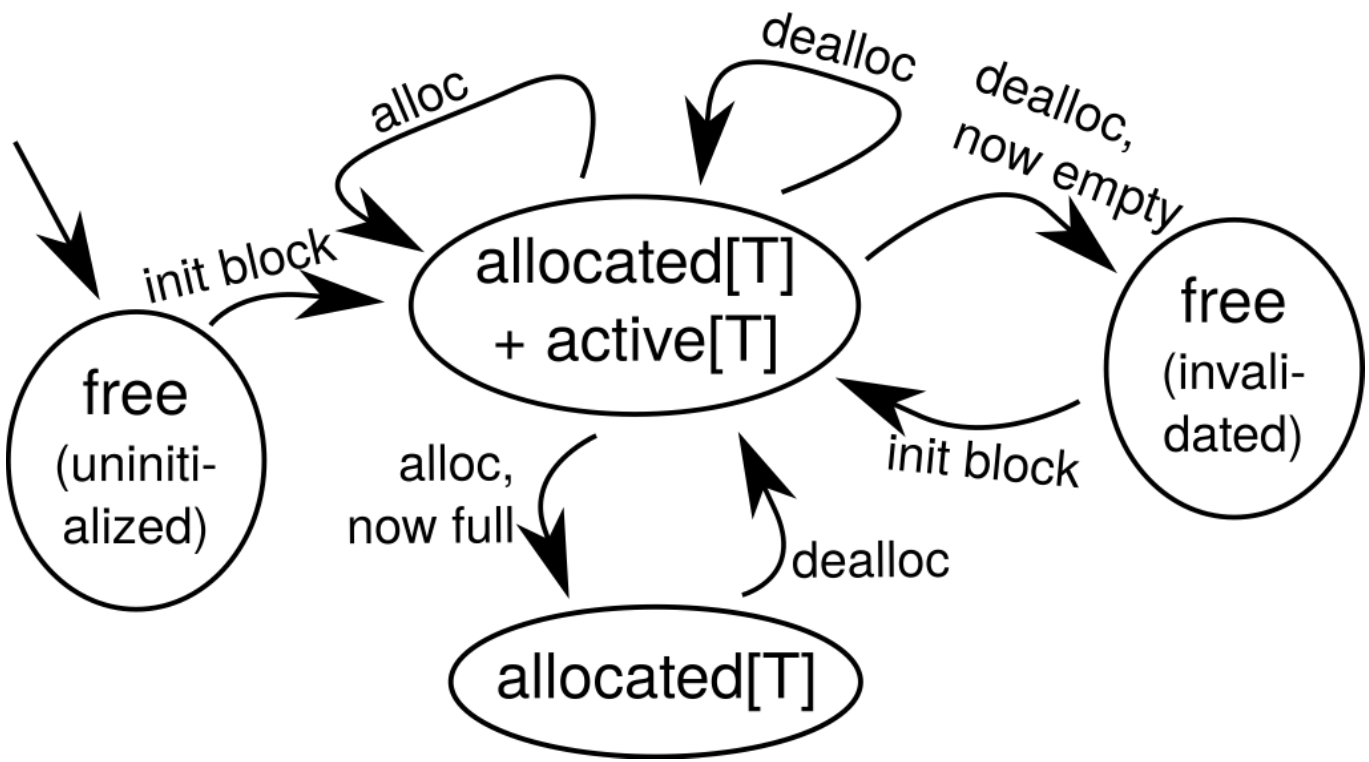


DynaSOAr Heap Layout

東京工業大学
Tokyo Institute of Technology



Block Multi-States



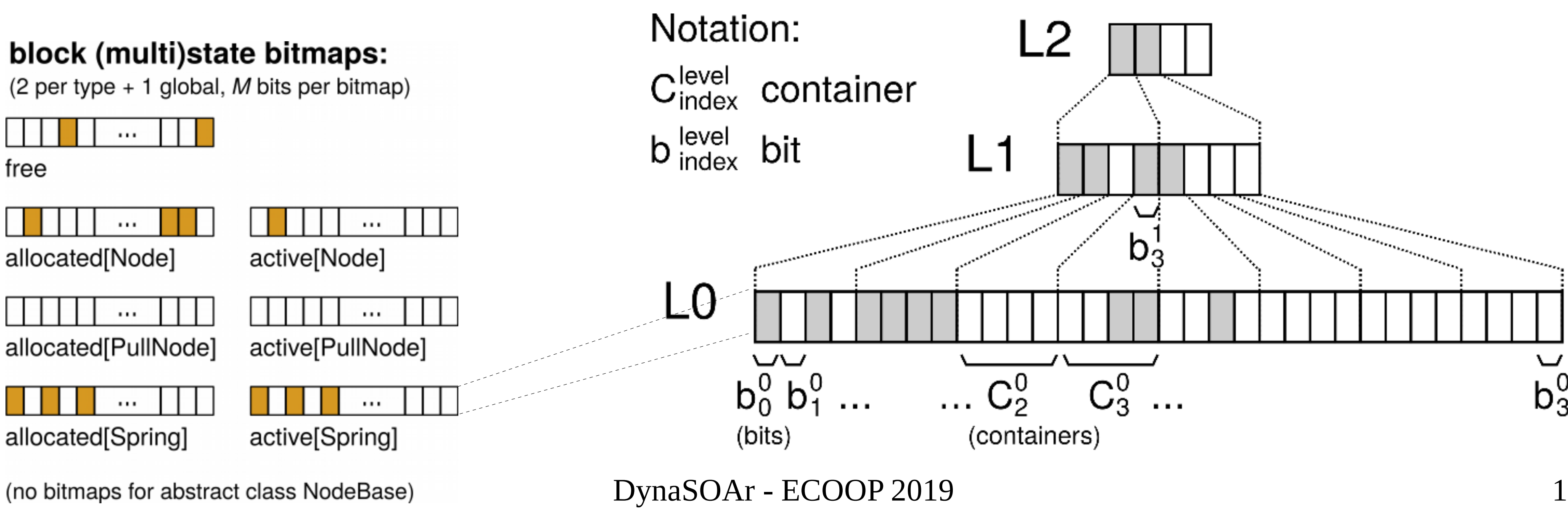
Allocation Algorithm

東京工業大学
Tokyo Institute of Technology

- To keep **fragmentation low**: Always allocate objects in *active[T]* blocks.
- Only if no *active[T]* block exists, initialize a new *active[T]* block from a *free* block.
- Reserve object slots within a block with **atomic operations** (atomically set bit to 1).

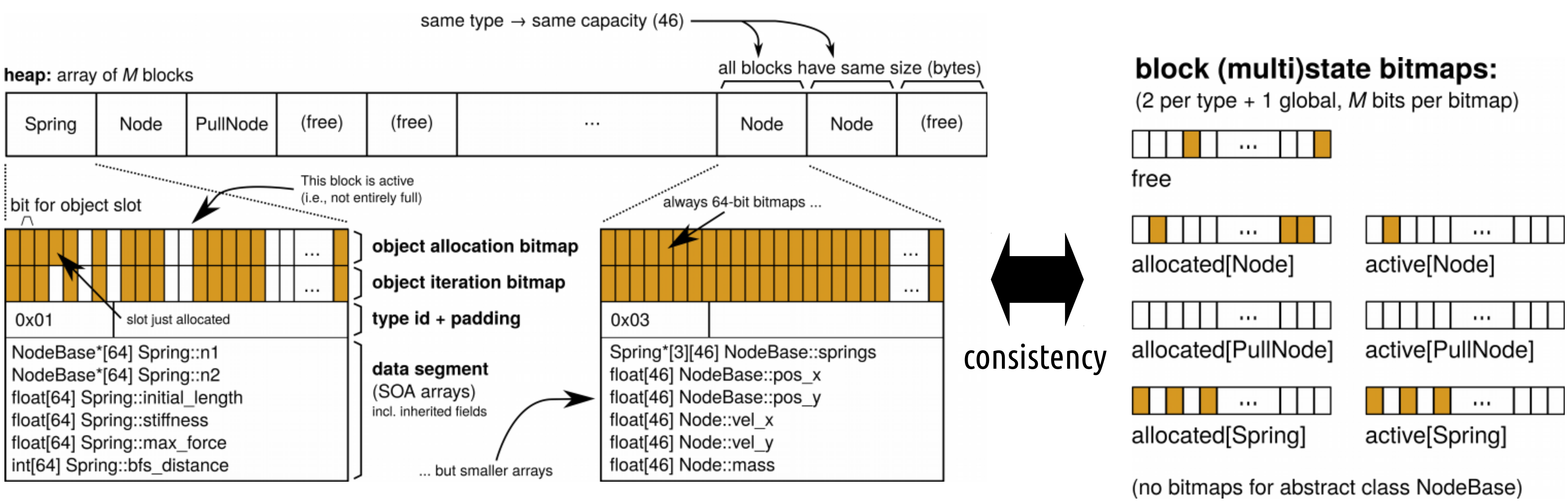
How to Find *active[T]/free* Blocks?

- We **index** block multi-states with bitmaps.
  - Can find blocks by scanning a bitmap.
  - Bitmaps are **hierarchical**: Find set bits with a **logarithmic** order of accesses.
  - #bitmaps depends on #classes.



Challenges

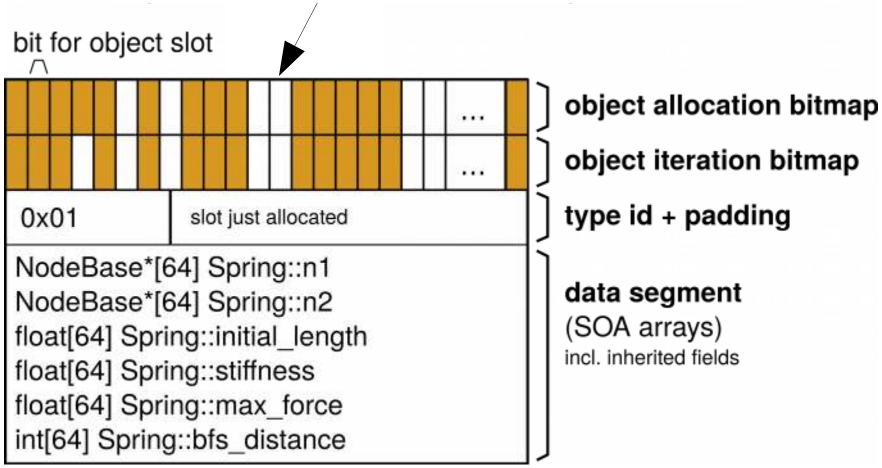
- **Eventual consistency** of data structures.
  - Block multi-states ↔ Block multi-state bitmaps (indices)
  - Different levels of block multi-state bitmap hierarchy
  - Algorithms must be able to handle temporary inconsistencies: **Optimistic**, rollback if inconsistency detected.



Challenges

- **Eventual consistency** of data structures.
  - Block multi-states ↔ Block multi-state bitmaps (indices)
  - Different levels of block multi-state bitmap hierarchy
  - Algorithms must be able to handle temporary inconsistencies: **Optimistic**, rollback if inconsistency detected.
- Reduce **allocation contention**: Multiple threads trying to allocate the same memory location. (Only one can succeed.)

e.g.: 2 threads trying to allocate in this object slot

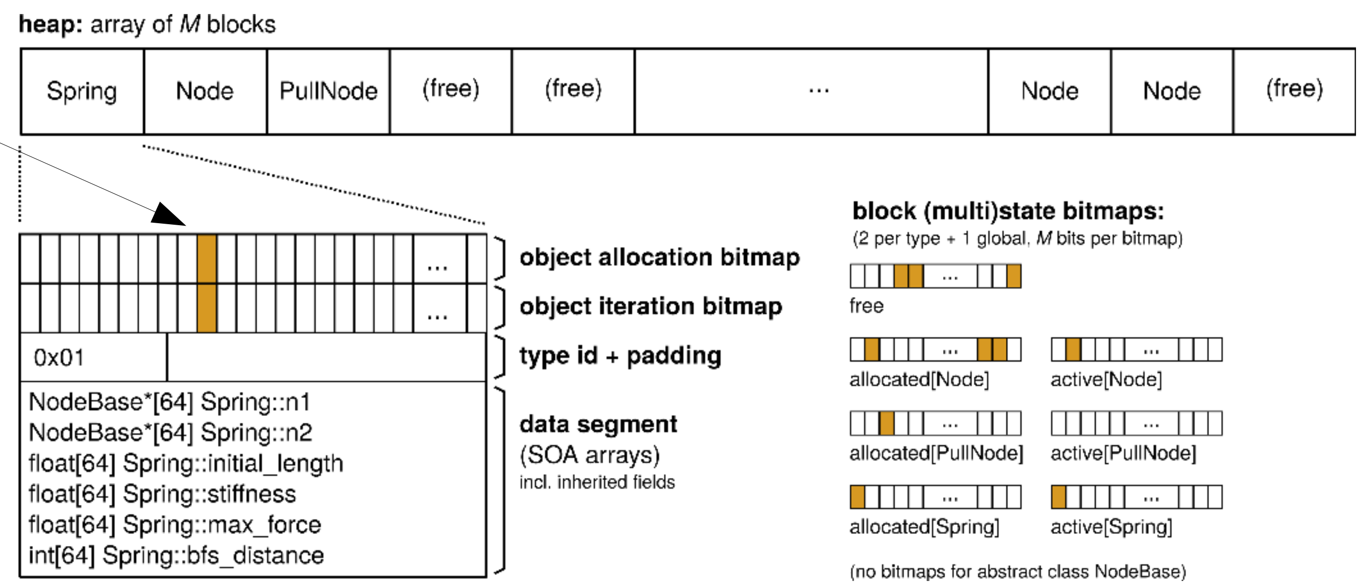


東京工業大学
Tokyo Institute of Technology

Challenges

Thread deallocated last object in block.

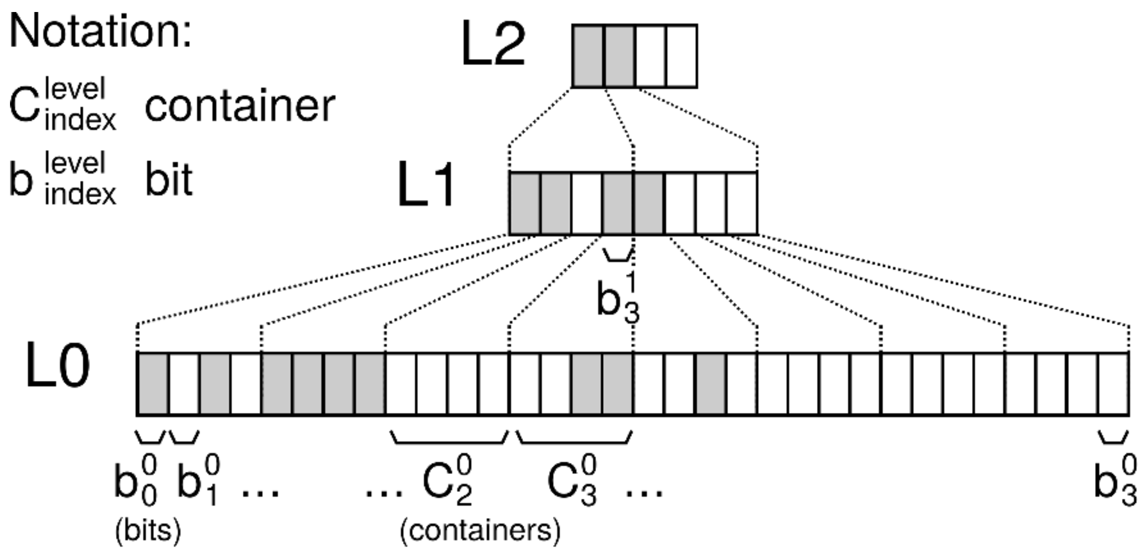
Can I delete this block now?
(I.e., reset type ID and put back in *free* bitmap)



- **Safe memory reclamation**: When is it safe to delete a block?
  - All blocks have **same structure**: All blocks have the same byte size. Object allocation bitmaps are always located at the same offset.
  - **Block invalidation**: Atomically set all bits to 1. Block seems full to other threads and no allocation can succeed. Then it is safe to delete.

Optimizations

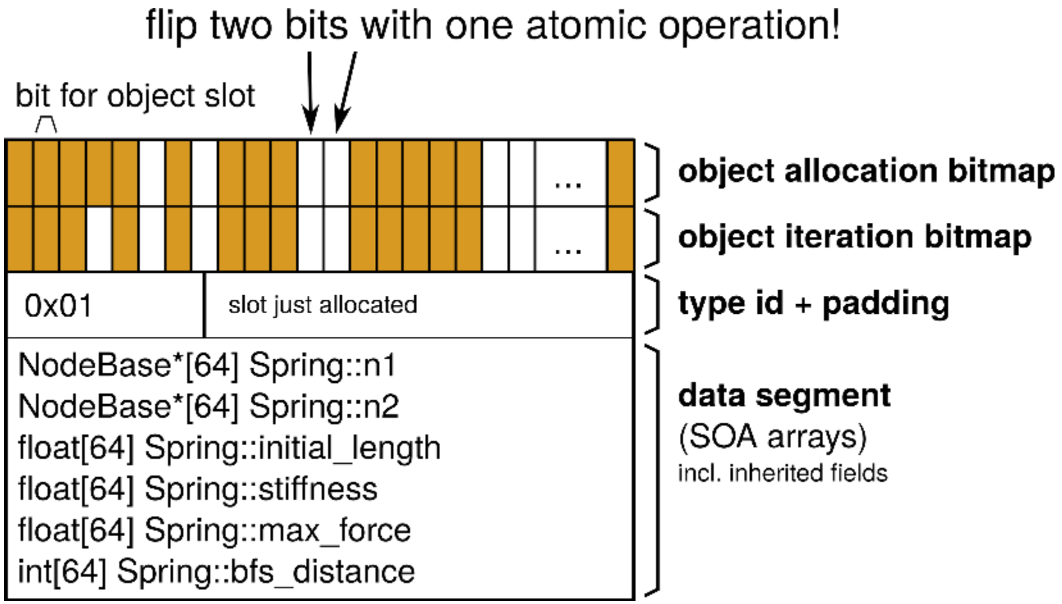
- **Hierarchical Bitmaps**.



Optimizations

東京工業大学
Tokyo Institute of Technology

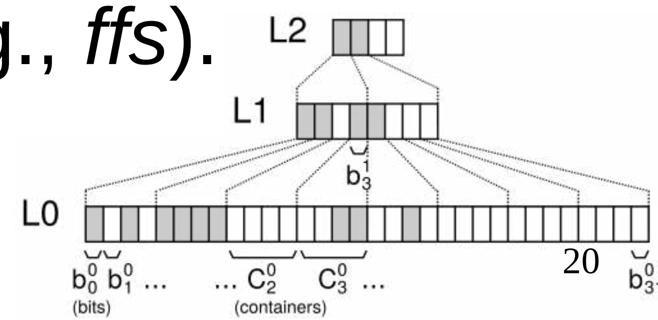
- **Hierarchical Bitmaps**.
- **Allocation Request Coalescing** [4]: A **leader** thread reserves object slots **on behalf of all allocating threads** in a warp.



[4] X. Huang, et. al. **XMalloc: A Scalable Lock-free Dynamic Memory Allocator for Many-core Machines**. In: CIT 2010.

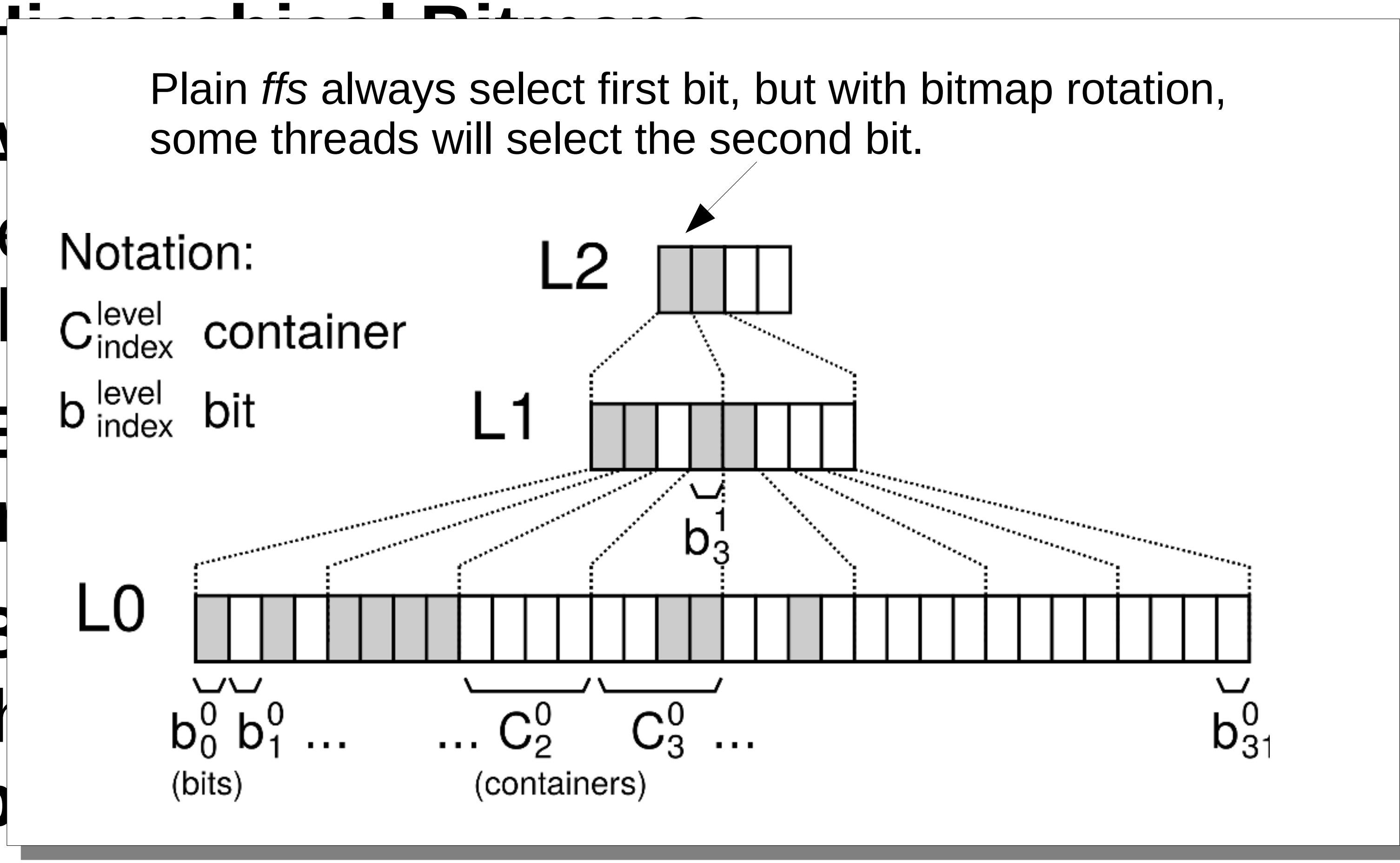
Optimizations

- **Hierarchical Bitmaps**.
- **Allocation Request Coalescing**: A **leader** thread reserves object slots **on behalf of all allocating threads** in a warp.
- **Efficient Bit Operations**: Utilize bit-level **integer intrinsics** (e.g., *ffs*).
- **Bitmap Rotation**: To reduce the probability of threads choosing the same bit, **rotate-shift bitmaps** before selecting a bit (e.g., *ffs*).



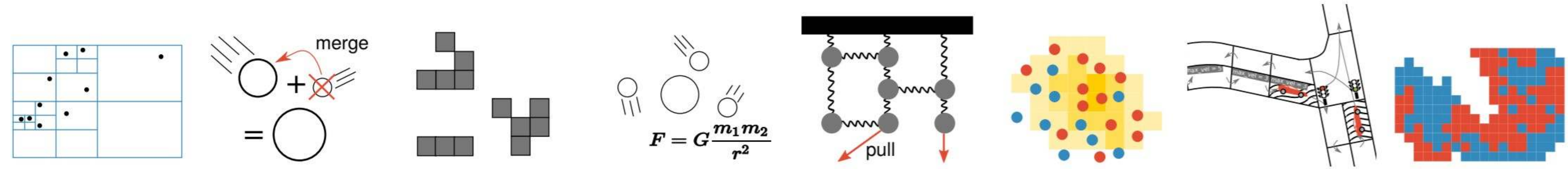
Optimizations

- H
- A … thread re … ating th
- E … teger in
- B … of th b



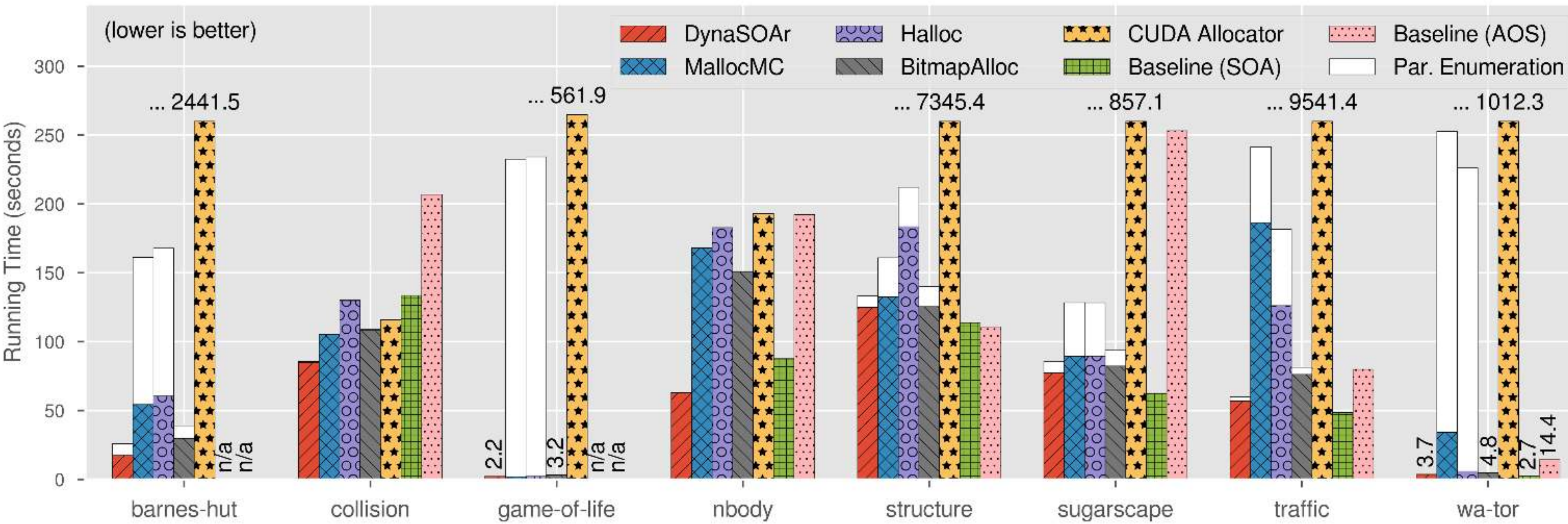
Benchmarks

Benchmark Results



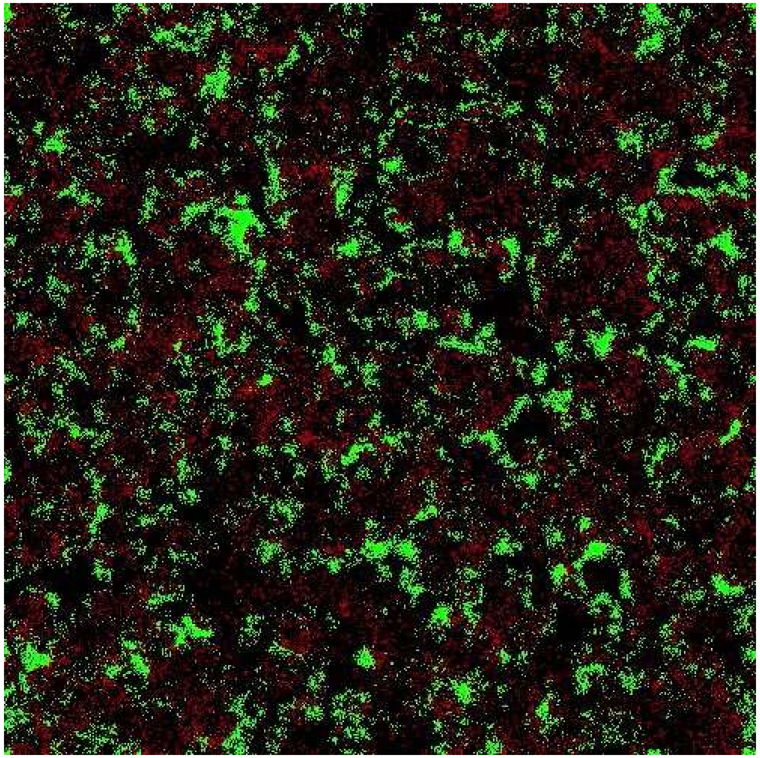
(a) barnes-hut [4]: Parallel Tree Constr. **(b)** collisions: Particle System **(c)** game-of-life: Cellular Automaton **(d)** nbody: Particle System **(e)** structure [14]: Finite Elem. Method **(f)** sugarscape [8]: Agent-based Sim. **(g)** traffic [17]: Nagel-Schr. Model **(h)** wator [6]: Agent-based Sim.

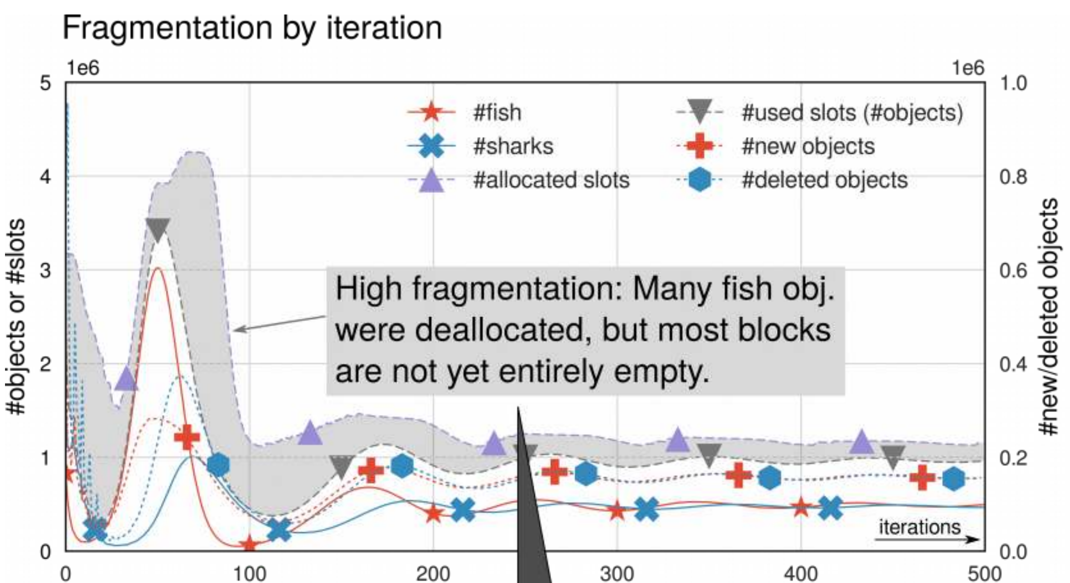
(These are all SMMO applications [ECOOP Artifact])



Benchmark Results

- wa-tor: Fish-and-Shark simulation (predatory/prey ecosystem)
- Objects: `Fish`, `Shark`, `Cell`





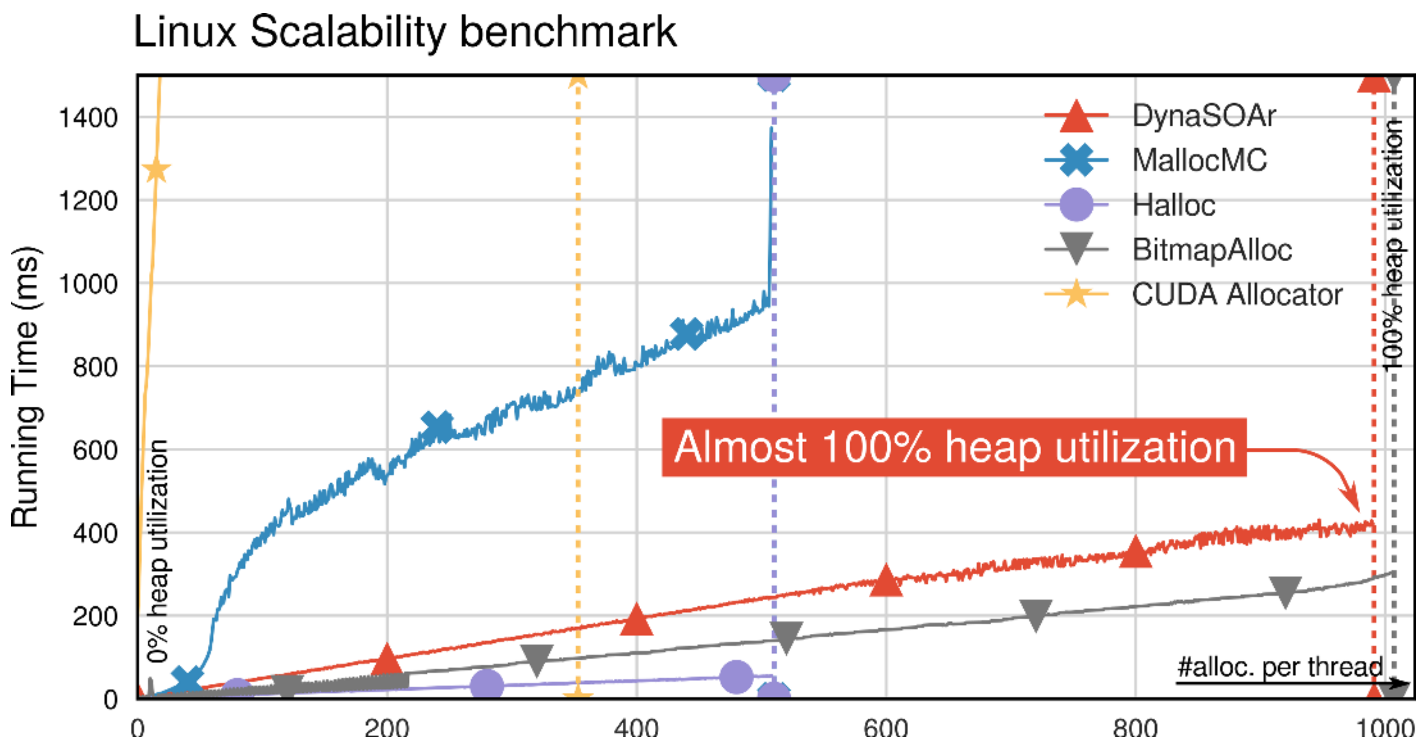
$$F = \frac{1}{\#\text{Blocks}} \sum_{b \in Blocks} \frac{\#\text{free slots}(b)}{\#\text{slots}(b)}$$

(only allocated blocks)

Fragmentation can be eliminated with memory defragmentation [5].
[5] M. Springer, H. Masuhara. Massively Parallel GPU Memory Compaction. ISMM 2019.

Linux Scalability Benchmark

東京工業大学
Tokyo Institute of Technology



- Only (de)allocation, does not access memory.
- DynaSOAr can utilize **almost entire heap**.

Conclusion

Conclusion

東京工業大学
Tokyo Institute of Technology

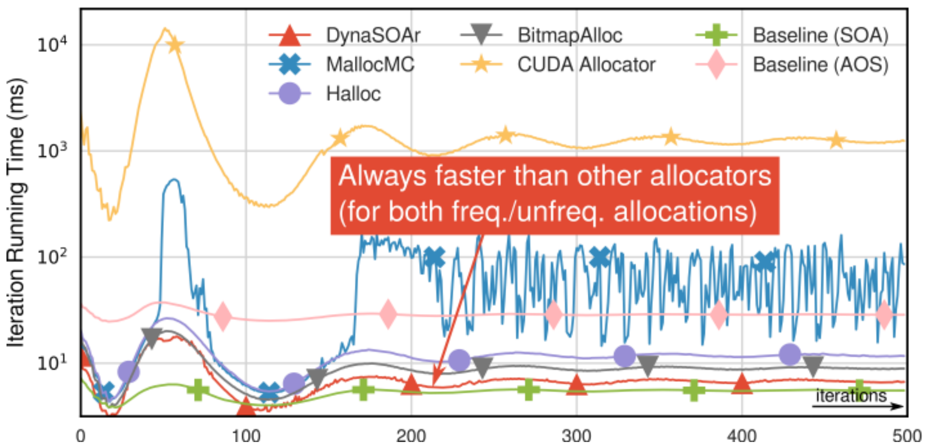
- Optimize not only for raw (de)allocation but also for efficient access of **allocated memory**.
- GPUs/SIMD arch. require special optimizations for better **vectorized access**.
  - For structured data: SOA data layout.
  - Low fragmentation is even more important!
- **Atomic memory operations** became much faster with recent GPU architectures [6].
  → Allows us to reduce fragmentation more aggressively.

[6] A. Gaihre, et. al. XBFS: eXploring Runtime Optimizations for Breadth-First Search on GPUs. In: HPDC 2019.

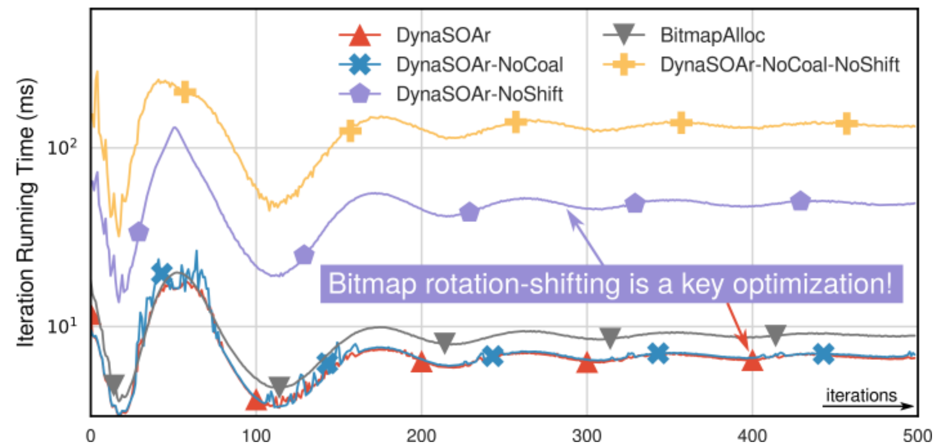
Backup Slides

Pinpointing Source of Speedup

All other graphs: wa-tor benchmark



Performance benefit of specific optimizations



- **Bitmap rotation** is the most important optimization (besides SOA data layout).
- Other allocators reduce allocation contention with hashing, DynaSOAr uses bitmap rotation.