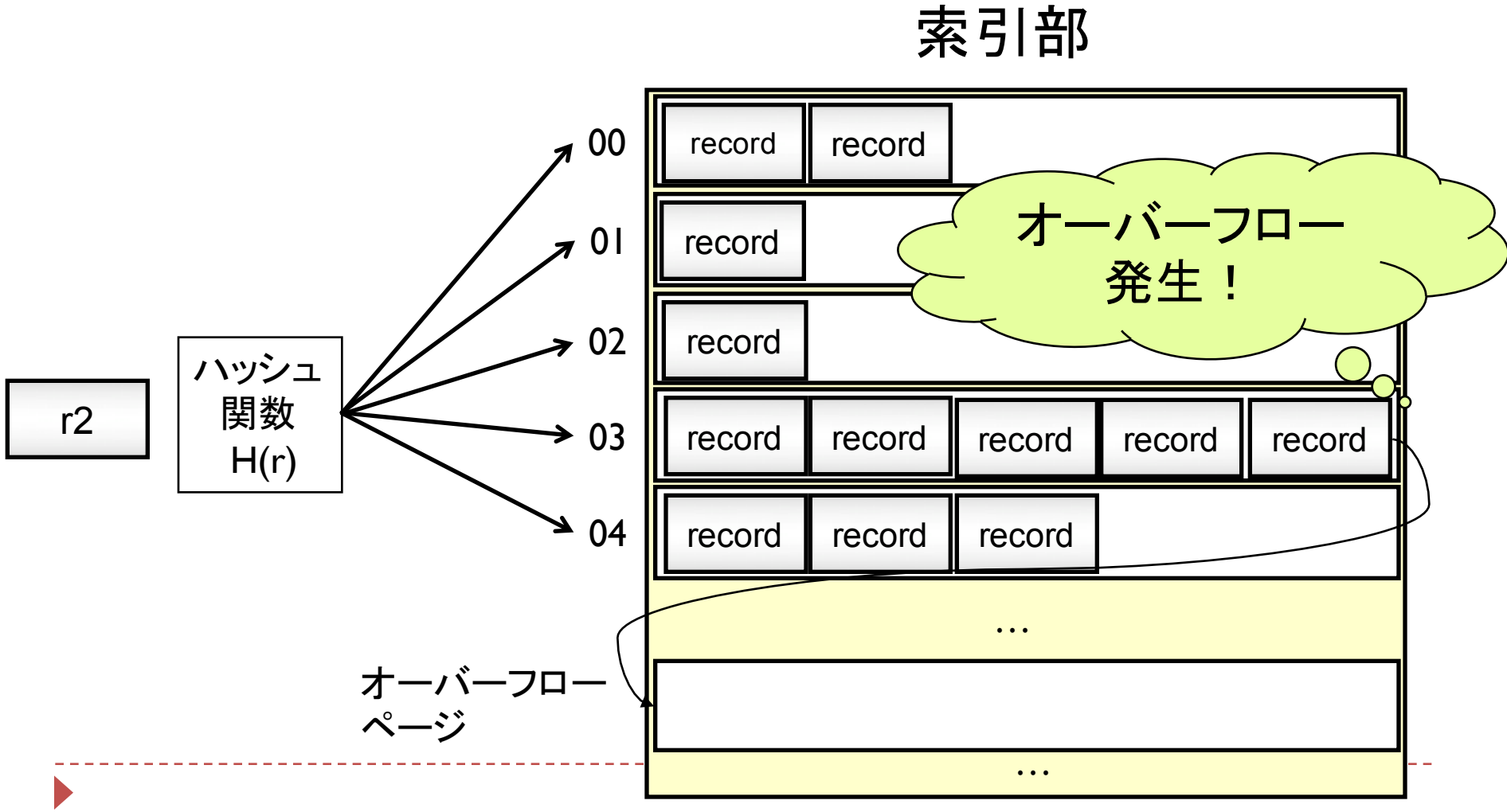
データベースシステム

データの格納方式 (2)

線形ハッシュと二次索引

ハッシュ索引

- 静的ハッシング



ハッシュ索引

- 静的ハッシングの問題点

  ハッシュのバケット分割に大きな偏りが生じた場合，バケットオーバーフローが数多く発生してしまいIOコストが急激に低下する可能性がある

- 動的ハッシング
  - 拡張可能ハッシング
  - 線形ハッシング

動的ハッシュ法
（線形ハッシュ）

線形ハッシング

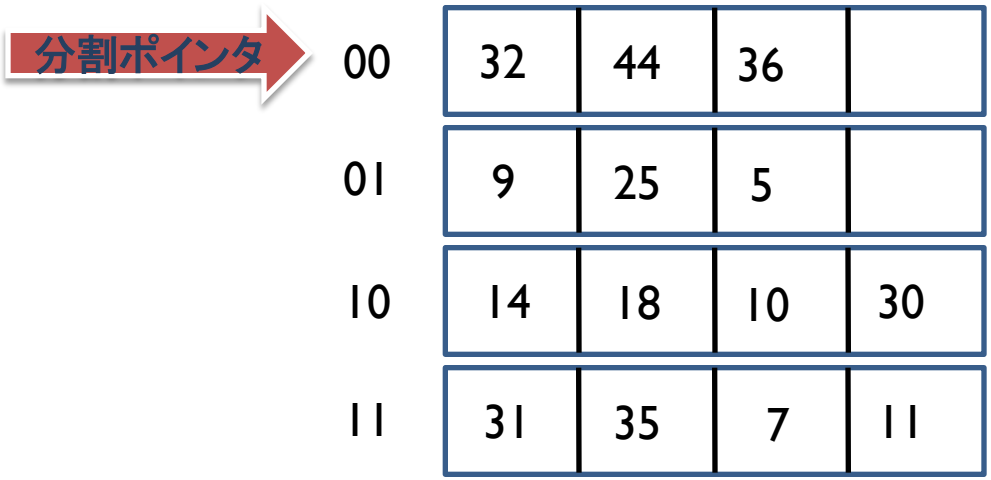
- できるだけオーバーフローをなくすことができる。
  （オーバーフローを一時的に許すのがポイント）
- *基本アイデア*
  - ハッシュ関数集合を使う　$\mathbf{H}_0, \mathbf{H}_1, \mathbf{H}_2, \ldots$
  - $\mathbf{H}_i(key) = \mathbf{h}(key) \bmod (2^i N)$
    - N：最初のバケット数
    - i：レベル数
  - 次の例ではN=4(=2^2)とする。
  - Nが2^{d0}である時、$2^i N (= 2^{i+d0})$で割っていくつ余るかを考えることは、
    下位i+d0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

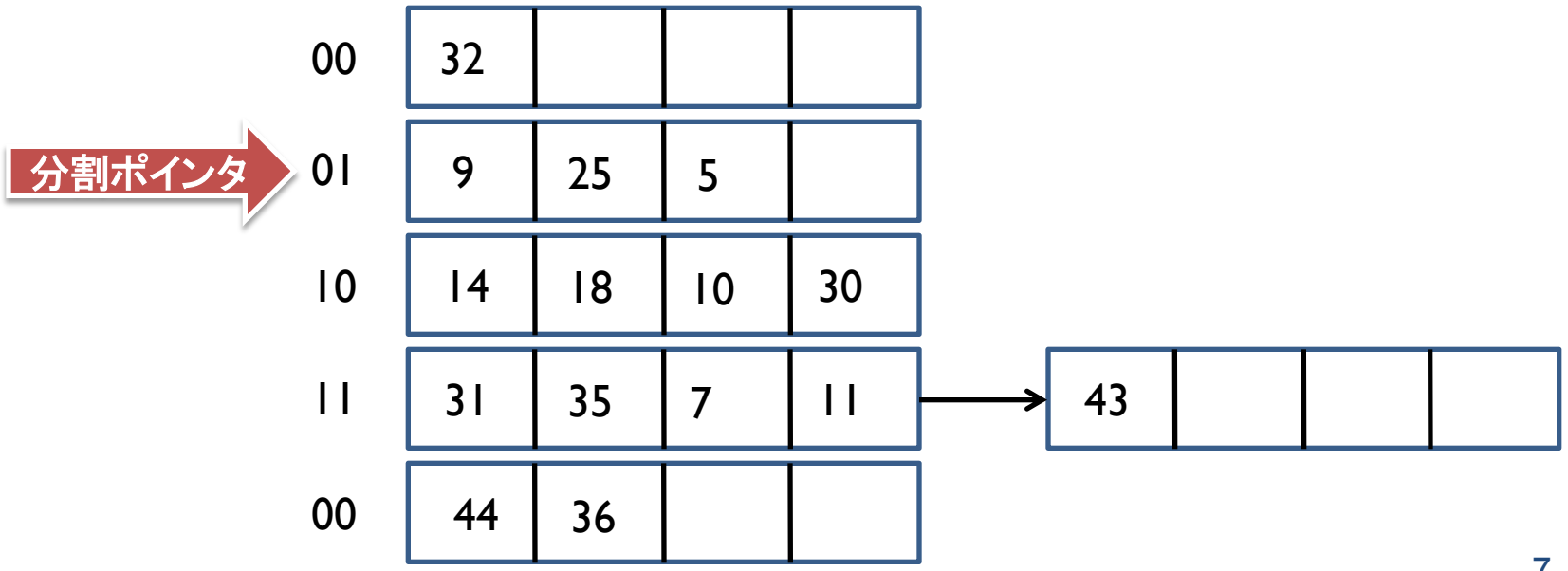
Level = 0　　最初のバケット数 N=4とする



線形ハッシング（挿入）

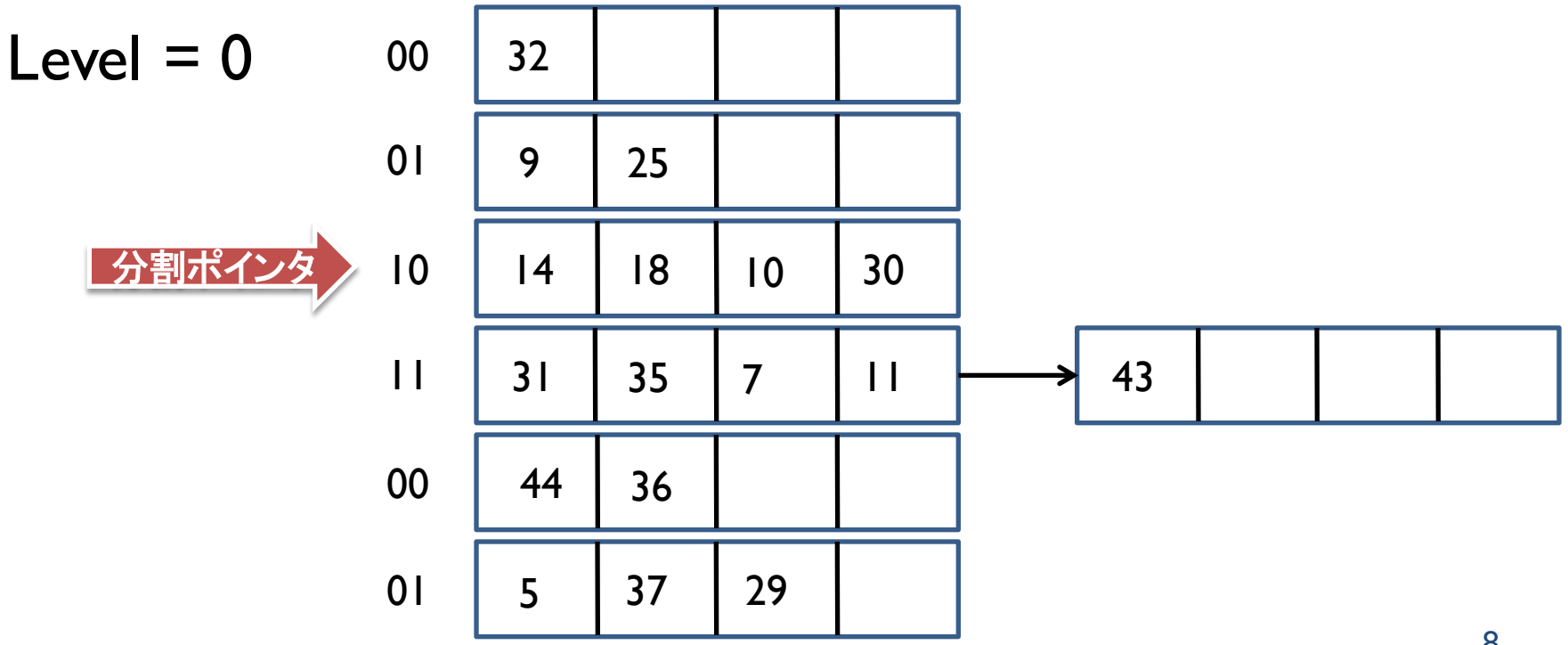
- 37, 29を挿入してみよう
  - h(37)=37 mod 2^{i}*4=1 → 01に入れる
  - H(29)=29 mod 2^{i}*4=1 → 01に入れる→オーバーフロー

Level = 0



線形ハッシング（挿入）

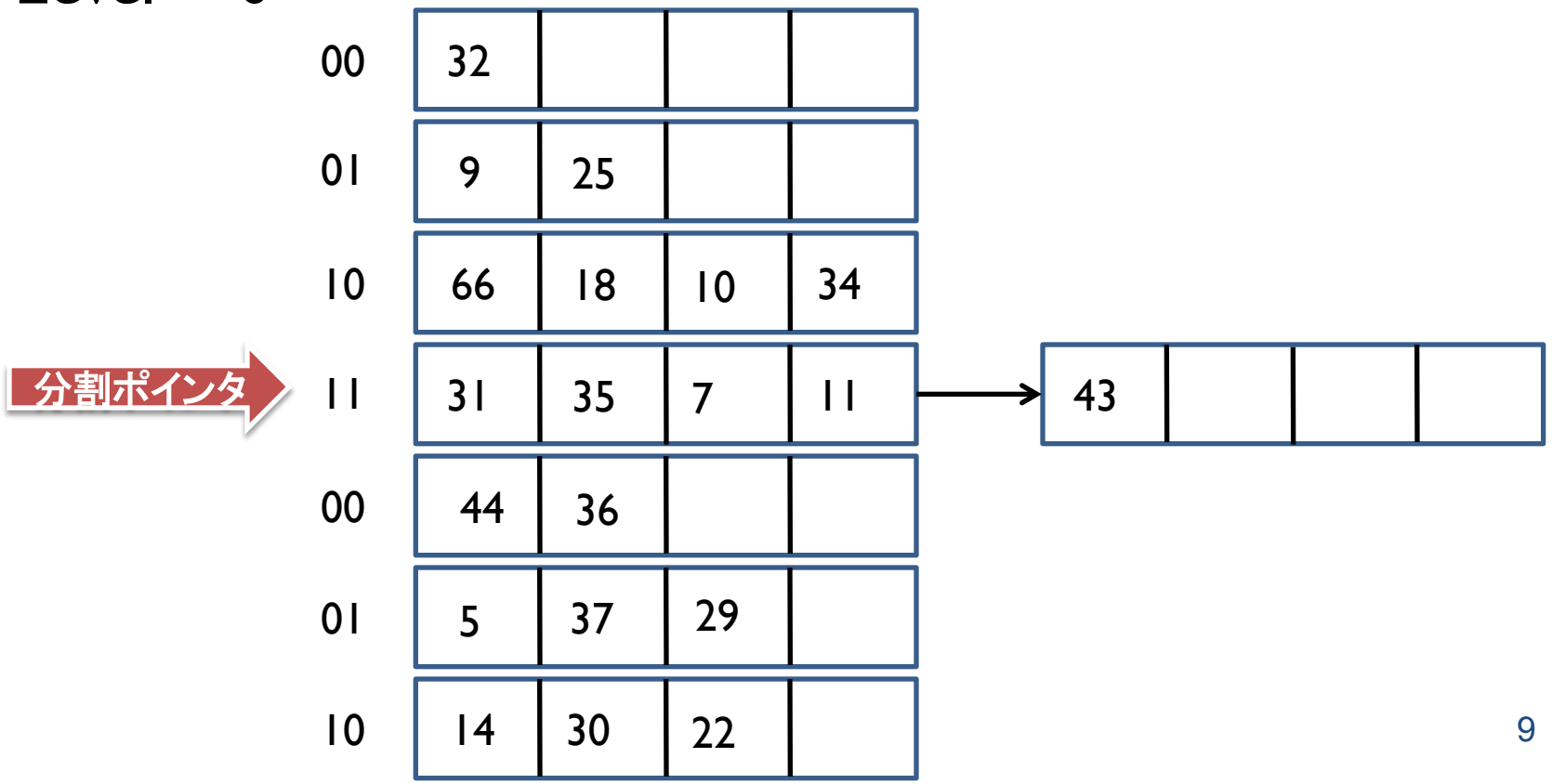
- 22,66,34を挿入してみよう
  - H0(22)=2 (10) → オーバーフロー
  - H0(66)=2 (10),H1(66)=2(10)
  - H(34)=2 (10),H1(34)=2(10)



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！

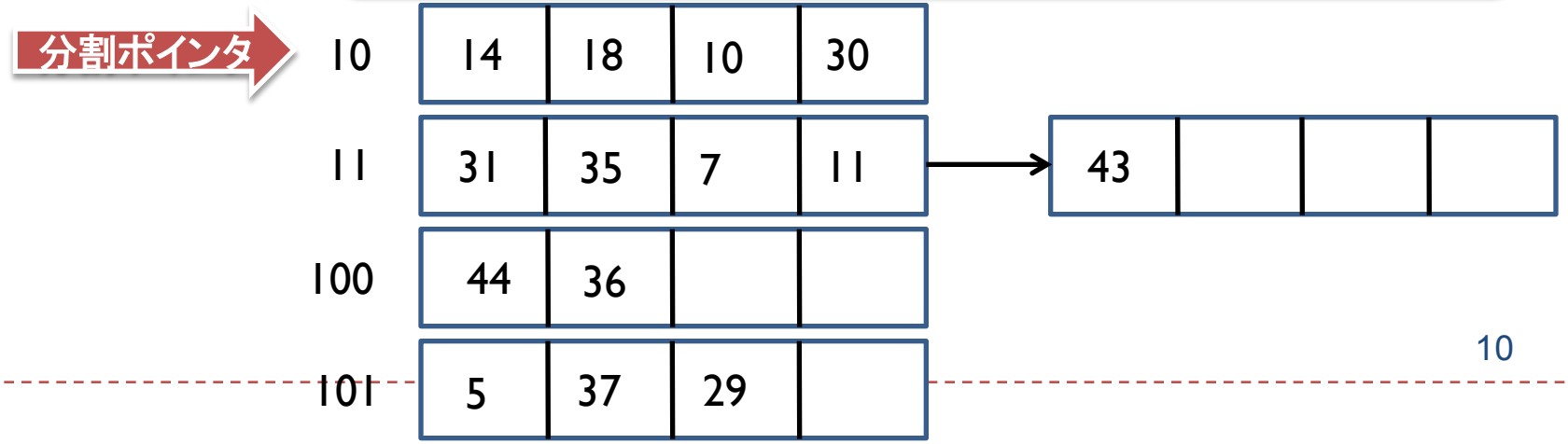
しかも52は本来ならばバケット100のほうに行くはず

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



演習1

- 線形ハッシュアルゴリズム（最初のバケット数N=2, バケット内の要素数を2とする）を使って以下の挿入を行った後のハッシュテーブルを求めよ。線形ハッシュでは最初のバケット数をNとし、レベルをiとしたとき、レベルiのハッシュ関数は

  $$Hi(key) = h(key) \bmod (2^i \cdot N)$$

  となる。なおこの問題でのハッシュ値 h(key)は簡単のため

  $$h(key)=key$$

  とする

  19, 39,12,3,53,66,34,30,78,23,11

  なおこれらをそれぞれ2進数にすると以下のとおりとなる

  10011(19), 100111(39), 1100(12), 11(3), 110101(53), 1000010(66), 100010(34), 11110(30), 1001110(78), 10111(23), 1011(11)

演習２：ＩＯコストの比較

	Heap file	Hash File	Linear Hash
スキャン			
範囲問合せ			
完全一致			
挿入			
削除			
更新			

データ格納方式と索引の付与

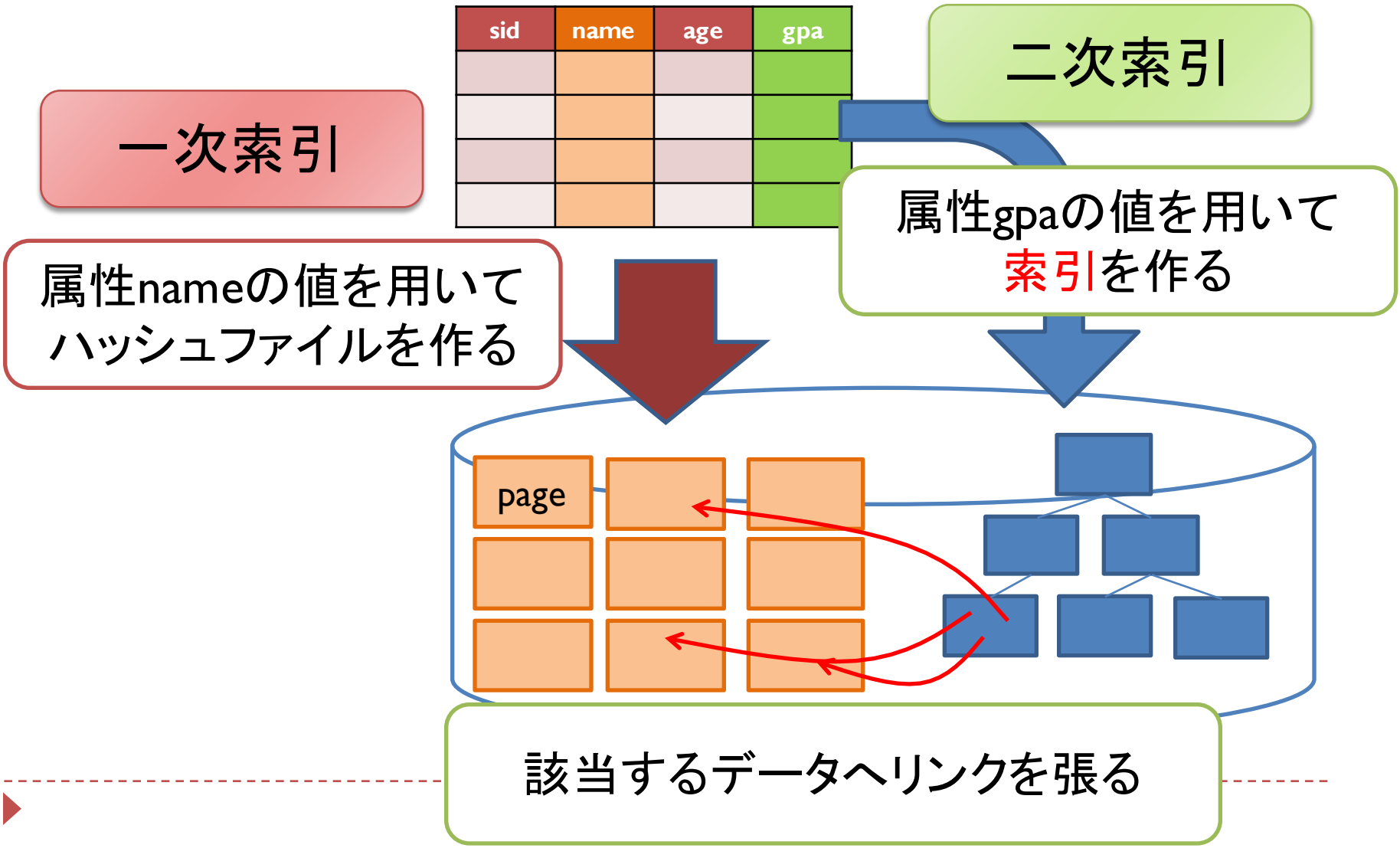
以下のような場合どうする？

- nameを条件にした問合せが最も多いが、gpaを条件にした問合せも多い
- どちらの問合せも高速に処理したい

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

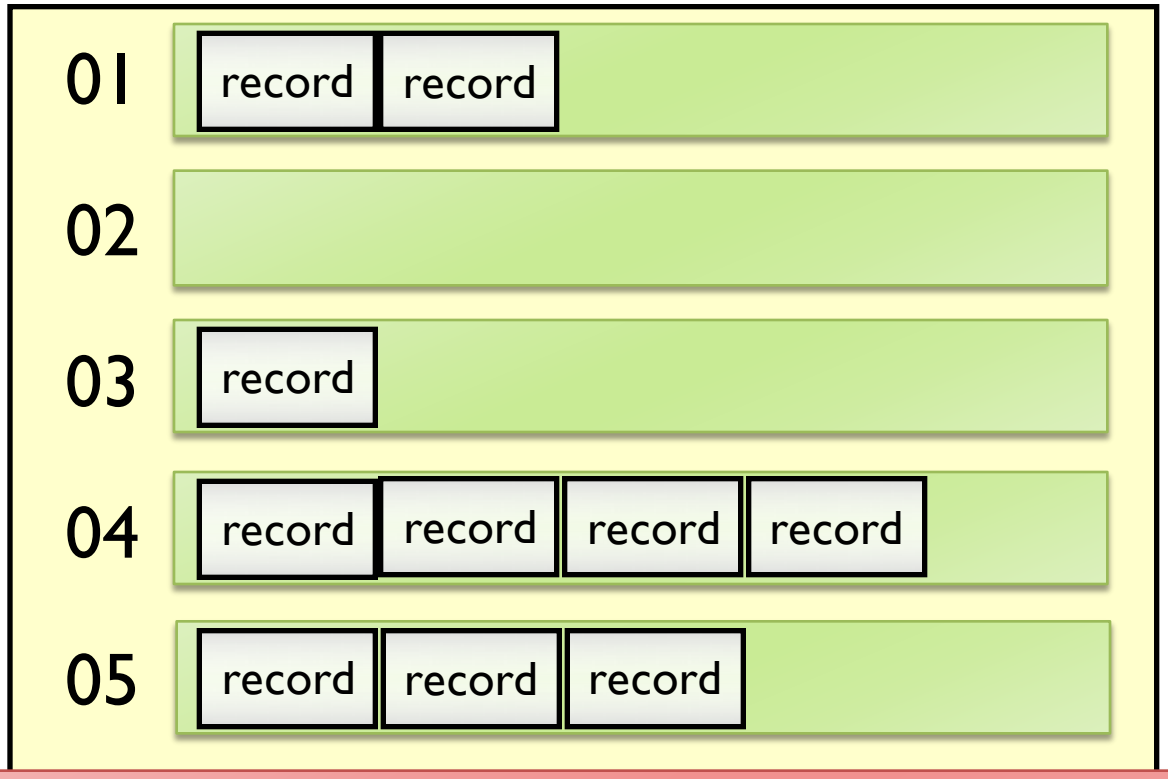
- nameでハッシュファイルを作る
  - gpaによる問合せは高速化されない
- gpaでB+-treeを作る
  - nameによる問合せは高速化されない

二次索引


sid
name
age
gpa
一次索引
二次索引
属性gpaの値を用いて
索引を作る
属性nameの値を用いて
ハッシュファイルを作る
page
該当するデータへリンクを張る

ハッシュファイルを二次索引として使うと？

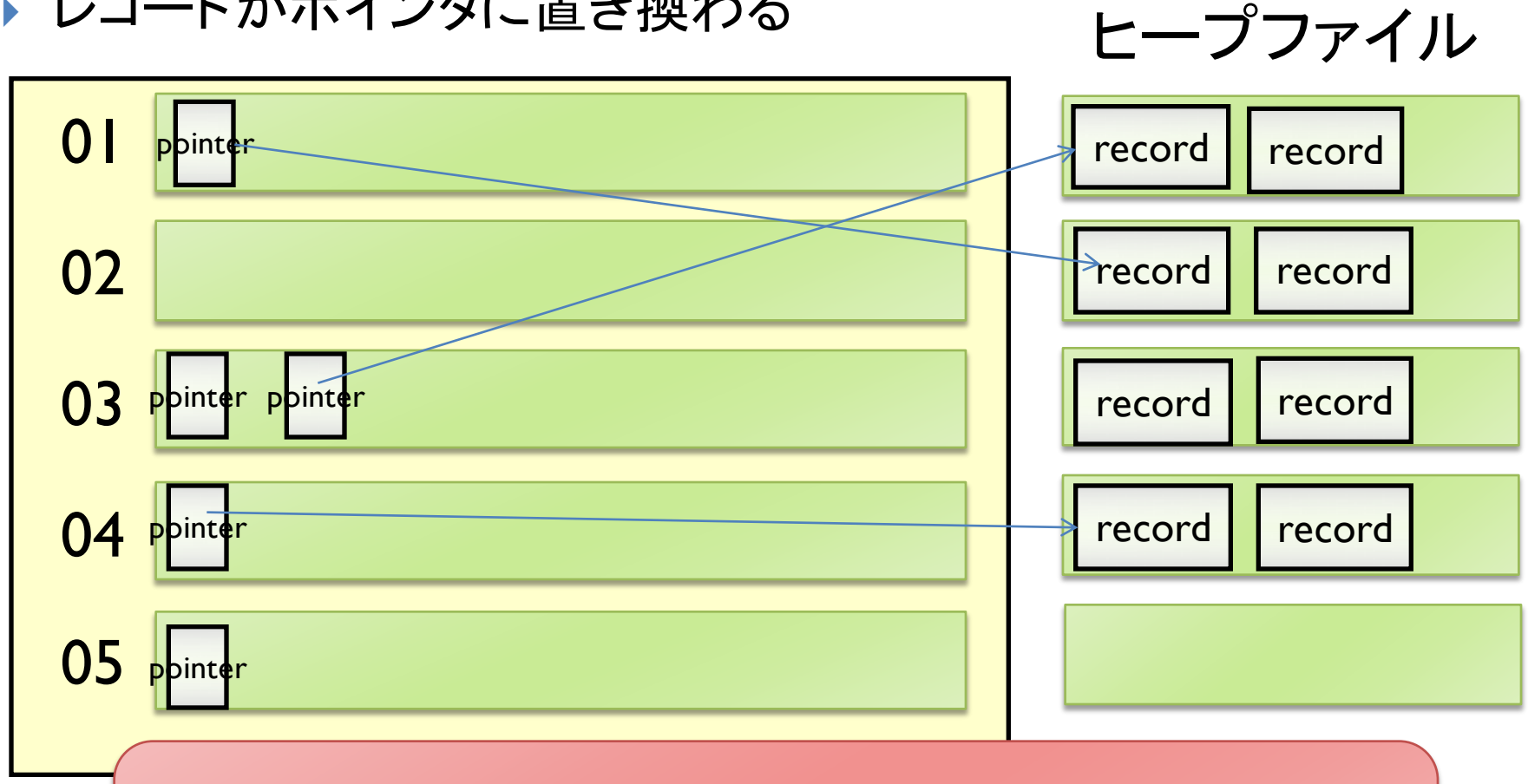
- 復習：これは一次索引の場合



問題：物理格納方式をヒープファイル、
二次索引でハッシュファイルを使ったらどう変わる？

ハッシュファイルを二次索引として使うと？

- レコードがポインタに置き換わる



バケットに入るポインタの数が増える
ポインタの大きさはバケットの1/10くらい

演習 3

- ハッシュファイルをデータ格納方式にしていた時Nページあったとする。データ格納方式をヒープファイルにしてハッシュファイルを二次索引にするとしたら、ページ数はいくつ必要になるだろうか？
  - ヒント
    - ポインタの大きさはレコードの1/10
    - ページやバケットの大きさは同じ
    - バケットの中の5/6はポインタで埋まるようにする。
      - バケットの中身をレコードからポインタにしたら 5/60しか埋まらなくなってしまうので、5/6にするようにバケットの数を減らす

演習 4

- ヒープファイルをデータ格納方式に使い、ハッシュ索引を二次索引に使った時のコストを計算しよう
  - ヒープファイルはNページ
  - ページに入るレコードの数はRレコードとする

スキャン

- 全部のデータを読み込む

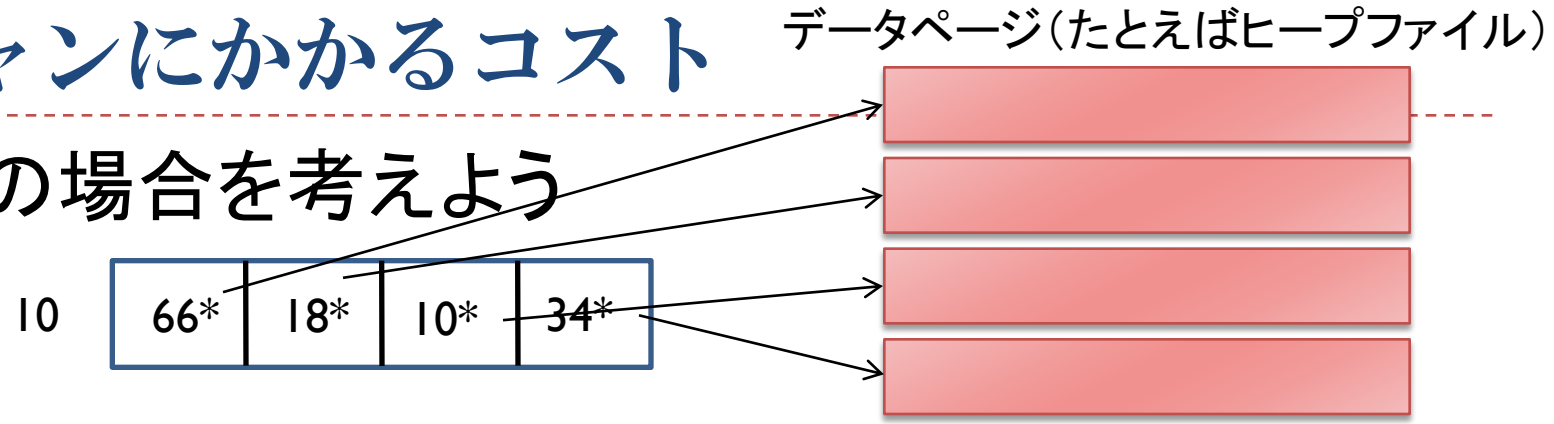
範囲問合せ

- 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

スキャンにかかるコスト



- 最悪の場合を考えよう

1バケットにつき，最悪そのバケットに含まれるデータエントリ分のデータページを読みだす必要がある

最悪の場合のデータページアクセス数は

$$R*10*(5/6)=\frac{25}{3}R$$

- ヒープファイルがNページの時、ハッシュファイル（二次索引）のバケット数は？

$$N*\frac{6}{5}*\frac{1}{10}=\frac{3}{25}N$$

演習4

- ヒープファイルをデータ格納方式に使い、ハッシュ索引を二次索引に使った時のコストを計算しよう

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

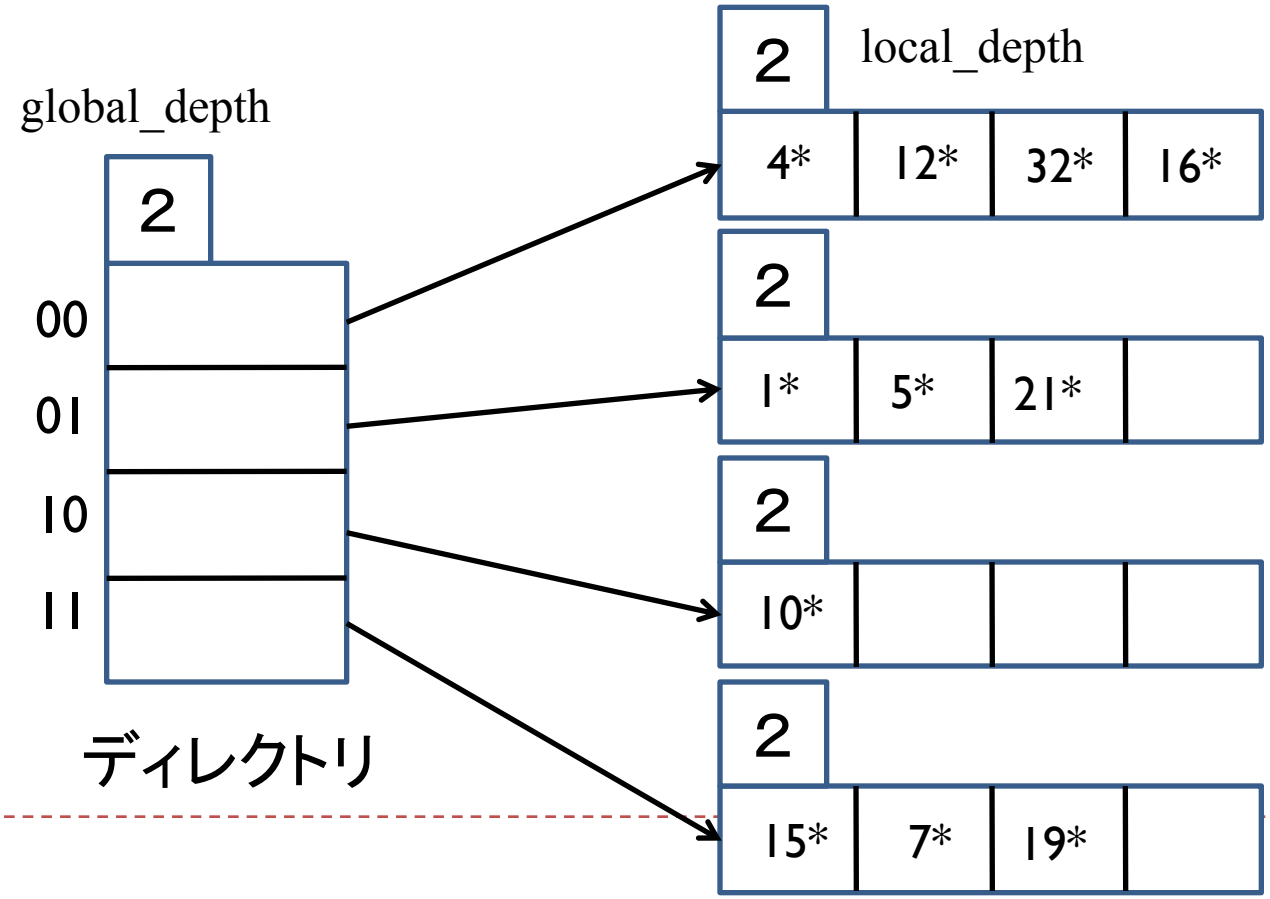
これまでのIOコストを比較しよう

	Heap file	Hash File	線形	線形（二次）
スキャン	N	N		
範囲問合せ	N	N		
完全一致	N	1+α		
挿入	2	2		
削除	(N+1)/2+1	2+α		
更新	(N+1)/2+1	2+α		

おまけ
拡張可能ハッシング

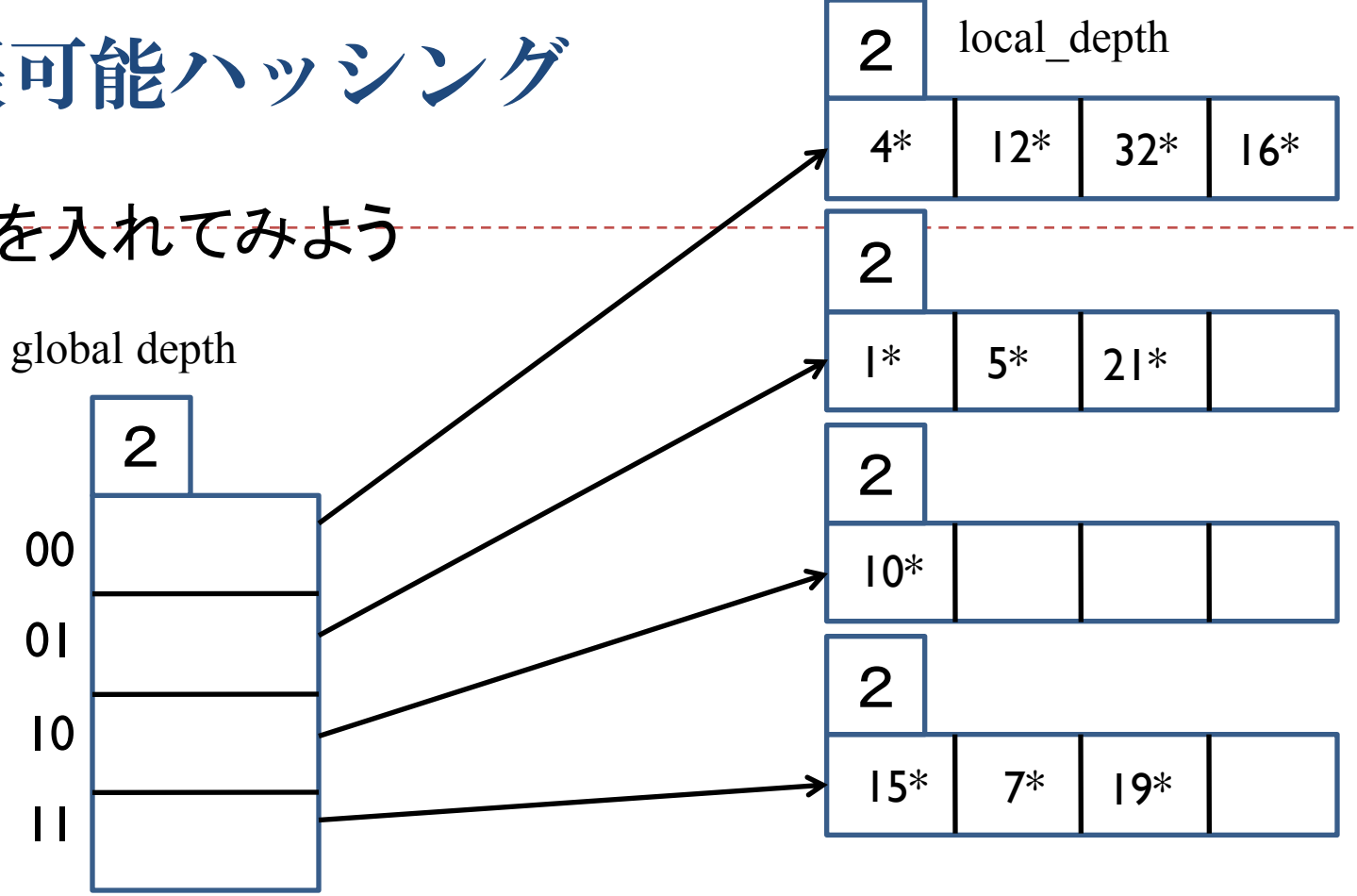
拡張可能ハッシング

- オーバーフローがいやならバケットを増やせばいい
- ハッシュページにディレクトリを用意する
- $h(v) = v \bmod 2^{\text{global\_depth}}$ とする



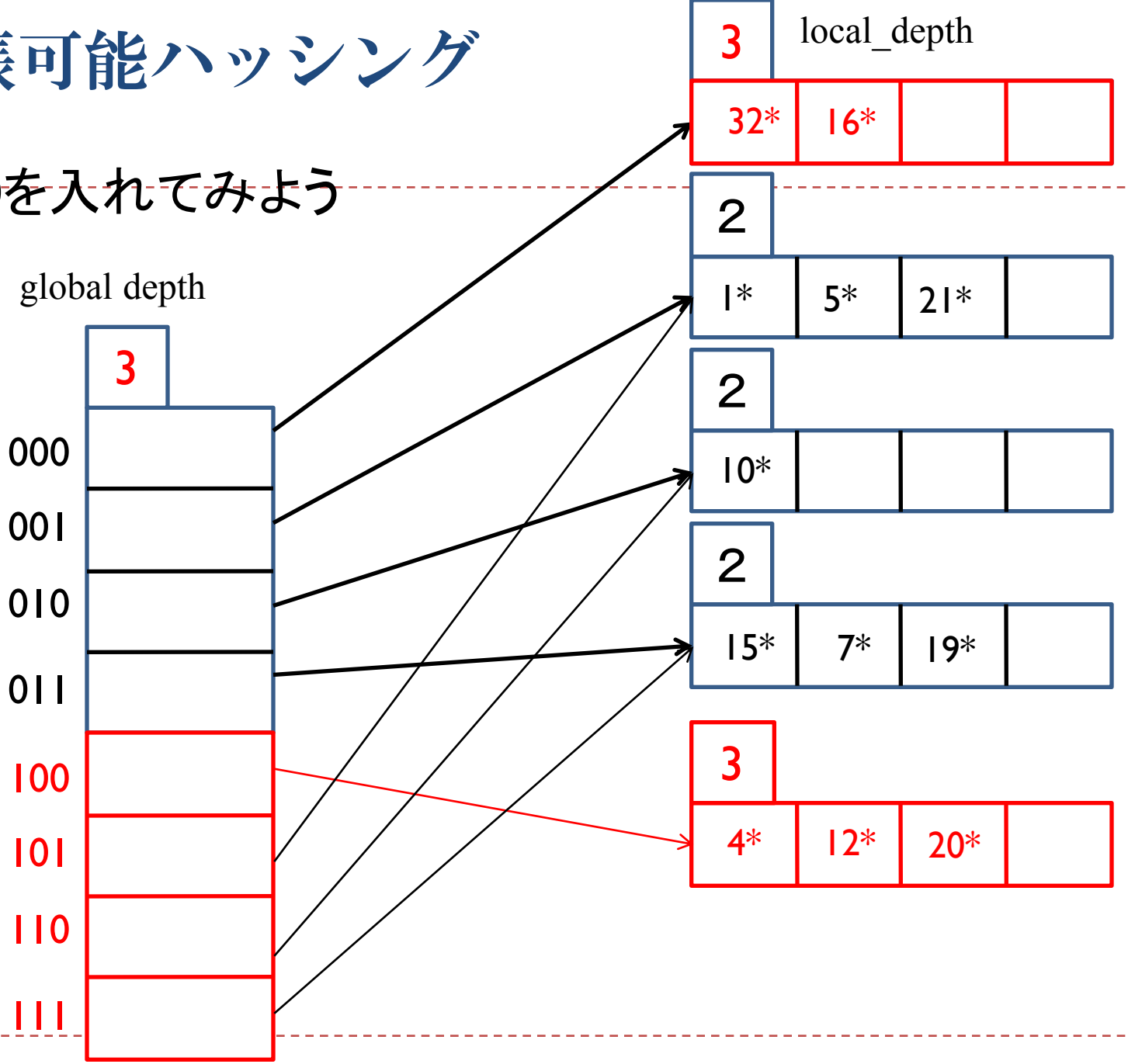
拡張可能ハッシング

- 20を入れてみよう



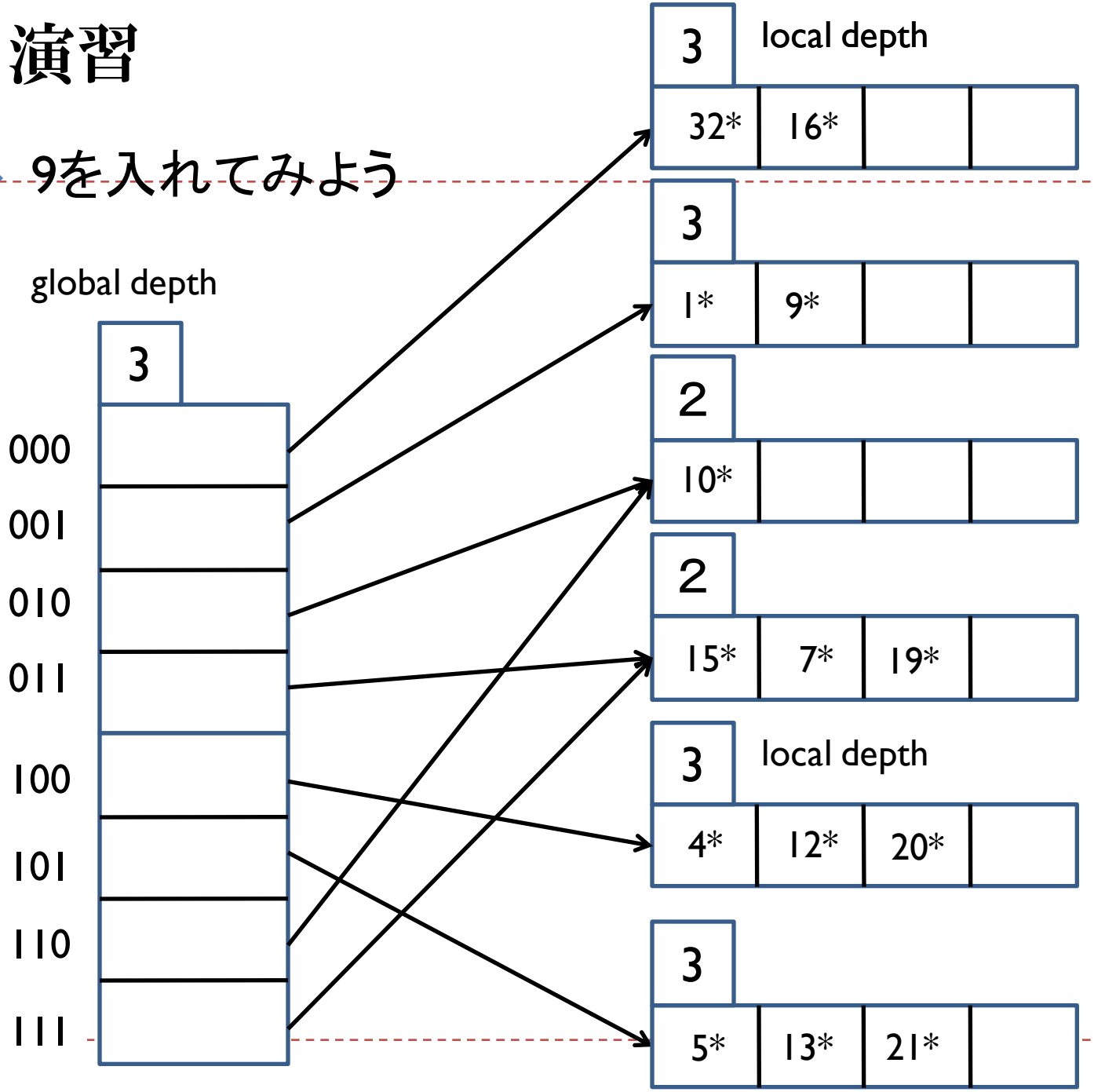
拡張可能ハッシング

- 20を入れてみよう



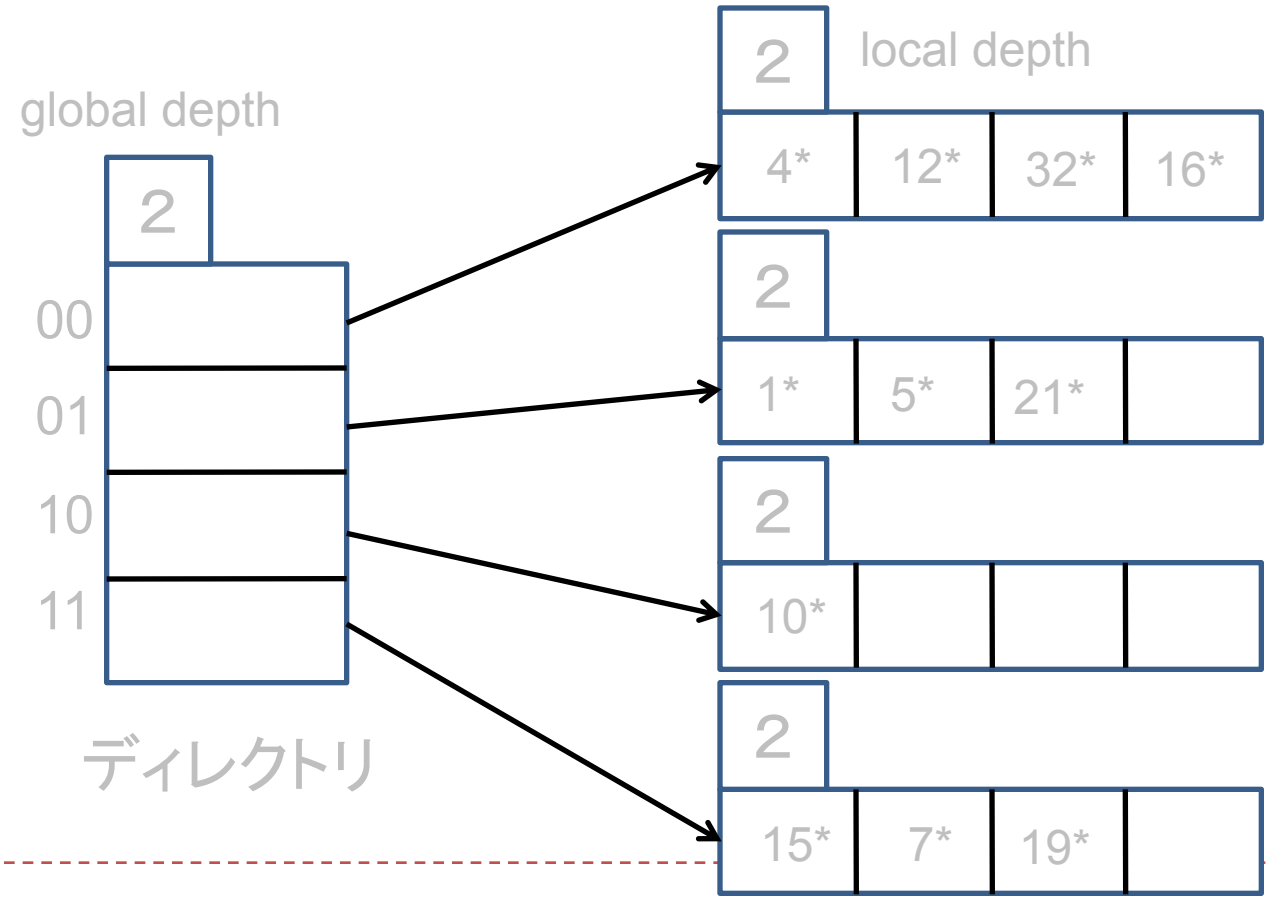
演習

- 9を入れてみよう


global depth
3
000
001
010
011
100
101
110
111
3
local depth
32*
16*
3
1*
9*
2
10*
2
15*
7*
19*
3
local depth
4*
12*
20*
3
5*
13*
21*

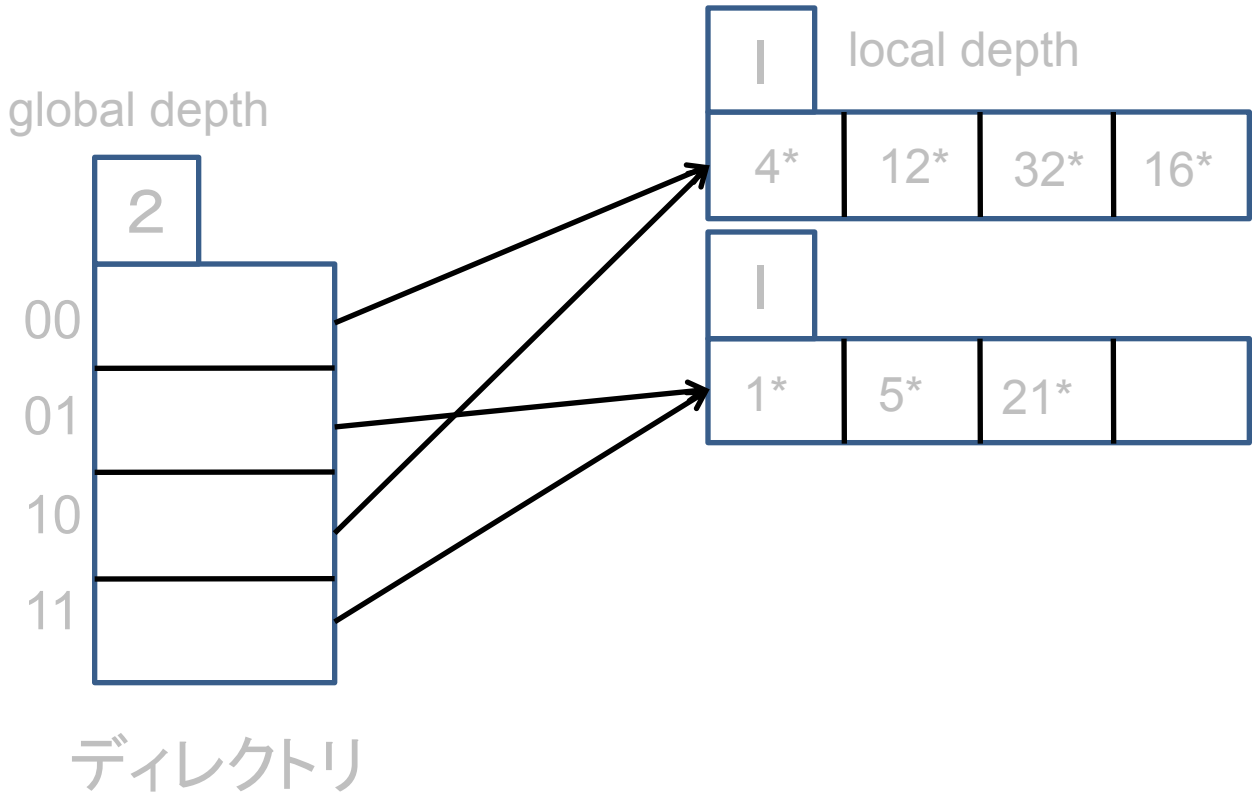
拡張可能ハッシング

- 10を消してみよう



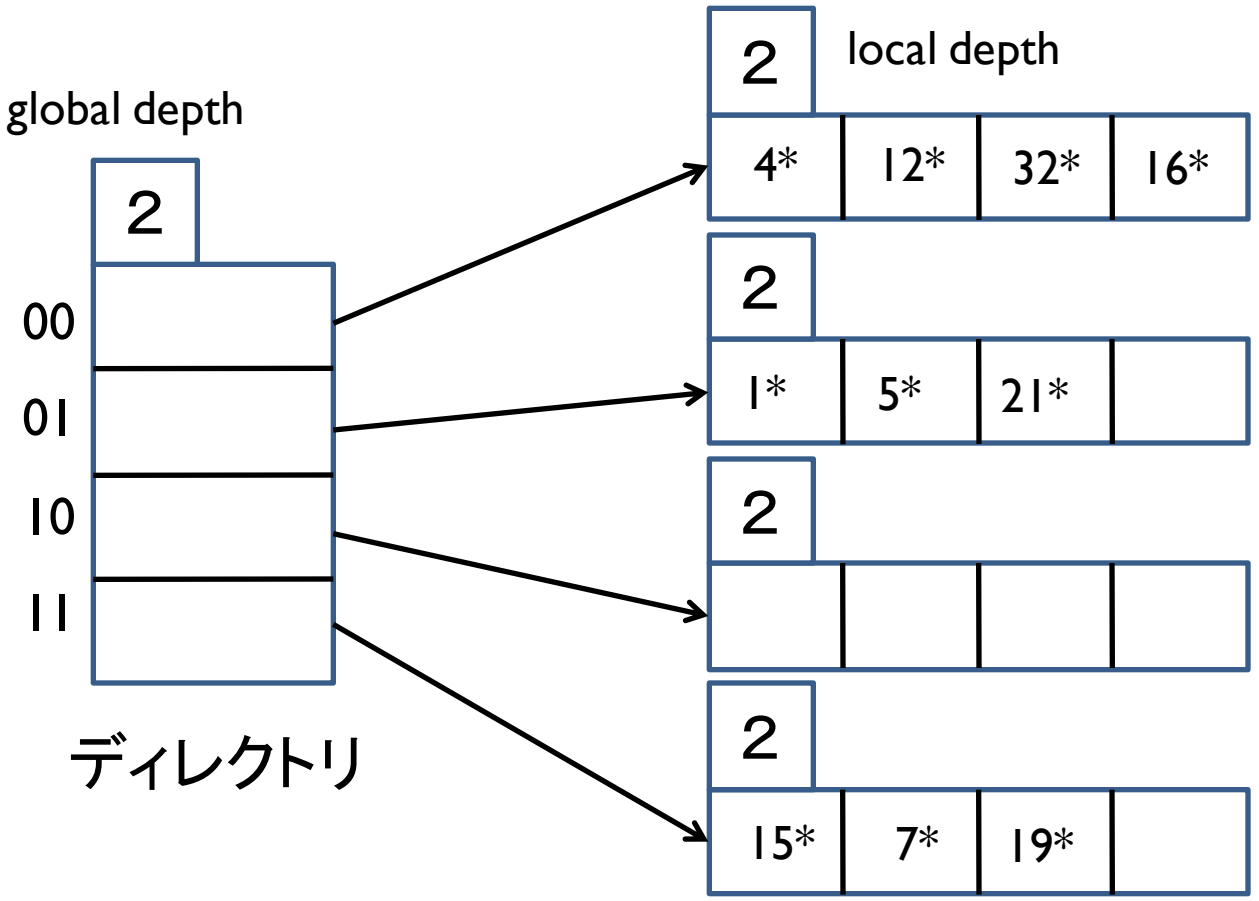
拡張可能ハッシング

- さらに15,7,19を消してみよう



拡張可能ハッシング

- 13を入れてみよう



拡張可能ハッシング

- 利点
  - 分かりやすい
- 問題点
  - バケットのどれか一つがオーバーフローしたらディレクトリの大きさが一気に二倍になる
  - アクセスするページ数が（ディレクトリ分）多くなる
    - ディレクトリ＋バケット