

Advanced Topics in Data Management

C-Store, PAX

Review: Page Formats

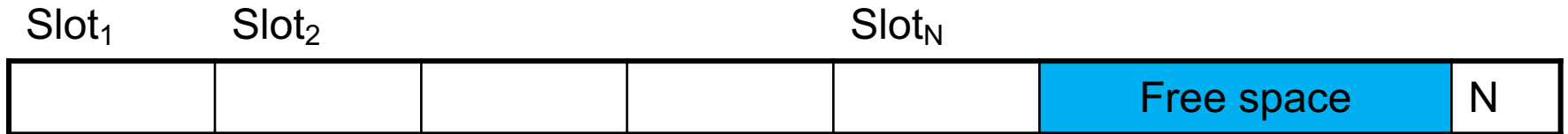
- 1 page = 1 disk block = fixed size (e.g. 8KB)
- Records:
 - Fixed length
 - Variable length
- Record id = RID
 - Typically, RID = (PageID, SlotNumber)

Page Format: Fixed-length Records

Fixed-length records: packed representation

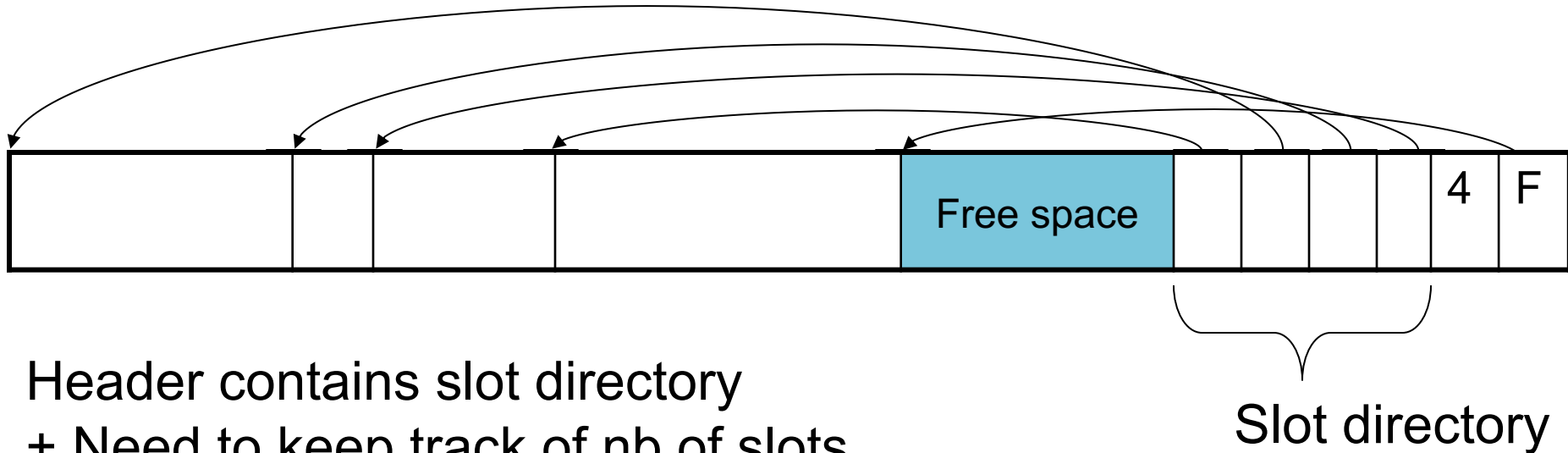
Divide page into **slots**. Each slot can hold one tuple

Record ID (RID) for each tuple is (PageID, SlotNb)



Number of records

Page Format: Variable-length Records



Header contains slot directory

+ Need to keep track of nb of slots

+ Also need to keep track of free space (F)

Record Formats: Fixed-length Fields

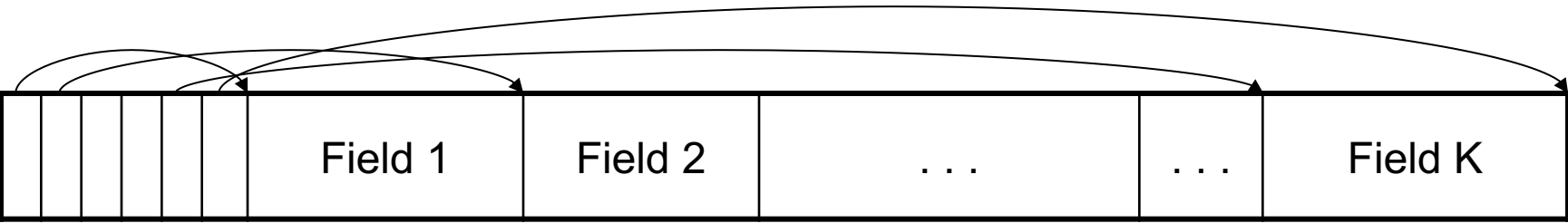
Fixed-length records => Each field has a fixed length (i.e., it has the same length in all the records)



Information about field lengths and types is in the catalog

Record Formats: Variable-length Fields

Variable length records



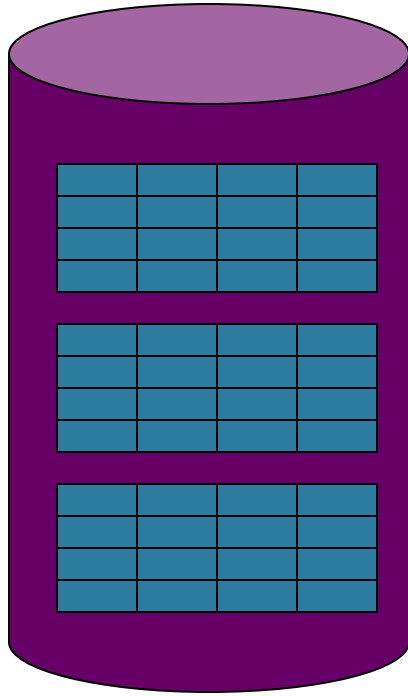
Record header

Remark: NULLS require no space at all (why ?)

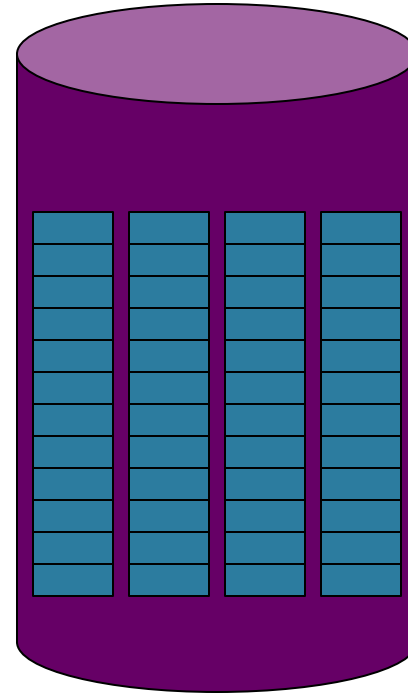
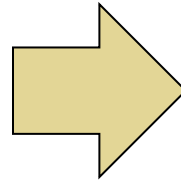
Column-Store

- Store an entire attribute in a different file
- While the idea had been around, even before PAX, but getting all the details right in order to extract the extra performance took a long time

From Row-Store to Column-Store



Rows stored
contiguously on disk
(+ tuple headers)



Columns stored
contiguously on disk
(no headers needed)

C-Store Illustration

Row-based
(4 pages)

Column-based
(4 pages)

Page

Page	A	1
	A	2
	A	2
	A	2
	B	2
	B	4
	C	4
	C	4

A	1
A	2
A	2
A	2
B	2
B	4
C	4
C	4

Page

C-Store also
avoids large
tuple headers

Column-Oriented Databases

- Main idea:
 - **Physical storage**: complete vertical partition; each column stored separately: R.A, R.B, R.A
 - **Logical schema**: remains the same R(A,B,C)
- Main advantage:
 - **Improved transfer rate**: disk to memory, memory to CPU, better cache locality

Basic Trade-Off

- **Row stores**
 - Quick to update entire tuple (1 page IO)
 - Quick to access a single tuple
- **Column stores**
 - Avoid reading unnecessary columns
 - Better compression
- **Entire system needs a different design**
 - Not only storage manager
 - To achieve high performance

From Row to Column Storage (Modern Designs)

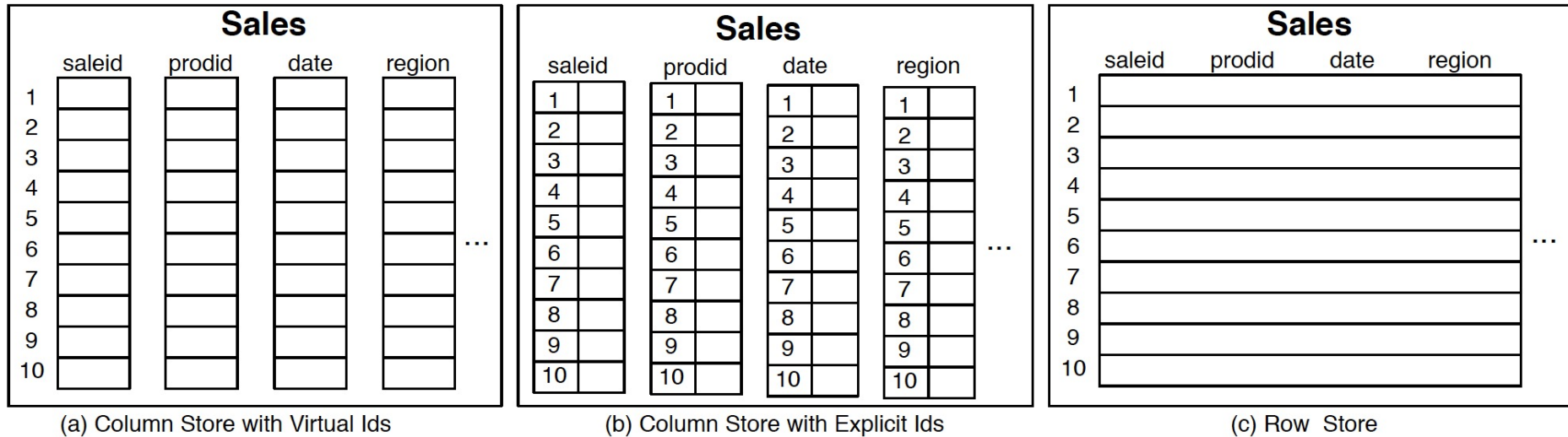


Figure 1.1: Physical layout of column-oriented vs row-oriented databases.

Basic tradeoffs:

- Reading all attributes of one records, v.s.
- Reading some attributes of many records

Fig. 1.2

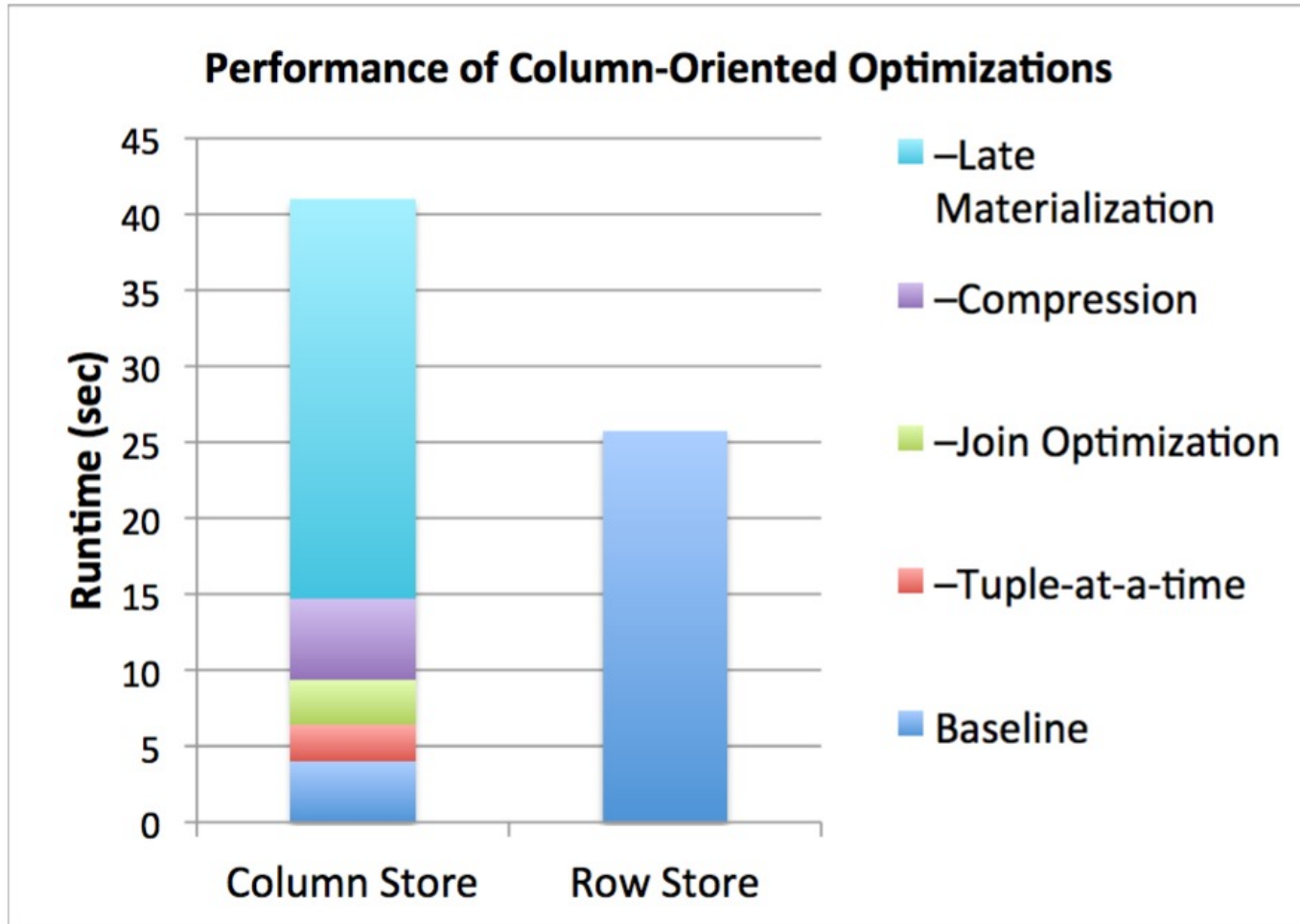


Figure 1.2: Performance of C-Store versus a commercial database system on the SSBM benchmark, with different column-oriented optimizations enabled.

Key Architectural Trends (Sec.1)

- Virtual IDs
- Block-oriented and vertical processing
- Late materialization
- Column-specific compression

Key Architectural Trends (Sec.1)

- Virtual IDs
 - Offsets (arrays) instead of keys
- Block-oriented and vertical processing
 - Iterator model: one tuple → one block of tuples
- Late materialization
 - Postpone tuple reconstruction in query plan
- Column-specific compression
 - Much better than row-compression (why?)

Vectorized Processing

Review:

- Volcano-style iterator model
 - Next() method
 - Pipelining
- Materialization of all intermediate results
- Discuss in class:

```
select avg(A) from R where A < 100
```


Vectorized Processing

- Vectorized processing:
 - Next() returns a block of tuples (e.g. N=1000) instead of single tuple
- Pros:
 - No more large intermediate results
 - Tight inner loop for selection and/or avg
- Discuss in class:

```
select avg(A) from R where A < 100
```

Compression (Sec. 4)

- What is the advantage of compression in databases?
- Discuss main column-at-a-time compression techniques

Compression (Sec. 4)

- What is the advantage of compression in databases?
- Discuss main column-at-a-time compression techniques
 - Row-length encoding: F,F,F,F,M,M \rightarrow 4F,2M
 - Bit-vector (see also bit-map indexes)
 - Dictionary. More generally: Ziv-Lempel

Compression (Sec. 4)

Row-based
(4 pages)

Column-based
(4 pages)

Compressed
(2 pages)

Page

Page {	A	1
	A	2
	A	2
	A	2
	B	2
	B	4
	C	4
	C	4

A	1
A	2
A	2
A	2
B	2
B	4
C	4
C	4

4XA	1X1
2XB	4X2
2XC	5X4

Page

Late Materialization (Sec. 4)

- What is it?
- Discuss $\Pi_B(\sigma_{A='a' \wedge D='d'}(R(A,B,C,D,\dots)))$

Late Materialization (Sec. 4)

- What is it?
- Discuss $\Pi_B(\sigma_{A='a' \wedge D='d'}(R(A,B,C,D,\dots)))$
- Early materialization:
 - Retrieve positions with 'a' in column A: 2, 4, 5, 9, 25...

Late Materialization (Sec. 4)

- What is it?
- Discuss $\Pi_B(\sigma_{A='a' \wedge D='d'}(R(A,B,C,D,\dots)))$
- Early materialization:
 - Retrieve positions with 'a' in column A: 2, 4, 5, 9, 25...
 - Retrieve those values in column D: 'x', 'd', 'y', 'd', 'd',...

Late Materialization (Sec. 4)

- What is it?
- Discuss $\Pi_B(\sigma_{A='a' \wedge D='d'}(R(A,B,C,D,\dots)))$
- Early materialization:
 - Retrieve positions with 'a' in column A: 2, 4, 5, 9, 25...
 - Retrieve those values in column D: 'x', 'd', 'y', 'd', 'd',...
 - Retain only positions with 'd': 4, 9, ...

Late Materialization (Sec. 4)

- What is it?
- Discuss $\Pi_B(\sigma_{A='a' \wedge D='d'}(R(A,B,C,D,\dots)))$
- Early materialization:
 - Retrieve positions with 'a' in column A: 2, 4, 5, 9, 25...
 - Retrieve those values in column D: 'x', 'd', 'y', 'd', 'd',...
 - Retain only positions with 'd': 4, 9, ...
 - Lookup values in column B: B[4], B[9], ...

Late Materialization (Sec. 4)

- What is it?
- Discuss $\Pi_B(\sigma_{A='a' \wedge D='d'}(R(A,B,C,D,\dots)))$
- Early materialization:
 - Retrieve positions with 'a' in column A: 2, 4, 5, 9, 25...
 - Retrieve those values in column D: 'x', 'd', 'y', 'd', 'd',...
 - Retain only positions with 'd': 4, 9, ...
 - Lookup values in column B: B[4], B[9], ...
- Late materialization
 - Retrieve positions with 'a' in column A: 2, 4, 5, 9, 25...

Late Materialization (Sec. 4)

- What is it?
- Discuss $\Pi_B(\sigma_{A='a' \wedge D='d'}(R(A,B,C,D,\dots)))$
- Early materialization:
 - Retrieve positions with 'a' in column A: 2, 4, 5, 9, 25...
 - Retrieve those values in column D: 'x', 'd', 'y', 'd', 'd',...
 - Retain only positions with 'd': 4, 9, ...
 - Lookup values in column B: B[4], B[9], ...
- Late materialization
 - Retrieve positions with 'a' in column A: 2, 4, 5, 9, 25...
 - Retrieve positions with 'd' in column D: 3, 4, 7, 9, 12,...

Late Materialization (Sec. 4)

- What is it?
- Discuss $\Pi_B(\sigma_{A='a' \wedge D='d'}(R(A,B,C,D,\dots)))$
- Early materialization:
 - Retrieve positions with 'a' in column A: 2, 4, 5, 9, 25...
 - Retrieve those values in column D: 'x', 'd', 'y', 'd', 'd',...
 - Retain only positions with 'd': 4, 9, ...
 - Lookup values in column B: B[4], B[9], ...
- Late materialization
 - Retrieve positions with 'a' in column A: 2, 4, 5, 9, 25...
 - Retrieve positions with 'd' in column D: 3, 4, 7, 9, 12,...
 - Intersect: 4, 9, ...

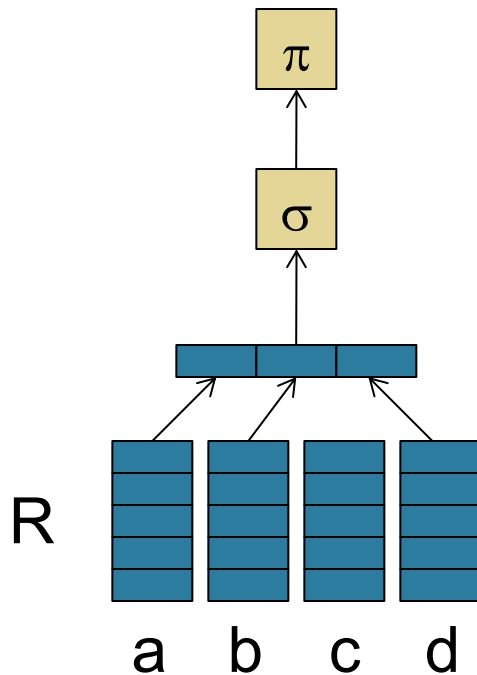
Late Materialization (Sec. 4)

- What is it?
- Discuss $\Pi_B(\sigma_{A='a' \wedge D='d'}(R(A,B,C,D,\dots)))$
- Early materialization:
 - Retrieve positions with 'a' in column A: 2, 4, 5, 9, 25...
 - Retrieve those values in column D: 'x', 'd', 'y', 'd', 'd',...
 - Retain only positions with 'd': 4, 9, ...
 - Lookup values in column B: B[4], B[9], ...
- Late materialization
 - Retrieve positions with 'a' in column A: 2, 4, 5, 9, 25...
 - Retrieve positions with 'd' in column D: 3, 4, 7, 9, 12,...
 - Intersect: 4, 9, ...
 - Lookup values in column B: B[4], B[9], ...

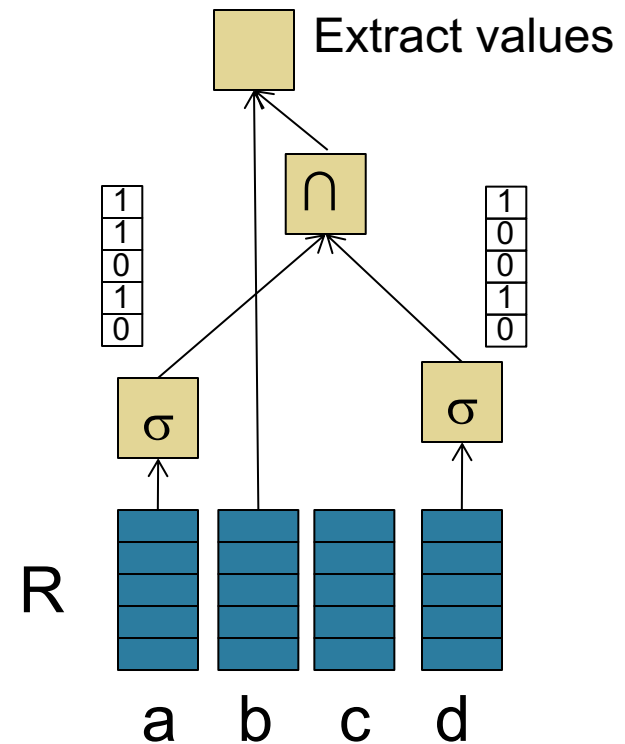
Late Materialization (Sec. 4)

Ex: `SELECT R.b from R where R.a=X and R.d=Y`

Early materialization



Late materialization



Jive Join (Sec. 4)

```
SELECT emp.age, dept.name  
FROM emp, dept  
WHERE emp.dept_id = dept.id
```

emp.dept_id

dept.id

42
36
42
44
38

 ⋈

38
42
46
36

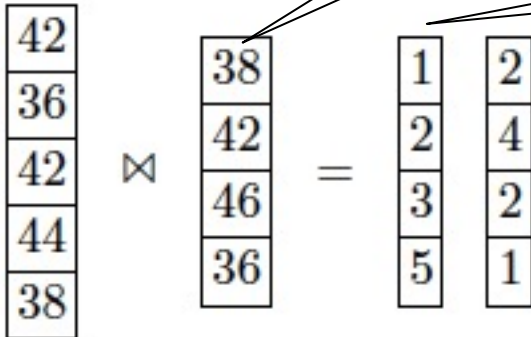
Jive Join (Sec. 4)

```
SELECT emp.age, dept.name  
FROM emp, dept  
WHERE emp.dept_id = dept.id
```

emp.dept_id

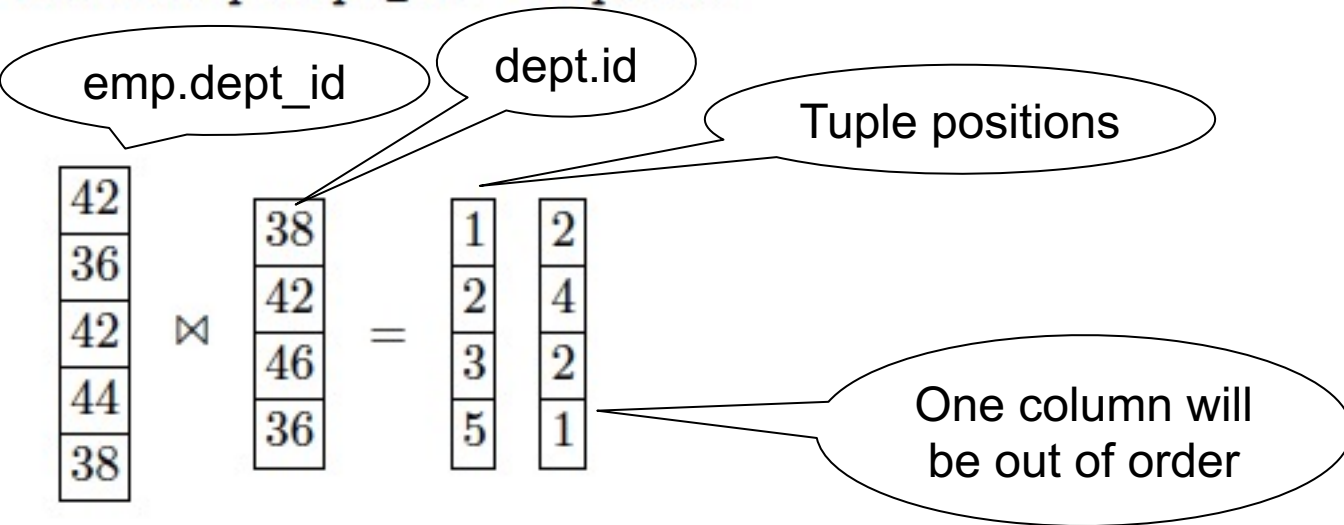
dept.id

Tuple positions



Jive Join (Sec. 4)

```
SELECT emp.age, dept.name  
FROM emp, dept  
WHERE emp.dept_id = dept.id
```



Jive Join (Sec. 4)

```
SELECT emp.age, dept.name  
FROM emp, dept  
WHERE emp.dept_id = dept.id
```

emp.dept_id

dept.id

Tuple positions

42
36
42
44
38

⋈

38
42
46
36

=

1
2
3
5

2
4
2
1

One column will
be out of order

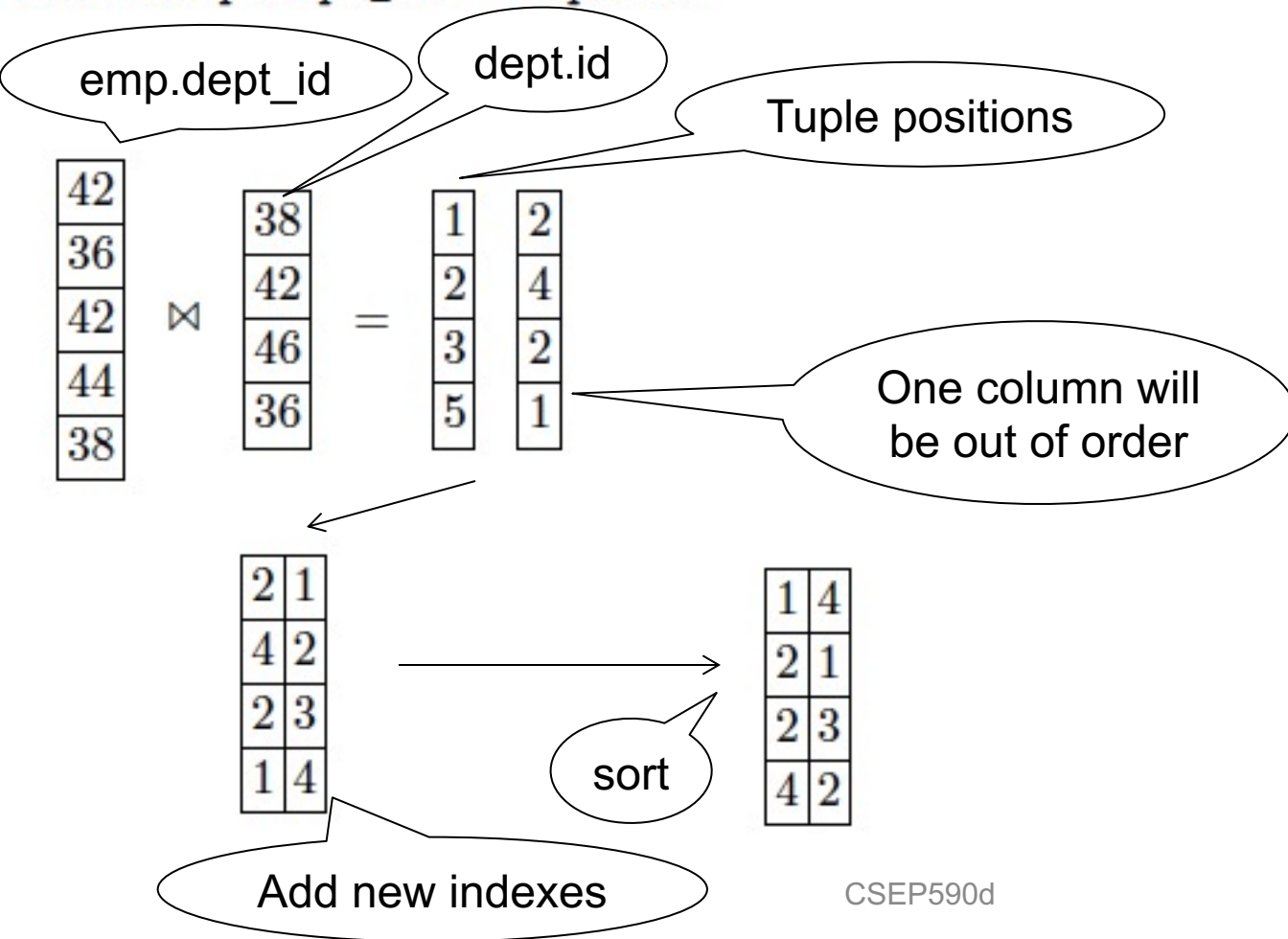
←

2	1
4	2
2	3
1	4

Add new indexes

Jive Join (Sec. 4)

```
SELECT emp.age, dept.name  
FROM emp, dept  
WHERE emp.dept_id = dept.id
```



Jive Join (Sec. 4)

```
SELECT emp.age, dept.name
FROM emp, dept
WHERE emp.dept_id = dept.id
```



42
36
42
44
38

 \bowtie

38
42
46
36

1	2
2	4
3	2
5	1

One column will be out of order

dept.name????

2	1
4	2
2	3
1	4

sort

1	4
2	1
2	3
4	2

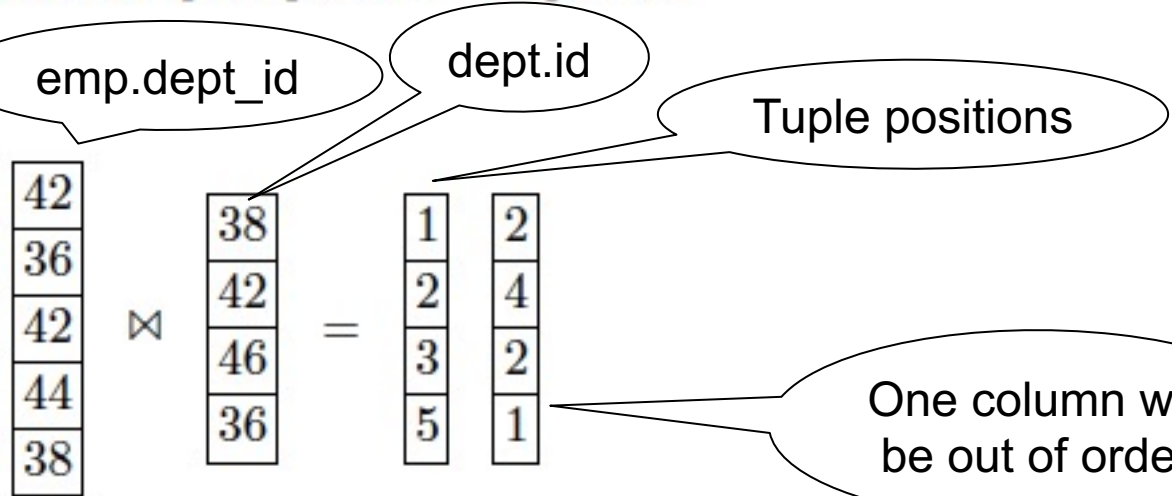
Fetch dept.name

1	4	Smith
2	1	Johnson
2	3	Johnson
4	2	Jones

Add new indexes

Jive Join (Sec. 4)

```
SELECT emp.age, dept.name
FROM emp, dept
WHERE emp.dept_id = dept.id
```



One column will be out of order

re-sort

2	1	Johnson
4	2	Jones
2	3	Johnson
1	4	Smith

dept.name????

2	1
4	2
2	3
1	4

1	4
2	1
2	3
4	2

1	4	Smith
2	1	Johnson
2	3	Johnson
4	2	Jones

sort

Fetch dept.name

Add new indexes

Late Materialization

```
select sum(R.a) from R, S
where R.c = S.b
  and 5 < R.a < 20 and 40 < R.b < 50
  and 30 < S.a < 40
```

Initial Status

Relation R			Relation S	
Ra	Rb	Rc	Sa	Sb
3	12	12	17	11
16	34	34	49	35
56	75	53	58	62
9	45	23	99	44
11	49	78	64	29
27	58	65	37	78
8	97	33	53	19
41	75	21	61	81
19	42	29	32	26
35	55	0	50	23

Late Materialization

```
select sum(R.a) from R, S
where R.c = S.b
and 5 < R.a < 20 and 40 < R.b < 50
and 30 < S.a < 40
```

select(Ra,5,20)

Ra **inter1**

3	2
16	4
56	5
9	7
11	9
27	
8	
41	
19	
35	

(1)

Late Materialization

```
select sum(R.a) from R, S
where R.c = S.b
and 5 < R.a < 20 and 40 < R.b < 50
and 30 < S.a < 40
```

select(Ra,5,20)

reconstruct(Rb,inter1)

Ra

inter1

inter1

Rb

inter2

3
16
56
9
11
27
8
41
19
35

2
4
5
7
9

2
4
5
7
9

12
34
75
45
49
58
97
75
42
55

2 34
4 45
5 49
7 97
9 42

(1)

(2)



Late Materialization

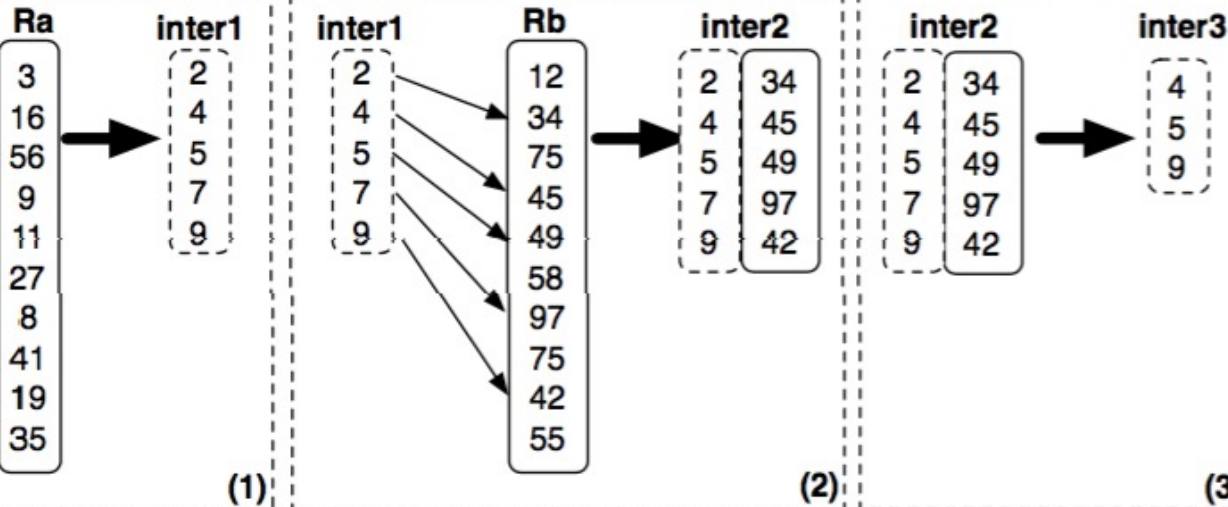
select sum(R.a) from R, S
where R.c = S.b
and 5 < R.a < 20 and 40 < R.b < 50
and 30 < S.a < 40

40,50

select(Ra,5,20)

reconstruct(Rb,inter1)

select(inter2,30,40)



Late Materialization

select sum(R.a) from R, S
 where R.c = S.b
 and 5 < R.a < 20 and 40 < R.b < 50
 and 30 < S.a < 40

40,50

select(Ra,5,20)

reconstruct(Rb,inter1)

select(inter2,30,40)

reconstruct(Rc,inter3)

Ra
 3
 16
 56
 9
 11
 27
 8
 41
 19
 35

inter1
 2
 4
 5
 7
 9

inter1
 2
 4
 5
 7
 9

Rb
 12
 34
 75
 45
 49
 58
 97
 75
 42
 55

inter2
 2 34
 4 45
 5 49
 7 97
 9 42

inter2
 2 34
 4 45
 5 49
 7 97
 9 42

inter3
 4
 5
 9

inter3
 4
 5
 9

Rc
 12
 34
 53
 23
 78
 65
 33
 21
 29
 0

join_input_R
 4 23
 5 78
 9 29

(1)

(2)

(3)

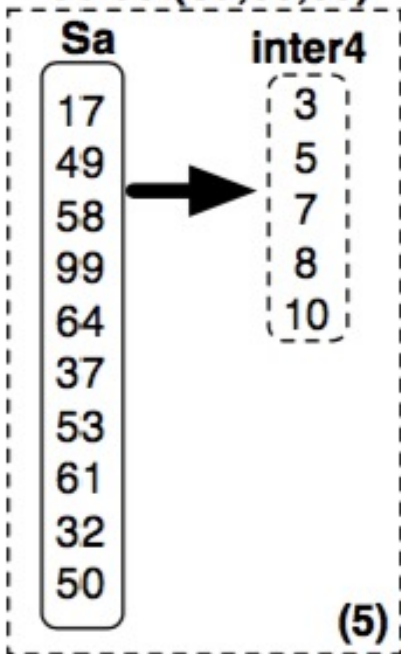
(4)

Late Materialization

```
select sum(R.a) from R, S
where R.c = S.b
and 5 < R.a < 20 and 40 < R.b < 50
and 30 < S.a < 40
```

???

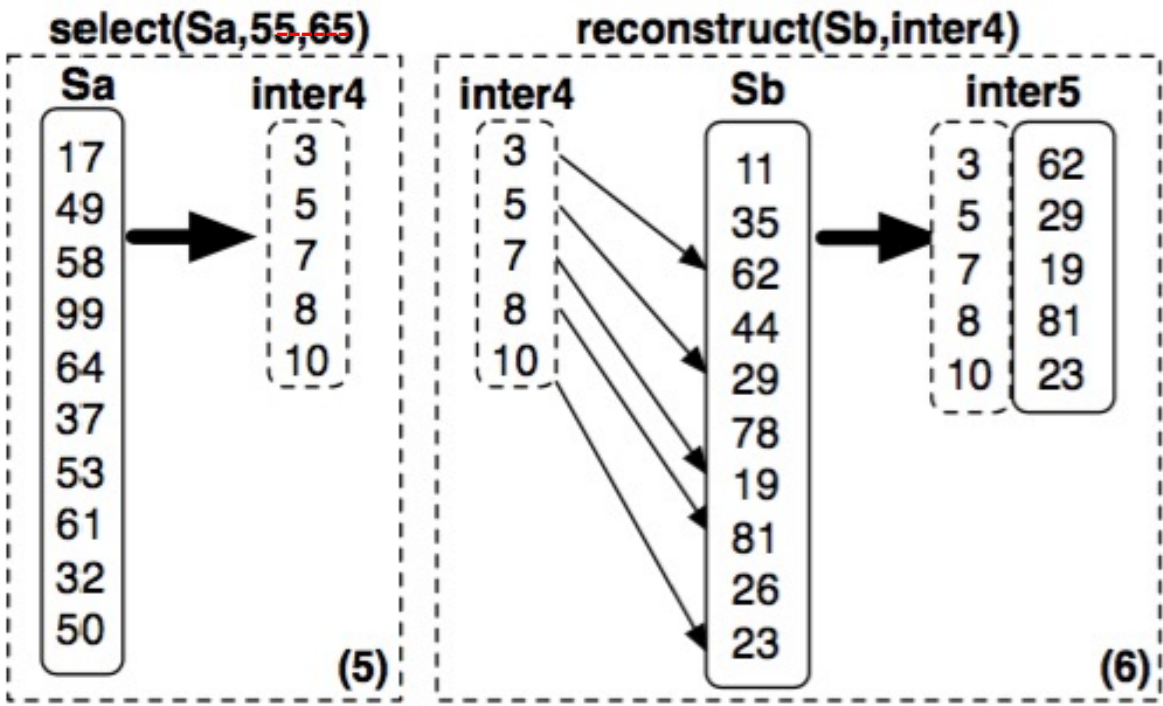
~~select(Sa, 55, 65)~~



Late Materialization

select sum(R.a) from R, S
where R.c = S.b
and 5 < R.a < 20 and 40 < R.b < 50
and 30 < S.a < 40

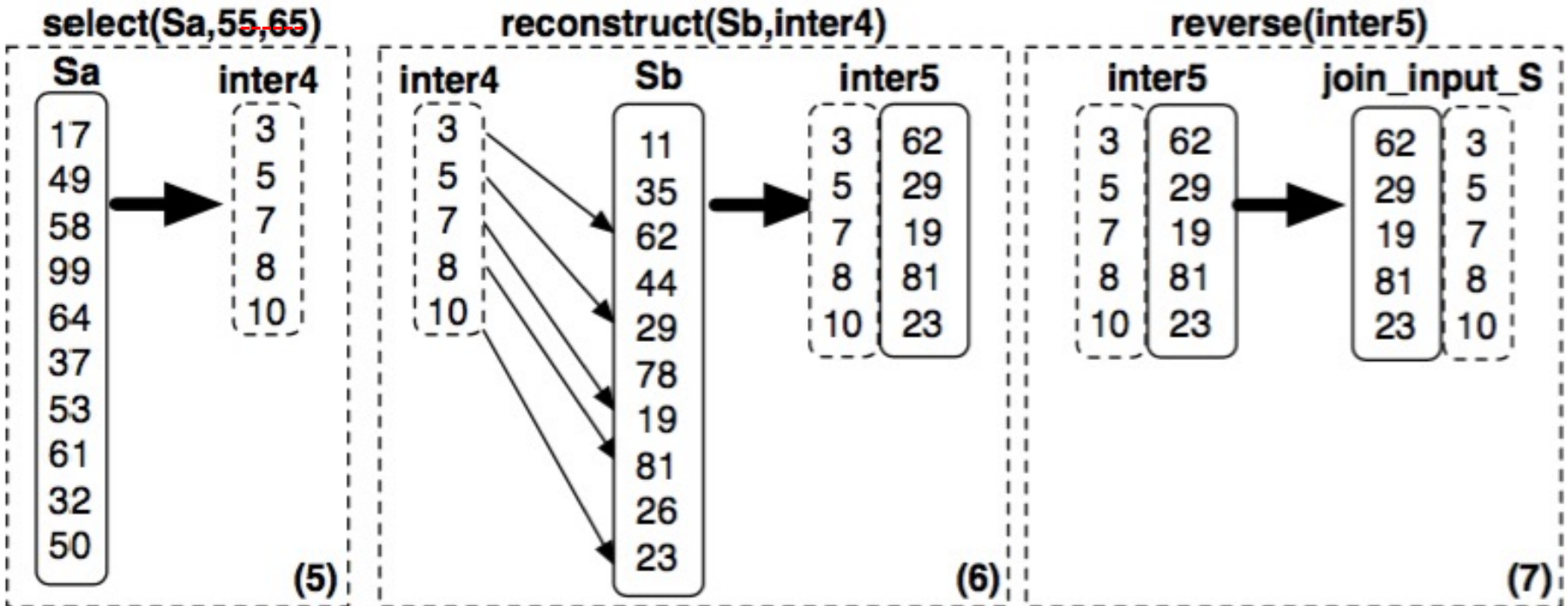
???



Late Materialization

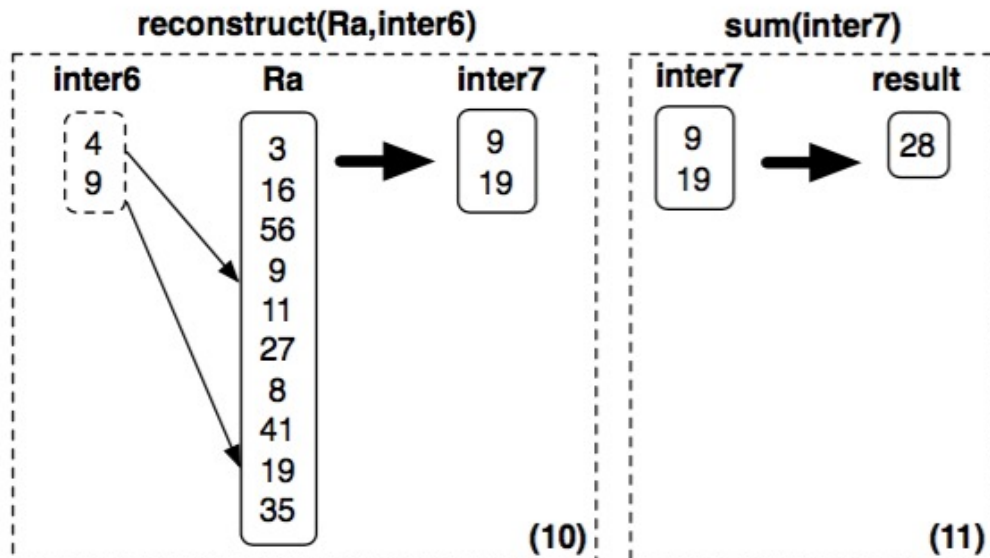
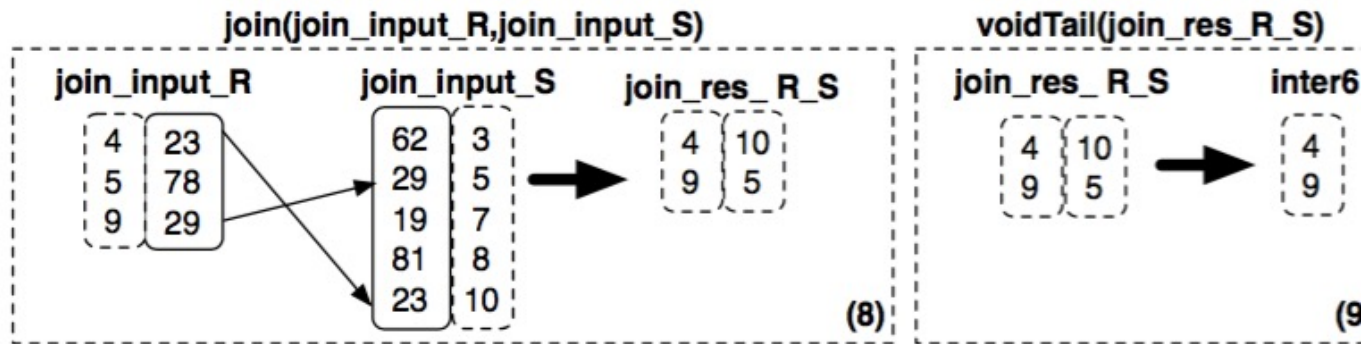
select sum(R.a) from R, S
where R.c = S.b
and 5 < R.a < 20 and 40 < R.b < 50
and 30 < S.a < 40

???



Late Materialization

select sum(R.a) from R, S
 where R.c = S.b
 and 5 < R.a < 20 and 40 < R.b < 50
 and 30 < S.a < 40



More Details

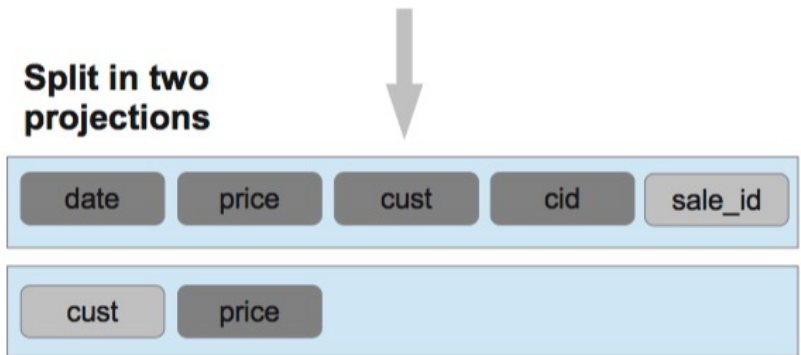
- Sort columns according to some criterion
 - Helps with range queries on that column
 - Helps compressing that column
 - But need to sort all the other columns the same way
- Create additional (redundant) "views", called "projections", by sorting on different columns

Vertica Data Model Details

Data organized into **projections**:
Sorted subsets of the attributes
Each table has one super projection
Includes all table attributes

Original Data

sale_id	cid	cust	date	price
1	11	Andrew	01/01/06	\$100
2	17	Chuck	01/05/06	\$98
3	27	Nga	01/02/06	\$90
4	28	Matt	01/03/06	\$101
5	89	Ben	01/01/06	\$103
1000	89	Ben	01/02/06	\$103
1001	11	Andrew	01/03/06	\$95



Super projection sorted by date

Non-super projection containing only(cust, price)
attributes, sorted by cust

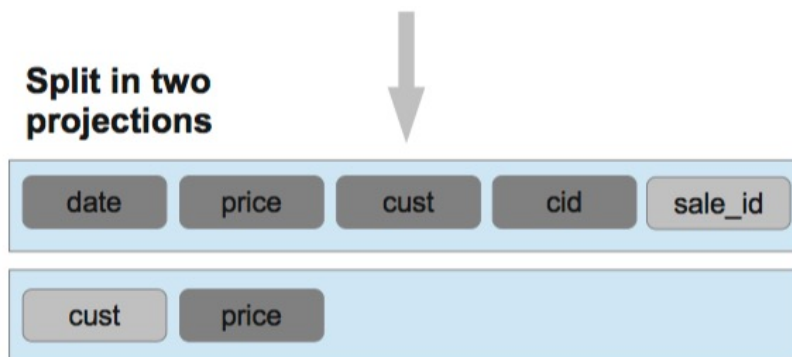
From: The Vertica Analytic Database: CStore 7 Years Later. Lamb
et. Al. VLDB'12

Parallel Processing

- Segment data horizontally across nodes
- Organize as column store on each node

Original Data

sale_id	cid	cust	date	price
1	11	Andrew	01/01/06	\$100
2	17	Chuck	01/05/06	\$98
3	27	Nga	01/02/06	\$90
4	28	Matt	01/03/06	\$101
5	89	Ben	01/01/06	\$103
1000	89	Ben	01/02/06	\$103
1001	11	Andrew	01/03/06	\$95

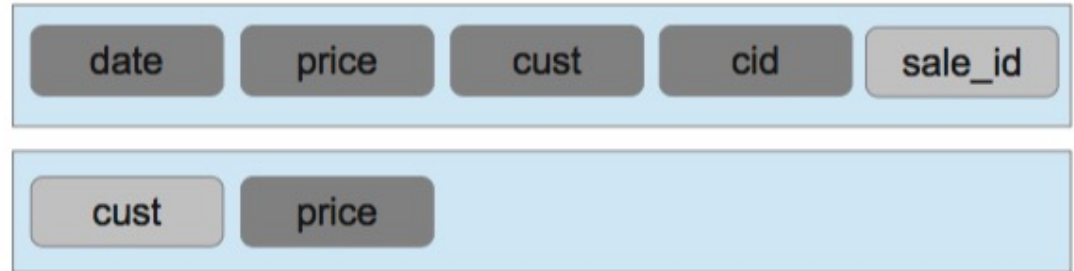


Super projection sorted by date & segmented by hash(sale_id)

Non-super projection containing only(cust, price) attributes, sorted by cust, segmented by hash(cust)

Vertica Data Model Details

Split in two projections



Segmented on several nodes

date	price	cid	cust	sale_id
01/02/06	\$90.00	27	Nga	3
01/03/06	\$95.00	11	Andrew	1001
01/03/06	\$101.00	28	Matt	4

cust	price
Andrew	\$95.00
Andrew	\$100.00
Chuck	\$98.00
Nga	\$90.00

Node 1

date	price	cid	cust	sale_id
01/01/06	\$100.00	11	Andrew	1
01/01/06	\$103.00	89	Ben	5
01/02/06	\$103.00	89	Ben	1000
01/05/06	\$98.00	17	Chuck	2

cust	price
Ben	\$103.00
Ben	\$103.00
Matt	\$101.00

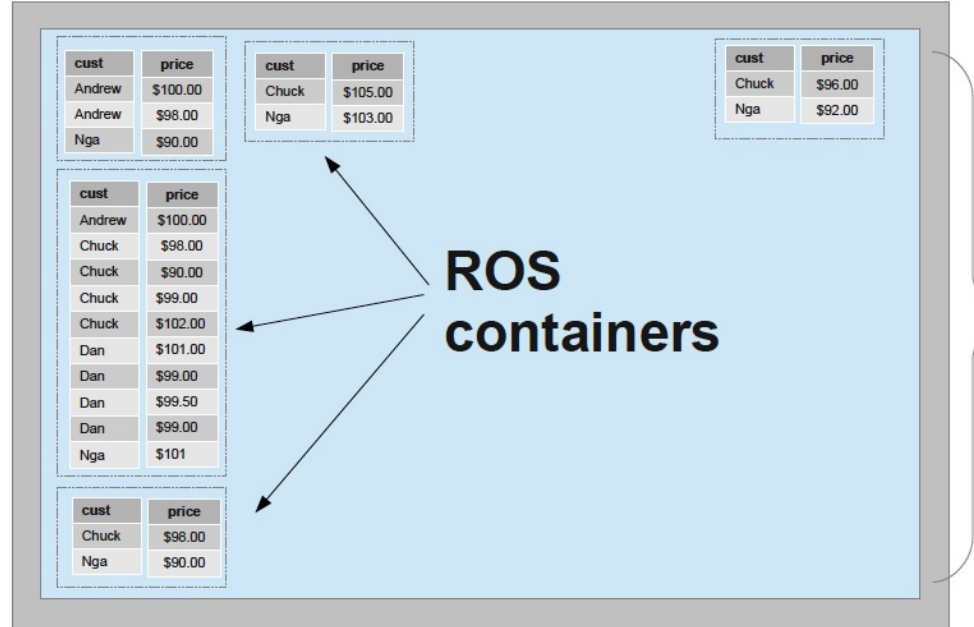
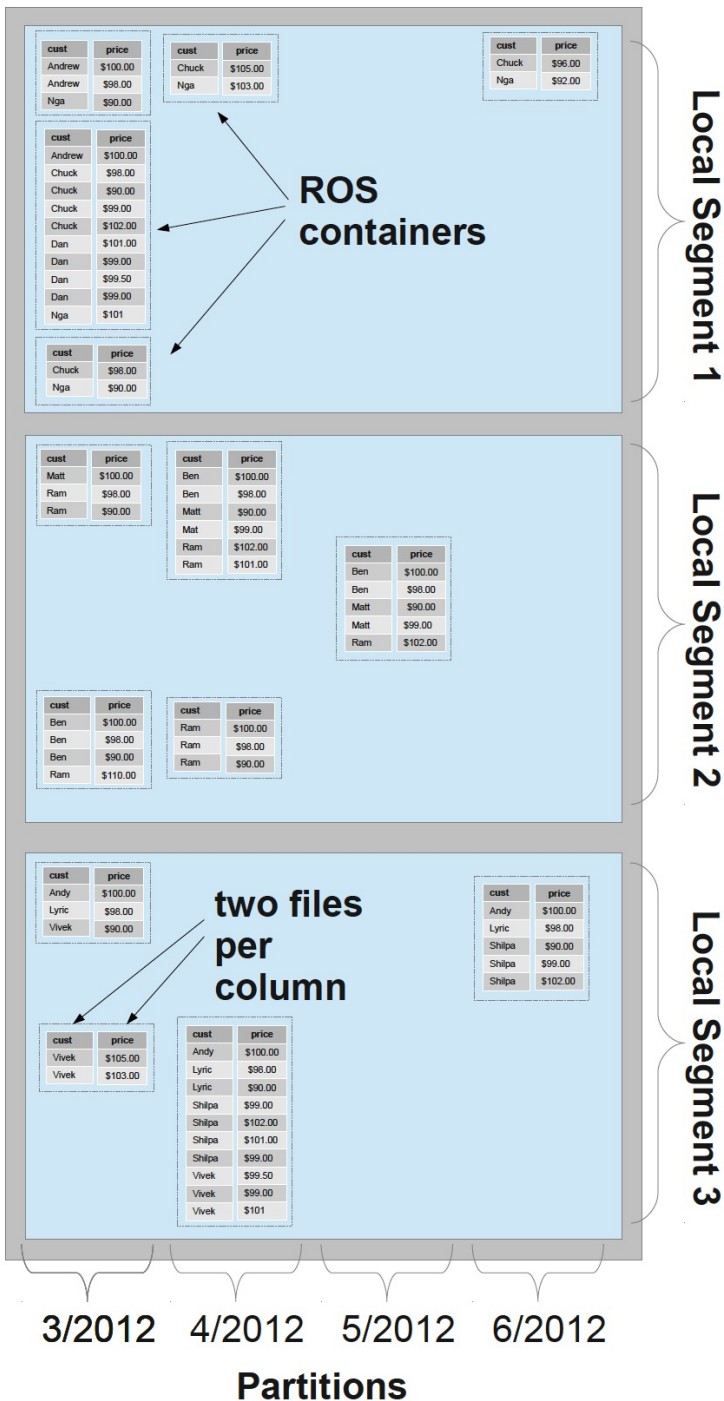
Node 2

Vertica Data Partitioning

- Cross-node partitioning called “segmentation”
 - Hash-partitioning
 - Other expression
- Each node assigned multiple local segments
 - To facilitate elasticity
 - Enables moving segments as cluster size changes
- Can also replicate all tuples in projection

Vertica Intra-Node Partitioning

- C-store proposed intra-node data partitioning
 - Similar to other parallel DBMS such as Teradata
- In contrast, Vertica divides each on-disk structure into logical regions at runtime and processing the regions in parallel
- Vertica also supports explicit data partitioning
 - Partitions segments within nodes into smaller pieces
 - `CREATE TABLE ... PARTITION BY <expr>`
 - Benefits:
 - Fast deletion
 - Pruning of partitions during query execution



Segmentation = horizontal partitioning *across* nodes
 → Each projection has own segmentation
 → More segments than nodes for elasticity
 Partition = horizontal *within* a node
 → Same partition for all projections & nodes
 ROS = Read Optimized Store
 Each column's data within its ROS container is stored as a single file
 → Total of 28 files of user data

Updates

- What is the issue?
- How does the paper address this?

Updates

- What is the issue?
 - Updates in a sorted column require reordering of the entire column, and the other columns as well
- How does the paper address this?

Updates

- What is the issue?
 - Updates in a sorted column require reordering of the entire column, and the other columns as well
- How does the paper address this?
 - Update to Write Optimized Store (WOS)
 - Queries on Read Optimized Store (ROS)

C-Store/Vertica Design

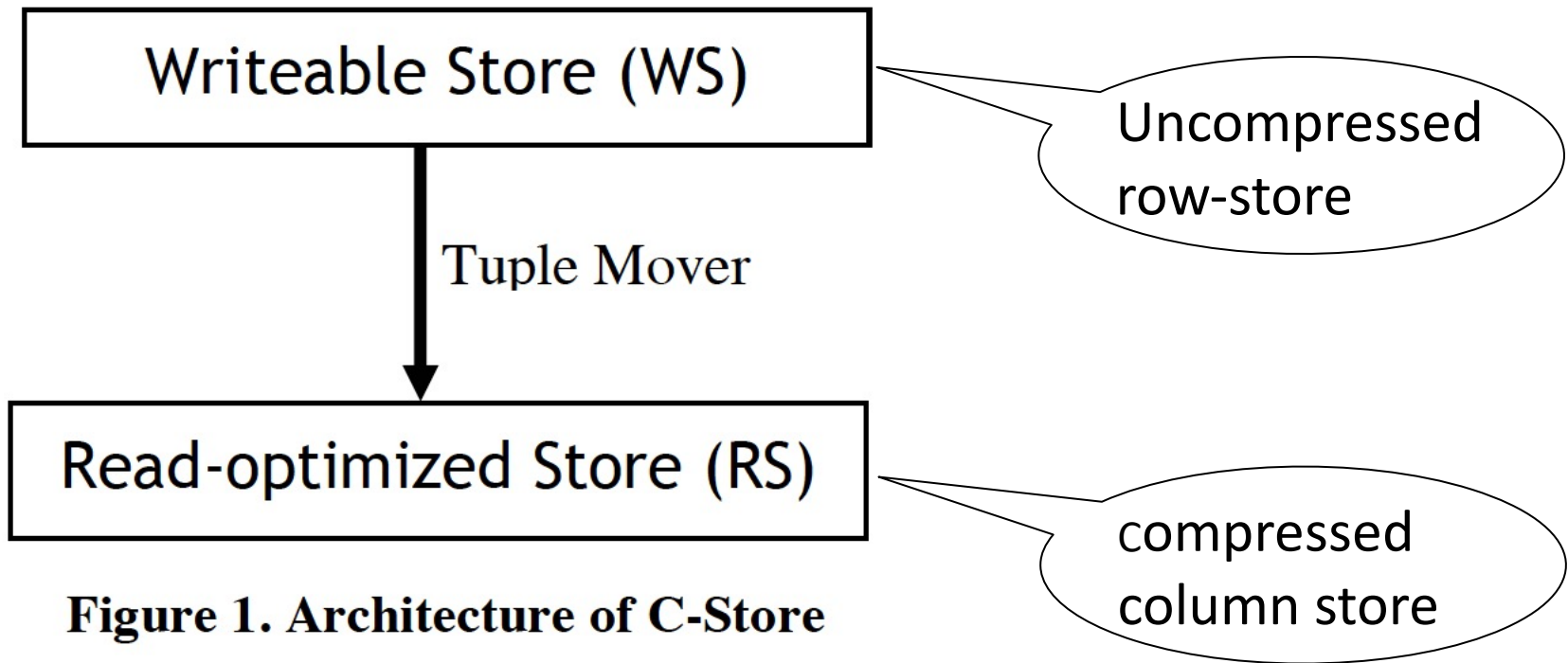


Figure 1. Architecture of C-Store

From: C-Store: A Column-oriented DBMS. Stonebraker et. Al. VLDB'05

Read and Write Optimized Stores

- **Write Optimized Store (WOS)**
 - In memory data: buffer delete/insert/update operations
 - Column vs row does not matter
- **Tuples never modified in place**
 - Use “delete vector” to track deleted tuples
 - Eventually removed by tuple mover during ROS merge
- **Tuple mover**
 - Move between WOS and ROS
 - When moving tuples out, creates a new ROS container
 - Merges ROS files together
 - Better compression & faster processing (fewer files to merge)

Read and Write Optimized Stores

- **Read Optimized Store (ROS)**
 - Multiple ROS containers
 - Stored on standard file system
 - Logically contains some number of complete tuples sorted by the *projection's* sort order, stored as a pair of files per column: position index & data
 - The position index = only metadata per disk block
 - Column files may be independently retrieved

Final Thoughts

Simulating a Column-Store in a Row-Store DBMS:

- Vertical partitioning
 - Two-column tables: (key, attribute)
- Index-only plans
 - Create a B+ tree index on each attribute
 - Answer queries using indexes only, without reading actual data
- Materialized views
 - Each view contains a subset of columns

PAX

An Intermediate Format: PAX

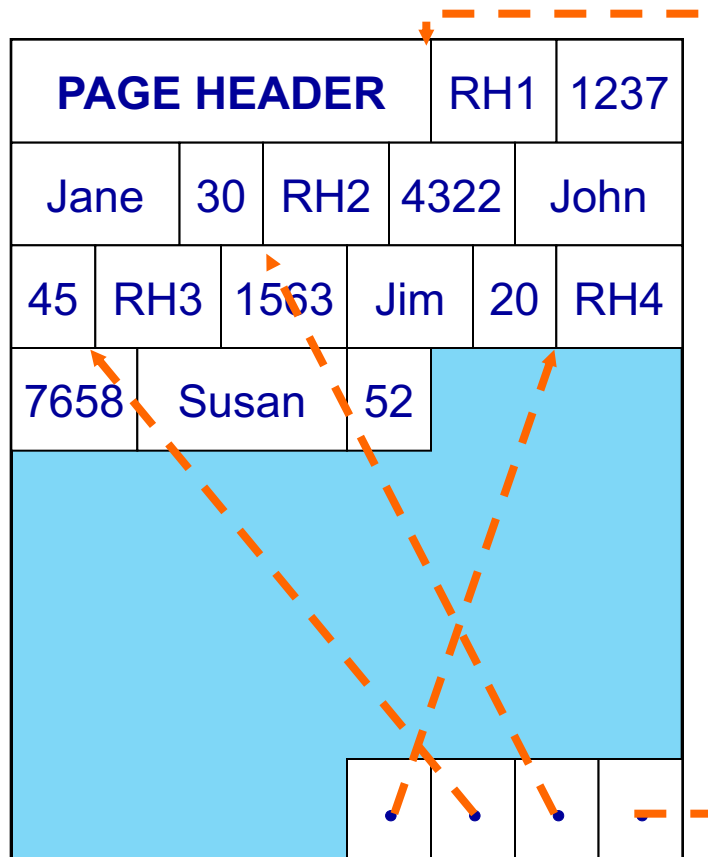
- PAX = Partition Attributes Across
- Addresses memory access bottleneck (not the disk bottleneck)

Current Scheme: Slotted Pages

Formal name: NSM (N-ary Storage Model)

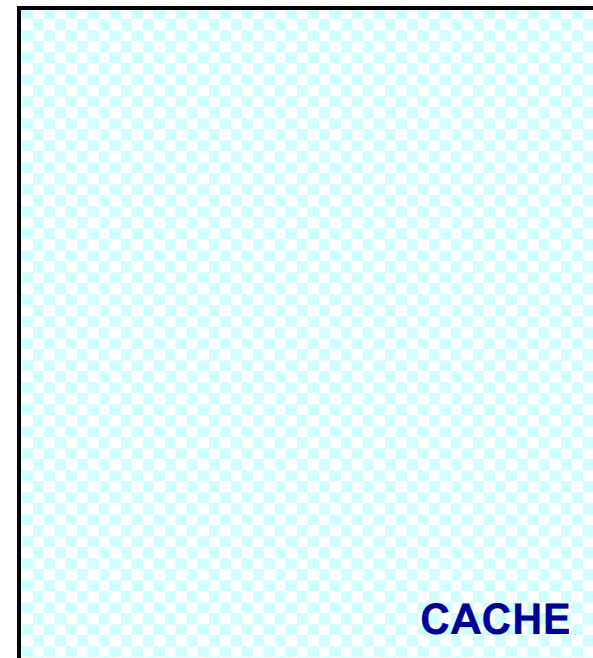
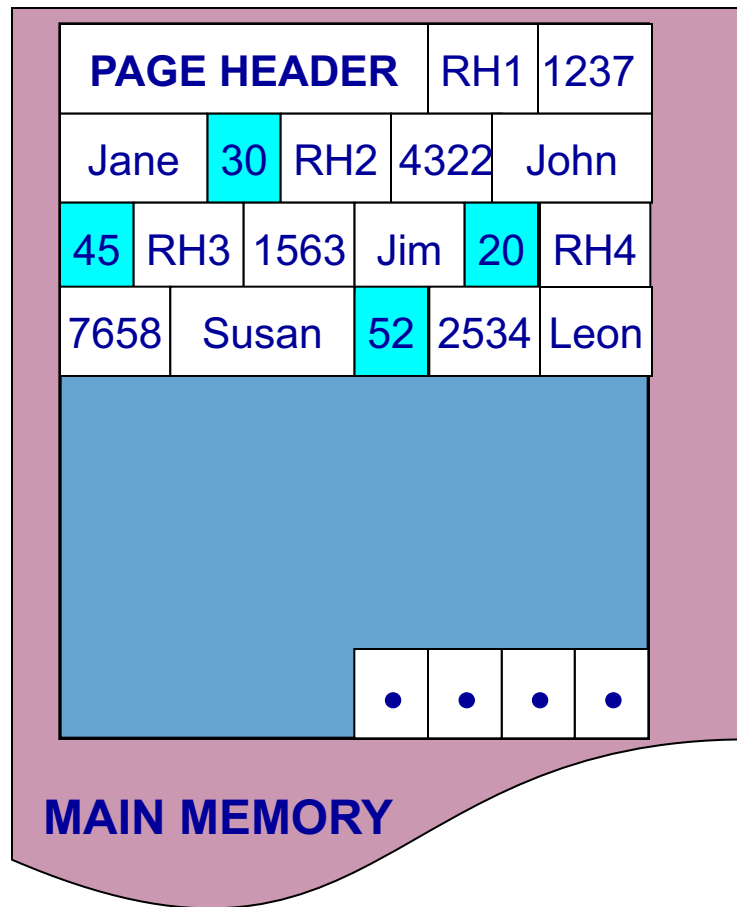
R

RID	SSN	Name	Age
1	1237	Jane	30
2	4322	John	45
3	1563	Jim	20
4	7658	Susan	52
5	2534	Leon	43
6	8791	Dan	37



- Records are stored sequentially
- Offsets to start of each record at end of page

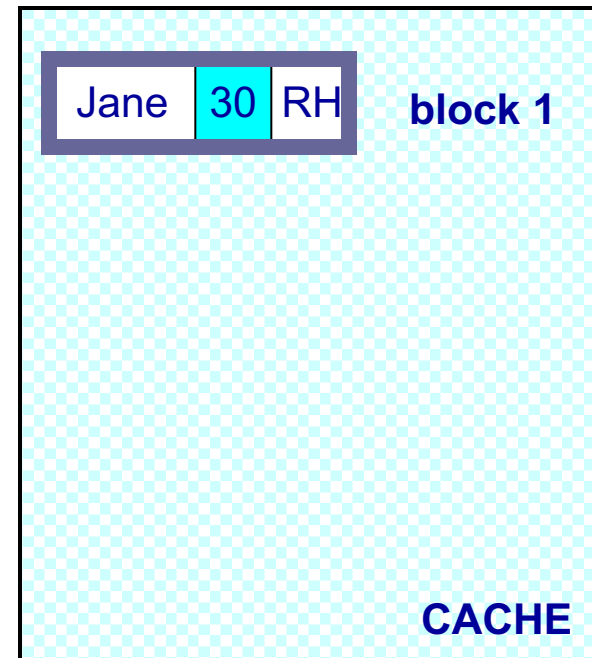
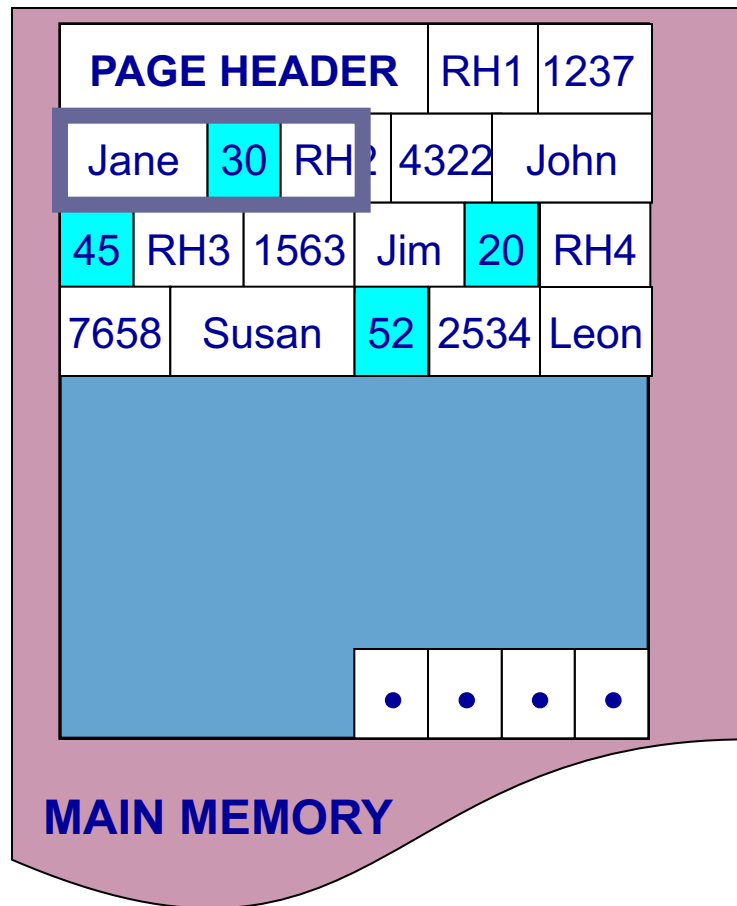
Predicate Evaluation using NSM



```
select name  
from R  
where age > 50
```

NSM pushes non-referenced data to the cache

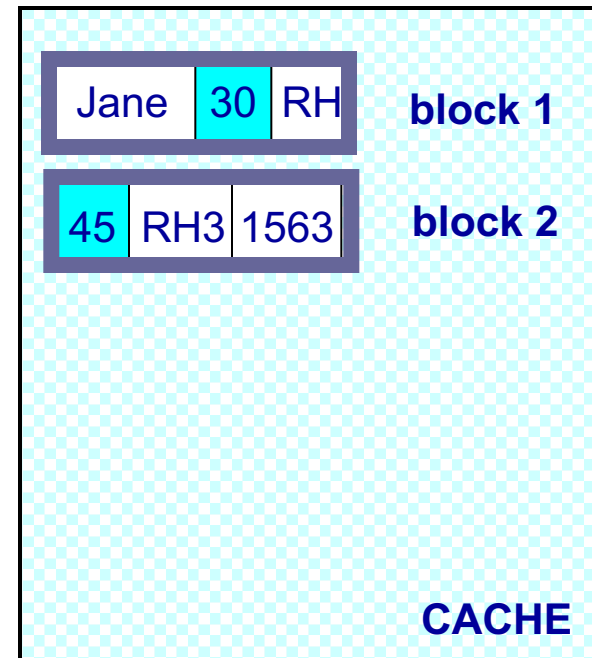
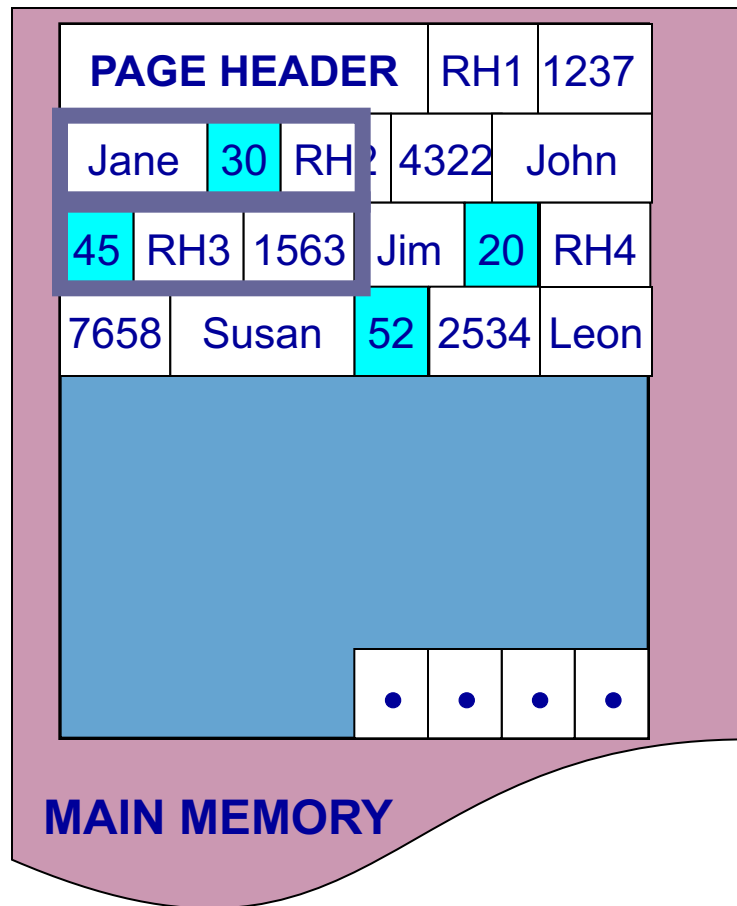
Predicate Evaluation using NSM



*select name
from R
where age > 50*

NSM pushes non-referenced data to the cache

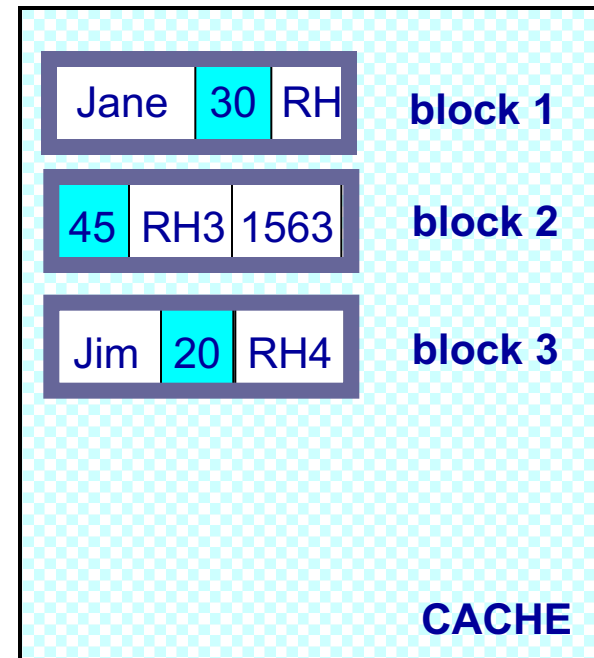
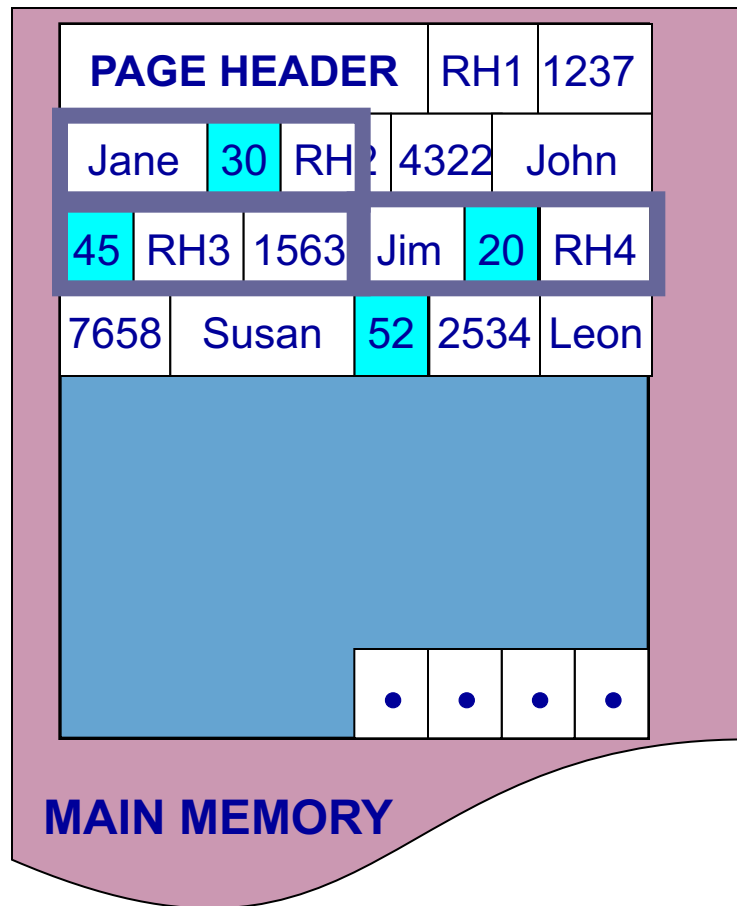
Predicate Evaluation using NSM



*select name
from R
where age > 50*

NSM pushes non-referenced data to the cache

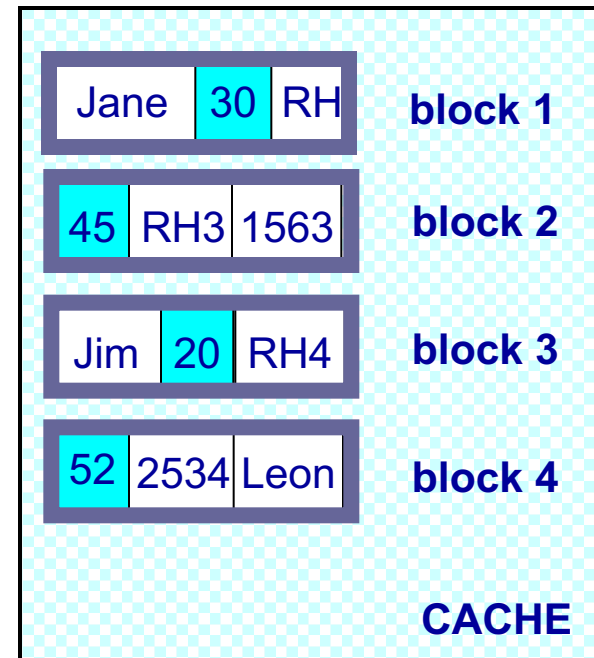
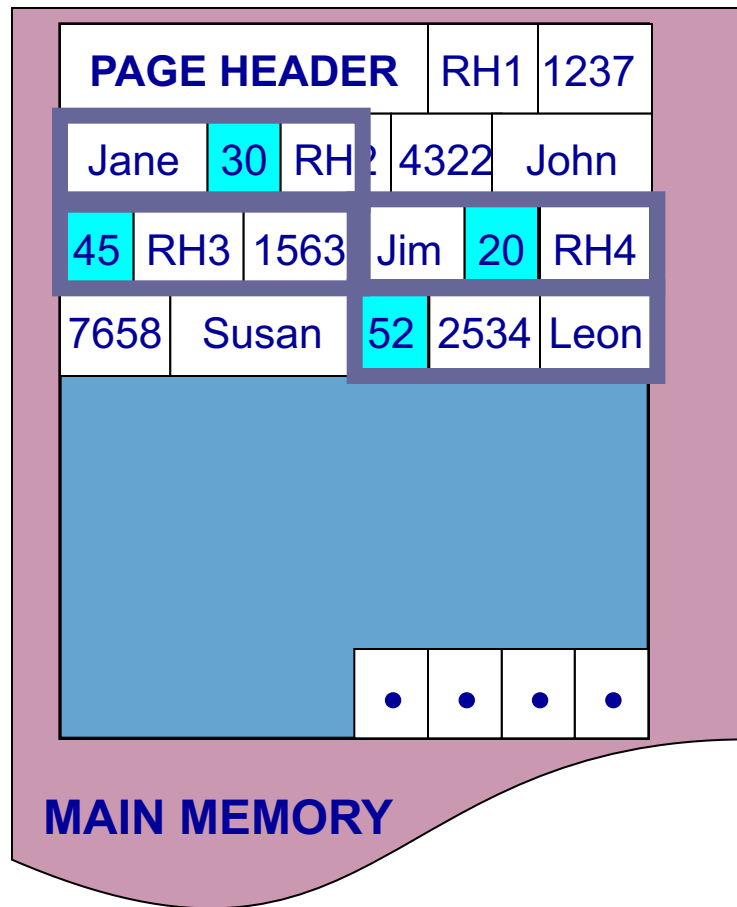
Predicate Evaluation using NSM



*select name
from R
where age > 50*

NSM pushes non-referenced data to the cache

Predicate Evaluation using NSM



*select name
from R
where age > 50*

NSM pushes non-referenced data to the cache

Need New Data Page Layout

- Eliminates unnecessary memory accesses
- Improves inter-record locality
- Keeps a record's fields together
- Does not affect I/O performance

and, most importantly, is...

low-implementation-cost, high-impact

Partition Attributes Across (PAX)

NSM PAGE

PAGE HEADER				RH1	1237		
Jane	30	RH2	4322	John			
45	RH3	1563	Jim	20	RH4		
7658	Susan	52					
						•	•

PAX PAGE

PAGE HEADER				1237	4322		
1563	7658						
						•	•
		Jane	John	Jim	Susan		
				•	•	•	•
				30	52	45	20
				•	•	•	•

Partition data *within* the page for spatial locality

Partition Attributes Across (PAX)

NSM PAGE

PAGE HEADER		RH1	1237				
Jane	30	RH2	4322	John			
45	RH3	1563	Jim	20	RH4		
7658	Susan	52					

PAX PAGE

PAGE HEADER		1237	4322		
1563	7658				
Jane	John	Jim	Susan		
30	52	45	20		

Partition data *within* the page for spatial locality

Partition Attributes Across (PAX)

NSM PAGE

PAGE HEADER			RH1	1237	
Jane	30	RH2	4322	John	
45	RH3	1563	Jim	20	RH4
7658	Susan	52			
				.	
				.	
				.	
				.	

PAX PAGE

PAGE HEADER		1237	4322
1563	7658		
Jane	John	Jim	Susan
30	52	45	20

Partition data *within* the page for spatial locality

Partition Attributes Across (PAX)

NSM PAGE

PAGE HEADER		RH1	1237
Jane	30	RH2	4322
45	RH3	1563	Jim 20 RH4
7658	Susan	52	
• • • •			

PAX PAGE

PAGE HEADER		1237	4322
1563	7658		
Jane	John	Jim	Susan
• • • •			
		30 52 45 20	

Partition data *within* the page for spatial locality

Partition Attributes Across (PAX)

NSM PAGE

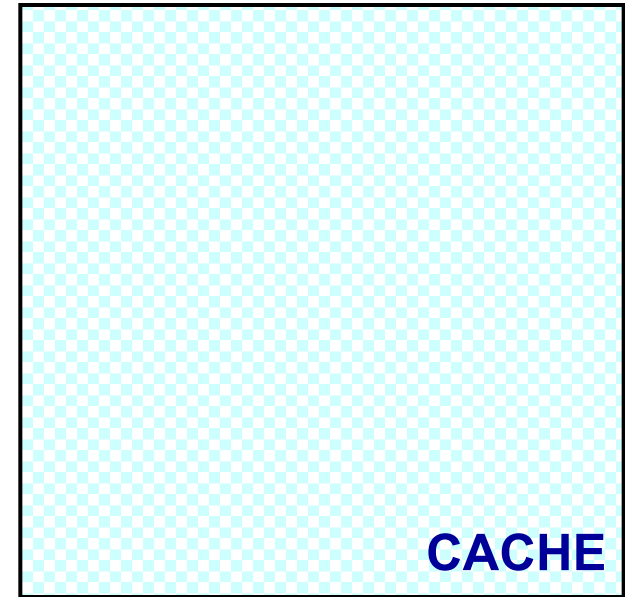
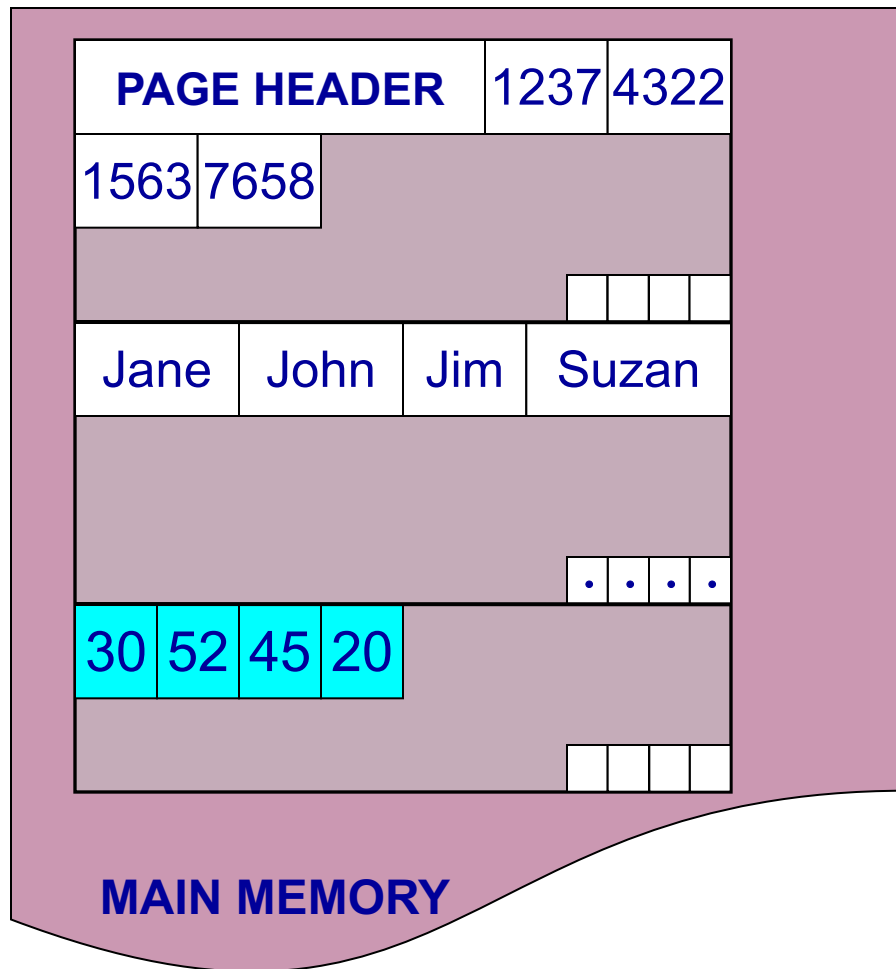
PAGE HEADER		RH1	1237		
Jane	30	RH2	4322	John	
45	RH3	1563	Jim	20	
7658	Susan	52	[Light Blue Area]		
[Light Blue Area]					
[Light Blue Area]					
				[...]	

PAX PAGE

PAGE HEADER		1237	4322
1563	7658	[Grey Area]	
[Grey Area]			
[Grey Area]		[...]	
Jane	John	Jim	Susan
[Grey Area]			
[Grey Area]		[...]	
30	52	45	20
[Grey Area]			
[Grey Area]		[...]	

Partition data *within* the page for spatial locality

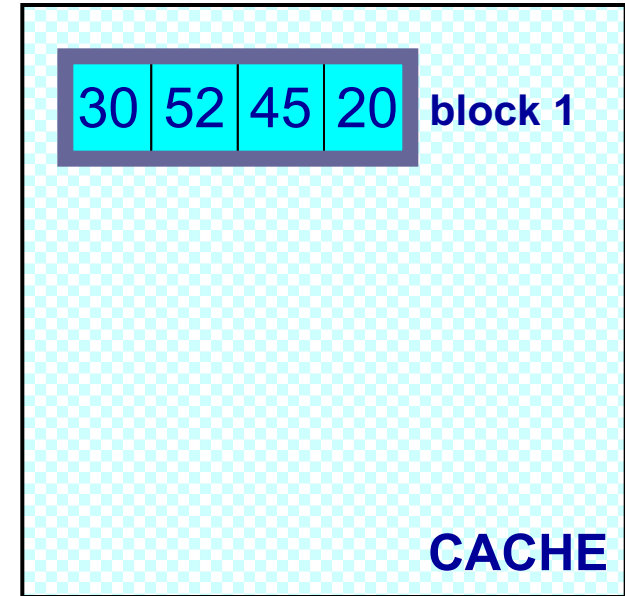
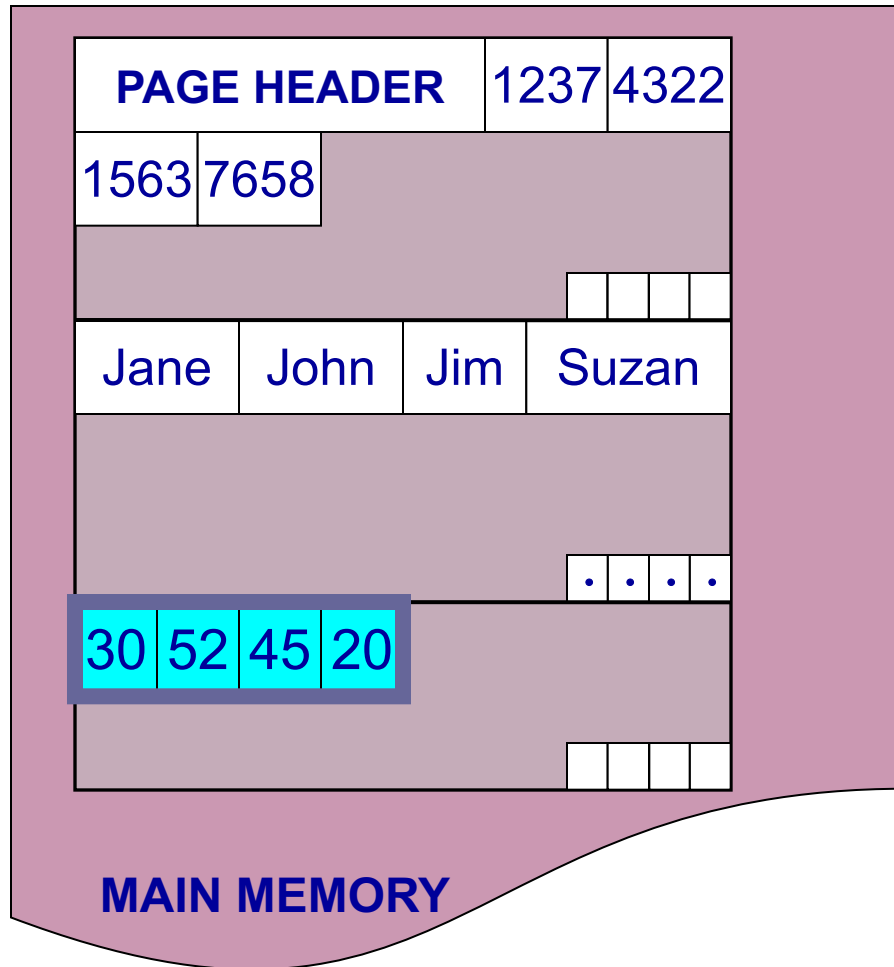
Predicate Evaluation using PAX



```
select name  
from R  
where age > 50
```

Fewer cache misses, low reconstruction cost

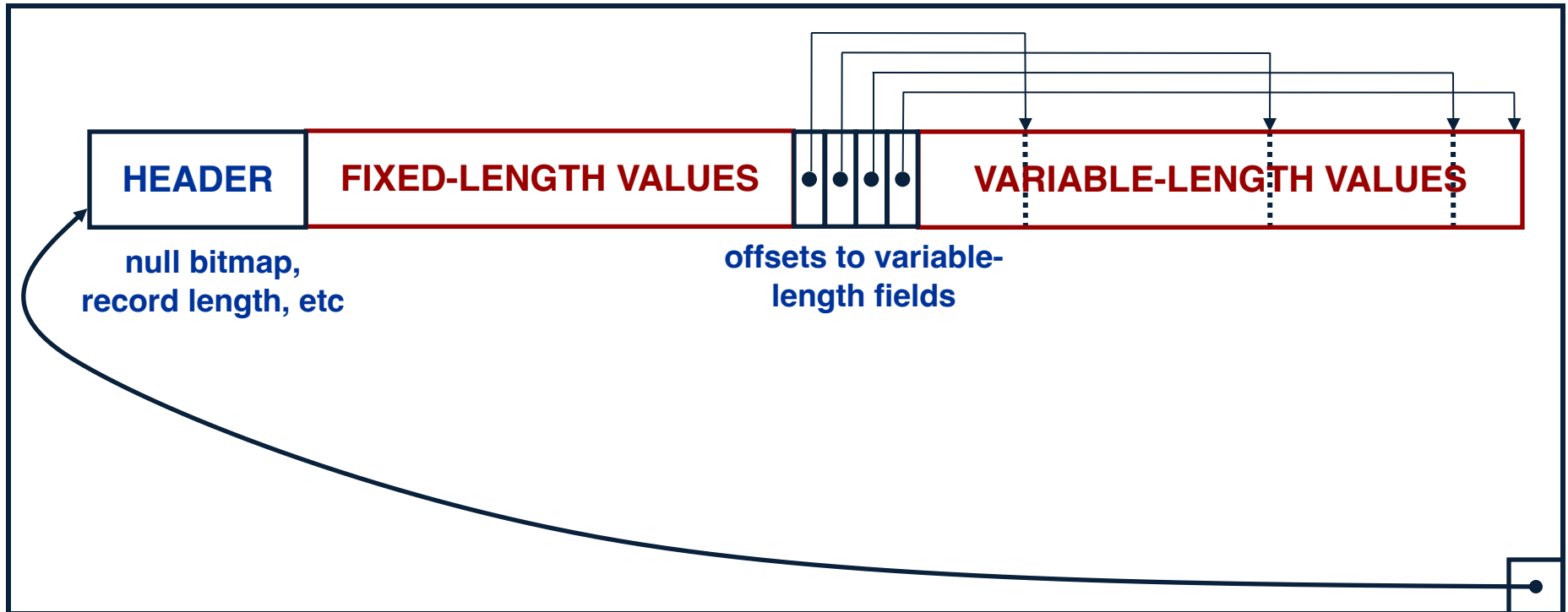
Predicate Evaluation using PAX



*select name
from R
where age > 50*

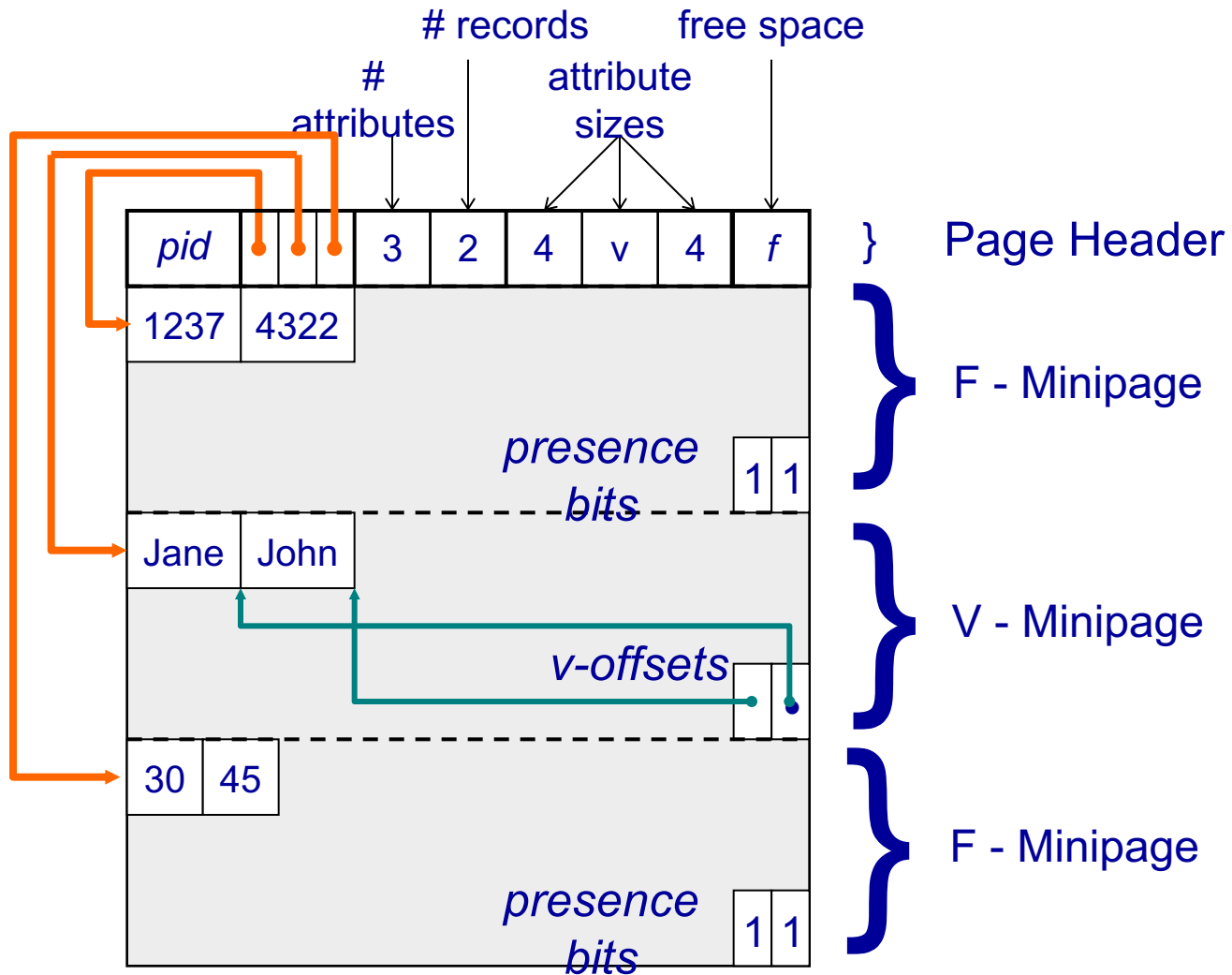
Fewer cache misses, low reconstruction cost

A Real NSM Record



NSM: All fields of record stored together + slots

PAX: Detailed Design



PAX: Group fields + amortizes record headers

PAX - Summary

- Improves processor cache locality
- Does not affect I/O behavior
 - Same disk accesses for NSM or PAX storage
 - No need to change the buffer manager
- Today:
 - Most (all?) commercial engines use a PAX layout of the disk
 - Beyond disk: Snowflake partitions tables horizontally into files, then uses column-store inside each file (hence, PAX)