Register Efficient Memory Allocator for GPUs

Marek Vinkler

Masaryk University in Brno, Czech Republic

Vlastimil Havran

Czech Technical University in Prague, Czech Republic

Motivation

- ray tracing
- data structure (kd-tree)
  - GPU memory allocation
  - existing allocators slow

Motivation contd.

- sophisticated algorithms
  - complex memory usage
- kernel launch overhead
- CPU allocators inefficient

- http://decibel.fi.muni.cz/~xvinkl/CMalloc

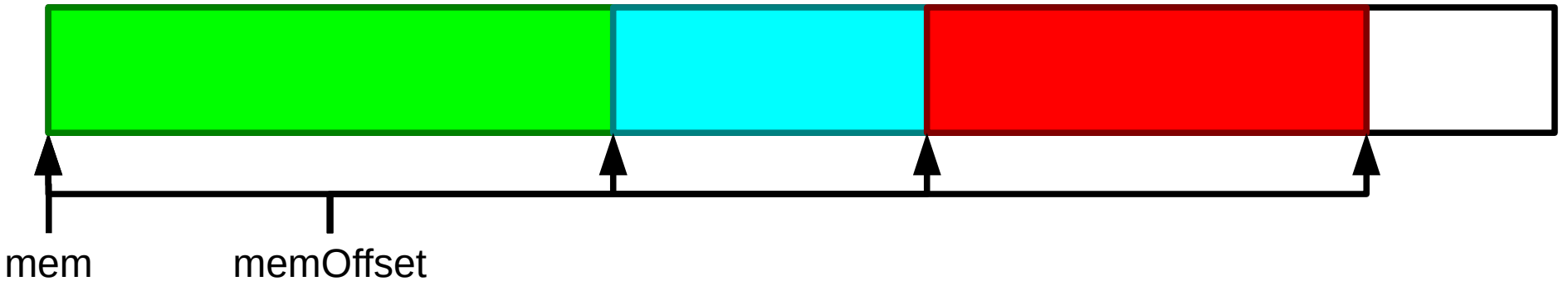
State-of-the-art

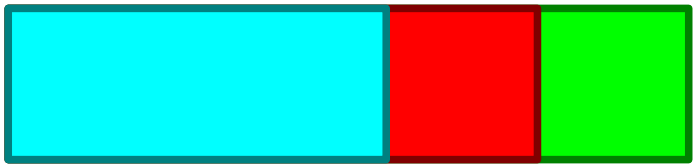
- AtomicMalloc [Tzeng et al., 2010] (hinted)
- CudaMalloc (CUDA built-in, 2010)
- ScatterAlloc [Steinberger et al., 2012]
- FDGMalloc [Widmer et al., 2013]

AtomicMalloc [Tzeng et al., 2010]

- simplest: one atomic operation
- very fast
- serialization of execution
- cannot free memory

AtomicMalloc contd.

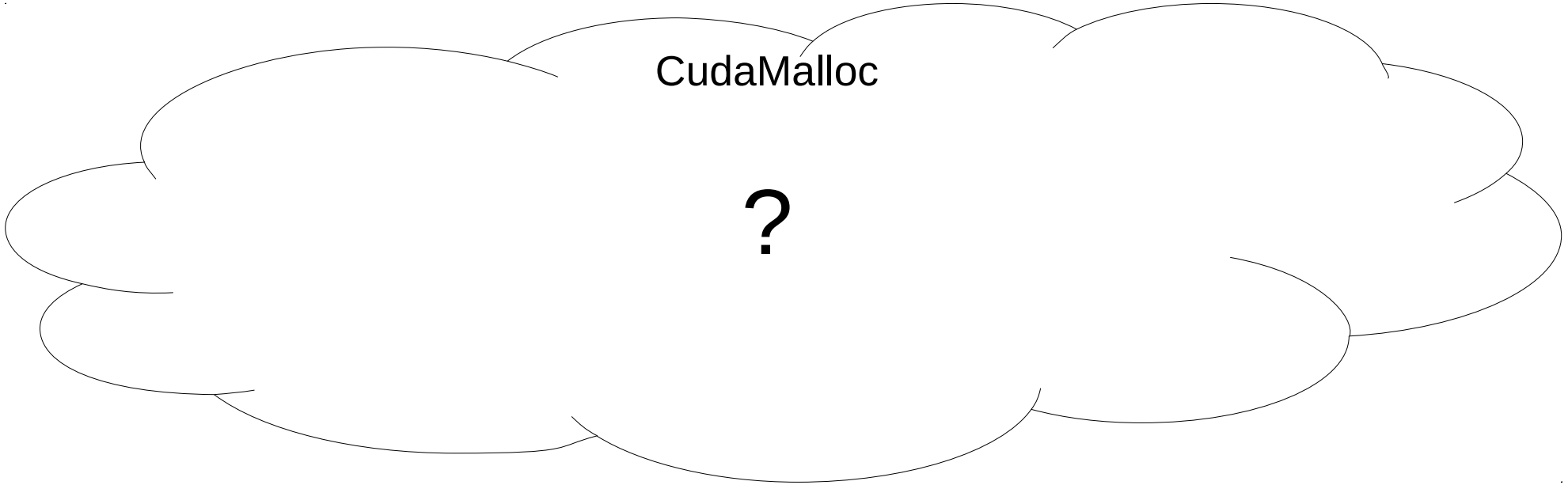




CudaMalloc [2010]

- CUDA 3.2 and higher built-in
- unpublished algorithm
- very slow

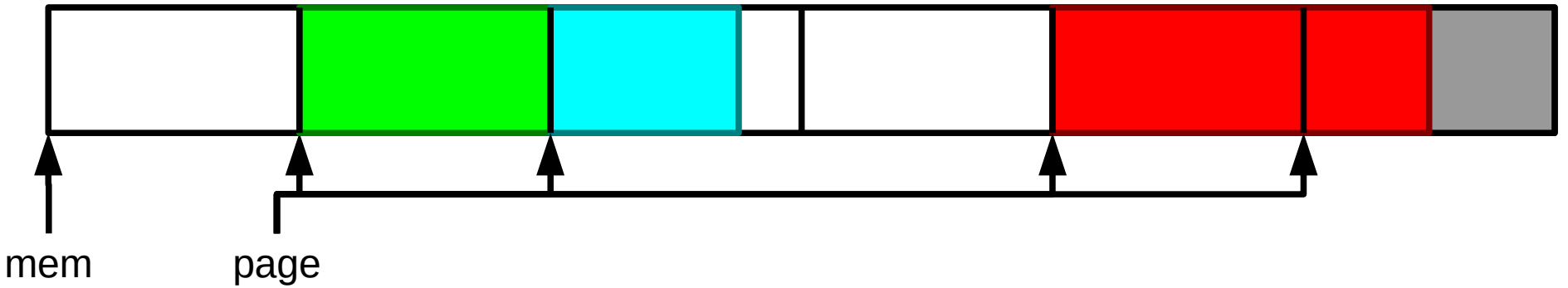
CudaMalloc contd.

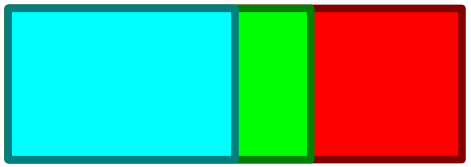


ScatterAlloc [Steinberger et al., 2012]

- hashing
  - many parallel allocations
  - roughly the same size of requests
- very fast on small allocations
- very slow on large allocations

ScatterAlloc contd.

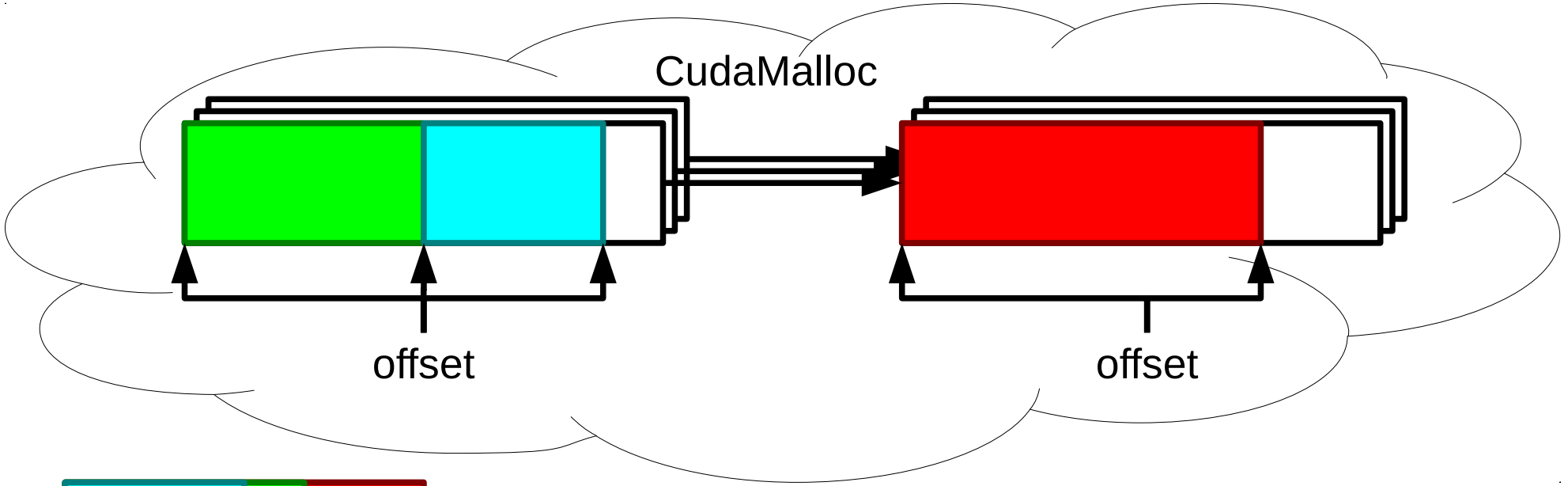


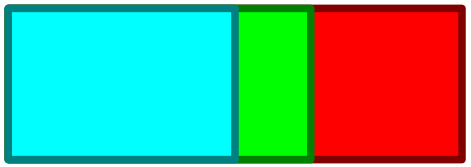


FDGMalloc [Widmer et al., 2013]

- CudaMalloc for superblocks
- local management in superblocks
  - low overhead
- very fast subsequent allocations
- cannot free memory separately

FDGMalloc contd.





Limitations

- slow on large/variable sized allocations
  - CudaMalloc (CUDA built-in, 2010)
  - ScatterAlloc [Steinberger et al., 2012]
- cannot reuse memory
  - AtomicMalloc [Tzeng et al. 2010]
  - FDGMalloc [Widmer et al. 2013]

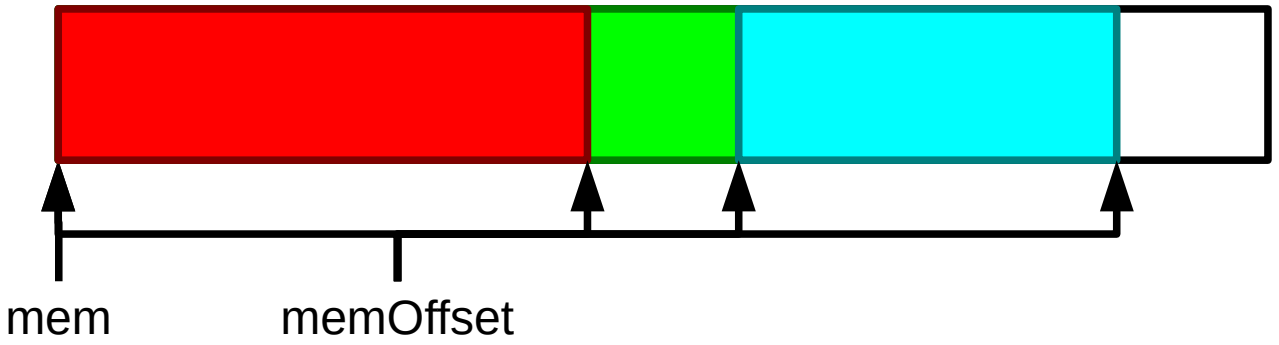
Two Proposed GPU Allocators

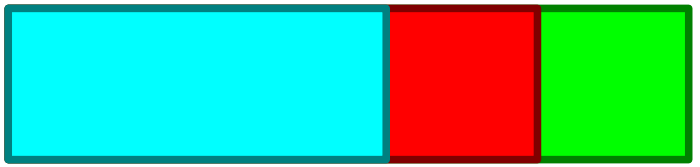
- AWMalloc
  - variant of AtomicMalloc
  - can reuse memory
- CMalloc
  - practical list-based allocator

AWMalloc

- circular memory pool
- used memory can be overwritten
- large memory pool and short-lived allocations

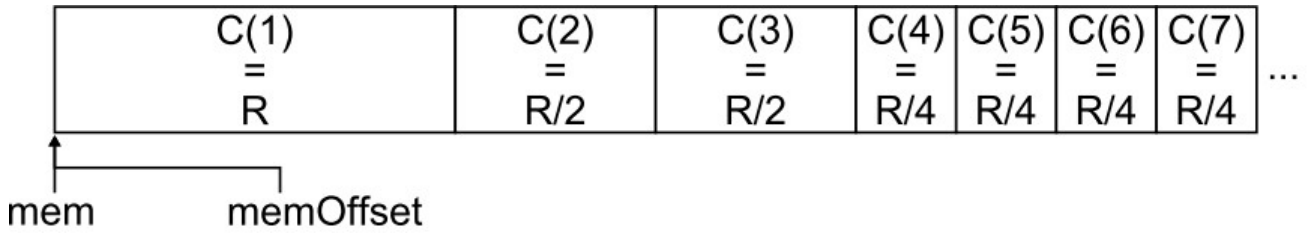
AWMalloc contd.



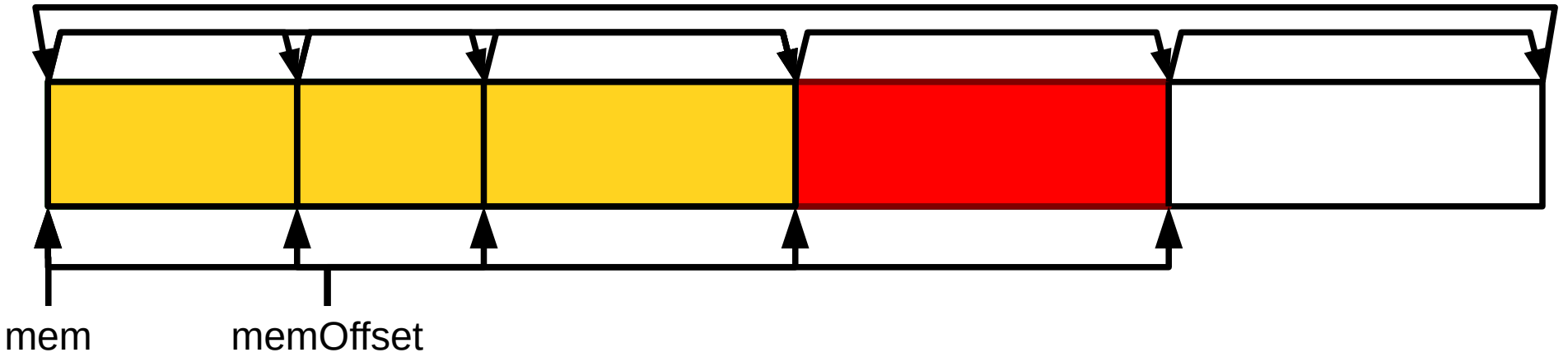


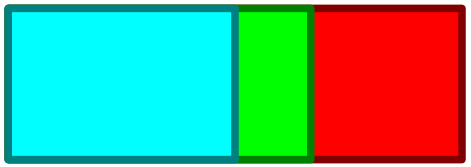
CMalloc

- circular memory pool
- header for individual allocations
  - correctness
  - split / merge
- initial splitting: $C(i) = R/2^{\lfloor log_2(i) \rfloor}$



CMalloc contd.





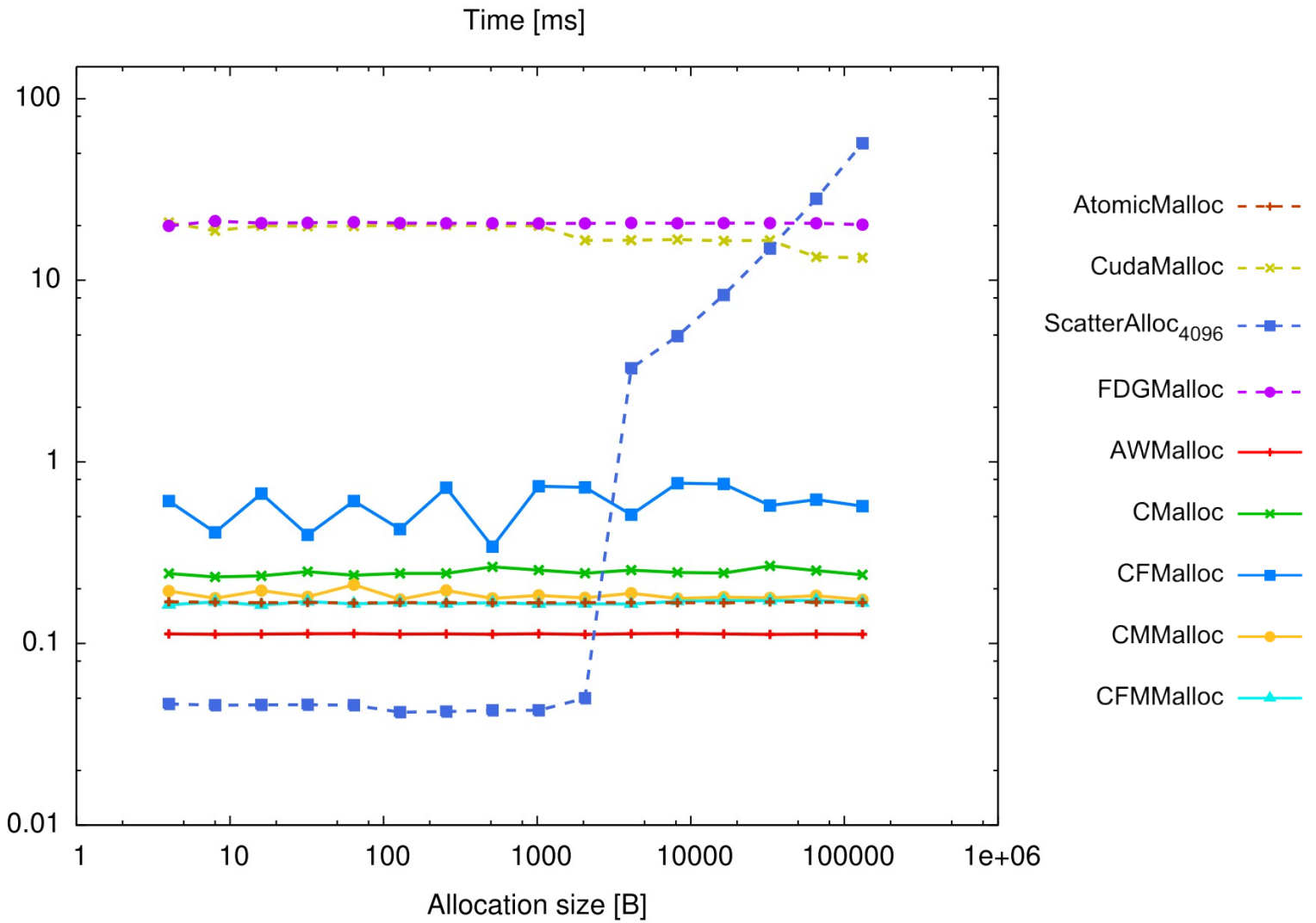
CMalloc Variants

- CFMalloc
  - fused flag and next pointer
  - single word read/write
- CMMalloc & CFMMalloc
  - multiple offsets into memory pool
  - fewer conflicts
  - fused variant

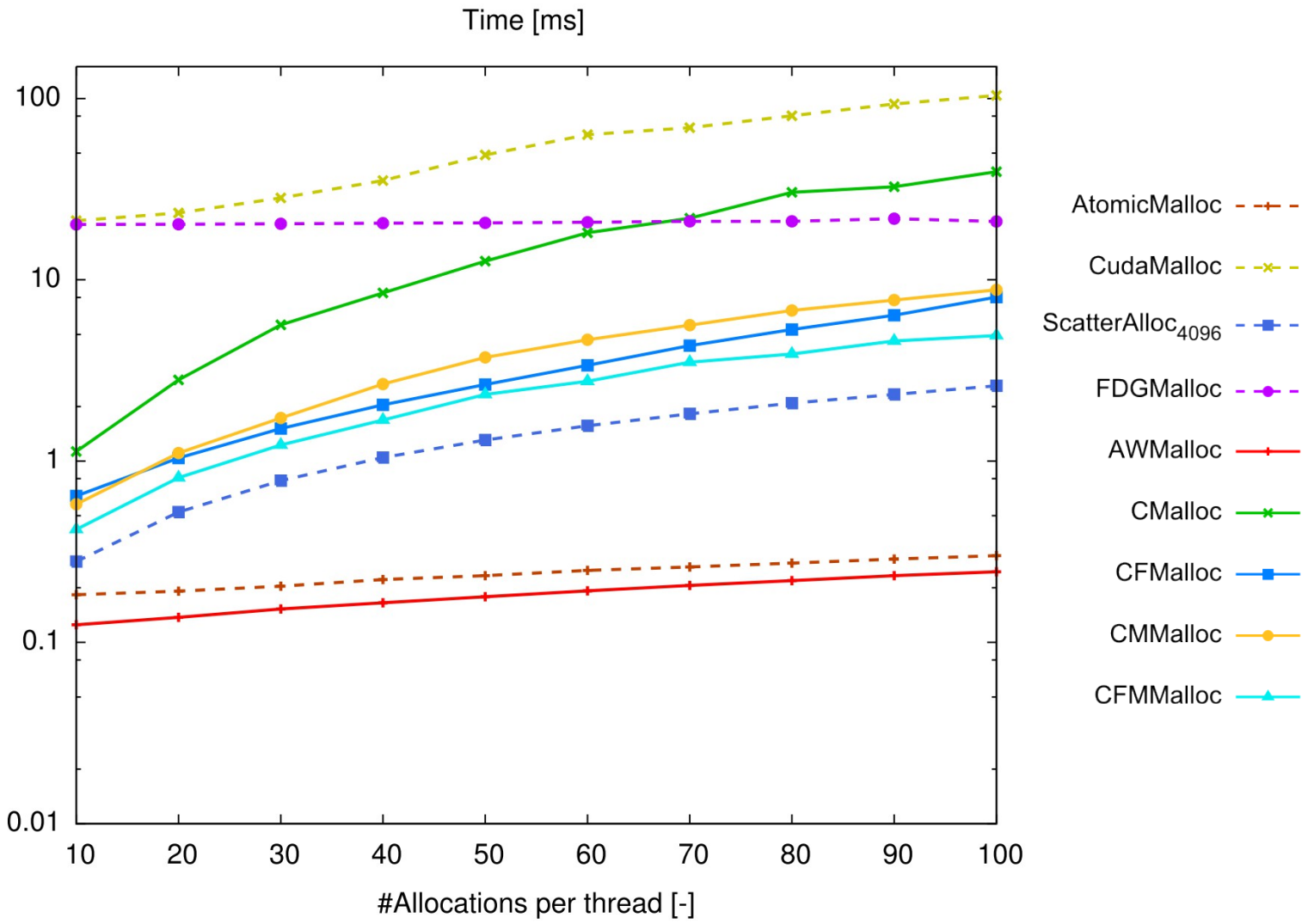
Four Evaluation Tests

- Alloc Dealloc [Huang et al., 2010]
  - allocation followed by deallocation
- Alloc Cycle Dealloc [Widmer et al. 2013]
  - many allocations followed by deallocation of all memory
- Probability [Steinberger et al., 2012]
  - multiple kernels
  - allocation with probability pAlloc, deallocation with probability pFree
- Data Structure Build
  - kd-tree build
  - allocation of children index arrays

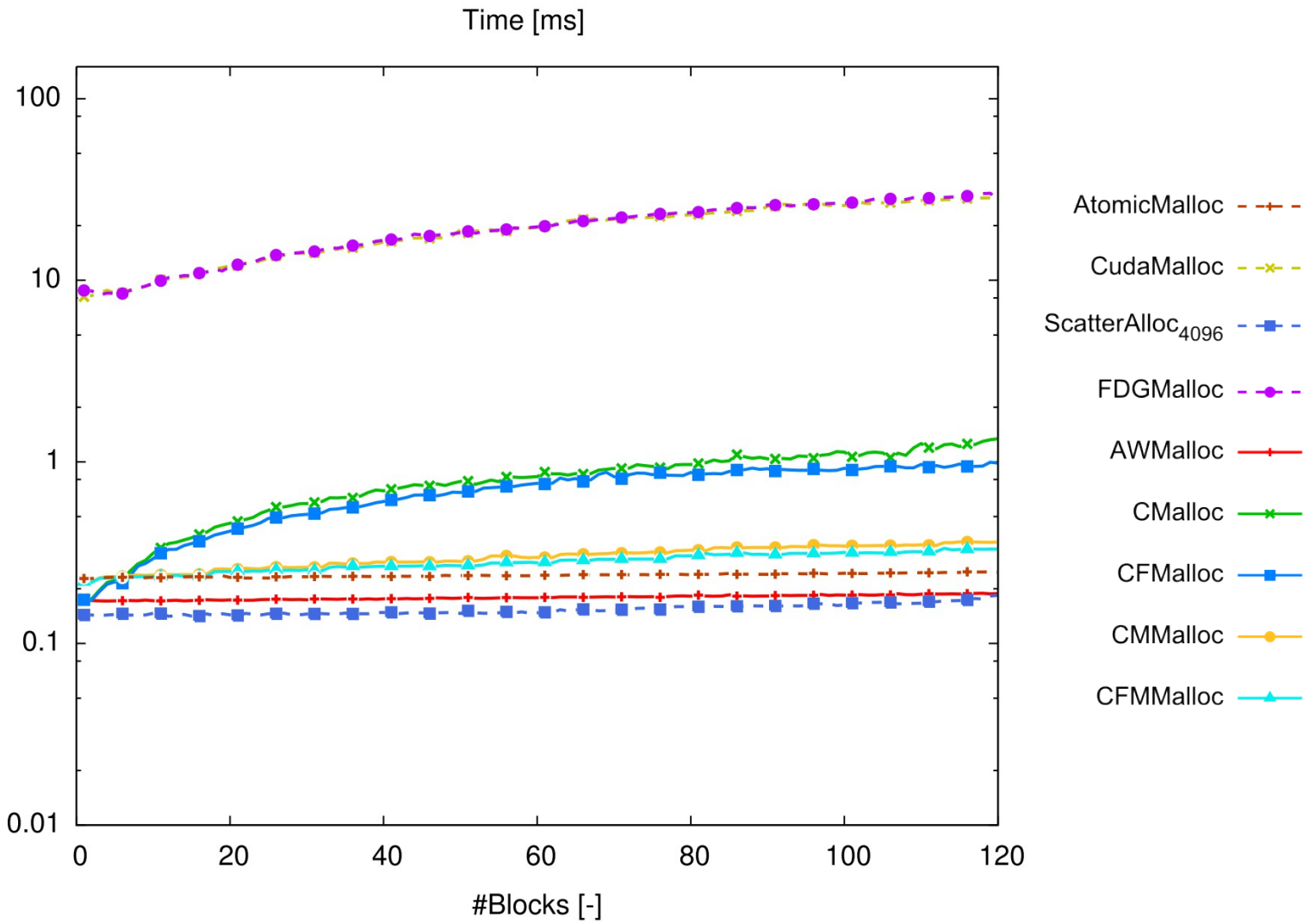
Alloc Dealloc – Allocation Size



Alloc Cycle Dealloc – Allocations per Thread



Probability – Number of CUDA Blocks



Data Structure Build (Slowdown)

		Kd-tree build time (the lower the better)			
Allocator	#Registers	Sponza	Crytek Sponza	Dragon	Sodahall
AtomicMalloc*	4	18.4 ms	43.2 ms	378.9 ms	325.5 ms
CudaMalloc	6	2.99x	3.82x	20.95x	3.38x
ScatterAlloc	38	1.74x	2.01x	1.78x	5.08x
AWMalloc*	6	0.88x	0.96x	0.90x	0.95x
CMalloc	16	**1.04x**	1.14x	1.27x	**1.14x**
CFMalloc	10	1.20x	**1.08x**	**1.17x**	1.17x

* Not full dynamic memory allocator

Conclusions

- four evaluation tests (new kd-tree build)
- AtomicMalloc / AWMalloc
  - no deallocation
- ScatterAlloc
  - similarly sized allocations, enough registers
- FDGMalloc
  - successive allocations
- CMalloc / CFMalloc
  - variably sized allocations / unknown allocation properties
  - up to 5.08 / 1.14 = 4.46x faster than ScatterAlloc

Project Webpage

- http://decibel.fi.muni.cz/~xvinkl/CMalloc