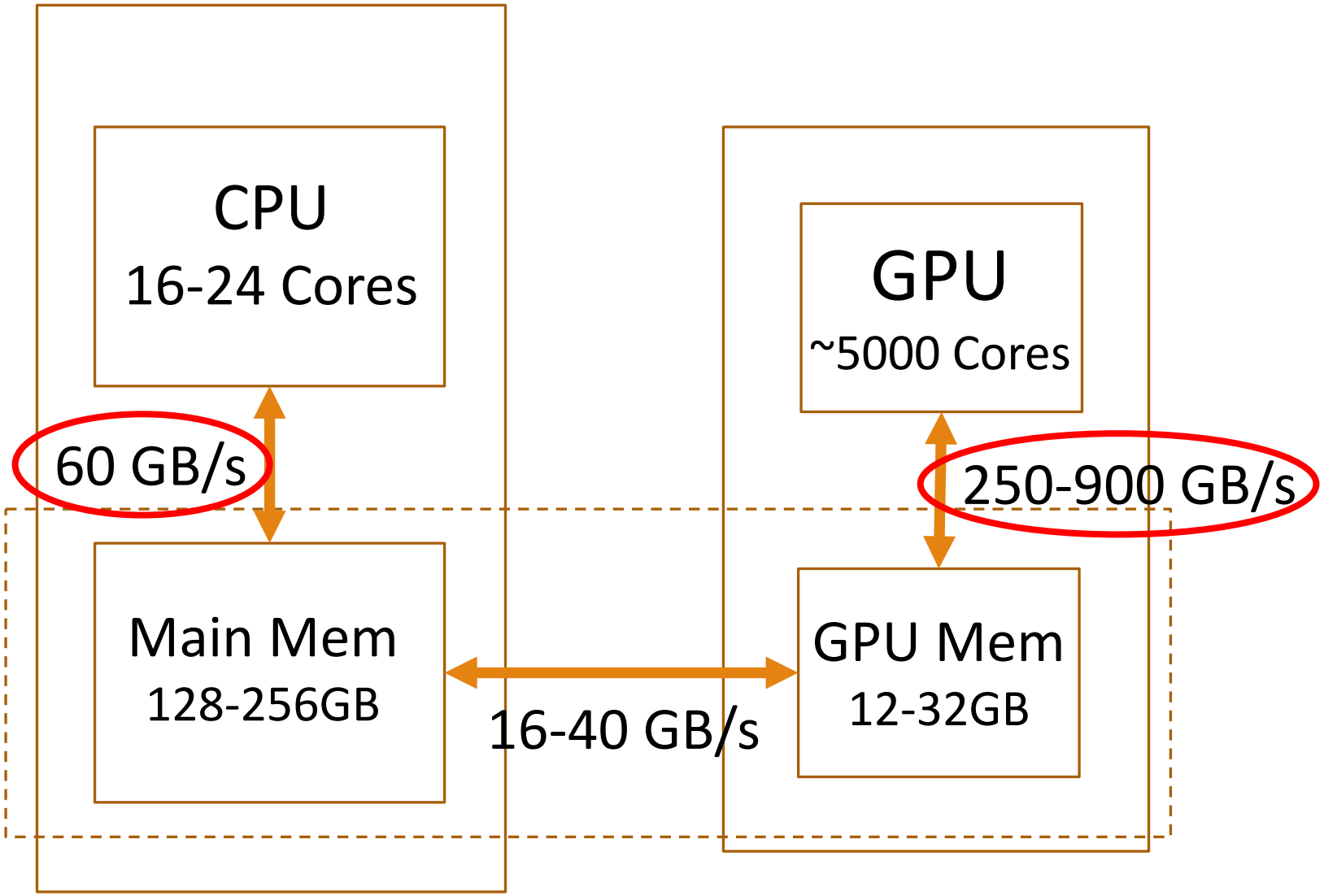
Efficient Top-K Query Processing on Massively Parallel Hardware

ANIL SHANBHAG, HOLGER PIRK, SAM MADDEN


amazon
web services


Microsoft Azure


CPU
16-24 Cores
GPU
~5000 Cores
60 GB/s
250-900 GB/s
Main Mem
128-256GB
16-40 GB/s
GPU Mem
12-32GB


MAPD


kinetica


SQREAM


BLAZINGDB

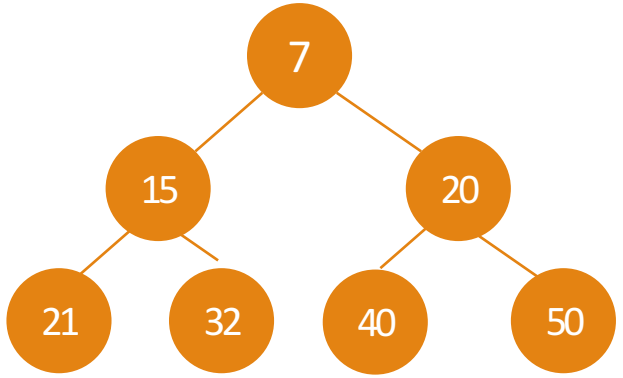
Top-K

```
SELECT id FROM tweets WHERE tweet_time ∈ [X,Y]
ORDER BY retweet_count + 0.5*likes_count DESC LIMIT K
```

Typical K is 5-100

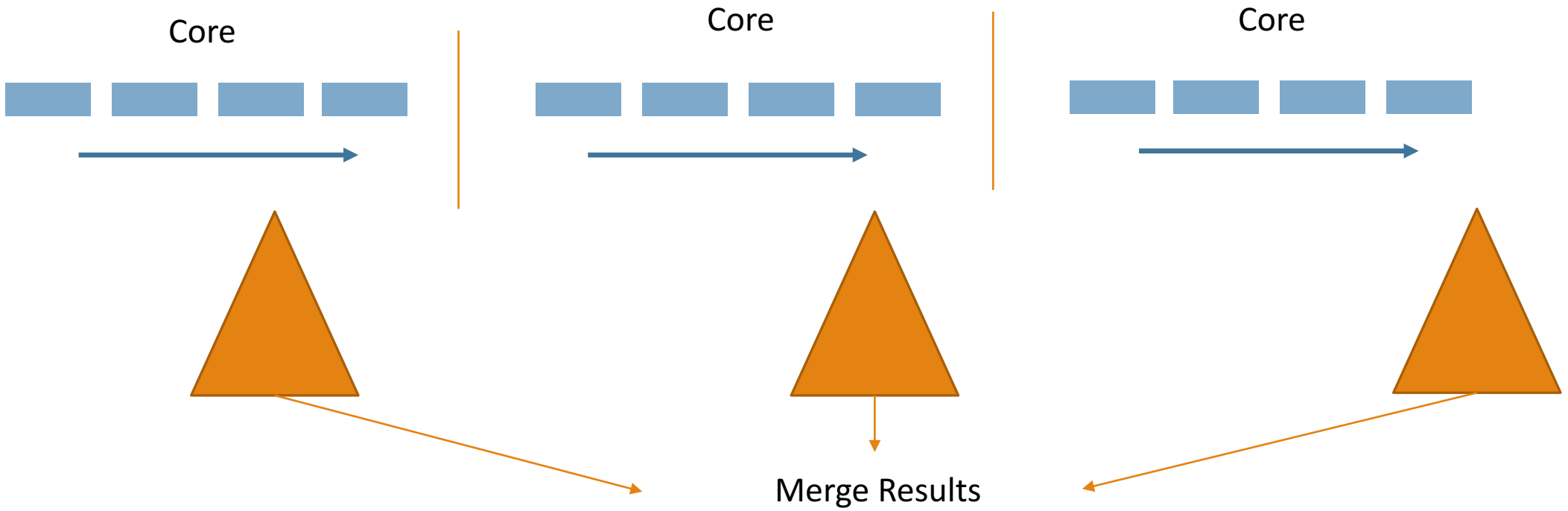
Top-K

Classic Sequential Algorithm:
Use a min-heap of size k to maintain the top-k items



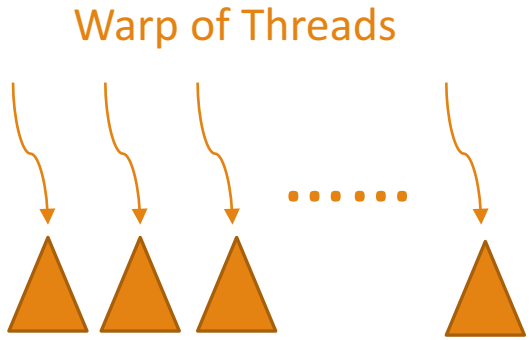
Partition and Merge

On Multi-core CPU: Partition data



On GPU

Does not work well on GPU execution model



PROBLEMS !

- Significant thread divergence
- Maintaining heap of size k per thread limits performance

MAPD

TensorFlow

{} ArrayFire

Intuition

	Sequential	Parallel
Top-K	Priority Queue	**Bitonic Top-K**
Sort	Heap Sort	Bitonic Sort

Sort + Top-K

Heap Per-Thread

Radix-Select

Bucket-Select

Bitonic Top-K

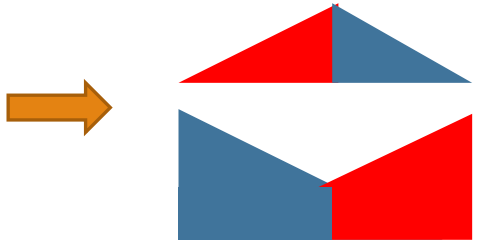
Bitonic Sequence

Two monotonic sequences

Sequence S = <a_0, a_1, a_2 … a_{n-1}> such that

- $a_0 \leq a_1 \leq \ .. \leq a_k$
- $a_{k+1} \geq a_{k+2} \geq ... \geq a_{n-1}$

Bitonic Merge



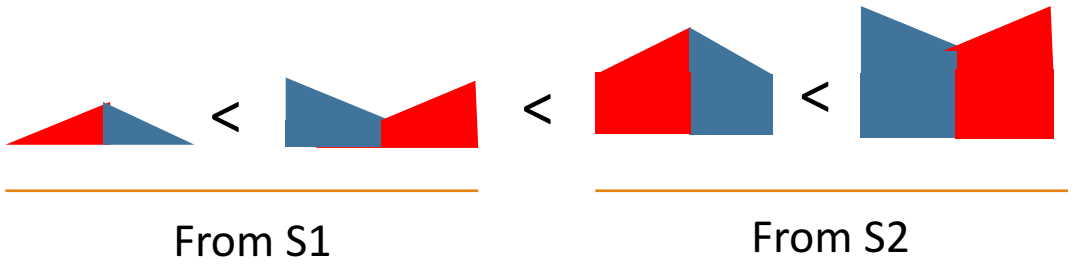
$S_1 = \langle \min(a_0, a_{n/2}), \min(a_1, a_{n/2+1}), \ldots \min(a_{n/2-1}, a_{n-1}) \rangle$

$S_2 = \langle \max(a_0, a_{n/2}), \max(a_1, a_{n/2+1}), \ldots \max(a_{n/2-1}, a_{n-1}) \rangle$

S_1 and S_2 are both bitonic

$S_1 < S_2$: Every element in S_1 is smaller than any element of S_2

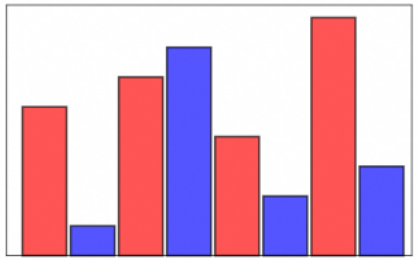
Apply recursively on S_1 and S_2 =>



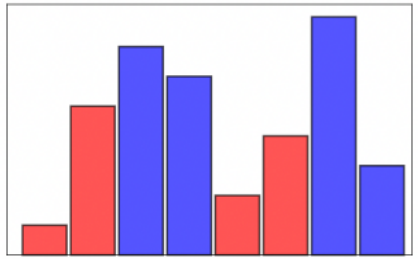
Sort Entire Sequence -> log(n) rounds.

Bitonic Sort

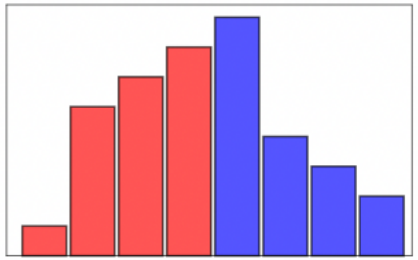
Complexity: $O(n(\log n)^2)$



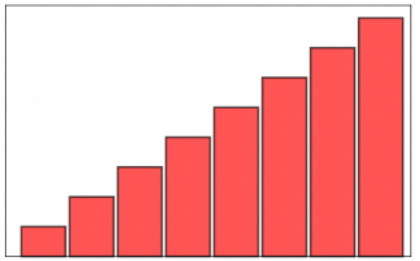
Unsorted Input



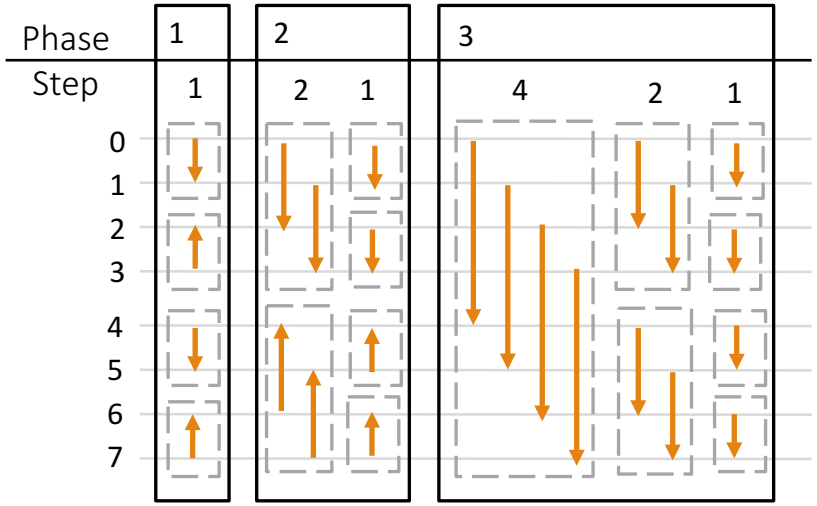
After Phase 1



After Phase 2

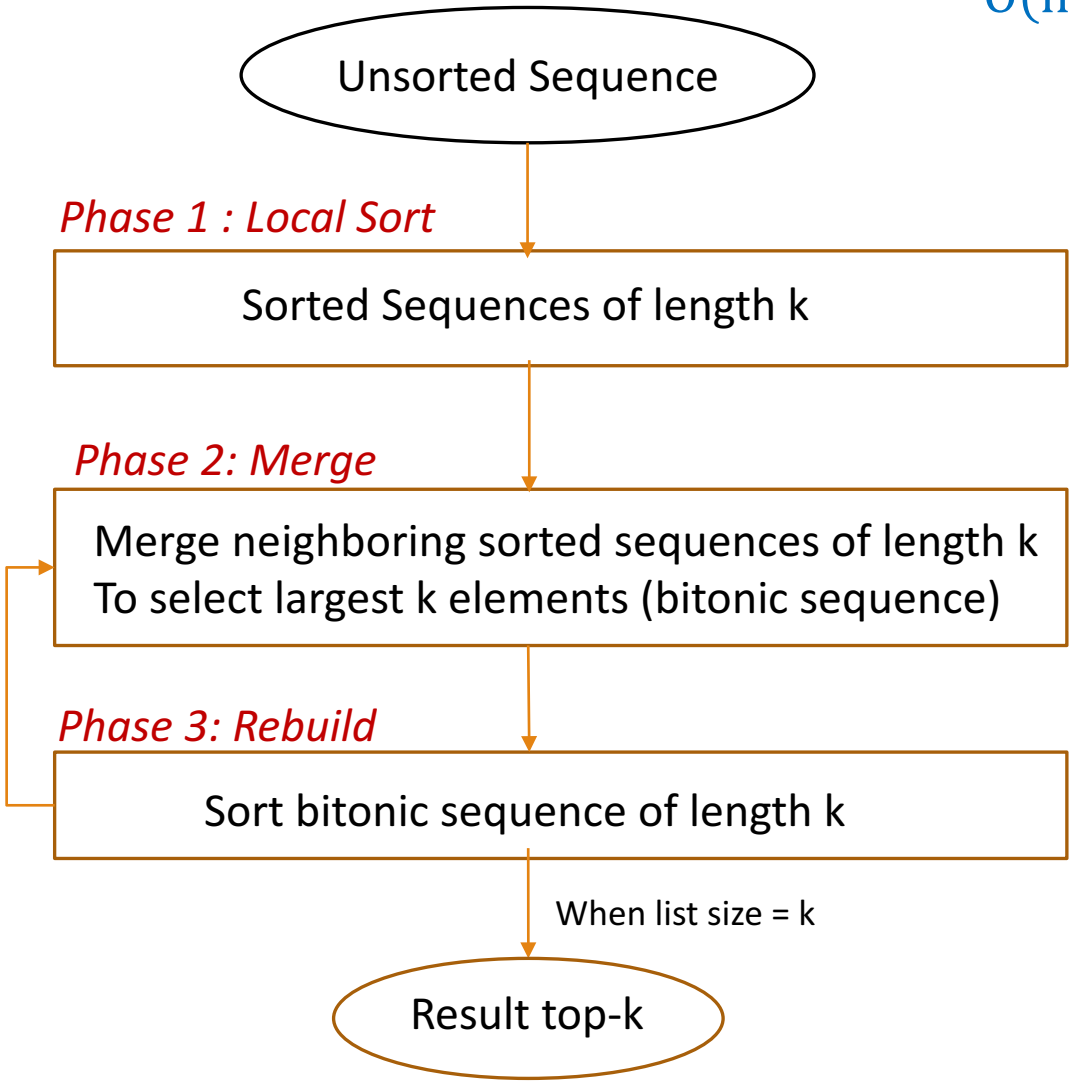


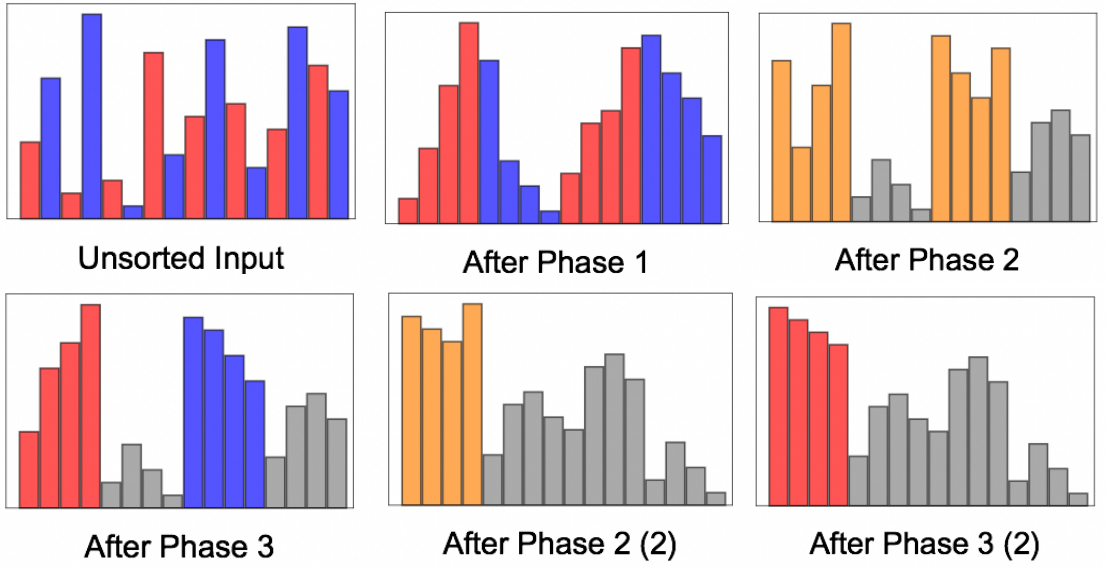
After Phase 3



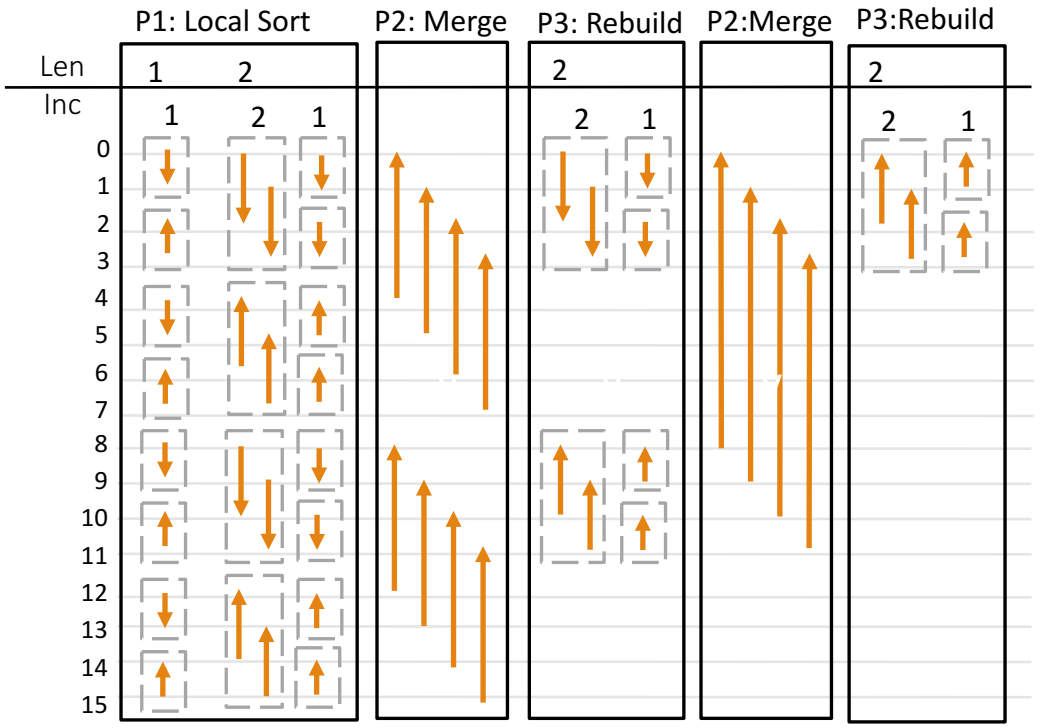
Bitonic Top-K

Complexity: $O(n(\log k)^2)$

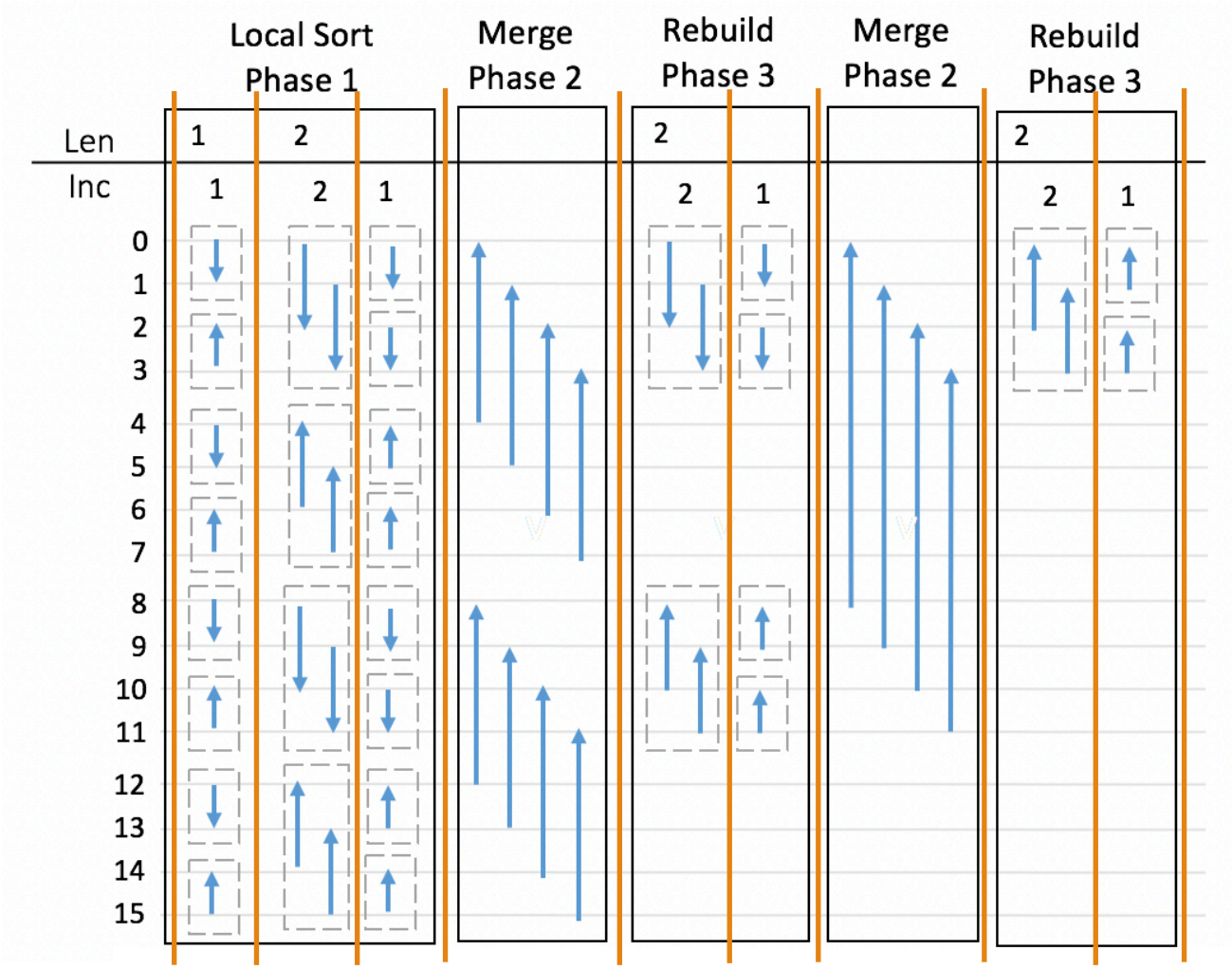




Finding Top-4 in 16 elements

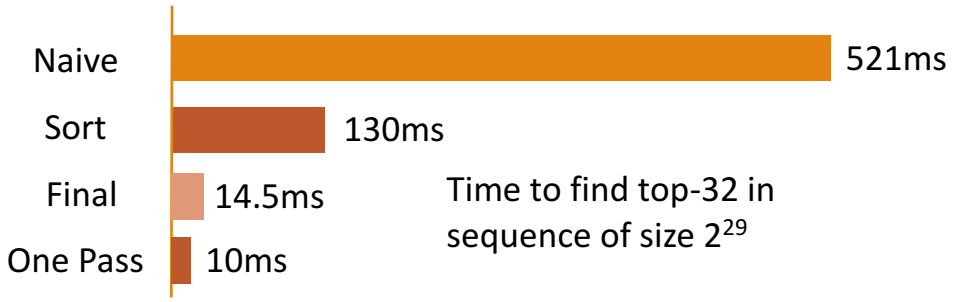


On the GPU



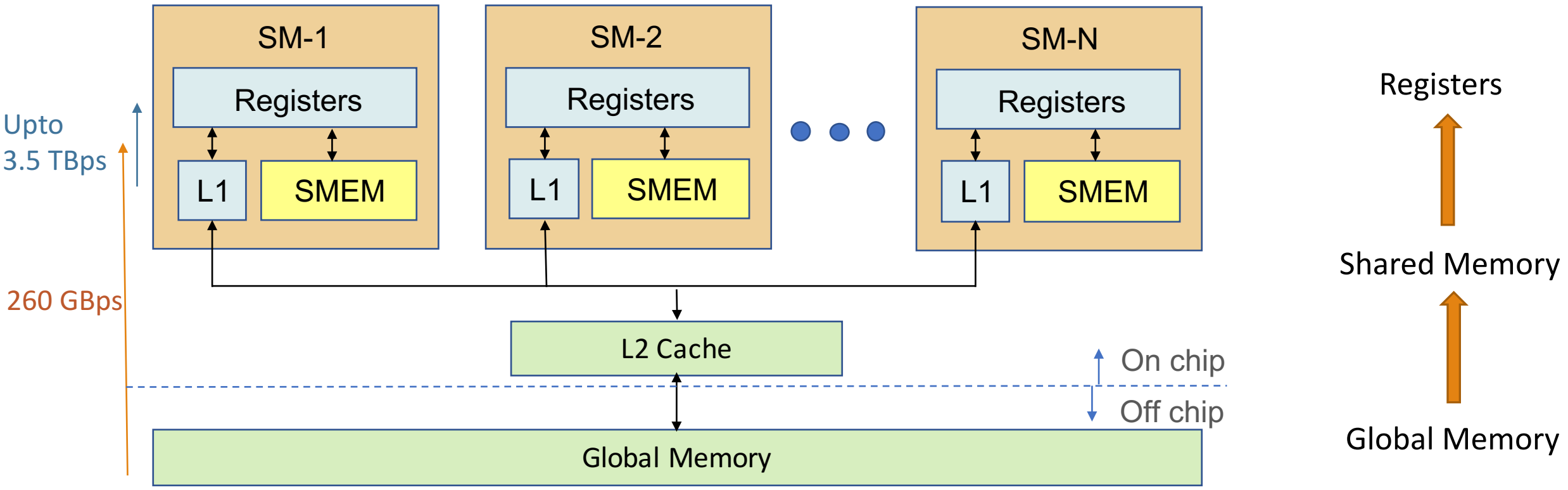
Simplest way to partition into kernels:
Each column has a kernel invocation

Each thread does 1 comparison
n/2 comparisons needed => n/2 threads launched



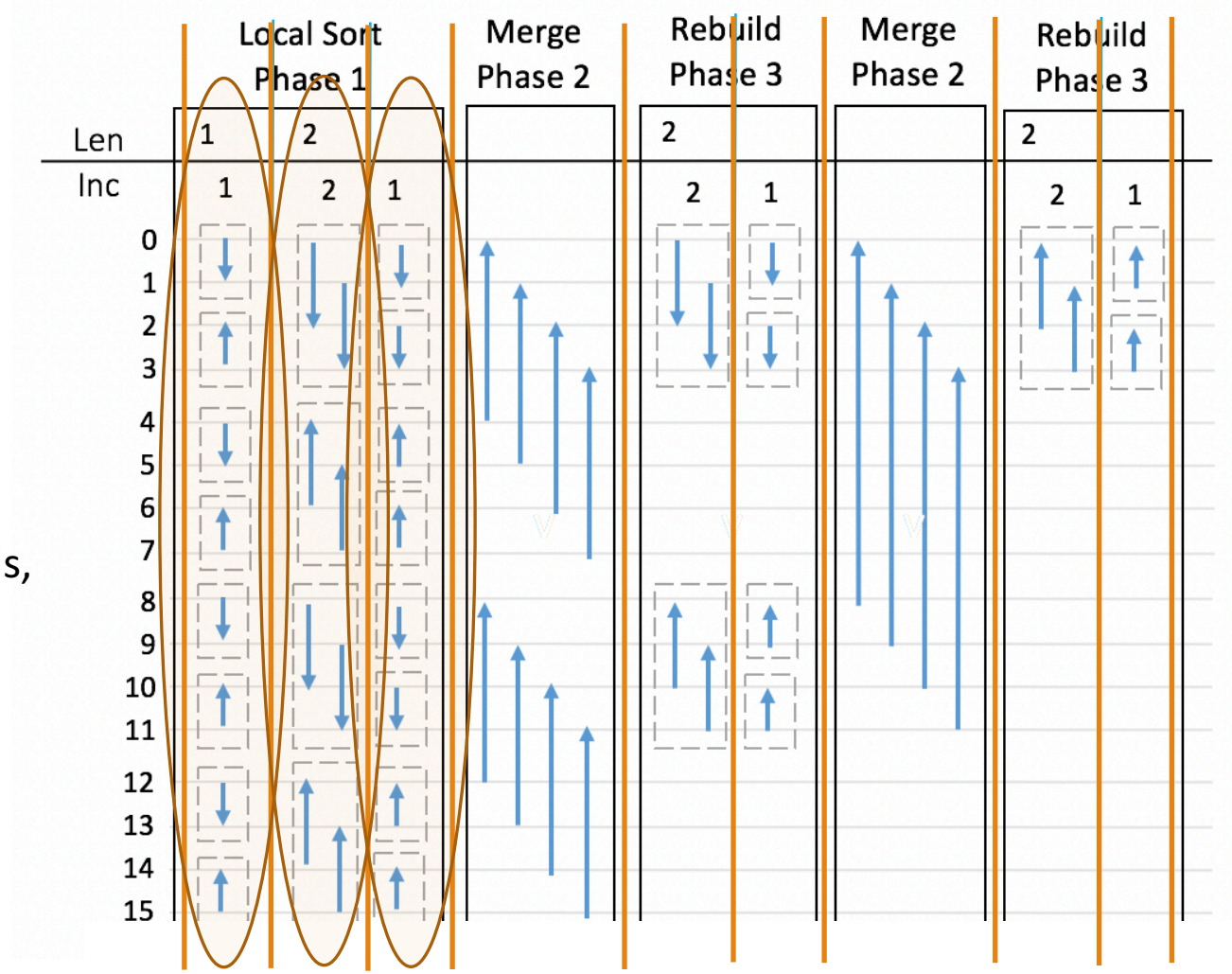
Optimizations

Optimizations



Optimization 1: Using Shared Memory

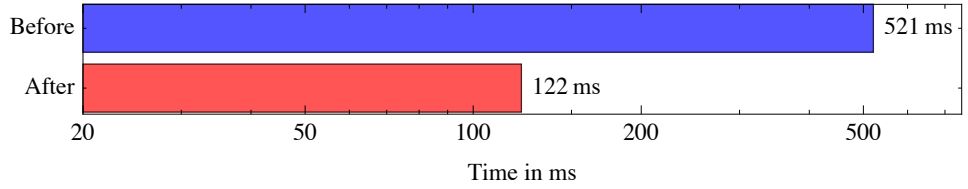
For thread block with T threads, load 2T elements into shared memory



Global memory access

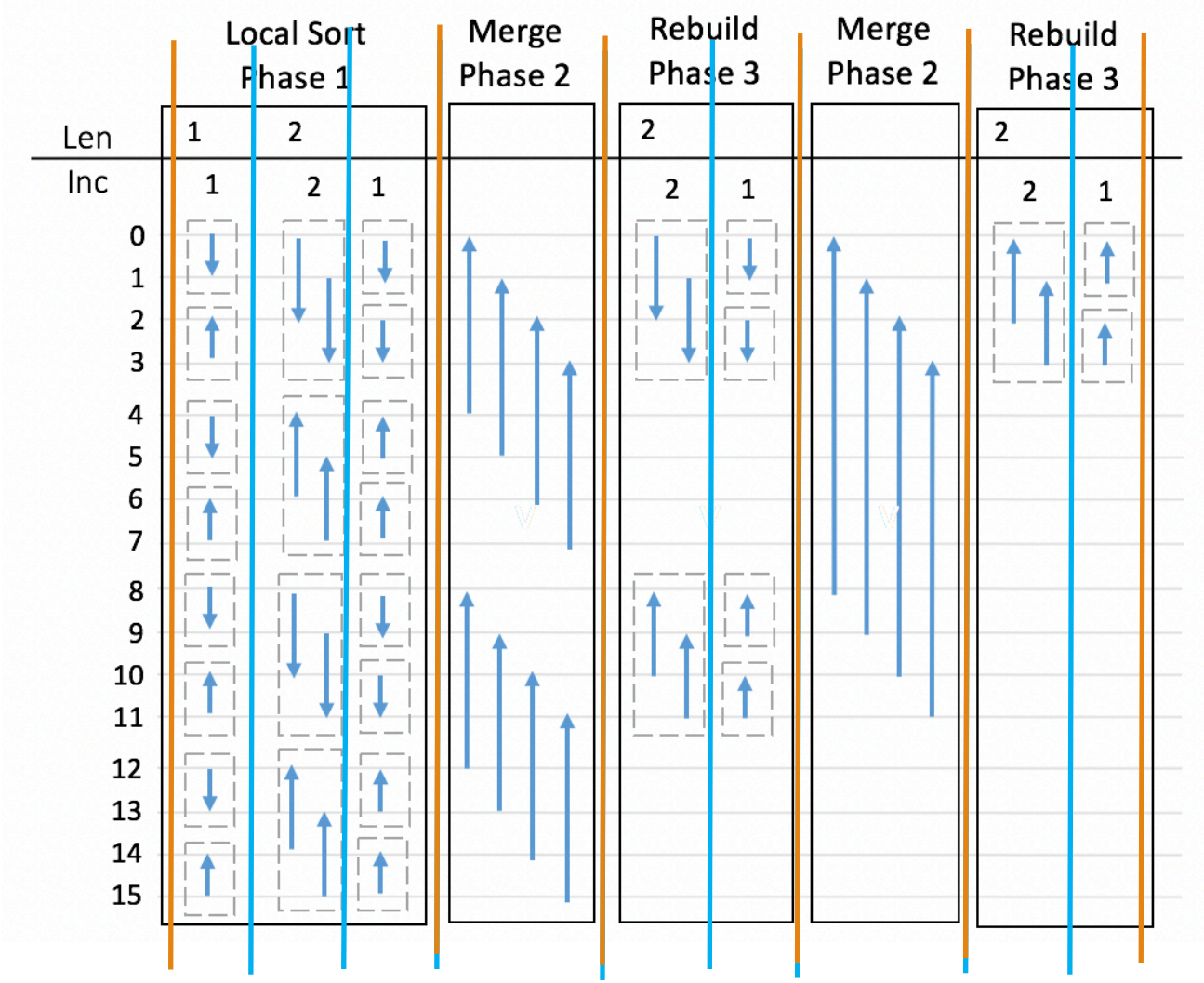
Shared memory access

Time to find top-32 in sequence of size 2^{29}



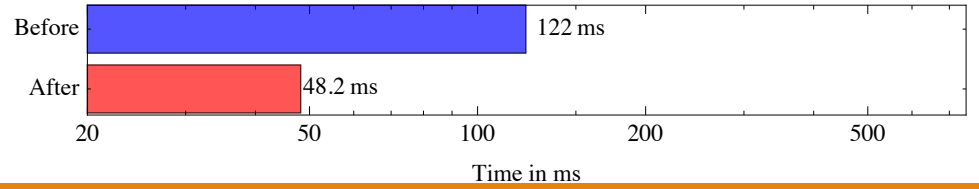
Optimization 2: Combining Phases

Instead of loading 2T, lets load 8T elements and combine the 5 phases



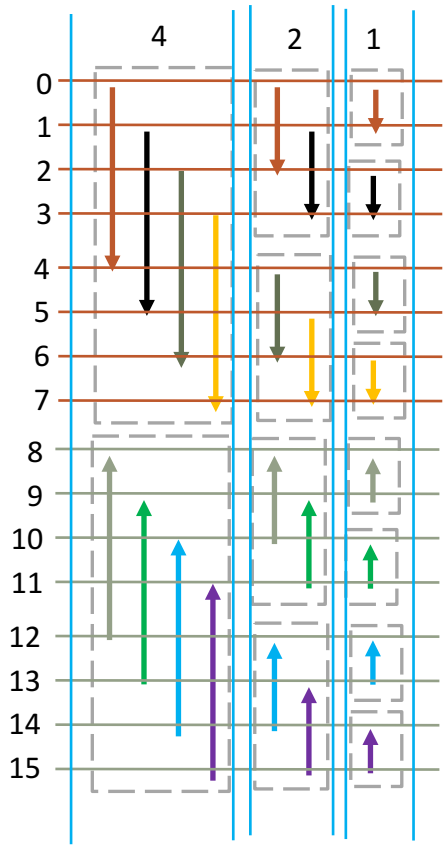
Global memory access

Shared memory access

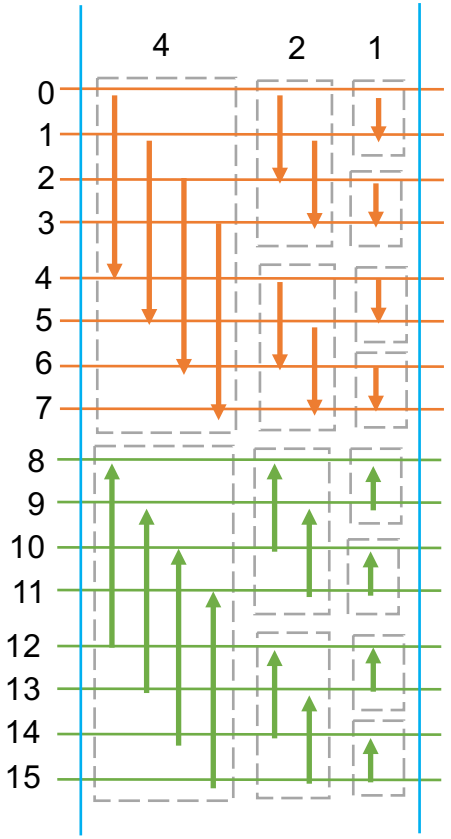


Shared Memory Bandwidth Bound

Optimization 3: Combining Steps

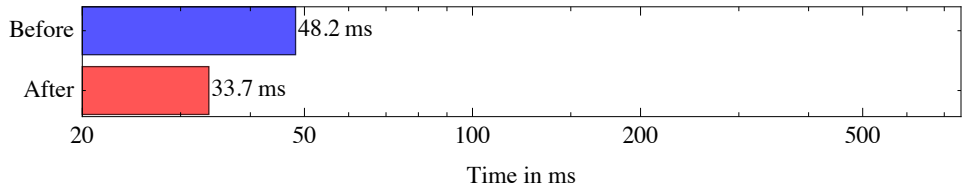


One step at a time

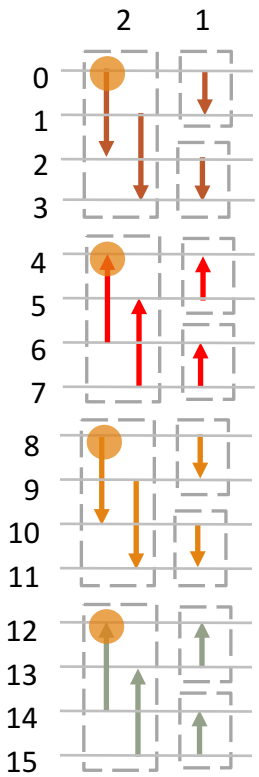


Three steps at a time

Shared memory access



Optimization 4: Padding



Memory Bank

Address	0	1	2	3	4	5	6	7
0					1			
2					3			
4					5			
6					7			

Before Padding

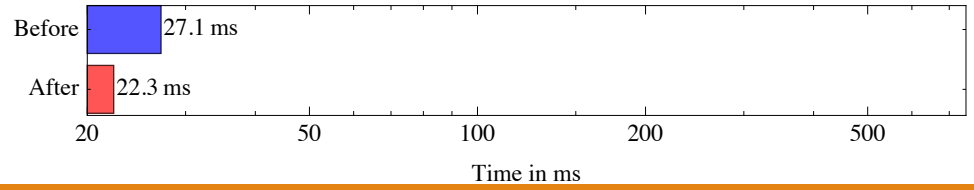
Memory Bank

Address	0	1	2	3	4	5	6	7
0					1			
X	2					3		
	X	4					5	
		X	6					7

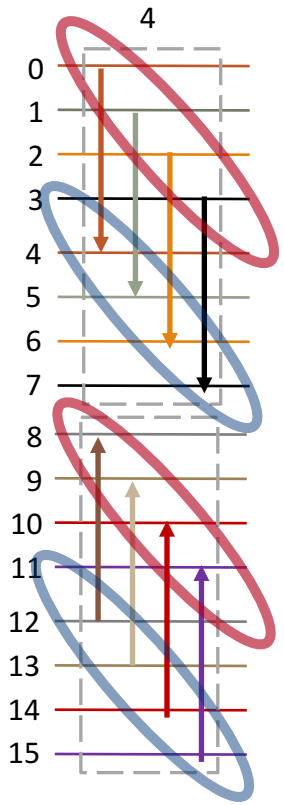
After Padding

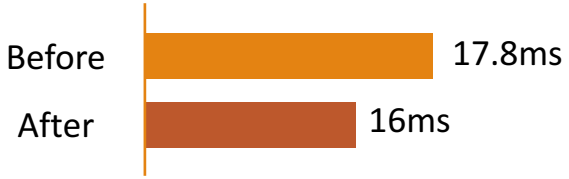
Thread Access

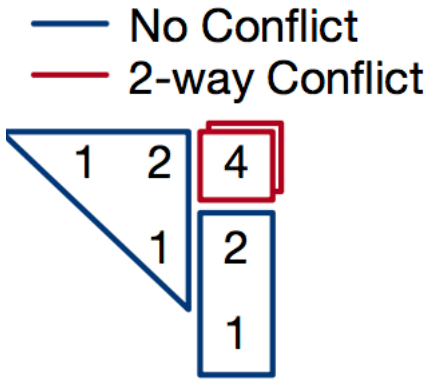
Unused Cell

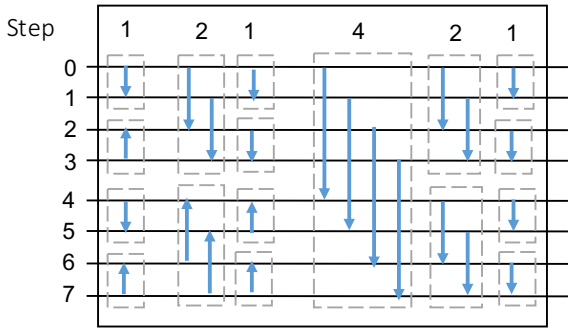


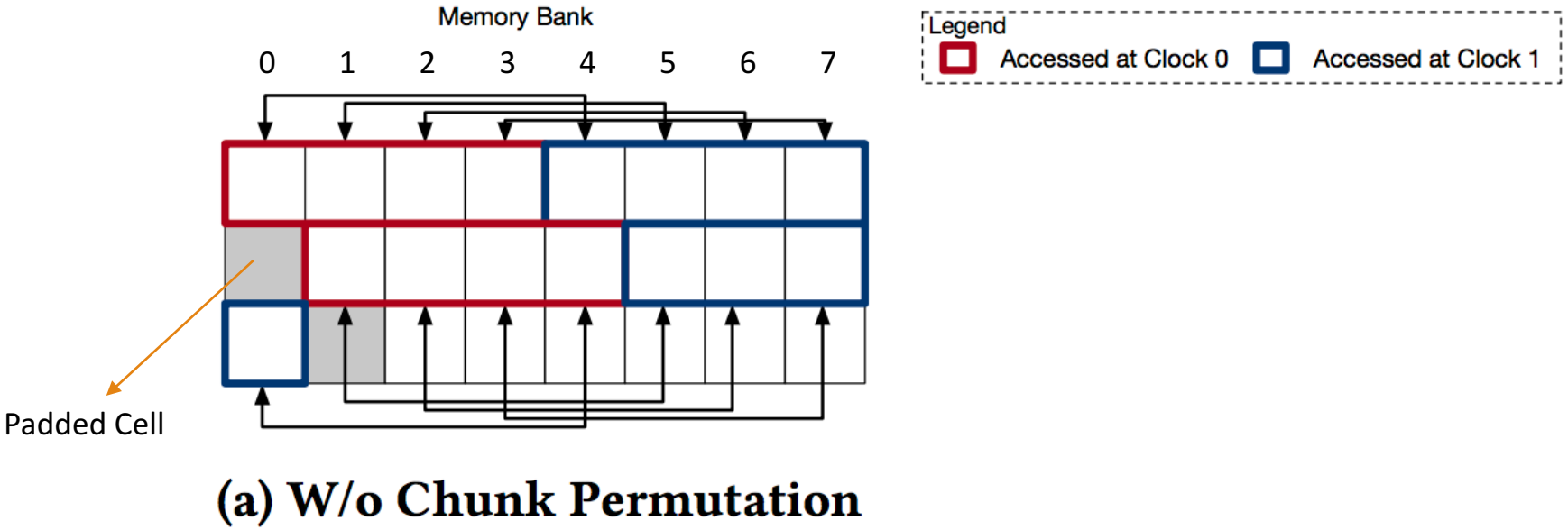
Optimization 5: Chunk Permutation







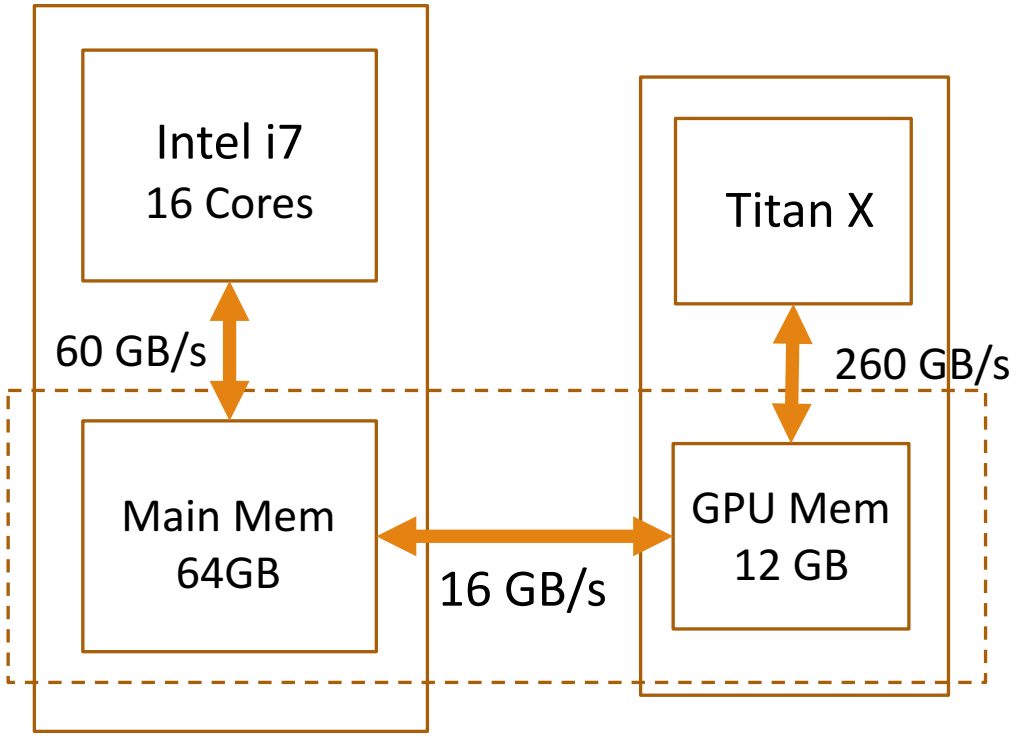




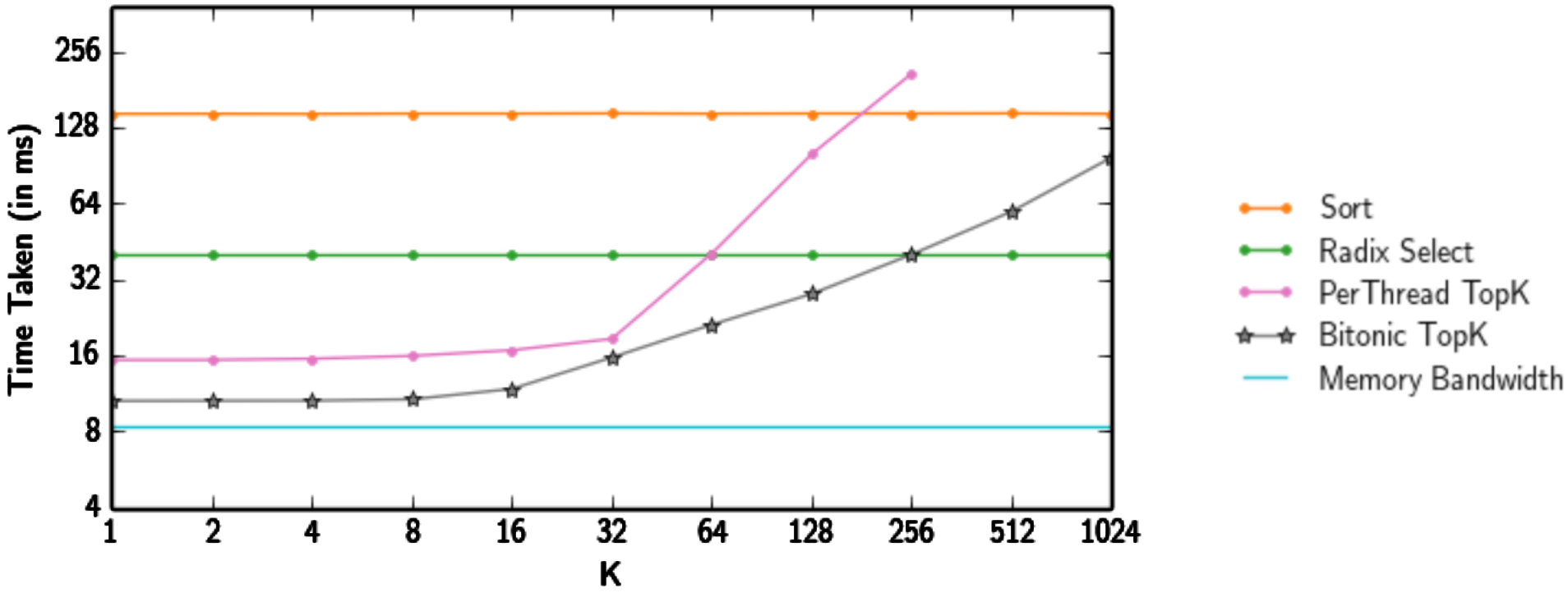
(a) W/o Chunk Permutation

Evaluation

Setup

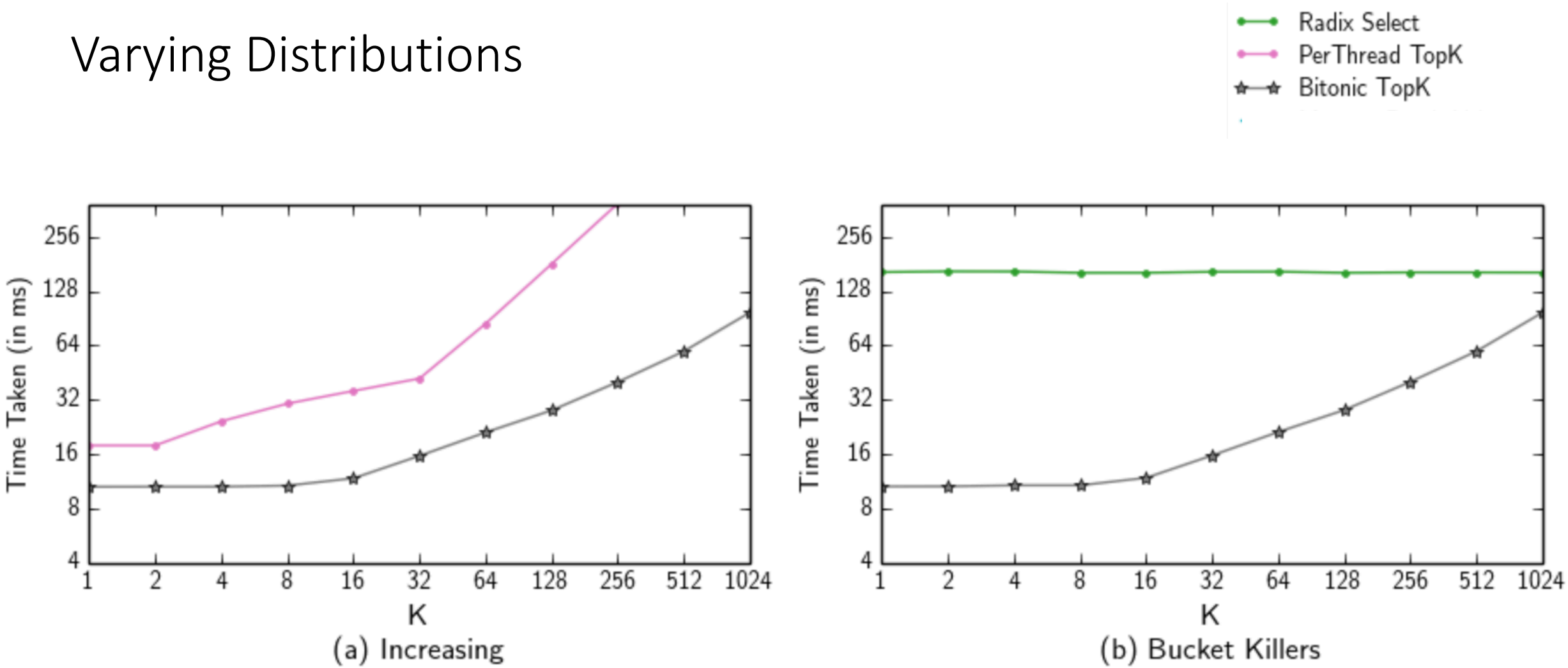

Intel i7
16 Cores
60 GB/s
Main Mem
64GB
16 GB/s
Titan X
260 GB/s
GPU Mem
12 GB

Varying K



For 2^29 (1/2 billion) floats from U(0,1)

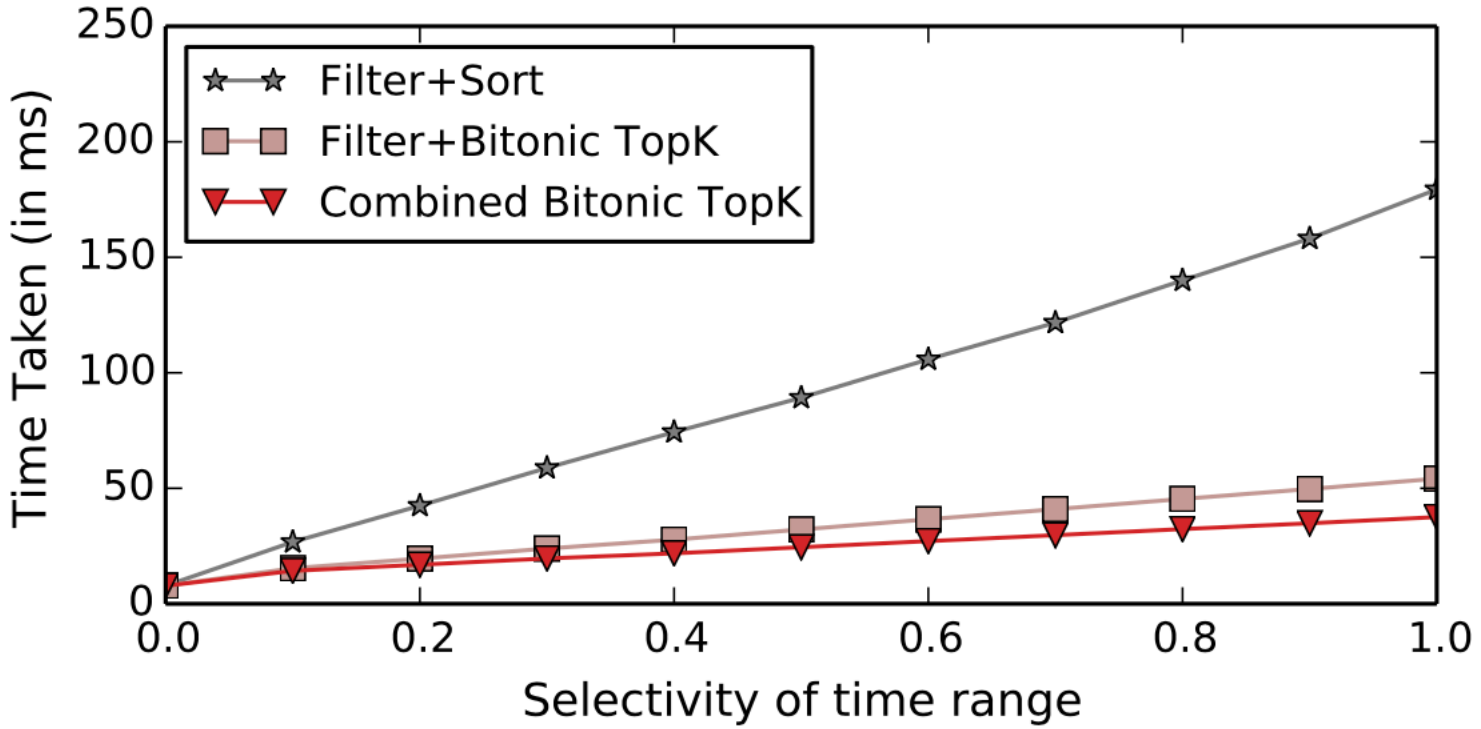
Varying Distributions



Integration

Dataset: 250 million tweets May 2017

```
SELECT id FROM tweets WHERE tweet_time < X
ORDER BY retweet_count DESC LIMIT 50
```



4.5 x Faster

Conclusion

Data analytics on GPUs increasingly common and Top-K on GPU non-trivial

Bitonic Top-k: Novel Top-K algorithm for GPU

- Distribution Independent
- Best performing for K <= 256

Integrated into a real database - >4x performance improvement