Cache and Data Structures

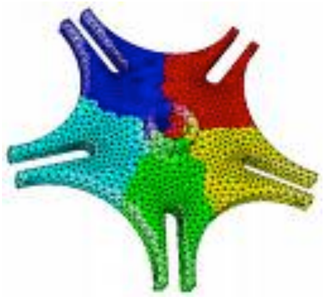
Bruno Raffin
DataMove
INRIA - LIG
GRENOBLE

Part of the Slides are from

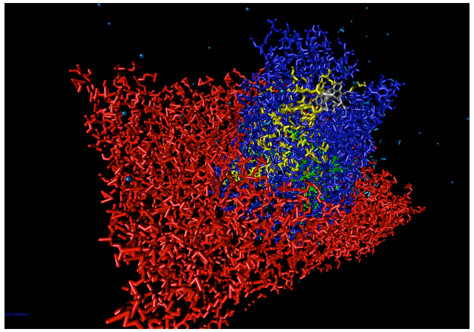
Marc Tchiboukdjian

and

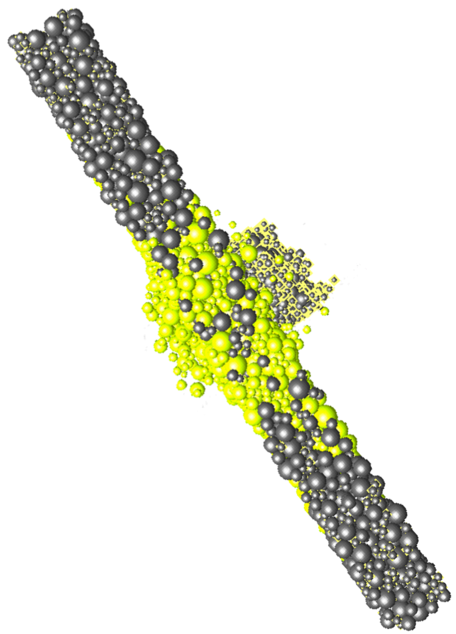
Marie Durand



Motivation 1/2



- Numerical simulations:
  - 3D objects: meshes, particles
  - Spatial and temporal coherency

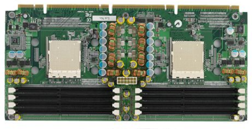
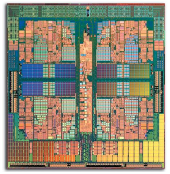


- Computer memories: 1D

Motivation 2/2

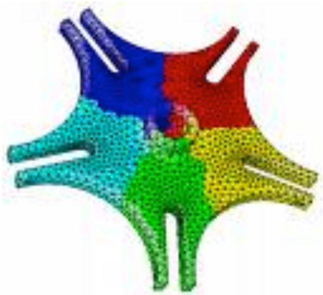
Today's machines:

-> complex memory hierarchies

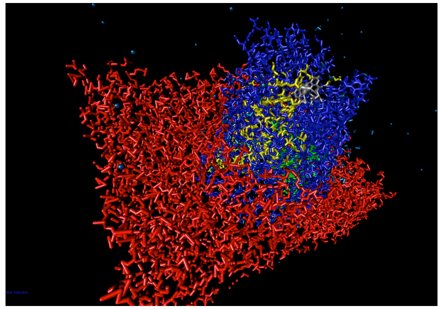





Access by blocks of continuous data (memory pages, cache lines, read/write coalescing)

Need to carefully consider data access schemes and memory layouts



Spatial Coherency



(3D) Neighbor data tend to be accessed together

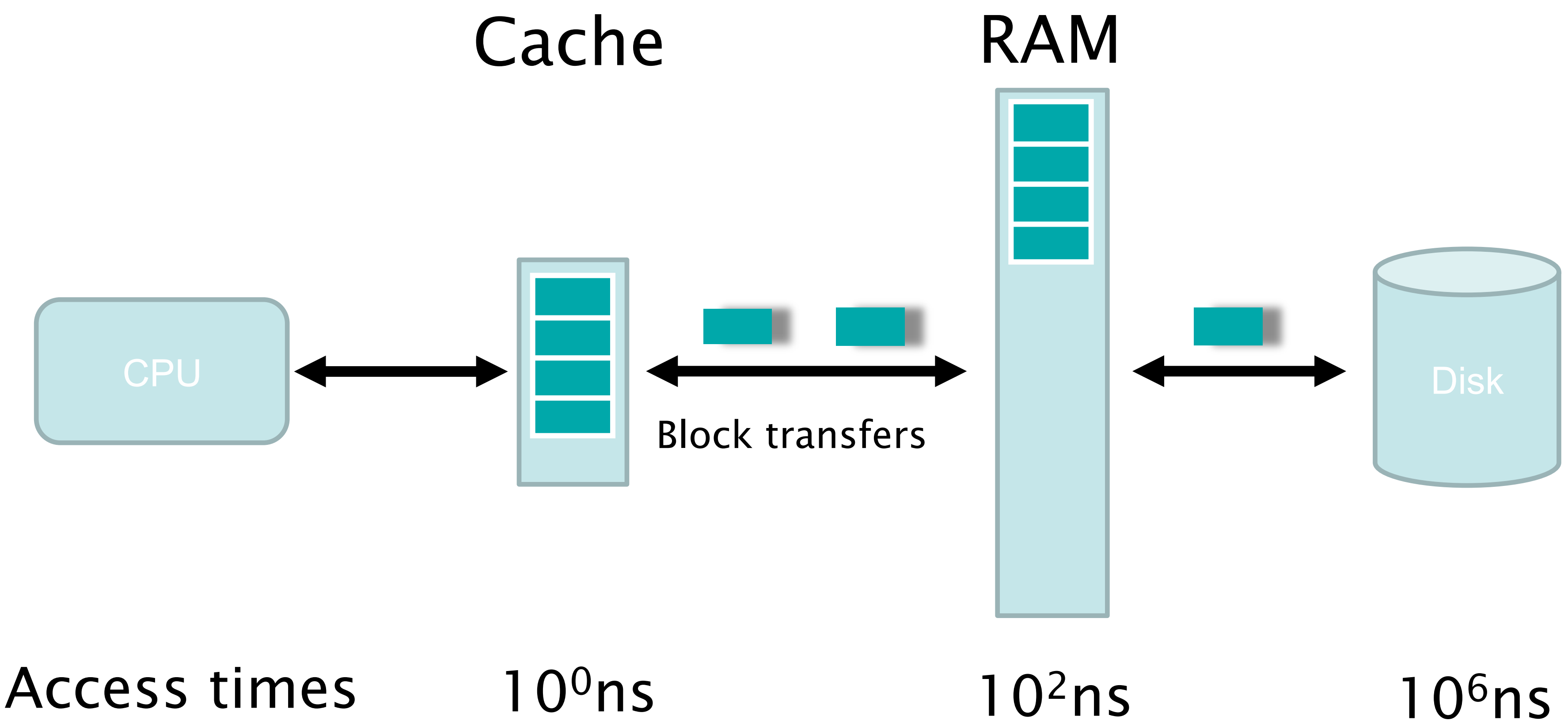
-> Mesh topology, Atoms, etc.

Try to keep this 3D locality when projecting the data in the 1D memory:

Goal: Access n neighbor data by

n/B memory block transfers (B-size)

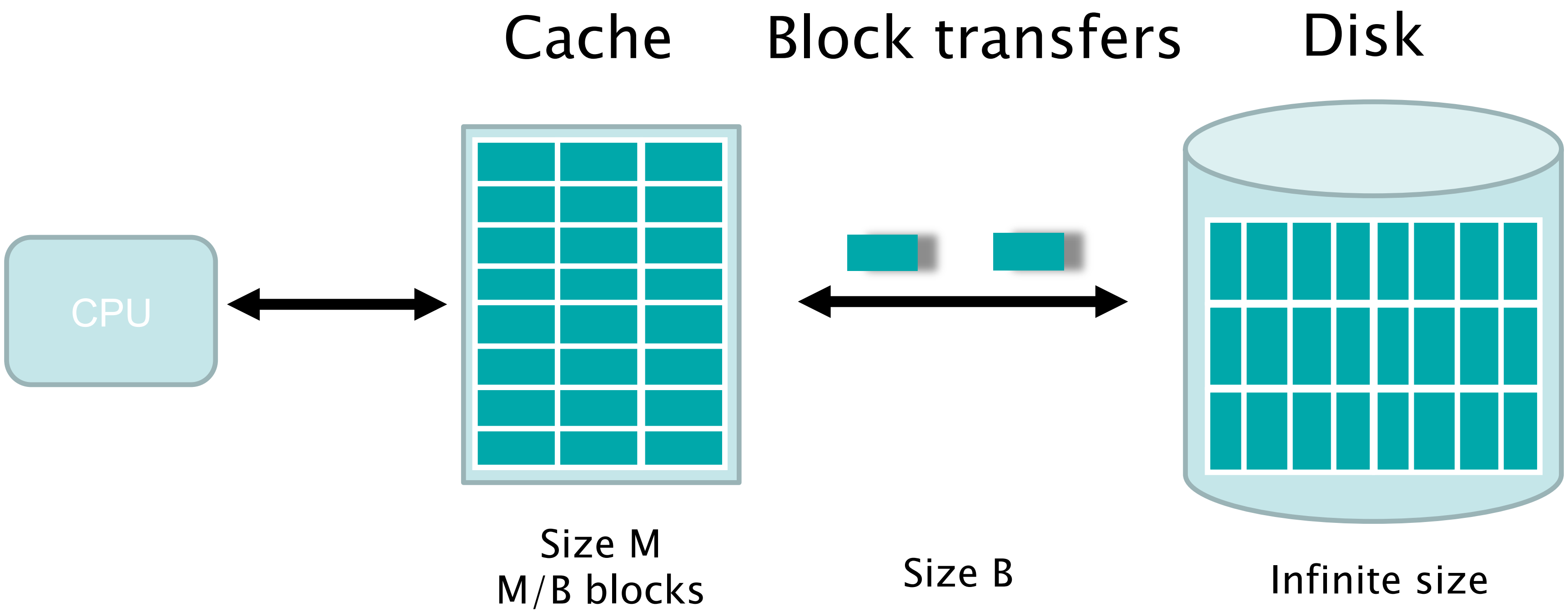
Memory Hierarchy



Disk Access Model (DAM)

or external memory
out-of-core
cache-aware
I/O model

[Aggarwal and Vitter 1988]

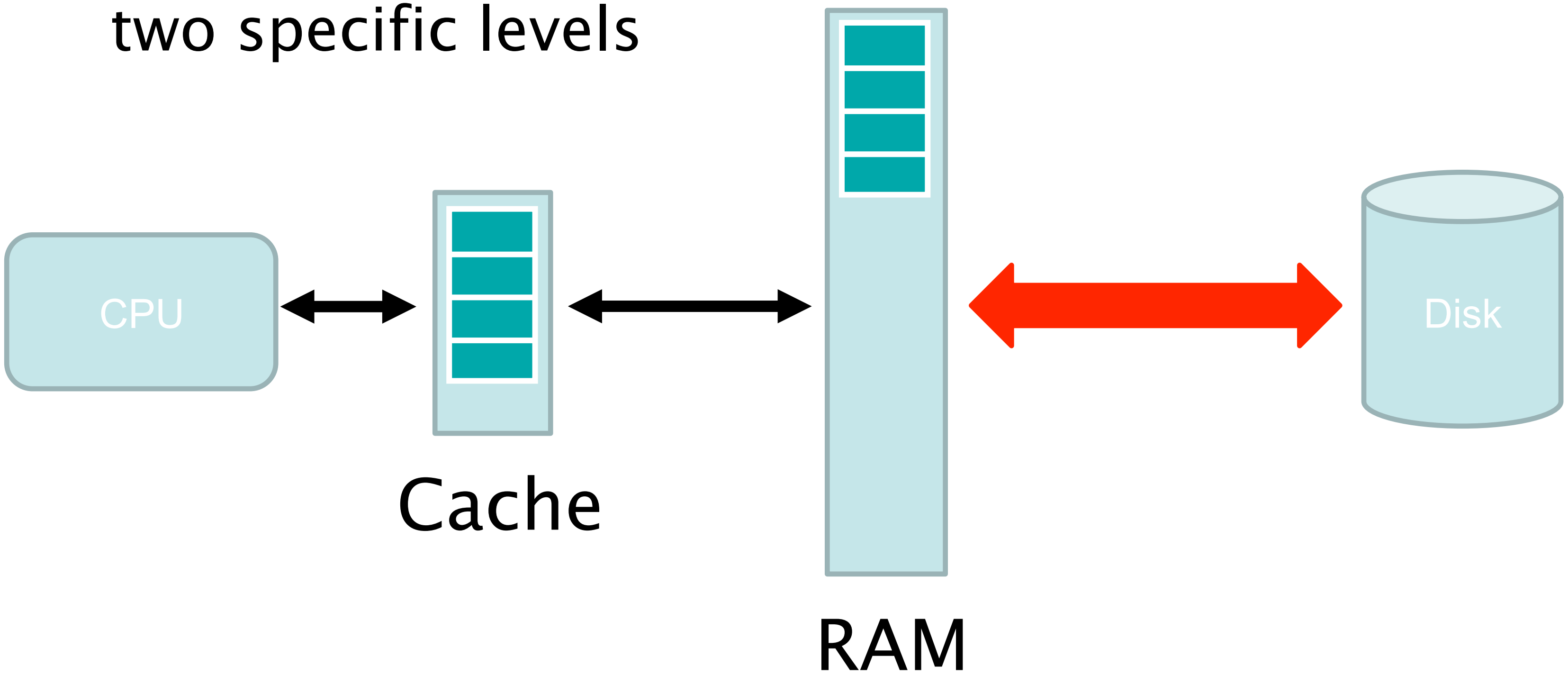


W: #operations CPU

Q: #block transfers

Advantages of the DAM model

- Simple: only two levels
- Good when the bottleneck is between two specific levels

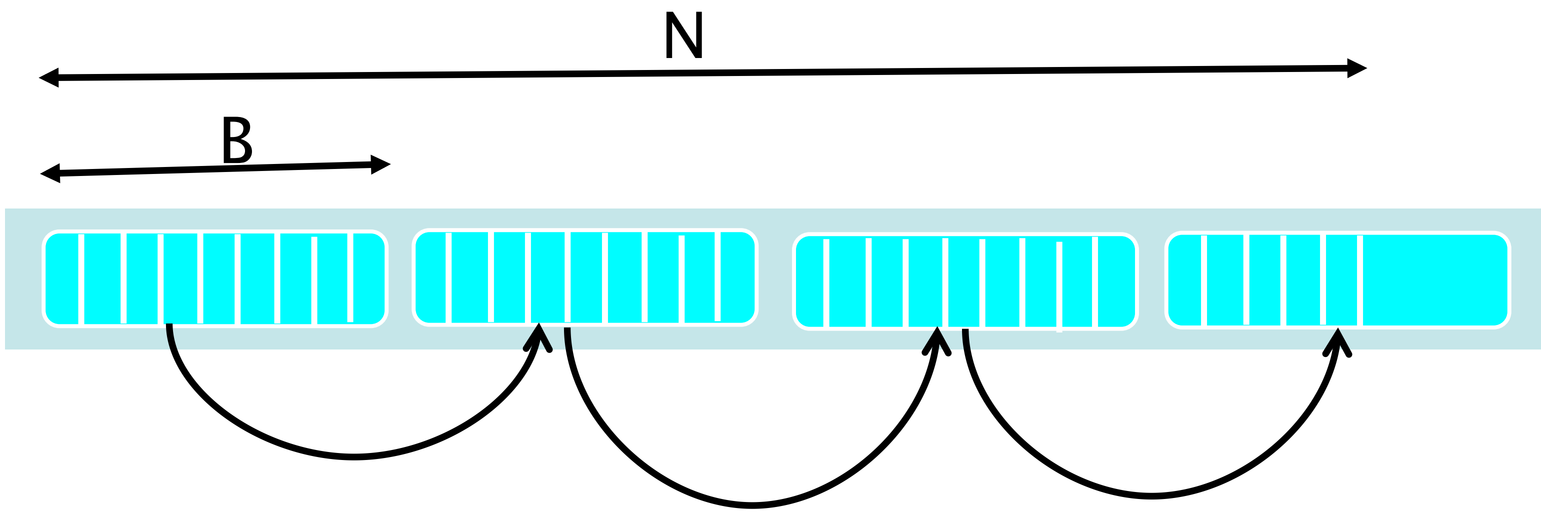


Principles of external-memory algorithm design

- *Internal efficiency:* work is comparable to the best internal memory algorithms
- *Spatial locality:* a block should contain as much useful data as possible
- *Temporal locality:* as much useful work as possible before the block is ejected

Scanning in the DAM model

Read an N-elements array: the naive algorithm is optimal



$$W(N) = N$$

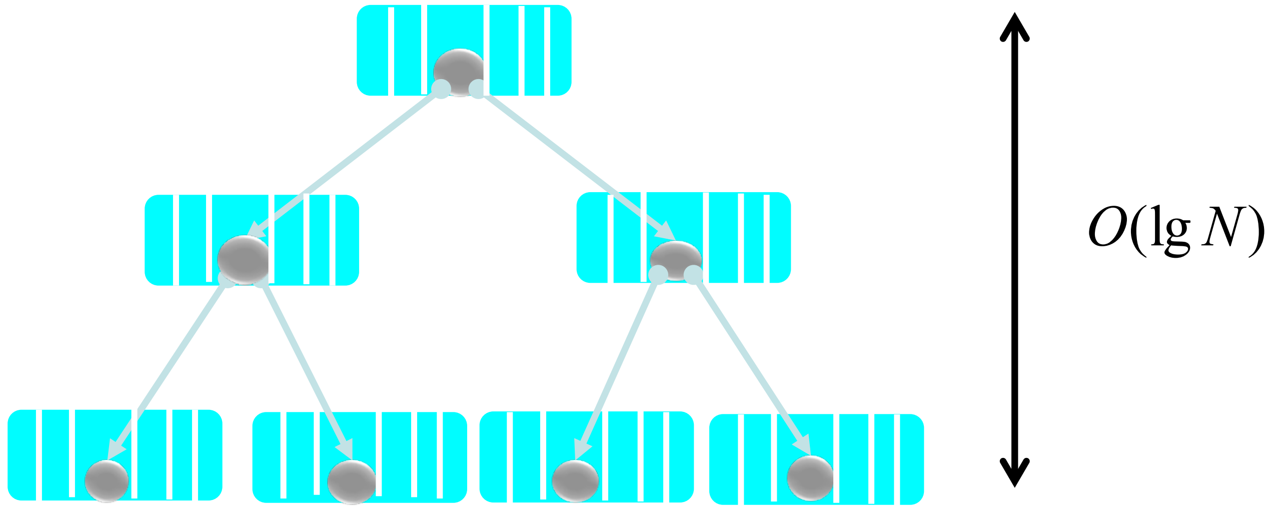
$$Q(N) = \lceil N / B \rceil$$

$$scan(N) = \lceil N / B \rceil$$

this bound is optimal

Searching in the DAM model

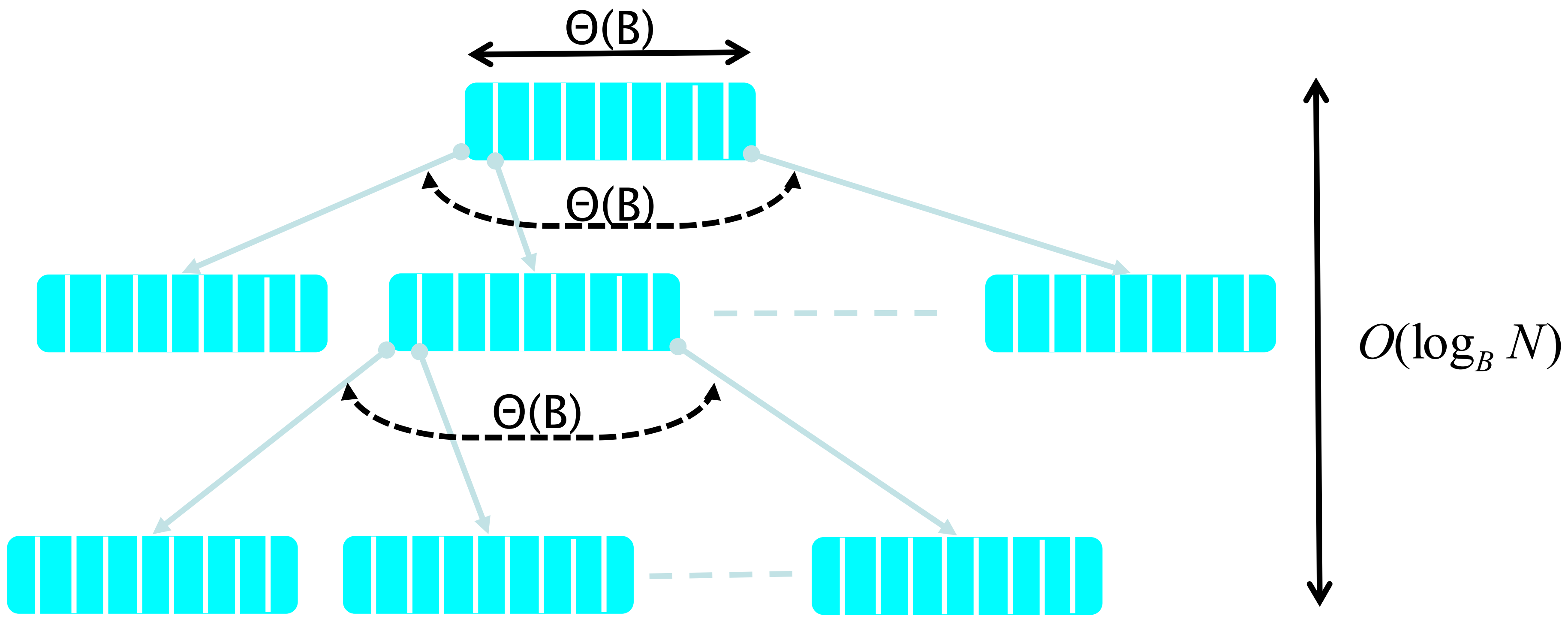
Searching a key in an N-nodes balanced binary tree : naive doesn't work



$$W(N) = 1.O(\lg N) = O(\mathbf{l}\mathrm{g}N)$$
$$Q(N) = 1.O(\lg N) = O(\lg N)$$

Searching in the DAM model

Searching a key in an N-elements B-tree [Bayer and McCreight 1972]



$$W(N) = \lg B . O(\log_B N) = O(\lg N)$$

$$Q(N) = 1 . O(\log_B N) = O(\log_B N)$$

Multiplying in the DAM model

NxN matrices in row-major order : naive doesn't work

Using the naive N^3 algorithm:

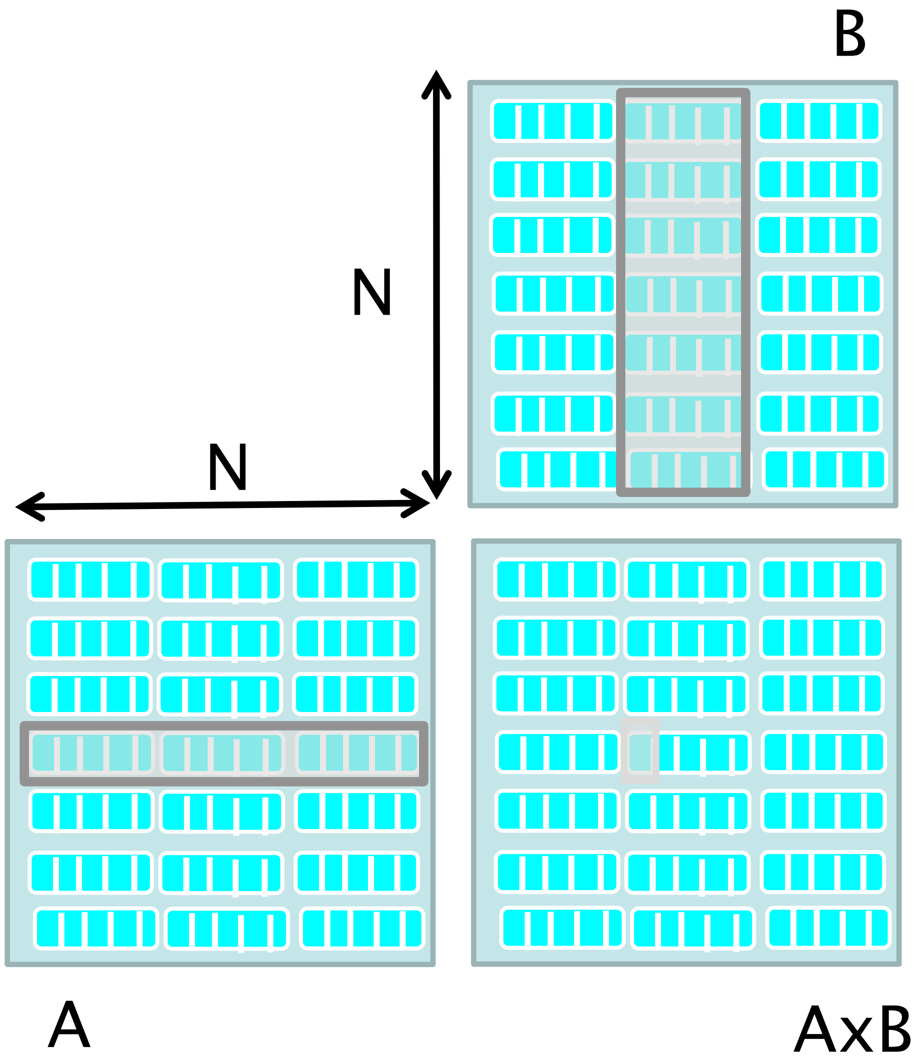
$$W(N) = O(N).N^2$$

$$W(N) = O(N^3)$$

Memory accesses in B are suboptimal:

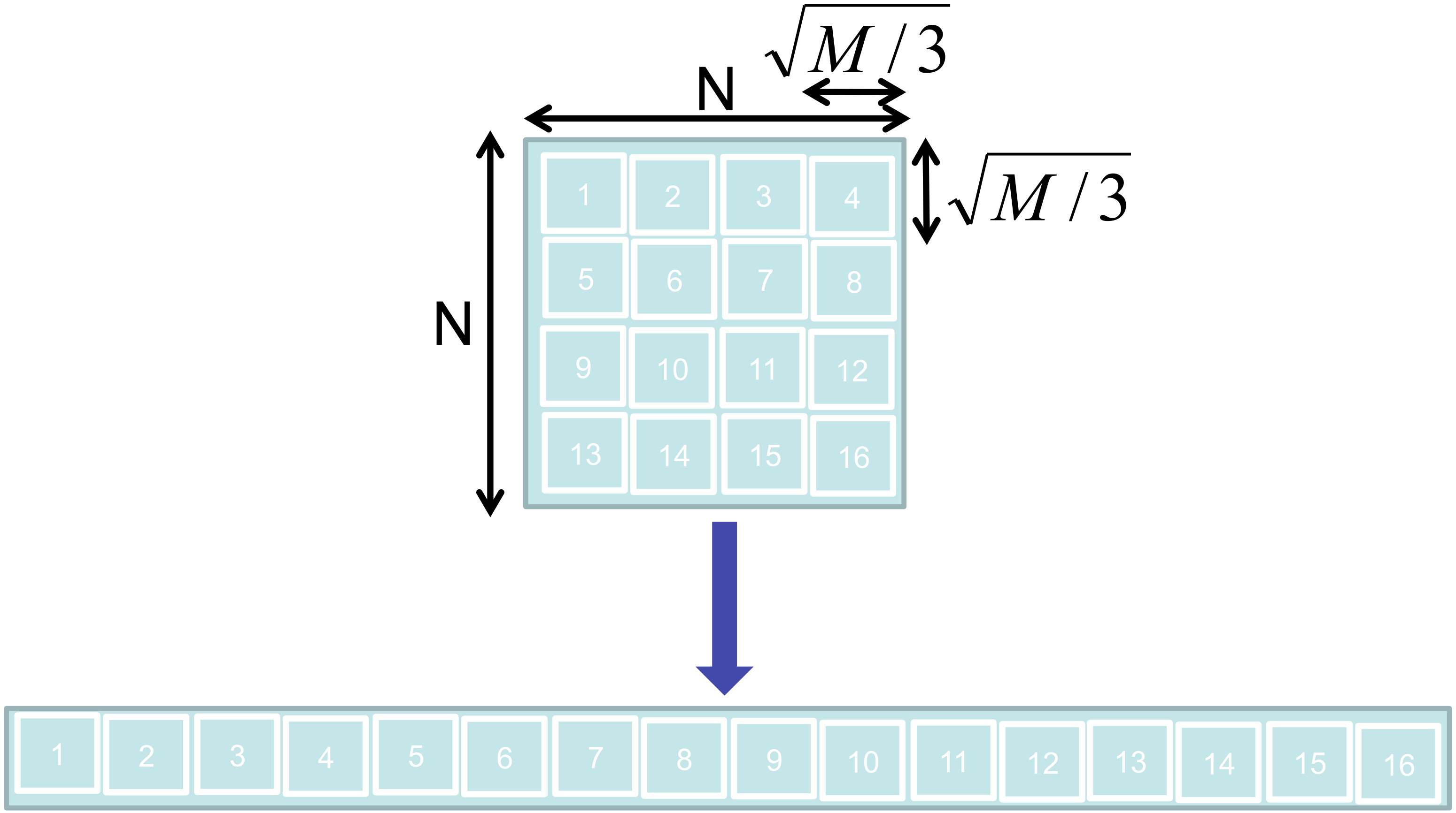
$$Q(N) = O\left(\frac{N}{B} + N\right).N^2$$

$$Q(N) = O\left(N^3\right)$$



Multiplying in the DAM model

NxN matrices in submatrices



Multiplying in the DAM model

NxN matrices in submatrices

- Cost for two sub-matrices

$$W(N) = O\left(\sqrt{M}^3\right) \quad Q(N) = O\left(\frac{M}{B}\right)$$

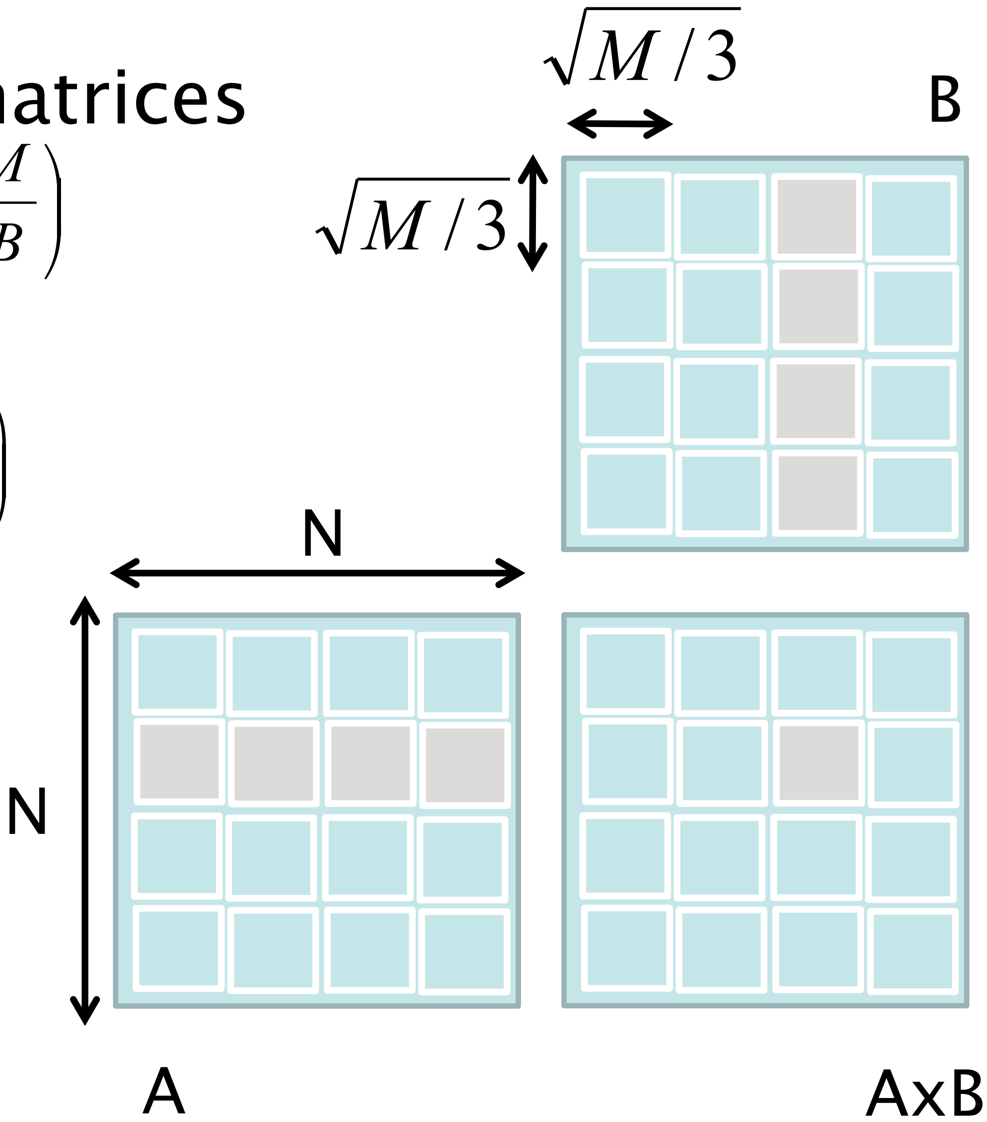
- Total cost

$$W(N) = O\left(\sqrt{M}^3\right).O\left(\frac{N}{\sqrt{M}}\right).O\left(\frac{N^2}{M}\right)$$

$$W(N) = O\left(N^3\right)$$

$$Q(N) = O\left(\frac{M}{B}\right).O\left(\frac{N}{\sqrt{M}}\right).O\left(\frac{N^2}{M}\right)$$

$$Q(N) = O\left(\frac{N^3}{B\sqrt{M}}\right)$$



Sorting in the DAM model

M/B-way merge sort of an N-elements array

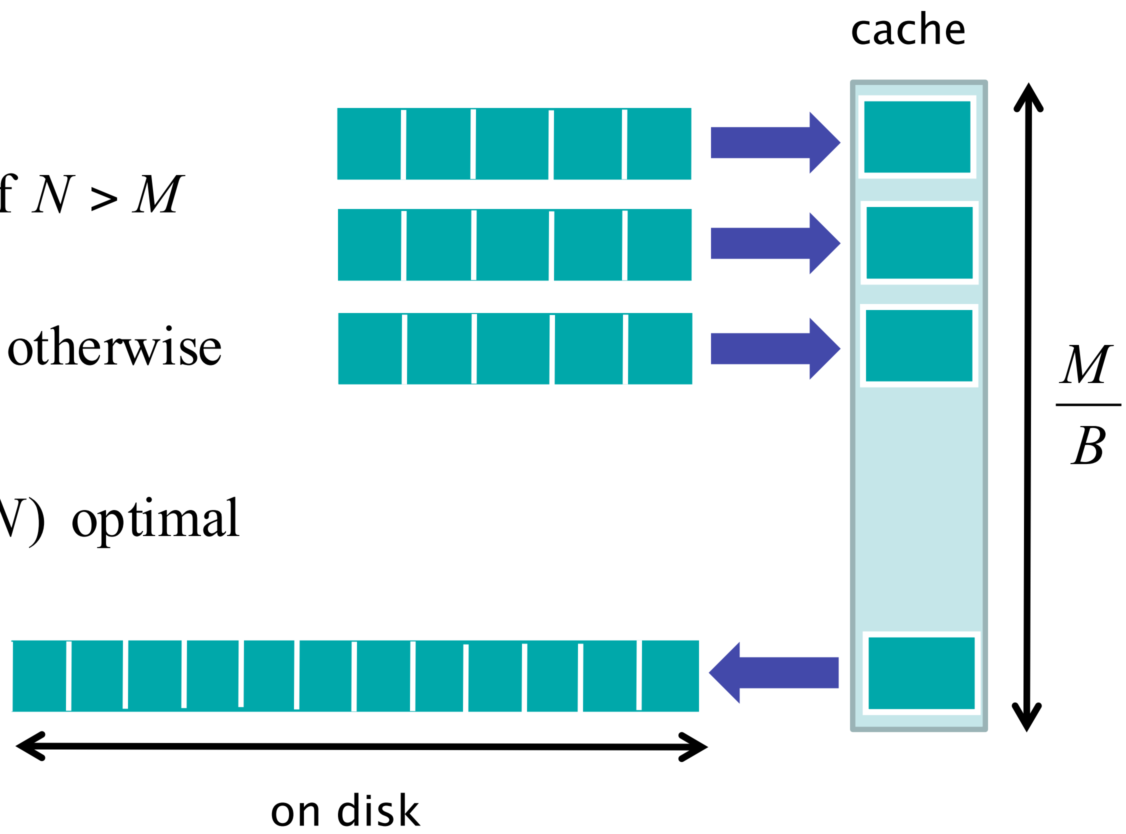
$$W(N) = \begin{cases} \frac{M}{B} W\left(\frac{N}{M/B}\right) + N.O\left(\log \frac{M}{B}\right) & \text{if } N > 1 \\ O(1) & \text{otherwise} \end{cases}$$

$$W(N) = O(N \log N)$$

$$Q(N) = \begin{cases} \frac{M}{B} Q\left(\frac{N}{M/B}\right) + O\left(\frac{N}{B}\right) & \text{if } N > M \\ O\left(\frac{N}{B}\right) & \text{otherwise} \end{cases}$$

$$Q(N) = O\left(\frac{N}{B} \log_{M/B} \frac{N}{B}\right) = sort(N) \text{ optimal}$$

- Cut into M/B sublists
- Recursively sort them
- Merge using a heap of size M/B



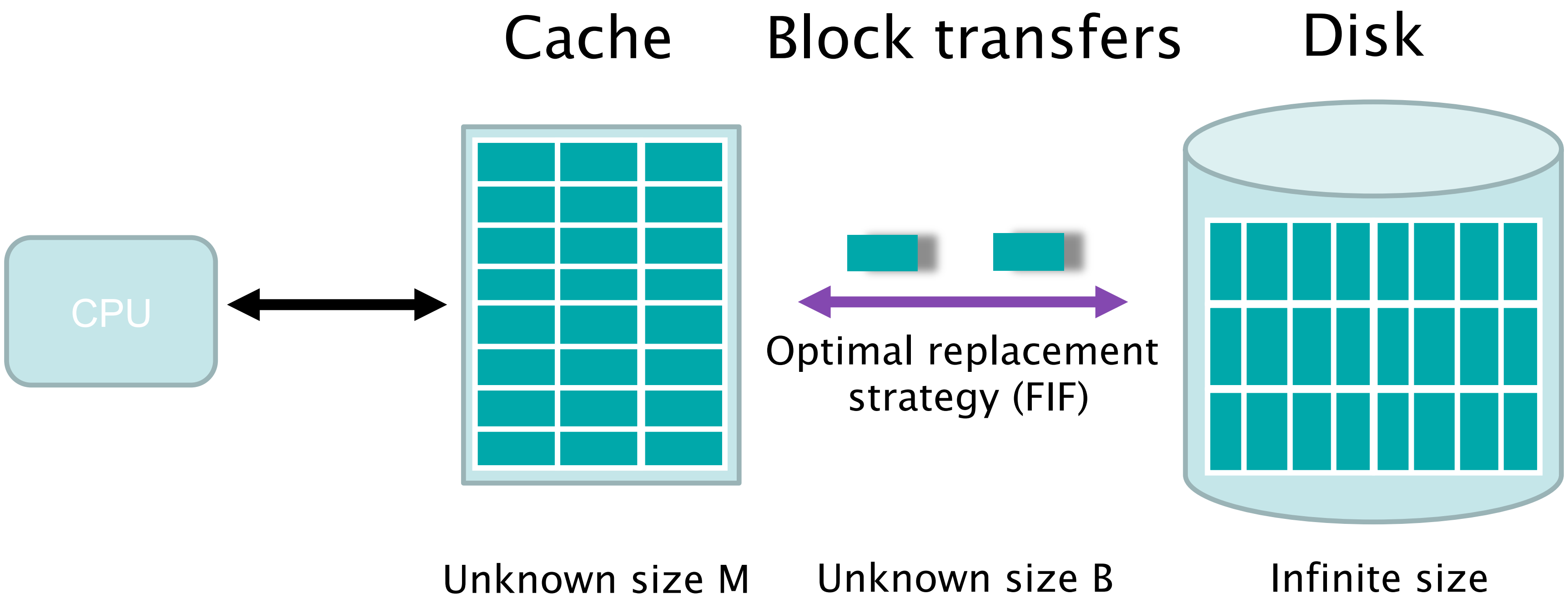
Limitations of the DAM model

- B and M are needed to design the algorithm
- Only two levels of the hierarchy
- B and M can vary
  - e.g. multi-process scheduling
- Block transfer cost is not uniform
  - disk seek time

DAM Based Algorithms are said to be "cache-aware "

Cache-Oblivious Model (CO)

[Frigo et al 1999]



Advantages of the CO Model

Parameters are unknown when writing the algorithm (block and cache size):

- Machine-independent
- Efficient with all levels of the memory hierarchy

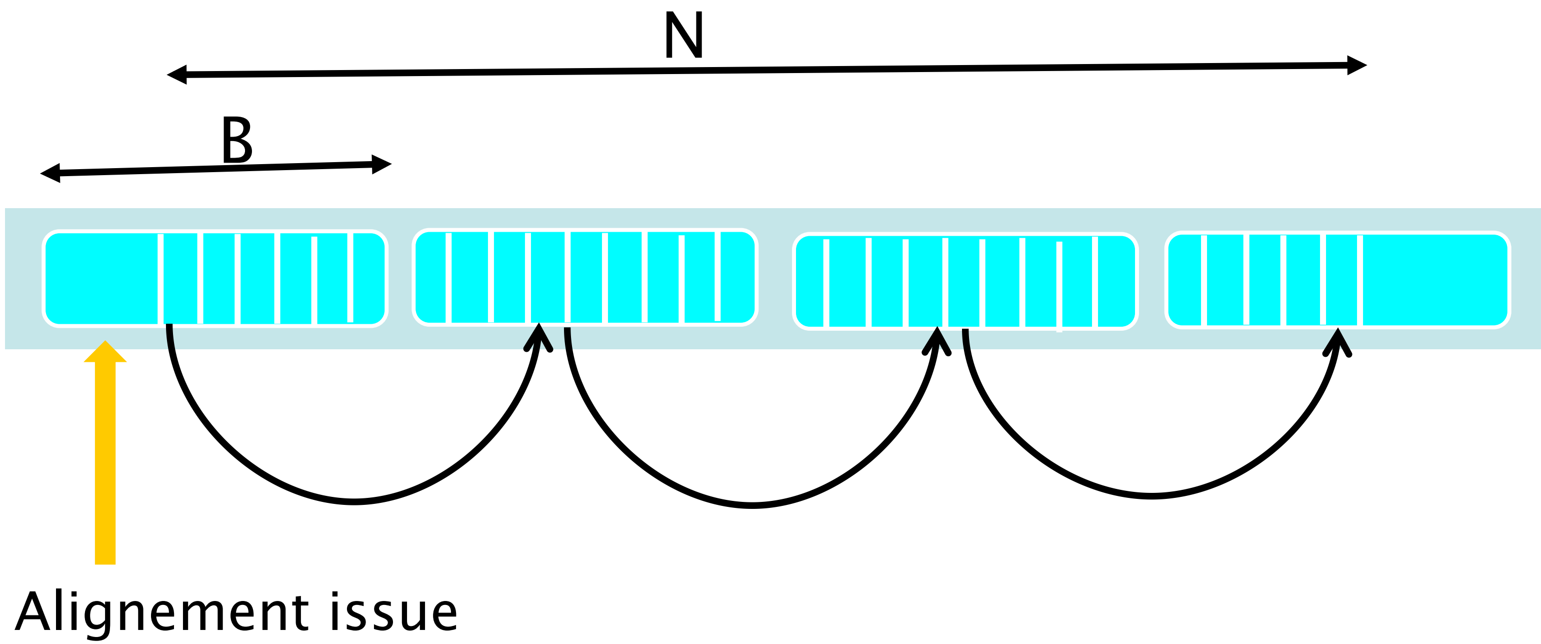
Assumptions

- Optimal replacement
- Only two levels of memory
- Full associativity
- Tall-cache assumption

  $M = \Omega(B^2)$

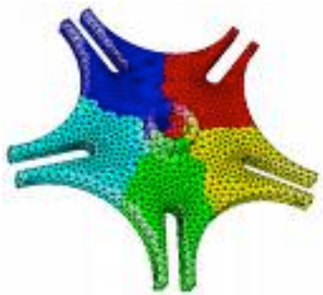
  $M = \Omega(B^{1+\varepsilon})$

Scanning in the CO model

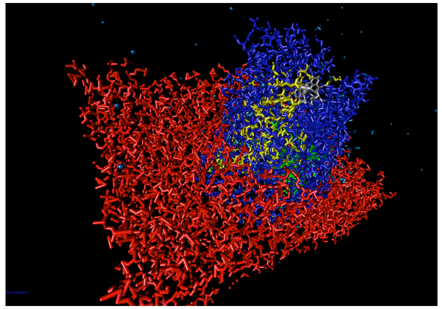


$$W(N) = N$$

$$Q(N) = \lceil N / B \rceil + 1$$



Spatial Coherency



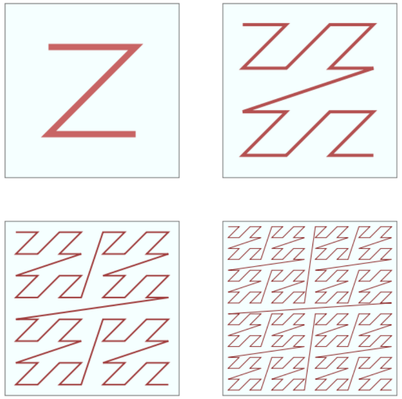
(3D) Neighbor data tend to be accessed together

-> Mesh topology, Atoms, etc.

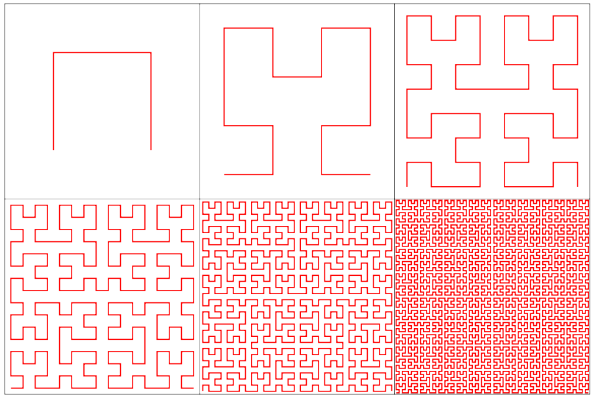
Try to keep this 3D locality when projecting the data in the 1D memory:

Goal: Access n neighbor data by
n/B memory block transfers (B-size)

Space-filling Curves



Morton Curve

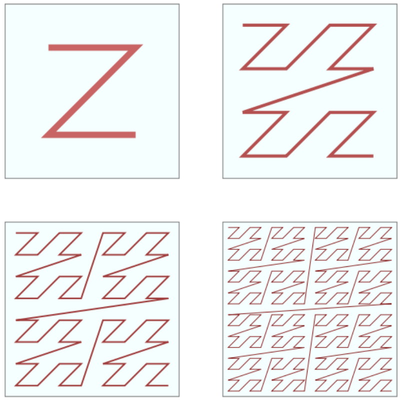


Hilbert Curve

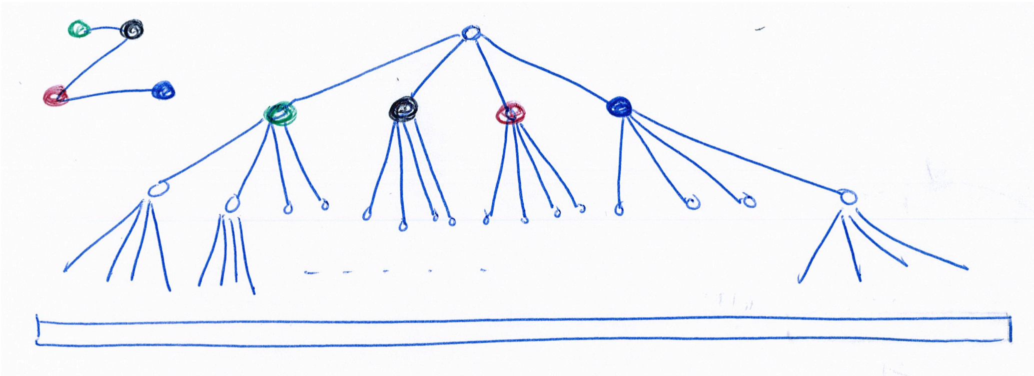
(x,y,z) -> Z-index by bit switches

Examples in 2D, but extends to higher dimensions

Morton Curve

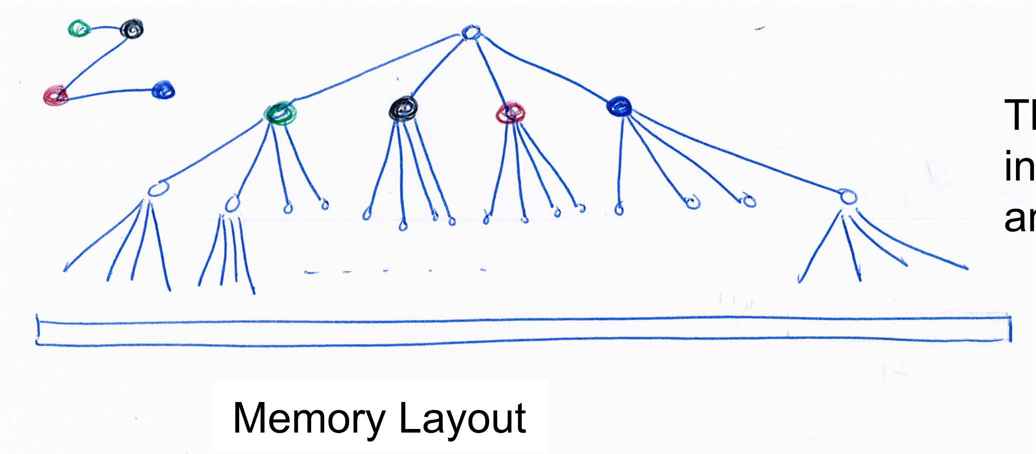


Morton Curve



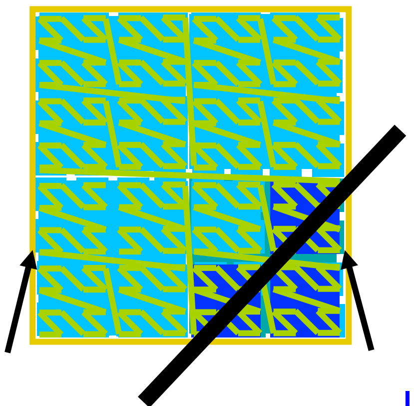
Memory Layout

Morton Curve: Cache Oblivious Data Layouts



The layout is computed independently from a given M and B

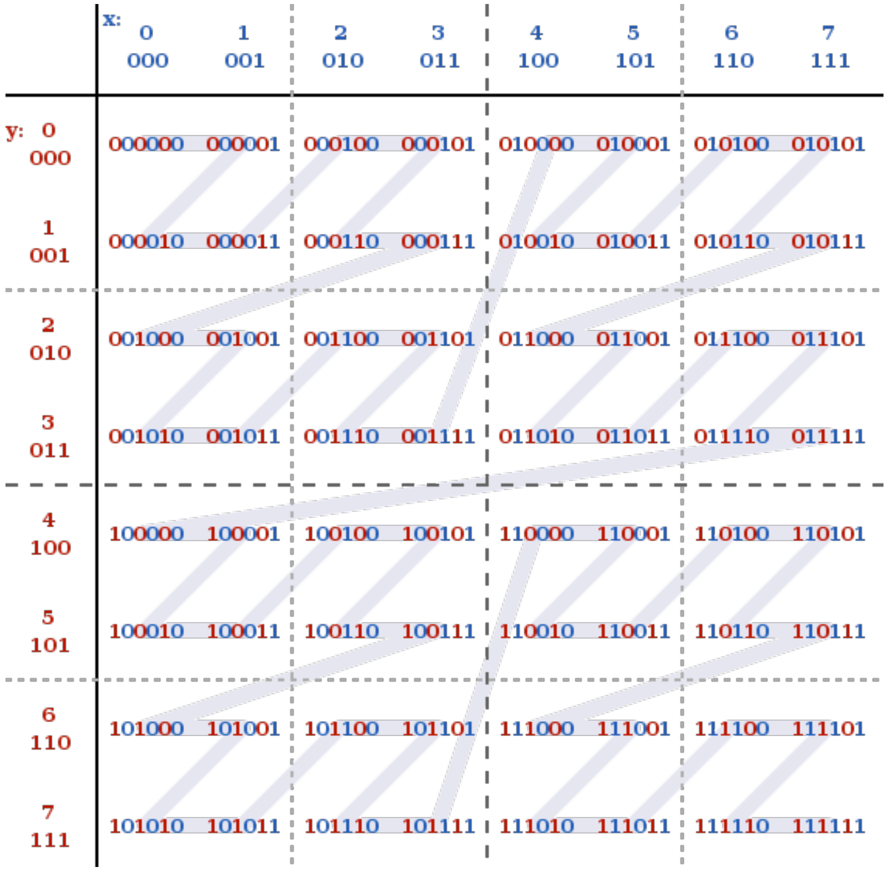
Spatially coherent accesses will show a good cache behavior



See [Pascucci, Siggraph-2005] or [Tchiboukdjian, TVCG 2010] for mesh specific CO layouts

Morton Curve Indexing

Z-index obtained by Interleaving the binary coordinates of x and y



Z-index

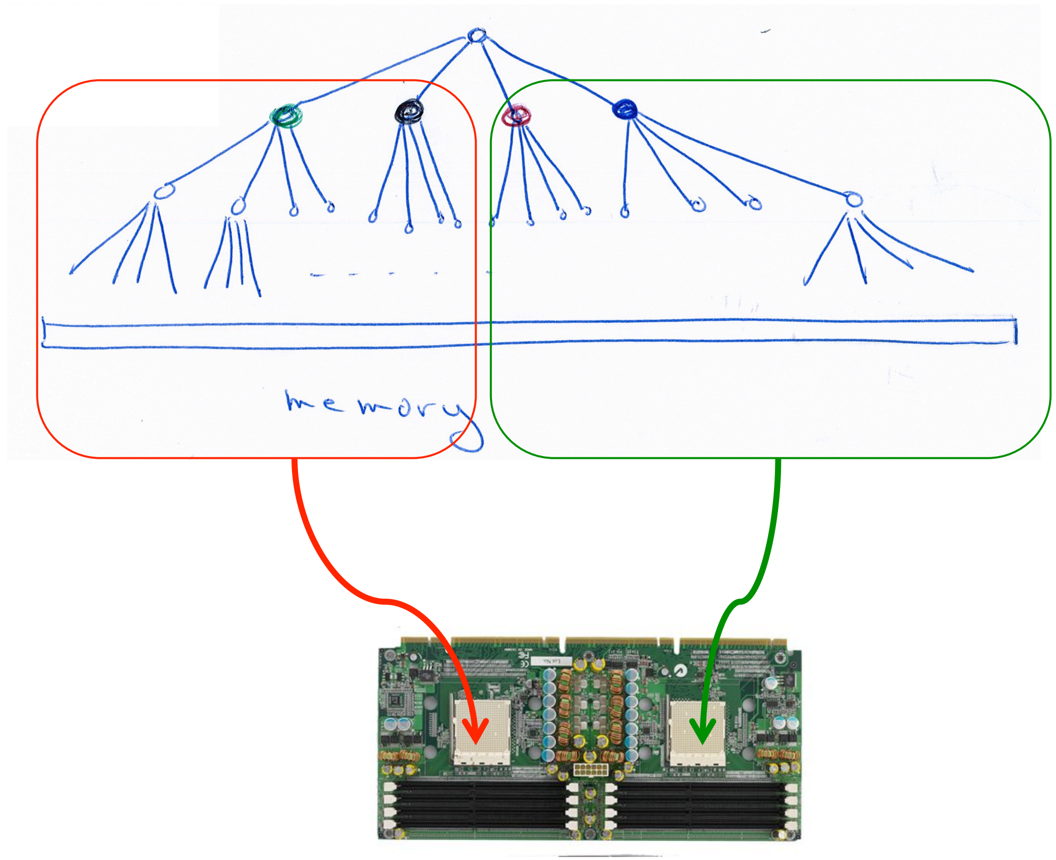
Z-curves are used by some databases data structures (trees, hash tables), or for data partitioning in numerical simulations.

Morton Curve Based CO layout and Parallelism

- How would you manage such CO layout on a NUMA node ?
- How would you implement parallel element searches ?
- What are the benefits of this CO layout (on a NUMA node) ?

Morton Curve Based CO layout and Parallelism

- Map data blocks to memory banks according to CO layout



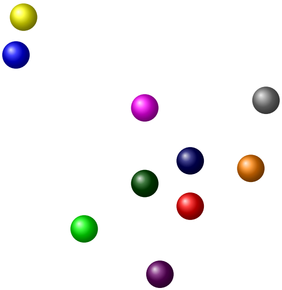
- Make sure threads access local banks first

What about GPUs ?

Packed Memory Array

A Cache Oblivious data structure for dynamics data.

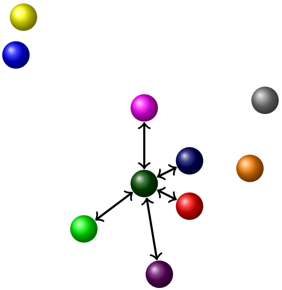
Cache Oblivious Data Structure for Moving Particles



Cache Oblivious Data Structure for Moving Particles

short-range
interaction

particles

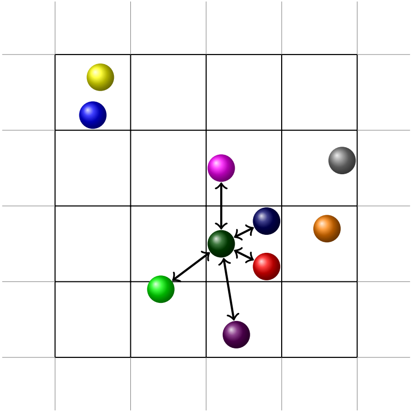


simulation

movement
$\vec{a} = \frac{1}{m} \sum \vec{F}$

Cache Oblivious Data Structure for Moving Particles

particles

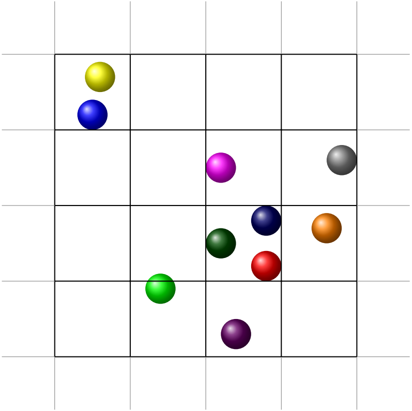


short-range
interaction

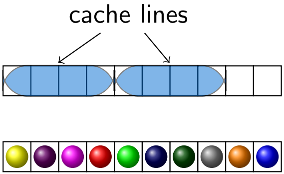
simulation

movement
$\vec{a} = \frac{1}{m} \sum \vec{F}$

Spatial Locality Preserving Memory Layout

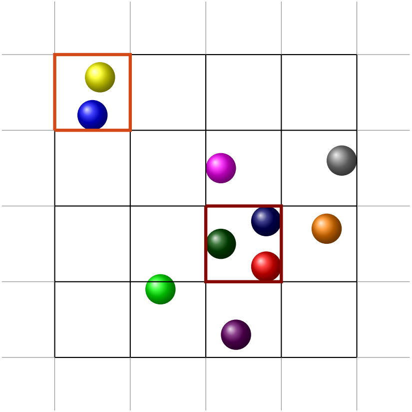


3D Space

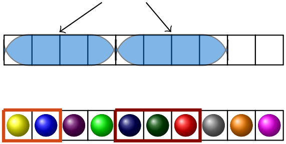


1D memory

Spatial Locality Preserving Memory Layout



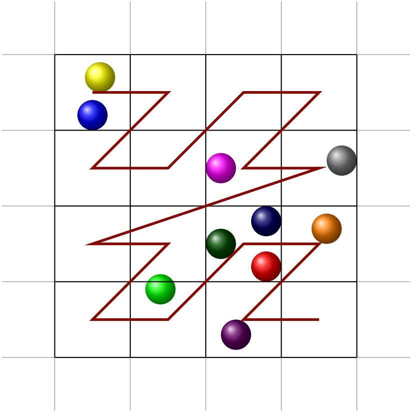
3D Space



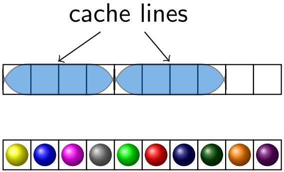
Group particles by cell.

1D memory

Spatial Locality Preserving Memory Layout



3D Space



Cell index sorting: Z-order

1D memory

Related Data Structures

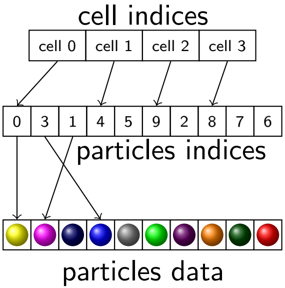
Classical approaches:

- Z-index sorting
- Compact (spatial) hashing

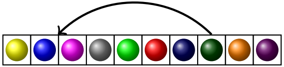
Periodically (1/100):

- sort particles data array

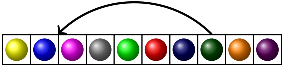
[Ihmsen et al., 2011]

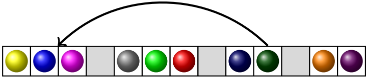


Idea



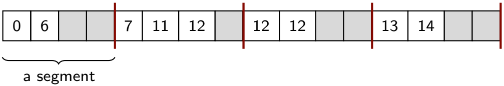
Idea





Packed Memory Array (PMA)

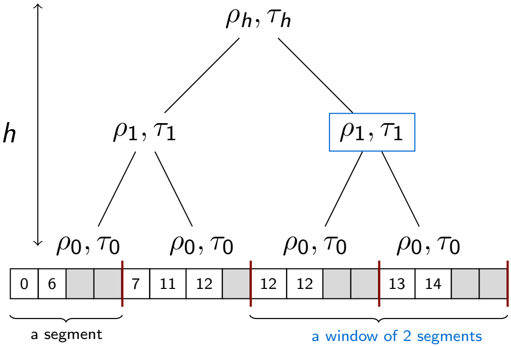
- K elements in an array of size N ($N - K$ gaps)
- Segments of size $\log N$
- #segments: power of 2



[Bender et al. (2000, 2005)]

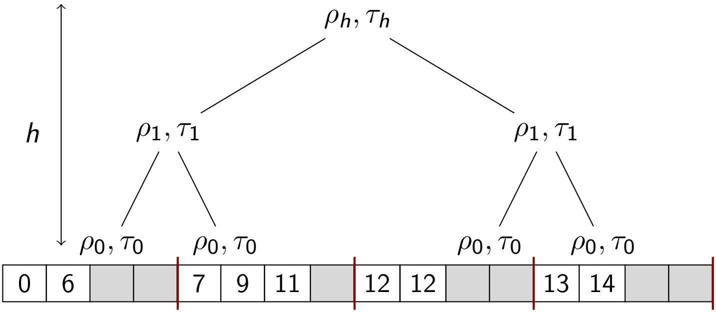
Packed Memory Array (PMA)

- Densities min ρ, max τ
- $\rho_0 < ... < \rho_h < \tau_h < ... < \tau_0$

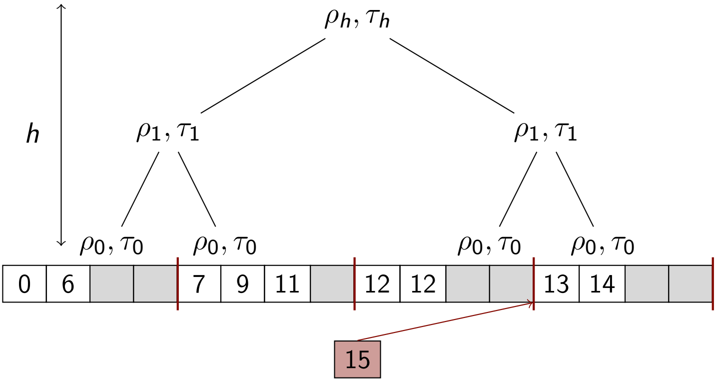


[Bender et al. (2000, 2005)]

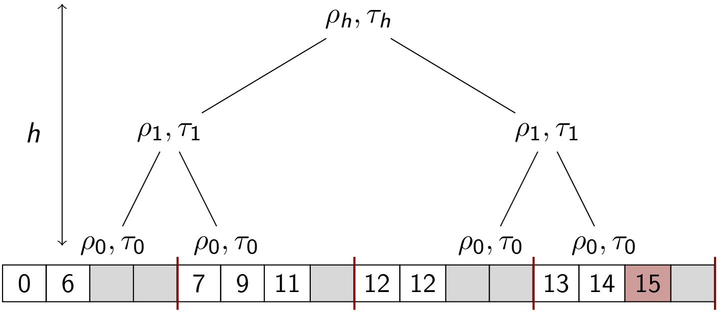
How Does the Original PMA Work ?



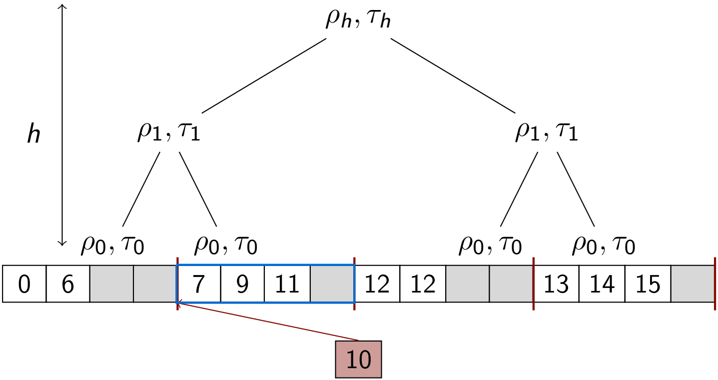
How Does the Original PMA Work ?



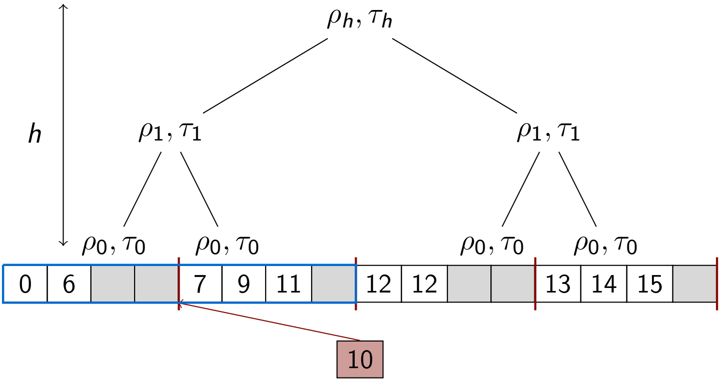
How Does the Original PMA Work ?



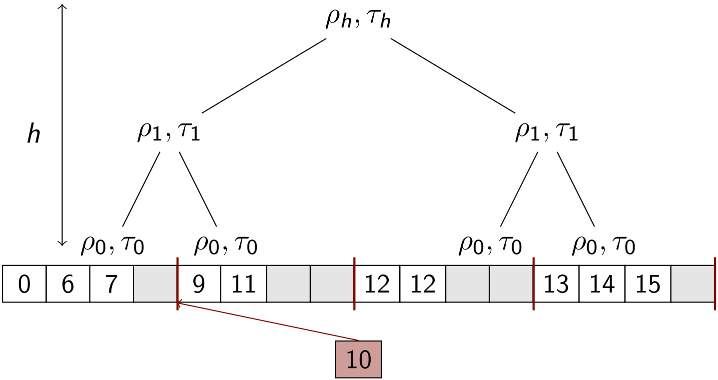
How Does the Original PMA Work ?



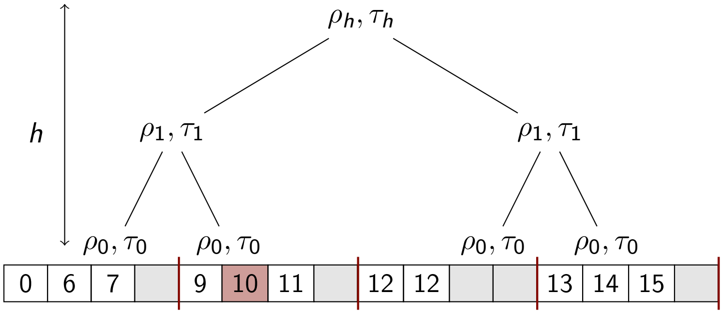
How Does the Original PMA Work ?



How Does the Original PMA Work ?

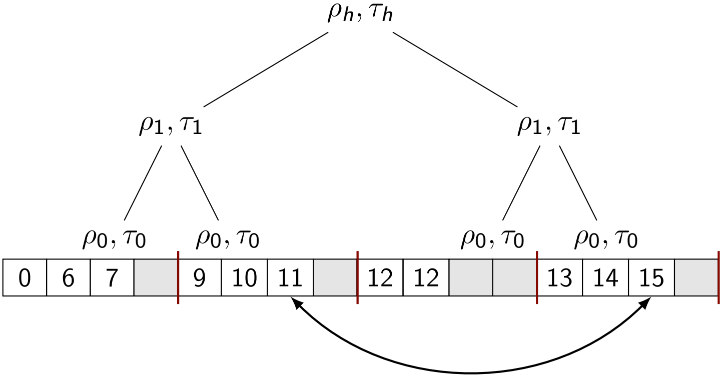


How Does the Original PMA Work ?

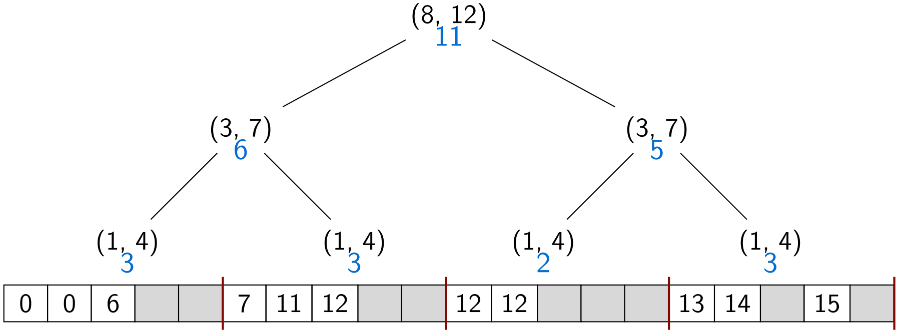


Amortized number of moves: $O(\log^2 N)$.

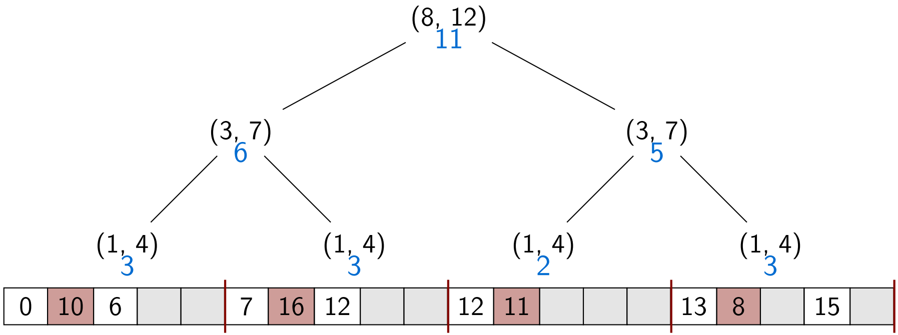
Element Moves



Applying Moves by Batches

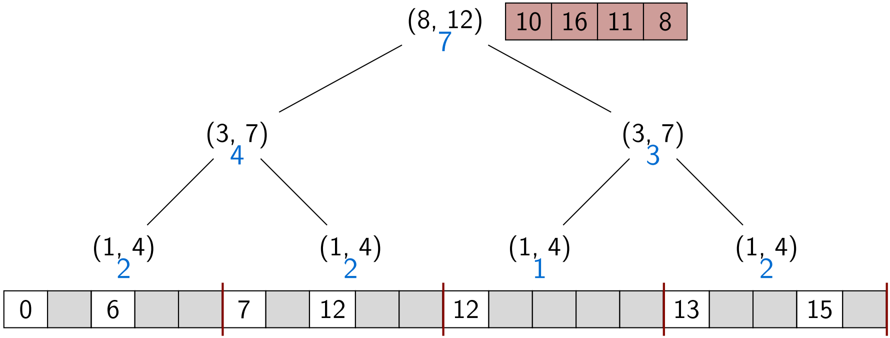


Applying Moves by Batches



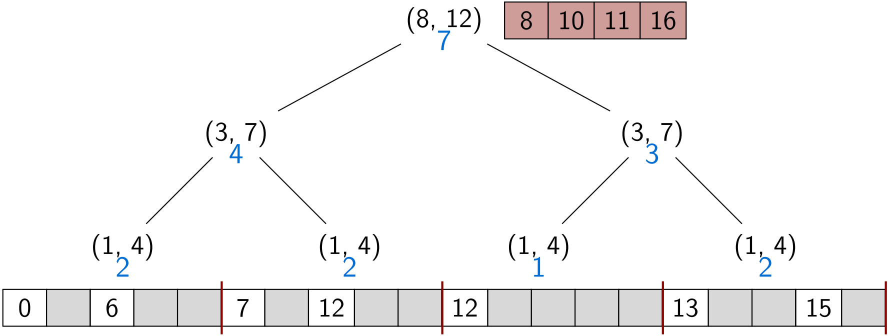
1. Some values change

Applying Moves by Batches



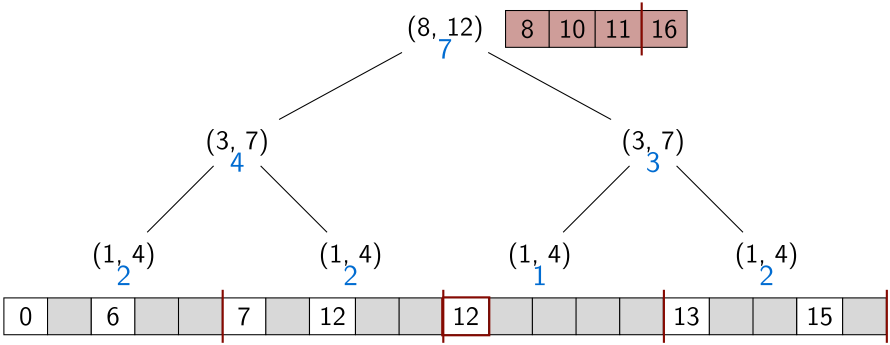
1. Some values change: gather in an array

Applying Moves by Batches



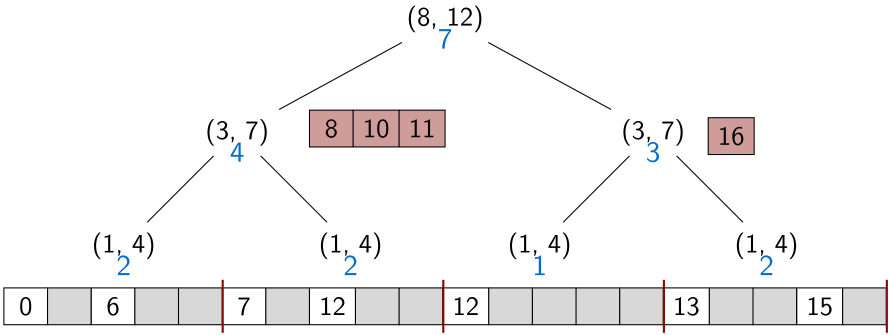
1. Some values change
2. Sort of moving elements

Applying Moves by Batches



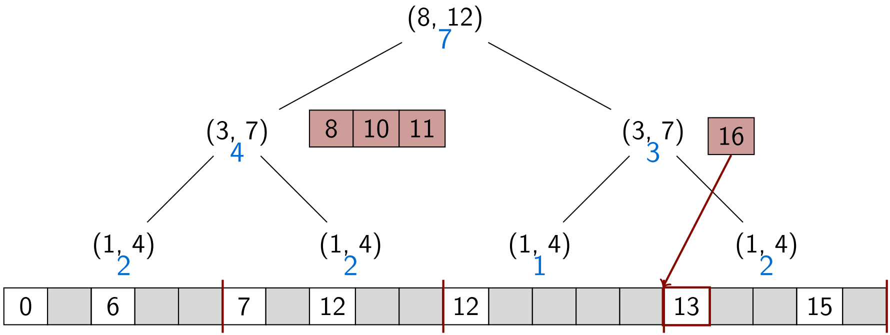
1. Some values change
2. Sort of moving elements

Applying Moves by Batches



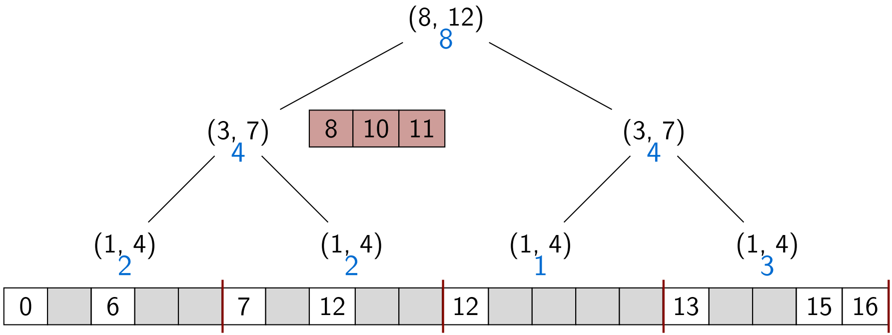
1. Some values change
2. Sort of moving elements
3. Recursively split array according to window middle value

Applying Moves by Batches



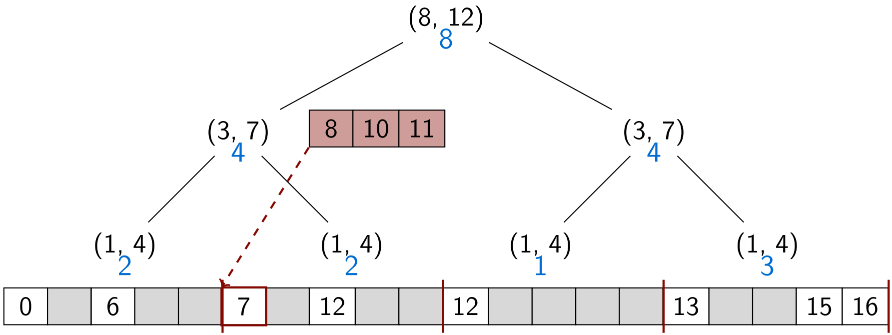
1. Some values change
2. Sort of moving elements
3. Recursively split array according to window middle value

Applying Moves by Batches



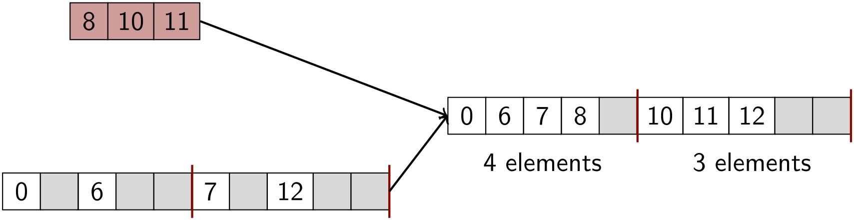
1. Some values change
2. Sort of moving elements
3. Recursively split array according to window middle value
4. Direct insertion

Applying Moves by Batches



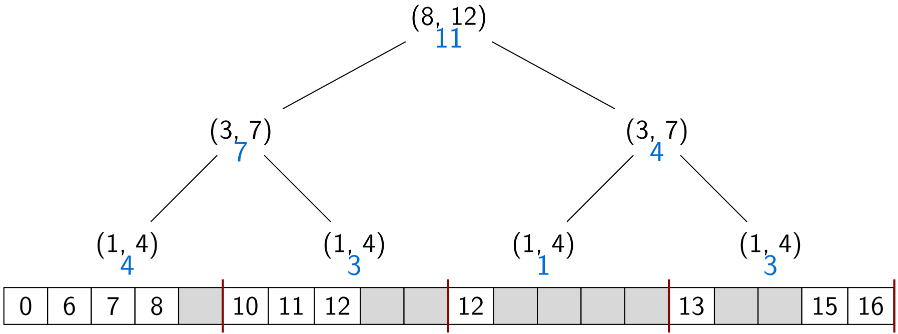
1. Some values change
2. Sort of moving elements
3. Recursively split array according to window middle value
4. Direct insertion or rebalance

Rebalance with a Single Scan



Merging two sorted lists: one scan, in place.

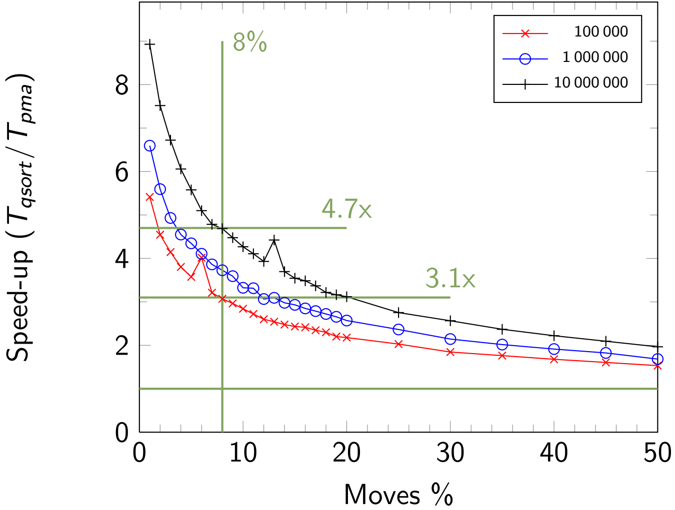
Applying Moves by Batches



Supports moves, insertions and deletions.

Experimental Results: Moving Integers

Sorting Moving Elements: Speed-up of PMA vs Qsort (Libc)



$T_{isort} = 500xT_{pma}$ with 100 000 elements and 10% of moves

Array filled with random elements. Execution time measured around several applications of a given percentage of randomly selected moves.

Scan Performance

Dense array:

```
for i in 1 to K do
    sum += a[i]
```

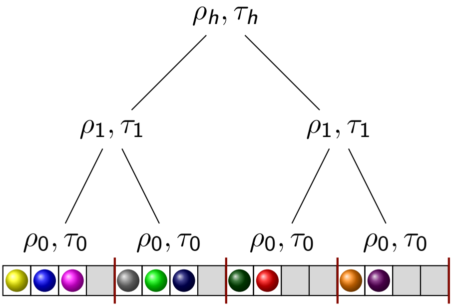
PMA:

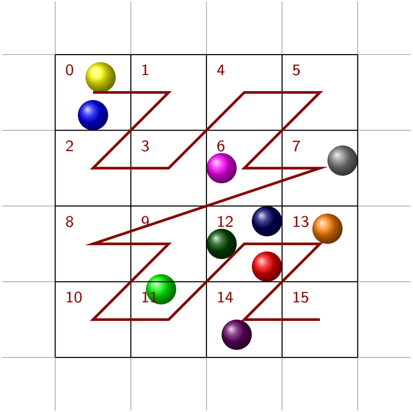
```
for i in 1 to N do
    if isValid(a[i])
        sum += a[i]
```

K	N	N/K	T_{PMA}/T_{array}
100 000	163 840	1.64	1.69
1 000 000	1 572 864	1.57	1.86
2 900 000	4 456 448	1.54	1.74
10 000 000	15 728 640	1.57	1.78

- Test case built to exacerbate the overhead. On realistic computation schemes it fades away.

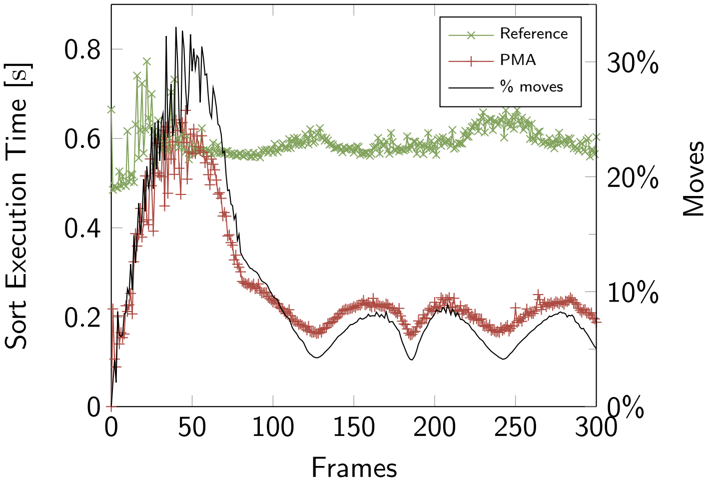
Application to Particles





Application to Particles: Results

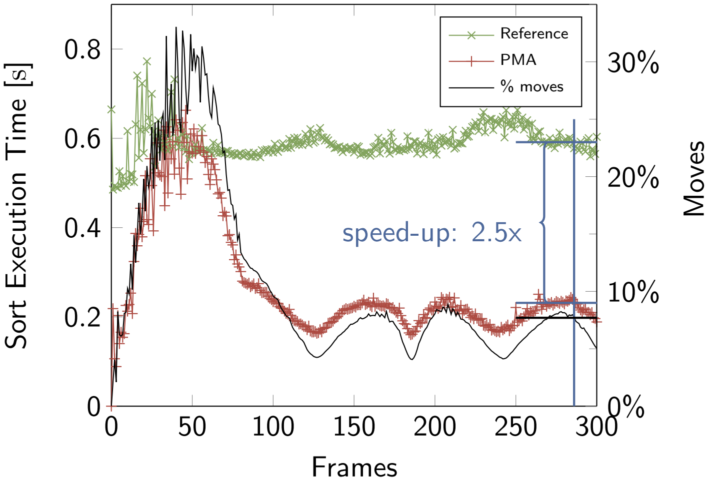
Implementation in Fluids [Hoetzlein, 2008]: $2.9\,10^6$particles



Global performance: 2.8% (sort is 4.5% of total simulation time).

Application to Particles: Results

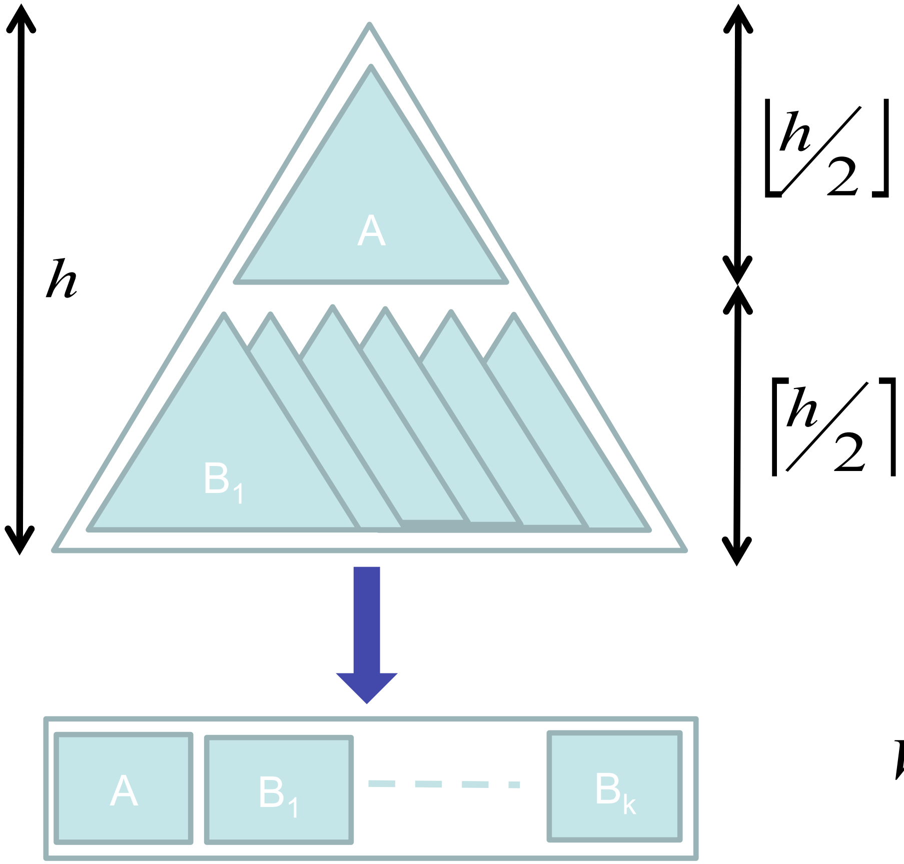
Implementation in Fluids [Hoetzlein, 2008]: $2.9\,10^6$ particles

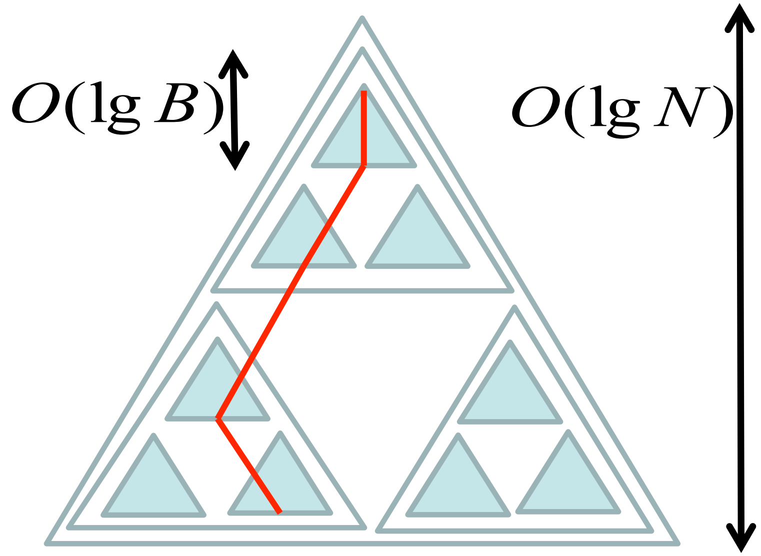


Global performance: 2.8% (sort is 4.5% of total simulation time).

Searching in the CO model

Binary tree mapped in memory using a recursive layout [Bender et al 2000]



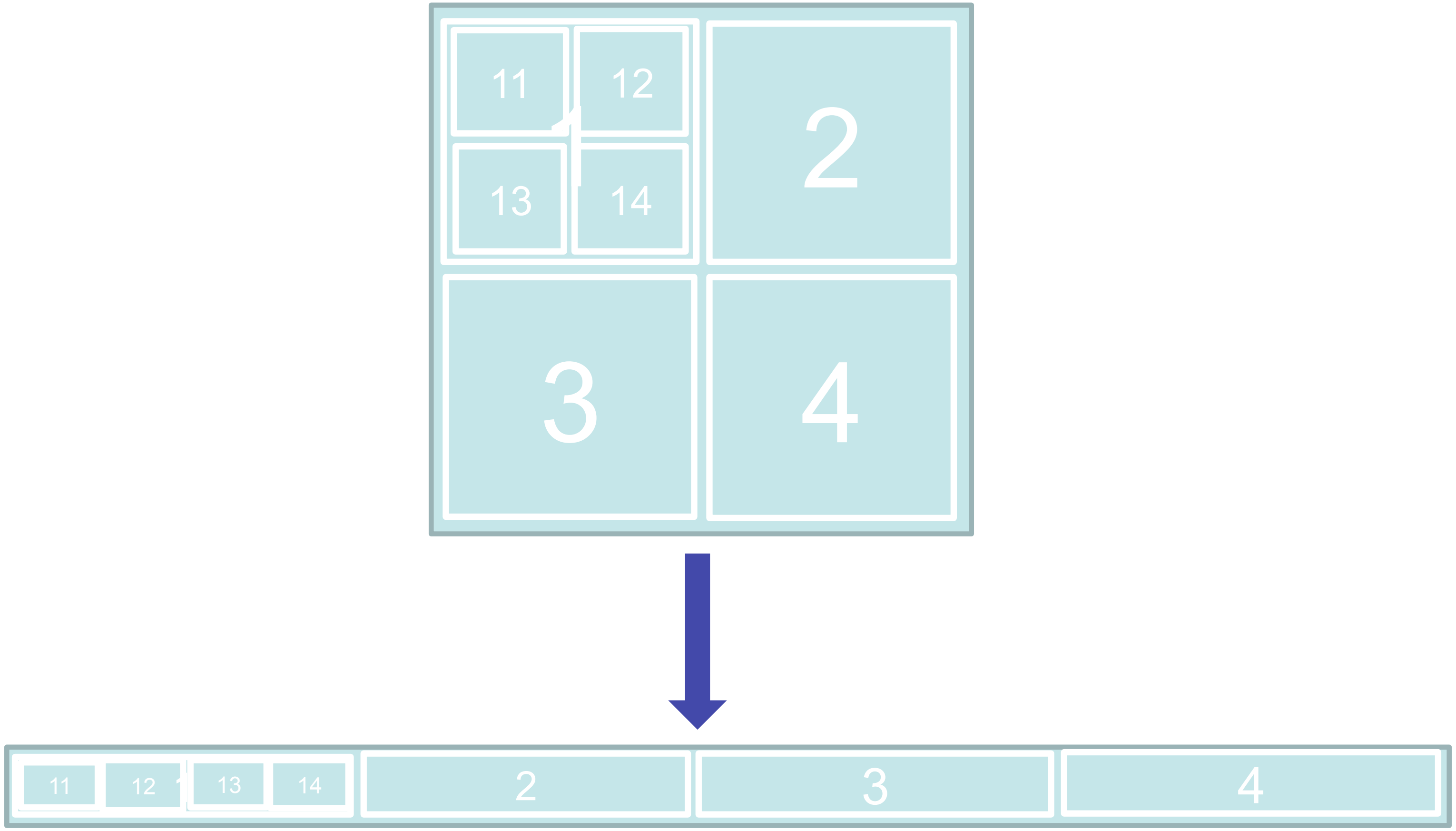


$$W(N) = O(\lg N)$$

$$Q(N) = O(1).\frac{O(\lg N)}{O(\lg B)} = O(\log_B N)$$

Multiplying in the CO model

D&C matrix multiplication using a recursive layout



Multiplying in the CO model

D&C matrix multiplication using a recursive layout

$$W(N) = \begin{cases} 8W\left(N/2\right) + O\left(N^2\right) & \text{if } N > 1 \\ O(1) & \text{otherwise} \end{cases}$$

$$W(N) = O\left(N^3\right)$$

$$Q(N) = \begin{cases} 8Q\left(N/2\right) + O\left(N^2/B\right) & \text{if } N^2 > M/3 \\ O\left(N^2/B\right) & \text{otherwise} \end{cases}$$

$$Q(N) = O\left(N^3/B\sqrt{M}\right)$$