Cache-Oblivious Algorithms and Datastructures

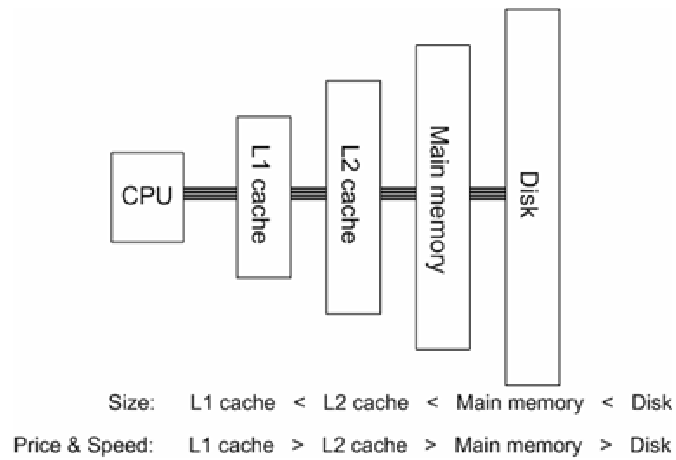
- What is Cache-Obliviousness?
- External-Memory Model (EMM)
- Cache-Oblivious Model (COM)
- Scanning
  - Ordinary array
- Searching
  - Height Partitioning Tree
- Sorting
  - K-funnel

What is cache-obliviousness?

Definition [Prokop:1999]:
An algorithm is cache-oblivious if no program variables dependent on hardware parameters, such as cache size and cache-line length, need to be tuned to minimize the number of cache misses.

What does this mean and why is it interesting?

Memory Hierarchy

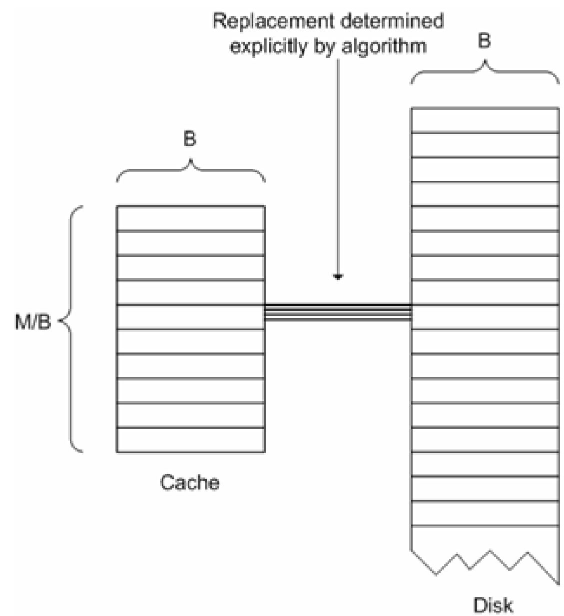


Principle of locality:

- Temporal locality
- Spacial locality

External-Memory Model

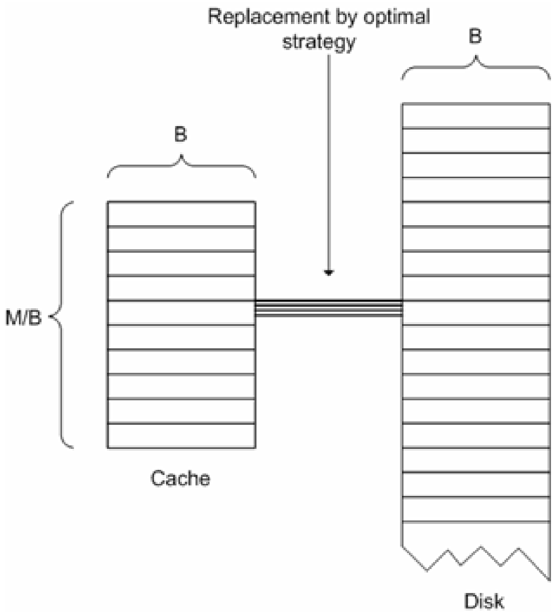
- Replacement programmed explicitly
- M and B are known



N = Problem size
M = Cache size
B = Block size

Cache-Oblivious Model

- Optimal replacement strategy assumed
- Fully associative
- Tall-Cache $M = \Omega(B^2)$



N = Problem size
M = Cache size
B = Block size

Cache-Oblivious Model

- Is the model realistic?

Optimal replacement strategy: *If an algorithm makes T memory transfers on a cache of size M with optimal replacement, then it makes at most 2T memory transfers on a cache of size M/2 with LRU or FIFO replacement.*

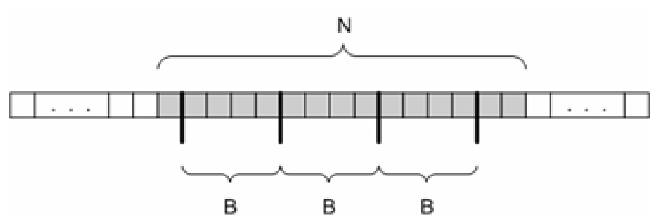
Cache-Oblivious Model

Why is the COM appealing?

- Clean
- Multi-level memory hierarchies
- Self-tuning
- Easier to program

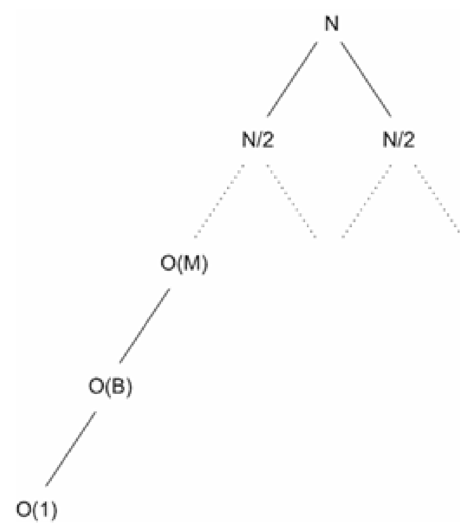
Scanning

- Scanning N elements stored in a contiguous segment of memory (array) costs at most $\lceil N/B \rceil + 1$ memory transfers.

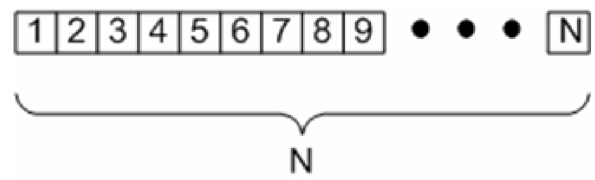


Divide and conquer

- Usually we consider O(1) as base case
- In COM we often consider O(M) or O(B)



Binary Search – Divide and Conquer



Binary search on a ordinary sorted array incurs $\Theta(\log_2 N - \log_2 B)$ memory transfers.

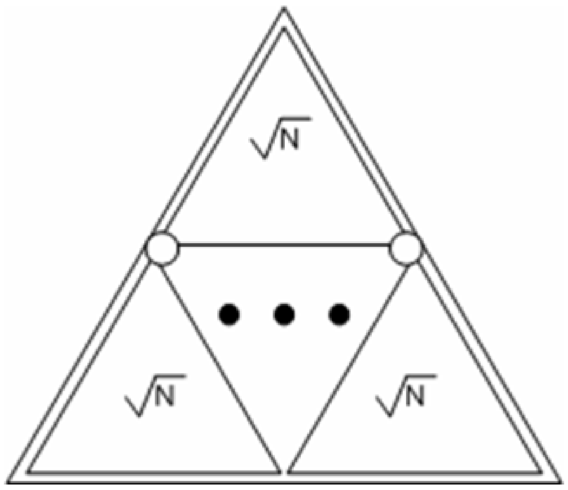
$$T(N) = T(N/2) + O(1)$$

Base case: $T(O(B)) = O(1)$

This can be done more efficiently using the Height Partitioning Layout

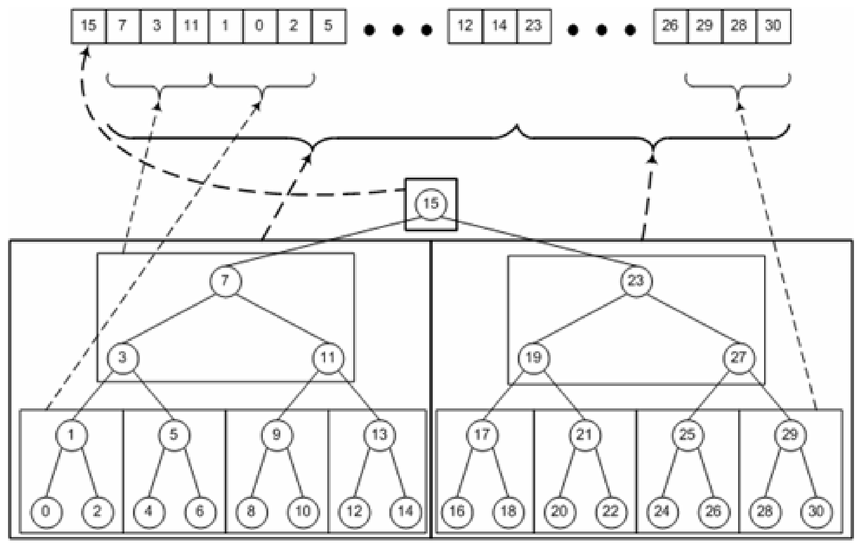
Cache-Oblivious Search

- Construct a complete binary tree with N nodes storing the N elements in search-tree order.
- Store the tree sequentially in memory according to a recursive Height Partitioning Layout
- Search now done in $2*(1+(\log_2 N)/(\log_2 B/2)) = 2 + 4\log_B N$ transfers, which is optimal



Cache-Oblivious Search

Height Partitioning Tree

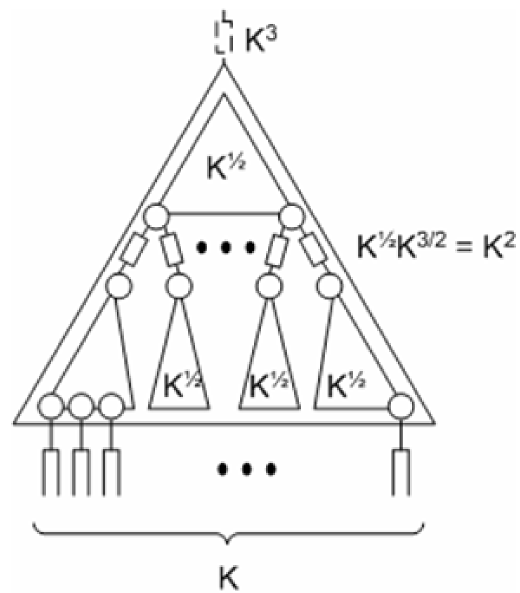


Sorting

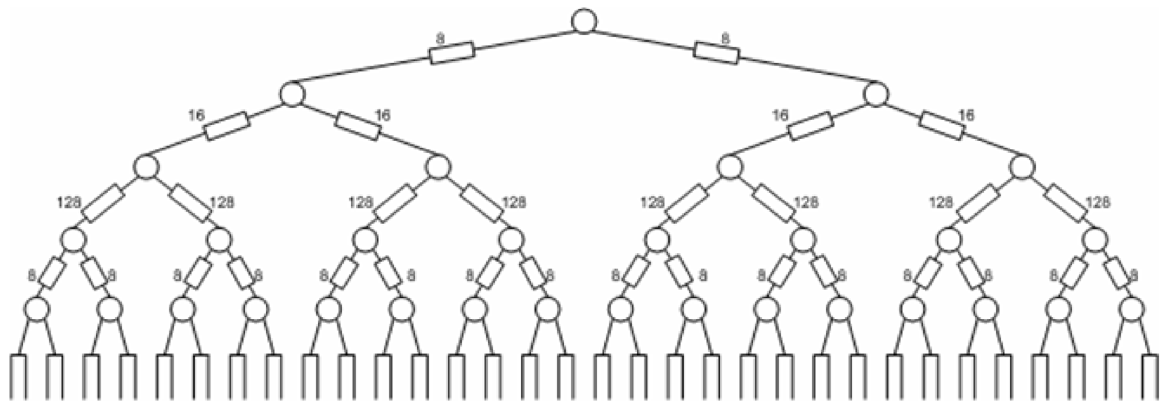
- Memory transfers of M/B-way mergesort in EMM bounded by
$$\Theta((N/B)\log_{M/B}(N/B))$$
- Memory transfers of 2-way mergesort is given by
$$T(N) = 2T(N/2) + \Theta(N/B)$$
with solution
$$T(N) = \Theta(N/B \log_2 N/B)$$
- Can we change $\log_2$ to $\log_{M/B}$?
- Yes! Using the K-funnel

K-funnel

- Complete binary tree with buffers on edges
- Buffer sizes are defined recursively
- A K-funnel merges K sorted lists of total size K^3 in $O((K^3/B)\log_{M/B}(K^3/B)+K)$ memory transfers
- The K-funnel occupies K^2 space



32-funnel



Funnelsort

1. Split the array into $K = N^{1/3}$ contigous segments each of size $N/K = N^{2/3}$
2. Recursively sort each segment
3. Apply the K-funnel to merge the sorted segments

Memory transfers are made in 2 and 3 leading to

$$T(N) = N^{1/3}T(N^{2/3}) + O((N/B)\log_{M/B}(N/B) + N^{1/3})$$

with solution $O((N/B)\log_{M/B}(N/B))$

Selected References

M.A. Bender, R. Cole, and R. Raman. Exponential structures for efficient cache-oblivious algorithms. In *Proc. 29th International Colloquium on Automata, Languages, and Programming (ICALP)*, 2002.

M.A. Bender, E. Demaine, and M. Farach-Colton. Cache-oblivious B-trees. In *Proc. 41st Ann. Symp. On Foundations of Computer Science*, pages 399-409, 2000.

G.S. Brodal & R. Fagerberg. Cache-oblivious distribution sweeping. Technical Report RS-02-18, BRICS, Dept. Of Computer Science, University of Aarhus, 2002.

H. Prokop. Cache-oblivious algorithms. Master's thesis, Massachusetts Institute of Technology, June 1999.

J.S. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, 33(2):209-271, June 2001.