Overlap Set Similarity Join

Dong Deng

Problem Definition

- *Input*:
  - a collection of sets R
  - a constant integer threshold $\boldsymbol{c}$
- *Output*:
  - all pairs $(X, Y) \in R \times R \ s.t. |X \cap Y| \geq \boldsymbol{c}$

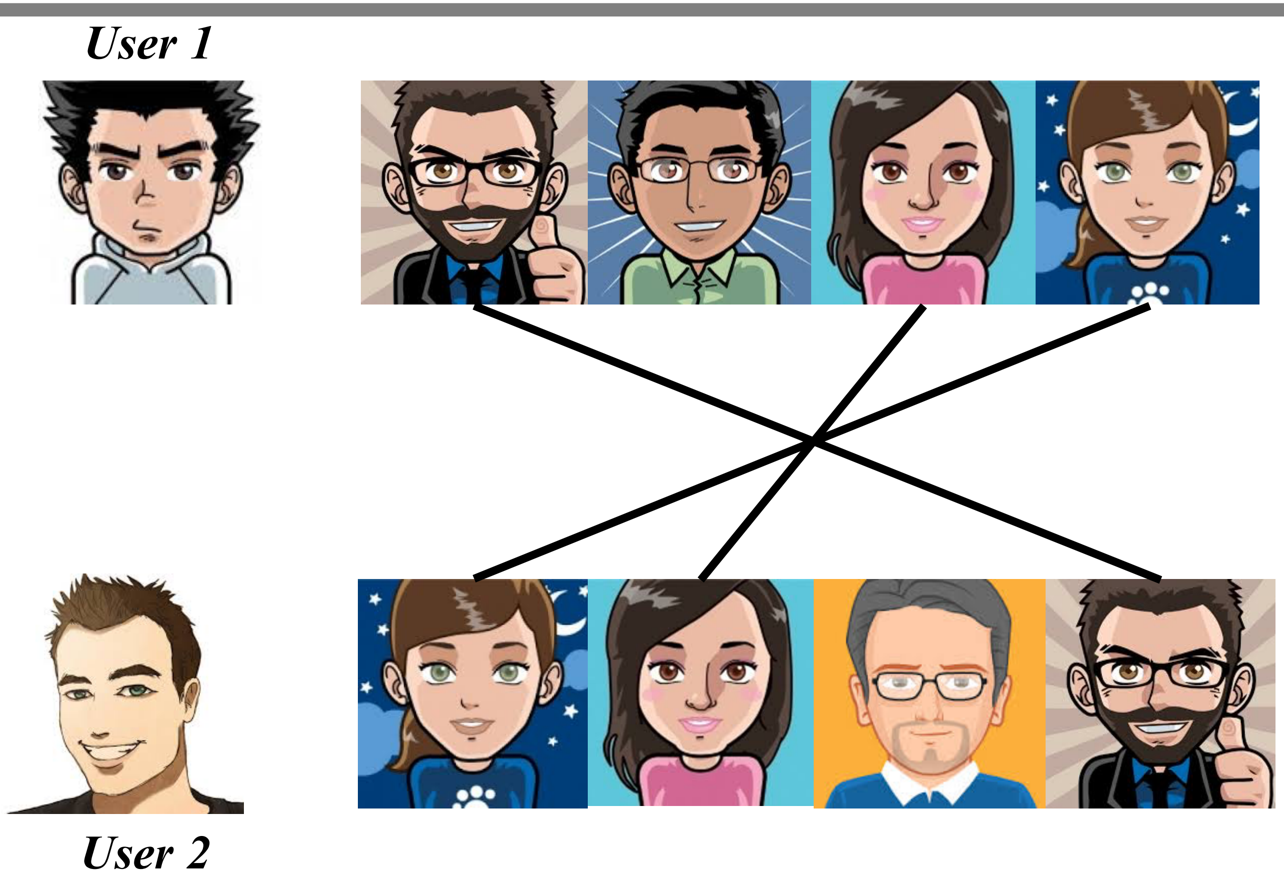
Example

- *Input*:
  - *R*
  - ***c* = 2**
- *Output*:
  - all set pairs with overlap size no smaller than ***c***

id	set
R_1	$\{e_1, e_2, e_3\}$
R_2	$\{e_1, e_3, e_4, e_7\}$
R_3	$\{e_1, e_3, e_5, e_7\}$
R_4	$\{e_2, e_4, e_5, e_6\}$
R_5	$\{e_2, e_4, e_5, e_6, e_8, e_9, e_{10}, e_{11}\}$
R_6	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}\}$
R_7	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}, e_{19}\}$

$|R_1 \cap R_3| = 2, \ |R_1 \cap R_2| = 2, \ |R_2 \cap R_3| = 3, \ |R_4 \cap R_5| = 4, \ |R_6 \cap R_7| = 8$

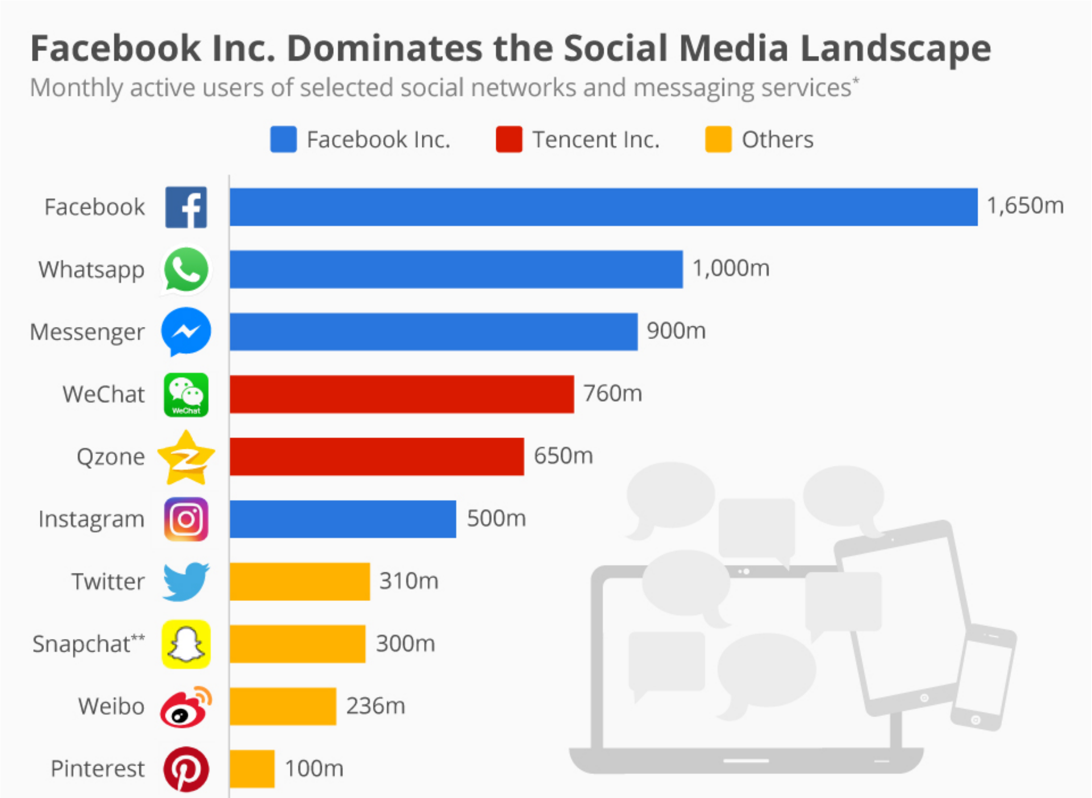
Application - Friend Recommendation

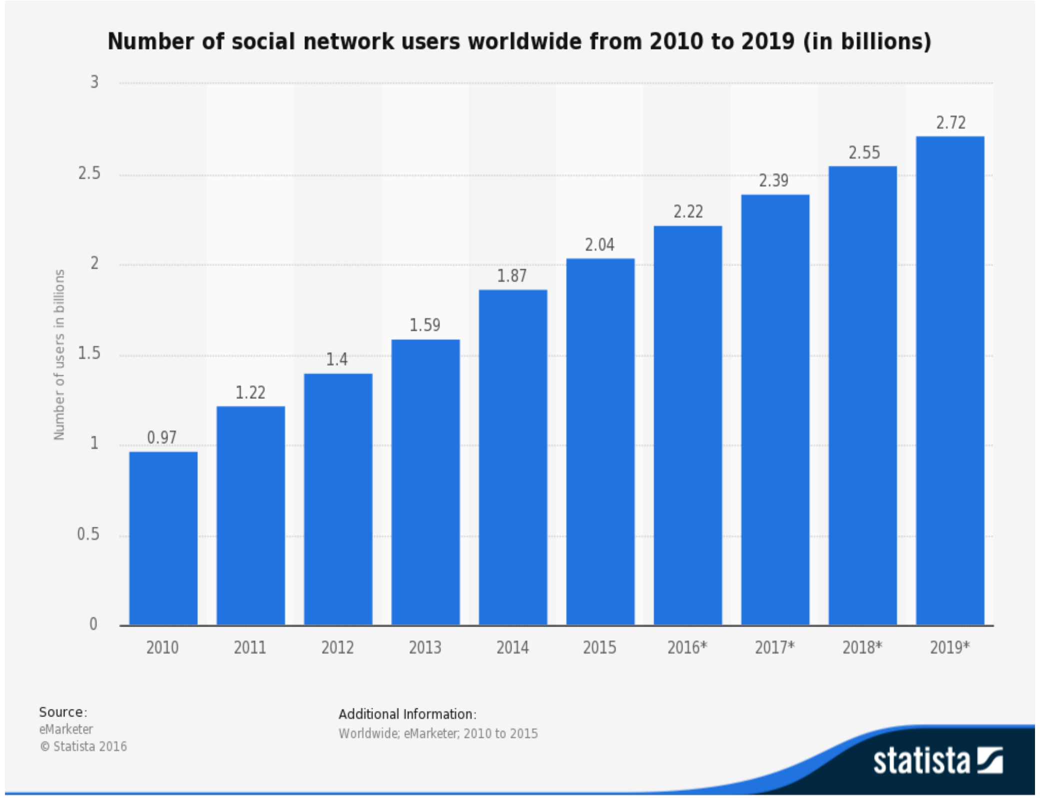


More Applications

- Data Management
  - Data Integration and Cleaning
  - Keyword Subscription
- Data Mining
  - Frequent Pattern Mining
  - Recommendation
- Computer Vision
  - Scene Reconstruction
- Machine Learning
  - Large Entries Retrieval in Matrix Productions
  - Non-negative Matrix Factorization
  - Singular Value Decomposition

Challenge


Facebook Inc. Dominates the Social Media Landscape
Monthly active users of selected social networks and messaging services*
Facebook Inc.
Tencent Inc.
Others
Facebook
1,650m
Whatsapp
1,000m
Messenger
900m
WeChat
760m
Qzone
650m
Instagram
500m
Twitter
310m
Snapchat**
300m
Weibo
236m
Pinterest
100m

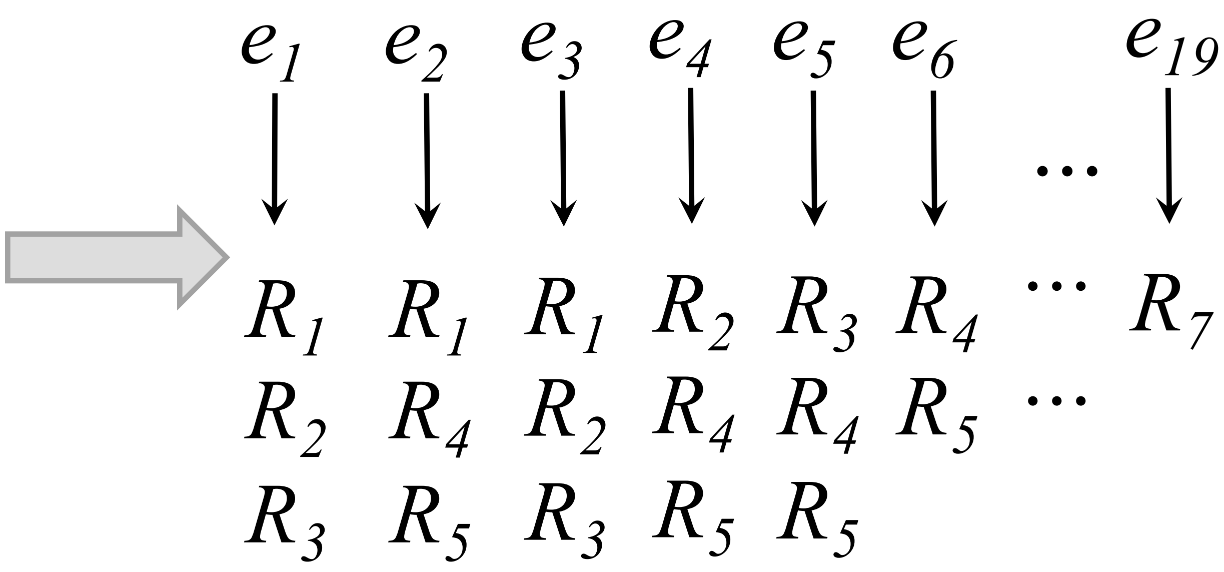

Number of social network users worldwide from 2010 to 2019 (in billions)
Number of users in billions
3
2.5
2
1.5
1
0.5
0
0.97
1.22
1.4
1.59
1.87
2.04
2.22
2.39
2.55
2.72
2010
2011
2012
2013
2014
2015
2016*
2017*
2018*
2019*
Source:
eMarketer
© Statista 2016
Additional Information:
Worldwide; eMarketer; 2010 to 2015
statista

Solution?

Naïve Method: ScanCount

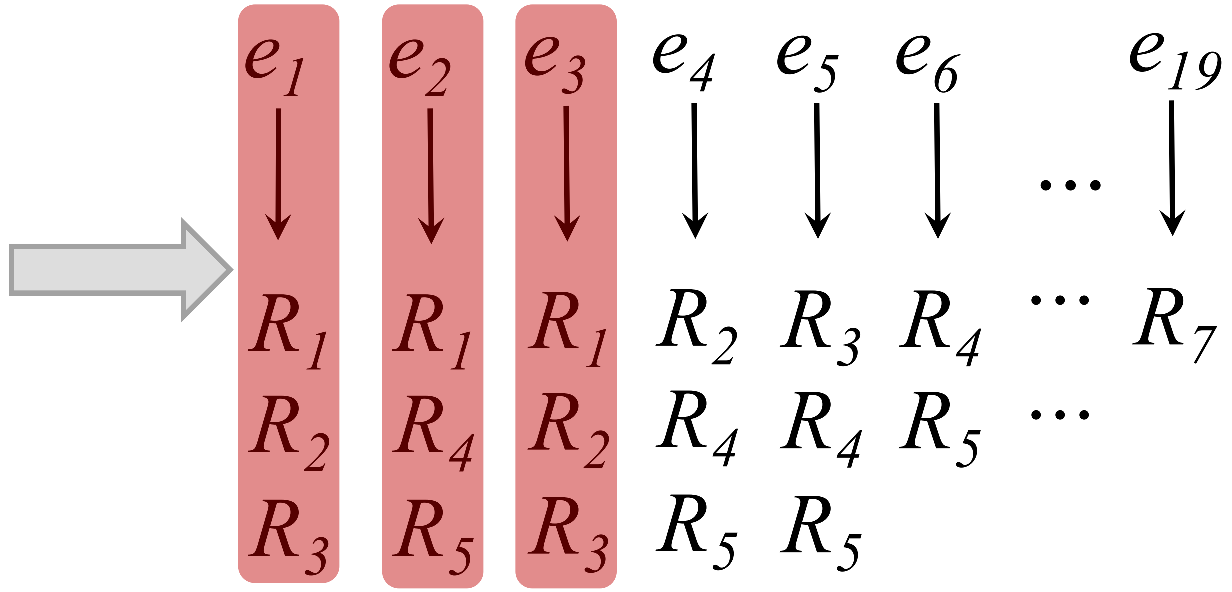
id	set
R_1	$\{e_1, e_2, e_3\}$
R_2	$\{e_1, e_3, e_4, e_7\}$
R_3	$\{e_1, e_3, e_5, e_7\}$
R_4	$\{e_2, e_4, e_5, e_6\}$
R_5	$\{e_2, e_4, e_5, e_6, e_8, e_9, e_{10}, e_{11}\}$
R_6	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}\}$
R_7	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}, e_{19}\}$

Step 1: build an inverted index



Naïve Method: ScanCount

id	set
R_1	$\{e_1, e_2, e_3\}$
R_2	$\{e_1, e_3, e_4, e_7\}$
R_3	$\{e_1, e_3, e_5, e_7\}$
R_4	$\{e_2, e_4, e_5, e_6\}$
R_5	$\{e_2, e_4, e_5, e_6, e_8, e_9, e_{10}, e_{11}\}$
R_6	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}\}$
R_7	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}, e_{19}\}$



Step 1: build an inverted index

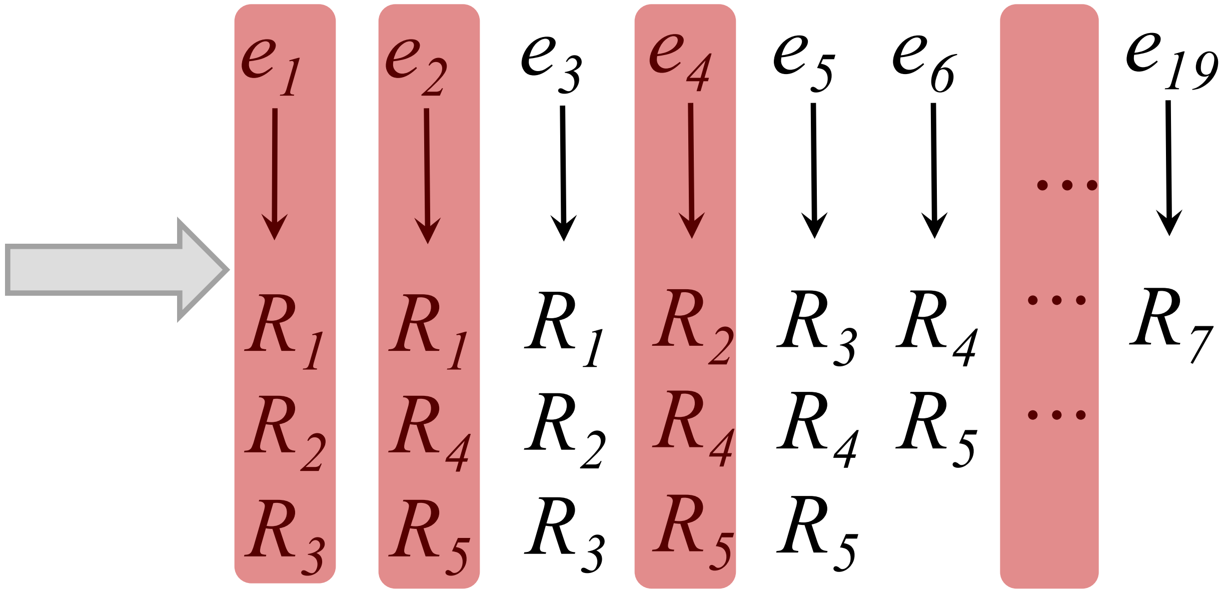
Step 2: scan each set and count

Count: R_2: 2 ✓ R_3: 2 ✓ R_4: 1 ⊘ R_5: 1 ⊘

Results: (R_1, R_2) (R_1, R_3)

Naïve Method: ScanCount

id	set
R_1	$\{e_1, e_2, e_3\}$
R_2	$\{e_1, e_3, e_4, e_7\}$
R_3	$\{e_1, e_3, e_5, e_7\}$
R_4	$\{e_2, e_4, e_5, e_6\}$
R_5	$\{e_2, e_4, e_5, e_6, e_8, e_9, e_{10}, e_{11}\}$
R_6	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}\}$
R_7	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}, e_{19}\}$



Step 1: build an inverted index

Step 2: scan each set and count

Count: R_3: 2 ✓ R_4: 2 ⊘ R_5: 1 ⊘

Results: (R_2, R_3)

Analysis

How many times is this list scanned?

$$e \longrightarrow R_1\, R_2\, R_3 \;\ldots\ldots\; R_m$$

is scanned m times, each time takes O(m); *in total* O(m^2)

Analysis

Some very frequent elements yield excessive long inverted lists

the long inverted lists are scanned many times.

$$e \longrightarrow R_1 \, R_2 \, R_3 \ \ldots\ldots \ R_m$$

for m=1 million, ScanCount takes ~1 trillion operations

Naïve Method: Subset Enumeration

threshold $c = 2$

id	set
R_1	$\{e_1, e_2, e_3\}$
R_2	$\{e_1, e_3, e_4, e_7\}$
R_3	$\{e_1, e_3, e_5, e_7\}$
R_4	$\{e_2, e_4, e_5, e_6\}$
R_5	$\{e_2, e_4, e_5, e_6, e_8, e_9, e_{10}, e_{11}\}$
R_6	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}\}$
R_7	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}, e_{19}\}$

R_1 → e_1e_2 e_1e_3 e_2e_3

R_2 → e_1e_3 e_1e_4 e_1e_7 e_3e_4 e_3e_7 e_4e_7

⋮

(R_1, R_2)

Step 1. for each set, enumerate all subsets of size ***c*** (*c*-subset for short)

Step 2. output all set pairs sharing a common ***c***-subset

Analysis

how many c-subsets are generated from each set?

$$R = \{e_1\ e_2\ e_3\ \ldots\ldots\ e_m\}$$

$\binom{m}{c}$ *c-subsets.*

Analysis

some very large sets yield a huge number of c-subsets.

$$R = \{e_1\ e_2\ e_3\ \ldots\ldots\ e_m\}$$

for m=1000 and c=3, it generates 166 million!

DossJoin: Combination of two methods

Small Sets

R_1	$\{e_1, e_2, e_3\}$
R_2	$\{e_1, e_3, e_4, e_7\}$
R_3	$\{e_1, e_3, e_5, e_7\}$
R_4	$\{e_2, e_4, e_5, e_6\}$

$\bowtie$

R_1	$\{e_1, e_2, e_3\}$
R_2	$\{e_1, e_3, e_4, e_7\}$
R_3	$\{e_1, e_3, e_5, e_7\}$
R_4	$\{e_2, e_4, e_5, e_6\}$

Naïve Algorithm 1 – Subset Enumeration

size boundary: x

Large Sets

R_5	$\{e_2, e_4, e_5, e_6, e_8, e_9, e_{10}, e_{11}\}$
R_6	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}\}$
R_7	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}, e_{19}\}$

$\bowtie$

R_1	$\{e_1, e_2, e_3\}$
R_2	$\{e_1, e_3, e_4, e_7\}$
R_3	$\{e_1, e_3, e_5, e_7\}$
R_4	$\{e_2, e_4, e_5, e_6\}$
R_5	$\{e_2, e_4, e_5, e_6, e_8, e_9, e_{10}, e_{11}\}$
R_6	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}\}$
R_7	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}, e_{19}\}$

Naïve Algorithm 2 – ScanCount

DossJoin: Combination of two methods

Small Sets

R_1	$\{e_1, e_2, e_3\}$
R_2	$\{e_1, e_3, e_4, e_7\}$
R_3	$\{e_1, e_3, e_5, e_7\}$
R_4	$\{e_2, e_4, e_5, e_6\}$

bounded by x

size boundary: x

Large Sets

R_5	$\{e_2, e_4, e_5, e_6, e_8, e_9, e_{10}, e_{11}\}$
R_6	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}\}$
R_7	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}, e_{19}\}$

bounded by $\frac{n}{x}$ Why?

1. each large set has at least x elements
2. there are n elements in total

Time Complexity Analysis for Large Sets

size boundary: x -

Large Sets

R_5	$\{e_2, e_4, e_5, e_6, e_8, e_9, e_{10}, e_{11}\}$
R_6	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}\}$
R_7	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}, e_{19}\}$

$\bowtie$

R_1	$\{e_1, e_2, e_3\}$
R_2	$\{e_1, e_3, e_4, e_7\}$
R_3	$\{e_1, e_3, e_5, e_7\}$
R_4	$\{e_2, e_4, e_5, e_6\}$
R_5	$\{e_2, e_4, e_5, e_6, e_8, e_9, e_{10}, e_{11}\}$
R_6	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}\}$
R_7	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}, e_{19}\}$

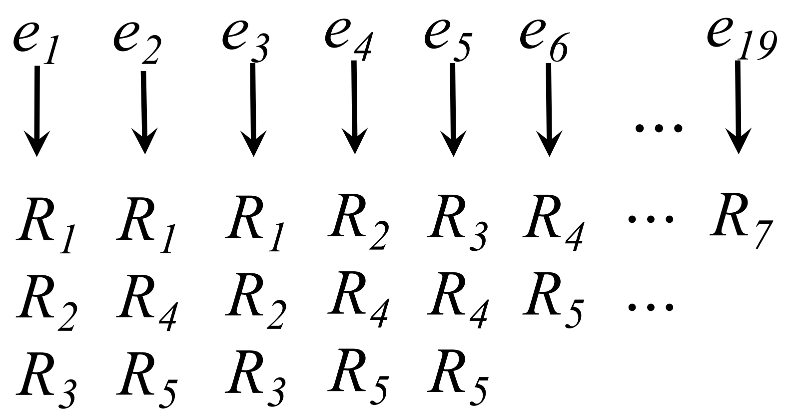
Naïve Algorithm 2 – ScanCount

Time Complexity Analysis for Large Sets

size boundary: x

Large Sets

R_5	$\{e_2, e_4, e_5, e_6, e_8, e_9, e_{10}, e_{11}\}$
R_6	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}\}$
R_7	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}, e_{19}\}$



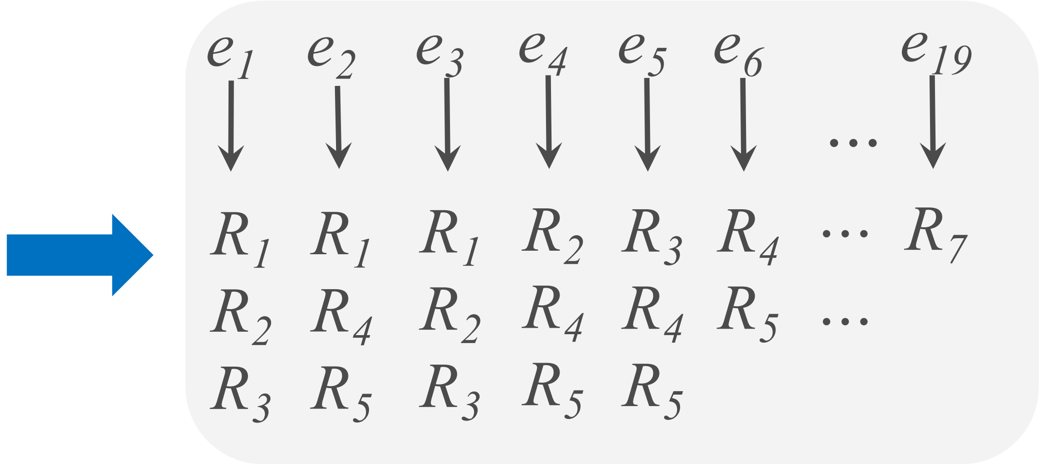
inverted index on all sets

Time Complexity Analysis for Large Sets

size boundary: x

Large Sets

R_5	$\{e_2, e_4, e_5, e_6, e_8, e_9, e_{10}, e_{11}\}$
R_6	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}\}$
R_7	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}, e_{19}\}$



each set takes O(n) time

Why?

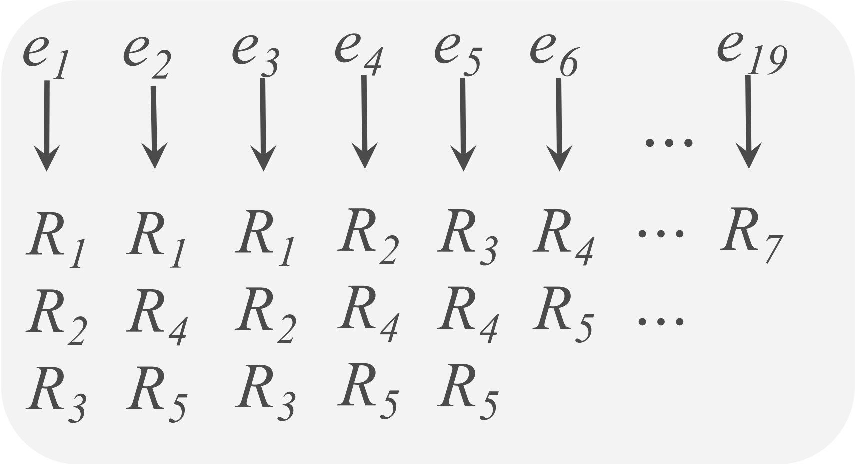
Time Complexity Analysis for Large Sets

size boundary: x

Large Sets

R_5	$\{e_2, e_4, e_5, e_6, e_8, e_9, e_{10}, e_{11}\}$
R_6	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}\}$
R_7	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}, e_{19}\}$

at most $\frac{n}{x}$ large sets as discussed



each set takes $O(n)$ time

Time Complexity Analysis for Large Sets

size boundary: x

Large Sets

R_5	$\{e_2, e_4, e_5, e_6, e_8, e_9, e_{10}, e_{11}\}$
R_6	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}\}$
R_7	$\{e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}, e_{19}\}$

at most $\frac{n}{x}$ *large sets as discussed*

e_1	e_2	e_3	e_4	e_5	e_6	…	e_{19}
R_1	R_1	R_1	R_2	R_3	R_4	…	R_7
R_2	R_4	R_2	R_4	R_4	R_5	…	
R_3	R_5	R_3	R_5	R_5			

each set takes O(n) time

Time Complexity: $O(\frac{n^2}{x})$

Time Complexity Analysis for Small Sets

Step 1: enumerate c-subsets, takes O(total # of c-subsets)

R_1	$\{e_1, e_2, e_3\}$	$\rightarrow \leq \binom{\lvert R_1 \rvert}{c}$
R_2	$\{e_1, e_3, e_4, e_7\}$	
R_3	$\{e_1, e_3, e_5, e_7\}$	$\cdots$
R_4	$\{e_2, e_4, e_5, e_6\}$	$\rightarrow \leq \binom{\lvert R_4 \rvert}{c}$

Time Complexity Analysis for Small Sets

Step 1: enumerate c-subsets, takes O(total # of c-subsets)

R_1	$\{e_1, e_2, e_3\}$	$\longrightarrow \leq \binom{\lvert R_1\rvert}{c} \leq \lvert R_1\rvert^c$
R_2	$\{e_1, e_3, e_4, e_7\}$	
R_3	$\{e_1, e_3, e_5, e_7\}$	$\ldots \quad \ldots$
R_4	$\{e_2, e_4, e_5, e_6\}$	$\longrightarrow \leq \binom{\lvert R_4\rvert}{c} \leq \lvert R_4\rvert^c$

Time Complexity Analysis for Small Sets

Step 1: enumerate c-subsets, takes O(total # of c-subsets)

R_1	$\{e_1, e_2, e_3\}$
R_2	$\{e_1, e_3, e_4, e_7\}$
R_3	$\{e_1, e_3, e_5, e_7\}$
R_4	$\{e_2, e_4, e_5, e_6\}$

$R_1 \rightarrow \le \binom{|R_1|}{c} \le |R_1|^c \le |R_1| * x^{c-1}$

… …

$R_4 \rightarrow \le \binom{|R_4|}{c} \le |R_4|^c \le |R_4| * x^{c-1}$

$\Sigma|\text{Rsmall}| \le n$

Time Complexity Analysis for Small Sets

Step 1: enumerate c-subsets, takes O(total # of c-subsets)

R_1	$\{e_1, e_2, e_3\}$
R_2	$\{e_1, e_3, e_4, e_7\}$
R_3	$\{e_1, e_3, e_5, e_7\}$
R_4	$\{e_2, e_4, e_5, e_6\}$

$R_1 \longrightarrow \le \binom{|R_1|}{c} \le |R_1|^c \le |R_1| * x^{c-1}$

$\cdots \quad \cdots$

$R_4 \longrightarrow \le \binom{|R_4|}{c} \le |R_4|^c \le |R_4| * x^{c-1}$

$\Bigg\} \le nx^{c-1}$

$\Sigma|\text{Rsmall}| \le n$

total # of c-subsets $\le nx^{c-1}$

Time Complexity Analysis for Small Sets

Step 2: output all set pairs sharing a c-subset

R_1	$\{e_1, e_2, e_3\}$
R_2	$\{e_1, e_3, e_4, e_7\}$
R_3	$\{e_1, e_3, e_5, e_7\}$
R_4	$\{e_2, e_4, e_5, e_6\}$

build an inverted index for all c-subsets

for each inverted list, output all pairs of sets

	e_1e_2	e_1e_3	e_1e_4	e_1e_5	e_1e_7	e_2e_3	e_2e_4	e_2e_5	e_2e_6	e_3e_4	e_3e_5	e_3e_7	e_4e_5	e_4e_6	e_4e_7	e_5e_6	e_5e_7
R_1	■	■				■											
R_2		■	■		■					■		■			■		
R_3		■		■	■						■	■					■
R_4							■	■	■				■	■		■	

Time Complexity Analysis for Small Sets

Step 2: output all set pairs sharing a c-subset

	e_1e_2	e_1e_3	e_1e_4	e_1e_5	e_1e_7	e_2e_3	e_2e_4	e_2e_5	e_2e_6	e_3e_4	e_3e_5	e_3e_7	e_4e_5	e_4e_6	e_4e_7	e_5e_6	e_5e_7
R_1	■	■				■											
R_2		■	■		■					■		■			■		
R_3		■		■	■						■	■					■
R_4							■	■	■				■	■		■	

what's the total length of all inverted lists?

exactly the total # of c-subsets $\leq nx^{c-1}$

Time Complexity Analysis for Small Sets

Step 2: output all set pairs sharing a c-subset

	e_1e_2	e_1e_3	e_1e_4	e_1e_5	e_1e_7	e_2e_3	e_2e_4	e_2e_5	e_2e_6	e_3e_4	e_3e_5	e_3e_7	e_4e_5	e_4e_6	e_4e_7	e_5e_6	e_5e_7
R_1	■	■				■											
R_2		■	■		■					■		■			■		
R_3		■		■	■						■	■					■
R_4							■	■	■				■	■		■	

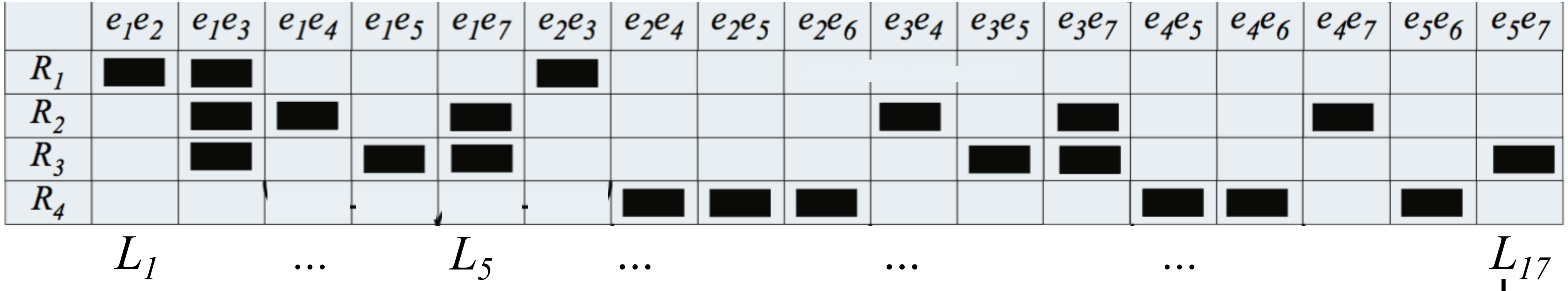
$\leq \sqrt{k}$

what's the maximum length of any inverted list?

$number\ of\ results = L^2 \leq k$ $\qquad$ $L \leq \sqrt{k}$

Time Complexity Analysis for Small Sets

Step 2: output all set pairs sharing a c-subset



$\leq |L_1| * |L_1|$ $\qquad$ $\leq |L_5| * |L_5|$ $\qquad$ maximum length of any inverted list $\qquad$ $\leq |L_{17}| * |L_{17}|$

$\leq |L_1| * \sqrt{k}$ $\qquad$ $\leq |L_5| * \sqrt{k}$ $\qquad$ total length of all inverted lists $\qquad$ $\leq |L_{17}| * \sqrt{k}$

$$\leq nx^{c-1}\sqrt{k}$$

Time Complexity Analysis

overall time complexity: $\mathcal{O}\left(\frac{n^2}{x} + x^{c-1} n\sqrt{k}\right)$

let $x = (n/\sqrt{k})^{1/c}$

the time complexity is $\mathcal{O}\left(n^{2-\frac{1}{c}} k^{\frac{1}{2c}}\right) = o(n^2) + O(k)$

why?

$case\ 1: k = o(n^2)$
$case\ 2: k = O(n^2)$

Two Practical Problems

- Needs to enumerate all c-subsets, whose number can be huge in practice.

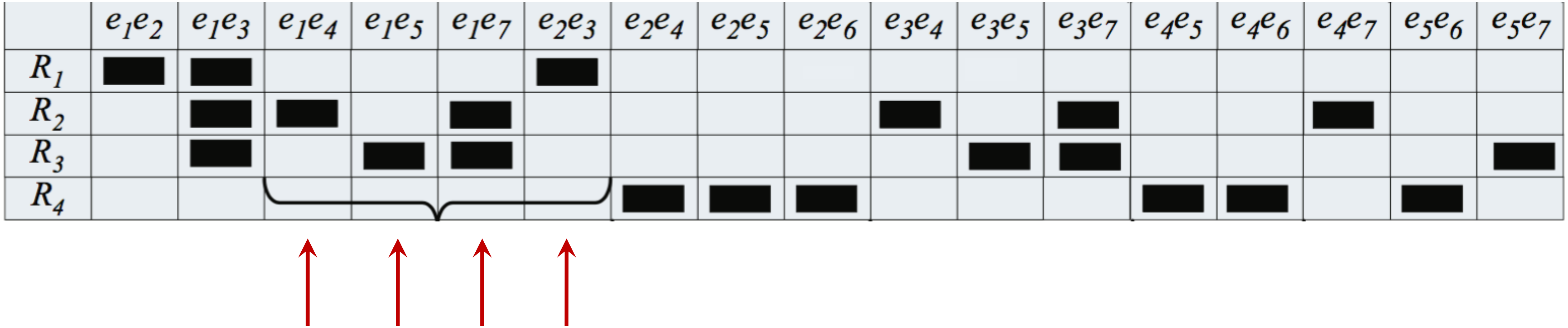
- The size boundary x set by the theoretical analysis is not practical, as it overestimates the cost for small sets a lot.

Skip Unnecessary c-subsets

	e_1e_2	e_1e_3	e_1e_4	e_1e_5	e_1e_7	e_2e_3	e_2e_4	e_2e_5	e_2e_6	e_3e_4	e_3e_5	e_3e_7	e_4e_5	e_4e_6	e_4e_7	e_5e_6	e_5e_7
R_1	■	■				■											
R_2		■	■		■					■		■			■		
R_3		■		■	■						■	■					■
R_4							■	■	■				■	■		■	
	↑		↑	↑		↑	↑	↑	↑	↑	↑		↑	↑	↑	↑	↑

Observation 1: ***Unique c-subsets*** cannot generate any result and we can skip them

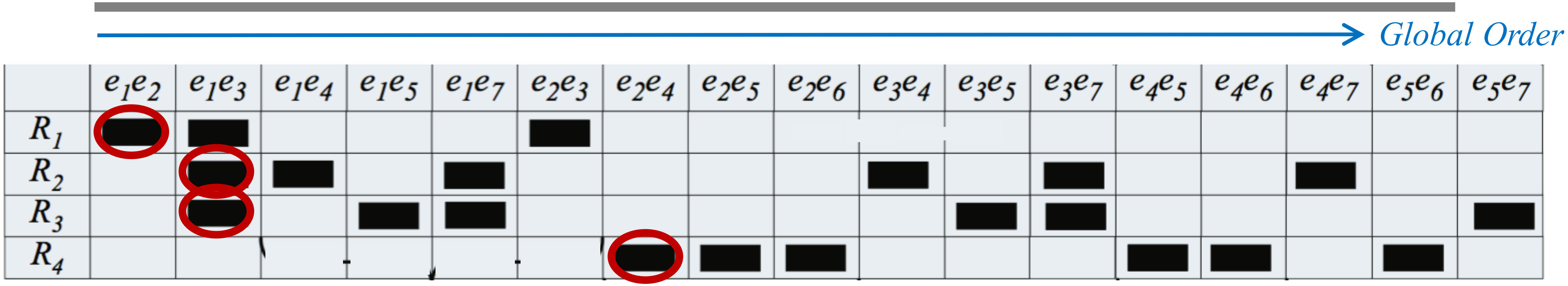
Skip Unnecessary c-subsets



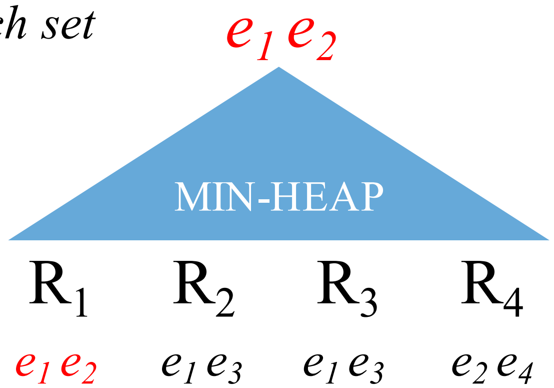
Observation 2: ***Redundant c-subsets*** only generate duplicate results and we can skip them

How to skip the unnecessary c-subsets?

Heap-based Method



min-subset: the smallest unvisited c-subset in each set



Goal: generate all the c-subsets in the global order, one by one

Heap-based Method

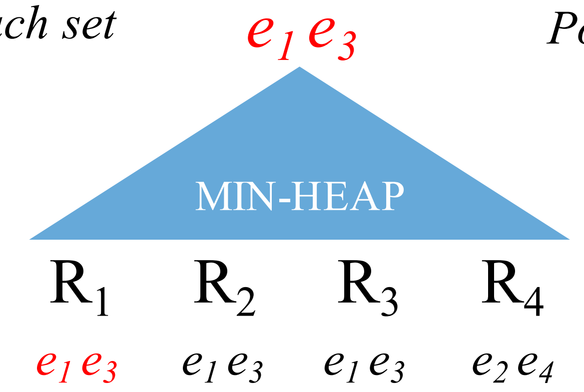
Global Order →

	e_1e_2	e_1e_3	e_1e_4	e_1e_5	e_1e_7	e_2e_3	e_2e_4	e_2e_5	e_2e_6	e_3e_4	e_3e_5	e_3e_7	e_4e_5	e_4e_6	e_4e_7	e_5e_6	e_5e_7
R_1	■	■				■											
R_2		■	■		■					■		■			■		
R_3		■		■	■						■	■					■
R_4							■	■	■				■	■		■	

min-subset: the smallest unvisited c-subset in each set

Pop & Push

$e_1 e_2 \leftarrow R_1$



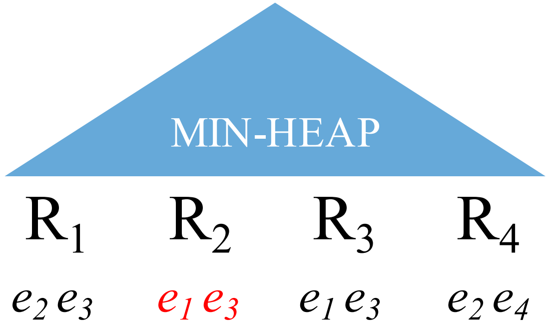
Heap-based Method

Global Order →

	e_1e_2	e_1e_3	e_1e_4	e_1e_5	e_1e_7	e_2e_3	e_2e_4	e_2e_5	e_2e_6	e_3e_4	e_3e_5	e_3e_7	e_4e_5	e_4e_6	e_4e_7	e_5e_6	e_5e_7
R_1	■	■				■ (circled)											
R_2		■ (circled)	■		■					■		■			■		
R_3		■ (circled)		■	■						■	■					■
R_4							■ (circled)	■	■				■	■		■	

(circle) *min-subset: the smallest unvisited c-subset in each set*

$e_1\, e_3$



Pop & Push

$e_1\, e_2 \leftarrow R_1$

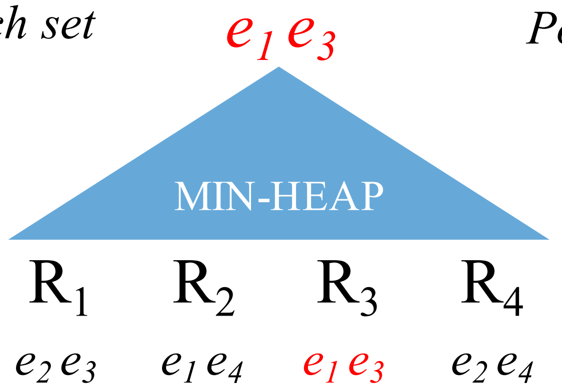
$e_1\, e_3 \leftarrow R_1$

Heap-based Method

Global Order →

	e_1e_2	e_1e_3	e_1e_4	e_1e_5	e_1e_7	e_2e_3	e_2e_4	e_2e_5	e_2e_6	e_3e_4	e_3e_5	e_3e_7	e_4e_5	e_4e_6	e_4e_7	e_5e_6	e_5e_7
R_1	■	■				■											
R_2		■	■		■					■		■			■		
R_3		■		■	■						■	■					■
R_4							■	■	■				■	■		■	

min-subset: the smallest unvisited c-subset in each set



Pop & Push

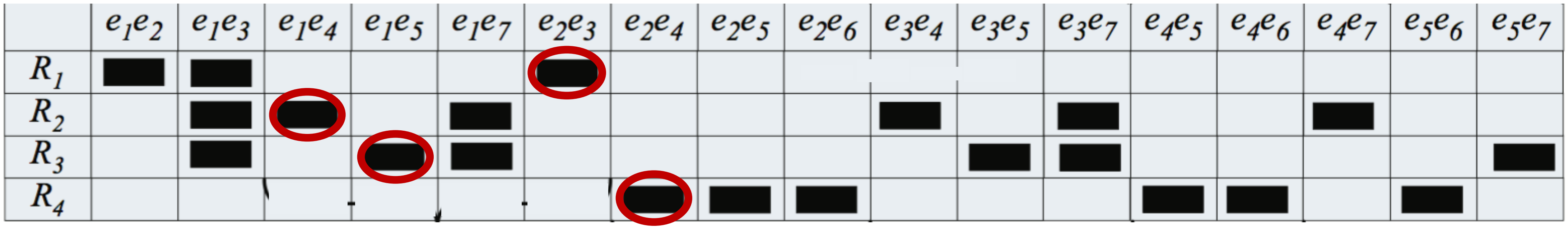
$e_1 e_2 \leftarrow R_1$

$e_1 e_3 \leftarrow R_1$

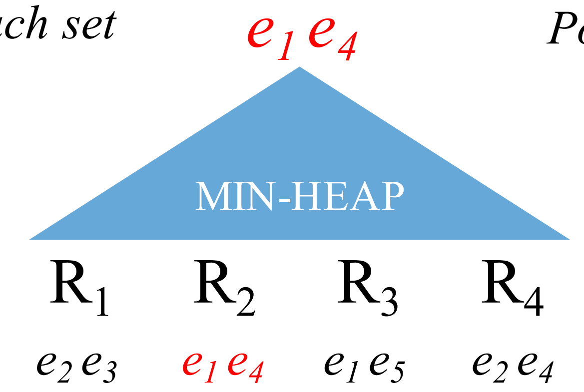
$e_1 e_3 \leftarrow R_2$

Heap-based Method

Global Order



min-subset: the smallest unvisited c-subset in each set



Pop & Push

$e_1 e_2 \leftarrow R_1$

$e_1 e_3 \leftarrow R_1$

$e_1 e_3 \leftarrow R_2$

$e_1 e_3 \leftarrow R_3$

Heap-based Method

Global Order →

	e_1e_2	e_1e_3	e_1e_4	e_1e_5	e_1e_7	e_2e_3	e_2e_4	e_2e_5	e_2e_6	e_3e_4	e_3e_5	e_3e_7	e_4e_5	e_4e_6	e_4e_7	e_5e_6	e_5e_7
R_1	■	■				■											
R_2		■	■		■					■		■			■		
R_3		■		■	■						■	■					■
R_4							■	■	■				■	■		■	

min-subset: the smallest unvisited c-subset in each set

$e_1 e_5$

R_1	R_2	R_3	R_4
$e_2 e_3$	$e_1 e_7$	$e_1 e_5$	$e_2 e_4$

Pop & Push

$e_1 e_2 \leftarrow R_1$

$e_1 e_3 \leftarrow R_1$

$e_1 e_3 \leftarrow R_2$

$e_1 e_3 \leftarrow R_3$

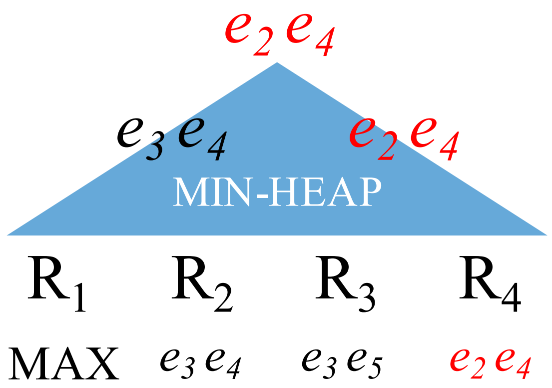
$e_1 e_4 \leftarrow R_2$

Skip the Unique c-subsets using heap

Heap-based Method

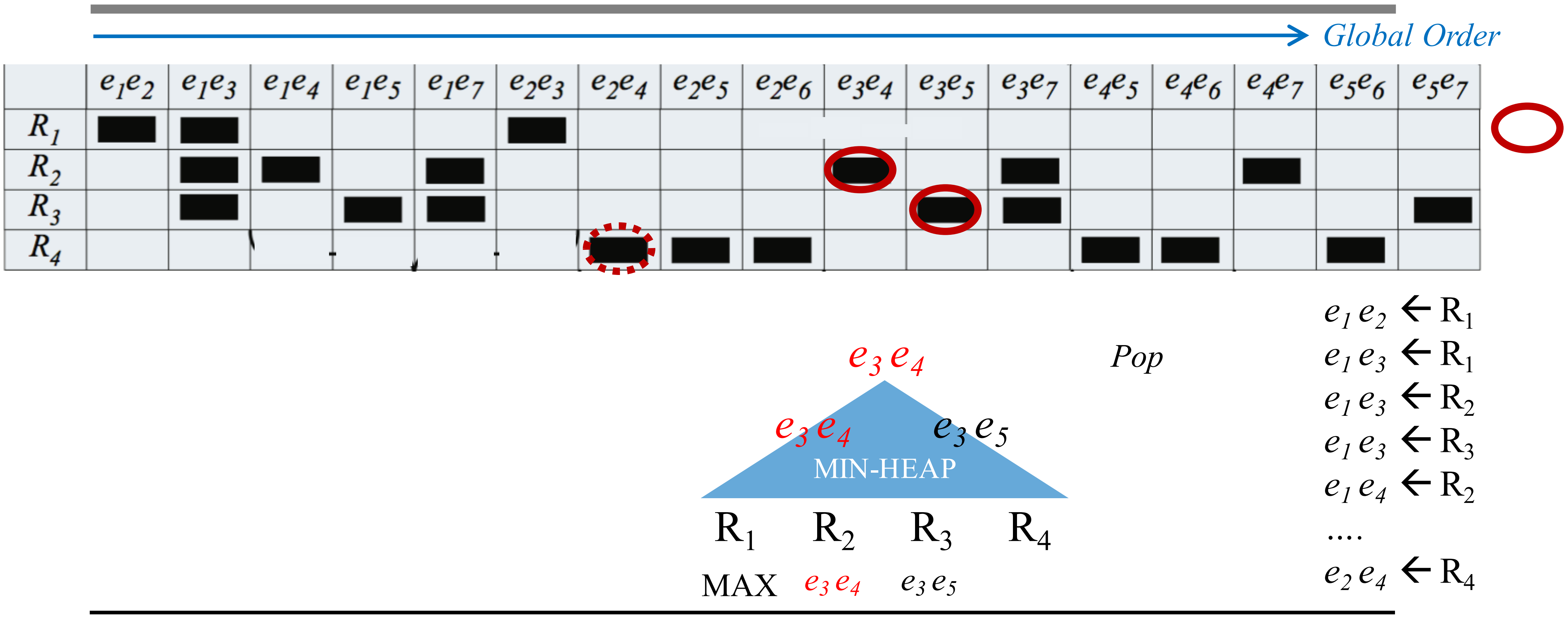
Global Order →

	e_1e_2	e_1e_3	e_1e_4	e_1e_5	e_1e_7	e_2e_3	e_2e_4	e_2e_5	e_2e_6	e_3e_4	e_3e_5	e_3e_7	e_4e_5	e_4e_6	e_4e_7	e_5e_6	e_5e_7
R_1	■	■				■											
R_2		■	■		■					■		■			■		
R_3		■		■	■						■	■					■
R_4							■	■	■				■	■		■	

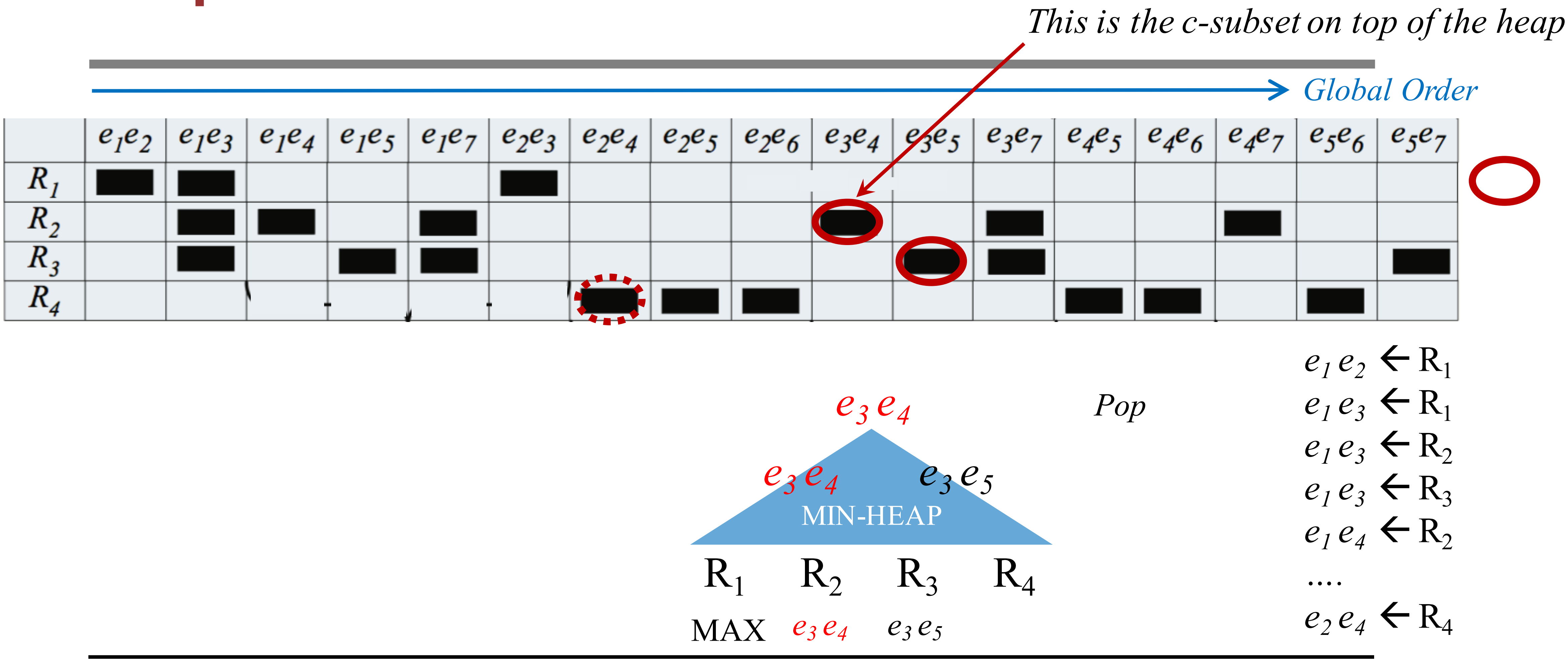


$e_1 e_2 \leftarrow R_1$
$e_1 e_3 \leftarrow R_1$
$e_1 e_3 \leftarrow R_2$
$e_1 e_3 \leftarrow R_3$
$e_1 e_4 \leftarrow R_2$
....

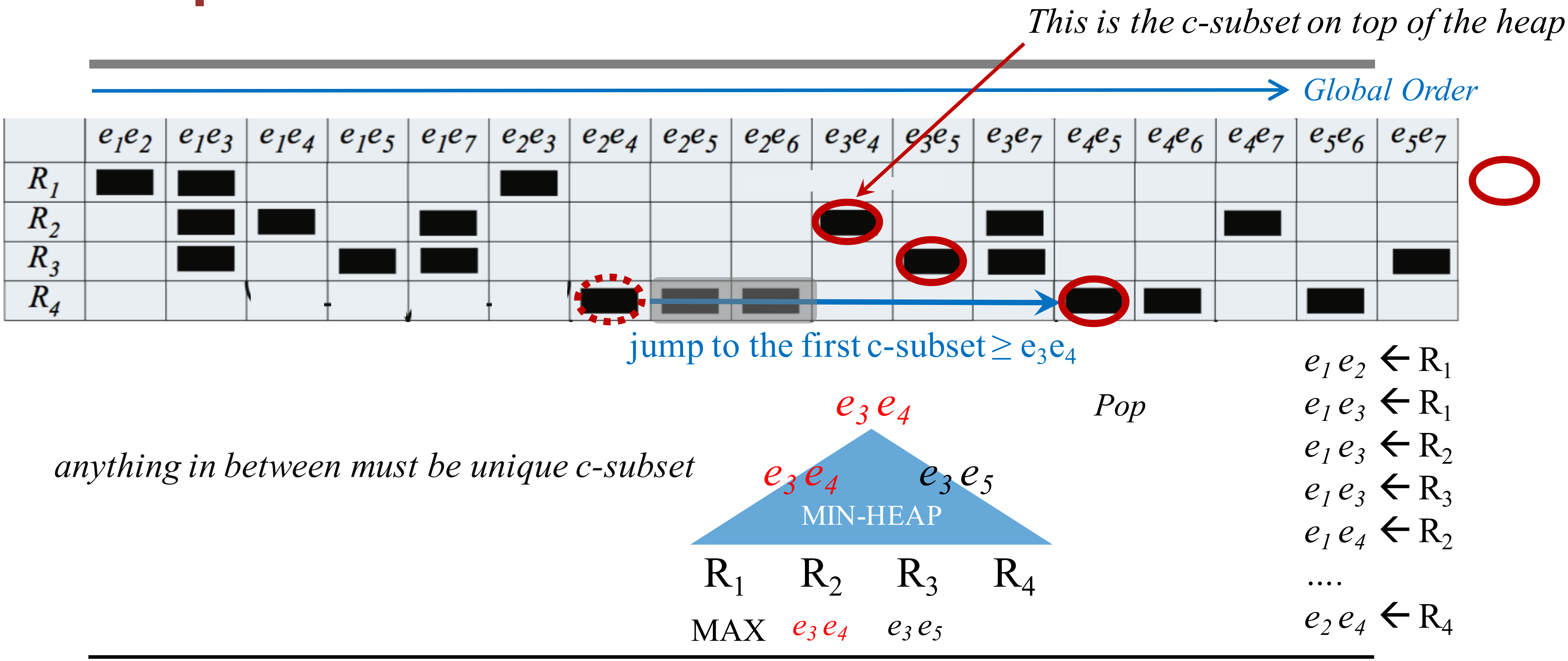
Heap-based Method



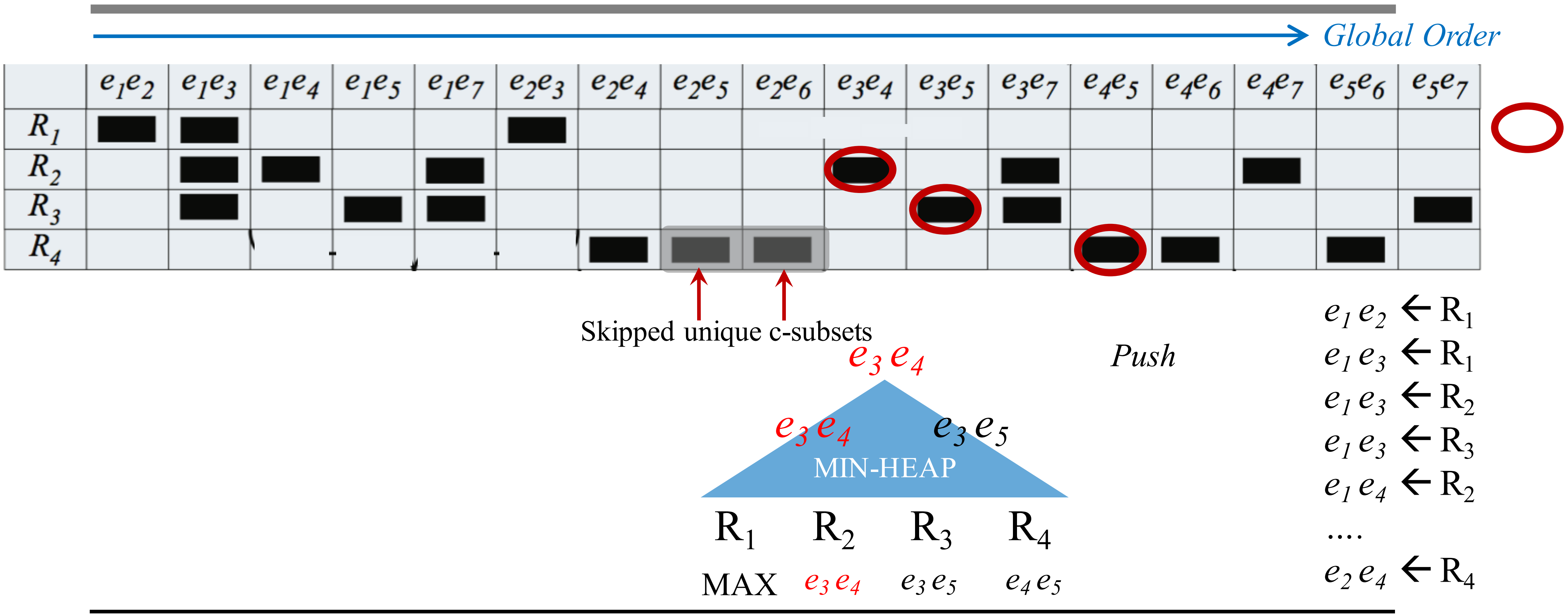
Heap-based Method



Heap-based Method

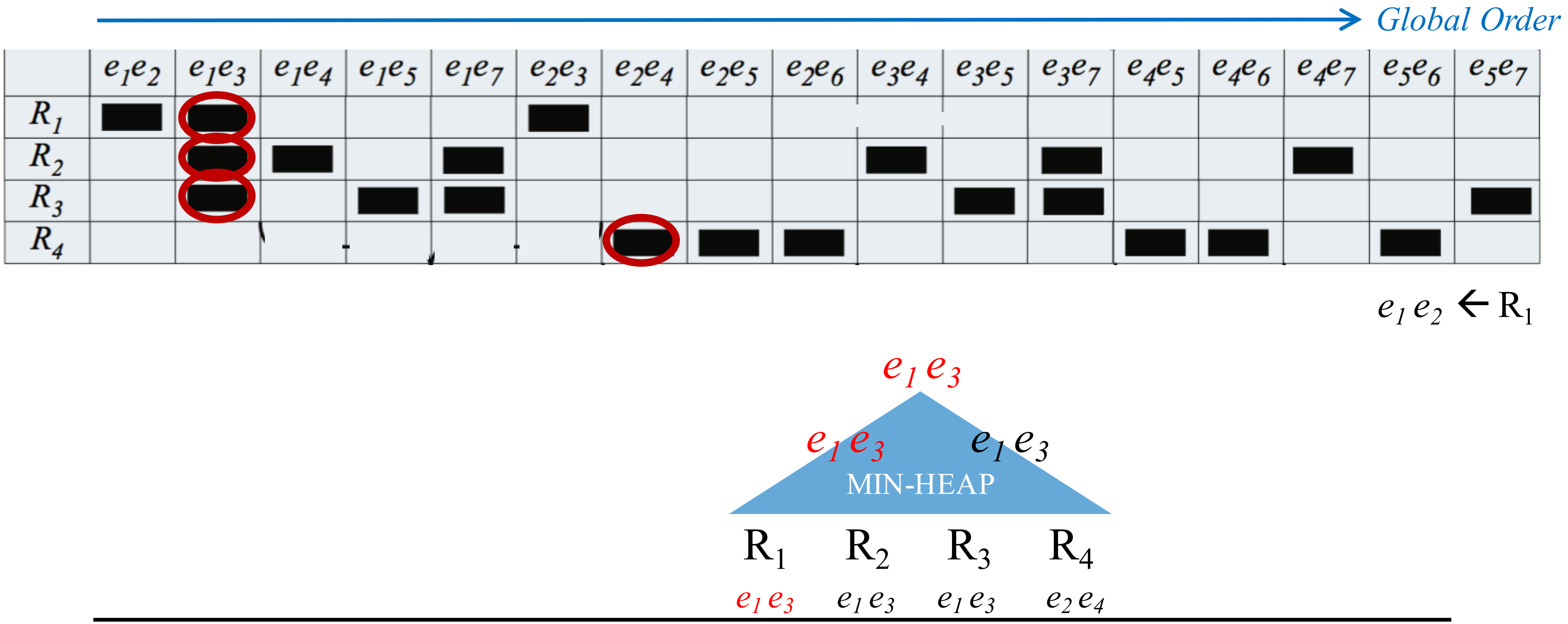


Heap-based Method

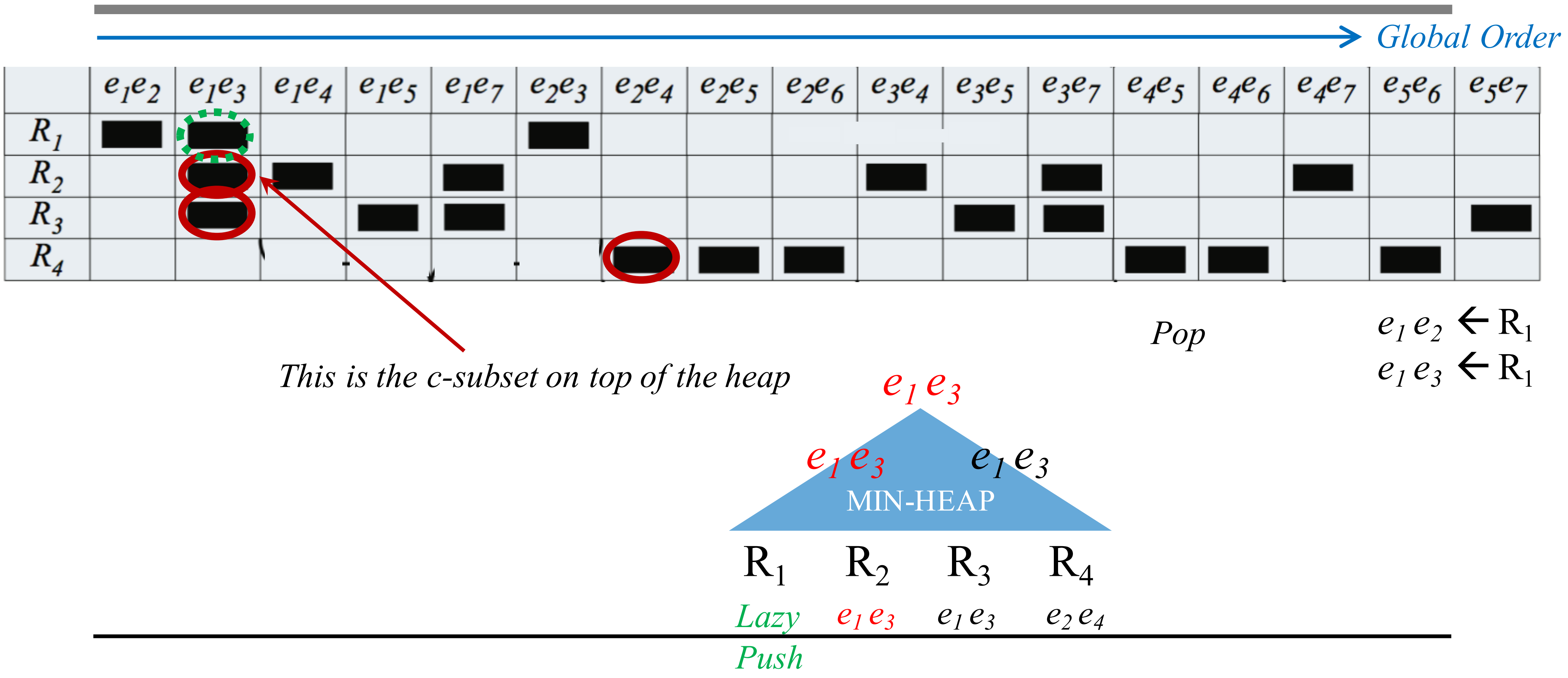


Skip the redundant c-subsets using heap

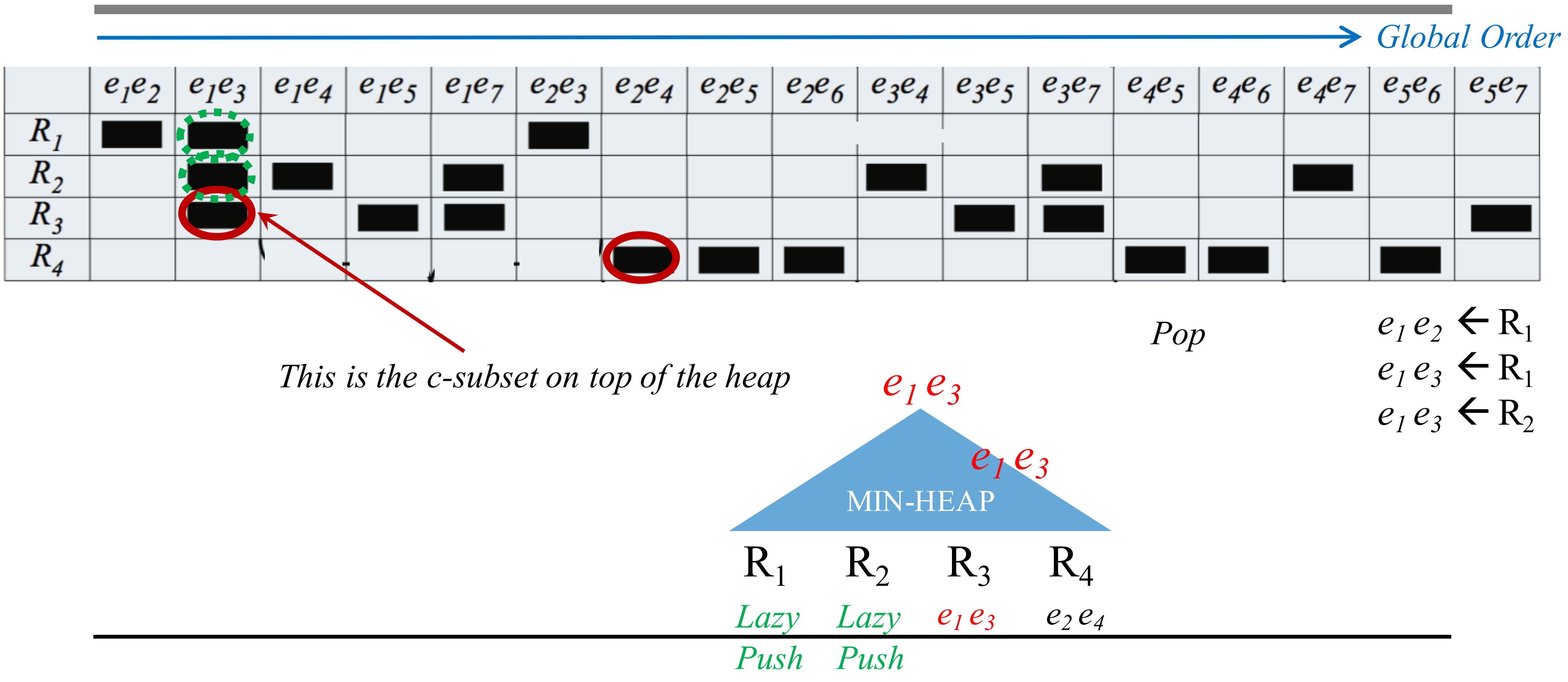
Heap-based Method



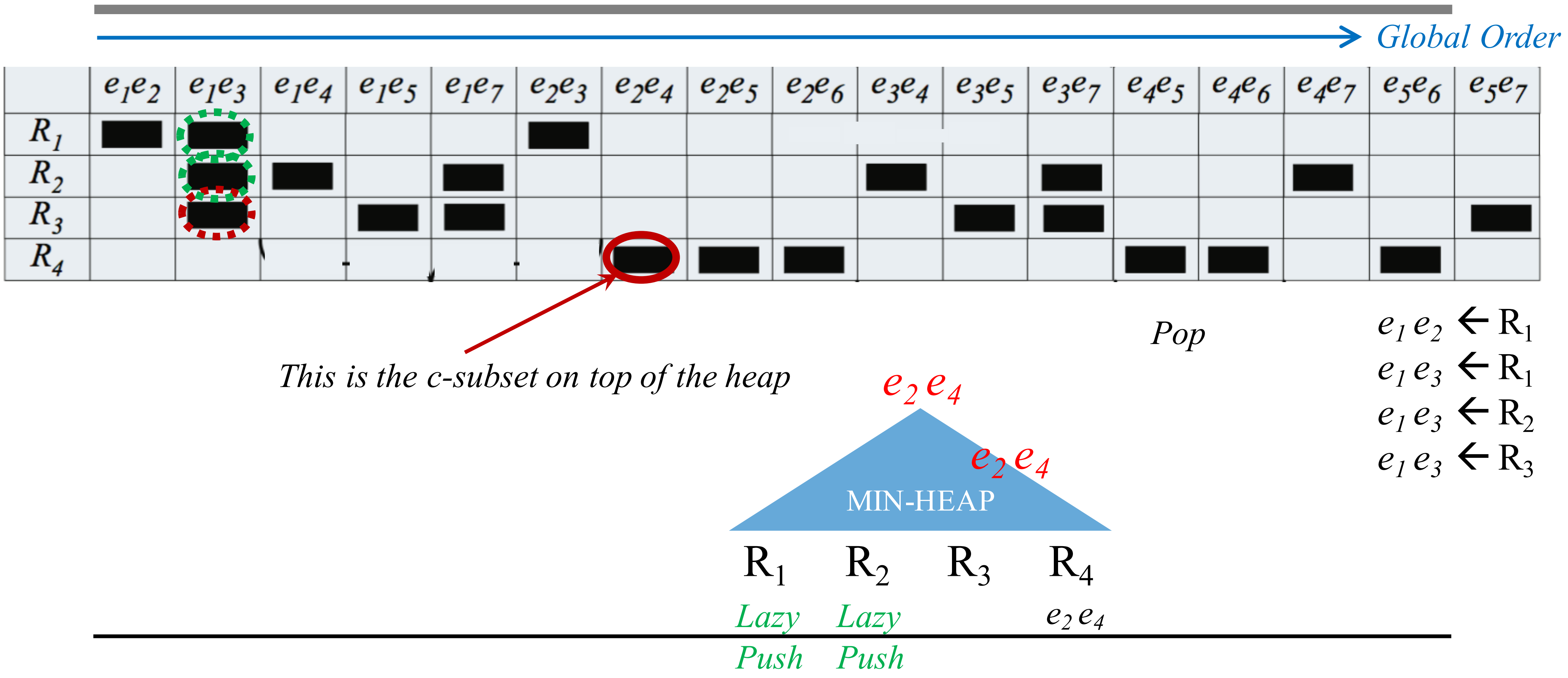
Heap-based Method



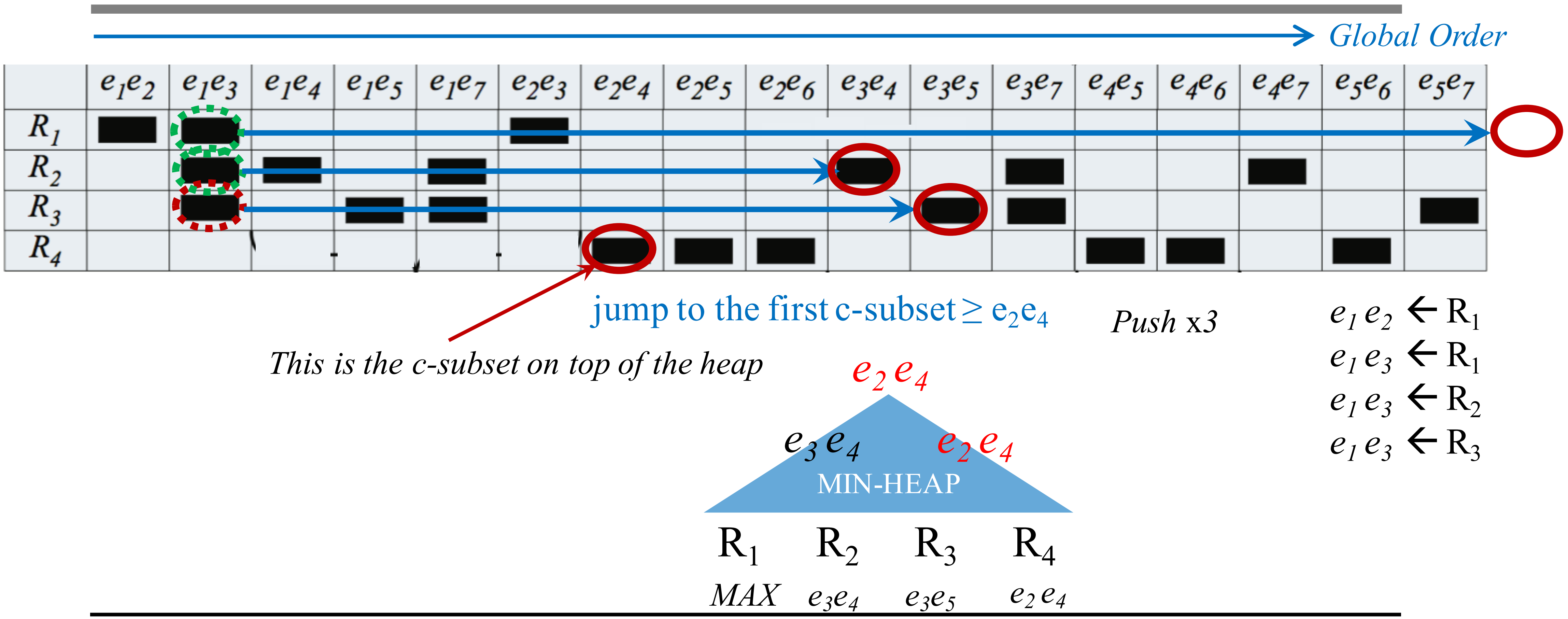
Heap-based Method



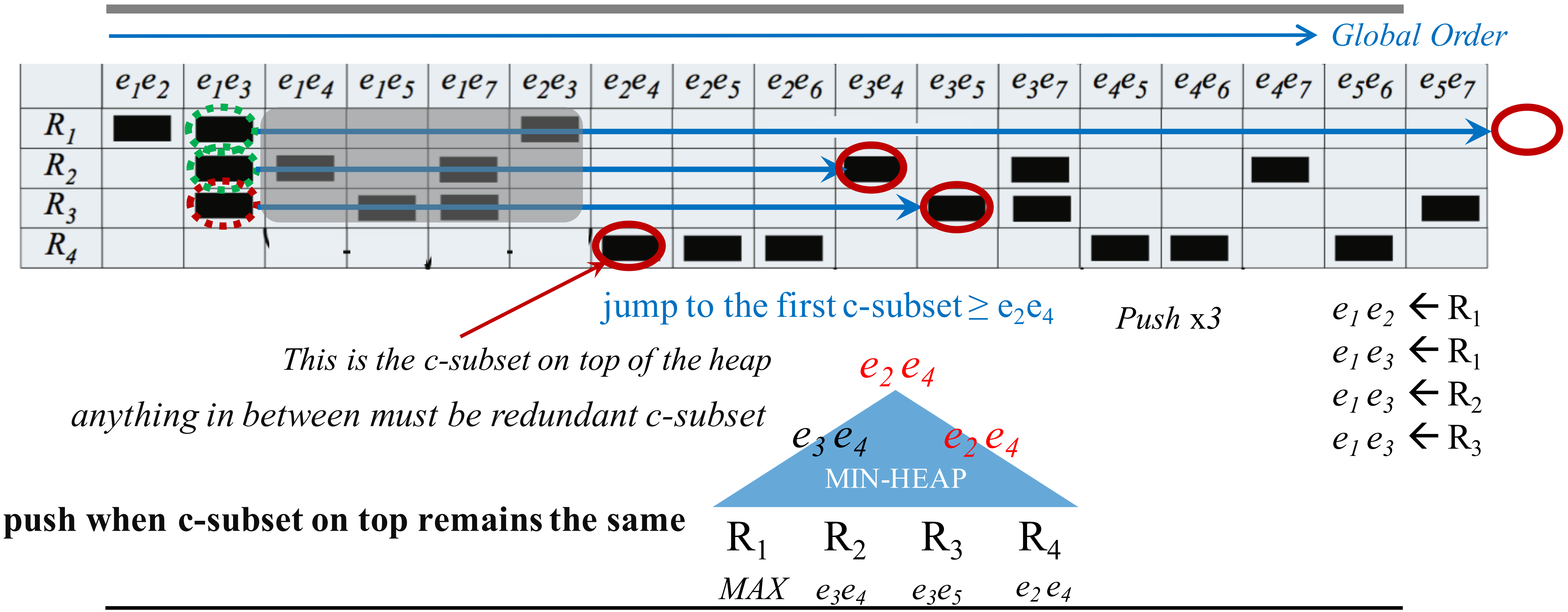
Heap-based Method



Heap-based Method



Heap-based Method



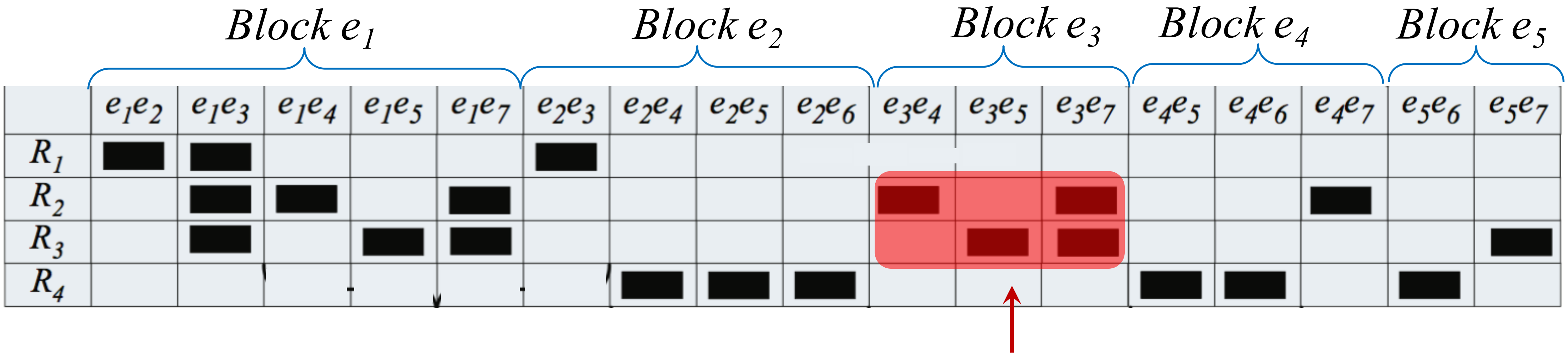
Lasy push when c-subset on top remains the same

How to reduce the heap-adjustment cost?

each heap adjustment takes log_2m time, where m is the number of small c-subsets

Heap Blocking

block by the minimum element in the c-subsets

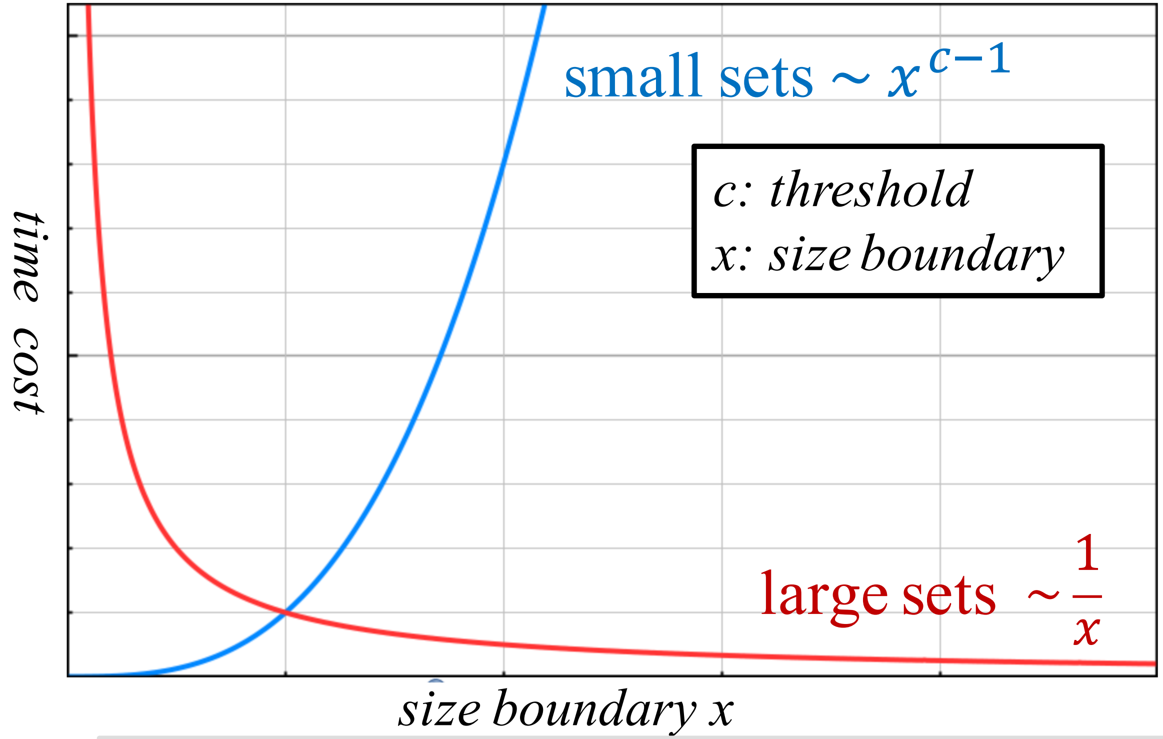


invoke the heap-based method in each block with threshold c -1.

the heap size will be much smaller than m

How to select a practical size boundary?

Size Boundary Selection



cost (slope) goes up smoothly first, and then rapidly

benefit (slope) goes down rapidly first, and then smoothly

Benefit: the reduction of time spend on large sets

Cost: the increase of time spend on small sets

increase x little by little, stop right after estimated benefit is less than estimated cost

Estimating large set processing cost

- For each large set, add up all its corresponding inverted list lengths
- Use the summation as the cost

Estimating small set processing cost

- Heap adjustment cost and binary search costs.
  - Sampling some blocks and run the heap-based method to get the estimation
- Result Generation Costs
  - R = {e_1 e_2 e_3 e_4} and S = {e_1 e_2 e_3 e_5}
  - The pair <R, S> is generated 3 times by our method (Why?)
  - ***Thus the cost is proportional to the # of c-subsets shared by small sets***
  - Sample a small number of set pairs
  - For each set pair, compute their overlap as P
  - The # of shared c-subsets between them is $\binom{P}{c}$

Conclusion

- $o(n^2) + O(k)$ -- Sub-quadratic Algorithm whenever possible
- Heap-based Methods for Small Sets
- Size-boundary Selection Method