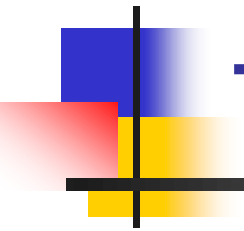


データベース (第10回)

情報工学科　木村昌臣

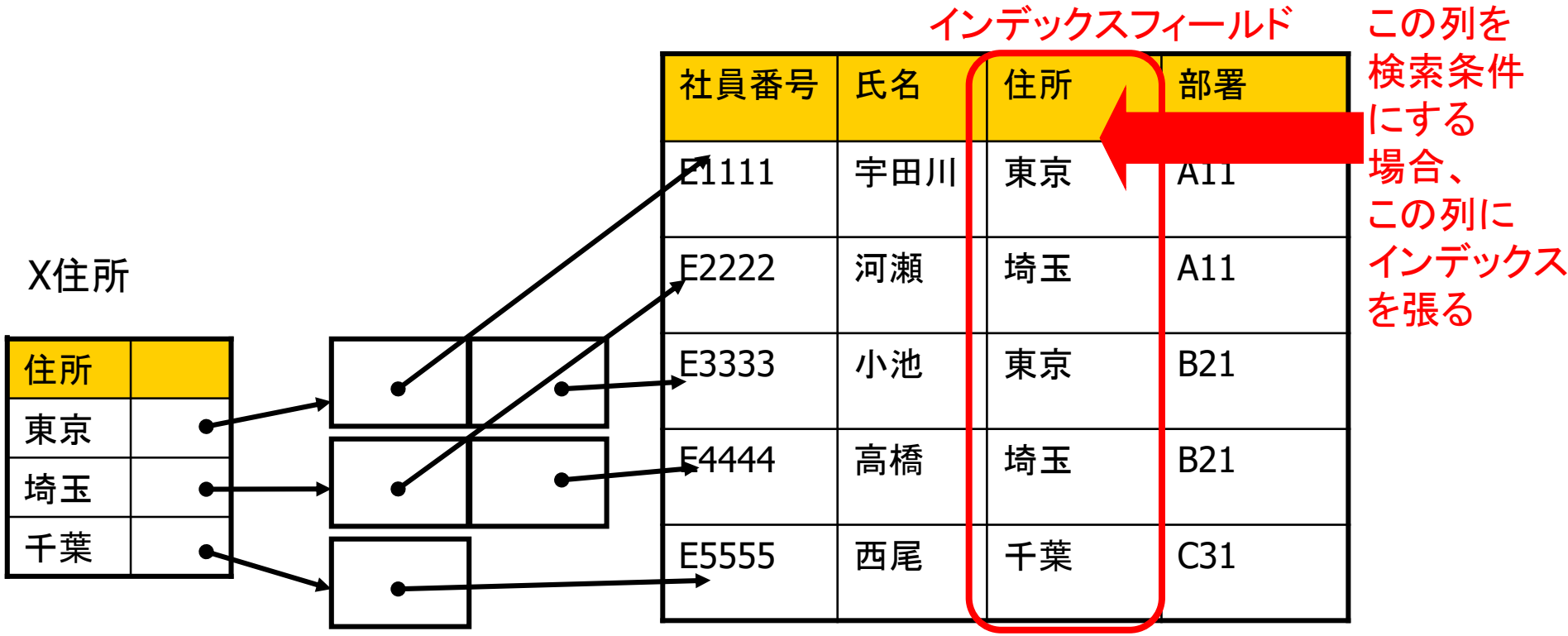
【復習】ファイルアクセス方式

- ファイルの中にあるレコードを探し出す方式
  - 一つのテーブルは、ひとつのファイルとして実現されることが多いことに注意
- 代表的な例は
  - スキャン（線型探索）
  - 探索
  - インデックス法
  - ハッシュ法

先回はここの話をした！

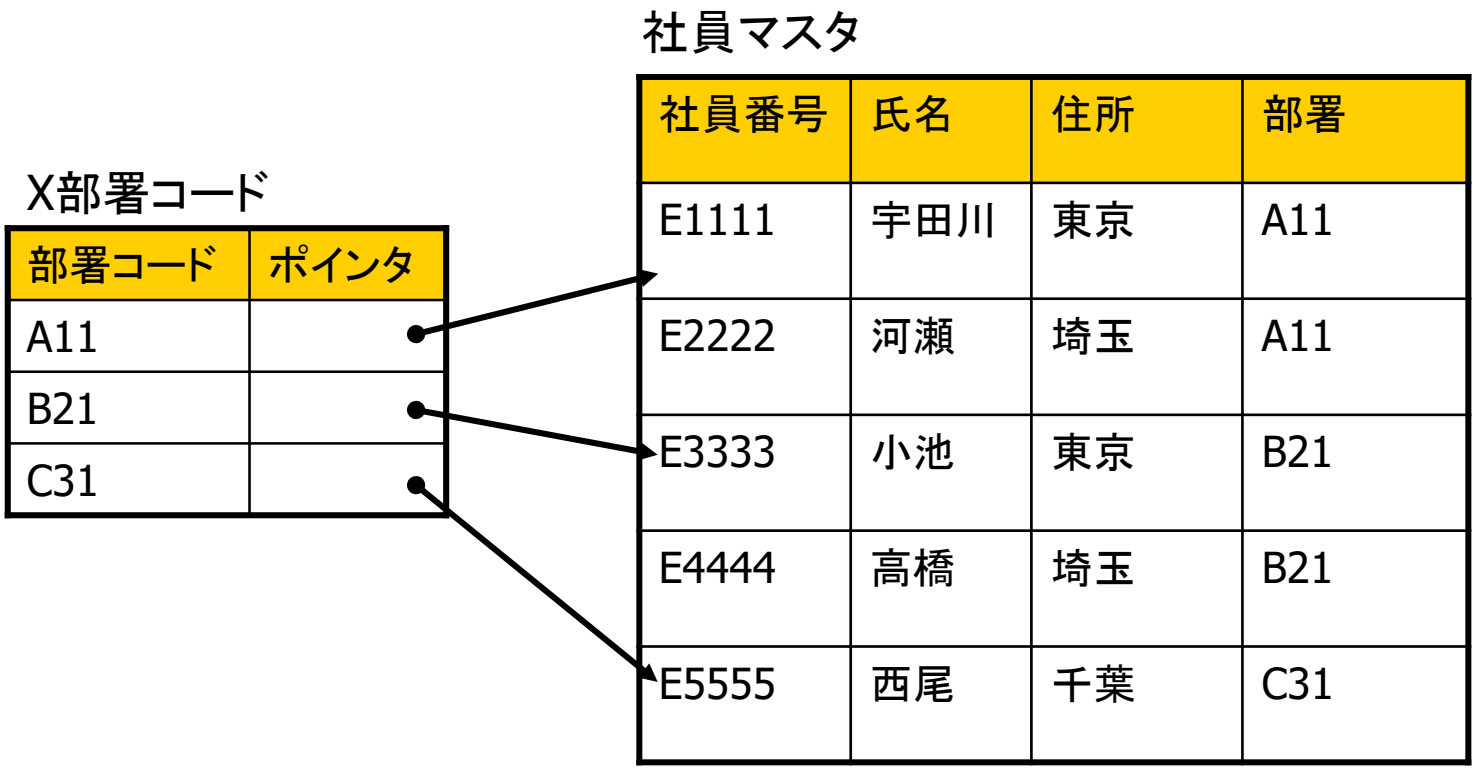
【復習】インデックス

テーブルのある列の値および
その値をもつレコードへのポインタ
の対からなるレコード群



クラスタリングインデックス

順序フィールドだが候補キーでない場合、その列の上で定義されたインデックスを特に「クラスタリングインデックス」と呼ぶ



多段インデックス

- インデックスフィールドの値の種類が増えると、インデックスの「レコード数」が増えてしまう
  - 結果的にスキャンと変わらなくなってしまう
  - インデックスの参照量を減らしたい

多段インデックス

- ISAMインデックス

非平衡木であり、更新が速い
（＝検索速度がインデックスなしと同等になってしまう場合もある）

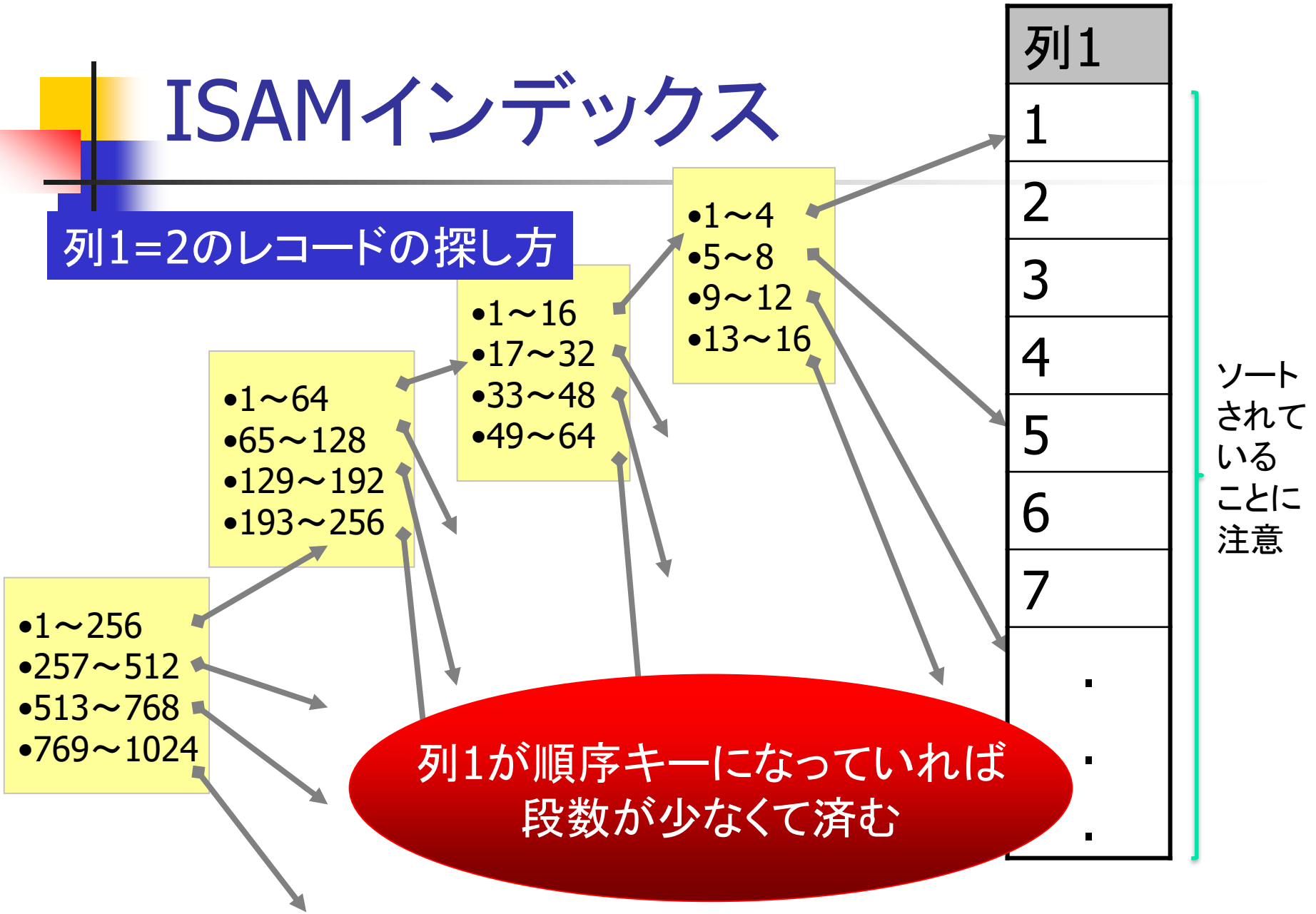
インデックスフィールドが順序キーになっていることが前提

インデックスフィールドの値によって、レコードがソートされており且つ一意に決まる。

- B+木インデックス

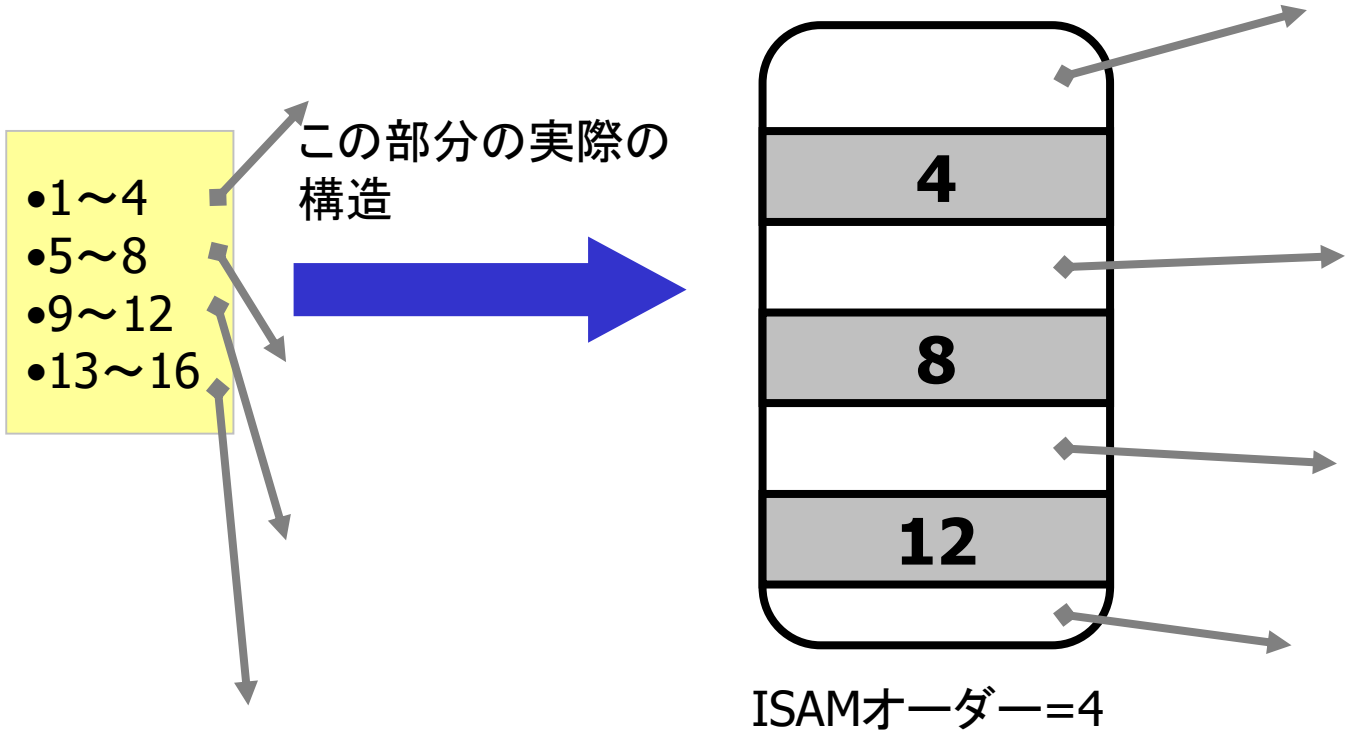
平衡木をつくり、常に検索が速い（更新が頻繁にあると遅い）

ISAMインデックス


列1=2のレコードの探し方
•1～256
•257～512
•513～768
•769～1024
•1～64
•65～128
•129～192
•193～256
•1～16
•17～32
•33～48
•49～64
•1～4
•5～8
•9～12
•13～16
列1
1
2
3
4
5
6
7
.
.
.
ソートされていることに注意
列1が順序キーになっていれば
段数が少なくて済む

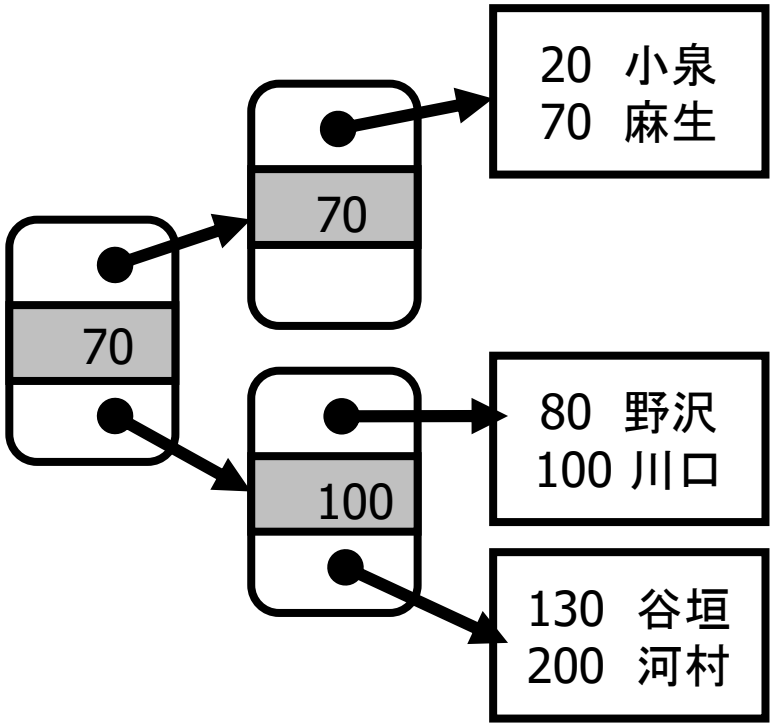
ISAMインデックス

- インデックス付き順序アクセス法
  - Indexed Sequential Access Method



ISAMインデックス

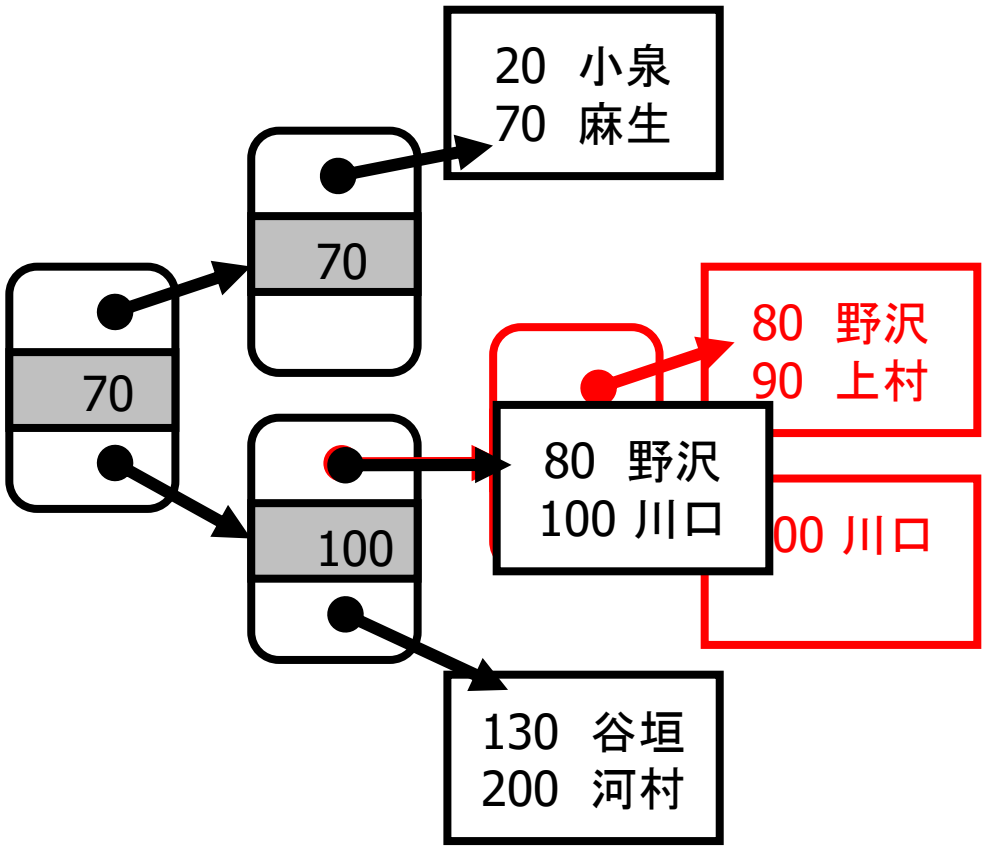
右のテーブルの、ISAMオーダーが2のISAMインデックスを考える。
また、1ページ(ブロック)には2レコード入るとする。



社員番号	名前
20	小泉
70	麻生
80	野沢
100	川口
130	谷垣
200	河村

ISAMインデックス

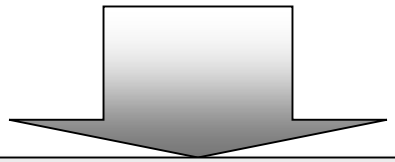
社員番号 90 名前 上村 という社員のレコードが追加されると・・・・



社員番号	名前
20	小泉
70	麻生
80	野沢
90	上村
100	川口
130	谷垣
200	河村

ISAMインデックスの問題点

- 動的再配置されない
  - ある葉にレコードがたくさん追加されても別の葉に既存のレコードを移し変えることはない



一部のノードの下に
レコードが集中してしまう
非平衡木

例えば、社員番号70以下のレコードが増えると小泉・麻生がいたノードにレコードが集中してしまう

B+木

pをB+木の探索キー 社員番号のオーダーとする。(右の図ではp=3)

中間ノード:

■他の中間ノードもしくは葉ノードへのポインタ(木ポインタ)をもつ

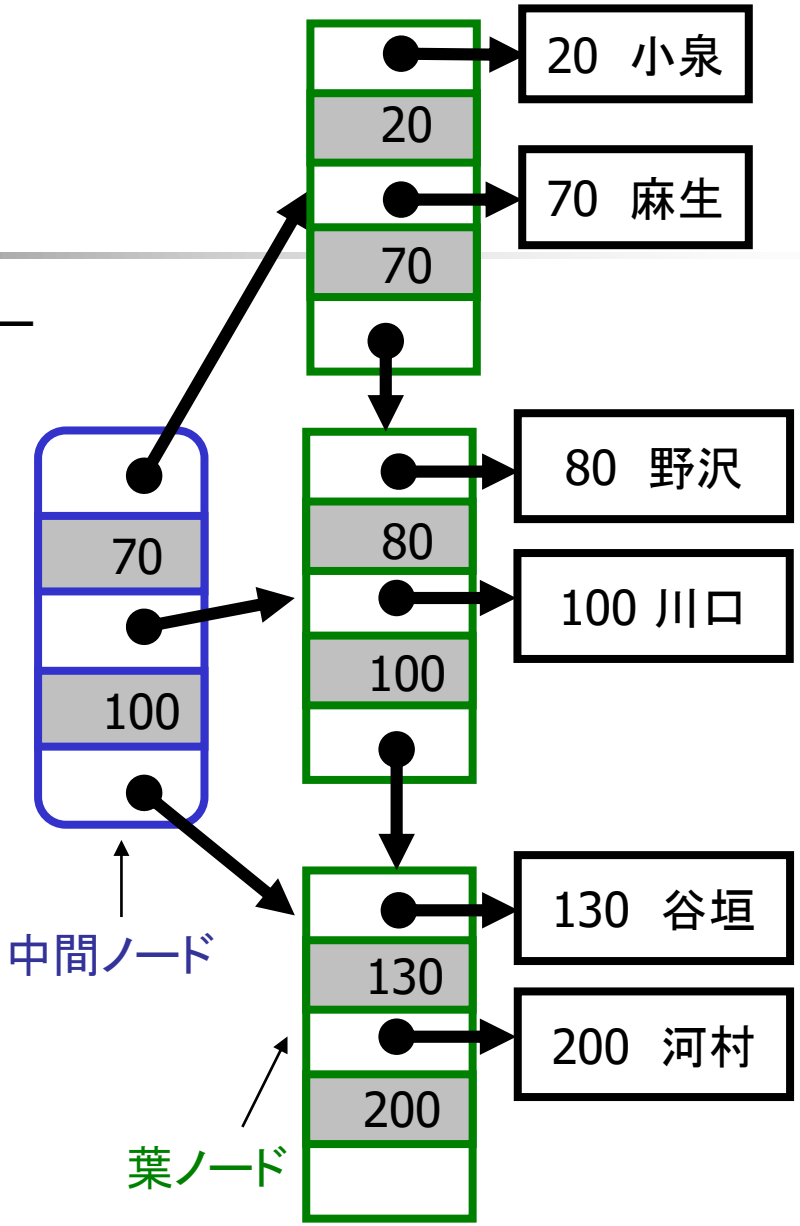
■p/2～p個の木ポインタを持たなければならない

葉ノード:

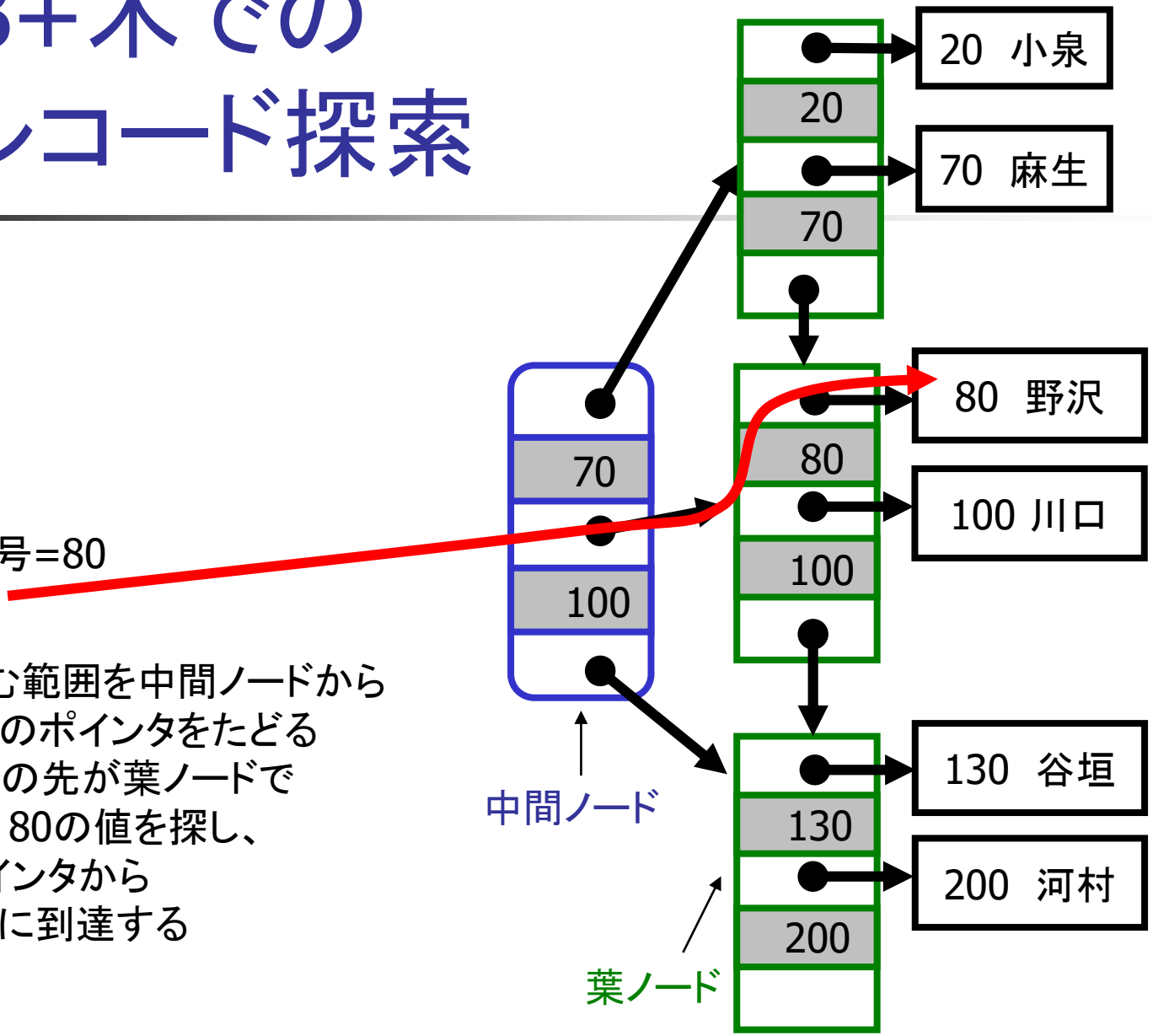
■データへのポインタ(データポインタ)は葉ノードのみ持つ

■最後のポインタは隣の葉ノードをさす

■(p-1)/2～p-1個のデータポインタを持たなければならない



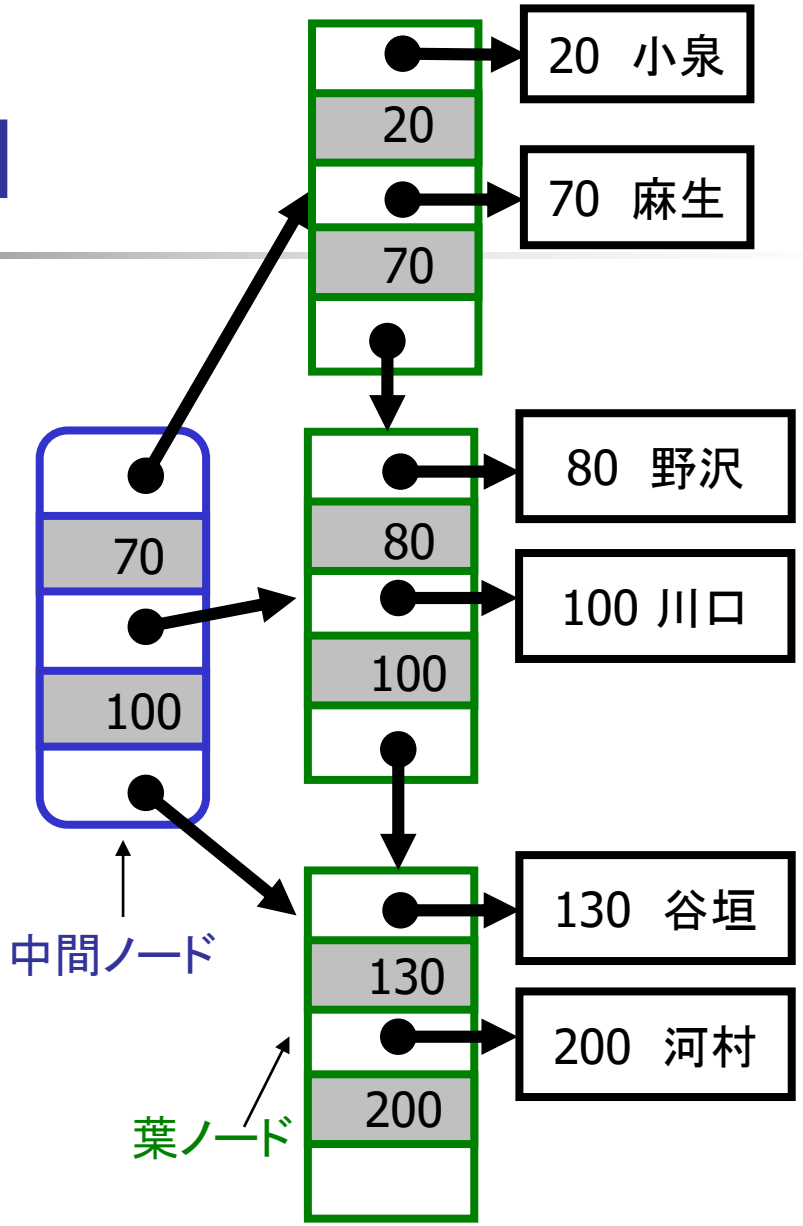
B+木での レコード探索



1. 80を含む範囲を中間ノードから探し、そのポインタをたどる
2. ポインタの先が葉ノードであれば、80の値を探し、そのポインタからレコードに到達する

B+木での レコード追加

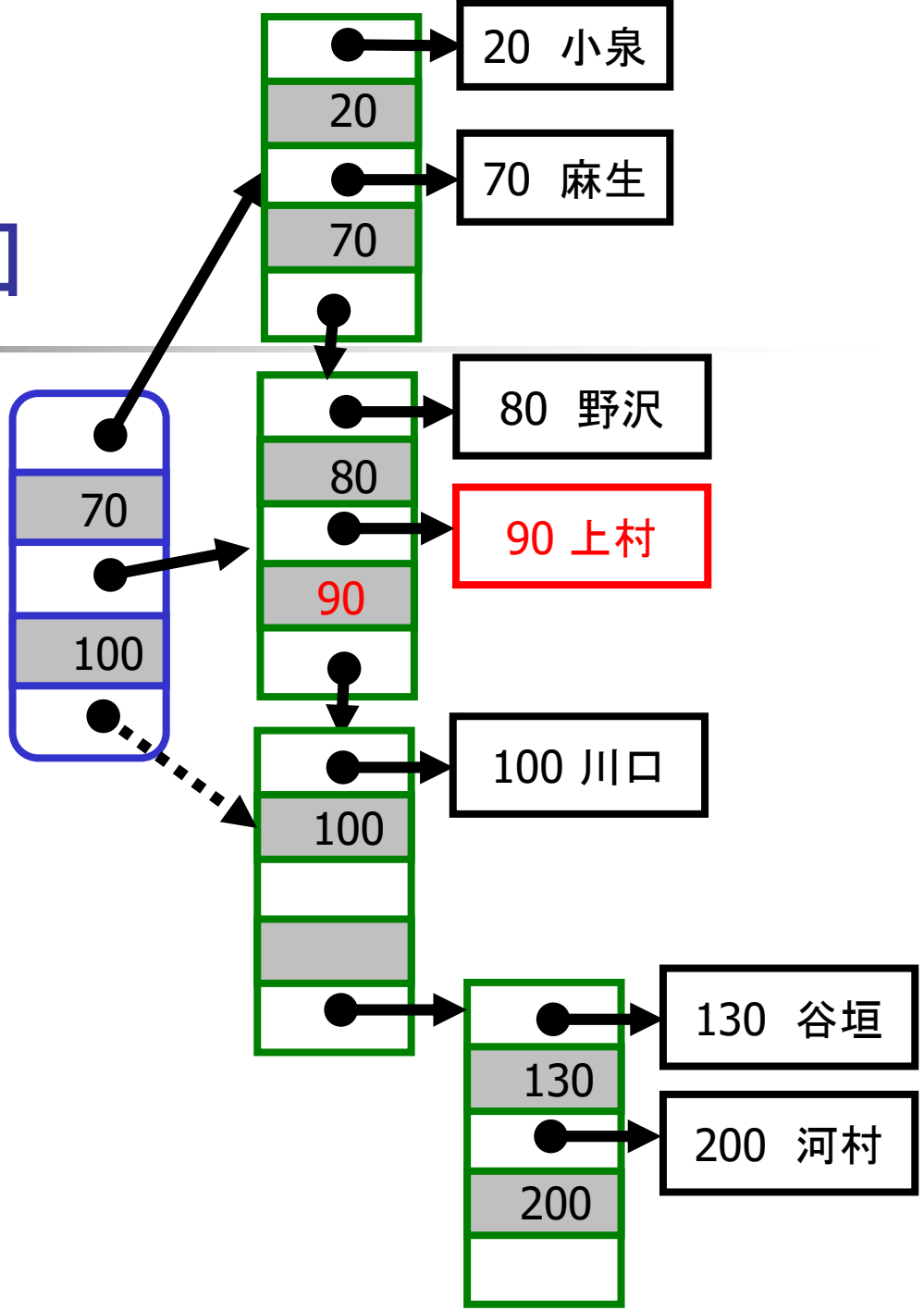
社員番号 90 名前 上村 という社員のレコードが追加されると・・・・



- ■空きがないとき葉ノードを分ける。
- ■親ノードのポインタの空きが足りなくなったら、さらにその親ノードを追加する。
- ■子供の部分木の最大値をキーの値として振る。

B+木での レコード追加

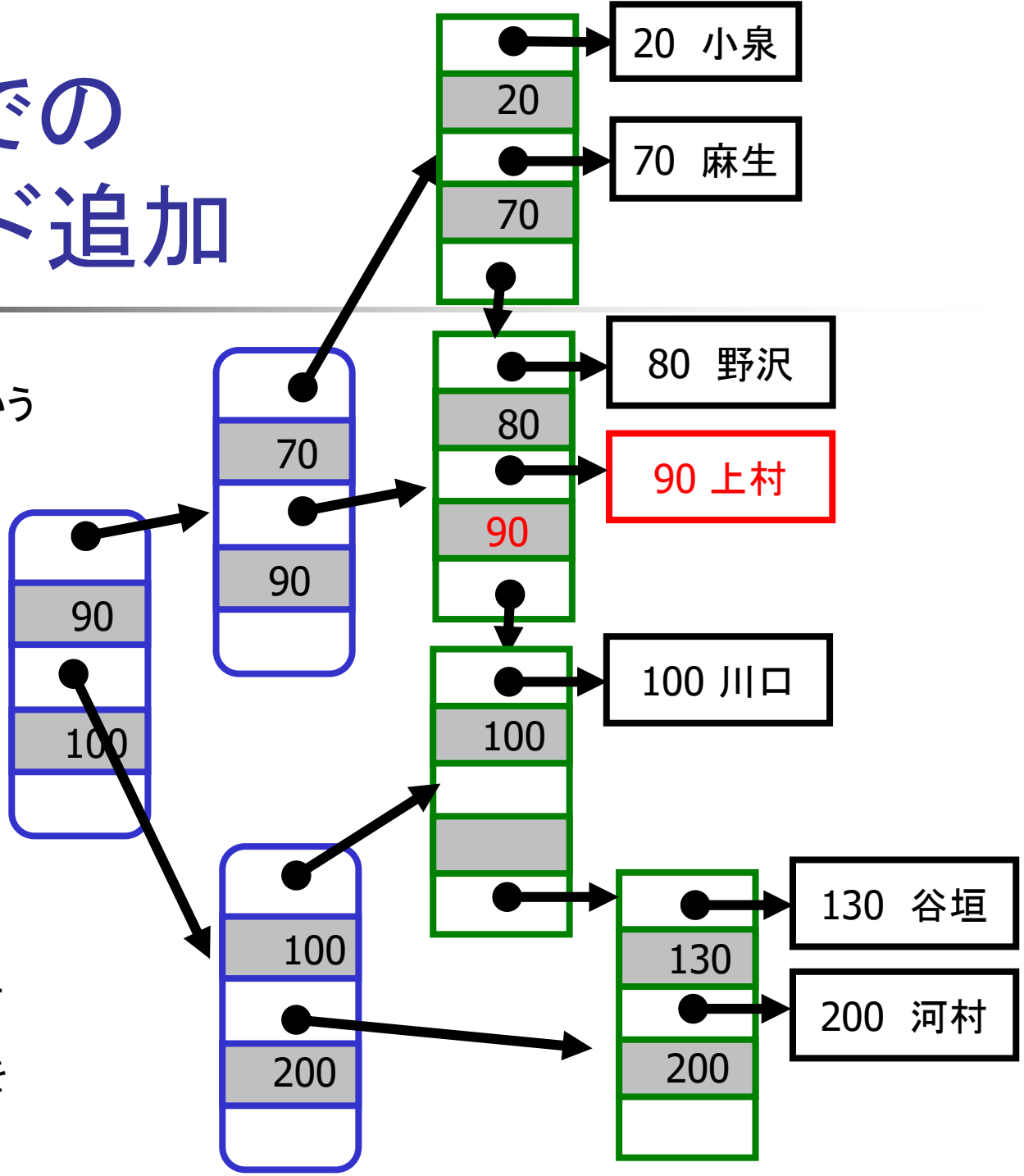
社員番号 90 名前 上村 という社員のレコードが追加されると・・・・



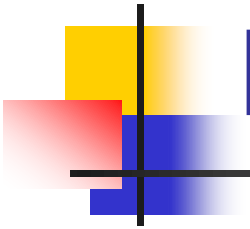
■空きがないとき葉ノードを分ける。
■親ノードのポインタの空きが足りなくなったら、さらにその親ノードを追加する。
■子供の部分木の最大値をキーの値として振る。

B+木での レコード追加

社員番号 90 名前 上村 という社員のレコードが追加されると・・・・



- ■空きがないとき葉ノードを分ける。
- ■親ノードのポインタの空きが足りなくなったら、さらにその親ノードを追加する。
- ■子供の部分木の最大値をキーの値として振る。



B+木とISAM

- 両方とも多段インデックス
- ISAMは非平衡木だが、B+木は平衡木
- インデックスフィールド
  - ISAMは順序キーでなくてはならない
  - B+木は探索キーでよい

ハッシュ法

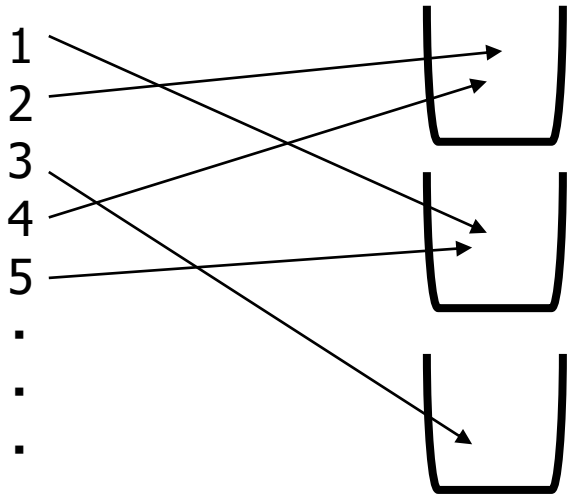
- ハッシュ関数を使って、探索キーの値から直接、レコードのありかを特定する方法
  - 当然、ファイル（=テーブルの物理的実体）はこれに見合うレコードの持ち方をしていなければならない。（=ファイル編成法）
- ハッシュ法を使うときの探索キーをハッシュキーという。
- 静的ハッシュ法と動的ハッシュ法がある。（後述）

ハッシュ関数

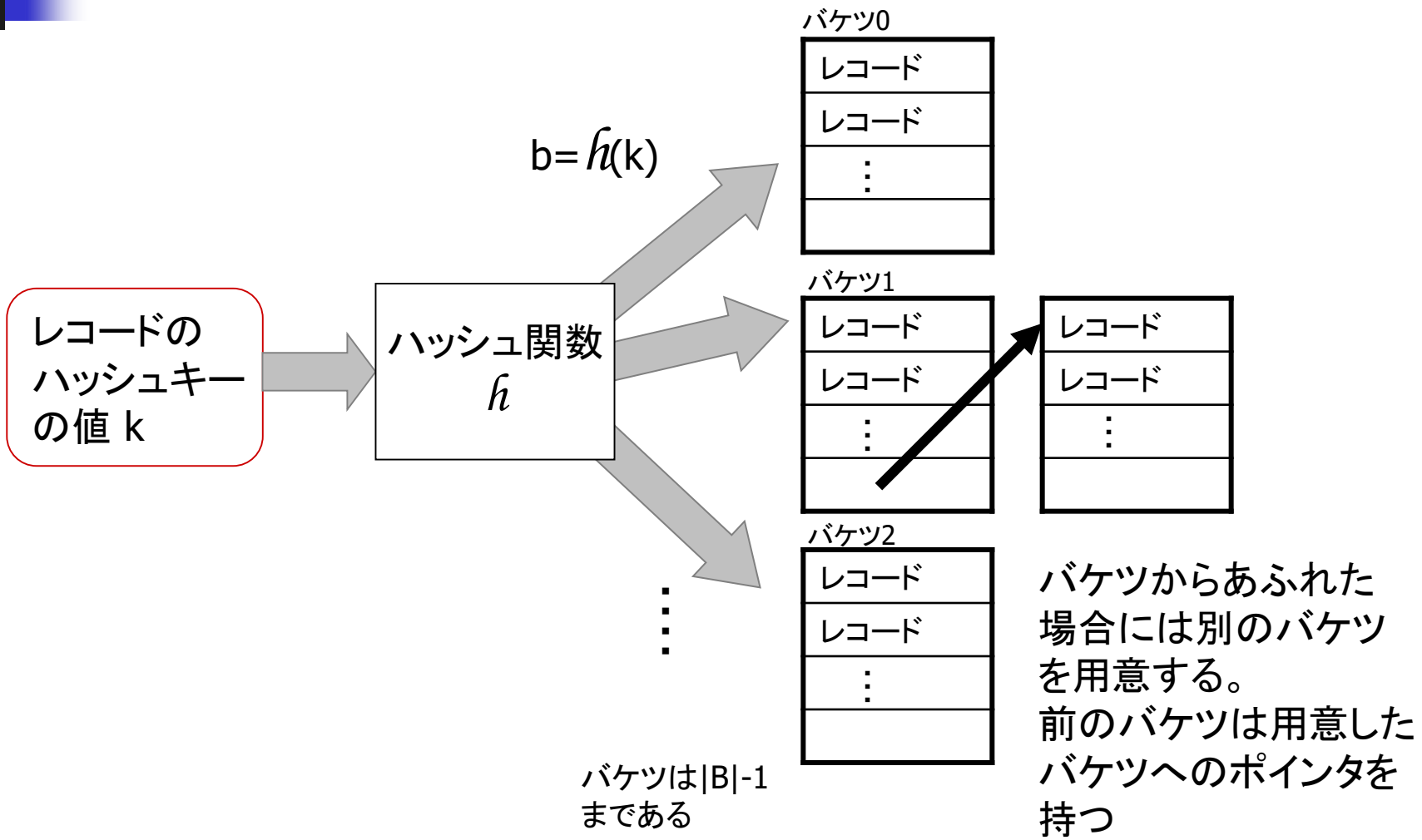
- レコードを格納する器をバケツと呼ぶ
- ハッシュキー（探索キー）の値の集合Kから、バケツの集合Bへの写像は

$$h : K \rightarrow B$$

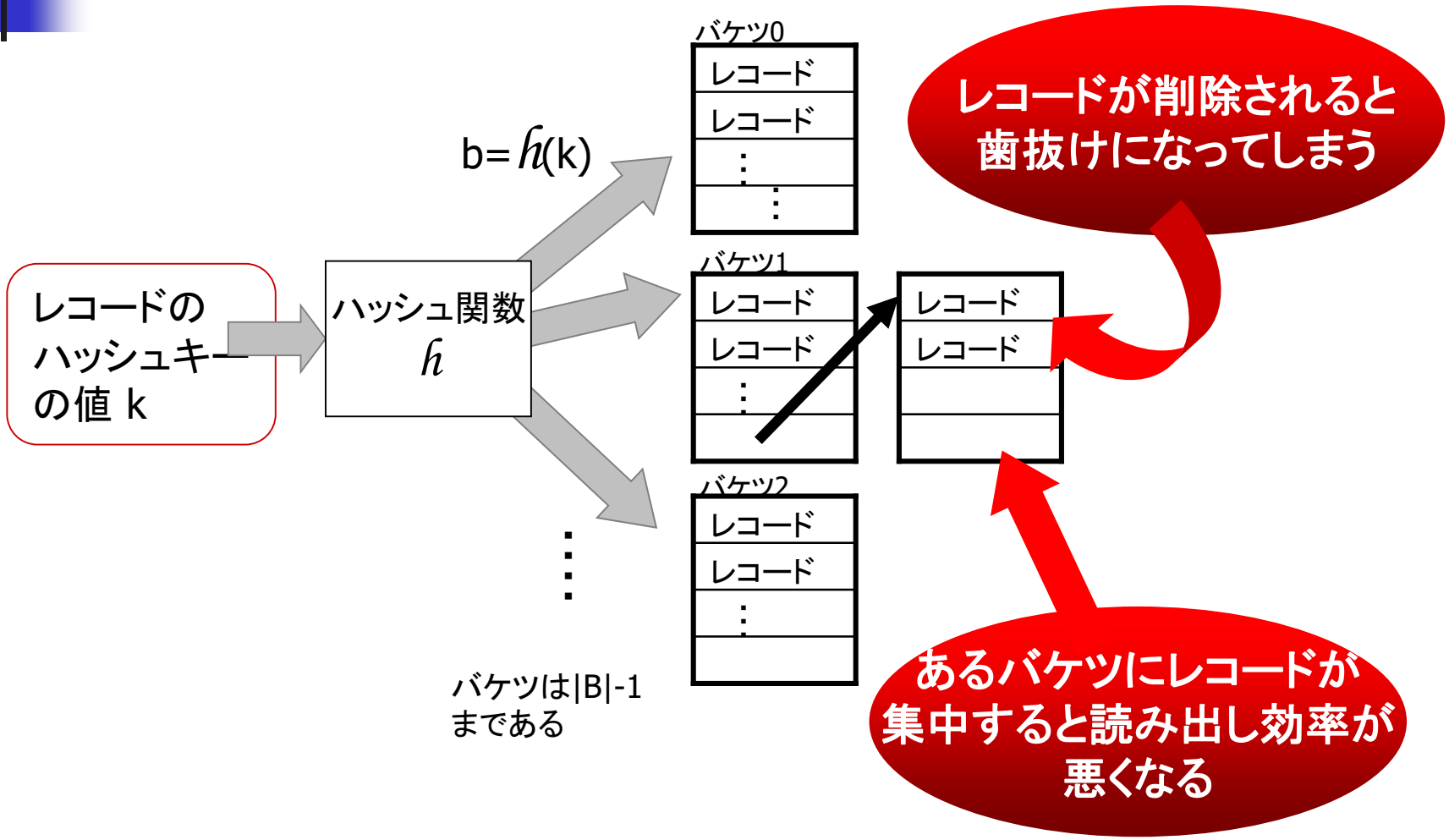
- このような写像のなかで次の条件を満たすものをハッシュ関数と呼ぶ
  - 個々のバケツへ写像されるハッシュキーの数が一定（均一性）
  - ハッシュキーの二つの値が近くても対応するバケツはバラバラ（ランダム性）

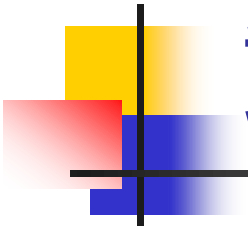


静的ハッシュ法


バケツ0
レコード
レコード
b=h(k)
レコードの
ハッシュキー
の値 k
ハッシュ関数
h
バケツ1
レコード
レコード
レコード
レコード
バケツ2
レコード
レコード
バケツからあふれた
場合には別のバケツ
を用意する。
前のバケツは用意した
バケツへのポインタを
持つ
バケツは|B|-1
まである

静的ハッシュ法の欠点

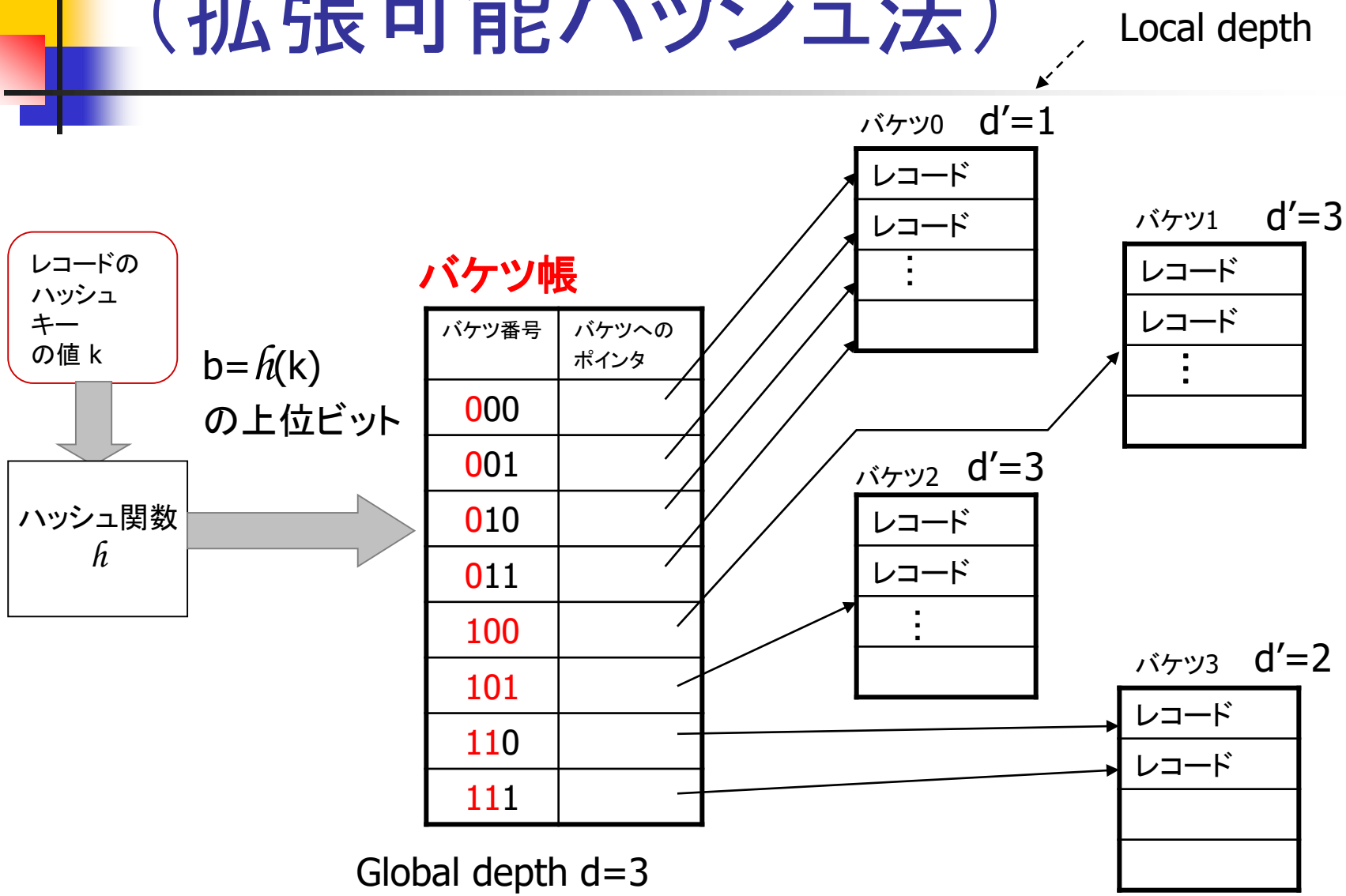

レコードの
ハッシュキー
の値 k
ハッシュ関数
h
b= h(k)
バケツ0
レコード
レコード
バケツ1
レコード
レコード
レコード
レコード
バケツ2
レコード
レコード
バケツは|B|-1
まである
レコードが削除されると
歯抜けになってしまう
あるバケツにレコードが
集中すると読み出し効率が
悪くなる



動的ハッシュ法

- ハッシュ関数を動的に変化させる手法
  - 線型ハッシュ法
  - **拡張可能ハッシュ法**　←広く使われている

動的ハッシュ法（拡張可能ハッシュ法）



拡張可能ハッシュ法でのレコード追加