## Lecture 14:

# Random Stuff

# (graphs, n-body, FFT, wavefront)


**Modern Parallel Computing**

**John Owens**

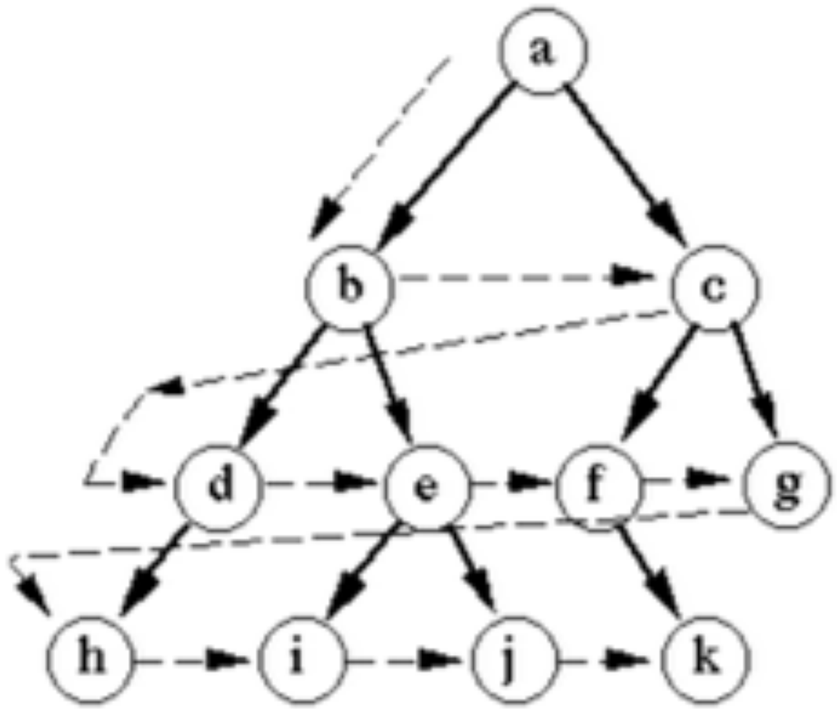
**EEC 289Q, UC Davis, Winter 2018**

# BFS on Graphs

. Duane Merrill, Michael Garland, and Andrew Grimshaw. Scalable GPU graph traversal. In *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '12, pages 117–128, February 2012.

- *Unknowingly, Benwen Zhang (Delaware) also supplied slides. Thanks!*

# Introduction




Breadth-first search

**Breadth-first search (BFS) is a core primitive for graph traversal and a basis for many higher-level graph analysis algorithms.**

**CISC 879 : Advanced Parallel Programming**

# Background

## Parallel breadth-first search

### 1. Quadratic parallelizations

- inspect every edge or every vertex during every iteration
- work complexity is $O(n^2+m)$ as there may $n$ BFS iterations in the worst case

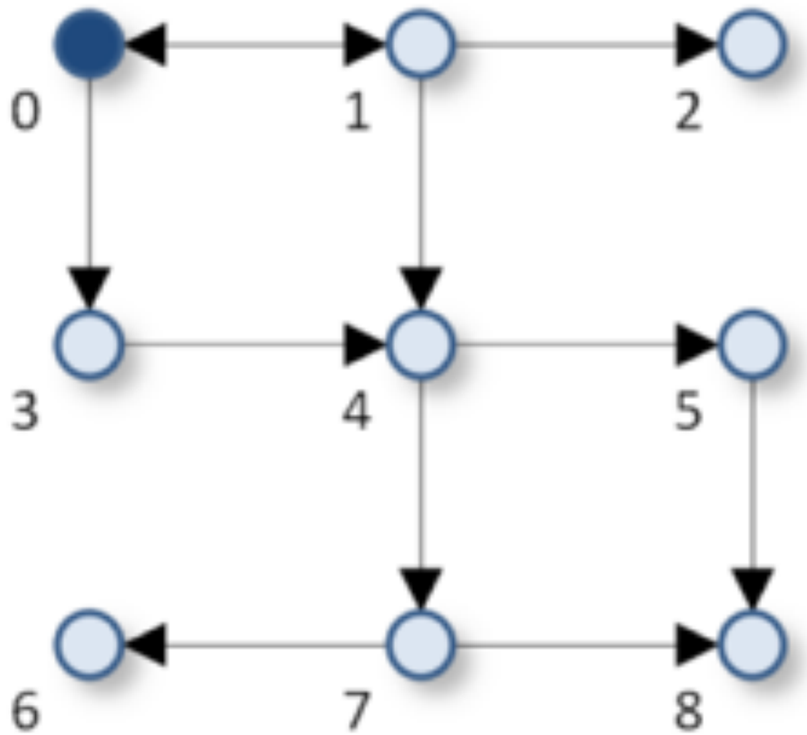
### 2. Linear parallelizations

- each iteration examine only the edges and vertices in that iteration's logical edge and vertex-frontiers, respectively.
- work-efficient parallel BFS algorithm should perform $O(n+m)$ work

### 3. Distributed parallelizations

- partition the graph structure amongst multiple processors, particularly for very large datasets that that are too large to fit within the main memory of a single node.

# Background

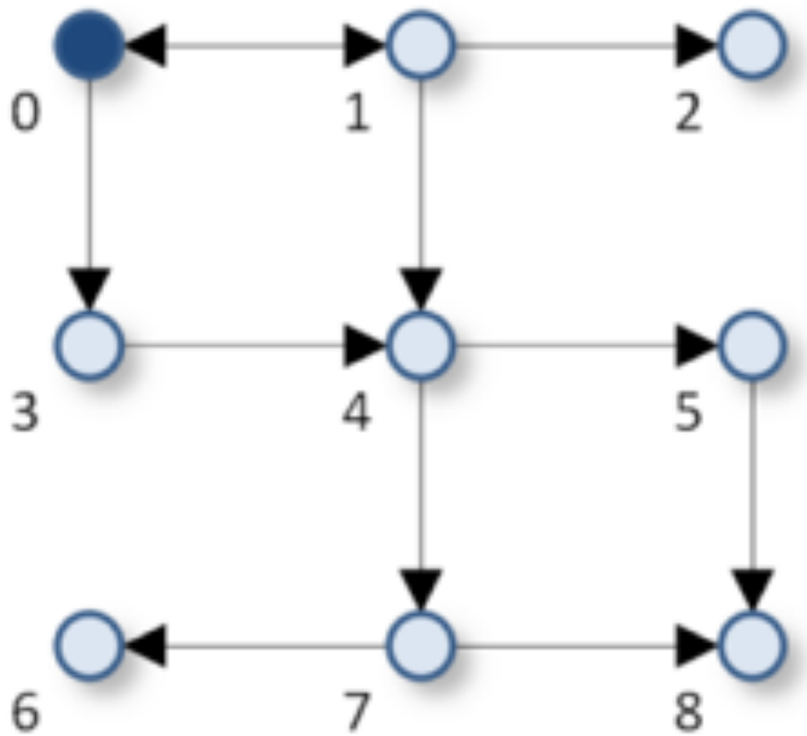



| Traversal from source vertex $v_0$ | | |
|---|---|---|
| **BFS Iteration** | **Vertex frontier** | **Edge frontier** |
| 1 | {0} | {1,3} |
| 2 | {1,3} | {0,2,4,4} |
| 3 | {2,4} | {5,7} |
| 4 | {5,7} | {6,8,8} |
| 5 | {6,8} | {} |

**BFS Graph Traversal**

**CISC 879 : Advanced Parallel Programming**

# Background




*C:*

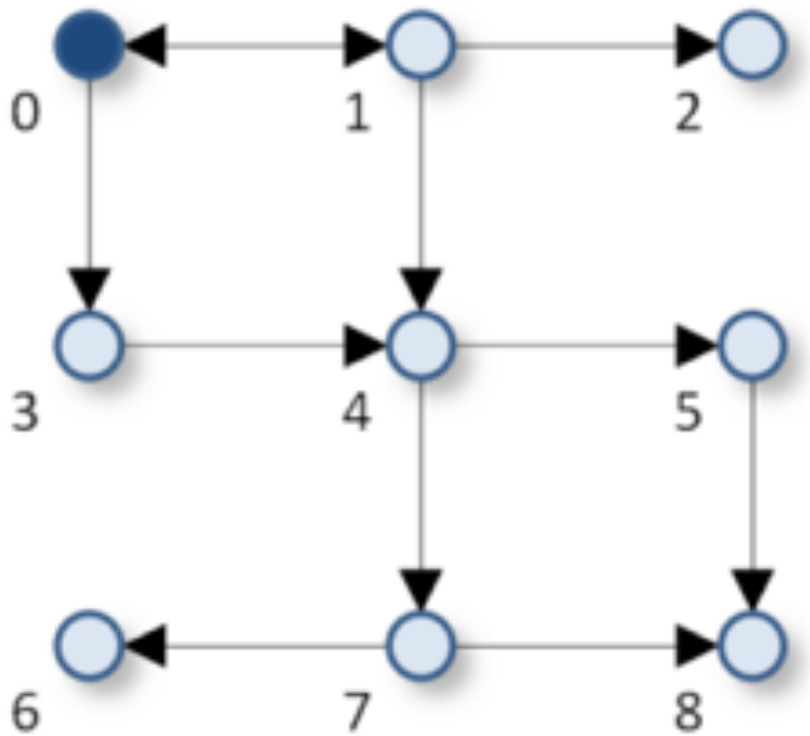
| 1,3 | 0,2,4 | 4 | 5,7 | 8 | 6,8 |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

*R:*

| 0 | 2 | 5 | 5 | 6 | 8 | 9 | 9 | 11 | 11 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

**compressed sparse row (CSR) sparse matrix format**

# Background




```
1     Q := {}
2     for i in 0 .. |V|-1:
3       dist[i] := ∞
4     dist[s] := 0
5     Q.Enqueue(s)
6     while (Q != {}) :
7       i = Q.Dequeue()
8       for offset in R[i] .. R[i+1]-1 :
9         j := C[offset]
10        if (dist[j] == ∞)
11          dist[j] := dist[i] + 1;
12          Q.Enqueue(j)
```

**Sequential Breadth-First Search Algorithm**

**CISC 879 : Advanced Parallel Programming**

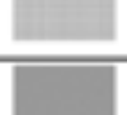
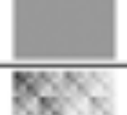
| Name | Sparsity Plot | Description | $n$ ($10^6$) | $m$ ($10^6$) | $\overline{d}$ | Avg. Search Depth |
|---|---|---|---|---|---|---|
| europe.osm |  | European road network | 50.9 | 108.1 | 2.1 | 19314 |
| grid5pt.5000 |  | 5-point Poisson stencil (2D grid lattice) | 25.0 | 125.0 | 5.0 | 7500 |
| hugebubbles-00020 |  | Adaptive numerical simulation mesh | 21.2 | 63.6 | 3.0 | 6151 |
| grid7pt.300 |  | 7-point Poisson stencil (3D grid lattice) | 27.0 | 188.5 | 7.0 | 679 |
| nlpkkt160 |  | 3D PDE-constrained optimization | 8.3 | 221.2 | 26.5 | 142 |
| audikw1 |  | Automotive finite element analysis | 0.9 | 76.7 | 81.3 | 62 |
| cage15 |  | Electrophoresis transition probabilities | 5.2 | 94.0 | 18.2 | 37 |
| kkt_power |  | Nonlinear optimization (KKT) | 2.1 | 13.0 | 6.3 | 37 |
| coPapersCiteseer |  | Citation network | 0.4 | 32.1 | 73.9 | 26 |
| wikipedia-20070206 |  | Links between Wikipedia pages | 3.6 | 45.0 | 12.6 | 20 |
| kron_g500-logn20 |  | Graph500 RMAT ($A$=0.57, $B$=0.19, $C$=0.19) | 1.0 | 100.7 | 96.0 | 6 |
| random.2Mv.128Me |  | $G(n, M)$ uniform random | 2.0 | 128.0 | 64.0 | 6 |
| rmat.2Mv.128Me |  | RMAT ($A$=0.45, $B$=0.15, $C$=0.15) | 2.0 | 128.0 | 64.0 | 6 |

**Table 1.** Suite of benchmark graphs

# Possible load-balance strategies

- Assign vertices to threads
  - Advantage: Simple
  - Disadvantage: Load balance
  - Analog: SpMV row per thread
- Assign edges to threads
  - Advantage: Better load balancing
  - Disadvantage: High set-up cost
  - Analog: SpMV element per thread

# Isolated neighbor-gathering

Serial gathering


**(a) serial**

each thread serially expand neighbors from the column-indices array $C$.

non-uniform degree distributions can impose significant load imbalance between threads

# Isolated neighbor-gathering

Coarse-grained, warp*-based gathering

(b) coarse-grained, warp-based cooperative expansion
(emphasis on controlling thread)

each thread enlists its entire warp to gather its assigned adjacency list

this approach can suffer underutilization within the warp

*Warp: the set of 32 parallel threads that execute a SIMD instruction

# Isolated neighbor-gathering

Fine-grained, scan-based gathering


**(c) fine-grained, scan-based cooperative expansion**

Threads construct a shared array of column-indices offsets corresponding to a CTA**-wide concatenation of their assigned adjacency lists.

the entire CTA to gather the referenced neighbors from the column-indices array *C* using this perfectly packed gather vector.

workload imbalance can occur in the form of underutilized cycles during offset-sharing

****A CTA is an array of concurrent threads that cooperate to compute a result**

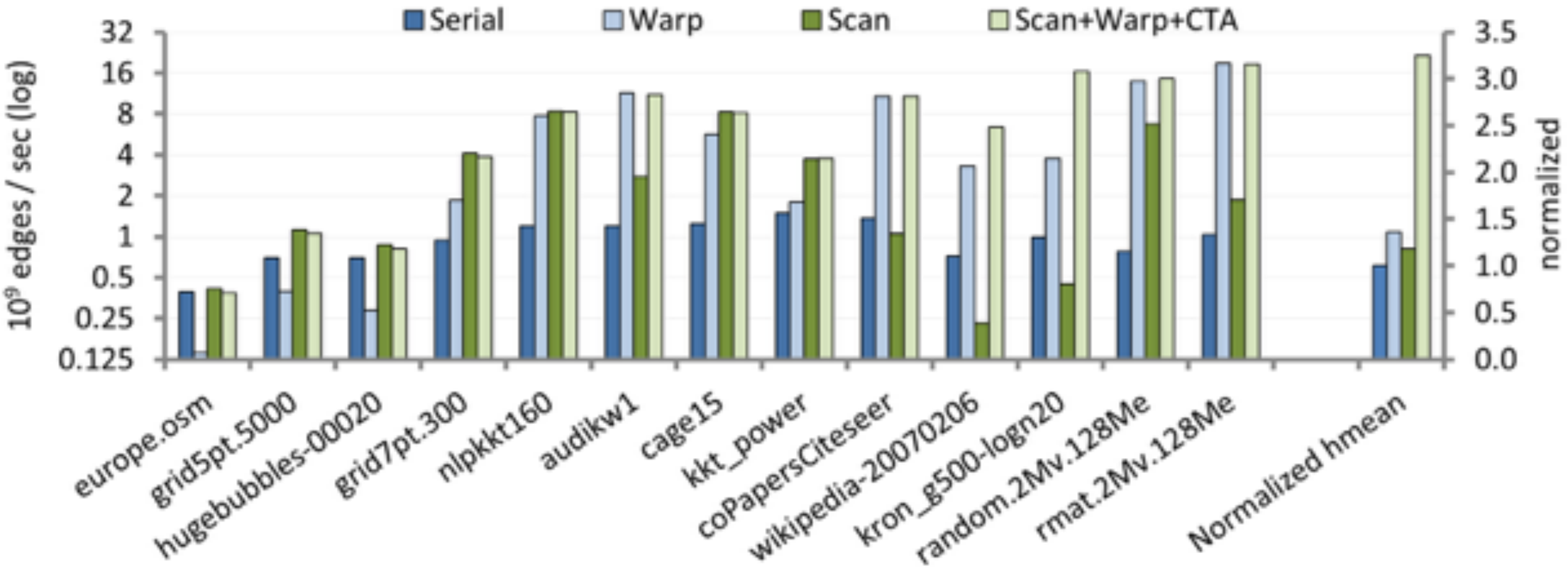
# Isolated neighbor-gathering

## Scan+warp+CTA gathering(Hybrid)

We can further mitigate inter-warp workload imbalance by introducing a third granularity of thread-enlistment: the entire CTA.

- **CTA-wide gathering** process very large adjacency lists.
- apply **warp-based gathering** to acquire adjacency smaller than the CTA size, but greater than the warp width.
- perform **scan-based gathering** to efficiently acquire the remaining "loose ends".

# neighbor-gathering analysis

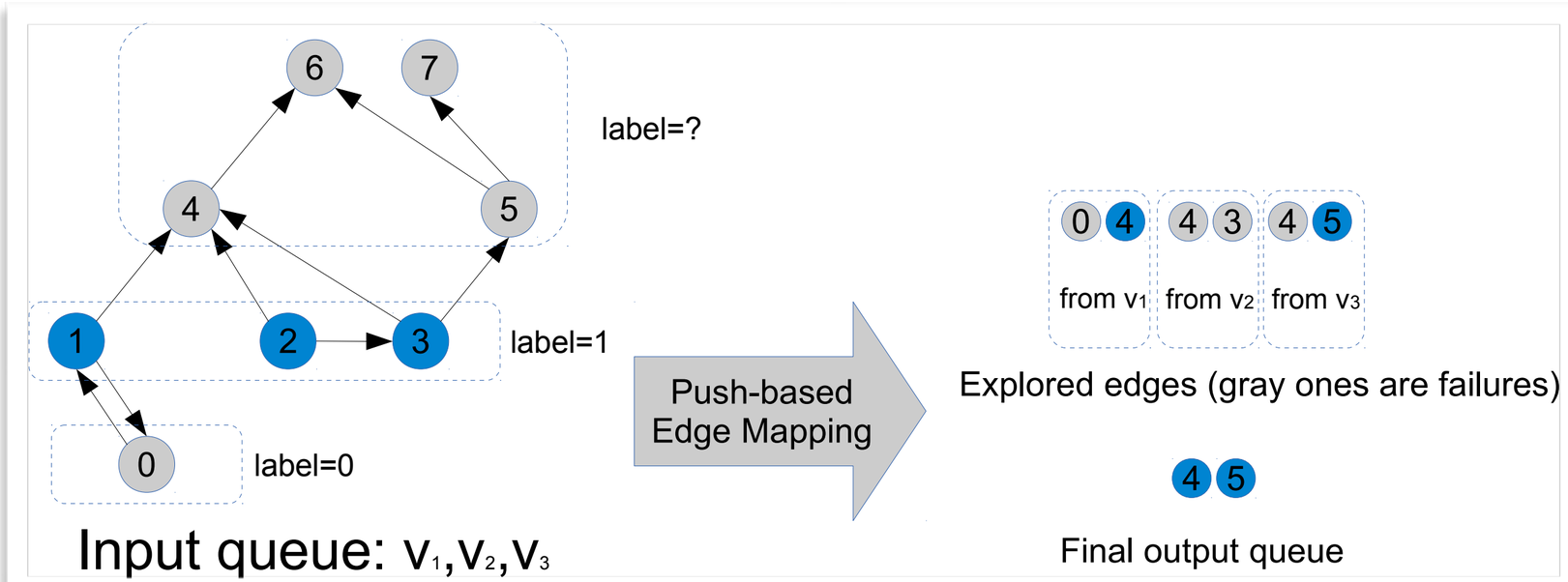



**(a) Average gather rate (log)**

**CISC 879 : Advanced Parallel Programming**

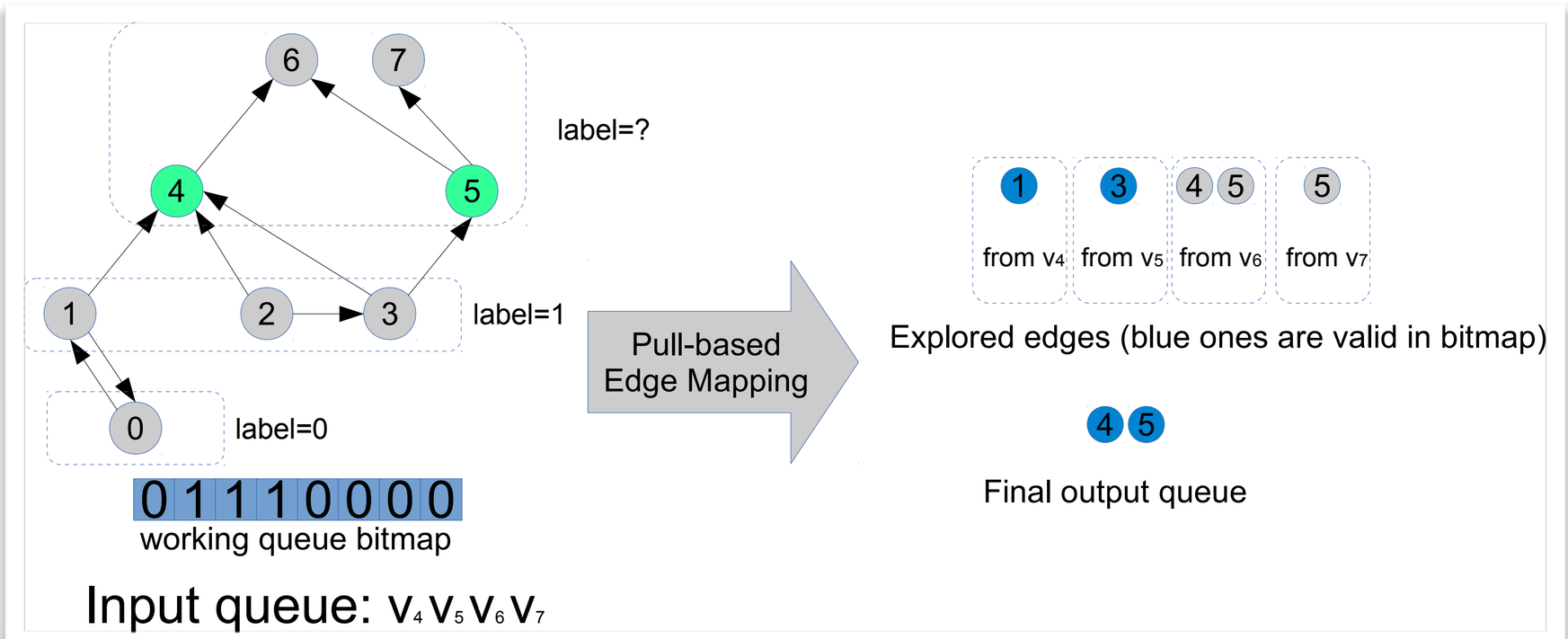
# Push vs. Pull

- Normal operation (“push”): vertex frontier visits every outbound edge, generates list of connected unvisited vertices
  - Works great when you’re expanding: more new vertices than old
  - Works poorly with few unvisited vertices



# Push vs. Pull

- We also support pull: Start with unvisited vertices, check which have inbound edges from frontier



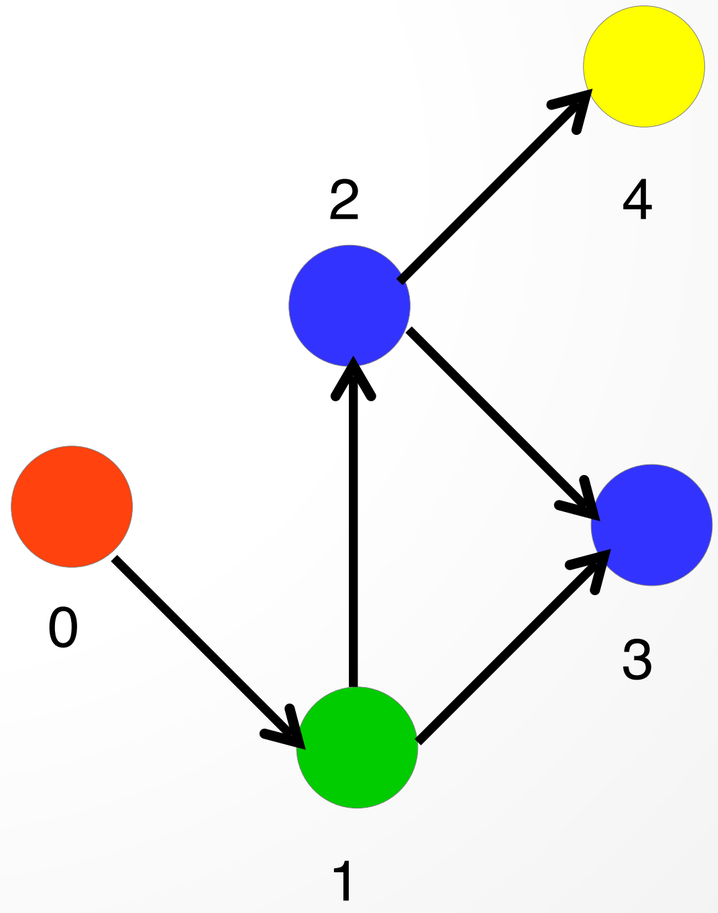

- Difficult to express in compute-focused APIs
- Gunrock: Frontier is unvisited vertices; pull from that frontier

# Graph BLAS

- Thanks to Aydin Buluc (LBNL) and Carl Yang (UC Davis)

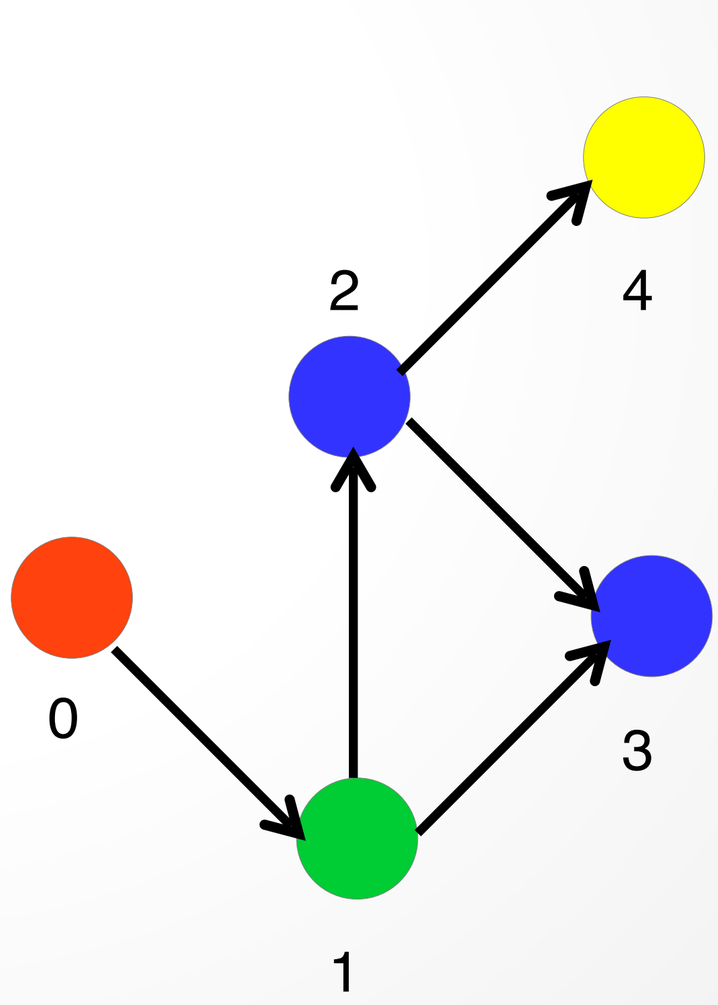
# Duality between SpMV and BFS

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0
\end{bmatrix}
\times
\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}
=
\begin{bmatrix} 0 \\ - \\ - \\ - \\ - \end{bmatrix}
$$



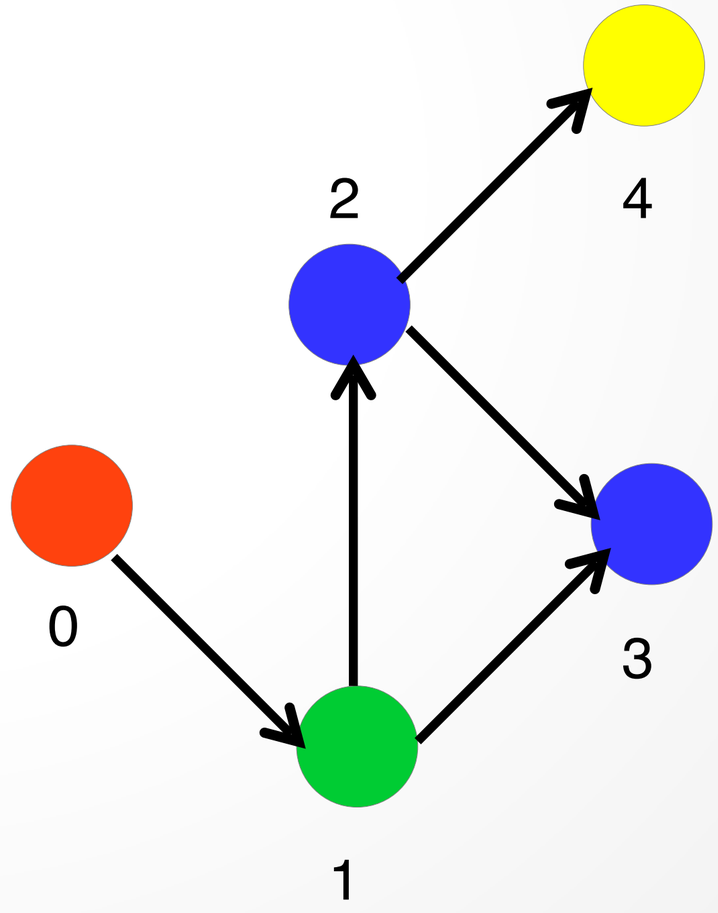
# Duality between SpMV and BFS

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | x | 0 | = | - |
| 0 | 1 | 1 | 0 | 0 | | 0 | 0 | - |
| 0 | 0 | 1 | 1 | 0 | | 0 | 0 | - |



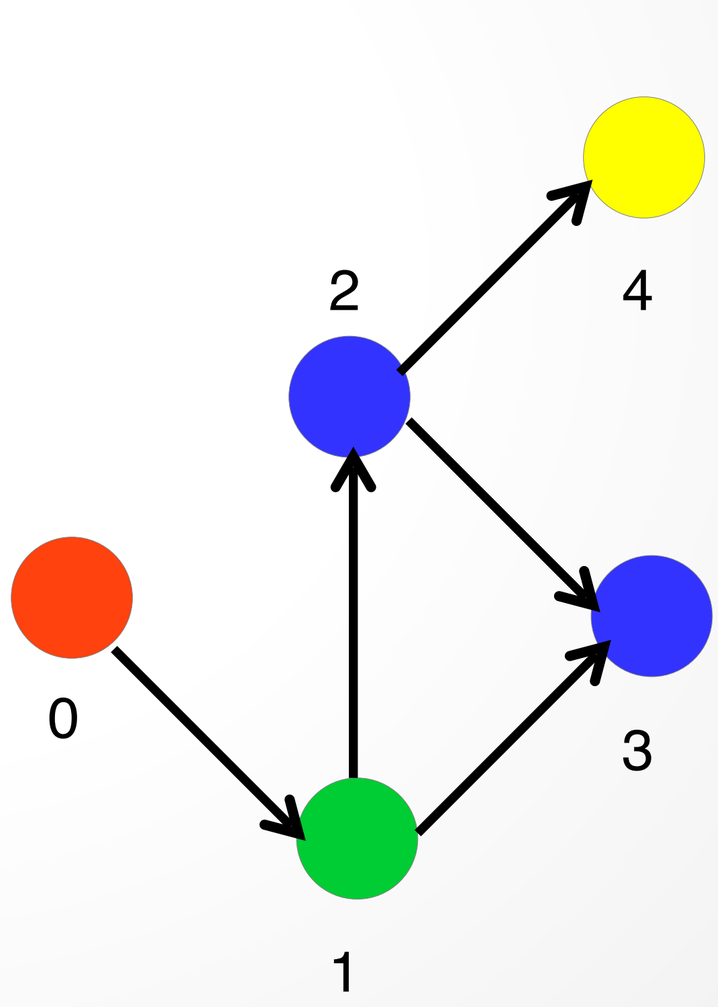
# Duality between SpMV and BFS

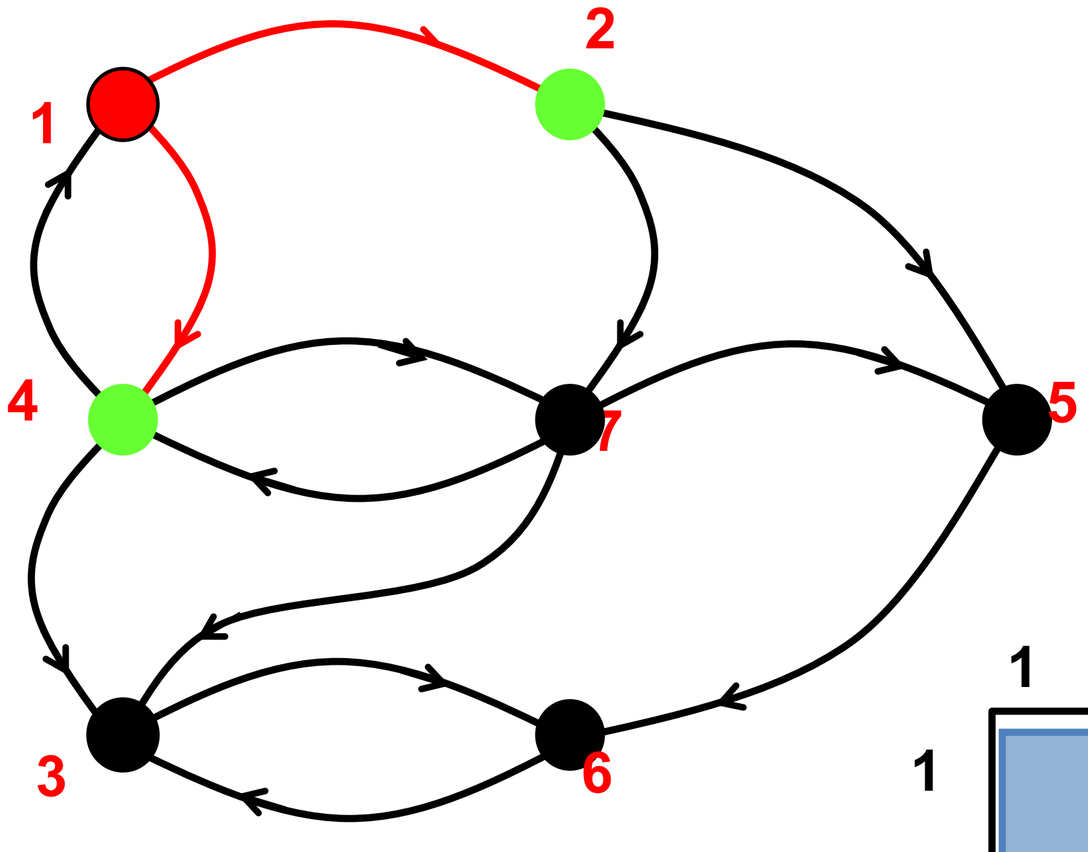
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | x | 0 | = | 1 | 2 |
| 0 | 1 | 1 | 0 | 0 | | 0 | 1 | 2 |
| 0 | 0 | 1 | 1 | 0 | | 0 | 0 | - |



# Duality between SpMV and BFS

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | x | 1 | = 0 | 2 |
| 0 | 1 | 1 | 0 | 0 | | 1 | 1 | 2 |
| 0 | 0 | 1 | 1 | 0 | | 0 | 1 | 3 |






Particular semiring operations:

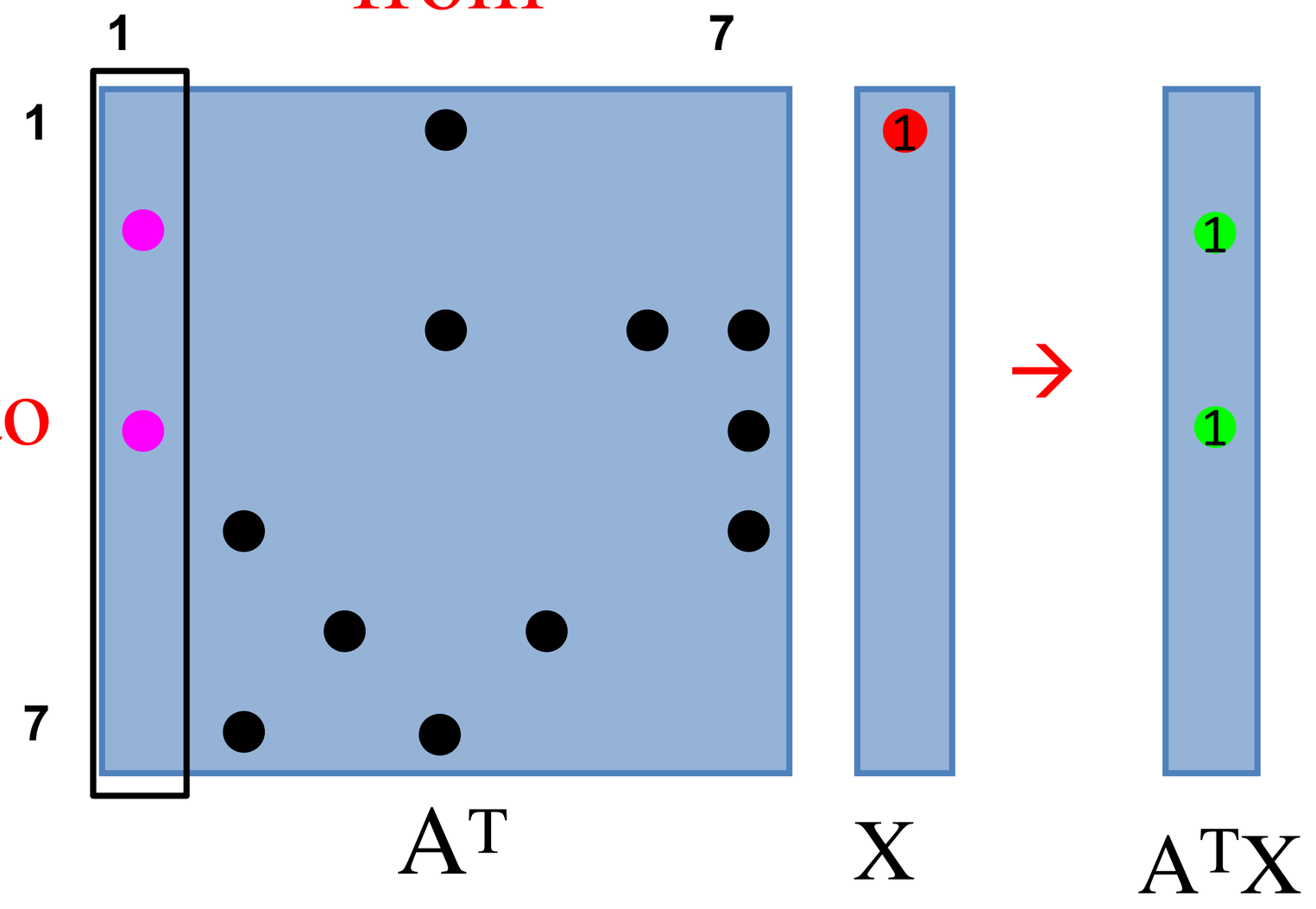
**Multiply:** select

**Add:** minimum

parents:


from

to



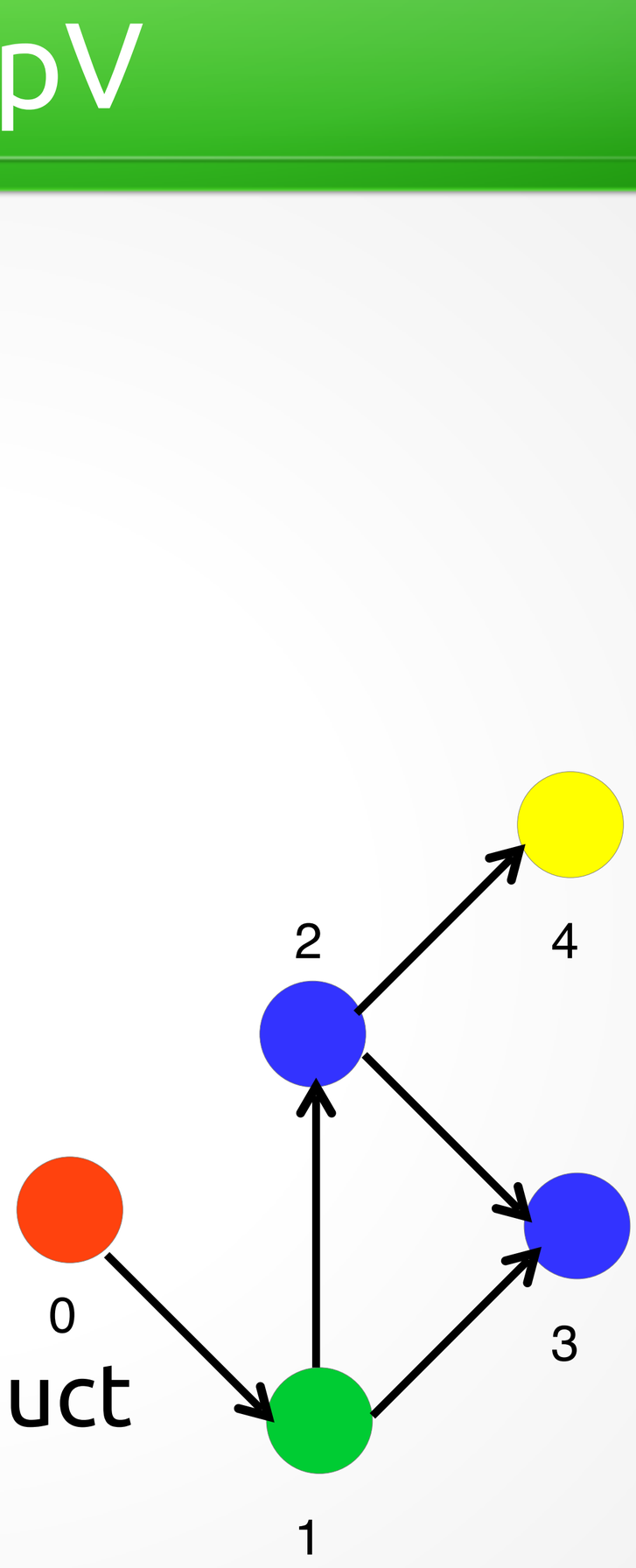
# Examples of semirings in graph algorithms

| | |
|---|---|
| Real field: **(R, +, x)** | Classical numerical linear algebra |
| Boolean algebra: **({0 1}, \|, &)** | Graph traversal |
| Tropical semiring: **(R U {∞}, min, +)** | Shortest paths |
| **(S, select, select)** | Select subgraph, or contract nodes to form quotient graph |
| (edge/vertex attributes, vertex data aggregation, edge data processing) | Schema for user-specified computation at vertices and edges |
| **(R, max, +)** | Graph matching &network alignment |
| **(R, min, times)** | Maximal independent set |

- **Shortened semiring notation: (Set, Add, Multiply)**. Both identities omitted.
- **Add:** Traverses edges, **Multiply:** Combines edges/paths at a vertex
- Neither add nor multiply needs to have an inverse.
- Both **add** and **multiply** are **associative**, **multiply distributes** over **add**

# SpMSpV

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | x | 1 | = 0 | 2 |
| 0 | 1 | 1 | 0 | 0 | | 1 | 1 | 2 |
| 0 | 0 | 1 | 1 | 0 | | 0 | 1 | 3 |
| 0 | 0 | 1 | 1 | 0 | | | | |



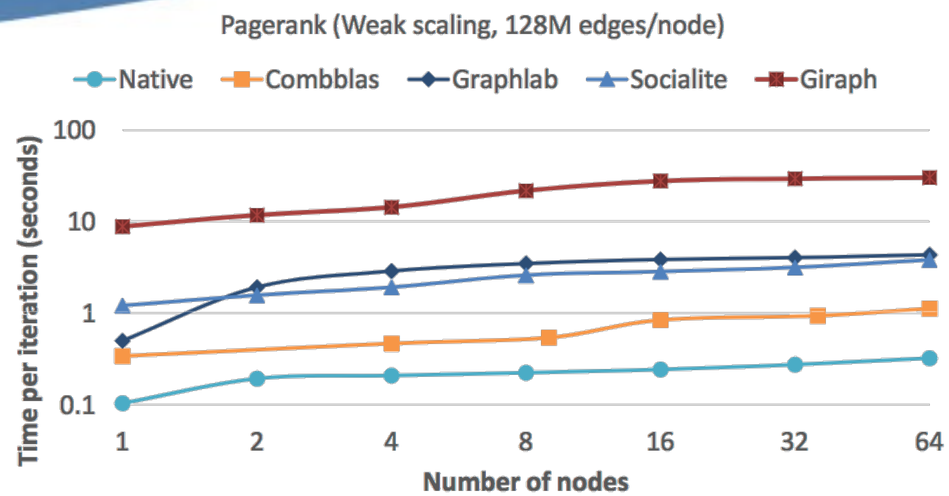

Want: Use (2 3) to find product

# Performance of Linear Algebraic Graph Algorithms

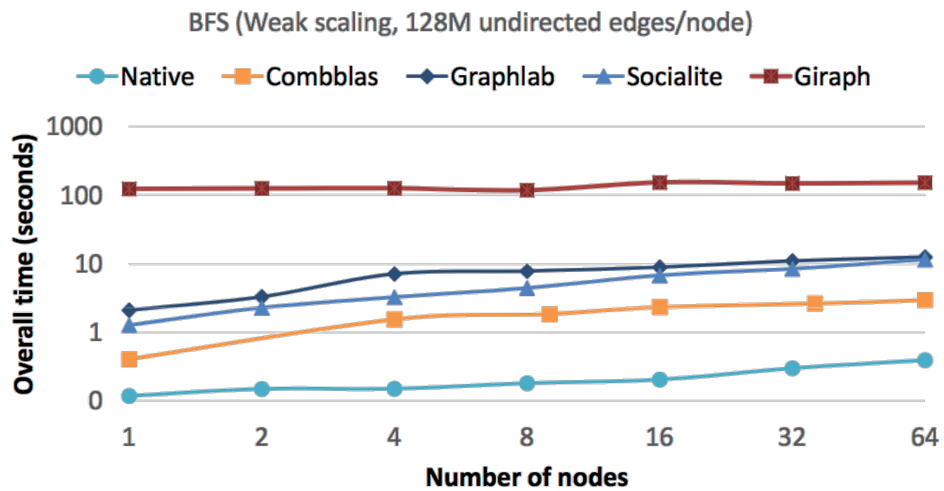
*Combinatorial BLAS fastest among all tested graph processing frameworks on 3 out of 4 benchmarks in an independent study by Intel.*

*The linear algebra abstraction enables high performance, within 4X of native performance for PageRank and Collaborative filtering.*

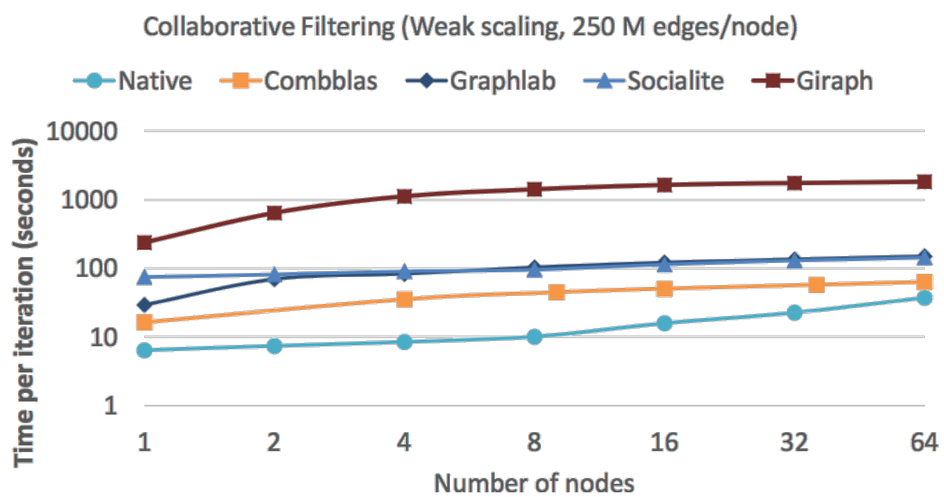
Satish, Nadathur, et al. "Navigating the Maze of Graph Analytics Frameworks using Massive Graph Datasets", in SIGMOD'14



(a) PageRank



(b) Breadth-First Search



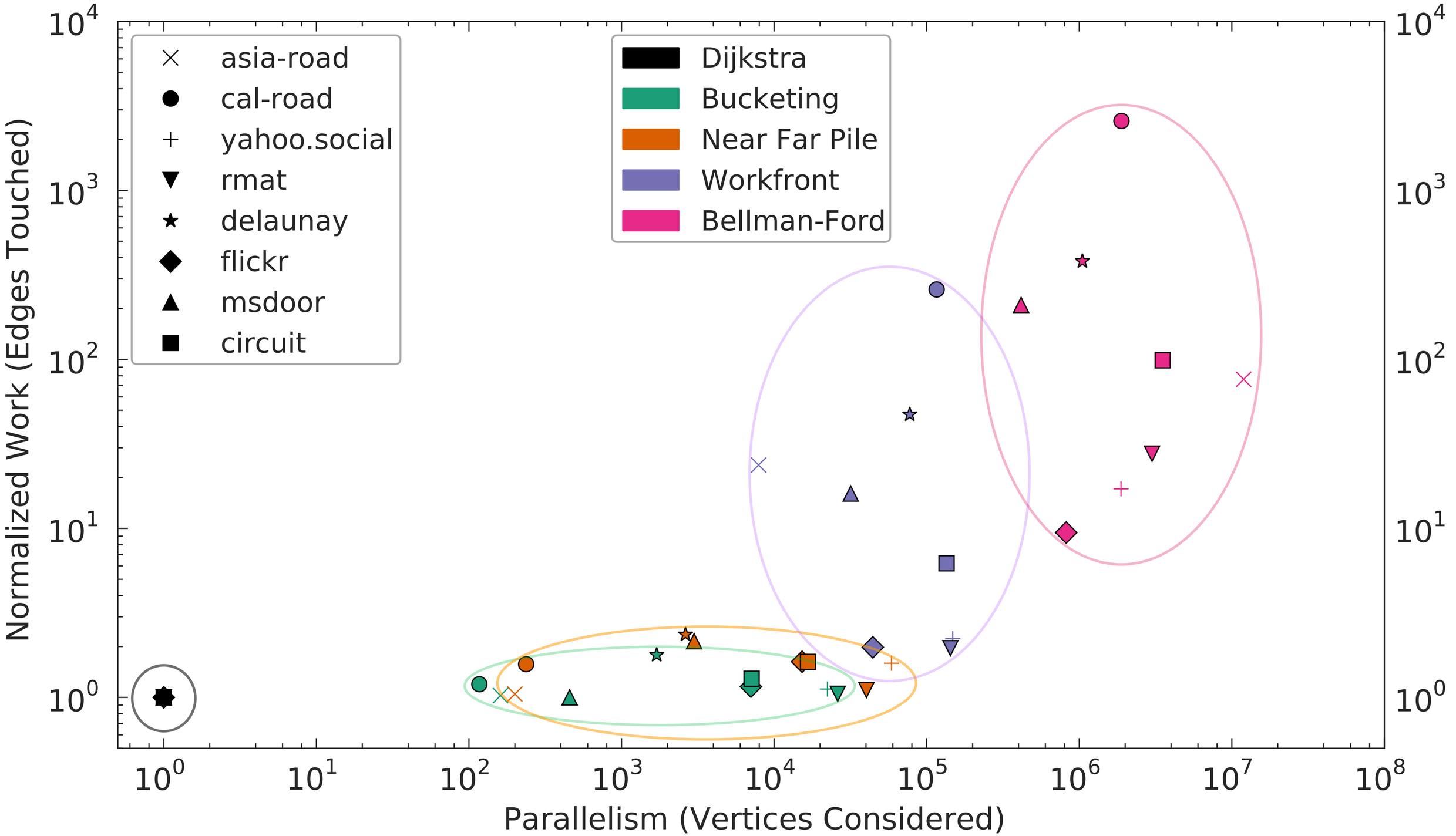
(c) Collaborative Filtering

# Broader Graph Concerns

# Graph challenges on GPUs

- Load-balancing due to irregularity
- Efficient parallel algorithms
  - Different balance between brute-force and elegant than on CPUs (next slide)
- Moving beyond simple algorithms
- Graph representations
- Scalability (memory constraints)

# Algorithm example: SSSP




*Andrew Davidson, Sean Baxter, Michael Garland, and John D. Owens.* ***Work-Efficient Parallel GPU Methods for Single Source Shortest Paths****. In Proceedings of the 28th IEEE International Parallel and Distributed Processing Symposium, May 2014.*

# Industry interest examples

- Twitter: “Who To Follow” service
  - Historically: SALSA (“Stochastic Approach for Link-Structure Analysis”)
  - Personalized PageRank generates circle of trust
  - Hubs & authorities, random walks
- Facebook
  - PageRank & Personalized PageRank
  - Label propagation
  - Graph embeddings into $R^n$ so similar nodes are close

# Industry interest examples

- Twitter: “Who To Follow” service
  - Historically: SALSA (“Stochastic Approach for Link-Structure Analysis”)
  - Personalized PageRank generates circle of trust
  - Hubs & authorities, random walks
- Facebook
  - PageRank & Personalized PageRank
  - Label propagation
  - Graph embeddings into $R^n$ so similar nodes are close

*Afton Geil, Yangzihao Wang, and John D. Owens.* ***WTF, GPU! Computing Twitter's Who-To-Follow on the GPU****. In Proceedings of the Second ACM Conference on Online Social Networks, COSN '14, pages 63–68, October 2014.*

# Graph Matching

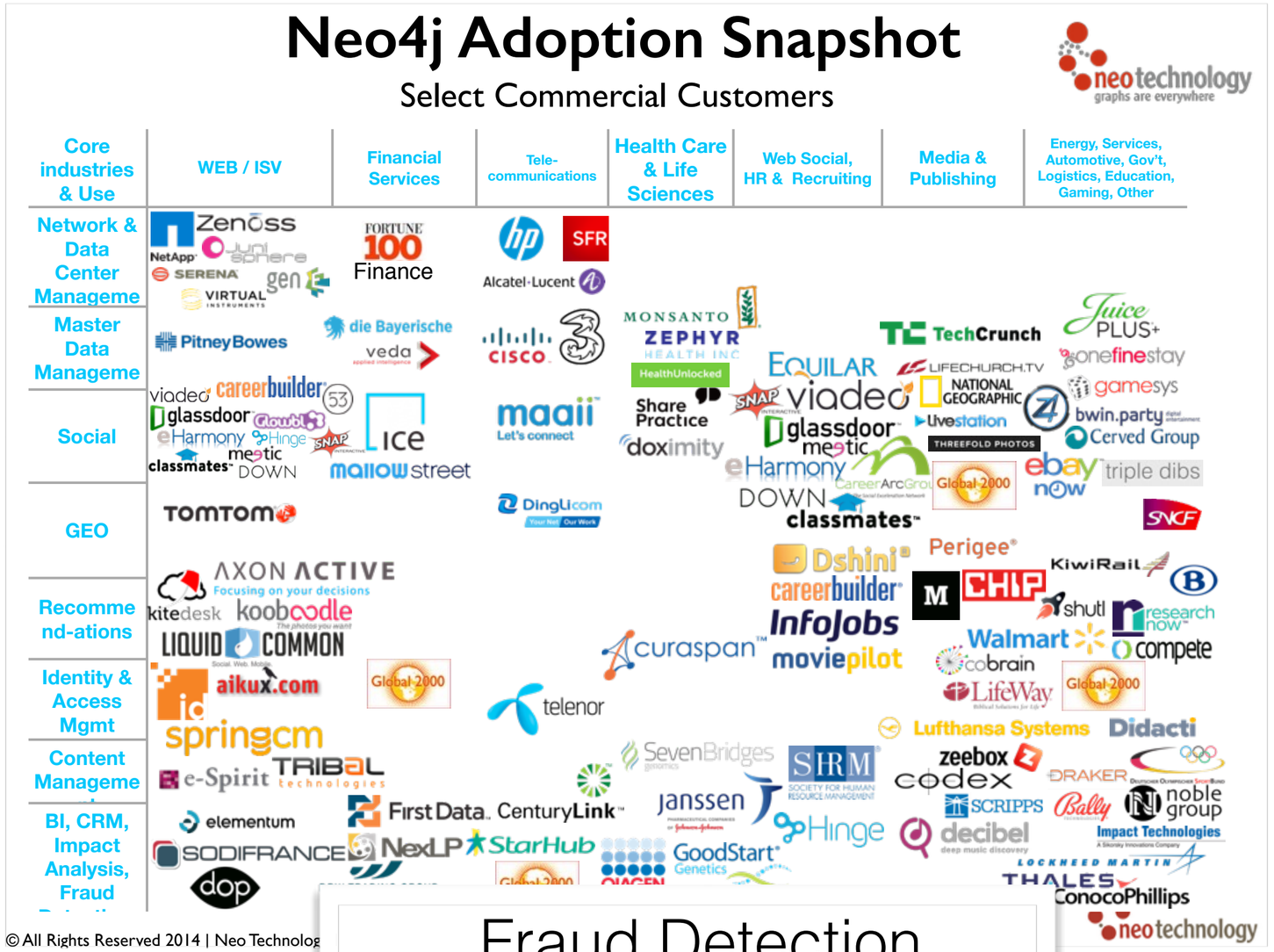
## Cypher: Basic Example

- Declarative query langue with SQL-like clause syntax
- Visual graph patterns
- Tabular results

```
// get node
MATCH (a:Person {id: 0}) RETURN a

// return friends
MATCH (a:Person {id: 0})-->(b) RETURN b

// return friends of friends
MATCH (a:Person {id: 0})--()--(c) RETURN c
```




## Fraud Detection

```
MATCH       (accountHolder:AccountHolder)-[]->(contactInformation)
WITH        contactInformation,
            count(accountHolder) AS RingSize
MATCH       (contactInformation)<-[]-(accountHolder),
            (accountHolder)-[r:HAS_CREDITCARD|HAS_UNSECUREDLOAN]->(unsecuredAccount)
WITH        collect(DISTINCT accountHolder.UniqueId) AS AccountHolders,
            contactInformation, RingSize,
            SUM(CASE type(r)
                WHEN 'HAS_CREDITCARD' THEN unsecuredAccount.Limit
                WHEN 'HAS_UNSECUREDLOAN' THEN unsecuredAccount.Balance
                ELSE 0
            END) as FinancialRisk
WHERE       RingSize > 1
RETURN      AccountHolders AS FraudRing,
            labels(contactInformation) AS ContactType,
            RingSize,
            round(FinancialRisk) as FinancialRisk
ORDER BY    FinancialRisk DESC
```

# Research Directions: Broader Graph Types

- Bipartite graphs (SALSA, matching, link prediction, personalized PageRank)
- Asynchronous frameworks
- Mutable graphs
- Graphs that change as a result of the computation (Borůvka minimum-spanning-tree, Delaunay triangulation)
- Graphs that require modifying the graph to compute (Karger's mincut)
- In general, **significant data structure challenges**.
  - Is CSR the right format?

# Research Directions: Scalability

- Intel's recent CPUs: 1.5 TB
- Biggest NVIDIA GPU: 12 GB
- Directions:
  - Multi-GPU, single node
  - Out of core?
  - Multi-node?

- Long term: heterogeneous single-chip processors

# N-Body

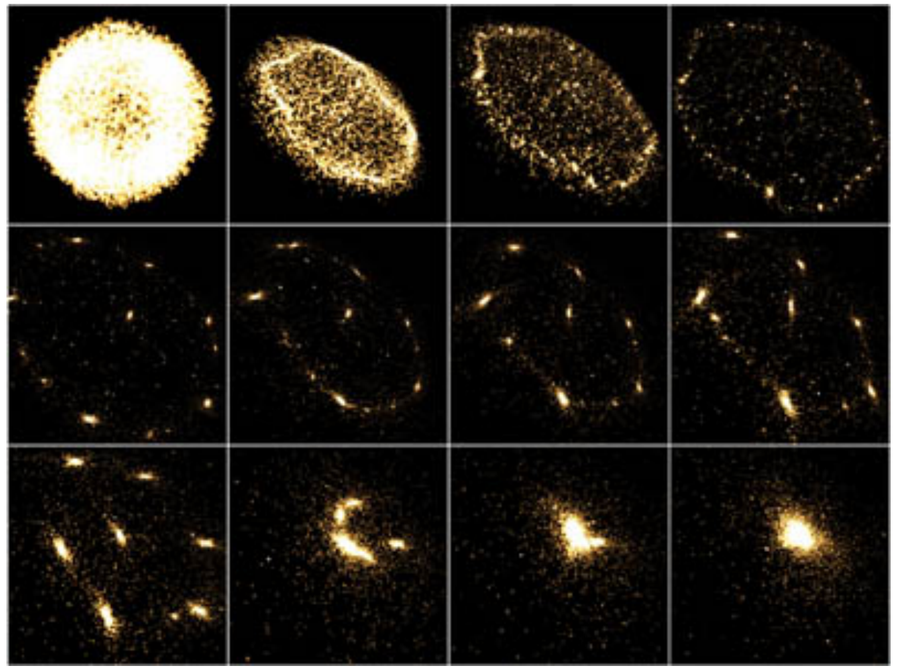
Kerry Seitz originally developed this lecture.

# Credits

- L. Nyland, M. Harris, and J. Prins, “Fast N-Body Simulations with CUDA,” GPU Gems 3, Chapter 31, Jul. 2007.
  - http://http.developer.nvidia.com/GPUGems3/gpugems3_ch31.html
- J. Bédorf, E. Gaburov, and S. P. Zwart, “A sparse octree gravitational N-body code that runs entirely on the GPU processor,” Journal of Computational Physics, vol. 231, no. 7, pp. 2825–2839, Apr. 2012.
- Original lecture by Kerry Seitz

# N-Body Simulations

- Approximates system where every body interacts with every other body



- Examples
  - Astrophysical simulations of galaxies and stars (gravitational force)
  - Protein folding (electrostatic and van der Waals forces)
  - Turbulent flow
  - Global Illumination (e.g., in ray tracing)

# Gravitational Simulation

- Want to see how the stars move over time due to the gravitational force exerted by all other stars

- Want to simulate N stars

- https://upload.wikimedia.org/wikipedia/commons/2/25/Galaxy_collision.ogv

# Gravitational Simulation

$$f_{ij} = G\frac{m_i m_j}{||\vec{r}_{ij}||^2} \cdot \frac{\vec{r}_{ij}}{||\vec{r}_{ij}||}$$

$$F_i = \sum f_{ij} = Gm_i \cdot \sum \frac{m_j \vec{r}_{ij}}{||\vec{r}_{ij}||^3}$$

$$F_i \approx Gm_i \cdot \sum \frac{m_i \vec{r}_{ij}}{(||\vec{r}_{ij}||^2 + \varepsilon^2)^{3/2}}$$

$$a_i \approx G \cdot \sum \frac{m_i \vec{r}_{ij}}{(||\vec{r}_{ij}||^2 + \varepsilon^2)^{3/2}}$$

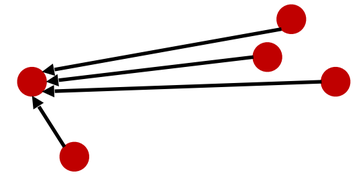
$f_{ij}$ = force on body i by body j
$F_i$ = total force on body i
$a_i$ = acceleration of body i ($a_i = F_i / m_i$)

$G$ = gravitational constant
$m_i$ = mass of object i
$\vec{r}_{ij}$ = distance vector from body i to body j
$\varepsilon$ = softening factor ($\varepsilon^2 > 0$)

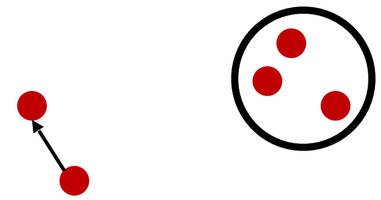
Prevents collisions
(reasonable if bodies represent galaxies that pass through each other)

# N-Body Methods

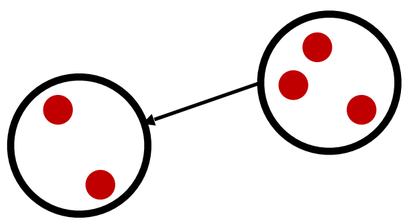
- All-pairs: $O(n^2)$
  - Full body-body calculation on all N items pairwise



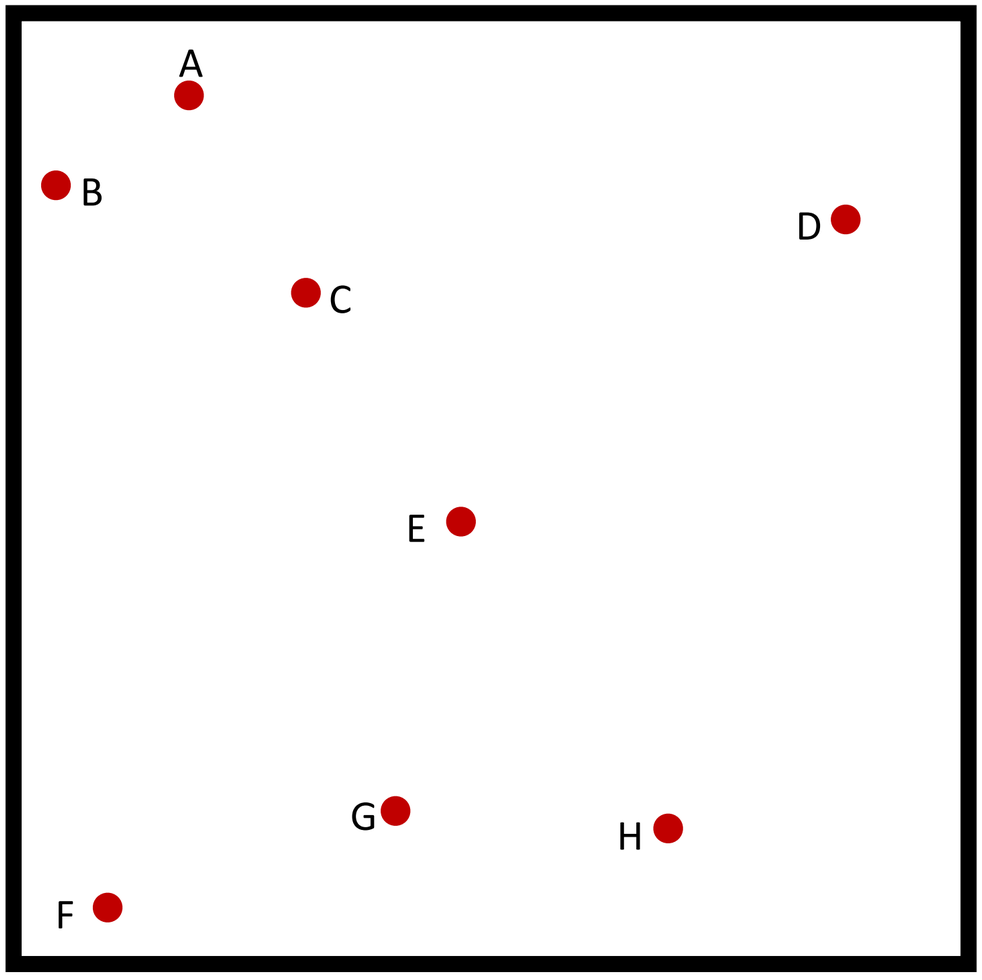
- Barnes-Hut: O(n log n)
  - Divide volume into cubic cells
  - Combine bodies in far away cells as a single cluster
  - Perform body-body calculations on bodies in nearby cells
  - Perform body-cluster calculations for far away interactions



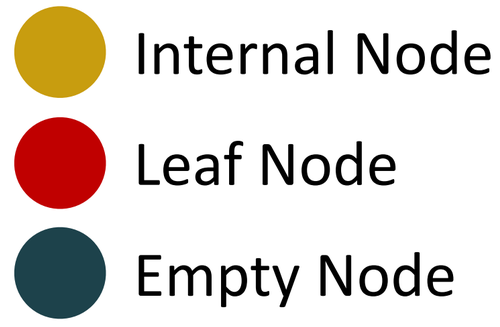
- Fast Multipole: O(n)
  - Similar to Barnes-Hut
  - Perform cluster-cluster calculations for far away interactions
  - Derive body-cluster interactions from cluster-cluster interactions
  - Provable error bounds



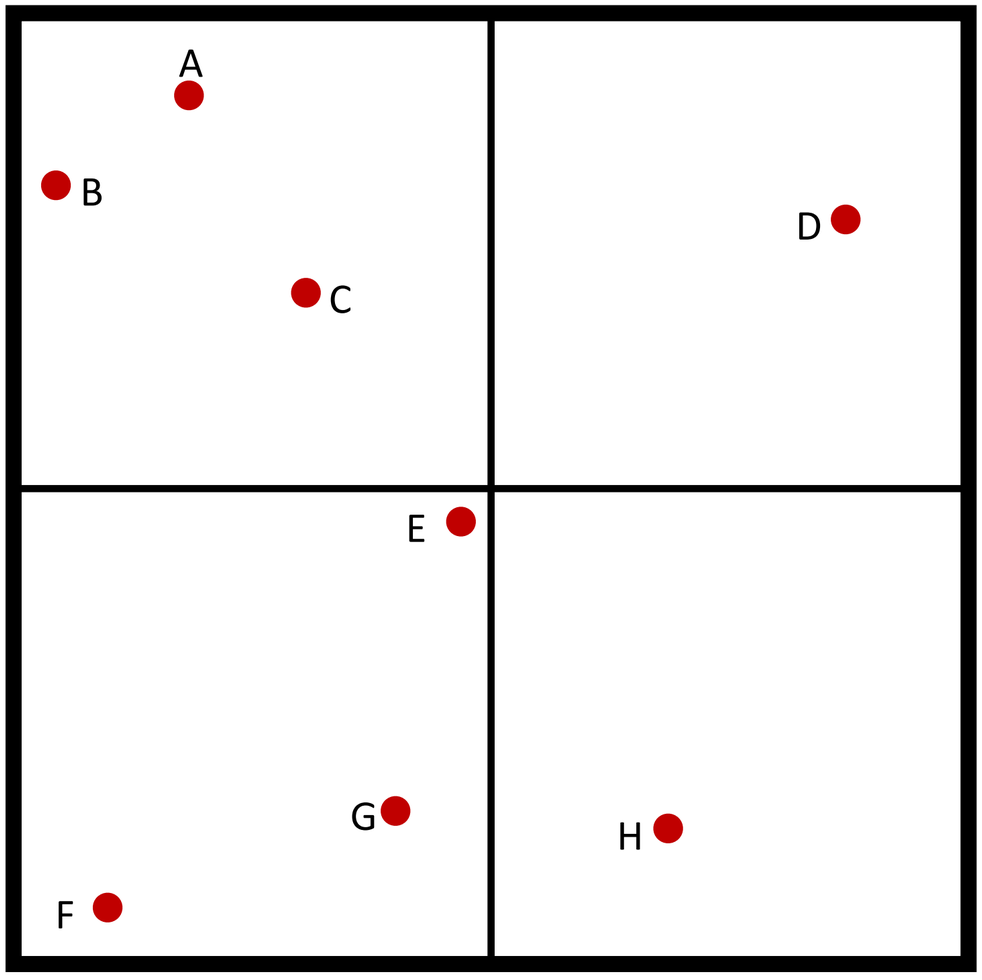
# Quadtree



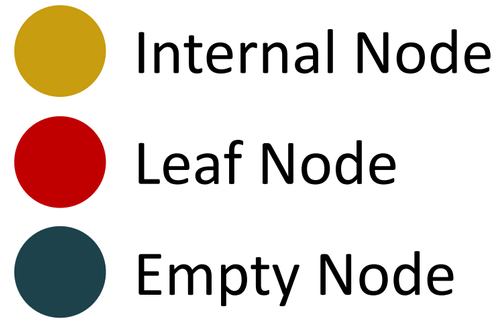




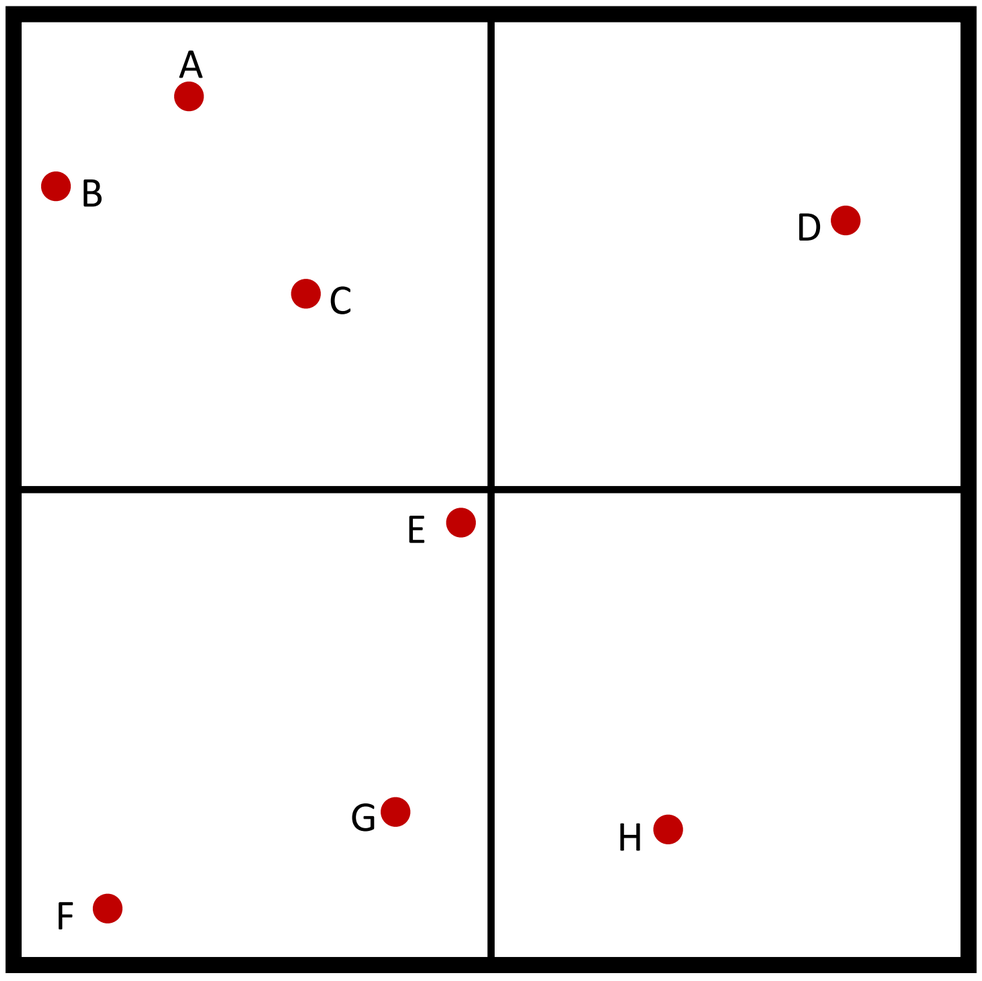
# Quadtree

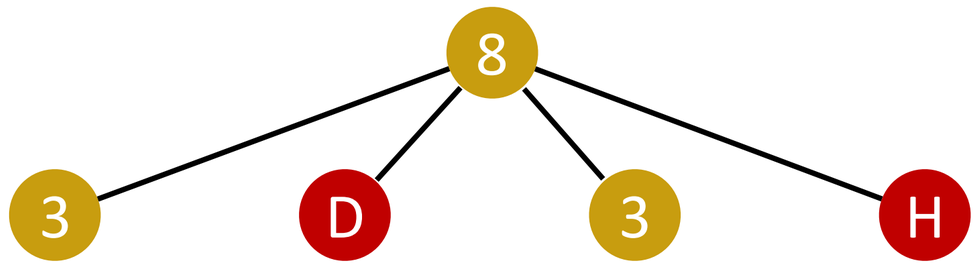


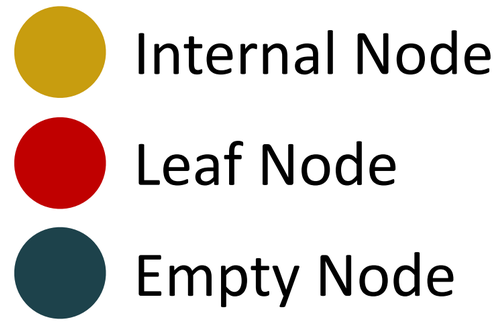




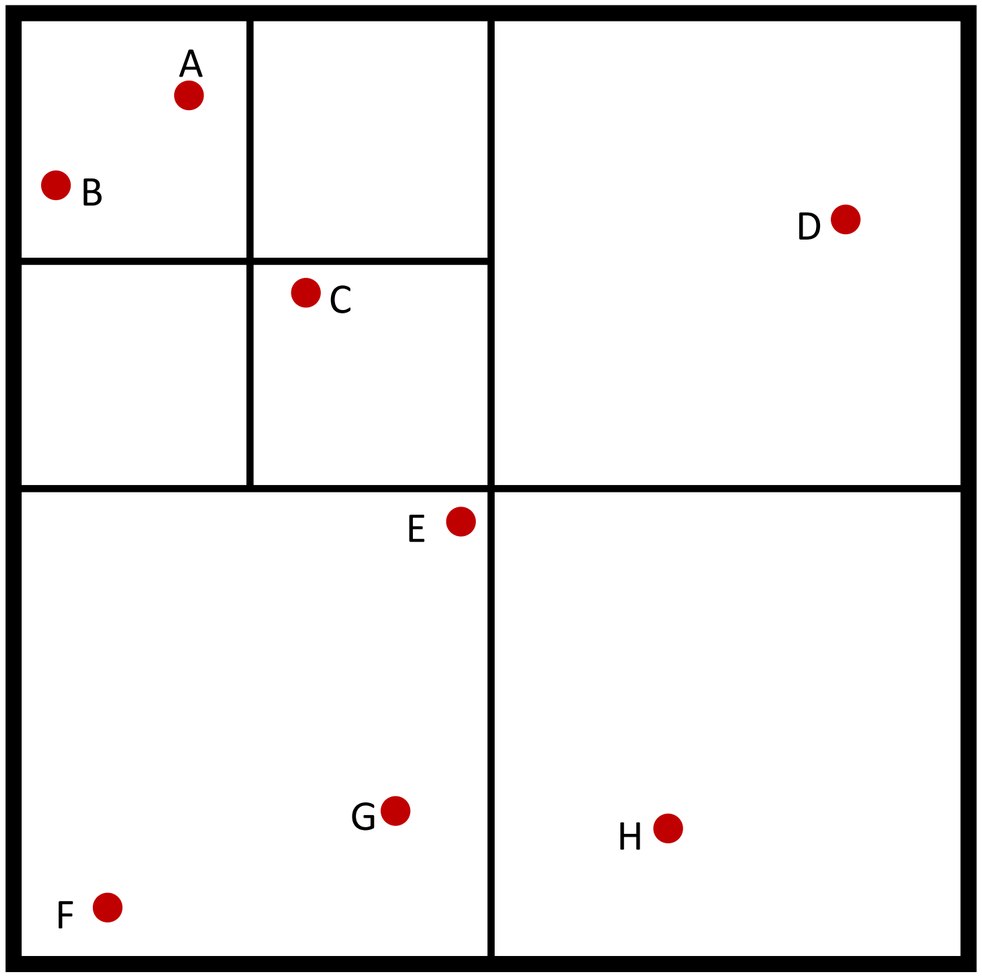
# Quadtree

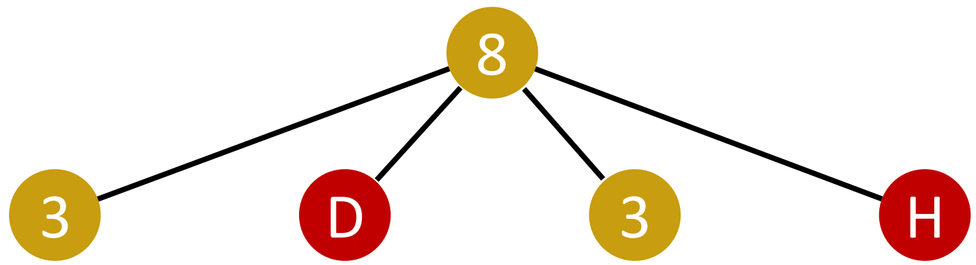


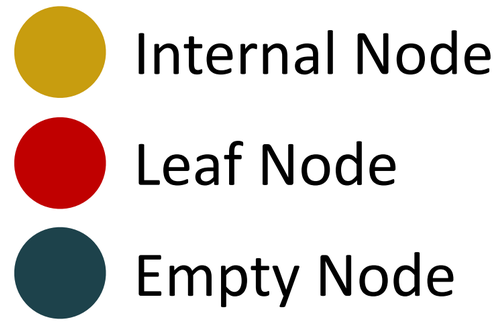



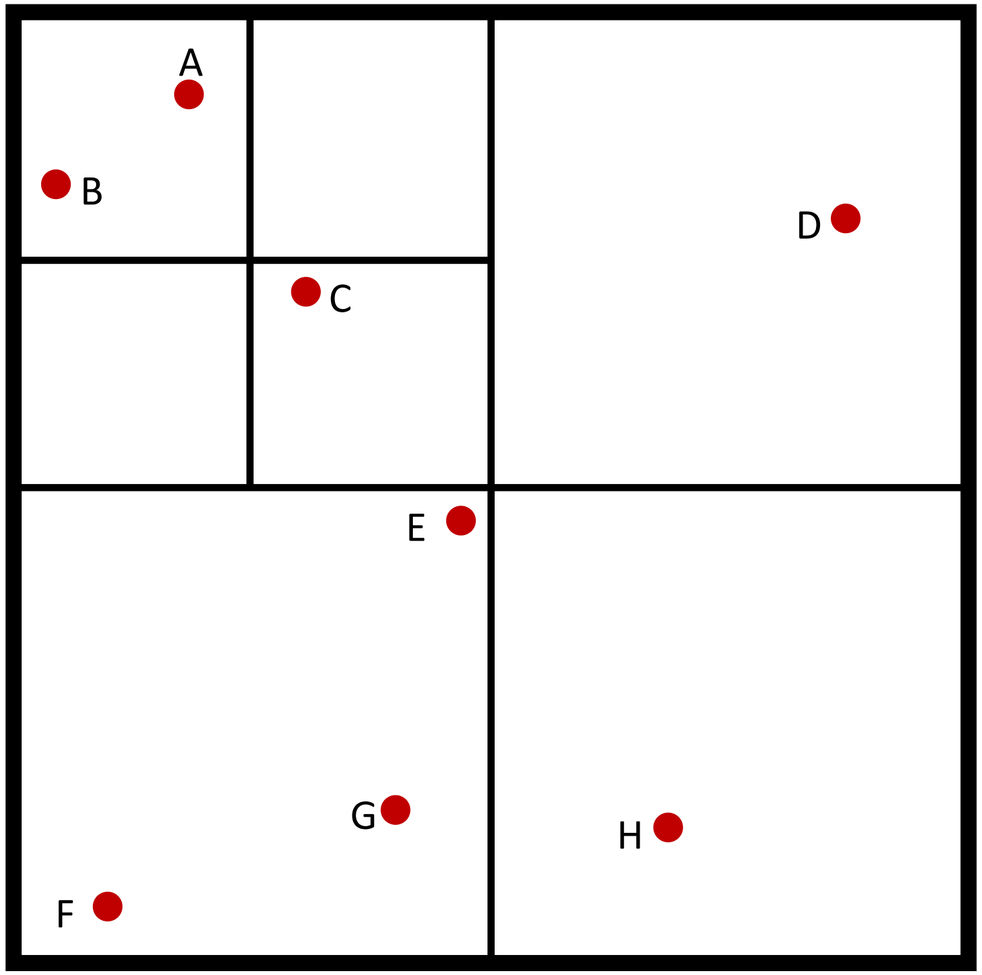
# Quadtree

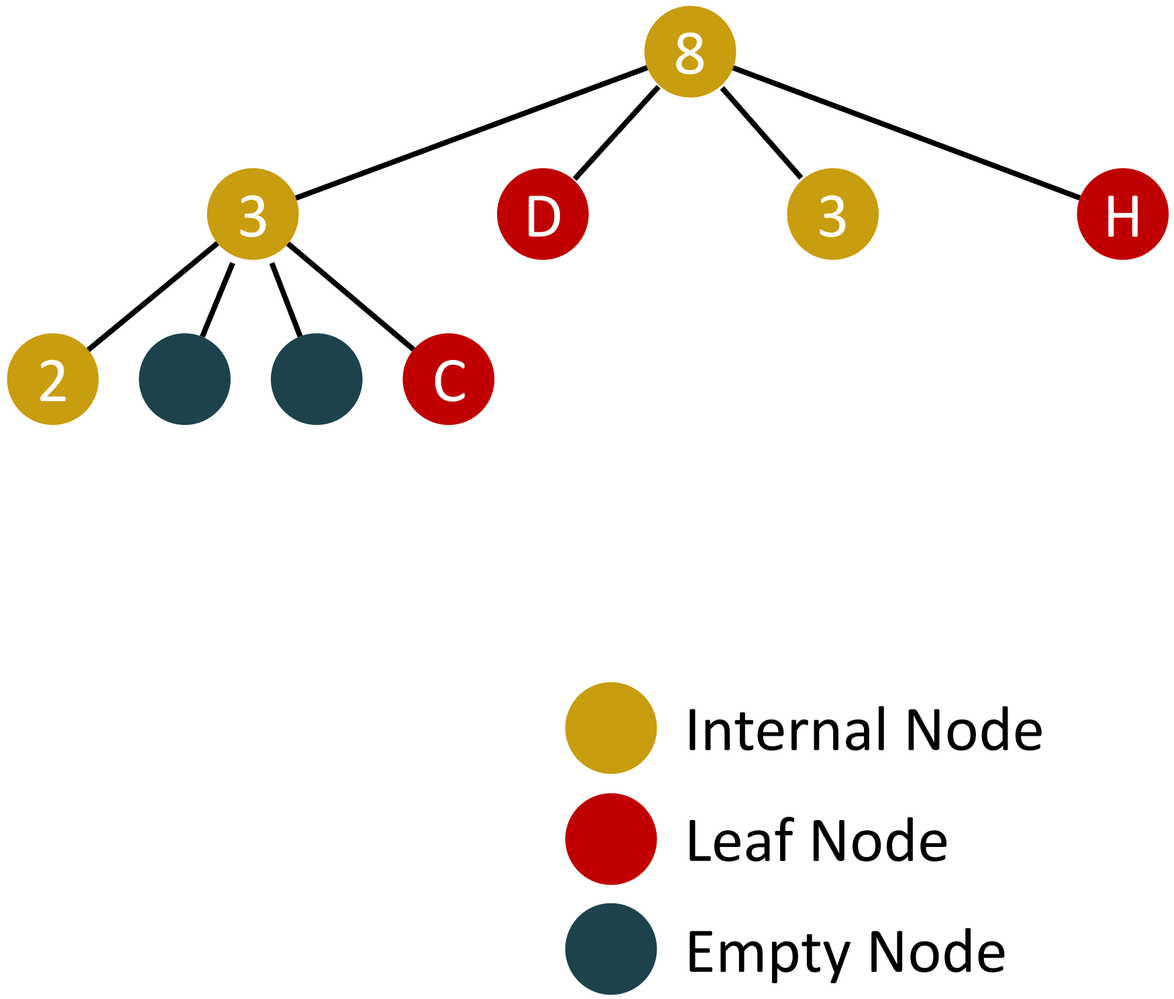


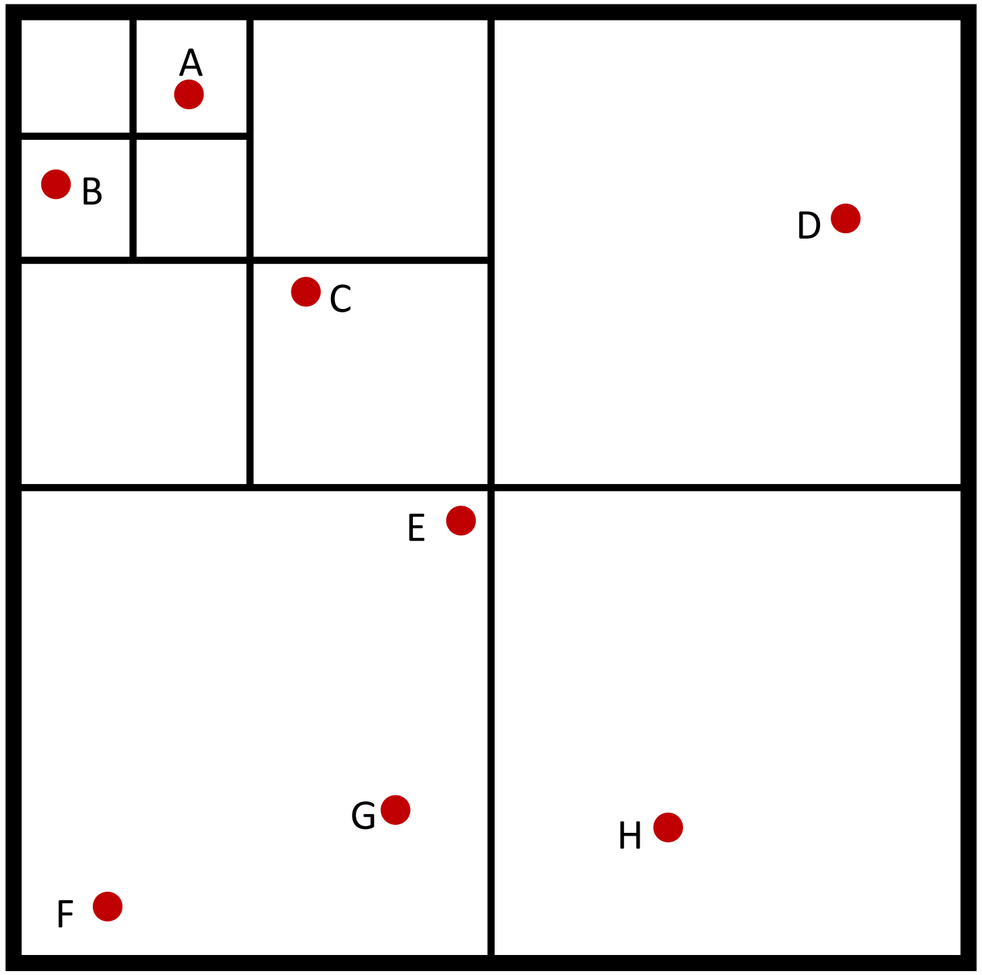



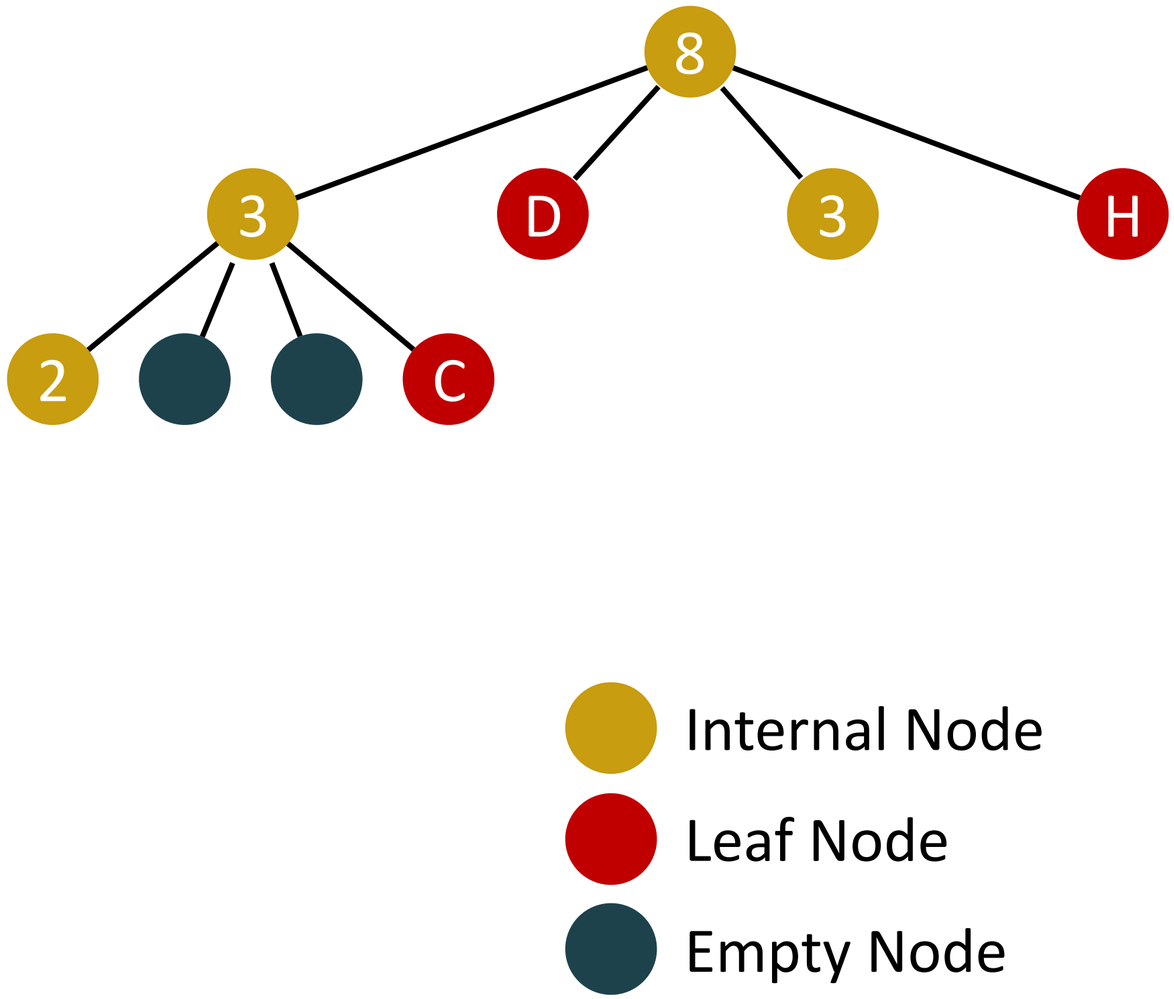
# Quadtree



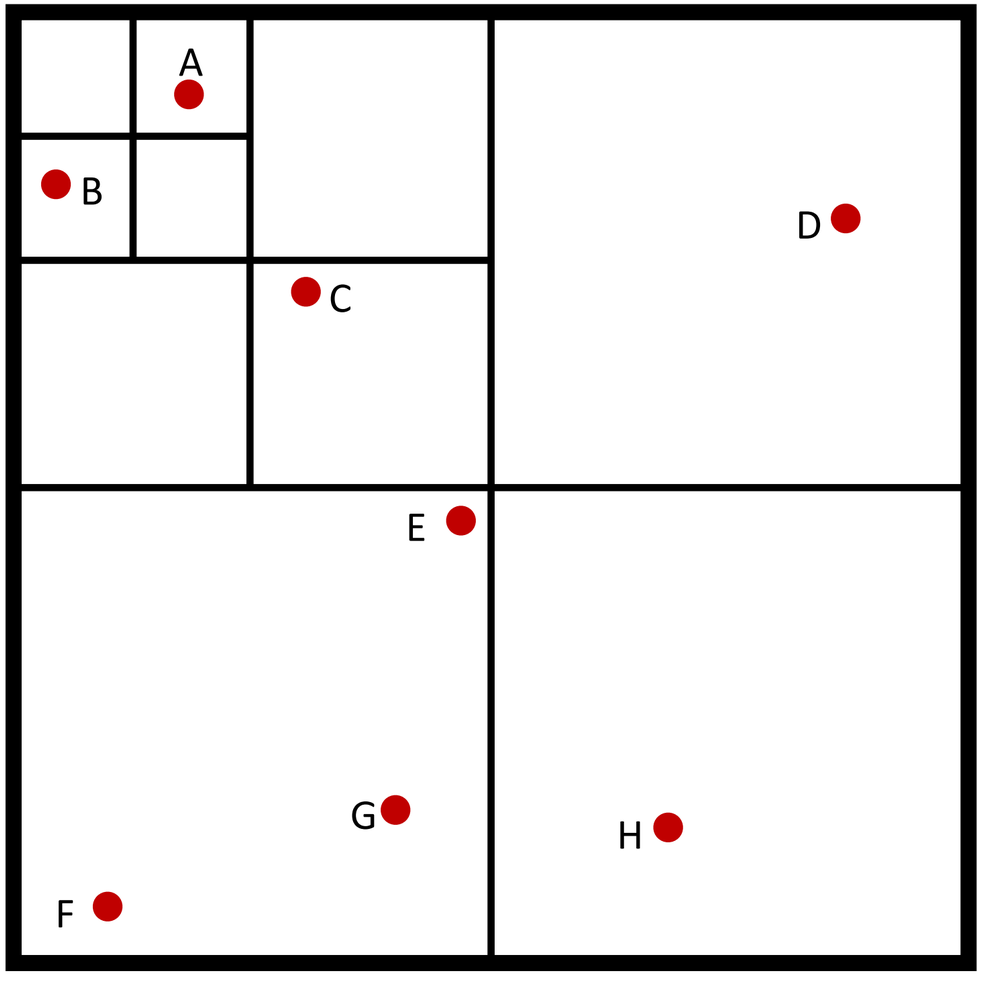



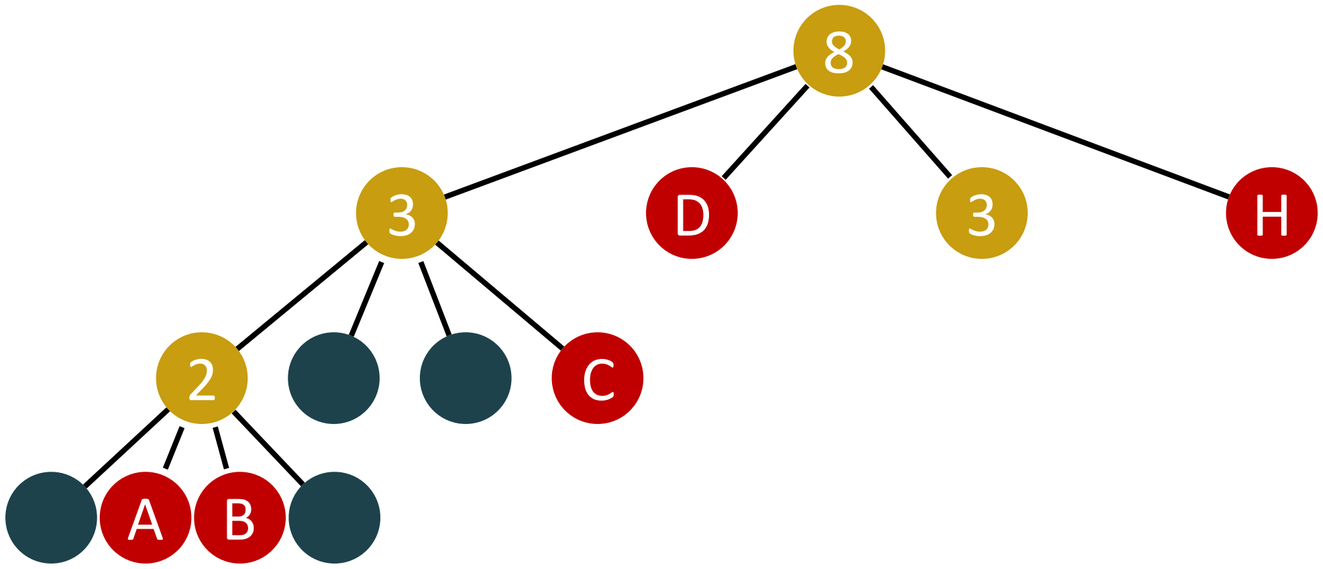
# Quadtree



# Quadtree



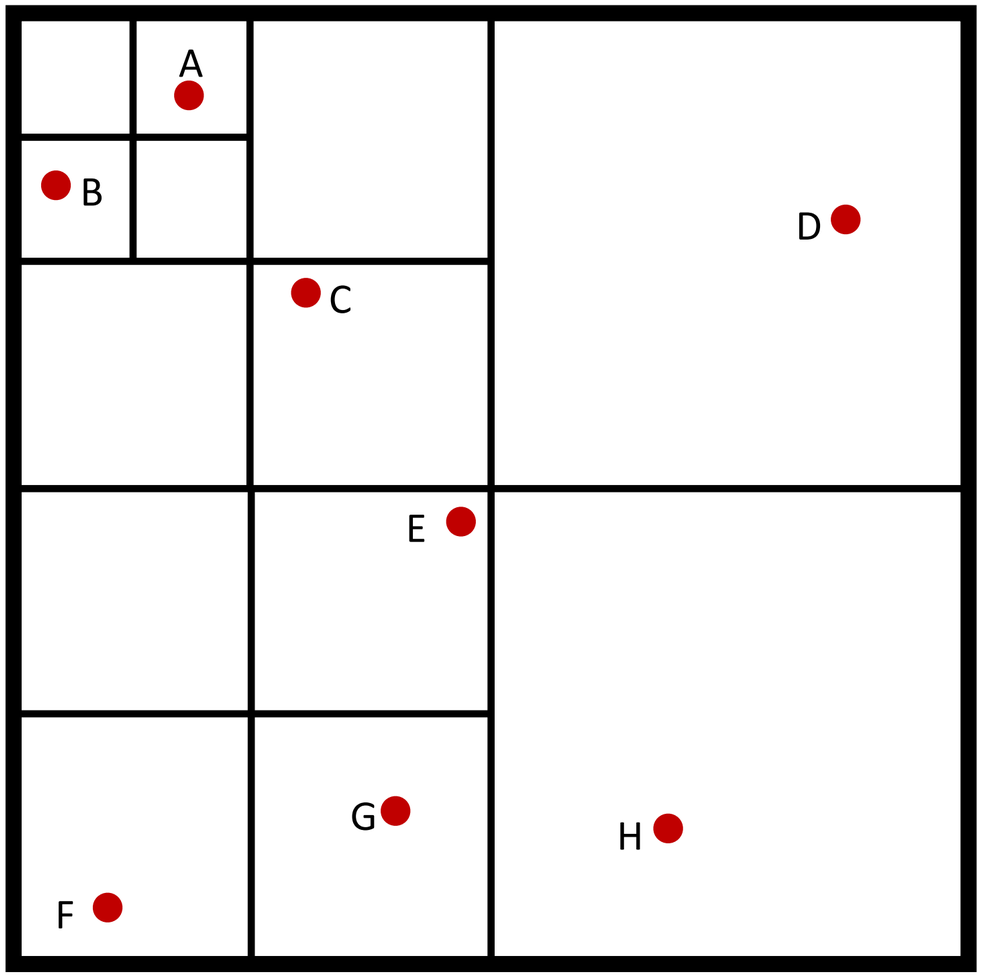



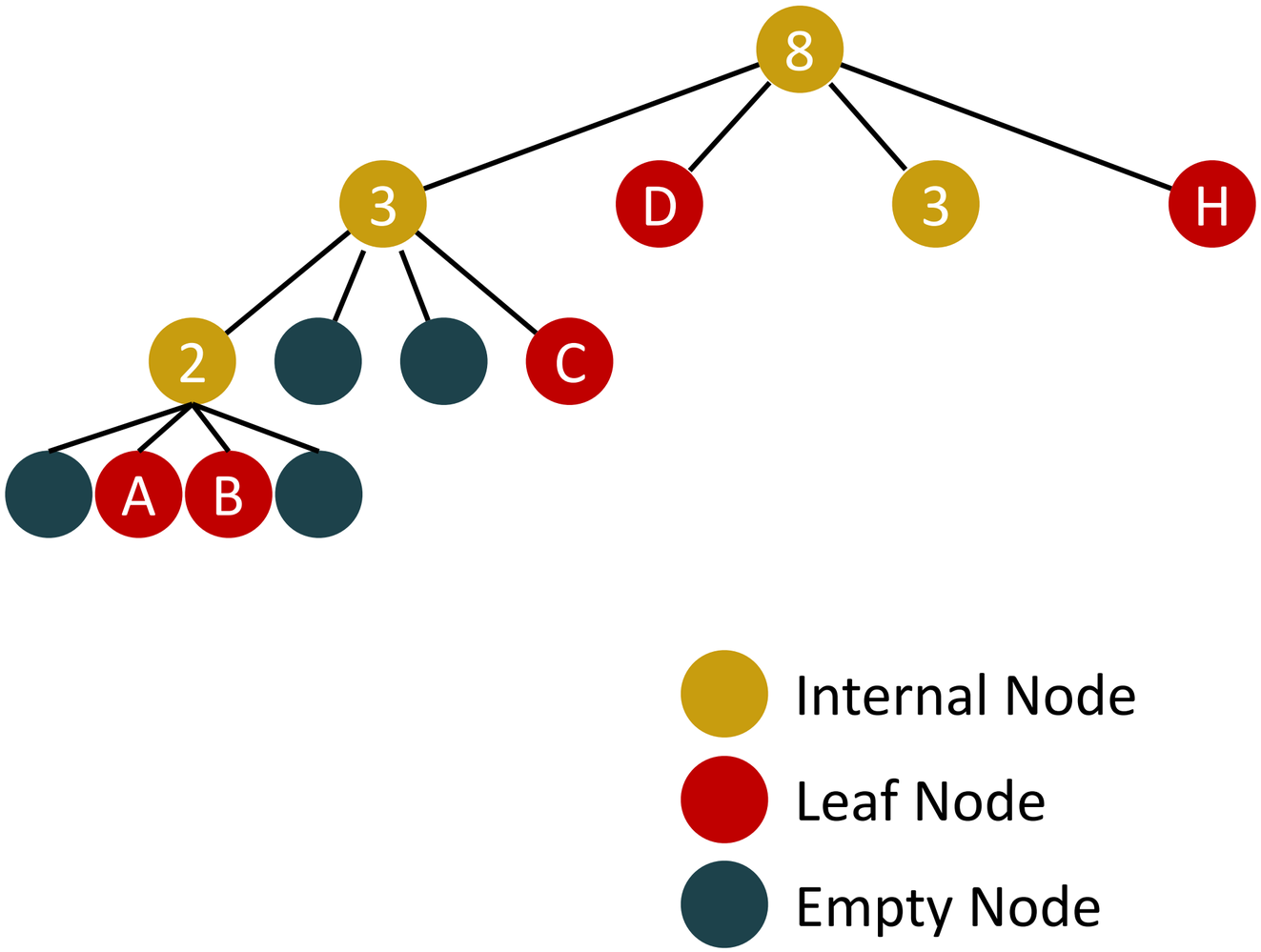

Internal Node

Leaf Node

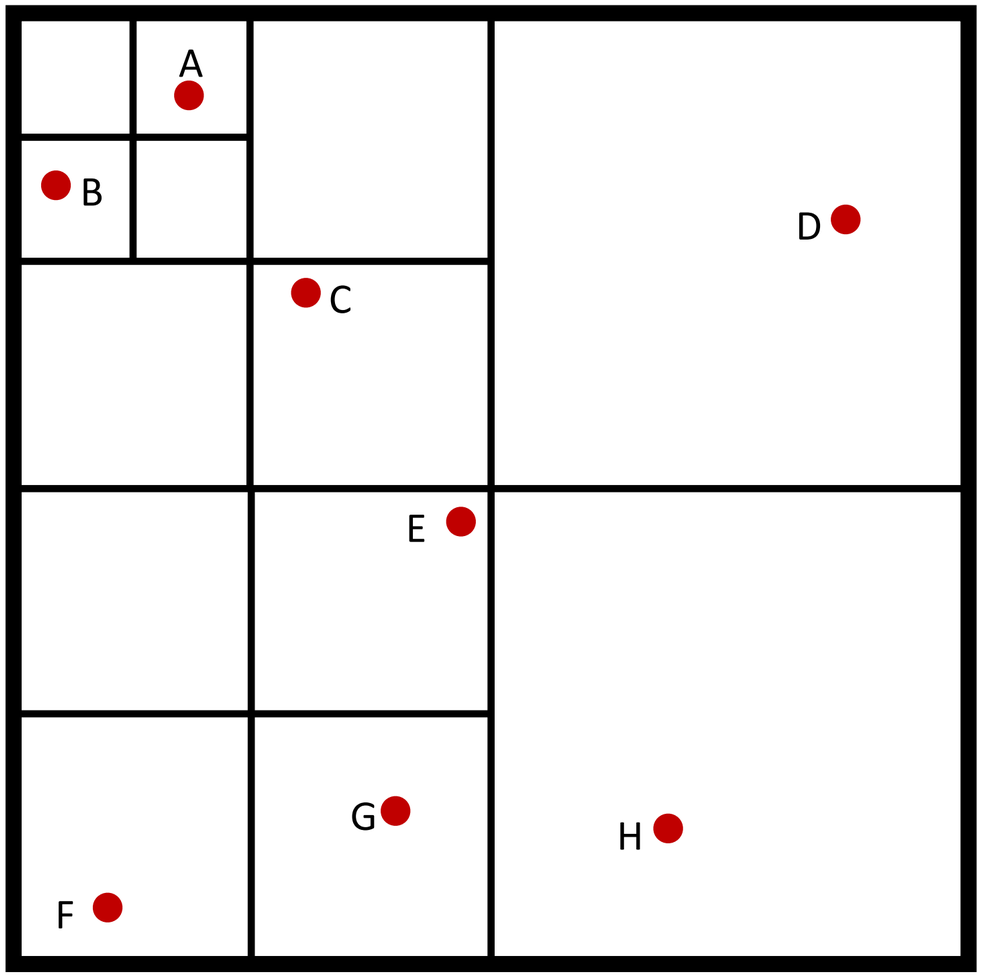
Empty Node

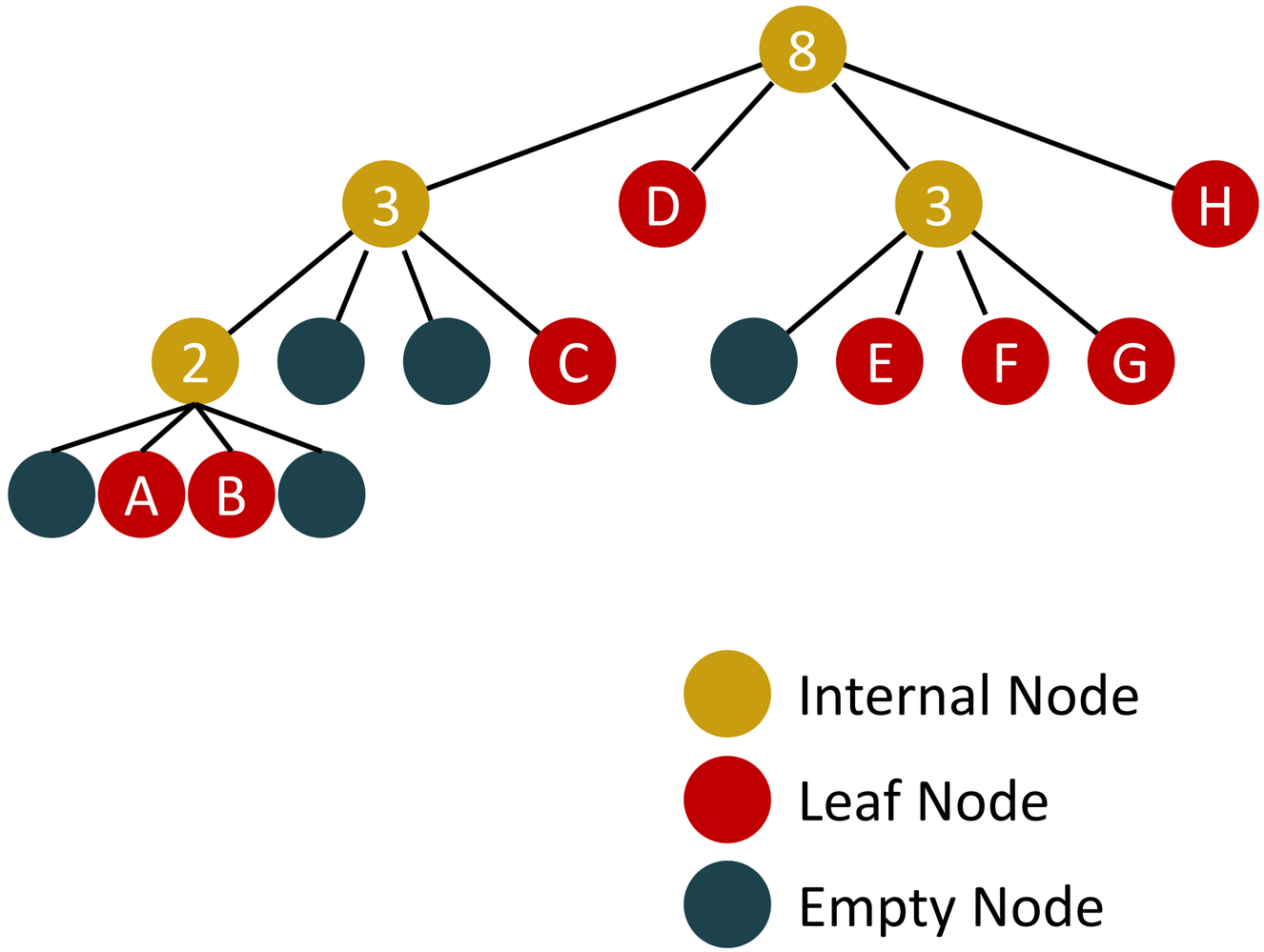
# Quadtree



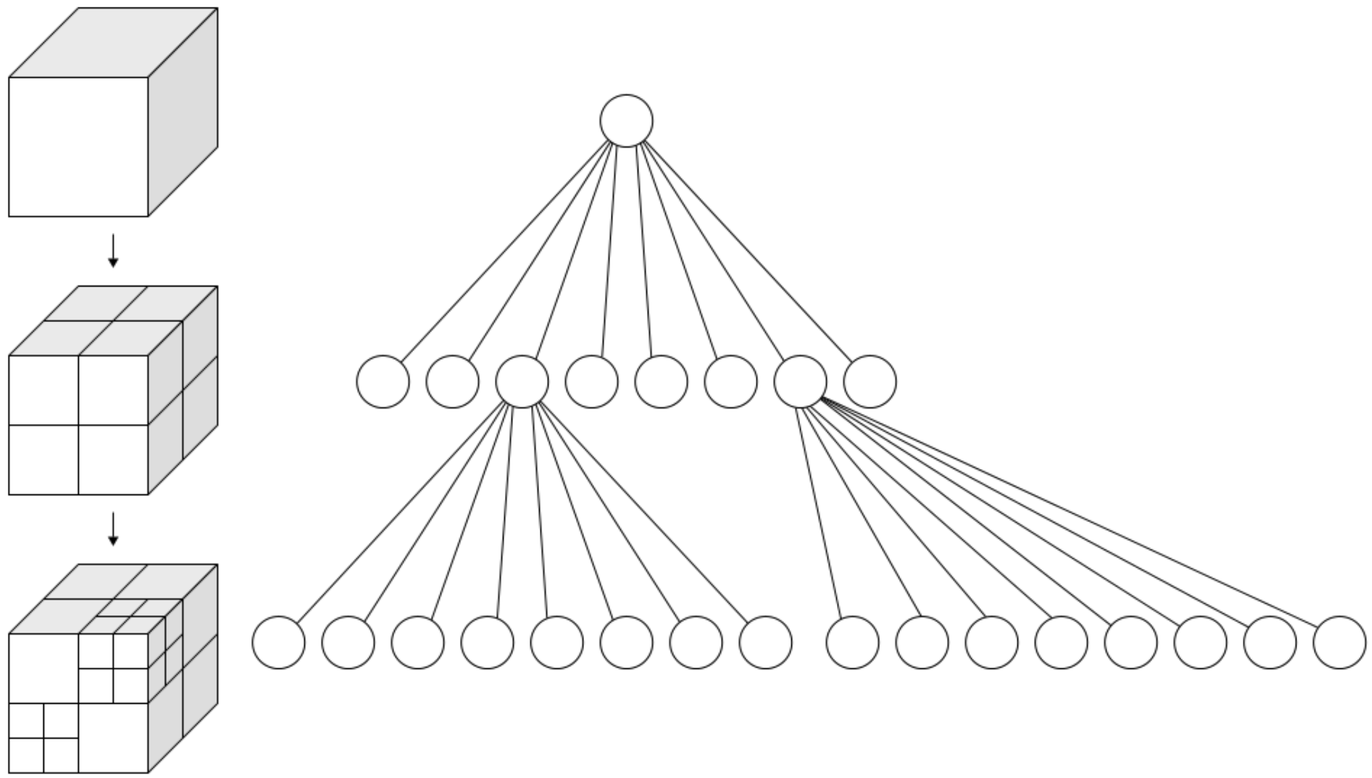



# Quadtree



# Octree – 3D Quad Tree



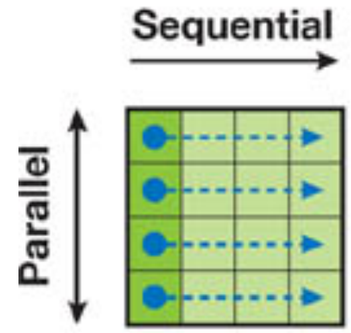
https://en.wikipedia.org/wiki/Octree

# All-pairs Method

- Used in the other methods as the innermost kernel
- Speedup in all-pairs means speedup overall
- Better all-pairs performance means we can use all-pairs for more of the interactions, leading to more accurate results

# All-pairs Implementation

- Compute each $f_{ij}$ in an *N*x*N* grid



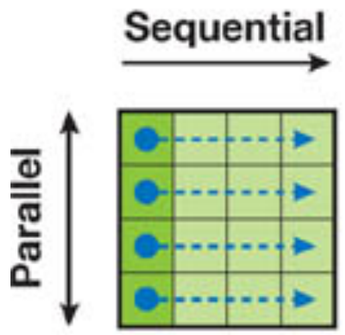

- Total for $F_i$ (or acceleration $a_i$) is sum of row *i*

- Everything is independent – lots of parallelism

- Bandwidth limited (low arithmetic intensity)

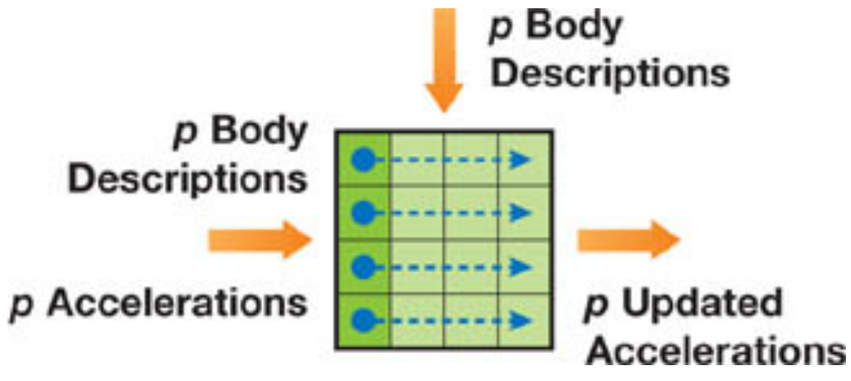
# All-pairs Implementation

- Arrange computation into 2D tiles of size *p* by *p*



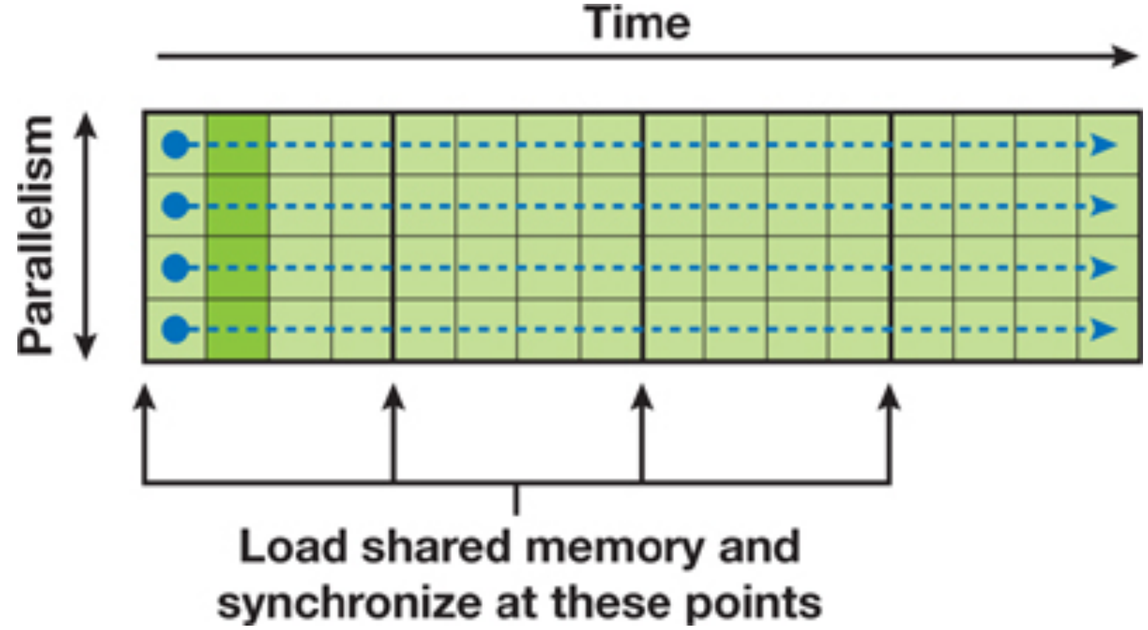

- Parallelize over rows

- Sequential within each row

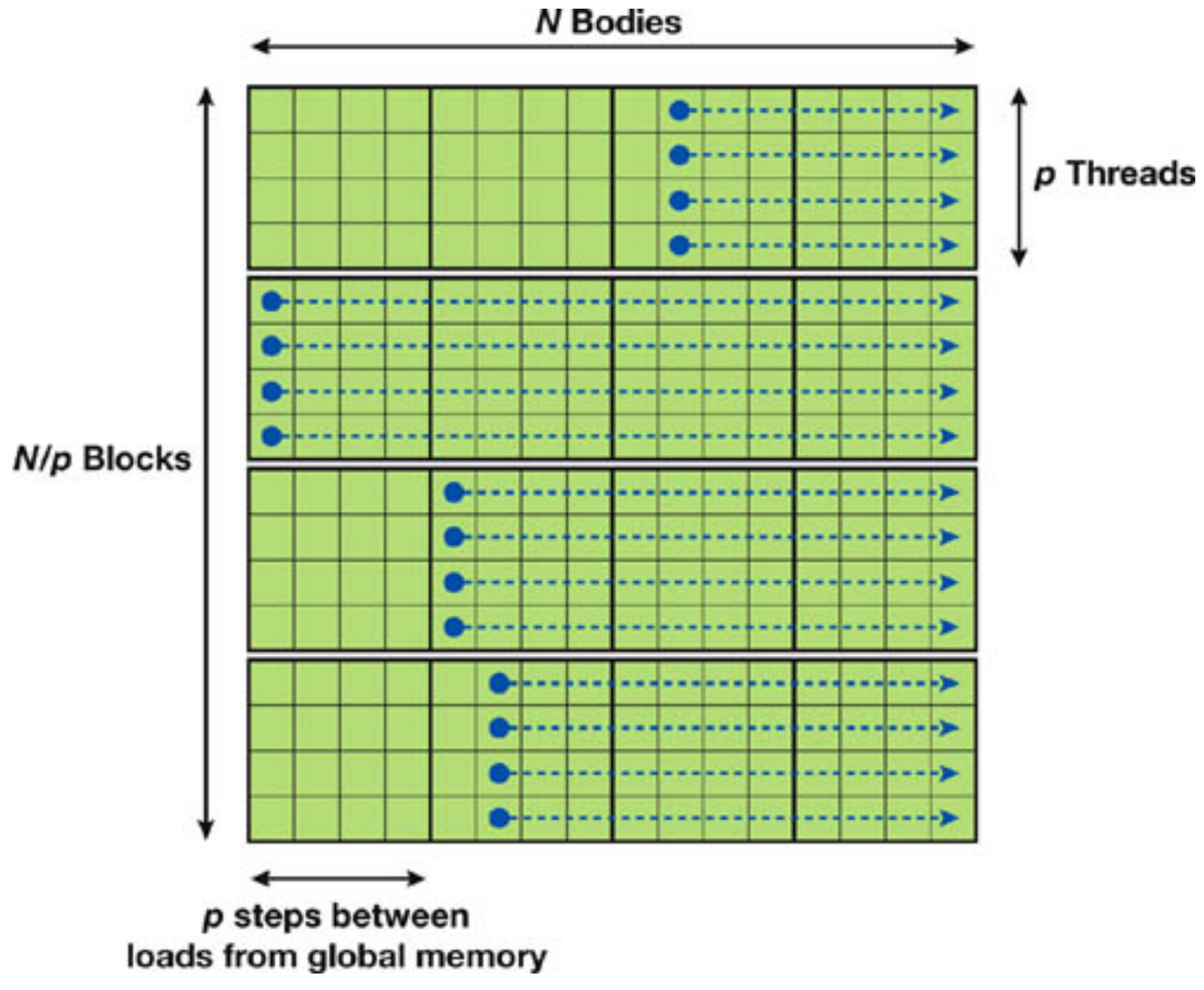


# All-pairs Algorithm

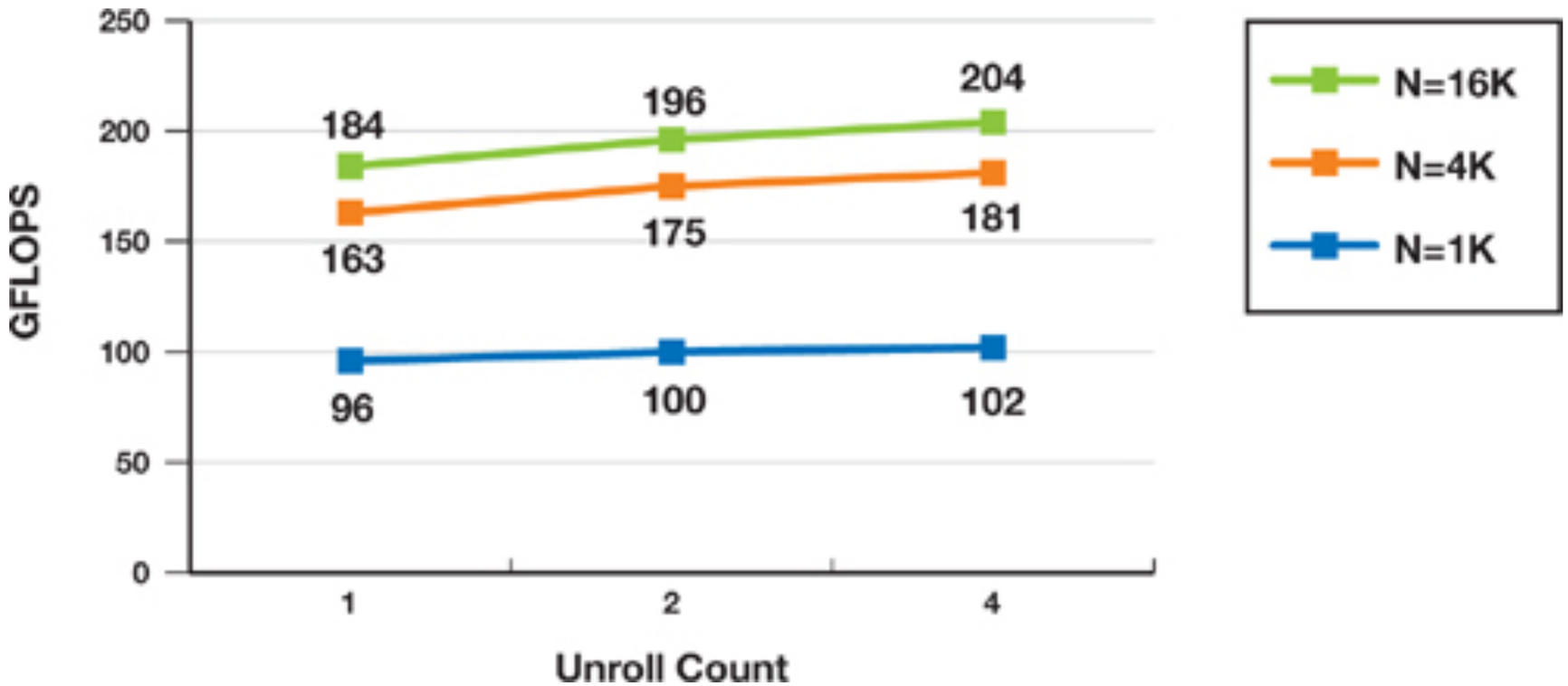
```
__global__ void calculate_forces(float4* positions, float3* accels) {
   extern __shared__ float4[] sPos;
   int gtid = blockIdx.x * blockDim.x + threadIdx.x;
   float4 myPos = positions[gtid];
   for(int i = 0, int tile = 0; i < N; i += p, tile++) {
      int idx = tile * blockDim.x + threadIdx.x;
      sPos[threadIdx.x] = positions[idx];
      __syncthreads();
      // calculate acceleration for myPos
      __syncthreads()
   }
   // write resulting acceleration to global memory
}
```



# All-pairs Algorithm

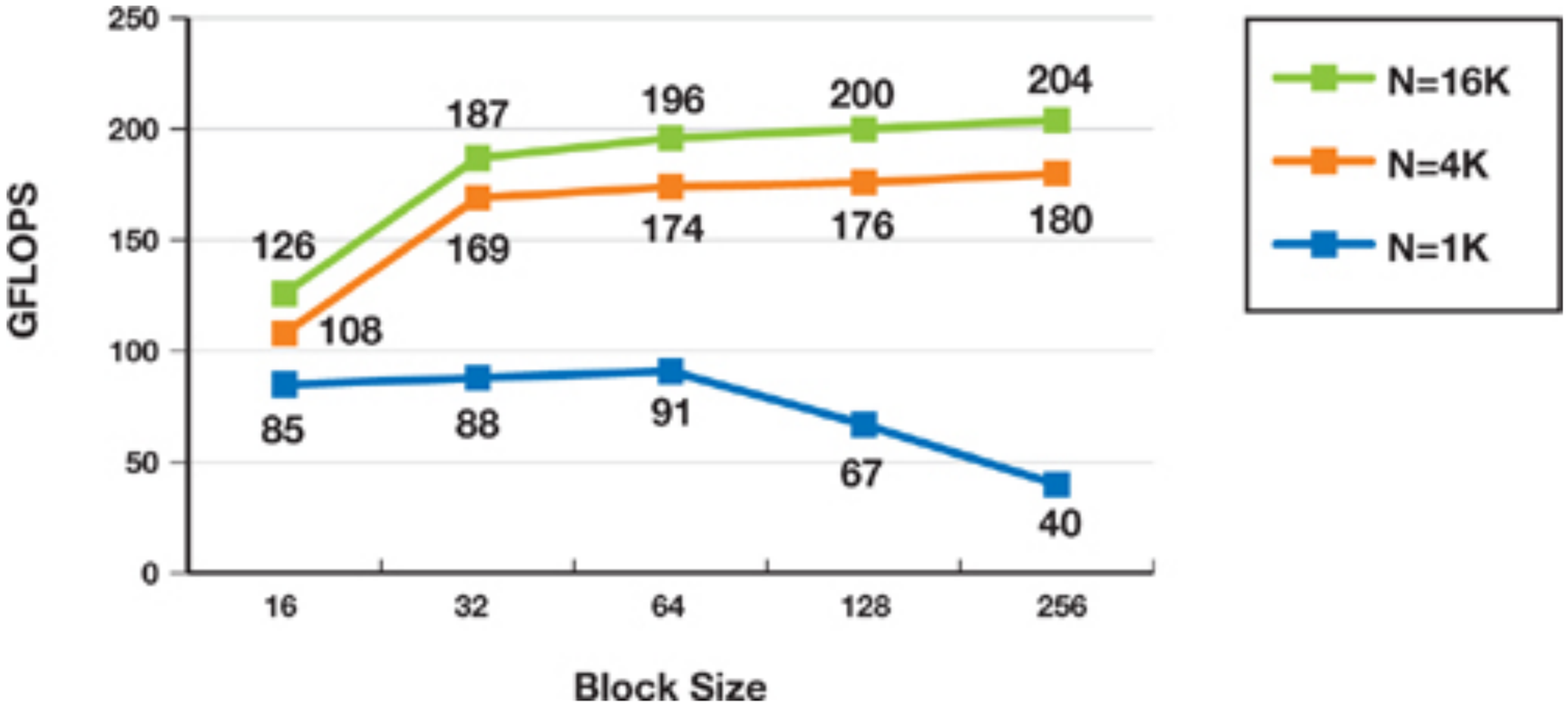


# Optimization – Loop Unrolling



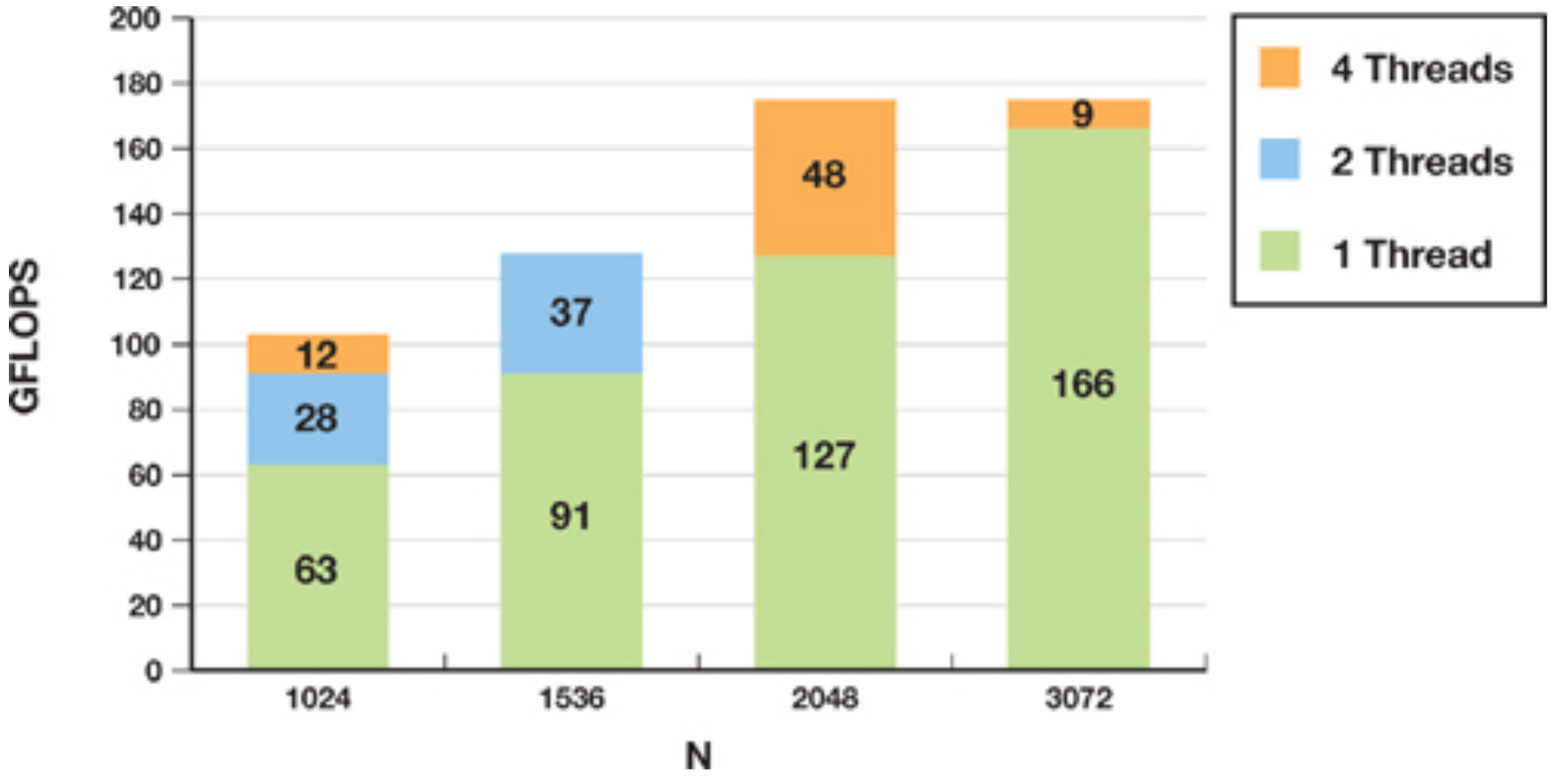

Performance measured using a NVIDIA GeForce GTX 8800

# Optimization – Block Size




Performance measured using a NVIDIA GeForce GTX 8800

# Optimization – Splitting the Rows




Performance measured using a NVIDIA GeForce GTX 8800

# Barnes-Hut Method (entirely on GPU)

- Build (sparse) octree

- Compute cluster data (at internal nodes) using bottom-up traversal
  - Parallel at each level

- Traverse octree breadth-first (for efficient GPU implementation)
  - Build up list of body-body and body-cluster interactions

- Evaluate body-body and body-cluster interactions to get the new accelerations for each body

- Predict velocities and positions using accelerations; correct velocities and positions

J. Bédorf, E. Gaburov, and S. P. Zwart, "A sparse octree gravitational N-body code that runs entirely on the GPU processor," Journal of Computational Physics, vol. 231, no. 7, pp. 2825–2839, Apr. 2012.

# Barnes-Hut Method
# Accuracy and Performance

- Both depend on a control parameter θ
  - Controls distance required to switch to body-cluster approximation
- ↓ θ means
  - Deeper tree traversal
  - Better accuracy
  - Worse performance
- ↑ θ means
  - Shallower tree traversal
  - Worse accuracy
  - Better performance

J. Bédorf, E. Gaburov, and S. P. Zwart, “A sparse octree gravitational N-body code that runs entirely on the GPU processor,” Journal of Computational Physics, vol. 231, no. 7, pp. 2825–2839, Apr. 2012.

# The Fast Fourier Transform


- Thanks to Akira Nukada, Tokyo Institute of Technology



[1] A. Nukada, Y. Maruyama, and S. Matsuoka, “High performance 3-D FFT using multiple CUDA GPUs,” presented at the GPGPU-5: Proceedings of the 5th Annual Workshop on General Purpose Processing with Graphics Processing Units, 2012.

[2] A. Nukada and S. Matsuoka, “Auto-tuning 3-D FFT library for CUDA GPUs,” *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, Nov. 2009.

[3] A. Nukada, Y. Ogata, T. Endo, and S. Matsuoka, “Bandwidth intensive 3-D FFT kernel for GPUs using CUDA,” presented at the High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for, 2008, pp. 1–11.

# FFT (Fast Fourier Transform)

FFT is a fast algorithm to compute DFT (Discrete Fourier Transform).

$$X'(k) = \sum_{j=0}^{N-1} X(j) e^{-2\pi ijk/N}$$

When the input size N can be factorized into M and L, N-point FFT is replaced by L x M-point FFTs, M x L-point FFT, and multiplications by twiddle factors.

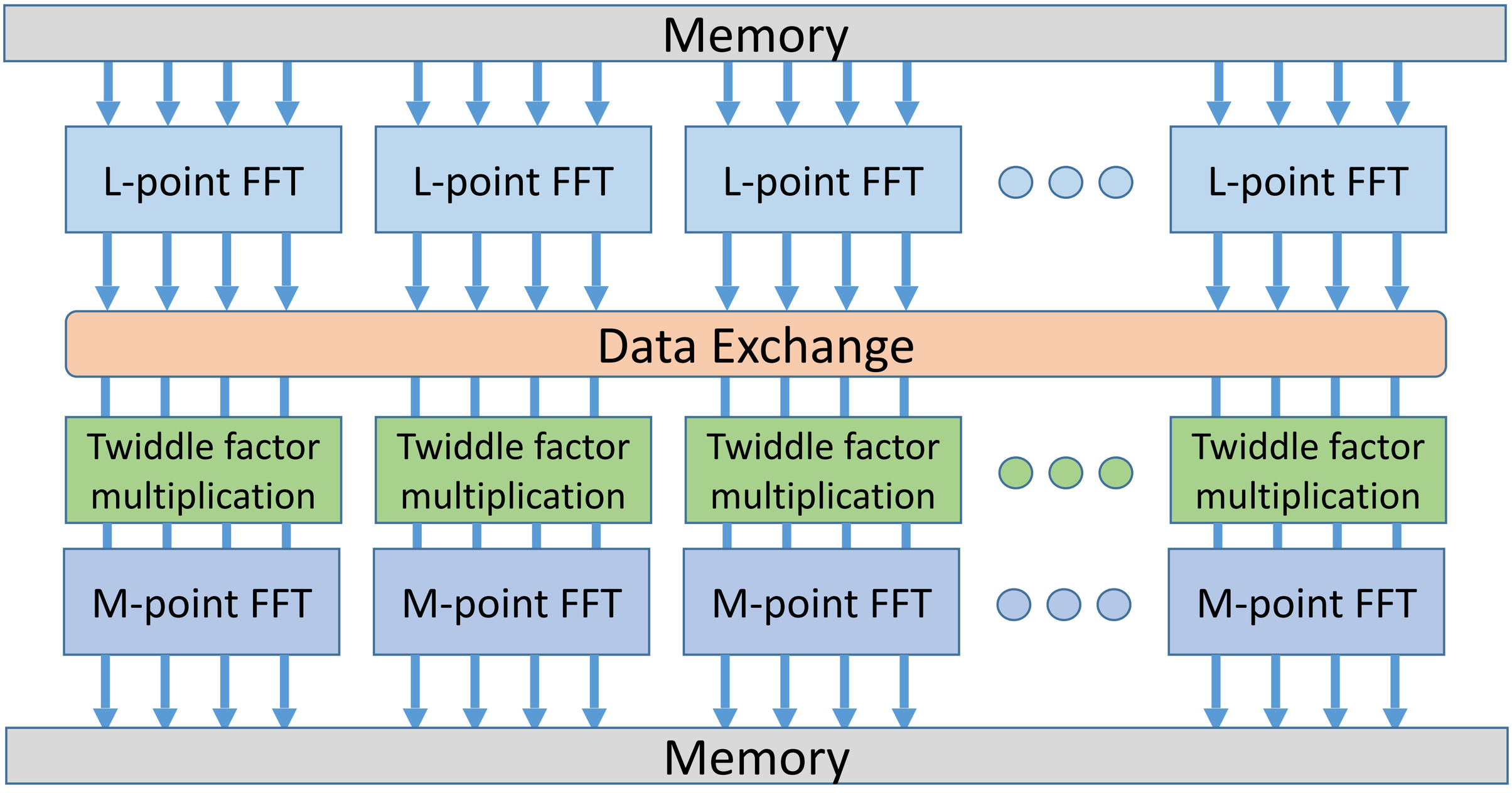
$$X'(k_0 + k_1 L) = \sum_{j_0=0}^{M-1} e^{-2\pi ij_0 k_1/M} \; e^{-2\pi ij_0 k_0/N} \sum_{j_1=0}^{L-1} X(j_0 + j_1 M) e^{-2\pi ij_1 k_0/L}$$

twiddle factors

L-point FFT

M-point FFT

# Fine-grain parallel computation of FFT



# 1-D FFT on CUDA GPUs

Data is read from global memory.


Many threads simultaneously compute small FFTs using registers.

Threads exchange data using shared memory.

Finally, data is written back to global memory.

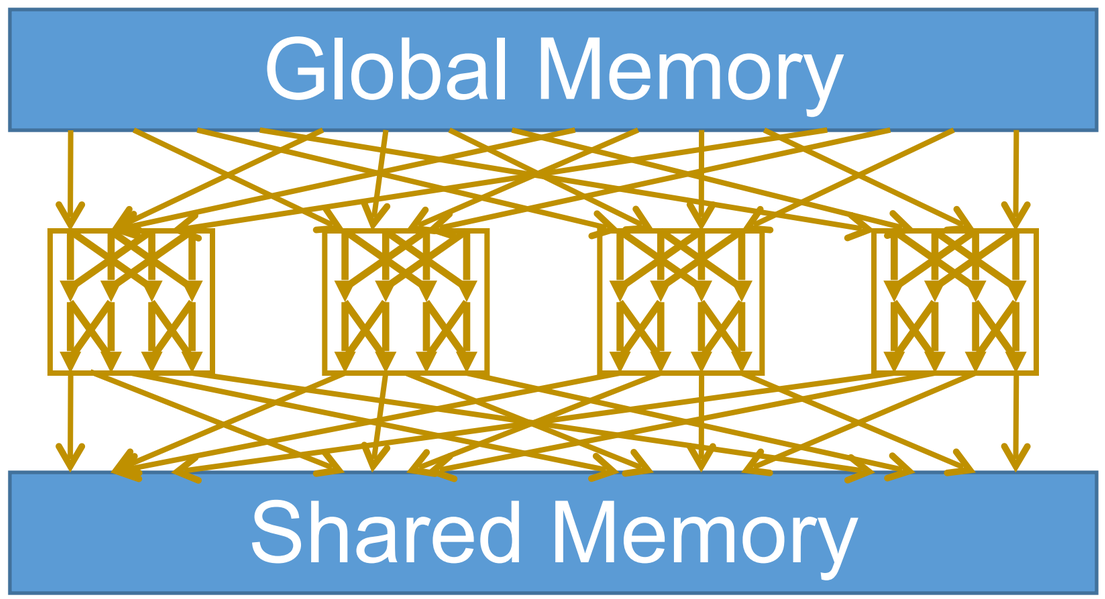
Three CUDA FFTs in 2008 based on this fine-grain parallel implementations

(1) Ours (2) N. Govindaraju, et al. (3) V. Volkov, et al.

# 1-D FFT on CUDA GPUs

Data is read from global memory.

Many threads simultaneously compute small FFTs using registers.




Threads exchange data using shared memory.

Finally, data is written back to global memory.

Three CUDA FFTs in 2008 based on this fine-grain parallel implementations

(1) Ours (2) N. Govindaraju, et al. (3) V. Volkov, et al.

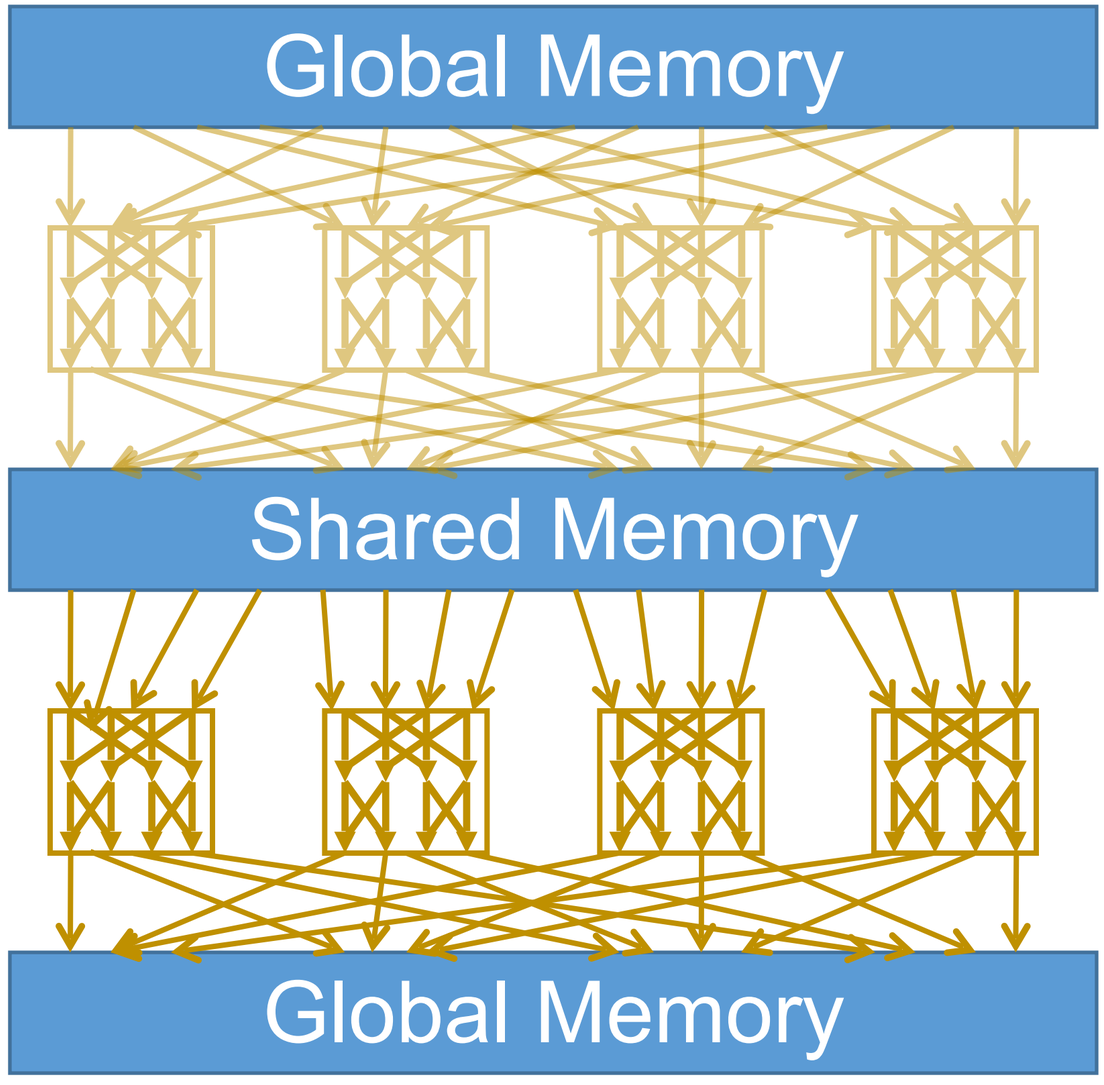
# 1-D FFT on CUDA GPUs

Data is read from global memory.

Many threads simultaneously compute small FFTs using registers.

Threads exchange data using shared memory.

Finally, data is written back to global memory.




Three CUDA FFTs in 2008 based on this fine-grain parallel implementations

(1) Ours (2) N. Govindaraju, et al. (3) V. Volkov, et al.

# Twiddle factor multiplication in CUDA FFT

Twiddle Factors are triangular functions,

and thread-dependent value.

In CUDA, they should come from one of

(1) registers.

(2) table on constant (cache) memory.

(3) table on texture (cache) memory.

(4) table on shared memory.

(5) calculate using SFU each time.

We selected ‘texture plan’ to reduce the

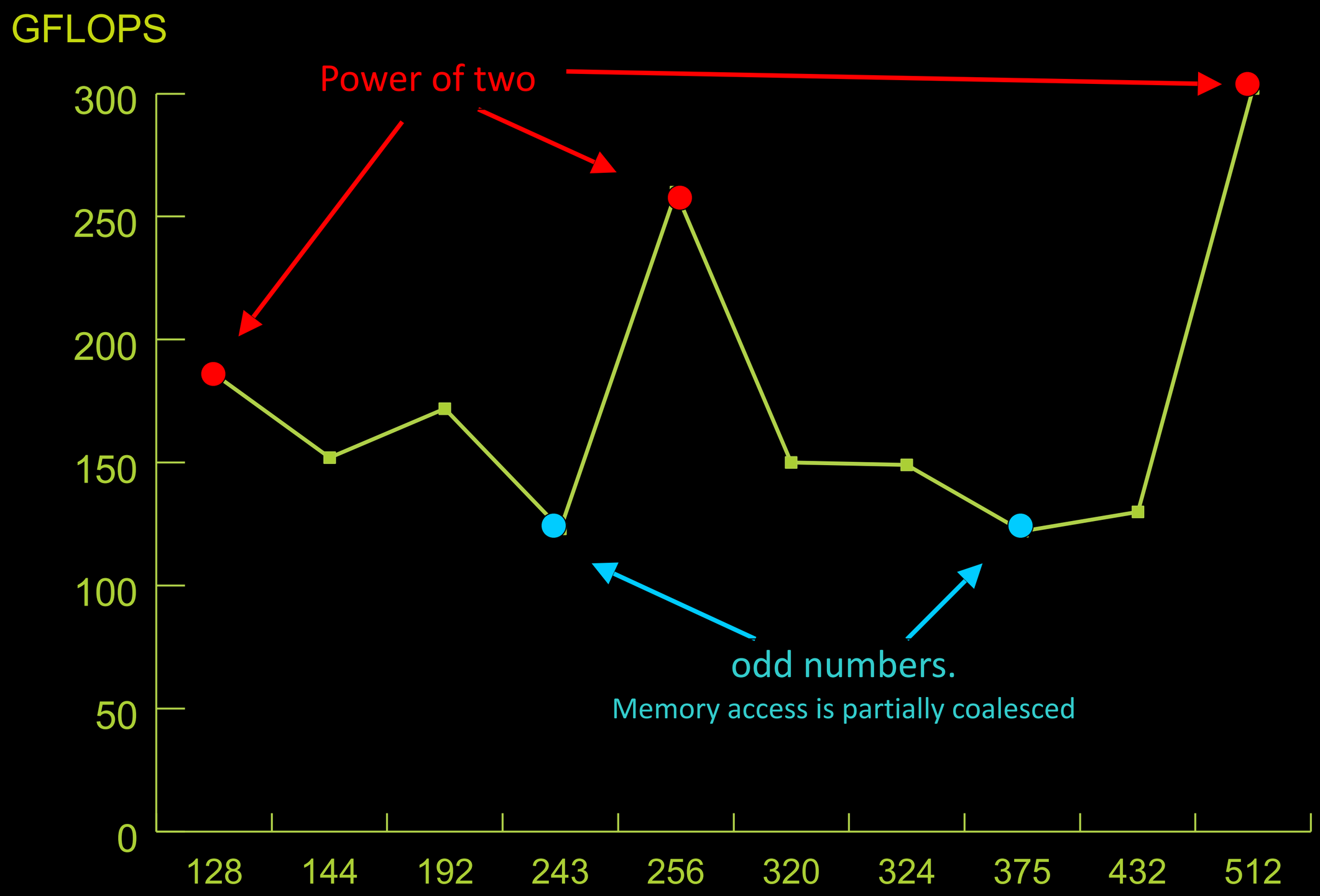
number of instructions and registers.

# Bottle-neck & efficiency | ratio to theoretical peak

- DP performance
  - Bottle-neck on GeForce
- Memory access efficiency
  - Double-complex (double2) data
  - Good for RADEON, Bad for GeForce

| | AMD RADEON HD 6970 | NVIDIA GeForce GTX 580 | NVIDIA Tesla C2050 |
|---|---|---|---|
| Peak DP performance | 675 GFLOPS | 197GFLOPS | 515GFLOPS |
| Achieved performance | 171GFLOPS (25.3%) | 144GFLOPS (73.1%) | 114GFLOPS (22.1%) |
| Peak Memory B/W | 176GB/s | 192GB/s | 144GB/s (128GB/s*9/8) |
| Achieved B/W | 137GB/s (77.8%) | 115GB/s (59.9%) | 91GB/s (71.1%) |
| Bottle-neck | Memory | DP perf. | Memory |

# Performance of 1-D FFT (GTX 280)



# Auto-Tuning of FFT on CUDA GPUs

Many varieties in computing environment:

- Generation of GPU (G8X/G9X, GT2XX)
  - # of registers per SM
- # of cores, clock frequency (shader, memory)
- Compiler version (optimization of PTXJIT)
  - Actual register usage is determined by compiler.
- Driver version (CUDA & display driver)

# Tuning Parameters

(1) Selection of radices of FFT kernels

- combination & ordering

ex) 240-point => 4, 4, 3, 5

# Tuning Parameters

(1) Selection of radices of FFT kernels

- combination & ordering

ex) 240-point => 4, 4, 3, 5

In CPUs, this parameter determines
  # of floating-point ops
  # of load/store to cache memory
In GPUs, this parameter also determines
  # of threads per thread block
  In above example, radix 3 is the smallest.
  Then, # of threads is set to 80 (240/3).

# Tuning Parameters

(1) Selection of radices of FFT kernels

- combination & ordering

ex) 240-point => 4, 4, 3, 5

(2) Selection of number of threads (generic for GPU)

Sufficient thread blocks to exploit memory bandwidth

# Tuning Parameters

(1) Selection of radices of FFT kernels

- combination & ordering

ex) 240-point => 4, 4, 3, 5

(2) Selection of number of threads (generic for GPU)

Sufficient thread blocks to exploit memory bandwidth

(3) Avoid bank conflicts on shared memory (CUDA & FFT)

Insert padding in a specific pattern/rule.

# Sequence Alignment (Wavefront Computation)

M. E. Belviranli, P. Deng, L. N. Bhuyan, R. Gupta, and Q. Zhu, “PeerWave: Exploiting Wavefront Parallelism on GPUs with Peer-SM Synchronization,” presented at the ICS '15: Proceedings of the 29th ACM on International Conference on Supercomputing, 2015.

# Aligning Sequences w/o Insertions and Deletions: Hamming Distance

Given two DNA sequences *v* and *w* :

*v* : A T A T A T A T

*w* : T A T A T A T A

The Hamming distance: $d_H(v, w) = 8$ is large but the sequences are very similar

NANYANG TECHNOLOGICAL UNIVERSITY

# Aligning Sequences w/ Insertions and Deletions

By shifting one sequence over one position:

*v* : A T A T A T A T

*w* :

NANYANG TECHNOLOGICAL UNIVERSITY

# Aligning Sequences w/ Insertions and Deletions

By shifting one sequence over one position:

*v* : A T A T A T A T --

*w* : -- T A T A T A T A

# Aligning Sequences w/ Insertions and Deletions

By shifting one sequence over one position:

*v* : A T A T A T A T --

*w* : -- T A T A T A T A

- The edit distance: $d_H(v, w) = 2$.

NANYANG TECHNOLOGICAL UNIVERSITY

# Aligning Sequences w/ Insertions and Deletions

By shifting one sequence over one position:

*v* : A T A T A T A T --

*w* : -- T A T A T A T A

- The edit distance: $d_H(v, w) = 2$.
- Hamming distance neglects insertions and deletions in DNA

# Edit Distance

*Levenshtein* (1966) introduced **edit distance** between two strings as the minimum number of elementary operations (**insertions**, **deletions**, and **substitutions**) to transform one string into the other

$d(v,w)$ = MIN number of elementary operations to transform $v \rightarrow w$

NANYANG TECHNOLOGICAL UNIVERSITY

# Edit Distance vs. Hamming Distance

Hamming distance *always* compares*: i*-th letter of *v* with *i*-th letter of *w*

*v* = ATATATAT
*w* = TATATATA

Hamming distance:
*d*(*v*,*w*)=8
Computing Hamming distance is trivial

# Edit Distance vs. Hamming Distance

Hamming distance *always* compares*:* *i*-th letter of *v* with *i*-th letter of *w*

*v* = ATATATAT

*w* = TATATATA

Just one shift

Make it all line up

Hamming distance:

*d*(*v*,*w*)=8

Computing Hamming distance is trivial

NANYANG TECHNOLOGICAL UNIVERSITY

# Edit Distance vs. Hamming Distance

Hamming distance *always* compares*: i*-th letter of *v* with *i*-th letter of *w*

Edit distance *may* compare *i*-th letter of *v* with *j*-th letter of *w*

*v* = ATATATAT

*w* = TATATATA

Just one shift

Make it all line up

Hamming distance:

*d*(*v*,*w*)=8

Computing Hamming distance is trivial

# Edit Distance vs. Hamming Distance

Hamming distance *always* compares*: i*-th letter of *v* with *i*-th letter of *w*

Edit distance *may* compare *i*-th letter of *v* with *j*-th letter of *w*

*v* = ATATATAT

*w* = TATATATA

Just one shift

Make it all line up

*v* = - ATATATAT

*w* = TATATATA

Hamming distance:

*d*(*v*,*w*)=8

Computing Hamming distance is trivial

# Edit Distance vs. Hamming Distance

Hamming distance *always* compares*: i*-th letter of *v* with *i*-th letter of *w*

Edit distance *may* compare *i*-th letter of *v* with *j*-th letter of *w*

*v* = ATATATAT
*w* = TATATATA

Just one shift
Make it all line up

*v* = - ATATATAT
*w* = TATATATA

Hamming distance:
*d*(*v*,*w*)=8
Computing Hamming distance is trivial

Edit distance:
*d*(*v*,*w*)=2
Computing Edit distance is non-trivial

NANYANG TECHNOLOGICAL UNIVERSITY

# Sequence Alignment

Given 2 DNA sequences v and w:

*v* : A T G T T A T  *m* = 7

*w* : A T C G T A C  *n* = 7

# Sequence Alignment

Given 2 DNA sequences v and w:

*v* : A T G T T A T  *m* = 7

*w* : A T C G T A C  *n* = 7

An alignment is a 2×*k* matrix (where *m*+*n* ≥ *k* ≥ max{*m*,*n*})

NANYANG TECHNOLOGICAL UNIVERSITY

# Sequence Alignment

Given 2 DNA sequences v and w:

*v* : A T G T T A T  *m* = 7

*w* : A T C G T A C  *n* = 7

An alignment is a 2×*k* matrix (where *m*+*n* ≥ *k* ≥ max{*m*,*n*})

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| letters of *v* | A | T | -- | G | T | T | A | T | -- |
| letters of *w* | A | T | C | G | T | -- | A | -- | C |

NANYANG TECHNOLOGICAL UNIVERSITY

# Sequence Alignment

Given 2 DNA sequences v and w:

*v* : A T G T T A T  *m* = 7

*w* : A T C G T A C  *n* = 7

An alignment is a 2×*k* matrix (where *m*+*n* ≥ *k* ≥ max{*m*,*n*})

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| letters of *v* | A | T | -- | G | T | T | A | T | -- |
| letters of *w* | A | T | C | G | T | -- | A | -- | C |

5 matches | 2 insertions | 2 deletions

NANYANG TECHNOLOGICAL UNIVERSITY

# Smith-Waterman

*Acceleration of the Smith–Waterman algorithm using single and multiple graphics processors*

*Ali Khajeh-Saeed, Stephen Poole, J. Blair Perot*

$$H_{i,j} = Max\begin{cases} Max(H_{i-1,j-1} + S_{i,j}, 0) \\ Max_{0<k<i}(H_{i-k,j} - (G_s + kG_e)) \\ Max_{0<k<j}(H_{i,j-k} - (G_s + kG_e)) \end{cases}$$

| | **0** | **C** | **A** | **G** | **C** | **C** | **U** | **C** | **G** | **C** | **U** | **U** | **A** | **G** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | *0* | *0* | *0* | *0* | *0* | *0* | *0* | *0* | *0* | *0* | *0* | *0* | *0* | *0* |
| **A** | *0* | 0 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | | | | | |
| **A** | *0* | 0 | 5 | 2 | 0 | 0 | 0 | 0 | 0 | | | | | |
| **U** | *0* | 0 | 0 | 2 | 0 | 0 | 5 | 0 | 0 | | | | | |
| **G** | *0* | 0 | 0 | 5 | 0 | 0 | 0 | 2 | 5 | | | | | |
| **C** | *0* | 5 | 0 | 0 | 10 | 5 | 0 | 5 | 0 | | | | | |
| **C** | *0* | 5 | 2 | 0 | 5 | 15 | 6 | 5 | **4** | | | | | |
| **A** | *0* | | | | | | | | | | | | | |
| **U** | *0* | | | | | | | | | | | | | |
| **U** | *0* | | | | | | | | | | | | | |
| **G** | *0* | | | | | | | | | | | | | |
| **C** | *0* | | | | | | | | | | | | | |
| **C** | *0* | | | | | | | | | | | | | |
| **G** | *0* | | | | | | | | | | | | | |
| **G** | *0* | | | | | | | | | | | | | |

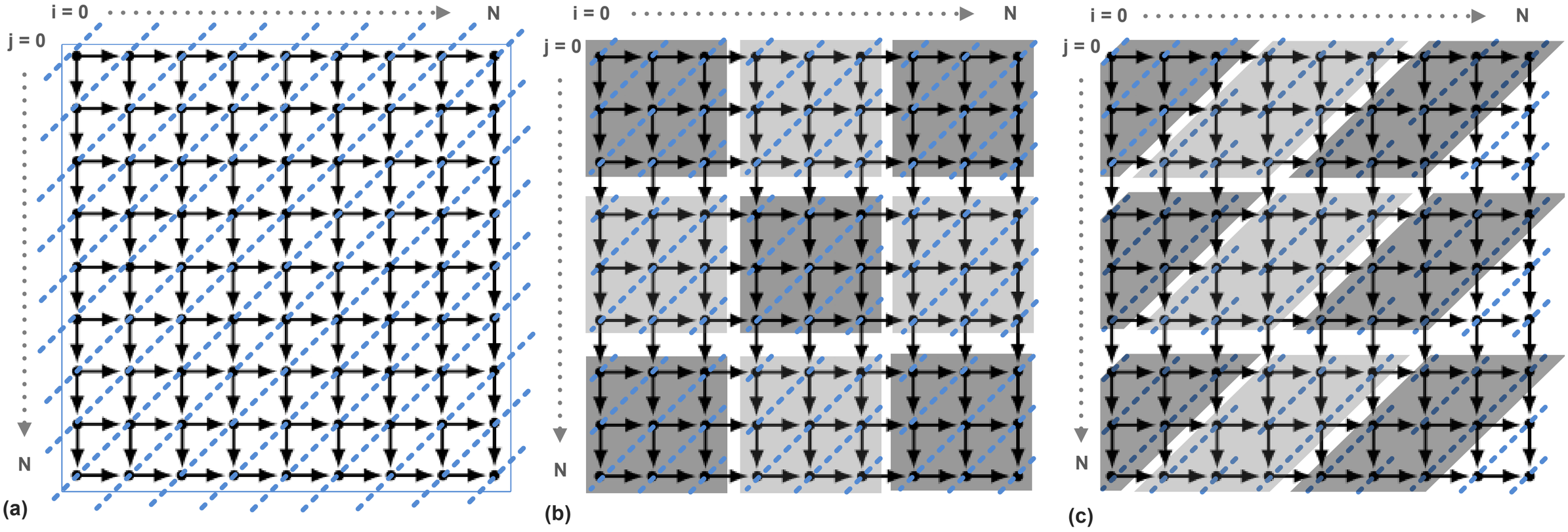
**Fig. 2.** Dependency of the values in the Smith–Waterman table.

# Smith-Waterman

| | 0 | C | A | G | C | C | U | C | G | C | U | U | A | G |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | *0* | *0* | *0* | *0* | *0* | *0* | *0* | *0* | *0* | *0* | *0* | *0* | *0* | *0* |
| **A** | *0* | 0 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ? | | | |
| **A** | *0* | 0 | 5 | 2 | 0 | 0 | 0 | 0 | 0 | ? | | | | |
| **U** | *0* | 0 | 0 | 2 | 0 | 0 | 5 | 0 | ? | | | | | |
| **G** | *0* | 0 | 0 | 5 | 0 | 0 | 0 | ? | | | | | | |
| **C** | *0* | 5 | 0 | 0 | 10 | 5 | ? | | | | | | | |
| **C** | *0* | 5 | 2 | 0 | 5 | | | | | | | | | |
| **A** | *0* | 0 | 10 | 1 | ? | | | | | | | | | |
| **U** | *0* | 0 | 1 | ? | | | | | | | | | | |
| **U** | *0* | 0 | ? | | | | | | | | | | | |
| **G** | *0* | ? | | | | | | | | | | | | |
| **C** | *0* | | | | | | | | | | | | | |
| **C** | *0* | | | | | | | | | | | | | |
| **G** | *0* | | | | | | | | | | | | | |
| **G** | *0* | | | | | | | | | | | | | |

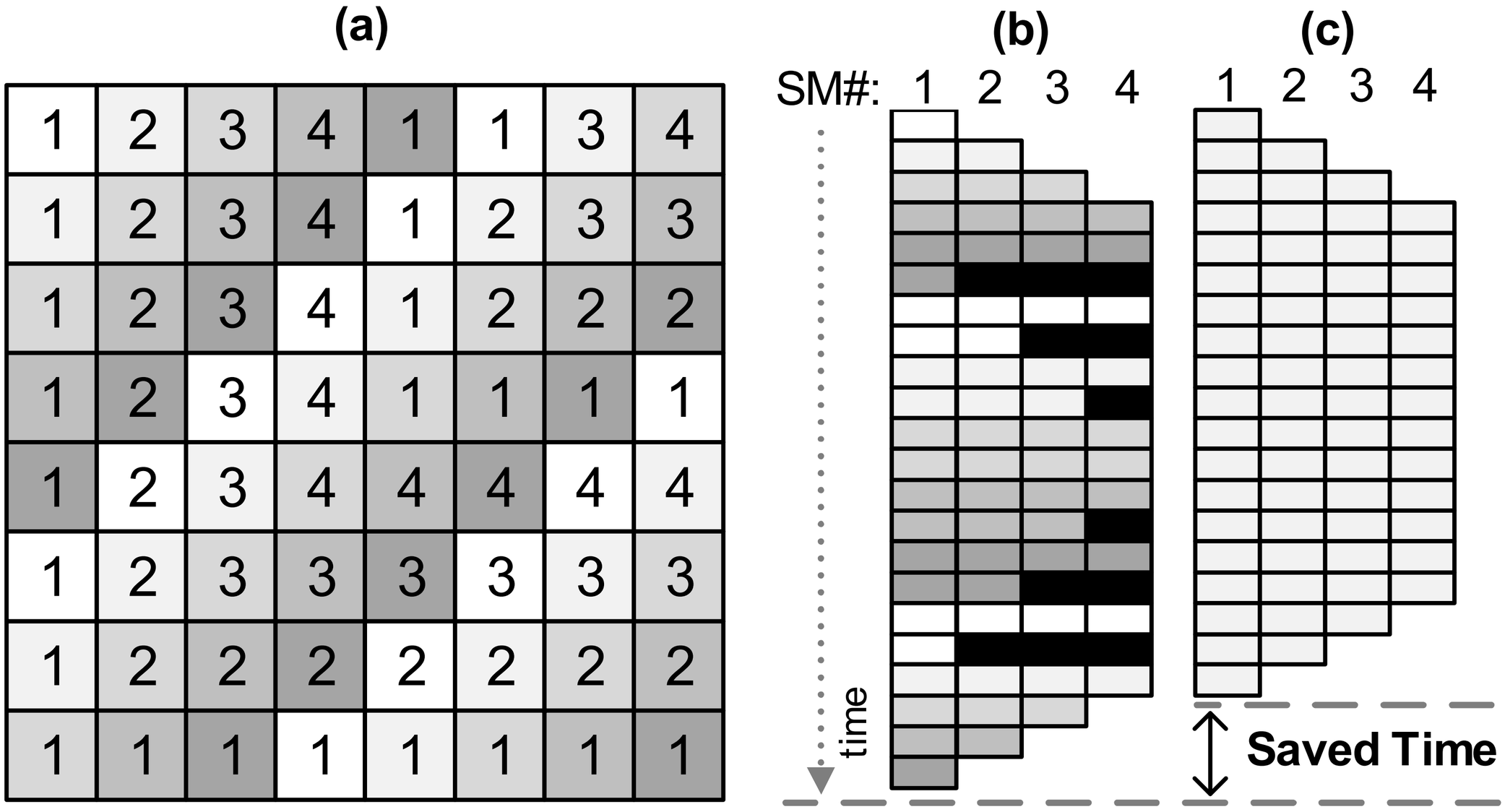
**Fig. 4.** Anti-diagonal method and dependency of the cells.

# Hyperplanes




M. E. Belviranli, P. Deng, L. N. Bhuyan, R. Gupta, and Q. Zhu, “PeerWave: Exploiting Wavefront Parallelism on GPUs with Peer-SM Synchronization,” presented at the ICS '15: Proceedings of the 29th ACM on International Conference on Supercomputing, 2015.

# But global barriers are expensive



# Solution: Peer-to-peer synchronization