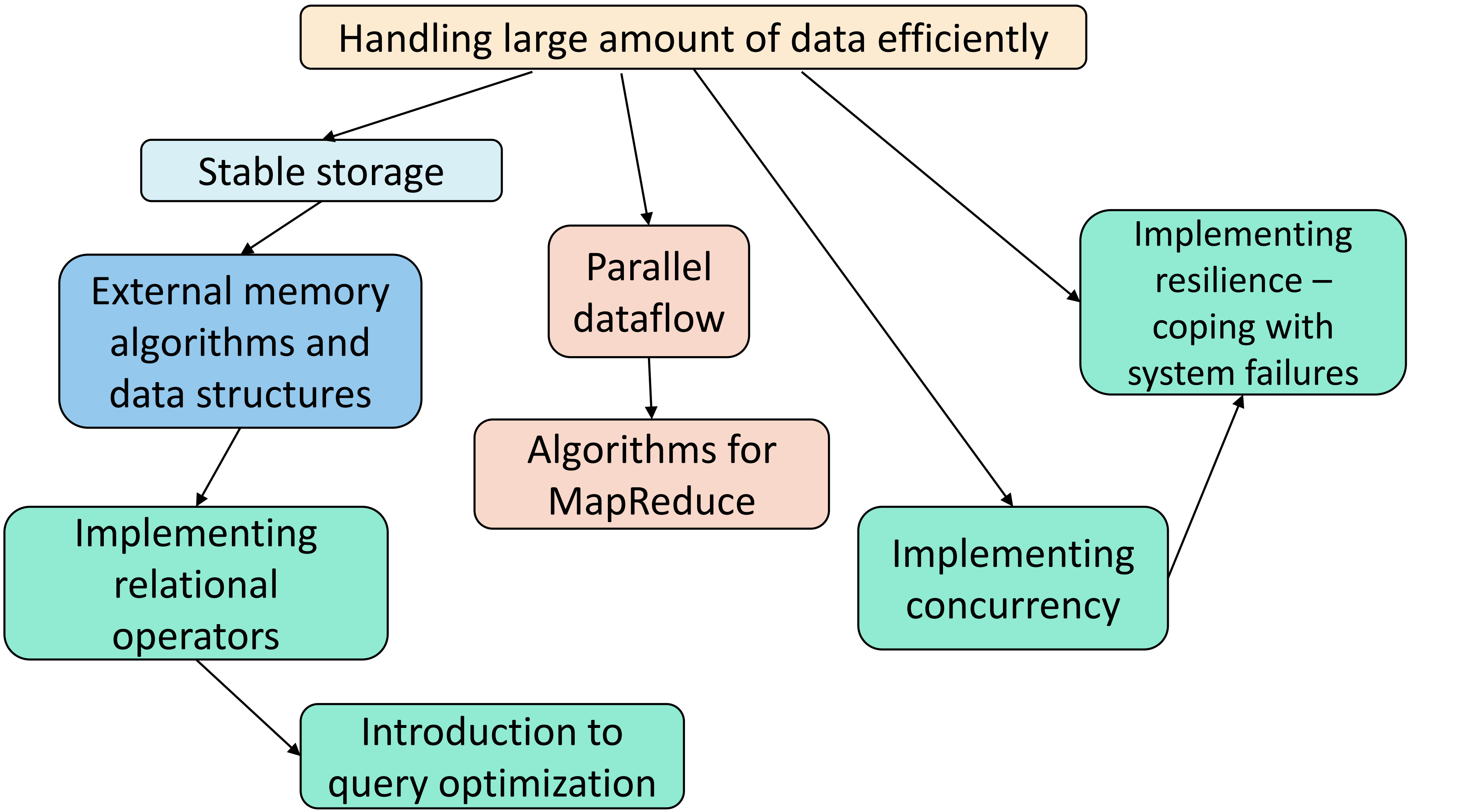
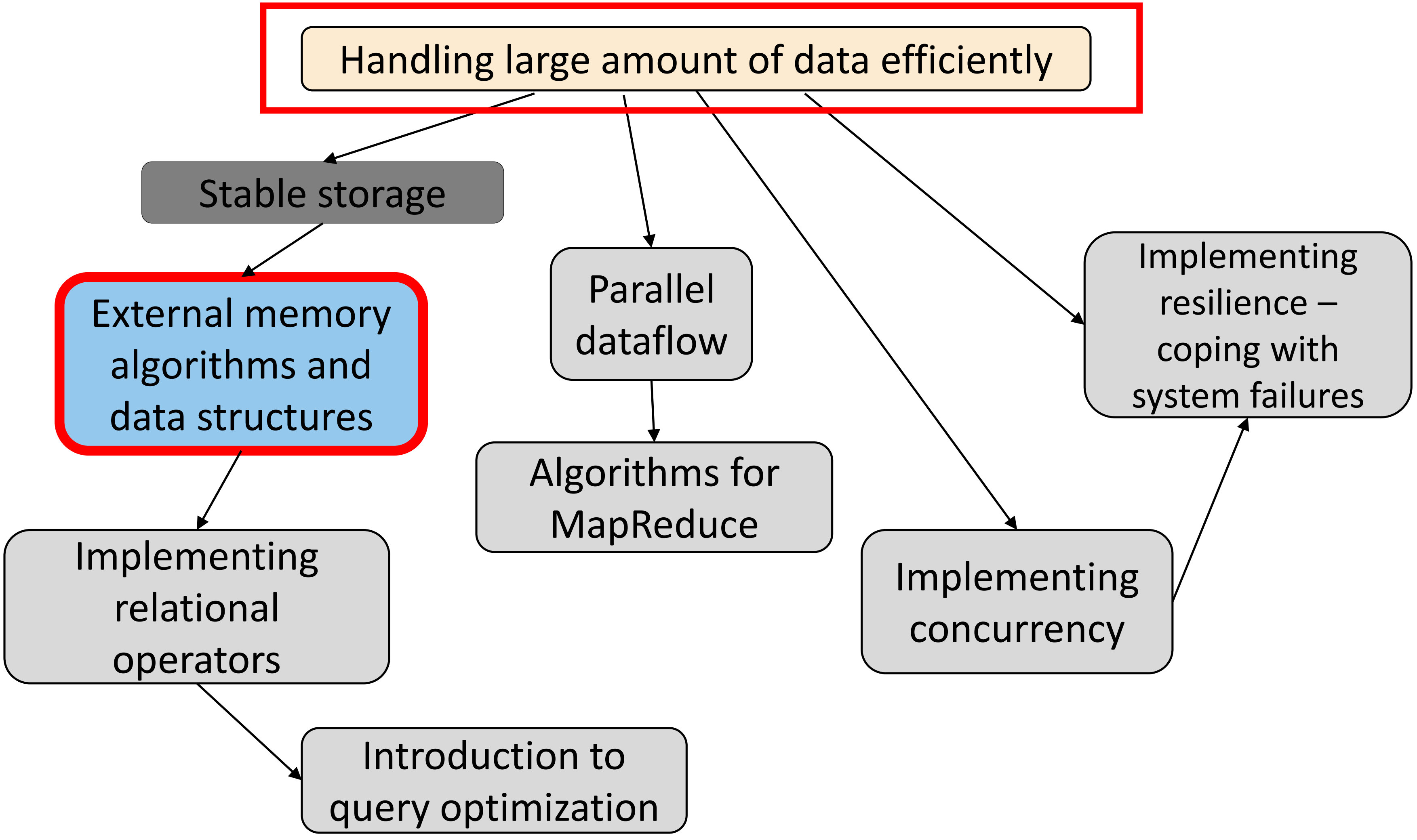
Roadmap



Handling large amount of data efficiently
Stable storage
External memory algorithms and data structures
Parallel dataflow
Algorithms for MapReduce
Implementing resilience – coping with system failures
Implementing relational operators
Implementing concurrency
Introduction to query optimization

Special algorithms for disks



Lecture 02.01

Algorithms for large inputs:

External-memory sorting

By Marina Barsky
Winter 2017, University of Toronto

Algorithms for external memory

- In most studies of algorithms, one assumes the ***"RAM model":***
  - Data is in main memory,
  - Access to any item of data takes as much time as any other.
- When implementing a DBMS, one must assume that the data does **NOT** fit into main memory.
- Often, the best algorithms for processing very large amounts of data differ from the best main-memory algorithms for the same problem.
- **There is a great advantage in choosing an algorithm that uses few disk accesses, even if the algorithm is not very efficient when viewed as a main-memory algorithm.**

I/O model of computation

- ***Disk I/O*** = read or write of a block is very expensive compared with what is likely to be done with the block once it arrives in main memory.
  Perhaps 1,000,000 machine instructions in the time to do one random disk I/O.

- The I/O model of computation measures the efficiency of an algorithm by counting how many disk reads and writes it needs.

- The unit of I/O is a **block**

- The model is oversimplified – no difference between consecutive reading of several blocks and random access

Best sorting algorithm?

- Common main-memory sorting algorithms don't look so good when you take disk I/O's into account.

- Variants of **Merge Sort** do better

In-memory merge sort

- ***Merge*** = take two sorted lists and repeatedly chose the smaller of the “heads” of the lists (head = first of the unchosen).
  - Example: merge 1,3,4,8 with 2,5,7,9 = 1,2,3,4,5,7,8,9.

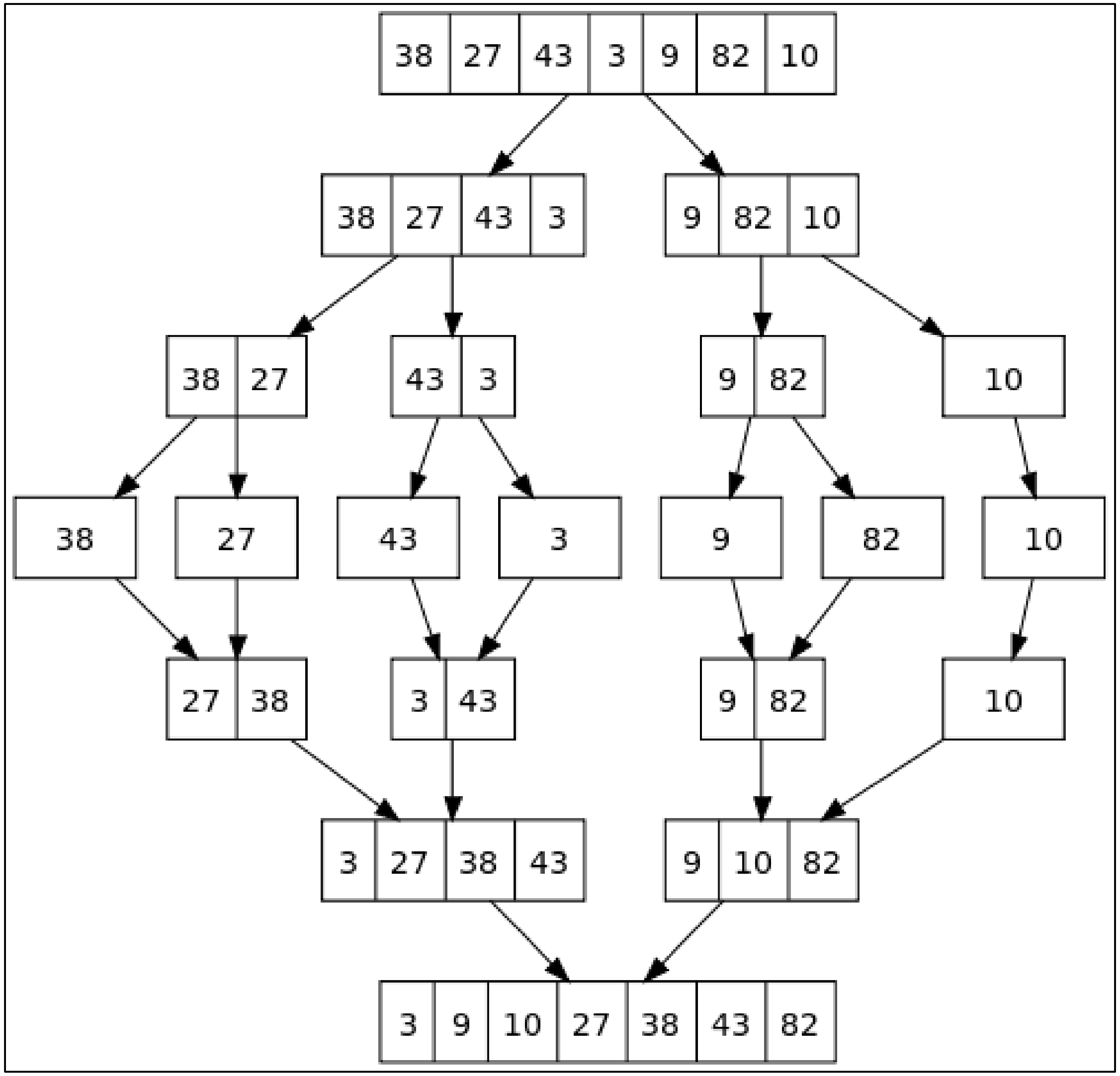
- Merge Sort is based on *recursive* algorithm:
  - divide records into two parts;
  - recursively mergesort the parts, and
  - merge the resulting lists.

Merge sort

```
algorithm mergesort (array A of size N)
    if ( N = 1 ) return A

    A1: = A[0 … N /2)
    A2: = A[N/2+1 … N)

    A1: = mergesort ( A1 )
    A2: = mergesort ( A2 )

    return merge (A1, A2)
```

```
merge (array X, array Y )
    result array Z
    i = 0  j = 0
    while ( i < |X| and j < |Y|)
        if ( X[i] > Y[j] )
            append Y[j] to Z
            j ++
        else
            append X[i] to Z
            i ++
    while (i < |X|)
        append X[i] to Z
        i ++
    while (j < |Y|)
        append Y[j] to Z
        j ++
```

2-way merge sort is not good enough for disk data!



- **If input is N blocks -**
- $\log_2 N$ passes - each block is read/written from disk $\log_2 N$ times during merge

- If data is on disk –

O (***N log N***) disk I/Os

Two-Phase, Multiway Merge Sort

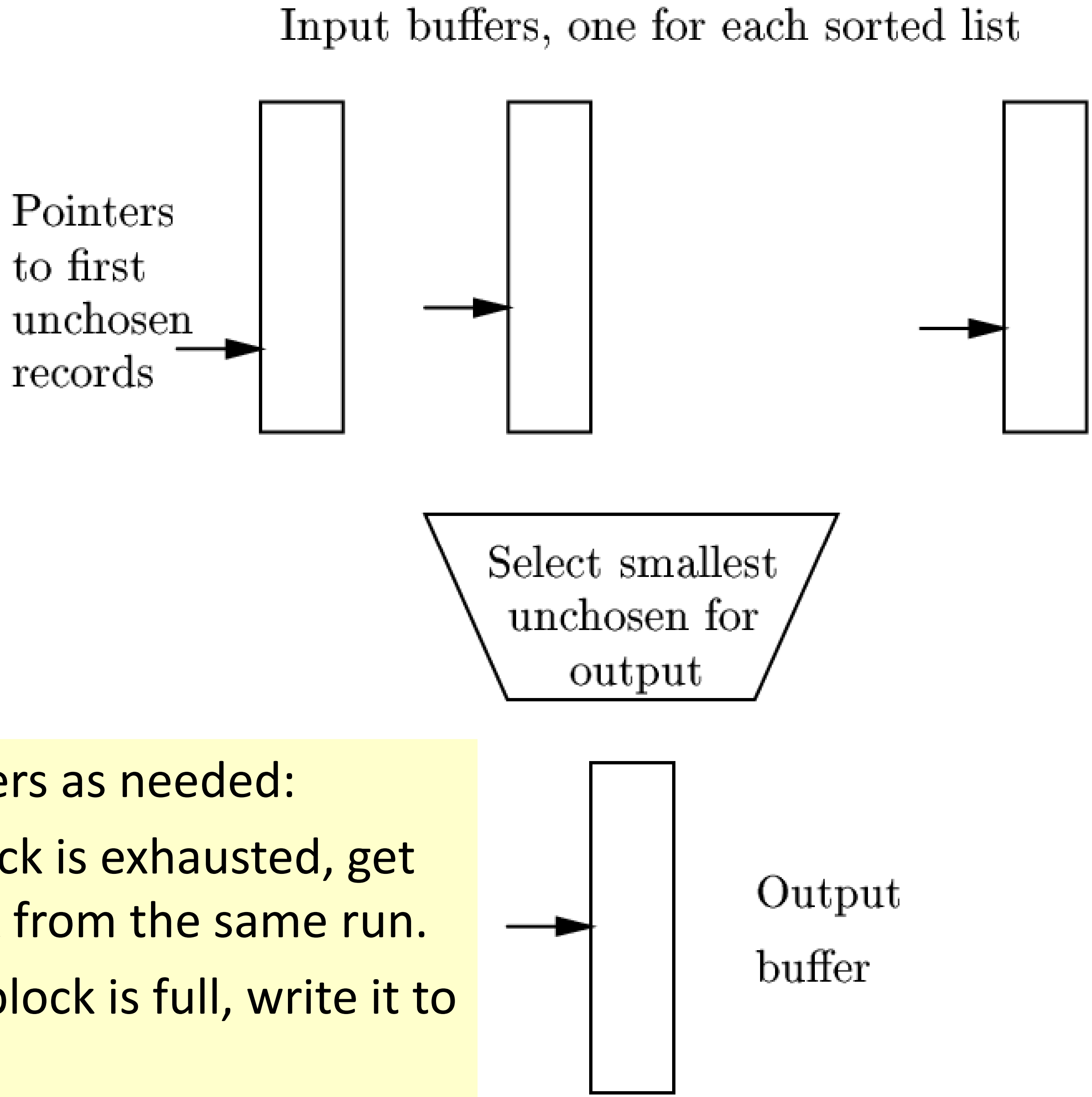
- The secondary-memory variant operates in a small number of *passes:*
  - in each pass every record is read into main memory **once** and written out to disk **once**.

- 2PMMS: 2 reads + 2 writes per block.

2PMMS: Phase 1

1. Fill main memory with blocks
2. Sort using favorite main-memory sort
3. Write sorted sublist to disk
4. Repeat until all records have been put into one of the sorted lists (*runs*)

2PMMS: Phase 2



- Manage the buffers as needed:
  - If an input block is exhausted, get the next block from the same run.
  - If the output block is full, write it to disk.

2PMMS: Toy Example

- 24 records with keys:

12 10 25 20 40 30 27 29 14 18 45 23 70 65 35 11 49 47 22 21 46 34 29 39

- Suppose 1 block can hold 2 records.
- Suppose main memory (MM) can hold 4 blocks i.e. 8 records.

Phase 1.

- Load 12 10 25 20 40 30 27 29 in MM, sort them and write the sorted sublist: 10 12 20 25 27 29 30 40
- Load 14 18 45 23 70 65 35 11 in MM, sort them and write the sorted sublist: 11 14 18 23 35 45 65 70
- Load 49 47 22 21 46 34 29 39 in MM, sort them and write the sorted sublist: 21 22 29 34 39 46 47 49

2PMMS example – Phase II

Phase 2.

On disk:

Sub-list 1: 10 12 20 25 27 29 30 40

Sub-list 2: 11 14 18 23 35 45 65 70

Sub-list 3: 21 22 29 34 39 46 47 49

Main Memory (4 buffers)

Input Buffer1:

Input Buffer2:

Input Buffer3:

Output Buffer:

Sorted list:

2PMMS example – Phase II

Phase 2.

On disk:

Sub-list 1: 10 12 20 25 27 29 30 40

Sub-list 2: 11 14 18 23 35 45 65 70

Sub-list 3: 21 22 29 34 39 46 47 49

Main Memory (4 buffers)

Input Buffer1:	10	12
Input Buffer2:	11	14
Input Buffer3:	21	22
Output Buffer:		

Sorted list:

2PMMS example – Phase II

Phase 2.

On disk:

Sub-list 1: 10 12 20 25 27 29 30 40

Sub-list 2: 11 14 18 23 35 45 65 70

Sub-list 3: 21 22 29 34 39 46 47 49

Main Memory (4 buffers)

Input Buffer1:	10	12
Input Buffer2:	11	14
Input Buffer3:	21	22
Output Buffer:	10	

Sorted list:

2PMMS example – Phase II

Phase 2.

On disk:

Sub-list 1: 10 12 20 25 27 29 30 40

Sub-list 2: 11 14 18 23 35 45 65 70

Sub-list 3: 21 22 29 34 39 46 47 49

Main Memory (4 buffers)

Input Buffer1:	10	12
Input Buffer2:	11	14
Input Buffer3:	21	22
Output Buffer:	10	11

Sorted list:

2PMMS example – Phase II

Phase 2.

On disk:

Sub-list 1: 10 12 20 25 27 29 30 40

Sub-list 2: 11 14 18 23 35 45 65 70

Sub-list 3: 21 22 29 34 39 46 47 49

Main Memory (4 buffers)

Input Buffer1:	10	**12**
Input Buffer2:	11	**14**
Input Buffer3:	**21**	22
Output Buffer:	**10**	**11**

Output Buffer full: flush to disk

Sorted list:

2PMMS example – Phase II

Phase 2.

On disk:

Sub-list 1: 10 12 20 25 27 29 30 40

Sub-list 2: 11 14 18 23 35 45 65 70

Sub-list 3: 21 22 29 34 39 46 47 49

Main Memory (4 buffers)

Input Buffer1:	10	**12**
Input Buffer2:	11	**14**
Input Buffer3:	**21**	22
Output Buffer:		

Sorted list: 10 11

2PMMS example – Phase II

Phase 2.

On disk:

Sub-list 1: 10 12 20 25 27 29 30 40

Sub-list 2: 11 14 18 23 35 45 65 70

Sub-list 3: 21 22 29 34 39 46 47 49

Main Memory (4 buffers)

Input Buffer1:	10	12
Input Buffer2:	11	14
Input Buffer3:	21	22
Output Buffer:	12	

Processed Input Buffer1 – upload from sub-list 1

Sorted list: 10 11

2PMMS example – Phase II

Phase 2.

On disk:

Sub-list 1: 10 12 20 25 27 29 30 40

Sub-list 2: 11 14 18 23 35 45 65 70

Sub-list 3: 21 22 29 34 39 46 47 49

Main Memory (4 buffers)

Input Buffer1:	20	25
Input Buffer2:	11	14
Input Buffer3:	21	22
Output Buffer:	12	

Sorted list: 10 11

2PMMS example – Phase II

Phase 2.

On disk:

Sub-list 1: 10 12 20 25 27 29 30 40

Sub-list 2: 11 14 18 23 35 45 65 70

Sub-list 3: 21 22 29 34 39 46 47 49

Main Memory (4 buffers)

Input Buffer1:	20	25
Input Buffer2:	11	14
Input Buffer3:	21	22
Output Buffer:	12	14

Processed Input Buffer2 – upload from sub-list 2

Output Buffer full: flush to disk

Sorted list: 10 11

2PMMS example – Phase II

Phase 2.

On disk:

Sub-list 1: 10 12 20 25 27 29 30 40

Sub-list 2: 11 14 18 23 35 45 65 70

Sub-list 3: 21 22 29 34 39 46 47 49

Main Memory (4 buffers)

Input Buffer1:	**20**	25
Input Buffer2:	**18**	23
Input Buffer3:	**21**	22
Output Buffer:		

We continue in this way until the sorted sub-lists are finished and we get on disk the whole sorted list of records.

Sorted list: 10 11 12 14 …