

Index SQL

BORN TO B-TREE

WHY !?



BUT YOU'RE SO



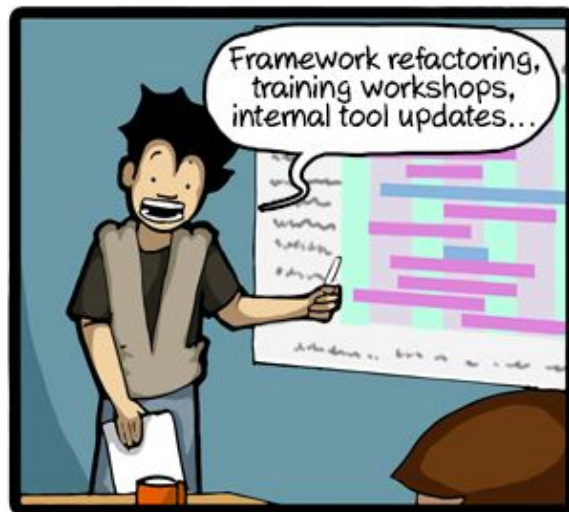
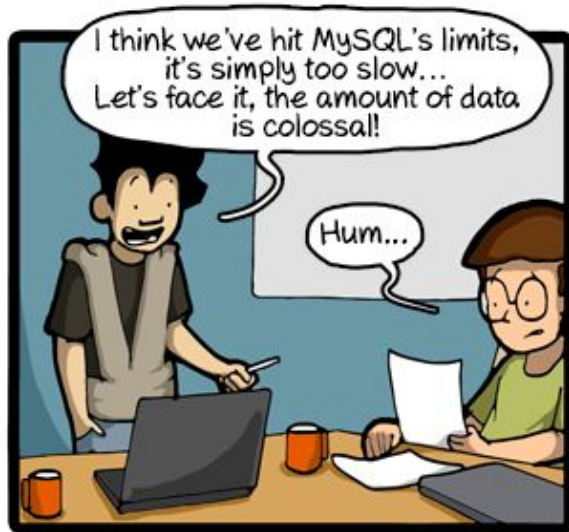
OLD????????

History

	SQL	MySQL
Creation	1986 (1974)	1995
Last Version	2011 http://modern-sql.com	2015 (5.7)

SQL vs NoSQL

Fight !





Damned SQL !

“Cover up that SQL, which I can’t endure to look on.”

Application

Framework

ORM

DBAL

SQL

Law of Leaky Abstraction

“All non-trivial abstractions, to some degree, are leaky.”

<http://www.joelonsoftware.com/articles/LeakyAbstractions.html>

J. Spolsky, 11/11/2002



Application

Framework

ORM

DBAL

SQL



YOU'RE DOING IT WRONG.

No matter how hard you try, it is impossible to fax a cat.



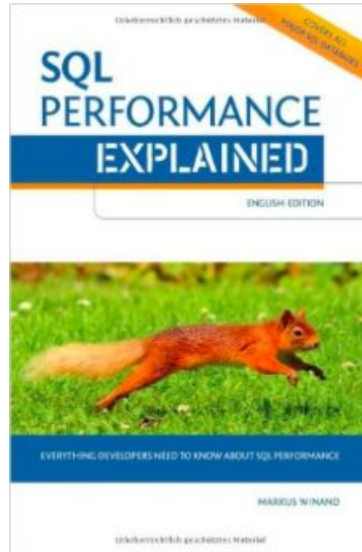
UNDERSTAND SQL

memegenerator.net



USE THE INDEX, LUKE

A guide to database performance for developers



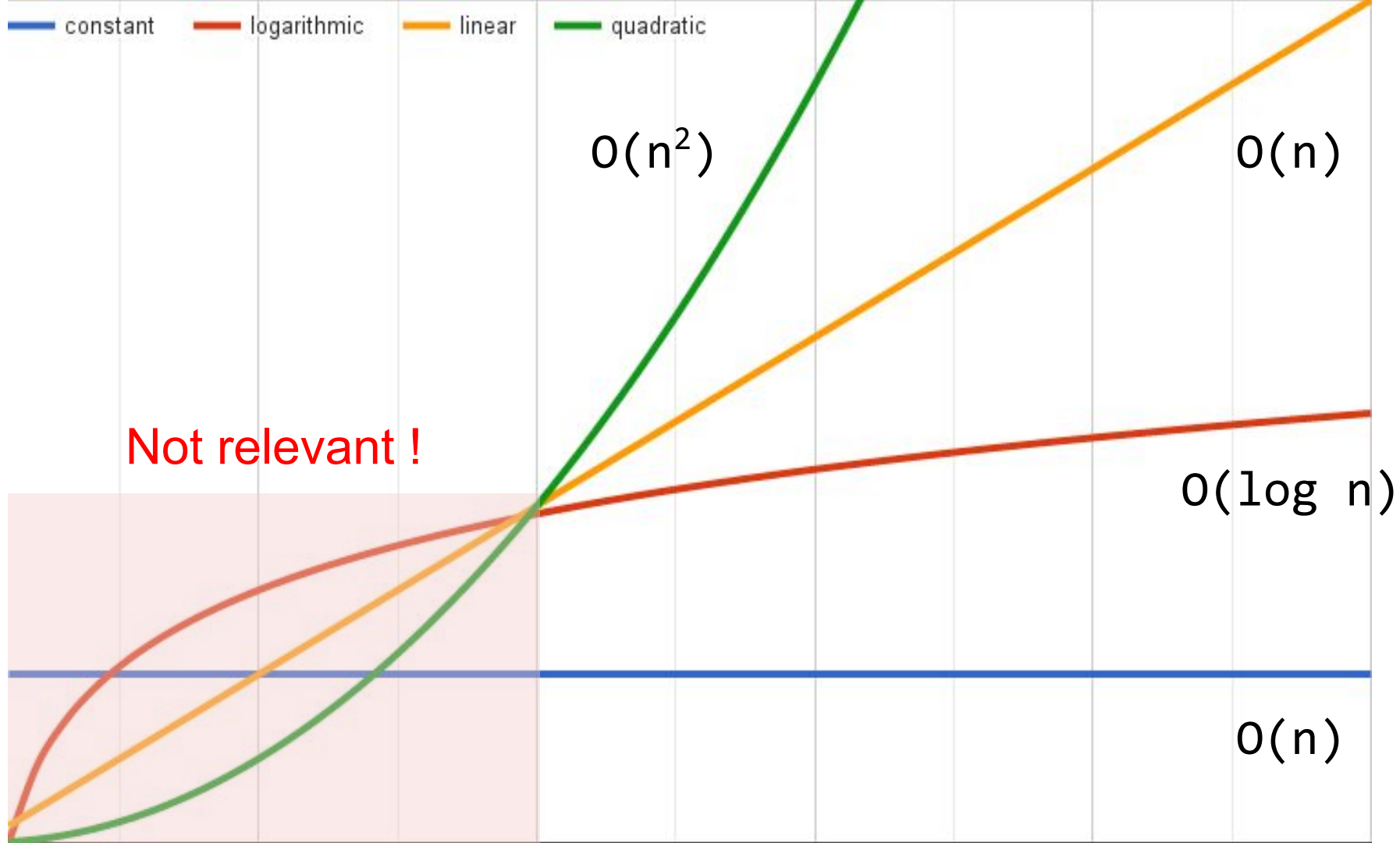
<http://use-the-index-luke.com>



<http://coding-geek.com/how-databases-work/>



WHAT ABOUT COMPLEXITY?



Tables

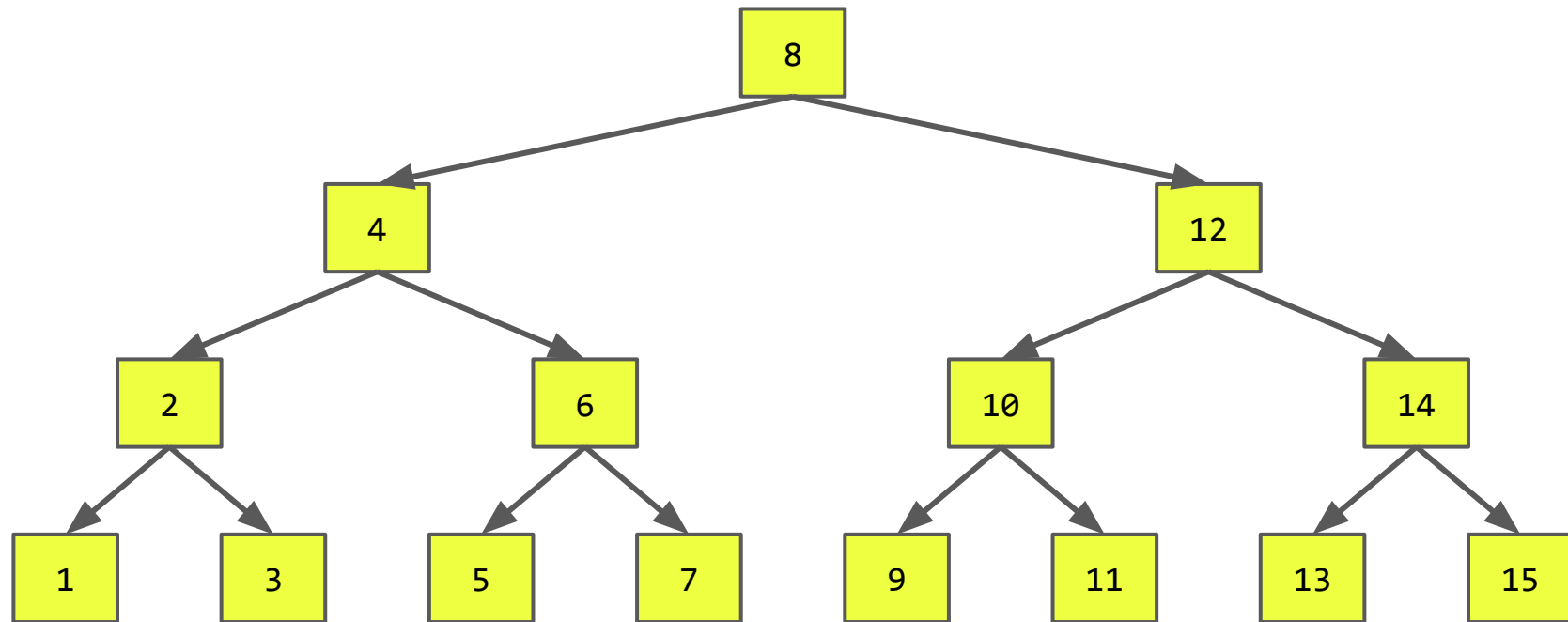
Back to basics...

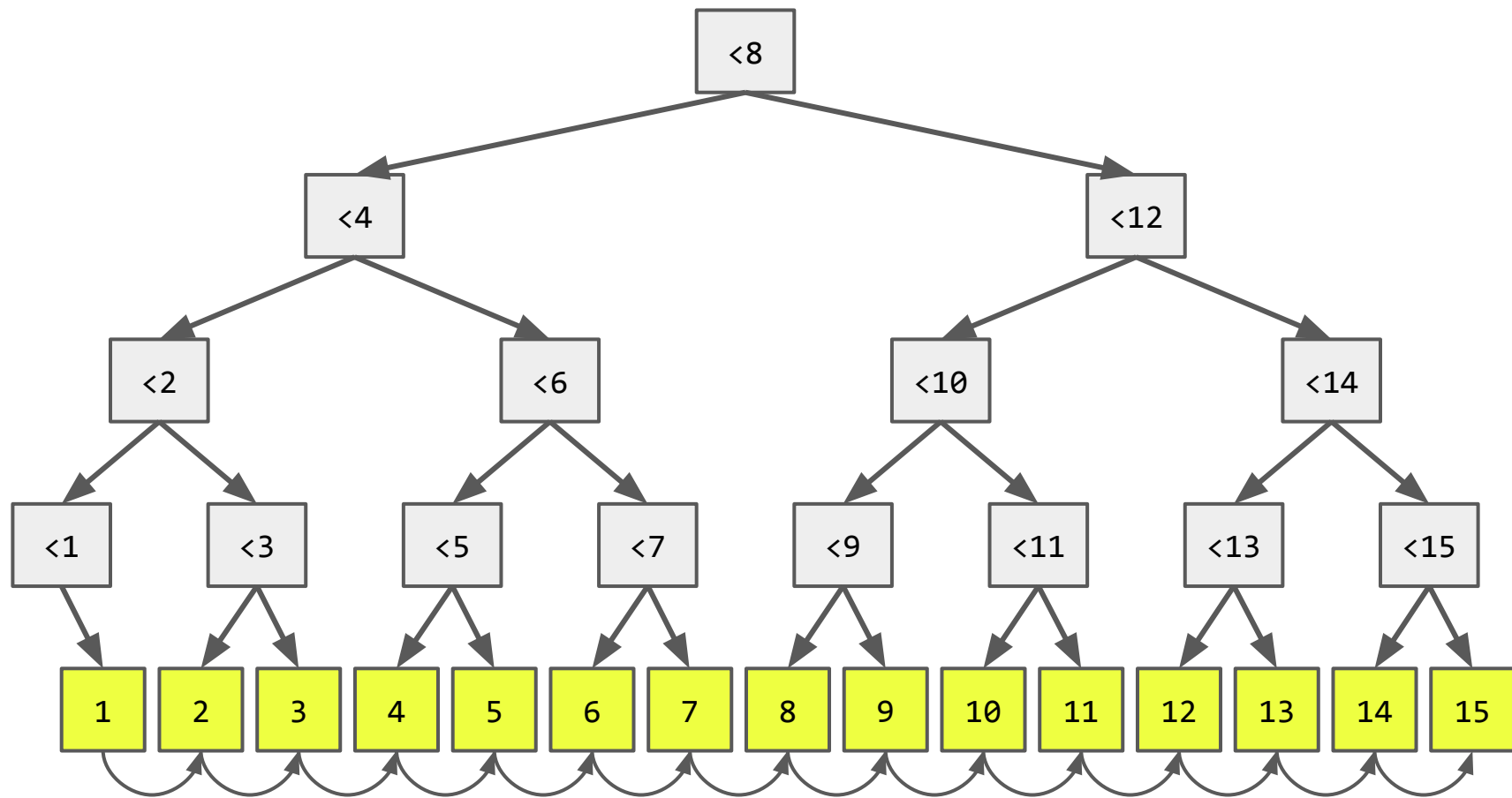
@	ID	NAME	DATE	DATA...
0x00	12	ABC	01/01/1970	...
0x01	1	BCD	02/02/1970	...
0x02	3	CDE	03/03/1970	...
0x03	8	DEF	04/04/1970	...
0x04	15	EFG	05/05/1970	...
0x05	4	FGH	06/06/1970	...
0x06	10	GHI	07/07/1970	...
0x07	6	HIJ	08/08/1970	...

Search = Full Table Scan
 $O(n)$

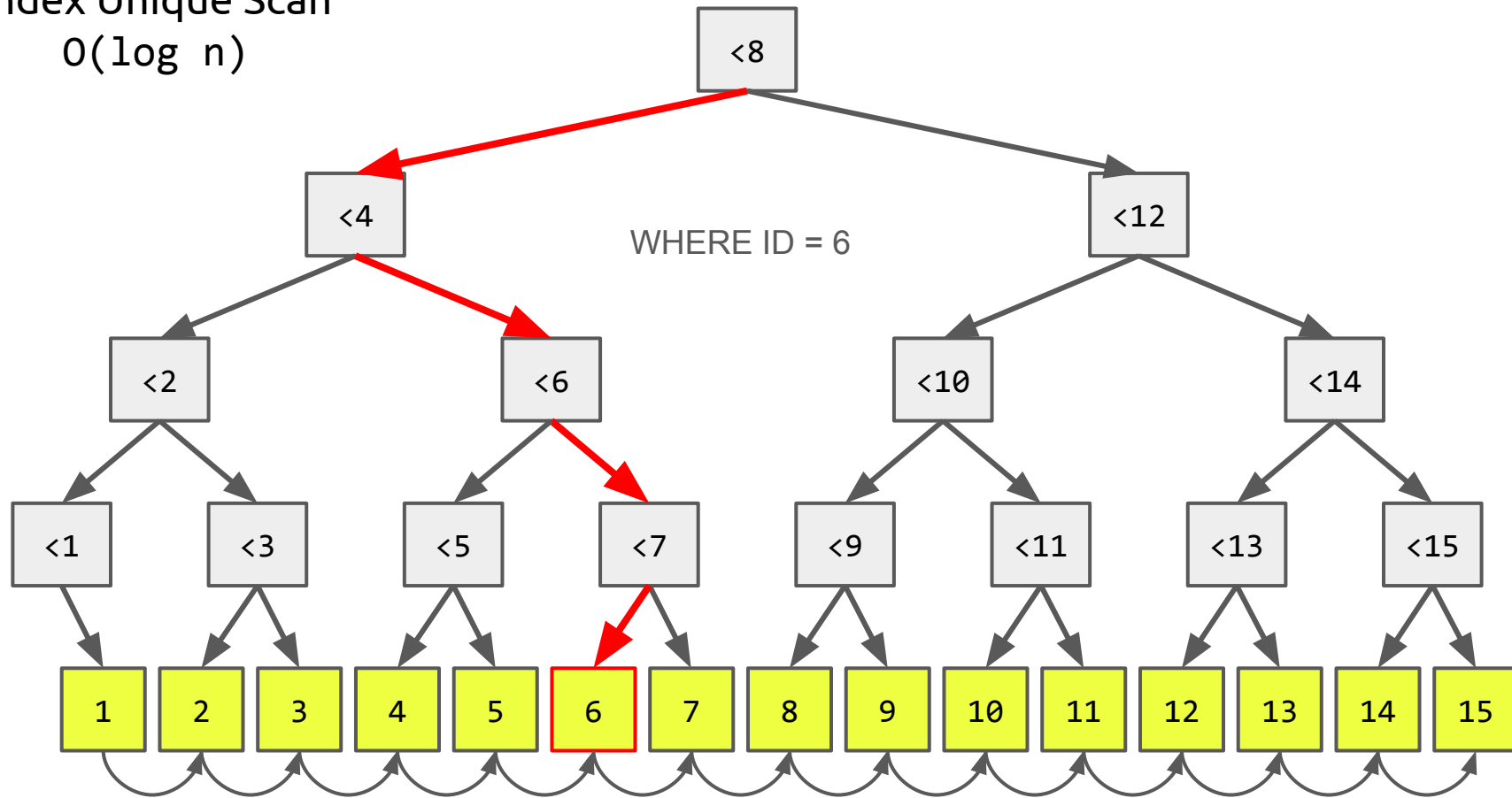
B-Tree

...or not to be...

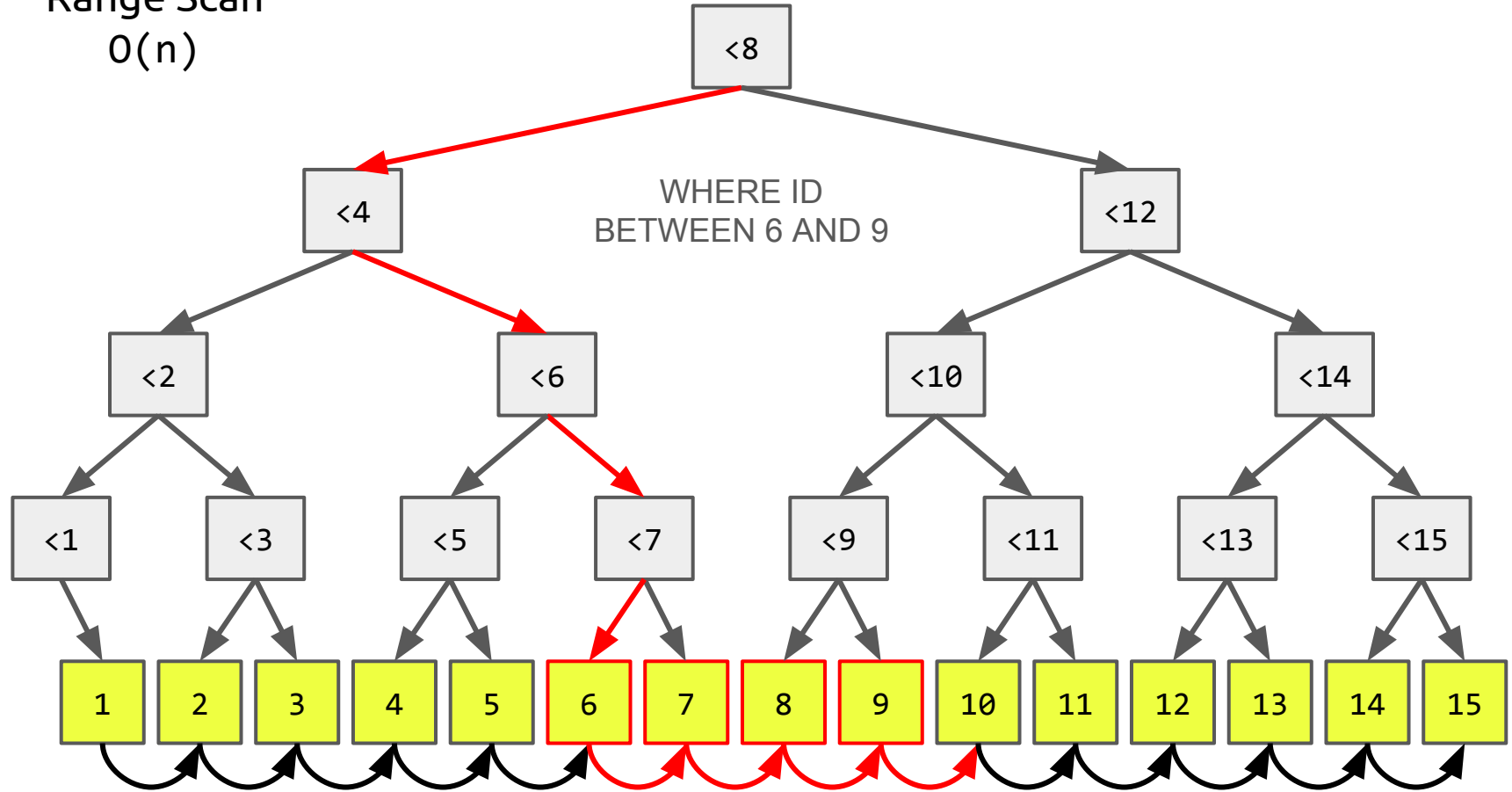




Index Unique Scan
 $O(\log n)$



Range Scan
 $O(n)$



Index Unique Scan $O(\log n)$

```
SELECT * FROM table WHERE id = ?
```

```
SELECT * FROM table WHERE id != ?
```

```
SELECT * FROM table WHERE id + 10 = ?
```


Range Scan $O(n)$

```
SELECT * FROM table WHERE id BETWEEN ? AND ?
```

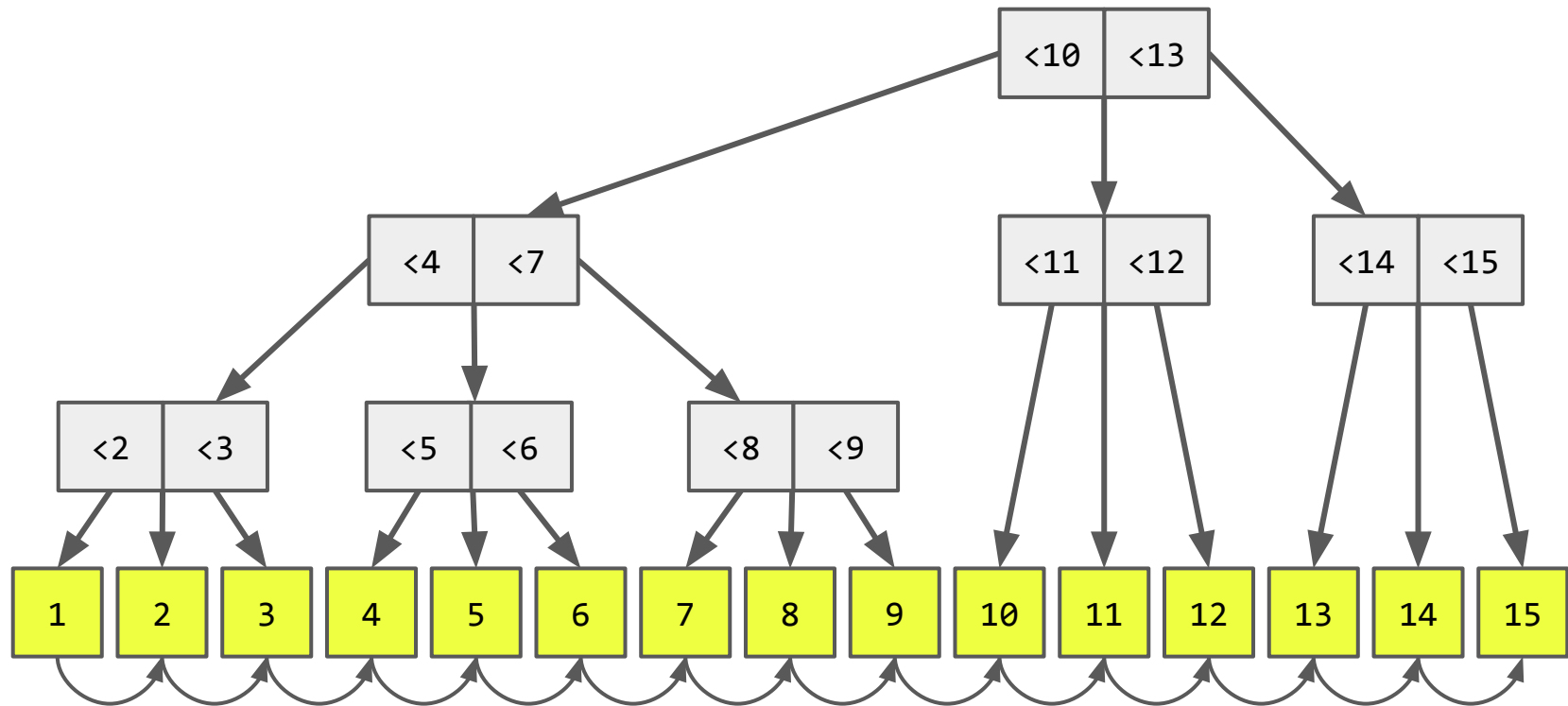
```
SELECT * FROM table WHERE id > ? OR data = ?
```

```
SELECT * FROM table WHERE id > 0
```

More nodes !

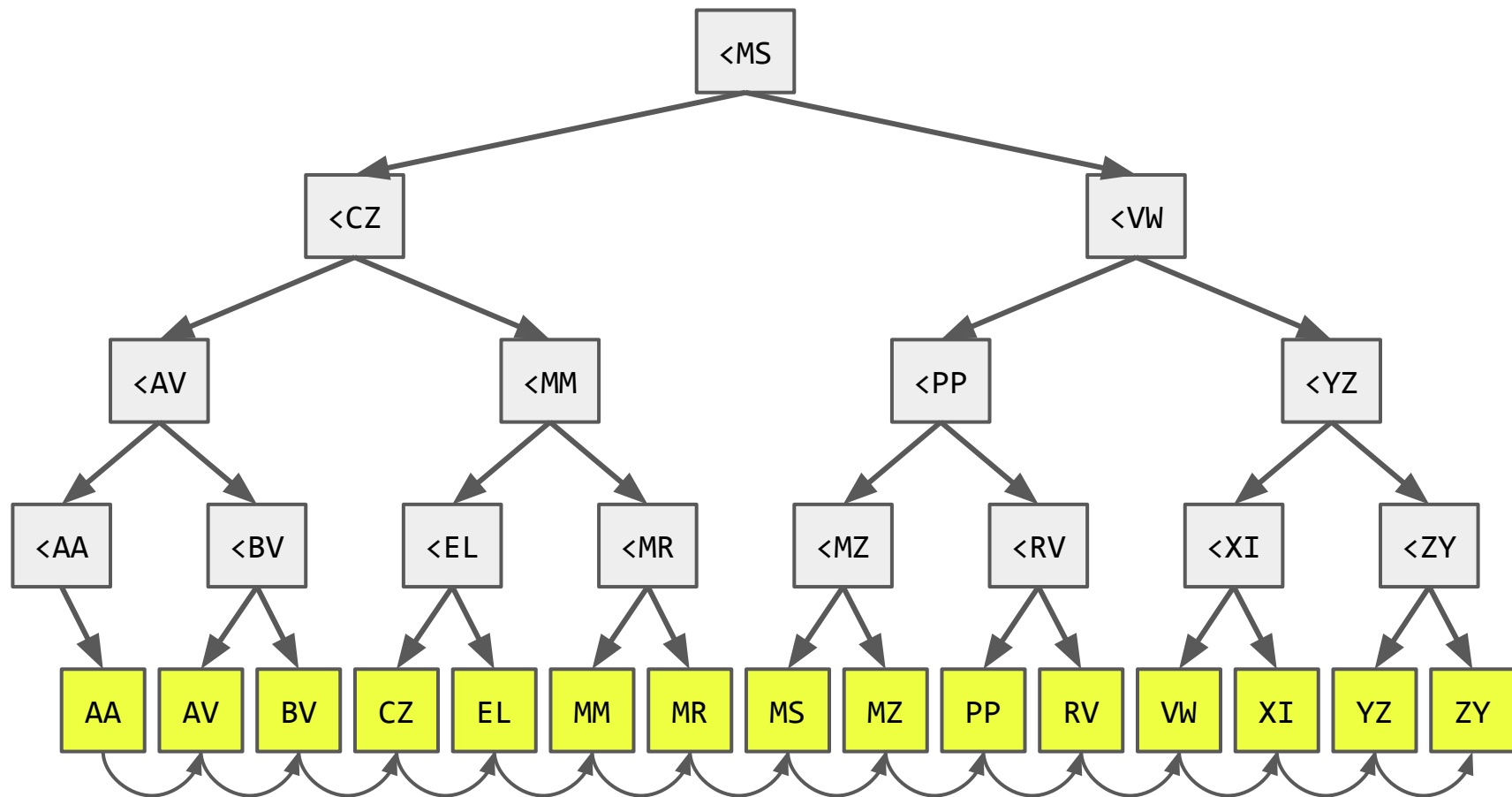
'B' does not mean "Binary"

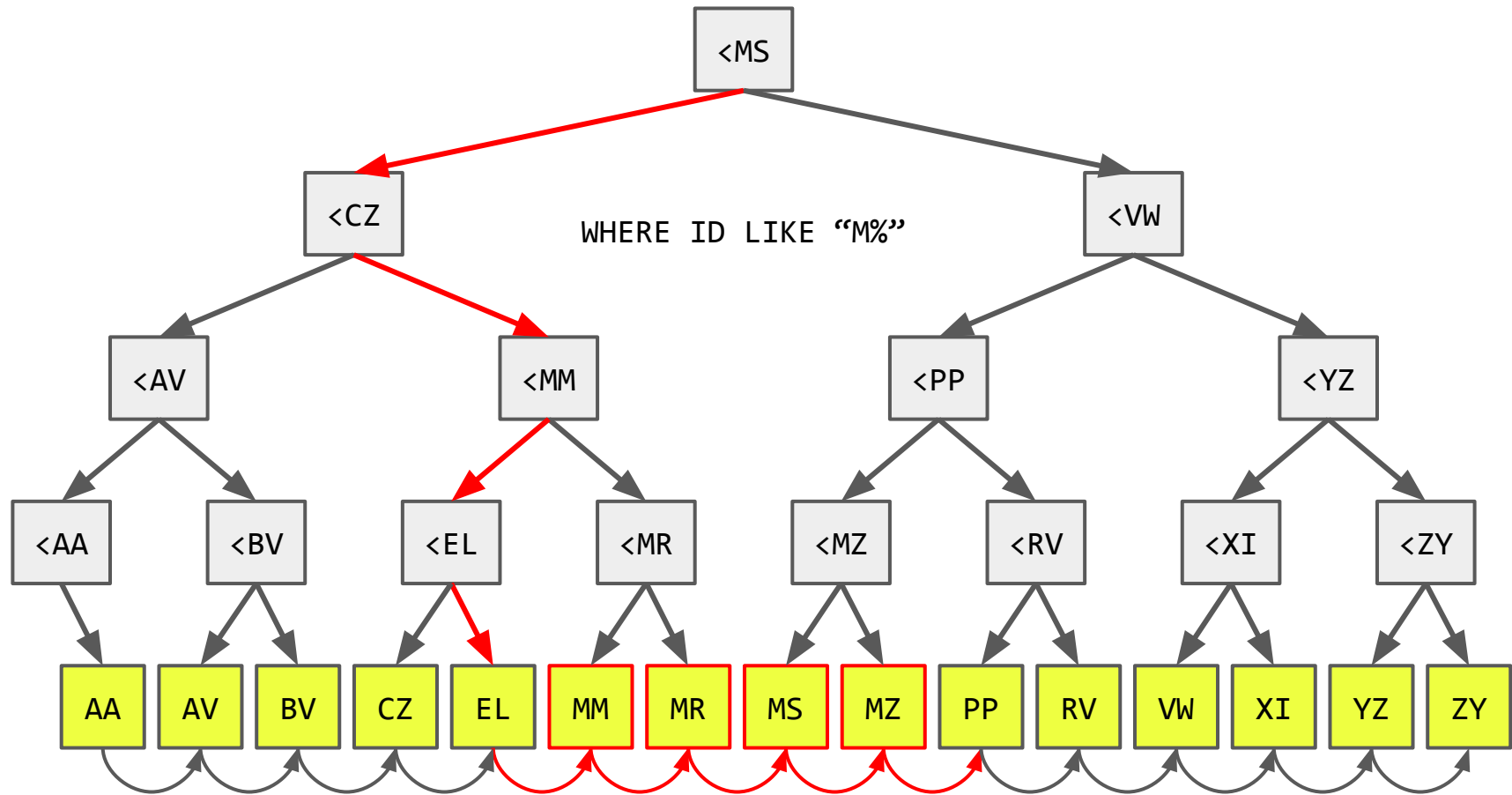
$\log_3 n$ VS $\log_2 n \Rightarrow O(\log n)$

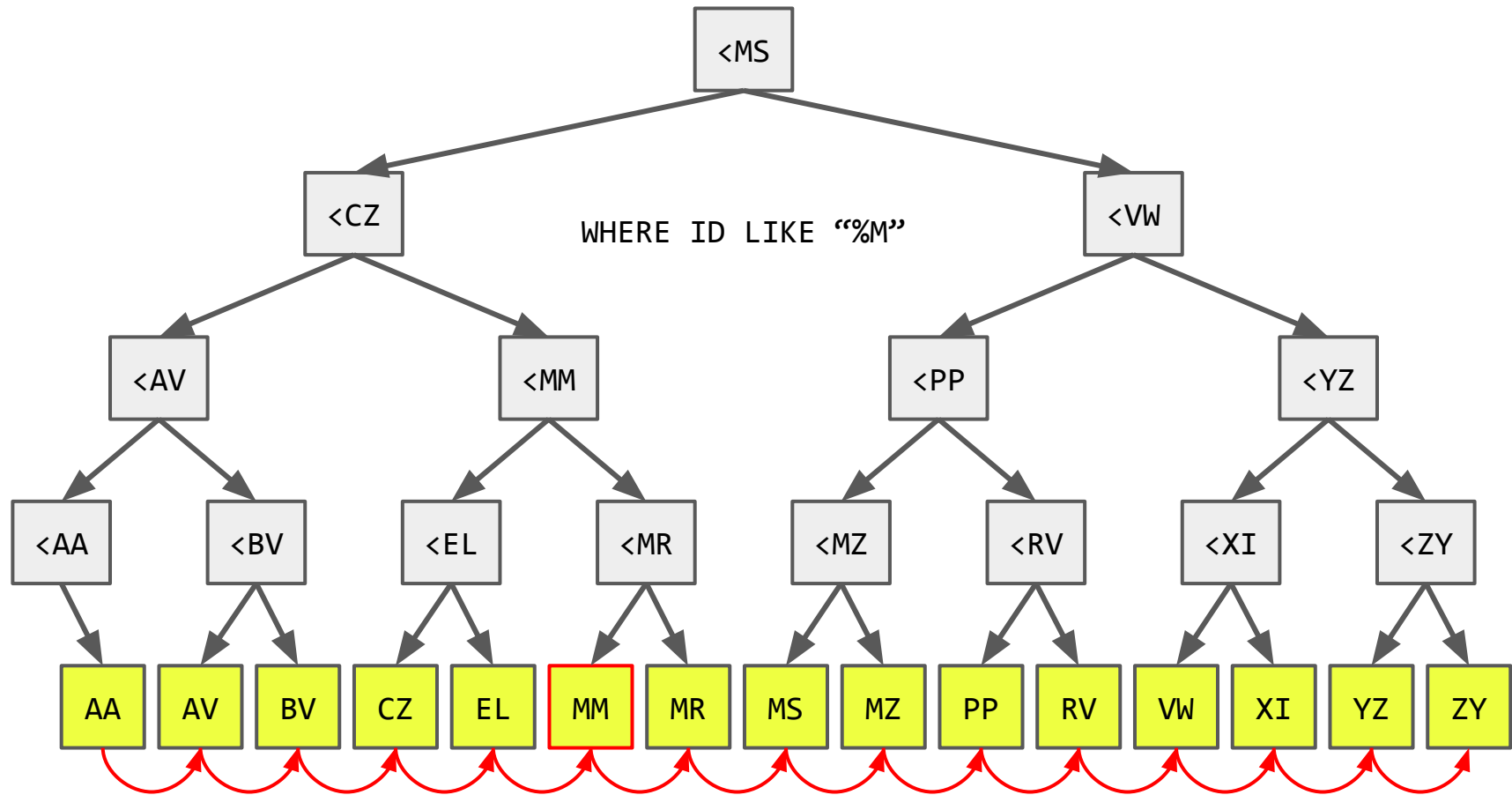


Text Search

LIKE operator







Range Scan $O(n)$

```
SELECT * FROM table WHERE text LIKE 'abc%'
```

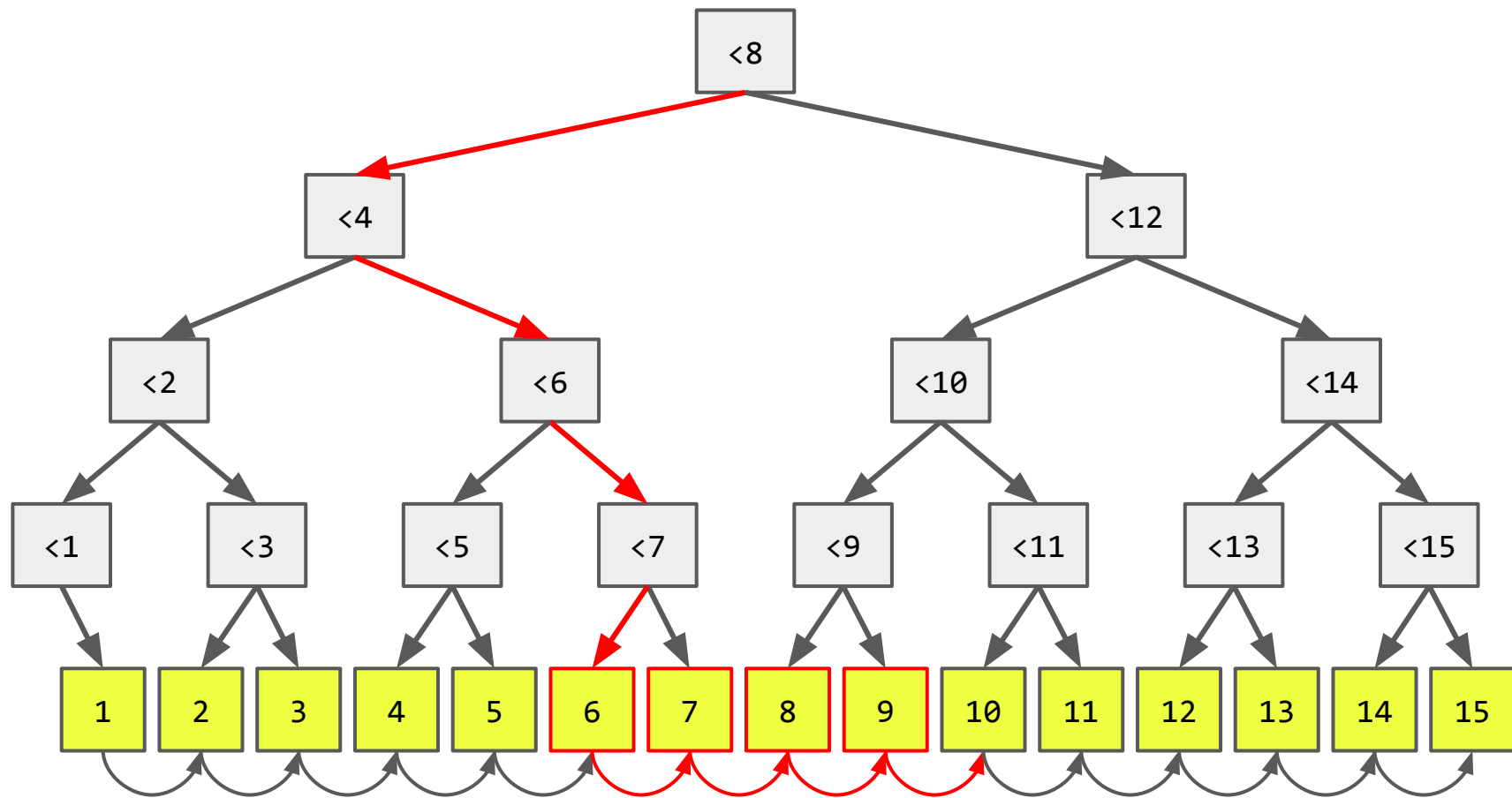
```
SELECT * FROM table WHERE text LIKE 'abc%fg'
```

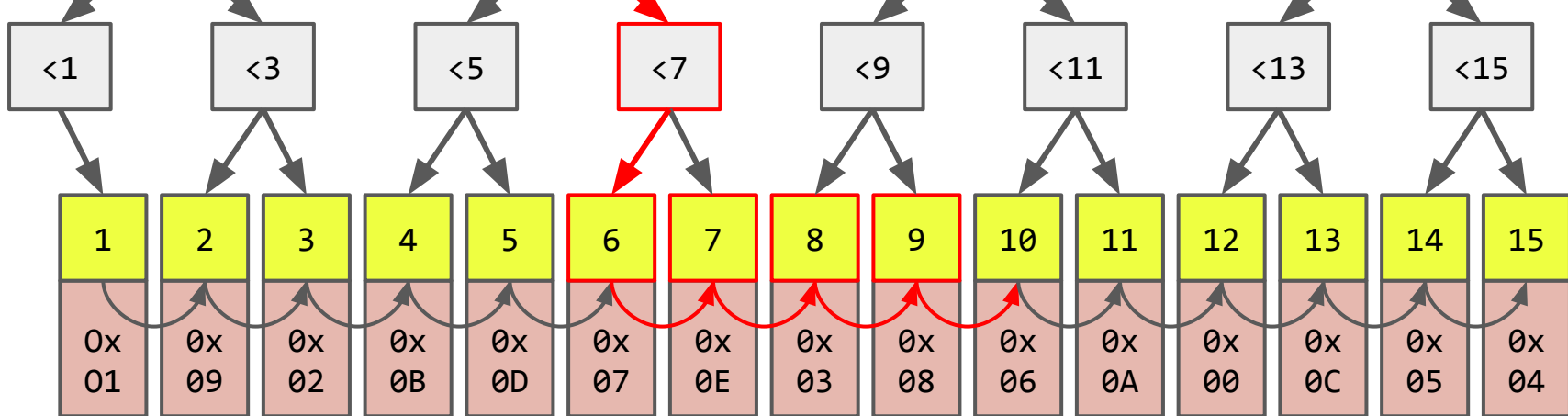
```
SELECT * FROM table WHERE text LIKE '%abc%'
```

```
SELECT * FROM table WHERE UPPER(text) = 'ABC'
```


Data access

The *hidden* cost for additional columns





@	ID	DATA...
0x00	12	...
0x01	1	...
0x02	3	...
0x03	8	...
0x04	15	...

@	ID	DATA...
0x05	14	...
0x06	10	...
0x07	6	...
0x08	9	...
0x09	2	...

@	ID	DATA...
0x0A	11	...
0x0B	4	...
0x0C	13	...
0x0D	5	...
0x0E	7	...

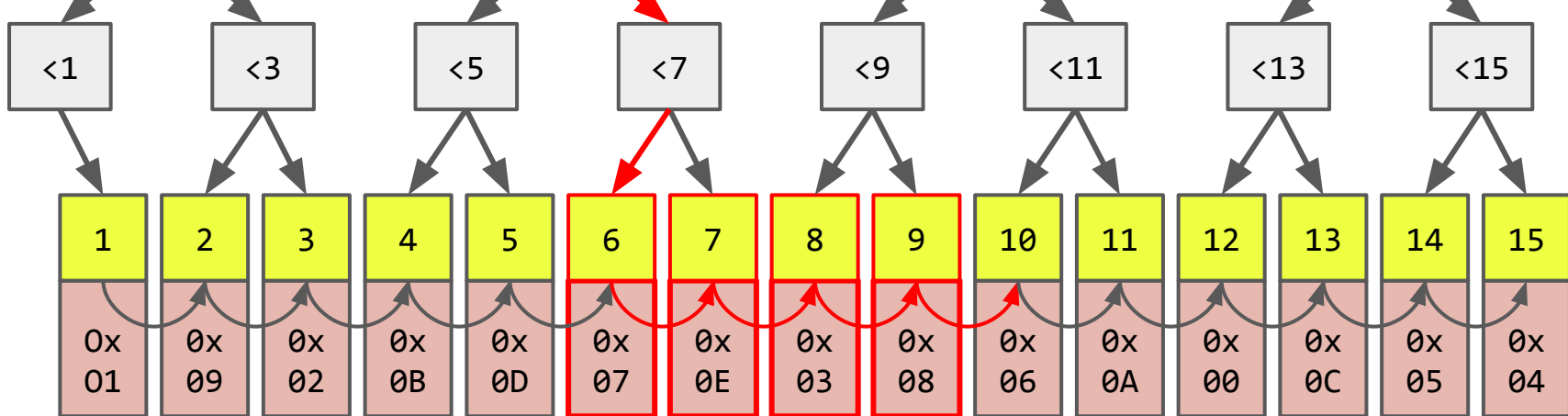


Table access by RowID $O(1)$

@	ID	DATA...
0x00	12	...
0x01	1	...
0x02	3	...
0x03	8	...
0x04	15	...

@	ID	DATA...
0x05	14	...
0x06	10	...
0x07	6	...
0x08	9	...
0x09	2	...

@	ID	DATA...
0x0A	11	...
0x0B	4	...
0x0C	13	...
0x0D	5	...
0x0E	7	...

Clustering

Index-Only scan

Index Only Scan $O(\log n)$

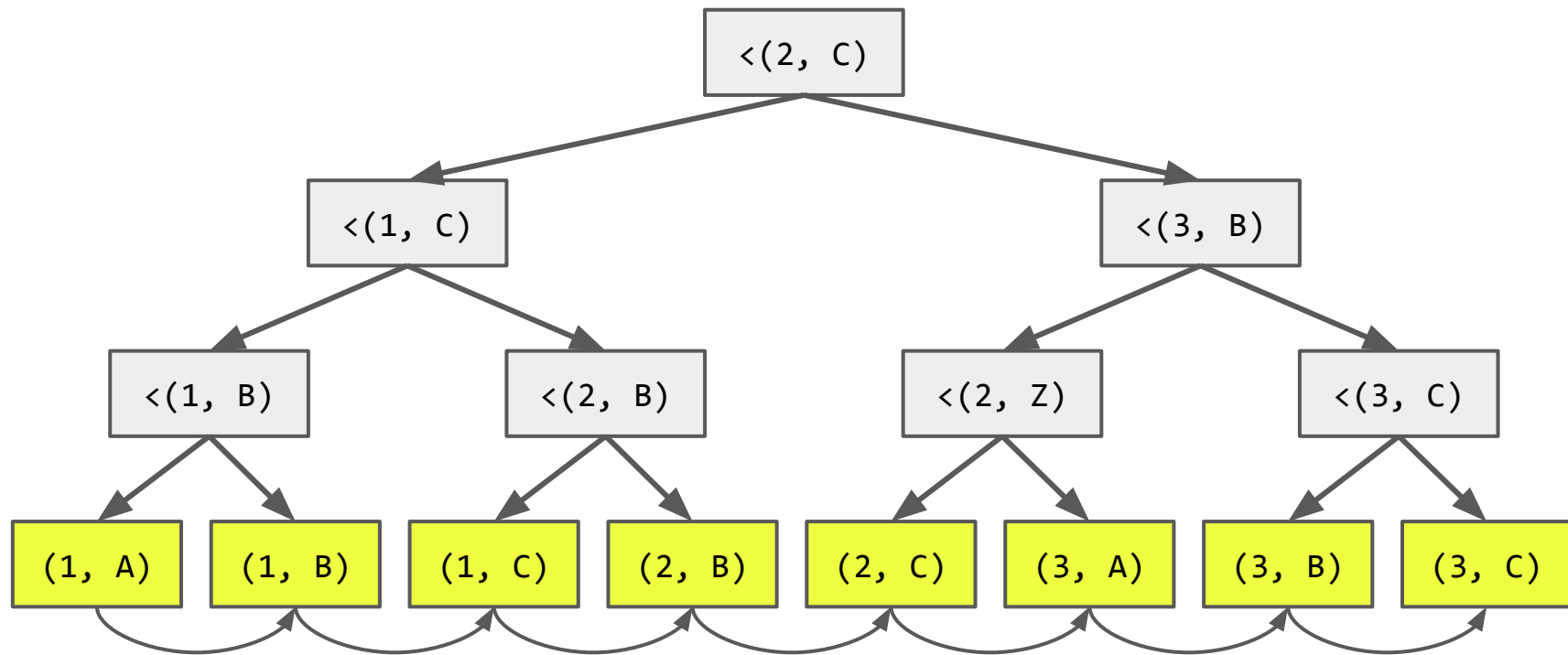
```
SELECT id FROM table WHERE id = ?
```

```
SELECT * FROM table WHERE id = ?
```

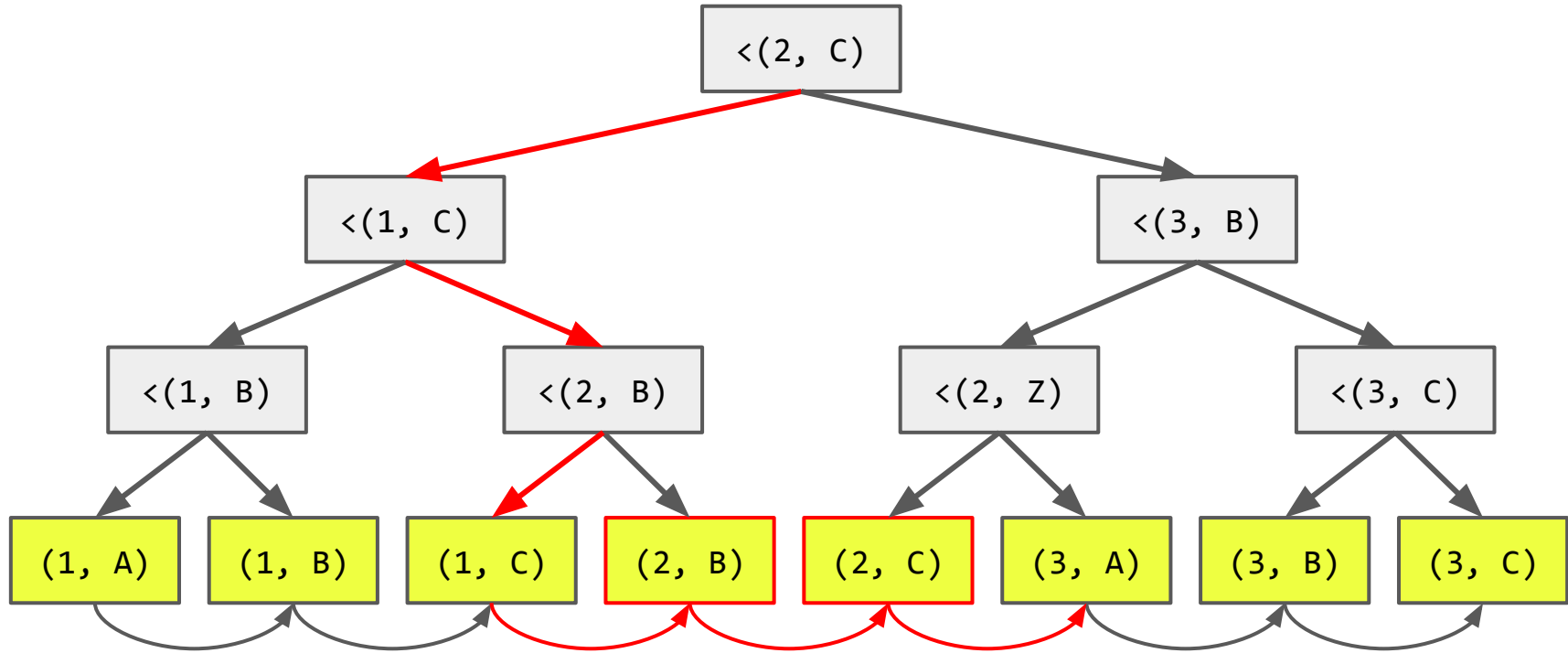
Multi-column index

Order matters !

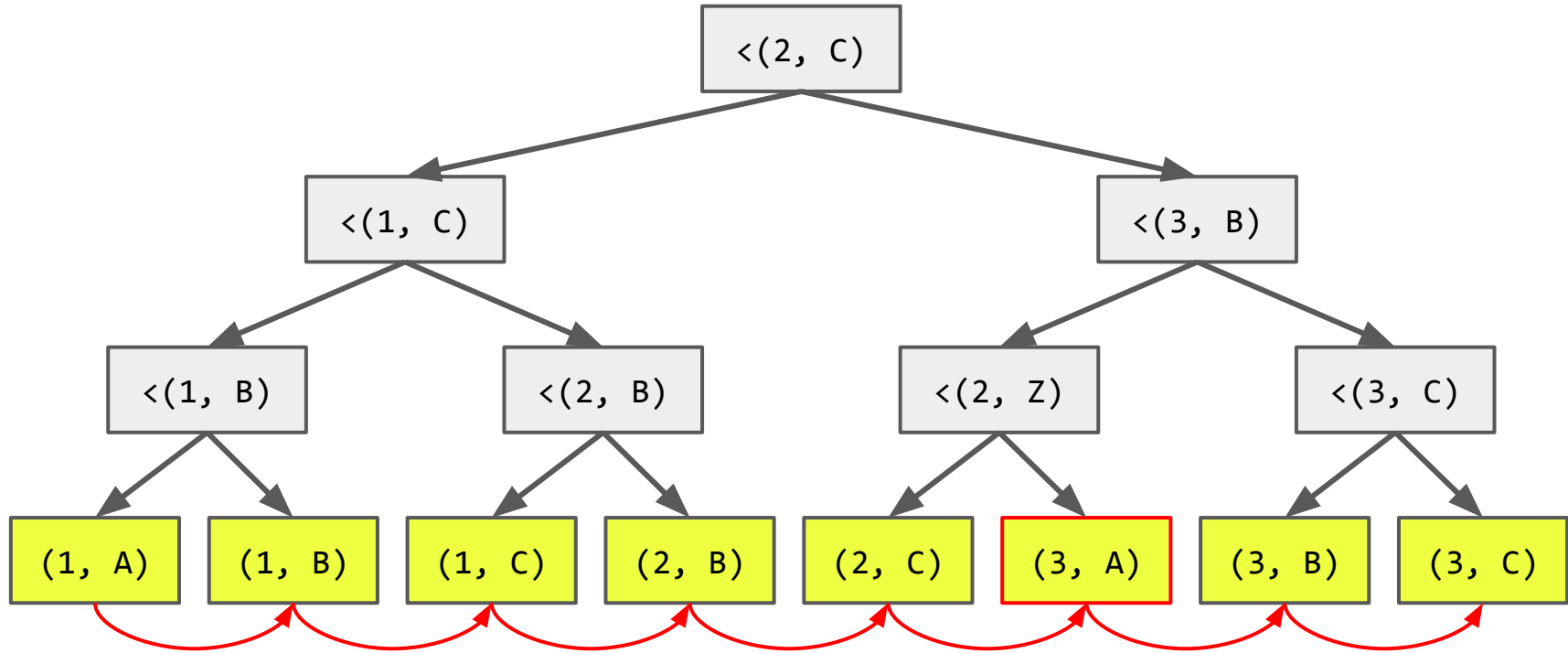




WHERE ID1 = 2 AND ID2 >= A



WHERE ID1 >= 2 AND ID2 = A



Range Scan $O(n)$

```
SELECT * FROM table WHERE id1 = ?
```

```
SELECT * FROM table WHERE id1 = ? AND id2 > ?
```

```
SELECT * FROM table WHERE id2 = ?
```

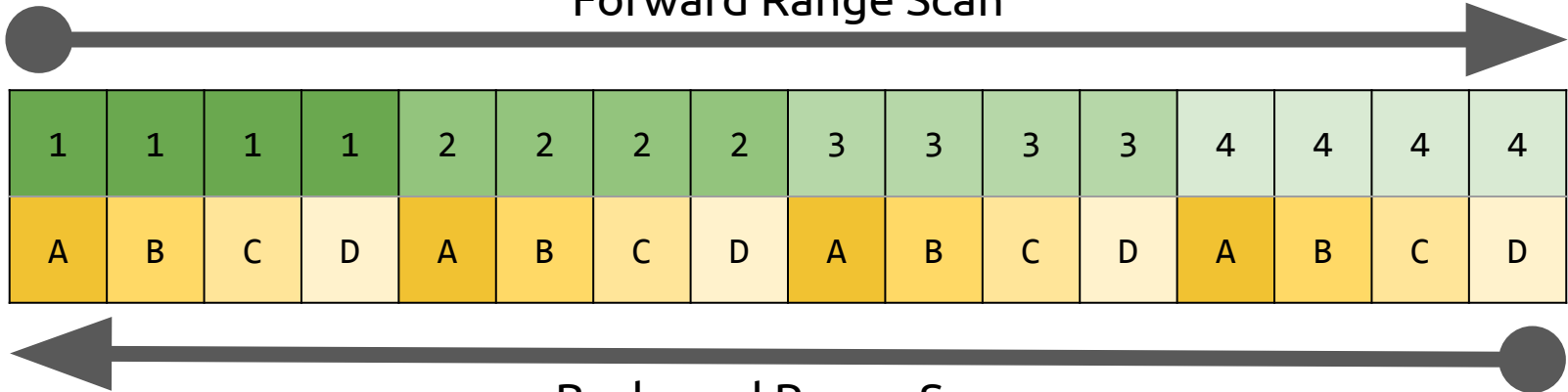
```
SELECT * FROM table WHERE id1 > ? AND id2 = ?
```

Sorting

Use the index, luke !

ORDER BY ID1 **ASC**, ID2 **ASC**

Forward Range Scan



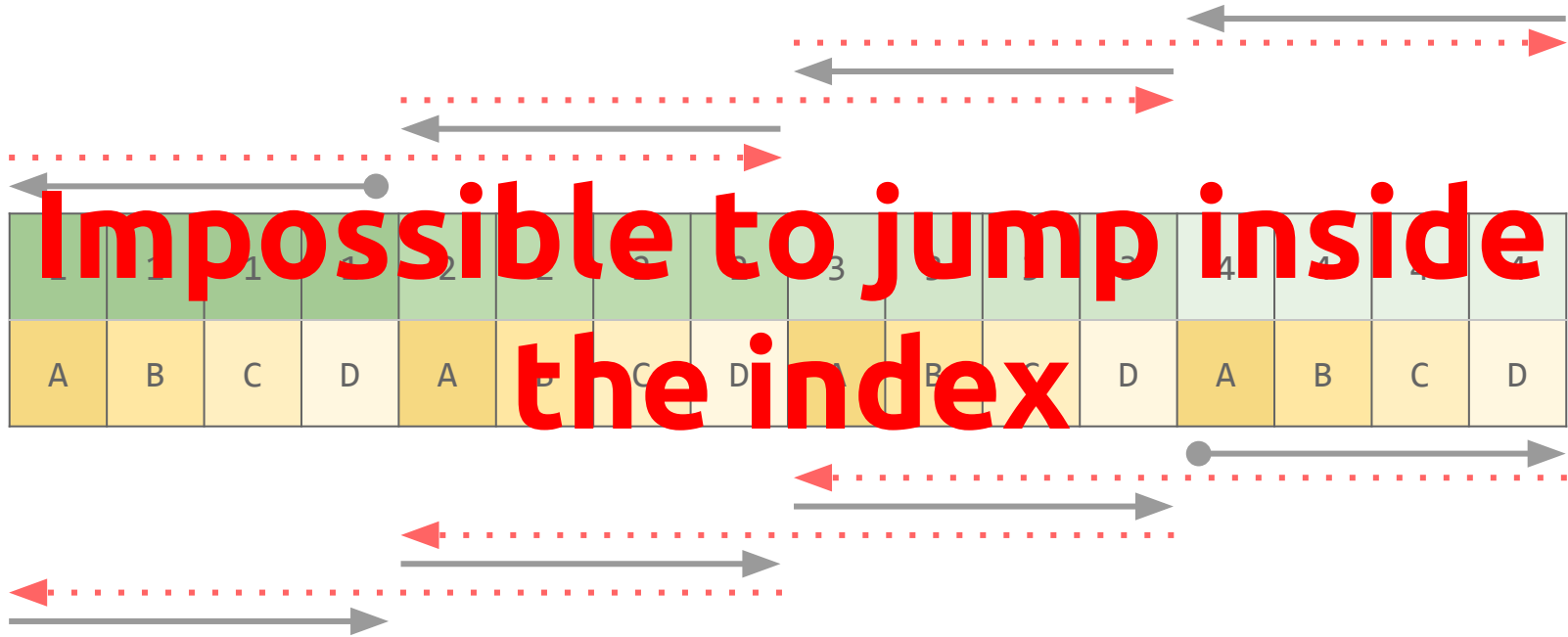
The diagram illustrates a Forward Range Scan. A horizontal arrow points from left to right, starting with a solid black circle on the left and ending with a solid black triangle on the right. Below this arrow is a table with 16 columns and 2 rows. The top row contains IDs (1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4) and the bottom row contains labels (A, B, C, D, A, B, C, D, A, B, C, D, A, B, C, D). The IDs are grouped by color: dark green for 1, medium green for 2, light green for 3, and very light green for 4. The labels are grouped by color: orange for A and yellow for B, C, and D. The scan direction is indicated by the arrow pointing from left to right.

1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4
A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D

Backward Range Scan

ORDER BY ID1 **DESC**, ID2 **DESC**

ORDER BY ID1 ASC, ID2 DESC



ORDER BY ID1 DESC, ID2 ASC

Grouping

Use the sorting, luke !

Top-N Queries

Avoid full table scan

Offset

Pagination helper



IT'S A TRAP !

Index cannot be used for an offset

1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4
A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D

● —————> LIMIT 0,3

●> —————> LIMIT 3,3

●> —————> LIMIT 6,3

●> —————> LIMIT 9,3

BRACE YOURSELF



JOINS ARE COMING

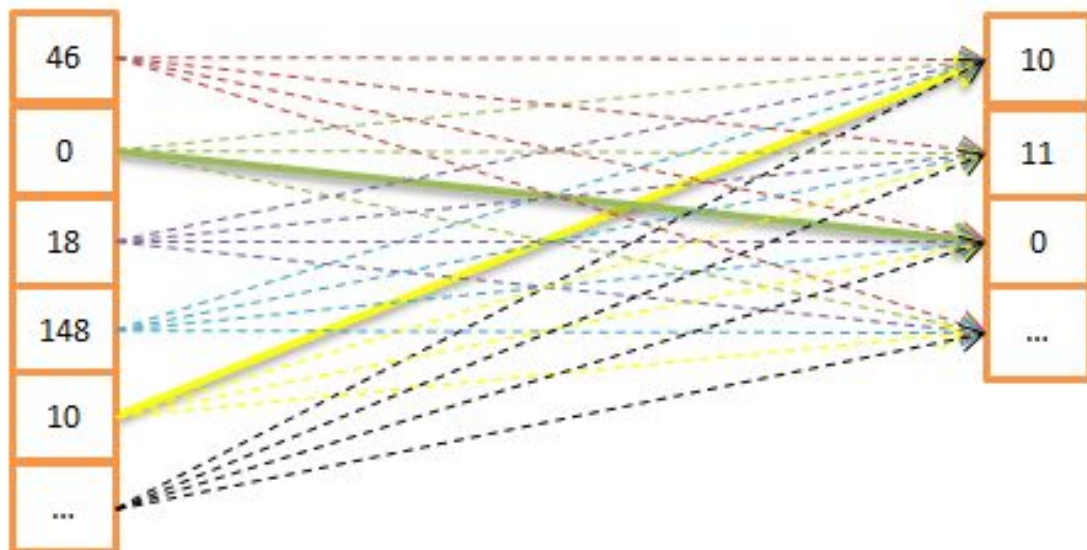
memegenerator.net

Join

Combinatorial power !



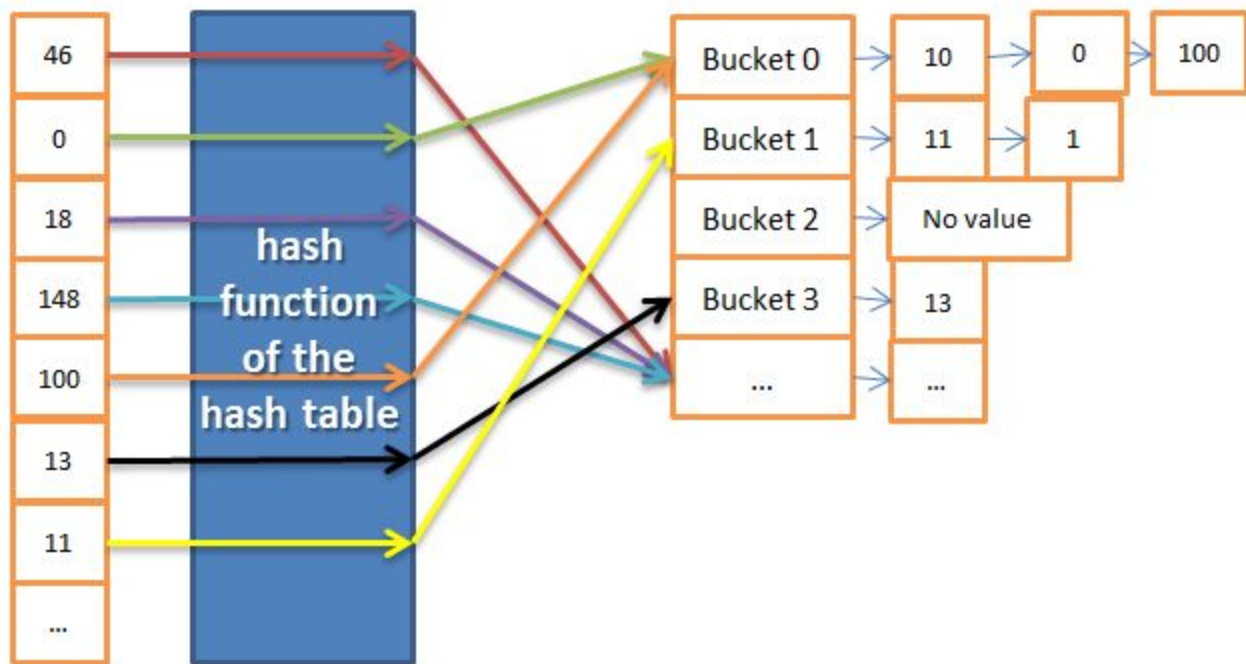
Nested Loop Join



Outer relation

Inner relation

Hash Join



Outer relation

Inner relation (in-memory hash table)

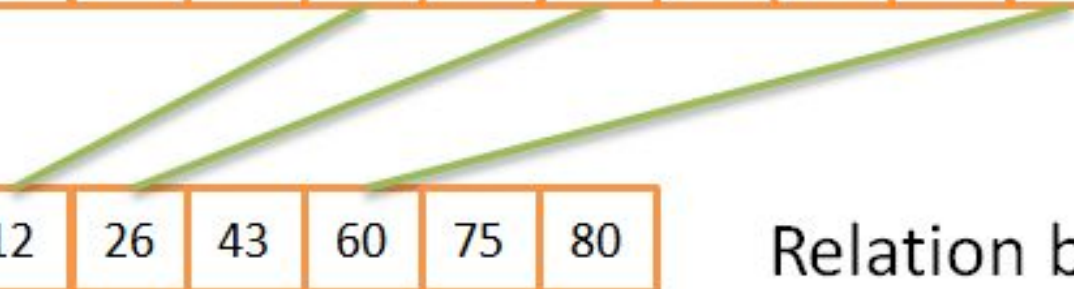
Merge Join

1	6	10	11	12	24	26	30	31	40	60	70
---	---	----	----	----	----	----	----	----	----	----	----

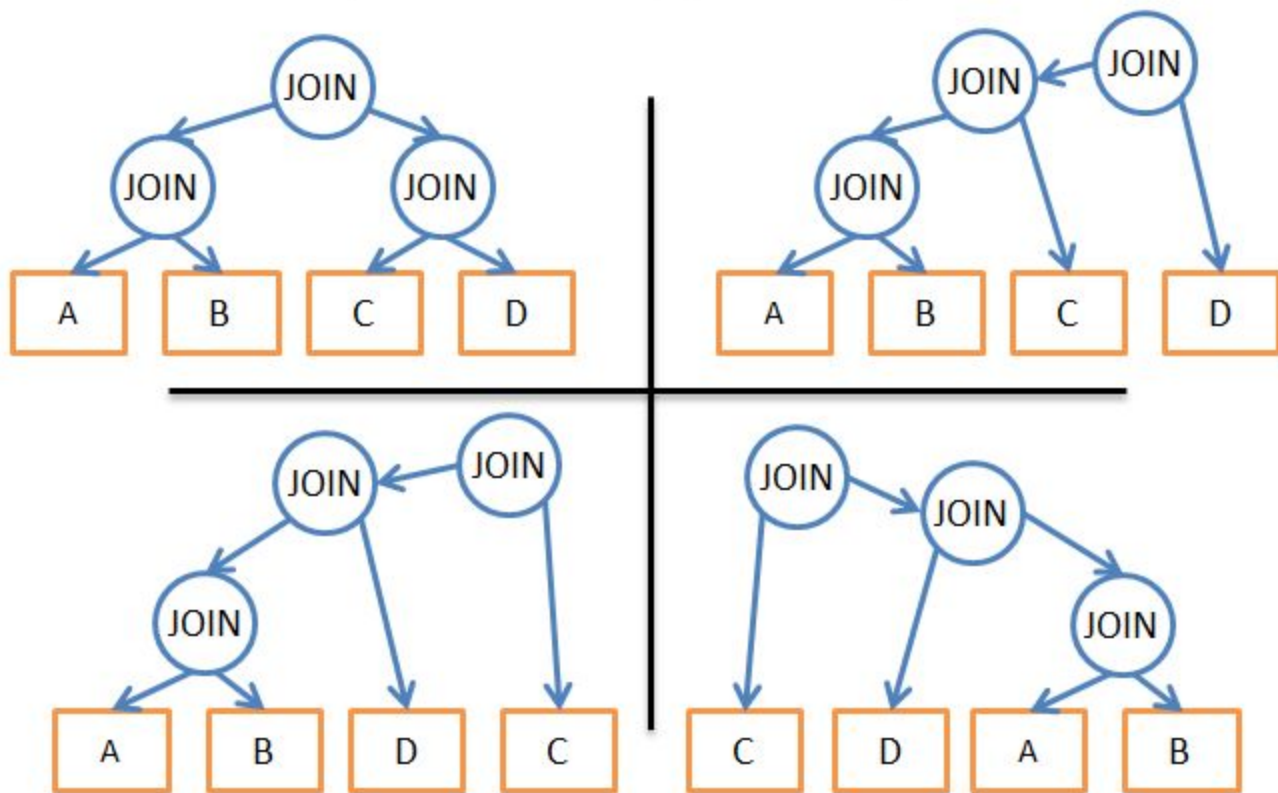
Relation a

3	12	26	43	60	75	80
---	----	----	----	----	----	----

Relation b



Join ordering problem



Statistics

Choosing the best execution plan





...

1 Query \approx 1 Index

Speed

VS

Memory & Update speed

And now ?

New Tables

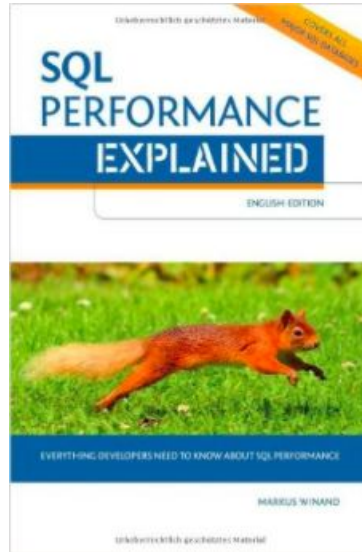
Slow Log

EXPLAIN



USE THE INDEX, LUKE

A guide to database performance for developers



<http://use-the-index-luke.com>



<http://coding-geek.com/how-databases-work/>



Thanks !