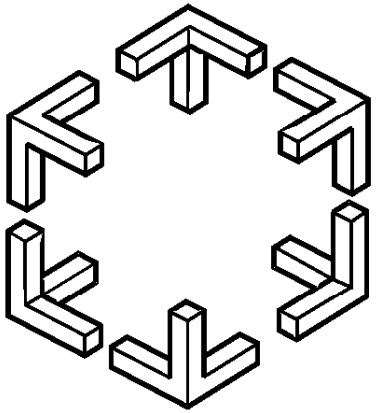


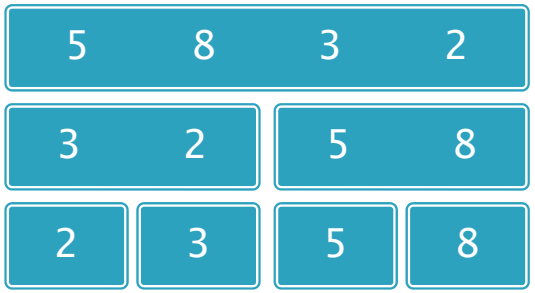
A Practical Quicksort Algorithm for Graphics Processors

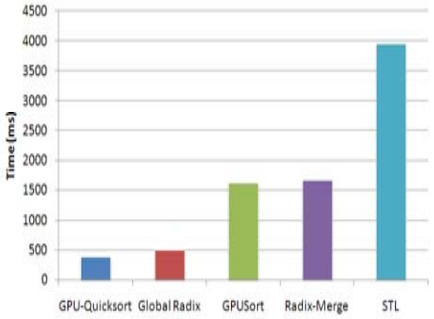
Daniel Cederman and Philippas Tsigas

Distributed Computing and Systems
Chalmers University of Technology

Overview

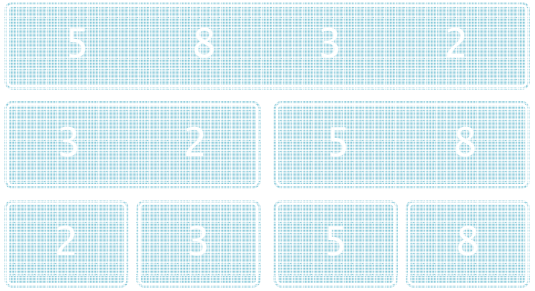
- System Model
- The Algorithm
- Experiments

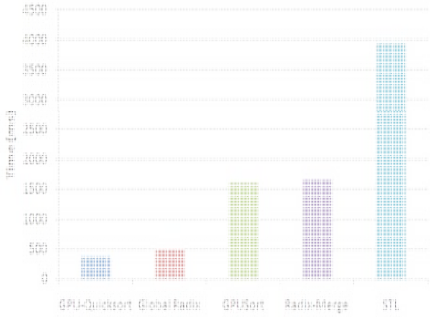




System Model

- System Model
- The Algorithm
- Experiments





CPU vs. GPU

- Multi-core
- Large cache
- Few threads
- Speculative execution

- Many-core
- Small cache
- Wide and fast memory bus
- Thousands of threads hides memory latency

Normal processors

Graphics processors

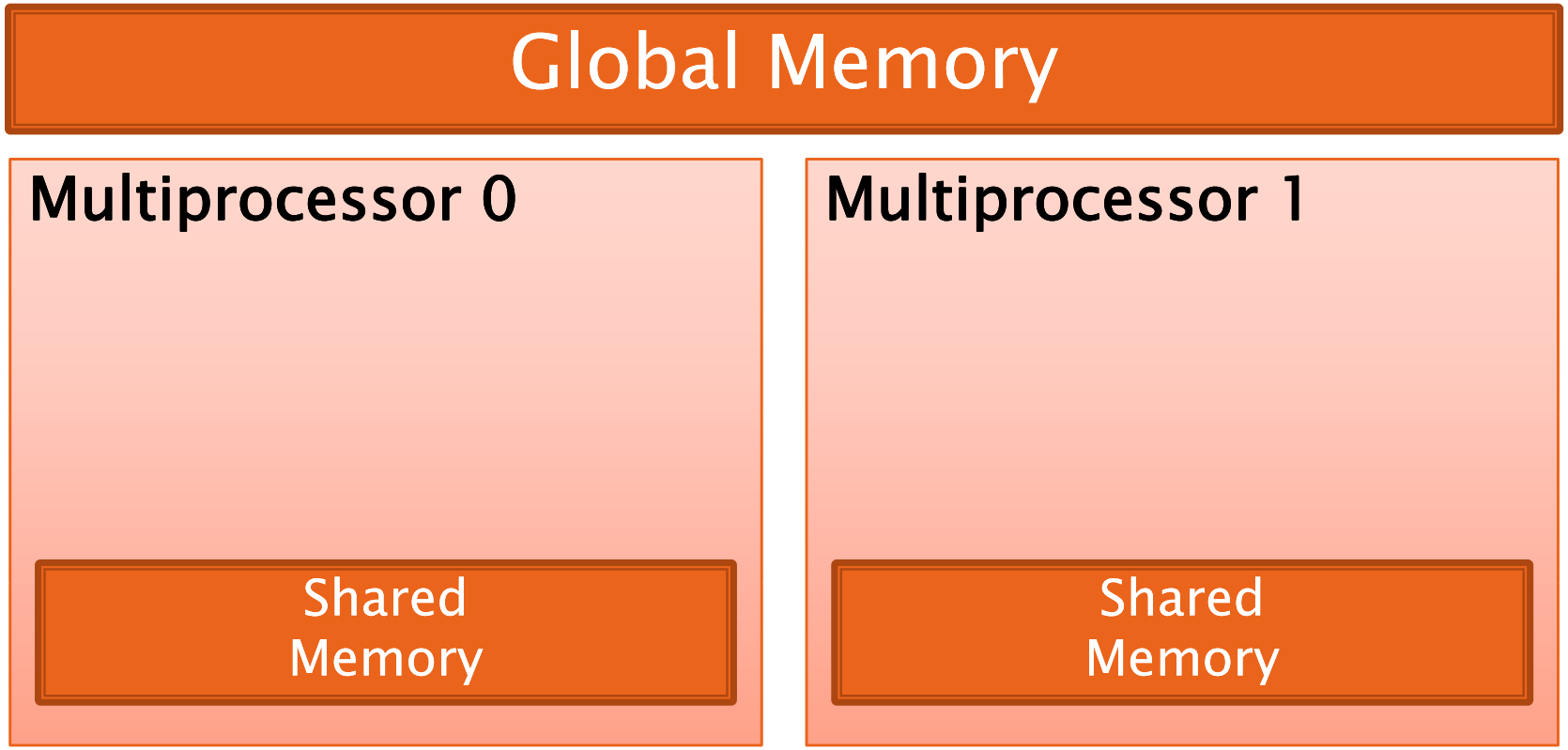
CUDA

- Designed for general purpose computation
- Extensions to C/C++

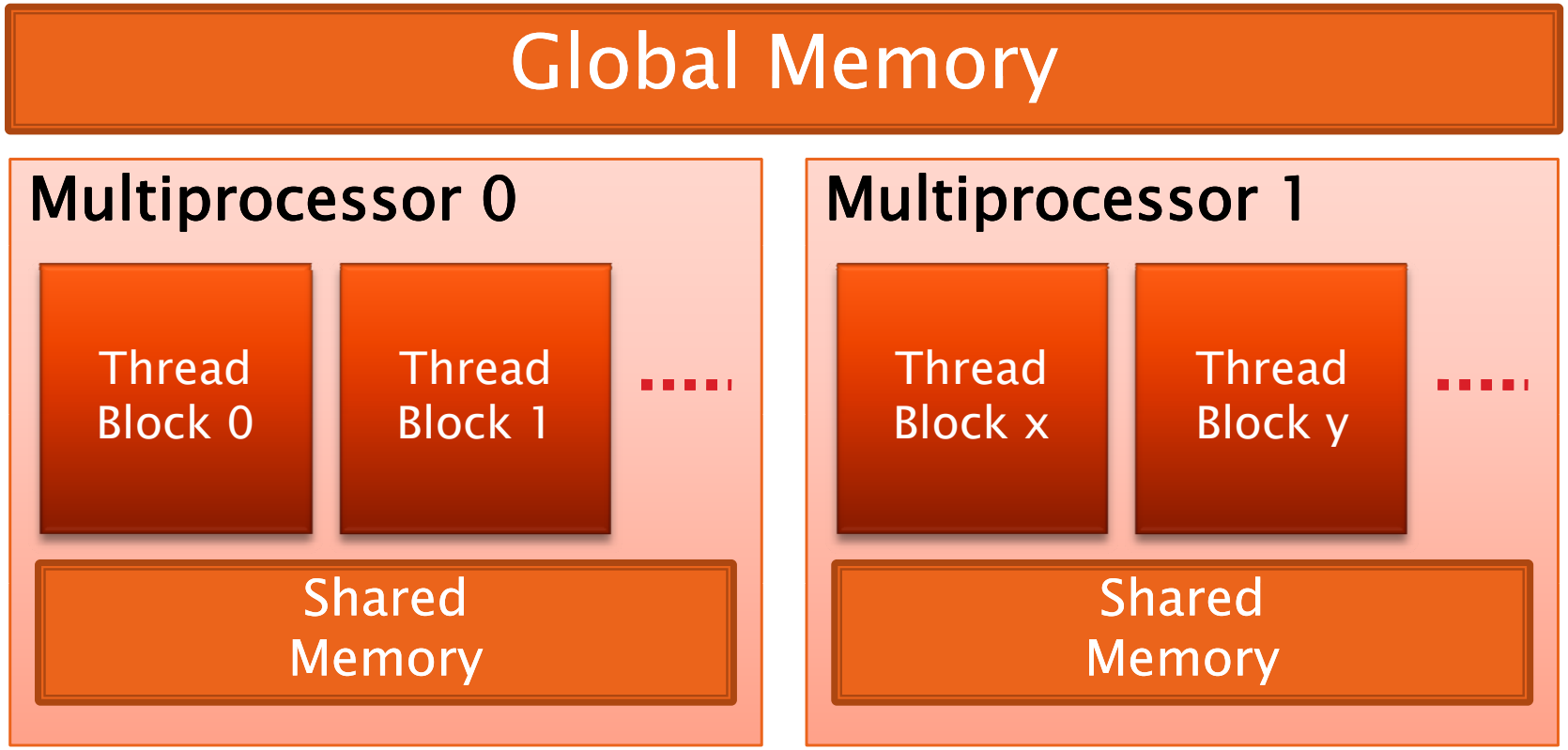
System Model – Global Memory

Global Memory

System Model – Shared Memory



System Model – Thread Blocks

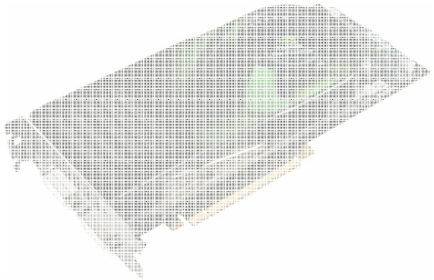


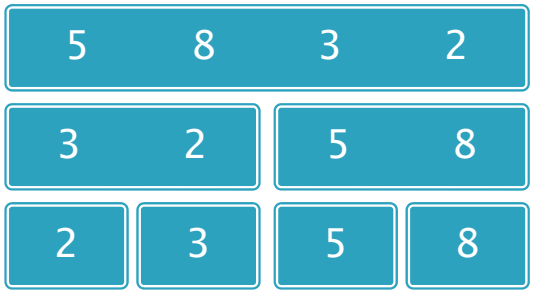
Challenges

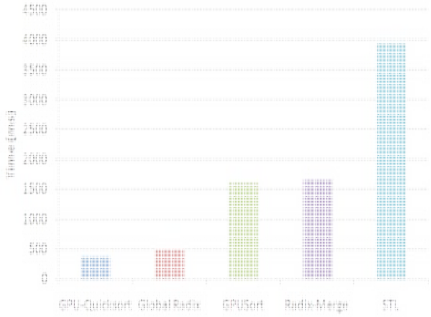
- Minimal, manual cache
- 32-word SIMD instruction
- Coalesced memory access
- No block synchronization
- Expensive synchronization primitives

The Algorithm

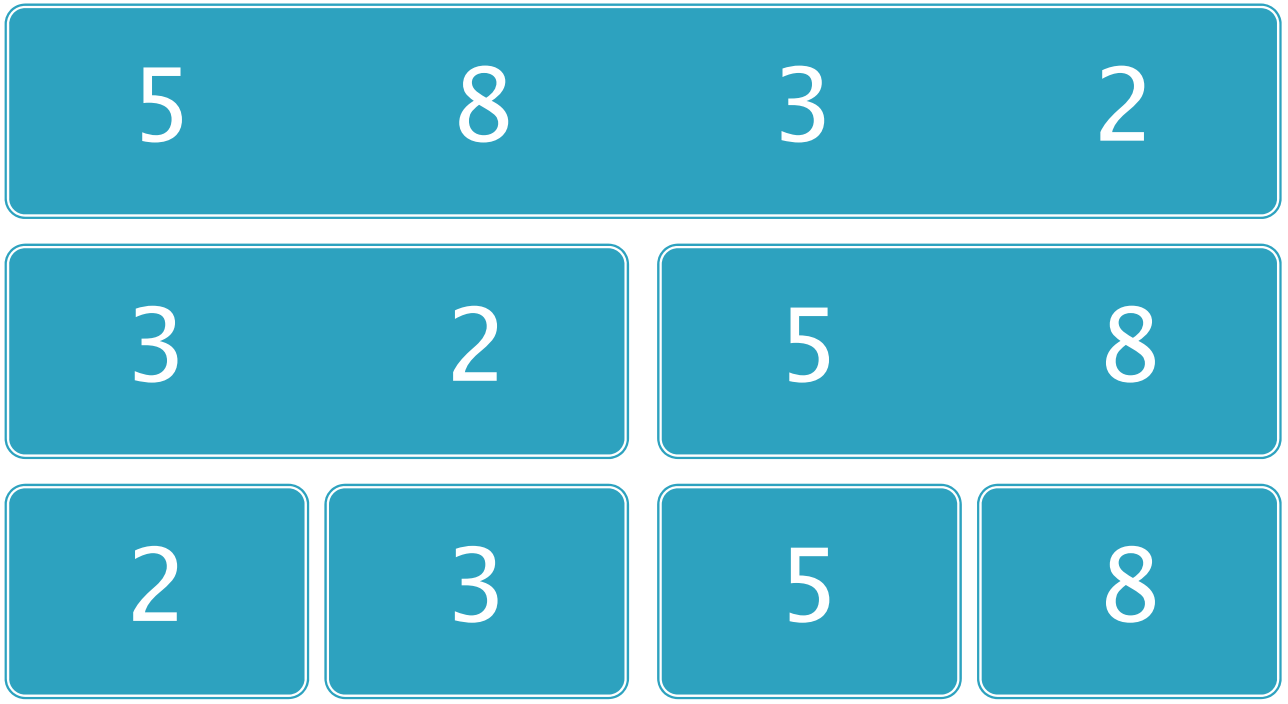
- System Model
- The Algorithm
- Experiments



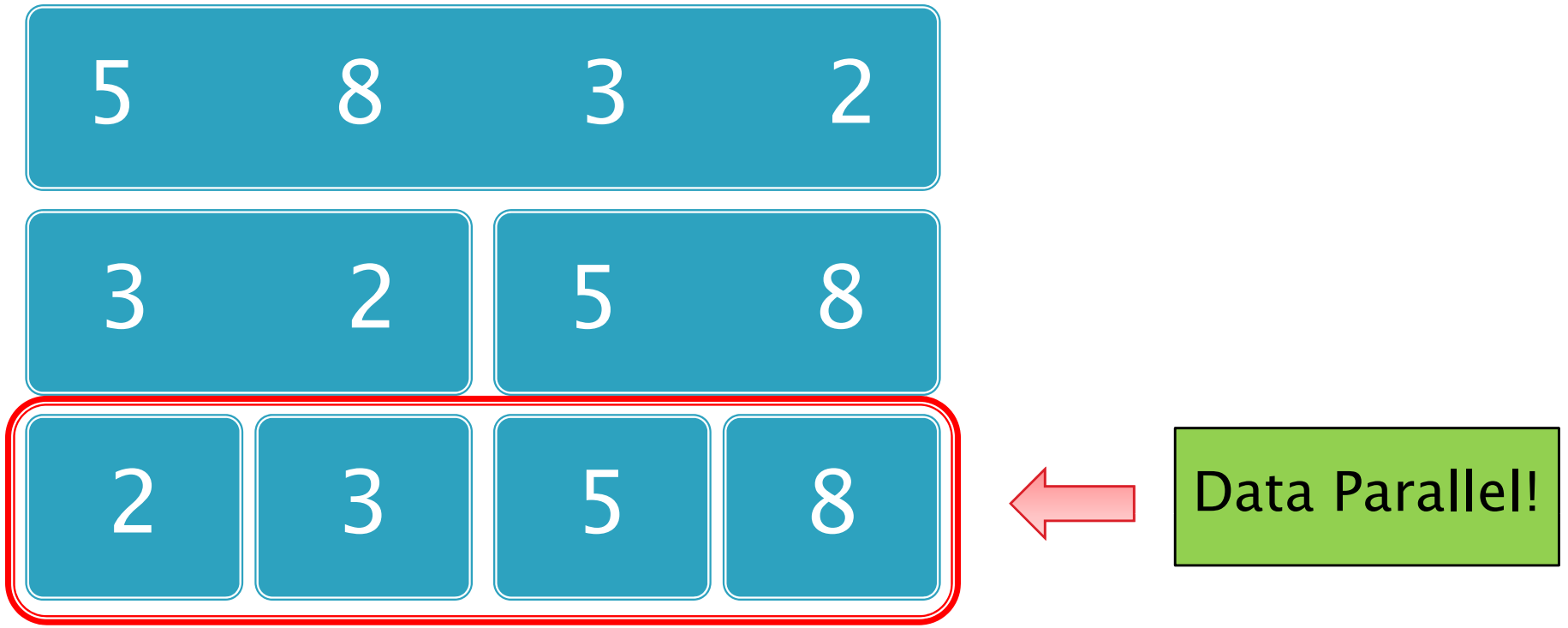




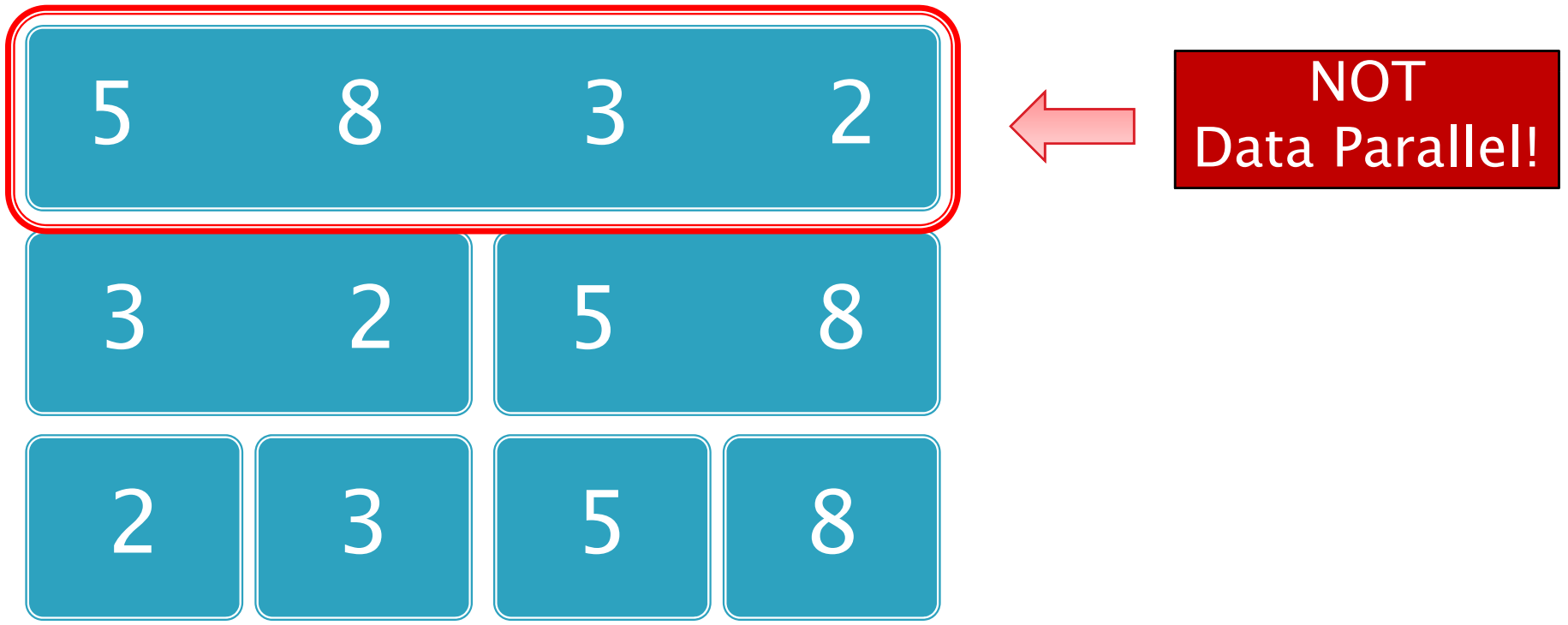
Quicksort



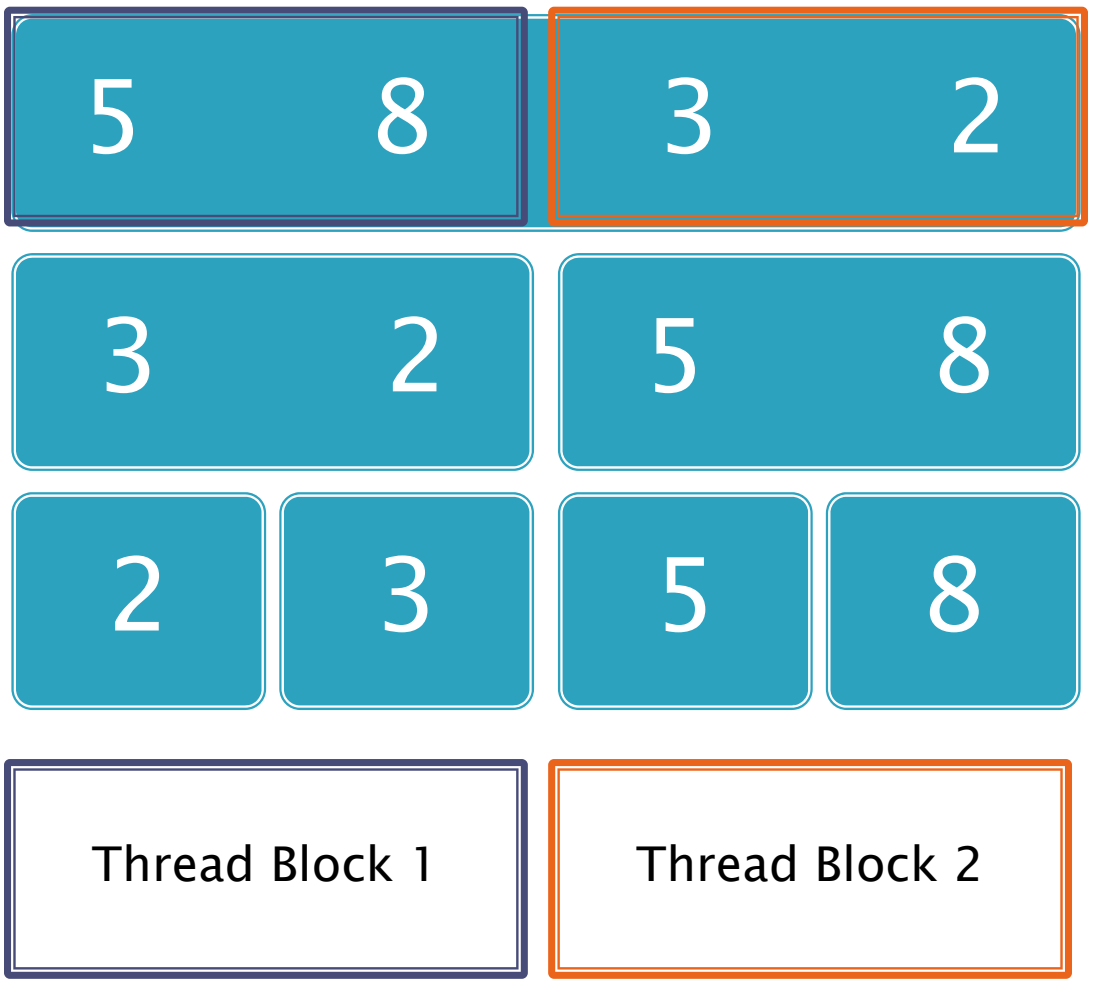
Quicksort



Quicksort

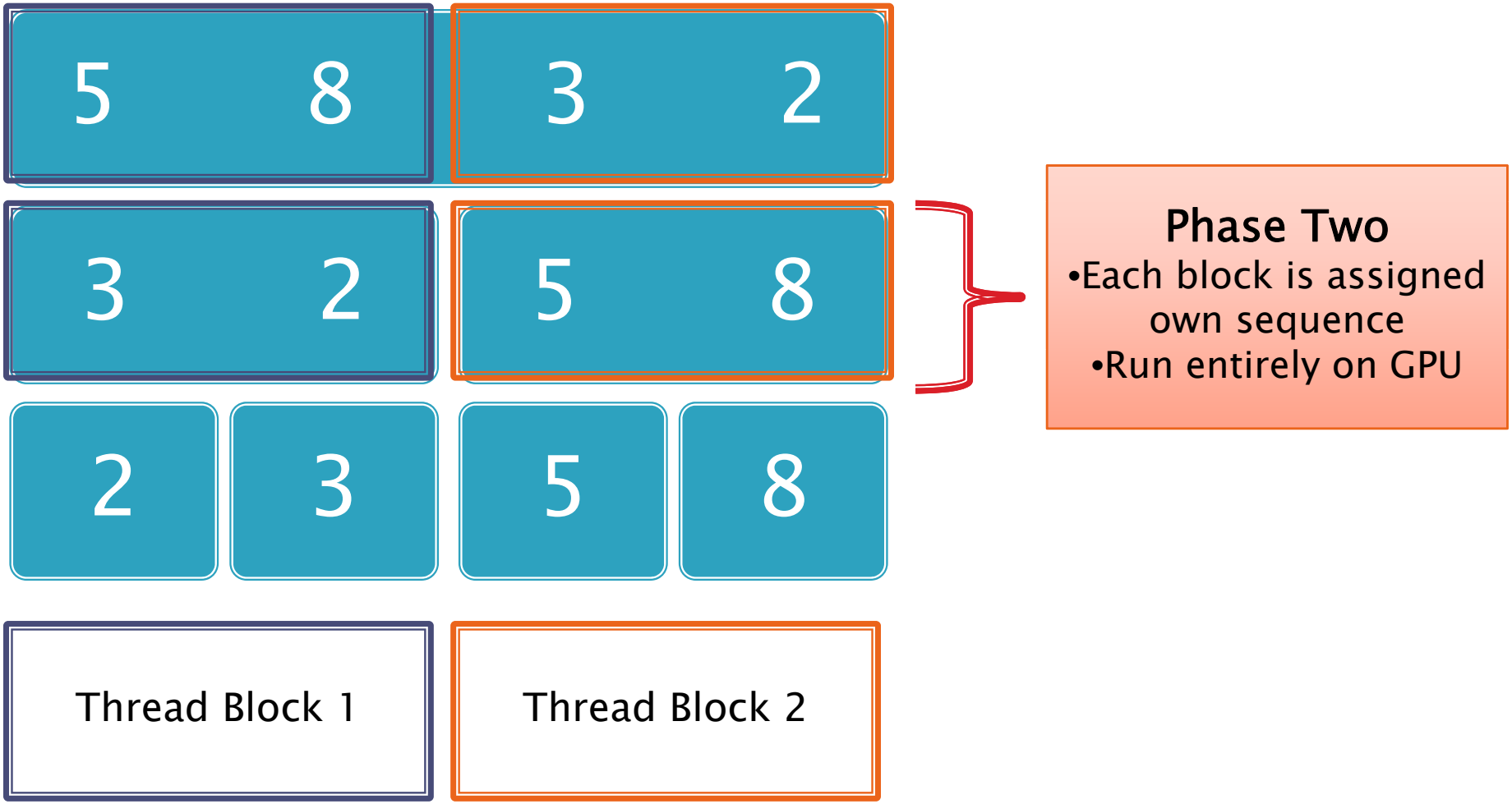


Quicksort



Phase One
- Sequences divided
- Block synchronization on the CPU

Quicksort


5 8
3 2
3 2
5 8
2 3 5 8
Thread Block 1
Thread Block 2
Phase Two
•Each block is assigned own sequence
•Run entirely on GPU

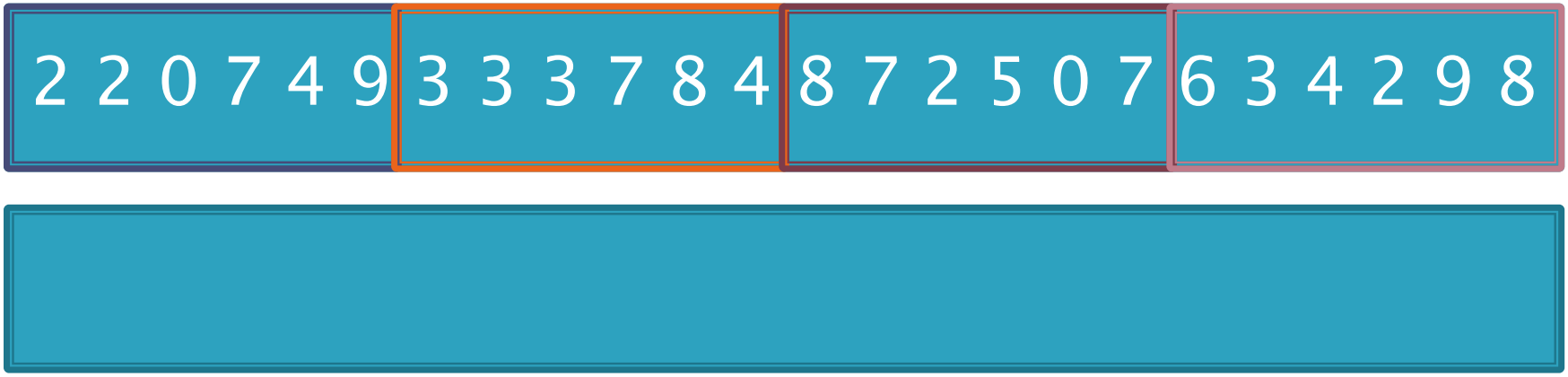
Quicksort – Phase One

2	2	0	7	4	9	3	3	3	7	8	4	8	7	2	5	0	7	6	3	4	2	9	8

Quicksort – Phase One

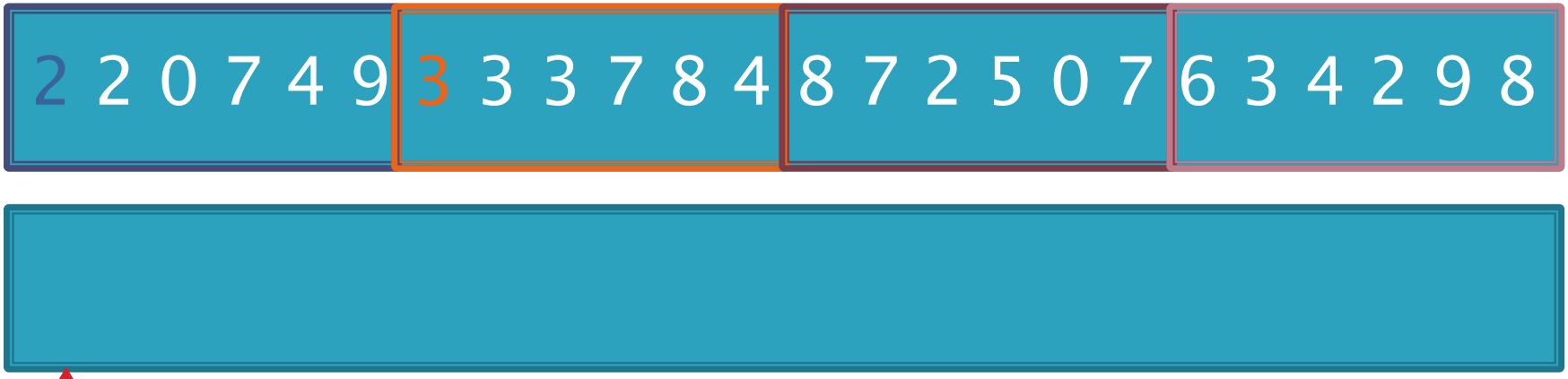
- Assign Thread Blocks

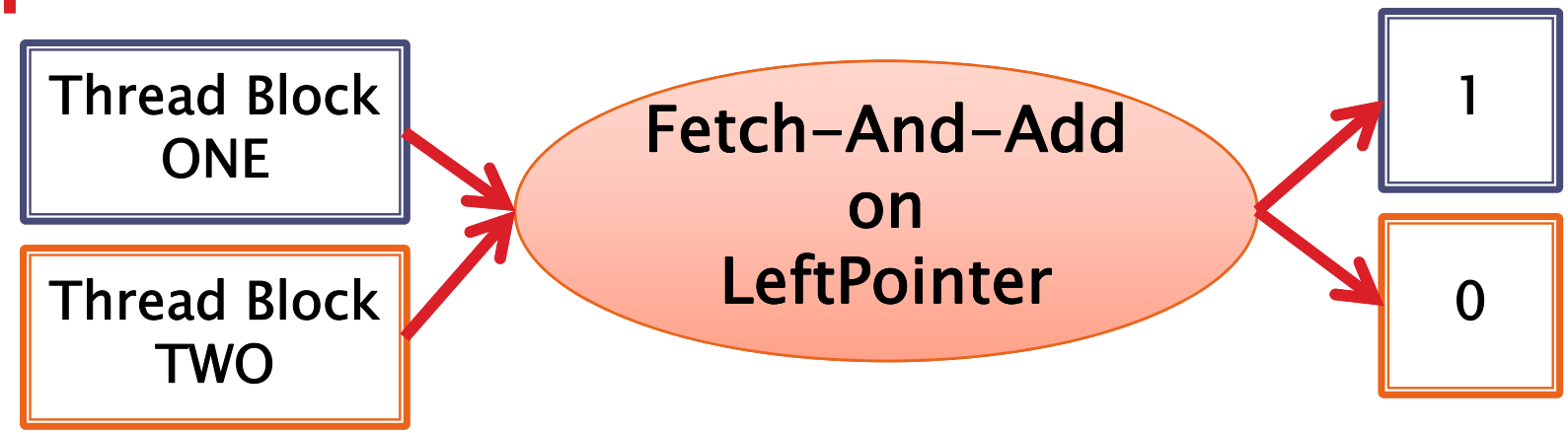
Quicksort – Phase One



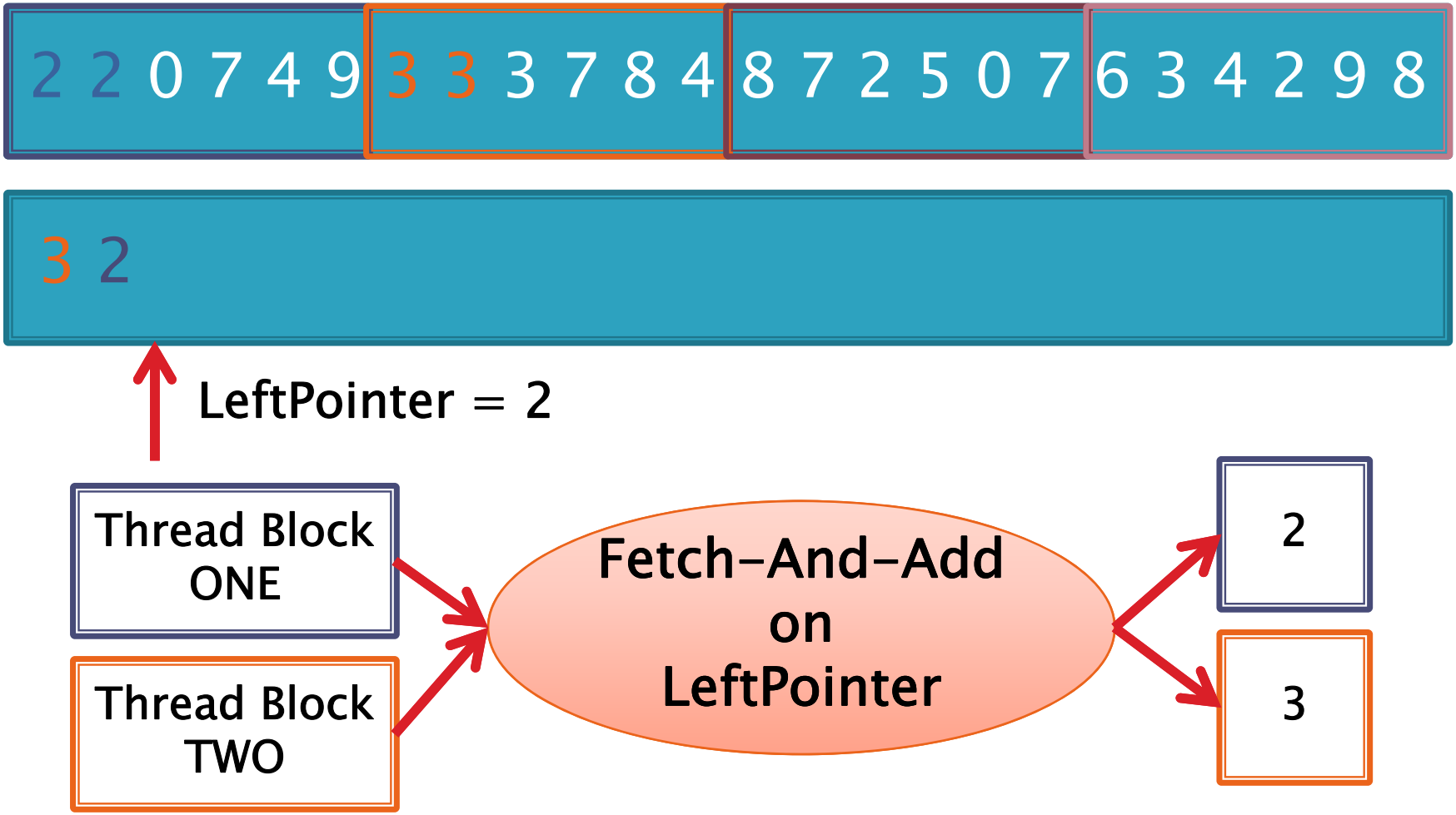
- Uses an auxiliary array
- In-place too expensive

Quicksort – Synchronization

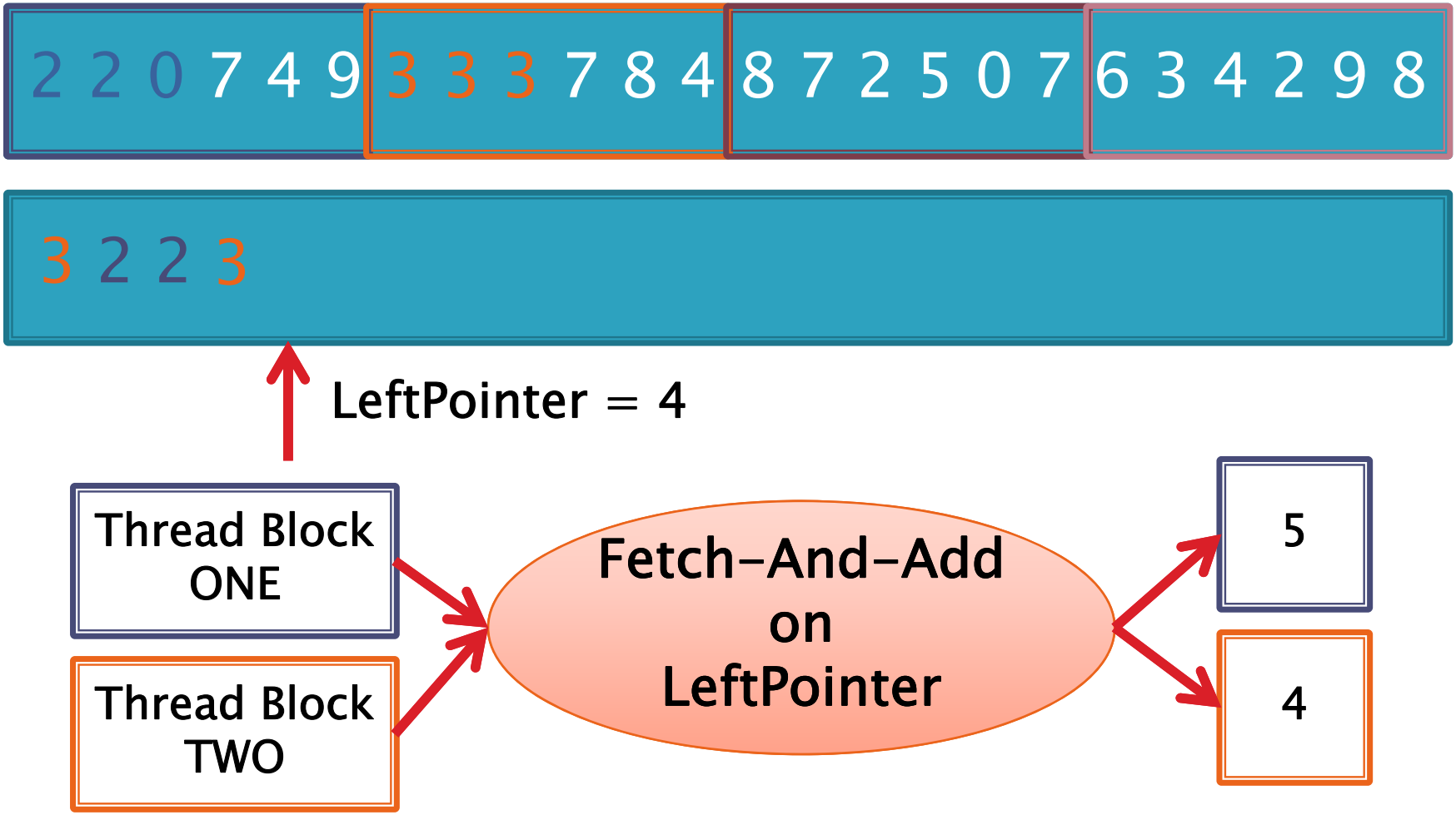




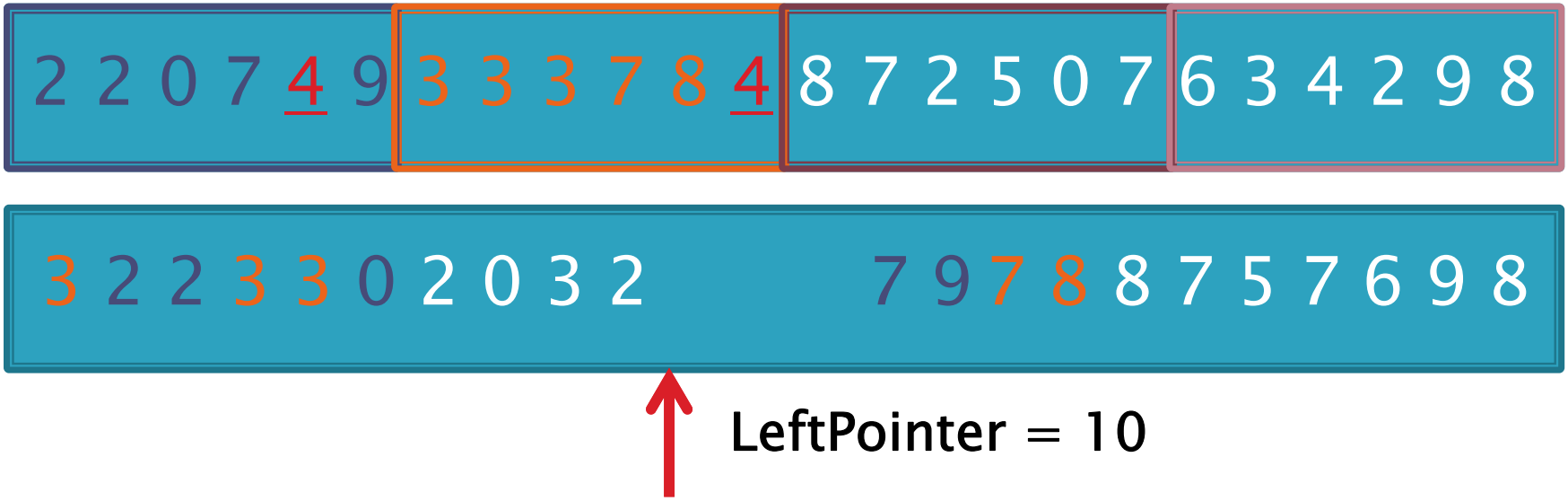
Quicksort – Synchronization



Quicksort – Synchronization



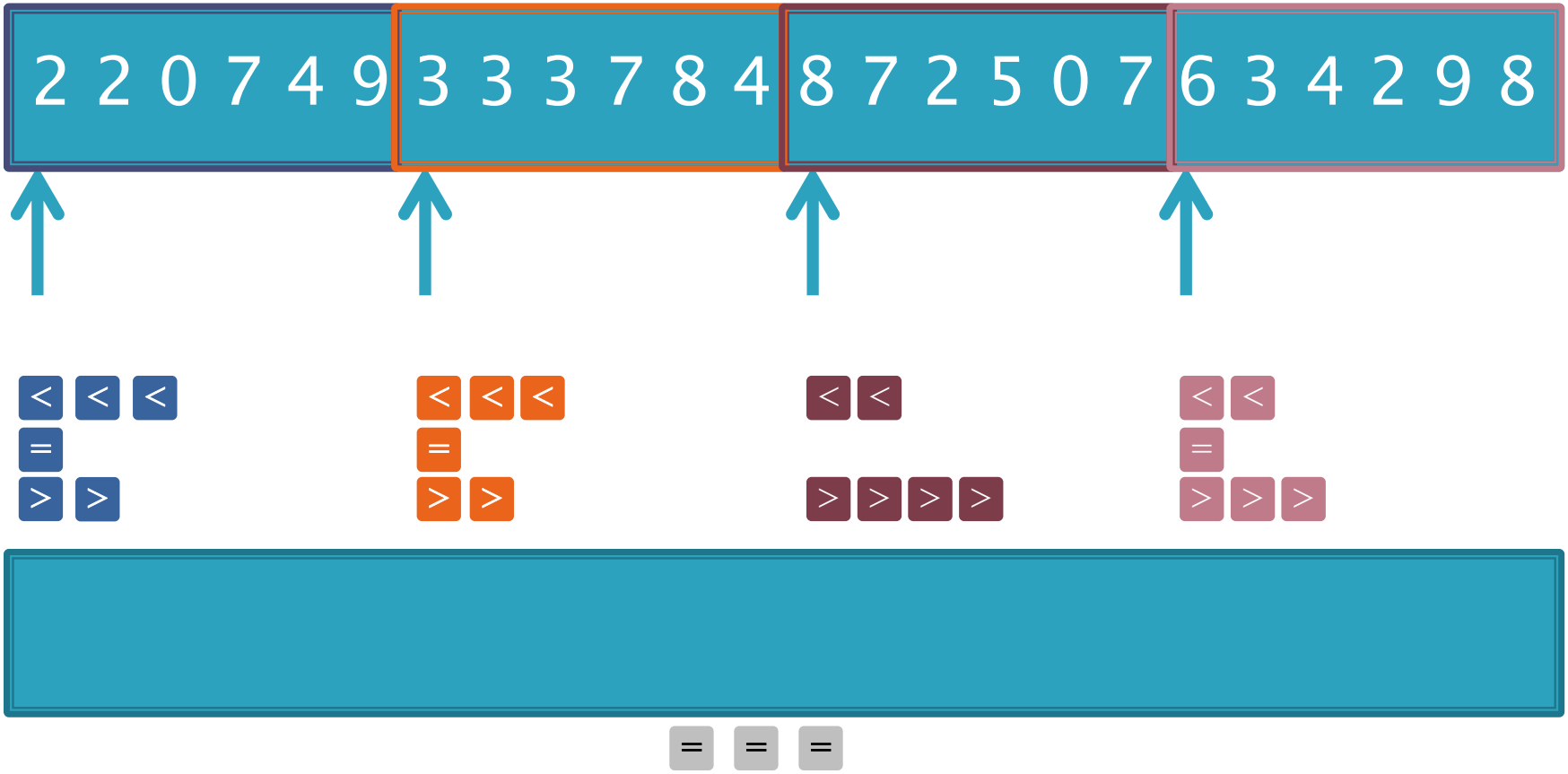
Quicksort – Synchronization



Quicksort – Synchronization

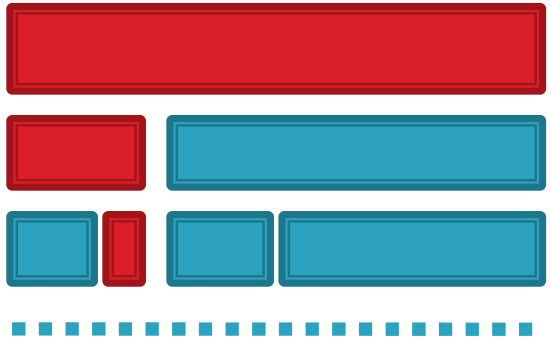
Three way partition

Two Pass Partitioning



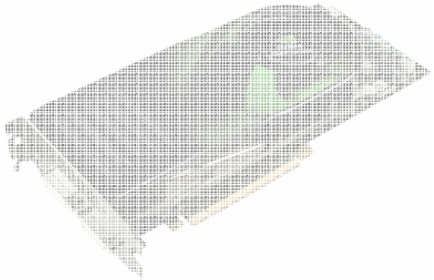
Second Phase

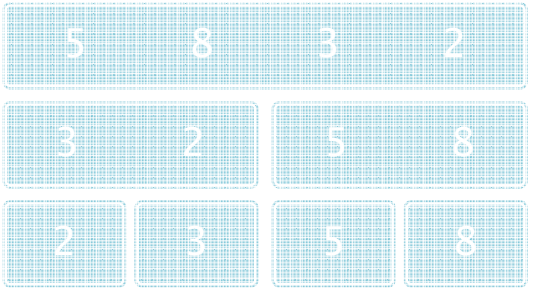
- Recurse on smallest sequence
  - Max depth log(n)
- Explicit stack
- Alternative sorting
  - Bitonic sort

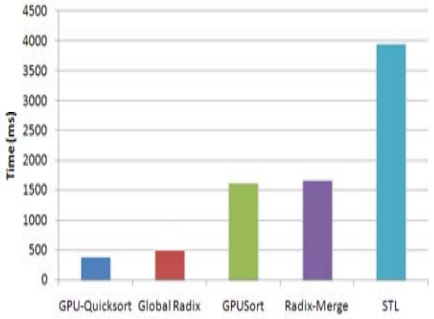


The Algorithm

- System Model
- The Algorithm
- Experiments







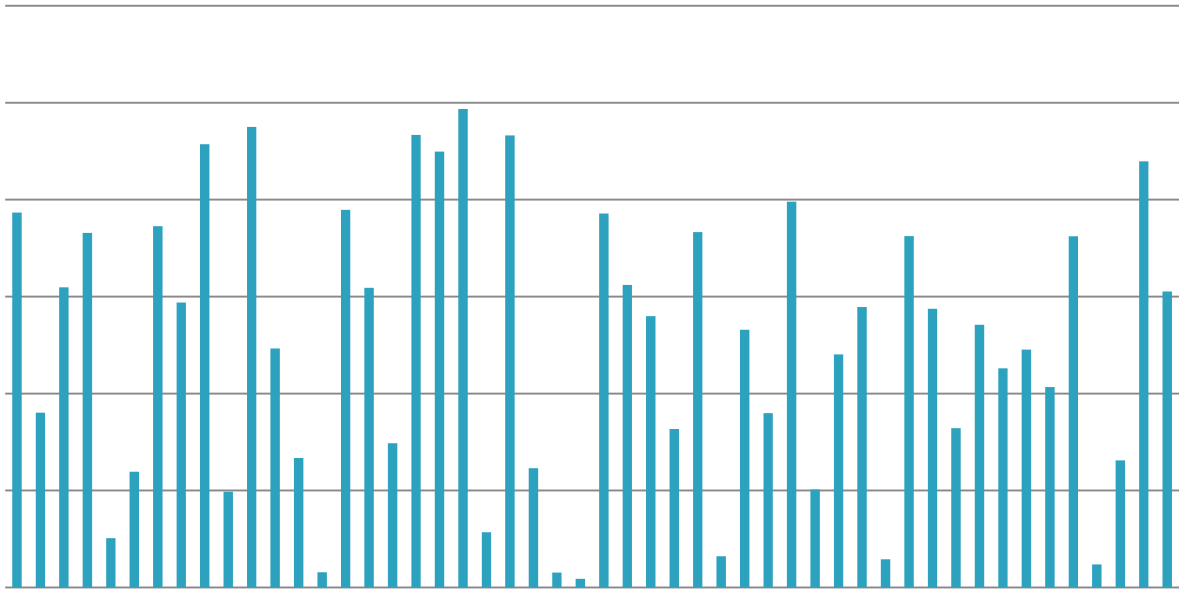
Set of Algorithms

GPU-Quicksort	Cederman and Tsigas, ESA08
GPUSort	Govindaraju et. al., SIGMOD05
Radix-Merge	Harris et. al., GPU Gems 3 '07
Global radix	Sengupta et. al., GH07
Hybrid	Sintorn and Assarsson, GPGPU07

Introsort	David Musser, Software: Practice and Experience

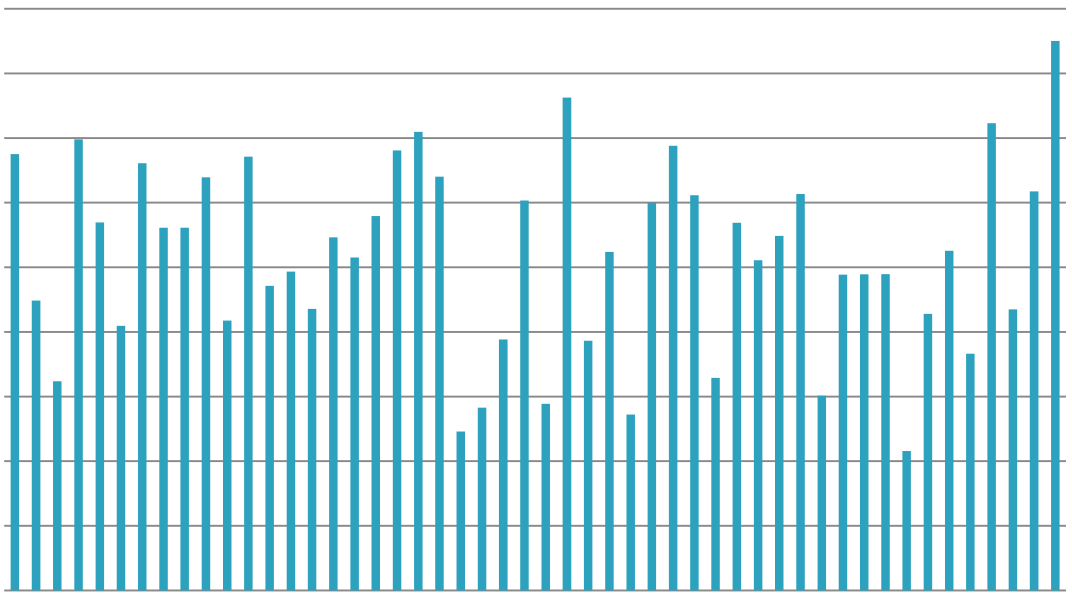
Distributions

- Uniform
- Gaussian
- Bucket
- Sorted
- Zero
- Stanford Models



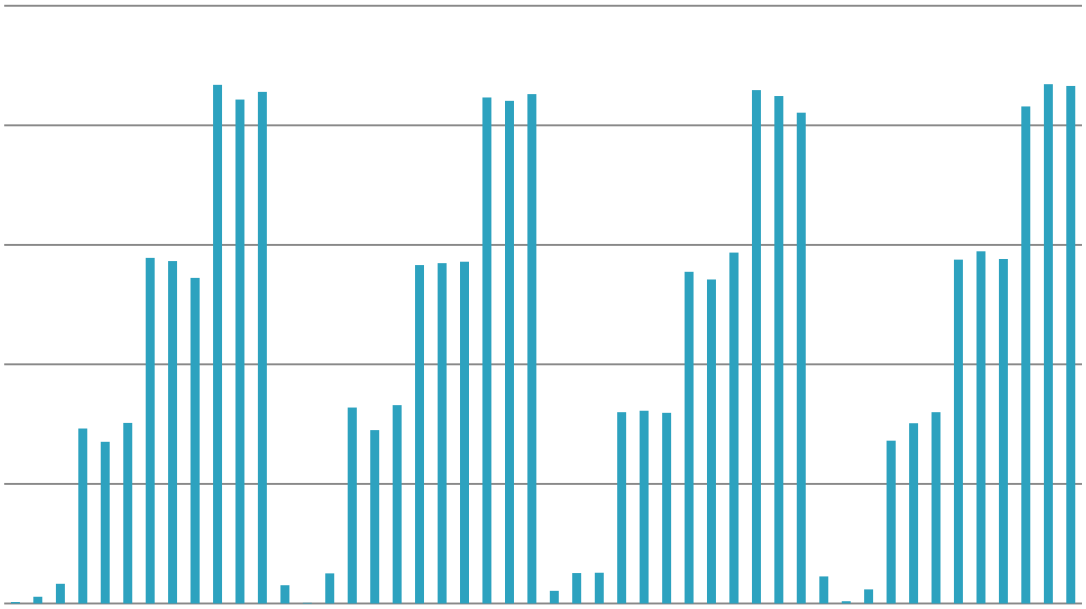
Distributions

- Uniform
- Gaussian
- Bucket
- Sorted
- Zero
- Stanford Models



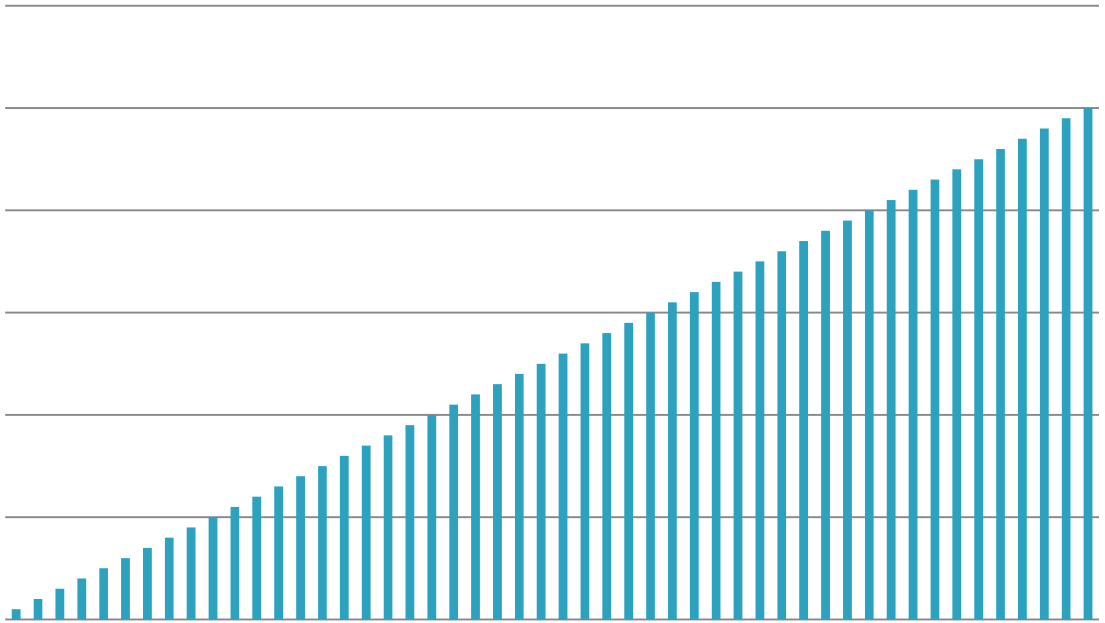
Distributions

- Uniform
- Gaussian
- Bucket
- Sorted
- Zero
- Stanford Models



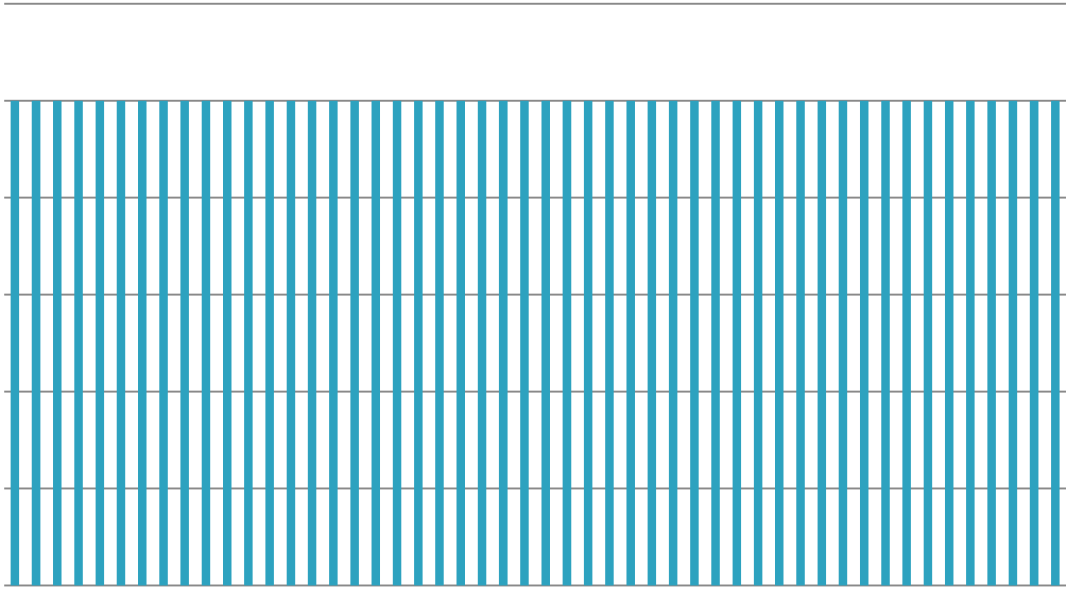
Distributions

- Uniform
- Gaussian
- Bucket
- Sorted
- Zero
- Stanford Models



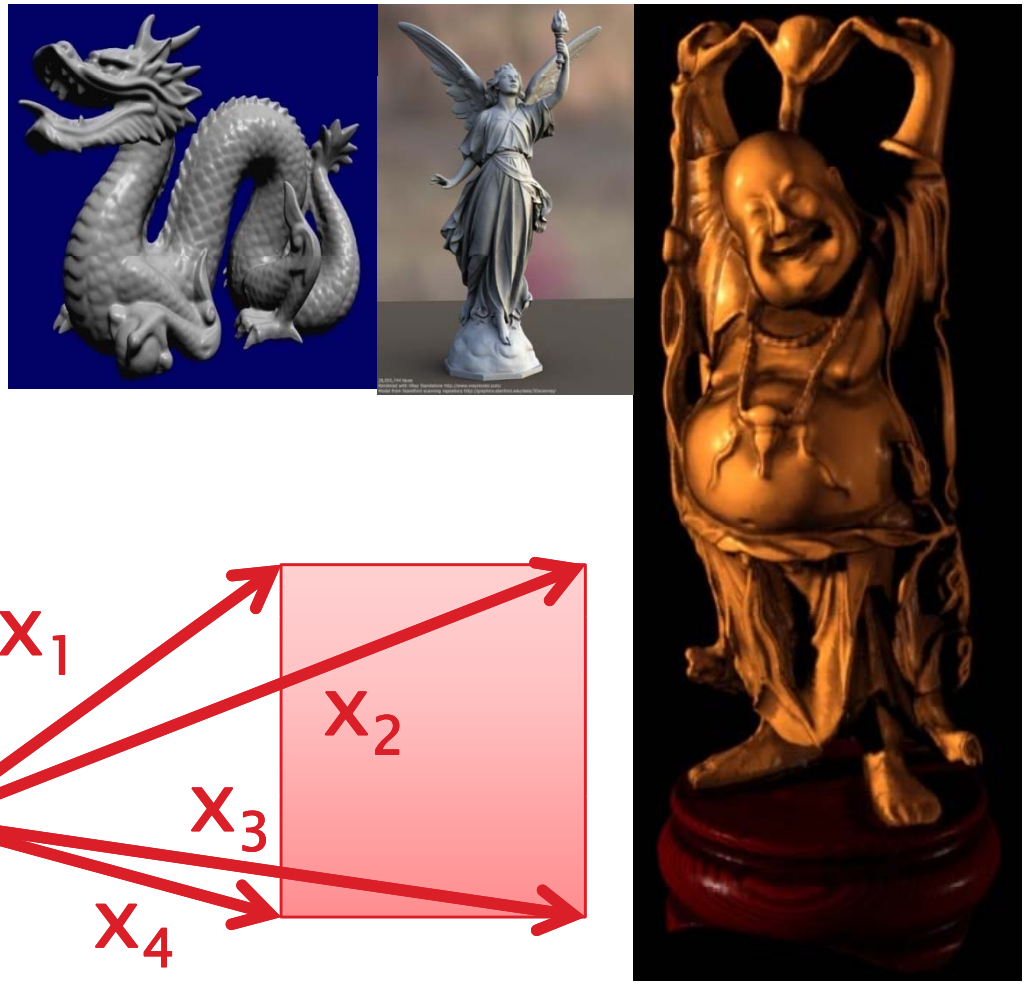
Distributions

- Uniform
- Gaussian
- Bucket
- Sorted
- Zero
- Stanford Models



Distributions

- Uniform
- Gaussian
- Bucket
- Sorted
- Zero
- Stanford Models



Pivot selection

- First Phase
  - Average of maximum and minimum element
- Second Phase
  - Median of first, middle and last element

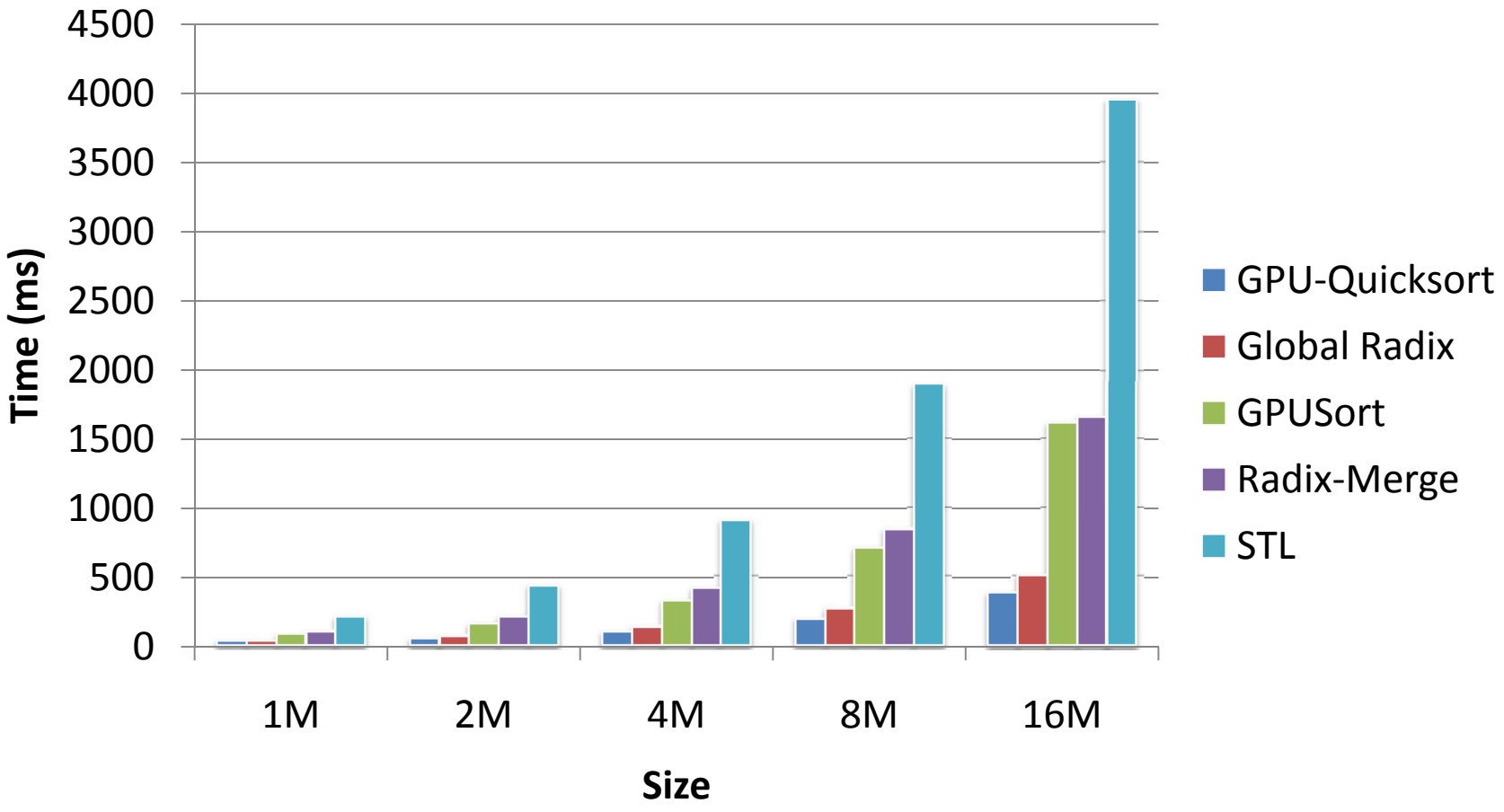
Hardware

- 8800GTX
  - 16 multiprocessors (128 cores)
  - 86.4 GB/s bandwidth

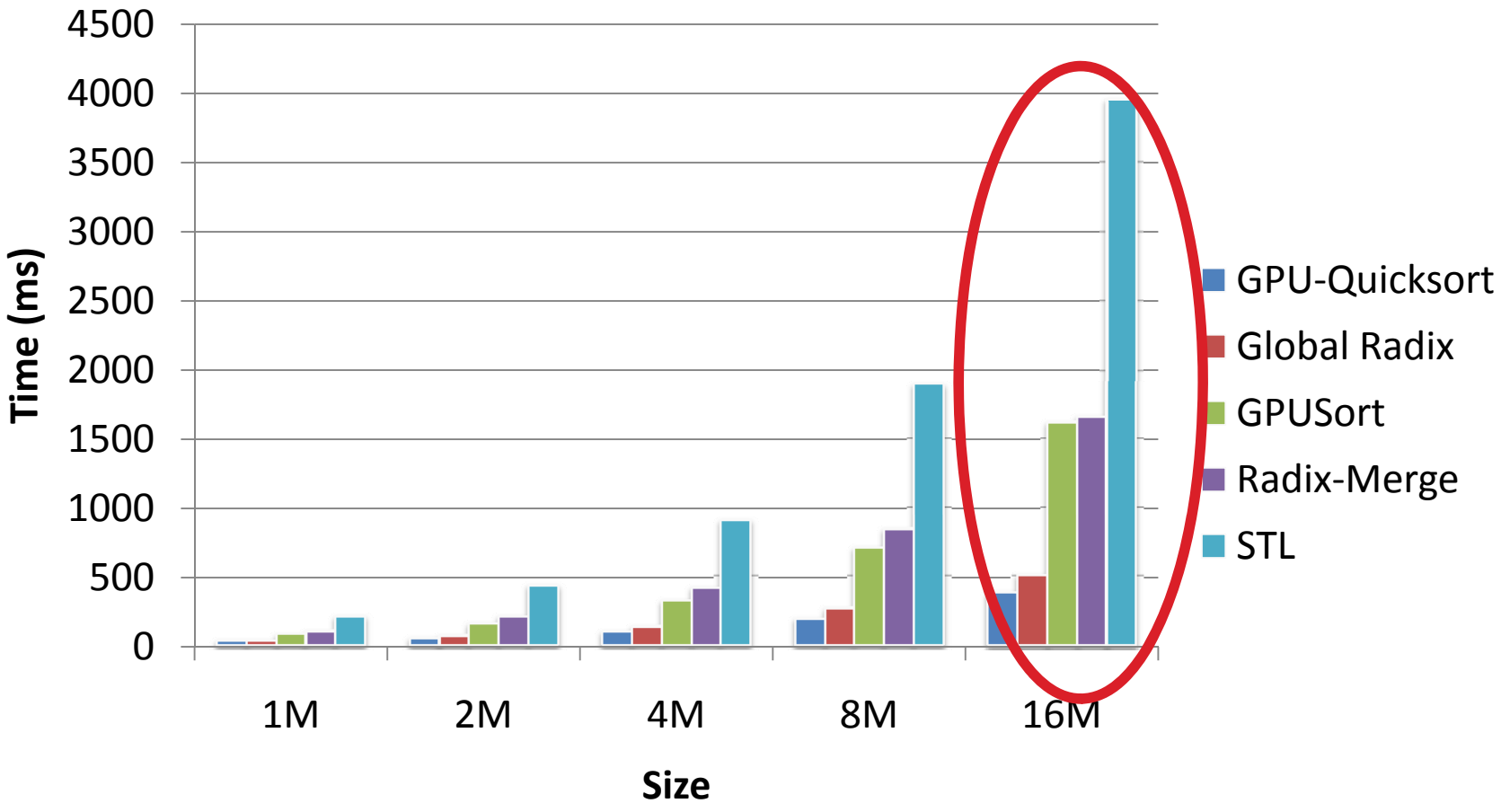
- 8600GTS
  - 4 multiprocessors (32 cores)
  - 32 GB/s bandwidth

- CPU-Reference
  - AMD Dual-Core Opteron 265 / 1.8 GHz

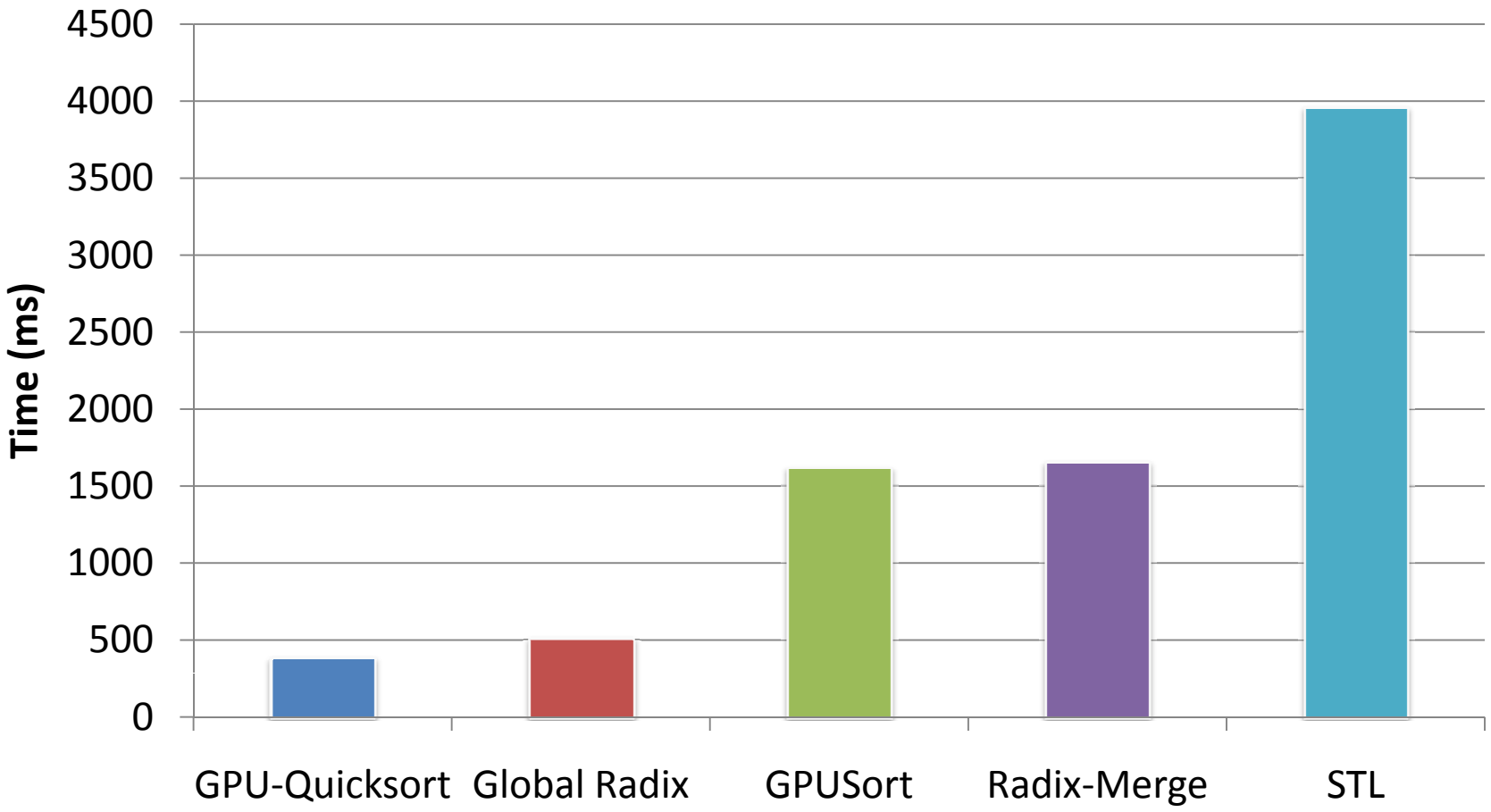
8800GTX – Uniform Distribution



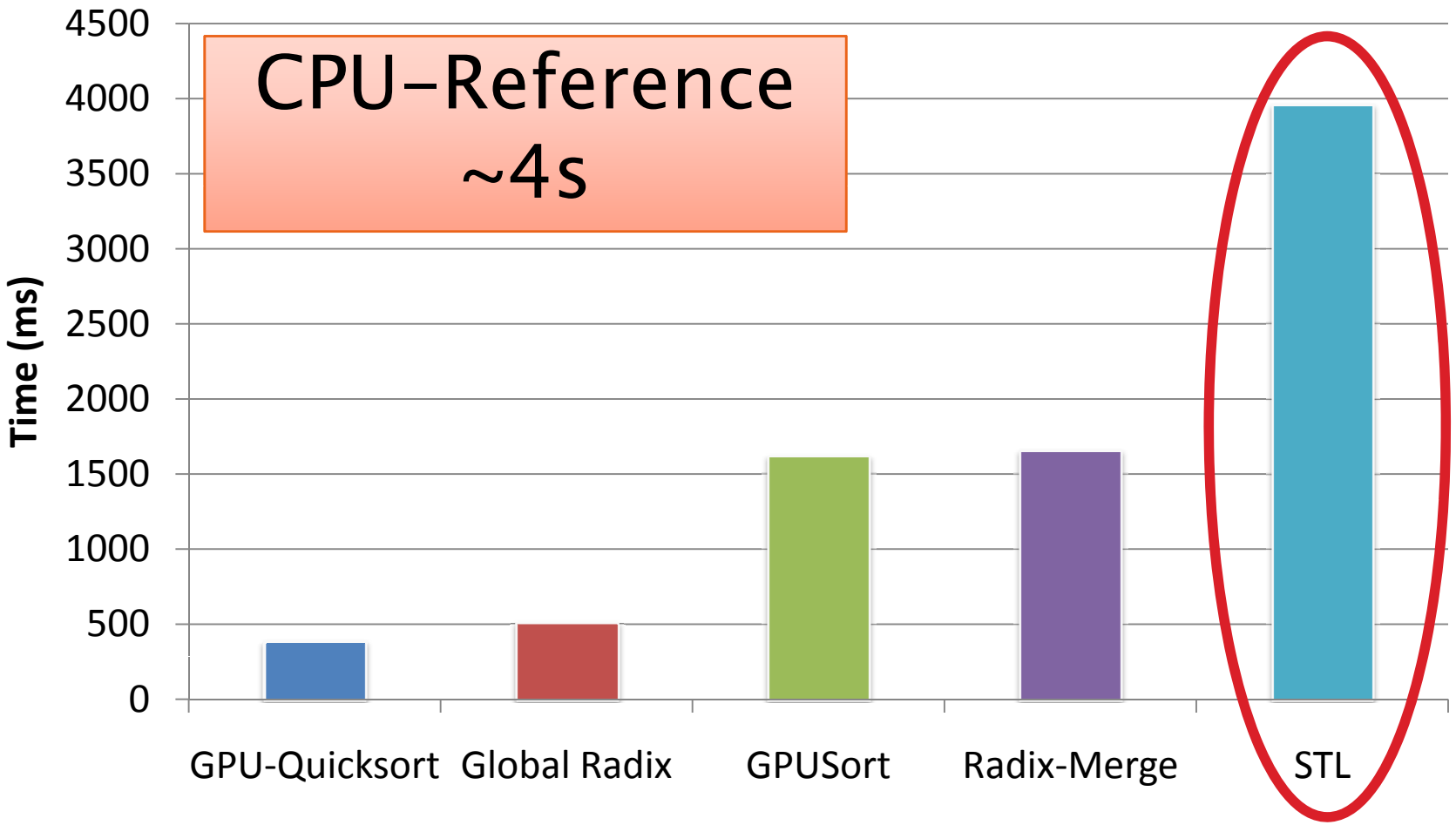
8800GTX – Uniform Distribution



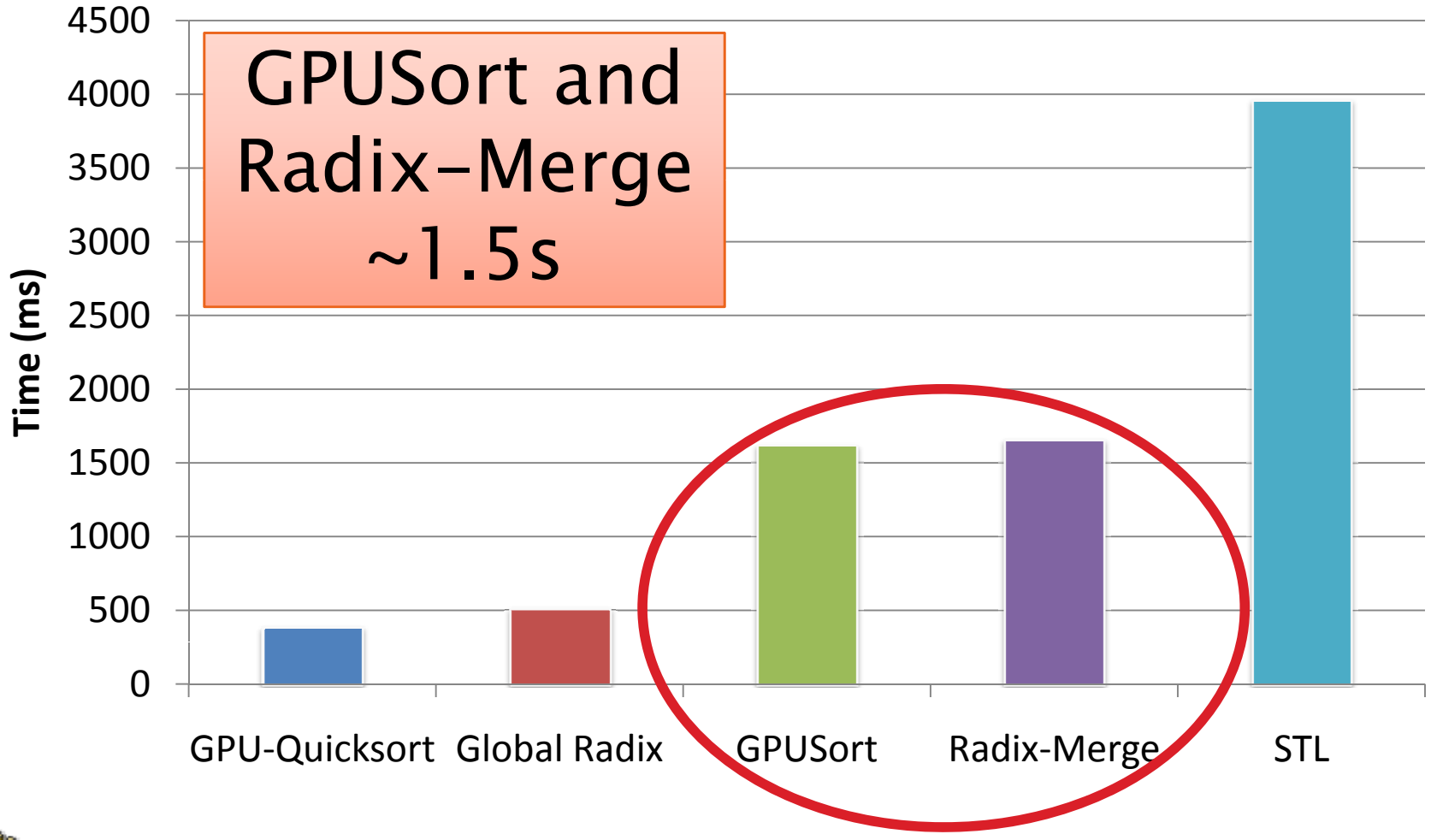
8800GTX – Uniform – 16M



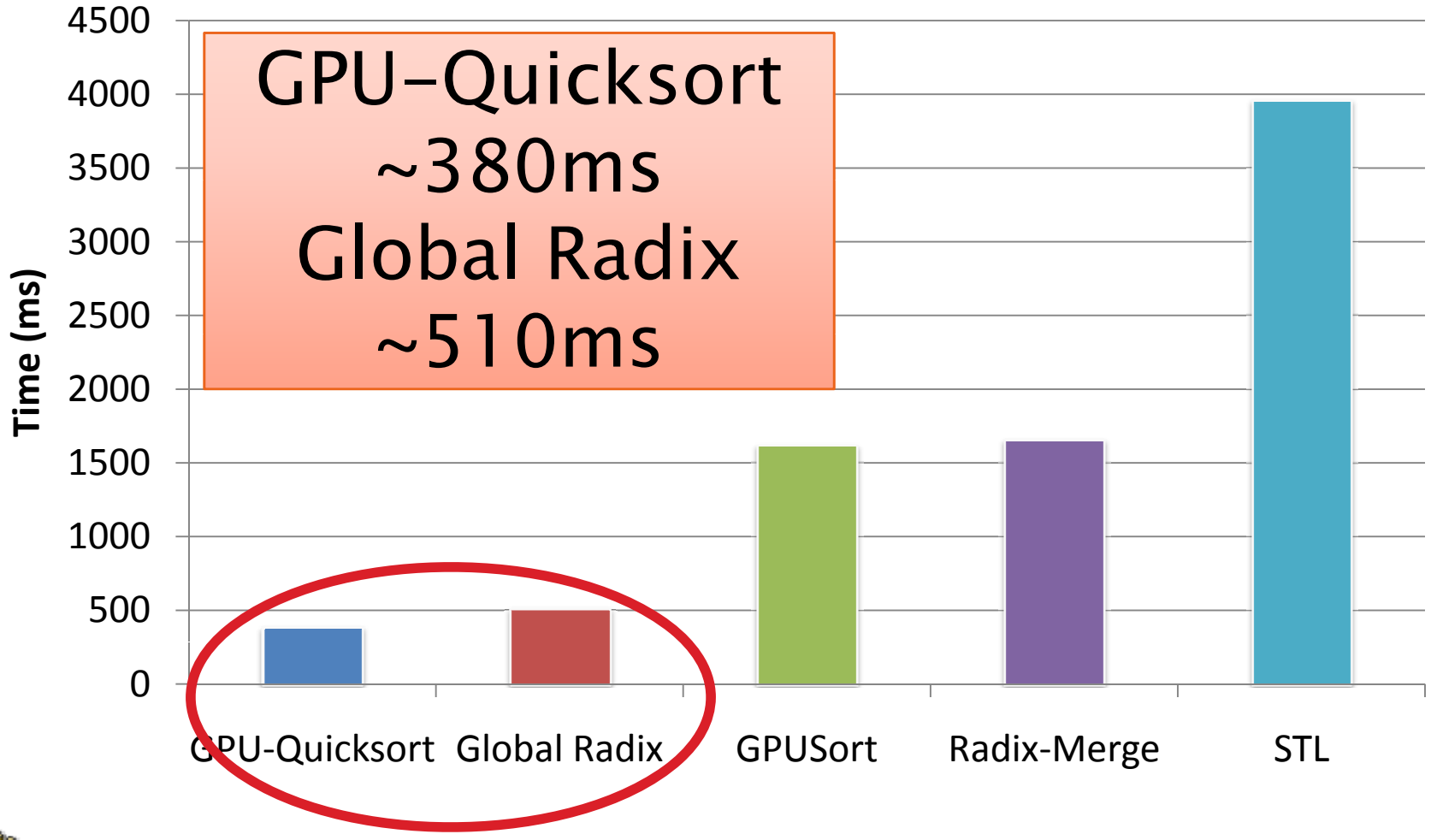
8800GTX – Uniform – 16M



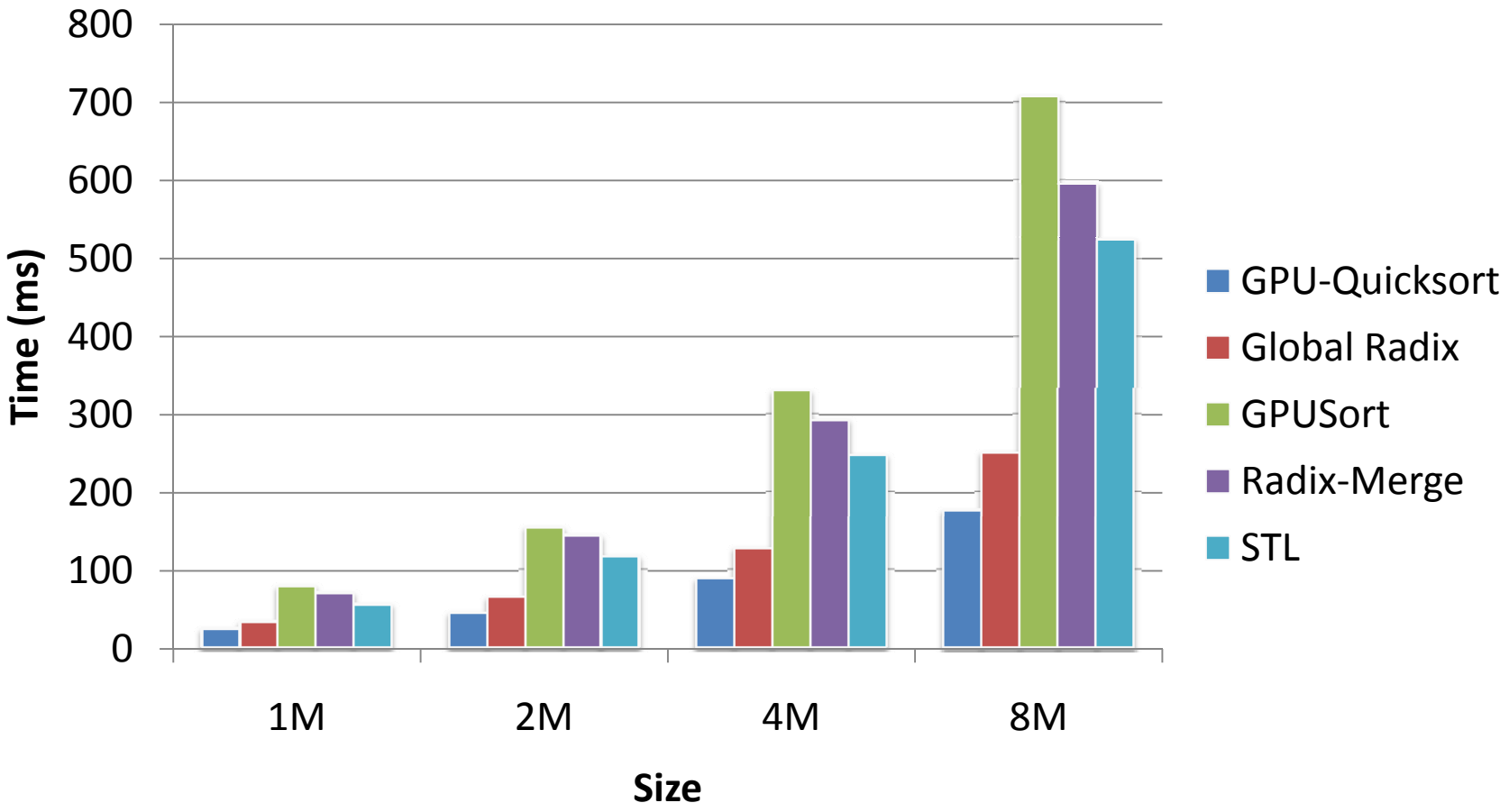
8800GTX – Uniform – 16M



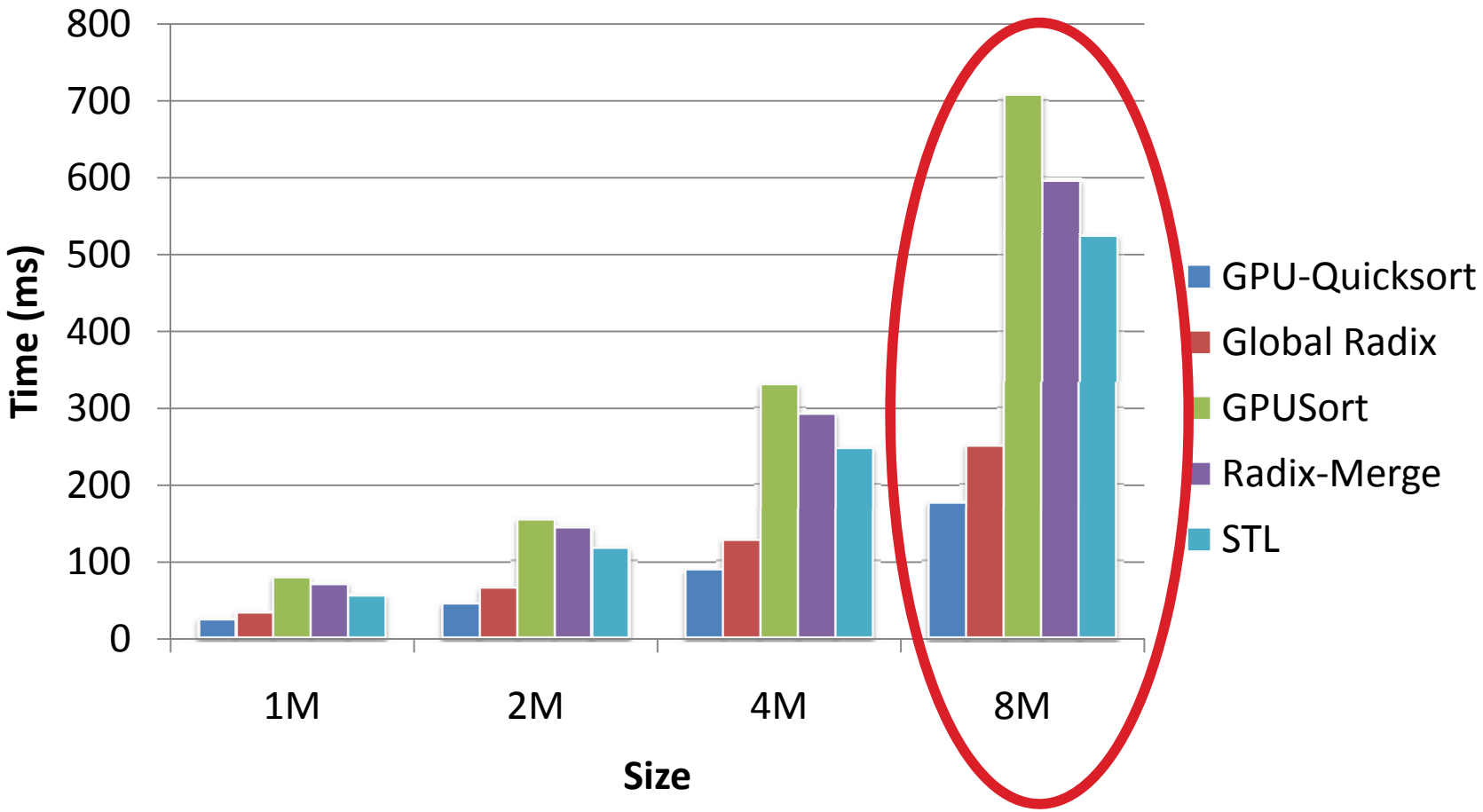
8800GTX – Uniform – 16M



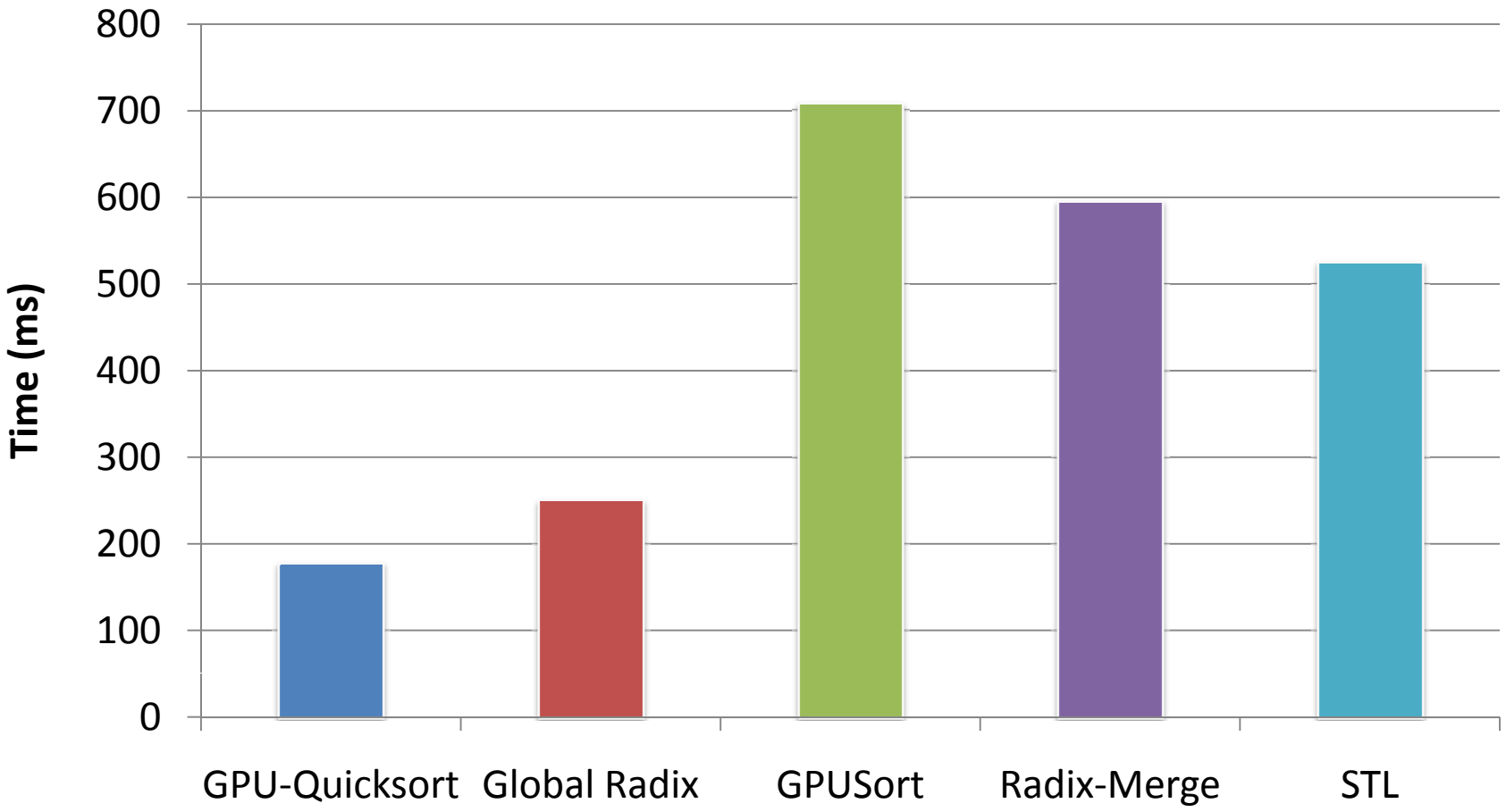
8800GTX – Sorted Distribution



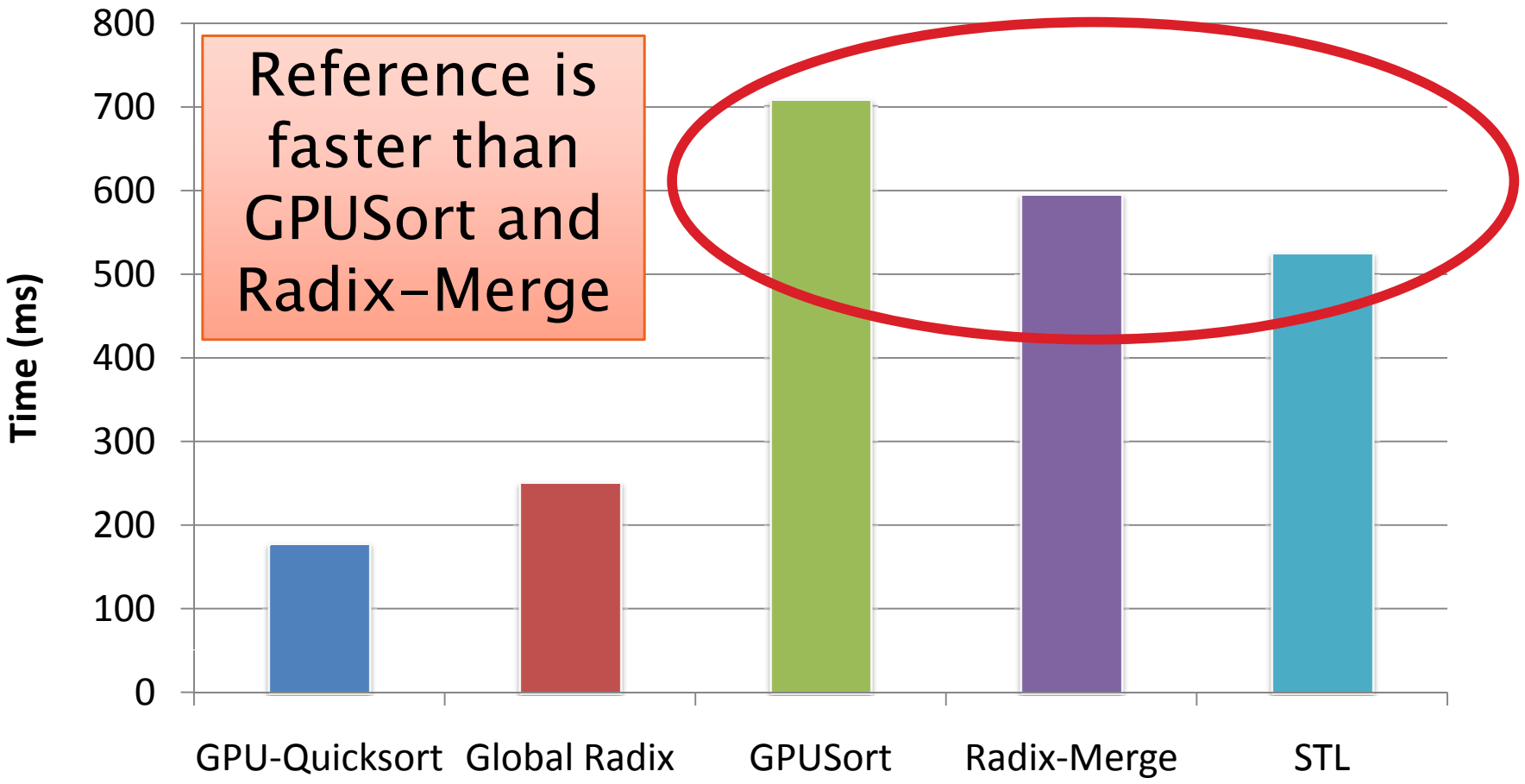
8800GTX – Sorted Distribution



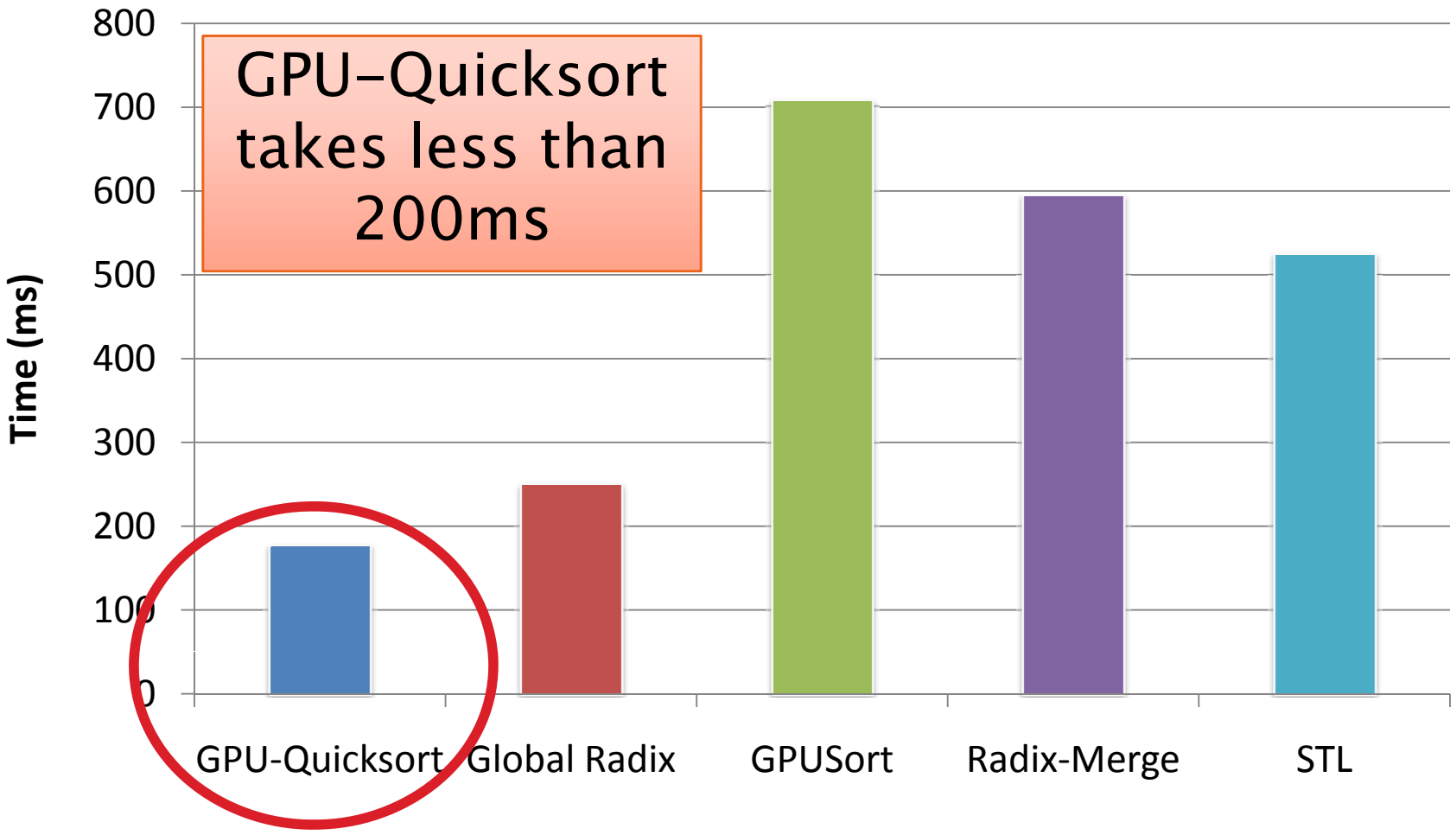
8800GTX – Sorted – 8M



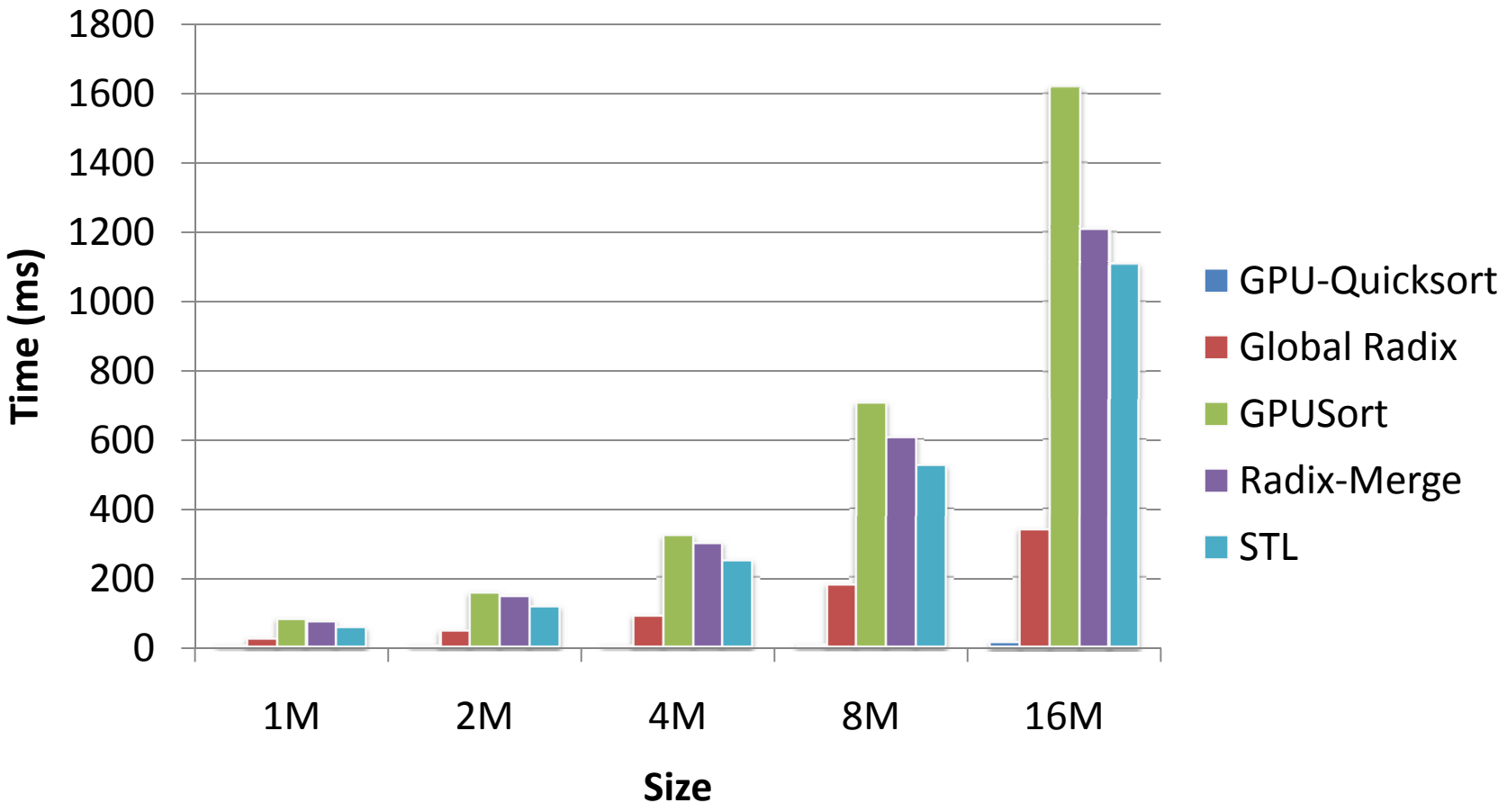
8800GTX – Sorted – 8M



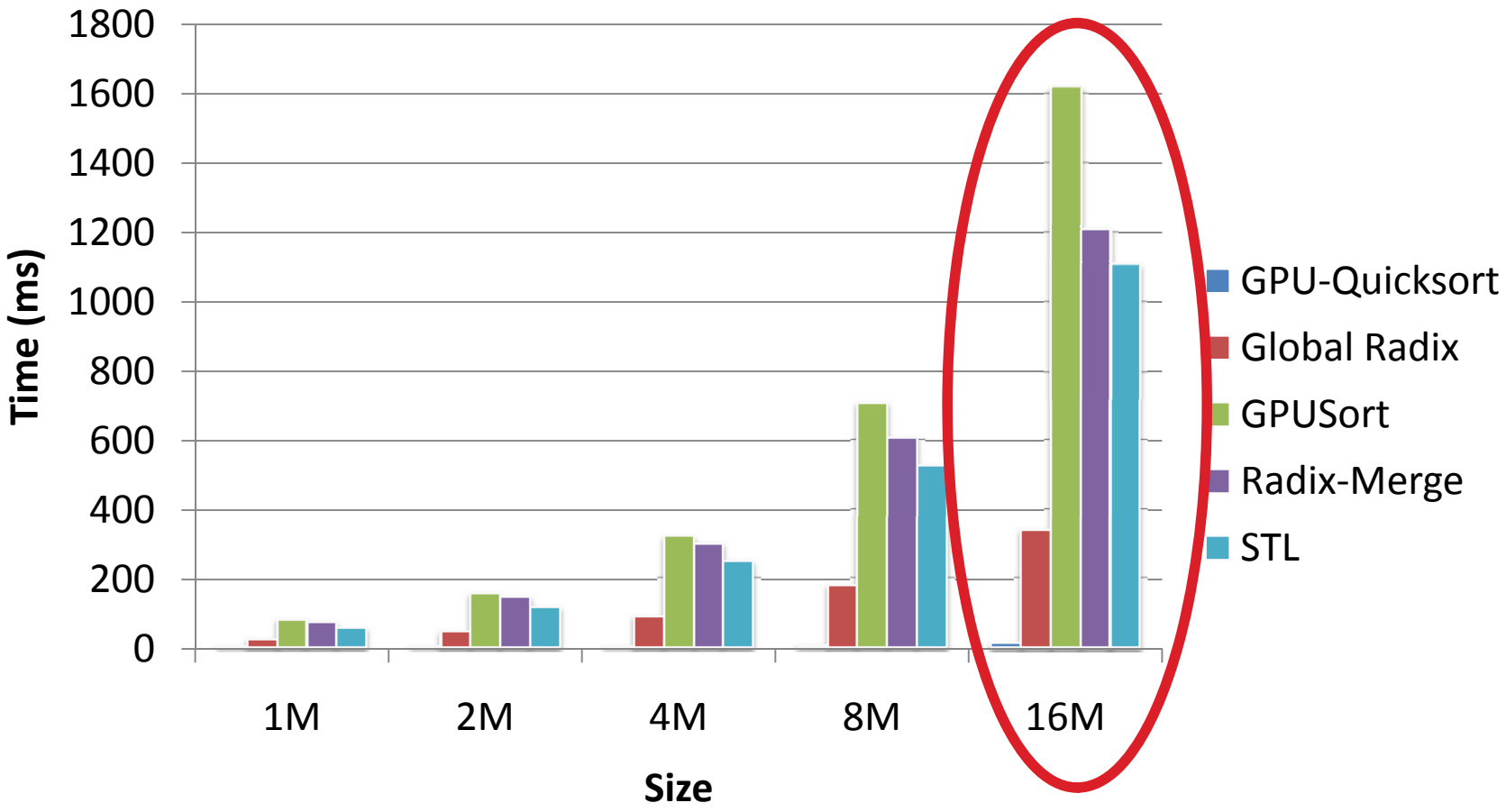
8800GTX – Sorted – 8M



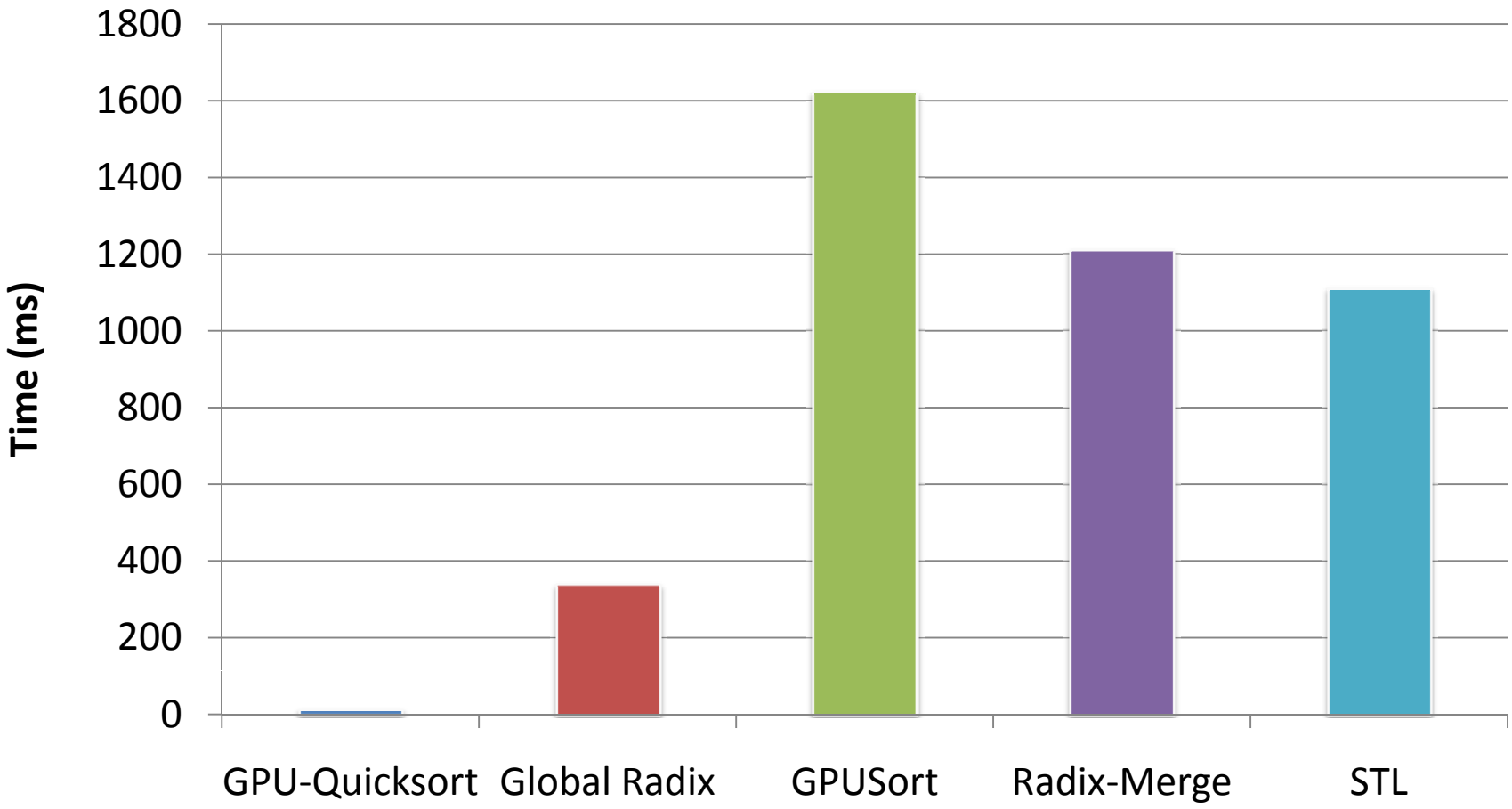
8800GTX – Zero Distribution



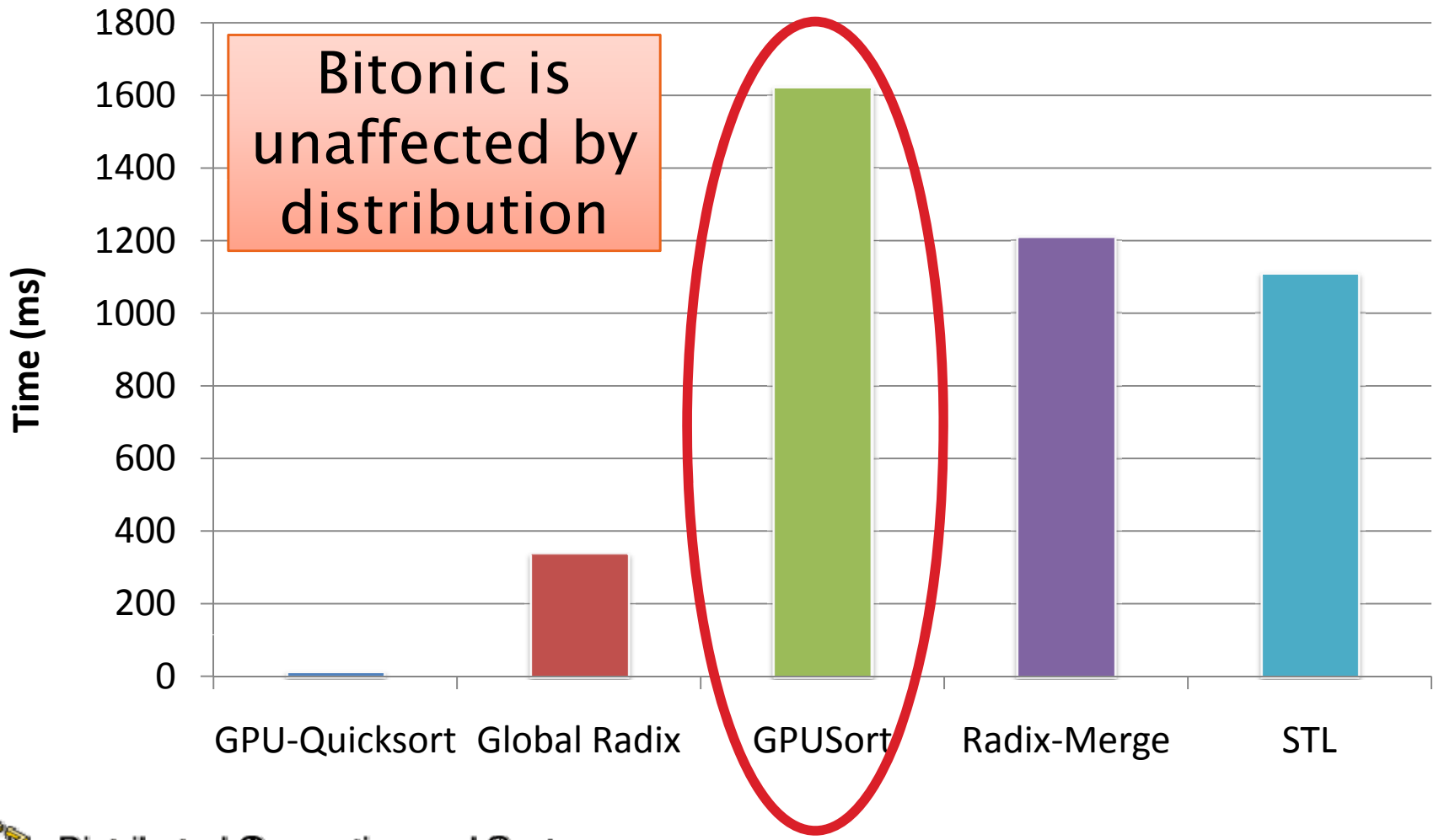
8800GTX – Zero Distribution



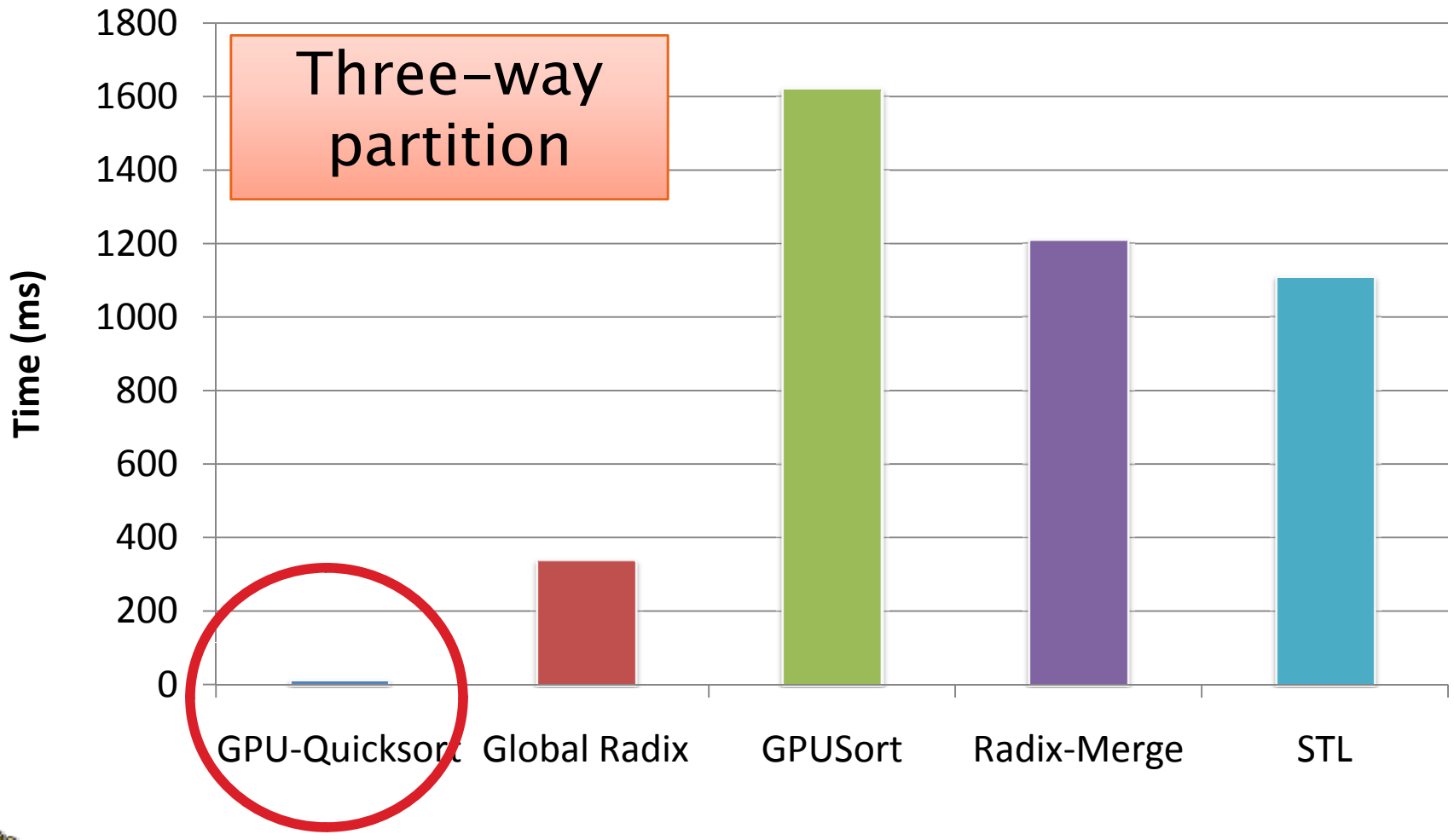
8800GTX – Zero – 16M



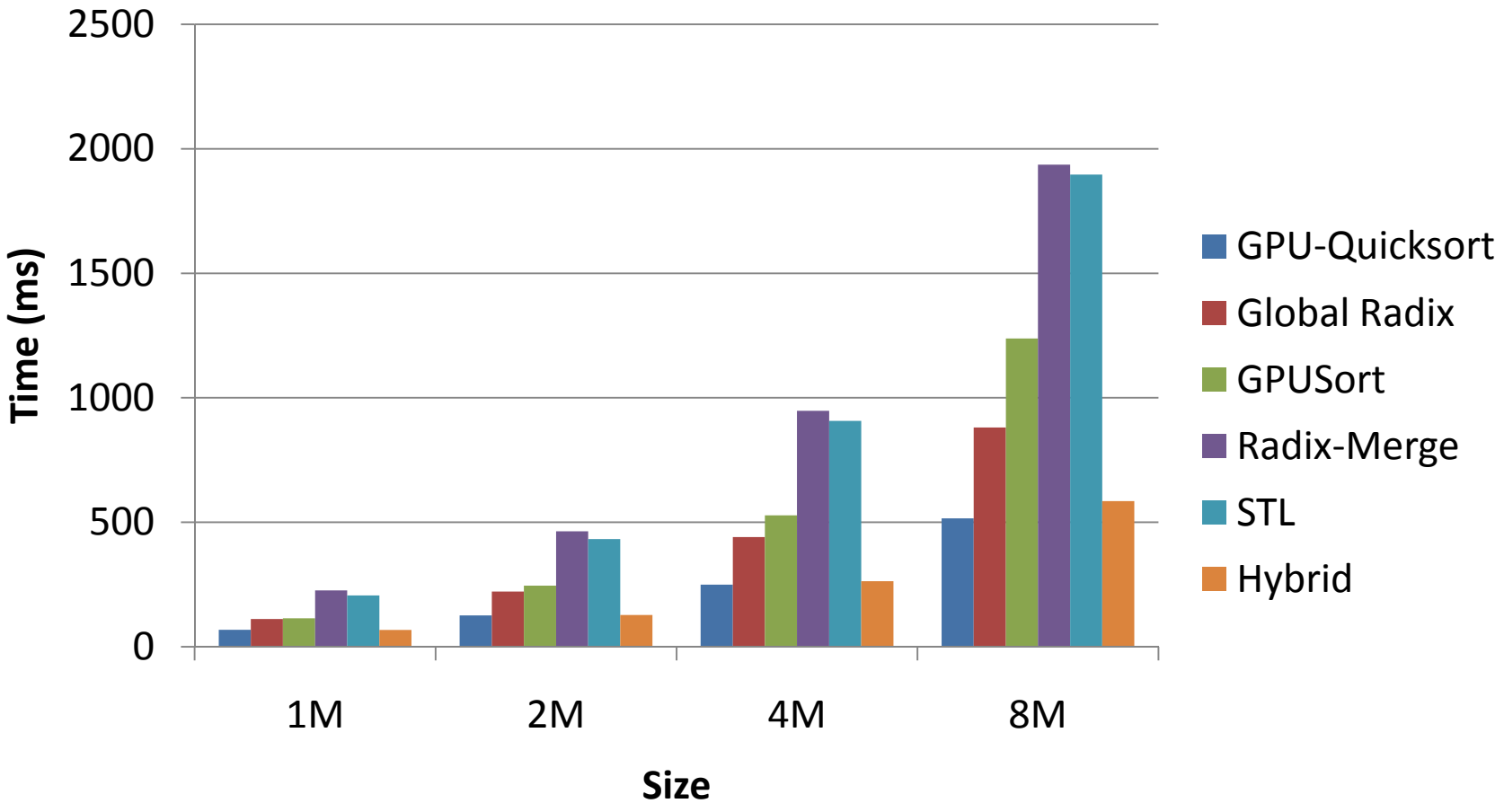
8800GTX – Zero – 16M



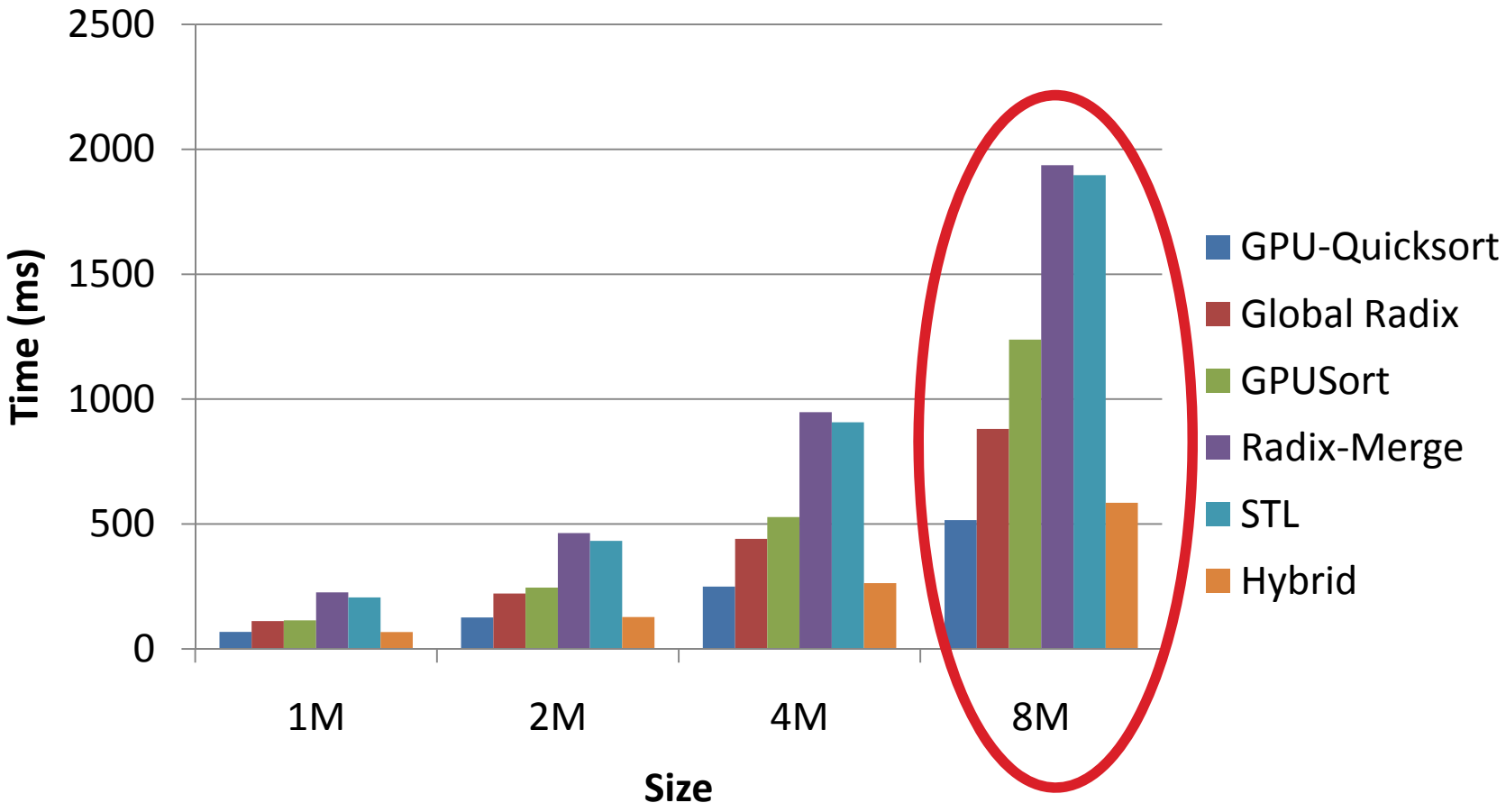
8800GTX – Zero – 16M



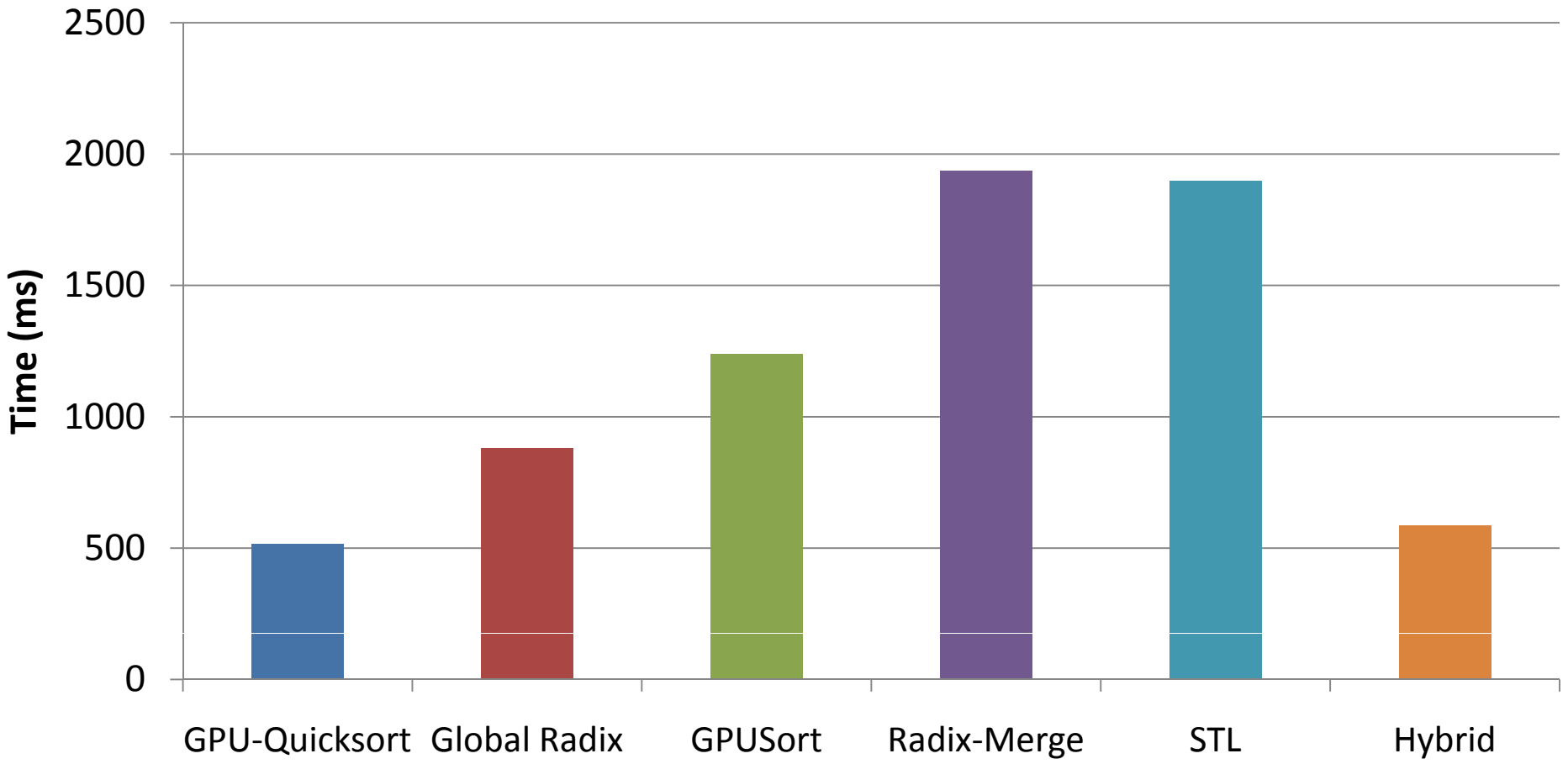
8600GTS – Uniform Distribution



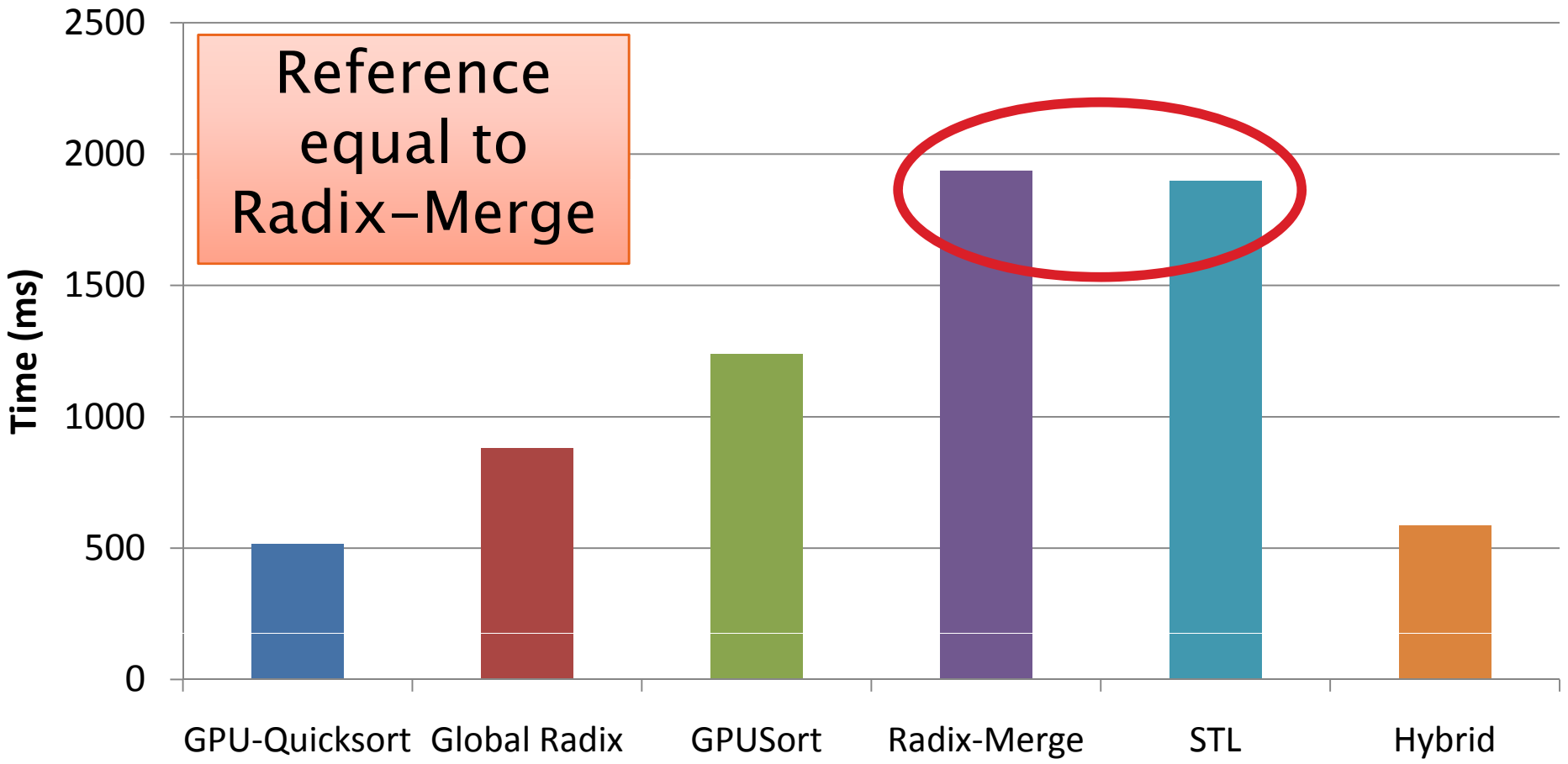
8600GTS – Uniform Distribution



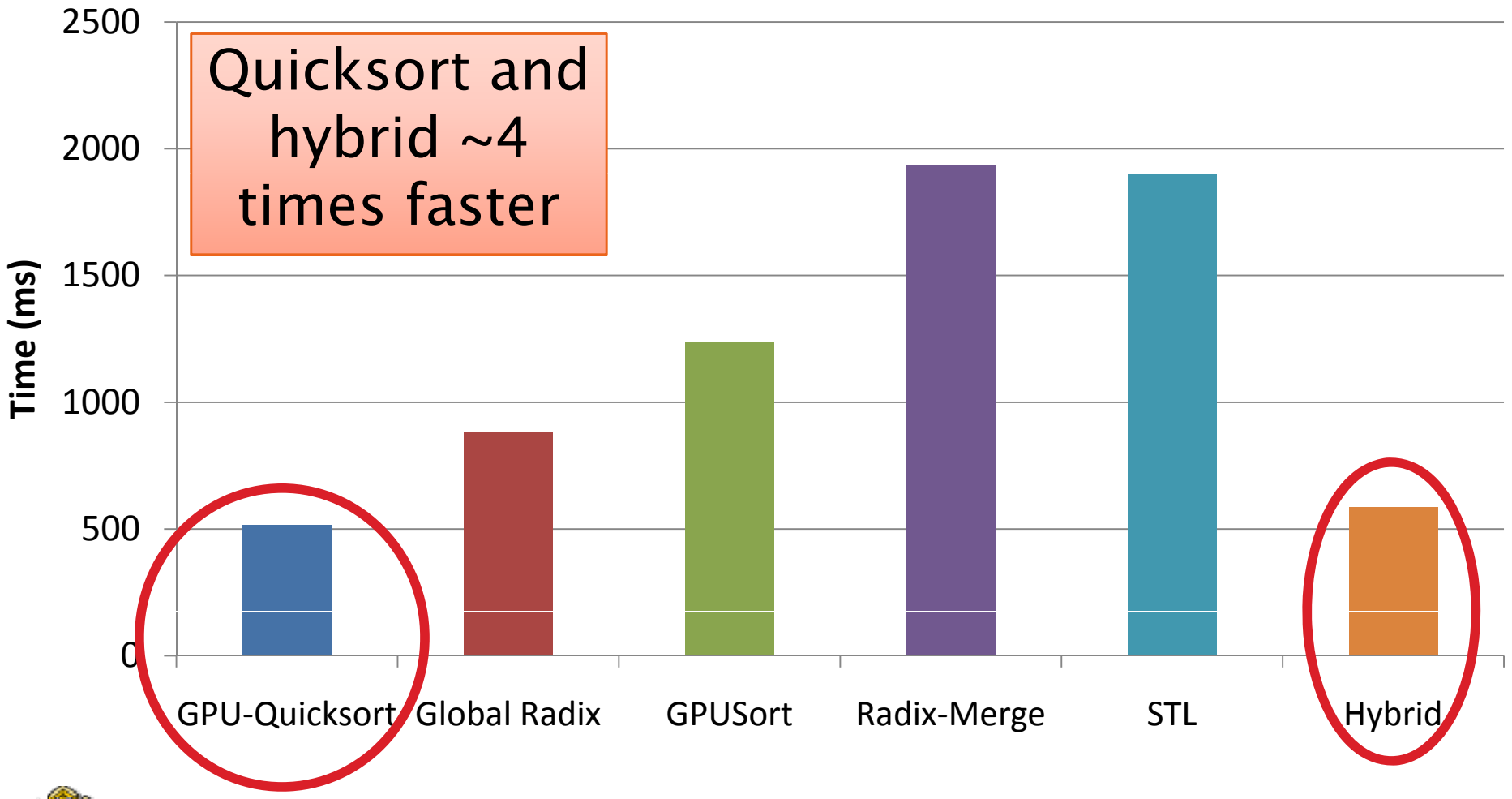
8600GTS – Uniform – 8M



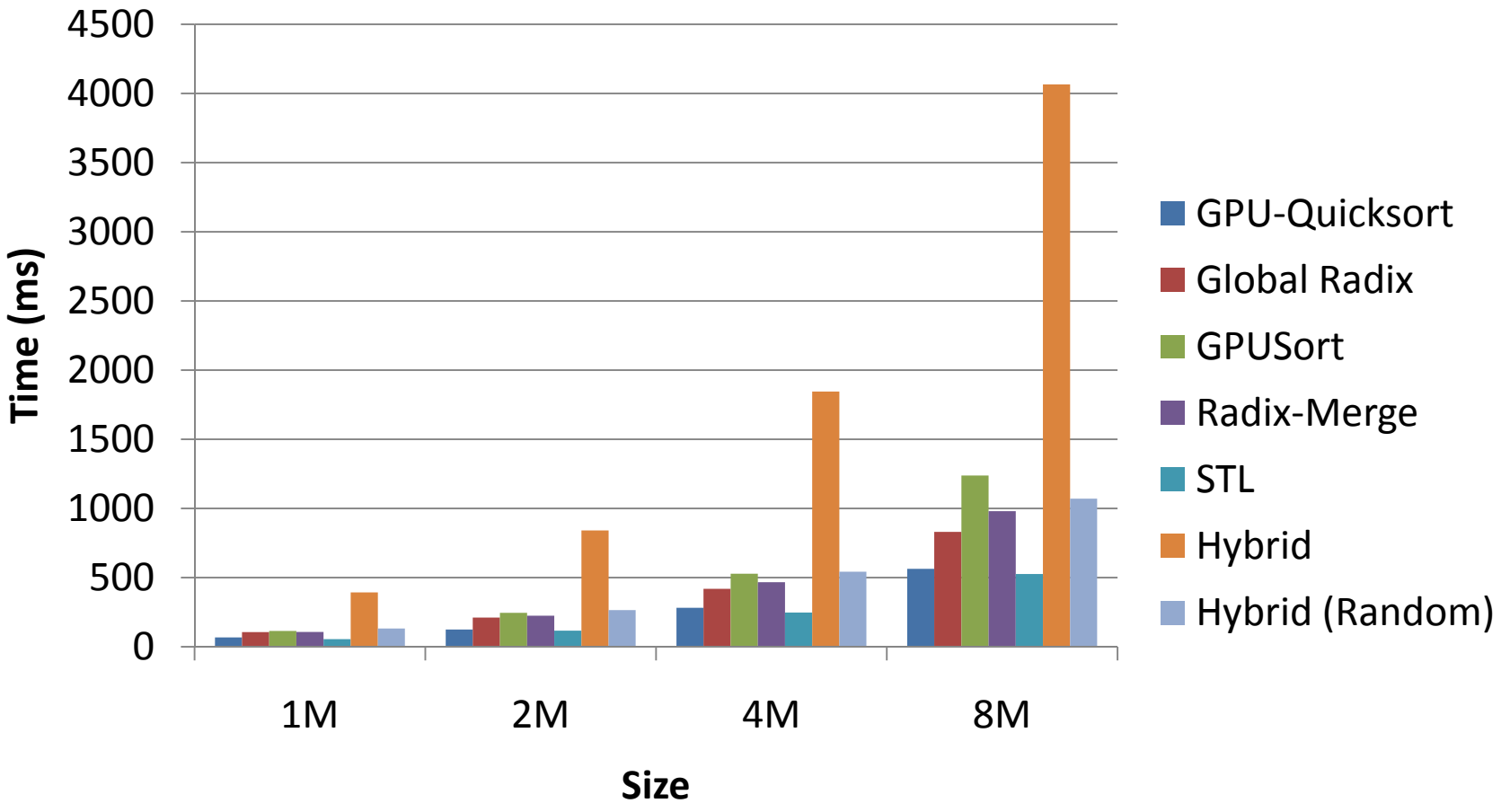
8600GTS – Uniform – 8M



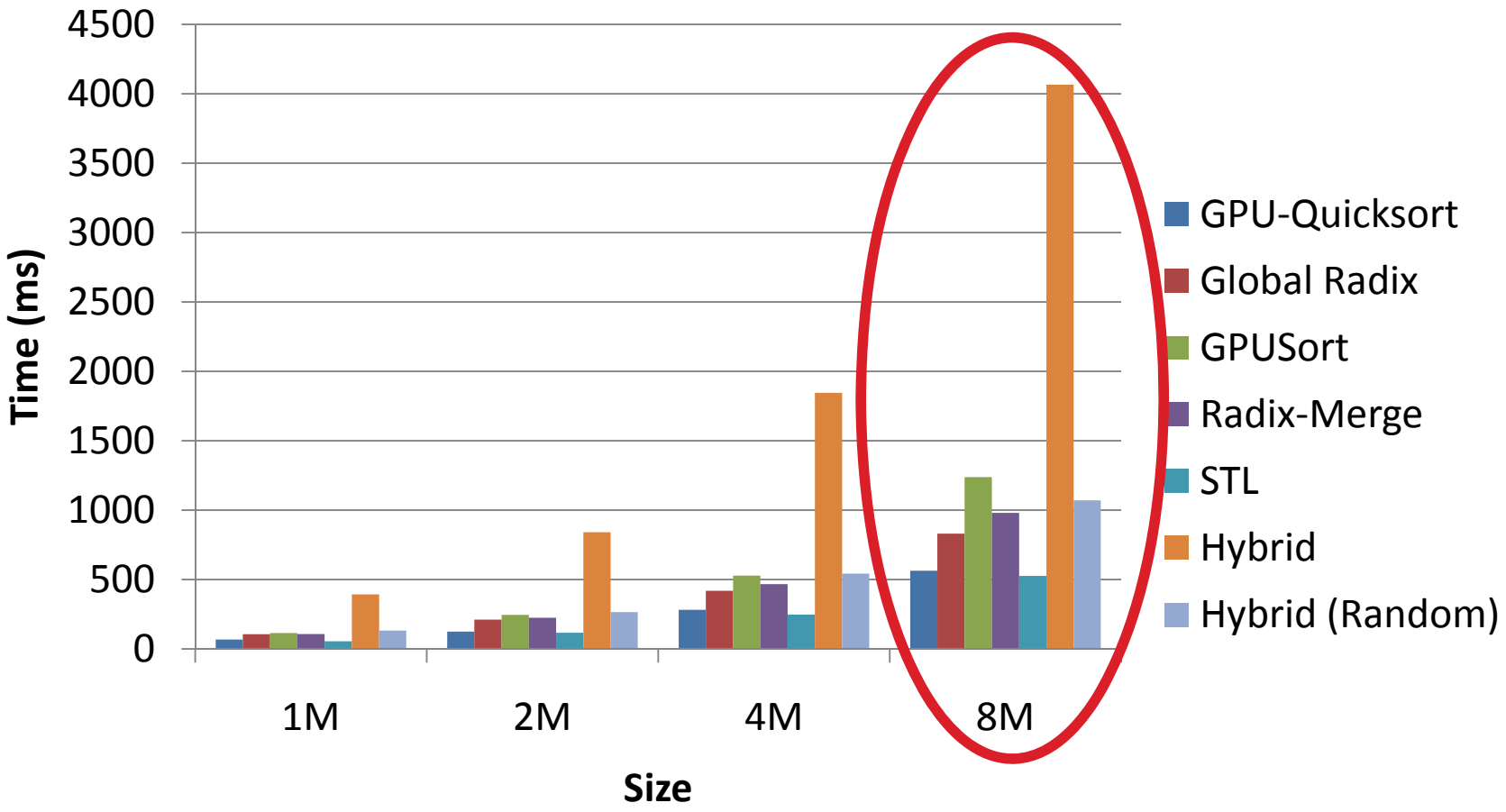
8600GTS – Uniform – 8M



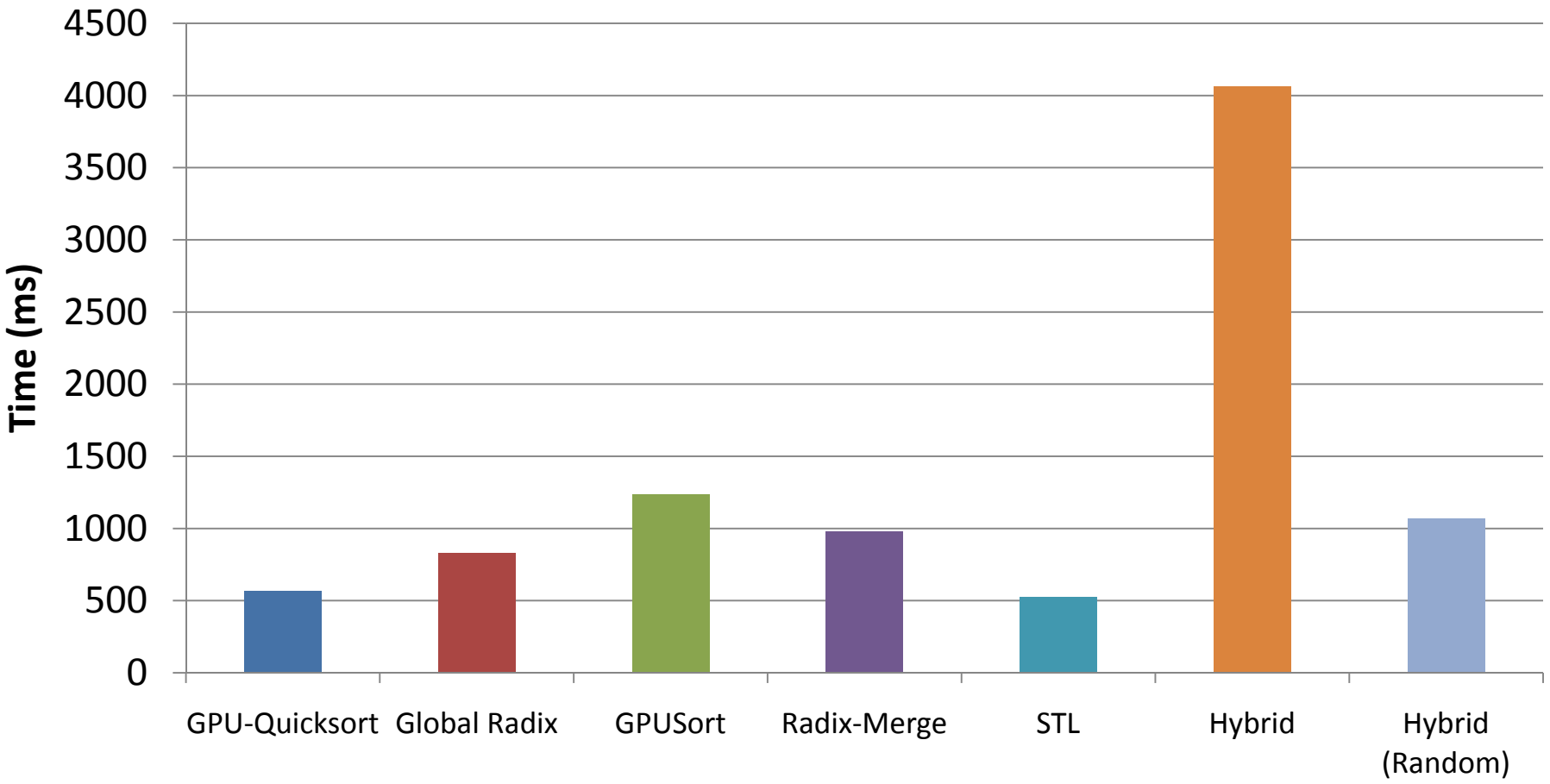
8600GTS – Sorted Distribution



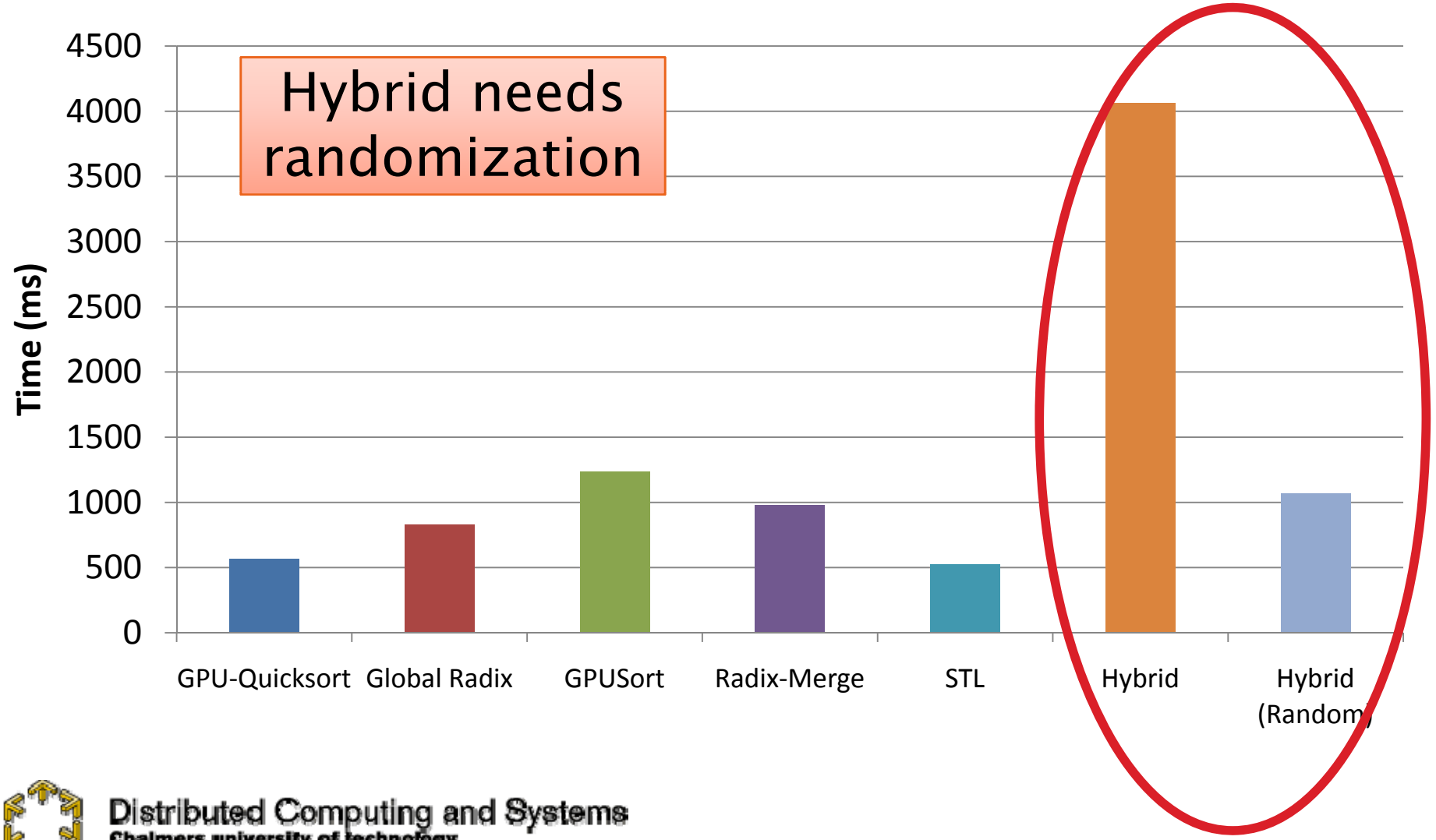
8600GTS – Sorted Distribution



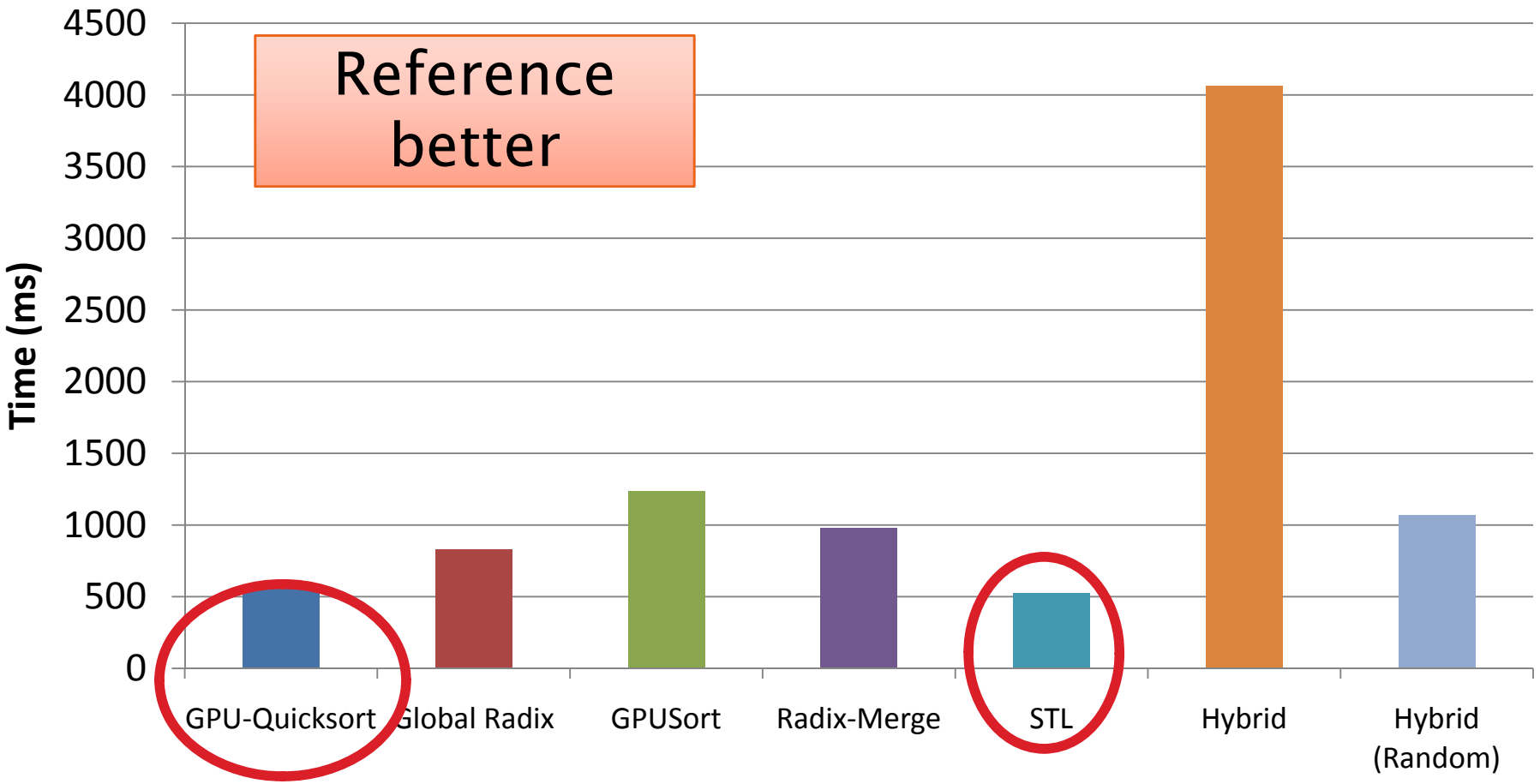
8600GTS – Sorted – 8M



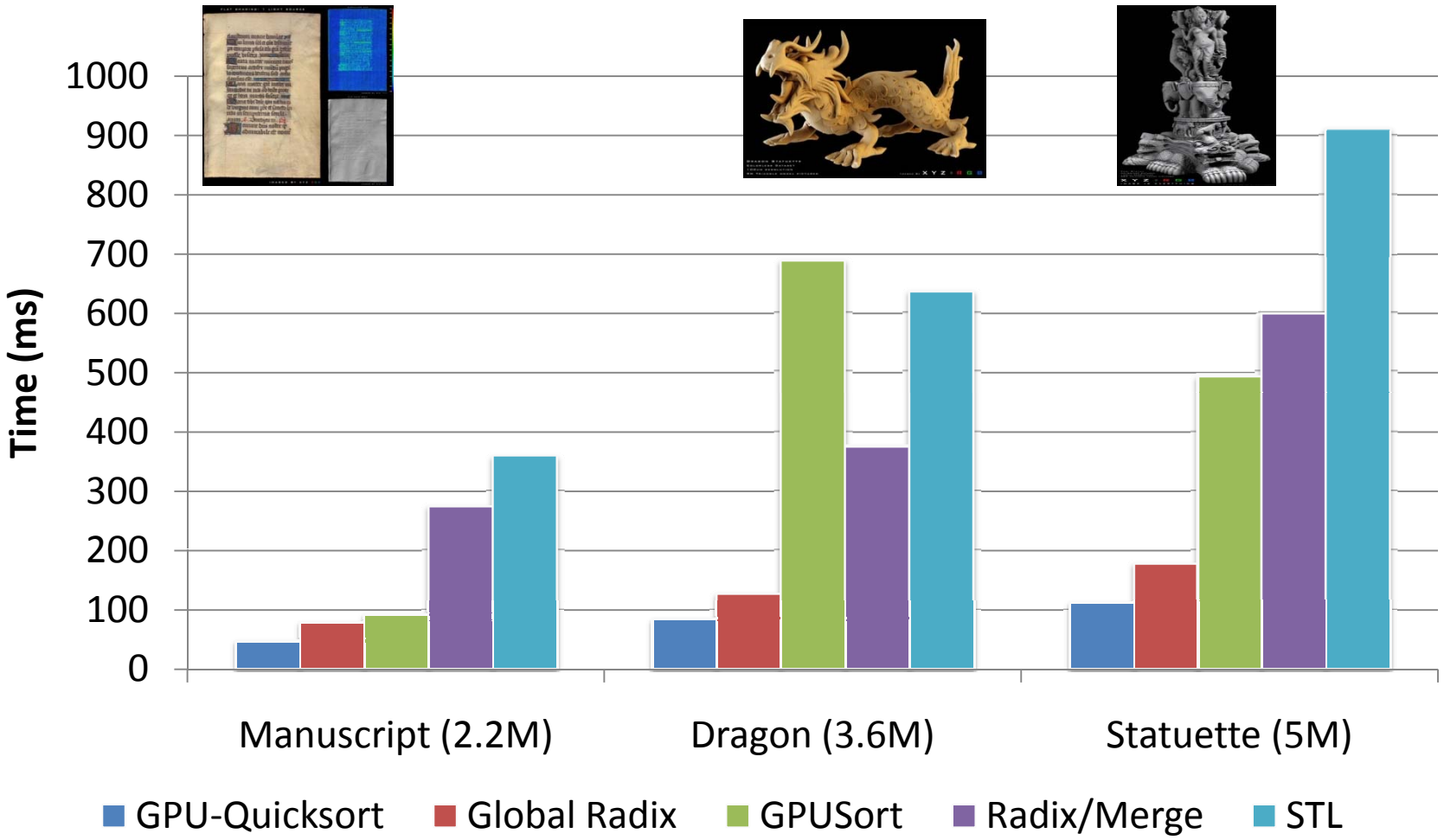
8600GTS – Sorted – 8M



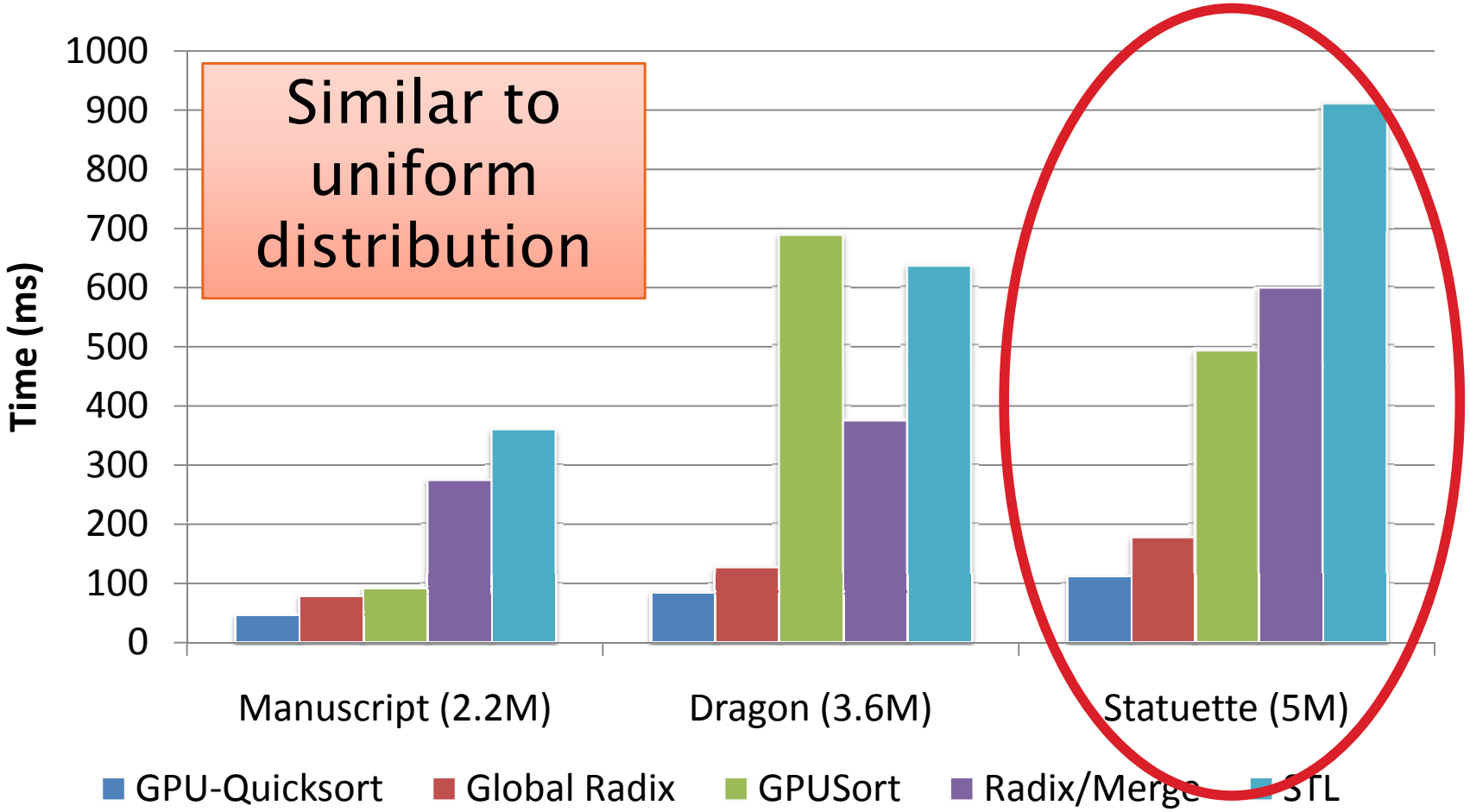
8600GTS – Sorted – 8M



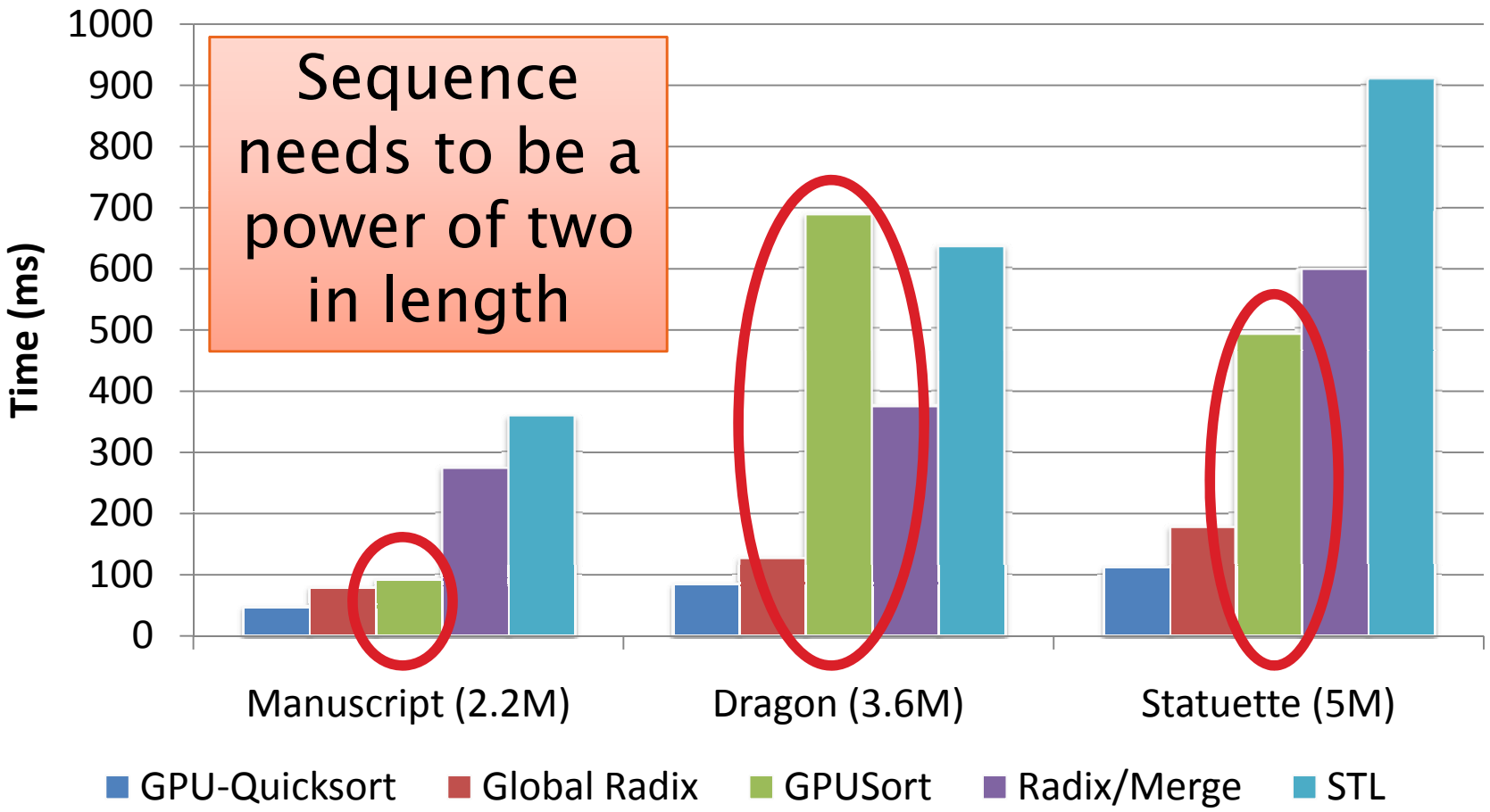
Visibility Ordering – 8800GTX



Visibility Ordering – 8800GTX



Visibility Ordering – 8800GTX



Conclusions

- Minimal, manual cache
  - Used only for prefix sum and bitonic
- 32-word SIMD instruction
  - Main part executes same instructions
- Coalesced memory access
  - All reads coalesced
- No block synchronization
  - Only required in first phase
- Expensive synchronization primitives
  - Two passes amortizes cost

Conclusions

- Quicksort is a viable sorting method for graphics processors and can be implemented in a data parallel way
- It is competitive

Thank you!

www.cs.chalmers.se/~cederman