ハッシュ

世木寛之

本日お話する内容

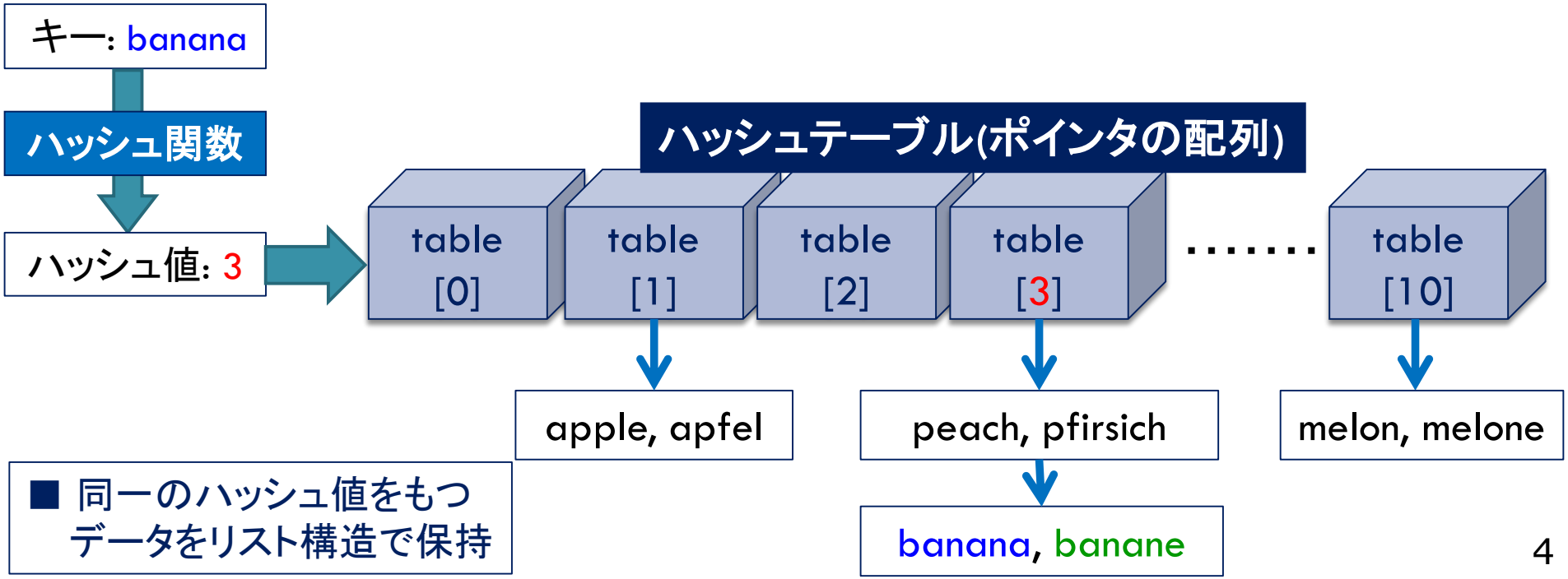
1. ハッシュについて

文字列から自然数への変換方法

apple、banana、peachなどの単語の頻度をカウントしたい場合、
これまでのようにリスト構造や2分木なども使えるが、
apple:0、banana:1、peach:2と番号をふっておき、
該当する単語がでてきたときに、hindo[0]++、hindo[1]++、hindo[2]++
などと頻度値を更新することもできる。
このとき、例えば、peachがどの番号かを探索する際に、
“単語と番号を記した表”が長いと、探索に時間がかかってしまうという問題がある。
2分木を構築しておけば、要素数をnとすると、$\log_2 n$のオーダーで探索できるが、
もっと速く探索したい場合には、オーダー1で探索できるハッシュを構築すれば良い。
このため、本講義では、ハッシュのデータ構造について説明する。
(STLのunordered_mapがハッシュに相当するが、
hindo["apple"]++、hindo["banana"]++、hindo["peach"]++のように記述することが可能)

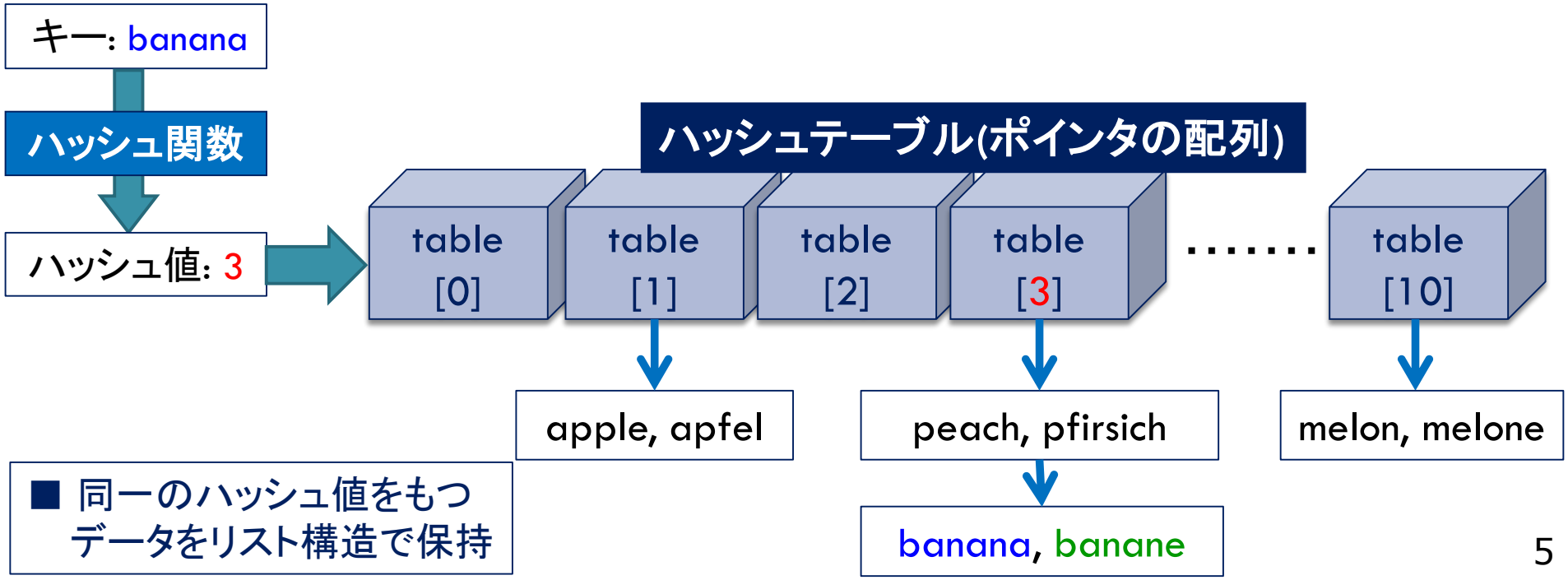
チェイン法によるハッシュのしくみ①

キーに対応する値(バリュー)を、理屈上、オーダー1で呼び出すことができる。
例えば、下記のようにbananaとbananeを対でハッシュテーブルの中に保持しておくと、
キーbanana(英語のバナナ)が与えられたときに、
バリューbanane(ドイツ語のバナナ)を取り出すことができる。



チェイン法によるハッシュのしくみ②

bananaとbananeのデータは、ハッシュ関数を用いて、
キーを整数に変換したハッシュ値(0からハッシュテーブルの大きさ-1までをとる整数)
の添え字(インデックス)のハッシュテーブルに保持しておく。
例えば、キーbananaをあるハッシュ関数の引数として渡して、3が返ってきたとすると、
bananaとbananeのデータは、ハッシュテーブルの3のインデックスのところに
保持しておく。

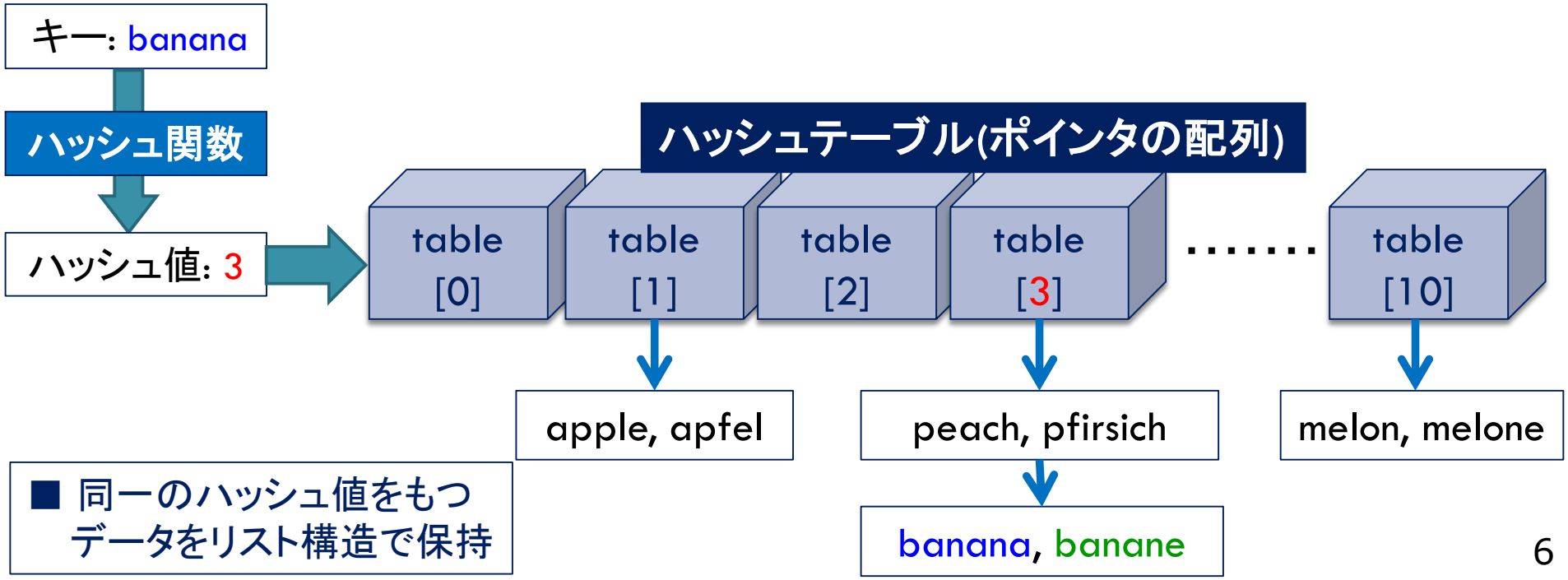


チェイン法によるハッシュのしくみ③

このため、ハッシュ関数は、キーからなるべく偏らないハッシュ値を生成できることが望ましい。例えば、いつもハッシュ値として3を出すようなハッシュ関数を用いると、単なるリスト構造と変わらなくなってしまう。

また、多くの場合、ハッシュテーブルの大きさは素数でとる。

これは、ハッシュ関数内で数値を計算した最後に素数で割った余りをとることにより、0から素数-1までの整数値に割り振ることができるためである。



ハッシュ関数

先ほども述べたように、ハッシュ関数は、キーからなるべく偏らないハッシュ値を生成できることが望ましい。このため、いろいろなハッシュ関数が存在するが、今回は、キー(=key)となる文字列の1文字ごとの文字コード（アスキーコード）を加算し、ハッシュテーブルの大きさ(=size)で割った余りをハッシュ値として返すハッシュ関数を用いる。

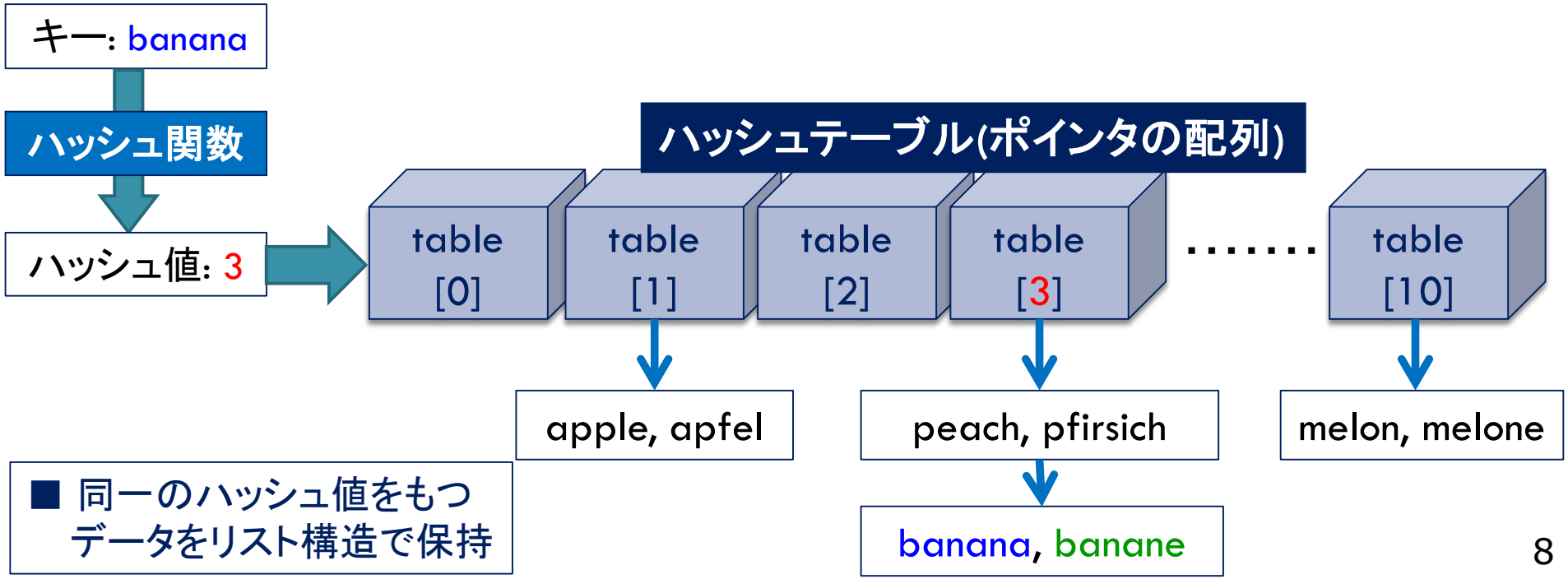
```
// ハッシュ値の取得(ハッシュ関数）
int Hash::get_hashvalue(string key){
    unsigned int v=0;
    for(unsigned int i=0; i<key.length(); i++){
        v += key[i];            //1文字ごとの文字コードを加算
    }
    return v%size;              //ハッシュテーブルの大きさで割った余りを返す
}
```

※このように除算を用いたハッシュ関数の実装においては、
素数で除算を行うハッシュ関数を用いると衝突が少なくなることが分かっていることから、
ハッシュテーブルのサイズ(上記のsize)は、必要なサイズより大きい直近の素数とするとよい。

ハッシュテーブル①

```
class Hash{
private:
    class Node {
    public:
        string key;          // キー
        string value;        // 値(バリュー)
        Node *next;          // 後続ノードへのポインタ(リスト構造)
        Node(){}
        Node(string k, string v, Node *n=NULL) {
            key=k; value=v; next=n;
        }
    };
    int size;                // ハッシュテーブルの大きさ
    Node **table;            // ハッシュ値が同じノードの先頭アドレス
    int get_hashvalue(string);// ハッシュ値の取得(ハッシュ関数)
```

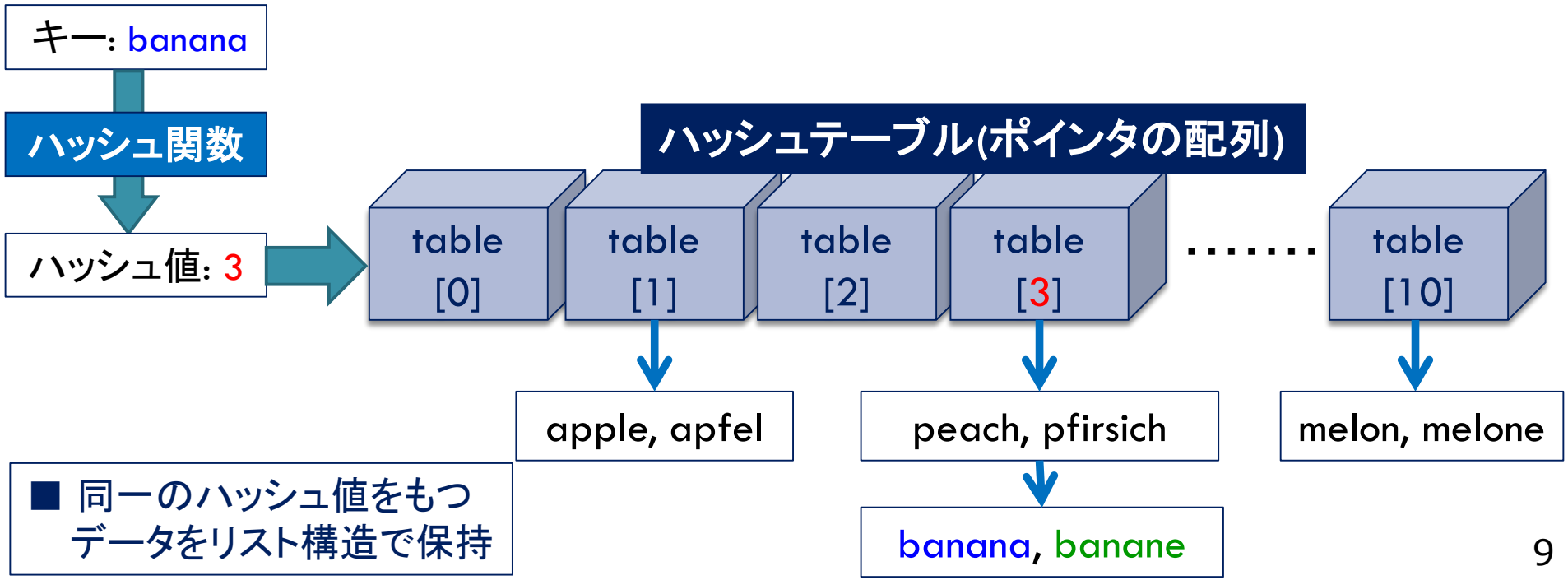
```
(左下からの続き)
public:
    Hash(int tablesize);
    Node* find(string key);
    string find_value(Node *findnode);
    string find_key(Node *findnode);
    int insert(string key, string value);
    void print();
};
```



■ 同一のハッシュ値をもつデータをリスト構造で保持

ハッシュテーブル②

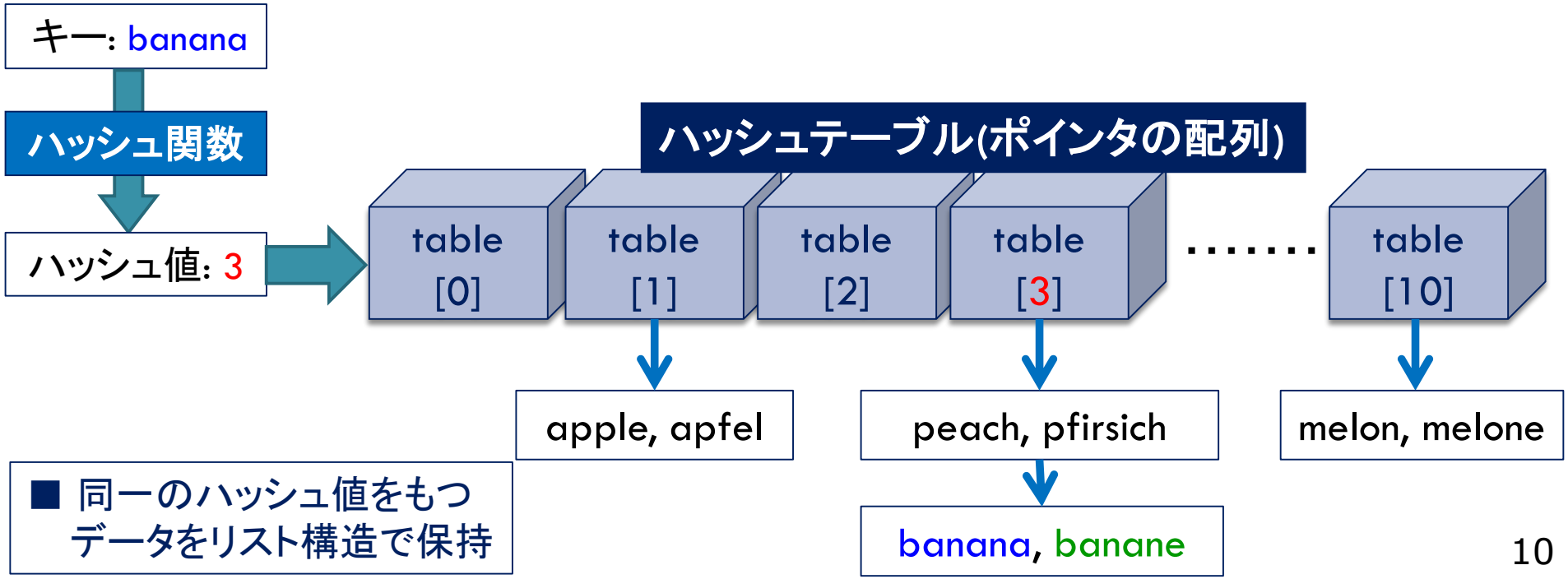
```
// コンストラクタ(ハッシュテーブルのメモリ確保と初期化)
Hash::Hash(int tablesize){
    table=(Node **) new Node[tablesize];//引数で指定された大きさのハッシュテーブルのメモリを確保
    size=tablesize;
    for(int i=0; i<tablesize; i++){
        table[i]=NULL;                          //NULLで初期化しておく
    }
}
```



■ 同一のハッシュ値をもつデータをリスト構造で保持

ハッシュテーブル③

```
// keyを保持するノードのポインタを戻す
Hash::Node* Hash::find(string key){
    int hashvalue=get_hashvalue(key);          //ハッシュ値の取得
    Node *p=table[hashvalue];//ハッシュテーブルに格納されたリストの先頭ノードのポインタを取得
    if(p==NULL){                               //NULLの場合
        return NULL;                           //NULLを返す(keyは登録されていない)
    }
    while(p !=NULL){                           //pがリストの末端まで行ききっていない場合
        if(p->key==key){                       //pが指しているノードのkeyが引数と一致していれば
            return p;                          //pを返す(keyを保持するノードを指すポインタがp)
        }
        p=p->next;                             //次のノードを調べる
    }
    return NULL;                               //リストにkeyが存在しないため、keyは登録されていない
}
```

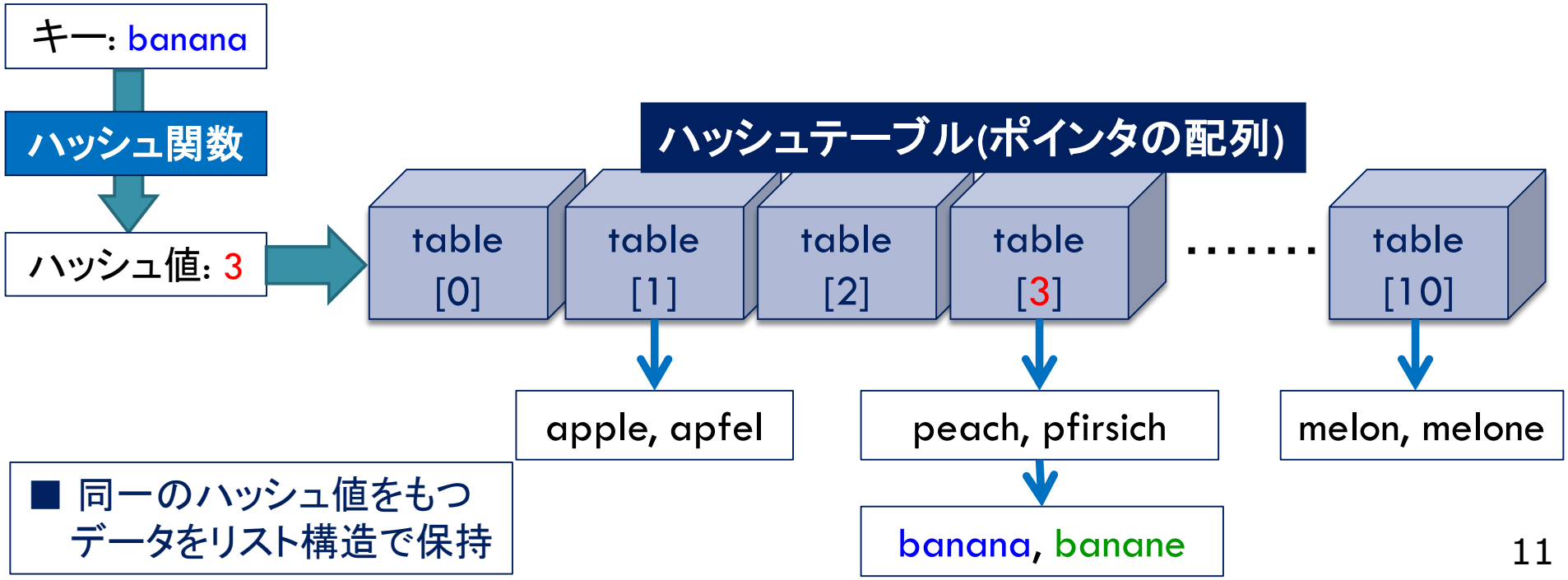


■ 同一のハッシュ値をもつデータをリスト構造で保持

ハッシュテーブル④

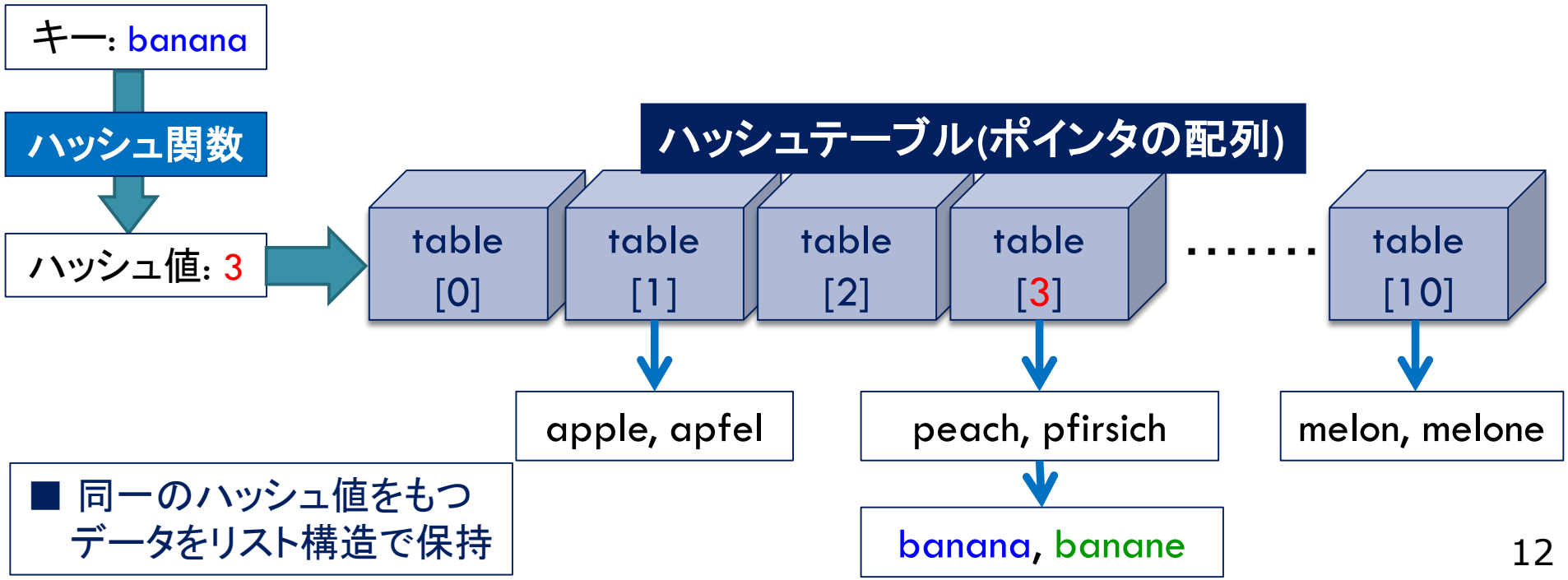
```
// key値に対応したnodepointerを用いてvalueを戻す
string Hash::find_value(Node *nodepointer){
    return nodepointer->value;
}

// nodepointerのkeyを戻す
string Hash::find_key(Node *nodepointer){
    return nodepointer->key;
}
```



ハッシュテーブルの出力

```
void Hash::print(){
    cout << "Print Hash Table\n";
    for(int i=0; i<size; i++){          //table[0]からtable[size-1]が出力対象
        Node *p=table[i];               //リストの大元のノードのポインタを取得
        cout << i;
        while (p != NULL) {             //pがNULLでない場合
            cout << "-> (" << p->key << "," << p->value << ") ";
            p = p->next;                //次のノードを調べる
        }
        cout << "\n";
    }
}
```



■ 同一のハッシュ値をもつデータをリスト構造で保持

本日はここまでです
お疲れさまでした

質問があれば、下記までお願いします
11号館2階1212号室