

دانشگاه آزاد اسلامی واحد تبریز



نام درس: طراحی و تحلیل الگوریتم های پیشرفته

بخش: عملیات روی حافظه های خارجی

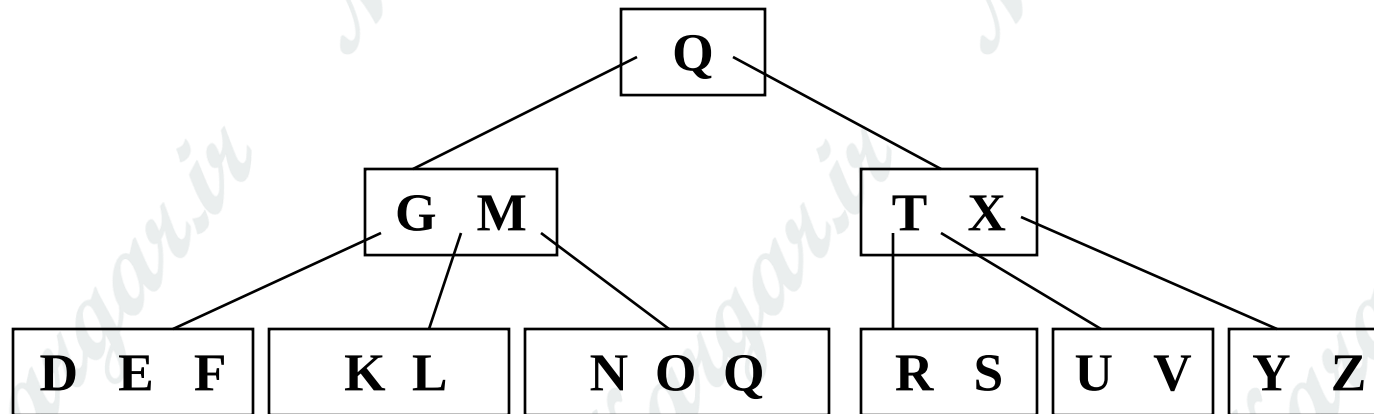
نام استاد: دکتر مسعود کارگر

External Mem. DS & Algorithms

- B-trees
- External-memory sorting
- Goals of the lecture:
 - *to understand the algorithms of B-tree and its variants and to be able to analyze them;*
 - *to understand how the different versions of **merge-sort** derived algorithms work in external memory and to be able to compare their efficiency;*
 - *to understand why the amount of available main-memory is an important parameter for the efficiency of external-memory algorithms.*

B-trees: Insert

- Insertion is always performed at the leaf level
- Let's do an example ($t = 2$):
 - Insert: H, J, P

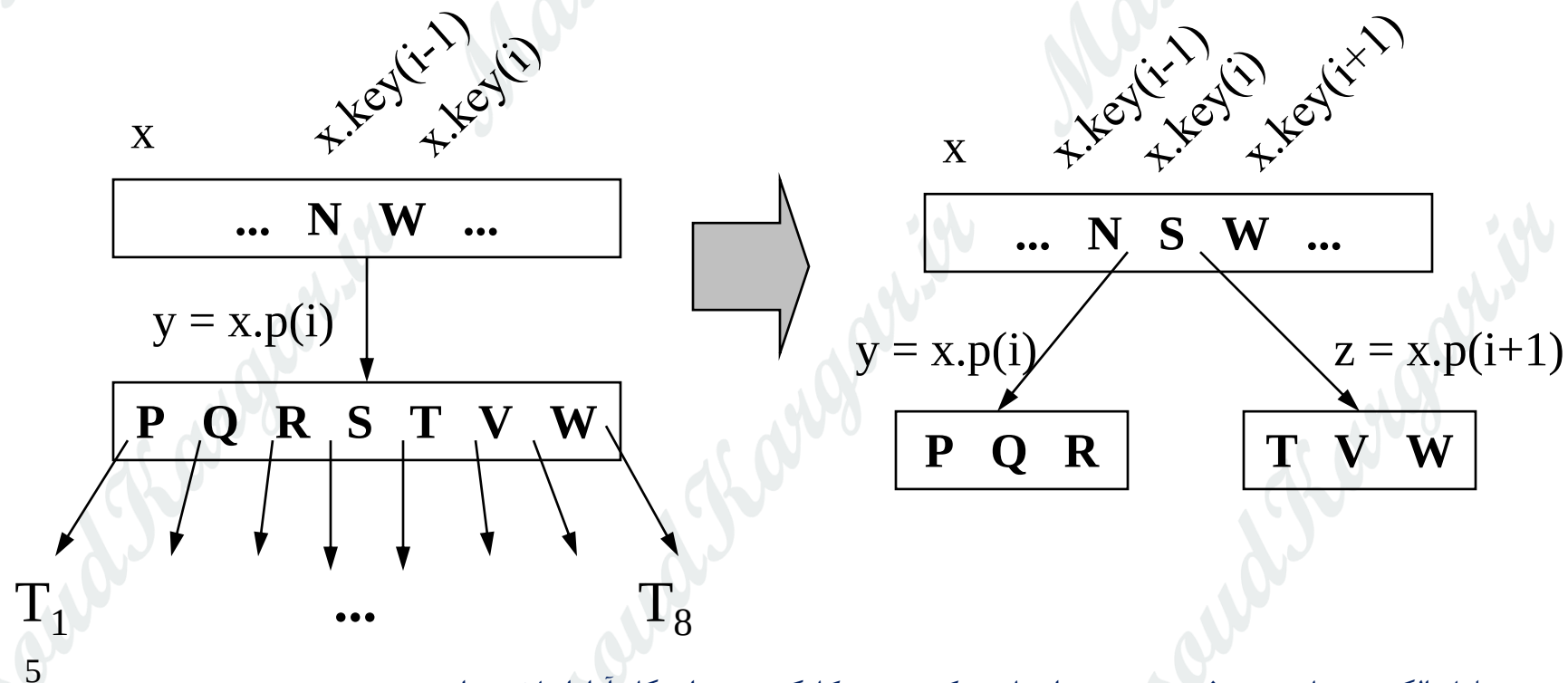


Splitting Nodes I

- Nodes fill up and reach their maximum capacity $2t - 1$
- Before we can insert a new key, we have to “make room,” i.e., split a node

Splitting Nodes II

- Result: one key of x moves up to parent + 2 nodes with $t-1$ keys
 - How many I/O operations?



Insert I

- Skeleton of the algorithm:
 - *Down-phase*: recursively traverse down and find the leaf
 - Insert the key
 - *Up-phase*: if necessary, split and propagate the splits up the tree
- Assumptions:
 - In the *down-phase* pointers to traversed nodes are saved in the stack. Function *parent(x)* returns a parent node of *x* (pops the stack)
 - *split(y:Bnode):(zk:key_t, z:Bnode)*

Insert II

DownPhase (x, k)

```
01 i ← 1
02 while i ≤ x.n() and k > x.key(i)
03     i ← i+1
04 if x.leaf() then
05     return x
06 else DiskRead(x.p(i))
07     return DownPhase(x.p(i), k)
```

Insert (T, k)

```
01 x ← DownPhase(T.root(), k)
02 UpPhase(x, k, nil)
```

Insert III

UpPhase (x, k, p)

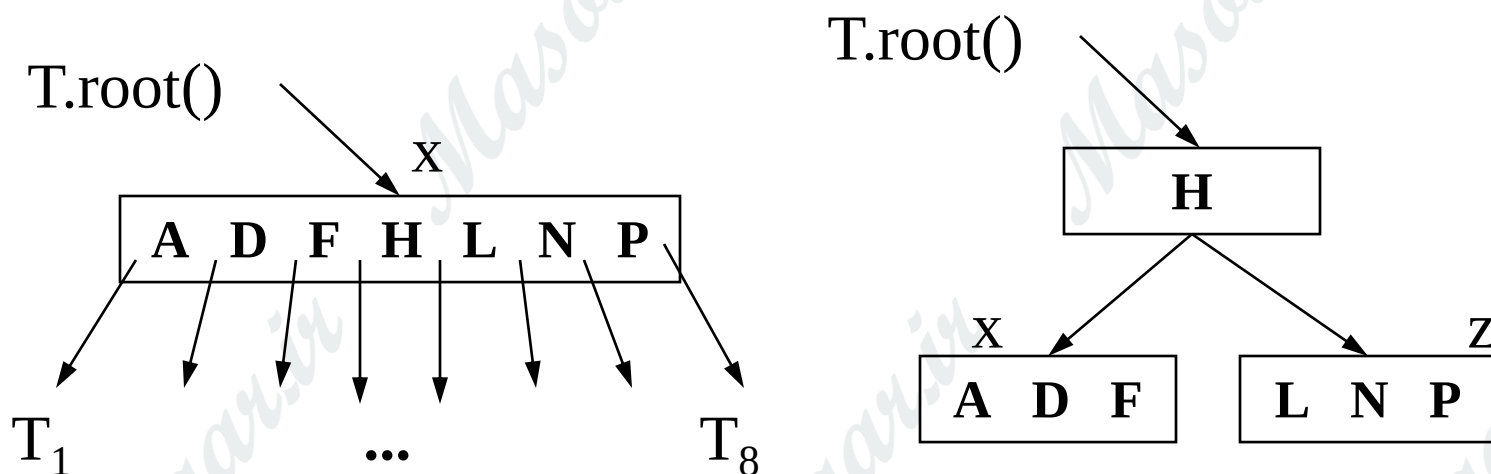
```
01 if  $x.n() = 2t-1$  then  
02    $(zk, z) \leftarrow \text{split}(x)$   
03   if  $k \leq zk$  then InsertIntoNode ( $x, k, p$ )  
04   else InsertIntoNode ( $z, k, p$ )  
05   if  $\text{parent}(x) = \text{nil}$  then (Create new root)  
06   else UpPhase ( $\text{parent}(x), zk, z$ )  
07 else InsertIntoNode ( $x, k, p$ )
```

InsertIntoNode (x, k, p)

Inserts the key k and the following pointer p (if not *nil*) into the sorted order of keys of x , so that all the keys before k are smaller or equal to k and all the keys after k are greater than k

Splitting the Root

- Splitting the root requires the creation of a new root



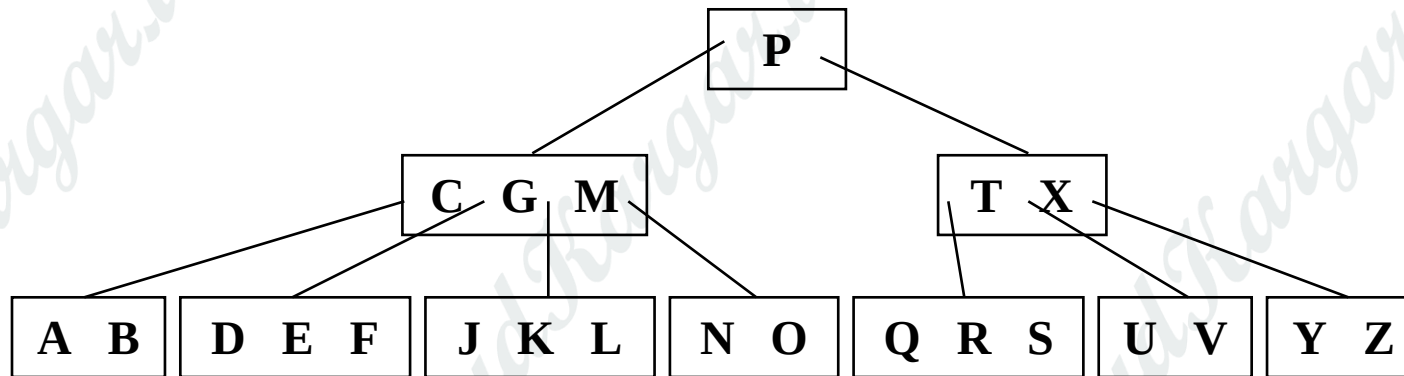
- The tree grows at the top instead of the bottom

One/Two Phase Algorithms

- Running time: $O(h) = O(\log_B n)$
- Insert could be done in one traversal down the tree (by splitting all full nodes that we meet, "just in case")
- Disadvantage of the two-phase algorithm:
 - Buffer of $O(h)$ pages is required

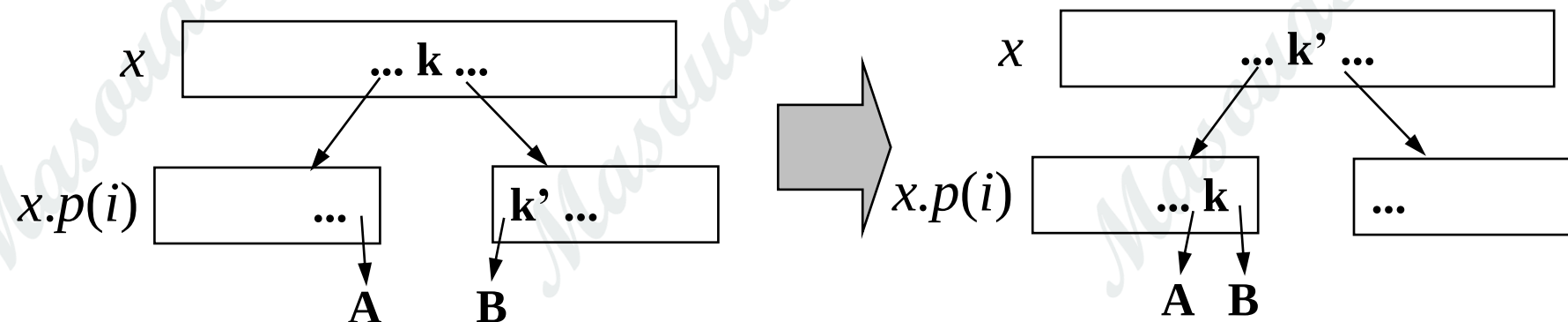
Deletion

- Case 1: A key k is in a non-leaf node
 - Delete its predecessor (which is always in a leaf, thus case 2) and put it in k 's place.
- Case 2: A key is in a leaf-node:
 - Just delete it and handle under-full nodes
- Try: delete M, B, K ($t = 3$)

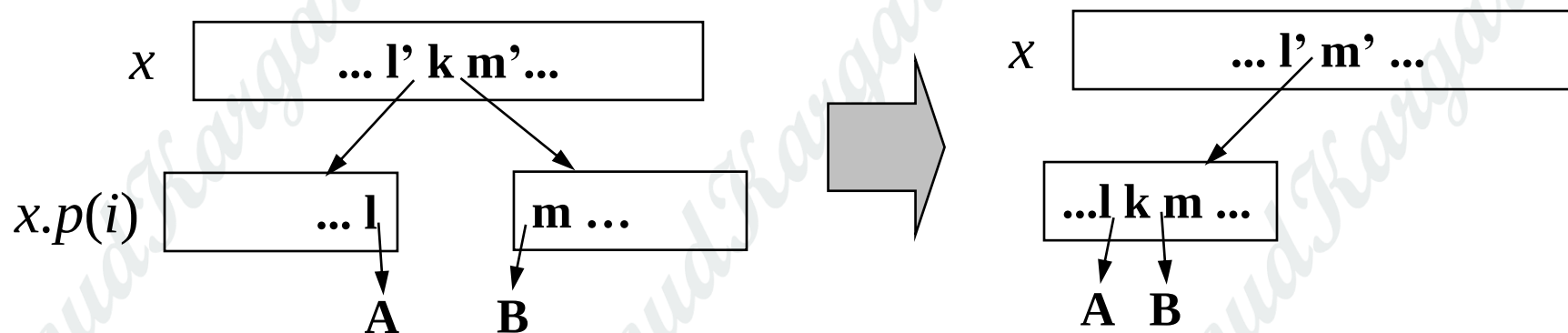


Handling Under-full Nodes

■ Distributing:



■ Merging:

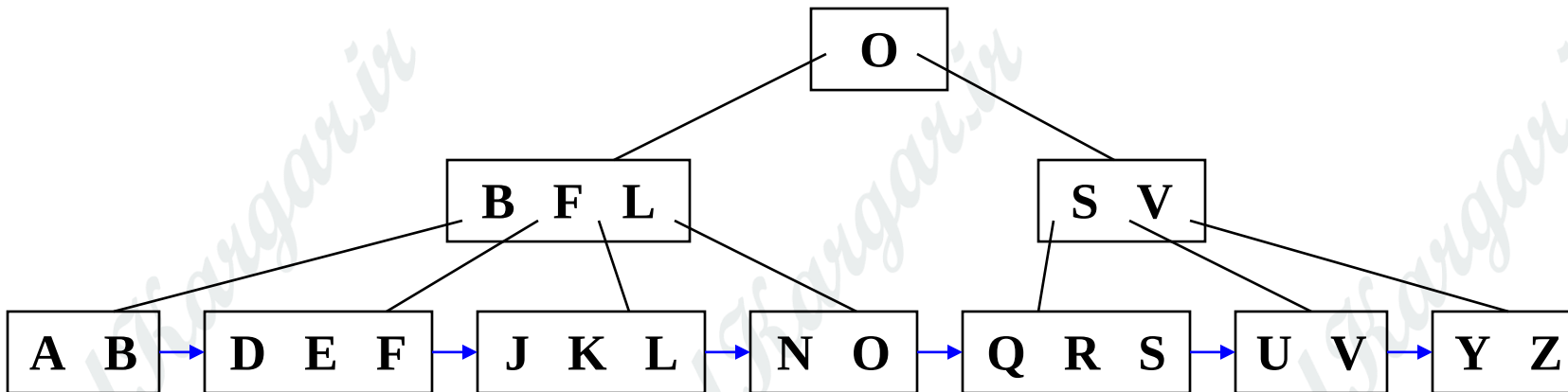


Sequential access

- Other useful ADT operator: *successor*
 - For example, range queries: *find all accounts with the amount in the range [100K – 200K]*.
 - How do you do that in B-trees?

B⁺-trees I

- B⁺-trees is a variant of B-trees:
 - All data keys are in leaf nodes
 - The split does not *move* the middle key to the parent, but *copies* it to the parent!
 - Leaf-nodes are connected into a (doubly) linked list



B⁺-trees II

- Lets draw a B⁺-tree ($t = 2$):
 - $A, B, C, D, E, F, G, H, I, J, K$
- How is the range query performed?
 - Compare with the B-tree

External-Memory Sorting

- External-memory algorithms
 - When data do not fit in main-memory
- External-memory sorting
 - Rough idea: sort peaces that fit in main-memory and “merge” them
- Main-memory merge sort:
 - The main part of the algorithm is Merge
 - Let's merge:
 - 3, 6, 7, 11, 13
 - 1, 5, 8, 9, 10

Main-Memory Merge Sort

Merge-Sort(A)

```
01 if length(A) > 1 then  
02     Copy the first half of A into array A1  
03     Copy the second half of A into array A2  
04     Merge-Sort(A1)  
05     Merge-Sort(A2)  
06     Merge(A, A1, A2)
```

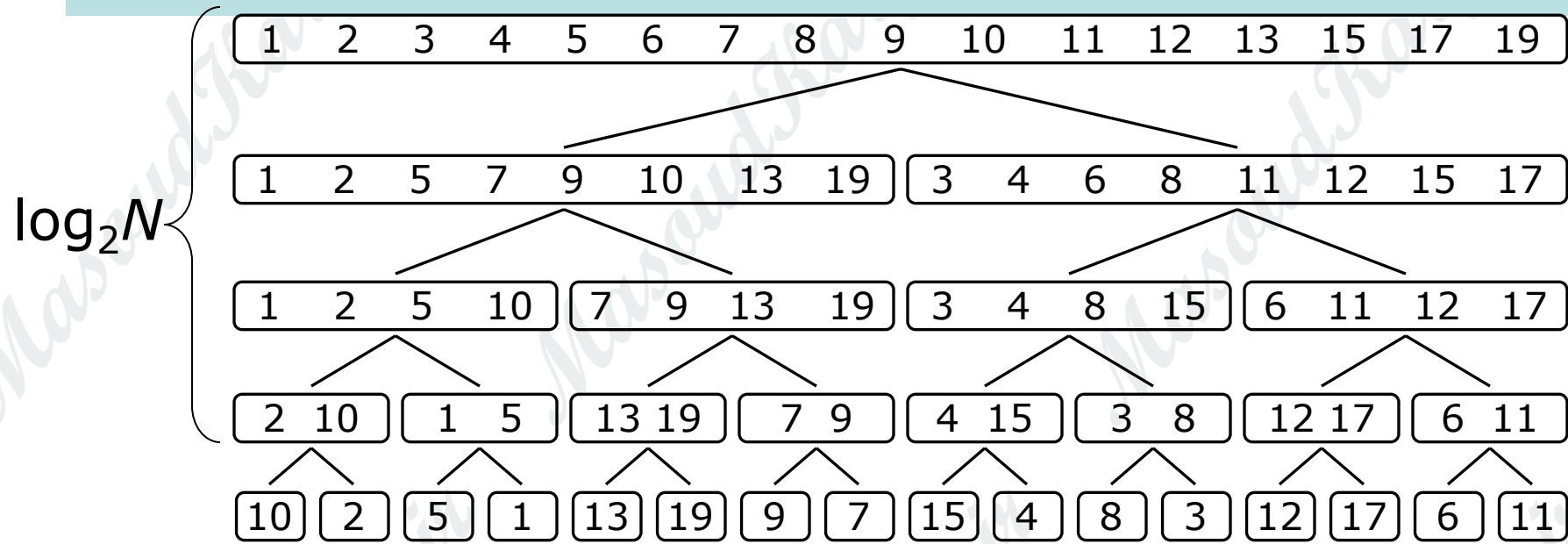
Divide

Conquer

Combine

■ Running time?

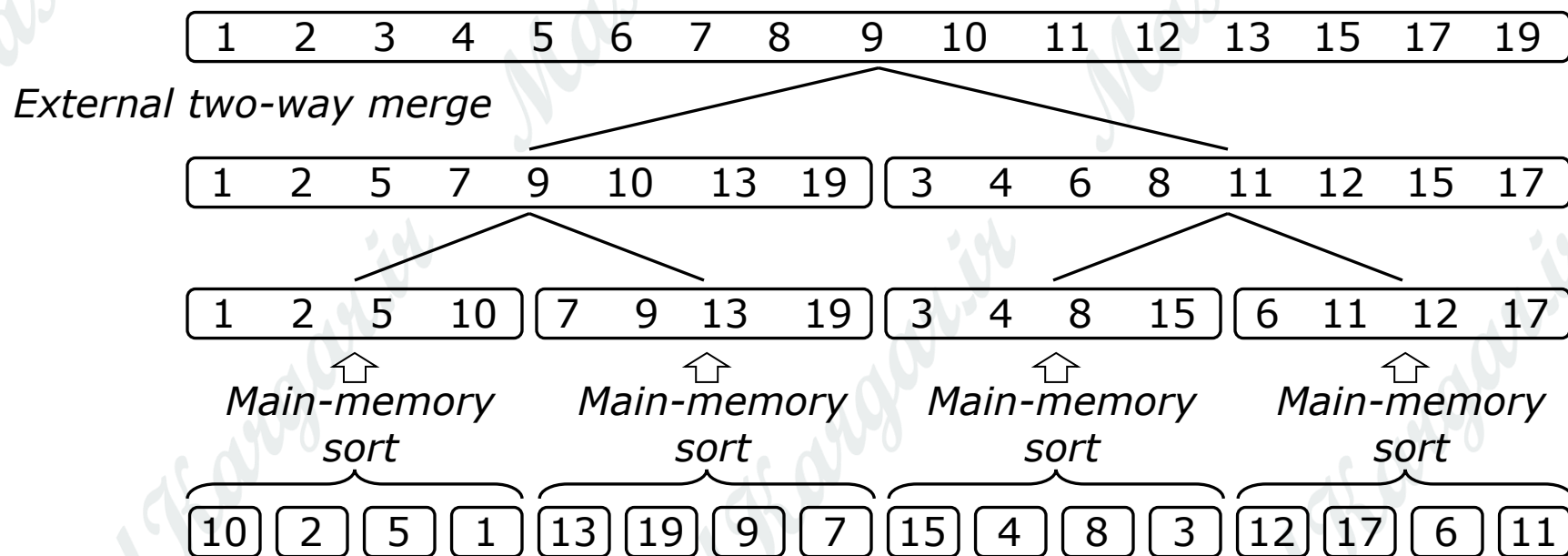
Merge-Sort Recursion Tree



- In each level: merge *runs* (sorted sequences) of size x into runs of size $2x$, decrease the number of runs twofold.
- What would it mean to run this on a file in external memory?

External-Memory Merge-Sort

- Idea: increase the size of initial runs!
 - Initial runs – the size of available main memory (M data elements)

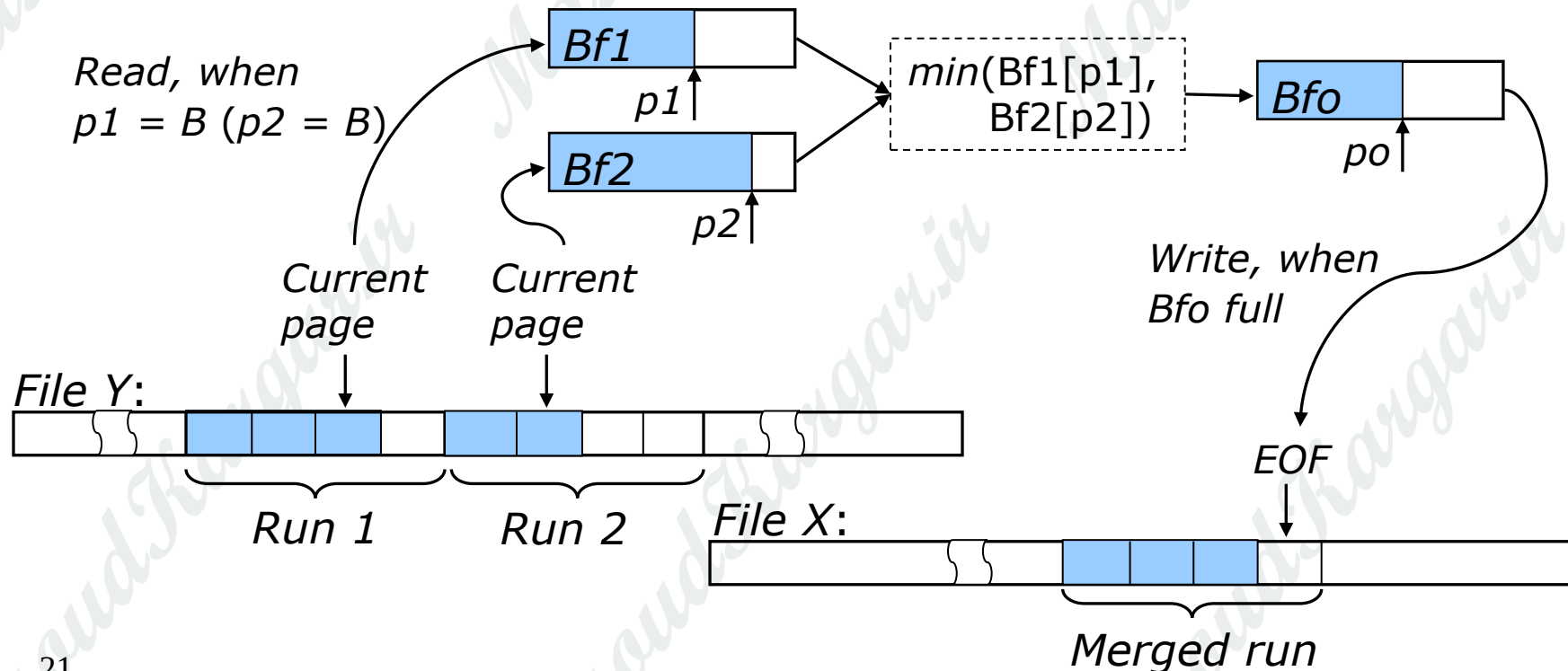


External-Memory Merge Sort

- Input file X , empty file Y
- *Phase 1*: Repeat until end of file X :
 - Read the next M elements from X
 - Sort them in main-memory
 - Write them at the end of file Y
- *Phase 2*: Repeat while there is more than one run in Y :
 - Empty X
 - $MergeAllRuns(Y, X)$
 - X is now called Y , Y is now called X

External-Memory Merging

- *MergeAllRuns(Y, X)*: repeat until the end of *Y*:
 - Call *TwowayMerge* to merge the next two runs from *Y* into one run, which is written at the end of *X*
- *TwowayMerge*: uses three main-memory arrays of size *B*



Analysis: Assumptions

- Assumptions and notation:

- Disk page size:

- B data elements

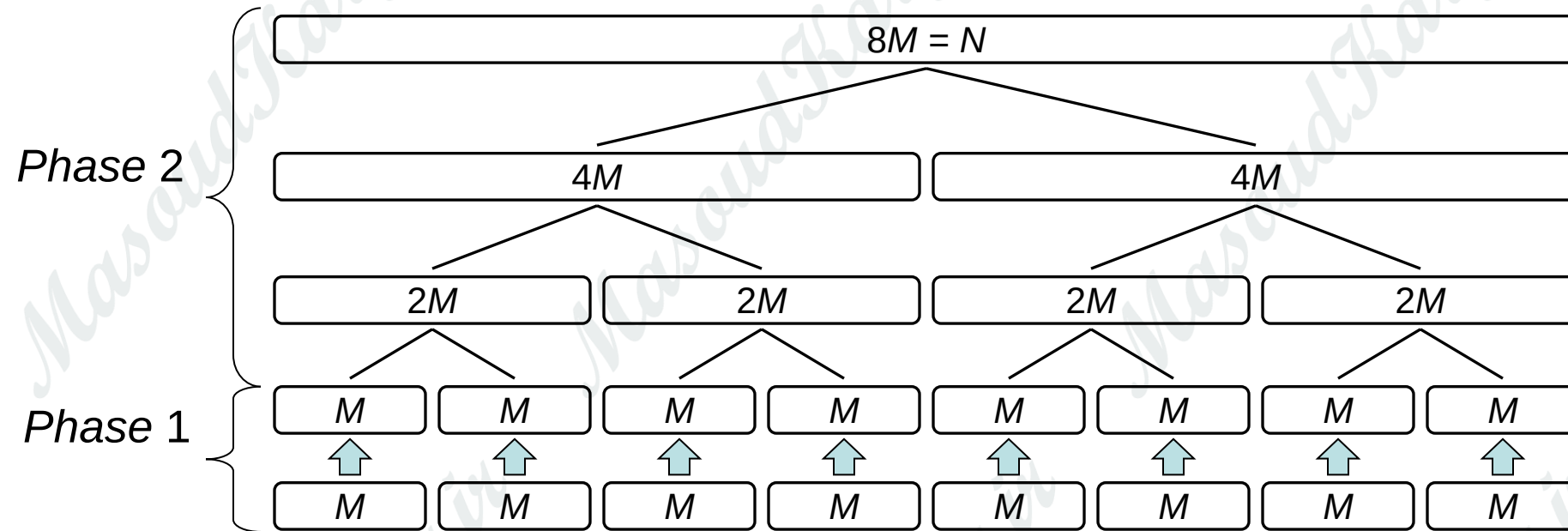
- Data file size:

- N elements, $n = N/B$ disk pages

- Available main memory:

- M elements, $m = M/B$ pages

Analysis



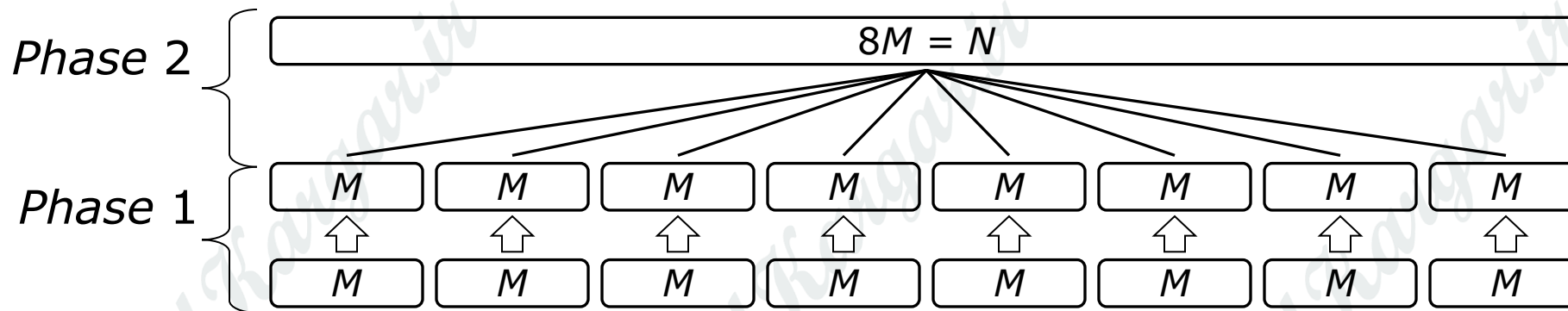
- Phase 1:
 - Read file X, write file Y: $2n = O(n)$ I/Os
- Phase 2:
 - One iteration: Read file Y, write file X: $2n = O(n)$ I/Os
 - Number of iterations: $\log_2 N/M = \log_2 n/m$

Analysis: Conclusions

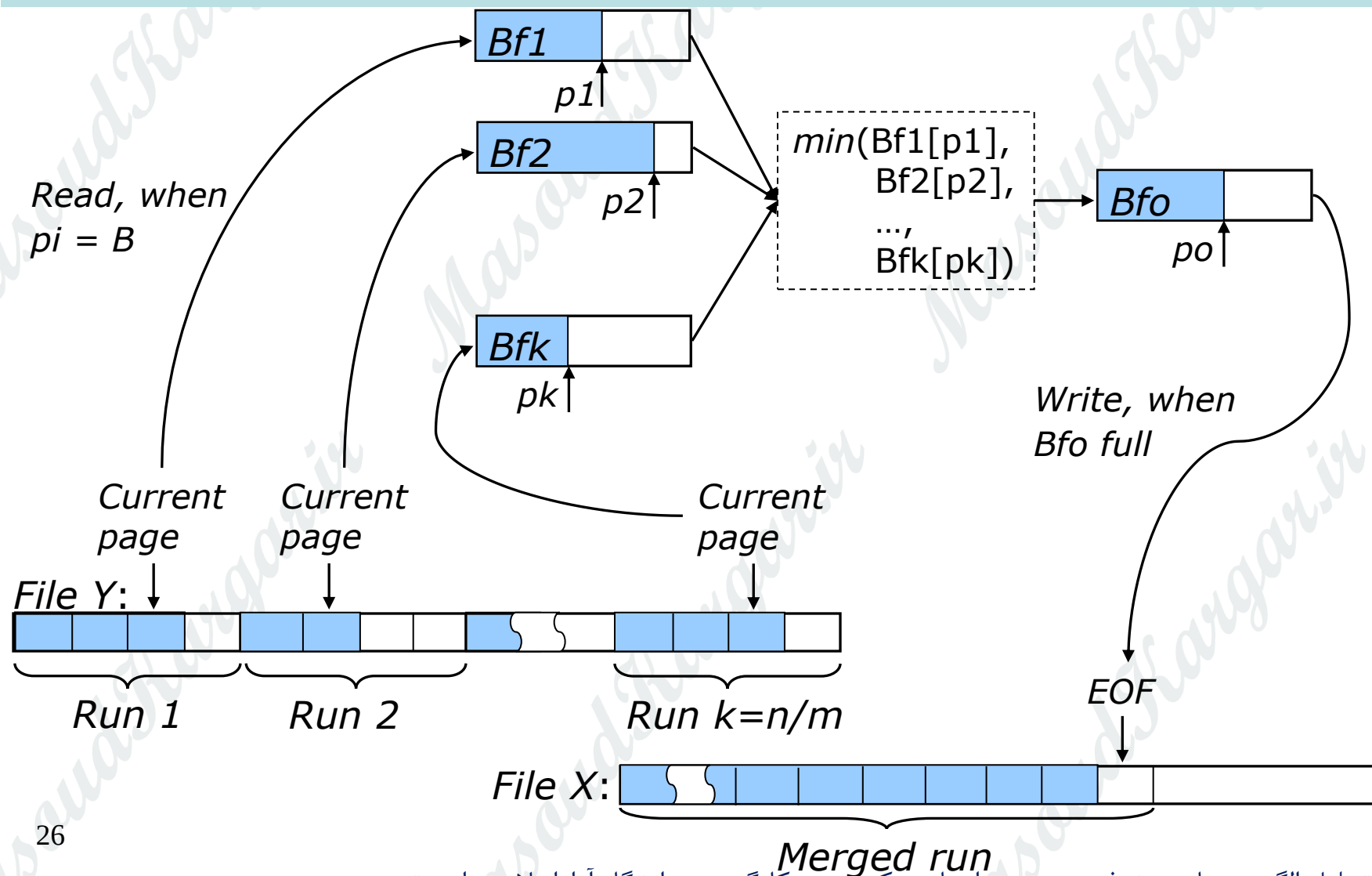
- Total running time of external-memory merge sort: $O(n \log_2 n/m)$
- We can do better!
- Observation:
 - Phase 1 uses all available memory
 - Phase 2 uses just 3 pages out of m available!!!

Two-Phase, Multiway Merge Sort

- Idea: merge all runs at once!
 - Phase 1: the same (do internal sorts)
 - Phase 2: perform *MultiwayMerge*(Y, X)



Multiway Merging



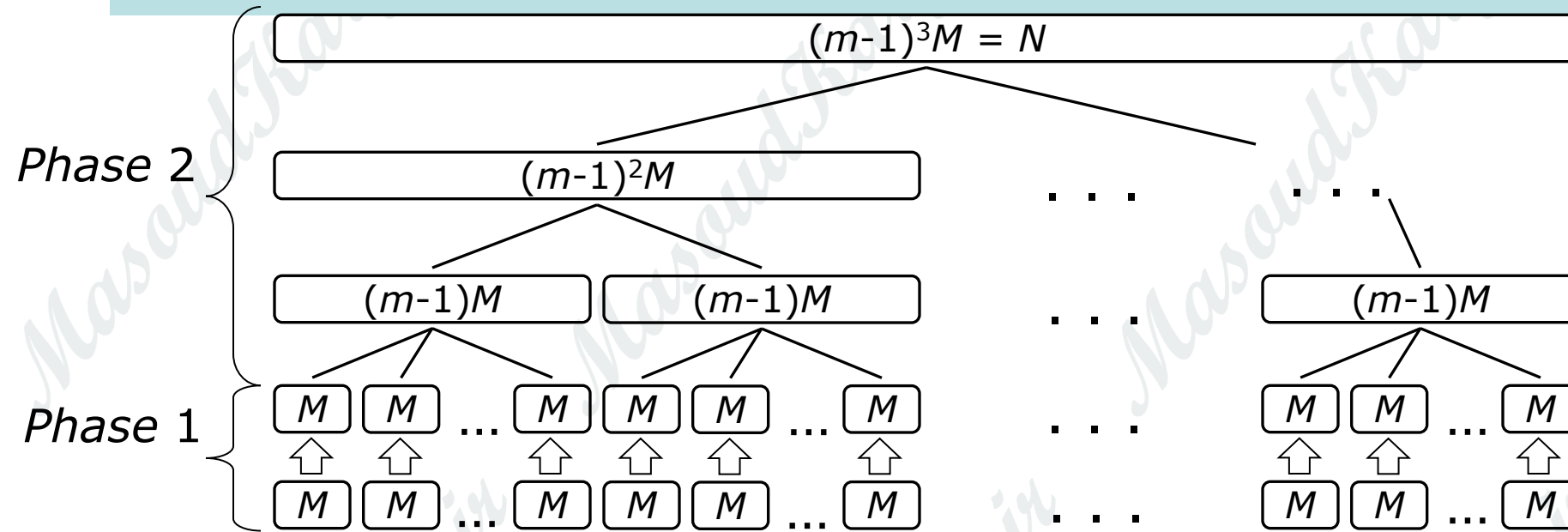
Analysis of TPMMS

- Phase 1: $O(n)$, Phase 2: $O(n)$
- Total: $O(n)$ I/Os!
- The catch: files only of “limited” size can be sorted
 - Phase 2 can merge a maximum of $m-1$ runs.
 - Which means: $N/M < m-1$
 - *How large files can we sort with TPMMS on a machine with 128Mb main memory and disk page size of 16Kb?*

General Multiway Merge Sort

- What if a file is very large or memory is small?
- General *multiway merge sort*:
 - Phase 1: the same (do internal sorts)
 - Phase 2: do as many iterations of merging as necessary until only one run remains
 - Each iteration repeatedly calls *MultiwayMerge*(Y, X) to merge groups of $m-1$ runs until the end of file Y is reached

Analysis



- Phase 1: $O(n)$, each iteration of phase 2: $O(n)$
- How many iterations are there in phase 2?
 - Number of iterations: $\log_{m-1} N/M = \log_m n$
- Total running time: $O(n \log_m n)$ I/Os

Conclusions

- External sorting can be done in $O(n \log_m n)$ I/O operations for any n
 - This is asymptotically optimal
- In practice, we can usually sort in $O(n)$ I/Os
 - Use two-phase, multiway merge-sort