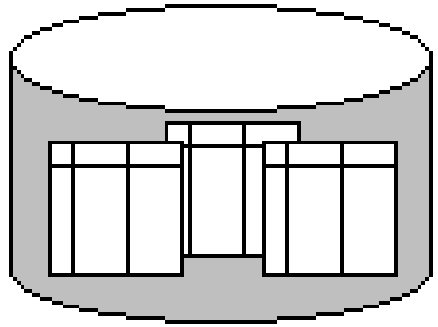
ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

Συστήματα Διαχείρισης Βάσεων Δεδομένων

Φροντιστήριο 5: Tutorial on External Sorting

Δημήτρης Πλεξουσάκης
Τμήμα Επιστήμης Υπολογιστών

TUTORIAL ON EXTERNAL SORTING

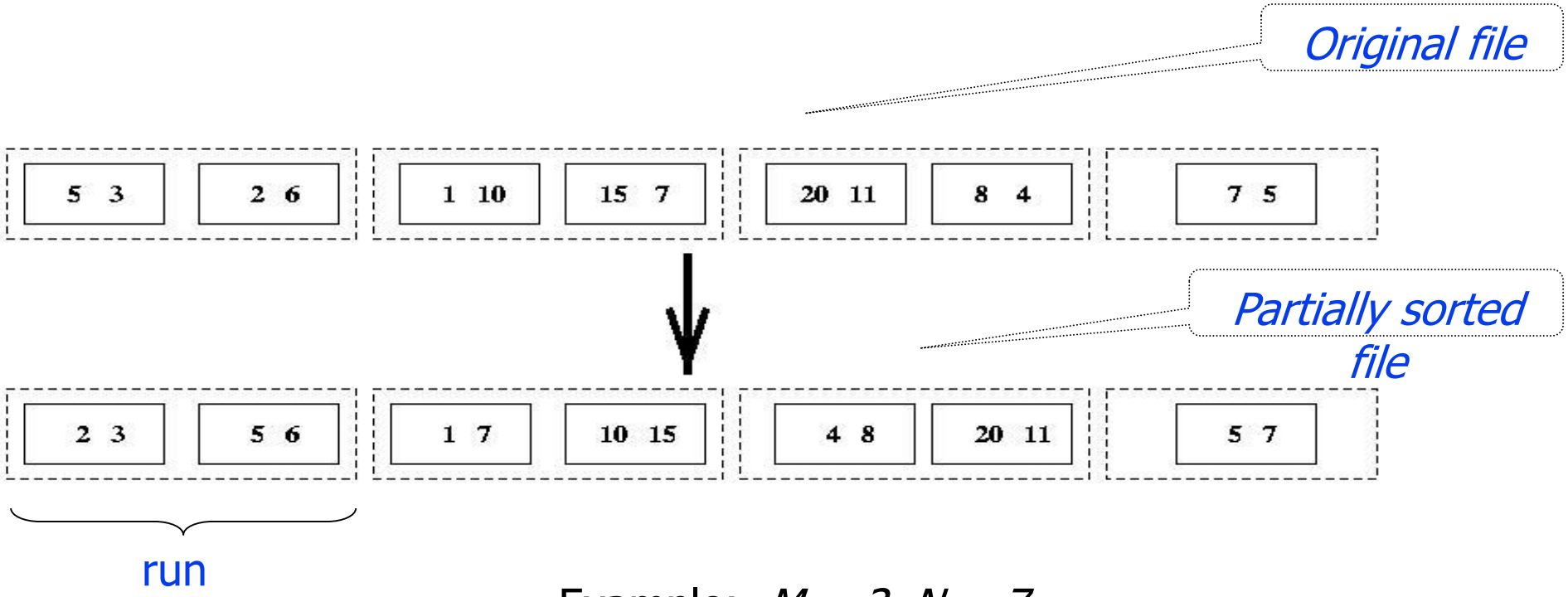


External Sorting

- External sorting has two main components:
  - Computation involved in sorting records in buffers in main memory
  - I/O necessary to move records between mass store and main memory

General Merge Sort Algorithm

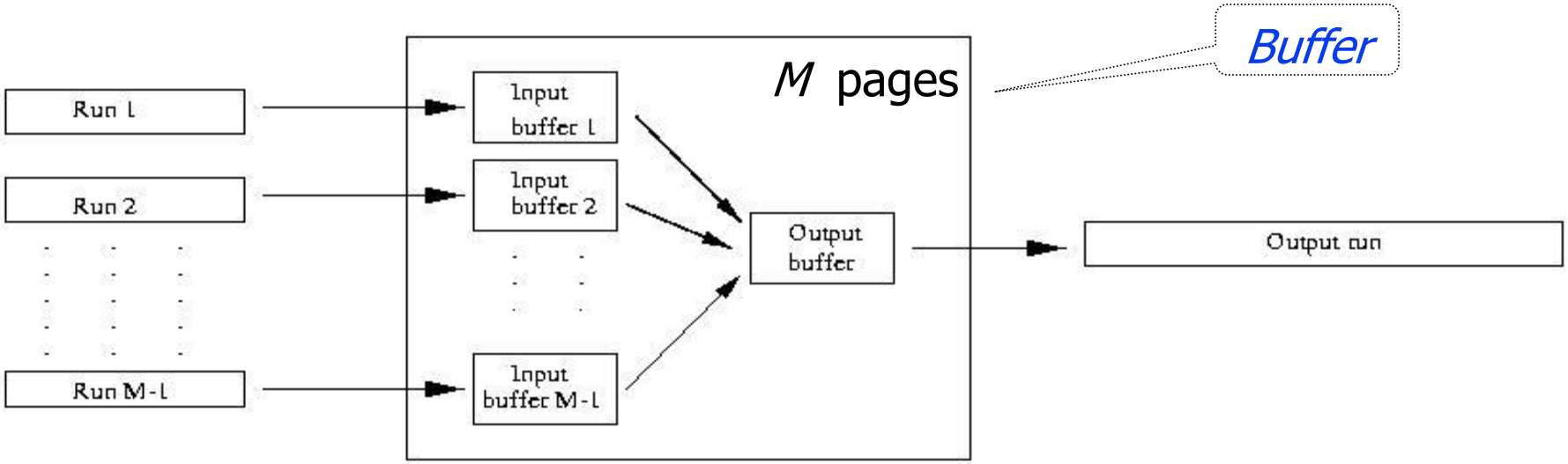
- *M* = number of main memory page buffers
- *N* = number of pages in file to be sorted
- Typical algorithm has two phases:
  - Partial sort phase: sort *M* pages at a time; create N/*M* sorted runs on mass store, cost = *2N*



Example: *M = 2, N = 7*

General Merge Sort Algorithm

- Merge Phase: merge all runs into a single run using M-1 buffers for input and 1 output buffer
  - Merge step: divide runs into groups of size M-1 and merge each group into a run; cost = 2N
  - each step reduces number of runs by a factor of M-1



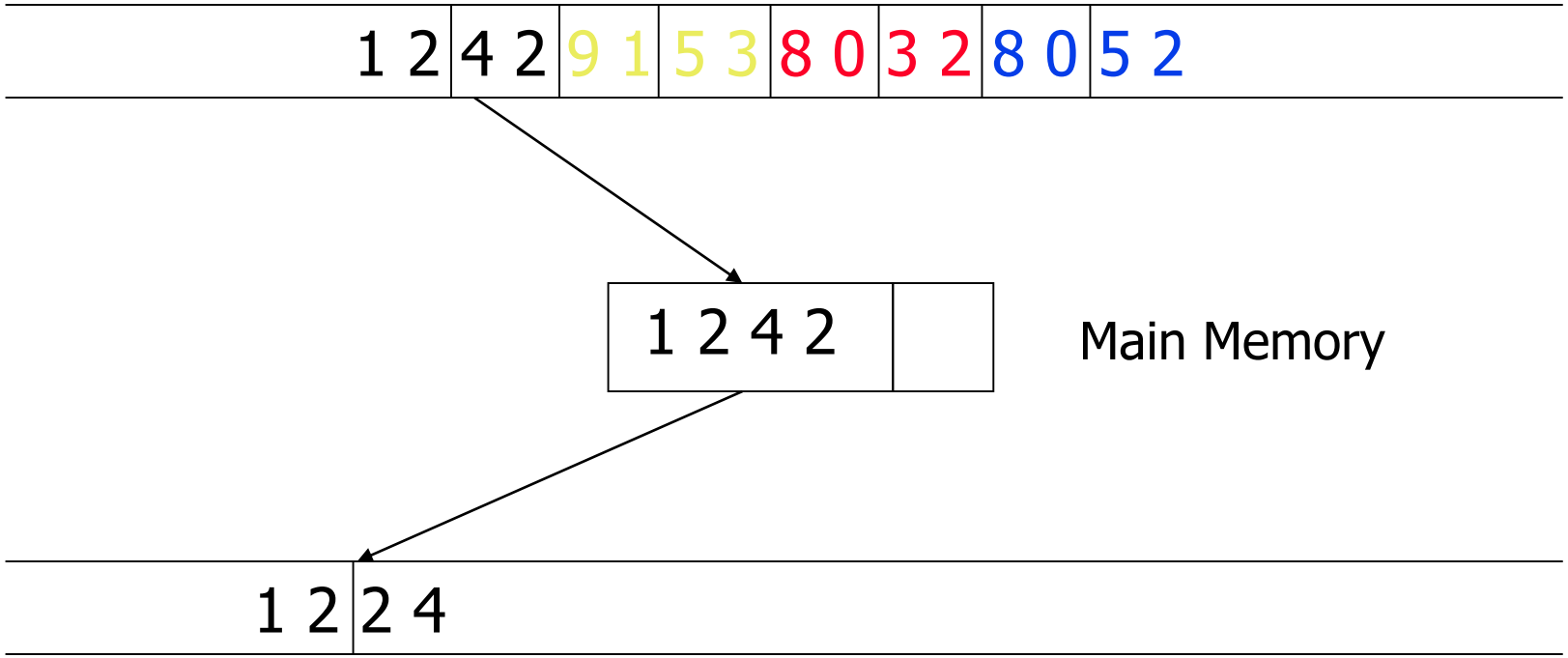
General Merge Sort Algorithm

- Cost of merge phase:
  - $(N/M)/(M-1)^k$ runs after k merge steps
  - $\lceil \log_{M-1}(N/M) \rceil$ merge steps needed to merge an initial set of N/M sorted runs
  - $cost = \lceil 2N \text{ Log}_{M-1}(N/M) \rceil \approx 2N(\log_{M-1}N - 1)$
- Total cost= cost of partial sort phase + cost of merge phase $\approx 2N \log_{M-1}N$

Merge Sort: Phase 1

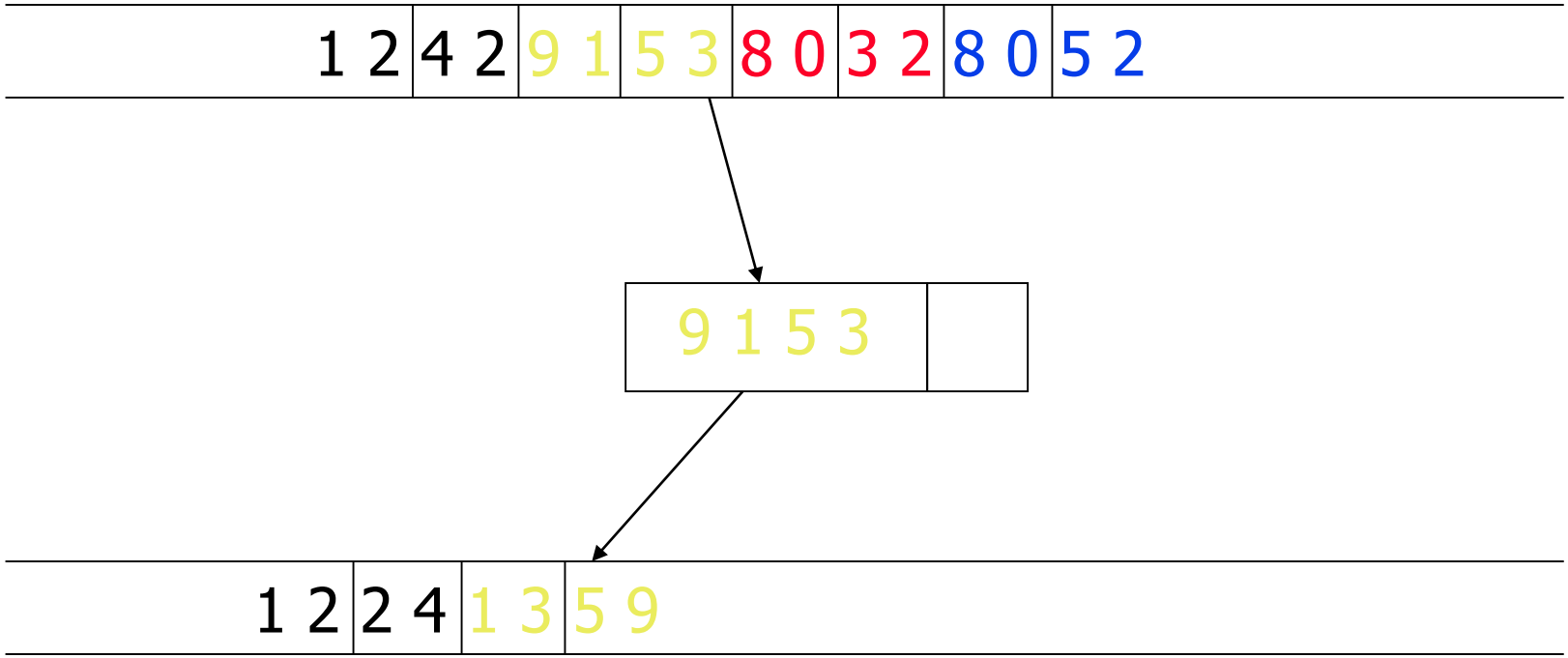
- Iteratively create large sorted groups and merge them
- Phase 1: Create *N/(M –1)* sorted groups of size *(M – 1)* blocks each

B = 2 objects; M = 3 blocks; N = 8 blocks N/(M –1) = 4



Merge Sort: Phase 1

B = 2; M = 3; N = 8 N/(M –1) = 4



End of Phase 1

B = 2; M = 3; N = 8 *N/(M –1) = 4*

1 2	4 2	9 1	5 3	8 0	3 2	8 0	5 2

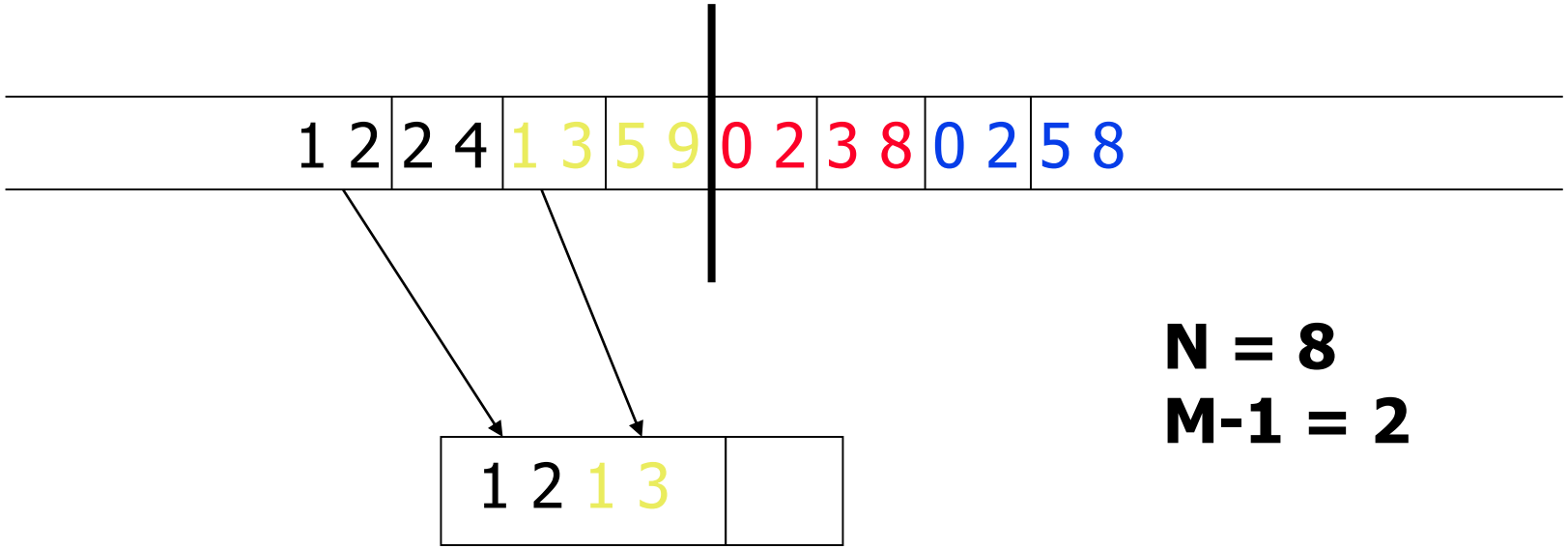
One Scan

1 2	2 4	1 3	5 9	0 2	3 8	0 2	5 8

N/(M-1) = 4 sorted groups of size (M-1) = 2 blocks each

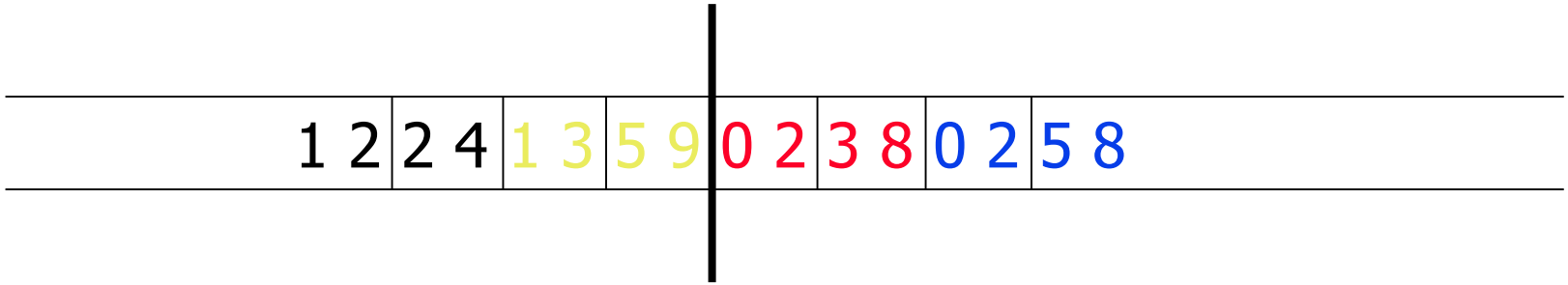
Second Phase: Merging Runs First Pass

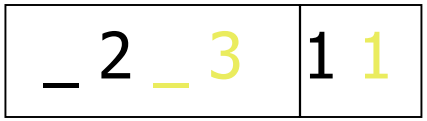
- Merge *(M-1)* sorted groups into one sorted group
- Iterate until all groups merged



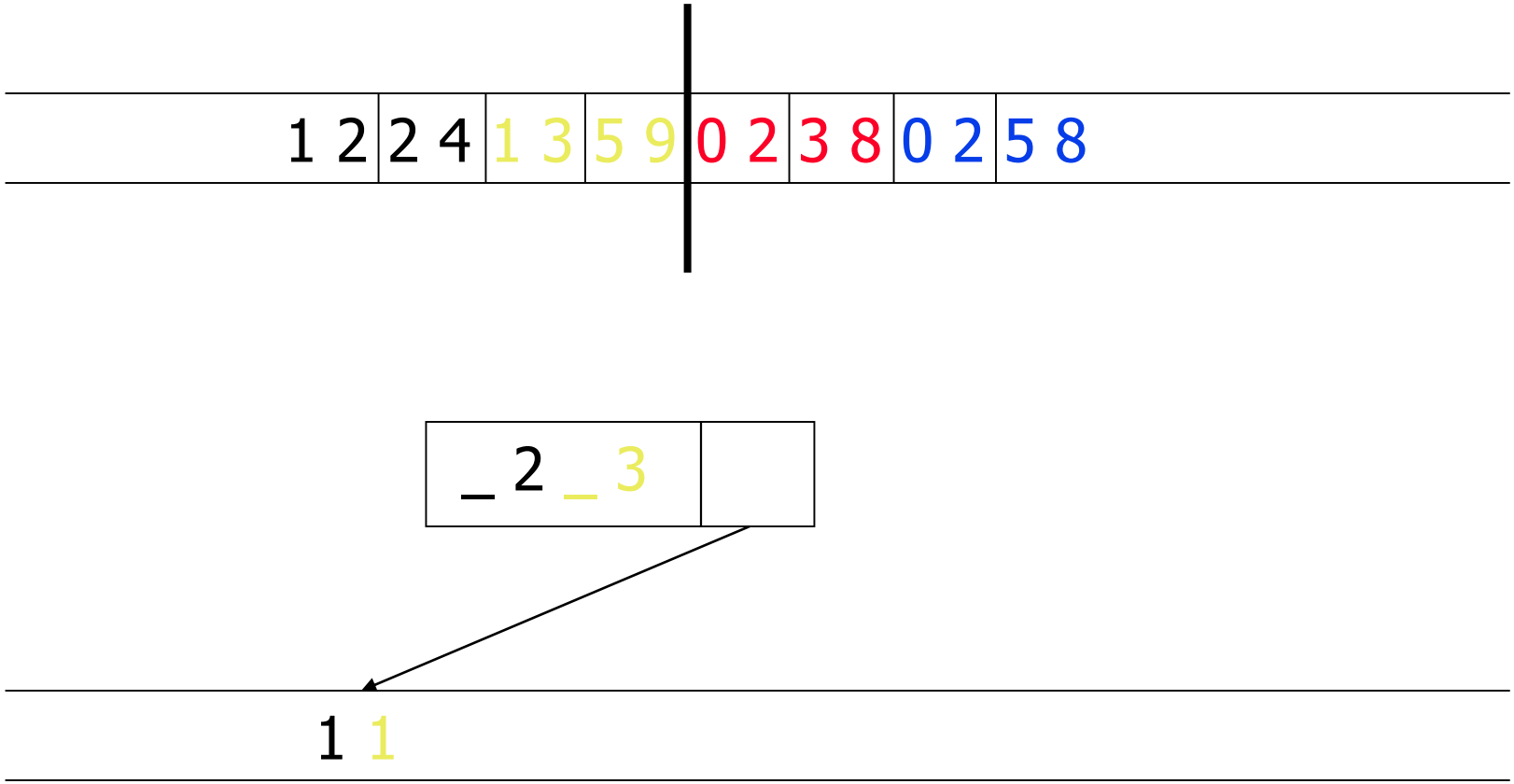
N = 8
M-1 = 2

Second Phase: Merging Runs First Pass

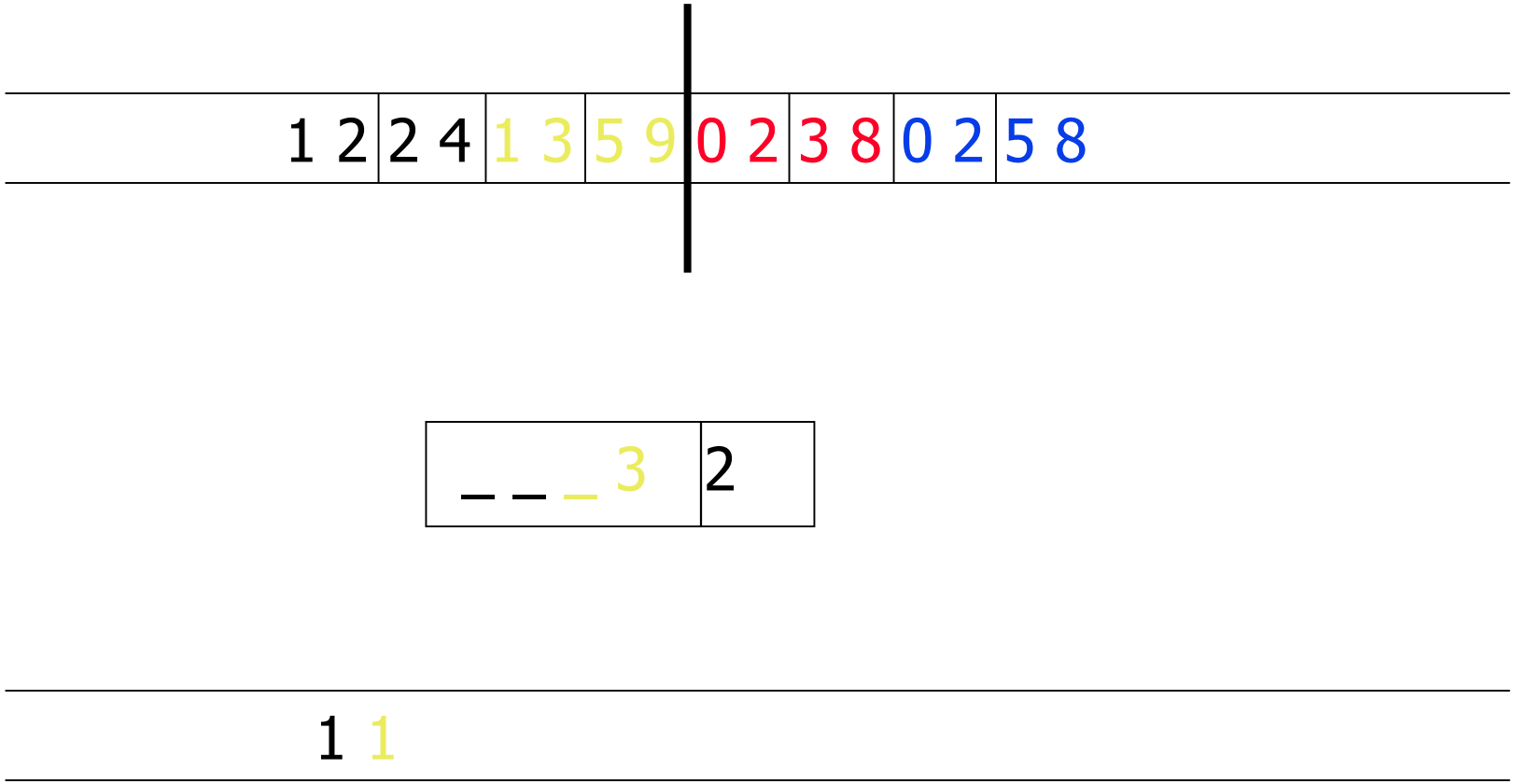




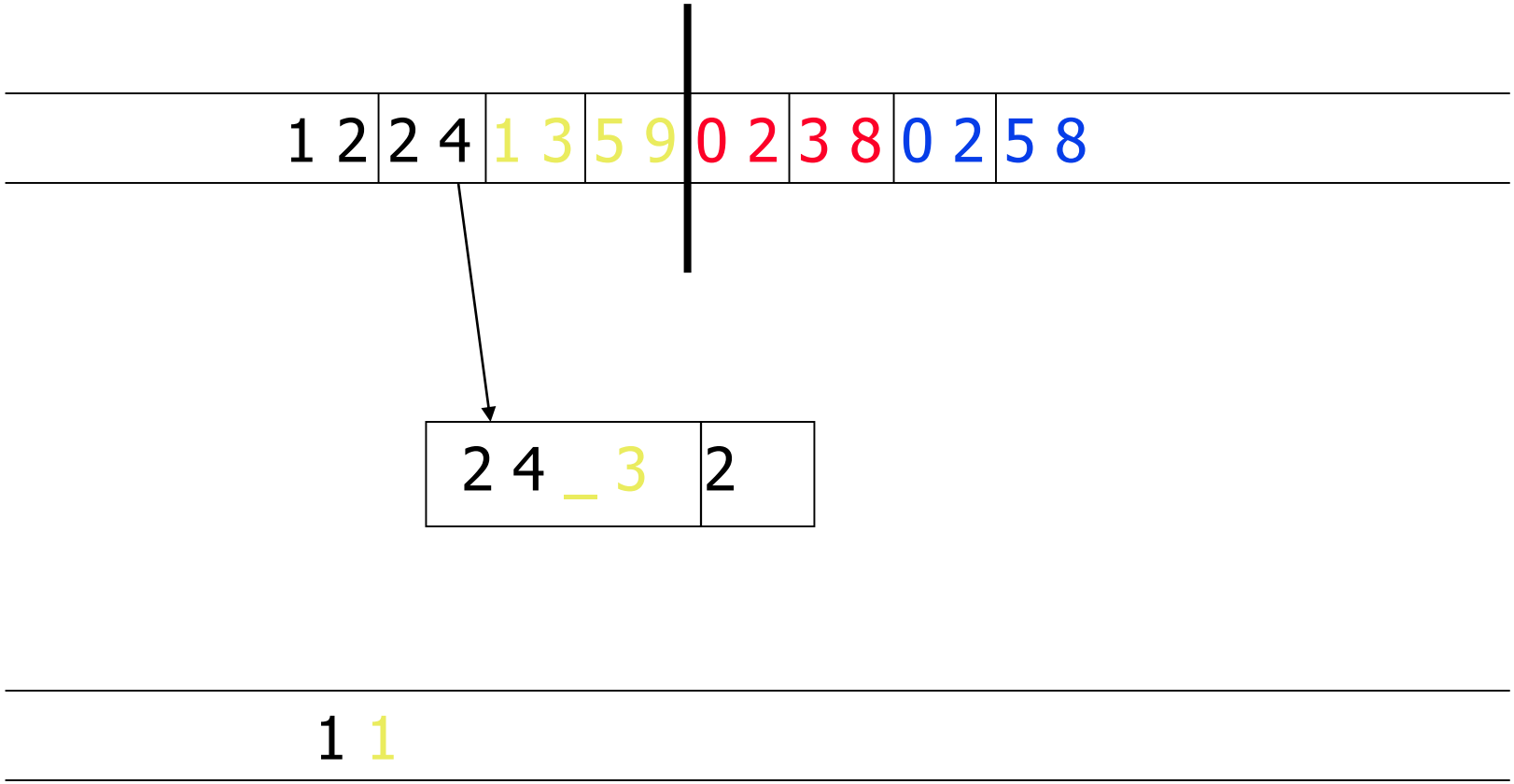
Second Phase: Merging Runs First Pass



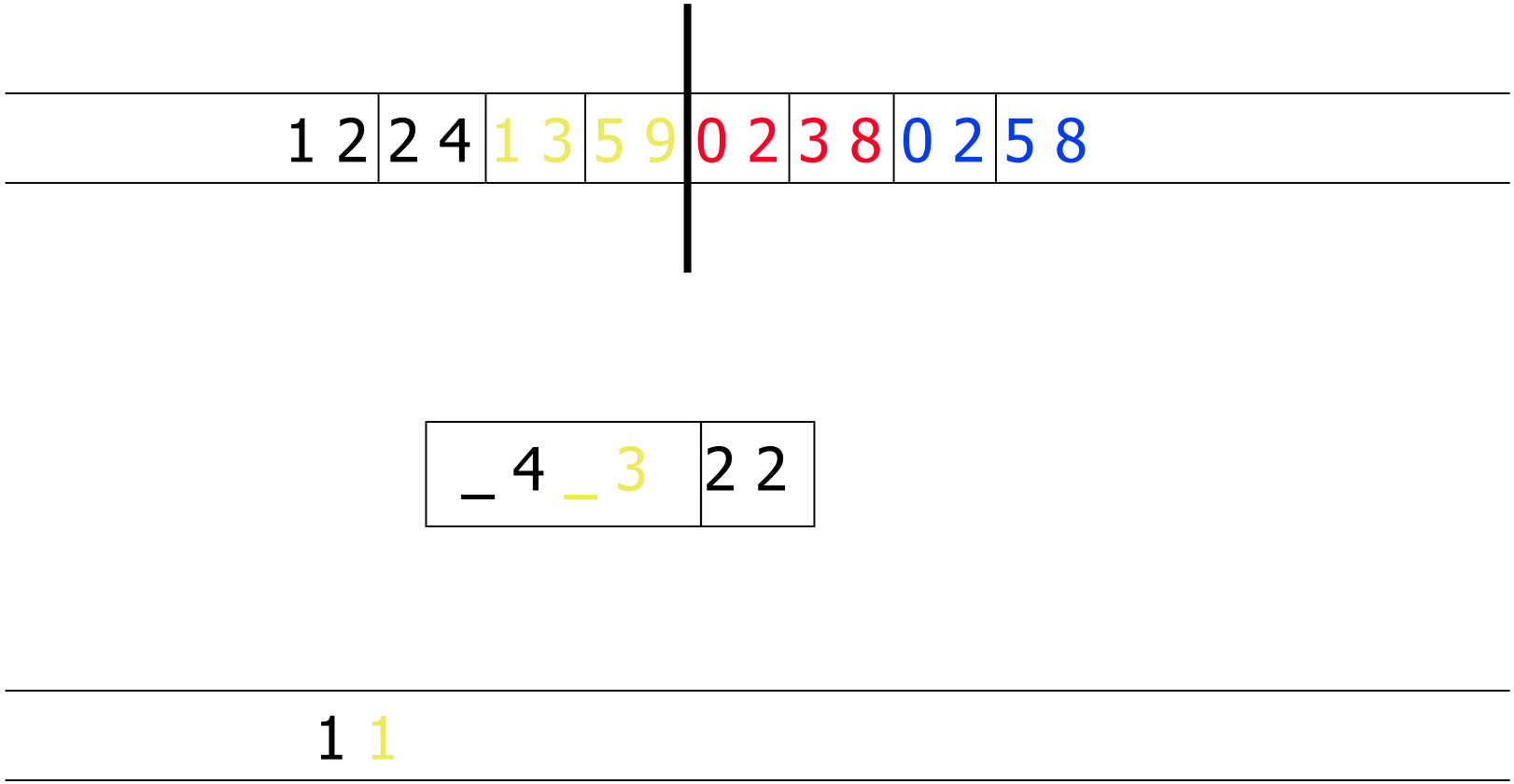
Second Phase: Merging Runs First Pass



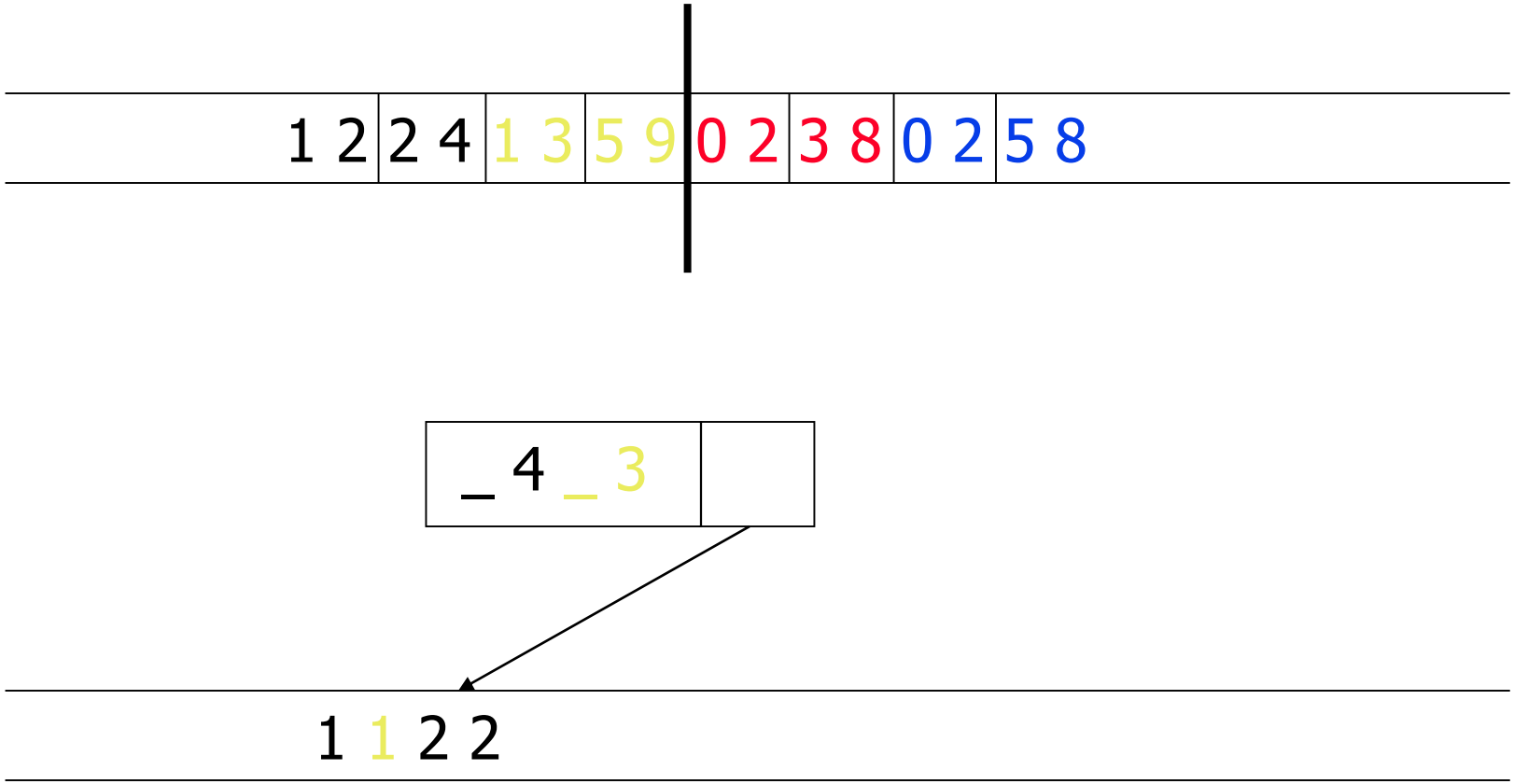
Second Phase: Merging Runs First Pass



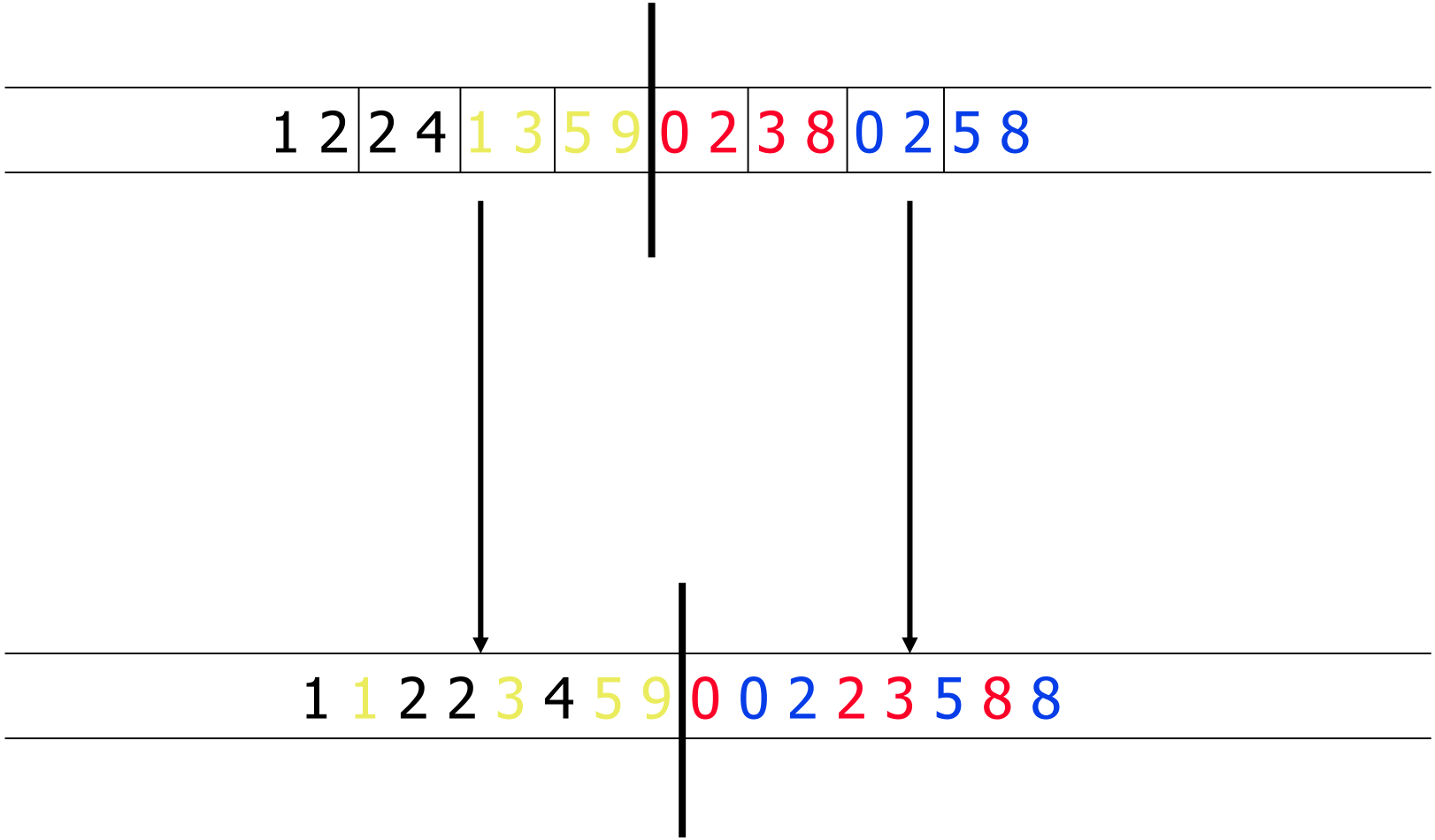
Second Phase: Merging Runs First Pass



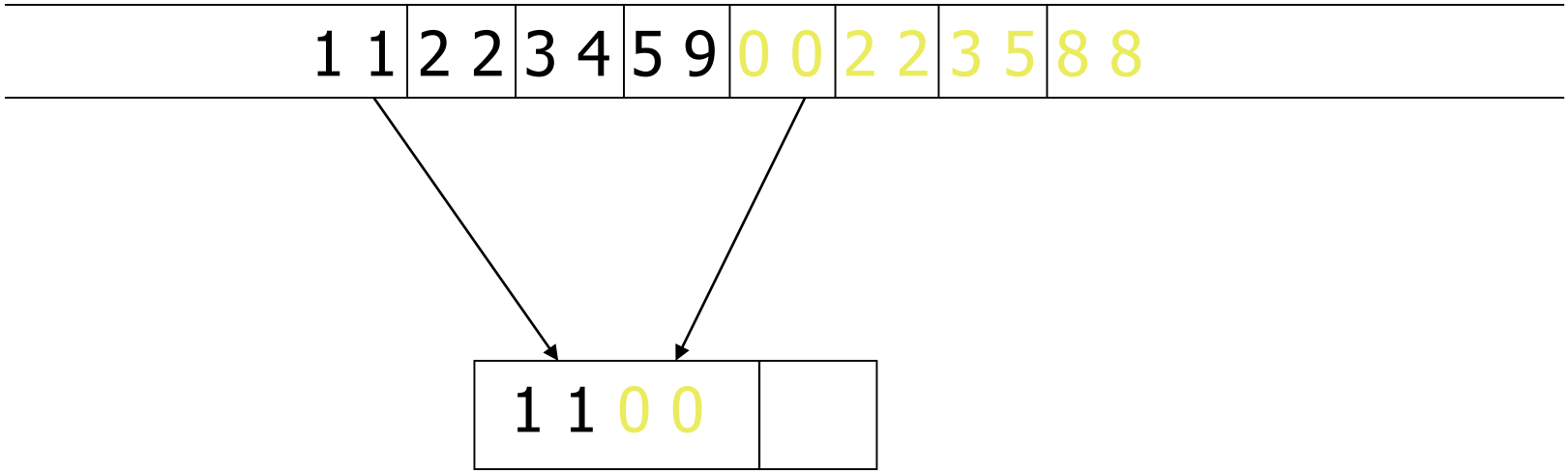
Second Phase: Merging Runs First Pass



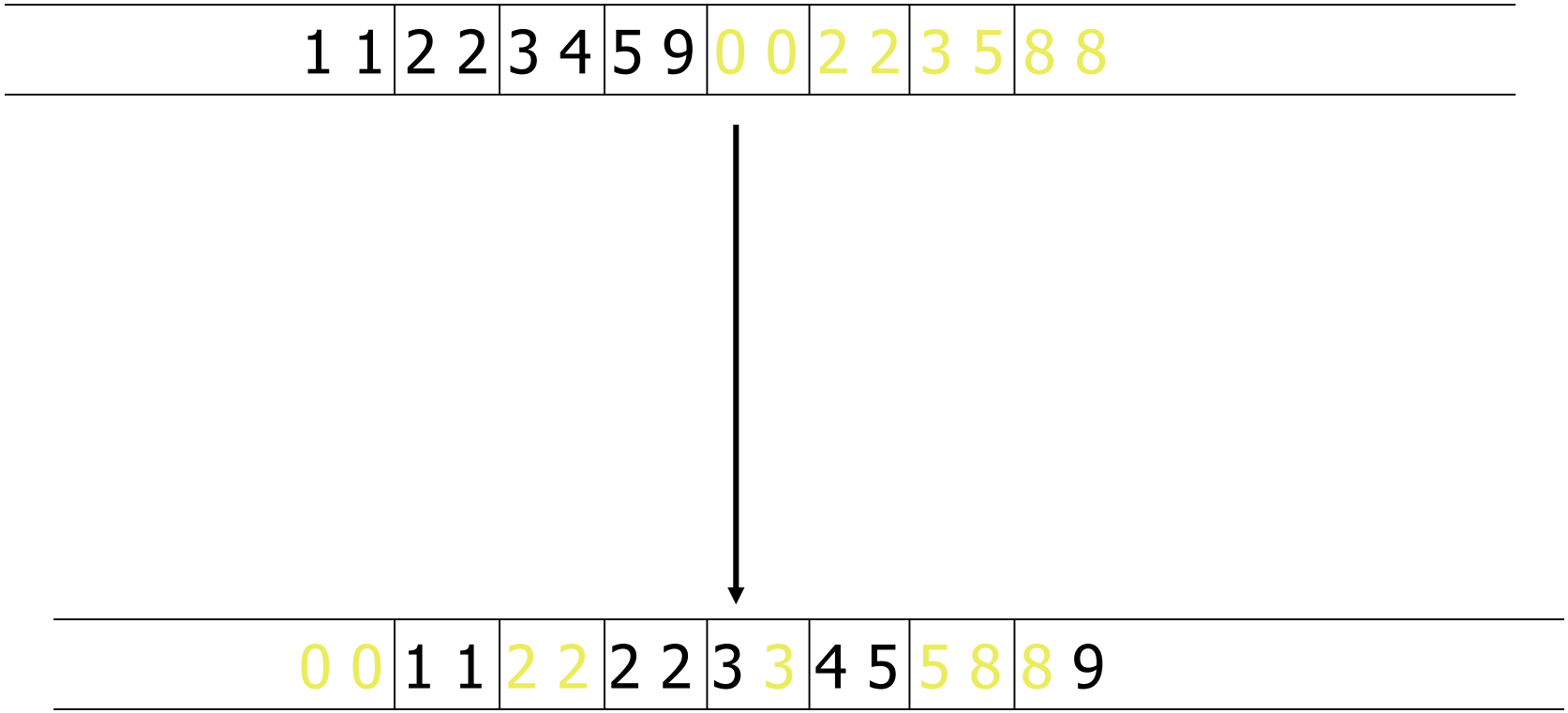
Second Phase: Merging Runs End of First Pass



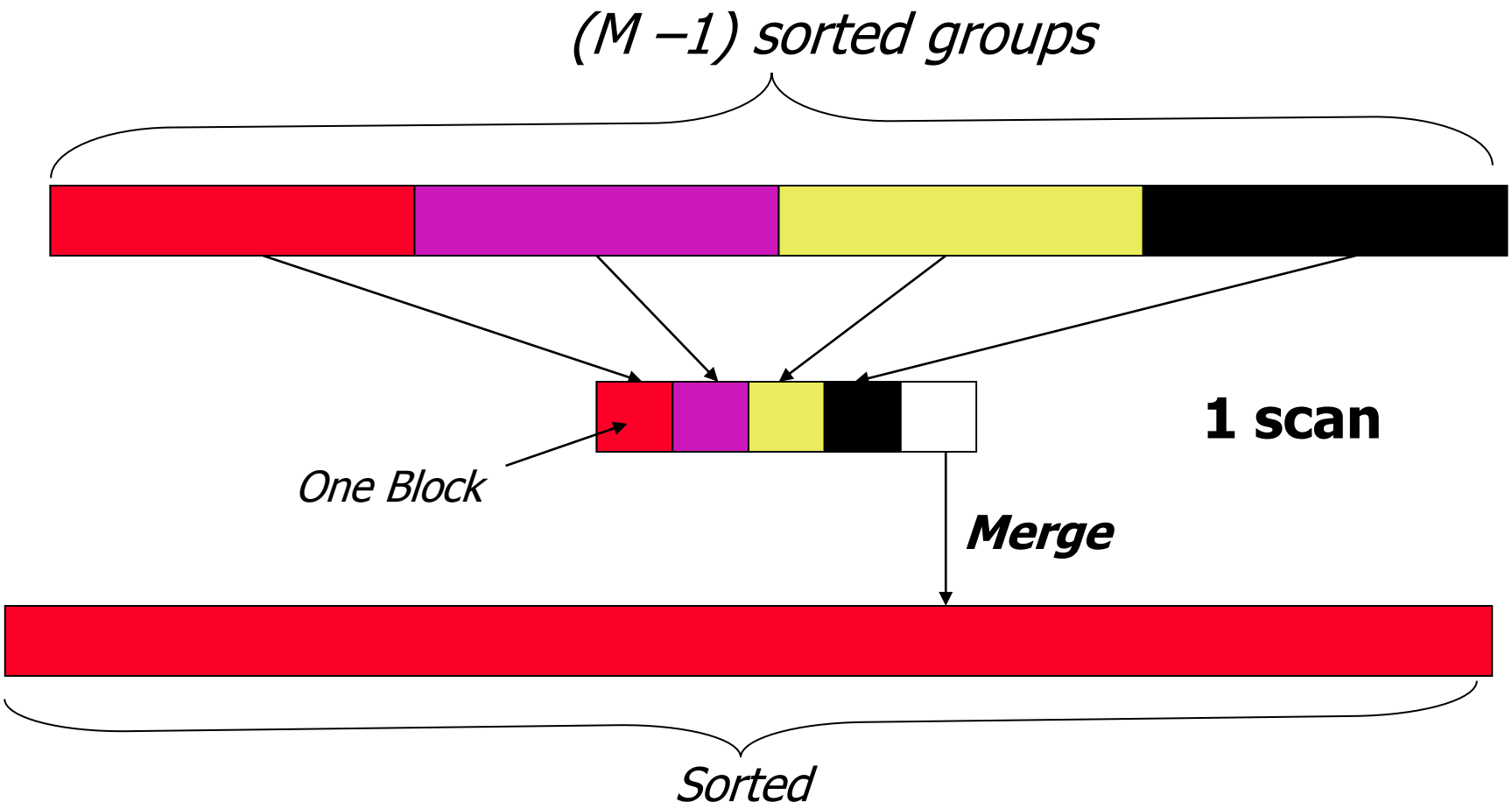
Second Phase: Merging Runs Second Pass



Second Phase: Merging Runs End of Second Pass



General Merging Step



Sort-Merge: Second Example

- Assume that only three tuples fit in a block and that main memory holds 4 blocks. Suppose that we sort this relation using sort-merge (instead of doing it in main memory). Show the runs created on each pass of the sort-merge algorithm, when applied to sort the following tuples on the first attribute:

i.e. Block = 3 Objects (tuples)

M = 4 blocks

N = 4 blocks

t_1	kangaroo	17
t_2	wallaby	21
t_3	emu	1
t_4	wombat	13
t_5	platypus	3
t_6	lion	8
t_7	warthog	4
t_8	zebra	11
t_9	meerkat	6
t_{10}	hyena	9
t_{11}	hornbill	2
t_{12}	baboon	12

Sort-Merge Second Example: Initial Sorted Runs

t_1	kangaroo	17
t_2	wallaby	21
t_3	emu	1
t_4	wombat	13
t_5	platypus	3
t_6	lion	8
t_7	warthog	4
t_8	zebra	11
t_9	meerkat	6
t_{10}	hyena	9
t_{11}	hornbill	2
t_{12}	baboon	12

- Let r_{ij} be the jth run computed on the ith pass

r_{11}	t_3	emu	1
	t_1	kangaroo	17
	t_2	wallaby	21
r_{12}	t_6	lion	8
	t_5	platypus	3
	t_4	wombat	13
r_{13}	t_9	meerkat	6
	t_7	warthog	4
	t_8	zebra	11
r_{14}	t_{12}	baboon	12
	t_{11}	hornbill	2
	t_{10}	hyena	9

Sort-Merge Second Example: Merging

r_{11}	t_3	emu	1
	t_1	kangaroo	17
	t_2	wallaby	21
r_{12}	t_6	lion	8
	t_5	platypus	3
	t_4	wombat	13
r_{13}	t_9	meerkat	6
	t_7	warthog	4
	t_8	zebra	11

output buffer

emu	1
kangaroo	17
lion	8

- First merge pass
- Memory holds four blocks
- Load three blocks of data and reserve one as output buffer
- Fill output buffer by selecting next tuple from across all input blocks
- Next: write output buffer to disk

Sort-Merge Second Example: Merging

r_{11}	t_3	emu	1
	t_1	kangaroo	17
	t_2	wallaby	21
r_{12}	t_6	lion	8
	t_5	platypus	3
	t_4	wombat	13
r_{13}	t_9	meerkat	6
	t_7	warthog	4
	t_8	zebra	11

output buffer

meerkat	6
platypus	3
wallaby	21

emu	1
kangaroo	17
lion	8

Sort-Merge Second Example: Merging

r_{11}

t_3 emu	1
t_1 kangaroo	17
t_2 wallaby	21

r_{12}

t_6 lion	8
t_5 platypus	3
t_4 wombat	13

r_{13}

t_9 meerkat	6
t_7 warthog	4
t_8 zebra	11

output buffer

warthog	4
wombat	13
zebra	11

emu	1
kangaroo	17
lion	8

meerkat	6
platypus	3
wallaby	21

Sort-Merge Second Example: Merging

- End of first merge pass
- Disk holds two runs:
  - First is the result of merging three runs
  - Second is the remaining, original 4th run
- Next: second merge pass

r_{21}

emu	1
kangaroo	17
lion	8
meerkat	6
platypus	3
wallaby	21
warthog	4
wombat	13
zebra	11

r_{22}

t_{12}	baboon	12
t_{11}	hornbill	2
t_{10}	hyena	9

Sort-Merge Second Example: Merging

r_{21}

t_3	emu	1
t_1	kangaroo	17
t_2	lion	8

r_{22}

t_6	baboon	12
t_5	hornbill	2
t_4	hyena	9

output buffer

r_{21}

emu	1
kangaroo	17
lion	8
meerkat	6
platypus	3
wallaby	21
warthog	4
wombat	13
zebra	11

r_{22}

t_{12}	baboon	12
t_{11}	hornbill	2
t_{10}	hyena	9

Sort-Merge Second Example: Merging

r_{21}

t_3	emu	1
t_1	kangaroo	17
t_2	lion	8

r_{22}

t_6	baboon	12
t_5	hornbill	2
t_4	hyena	9

output buffer

baboon	12
emu	1
hornbill	2

Sort-Merge Second Example: Merging

r_{21}

t_3	emu 1
t_1	kangaroo 17
t_2	lion 8

r_{22}

t_6	baboon 12
t_5	hornbill 2
t_4	hyena 9

baboon	12
emu	1
hornbill	2

- First output buffer written to disk
- Finished processing loaded runs
- Discard first block of run r_{21}
- Load next block of run r_{21}

output buffer

hyena	9
kangaroo	17
lion	8

Sort-Merge Second Example: Merging

r_{21}

t_3	meerkat	6
t_1	platypus	3
t_2	wallaby	21

r_{22}

t_6	baboon	12
t_5	hornbill	2
t_4	hyena	9

baboon	12
emu	1
hornbill	2
hyena	9
kangaroo	17
lion	8

output buffer

Sort-Merge Second Example: Merging

r_{21}

t_3	meerkat	6
t_1	platypus	3
t_2	wallaby	21

r_{22}

t_6	baboon	12
t_5	hornbill	2
t_4	hyena	9

baboon	12
emu	1
hornbill	2
hyena	9
kangaroo	17
lion	8

output buffer

meerkat	6
platypus	3
wallaby	21

Sort-Merge Second Example: Merging

r_{21}

t_3	warthog	4
t_1	wombat	13
t_2	zebra	11

r_{22}

t_6	baboon	12
t_5	hornbill	2
t_4	hyena	9

baboon	12
emu	1
hornbill	2
hyena	9
kangaroo	17
lion	8
meerkat	6
platypus	3
wallaby	21

output buffer

Sort-Merge Second Example: Merging

r_{21}

t_3	warthog	4
t_1	wombat	13
t_2	zebra	11

r_{22}

t_6	baboon	12
t_5	hornbill	2
t_4	hyena	9

output buffer

warthog	4
wombat	13
zebra	11

r_{31}

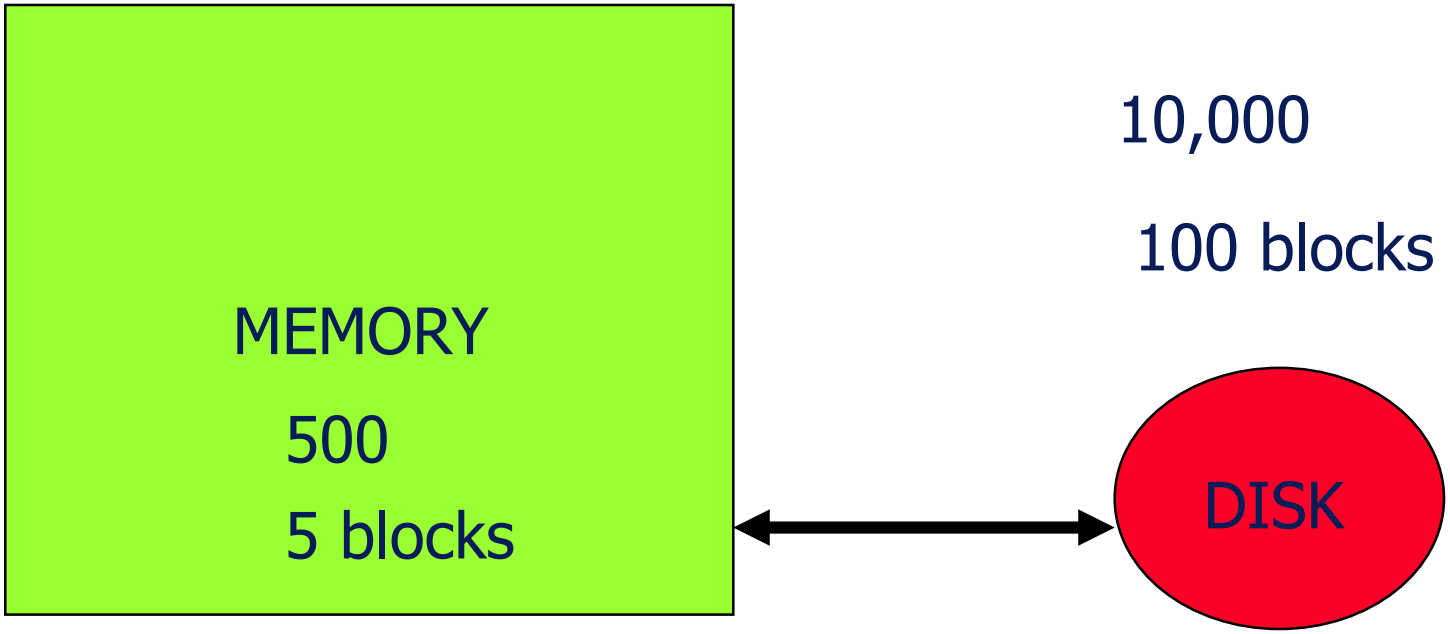
baboon	12
emu	1
hornbill	2
hyena	9
kangaroo	17
lion	8
meerkat	6
platypus	3
wallaby	21
warthog	4
wombat	13
zebra	11

- End!

External Merge Sort: Third Example

- Sort N=10000 records
- Enough memory for 500 records
- Block size is b=100 records, i.e. B=N/b=100 and M=5 (buffers)
- t_{IS} = time to internally sort 1 memory load
- t_{IM} = time to internally merge 1 block load

Run Generation



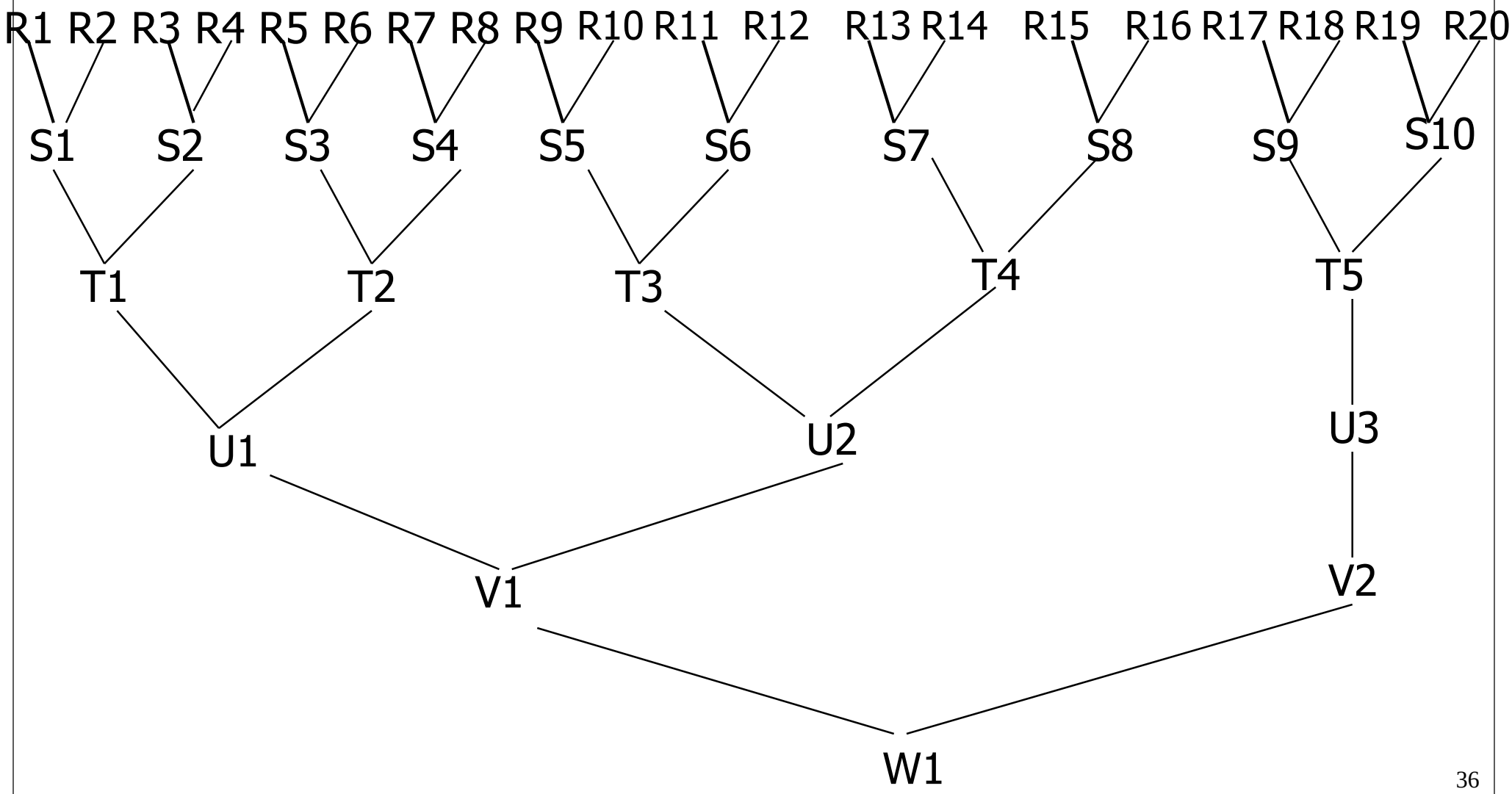
- Input 5 blocks
- Sort
- Output as a run
- Do 20 times

- M
- t_{IS}
- M
- $(B/M)*(2M+ t_{IS})=200+20\ t_{IS}$

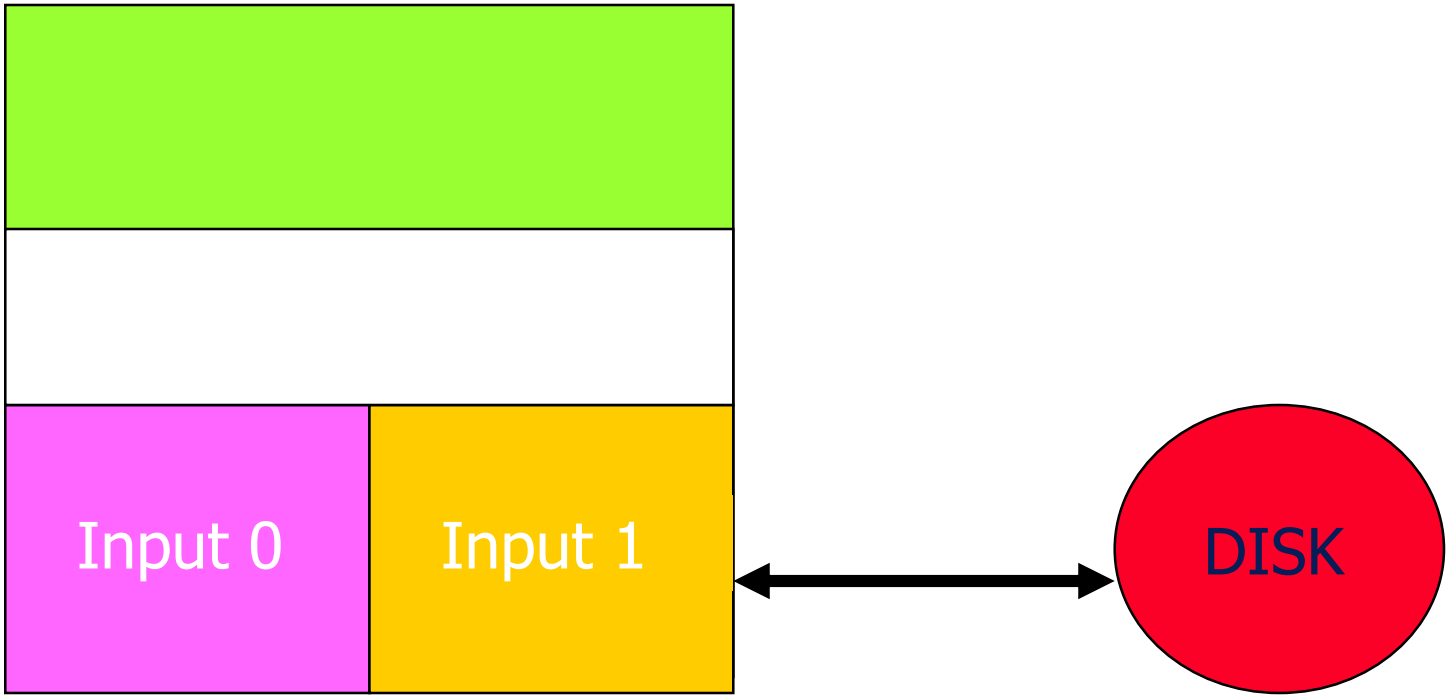
Run Merging

- Merge Pass
  - Pairwise merge the 20 runs into 10
  - In a merge pass all runs (except possibly one) are pairwise merged
- Perform 4 more merge passes, reducing the number of runs to 1

Merge 20 Runs



Merge R1 and R2



- Fill I0 (Input 0) from R1 and I1 from R2
- Merge from I0 and I1 to output buffer
- Write whenever output buffer full
- Read whenever input buffer empty

Time To Merge R1 and R2

- Each is M=5 blocks long
- Input time = 2*M = 10
- Write/output time = 2*M = 10
- Merge time = $2*M*t_{IM}$
- Total time = $20 + 2*M*t_{IM} = 20 + 10t_{IM}$

Time For Pass 1 (R ⟶ S)

- Time to merge one pair of runs = $20 + 10t_{IM}$
- Time to merge all 10 pairs of runs = $200 + 100t_{IM}$

Time To Merge S1 and S2

- Each is 2*M=10 blocks long
- Input time = 2*(2*M) = 20
- Write/output time = 2*(2*M) = 20
- Merge time = $2*(2*M)*t_{IM}$
- Total time = $40 + 20t_{IM}$

Time For Pass 2 (S⟶T)

- Time to merge one pair of runs = $40 + 20t_{IM}$
- Time to merge all 5 pairs of runs = $200 + 100t_{IM}$

Time For One Merge Pass

- Time to input all blocks = B = 100
- Time to output all blocks = B = 100
- Time to merge all blocks = B* t_{IM} = 100t_{IM}
- Total time for a merge pass = 200 + 100t_{IM}

Total Run-Merging Time

- (time for one merge pass) * (number of passes)

  = (time for one merge pass)
  $* \lceil \log_2 \text{number\_of\_initial\_runs} \rceil$

  = $(200 + 100t_{IM}) * \lceil \log_2 20 \rceil$

  = $(200 + 100t_{IM}) * 5$

- Neglecting t_{IM}, we get 200 * 5 = 1000 blocks I/O

Total time for External Sort

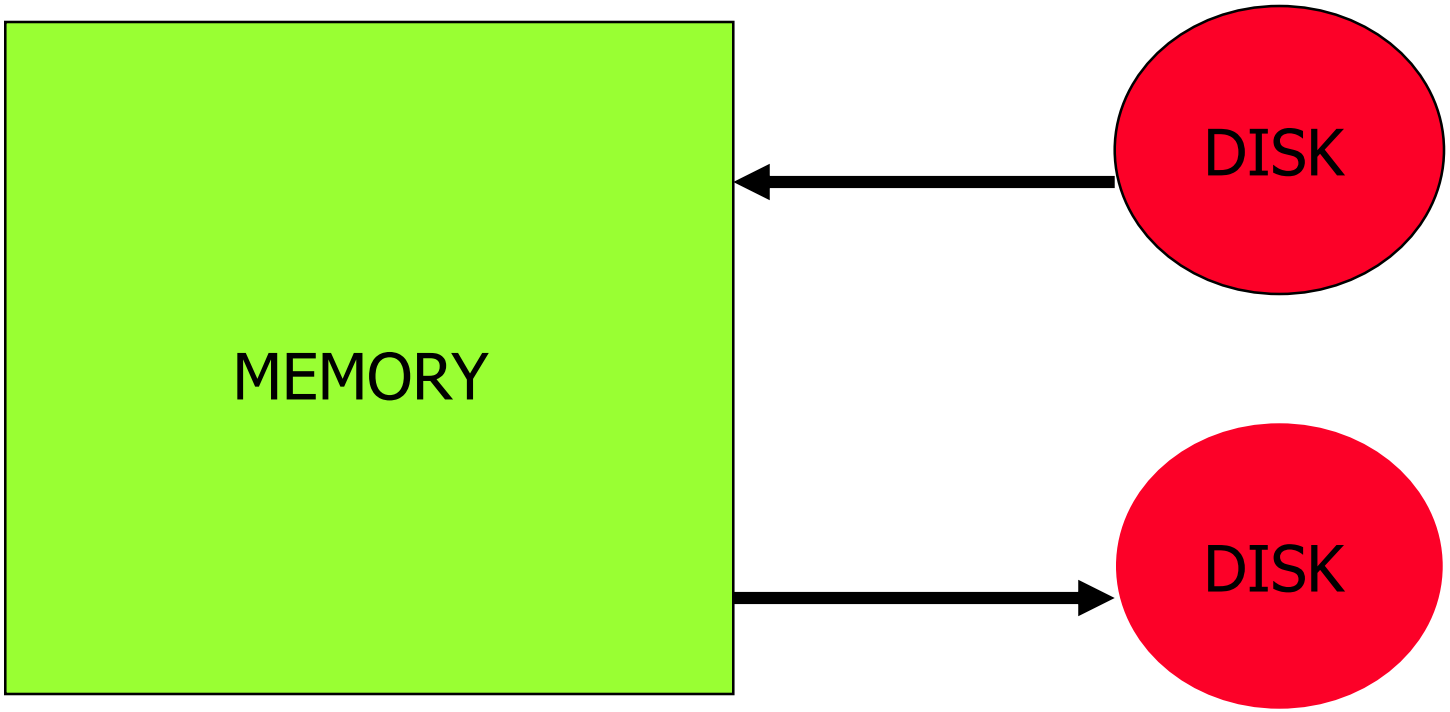
- Run_generation_time + run_merging_time = $(200+20t_{IS}) + (200 + 100t_{IM}) * 5 = 1200 + 20t_{IS} + 500t_{IM}$
- Neglecting t_{IS}, t_{IM} we get 1200 blocks I/O

Factors In Overall Run Time

- Run generation $200 + 20t_{IS}$
  - Internal sort time
  - Input and output time
- Run merging $(200 + 100t_{IM}) * \lceil \log_2 20 \rceil$
  - Internal merge time
  - Input and output time
  - Number of initial runs
  - Merge order (number of merge passes is determined by number of runs and merge order)

Improve Run Generation

- Overlap input,output, and internal sorting

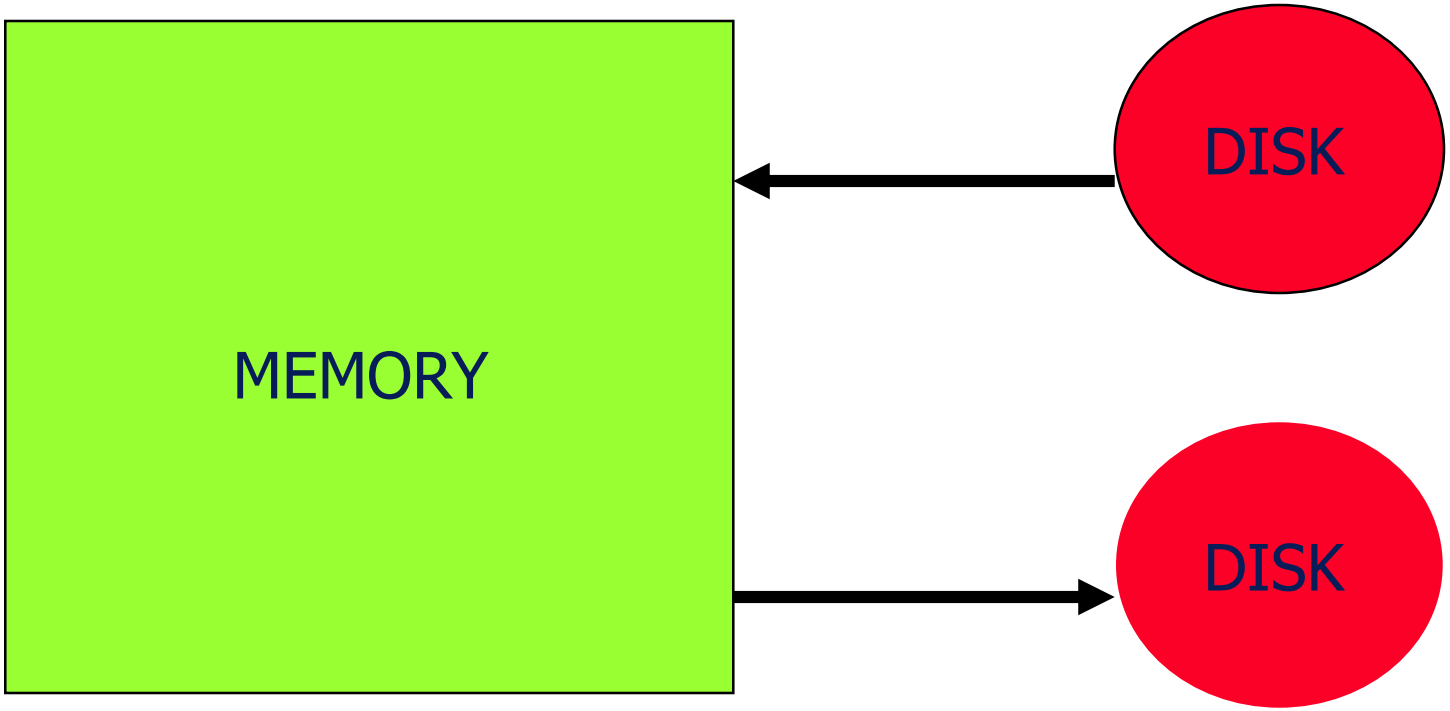


Improve Run Generation

- Generate runs whose length (on average) exceeds memory size
- Equivalent to reducing number of runs generated

Improve Run Merging

- Overlap input,output, and internal merging

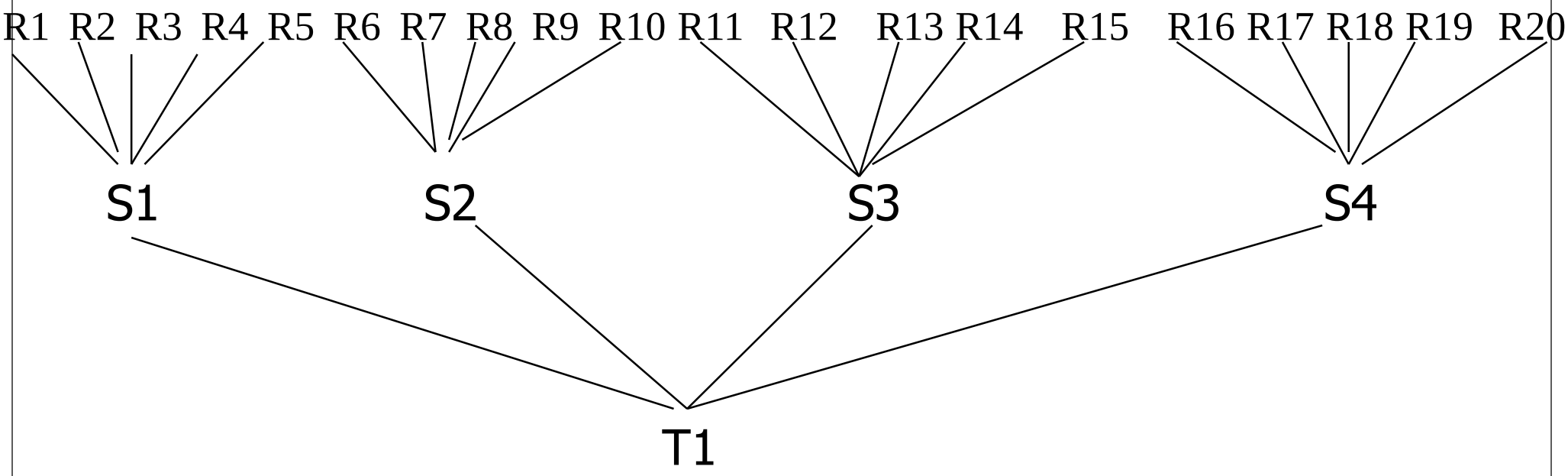


Improve Run Merging

- Reduce number of merge passes
  - Use higher-order merge
  - Number of passes = $\lceil \log_k \text{number\_of\_initial\_runs} \rceil$

    where k is the merge order

Merge 20 Runs Using 5-Way Merging

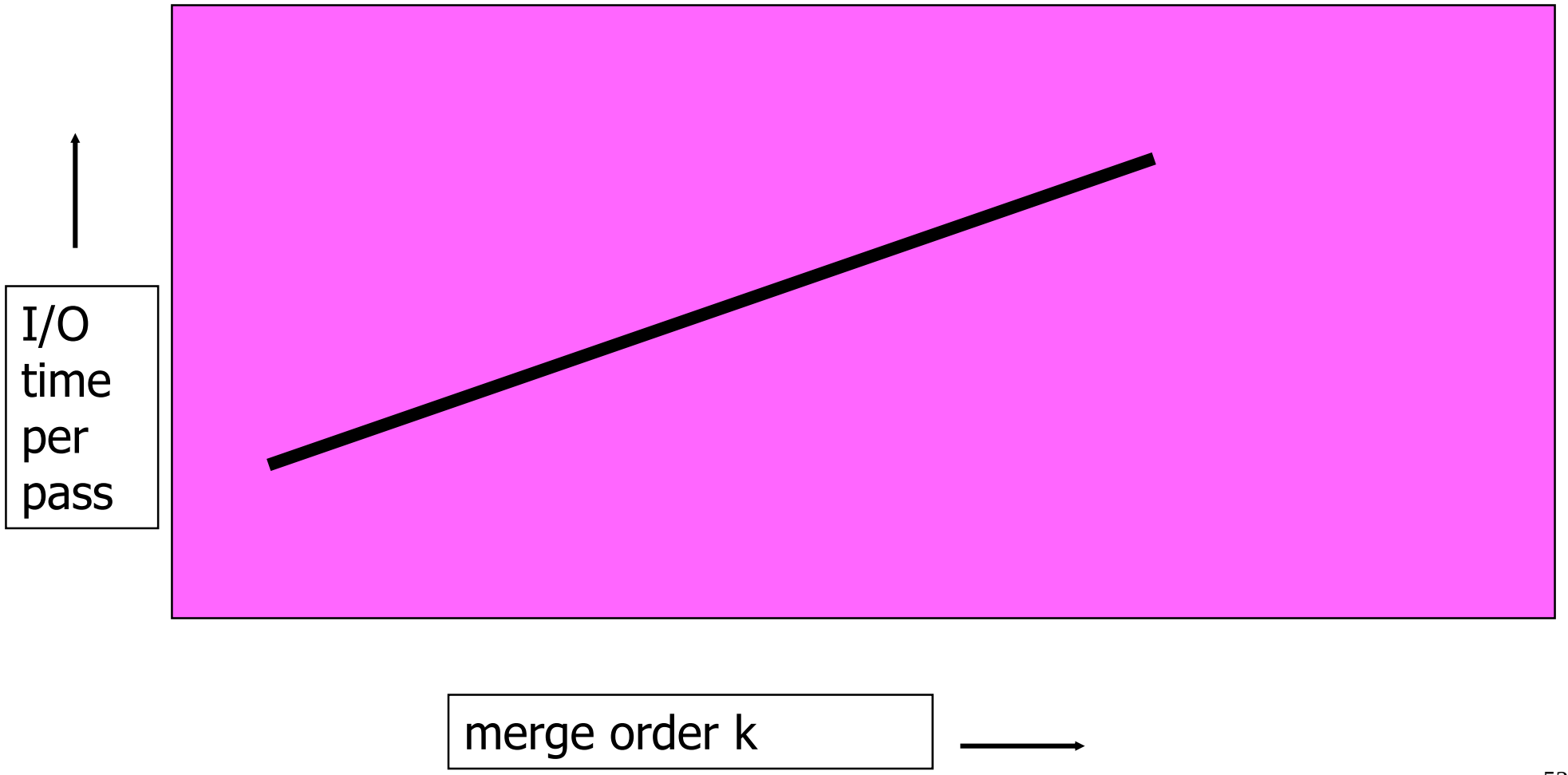


Number of passes = 2

I/O Time Per Merge Pass

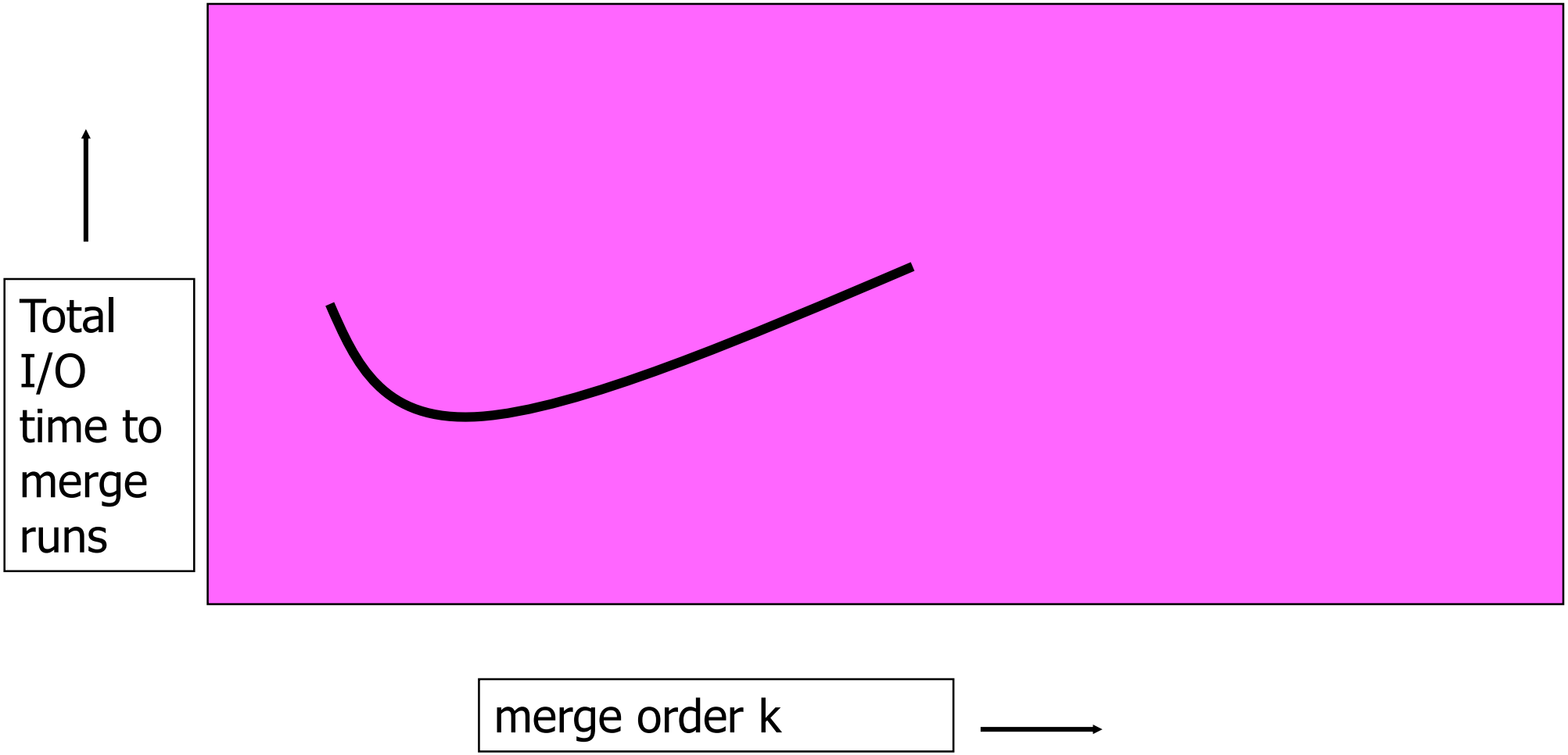
- Number of input buffers needed is linear in merge order k
- Since memory size is fixed, block size decreases as k increases
- So, number of blocks increases
- So, number of seek and latency delays per pass increases

I/O Time Per Merge Pass

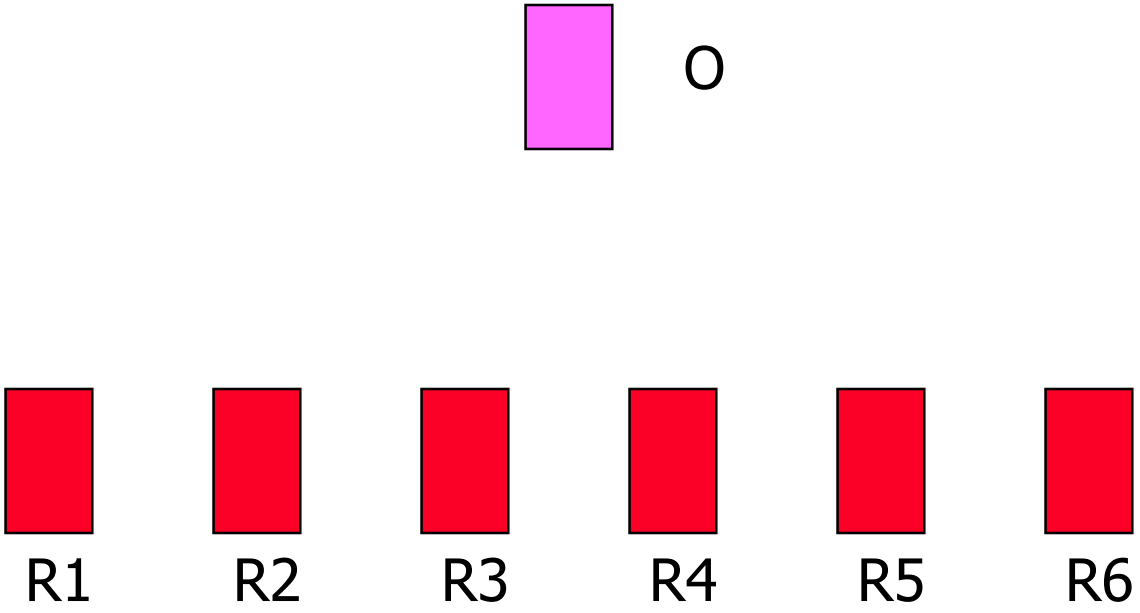


Total I/O Time To Merge Runs

- (I/O time for one merge pass) * $\lceil \log_k \text{number\_of\_initial\_runs} \rceil$

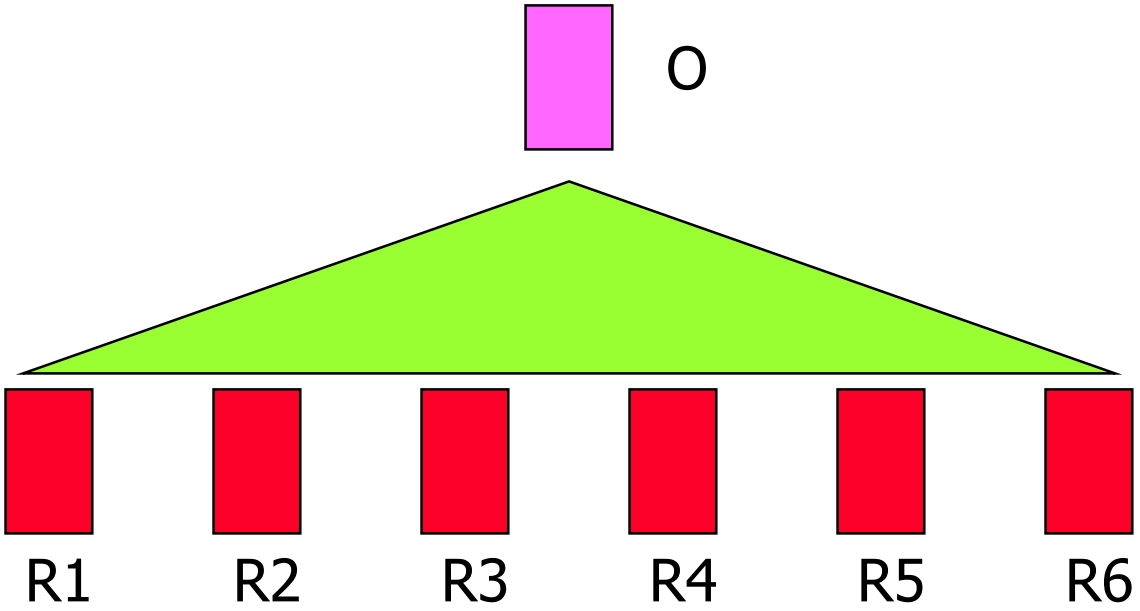


Internal Merge Time



- Naïve way k – 1 compares to determine next record to move to the output buffer
- Time to merge n records is c(k – 1)n, where c is a constant
- Merge time per pass is c(k – 1)n
- Total merge time is $c(k-1)n\log_k r = cn(k/\log_2 k)\log_2 r$

Merge Time Using A Tournament Tree



- Time to merge n records is $dn\log_2 k$, where d is a constant
- Merge time per pass is $dn\log_2 k$
- Total merge time is $(dn\log_2 k)\ \log_k r = dn\log_2 r$

Τέλος Ενότητας

Χρηματοδότηση

•Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.

•Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Κρήτης**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.

•Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Σημειώματα

Σημειώματα

Σημείωμα αδειοδότησης

.

Σημείωμα Αναφοράς