データベースシステム

データの格納方式

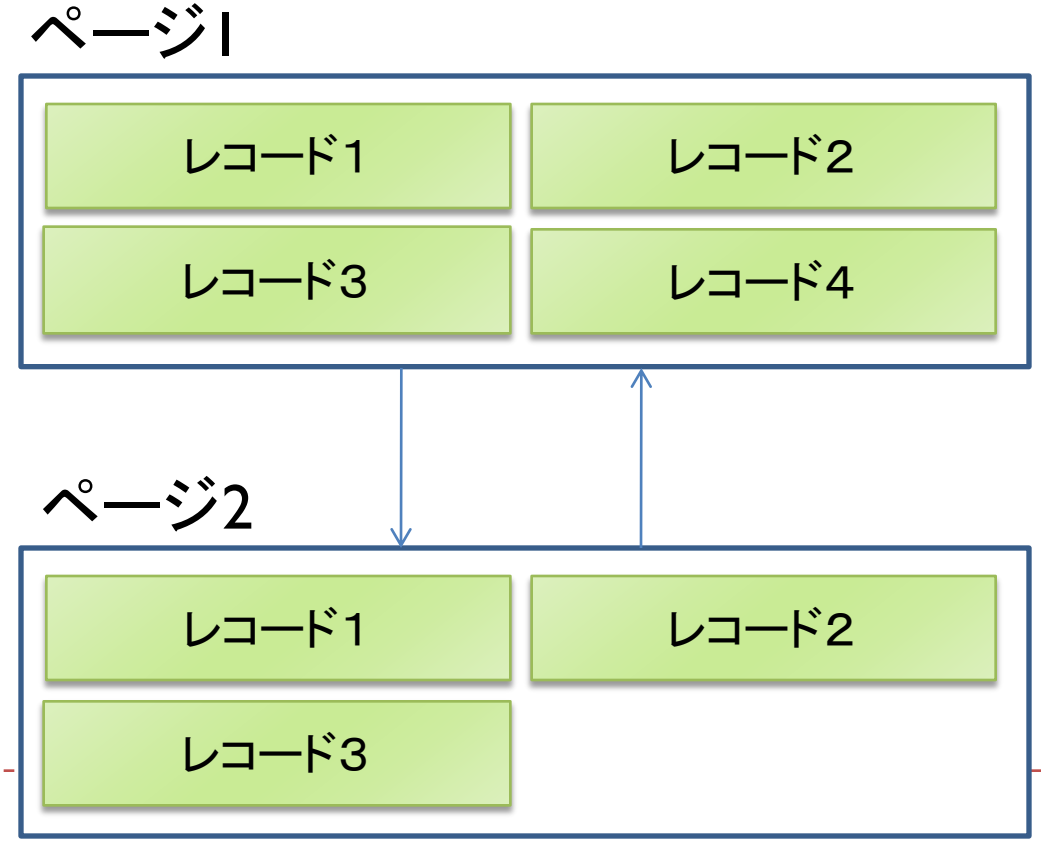
内容

1. 高速な検索を実現するために必要な技術
2. 検索時間を考えるための重要な用語
   「IOコスト」
3. もっとも簡単なデータ格納方式
   「ヒープファイル」
4. 問合せの種類別，IOコストの計算の仕方
   ～ヒープファイルを使って～
5. IOコストの比較
   ～ハッシュファイルと比較してみよう～

もっとも単純な格納方式
ヒープファイル

ヒープファイル

- もっとも基本的な格納の仕方
- ページにレコードを順次格納したファイル
- ページがレコードでいっぱいになったら新しいページを作る



データベースに対する問合せ（選択演算）

スキャン

- 全部のデータを読み込む

範囲問合せ

- age>10というような比較条件を用いたもの

完全一致問合せ

- 等号を使った問合せ（id='g0520434'）
- 答えは1つとは限らない

データベースに対する問合せ（更新処理）

挿入

- 1つのレコードを挿入する

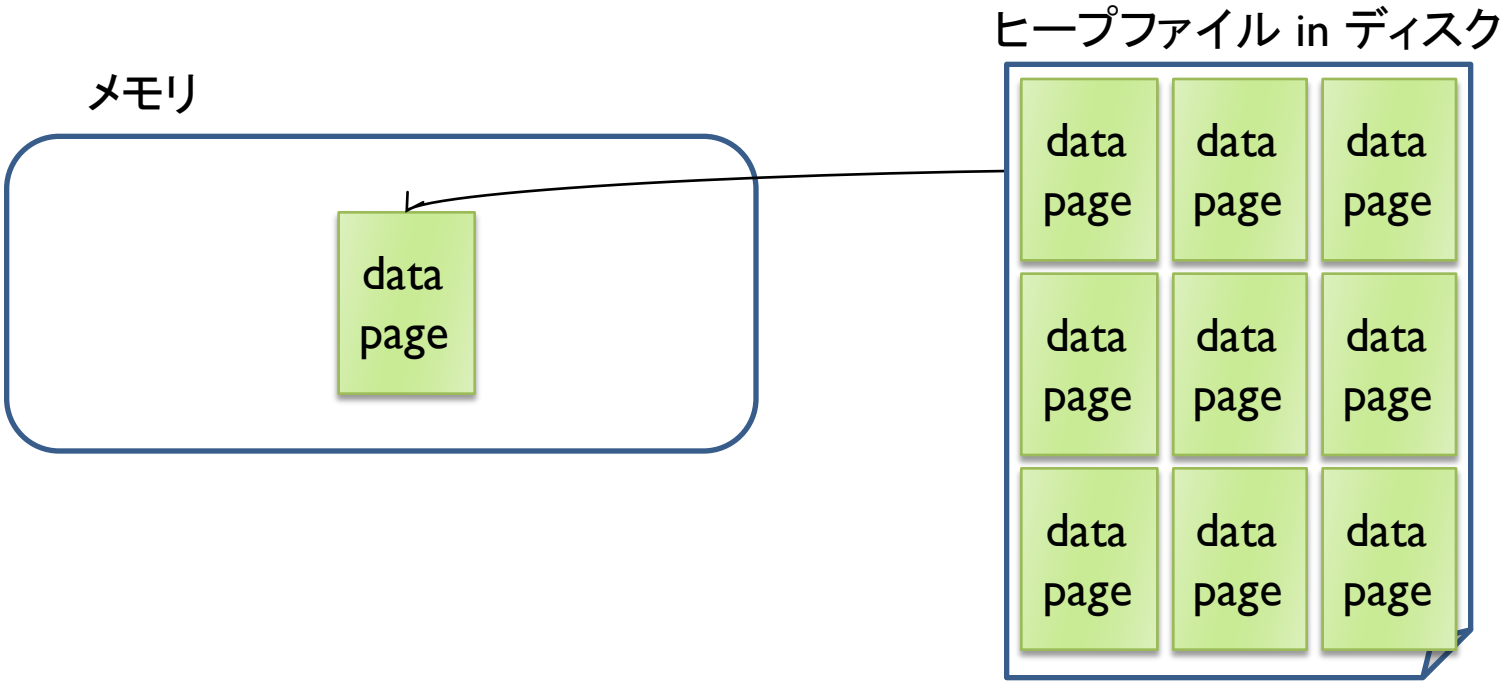
削除

- 1つのレコードを削除する

更新

- 1つのレコードの属性「氏名」を変更する

ヒープファイルをつかったスキャン



(1) 最初のデータページを取り出し，レコードを順に取り出す
(2) 全部のレコードを取り出し終えたら，次のデータページのアドレスを調べる（ページの中に書かれている）
(3) (2)で調べたアドレスから次のページを取り出す
(4) 次のデータページのアドレスがNULLになるまで(2),(3)を繰り返す

Question 1

- 以下の選択演算はどのように行うか処理の手順を書いてみましょう。また、ヒープファイルにNページあった時コストはどのくらいかかるか求めましょう

スキャン

- 全部のデータを読み込む

範囲問合せ

- age>10というような比較条件を用いたもの

完全一致問合せ

- 等号を使った問合せ（id='g0520434'）

Question 2

- 以下の演算はどのように行うか処理の手順を書いてみましょう。また、ヒープファイルにNページあった時コストはどのくらいかかるか求めましょう

挿入

- 1つのレコードを挿入する

削除

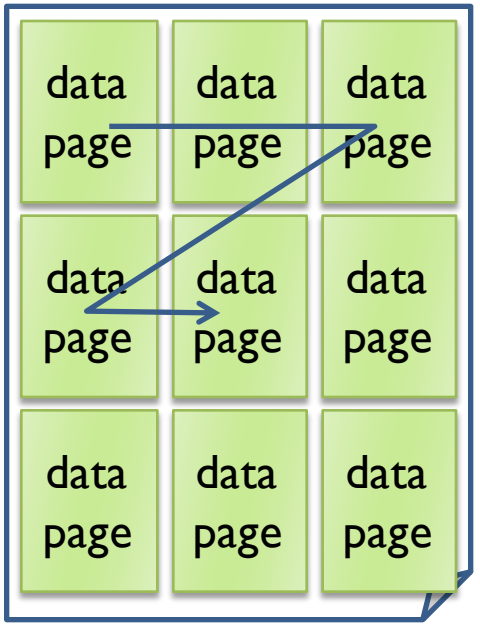
- 1つのレコードを削除する

更新

- 1つのレコードの属性「氏名」を変更する

ヒント：削除処理

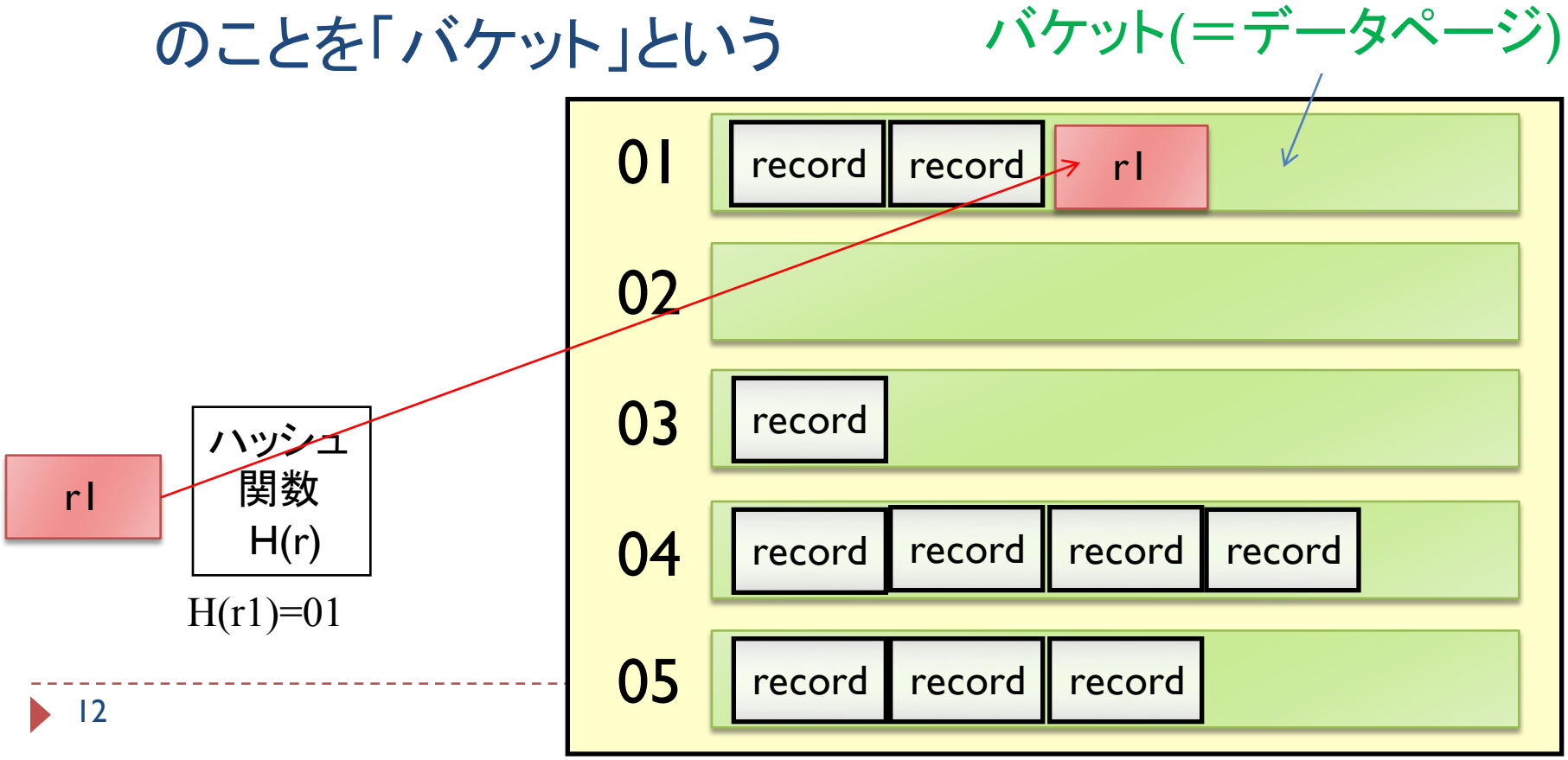
1. 対象レコードが見つかるまで読み込む
   - 最良の場合
     - 1ページ目で発見される
   - 最悪の場合
     - 最後まで見つからない
   - IOコストは平均で考える
2. 対象レコードを書き換えて
   ディスクに保存する
   - 書込みコストは1



IOコストの比較
~ ハッシュファイルとの比較 ~

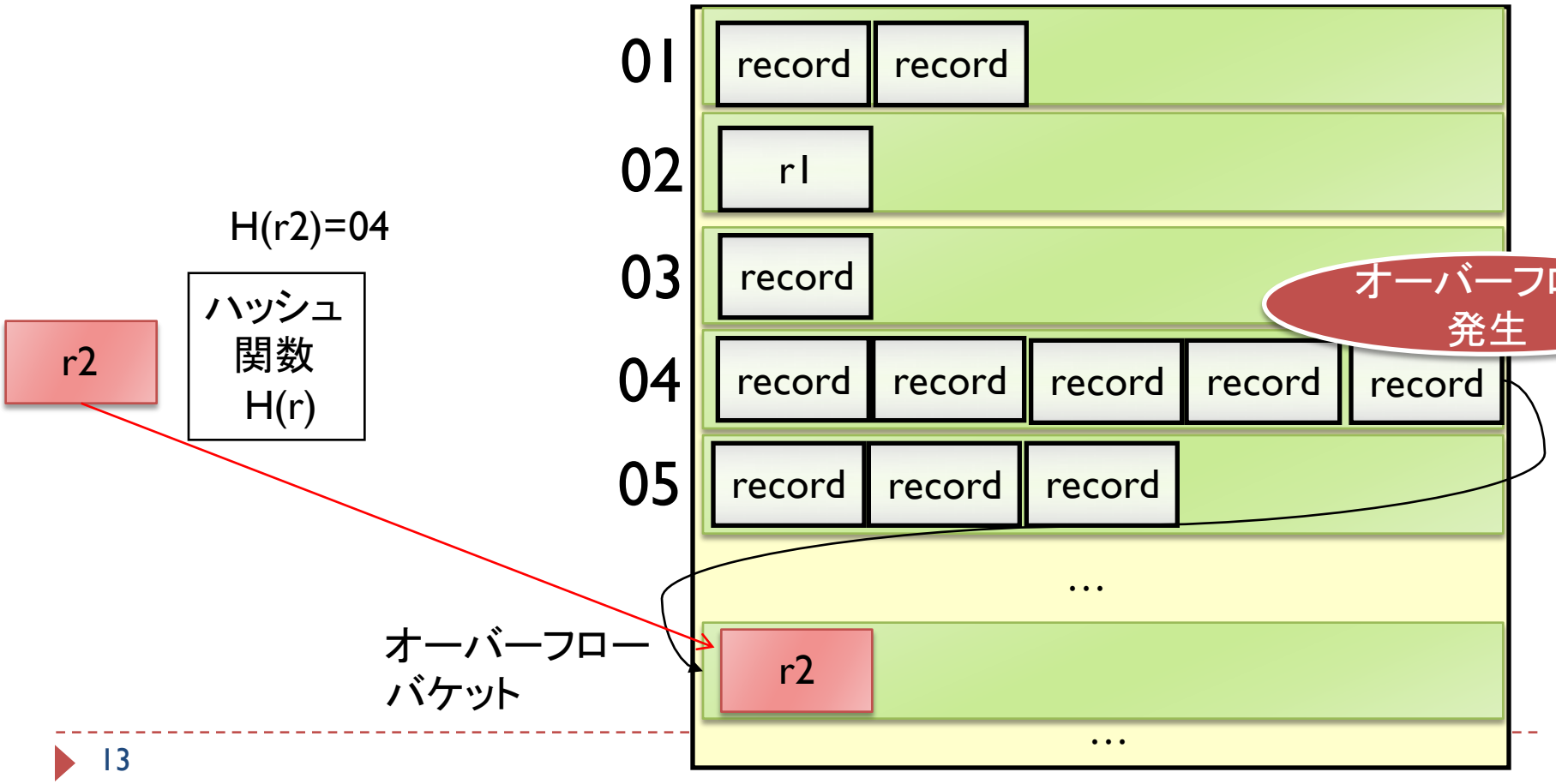
ハッシュファイル

- レコードRのキー値を適当なハッシュ関数を用いてハッシングし，格納するバケットを決定する
  - ハッシュファイルの場合ページのことを「バケット」という



レコードがいっぱいになった場合

- オーバーフローバケットを用意して、そこにレコードを追加する



バケットの数とハッシュ関数

- バケットの数は2のべき乗で，レコード数によって適切なバケット数を設定すべき
  - バケット数が多すぎると
    - 殆どレコードが入っていないバケットばかりでディスクに無駄な領域を確保しなければならない
  - バケット数が少なすぎると
    - すぐにバケットがいっぱいになり、オーバーフローが頻発する

ハッシュファイルに対する問合せ（選択演算）

スキャン

- 全部のデータを読み込む

範囲問合せ

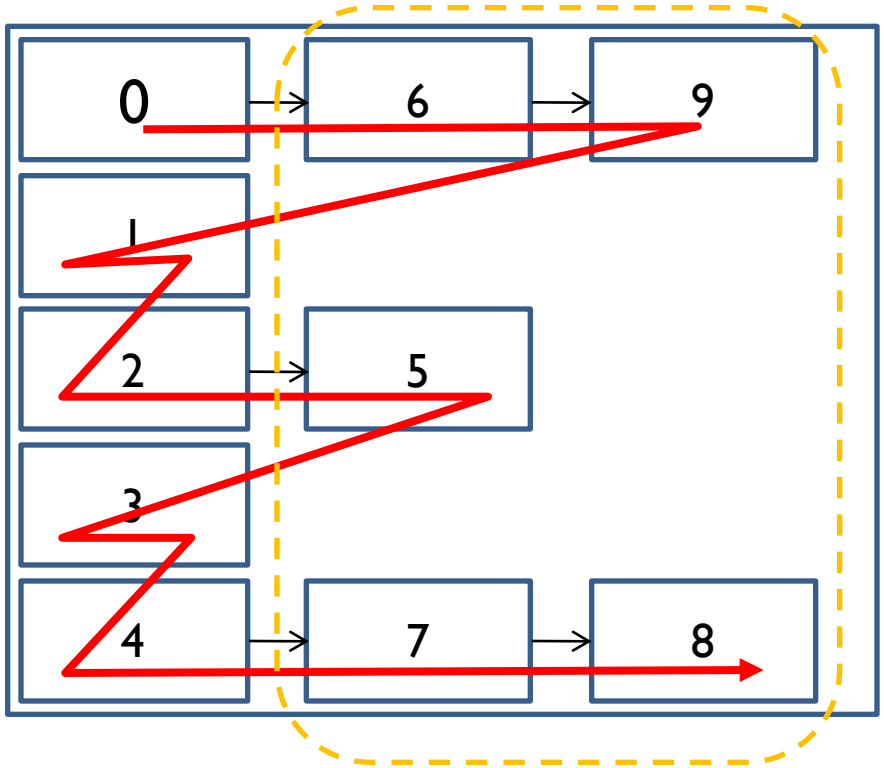
- age>10というような比較条件を用いたもの

完全一致問合せ

- 等号を使った問合せ（id='g0520434'）
- ハッシュ値はidをもとに求められるとする
- 答えは1つとは限らない

ハッシュファイルに対するスキャン

- ハッシュファイルに入っているバケットを上から順にメモリに呼び出してレコードを読み込む

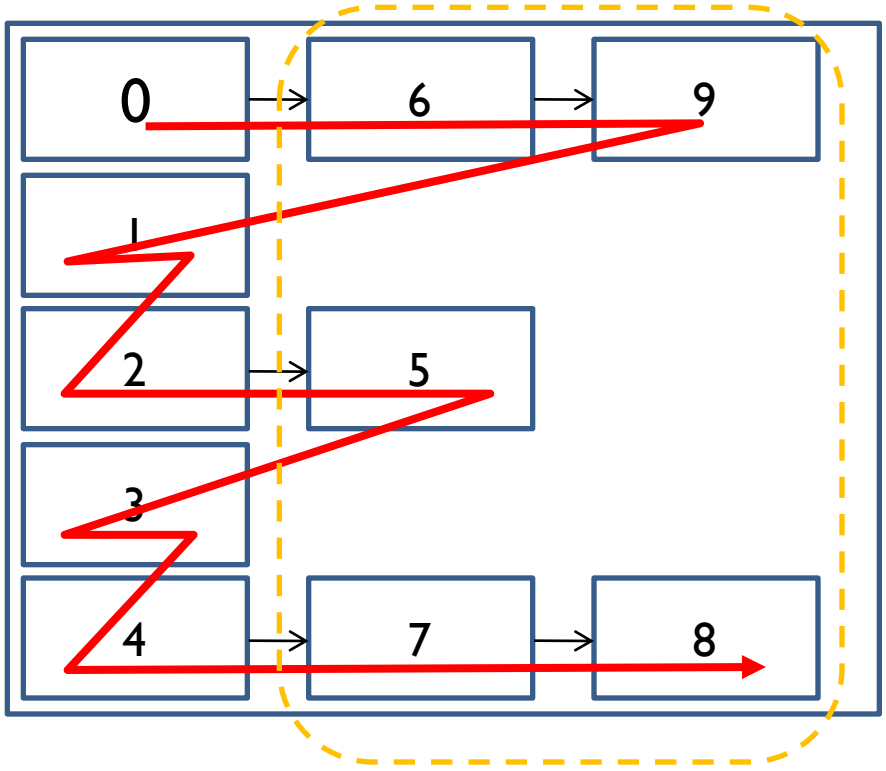


ハッシュファイルのバケット数見積もり

- 問題
  - ヒープファイルにレコードがぎっしり詰まってNバケットになった時、それと同数のレコードをハッシュファイルに入れたら、ハッシュファイルのバケット数はどのくらいになるだろうか？
    - ヒント：平均的に各バケットに5/6程度レコードが詰まっているとする
- 解答
  - A：N個
  - B：1.2 N個
  - C：2N個

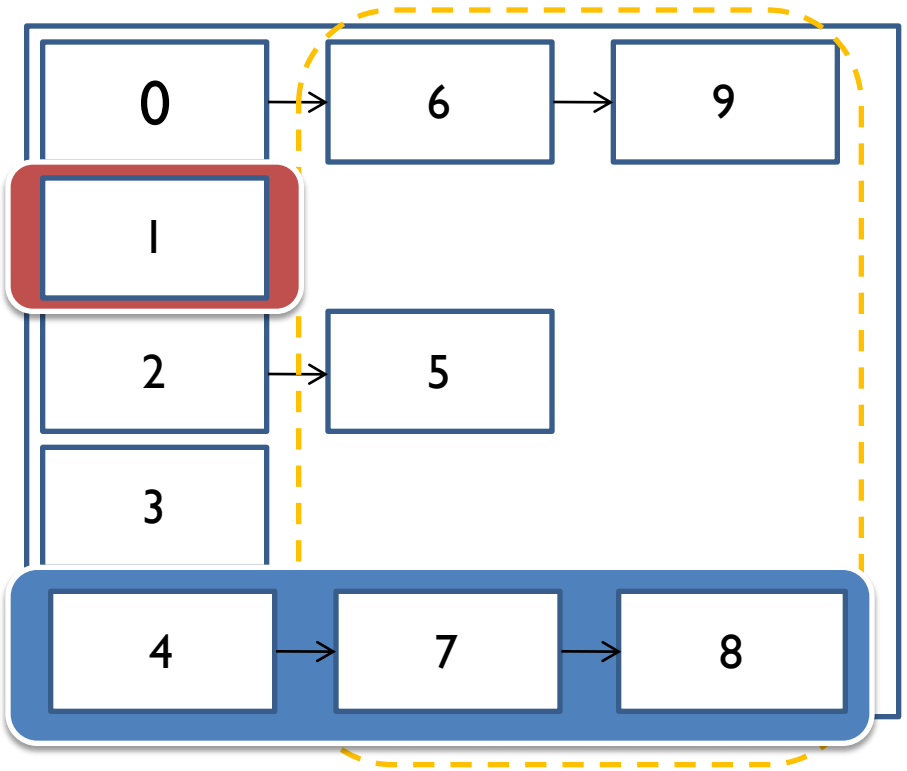
ハッシュファイルに対する範囲問合せ

- age>10というような比較条件を用いたもの
- 全てのバケットを上から順にメモリに呼び出してレコードを読み込み条件に合うかどうか調べる



ハッシュファイルに対する完全一致問合せ

- 検索するid番号からハッシュ値を求める
- ハッシュ値に該当するバケットを順番に読みだす
- 最良の場合
  - 1つのバケットしかない
- 最悪の場合
  - オーバーフローバケットがたくさん
  - 平均はとりづらいのでオーバーフロー分をαとしてコスト計算



Question 3

- 以下の選択演算はどのように行うか処理の手順を書いてみましょう。また、ヒープファイルでNページ分になる数のレコードをハッシュファイルで格納した時コストはどのくらいかかるか求めましょう

スキャン

- 全部のデータを読み込む

範囲問合せ

- age>10というような比較条件を用いたもの

完全一致問合せ

- 等号を使った問合せ（id='g0520434'）

Question 4

- 以下の演算はどのように行うか処理の手順を書いてみましょう。また、ヒープファイルでNページ分になる数のレコードをハッシュファイルで格納した時コストはどのくらいかかるか求めましょう

挿入

- 1つのレコードを挿入する

削除

- 1つのレコードを削除する

更新

- 1つのレコードの属性「氏名」を変更する

Question 5 (その1)

- ヒープファイルにNページあった場合の二つの格納形式のコストを比較し、どの処理にはどちらが向いているかを分析しよう

	ヒープファイル	静的ハッシュ	動的ハッシュ
スキャン			
範囲問合せ			
完全一致			
挿入			
削除			
更新			

動的ハッシュ法
（線形ハッシュ）

線形ハッシング

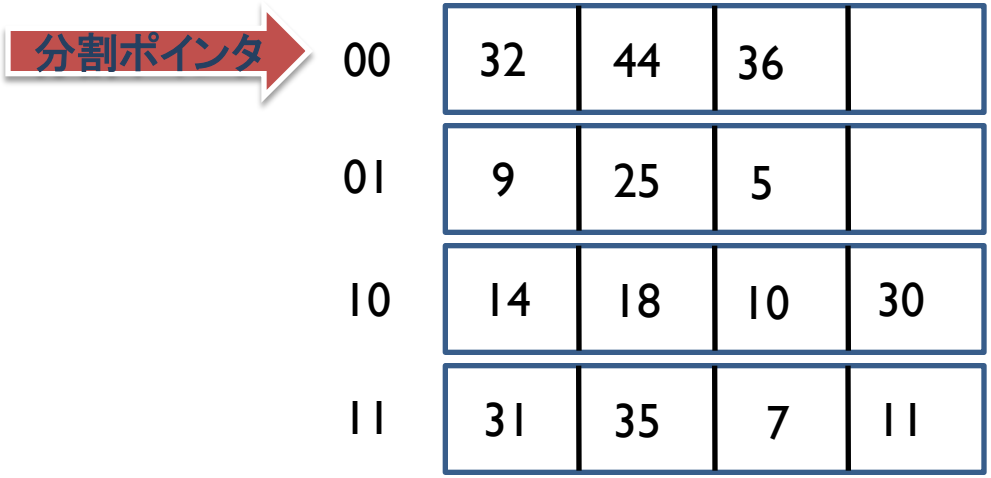
- できるだけオーバーフローをなくすことができる。
  （オーバーフローを一時的に許すのがポイント）
- *基本アイデア*
  - ハッシュ関数集合を使う　$\mathbf{H}_0, \mathbf{H}_1, \mathbf{H}_2, \ldots$
  - $\mathbf{H}_i(key) = \mathbf{h}(key) \bmod (2^i N)$
    - N：最初のバケット数
    - i：レベル数
  - 次の例ではN=4(=2^2)とする。
  - Nが2^{d0}である時、$2^i N (= 2^{i+d0})$で割っていくつ余るかを考えることは、
    下位i+d0 ビットを見ることと同等である。

注：ハッシュ関数はこの式が基本ですが後でちょっと変更がありますので注意！

線形ハッシング（挿入）

- 43を挿入する
- オーバーフローしたら分割。**でも分割するのは分割ポインタのあるバケット**

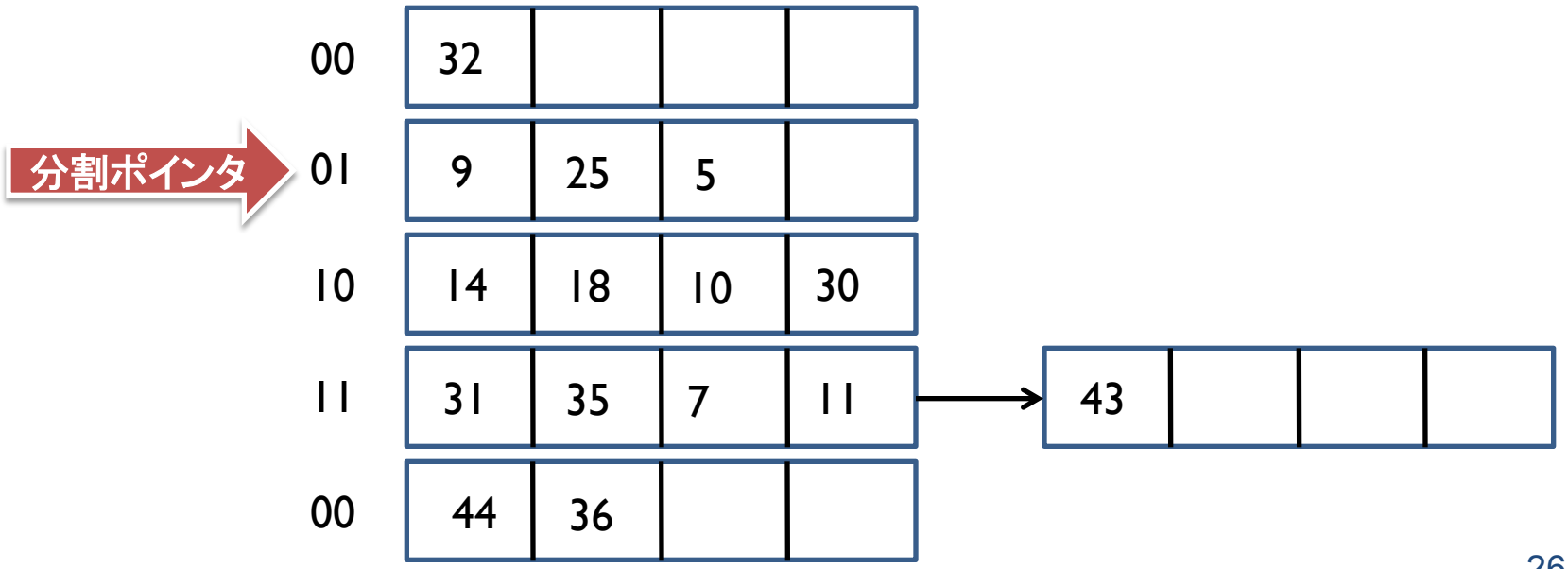
Level = 0　　最初のバケット数 N=4とする



線形ハッシング（挿入）

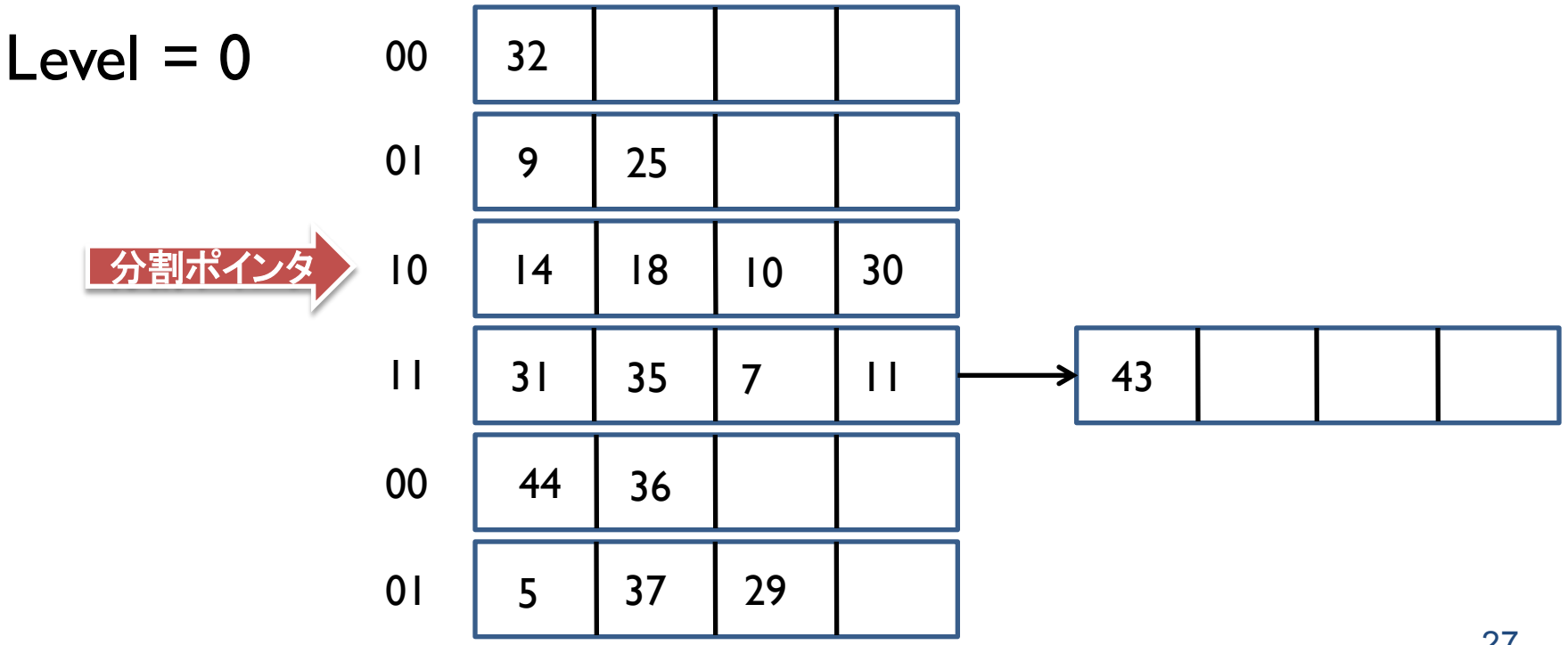
- 37, 29を挿入してみよう
  - h(37)=37 mod 2^{i}*4=1 → 01に入れる
  - H(29)=29 mod 2^{i}*4=1 → 01に入れる→オーバーフロー

Level = 0



線形ハッシング（挿入）

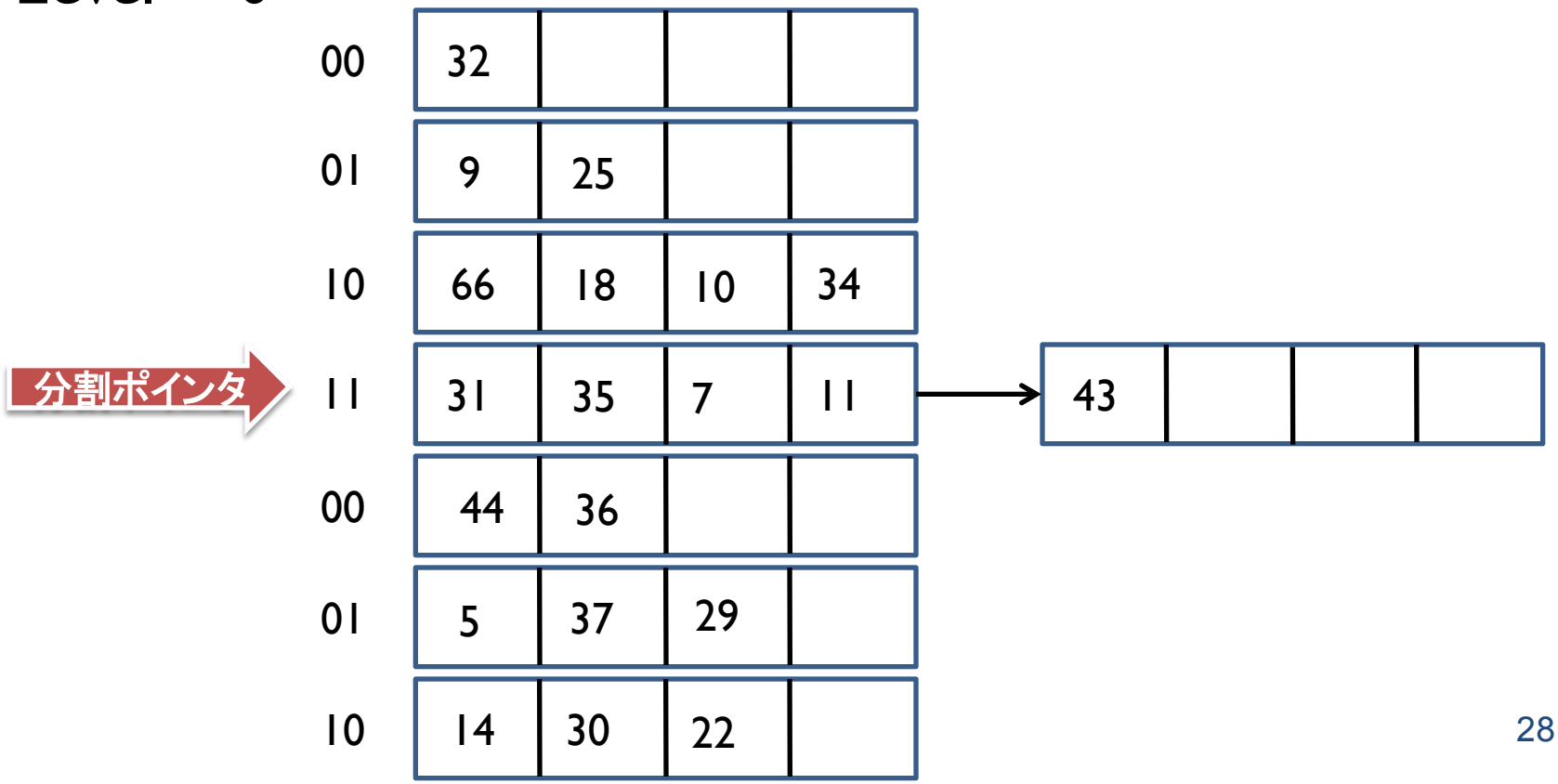
- 22,66,34を挿入してみよう
  - H0(22)=2 (10) → オーバーフロー
  - H0(66)=2 (10),H1(66)=2(10)
  - H(34)=2 (10),H1(34)=2(10)



線形ハッシング

- 50を挿入してみよう

Level = 0



ハッシュ関数について

- 下記の時点で52が来た場合

$H_0(52) = 52 \bmod (2^{2+0}) = 0$ ⇒ バケット00へ？

しかしバケット00はすでにバケット000とバケット100に分割されている！

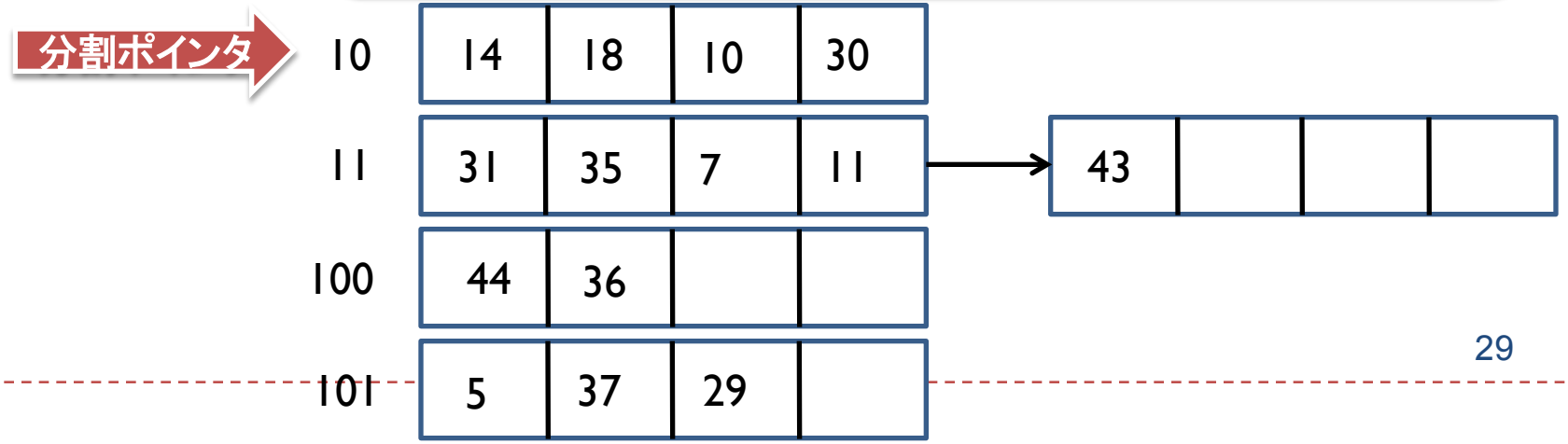
しかも52は本来ならばバケット100のほうに行くはず

Level = 0

$H_{Level}(x) = x \bmod (2^{2+Level})$ の値が分割ポインタの値より小さい場合は

$H_{Level+1}(x) = x \bmod (2^{2+Level+1})$

を適用する



Question 5 (その2)

- 動的ハッシュのコストも考えてみて、使い道の良し悪しをさらに分析してみよう

	ヒープファイル	静的ハッシュ	動的ハッシュ
スキャン			
範囲問合せ			
完全一致			
挿入			
削除			
更新			