Ταξινόμηση και εφαρμογές της

Γιάννης Κωτίδης

Checkpoint

- Έχουμε μιλήσει για
  - Εκτίμηση μεγέθους του αποτελέσματος ενός τελεστή μέσω απλών στατιστικών και ιστογραμμάτων
- Ας μπούμε τώρα στα βαθιά!
  - Υπολογισμός κόστους εκτέλεσης διαφόρων φυσικών τελεστών
  - Ας αρχίσουμε με την ταξινόμηση (Sort)

Ποιο είναι το κόστος της ταξινόμησης μίας σχέσης R

- ?

Απλοποίηση που έχουμε κάνει

- **Κόστος = #I/O = αριθμός σελίδων που διαβάζω ή γράφω στο δίσκο**

...επίσης, δε προσμετρώ το κόστος του να δώσω το αποτέλεσμα στο χρήστη

Εύκολη περίπτωση: Οι εγγραφές χωράνε στη μνήμη του ΣΔΒΔ

- Φέρε όλες τις εγγραφές της R στη μνήμη
- Ταξινόμησε τες (πως?)
- **Cost = B(R) !!!!**
  - **Κόστος ανάγνωσης (I/O) από το δίσκο στη μνήμη**
  - **Κόστος γραμμικό και <u>βέλτιστο</u>**

Τι γίνεται αν η σχέση είναι μεγαλύτερη από τη διαθέσιμη μνήμη M;

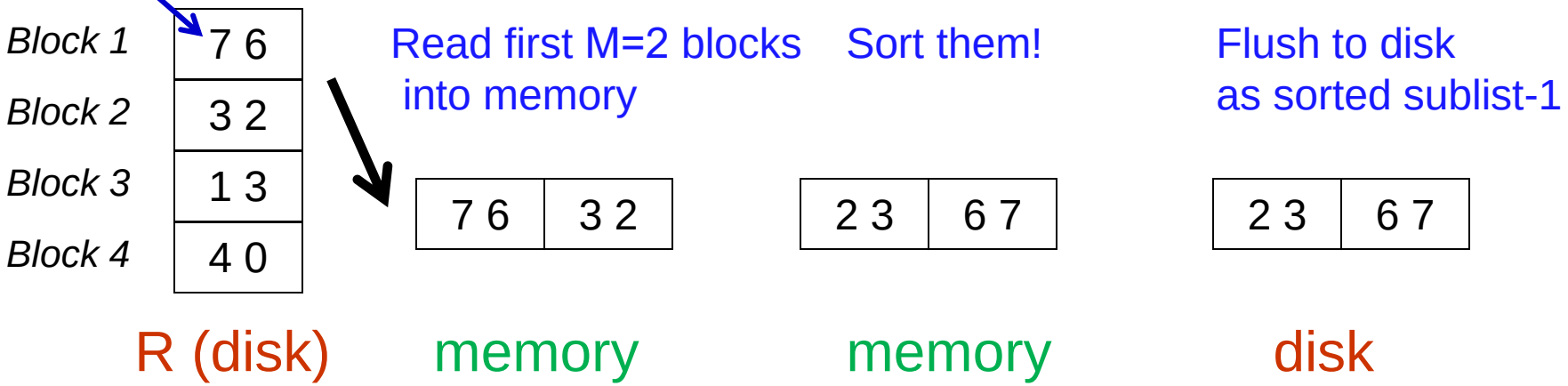
- Έστω M η διαθέσιμη μνήμη σε σελίδες
- Θα σχεδιάσουμε έναν αλγόριθμο ταξινόμησης 2 περασμάτων (two-pass)
  - Επεκτείνεται εύκολα σε περισσότερα περάσματα

Two-Phase Sort algorithm

- Phase 1:
  - Διάβασε τμηματικά τη σχέση R στη μνήμη σε κομμάτια μεγέθους M σελίδων το κάθε ένα
  - Ταξινόμησε κάθε κομμάτι στη μνήμη, γράψε το στο δίσκο ως λίστα (sublist) ταξινομημένων εγγραφών
- Phase 2:
  - Άνοιξε όλες τις λίστες και κάνε συγχώνευση (merge) παράγοντας έτσι το τελικό ταξινομημένο αποτέλεσμα

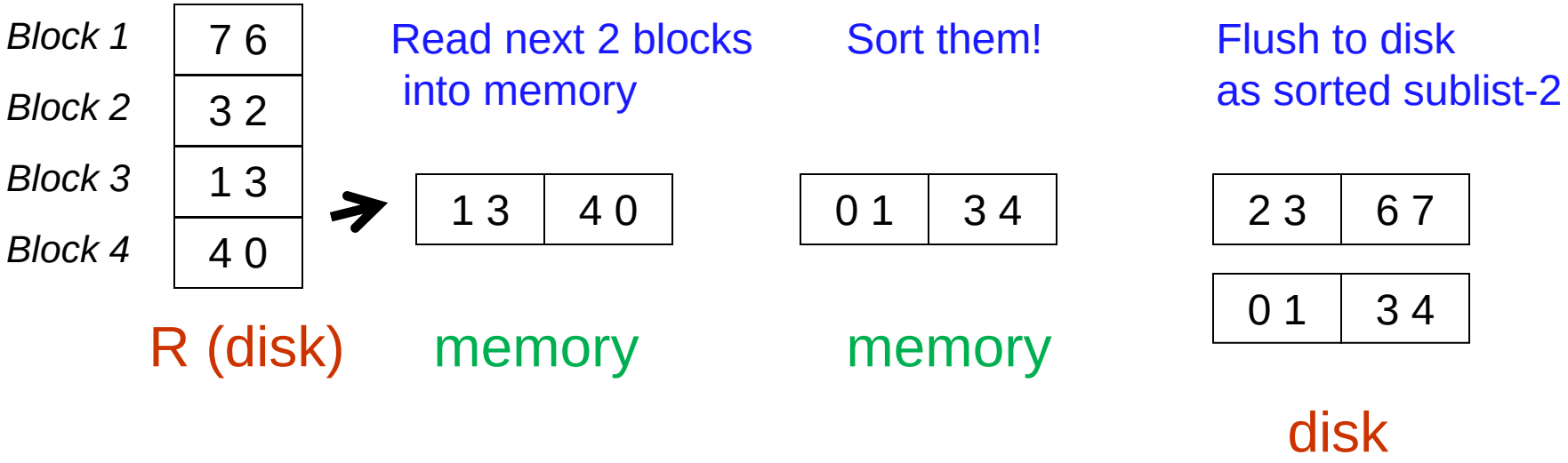
Two-phase Sort (Phase 1)

Τιμή γνωρίσματος στο οποίο ταξινομούμε τις εγγραφές



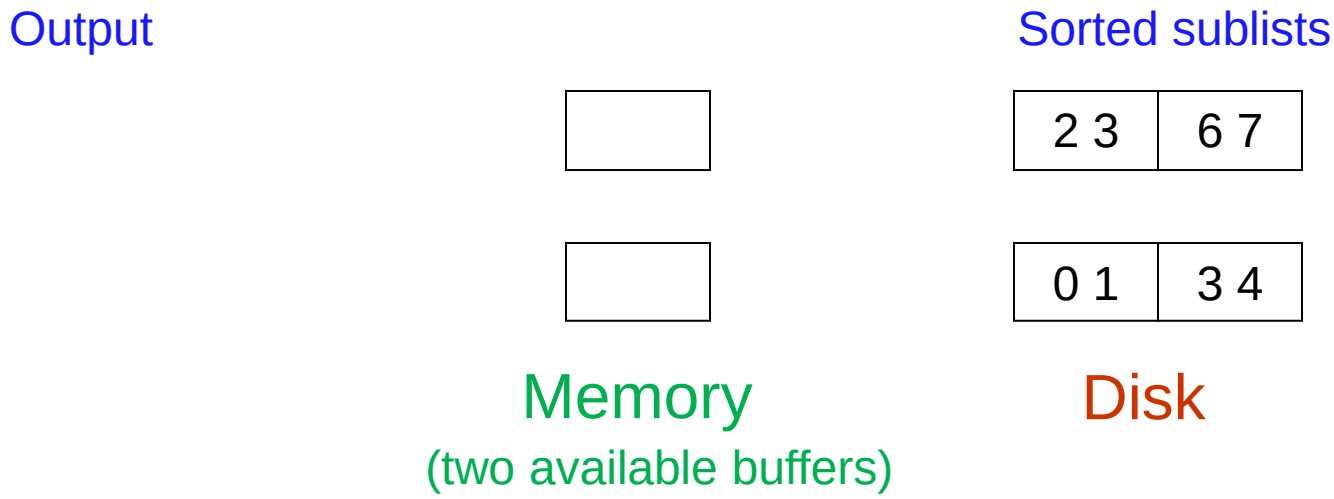
T(R)=8 records
B(R)=4 blocks (2 records/block)
Διαθέσιμη μνήμη για ταξινόμηση (sort buffer): M=2 blocks

Still Phase 1, read next 2 blocks

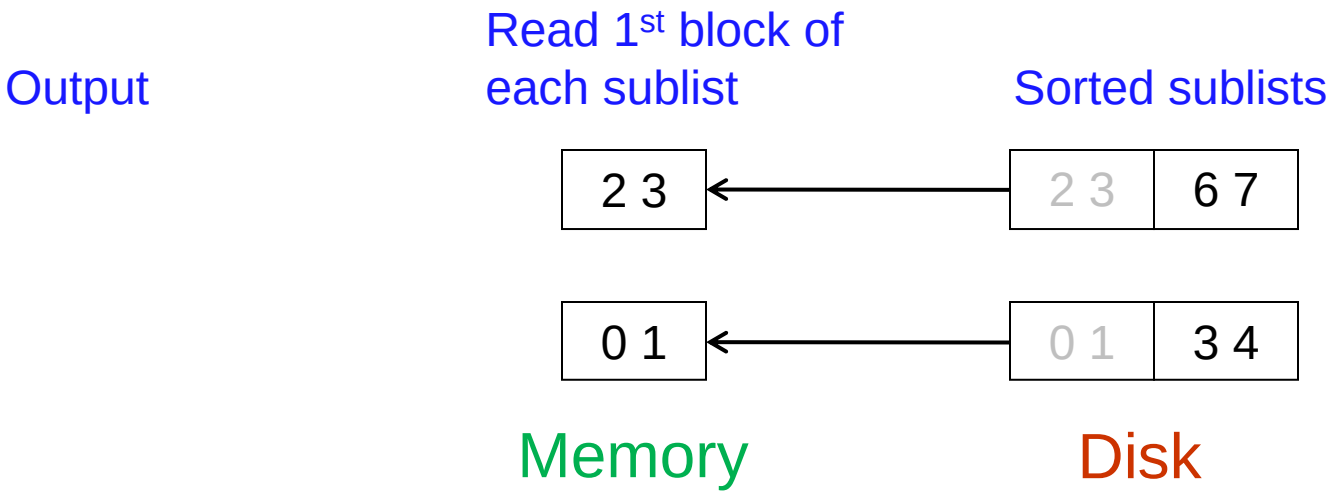


Πόσα I/Os έχω κάνει με το τέλος αυτής της φάσης?

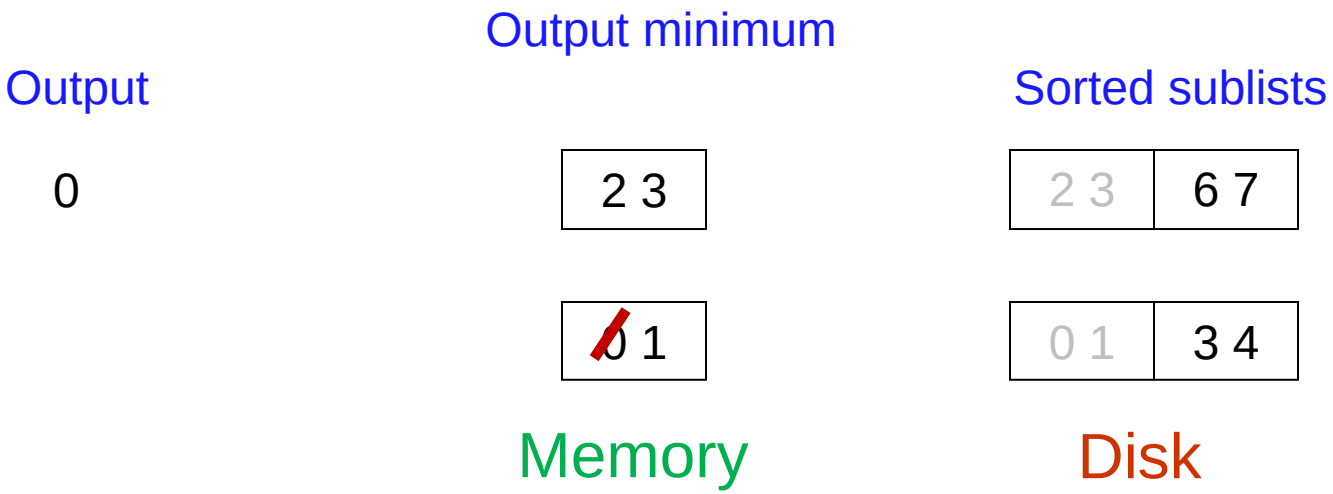
Two-phase Sort: Phase 2 (merge)



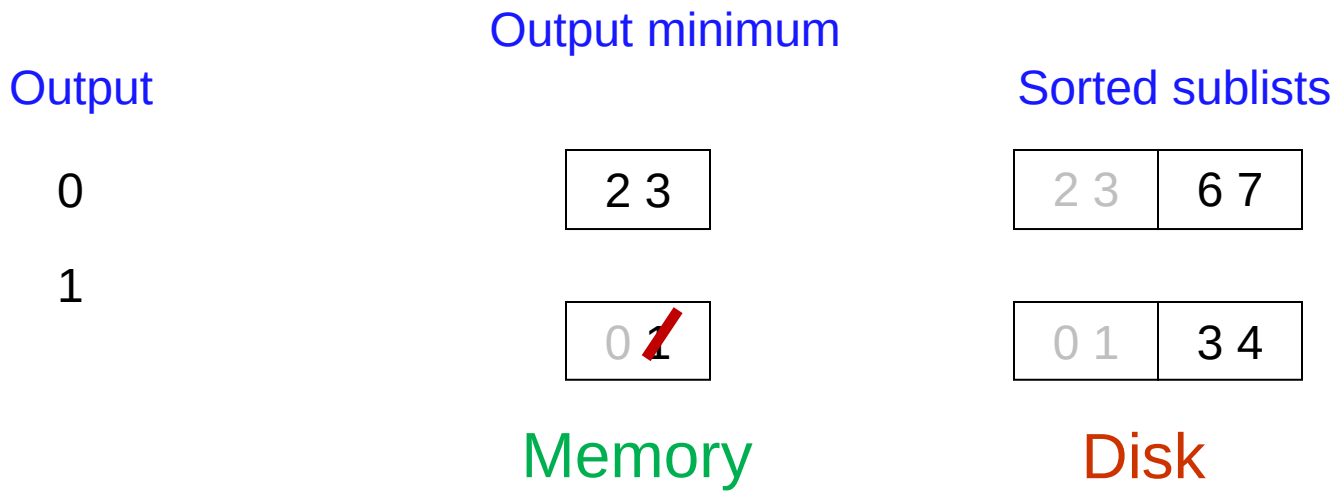
Two-phase Sort: Phase 2 (merge)



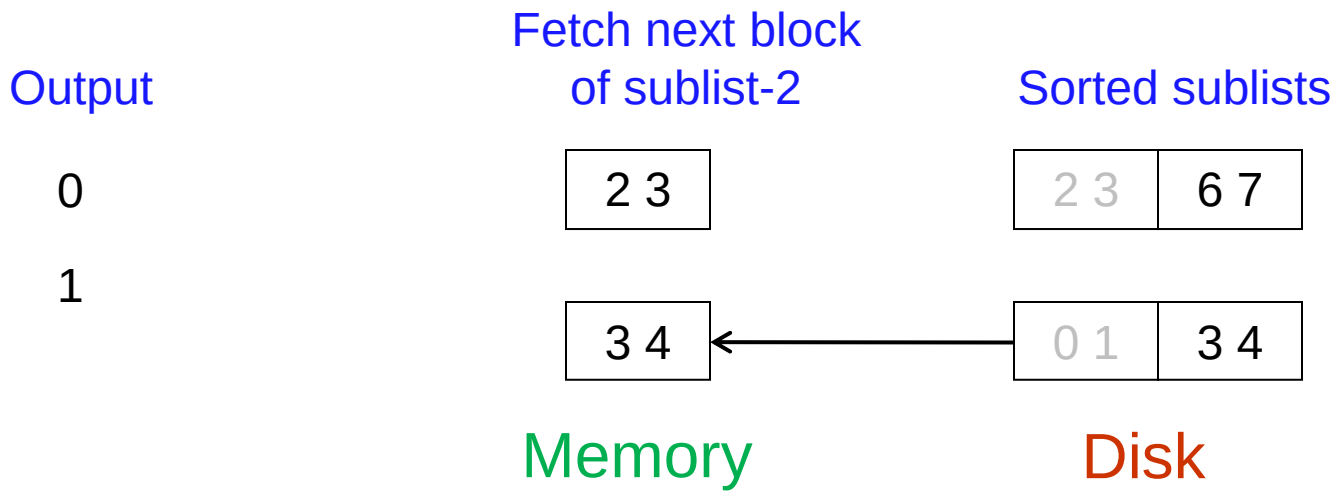
Two-phase Sort: Phase 2 (merge)



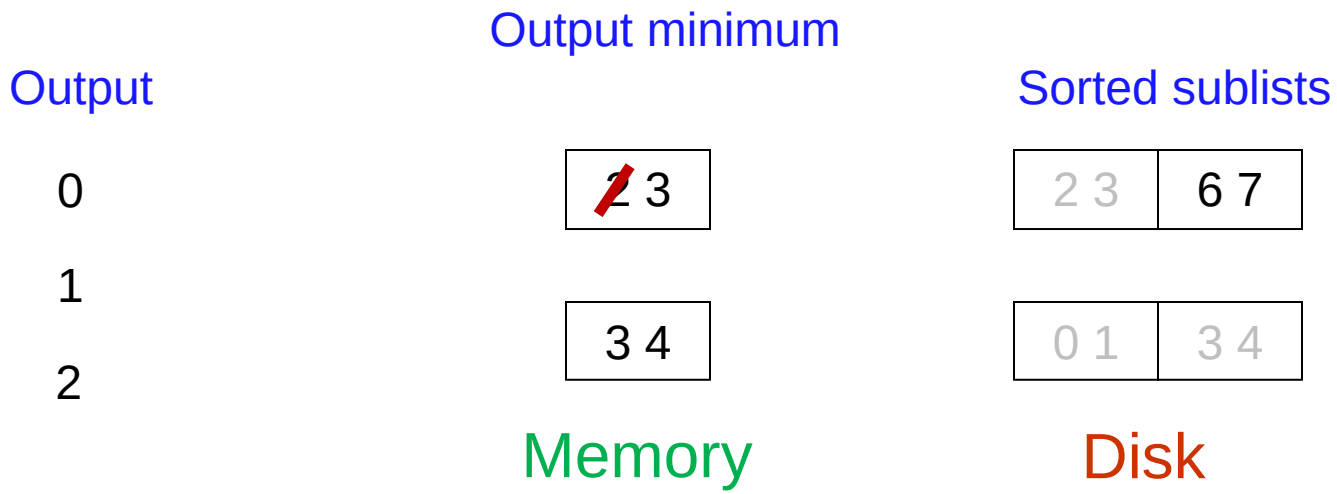
Two-phase Sort: Phase 2 (merge)



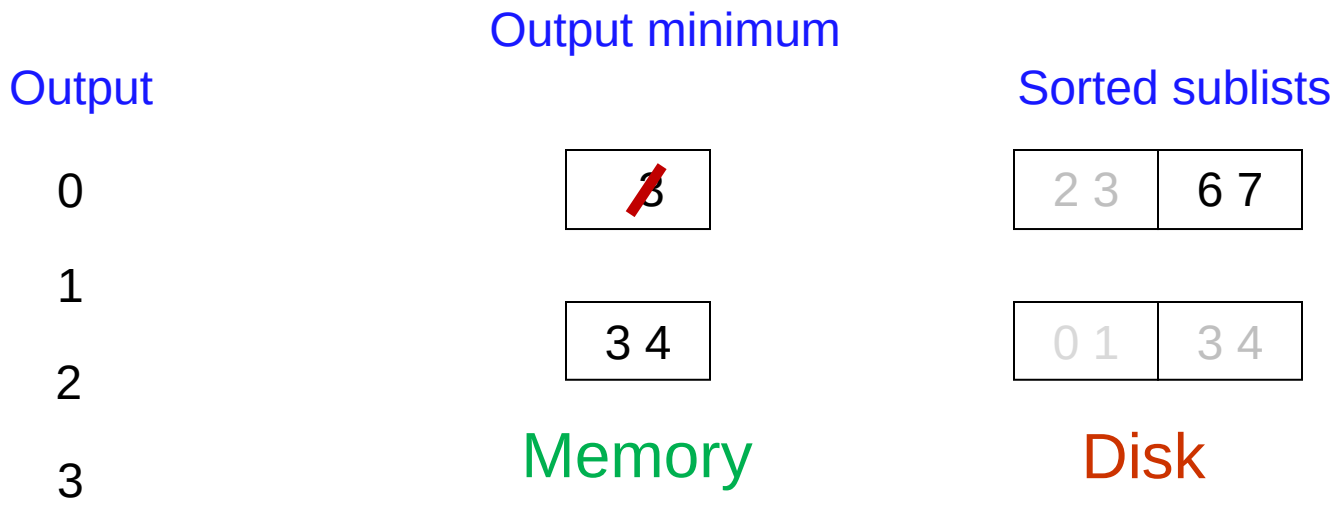
Two-phase Sort: Phase 2 (merge)



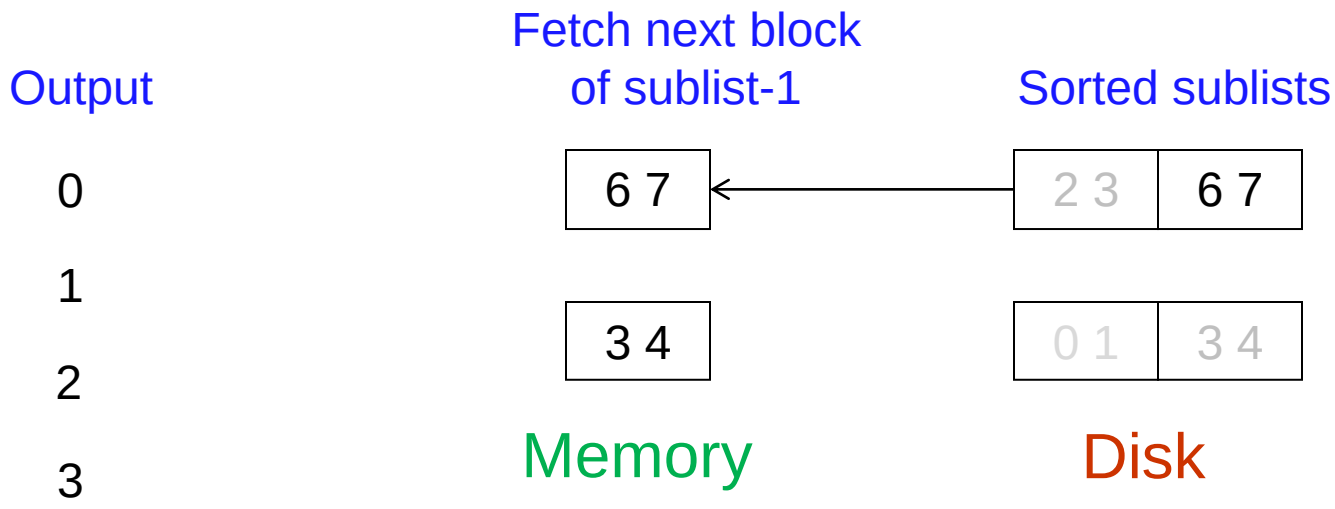
Two-phase Sort: Phase 2 (merge)



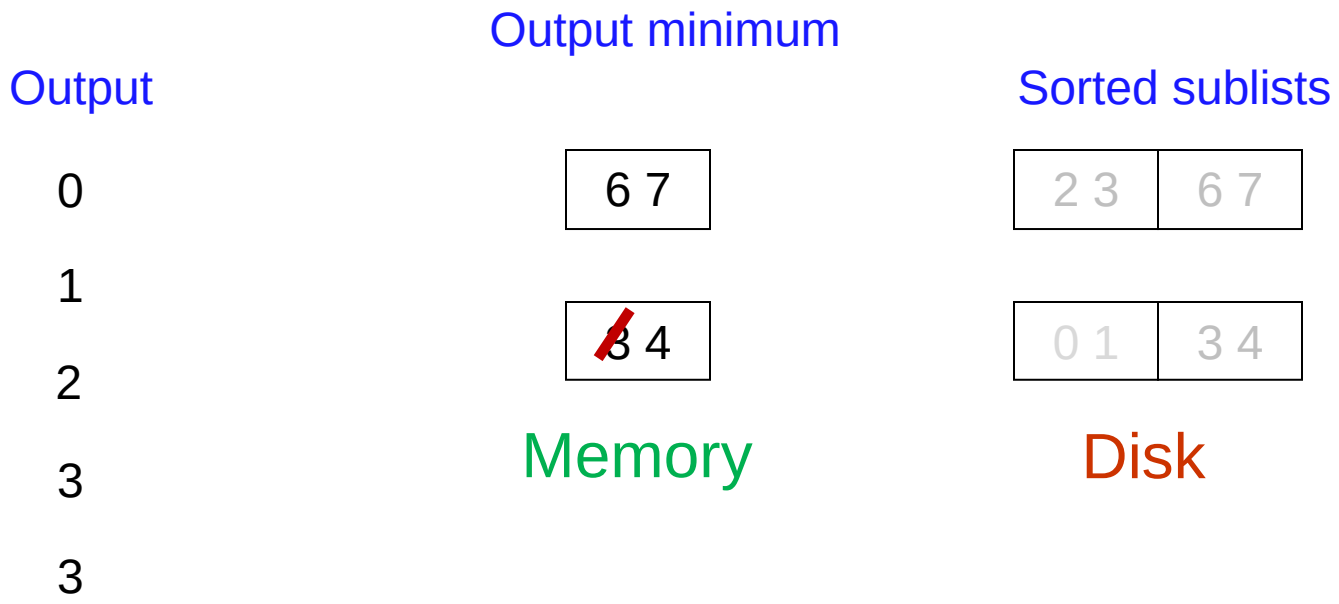
Two-phase Sort: Phase 2 (merge)



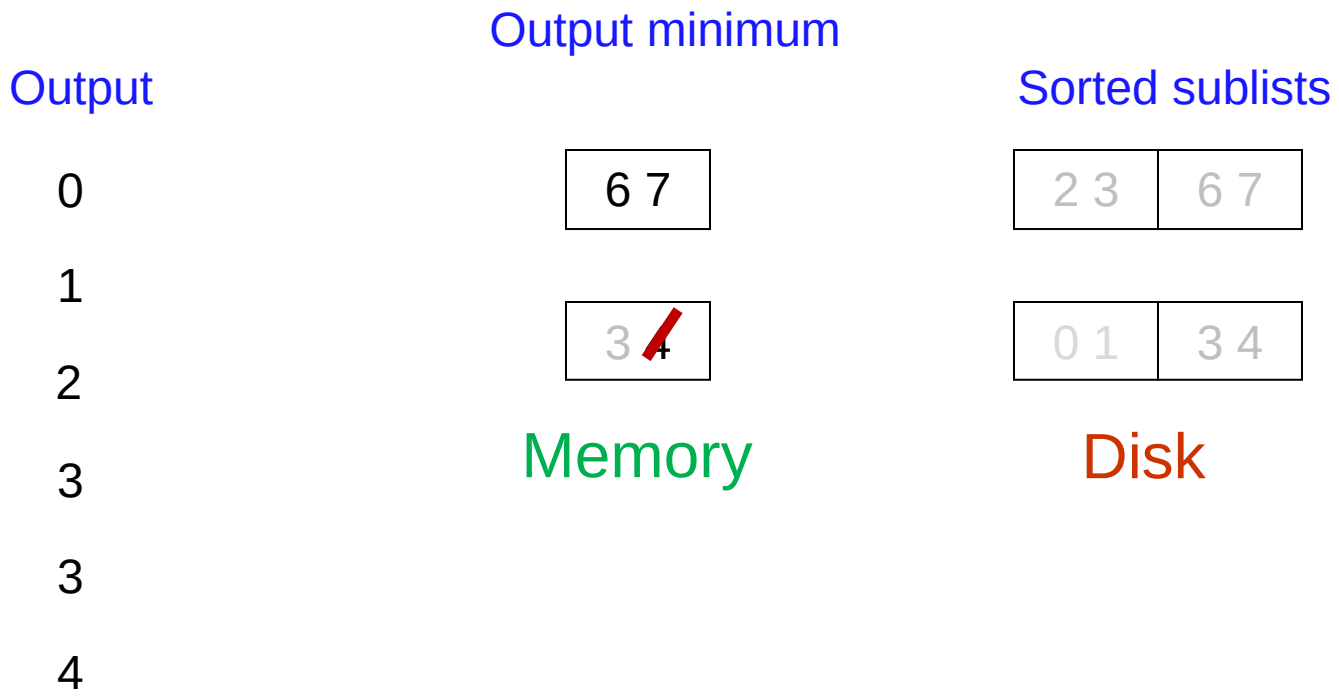
Two-phase Sort: Phase 2 (merge)



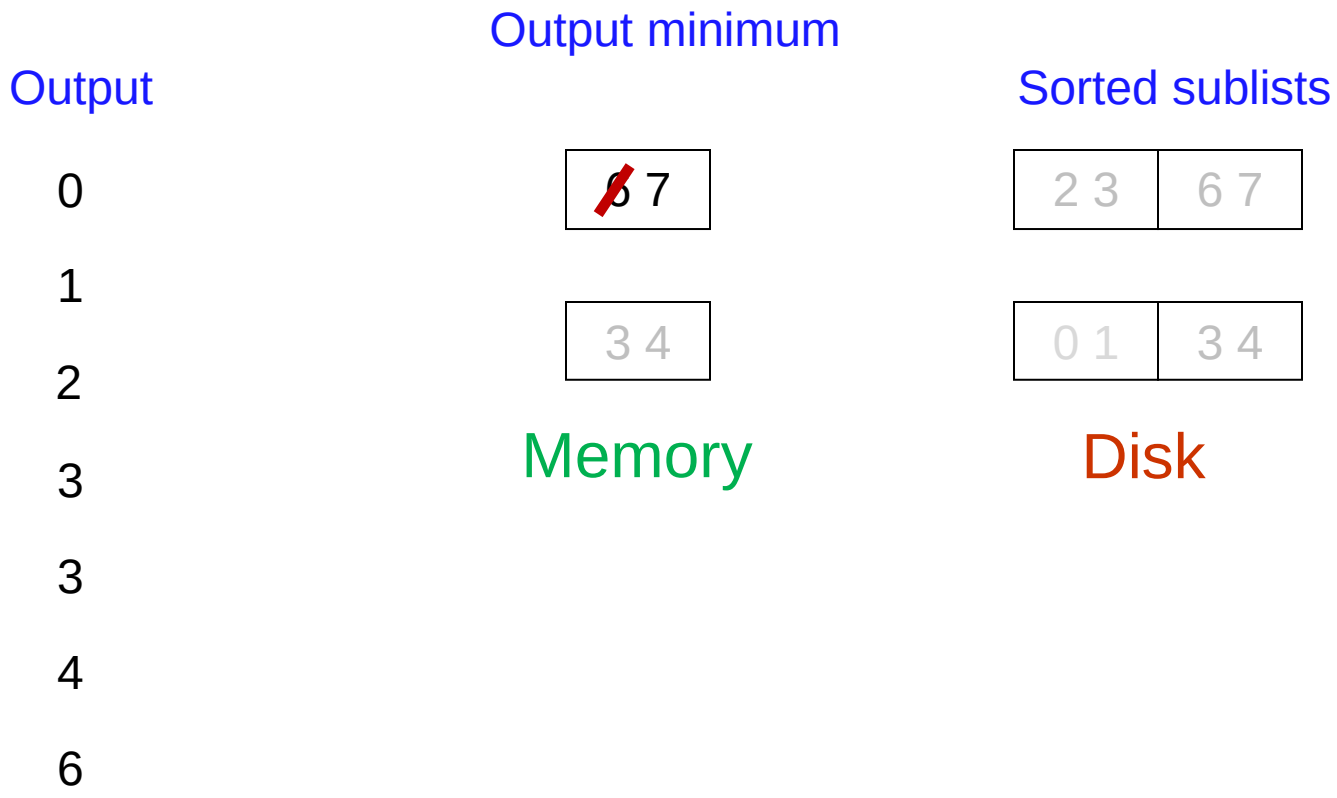
Two-phase Sort: Phase 2 (merge)



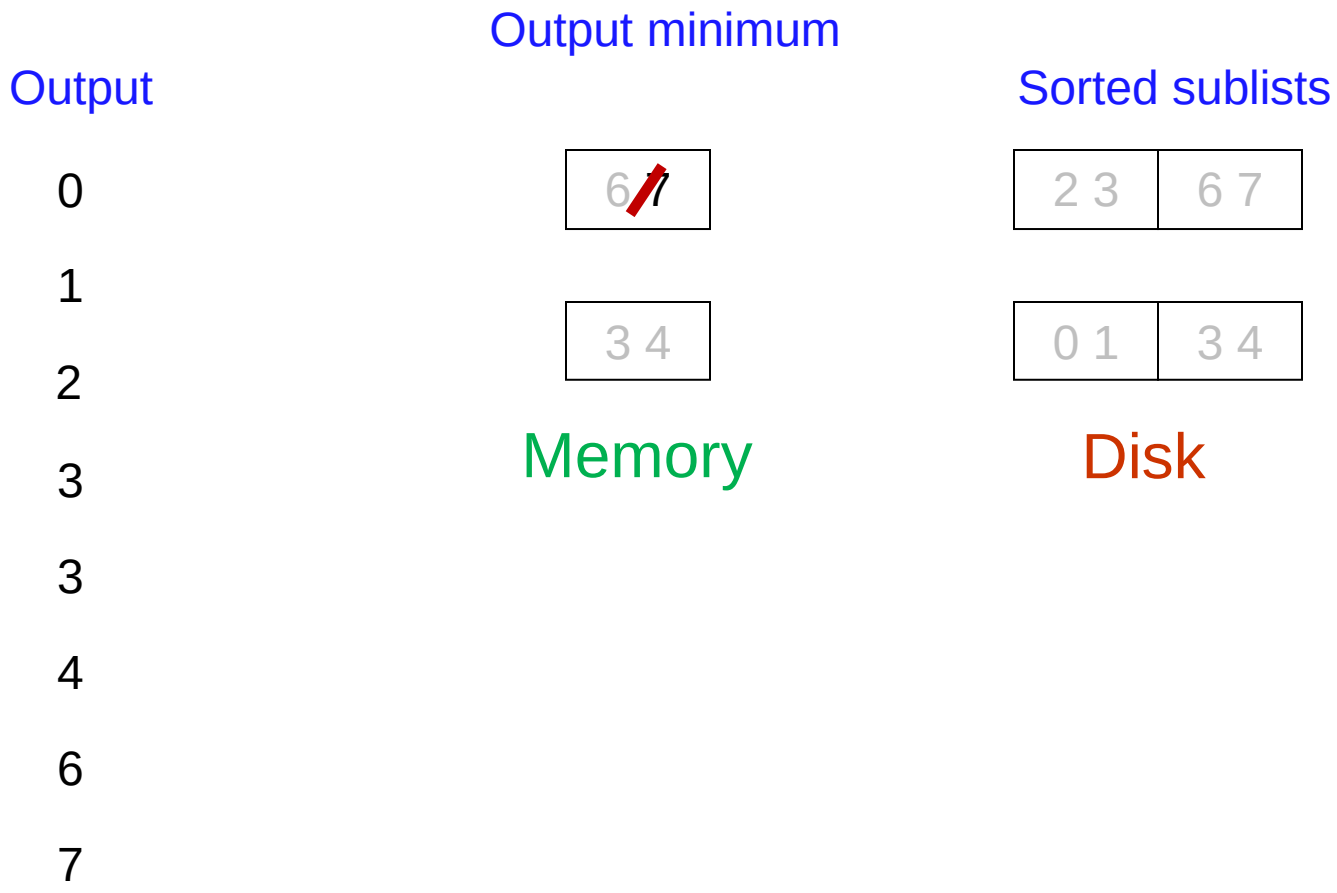
Two-phase Sort: Phase 2 (merge)



Two-phase Sort: Phase 2 (merge)



Two-phase Sort: Phase 2 (merge)

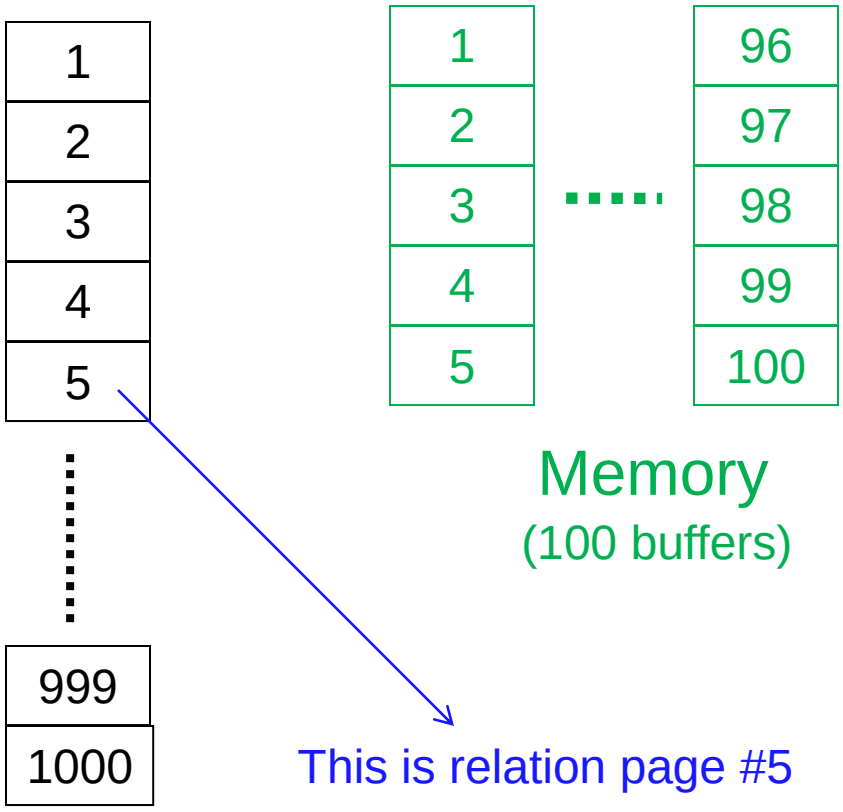


Φάση συγχώνευσης (merge)

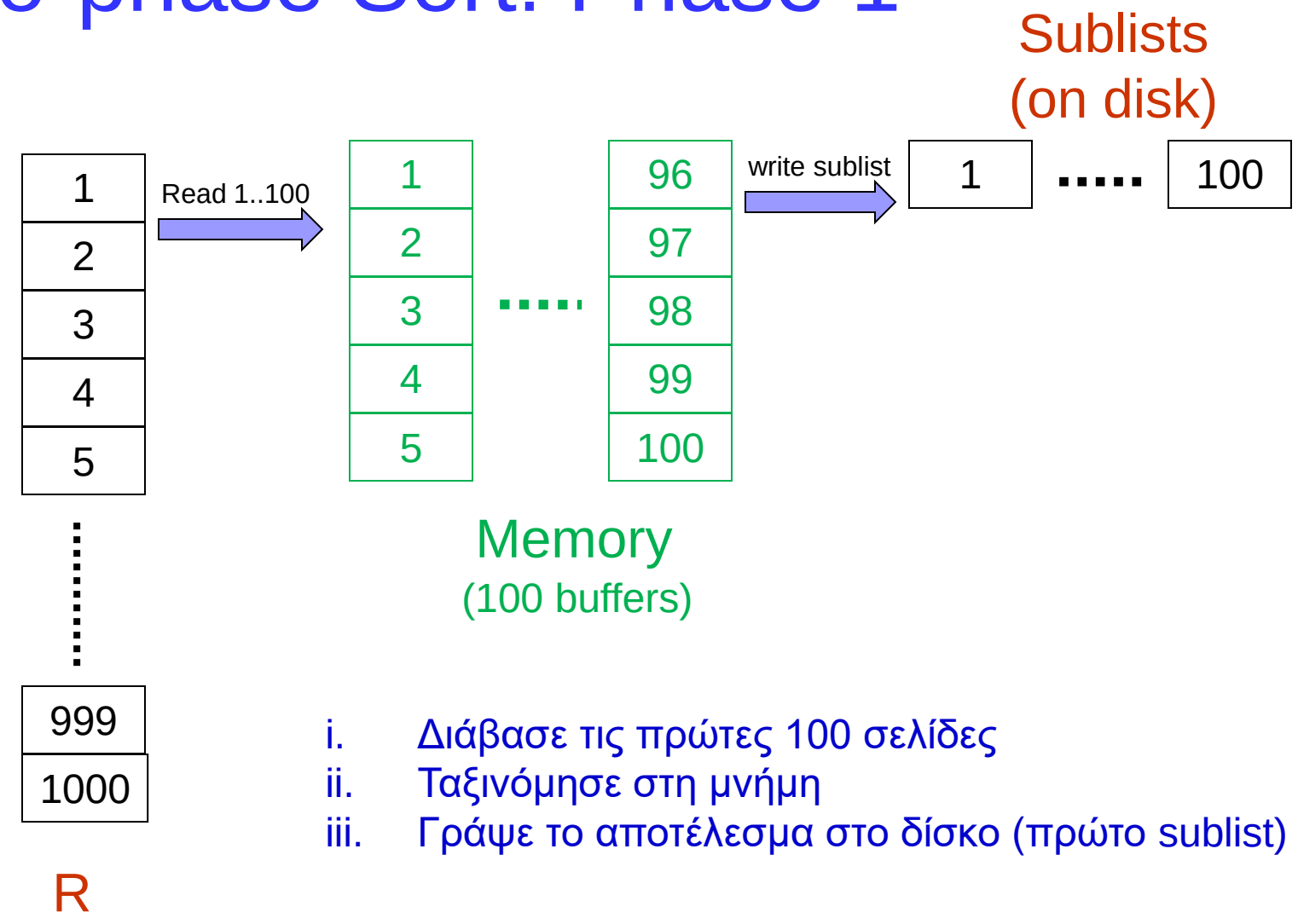
- Ας υποθέσουμε ότι με το τέλος της πρώτης φάσης έχω δημιουργήσει κ ταξινομημένες λίστες (sorted sublists)
  - Θυμήσου: στο δίσκο τα δεδομένα τα γράφω σε σελίδες
- Διαβάζω στη μνήμη την πρώτη σελίδα από κάθε λίστα
  - Χρειάζομαι κ σελίδες (input buffers) στη μνήμη κάθε φορά
  - Συγκρίνω την πρώτη εγγραφή από κάθε σελίδα (είναι η μικρότερη τιμή γι' αυτό το sublist) με αυτές των υπολοίπων σελίδων
    - Τραβάω την μικρότερη και μετακινώ τον δείκτη στην επόμενη εγγραφή αυτού του sublist
    - Όταν για κάποιο sublist έχω εξετάσει όλες τις εγγραφές του από τη σελίδα που έχω στη μνήμη, την πετάω και διαβάζω την επόμενη σελίδα γι αυτό από το δίσκο

Two-phase Sort: Phase 1

Παράδειγμα B(R) = 1000 σελίδες και M = 100 σελίδες

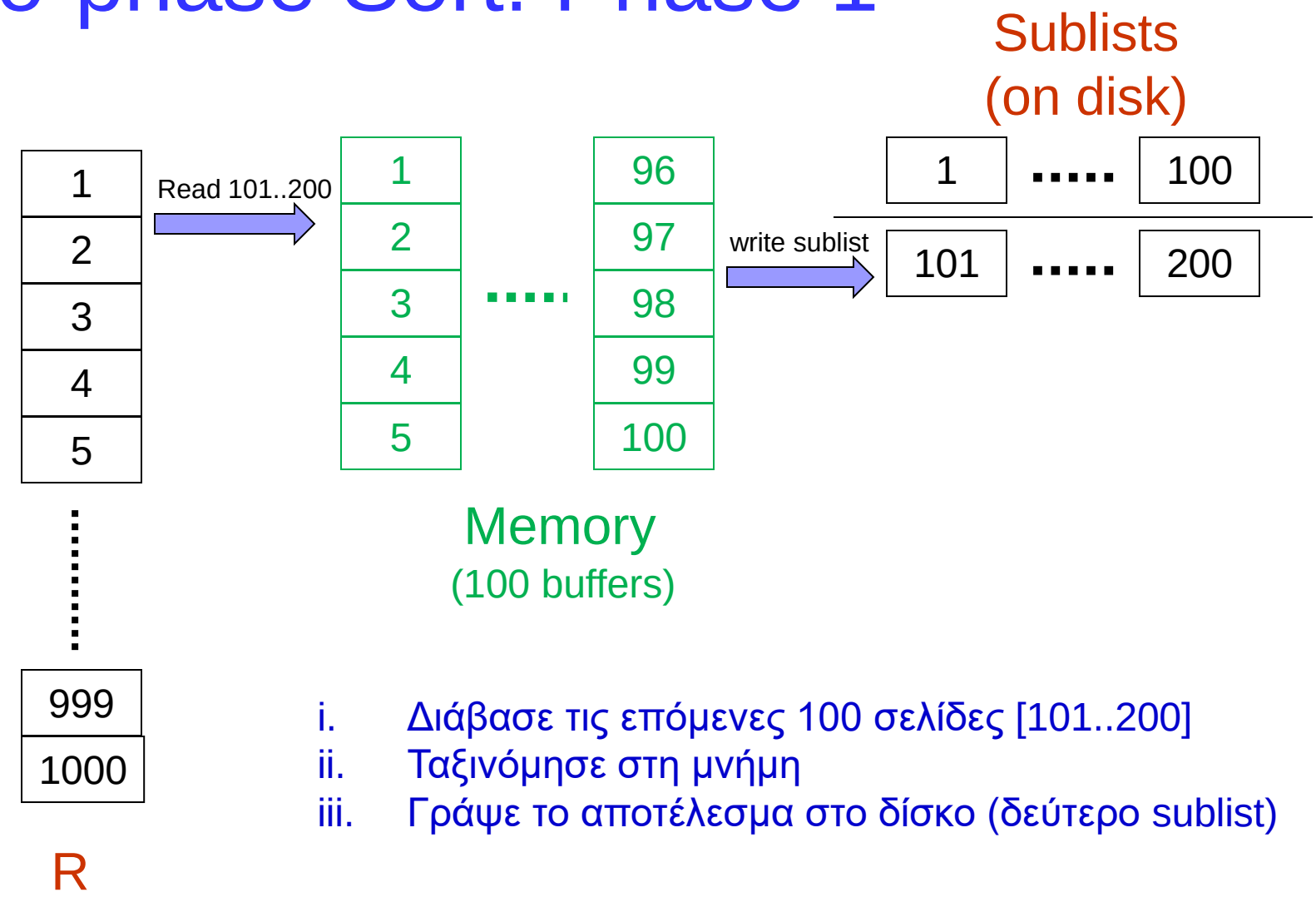


Two-phase Sort: Phase 1



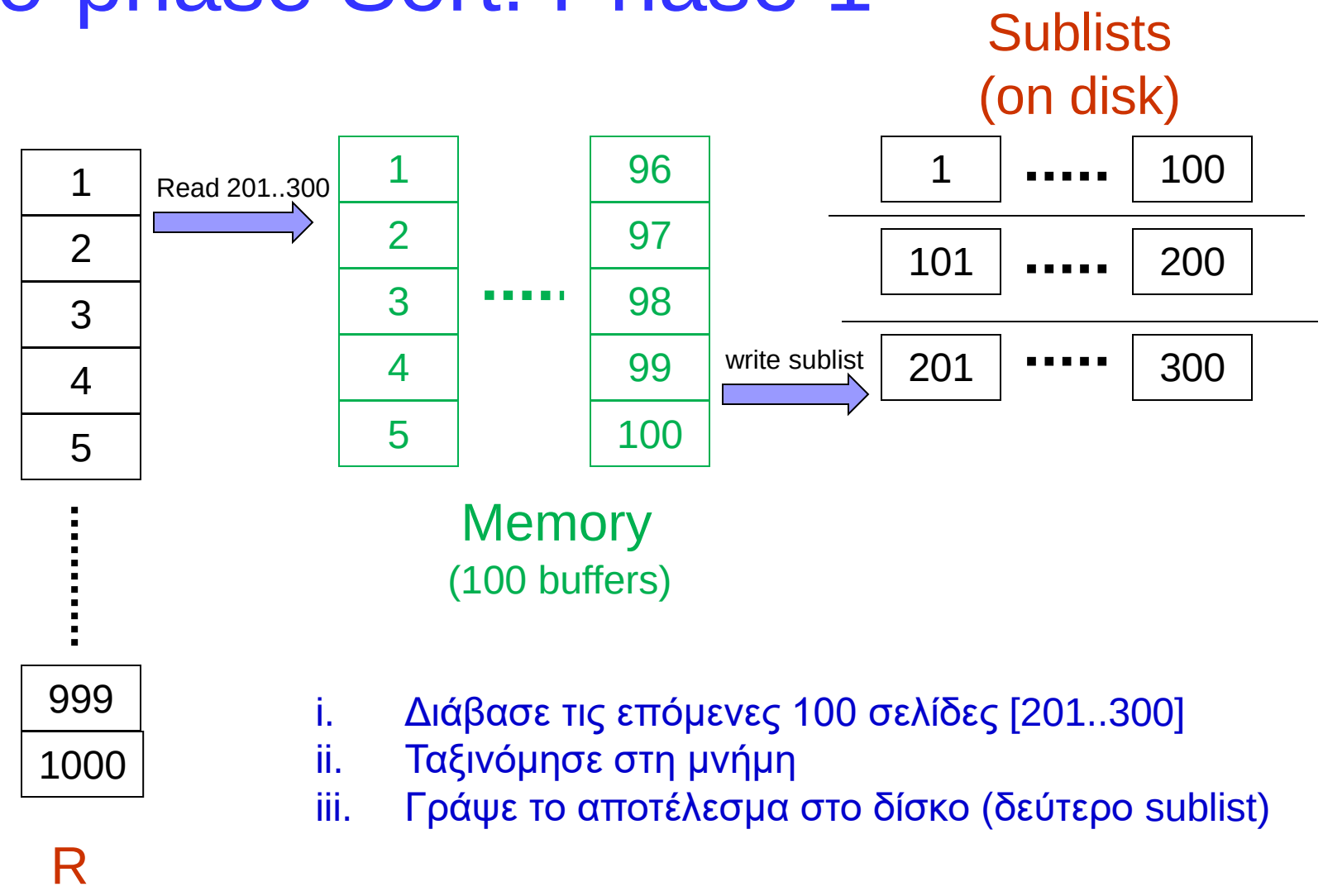
i. Διάβασε τις πρώτες 100 σελίδες
ii. Ταξινόμησε στη μνήμη
iii. Γράψε το αποτέλεσμα στο δίσκο (πρώτο sublist)

Two-phase Sort: Phase 1



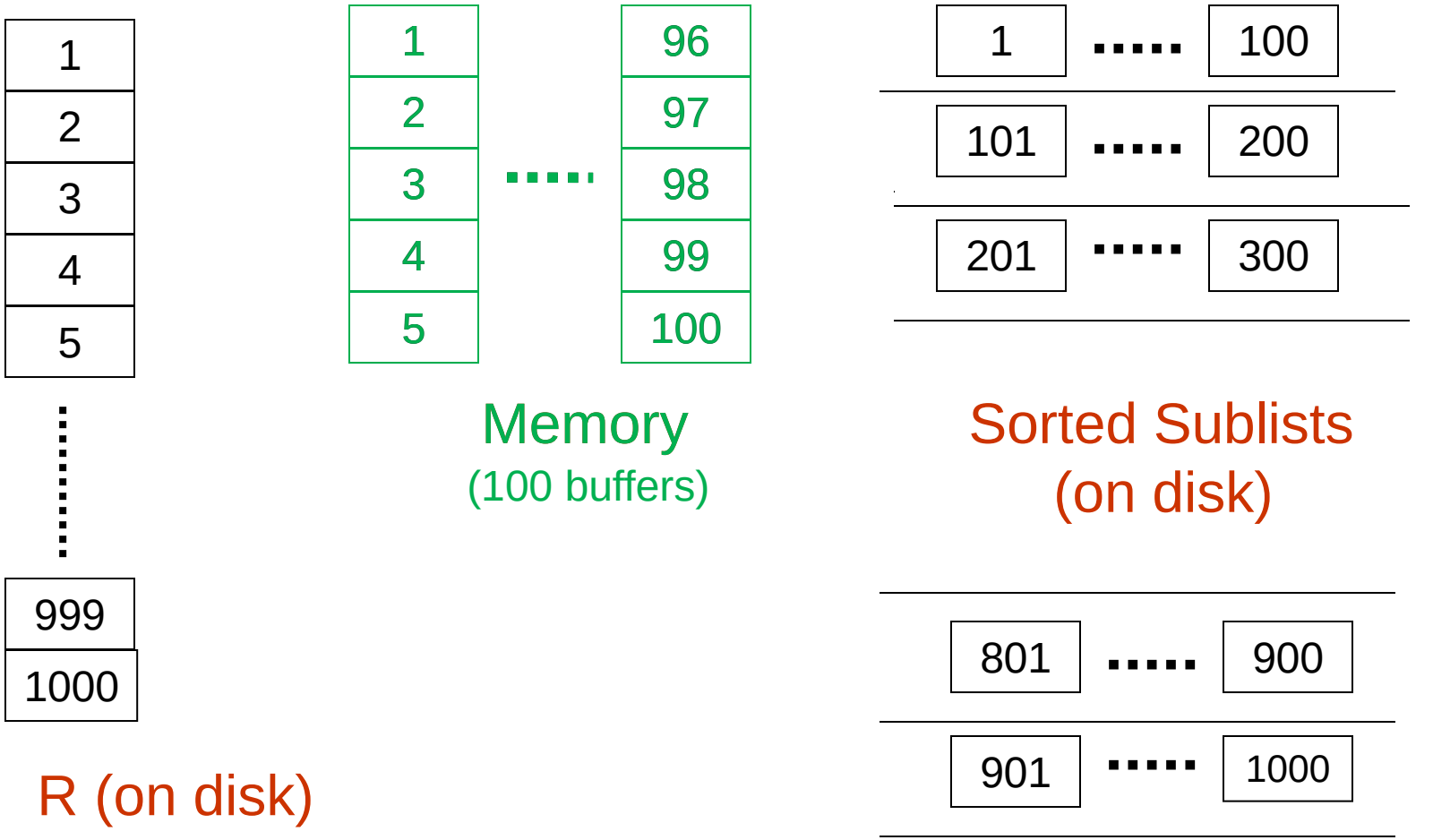
i. Διάβασε τις επόμενες 100 σελίδες [101..200]
ii. Ταξινόμησε στη μνήμη
iii. Γράψε το αποτέλεσμα στο δίσκο (δεύτερο sublist)

Two-phase Sort: Phase 1

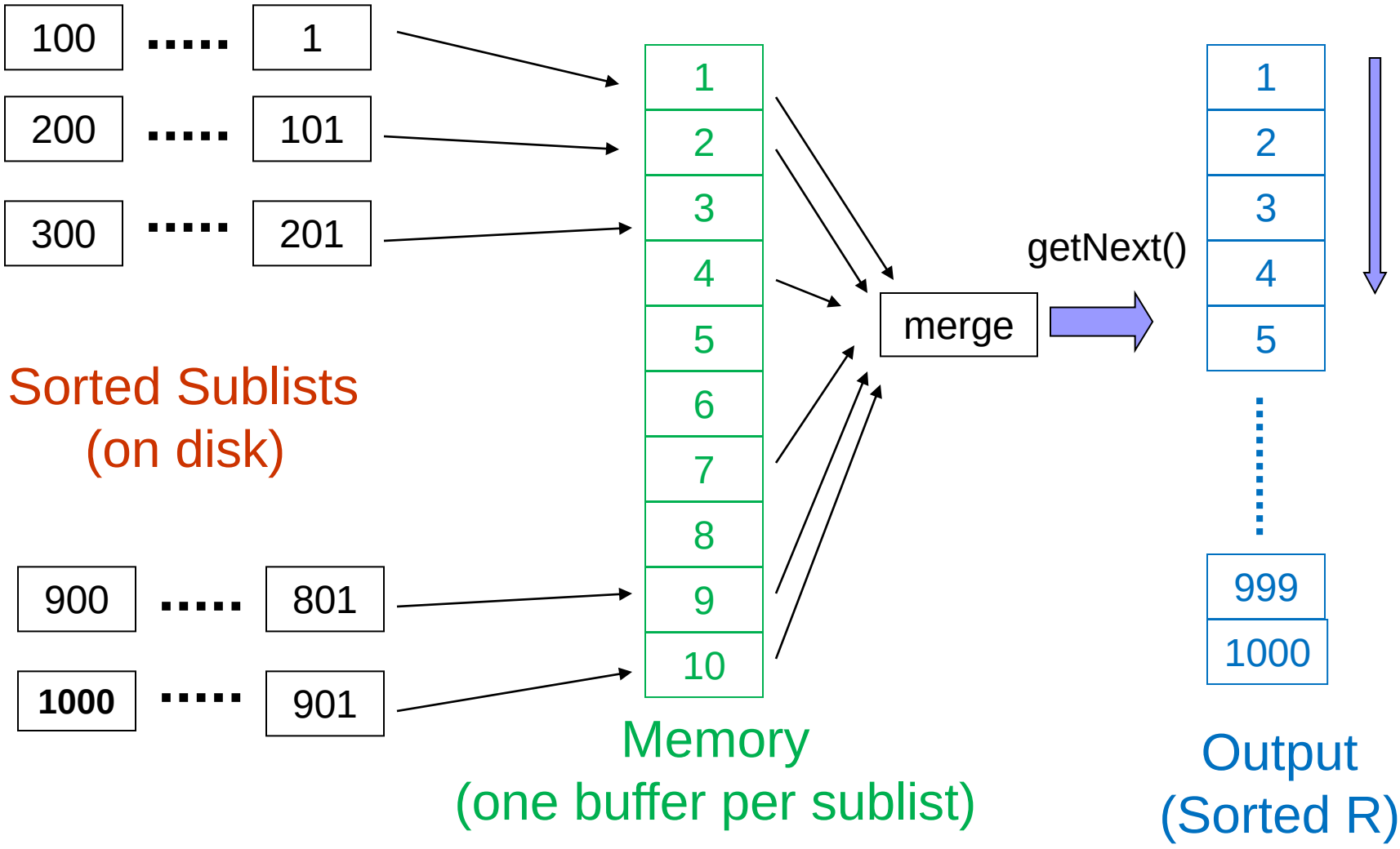


i. Διάβασε τις επόμενες 100 σελίδες [201..300]
ii. Ταξινόμησε στη μνήμη
iii. Γράψε το αποτέλεσμα στο δίσκο (δεύτερο sublist)

Two-phase Sort: End of Phase 1



Two-phase Sort: Phase 2



Κόστος (σε I/O) αλγορίθμου Two-Phase Sort

- Κάθε εγγραφή τη διαβάζω 1 φορά στη φάση-1 (sort) και 1 φορά στη φάση-2 (merge)
- Κάθε εγγραφή τη γράφω σε ένα sorted sublist
- Ο όγκος των εγγραφών σε σελίδες είναι B(R)
  - Κάνω B(R) reads + B(R) writes + B(R) reads

φάση-1 φάση-2

→ Κόστος= 3xB(R)

Απαιτήσεις σε μνήμη

- Πρέπει $M \geq \sqrt{B(R)}$
- Απόδειξη:
  - Κάθε sublist έχει μέγεθος M (σε αριθμό σελίδων)
  - Στη φάση 2 χρειάζομαι ένα buffer για κάθε sublist (δες παράδειγμα).
    - Άρα τα sublists είναι το πολύ M
  - Το μέγιστο μέγεθος δεδομένων προς ταξινόμηση είναι M sublists μεγέθους M σελίδες το καθένα δηλαδή

$$B(R) \leq M^2 \Leftrightarrow M \geq \sqrt{B(R)}$$

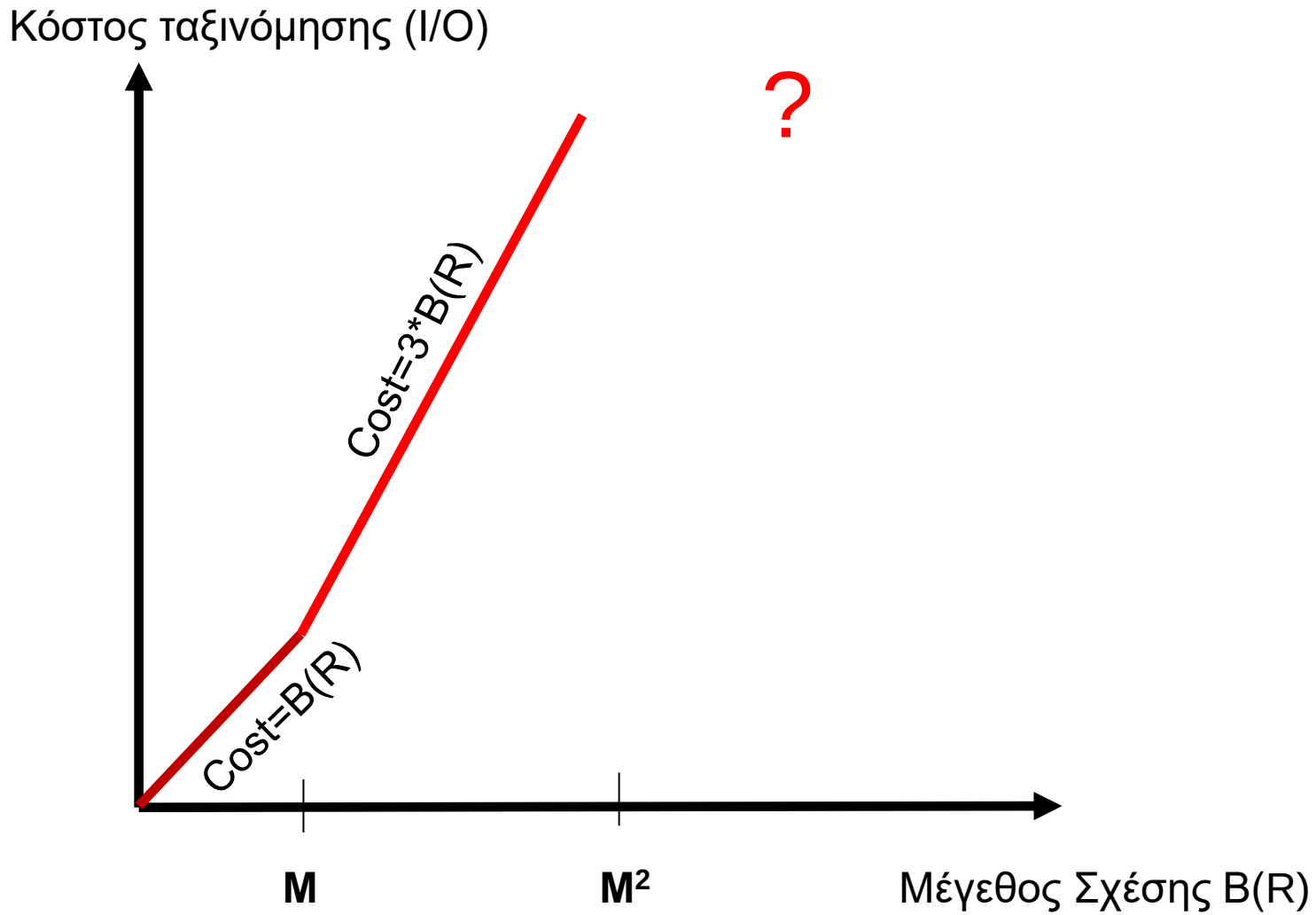
Τι σημαίνει αυτό στην πράξη;

- Έστω μέγεθος σελίδας 8KB
- Έστω μνήμη 4GB, δηλαδή $\frac{4\ GBytes}{8\ KBytes} = \frac{4*1024*1024*\cancel{1024}}{8*\cancel{1024}} =$ 524288 σελίδες των 8KB
- Μπορώ να ταξινομήσω σε 2 φάσεις σχέσεις μεγέθους ≤ 524288^2 * 8KB = 2048TB
- Έστω μέγεθος εγγραφής 128 bytes
- Μέγιστο μέγεθος πίνακα για ταξινόμηση: $\frac{2048\ TBytes}{128\ Bytes}$=17,592,186,044,416 εγγραφές

Παράδειγμα

- Η μεγαλύτερη σχέση στη βάση μου είναι B(R) = 2GB = 262144 σελίδες των 8 KB
- Αν ο sort buffer είναι τουλάχιστον 2GB η ταξινόμηση γίνεται σε ένα πέρασμα με κόστος B(R)
- Αν δεν έχω τόση μνήμη, αρκεί να θέσω για το sort buffer $M = \sqrt{B(R)} = 512$ σελίδες ή 4MB!!!!
- Είτε δώσω 4MB μνήμης είτε 1.99GB το κόστος της ταξινόμησης σε I/O θα είναι 3*B(R)
  - Φυσικά αν το ΣΔΒΔ εκτελεί παράλληλα πολλές επερωτήσεις οι οποίες χρησιμοποιούν ταξινόμηση, μεγαλώνουν ανάλογα και οι συνολικές απαιτήσεις σε μνήμη
  - Η συγχώνευση στο δεύτερο βήμα είναι στη πράξη ποιο γρήγορη αν οι λίστες είναι λιγότερες (γιατί?) παρότι τα I/Os είναι τα ίδια

Ας σκεφτούμε



Εφαρμογές ταξινόμησης

- Απαλοιφή διπλοτύπων
  - SELECT DISTINCT …
- Συνάθροιση
- Σύζευξη
- …

Απαλοιφή διπλοτύπων (duplicate elimination)

- Έστω η σχέση **Buys(ProdId,CustId,Date,Amt)**
- Θέλω να βρω τα προϊόντα με τουλάχιστον μία πώληση:

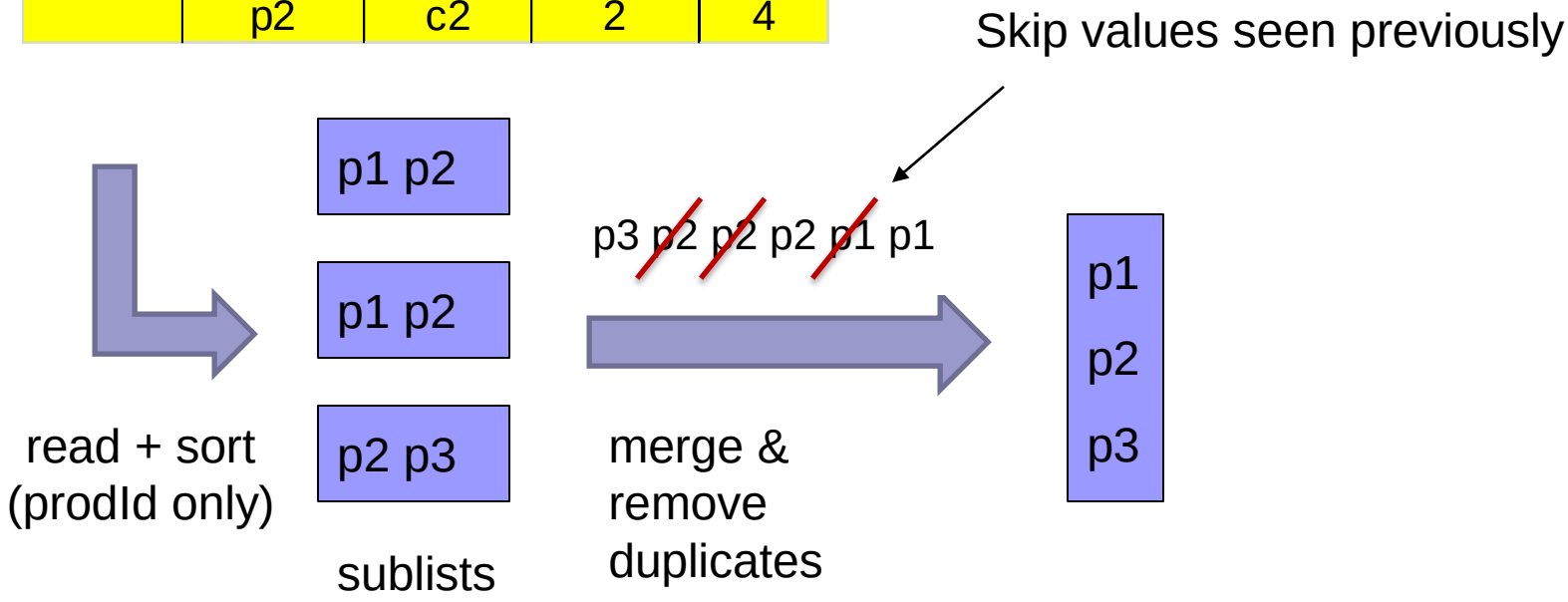
Select Distinct ProdId from Buys

Μία λύση

- Διάβασε τις εγγραφές τις σχέσης κρατώντας μόνο το ProdId
  - Ταξινόμησε τις εγγραφές με βάση το γνώρισμα ProdId
  - Κατά τη δημιουργία των sublists και στη φάση της συγχώνευσης απαλείφουμε διπλοεγγραφές
- Τελικά η απαλοιφή διπλοεγγραφών ανάγεται σε πρόβλημα ταξινόμησης

Select Distinct ProdId from Buys

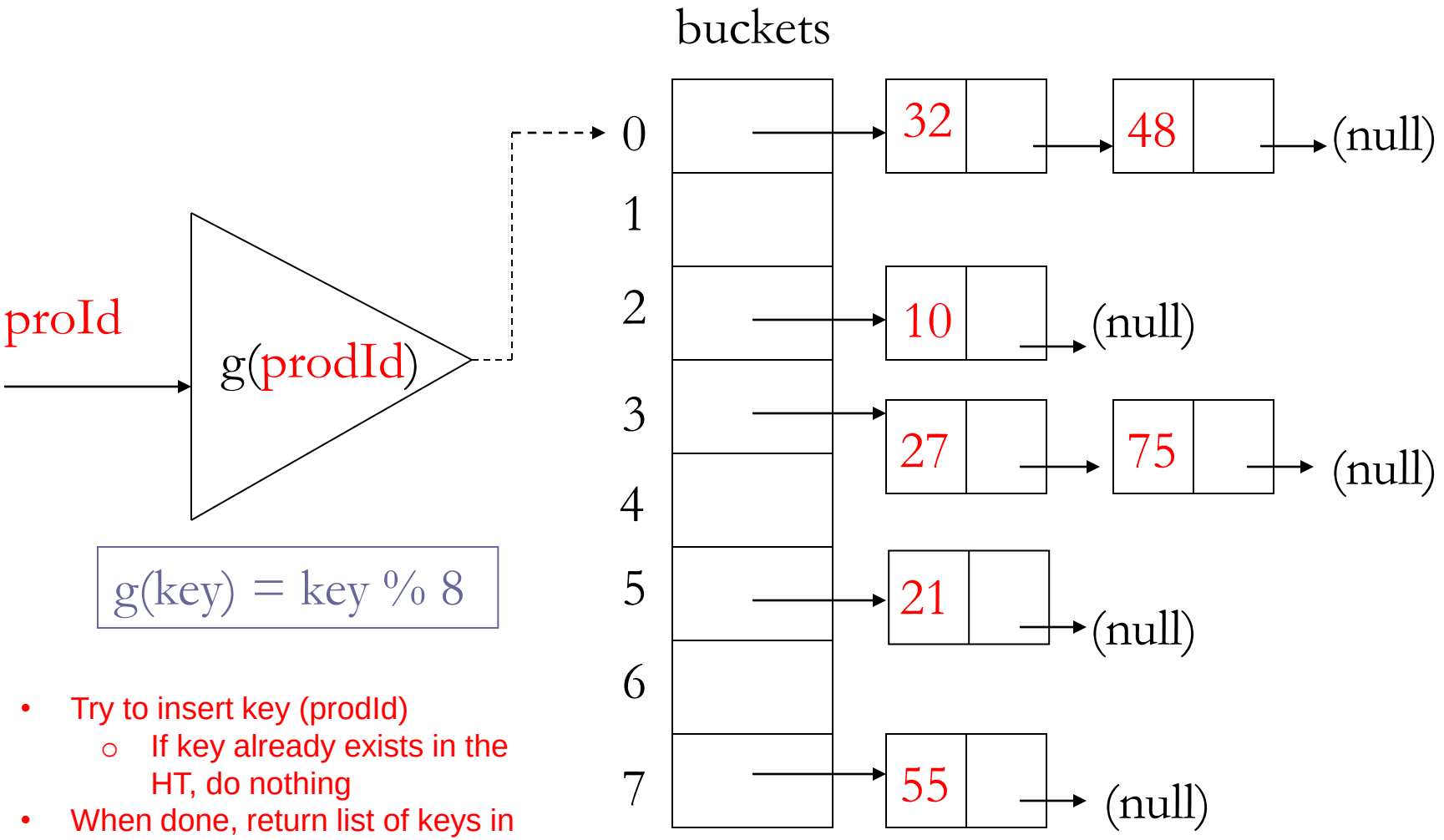
Buys	prodId	custId	date	amt
	p1	c1	1	12
	p2	c1	1	11
	p2	c3	1	50
	p1	c2	1	8
	p3	c1	2	44
	p2	c2	2	4



Λύση για το distinct χωρίς ταξινόμηση

- ?

Hash Table for key=prodId (assuming HT fits in main memory)

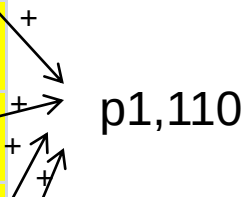


- Try to insert key (prodId)
  - If key already exists in the HT, do nothing
- When done, return list of keys in the HT

Εφαρμογή 2: Υπολογισμός συνάθροισης

- Έστω ο παρακάτω πίνακας πωλήσεων
  - Θέλω να υπολογίσω τις συνολικές πωλήσεις ανά προϊόν

sale	prodId	custId	date	amt
	p1	c1	1	12
	p2	c1	1	11
	p1	c3	1	50
	p2	c2	1	8
	p1	c1	2	44
	p1	c2	2	4



SQL?

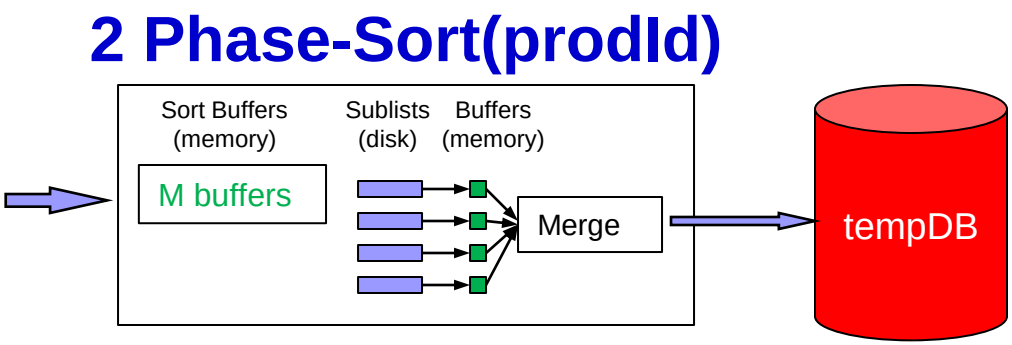
```
select prodId,SUM(amt)
from sale
group by prodId
```

Μία απλή λύση με χρήση 2-Phase-Sort

- Ταξινομώ με τον 2-Phase-Sort : 3B(R)
- Γράφω το αποτέλεσμα στο δίσκο : +B(R)
- Το διαβάσω και κάνω τη συνάθροιση αθροίζοντας πωλήσεις με ίδια τιμή στο prodId: +B(R)
  - Εύκολο γιατί οι εγγραφές είναι ταξινομημένες
- Κόστος = 3B(R)+B(R)+B(R) = 5B(R)

Βήμα 1: Ταξινόμησε με βάση το prodId, γράψε το αποτέλεσμα στο δίσκο

sale	prodId	custId	date	amt
	p1	c1	1	12
	p2	c1	1	11
	p1	c3	1	50
	p2	c2	1	8
	p1	c1	2	44
	p1	c2	2	4



tempDB

sortedSale	prodId	custId	date	amt
	p1	c1	1	12
	p1	c1	2	44
	p1	c2	2	4
	p1	c3	1	50
	p2	c1	1	11
	p2	c2	1	8

Βήμα 2:Συνάθροισε εγγραφές με την ίδια τιμή στο prodId

Διάβασε ταξινομημένες εγγραφές από δίσκο

sortedSale	prodId	custId	date	amt
	p1	c1	1	12
	p1	c1	2	44
	p1	c2	2	4
	p1	c3	1	50
	p2	c1	1	11
	p2	c2	1	8

Πωλήσεις για prodId=p1

Aggregate (memory)

Result

ans	prodId	sum
	p1	110
	p2	19

Παρατήρηση

- Η συνάθροιση SUM(amt) μπορεί να γίνει κατά τη δημιουργία των sublists και στη φάση της συγχώνευσης του αλγορίθμου two-phase sort
  - Δε χρειάζεται να γράψω στο δίσκο το ταξινομημένο αποτέλεσμα και να το διαβάσω μετά για να κάνω τη συνάθροιση. Αυτή γίνεται μαζί με την ταξινόμηση
- Κόστος βελτιωμένης υλοποίησης: 3B(R) αντί για 5B(R)

Calculating the SUM() while sorting

Output of 2-phase sort algorithm
(one row at a time)

Maintain partial sum

output

prodId	storeId	date	amt
p1	s1	1	12
p1	s1	2	44
p1	s2	2	4
p1	s3	1	50
p2	s1	1	11
p2	s2	1	8

+

SUM
12
56
60
110
11
19

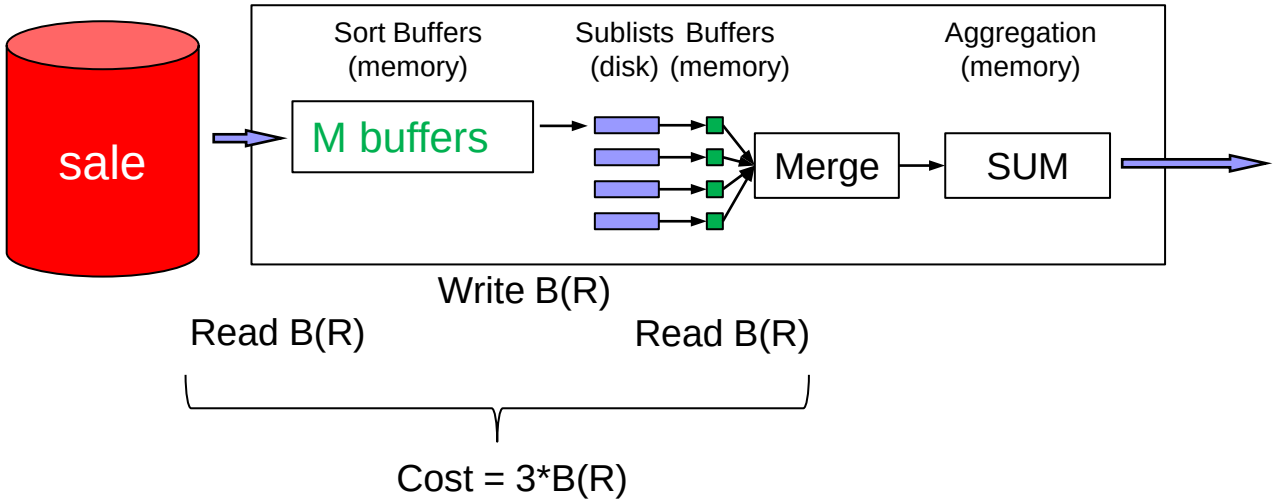
p1,110

p2,19

EOT (End-Of-Table)

- Each time we see a new prodId value we report the previous product id with the current value of SUM and then initialize SUM to the new amt
- Report last combination at EOT

I/O Optimized Group By Computation (for $M < B(R) \leq M^2$)



ans	prodId	sum
	p1	110
	p2	19

Additional optimizations?

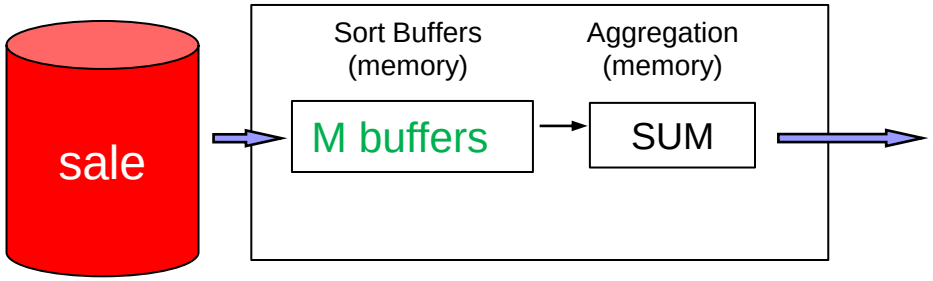
sale	prodId	custId	date	amt
	p1	c1	1	12
	p1	c1	2	44
	p1	c2	2	4
	p1	c3	1	50
	p2	c1	1	11
	p2	c2	1	8

Result

ans	prodId	sum
	p1	110
	p2	19

select prodId,SUM(amt)
from sale
group by prodId

Av M≥B(R)?



ans	prodId	sum
	p1	110
	p2	19

Read B(R)

Cost = B(R)

Άρα

- Ερωτήσεις με group by μπορεί να υπολογισθούν μέσω ταξινόμησης

Ερώτηση: Για ποιες συναρτήσεις δουλεύει η βελτιωμένη λύση?

- Υποθέσαμε την SUM()
- Δουλεύει για την AVERAGE()?
- Για τη MEDIAN()?

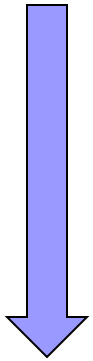
sale	prodId	custId	date	amt
	p1	c1	1	12
	p1	c1	2	44
	p1	c2	2	4
	p1	c3	1	50
	p2	c1	1	11
	p2	c2	1	8

Σύζευξη μέσω ταξινόμησης

R1(A,B,C)⋈R2(C,D)

Σχέση R1

R1.C
… 10
… 20
… 20
… 30
… 40



R2.C
5 ….
20 …
20 …
30 …
30 …
50 …
52 …

Σχέση R2

Σύζευξη «εύκολη» αν οι σχέσεις είναι ταξινομημένες στο κοινό γνώρισμα

Αλγόριθμος Sort-Merge Join (SMJ)

Algorithm SMJ(R1,R2)

//Βήμα 1: Sort

//Βήμα 2: MergeJoin

(1) if R1 and R2 not sorted, sort them

(2) $i \leftarrow 1$; $j \leftarrow 1$;

While $(i \leq T(R1)) \wedge (j \leq T(R2))$ do

if R1{ i }.C = R2{ j }.C then OutputTuples

else if R1{ i }.C > R2{ j }.C then $j \leftarrow j+1$

else if R1{ i }.C < R2{ j }.C then $i \leftarrow i+1$

Procedure Output-Tuples

While (R1{ i }.C = R2{ j }.C) $\wedge$ (i $\leq$ T(R1)) do

[jj $\leftarrow$ j;

while (R1{ i }.C = R2{ jj }.C) $\wedge$ (jj $\leq$ T(R2)) do

[output pair R1{ i }, R2{ jj };

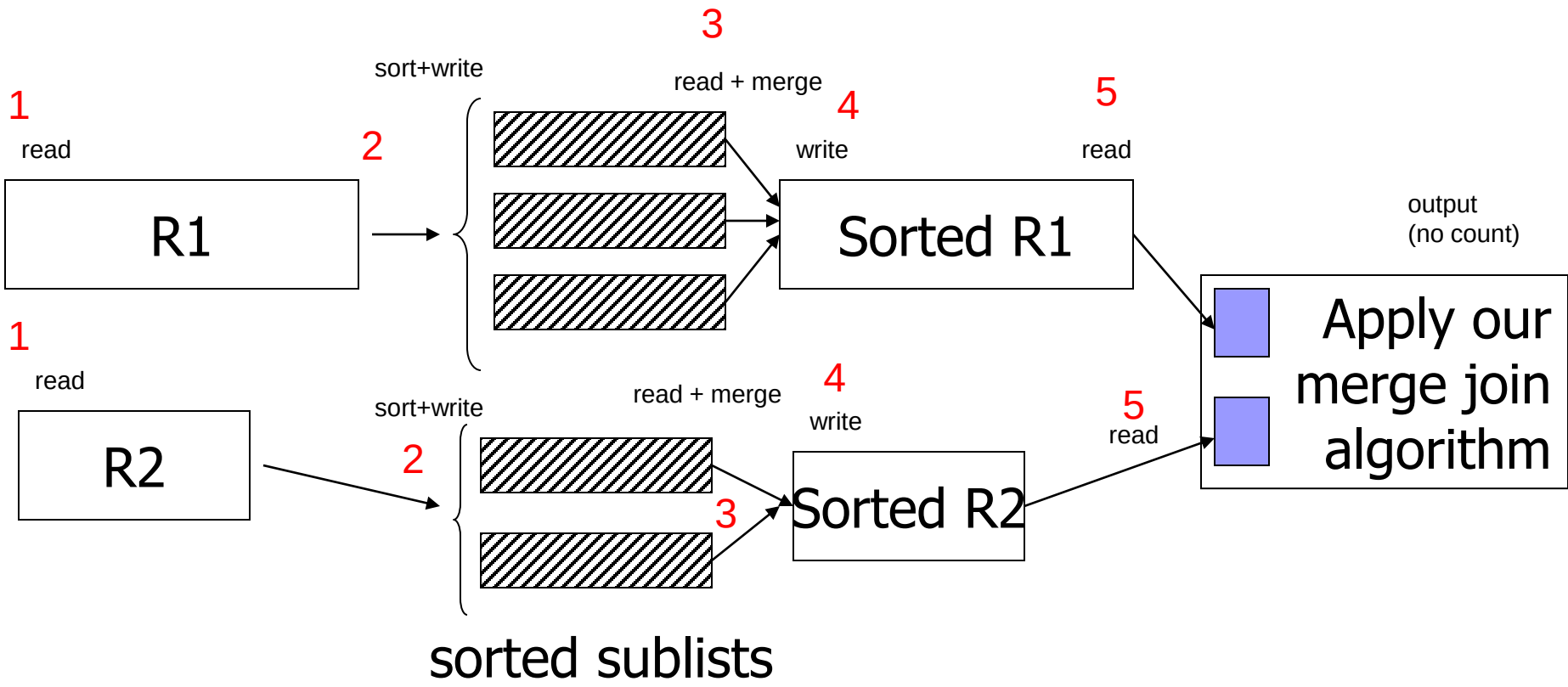
jj $\leftarrow$ jj+1]

i $\leftarrow$ i+1]

Παράδειγμα

i	R1{i}.C	R2{j}.C	j
1	10	5	1
2	20	20	2
3	20	20	3
4	30	30	4
5	40	30	5
		50	6
		52	7

Sort-Merge Join

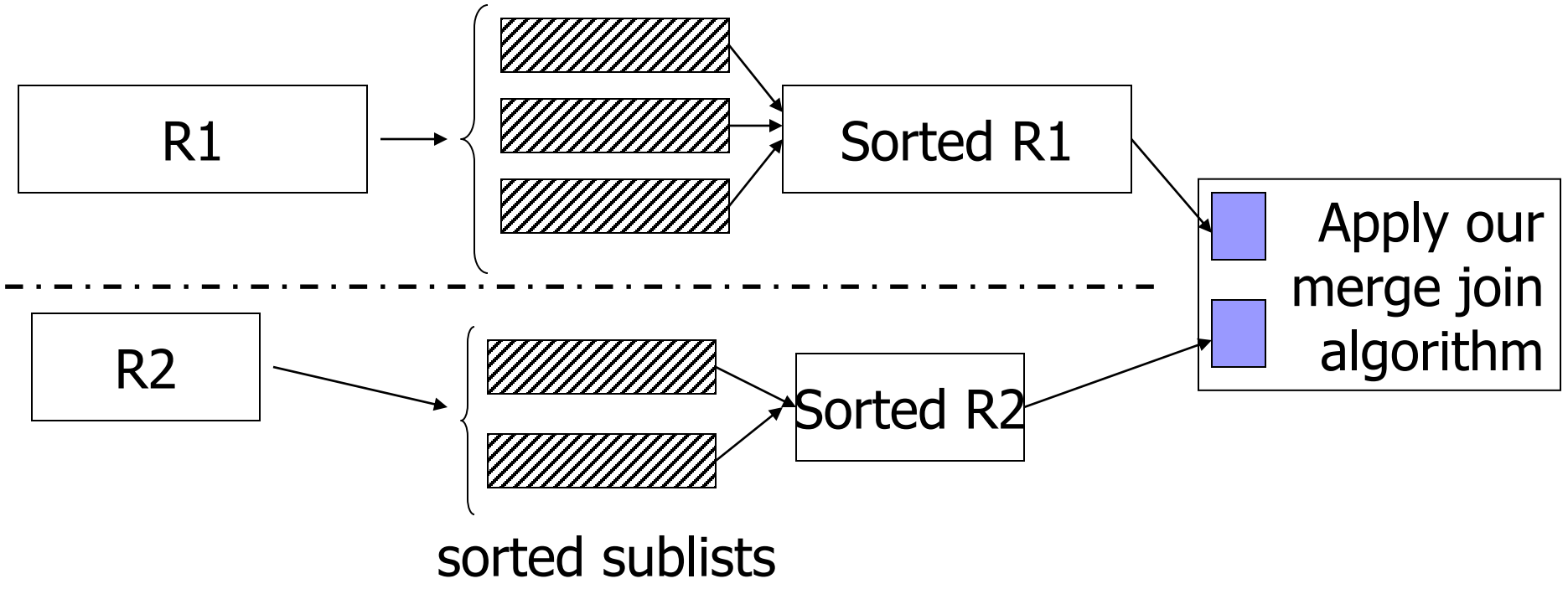


Ανάλυση

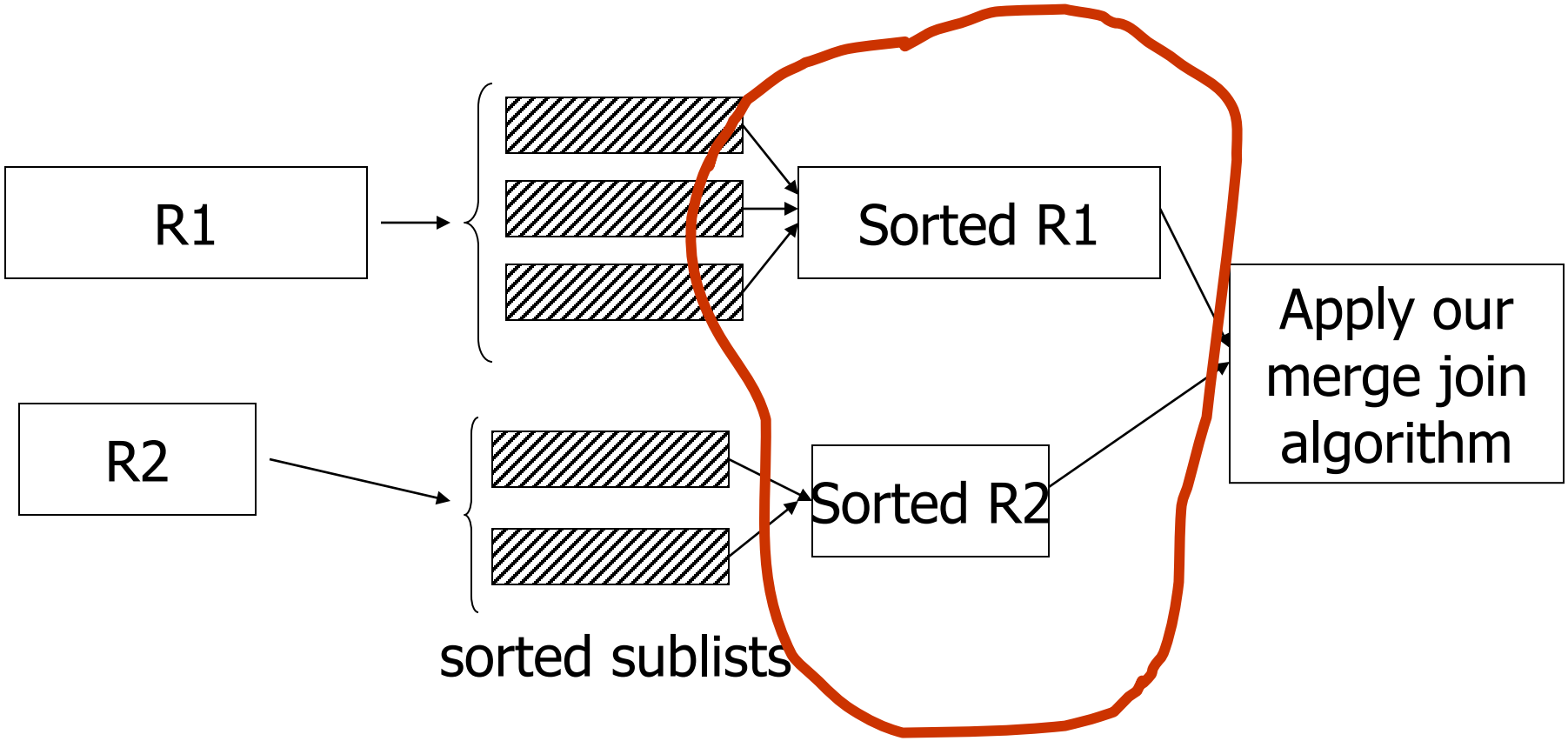
- Cost = 5 x (B(R1) + B(R2))
- Απαιτήσεις σε μνήμη :

$$M \geq \sqrt{\max(B(R1), B(R2))}$$

ΓΙΑΤΙ?

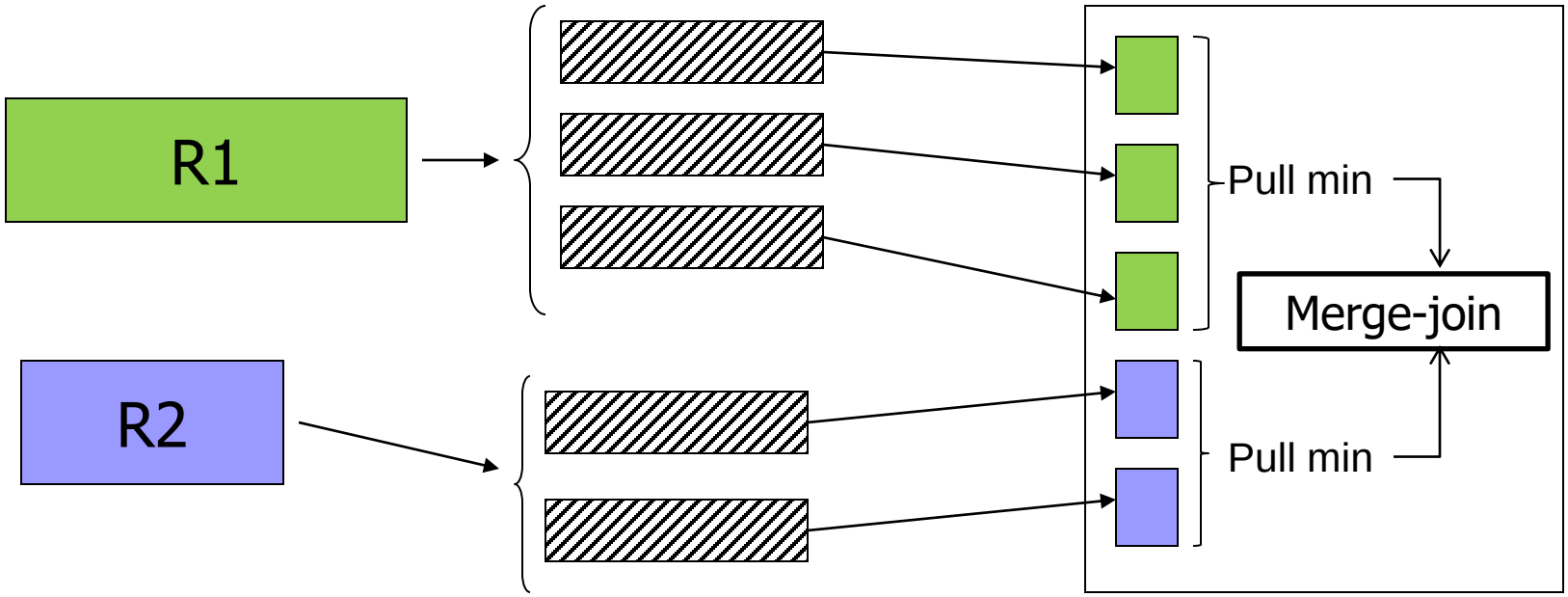


Μπορεί να γίνει ποιο αποδοτικός ο Sort-Merge Join?



Χρειάζεται να δημιουργήσουμε τις ταξινομημένες R1, R2 στο δίσκο?

Ποιο αποδοτική έκδοση του αλγορίθμου Sort-Merge Join



Ανάλυση της νέας έκδοσης του SMJ

- Μικρότερο κόστος: 3 x (B(R1) + B(R2))

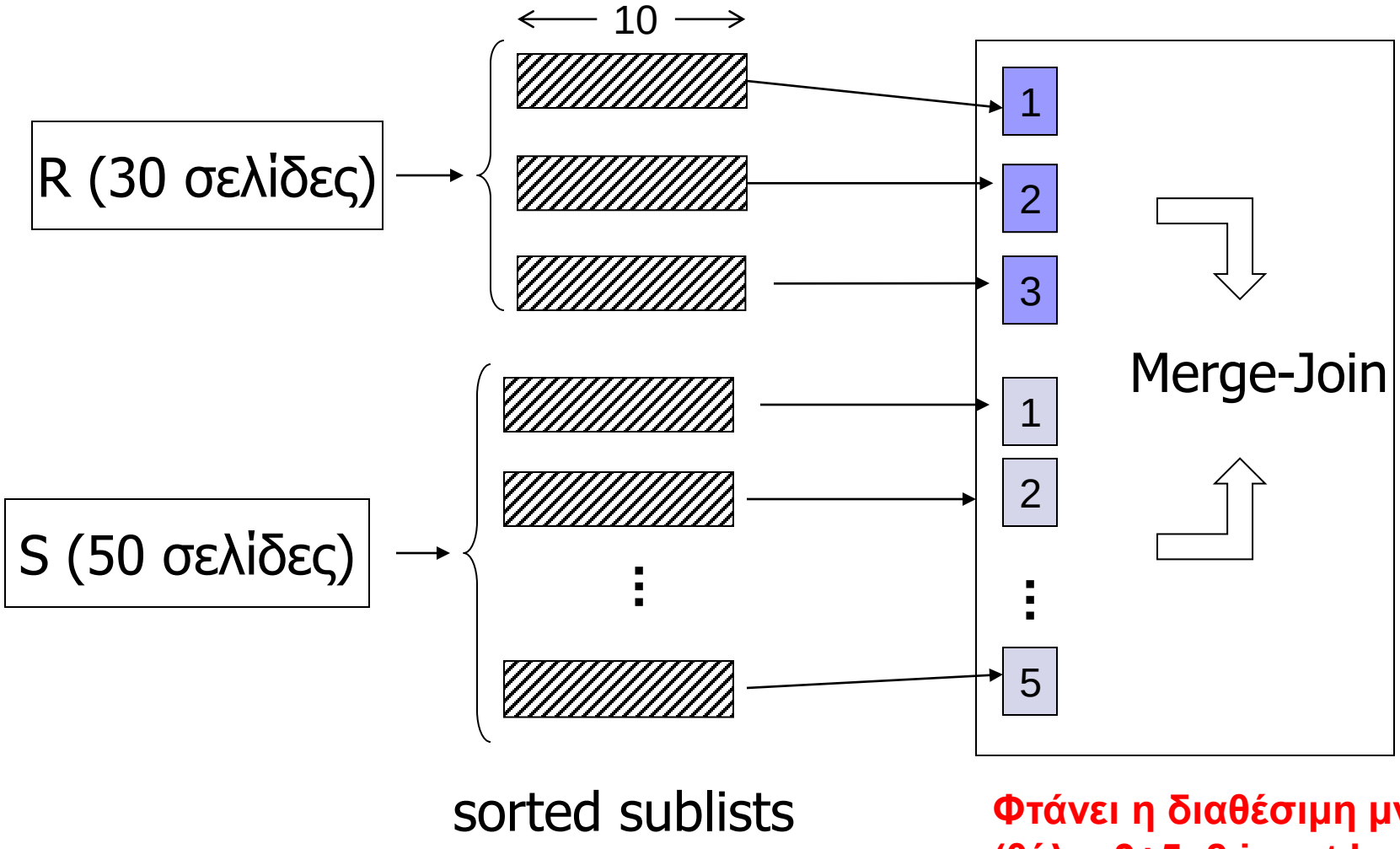
  [Αντί για 5 x (B(R1) + B(R2))]

- Λίγο μεγαλύτερες απαιτήσεις σε μνήμη:

  $M \geq \sqrt{B(R1) + B(R2)}$

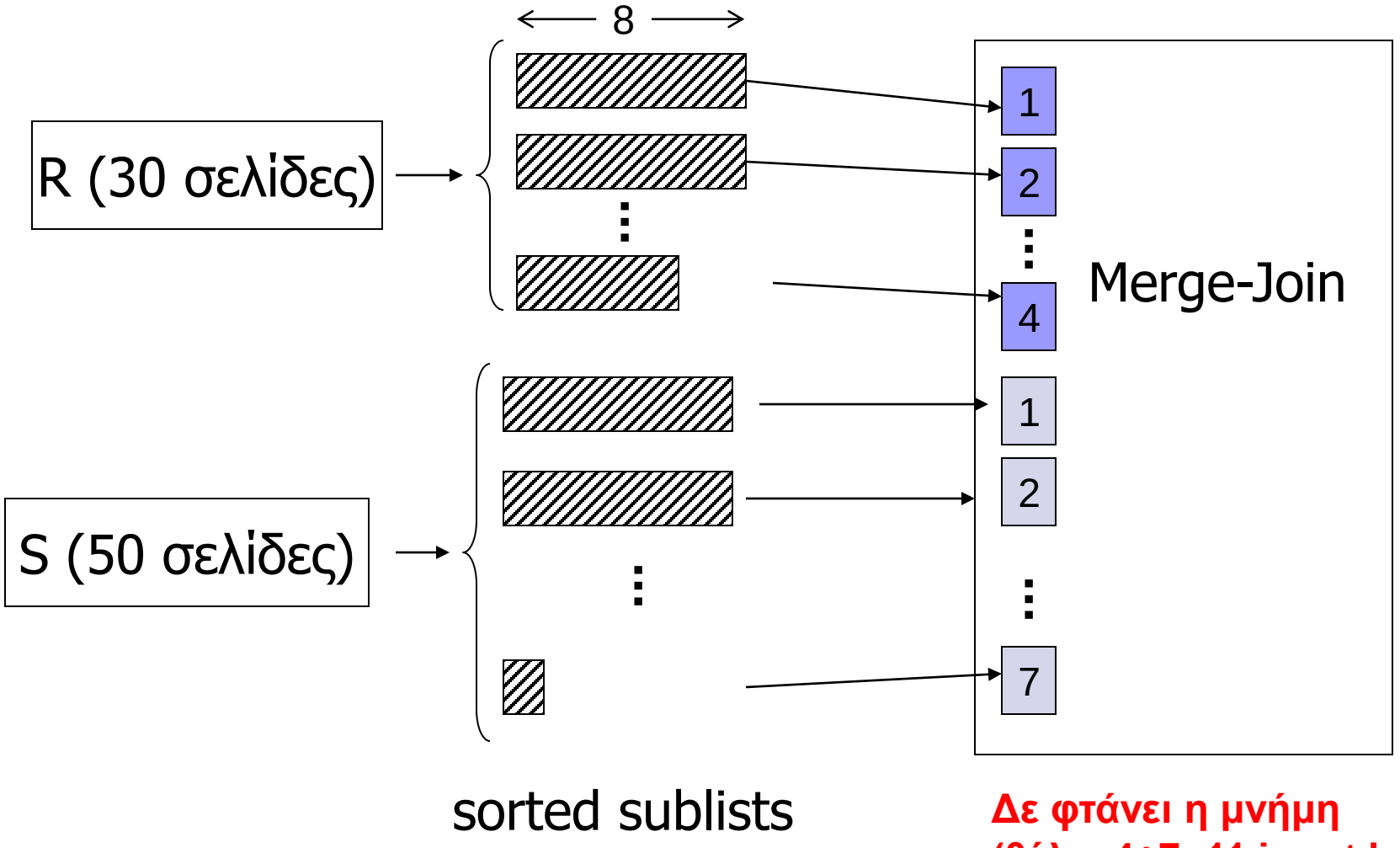
  [Αντί για $M \geq \sqrt{\max(B(R1), B(R2))}$

Αποδοτική έκδοση του αλγορίθμου Sort-Merge Join (M=10)?



Φτάνει η διαθέσιμη μνήμη (θέλω 3+5=8 input buffers)

Αποδοτική έκδοση του αλγορίθμου Sort-Merge Join (M=8)?



Δε φτάνει η μνήμη (θέλω 4+7=11 input buffers)