Programming Paradigms

- Procedural Programming
- Object-oriented programming
- Value-oriented programming
- Logic programming

Value-oriented Programming

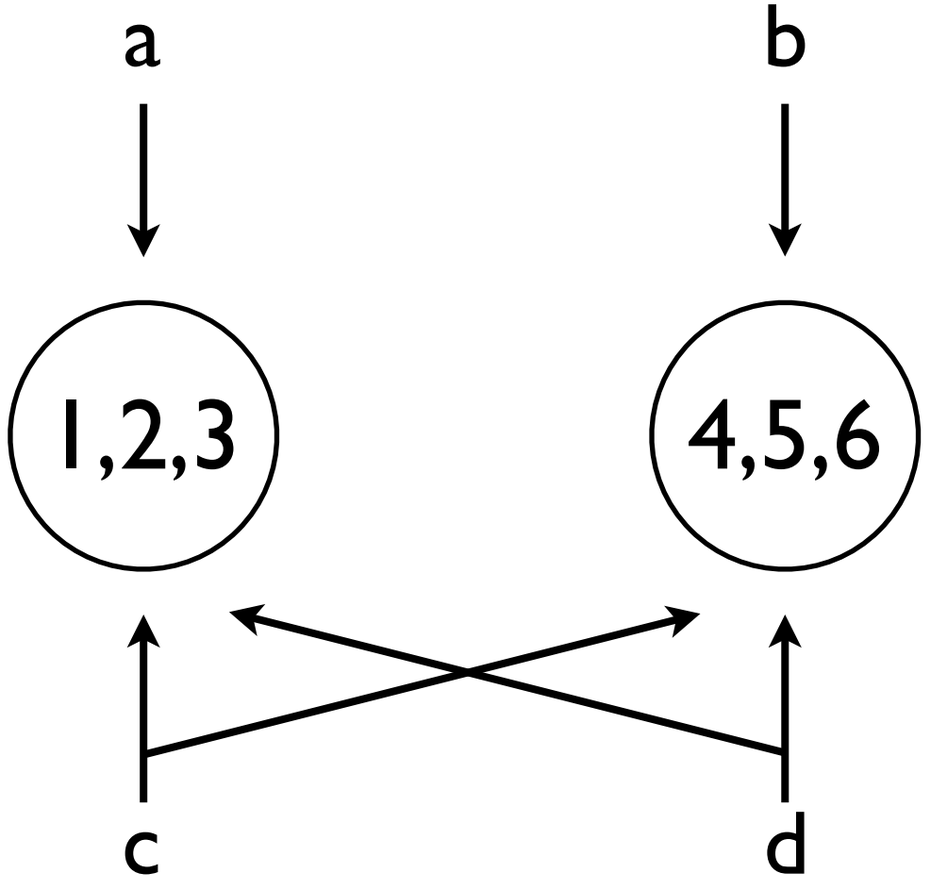
질문

```
a := set (1,2,3);
b := set (4,5,6);
c := setUnion (a,b);
d := setUnion (a,b);
```

- c의 집합이 d의 집합과 같은가?
- a의 집합을 바꾸면 c의 집합도 바뀔까?
- d의 집합을 바꾸면 c의 집합도 바뀔까?
- e := c를 수행하려면 c를 복사해야 하는가?
- c 를 다 썼으면 그 집합을 없애도 되는가?

항상 공유하는 경우

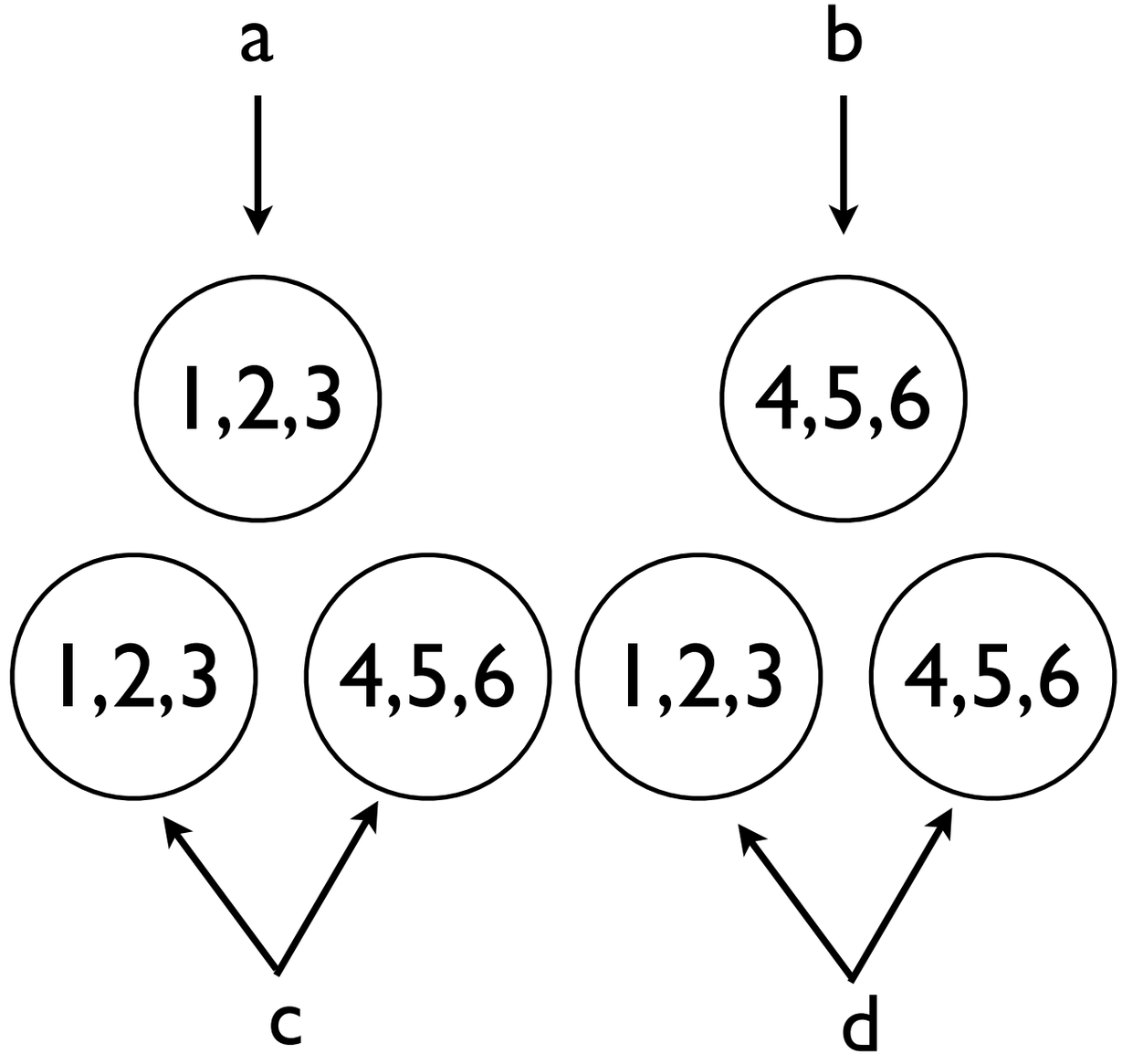
```
a := set (1,2,3);
b := set (4,5,6);
c := setUnion (a,b);
d := setUnion (a,b);
```



- c의 집합이 d의 집합과 같은가?
- a의 집합을 바꾸면 c의 집합도 바뀔까?
- d의 집합을 바꾸면 c의 집합도 바뀔까?
- e := c를 수행하려면 c를 복사해야 하는가?
- c 를 다 썼으면 그 집합을 없애도 되는가?

항상 복사하는 경우

```
a := set (1,2,3);
b := set (4,5,6);
c := setUnion (a,b);
d := setUnion (a,b);
```



- c의 집합이 d의 집합과 같은가?
- a의 집합을 바꾸면 c의 집합도 바뀔까?
- d의 집합을 바꾸면 c의 집합도 바뀔까?
- e := c를 수행하려면 c를 복사해야 하는가?
- c 를 다 썼으면 그 집합을 없애도 되는가?

값 중심의 프로그래밍: 두 방식에서 좋은 것을 취하기

```
a := set (1,2,3);
b := set (4,5,6);
c := setUnion (a,b);
d := setUnion (a,b);
```

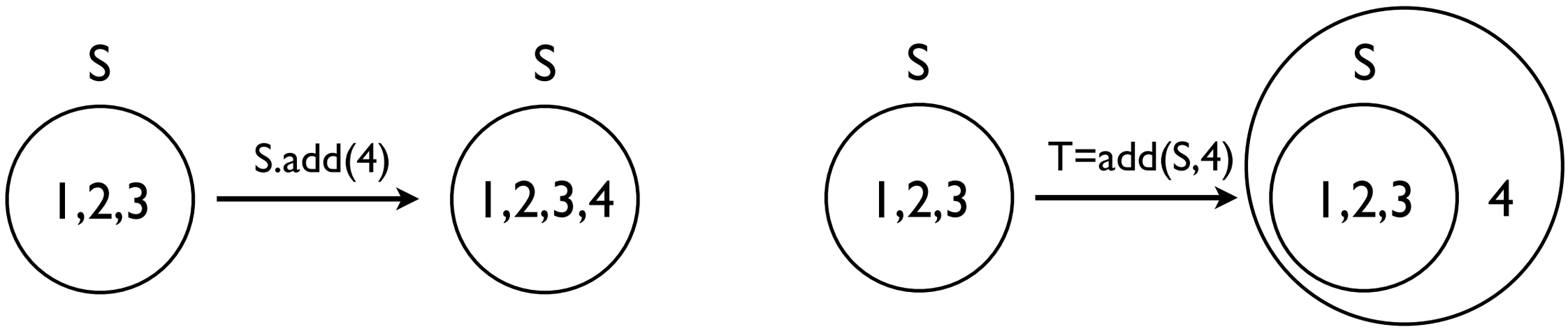
- c의 집합이 d의 집합과 같은가?
- a의 집합을 바꾸면 c의 집합도 바뀔까?
- d의 집합을 바꾸면 c의 집합도 바뀔까?
- e := c를 수행하려면 c를 복사해야 하는가?
- c 를 다 썼으면 그 집합을 없애도 되는가?

- 최대한 공유, 하지만 만들어진 집합들은 변경되지 않도록
- 프로그램 실행 중 만들어지는 집합들

{1,2,3}, {4,5,6}, {1,2,3,4,5,6}

값중심 vs. 물건중심
(value-oriented vs. object-oriented)

- 물건은 변하지만 값은 변하지 않는다.
- S = {1,2,3}
- S.add(4) vs. add(S,4)



수학에서의 서술방식

"V를 $(interior\, U) \cup f^{-1}(W)$ 라고 하자.
그러면 $V \cap (interior\, W) = f(U)$ 가 사실임을 알 수 있다."

- 기본가정: 한번 정의된 값은 변하지 않음
  - 위의 예에서 U,W
- 논증의 옳고 그름을 확인하기 편리 ➡ 수학/과학이 성공한 중요한 인프라

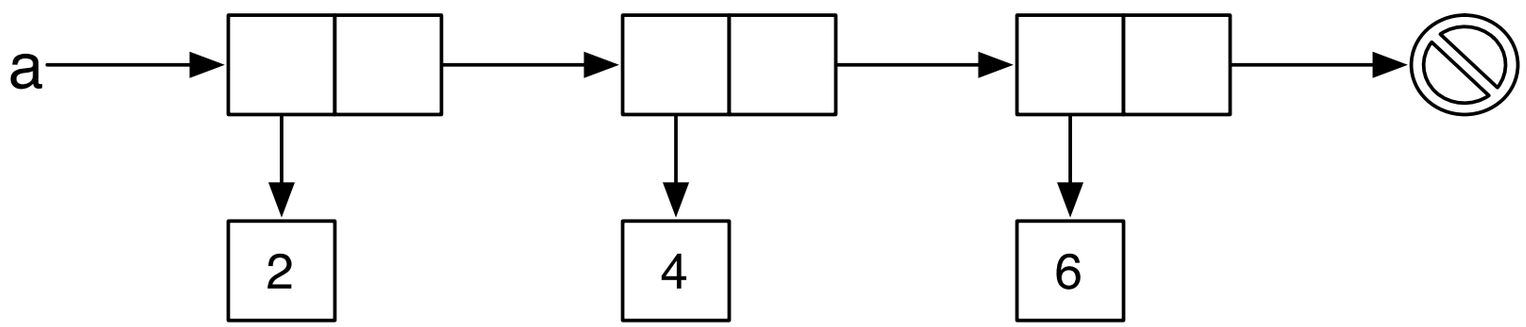
컴퓨터 프로그램은?

```
a := set (1,2,3);
b := set (4,5,6);
c := setUnion (a,b);
d := setRemove (a,1);
```

- d를 생성할 때 a,c의 값도 변경
  - 프로그램 작성/검증이 매우 까다로움
  - 프로그램의 옳고 그름 판명이 어려운 이유중 하나

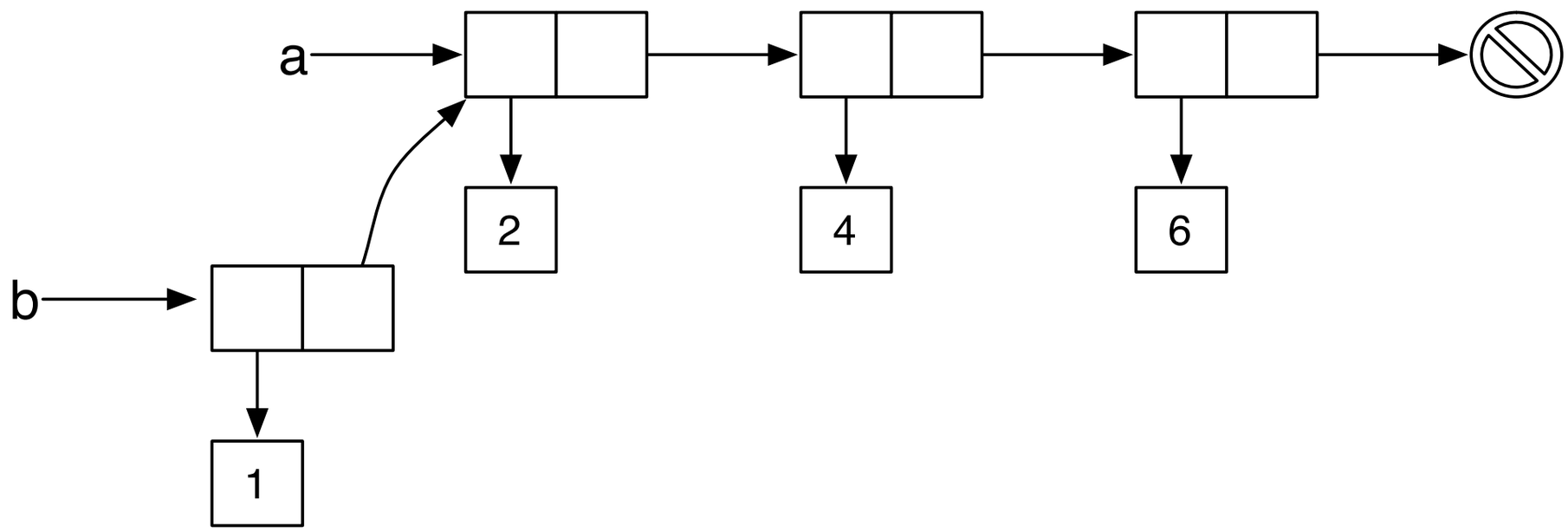
최대한 공유하기

```
a = [2,4,6]
b = 1::a
c = insert (a, 5)
```



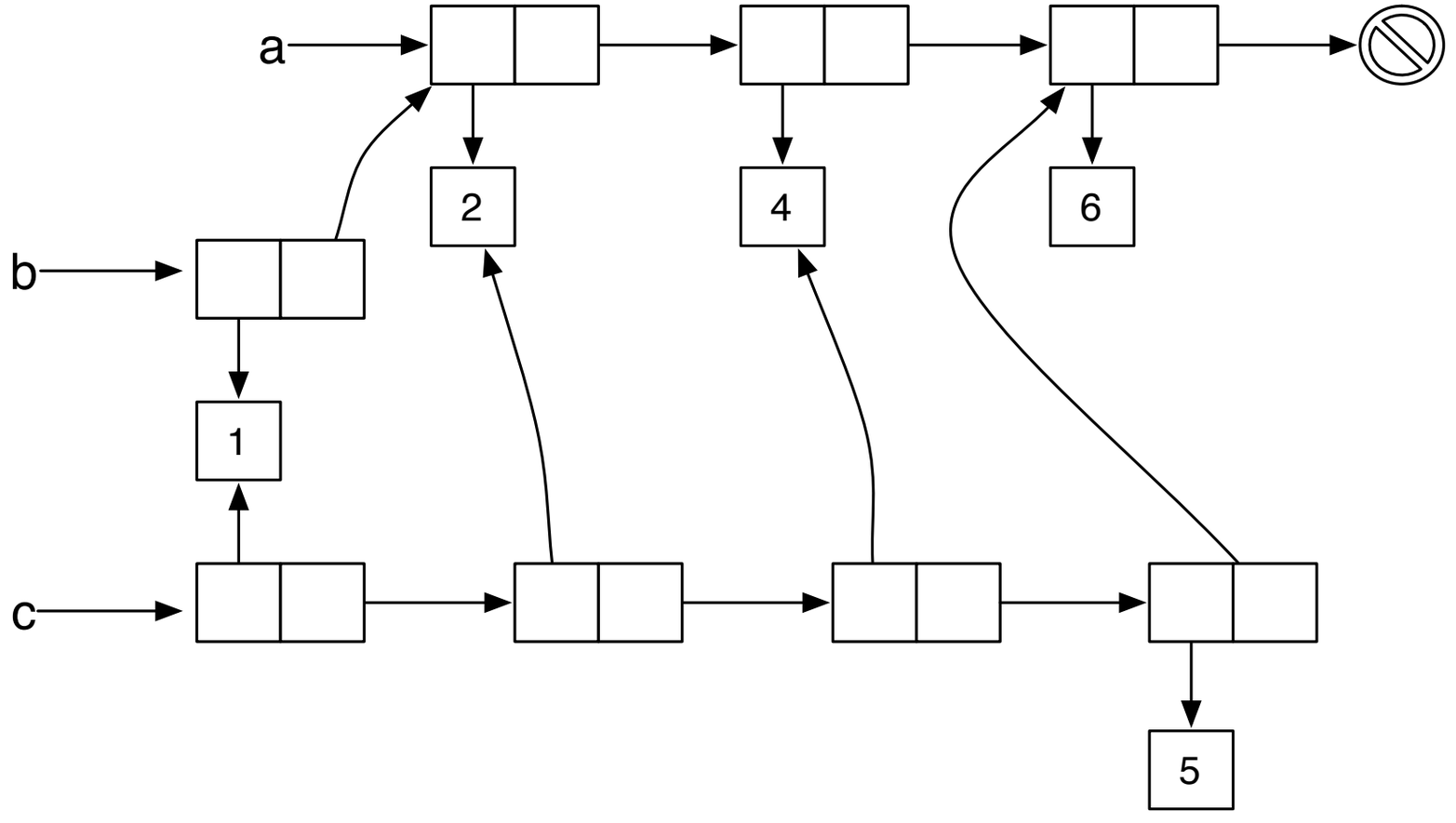
최대한 공유하기

```
a = [2,4,6]
b = 1::a
c = insert (a, 5)
```



최대한 공유하기

```
a = [2,4,6]
b = 1::a
c = insert (a, 5)
```



정리

- 차이
  - 물건중심의 프로그래밍 = 물건(object)을 만들고 변화 시키는 과정
  - 값중심의 프로그래밍 = 값을 정의하고 계산하는 과정
- 쉽고 간단한 소통방식, 프로그램 검증이 보다 쉬워짐
- 충분히 효율적