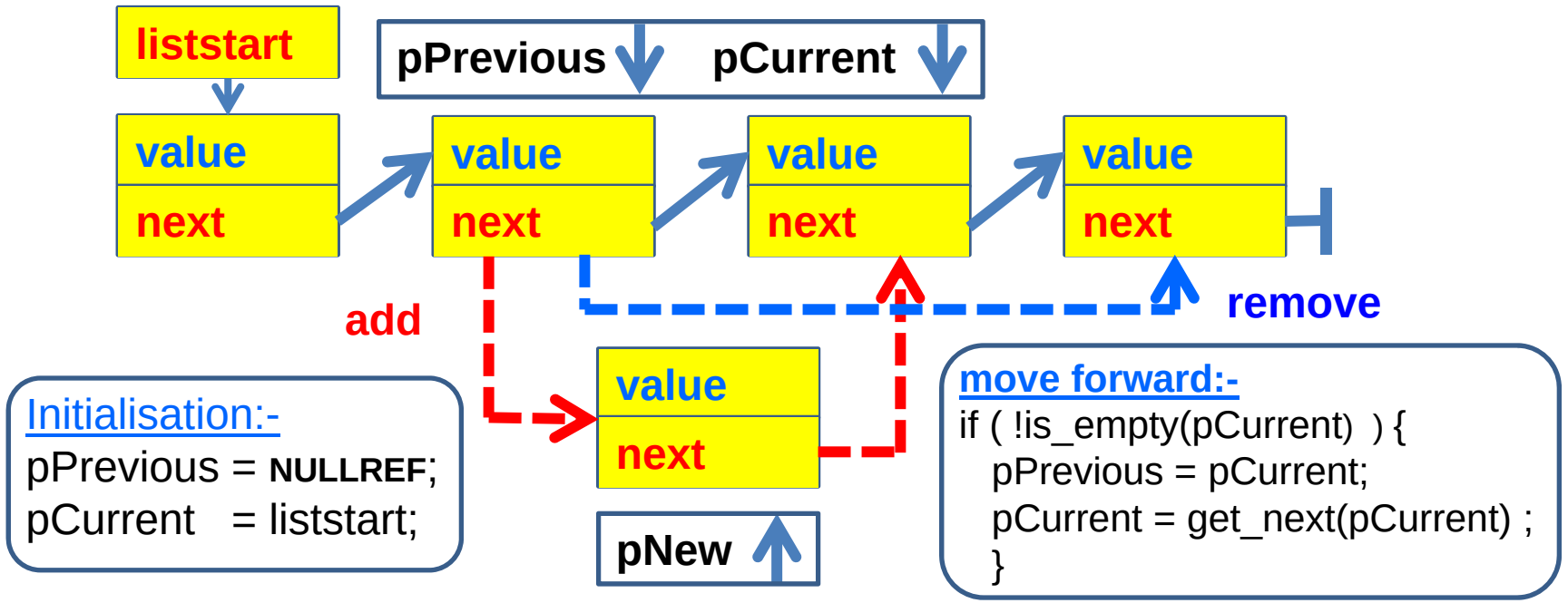
Iteration:
The role of pPrevious, pCurrent, pNew



(pPrevious, pCurrent) move as a pair along the list **(used in add /find/ remove)**

pNew is inserted between pPrevious and pCurrent **(used in add)**

Iteration: Navigation functions

- Navigation functions (using pPrevious & pCurrent)

```
void get_Seq_first()   { pPrevious = NULLREF; pCurrent = liststart; }
int    is_Seq_empty()  { return is_empty(pCurrent); }
void get_Seq_next()   {
   if ( !is_Seq_empty() ) {            // ➔ pCurrent != NULLREF
      pPrevious = pCurrent;
      pCurrent   = get_next(pCurrent);
      }
   }
```

pPrevious & pCurrent have been hidden (abstracted away)

- Navigation (iteration) through the list

```
get_Seq_first();
while ( !is_Seq_empty() ) { /* process element */ get_Seq_next();   }
```

Iteration versus recursion: count

```
int be_size_seq () {
int size=0;
get_seq_first();
while ( !is_seq_empty() ) { size++; get_seq_next(); }
return size;
}
```

```
Pattern:-
get_Seq_first();
while ( !is_Seq_empty() ) { /* process element */ get_Seq_next();   }
```

Iteration versus recursion: count

```
int be_size_seq(seqref S)
{ return is_empty(S) ? 0 : 1 + be_size_seq(tail(S)); }
```

empty list ➔ size 0

non-empty list ➔ size 1 (head) + size(tail)

Pattern:-

- **The empty case - non-recursive – stop condition**
- **The non-empty case - non-recursive – the head**
- **The recursive case - the tail**

Iteration versus recursion: add

```
void link_in(seqref pNew) {  if (!is_empty(pNew)) {
      set_next(pNew, pCurrent);
      if (is_empty(pPrevious)) liststart = pNew; else set_next(pPrevious, pNew);
      }
}
void be_add_val(valtype val) {
get_Seq_first();                          // navigate to correct position
while (!is_Seq_empty() && (val > get_Element_value()) )   get_Seq_next();
link_in(create_element(val));             // add the new element
}
```

Pattern:-

```
get_Seq_first();
while ( !is_Seq_empty() ) { /* process element */ get_Seq_next();   }
```

Iteration versus **recursion**: add

```
static seqref be_add_val(seqref S, valtype v)
{
 return is_empty(S)           ?   create_e(v)
  :   v < get_value(head(S)) ?   cons(create_e(v), S)
  :                               cons(head(S), be_add_val(tail(S),v));
}
```

cons: construct a list from the HEAD & TAIL

Pattern:-

- The empty case - non-recursive – stop condition
- The non-empty case - non-recursive – the head + L
- The recursive case - old head + the modified tail

Iteration versus recursion: find

```
seqref be_find_val( valtype val)  {
   get_Seq_first();
   while (!is_Seq_empty() && (val != get_Element_value()) )  get_Seq_next();
   return get_Current_ref();
   }
```

NB: returns a reference: Null ➔ False; non-Null ➔ True

Pattern:-

```
get_Seq_first();
while ( !is_Seq_empty() ) { /* process element */ get_Seq_next();   }
```

Iteration versus **recursion**: find

```
seqref be_find_val( seqref S, int v)
 { return (is_empty(S) || (v==get_value(head(S)))) ? S
     : be_find_val(tail(S), v);
}
```

NB: returns a reference: Null ➔ False; non-Null ➔ True

Pattern:-

- The empty case - non-recursive – stop condition
- The non-empty case - non-recursive – the head
- The recursive case - tail

Count: Iteration vs recursion

```
int be_size_seq() {
int size=0;
get_seq_first();
while ( !is_seq_empty() ) { size++; get_seq_next(); }
return size;
}
```

```
int be_size_seq(listref L)
{ return is_empty(L) ? 0 : 1 + be_size_seq(tail(L)); }
```

Add: Iteration vs recursion:

```
void be_add_val(valtype val) {
get_Seq_first();                          // navigate to correct position
while (!is_Seq_empty() && (val > get_Element_value()) )   get_Seq_next();
link_in(create_element(val));             // add the new element
}
```

```
seqref be_add_val(seqref S, valtype v)
{
 return is_empty(S)          ?   create_e(v)
  :   v < get_value(head(S)) ?   cons(create_e(v), S)
  :                              cons(head(S), be_add_val(tail(S),v));
}
```

cons: construct a list from the HEAD & TAIL

Find: Iteration versus recursion

```
seqref be_find_val( valtype val)  {
  get_Seq_first();
  while (!is_Seq_empty() && (val != get_Element_value()) )  get_Seq_next();
  return get_Current_ref();
  }
```

NB: returns a reference: Null ➔ False; non-Null ➔ True

```
seqref be_find_val( seqref S, int v)
 { return (is_empty(S) || (v==get_value(head(S)))) ? S
     : be_find_val(tail(S), v);
}
```

Remove: Iteration versus recursion

```
static void unlink(seqref fpCurrent) {
  if (!is_empty(fpCurrent)) {
    if (is_empty(pPrevious)) liststart = get_next(pCurrent);
    else set_next(pPrevious, get_next(pCurrent)); }
}
```

```
void be_rem_val(int fval) { unlink(be_find_val(fval)); }
```

```
seqref be_rem_val(seqref S, int v) {
return is_empty(S)                ? S
    : v == get_value(head(S))  ? tail(S)
    :                                cons(head(S), be_rem_val(tail(S),v));
}
```