SFU

Linked Lists Part II

CMPT 125
Mo Chen
SFU Computing Science
28/2/2020

Lecture 19

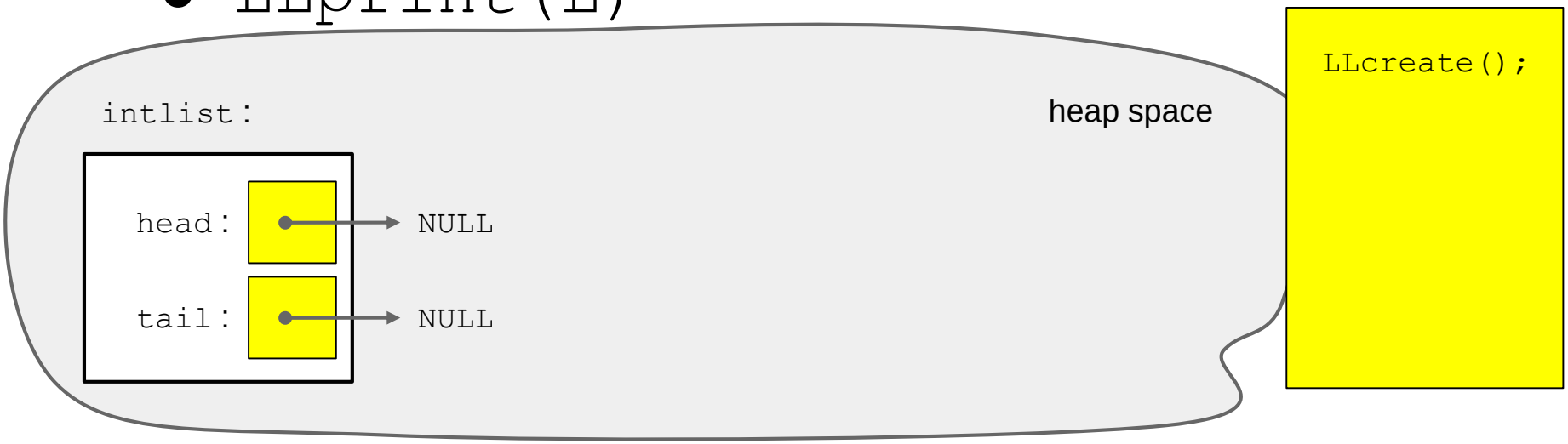
- Linked list implementation, continued
  - `create()`
  - `append()`
  - `print()`
  - `search()`

Linked Lists

A sequence of data / pointer nodes

Maintain pointers to the head and tail

- `LLcreate()`
- `LLappend(L, x)`
- `LLprint(L)`



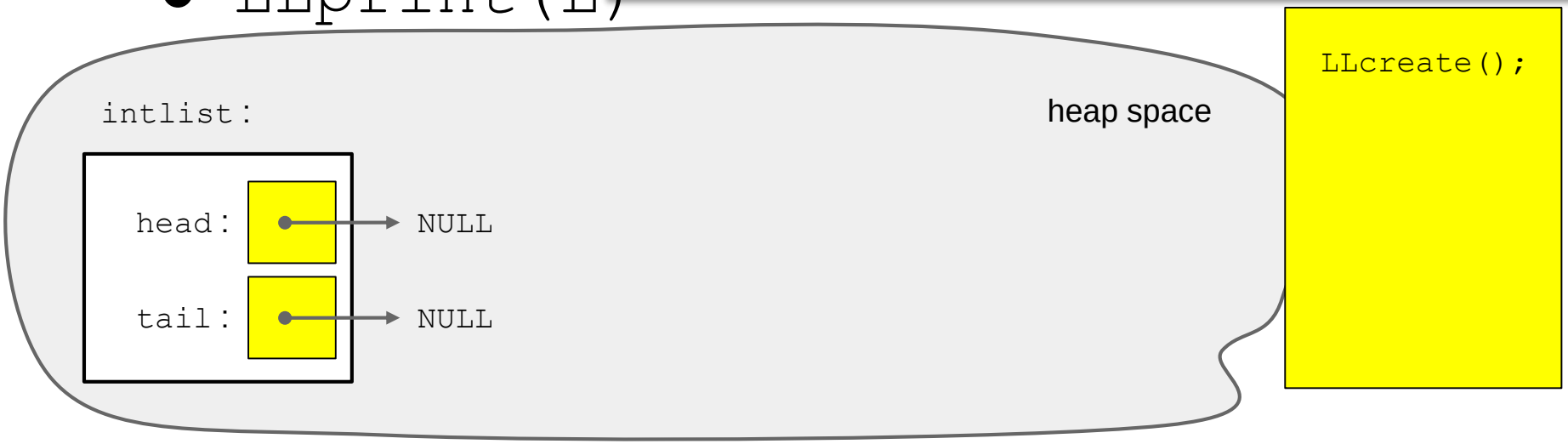
Linked Lists

A sequence of da

Maintain pointers

- `LLcreate(`
- `LLappend(`
- `LLprint(L)`

```c
// LL.c
#include "LL.h"
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>

// Creates and returns a new, empty list
LL_t * LLcreate() {
    LL_t * ret = malloc(sizeof(LL_t));
    if (ret != NULL) {
        ret->head = NULL;
        ret->tail = NULL;
    }
    return ret;
}
```



Code

```c
// LL-node.h
typedef struct _node {
        int data;
        struct _node * next;
} node_t;
```

```c
// LL.h
#include "LL-node.h"

typedef struct {
    node_t * head;
    node_t * tail;
} LL_t;

// Creates and returns a new, empty list
LL_t * LLcreate();
```

```c
// driver.c
#include <stdio.h>
#include "LL.h"


int main () {
    LL_t * intlist = LLcreate();
    if (intlist == NULL) {
        return 1;
    }
```

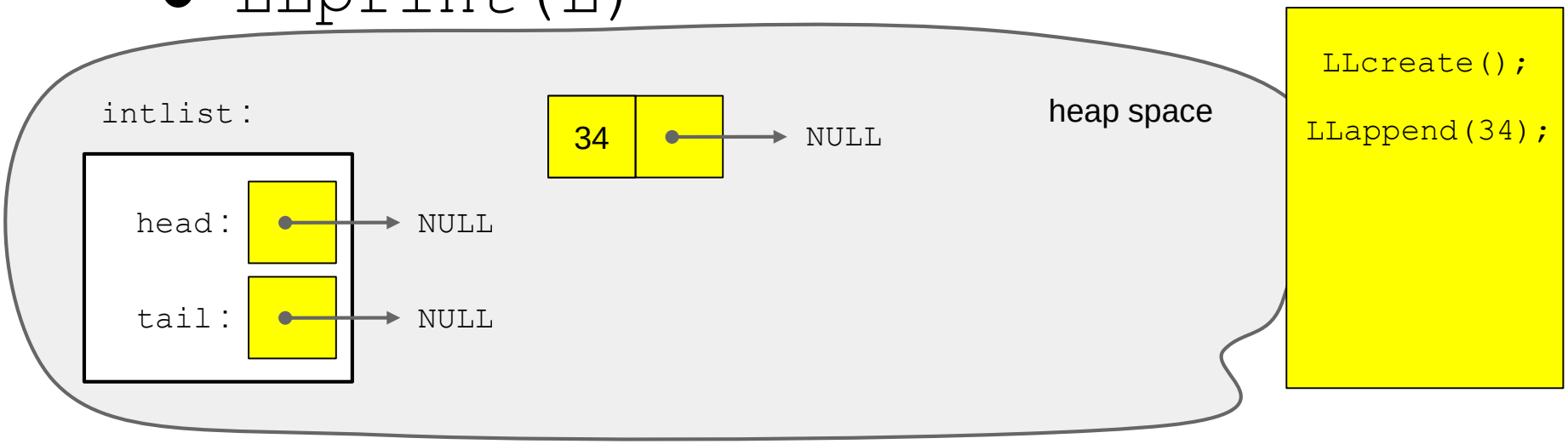
```c
// LL.c
#include "LL.h"
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>

// Creates and returns a new, empty list
LL_t * LLcreate() {
    LL_t * ret = malloc(sizeof(LL_t));
    if (ret != NULL) {
        ret->head = NULL;
        ret->tail = NULL;
    }
    return ret;
}
```

Linked Lists

A sequence of data / pointer nodes

Maintain pointers to the head and tail

- `LLcreate()`
- `LLappend(L, x)`
- `LLprint(L)`



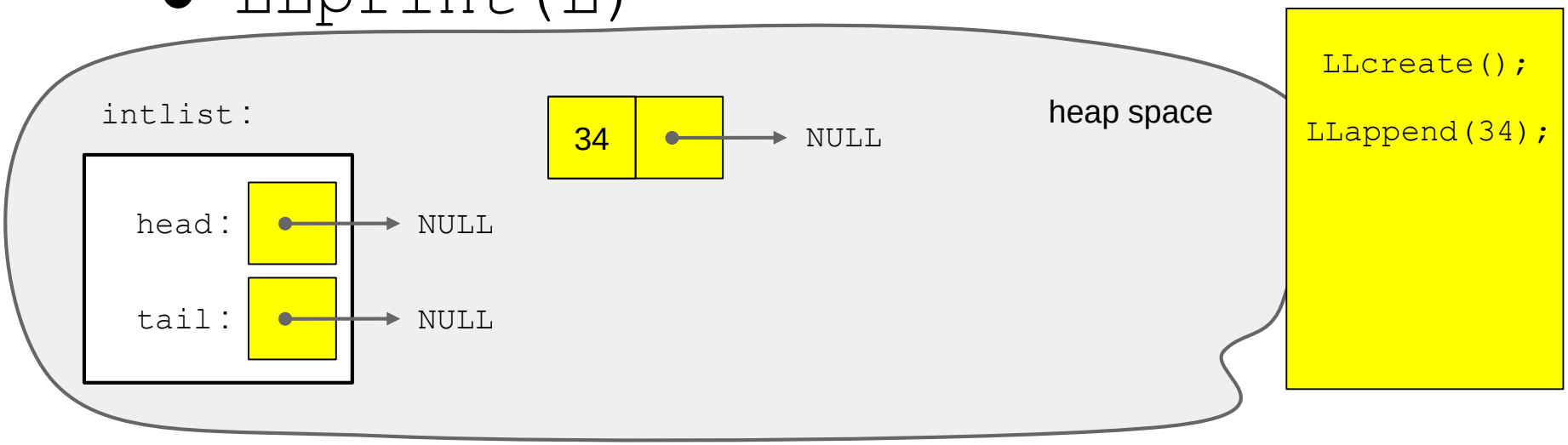
Linked Lists

A sequence of data / pointer nodes

Main

```
// Adds a new element to the tail end of a list
void LLappend(LL_t * intlist, int value) {
    node_t * newNode = malloc(sizeof(node_t));
    if (newNode != NULL) {
        newNode->data = value;
        newNode->next = NULL;
```

- LLappend(L, x)
- LLprint(L)

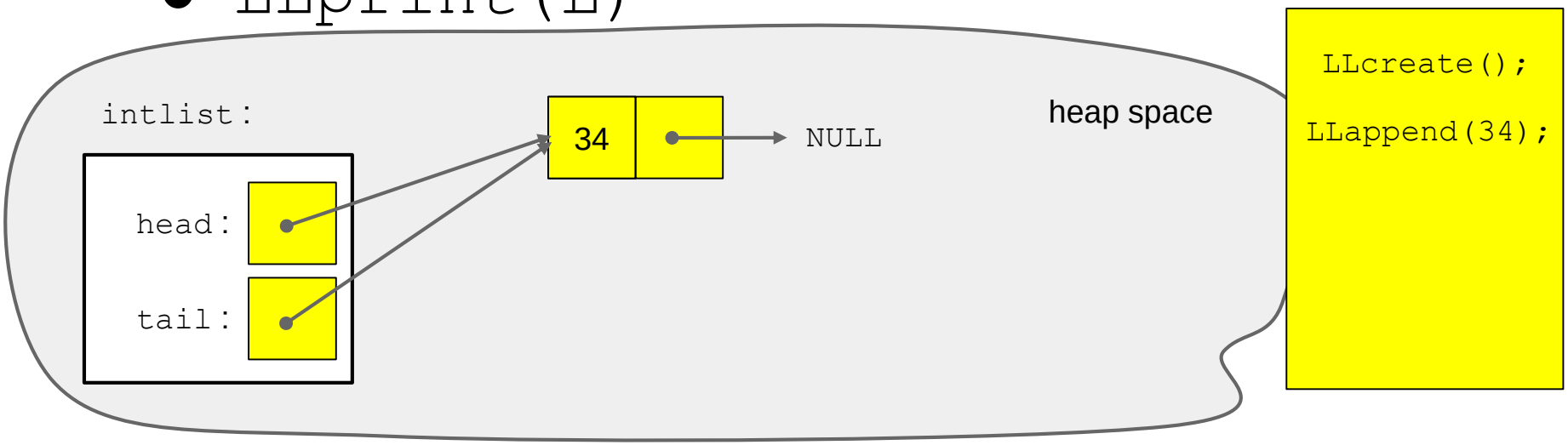


Linked Lists

A sequence of data / pointer nodes

Maintain pointers to the head and tail

- `LLcreate()`
- `LLappend(L, x)`
- `LLprint(L)`

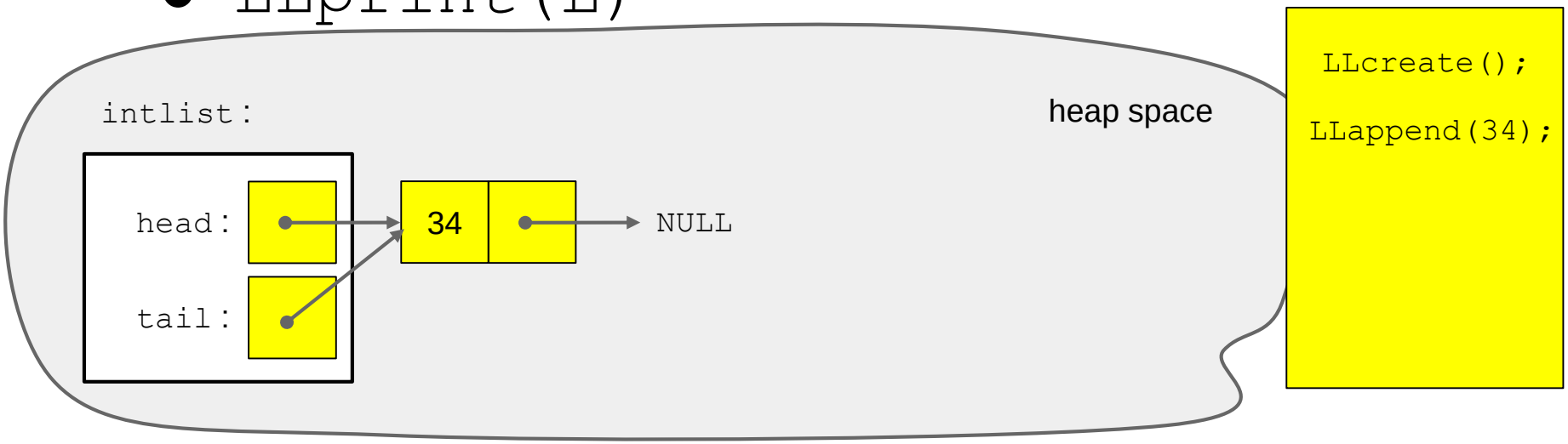


Linked Lists

A sequence of data / pointer nodes

Maintain pointers to the head and tail

- `LLcreate()`
- `LLappend(L, x)`
- `LLprint(L)`



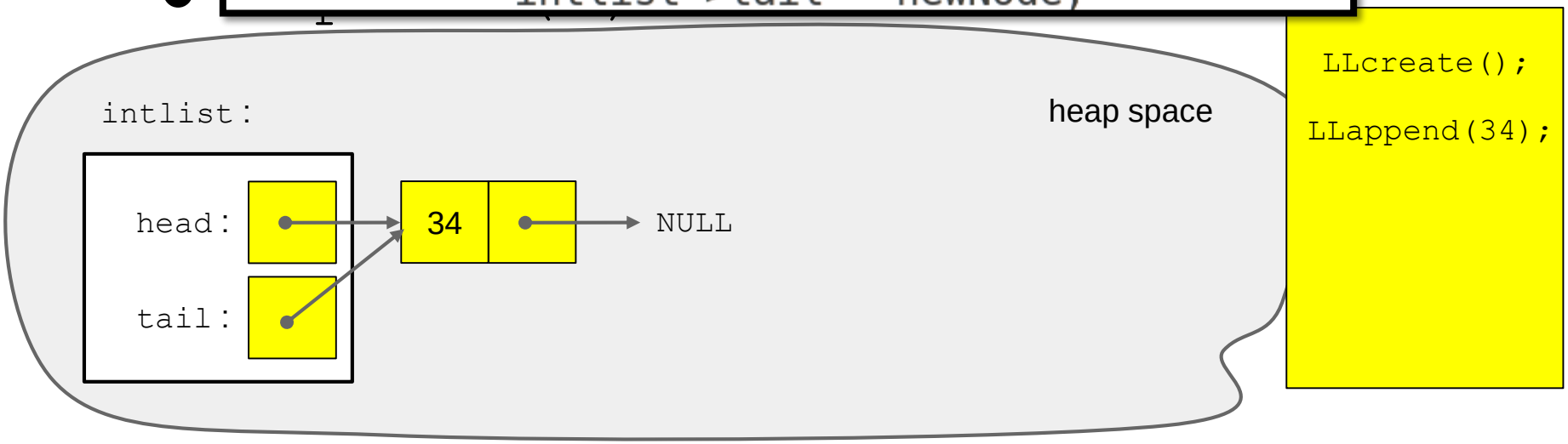
Linked Lists

A sec

Main

-
-
-

```
// Adds a new element to the tail end of a list
void LLappend(LL_t * intlist, int value) {
    node_t * newNode = malloc(sizeof(node_t));
    if (newNode != NULL) {
        newNode->data = value;
        newNode->next = NULL;
        if (intlist->tail == NULL) {
            // empty list
            assert(intlist->head == NULL);
            intlist->head = newNode;
            intlist->tail = newNode;
```



```c
// LL.h
#include "LL-node.h"

typedef struct {
    node_t * head;
    node_t * tail;
} LL_t;

// Creates and returns a new, empty list
LL_t * LLcreate();

// Adds a new element to the tail end of a list
void LLappend(LL_t * intlist, int value);
```

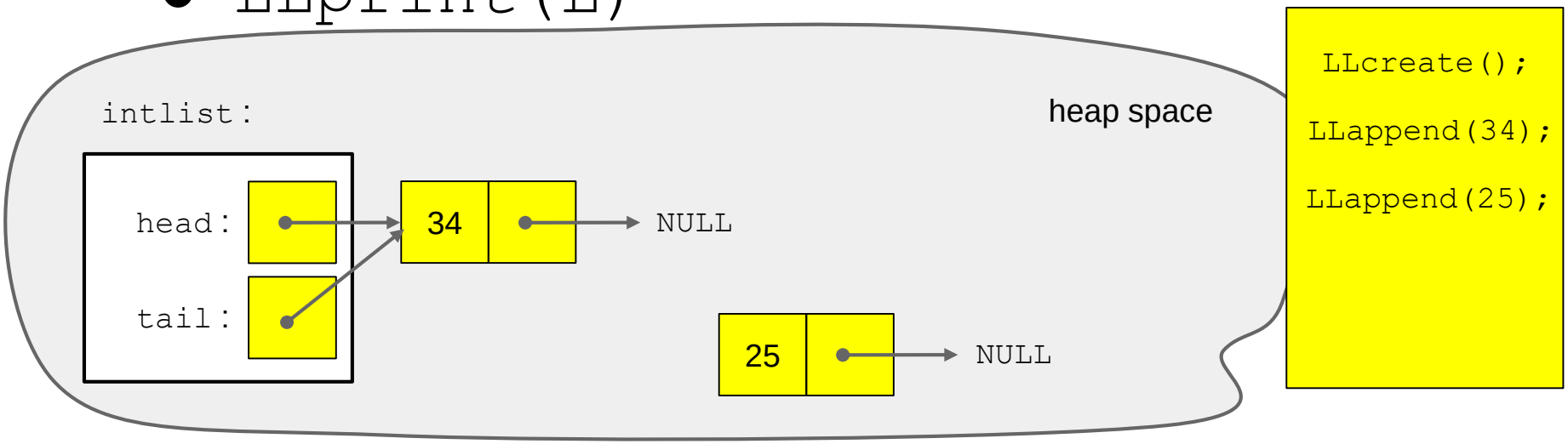
```c
// LL.c
#include "LL.h"
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>

// Creates and returns a new, empty list
LL_t * LLcreate() {
    LL_t * ret = malloc(sizeof(LL_t));
    if (ret != NULL) {
        ret->head = NULL;
        ret->tail = NULL;
    }
    return ret;
}


// Adds a new element to the tail end of a list
void LLappend(LL_t * intlist, int value) {
    node_t * newNode = malloc(sizeof(node_t));
    if (newNode != NULL) {
        newNode->data = value;
        newNode->next = NULL;
        if (intlist->tail == NULL) {
            // empty list
            assert(intlist->head == NULL);
            intlist->head = newNode;
            intlist->tail = newNode;
```

Linked Lists

A sequence of data / pointer nodes

Maintain pointers to the head and tail

- `LLcreate()`
- `LLappend(L, x)`
- `LLprint(L)`

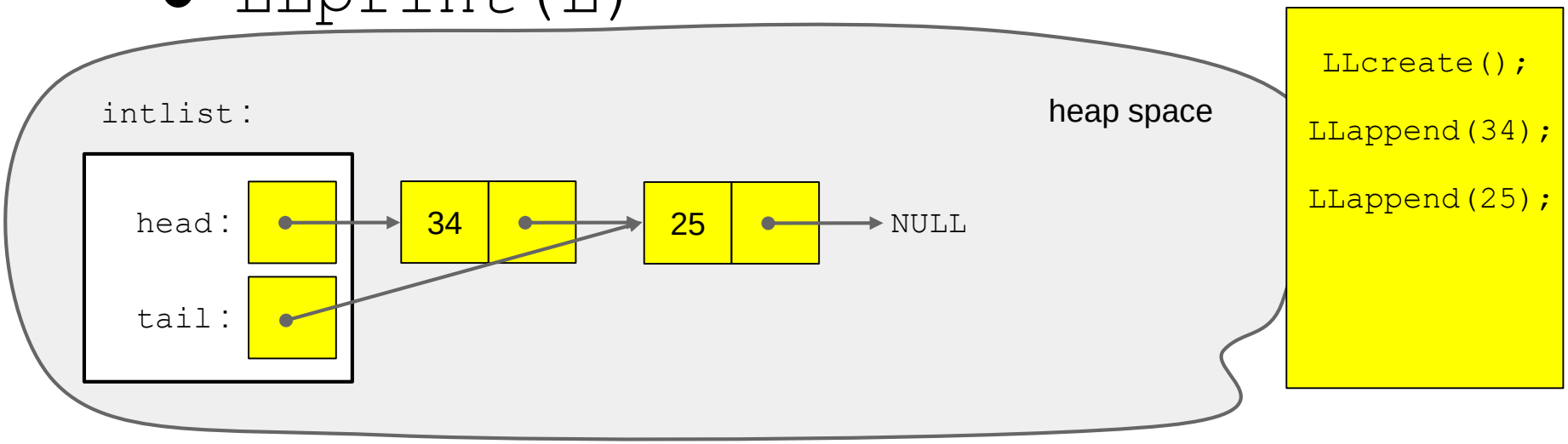


Linked Lists

A sequence of data / pointer nodes

Maintain pointers to the head and tail

- `LLcreate()`
- `LLappend(L, x)`
- `LLprint(L)`



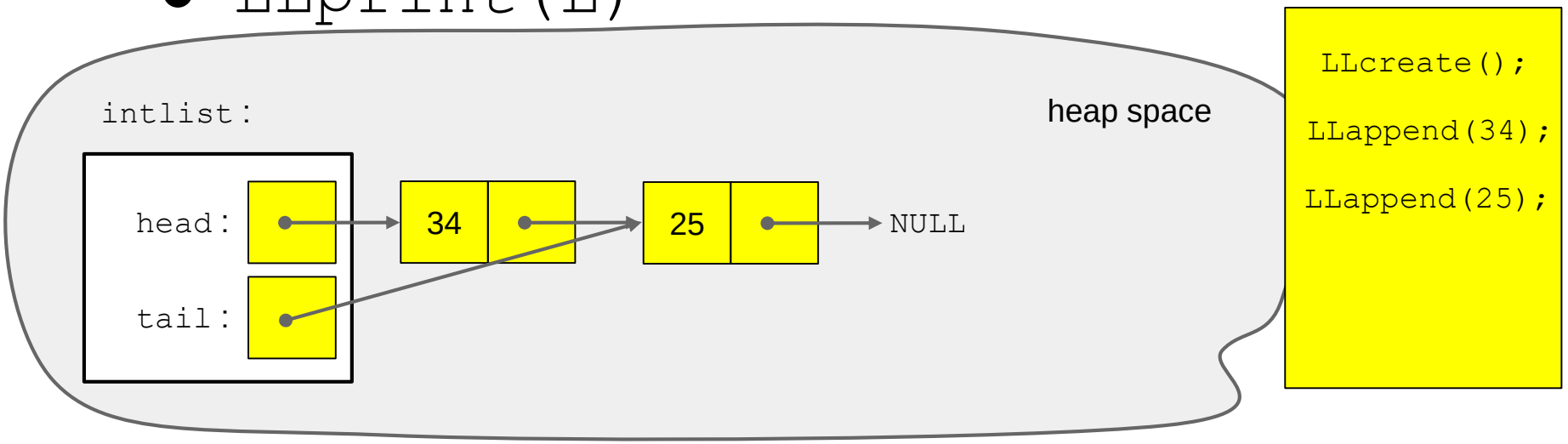
Linked Lists

A sequence of dat

Maintain pointers t

- `LLcreate()`
- `LLappend(I`
- `LLprint(L)`

```
// Adds a new element to the tail end of a list
void LLappend(LL_t * intlist, int value) {
    node_t * newNode = malloc(sizeof(node_t));
    if (newNode != NULL) {
        newNode->data = value;
        newNode->next = NULL;
        if (intlist->tail == NULL) {
            // empty list
            assert(intlist->head == NULL);
            intlist->head = newNode;
            intlist->tail = newNode;
        } else {
            // non empty list
            intlist->tail->next = newNode;
            intlist->tail = newNode;
        }
    }
}
```



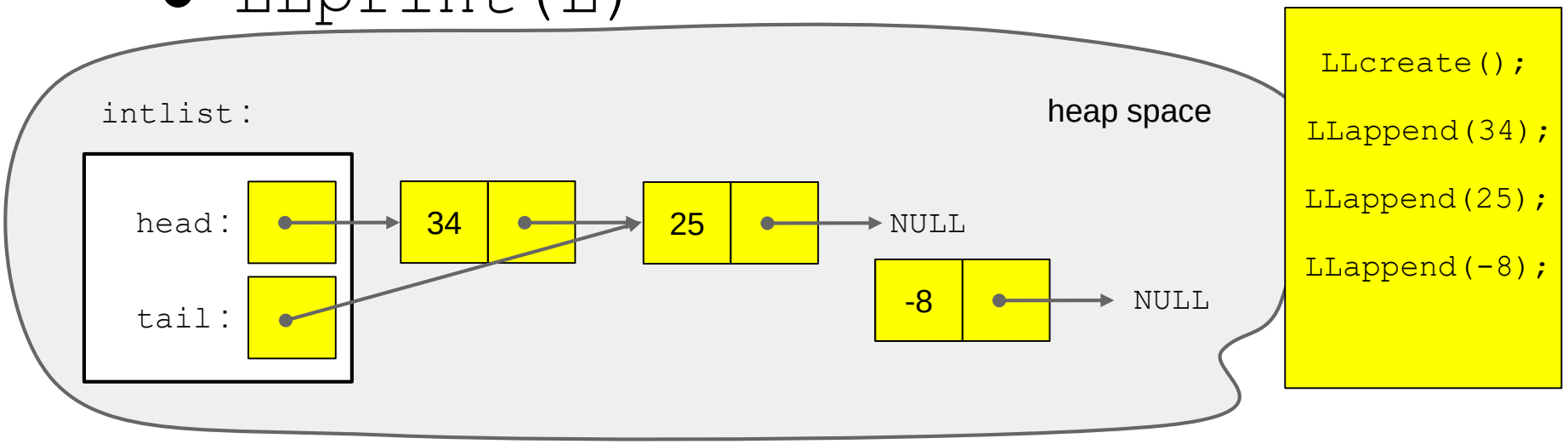
Linked Lists

A sequence of dat

Maintain pointers t

- `LLcreate()`
- `LLappend(I`
- `LLprint(L)`

```c
// Adds a new element to the tail end of a list
void LLappend(LL_t * intlist, int value) {
    node_t * newNode = malloc(sizeof(node_t));
    if (newNode != NULL) {
        newNode->data = value;
        newNode->next = NULL;
        if (intlist->tail == NULL) {
            // empty list
            assert(intlist->head == NULL);
            intlist->head = newNode;
            intlist->tail = newNode;
        } else {
            // non empty list
            intlist->tail->next = newNode;
            intlist->tail = newNode;
        }
    }
}
```



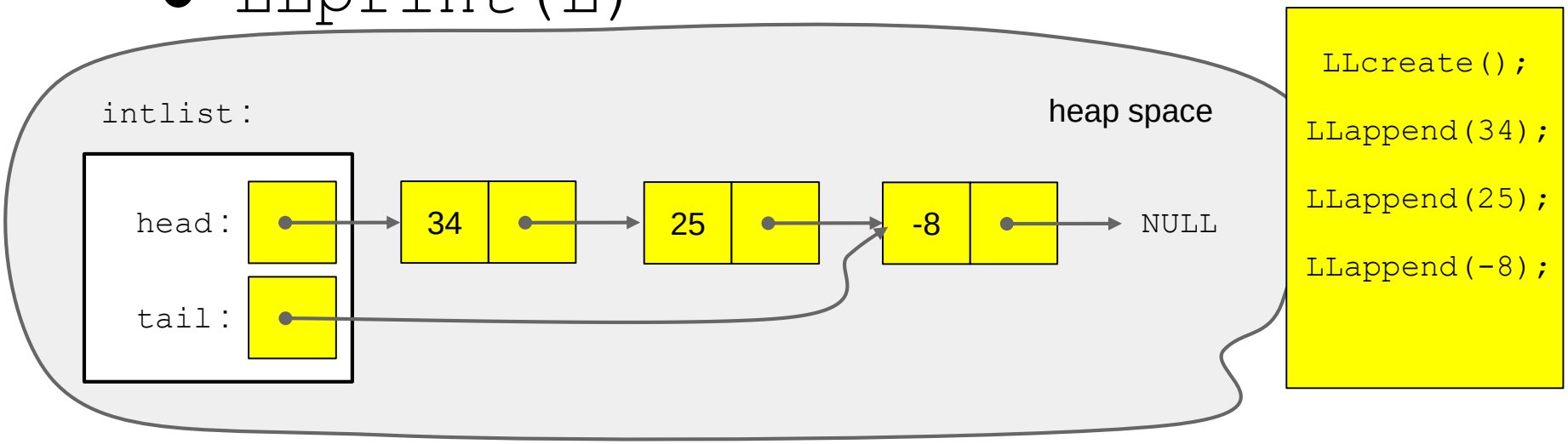
Linked Lists

A sequence of dat

Maintain pointers t

- `LLcreate()`
- `LLappend(I`
- `LLprint(L)`

```c
// Adds a new element to the tail end of a list
void LLappend(LL_t * intlist, int value) {
    node_t * newNode = malloc(sizeof(node_t));
    if (newNode != NULL) {
        newNode->data = value;
        newNode->next = NULL;
        if (intlist->tail == NULL) {
            // empty list
            assert(intlist->head == NULL);
            intlist->head = newNode;
            intlist->tail = newNode;
        } else {
            // non empty list
            intlist->tail->next = newNode;
            intlist->tail = newNode;
        }
    }
}
```



```c
// LL.h
#include "LL-node.h"

typedef struct {
    node_t * head;
    node_t * tail;
} LL_t;

// Creates and returns a new, empty list
LL_t * LLcreate();

// Adds a new element to the tail end of a list
void LLappend(LL_t * intlist, int value);
```

```c
// LL-node.h
typedef struct _node {
        int data;
        struct _node * next;
} node_t;
```

```c
// driver.c
#include <stdio.h>
#include "LL.h"


int main () {
    LL_t * intlist = LLcreate();
    if (intlist == NULL) {
        return 1;
    }

    LLappend(intlist, 34);
    LLappend(intlist, 25);
    LLappend(intlist, -8);
```

```c
// LL.c
#include "LL.h"
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>

// Creates and returns a new, empty list
LL_t * LLcreate() {
    LL_t * ret = malloc(sizeof(LL_t));
    if (ret != NULL) {
        ret->head = NULL;
        ret->tail = NULL;
    }
    return ret;
}


// Adds a new element to the tail end of a list
void LLappend(LL_t * intlist, int value) {
    node_t * newNode = malloc(sizeof(node_t));
    if (newNode != NULL) {
        newNode->data = value;
        newNode->next = NULL;
        if (intlist->tail == NULL) {
            // empty list
            assert(intlist->head == NULL);
            intlist->head = newNode;
            intlist->tail = newNode;
        } else {
            // non empty list
            intlist->tail->next = newNode;
            intlist->tail = newNode;
        }
    }
}
```

Linked Lists

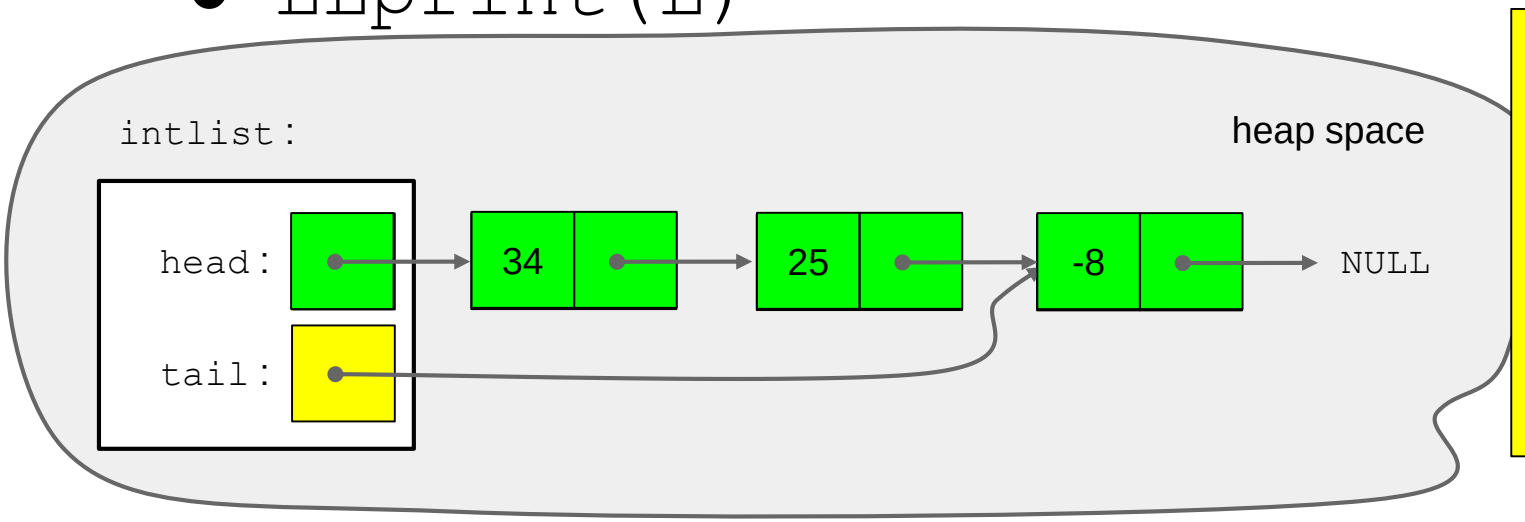
A sequence of data / pointer nodes

Maintain pointers to the head and tail

- `LLcreate()`
- `LLappend(L, x)`
- `LLprint(L)`

Output:

```
34  25  -8
```




```
LLcreate();
LLappend(34);
LLappend(25);
LLappend(-8);
LLprint();
```

Linked Lists

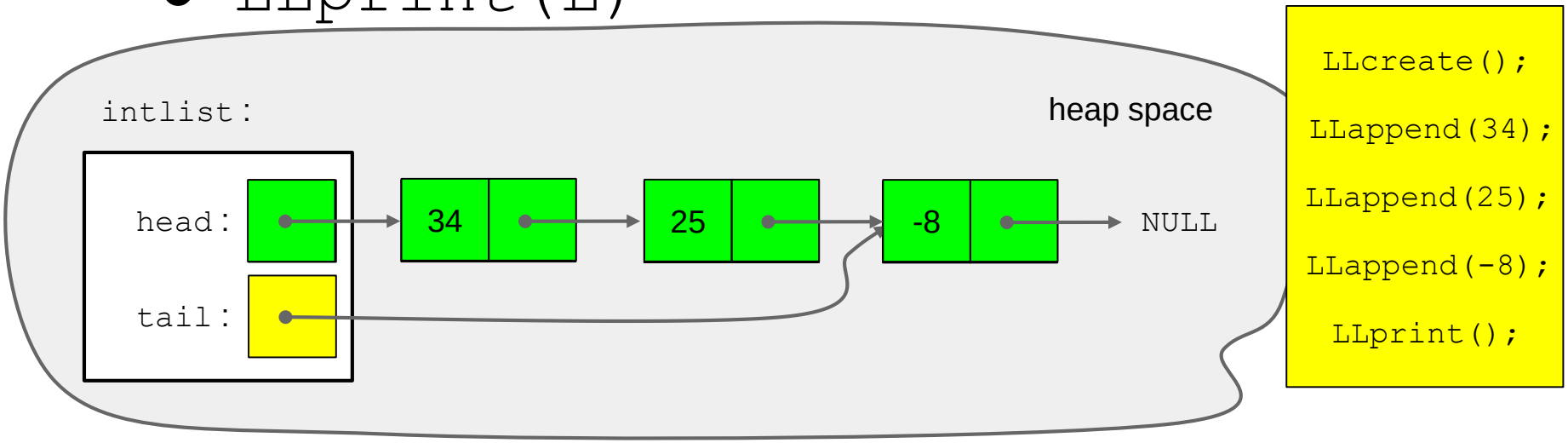
A se

Mai

```
// Outputs the list elements in order from head to tail
void LLprint(LL_t * intlist) {
    node_t * current = intlist->head;
    while (current != NULL) {
        printf(" %d", current->data);
        current = current->next;
    }
    putchar('\n');
}
```

- LLappend(L, x)
- LLprint(L)

```
34  25  -8
```



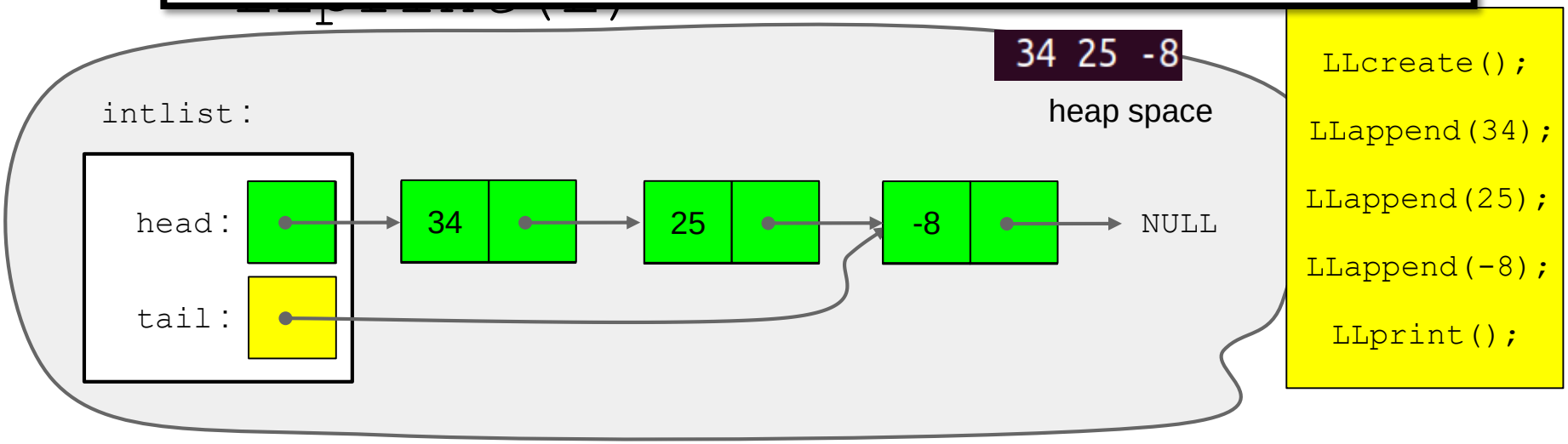

```
LLcreate();
LLappend(34);
LLappend(25);
LLappend(-8);
LLprint();
```

Linked Lists

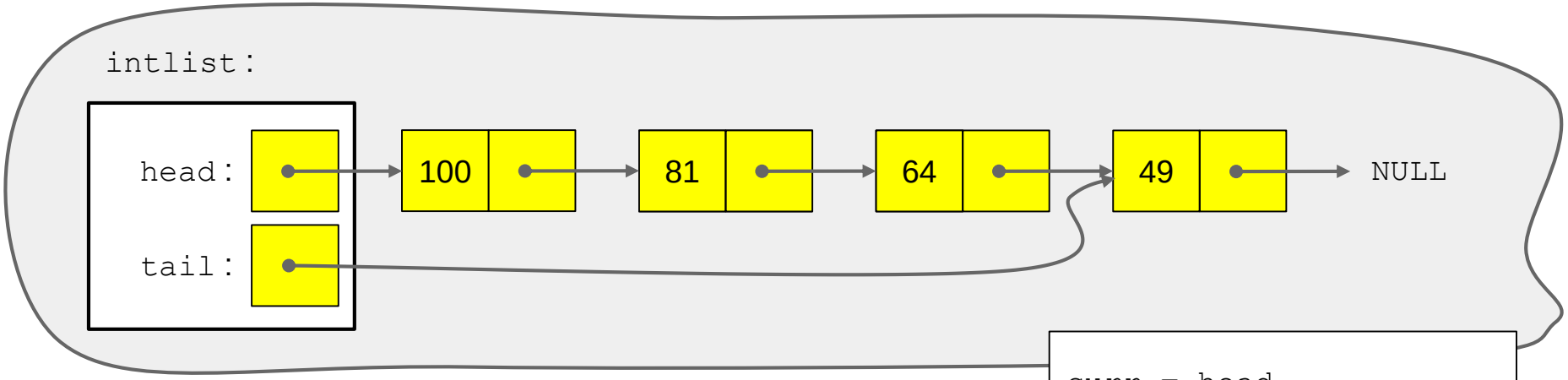
A s

Ma

```
// Outputs the list elements in order from head to tail
void LLprint(LL_t * intlist) {
    // for ( init ; entry ; increment ) {
    for ( node_t * current = intlist->head;
          current != NULL;
          current = current->next ) {
        printf(" %d", current->data);
    }
    putchar('\n');
}
```



Linked List: `search(target)`




```
curr = head
while(curr != NULL) {
   print curr->data
   curr = curr->next
}
return 0
```

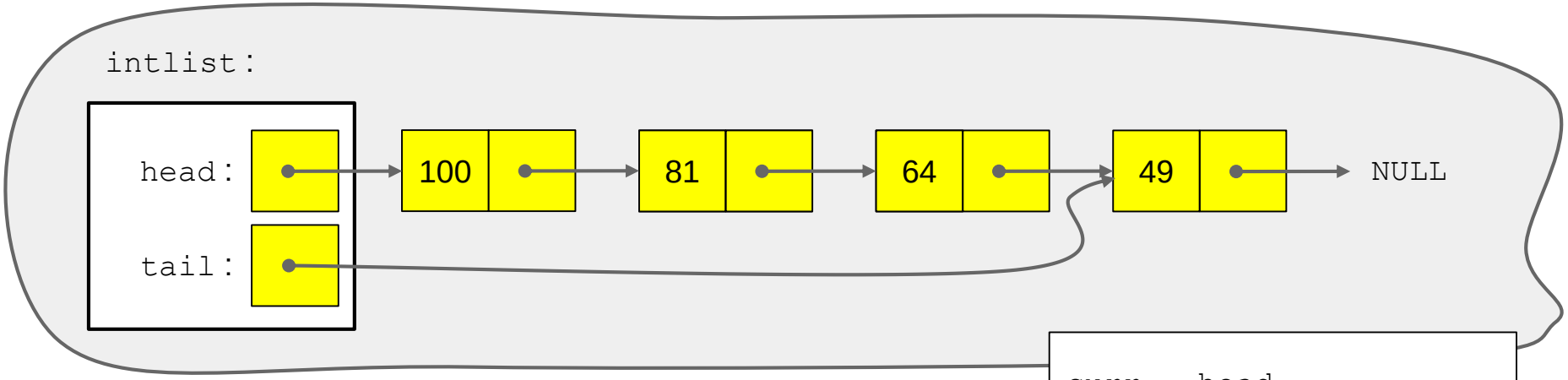
`search(64)` returns `1`

`search(58)` returns `0`

Q. What's the strategy this time?

- similar to `print()`

Linked List: `search(target)`




```
curr = head
while(curr != NULL) {
  if equal then
    return 1
  curr = curr->next
}
return 0
```

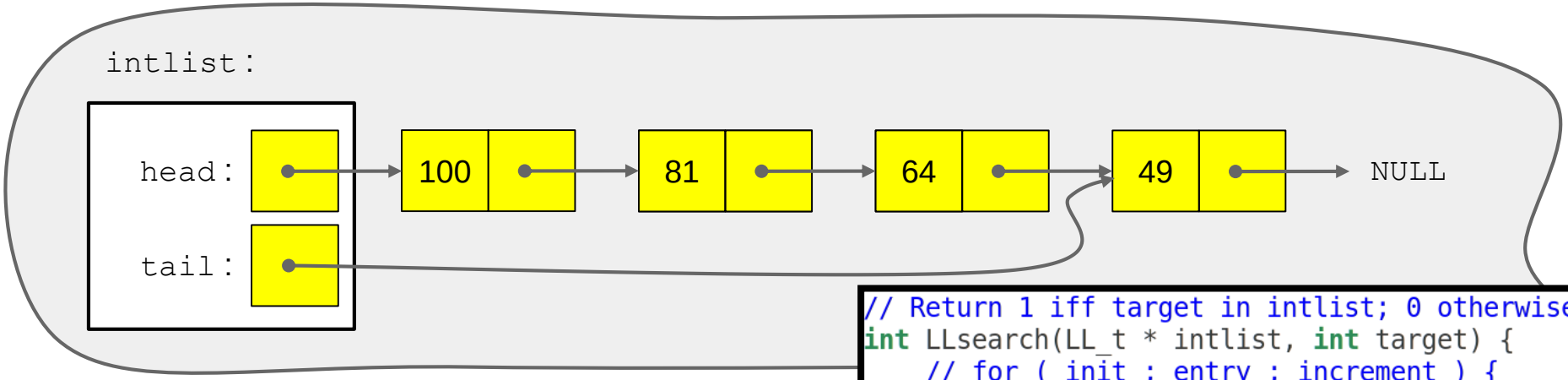
`search(64)` returns `1`

`search(58)` returns `0`

Q. What's the strategy this time?

- similar to `print()`
- instead of print, `return 1` if found

Linked List: `search(target)`




```c
// Return 1 iff target in intlist; 0 otherwise
int LLsearch(LL_t * intlist, int target) {
    // for ( init ; entry ; increment ) {


    node_t * current = intlist->head;
    while (current != NULL) {
        if (current->data == target) {
            return 1;
        }
        current = current->next;
    }
    return 0;
}
```

`search(64)` returns `1`

`search(58)` returns `0`

Q. What's the strategy this time?

- similar to `print()`
- instead of print, `return 1` if found

Q. What's the running time?