Motivation

- Problem:
  - select x from Y
    where z = 'k';
- Plan:

select z = 'k'

↑

scan Y

- Cost?

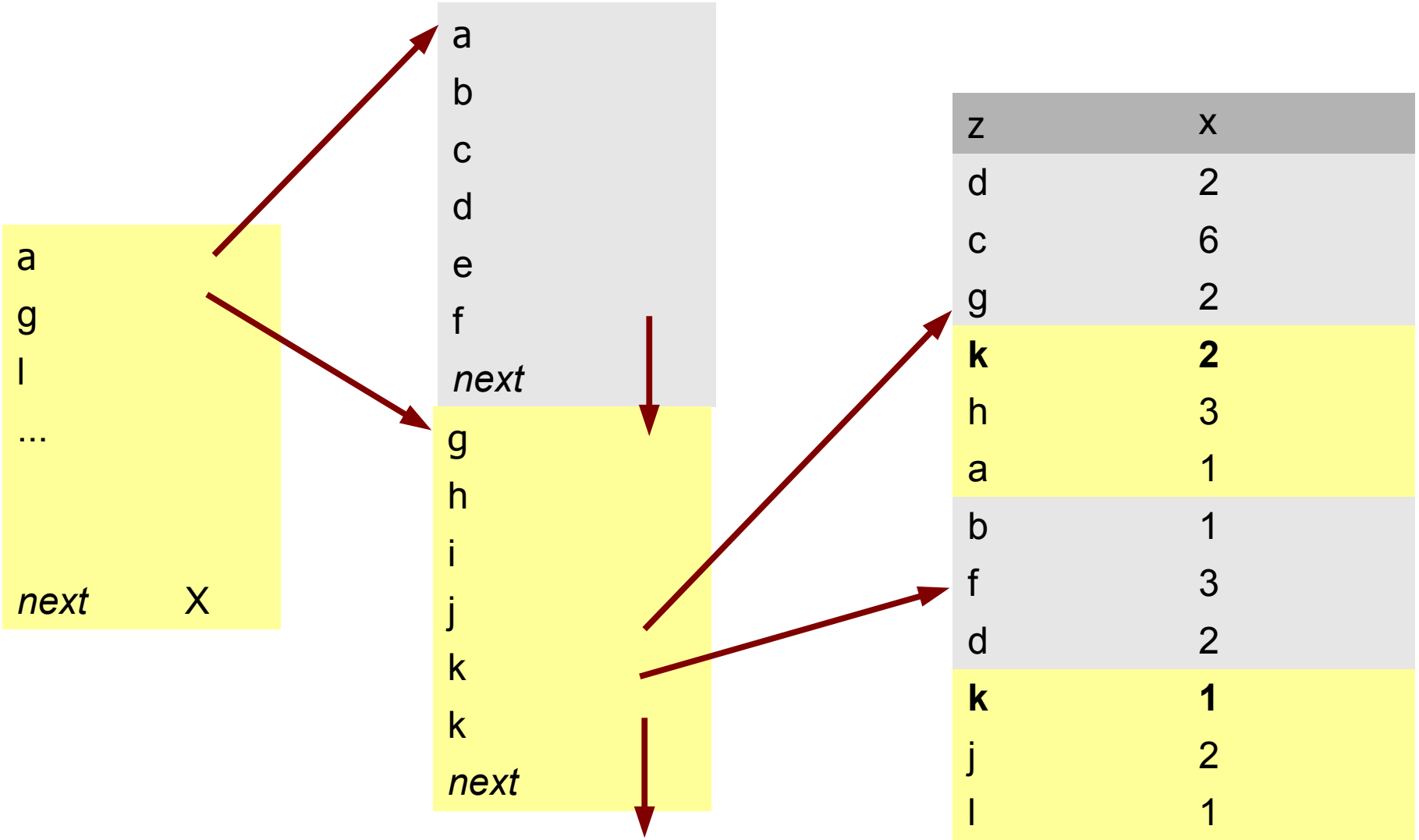
z	x
d	2
c	6
g	2
k	**2**
h	3
a	1
b	1
f	3
d	2
k	**1**
j	2
l	1
...	...

Index

- Makes it easy to find pages containing interesting data
- Smaller that data
  - Fits in memory?
- Efficient look-up:
  - Identity (=)
  - Ranges
  - LIKE
  - ...

z	x
d	2
c	6
g	2
k	**2**
h	3
a	1
b	1
f	3
d	2
k	**1**
j	2
l	1
...	...

B-Tree


a
g
l
...
next X
a
b
c
d
e
f
next
g
h
i
j
k
k
next
z x
d 2
c 6
g 2
k 2
h 3
a 1
b 1
f 3
d 2
k 1
j 2
l 1
... ...

B-Tree

- Insert:
  - If free entry not available, split leaf
    - Recursively insert new leaf in upper layer
    - Tree grows towards the root!
  - Add entry to leaf
- Delete:
  - Remove entry from leaf
  - If enough space available, collapse leafs
    - Recursively delete leaf in upper layer

B-Tree

- Desirable characteristics:
  - Balanced
  - Log(n) depth
  - Fit for block I/O
- Supports:
  - Identity look up
  - Range queries / Ordered scan
  - Updates

Clustered indexes

- Problem:
  - An index scan accesses each block multiple times
- A clustered index:
  - Records are (roughly) sorted according to the index
    - No sorting within a block is needed
  - Free space may be kept for insertions

More indexes

- Hash indexes
- Inverted indexes
- Bitmap indexes
- …

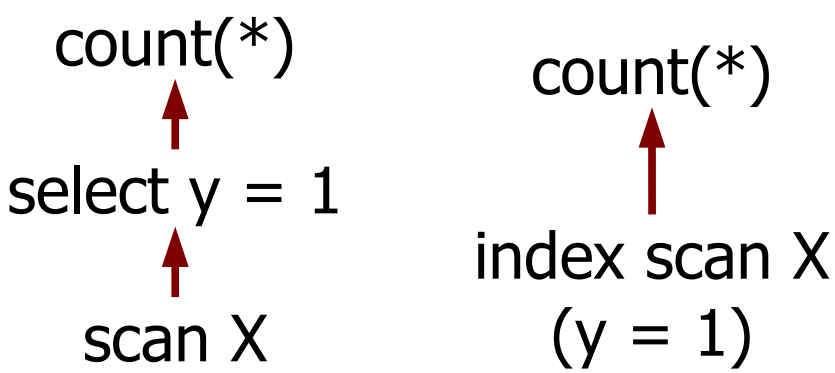
Motivation

- Assumptions:
  - Several TB of data
  - ~50%, y=1
  - ~50%, y=2
  - a few, y=3

z	y
d	**1**
c	2
g	**1**
k	2
h	3
a	**1**
b	**1**
f	2
d	2
k	**1**
j	2
l	**1**
...	...

Motivation

- Problem:
  - select count(*) from X
    where y = 1;
- Possible plans:



- Cost?

z	y
d	**1**
c	2
g	**1**
k	2
h	3
a	**1**
b	**1**
f	2
d	2
k	**1**
j	2
l	**1**
...	...

Motivation

- Keep results cached
- Update when needed

select y, count(*) from X group by y

y	count
1	**773647263**
2	765732332
3	1

X

z	y
d	**1**
c	2
g	**1**
k	2
h	3
a	**1**
b	**1**
f	2
d	2
k	**1**
j	2
l	**1**
...	...

Views

select count(*) from V

View usage

count(*)
↑
V

View definition

select y = 1
↑
scan X

count(*)
↑
select y = 1
↑
scan X

Plan executed

create view V as select * from X where y=1

Materialized views

select count(*) from V

View usage

count(*)
↑
V

Plan executed

View definition

materialize V
↑
select y = 1
↑
scan X

create materialized view V as select * from X where y=1

Maintaining materialized views

- Periodically run the query and update the view
- Update the view when data changes

DIY Materialized Views

- Updating with AFTER triggers:

```
-- Create view
select sum(value) into mv_sum_items from items;

-- Update view
create function upd_sum_items() returns trigger as '
    BEGIN
        update mv_sum_items set sum = sum + new.value - old.value;
        return new;
    END
' language 'plpgsql';

create trigger upd_sum
    after update on items
    for each row execute procedure upd_sum_items();
```

Using materialized views

- Automatically used by the planner:
  - Indexed views in MS SQL Server
- Used explicitly in queries:
  - Materialized views in Oracle
  - DIY materialized views everywhere
  - Developer tip:
    - Using views allows the DBA to select which ones to materialize

Conclusions

- Indexes and mat. views = Redundancy!
- Trade-off between:
  - Complexity of operations

  and:
  - Disk space used
  - Usage of main memory
  - Effort when updating
- Usefulness depends on workload mix