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)
if length(A) > 1 then
    Copy the first half of A into array A1
    Copy the second half of A into array A2
    Merge-Sort(A1)
    Merge-Sort(A2)
    Merge(A, A1, A2)
```

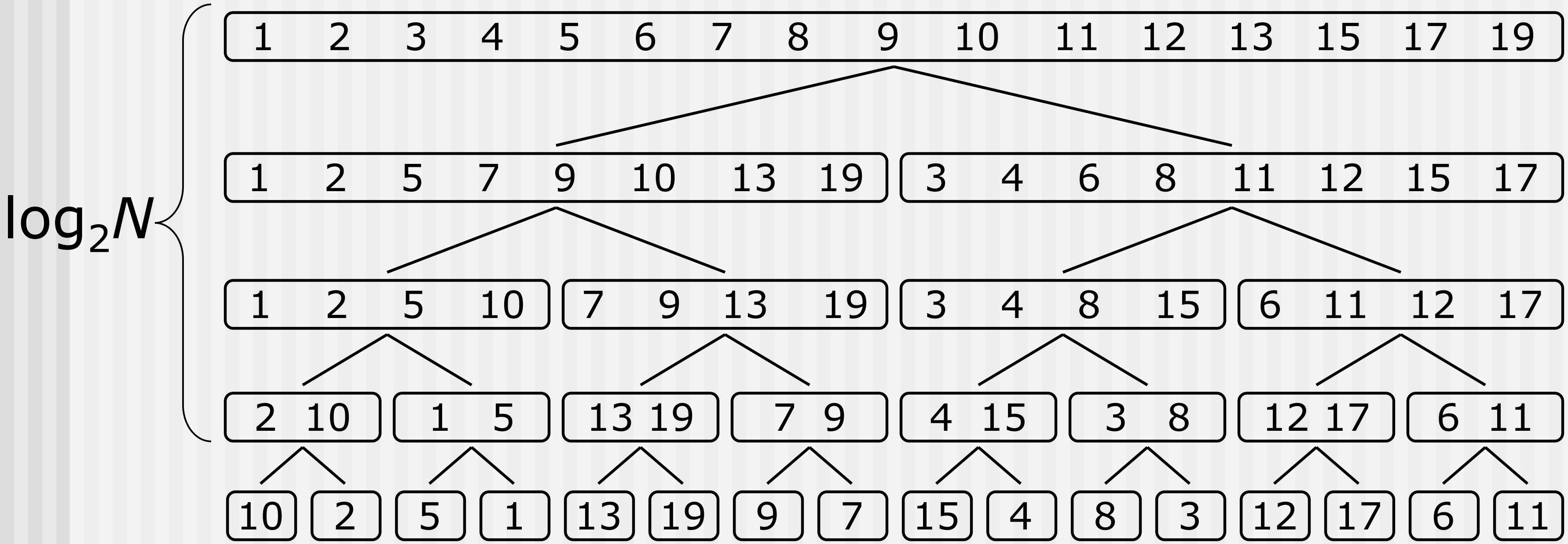
Divide

Conquer

Combine

- Running time?

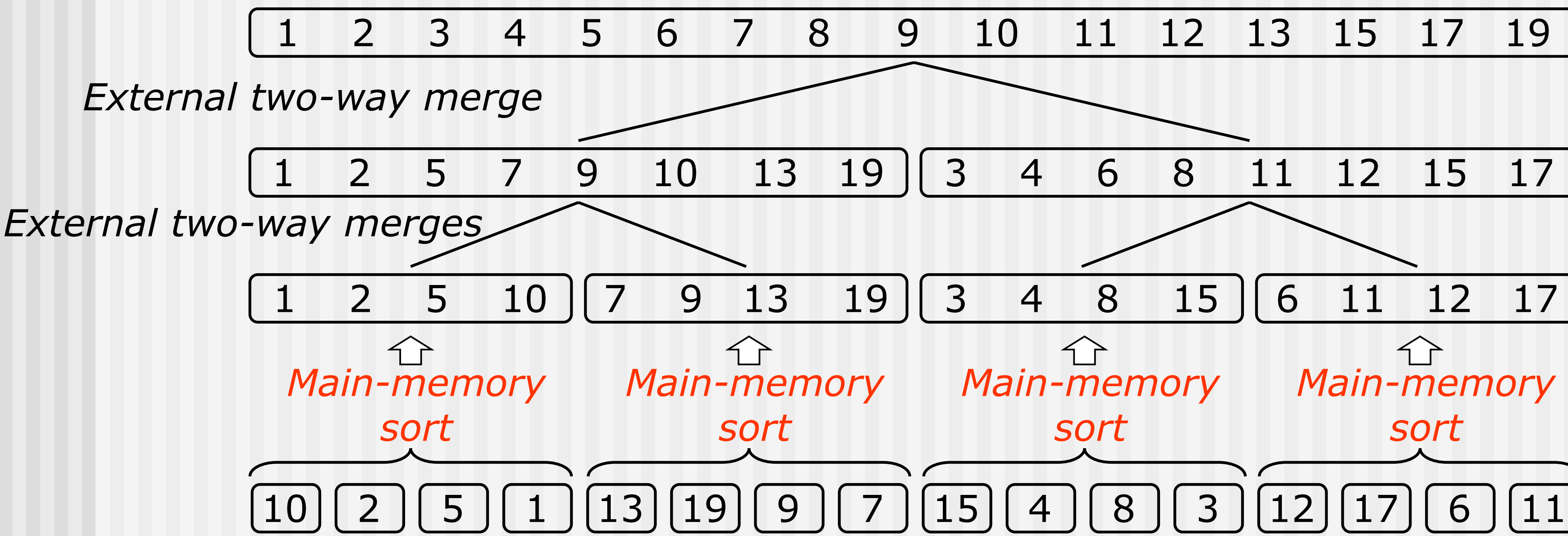
Merge-Sort Recursion Tree



- 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)



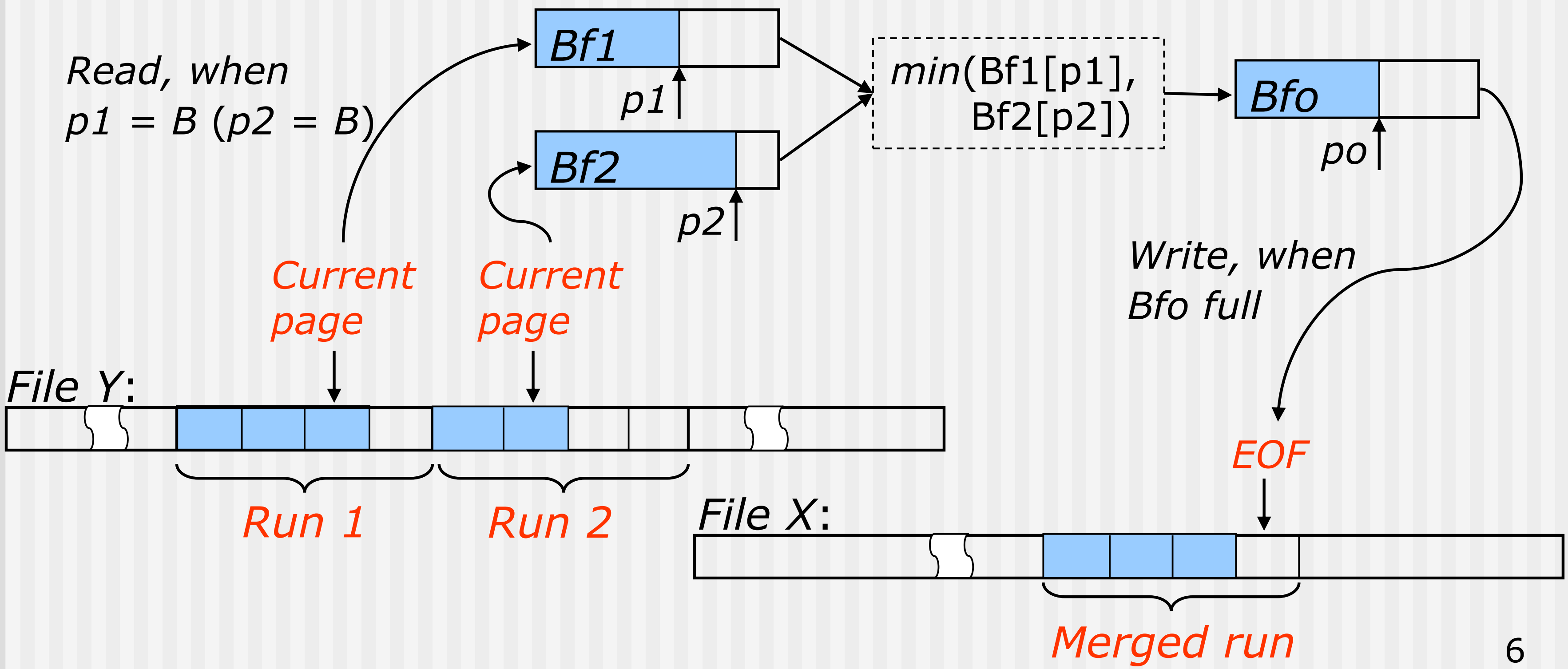
External-Memory Merge Sort

- 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

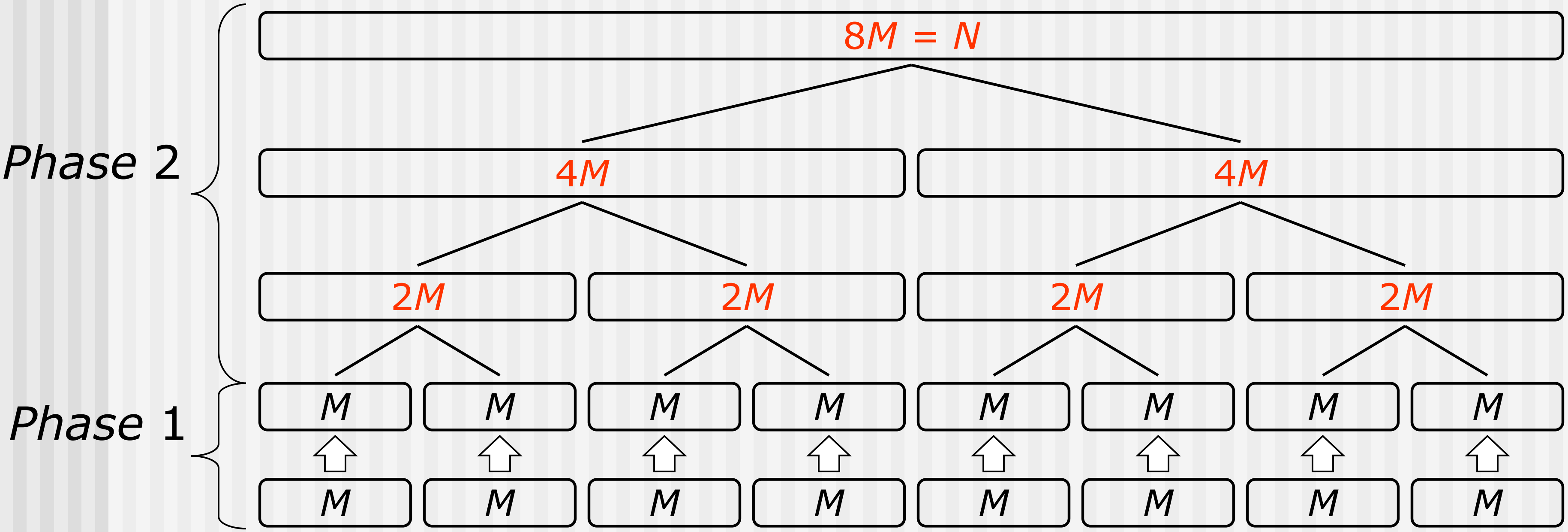
- *MergeAllRuns*(*Y*, *X*): repeat until the end of *Y*:
  - Call *TwowayMerge* to merge the next two runs from *Y* into one run, which is written at the end of *X*
- *TwowayMerge*: uses three main-memory arrays of size *B*



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



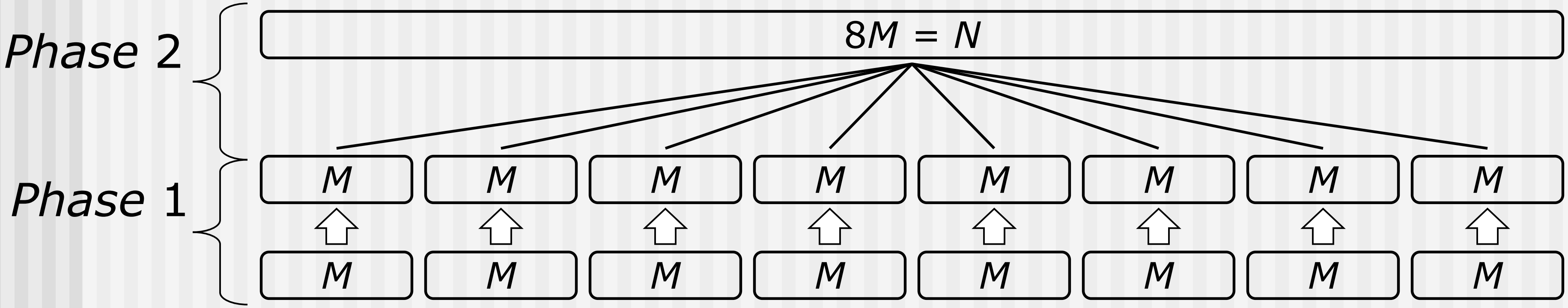
- Phase 1:
  - Read file X, write file Y: $2n = O(n)$ I/Os
- Phase 2:
  - One iteration: Read file Y, write file X: $2n = O(n)$ I/Os
  - Number of iterations: $\log_2 N/M = \log_2 n/m$

Analysis: Conclusions

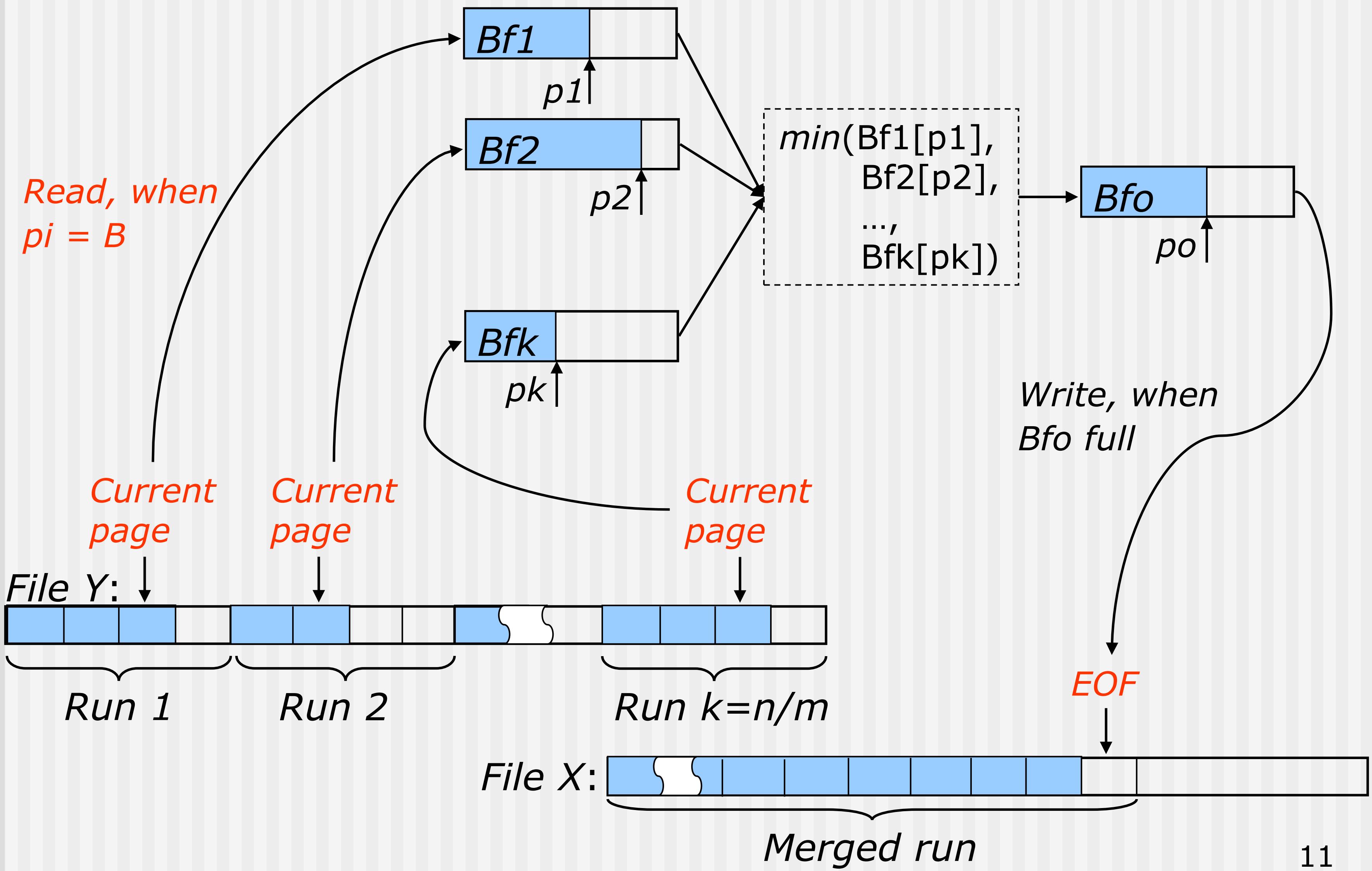
- Total running time of external-memory merge sort: $O(n \log_2 n/m)$
- We can do better!
- Observation:
  - Phase 1 uses all available memory
  - Phase 2 uses just 3 pages out of m available!!!

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

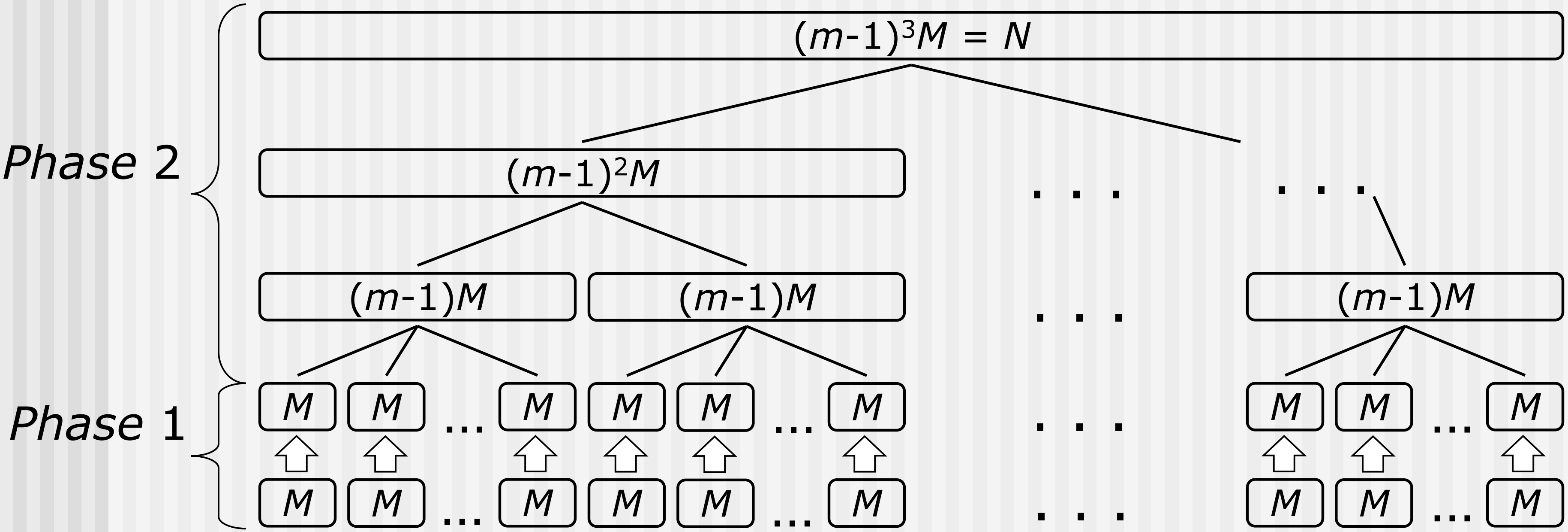
- What if a file is very large or memory is small?

- 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