Linked Lists

Part One

Outline for Today

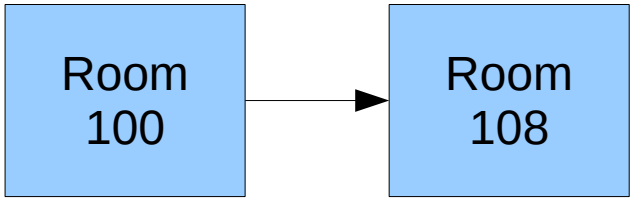
- ***Linked Lists, Conceptually***
  - A different way to represent a sequence.
- ***Linked Lists, In Code***
  - Some cool new C++ tricks.

Changing Offices

The Sign on Room 100

Dr. Cynthia Lee is no longer in room 100.

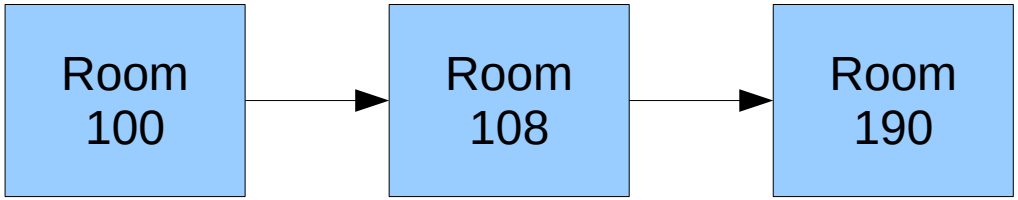
She can be found in room 108.



The Sign on Room 108

Dr. Cynthia Lee is no longer in room 108.

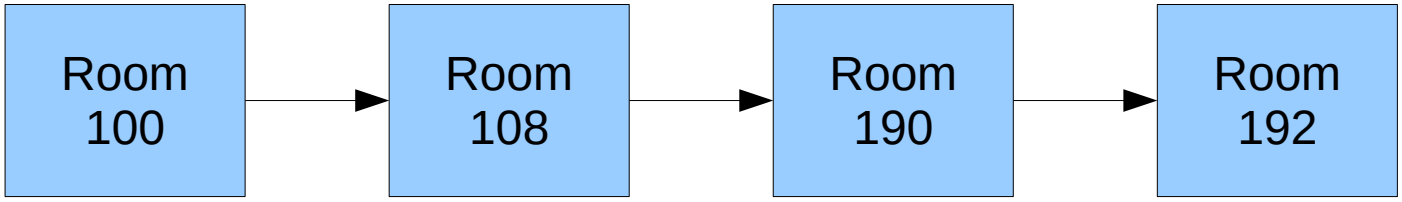
She can be found in room 190.



The Sign on Room 190

Dr. Cynthia Lee is no longer in room 190.

She can be found in room 192.



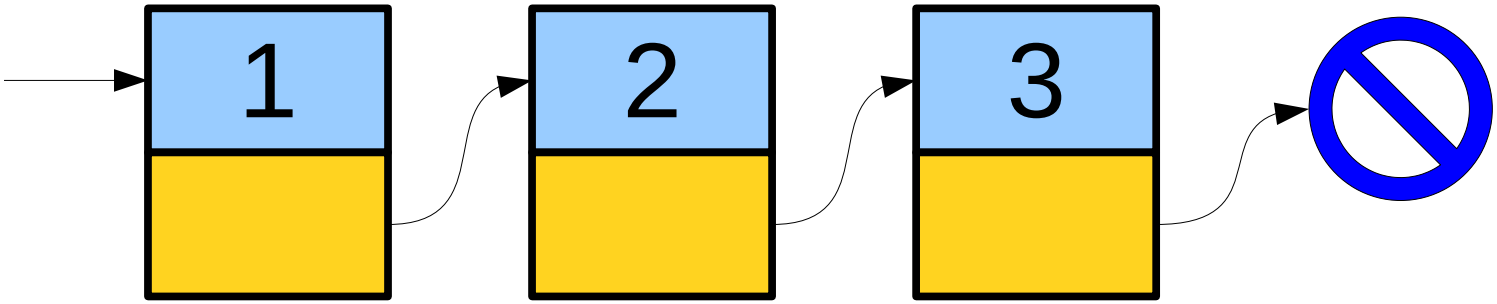
The Sign on Room 192

Welcome to Cynthia's Office!

Room 100 → Room 108 → Room 190 → Room 192 →

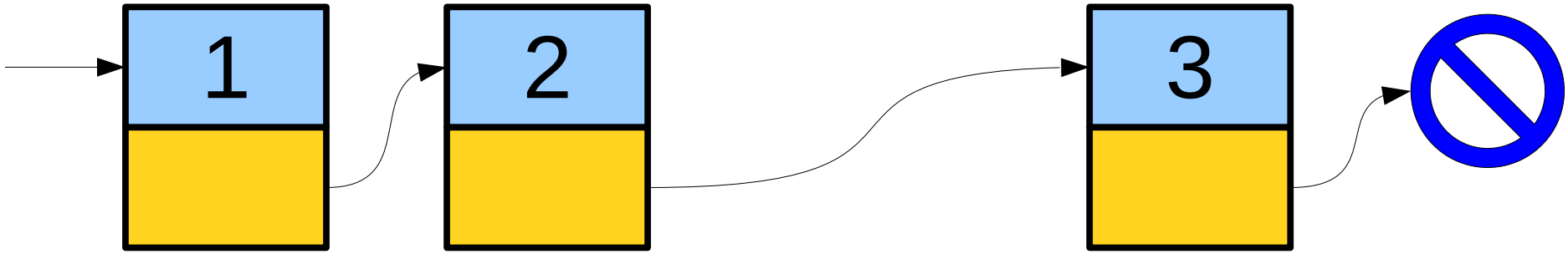
Linked Lists at a Glance

- 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.



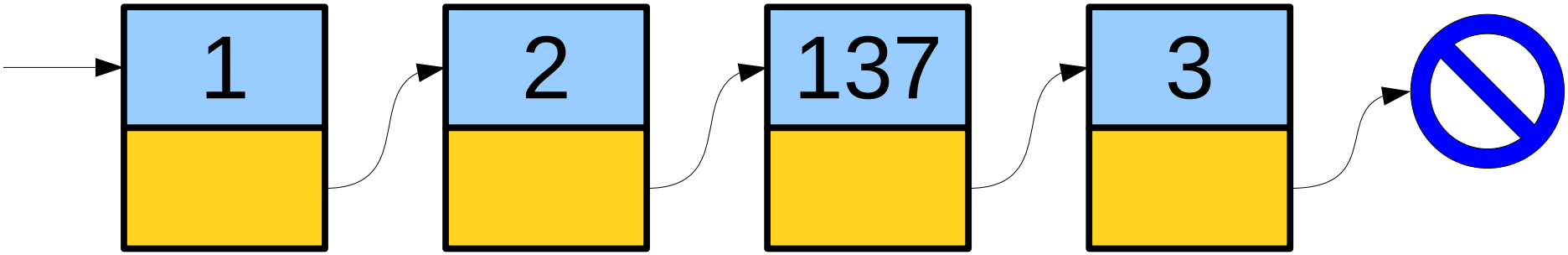
Linked Lists at a Glance

- 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.



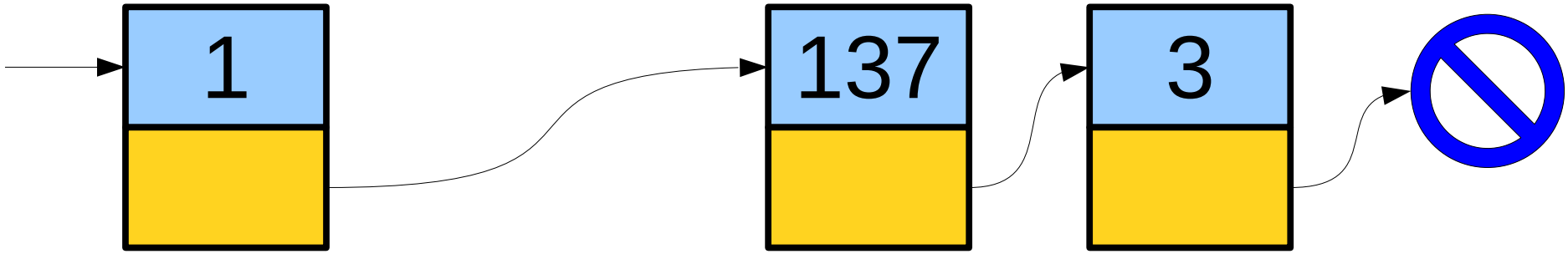
Linked Lists at a Glance

- 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.



Linked Lists at a Glance

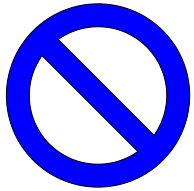
- A ***linked list*** is a data structure for storing a sequence of elements.
- Each element is stored separately from the rest.
- The elements are then chained together into a sequence.
- The end of the list is marked with some special indicator.

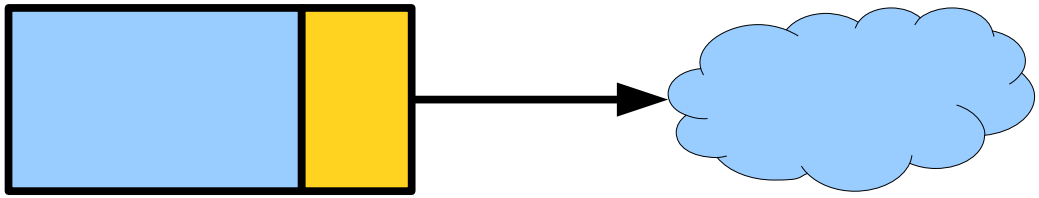


A Linked List is Either...

...an empty list,

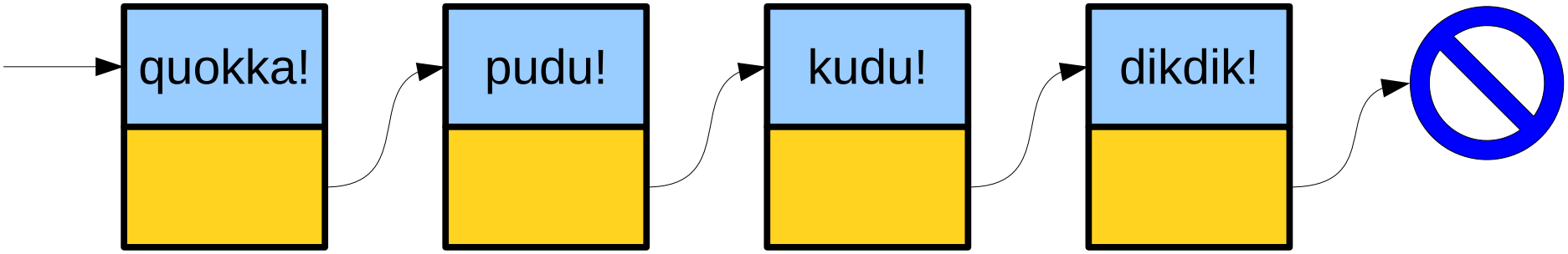
or...





a single cell...

... that points at another linked list.

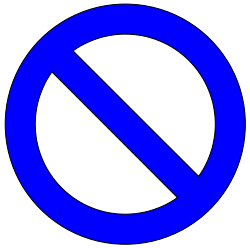


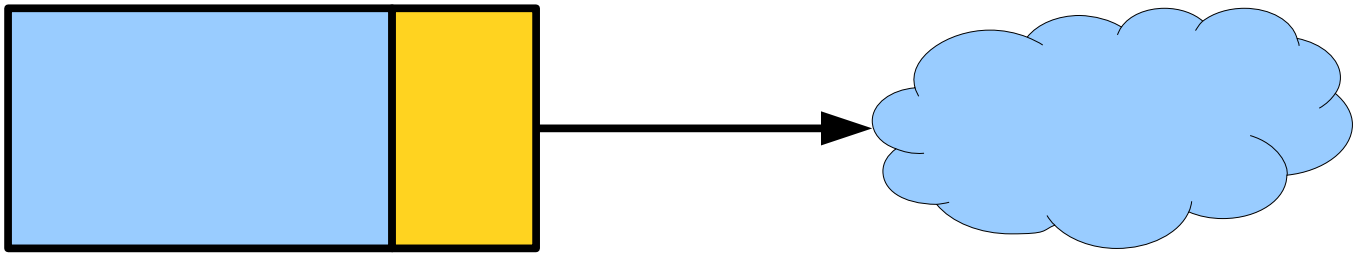
Representing Linked Lists

A Linked List is Either...

...an empty list,

or...





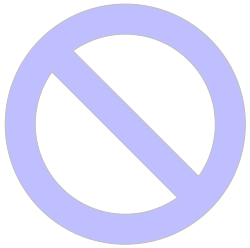
a single cell...

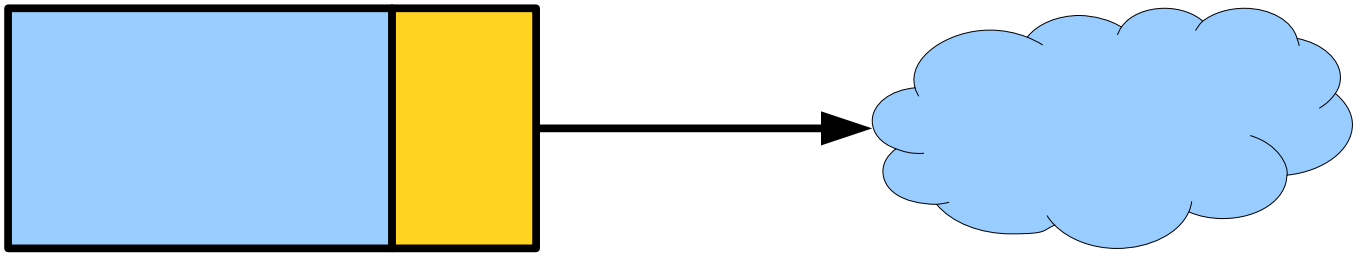
... that points at another linked list.

A Linked List is Either…

…an empty list,

or…

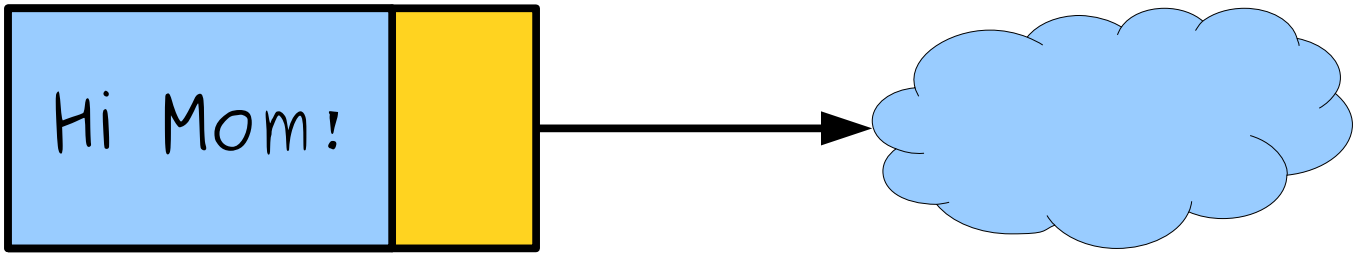




a single cell…

… that points at another linked list.

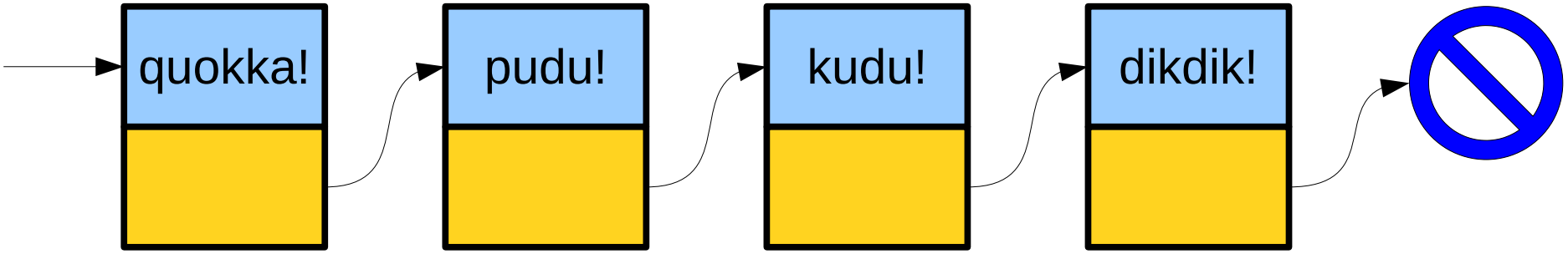
```
struct Cell {
    string value;
    Cell* next;
};
```



a single cell…

… that points at another linked list.

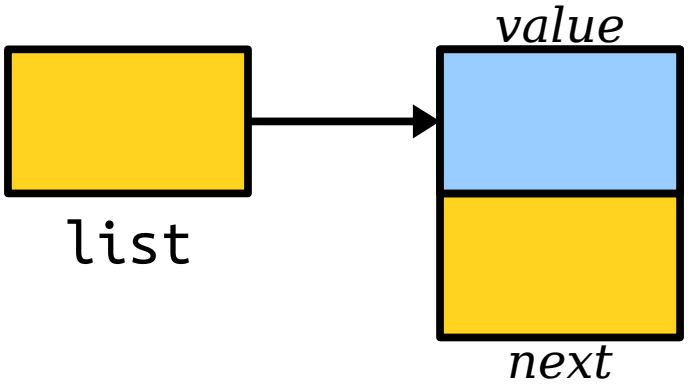
```
struct Cell {
    string value;
    Cell* next;
};
```



```
struct Cell {
    string value;
    Cell* next;
};

Cell* list = new Cell;
```

We just want a single cell, not an array of cells. To get the space we need, we'll just say `new Cell`.

Notice that `list` is still a `Cell*`, a pointer to a cell. It still says "look over there for your `Cell`" rather than "I'm a `Cell`!"

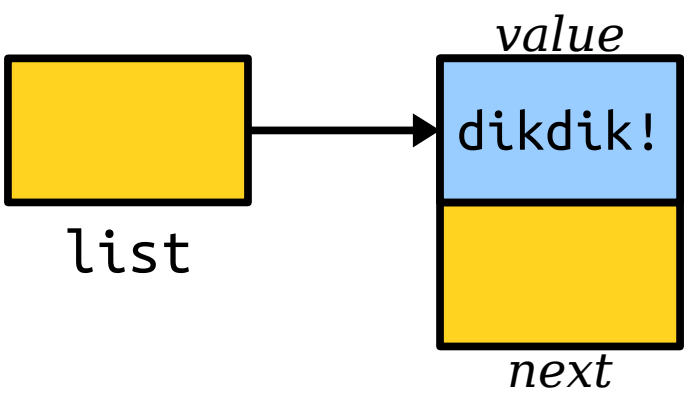


Yes, it's confusing that C++ uses the same types to mean "look over there for an array of `Cells`" and "look over there for a single `Cell`."

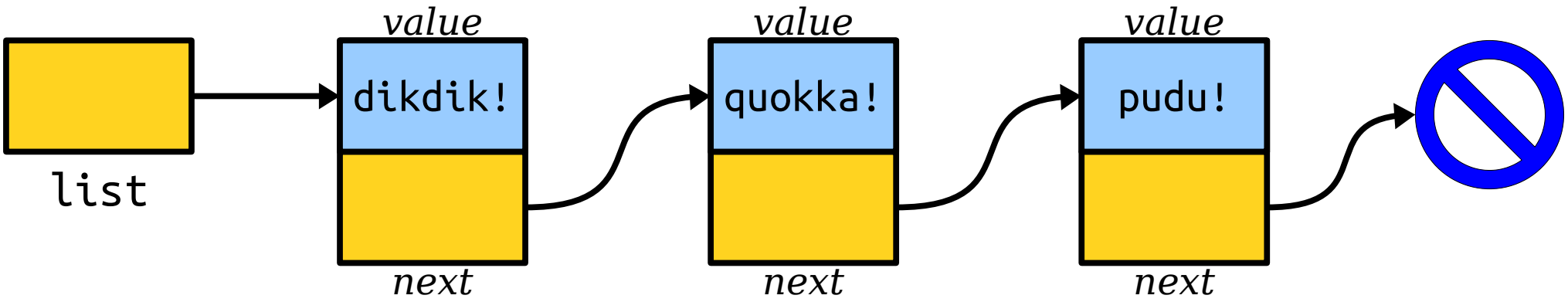
```
struct Cell {
    string value;
    Cell* next;
};

Cell* list = new Cell;
list->value = "dikdik!";
```

Because list is a pointer to a `Cell`, we use the arrow operator `->` instead of the dot operator.

Think of `list->value` as saying "start at `list`, follow an arrow, then pick the `value` field."



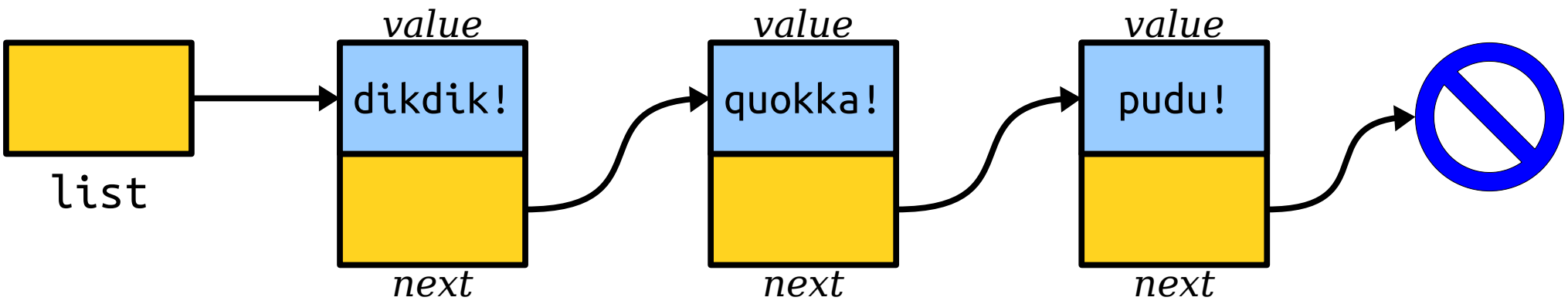
```
struct Cell {
    string value;
    Cell* next;
};

Cell* list = new Cell;
list->value = "dikdik!";
list->next = new Cell;
list->next->value = "quokka!";
list->next->next = new Cell;
list->next->next->value = "pudu!";
list->next->next->next = nullptr;
```



```
struct Cell {
    string value;
    Cell* next;
};

Cell* list = new Cell;
list->value = "dikdik!";
list->next = new Cell;
list->next->value = "quokka!";
list->next->next = new Cell;
list->next->next->value = "pudu!";
list->next->next->next = nullptr;
```

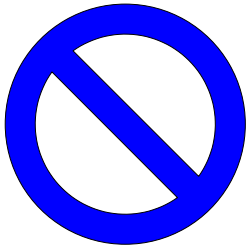
C++ uses the **nullptr** keyword to mean "a pointer that doesn't point at anything."

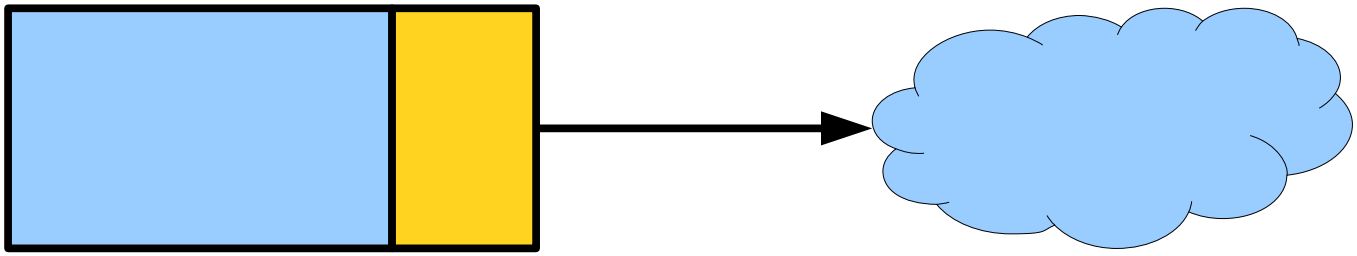
(Older code uses NULL instead of **nullptr**; that's also okay, but we recommend **nullptr**.)



A Linked List is Either…

…an empty list, represented by **nullptr**, or…





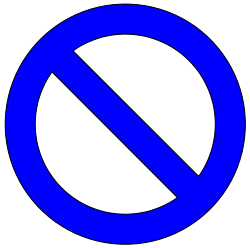
a single linked list cell that points…

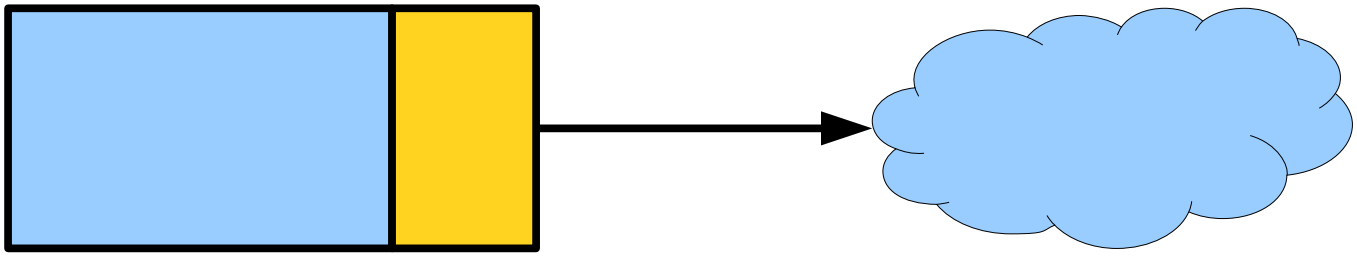
… at another linked list.

Measuring a Linked List

A Linked List is Either…

…an empty list, represented by **nullptr**, or…



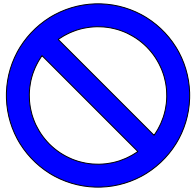


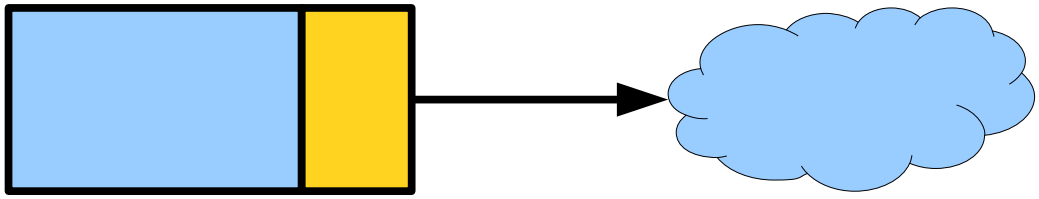
a single linked list cell that points…

… at another linked list.

A Linked List is Either…

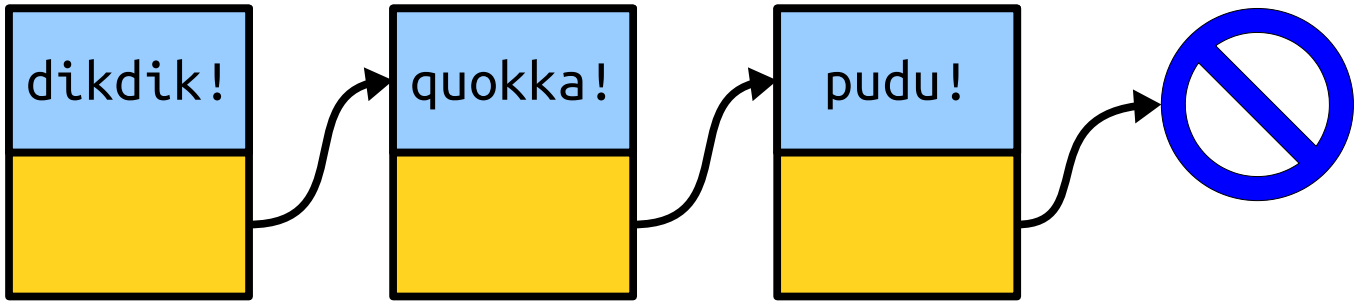
…an empty list, represented by **nullptr**, or…





a single linked list cell that points…

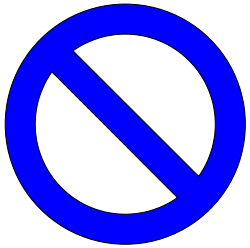
… at another linked list.

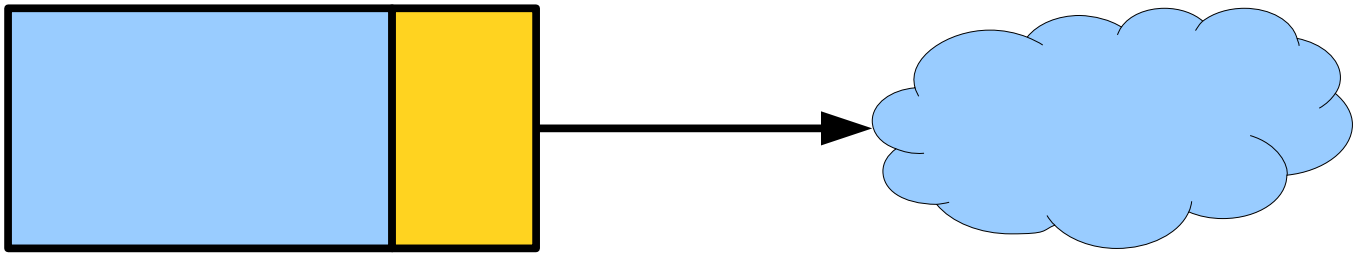


Printing a Linked List

A Linked List is Either…

…an empty list, represented by **nullptr**, or…



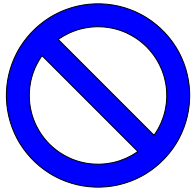


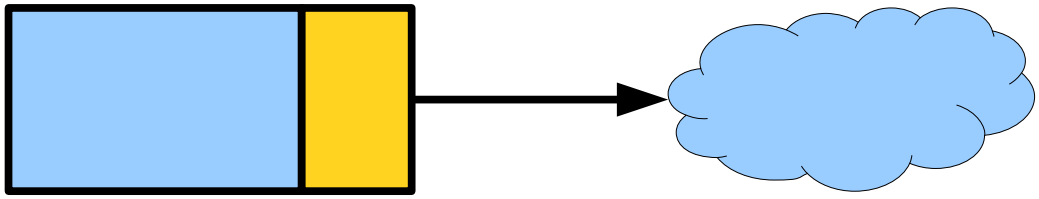
a single linked list cell that points…

… at another linked list.

A Linked List is Either…

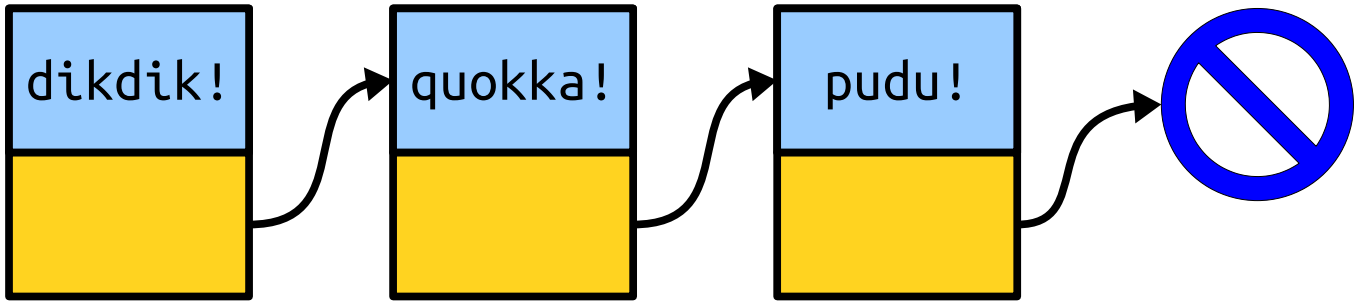
…an empty list, represented by **nullptr**, or…





a single linked list cell that points…

… at another linked list.



What happens if we switch the order of these two lines?

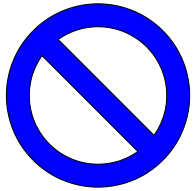
Formulate a hypothesis, but ***don't post anything in chat just yet***.

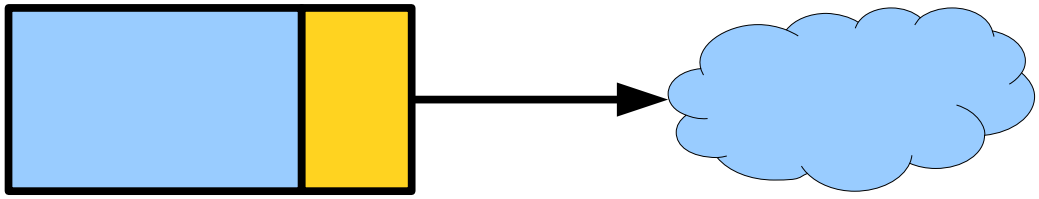
What happens if we switch the order of these two lines?

Now, post your hypothesis in chat. Not sure? Just post “??.”

A Linked List is Either…

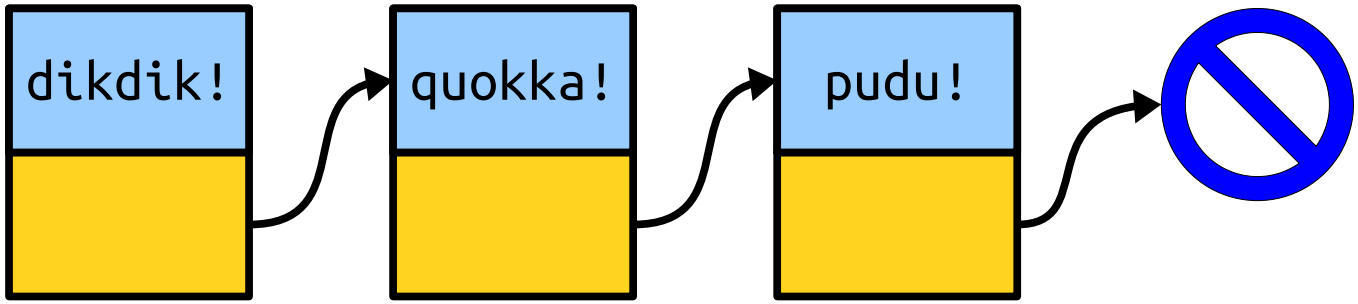
…an empty list, represented by **nullptr**, or…





a single linked list cell that points…

… at another linked list.



Time-Out for Announcements!

Assignment 7

- Assignment 6 was due today at 11:30AM.
  - Grace period for late submissions ends Sunday at 11:30AM Pacific time.
- Assignment 7 (***The Great Stanford Hash-Off***) goes out today. It's due next Friday.
  - Implement linear probing and Robin Hood hashing!
  - See how fast these approaches are and how they compare against chained hashing!
- As always, come talk to us if you have any questions! That's what we're here for.

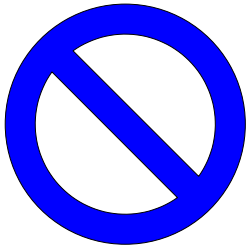
```
lecture = lecture->next;
```

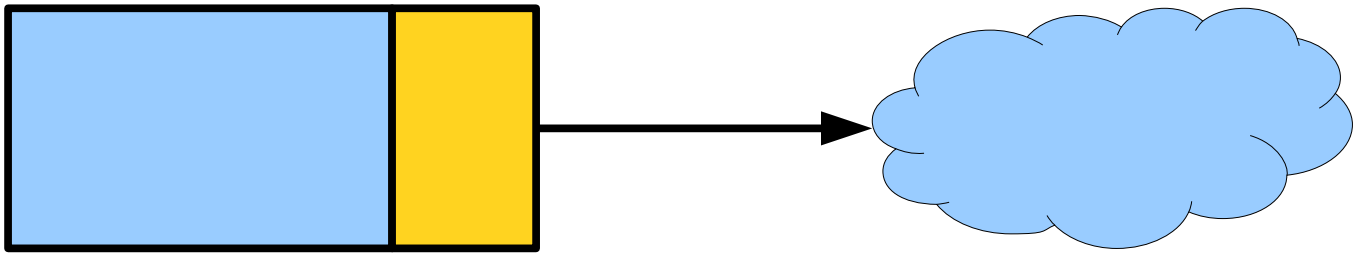
Building a Linked List

(without hardcoding it)

A Linked List is Either…

…an empty list, represented by **nullptr**, or…





a single linked list cell that points…

… at another linked list.

Cleaning Up a Linked List

Endearing C++ Quirks

- If you allocate memory using the `new[]` operator (e.g. `new int[137]`), you have to free it using the `delete[]` operator.

```
delete[] ptr;
```

- If you allocate memory using the `new` operator (e.g. `new Cell`), you have to free it using the `delete` operator.

```
delete ptr;
```

- ***Make sure to use the proper deletion operation***. Mixing these up is like walking off the end of an array or using an uninitialized pointer; it *might* work, or it might instantly crash your program, etc.

Cleaning Up Memory

- To free a linked list, we can't just do this:

```
delete list;
```

- Why not?

Formulate a hypothesis, but ***don't post anything in chat just yet***.

Cleaning Up Memory

- To free a linked list, we can't just do this:

```
delete list;
```

- Why not?

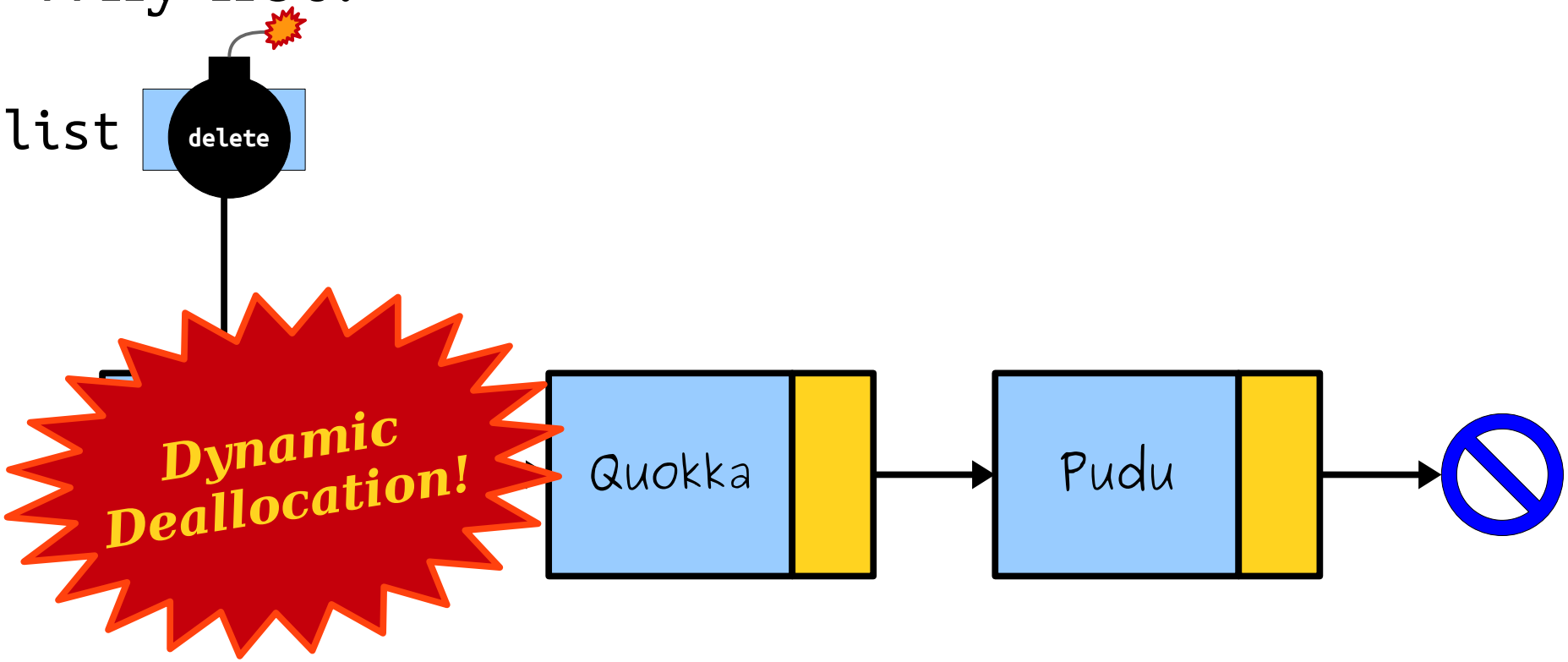
Now, post your hypothesis in chat.

Cleaning Up Memory

- To free a linked list, we can't just do this:

```
delete list;
```

- Why not?

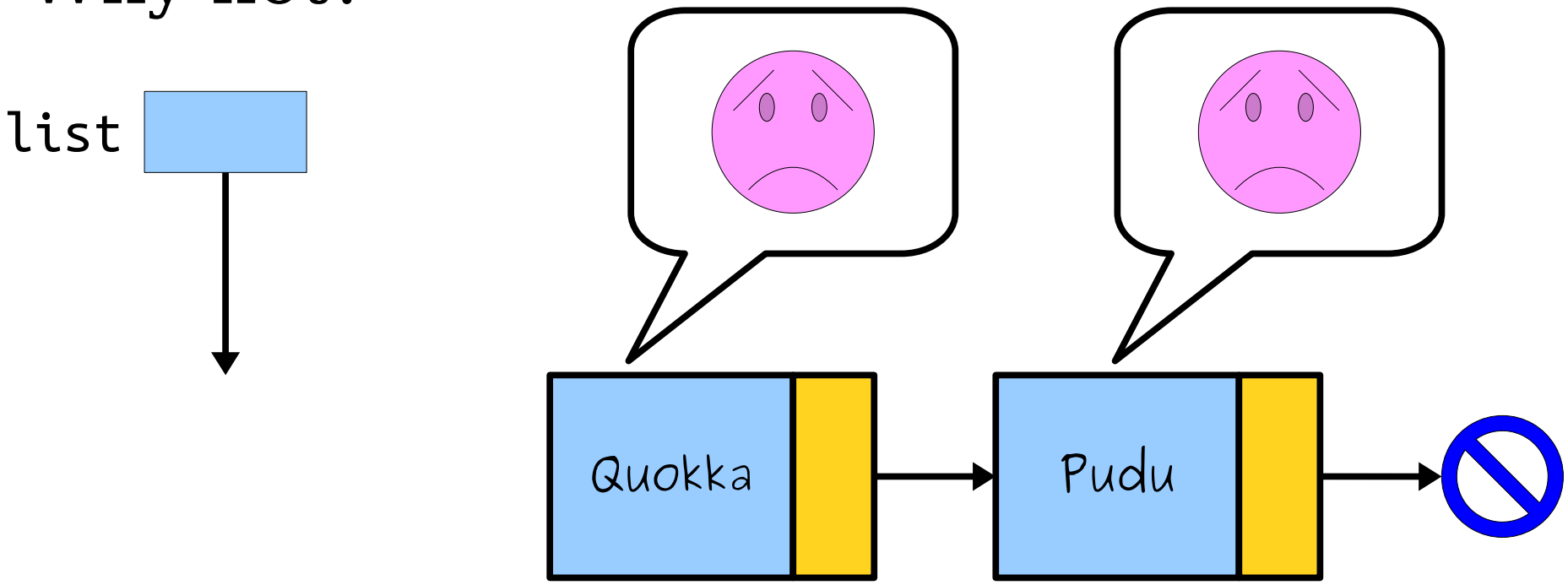


Cleaning Up Memory

- To free a linked list, we can't just do this:

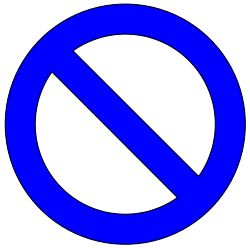
```
delete list;
```

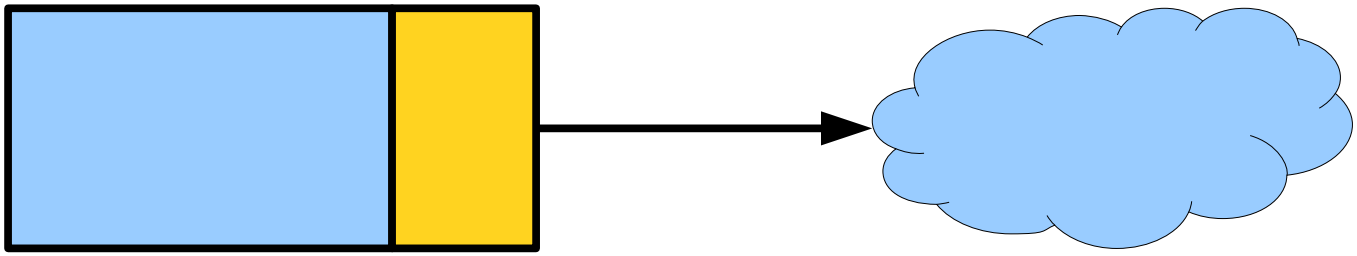
- Why not?



A Linked List is Either…

…an empty list, represented by **nullptr**, or…





a single linked list cell that points…

… at another linked list.

What’s wrong with this code?

Formulate a hypothesis, but ***don’t post anything in chat just yet***.

What’s wrong with this code?

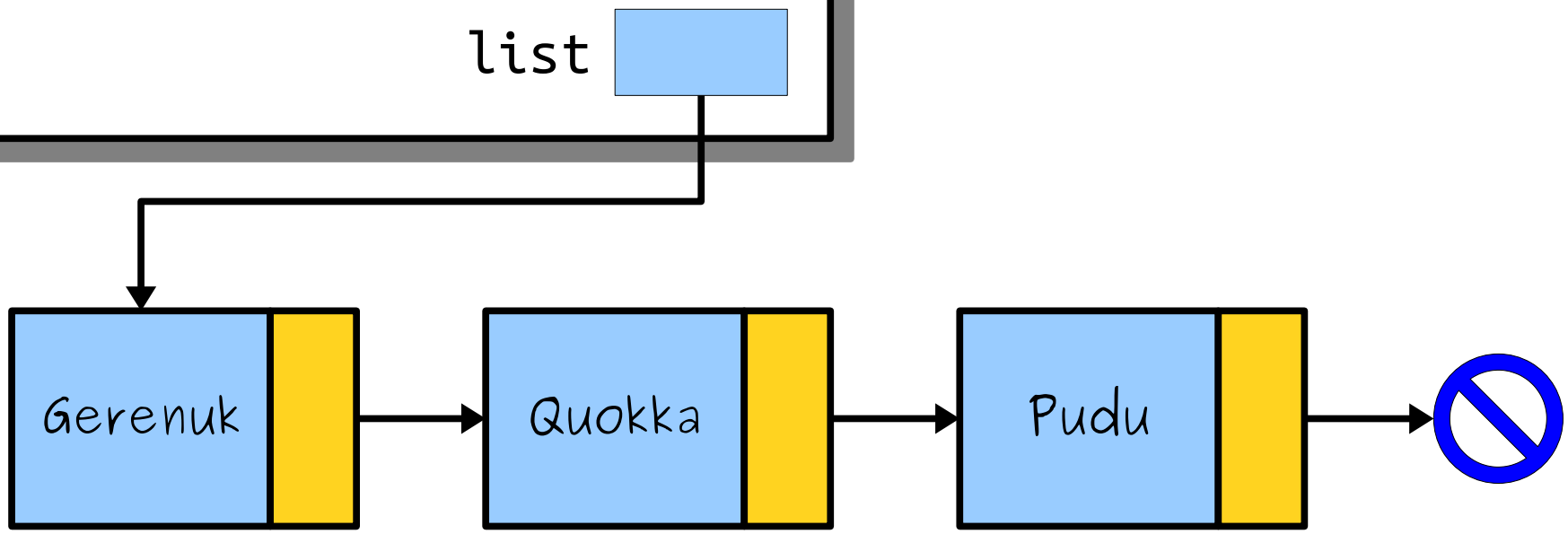
Now, post your hypothesis in chat. Not sure? Just post “??.”

Watch Out!

```
void deleteList(Cell* list) {
    if (list == nullptr) return;

    delete list;
    deleteList(list->next);
}
```

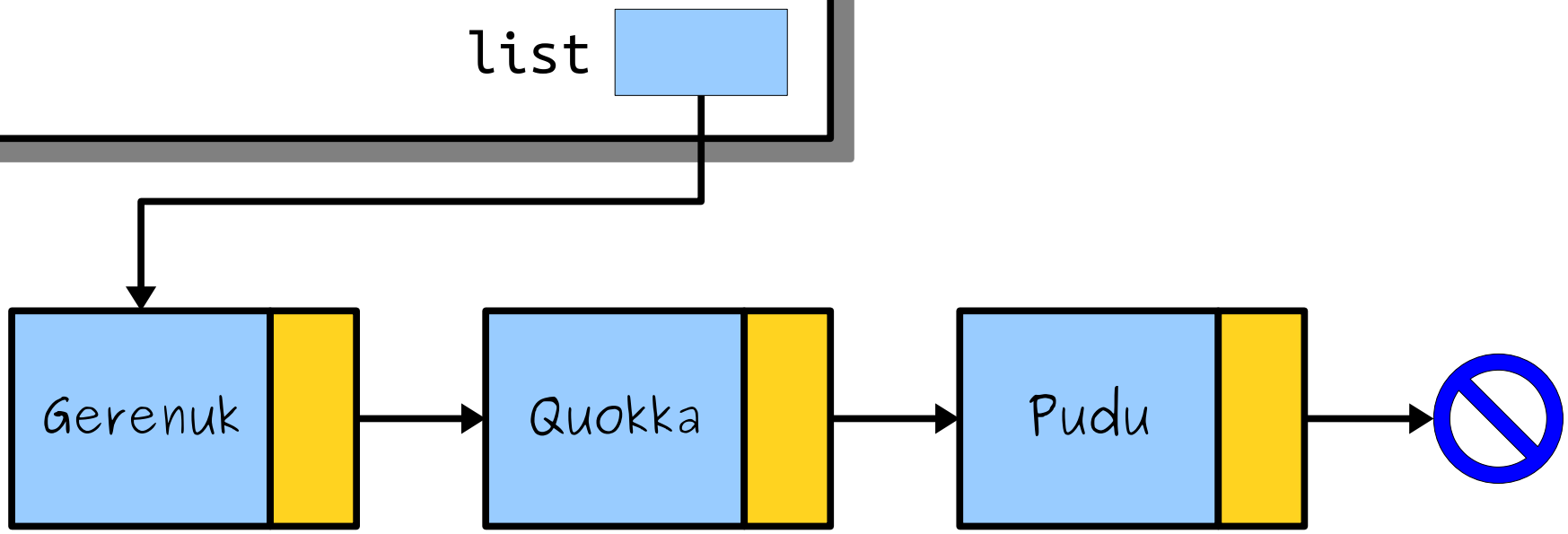
Watch Out!

```
void deleteList(Cell* list) {
    if (list == nullptr) return;

    delete list;
    deleteList(list->next);
}
```



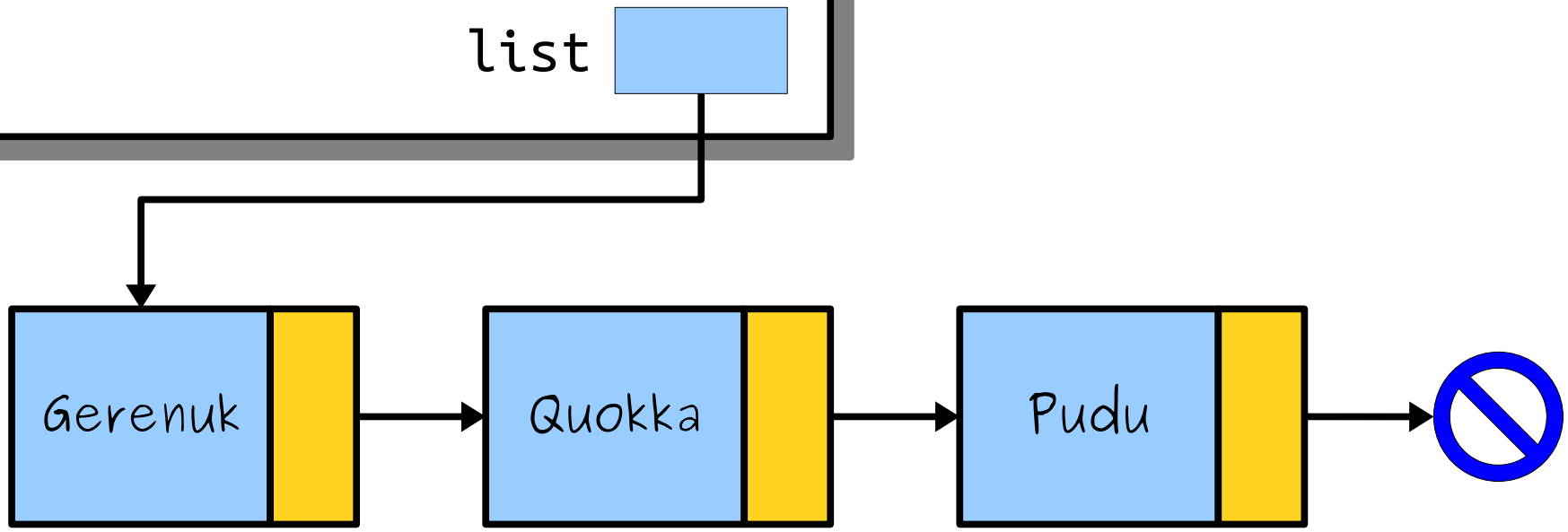
Watch Out!

```
void deleteList(Cell* list) {
   if (list == nullptr) return;

   delete list;
   deleteList(list->next);
}
```



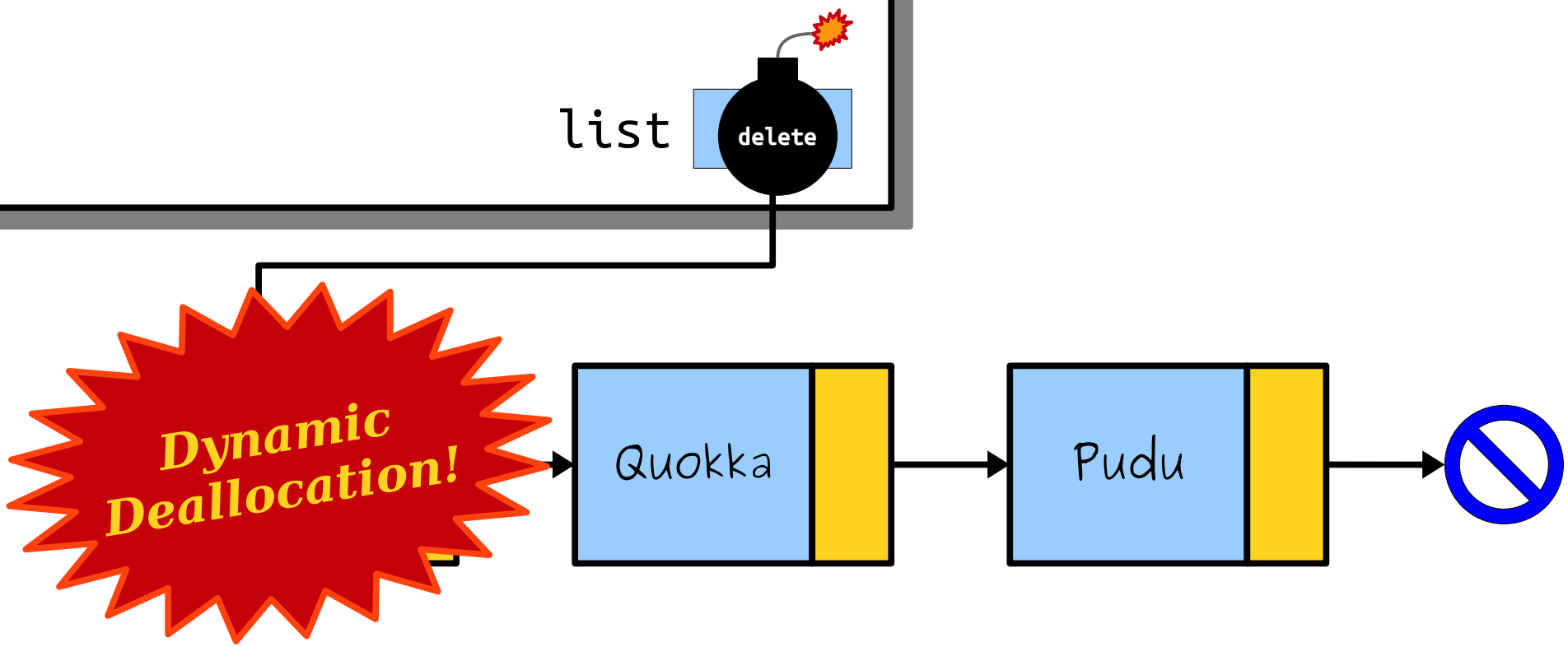
Watch Out!

```
void deleteList(Cell* list) {
    if (list == nullptr) return;

    delete list;
    deleteList(list->next);
}
```



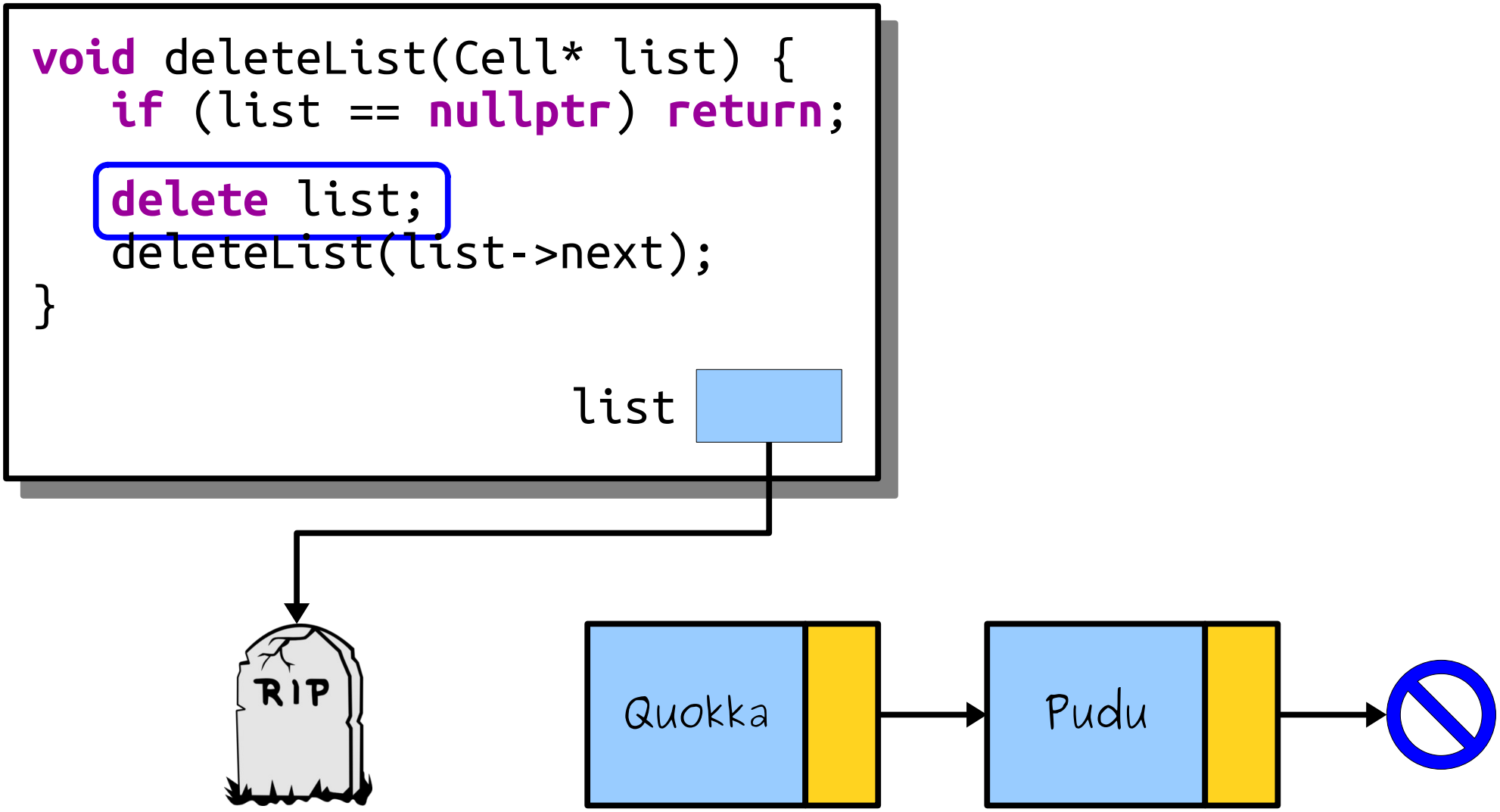
Watch Out!

```
void deleteList(Cell* list) {
    if (list == nullptr) return;

    delete list;
    deleteList(list->next);
}
```



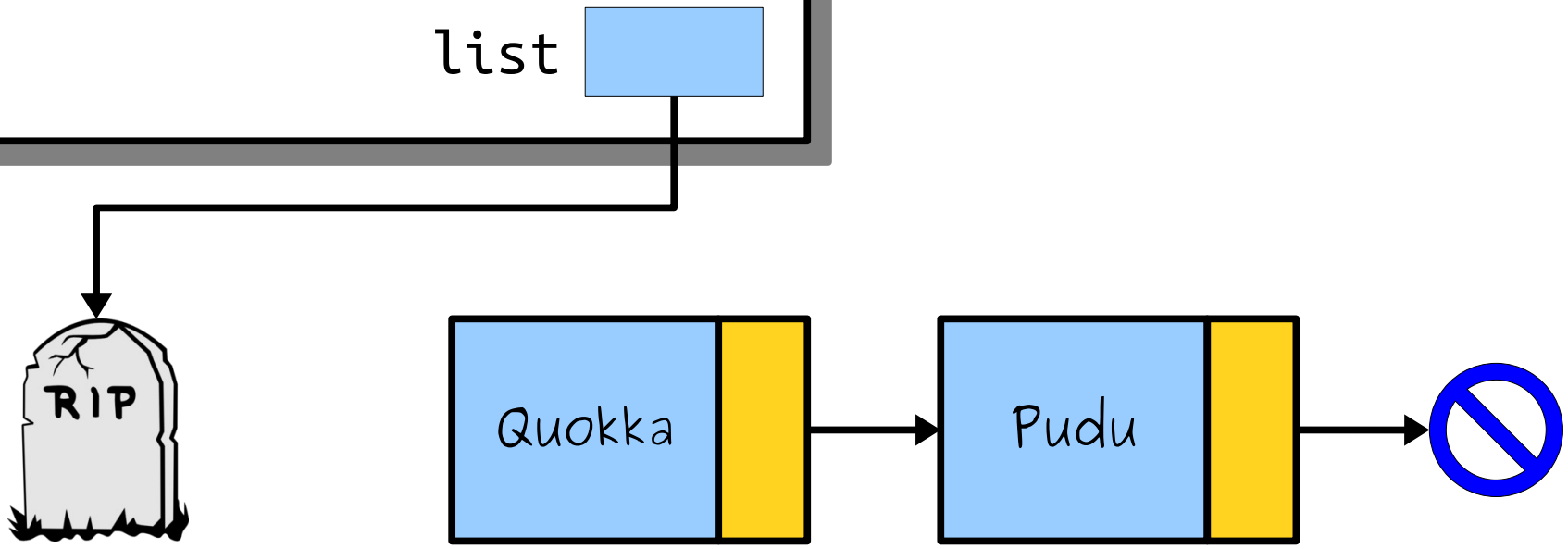
Watch Out!

```
void deleteList(Cell* list) {
    if (list == nullptr) return;

    delete list;
    deleteList(list->next);
}
```



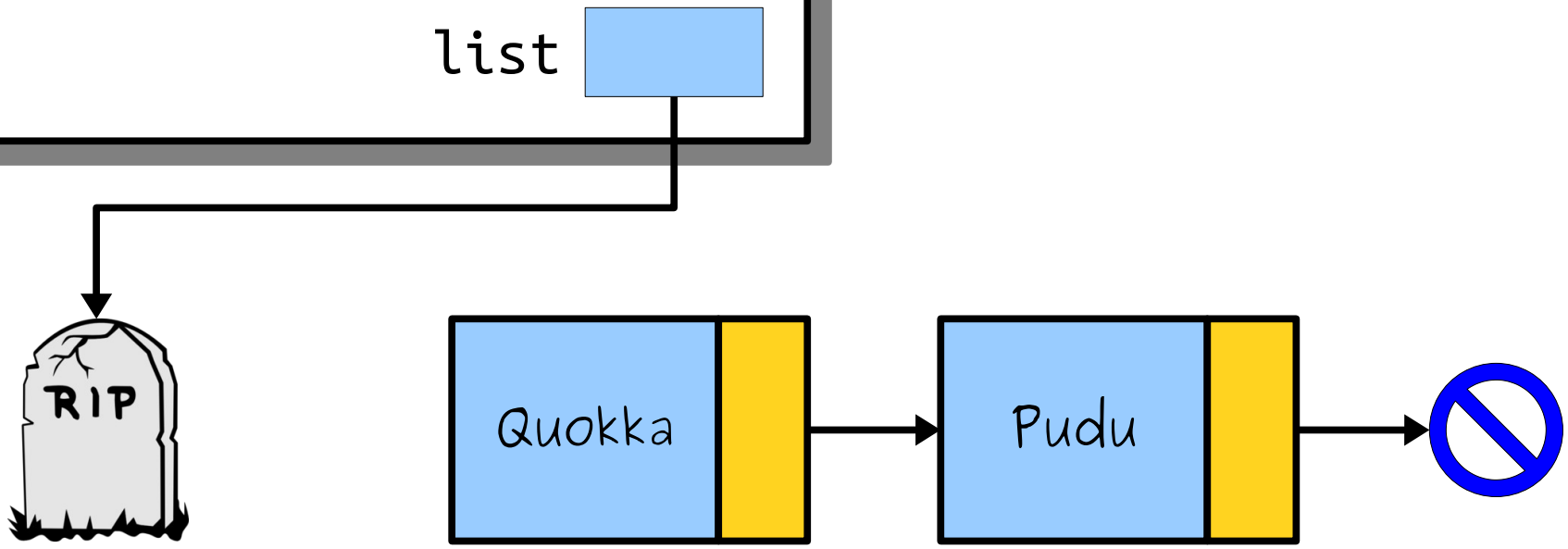
Watch Out!

```
void deleteList(Cell* list) {
    if (list == nullptr) return;

    delete list;
    deleteList(list->next);
}
```



Watch Out!

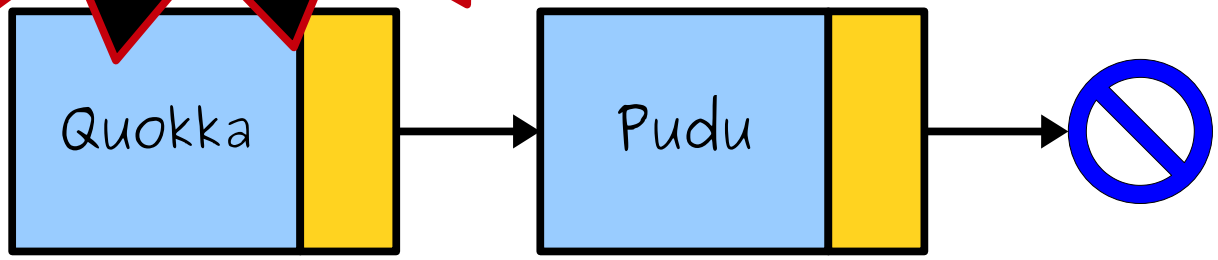
```
void deleteList(Cell* list) {
    if (list == nullptr) return;

    delete list;
    deleteList(list->next);
}
```



Watch Out!

```
void deleteList(Cell li
    if (list == n

    delete list
    deleteLi
}
```

Undefined behavior!

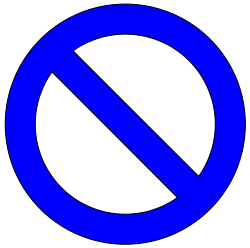


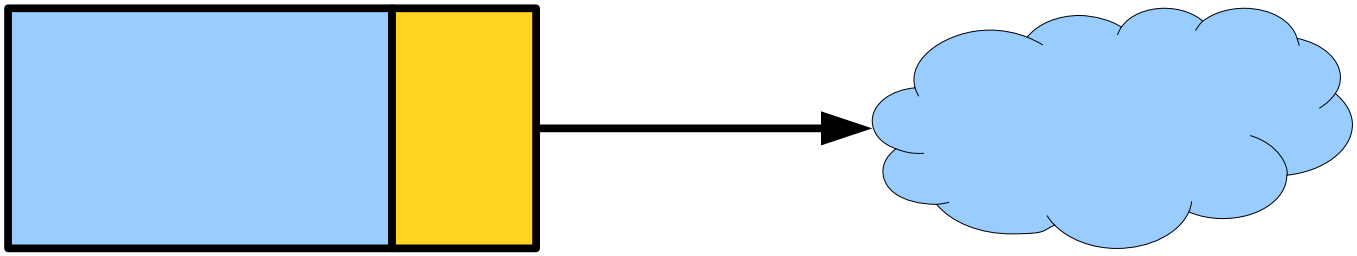
In the Land of C++, we
do not speak to the dead.

What should we do instead?

A Linked List is Either…

…an empty list, represented by **nullptr**, or…



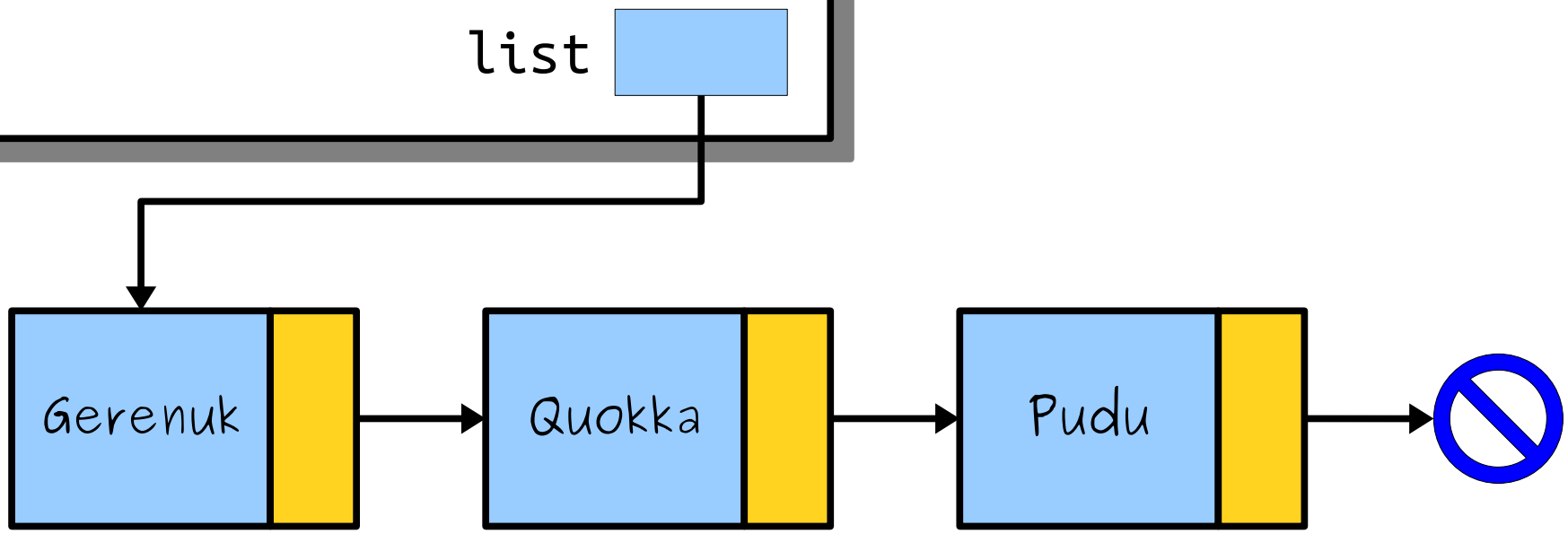


a single linked list cell that points…

… at another linked list.

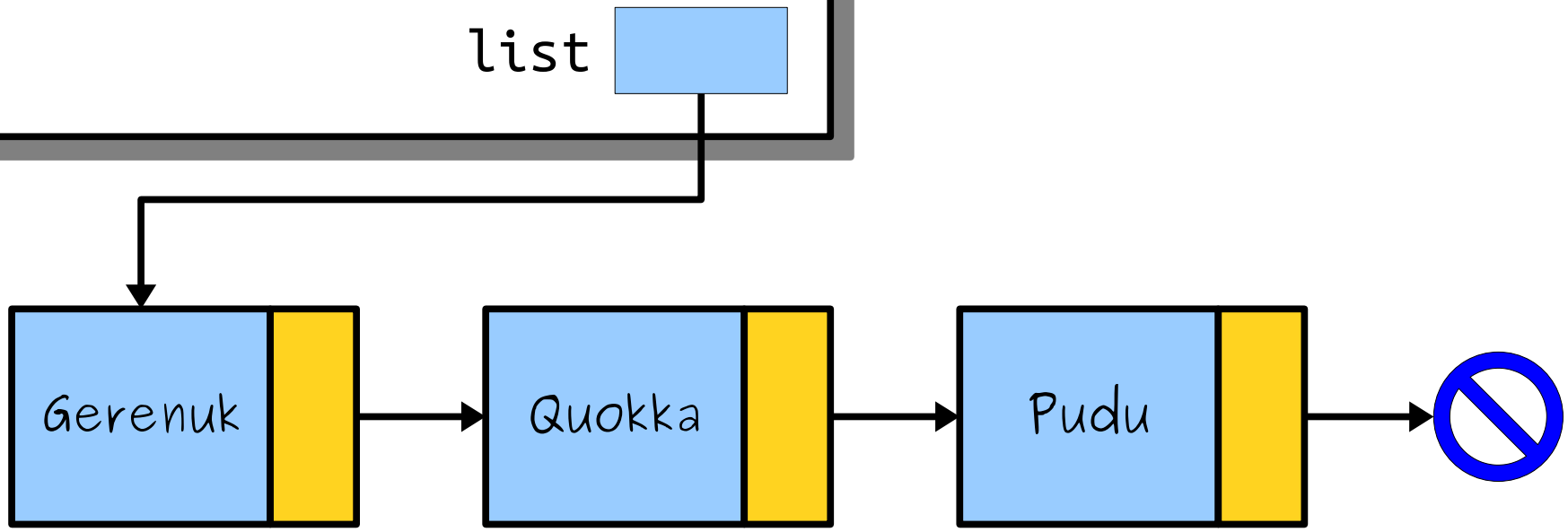
One Option

```
void deleteList(Cell* list) {
    if (list == nullptr) return;

    deleteList(list->next);
    delete list;
}
```



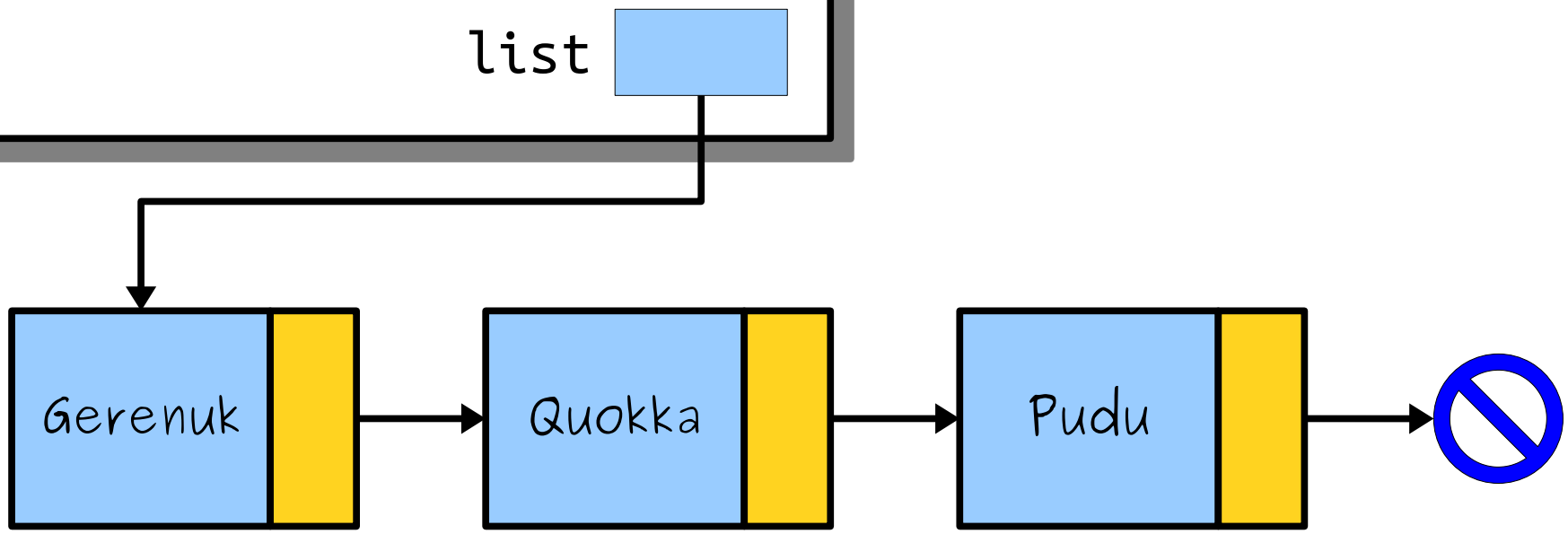
One Option

```
void deleteList(Cell* list) {
   if (list == nullptr) return;

   deleteList(list->next);
   delete list;
}
```



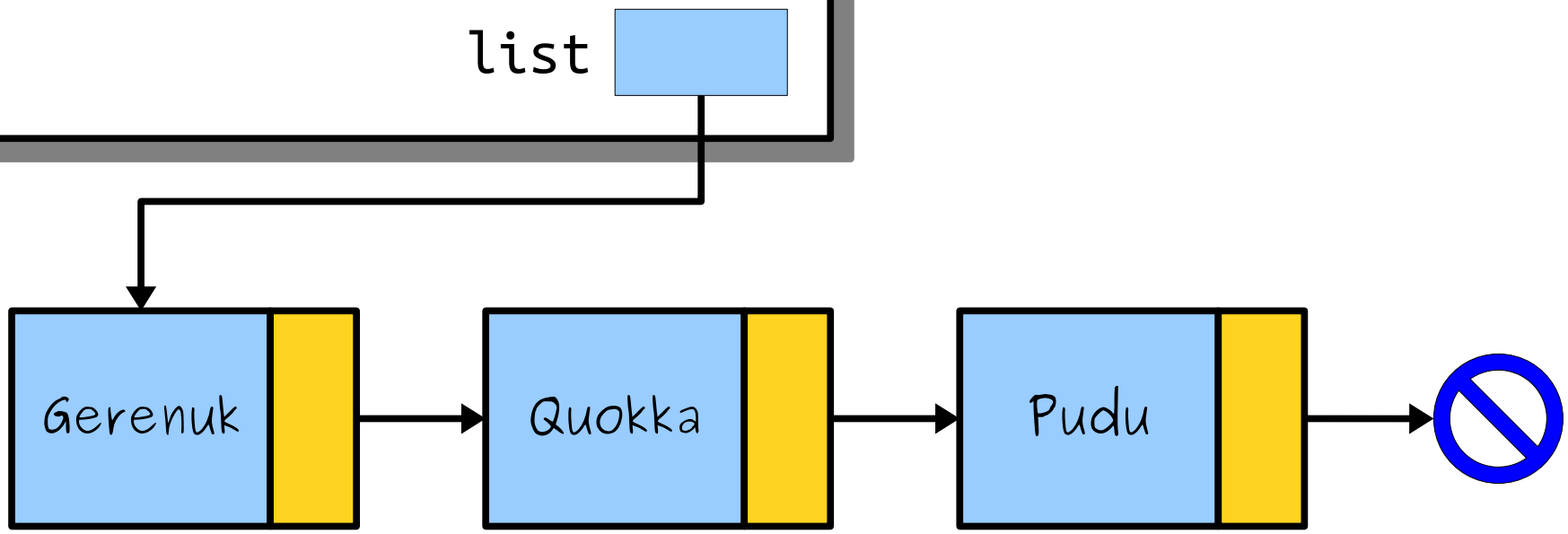
One Option

```
void deleteList(Cell* list) {
    if (list == nullptr) return;

    deleteList(list->next);
    delete list;
}
```



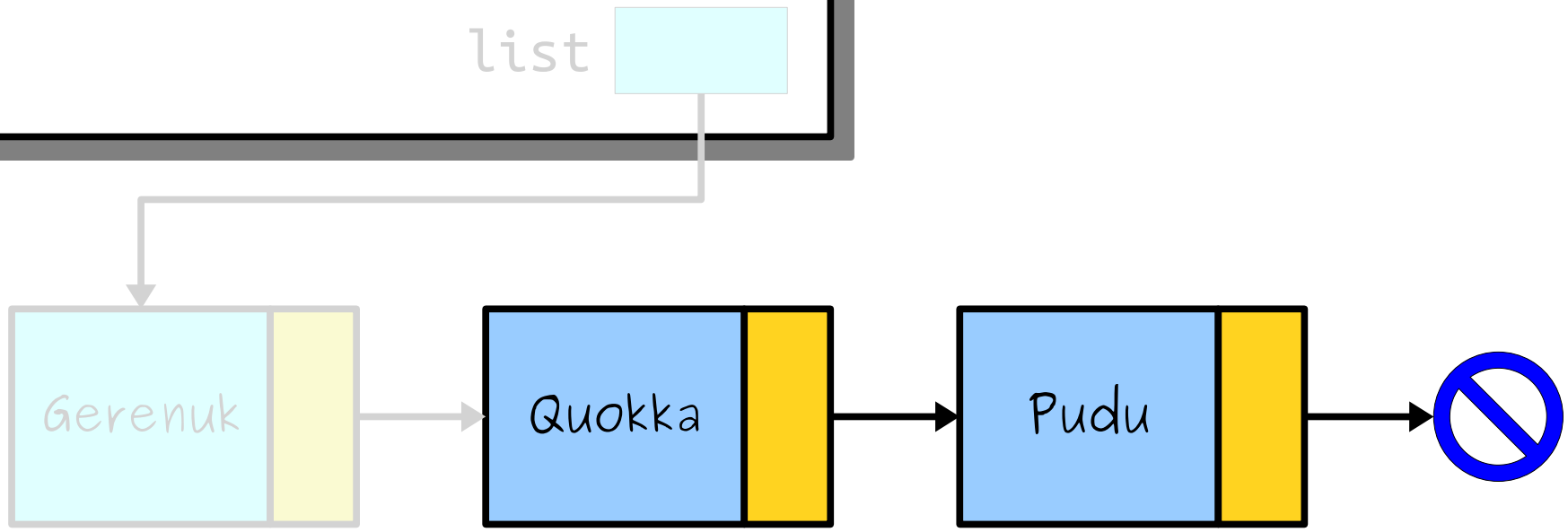
One Option

```
void deleteList(Cell* list) {
    if (list == nullptr) return;

    deleteList(list->next);
    delete list;
}
```



One Option

```
void deleteList(Cell* list) {
    if (list == nullptr) return;

    deleteList(list->next);
    delete list;
}
```

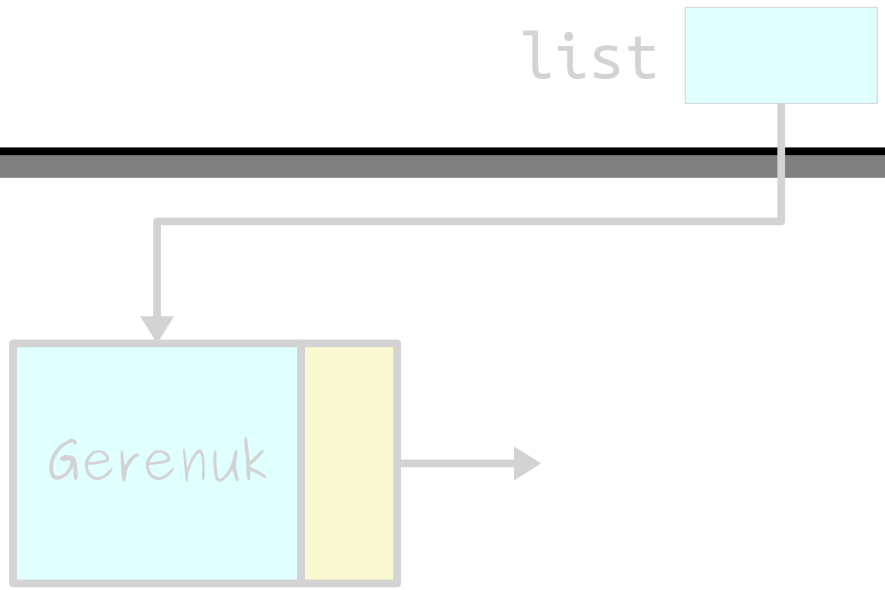


One Option

```
void deleteList(Cell* list) {
    if (list == nullptr) return;

    deleteList(list->next);
    delete list;
}
```

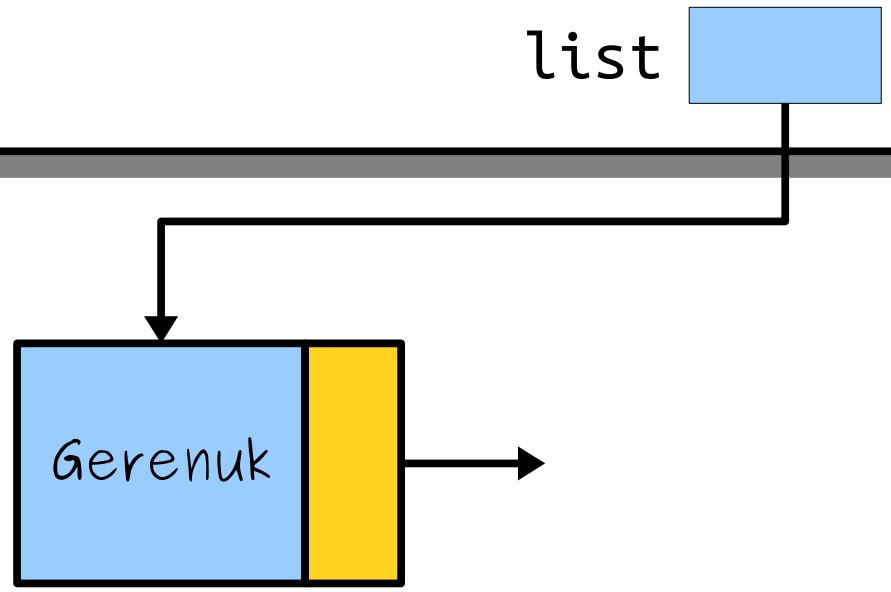
One Option

```
void deleteList(Cell* list) {
    if (list == nullptr) return;

    deleteList(list->next);
    delete list;
}
```



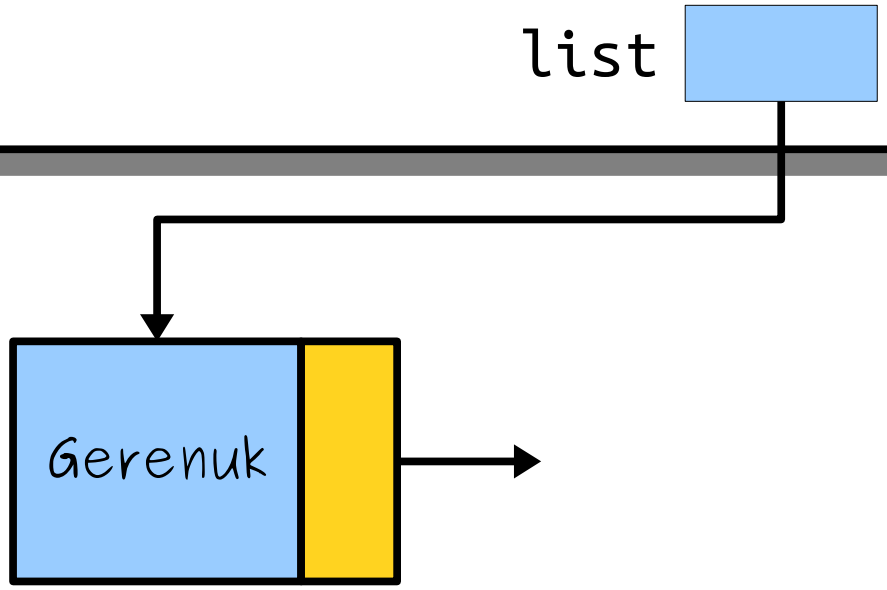
One Option

```
void deleteList(Cell* list) {
    if (list == nullptr) return;

    deleteList(list->next);
    delete list;
}
```



One Option

```
void deleteList(Cell* list) {
    if (list == nullptr) return;

    deleteList(list->next);
    delete list;
}
```



One Option

```
void deleteList(Cell* list) {
    if (list == nullptr) return;

    deleteList(list->next);
    delete list;
}
```

One Option

```
void deleteList(Cell* list) {
    if (list == nullptr) return;

    deleteList(list->next);
    delete list;
}
```

Your Action Items

- ***Read Chapter 12.1 – 12.3.***
  - There's lots of useful information in there about how to work with linked lists.
- ***Start Assignment 7***
  - Start working on linear probing. As always, come talk to us if you have any questions!

Next Time

- ***Linked Lists, Iteratively***
  - How do you manually walk a linked list?
- ***Pointers Into Lists***
  - Getting a helping hand.