Column-Oriented Database Systems

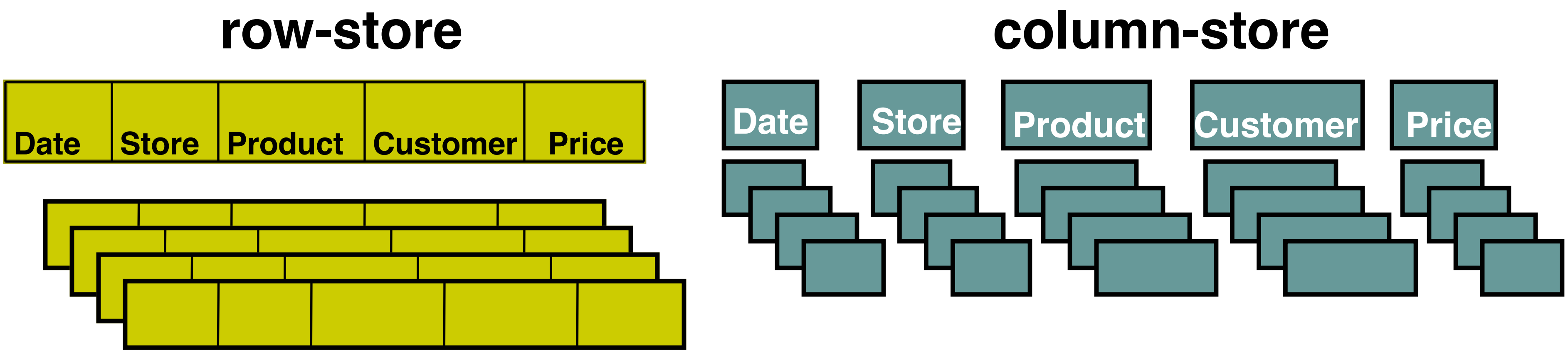
VLDB 2009 Tutorial

Part 1: Stavros Harizopoulos (HP Labs)

Part 2: Daniel Abadi (Yale)

Part 3: Peter Boncz (CWI)

What is a column-store?



+ easy to add/modify a record

- might read in unnecessary data

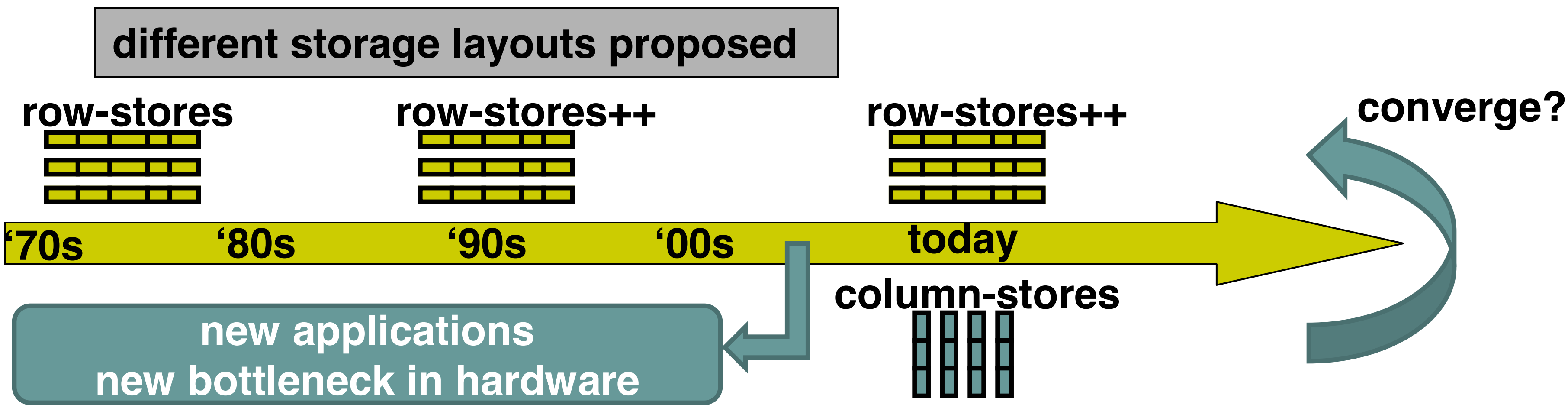
+ only need to read in relevant data

- tuple writes require multiple accesses

=> *suitable for read-mostly, read-intensive, large data repositories*

Are these two fundamentally different?

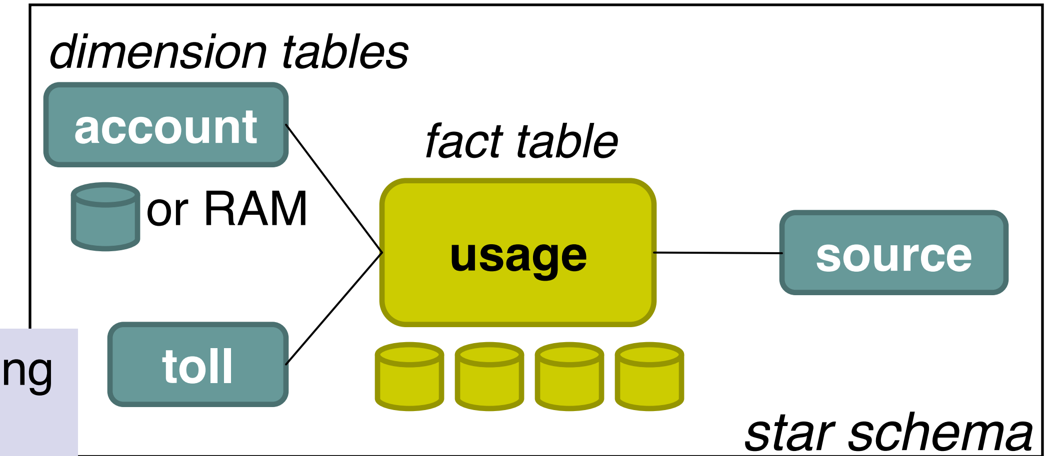
- The only fundamental difference is the storage layout
- However: we need to look at the big picture



Telco Data Warehousing example

- Typical DW installation
- Real-world example

“One Size Fits All? - Part 2: Benchmarking Results” Stonebraker et al. CIDR 2007



QUERY 2

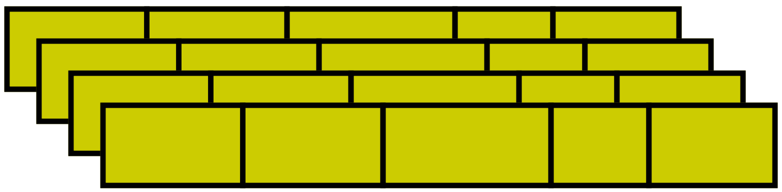
```
SELECT account.account_number,
sum (usage.toll_airtime),
sum (usage.toll_price)
FROM usage, toll, source, account
WHERE usage.toll_id = toll.toll_id
AND usage.source_id = source.source_id
AND usage.account_id = account.account_id
AND toll.type_ind in (‘AE’. ‘AA’)
AND usage.toll_price > 0
AND source.type != ‘CIBER’
AND toll.rating_method = ‘IS’
AND usage.invoice_date = 20051013
GROUP BY account.account_number
```

	Column-store	*Row-store*
Query 1	*2.06*	*300*
Query 2	*2.20*	*300*
Query 3	*0.09*	*300*
Query 4	*5.24*	*300*
Query 5	*2.88*	*300*

Why? Three main factors (next slides)

Telco example explained (1/3): *read efficiency*

row store



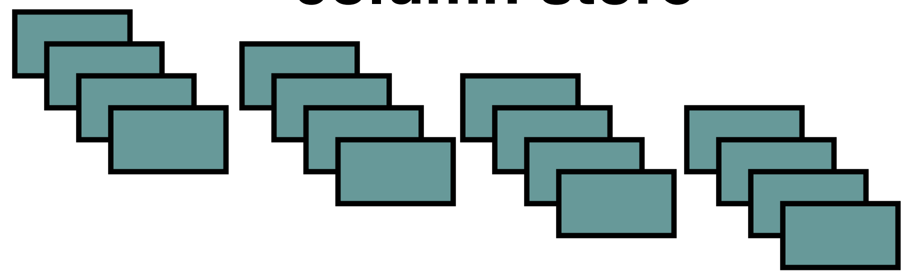
read pages containing entire rows

one row = 212 columns!

is this typical? (it depends)

What about vertical partitioning? (it does not work with ad-hoc queries)

column store



read only columns needed

in this example: 7 columns

caveats:

- "select * " not any faster
- clever disk prefetching
- clever tuple reconstruction

Telco example explained (2/3): *compression efficiency*

- Columns compress better than rows
  - Typical row-store compression ratio 1 : 3
  - Column-store 1 : 10
- Why?
  - Rows contain values from different domains

    => more entropy, difficult to dense-pack
  - Columns exhibit significantly less entropy
  - Examples:

    Male, Female, Female, Female, Male

    1998, 1998, 1999, 1999, 1999, 2000
  - Caveat: CPU cost (use lightweight compression)

Telco example explained (3/3): *sorting & indexing efficiency*

- Compression and dense-packing free up space
  - Use multiple overlapping column collections
  - Sorted columns compress better
  - Range queries are faster
  - Use sparse clustered indexes

What about heavily-indexed row-stores?
(works well for single column access,
cross-column joins become increasingly expensive)

Additional opportunities for column-stores

- Block-tuple / vectorized processing
  - Easier to build block-tuple operators
    - Amortizes function-call cost, improves CPU cache performance
  - Easier to apply vectorized primitives
    - Software-based: bitwise operations
    - Hardware-based: SIMD

Part 3

- Opportunities with compressed columns
  - *Avoid* decompression: operate directly on compressed
  - *Delay* decompression (and tuple reconstruction)
    - Also known as: *late materialization*

more in Part 2

- Exploit columnar storage in other DBMS components
  - Physical design (both static and dynamic)

See: *Database Cracking*, from CWI

MonetDB (more in Part 3)

- Late 1990s, CWI: Boncz, Manegold, and Kersten
- Motivation:
  - Main-memory
  - Improve computational efficiency by avoiding expression interpreter
  - DSM with virtual IDs natural choice
  - Developed new query execution algebra
- Initial contributions:
  - Pointed out memory-wall in DBMSs
  - Cache-conscious projections and joins
  - …

2005: the (re)birth of column-stores

- New hardware and application realities
  - Faster CPUs, larger memories, disk bandwidth limit
  - Multi-terabyte Data Warehouses
- New approach: combine several techniques
  - Read-optimized, fast multi-column access, disk/CPU efficiency, light-weight compression
- C-store paper:
  - First comprehensive design description of a column-store
- MonetDB/X100
  - “proper” disk-based column store
- Explosion of new products

Applications for column-stores

- Data Warehousing
  - High end (clustering)
  - Mid end/Mass Market
  - Personal Analytics
- Data Mining
  - E.g. Proximity
- Google BigTable
- RDF
  - Semantic web data management
- Information retrieval
  - Terabyte TREC
- Scientific datasets
  - SciDB initiative
  - SLOAN Digital Sky Survey on MonetDB

List of column-store systems

- Cantor (history)
- Sybase IQ
- SenSage (former Addamark Technologies)
- Kdb
- 1010data
- MonetDB
- C-Store/Vertica
- X100/VectorWise
- KickFire
- SAP Business Accelerator
- Infobright
- ParAccel
- Exasol

Simulate a Column-Store inside a Row-Store

Date	Store	Product	Customer	Price
01/01	BOS	Table	Mesa	$20
01/01	NYC	Chair	Lutz	$13
01/01	BOS	Bed	Mudd	$79

Option A: Vertical Partitioning

Date

TID	Value
1	01/01
2	01/01
3	01/01

Store

TID	Value
1	BOS
2	NYC
3	BOS

Product

TID	Value
1	Table
2	Chair
3	Bed

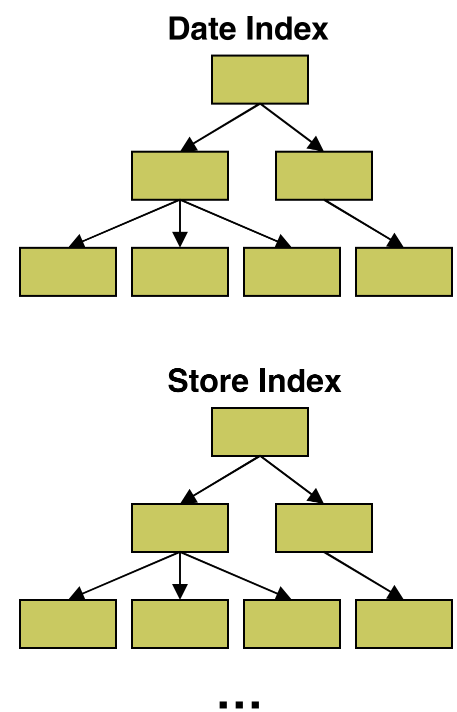
Customer

TID	Value
1	Mesa
2	Lutz
3	Mudd

Price

TID	Value
1	$20
2	$13
3	$79

Option B: Index Every Column



Simulate a Column-Store inside a Row-Store

Date	Store	Product	Customer	Price
01/01	BOS	Table	Mesa	$20
01/01	NYC	Chair	Lutz	$13
01/01	BOS	Bed	Mudd	$79

Option A: Vertical Partitioning

Date

Value	StartPos	Length
01/01	1	3

Store

TID	Value
1	BOS
2	NYC
3	BOS

Product

TID	Value
1	Table
2	Chair
3	Bed

Customer

TID	Value
1	Mesa
2	Lutz
3	Mudd

Price

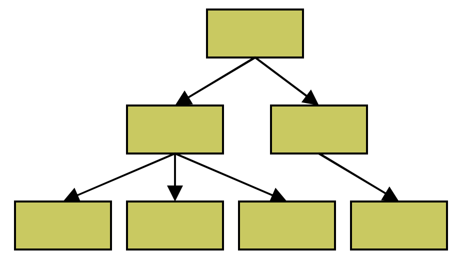
TID	Value
1	$20
2	$13
3	$79

Can explicitly run-length encode date

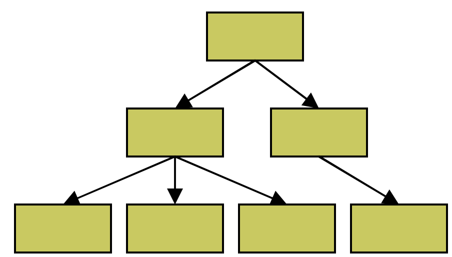
"Teaching an Old Elephant New Tricks."
Bruno, CIDR 2009.

Option B: Index Every Column

Date Index



Store Index



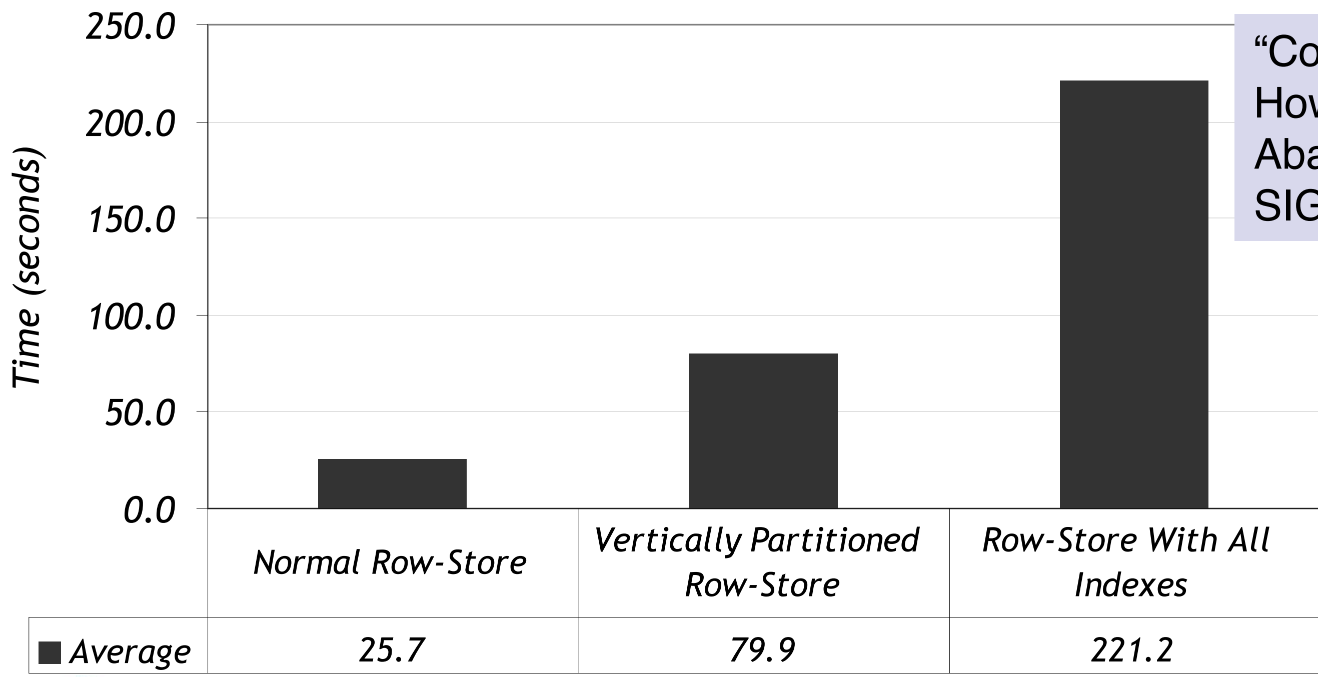
…

Experiments

- Star Schema Benchmark (SSBM)

Adjoined Dimension Column Index (ADC Index) to Improve Star Schema Query Performance”. O’Neil et. al. ICDE 2008.

- Implemented by professional DBA
- Original row-store plus 2 column-store simulations on same row-store product



“Column-Stores vs Row-Stores: How Different are They Really?” Abadi, Hachem, and Madden. SIGMOD 2008.

Column-Oriented Database Systems

VLDB 2009 Tutorial

Compression

“Super-Scalar RAM-CPU Cache Compression” Zukowski, Heman, Nes, Boncz, ICDE’06

“Integrating Compression and Execution in Column-Oriented Database Systems” Abadi, Madden, and Ferreira, SIGMOD ’06

•Query optimization in compressed database systems” Chen, Gehrke, Korn, SIGMOD’01

Compression

- Trades I/O for CPU
- Increased column-store opportunities:
  - Higher data value locality in column stores
  - Techniques such as run length encoding far more useful
  - Can use extra space to store multiple copies of data in different sort orders

Run-length Encoding

Quarter	Product ID	Price
Q1	1	5
Q1	1	7
Q1	1	2
Q1	1	9
Q1	1	6
Q1	2	8
Q1	2	5
…	…	…
Q2	1	3
Q2	1	8
Q2	1	1
Q2	2	4
…	…	…

→

Quarter

(value, start_pos, run_length)

(Q1, 1, 300)
(Q2, 301, 350)
(Q3, 651, 500)
(Q4, 1151, 600)

Product ID

(value, start_pos, run_length)

(1, 1, 5)
(2, 6, 2)
…
(1, 301, 3)
(2, 304, 1)
…

Price

5
7
2
9
6
8
5
…
3
8
1
4
…

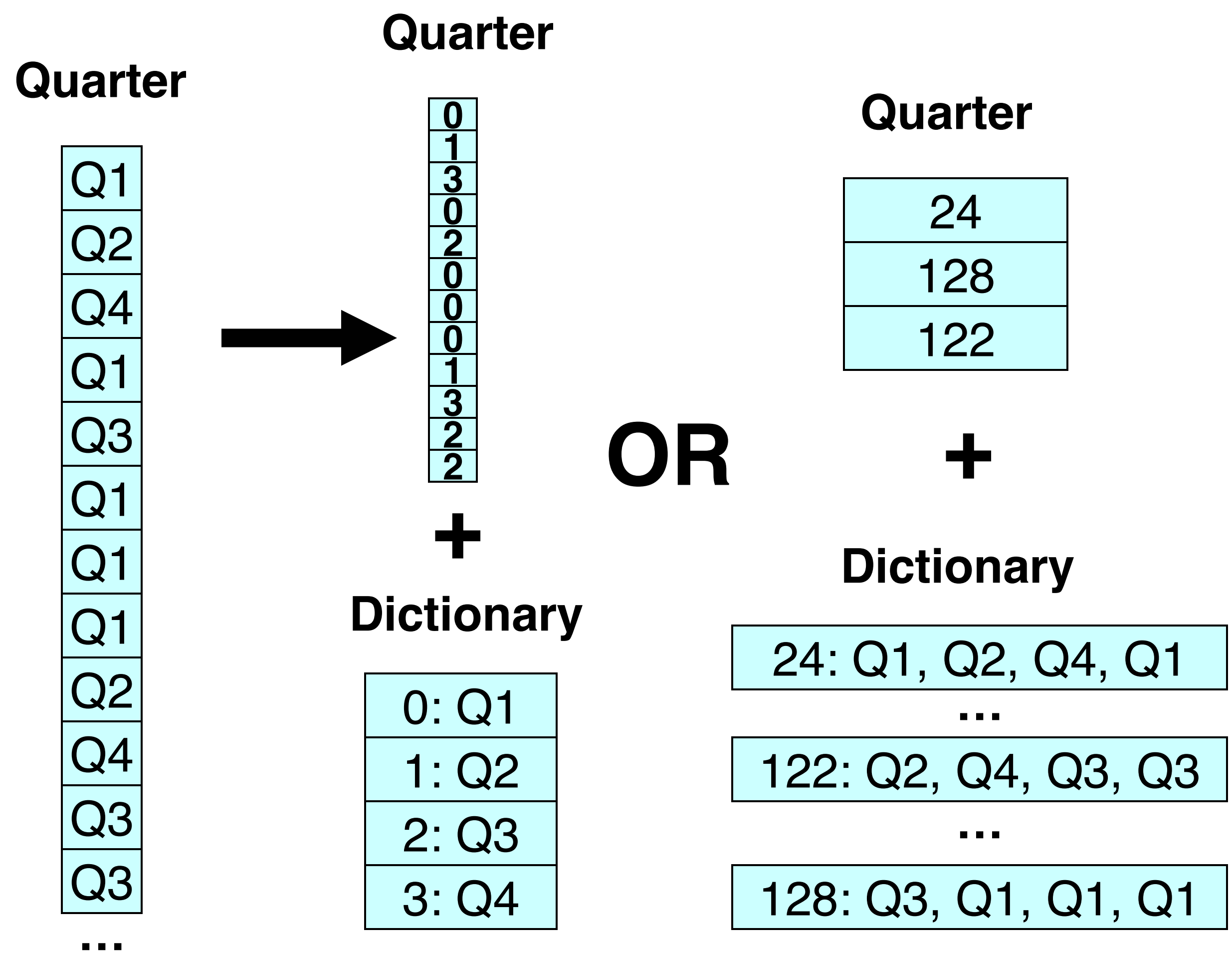
Bit-vector Encoding

- For each unique value, v, in column c, create bit-vector b
  - b[i] = 1 if c[i] = v
- Good for columns with few unique values
- Each bit-vector can be further compressed if sparse

Product ID		ID: 1	ID: 2	ID: 3	…
1	→	1	0	0	0
1		1	0	0	0
1		1	0	0	0
1		1	0	0	0
1		1	0	0	0
2		0	1	0	0
2		0	1	0	0
…		…	…	…	…
1		1	0	0	0
1		1	0	0	0
2		0	1	0	0
3		0	0	1	0
…		…	…	…	…

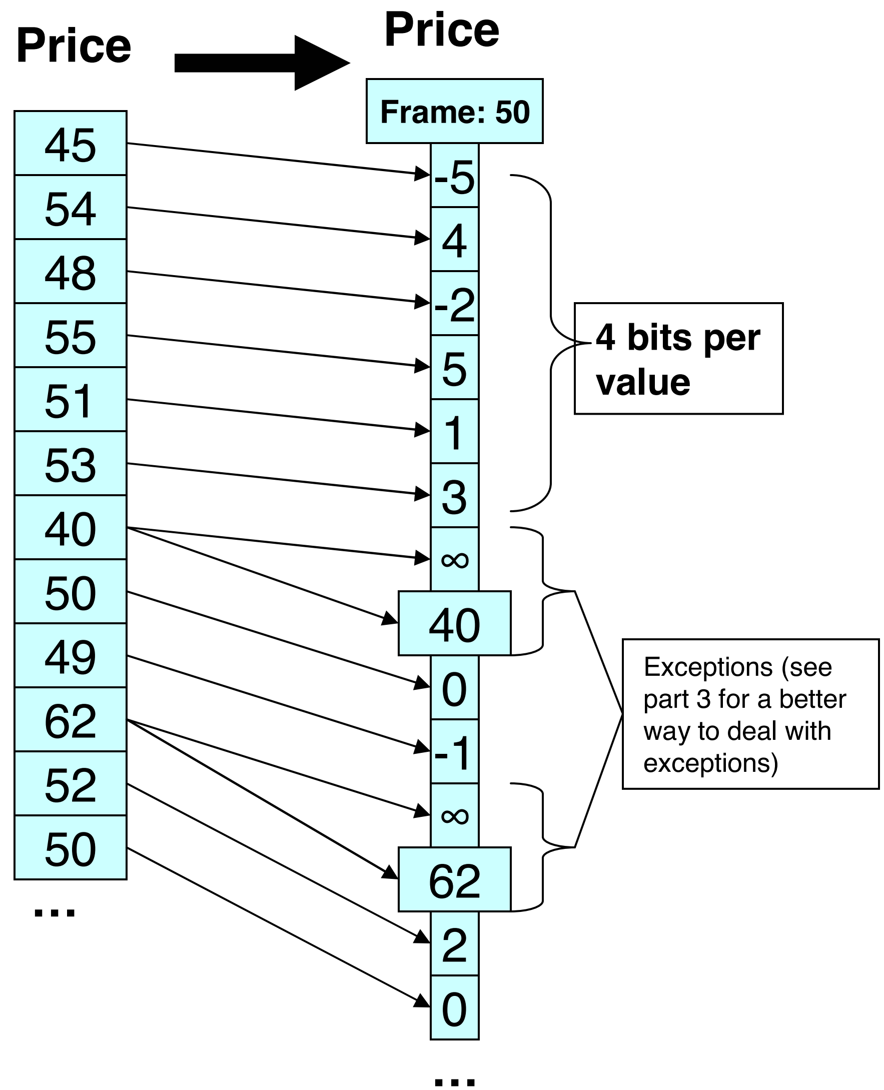
Dictionary Encoding

- **For each unique value create dictionary entry**
- **Dictionary can be per-block or per-column**
- **Column-stores have the advantage that dictionary entries may encode multiple values at once**



Frame Of Reference Encoding

- **Encodes values as b bit offset from chosen frame of reference**
- **Special escape code (e.g. all bits set to 1) indicates a difference larger than can be stored in b bits**
  - **After escape code, original (uncompressed) value is written**

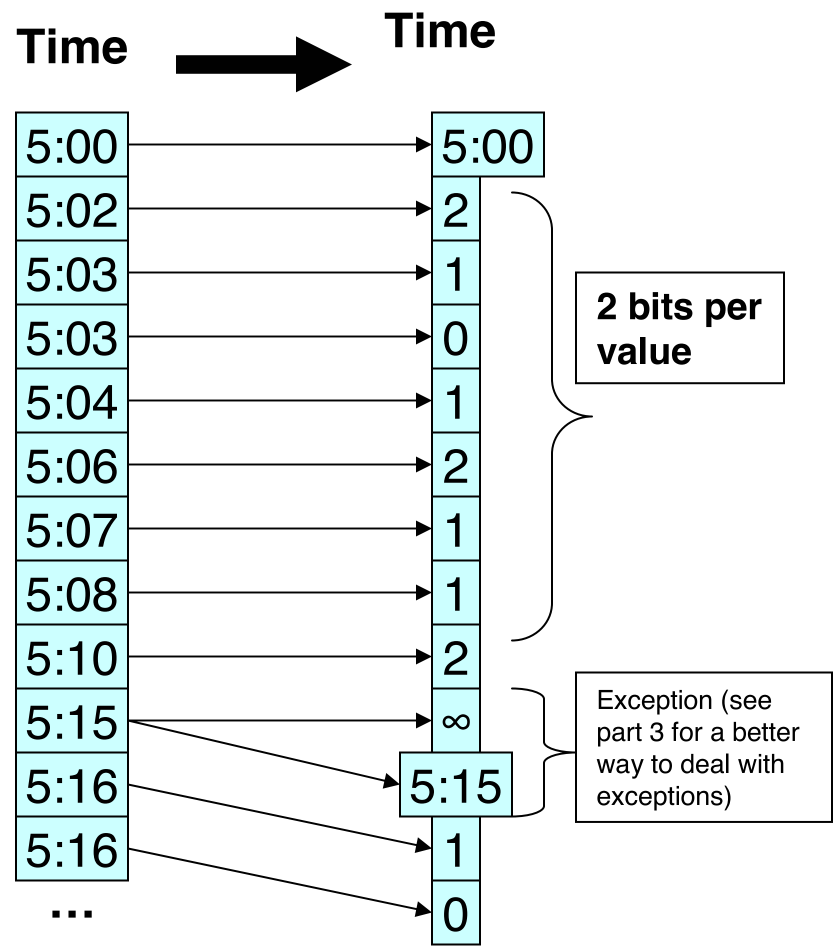


"Compressing Relations and Indexes" Goldstein, Ramakrishnan, Shaft, ICDE'98

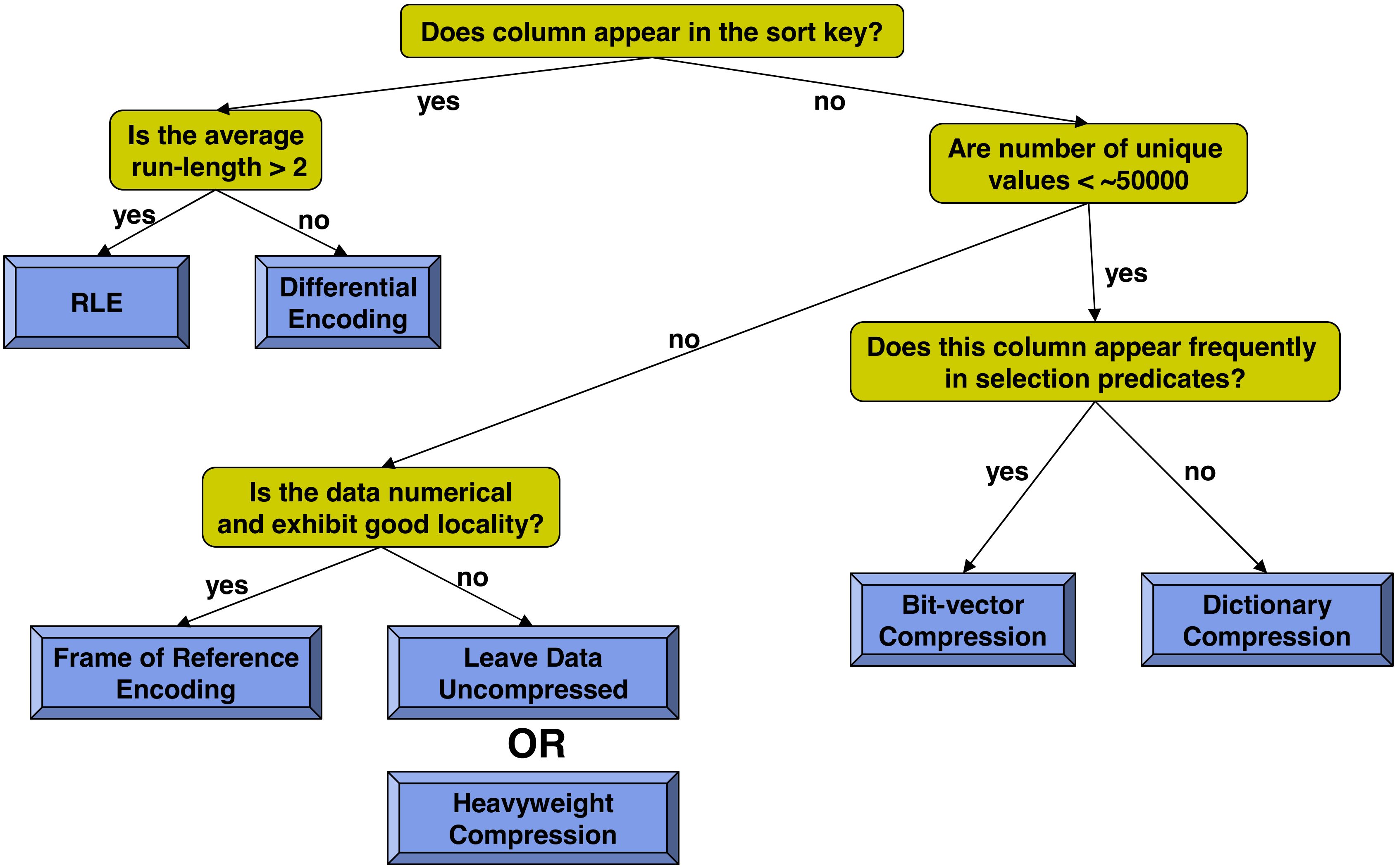
Differential Encoding

- **Encodes values as b bit offset from previous value**
- **Special escape code (just like frame of reference encoding) indicates a difference larger than can be stored in b bits**
  - **After escape code, original (uncompressed) value is written**
- **Performs well on columns containing increasing/decreasing sequences**
  - **inverted lists**
  - **timestamps**
  - **object IDs**
  - **sorted / clustered columns**

"Improved Word-Aligned Binary Compression for Text Indexing" Ahn, Moffat, TKDE'06



What Compression Scheme To Use?

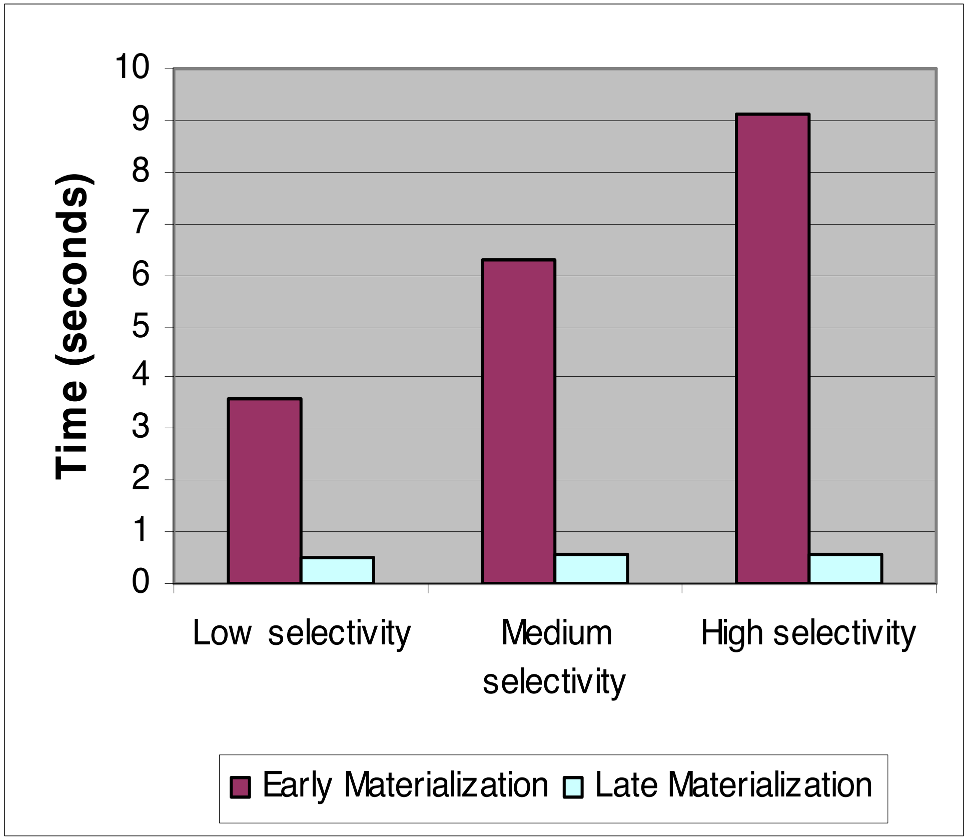


When should columns be projected?

- Where should column projection operators be placed in a query plan?
  - Row-store:
    - Column projection involves removing unneeded columns from tuples
    - Generally done as early as possible
  - Column-store:
    - Operation is almost completely opposite from a row-store
    - Column projection involves reading needed columns from storage and extracting values for a listed set of tuples
      - This process is called "materialization"
    - Early materialization: project columns at beginning of query plan
      - Straightforward since there is a one-to-one mapping across columns
    - Late materialization: wait as long as possible for projecting columns
      - More complicated since selection and join operators on one column obfuscates mapping to other columns from same table
    - Most column-stores construct tuples and column projection time
      - Many database interfaces expect output in regular tuples (rows)
      - Rest of discussion will focus on this case

"Materialization Strategies in a Column-Oriented DBMS" Abadi, Myers, DeWitt, and Madden. ICDE 2007.

For plans without joins, late materialization is a win



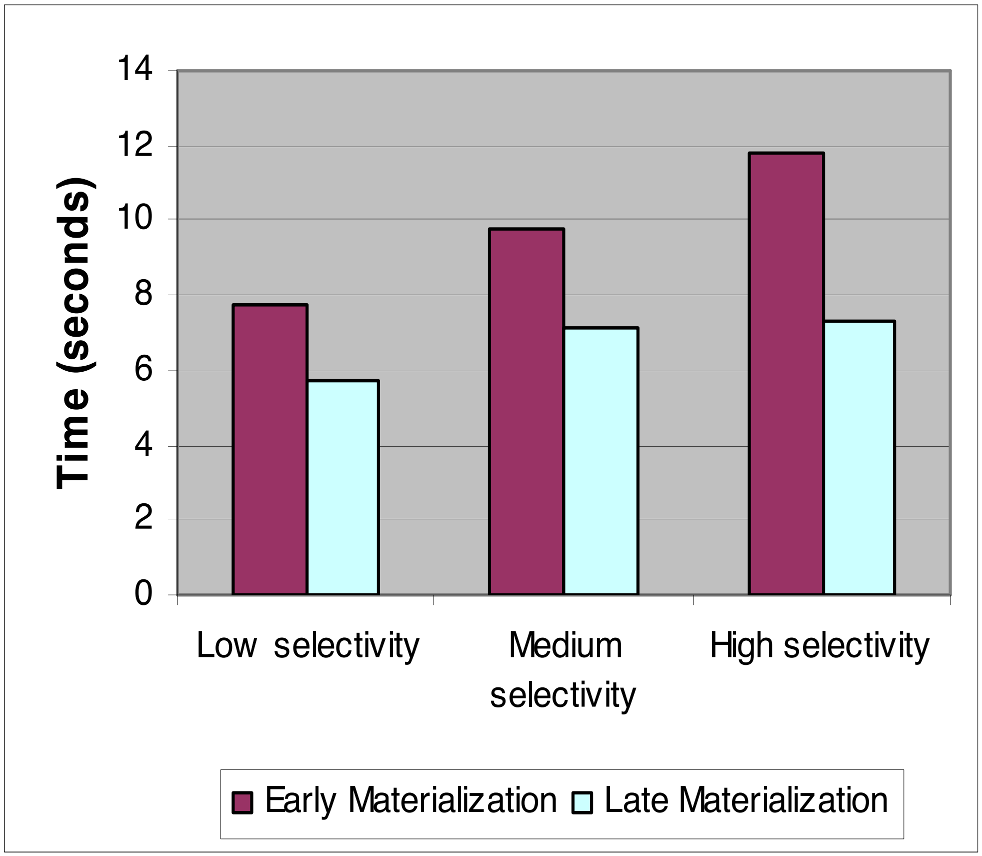
QUERY:

```
SELECT C1, SUM(C2)
FROM table
WHERE (C1 < CONST) AND
      (C2 < CONST)
GROUP BY C1
```

- **Ran on 2 compressed columns from TPC-H scale 10 data**

"Materialization Strategies in a Column-Oriented DBMS"
Abadi, Myers, DeWitt, and Madden. ICDE 2007.

Even on uncompressed data, late materialization is still a win



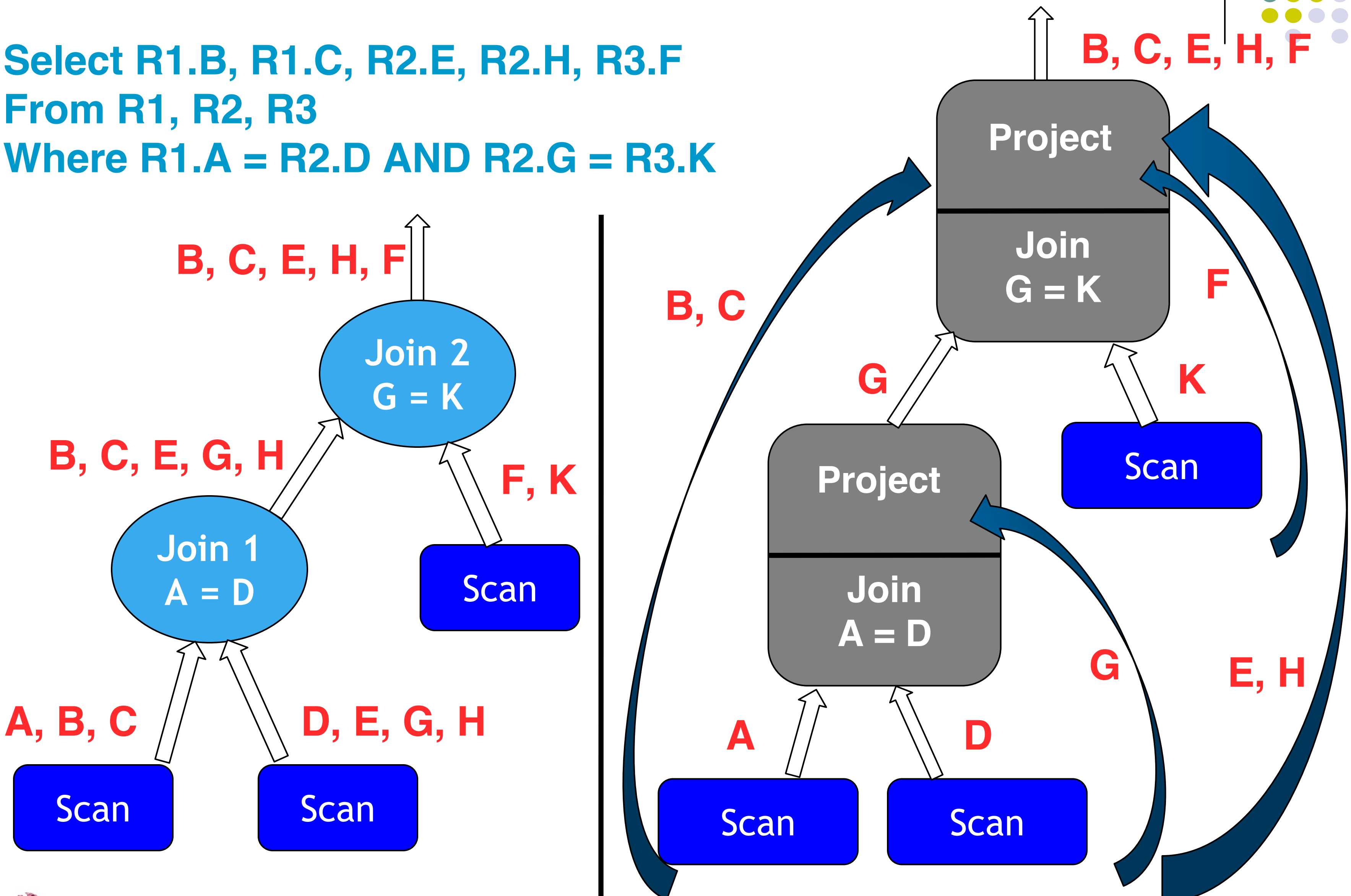
QUERY:

```
SELECT C1, SUM(C2)
FROM table
WHERE (C1 < CONST) AND
      (C2 < CONST)
GROUP BY C1
```

- **Materializing late still works best**

What about for plans with joins?

Select R1.B, R1.C, R2.E, R2.H, R3.F
From R1, R2, R3
Where R1.A = R2.D AND R2.G = R3.K



What about for plans with joins?

Select R1.B, R1.C, R2.E, R2.H, R3.F
From R1, R2, R3
Where R1.A = R2.D AND R2.G = R3.K