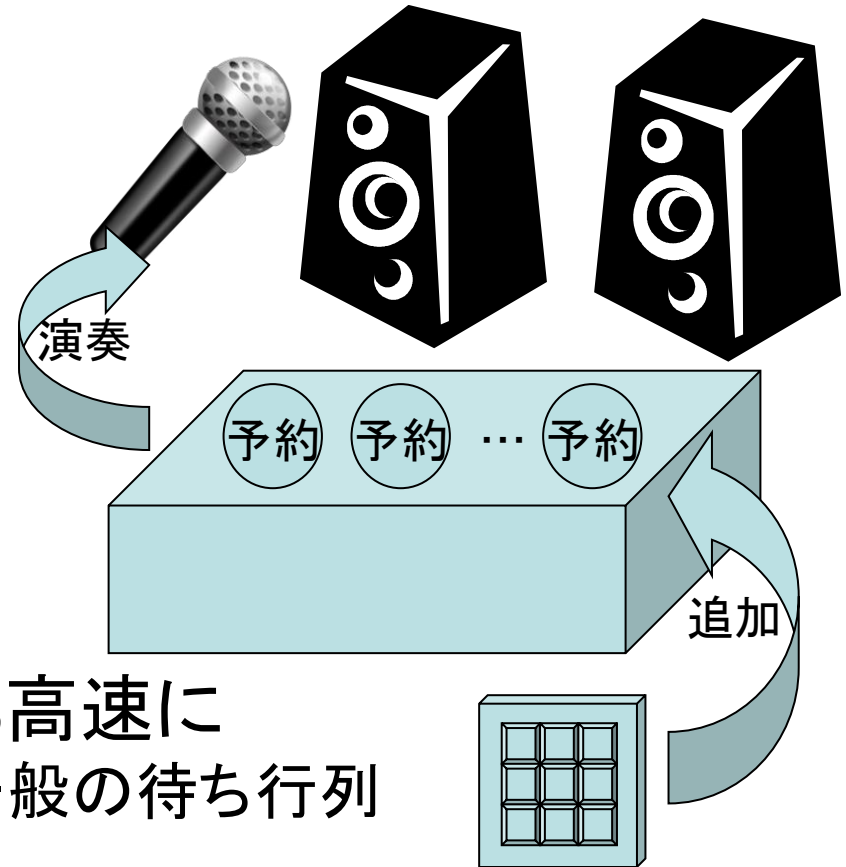


これまでのデータ「構造」と 再帰データ構造

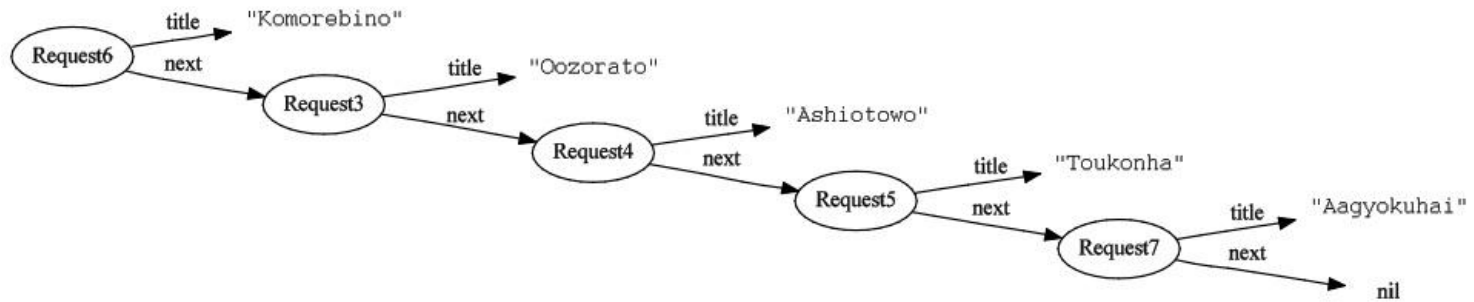
- これまで
 - データを組み合わせて大きなデータを作る
 - 大きなデータを分解すると最後は数値や文字列
 - 有限の大きさ(個数に制限がある)
- 再帰データ構造
 - 無限の大きさの(個数に制限がない) データを表現できる
 - 伸び縮みするデータの列や、関係ネットワークなどを表わすのに適している
 - (「再帰」の意味は後で)

例題：カラオケ演奏器

- ・ 予約管理機能
 - 予約を追加する
 - 先頭から演奏
 - 途中の予約を取消
 - 先頭に割り込み
- ・ 「最大予約数」を設けずに
- ・ 予約の数が非常に多くても高速に
(カラオケでは考えにくいですが、一般の待ち行列ではよくある)
- ・ どのようなデータ構造がよいか?

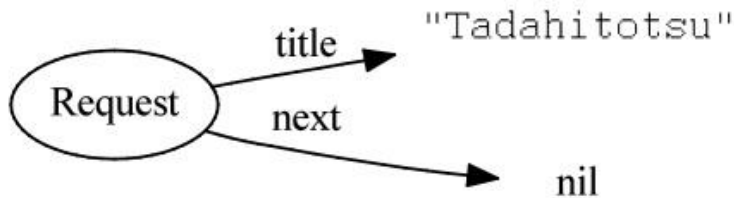


リスト構造で予約リストを表わす



- 1つの予約情報は
 - 予約している曲の情報
 - 次の予約情報
- いくらでもつなぐことができる
- 再帰的な構造
 - 予約情報から次の予約情報へ

1つの予約情報の定義



```
class Request
  attr_accessor("title", "next")

  def initialize(t)
    self.title = t
    self.next = nil
  end
end
```

request.rb

やってみよう
(配布プログラムview.rb,
request.rbをダウンロード)

```
> load("./request.rb")
> load("./view.rb")
> r0 = Request.new("Tadahitotsu")
=> #<Request:0x404545e4 @title = "Tadahitotsu ",
    @next=nil>
> view(r0)
```

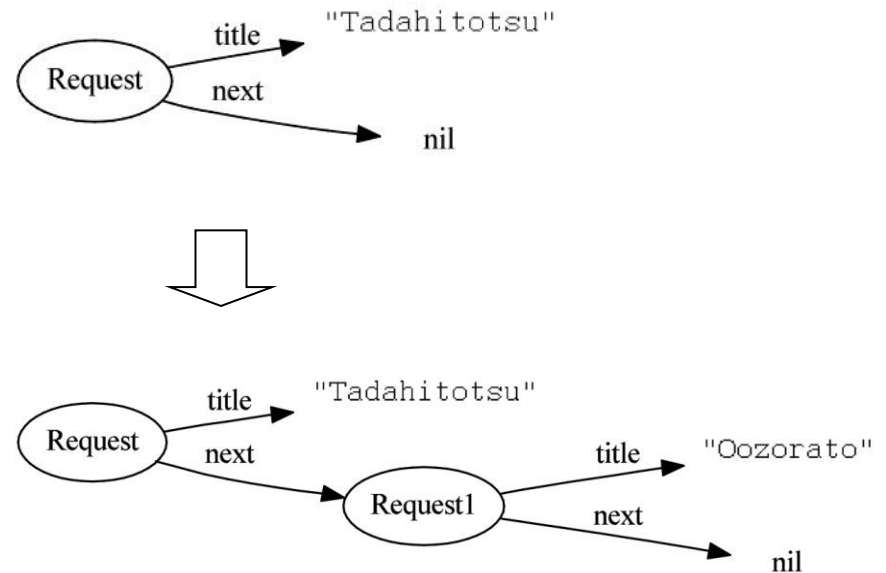
できたら 1 を投票してください

予約情報に対する操作：追加

- 追加する予約情報を作り、「最後の予約」の「次」に代入
- 個数に制限がない

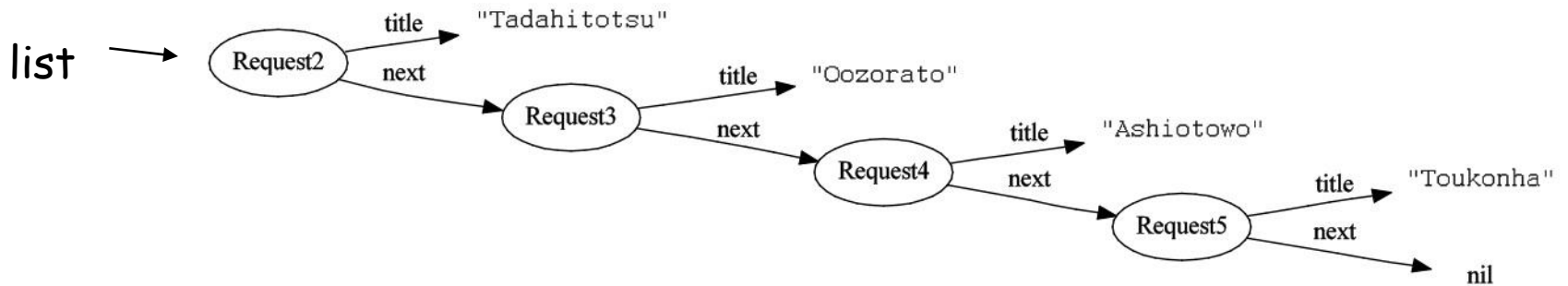
やってみよう

```
> r1 = Request.new("Oozorato")  
> r0.next = r1  
> view(r0)
```



できたら 1 を投票してください

予約リストの例



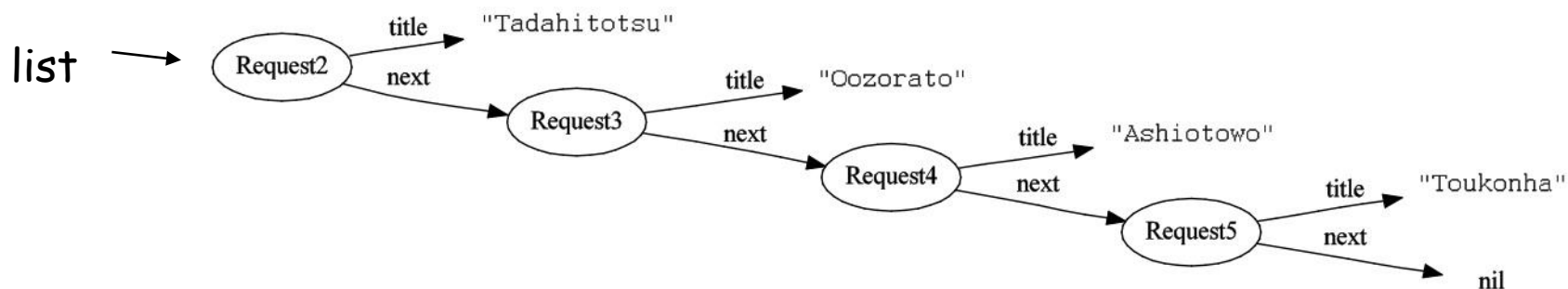
やってみよう

ut_songs.rbをダウンロード

```
> load("./ut_songs.rb")  
> list = ut_songs()  
> view(list)
```

できたら 1 を投票してください

予約情報に対する操作：取り出し



- 先頭を消す=先頭を「次の予約情報」に変更

list

.next

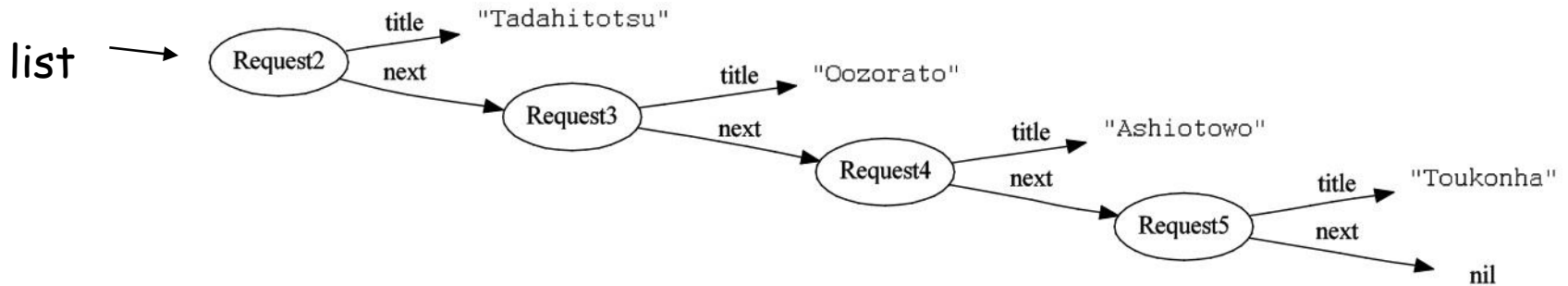
やってみよう

```
> list = list.next
```

```
> view(list)
```

できたら 1 を投票してください

予約リストに対する再帰的操作： 末尾の予約を探す



$$\text{last_}(r) = \begin{cases} r & (\text{次が空}) \\ \text{last_}(r\text{の次}) & (// \text{でない}) \end{cases}$$

やってみよう

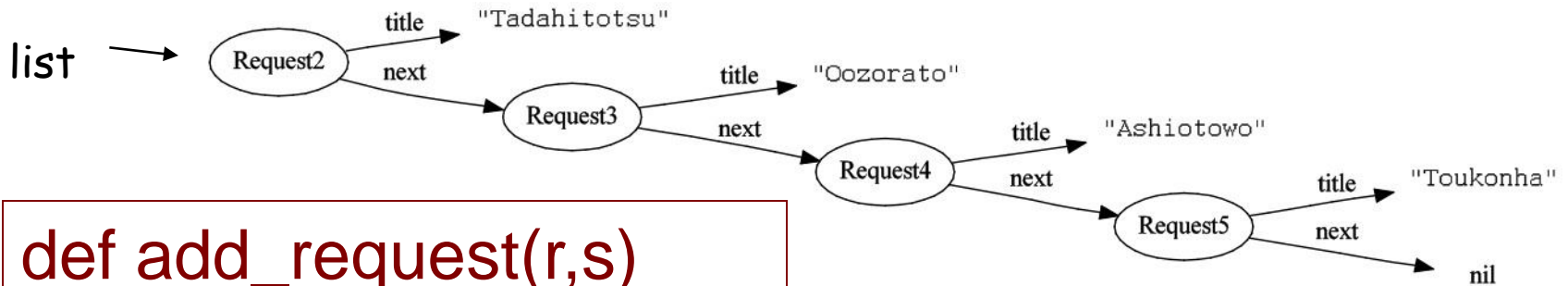
```
> last_request(list)
```

できたら 1 を投票してください

```
def last_request(r)
  if r.next == nil
    r
  else
    last_request(r.next)
  end
end
```

request.rb

予約リストに対する再帰的操作： 末尾への追加



```
def add_request(r,s)
  last = last_request(r)
  last.next = s
end
```

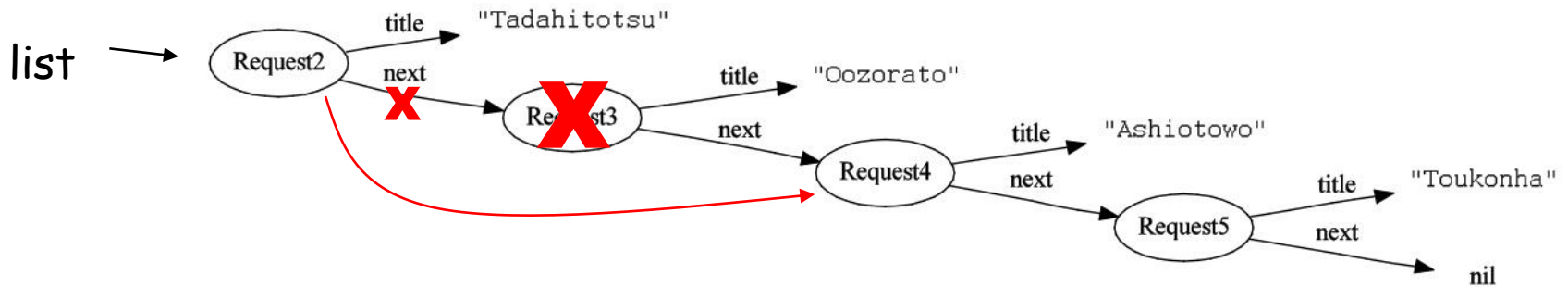
request.rb

やってみよう

```
> add_request(list, Request.new("Aagyokuhai"))
> view(list)
```

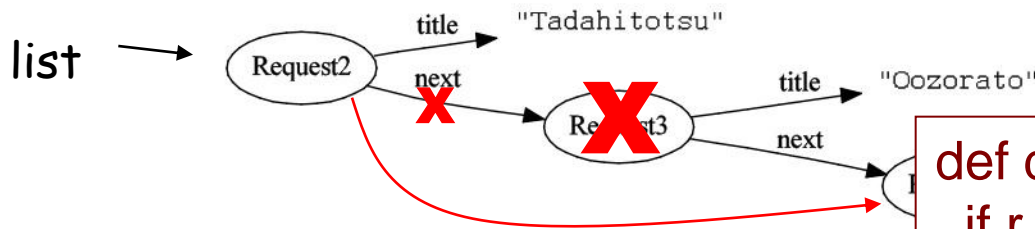
できたら 1 を投票してください

予約リストに対する再帰的操作： 名前で削除



- 曲名(title)がtである予約を消す
- = 1つ手前の予約の「次」を「次の次」に変更
- …手前に戻るには?
- …消される予約が最後でも大丈夫?
- …消される予約が先頭だったらどうする?

予約リストに対する再帰的操作： 名前で削除



```
def delete_request(r,t)
  if r.title == t
    r.next
  else
    r.next = delete_request(r.next, t)
  end
end
```

request.rb

やってみよう

```
> list = delete_request(list, "Oozorato")
```

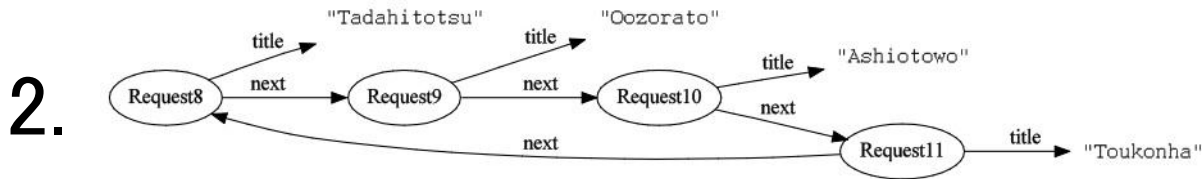
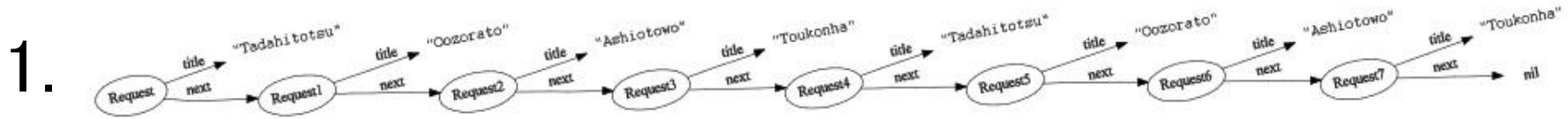
曲名(title)がtである予約を消したリストを求める

$$\text{delete}(r,t) = \begin{cases} r\text{の次} & (r\text{の曲名が}t) \\ r\text{の次をdelete}(r\text{の次},t)\text{にした}r \text{ (// でない)} \end{cases}$$

クイズ

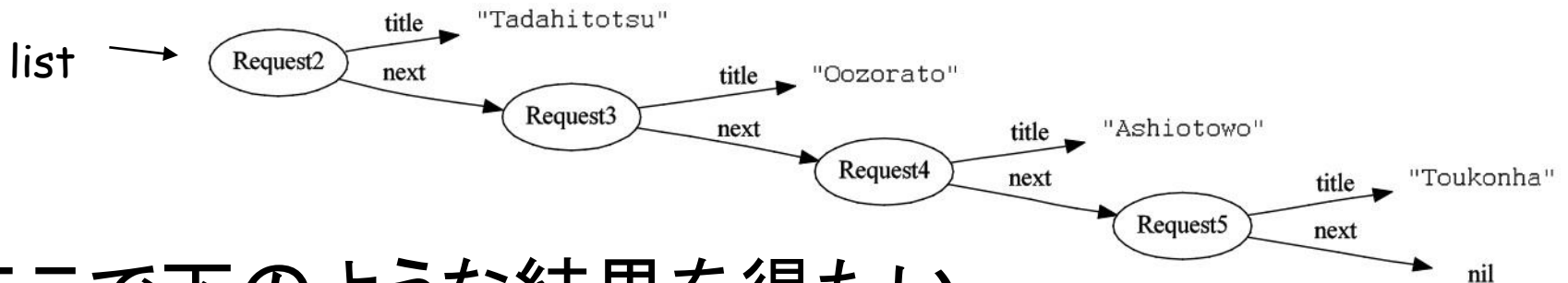


ここで `add_request(list, list)` をして
`view(list)` をすると?

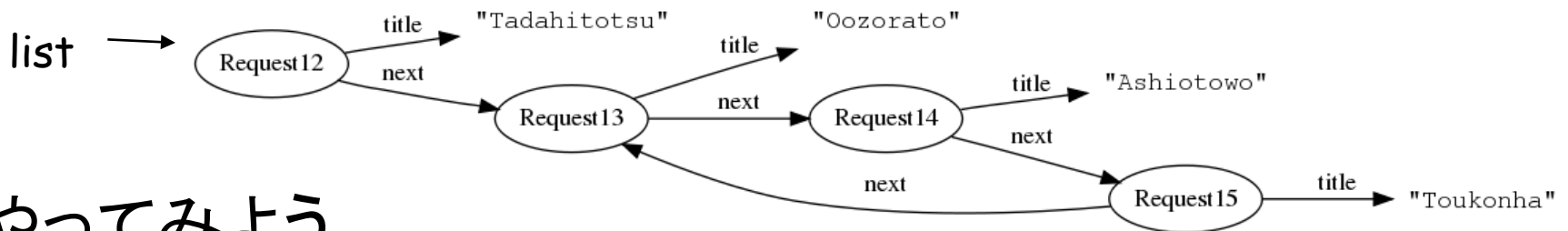


3. エラーになる

クイズと練習



ここで下のような結果を得たい



やってみよう

- > `add_request(list, ここに何か...)`
- > `view(list)` (失敗したら, `list=ut_songs()` する)

できたら 1 を投票して、
練習 9.5, 9.6, 9.7 に挑戦してください

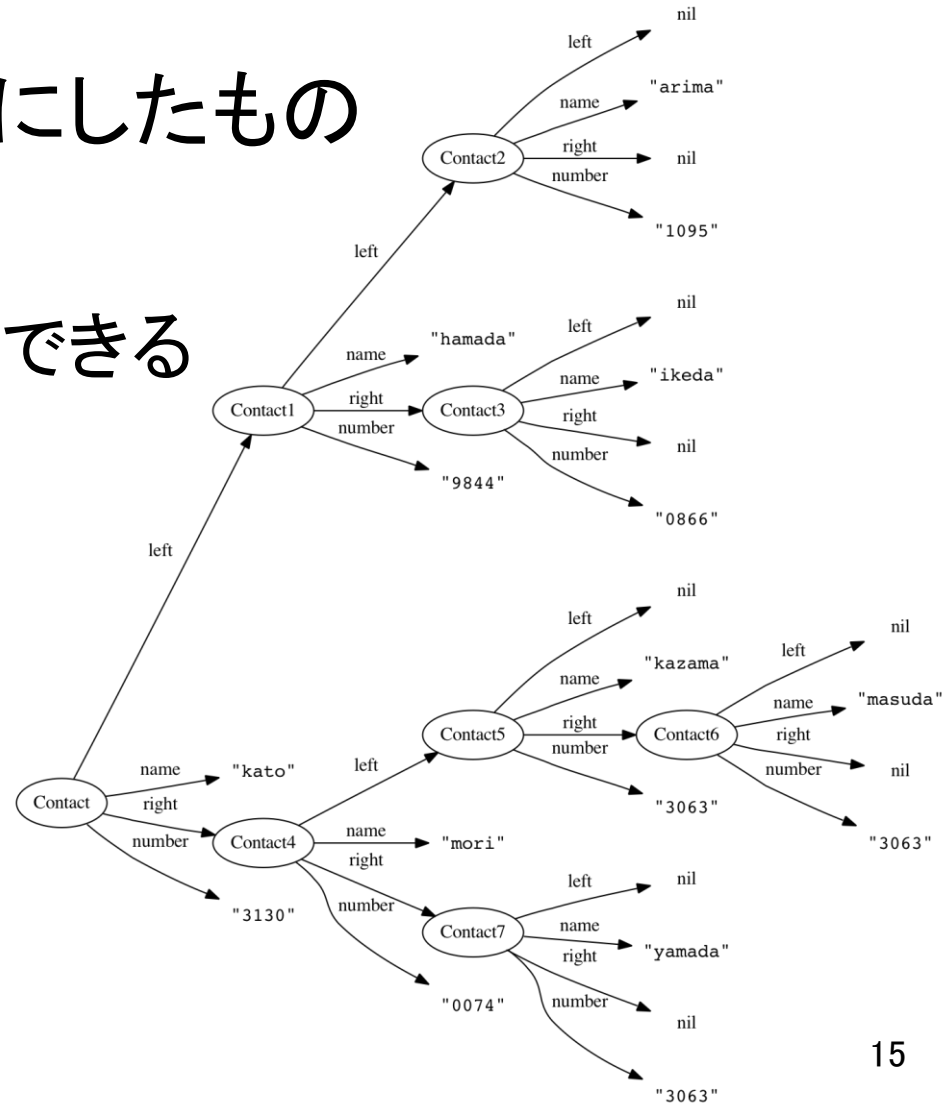
再帰データ構造の例その2: 木構造

- ・ リストの「次」を複数にしたもの
- ・ 応用

➤ 高速な検索・更新ができるデータベース

➤ 文や式の解析
(自然言語の文もプログラムも)

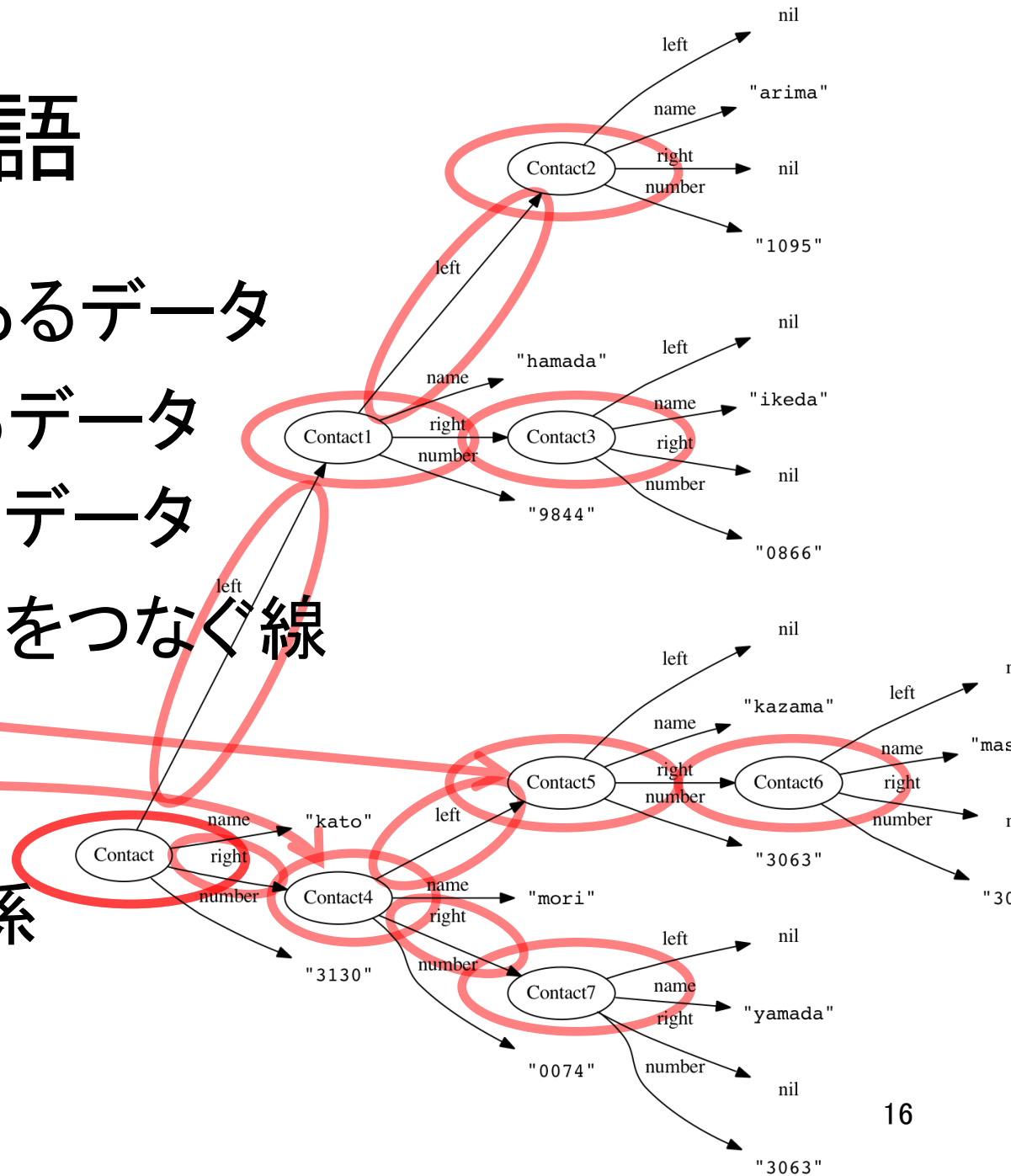
➤ ファイルシステム



木構造の用語

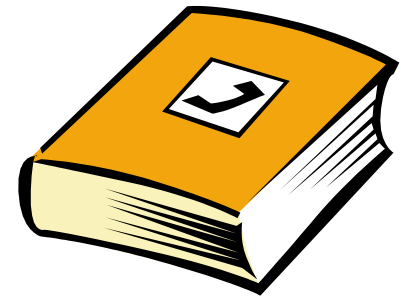
- ・ 根 : 出発点にあるデータ
- ・ 葉 : 末端にあるデータ
- ・ 節 : 続きがあるデータ
- ・ 枝 : データ同士をつなぐ線
- ・ **親子**

つながった
データ間の関係



例題: 電話帳

- 機能
 - 名前から電話番号を検索
 - データを追加
 - データを削除
 - 一覧表示
- 件数の上限を設けずに
- 大量にあっても高速に
(一般的なデータベースでは必須)



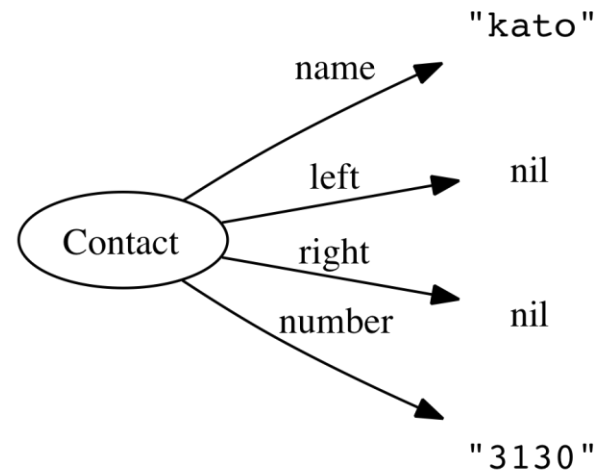
電話帳に適したデータ構造は？

	件数の上限				
		検索	追加	削除	並べ
配列	あり	遅い	速い	遅い	遅い
整列された配列	あり	速い	遅い	遅い	速い
リスト	なし	遅い	速い	遅い	—

- ・ 遅い…件数に比例した(orそれ以上の)時間がかかる → 数万件、数百万件は苦しい

連絡先情報

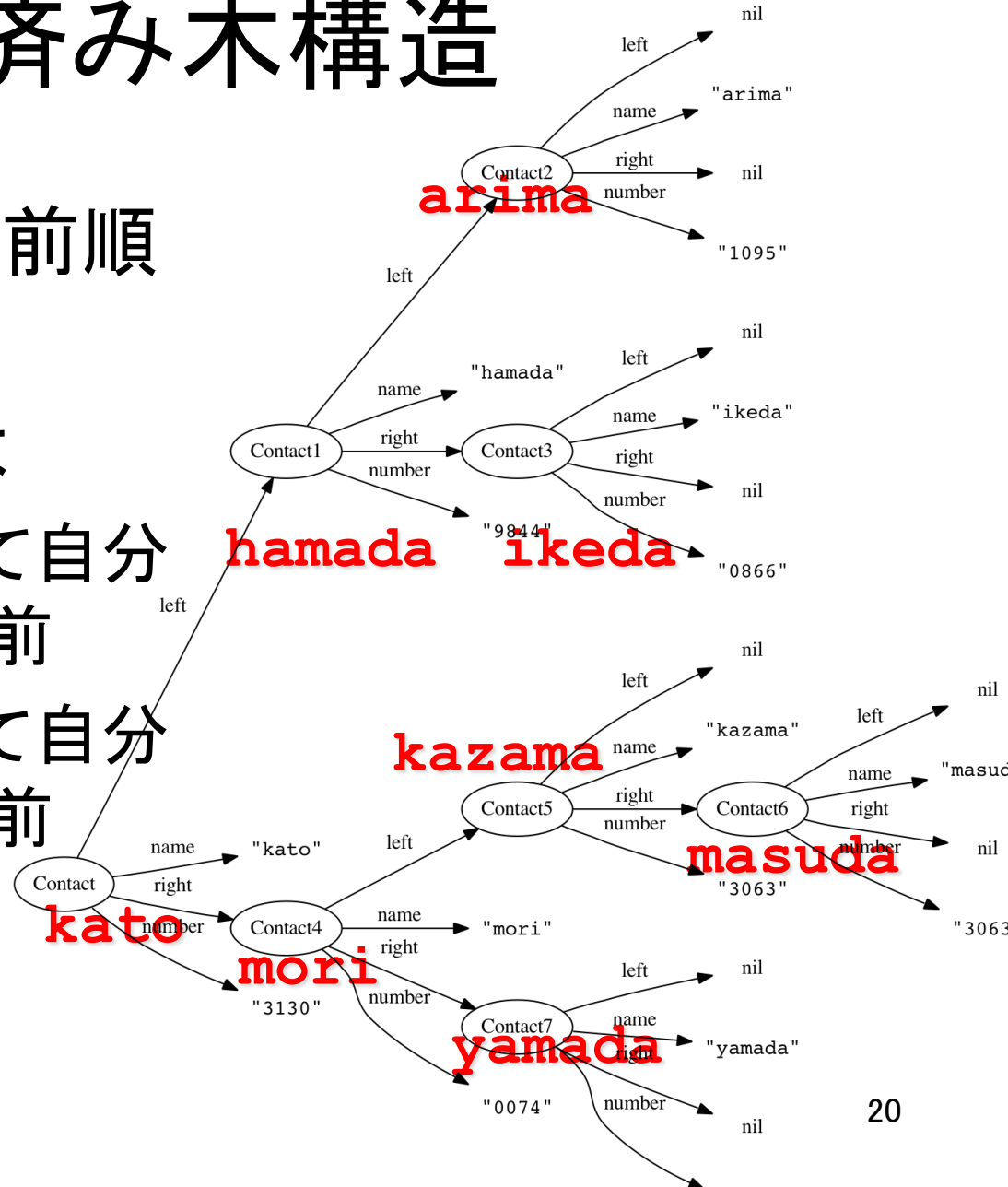
```
class Contact
  attr_accessor("name", "number", "left", "right")
  def initialize(name, number)
    self.name = name
    self.number = number
    self.left = nil
    self.right = nil
  end
end
```



contact.rb

整列済み木構造

- 電話帳の場合: 名前順
(辞書順)
 - ある連絡先からは
 - 「左」の子孫は全て自分よりも前に来る名前
 - 「右」の子孫は全て自分よりも後に来る名前
- となるように配置

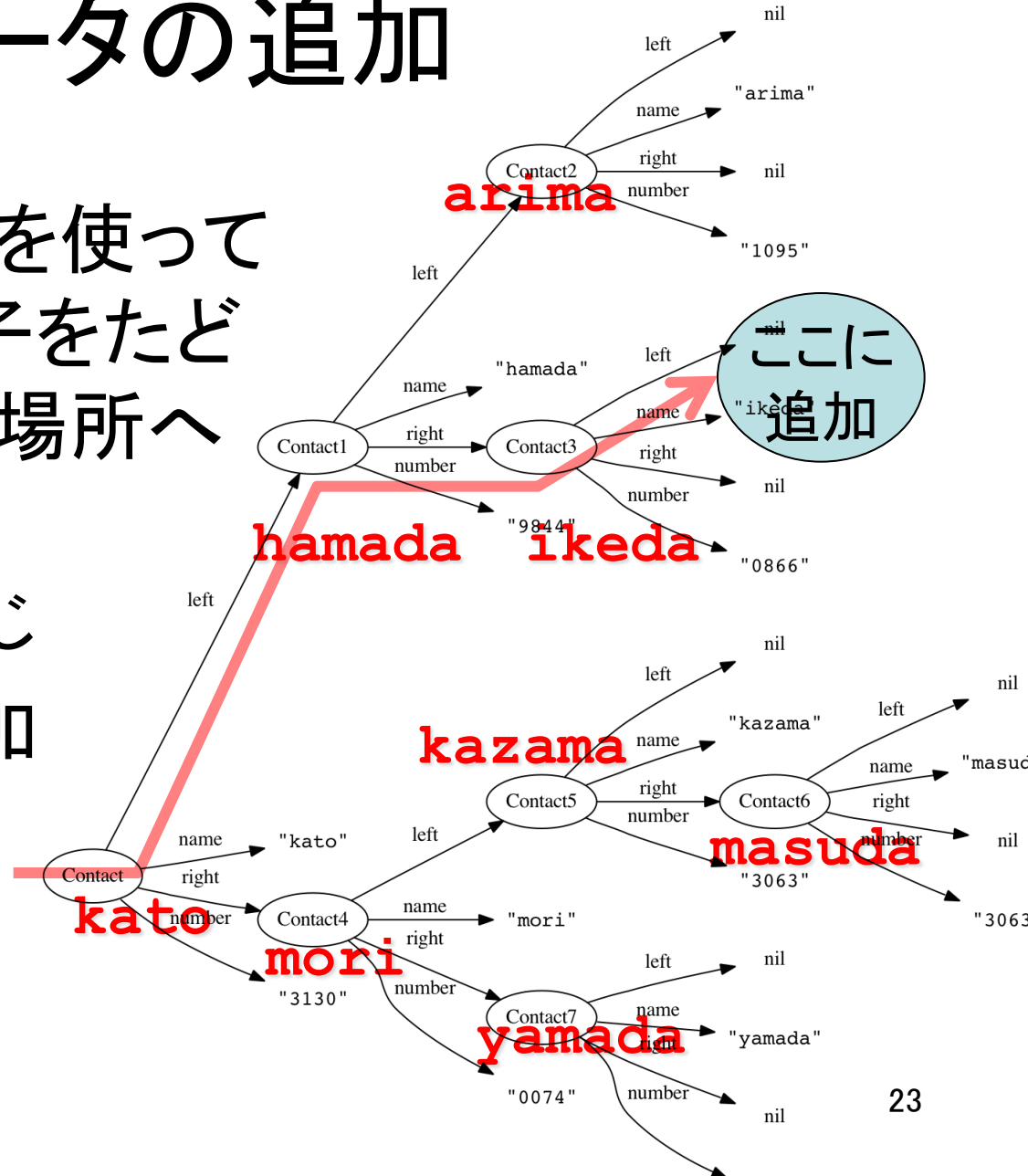



```
def find_contact(p,n)
  if p.name == n
    p
  else
    if n < p.name
      find_contact(p.left, n)
    else
      find_contact(p.right, n)
    end
  end
end
end
```

contact.rb

データの追加

- 名前の大小関係を使って左右どちらかの子をたどり、「空」になった場所へ追加
 - 速度は検索と同じ
- 例:「hirano」を追加



```
def add_contact(p,q)
  if p == nil
    q
  else
    if q.name < p.name
      p.left = add_contact(p.left,q)
    else
      p.right = add_contact(p.right, q)
    end
  end
  p
end
end
```



```
def tree_to_array(p,a,i)
  if p.left != nil
    i = tree_to_array(p.left,a,i)
  end
  i = i+1
  a[i] = p.name
  if p.right != nil
    i = tree_to_array(p.right,a,i)
  end
  i
end
```

binarysort.rb

