CSE235
database system
(Database Systems)
Lecture 03: Merging and Sorting Files

Professor in charge: Jeon Kang-wook (Department of Computer Engineering)

kw.chon@koreatech.ac.kr

* This lecture material is an edited version of the lecture material on file structure (written by Seok-ho Lee) provided by Jeongik Publishing.

Overview of Sort/Merge

- **Types of sorting: internal sort, external sort**

- **Internal alignment**
  - Used when there is little data and it can be stored all in main memory and sorted.
  - The time it takes to read and write records (I/O) is not an issue.

- **External alignment**
  - Used when arranging files stored in auxiliary storage devices because the data is large and exceeds the memory capacity.
  - The time for reading and writing records (I/O) is very important.
  - Sort/Merge
    - A file that is sorted using an internal sorting technique by dividing a single file into several subfiles.
    - Merge runs to produce a single, sorted file of your choice

Sort/Merge Files

- **Sorting step**
  - Split records of the file to be sorted into subfiles of specified length
  - The next step is to sort each one to create a run and distribute it to the input file.

- **Merger phase**
  - Merge sorted runs into larger runs
  - These runs are then redistributed back into the input file and merged, so that all records are ultimately included in one run.

Sorting step

- **How to create a run: How to divide the records in a file into multiple runs and sort them by key order**
  - Internal sort
  - replacement selection
  - natural selection

- **Example of input file (record key value)**

```
109 49 34 68 45 2 60 38 28 47 16 19 34 55 98
78 76 40 35 86 10 27 61 92 99 72 11 2 29 16 80
73 18 12 89 50 46 36 67 93 22 14 83 44 52 59
10 38 76 16 24 85
```

Internal alignment

- **How to create a run**
  - Split the file into groups of n records
  - Sorting the split records using an internal sorting technique

- **Run creation results**
  - All runs are the same length except the last one
  - Check the example when n=5 (main memory can hold 5 records)

Run 1: 34 45 49 68 109
Run 2: 2 28 38 47 60 Run
3: 16 19 34 55 98 Run 4:
35 40 76 78 86 Run 5: 10
27 61 92 99 Run 6: 2 11
16 29 72 Run 7: 12 18 73
80 89 Run 8: 36 46 50 67
93 Run 9: 14 22 44 52 83
Run 10: 10 16 38 59 76
Run 11: 24 85

Merger phase

- **How to perform a merge: How to merge multiple runs to create one aligned run**
  - m-way merge
  - balanced merge
  - polyphase merge
  - cascade merge

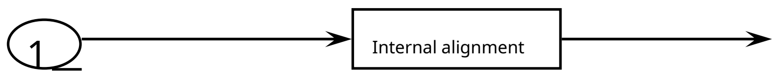
m-way merge

- **characteristic**
  - Merge multiple input files simultaneously
    - Run is more than m
  - Create one output file from m input files
    - Use a total of m+1 files
    - The number of input files is called the degree of the merge.
  - High I/O: If the merge does not finish in one pass**Need to copy and move runs to redistribute them to input files (requires multiple passes)**
  - Ideal Sort/Merge: Use m input files for m runs and complete with a single m-way merge.

- **In case of 2-won merger**
  - First pass: The size of the merged runs is doubled, and the number of runs is halved.
  - Number of passes for sorting a file divided into N runs:-log N-

2-Won Merge for 6 Runs

(1)Sorting step



1000Sorted by record
6Dog Run

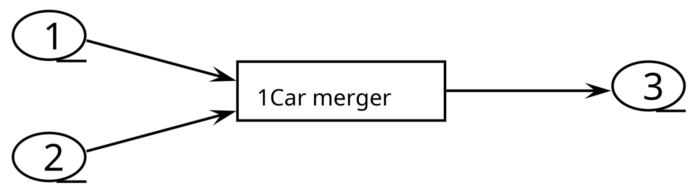
6000record

Sorted6Dog run2Distribute to input files of dogs

(2) 1Car merger

Run5 4001-5000 Run3 2001-3000 Run1 1-1000

Run6 5001-6000 Run4 3001-4000 Run2 1001-2000



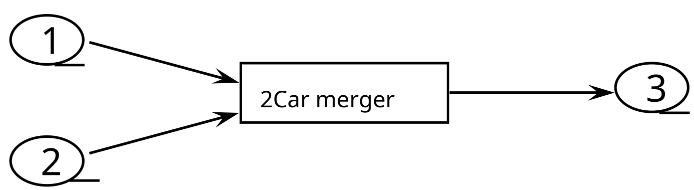
Run(1+2)
Run(3+4)
Run(5+6)

3In3Dog run2Distribute to input files of dogs

(3) 2Car merger

Run (5+6)Run(1+2)

Run(3+4)

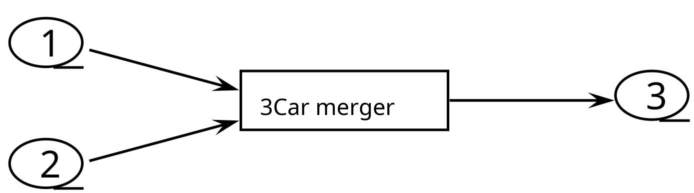


Run(1+2+3+4)
Run(5+6)

(4) 3Car merger

Run(1+2+3+4)

Run(5+6)



Run(1+2+3+4+5+6)

(2Dog's inputfile and 1 output file)

2-Won Merge for 12 Runs

(1)Sorting step

1 → Internal alignment → 500Sorted by record12Dog Run

6000record

Sorted12Dog run2Distribute to input files of dogs

(2) 1Car merger

Run11 Run9 ...... Run1 1

Run12 Run10 ...... Run2 2

1 → 1Car merger → 3

2 → 1Car merger

Run(1+2)

Run(3+4)

.....

Run(11+12)

3 In6Dog run2Distribute to input files of dogs

(3) 2Car merger

Run(9+10)Run(5+6)Run(1+2) 1

Run(11+12)Run(7+8)Run(3+4) 2

1 → 2Car merger → 3

2 → 2Car merger

Run(1+2+3+4)

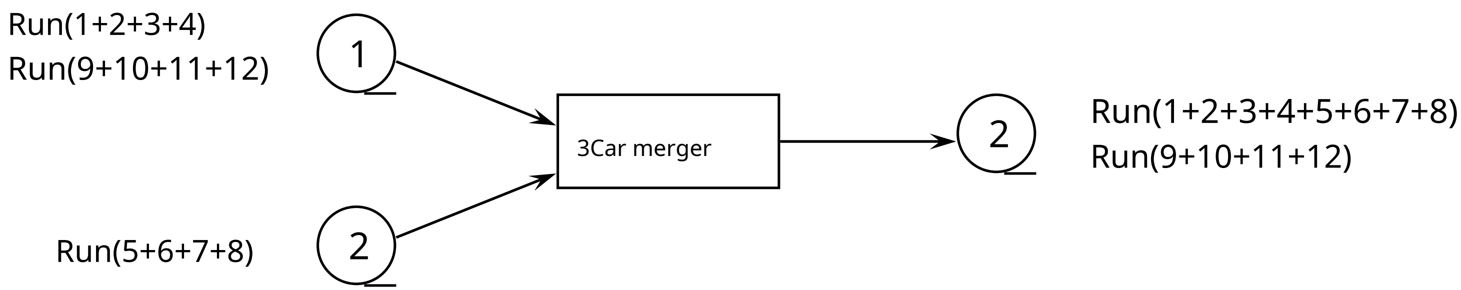
Run(5+6+7+8)

Run(9+10+11+12)

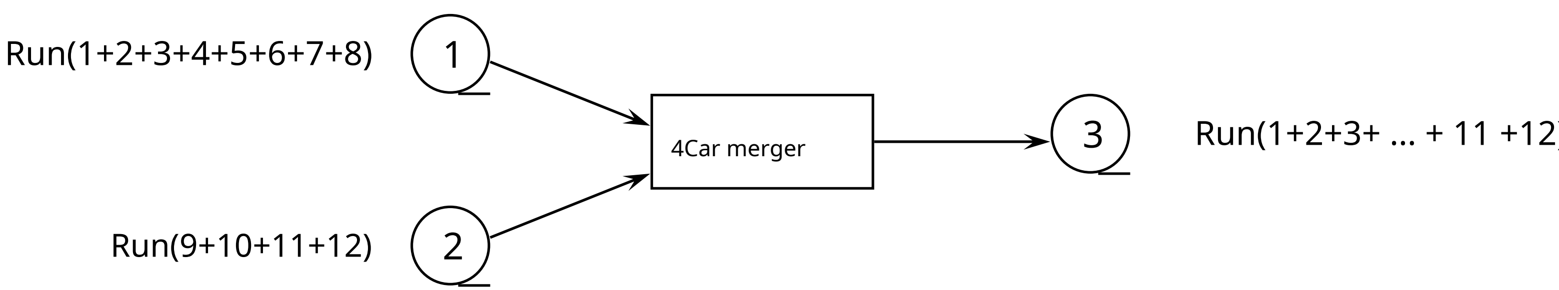
3In3Dog run2Distribute to input files of dogs

2-Way Merge for 12 Runs (Continued)

(4) 3Car merger

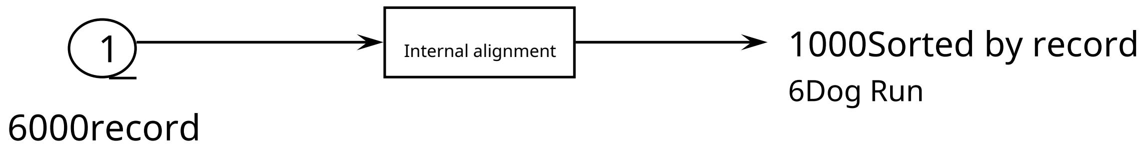


(5) 4Car merger



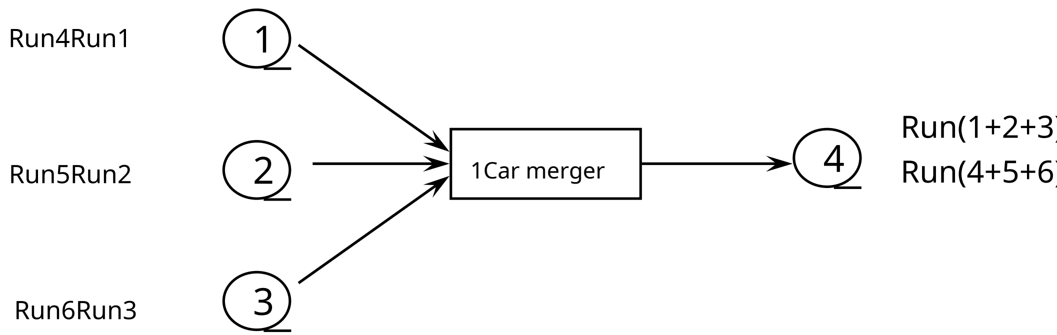
3-Way Merge for 6 Runs

(1)Sorting step



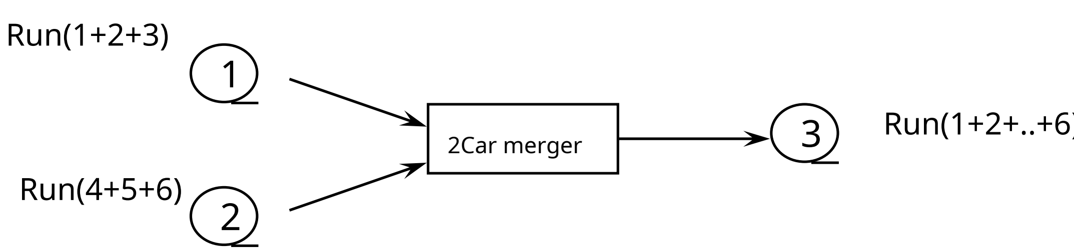
Sorted6Dog run3Distribute to input files of dogs

(2) 1Car merger



4In the run3Distribute to the input file of the dog (in this case,2(Only dog files are used)

(3) 2Car merger



(3The input file of the dog and1(dog output file)

Selection Tree

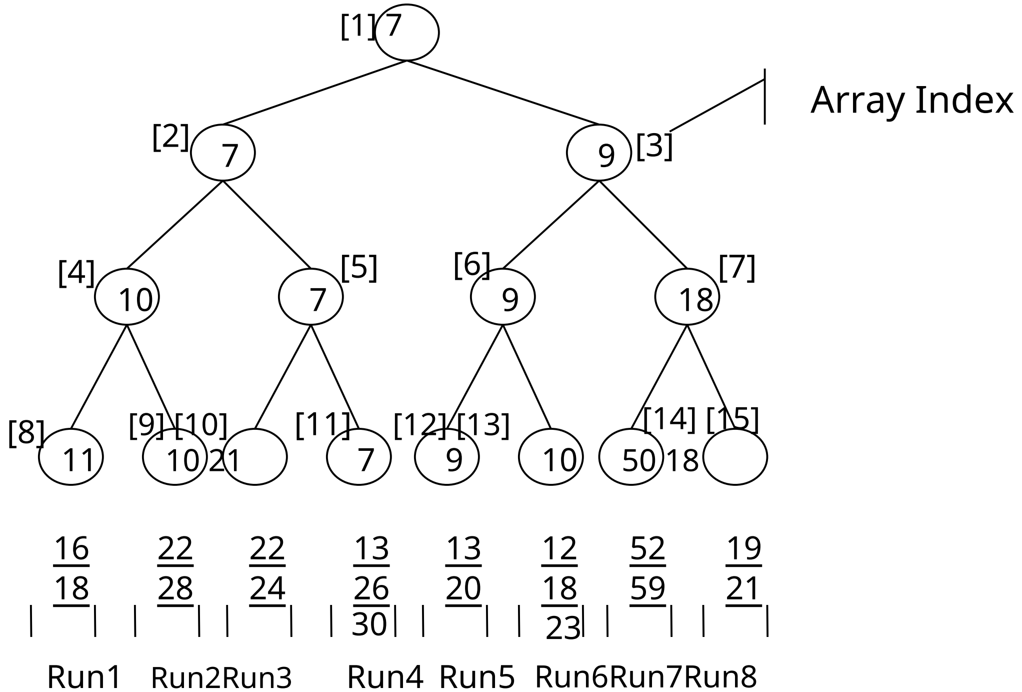
- **Sort m runs into one big run**
  - Continue selecting and outputting the record with the smallest key value among m runs
  - Simple method: m-1 comparison
  - Selection Tree: Can Reduce the Number of Comparisons
    - An efficient way to select the smallest value among m runs

- **Types of selection trees**
  - winner tree
  - loser tree

Winner Tree

- **characteristic**
  - Complete Binary Tree
  - Each terminal node represents the element with the minimum key value of each run.
  - An internal node represents the element with the smallest key value among its two children.

- **Example: Winner tree when there are 8 runs (k = 8)**

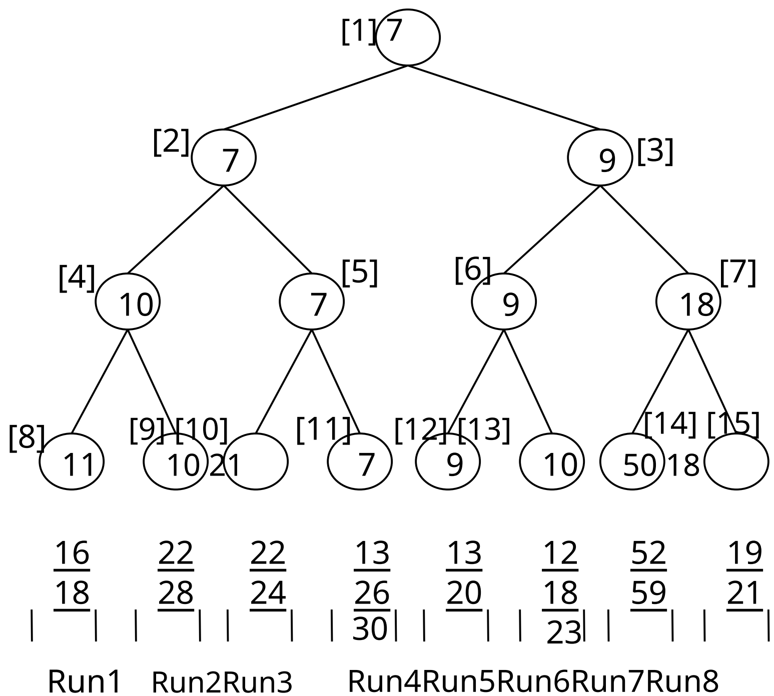


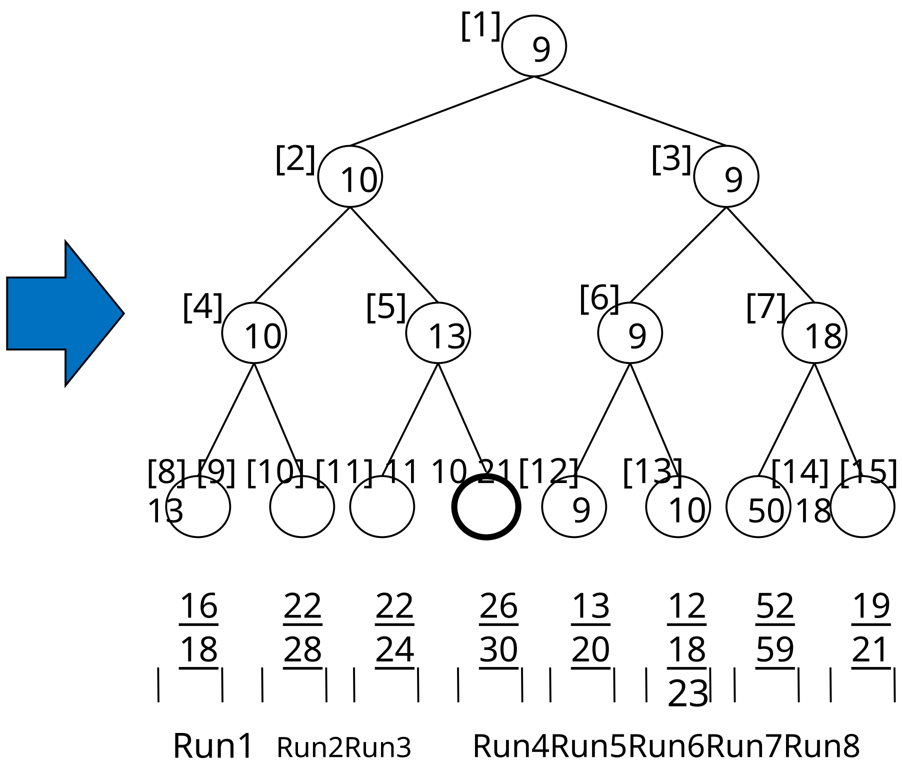
Winner Tree (continued)

- **Winner tree construction process**
  - Represented as a tournament game where the element with the smallest key value emerges as the winner.
  - For each internal node in the tree: a tournament winner of the two child node elements
  - Root node: The overall tournament winner, i.e. the node in the tree.**The smallest key value**Elements that have
- **Representation of the winner tree**
  - Since the winner tree is a complete binary tree, sequential representation is advantageous.
  - The two child indices of a node whose index value is i are 2i and 2i+1.
    - Very easy to express with an array
- **Progress of the merger**
  - Output sequentially in the order in which the roots are determined (7 in this example)
  - The next element, i.e. the element with key value 13, goes into the winner tree (13 goes in after 7 output)
  - Reconstruct the winner tree again
  - Example: Playing a tournament between sibling nodes along the path from node 11 (key value 13) to the root.

Winner Tree (continued)

- **Example: Reconstructing the winner tree**
  - Reconstruction after 7 is printed





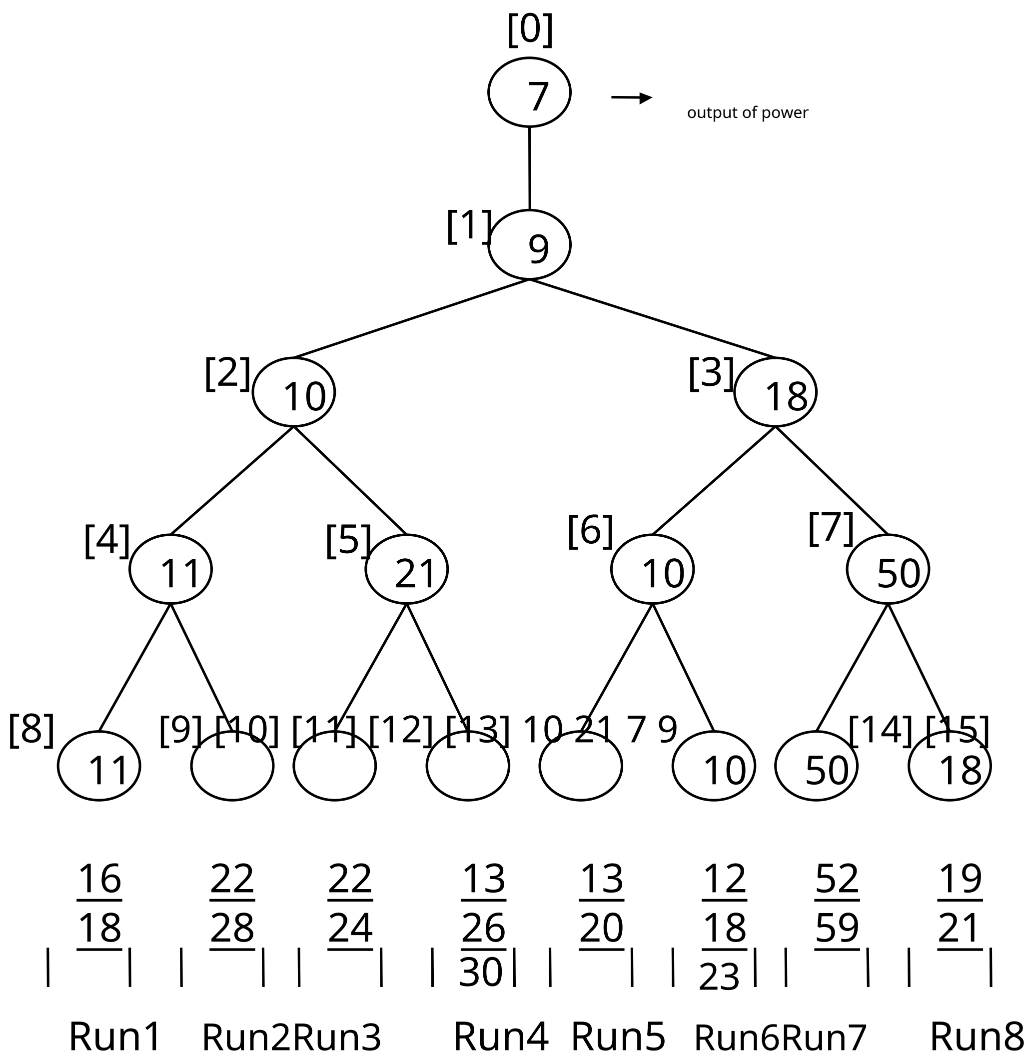
Loser Tree

- **A complete binary tree with a 0th node added above the root node.**

- **characteristic**
  - (1) Terminal node: The element with the minimum key value of each run.
  - (2) Internal nodes: Loser elements instead of winners of the tournament (see example in the next slide to understand)
  - (3) Root (Node 1): Loser of the final tournament
  - (4) Node 0: Overall winner (located separately above the root)

Loser Tree (continued)

- **Example of a loser tree: Example when there are 8 runs (k = 8)**

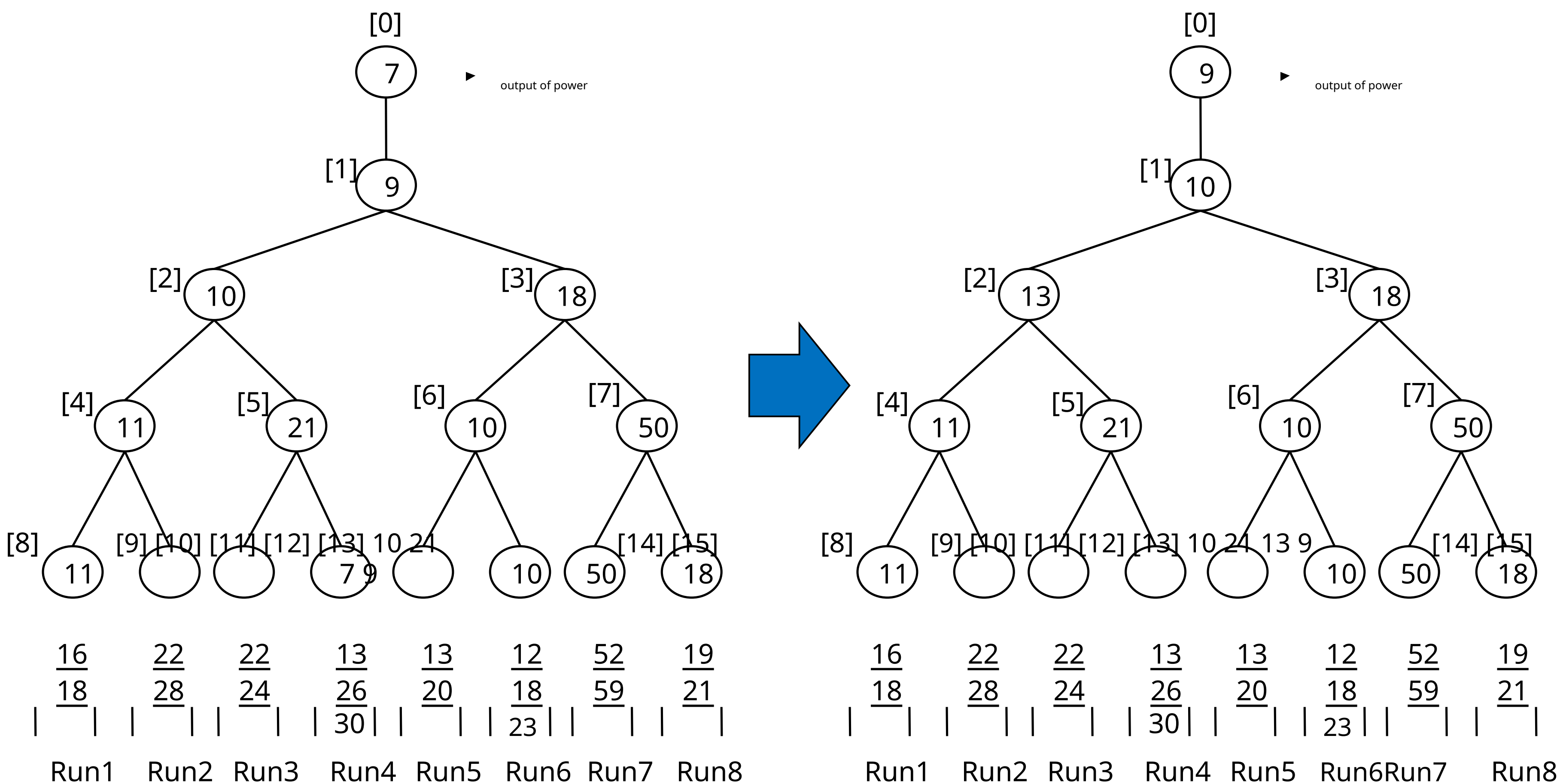


Loser Tree (continued)

- **Loser tree construction process**
  - Terminal node: the element with the minimum key value of each run.
  - Internal node
    - Two child nodes play tournament matches against their parent node.
    - Loser remains in parent node
    - The winner moves up to the parent node and continues the tournament play.
  - 1st root node
    - Likewise, the loser remains at the root node 1.
    - The winner is the winner of the entire tournament and goes up to node 0 and is output in order.
- **Progress of the merger**
  - In the example in the figure, the next element of run 4 to which the printed element (7) belongs, i.e. the element with key value 13, is inserted into node 11 of the loser tree.
  - Reconstruct the loser tree again
    - The tournament proceeds along the path from node 11 to root node 1.
    - However, the match is formally played against the parent node instead of the sibling node.

Loser Tree (continued)

- **Example of reconstruction of the loser tree (changes after 7 is printed)**



Balanced Merge

- **Disadvantages of basic m-one merger**
  - Redistribution of files during the merge process (1)→A lot of I/O occurs due to m)

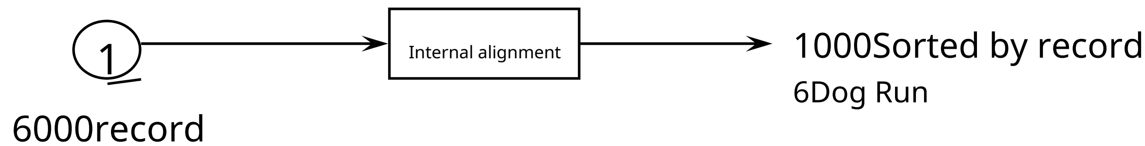
- **Balanced merger**
  - When outputting, redistribute the output file to the input file to be used in the next step in advance, i.e. use the output file directly as the input file for the next step.
  - m-source merge: requires m+1 files (m inputs, 1 output)
  - m-circle balance merge: requires 2m files (m inputs, m outputs)

- **After each merge step**
  - The total number of runs is reduced by the number of merges (m) (the number of runs decreases with each merge)
  - The length of the run increases by twice the merge order (m) (the length of the run increases with each merge)
  - $O(\log_m N)$, N = number of initial runs

Balanced Merge (continued)

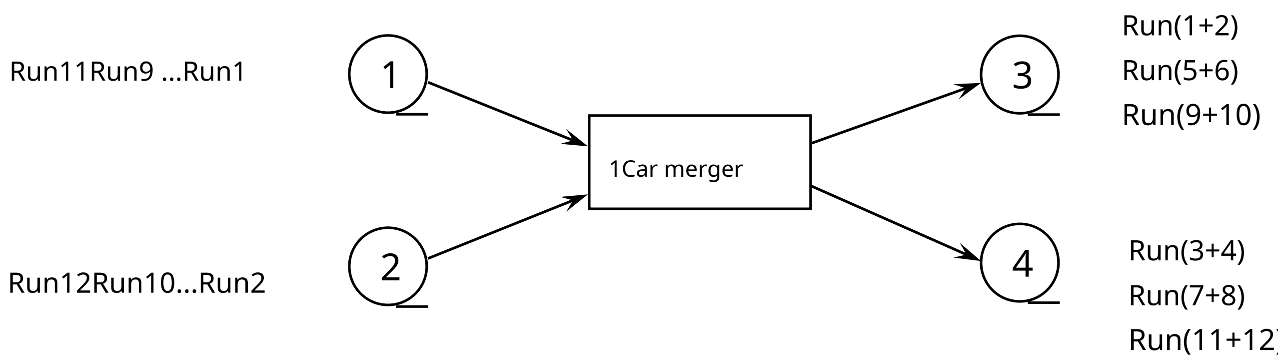
■ 2-Way Balanced Merge for 12 Runs

(1)Sorting step

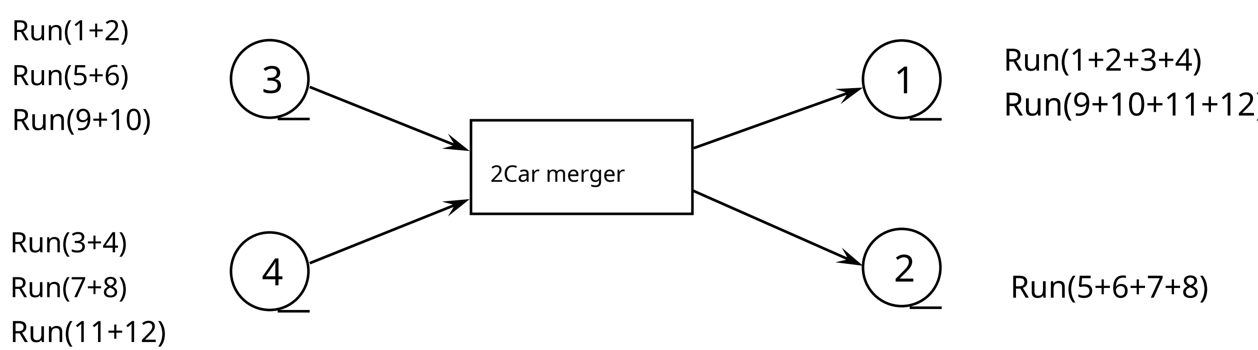


Generated12Dog run2Distribute to dog files

(2) 1Car merger



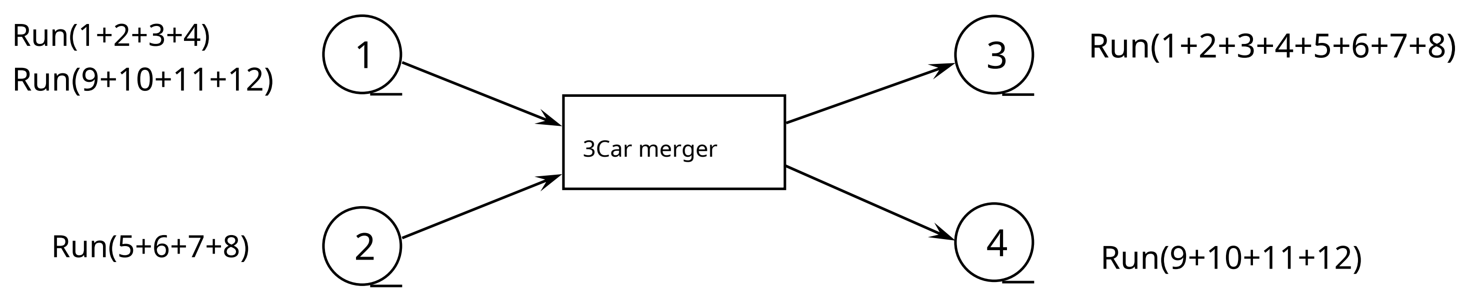
(3) 2Car merger



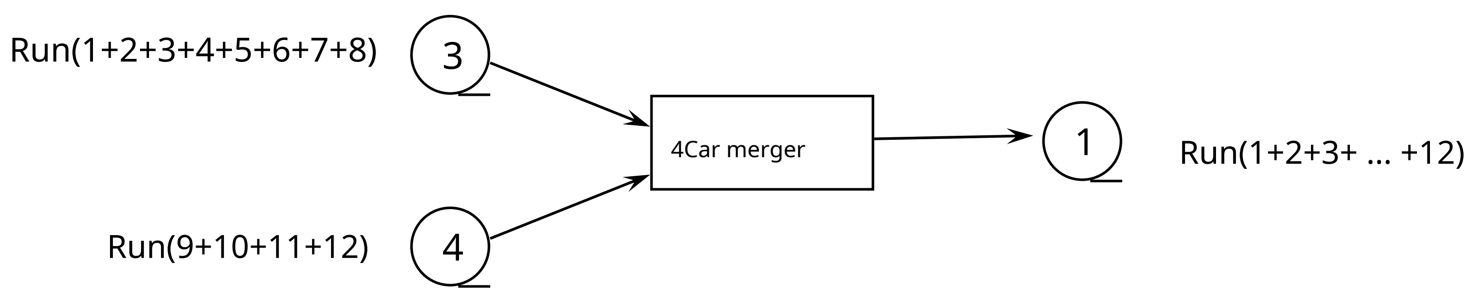
Balanced Merge (continued)

- **2-Way Balanced Merge for 12 Runs (Continued)**

(4) 3Car merger



(5) 4Car merger



caution Run(1)silver4Read/Write.

Polyphase Merge

- **Considerations on m-circle equilibrium merger**
  - **Requires 2m files (m inputs, m outputs)**
  - Cons: (Out of the output files) m-1 files are always idle
  - To address the file idle issue, improve the simple copy operation of the run.

- **m-one multi-level merger**
  - **m input files and 1 output file (same as basic m-one merge)**
  - Number of input/output files are not equal: “Unbalanced” merge
  - Distribution of runs for the initial input file is complex.

- **Each merge step (pass)**
  - of the input file**Merge runs until one is blank**
  - **The blank input file becomes the output file for the next merge step.**This is it

Polyphase Merge (continued)

- **Example of a 2-way multi-step merger**

50 110 95 15 100 30 150 40 120 60 70 130 20 140 80

File 1: | 50 95 110 | 40 120 150 | 20 80 140 |

File 2: | 15 30 100 | 60 70 130 |

File 3: gap

(a) Results of the sorting step

File 1: 20 80 140

File 2: gap

File 3: | 15 30 50 95 100 110 | 40 60 70 120 130 150 |

(b) Results of the first merger

File 1: gap

File 2: 15 20 30 50 80 95 100 110 140

File 3: 40 60 70 120 130 150

(c) Results of the second merger

File 1: 15 20 30 40 50 60 70 80 95 100 110 120 130 140 150

File 2: gap

File 3: gap

(d) Results of the 3rd merger

Polyphase Merge (continued)

- **Change in the number of runs in a two-way multi-stage merger**

- **Only one input file is left blank for each merge step.**

	array step	1car merger	2car merger	3car merger
file1	3	1	0	1
file2	2	0	1	0
file3	0	2	1	0
Run total	5	3	2	1

Polyphase Merge (continued)

- **Distribution method of initial runs: Fibonacci sequence**

- **Change in number of runs**
  - In case of 2-way merger: 1, 1, 2, 3, 5, 8, 13, …
  - In case of 3-way merge: [1, 1, 1], 3, 5, 9, 17, 31, …

- **Fibonacci sequence (m = 3)**
  - $T_i = T_{i-1} + T_{i-2} + T_{i-3},\ i > 3$
  - $T_i = 1,\ i \leq 3$

- **Fibonacci sequence (general form)**
  - $T_i = 1\ ,\ i \leq m$
  - $T_i = \sum_{k=i-m}^{i-1} T_k,\ i-m$

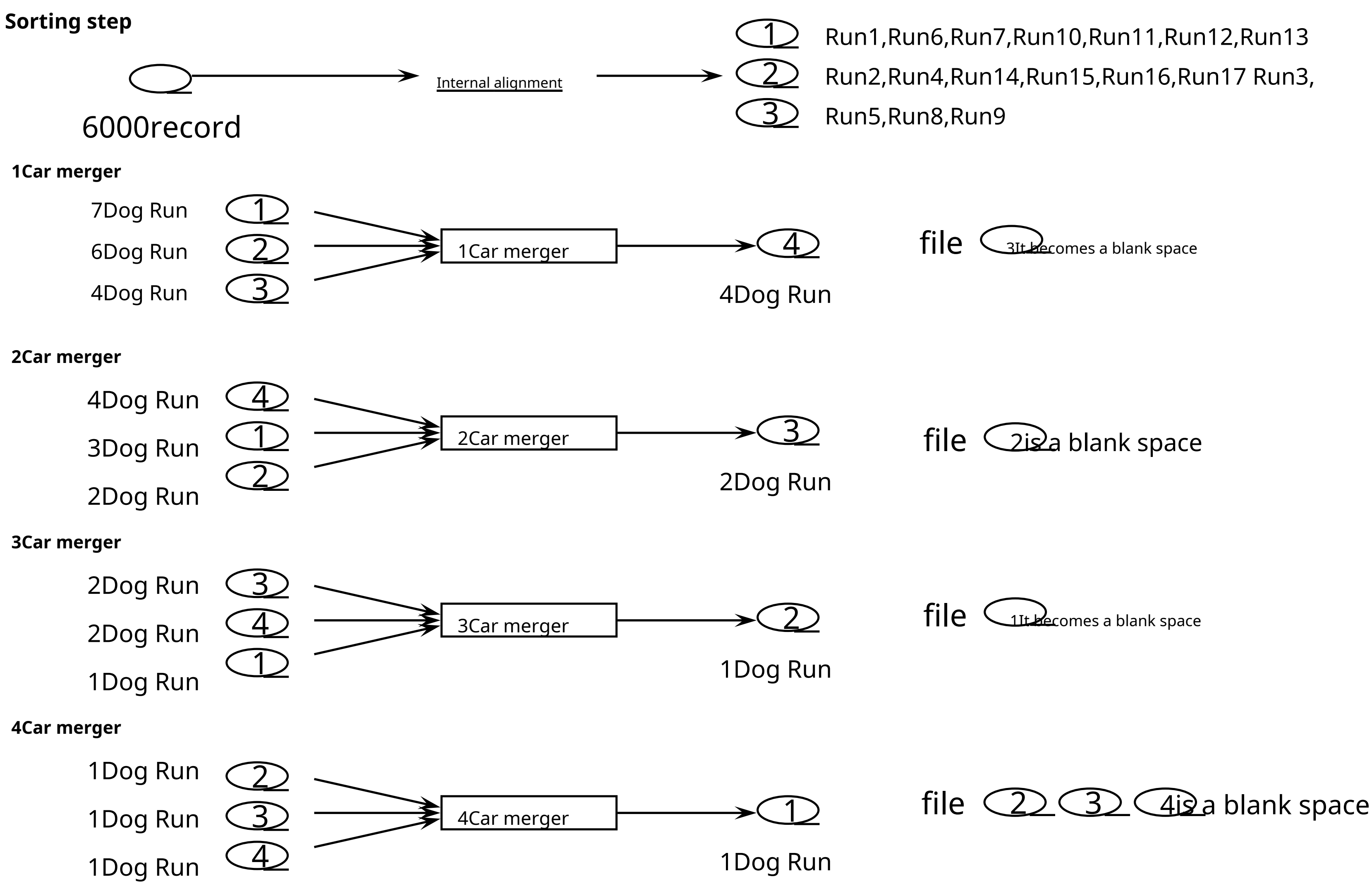
Polyphase Merge (continued)

- **Distribution method of initial runs: Fibonacci sequence**
- **Add the number displayed in a circle to another file in the next step.**

	1car	2car	3car	4car
File1	1	1	3	7
File2	1	2	2	6
File3	1	2	4	4
Run number	3	5	9	17

Polyphase Merge (continued)

- **3-Way Multi-Stage Merger**



Polyphase Merge (continued)

- **Change in number of runs per file at each merge step (m=3, number of files=17)**
- **Reverse of the complete Fibonacci distribution method for the initial run**

	Sort by total	1Tea set party	2Tea set party	3Tea set party	4Tea set party
file1	7	3	1	0	1
file2	6	2	0	1	0
file3	4	0	2	1	0
file4	0	4	2	1	0
Run sum account	17	9	5	3	1

Polyphase Merge (continued)

- **Example of a 3-way multi-level merger**

50 110 95 15 100 30 150 40 120 60 70 130 20 140 80

file1 : 50 110 95
file2 : 15 30 100 | 60 70 130
file3 : 40 120 150 | 20 80 140
file4 : gap

(a)Sorting step results

file1 : gap
file2 : 60 70 130
file3 : 20 80 140
file4 : 15 30 40 50 95 100 110 120 150

(b)merger1to give1Results of the car merger

file1 : 20 60 70 80 130 140
file2 : gap
file3 : gap
file4 : 15 30 40 50 95 100 110 120 150

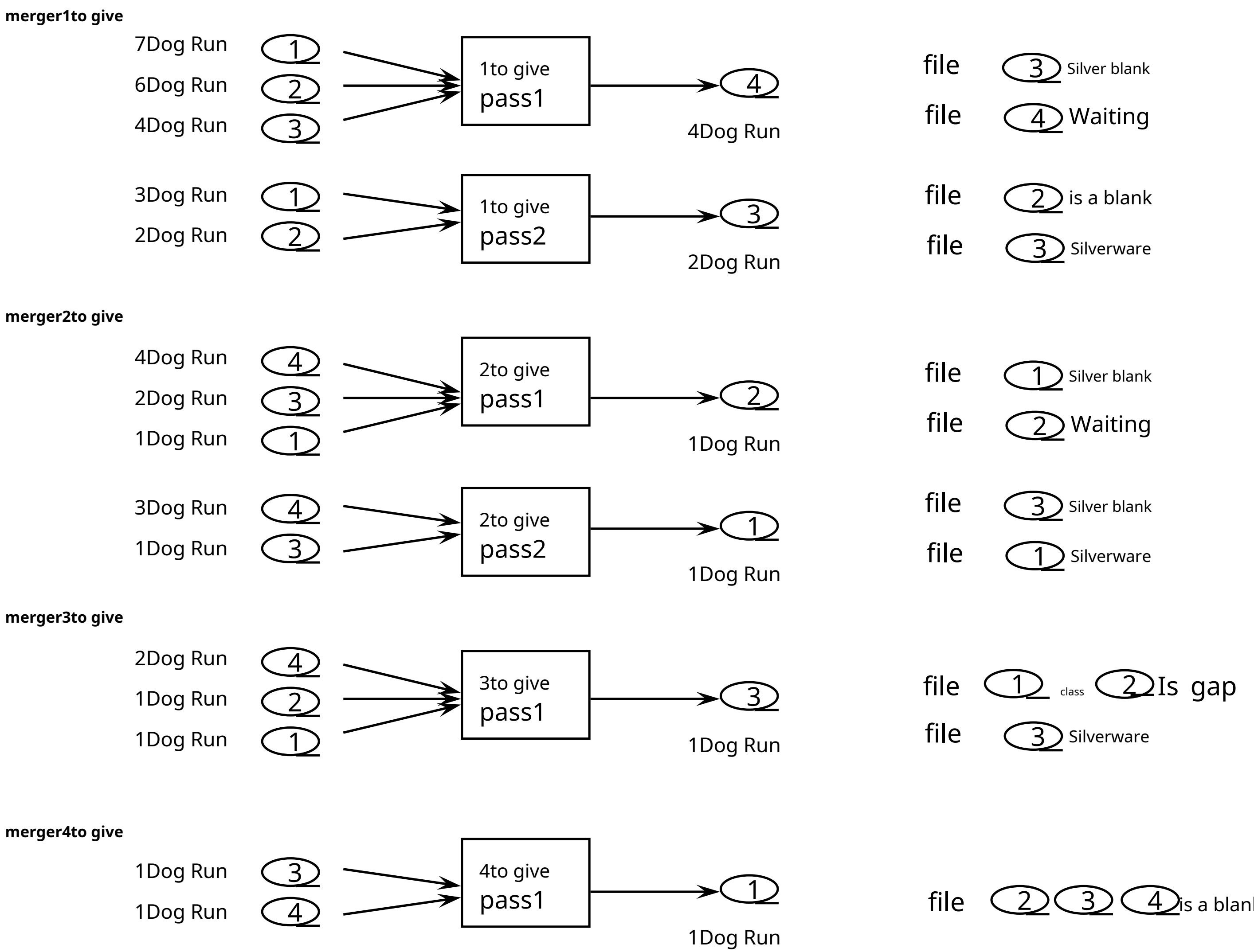
(c)merger1to give2Results of the car merger

file1 : gap
file2 : 15 20 30 40 50 60 70 80 95 100 110 120 130 140 150
file3 : gap
file4 : gap

(d)merger2to give1Results of the car merger

Polyphase Merge (continued)

- **Example of a 3-way multi-level merger**



Cascade Merge

- **Another form of unbalanced merging to reduce copying of runs during sorting/merging.**

- **m-way cascade merge**
  - Period: m, m-1, m-2, ..., and lastly, use 2 input files
  - Run Creation Phase: (as with multi-stage merges) the initial distribution of runs is important.
  - Merger phase
    - Merge m runs from m input files into one run and create an output file.
    - **The first blank input file becomes the new output file.**This is it
    - Merge m-1 input files into this new output file.
    - When the step of merging two input files is reached, one cycle of merging is completed.
    - Each record is processed once per cycle.

Cascade Merge (continued)

- **3-Way Stepwise Merger**

50 110 95 15 100 30 150 40 120 60 70 130 20 140 80

file1 :	50 110 95
file2 :	15 30 100 | 60 70 130
file3 :	40 120 150 | 20 80 140
file4 :	gap

(a)Sorting step results

file1 :	gap
file2 :	60 70 130
file3 :	20 80 140
file4 :	15 30 40 50 95 100 110 120 150

(b)merger1to give1Results of the car merger

file1 :	20 60 70 80 130 140
file2 :	gap
file3 :	gap
file4 :	15 30 40 50 95 100 110 120 150

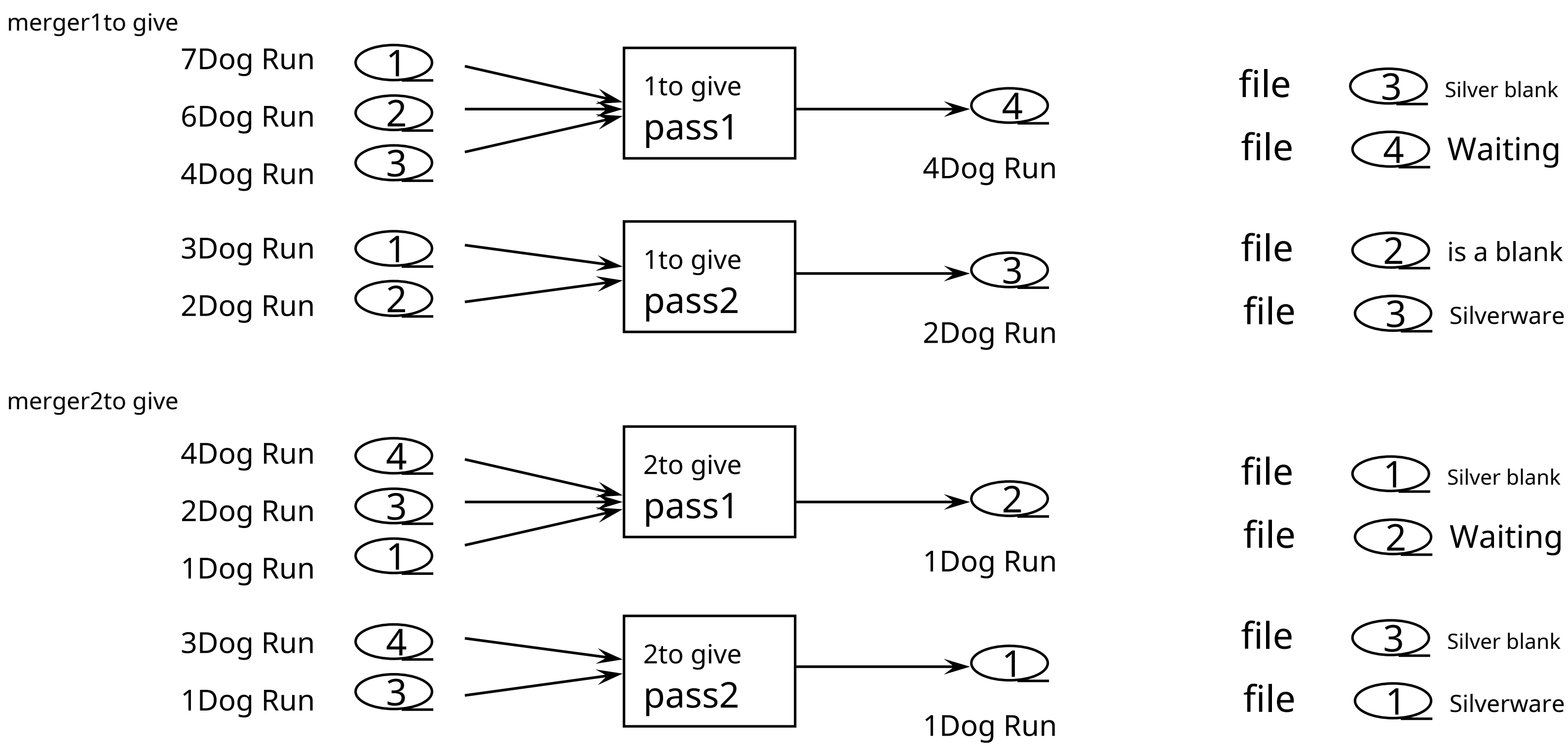
(c)merger1to give2Results of the car merger

file1 :	gap
file2 :	15 20 30 40 50 60 70 80 95 100 110 120 130 140 150
file3 :	gap
file4 :	gap

(d)merger2to give1Results of the car merger

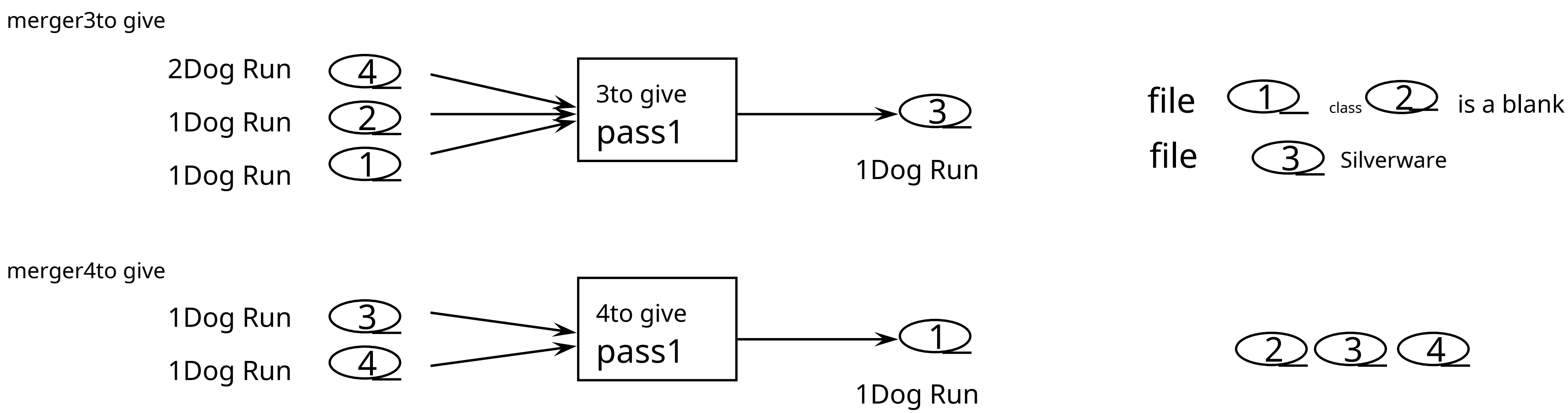
Cascade Merge (continued)

- **Change in number of runs in step merge (m=3, number of runs=17)**



Cascade Merge (continued)

- **Change in number of runs in stepwise merge (m=3, number of runs=17) (continued)**



thank you!

Professor in charge: Jeon Kang-wook (Department of Computer Engineering)

kw.chon@koreatech.ac.kr