CMSC 21

FUNDAMENTALS OF PROGRAMMING

Kristine Bernadette P. Pelaez

Institute of Computer Science
University of the Philippines Los Baños

LINKED LIST OPERATIONS

Operations

Insert (add a node)

Delete (delete a node)

Search (search the list)

View (print the contents of the list)

Delete

delete nodes from a linked list

Delete

delete nodes from a linked list

has three(3) cases:

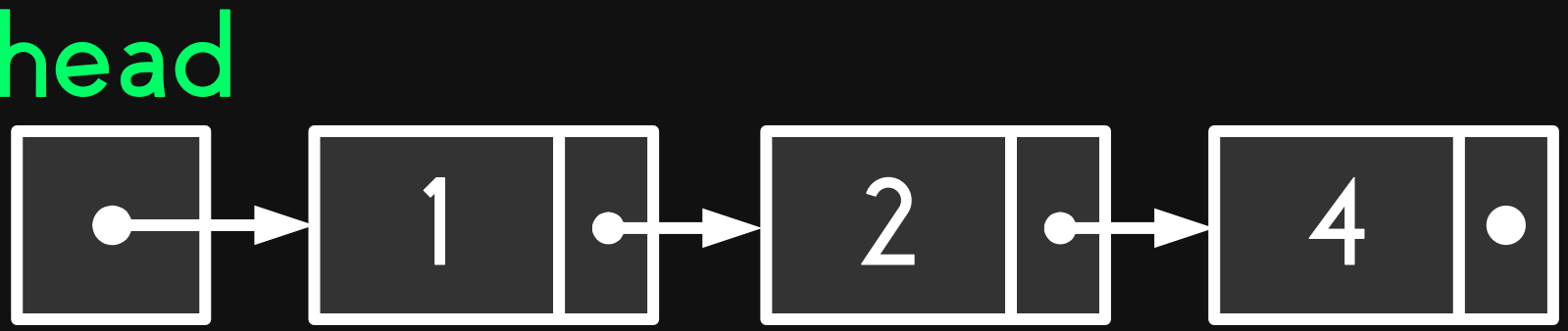
- delete at head
- delete at middle
- delete at tail

Delete at Head

delete the
first element
of the list

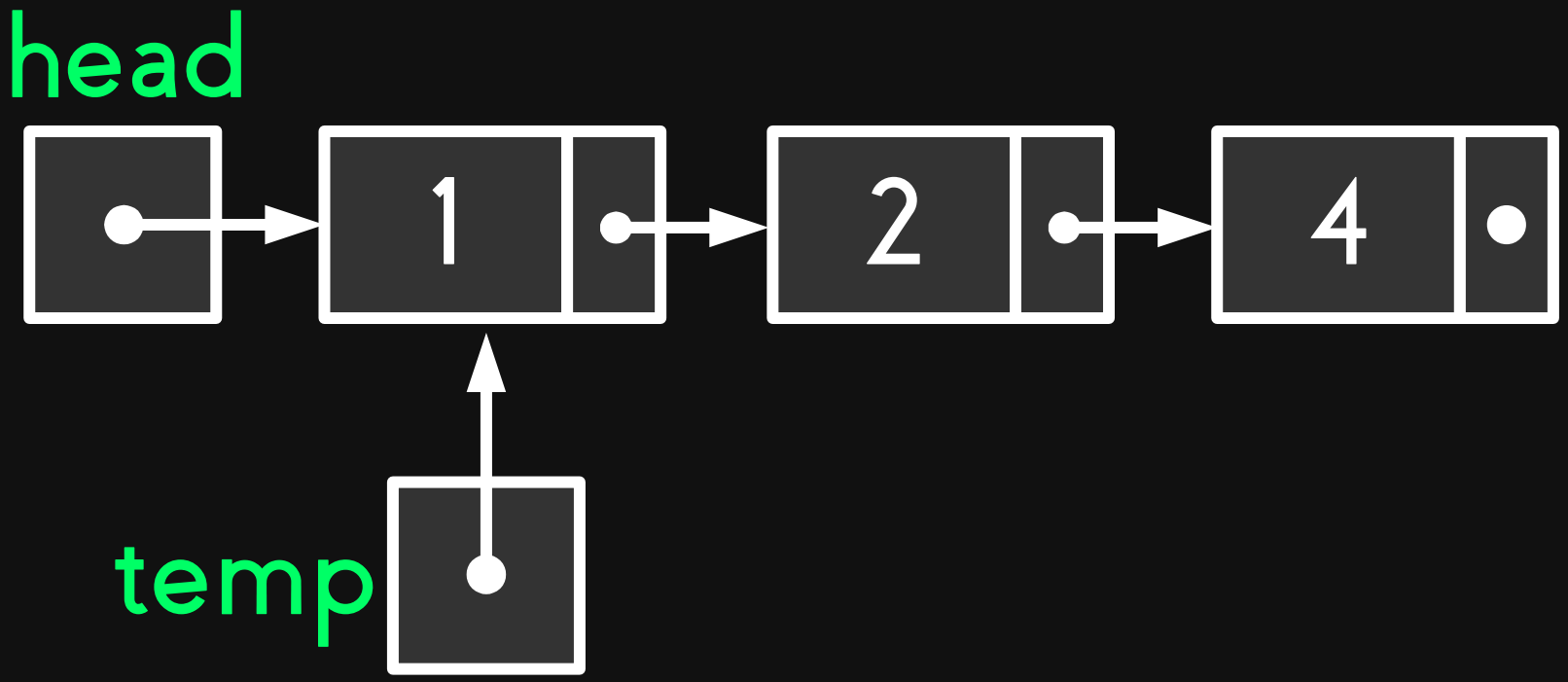
```
struct NODE{
   int num;
   struct NODE *next;
};
```

Delete at Head



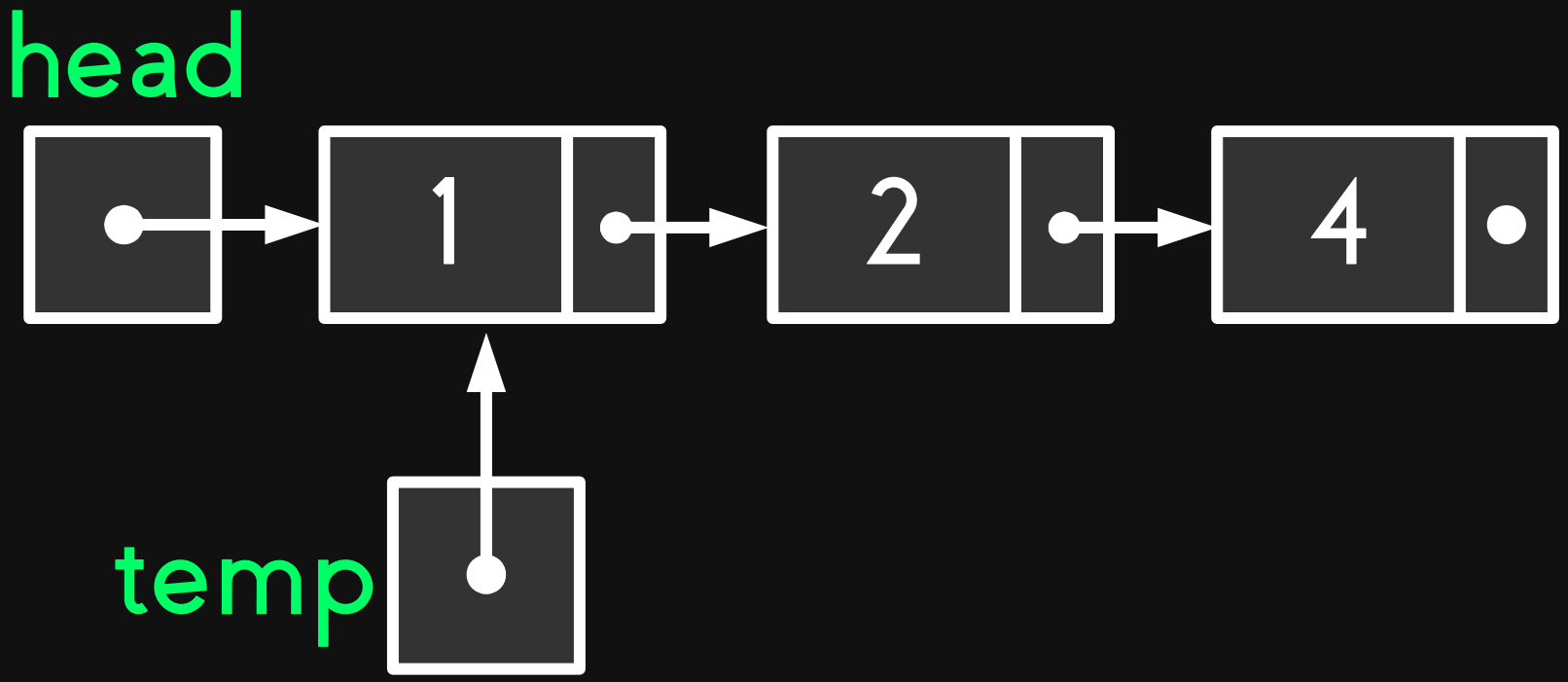
delete the first element (1)

Delete at Head



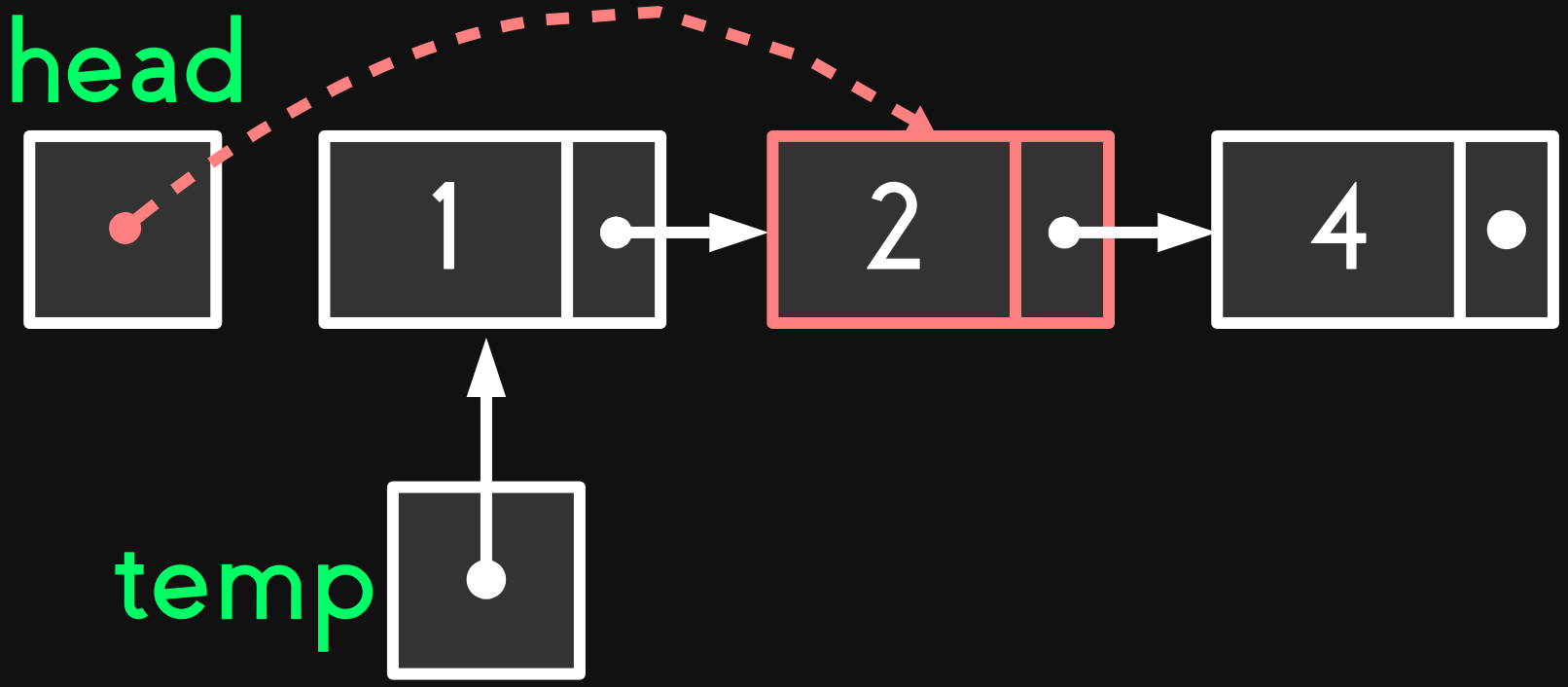
make a pointer (`temp`) point to
the node to be deleted

Delete at Head



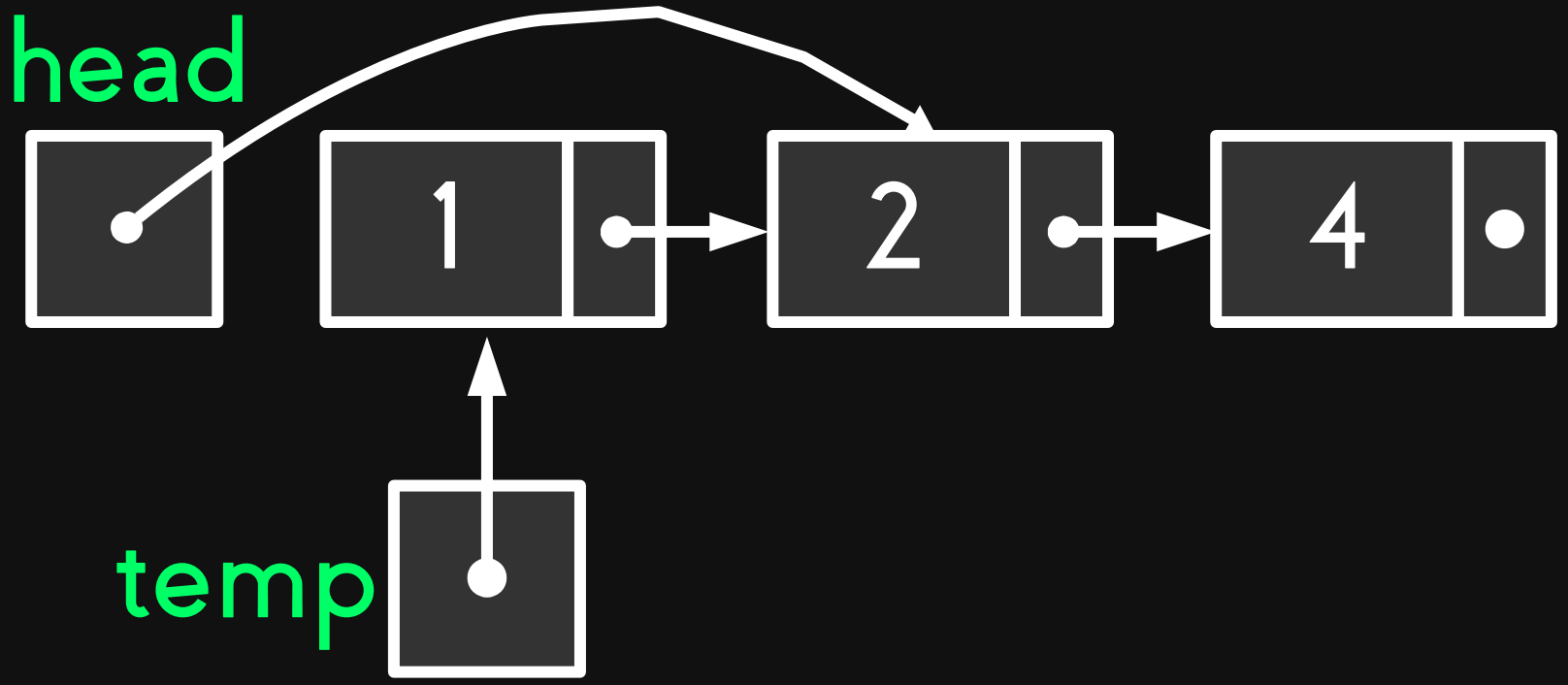
point head to the node after the
node to be deleted

Delete at Head

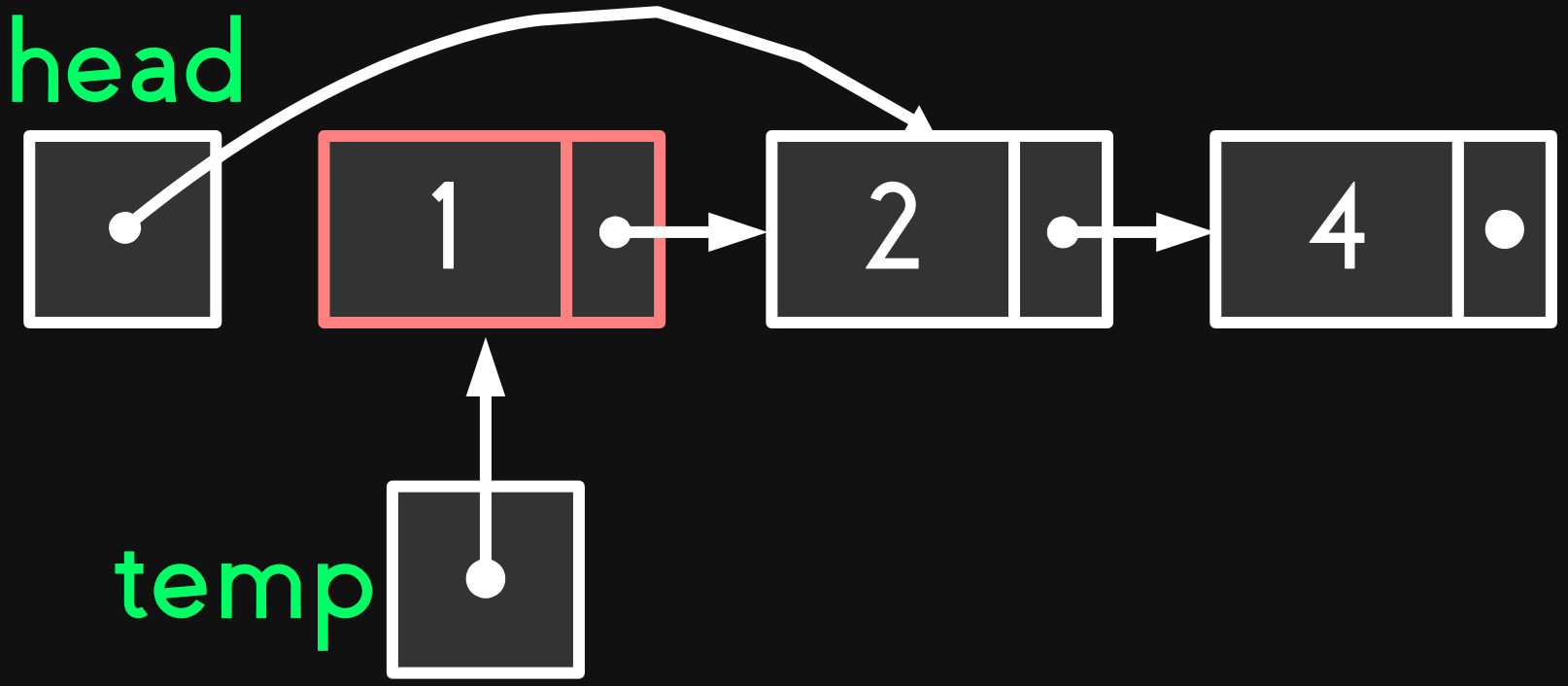


point head to the node after the
node to be deleted

Delete at Head

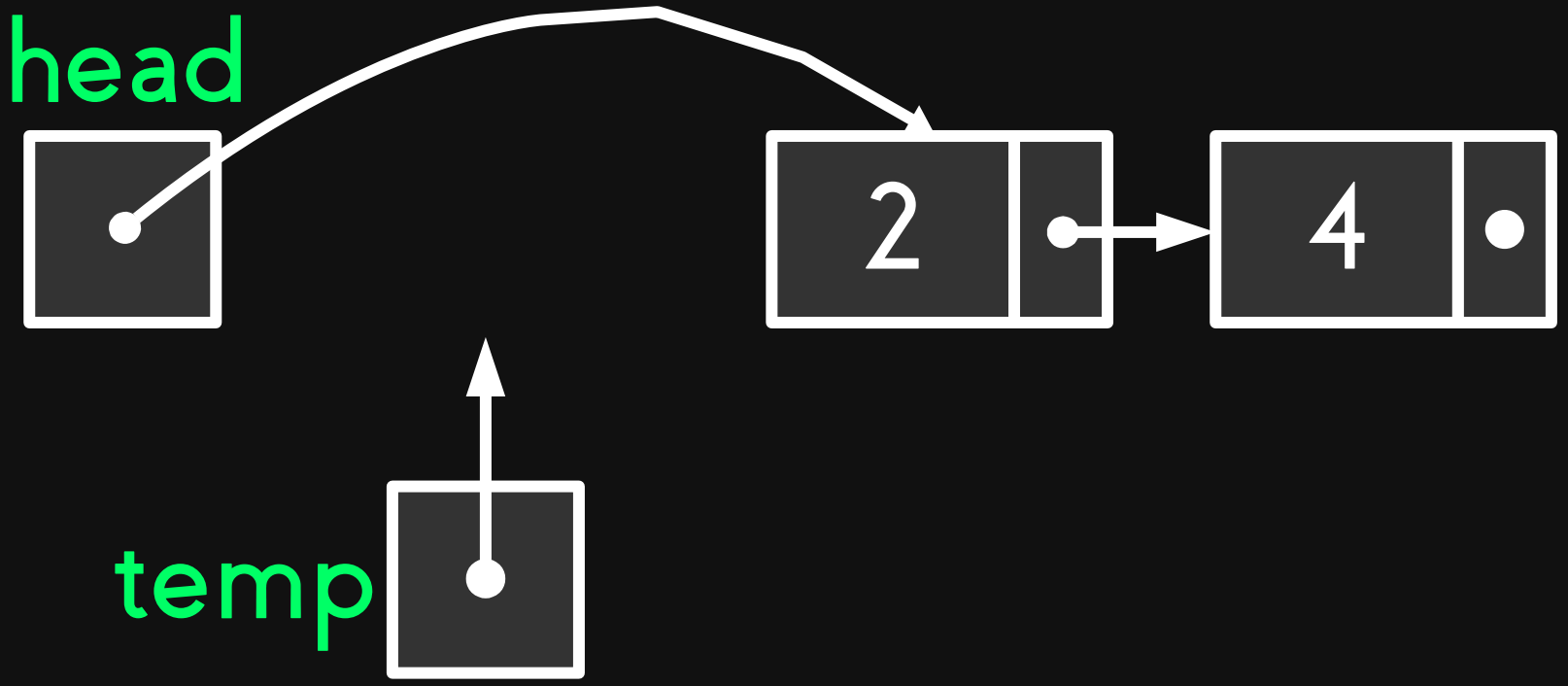


Delete at Head



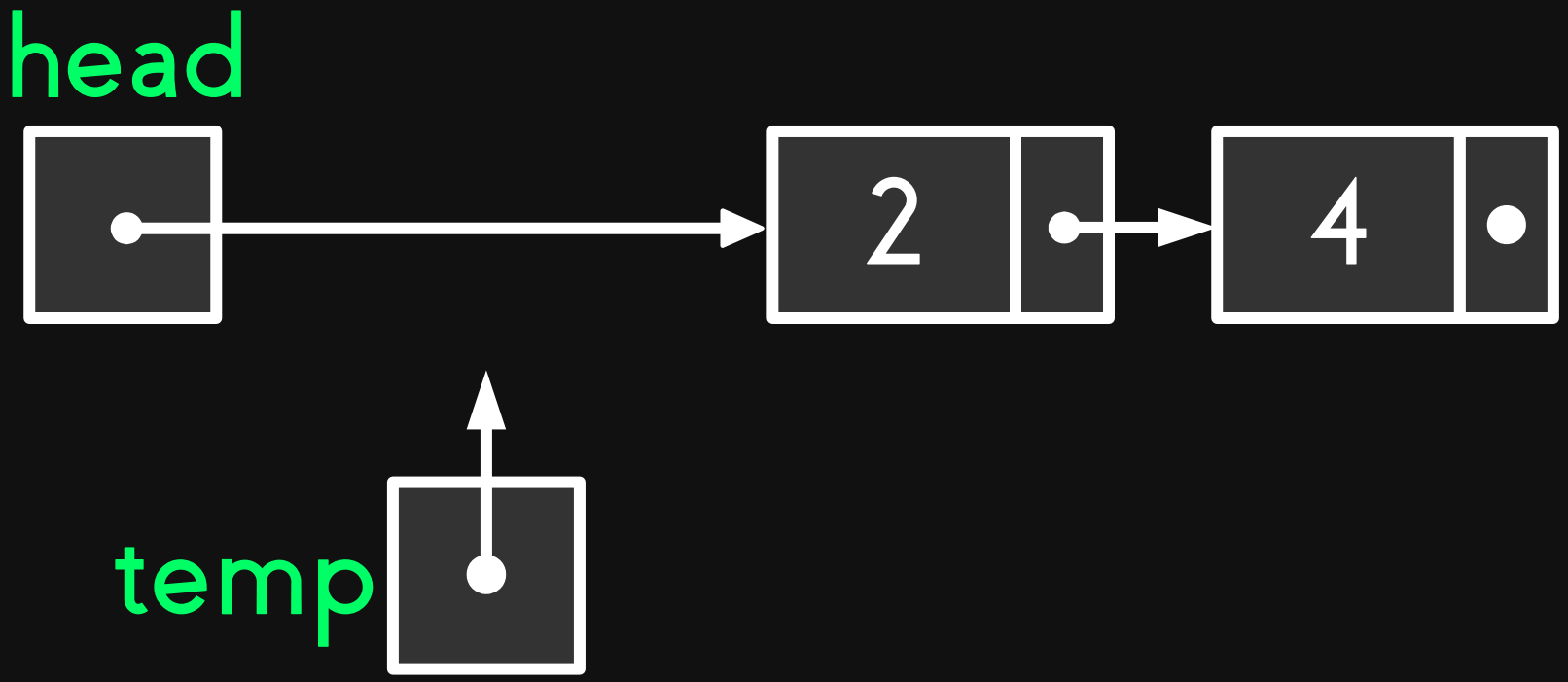
free the node being
pointed by temp

Delete at Head



free the node being
pointed by temp

Delete at Head



`temp` is now a dangling pointer.

Delete at Middle

delete a node that
is **in between two nodes**
in the linked list

```
struct NODE{
   int num;
   struct NODE *next;
};
```

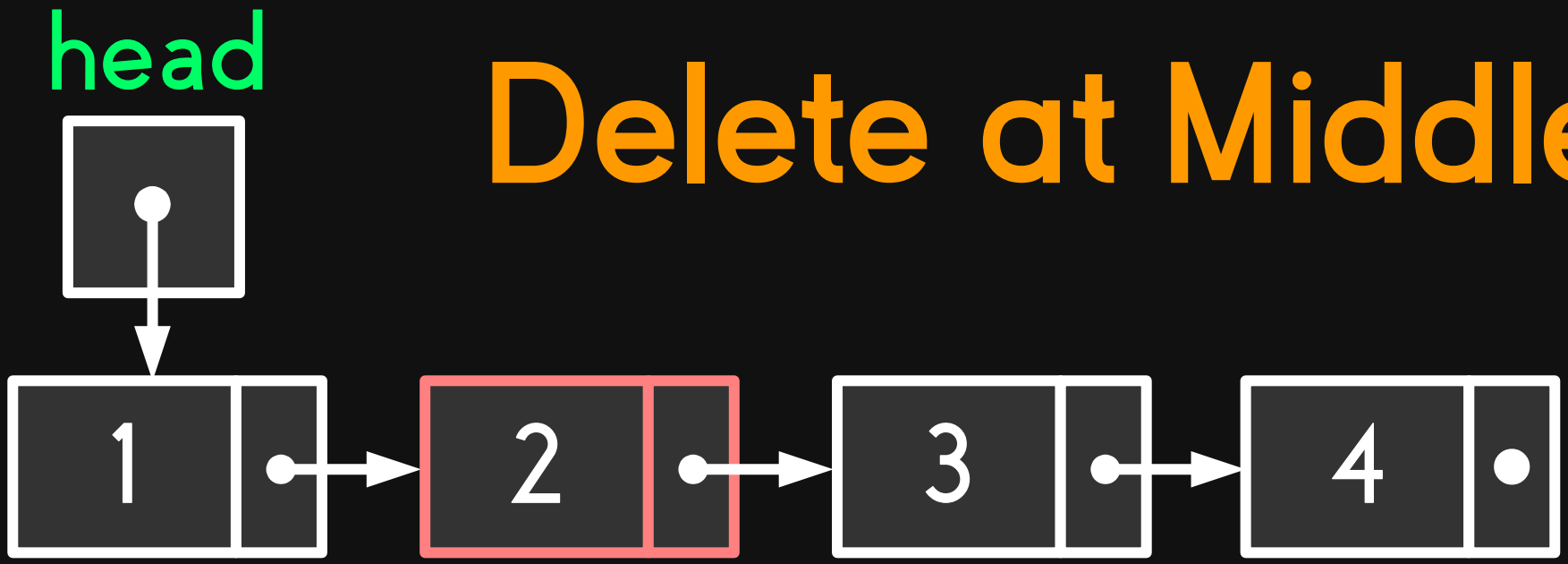
Delete at Middle

delete the node containing 3.

Delete at Middle

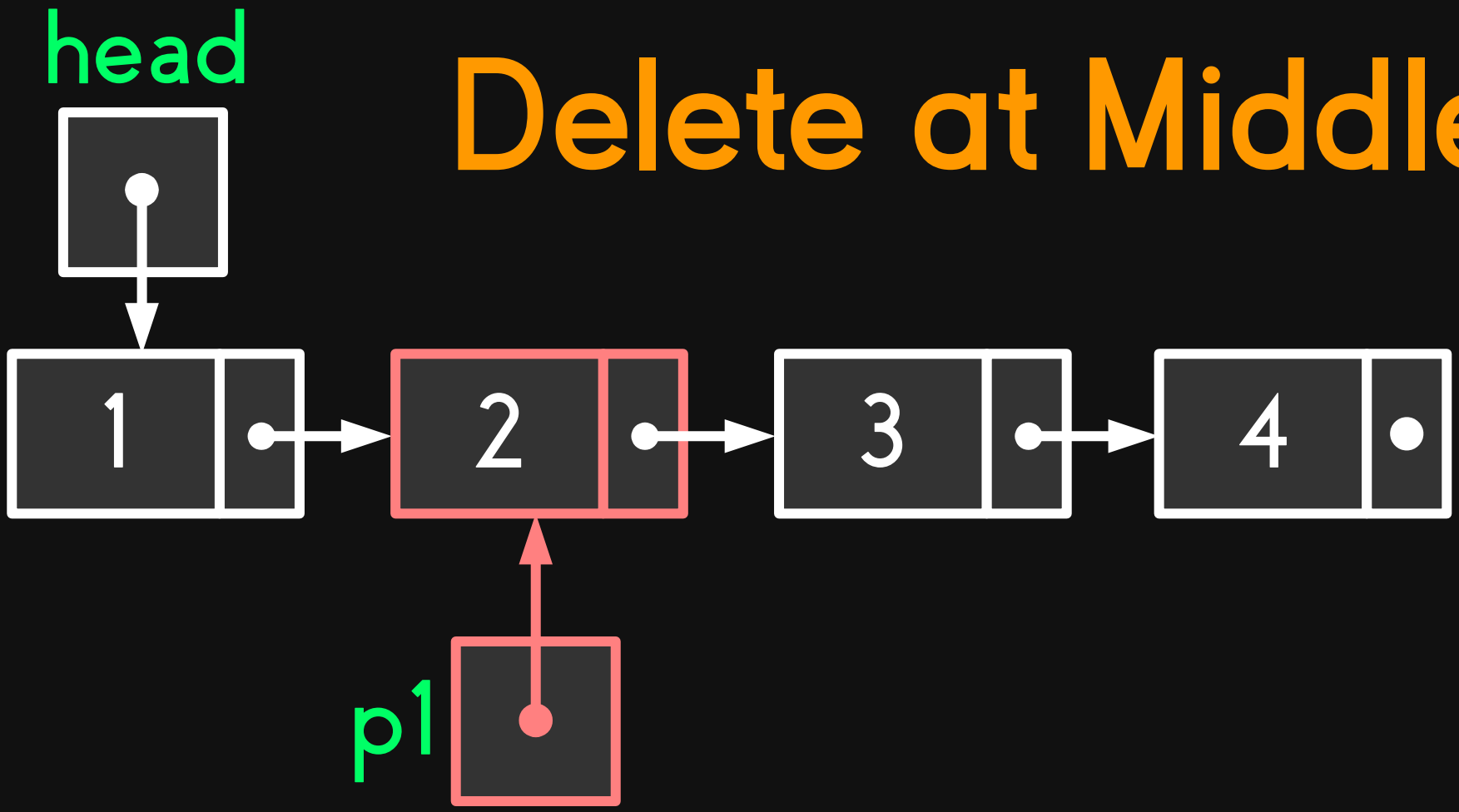
find the node before the node to be deleted.

Delete at Middle



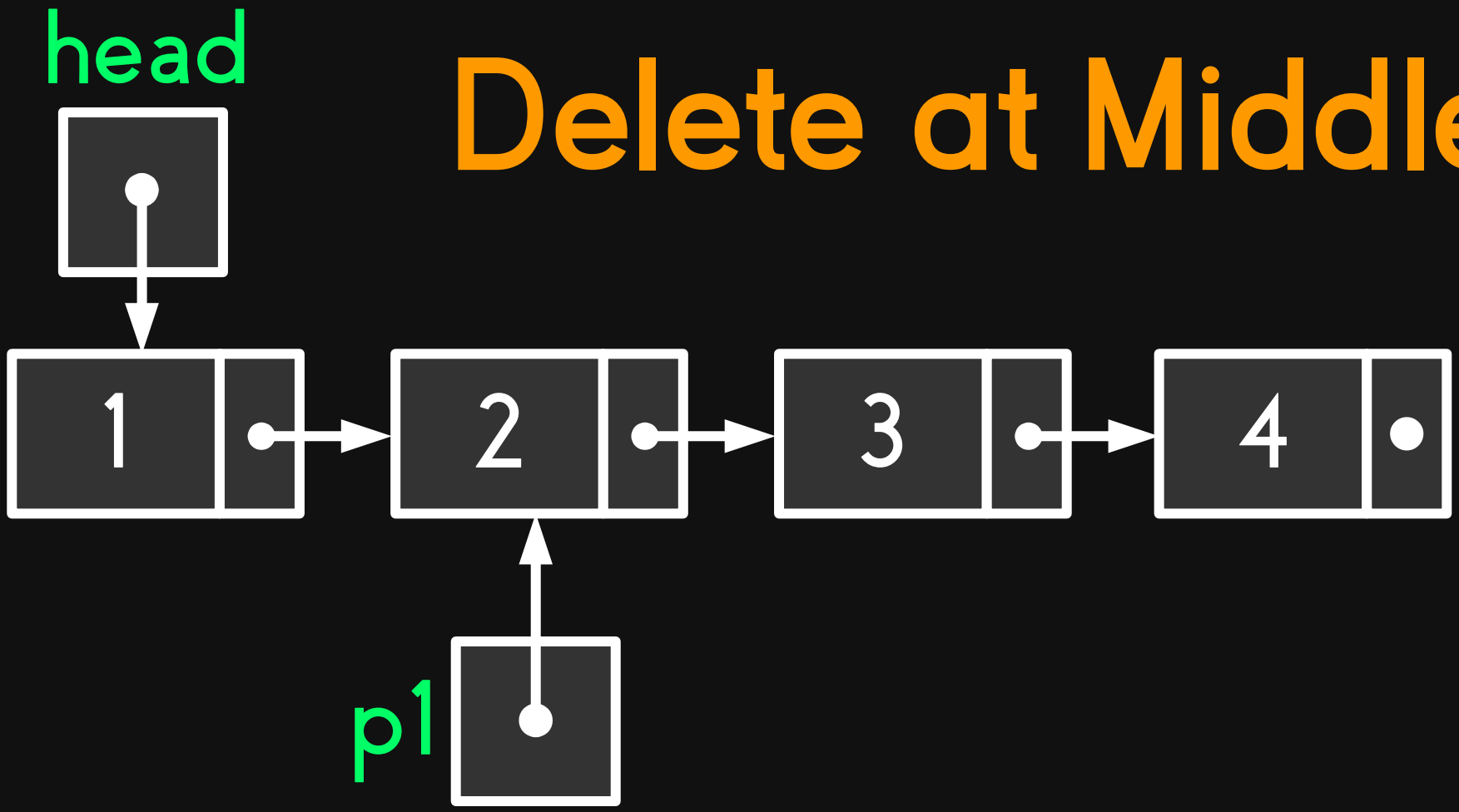
find the node before the node to be deleted.

Delete at Middle



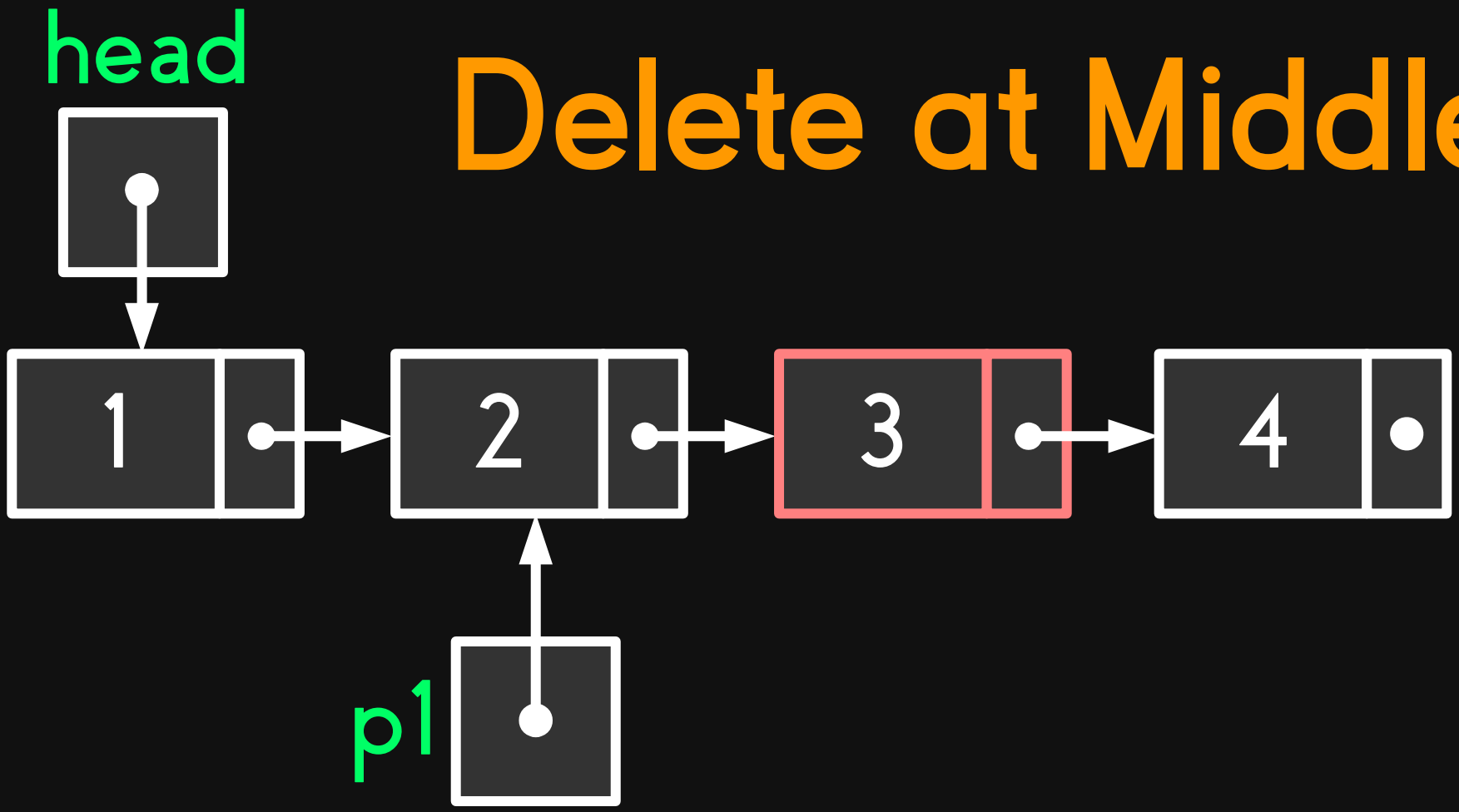
point a pointer (p1) to the node
before the node to be deleted.

Delete at Middle



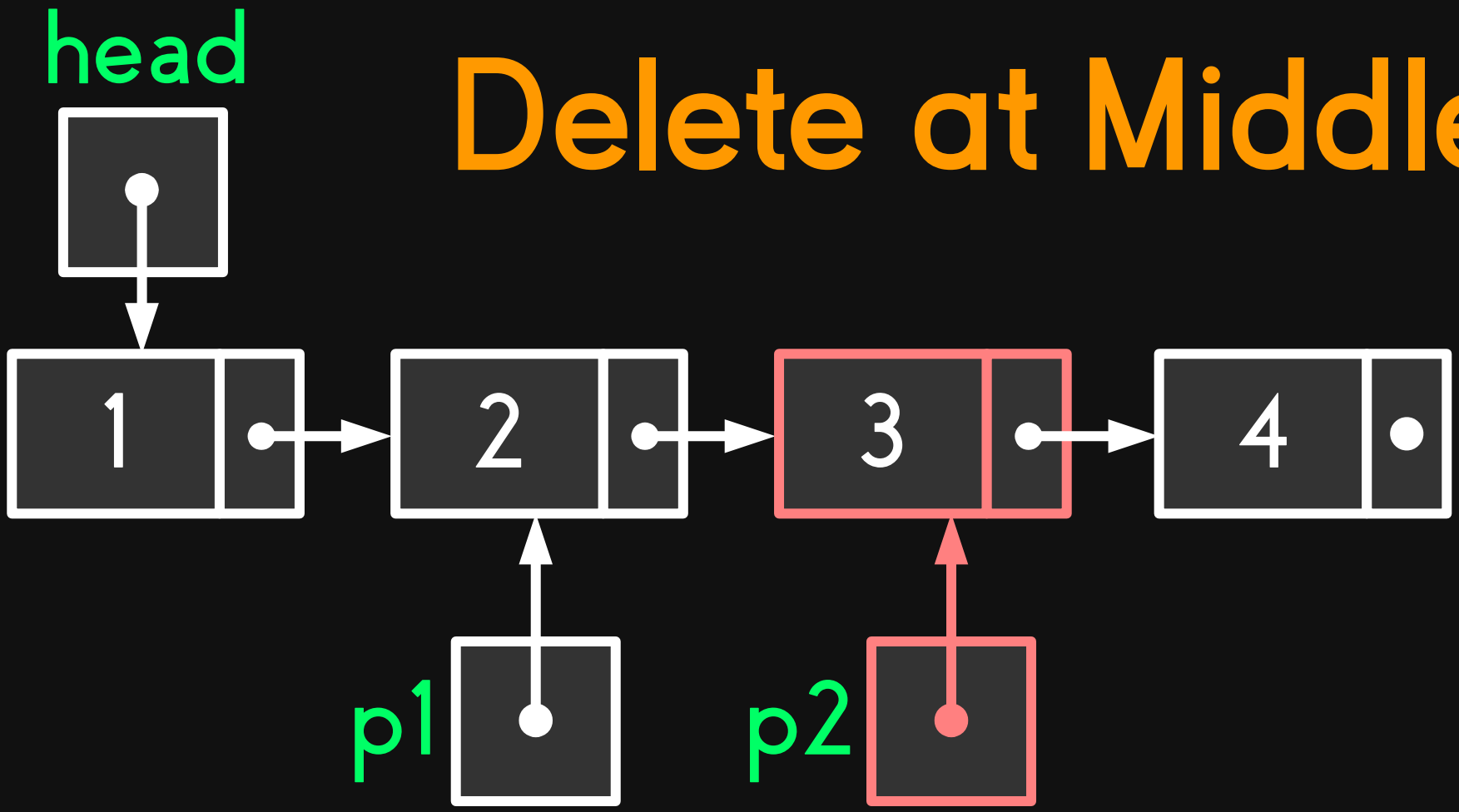
point another pointer (p2) to the node to be deleted.

Delete at Middle



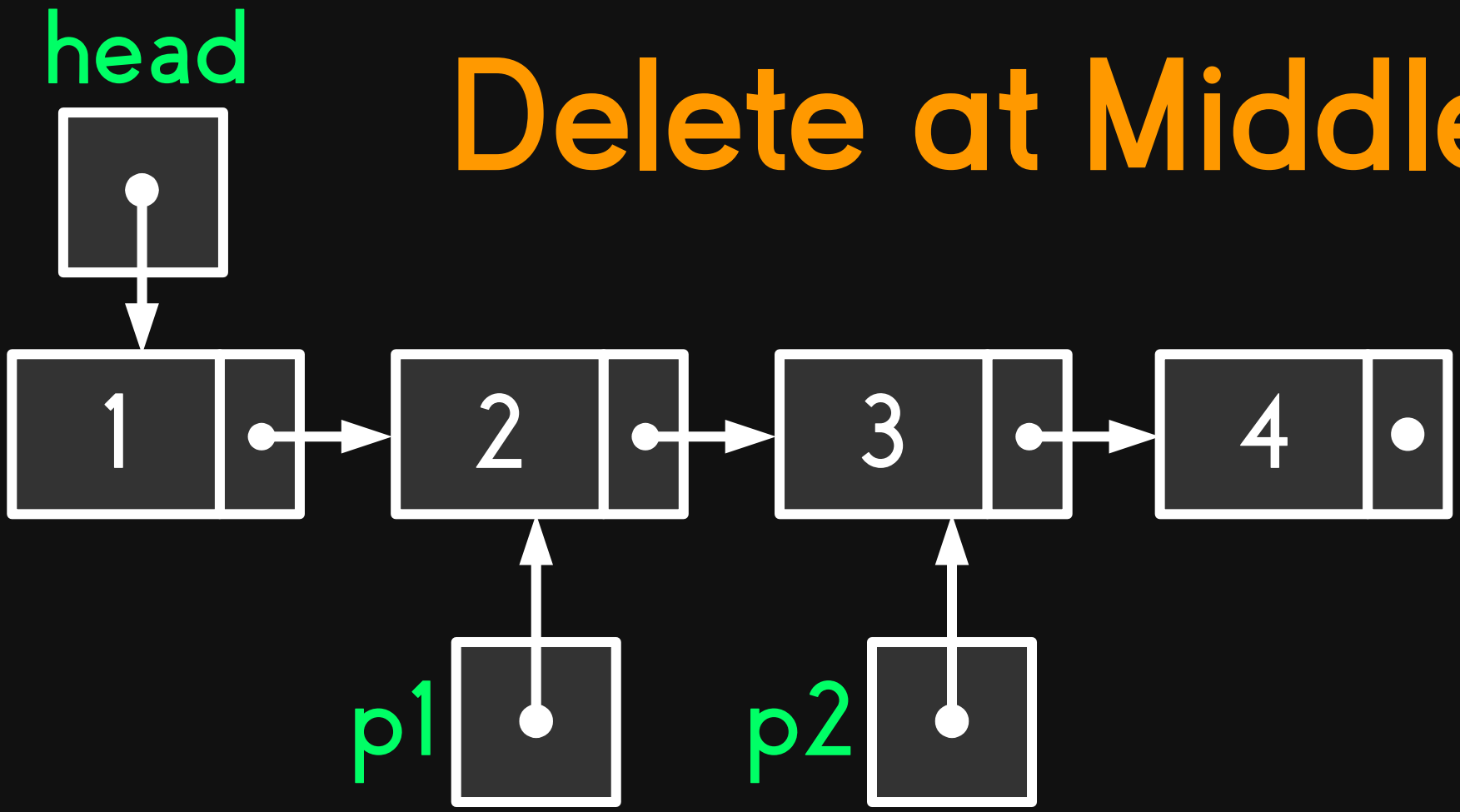
point another pointer (p2) to the node to be deleted.

Delete at Middle

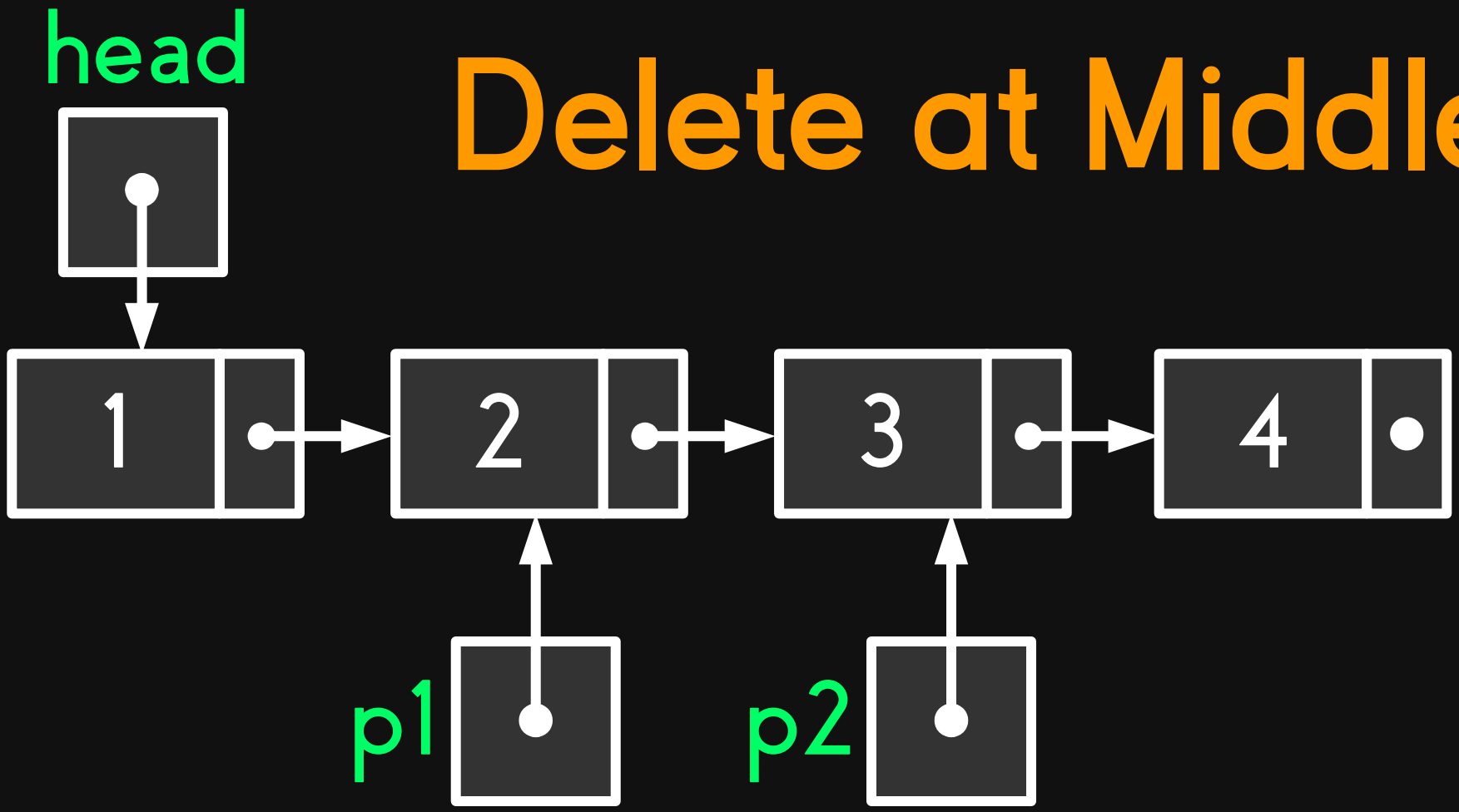


point another pointer (p2) to the node to be deleted.

Delete at Middle

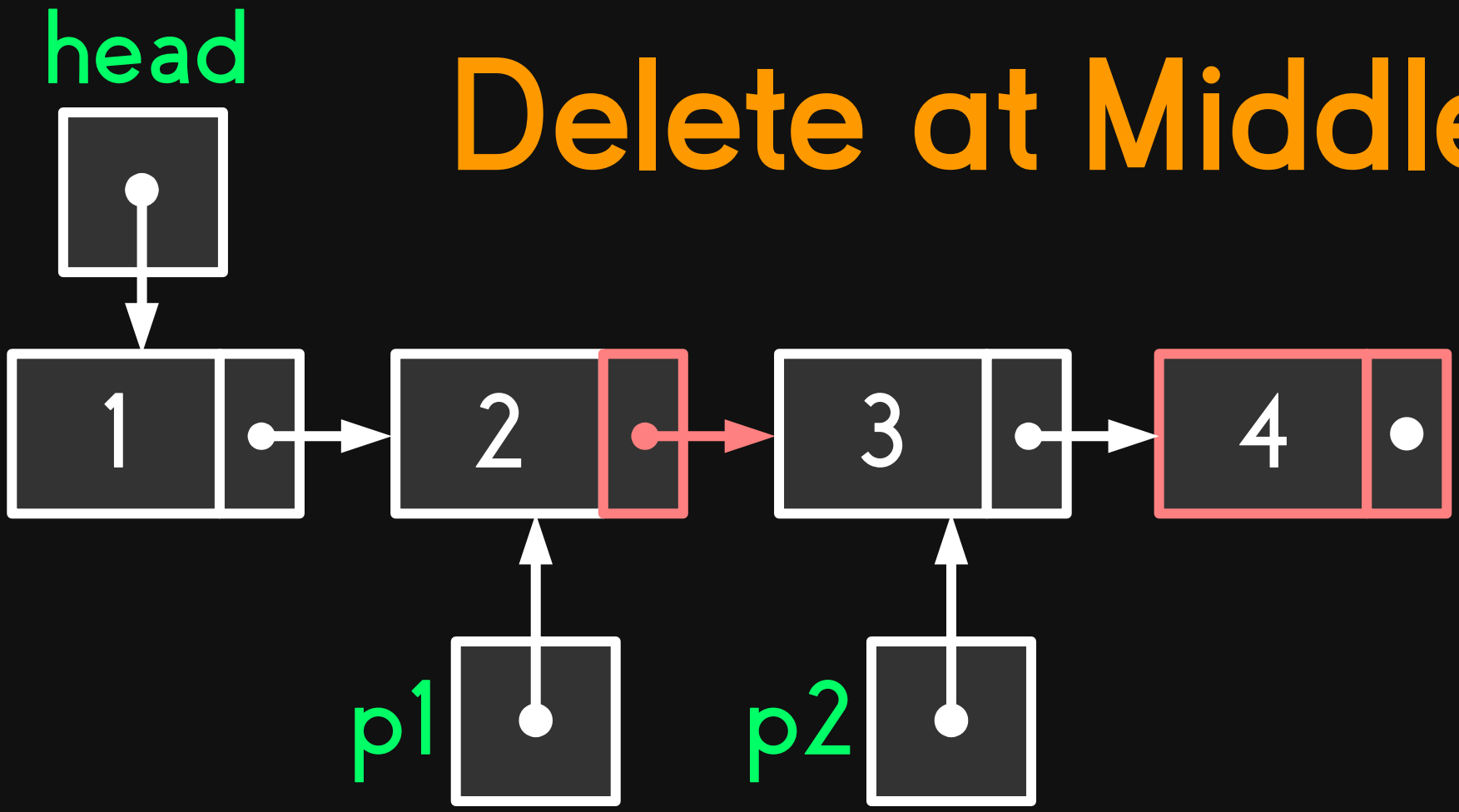


Delete at Middle



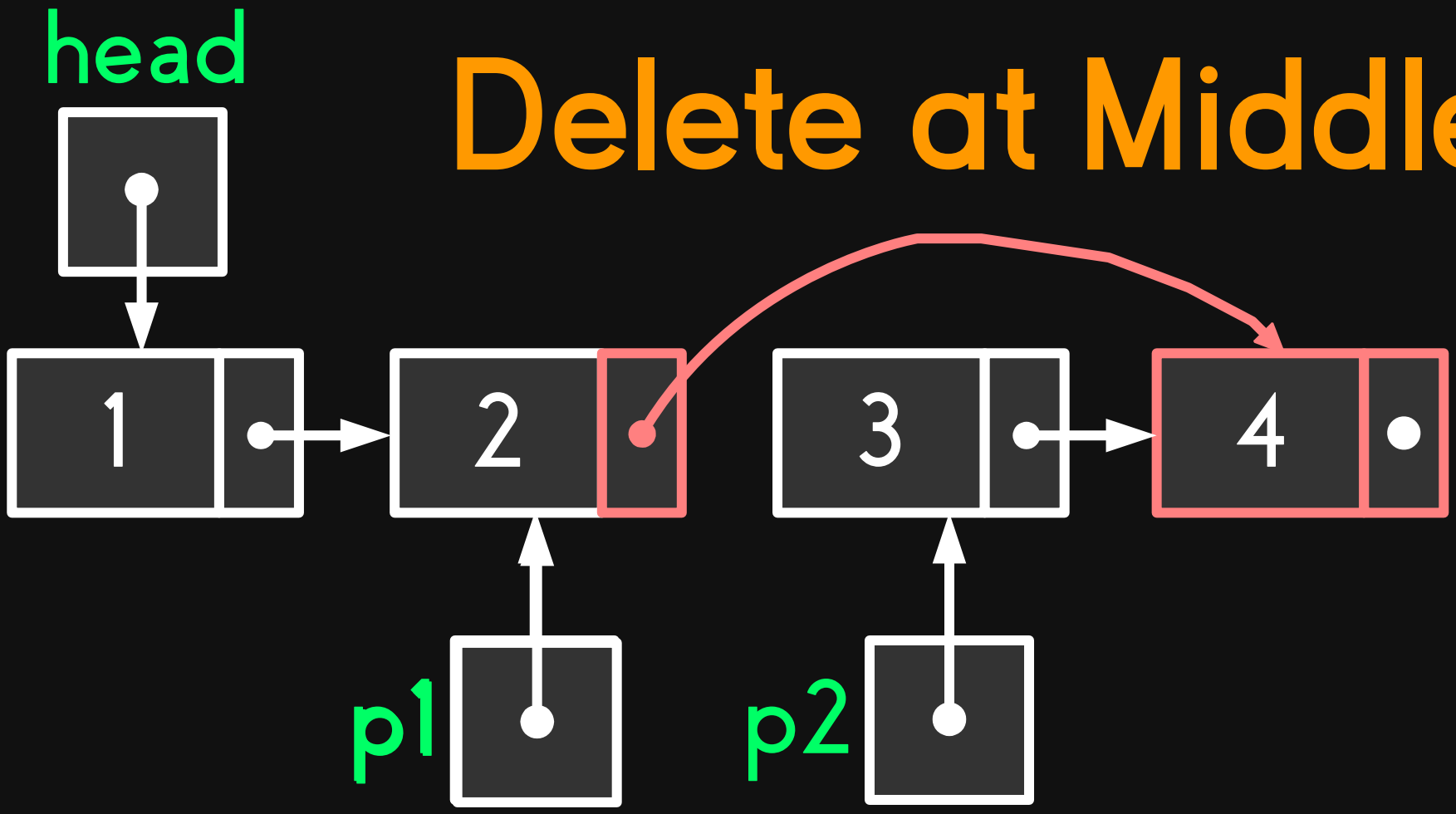
point the next pointer of the node being pointed by **p1** to the node after the node to be deleted

Delete at Middle



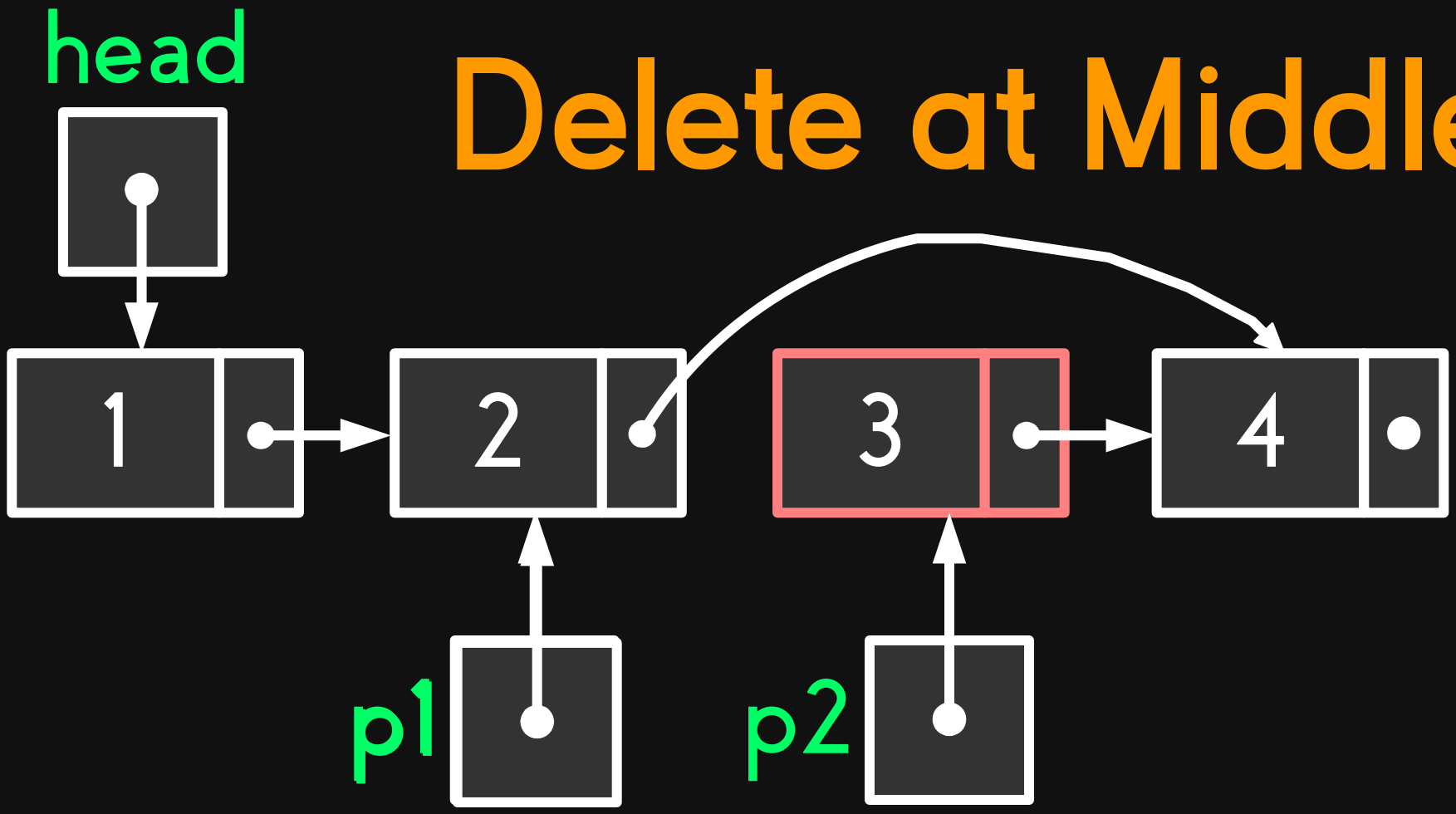
point the next pointer of the node
being pointed by **p1** to the node after the
node to be deleted

Delete at Middle



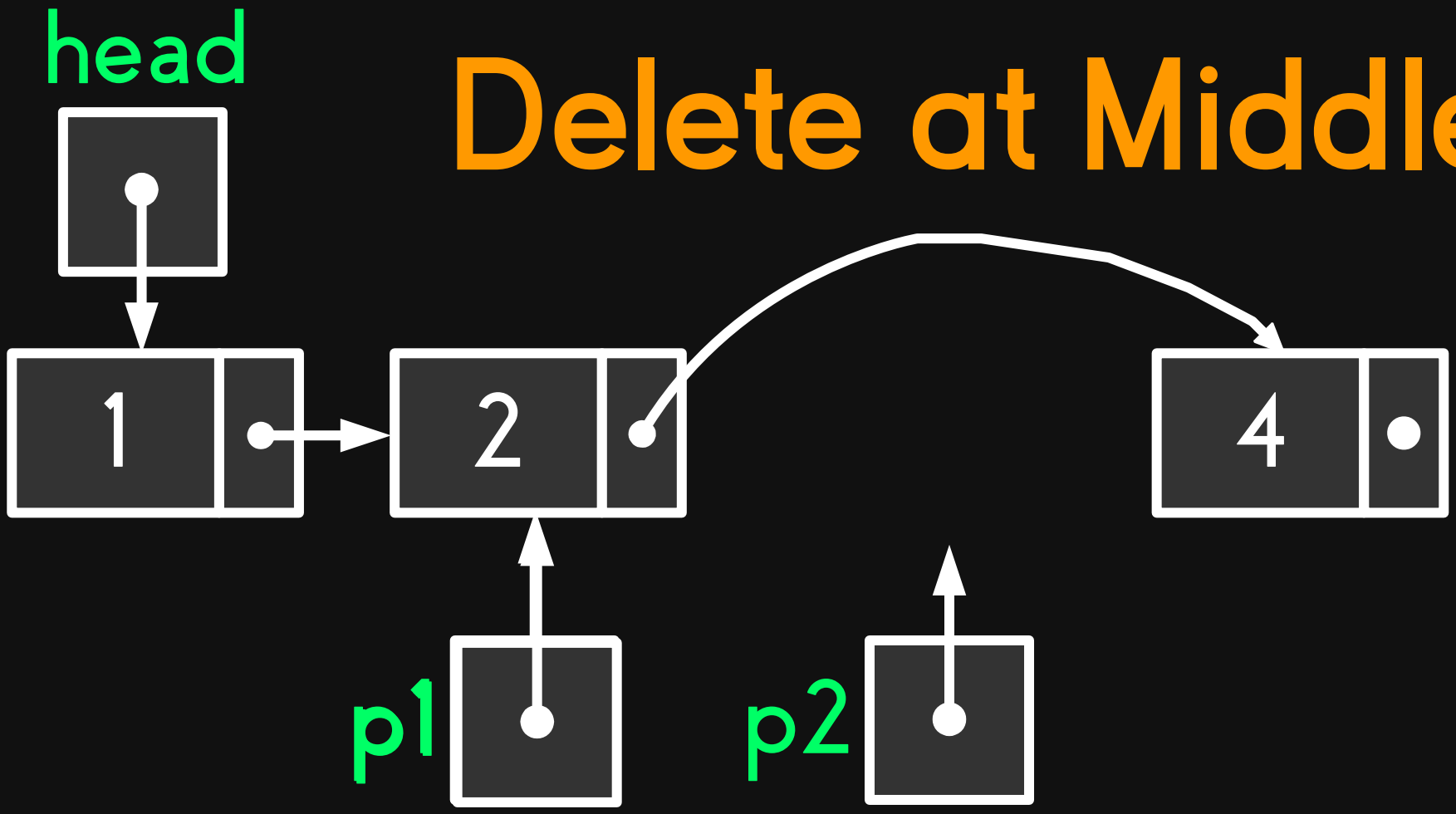
point the next pointer of the node being pointed by **p1** to the node after the node to be deleted

Delete at Middle



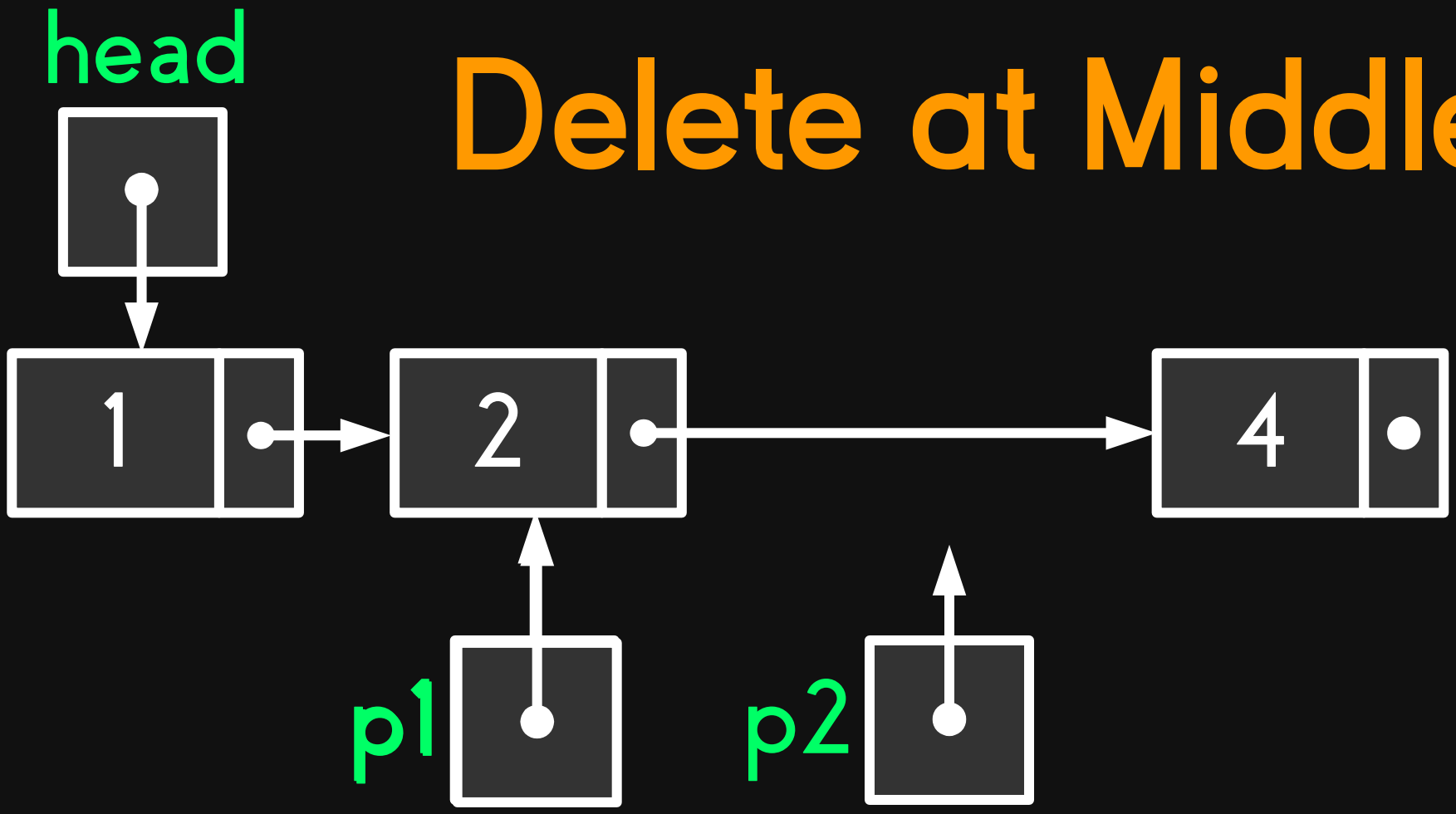
free the node being pointed by p2.

Delete at Middle



free the node being pointed by p2.

Delete at Middle



p2 is now a dangling pointer.

Delete at Tail

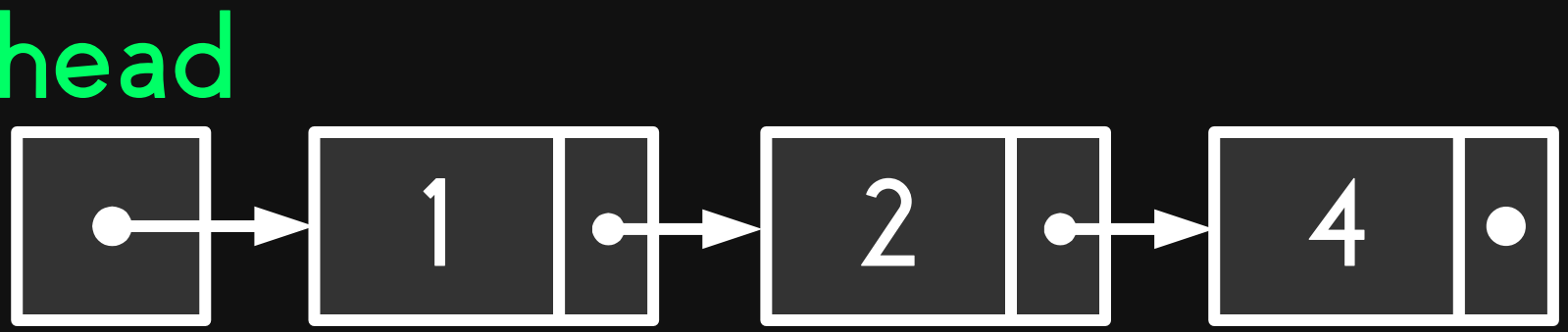
delete the
last element
of the list

Delete at Tail

just like in insert.
delete at tail is **a special case of delete at middle**

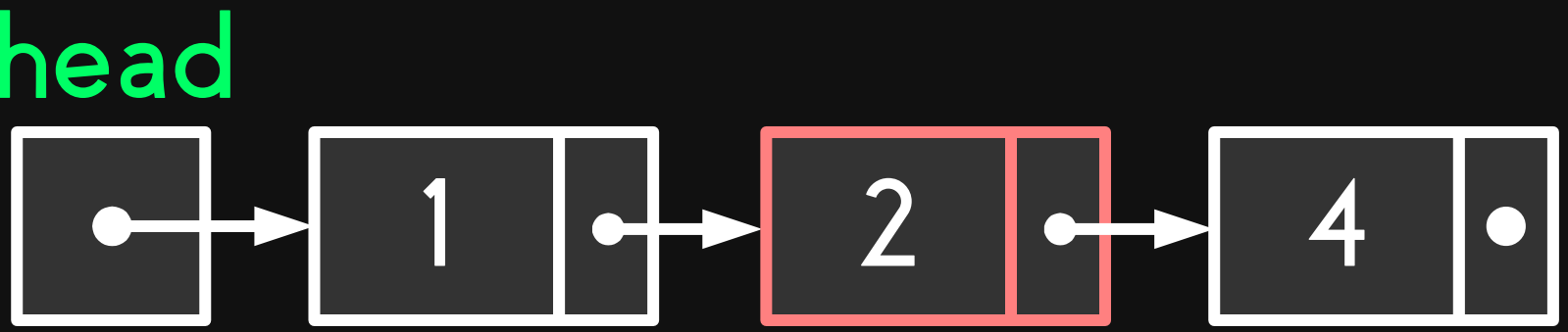
```
struct NODE{
   int num;
   struct NODE *next;
};
```

Delete at Tail



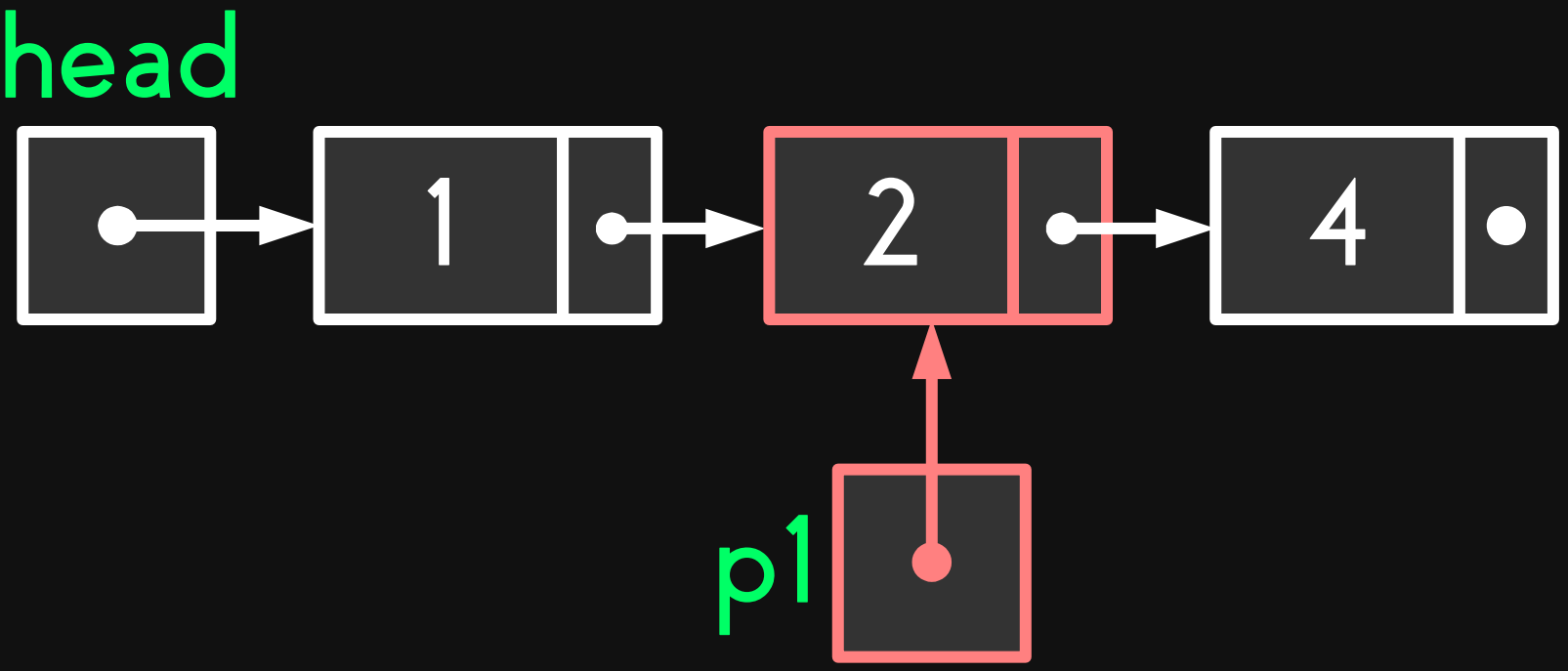
delete the last element (4)

Delete at Tail



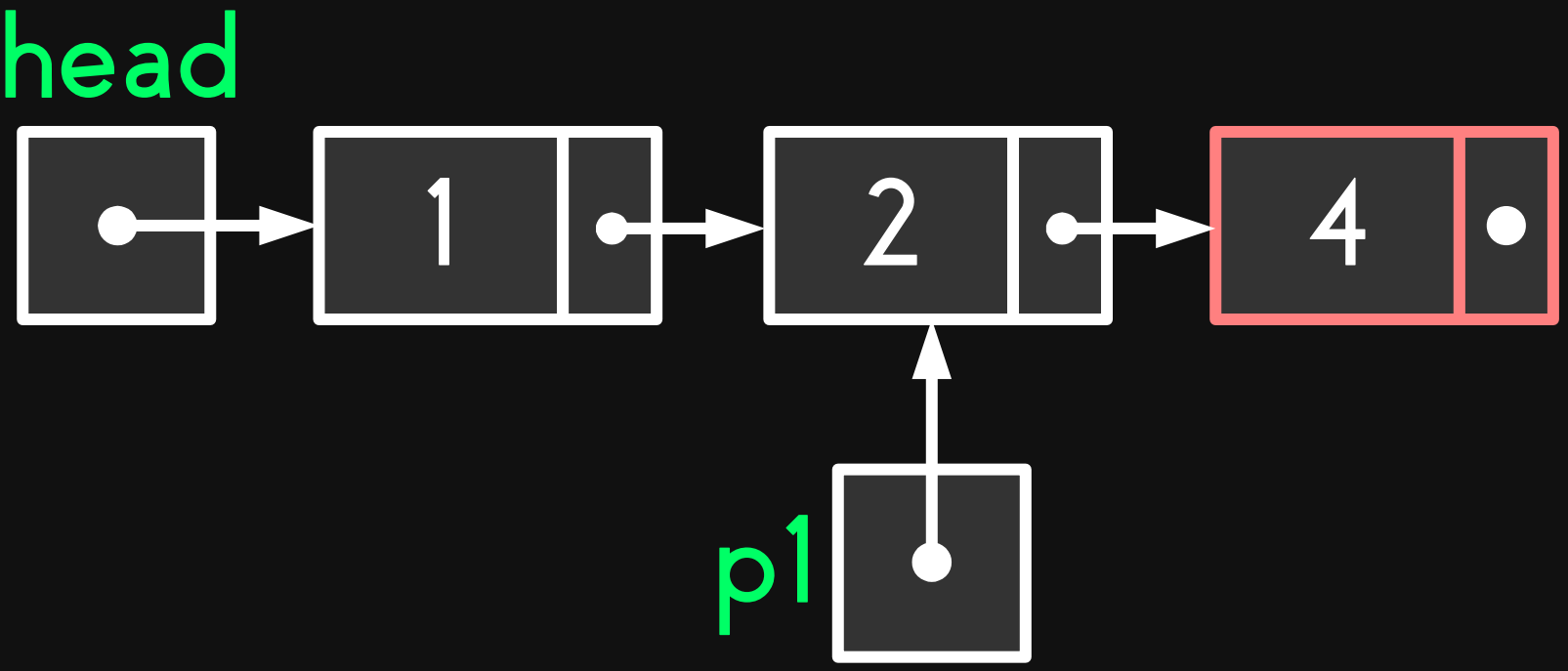
make a pointer (p1) point to the
node before the tail node

Delete at Tail



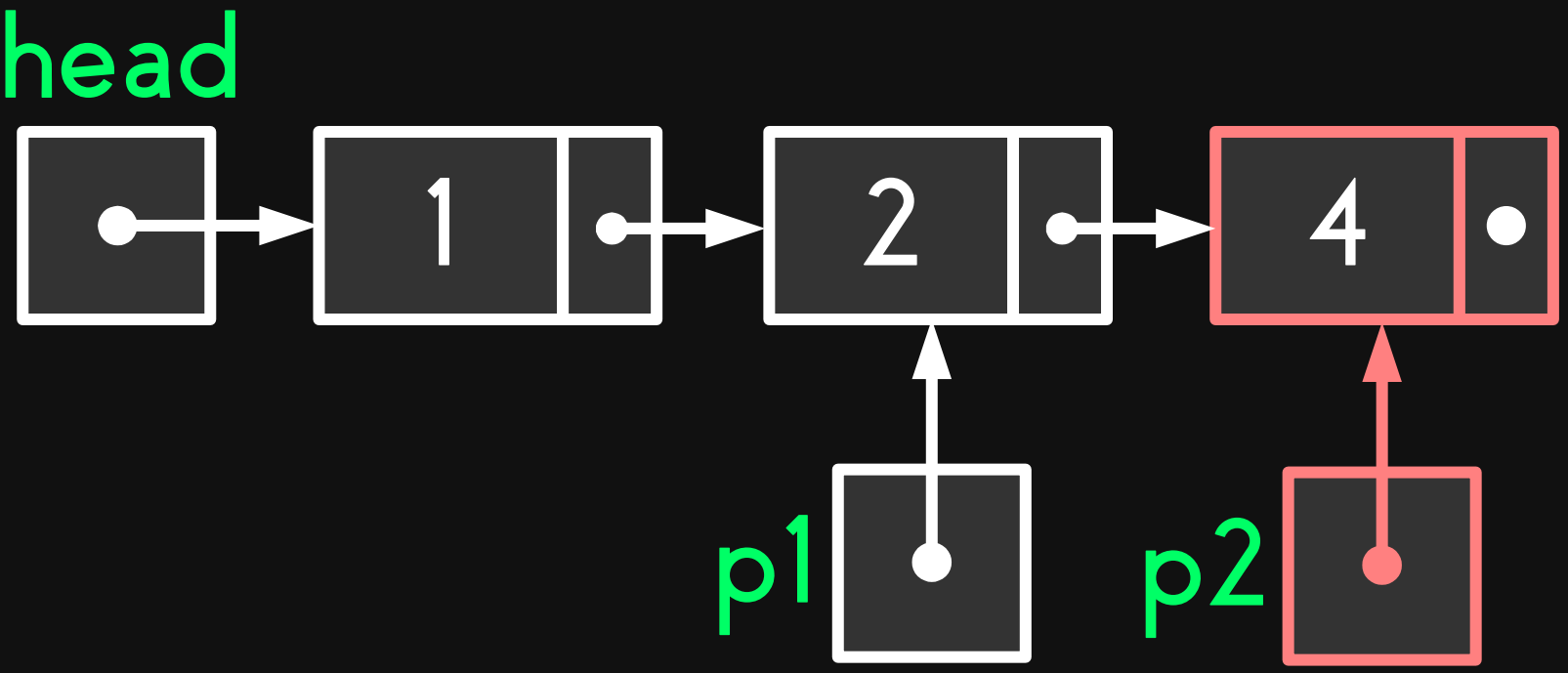
make a pointer (p1) point to the
node before the tail node

Delete at Tail



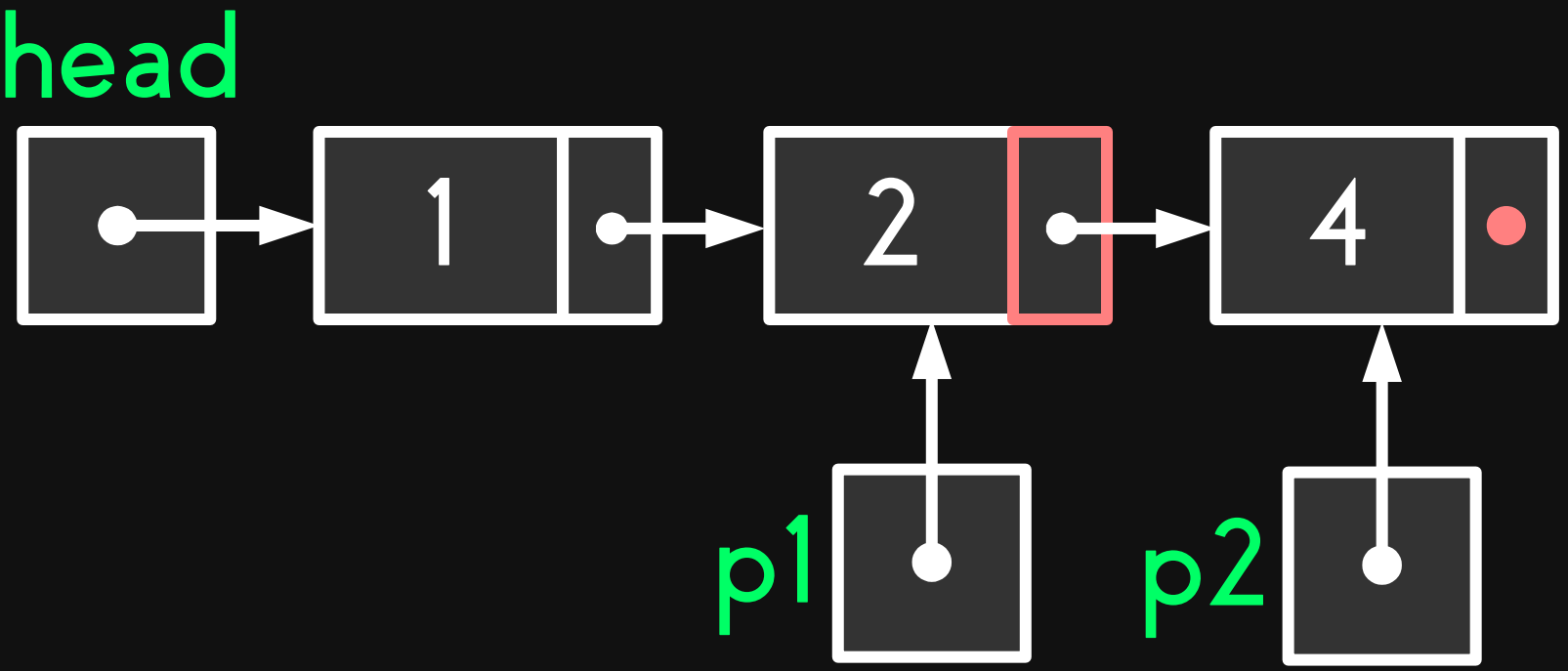
make a pointer (p2) point to the last node

Delete at Tail



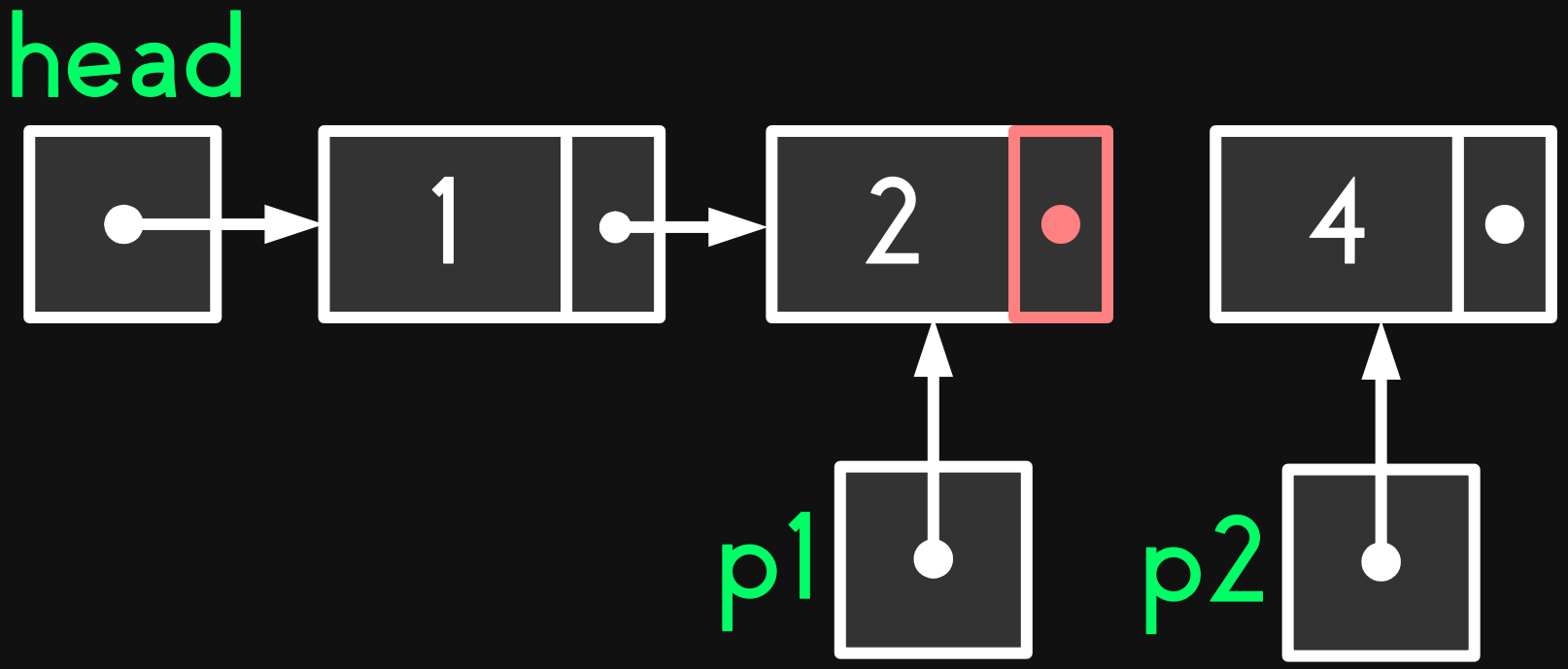
make a pointer (p2) point to the last node

Delete at Tail



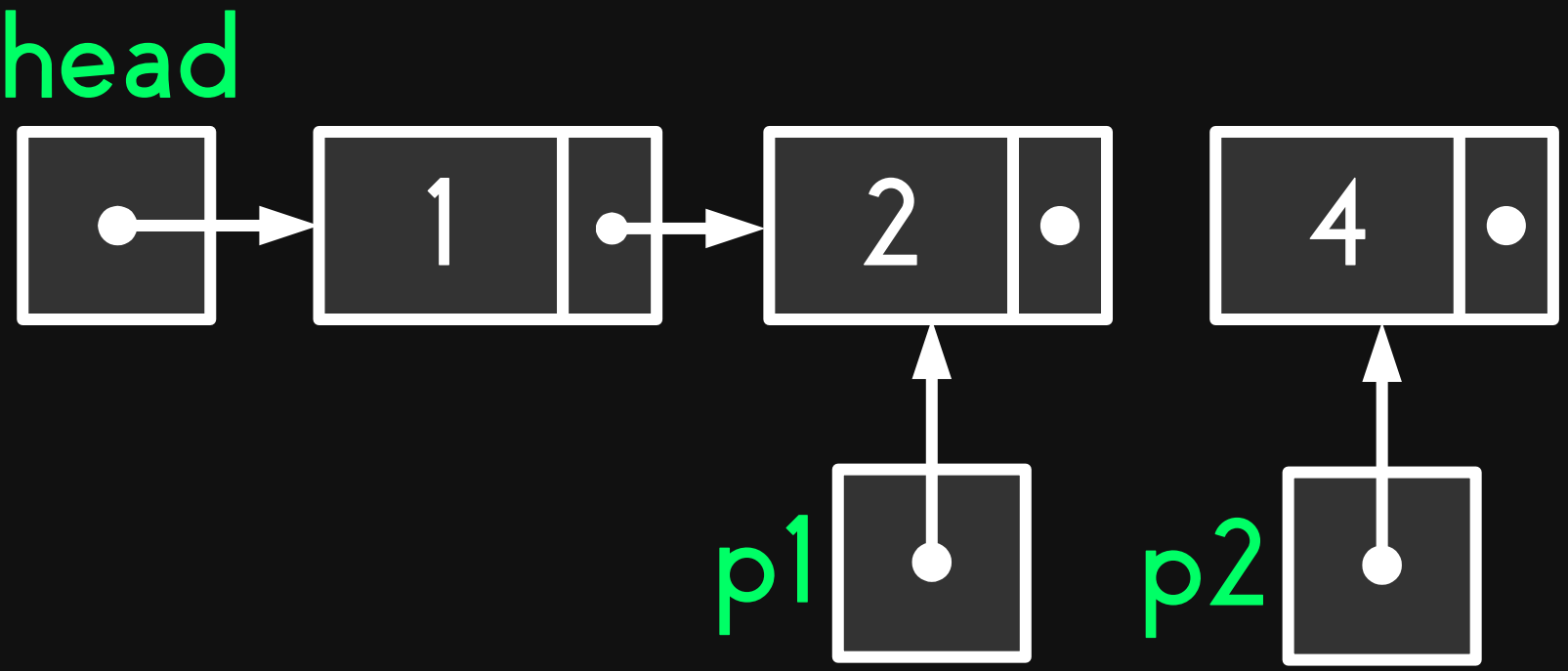
point the next pointer of the node
being pointed by **p1** to the node
after the node to be deleted

Delete at Tail



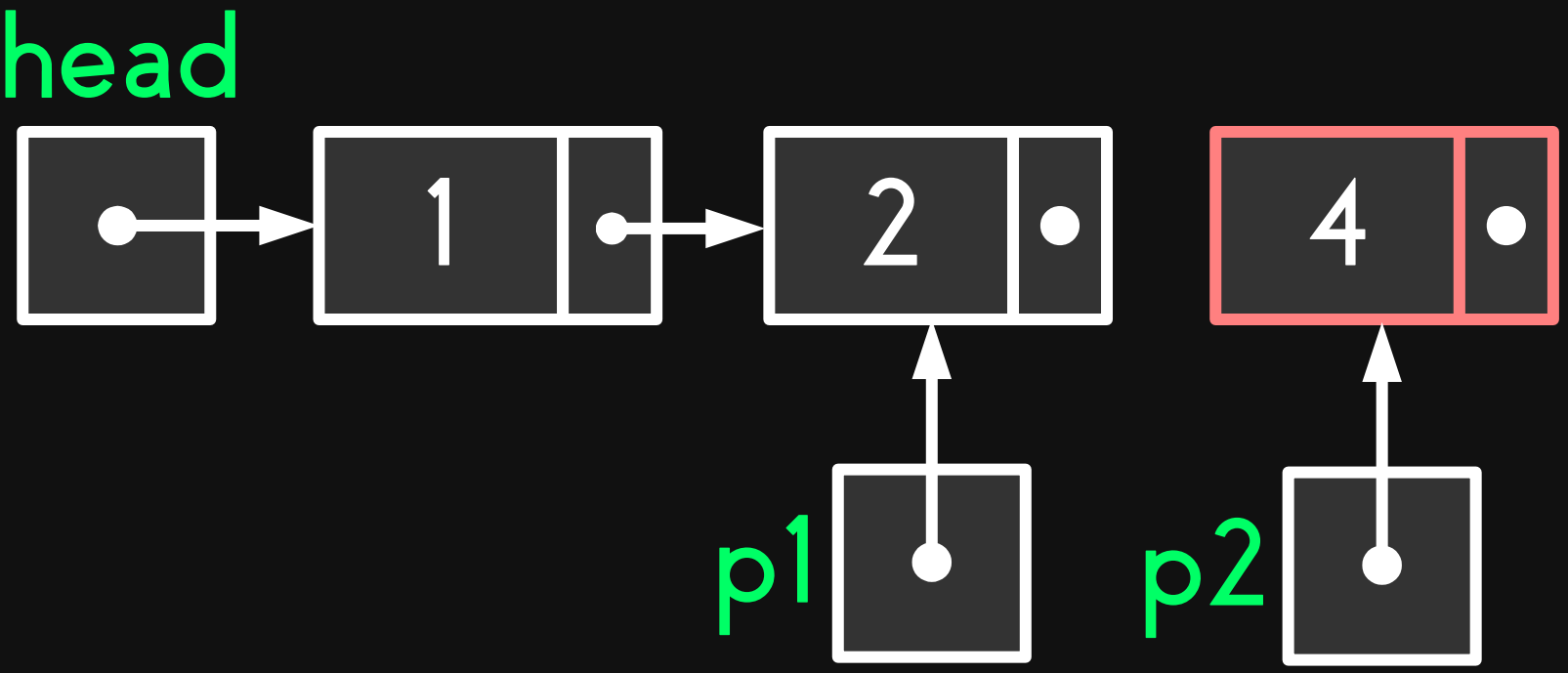
point the next pointer of the node
being pointed by **p1** to the node
after the node to be deleted

Delete at Tail



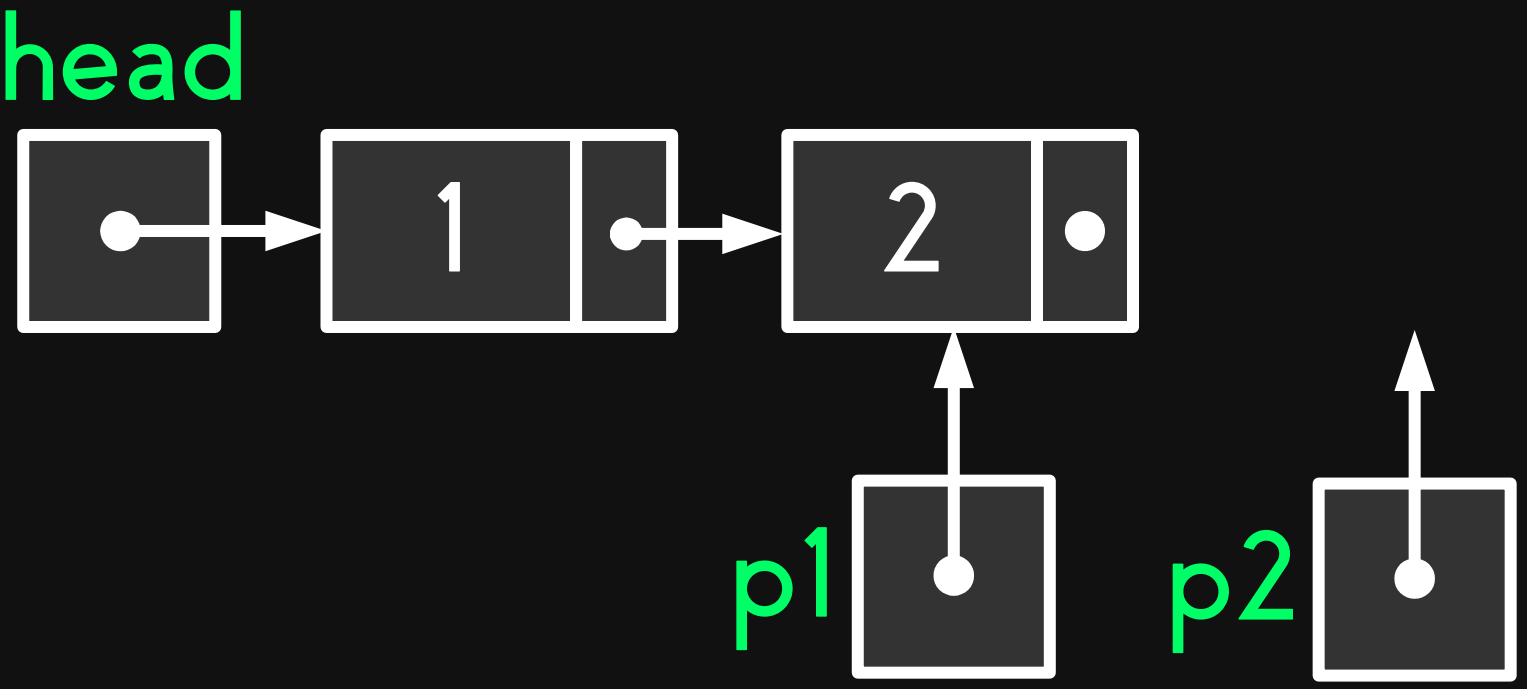
delete the node being pointed by
`p2` using the `free()` function

Delete at Tail



delete the node being pointed by
`p2` using the `free()` function

Delete at Tail



p2 is now a dangling pointer.

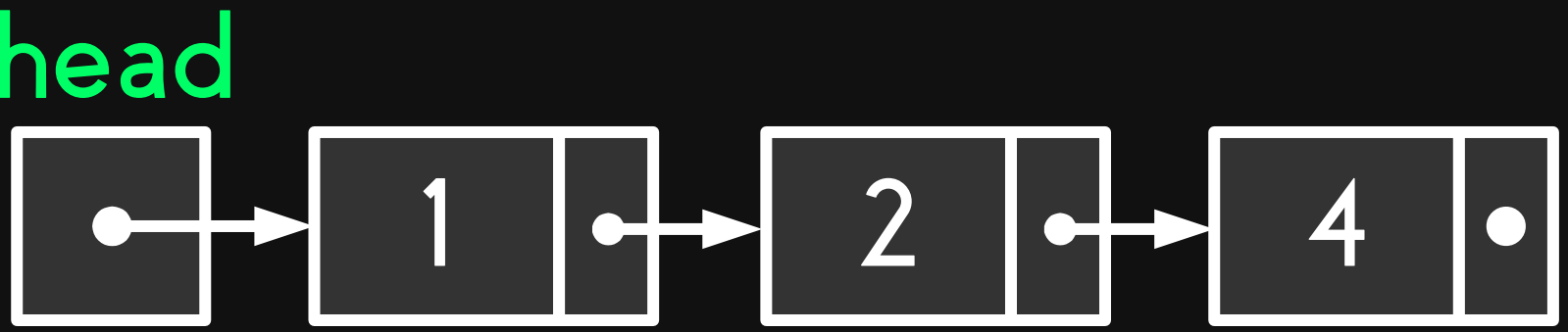
View

prints/shows the
details of the nodes
in a given linked list

View

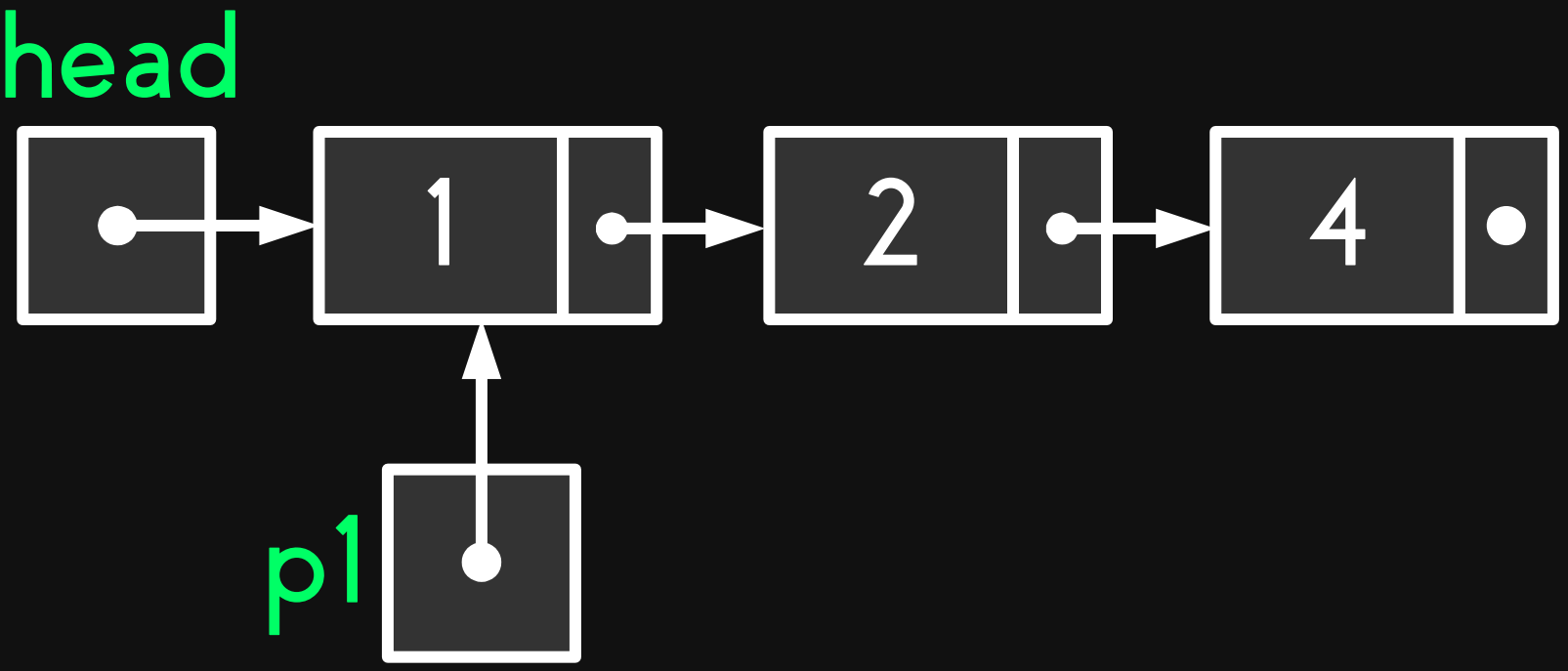
traverses the
linked list

View



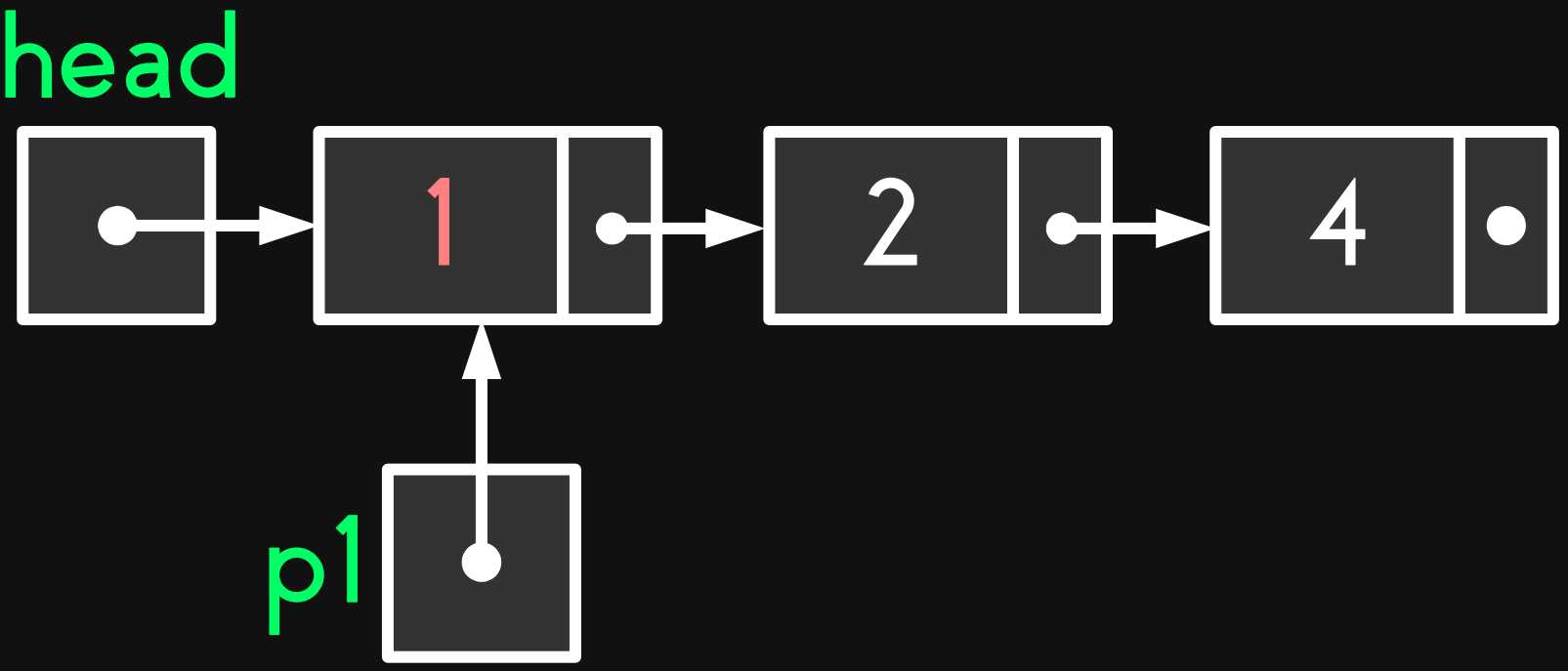
print all the details
of all the nodes

View



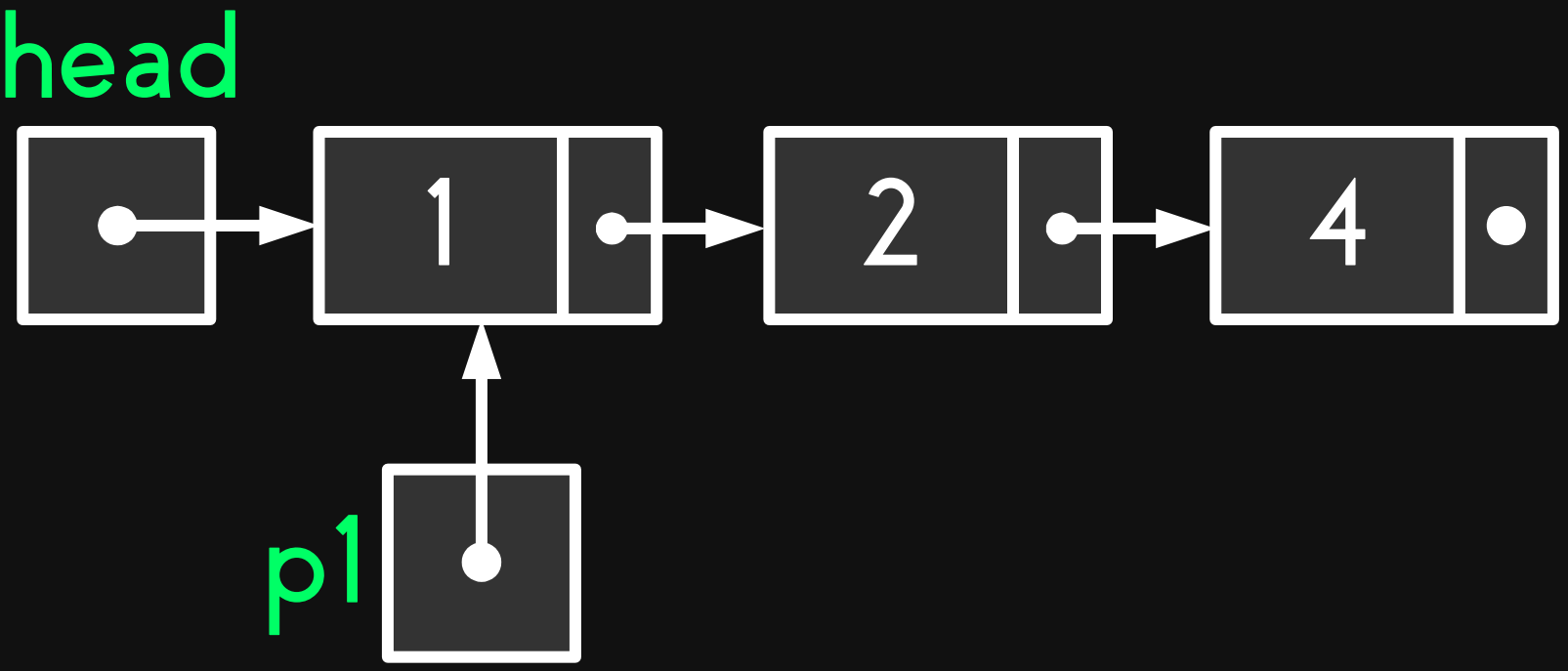
make a pointer point to the
head node

View



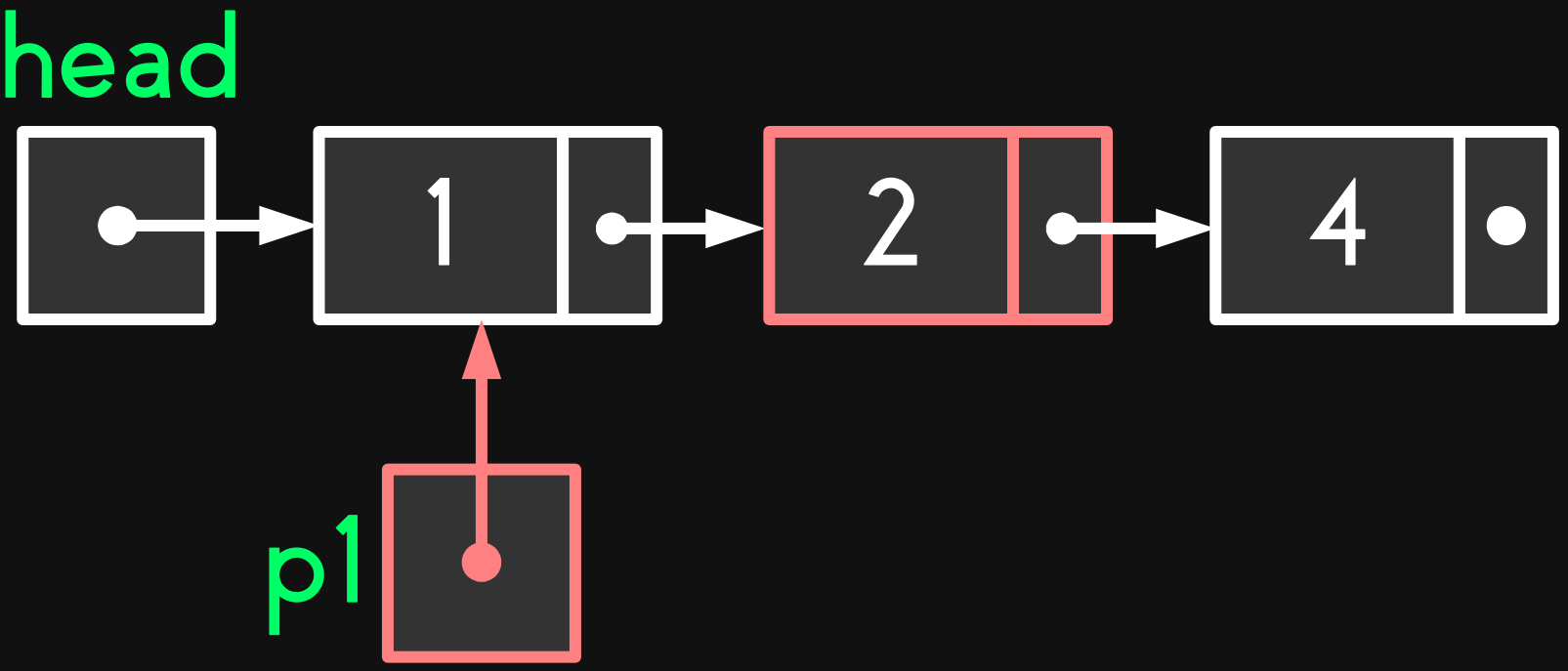
print the data in the node
being pointed by **p1**.

View



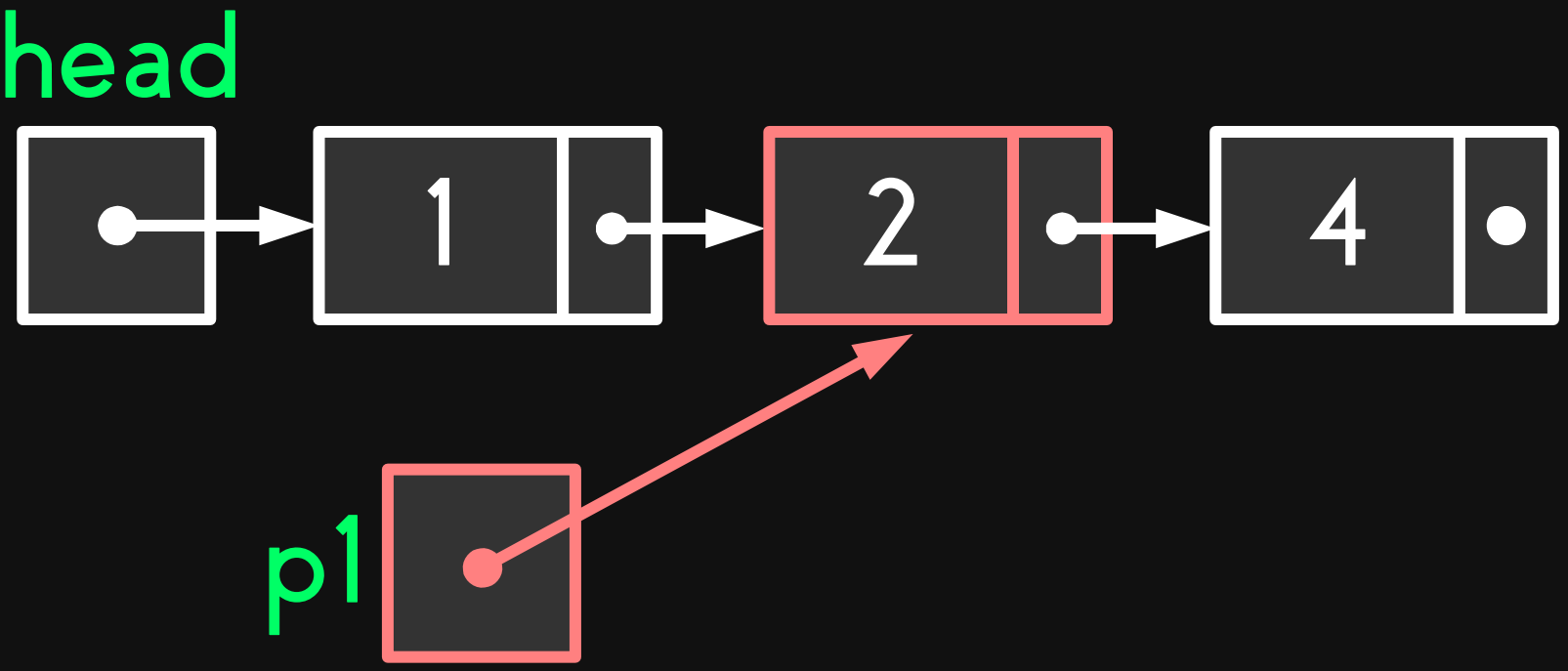
point p1 to the next node.

View



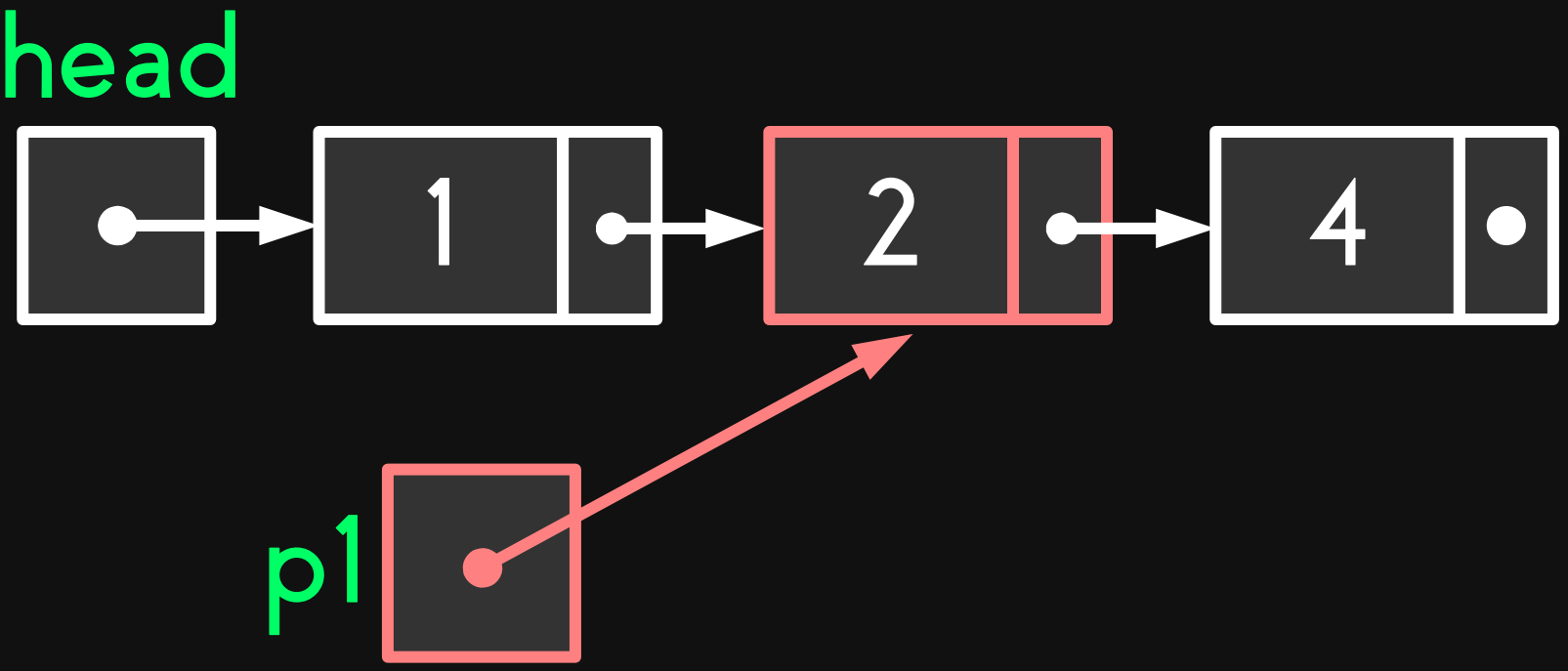
point p1 to the next node.

View



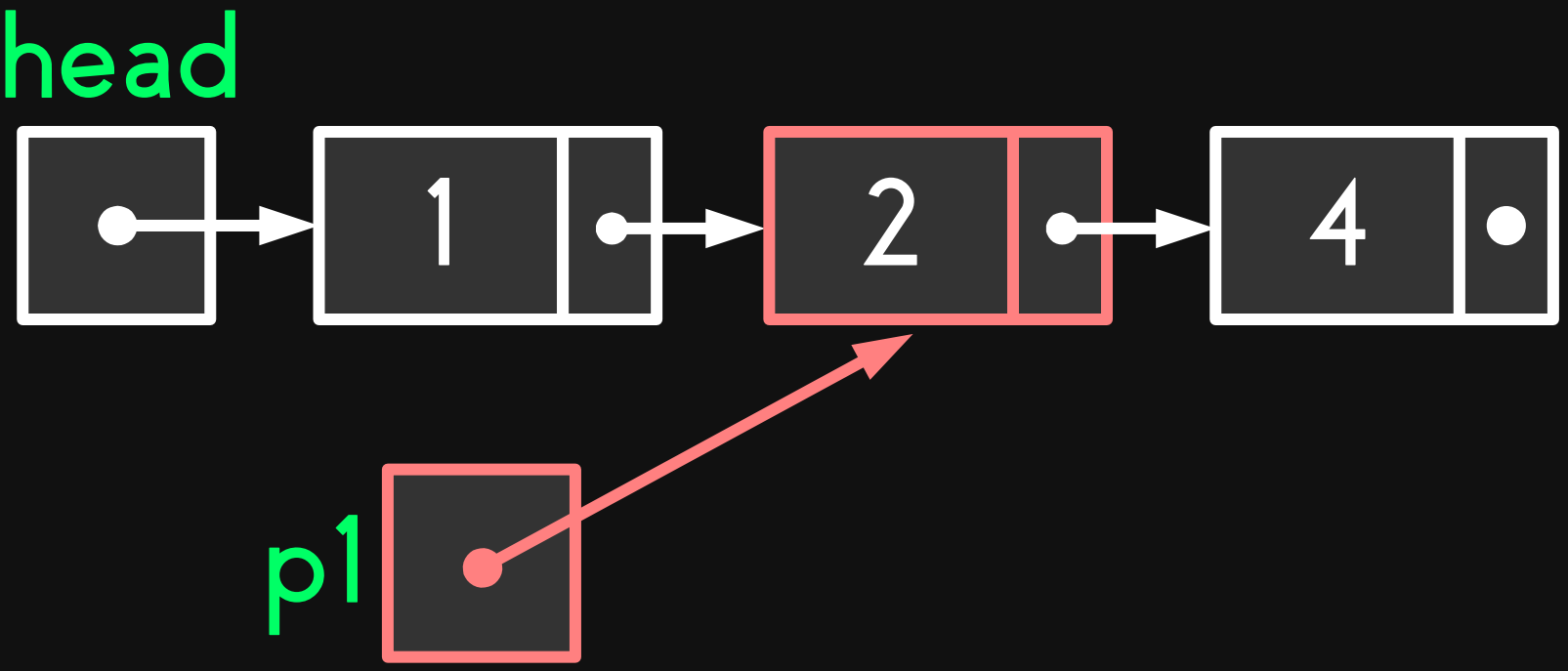
point p1 to the next node.

View



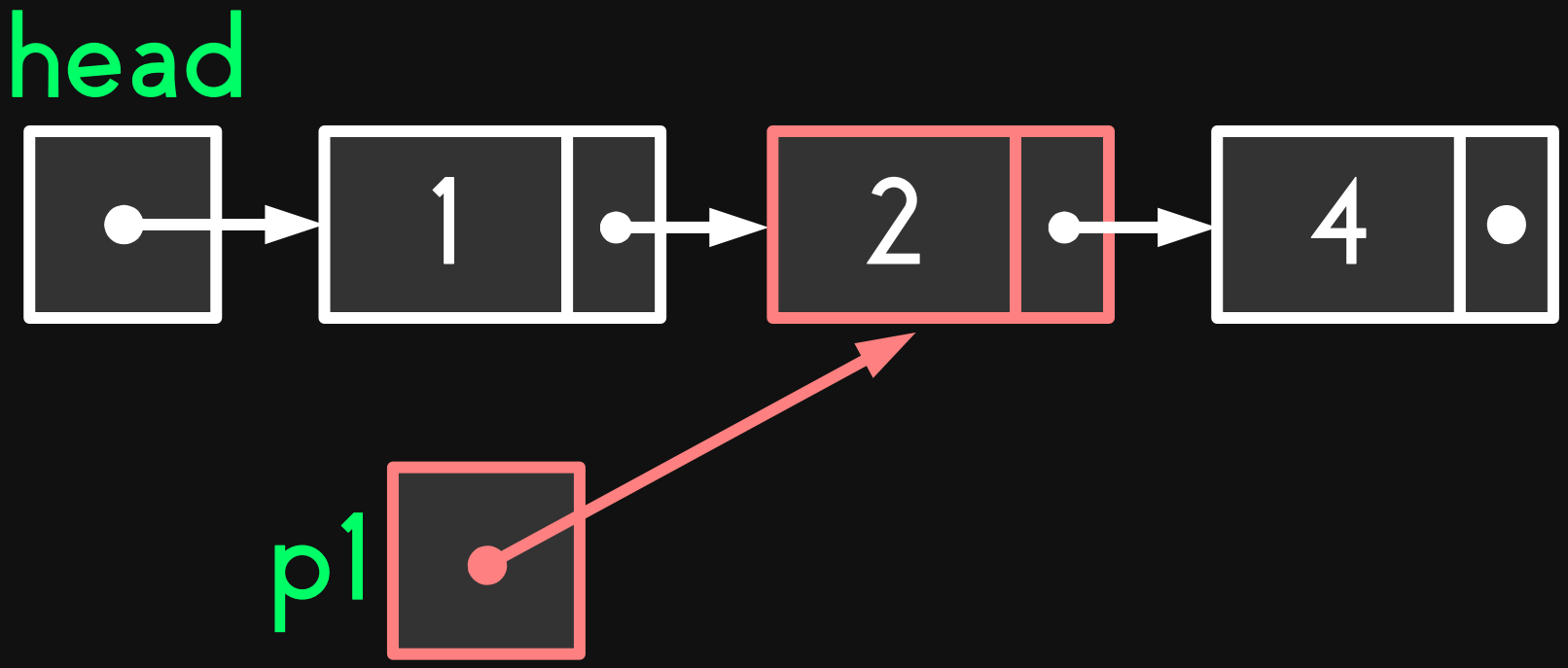
print the data in the node
being pointed by **p1**.

View



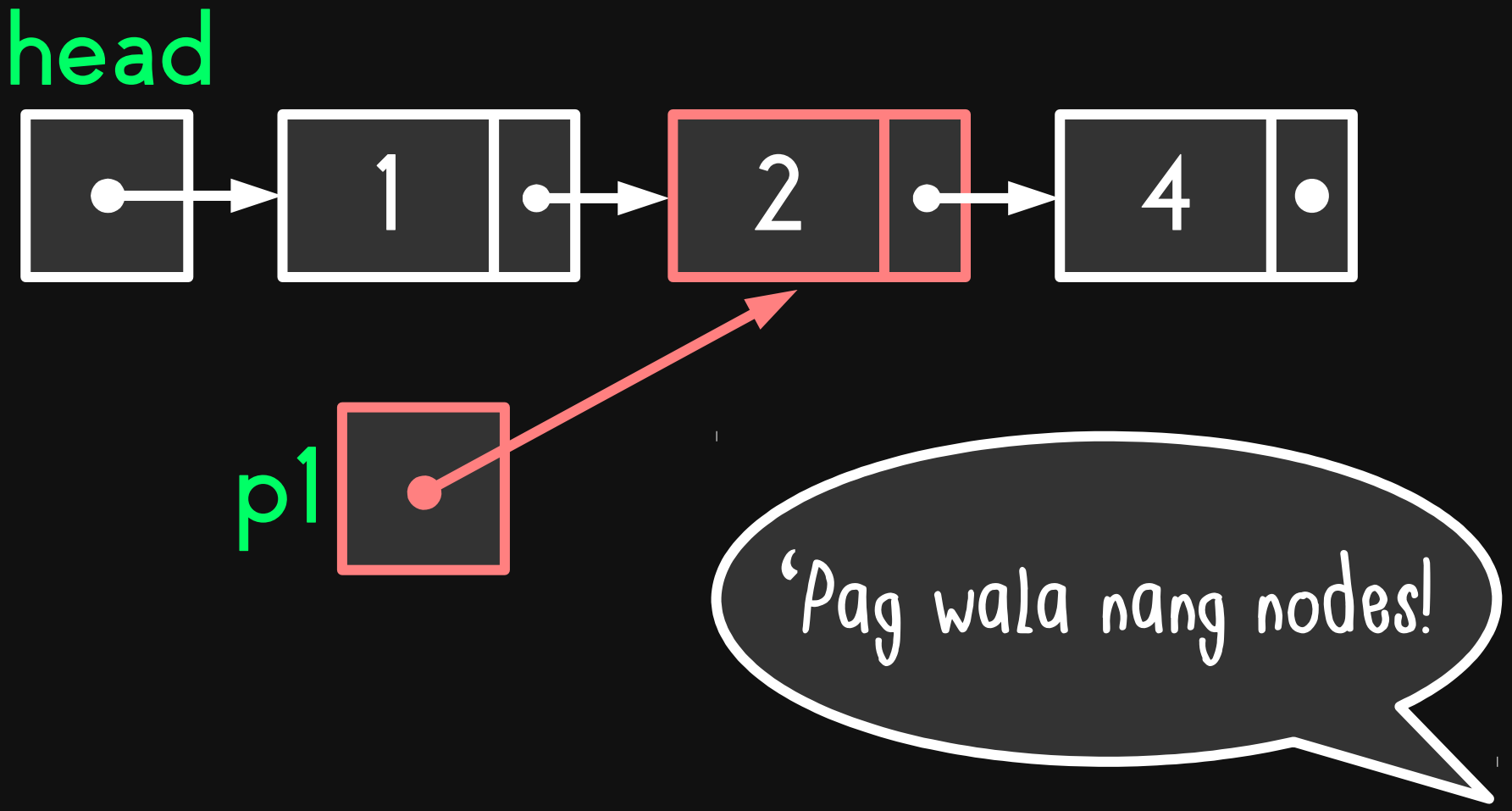
and then move to the next node.
print the data.
move to the next node.
and so on.

View



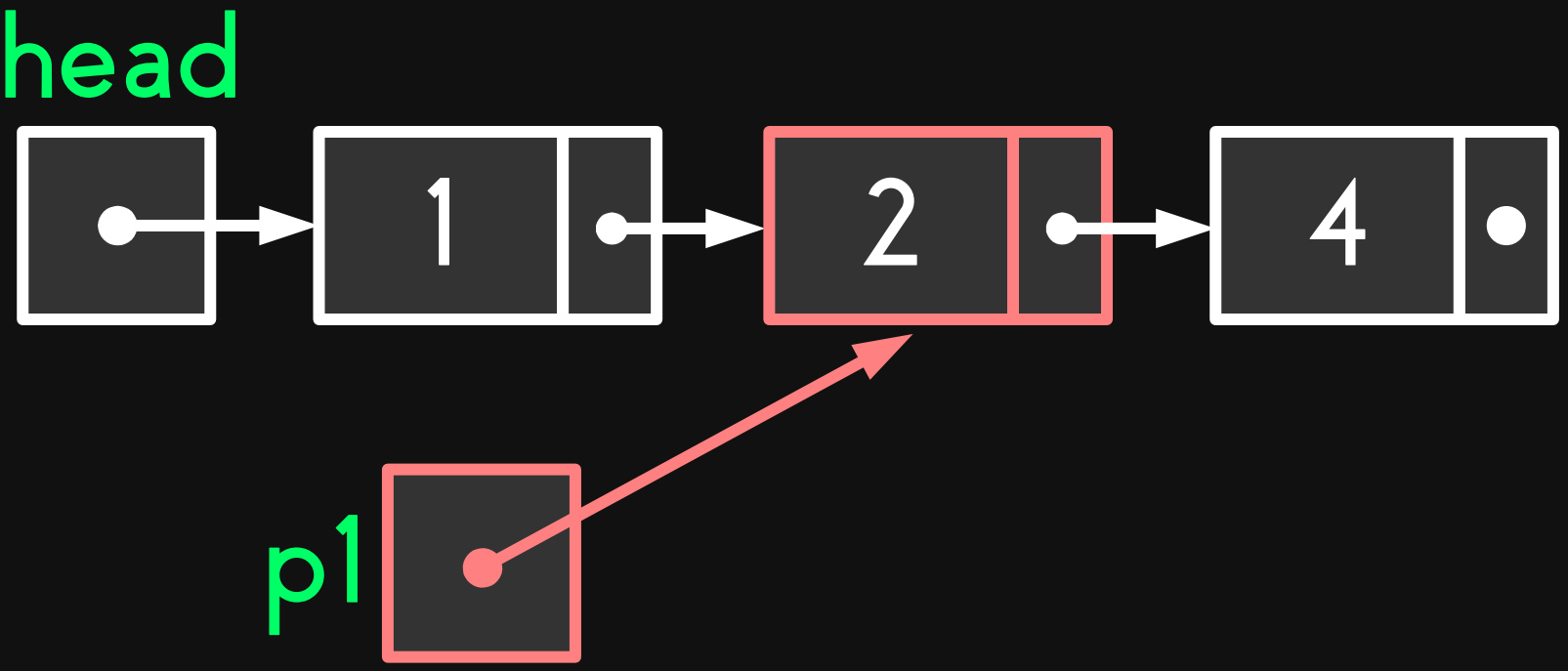
When will this stop?

View



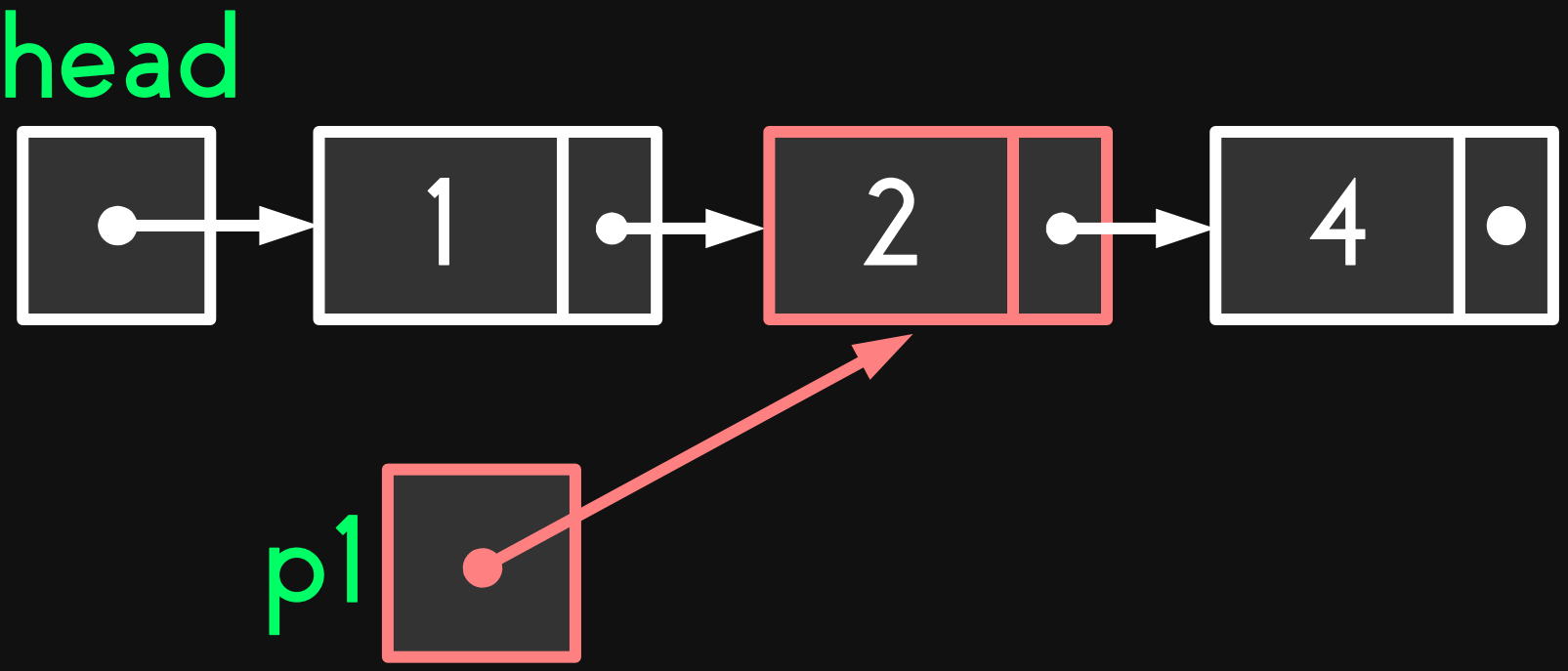
When will this stop?

View



When will you know that there are no more nodes whose data must be printed?

View



HINT: What will be the value of **p1** if there are no more nodes to be printed?

Search

finds a specific item
from the linked list

Search

finds a specific item
from the linked list

similar to the view operation

Search

stops once
the item is found
or the end of the list
is reached

LINKED LISTS VS ARRAYS

Linked Lists vs Arrays

linked lists

save memory

Linked Lists vs Arrays

allocated **memory**
will NEVER exceed
what is needed by
the program

Linked Lists vs Arrays

dynamic **arrays**
can handle the change
in maximum size but
there is **a possibility that**
there will be unused
allocated memory.

CMSC 21

FUNDAMENTALS OF PROGRAMMING

Kristine Bernadette P. Pelaez

Institute of Computer Science
University of the Philippines Los Baños