データ構造とアルゴリズム

第６回 辞書とハッシュ

小池 英樹 (*koike@c.titech.ac.jp*)

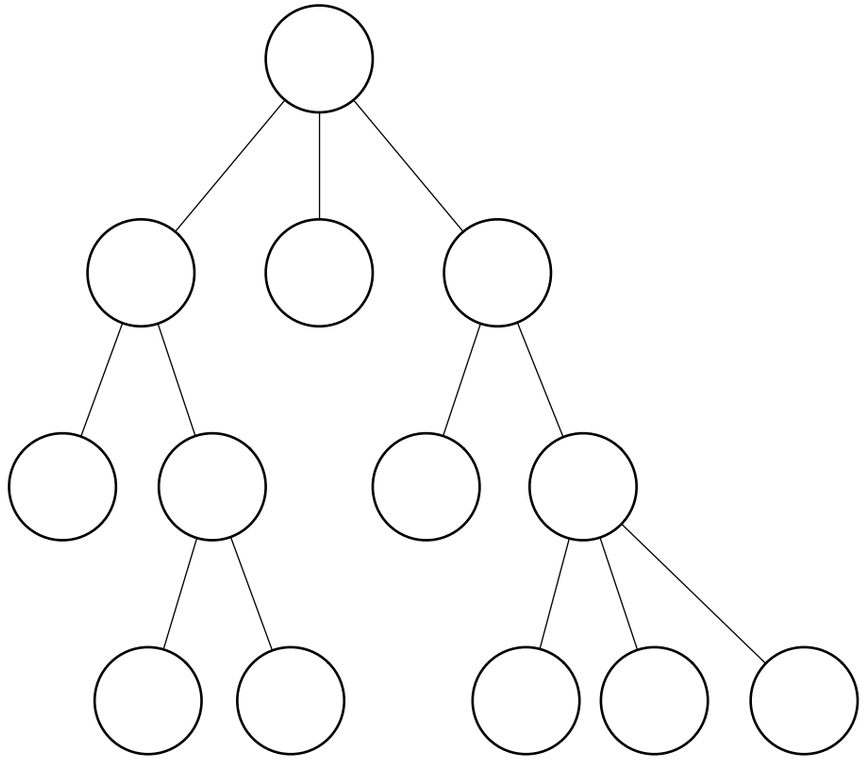
平衡探索木

- 平衡探索木(balanced search tree: BST)
  - 各節点において，その子節点を根とする全ての部分木の高さがほぼ平衡している探索木
- 主な平衡探索木
  - B木：
  - 2-3木：
  - AVL木：
  - ２色木(red-black tree)：

B木

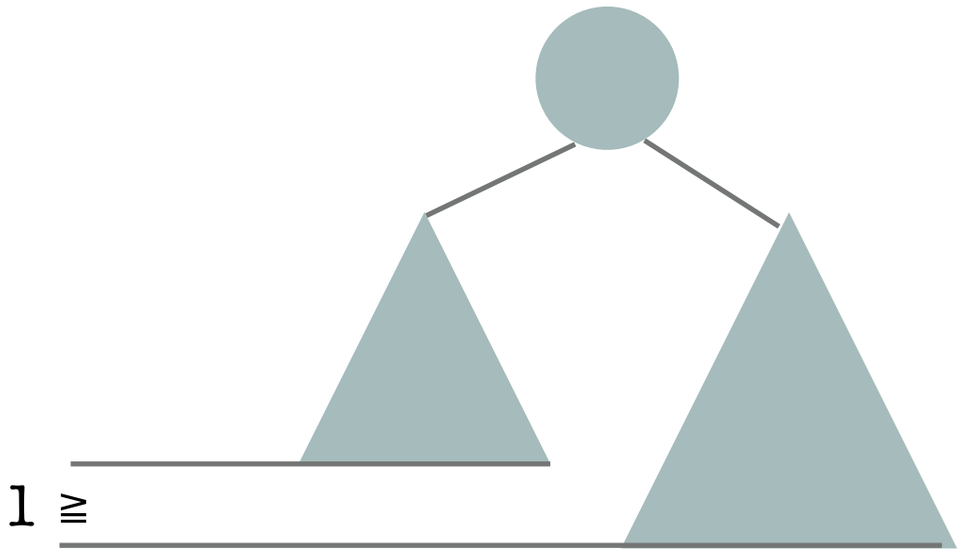
- 根と葉を除く各節点が⌈m/2⌉個以上，m個以下の子を持つ探索木
- m=3の場合のB木を2-3木という

2-3木



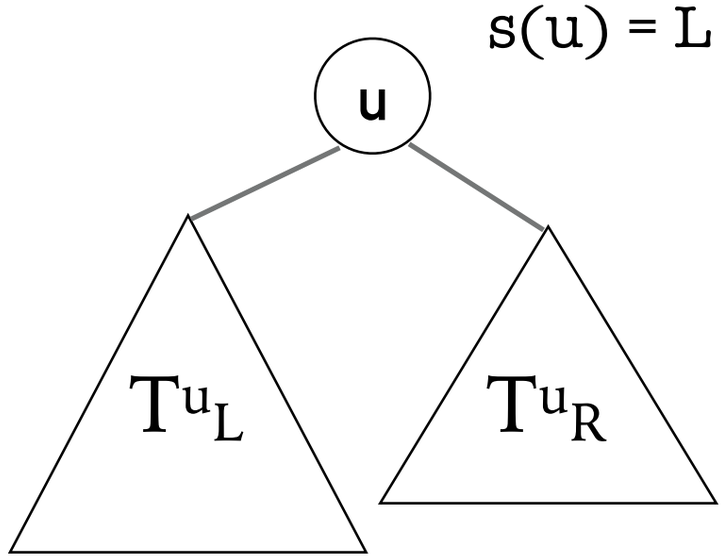
AVL木

- どの節点においても，その左部分木と右部分木の高さの差は1以下である２分探索木
- AVL: G.M. Adel’son-Vel’skii and Y.M. Landis

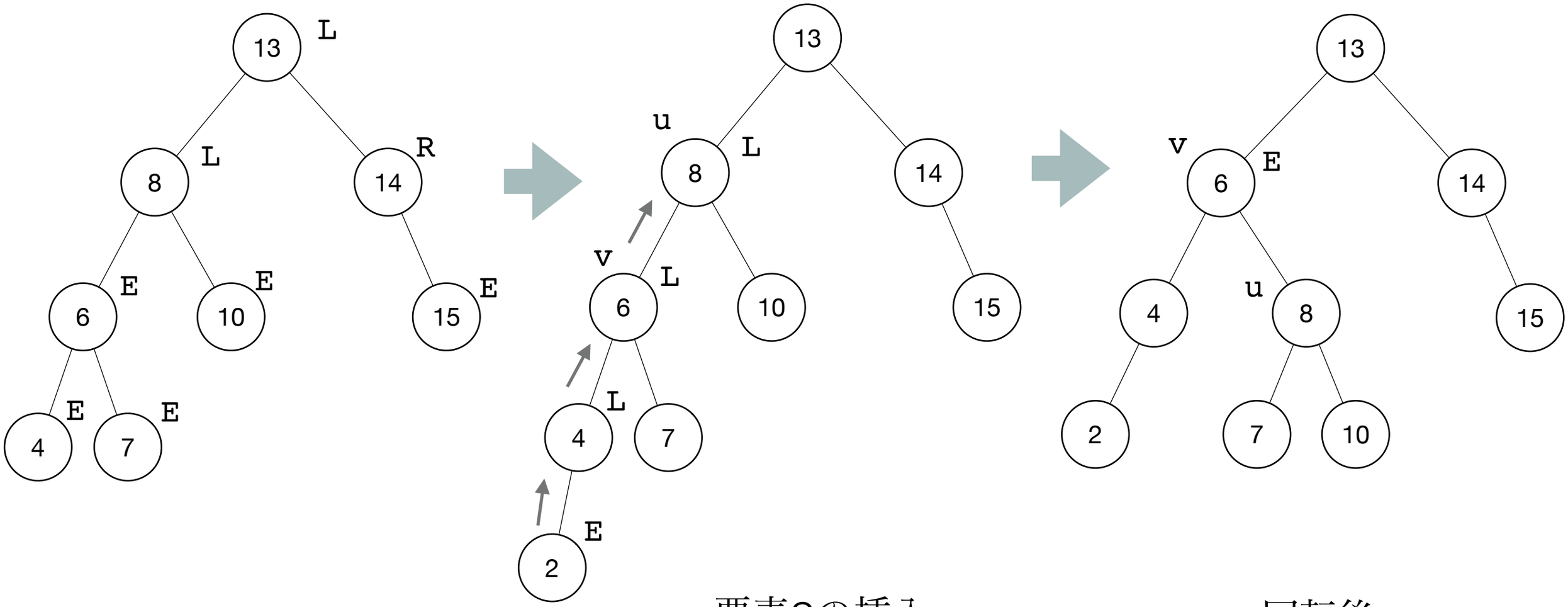


AVL木における操作

- Tu_L: 節点uの左部分木
- Tu_R: 節点uの右部分木
- s(u): 節点uの状態
  - L, if Tu_Lの高さ > Tu_Rの高さ
  - E, if Tu_Lの高さ = Tu_Rの高さ
  - R, if Tu_Lの高さ < Tu_Rの高さ



AVL木のINSERT



要素2の挿入　　回転後

AVL木のINSERT

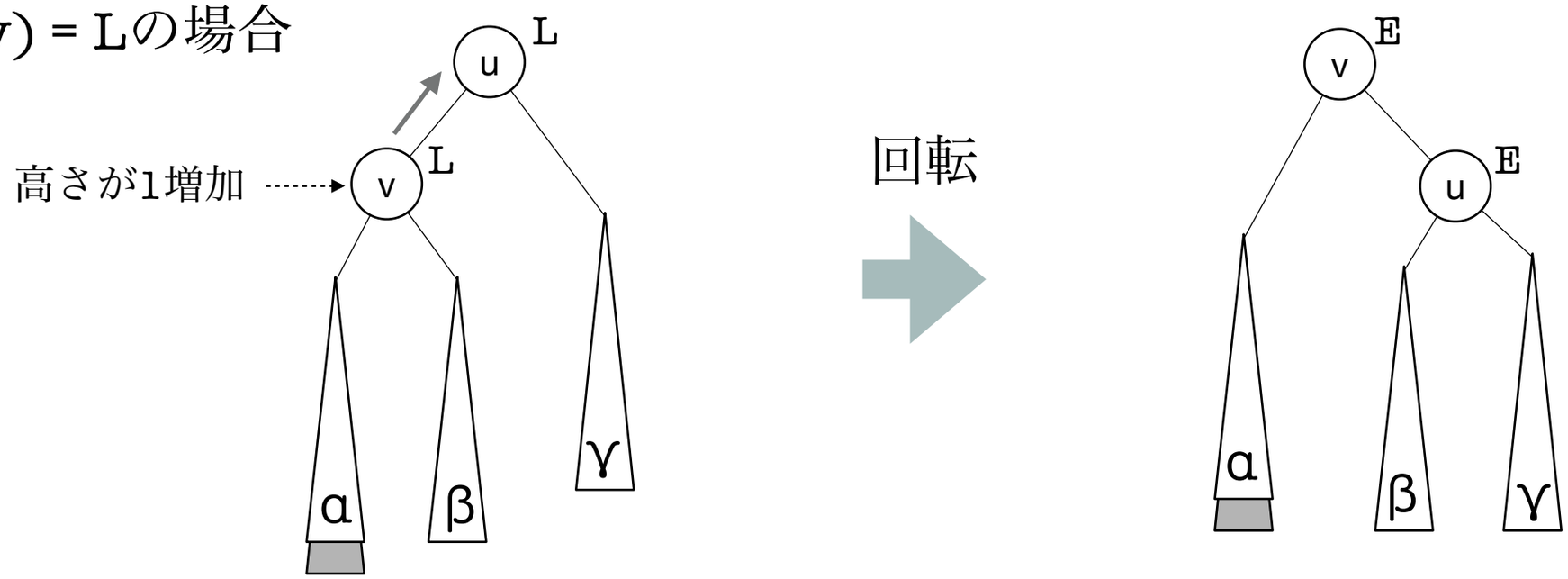
(1) s(u)＝R: 左部分木の高さが 1 増えたためs(u)＝Eに変わる．u を根とする部分木の高さは変化しないので，修正は終了．

(2) s(u) ＝ E: uの新しい状態はs(u)＝L．uを根とする部分木はAVL木であるが，uの高さが1増えているので，uの親へのさかのぼり修正を続行する．uが根なら修正は終了．

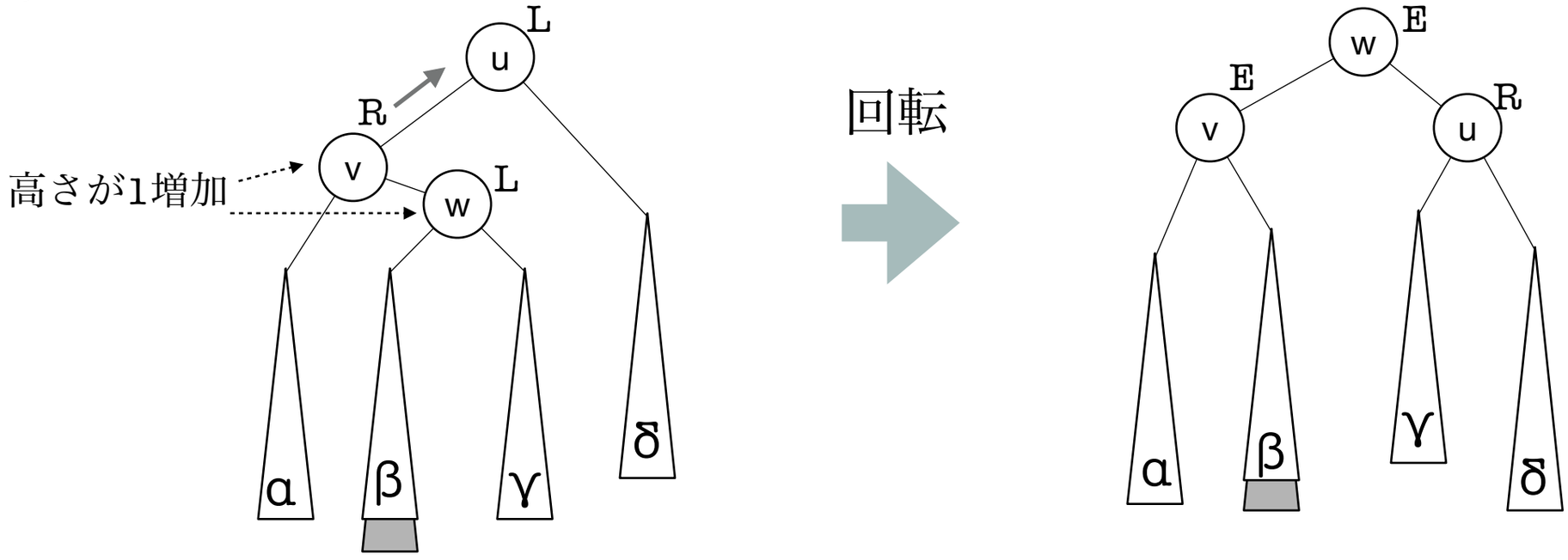
(3) s(u) ＝ L: uはAVL木の条件を満たさなくなるので，左子節点vのs(v)に応じて回転操作を行う．

AVL木のINSERT

s(v) = Lの場合



s(v) = Rの場合



AVL木のDELETE

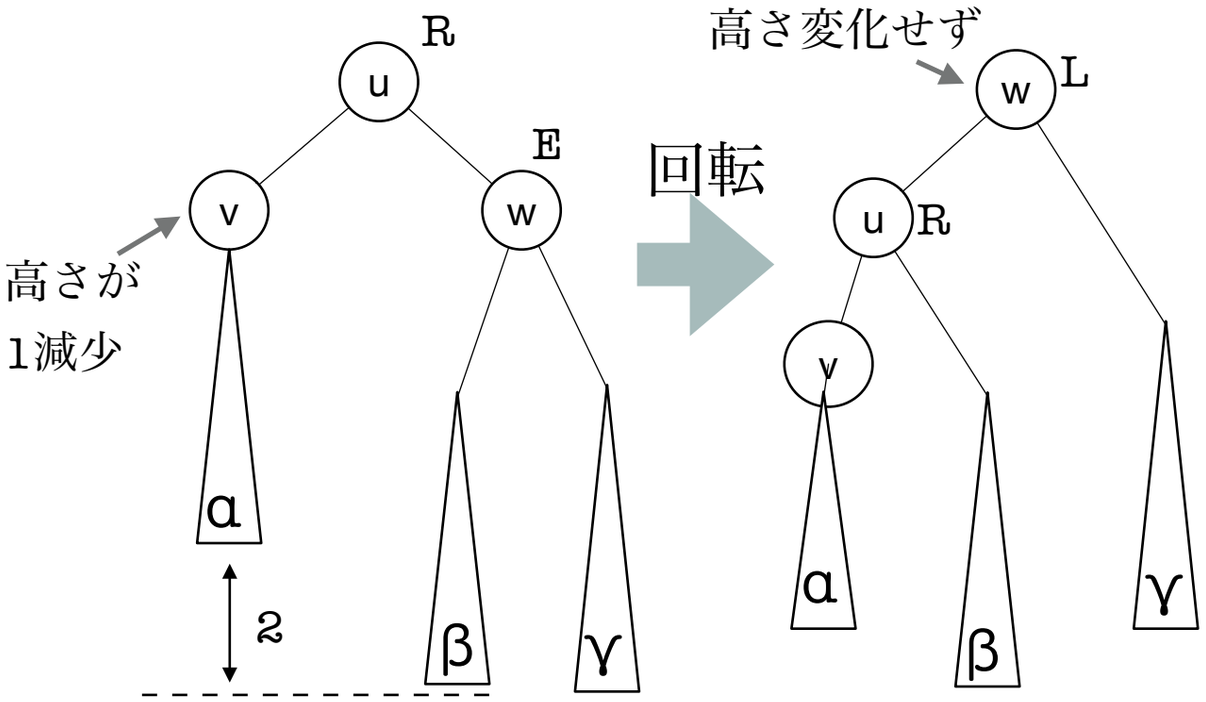
(1) s(u)=L: 左部分木の高さが 1 減ったためs(u)=Eとなる．uの高さは1減少し，修正はuの親へ続ける．uが根ならここで終了.

(2) s(u) = E: uの新しい状態をs(u)=Rとする．uを根とする部分木はAVL木の条件を満たすのでここで終了.

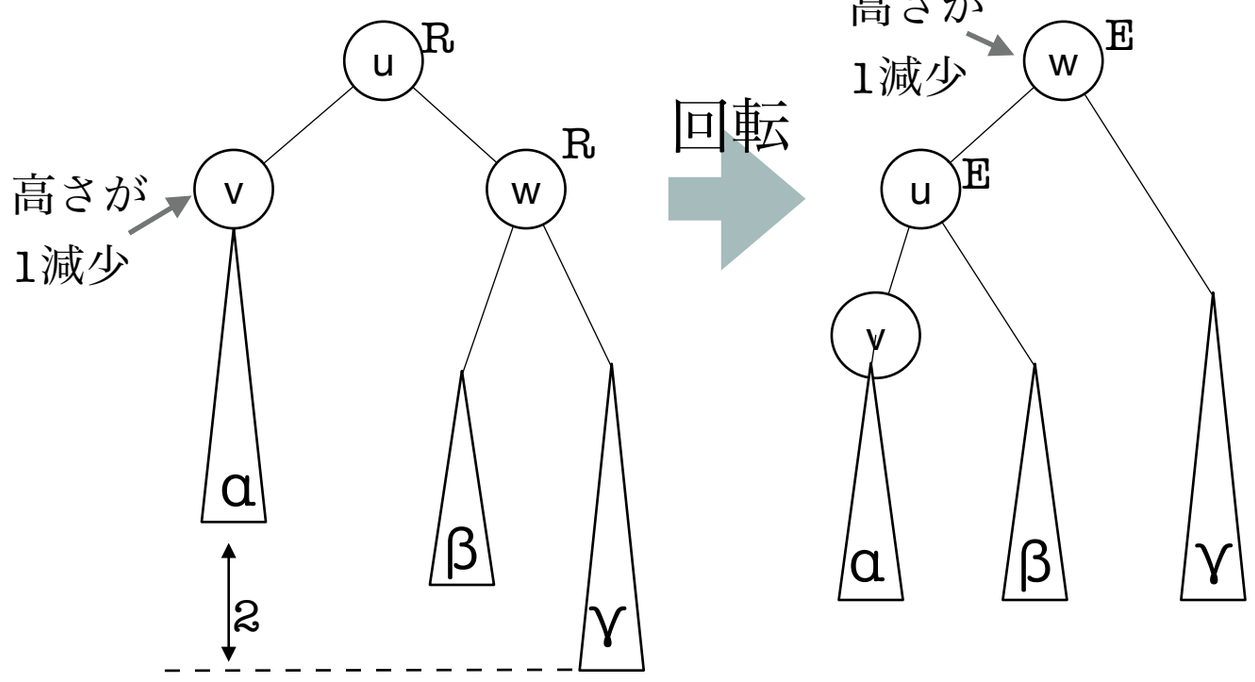
(3) s(u) = R: uはAVL木の条件を満たさなくなるので，右子節点wのs(w)に応じて次スライドの図(a)(b)(c)の回転操作を行う.
(a)の場合，部分木の新しい根wの高さは回転前のuの高さと同一なので修正終了．(b)(c)は高さが減少するので，uの新しい状態をs(u)=Eとして，親へ向かって修正続行.

AVL木のDELETE

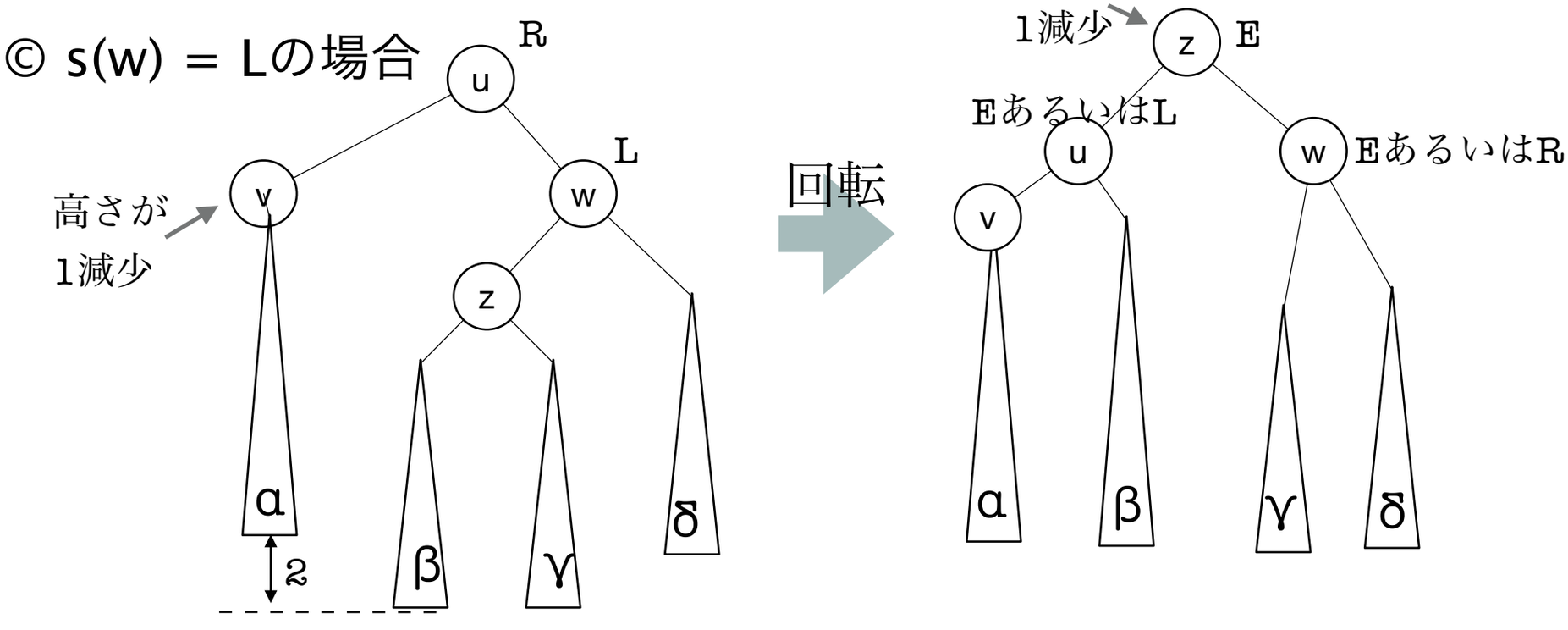
(a) s(w) = Eの場合



(b) s(w) = Rの場合



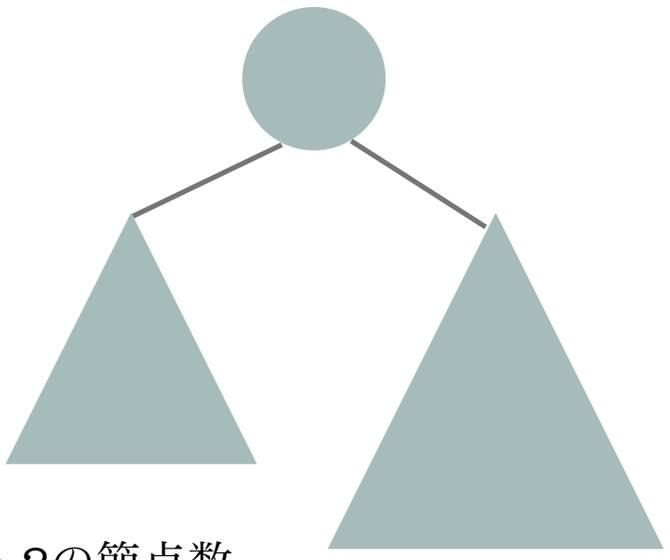
© s(w) = Lの場合



AVL木における計算量

- 最悪 O(log n)
- 平均 O(log n)

高さhの節点数最小の木



高さh-2の節点数
最小の木

高さh-1の節点数
最小の木

（証明）

高さhのAVL木の最小の節点数をf(h)とすると,

$f(h) = f(h-1) + f(h-2) + 1$
$f(0) = 1$
$f(1) = 2$

F(h) = f(h)+1とおくと

$F(h) = F(h-1) + F(h-2)$
$F(0) = 2$
$F(1) = 3$

これはフィボナッチ数列なので

$F(h) = (\varphi^{h+3}{}_1 - \varphi^{h+3}{}_2) / \sqrt{5}$
$\varphi_1 = (1+\sqrt{5}) / 2, \quad \varphi_2 = (1 - \sqrt{5}) / 2$

つまり

$f(h) = (\varphi^{h+3}{}_1 - \varphi^{h+3}{}_2) / \sqrt{5} - 1$

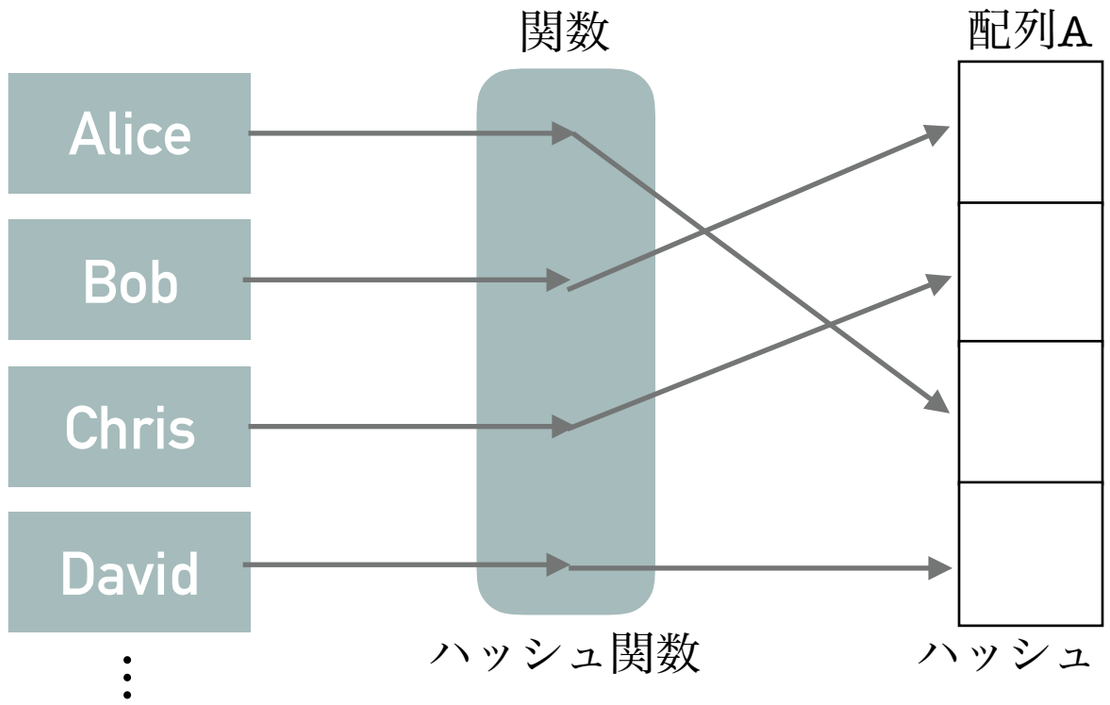
$f(h) \leqq n$ を解いて $h=O(\log n)$ ←自分でやってみること

ハッシュ(HASH)

- 線形探索ではO(n)
- ２分探索は半分ずつ捨てる方策なので，O(log n)が限界.
- より早く探索する方法はないか．．．

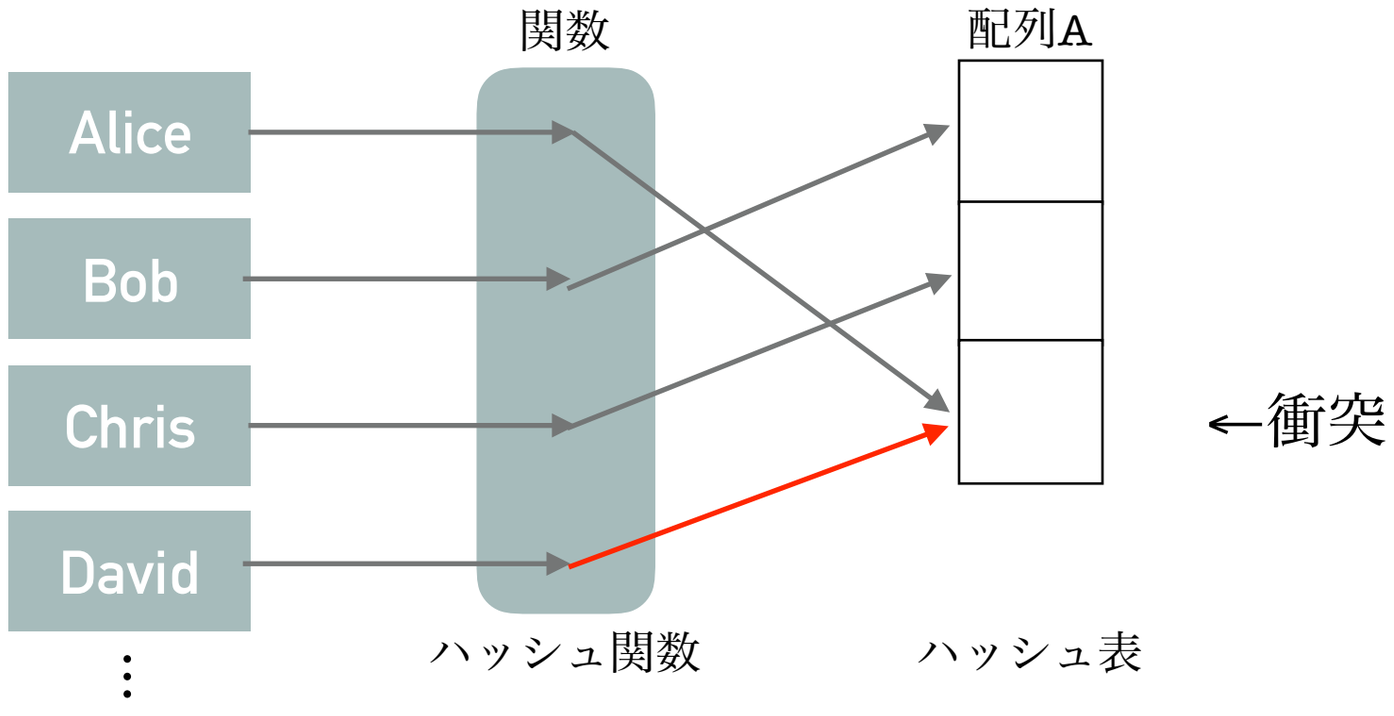
ハッシュの考え方

- 1～100の整数をkeyとするデータにアクセスする場合，最速の探索法は配列A[100]を用意すること．->整数keyにしか使えない．
- keyの値を引数として関数値を計算し，この関数値でデータにアクセスする．-> O(1)
- この関数をハッシュ関数といい，ハッシュ関数値をkeyとするデータ表をハッシュ表（あるは単にハッシュ）という



ハッシュの考え方

- 一般に使用するkey数は全データ数と比較して多くない.
  - 例：プログラム内における変数の名前(i, j, x, p, tmp, …)
    - 英単語全体に比べてわずか(せいぜい100〜1000個程度)
- データテーブル（辞書）を小さくして，keyを管理する
- 問題点：異なるkeyでもハッシュ関数の値が同じになる可能性がある -> 衝突(collision)



UNIFORM HASHING ASSUMPTION

- Uniform hashing assumption
  - 各keyは0〜M-1の間に平等に分散される

例：ある小説に出現する単語のハッシュ値の分散

ハッシュ：衝突の回避

- 外部ハッシュ法 (open hashing, chaining)
- 内部ハッシュ法 (closed hashing, open addressing)

ハッシュ関数(HASH FUNCTION)

- 例：$x = a_1a_2...a_6$ (各a_iはアルファベットの 1 文字）に対し

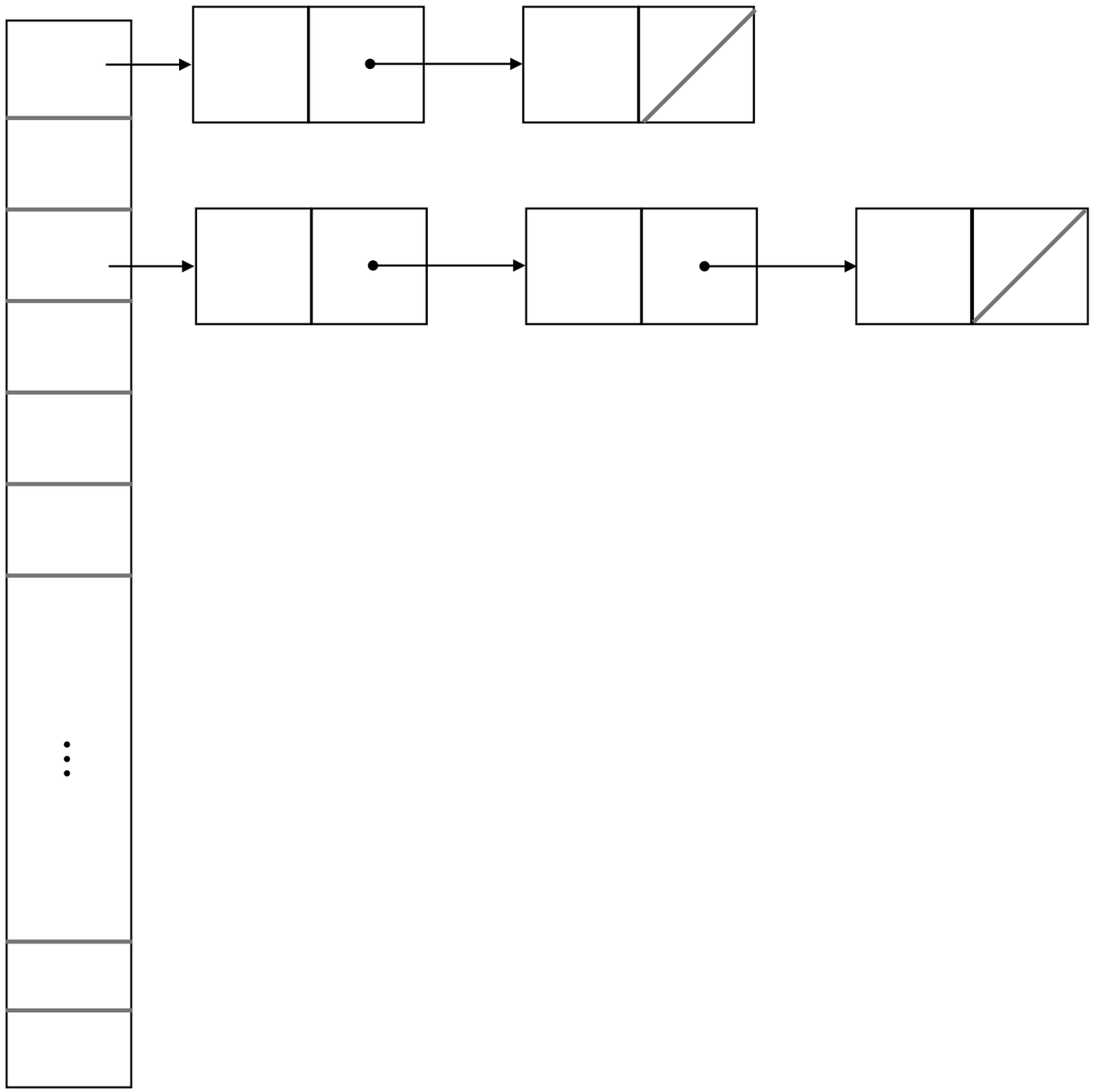
$$h(x) = \sum^{6} \mathrm{ord}(a_i) \pmod{B}$$

ただし，ord(a)はaの整数コード(ASCII, JIS...)

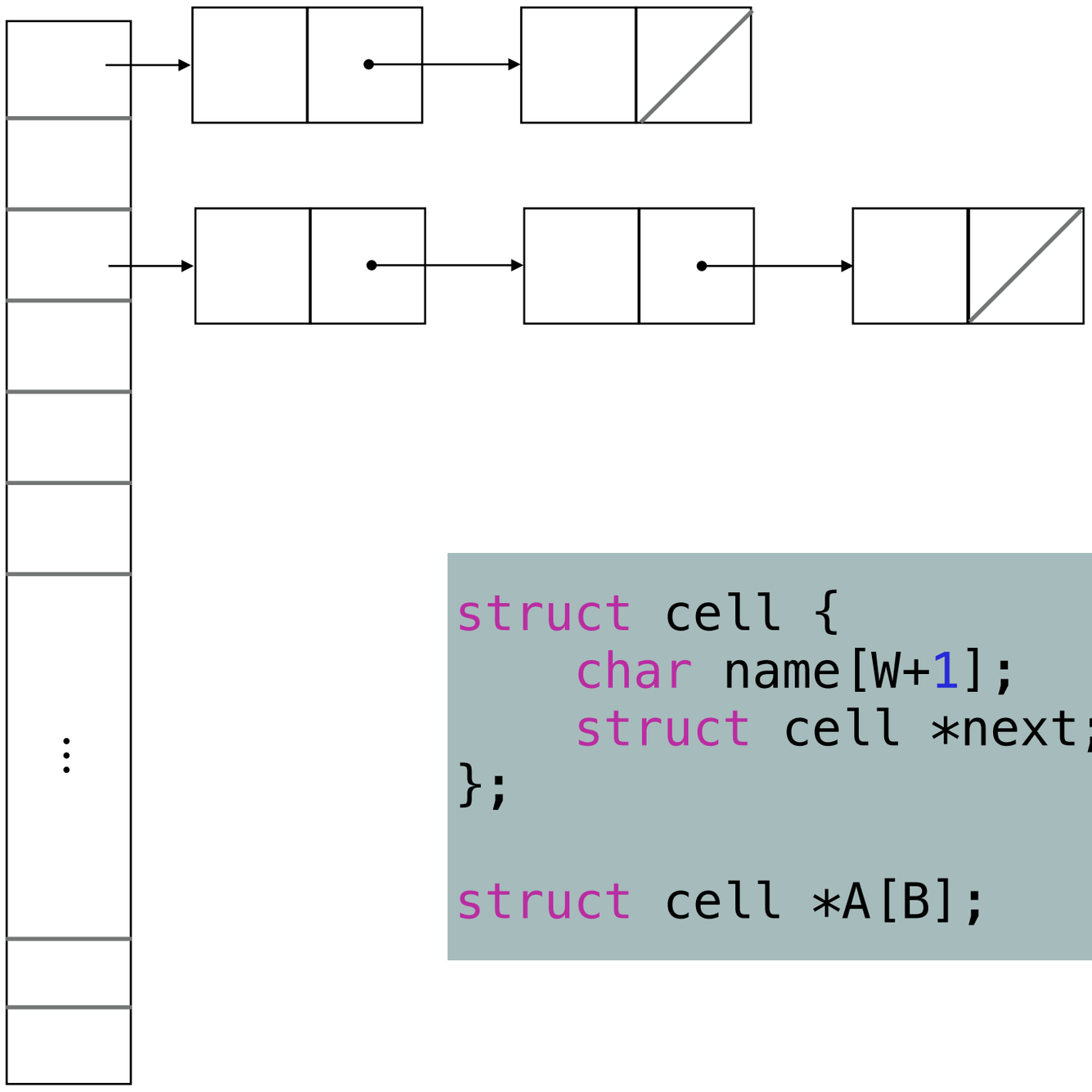
```
int h(char *x) {
    int i, hash;

    hash = i = 0;
    while(x[i] != 0 && i<W) {
        hash = hash + (int)x[i];
        i =i+1;
    }
    hash = hash % B;
    return(hash);
}
```

外部ハッシュ法：データ構造

- 外部ハッシュ法
- ハッシュ関数で添字を計算し，要素がなければ新たなセルを作成，要素がある場合は，リストに挿入

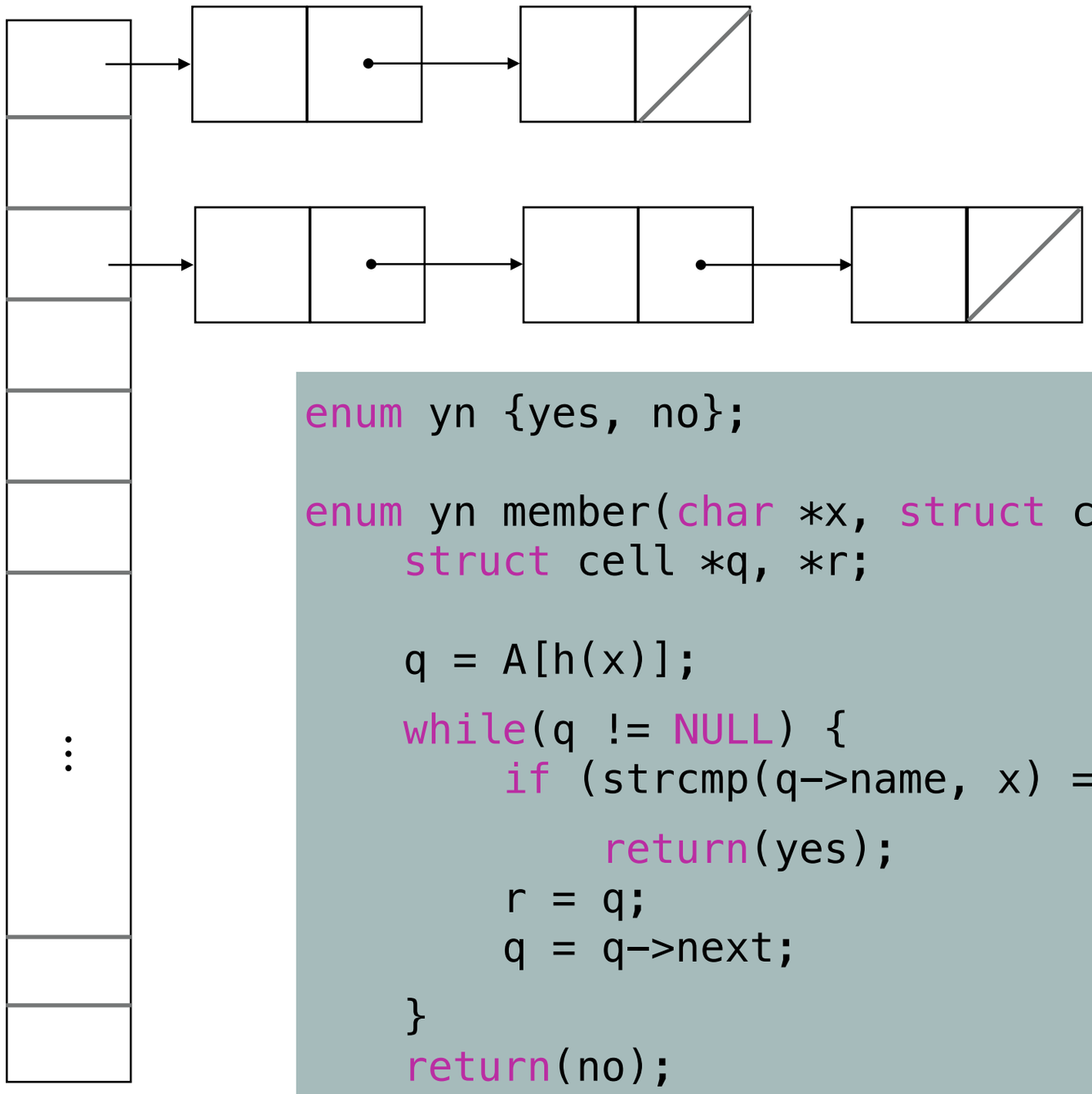


外部ハッシュ法：データ構造



```
struct cell {
    char name[W+1];
    struct cell *next;
};

struct cell *A[B];
```

外部ハッシュ法：探索



```
enum yn {yes, no};

enum yn member(char *x, struct cell **A) {
    struct cell *q, *r;

    q = A[h(x)];                          /* h(x)内でxの探索 */
    while(q != NULL) {
        if (strcmp(q->name, x) == 0) /* xの発見 */
            return(yes);
        r = q;
        q = q->next;                      /* 次へ */
    }
    return(no);                          /* xは存在しない */
}
```

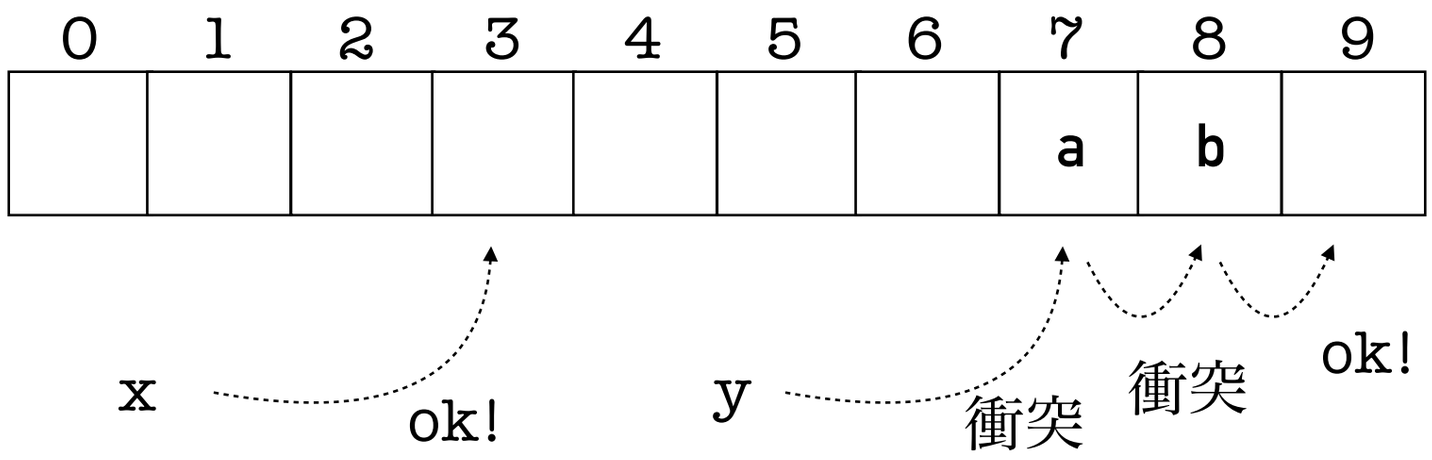
外部ハッシュ法：挿入

```
void insert(char *x, struct cell **A) {
    int k;
    struct cell *p, *q, *r;

    k = h(x);                                          /* xの入るパケット番号 */
    q = A[k];                                          /* パケットk内の探索 */
    p = (struct cell *)malloc(sizeof(struct cell)); /* 新しいポインタの獲得 */
    if (q == NULL) {
        A[k] = p;
    } else {
        while (q != NULL) {                            /* xの存在のチェック */
            if (strcmp(q->name, x) == 0) {             /* xは既に存在 */
                free(p);
                return;
            } else {                                   /* 次へ */
                r = q;
                q = q->next;
            }
        }
        r->next = p;
    }
    strcpy(p->name, x);                                /* xの挿入 */
    p->next = NULL;
    return;
}
```

内部ハッシュ法：データ構造

- 内部ハッシュ法
- ハッシュ関数で添字を計算し，要素がなければその位置にデータを格納，要素がある場合は 1 つずつ添字をずらしていく．

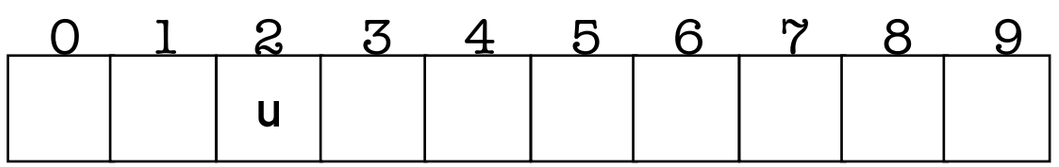
$$h(x) \equiv h(x) + i \pmod{B}$$



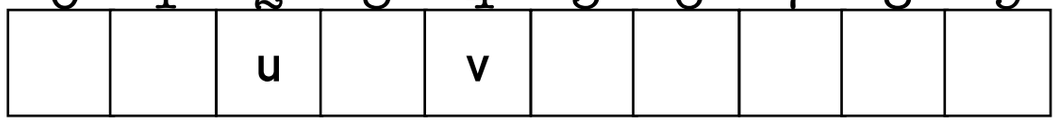
内部ハッシュ法：データ構造

- 例：h(u)＝2, h(v)＝4, h(w)＝0, h(x)＝2, h(y)＝3, h(z)＝8のとき， u, v, …, zの順にデータを挿入した結果

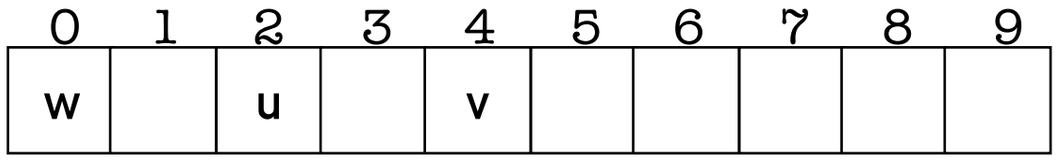
uを挿入　　h(u)=2



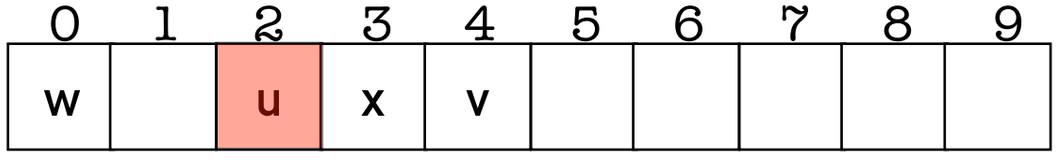
vを挿入　　h(v)=4



wを挿入　　h(w)=0

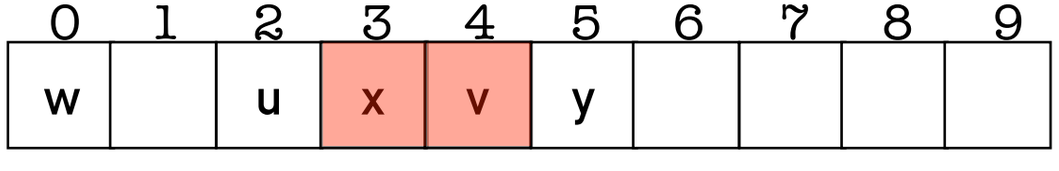


xを挿入　　h(x)=2



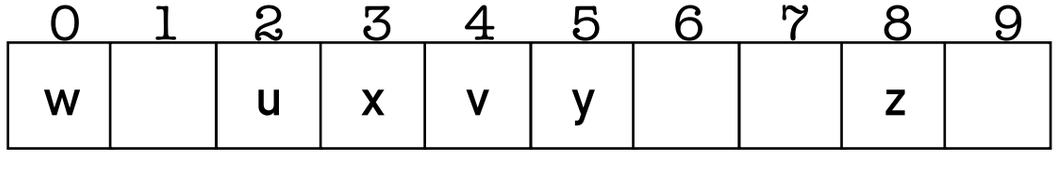
<-衝突

yを挿入　　h(y)=3



<-衝突

zを挿入　　h(z)=8



内部ハッシュ法：データ構造

```
enum oed {occupied, empty, deleted};

struct word {
    char name[W+1];
    enum oed;
};

struct word A[B];
```

内部ハッシュ法：探索

```
enum yn member(char *x, struct word *A) {
    int i, k;
    enum oed cstate;

    k = i = h(x);
    do {
        cstate = A[k].state;
        if (ctate == occupied) {
            if (strcmp(x, A[k].name) == 0)  /* xの発見 */
                return(yes);
        }
        k = (k+1) % B;                          /* 次のセルへ */
    } while (cstate != empty && k!=i);
    return(no);                                 /* xは存在しない */
}
```

内部ハッシュ法：挿入

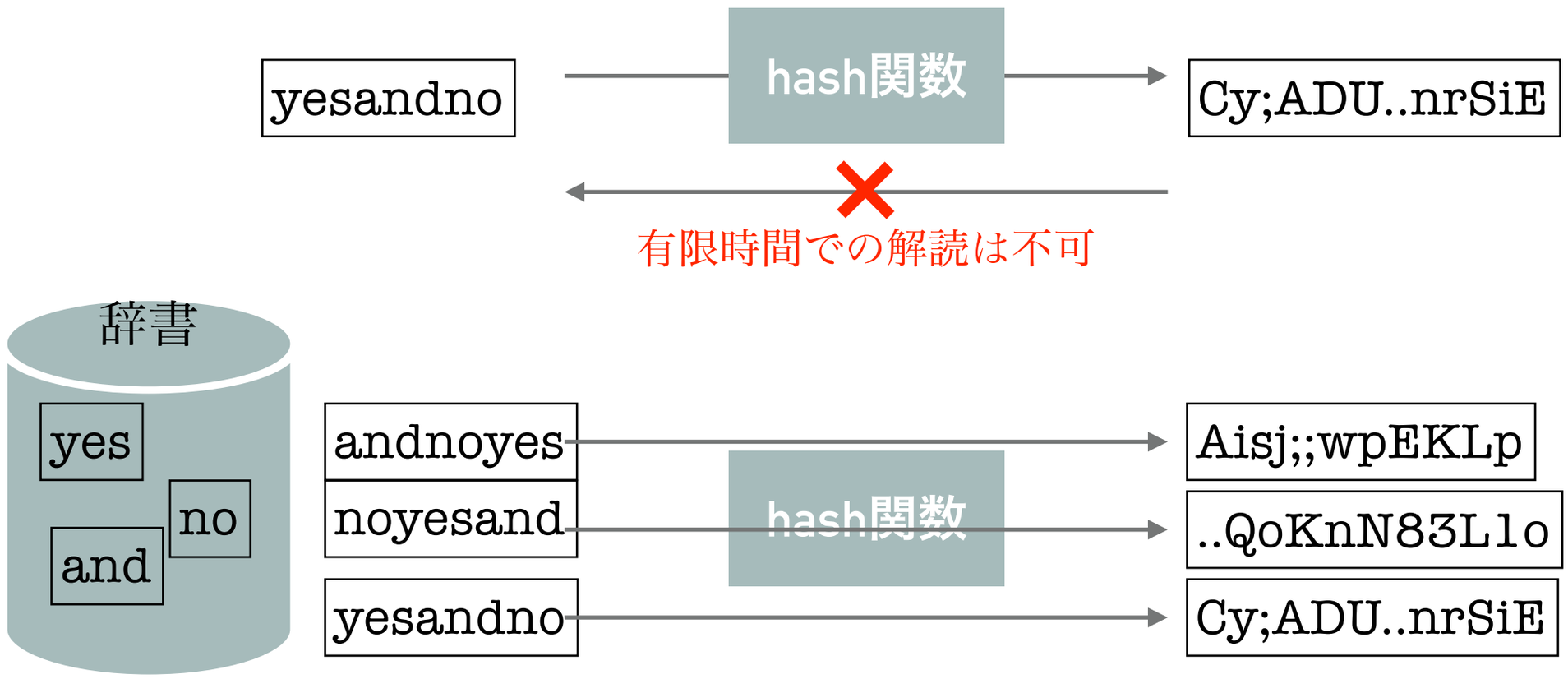
```
void insert(char *x, struct word *A) {
    int i, k, found = -1;
    enum oed cstate;

    k = i = h(x);
    do {
        cstate = A[k].state;
        if (cstate == empty || cstate == deleted) {
            if (found<0)
                found = k;
        } else {
            if (strcmp(x, A[k].name) == 0) /* xは既に存在 */
                return;
        }
        k = (k+1)%B;                       /* 次のセルへ */
    } while (cstate != empty && k!=i);
    if (found < 0) {                       /* Aは満杯 */
        printf("Error: Dictionary is full.\n");
        eixt(1);
    }
    strcpy(A[found].name, x);              /* A[found]へxの挿入 */
    A[found].state = occupied;
    return;
}
```

ハッシュの応用例

- 暗号，改ざん検知
  - 文字列を入力し，暗号文を出力する
    - 例：コンピュータのパスワードデータベース等
  - 逆方向への変換が難しい
  - ところが，，，辞書攻撃(brute-force attack)

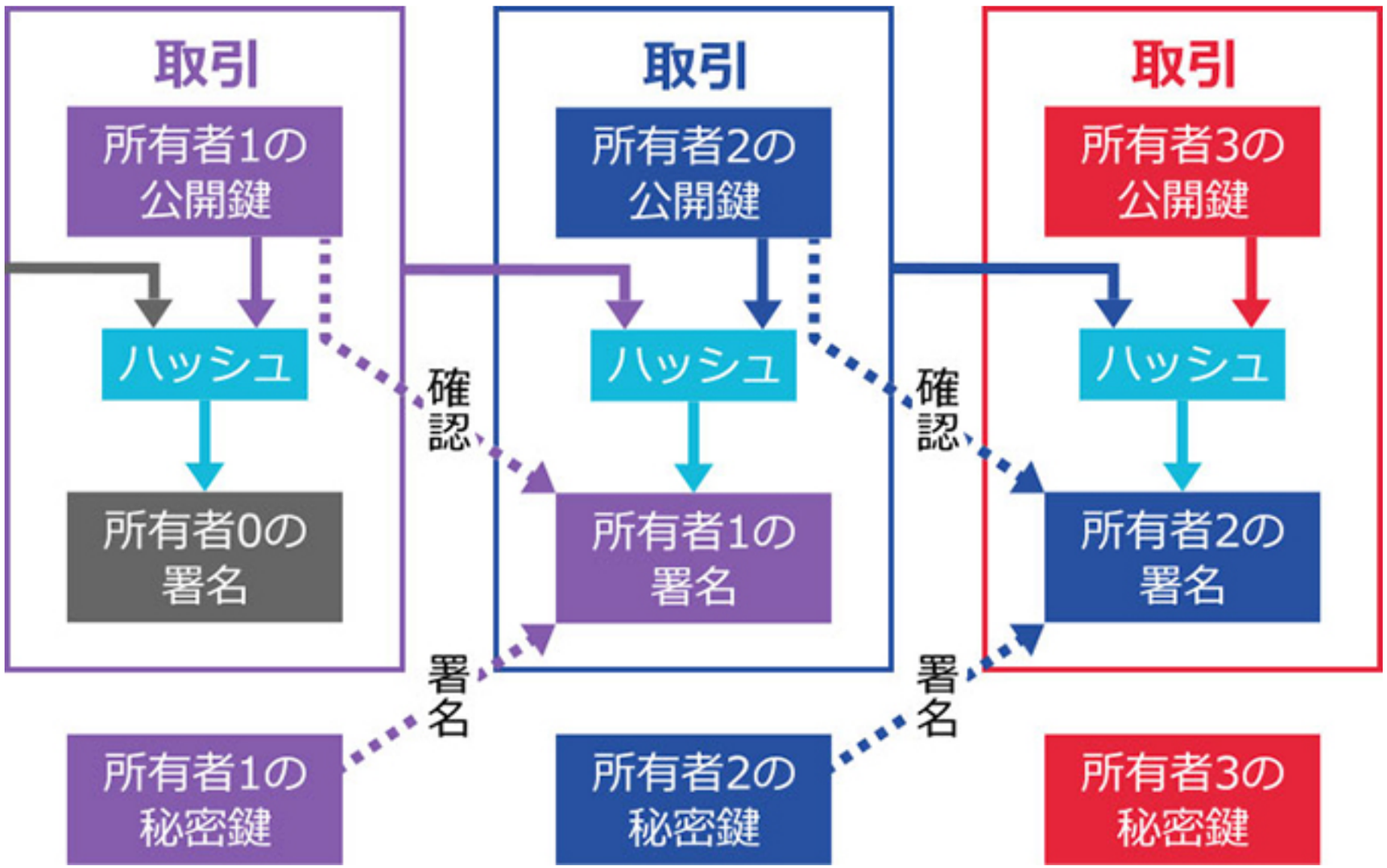
ハッシュの応用例

- 暗号，改ざん検知
  - 文字列を入力し，暗号文を出力する
    - 例：コンピュータのパスワードデータベース等
  - 逆方向への変換が難しい
  - ところが，，，辞書攻撃(brute-force attack)



ハッシュの応用例

➤ ビットコイン



http://orekabu.jp/bitcoin-structure/

比較

	線形探索	２分探索	２分探索木	ハッシュ
探索	O(n)	O(log n)	O(log n) (ただし最悪O(n))	O(1) (ただし最悪O(n))
挿入	O(1)	O(n)	O(log n)	O(1) (ただし最悪O(n))
削除	O(n)	O(n)	O(log n)	O(1) (ただし最悪O(n))

告知

- 1/9(火)は中間試験
- 範囲は今日の分まで.
- 教科書，ノート，プリント等の持ち込み不可.