CSc 352

Basic C Structs

Russell Lewis (sub for Benjamin Dicken)

Announcements

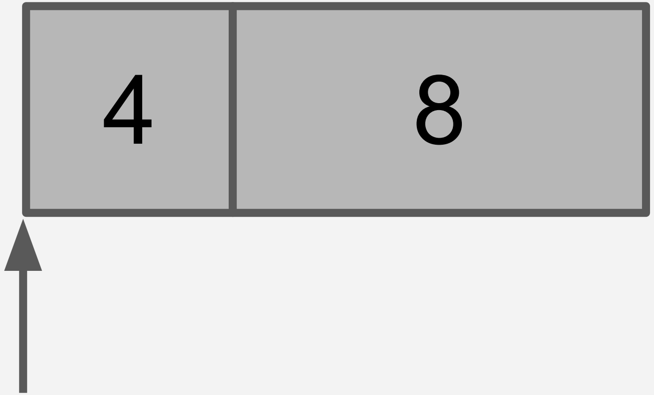
- Ben is on paternity leave
  - Back in 3 weeks (Apr 4)
- New Lecture form:
  - 50 minutes in class
  - 25 minutes by video (posted next day)
    - https://www.youtube.com/playlist?list=PL-F3IhGTDSSqe5cMDqrLdHkG0bleuA_xq
  - Slides posted on D2L
- My 352 Office Hours: 11am-noon, MWF
  - Online only this week, will be live+online next Mon (G/S 837)

Recap:

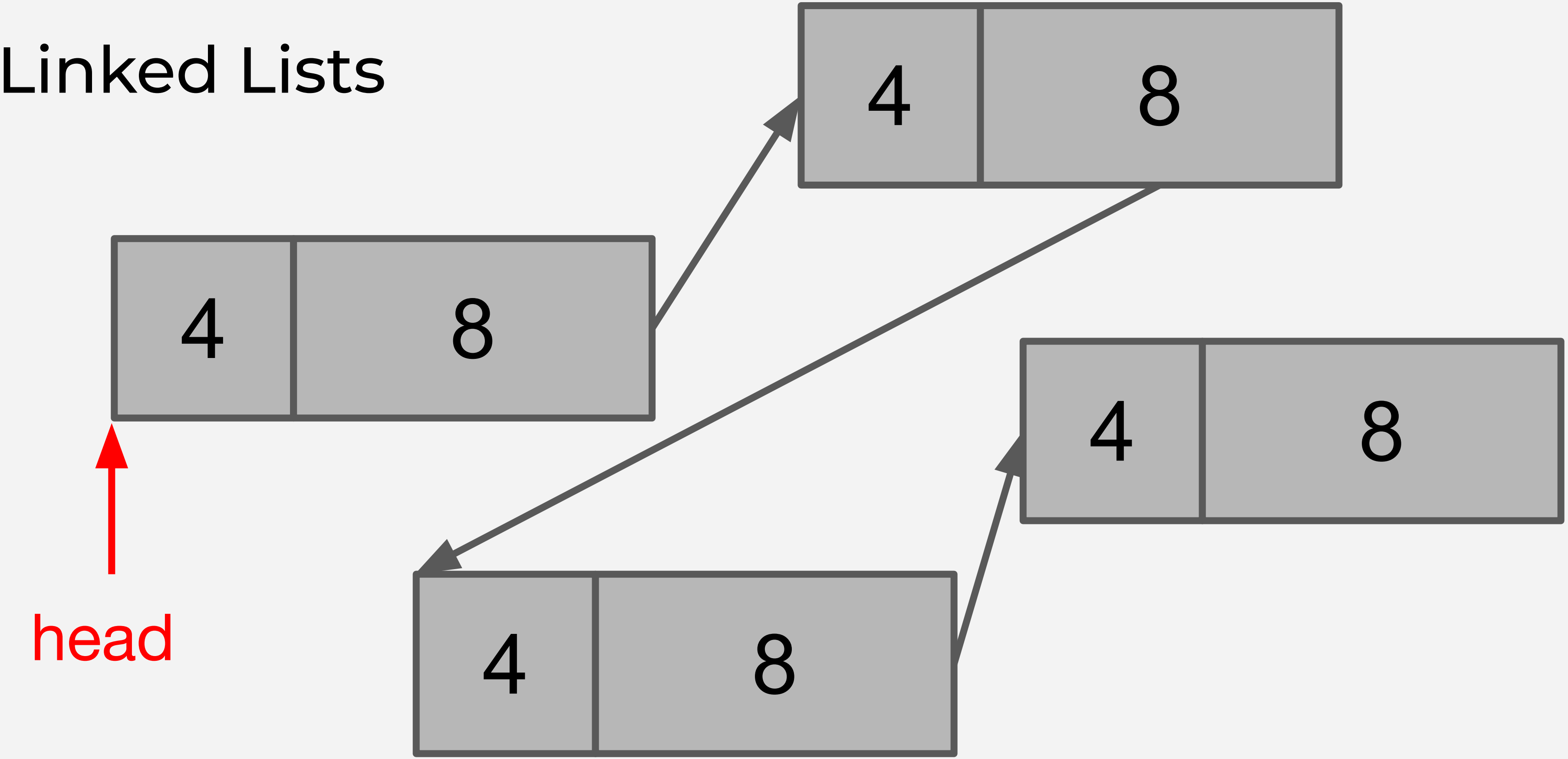
- `void* malloc(size_t size);`
  - Allocates **size** bytes and returns the pointer to it, or NULL if failed to alloc
- `void* calloc(size_t n_items, size_t size);`
  - Allocates **(n_items* size)** bytes and returns the pointer to it, or NULL if failed to alloc
- `void free(void * ptr);`
  - Frees the memory pointer to by **ptr** so that your program can no longer rely on having access to that memory

Reminder:

- Ben showed you a linked list based only on pointer arithmetic
    - `typedef void* lln;`
    - 4 bytes for integer data
    - 8 bytes for pointer

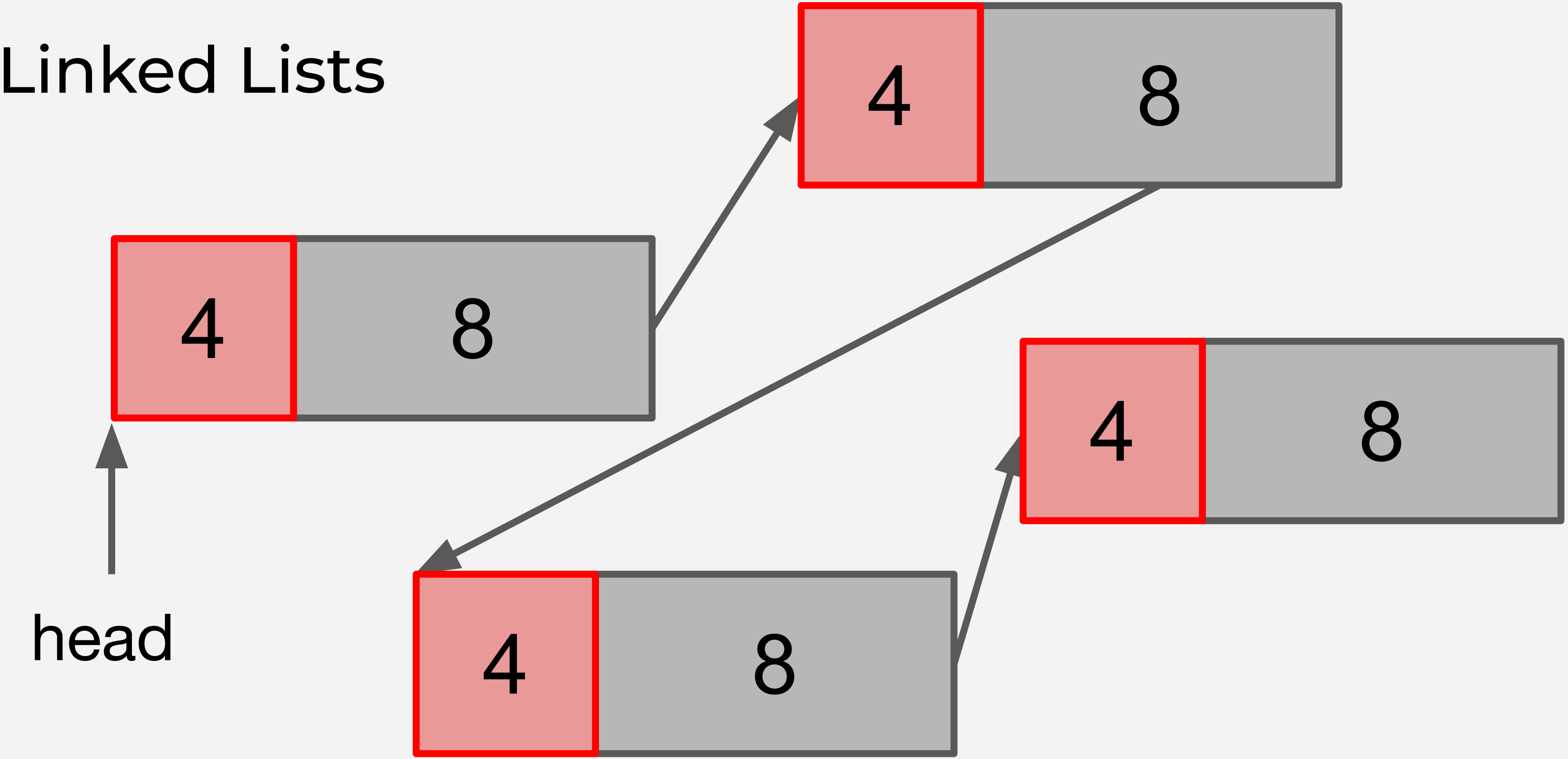


Linked Lists



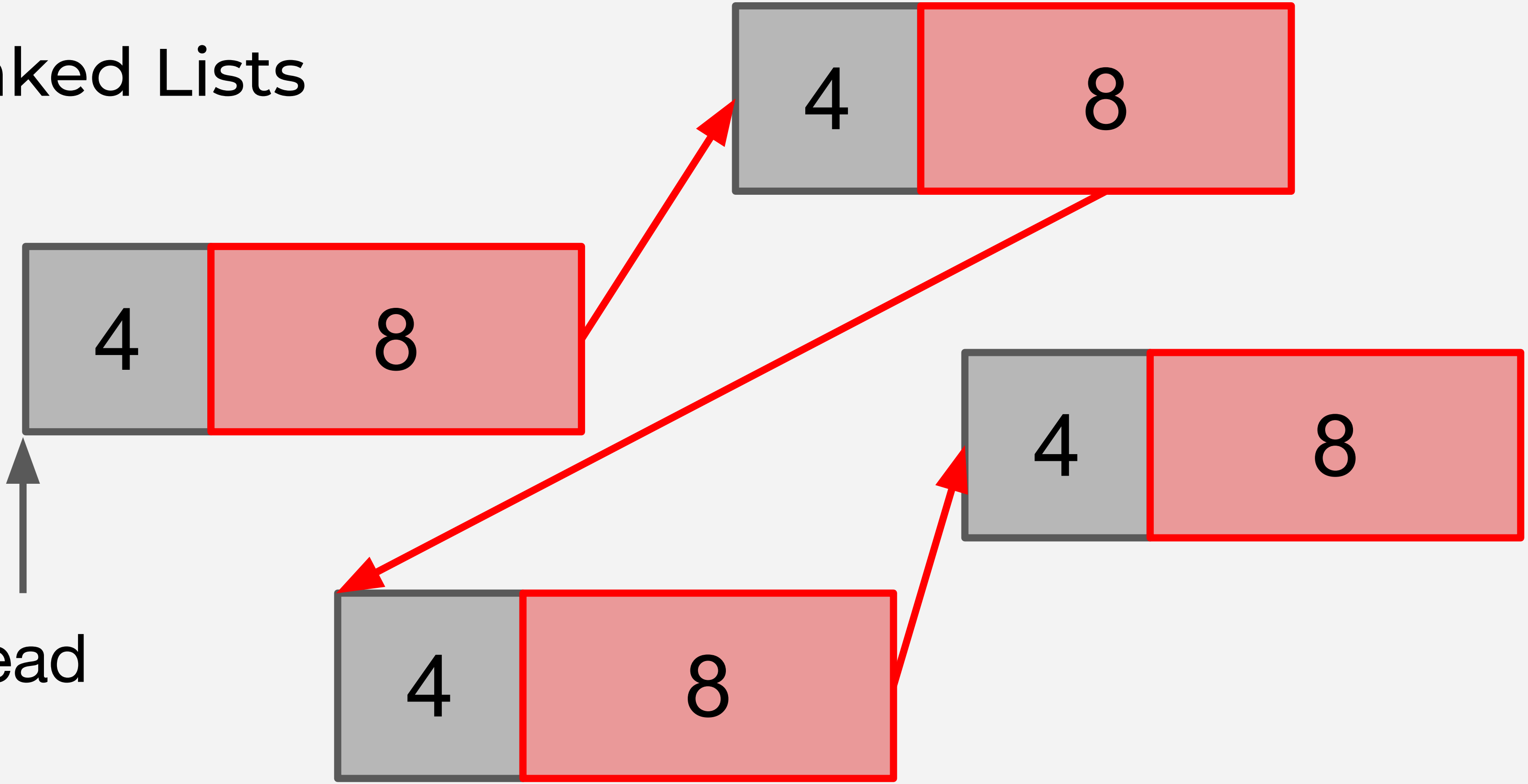
- Head pointer

Linked Lists



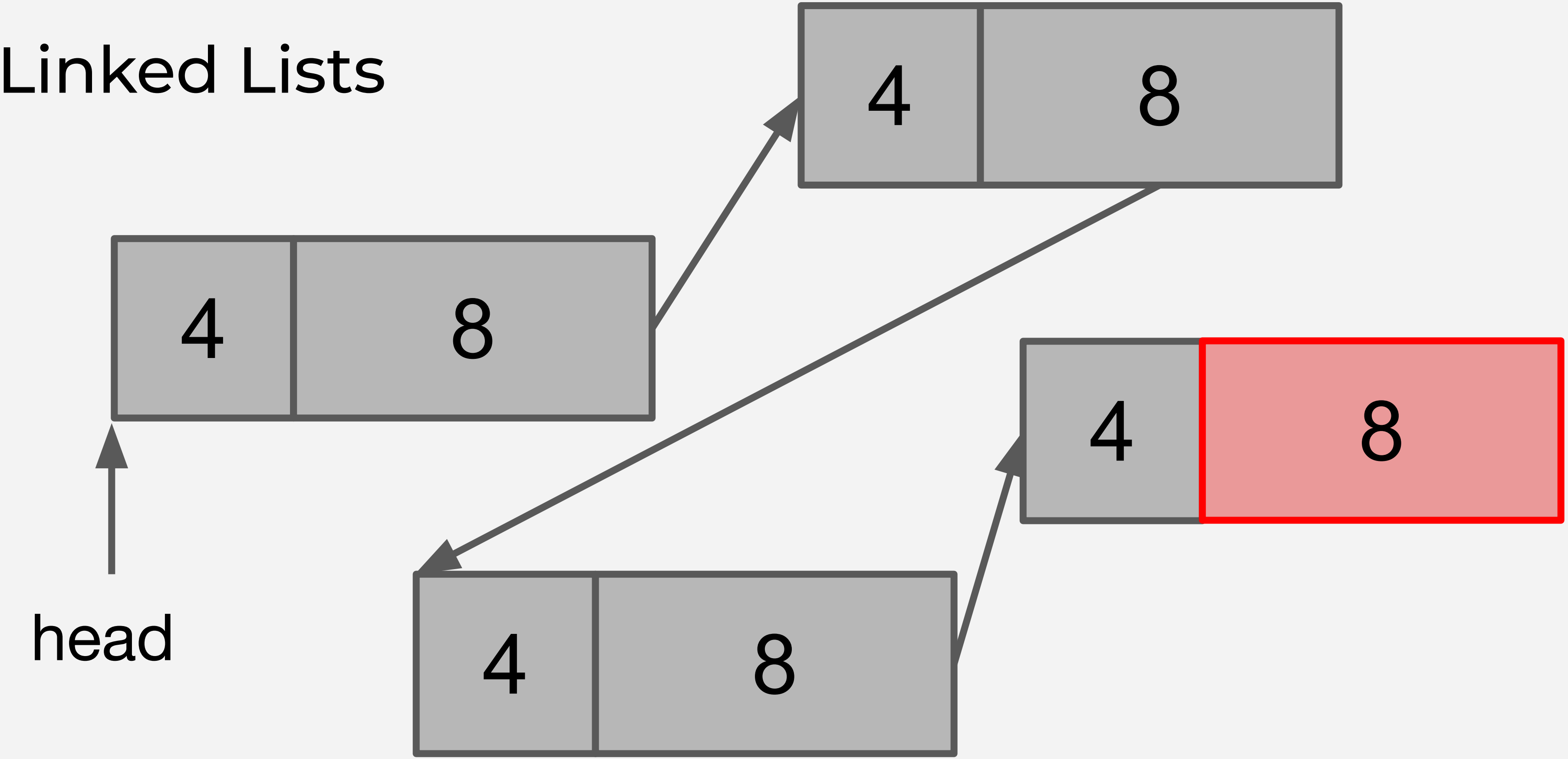
- Values

Linked Lists



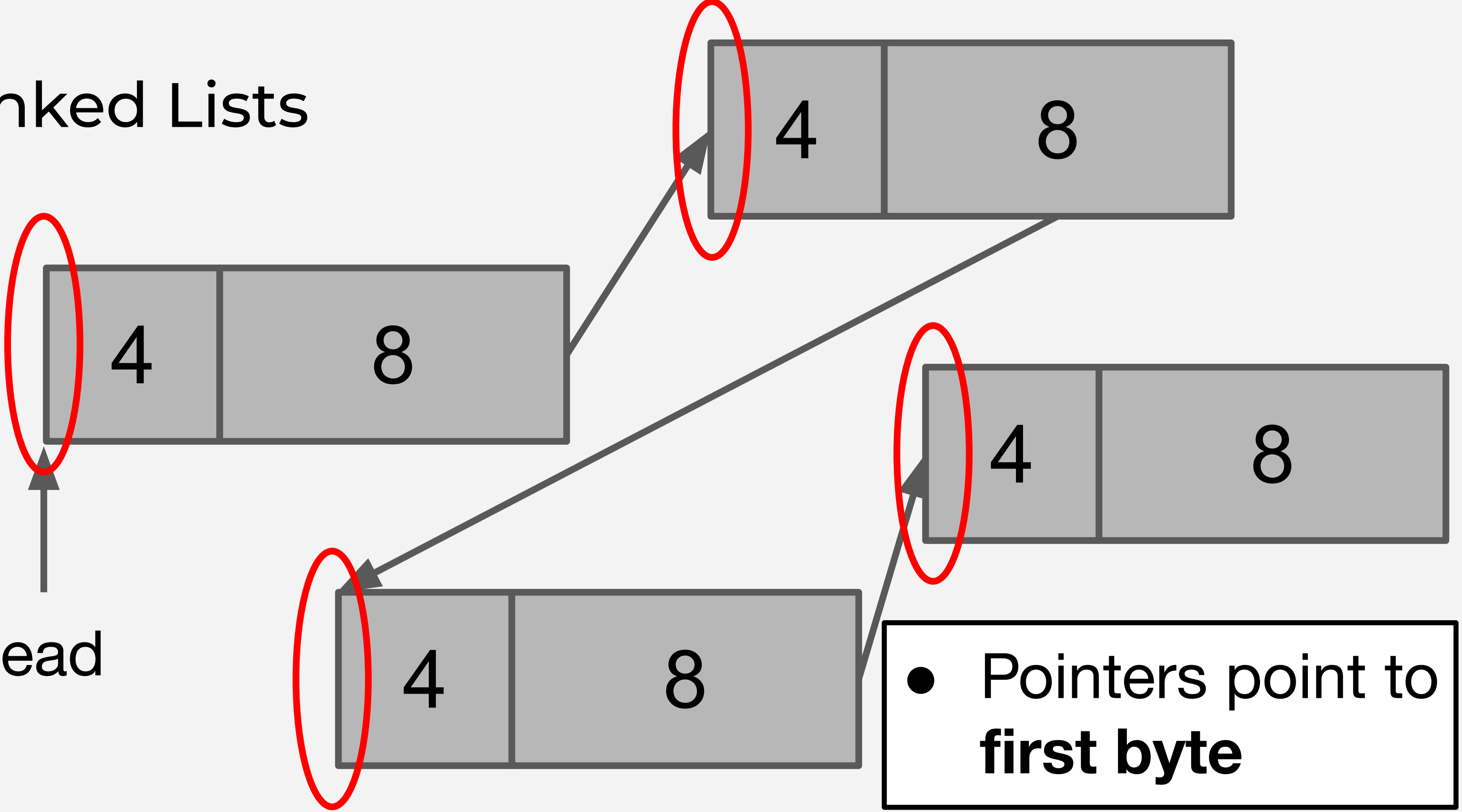
- Next pointers

Linked Lists



- NULL pointer

Linked Lists



- Pointers point to **first byte**

Activity

Linked Lists in Python, Java

- Write this function in both Python and Java (they will be pretty similar)

  ```
  lln_print(head)
  ```

  - head: input list (might be empty)
    - type `ListNode`
  - output looks like this:

    ```
    val1 -> val2 -> val3 -> END
    ```

  - loop encouraged

PYTHON

```
def lln_print(head):
    cur = head
    while cur is not None:
        print(f“{head.val} -> ”, end=“”)
        cur = cur.next
    print(“END”)
```

Java

```
void lln_print(ListNode head) {
    ListNode cur = head;
    while (cur != null) {
        System.out.print(str(head.val)+“ -> ”);
        cur = cur.next;
    }
    System.out.println(“END”);
}
```

Activity

Linked Lists in Python, Java

- Write this function in both Python and Java (they will be pretty similar)

  `lln_add_tail(head, val)`

  - head: input list (might be empty)
  - val: integer
  - returns updated list
  - assume `ListNode` class exists
    - call `ListNode(val)` to create a new node
    - `new ListNode(val)` in Java
  - recursion encouraged

PYTHON

```
def lln_add_tail(head, val):
    if head is None:
        return ListNode(val)
    head.next = lln_add_tail(head.next, val)
    return head
```

Java

```
ListNode lln_add_tail(ListNode head, int val)
{
    if (head == null)
        return new ListNode(val);
    head.next = lln_add_tail(head.next, val);
    return head;
}
```

Activity

Linked Lists in C

- Rewrite `lln_print(head)` in C - compare to your Java code
  - `ListNode*` parameter
    - We’ll declare this type later
    - Use arrow syntax to access fields:

      `cur->next`
  - `NULL`
  - `printf()`

C

```
void lln_print(ListNode *head) {
    ListNode *cur = head;
    while (cur != NULL) {
        printf(“%d -> ”, cur->val);
        cur = cur->next;
    }
    printf(“END\n”);
}
```

Java

```
void lln_print(ListNode head) {
    ListNode cur = head;
    while (cur != null) {
        System.out.print(str(head.val)+“ -> ”);
        cur = cur.next;
    }
    System.out.println(“END”);
}
```

Declaring a `struct`

- In C, a “**`struct`**” is a pattern for how to arrange variables in memory
- Used very much like `class`-es in Python, Java
- But no member functions
  - Therefore, no private data

Activity

Linked Lists in Python, Java

- Declare a `ListNode` class in Python and Java
  - Java (not Python): declare fields
  - Write a constructor, no other methods
    - Constructor takes `val` parameter
    - Initializes `val` field, sets `next` to `None/null`

PYTHON

```python
class ListNode:
    def __init__(self, val):
        self.val  = val
        self.next = None
```

Java

```java
class ListNode {
    public int      val;
    public ListNode next;

    public ListNode(int val) {
        this.val  = val;
        this.next = null;
    }
}
```

Activity

Linked Lists in C

- Declare a `ListNode` **struct** in C

- Basically follow the Java pattern, except:
  - `struct` instead of `class`
  - No `public` keyword
  - No member functions (including constructor)

NOTE:
There are a couple more details as well, but we'll see them in my solution.
You don't need to know them yet.

C

```c
typedef struct ListNode {
    int             val;
    struct ListNode *next;
} ListNode;
```

Java

```java
class ListNode {
    public int      val;
    public ListNode next;

    public ListNode(int val) {
        this.val  = val;
        this.next = null;
    }
}
```

C

```
typedef struct ListNode {
    int             val;
    struct ListNode *next;
} ListNode;
```

Java

```
class ListNode {
    public int      val;
    public ListNode next;

    public ListNode(int val) {
        this.val  = val;
        this.next = null;
    }
}
```

- `struct` instead of `class`

C

```c
typedef struct ListNode {
    int             val;
    struct ListNode *next;
} ListNode;
```

- Pointer instead of reference

Java

```java
class ListNode {
    public int      val;
    public ListNode next;

    public ListNode(int val) {
        this.val  = val;
        this.next = null;
    }
}
```

C

```
typedef struct ListNode {
    int                 val;
    struct ListNode *next;
} ListNode;
```

- No member functions

Java

```
class ListNode {
    public int      val;
    public ListNode next;

    public ListNode(int val) {
        this.val  = val;
        this.next = null;
    }
}
```

C

```
typedef struct ListNode {
    int              val;
    struct ListNode *next;
} ListNode;
```

- No public / private

Java

```
class ListNode {
    public int      val;
    public ListNode next;

    public ListNode(int val) {
        this.val  = val;
        this.next = null;
    }
}
```

C

```c
typedef struct ListNode {
    int              val;
    struct ListNode *next;
} ListNode;
```

- Trailing semicolon **req’d**

Java

```java
class ListNode {
    public int      val;
    public ListNode next;

    public ListNode(int val) {
        this.val  = val;
        this.next = null;
    }
}
```

Apologies!

- The last difference is a ***misfeature***
- Was removed in C++
- Many C/C++ compilers allow you to skip this
- But it’s technically still part of the language spec

… so here goes …

C

```c
typedef struct ListNode {
    int             val;
    struct ListNode *next;
} ListNode;
```

Java

```java
class ListNode {
    public int      val;
    public ListNode next;

    public ListNode(int val) {
        this.val  = val;
        this.next = null;
    }
}
```

- `struct` prefix when using the type

C

```c
typedef struct ListNode {
    int             val;
    struct ListNode *next;
} ListNode;
```

- `typedef` means users won't need the `struct` prefix

Java

```java
class ListNode {
    public int      val;
    public ListNode next;

    public ListNode(int val) {
        this.val  = val;
        this.next = null;
    }
}
```

C

```
typedef struct ListNode ListNode;

struct ListNode {
    int         val;
    ListNode *next;
};
```

- If you want, you can do the `typedef` beforehand
- Don’t need the `struct` prefix inside, if you do.

Constructors & Functions

- `struct`-s don’t have constructors or other member functions
- But still very important to practice encapsulation!

- C often uses functions that take a struct pointer as the first parameter
  - Like `self` in Python, or (implicit) `this` in Java

`malloc()` and `free()`

- `malloc()` doesn’t care what you are going to use the memory for - so you can use it to allocate a struct on the heap
  - Use `sizeof()` to find the right size
  - `malloc()` returns `void*`, save it into a pointer of your choice
  - Then fill in fields using arrow syntax: `obj->field`

- Use `free()` to deallocate memory

Activity

Linked Lists in C

- Write a function, `lln_create()`, which `malloc()`s a `ListNode`, fills in the fields (like the constructors in Python,Java), and returns the new object
  - What parameters? What return value?
- Write a function, `lln_destroy()`, which `free()`s a `ListNode`.
  - What parameters? What return value?
  - Should you destroy the rest of the list, too?

```
typedef struct ListNode {
    int              val;
    struct ListNode *next;
} ListNode;
```

```
ListNode *lln_create(int val) {
    ListNode *retval = malloc(sizeof(ListNode));
    if (retval == NULL)
        return NULL;
    retval->val  = val;
    retval->next = NULL;
    return retval;
}

void lln_destroy(ListNode *node) {
    free(node);
}
```