Detecting Cycle in Singly Linked List

A slightly silly Microsoft interview question.
But still a nice question.
Cycle in List

head → [Diagram] head

Goal: distinguish one situation from the other.
One (Bad) Idea

Somehow mark nodes that have been visited.

Node visit = head;
The Bad Idea

We’ll leave it open how to mark the node.

\[
\text{visit} = \text{visit.next};
\]
The Bad Idea

<mark node>
visit = visit.next;

![Diagram of linked list with marked node and visit variable]
The Bad Idea

<mark node>
visit = visit.next;

head → node1 → node2 → node3 → node4 → node5 → head

visit → node1

visit → node2

visit → node4

visit → node5
The Bad Idea

\[
\text{mark node}
\]\n\[
\text{visit} = \text{visit.next};
\]
The Bad Idea

<mark node>
visit = visit.next;

head → mark → mark → mark → mark → visit

visit
The Bad Idea

<mark node>
visit = visit.next;

The Bad Idea

<mark node>
visit = visit.next;

head → <mark node> visit → <mark node> visit → <mark node> visit → <mark node> visit → <mark node> visit

visit → <mark node> visit → <mark node> visit → <mark node> visit → <mark node> visit
The Bad Idea

Stop when we see visited node (or null).
Why a Bad Idea?

Well, it’s not really bad. But we either need to add a field `visited` to each node or maintain an array of the nodes seen so far. Requires extra space and possibly time.
The original question

Use a constant amount of extra space (irrespective of the number of nodes in the list).
The Nice Idea

Pointer jumping:

Node slow = head;
Node fast = head;
The Nice Idea

We’ll live dangerously with no error checking.

```java
slow = slow.next;
fast = fast.next.next;
```
The Nice Idea

slow = slow.next;
fast = fast.next.next;
The Nice Idea

slow = slow.next;
fast = fast.next.next;
The Nice Idea

slow = slow.next;
fast = fast.next.next;
The Nice Idea

slow = slow.next;
fast = fast.next.next;
The Nice Idea

slow = slow.next;
fast = fast.next.next;
Stop Now

do all this
while (slow != fast)

Since the pointers collide, there is a cycle.

If either pointer hits null, then no cycle.