CS 261: Data Structures

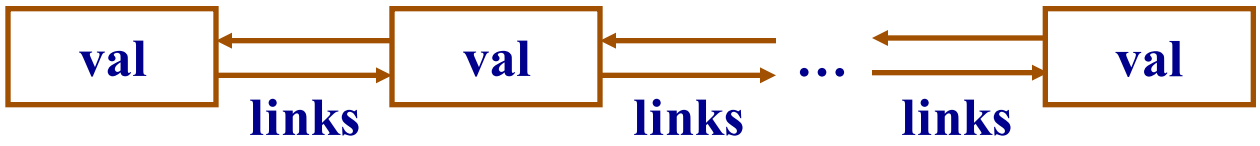
Double Linked List Deque

Double Linked List Bag

Double Links

- Allow access to both next and previous link

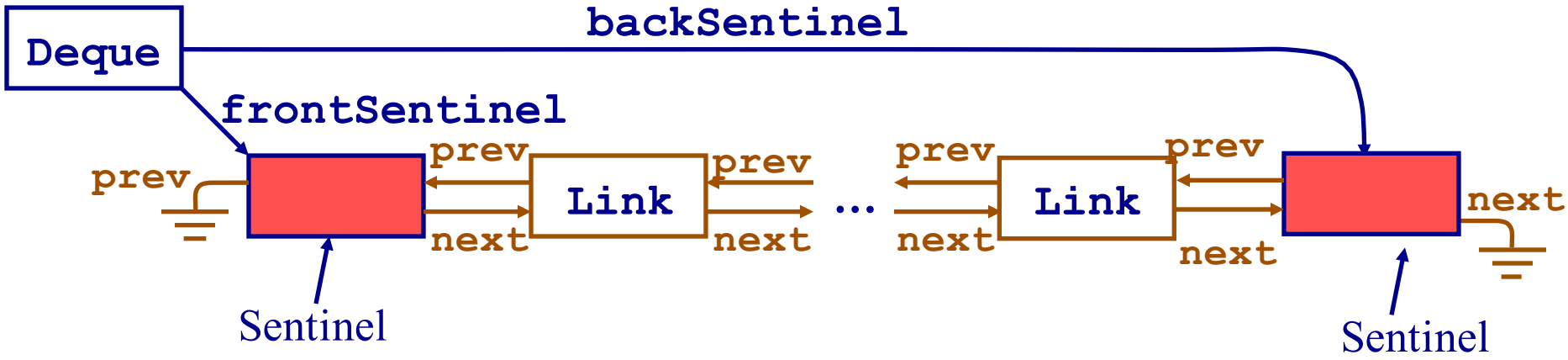
```
struct dlink {
    TYPE value;
    struct dlink * next;
    struct dlink * previous;
};
```



Deque

```
struct dlink{
   TYPE value;
   struct dlink *next;
   struct dlink *prev;
};
```

```
struct listDeque{
   int size;
   struct dlink * frontSentinel;
   struct dlink * backSentinel;
};
```



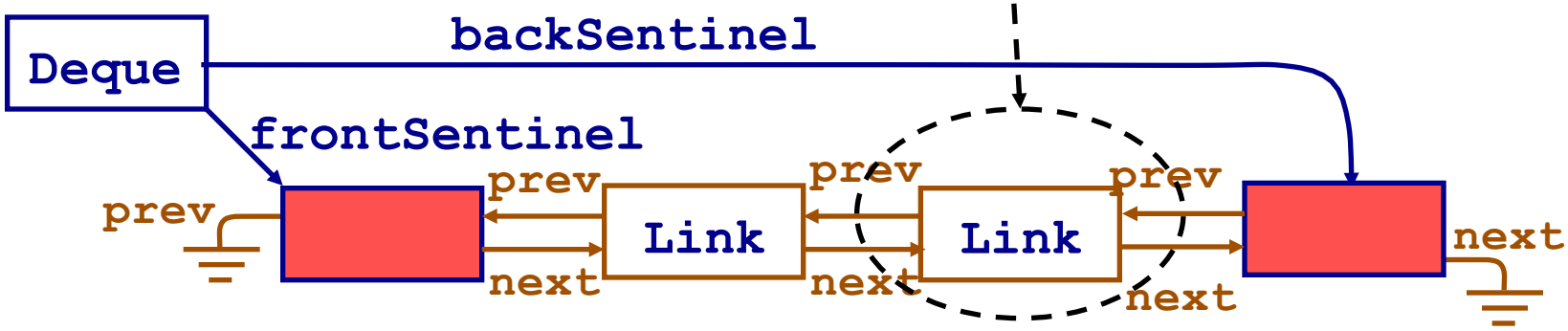
Remove from Front or Back

```
void removeFrontDeque (struct listDeque *dq)
{
      _removeDeque(dq, dq->frontSentinel->next);
}
```

```
void removeBackDeque (struct listDeque *dq)
{
      _removeDeque(dq, dq->backSentinel->prev);
}
```

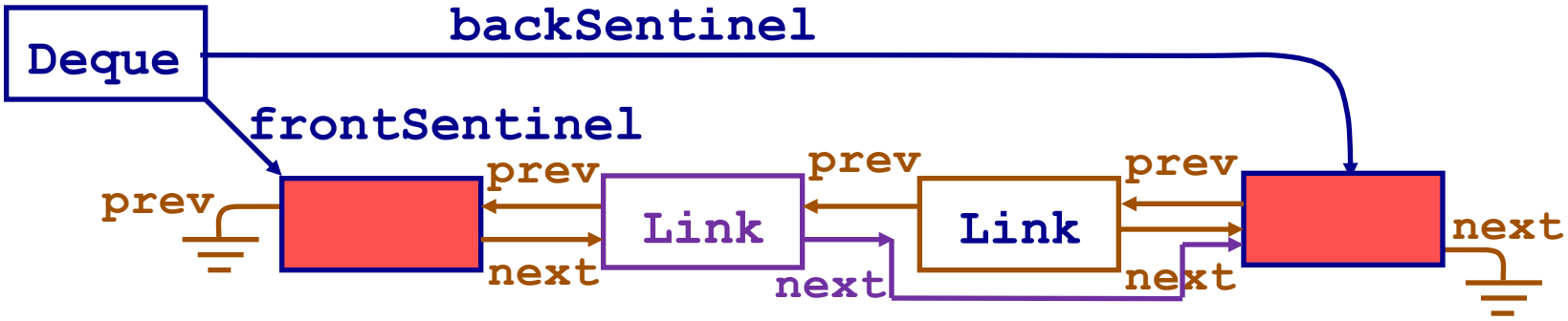
Remove Double Link from Deque

```
void _removeDeque (struct linkedList *dq,
                   struct dlink *lnk)
{
      assert(!isEmptyDeque(dq));




}
```



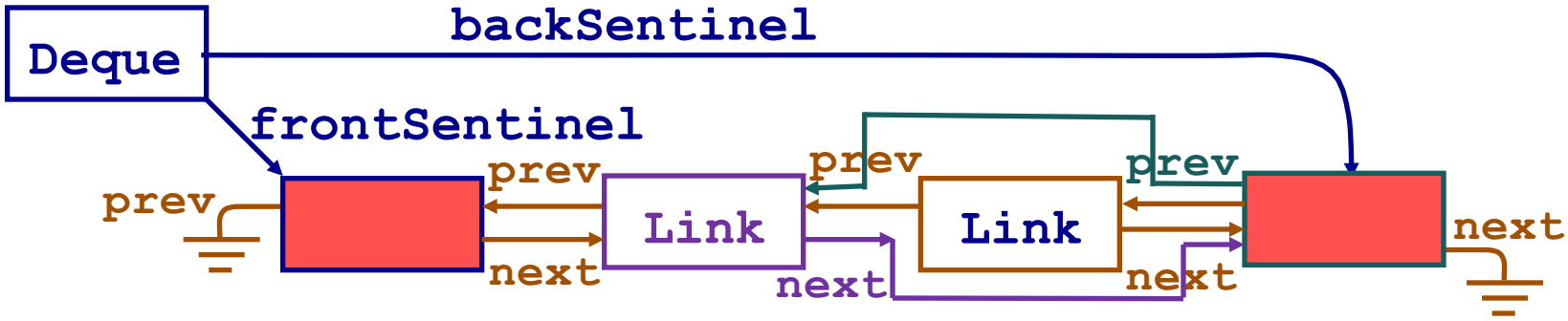
Remove Double Link from Deque

```
void _removeDeque (struct linkedList *dq,
                   struct dlink *lnk)
{
        assert(!isEmptyDeque(dq));
        lnk->prev->next = lnk->next;



}
```



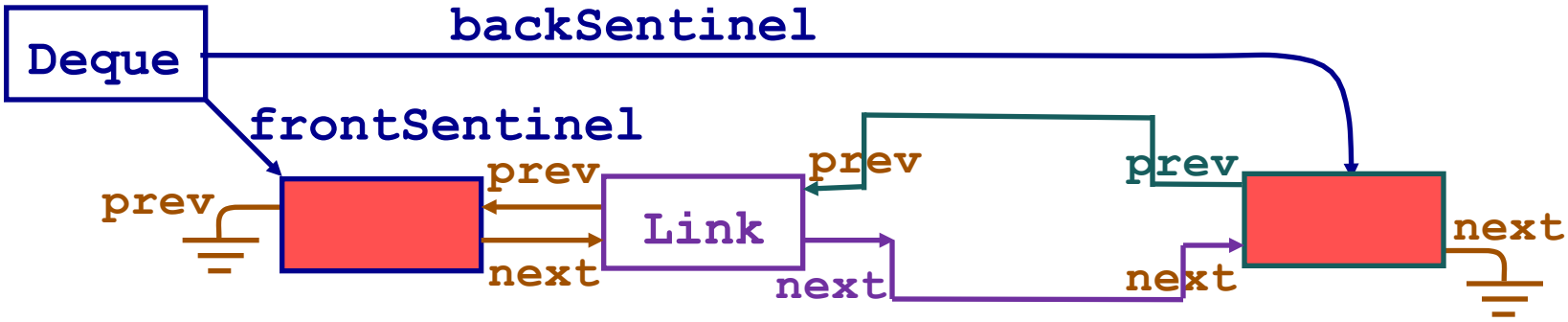
Remove Double Link from Deque

```
void _removeDeque (struct linkedList *dq,
                   struct dlink *lnk)
{
        assert(!isEmptyDeque(dq));
        lnk->prev->next = lnk->next;
        lnk->next->prev = lnk->prev;



}
```



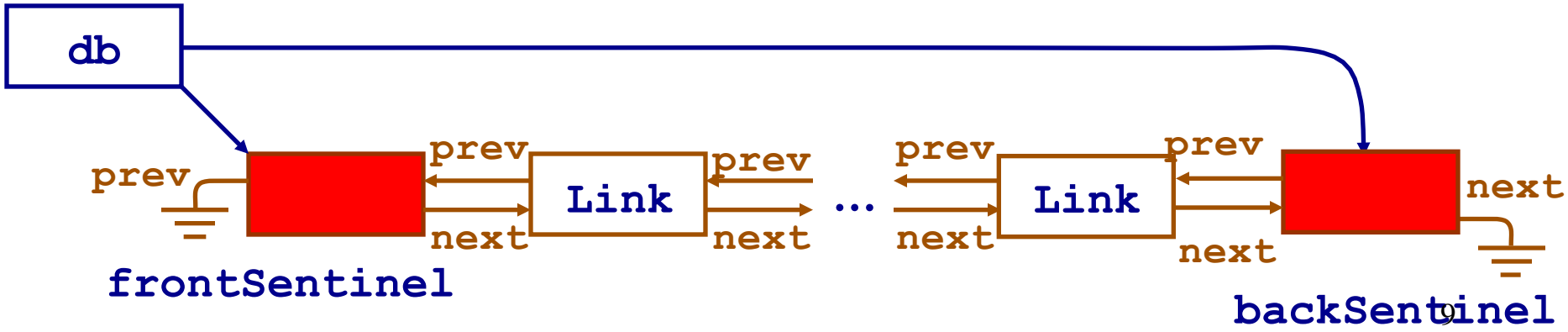
Remove Double Link from Deque

```
void _removeDeque (struct linkedList *dq,
                   struct dlink *lnk)
{
        assert(!isEmptyDeque(dq));
        lnk->prev->next = lnk->next;
        lnk->next->prev = lnk->prev;
        free(lnk);
        dq->size--;
}
```



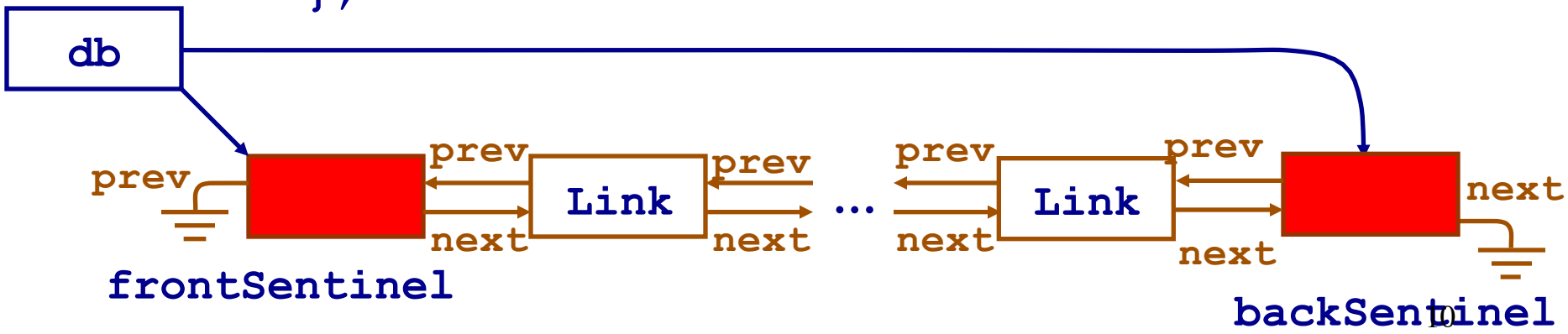
D-Bag

Arbitrary ordering of elements in the collection



D-Bag Structure

```
struct dlink {
    TYPE value;
    struct dlink * next;
    struct dlink * prev;
};
struct listDBag {
    int size;
    struct dlink * frontSentinel;
    struct dlink * backSentinel;
};
```

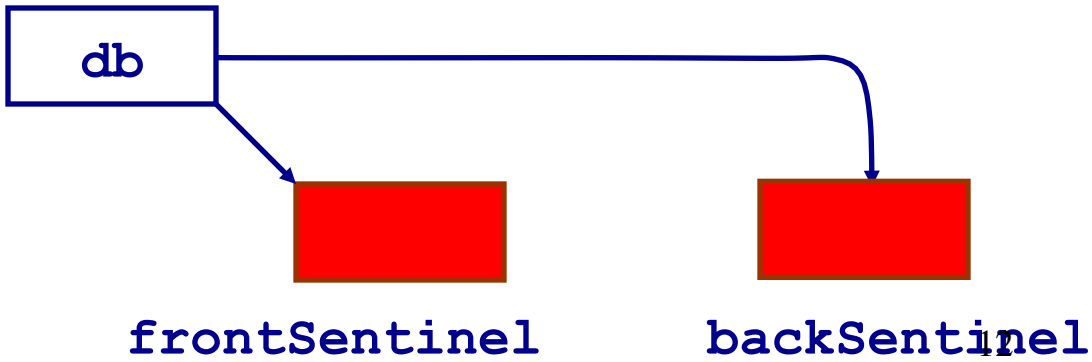


Bag Interface

```
void initBag();     /* Initialize the bag*/

int isEmptyBag();   /* Check if the bag is empty*/

void addBag();      /* Add value*/

void removeBag();   /* Remove value*/

void containsBag();/* Check if a value is in the bag*/
```

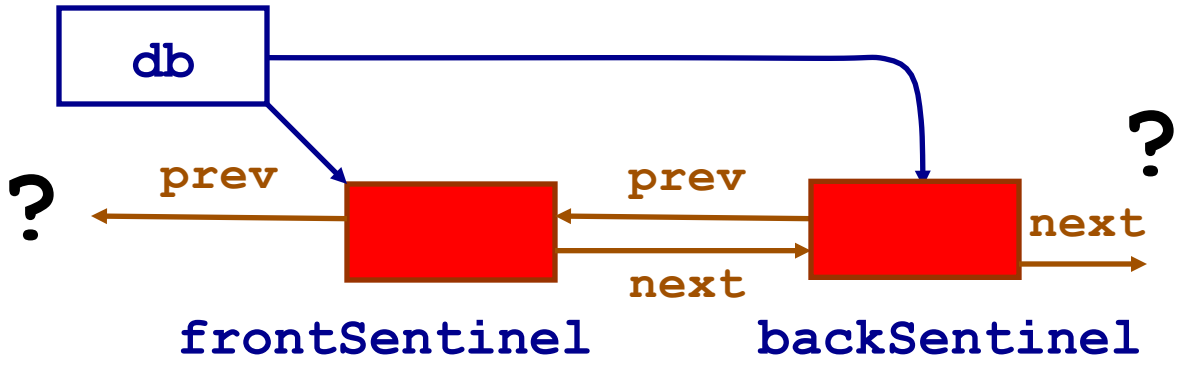
initDBag

```
void initDBag (struct listDBag *db) {
   assert(db);
   db->frontSentinel = (struct dlink *)
                       malloc(sizeof(struct dlink));
   assert(db->frontSentinel != 0);
   db->backSentinel = (struct dlink *)
                       malloc(sizeof(struct dlink));
   assert(db->backSentinel != 0);
   ...

}
```



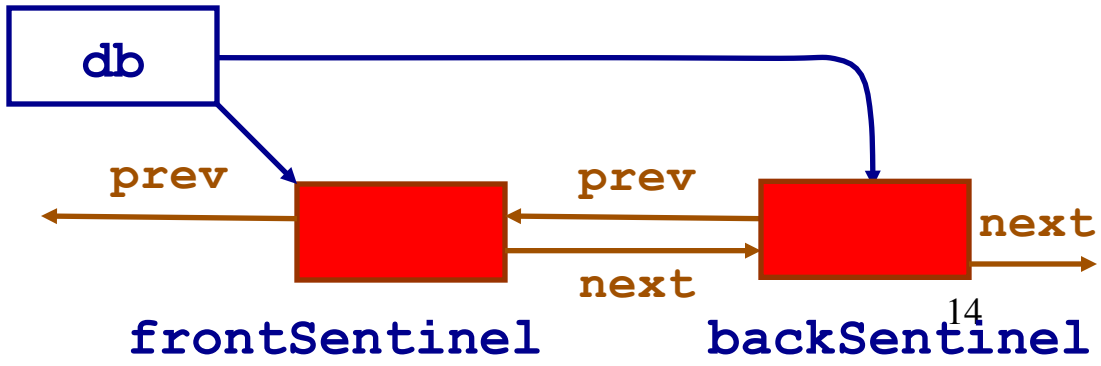
initDBag

```
void initDBag (struct listDBag *db) {
   assert(db);
   db->frontSentinel = (struct dlink *)
                       malloc(sizeof(struct dlink));
   assert(db->frontSentinel != 0);
   db->backSentinel = (struct dlink *)
                       malloc(sizeof(struct dlink));
   assert(db->backSentinel != 0);
   db->frontSentinel->next = db->backSentinel;
   db->backSentinel->prev = db->frontSentinal;
   db->size = 0;
}
```



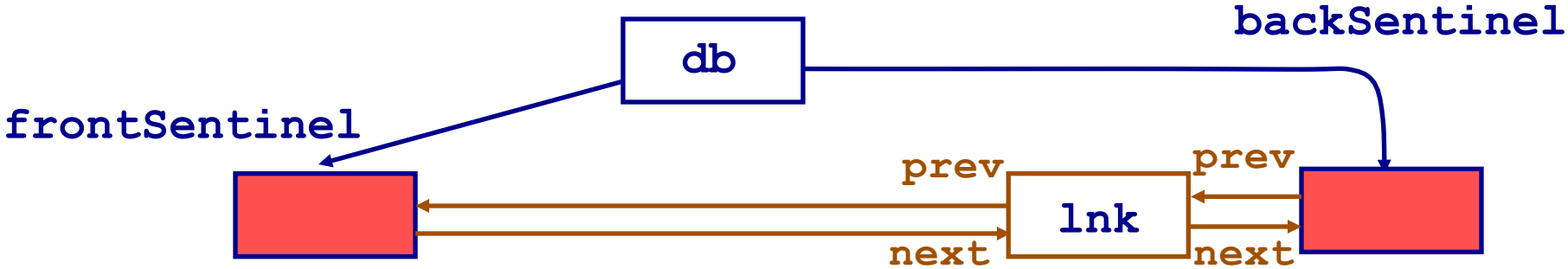
isEmpty

```
void isEmptyDBag (struct listDBag *db){

  assert(db);

  return db->size == 0;

}
```



AddDBag

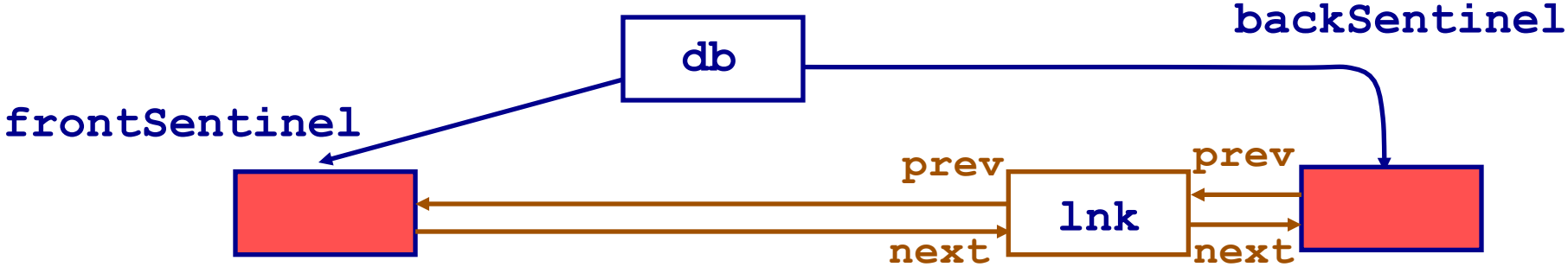
```
void addDBag (struct listDBag *db, TYPE e) {
    assert(db);
    _addDList(db, db->frontSentinel->next, e);
}
```



AddDList

```
void _addDList (struct listDBag *db, struct dlink *lnk, TYPE e) {

}
```

VOLUNTEERS !

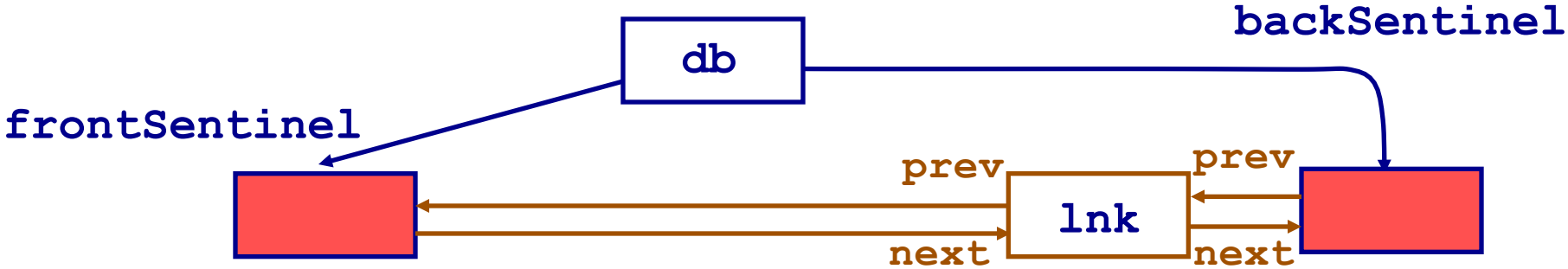


AddDList

```
void _addDList (struct listDBag *db, struct dlink *lnk, TYPE e) {
  struct dlink *new;
  new =(struct dlink *) malloc(sizeof(struct dlink)); assert(new);
  ...

}
```

1st step:

memory allocation

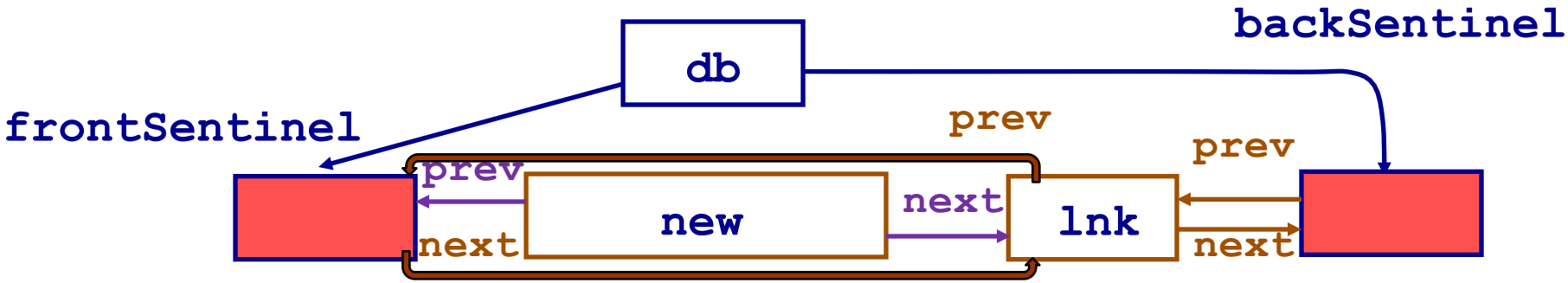


AddDList

```
void _addDList (struct listDBag *db, struct dlink *lnk, TYPE e) {
  struct dlink *new;
  new =(struct dlink *) malloc(sizeof(struct dlink)); assert(new);
  new->value = e; new->next = lnk; new->prev = lnk->prev;
  ...
}
```

2st step:

putting the new link in the bag

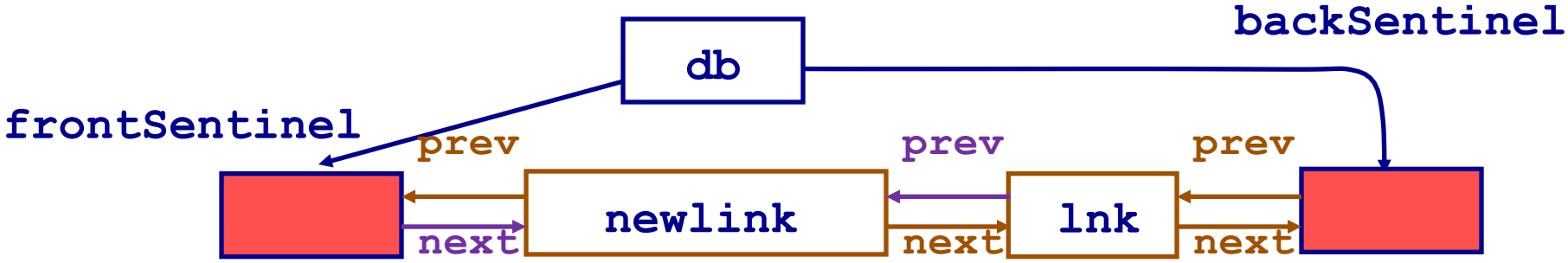


AddDList

```
void _addDList (struct listDBag *db, struct dlink *lnk, TYPE e) {
  struct dlink *new;
  new =(struct dlink *) malloc(sizeof(struct dlink)); assert(new);
  new->value = e; new->next = lnk; new->prev = lnk->prev;
  lnk->prev->next = new; lnk->prev = new;
}
```

3st step:

re-connecting the linked list

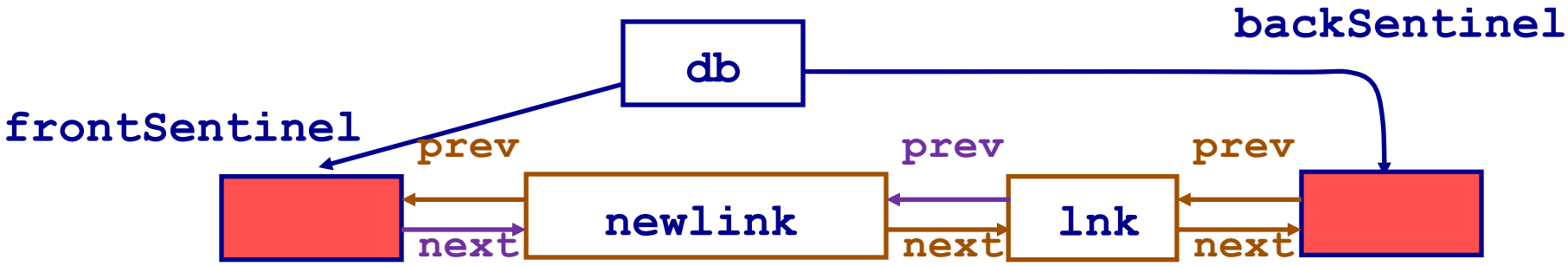


AddDList

```
void _addDList (struct listDBag *db, struct dlink *lnk, TYPE e) {
  struct dlink *new;
  new =(struct dlink *) malloc(sizeof(struct dlink)); assert(new);
  new->value = e; new->next = lnk; new->prev = lnk->prev;
  lnk->prev->next = new; lnk->prev = new;
   db->size++;
}
```

4th step:

increment the size

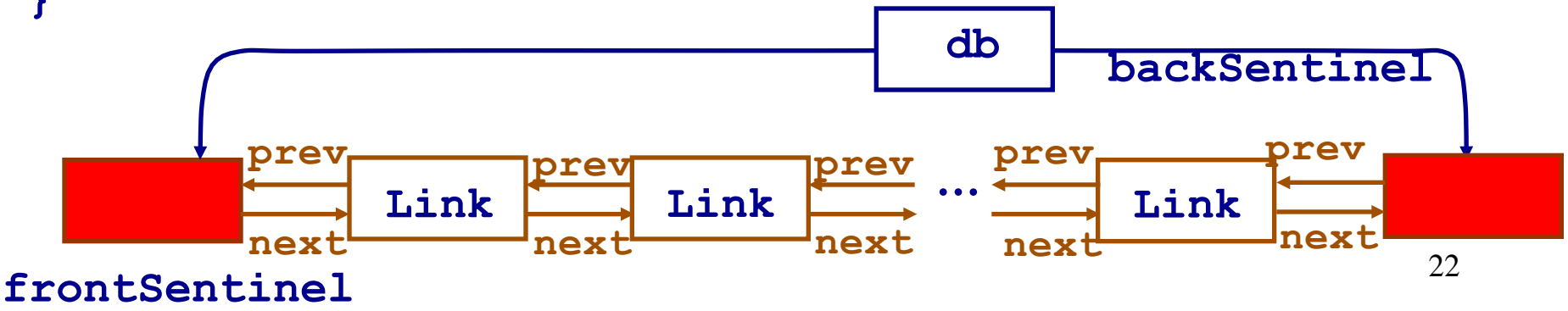


Contains Bag

```
int containsDBag (struct listDBag *db, TYPE e) {

   struct dlink *lnk;

   assert(!isEmptyBag(db));

   /*Find the link of e in the d-linked list bag*/

   lnk = _containsDList(db,e);

   if (lnk) return 1;

   return 0;

}
```

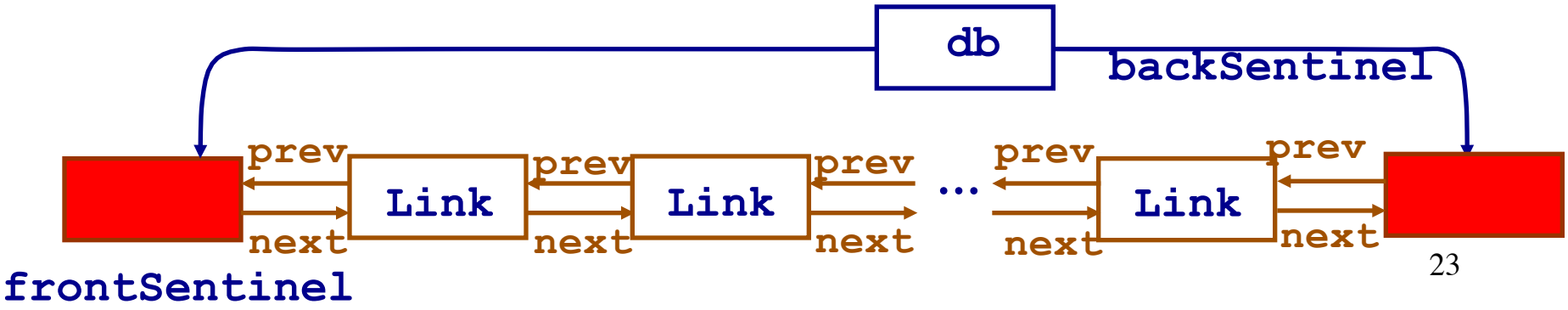
Contains Returning the Pointer

```
struct dlink* _containsDList(struct listDBag *db, TYPE e)
{
    struct dlink *current = db->frontSentinel->next;
    while(current != db->backSentinel){
        if(current->value == e) return current;
        current = current->next;
    }
    return NULL; /* e not found */
}
```



Remove Bag

```
void removeDBag (struct listDBag *db, TYPE e) {
     struct dlink *lnk
     assert(!isEmptyBag(db));
     lnk = _containsDList(db,e);
               ...


}
```

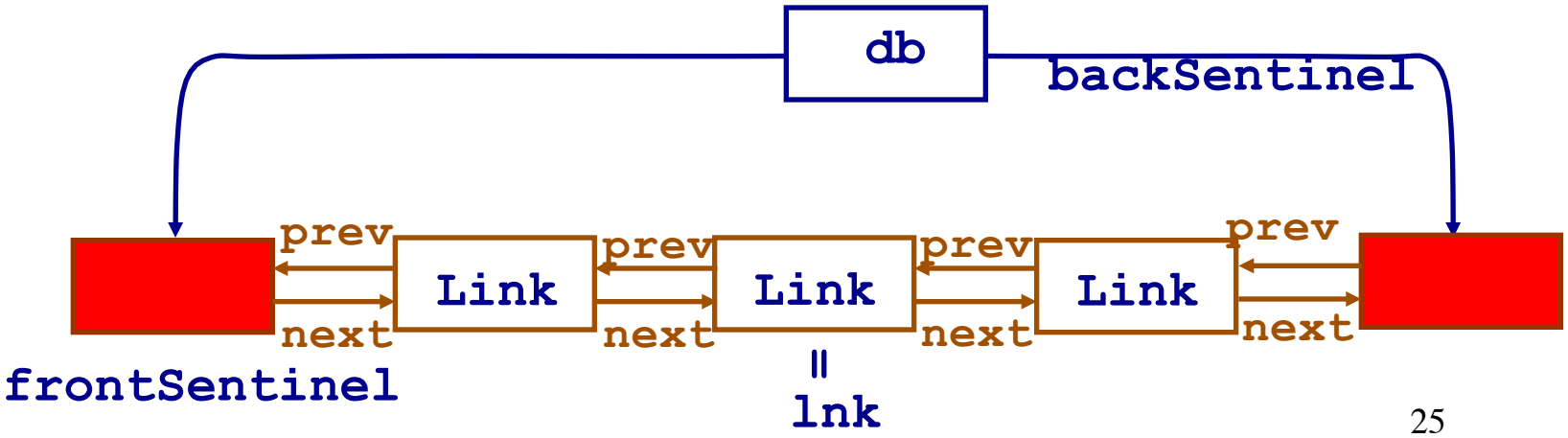


Remove Bag

```
void removeDBag (struct listDBag *db, TYPE e) {
    struct dlink *lnk
    assert(!isEmptyBag(db));
    lnk = _containsDList(db,e);
    /* Removes only 1 copy */
    _removeDLink(db, lnk);
}
```

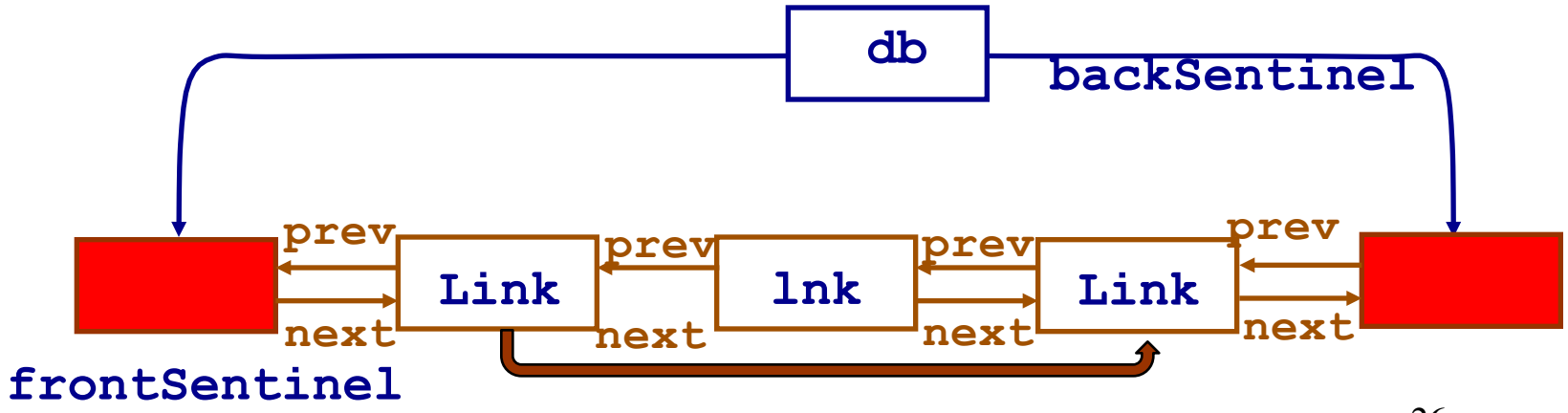
Remove Link

```
void _removeLink (struct listDBag *db, struct dlink *lnk)
{
   /* already checked that db is not empty */

            ...

}
```



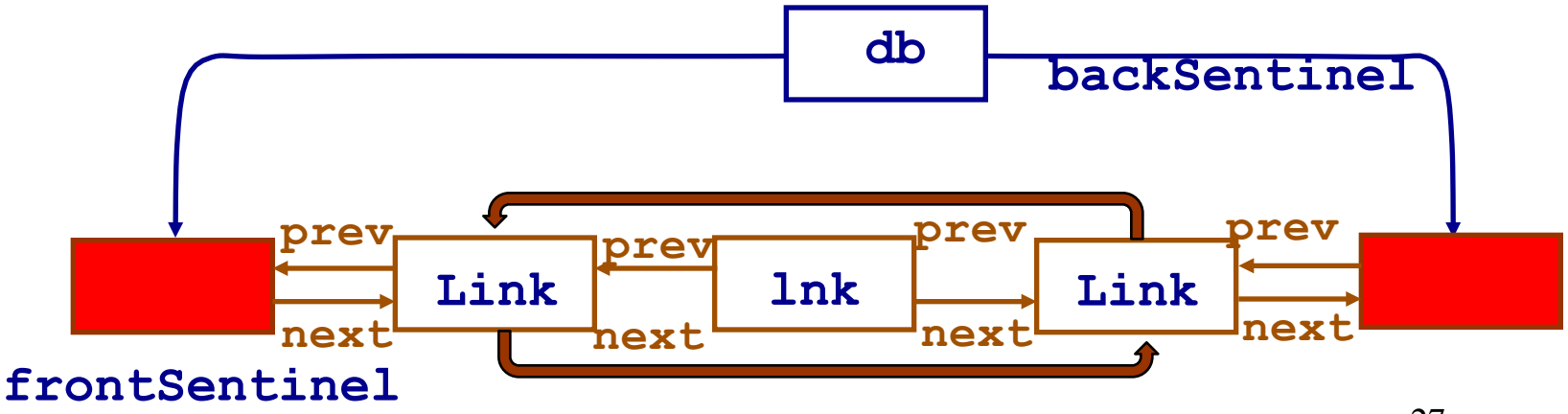
Remove Link

```
void _removeLink (struct listDBag *db, struct dlink *lnk)
{
   /* already checked that db is not empty */
   lnk->prev->next = lnk->next;
            ...


}
```



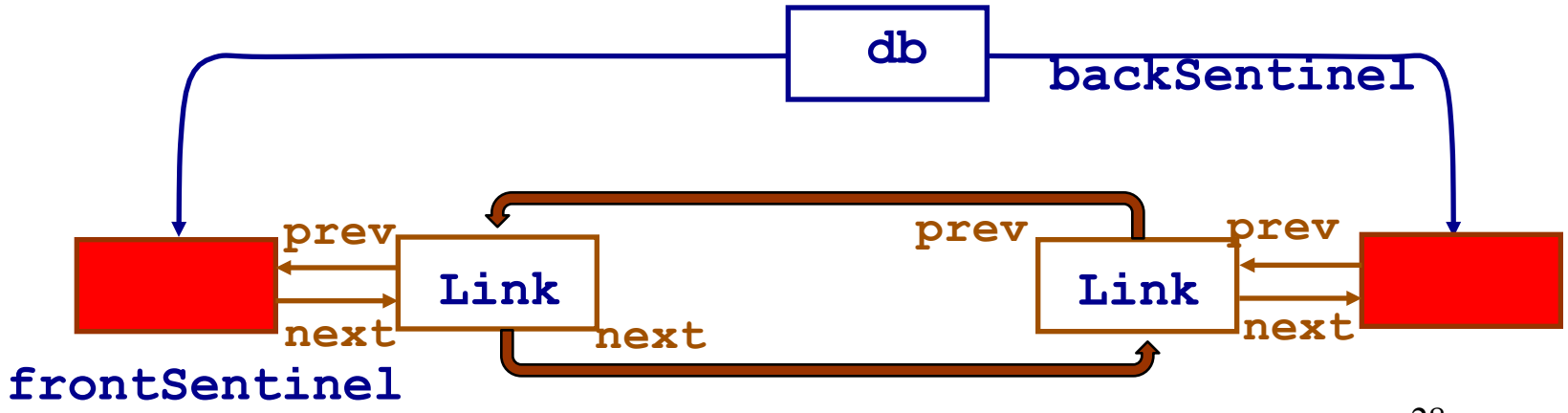
Remove Link

```
void _removeLink (struct listDBag *db, struct dlink *lnk)
{
   /* already checked that db is not empty */
   lnk->prev->next = lnk->next;
   lnk->next->prev = lnk->prev;
            ...
}
```



Remove Link

```
void _removeLink (struct listDBag *db, struct dlink *lnk)
{
   /* already checked that db is not empty */
   lnk->prev->next = lnk->next;
   lnk->next->prev = lnk->prev;
   free(lnk);
   db->size--;
}
```



Remove Bag All Copies

```
void removeDBag (struct listDBag *db, TYPE e) {
    struct dlink *lnk
    assert(!isEmptyBag(db));
    lnk = _containsDList(db,e);
    while(lnk){
       _removeDLink(db, lnk);
       lnk = _containsDList(db,e);
    }
}
```