SoaAlloc: Accelerating Single-Method Multiple-Objects Applications on GPUs

Student Research Competition @ SPLASH 2018

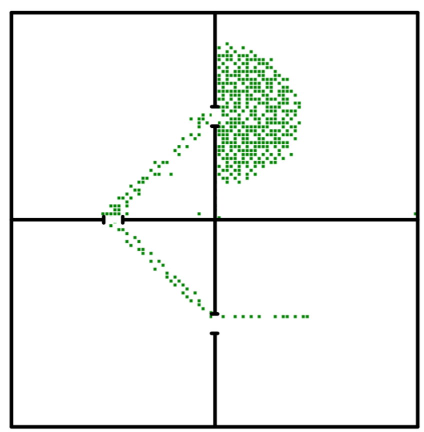
Matthias Springer

Tokyo Institute of Technology

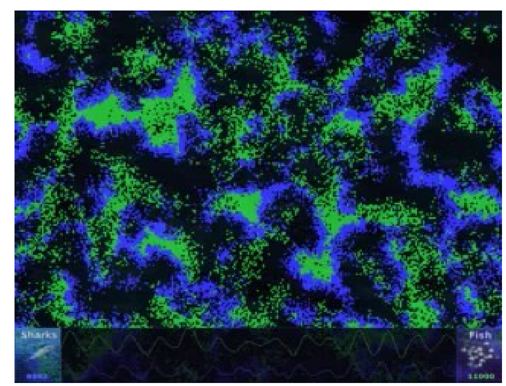
Research Goal: OOP for GPUs

- Fast Object-oriented Programming (OOP) on GPUs
- SIMD-friendly class of OOP applications:
  Single-Method Multiple-Objects (SMMO)
- Many practical SMMO applications in HPC, e.g.:

Traffic Flow Simulation [1]



Evacuation Simulation [2]

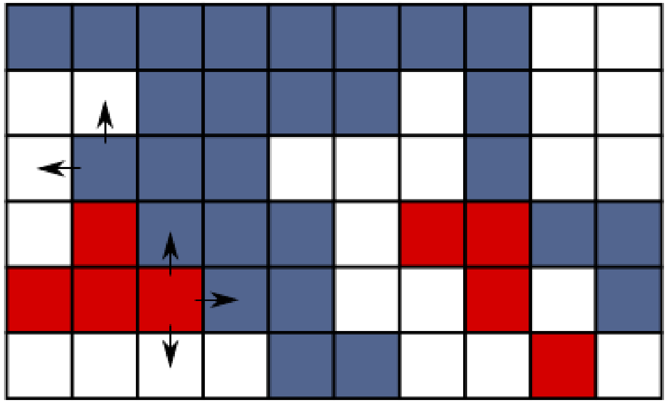


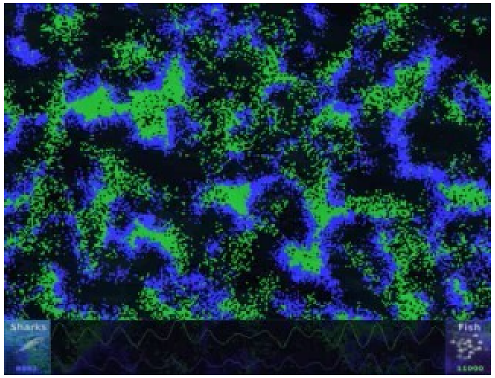
Predator-Prey

[1] D. Strippgen et. al. Mult-agent Traffic Simulation with CUDA. HPCSIM '09.
[2] X. Li et. al. Cloning Agent-based Simulation on GPU. SIGSIM-PADS '15.
Animation: https://en.wikipedia.org/wiki/Wa-Tor

Single-Method Multiple-Objects

- Run same method for all objects of a type
- Running Example: Fish-and-Shark Simulation





- Creating and deleting objects (fish, sharks) all the time!
- Run move() method for all fish and shark objects in parallel

For Good Performance: SOA Data Layout

- Standard optimization on GPUs for good memory bandwidth utilization and better cache performance

```
class Shark {
    float health;
    Cell* position;
    /* ... /*

    void step_health() {
        health = health – 1;
        if (health == 0)
            delete this;
    }
};

Shark sharks[1000];
```

Array of Structures (AOS)

```
float S_health[1000];
Cell* S_position[1000];
```

vector load possible

```
void S_step_health(int id) {
    S_health[id] =
            S_health[id] – 1;
    if (S_health[id] == 0)
        S_destruct(id);
}
```

Structure of Arrays (SOA)

Main Challenges

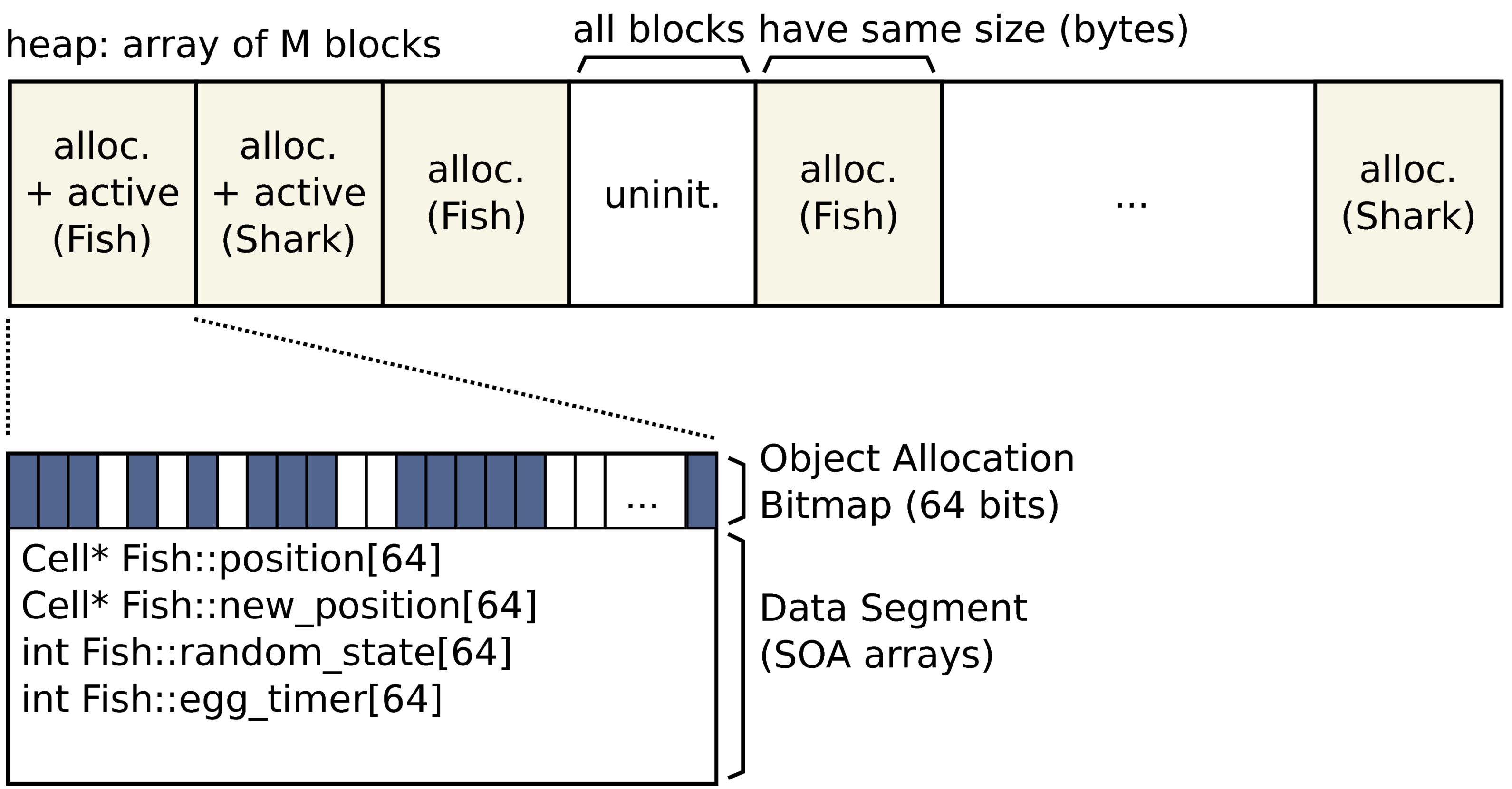
- How to combine dynamic memory allocations with SOA?
- How to keep fragmentation low?
- With thousands of parallel threads, how to implement all of this in a lock-free fashion?
  (Memory allocator runs entirely on the GPU!)
- Allocator Interface:
  `new<T>(), delete<T>(), do_all<T>(func*)`

Based on Ideas from Related Work

- Other GPU memory allocators (e.g., [3]):
  Fast allocations, but slow memory access
- How to represent pointers? E.g.: global references [4]
- C++/CUDA DSLs for SOA data layout [5, 6]

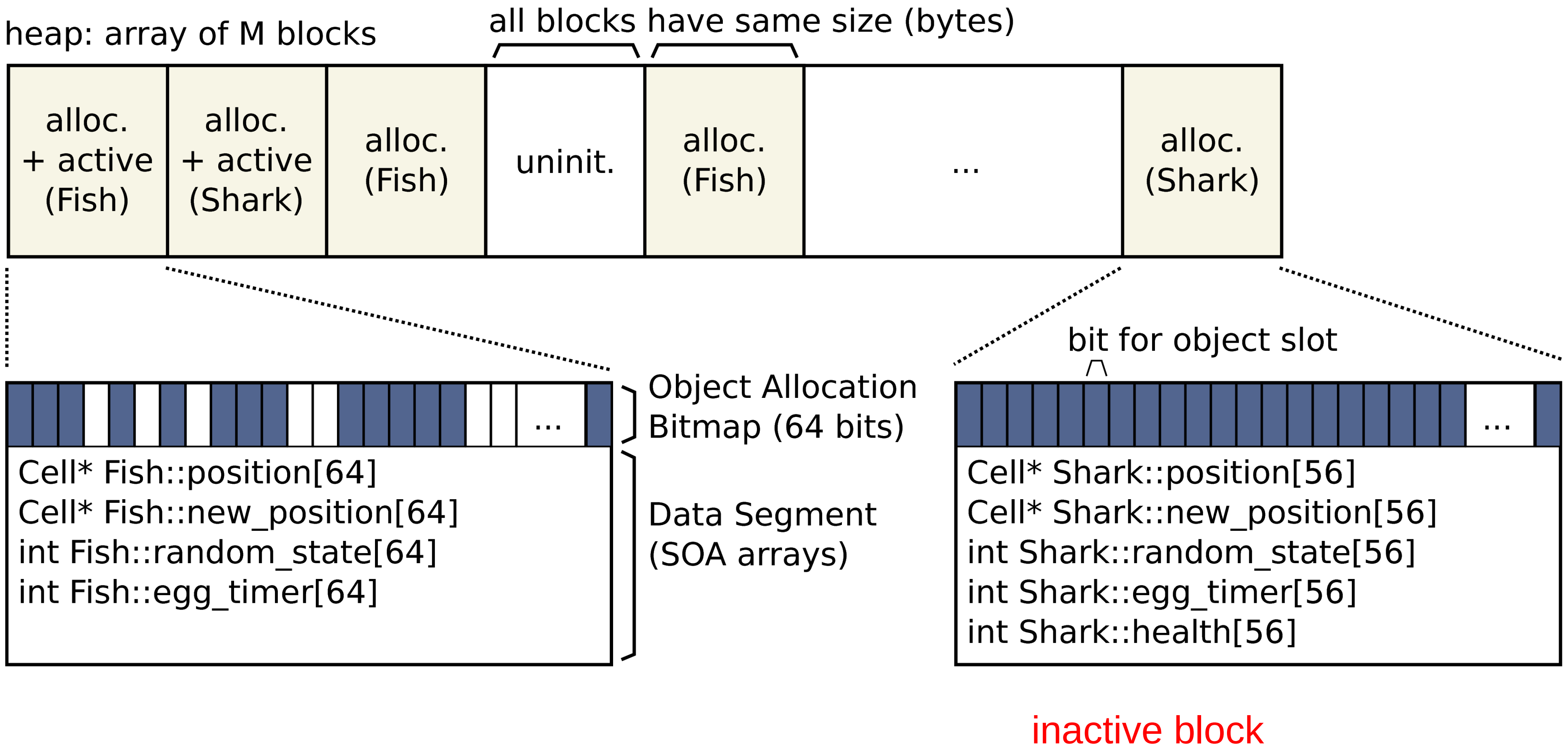
[3] M. Steinberger et al. ScatterAlloc: Massively Parallel Dynamic Memory Allocation for the GPU. InPar 2012.
[4] J. Franco et al. You Can Have It All: Abstraction and Good Cache Performance. Onward! 2017.
[5] R. Strzodka. Abstraction for AoS and SoA Layout in C++. GPU Computing Gems Jade Edition, 2012.
[6] M. Springer et al. Ikra-Cpp: A C++/CUDA DSL for Object-oriented Programming with SOA Layout. WPMVP 2018.

Allocation Data Structure



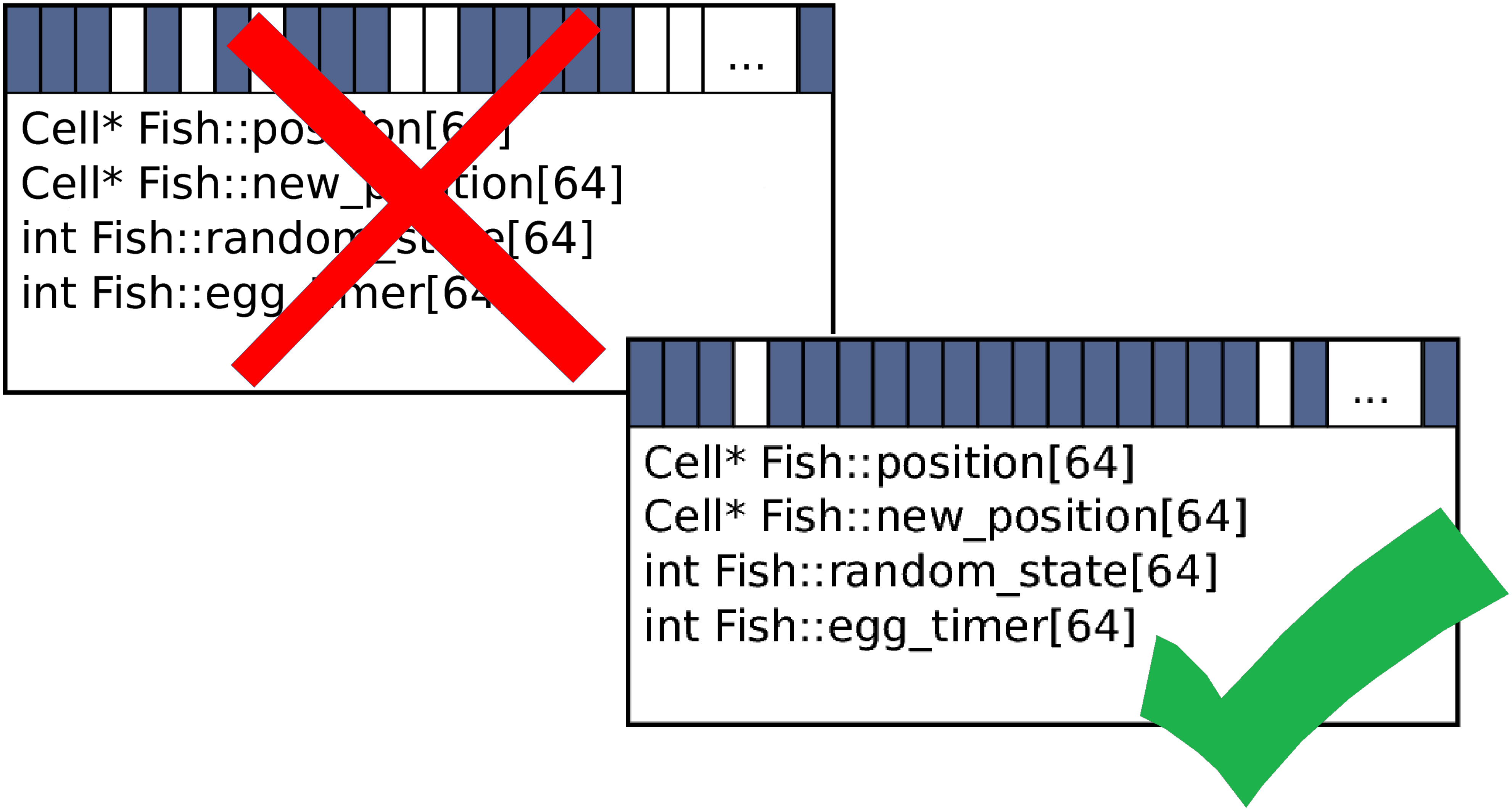
active block

Allocation Data Structure

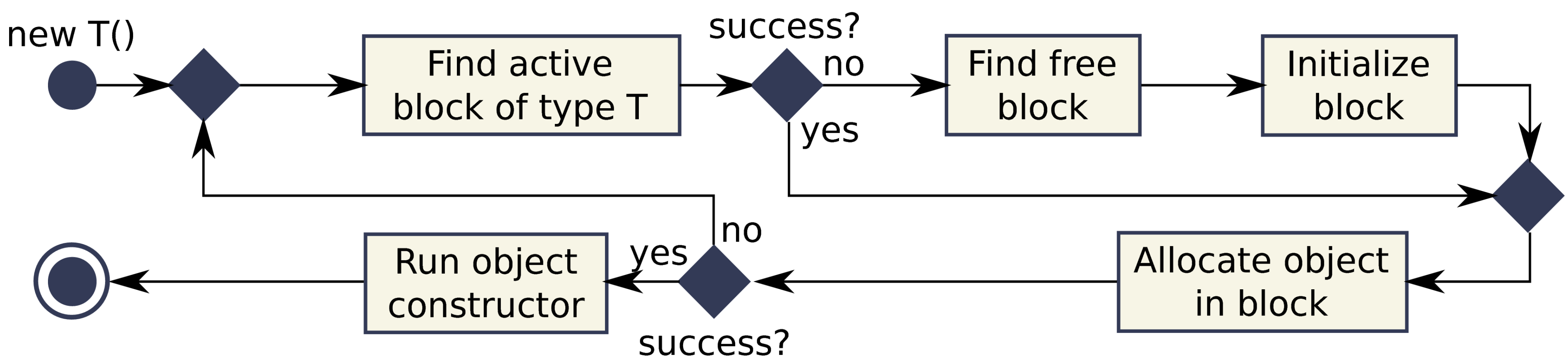


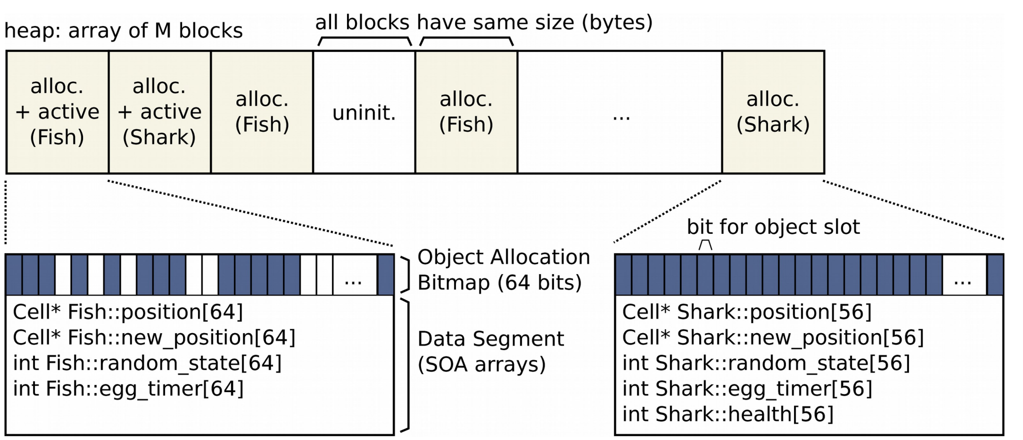
Fragmentation: Lower is Better

Fragmentation: Lower is Better

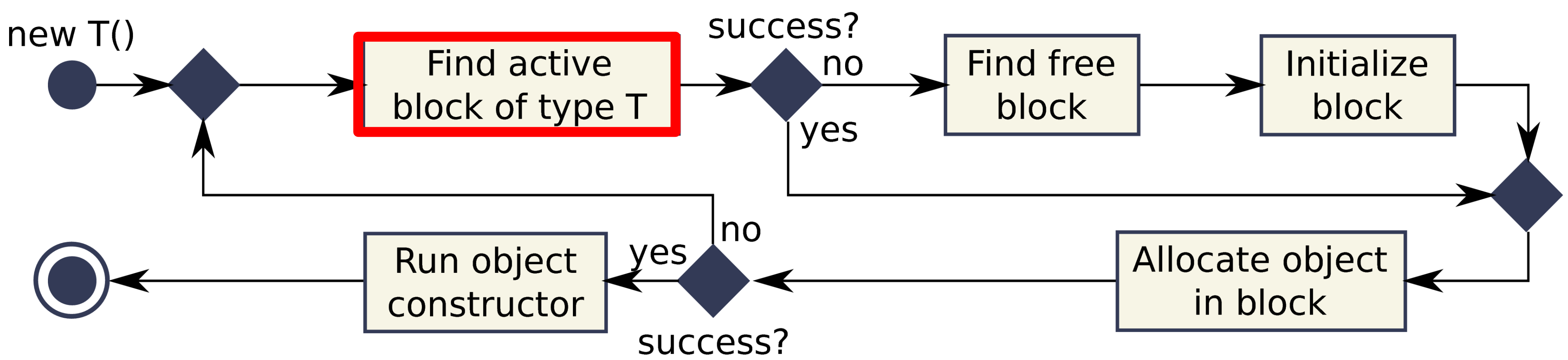


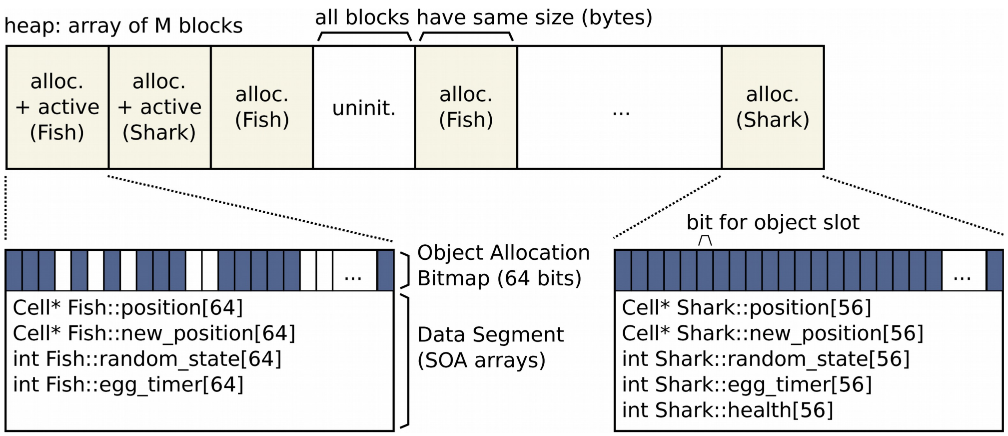
Object Allocation



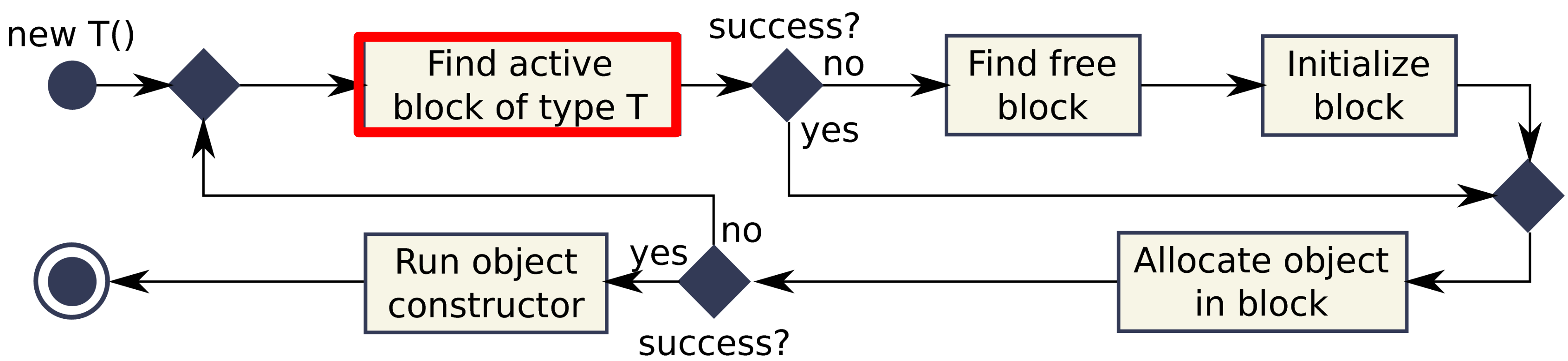


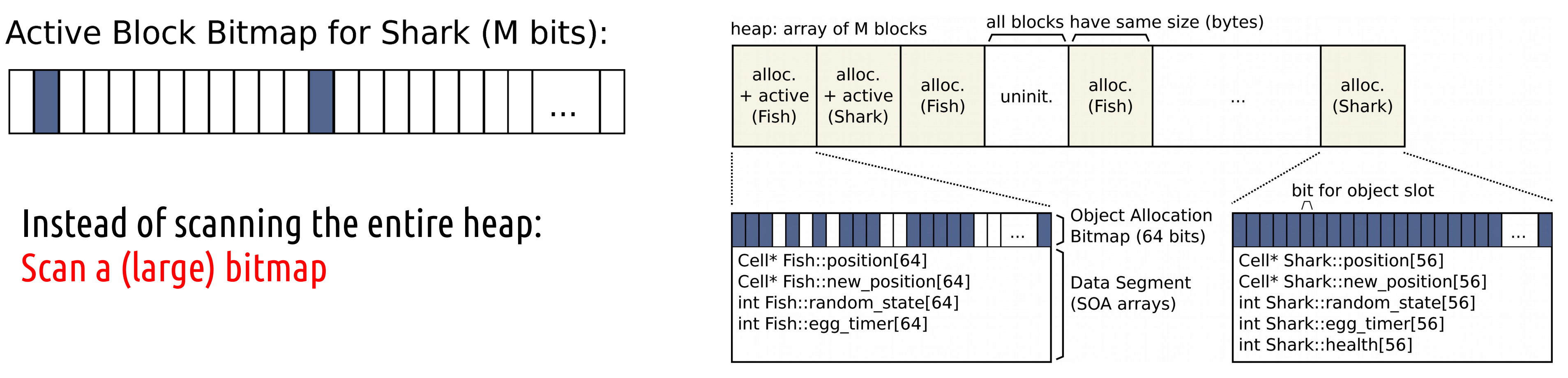
How to find blocks?





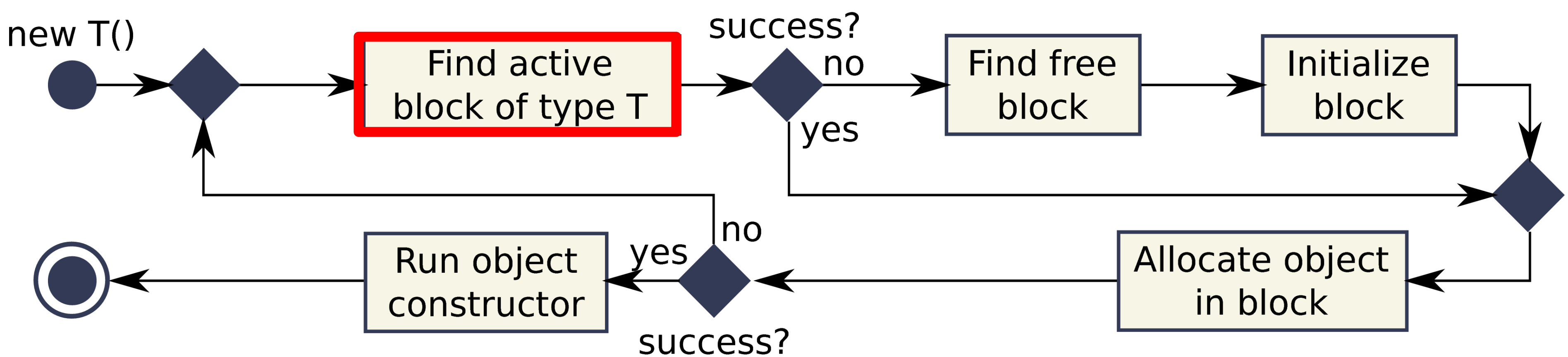
How to find blocks?

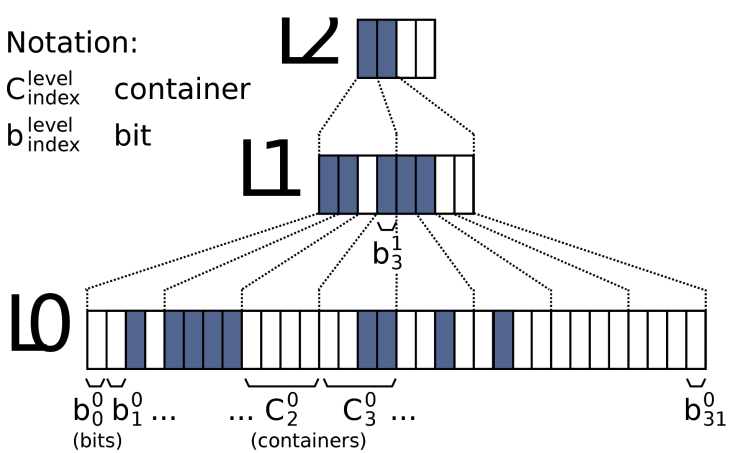


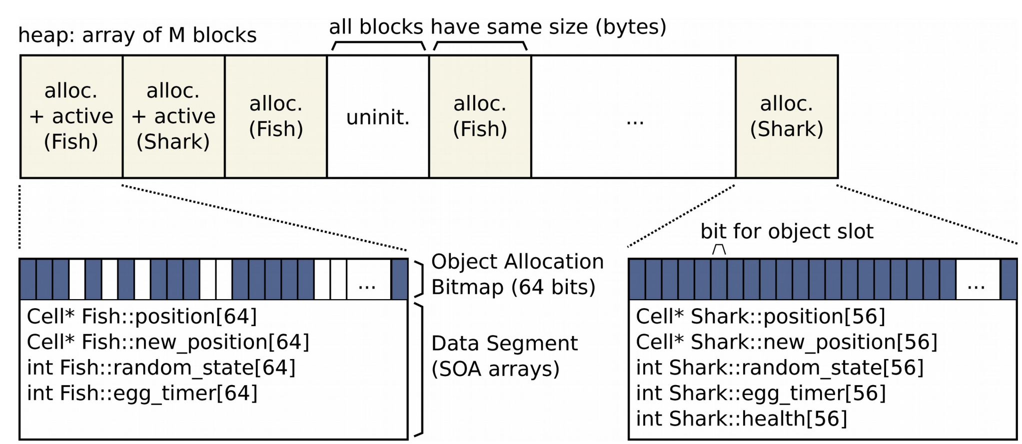


Instead of scanning the entire heap:
Scan a (large) bitmap

How to find blocks?

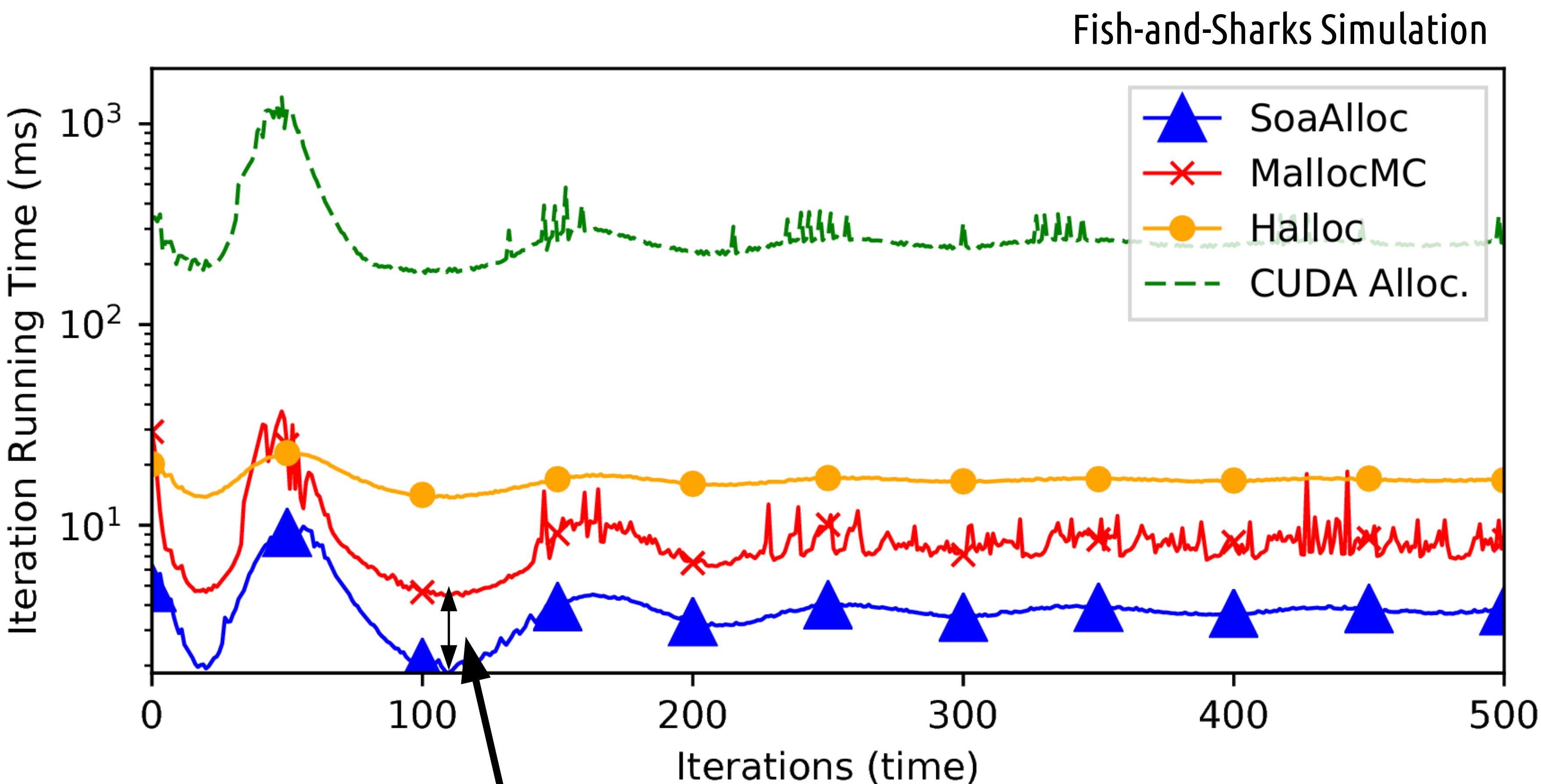






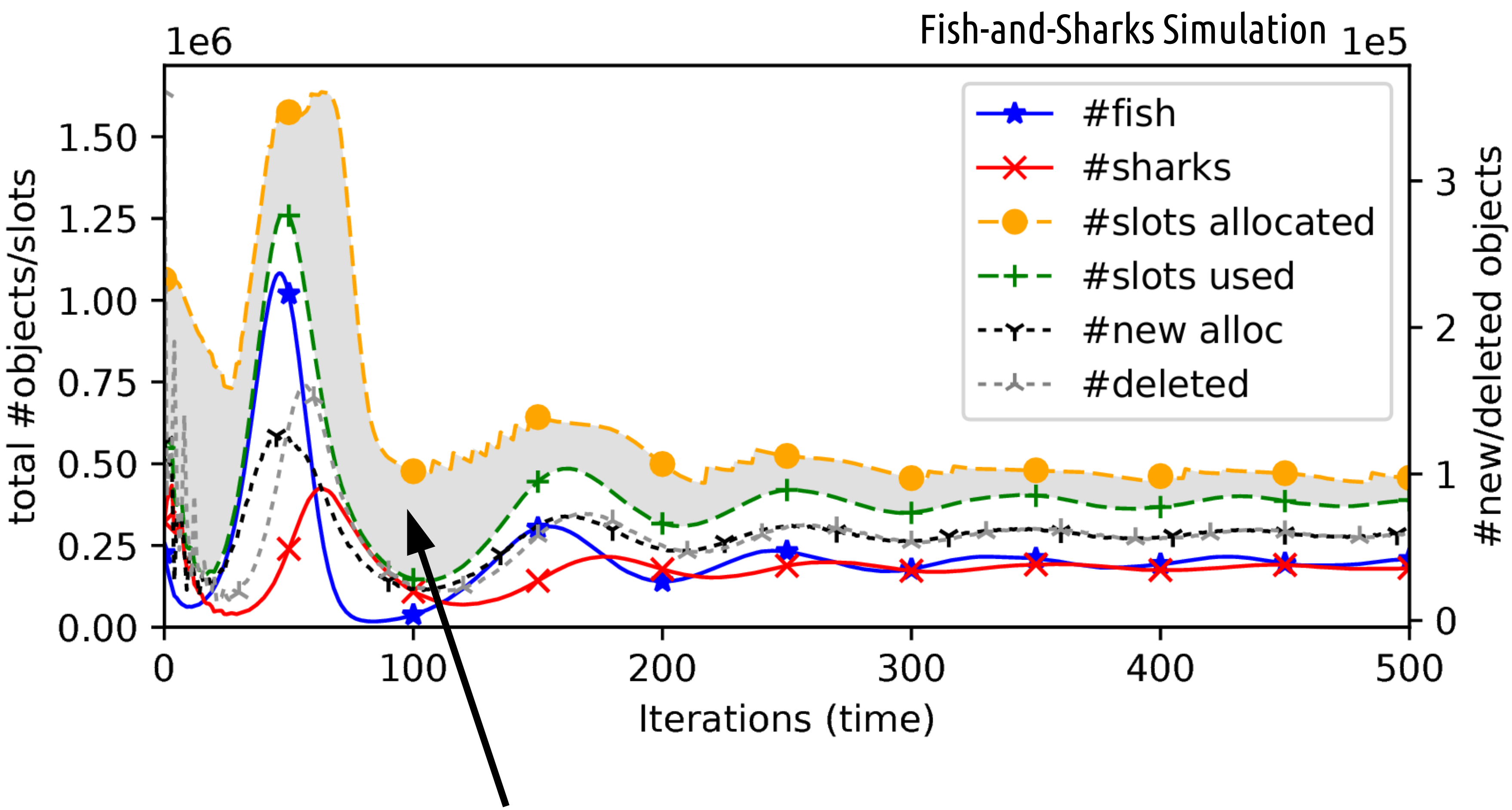
Instead of scanning the entire heap:
Traverse a hierarchical bitmap

Preliminary Benchmark Results



More than 2x speedup compared to MallocMC/ScatterAlloc

Preliminary Benchmark Results



Gray area: Fragmentation overhead
(allocated but unused memory)

Future Work

- Evaluate SoaAlloc with more benchmarks
- Explicit memory defragmentation may lead to further speedups
- Refine implementation:
  e.g., per-warp private blocks (similar to private heaps)

Preliminary Benchmark Results