

# アルゴリズムとデータ構造

## 第9回 連結リスト

# 第9回のキーワード

2

## アルゴリズム関係

- 連結リスト (linked list)
- 単方向連結リスト (single linked list)
- ノード (node) / セル (cell)
- リンク (link)
- 再帰データ型 (recursive data type)
- 連結リストの線形探索
- 番兵法 (sentinel method)

## Java関係

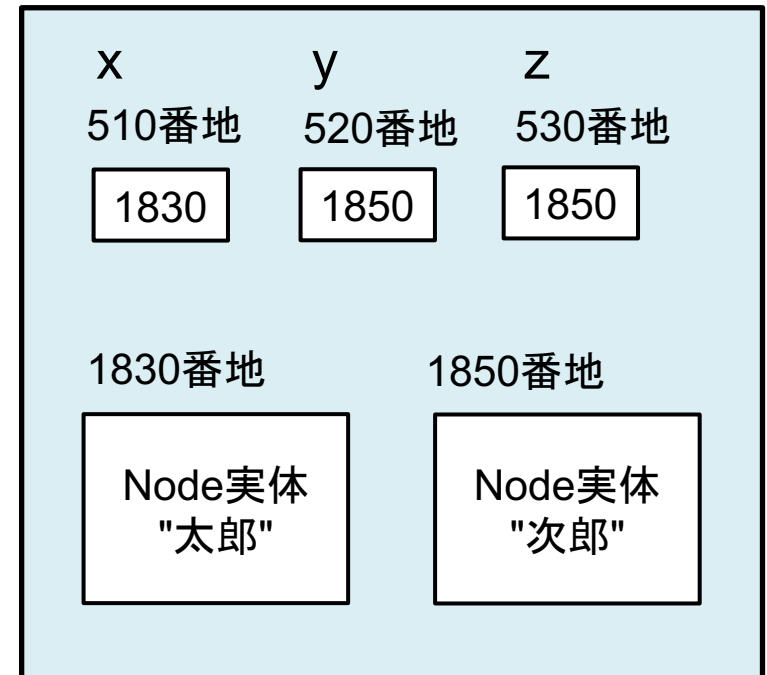
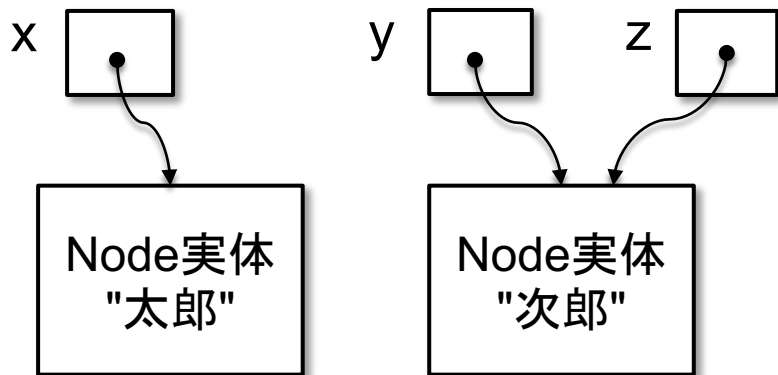
- 参照型 (reference type)
- 自己参照クラス (self-referential class)
- `node1.next = node2`
- `head = null`
- `for (Node n = head; n != null; n = n.next)`

# “参照型”の復習

3

- Javaでは、クラス型や配列型の変数やフィールドは、インスタンス(実体)への参照(=つながり)しか持たない

```
Node x = new Node("太郎");  
Node y = new Node("次郎");  
Node z = y;
```



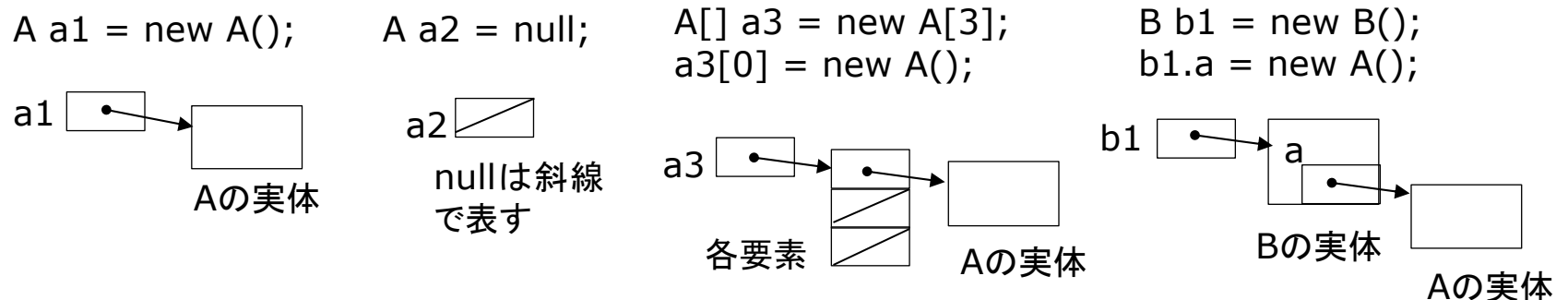
メモリ内でのイメージ(番地は適当)

# 確認問題

4

- Javaにおいてクラスは参照型であり，クラス型の変数やフィールドは，その実体（インスタンスの内容）ではなく，メモリ内でのアドレスを保持する
- 例えば，次のようにクラスA, B, Cを定義した場合を想定する
 

```
class A { }   class B { A a; }   class C { C c; }
```
- このとき，次のようなコードを考えるとそれぞれその下のように図示できる



- 同様にして，以下のコードを図示せよ

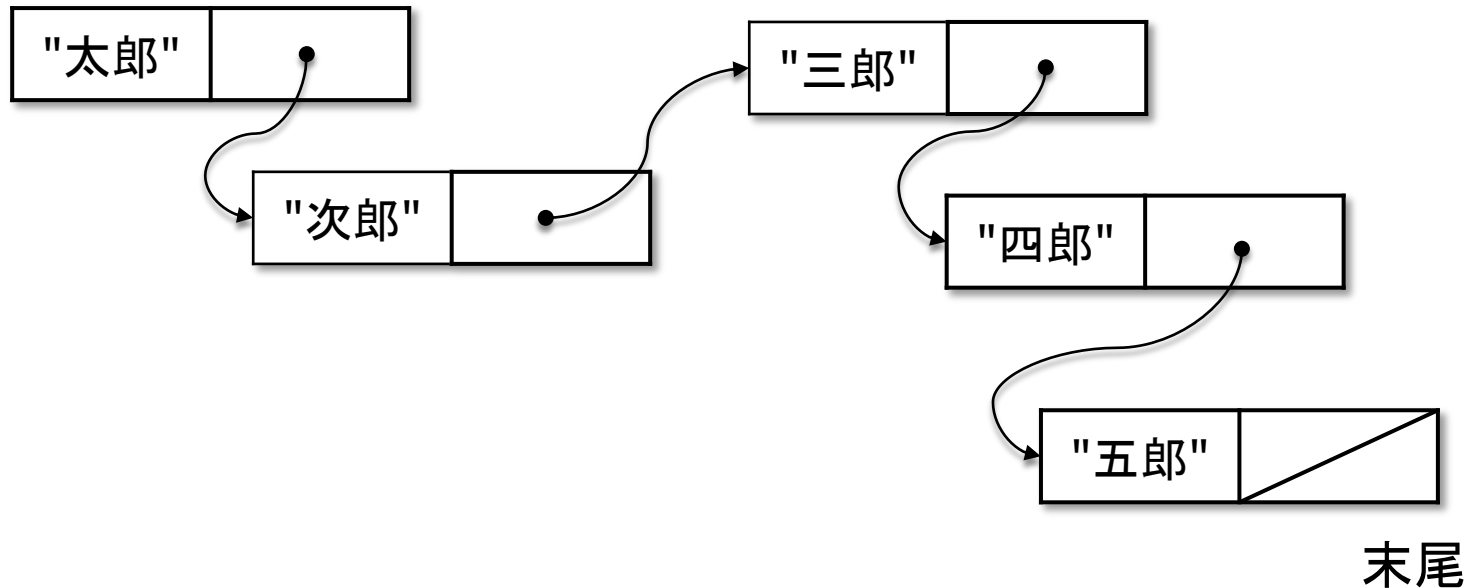
```
C c1 = new C();
C c2 = new C();
c1.c = c2;
c2.c = null;
```

# 連結リスト(linked list)

5

- データを次々と数珠つなぎにしたデータ構造

先頭



- 利点: 登録数が無制限&途中での挿入・削除が容易

# ノード(セル)

6

- 連結リストでデータを入れるための箱
  - 「データ」と次のノードへの「つながり」を持つ



「つながり」を英語で  
言えば「リンク」

- クラスによる表現(自己参照クラス / 再帰データ型)

```
class Node {  
    String data;  
    Node next;  
}
```

え!?! ノードの中  
にノードが入って  
いるの?

そうではなくて、  
次のノードへの  
「つながり」

# ノードの連結

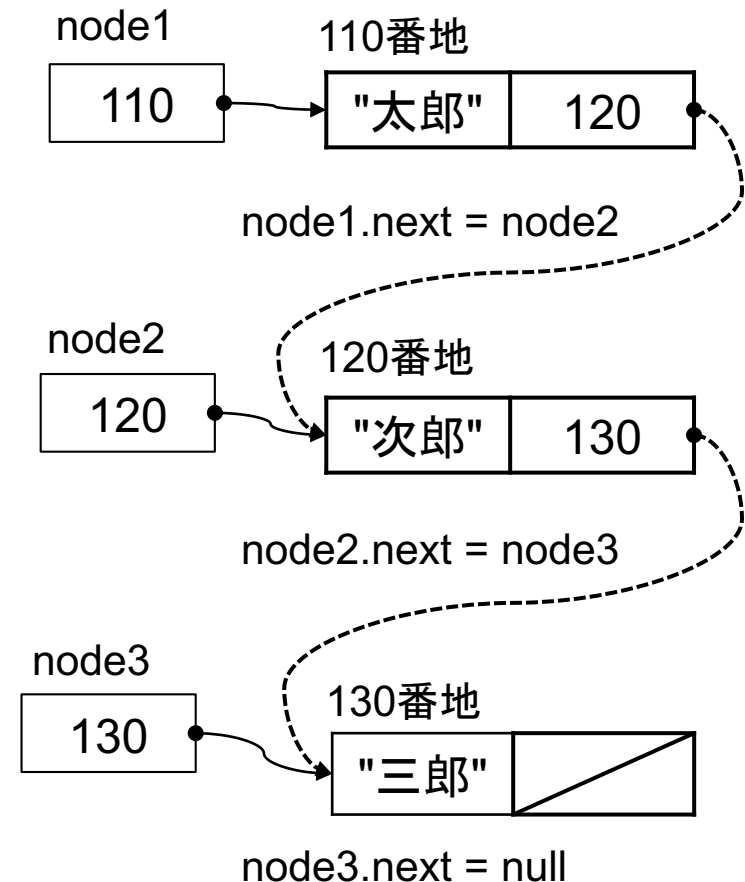
7

- 参照型の性質により，代入操作によってノードの連結を実現できる

```
Node node1 = new Node("太郎");
Node node2 = new Node("次郎");
Node node3 = new Node("三郎");
```

```
node1.next = node2;
node2.next = node3;
node3.next = null;
```

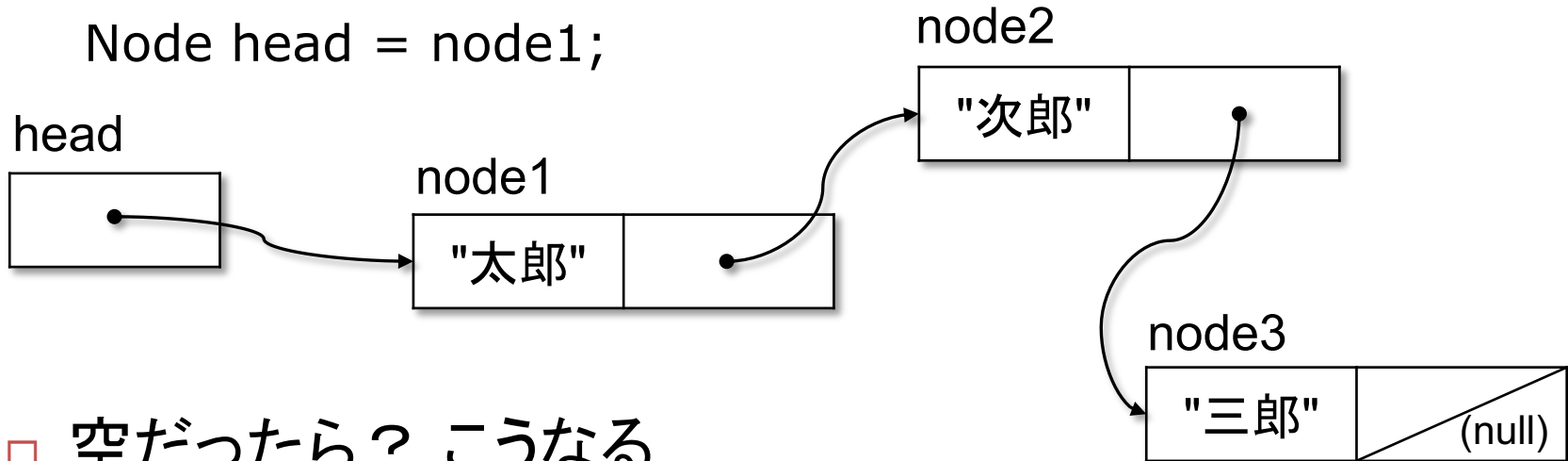
「終」のマークとして  
nullを使う



# スタート地点は必要

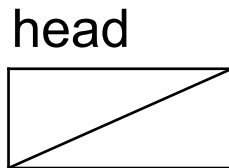
8

- リストの要素を順にたどれるようにするためにはいつも先頭ノードを指す変数 (head) が要る



- 空だったら？ こうなる

Node head = null;



nullが入っていることを  
図では斜線で表す

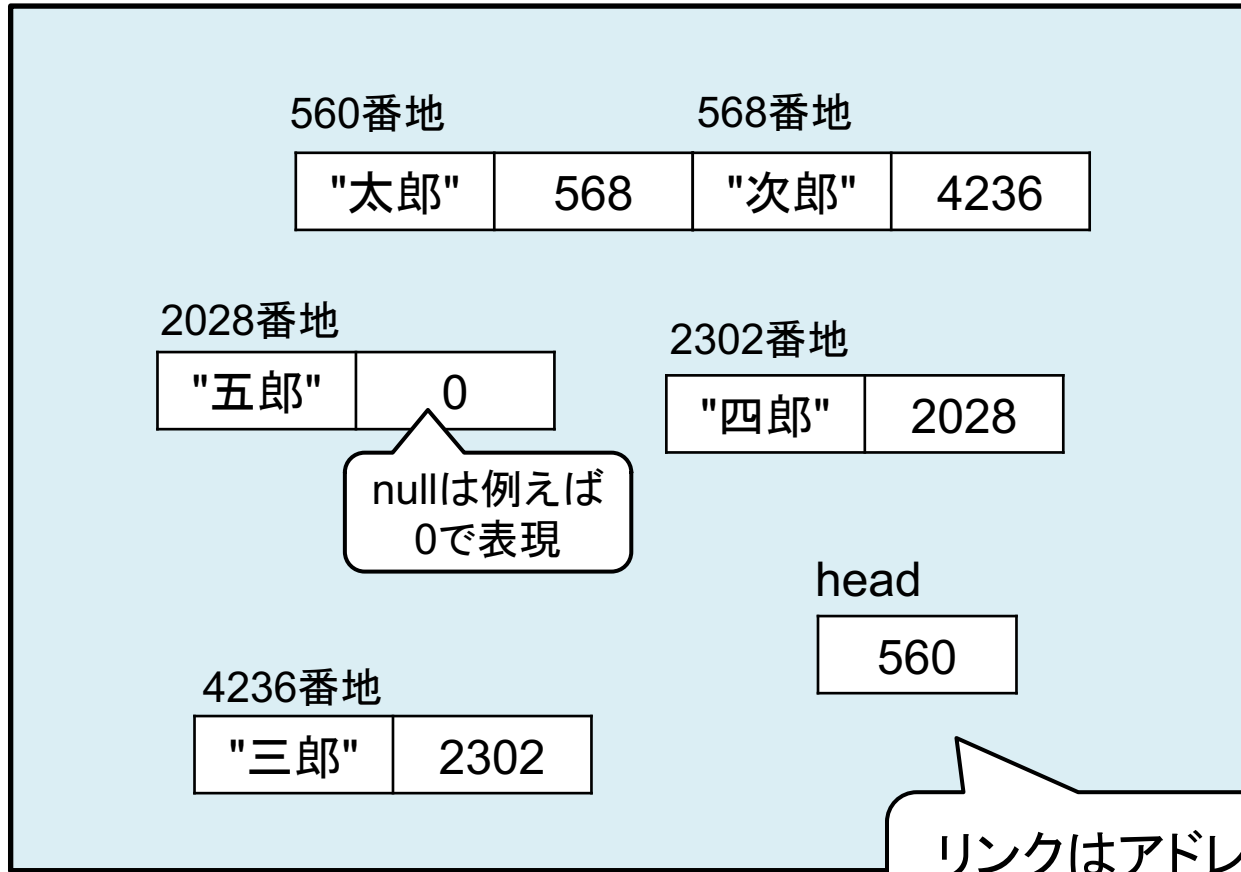


# メモリ内でのイメージ例

9

アドレス

0番地  
1000番地  
2000番地  
3000番地  
4000番地  
5000番地



nullは例えば  
0で表現

リンクはアドレス  
(番地)で実現

メモリマップ

# 確認問題

10

## □ 連結リストの構造

- 下図はdouble型を要素とする連結リストが格納されているメモリの内容(の概念例)である。先頭を指すhead変数は、5000番地に格納されており、nullの値は0番地である。
- ノードを表すクラスは次のように定義されているものとする。  
class Node { double data; Node next; }
- この連結リストの構造を図示せよ。

アドレス	5000	5008	5016	5024	5032	5040	5048
内容	5008	3.14	5024	0	5040	1000	0

## □ 連結リストの構成

- 上記の連結リストを構成するJavaコードの概略を示せ。

# リンクをたどる

11

## □ こういうふうにも書ける

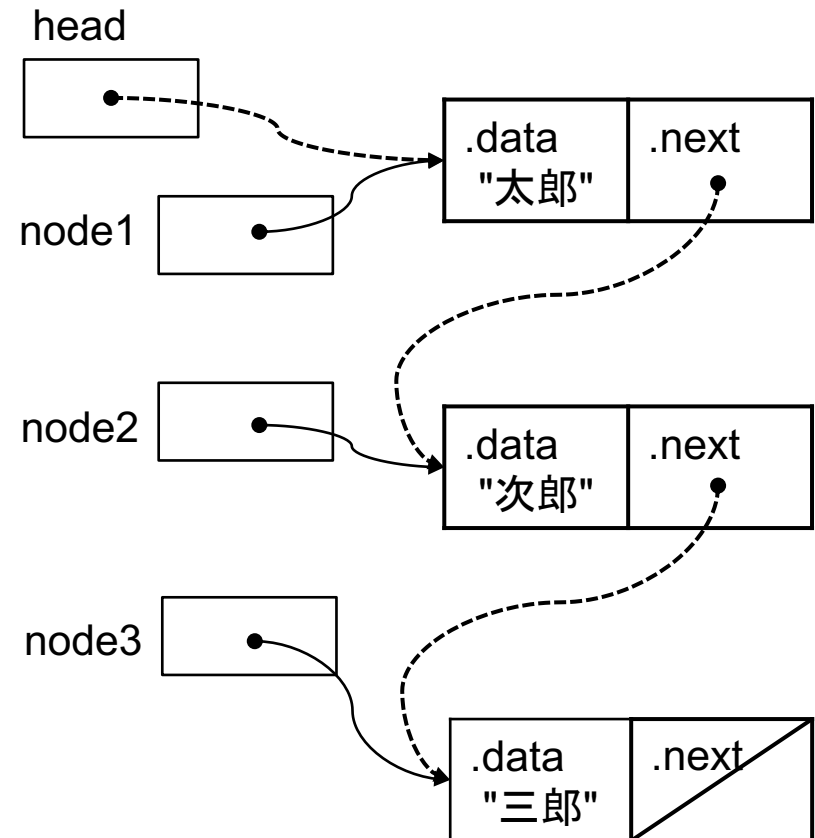
### ▣ nextを連鎖的にたどる

```
head = node1;  
head.next = node2;  
head.next.next = node3;  
head.next.next.next = null;
```

## □ for文を使ってたどれる

### ▣ 探索や内容表示でよく使う

```
for (Node n = head; n != null; n = n.next) {  
    System.out.println(n.data);  
}
```



# 確認問題

12

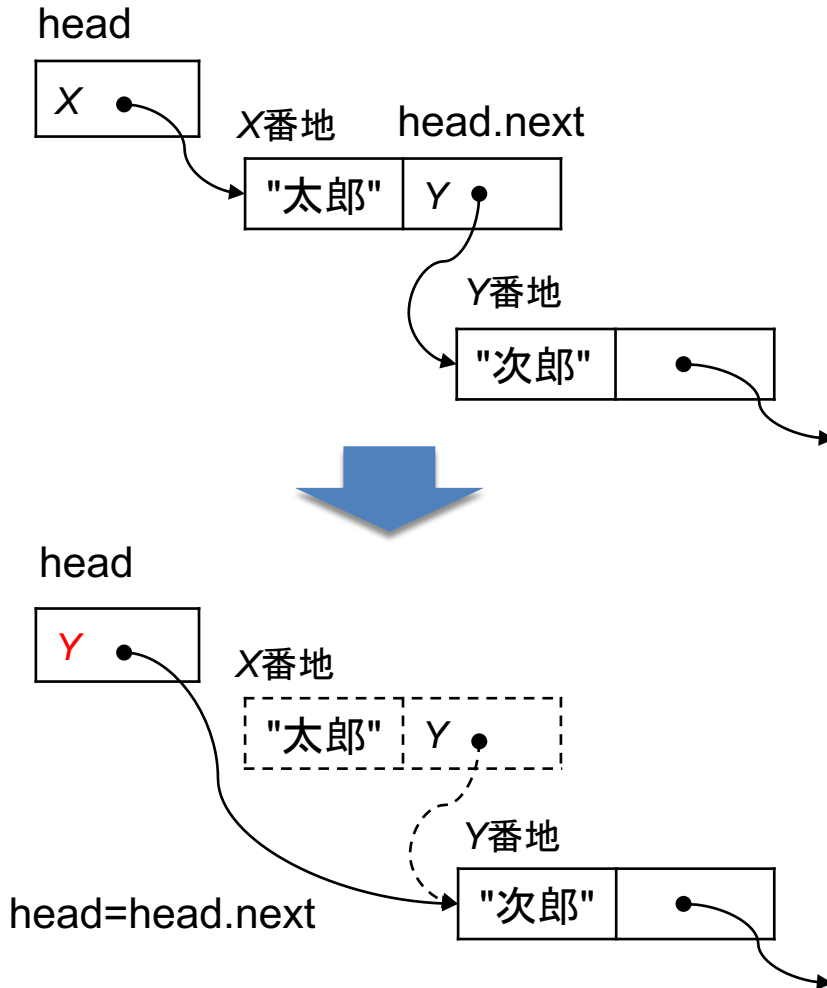
- リスト先頭での挿入と削除
  - 連結リストのノードを表すクラスが次のように定義されており、変数headは先頭ノードを指すものとする。

```
class Node { int data; Node next; }
```
  - 新しいノード $n$ を生成し、連結リストの先頭に挿入する処理の概略を、まず図示してからJavaのコードで示せ。
  - 連結リストの先頭からノードを1つ削除する処理の概略を、まず図示してからJavaのコードで示せ。
  
- 連結リストの線形探索
  - 上記と同様に定義された連結リストから、値として「10」を保持するノードを探索する処理をJavaのコードで示せ。

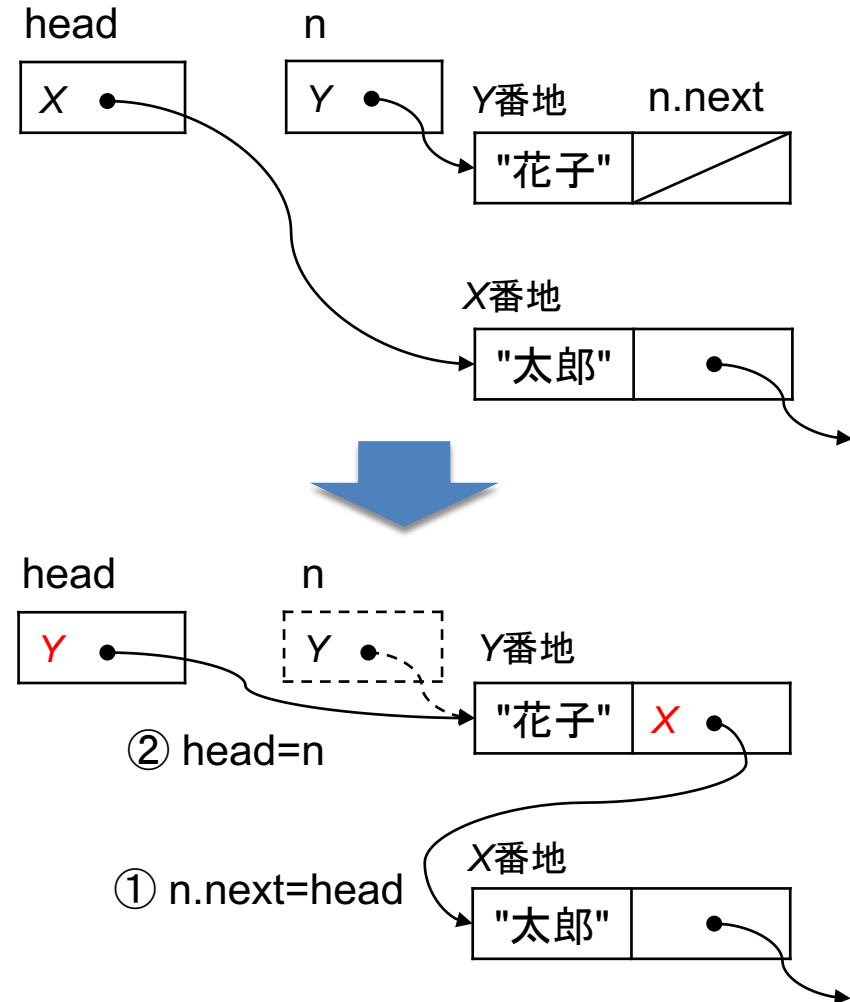
# リスト先頭での削除と挿入

13

## □ 先頭ノードの削除 (pop)



## □ 先頭ノードの挿入 (push)



# 番兵法

14

- 番兵(番人)
  - ▣ 探索において、最後の探索要素に(偽の)探索値(番兵)を入れておく
  - ▣ すると、探索値は必ず発見されるようになる(元のデータに探索値がなくても偽の探索値で止まる)
  - ▣ よって、探索範囲の境界チェックの処理(オーバーランの防止)が不要になる

