IIT Bombay

Computer Programming

Dr. Deepak B Phatak
Dr. Supratik Chakraborty
Department of Computer Science and Engineering
IIT Bombay

Session: Structures and Pointers – Part 2

Quick Recap of Relevant Topics

- Structures as collections of variables/arrays/other structures
- Statically declared structures
- Pointers to structures
- Accessing members of structures through pointers

Overview of This Lecture

- Pointers as members of structures
- Linked structures
- Dynamic allocation and de-allocation of structures

Acknowledgment

- Some examples in this lecture are from

  An Introduction to Programming Through C++

  by Abhiram G. Ranade

  McGraw Hill Education 2014

- All such examples indicated in slides with the citation **AGRBook**

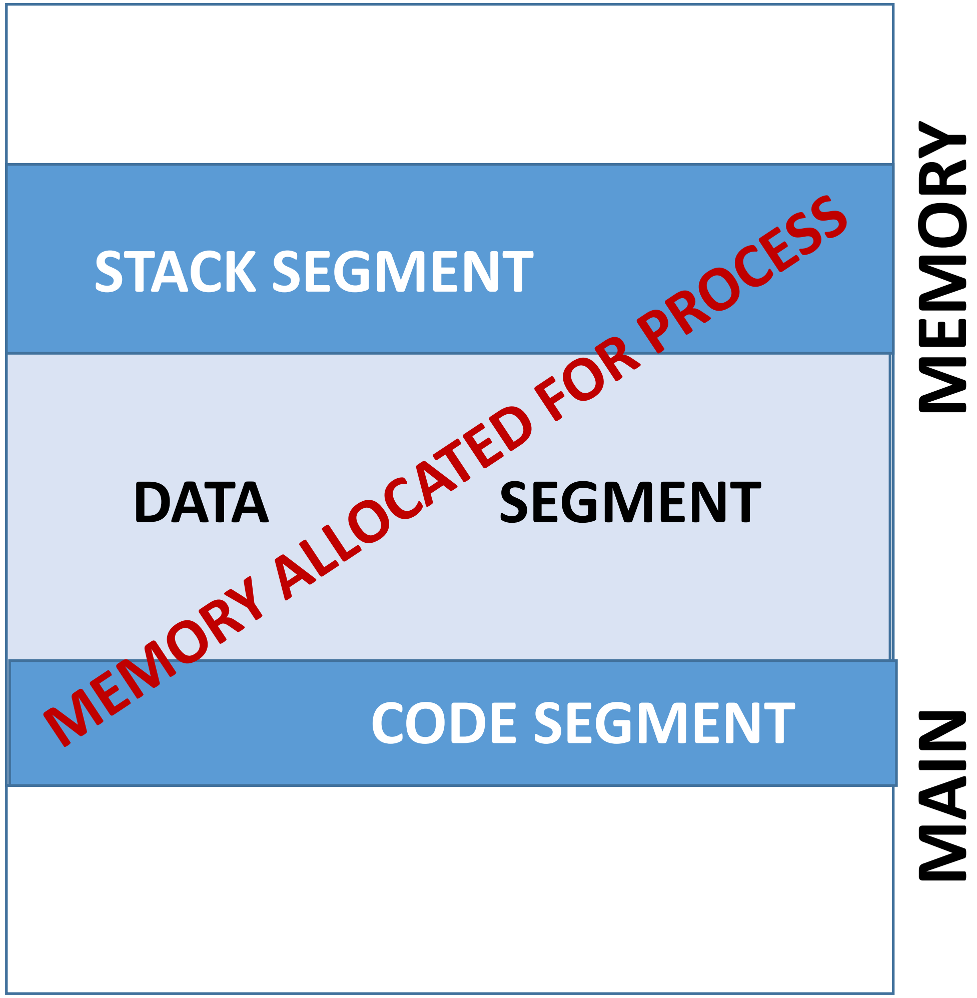
Memory for Executing a Program (Process)

- Operating system allocates a part of main memory for use by a process
- Divided into:

Code segment: Stores executable instructions in program

Data segment: For dynamically allocated data

Stack segment: Call stack



A Taxi Queuing System [Inspired by AGRBook]

```
int main()
{  struct Driver {char name[50]; int id;};
   struct Taxi {int id; Driver *drv;};
   Driver d1; Taxi t1;

… Rest of code …


  return 0;
}
```

A Taxi Queuing System [Inspired by AGRBook]

```
int main()
{  struct Driver {char name[50]; int id;};
   struct Taxi {int id; Driver *drv;};
   Driver d1; Taxi t1;

… Rest of code …


  return 0;
}
```

Member type:
Pointer-to-Driver

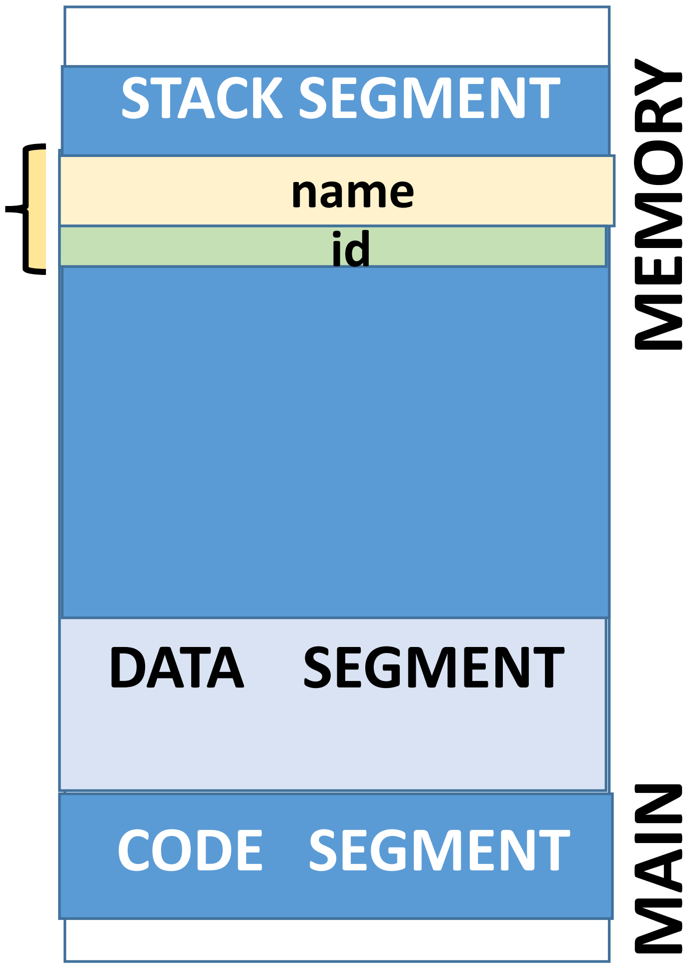
Assume requires
32 bits of storage

Structures in Main Memory

```
int main()
{  struct Driver {char name[50]; int id;};
   struct Taxi {int id; Driver *drv;};
   Driver d1; Taxi t1;

... Rest of code ...


  return 0;
}
```



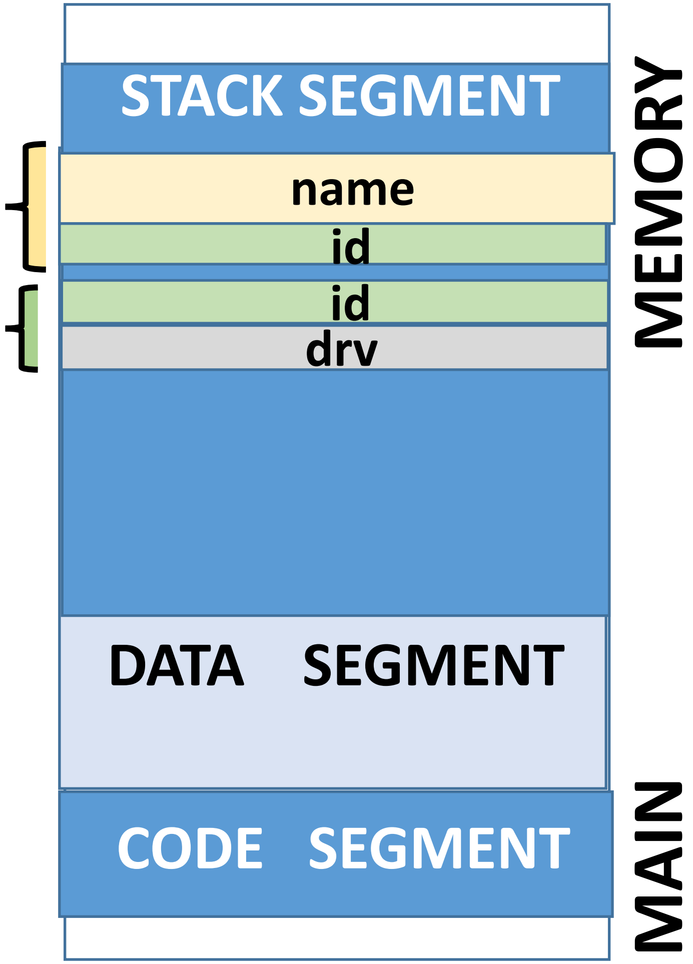
Structures in Main Memory

```
int main()
{  struct Driver {char name[50]; int id;};
   struct Taxi {int id; Driver *drv;};
   Driver d1; Taxi t1;

... Rest of code ...



  return 0;
}
```

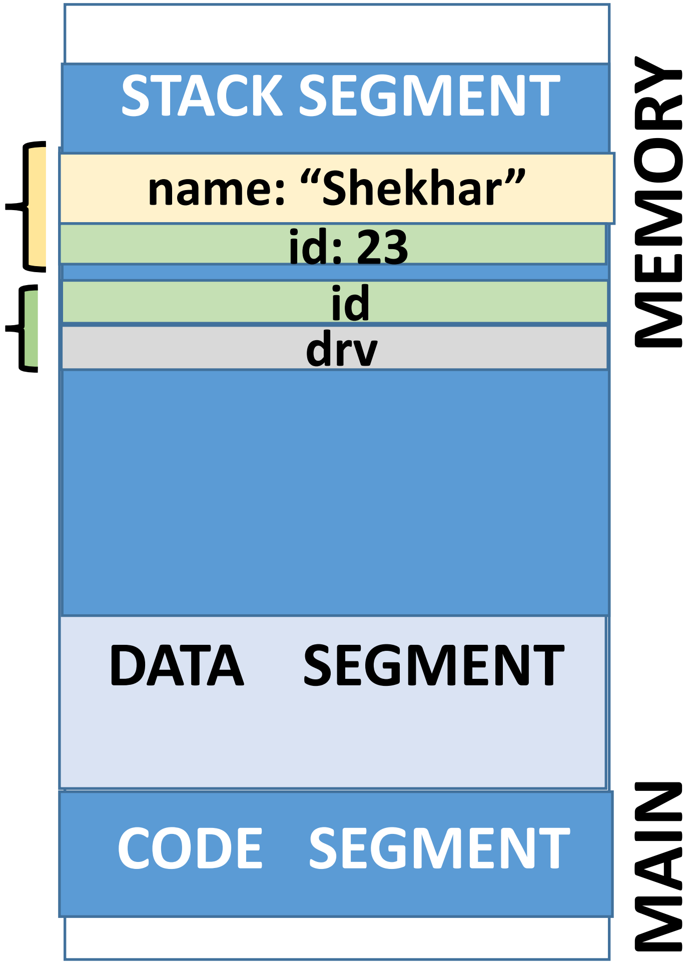


Structures in Main Memory

IIT Bombay

```
int main()
{  struct Driver {char name[50]; int id;};
   struct Taxi {int id; Driver *drv;};
   Driver d1; Taxi t1;
   d1 = {“Shekhar”, 23};
... Rest of code ...


  return 0;
}
```

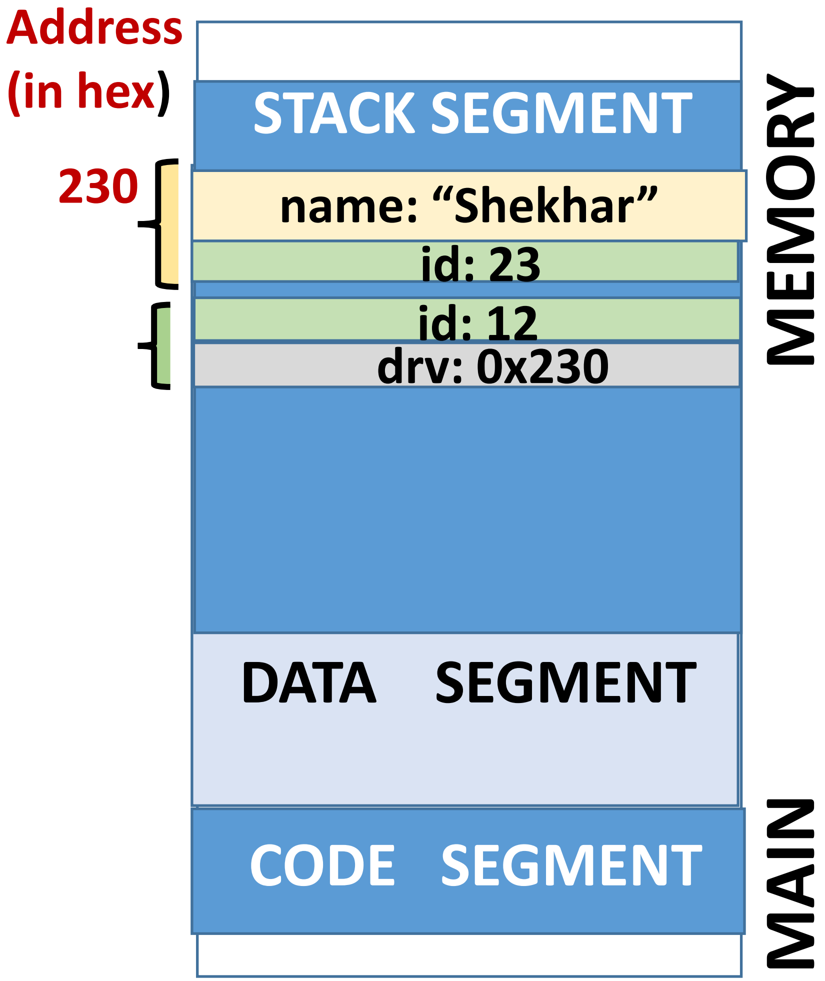


Structures in Main Memory

IIT Bombay

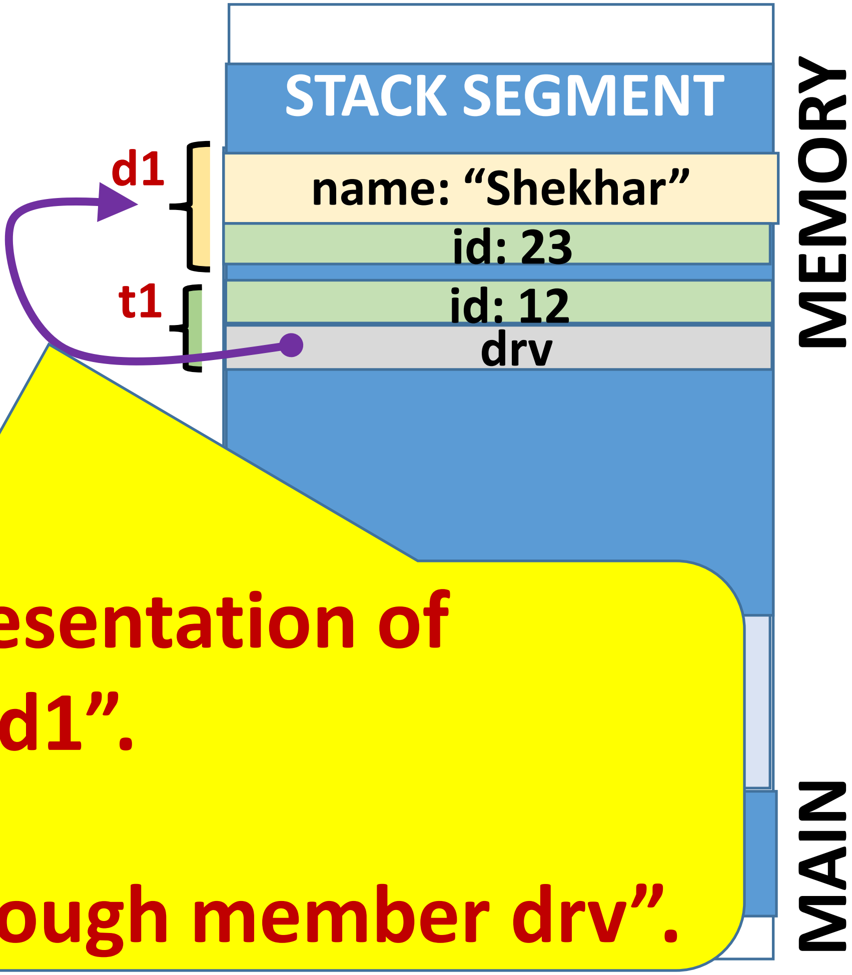
```
int main()
{  struct Driver {char name[50]; int id;};
   struct Taxi {int id; Driver *drv;};
   Driver d1; Taxi t1;
   d1 = {“Shekhar”, 23};
   t1.id = 12;   t1.drv = &d1;
… Rest of code …


  return 0;
}
```



Structures in Main Memory

```
int main()
{  struct Driver {char name[50]; int id;};
   struct Taxi {int id; Driver *drv;};
   Driver d1; Taxi t1;
   d1 = {"Shekhar", 23};
   t1.id = 12;   t1.drv = &d1;
```



Convenient pictorial representation of "t1.drv points to d1".

Informally, "t1 is linked to d1 through member drv".

Can We Link Taxi Structures?

We want to have a taxi in the queue have information about the next taxi in the queue.

Can we use

```
struct LinkedTaxi {
   int id; Driver *drv;
   LinkedTaxi next;
};
```

Object of type LinkedTaxi would require infinite storage

Can We Link Taxi Structures?

IIT Bombay

What about the following?

```
struct LinkedTaxi {
    int id; Driver *drv;
    LinkedTaxi *next;
};
```

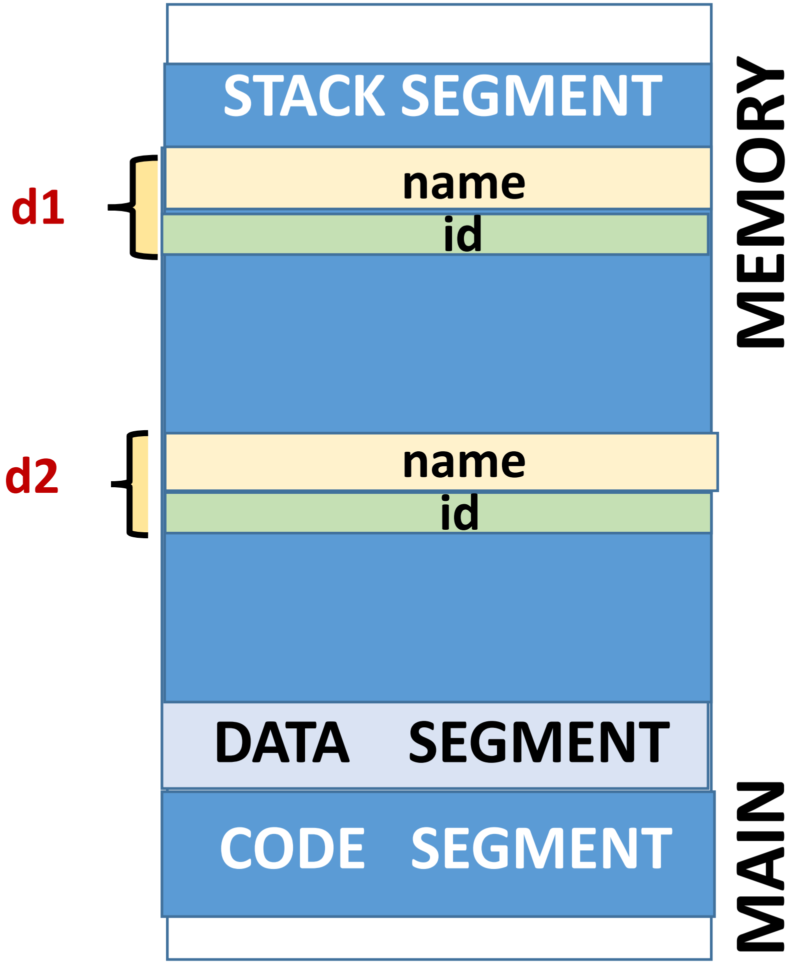
member of type Pointer-to-LinkedTaxi

Does a LinkedTaxi structure require infinite storage?

NO!!! Each member of pointer type requires 4 bytes

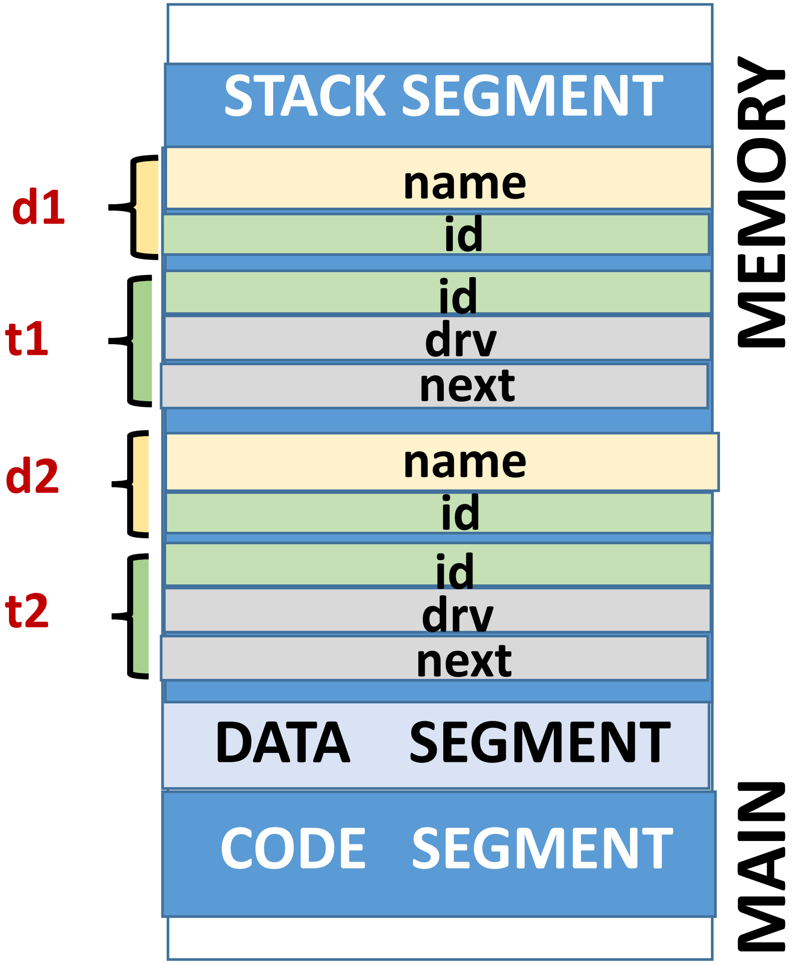
Linked Structures in Main Memory

```
int main()
{  struct Driver {char name[50]; int id;};
   struct LinkedTaxi {
        int id; Driver *drv;
        LinkedTaxi *next;};
   Driver d1, d2; Taxi t1, t2;
   d1 = {“Shekhar”, 23};
   d2 = {“Abdul”, 34};
   t1.id = 12;  t1.drv = &d1; t1.next = NULL;
   t2.id = 11;  t2.drv = &d2; t2.next = &t1;
   cout << (t2.next)->drv->name;  return 0;
}
```



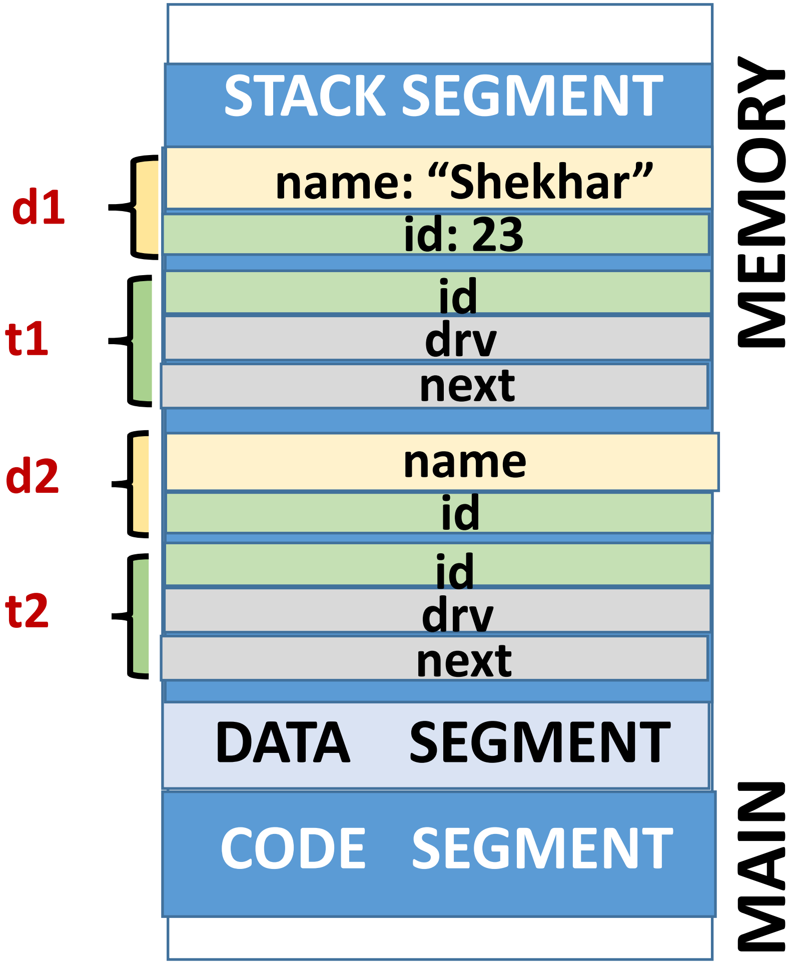
Linked Structures in Main Memory

```
int main()
{  struct Driver {char name[50]; int id;};
   struct LinkedTaxi {
        int id; Driver *drv;
        LinkedTaxi *next;};
   Driver d1, d2; Taxi t1, t2;
   d1 = {“Shekhar”, 23};
   d2 = {“Abdul”, 34};
   t1.id = 12;  t1.drv = &d1; t1.next = NULL;
   t2.id = 11;  t2.drv = &d2; t2.next = &t1;
   cout << (t2.next)->drv->name;  return 0;
}
```



Linked Structures in Main Memory

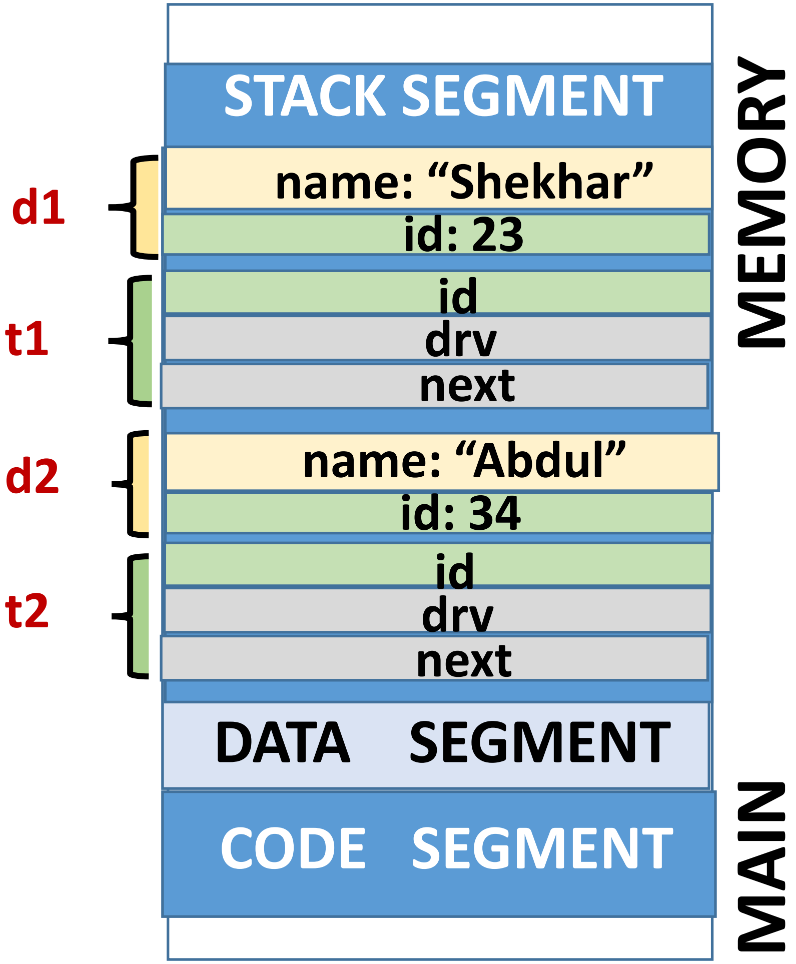
```
int main()
{  struct Driver {char name[50]; int id;};
   struct LinkedTaxi {
        int id; Driver *drv;
        LinkedTaxi *next;};
   Driver d1, d2; Taxi t1, t2;
   d1 = {“Shekhar”, 23};
   d2 = {“Abdul”, 34};
   t1.id = 12;  t1.drv = &d1; t1.next = NULL;
   t2.id = 11;  t2.drv = &d2; t2.next = &t1;
   cout << (t2.next)->drv->name;  return 0;
}
```



Linked Structures in Main Memory

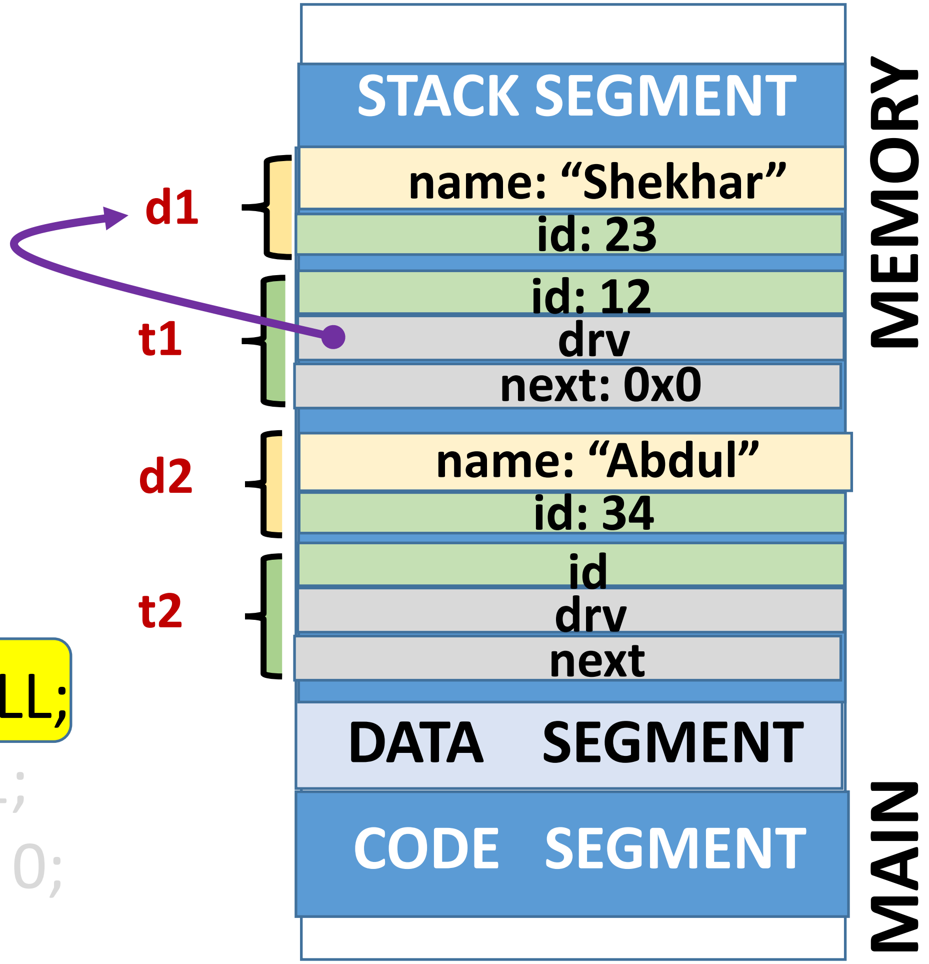
IIT Bombay

```
int main()
{  struct Driver {char name[50]; int id;};
   struct LinkedTaxi {
        int id; Driver *drv;
        LinkedTaxi *next;};
   Driver d1, d2; Taxi t1, t2;
   d1 = {“Shekhar”, 23};
   d2 = {“Abdul”, 34};
   t1.id = 12;  t1.drv = &d1; t1.next = NULL;
   t2.id = 11;  t2.drv = &d2; t2.next = &t1;
   cout << (t2.next)->drv->name;  return 0;
}
```



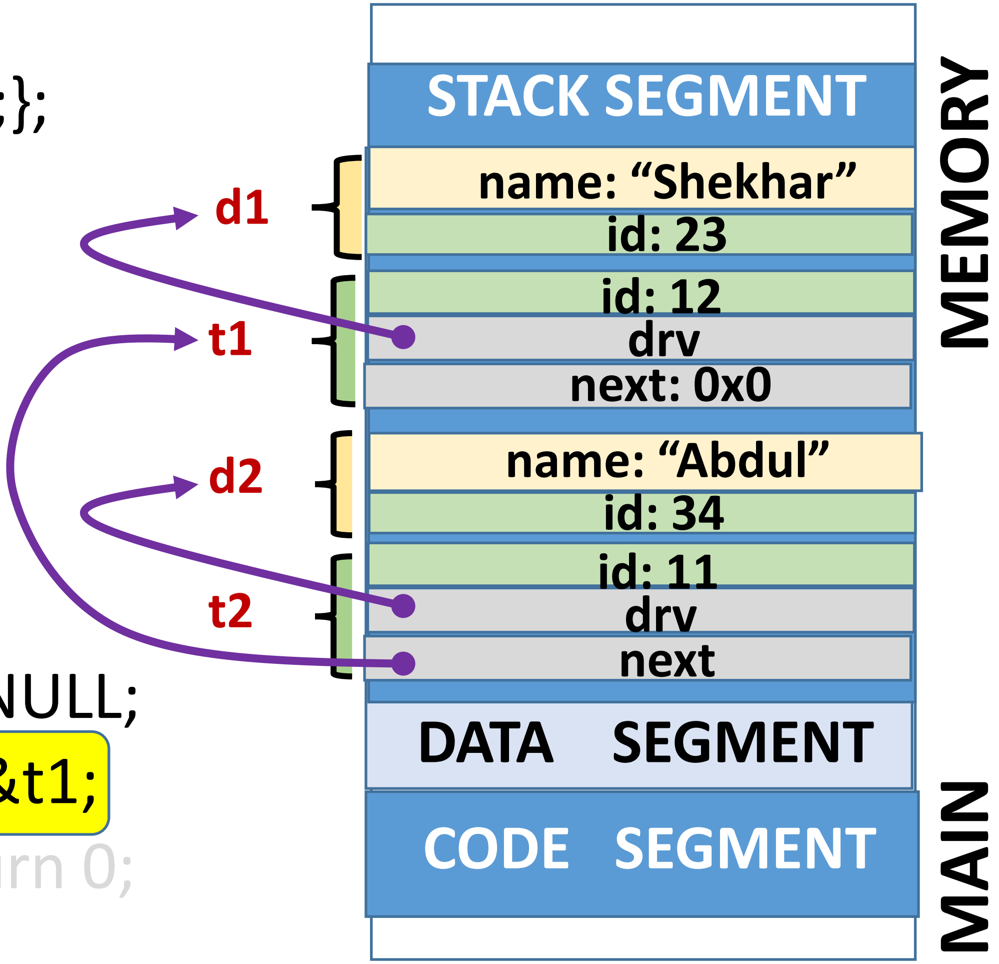
Linked Structures in Main Memory

```
int main()
{  struct Driver {char name[50]; int id;};
   struct LinkedTaxi {
        int id; Driver *drv;
        LinkedTaxi *next;};
   Driver d1, d2; Taxi t1, t2;
   d1 = {“Shekhar”, 23};
   d2 = {“Abdul”, 34};
   t1.id = 12;  t1.drv = &d1; t1.next = NULL;
   t2.id = 11;  t2.drv = &d2; t2.next = &t1;
   cout << (t2.next)->drv->name;  return 0;
}
```



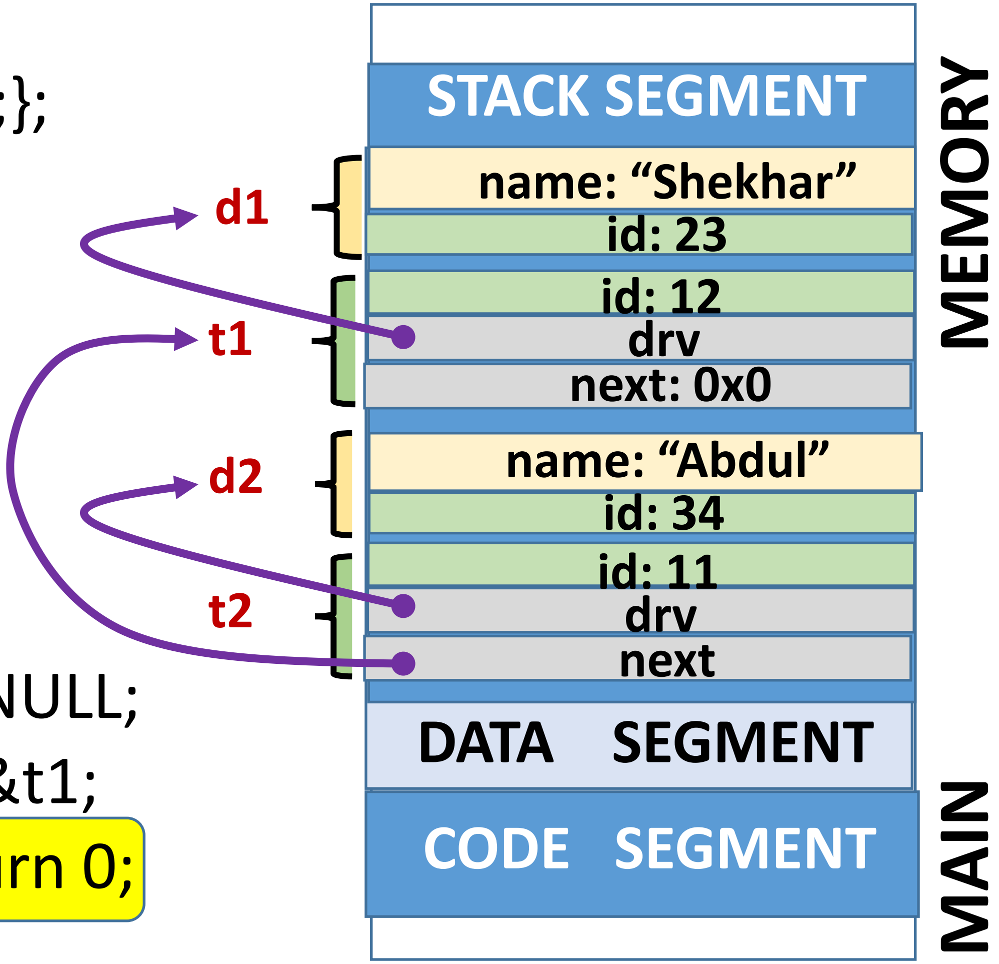
Linked Structures in Main Memory

```
int main()
{  struct Driver {char name[50]; int id;};
   struct LinkedTaxi {
        int id; Driver *drv;
        LinkedTaxi *next;};
   Driver d1, d2; Taxi t1, t2;
   d1 = {“Shekhar”, 23};
   d2 = {“Abdul”, 34};
   t1.id = 12;  t1.drv = &d1; t1.next = NULL;
   t2.id = 11;  t2.drv = &d2; t2.next = &t1;
   cout << (t2.next)->drv->name;  return 0;
}
```



Linked Structures in Main Memory

```
int main()
{  struct Driver {char name[50]; int id;};
   struct LinkedTaxi {
        int id; Driver *drv;
        LinkedTaxi *next;};
   Driver d1, d2; Taxi t1, t2;
   d1 = {"Shekhar", 23};
   d2 = {"Abdul", 34};
   t1.id = 12;  t1.drv = &d1; t1.next = NULL;
   t2.id = 11;  t2.drv = &d2; t2.next = &t1;
   cout << (t2.next)->drv->name;  return 0;
}
```

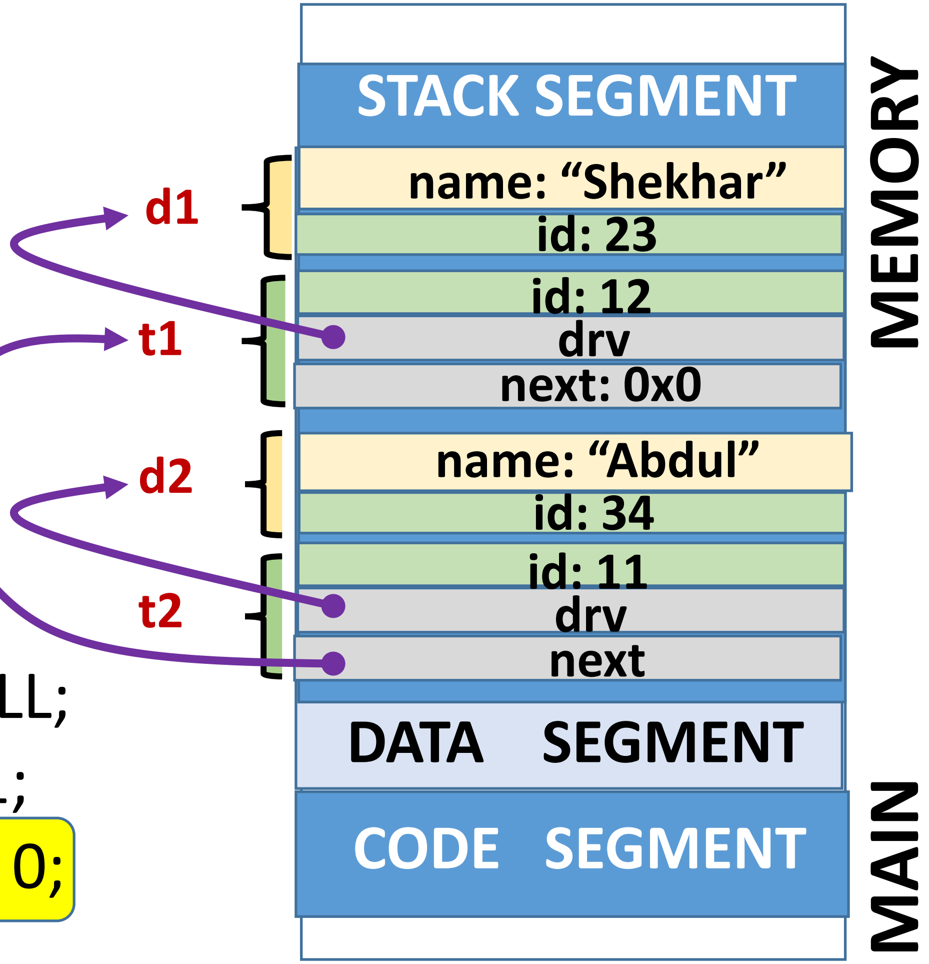


Linked Structures in Main Memory

```
int main()
{  struct Driver {char name[50]; int id;};
   struct LinkedTaxi {
       int id; Driver *drv;
```

Program output:
Shekhar

```
   t1.id = 12;  t1.drv = &d1; t1.next = NULL;
   t2.id = 11;  t2.drv = &d2; t2.next = &t1;
   cout << (t2.next)->drv->name;  return 0;
}
```



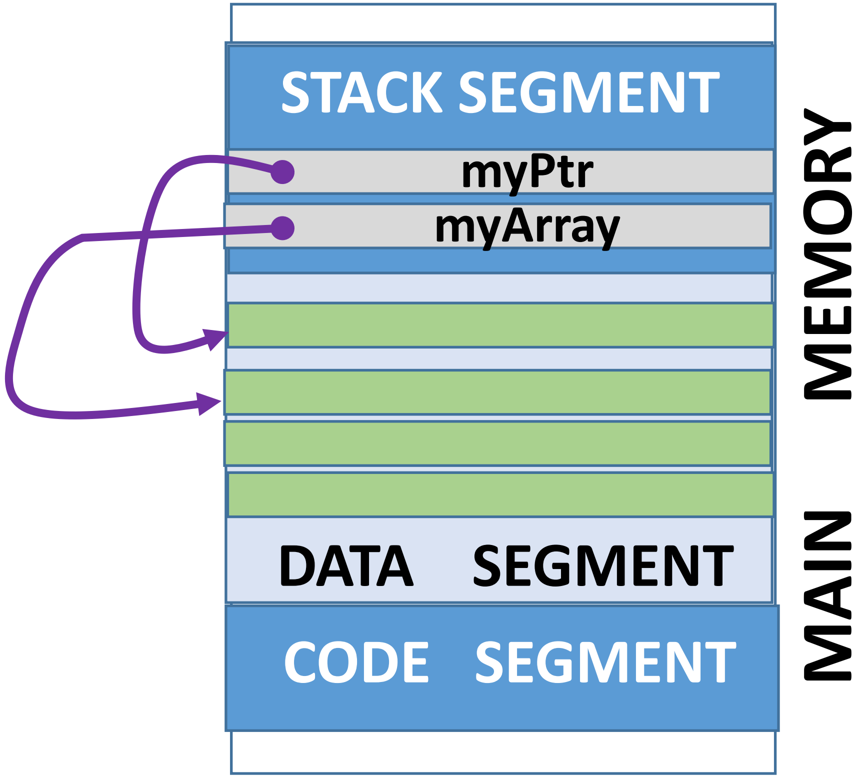
Recall: Dynamic Memory Allocation/De-allocation

IIT Bombay

- Recall “new”/“delete” for dynamically allocating/de-allocating memory for variables/arrays of basic data types

```
int * myPtr = new int;
int * myArray = new int[3];

     … Some code …
```

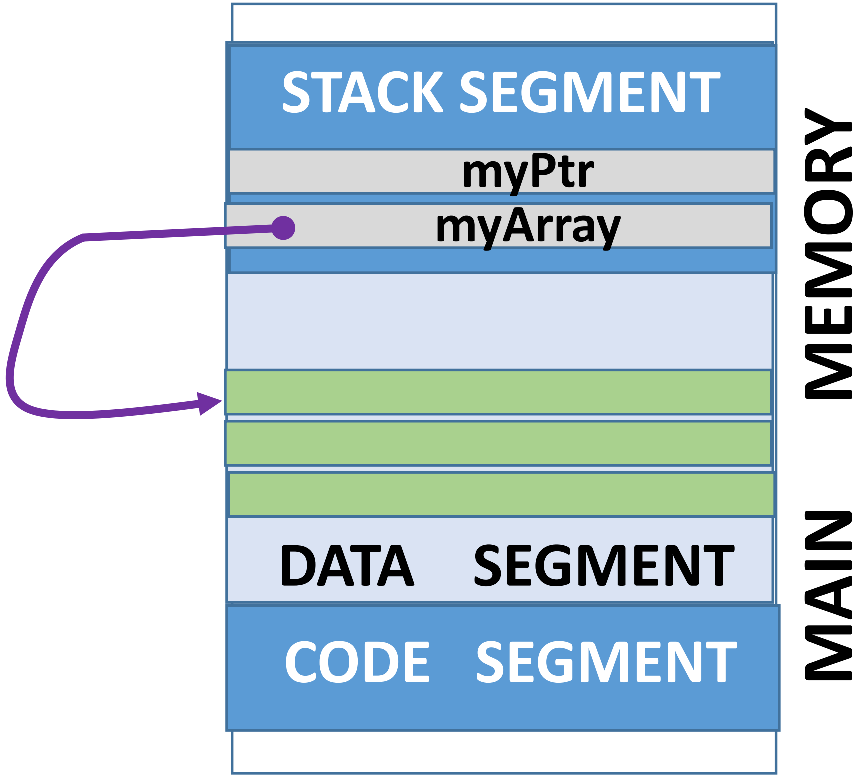


Recall: Dynamic Memory Allocation/De-allocation

- Recall "new"/"delete" for dynamically allocating/de-allocating memory for variables/arrays of basic data types

```
int * myPtr = new int;
int * myArray = new int[3];

     … Some code …
if (myPtr != NULL) delete myPtr;
```



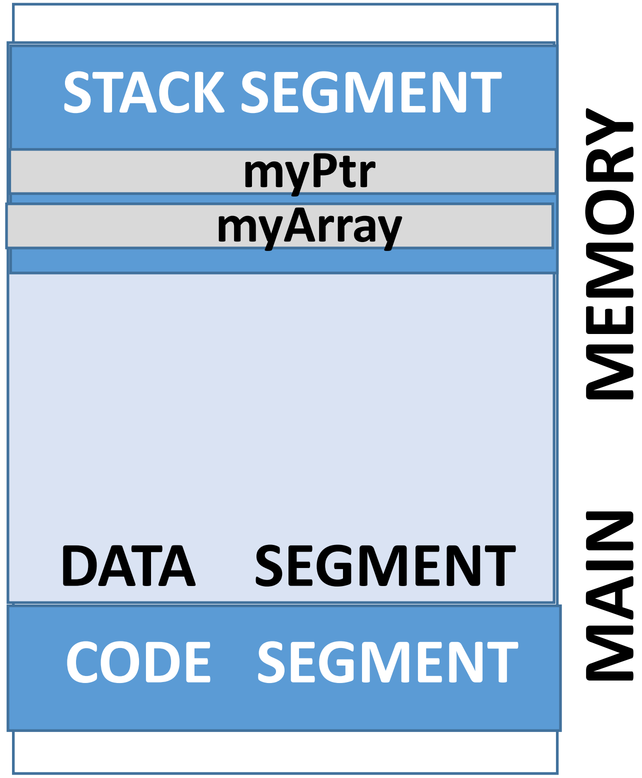
Recall: Dynamic Memory Allocation/De-allocation

IIT Bombay

- Recall “new”/“delete” for dynamically allocating/de-allocating memory for variables/arrays of basic data types

```
int * myPtr = new int;
int * myArray = new int[3];

     … Some code …
if (myPtr != NULL) delete myPtr;
if (myArray != NULL)
              delete [] myArray;
```

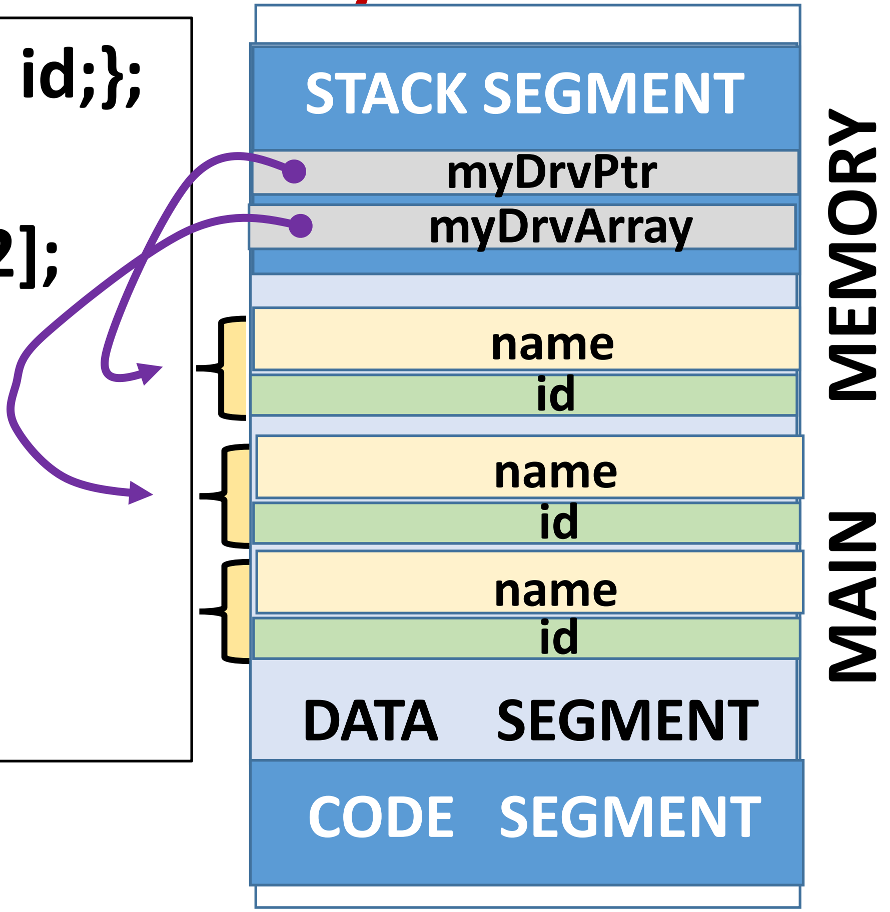


Dynamically Allocating Structures

- **“new”/“delete” work in exactly the same way for structures**

```
struct Driver {char name[50]; name id;};
Driver * myDrvPtr = new Driver;
Driver * myDrvArray = new Driver[2];

     … Some code …
```

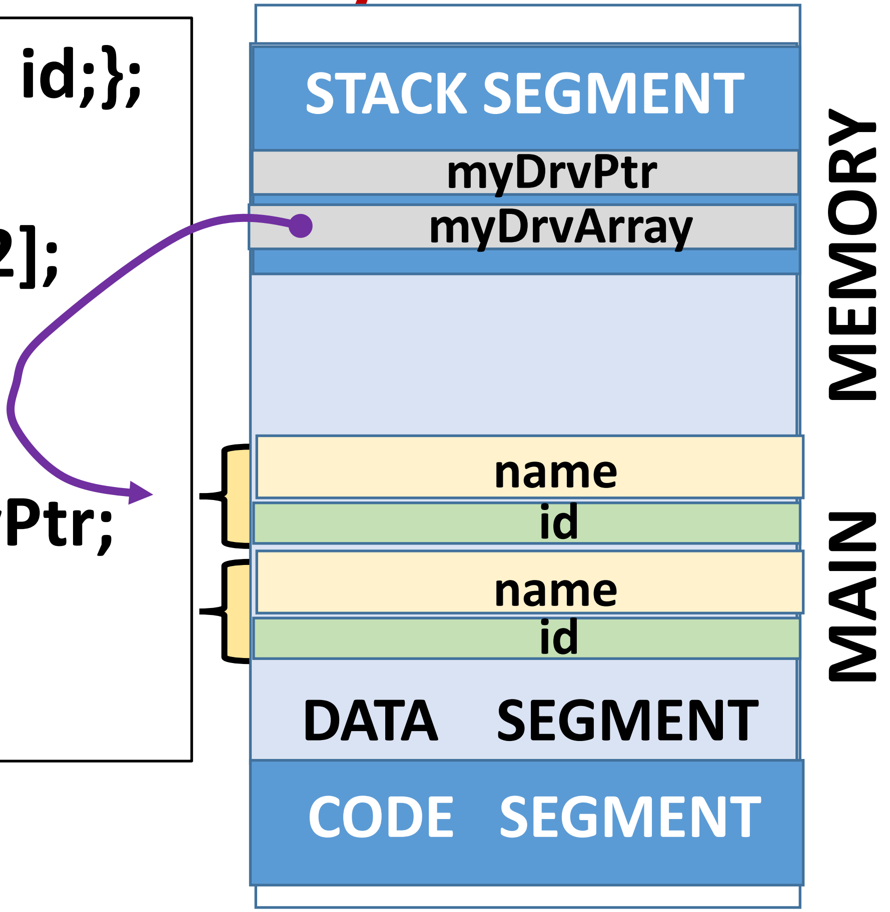


Dynamically Allocating Structures

- “new”/“delete” work in exactly the same way for structures

```
struct Driver {char name[50]; name id;};
Driver * myDrvPtr = new Driver;
Driver * myDrvArray = new Driver[2];

      … Some code …
if (myDrvPtr != NULL) delete myDrvPtr;
```

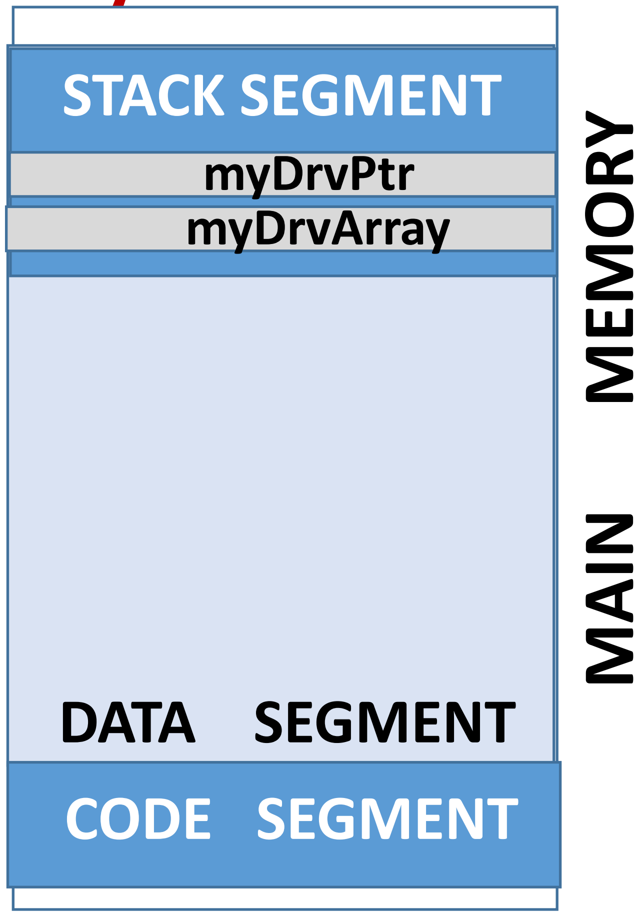


Dynamically Allocating Structures

- “new”/“delete” work in exactly the same way for structures

```
struct Driver {char name[50]; name id;};
Driver * myDrvPtr = new Driver;
Driver * myDrvArray = new Driver[2];

    ... Some code ...
if (myDrvPtr != NULL) delete myDrvPtr;
if (myDrvArray != NULL)
          delete [] myDrvArray;
```



Caveats when using “new”

- Same caveats as studied earlier
  - Do not assume “new” always succeeds in allocating memory
  - “new” may fail and return NULL
  - Always check if pointer returned by “new” is non-NULL before dereferencing it.

```
Driver *myDrvPtr = new Driver;
if (myDrvPtr != NULL) {
    myDrvPtr->id = 23;
}
```

Caveats when using “delete”

- Same caveats as studied earlier
  - Always check if pointer is non-NULL before calling “delete”

```
Driver *myDrvArray = new Driver[2];
  … Some code …
if (myDrvArray != NULL) {
   delete [] myDrvArray;
}
```

Summary

- Members of pointer data types in structures
- Linked structures
- Dynamic allocation/de-allocation of structures in data segment (heap)