オペレーティングシステム

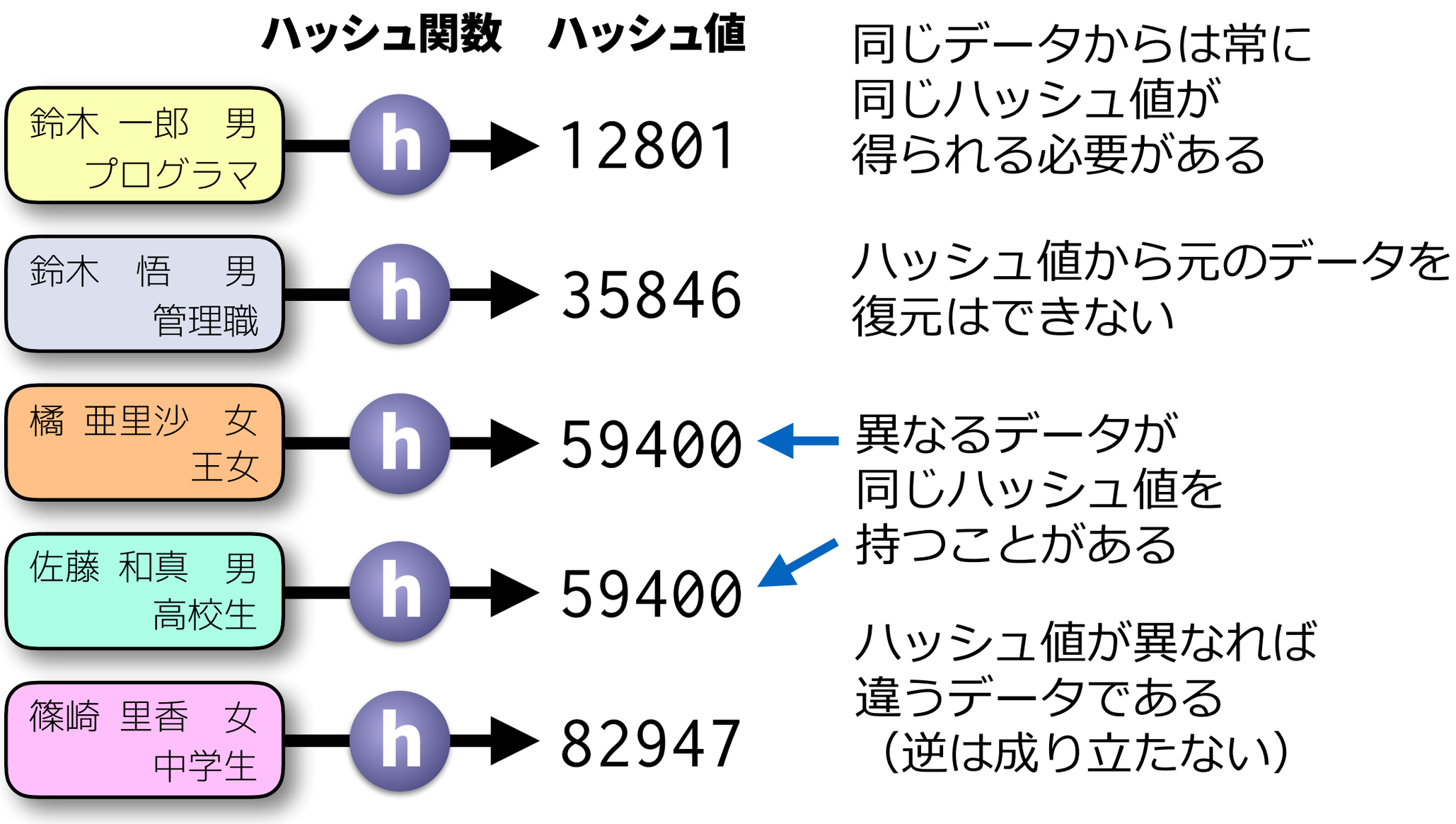
追加資料. ハッシング

令和元年度 秋学期

担当: 荻原剛志

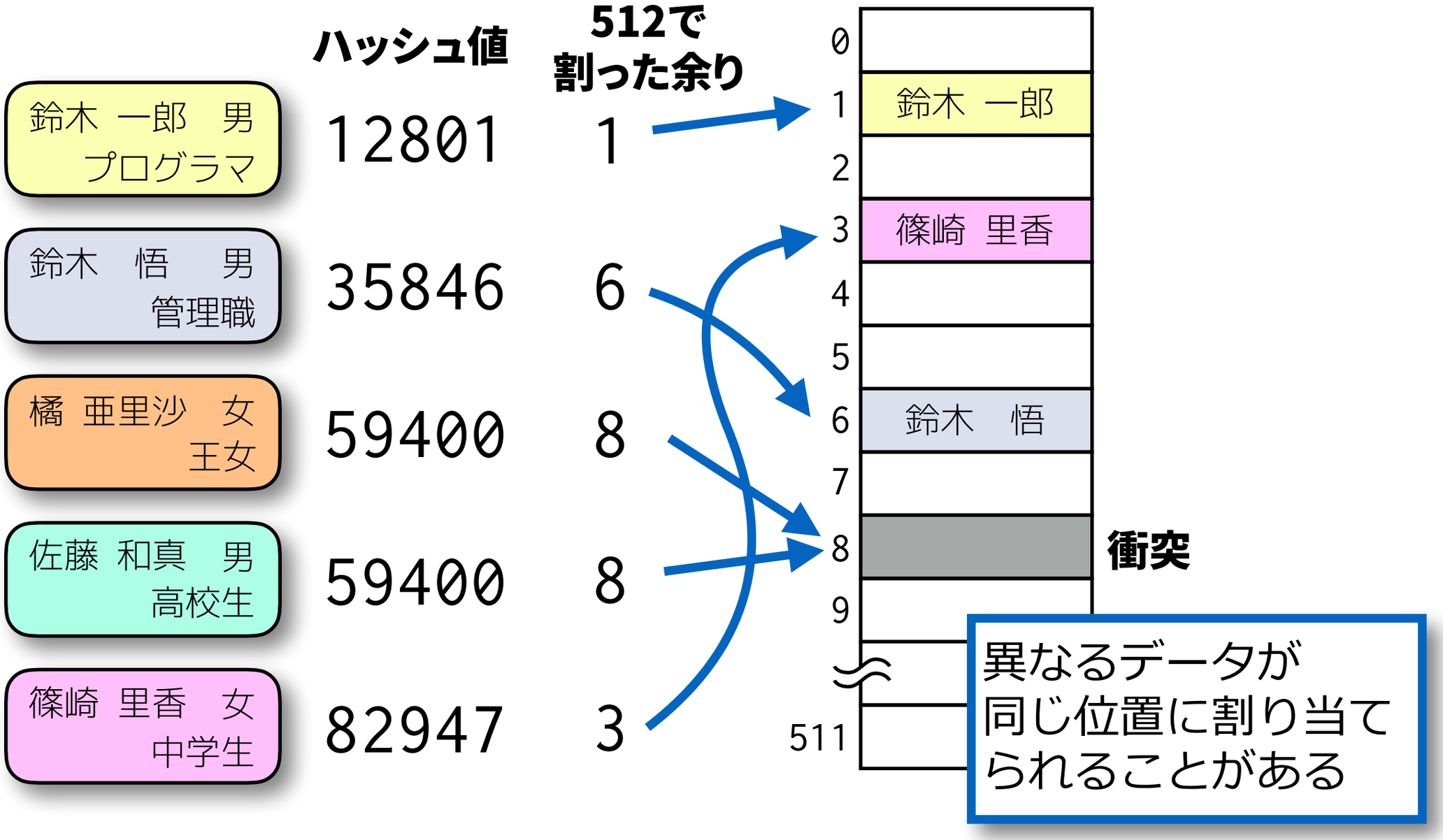
ハッシュ関数

- 何らかのデータを一定長のデータ（ビット列=整数）に要約する関数



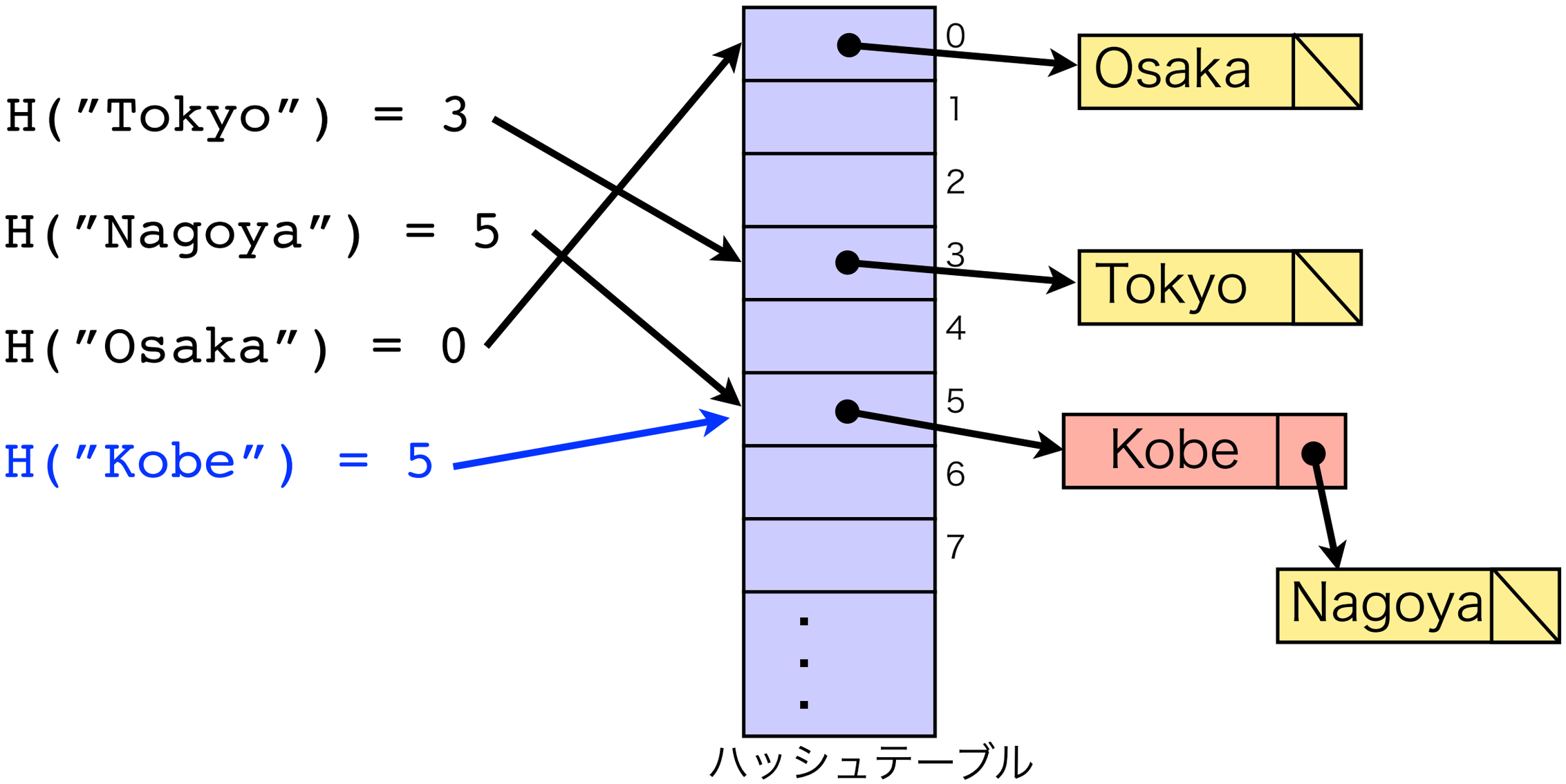
ハッシュ値による検索

- ハッシュ値から配列の添字を得て、そこに値を格納
- 極めて高速に検索できる（計算量は $O(1)$ で済む）



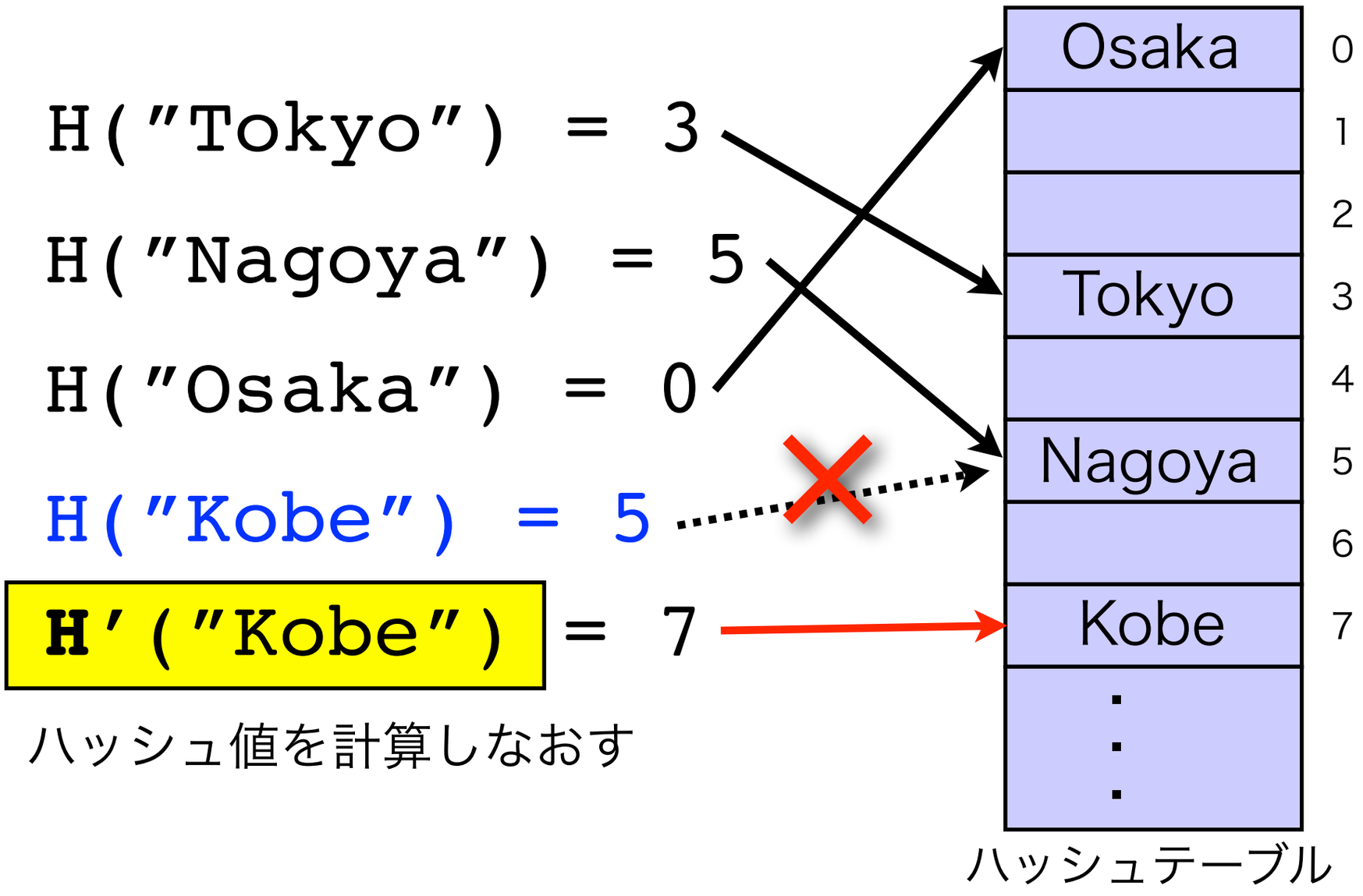
衝突への対応：チェイニング方式

- 衝突した要素をリストにつなげて格納しておく。ハッシュテーブルはリストの先頭へのポインタを持つ。



衝突への対応：オープンアドレッシング方式

- データをハッシュテーブル自体に格納する。衝突した要素は別の位置に格納する。



ハッシュ関数に求められる要件

- データ検索に用いるハッシュ関数に必要な条件

1. 計算が高速に行えること
2. 同じデータには常に同じハッシュ値が得られること
   - ✦ 逆に、ハッシュ値が同じでも元のデータが同じとは限らないので注意
3. 比較的桁数の多い整数値が得られること
   - ✦ 桁数が多い方が応用がきく
4. 異なるデータに対しては、得られるハッシュ値が適当に（不規則に）「ばらける」こと
   - ✦ 検索の際の衝突の可能性を低くする

例: 文字列データのハッシュ値

- 異なる文字列には、できればかなり違う値を返すようにする。ハッシュ値をばらけさせるために、むしろ意味はなくてよい。ただし計算は高速に行えることが望ましい。

```
long hash(const char *p)
{
    int b;
    long v = (long)*p++ * 7013;
    while ((b = *p++) != 0 && v < (LONG_MAX >> 8))
        v = ((v * 11) >> 3) ^ (b << 16);
    return v;
}
```

これは例であって、ハッシュ値として適しているかどうかは実際に適用する値の傾向による

入力		出力
“a”	=	680261
“b"	=	687274
“T1”	=	4021265
“T2"	=	4086801
“Aqua"	=	25039096
“Megumin”	=	60265990
“Darkness"	=	80105640

例: 整数に対するハッシュ値

- 整数をそのままハッシュ値として使ってもよいが、値が連続したり分布が偏ったりしているとハッシュ値としての性質は良いとは言えない。

```
long hash(long n)
{
    long t = 0xff & ~(n >> 8);
    long v = (n & 0xff) * 211;
    return (t ^ v);
}
```

これは例であって、ハッシュ値として適しているかどうかは実際に適用する値の傾向による

入力		出力
1	-->	44
2	-->	345
3	-->	646
4	-->	947
20	-->	4227
50	-->	10697
55	-->	11690
100	-->	21139
1200	-->	37355
5444	-->	14566
12300	-->	2347
45600	-->	6701