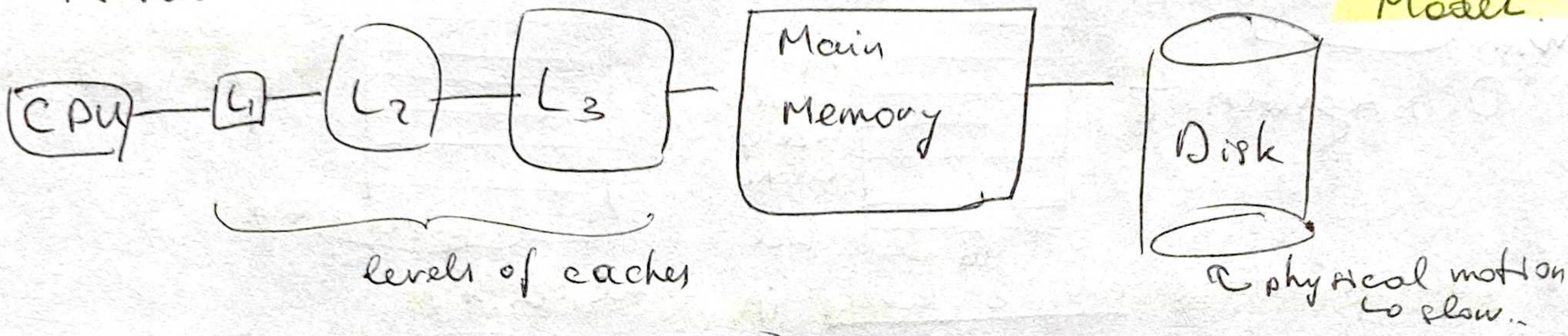
Cache-oblivious algorithms

Modern Memory hierarchy

Random access model : access in memory at $O(1)$. Different Model.



1) slower [redacted] (higher latency)
2) and bigger space
3) bandwith should be same

the larger space, the more time to access speed of light as fundamental limit.

Cost to Access

$$= \underbrace{\text{latency}}_{} + \frac{\text{amount of data}}{\text{bandwidth}}$$

limited by speed of light

rate at which can get data out, assume const/fixed

Idea: as latency goes up, increase the amount of data, then the amortized cost to access an element (fixed bandwidth) goes down

$$\text{Amortized cost to access one element} = \frac{\text{latency}}{\text{amount}} + \frac{1}{\text{Bandwidth}}$$

latency amortized by amount / size of block

Spatial locality

Want algorithms to use all elements in a block (after getting it from disk into memory, slow to faster)

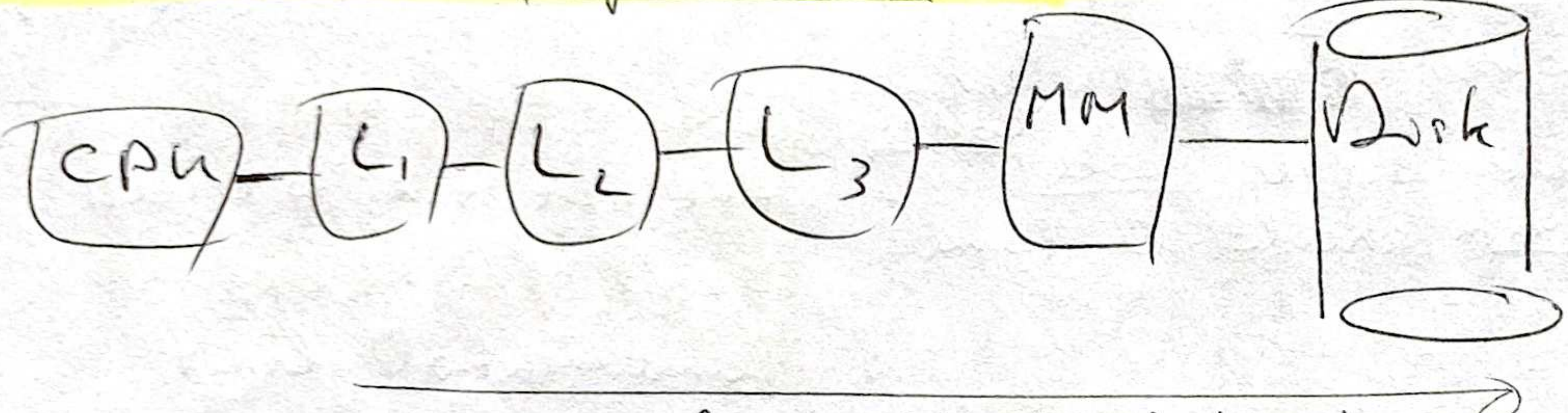
Temporal locality

Want ideally to reuse blocks (if algorithm is above linear will use elements in input more than once)

(cache-aware model)

Two-level[redacted] memory model

can represent any 2 levels in (incl. Disk)



slower (higher latency)
more space
bandwidth same

Assume CPU can access cache instanteneously (free)
If CPU needs data, check cache if there get it free,
else get entire block from main memory,
potentially need to [redacted]remove data from cache
and write it back to main memory to free up cache.

- accesses to cache free (but still measure computation time)
- count block memory transfers between cache and main memory

memory transfer: read or write a block from/into main memory

$$MT(N) \; (= MT_{B,M}(N))$$

size of problem (the only variable that can change → cannot recurse [redacted] on Block size or total cache size (fixed parameters)

block size, total size cache } fixed parameters

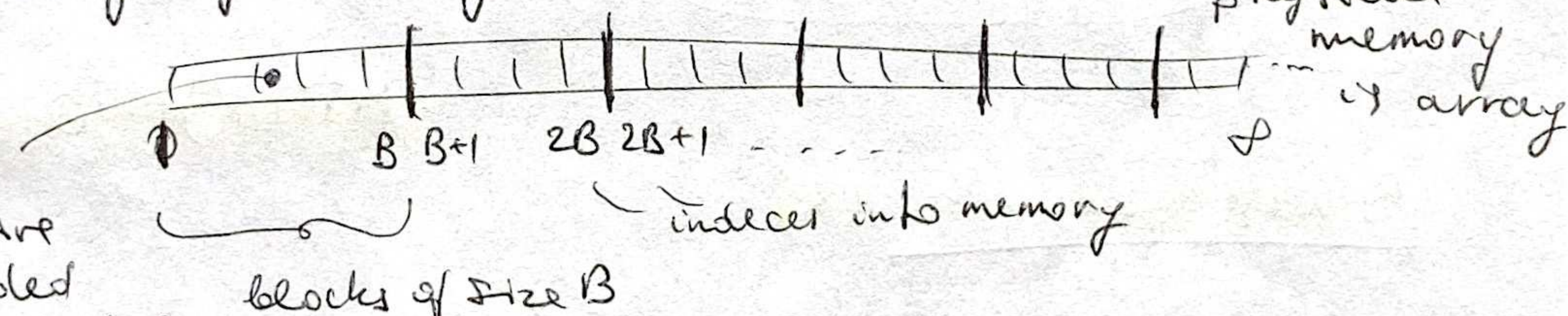
Cache aware algorithm: B trees (know fixed cache parameters)

Cache-oblivious motivation:

1) want to design algorithms that perform well no matter the values of B and M
2) most "just memory" assuming algorithms are already cache-oblivious. Some will do well in the two-level memory model and some will not.

Cache-oblivious algorithm:

- algorithm does not know B & M
- accessing element automatically fetch block containing it & evicts block that will be used furthest in future
- blocking of memory (a linear array)

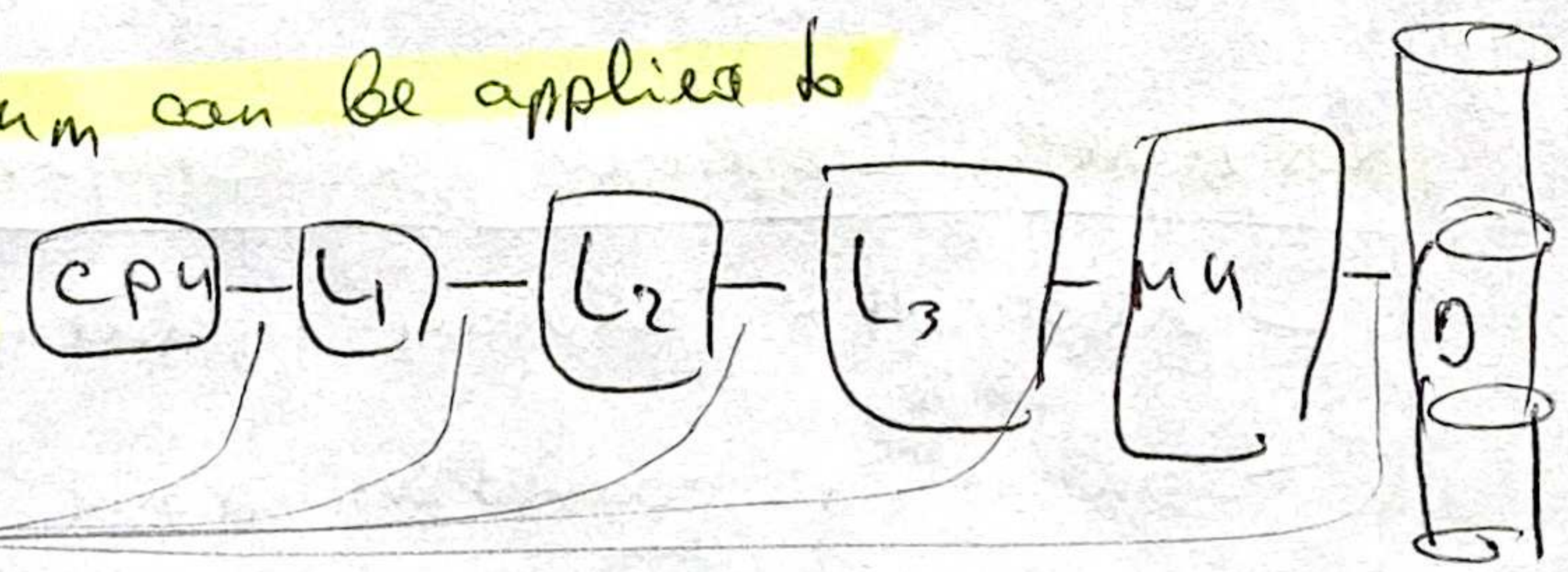


access ⇒ get entire block rounded down to prev. multiple of B, and rounded up to the next multiple of B

Fact:

[if efficient on 2 levels & cache oblivious ⟹ efficient on l levels]

Efficient cache oblivious algorithm can be applied to any adj. pair of levels in (i.e. any memory transfer level):

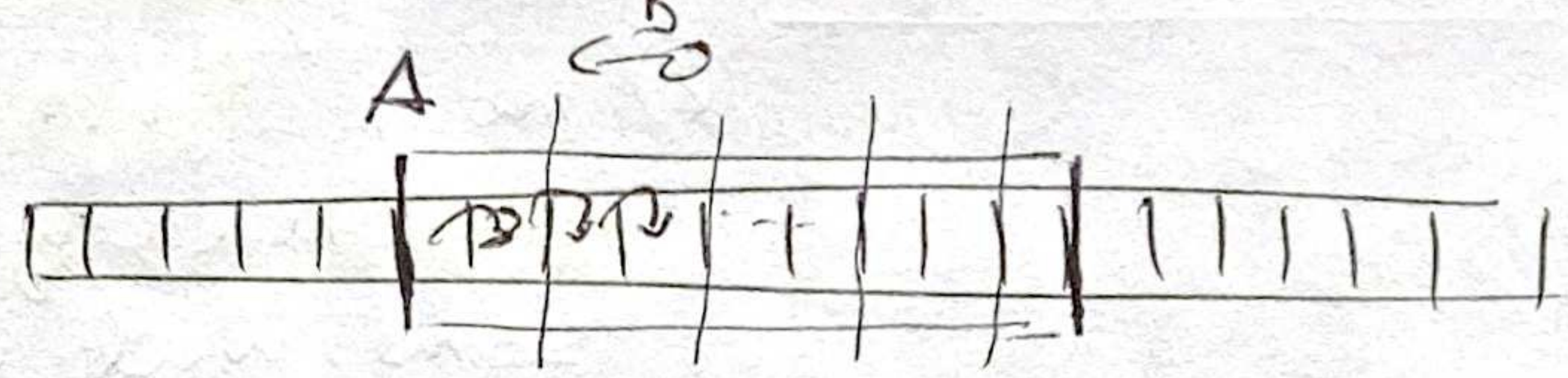


Because cache-oblivious algs do not tune to particular values of B & M.

Basic algorithms

Scanning (A, N) // visiting items in array in order
e.g. sum array

for $i \leftarrow 1$ to N
  do visit $A[i]$



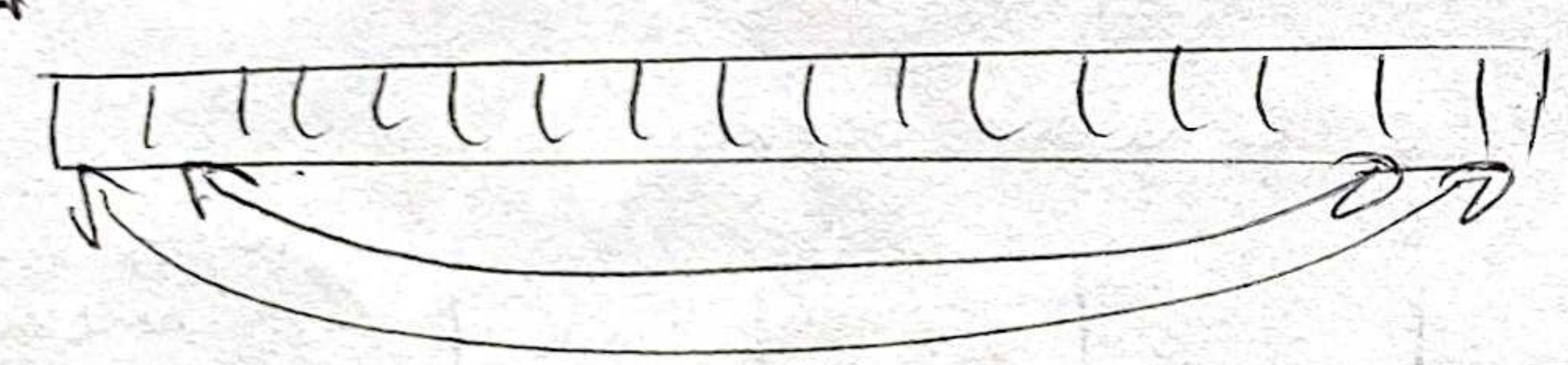
$$MT(N) = O\left(\frac{N}{B} + 1\right)$$

N could be < B, more precisely should be 2 since first and last block can be non-full

O(1) parallel scans

Reverse (A, N)

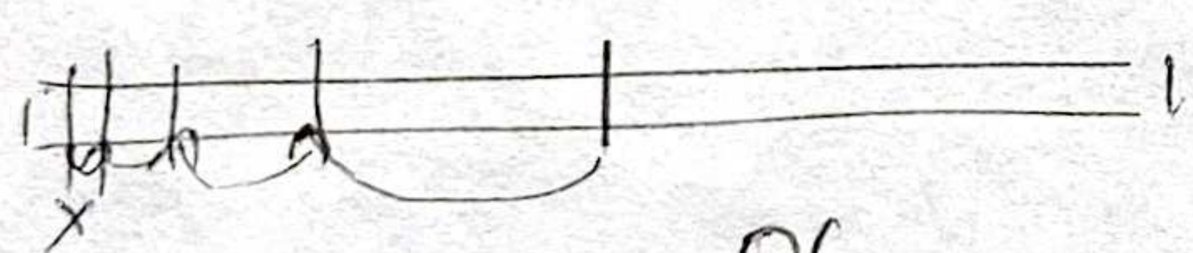
for $i \leftarrow 1$ to $\lfloor N/2 \rfloor$
  do exchange $A[i] \leftrightarrow A[N-i+1]$



Assuming $\frac{M}{B} \geq 2$ (cache can store at least 2 blocks)

$$MT(N) = O\left(\frac{N}{B} + 1\right)$$

Binary Search (x)



if x equals to compared values early
O( )

hope: $\log_B N$ without knowing B

$$MT(N) = O\left(\lg\left(\frac{N}{B} + 1\right)\right) = O(\lg N - \lg B + 1) \Big\} \text{ bad}$$

once search narrows down to 1 block

Divide & Conquer algorithms (incl. Binary search)

- algorithm divides problem down to O(1) size
- analysis considers point at which:

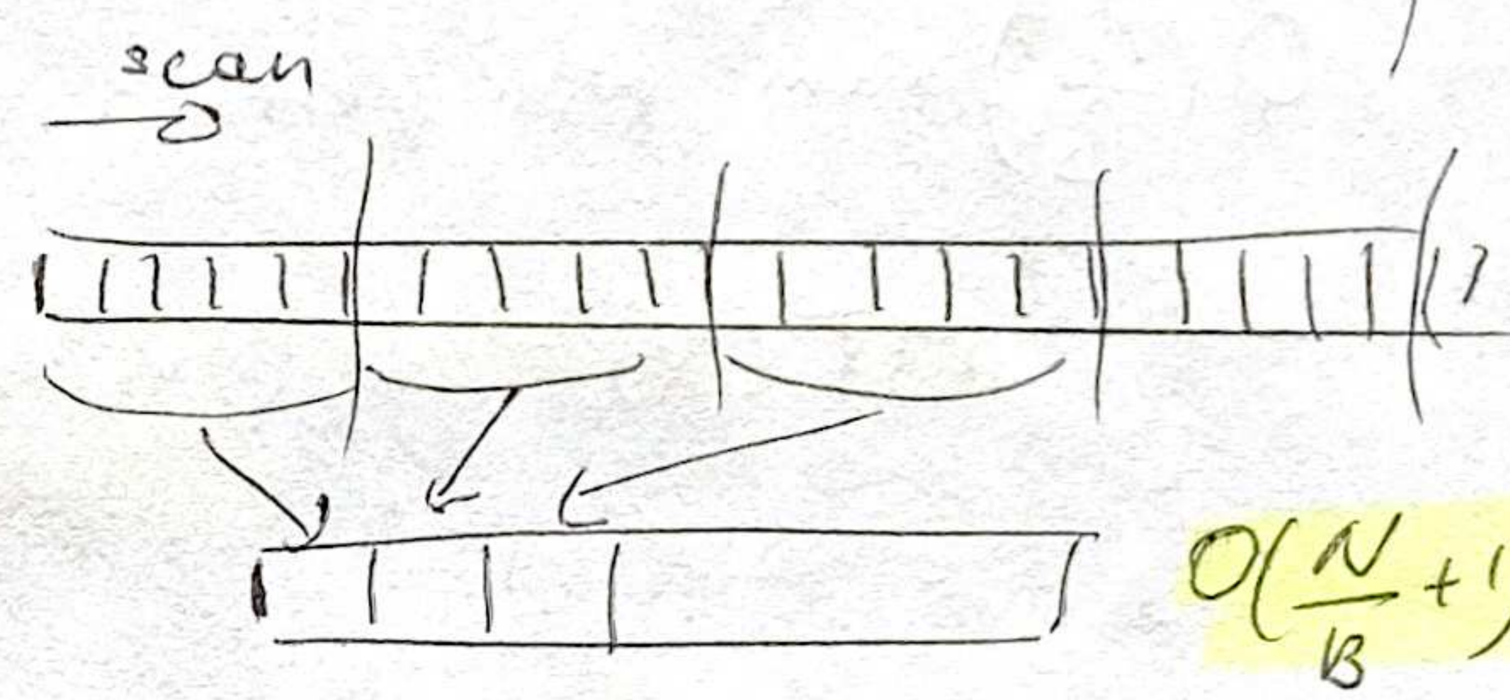
change base case of recurrence
- problem fits in cache ($\leq M$)
- problem fits in O(1) blocks

linear time worst case

Order statistics (median)

1. conceptually partition array into $\frac{N}{5}$ 5-tuples
2. compute median of each tuple
3. recursively compute median of these medians x $\}\ MT\left(\frac{N}{5}\right)$
4. partition around x
5. recurse on one side

$MT\left(\frac{3}{4}N\right)$

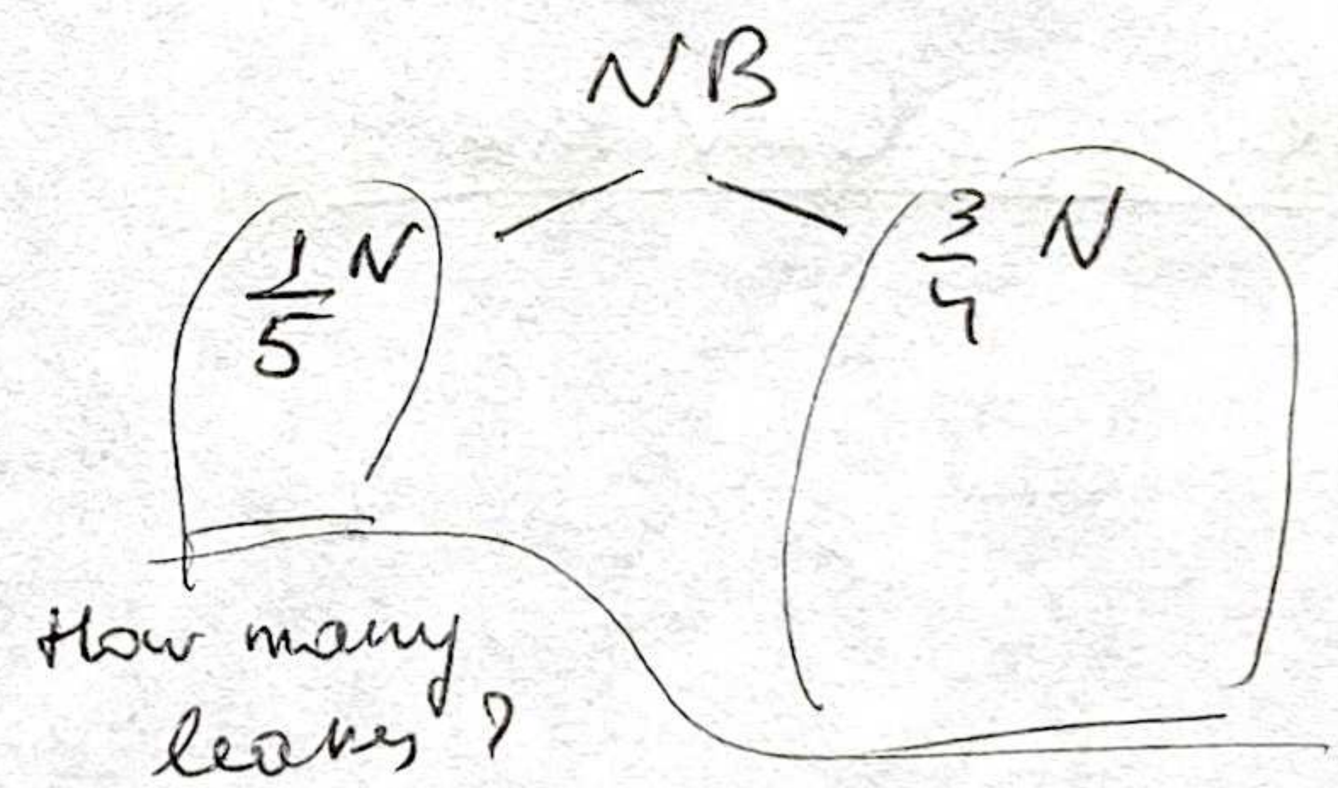


$O\left(\frac{N}{B}+1\right)$

Analysis

$$MT(N) = MT\left(\frac{N}{5}\right) + MT\left(\frac{3}{4}N\right) + O\left(\frac{N}{B}+1\right)$$

$MT(1) \Leftarrow$ bad



leaves: $L(N) = L\left(\frac{N}{5}\right) + L\left(\frac{3}{4}N\right)$

$L(1) = 1$

$$N^{\alpha} = \left(\frac{N}{5}\right)^{\alpha} + \left(\frac{3}{4}N\right)^{\alpha}$$

$$1 = \left(\frac{1}{5}\right)^{\alpha} + \left(\frac{3}{4}\right)^{\alpha} \quad \alpha \approx 0.8398$$

$$\Rightarrow L(N) = \left(\frac{N}{5}\right)^{\alpha} + \left(\frac{3}{4}N\right)^{\alpha} = \omega\left(\frac{N}{B}\right)$$ bad

$MT(B) = O(1)$

leaves $= \left(\frac{N}{B}\right)^{\alpha} = o\left(\frac{N}{B}\right)$

cost roughly geometric down the tree

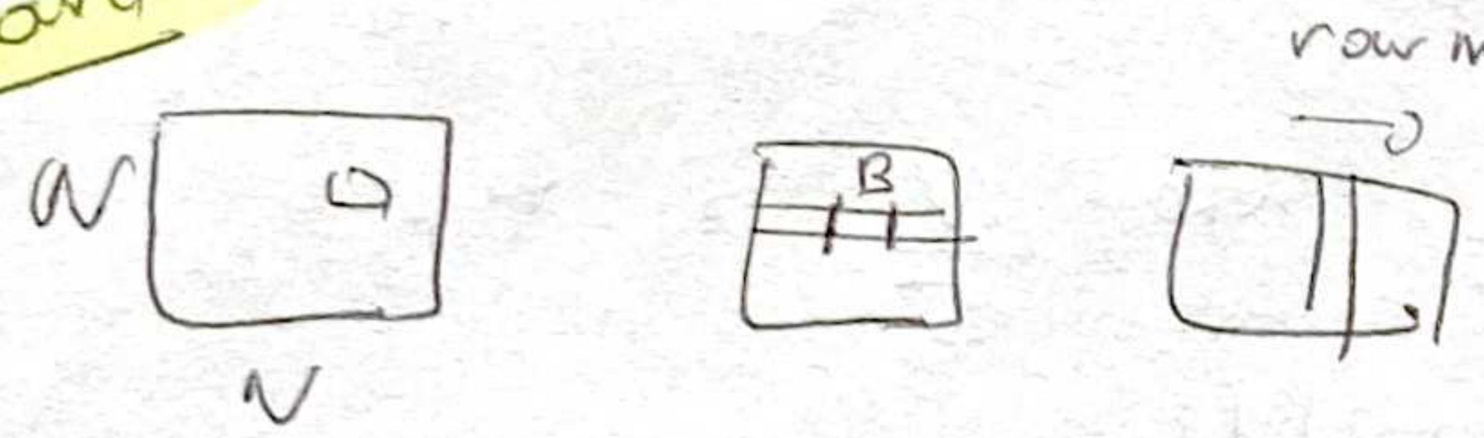
$\Rightarrow$ root dominates

$$\Rightarrow MT(N) = O\left(\frac{N}{B}\right)$$

can't do better because need to read all N.

Matrix Multiplication:

standard



$C = A \cdot B$

Memory layout

assume C stored in row-major

A row-major

B col-major

$O\left(\frac{N}{B}\right)$ mem. transfers to compute c_{ij}

$\Rightarrow MT(N) = O\left(\frac{N^3}{B}\right)$

col [redacted] in B moves for each value [redacted] in A's row segment

[redacted] but want closer to $O\left(\frac{N^2}{B}\right)$

under memory layout assumption we have
good spatial locality
bad temporal locality

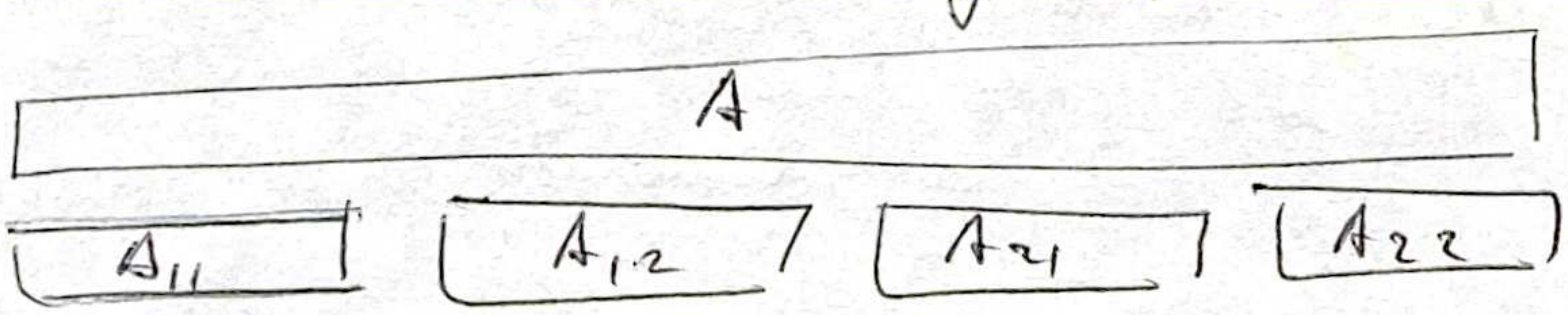
assume one row fits in cache
then for each row in C
must [redacted] transfer a row in A [redacted]
and entire B. [redacted]

Block algorithm

$$\begin{array}{c|c|c|c} & \frac{n}{2} & n & \frac{n}{2} \\ \hline \frac{n}{2} & C_{11} & & C_{12} \\ n & & & \\ \frac{n}{2} & C_{21} & & C_{22} \end{array} = \begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array} \cdot \begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array} =$$

$$= \begin{array}{c|c} A_{11}B_{11} & A_{11}B_{12} \\ \hline A_{21}B_{11} & A_{21}B_{12} \end{array} + \begin{array}{c|c} A_{12}B_{21} & A_{12}B_{22} \\ \hline A_{22}B_{21} & A_{22}B_{22} \end{array}$$

Store matrices recursively by block in memory.



same layout for C and B

recursively in memory
in one array of size N

← powerful idea in cache-oblivious algorithms

$$MT(N) = \underbrace{8\,MT\left(\frac{N}{2}\right)}_{\text{relies on recursive memory layout}} + O\left(\frac{N^2}{B}\right)$$

scan same consecutive order for A, B, C

$MT(B) = O(1) \leftarrow$ bad

$$MT(c\sqrt{M}) = O\left(\frac{M}{B}\right)$$

N^2/B

$\left(\frac{N}{2}\right)^2/B \quad \} \quad \frac{2N^2}{B}$

increase

leaves matter

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

$\underbrace{\qquad\qquad}_{O\left(\left(\frac{N}{\sqrt{M}}\right)^3\right) \text{ leaves}}$

$$MT(N) \overset{\Rightarrow}{=} O\left(\frac{N^3}{M^{3/2}} \cdot \frac{M}{B}\right) = O\left(\frac{N^3}{B\sqrt{M}}\right)$$

Optimal! in the two-level memory model.