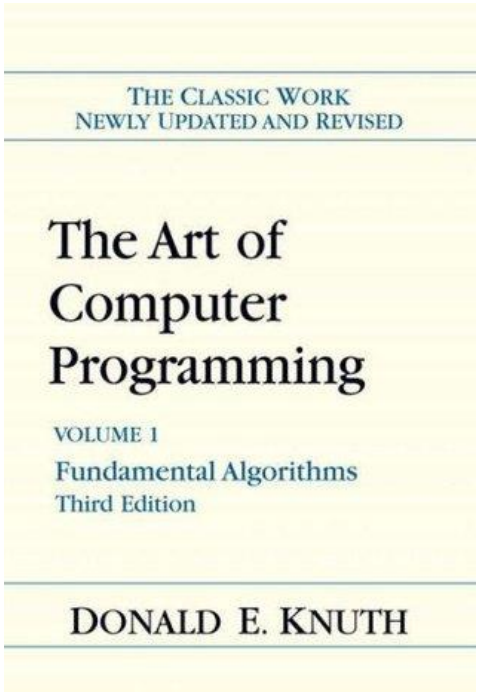
How Emerging Memory Technologies Will Have You Rethinking Algorithm Design

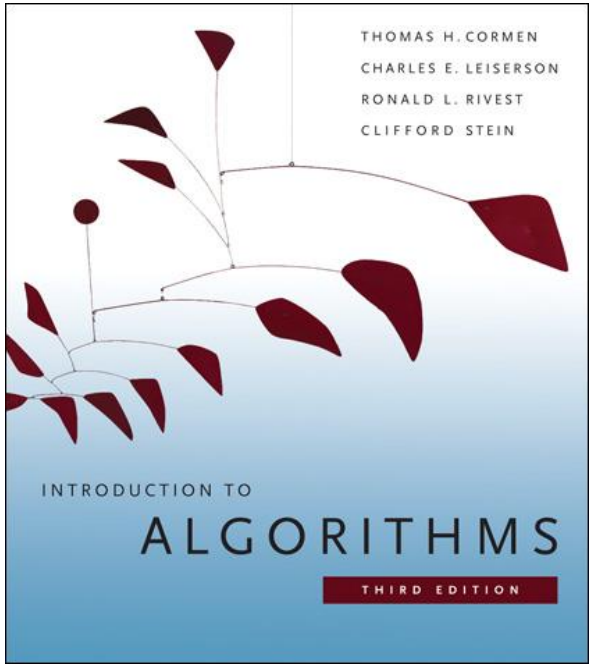
Phillip B. Gibbons

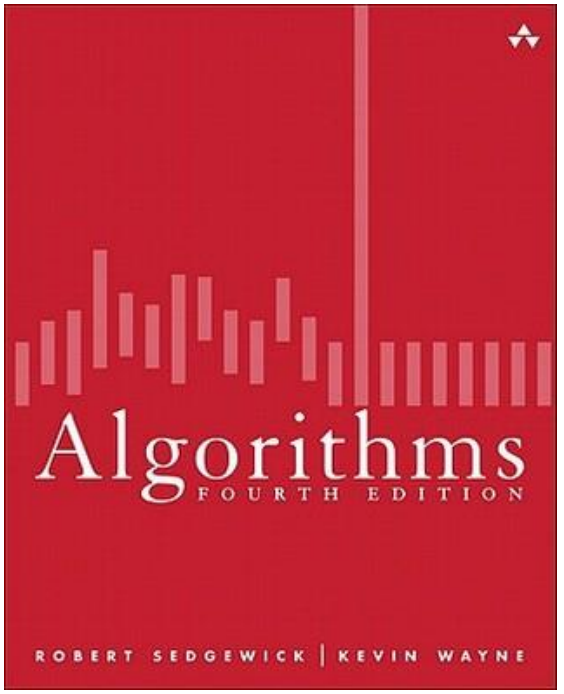
Carnegie Mellon University

PODC'16 Keynote Talk, July 28, 2016

50 Years of Algorithms Research







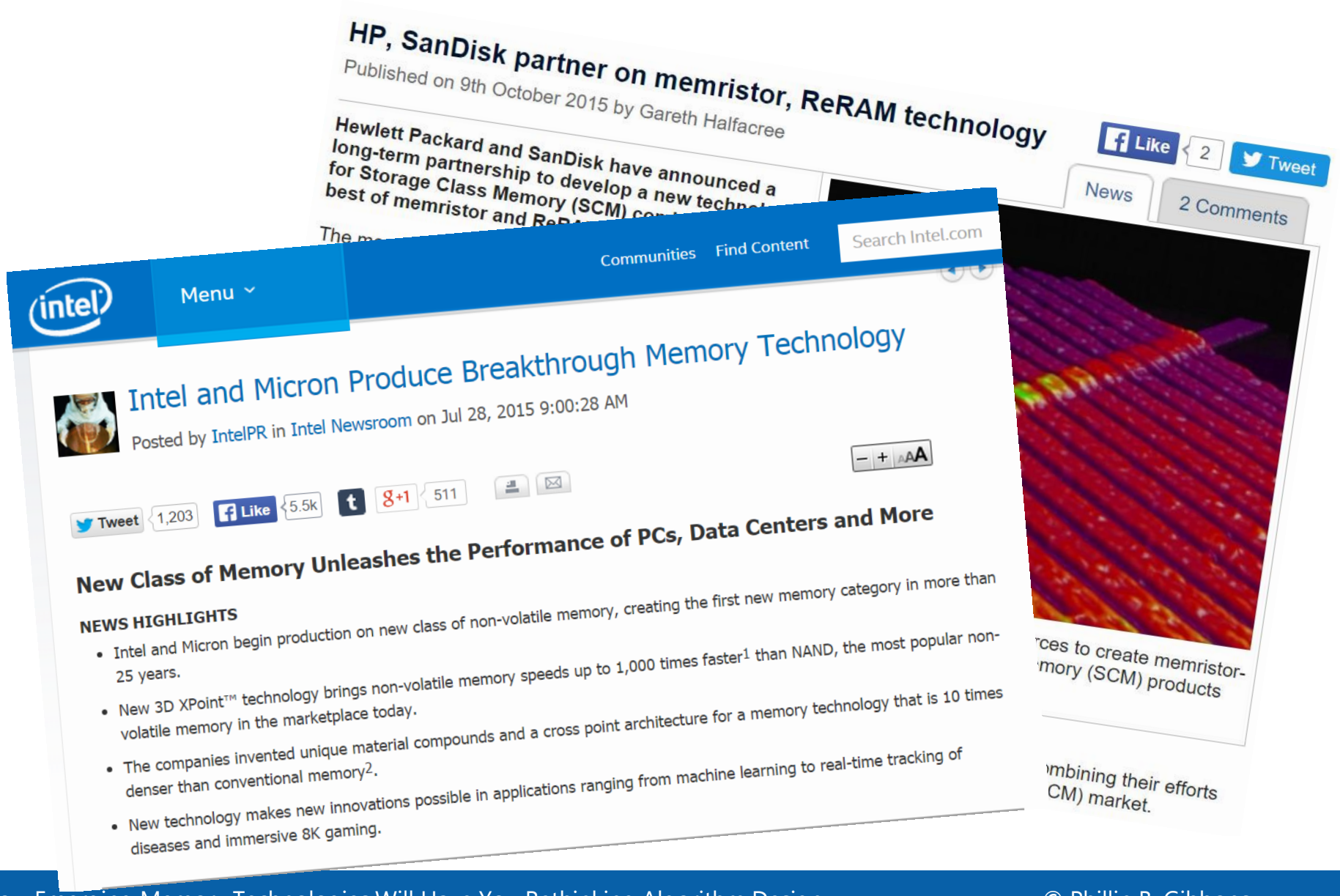
…has focused on settings in which reads & writes to memory have equal cost

But what if they have very DIFFERENT costs? How would that impact Algorithm Design?

Key Take-Aways

- **Main memory will be persistent and asymmetric**
  - Reads much cheaper than Writes
- **Very little work to date on Asymmetric Memory**
  - Not quite: space complexity, CRQW, RMRs, Flash…
- **Highlights of our results to date:**
  - Models: (M,ω)-ARAM; with parallel & block variants
  - Asymmetric memory is not like symmetric memory
  - New techniques for old problems
  - Lower bounds for block variant are depressing

Emerging Memory Technologies



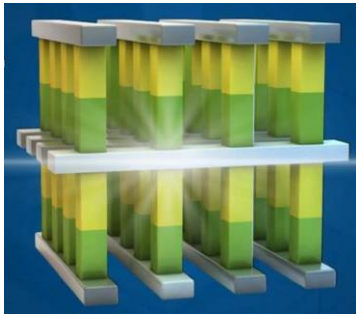
Emerging Memory Technologies

Motivation:

- DRAM (today's main memory) is volatile
- DRAM energy cost is significant (~35% of DC energy)
- DRAM density (bits/area) is limited

Promising candidates:

- **Phase-Change Memory (PCM)**
- **Spin-Torque Transfer Magnetic RAM (STT-RAM)**
- **Memristor-based Resistive RAM (ReRAM)**
- **Conductive-bridging RAM (CBRAM)**



3D XPoint

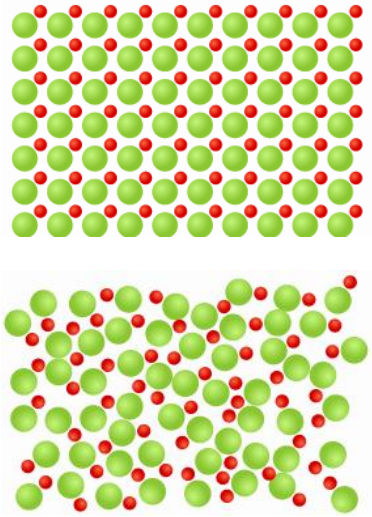
Key properties:

- **Persistent**, significantly lower energy, can be higher density
- Read latencies approaching DRAM, byte-addressable

Another Key Property: Writes More Costly than Reads

In these emerging memory technologies, bits are stored as "states" of the given material

- **No energy to retain state**
- **Small energy to read state**
  - Low current for short duration
- **Large energy to change state**
  - High current for long duration



PCM

Writes incur higher energy costs, higher latency, lower per-DIMM bandwidth (power envelope constraints), endurance problems

Cost Examples from Literature (Speculative)

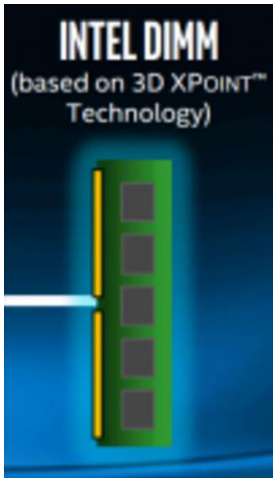
- PCM: writes 15x slower, 15x less BW, 10x more energy
- PCM L3 Cache: writes up to 40x slower,17x more energy
- STT-RAM cell: writes 71x slower, 1000x more energy @ material level
- ReRAM DIMM: writes 117x slower, 125x more energy
- CBRAM: writes 50x more energy

Sources: [KPMWEVNBPA14] [DJX09] [XDJX11] [GZDCH13]

Costs are a well-kept secret by Vendors

Are We There Yet?

- **3D XPoint will first come out in SSD form factor**
  - No date announced: expectation is 2017



- **Later will come out in DIMM form factor**
  - Main memory: Loads/Stores on memory bus
  - No date announced: perhaps 2018

- **Energy/density/persistence advantages ⇨ Projected to become dominant main memory**

In near future:
Main memory will be persistent & asymmetric

Write-Efficient Algorithm Design

Goal: Design write-efficient algorithms (write-limited, write-avoiding)

- **Fewer writes ⇨ Lower energy, Faster**

How we model the asymmetry: In asymmetric memory, writes are ω times more costly than reads

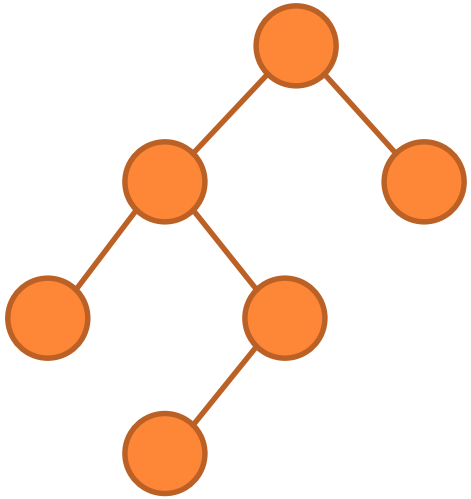
Warm up: Write-efficient Sorting

How does one sort n elements using $O(n \log n)$ instructions (reads) but only $O(n)$ writes?

- **Swap-based sorting (i.e. quicksort, heap sort) does $O(n \log n)$ writes**
- **Mergesort requires n writes for $\log n$ rounds**

Solution:

- **Insert each key in random order into a binary search tree**
- **An in-order tree traversal yields the sorted array**



Asymmetric Read-Write Costs: Prior Work (1)

- **Space complexity classes such as L**
  - Can repeatedly read input
  - Only limited amount of working space

  What's missing: Doesn't charge for number of writes
  - OK to write every step

- **Similarly, streaming algorithms have limited space**
  - But OK to write every step

Asymmetric Read-Write Costs: Prior Work (2)

- **Reducing writes to contended shared memory vars**
  - Multiprocessor cache coherence serializes writes, but reads can occur in parallel
  - Concurrent-read-queue-write (CRQW) model [GMR98]
  - Contention in asynchronous shared memory algs [DHW97]
  - Etc, etc

  What's missing: Cost of writes to even un-contended vars
  - OK to write every step to disjoint vars (disjoint cache lines)

- **Similarly, reducing writes to minimize locking/synch**
  - But OK for sequential code to write like a maniac!

Asymmetric Read-Write Costs: Prior Work (3)

- **Remote Memory References (RMR)** [YA95]
  - Only charge for remote memory references, i.e., references that require an interconnect traversal
  - In cache-coherent multiprocessors, only charge for:
    - A read(x) that gets its value from a write(x) by another process
    - A write(x) that invalidates a copy of x at another process
  - Thus, writes make it costly

What's missing: Doesn't charge for number of writes

Asymmetric Read-Write Costs: Prior Work (4)

- **NAND Flash. This work focused on:**
  - Asymmetric **granularity** of writes (must erase large blocks)
  - Asymmetric **endurance** of writes [GT05, EGMP14]

No granularity issue for emerging NVM

- Byte-addressable for both reads and writes

Individual cell endurance not big issue for emerging NVM

- Can be handled by system software

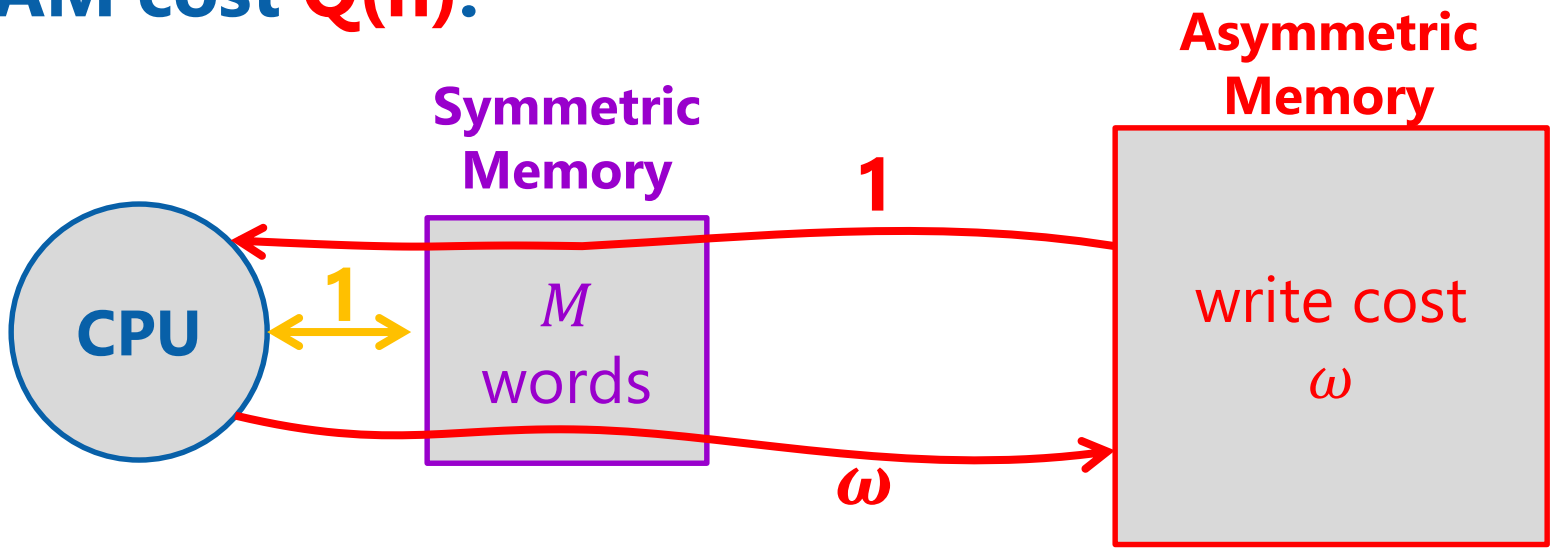
Key Take-Aways

- **Main memory will be persistent and asymmetric**
  - Reads much cheaper than Writes
- **Very little work to date on Asymmetric Memory**
  - Not quite: space complexity, CRQW, RMRs, Flash,…
- **Highlights of our results to date:** ⬅ **You Are Here**
  - Models: (M,ω)-ARAM; with parallel & block variants
  - Asymmetric memory is not like symmetric memory
  - New techniques for old problems
  - Lower bounds for block variant are depressing

(M, ω)-Asymmetric RAM (ARAM)

[BFGGS16]

- **Comprised of:**
  - processor executing RAM instructions on $\Theta(\log n)$-bit words
  - a symmetric memory of M words
  - an asymmetric memory of unbounded size, with write cost ω
- **ARAM cost Q(n):**



- **Time T(n) = Q(n) + # of instructions**

Write-efficient Algorithms

Problem	Read (unchanged)	Previous write	Current write	Reduction ratio
Comparison sort	$\Theta(n \log n)$	$O(n \log n)$	$\Theta(n)$	$O(\log n)$
Search tree, priority queue	$\Theta(\log n)$	$O(\log n)$	$\Theta(1)$	$O(\log n)$
2D convex hull, triangulation	$\Theta(n \log n)$	$O(n \log n)$	$\Theta(n)$	$O(\log n)$
BFS, DFS, topological sort, bi-CC, SCC	$\Theta(n + m)$	$O(n + m)$	$\Theta(n)$	$O(m/n)$

- **Trivial**
- **Significant reduction. M can be O(1)**

Reduction in Writes depends on *M*, ω, input size

Problem	ARAM cost $Q(n,m)$	
	Classic algorithms	New algorithms
Single-source shortest-path	$O\big(\omega(m + n\log n)\big)$	$O(\min(n(\omega + m/M),$ $\omega(m + n\log n),$ $m(\omega + \log n)))$
Minimum spanning tree	$O(m\omega)$	$O(\min(m\omega,$ $m\min(\log n, n/M) + \omega n))$

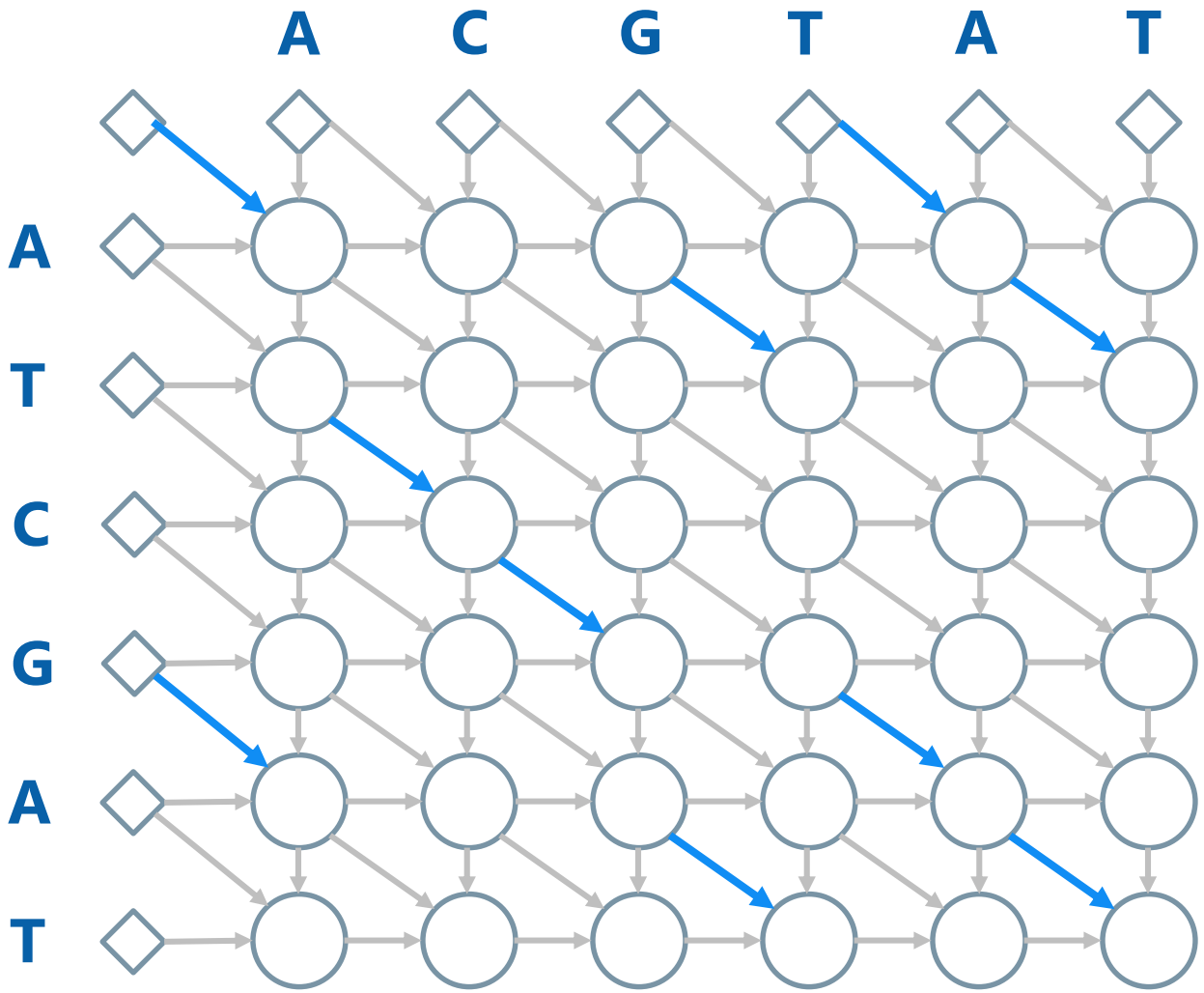
- **SSSP: Phased Dijkstra that uses phases & keeps a truncated priority queue in symmetric memory**
- **Write-efficient bookkeeping is often challenging**

No (significant) improvement with cheaper reads

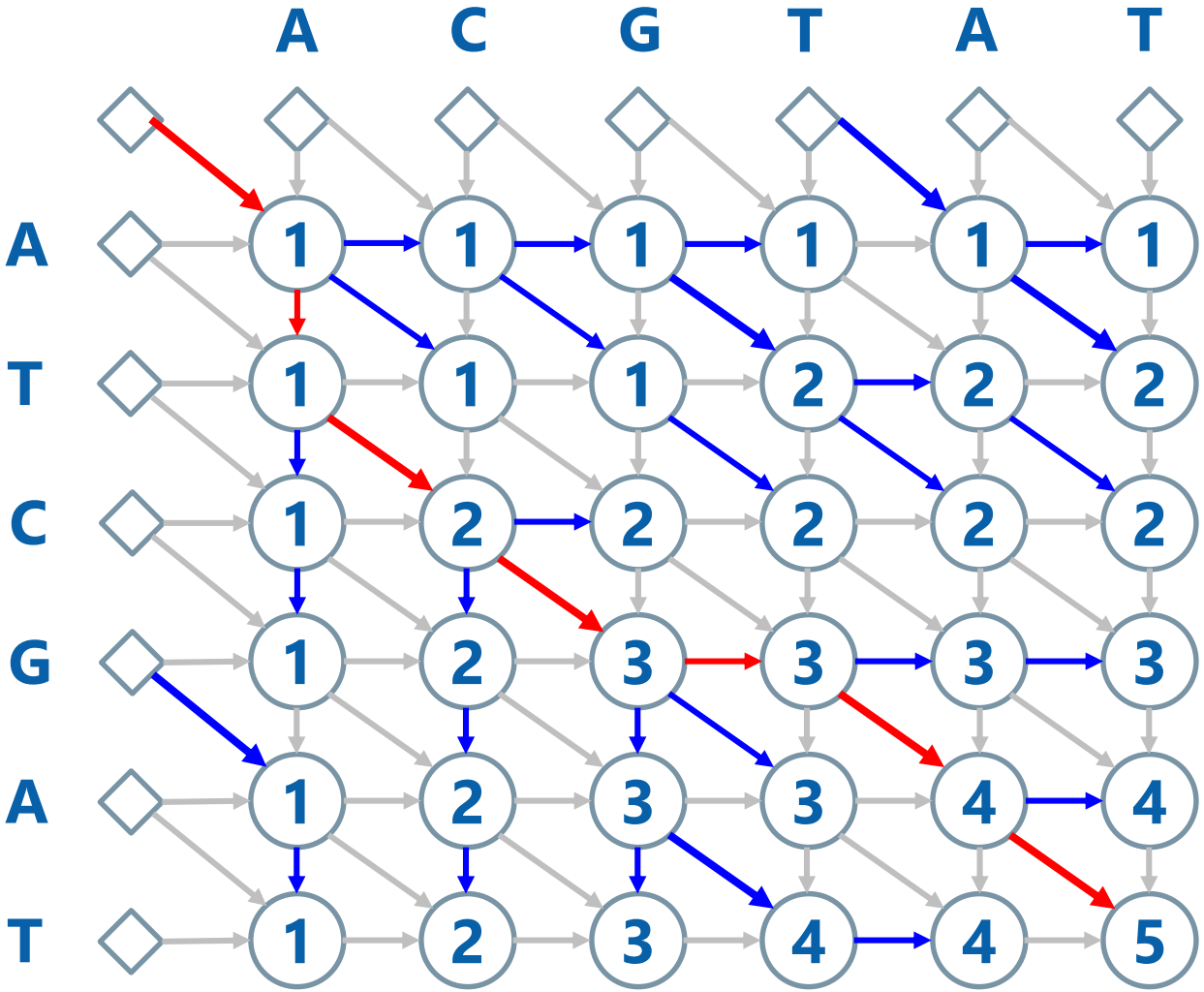
Problem	ARAM cost *Q(n)*	
	Classic Algorithm	New Lower Bound
Sorting networks and Fast Fourier Transform	$\Theta\left(\omega n \frac{\log n}{\log M}\right)$	$\Theta\left(\omega n \frac{\log n}{\log \omega M}\right)$
Diamond DAG (ala LCS, edit distance)	$\Theta\left(\frac{n^2\omega}{M}\right)$	$\Theta\left(\frac{n^2\omega}{M}\right)$

- **New FFT lower bound technique (generalizes** [HK81]**)**
- **Gap between comparison sorting & sorting networks**
  - No gap for classic RAM setting, PRAM, etc

An Example of a Diamond DAG: Longest Common Subsequence (LCS)

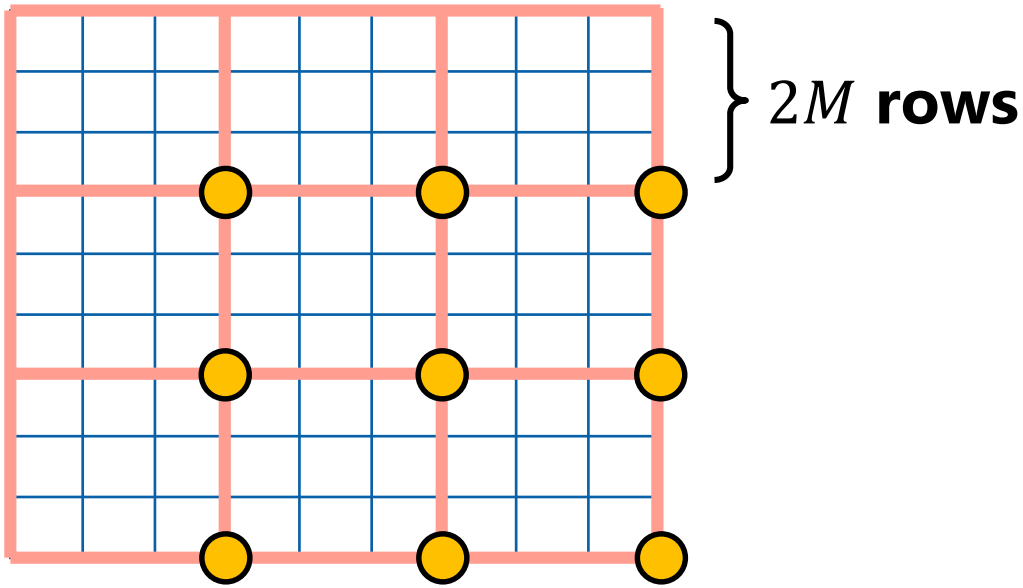


An Example of a Diamond DAG: Longest Common Subsequence (LCS)



Proof sketch of $\Theta\left(\frac{\omega n^2}{M}\right)$ diamond DAG lower bound

- $k \times k$ **diamond requires** k **storage to compute** [CS76]
- **Computing any** $2M \times 2M$ **diamond requires** M **writes to the asymmetric memory**
  - $2M$ storage space, M from symmetric memory
- **Tiling with** $2M \times 2M$ **sub-DAGs yields** $n^2/(2M)^2$ **tiles**



Asymmetric Memory is not like Symmetric Memory

Problem	ARAM cost *Q(n)*	
	Classic Algorithm	New Lower Bound
Sorting networks and Fast Fourier Transform	$\Theta\left(\omega n \frac{\log n}{\log M}\right)$	$\Theta\left(\omega n \frac{\log n}{\log \omega M}\right)$
Diamond DAG (ala LCS, edit distance)	$\Theta\left(\frac{n^2\omega}{M}\right)$	$\Theta\left(\frac{n^2\omega}{M}\right)$

- **DAG rule: Compute a node after all its inputs ready**
- **By breaking this rule: LCS cost reduced by $O(\omega^{1/3})$**
  - New "path sketch" technique

- **Classic RAM: No gap between Diamond DAG & LCS/Edit Distance**
- **Classic RAM: No gap between Sorting Networks & Comparison Sort**

Asymmetric Shared Memory

- **(M, ω)-Asymmetric PRAM** (machine model) [BFGGS15]
  - P processors, each with local memory of size M
  - Unbounded asymmetric shared memory, write cost ω

- **Asymmetric Nested-Parallel Model** [BBFGGMS16]
  - Processor oblivious
  - Provably good with work-stealing schedulers

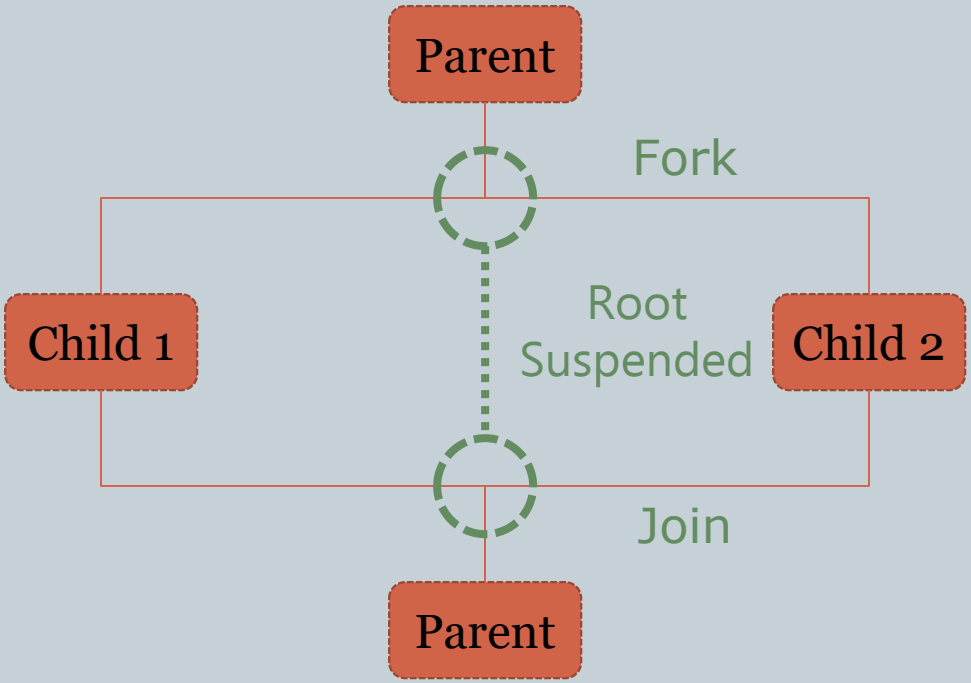
Reduce on Asymmetric PRAM Model

```
Reduce(list L, function F, identity I){
    if(L.length == 0){
        return I;
    }
    if(L.length == 1){
        return L[0];
    }
    L1, L2 = split(L);
    R1 = Reduce(L1, F, I);
    R2 = Reduce(L2, F, I);
    return F(R1, R2);
}
```

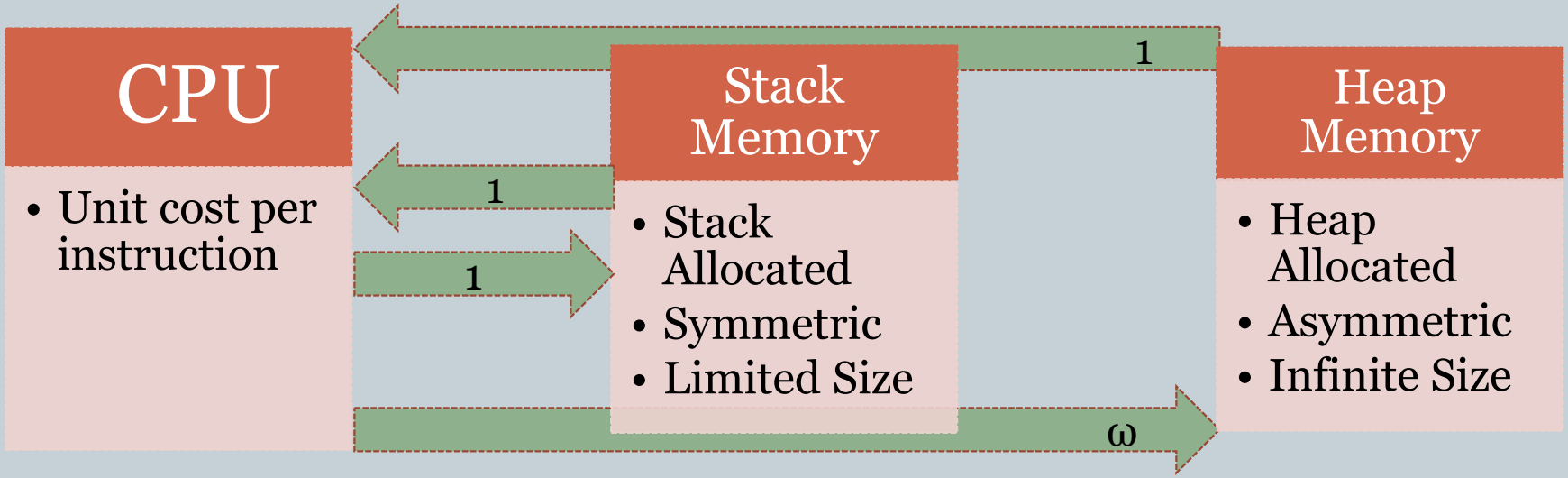
- Assume $\theta(1)$ work for F
- Each write costs ω
  - Split takes $\theta(\omega)$ work
- Work
  - $W(n) = 2W\left(\frac{n}{2}\right) + \theta(\omega)$
  - $W(n) = \theta(\omega n)$
- Span
  - $D(n) = D\left(\frac{n}{2}\right) + \theta(\omega)$
  - $D(n) = \theta(\omega \log(n))$

Too conservative: All intermediate results written to shared memory
Must explicitly schedule computation on processors

Asymmetric Nested-Parallel (NP) Model: Fork-Join



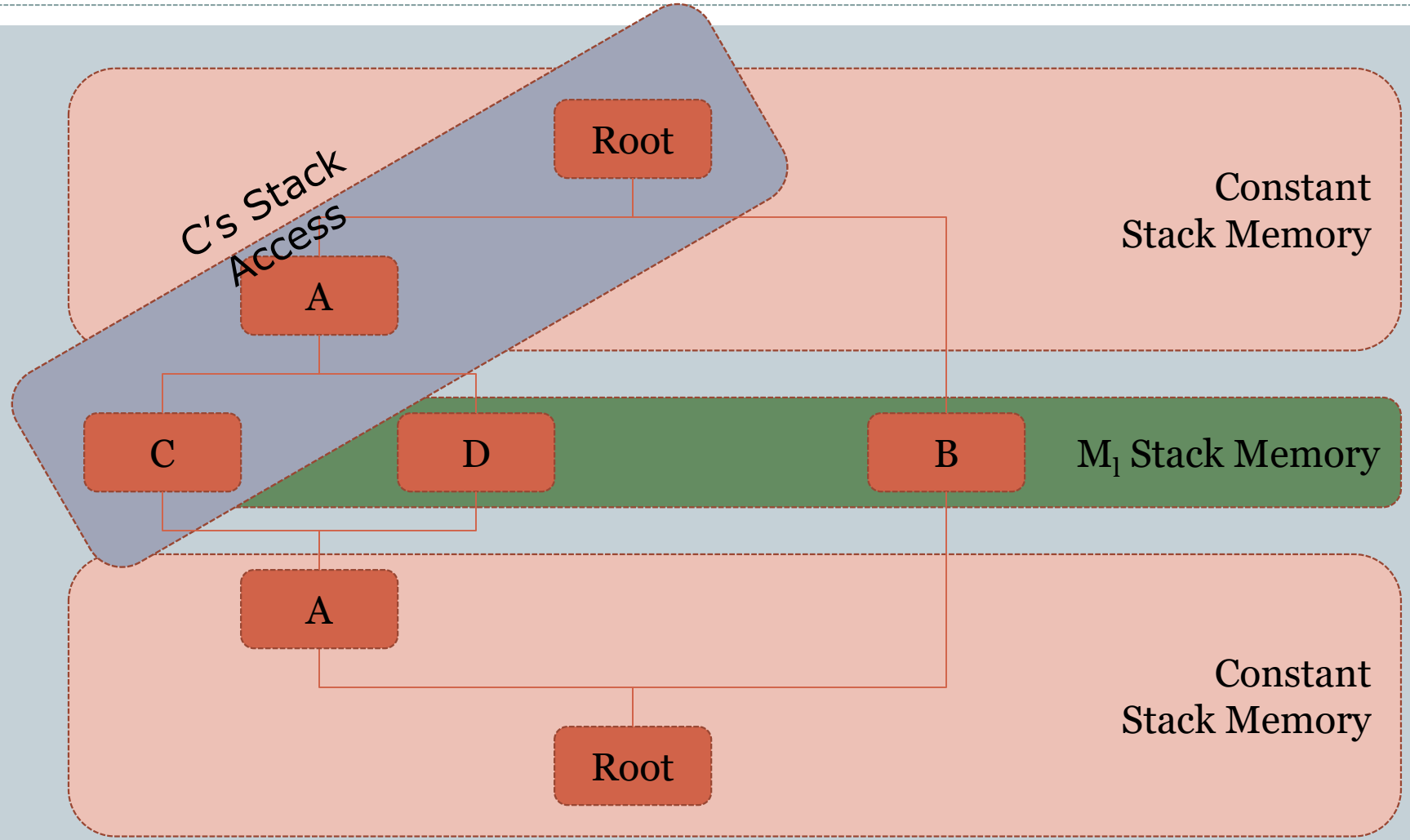
Asymmetric NP Model: Memory Model



Key feature: Algorithmic cost model

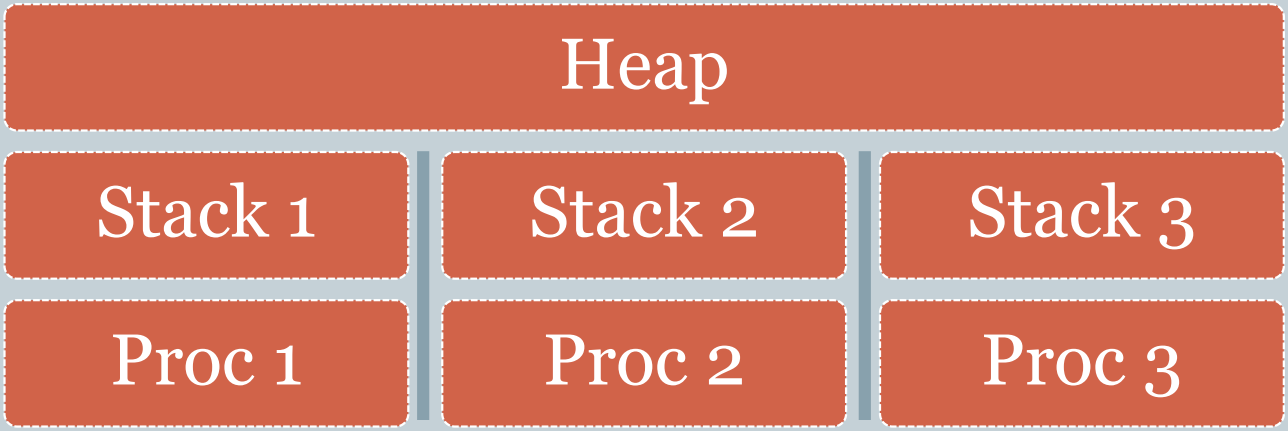
➡ **Processor-oblivious**

Asymmetric NP Model: Stack Memory

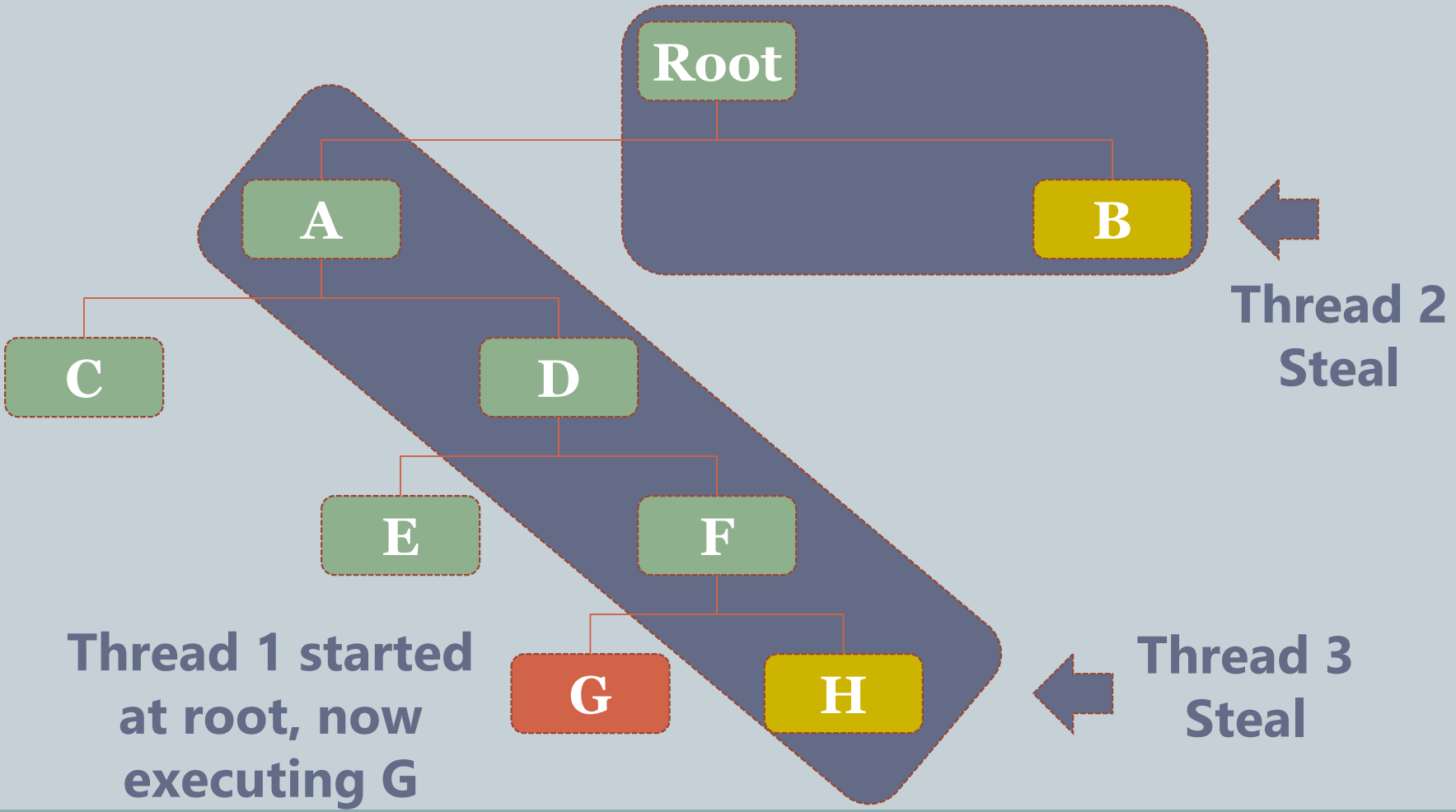


Asymmetric NP Model: Work Stealing Issues

- Thieves need access to stack memory of stolen task
  - Good news: Non-leaf stacks are $O(1)$ size
- Approach 1: Write out all stack memory every fork
  - Have to pay $\theta(\omega)$for each fork!
- Approach 2: Write stack memory only on steal
  - Challenge: Need to limit number of stacks written per steal



Write Stack Memory Only on Steal: Problem Scenario



Asymmetric NP Model: Work Stealing Theorem

A computation with binary branching factor on the **Asymmetric NP model** can be simulated on the **(M, ω)-Asymmetric PRAM machine model** in:

$$O\left(\frac{W}{P} + \omega D\right) \text{ Expected Time}$$

where:

P = processors

D = span

W = work

δ = nesting depth

M_l = leaf stack memory

$M = \theta(\delta + M_l)$

Reduce: Asymmetric NP Model

```
Reduce(list L, function F, identity I){
    if(L.length == 0){
        return I;
    }
    if(L.length == 1){
        return L[0];
    }
    L1, L2 = split(L);
    R1 = Reduce(L1, F, I);
    R2 = Reduce(L2, F, I);
    return F(R1, R2);
}
```

- Assume $\theta(1)$ work for F
- Minimize writes to large memory
  - Children are forked tasks
  - Tasks store list start & end
  - Only write final answer
- Work
  - $W(n) = \theta(n + \omega)$
- Span
  - $D(n) = \theta(\log(n) + \omega)$

Intermediate results NOT written to memory
Scheduler handles inter-processor communication & its costs

Write-Efficient Shared Memory Algorithms

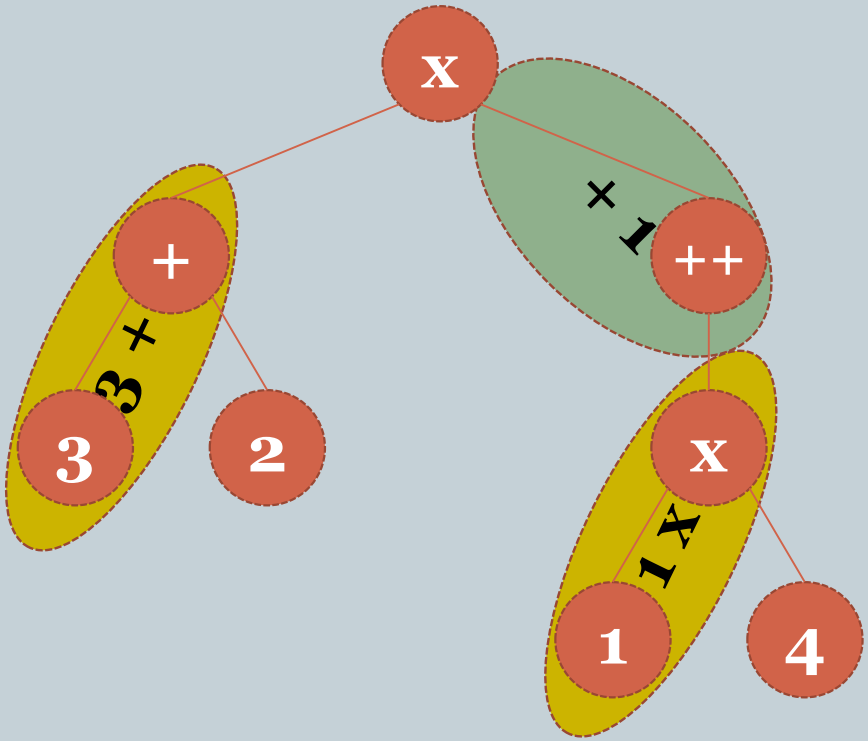
Problem	Work (W)	Span (D)	Reduction of Writes
Reduce	$\theta(n+\omega)$	$\theta(\log n+\omega)$	$\theta(n)$
Ordered Filter	$\theta(n+\omega k)^*$	$O(\omega \log n)^*$	$\theta(\log n)$
List Contraction	$\theta(n+\omega)$	$O(\omega \log n)^*$	$\theta(\omega)$
Tree Contraction	$\theta(n+\omega)$	$O(\omega \log n)^*$	$\theta(\omega)$
Minimum Spanning Tree	$O\left(\alpha(n)m + \omega n \log\left(\min\left(\frac{m}{n},\omega\right)\right)\right)$	$O(\omega \operatorname{polylog}(m))^*$	$O\left(\frac{m}{\left(n\cdot\log\left(\min\left(\frac{m}{n},\omega\right)\right)\right)}\right)$
2D Convex Hull	$O(n\log k+\omega n\log\log k)^\wedge$	$O\left(\omega(\log n)^2\right)^*$	Output Sensitive
BFS Tree	$\theta(\omega n+m)^\wedge$	$\theta(\omega\delta\log n)^*$	$O\left(\frac{m}{n}\right)$

k = output size
δ = graph diameter
α = inverse Ackerman function

* = with high probability
$^\wedge$ = expected

[BBFGGMS16]

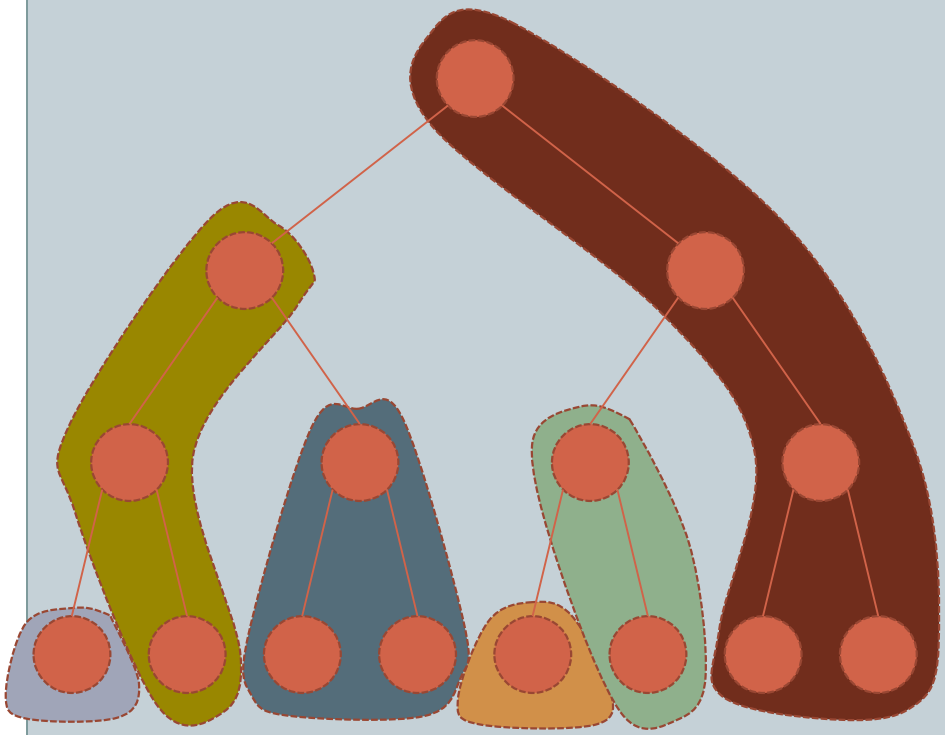
Tree Contraction: Current Methods



- Rake leaves
- Compress Chains

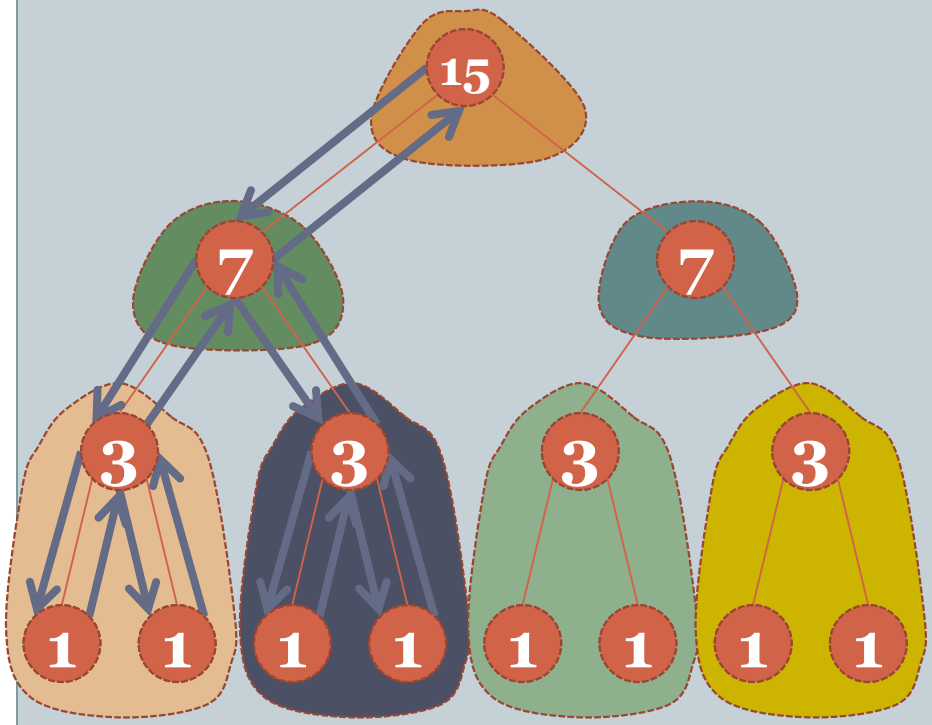
- Each rake or compress operation costs a write
- Total number of rakes and compresses is $\theta(n)$
- Work is $\theta(\omega n)$
- Span is $\theta(\omega \log n)$

Tree Contraction: High-level Approach



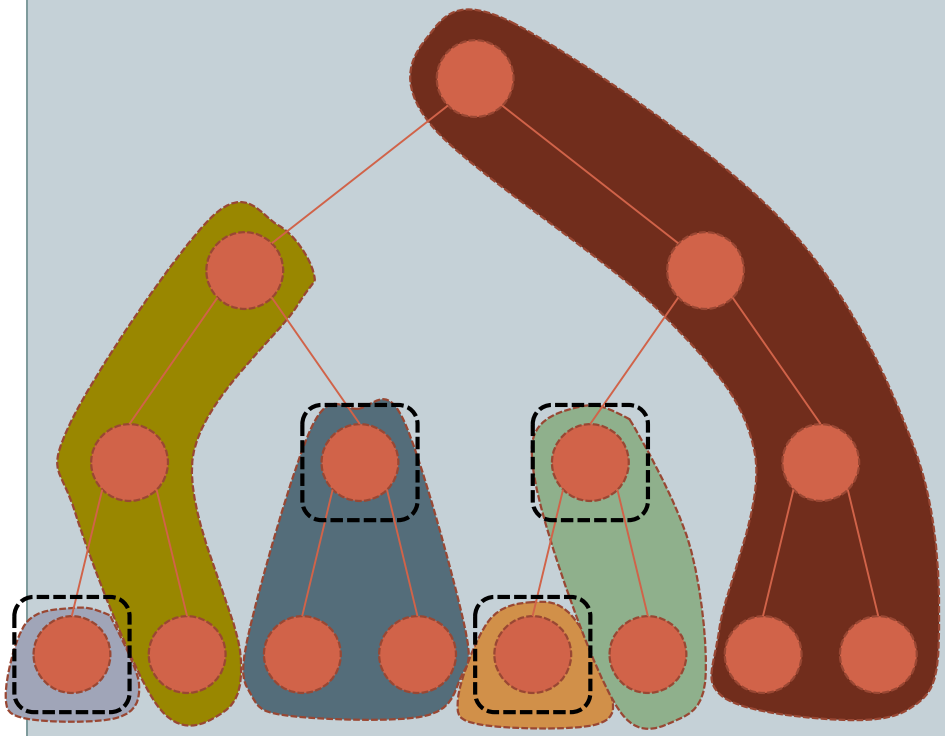
- Assume that M_L is $\Omega(\omega)$
- Partition the tree into $\theta\left(\frac{n}{\omega}\right)$ components of size $\theta(\omega)$
- Sequentially contract each component
- Use a classic parallel algorithm to contract the resulting tree of size $\theta\left(\frac{n}{\omega}\right)$

Tree Contraction: Classic Partitioning



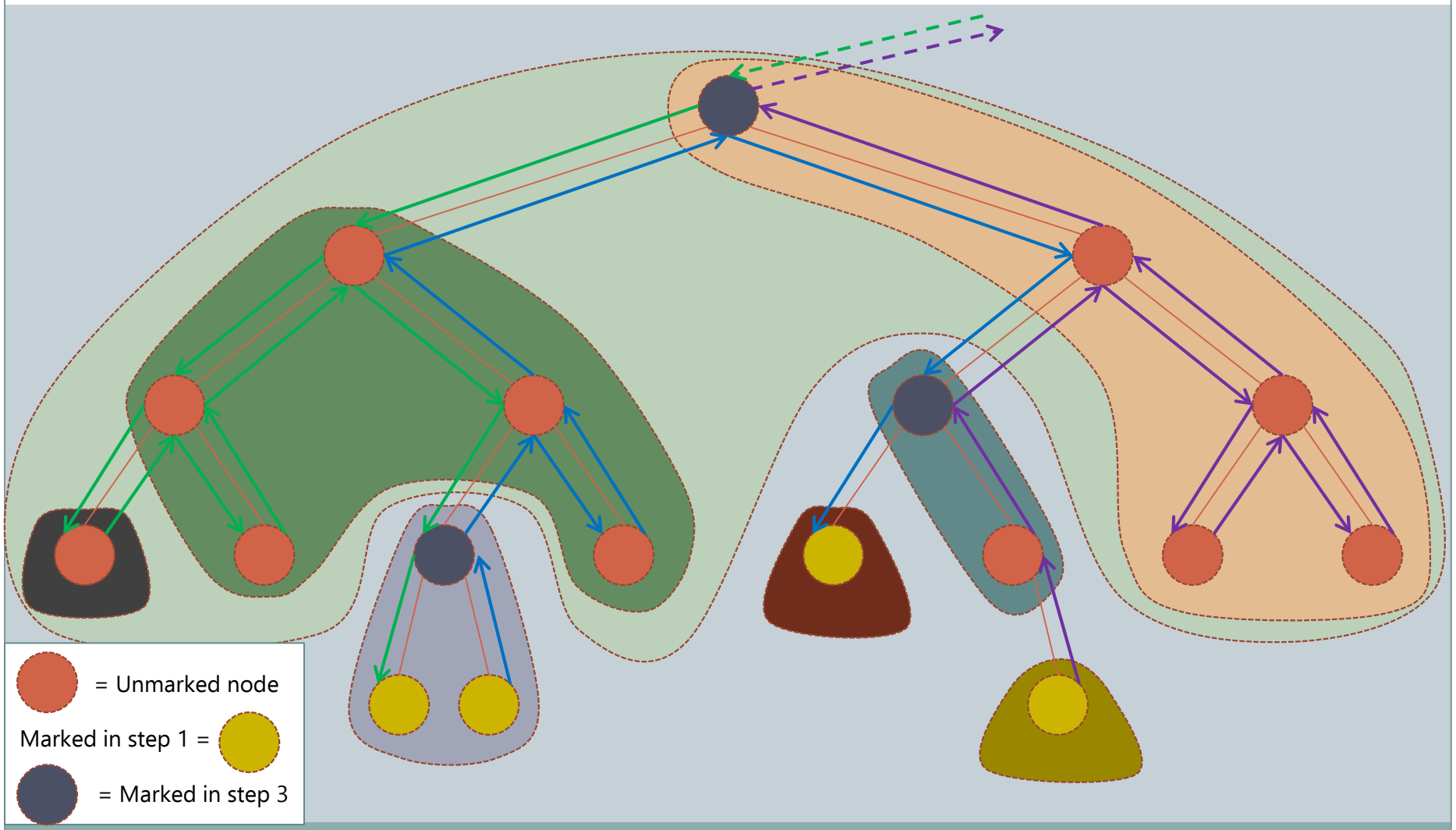
- Follow the Euler Tour
- Generate subtree size
- Find the m-critical points
- Partition the tree
- Requires a write for each node

Tree Contraction: Write-Efficient Partitioning



- Mark each node with probability $\frac{1}{\omega}$
- Traverse the Euler Tour from each marked node and mark every ω^{th} node
- Mark the highest node on each path between marked nodes
- Each marked node starts a new component

Tree Contraction: Contract-ability of Partitions



Tree Contraction: A New Approach

- Assume that M_L is $\Omega(\omega)$
- Partition the tree into $\theta\left(\frac{n}{\omega}\right)$ components of size $\theta(\omega)$
- Sequentially contract each component
- Use a classic algorithm to contract the resulting tree of size $\theta\left(\frac{n}{\omega}\right)$

Work: $\theta\left(n + \frac{n}{\omega} * \omega\right) + \theta\left(n + \frac{n}{\omega} * \omega\right) + \theta(n) = \theta(n + \omega)$

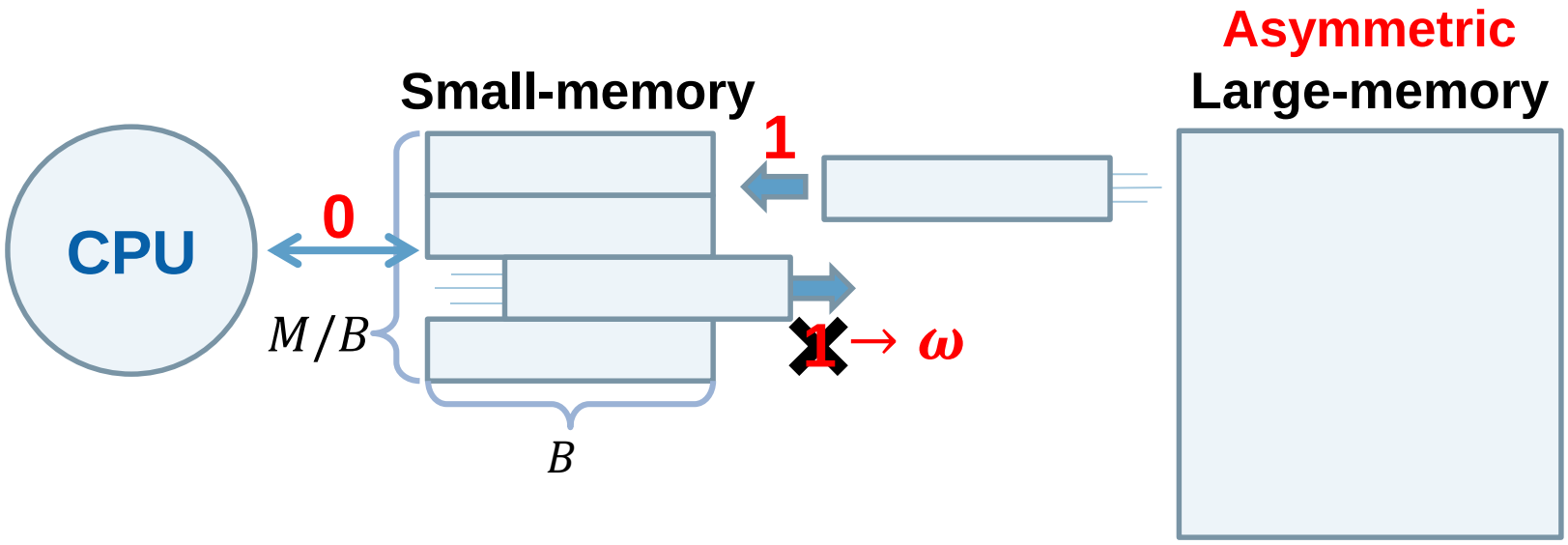
Span: $\theta(\omega \log n) + \theta(\omega) + \theta\left(\omega \log\left(\frac{n}{\omega}\right)\right) = \theta(\omega \log n)$

The Asymmetric External Memory model

[BFGGS15]

- **AEM has two *memory transfer* instructions:**
  - **Read transfer**: load a block from large-memory
  - **Write transfer**: write a block to large-memory
- **The complexity of an algorithm on the AEM model (I/O complexity) is measured by:**

$$\#(read\ transfer) + \omega \cdot \#(write\ transfer)$$



Sorting algorithms on the Asymmetric EM model

- **Sorting n records in AEM model has I/O complexity of**

$$O\left(\frac{\omega n}{B}\log_{\frac{\omega M}{B}}\frac{n}{B}\right)$$

can be achieved by:

- Multi-way mergesort
- Sample sort
- Heap sort based on buffer trees

- **Matching lower bound** [Sitchinava16]
  - No asymptotic advantage whenever ω is $O(M^c)$ for a constant c
  - Depressing…because so many problems can't beat an EM sorting lower bound

Key Take-Aways

- **Main memory will be persistent and asymmetric**
  - Reads much cheaper than Writes
- **Very little work to date on Asymmetric Memory**
  - Not quite: space complexity, CRQW, RMRs, Flash,…
- **Highlights of our results to date:**
  - Models: (M,ω)-ARAM; with parallel & block variants
  - Asymmetric memory is not like symmetric memory
  - New techniques for old problems
  - Lower bounds for block variant are depressing

⬅ You Are Here

Thanks to Collaborators


Naama Ben-David


Guy Blelloch


Jeremy Fineman


Yan Gu


Charles McGuffey


Julian Shun

(Credit to Yan and Charlie for some of these slides)

& Sponsors

- National Science Foundation
- Natural Sciences and Engineering Research Council of Canada
- Miller Institute for Basic Research in Sciences at UC Berkeley
- Intel (via ISTC for Cloud Computing & new ISTC for Visual Cloud)

References

(in order of first appearance)

[KPMWEVNBPA14] Ioannis Koltsidas, Roman Pletka, Peter Mueller, Thomas Weigold, Evangelos Eleftheriou, Maria Varsamou, Athina Ntalla, Elina Bougioukou, Aspasia Palli, and Theodore Antonakopoulos. PSS: A Prototype Storage Subsystem based on PCM. NVMW, 2014.

[DJX09] Xiangyu Dong, Norman P. Jouupi, and Yuan Xie. PCRAMsim: System-level performance, energy, and area modeling for phase-change RAM. ACM ICCAD, 2009.

[XDJX11] Cong Xu, Xiangyu Dong, Norman P. Jouppi, and Yuan Xie. Design implications of memristor-based RRAM cross-point structures. IEEE DATE, 2011.

[GZDCH13] Nad Gilbert, Yanging Zhang, John Dinh, Benton Calhoun, and Shane Hollmer, "A 0.6v 8 pj/write non-volatile CBRAM macro embedded in a body sensor node for ultra low energy applications", IEEE VLSIC, 2013.

[GMR98] Phillip B. Gibbons, Yossi Matias, and Vijaya Ramachandran. The Queue-Read Queue-Write PRAM Model: Accounting for Contention in Parallel Algorithms, SIAM J. on Computing 28(2), 1998.

[DHW97] Cynthia Dwork, Maurice Herlihy, and Orli Waarts. Contention in Shared Memory Algorithms. ACM STOC, 1993.

[YA95] Jae-Heon Yang and James H. Anderson. A Fast, Scalable Mutual Exclusion Algorithm. Distributed Computing 9(1), 1995.

[GT05] Eran Gal and Sivan Toledo. Algorithms and data structures for flash memories. ACM Computing Surveys, 37(2), 2005.

[EGMP14] David Eppstein, Michael T. Goodrich, Michael Mitzenmacher, and Pawel Pszona. Wear minimization for cuckoo hashing: How not to throw a lot of eggs into one basket. ACM SEA, 2014.

[BFGGS16] Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, Yan Gu, and Julian Shun. Efficient Algorithms with Asymmetric Read and Write Costs. ESA, 2016.

[HK81] Jia-Wei Hong and H. T. Kung. I/O complexity: The red-blue pebble game. ACM STOC, 1981.

[CS76] Stephen Cook and Ravi Sethi. Storage requirements for deterministic polynomial time recognizable languages. JCSS, 13(1), 1976.

References (cont.)

[BFGGS15] Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, Yan Gu, and Julian Shun. Sorting with Asymmetric Read and Write Costs. ACM SPAA, 2015.

[BBFGGMS16] Naama Ben-David, Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, Yan Gu, Charles McGuffey, and Julian Shun. Parallel Algorithms for Asymmetric Read-Write Costs. ACM SPAA, 2016.

[Sitchinava16] Nodari Sitchinava, personal communication, June 2016.

Some additional related work:

[CDGKKSS16] Erin Carson, James Demmel, Laura Grigori, Nicholas Knight, Penporn Koanantakool, Oded Schwartz, and Harsha Vardhan Simhadri. Write-Avoiding Algorithms. IEEE IPDPS, 2016.

[CGN11] Shimin Chen, Phillip B. Gibbons, and Suman Nath. Rethinking Database Algorithms for Phase Change Memory. CIDR, 2011.

[Viglas14] Stratis D. Viglas. Write-limited sorts and joins for persistent memory. VLDB 7(5), 2014.