Efficient Bulk Deletes in Relational Databases

A. Gärtner[1] A. Kemper[1] D. Kossmann[2] B. Zeller[1]

[1] Universität Passau
94030 Passau, Germany
<*lastname*>@db.fmi.uni-passau.de

[2] Technische Universität München
81667 München, Germany
kossmann@in.tum.de

Outline

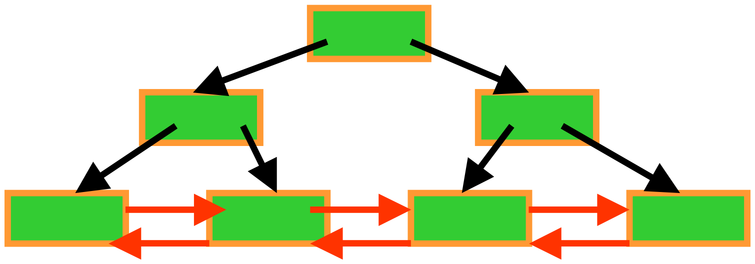
- **Motivation**
- **Related work**
- **Idea: traditional horizonzal vs. new vertical approach**
- **Implementation**
- **Concurrency control and reorganisation**
- **Benchmark results**
- **Conclusion and future work**

Motivation

- bulk deletes used in:
  - SAP R/3 → Archiving
  - data warehouses → window technique
- today: 500 MB, 3 indices, 15% deleted → 2 h 50 min
- partitioning → drop partition, but :
  - orthogonal deletes:
    - data is **partitioned** according to **orderdate**
    - data is **deleted** according to the **orderstatus** flag
  - semantic restrictions
    - „delete all orders with orderdate < 1995 **but only if the order is fully processed**“

Index Structure

- **B$^+$-Trees used for indexing**
- Leaf pages linked together



root

inner nodes

leaves

- Entries in leaves are sorted according to indexed values
- Record identified by unique Row Identifier (RID)
- RID contains physical address

RID:	001	005	017
	FileNr	BlockNr	SlotNr

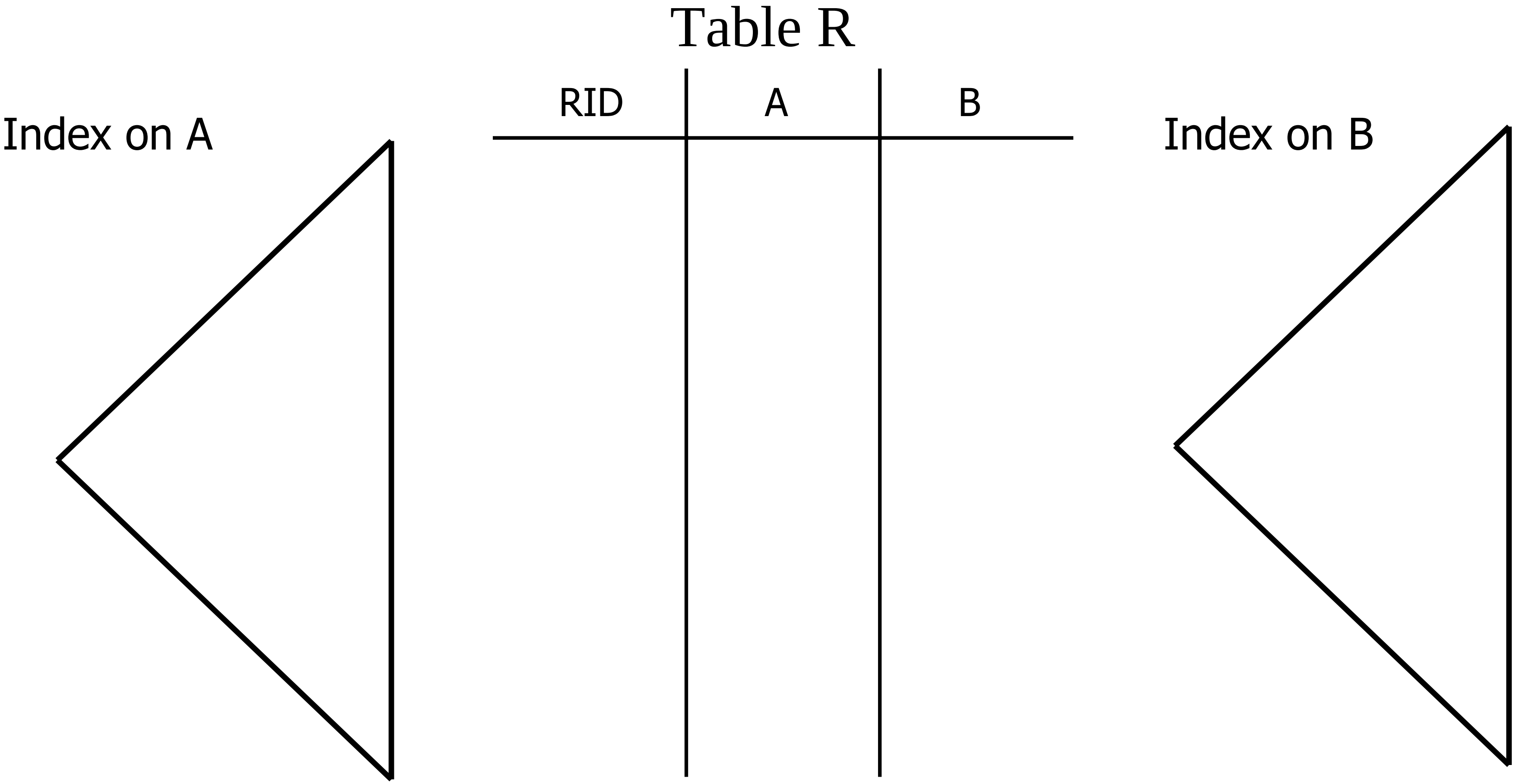
Related Work

- deletion in B$^+$-Trees
  - J.Jannink. **Implementing deletion in B$^+$-Trees**. ACM SIGMOD 1994
  - T.Johnson and D.Sasha. **Utilization of B-Trees with inserts, deletes and modifies**. ACM SIGMOD 1989.
- bulk loading
  - J.v.d.Bercken, B.Seeger, and P.Widmayer. **A generic approach to bulk loading multidimensional index structures**. VLDB 1997
  - J.Wiener and J.Naughton. **OODB bulk loading revisited: The partitioned list approach.** VLDB 1995.

Related Work

- Concurrent transactions
  - C.Zou and B.Salzberg. **On-line reorganisation of sparsely-populated B$^+$- Trees**. ACM SIGMOD 1996.
  - C.Mohan and F.Levine. **ARIES/IM: An efficient and high concurrency index management method using write-ahead logging.** ACM SIGMOD 1992
- Pointer join processing
  - E.Shekita and M.Cary. **A performance evaluation of pointer based joins**. ACM SIGMOD 1990.
  - R.Braumandl, J.Claussen, A.Kemper, and D.Kossmann. **Functional join processing**. VLDB 1998.

Traditional Horizontal Approach (Record-at-a-time approach)



Traditional Horizontal Approach (Record-at-a-time approach)

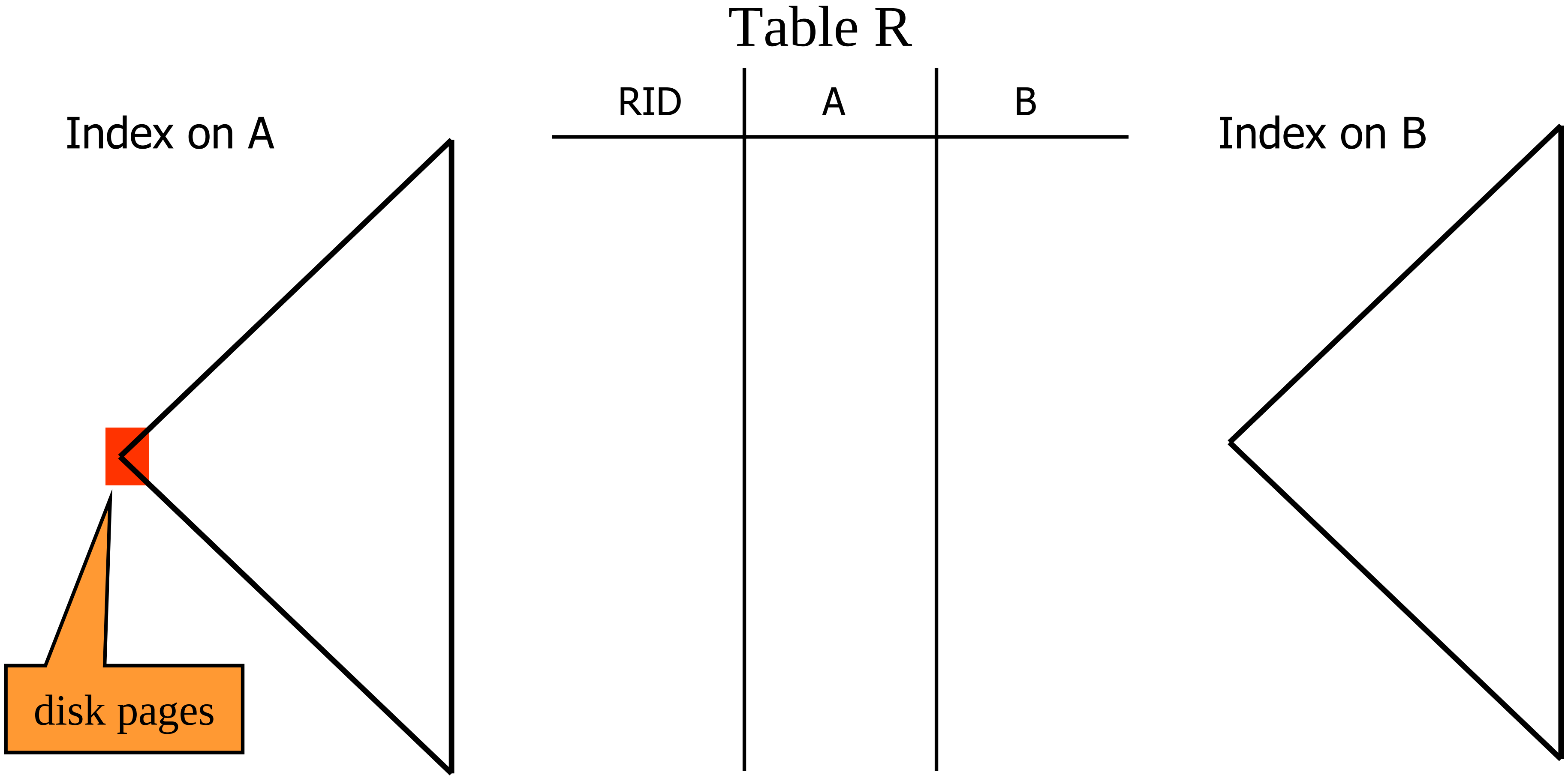


Table R

RID	A	B

Index on A

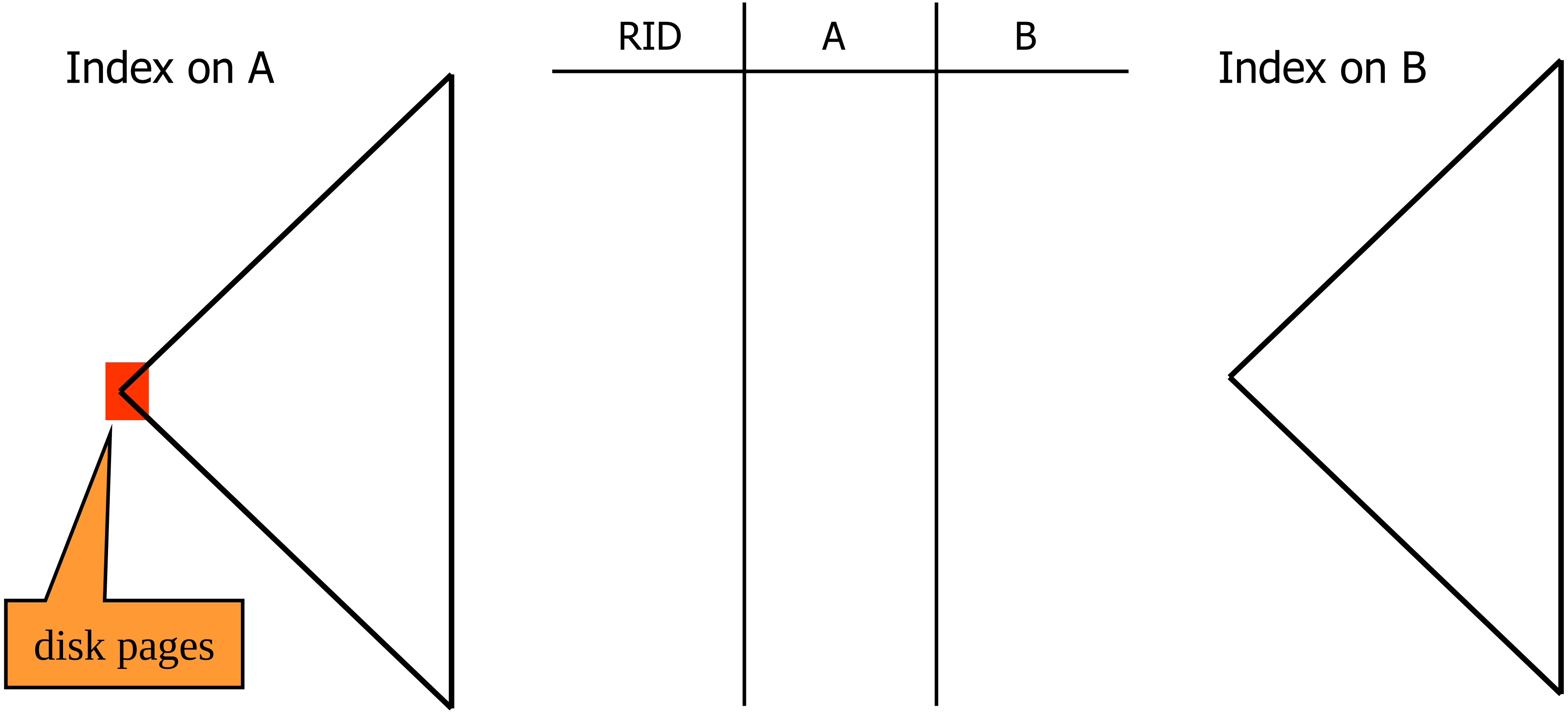
Index on B

disk pages

Traditional Horizontal Approach (Record-at-a-time approach)

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)

Table R



RID	A	B

Index on A

Index on B

disk pages

Traditional Horizontal Approach (Record-at-a-time approach)

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)

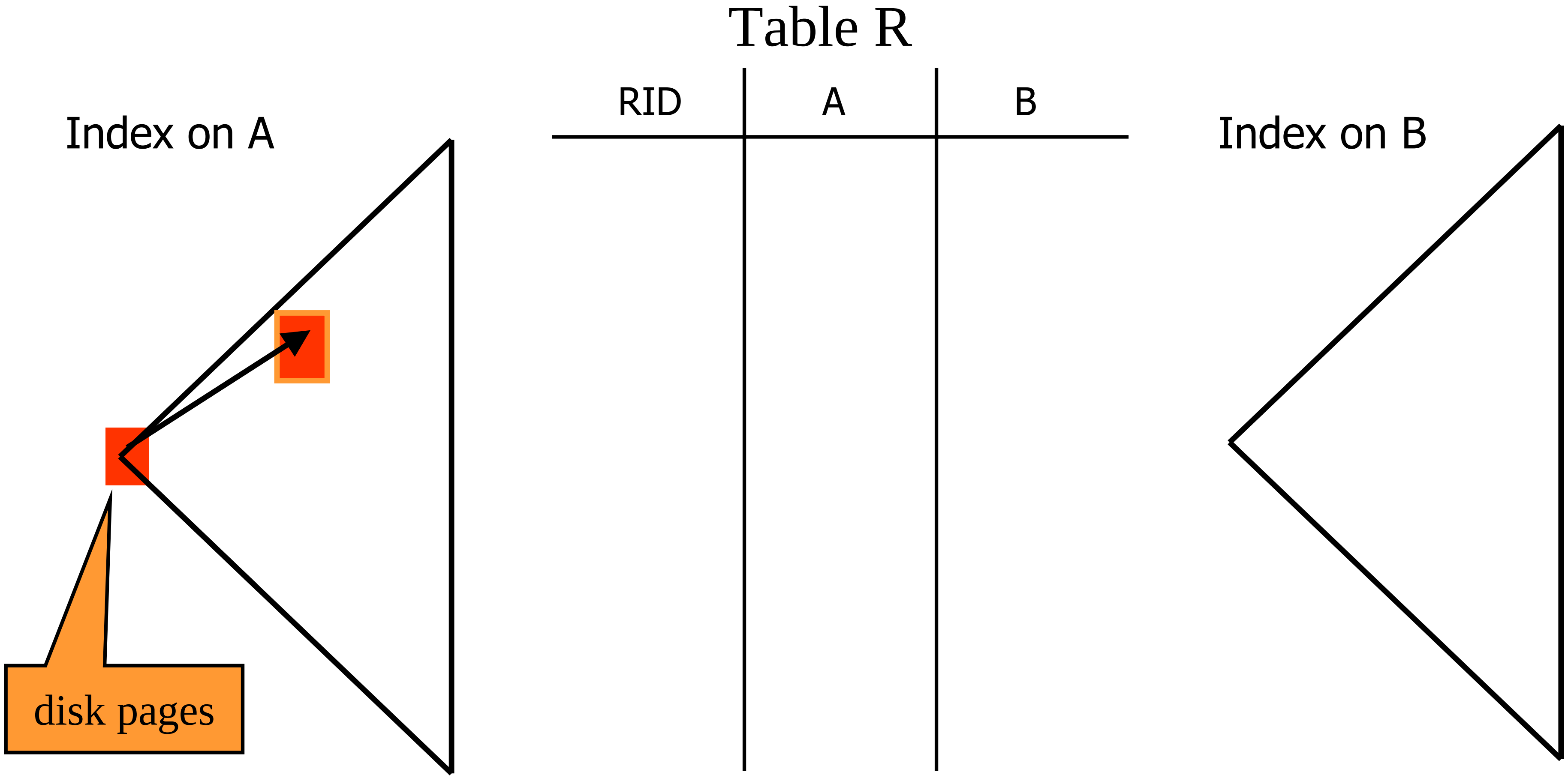


Table R

RID A B

Index on A

Index on B

disk pages

Traditional Horizontal Approach (Record-at-a-time approach)

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)

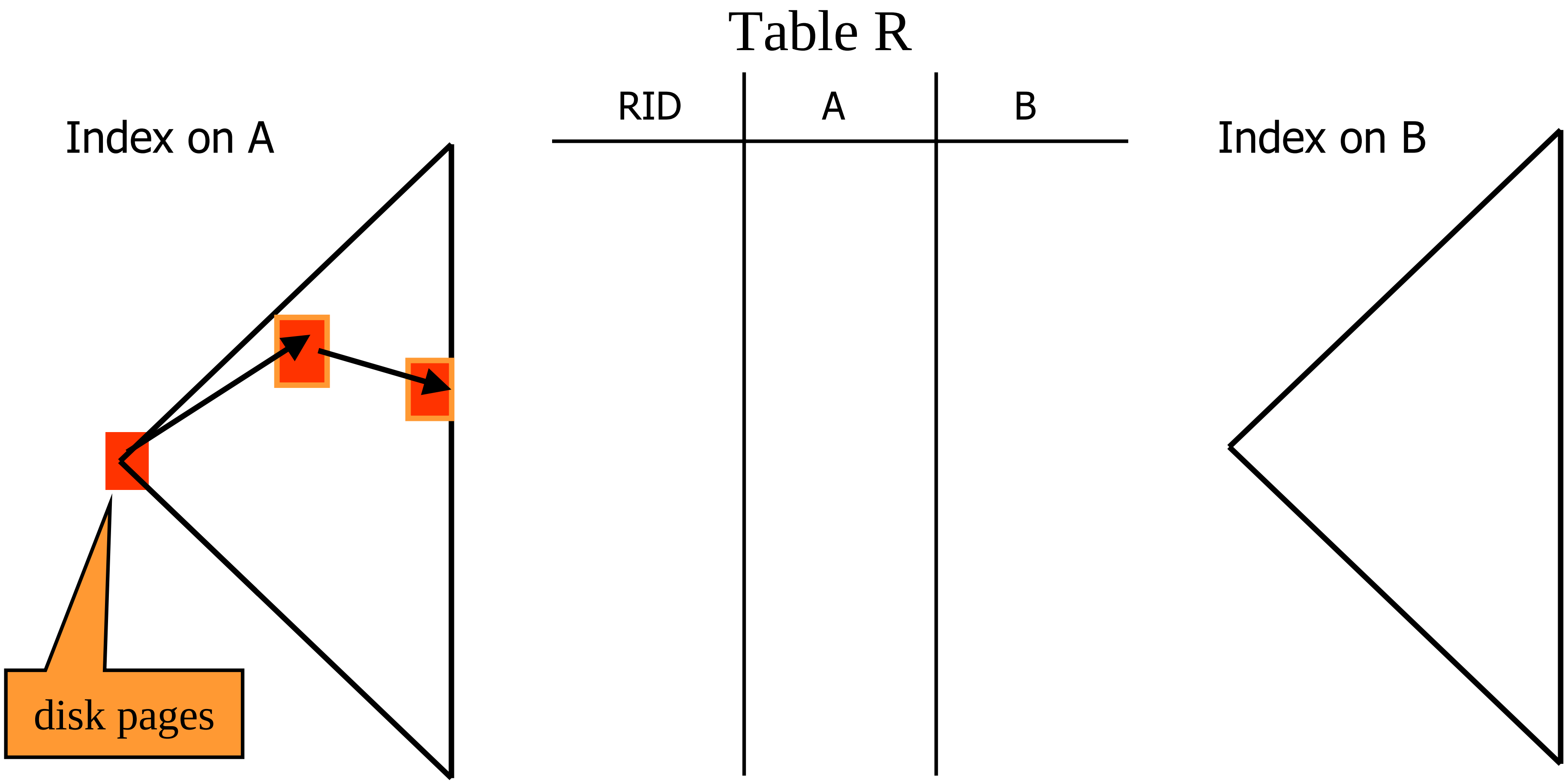


Table R

RID	A	B

Index on A

Index on B

disk pages

Traditional Horizontal Approach (Record-at-a-time approach)

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)

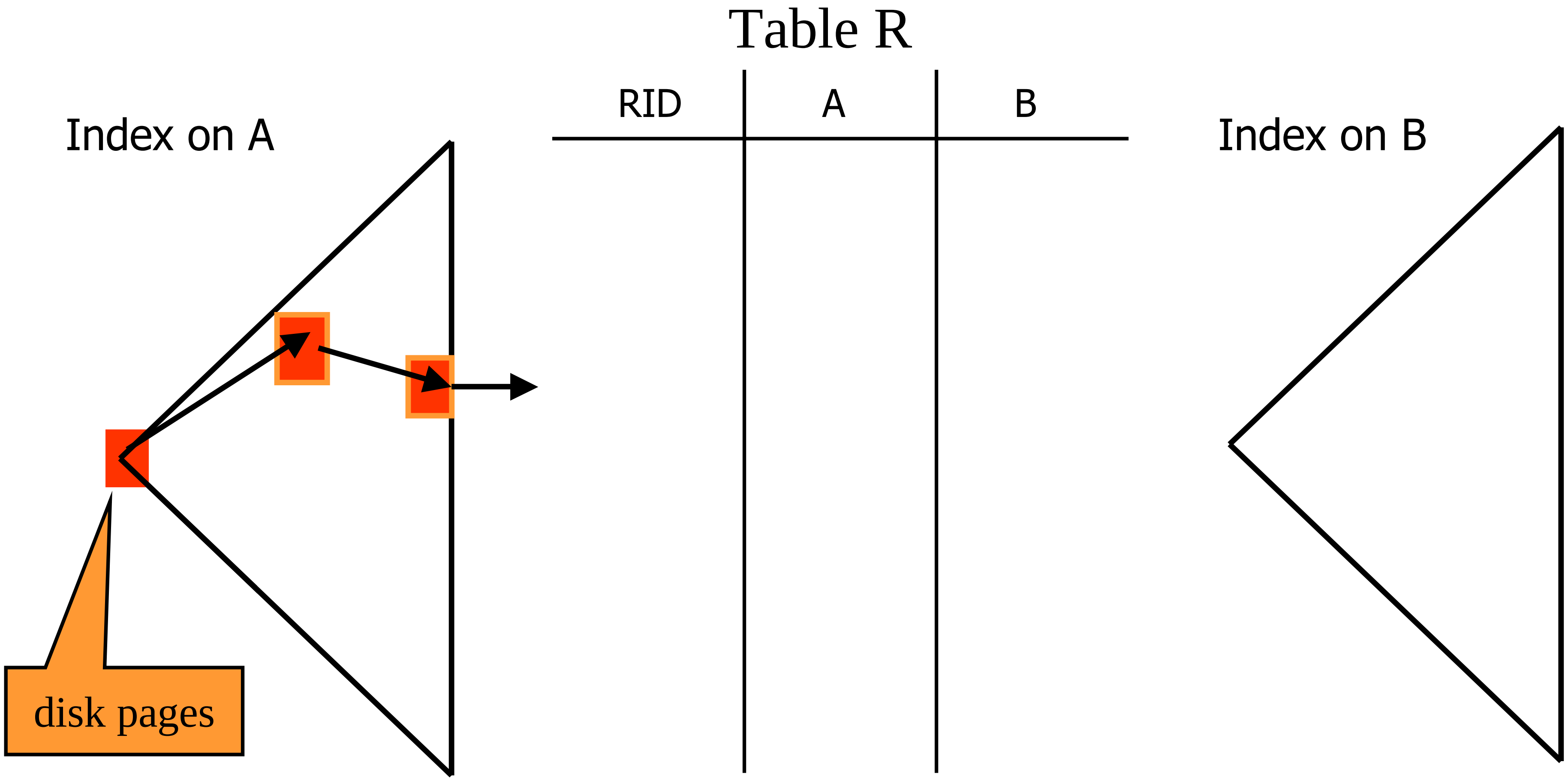


Table R

RID	A	B

Index on A

Index on B

disk pages

Traditional Horizontal Approach (Record-at-a-time approach)

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)

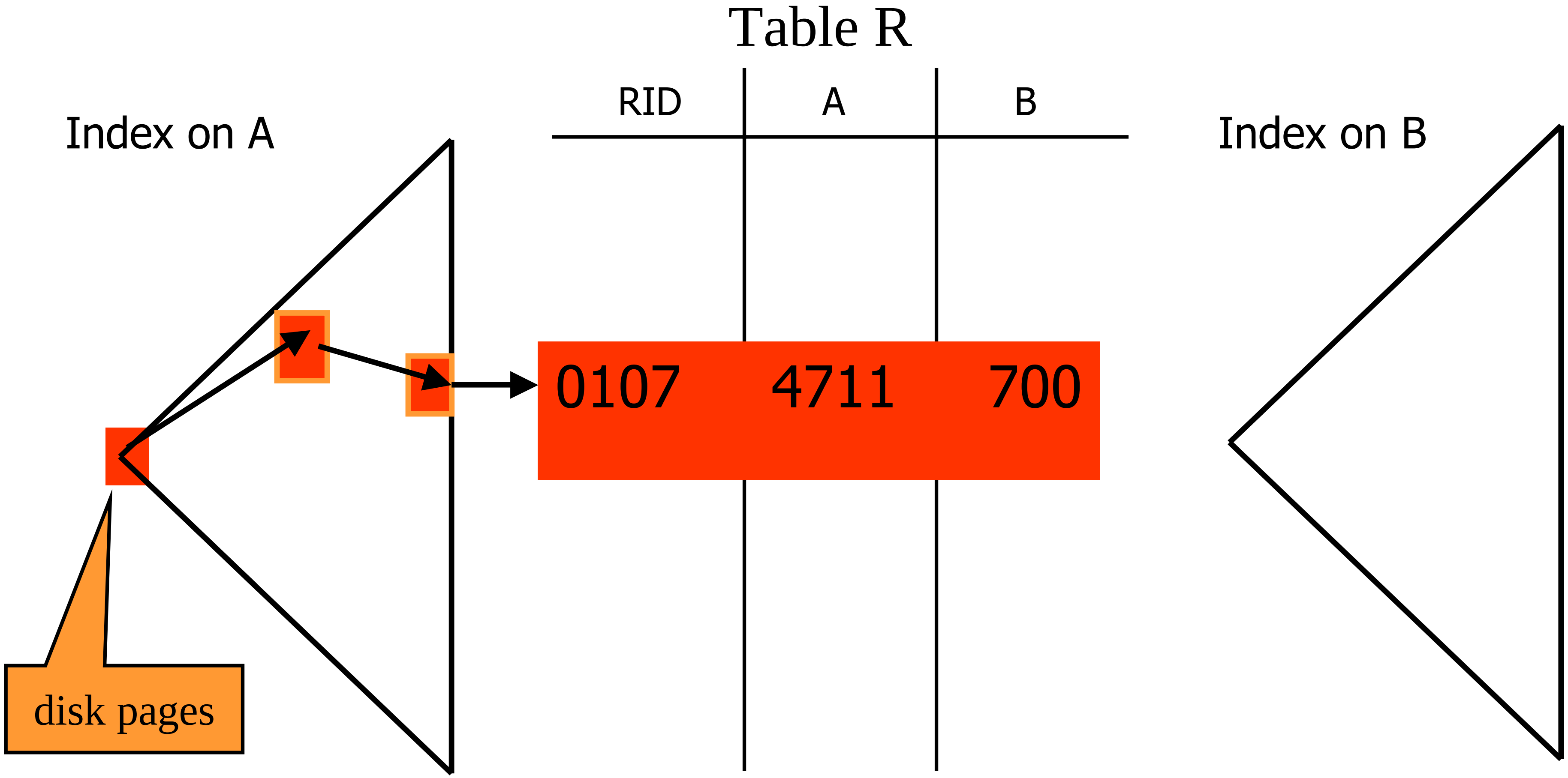


Table R

RID	A	B
0107	4711	700

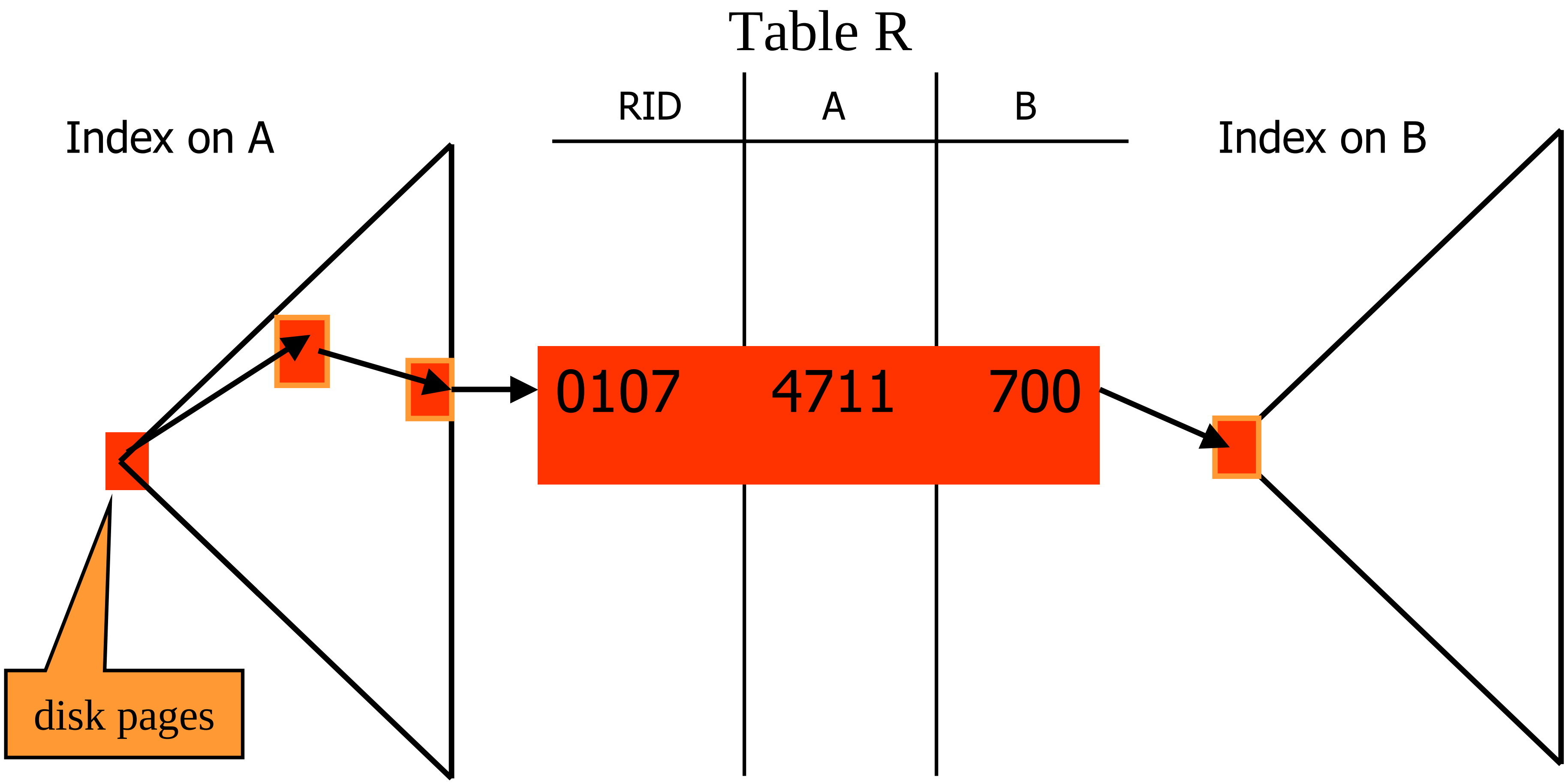
Index on A

Index on B

disk pages

Traditional Horizontal Approach (Record-at-a-time approach)

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)



Traditional Horizontal Approach (Record-at-a-time approach)

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)

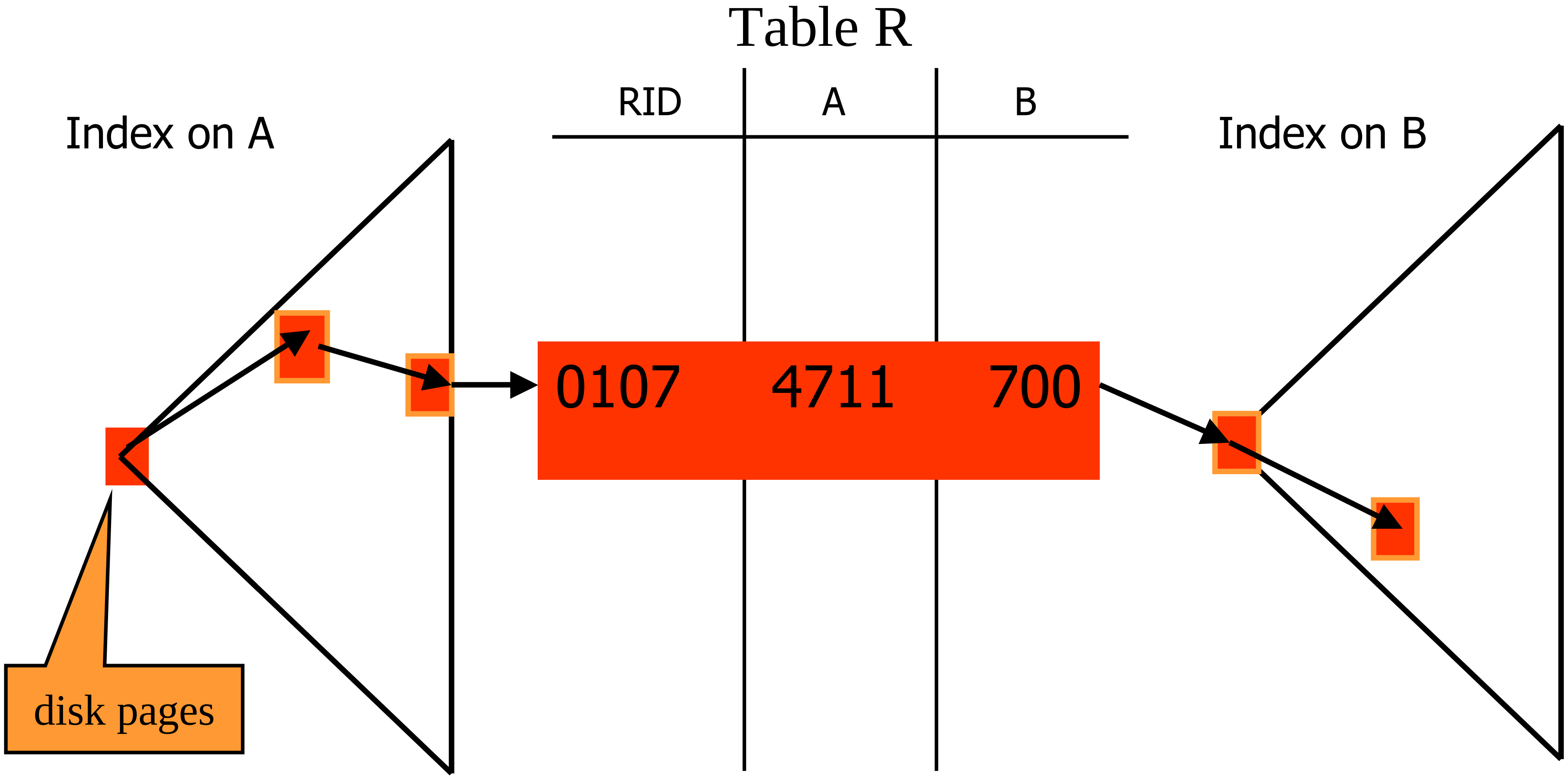


Table R

RID	A	B
0107	4711	700

Traditional Horizontal Approach (Record-at-a-time approach)

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)

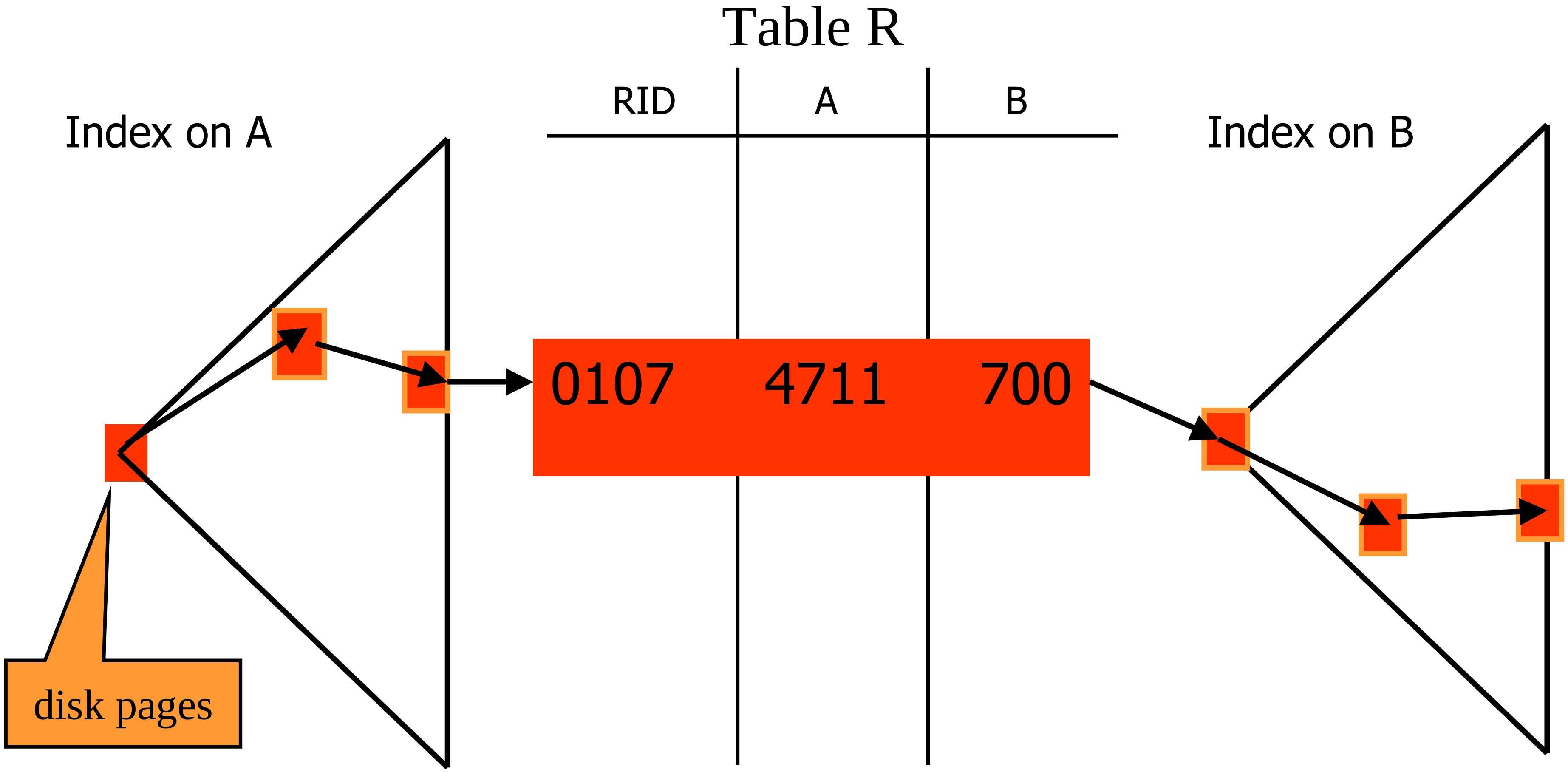


Table R

RID	A	B
0107	4711	700

Traditional Horizontal Approach (Record-at-a-time approach)

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)

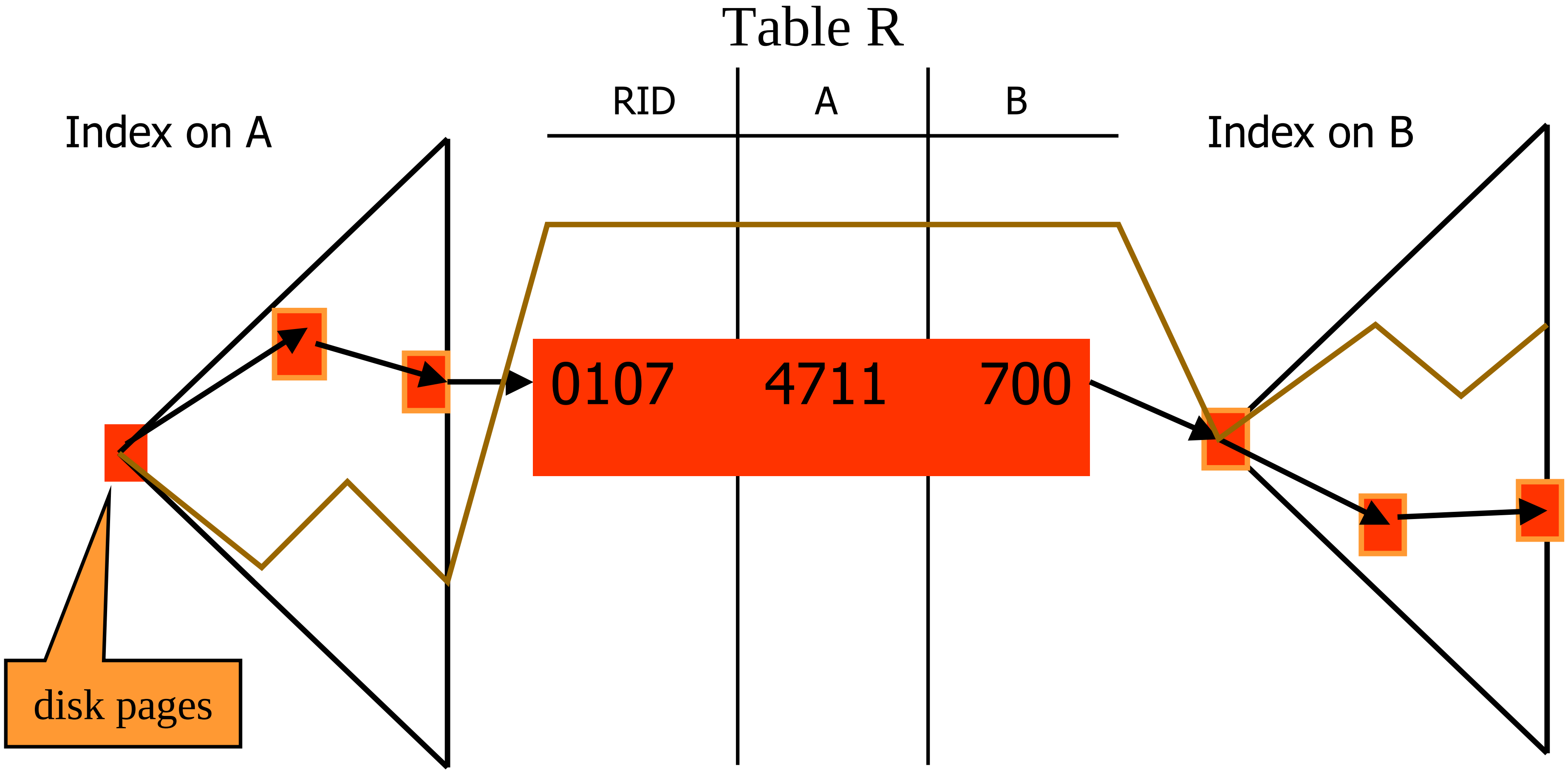
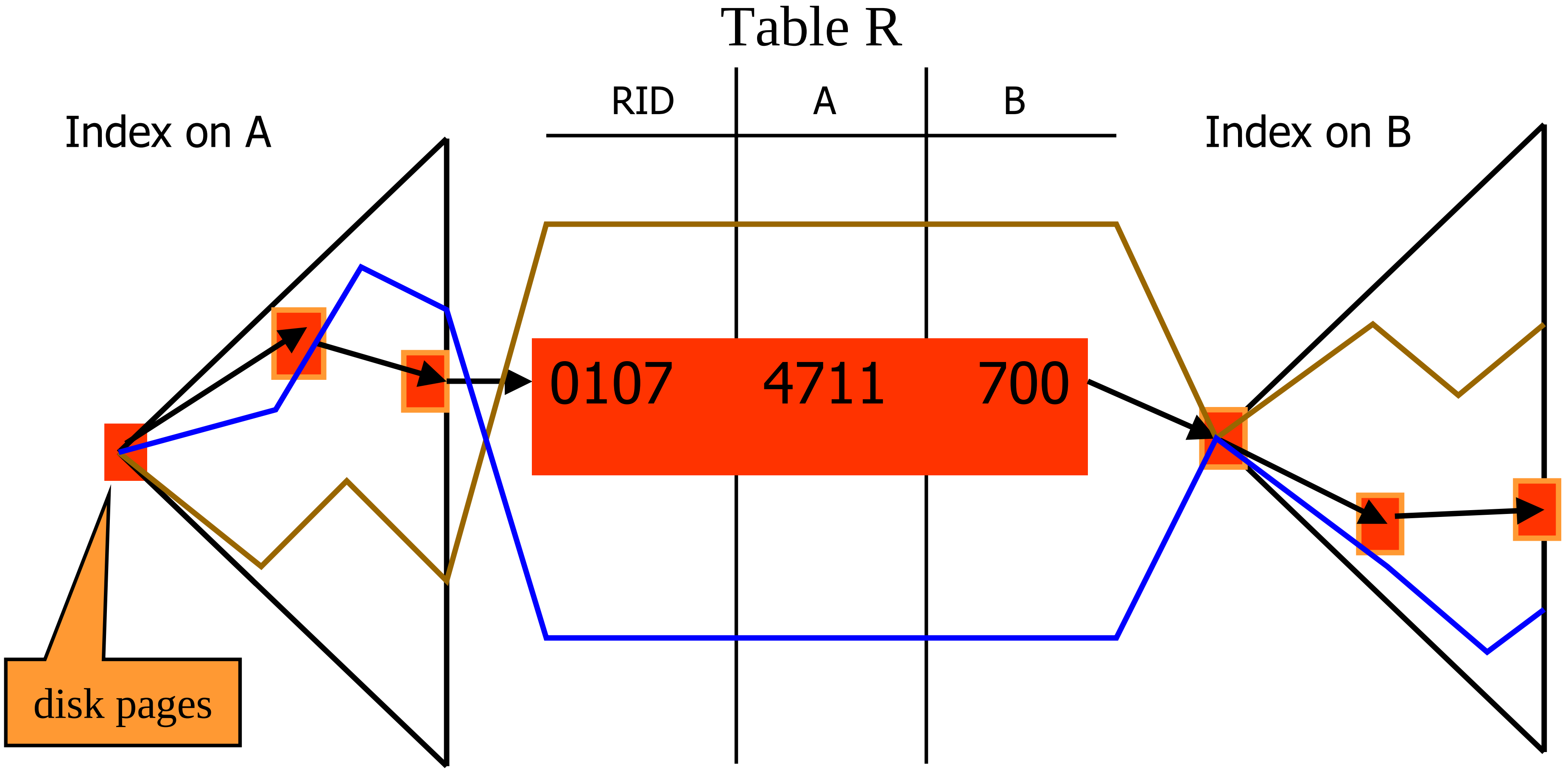


Table R

RID	A	B
0107	4711	700

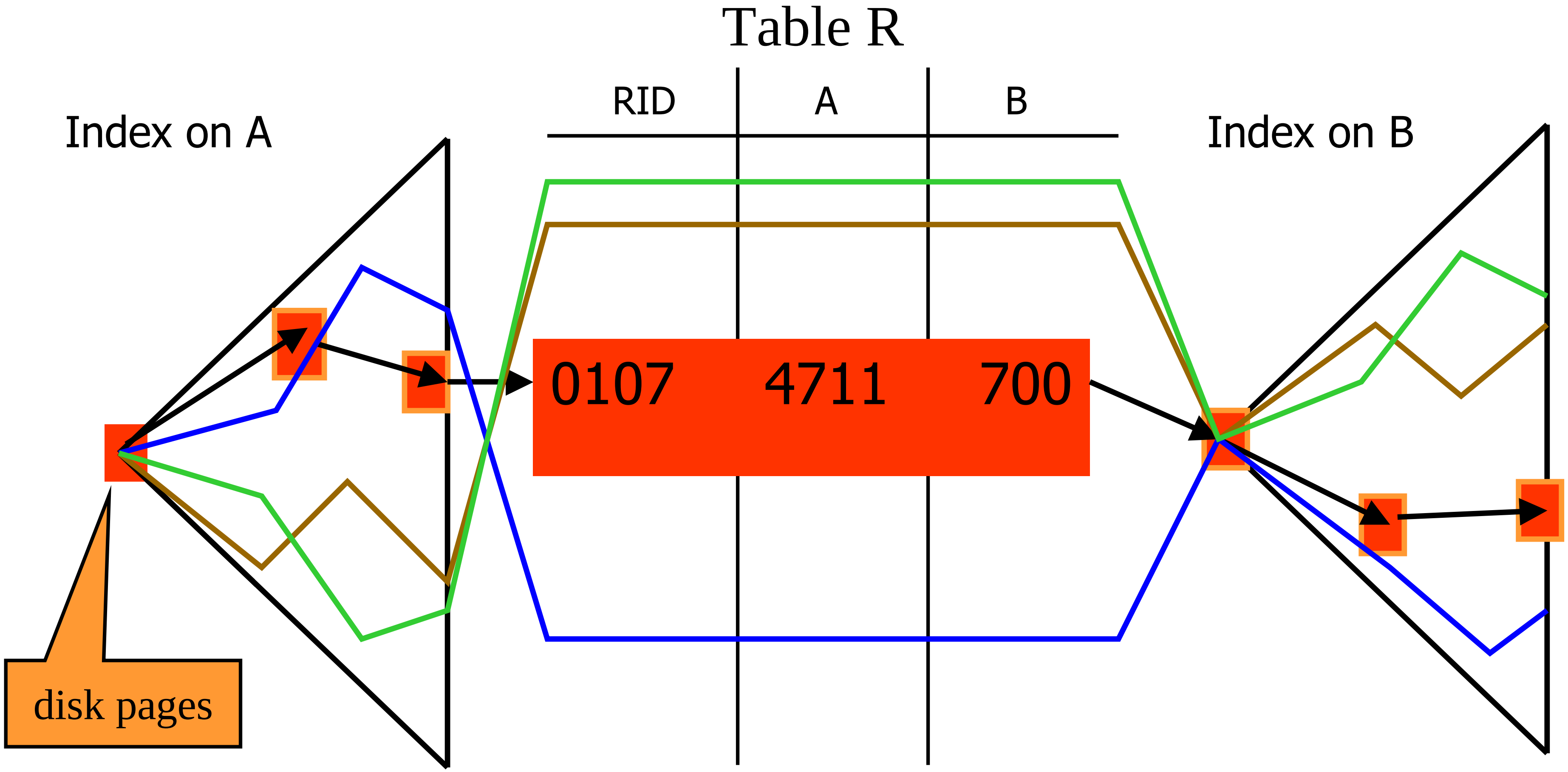
Traditional Horizontal Approach (Record-at-a-time approach)

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)



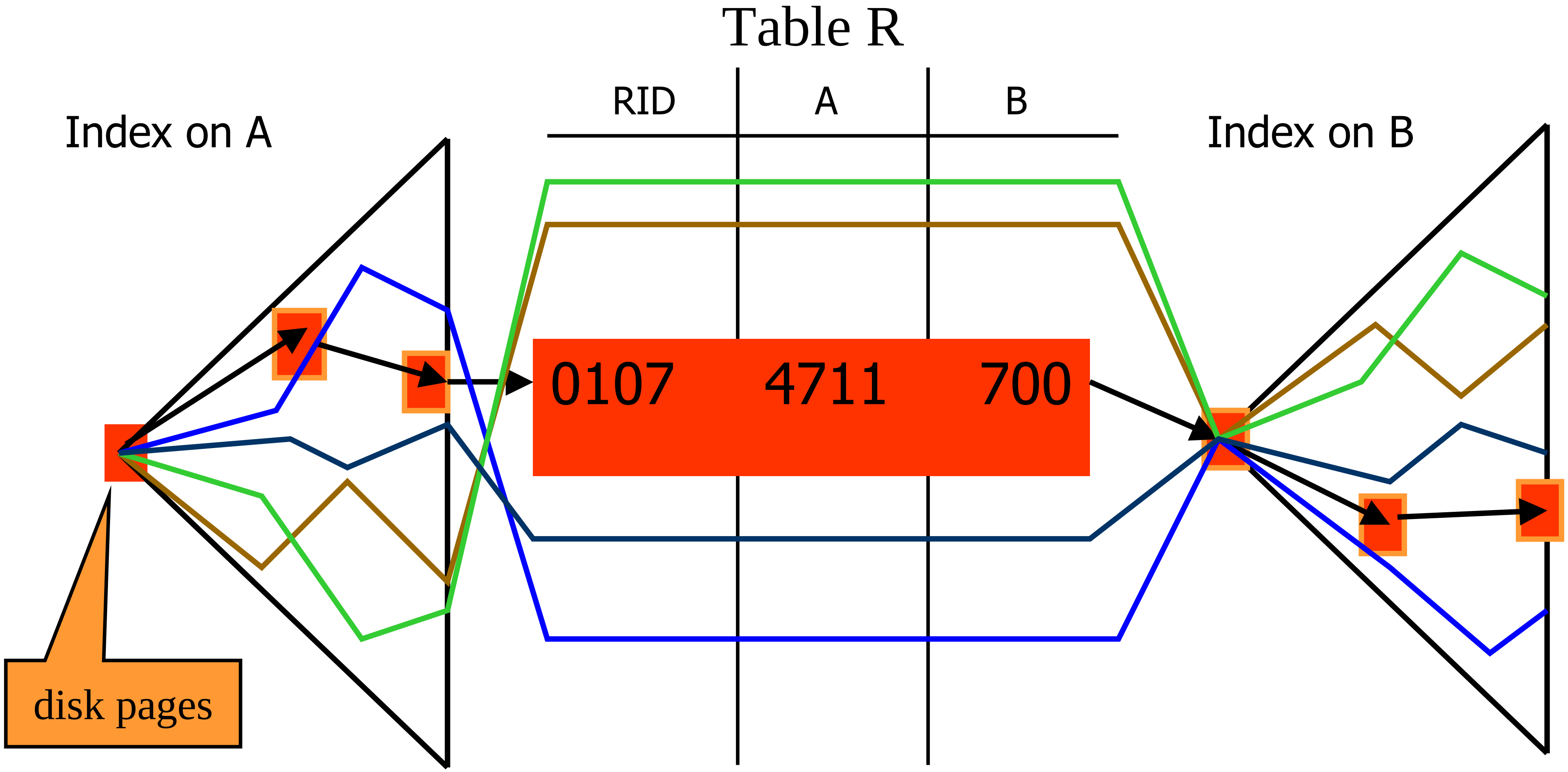
Traditional Horizontal Approach (Record-at-a-time approach)

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)



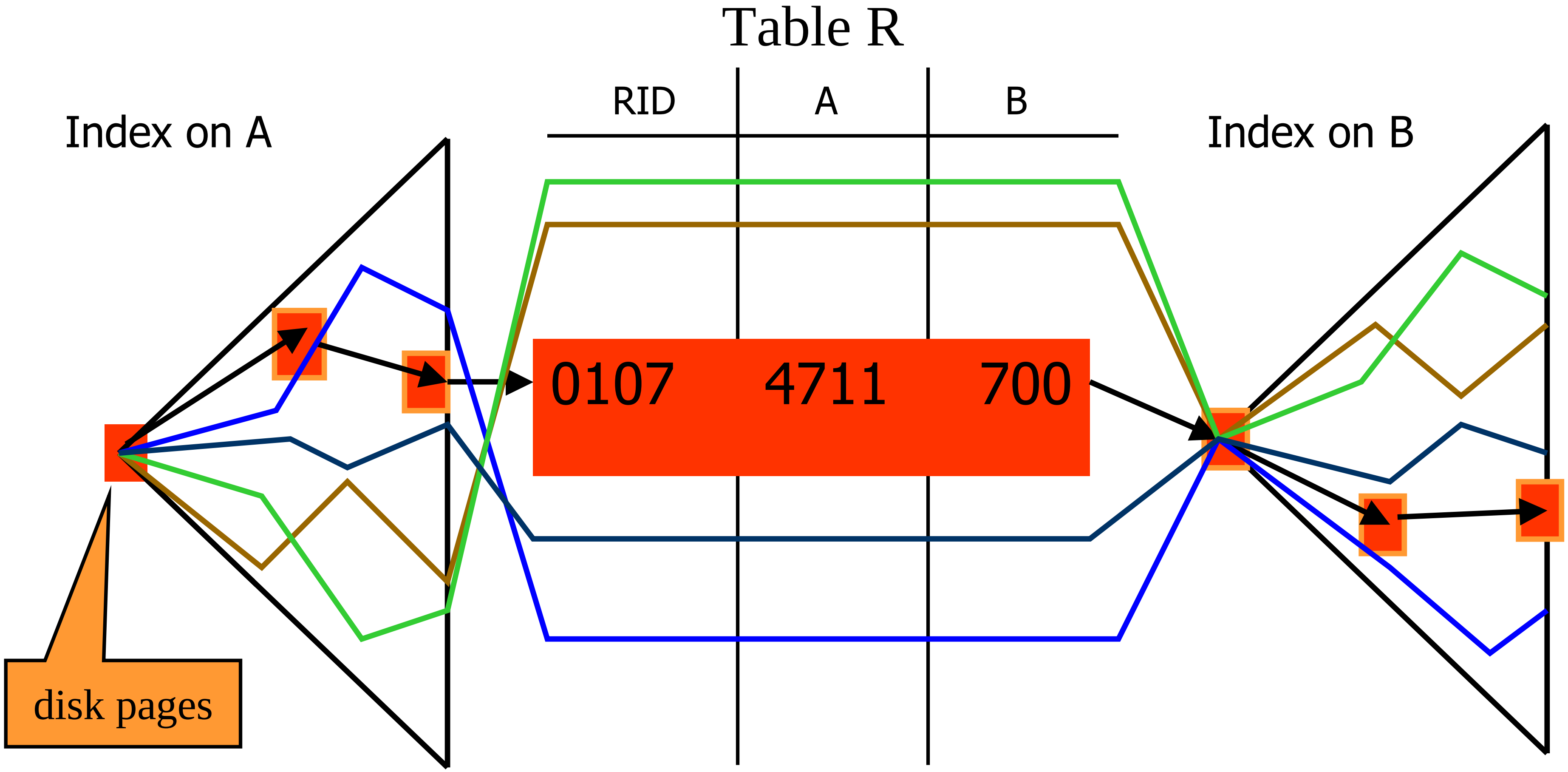
Traditional Horizontal Approach (Record-at-a-time approach)

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)



Traditional Horizontal Approach (Record-at-a-time approach)

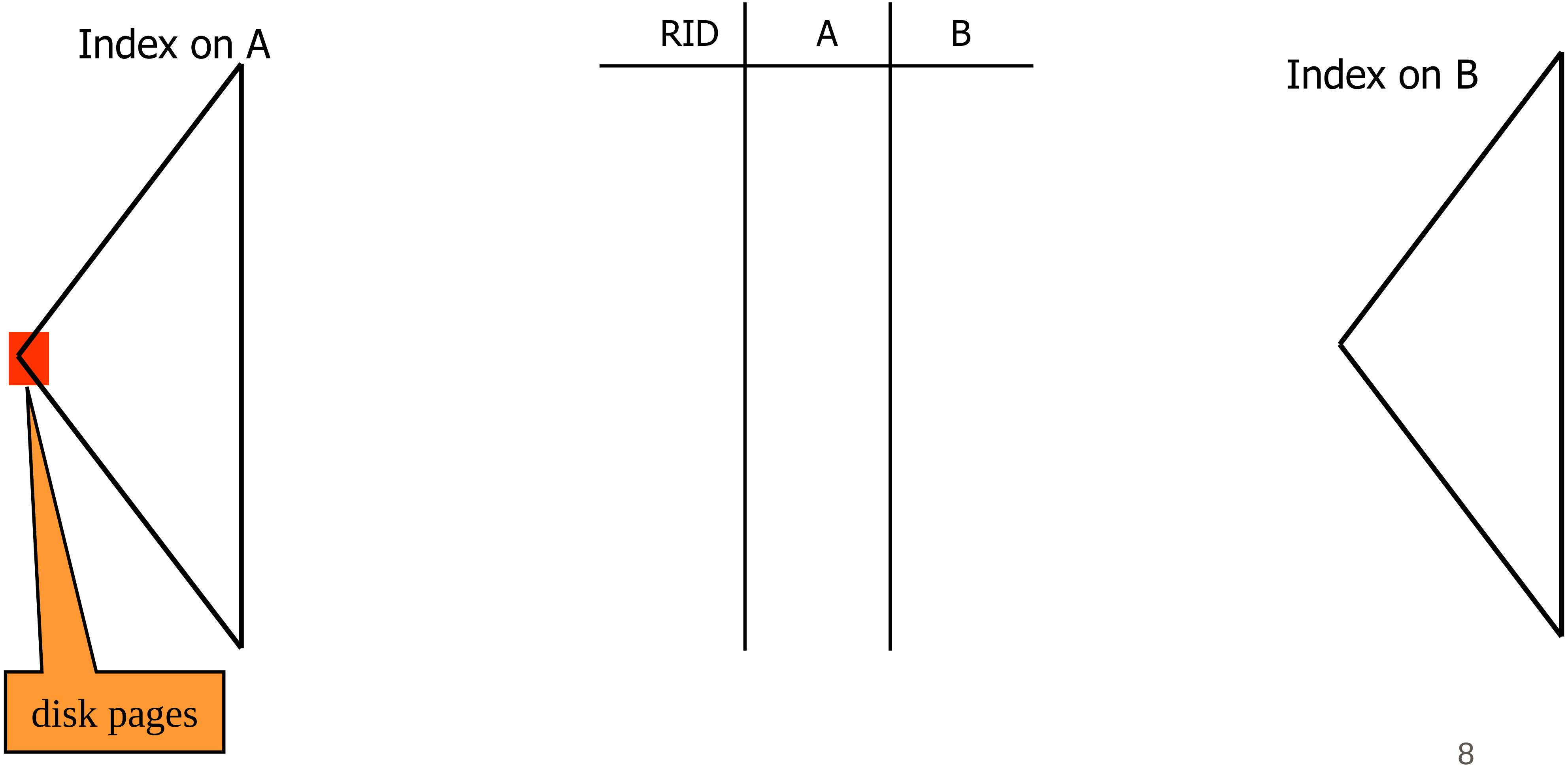
Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)



=> Random I/O on index leaf pages and table pages

New Set-Oriented Vertical Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)



Index on A

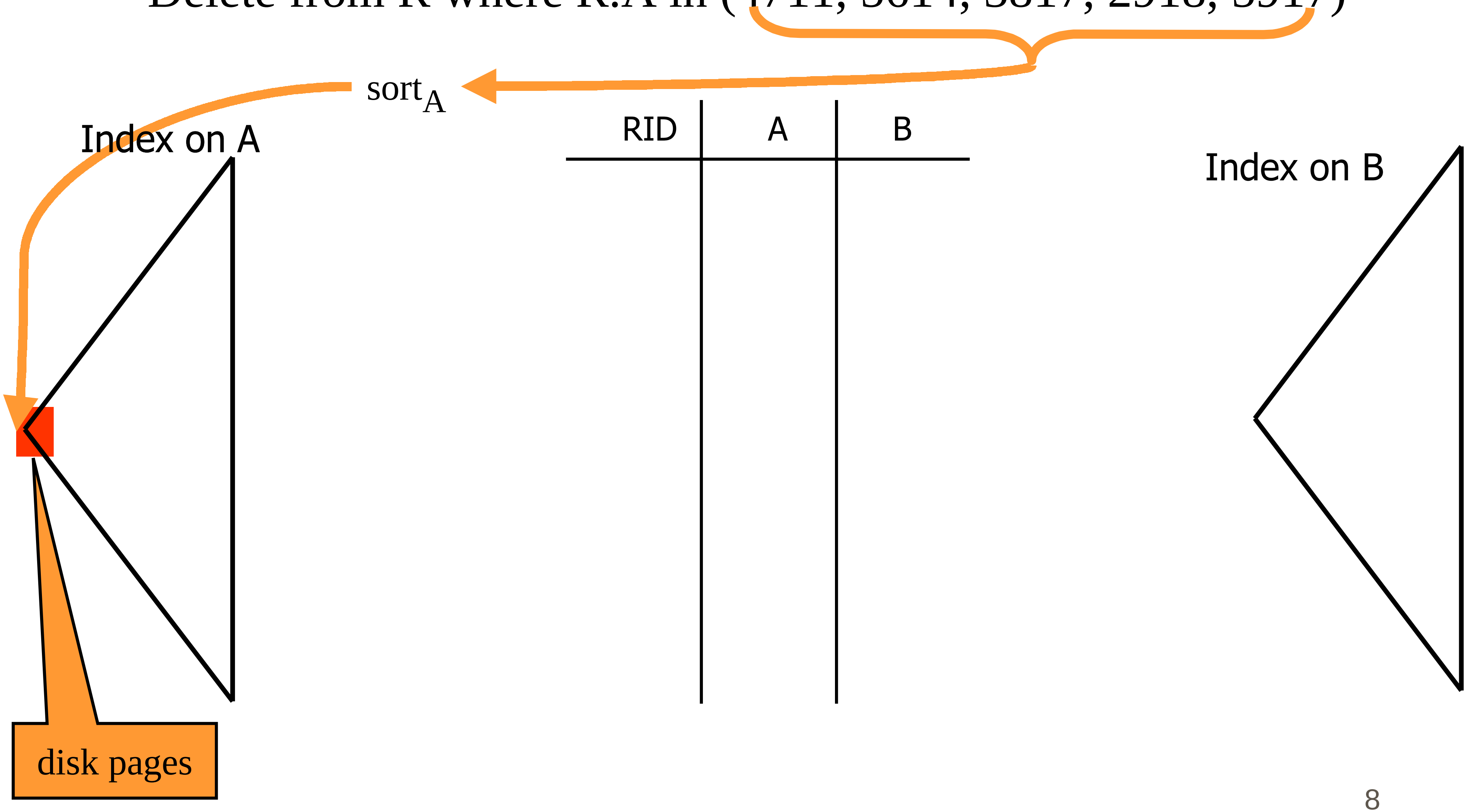
RID	A	B

Index on B

disk pages

New Set-Oriented Vertical Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)



sort$_A$

Index on A

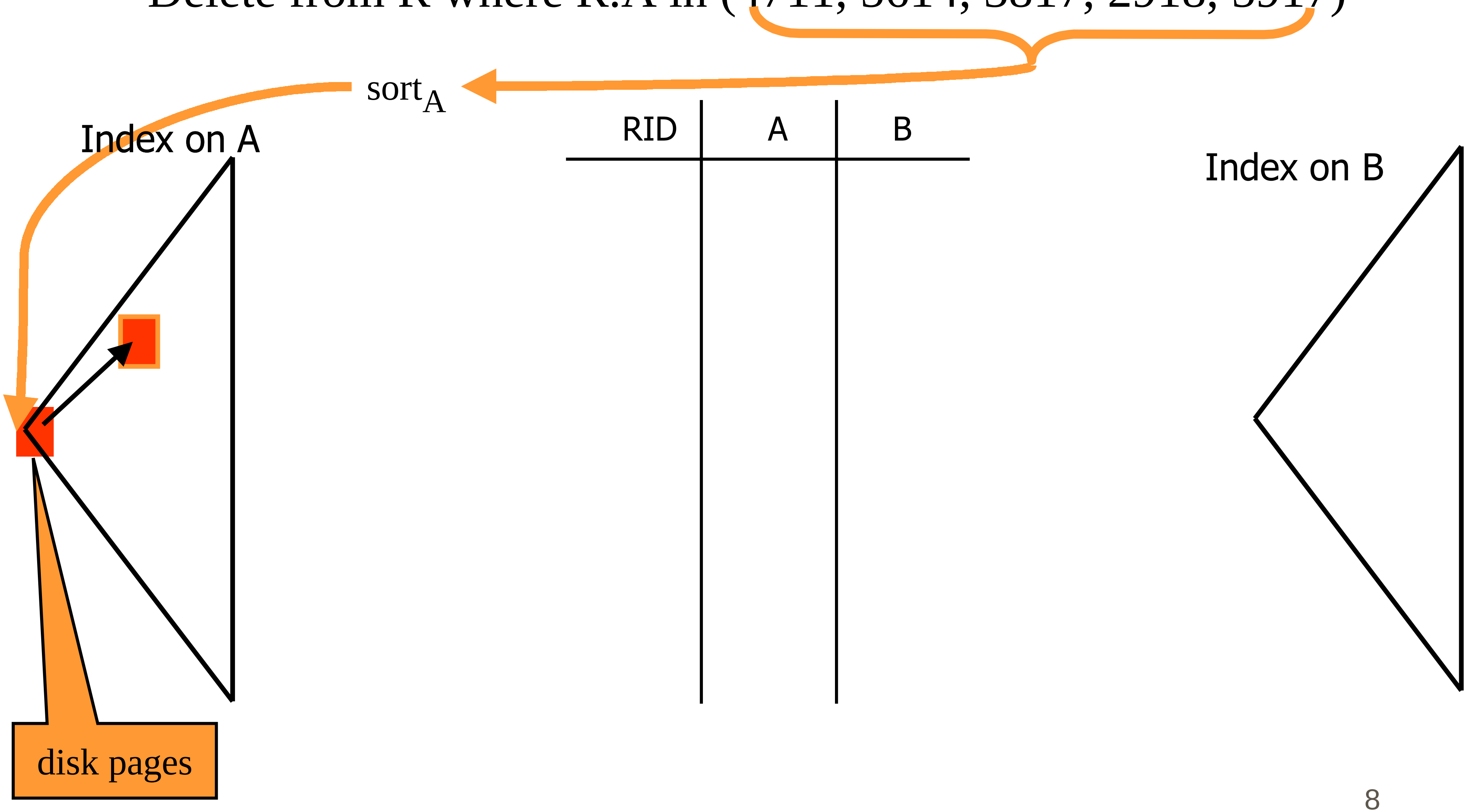
RID | A | B

Index on B

disk pages

New Set-Oriented Vertical Approach

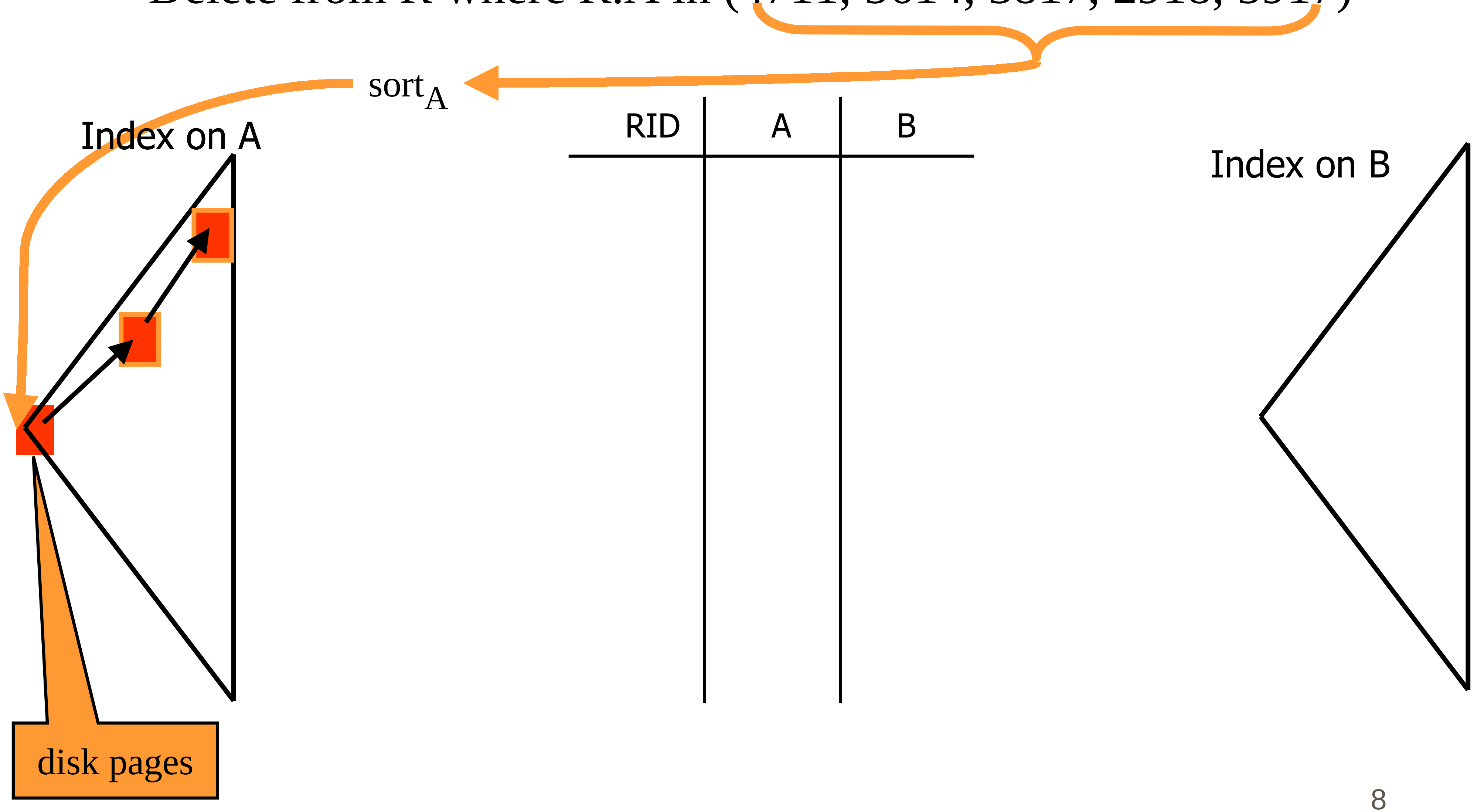
Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)



RID	A	B

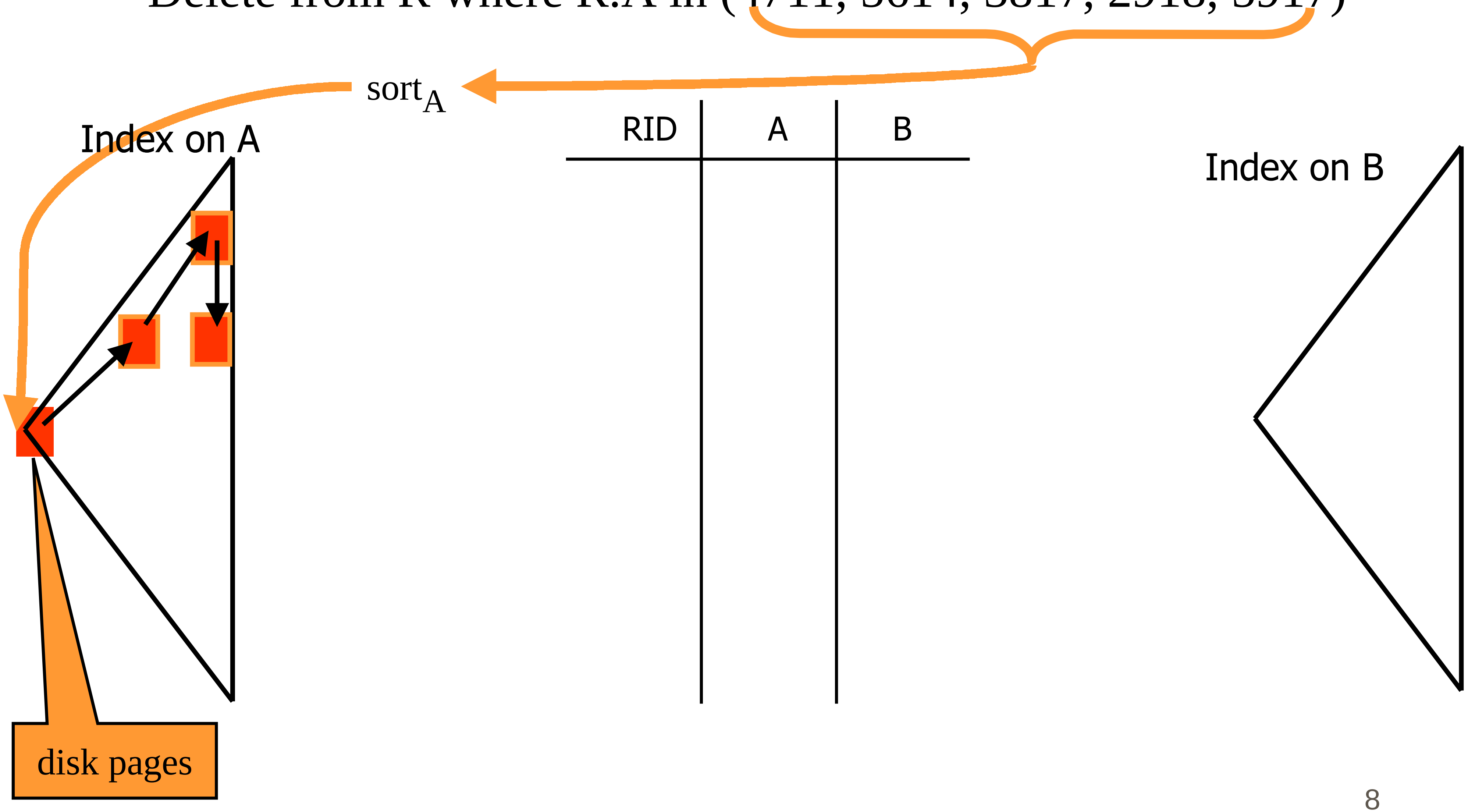
New Set-Oriented Vertical Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)



New Set-Oriented Vertical Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)



sort$_A$

Index on A

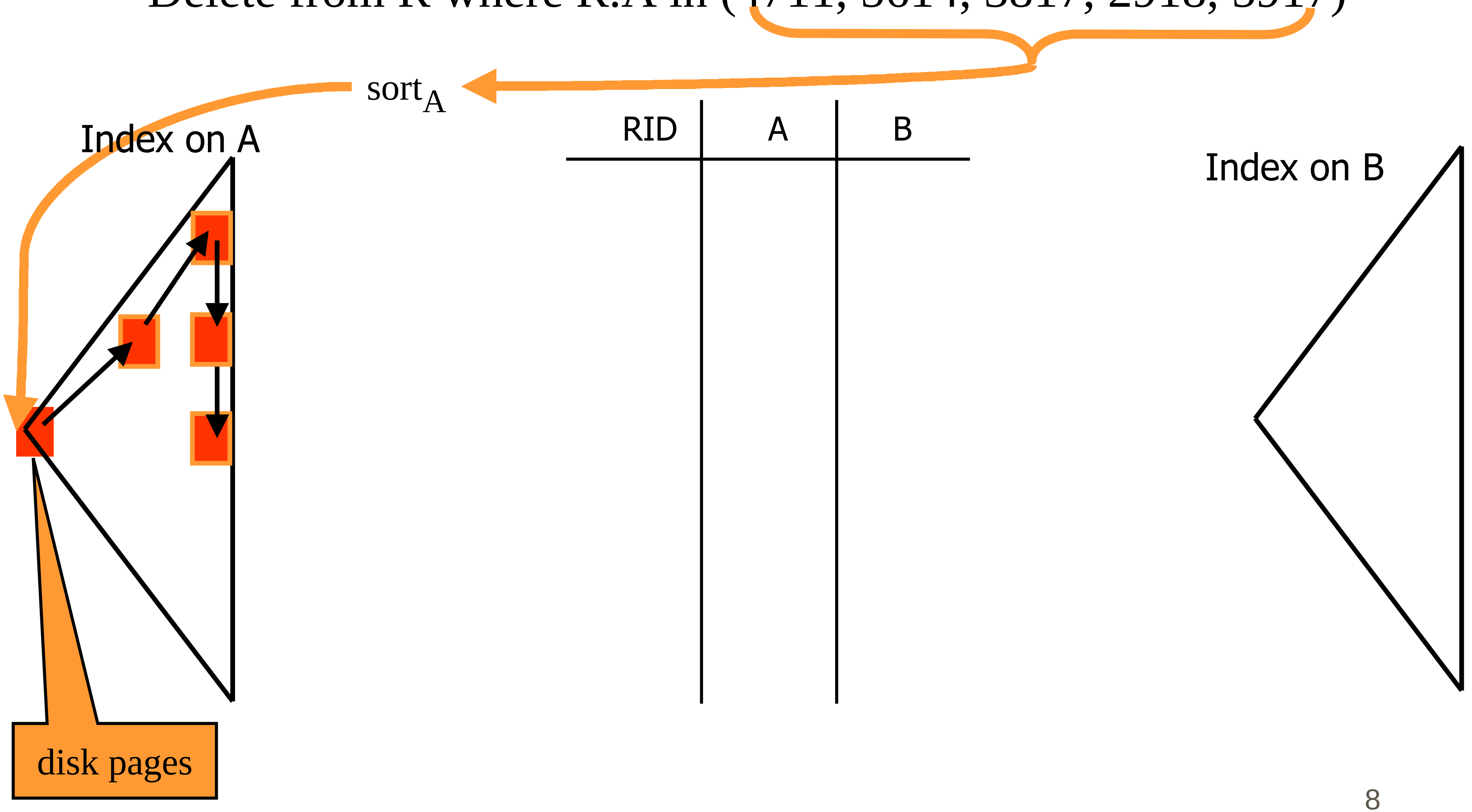
RID | A | B

Index on B

disk pages

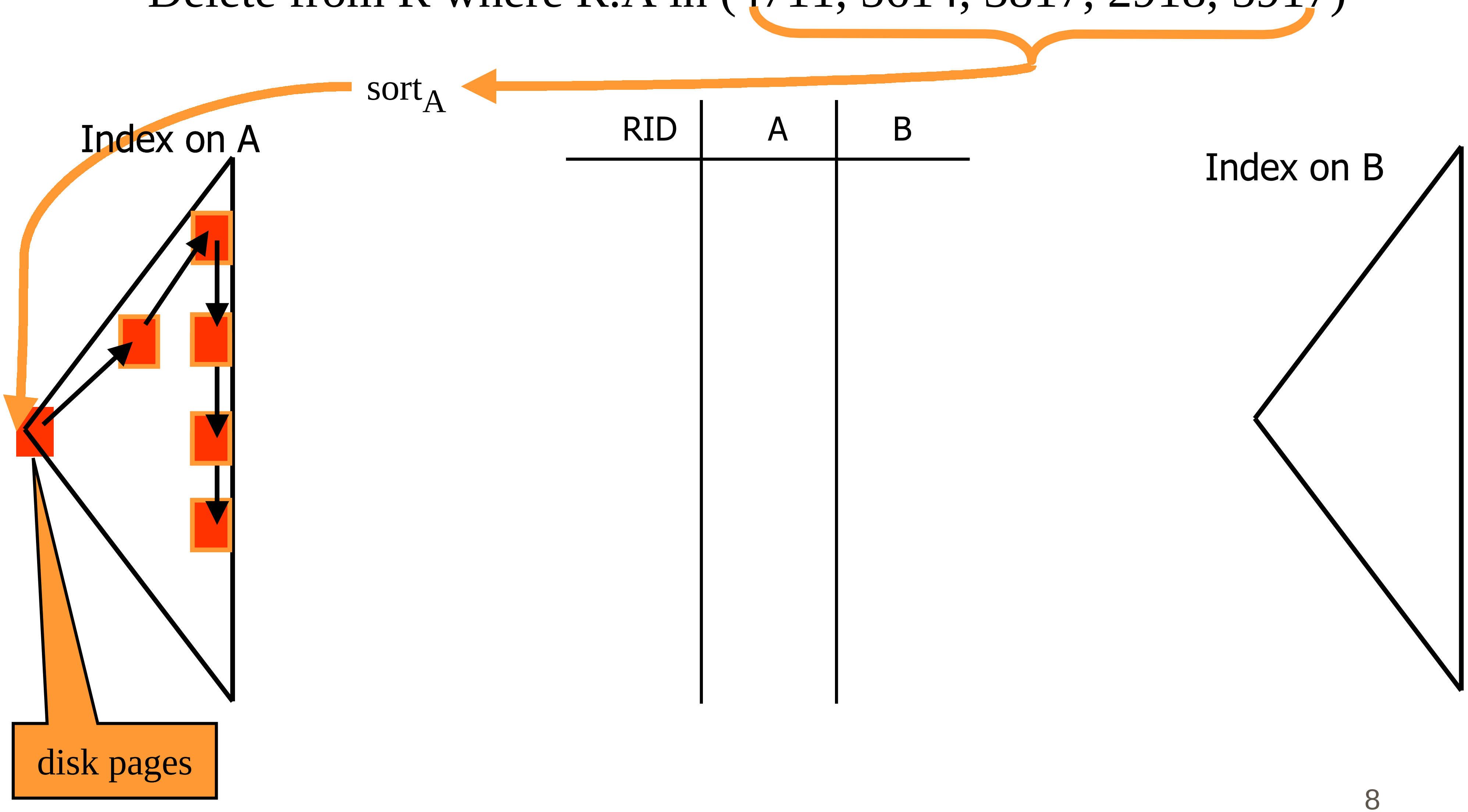
New Set-Oriented Vertical Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)



New Set-Oriented Vertical Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)



sort_A

Index on A

RID

A

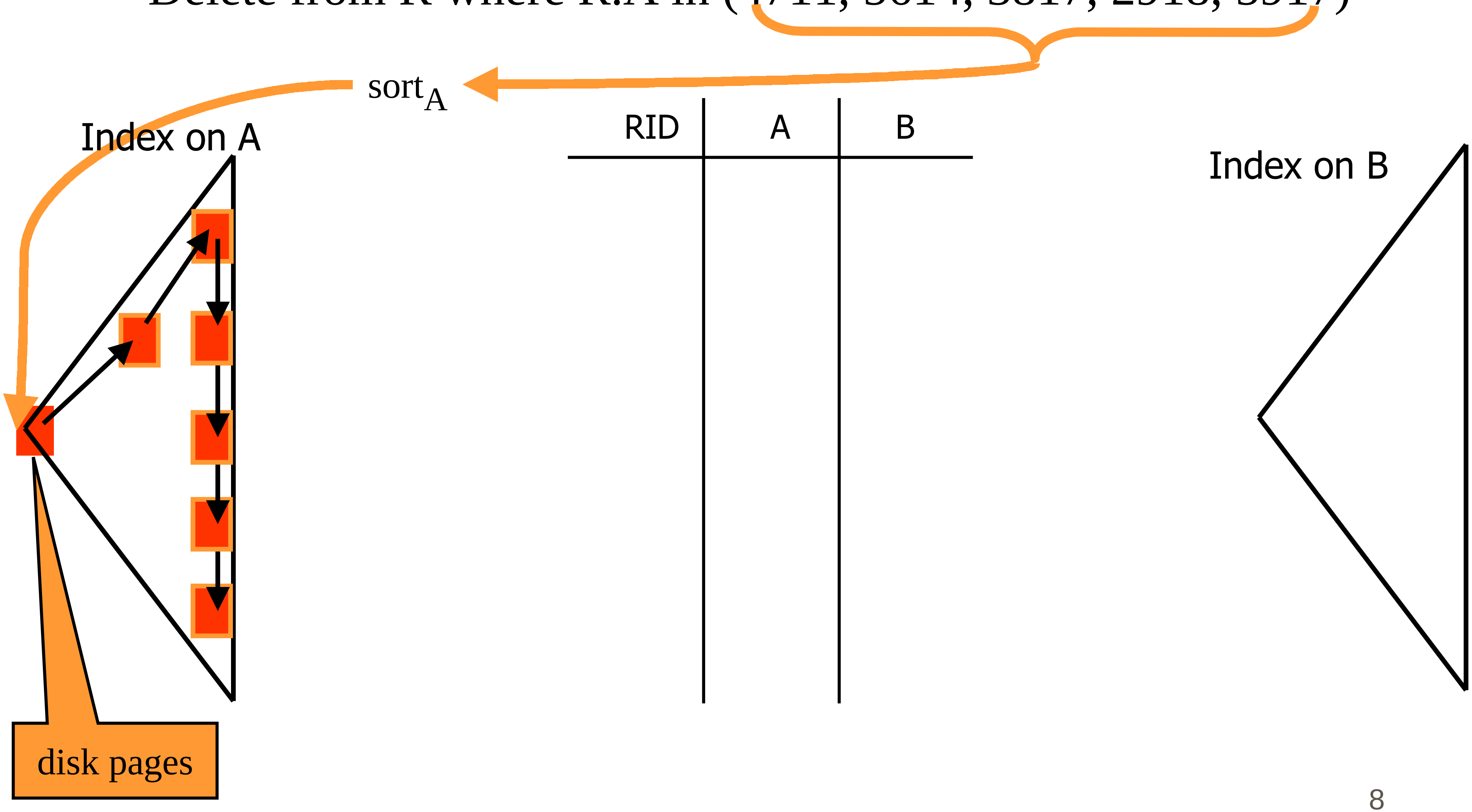
B

Index on B

disk pages

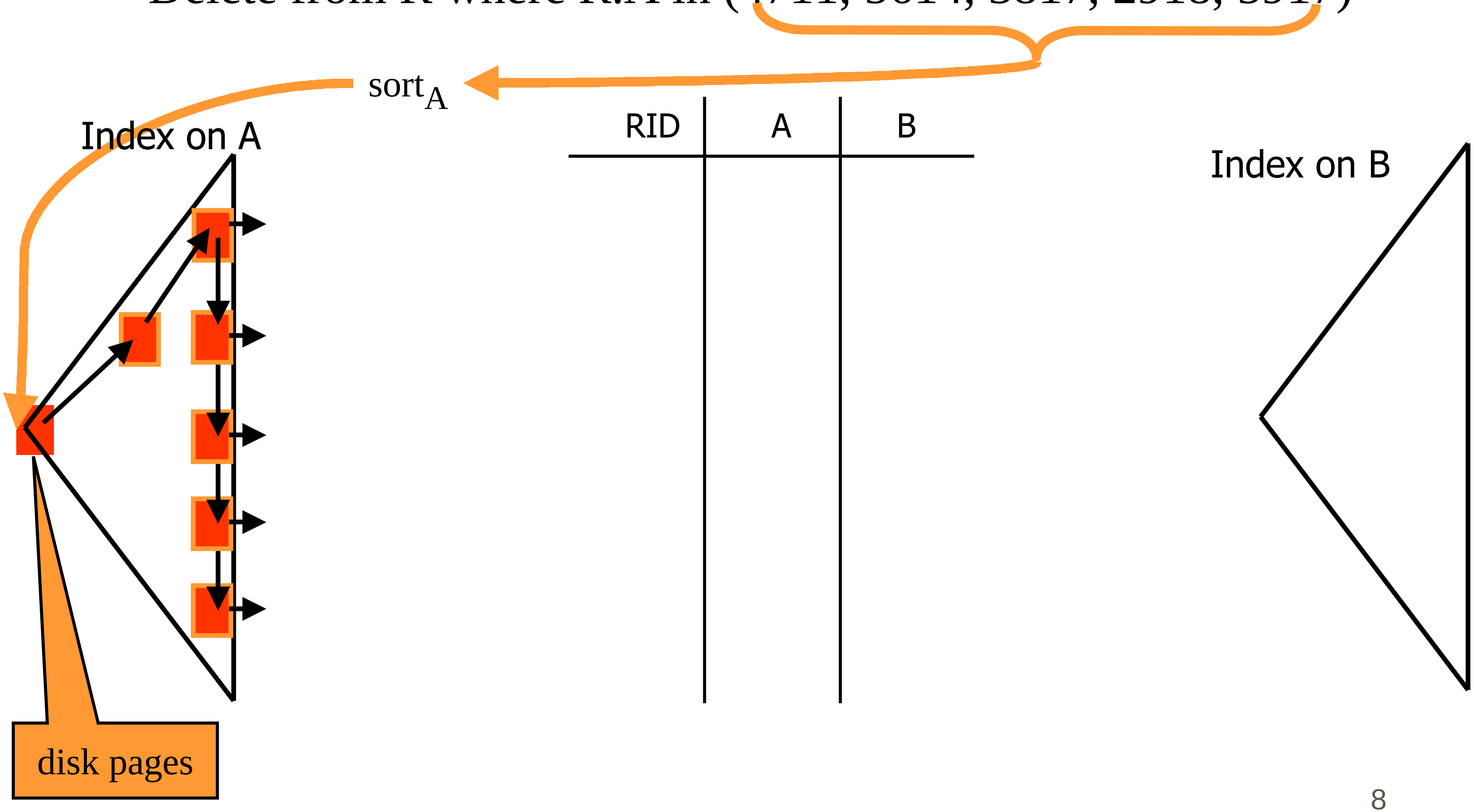
New Set-Oriented Vertical Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)



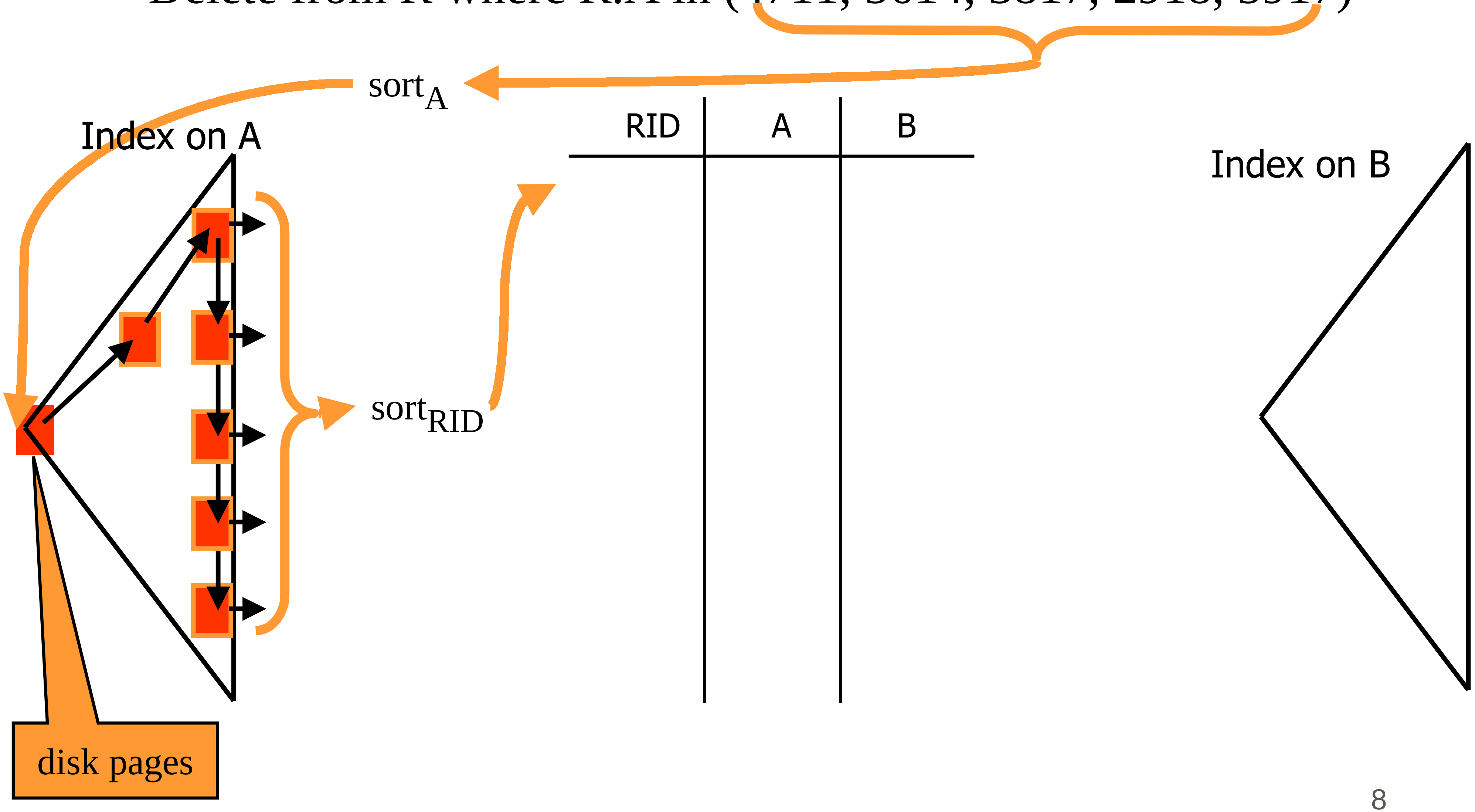
New Set-Oriented Vertical Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)



New Set-Oriented Vertical Approach

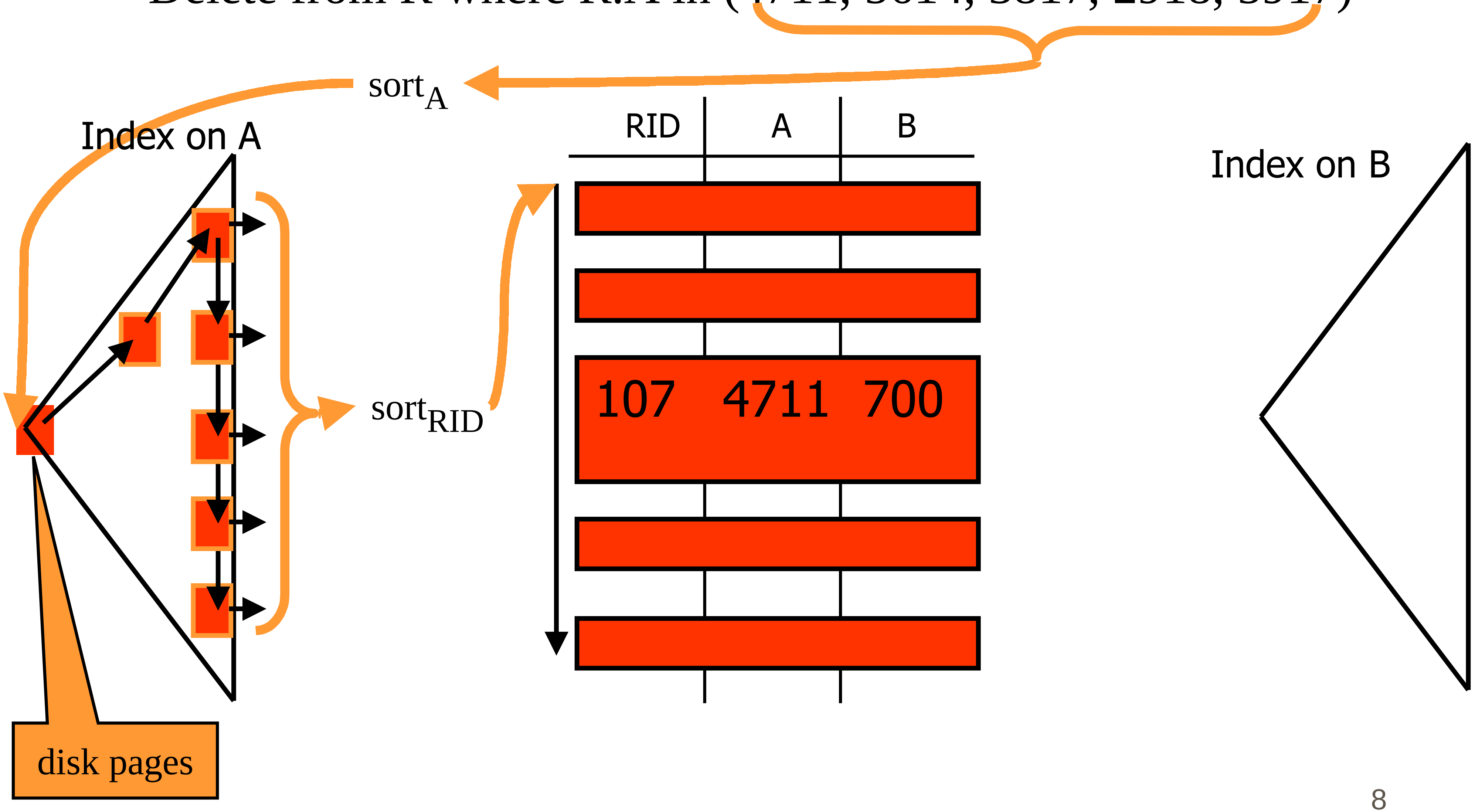
Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)



RID	A	B

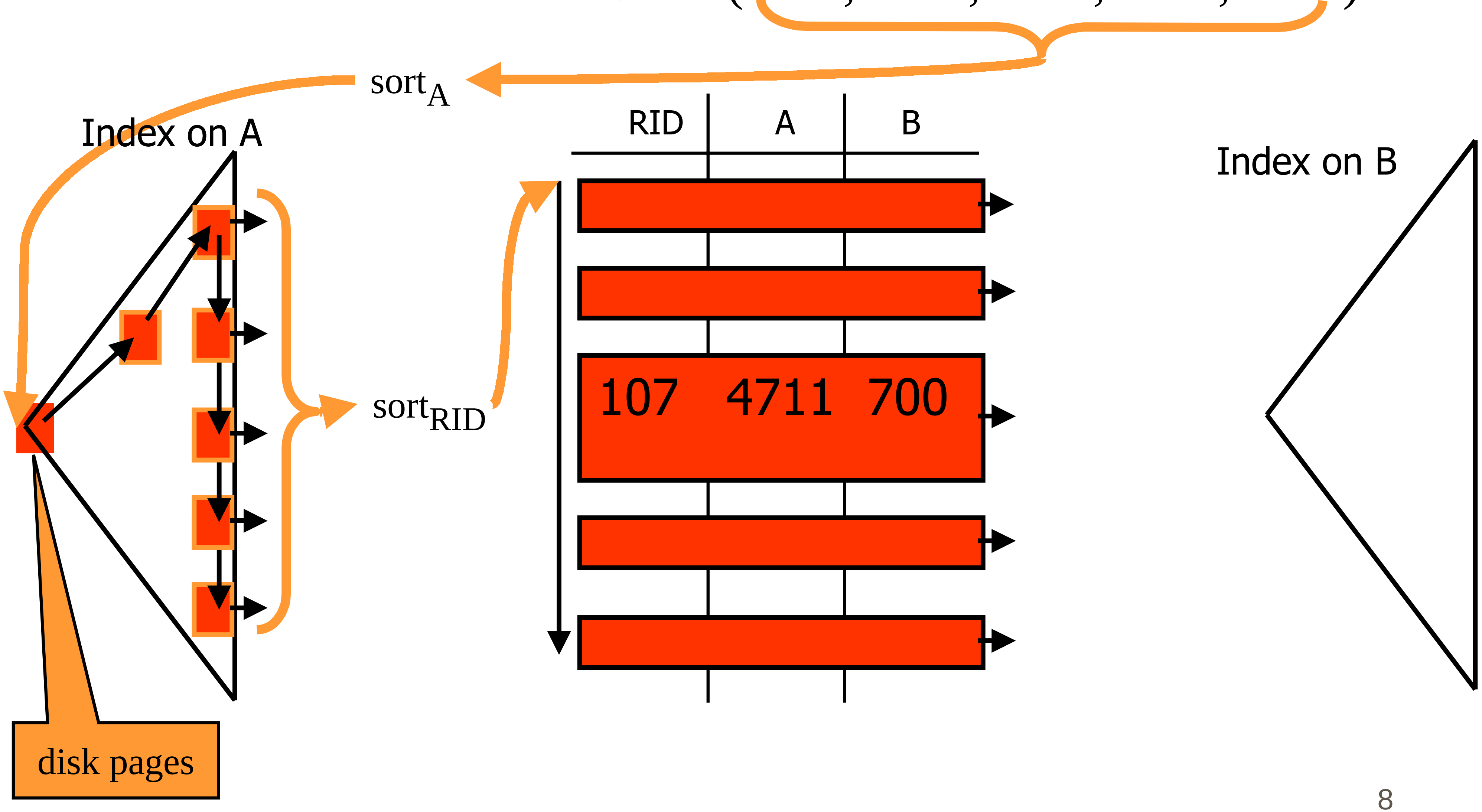
New Set-Oriented Vertical Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)



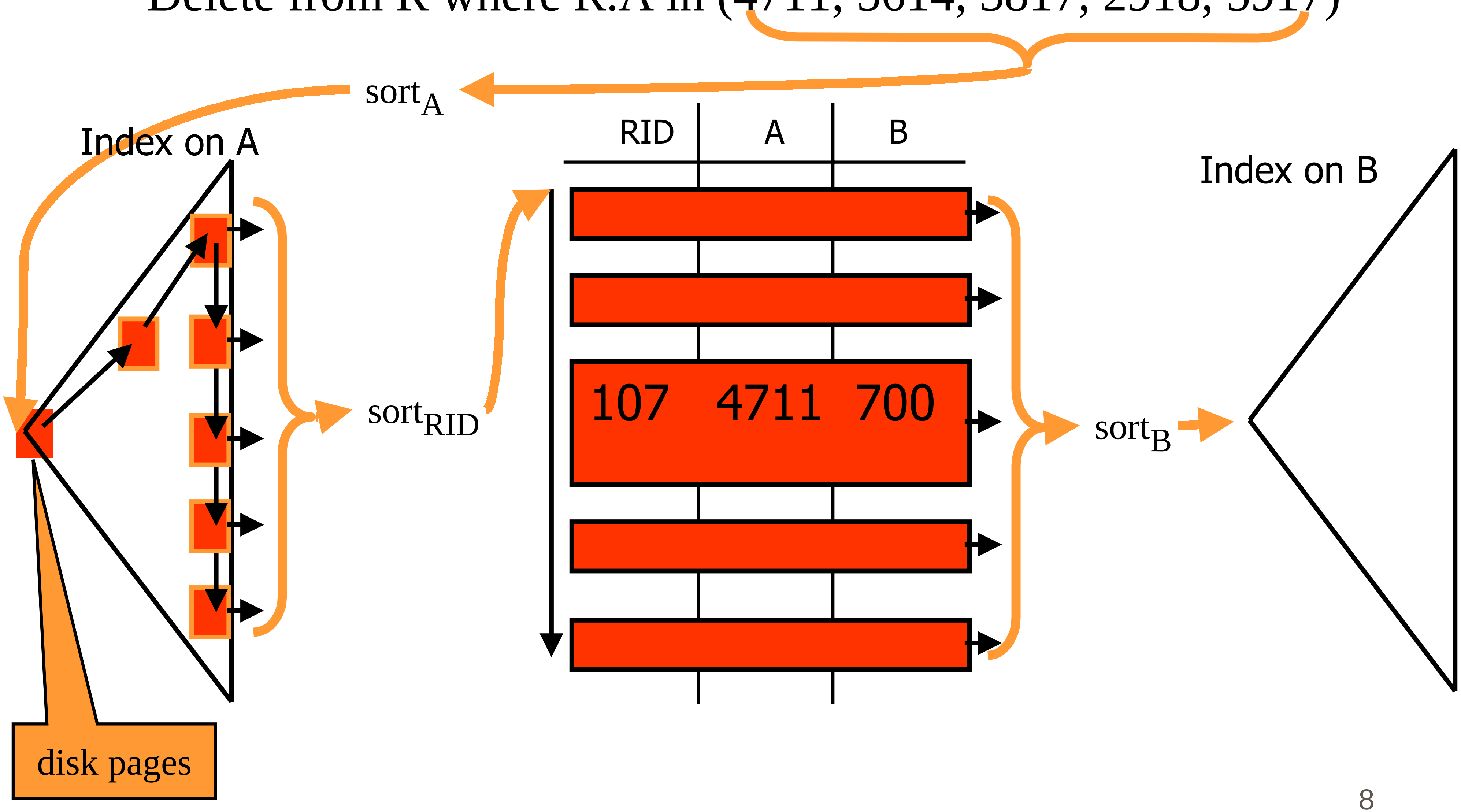
New Set-Oriented Vertical Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)

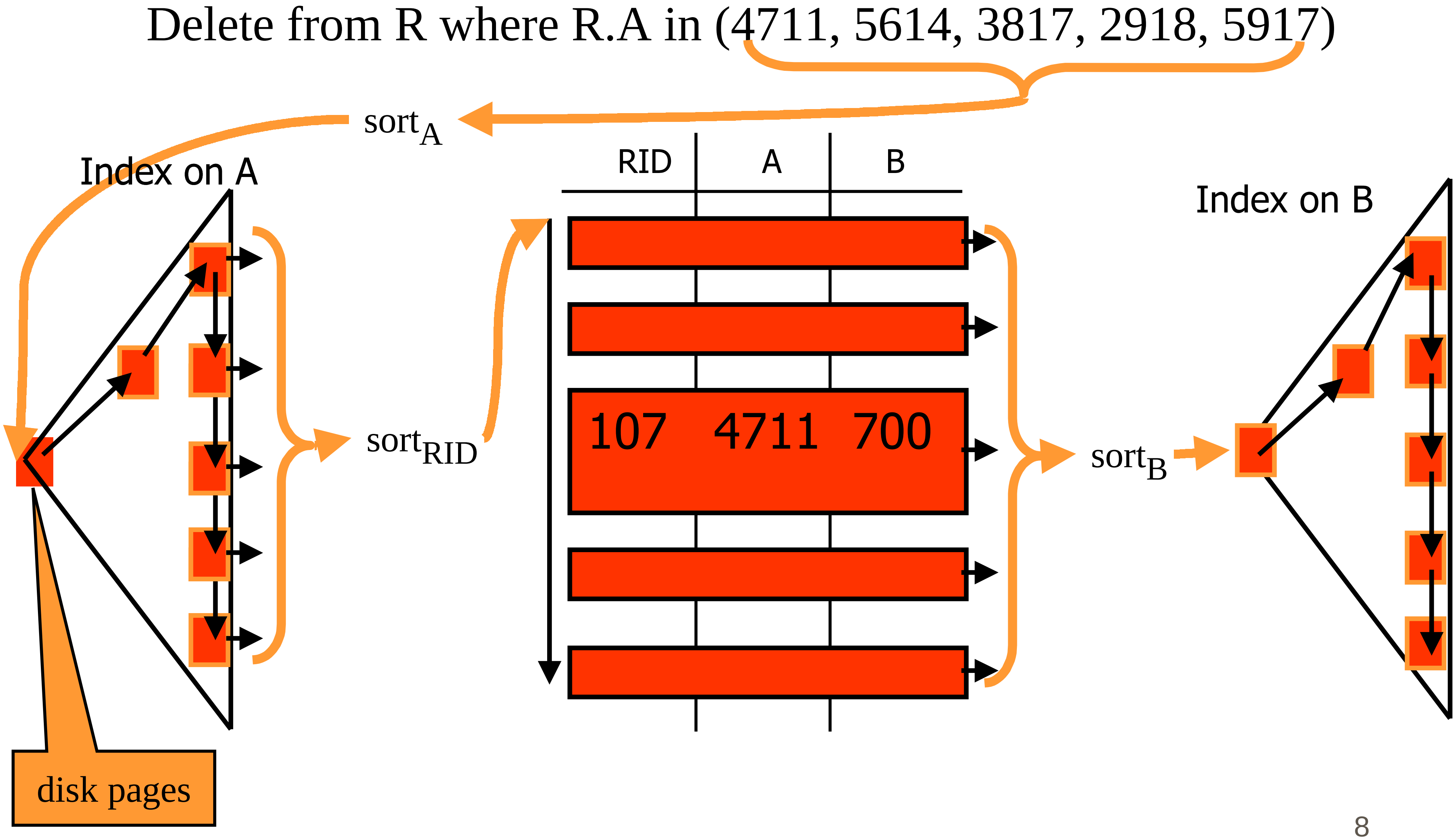


New Set-Oriented Vertical Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)

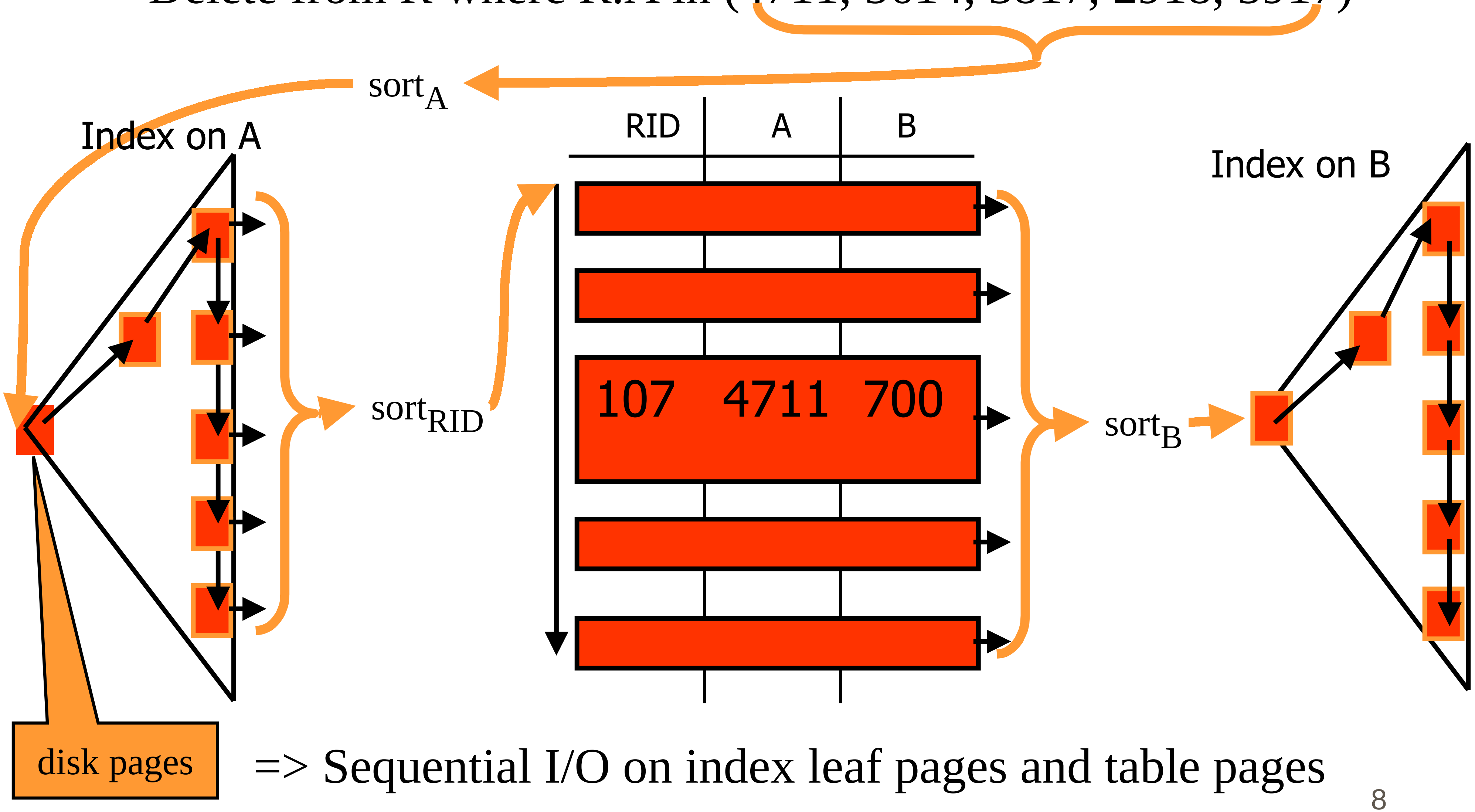


New Set-Oriented Vertical Approach



New Set-Oriented Vertical Approach

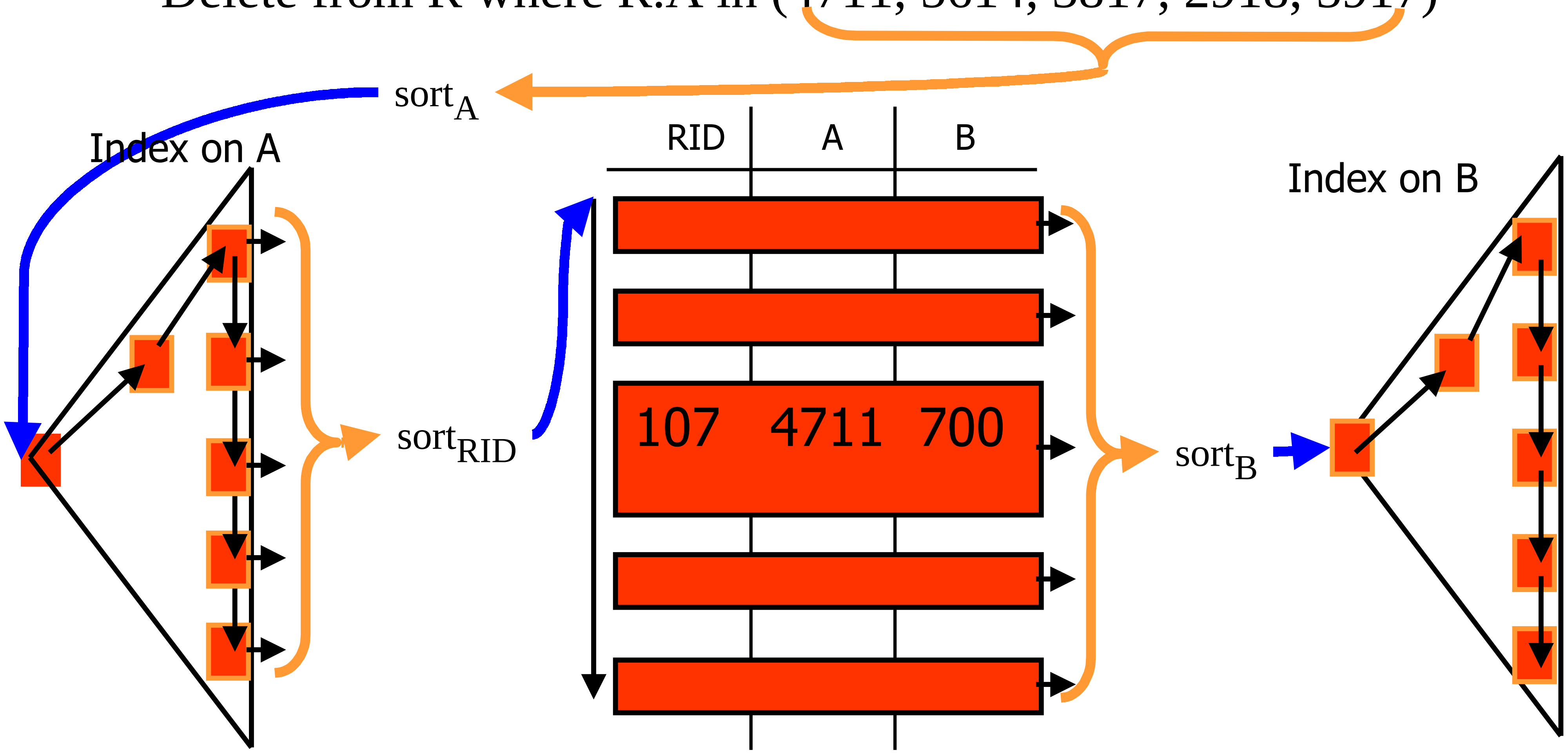
Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)



=> Sequential I/O on index leaf pages and table pages

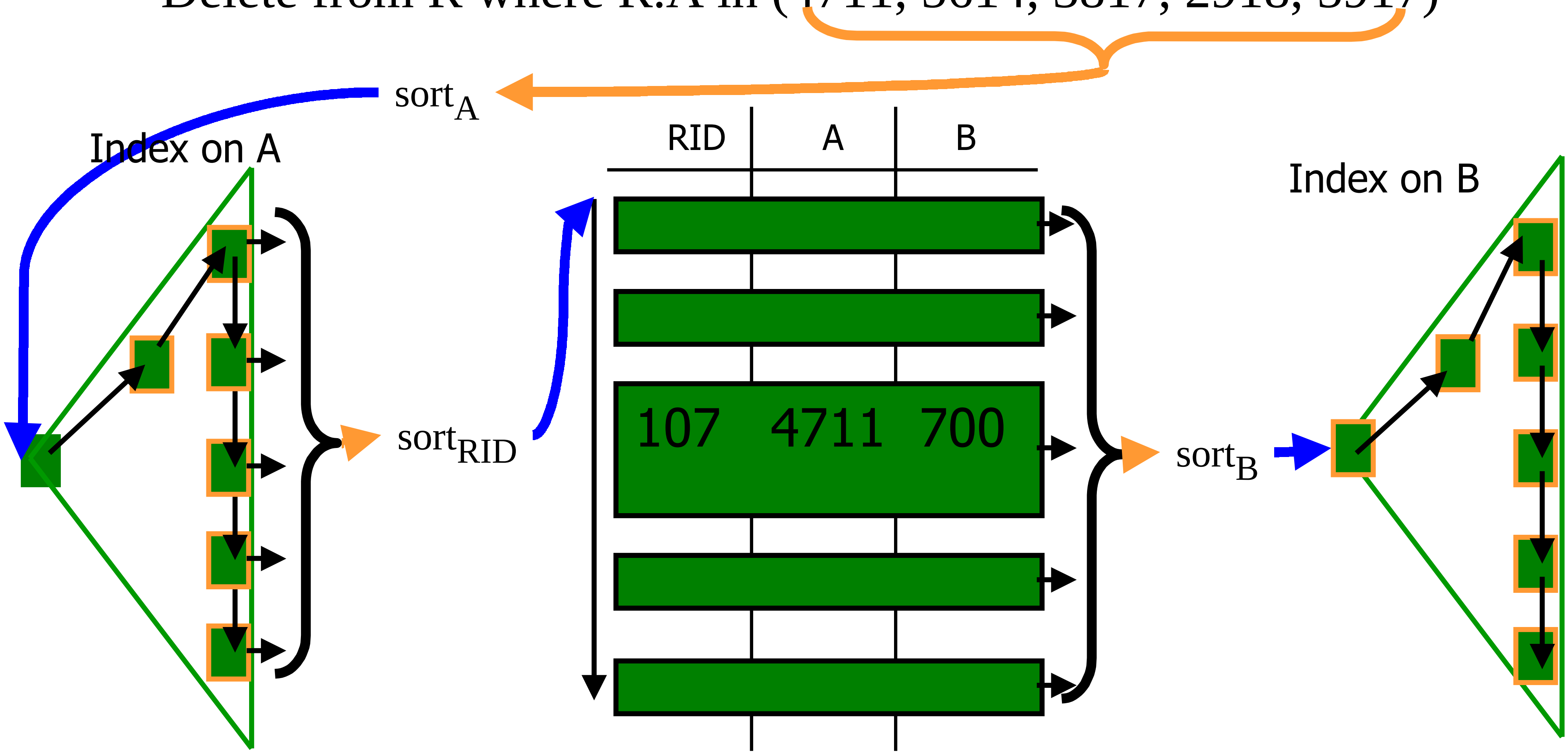
Implementing the new Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)



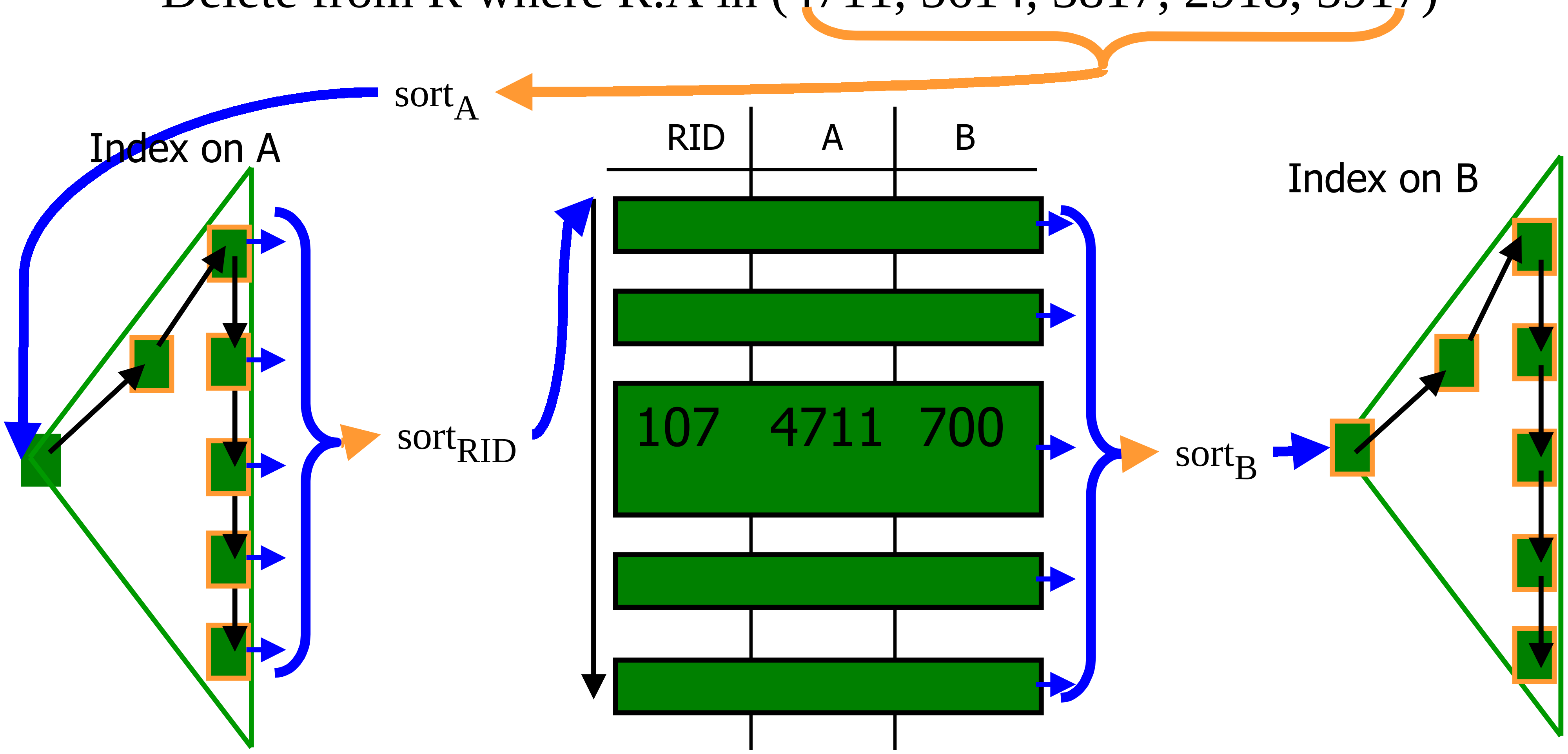
Implementing the new Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)



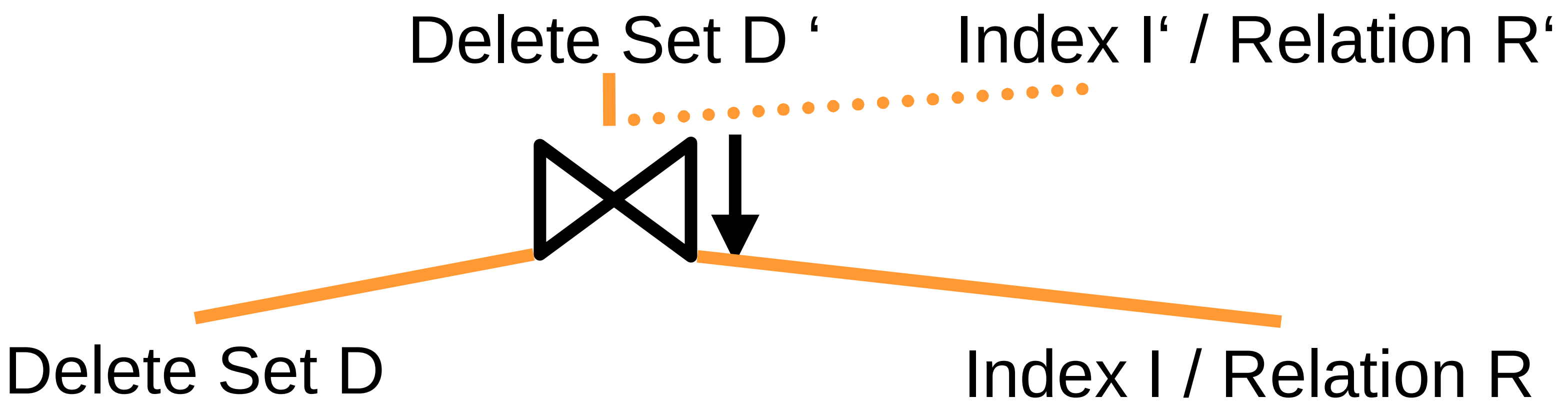
Implementing the new Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)



Reuse of traditional techniques

- Join operator → bulk delete operator:



Delete Set D ‘

Index I‘ / Relation R‘

Delete Set D

Index I / Relation R

- generating bulk delete evaluation plans using exisiting optimizers
  - ⋈↓ method
  - ⋈↓ order
  - primary ⋈↓ predicate

- different join techniques, but one argument in place

Example

- Delete from R where R.A in D(A)
- Schema:

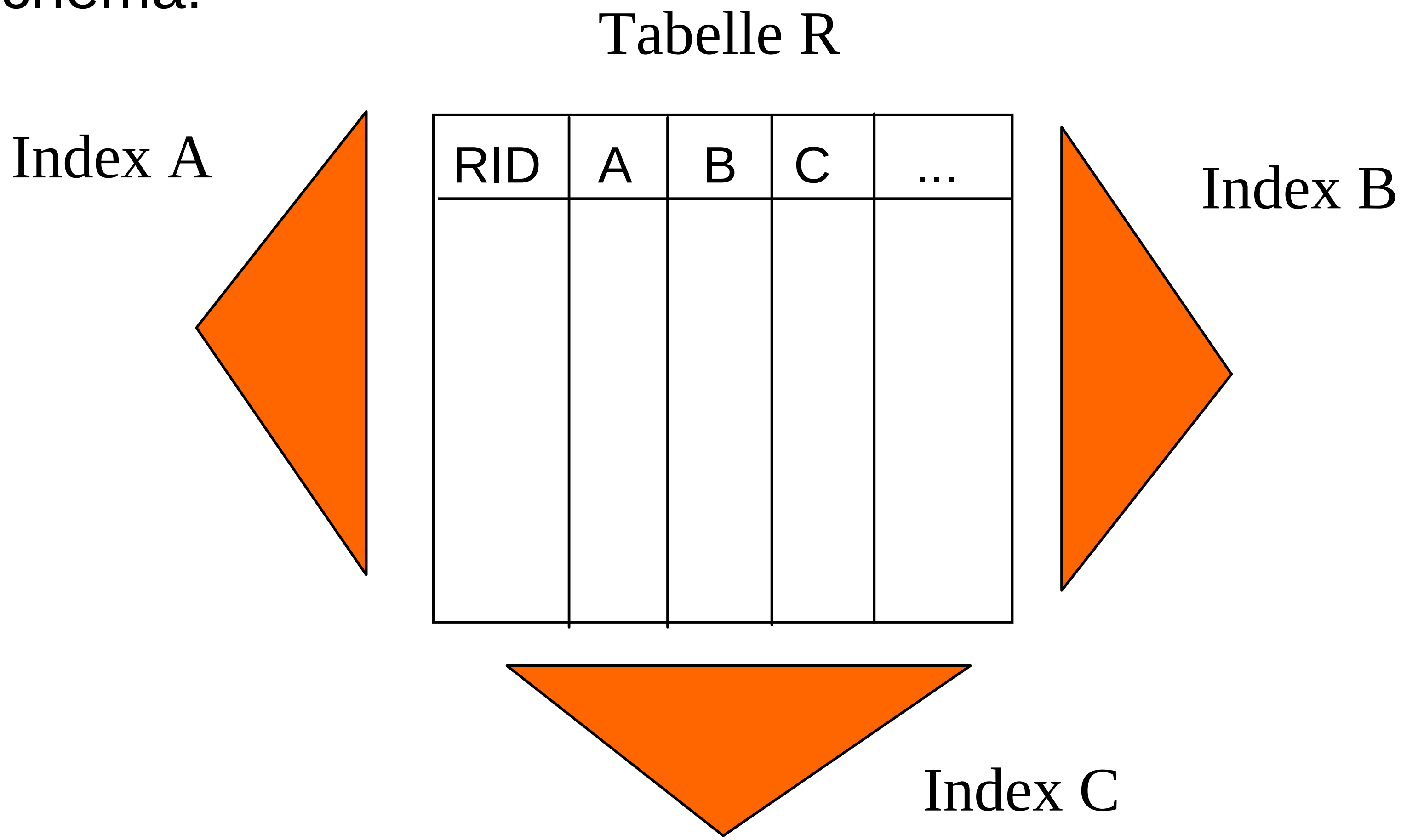


Tabelle R

Index A

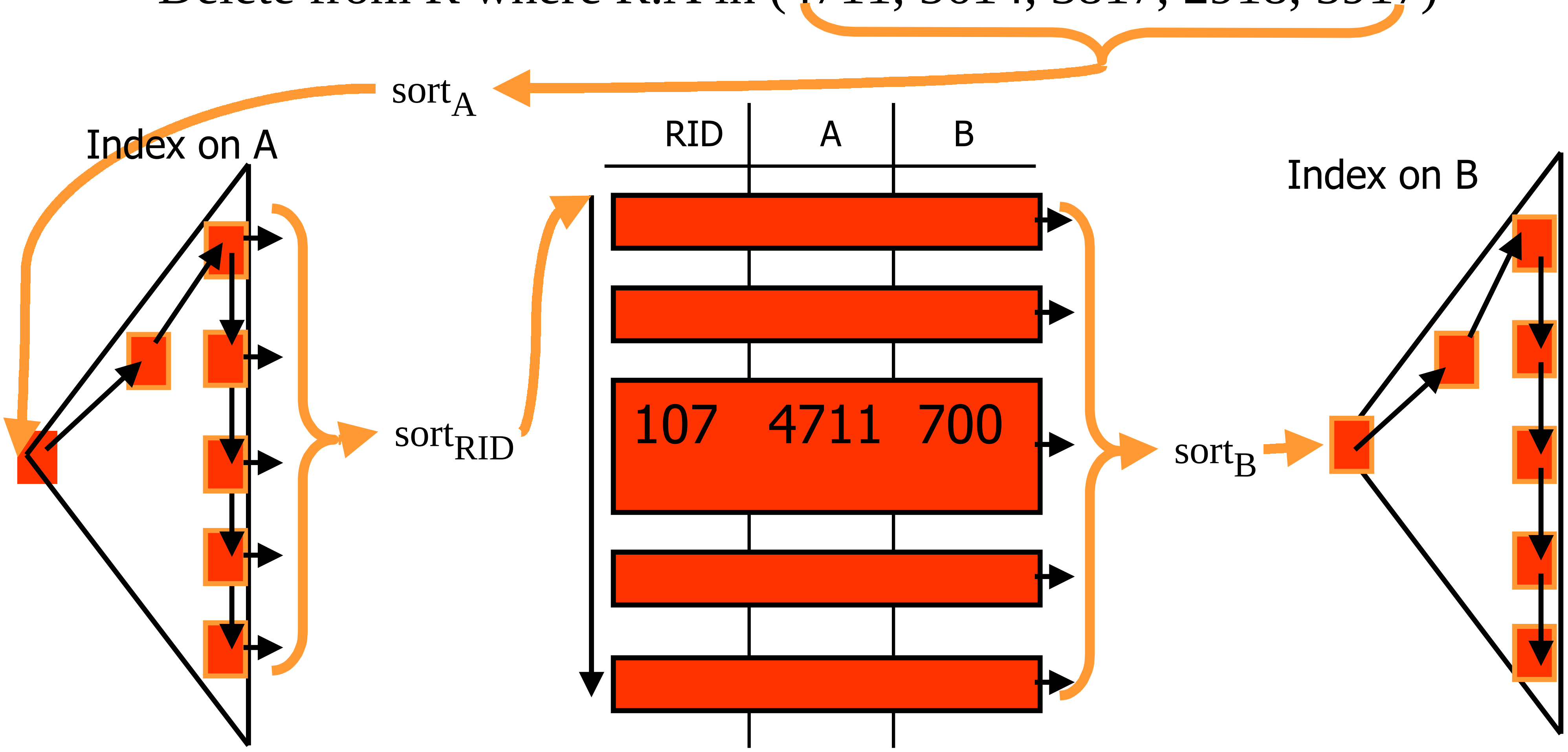
RID	A	B	C	...

Index B

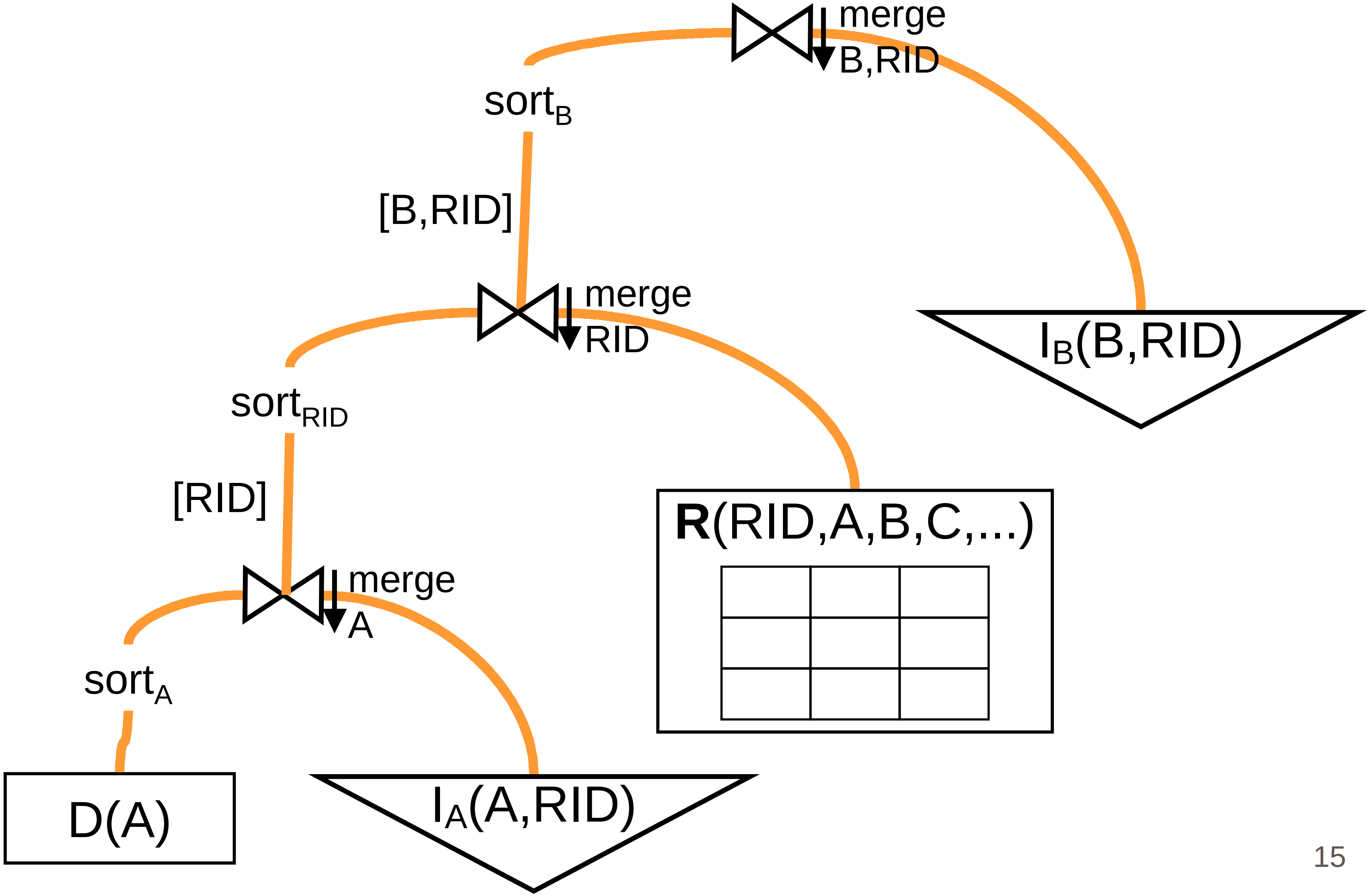
Index C

New Set-Oriented Vertical Approach

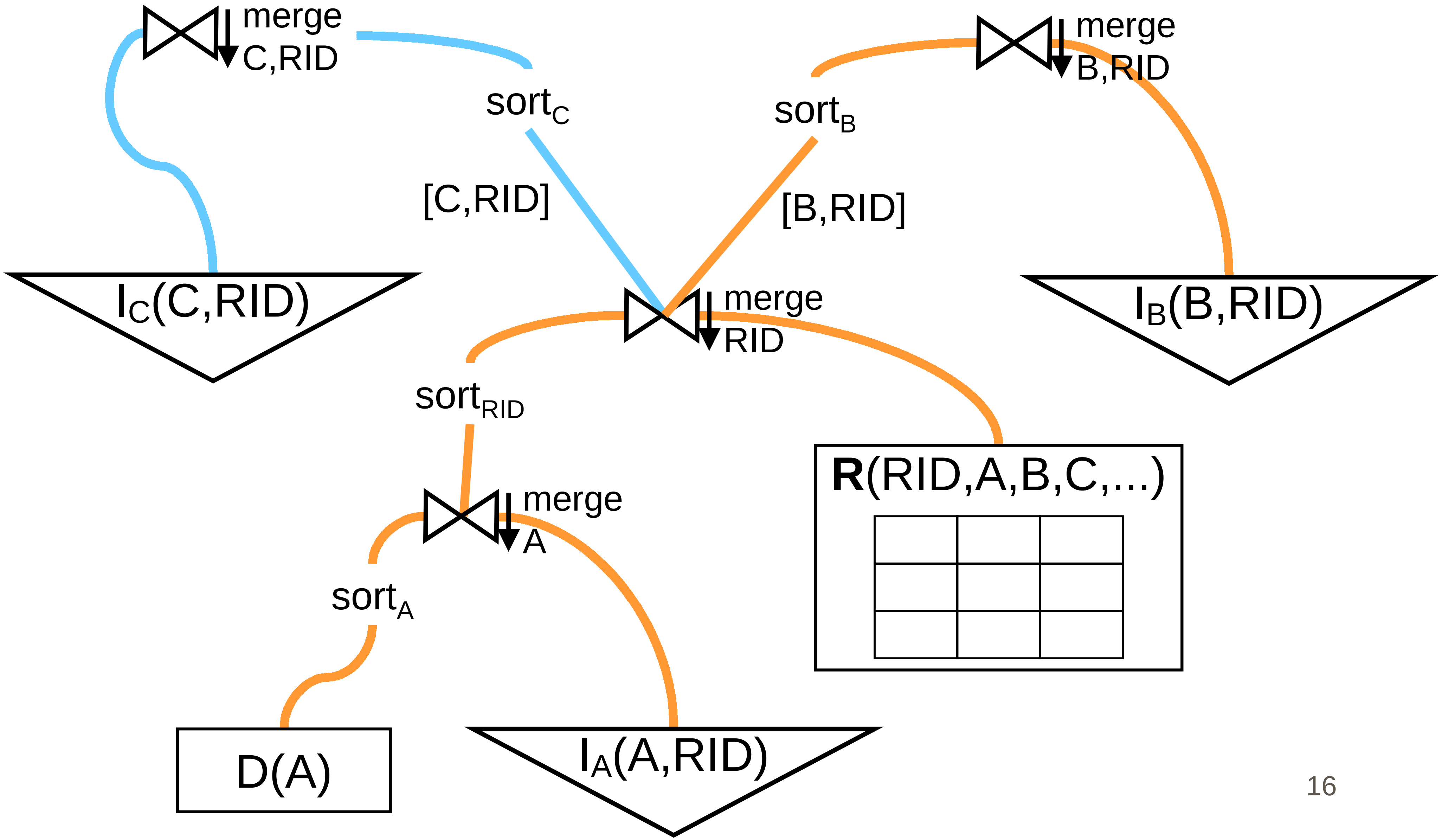
Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)



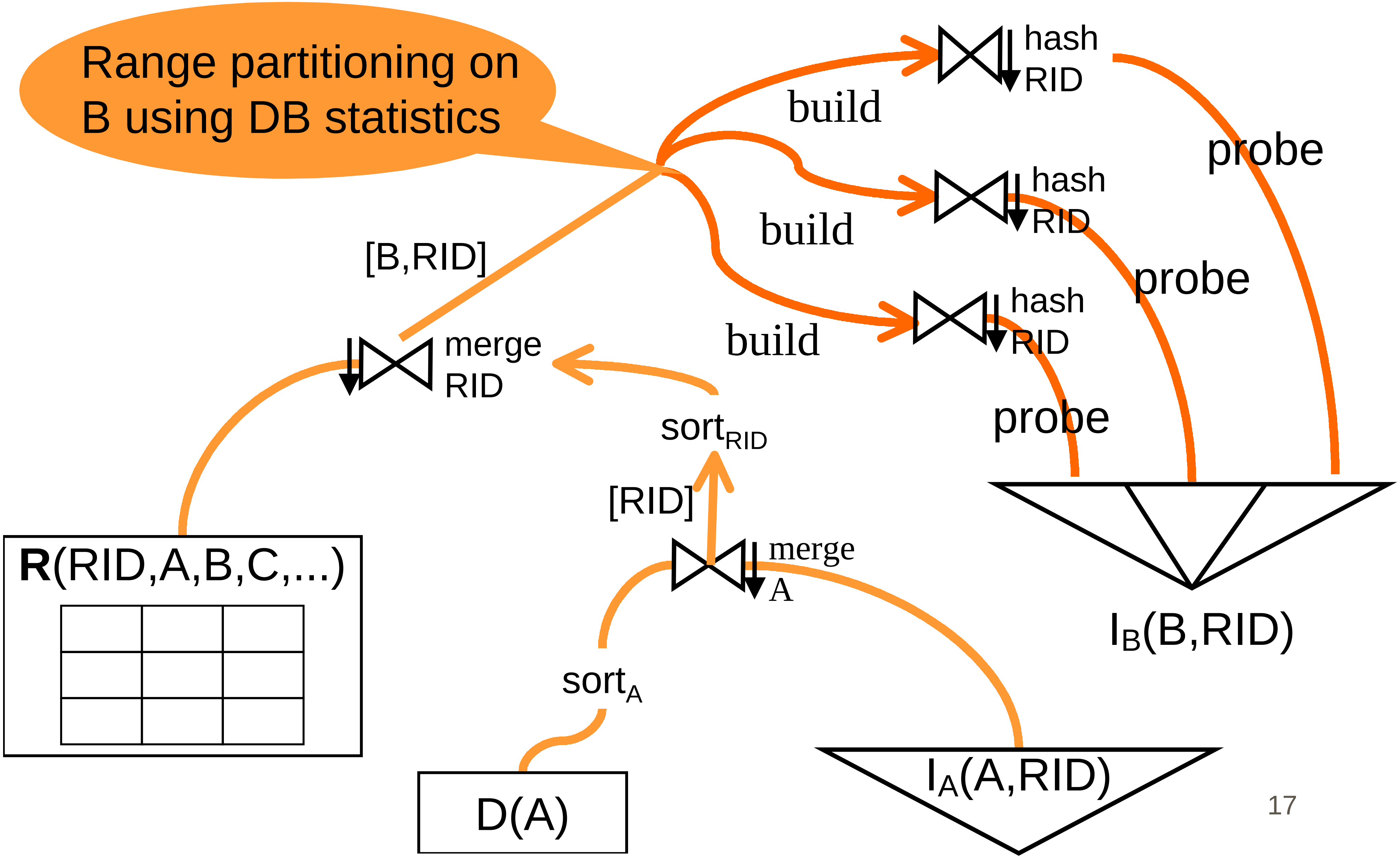
Query Evaluation Plans: Retrieve Index Keys B from R



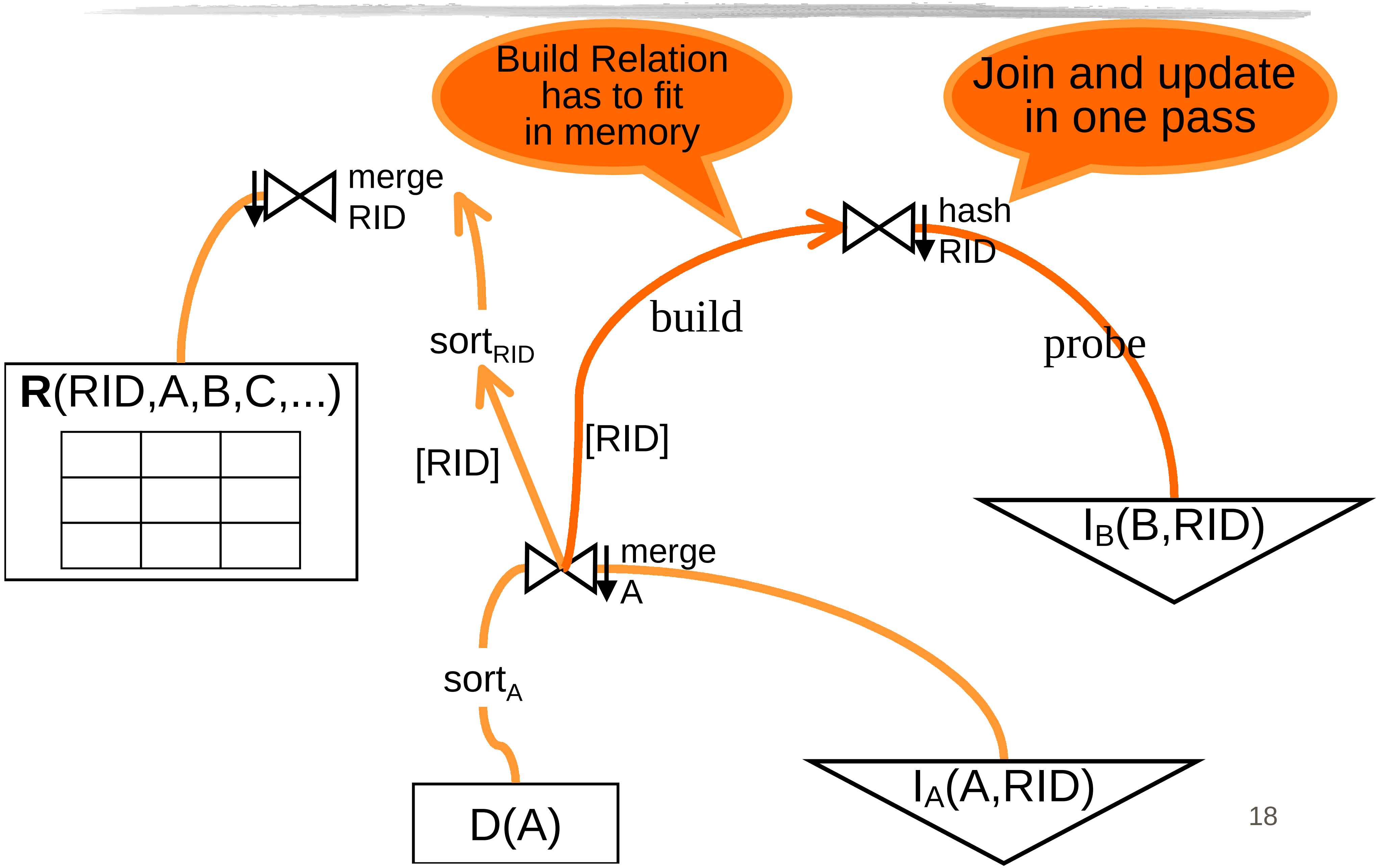
Query Evaluation Plans: Retrieve Index Keys B, C from R



Query Evaluation Plans: RID-Join using range partitioning



Query Evaluation Plans: RID-Join on Indices



Concurrency Control

Concurrency Control

- Take table R and all indices off-line (avoid lock escalation)

Concurrency Control

- Take table R and all indices off-line (avoid lock escalation)
- Process R and all indices with unique constraints

Concurrency Control

- Take table R and all indices off-line (avoid lock escalation)

- Process R and all indices with unique constraints

- Bring R and indices with unique constraints on-line

Concurrency Control

- Take table R and all indices off-line (avoid lock escalation)
- Process R and all indices with unique constraints
- Bring R and indices with unique constraints on-line
- Remaining indices can be processed concurrently using sidefiles/direct propagation

  (C.Mohan and I.Narang. Algorithms for creating indexes for very large tables without quiescing updates. ACM SIGMOD 1992.)

Concurrency Control

- Take table R and all indices off-line (avoid lock escalation)
- Process R and all indices with unique constraints
- Bring R and indices with unique constraints on-line
- Remaining indices can be processed concurrently using sidefiles/direct propagation

  (C.Mohan and I.Narang. Algorithms for creating indexes for very large tables without quiescing updates. ACM SIGMOD 1992.)
- Checkpoints: save high water mark

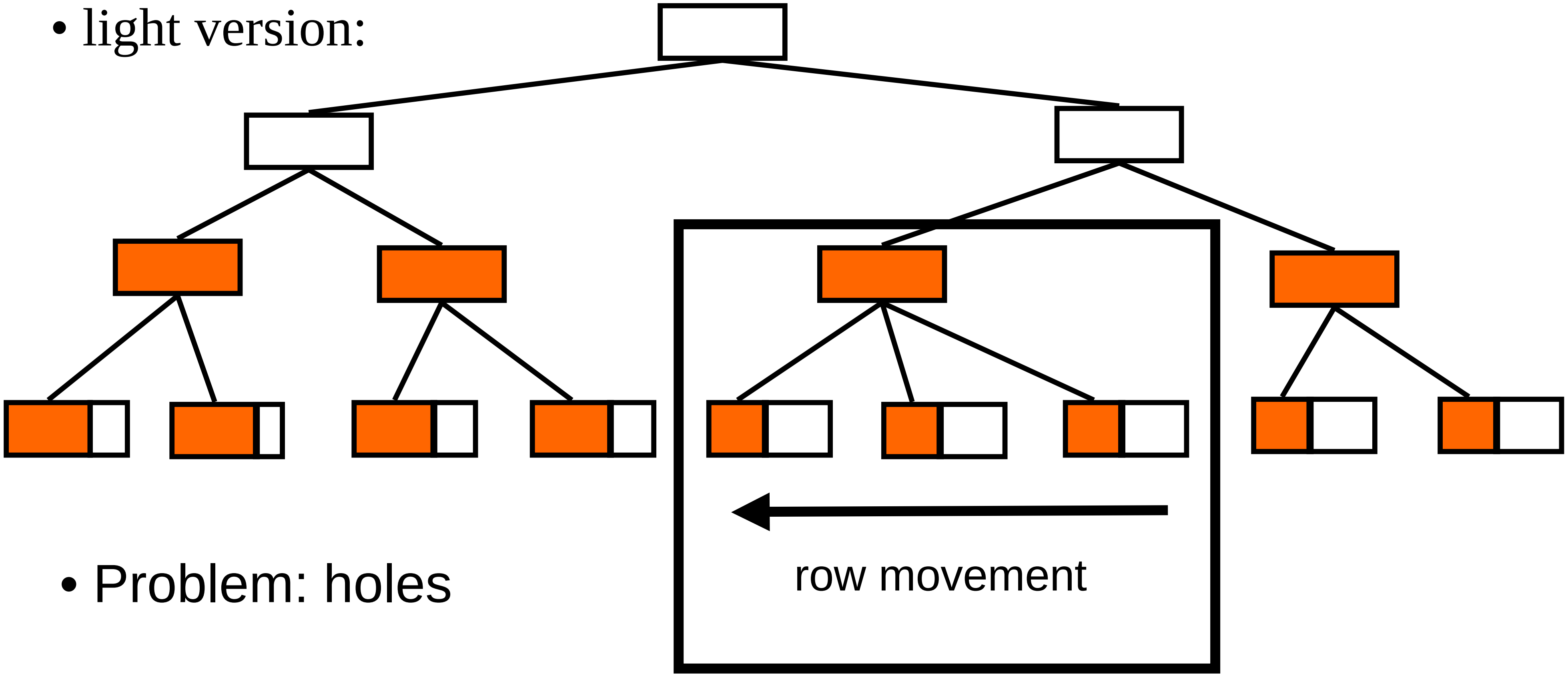
Concurrency Control

- Take table R and all indices off-line (avoid lock escalation)
- Process R and all indices with unique constraints
- Bring R and indices with unique constraints on-line
- Remaining indices can be processed concurrently using sidefiles/direct propagation

  (C.Mohan and I.Narang. Algorithms for creating indexes for very large tables without quiescing updates. ACM SIGMOD 1992.)
- Checkpoints: save high water mark
- Recovery: roll forward to save work already done

Reorganisation

- C.Zou and B.Salzberg. **On-line reorganisation of sparsely-populated B$^+$- Trees**. ACM SIGMOD 1996.
- light version:



- Problem: holes

Benchmark Environment

- Prototype database
- B^+-Trees based on Jan Janninks implementation
- Table R: 1.000.000 records (500 MB)
- Delete set D
- One index I_A on attribute A
- Statement:

  delete from R where R.A in (select D.A from D)
- sort merge evaluation plan used

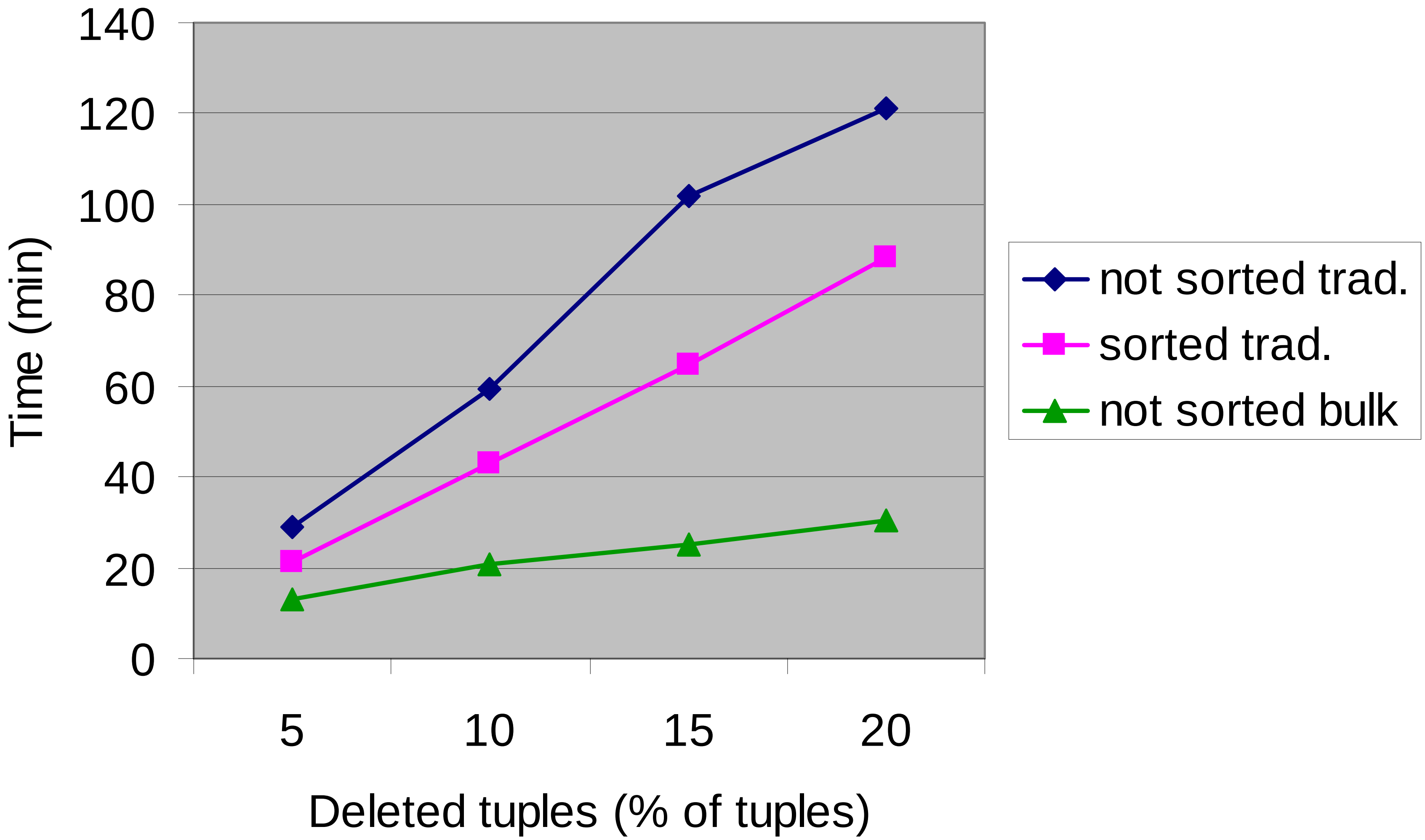
Tests

- Vary number of deleted records
- Vary number of indices
- Vary height of indices
- Vary size of memory
- Clustered table

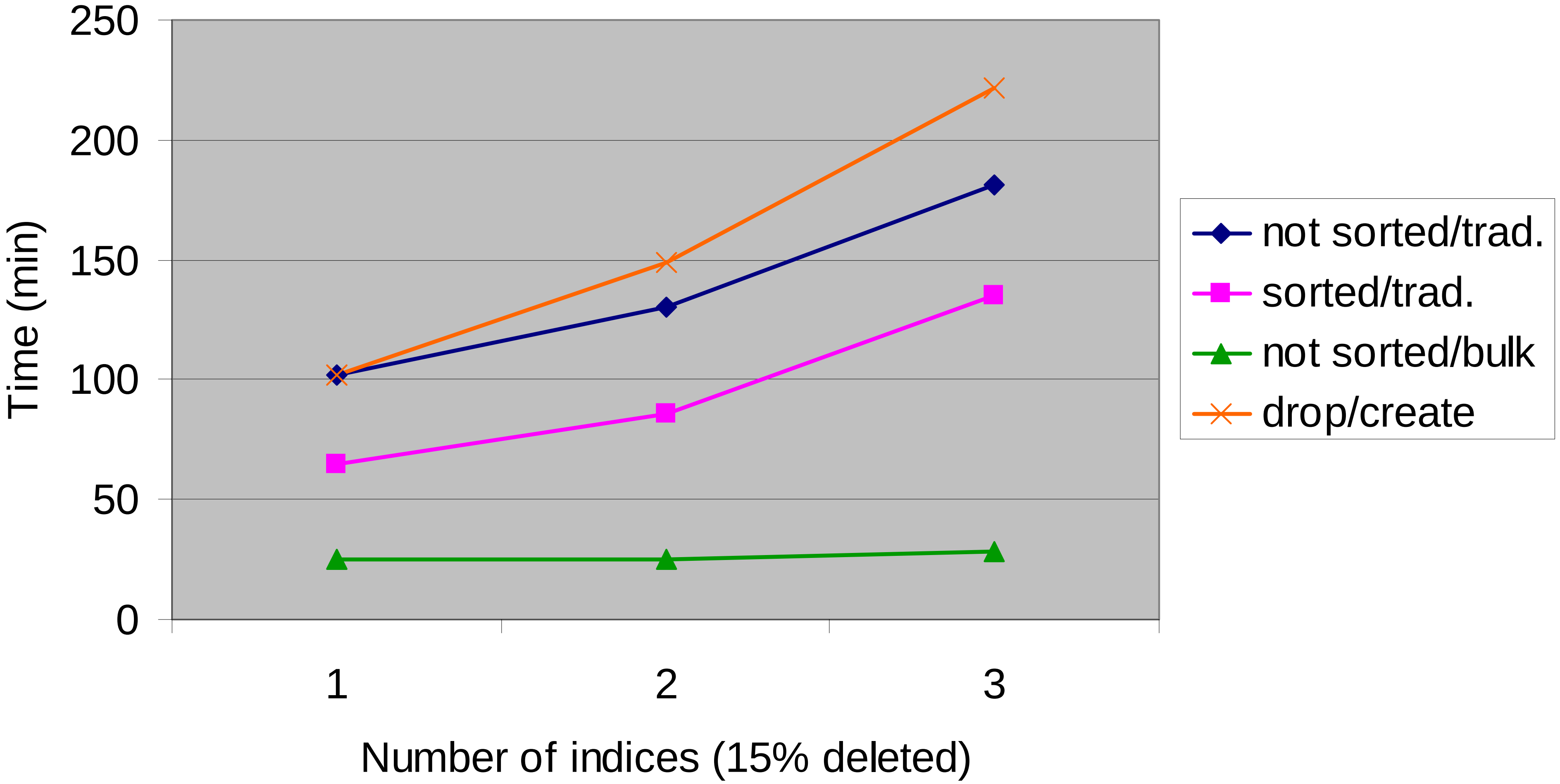
Diagrams:

• not sorted/trad:	Delete set not sorted, traditional approach
• sorted/trad	Delete set sorted, traditional approach
• not sorted/bulk	Delete set not sorted, vertical approach

Vary Number of Deleted Records



Vary Number of Indices

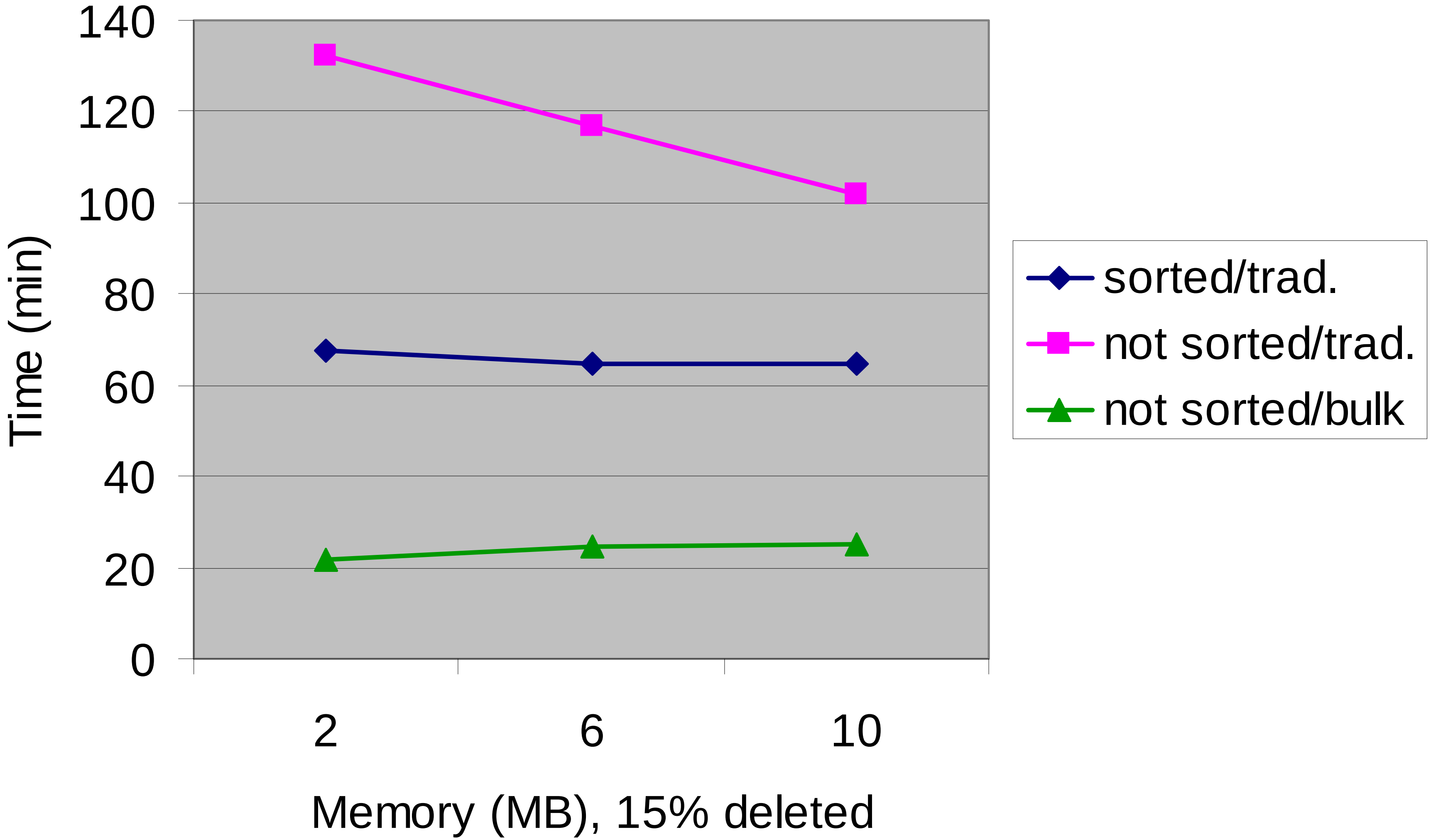


Vary Height of Indices

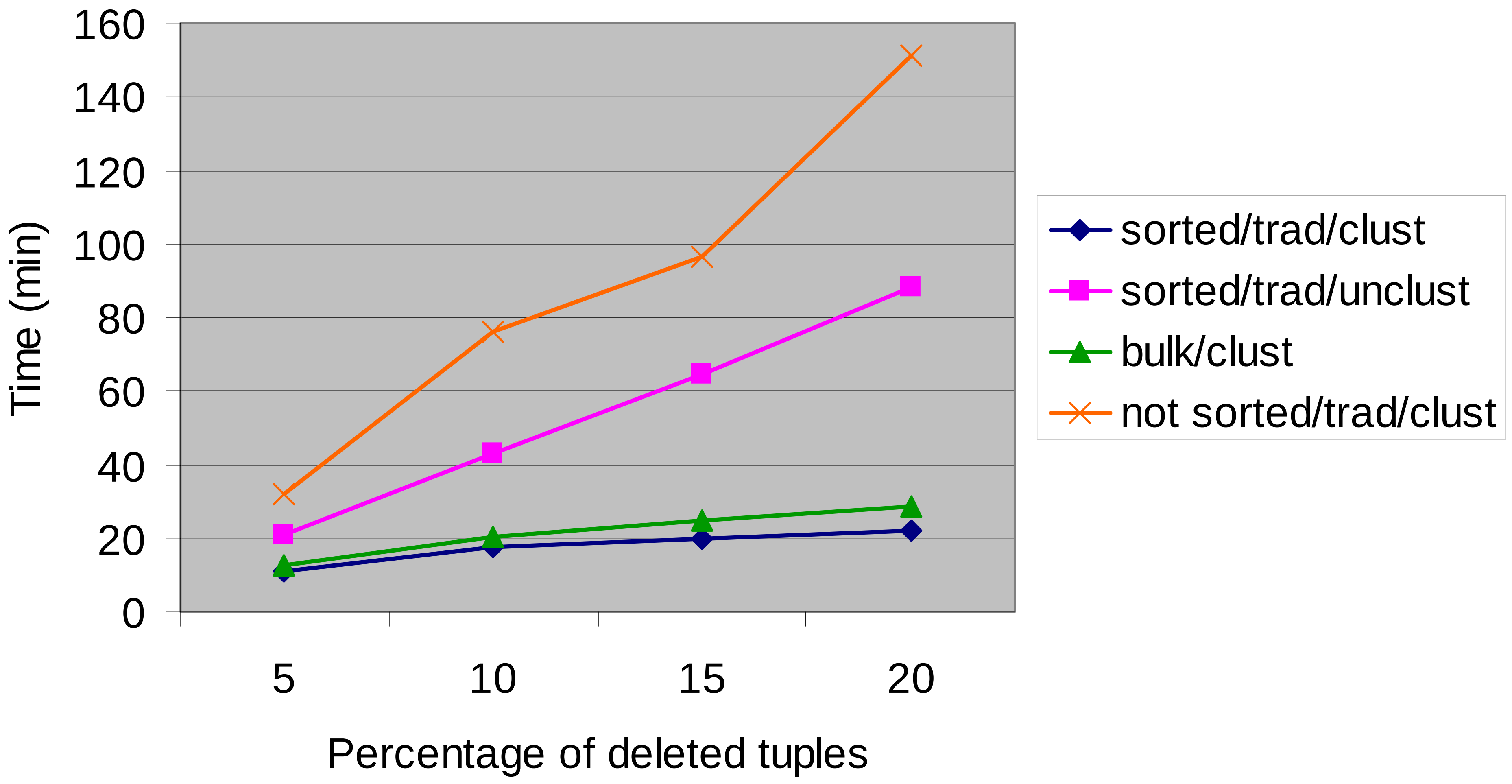
15% deleted

	index height 3 (min)	index height 4 (min)
not sorted/bulk	25	27
sorted/trad	65	81
not sorted/trad	102	136

Vary Size of Memory



Clustered Table



Conclusion

- New set-oriented vertical approach
- Implementation using evaluation plans
    - bulk delete operator
    - retrieve index keys from R
    - RID join using hashing
- Concurrency control and reorganisation
- Benchmark results
  → new approach outperforms trad. approach
- Future Work: generalize the approach to hash tables,
  R- trees, grid files, bitmap indices