QUERY PLANNER BASICS

BEST THING ABOUT SQL?

Incredibly simple to query and filter arbitrary data

WORST THING ABOUT SQL?

Hidden complexity.

```
select *
from students
where name = 'Tomas';
```

```
f = fopen('database.dat')
while(row = read(f)) {
   data = parse(row);
   if(data['name'] == 'Tomas') {
      emit(data);
   }
}
```

QUERY PLANNER

- Database component
- In: SQL query
- Out: Execution plan
- Show query plan for any query: prefix it with "EXPLAIN" keyword

- **explain** select * from students where name = 'Tomas';

```
oz=# explain select * from documents;
                          QUERY PLAN
---------------------------------------------------------------
 Seq Scan on documents  (cost=0.00..16520.24 rows=351224
width=303)
(1 row)
```

```
oz=# explain select * from documents where supplier = 'SPP'
order by created_at desc limit 10;
```

```
 Limit  (cost=17398.42..17398.44 rows=8 width=303)
   ->  Sort  (cost=17398.42..17398.44 rows=8 width=303)
         Sort Key: created_at
         ->  Seq Scan on documents  (cost=0.00..17398.30
rows=8 width=303)
               Filter: ((supplier)::text = 'SPP'::text)
(5 rows)
```

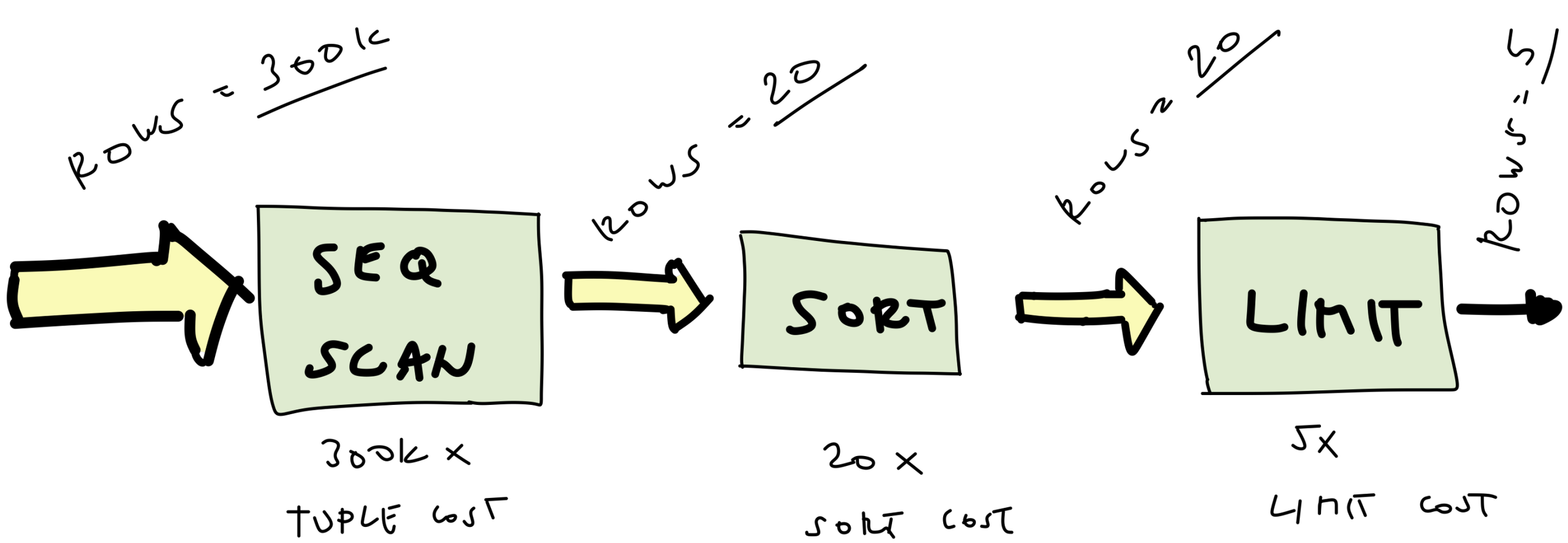
ROW ESTIMATION

```
select
tablename,attname,null_frac,avg_width,n_distinct,co
rrelation,most_common_vals,most_common_freqs,histog
ram_bounds
from pg_stats
where tablename='documents';
```

- https://www.postgresql.org/docs/11/row-estimation-examples.html

COST BASED OPTIMIZER

- Generate all (if feasible) possible plans
- Estimate data flow of each plan and compute a "cost"
- Select the cheapest plan = minimal cost



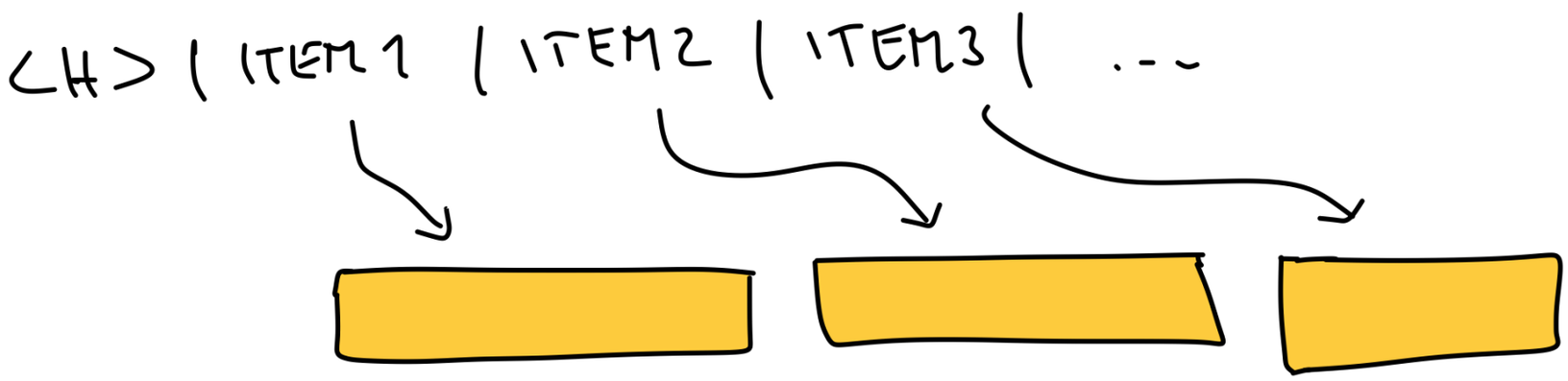
COSTS

- Accessing page sequentially - seq_page_cost
- Accessing page randomly - random_page_cost
- Processing index entry - cpu_index_tuple_cost
- Processing each row - cpu_tuple_cost
- ... https://www.postgresql.org/docs/11/runtime-config-query.html

DATA ORGANIZATION

- Tuple = row
- Block/page = 8kb of tuples
- Heap
- CTID = special (hidden) column, page number + index

```
oz=# select ctid, * from documents limit 100;
```



SEQUENTIAL VS. RANDOM ACCESS

SEQ. SCAN

RANDOM ACCESS

SSD DISKS ARE COOL BUT $$$

- SSD @ AWS = $0.125 per Gb / month
- HDD @ AWS = $0.045 per Gb / month

INDEX

- Data access method
- Makes queries fast! (not necessarily)
- Most common type: B+ tree (hash, inverted index, ...)
- Single column or multi-column (composite)

```
➤ explain analyze select * from documents
  where supplier = 'SPP';
➤ create index
  index_documents_on_supplier on
  documents(supplier);
➤ explain analyze select * from documents
  where supplier = 'SPP';
```

```
oz=# explain analyze select * from documents
where supplier = 'SPP';

--------------------------------------------

 Seq Scan on documents  (cost=0.00..17398.30
rows=8 width=303) (actual time=0.841..64.837
rows=32 loops=1)

   Filter: ((supplier)::text = 'SPP'::text)

   Rows Removed by Filter: 351192

 Total runtime: 64.873 ms

(4 rows)
```

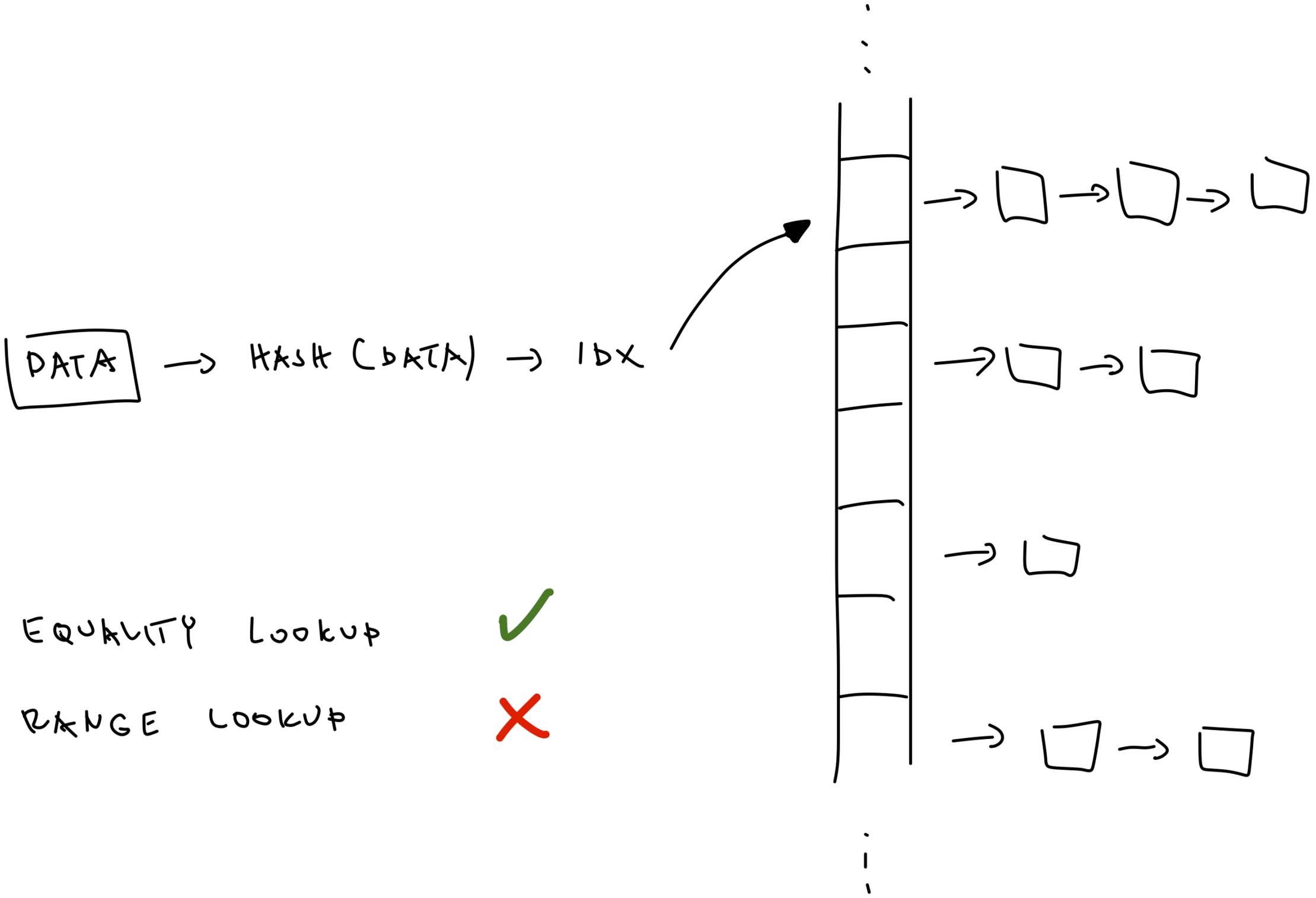
```
oz=# explain analyze select * from documents where
supplier = 'SPP';
```

```
------------------------------------------------------------
```

```
 Bitmap Heap Scan on documents  (cost=4.48..35.99
rows=8 width=303) (actual time=0.049..0.075 rows=32
loops=1)
   Recheck Cond: ((supplier)::text = 'SPP'::text)
   ->  Bitmap Index Scan on
index_documents_on_supplier  (cost=0.00..4.48 rows=8
width=0) (actual time=0.040..0.040 rows=32 loops=1)
         Index Cond: ((supplier)::text = 'SPP'::text)
 Total runtime: 0.108 ms
(5 rows)
```

600 x FKSTER

HASH INDEX

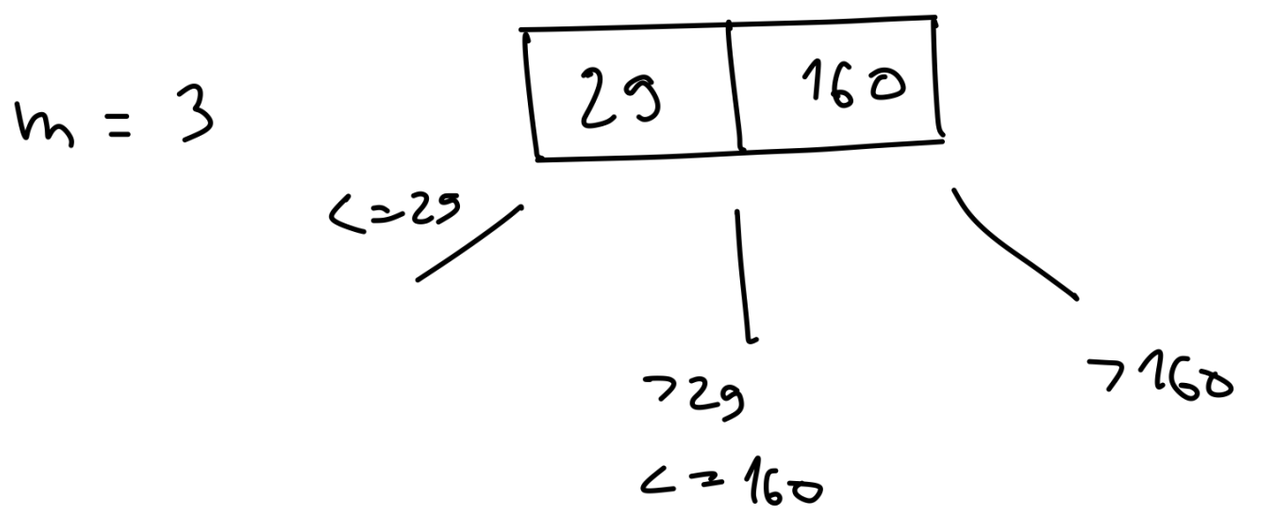


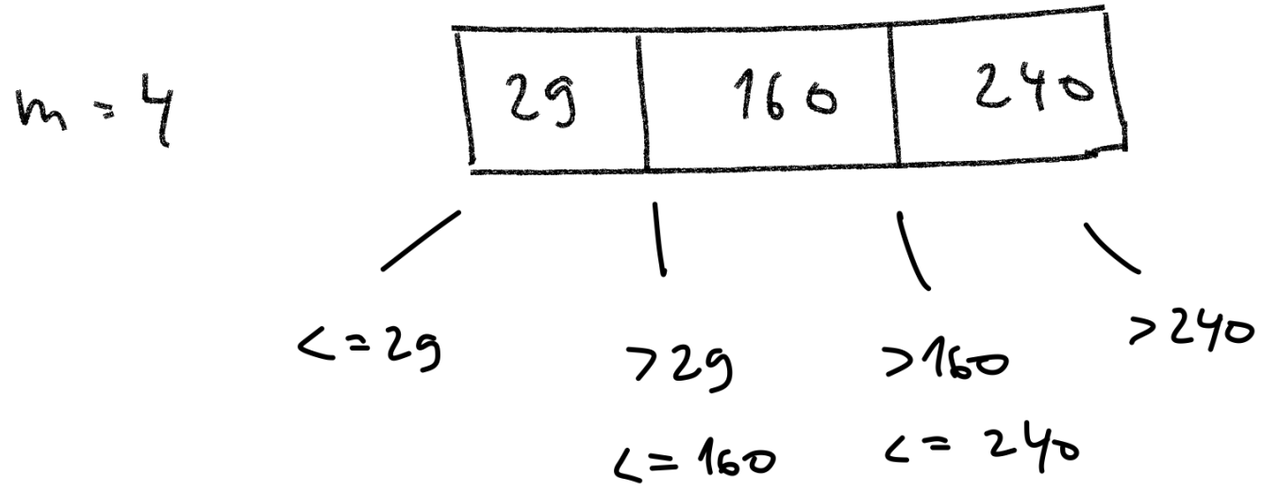
EQUALITY LOOKUP ✓

RANGE LOOKUP ✗

B-TREE INDEX

- B-tree degree
- Left-biased, right-biased
- Balanced
- "Low-depth"

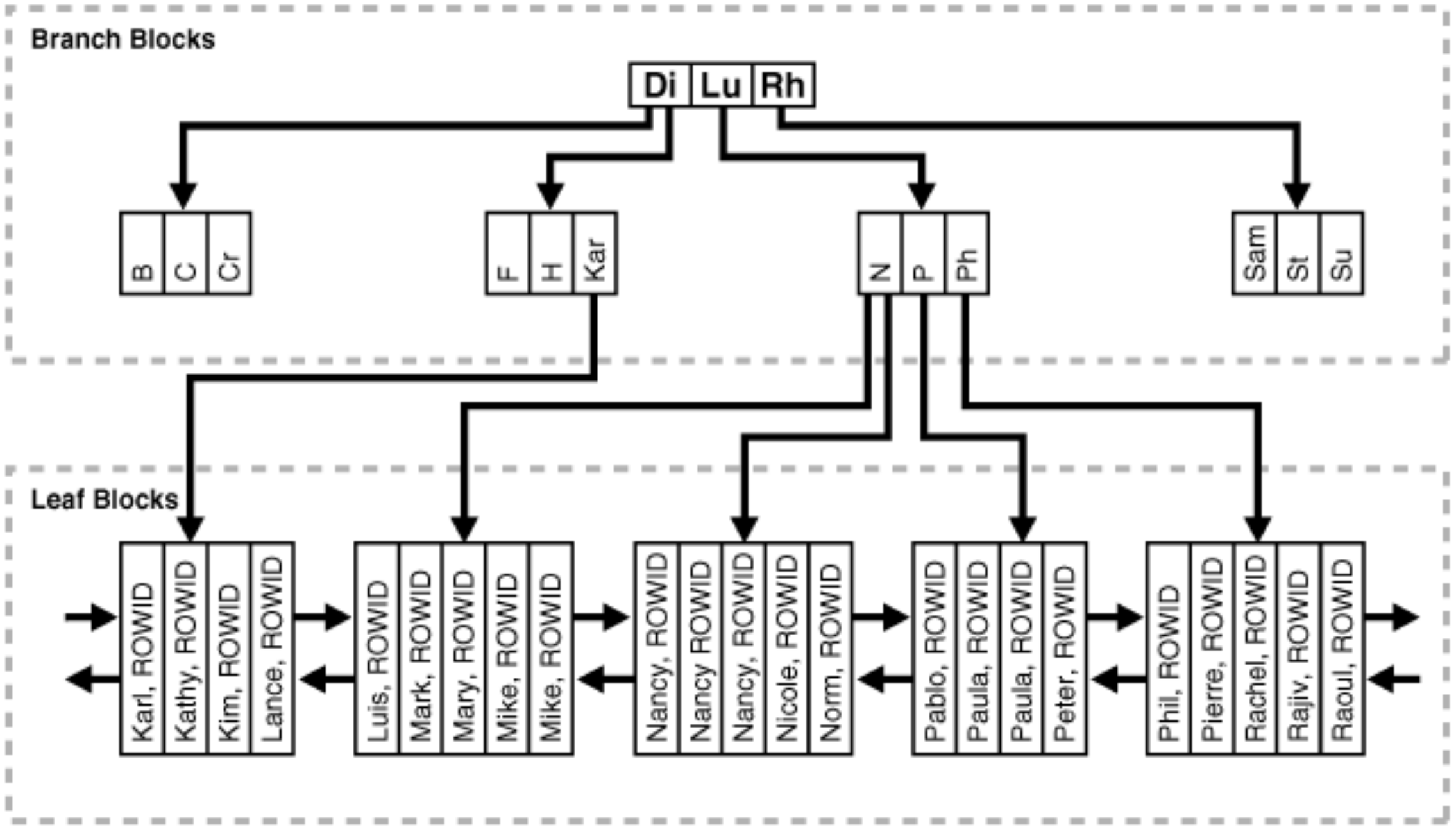




B-TREE FROM SCRATCH

https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html

B-TREE INDEX

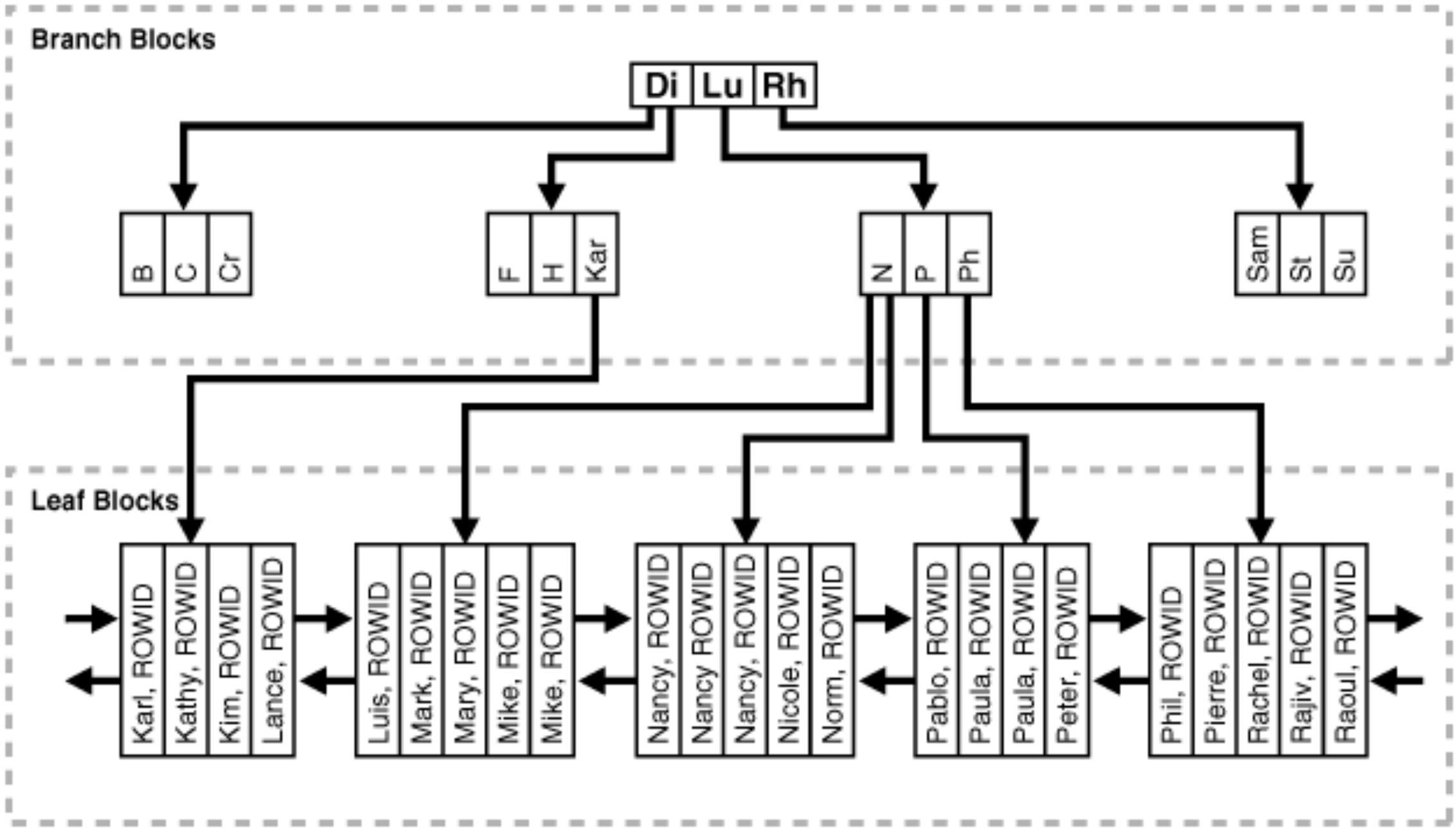


https://docs.oracle.com/cd/B19306_01/server.102/b14220/img/cncpt169.gif

B-TREE INDEX

EQUALITY Lookup ✔

RANGE Lookup ✔



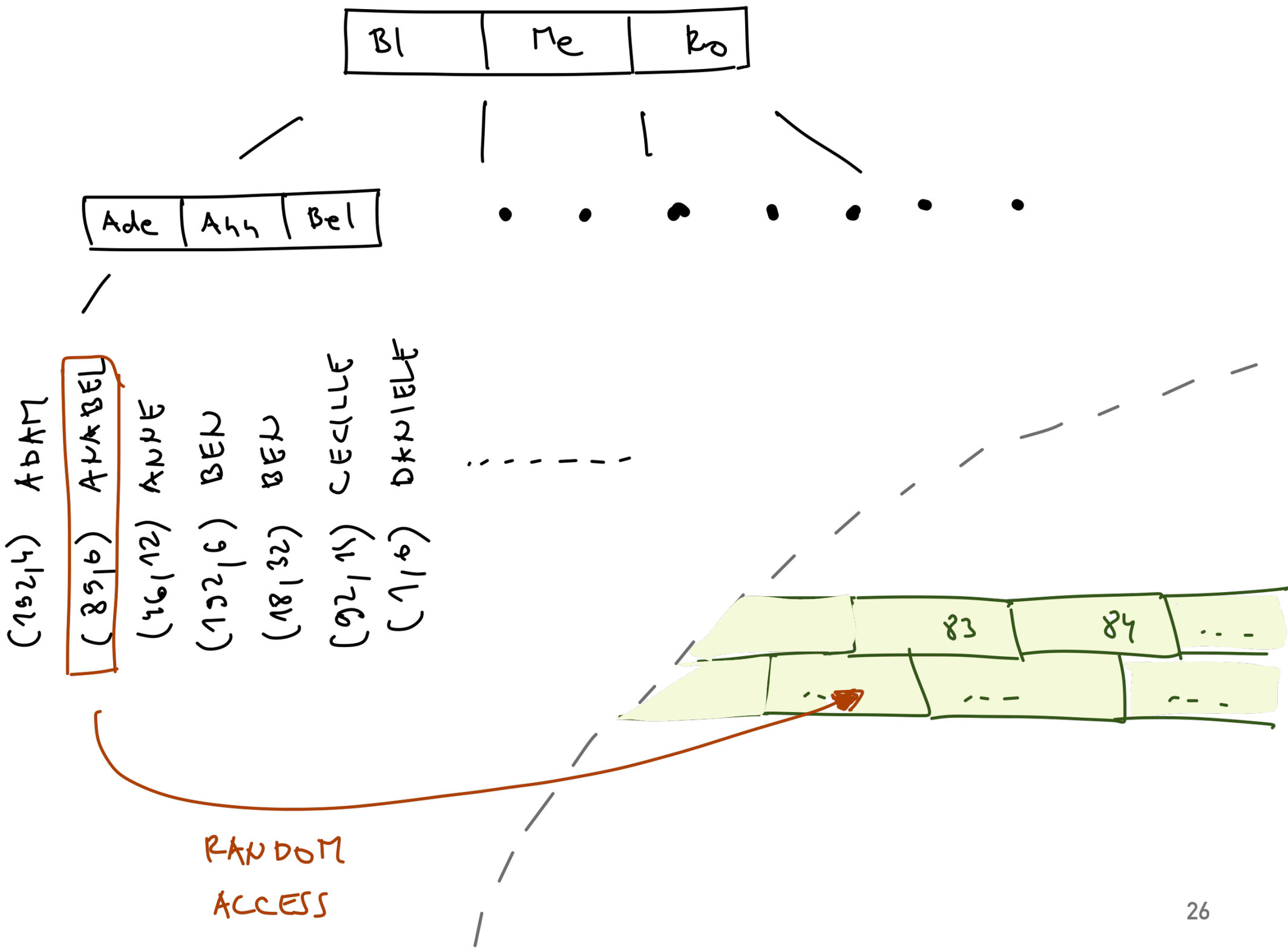
https://docs.oracle.com/cd/B19306_01/server.102/b14220/img/cncpt169.gif

3 QUERIES, 3 DIFFERENT PLANS..

```
create index index_documents_on_type on
documents(type);

select * from documents where type =
'Egovsk::Appendix';
```

```
Index Scan using index_documents_on_type on
documents  (cost=0.42..1409.38 rows=843
width=303)
   Index Cond: ((type)::text =
'Egovsk::Appendix'::text)
```

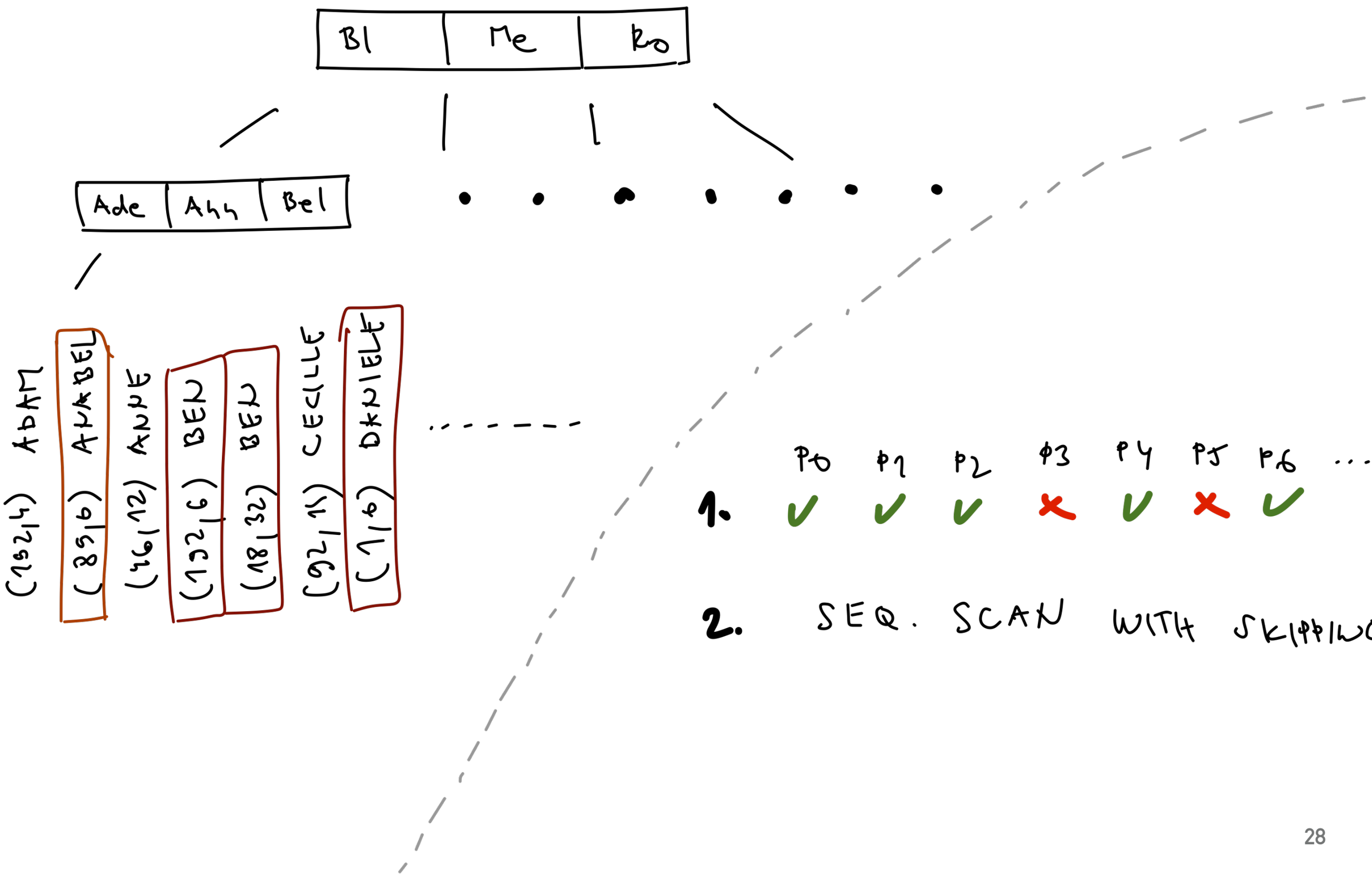
INDEX SCAN


Bl
Me
Ro
Ade
Ahh
Bel
(192,4) ADAM
(89,6) AMABEL
(46,12) ANNE
(132,6) BEN
(18,32) BEN
(92,11) CECILLE
(1,6) DANIELE
83
84
RANDOM
ACCESS

3 QUERIES, 3 DIFFERENT PLANS..

```
select * from documents where type =
'Crz::Appendix';

Bitmap Heap Scan on documents
(cost=400.53..14317.46 rows=17046 width=303)
   Recheck Cond: ((type)::text =
'Crz::Appendix'::text)
   ->  Bitmap Index Scan on
index_documents_on_type  (cost=0.00..396.27
rows=17046 width=0)
         Index Cond: ((type)::text =
'Crz::Appendix'::text)
```

BITMAP INDEX SCAN && BITMAP HEAP SCAN

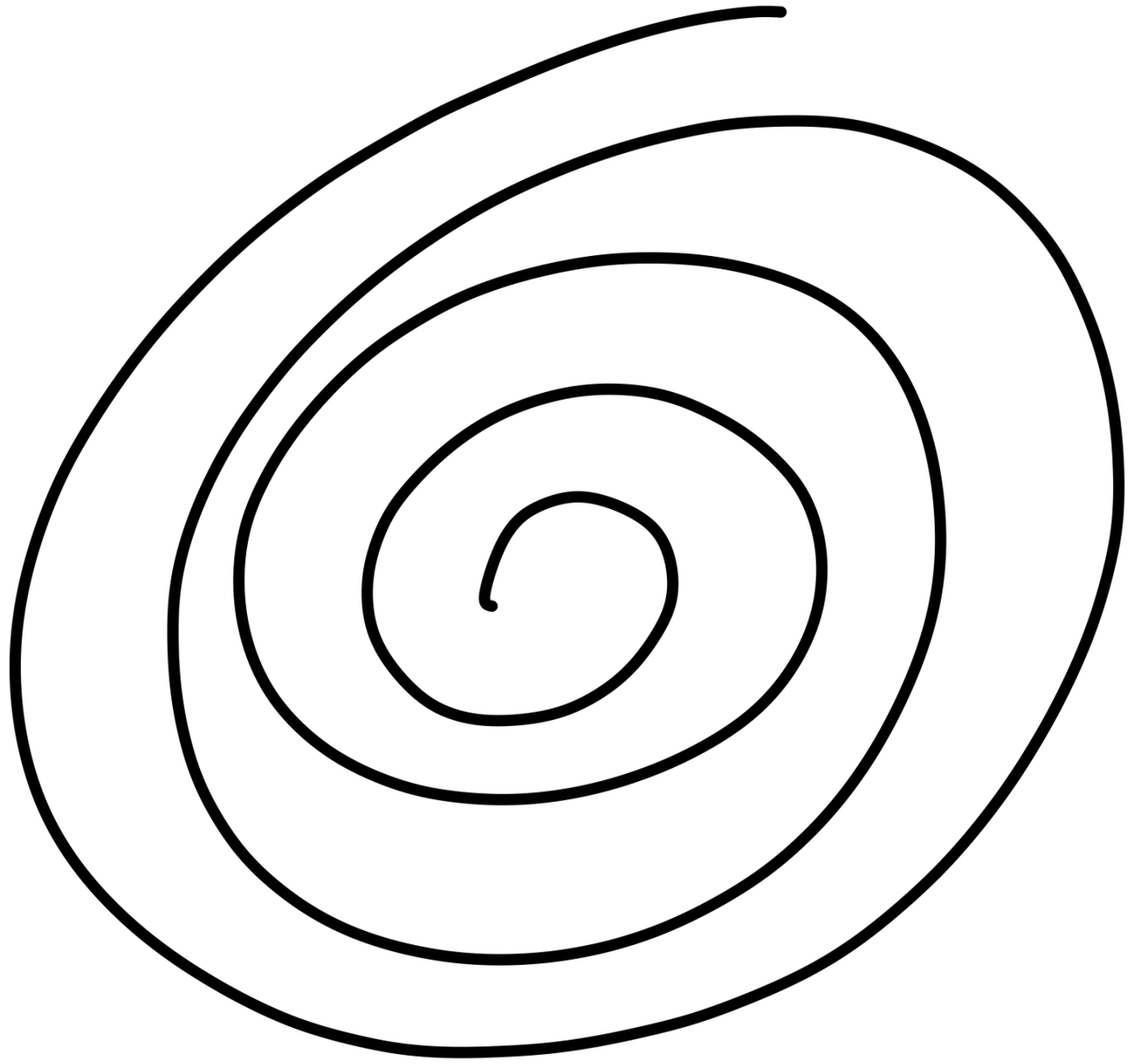


3 QUERIES, 3 DIFFERENT PLANS..

```
select * from documents where type =
'Crz::Contract';
```

```
Seq Scan on documents
(cost=0.00..17398.30 rows=322529
width=303)
   Filter: ((type)::text =
'Crz::Contract'::text)
```

SEQ SCAN?!



WHY 3 DIFFERENT PLANS?

```
select type, count(*)
from documents
group by type;
```

```
        type          | count
----------------------+-------
 Egovsk::Appendix     |   870
 Crz::Appendix        | 16921
 Egovsk::Contract     | 11262
 Crz::Contract        | 322171
```

ACCESS METHODS COMPARISON

- Index Scan
  - Involves random heap access
  - Suitable when expected output is "small"
- Bitmap index scan
  - Involves targeted sequential heap access
  - Suitable when expected output is "larger"
- Sequential scan
  - Inefficient sequential scan over heap
  - Used for non-selective indices

INDEX SELECTIVITY

- Index selectivity = how much can a filter on that particular column value narrow down the output
- Selective index = narrows the search significantly
  - E.g.: personal ID, birth certificate number, email address
- Non-selective index = large part of the data fits the condition
  - E.g.: gender

CBO RECAP

1. Generate all (*) query plans
2. Estimate data flow through each query plan
3. Compute plan cost
4. Select plan with minimal cost

RULE BASE OPTIMIZER

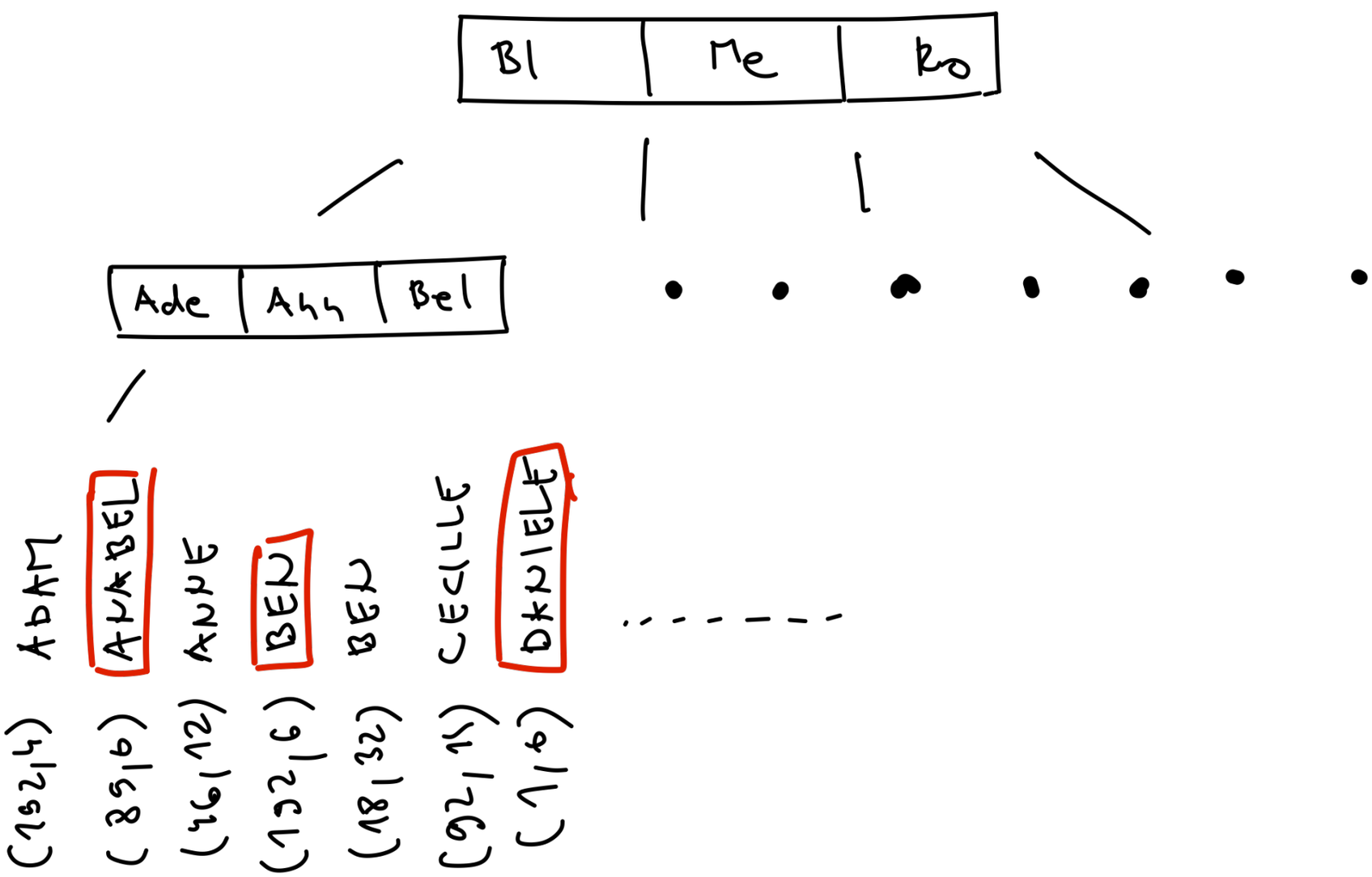
- Doesn't use estimates, just simple rules
- E.g.: Can an index be used? -> Use it.

INDEX ONLY SCAN

```
select type from documents where type =
'Egovsk::Appendix';


Index Only Scan using
index_documents_on_type on documents
(cost=0.42..1409.38 rows=843 width=14)
   Index Cond: (type =
'Egovsk::Appendix'::text)
```

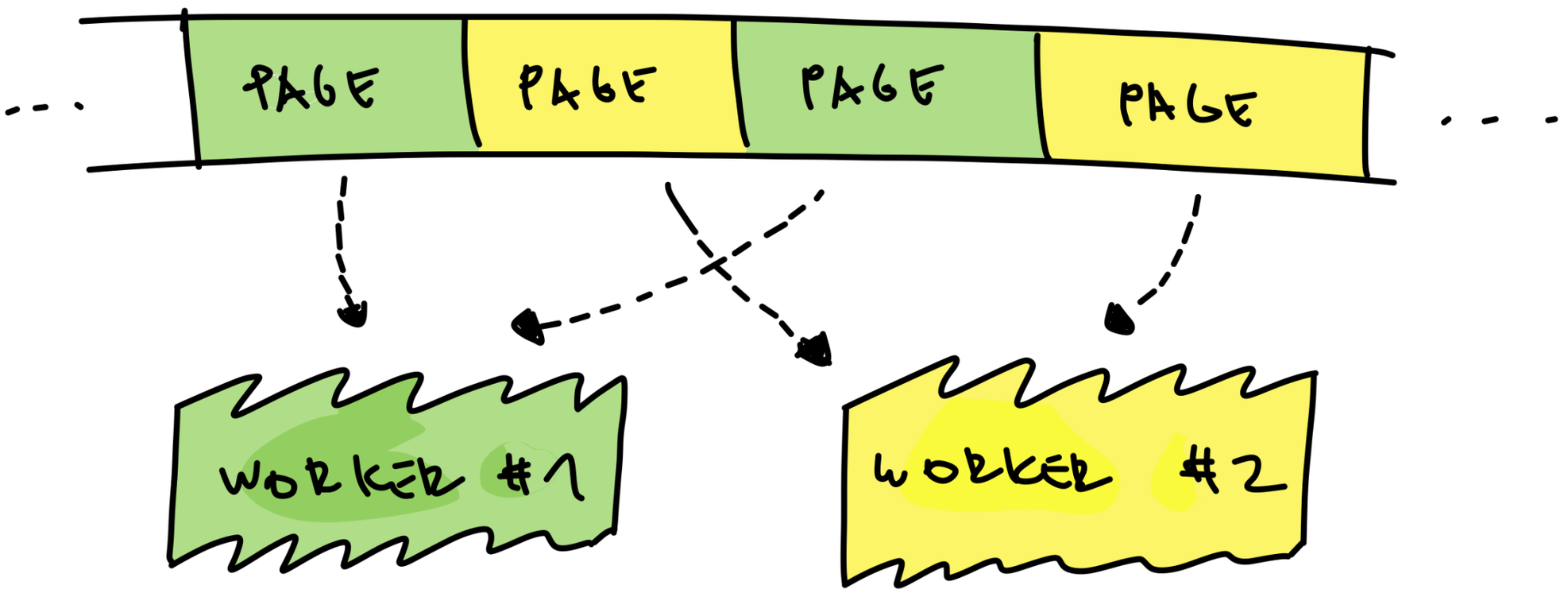
* not useful for single-column indices obviously, but very useful for composite indices

INDEX ONLY SCAN



PARALLEL SEQUENTIAL SCAN

- Since PostgreSQL 9.6
- Maintains sequential scan, but blocks are processed in parallel
- Requires a where clause



```
CREATE TABLE test (i int);
INSERT INTO test SELECT generate_series(1,100000000);

EXPLAIN ANALYZE SELECT * FROM test WHERE i=1;


Gather  (cost=1000.00..964309.43 rows=1 width=4) (actual
time=0.435..6586.104 rows=1 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Seq Scan on test  (cost=0.00..963309.33
rows=1 width=4) (actual time=4383.339..6576.226 rows=0
loops=3)
         Filter: (i = 1)
         Rows Removed by Filter: 33333333
 Planning Time: 0.219 ms
 Execution Time: 6586.138 ms
```

OTHER PARALLEL OPERATIONS

- Parallel bitmap heap scan
- Parallel index scan
- Parallel index-only scan
- Parallel joins
- Parallel aggregations

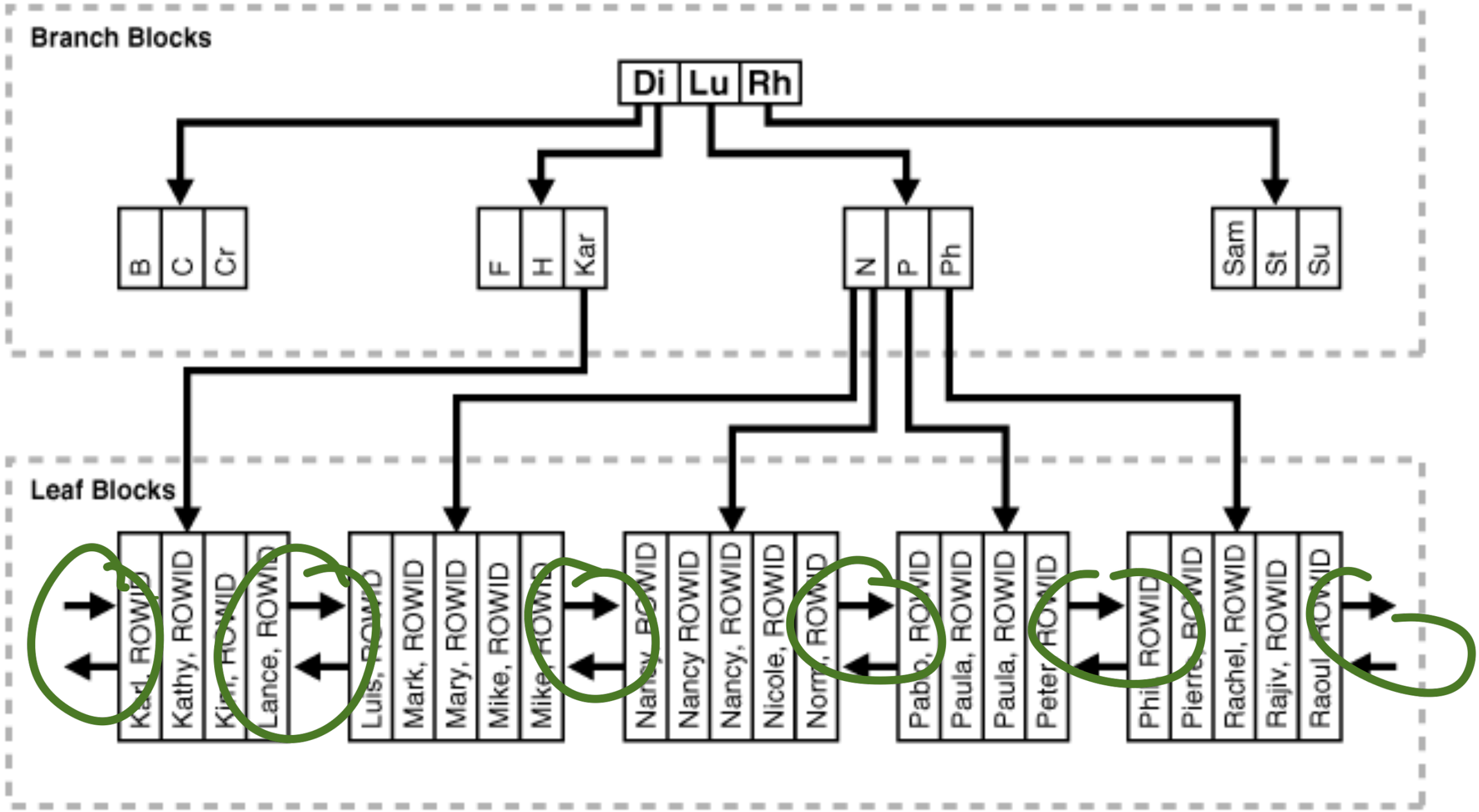
USING INDEX FOR "ORDER BY"

```
select * from documents order by published_on asc;

select * from documents order by published_on desc;

create index index_documents_on_published_on on documents(published_on);
```

```
Index Scan using index_documents_on_published_on on documents
(cost=0.42..61164.78 rows=351224 width=312)
```

```
Index Scan Backward using index_documents_on_published_on on documents
(cost=0.42..61164.78 rows=351224 width=312)
```

USING INDEX FOR "ORDER BY"



PIPELINED ORDER BY

```
select * from documents order by published_on limit 10;

Limit  (cost=24110.06..24110.09 rows=10 width=304)
   ->  Sort  (cost=24110.06..24988.12 rows=351224
width=304)
         Sort Key: published_on
         ->  Seq Scan on documents  (cost=0.00..16520.24
rows=351224 width=304)
```

```
create index index_documents_on_published_on on
documents(published_on);

select * from documents order by published_on limit 10;

Limit  (cost=0.42..1.33 rows=10 width=304)
   ->  Index Scan using index_documents_on_published_on on
documents  (cost=0.42..31855.74 rows=351224 width=304)
```

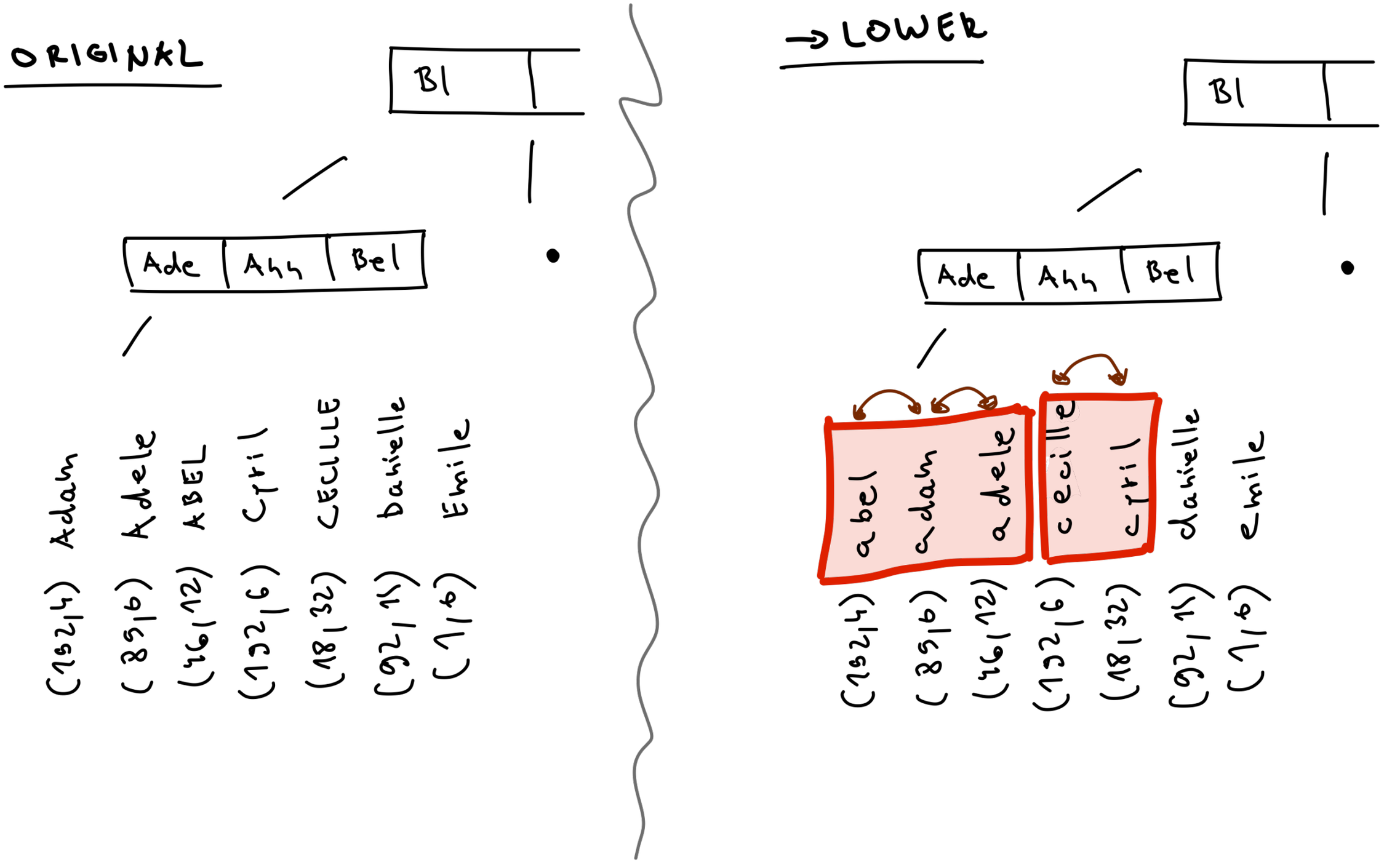
INDICES & MODIFICATIONS

```
create index index_documents_on_supplier on
documents(supplier);
```

```
select * from documents where lower(supplier) = 'spp';
```

```
Seq Scan on documents  (cost=0.00..18276.36 rows=1756
width=304)
   Filter: (lower((supplier)::text) = 'spp'::text)
```

INDICES & MODIFICATIONS



INDICES & MODIFICATIONS

```
create index index_documents_on_lower_supplier on
documents(lower(supplier));

select * from documents where lower(supplier) = 'spp';


Bitmap Heap Scan on documents  (cost=4.48..35.99 rows=8
width=304)
   Recheck Cond: ((supplier)::text = 'spp'::text)
   ->  Bitmap Index Scan on index_documents_on_supplier
(cost=0.00..4.48 rows=8 width=0)
         Index Cond: ((supplier)::text = 'spp'::text)
```

INDEX PROS & CONS

- Pros:
  - Makes filtering faster (sometimes)
  - Makes ordering faster
- Cons:
  - Expensive to build
  - Expensive to maintain = slows down writes (insert, update)

PARTIAL INDICES

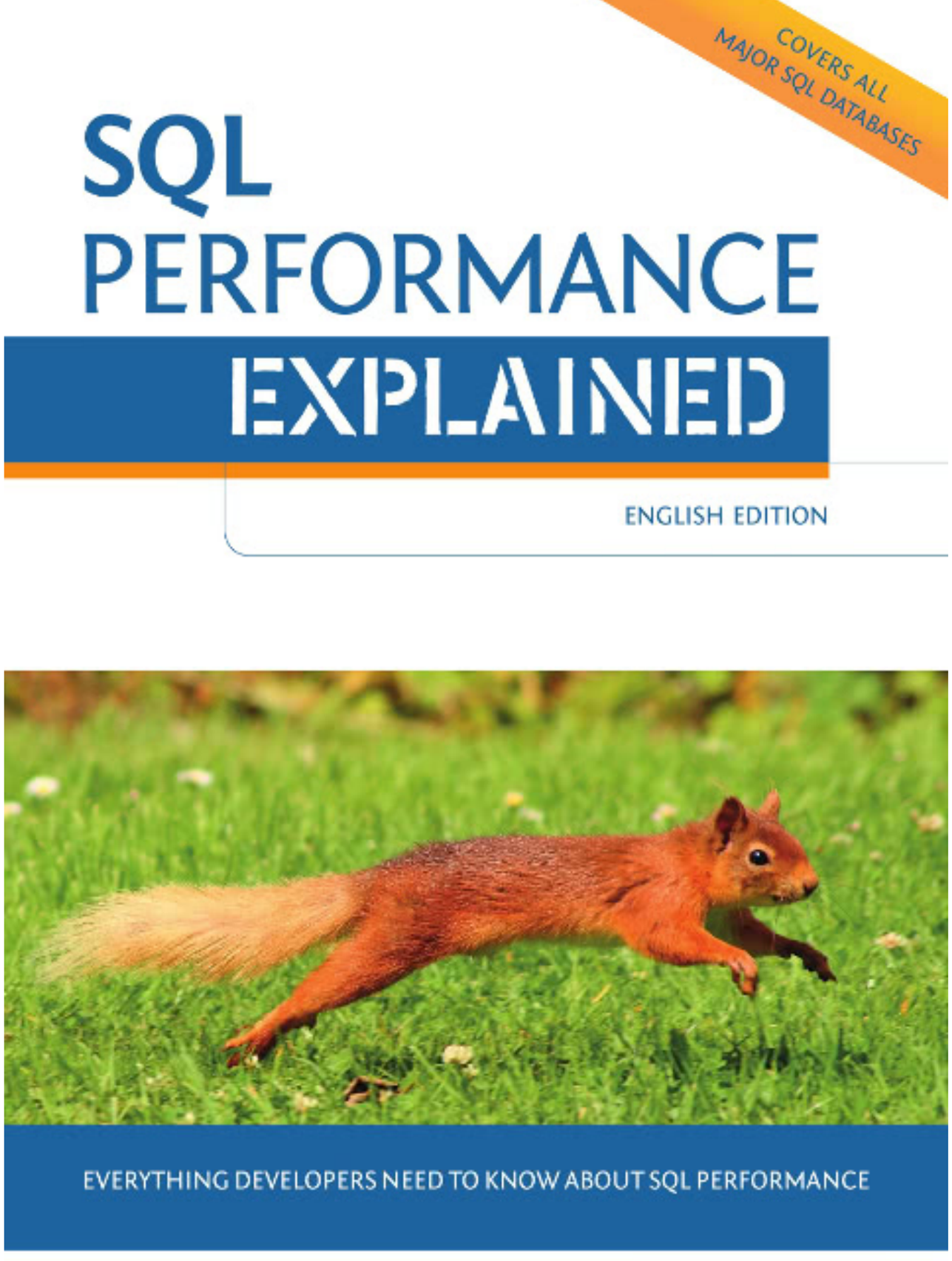
```
create index index_documents_on_type on documents(type)
where type = 'Egovsk::Appendix';


explain select * from documents where type =
'Egovsk::Appendix';

 Index Scan using index_documents_on_type on documents
(cost=0.28..1452.10 rows=866 width=304)

   Index Cond: ((type)::text = 'Egovsk::Appendix'::text)


explain select * from documents where type =
'Crz::Appendix';

 Seq Scan on documents  (cost=0.00..17398.30 rows=17034
width=304)

   Filter: ((type)::text = 'Crz::Appendix'::text)
```

CONDITIONS ARE SMART

```
create index index_documents_on_type on
documents(type) where type != 'Crz::Contract';


explain select * from documents where type =
'Crz::Appendix';
 Bitmap Heap Scan on documents
 (cost=432.30..14349.10 rows=17034 width=304)
    ->  Bitmap Index Scan ...
```

SUMMARY (WHAT YOU SHOULD KNOW)

- Understand b-trees, draw a simple b-tree of given degree from sample input data
- Understand CBO and the data access methods (seq. scan, index scan, bitmap index scan...), draw a query plan for a simple query.
- Pros & cons of indexing, index selectivity



RECOMMENDED READING

- Book ($$$)
- Use the Index, Luke!

https://use-the-index-luke.com