CMSC 21

FUNDAMENTALS OF PROGRAMMING

Kristine Bernadette P. Pelaez

Institute of Computer Science
University of the Philippines Los Baños

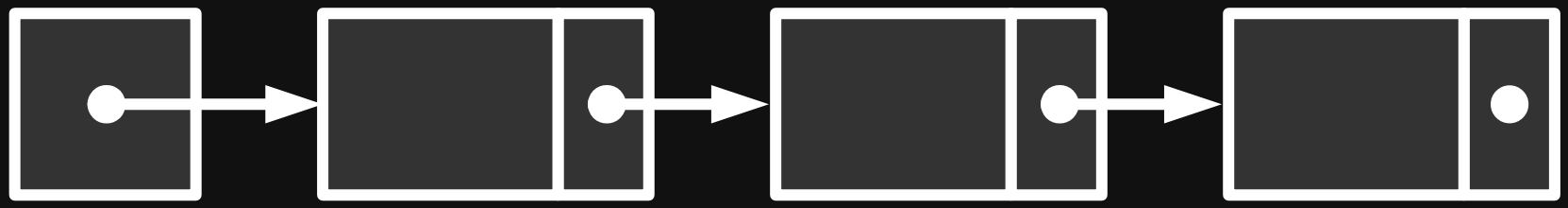
LINKED LISTS

Linked Lists

a data structure that
consists of dynamic variables
linked together to form a
chain-like structure

Linked Lists

a data structure that
consists of **dynamic variables**
linked together to form a
chain-like structure

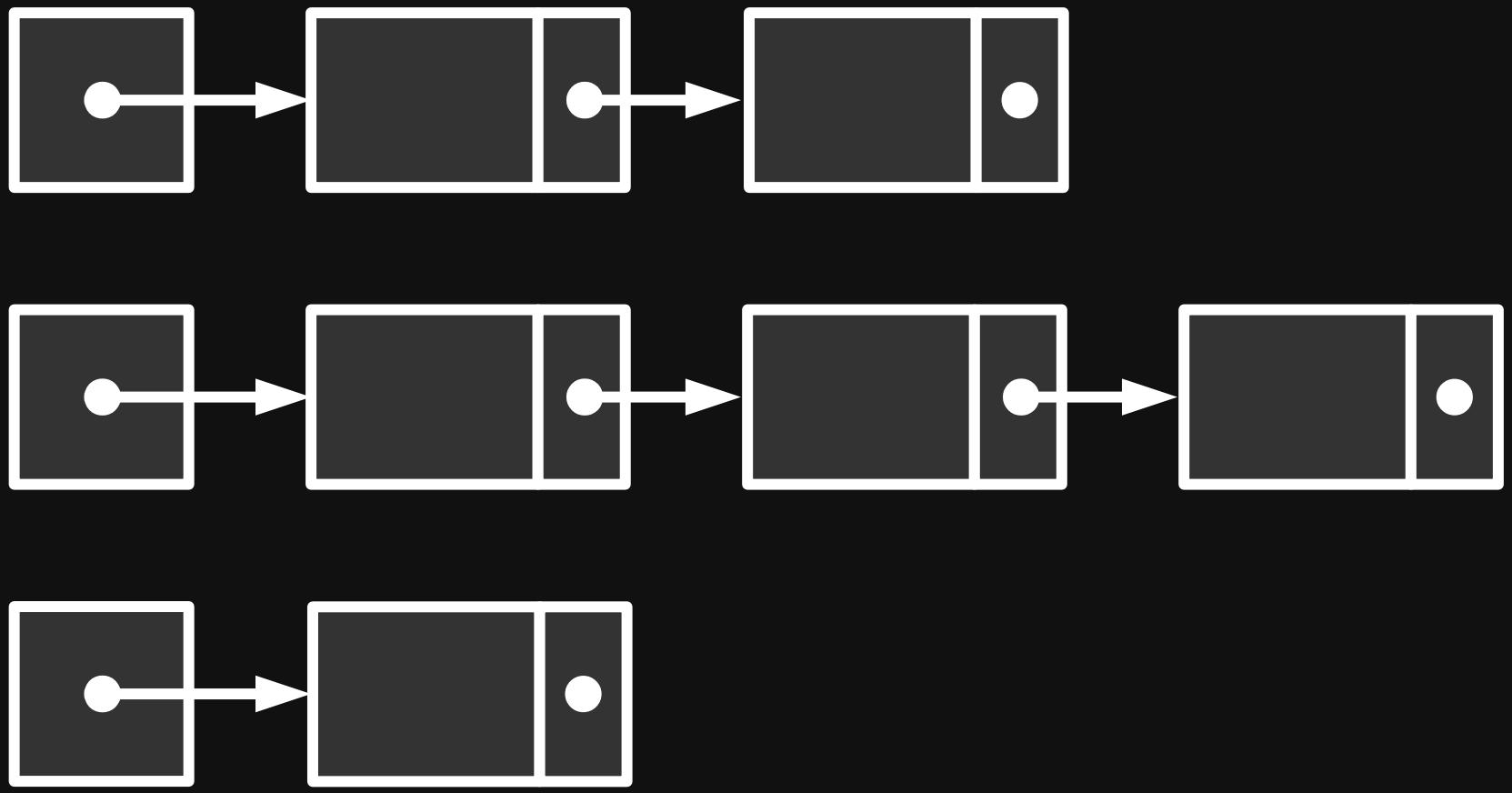


Linked Lists

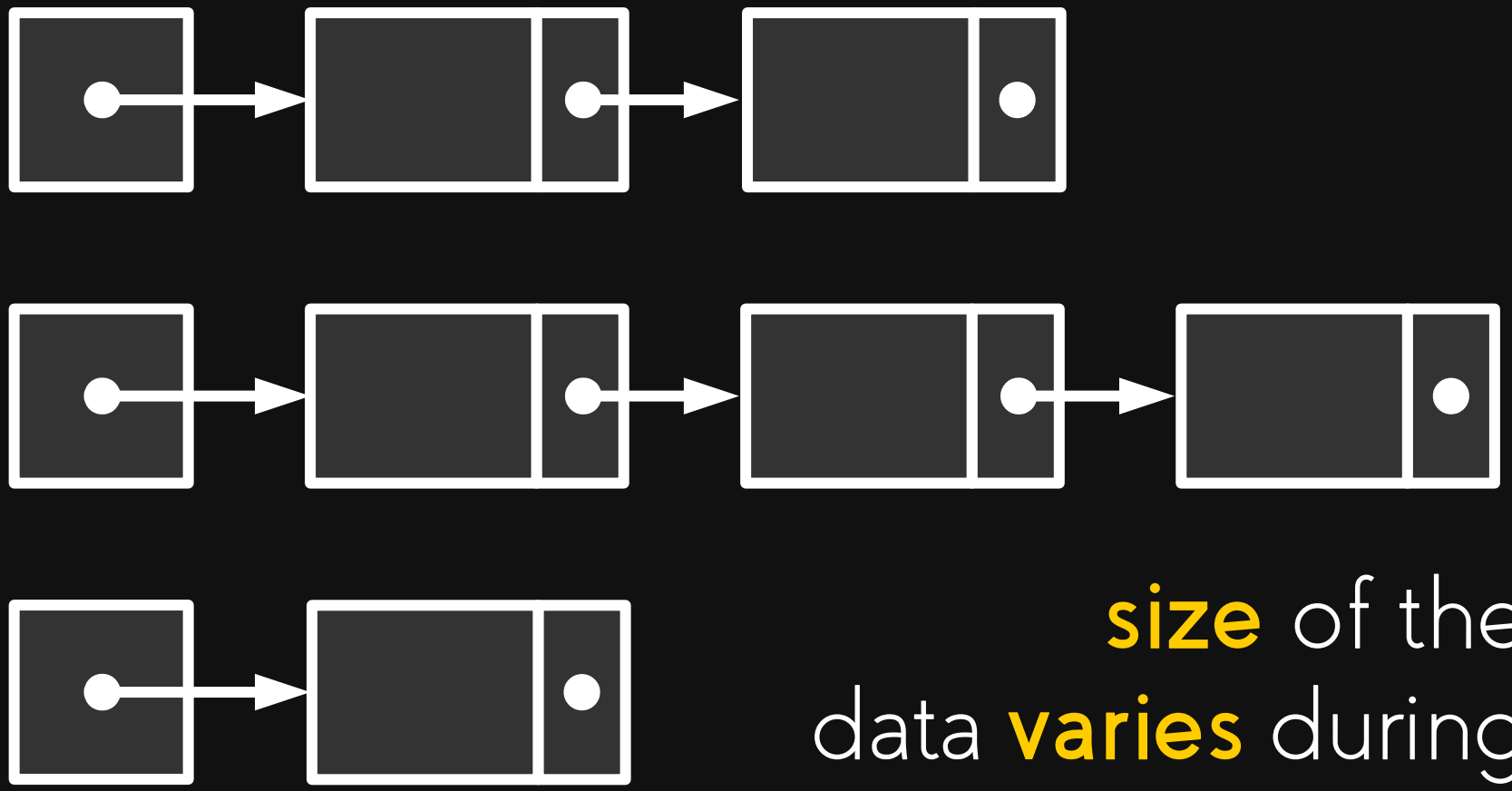
an **alternative to arrays**

during execution. **linked lists**
can either grow or shrink
following the user's needs

Linked Lists

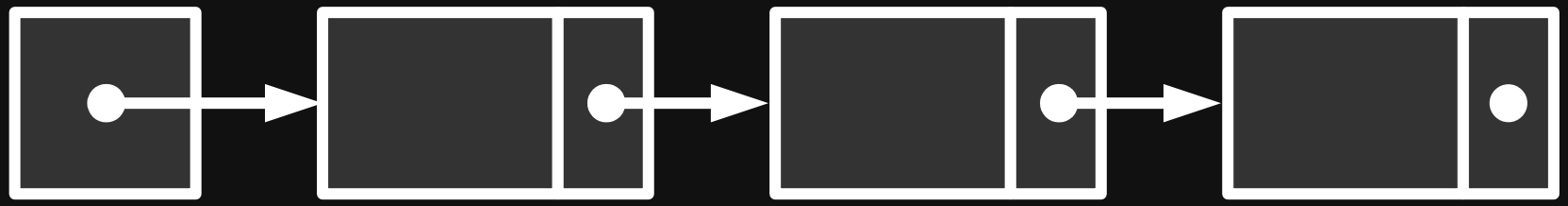


Linked Lists



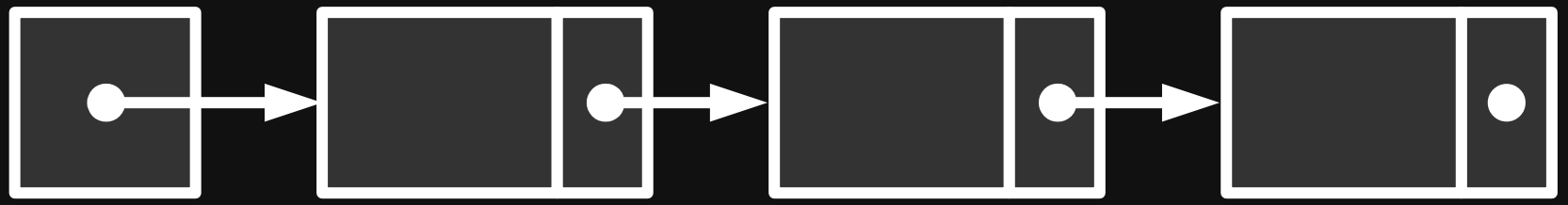
size of the
data **varies** during
execution

Linked Lists



head

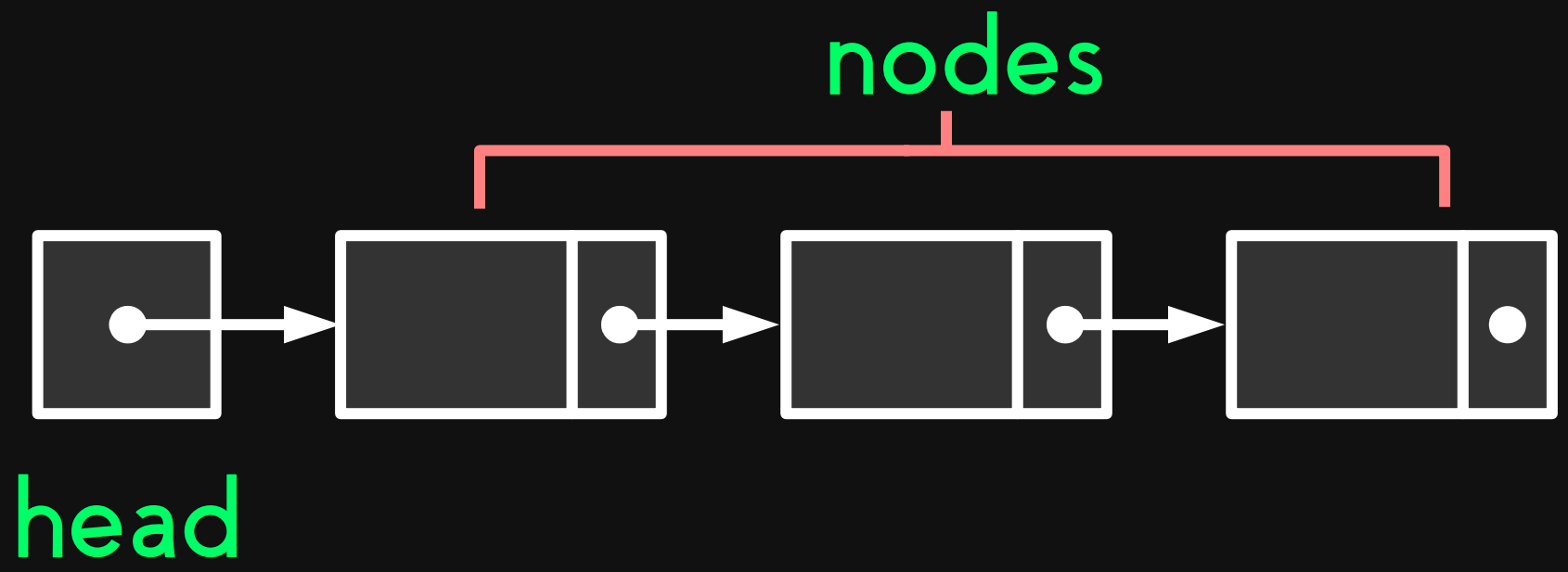
Linked Lists



head

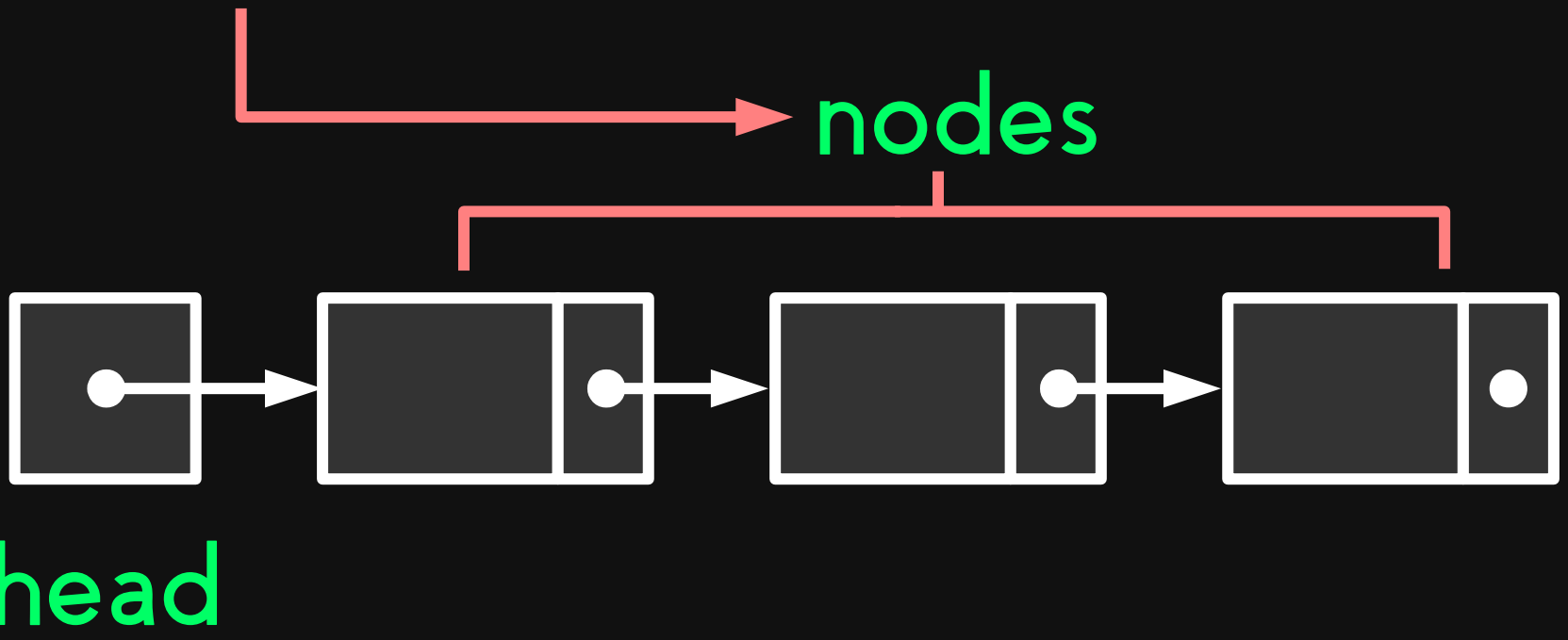
pointer to the first
element of the linked list

Linked Lists

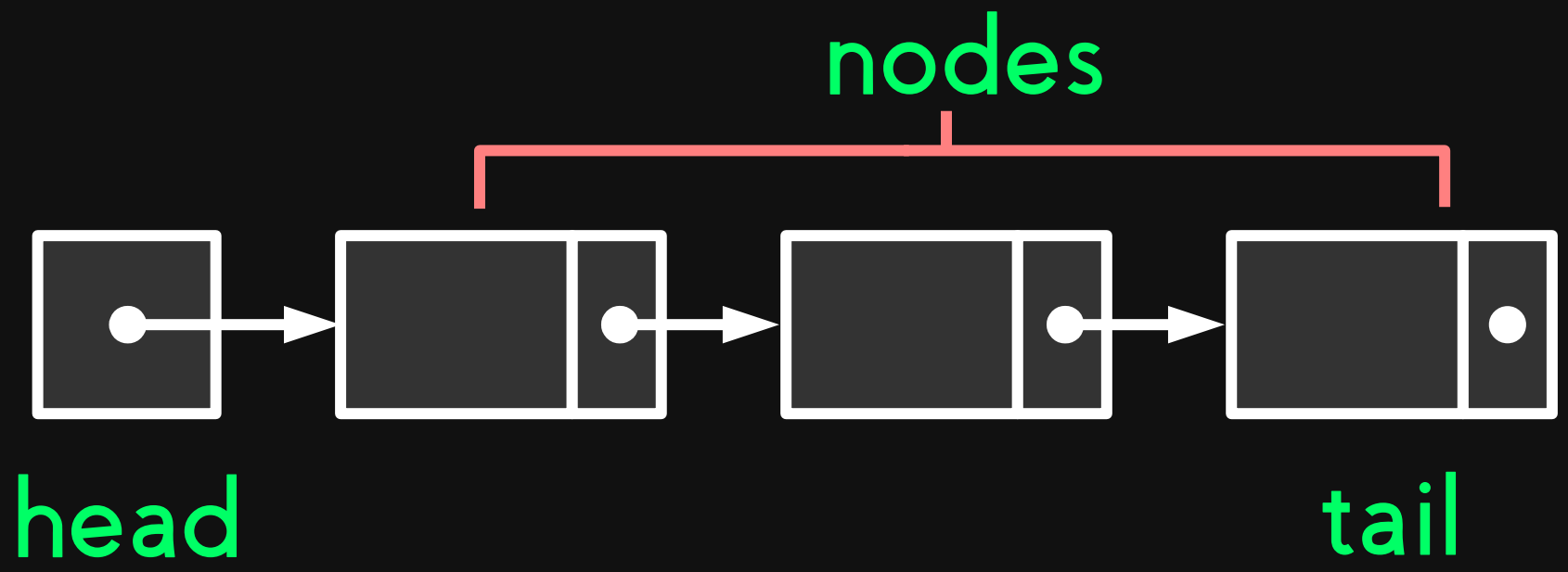


Linked Lists

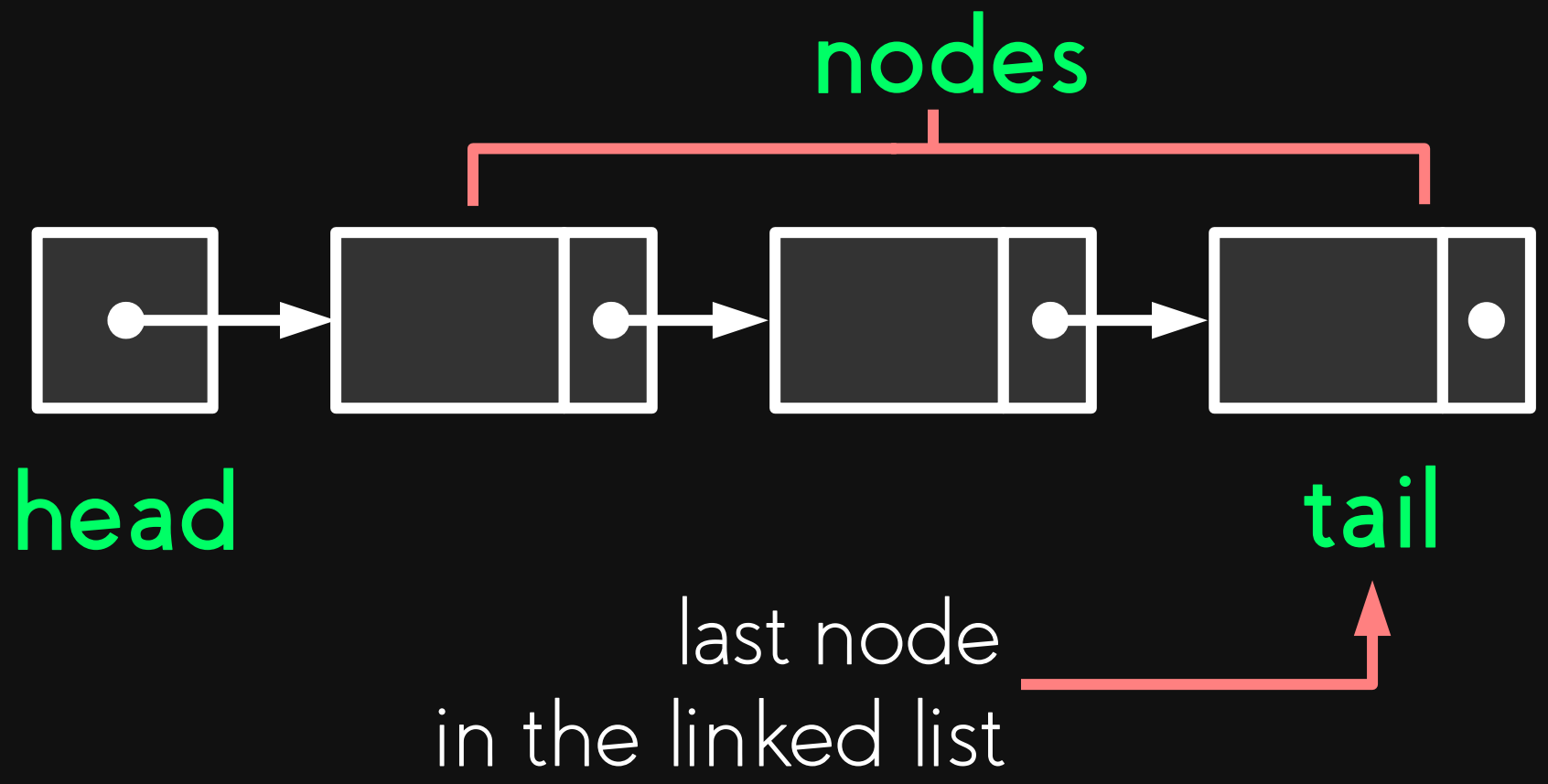
the elements of
the linked list



Linked Lists



Linked Lists



Nodes

a self-referential structure*

Nodes

a **self-referential** structure*

*a **structure that has** a member
that is **a pointer to itself**

Nodes

a **self-referential** structure*

*a **structure that has** a member
that is **a pointer to itself**

*a structure that **contains a pointer**
to a structure **of the same data type**

Self-Referential Structures

```
struct pokemon{
   int HP;
   char name[64];
   //ptr is a pointer to an instance
   //of struct pokemon (which is itself)
   struct pokemon *ptr;
};
```

Self-Referential Structures

```
typedef struct queue{
   int num;
   char name[64];
   struct queue *next;
}pila;
```

Self-Referential Structures

```
typedef struct{
   int data;
   LIST *next;
}LIST;
```

IS THIS VALID?

Self-Referential Structures

```
typedef struct{
   int data;
   LIST *next;
}LIST;
```

NO.

IS THIS VALID?

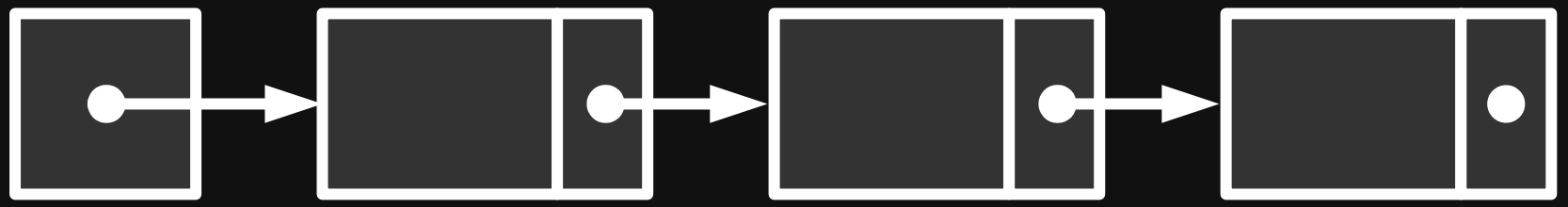
Self-Referential Structures

```
typedef struct{
   int data;
   LIST *next;
}LIST;
```

The synonym **LIST** is **not yet recognized** by the compiler.

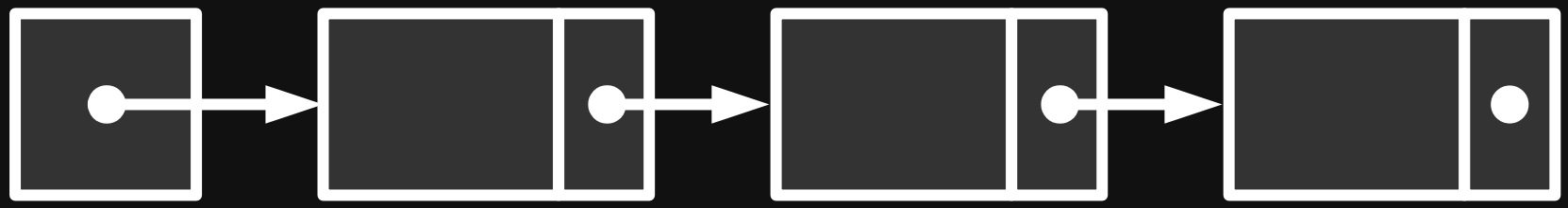
REMINDERS

the head pointer MUST
hold the address of the
first element of the linked list



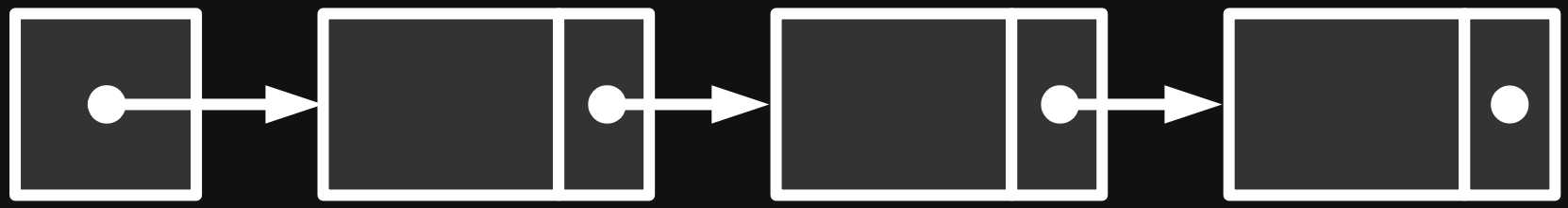
REMINDERS

the pointer of **each node**
MUST point to the next node
in the linked list



REMINDERS

if the pointers do not point to any node. they must have a value of NULL.



REMINDERS

`malloc()` is used to
dynamically **grow** the list
[add a node to the list]

REMINDERS

`malloc()` is used to
dynamically **grow** the list
[add a node to the list]

`free()` is used to
dynamically **shrink** the list
[delete a node in the list]

Kinds of Linked Lists

Singly Linked Lists

Doubly Linked Lists

Circular Linked Lists

Linked Lists with Dummies

LINKED LIST OPERATIONS

Operations

Insert (add a node)

Delete (delete a node)

Search (search the list)

View (print the contents of the list)

Insert

insert values to a linked list

Insert

insert values to a linked list

has three(3) cases:

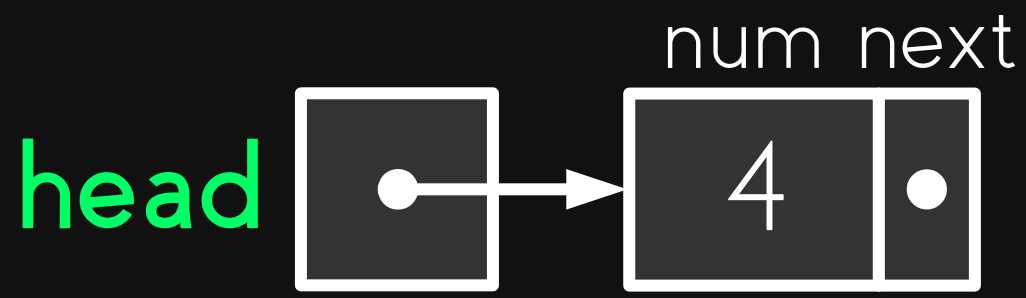
- insert at head
- insert at middle
- insert at tail

Insert at Head

insert a node
at the beginning
of the list

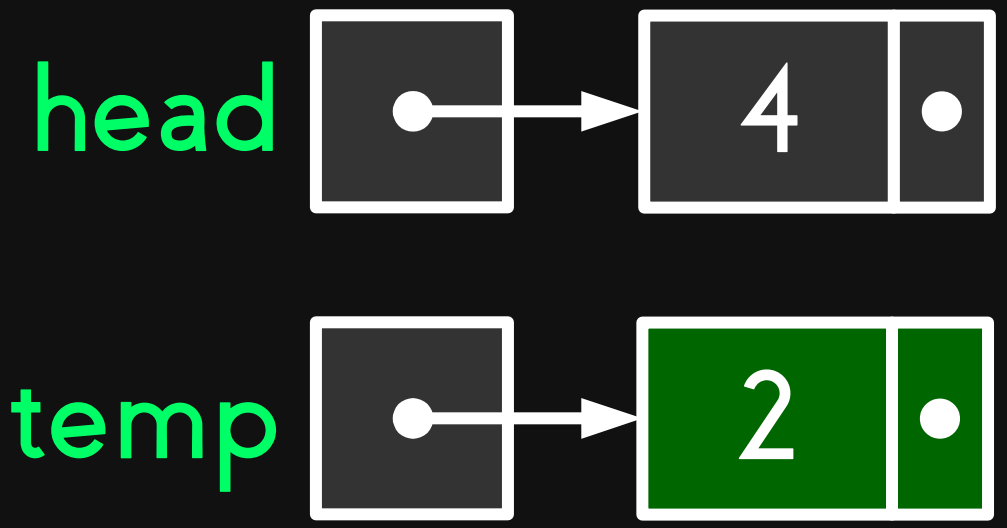
```
struct NODE{
   int num;
   struct NODE *next;
};
```

Insert at Head



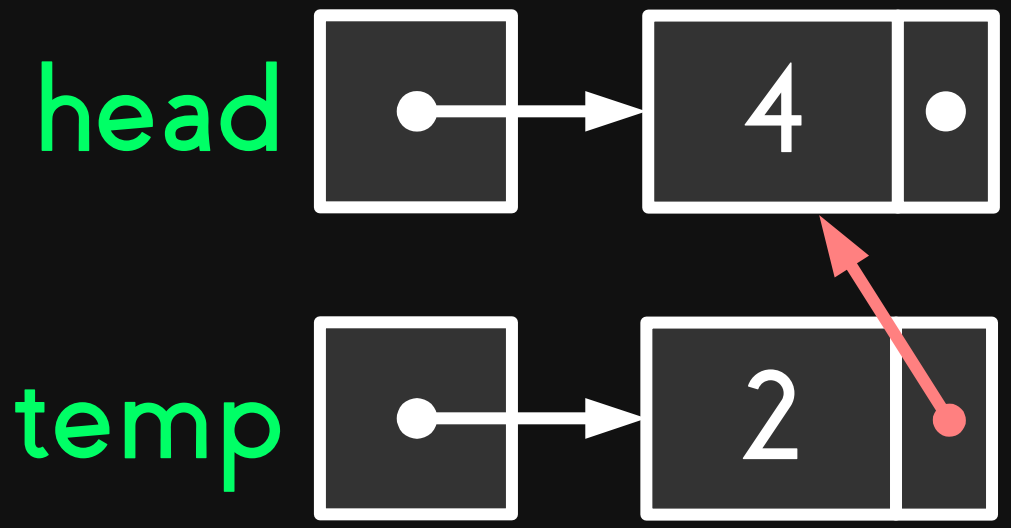
insert 2 at the start of the list

Insert at Head



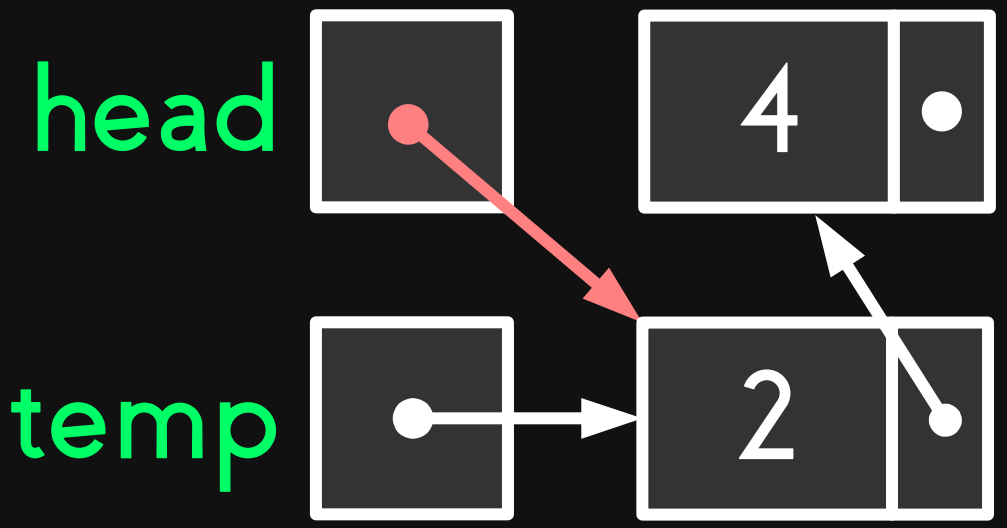
`malloc()` a new node for 2.
give it to a new pointer (temp).

Insert at Head



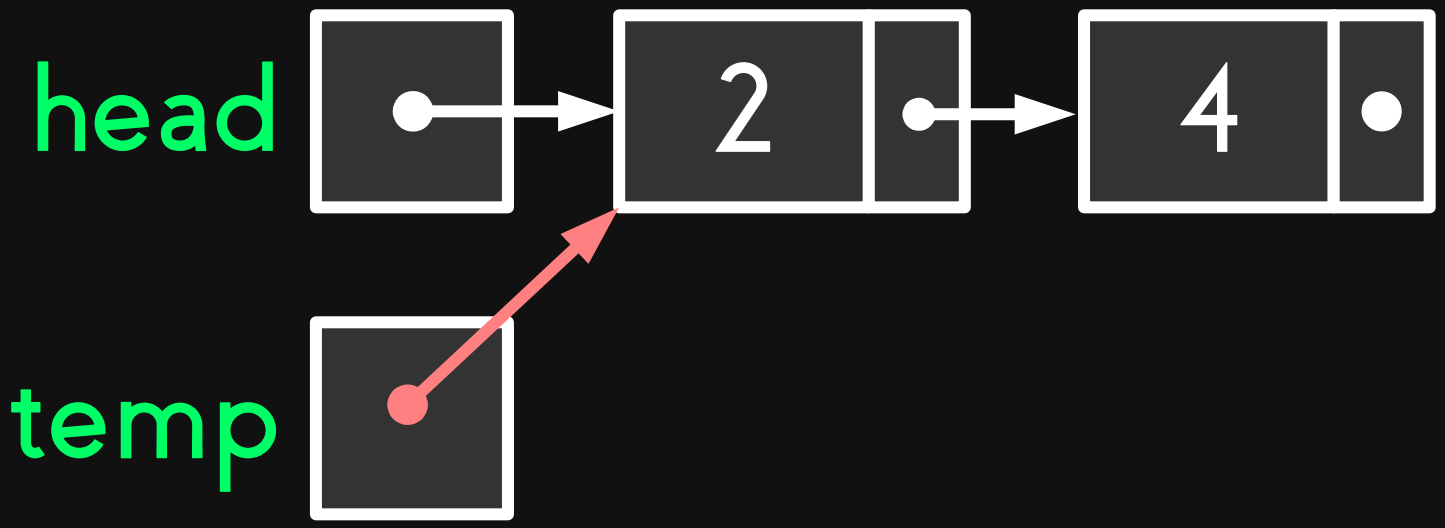
make the `next` pointer of the new node (2) point to the current head (4).

Insert at Head



make the head pointer point
to the new node (2).

Insert at Head



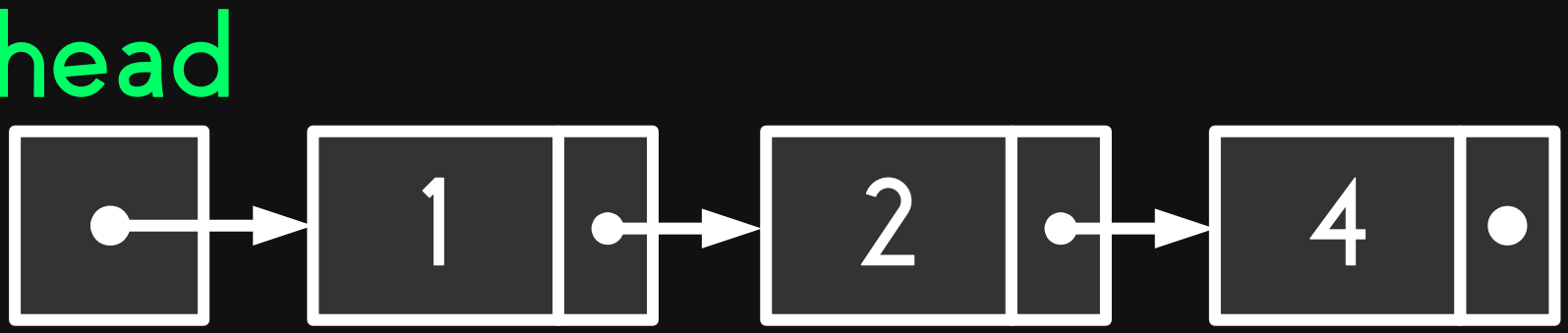
rearrangement of the
previous diagram.

Insert at Middle

insert a node
between two nodes
of the list

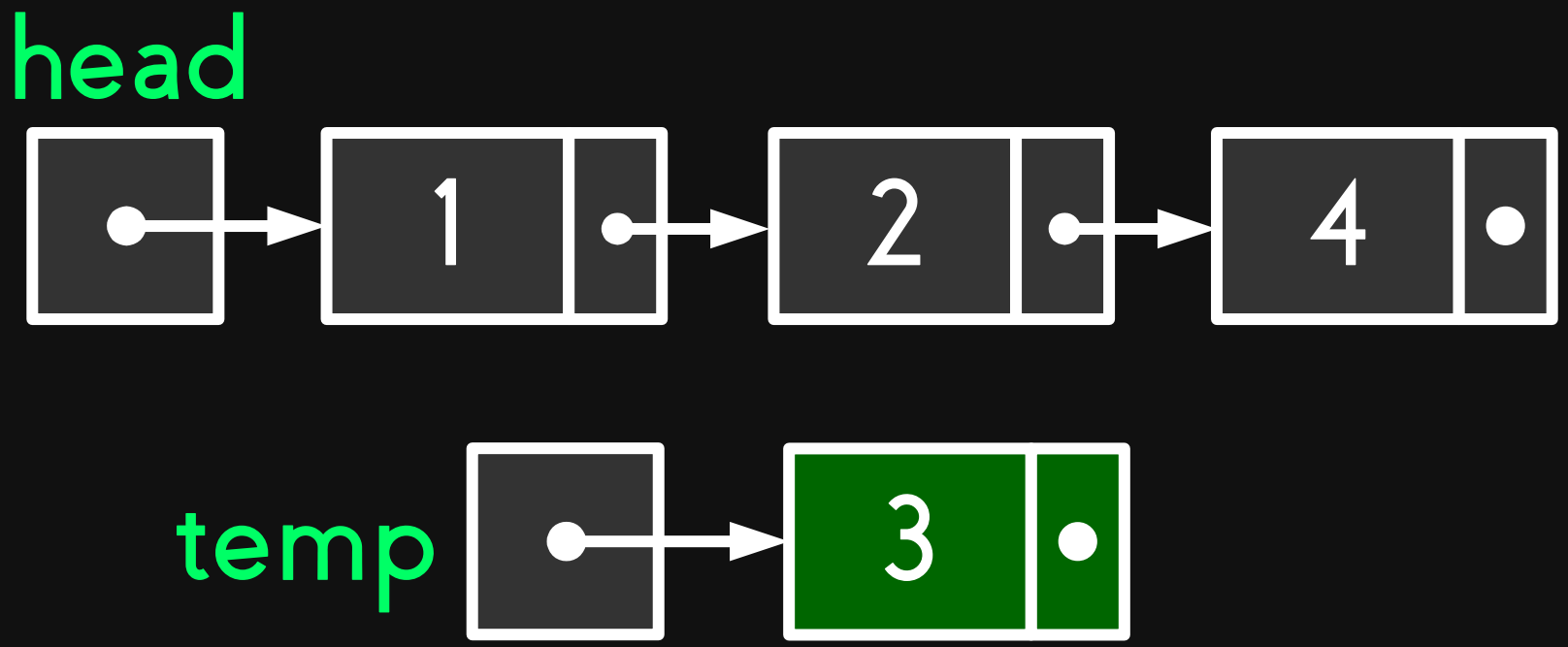
```
struct NODE{
   int num;
   struct NODE *next;
};
```

Insert at Middle



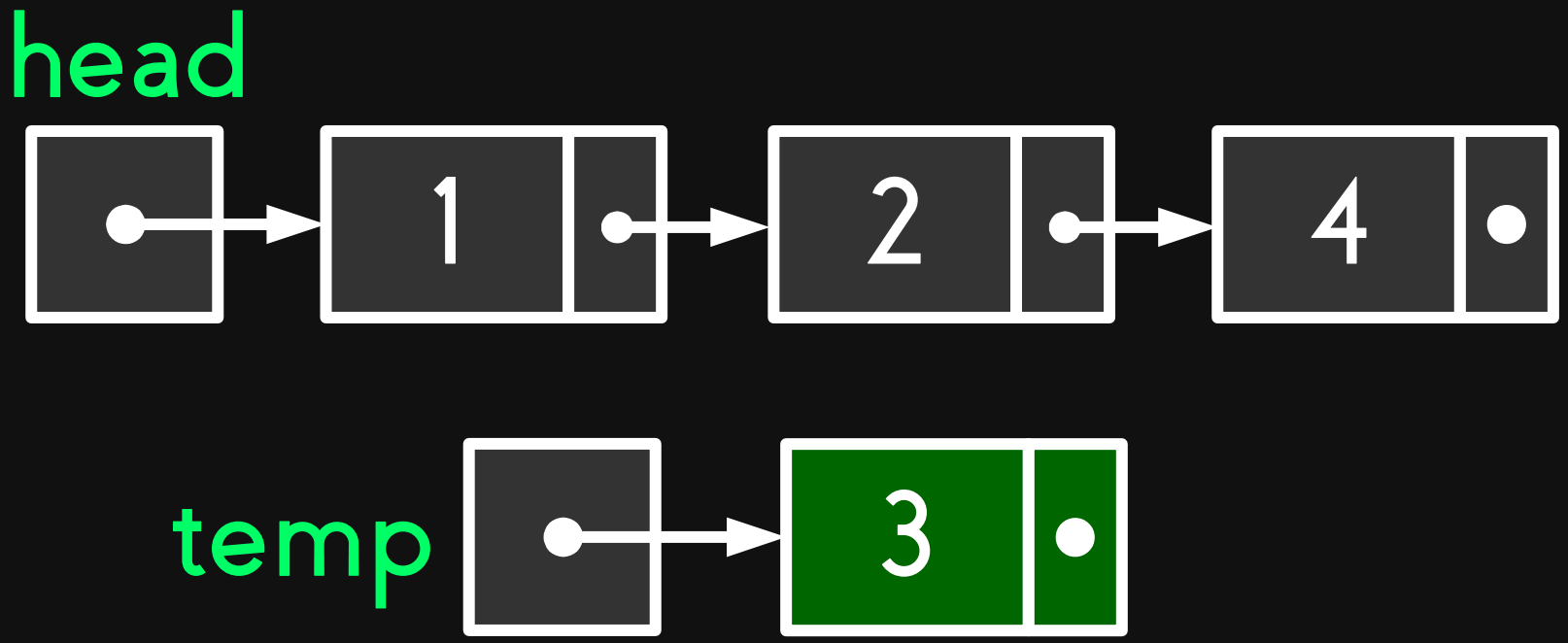
insert 3 in the middle
of the linked list.

Insert at Middle



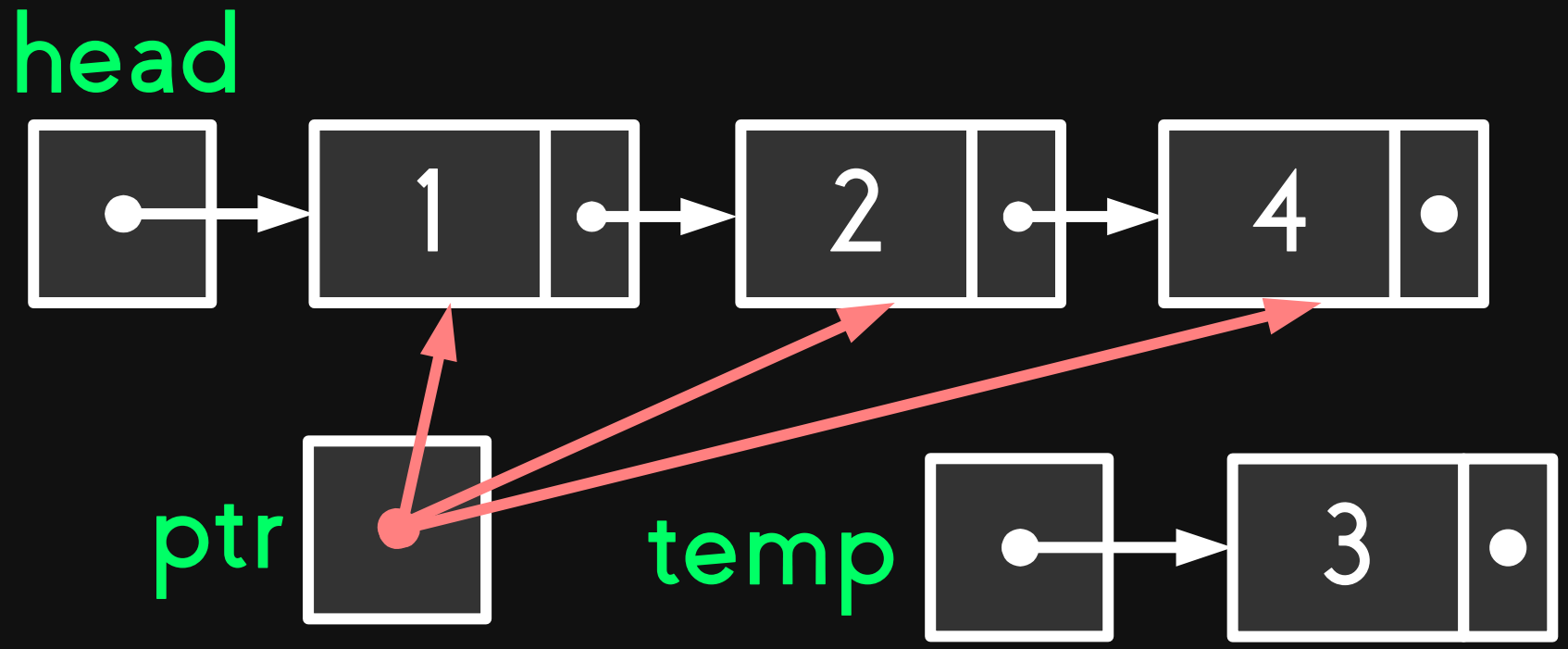
`malloc()` a new node for 3.
give it to a new pointer (temp).

Insert at Middle



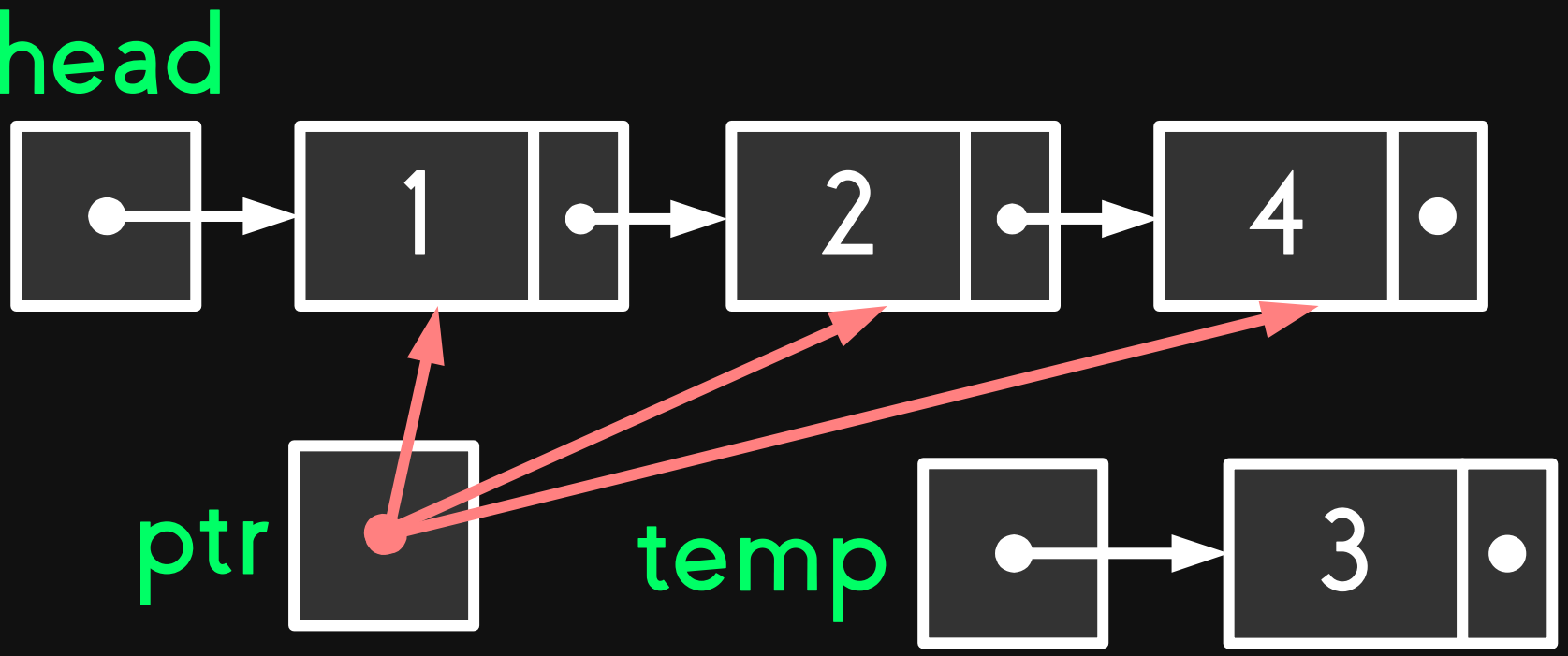
find the position where the new node is to be inserted.

Insert at Middle



select the node **before** the position
where the new node will be inserted

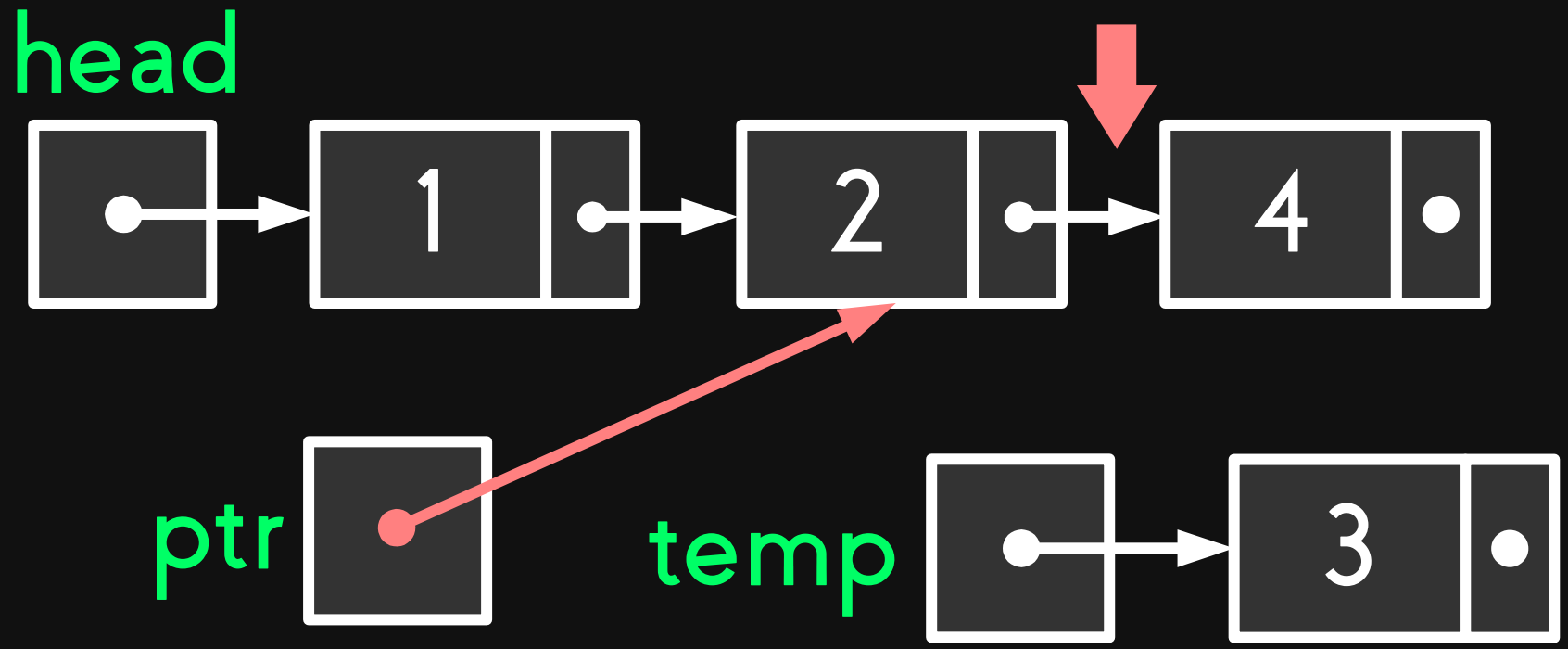
Insert at Middle



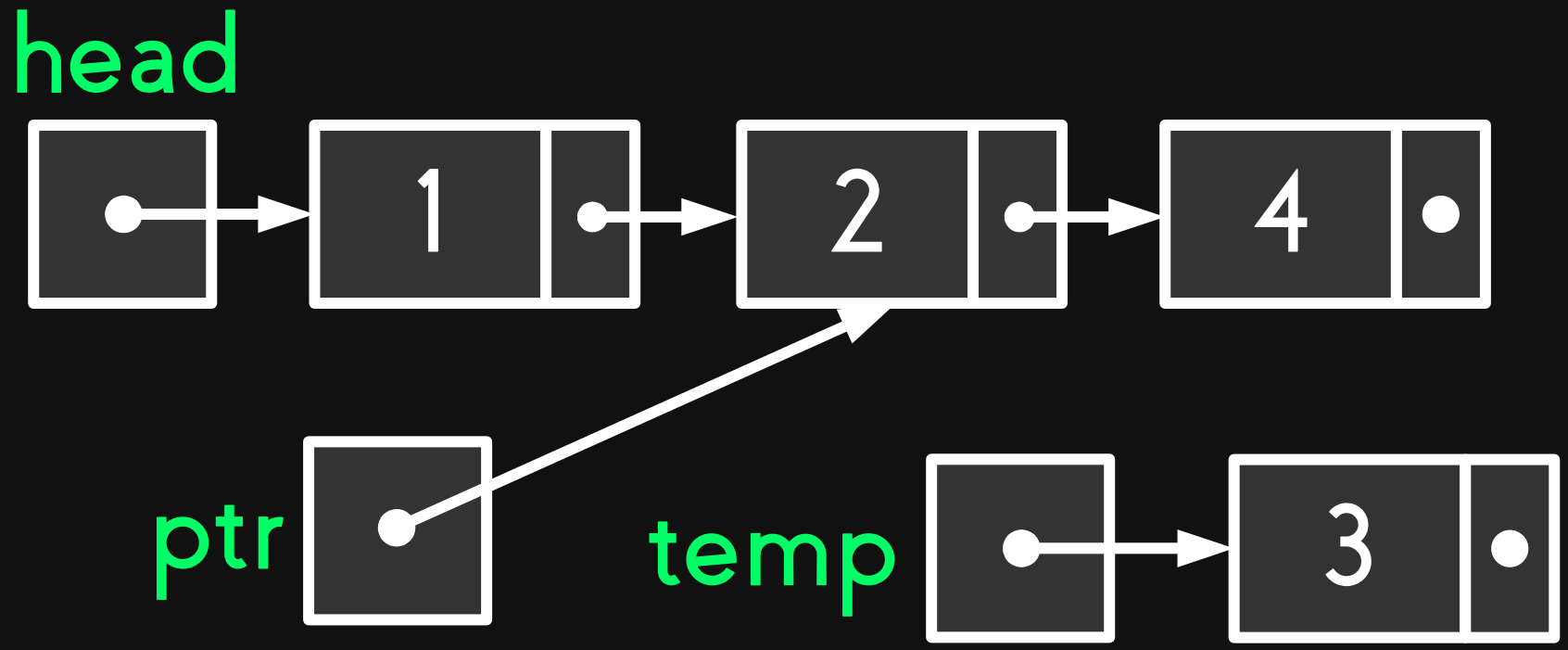
let's say we want to insert the node containing 3 in between the nodes containing 2 and 4.

Where should ptr point?

Insert at Middle

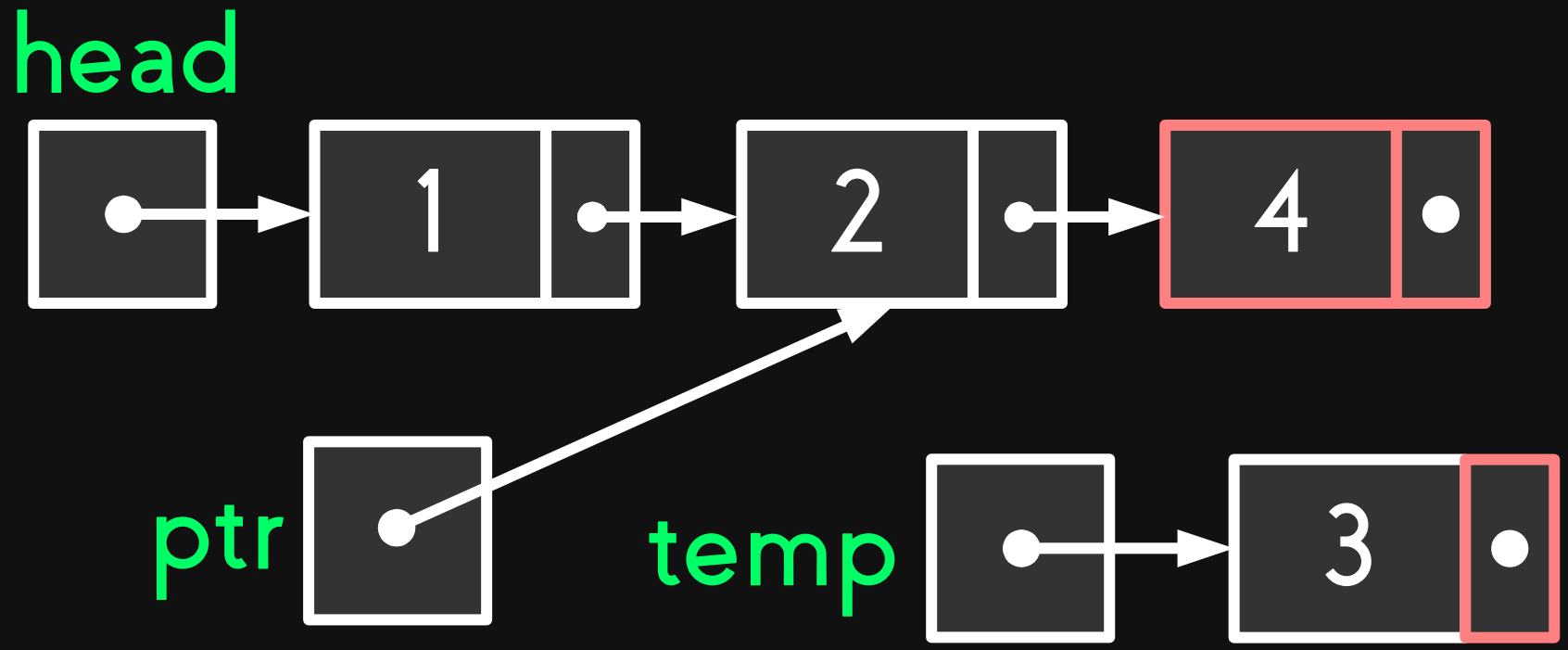


Insert at Middle



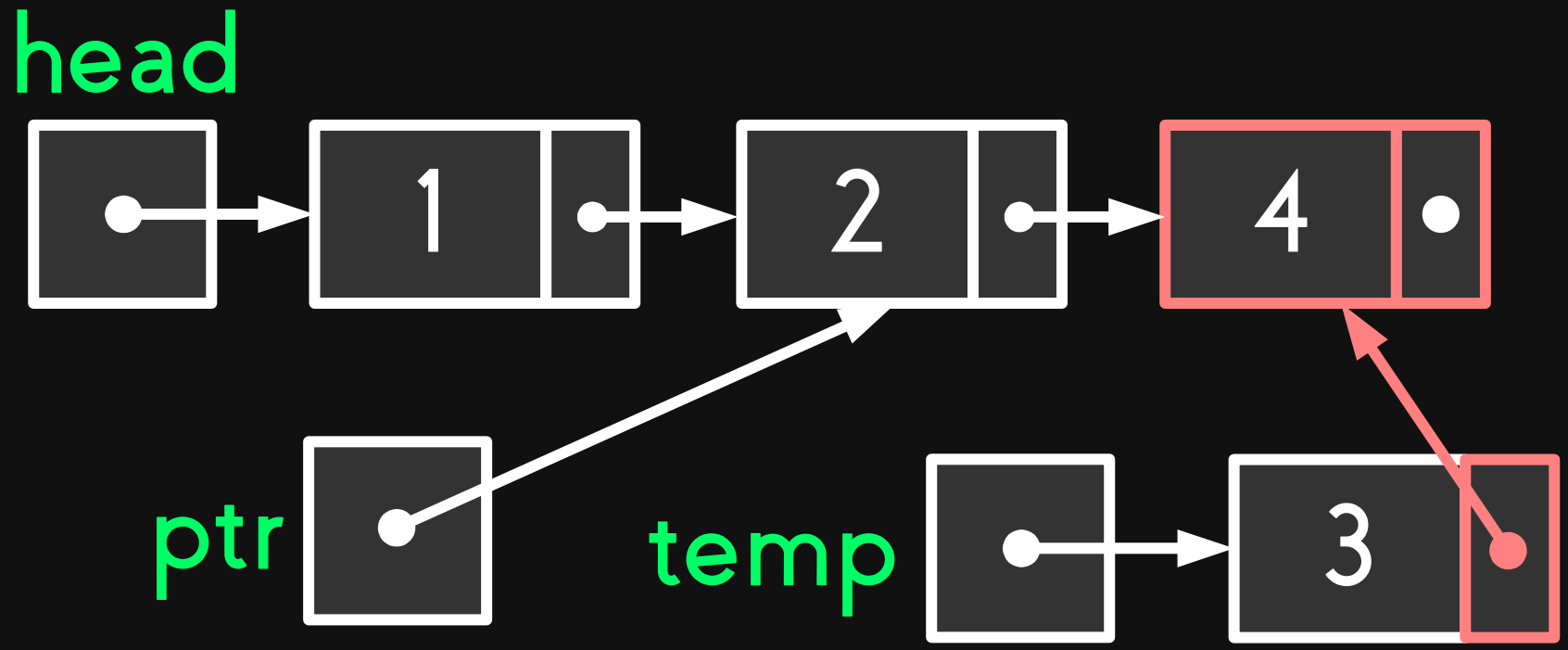
point the `next` pointer of the new node to the node being pointed by the `next` of the node being pointed by `ptr`

Insert at Middle



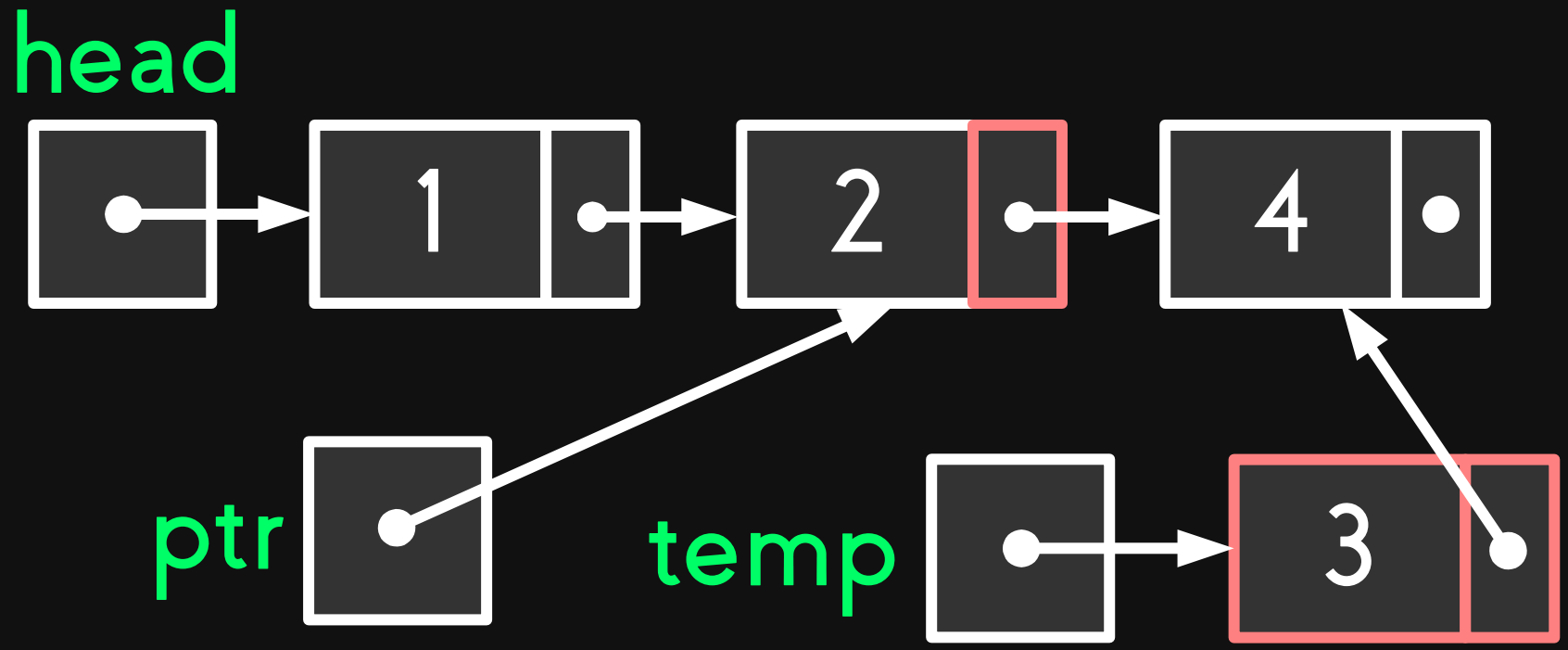
point the `next` pointer of the new node to the node being pointed by the `next` of the node being pointed by `ptr`

Insert at Middle



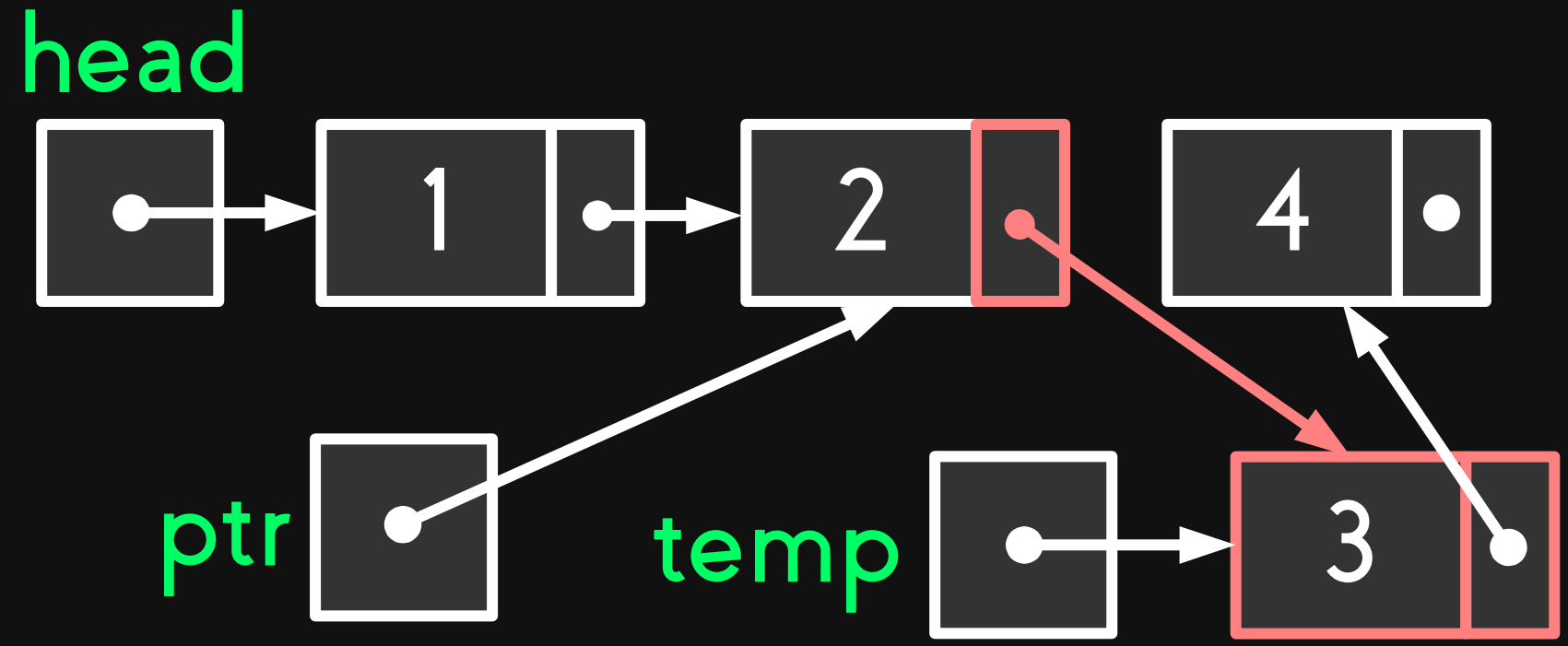
point the `next` pointer of the new node to the node being pointed by the `next` of the node being pointed by `ptr`

Insert at Middle



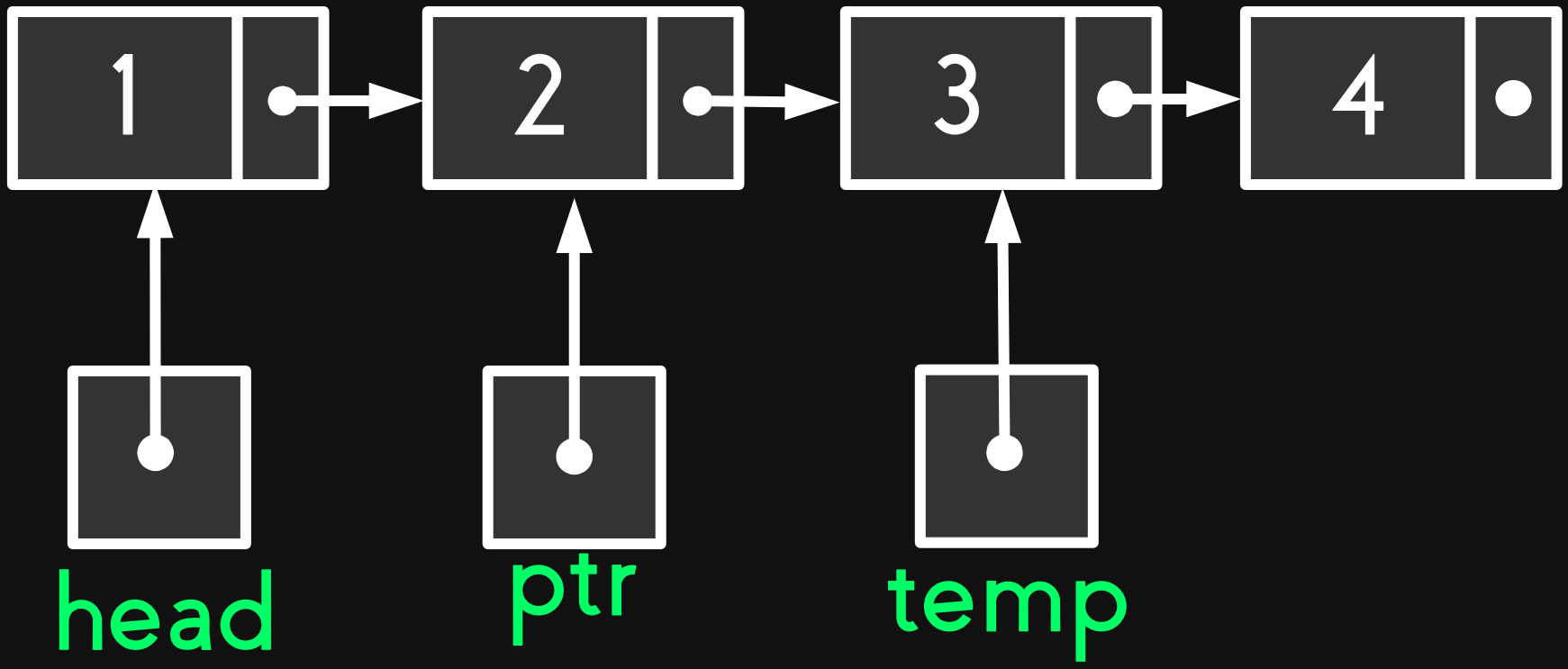
point the `next` pointer of the node being pointed by `ptr` to the new node

Insert at Middle



point the `next` pointer of the node being pointed by `ptr` to the new node

Insert at Middle



Rearrangement of the nodes.

Insert at Tail

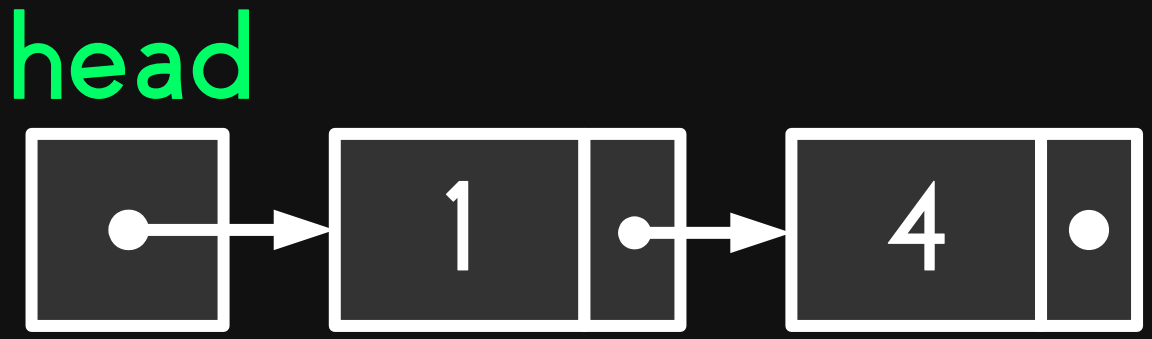
can be considered
as a special case of
insert at middle

Insert at Tail

insert a node
after the last node
of the list

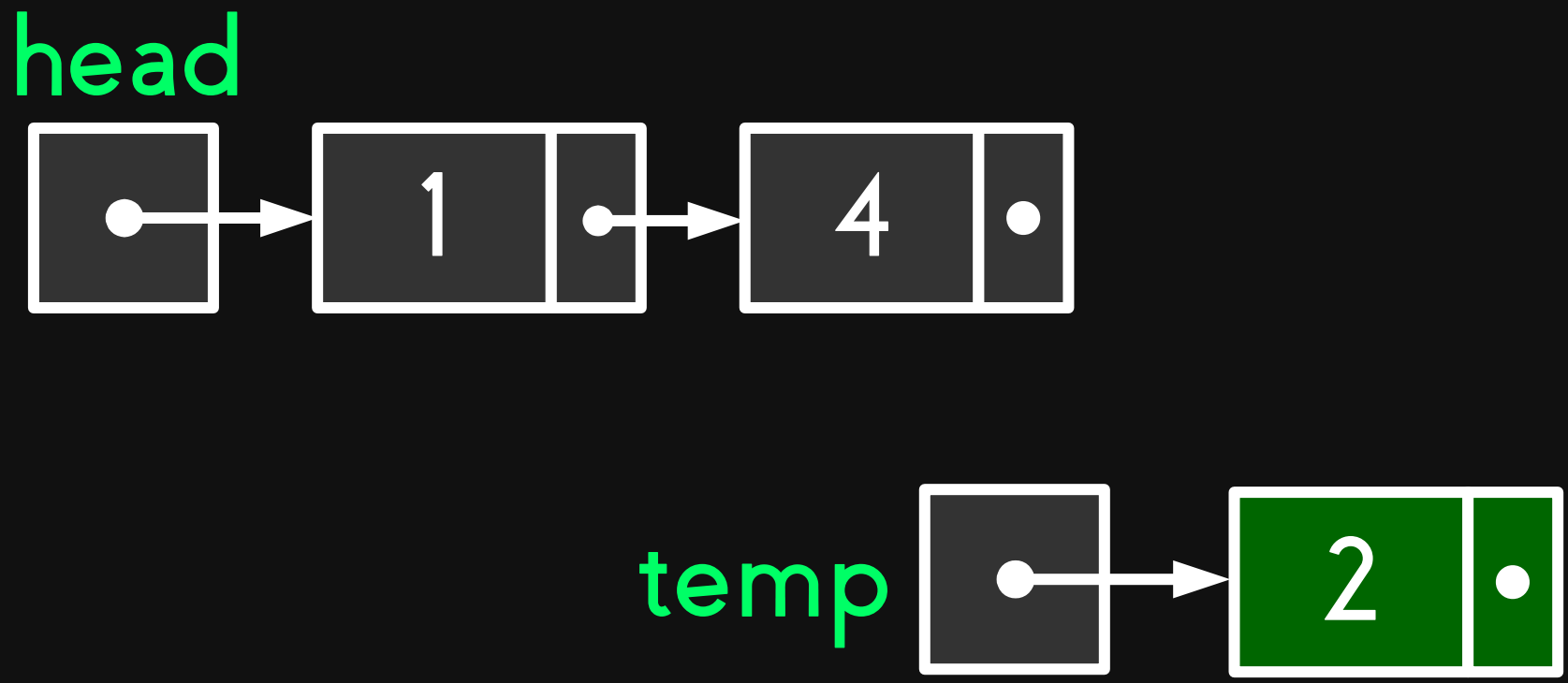
```
struct NODE{
   int num;
   struct NODE *next;
};
```

Insert at Tail



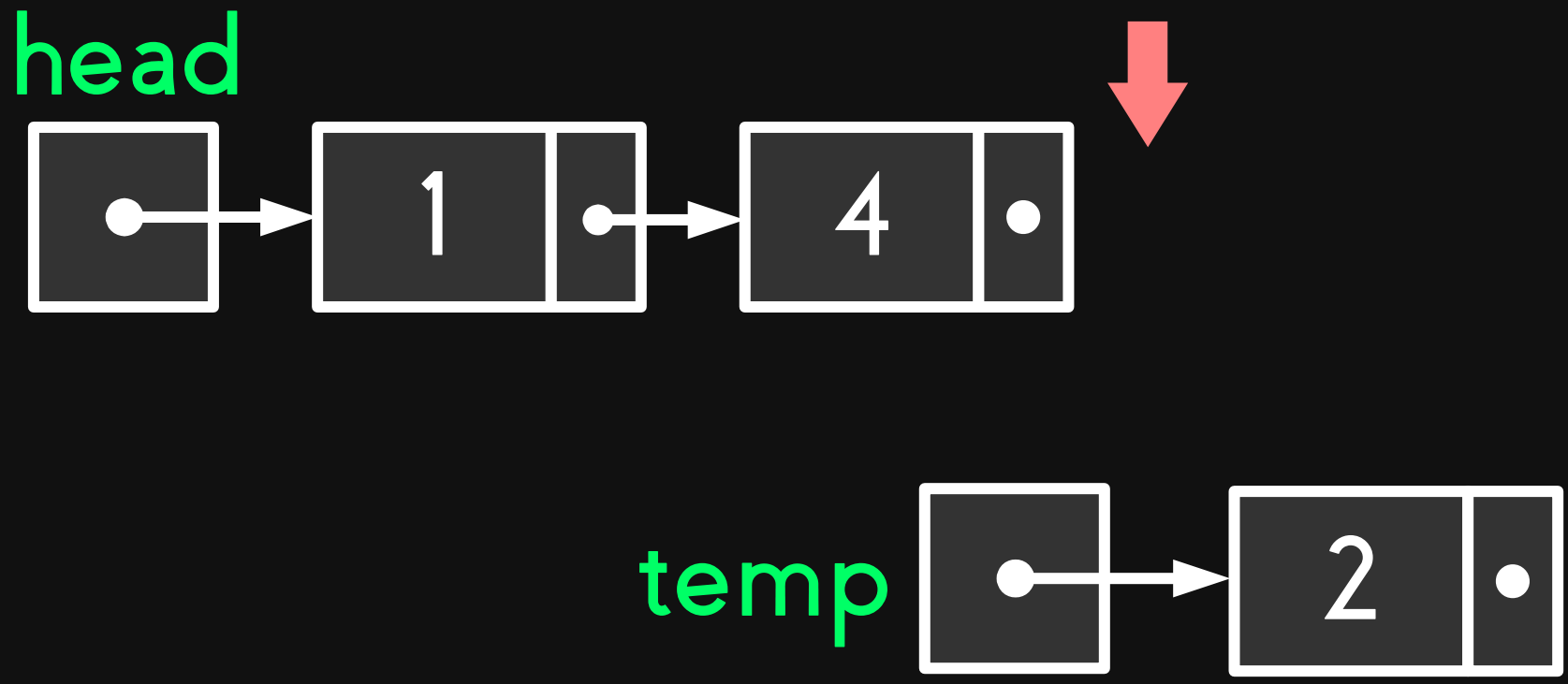
insert 2 at the end
of the linked list.

Insert at Tail



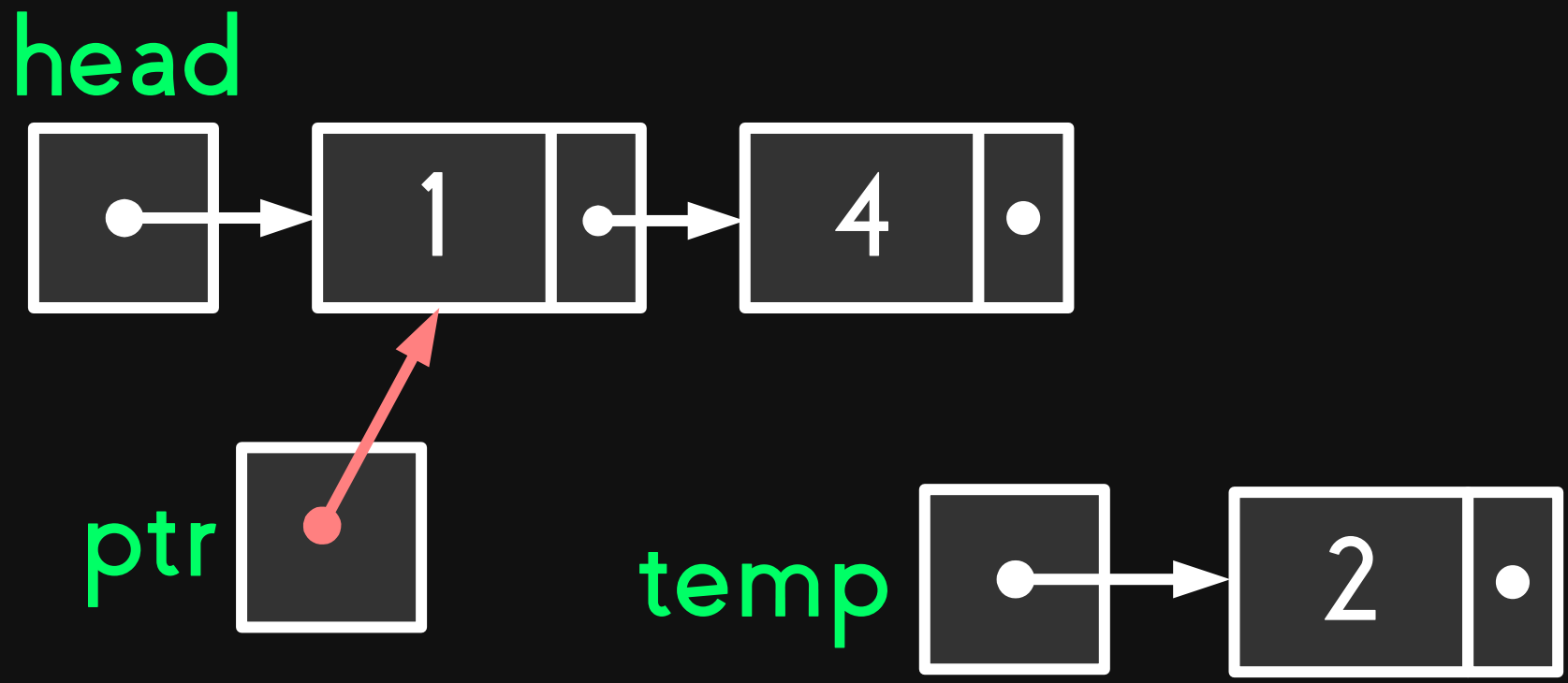
`malloc()` a new node for 2.
give it to a new pointer (temp).

Insert at Tail



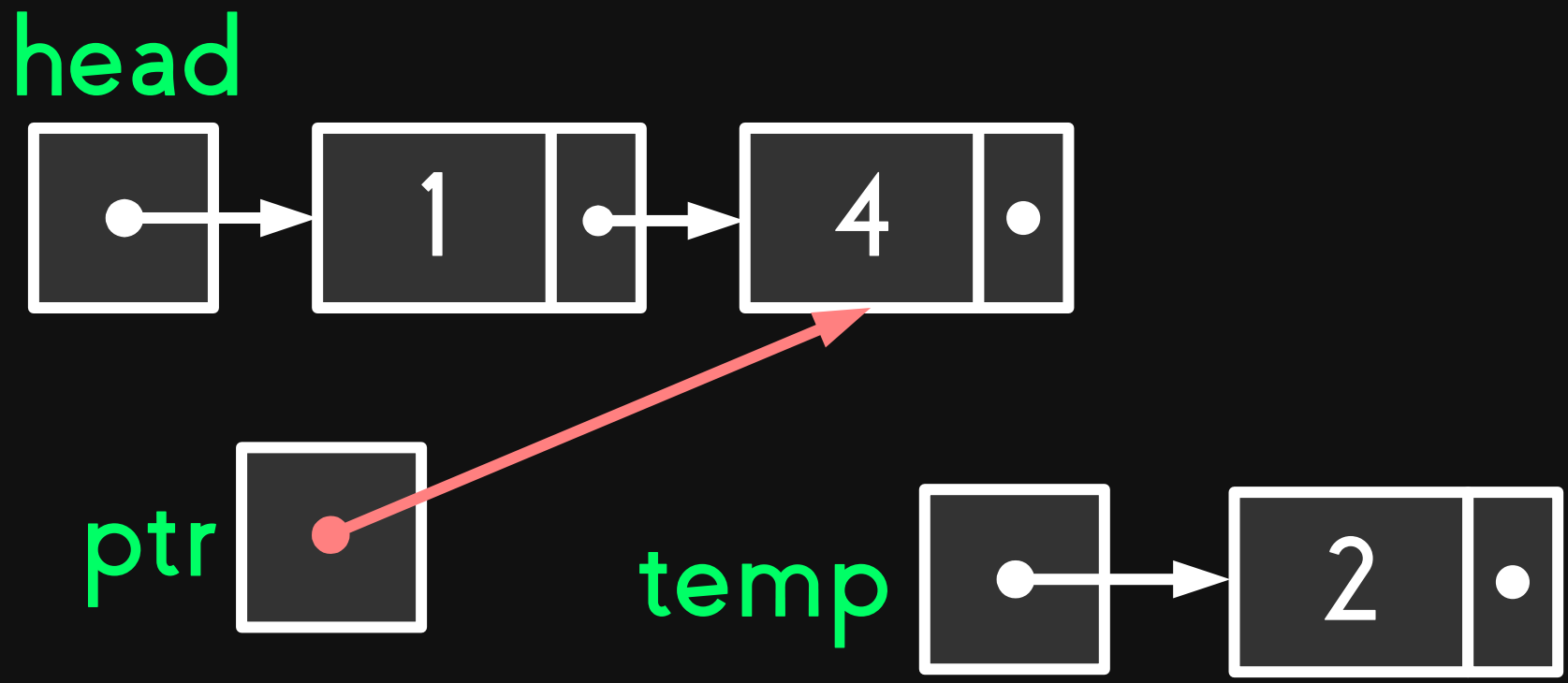
find the tail node.

Insert at Tail



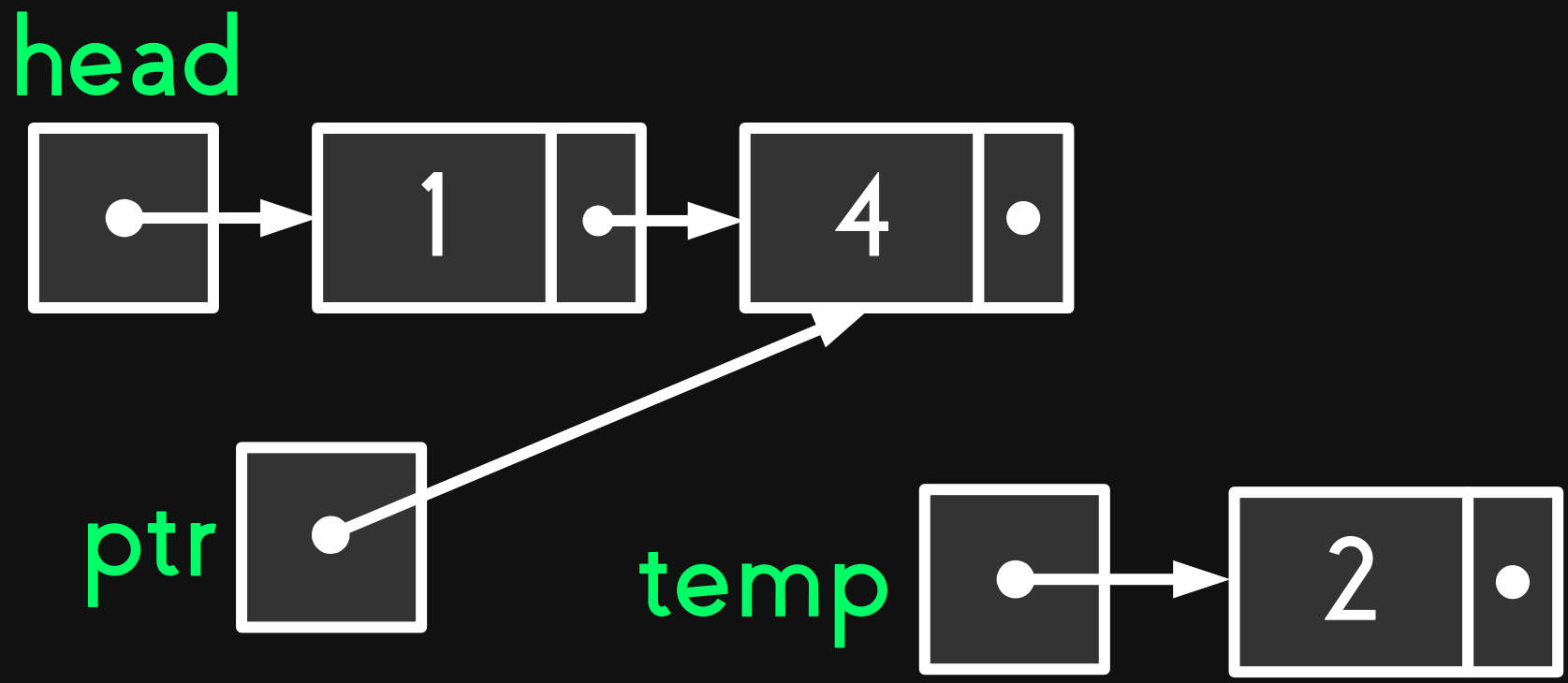
find the tail node.

Insert at Tail



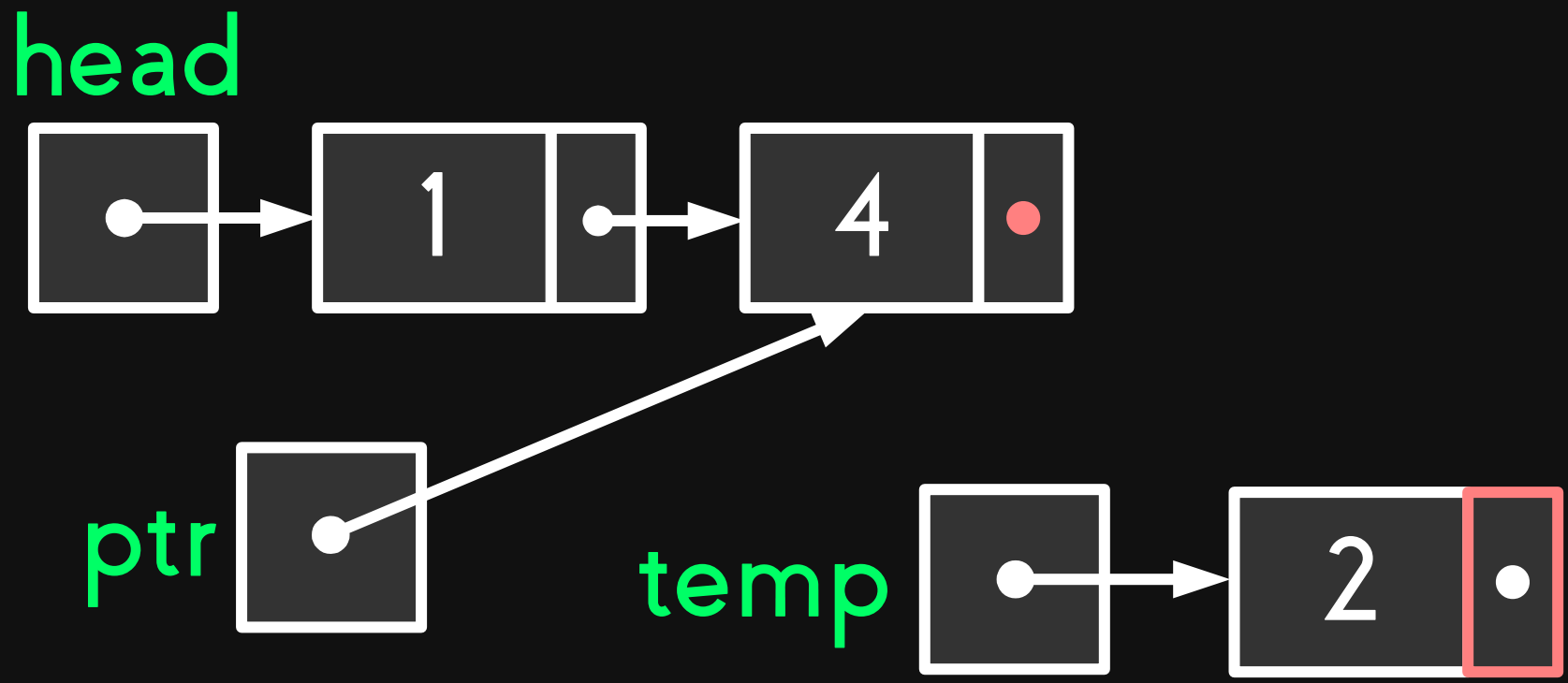
find the tail node.

Insert at Tail



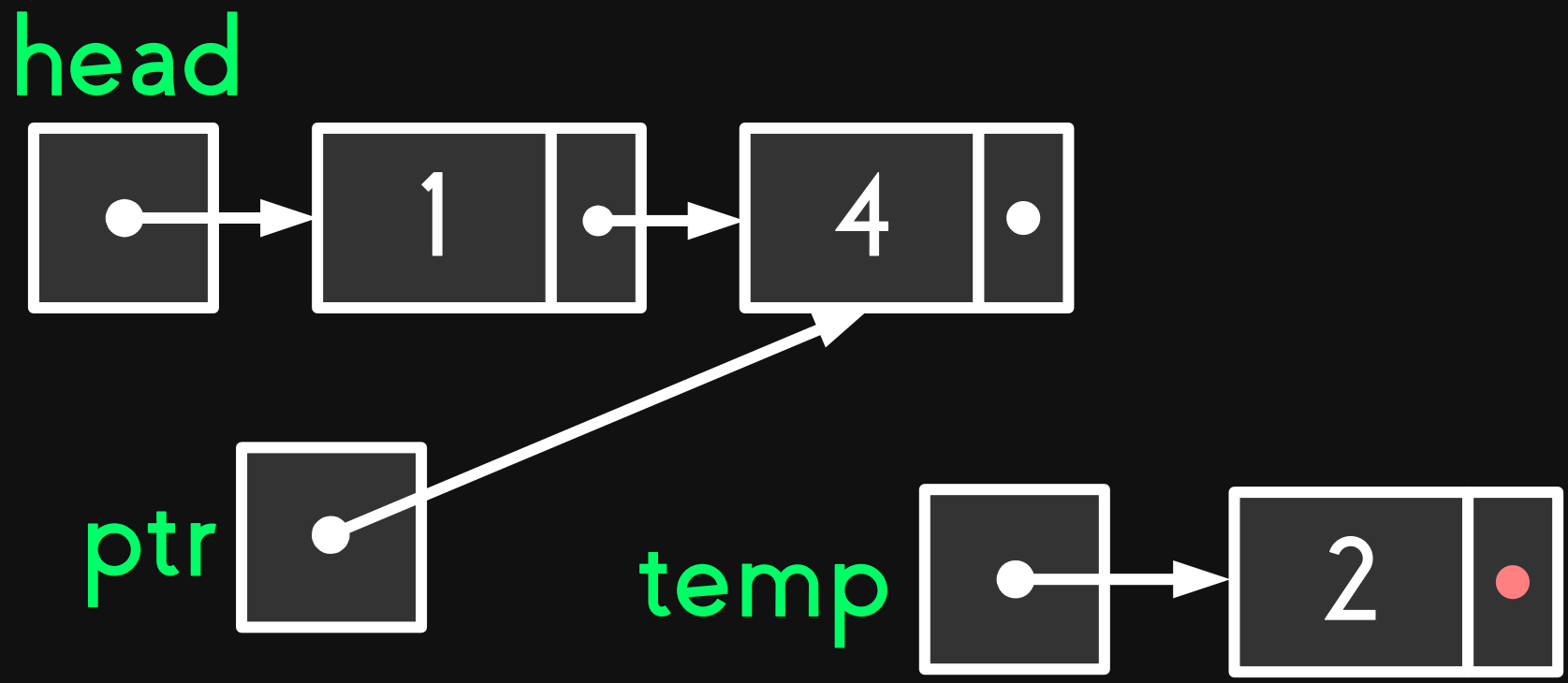
make the `next` pointer of the new node point to the node being pointed by the `next` of the node being pointed by `ptr`.

Insert at Tail



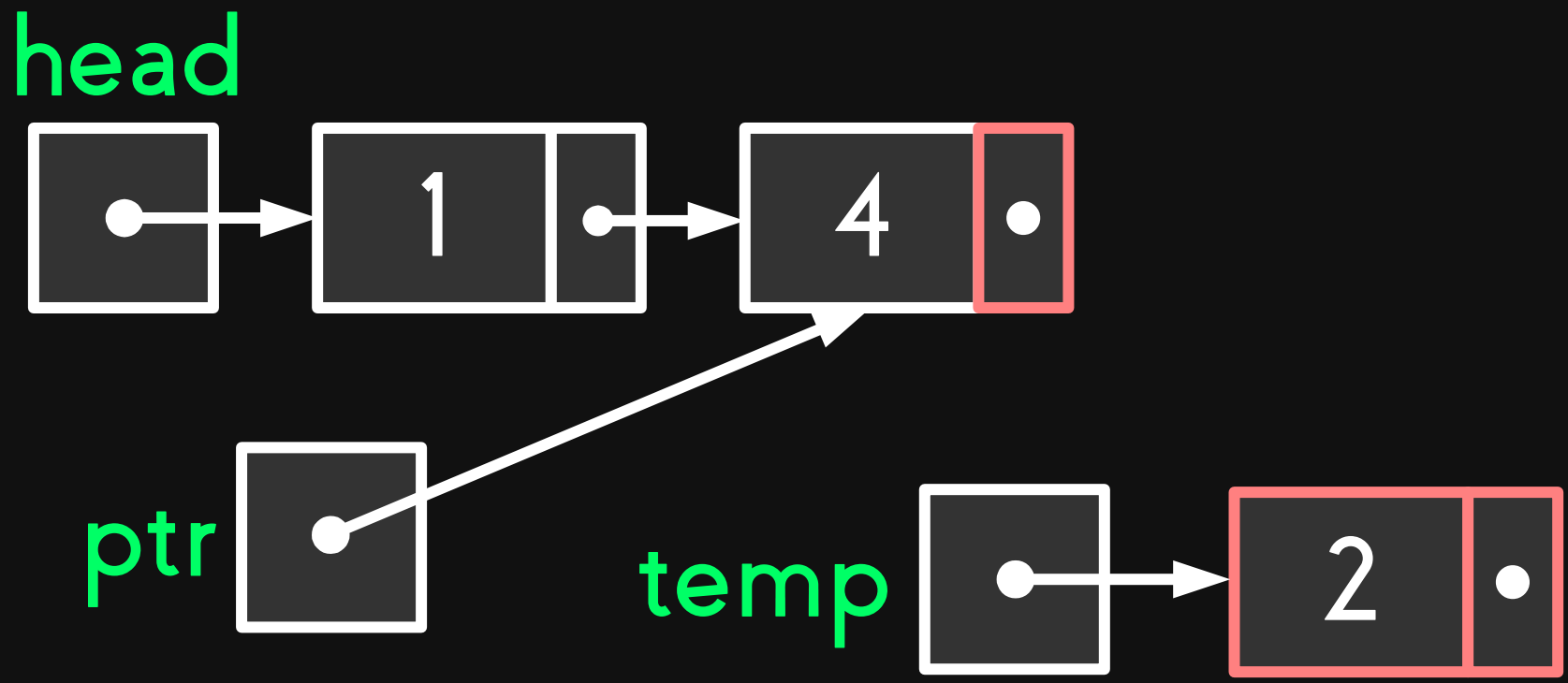
make the `next` pointer of the new node point to the node being pointed by the `next` of the node being pointed by `ptr`.

Insert at Tail



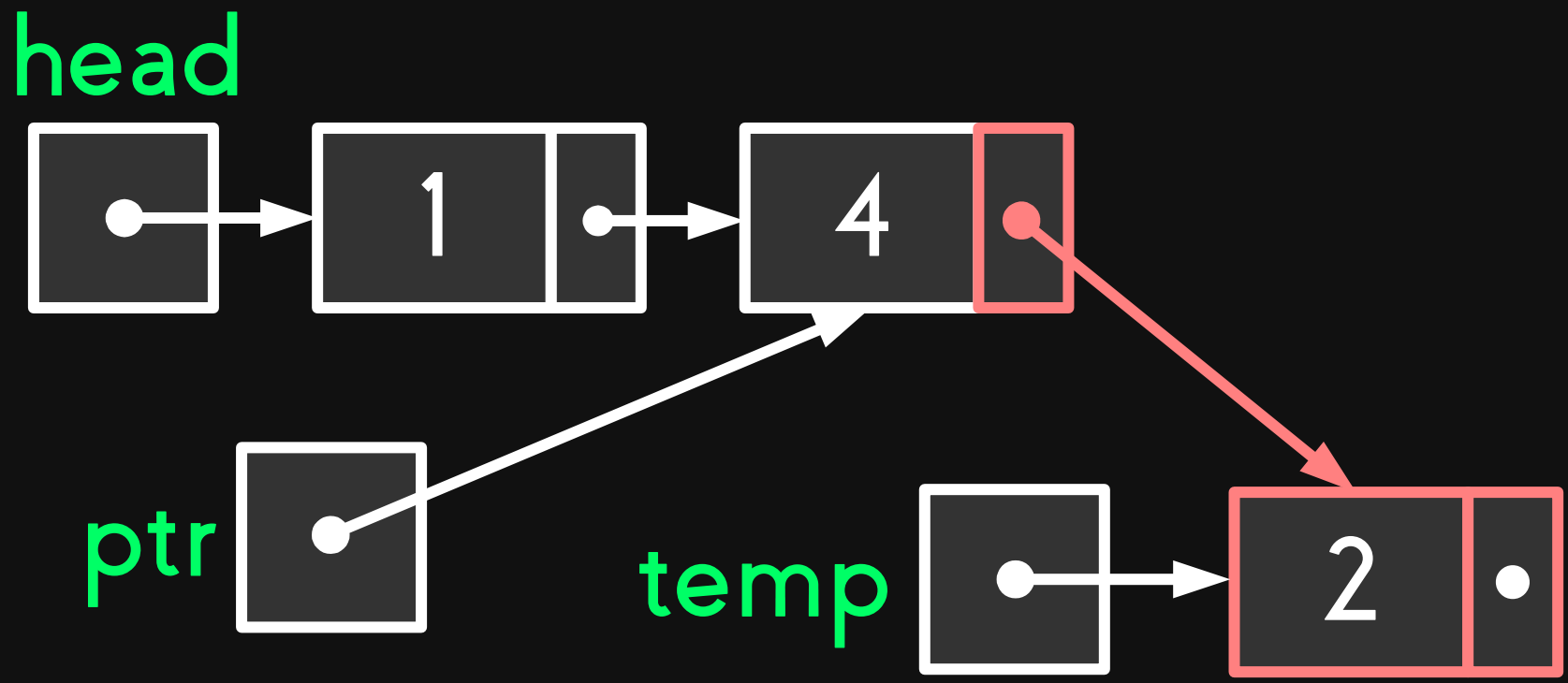
make the `next` pointer of the new node point to the node being pointed by the `next` of the node being pointed by `ptr`.

Insert at Tail



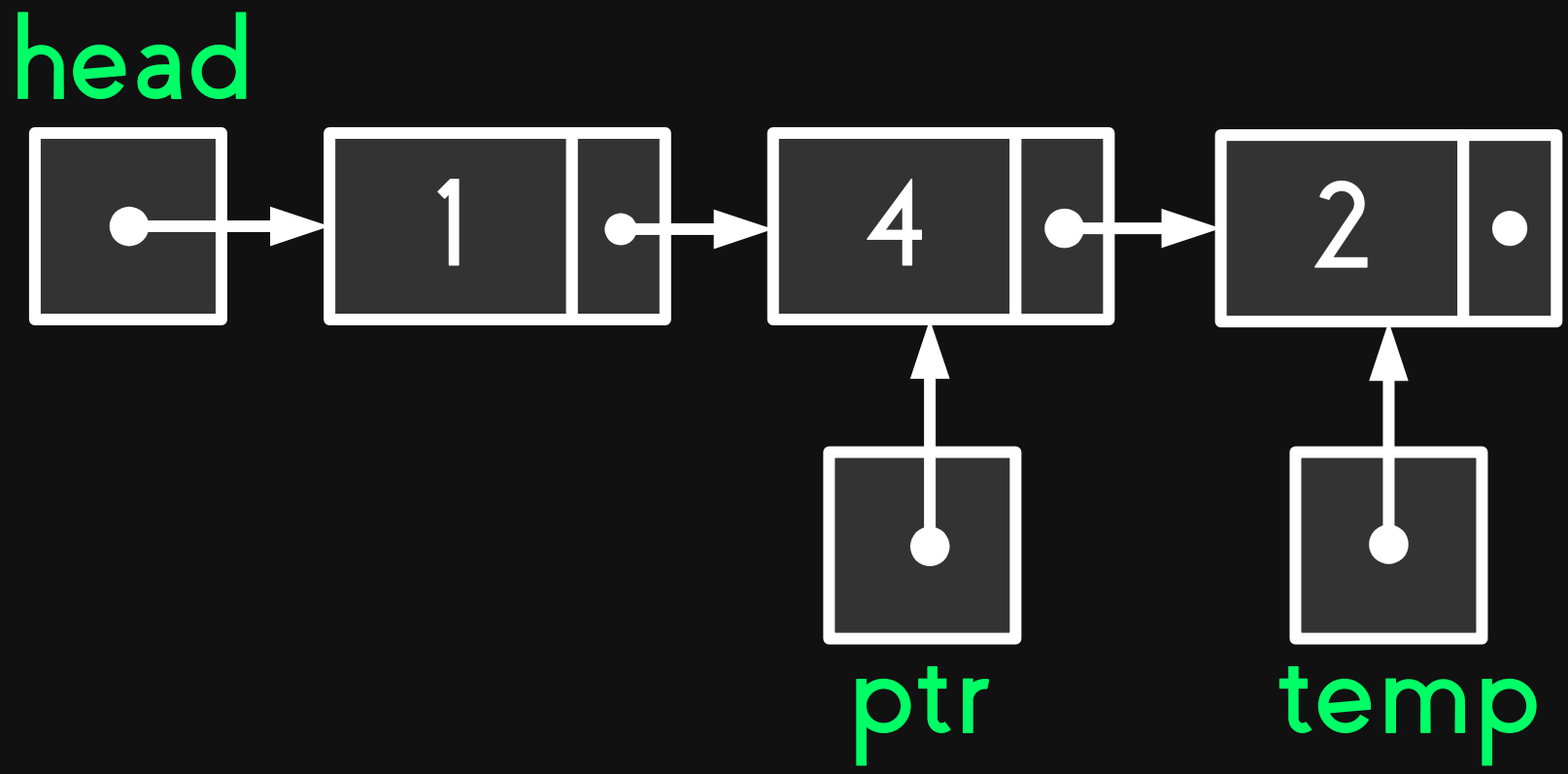
point the `next` of the node being
pointed by `ptr` to the new node

Insert at Tail



point the `next` of the node being
pointed by `ptr` to the new node

Insert at Tail



Rearrangement of the list.

CMSC 21

FUNDAMENTALS OF PROGRAMMING

Kristine Bernadette P. Pelaez

Institute of Computer Science
University of the Philippines Los Baños