CSE544: Principles of Database Systems

Lectures 3

Storage and Indexes

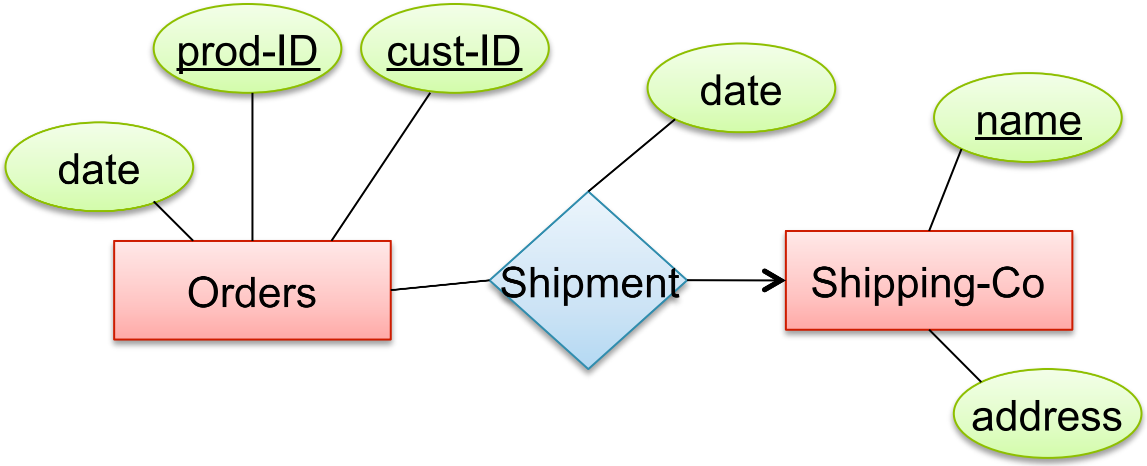
Announcement

- Homework 1 is due tomorrow, Tuesday, 11pm
- Max 2 late days per homework, max 4 per quarter
- Homework 2 will be posted tomorrow, due on Friday, Oct. 30, 11pm.

Review of Lecture 2

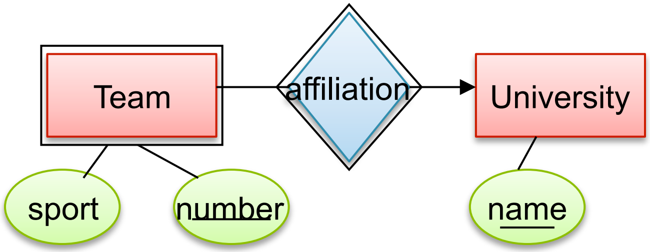
- What is a many-to-many relationship? What is a many-to-one relationship?

- What does this mean?

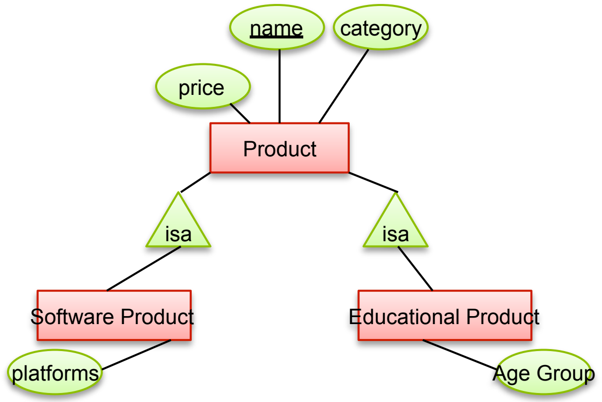


Review of Lecture 2

- What is a weak entity set?



- How do we represent IsA relationships in tables?



Outline

- Functional Dependencies and BCNF
  - Continued from previous lecture
  - Note: will skip some slides, please review at home

- Storage and Indexes
  - Book: Ch. 8-11, and 20
  - PAX paper

Review of Lecture 2

- What are *data anomalies?*

- What is a *functional dependency*?

Functional Dependencies (FDs)

Definition

If two tuples agree on the attributes

$A_1, A_2, \ldots, A_n$

then they must also agree on the attributes

$B_1, B_2, \ldots, B_m$

Formally:

$A_1 \ldots A_n$ **determines** $B_1 .. B_m$

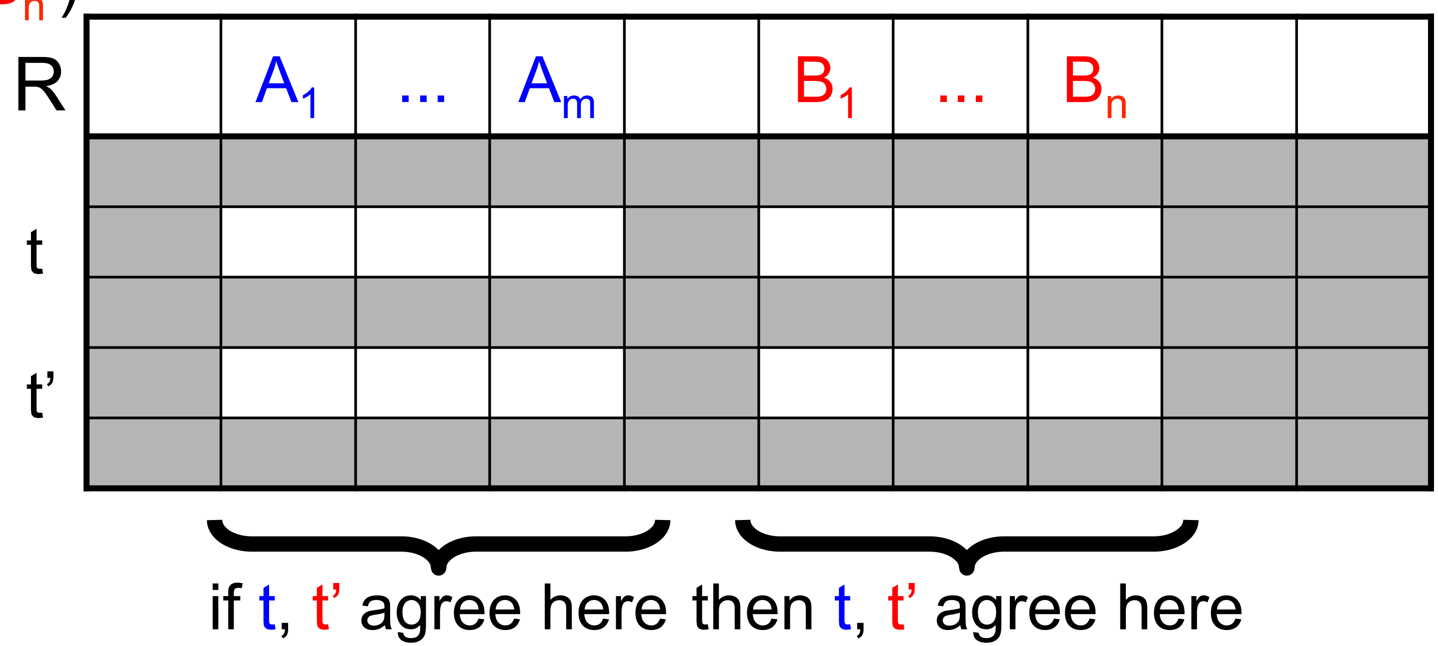
$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$

Functional Dependencies (FDs)

Definition $A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ **holds** in R if:

$\forall t, t' \in R,$

$(t.A_1 = t'.A_1 \wedge \ldots \wedge t.A_m = t'.A_m \Rightarrow t.B_1 = t'.B_1 \wedge \ldots \wedge t.B_n = t'.B_n)$



Example

An FD holds, or does not hold on an instance:

EmpID	**Name**	**Phone**	**Position**
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

EmpID → Name, Phone, Position

Position → Phone

but not Phone → Position

Example

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876 ←	Salesrep
E1111	Smith	9876 ←	Salesrep
E9999	Mary	1234	Lawyer

Position → Phone

Example

EmpID	Name	Phone	Position
E0045	Smith	1234 →	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234 →	Lawyer

But not Phone → Position

Example

name → color
category → department
color, category → price

name	category	color	department	price
Gizmo	Gadget	Green	Toys	49
Tweaker	Gadget	Green	Toys	99

Do all the FDs hold on this instance?

Example

name → color
category → department
color, category → price

name	category	color	department	price
Gizmo	Gadget	Green	Toys	49
Tweaker	Gadget	Black	Toys	99
Gizmo	Stationary	Green	Office-supp.	59

What about this one ?

Terminology

FD **holds** or **does not hold** on an instance

If we can be sure that *every instance of R* will be one in which a given FD is true, then we say that **R satisfies the FD**

If we say that R satisfies an FD F, we are **stating a constraint on R**

An Interesting Observation

If all these FDs are true:

name → color
category → department
color, category → price

Then this FD also holds:

name, category → price

If we find out from application domain that a relation satisfies some FDs, it doesn't mean that we found all the FDs that it satisfies!
There could be more FDs implied by the ones we have.

Closure of a set of Attributes

Given a set of attributes $A_1, \ldots, A_n$

The **closure**, $\{A_1, \ldots, A_n\}^+$ = the set of attributes B s.t. $A_1, \ldots, A_n \rightarrow B$

Example:

1. name → color
2. category → department
3. color, category → price

Closures:

name$^+$ = {name, color}

{name, category}$^+$ = {name, category, color, department, price}

color$^+$ = {color}

Closure Algorithm

X={A1, …, An}.

Repeat until X doesn't change **do**:
  if $B_1, \ldots, B_n \rightarrow C$ is a FD **and**
    $B_1, \ldots, B_n$ are all in X
  then add C to X.

Example:

1. name → color
2. category → department
3. color, category → price

$\{name, category\}^+ =$
$\{ \quad \}$

Closure Algorithm

X={A1, …, An}.

Repeat until X doesn't change **do**:
  if $B_1, \ldots, B_n \rightarrow C$ is a FD **and**
  $B_1, \ldots, B_n$ are all in X
  then add C to X.

Example:

1. name → color
2. category → department
3. color, category → price

$\{$name, category$\}^+$ =
  { name, category, color, department, price }

Closure Algorithm

X={A1, …, An}.

Repeat until X doesn't change **do**:
if $B_1, \ldots, B_n \rightarrow C$ is a FD **and**
$B_1, \ldots, B_n$ are all in X
then add C to X.

Example:

1. name → color
2. category → department
3. color, category → price

$\{\text{name, category}\}^+ =$
{ name, category, color, department, price }

Hence: name, category → color, department, price

Example

In class:

R(A,B,C,D,E,F)

A, B	$\rightarrow$	C
A, D	$\rightarrow$	E
B	$\rightarrow$	D
A, F	$\rightarrow$	B

Compute $\{A,B\}^+$ X = {A, B, }

Compute $\{A, F\}^+$ X = {A, F, }

Example

In class:

R(A,B,C,D,E,F)

$A, B \rightarrow C$
$A, D \rightarrow E$
$B \rightarrow D$
$A, F \rightarrow B$

Compute $\{A,B\}^+$ $X = \{A, B, C, D, E\}$

Compute $\{A, F\}^+$ $X = \{A, F, \qquad\qquad\qquad \}$

Example

In class:

R(A,B,C,D,E,F)

$A, B \rightarrow C$
$A, D \rightarrow E$
$B \rightarrow D$
$A, F \rightarrow B$

Compute $\{A,B\}^+$ X = {A, B, C, D, E }

Compute $\{A, F\}^+$ X = {A, F, B, C, D, E }

Example

In class:

R(A,B,C,D,E,F)

$A, B \rightarrow C$
$A, D \rightarrow E$
$B \rightarrow D$
$A, F \rightarrow B$

Compute $\{A,B\}^+$ $X = \{A, B, C, D, E\}$

Compute $\{A, F\}^+$ $X = \{A, F, B, C, D, E\}$

What is the key of R?

Practice at Home

Find all FD's implied by:

$A, B \rightarrow C$
$A, D \rightarrow B$
$B \rightarrow D$

Practice at Home

Find all FD's implied by:

$A, B \rightarrow C$
$A, D \rightarrow B$
$B \rightarrow D$

Step 1: Compute X^+, for every X:

A+ = A, B+ = BD, C+ = C, D+ = D

AB+ =ABCD, AC+=AC, AD+=ABCD,
BC+=BCD, BD+=BD, CD+=CD

ABC+ = ABD+ = ACD^+ = ABCD (no need to compute– why ?)

BCD^+ = BCD, ABCD+ = ABCD

Practice at Home

Find all FD's implied by:

$A, B \rightarrow C$
$A, D \rightarrow B$
$B \rightarrow D$

Step 1: Compute X^+, for every X:

$A+ = A$, $B+ = BD$, $C+ = C$, $D+ = D$

$AB+ = ABCD$, $AC+ = AC$, $AD+ = ABCD$,
$BC+ = BCD$, $BD+ = BD$, $CD+ = CD$

$ABC+ = ABD+ = ACD^+ = ABCD$ (no need to compute– why ?)

$BCD^+ = BCD$, $ABCD+ = ABCD$

Step 2: Enumerate all FD's $X \rightarrow Y$, s.t. $Y \subseteq X^+$ and $X \cap Y = \emptyset$:

$AB \rightarrow CD$, $AD \rightarrow BC$, $ABC \rightarrow D$, $ABD \rightarrow C$, $ACD \rightarrow B$

Keys

- A **superkey** is a set of attributes $A_1, ..., A_n$ s.t. for any other attribute B, we have $A_1, ..., A_n \rightarrow B$

- A **key** is a minimal superkey
  – A superkey and for which no subset is a superkey

Computing (Super)Keys

- For all sets X, compute X^+
- If X^+ = [all attributes], then X is a superkey
- Try only the minimal X's to get the keys

Example

Product(name, price, category, color)

name, category → price
category → color

What is the key ?

Example

Product(name, price, category, color)

name, category → price
category → color

What is the key ?

(name, category) + = { name, category, price, color }

Hence (name, category) is a key

Key or Keys ?

Can we have more than one key ?

Given R(A,B,C) define FD's s.t. there are two or more keys

Key or Keys ?

Can we have more than one key ?

Given R(A,B,C) define FD's s.t. there are two or more keys

$A \rightarrow B$
$B \rightarrow C$
$C \rightarrow A$

or

$AB \rightarrow C$
$BC \rightarrow A$

or

$A \rightarrow BC$
$B \rightarrow AC$

what are the keys here ?

Eliminating Anomalies

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield
Joe	987-65-4321	908-555-1234	Westfield

SSN → Name, City

What is the key?

Suggest a rule for decomposing the table to eliminate anomalies

Eliminating Anomalies

Main idea:

- X → A is OK if X is a (super)key

- X → A is not OK otherwise
  - Need to decompose the table, but how?

Boyce-Codd Normal Form

There are no "bad" FDs:

Definition. A relation R is in BCNF if:

Whenever $X \rightarrow B$ is a non-trivial dependency, then X is a superkey.

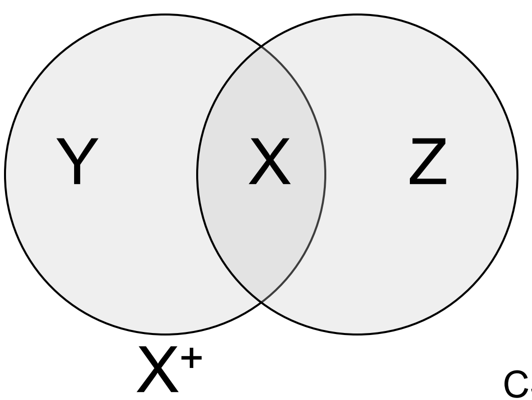
Equivalently:

Definition. A relation R is in BCNF if:

$\forall$ X, either $X^+ = X$ or $X^+ =$ [all attributes]

BCNF Decomposition Algorithm

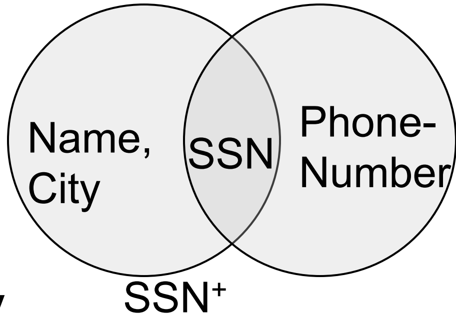
Normalize(R)
  find X s.t.: $X \neq X^+ \neq$ [all attributes]
  <u>if</u> (not found) **<u>then</u>** "R is in BCNF"
  <u>let</u> $Y = X^+ - X$; $Z =$ [all attributes] $- X^+$
  decompose R into $R1(X \cup Y)$ and $R2(X \cup Z)$
  Normalize(R1); Normalize(R2);



Example

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield
Joe	987-65-4321	908-555-1234	Westfield

SSN → Name, City



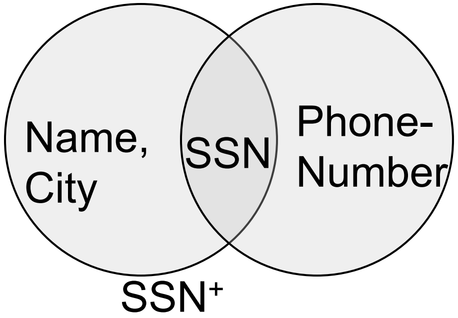
The only key is: {SSN, PhoneNumber}
Hence SSN → Name, City is a “bad” dependency

In other words:
SSN+ = Name, City and is neither SSN nor All Attributes

Example BCNF Decomposition

Name	SSN	City
Fred	123-45-6789	Seattle
Joe	987-65-4321	Westfield

SSN → Name, City



SSN	PhoneNumber
123-45-6789	206-555-1234
123-45-6789	206-555-6543
987-65-4321	908-555-2121
987-65-4321	908-555-1234

Let's check anomalies:

Redundancy ?

Update ?

Delete ?

Find X s.t.: X ≠ X^+ ≠ [all attributes]

Example BCNF Decomposition

Person(name, SSN, age, hairColor, phoneNumber)

SSN → name, age

age → hairColor

Find X s.t.: X $\neq X^+ \neq$ [all attributes]

Example BCNF Decomposition

Person(name, SSN, age, hairColor, phoneNumber)

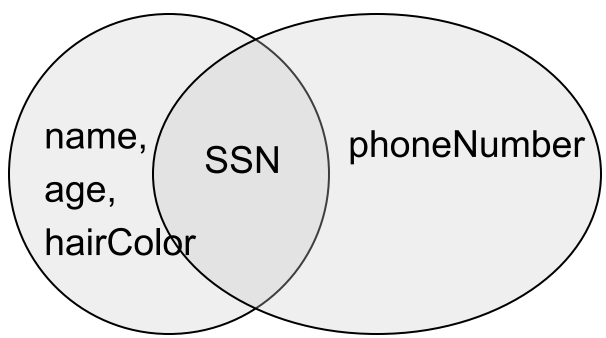
SSN → name, age

age → hairColor

Iteration 1: Person: SSN+ = SSN, name, age, hairColor

Decompose into: P(SSN, name, age, hairColor)

Phone(SSN, phoneNumber)



Find X s.t.: $X \neq X^{+} \neq$ [all attributes]

Example BCNF Decomposition

Person(name, SSN, age, hairColor, phoneNumber)

SSN → name, age

age → hairColor

What are the keys ?

Iteration 1: Person: SSN+ = SSN, name, age, hairColor

Decompose into: P(SSN, name, age, hairColor)
Phone(SSN, phoneNumber)

Iteration 2: P: age+ = age, hairColor

Decompose: People(SSN, name, age)
Hair(age, hairColor)
Phone(SSN, phoneNumber)

Find X s.t.: $X \neq X^+ \neq$ [all attributes]

Example BCNF Decomposition

Person(name, SSN, age, hairColor, phoneNumber)

SSN → name, age

age → hairColor

Note the keys!

Iteration 1: Person: SSN+ = SSN, name, age, hairColor

Decompose into: P(<u>SSN</u>, name, age, hairColor)
Phone(<u>SSN, phoneNumber</u>)

Iteration 2: P: age+ = age, hairColor

Decompose: People(<u>SSN</u>, name, age)
Hair(<u>age</u>, hairColor)
Phone(<u>SSN, phoneNumber</u>)

R(A,B,C,D)

$A \rightarrow B$
$B \rightarrow C$

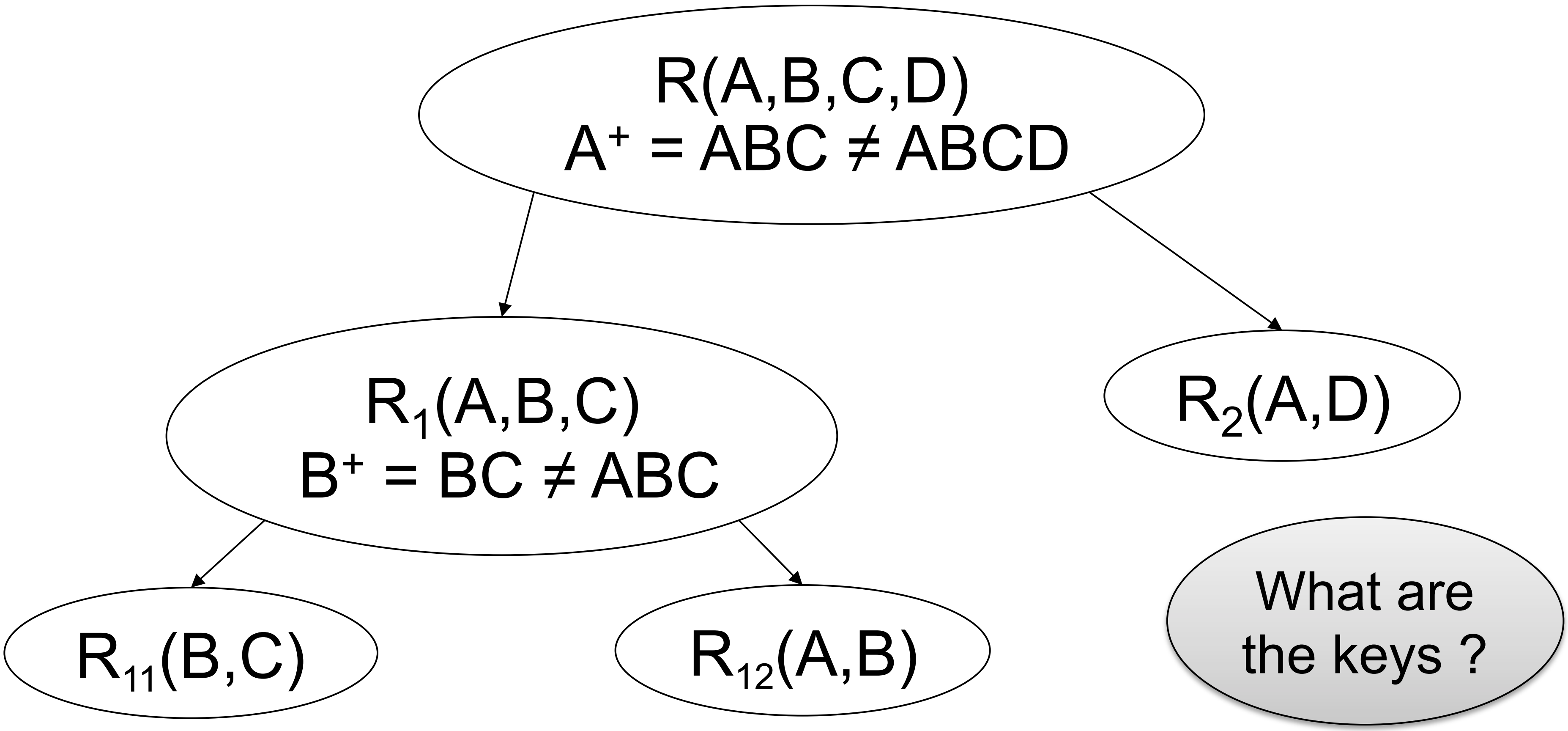
Practice at Home

R(A,B,C,D)
$A^+ = ABC \neq ABCD$

R(A,B,C,D)

$A \rightarrow B$
$B \rightarrow C$

Practice at Home



What happens if in R we first pick B^+ ? Or AB^+ ?

Schema Refinements = Normal Forms

- 1st Normal Form = all tables are flat
- 2nd Normal Form = obsolete
- Boyce Codd Normal Form = today
- 3rd Normal Form = see book

Outline

- Functional Dependencies and BCNF
  - Continued from previous lecture
- Storage and Indexes
  - Book: Ch. 8-11, and 20
  - PAX paper

Where We Are

- Part 1: The relational data model
  – First 2.5 lectures
- Part 2: Database Systems
  – Next 2.5 lectures
- Part 3: Transactions

- Part 4: Miscellaneous

The Mechanics of Disk

Mechanical characteristics:

- Rotation speed (5400RPM)
- Number of platters (1-30)
- Number of tracks (<=10000)
- Number of bytes/track(10^5)

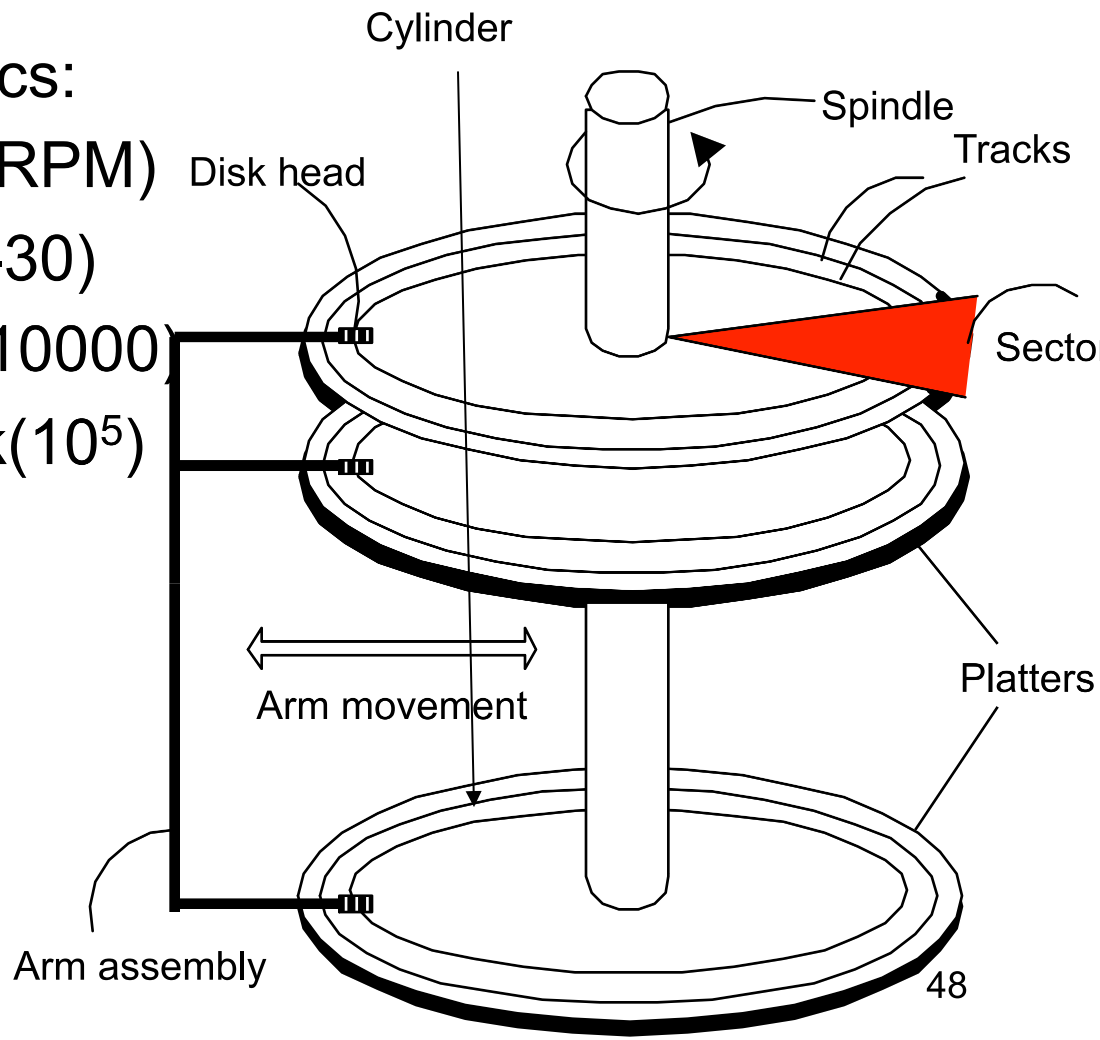
Unit of read or write:

disk block

Once in memory:

page

Typically: 4k or 8k or 16k



Disk Access Characteristics

- Disk latency
  - Time between when command is issued and when data is in memory
  - Equals = seek time + rotational latency
- Seek time = time for the head to reach cylinder
  - 10ms – 40ms
- Rotational latency = time for the sector to rotate
    - Rotation time = 10ms
    - Average latency = 10ms/2
- Transfer time = typically 40MB/s

Disks always read/write an entire block at a time

RAID: Redundant Array of Inexpensive Disks

Several disks that work in parallel

- Redundancy: use parity to recover from disk failure
- Speed: read from several disks at once

Various configurations (called *levels*):

- RAID 1 = mirror
- RAID 4 = n disks + 1 parity disk
- RAID 5 = n+1 disks, assign parity blocks round robin
- RAID 6 = “Hamming codes”

Storage Model

- DBMS needs spatial and temporal control over storage
  - Spatial control for performance
  - Temporal control for correctness and performance

- For spatial control, two alternatives
  - Use “raw” disk device interface directly
  - Use OS files

Spatial Control
Using “Raw” Disk Device Interface

- **Overview**
  - DBMS issues low-level storage requests directly to disk device
- **Advantages**
  - DBMS can ensure that important queries access data sequentially
  - Can provide highest performance
- **Disadvantages**
  - Requires devoting entire disks to the DBMS
  - Reduces portability as low-level disk interfaces are OS specific
  - Many devices are in fact “virtual disk devices”

Spatial Control Using OS Files

- **Overview**
  - DBMS creates one or more very large OS files
- **Advantages**
  - Allocating large file on empty disk can yield good physical locality
- **Disadvantages**
  - OS can limit file size to a single disk
  - OS can limit the number of open file descriptors
  - But these drawbacks have mostly been overcome by modern OSs

Commercial Systems

- Most commercial systems offer both alternatives
  - Raw device interface for peak performance
  - OS files more commonly used
- In both cases, we end-up with a DBMS file abstraction implemented on top of OS files or raw device interface

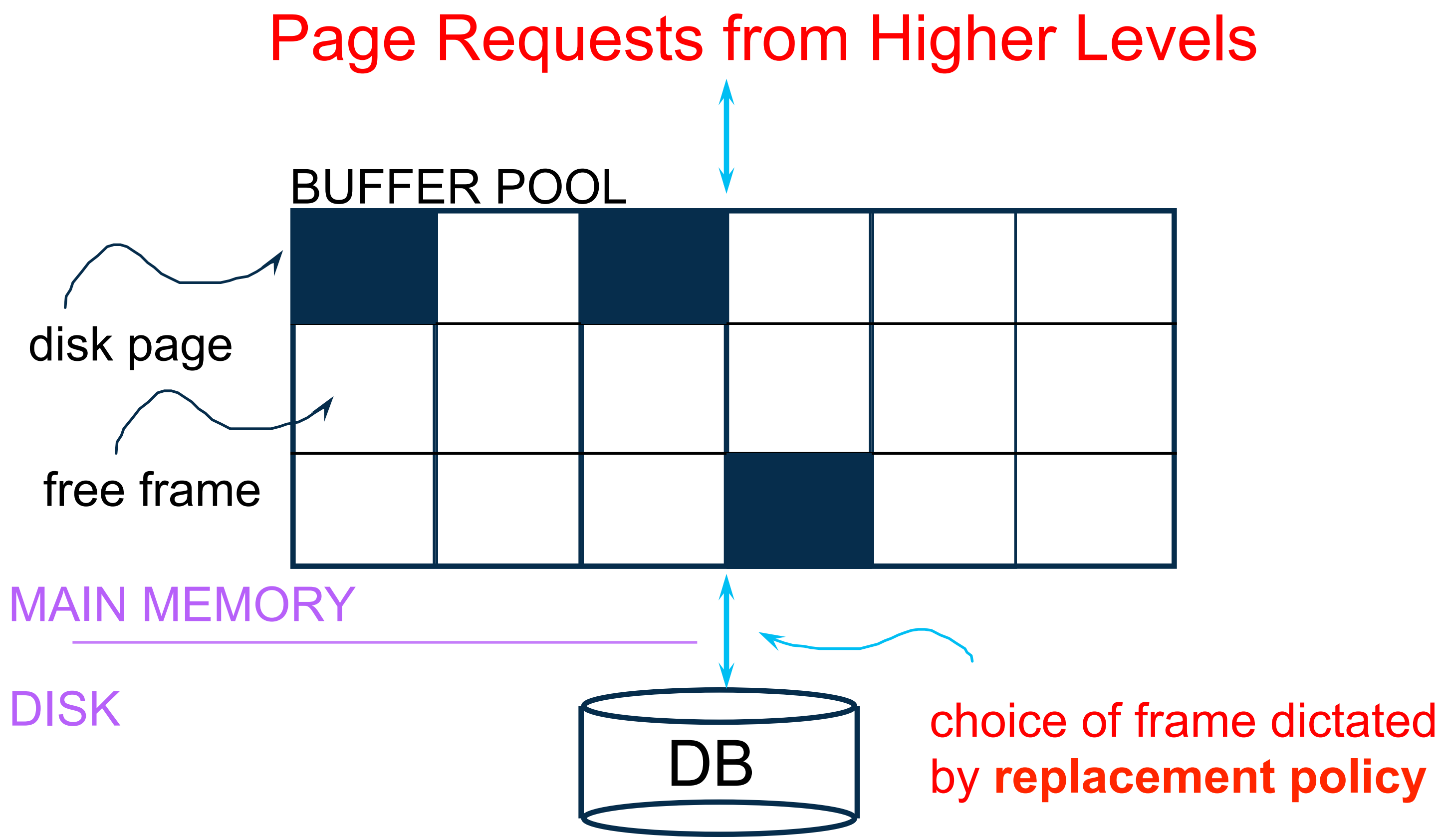
File Types

The data file can be one of:

- Heap file
  - Set of records, partitioned into blocks
  - Unsorted
- Sequential file
  - Sorted according to some attribute(s) called *key*

Note: "key" here means something else than "primary key"

Buffer Manager



- Data must be in RAM for DBMS to operate on it!
- Table of <frame#, pageid> pairs is maintained

Buffer Manager

Needs to decide on page replacement policy

- LRU
- Clock algorithm

Both work well in OS, but not always in DB

Enables the higher levels of the DBMS to assume that the needed data is in main memory.

Arranging Pages on Disk

A disk is organized into blocks (a.k.a. pages)

- blocks on same track, followed by
- blocks on same cylinder, followed by
- blocks on adjacent cylinder

A file should (ideally) consists of sequential blocks on disk, to minimize seek and rotational delay.

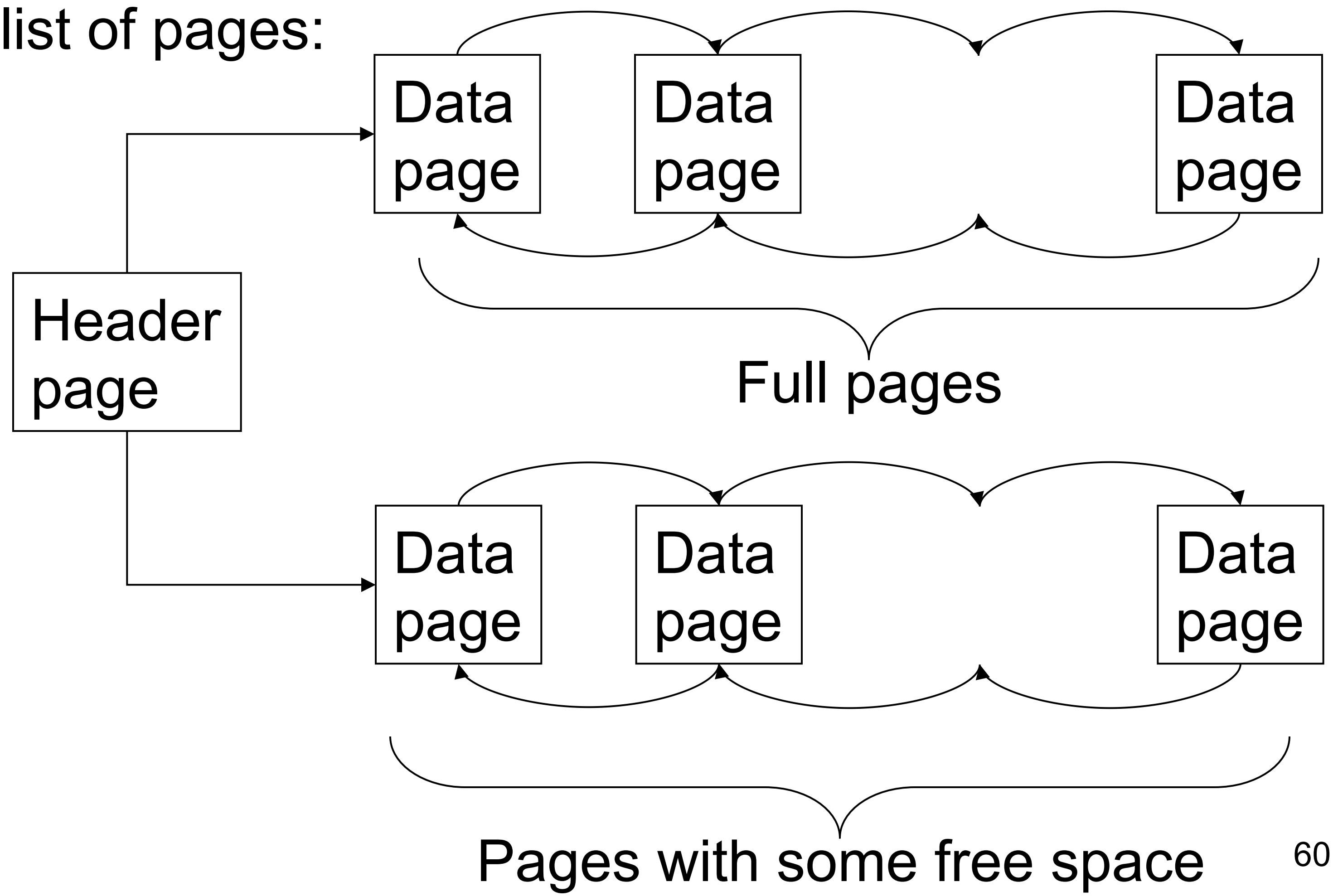
For a sequential scan, pre-fetching several pages at a time is a big win!

Issues

- File Organization
- Represent the records inside the blocks
- Represent attributes inside the records

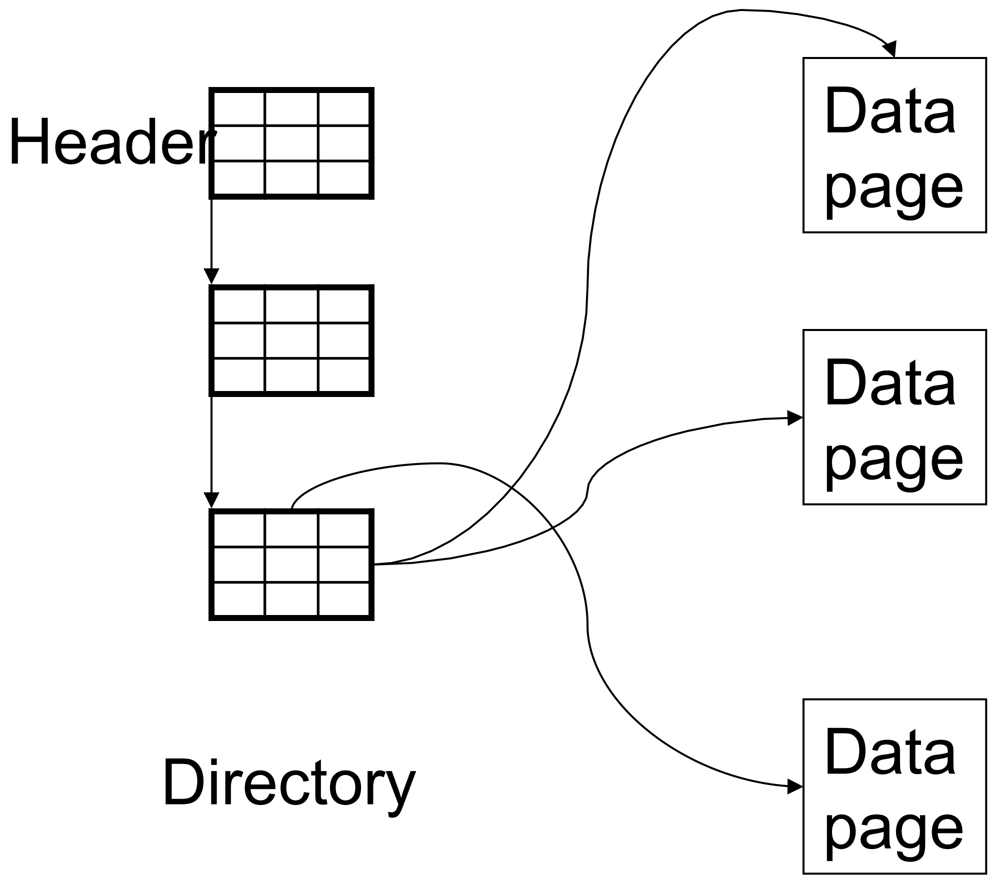
File Organization

Linked list of pages:



File Organization

Better: directory of pages



Page Formats

Issues to consider

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

Why do we need RID's in a relational DBMS ?

Record Format

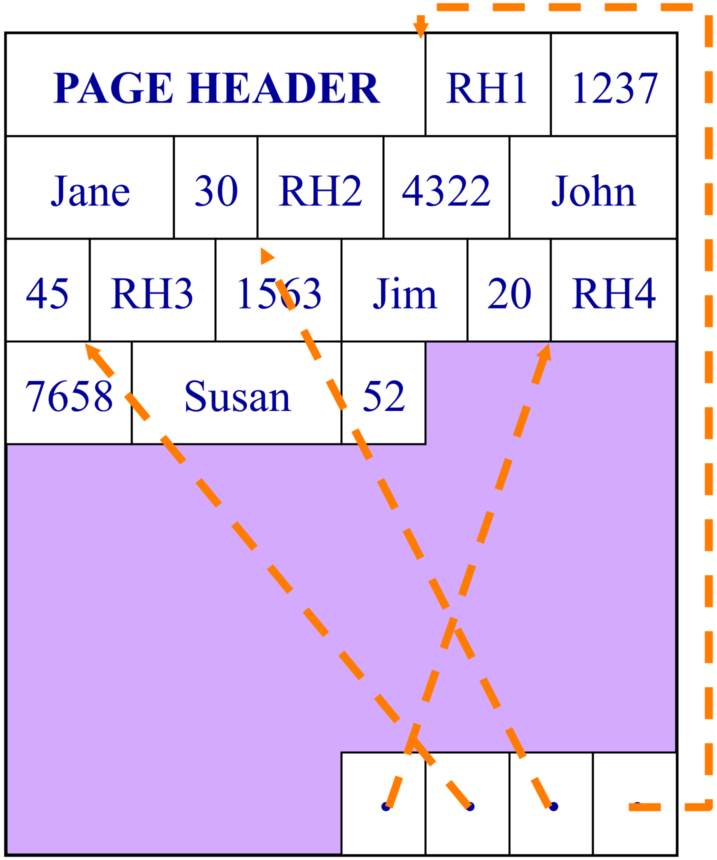
- Will discuss the PAX paper
- Q: Which of these bottlenecks does PAX address?
  - Disk access bottleneck
  - Memory access bottleneck

Current Scheme: Slotted Pages

Formal name: NSM (N-ary Storage Model)

R

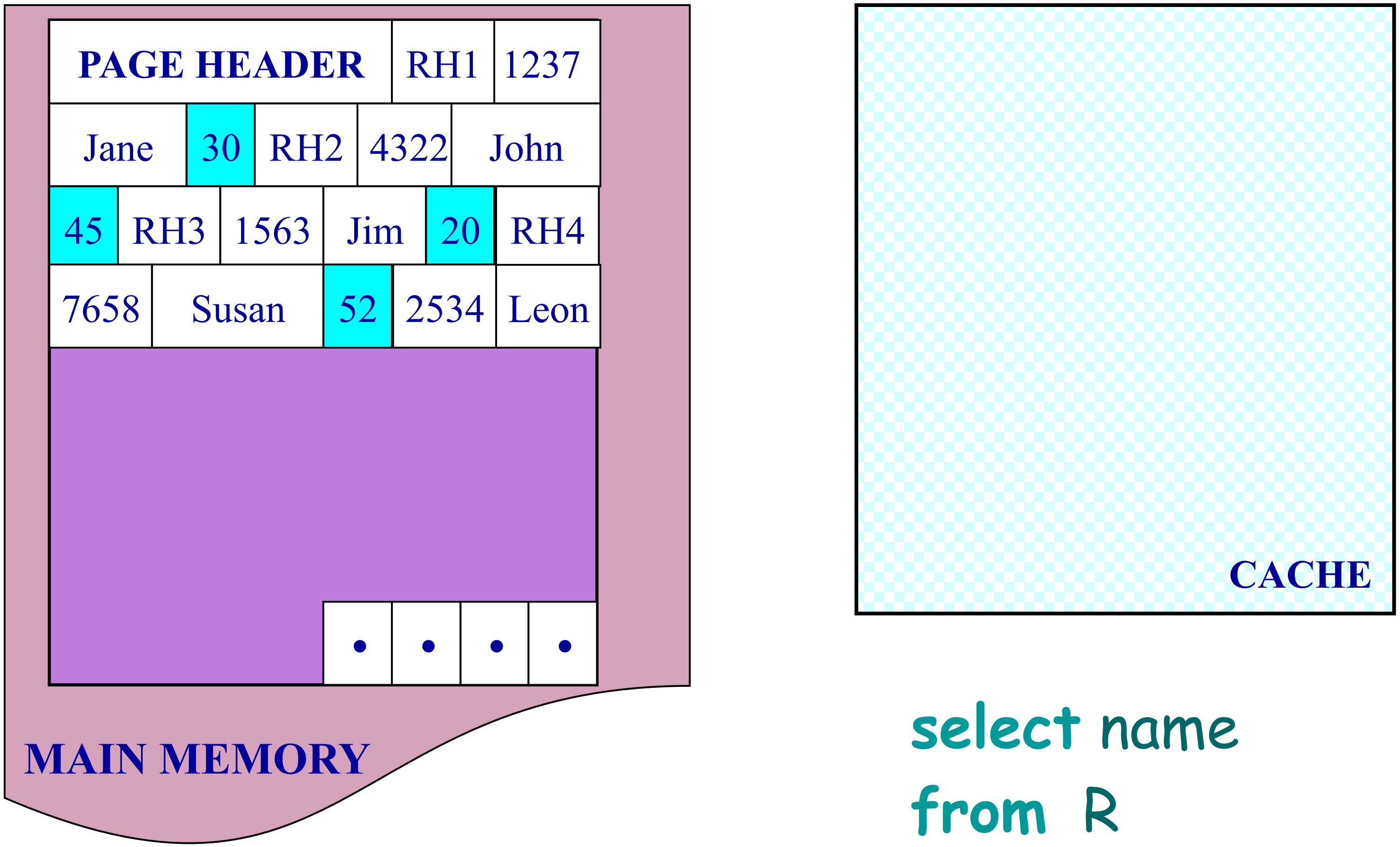
RID	SSN	Name	Age
1	1237	Jane	30
2	4322	John	45
3	1563	Jim	20
4	7658	Susan	52
5	2534	Leon	43
6	8791	Dan	37



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

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

Predicate Evaluation using NSM

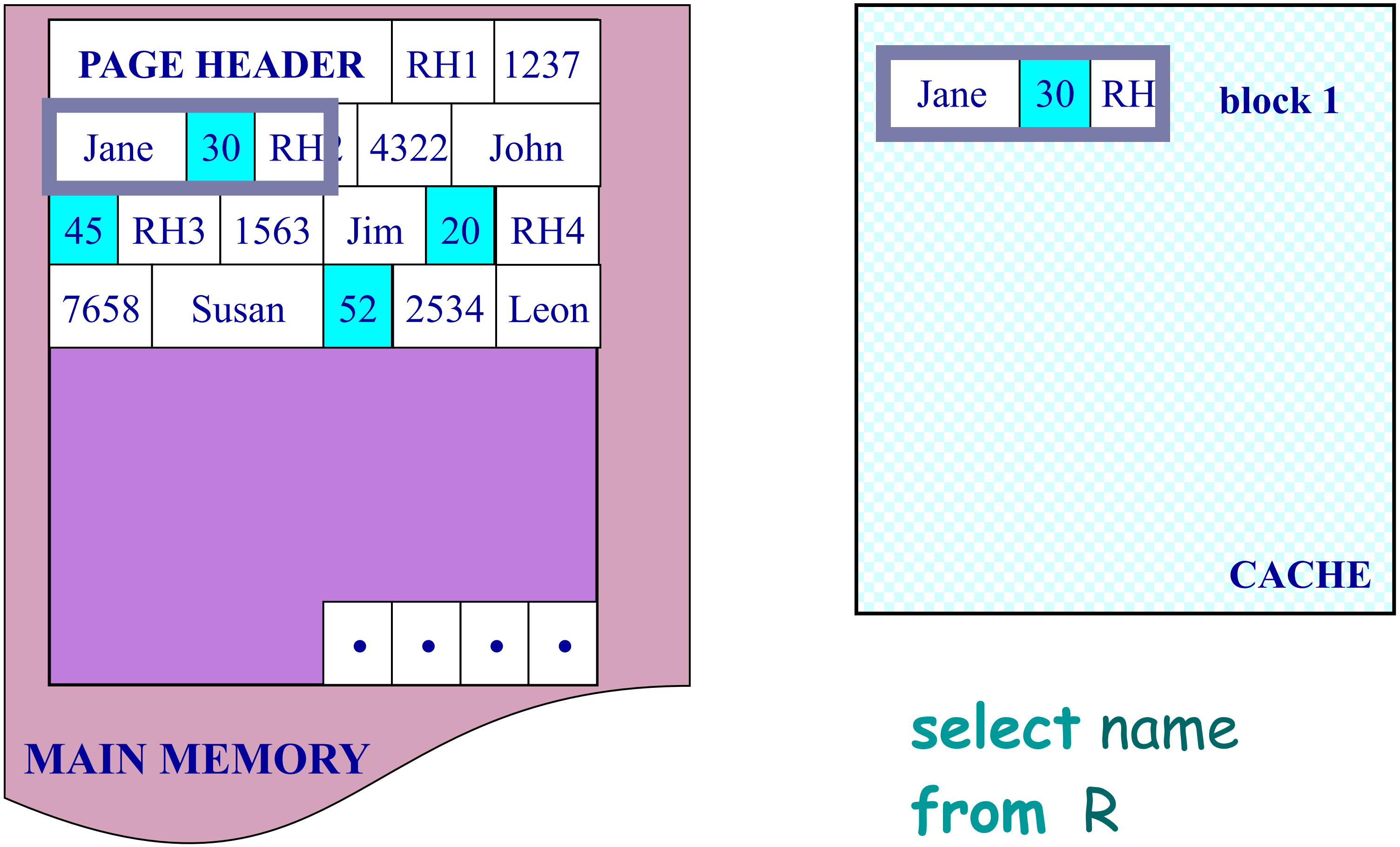



```
select name
from R
where age > 50
```

NSM pushes non-referenced data to the cache

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

Predicate Evaluation using NSM

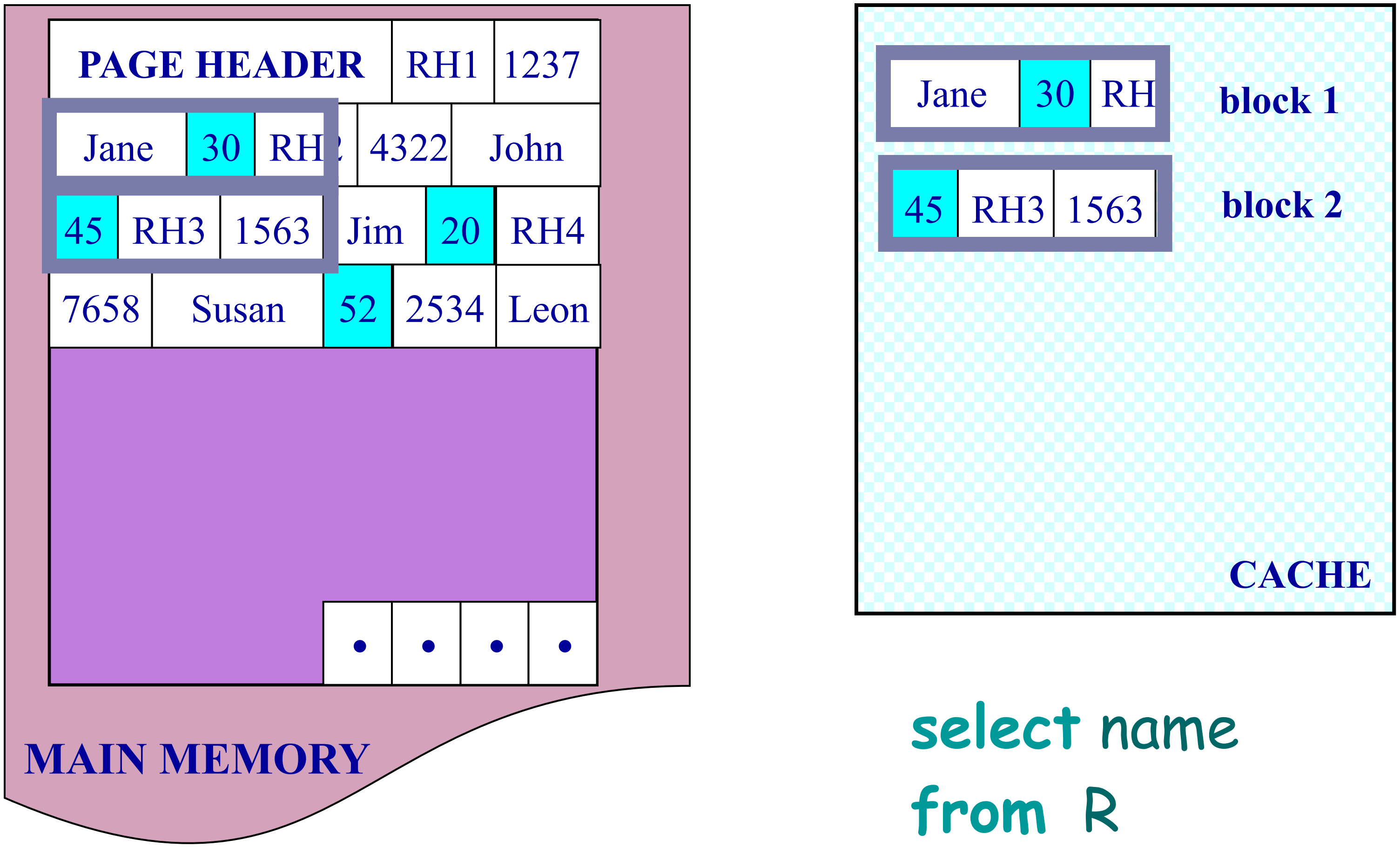


select name
from R
where age > 50

NSM pushes non-referenced data to the cache

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

Predicate Evaluation using NSM

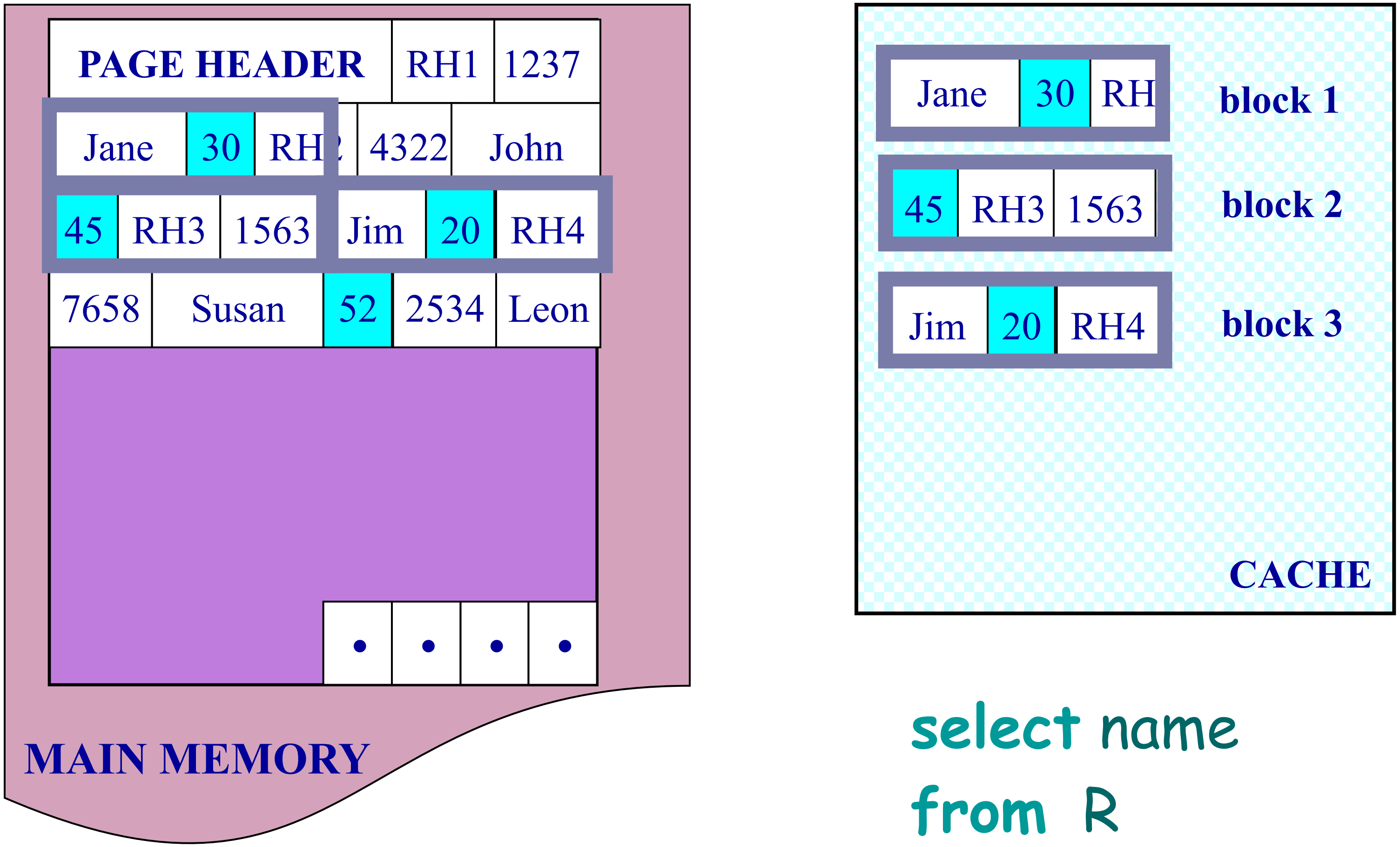


select name
from R
where age > 50

NSM pushes non-referenced data to the cache

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

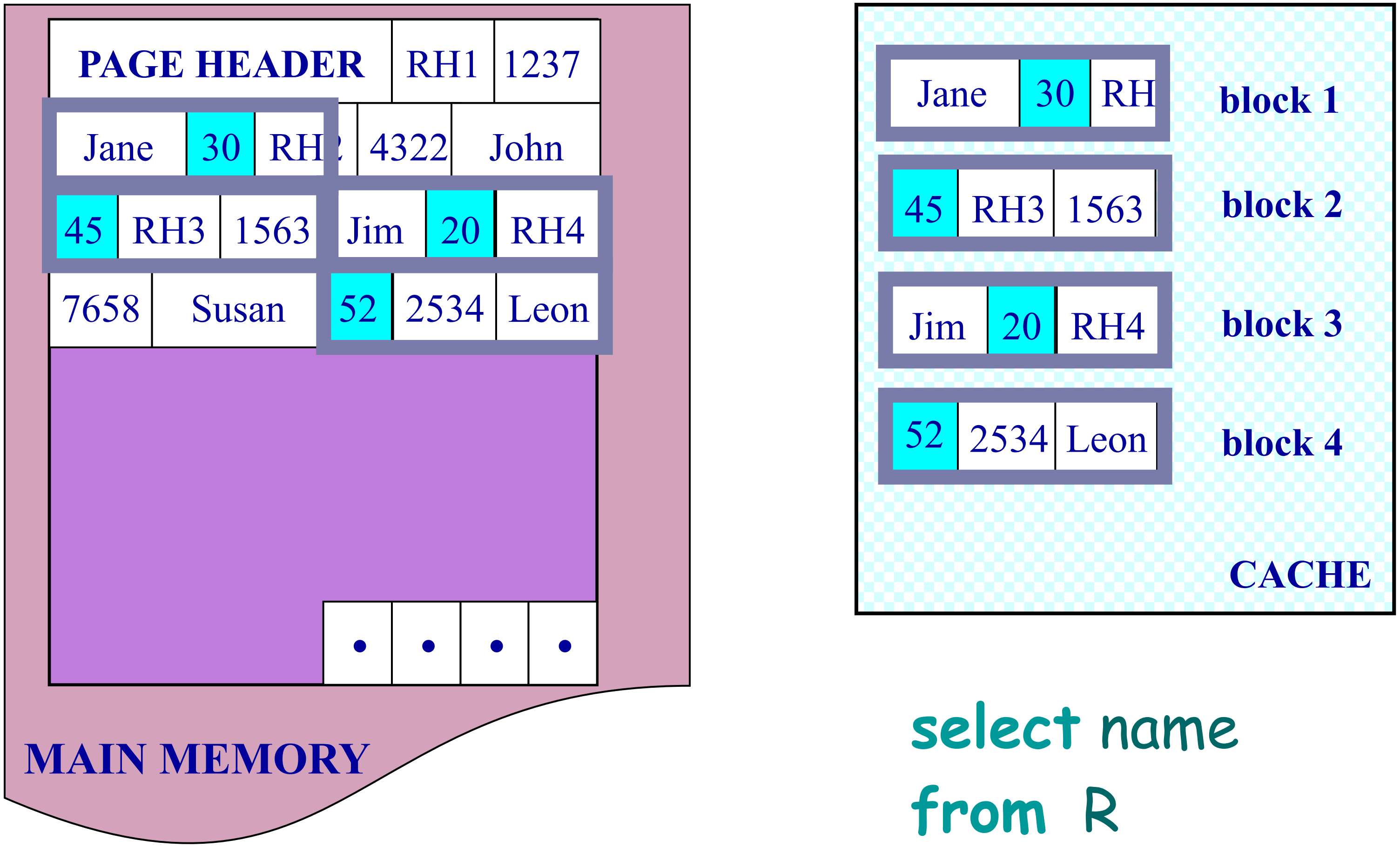
Predicate Evaluation using NSM



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

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

Need New Data Page Layout

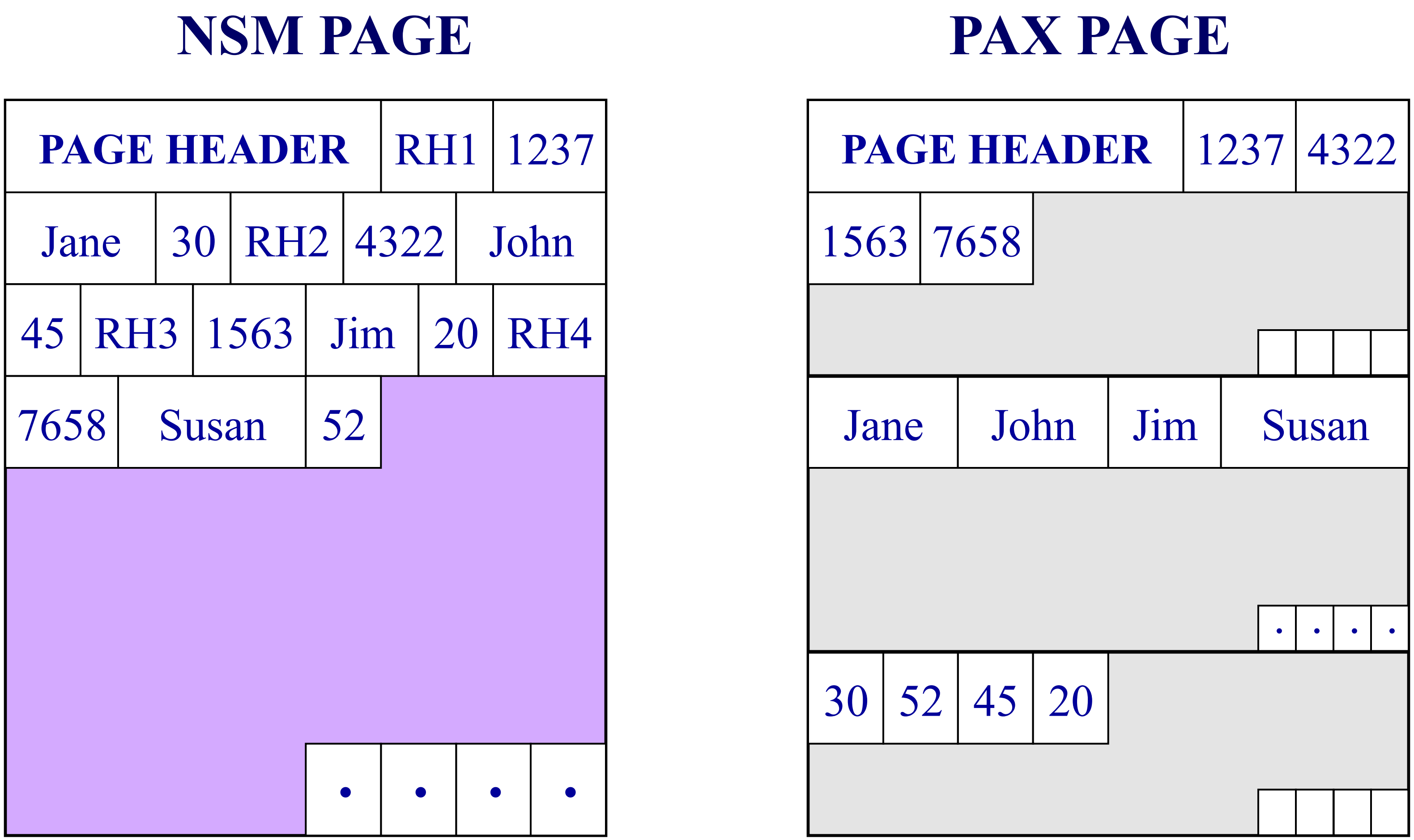
- Eliminates unnecessary memory accesses
- Improves inter-record locality
- Keeps a record’s fields together
- Does not affect I/O performance

and, most importantly, is…

low-implementation-cost, high-impact

Ailamaki VLDB’01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

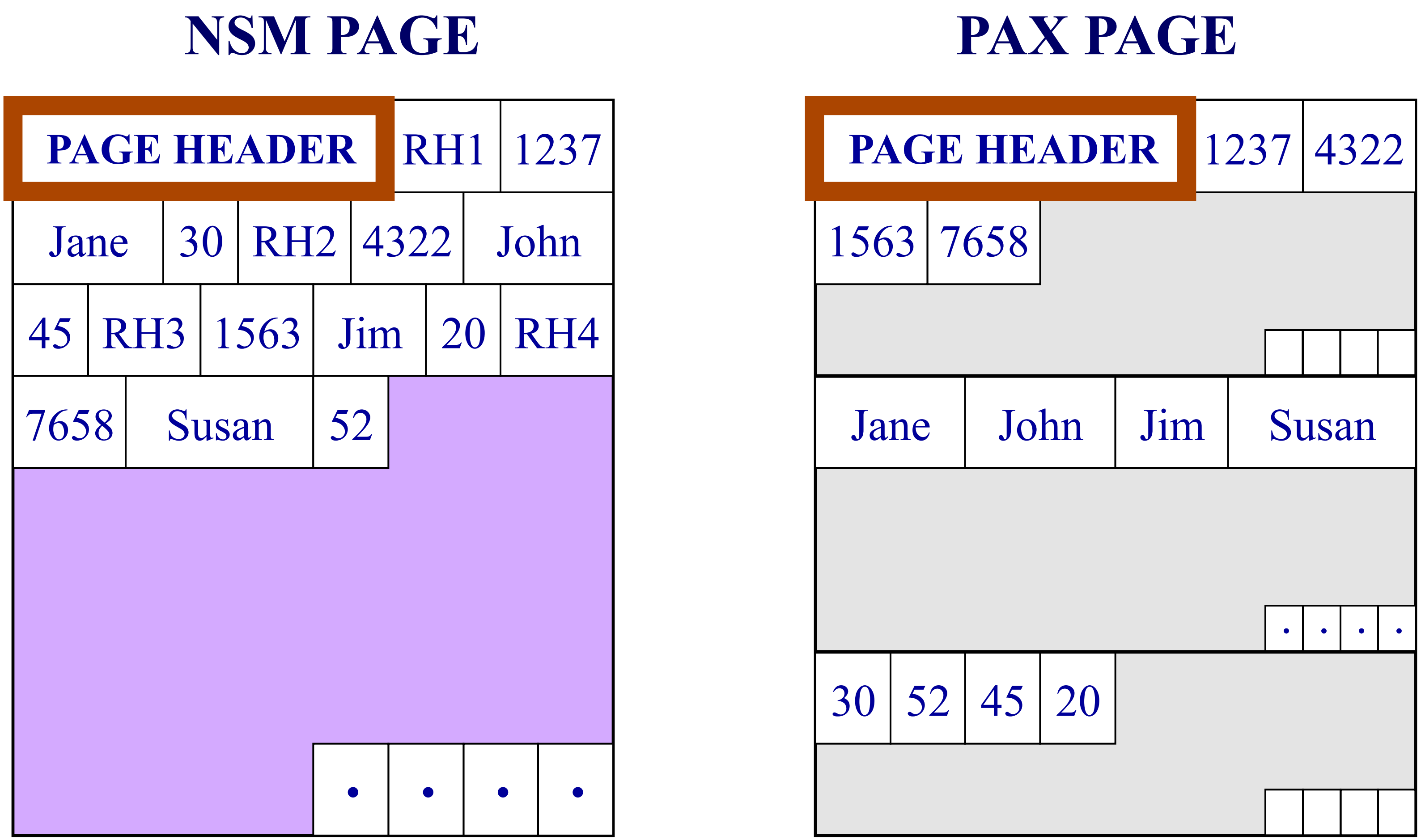
Partition Attributes Across (PAX)



Partition data *within* the page for spatial locality

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

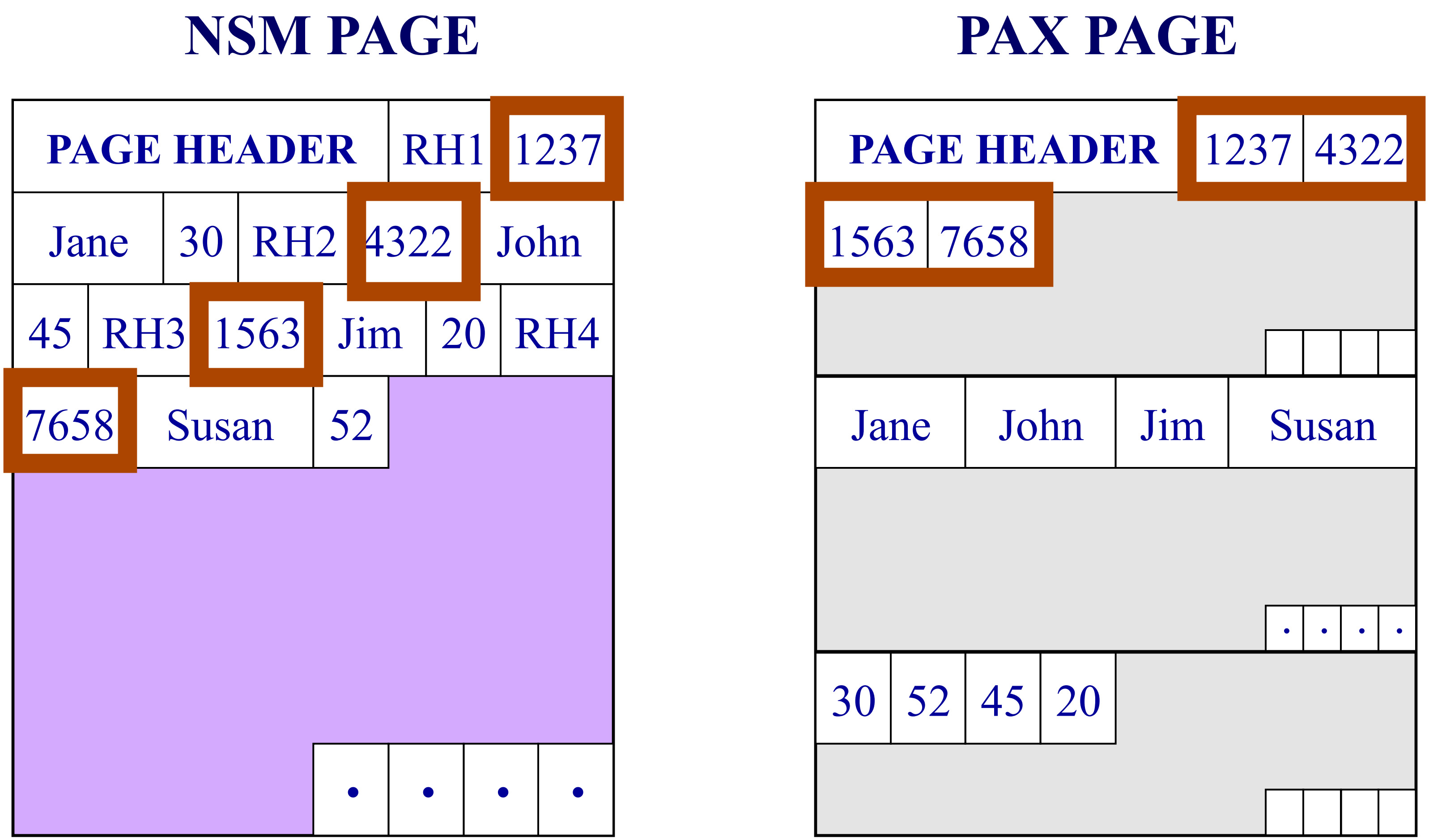
Partition Attributes Across (PAX)



Partition data *within* the page for spatial locality

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

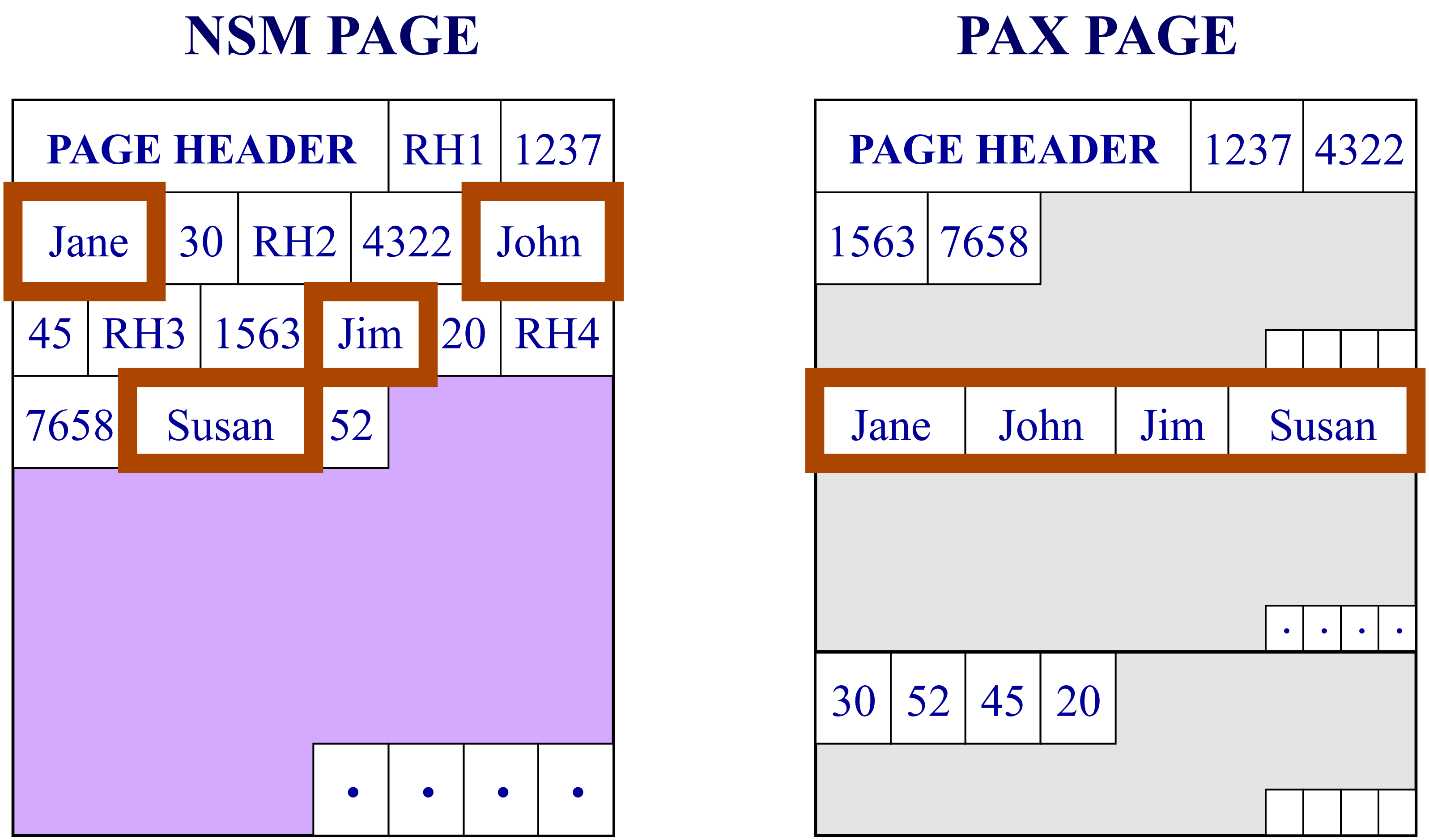
Partition Attributes Across (PAX)



Partition data *within* the page for spatial locality

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

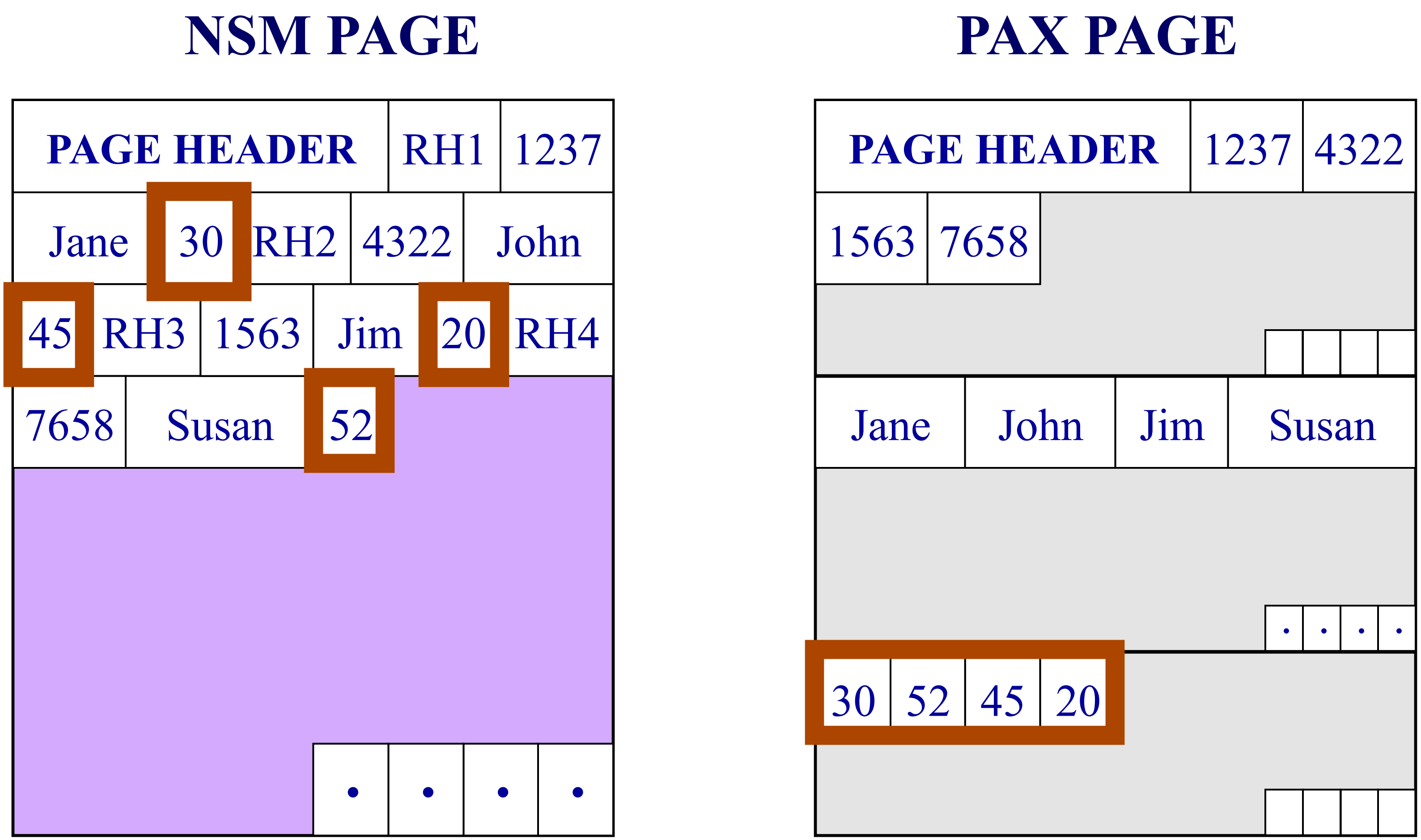
Partition Attributes Across (PAX)



Partition data *within* the page for spatial locality

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

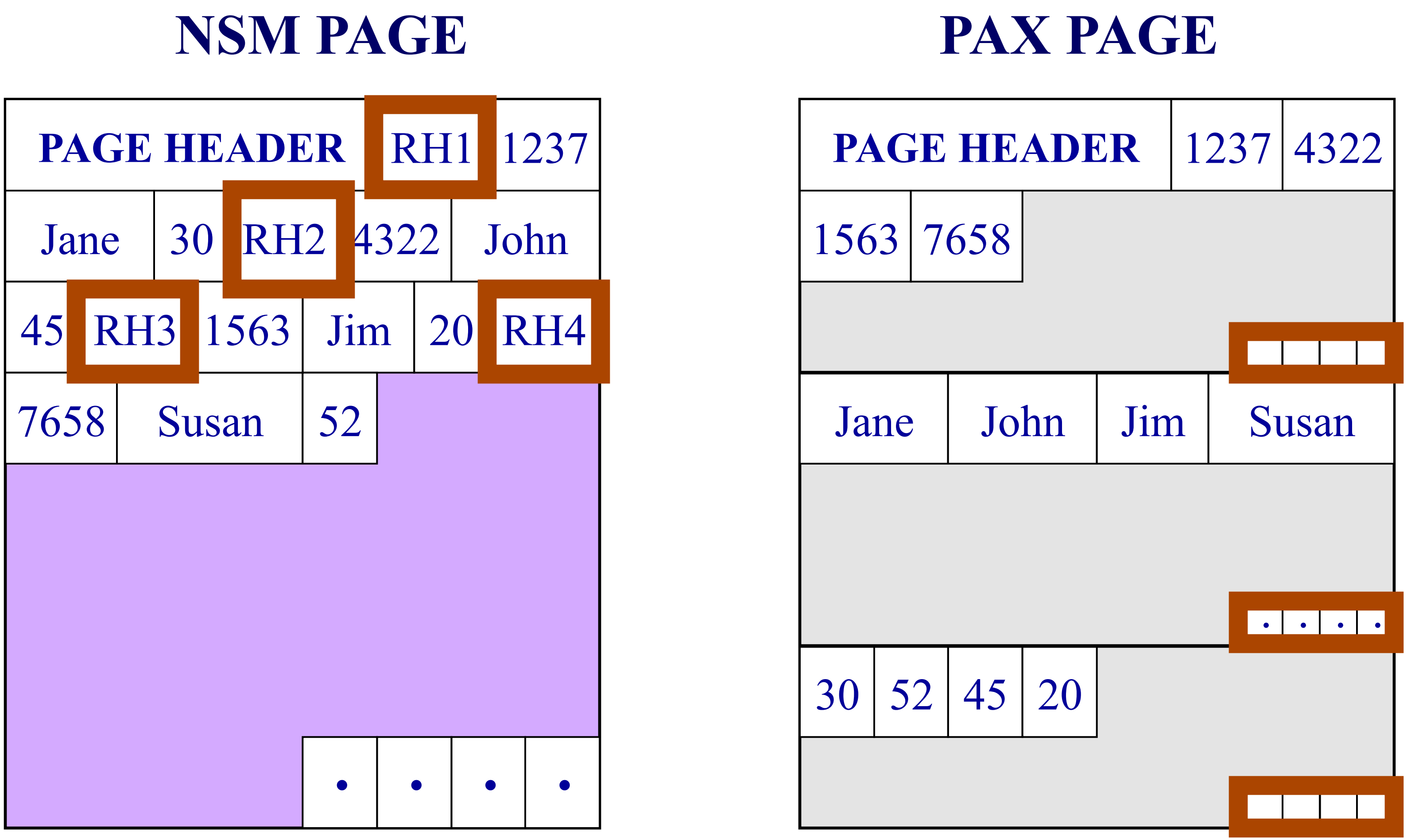
Partition Attributes Across (PAX)



Partition data *within* the page for spatial locality

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

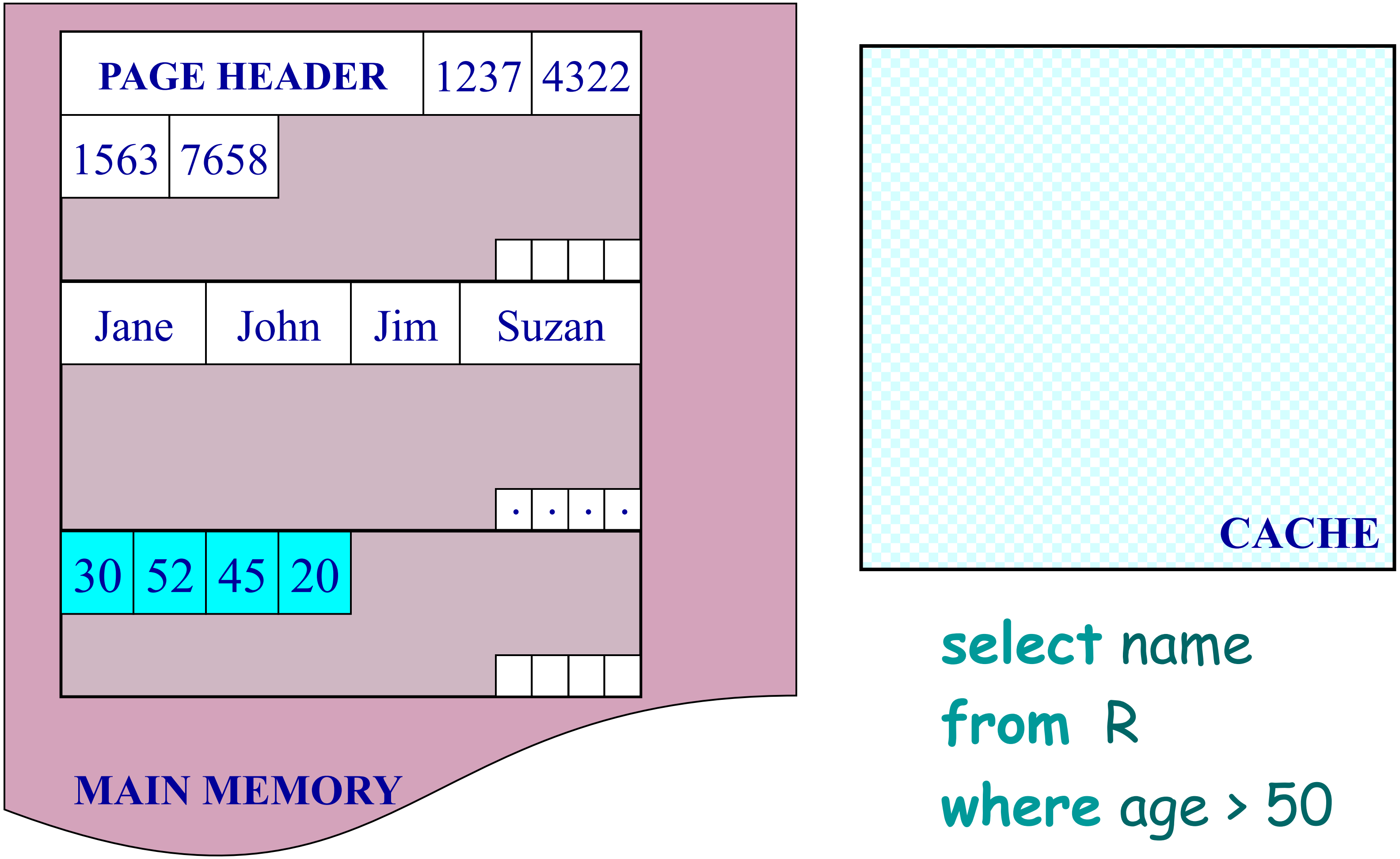
Partition Attributes Across (PAX)



Partition data *within* the page for spatial locality

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

Predicate Evaluation using PAX

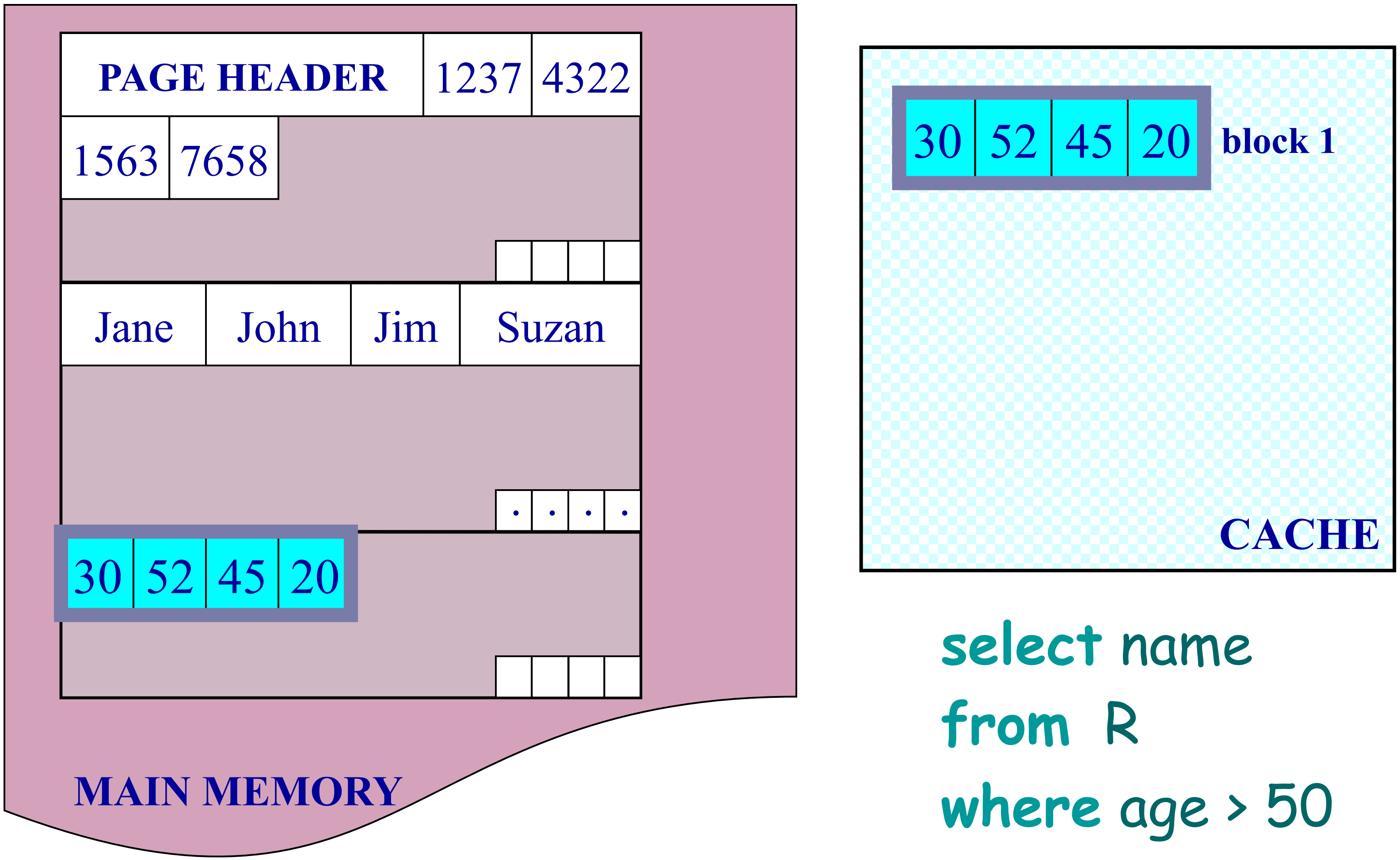



```
select name
from R
where age > 50
```

Fewer cache misses, low reconstruction cost

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

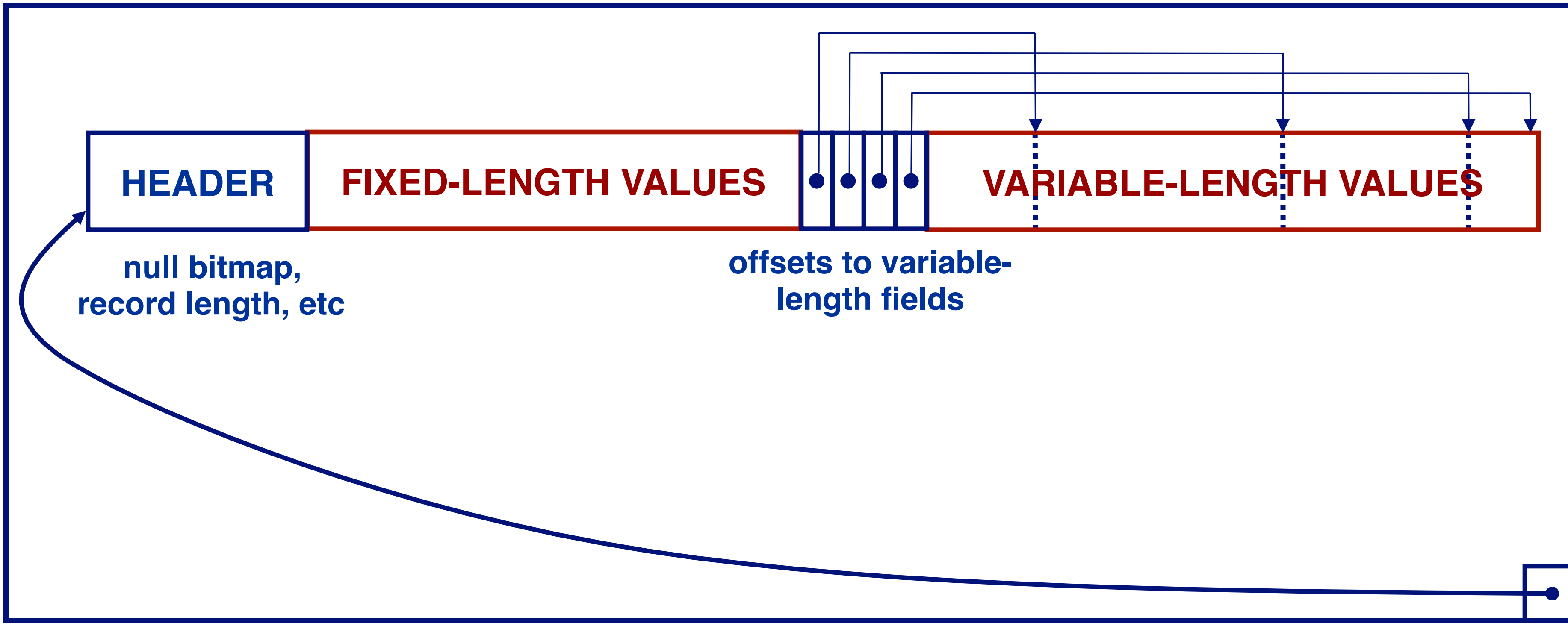
Predicate Evaluation using PAX



select name
from R
where age > 50

Fewer cache misses, low reconstruction cost

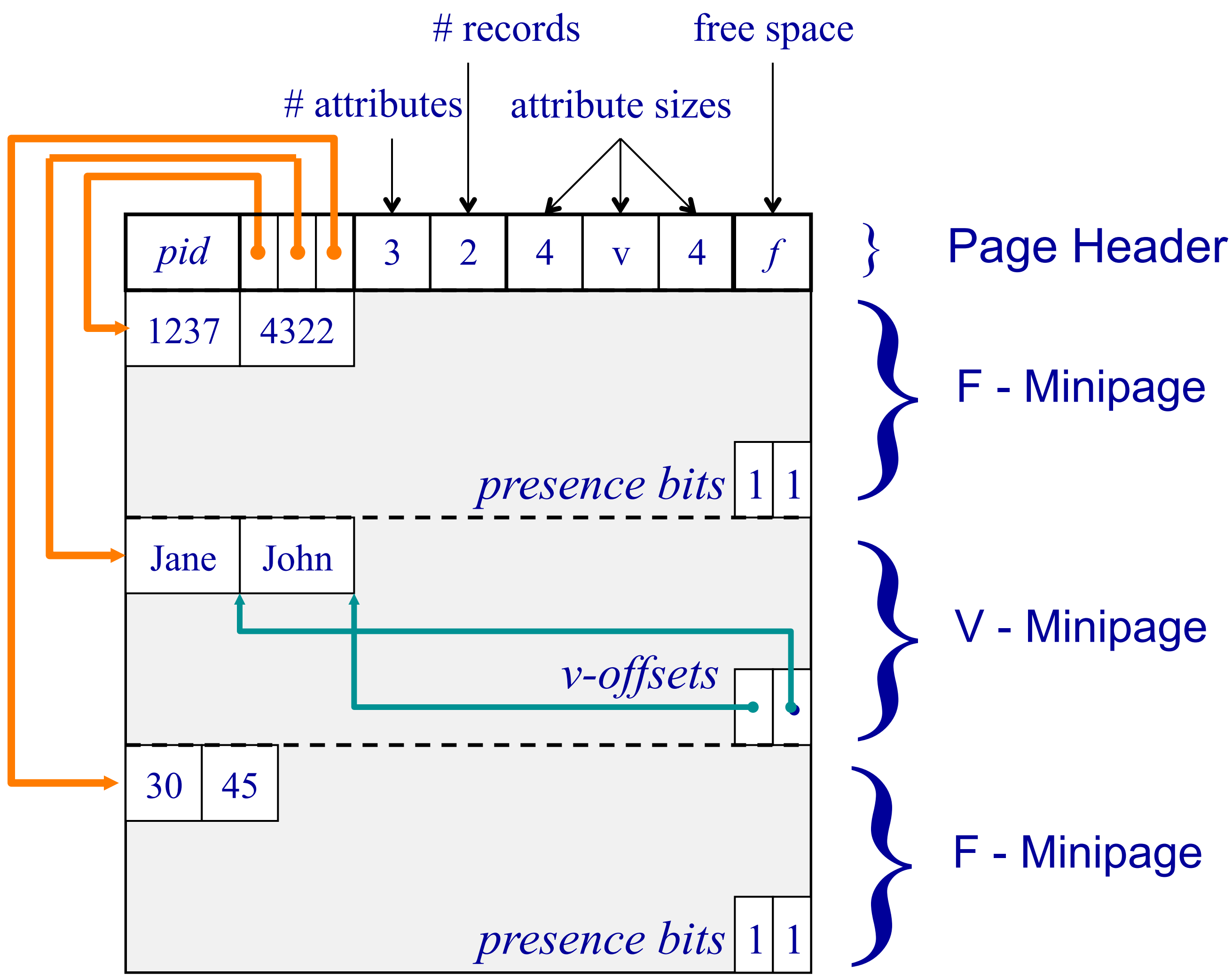
A Real NSM Record



NSM: All fields of record stored together + slots

Ailamaki VLDB’01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

PAX: Detailed Design



PAX: Group fields + amortizes record headers

Ailamaki VLDB’01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

BLOB

- Binary large objects
- Supported by modern database systems
- E.g. images, sounds, etc.
- Storage: attempt to cluster blocks together

CLOB = character large object

- Supports only restricted operations

File Organizations

- Heap files (random order)

- Sorted Files

- Indexes Trees or hash tables.
  - Like sorted files, they speed up searches for a subset of records, based on values in certain (“search key”) fields
  - Updates are much faster than in sorted files.

Index

- A (possibly separate) file, that allows fast access to records in the data file
- Contains (key, value) pairs:
  - The key k = an attribute value
  - The value k* = one of:
    - pointer to the record — *secondary index*
    - or the record itself — *primary index*

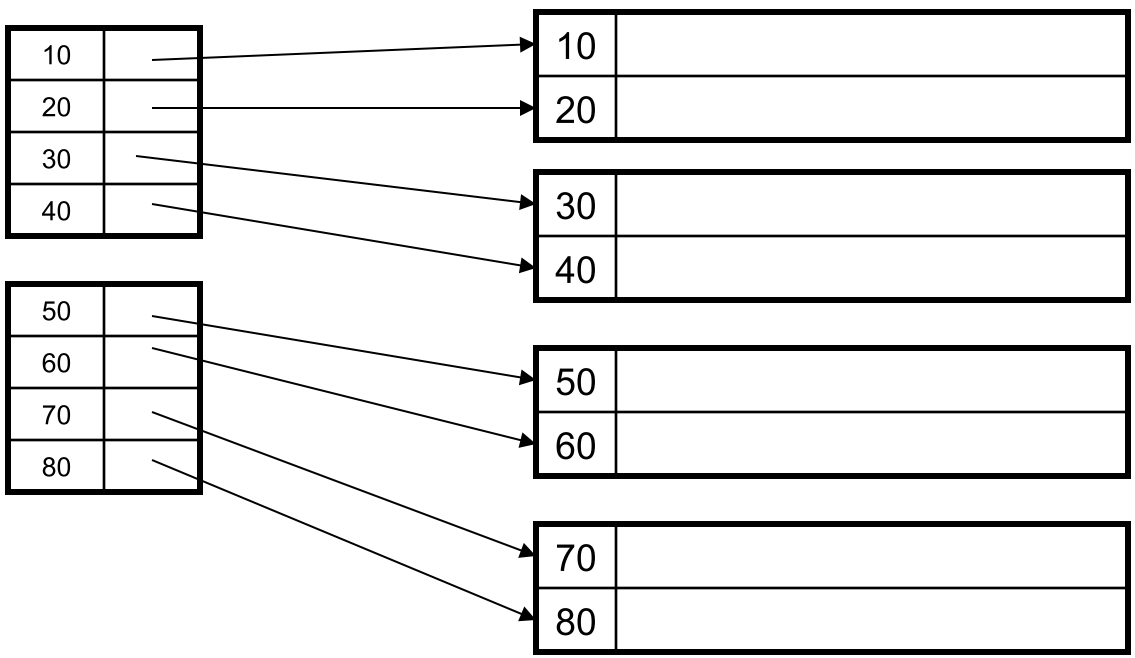
Note: “key” (aka “search key”) again means something else

Index Classification

- Clustered/unclustered
  - Clustered = records close in index are close in data
  - Unclustered = records close in index may be far in data
- Primary/secondary
  - Meaning 1:
    - Primary = is over attributes that include the primary key
    - Secondary = otherwise
  - Meaning 2: means the same as clustered/unclustered
- Organization B+ tree or Hash table

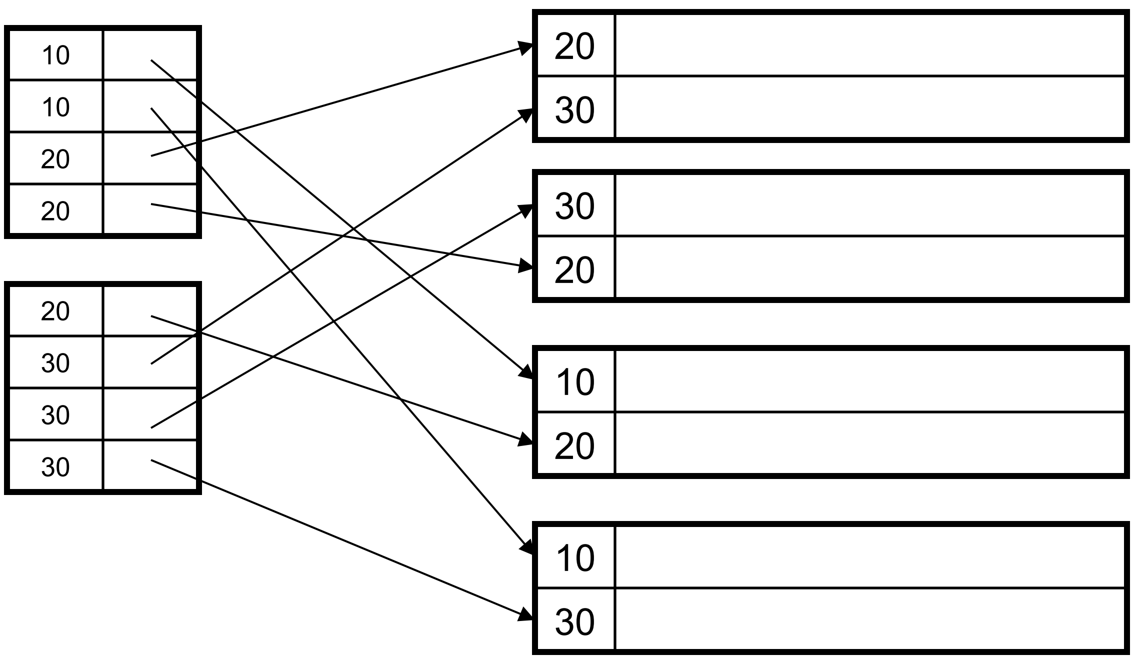
Clustered Index

- File is sorted on the index attribute
- Only one per table

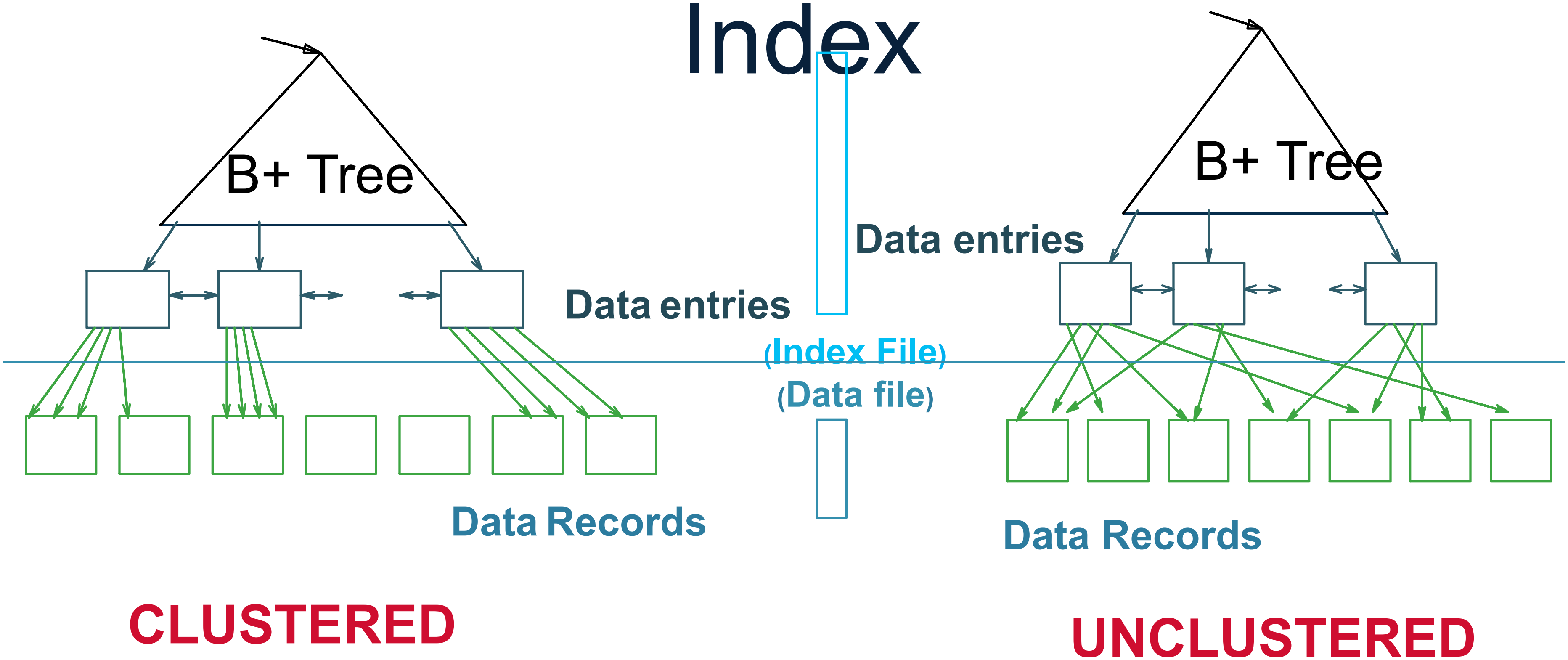


Unclustered Index

- Several per table

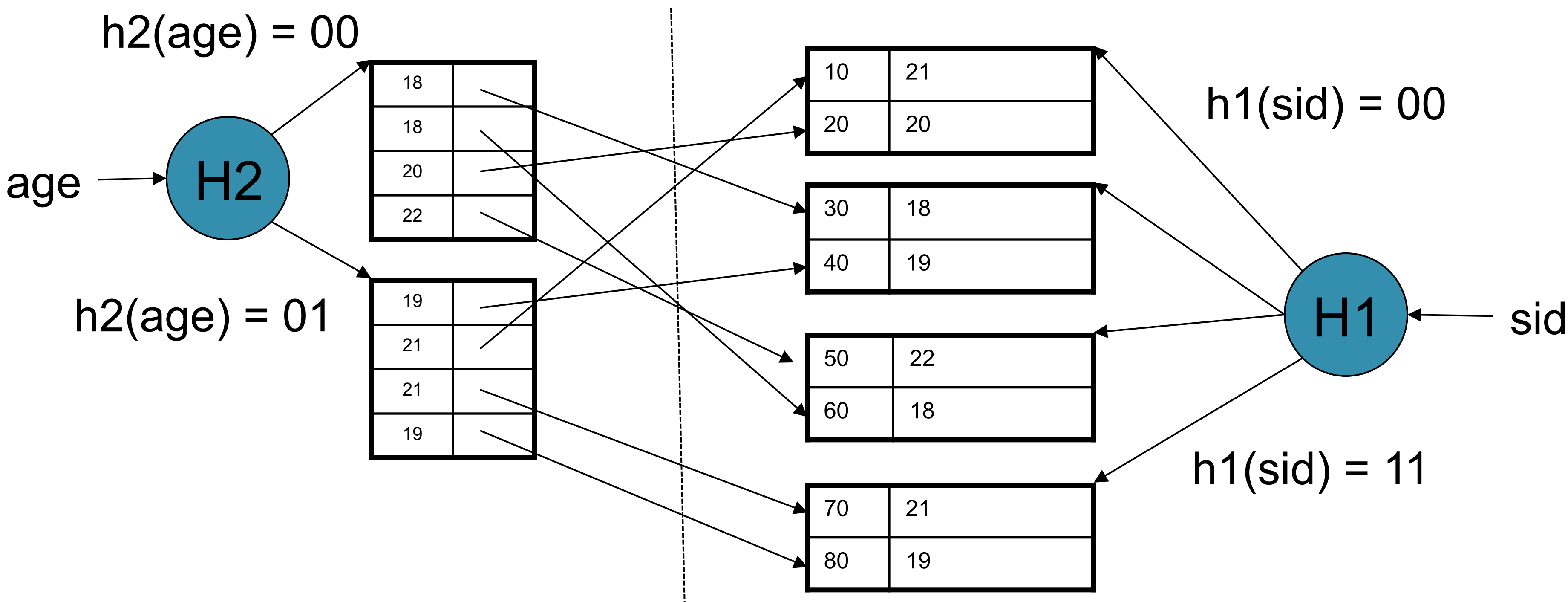


Clustered vs. Unclustered Index



Hash-Based Index

Good for point queries but not range queries



Another example
of unclustered/secondary index

Another example of
clustered/primary index

Alternatives for Data Entry k* in Index

Three alternatives for **k***:

- Data record with key value **k**
- <**k**, **rid** of data record with key = **k**>
- <**k**, list of **rids** of data records with key = **k**>

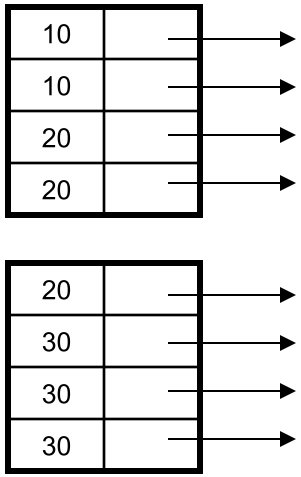
Alternatives 1, 2, 3

k* = data record

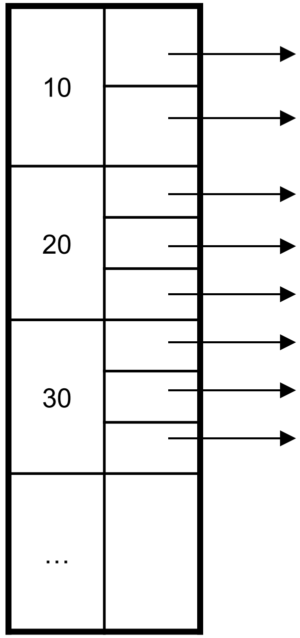
10	ssn	age	…
10	ssn	age	…
20	ssn	age	…
20	ssn	age	…

20	ssn	age	…
30	ssn	age	…
30	ssn	age	…
30	ssn	age	…

k* = rid



k* = list of rid's



B+ Trees

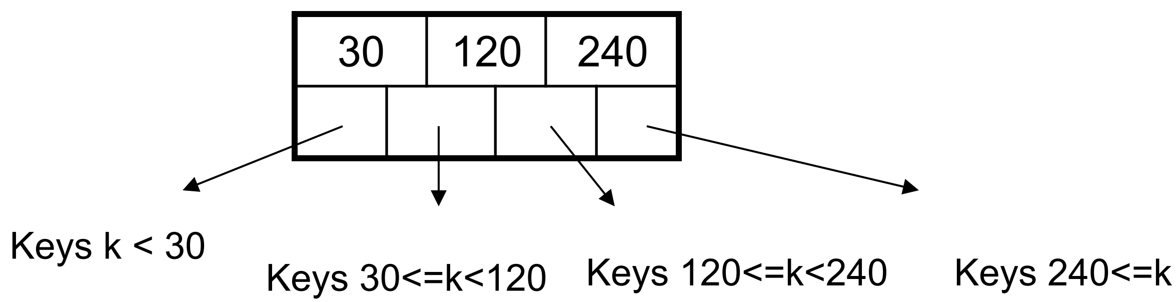
- Search trees

- Idea in B Trees
  - Make 1 node = 1 block
  - Keep tree balanced in height

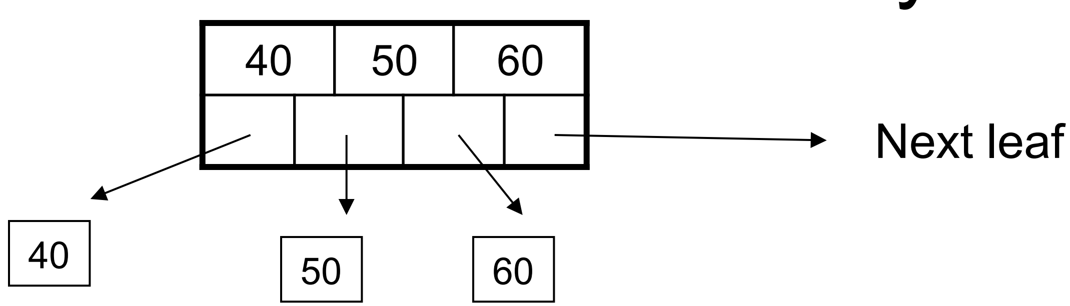
- Idea in B+ Trees
  - Make leaves into a linked list: facilitates range queries

B+ Trees Basics

- Parameter d = the *degree*
- Each node has >= d and <= 2d keys (except root)



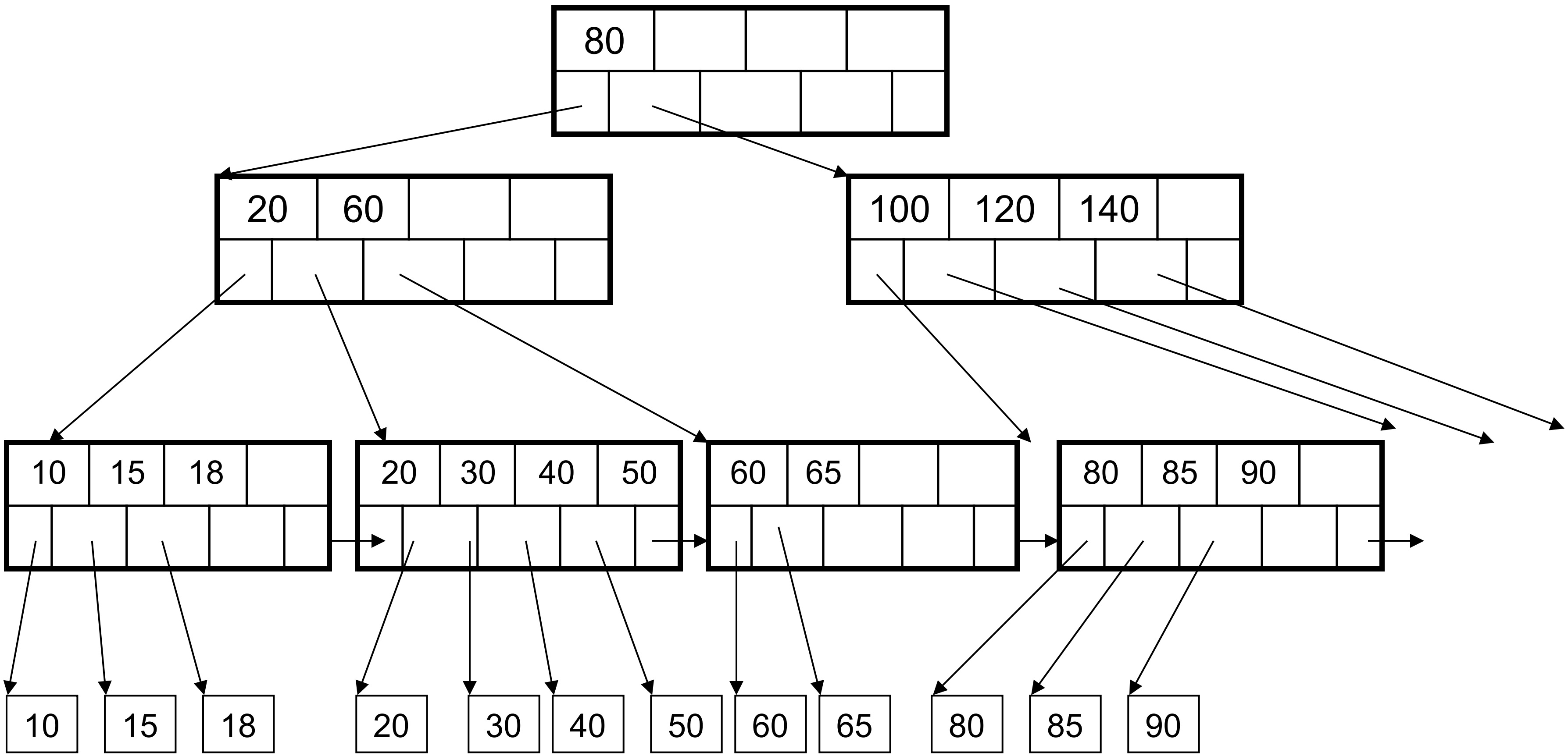
- Each leaf has >=d and <= 2d keys:



B+ Tree Example

d = 2

Find the key 40

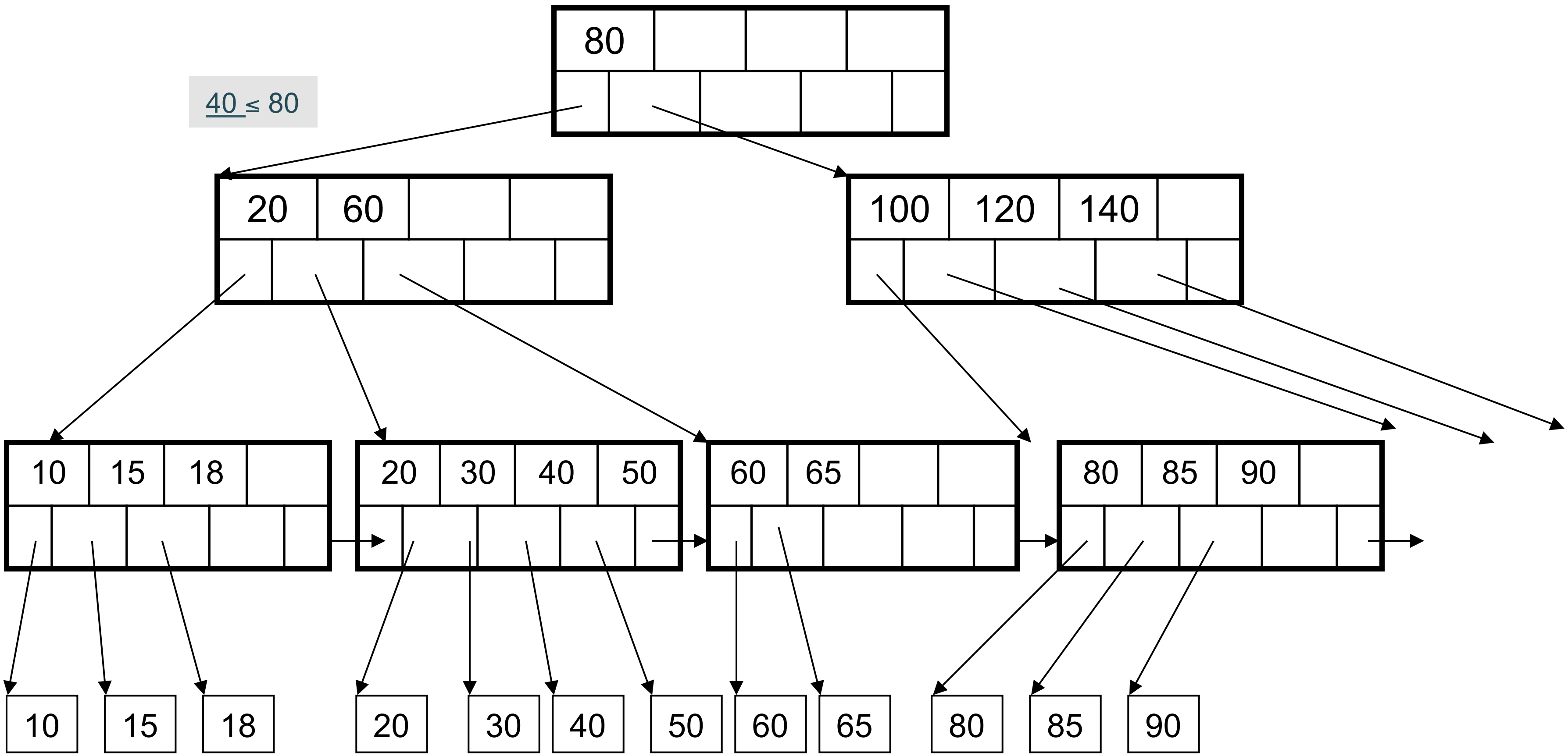


B+ Tree Example

d = 2

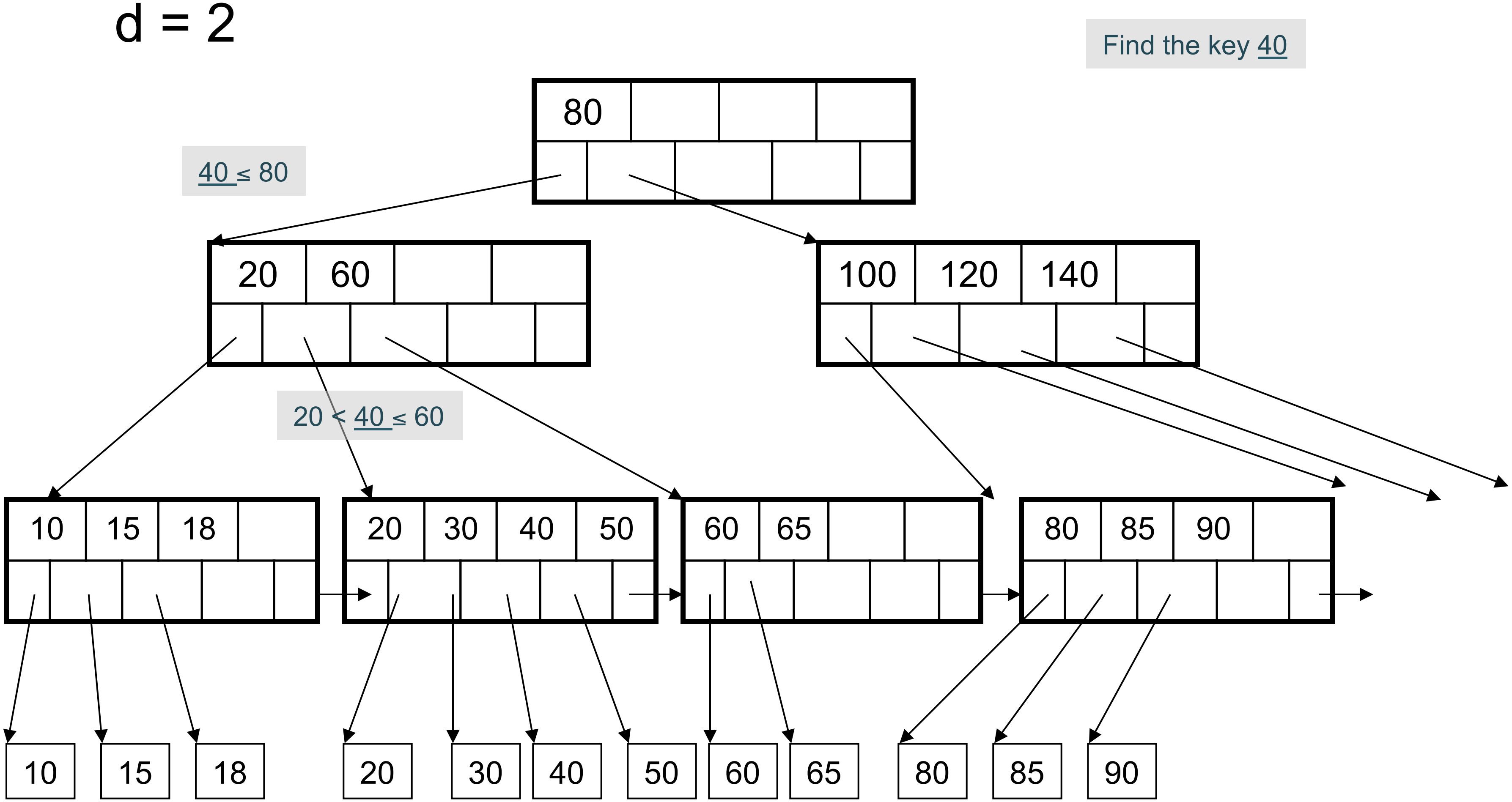
Find the key <u>40</u>

<u>40</u> ≤ 80



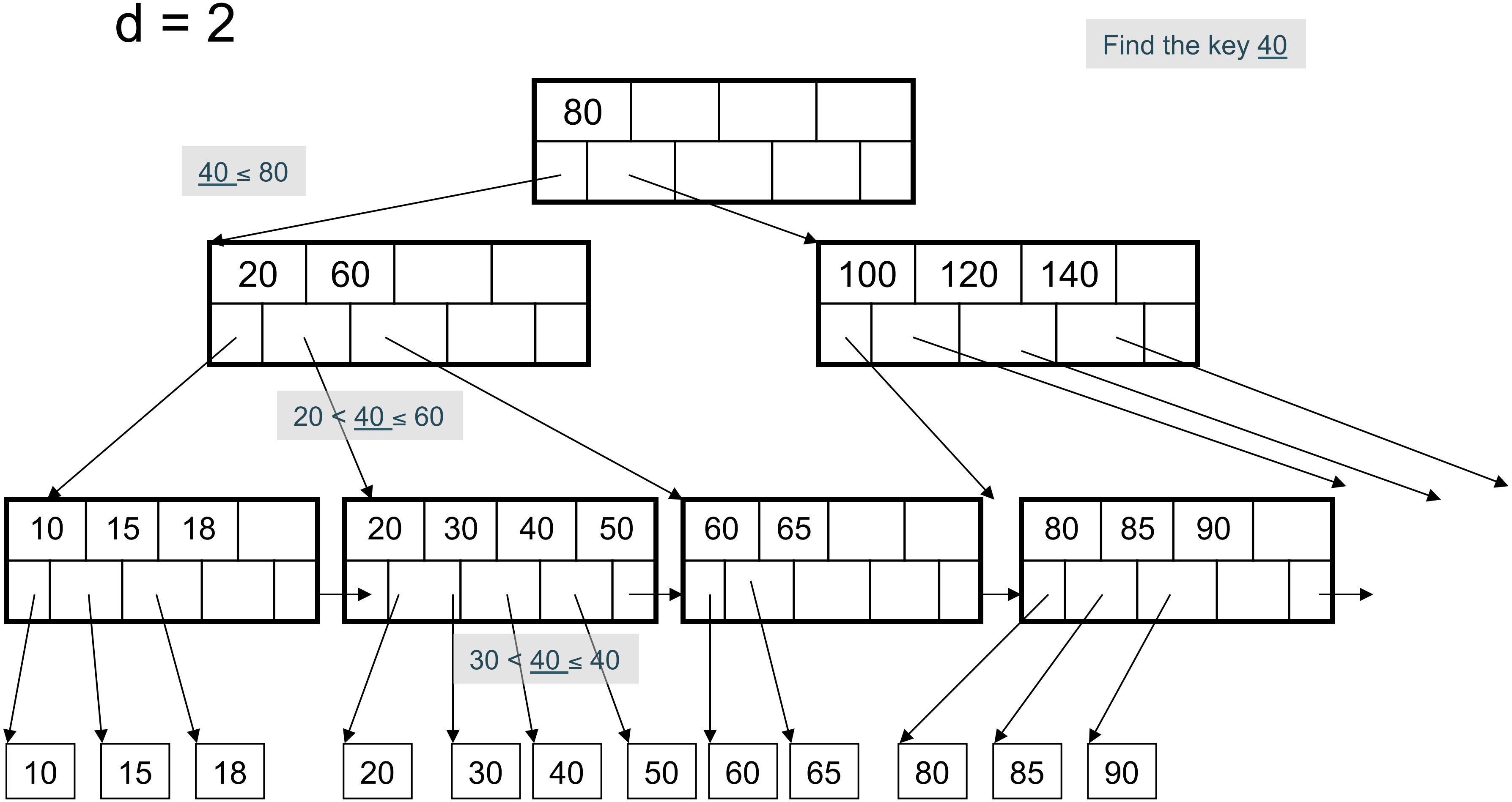
B+ Tree Example

d = 2



B+ Tree Example

d = 2



Using a B+ Tree

Index on People(age)

- Exact key values:
  - Start at the root
  - Proceed down, to the leaf

```
SELECT name
FROM People
WHERE age = 25
```

- Range queries:
  - As above
  - Then sequential traversal

```
SELECT name
FROM People
WHERE 20 <= age
  and  age <= 30
```

Which queries can use this index ?

Index on People(name, zipcode)

```
SELECT *
FROM People
WHERE name = 'Smith'
   and zipcode = 12345
```

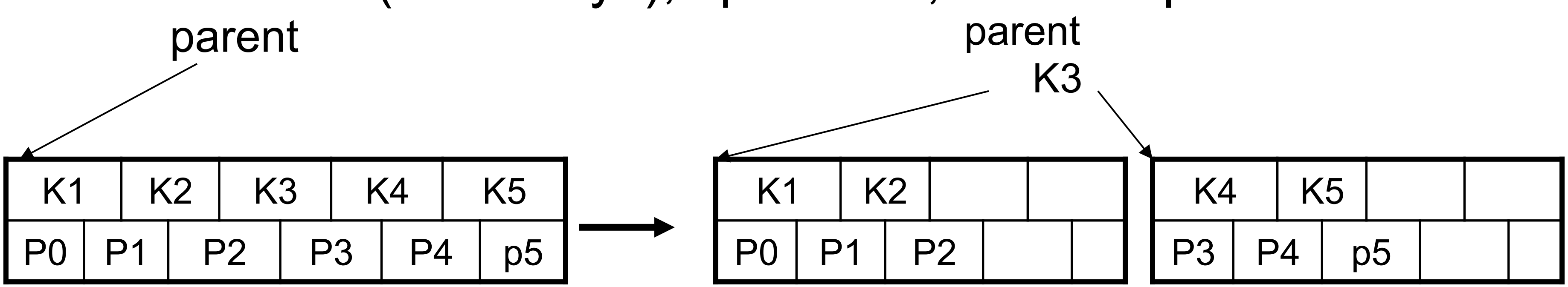
```
SELECT *
FROM People
WHERE name = 'Smith'
```

```
SELECT *
FROM People
WHERE zipcode = 12345
```

Insertion in a B+ Tree

Insert (K, P)

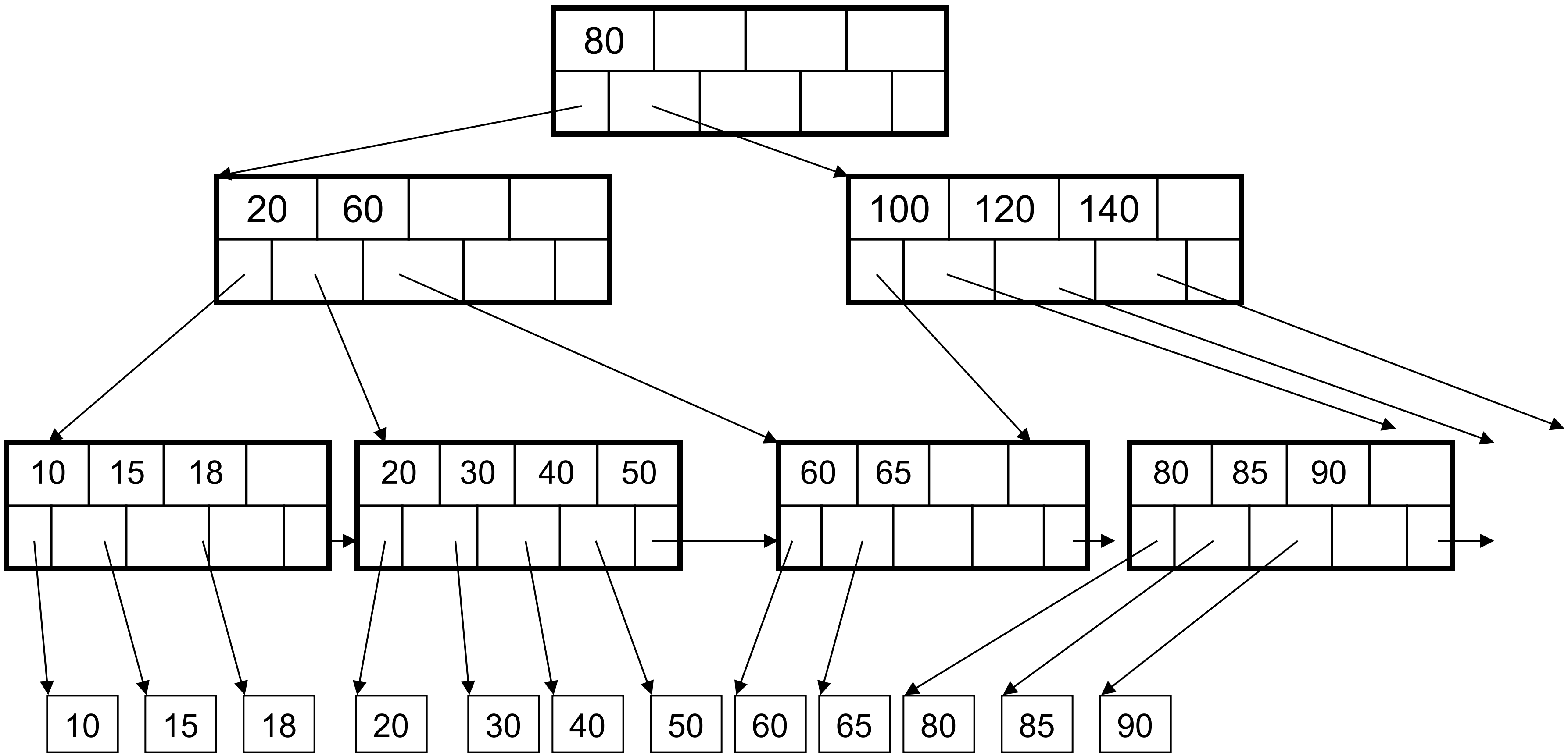
- Find leaf where K belongs, insert
- If no overflow (2d keys or less), halt
- If overflow (2d+1 keys), split node, insert in parent:



- If leaf, keep K3 too in right node
- When root splits, new root has 1 key only

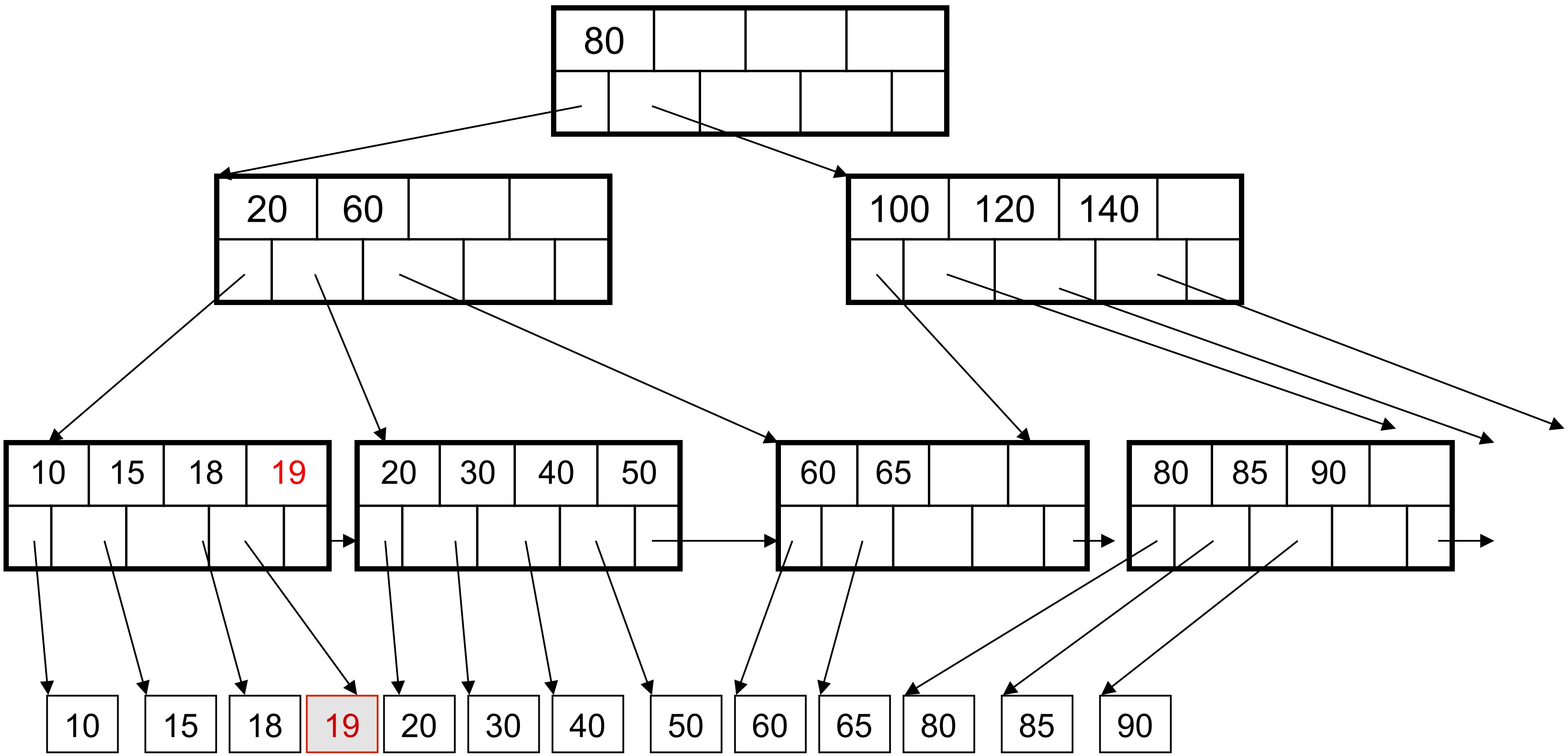
Insertion in a B+ Tree

Insert K=19



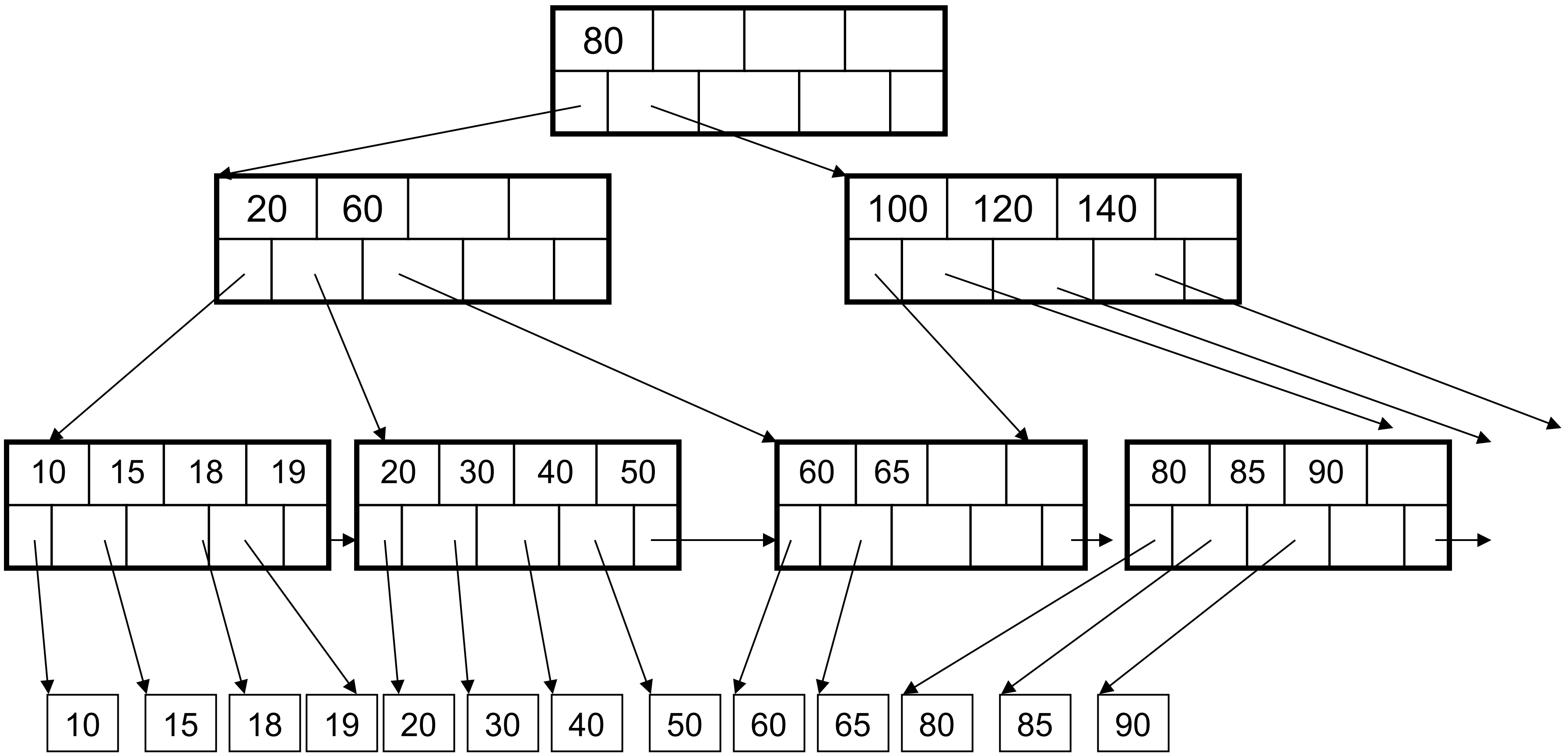
Insertion in a B+ Tree

After insertion



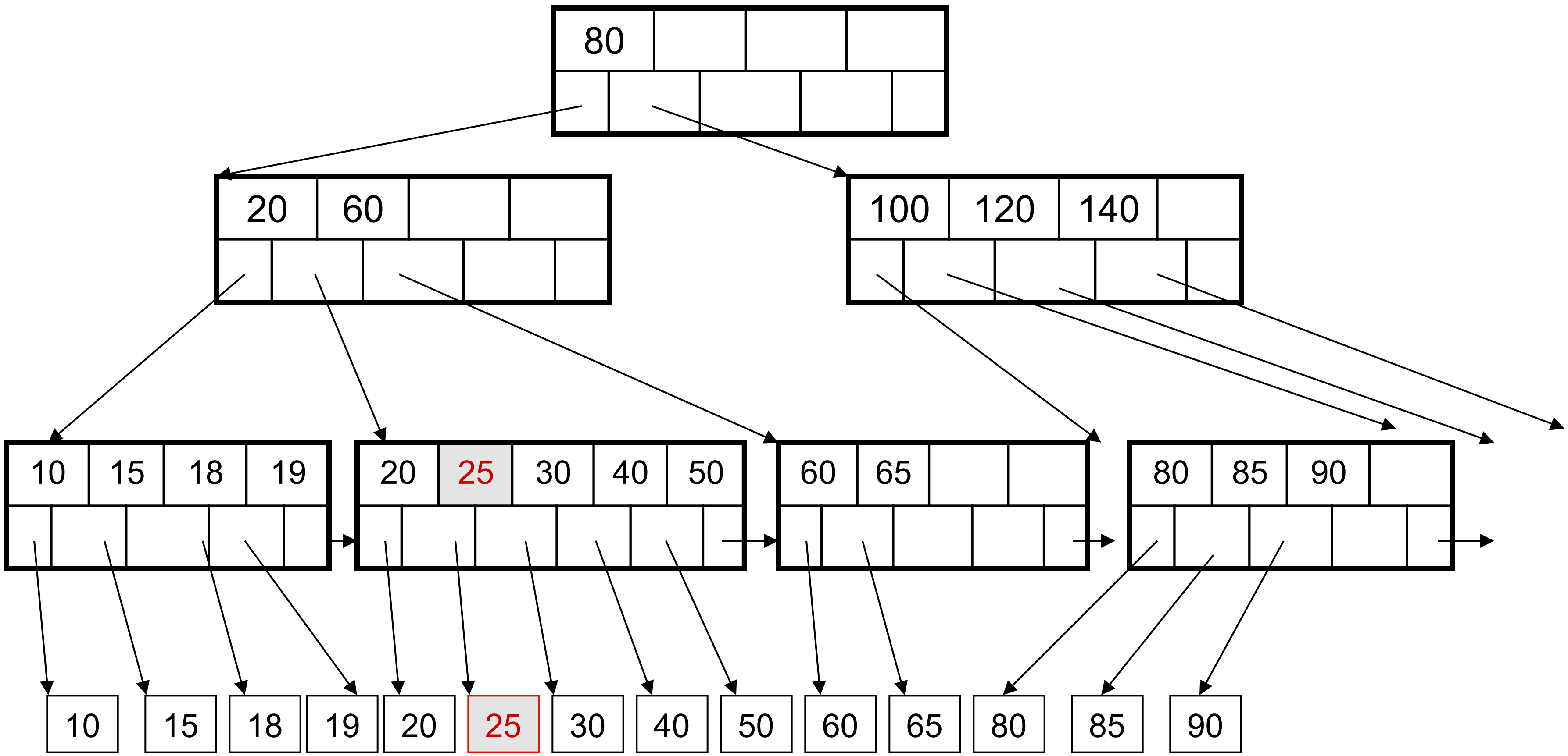
Insertion in a B+ Tree

Now insert 25



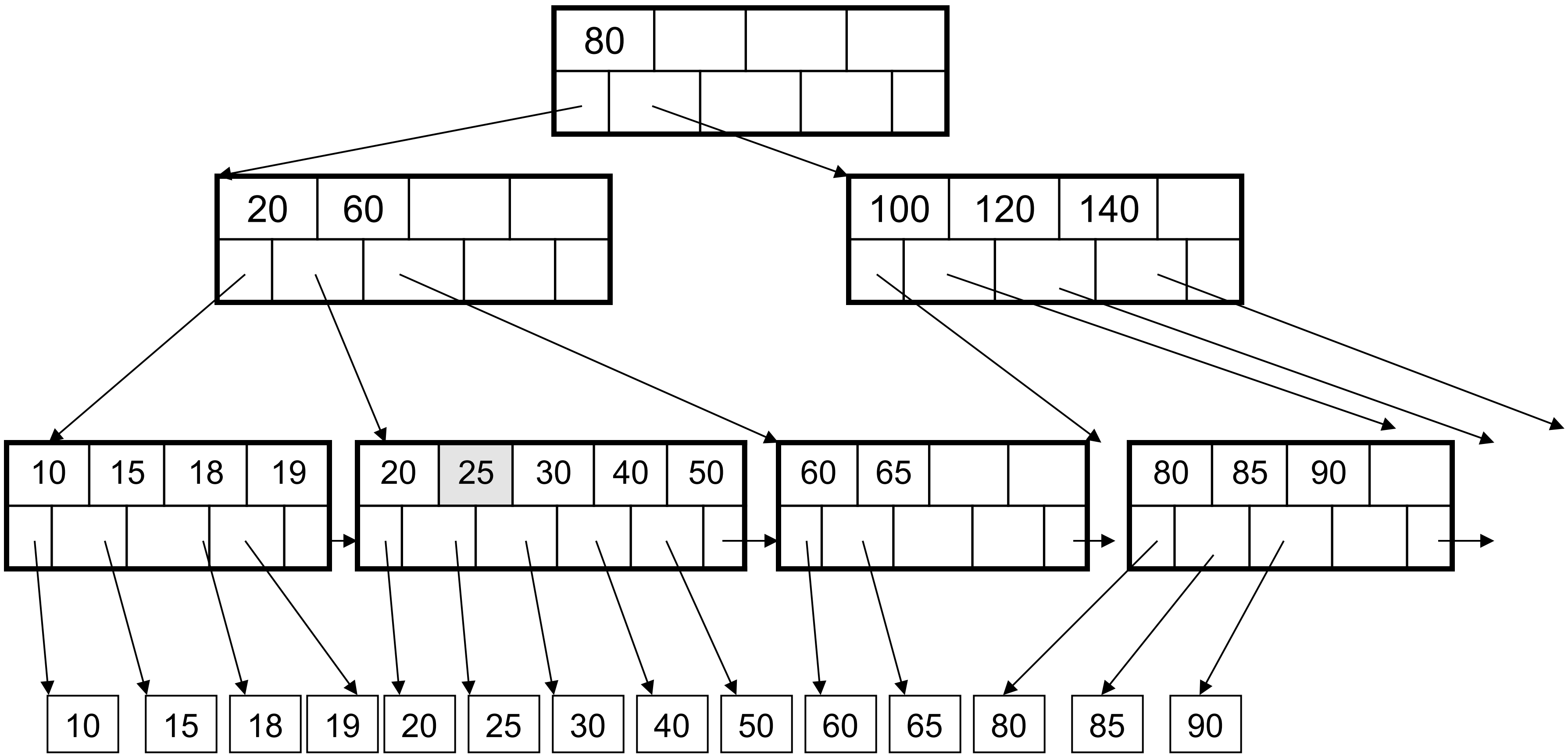
Insertion in a B+ Tree

After insertion



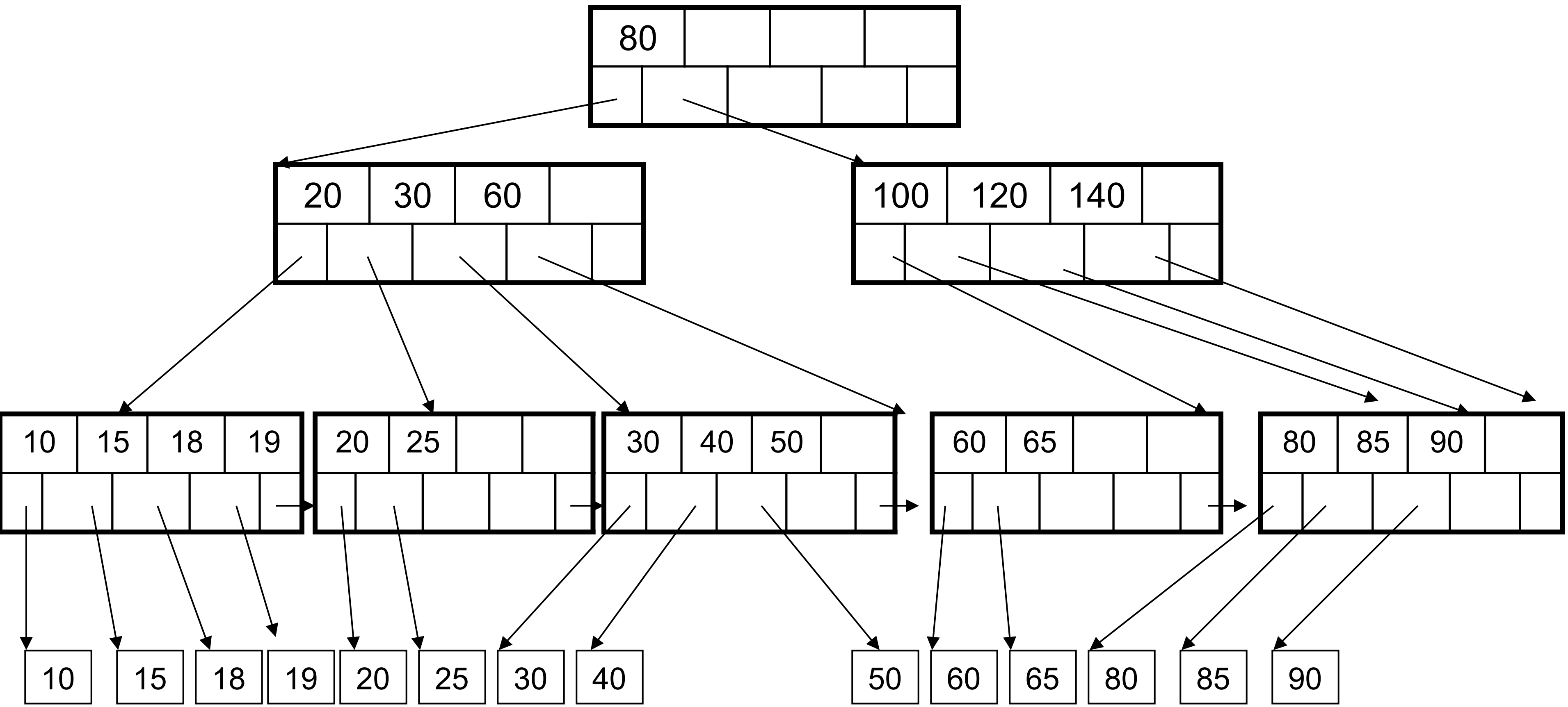
Insertion in a B+ Tree

But now have to split !



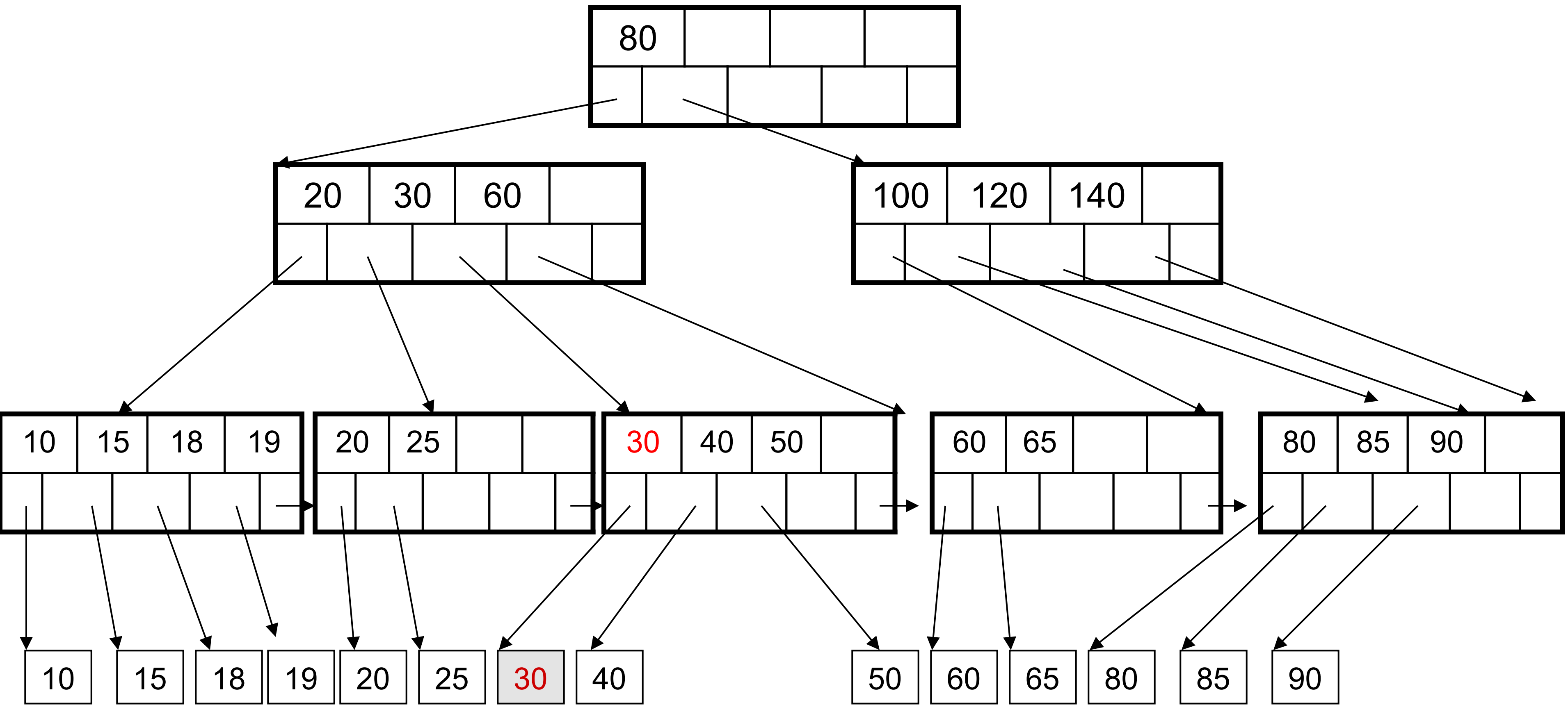
Insertion in a B+ Tree

After the split



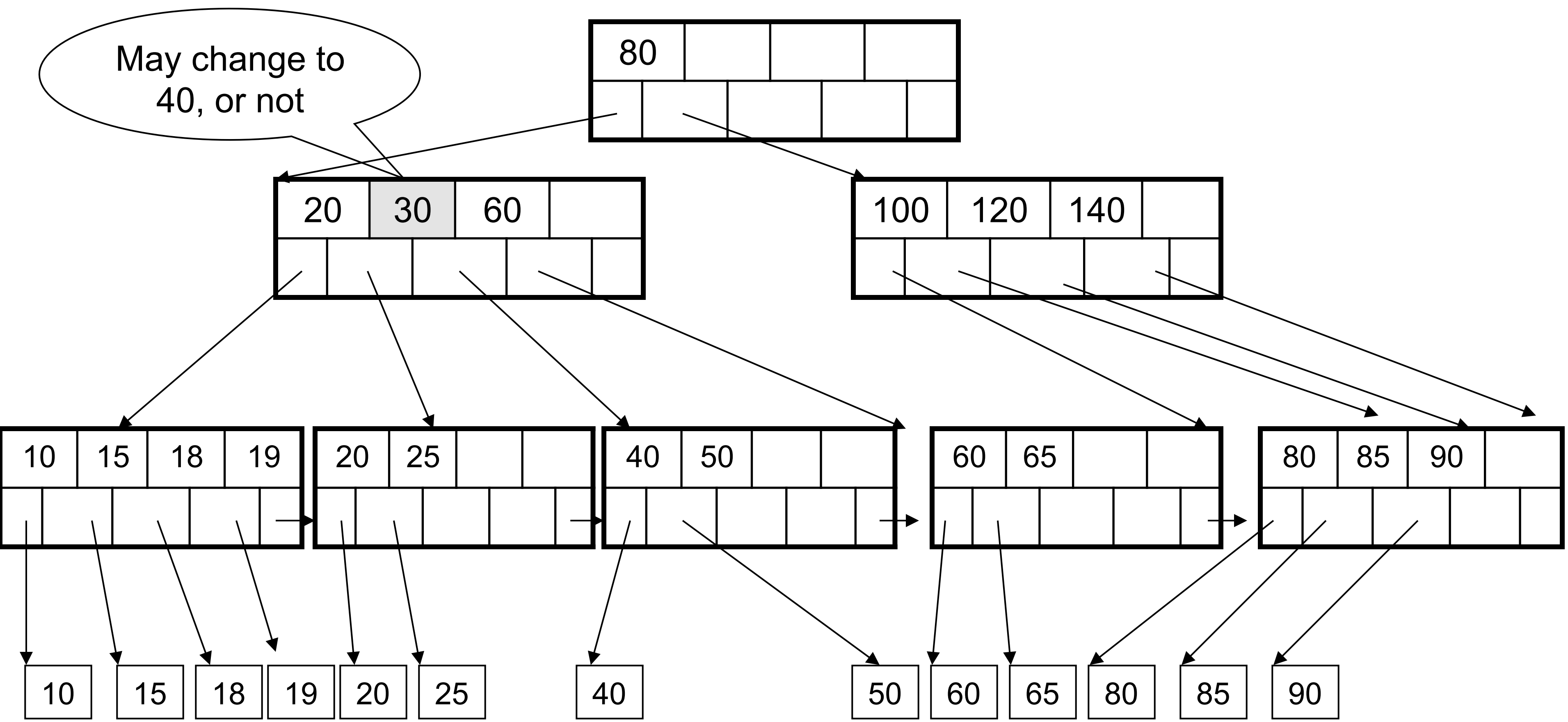
Deletion from a B+ Tree

Delete 30



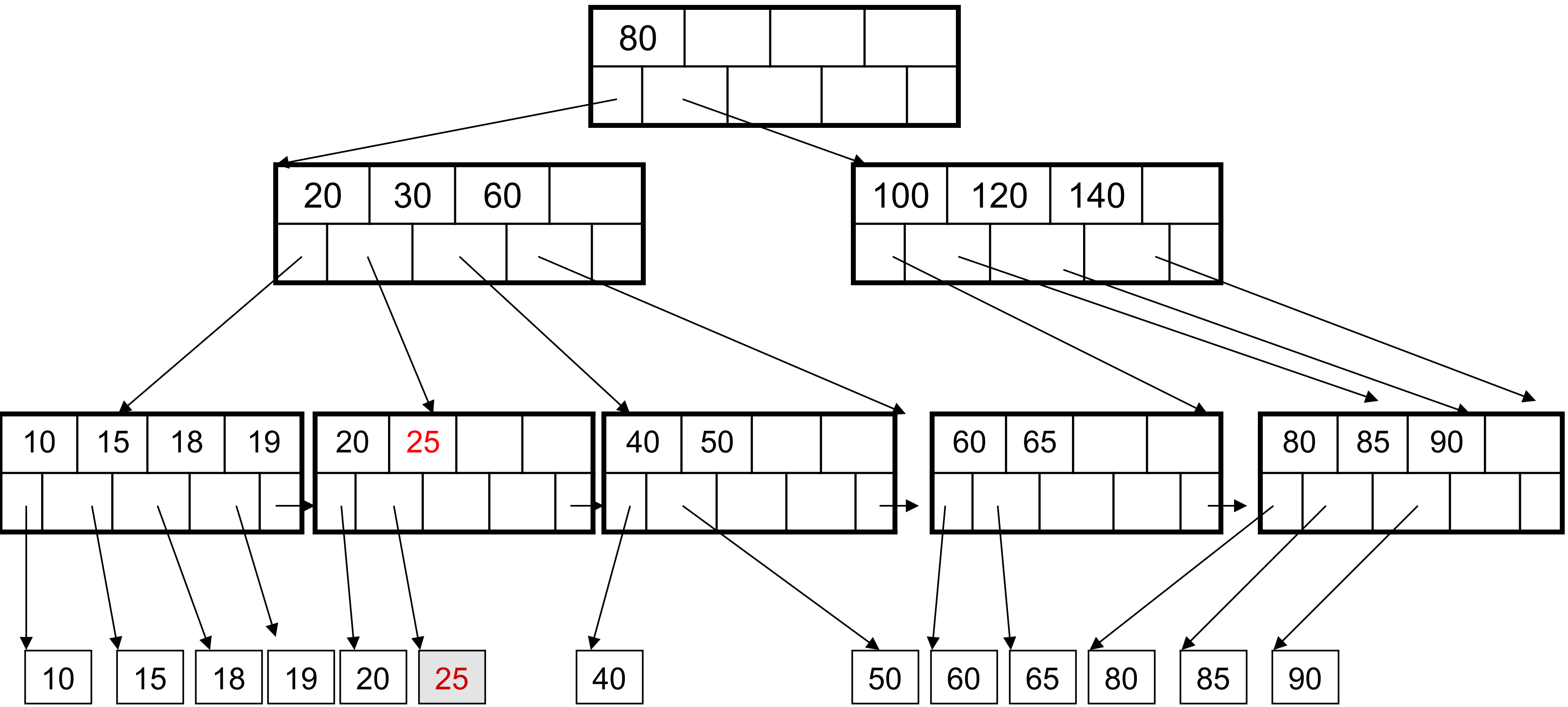
Deletion from a B+ Tree

After deleting 30



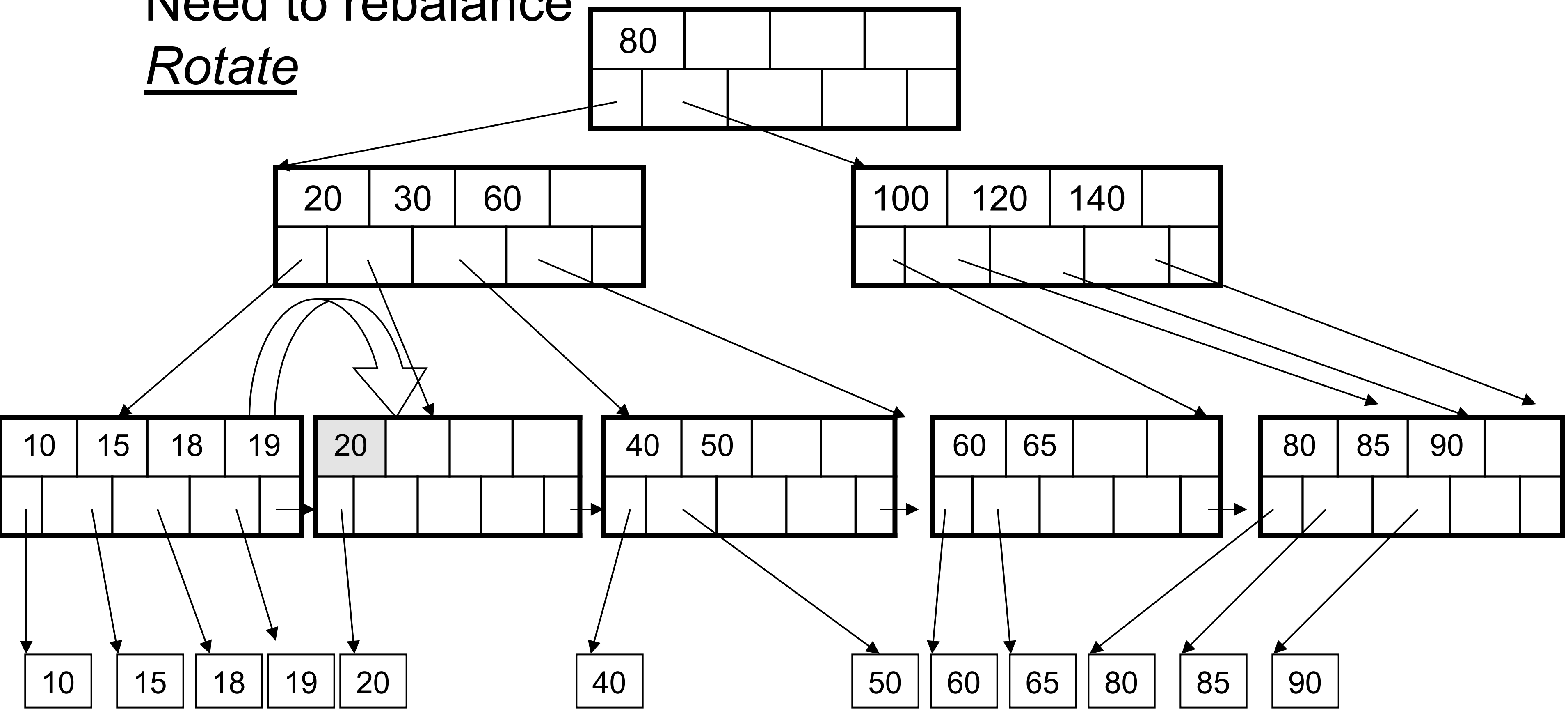
Deletion from a B+ Tree

Now delete 25



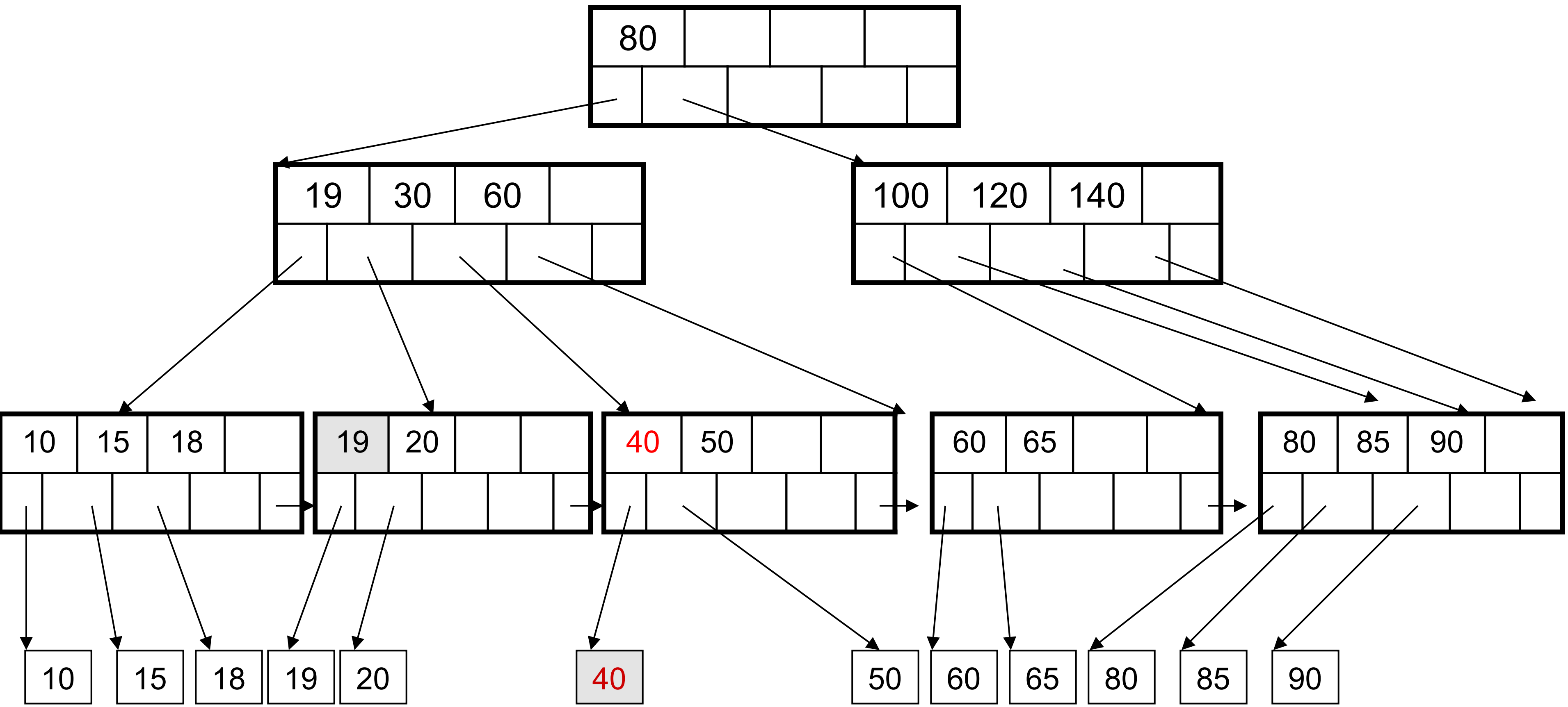
Deletion from a B+ Tree

After deleting 25
Need to rebalance
Rotate



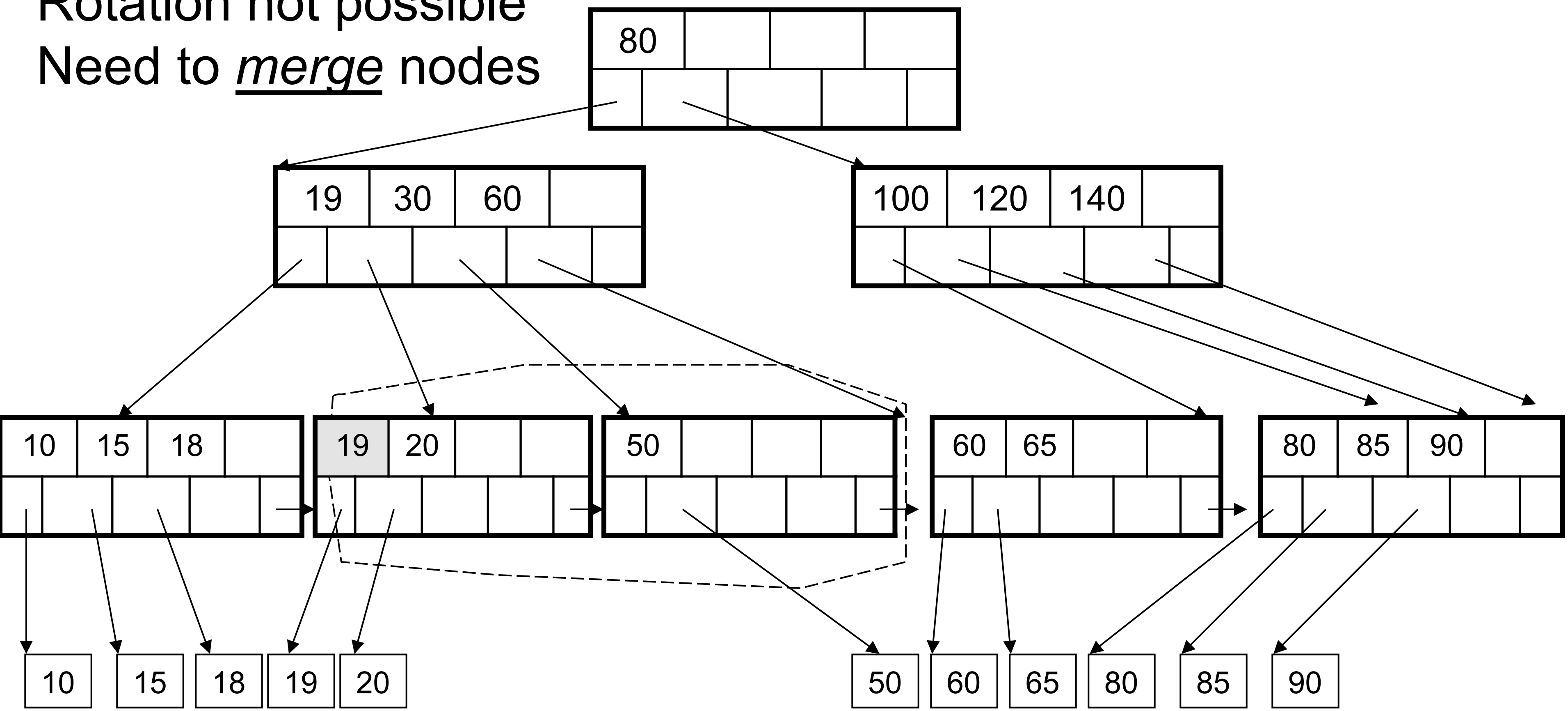
Deletion from a B+ Tree

Now delete 40



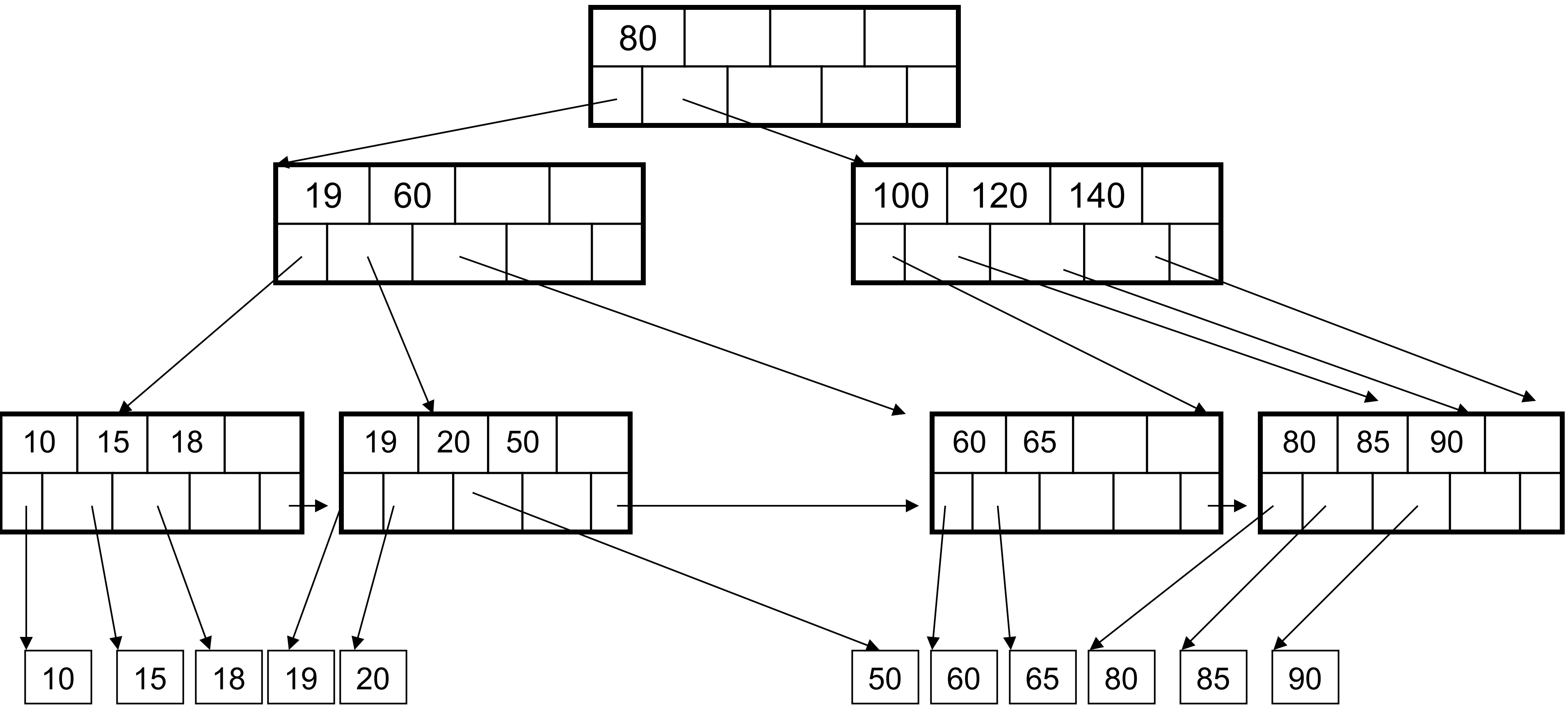
Deletion from a B+ Tree

After deleting 40
Rotation not possible
Need to *merge* nodes



Deletion from a B+ Tree

Final tree



B+ Tree Design

- How large d ?
- Example:
  - Key size = 4 bytes
  - Pointer size = 8 bytes
  - Block size = 4096 byes
- $2d \times 4 + (2d+1) \times 8 <= 4096$
- $d = 170$

B+ Trees in Practice

- Typical order: 100. Typical fill-factor: 67%
  - average fanout = 133
- Typical capacities
  - Height 4: 133^4 = 312,900,700 records
  - Height 3: 133^3 = 2,352,637 records
- Can often hold top levels in buffer pool
  - Level 1 = 1 page = 8 Kbytes
  - Level 2 = 133 pages = 1 Mbyte
  - Level 3 = 17,689 pages = 133 Mbytes

Practical Aspects of B+ Trees

Key compression:

- Each node keeps only the delta from parent keys
- Jonathan, John, Johnsen, Johnson … →
  - Parent: Jo
  - Child: nathan, hn, hnsen, hnson, …

Practical Aspects of B+ Trees

Bulk insertion: When a new index is created there are two options:

- Start from empty tree, insert each key one-by-one
- Do *bulk insertion* – what does that mean ?

Practical Aspects of B+ Trees

Concurrency control

- The root of the tree is a “hot spot”
  - Lock contention during insert/delete
- Solution: proactive split at insert, proactive merge at delete
  - Insert/delete now require only one traversal, from the root to a leaf
  - Use the “tree locking” protocol

Summary on B+ Trees

- Default index structure on most DBMS
- Very effective at answering ‘point’ queries:
  productName = ‘gizmo’
- Effective for range queries:
  50 < price AND price < 100
- Less effective for multirange:
  50 < price < 100 AND 2 < quant < 20

Indexes in Postgres

```
CREATE TABLE  V(M int,  N varchar(20),  P int);
```

```
CREATE INDEX V1_N ON V(N)
```

```
CREATE INDEX V2 ON V(P, M)
```

```
CREATE INDEX VVV ON V(M, N)
```

```
CLUSTER V USING V2
```

Makes V2 clustered

Index Selection Problem 1

```
V(M, N, P);
```

Your workload is this

100000 queries:

```
SELECT *
FROM V
WHERE N=?
```

100 queries:

```
SELECT *
FROM V
WHERE P=?
```

Which indexes should we create?

Index Selection Problem 1

V(M, N, P);

Your workload is this

100000 queries:

```
SELECT *
FROM V
WHERE N=?
```

100 queries:

```
SELECT *
FROM V
WHERE P=?
```

A: V(N) and V(P) (hash tables or B-trees)

Index Selection Problem 2

```
V(M, N, P);
```

Your workload is this

100000 queries:

```sql
SELECT *
FROM V
WHERE N>? and N<?
```

100 queries:

```sql
SELECT *
FROM V
WHERE P=?
```

100000 queries:

```sql
INSERT INTO V
VALUES (?, ?, ?)
```

Which indexes should we create?

Index Selection Problem 2

V(M, N, P);

Your workload is this

100000 queries:

```
SELECT *
FROM V
WHERE N>? and N<?
```

100 queries:

```
SELECT *
FROM V
WHERE P=?
```

100000 queries:

```
INSERT INTO V
VALUES (?, ?, ?)
```

A: definitely V(N) (must B-tree); unsure about V(P)

Index Selection Problem 3

```
V(M, N, P);
```

Your workload is this

100000 queries:

```
SELECT *
FROM V
WHERE N=?
```

1000000 queries:

```
SELECT *
FROM V
WHERE N=? and P>?
```

100000 queries:

```
INSERT INTO V
VALUES (?, ?, ?)
```

Which indexes should we create?

Index Selection Problem 3

V(M, N, P);

Your workload is this

100000 queries:

```
SELECT *
FROM V
WHERE N=?
```

1000000 queries:

```
SELECT *
FROM V
WHERE N=? and P>?
```

100000 queries:

```
INSERT INTO V
VALUES (?, ?, ?)
```

A: V(N, P)

Index Selection Problem 4

V(M, N, P);

Your workload is this

1000 queries:

```
SELECT *
FROM V
WHERE N>? and N<?
```

100000 queries:

```
SELECT *
FROM V
WHERE P>? and P<?
```

Which indexes should we create?

Index Selection Problem 4

```
V(M, N, P);
```

Your workload is this

1000 queries:

```
SELECT *
FROM V
WHERE N>? and N<?
```

100000 queries:

```
SELECT *
FROM V
WHERE P>? and P<?
```

A: V(N) secondary, V(P) primary index

To Cluster or Not

- Range queries benefit mostly from clustering
- Covering indexes do *not* need to be clustered: they work equally well unclustered

SELECT *
FROM R
WHERE K>? and K<?