

Applied Algorithms

CSCI-B505 / INFO-I500

Lecture 3.

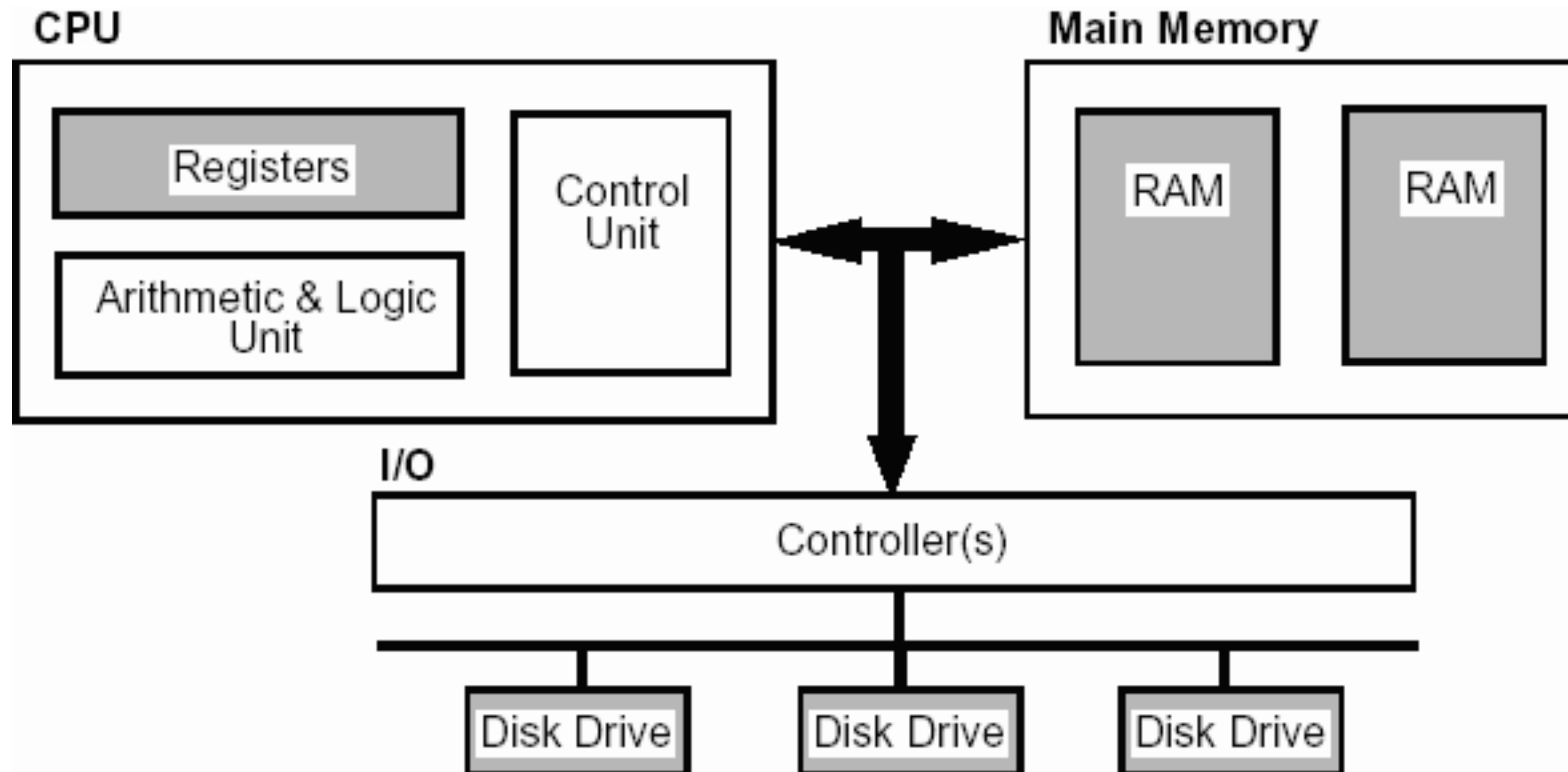
Review of Basic Data Structures - 1

M. Oguzhan Kulekci

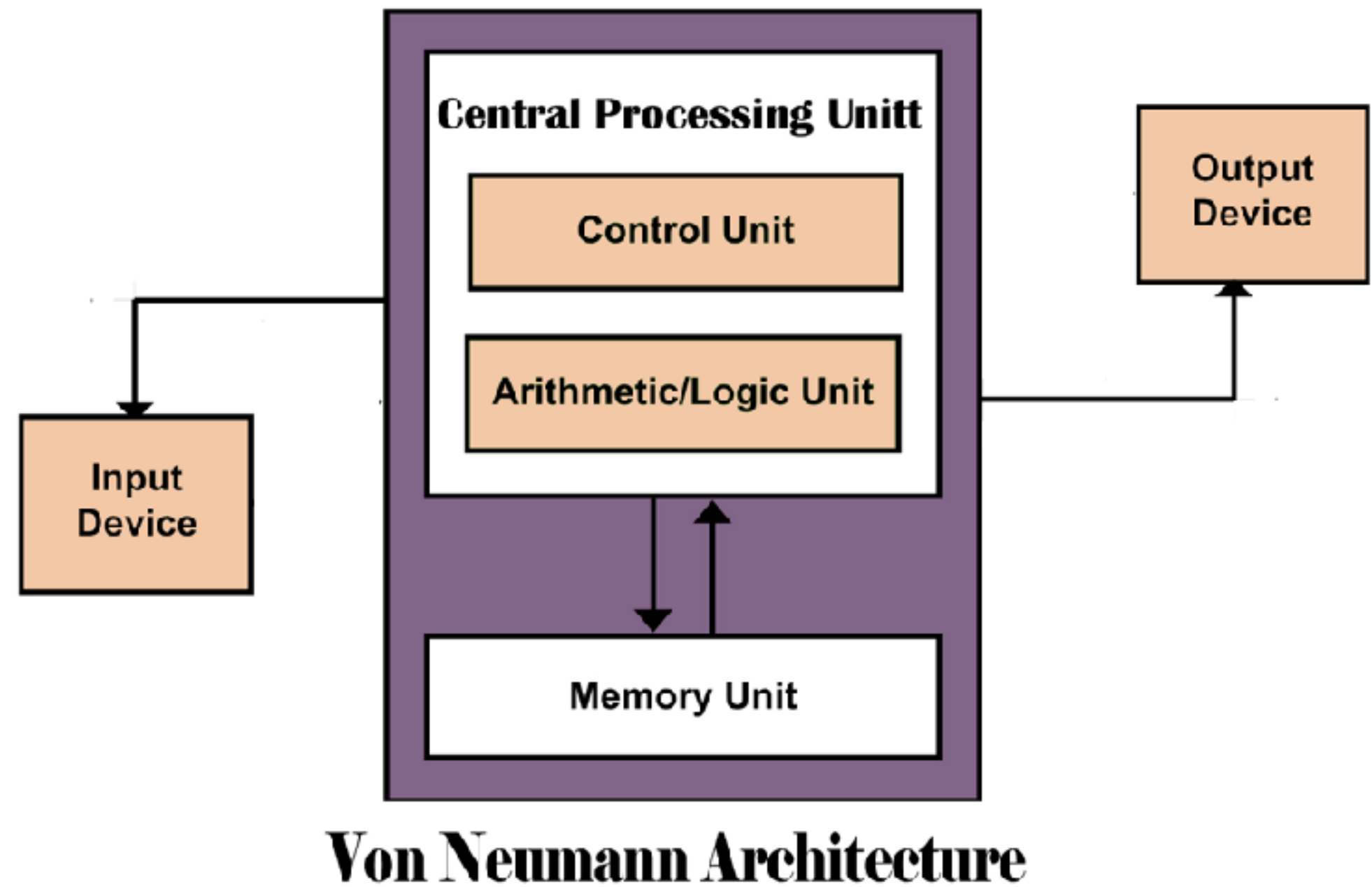
- An overview of the basic memory architecture
- Locality of reference principle
- Arrays
 - *Range-Minimum-Queries*
- Linked Lists

Computing Model

- The architecture of a machine that executes the software (= DS + Algorithms, you know:).
- Specs are important while devising an algorithm !



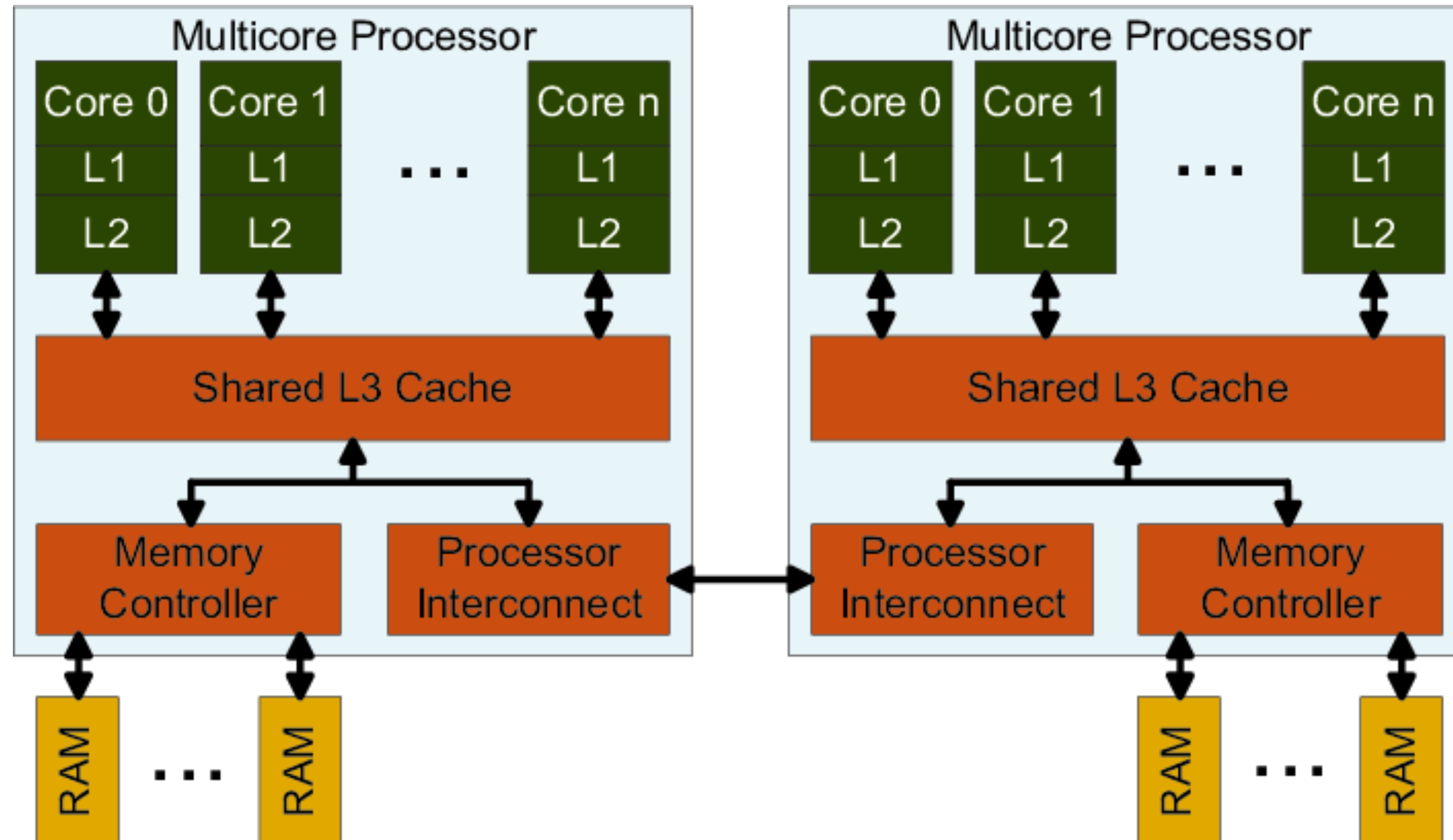
RAM Model



- In-order (!) **sequential** execution of instructions
- Unbounded memory of **words**
- A word is of size $O(\log n)$, for any n
- Constant-time access to each word in memory
- Equal amount of execution time per instruction

Real computing environments are a bit (!!!) different than these assumptions ?

Modern Computer Systems...



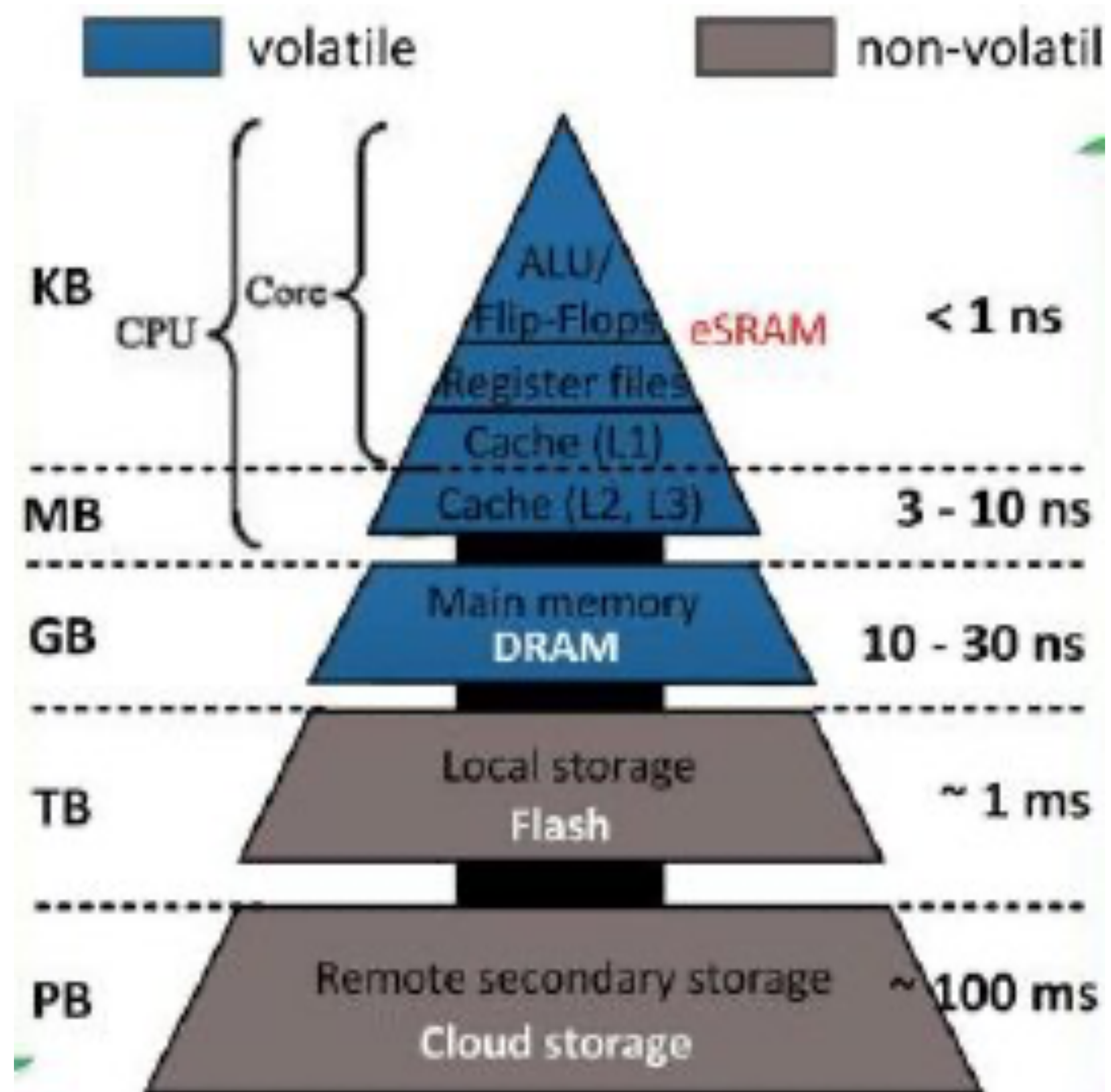
- Sequential execution
- Unbounded memory of **words**
- A word is of size $O(\log n)$
- Constant-time access to each word in memory
- Equal amount of execution time per instruction

THEORY VS PRACTICE

- **Parallel** execution (out-of-order, multi-core, vectorization, pipelining, etc...)
- **Limited** memory of **words**
- A **word** is of **constant** number of bits
- **Variable-time** access due to **memory hierarchy**
- **Different** amount of execution time per instruction

Memory Hierarchy

- CPUs operate on data that resides in their registers !
- In a computer, data flows respecting the memory hierarchy.
- It is a lot more efficient to have the data to be processed next as close as possible to CPU.
- Cache-efficient, cache-friendly, cache-aware, cache-oblivious data structures and related algorithms...



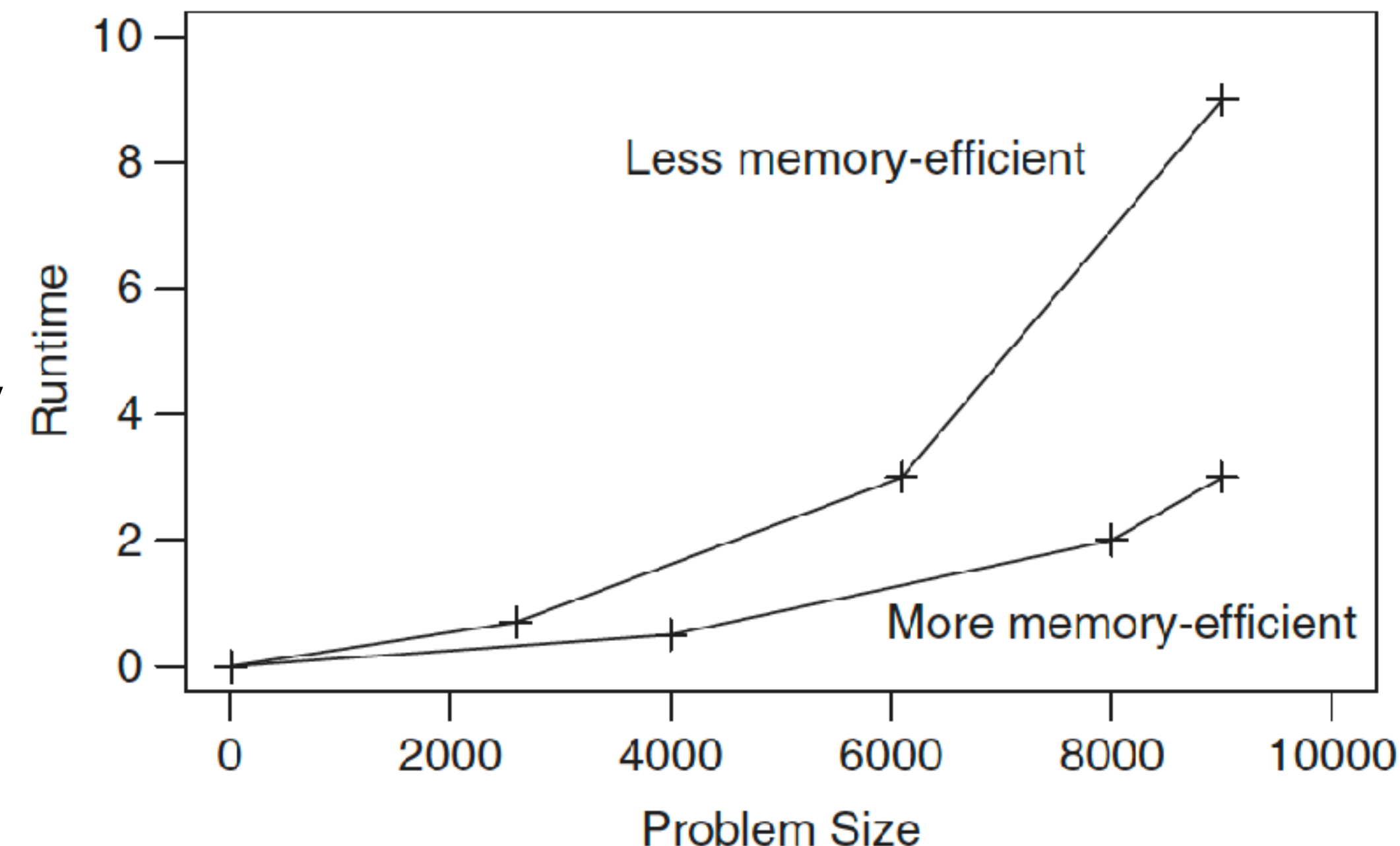
Locality of Reference

- **Temporal Locality or *Locality in Time*:**

If a memory location is accessed, it is likely that it will be accessed again.

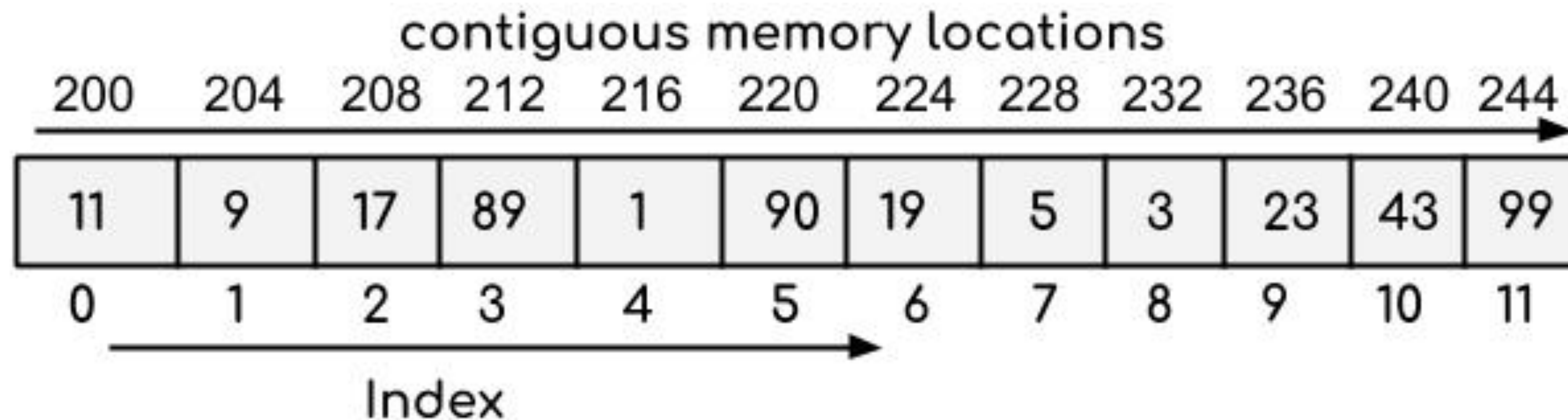
- **Spatial Locality or *Locality in Space*:**

During a time interval, the items near the recently accessed items are more likely to be accessed.



Arrays

- A **contiguous** memory block, in which data resides in **fixed-size** chunks.
- Respects the locality of reference (particularly the *spatial* one)
- Most basic data structure that is used in many others...



- **Random access in $O(1)$ -time.**
- **Insertion/deletion is $O(n)$ -time.**

Range Minimum Queries on Arrays

- RMQ: Given an array of integers $A[1..n]$, find the minimum in the region $A[x..y]$, for the queried x and y positions, where $1 \leq x \leq y \leq n$.
- A primitive building block of many algorithms and related applications

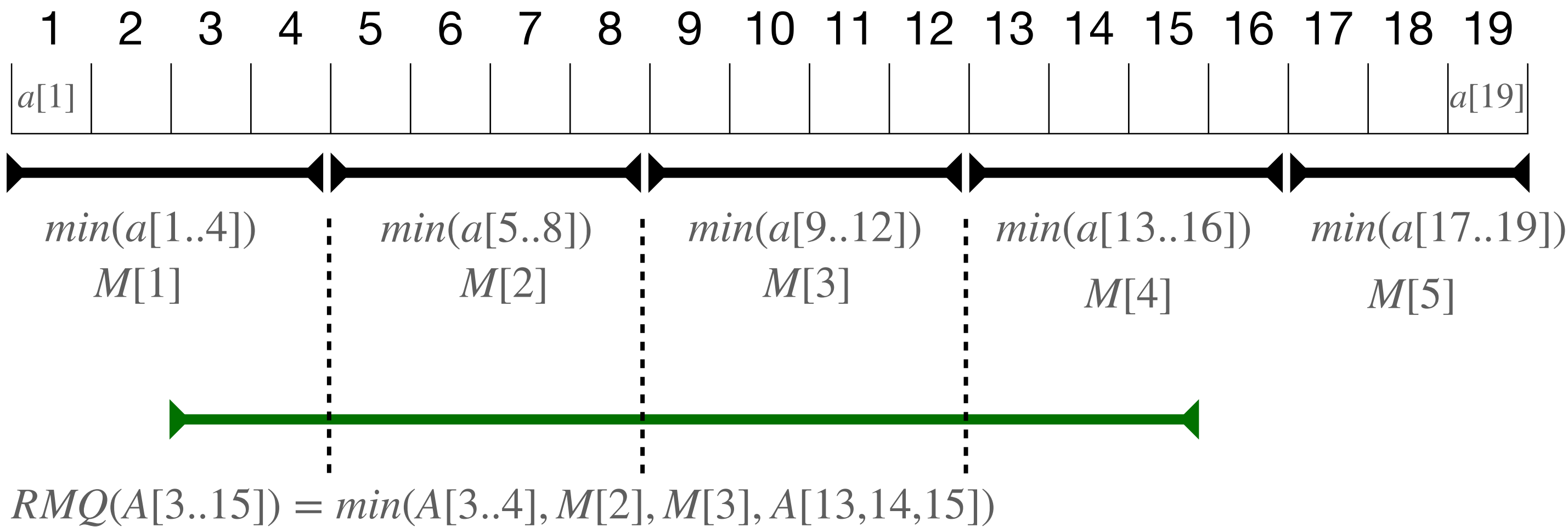
Naive solution: Pass over $A[x..y]$ and return the minimum.

Worst-case time complexity is $O(n)$. Why?

We can do better ...

Range Minimum Queries

$O(\sqrt{n})$ -time , $O(\sqrt{n})$ -space



Keep the minimum of the \sqrt{n} - length intervals in an additional array, which introduces $\left\lceil \frac{n}{\sqrt{n}} \right\rceil \in O(\sqrt{n})$ space complexity

$RMQ(A[3..15])$

What can be the maximum number of comparisons ? Total number of comparisons $\leq 1 + 3\sqrt{n}$, which makes $O(\sqrt{n})$ - time complexity.

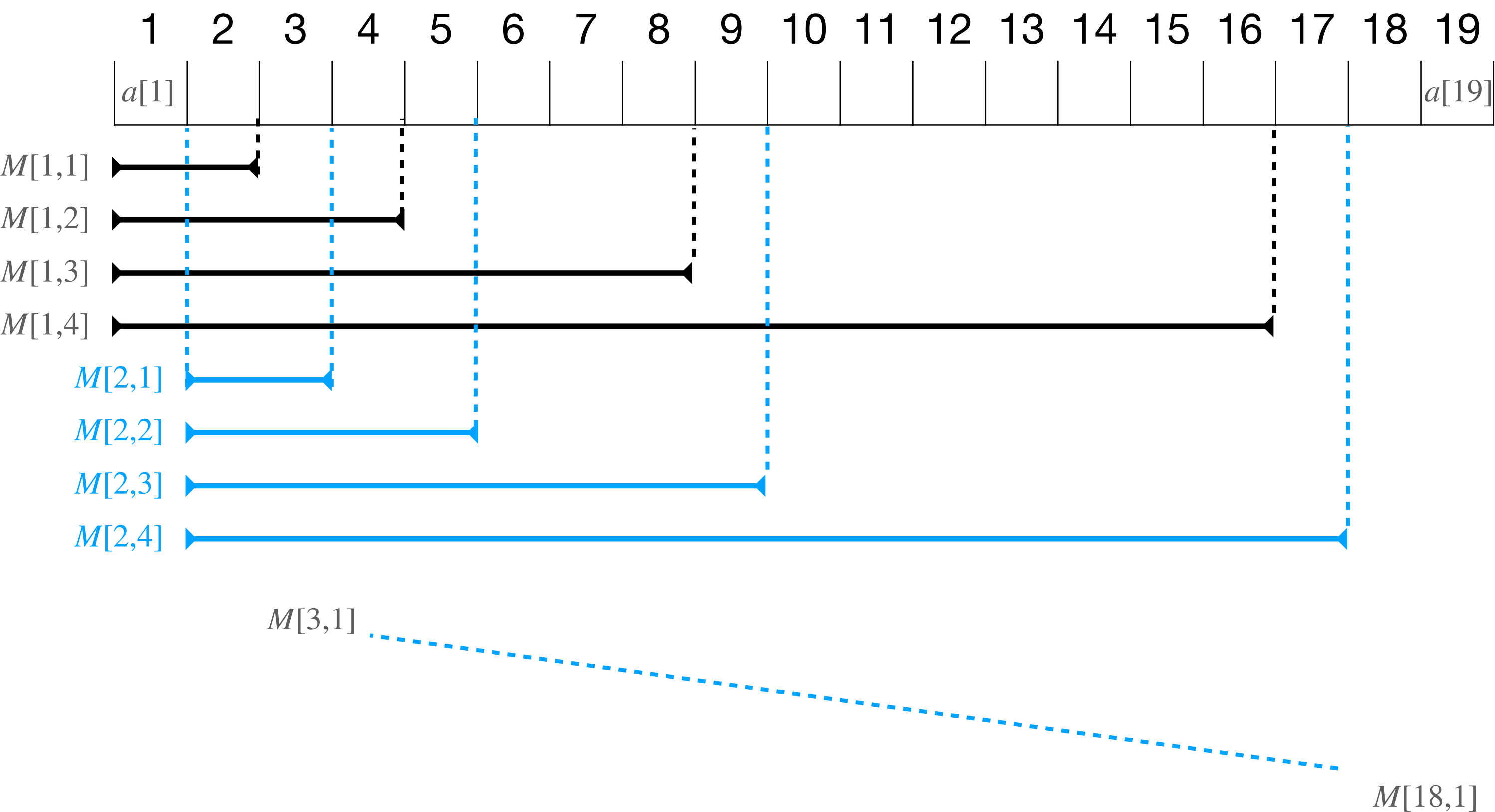
Number of items in the first and last sections $< 2\sqrt{n}$

Number of items from the saved minimums $\leq 1 + \sqrt{n}$

Range Minimum Queries

$O(1)$ -time , $O(n \log n)$ -space

For each $i = 1$ to $n - 1$, and $j = 1$ to $\log n$, such that $i + 2^j - 1 \leq n$,
store $M[i, j] = \min(a[i .. i + 2^j - 1])$ in the look up table M.



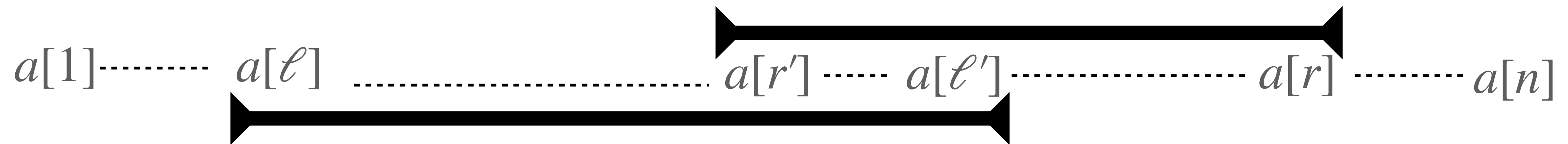
$M[i, j]$	1	2	3	4
1				
2				
...				
18		X	X	X

$O(n \log n)$ space,
as n rows with at most $\log n$ entries

Range Minimum Queries

$O(1)$ -time , $O(n \log n)$ -space

For each $i = 1$ to $n - 1$, and $j = 1$ to $\log n$, such that $i + 2^j - 1 \leq n$,
store $M[i, j] = \min(a[i..i + 2^j - 1])$ in the look up table M.

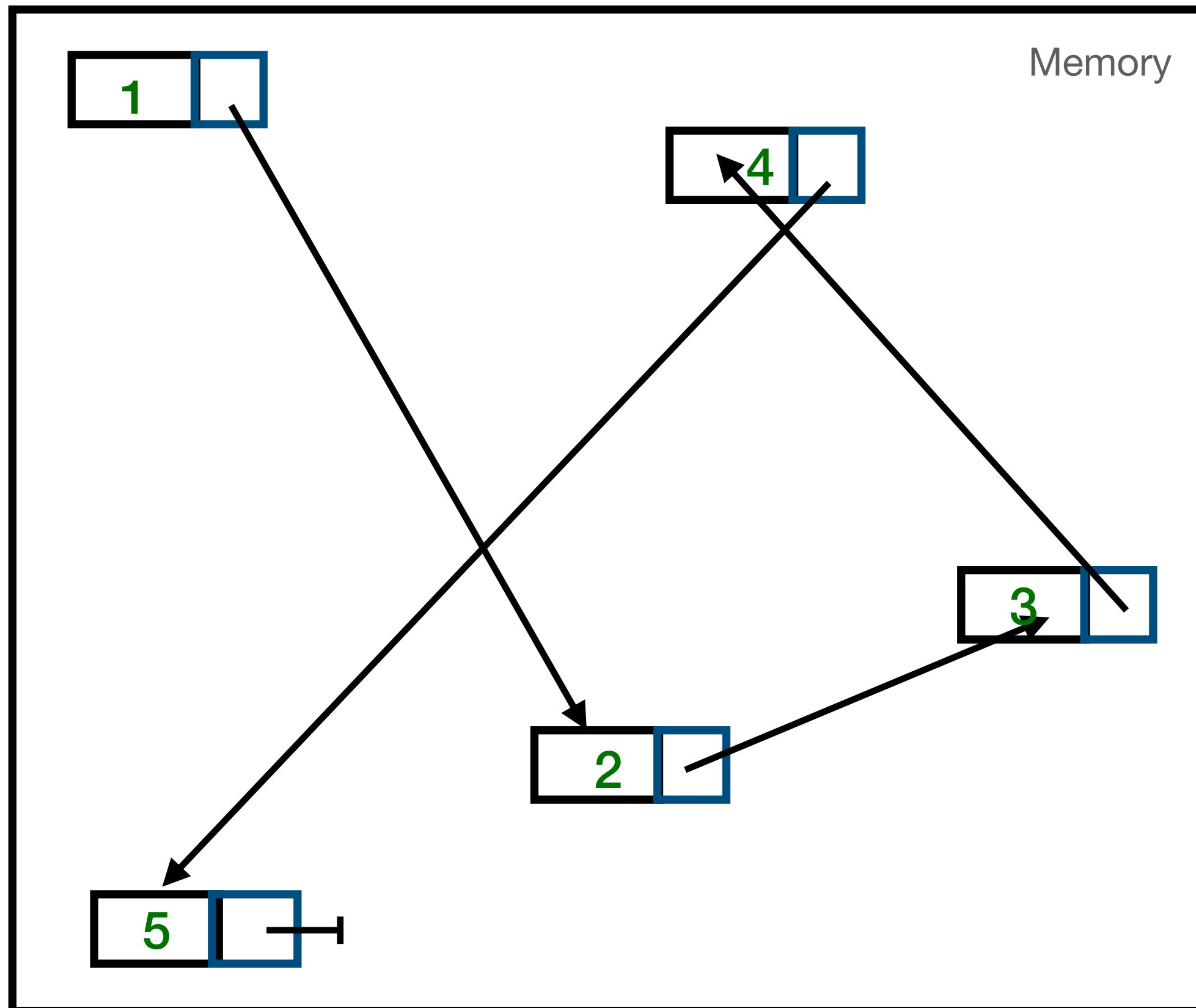


$$\left. \begin{array}{l} \ell' = \ell + 2^i - 1 \text{ for largest } i \text{ such that } \ell + 2^{i+1} - 1 > r \\ r' = r - 2^j + 1 \text{ for largest } j \text{ such that } r - 2^{j+1} + 1 < \ell \end{array} \right\} (\ell' - r') \geq -1 ?$$

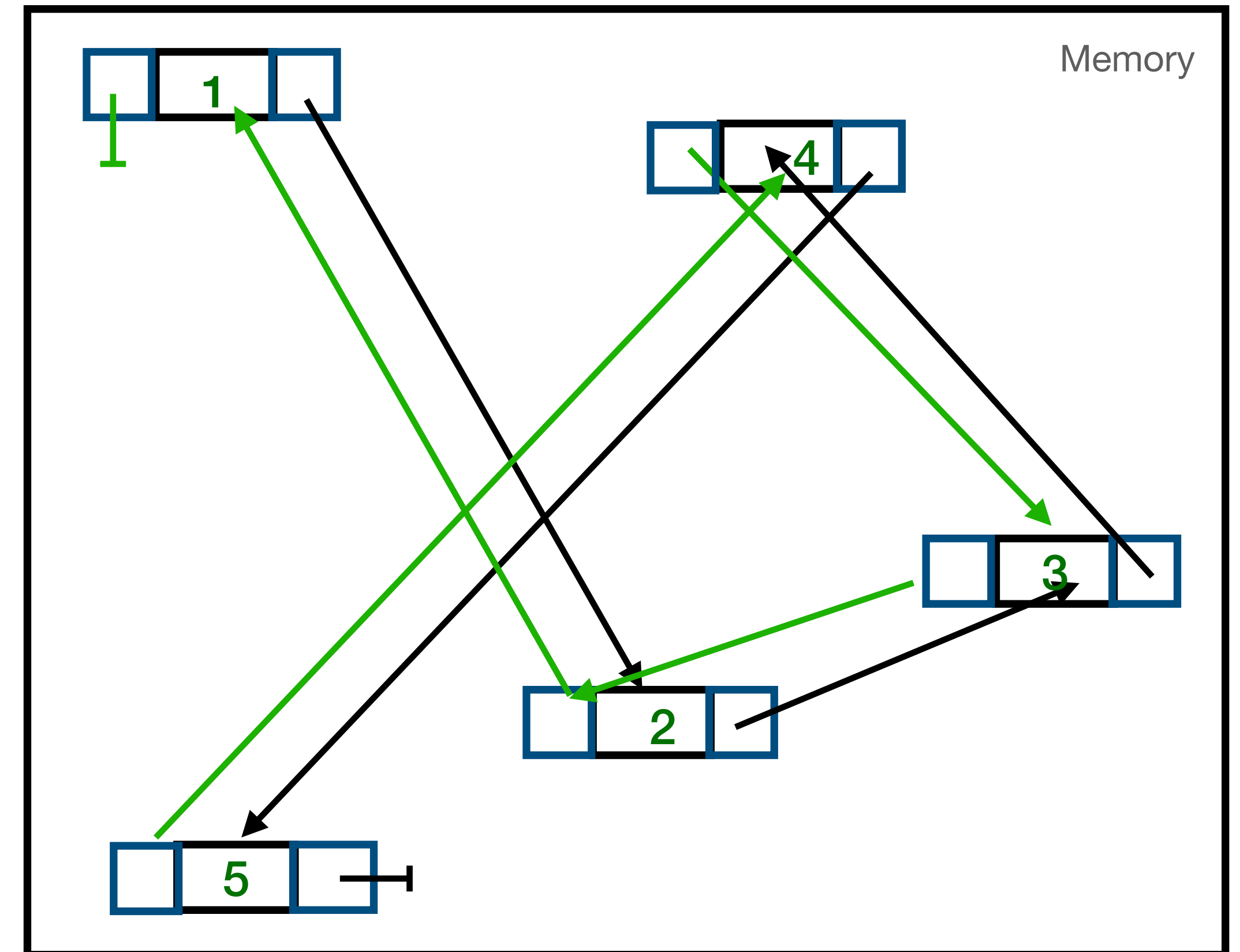
$$RMQ(\ell, r) = \min(M[\ell, i], M[r', j])$$

Linked List

Singly linked list
next ptr, head ptr...



Doubly linked list
next ptr, prev ptr, head ptr, tail ptr...



- Access in $O(n)$ - time
- Insert/delete in $O(1)$ -time (*assuming we are at the position to insert/delete*)

Linked List

On a given doubly linked list of sorted integers, find the pairs that sum up to a queried value K.

On an input $1 \leftrightarrow 2 \leftrightarrow 4 \leftrightarrow 5 \leftrightarrow 6 \leftrightarrow 8 \leftrightarrow 9$ and $K=7$, then (1,6), (2,5) are the pairs.

How can you find? Complexity?

Linked List vs Array

- Random access : $O(n)$ vs $O(1)$
- Insert/delete: $O(1)$ vs $O(n)$
- Space usage
- Sensitivity to the locality of reference

- **Search on sorted sequences: $O(n)$ vs $O(\log n)$**
- **Insert/delete on sorted sequences: $O(1)$ vs $O(n)$**
- **We can have $O(\log n)$ time search with $O(\log n)$ -time insert/delete**

- Please read related array/linked list topics from the suggested text books.
- For the range minimum queries, you can find many resources on the internet.
- Next week we will start with the Skip List data structure. You can find a good coverage related to it in the Goodrich et. al.'s book.
- We will continue with the stacks, queues, trees ...