“Reverse a Linked List on the Whiteboard”

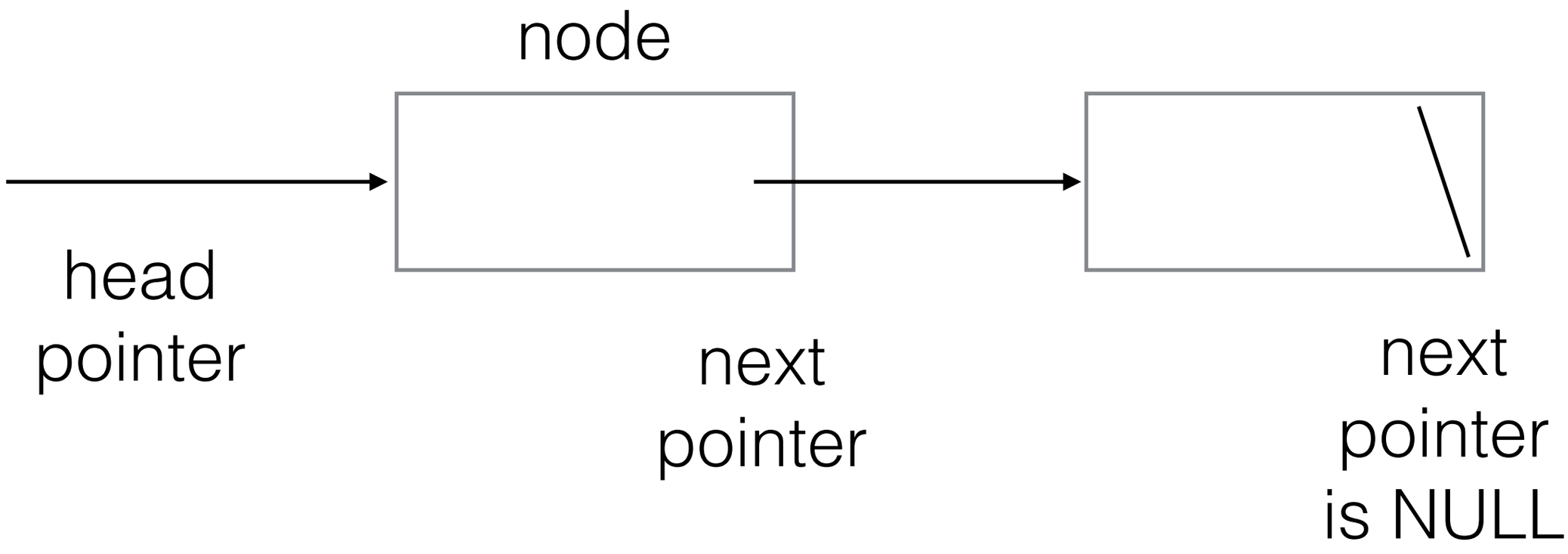
and other lamentable interview questions

Anton Gerdelan <gerdela@scss.tcd.ie>

First of All - Let's Look at Linked Lists Again

- Diagram style with pointers and boxes
- What code goes into a node?
- Pros and cons of a dynamic list
- Can't we just implement them in an array?
- Variations and similar data structures

Linked List Diagrams



More Risky Live Coding

```
21  typedef struct Node Node; // forward declare type and alias `struct Node` with `Node`
22  struct Node{
23          char data[256];
24          Node* next; // note: we refer to our own type here
25  };
```

- Create a list **head** pointer
- I add a first node to it here
  `Node* head = (Node*)malloc( sizeof( Node ) );`
- Remember to initialise the node's memory
  - use `calloc()` instead of `malloc()` for all zero
  - or `memset()`
  - or manually set each item
    `head->next = NULL;` …

Pros and Cons vs Static Arrays

- Don’t need to know size in advance
    - Can shrink and expand - can’t regular dynamic memory do this too?
- Indexing and random access
- Can remove any node and shuffle sort-of easily
- Lots of pointers
- Memory allocation all over the shop

Array Implementation

- Array underneath and add() remove() insert() delete() on top?
- Or linked list underneath and index(i) on top?
- This do-both version is usually called a "**List**" or a "**Vector**"
- Keep memory complexity in mind

Reverse a Linked List

- Consider replacement with a **doubly-linked list** or **array** instead
  - reason about pros/cons of this (space, time)
- You need to **add extra pointers** to help you
  - Because links will be broken and you don't want to lose nodes
  - Add Previous, Current, Next pointers
  - I don't think it's possible with fewer.
  - If in doubt, add as many pointers as you want and optimise later.
  - Q. What is the time T() complexity of this operation?

Blackboard

Another Look

- What is the job of a computer program / algorithm again?
- Is [ ]->[ ]->[/] what our input/output data actually looks like?
- What does it actually look like?
- What if our list is implemented such that node data is <u>contiguous</u> in memory in input instance and output permutation?
- Can we build a better algorithm knowing that?

Back to the Blackboard!

- Is this easier to implement in code?
- What is the T() complexity of this?
    - How long is the loop?
- Note: this is the same algorithm for flipping your image upside-down
    - replace ‘node size’ with ‘row size’

Tech Interviews / Exams

- “You have 2 days/2 hours to code this problem - send it back to us, don’t use the web”.
- Definitely use the web - know the simple answer (but maybe don’t rely on the first answer from SE)
- Don’t waste any time - simple, clear, code.
- Fit in with their conventions, don’t force your own style in.

Tech Interviews / Exams

- “Use this telecom software and live code this problem while we watch”
    - Get used to coding in front of / helping your peers (or sharing open-source projects)
    - Practise reading and building/modifying e.g. open source code
    - Be prepared - batteries/lighting/stable internet/ good computer/distraction free

Tech Interviews / Exams

- “Find the bugs in this code”
    - Get used to reading other code/projects
    - Know how to use the basic tools
        - Debugger (including the on-paper version)
        - Memory inspection (and/or Valgrind)
        - Profiler
- Have a folder of your own code to show (or e.g. GitHub project)
- **“How could my/this code be more efficient?”**