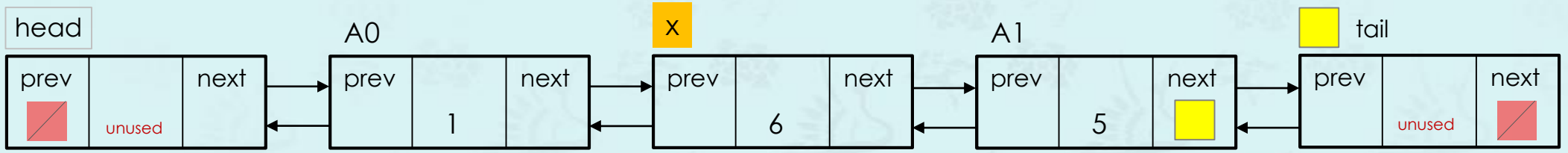
Data Structures
Chapter 4

1. Singly Linked List
2. Doubly Linked List
   - Revisit – Singly Linked List
   - Sentinel Nodes & Basic Operations
   - **Two Key Operations: erase, insert**
   - Advanced Operations

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

Key Operations:

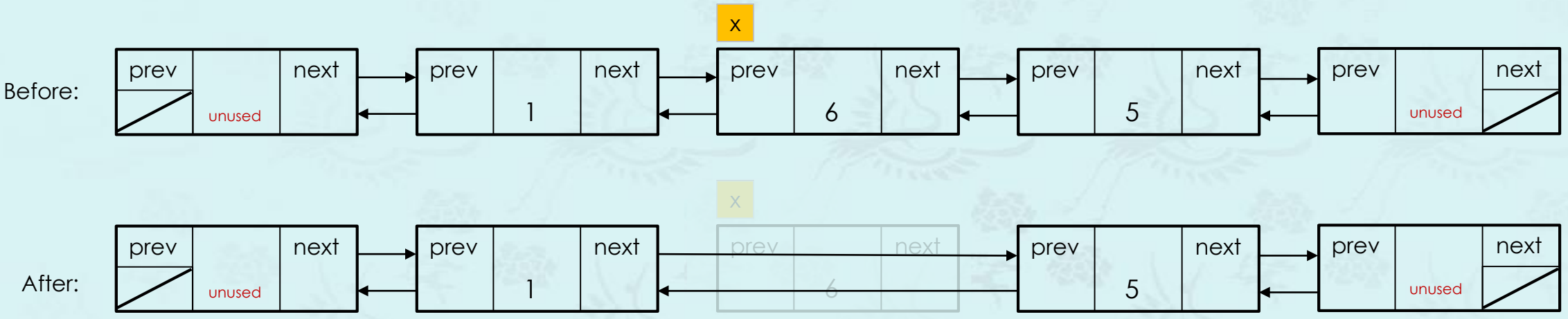



```
pNode find(pList p, int value){
  pNode x = begin(p);
  while(x != end(p) && x->data != value)
    x = x->next;
  return x;
}
```

- Can we reduce the number lines by two?

```
pNode find(pList p, int value){
  pNode x = begin(p);
  while(x != end(p)) {
    if (x->data == value) return x;
    x = x->next;
  }
  return x;
}
```

Key Operations: Erase the node x



recall x alone is known

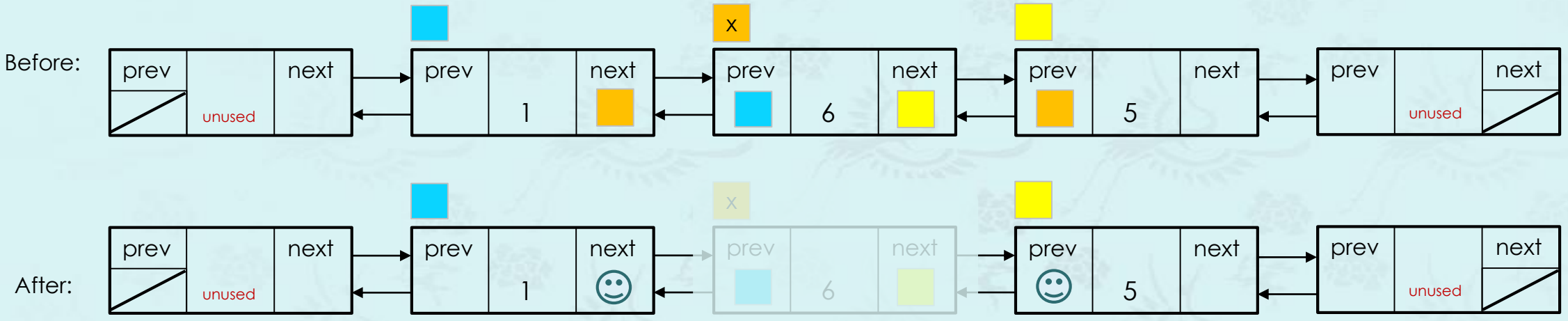
recall x is not pList, but pNode

```
void erase(pNode x){



}
```

Key Operations: Erase the node x



recall x alone is known

recall x is not pList, but pNode

```
void erase(pNode x){



}
```

Key Operations: Erase the node x

Before:

After:

Before:

x == x
==
==
==
==

Express each color code in terms of x.

recall x alone is known

recall x is not pList, but pNode

```
void erase(pNode x){



}
```

Key Operations: Erase the node x

Before:

After:

Before:

```
                x == x
x->prev->next ==
x->next->prev ==
              ==
              ==
```

```
void erase(pNode x){

}
```

recall x alone is known

recall x is not pList, but pNode

Key Operations: Erase the node x

Before:

After:

Before:

```
                x == x
x->prev->next ==
x->next->prev ==
      x->prev ==
      x->next ==
```

recall x alone is known

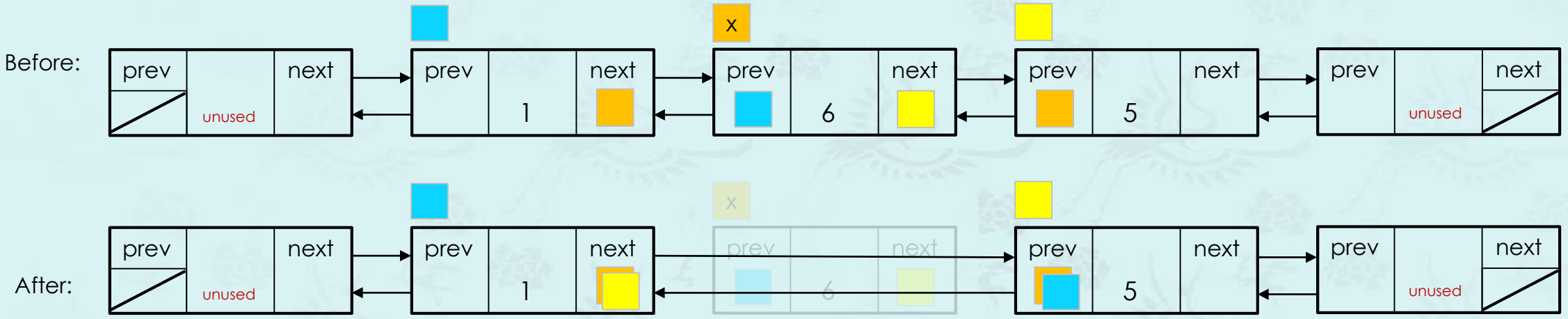
recall x is not pList, but pNode

```
void erase(pNode x){



}
```

Key Operations: Erase the node x



Before:

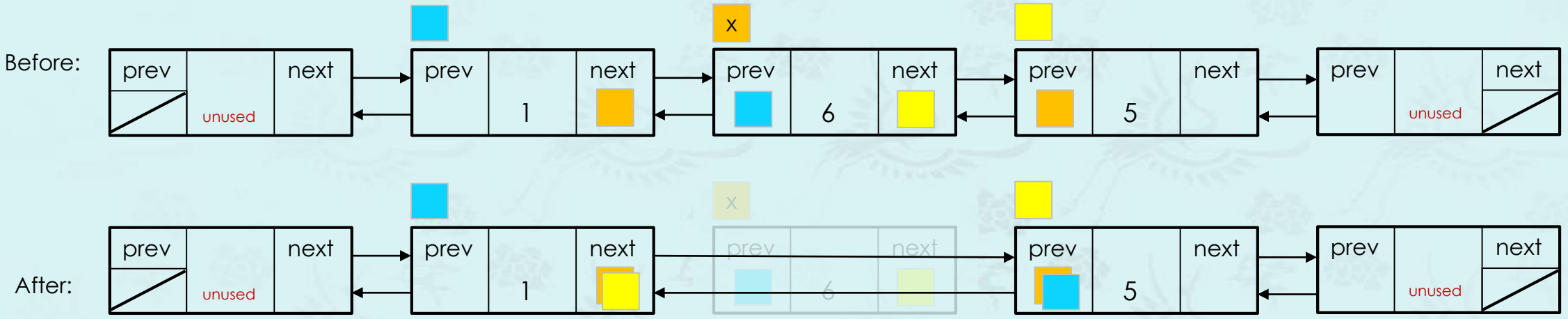
```
                x == x
x->prev->next ==
x->next->prev ==
      x->prev ==
      x->next ==
```

```
void erase(pNode x){



}
```

Key Operations: Erase the node x



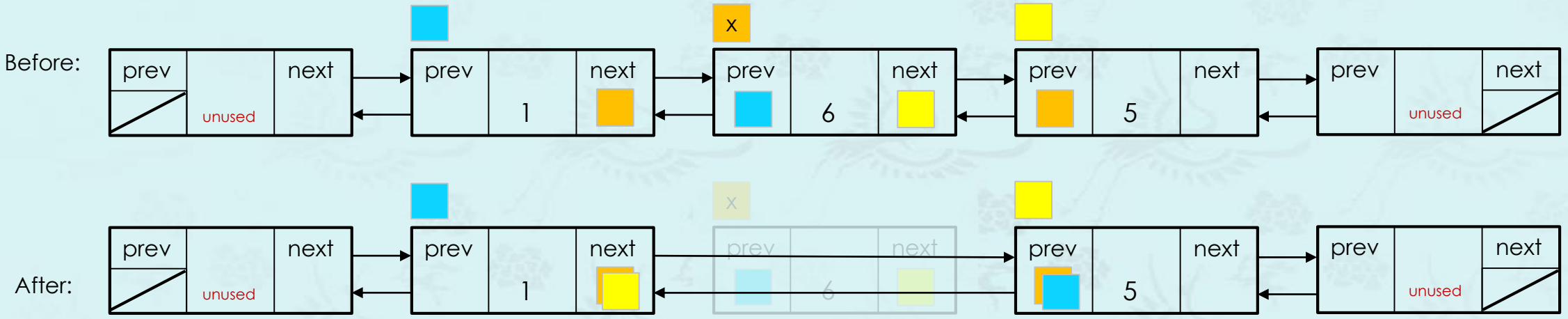
Before:

```
                x == x
x->prev->next ==
x->next->prev ==
      x->prev ==
      x->next ==
```

```
void erase(pNode x){
  x->prev->next = x->next;
  x->next->prev = x->prev;

}
```

Key Operations: Erase the node x



Before:

```
                x == x
x->prev->next ==
x->next->prev ==
      x->prev ==
      x->next ==
```

```
void erase(pNode x){
  x->prev->next = x->next;
  x->next->prev = x->prev;
  delete x;
} //stay tuned for the better
```

Pop by value

- Implement `pop()` using `erase()` and `find()`.

```
void pop(pList p, int value){
  pNode node = find(p, value);
  erase(node);
}
```

```
void erase(pNode x){
  x->prev->next = x->next;
  x->next->prev = x->prev;
  delete x;
} //stay tuned for the better
```

```
pNode find(pList p, int value)
```

Pop by value

- Implement `pop()` using `erase()` and `find()`.

```
void pop(pList p, int value){
  pNode node = find(p, value);
  erase(node);
}
```

```
void pop(pList p, int value){
  erase(find(p, value));
}
```
//stay tuned

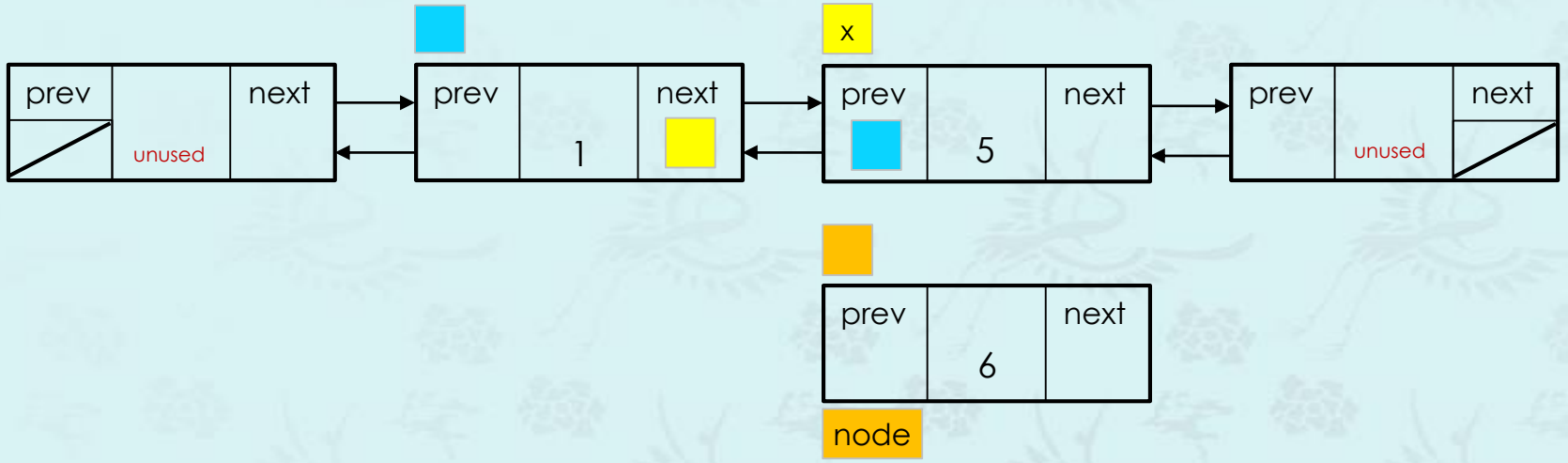
This code may not work some cases?
How can you fix it?

```
void erase(pNode x){
  x->prev->next = x->next;
  x->next->prev = x->prev;
  delete x;
} //stay tuned for the better
```

```
pNode find(pList p, int value)
```

Key Operations: Insert a new node(value) **in place of the node x**

Before:



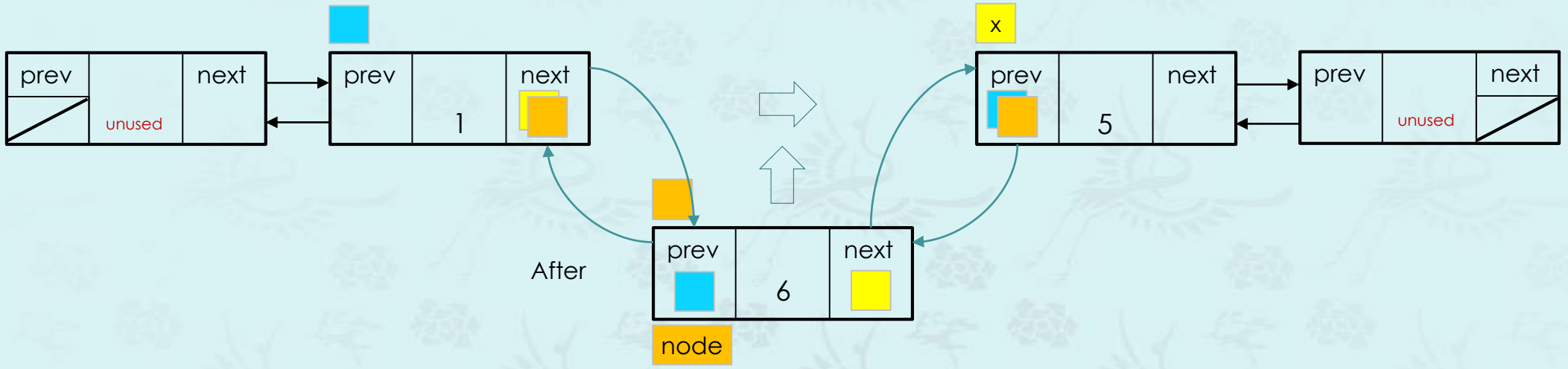

```
void insert(pNode x, int value){



}
```

Key Operations: Insert a new node(value) **in place of the node x**

Before:



Before:

```
                x ==
x->prev->next ==
        x->prev ==
```

Express each color code in terms of x.

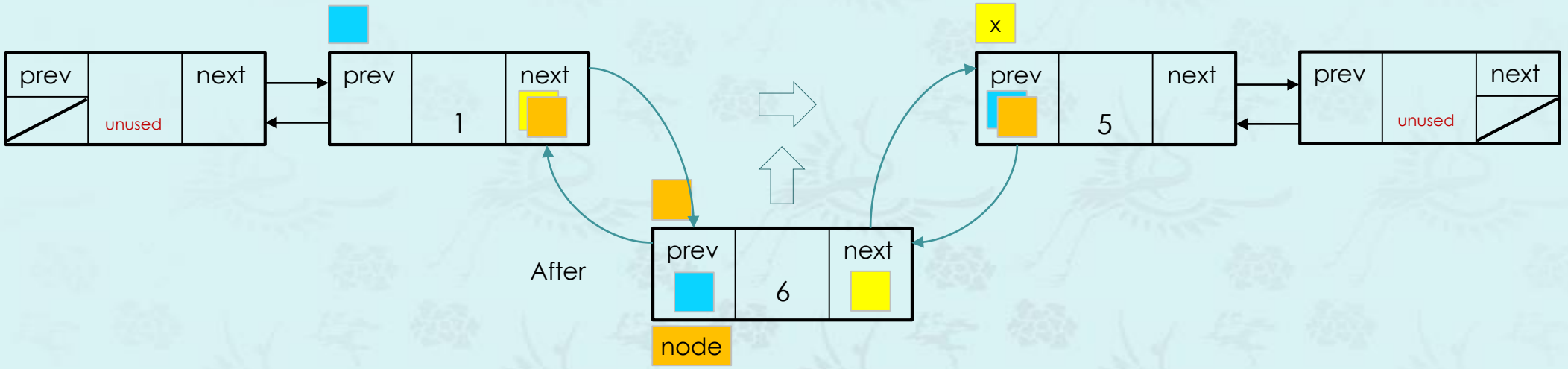
```
void insert(pNode x, int value){



}
```

Key Operations: Insert a new node(value) **in place of the node x**

Before:



Before:

```
                x ==
x->prev->next ==
        x->prev ==
```

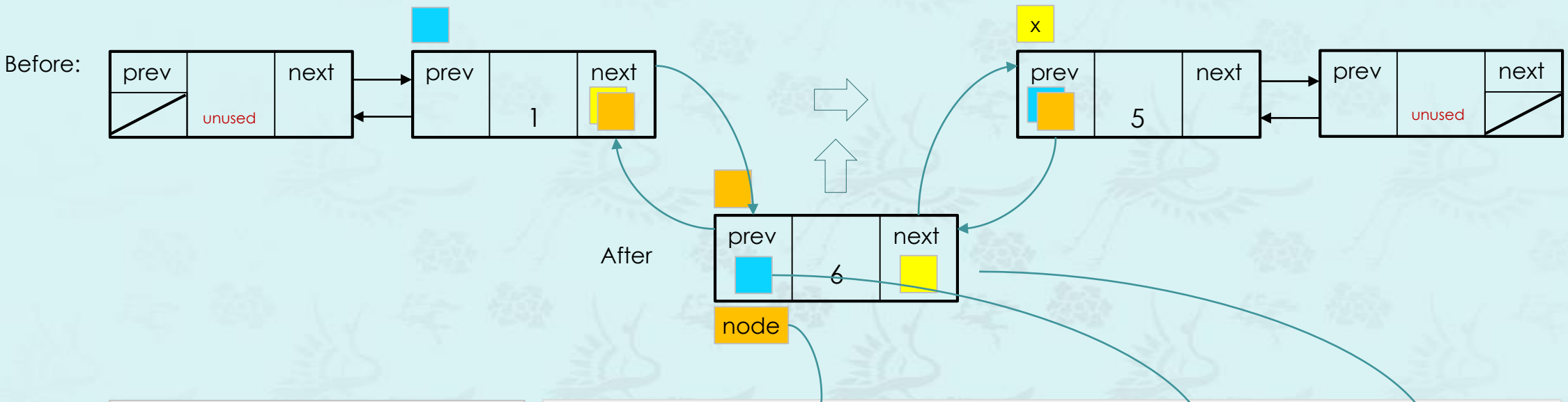
Express each color code in terms of x.

```
void insert(pNode x, int value){

}
```

node

Key Operations: Insert a new node(value) **in place of the node x**



Before:

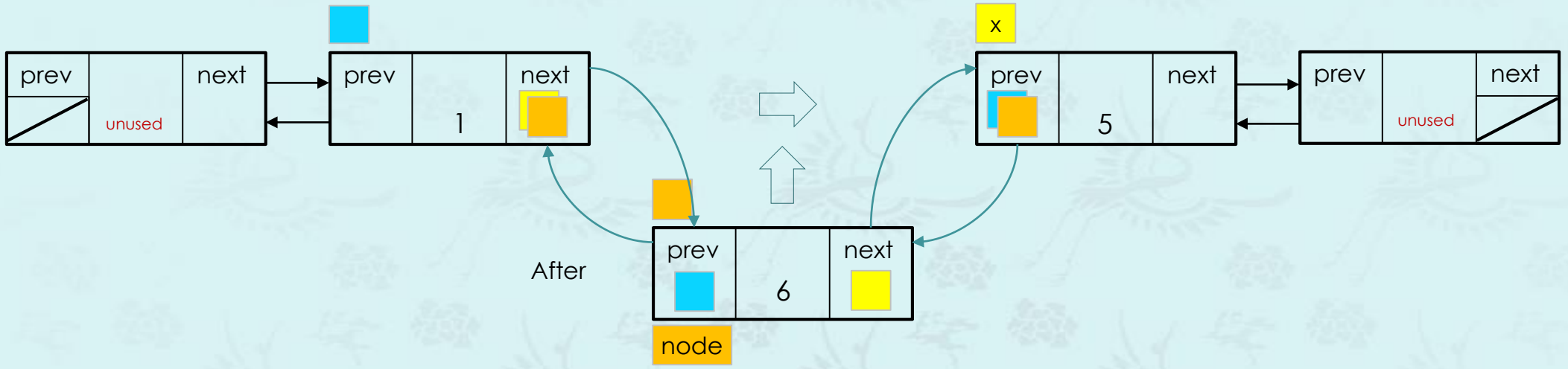
```
                x ==
x->prev->next ==
        x->prev ==
```

Express each color code in terms of x.

```
void insert(pNode x, int value){
  pNode          = new Node{value,

}
```

Key Operations: Insert a new node(value) **in place of the node x**



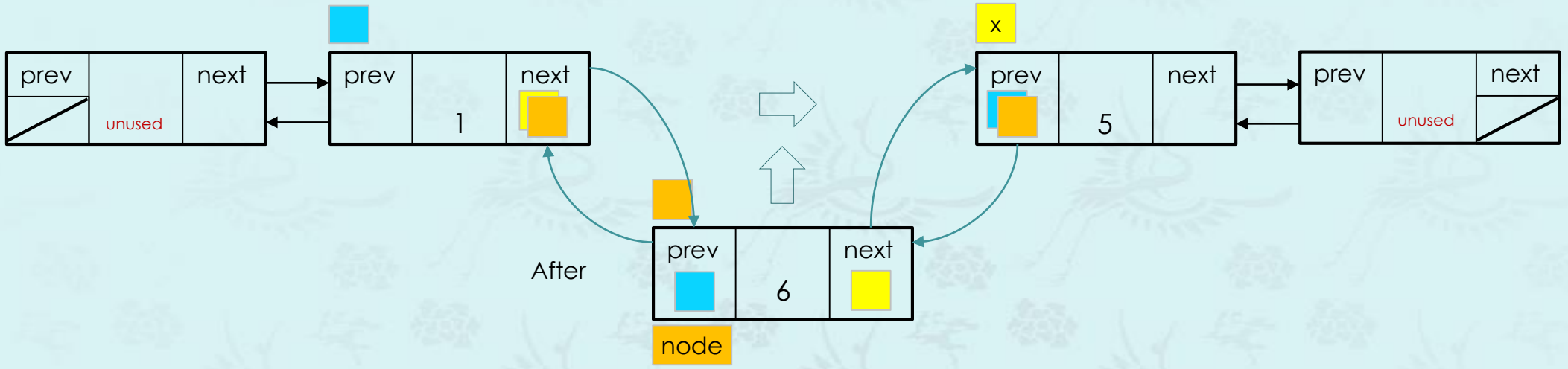
Before:

```
            x ==
x->prev->next ==
      x->prev ==
```

Express each color code in terms of x.

```
void insert(pNode x, int value){
  pNode node = new Node{value, x->prev, x};
  x->prev->next = node;
  x->prev = node;
}
```

Key Operations: Insert a new node(value) **in place of the node x**



Before:

```
              x == x
x->prev->next ==
      x->prev ==
```

Express each color code in terms of x.

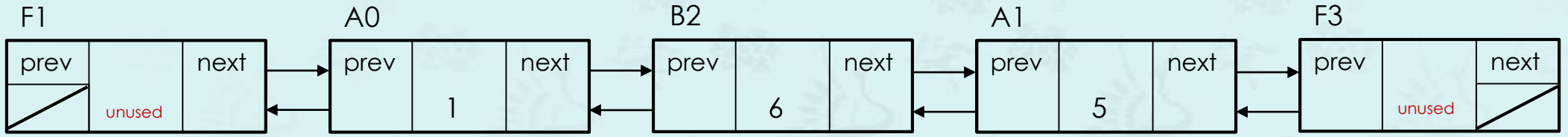
```
void insert(pNode x, int value){
  pNode node = new Node{value, x->prev, x};
  x->prev->next = node;
  x->prev = node;
}
```

Can we replace two lines above with _____

It is a matter of associativity!
Then, what is the associativity of '='?
right to left

```
(1) x->prev = x->prev->next = node;
(2) x->prev->next = x->prev = node;
(3) either one
```

push_front(), push_back(), push()



Case 1: in front

```
void push_front(pList p, int value){
  insert(begin(p)  , value);
}
```

Case 3: at end

```
void push_back(pList p, int value){
  insert(end(p)    , value);
}
```

Case 2: in middle

```
void push(pList p, int value, int z){
  insert(find(p, z), value);
}
```

```
void insert(pNode x, int value){
  pNode node = new Node{value, x->prev, x};
  x->prev = x->prev->next = node;
}
```

```
pNode begin(pList p);                          // returns the first node, not sentinel node
pNode end(pList p);                            // returns the ending sentinel node
pNode half(pList p)                            // returns the node in the middle of the list
pNode find(pList p, int value);                // returns the first node with value
void clear(pList p);                           // free list of nodes
bool empty(pList p);                           // true if empty, false if no empty
int size(pList p);                             // returns size in the list
void insert(pNode x, int value);               // inserts a new node with value at the node x
void erase (pNode x);                          // deletes a node and returns the previous node

void push(pList p, int value, int z);     // inserts a node with value at the node with x
void push_front(pList p, int value);      // inserts a node at front of the list
void push_back(pList p, int value);       // inserts a node with value at end of the list
void push_sorted(pList p, int value, bool ascending = true); // inserts a node in sorted
void pop(pList p, int value);             // deletes the first node with value
void pop_front(pList p);                  // deletes the first node in the list
void pop_back(pList p);                   // deletes the last node in the list, O(1)
void pop_backN(pList p);                  // deletes all the nodes O(n)
void pop_all(pList p, int value);         // deletes all the nodes with value

pList sort(pList p);                      // returns a `new list` sorted
bool sorted(pList p);                     // returns true if the list is sorted
void unique(pList p);                     // returns list with no duplicates, sorted
void reverse(pList p);                    // reverses the sequence
void shuffle(pList p);                    // shuffles the list
void show(pList p);                       // shows all data items in linked list
```

stay tuned for enhancement

Data Structures
Chapter 4

1. Singly Linked List
2. Doubly Linked List
   - Revisit – Singly Linked List
   - Sentinel Nodes & Basic Operations
   - Two Key Operations: erase, insert
   - **Advanced Operations**

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University