

Linked Lists

Part Two

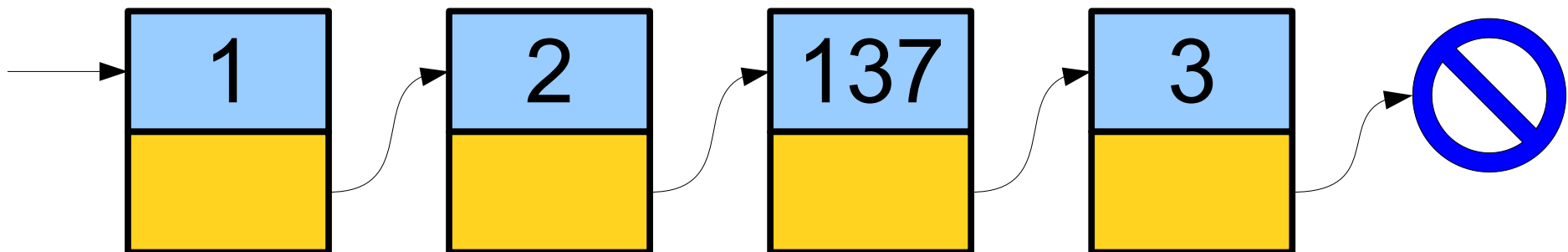
Outline for Today

- ***Pointers by Reference***
 - Changing where you're looking.
- ***Tail Pointers***
 - Speeding up list operations.
- ***Doubly-Linked Lists***
 - A preview of things to come.

Recap from Last Time

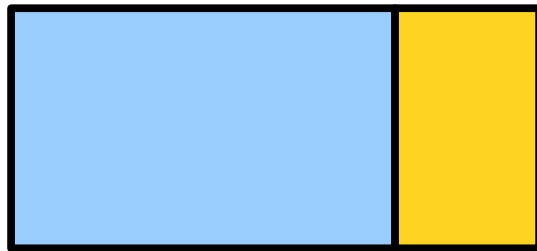
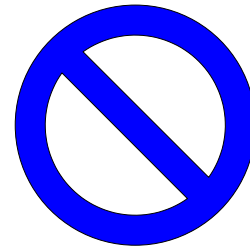
Linked Lists

- A **linked list** is a data structure for storing a sequence of elements.
- Each element is stored separately from the rest.
- The elements are then chained together into a sequence.
- The end of the list is marked with some special indicator.

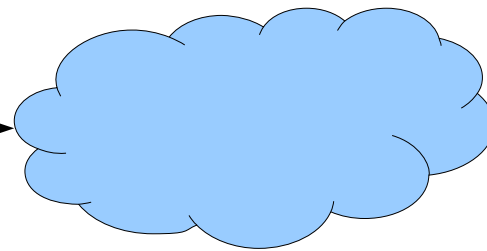


A Linked List is Either...

...an empty list,
represented by
nullptr, or...



a single linked list
cell that points...

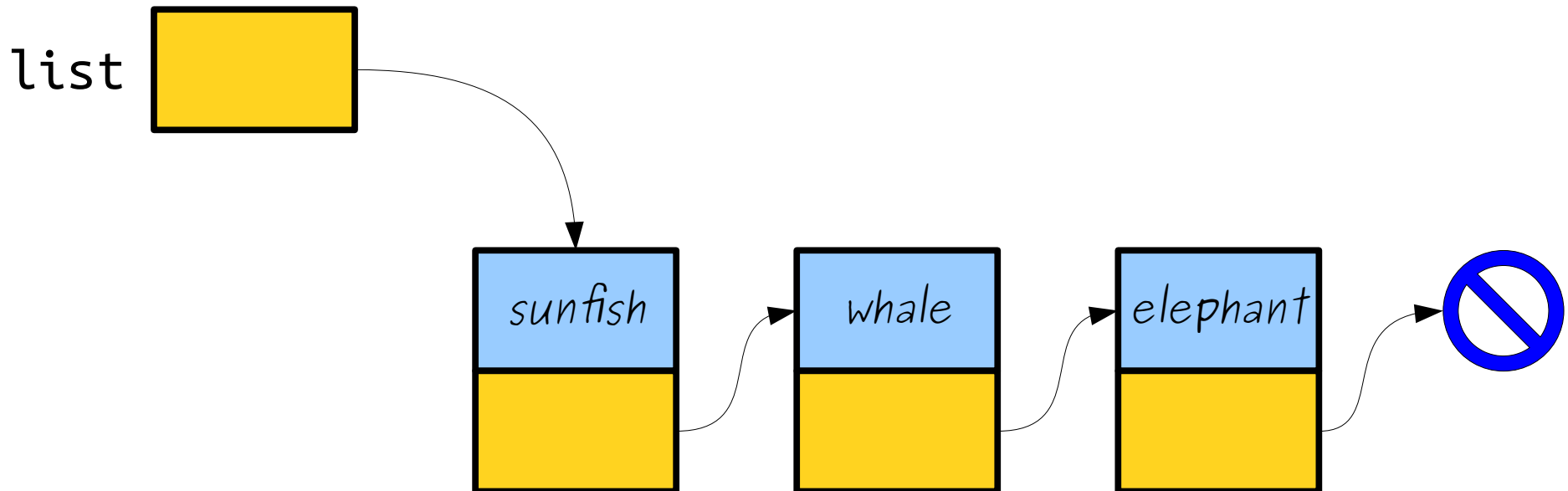


... at another linked
list.

Pointers and References

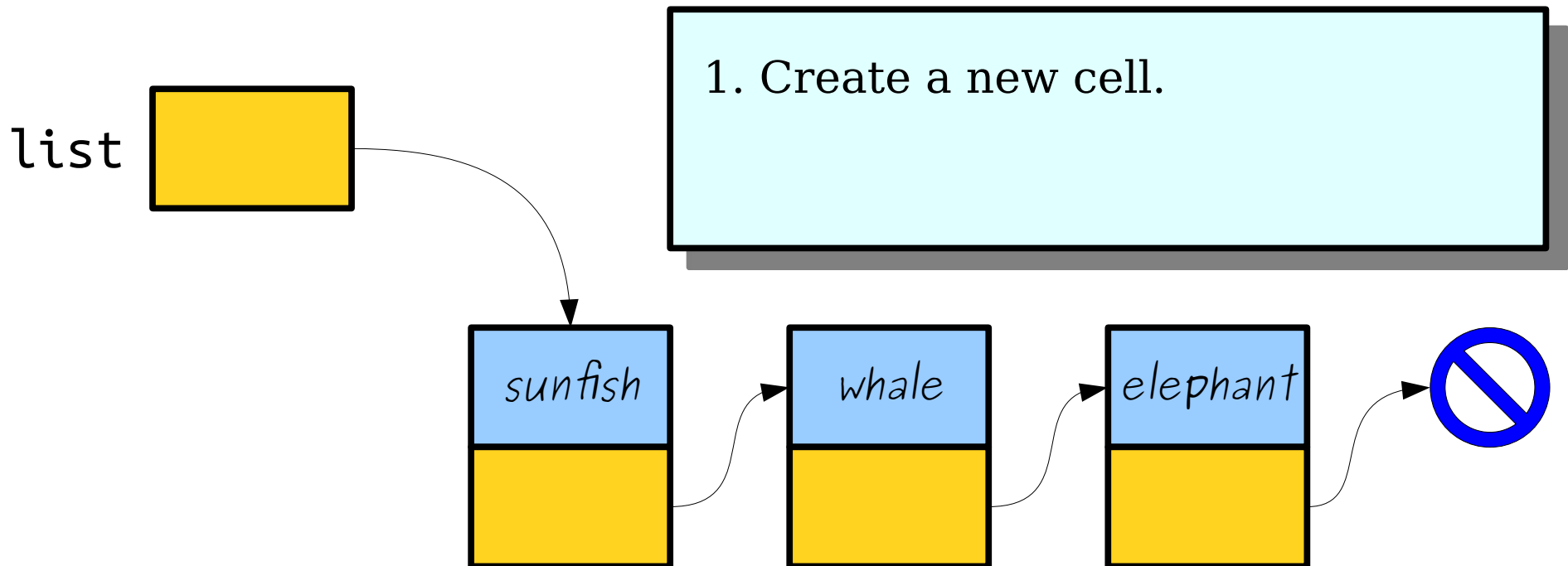
Prepending an Element

- Suppose that we want to write a function that will add an element to the front of a linked list.
- What might this function look like?



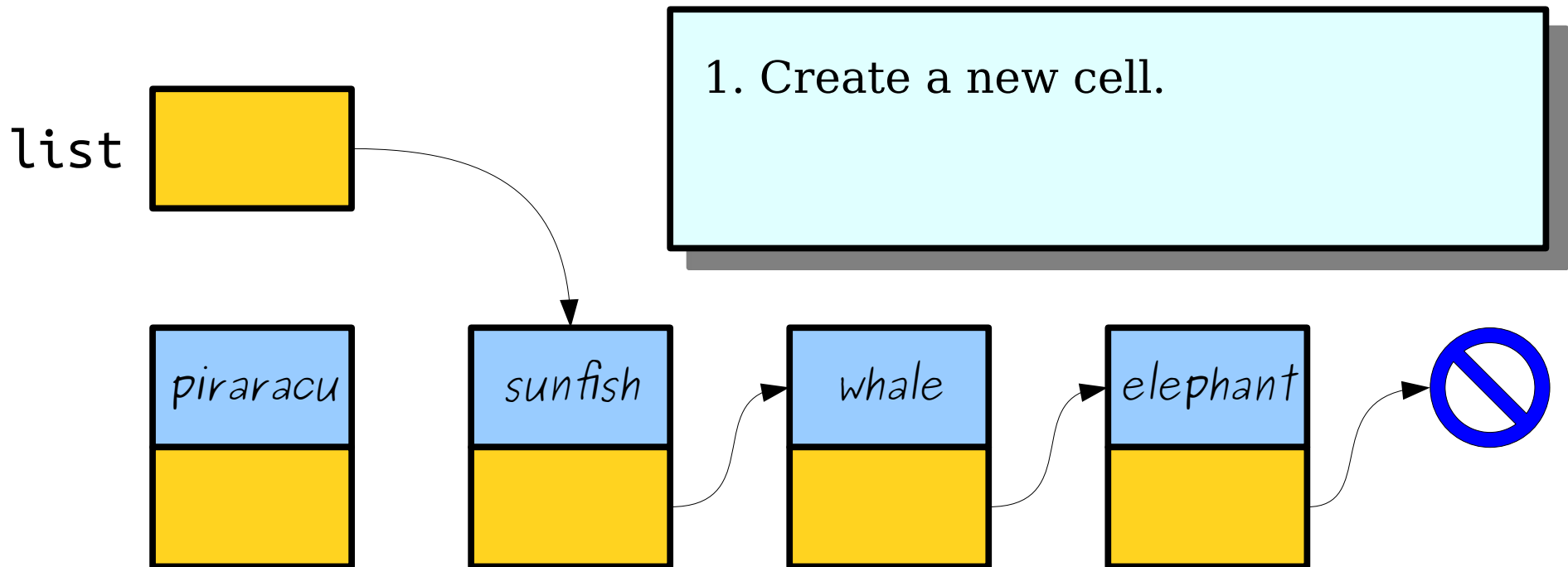
Prepending an Element

- Suppose that we want to write a function that will add an element to the front of a linked list.
- What might this function look like?



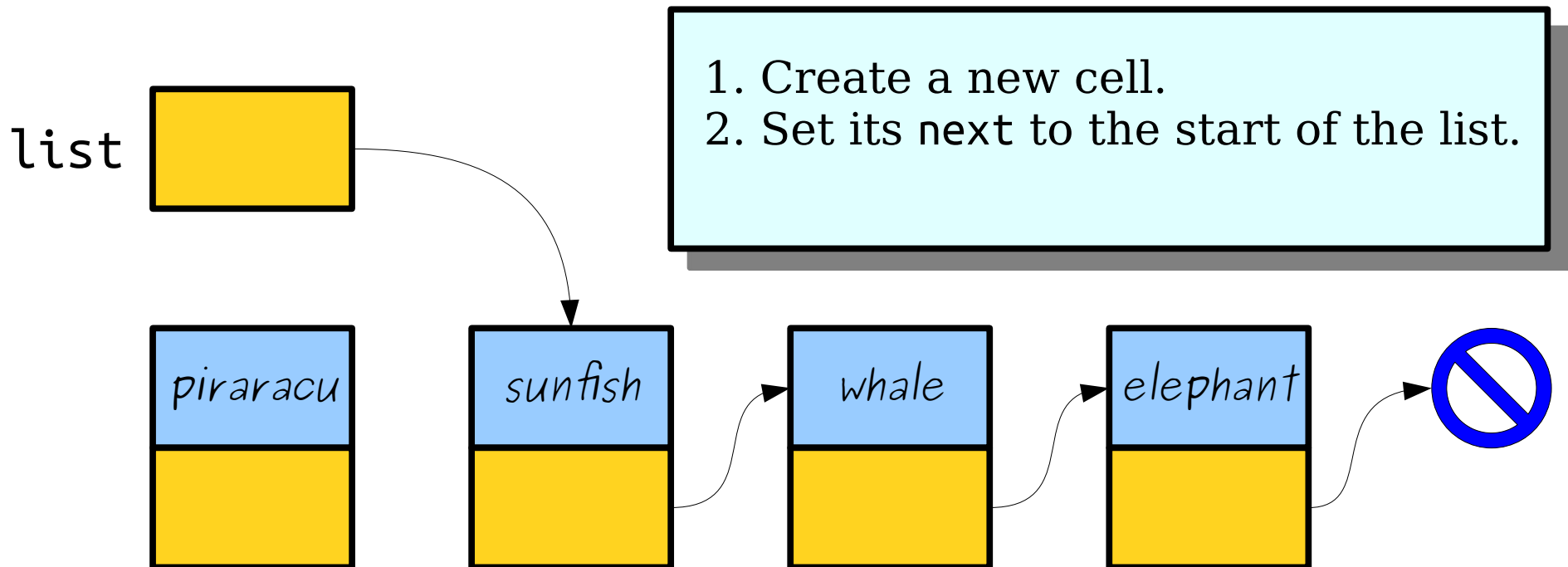
Prepending an Element

- Suppose that we want to write a function that will add an element to the front of a linked list.
- What might this function look like?



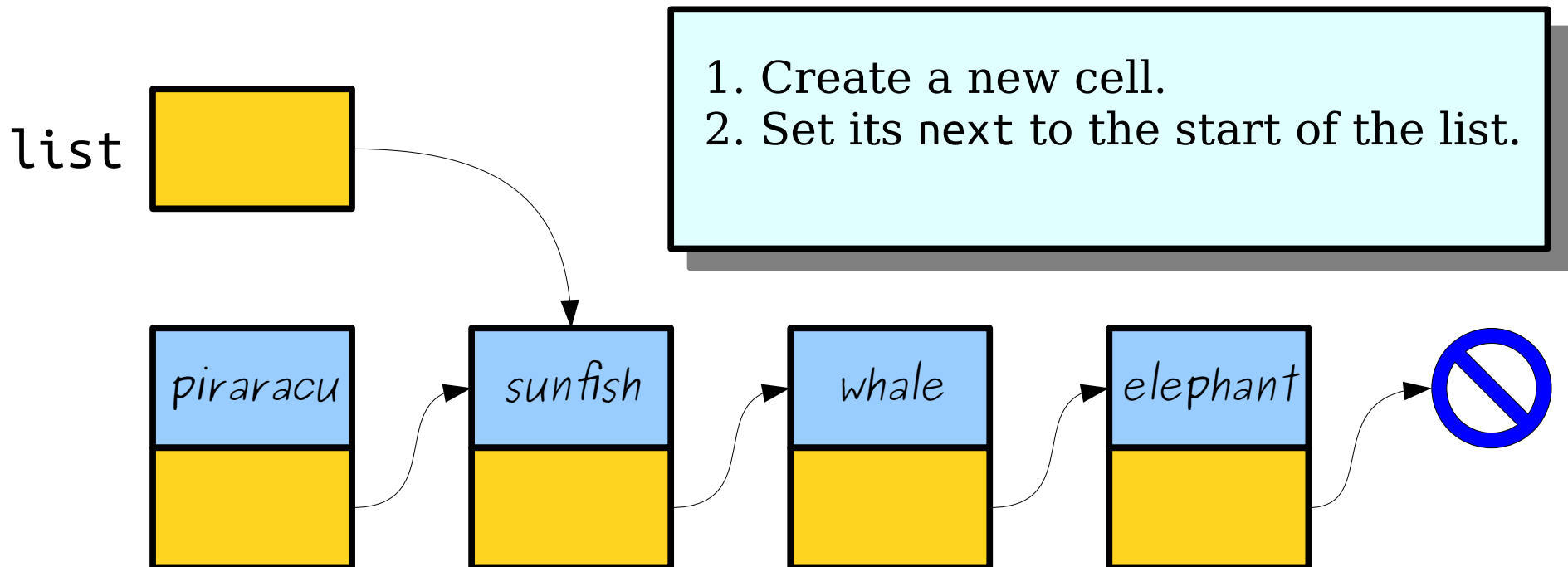
Prepending an Element

- Suppose that we want to write a function that will add an element to the front of a linked list.
- What might this function look like?



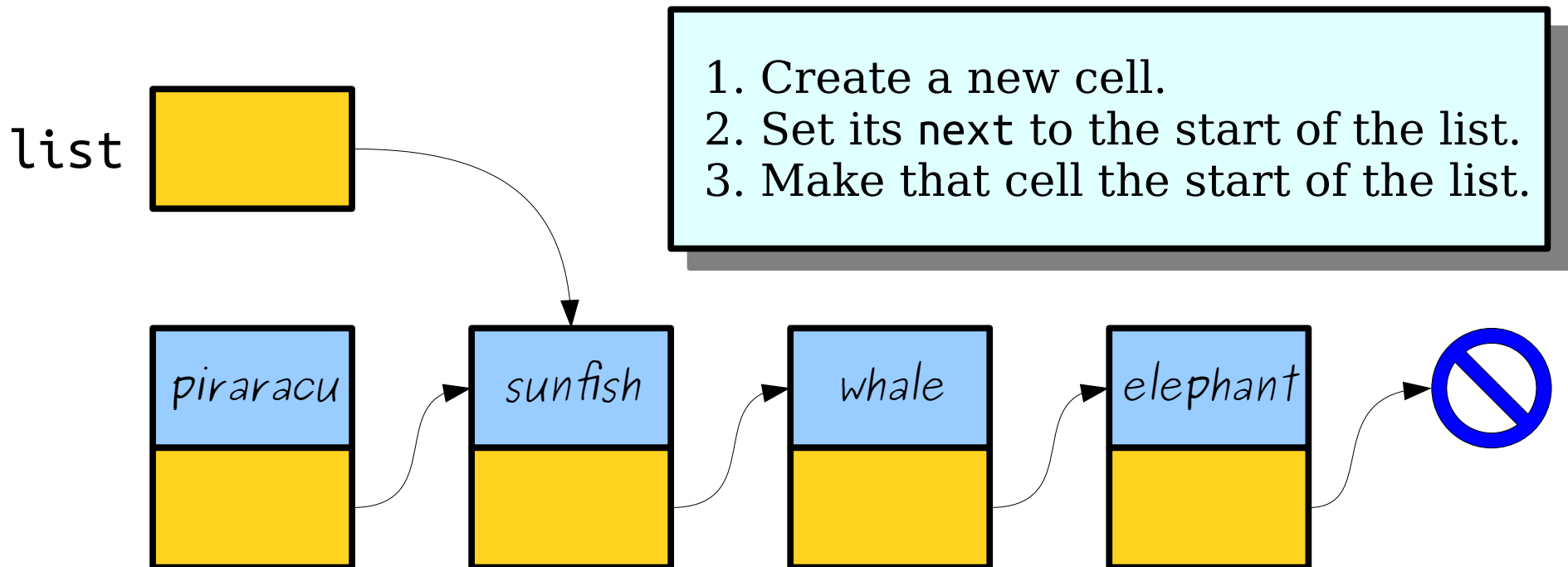
Prepending an Element

- Suppose that we want to write a function that will add an element to the front of a linked list.
- What might this function look like?



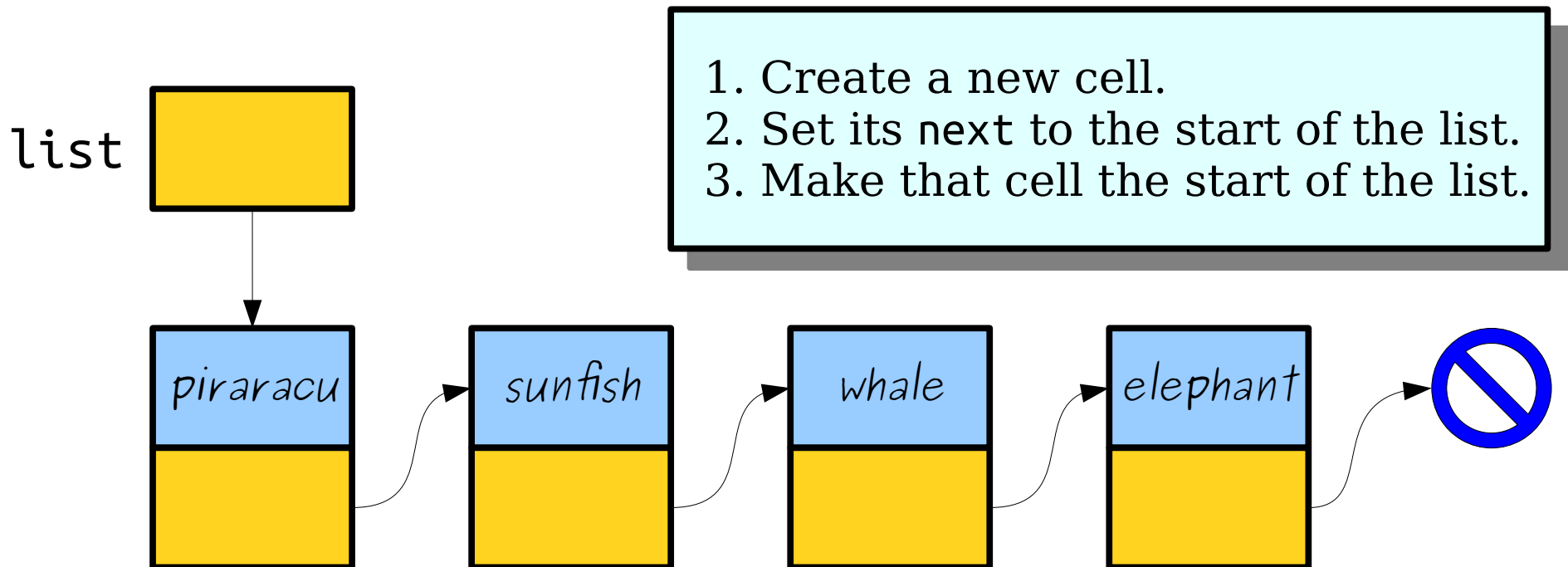
Prepending an Element

- Suppose that we want to write a function that will add an element to the front of a linked list.
- What might this function look like?



Prepending an Element

- Suppose that we want to write a function that will add an element to the front of a linked list.
- What might this function look like?



What went wrong?

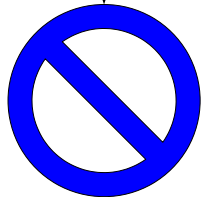
```
int main() {  
    Cell* list = nullptr;  
    prependTo(list, "Sartre");  
    prependTo(list, "Camus");  
    prependTo(list, "Nietzsche");  
  
    return 0;  
}
```

```
int main() {  
    Cell* list = nullptr;  
    prependTo(list, "Sartre");  
    prependTo(list, "Camus");  
    prependTo(list, "Nietzsche");  
    return 0;  
}
```



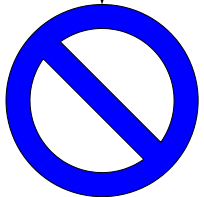
```
int main() {  
    Cell* list = nullptr;  
    prependTo(list, "Sartre");  
    prependTo(list, "Camus");  
    prependTo(list, "Nietzsche");  
    return 0;  
}
```

list



```
int main() {  
    Cell* list = nullptr;  
    prependTo(list, "Sartre");  
    prependTo(list, "Camus");  
    prependTo(list, "Nietzsche");  
    return 0;  
}
```

list



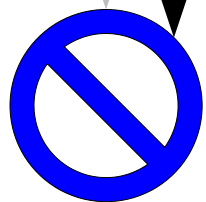
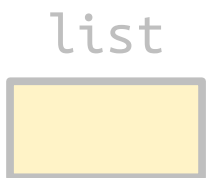
```
int main() {  
    Cell* list = nullptr;  
    prependTo(list, "Sartre");  
    prependTo(list, "Camus");  
    prependTo(list, "Nietzsche");  
    return 0;  
}
```

list



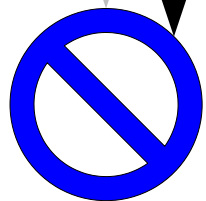
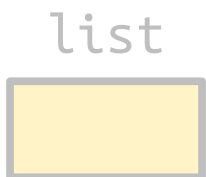
```
int main() {  
    Cell* list = nullptr;  
    prep  
    prep  
    prep  
    retu  
}
```

```
void prependTo(Cell* list, const string& val) {  
    Cell* cell = new Cell;  
    cell->value = val;  
  
    cell->next = list;  
    list = cell;  
}
```



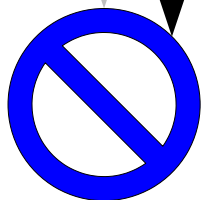
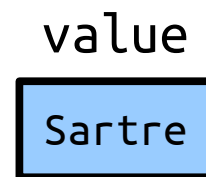
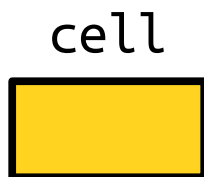
```
int main() {  
    Cell* list = nullptr;  
    prep  
    prep  
    prep  
    retu  
}
```

```
void prependTo(Cell* list, const string& val) {  
    Cell* cell = new Cell;  
    cell->value = val;  
    cell->next = list;  
    list = cell;  
}
```



```
int main() {  
    Cell* list = nullptr;  
    prep  
    prep  
    prep  
    retu  
}
```

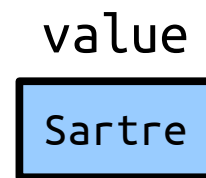
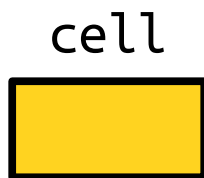
```
void prependTo(Cell* list, const string& val) {  
    Cell* cell = new Cell;  
    cell->value = val;  
    cell->next = list;  
    list = cell;  
}
```



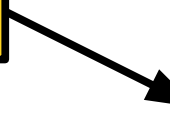
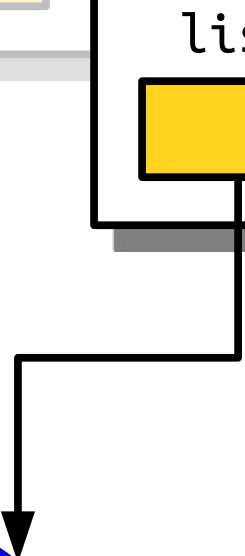
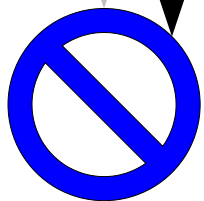
?

```
int main() {  
    Cell* list = nullptr;  
    prep  
    prep  
    prep  
    retu  
}
```

```
void prependTo(Cell* list, const string& val) {  
    Cell* cell = new Cell;  
    cell->value = val;  
    cell->next = list;  
    list = cell;  
}
```

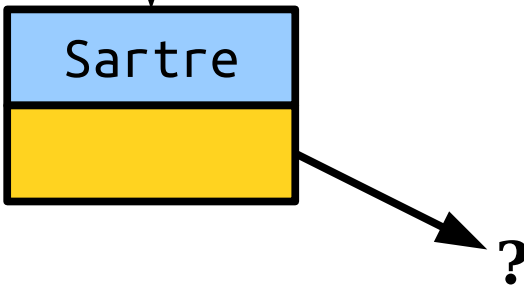
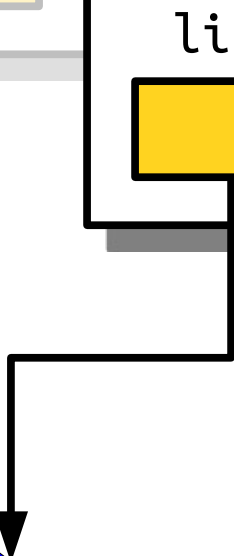
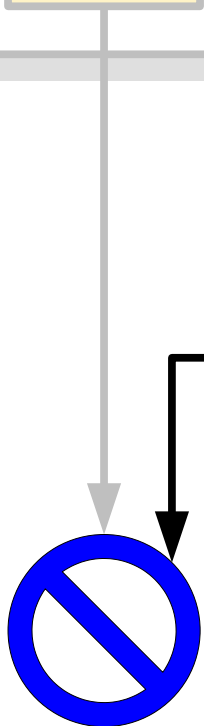
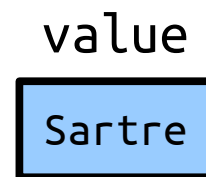
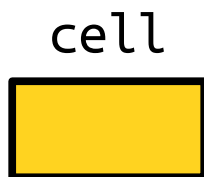
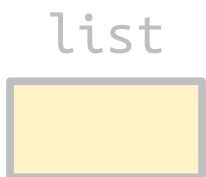


?



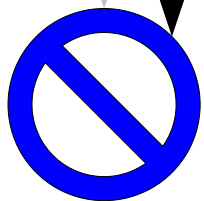
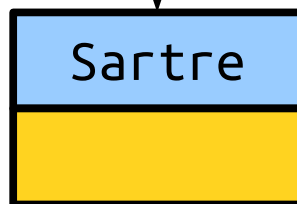
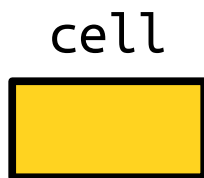
```
int main() {  
    Cell* list = nullptr;  
    prep  
    prep  
    prep  
    retu  
}
```

```
void prependTo(Cell* list, const string& val) {  
    Cell* cell = new Cell;  
    cell->value = val;  
    cell->next = list;  
    list = cell;  
}
```



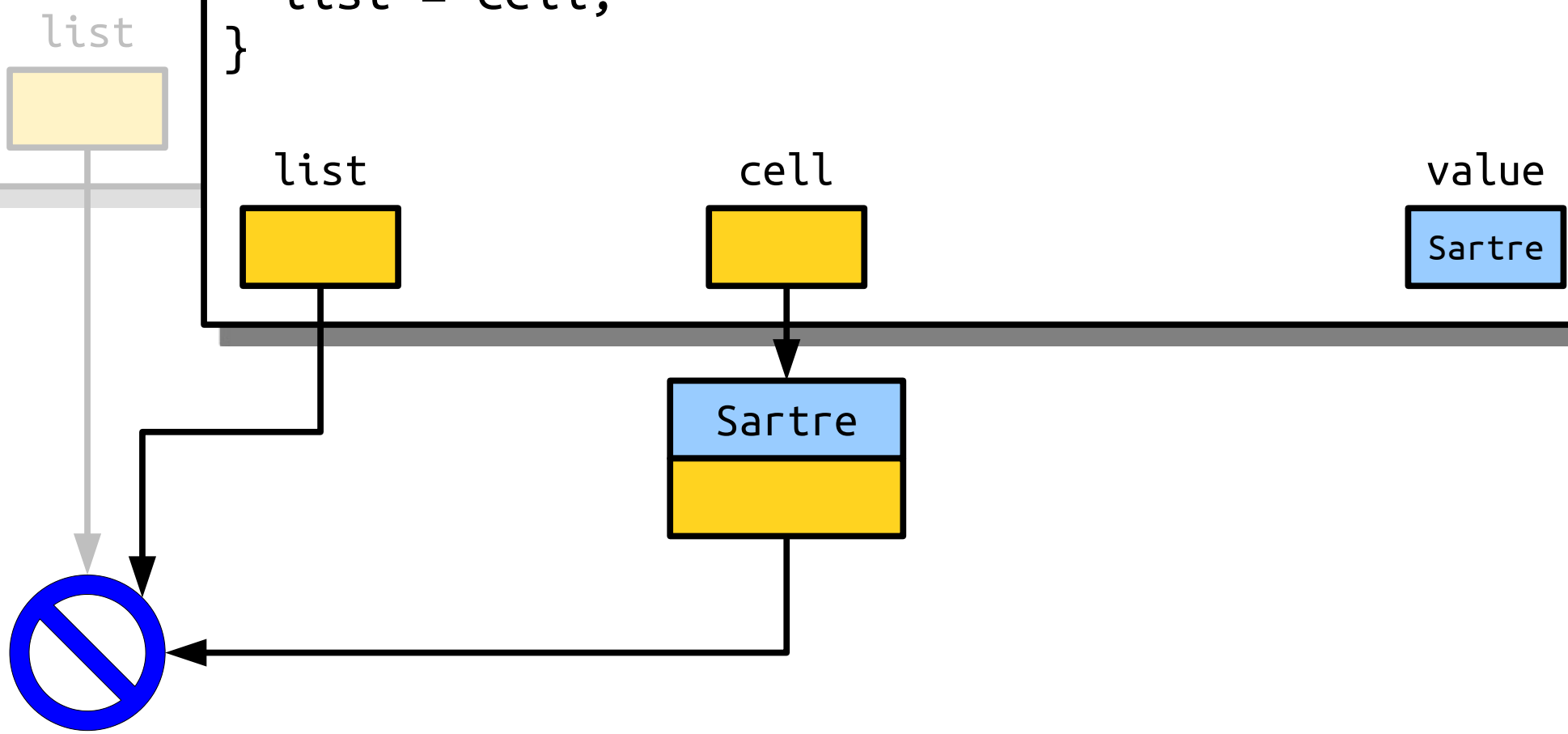

```
int main() {  
    Cell* list = nullptr;  
    prep  
    prep  
    prep  
    retu  
}
```

```
void prependTo(Cell* list, const string& val) {  
    Cell* cell = new Cell;  
    cell->value = val;  
    cell->next = list;  
    list = cell;  
}
```



```
int main() {  
    Cell* list = nullptr;  
    prep  
    prep  
    prep  
    retu  
}
```

```
void prependTo(Cell* list, const string& val) {  
    Cell* cell = new Cell;  
    cell->value = val;  
    cell->next = list;  
    list = cell;  
}
```



```
int main() {  
    Cell* list = nullptr;
```

```
    prep
```

```
    prep
```

```
    prep
```

```
    retu
```

```
}
```

```
void prependTo(Cell* list, const string& val) {  
    Cell* cell = new Cell;  
    cell->value = val;  
    cell->next = list;  
    list = cell;  
}
```

list



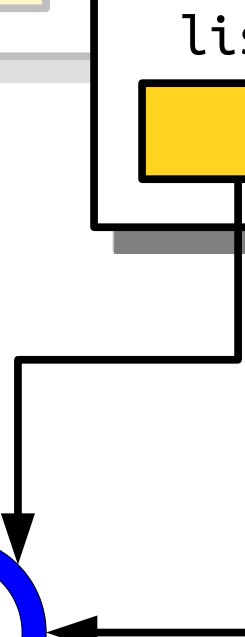
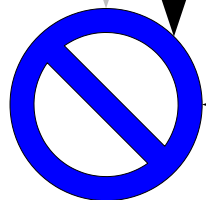
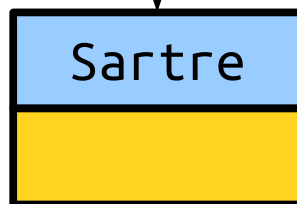
list



cell

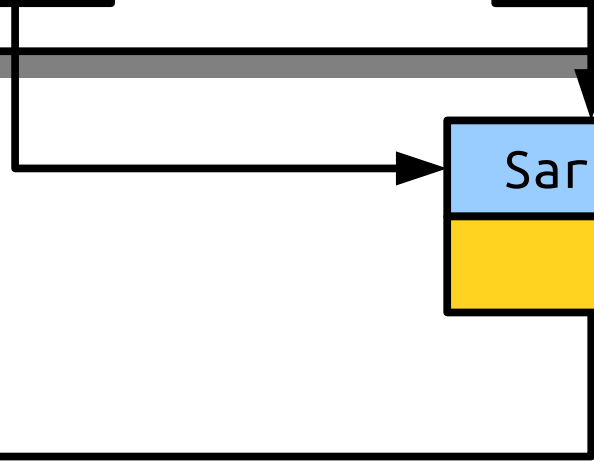
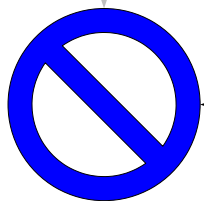
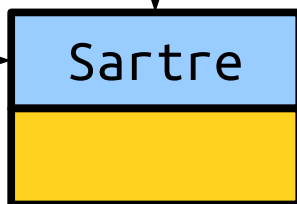
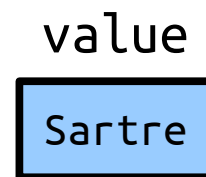
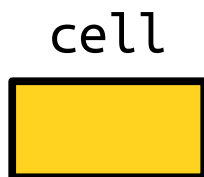


value



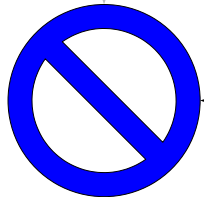
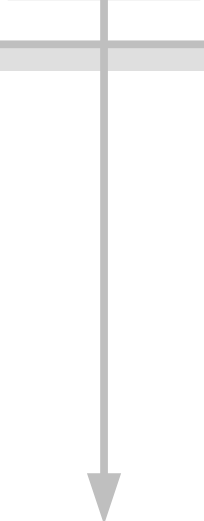
```
int main() {  
    Cell* list = nullptr;  
    prep  
    prep  
    prep  
    retu  
}
```

```
void prependTo(Cell* list, const string& val) {  
    Cell* cell = new Cell;  
    cell->value = val;  
    cell->next = list;  
    list = cell;  
}
```



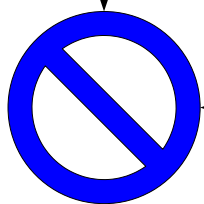
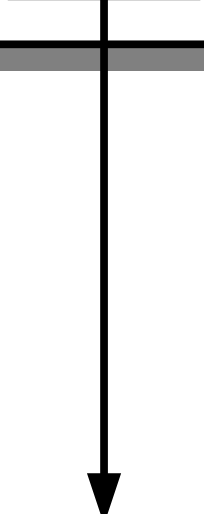
```
int main() {  
    Cell* list = nullptr;  
    prependTo(list, "Sartre");  
    prependTo(list, "Camus");  
    prependTo(list, "Nietzsche");  
    return 0;  
}
```

list



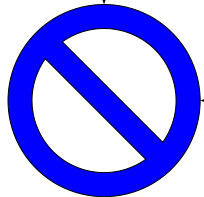
```
int main() {  
    Cell* list = nullptr;  
    prependTo(list, "Sartre");  
    prependTo(list, "Camus");  
    prependTo(list, "Nietzsche");  
    return 0;  
}
```

list



```
int main() {  
    Cell* list = nullptr;  
    prependTo(list, "Sartre");  
    prependTo(list, "Camus");  
    prependTo(list, "Nietzsche");  
    return 0;  
}
```

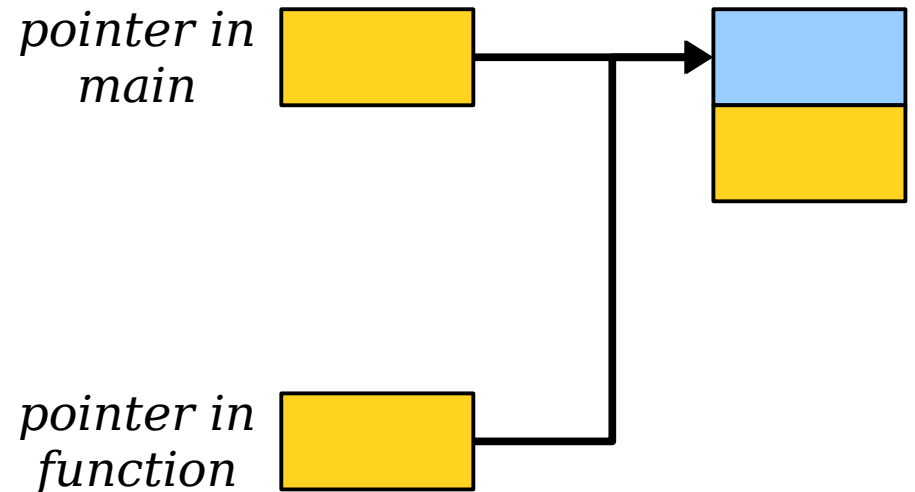
list



Hell is other pointers

Pointers By Value

- Unless specified otherwise, function arguments in C++ are passed by value.
- This includes pointers!
- A function that takes a pointer as an argument gets a copy of the pointer.
- We can change where the *copy* points, but not where the original pointer points.



Pointers by Reference

- To resolve this problem, we can pass the linked list pointer by reference.
- Our new function:

```
void prependTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = list;  
    list = cell;  
}
```

Pointers by Reference

- To resolve this problem, we can pass the linked list pointer by reference.
- Our new function:

```
void prependTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = list;  
    list = cell;  
}
```

Pointers by Reference

- To resolve this problem, we can pass the linked list pointer by reference.
- Our new function:

```
void prependTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = list;  
    list = cell;  
}
```

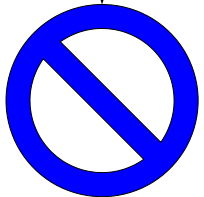
This is a *reference to a pointer to a Cell*. If we change where list points in this function, the changes will stick!

```
int main() {  
    Cell* list = nullptr;  
    prependTo(list, "Descartes");  
    prependTo(list, "Kant");  
    prependTo(list, "Bentham");  
  
    return 0;  
}
```

```
int main() {  
    Cell* list = nullptr;  
    prependTo(list, "Descartes");  
    prependTo(list, "Kant");  
    prependTo(list, "Bentham");  
    return 0;  
}
```

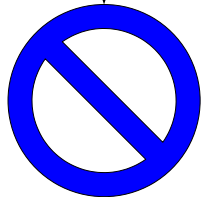
```
int main() {  
    Cell* list = nullptr;  
    prependTo(list, "Descartes");  
    prependTo(list, "Kant");  
    prependTo(list, "Bentham");  
    return 0;  
}
```

list



```
int main() {  
    Cell* list = nullptr;  
    prependTo(list, "Descartes");  
    prependTo(list, "Kant");  
    prependTo(list, "Bentham");  
    return 0;  
}
```

list



```
int main() {  
    Cell* list = nullptr;  
    prep  
    prep  
    prep  
    retu  
}
```

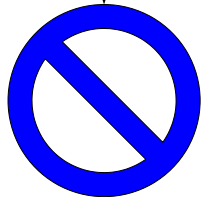
```
void prependTo(Cell*& list, const string& val) {  
    Cell* cell = new Cell;  
    cell->value = val;  
  
    cell->next = list;  
    list = cell;  
}
```

list



value

Descartes




```
int main() {  
    Cell* list = nullptr;  
    prep  
    prep  
    prep  
    retu  
}
```

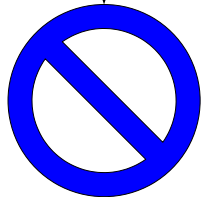
```
void prependTo(Cell*& list, const string& val) {  
    Cell* cell = new Cell;  
    cell->value = val;  
    cell->next = list;  
    list = cell;  
}
```

list



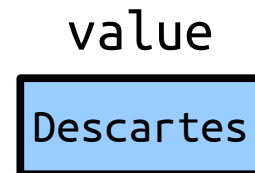
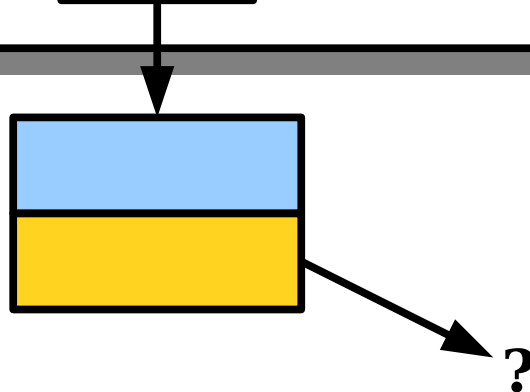
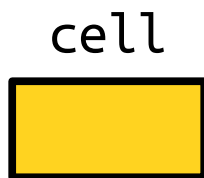
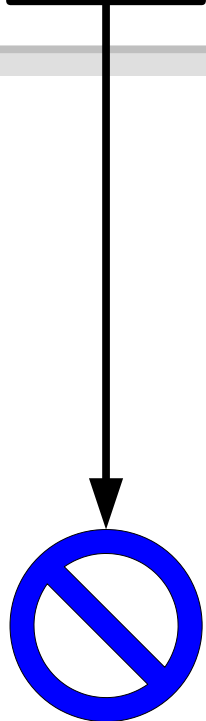
value

Descartes



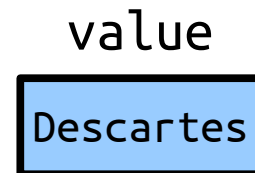
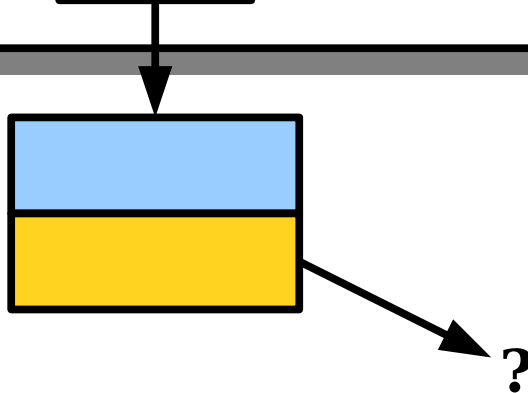
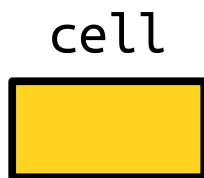
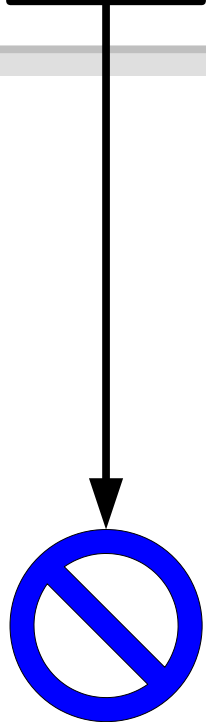
```
int main() {  
    Cell* list = nullptr;  
    prep  
    prep  
    prep  
    retu  
}
```

```
void prependTo(Cell*& list, const string& val) {  
    Cell* cell = new Cell;  
    cell->value = val;  
    cell->next = list;  
    list = cell;  
}
```



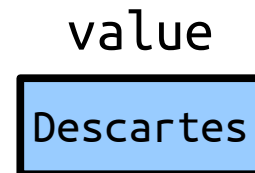
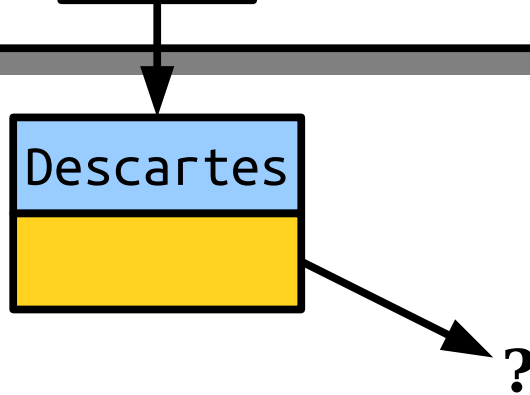
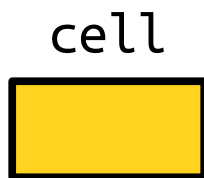
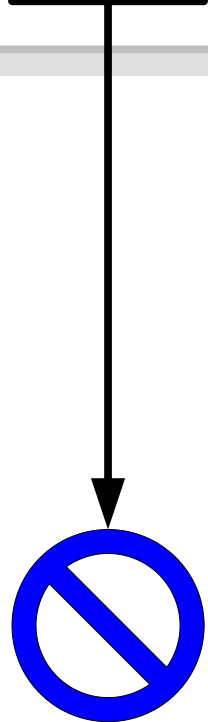
```
int main() {  
    Cell* list = nullptr;  
    prep  
    prep  
    prep  
    retu  
}
```

```
void prependTo(Cell*& list, const string& val) {  
    Cell* cell = new Cell;  
    cell->value = val;  
    cell->next = list;  
    list = cell;  
}
```



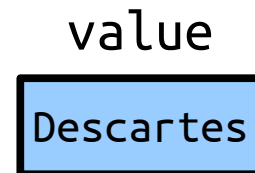
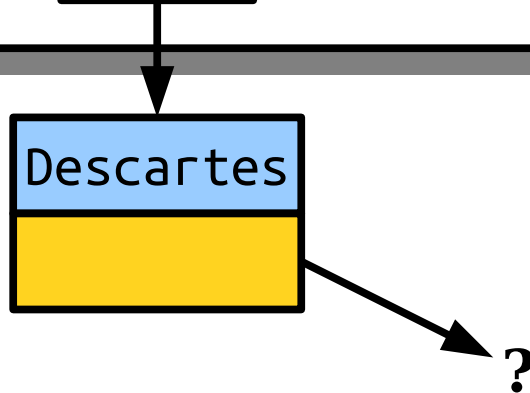
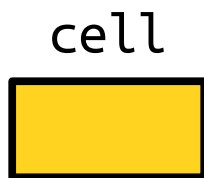
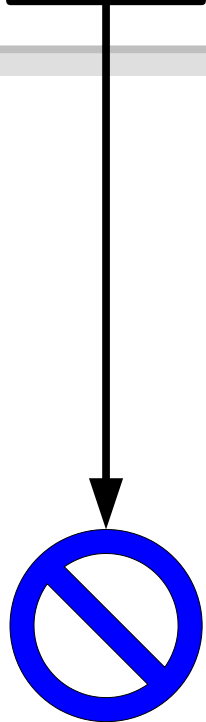
```
int main() {  
    Cell* list = nullptr;  
    prep  
    prep  
    prep  
    retu  
}
```

```
void prependTo(Cell*& list, const string& val) {  
    Cell* cell = new Cell;  
    cell->value = val;  
    cell->next = list;  
    list = cell;  
}
```



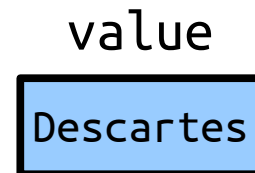
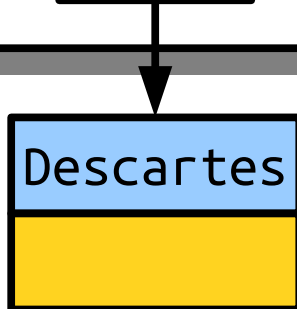
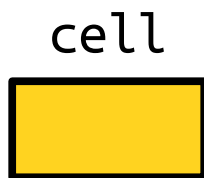
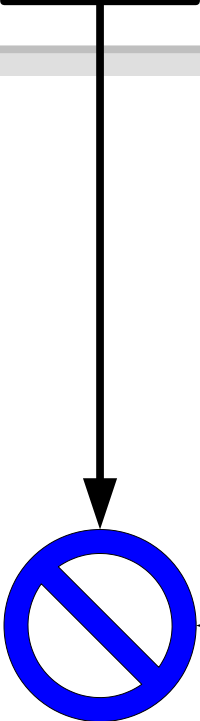
```
int main() {  
    Cell* list = nullptr;  
    prep  
    prep  
    prep  
    retu  
}
```

```
void prependTo(Cell*& list, const string& val) {  
    Cell* cell = new Cell;  
    cell->value = val;  
    cell->next = list;  
    list = cell;  
}
```



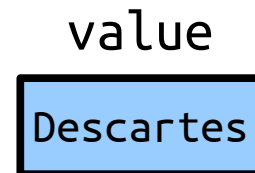
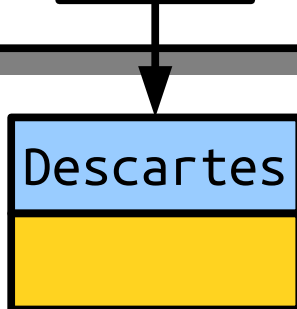
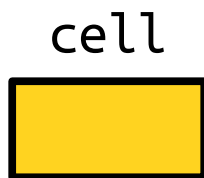
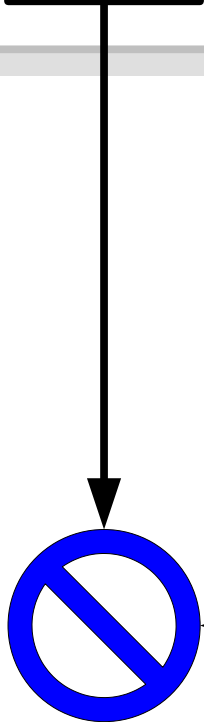
```
int main() {  
    Cell* list = nullptr;  
    prep  
    prep  
    prep  
    retu  
}
```

```
void prependTo(Cell*& list, const string& val) {  
    Cell* cell = new Cell;  
    cell->value = val;  
    cell->next = list;  
    list = cell;  
}
```



```
int main() {  
    Cell* list = nullptr;  
    prep  
    prep  
    prep  
    retu  
}
```

```
void prependTo(Cell*& list, const string& val) {  
    Cell* cell = new Cell;  
    cell->value = val;  
    cell->next = list;  
    list = cell;  
}
```



```
int main() {  
    Cell* list = nullptr;
```

```
    prep
```

```
    prep
```

```
    prep
```

```
    retu
```

```
}
```

```
void prependTo(Cell*& list, const string& val) {  
    Cell* cell = new Cell;  
    cell->value = val;  
    cell->next = list;  
    list = cell;  
}
```

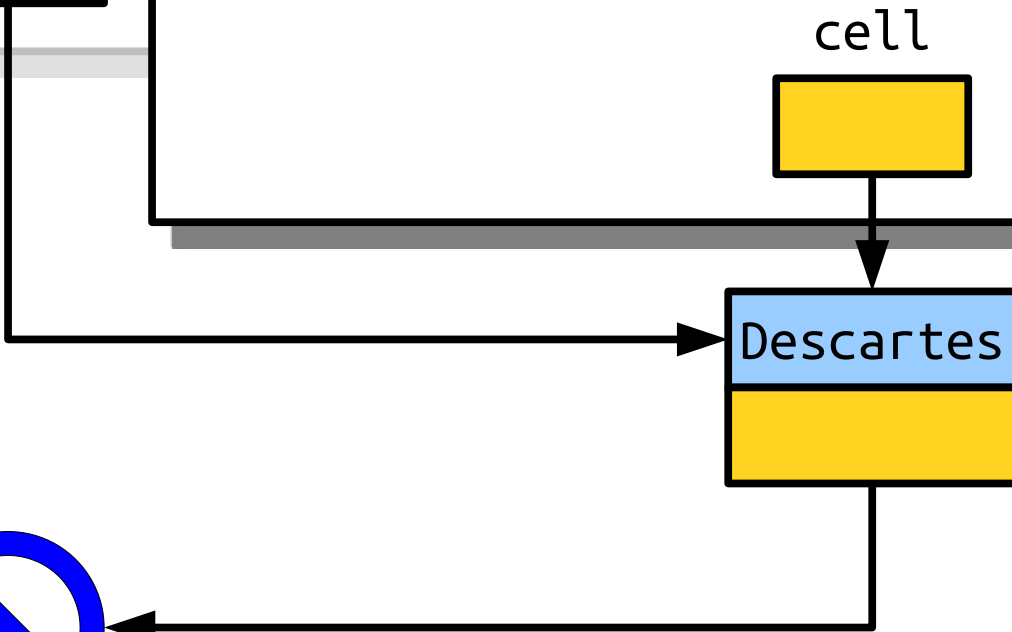
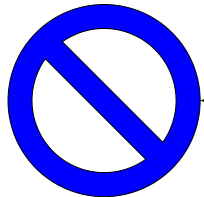
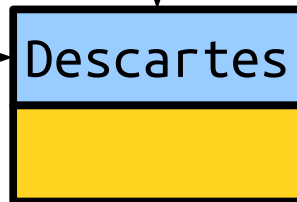
list



cell

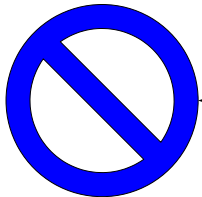


value



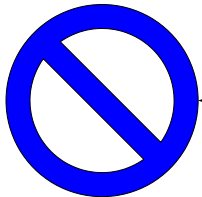
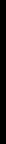
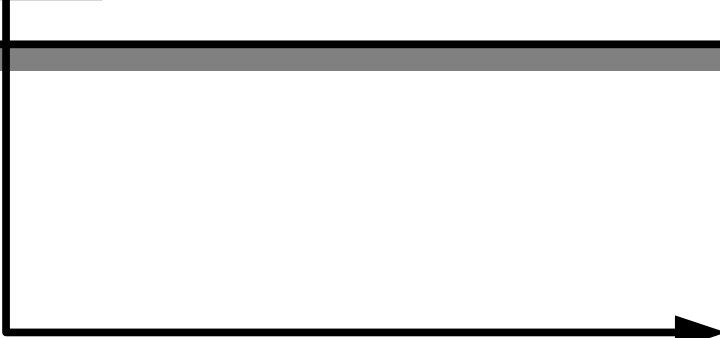

```
int main() {  
    Cell* list = nullptr;  
    prependTo(list, "Descartes");  
    prependTo(list, "Kant");  
    prependTo(list, "Bentham");  
    return 0;  
}
```

list



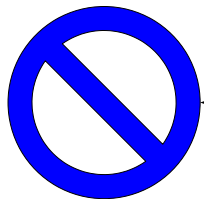
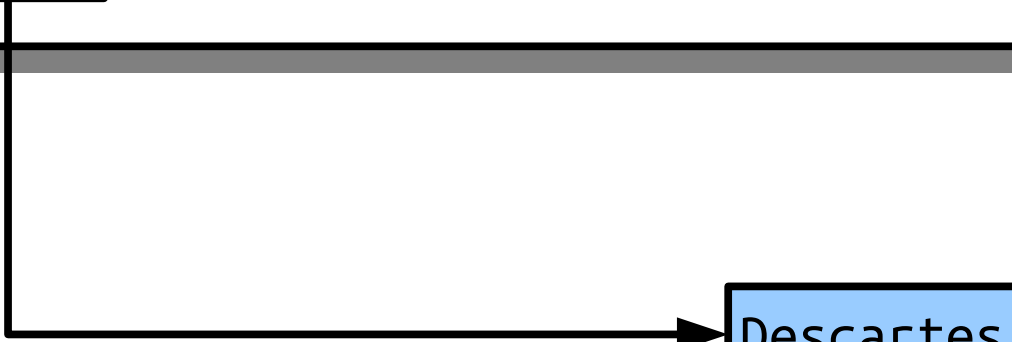
```
int main() {  
    Cell* list = nullptr;  
    prependTo(list, "Descartes");  
    prependTo(list, "Kant");  
    prependTo(list, "Bentham");  
    return 0;  
}
```

list



```
int main() {  
    Cell* list = nullptr;  
    prependTo(list, "Descartes");  
    prependTo(list, "Kant");  
    prependTo(list, "Bentham");  
    return 0;  
}
```

list



*I link,
therefore I am.*

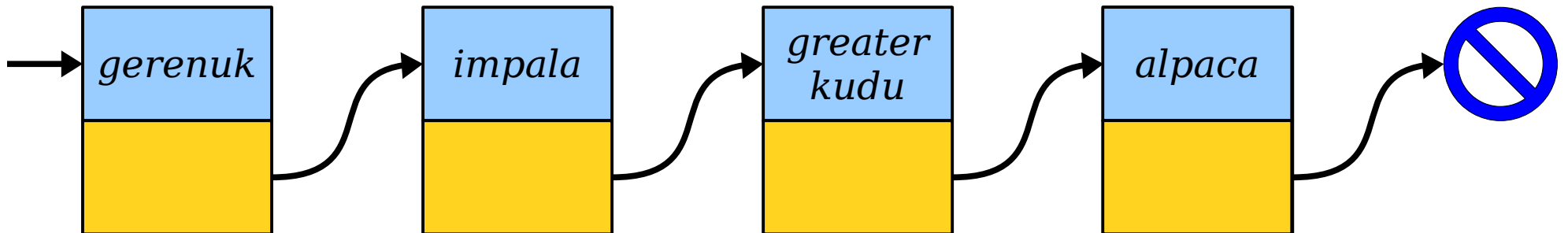
Pointers by Reference

- If you pass a pointer into a function *by value*, you can change the contents at the object you point at, but not *which* object you point at.
- If you pass a pointer into a function *by reference*, you can *also* change *which* object is pointed at.

Appending to a List

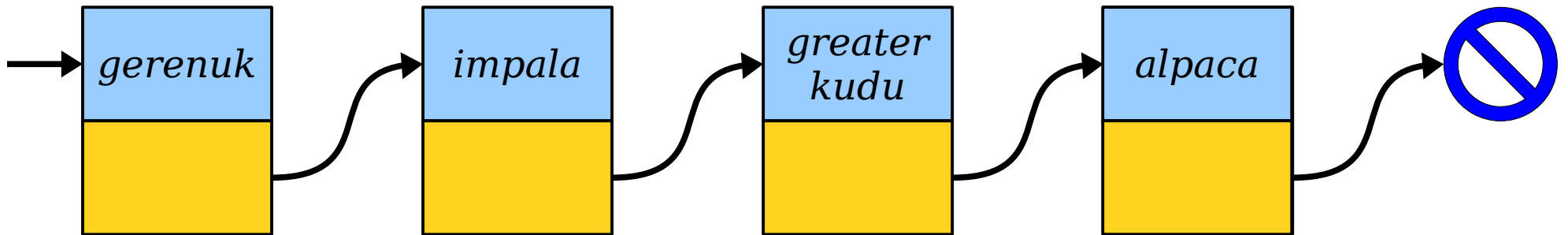
Appending to a List

- Think about which link needs to get changed to append something to this list:



Appending to a List

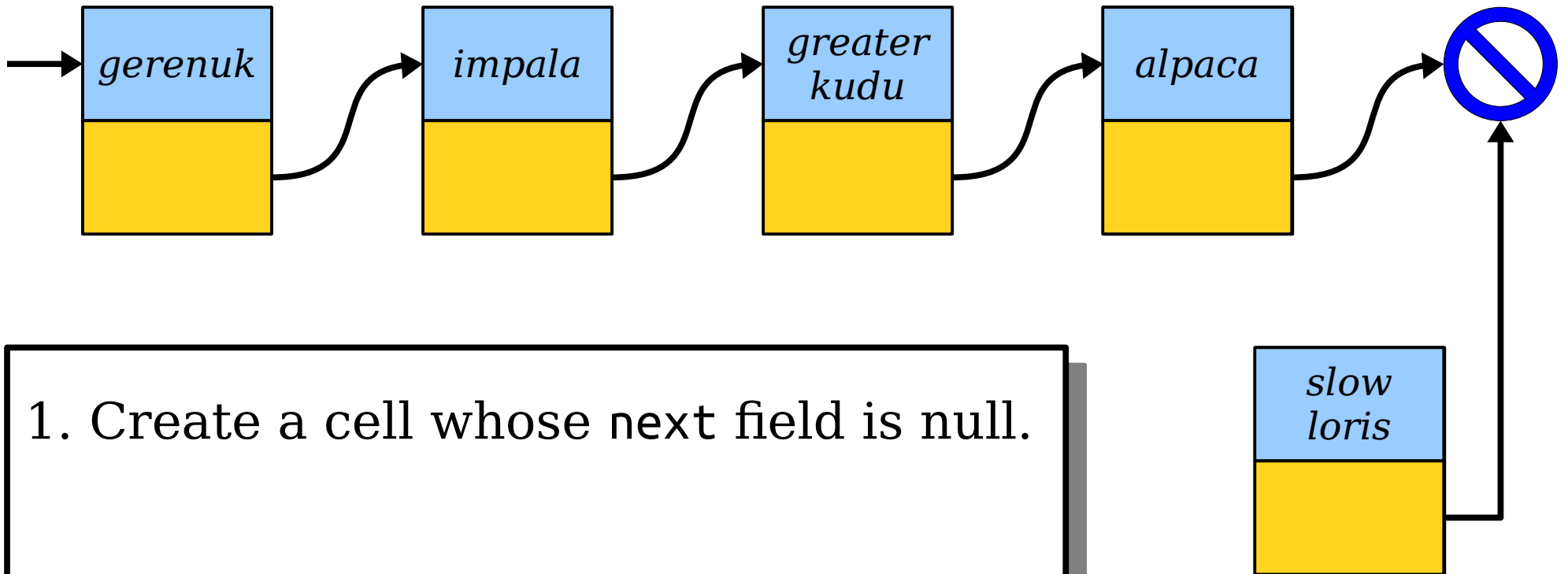
- Think about which link needs to get changed to append something to this list:



1. Create a cell whose next field is null.

Appending to a List

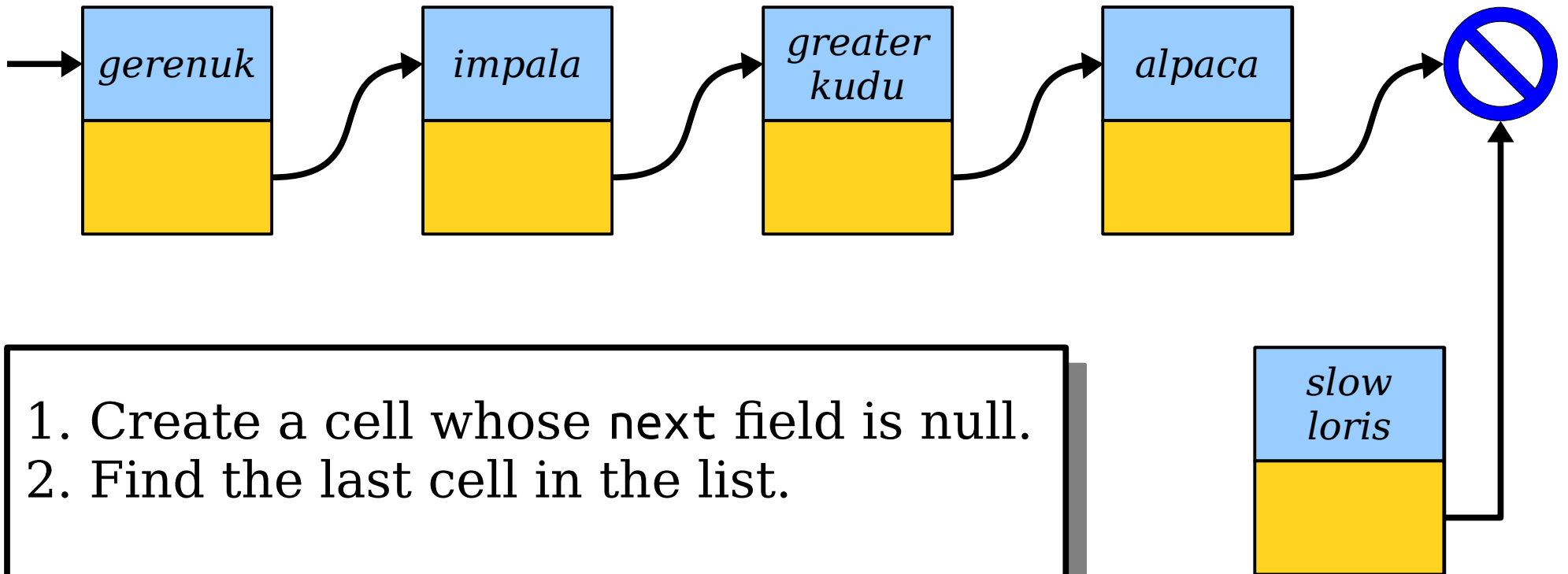
- Think about which link needs to get changed to append something to this list:



1. Create a cell whose next field is null.

Appending to a List

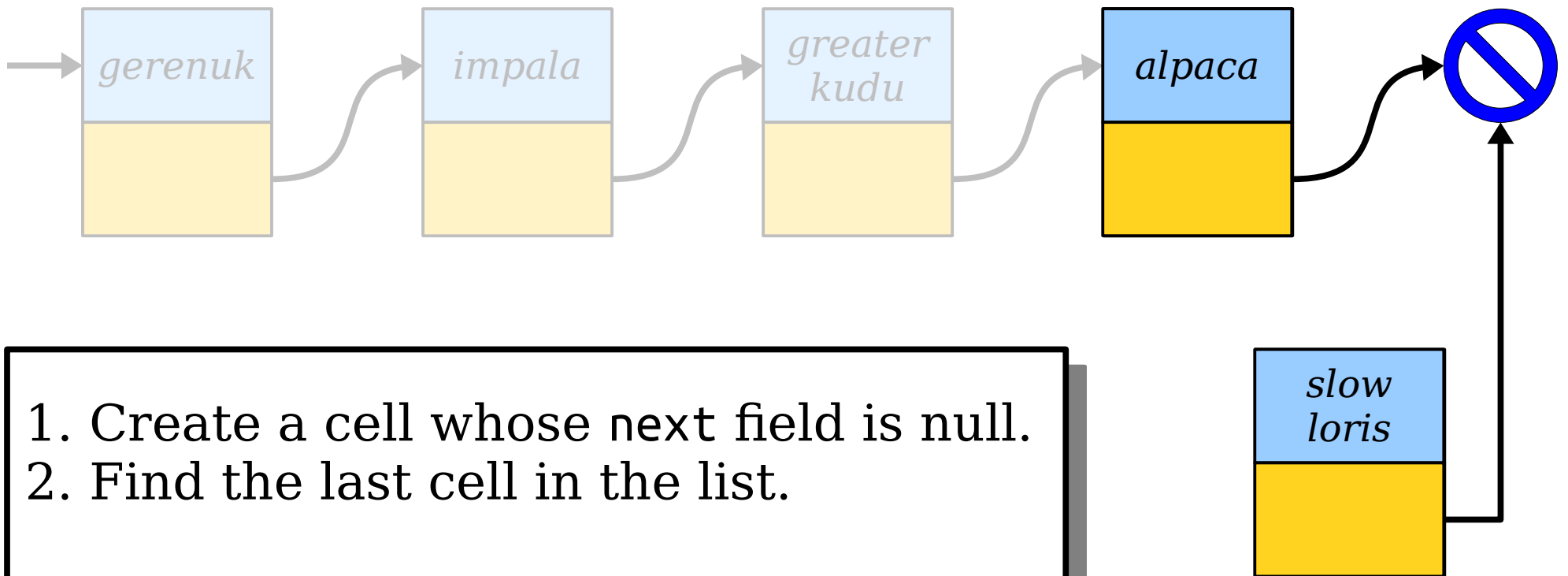
- Think about which link needs to get changed to append something to this list:



1. Create a cell whose next field is null.
2. Find the last cell in the list.

Appending to a List

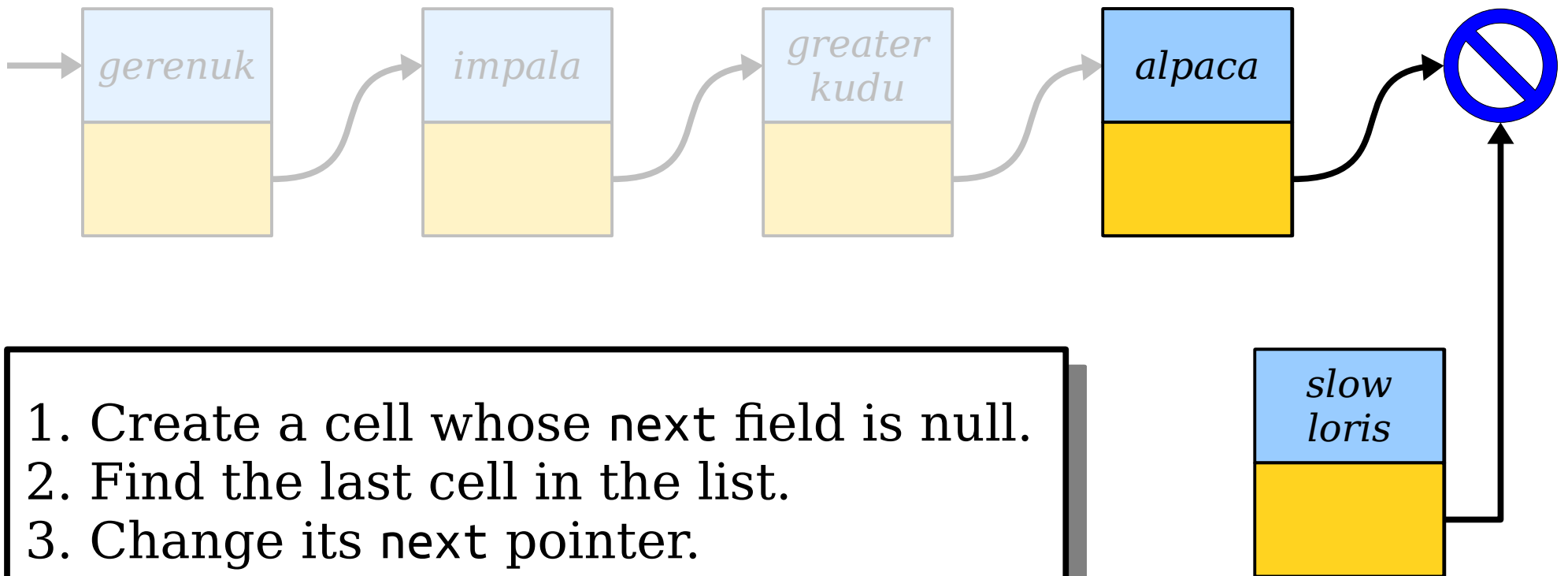
- Think about which link needs to get changed to append something to this list:



1. Create a cell whose next field is null.
2. Find the last cell in the list.

Appending to a List

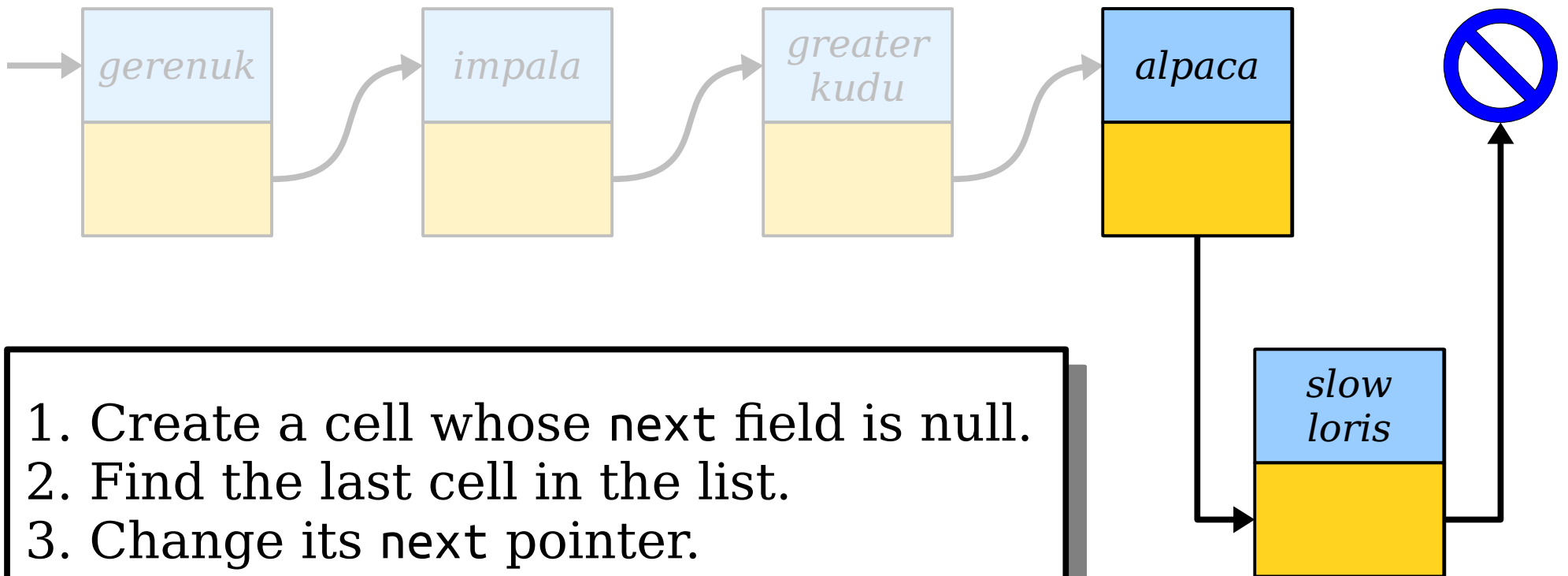
- Think about which link needs to get changed to append something to this list:



1. Create a cell whose next field is null.
2. Find the last cell in the list.
3. Change its next pointer.

Appending to a List

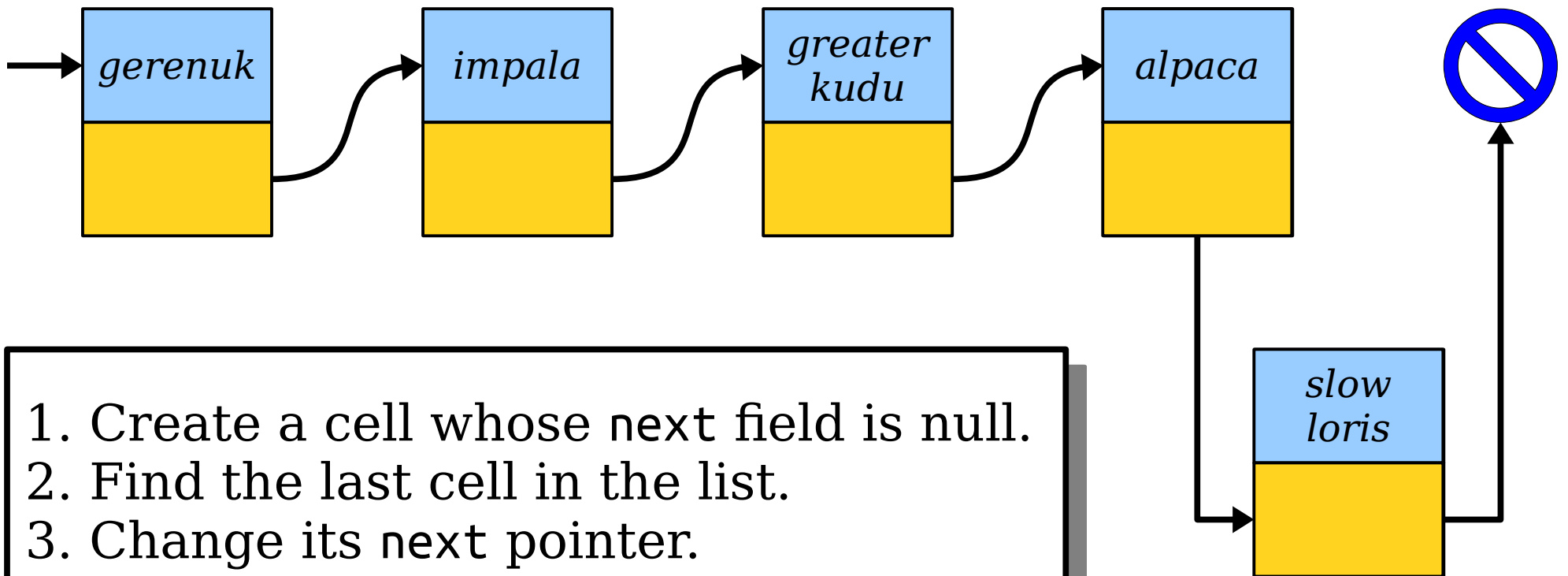
- Think about which link needs to get changed to append something to this list:



1. Create a cell whose next field is null.
2. Find the last cell in the list.
3. Change its next pointer.

Appending to a List

- Think about which link needs to get changed to append something to this list:



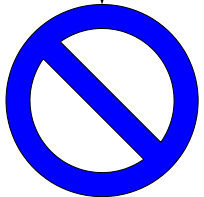
1. Create a cell whose next field is null.
2. Find the last cell in the list.
3. Change its next pointer.

```
int main() {  
    Cell* list = nullptr;  
    appendTo(list, "Elephant");  
    appendTo(list, "Sunfish");  
    appendTo(list, "Whale");  
    appendTo(list, "Piraracu");  
  
    /* ... other listy things. ... */  
}
```

```
int main() {  
    Cell* list = nullptr;  
    appendTo(list, "Elephant");  
    appendTo(list, "Sunfish");  
    appendTo(list, "Whale");  
    appendTo(list, "Piraracu");  
  
    /* ... other listy things. ... */  
}
```

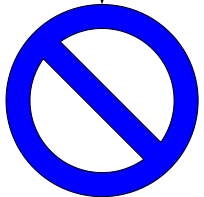
```
int main() {  
    Cell* list = nullptr;  
    appendTo(list, "Elephant");  
    appendTo(list, "Sunfish");  
    appendTo(list, "Whale");  
    appendTo(list, "Piraracu");  
  
    /* ... other listy things. ... */  
}
```

list



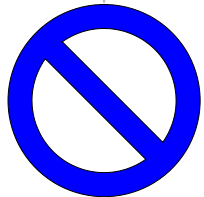

```
int main() {  
    Cell* list = nullptr;  
    appendTo(list, "Elephant");  
    appendTo(list, "Sunfish");  
    appendTo(list, "Whale");  
    appendTo(list, "Piraracu");  
  
    /* ... other listy things. ... */  
}
```

list



```
int main() {  
    Cell* list = nullptr;  
    appendTo(list, "Elephant");  
    appendTo(list, "Sunfish");  
    appendTo(list, "Whale");  
    appendTo(list, "Piraracu");  
  
    /* ... other listy things. ... */  
}
```

list

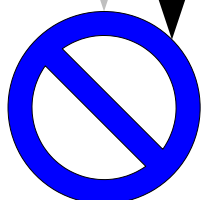
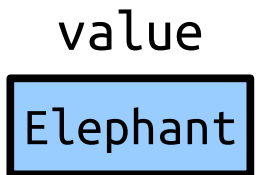
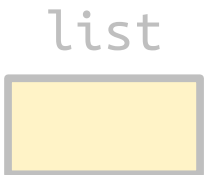


```
int main() {
    Cell* list;
    appendT
    appendT
    appendT
    appendT
    /* ... ot
}
```

```
void appendTo(Cell* list, const string& value) {
    Cell* cell = new Cell;
    cell->value = value;
    cell->next = nullptr;

    while (list->next != nullptr) {
        list = list->next;
    }

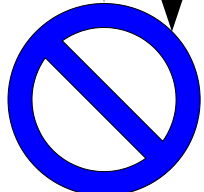
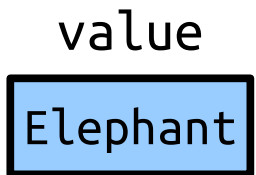
    list->next = cell;
}
```



```
int main() {  
    Cell* list;  
    appendTo(list, "Elephant");  
    appendTo(list, "Giraffe");  
    appendTo(list, "Horse");  
    appendTo(list, "Lion");  
    /* ... other code ...  
}
```

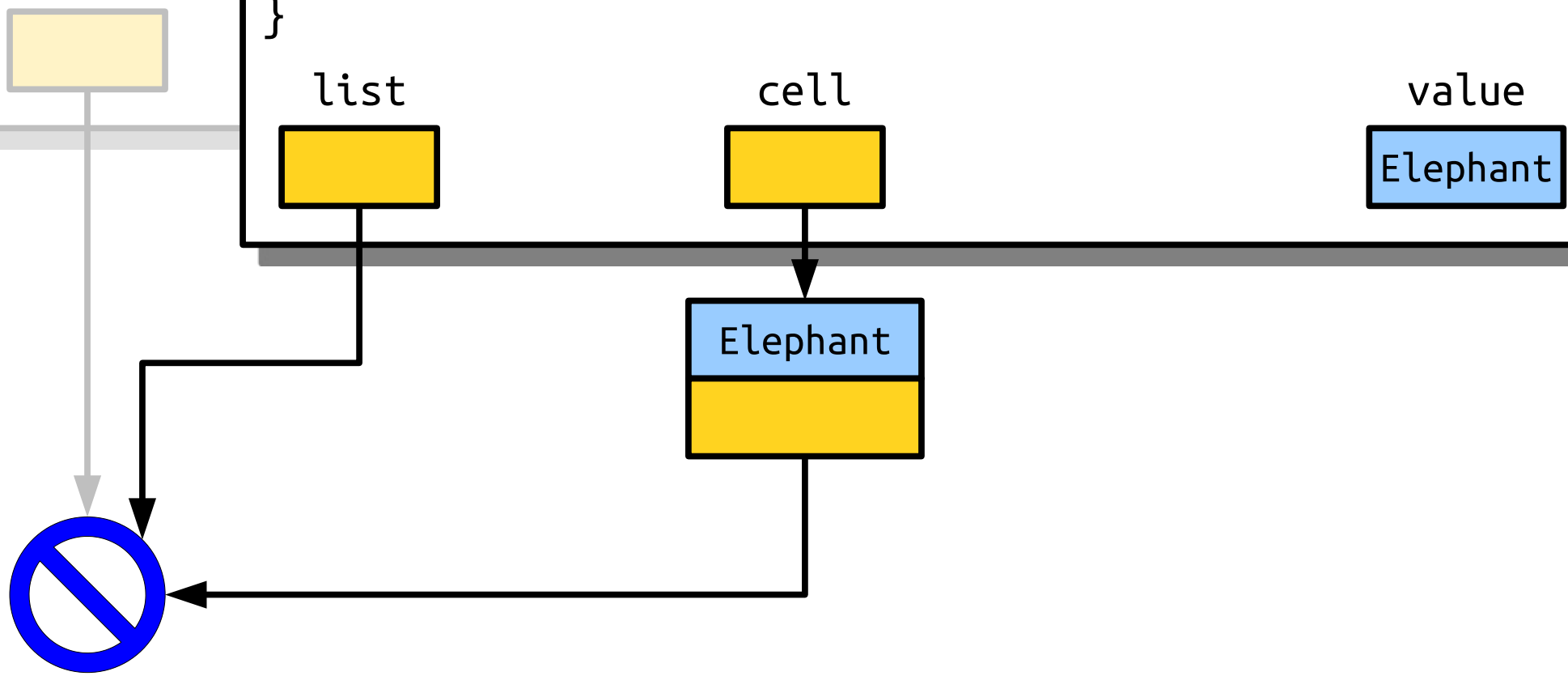


```
void appendTo(Cell* list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    while (list->next != nullptr) {  
        list = list->next;  
    }  
    list->next = cell;  
}
```



```
int main() {  
    Cell* list;  
    appendTo(list, "Elephant");  
    appendTo(list, "Giraffe");  
    appendTo(list, "Lion");  
    appendTo(list, "Zebra");  
    /* ... other code ...  
}
```

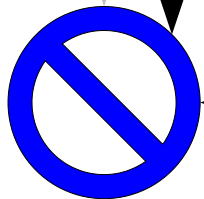
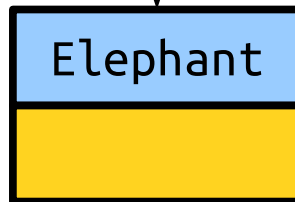
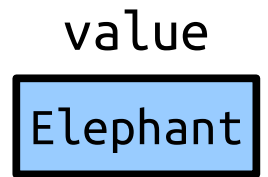
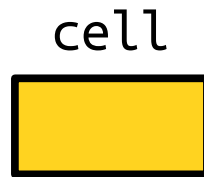
```
void appendTo(Cell* list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    while (list->next != nullptr) {  
        list = list->next;  
    }  
    list->next = cell;  
}
```



```
int main() {  
    Cell* list;  
    appendT  
    appendT  
    appendT  
    appendT  
    /* ... ot  
}
```



```
void appendTo(Cell* list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
    while (list->next != nullptr) {  
        list = list->next;  
    }  
    list->next = cell;  
}
```



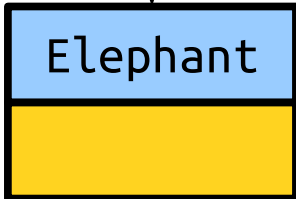
```
int main() {  
    Cell* list = nullptr;  
    appendT  
    appendT  
    appendT  
    appendT  
    /* ... ot  
}  
list  
  
void appendTo(Cell* list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
    while (list->next != nullptr) { // Uh oh!  
        list = list->next;  
    }  
    list->next = cell;  
}
```

while (list->next != nullptr) { // Uh oh!
list = list->next;

list

cell

value

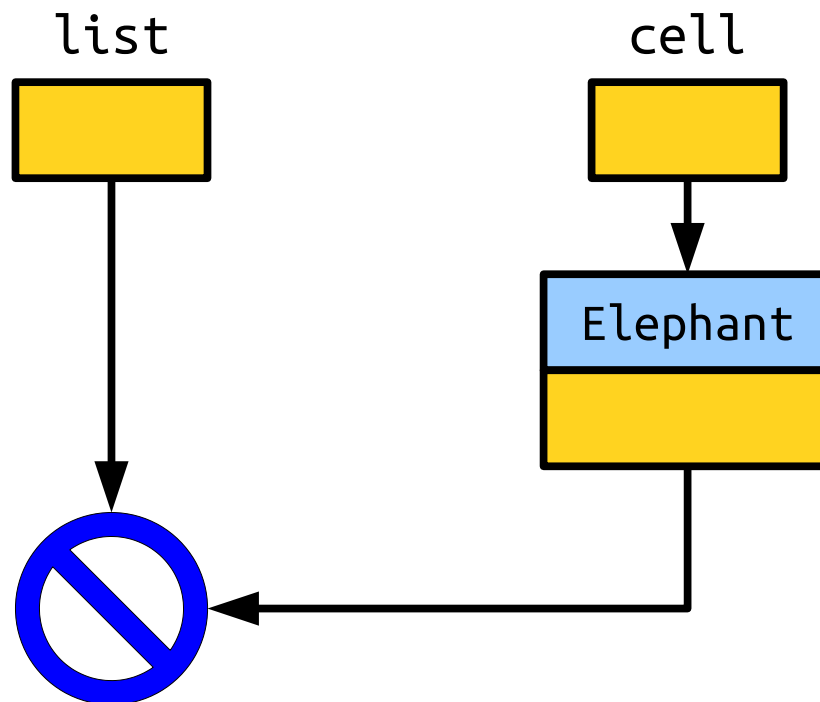


**Null Pointer
Dereference!**



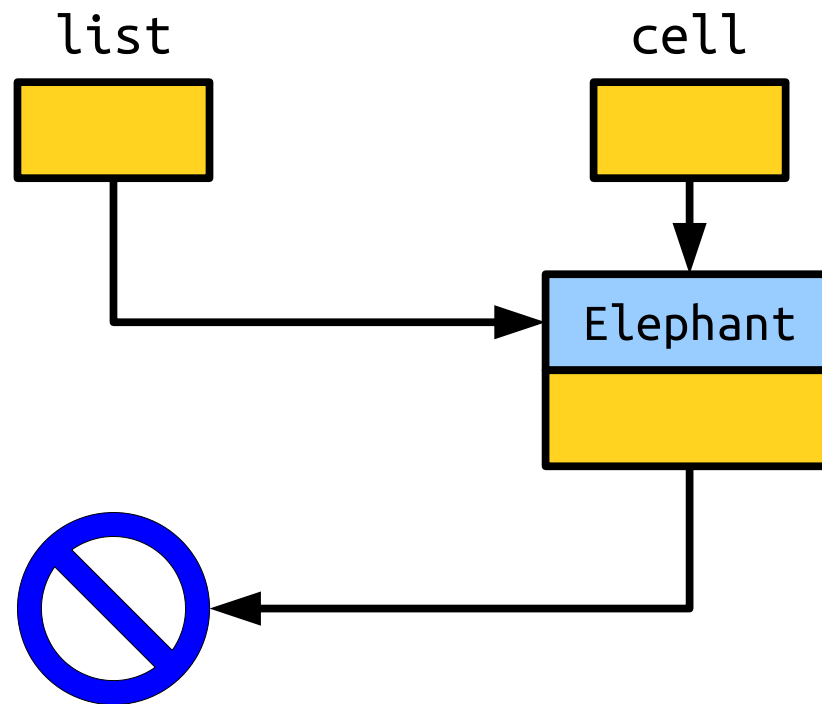
Appending to a List

- There's an edge case we missed! We need to account for the list being empty.
- If the list is empty, we should change the list pointer to point to our new cell.
- Let's change things up and see if we can fix this problem.



Appending to a List

- There's an edge case we missed! We need to account for the list being empty.
- If the list is empty, we should change the list pointer to point to our new cell.
- Let's change things up and see if we can fix this problem.



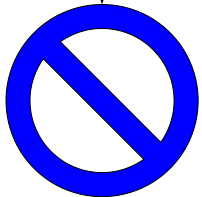
```
int main() {
    Cell* list = nullptr;
    appendTo(list, "Elephant");
    appendTo(list, "Sunfish");
    appendTo(list, "Whale");
    appendTo(list, "Piraracu");

    /* ... other listy things. ... */
}
```

```
int main() {  
    Cell* list = nullptr;  
    appendTo(list, "Elephant");  
    appendTo(list, "Sunfish");  
    appendTo(list, "Whale");  
    appendTo(list, "Piraracu");  
  
    /* ... other listy things. ... */  
}
```

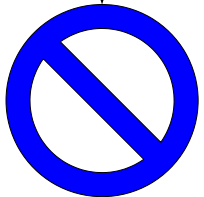
```
int main() {  
    Cell* list = nullptr;  
    appendTo(list, "Elephant");  
    appendTo(list, "Sunfish");  
    appendTo(list, "Whale");  
    appendTo(list, "Piraracu");  
  
    /* ... other listy things. ... */  
}
```

list

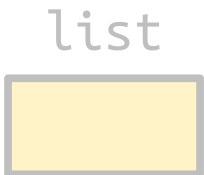


```
int main() {  
    Cell* list = nullptr;  
    appendTo(list, "Elephant");  
    appendTo(list, "Sunfish");  
    appendTo(list, "Whale");  
    appendTo(list, "Piraracu");  
  
    /* ... other listy things. ... */  
}
```

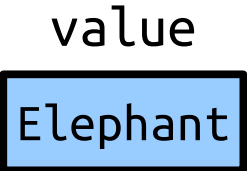
list



```
int main() {  
    Cell* list = nullptr;  
    appendT  
    appendT  
    appendT  
    appendT  
    /* ... ot  
}
```



```
void appendTo(Cell* list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        while (list->next != nullptr) {  
            list = list->next;  
        }  
        list->next = cell;  
    }  
}
```



```
int main() {  
    Cell* list;  
    appendT  
    appendT  
    appendT  
    appendT  
    /* ... ot  
}
```

```
void appendTo(Cell* list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        while (list->next != nullptr) {  
            list = list->next;  
        }  
        list->next = cell;  
    }  
}
```



list



list

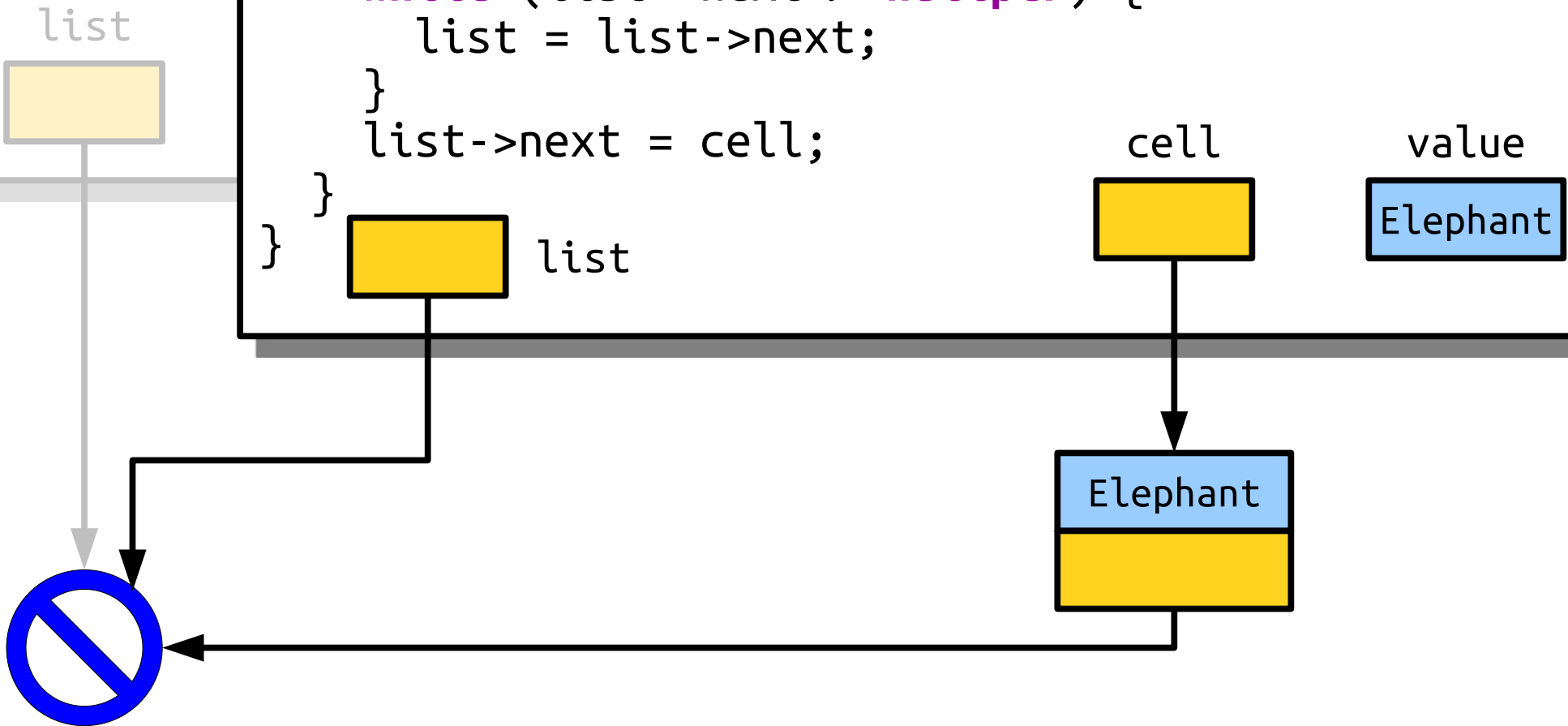
value

Elephant

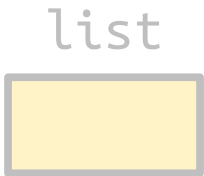


```
int main() {  
    Cell* list;  
    appendTo(list, "Elephant");  
    appendTo(list, "Giraffe");  
    appendTo(list, "Horse");  
    appendTo(list, "Lion");  
    /* ... other code ...  
}
```

```
void appendTo(Cell* list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        while (list->next != nullptr) {  
            list = list->next;  
        }  
        list->next = cell;  
    }  
}
```




```
int main() {  
    Cell* list;  
    appendT  
    appendT  
    appendT  
    appendT  
    /* ... ot  
}
```



```
void appendTo(Cell* list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        while (list->next != nullptr) {  
            list = list->next;  
        }  
        list->next = cell;  
    }  
}
```



list

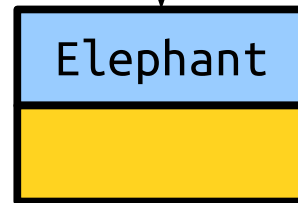


cell



value

Elephant

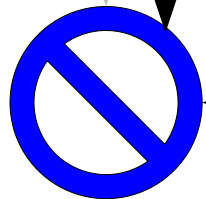
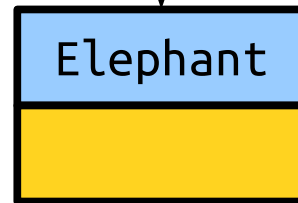
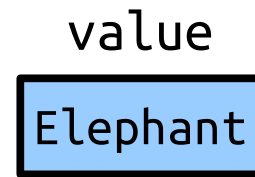
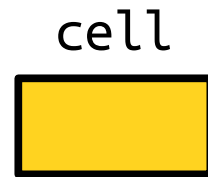


Elephant



```
int main() {  
    Cell* list;  
    appendT  
    appendT  
    appendT  
    appendT
```

```
void appendTo(Cell* list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        while (list->next != nullptr) {  
            list = list->next;  
        }  
        list->next = cell;  
    }  
}
```



```
int main() {  
    Cell* list;  
    appendT  
    appendT  
    appendT  
    appendT  
    /* ... ot  
}
```

list



```
void appendTo(Cell* list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        while (list->next != nullptr) {  
            list = list->next;  
        }  
        list->next = cell;  
    }  
}
```

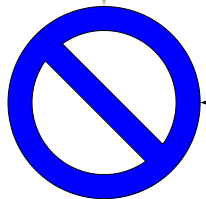


list

cell

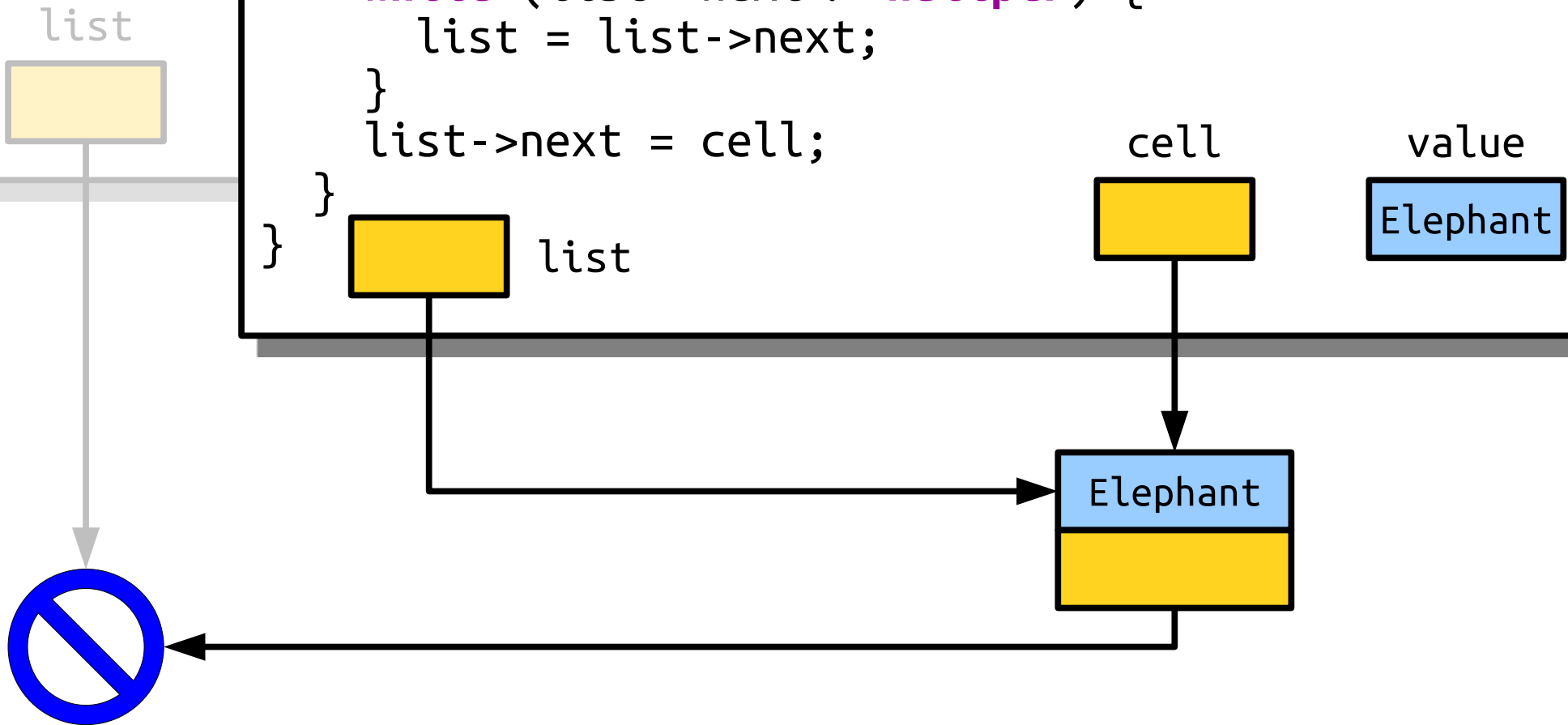


value



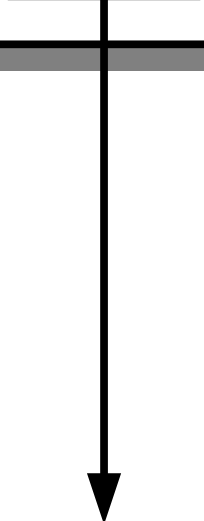
```
int main() {  
    Cell* list;  
    appendT  
    appendT  
    appendT  
    appendT  
    /* ... ot  
}
```

```
void appendTo(Cell* list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        while (list->next != nullptr) {  
            list = list->next;  
        }  
        list->next = cell;  
    }  
}
```



```
int main() {  
    Cell* list = nullptr;  
    appendTo(list, "Elephant");  
    appendTo(list, "Sunfish");  
    appendTo(list, "Whale");  
    appendTo(list, "Piraracu");  
  
    /* ... other listy things. ... */  
}
```

list



What Went Wrong (This Other Time)?

```
int main() {
    Cell* list = nullptr;
    appendTo(list, "Elephant");
    appendTo(list, "Sunfish");
    appendTo(list, "Whale");
    appendTo(list, "Piraracu");

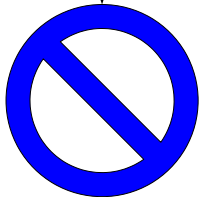
    /* ... other listy things. ... */
}
```

```
int main() {  
    Cell* list = nullptr;  
    appendTo(list, "Elephant");  
    appendTo(list, "Sunfish");  
    appendTo(list, "Whale");  
    appendTo(list, "Piraracu");  
  
    /* ... other listy things. ... */  
}
```



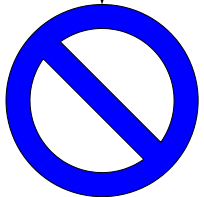
```
int main() {  
    Cell* list = nullptr;  
    appendTo(list, "Elephant");  
    appendTo(list, "Sunfish");  
    appendTo(list, "Whale");  
    appendTo(list, "Piraracu");  
  
    /* ... other listy things. ... */  
}
```

list



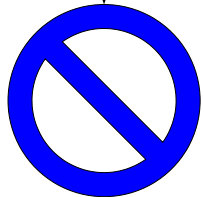
```
int main() {  
    Cell* list = nullptr;  
    appendTo(list, "Elephant");  
    appendTo(list, "Sunfish");  
    appendTo(list, "Whale");  
    appendTo(list, "Piraracu");  
  
    /* ... other listy things. ... */  
}
```

list



```
int main() {  
    Cell* list = nullptr;  
    appendT  
    appendT  
    appendT  
    appendT  
    /* ... ot  
}
```

list



```
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        while (list->next != nullptr) {  
            list = list->next;  
        }  
        list->next = cell;  
    }  
}
```

value

Elephant

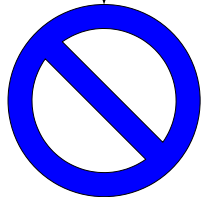
```
int main() {
    Cell* list;
    appendT
    appendT
    appendT
    appendT
    /* ... ot
}
```

```
void appendTo(Cell*& list, const string& value) {
    Cell* cell = new Cell;
    cell->value = value;
    cell->next = nullptr;
    if (list == nullptr) {
        list = cell;
    } else {
        while (list->next != nullptr) {
            list = list->next;
        }
        list->next = cell;
    }
}
```

list



value




```
int main() {  
    Cell* list = nullptr;  
    appendT  
    appendT  
    appendT  
    appendT  
    /* ... ot  
}  
  
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        while (list->next != nullptr) {  
            list = list->next;  
        }  
        list->next = cell;  
    }  
}
```

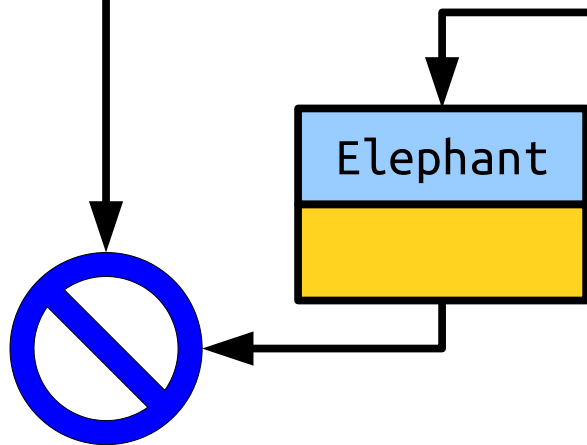

list



cell

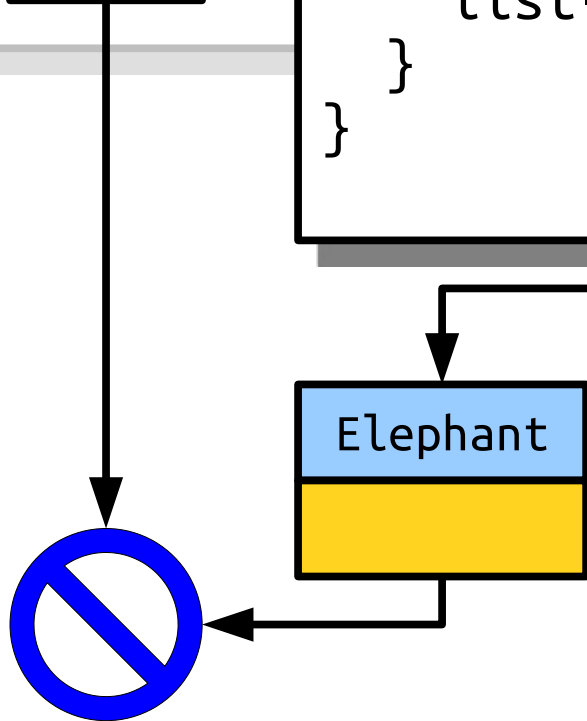
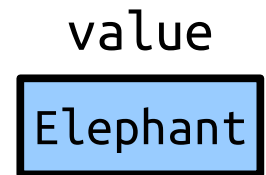
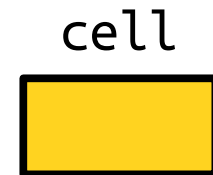


value
Elephant



```
int main() {  
    Cell* list = nullptr;  
    appendT  
    appendT  
    appendT  
    appendT  
    /* ... ot  
}
```

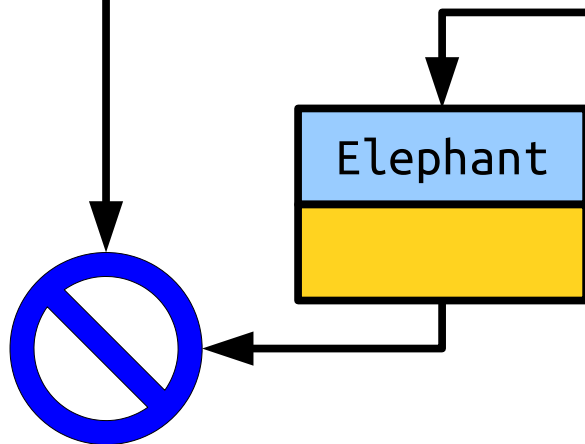
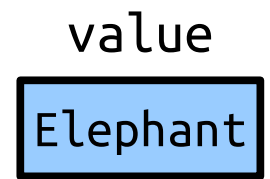
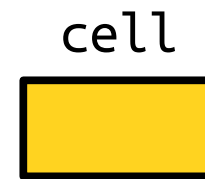
```
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        while (list->next != nullptr) {  
            list = list->next;  
        }  
        list->next = cell;  
    }  
}
```



```
int main() {
    Cell* list;
    appendT
    appendT
    appendT
    appendT
    /* ... ot
}
```

```
void appendTo(Cell*& list, const string& value) {
    Cell* cell = new Cell;
    cell->value = value;
    cell->next = nullptr;

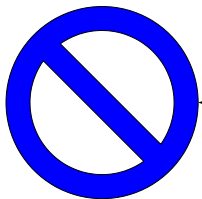
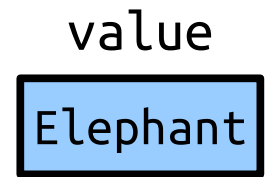
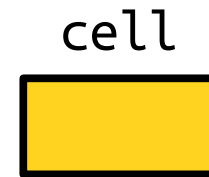
    if (list == nullptr) {
        list = cell;
    } else {
        while (list->next != nullptr) {
            list = list->next;
        }
        list->next = cell;
    }
}
```



```
int main() {
    Cell* list;
    appendT
    appendT
    appendT
    appendT
    /* ... ot
}
```

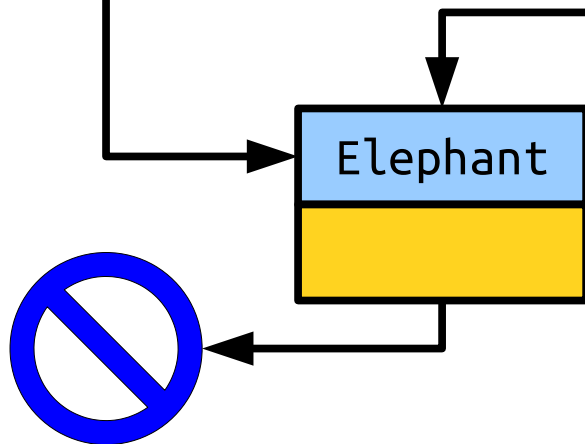
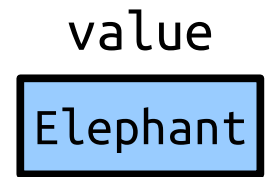
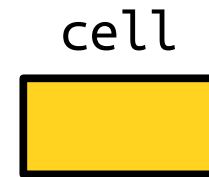
```
void appendTo(Cell*& list, const string& value) {
    Cell* cell = new Cell;
    cell->value = value;
    cell->next = nullptr;

    if (list == nullptr) {
        list = cell;
    } else {
        while (list->next != nullptr) {
            list = list->next;
        }
        list->next = cell;
    }
}
```



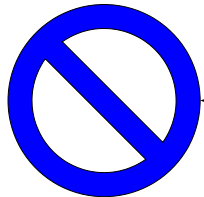
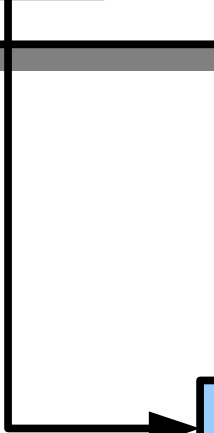

```
int main() {  
    Cell* list;  
    appendT  
    appendT  
    appendT  
    appendT  
    /* ... ot  
}
```

```
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        while (list->next != nullptr) {  
            list = list->next;  
        }  
        list->next = cell;  
    }  
}
```



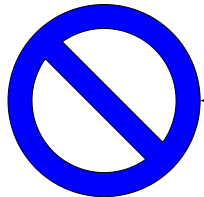
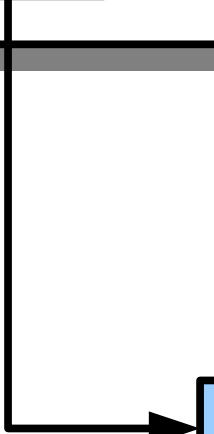
```
int main() {  
    Cell* list = nullptr;  
    appendTo(list, "Elephant");  
    appendTo(list, "Sunfish");  
    appendTo(list, "Whale");  
    appendTo(list, "Piraracu");  
  
    /* ... other listy things. ... */  
}
```

list



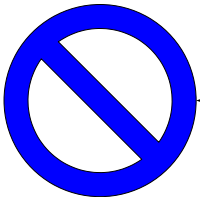
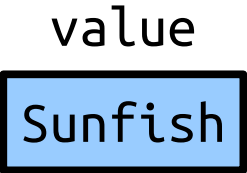
```
int main() {  
    Cell* list = nullptr;  
    appendTo(list, "Elephant");  
    appendTo(list, "Sunfish");  
    appendTo(list, "whale");  
    appendTo(list, "Piraracu");  
  
    /* ... other listy things. ... */  
}
```

list



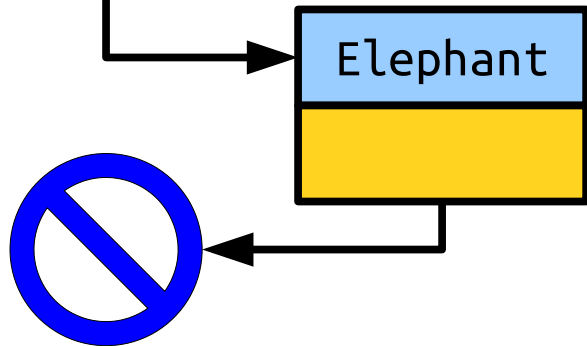
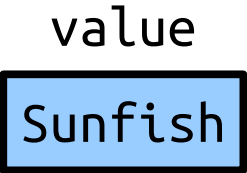
```
int main() {  
    Cell* list = nullptr;  
    appendT  
    appendT  
    appendT  
    /* ... ot  
}
```

```
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        while (list->next != nullptr) {  
            list = list->next;  
        }  
        list->next = cell;  
    }  
}
```



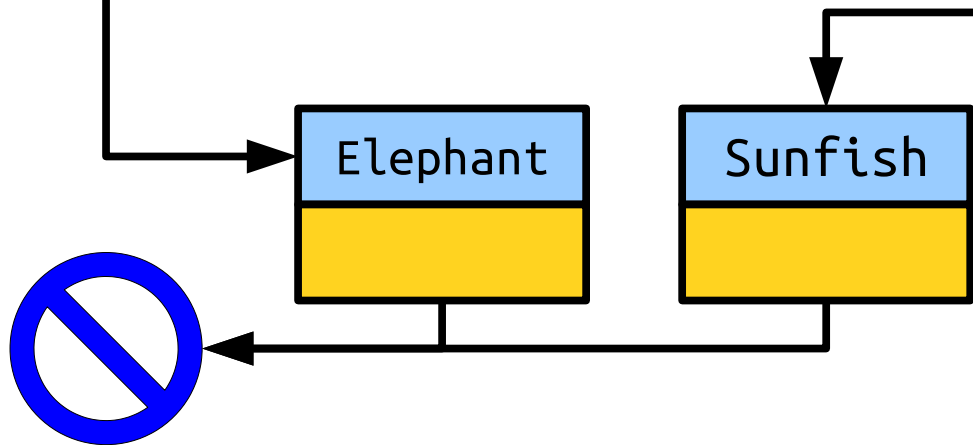
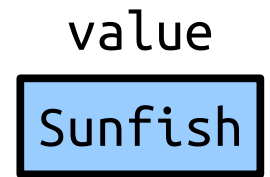
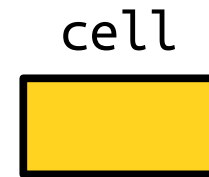
```
int main() {  
    Cell* list = nullptr;  
    appendT  
    appendT  
    appendT  
    appendT  
  
    /* ... ot  
}
```

```
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        while (list->next != nullptr) {  
            list = list->next;  
        }  
        list->next = cell;  
    }  
}
```



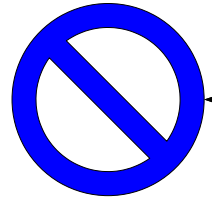
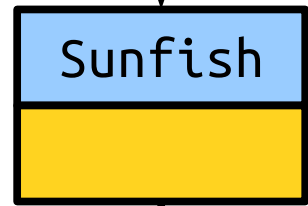
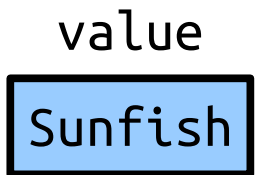
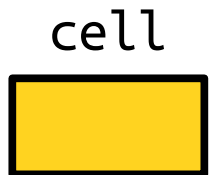
```
int main() {  
    Cell* list = nullptr;  
    appendT  
    appendT  
    appendT  
    /* ... ot  
}
```

```
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        while (list->next != nullptr) {  
            list = list->next;  
        }  
        list->next = cell;  
    }  
}
```



```
int main() {
    Cell* list;
    appendT
    appendT
    appendT
    appendT
    /* ... ot
}
```

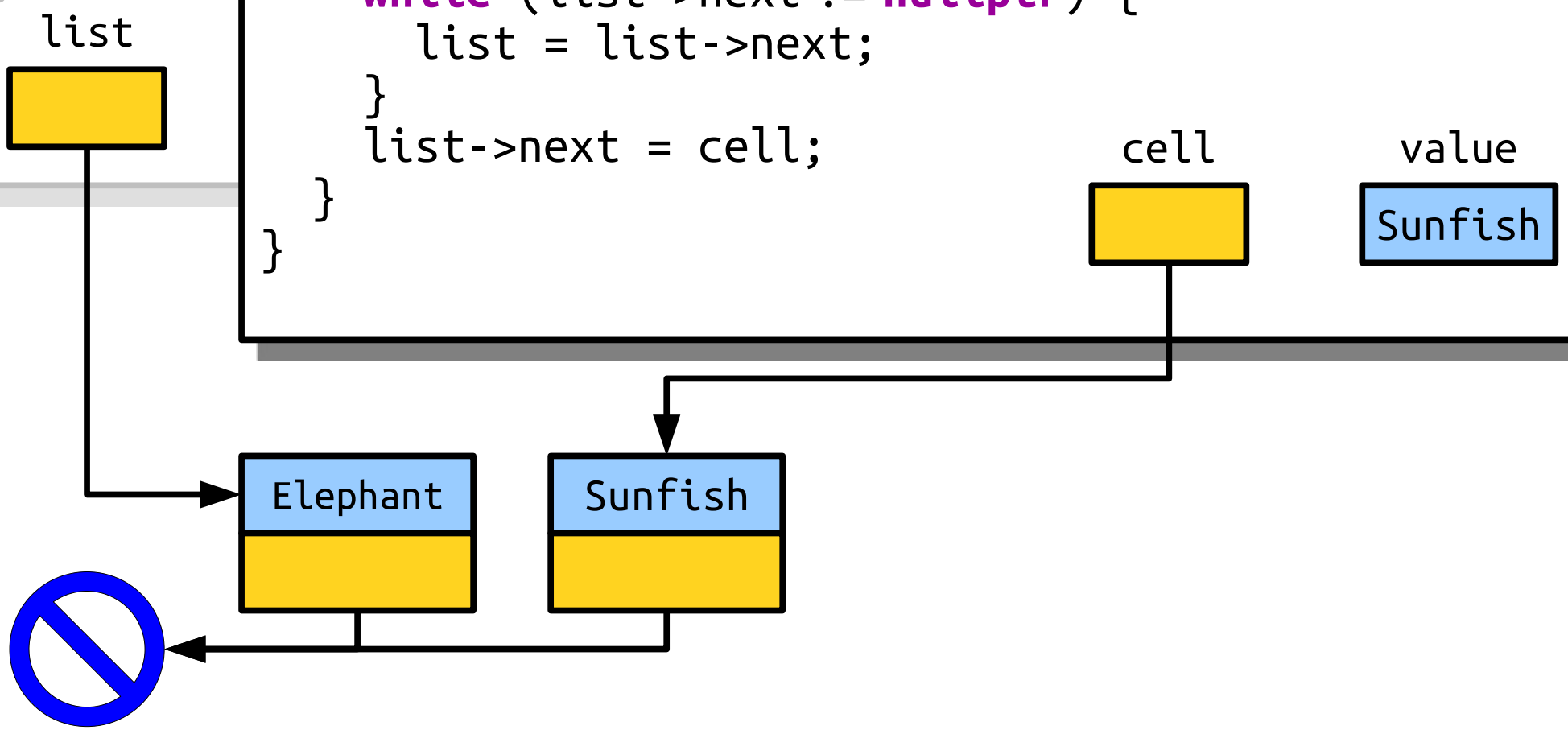
```
void appendTo(Cell*& list, const string& value) {
    Cell* cell = new Cell;
    cell->value = value;
    cell->next = nullptr;
    if (list == nullptr) {
        list = cell;
    } else {
        while (list->next != nullptr) {
            list = list->next;
        }
        list->next = cell;
    }
}
```



```
int main() {
    Cell* list;
    appendT
    appendT
    appendT
    appendT
    /* ... ot
}
}
```

```
void appendTo(Cell*& list, const string& value) {
    Cell* cell = new Cell;
    cell->value = value;
    cell->next = nullptr;

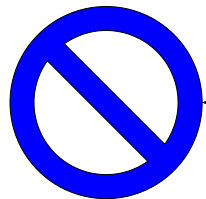
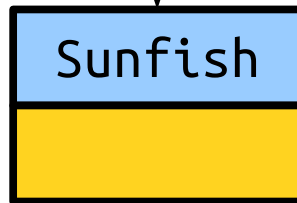
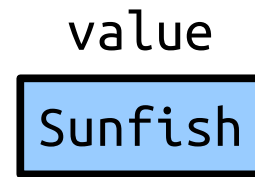
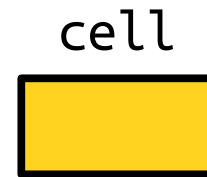
    if (list == nullptr) {
        list = cell;
    } else {
        while (list->next != nullptr) {
            list = list->next;
        }
        list->next = cell;
    }
}
```




```
int main() {
    Cell* list;
    appendT
    appendT
    appendT
    appendT
    /* ... ot
}
```

```
void appendTo(Cell*& list, const string& value) {
    Cell* cell = new Cell;
    cell->value = value;
    cell->next = nullptr;

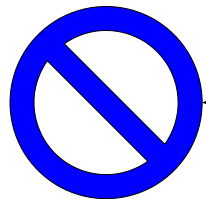
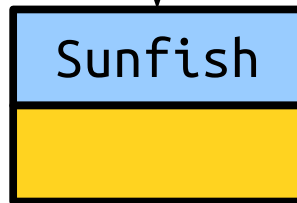
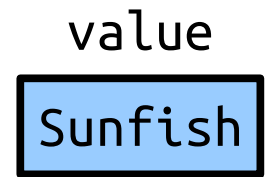
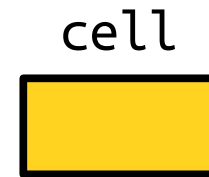
    if (list == nullptr) {
        list = cell;
    } else {
        while (list->next != nullptr) {
            list = list->next;
        }
        list->next = cell;
    }
}
```



```
int main() {
    Cell* list;
    appendT
    appendT
    appendT
    appendT
    /* ... ot
}
```

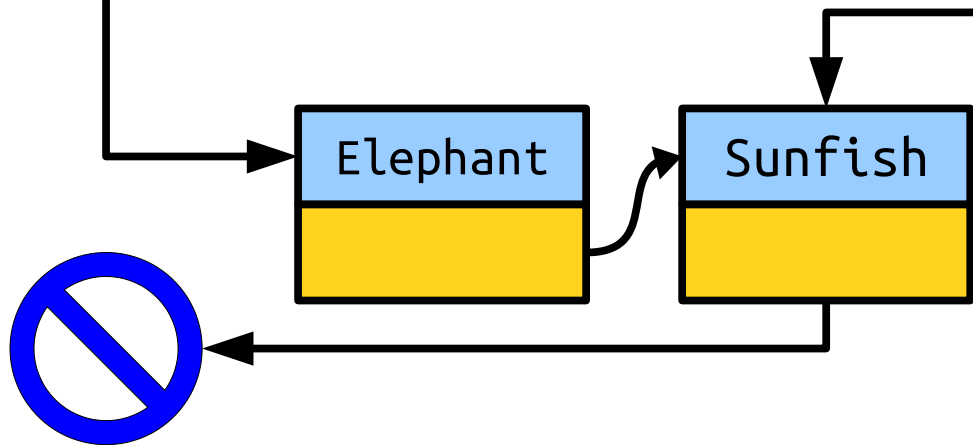
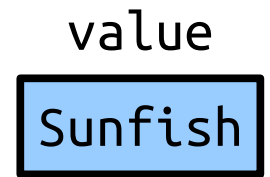
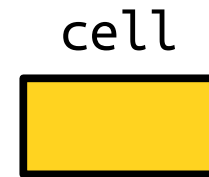
```
void appendTo(Cell*& list, const string& value) {
    Cell* cell = new Cell;
    cell->value = value;
    cell->next = nullptr;

    if (list == nullptr) {
        list = cell;
    } else {
        while (list->next != nullptr) {
            list = list->next;
        }
        list->next = cell;
    }
}
```



```
int main() {  
    Cell* list = nullptr;  
    appendT  
    appendT  
    appendT  
    /* ... ot  
}
```

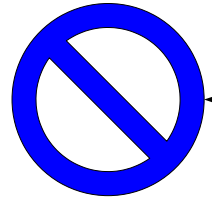
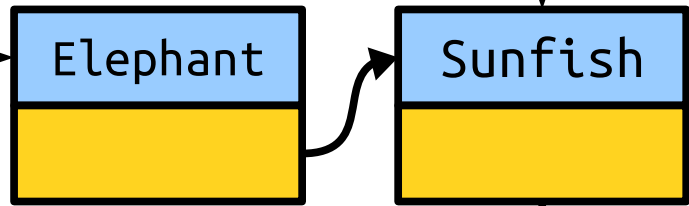
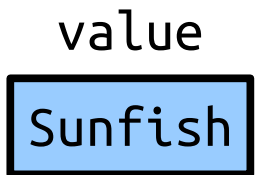
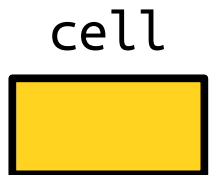
```
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        while (list->next != nullptr) {  
            list = list->next;  
        }  
        list->next = cell;  
    }  
}
```



```
int main() {
    Cell* list;
    appendT
    appendT
    appendT
    appendT
    /* ... ot
}
```

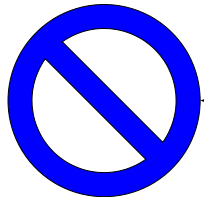
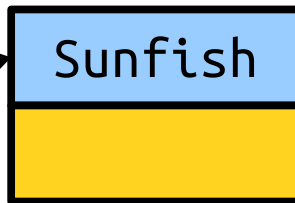
```
void appendTo(Cell*& list, const string& value) {
    Cell* cell = new Cell;
    cell->value = value;
    cell->next = nullptr;

    if (list == nullptr) {
        list = cell;
    } else {
        while (list->next != nullptr) {
            list = list->next;
        }
        list->next = cell;
    }
}
```



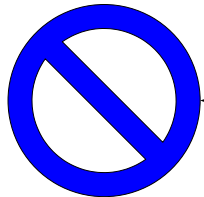
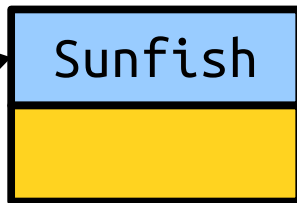
```
int main() {  
    Cell* list = nullptr;  
    appendTo(list, "Elephant");  
    appendTo(list, "Sunfish");  
    appendTo(list, "whale");  
    appendTo(list, "Piraracu");  
  
    /* ... other listy things. ... */  
}
```

list



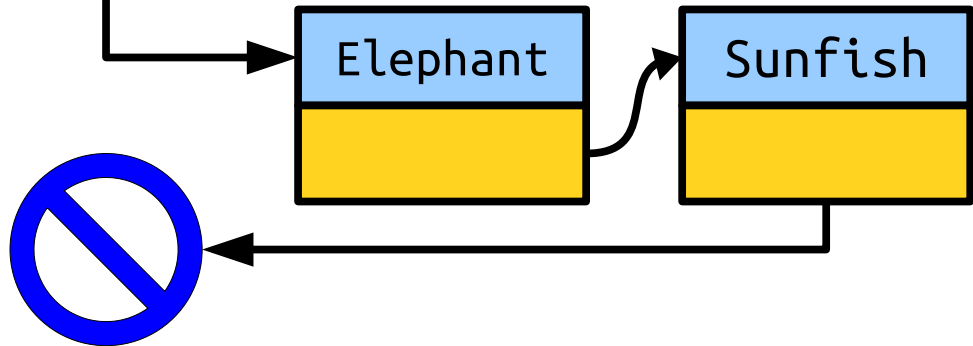
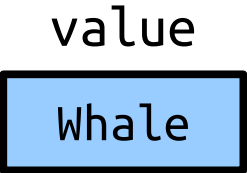
```
int main() {  
    Cell* list = nullptr;  
    appendTo(list, "Elephant");  
    appendTo(list, "Sunfish");  
    appendTo(list, "Whale");  
    appendTo(list, "Piraracu");  
  
    /* ... other listy things. ... */  
}
```

list



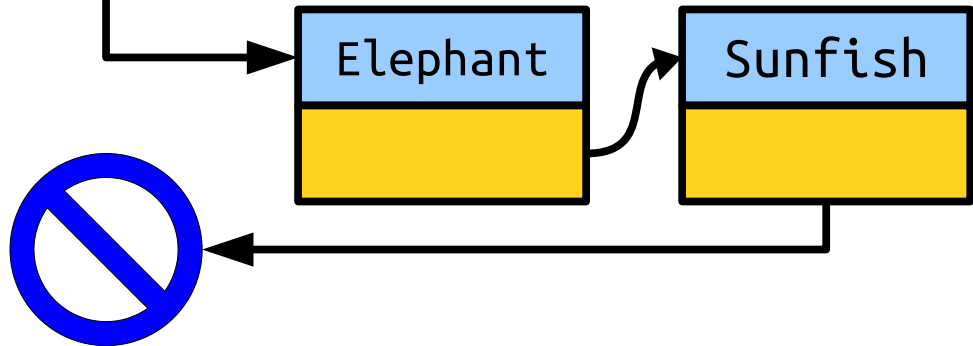
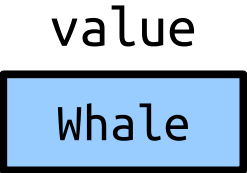
```
int main() {  
    Cell* list = nullptr;  
    appendT("Elephant");  
    appendT("Sunfish");  
    appendT("Whale");  
    /* ... other code ... */  
}
```

```
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        while (list->next != nullptr) {  
            list = list->next;  
        }  
        list->next = cell;  
    }  
}
```



```
int main() {  
    Cell* list = nullptr;  
    appendT("Elephant");  
    appendT("Sunfish");  
    appendT("Whale");  
    /* ... other code ...  
}
```

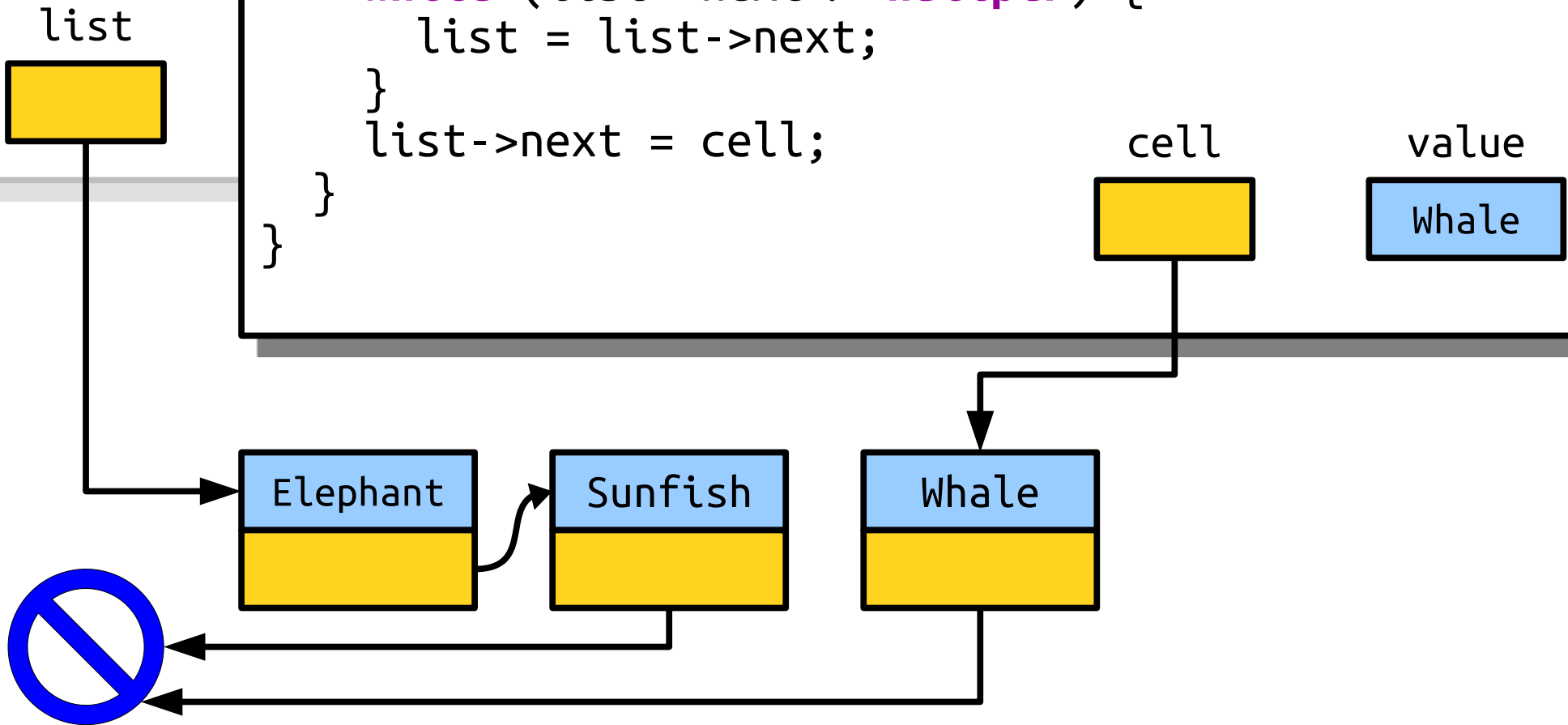
```
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        while (list->next != nullptr) {  
            list = list->next;  
        }  
        list->next = cell;  
    }  
}
```




```
int main() {
    Cell* list;
    appendTo(list, "Elephant");
    appendTo(list, "Sunfish");
    appendTo(list, "Whale");
    /* ... other code ...
}
```

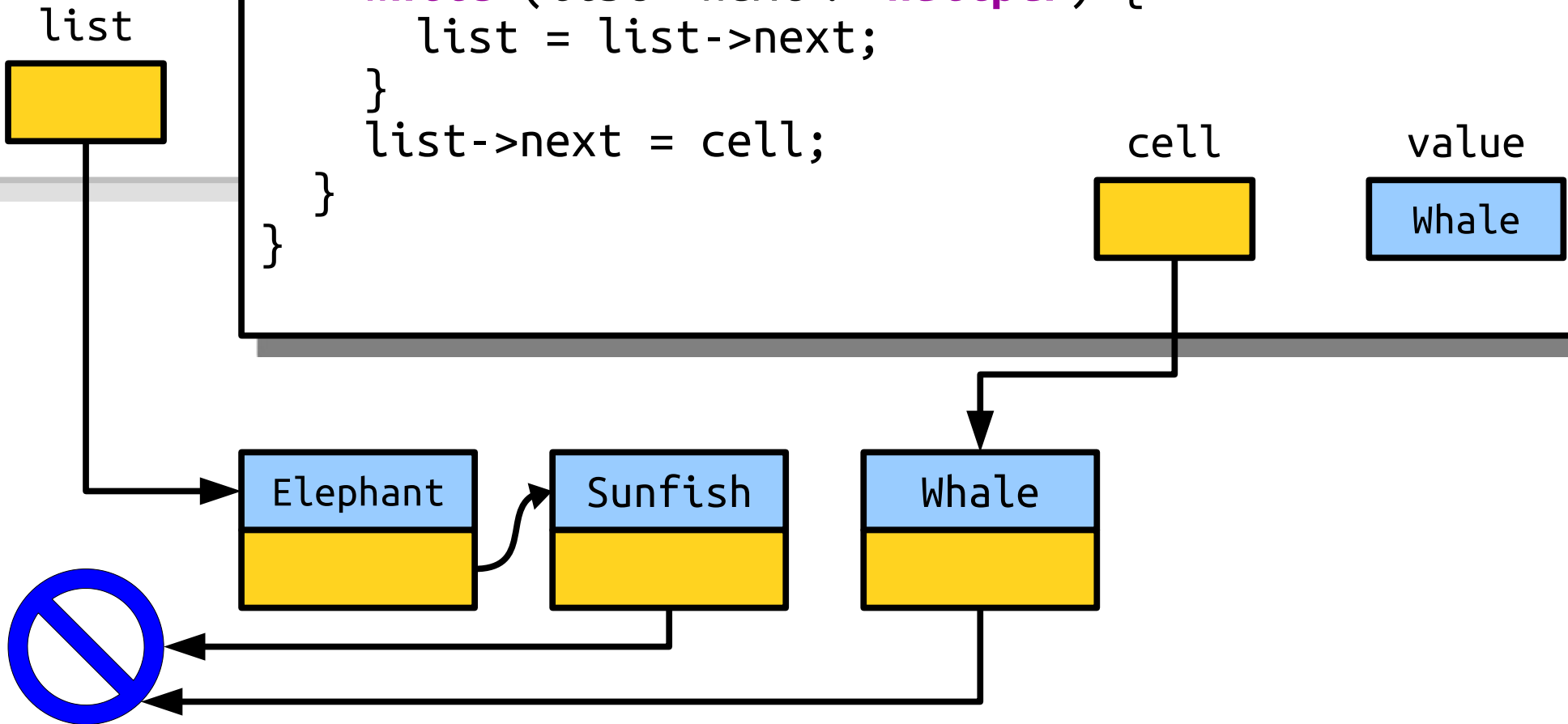
```
void appendTo(Cell*& list, const string& value) {
    Cell* cell = new Cell;
    cell->value = value;
    cell->next = nullptr;

    if (list == nullptr) {
        list = cell;
    } else {
        while (list->next != nullptr) {
            list = list->next;
        }
        list->next = cell;
    }
}
```



```
int main() {  
    Cell* list = nullptr;  
    appendT("Elephant");  
    appendT("Sunfish");  
    appendT("Whale");  
    /* ... other code ... */  
}
```

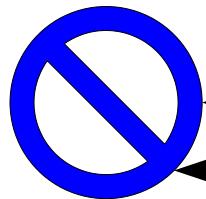
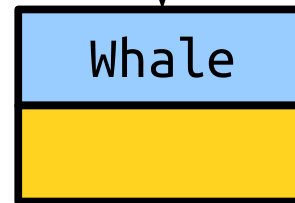
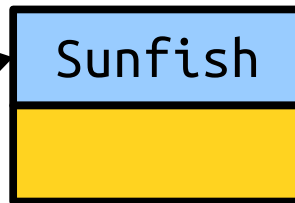
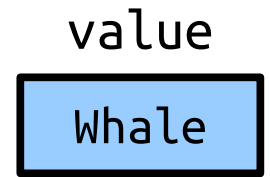
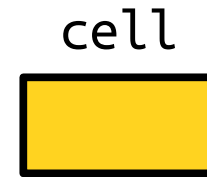
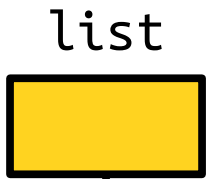
```
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        while (list->next != nullptr) {  
            list = list->next;  
        }  
        list->next = cell;  
    }  
}
```



```
int main() {
    Cell* list;
    appendT
    appendT
    appendT
    appendT
    appendT
    /* ... ot
}
```

```
void appendTo(Cell*& list, const string& value) {
    Cell* cell = new Cell;
    cell->value = value;
    cell->next = nullptr;

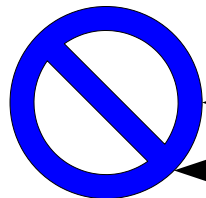
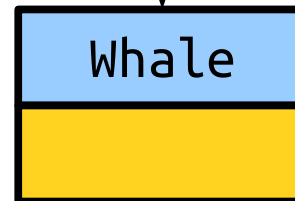
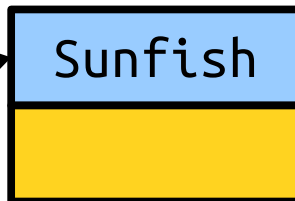
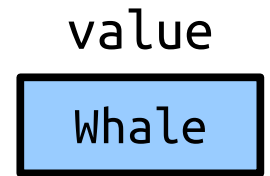
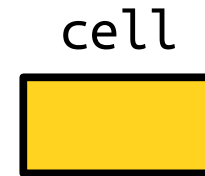
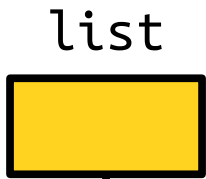
    if (list == nullptr) {
        list = cell;
    } else {
        while (list->next != nullptr) {
            list = list->next;
        }
        list->next = cell;
    }
}
```



```
int main() {
    Cell* list;
    appendT
    appendT
    appendT
    appendT
    appendT
    /* ... ot
}
```

```
void appendTo(Cell*& list, const string& value) {
    Cell* cell = new Cell;
    cell->value = value;
    cell->next = nullptr;

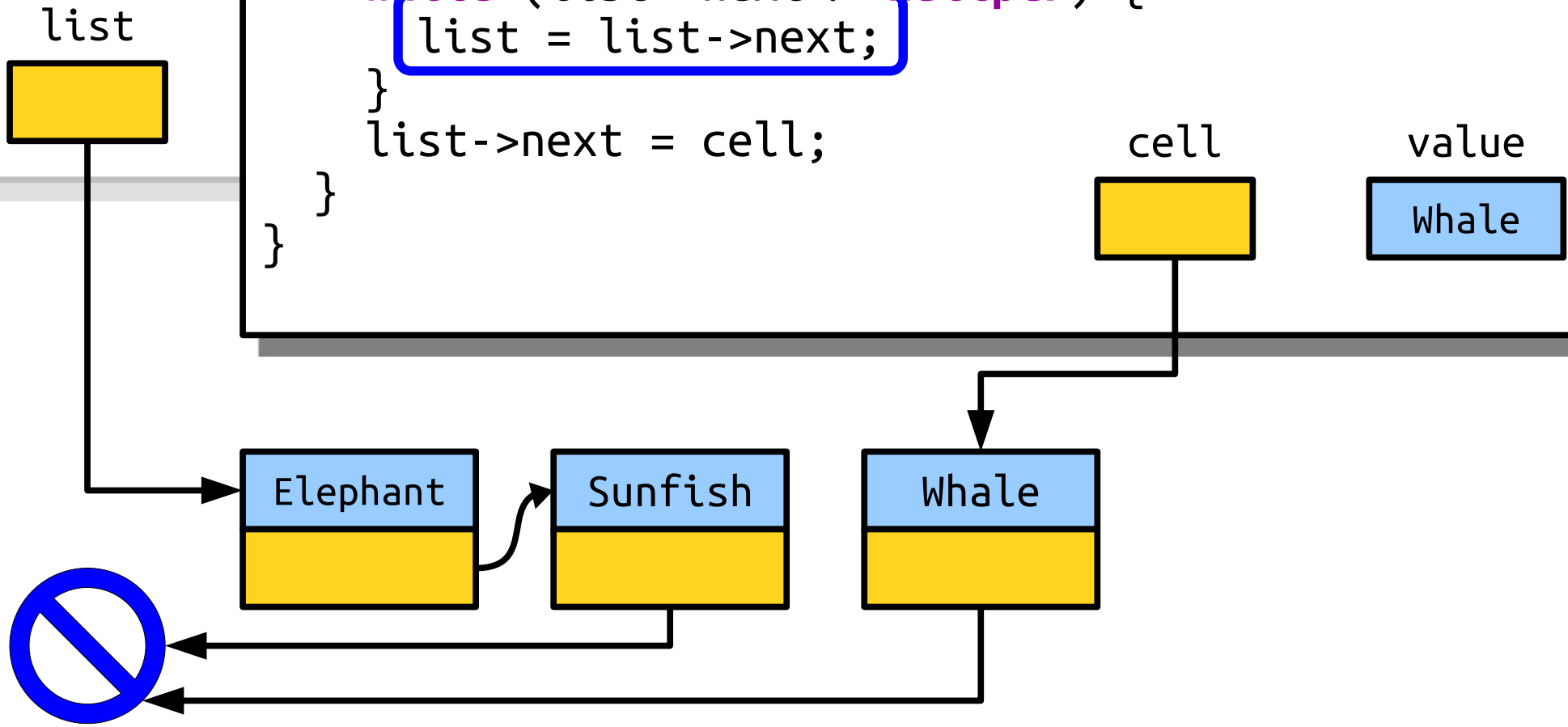
    if (list == nullptr) {
        list = cell;
    } else {
        while (list->next != nullptr) {
            list = list->next;
        }
        list->next = cell;
    }
}
```



```
int main() {
  Cell* list;
  appendT
  appendT
  appendT
  appendT
  appendT
  /* ... ot
}
```

```
void appendTo(Cell*& list, const string& value) {
  Cell* cell = new Cell;
  cell->value = value;
  cell->next = nullptr;

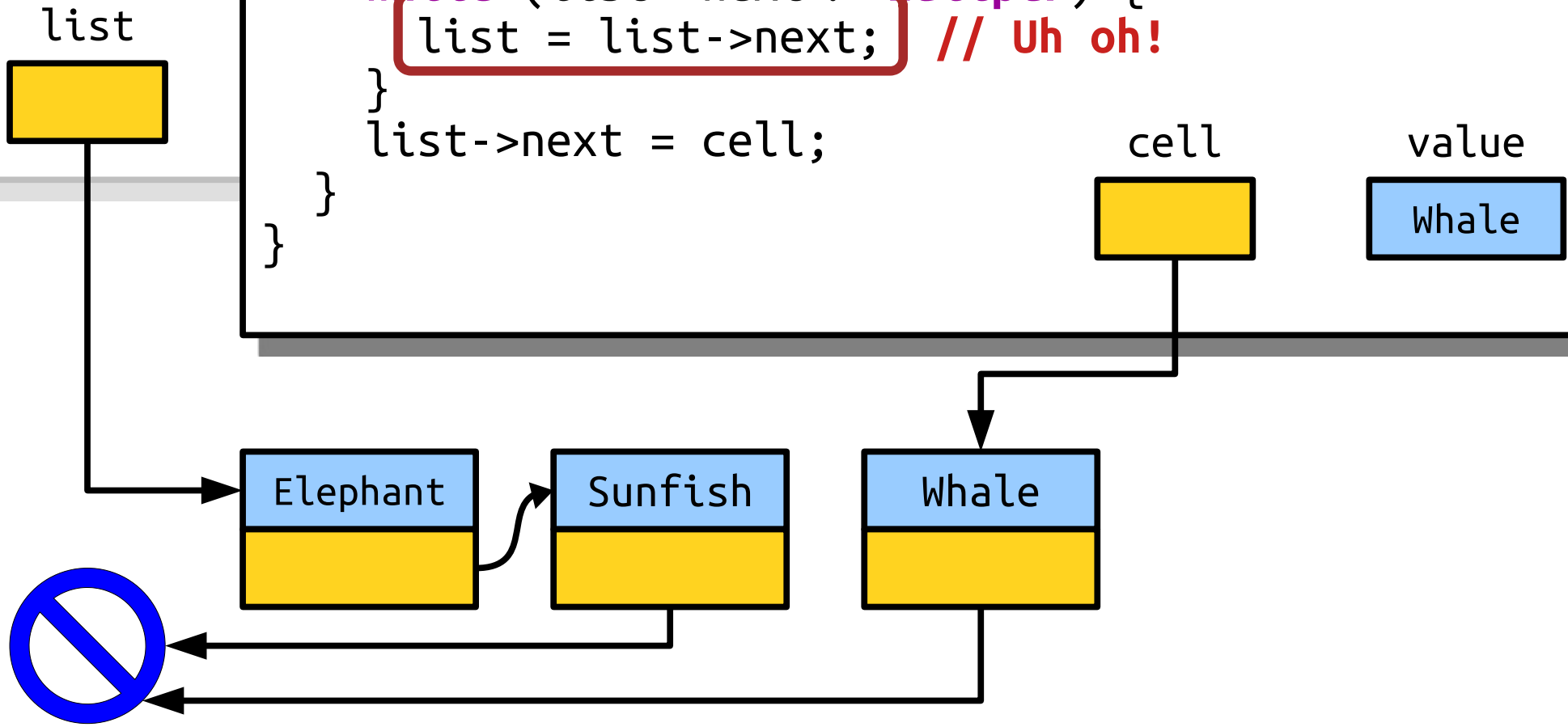
  if (list == nullptr) {
    list = cell;
  } else {
    while (list->next != nullptr) {
      list = list->next;
    }
    list->next = cell;
  }
}
```



```
int main() {
  Cell* list;
  appendT
  appendT
  appendT
  appendT
  appendT
  /* ... ot
}
```

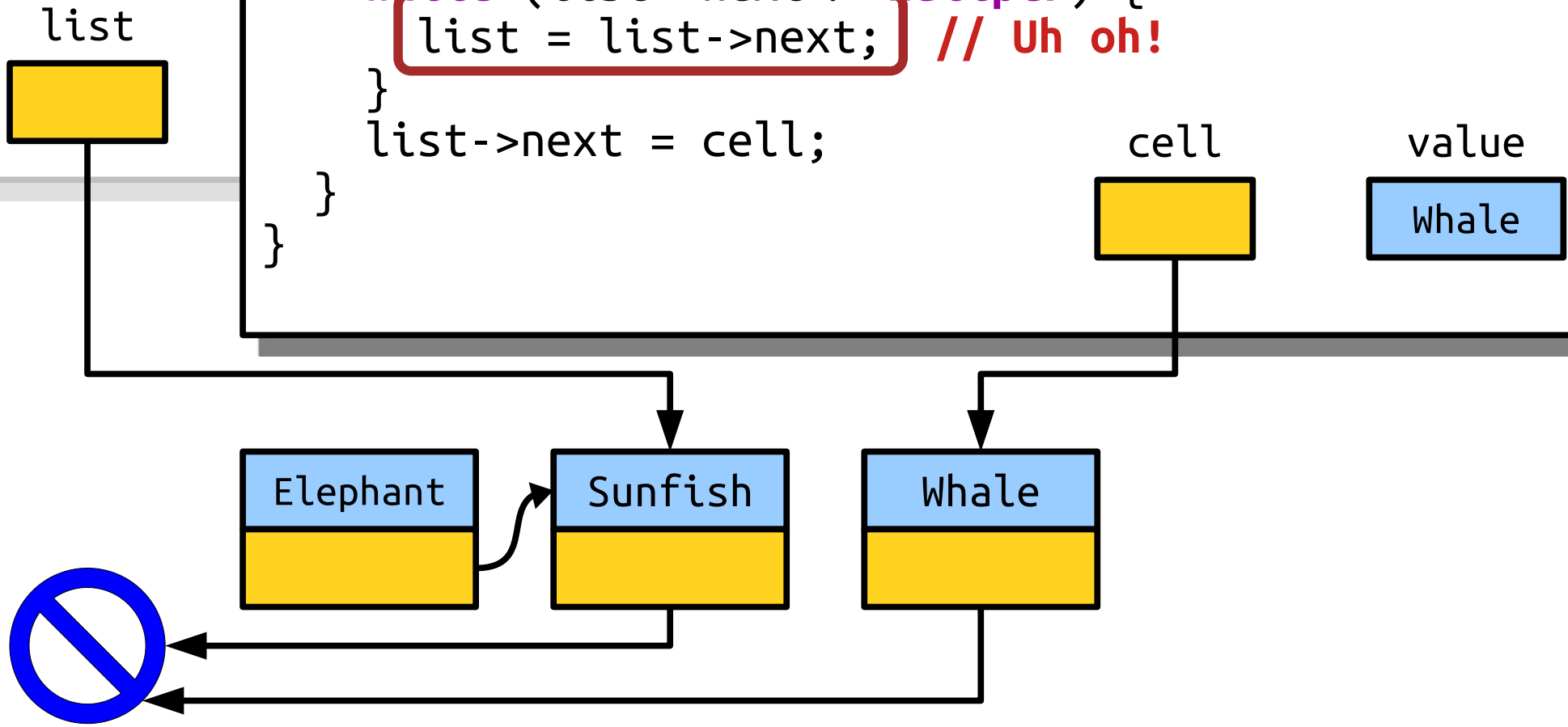
```
void appendTo(Cell*& list, const string& value) {
  Cell* cell = new Cell;
  cell->value = value;
  cell->next = nullptr;

  if (list == nullptr) {
    list = cell;
  } else {
    while (list->next != nullptr) {
      list = list->next; // Uh oh!
    }
    list->next = cell;
  }
}
```



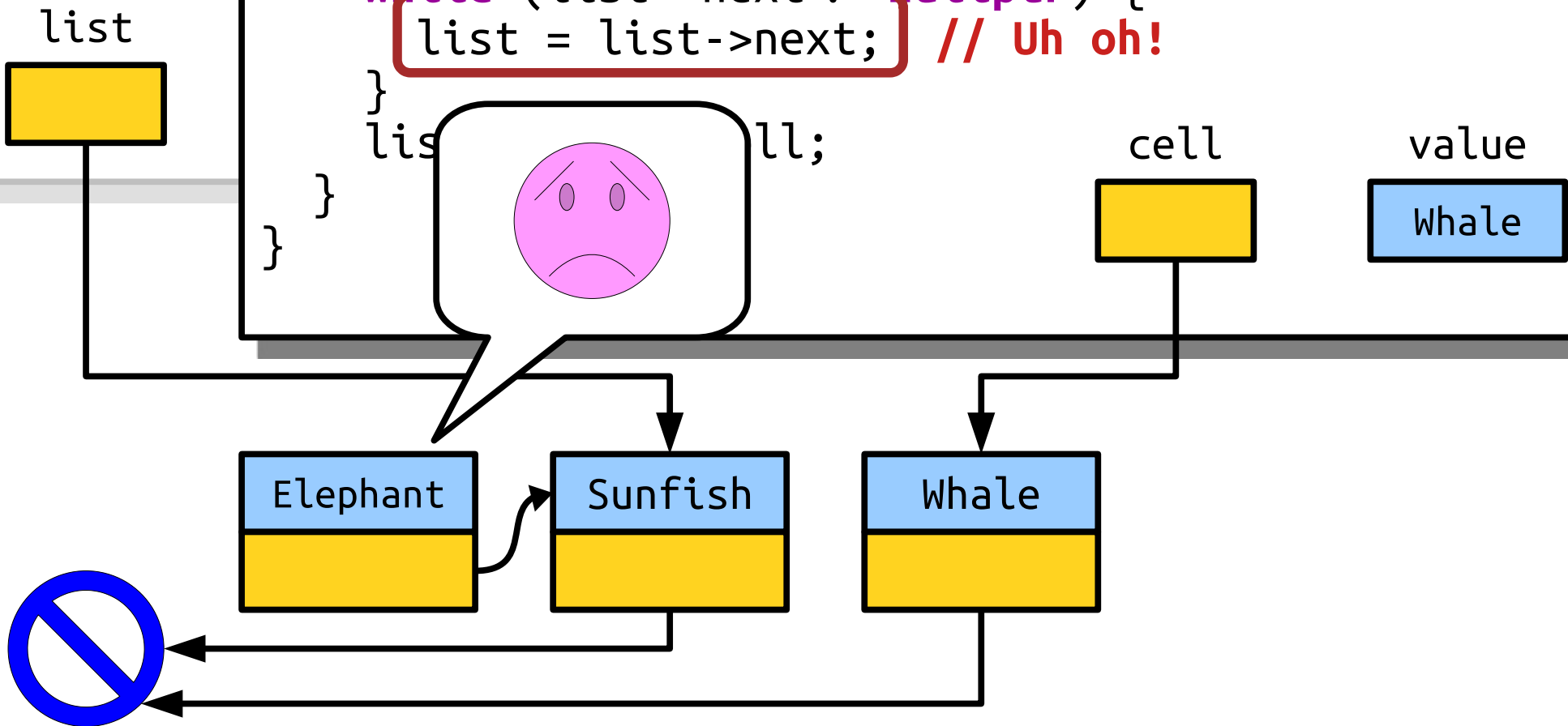
```
int main() {  
    Cell* list = nullptr;  
    appendT("Elephant");  
    appendT("Sunfish");  
    appendT("Whale");  
    /* ... other code ... */  
}
```

```
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        while (list->next != nullptr) {  
            list = list->next; // Uh oh!  
        }  
        list->next = cell;  
    }  
}
```



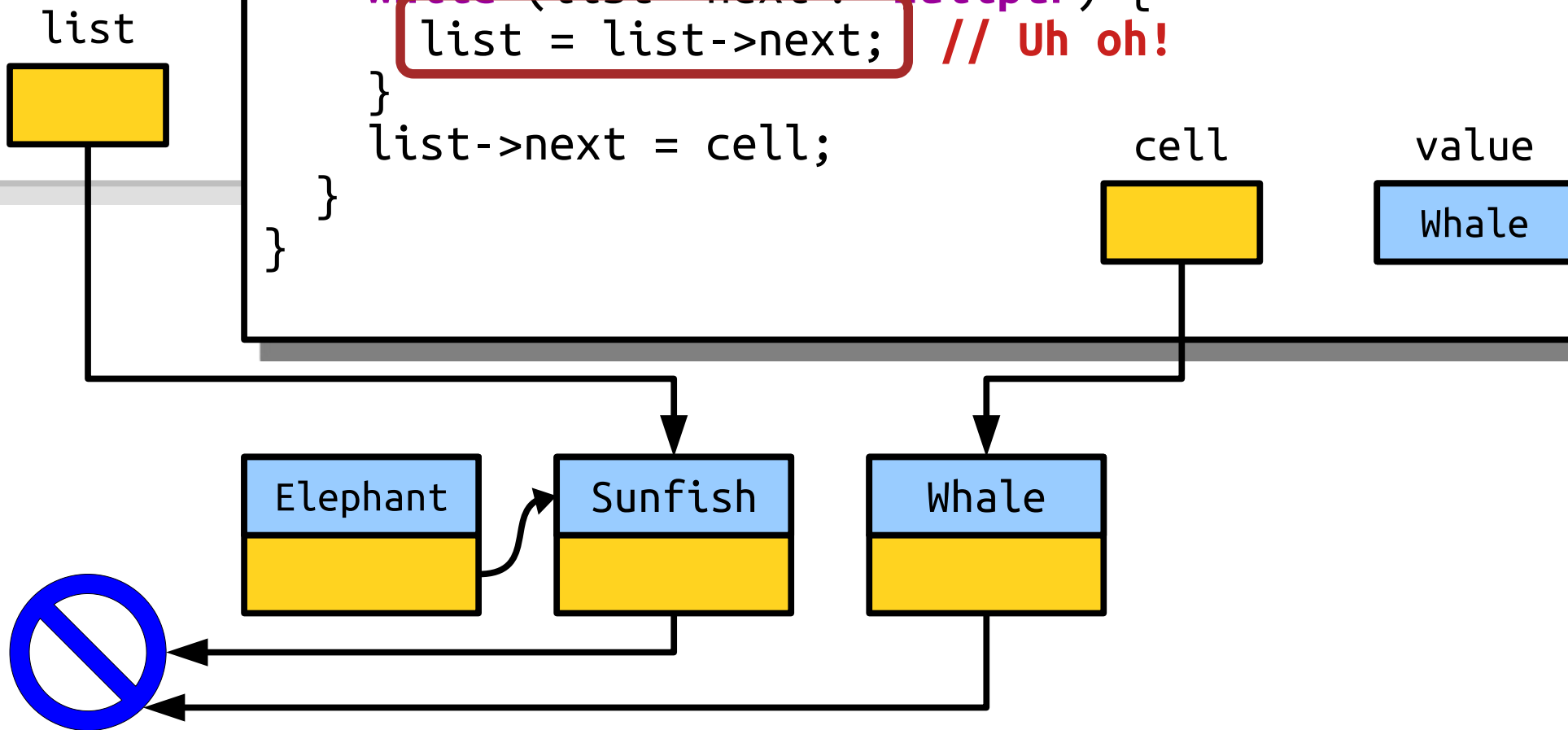
```
int main() {  
    Cell* list = nullptr;  
    appendTo(list, "Elephant");  
    appendTo(list, "Sunfish");  
    appendTo(list, "Whale");  
    /* ... other code ... */  
}
```

```
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        while (list->next != nullptr) {  
            list = list->next; // Uh oh!  
        }  
        list->next = cell;  
    }  
}
```



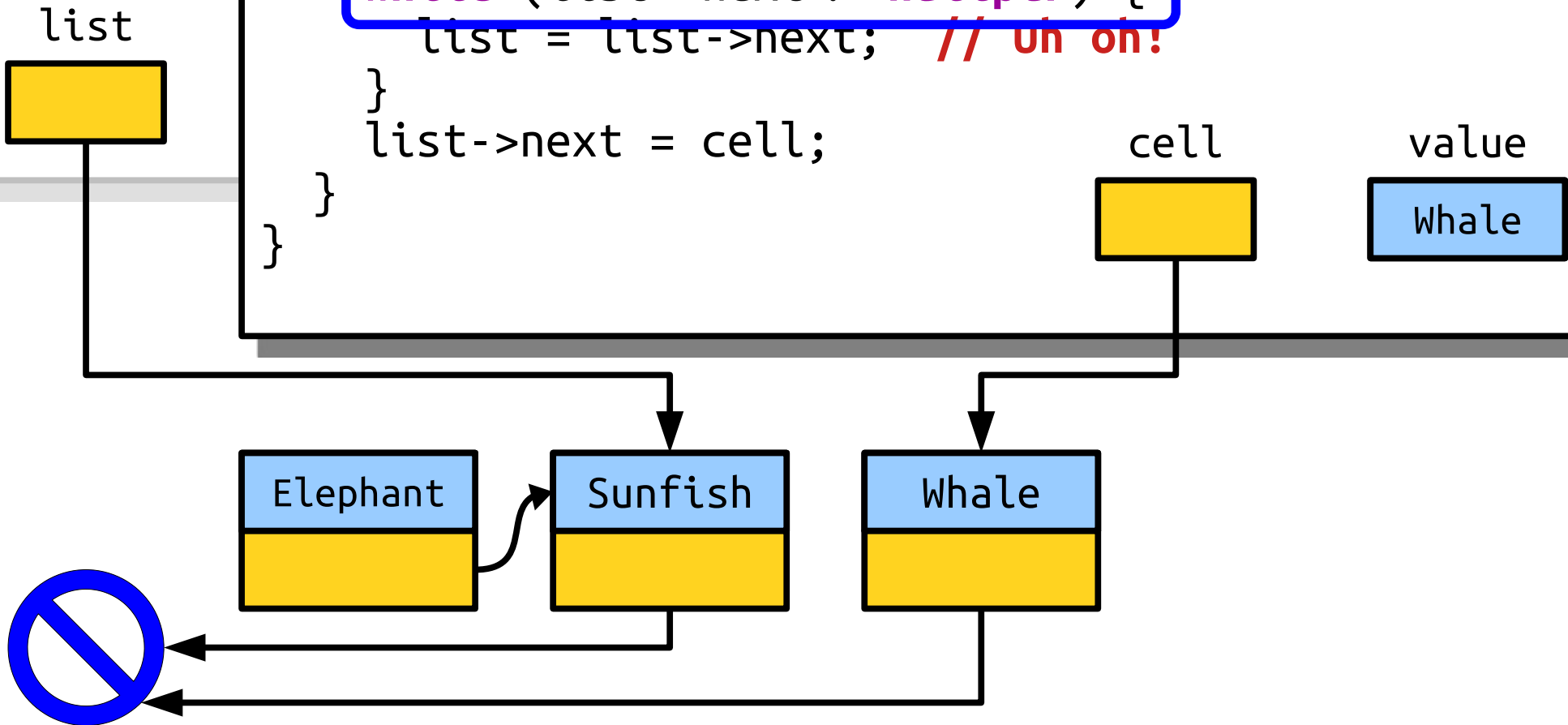

```
int main() {  
    Cell* list = nullptr;  
    appendTo(list, "Elephant");  
    appendTo(list, "Sunfish");  
    appendTo(list, "Whale");  
    /* ... other code ... */  
}
```

```
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        while (list->next != nullptr) {  
            list = list->next; // Uh oh!  
        }  
        list->next = cell;  
    }  
}
```



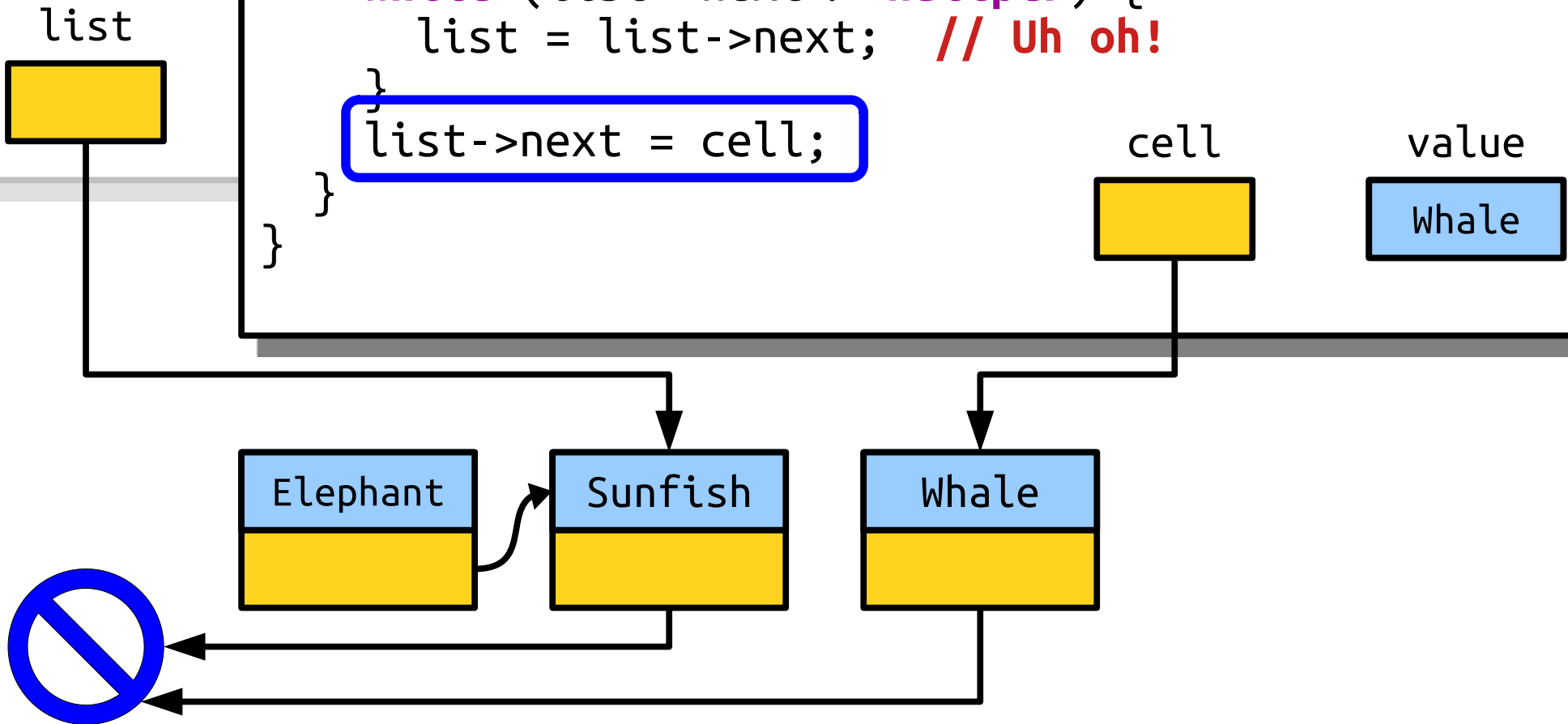
```
int main() {  
    Cell* list = nullptr;  
    appendT("Elephant");  
    appendT("Sunfish");  
    appendT("Whale");  
    /* ... other code ... */  
}
```

```
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        while (list->next != nullptr) {  
            list = list->next; // Uh oh!  
        }  
        list->next = cell;  
    }  
}
```



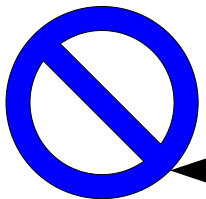
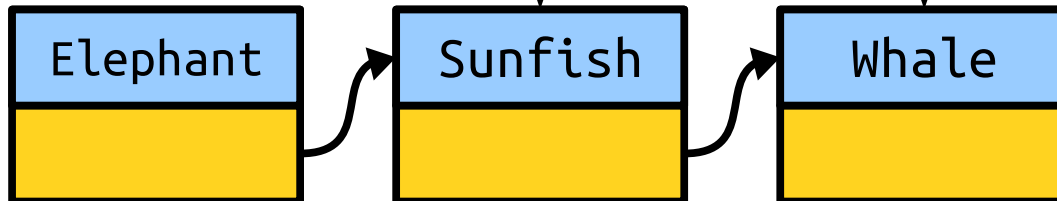
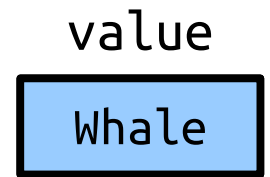
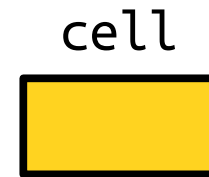
```
int main() {  
    Cell* list = nullptr;  
    appendT("Elephant");  
    appendT("Sunfish");  
    appendT("Whale");  
    /* ... other code ... */  
}
```

```
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        while (list->next != nullptr) {  
            list = list->next; // Uh oh!  
        }  
        list->next = cell;  
    }  
}
```



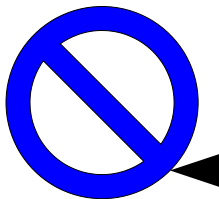
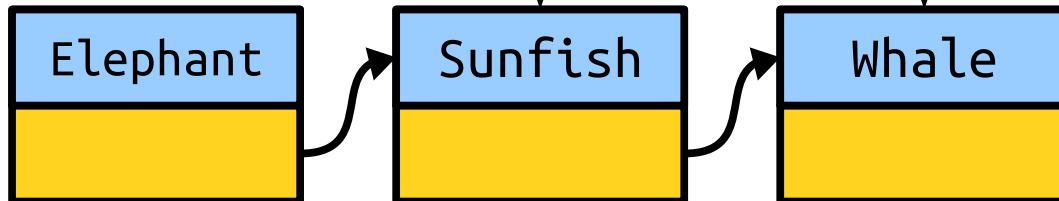
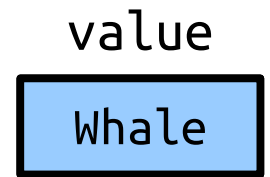
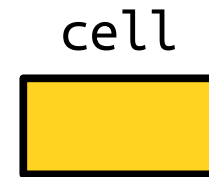
```
int main() {  
    Cell* list;  
    appendT  
    appendT  
    appendT  
    appendT  
    /* ... ot  
}
```

```
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        while (list->next != nullptr) {  
            list = list->next; // Uh oh!  
        }  
        list->next = cell;  
    }  
}
```



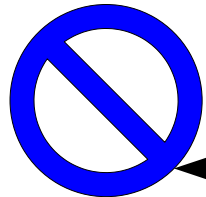
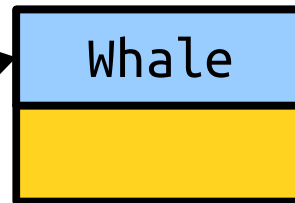
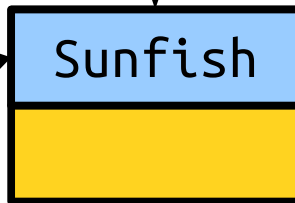
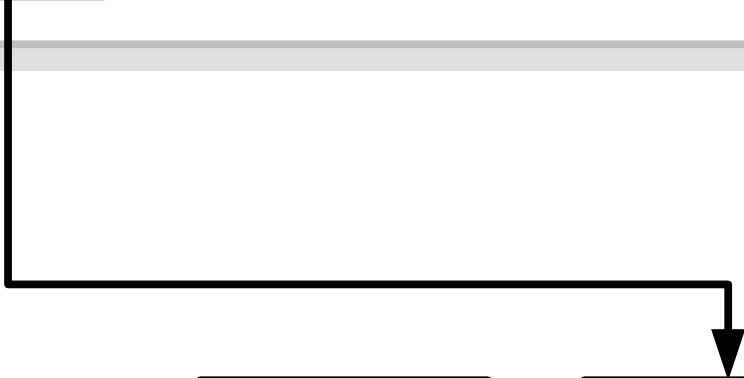
```
int main() {  
    Cell* list = nullptr;  
    appendT("Elephant");  
    appendT("Sunfish");  
    appendT("Whale");  
    /* ... other code ...  
}
```

```
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        while (list->next != nullptr) {  
            list = list->next; // Uh oh!  
        }  
        list->next = cell;  
    }  
}
```



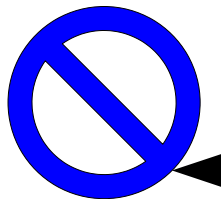
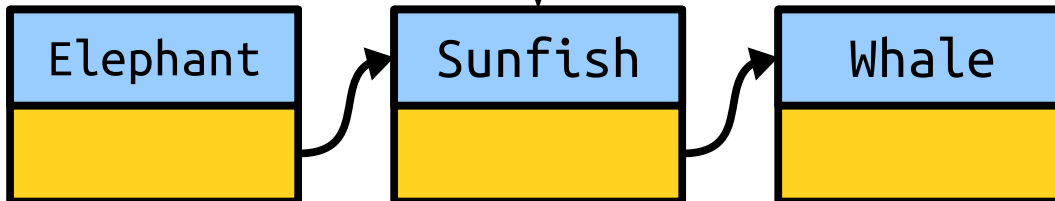
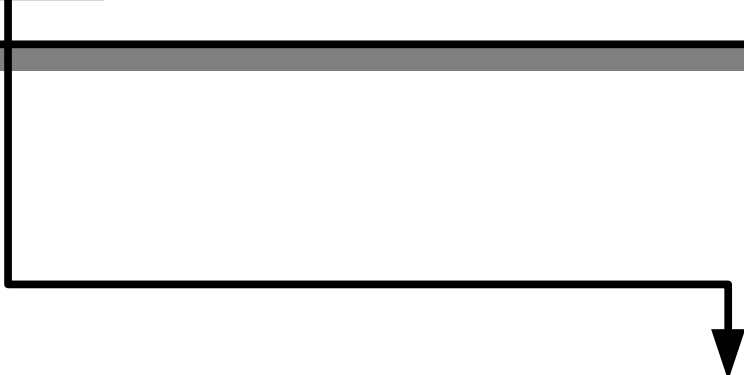
```
int main() {  
    Cell* list = nullptr;  
    appendTo(list, "Elephant");  
    appendTo(list, "Sunfish");  
    appendTo(list, "Whale");  
    appendTo(list, "Piraracu");  
  
    /* ... other listy things. ... */  
}
```

list



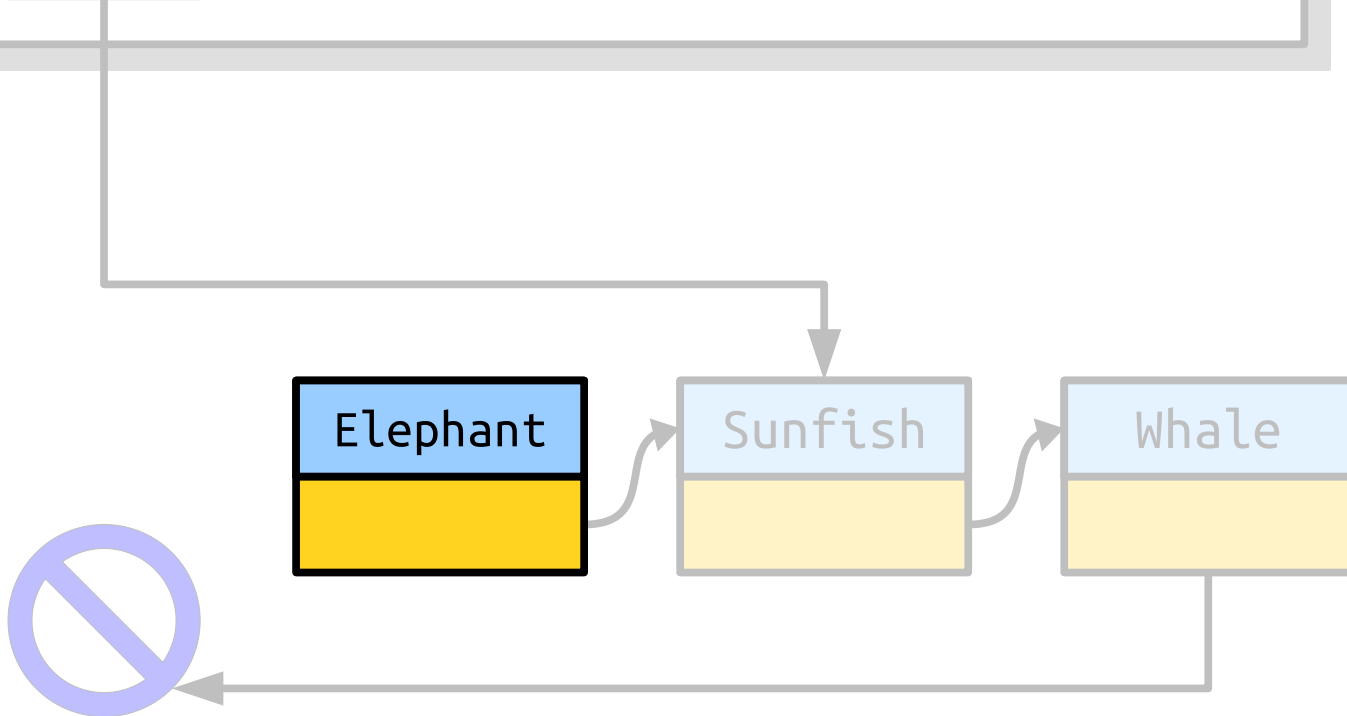
```
int main() {  
    Cell* list = nullptr;  
    appendTo(list, "Elephant");  
    appendTo(list, "Sunfish");  
    appendTo(list, "Whale");  
    appendTo(list, "Piraracu");  
  
    /* ... other listy things. ... */  
}
```

list



```
int main() {  
    Cell* list = nullptr;  
    appendTo(list, "Elephant");  
    appendTo(list, "Sunfish");  
    appendTo(list, "Whale");  
    appendTo(list, "Piraracu");  
  
    /* ... other listy things. ... */  
}
```

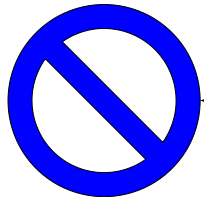
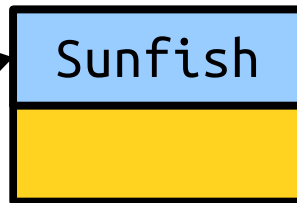
list



When passing in pointers by reference,
be careful not to change the pointer
unless you really want to change where it's
pointing!

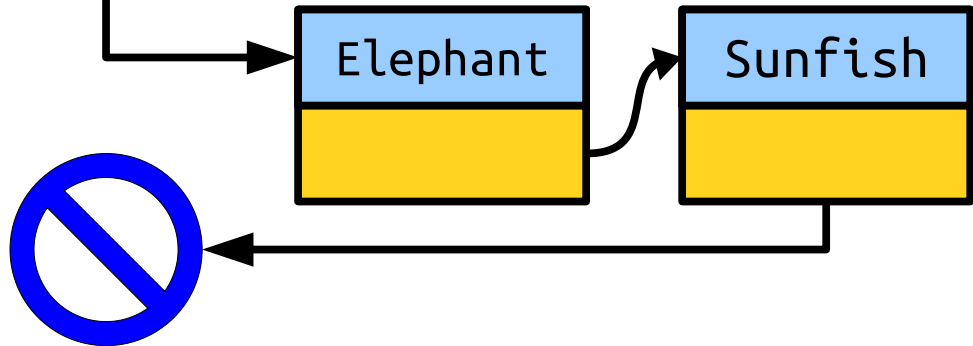
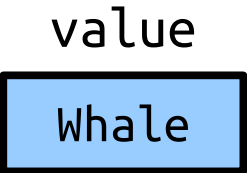
```
int main() {  
    Cell* list = nullptr;  
    appendTo(list, "Elephant");  
    appendTo(list, "Sunfish");  
    appendTo(list, "Whale");  
    appendTo(list, "Piraracu");  
  
    /* ... other listy things. ... */  
}
```

list



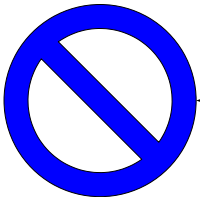
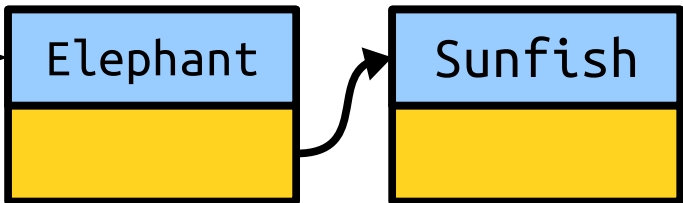
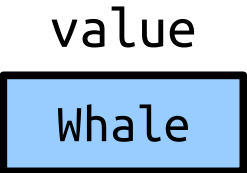
```
int main() {  
    Cell* list;  
    appendT  
    appendT  
    appendT  
    appendT  
    /* ... ot  
}
```

```
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        Cell* end = list;  
        while (end->next != nullptr) {  
            end = end->next;  
        }  
        end->next = cell;  
    }  
}
```



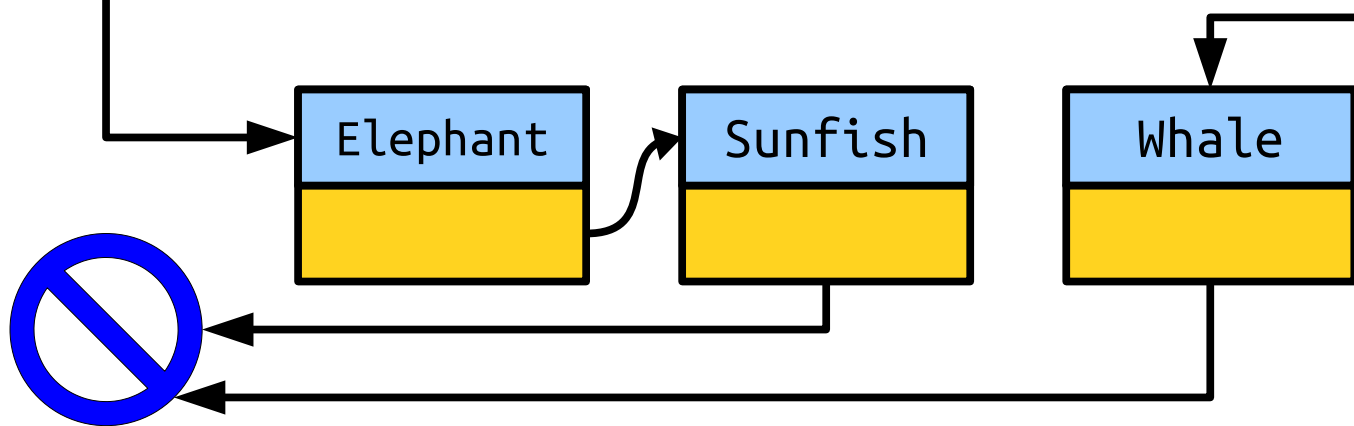
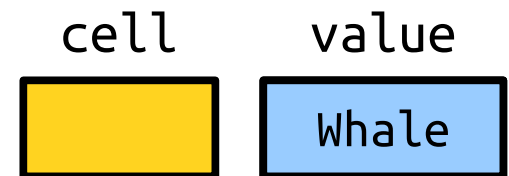
```
int main() {
    Cell* list;
    appendT
    appendT
    appendT
    appendT
    /* ... ot
}
```

```
void appendTo(Cell*& list, const string& value) {
    Cell* cell = new Cell;
    cell->value = value;
    cell->next = nullptr;
    if (list == nullptr) {
        list = cell;
    } else {
        Cell* end = list;
        while (end->next != nullptr) {
            end = end->next;
        }
        end->next = cell;
    }
}
```



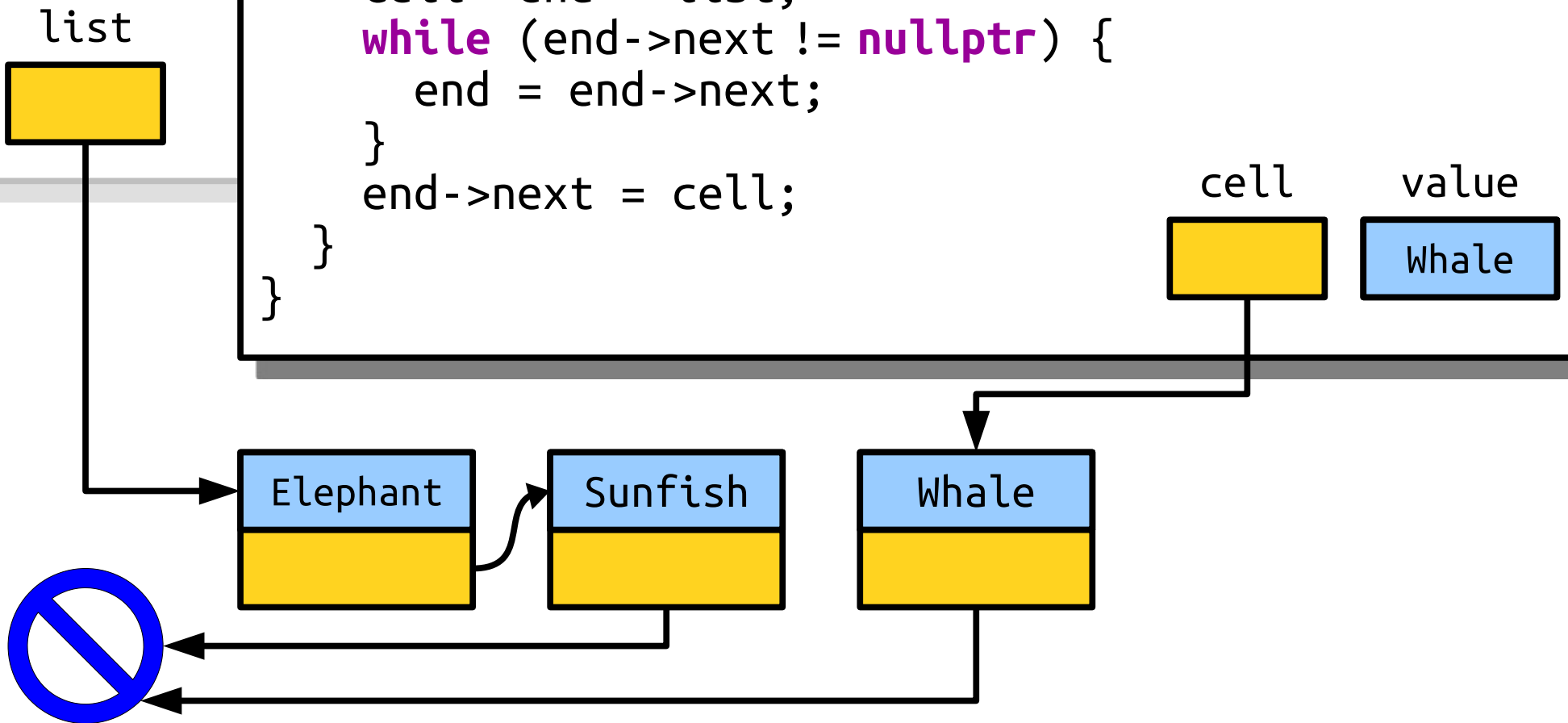
```
int main() {  
    Cell* list;  
    appendTo(list, "Elephant");  
    appendTo(list, "Sunfish");  
    appendTo(list, "Whale");  
    /* ... other code ...  
}
```

```
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        Cell* end = list;  
        while (end->next != nullptr) {  
            end = end->next;  
        }  
        end->next = cell;  
    }  
}
```




```
int main() {  
    Cell* list = nullptr;  
    appendT("Elephant");  
    appendT("Sunfish");  
    appendT("Whale");  
    /* ... other code ... */  
}
```

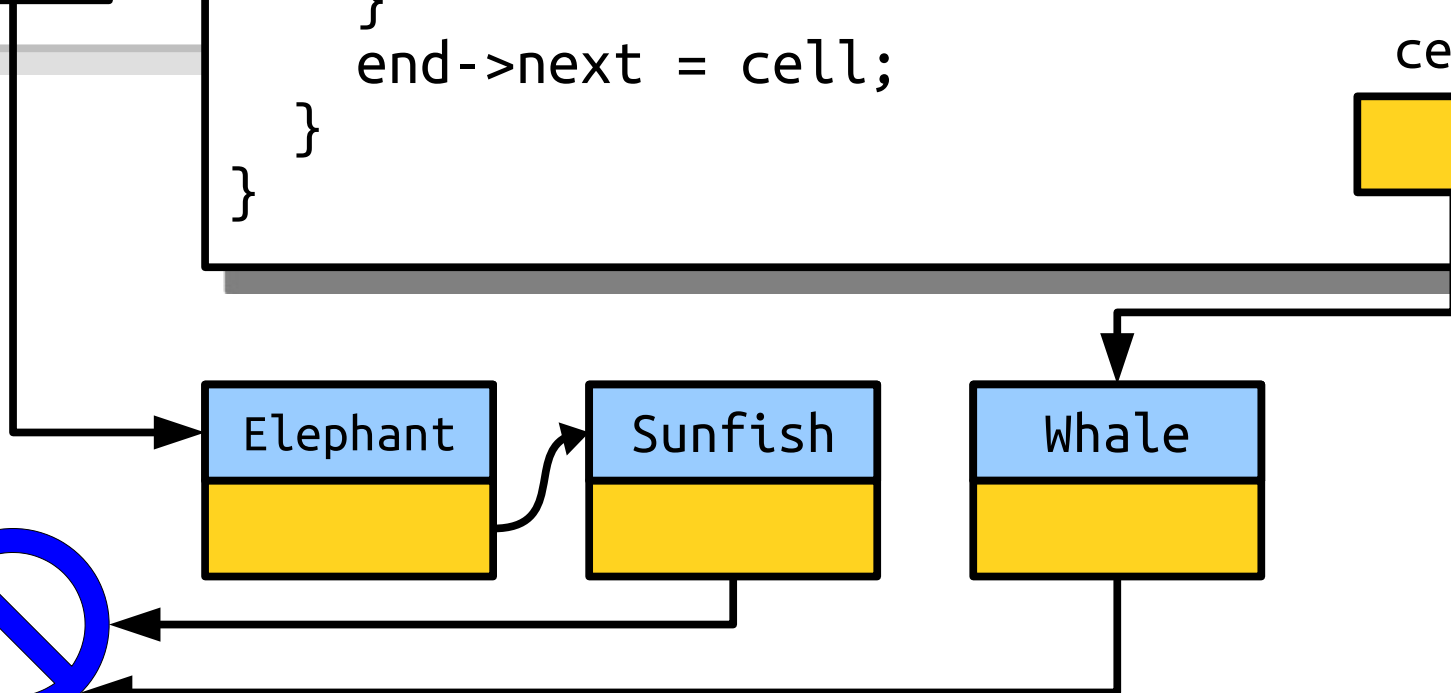
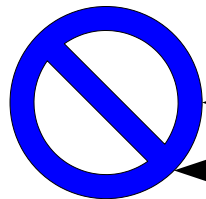
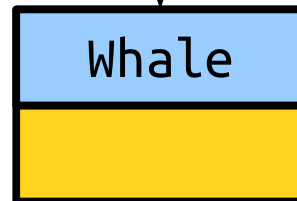
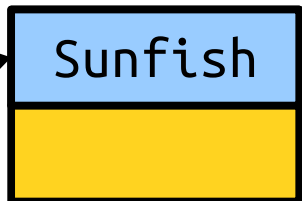
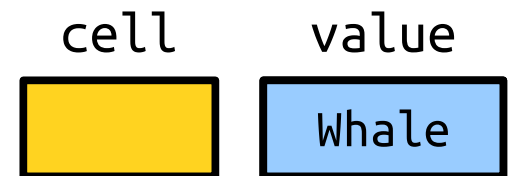
```
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        Cell* end = list;  
        while (end->next != nullptr) {  
            end = end->next;  
        }  
        end->next = cell;  
    }  
}
```



```
int main() {
    Cell* list;
    appendT
    appendT
    appendT
    appendT
    appendT
    /* ... ot
}
```

```
void appendTo(Cell*& list, const string& value) {
    Cell* cell = new Cell;
    cell->value = value;
    cell->next = nullptr;

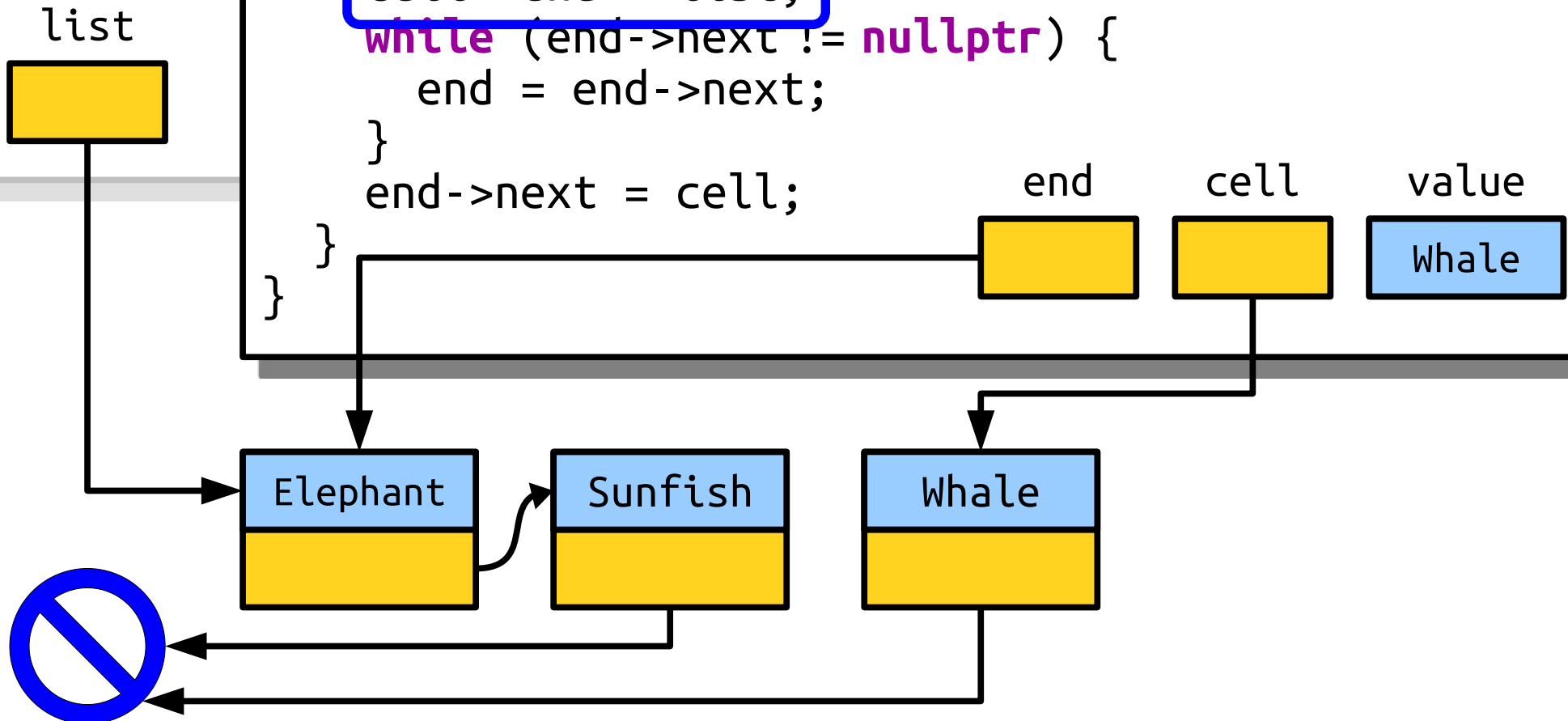
    if (list == nullptr) {
        list = cell;
    } else {
        Cell* end = list;
        while (end->next != nullptr) {
            end = end->next;
        }
        end->next = cell;
    }
}
```




```
int main() {
    Cell* list;
    appendT
    appendT
    appendT
    appendT
    appendT
    /* ... ot
}
```

```
void appendTo(Cell*& list, const string& value) {
    Cell* cell = new Cell;
    cell->value = value;
    cell->next = nullptr;

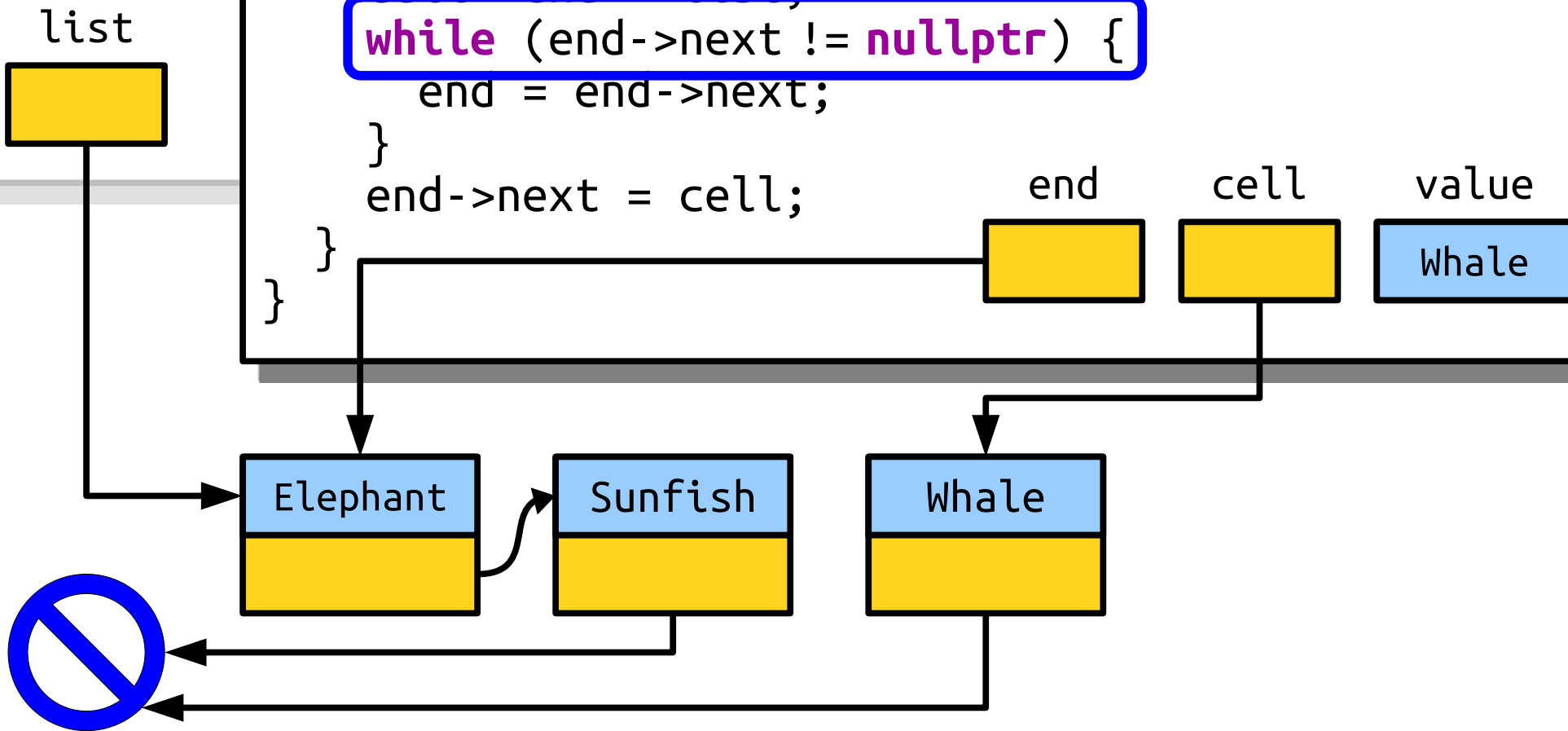
    if (list == nullptr) {
        list = cell;
    } else {
        Cell* end = list;
        while (end->next != nullptr) {
            end = end->next;
        }
        end->next = cell;
    }
}
```



```
int main() {
    Cell* list;
    appendT
    appendT
    appendT
    appendT
    appendT
    /* ... ot
}
```

```
void appendTo(Cell*& list, const string& value) {
    Cell* cell = new Cell;
    cell->value = value;
    cell->next = nullptr;

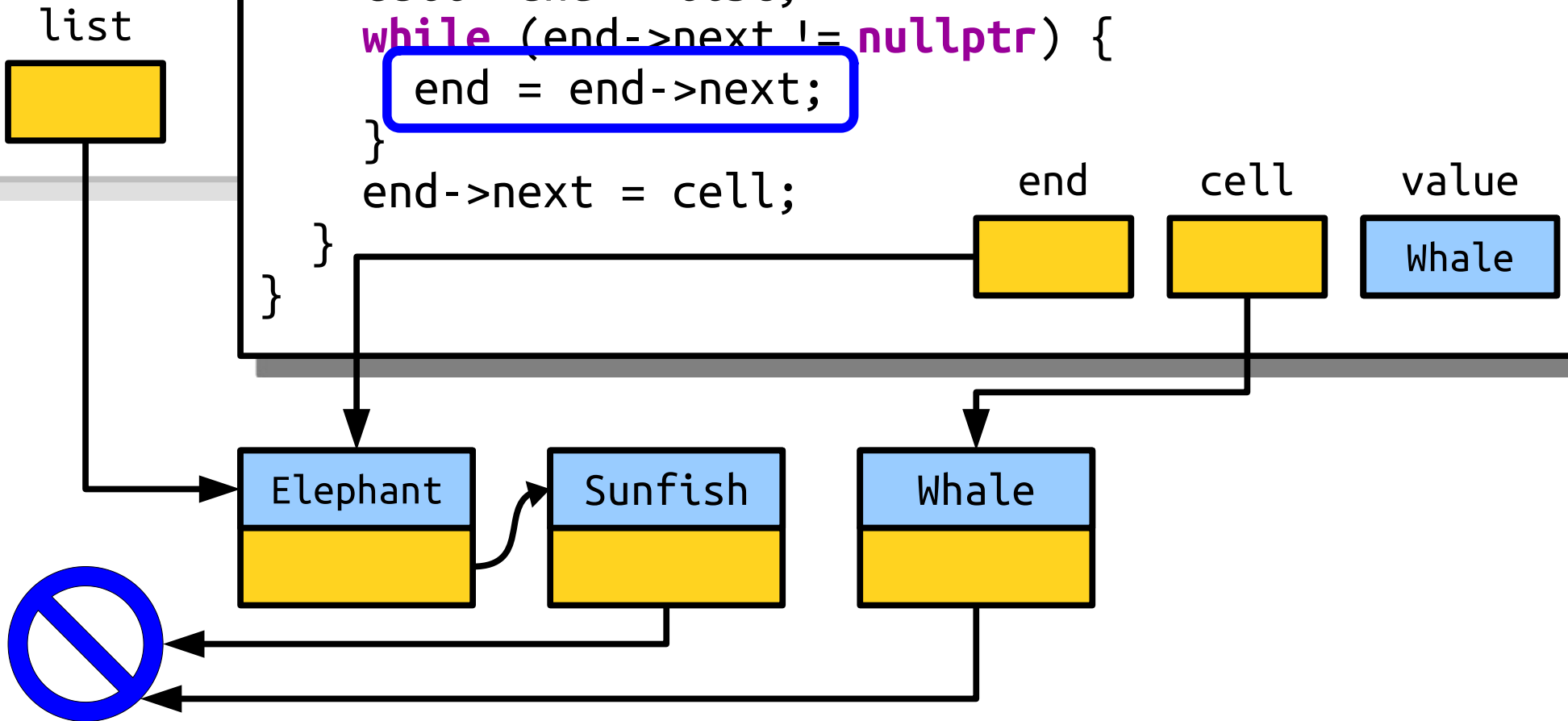
    if (list == nullptr) {
        list = cell;
    } else {
        Cell* end = list;
        while (end->next != nullptr) {
            end = end->next;
        }
        end->next = cell;
    }
}
```



```
int main() {
    Cell* list;
    appendT
    appendT
    appendT
    appendT
    appendT
    /* ... ot
}
```

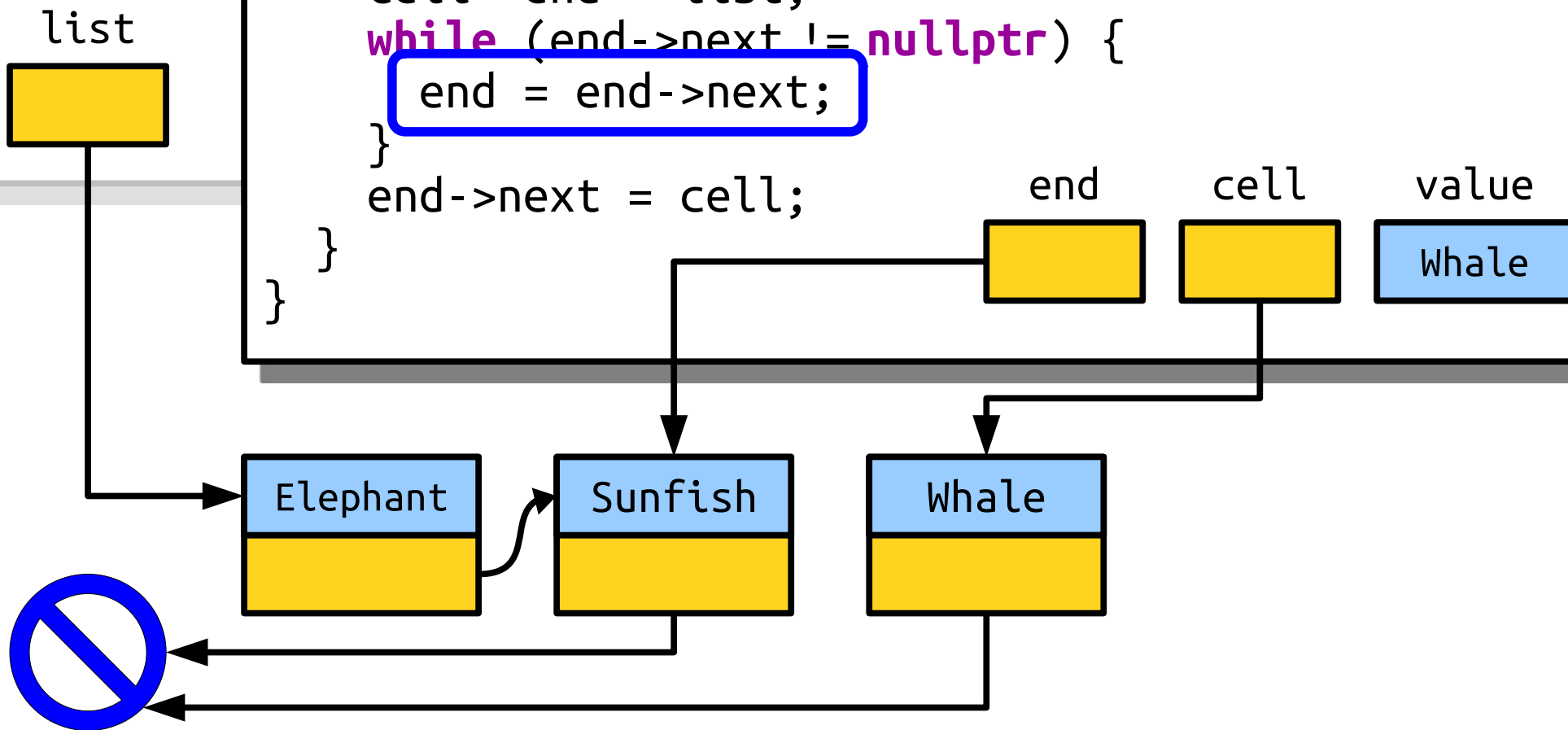
```
void appendTo(Cell*& list, const string& value) {
    Cell* cell = new Cell;
    cell->value = value;
    cell->next = nullptr;

    if (list == nullptr) {
        list = cell;
    } else {
        Cell* end = list;
        while (end->next != nullptr) {
            end = end->next;
        }
        end->next = cell;
    }
}
```



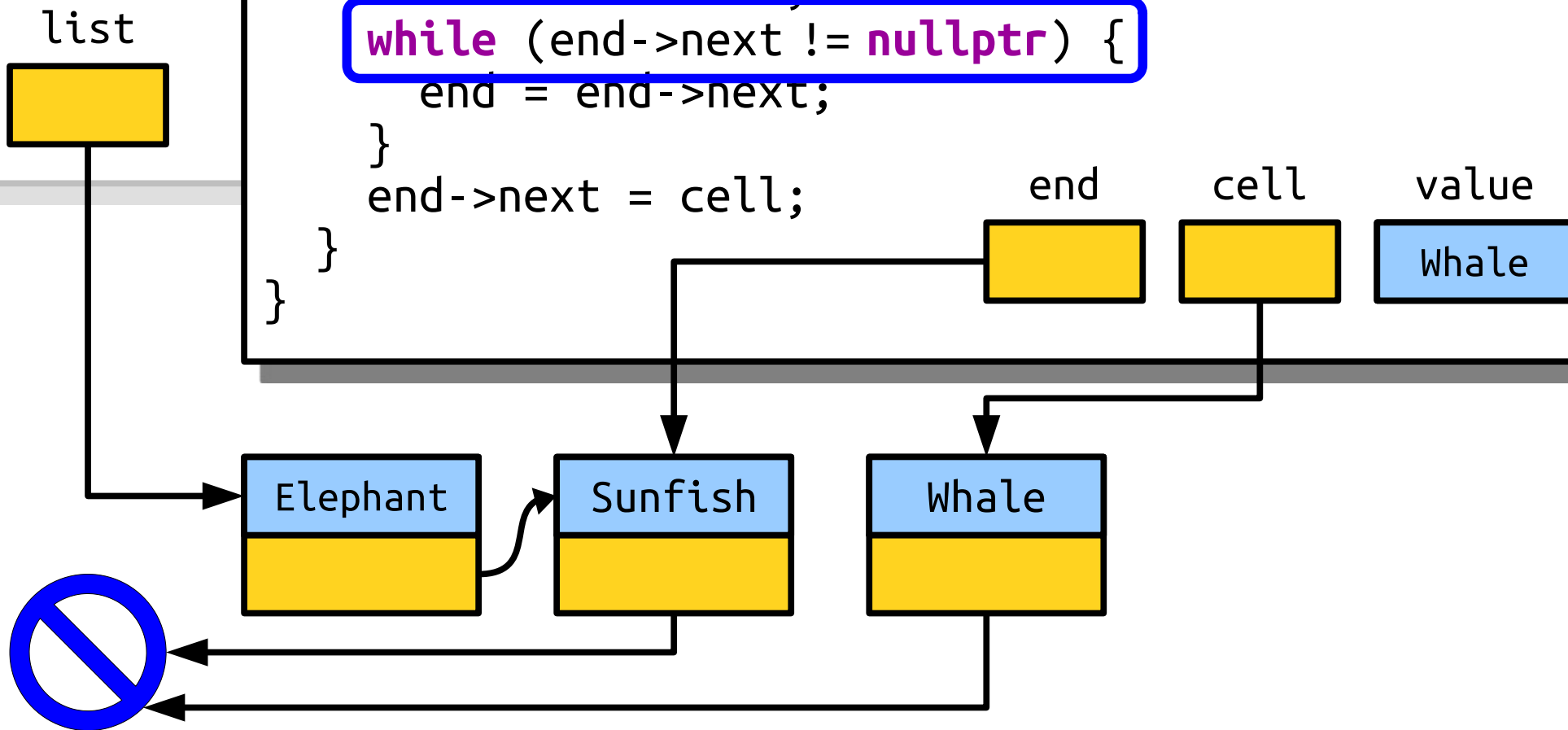
```
int main() {  
    Cell* list;  
    appendT  
    appendT  
    appendT  
    appendT  
    /* ... ot  
}
```

```
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        Cell* end = list;  
        while (end->next != nullptr) {  
            end = end->next;  
        }  
        end->next = cell;  
    }  
}
```



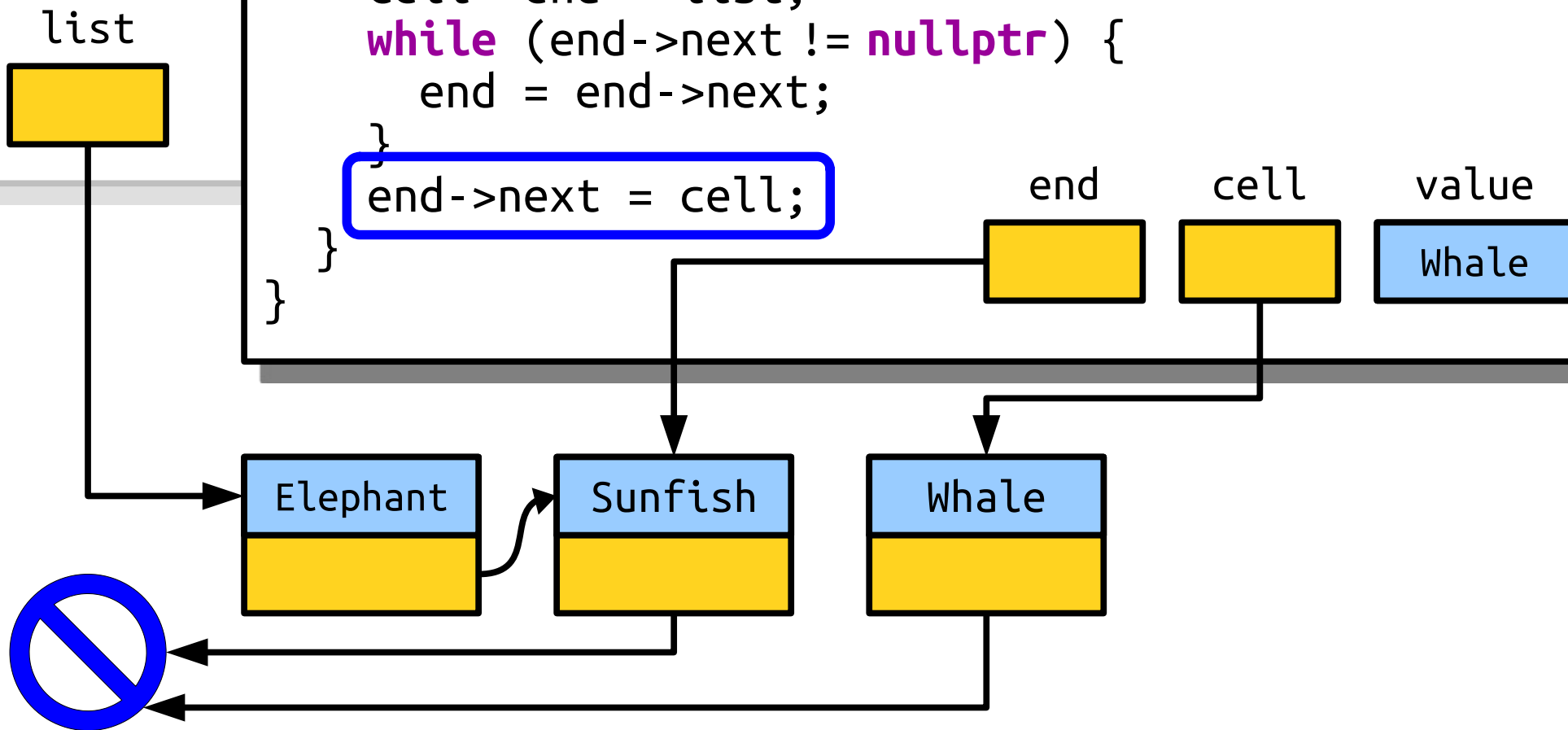
```
int main() {  
    Cell* list;  
    appendT  
    appendT  
    appendT  
    appendT  
    /* ... ot  
}
```

```
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        Cell* end = list;  
        while (end->next != nullptr) {  
            end = end->next;  
        }  
        end->next = cell;  
    }  
}
```



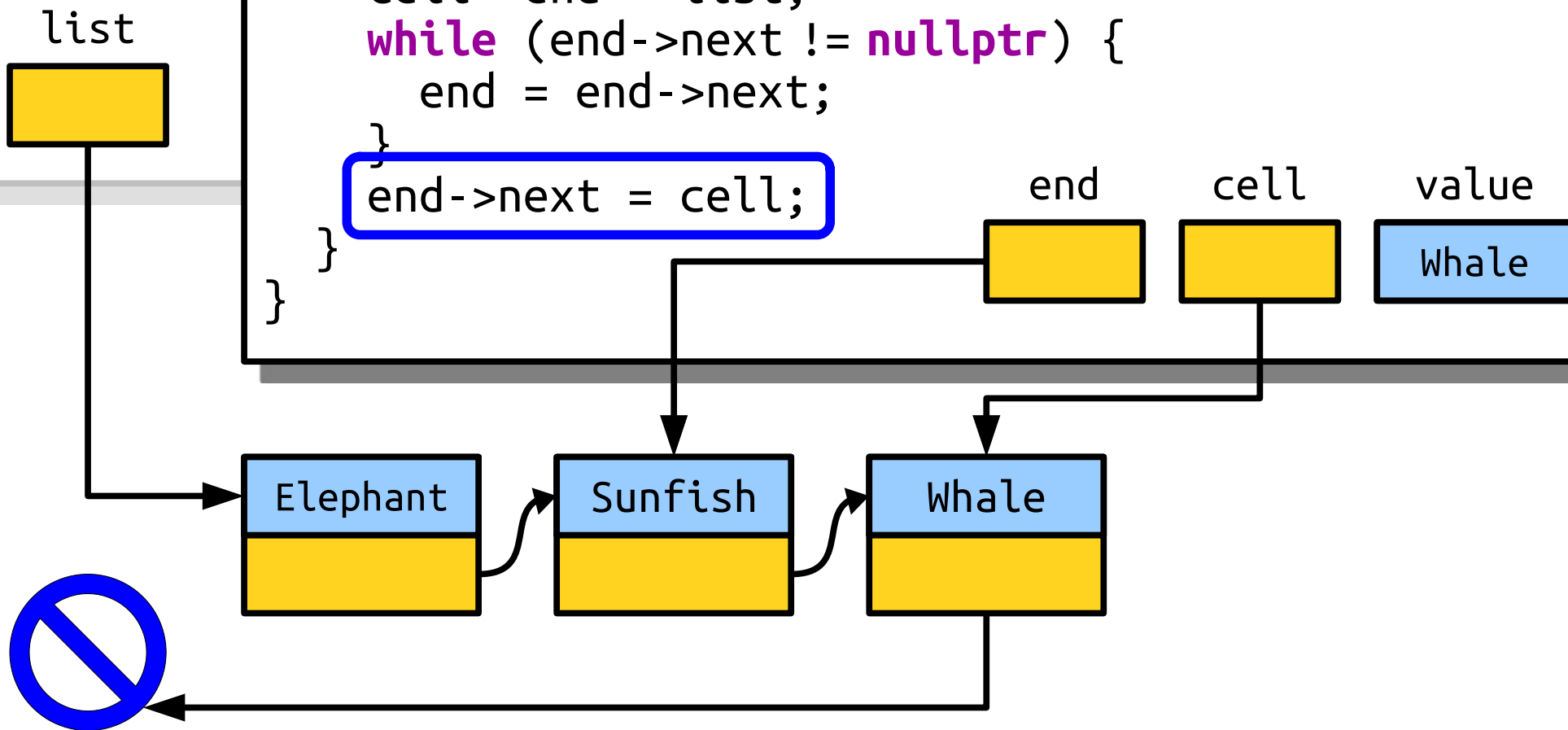
```
int main() {  
    Cell* list = nullptr;  
    appendT  
    appendT  
    appendT  
    appendT  
    /* ... ot  
}
```

```
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        Cell* end = list;  
        while (end->next != nullptr) {  
            end = end->next;  
        }  
        end->next = cell;  
    }  
}
```



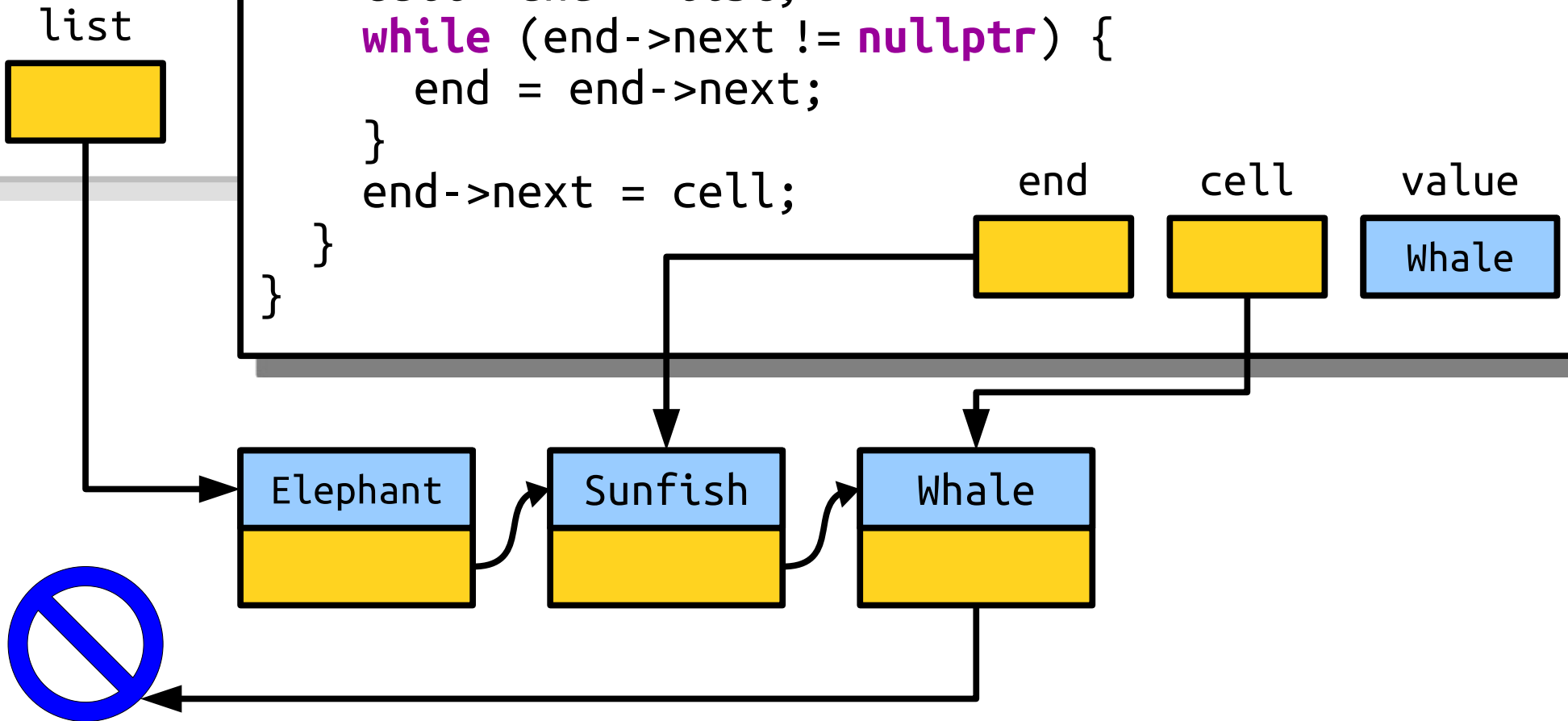
```
int main() {  
    Cell* list;  
    appendT  
    appendT  
    appendT  
    appendT  
    /* ... ot  
}
```

```
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        Cell* end = list;  
        while (end->next != nullptr) {  
            end = end->next;  
        }  
        end->next = cell;  
    }  
}
```



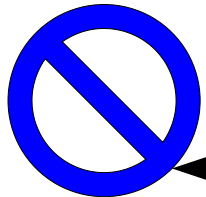
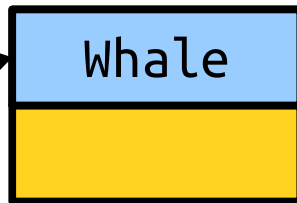
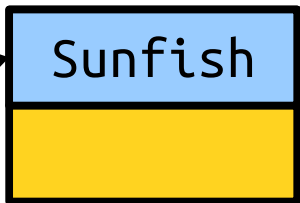
```
int main() {  
    Cell* list = nullptr;  
    appendT  
    appendT  
    appendT  
    appendT  
    /* ... ot  
}
```

```
void appendTo(Cell*& list, const string& value) {  
    Cell* cell = new Cell;  
    cell->value = value;  
    cell->next = nullptr;  
  
    if (list == nullptr) {  
        list = cell;  
    } else {  
        Cell* end = list;  
        while (end->next != nullptr) {  
            end = end->next;  
        }  
        end->next = cell;  
    }  
}
```



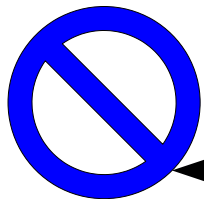
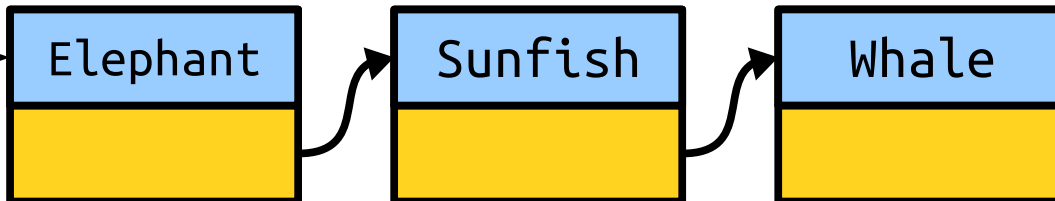

```
int main() {  
    Cell* list = nullptr;  
    appendTo(list, "Elephant");  
    appendTo(list, "Sunfish");  
    appendTo(list, "Whale");  
    appendTo(list, "Piraracu");  
  
    /* ... other listy things. ... */  
}
```

list



```
int main() {  
    Cell* list = nullptr;  
    appendTo(list, "Elephant");  
    appendTo(list, "Sunfish");  
    appendTo(list, "Whale");  
    appendTo(list, "Piraracu");  
  
    /* ... other listy things. ... */  
}
```

list

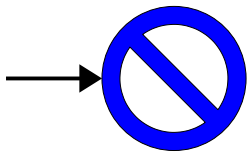


What Went Wrong (Yet Again)?

A Question of Efficiency

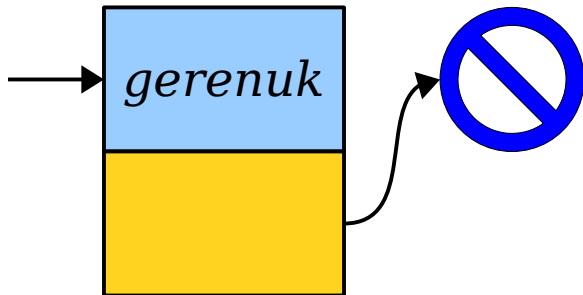
Appending to a List

- What is the big-O complexity of appending to the back of a linked list using our algorithm?
- **Answer: $O(n)$** , where n is the number of elements in the list, since we have to find the last position each time.



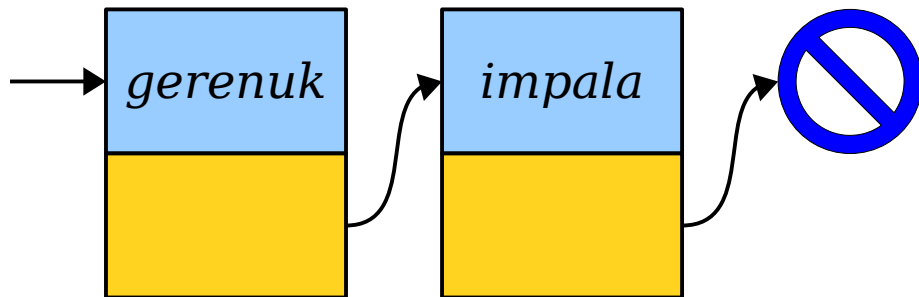
Appending to a List

- What is the big-O complexity of appending to the back of a linked list using our algorithm?
- **Answer: $O(n)$** , where n is the number of elements in the list, since we have to find the last position each time.



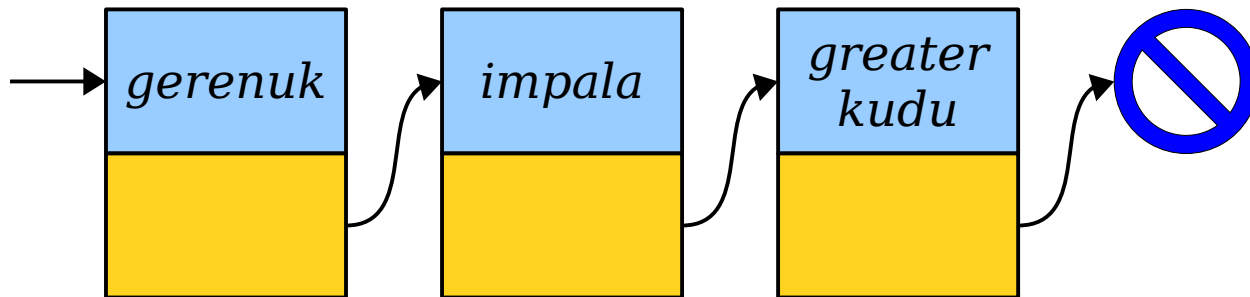
Appending to a List

- What is the big-O complexity of appending to the back of a linked list using our algorithm?
- **Answer: $O(n)$** , where n is the number of elements in the list, since we have to find the last position each time.



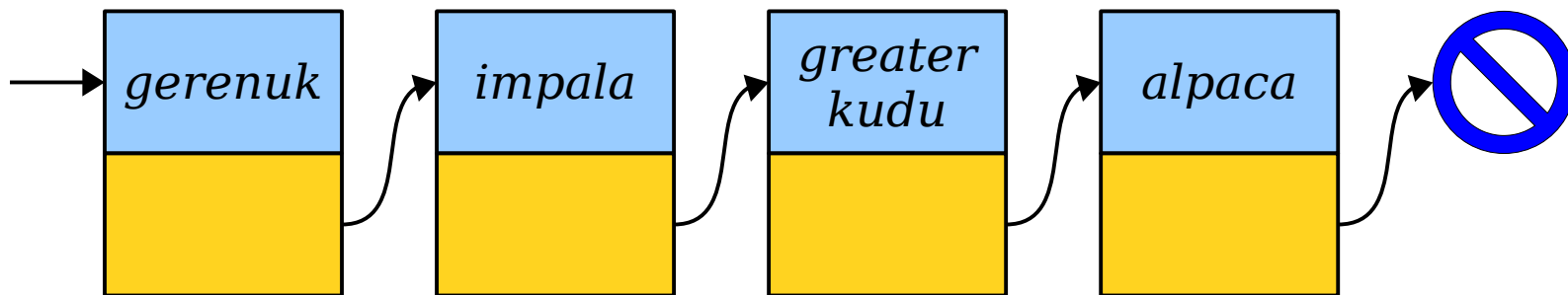
Appending to a List

- What is the big-O complexity of appending to the back of a linked list using our algorithm?
- **Answer: $O(n)$** , where n is the number of elements in the list, since we have to find the last position each time.



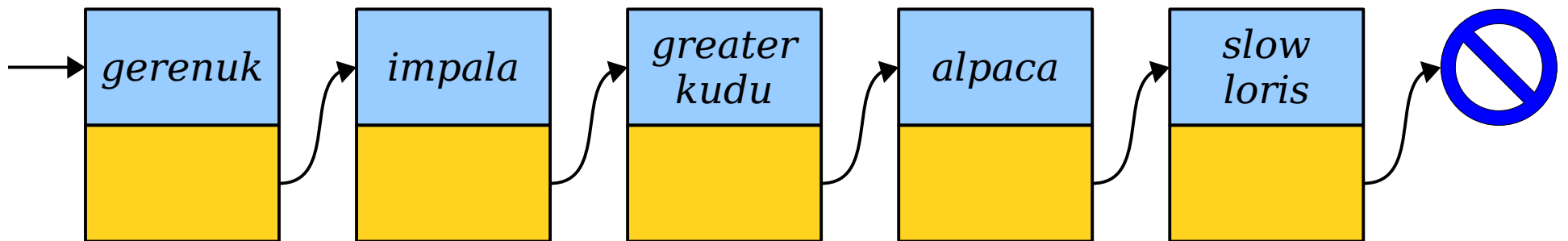
Appending to a List

- What is the big-O complexity of appending to the back of a linked list using our algorithm?
- **Answer: $O(n)$** , where n is the number of elements in the list, since we have to find the last position each time.



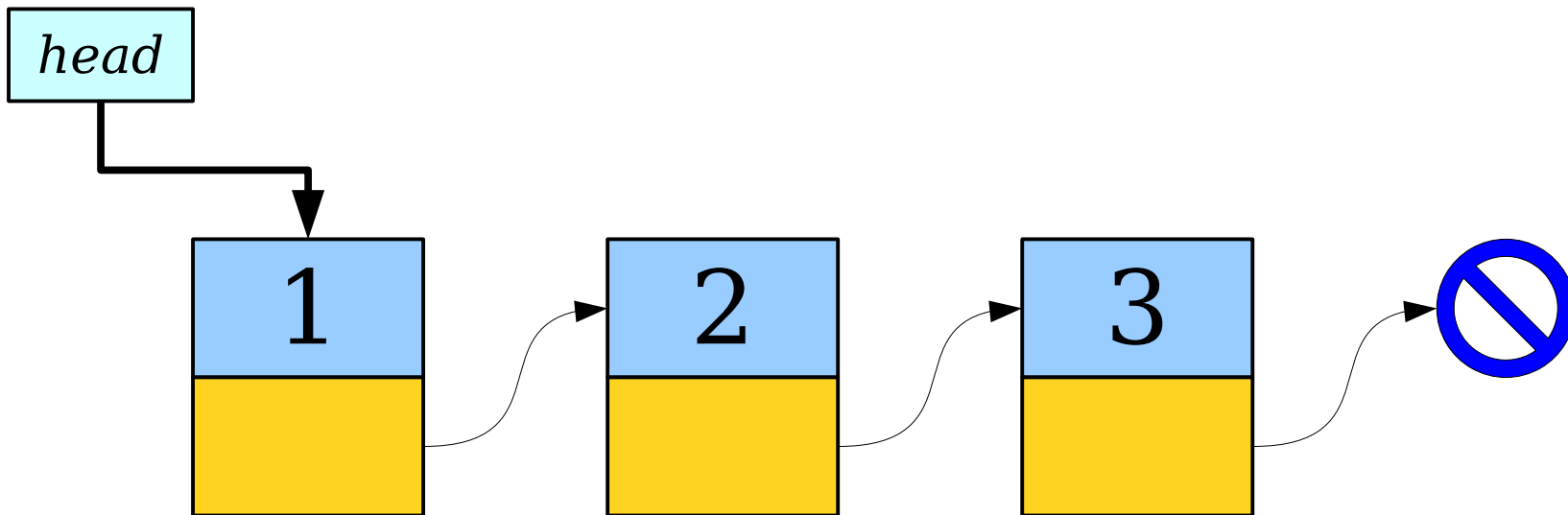
Appending to a List

- What is the big-O complexity of appending to the back of a linked list using our algorithm?
- **Answer: $O(n)$** , where n is the number of elements in the list, since we have to find the last position each time.



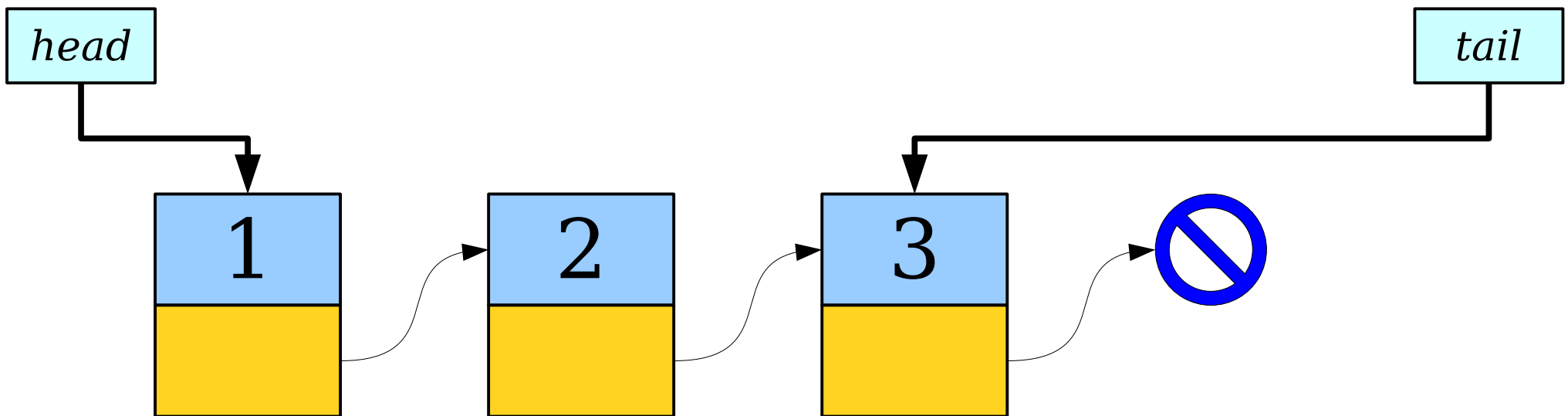
Tail Pointers

- A ***tail pointer*** is a pointer to the last element of a linked list.
- Tail pointers make it easy and efficient to add new elements to the back of a linked list.



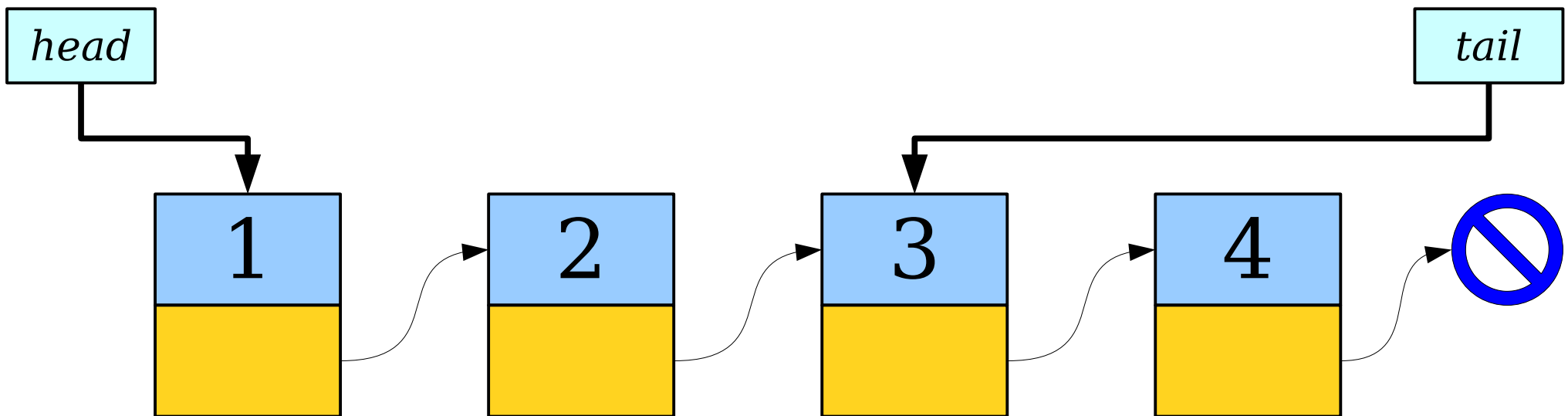
Tail Pointers

- A ***tail pointer*** is a pointer to the last element of a linked list.
- Tail pointers make it easy and efficient to add new elements to the back of a linked list.



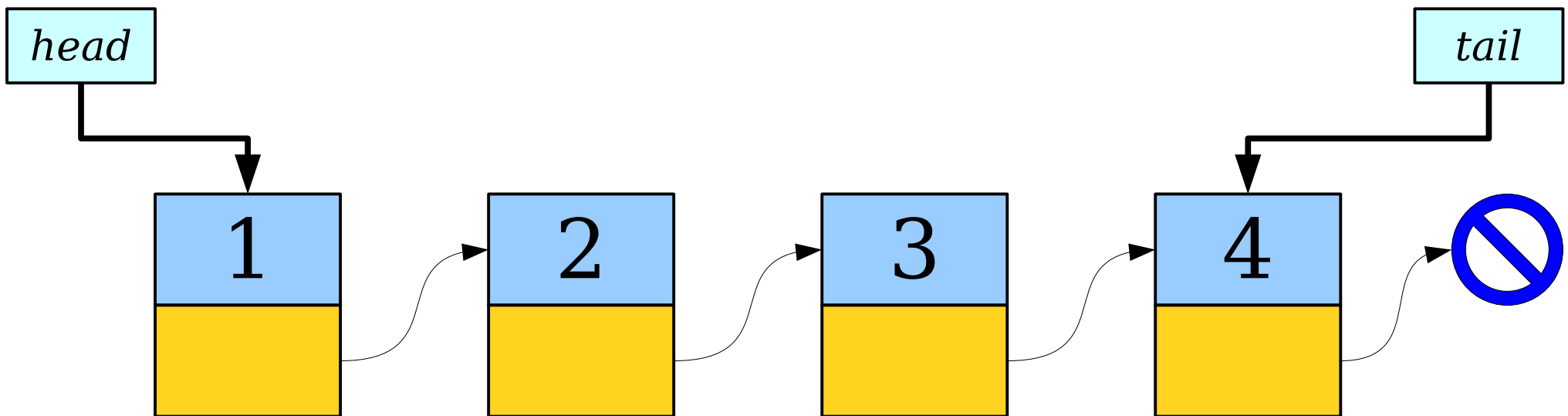
Tail Pointers

- A ***tail pointer*** is a pointer to the last element of a linked list.
- Tail pointers make it easy and efficient to add new elements to the back of a linked list.



Tail Pointers

- A ***tail pointer*** is a pointer to the last element of a linked list.
- Tail pointers make it easy and efficient to add new elements to the back of a linked list.



Appending Things Quickly

- **Case 1:** The list is empty.

head

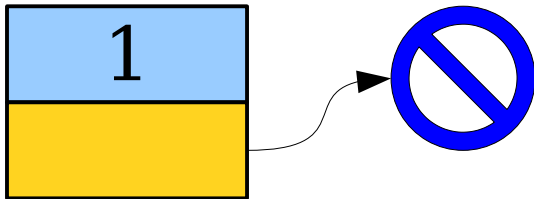
tail

Appending Things Quickly

- **Case 1:** The list is empty.

head

tail



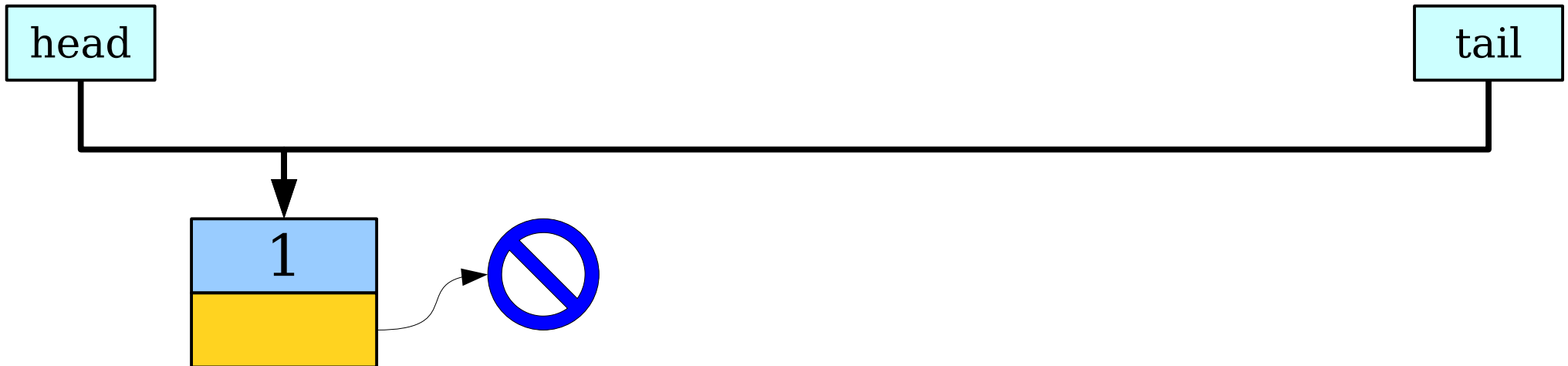
Appending Things Quickly

- **Case 1:** The list is empty.

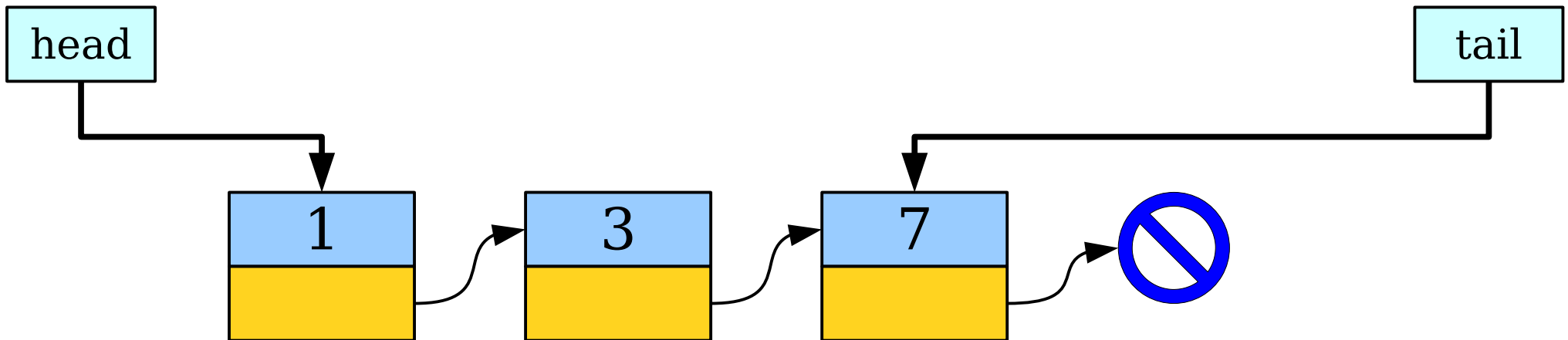


Appending Things Quickly

- **Case 1:** The list is empty.

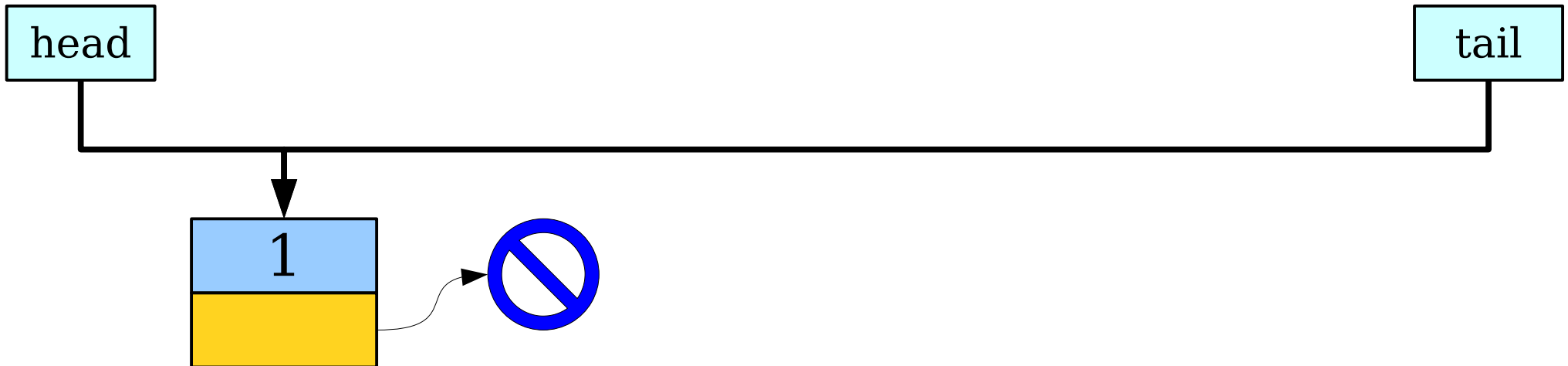


- **Case 2:** The list is not empty.

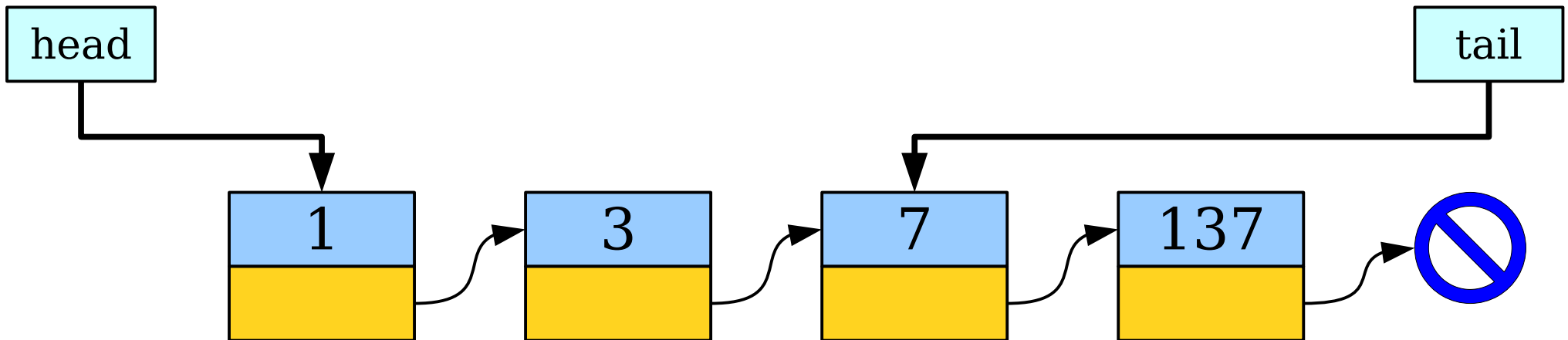


Appending Things Quickly

- **Case 1:** The list is empty.

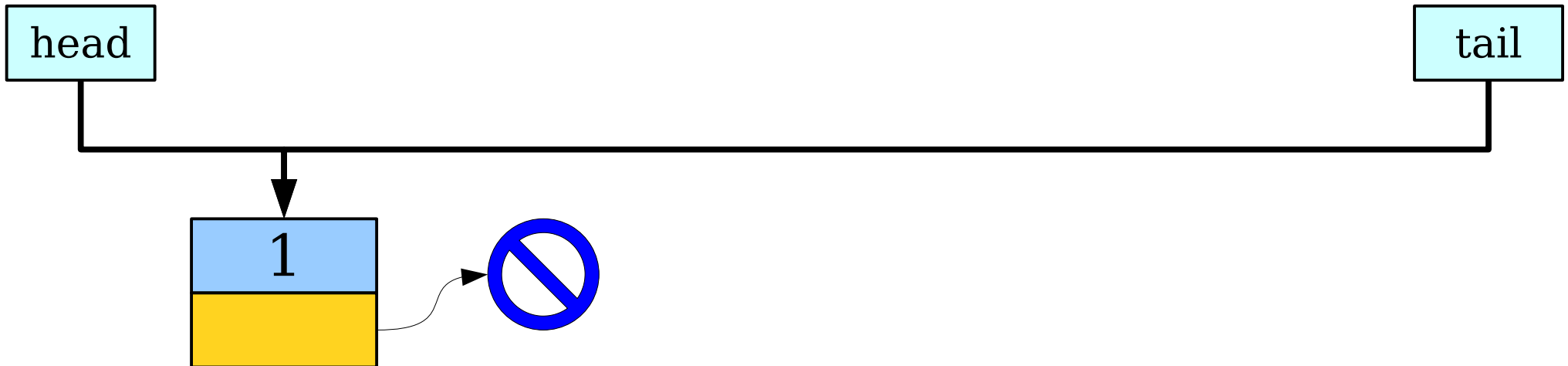


- **Case 2:** The list is not empty.

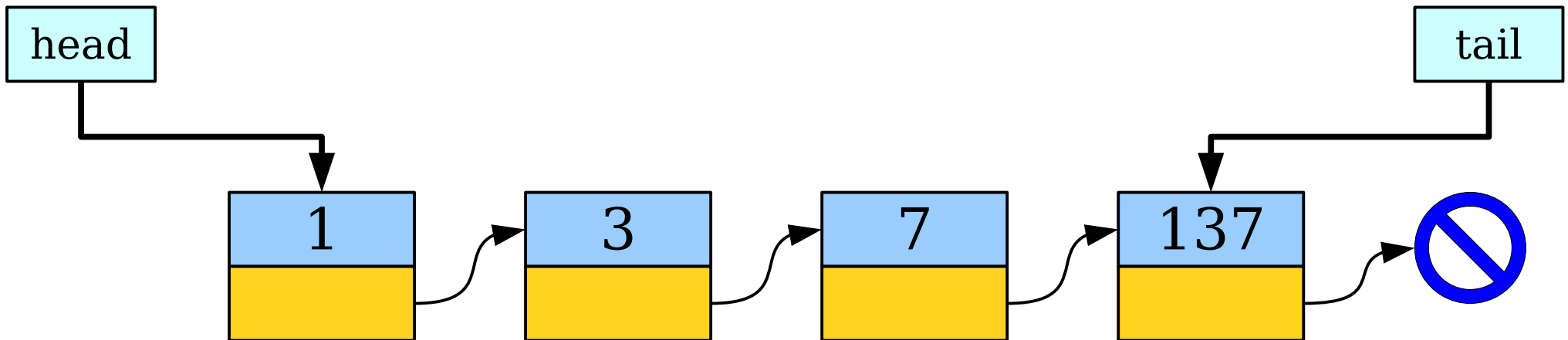


Appending Things Quickly

- **Case 1:** The list is empty.

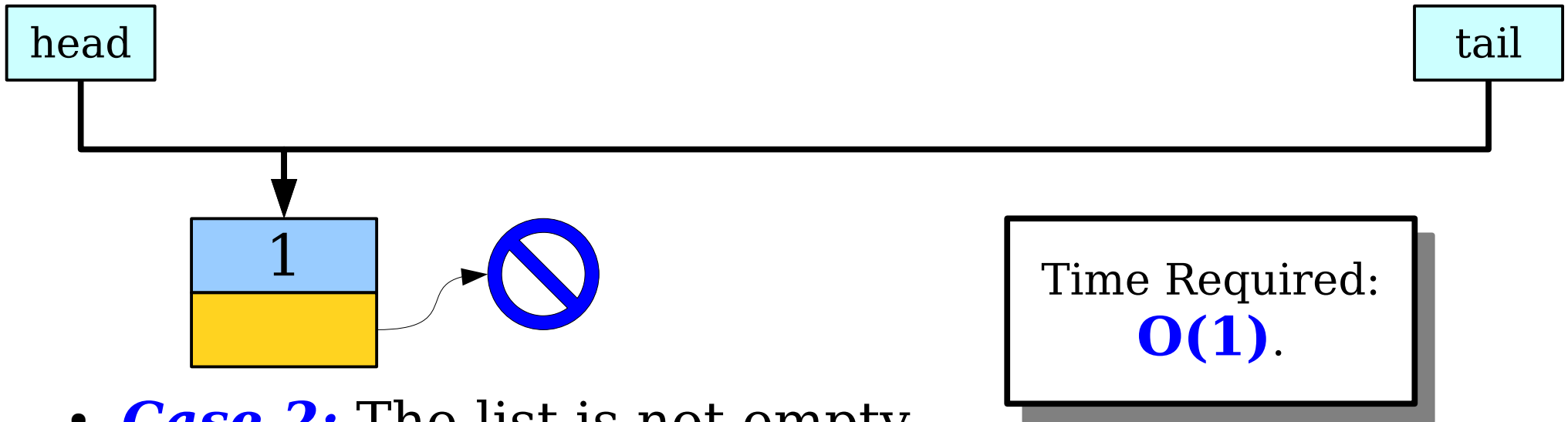


- **Case 2:** The list is not empty.

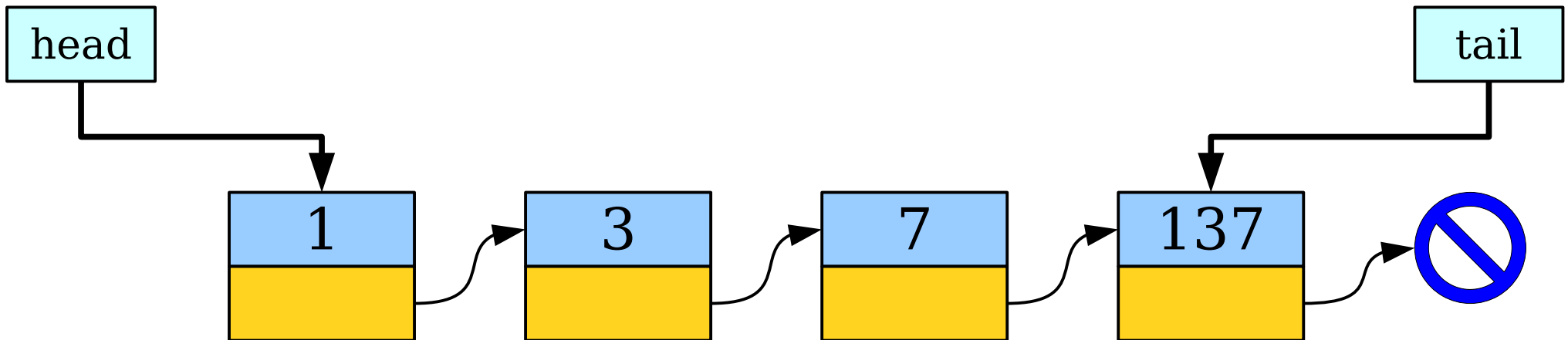


Appending Things Quickly

- **Case 1:** The list is empty.



- **Case 2:** The list is not empty.



Coda: Doubly-Linked Lists

Doubly-Linked Lists

- There's a strange asymmetry in a linked list: you can easily move forward in a list, but there's no easy way to move backwards.
- A ***doubly-linked list*** is a list where each cell stores two pointers: one to the next element in the list, and one to the previous element.



Doubly-Linked Lists

- In many cases, doubly-linked lists are similar to singly-linked lists.
- For example, if you're just moving from the left to the right, then code on doubly-linked lists looks really similar to code on singly-linked lists.



Doubly-Linked Lists

- In many cases, doubly-linked lists are similar to singly-linked lists.
- For example, if you're just moving from the left to the right, then code on doubly-linked lists looks really similar to code on singly-linked lists.

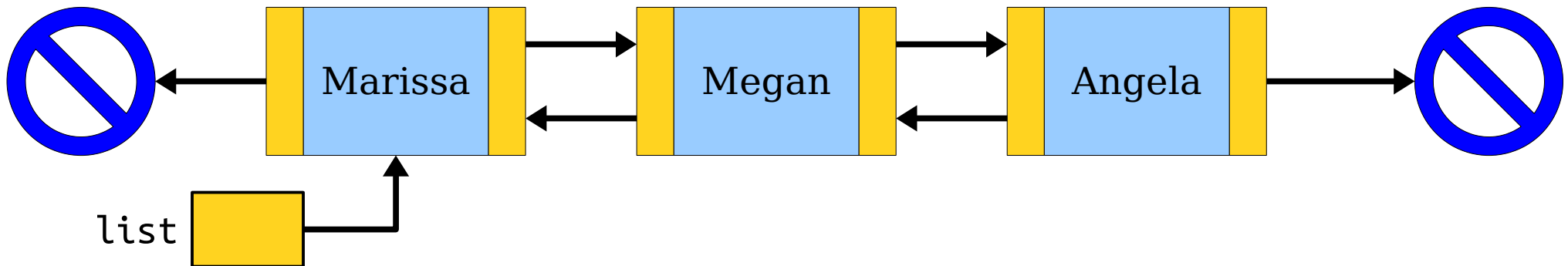
```
Cell* list = /* first cell */;
```



Doubly-Linked Lists

- In many cases, doubly-linked lists are similar to singly-linked lists.
- For example, if you're just moving from the left to the right, then code on doubly-linked lists looks really similar to code on singly-linked lists.

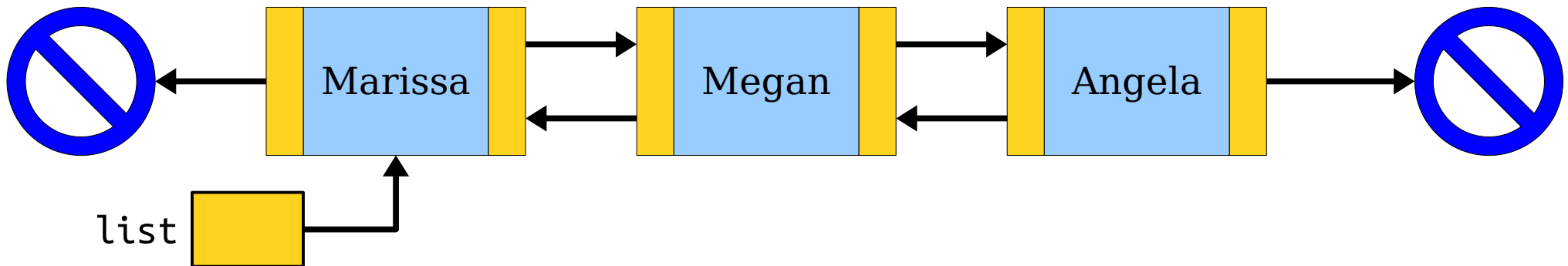
```
Cell* list = /* first cell */;
```



Doubly-Linked Lists

- In many cases, doubly-linked lists are similar to singly-linked lists.
- For example, if you're just moving from the left to the right, then code on doubly-linked lists looks really similar to code on singly-linked lists.

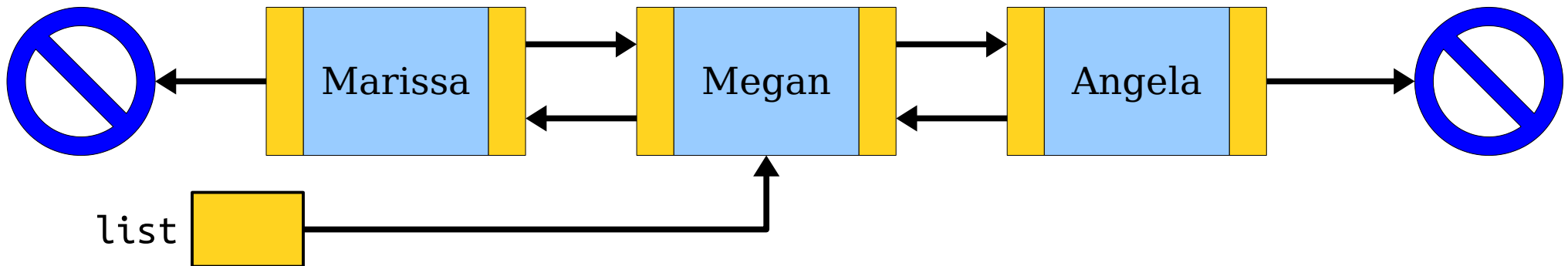
```
Cell* list = /* first cell */;  
list = list->next;
```



Doubly-Linked Lists

- In many cases, doubly-linked lists are similar to singly-linked lists.
- For example, if you're just moving from the left to the right, then code on doubly-linked lists looks really similar to code on singly-linked lists.

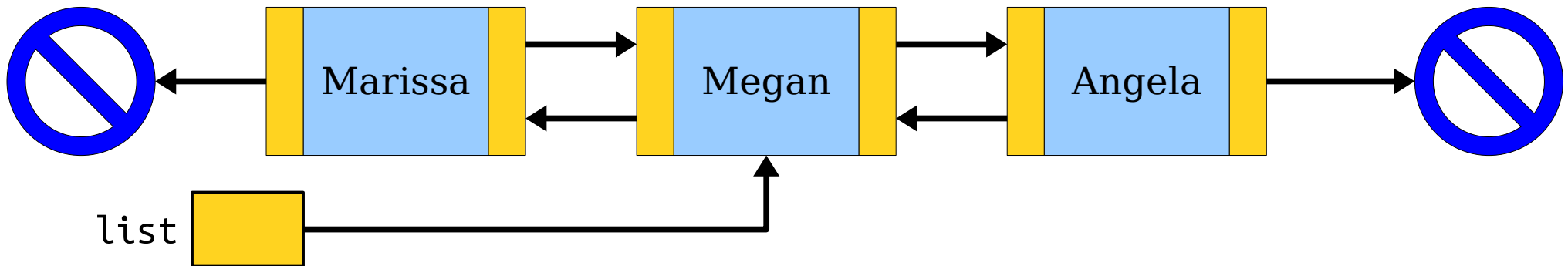
```
Cell* list = /* first cell */;  
list = list->next;
```



Doubly-Linked Lists

- We can also move backwards in a doubly-linked list.
- Many algorithms are a lot easier to write if you can do this!

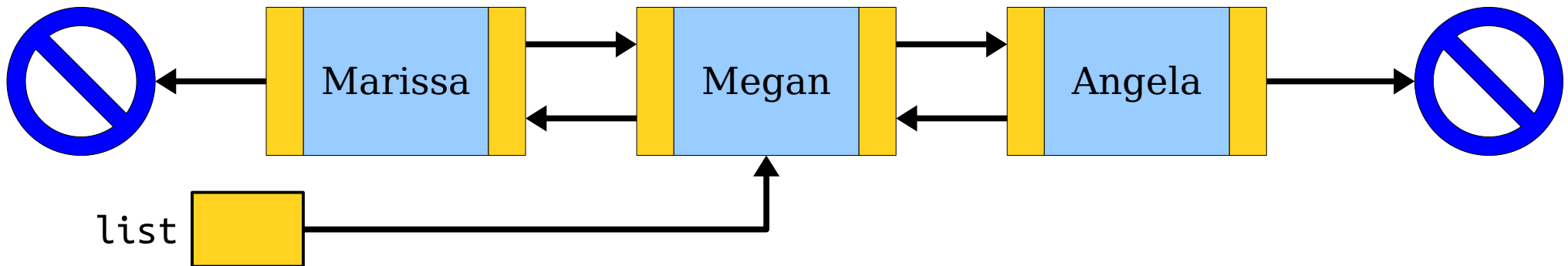
```
Cell* list = /* first cell */;  
list = list->next;
```



Doubly-Linked Lists

- We can also move backwards in a doubly-linked list.
- Many algorithms are a lot easier to write if you can do this!

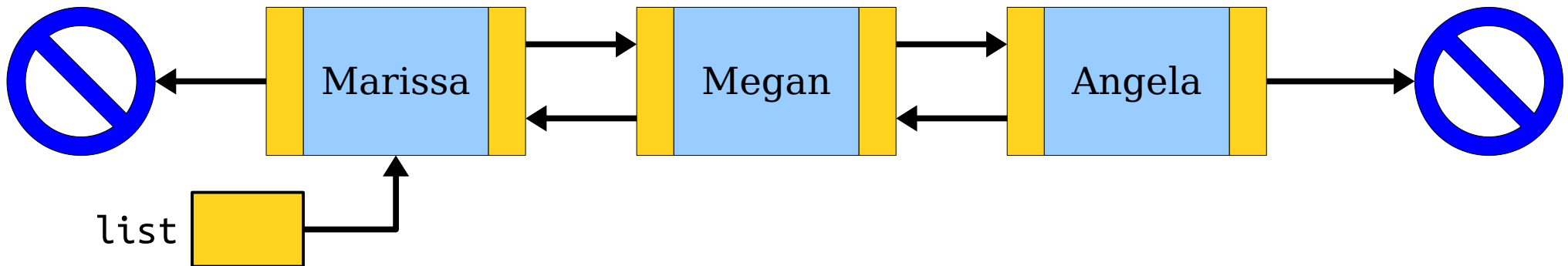
```
Cell* list = /* first cell */;  
list = list->next;  
list = list->prev;
```



Doubly-Linked Lists

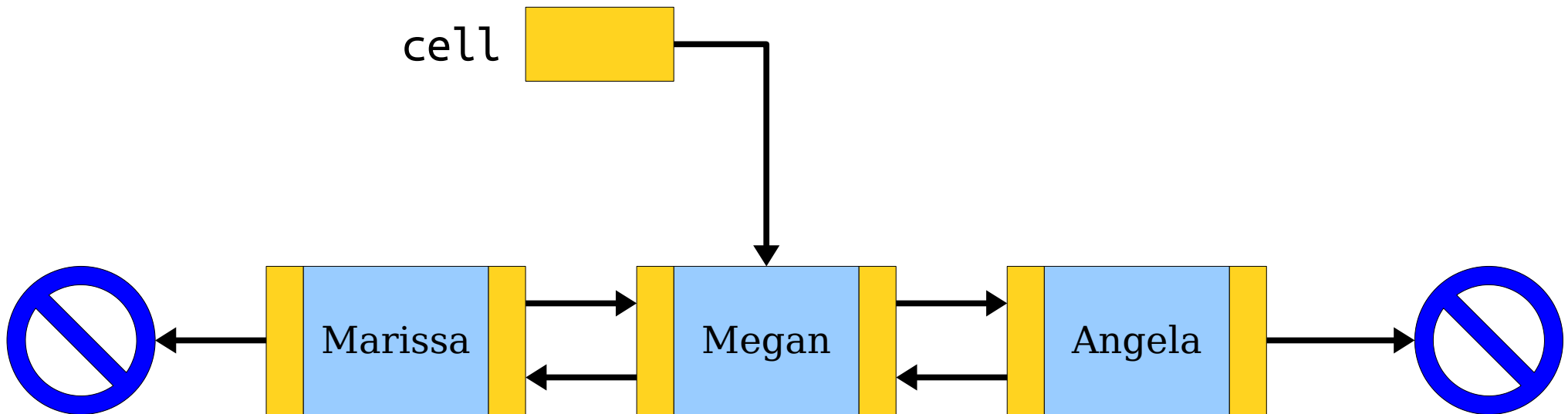
- We can also move backwards in a doubly-linked list.
- Many algorithms are a lot easier to write if you can do this!

```
Cell* list = /* first cell */;  
list = list->next;  
list = list->prev;
```



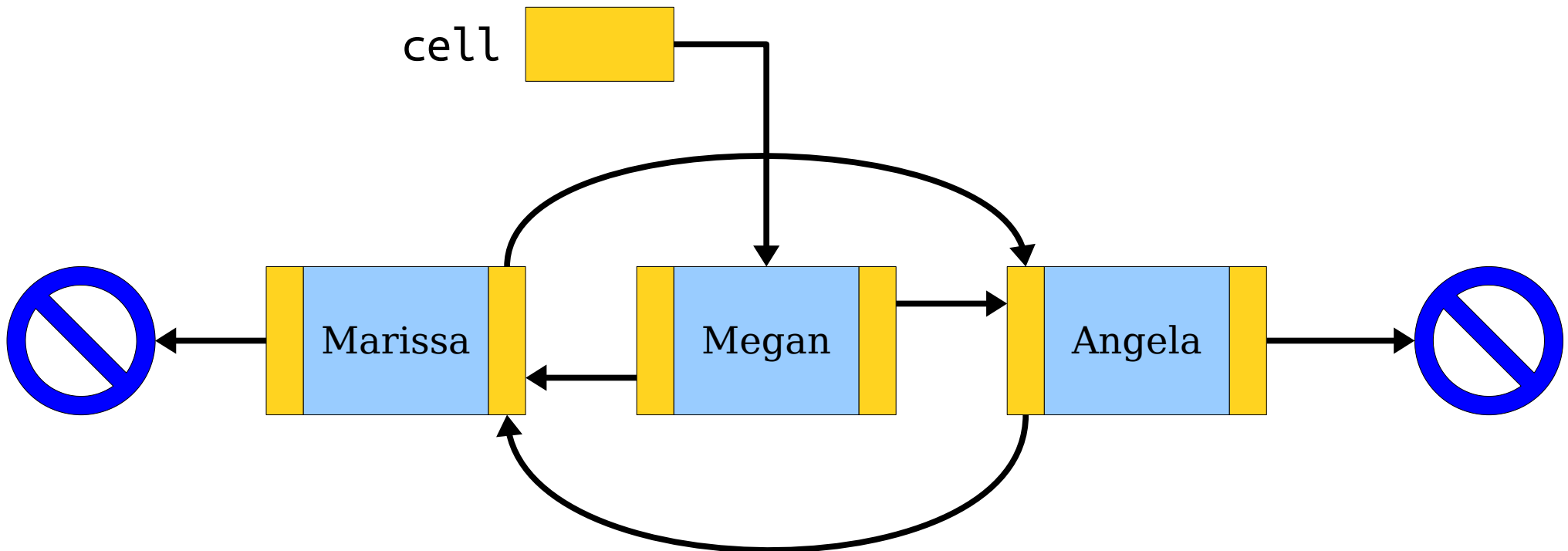
Doubly-Linked Lists

- It's easy to remove a cell from a doubly-linked list: just wire the nodes next to it around it.
- (Don't forget to handle edge cases!)



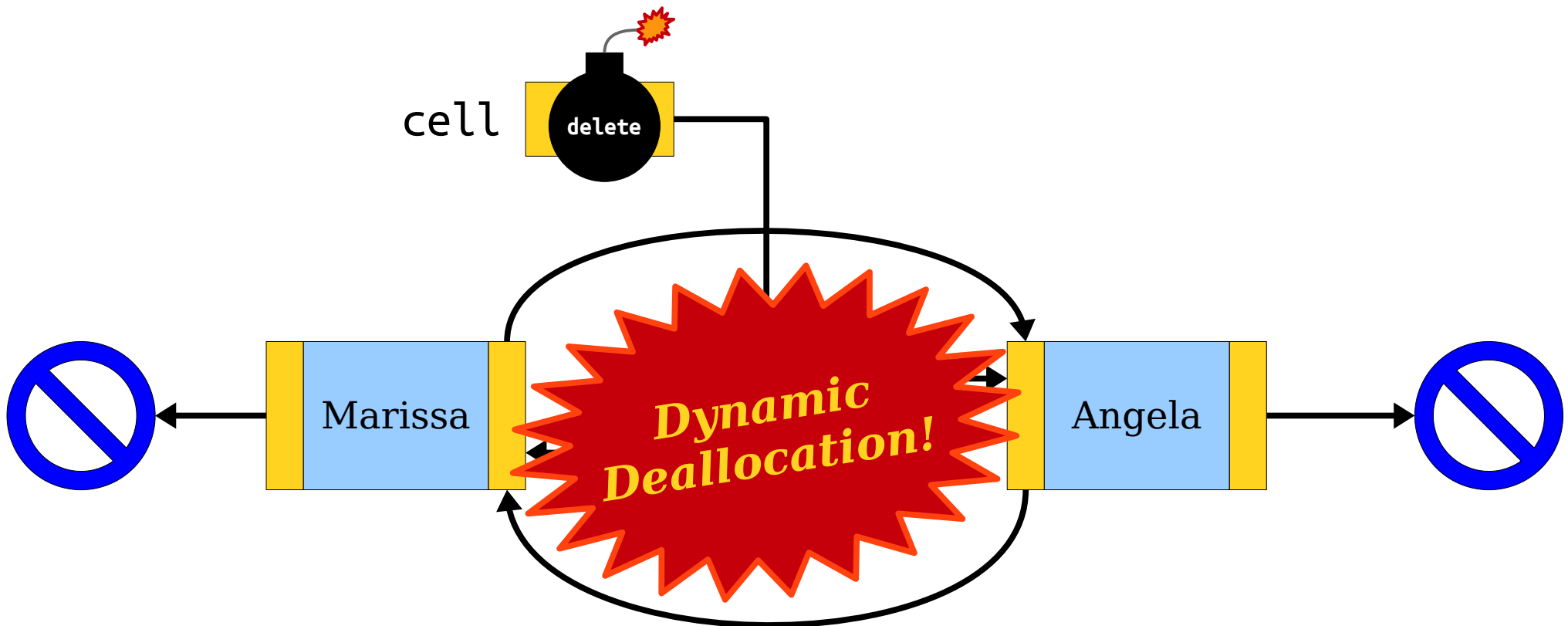
Doubly-Linked Lists

- It's easy to remove a cell from a doubly-linked list: just wire the nodes next to it around it.
- (Don't forget to handle edge cases!)



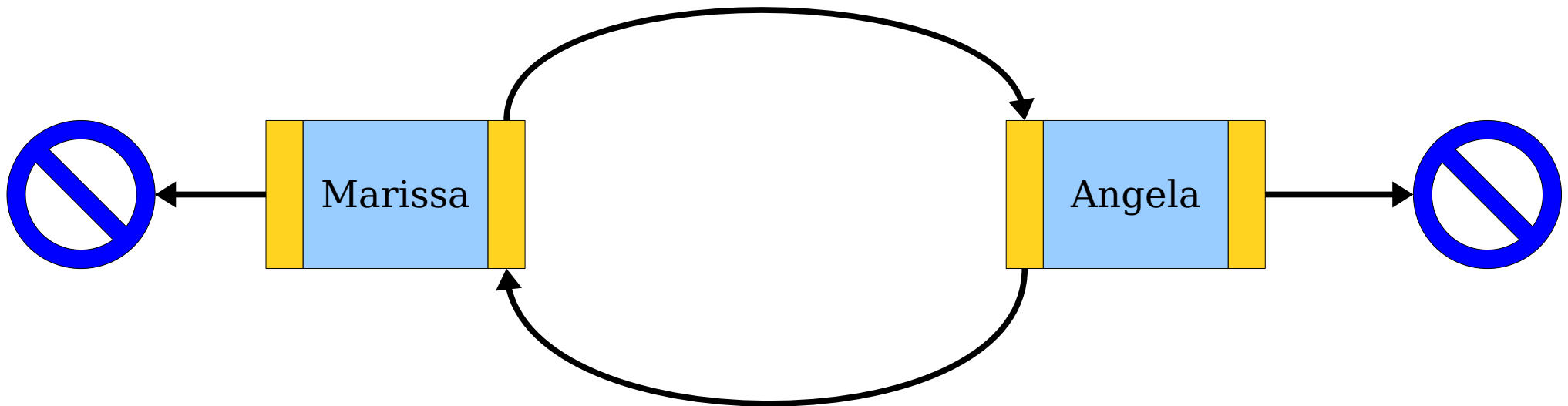
Doubly-Linked Lists

- It's easy to remove a cell from a doubly-linked list: just wire the nodes next to it around it.
- (Don't forget to handle edge cases!)



Doubly-Linked Lists

- It's easy to remove a cell from a doubly-linked list: just wire the nodes next to it around it.
- (Don't forget to handle edge cases!)



For more on doubly-linked lists, check ***Section Problems 7*** and ***Chapter 13*** of the textbook.

To Recap

- If you want a function to change *which object* a pointer points to, pass that pointer in by reference.
- When passing pointers by reference, don't change the pointer unless you really mean it.
- Tail pointers make it easy to find the end of a linked list - a handy tool to keep in mind!
- Doubly-linked lists have each cell store pointers to both the next and previous cells in the list. They're useful for when you need to remove out of a list.

Your Action Items

- ***Read Chapter 13.***
 - It's all about different representations for data and the relative tradeoffs. And there's some great coverage of linked lists in there!
- ***Finish Assignment 6.***
 - If you're following our suggested timeline, you'll have completed your implementation of Linear Probing by today.
 - Remember to leave appropriate buffer time for the performance analysis section!

Next Time

- ***Tree Structures***
 - Representing branching structures in code.
- ***Binary Search Trees***
 - Maintaining order at a low cost!