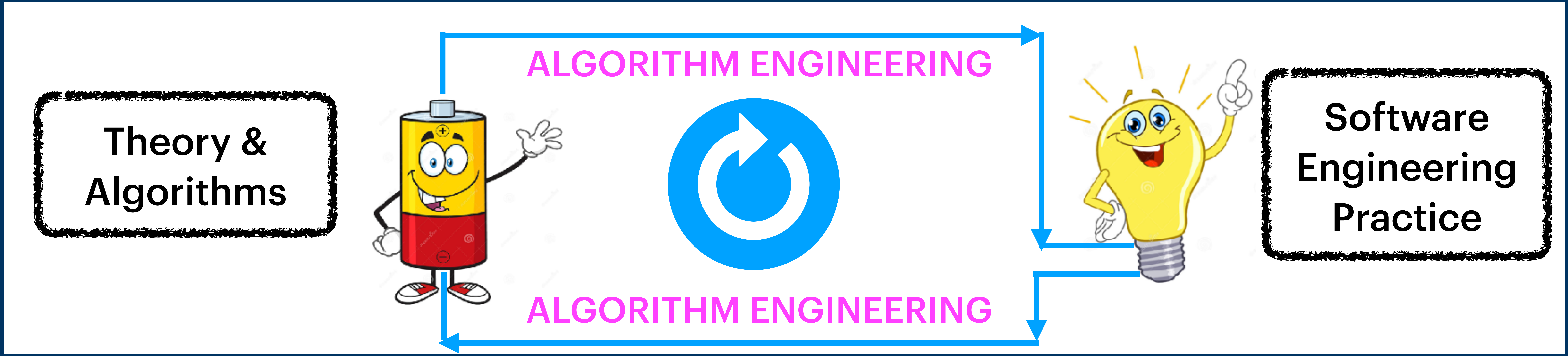


ALGORITHM ENGINEERING

Lecture 8:
Implementation Phase - 3: Memory and I/O optimization

M. Oğuzhan Külekci - kulekci@itu.edu.tr

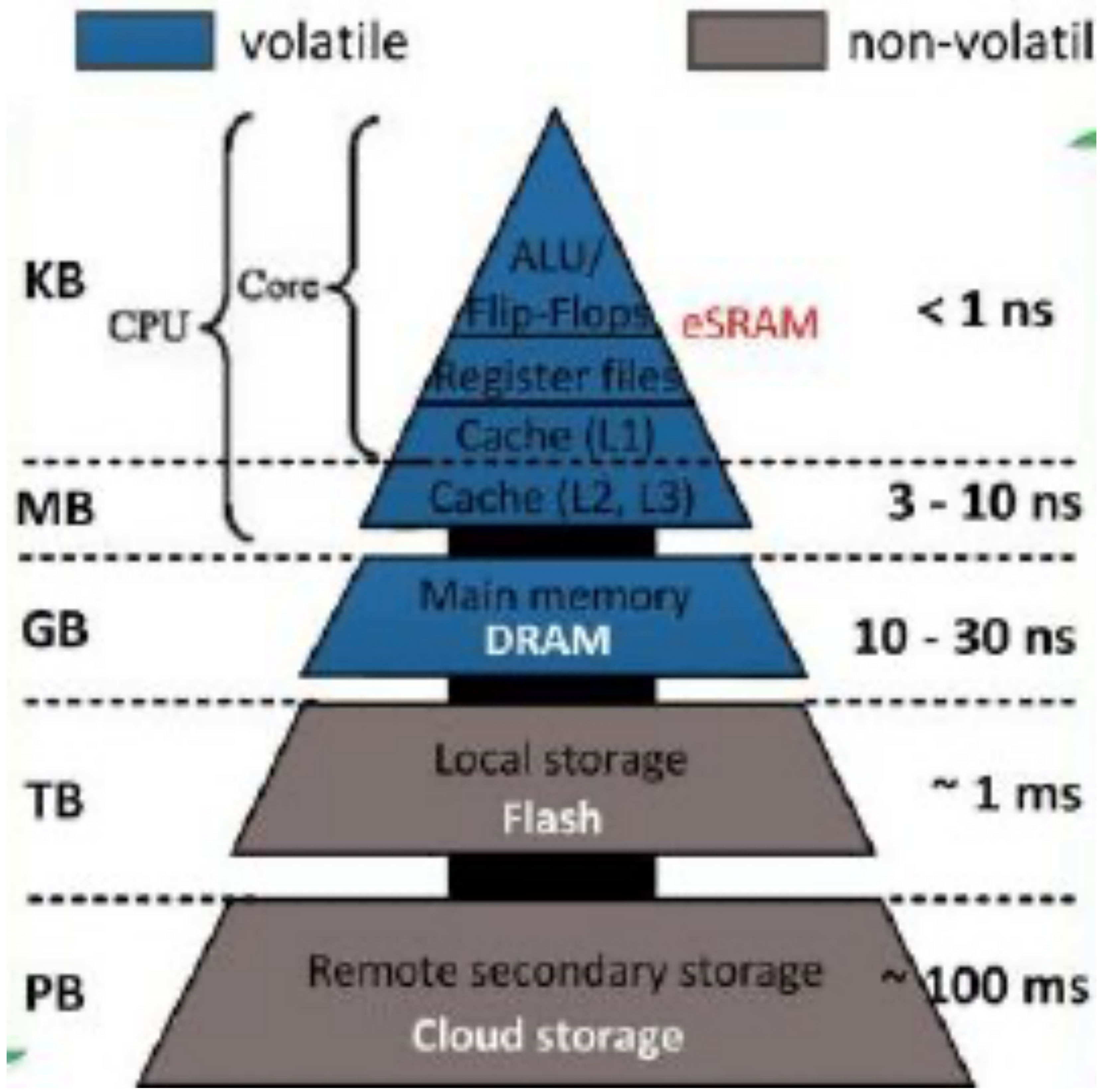
How to make it run faster ?

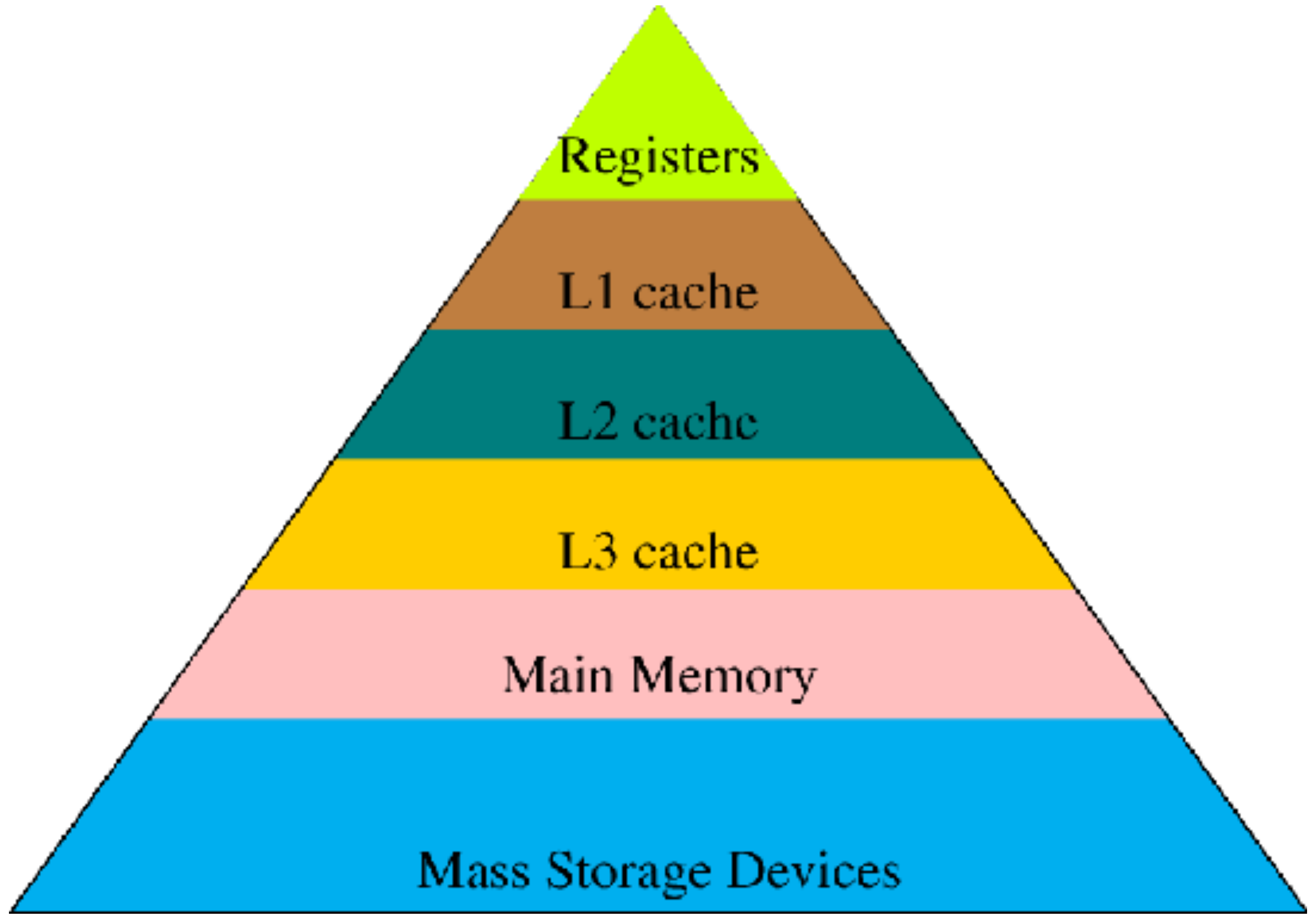
The central question in algorithm engineering

- Every step in the design of a solution has an effect on speed.
- Algorithm design, analysis, and a basic implementation are done, and we want to improve that implementation.

Today we will focus on the effect of memory and I/O utilizations.

The memory and I/O system



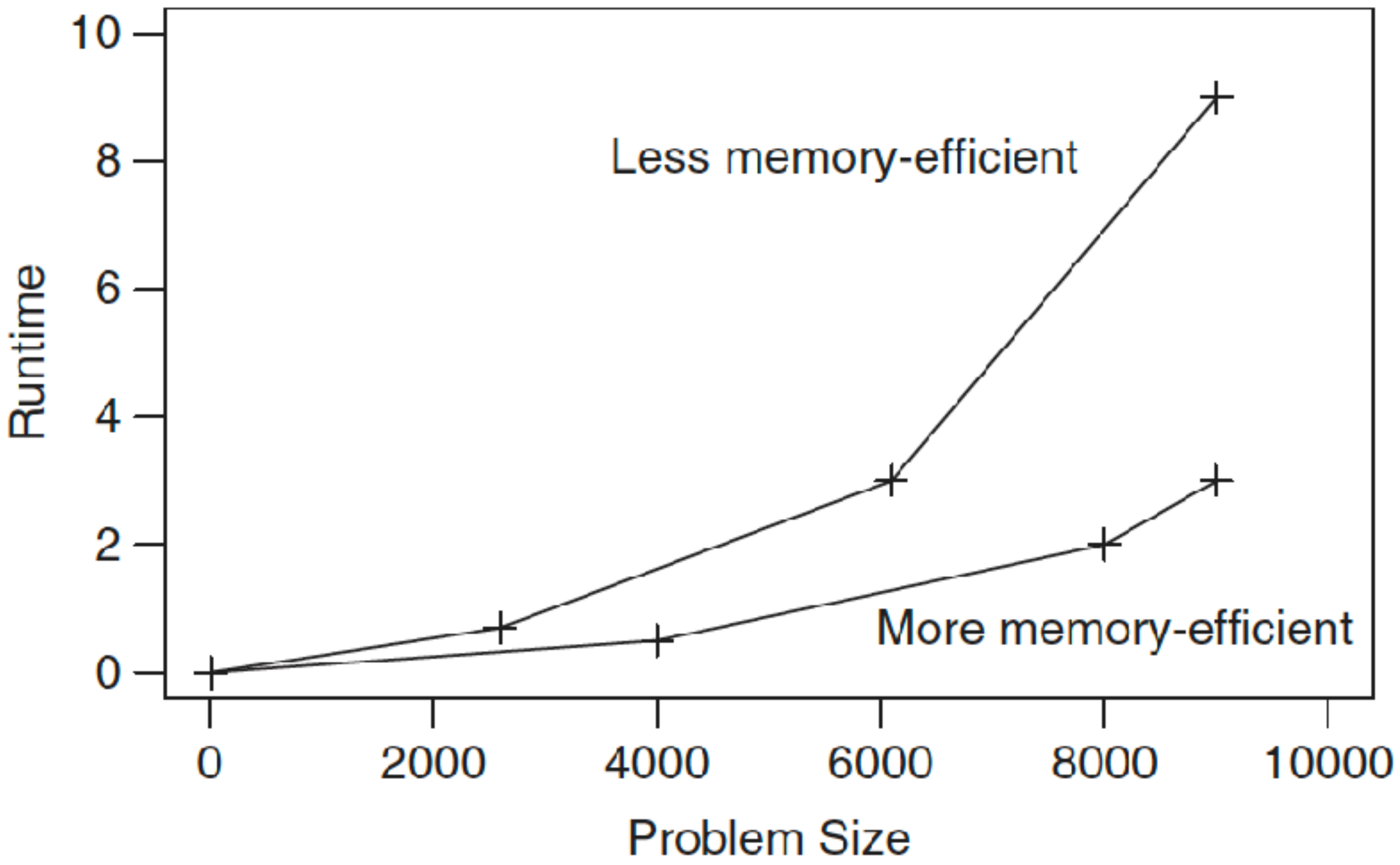


Keep your data as close as possible to the CPU !

Memory Issues

Locality of reference rule:

- Temporal Locality: If a memory location is accessed, it is likely that it will be accessed again.
- Spatial Locality: During a time interval, the items near the recently accessed items are more likely to be accessed.



This is why the computing systems bring requested elements as a block to the CPU !

Memory and I/O Optimization

General approaches:

1. Reduce the memory footprint of the program
2. Reorganize the data access pattern in your algorithm
3. Respect I/O efficiency, particularly on external memory algorithms

Reducing the memory-footprint

- Avoid memory leaks
- Change the data representation, remove unnecessary elements (those can be computed from others)
- Make use of data compression and **compressed data structures**

Try to keep the space consumption of the program data as less as possible to make it as close as possible to CPU!

NOTICE: This is totally against our previously visited “memoization” ! ??

How to decide which one to follow ?

Cache Efficiency

Access to memory elements displays as much temporal and spatial locality as possible.

- **Reuse sooner than later:** Try doing whatever you should in one pass,

  e.g., min, max of an array should be done in one pass
- **Respect memory layout:** Arrays are usually stored in row-major order. Watch this order.

  While passing over 2-d arrays A[i,j], the first loop should consider i and the next one j.
- **Stride-right :** In a loop, the increments of a control variable are better to be small, rather than large
- **Tiling:** In case we are multiplying matrices, consider computing the result block-by-block rather than traversing line by line
- **Fields and arrays:** Try to group the elements into structs rather than maintaining separately.

```
// simple matrix multiplication
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    for (k = 0; k < n; k++)
      c[i][j] = c[i][j] + a[i][k] * 
```

```
// tiled matrix multiplication using 2x2 blocks

for (i =  0; i < n; i += 2)
  for (j = 0; j < n; j += 2)
    for (k = 0; k < n;  k++)
      for (x = i; x < min(i+2, n); x ++)
        for (y = j;  y < min (i+2, n); y++)
          c[x][y]  = c[x][y] + a[x][k] * b[k][y];
```

Whack-a-mole and Locality of Reference

- **Temporal Locality:** The moles near to recent pop-up are more likely to pop-up.
- **Spatial Locality:** Once a mole pops up, it is likely that it will pop up again.

Cache-efficient Binary Search

Normal

Key	10	20	30	40	50	60	70
Probes	3	2	3	1	3	2	3

Cache-efficient

Key	40	20	60	10	30	50	70
Probes	1	2	2	3	3	3	3

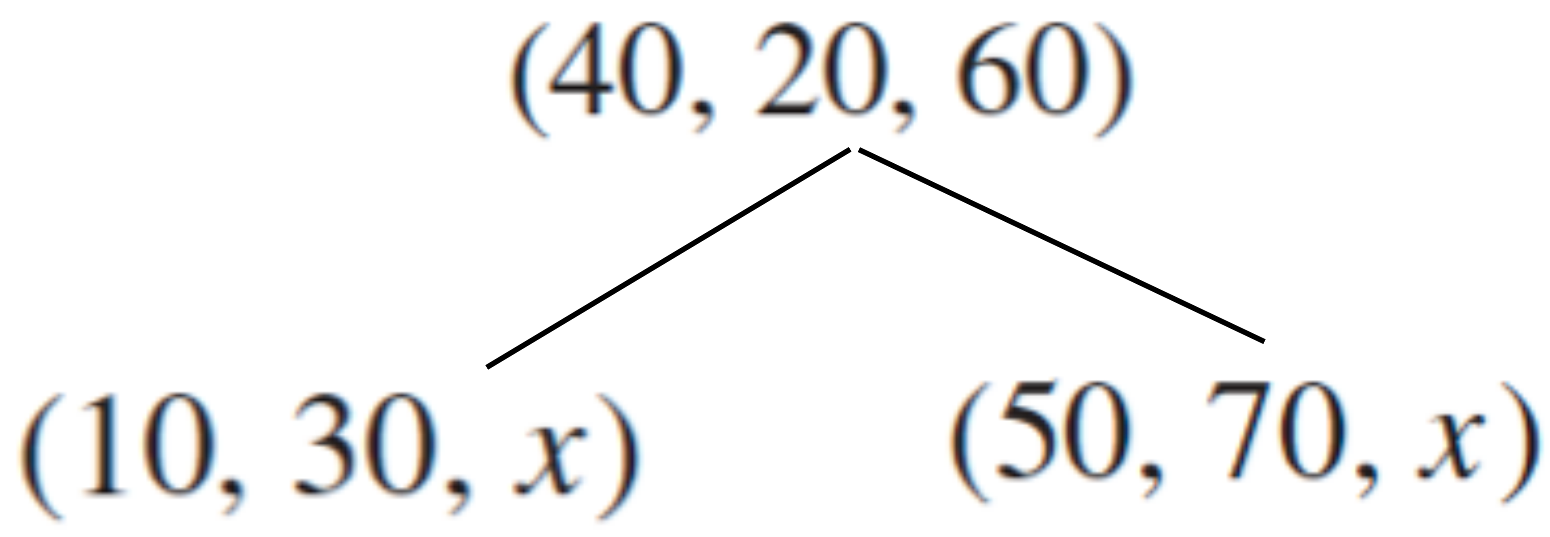
Even better binary search

- Maintain keys in lists (or groups)
- Represent the whole data on a heap-like data structure to get rid of pointers

Cache-aware

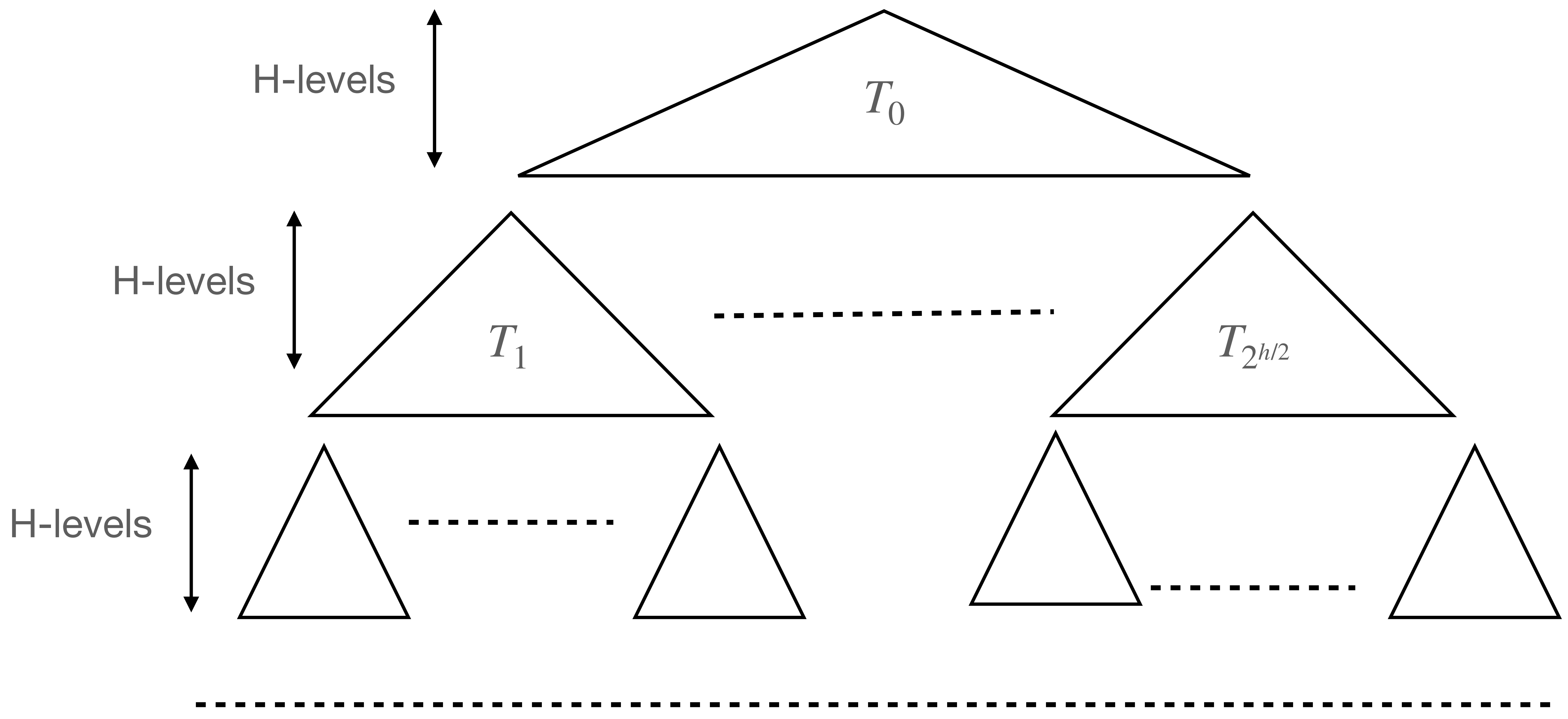
(Remember cache-size is a parameter)

Keys	(40, 20, 60)	(10, 30, x)	(50, 70, x)
Cache loads	1	2	2



Cache-oblivious

(No need to cache-size as a parameter)

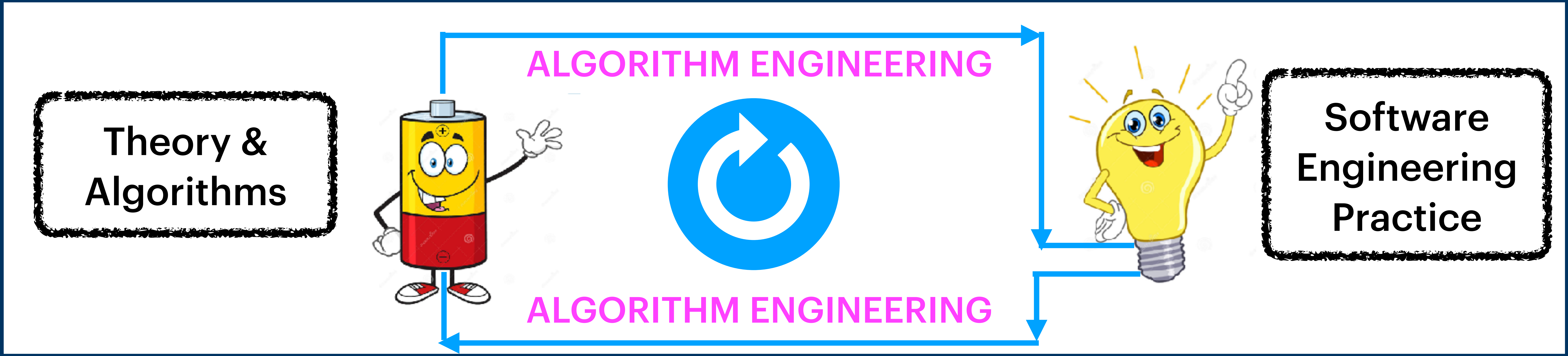


Individual nodes on the tree, but they are grouped according to levels

I/O efficiency

- When working on huge data, so external memory algorithms are used

1. Minimize open-close operations: File/stream opening and closing are costly, so minimize them
2. Reduce Latency: Read/write elements in chunks rather than individuals
3. Decouple I/O and instruction execution: If possible remove I/O from the loops, so as they do not wait I/O during the execution
4. Exploit locality: Same as in memory…



ALGORITHM ENGINEERING

Lecture 9:
Implementation Phase - 4: Concurrency and I/O optimization

M. Oğuzhan Külekci - kulekci@itu.edu.tr