REFRESHER

Programming Concepts

RECAP OF

- Data Types
- Functions
- Pointers
- Array
- Dynamic Memory Allocation
- Structures
- Linked List

PRIMITIVE DATA TYPES

- **int** - whole number (4 bytes) 1, 2, 3
- **char** - single character (1 byte) A, B, C
- **float** - decimal number (4 bytes) 1.012
- **double** - larger decimal number (8 bytes) 1.01234
- **void** - holds no value
- **int*, float*, char*** - pointer type (8 bytes)

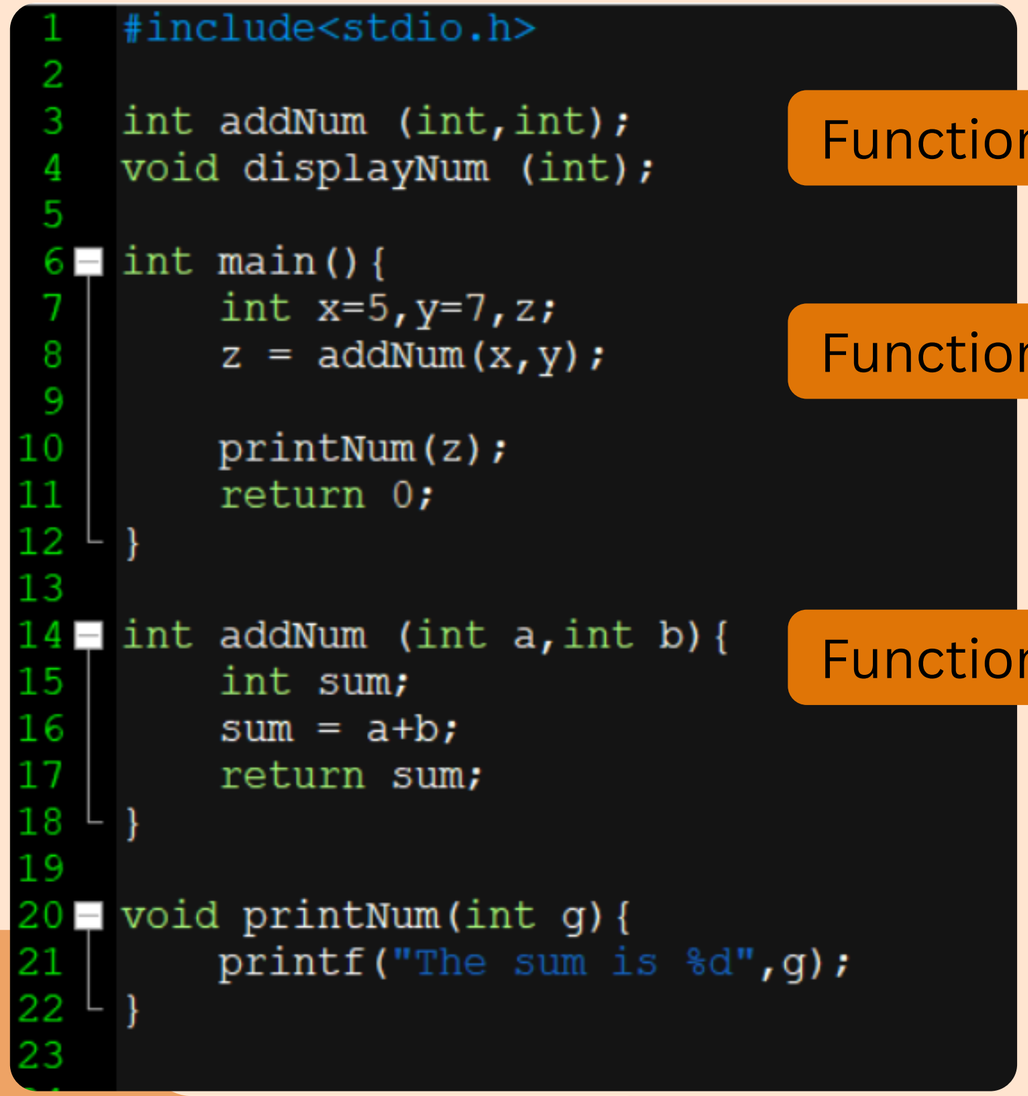
DERIVED DATA TYPES

- **Arrays** - sequence of data items with same type of values
- **Pointers** - accesses the memory and deals with the addresses of which it is pointed to

USER DEFINED DATA TYPES

- **structure** - a package of different types of variables under one name, defined by keyword “*struct*”
- **union** - allows storing of varying data types in the same memory location that can be defined with different members but only a single member can contain a value at any given time
- **enum** - consists of integral constants

FUNCTIONS



Function Prototype

Function Call

Function Definition

POINTERS

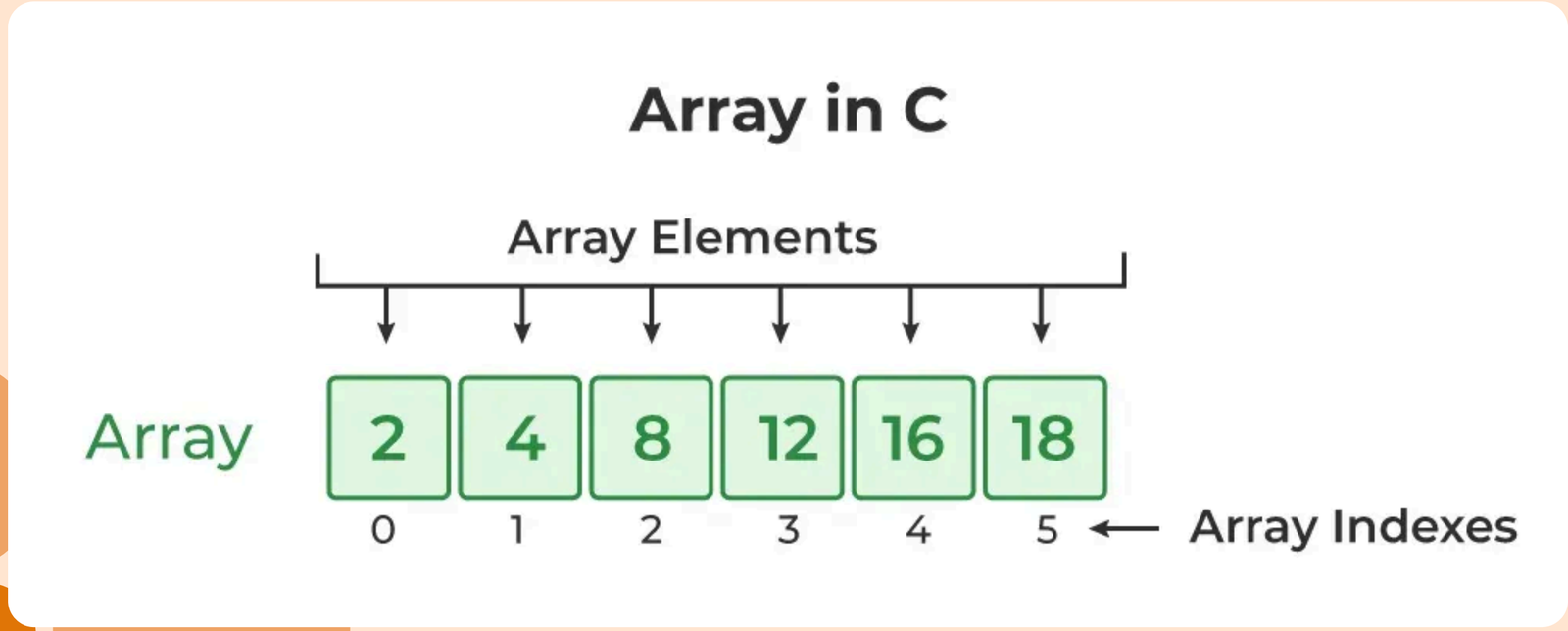
- Value → address of another variable
- datatype → any

- Declaration
  - `datatype *variableName;`

	stores	**returns**	
`*ptr`	value	value	`*ptr = x;`
`ptr`	address	address	`ptr = &x;`

ARRAYS

- A fixed-size collection of elements of the same data type
- Stores multiple values in a single variable for easy data manipulation
  - `ex. int array[6];`



ARRAYS

- Declaration
  - `datatype variableName[size];`
- Initialization
  - `int array[6] = {2,4,8,12,16,18};`
- Accessing the Array
  - refer to its index
  - `variableName[index];`
  - `int array[6] = {2,4,8,12,16,18};`
  - `array[4] /*accessing the 5th index*/`

ARRAYS

- Passing Arrays to Functions
- Function Definition

```
void sort(int arr[]){
    /*code*/
}
```

or

```
void sort(int *arr){
    /*code*/
}
```

- Function Prototype

```
void sort(int[]);
```

or

```
void sort(int*);
```

Function Call

```
sort(arr);
```

DYNAMIC MEMORY ALLOCATION

- allocating memory during runtime program

`malloc()`	allocates a single block of memory and returns pointer to the first byte of allocated space
`calloc()`	allocates a specified number of blocks of memory and initializes all bytes to zero
`realloc()`	modifies the size of a previously allocated memory space
`free()`	de-allocates the memory allocated

DYNAMIC MEMORY ALLOCATION

- Syntax

malloc()	ptr=(cast-type*)malloc(byte-size);
calloc()	ptr=(cast-type*)calloc(n,element-size);
realloc()	ptr=malloc(ptr,newSize);
free()	free(ptr);

STRUCTURE

- collection of one or more variables which can be of different data types stored contiguously in memory
- Syntax

```
struct tagName{
        datatype variableName;
}variableList;
```

Example

```
struct Fraction{
        int numerator;
        int denominator;
}A,B;
```

STRUCTURE

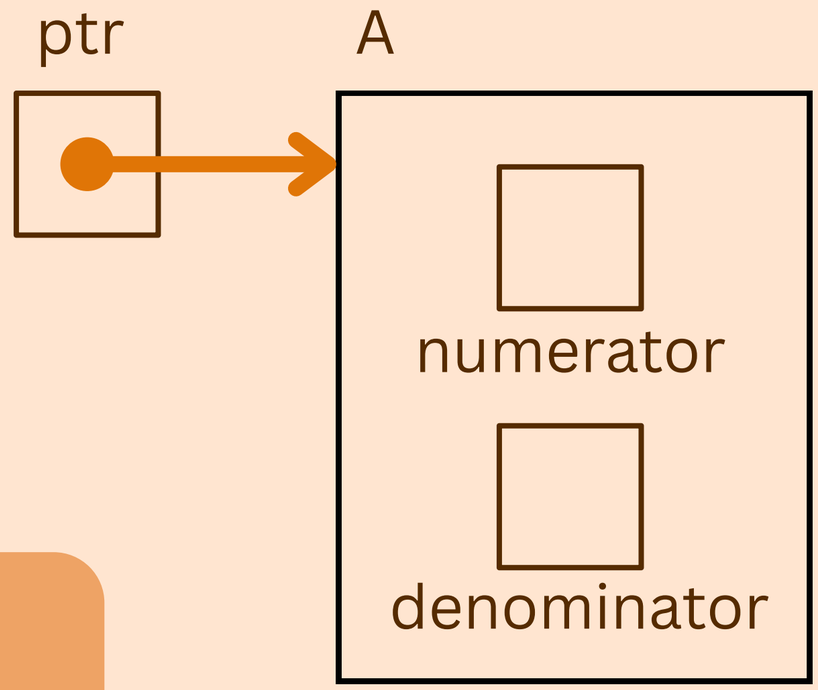
- using typedef

```
typedef struct tagName{
        datatype variableName;
}NewType;
```

Example

```
typedef struct Fraction{
        int numerator;
        int denominator;
}Frac;
```

POINTER TO STRUCTURE



Example

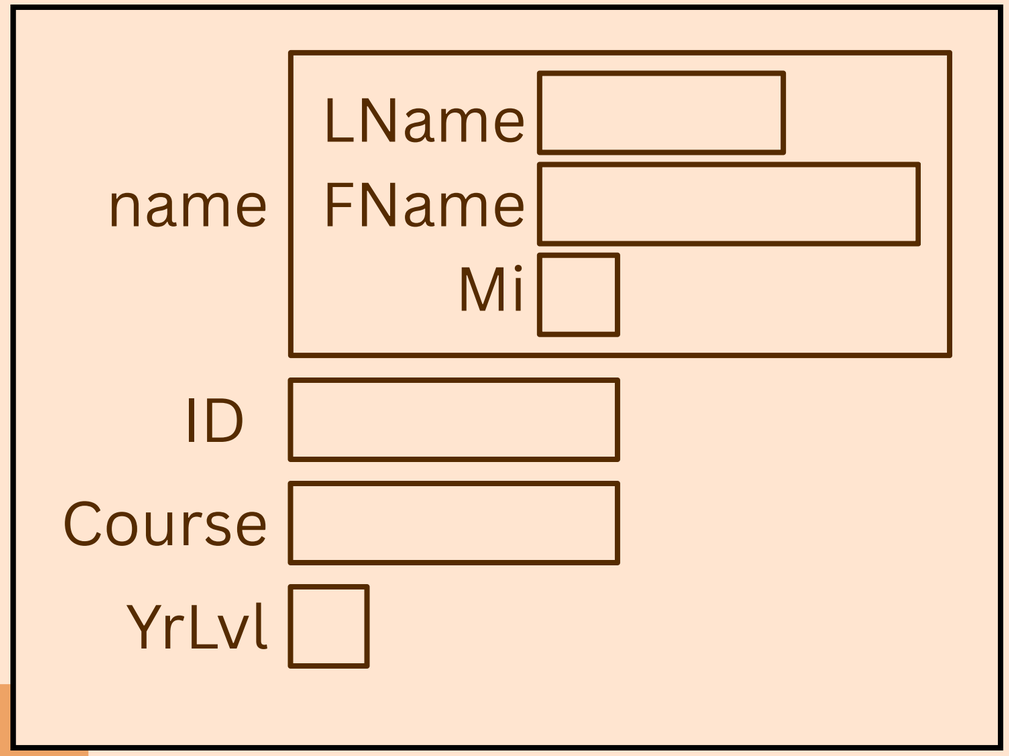
```
typedef struct Fraction{
        int numerator;
        int denominator;
}Frac;

Frac A=3;
Frac *ptr;
ptr = &A;
```

ACCESSING STRUCTURES

- Access the variable numerator in three ways

```
A.numerator;

(*ptr).numerator;

ptr->numerator;
```

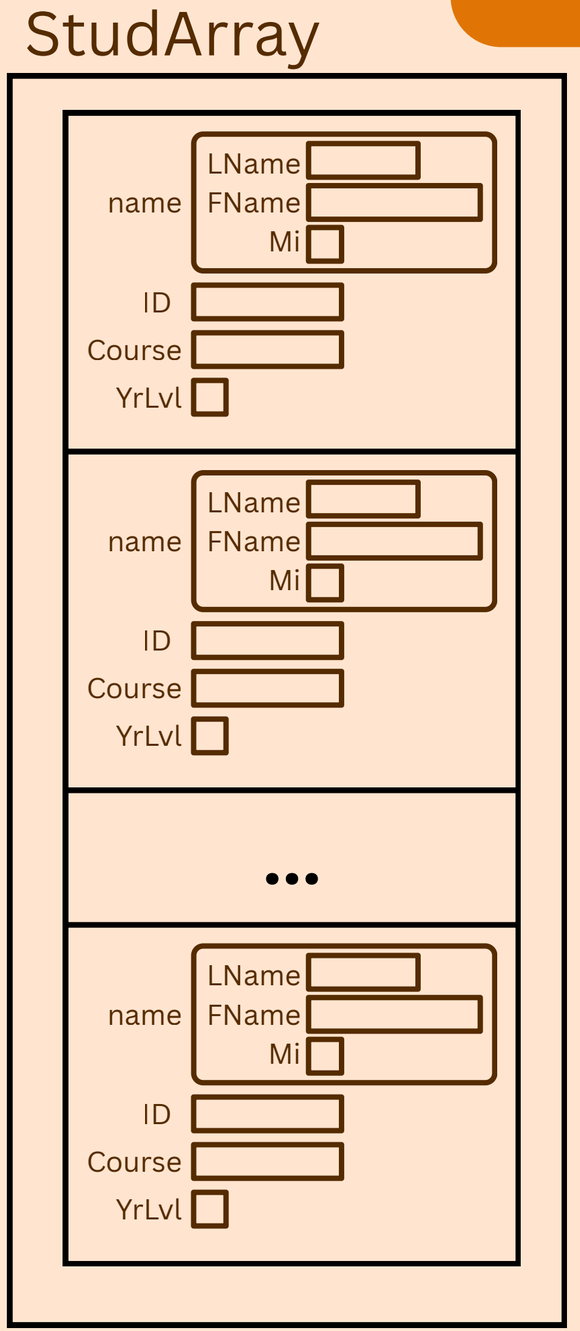
STRUCTURES




```
typedef struct{
    char LName[16];
    char FName[24];
    char Mi;
}Nametype;

typedef struct{
    Nametype name;
    unsigned int ID;
    char Course[8];
    int YrLvl;
}Studtype;
```

ARRAY OF STRUCTURES

```
typedef struct{
   char LName[16];
   char FName[24];
   char Mi;
}Nametype;

typedef struct{
   Nametype name;
   unsigned int ID;
   char Course[8];
   int YrLvl;
}Studtype;

Studtype StudArray[size];
```



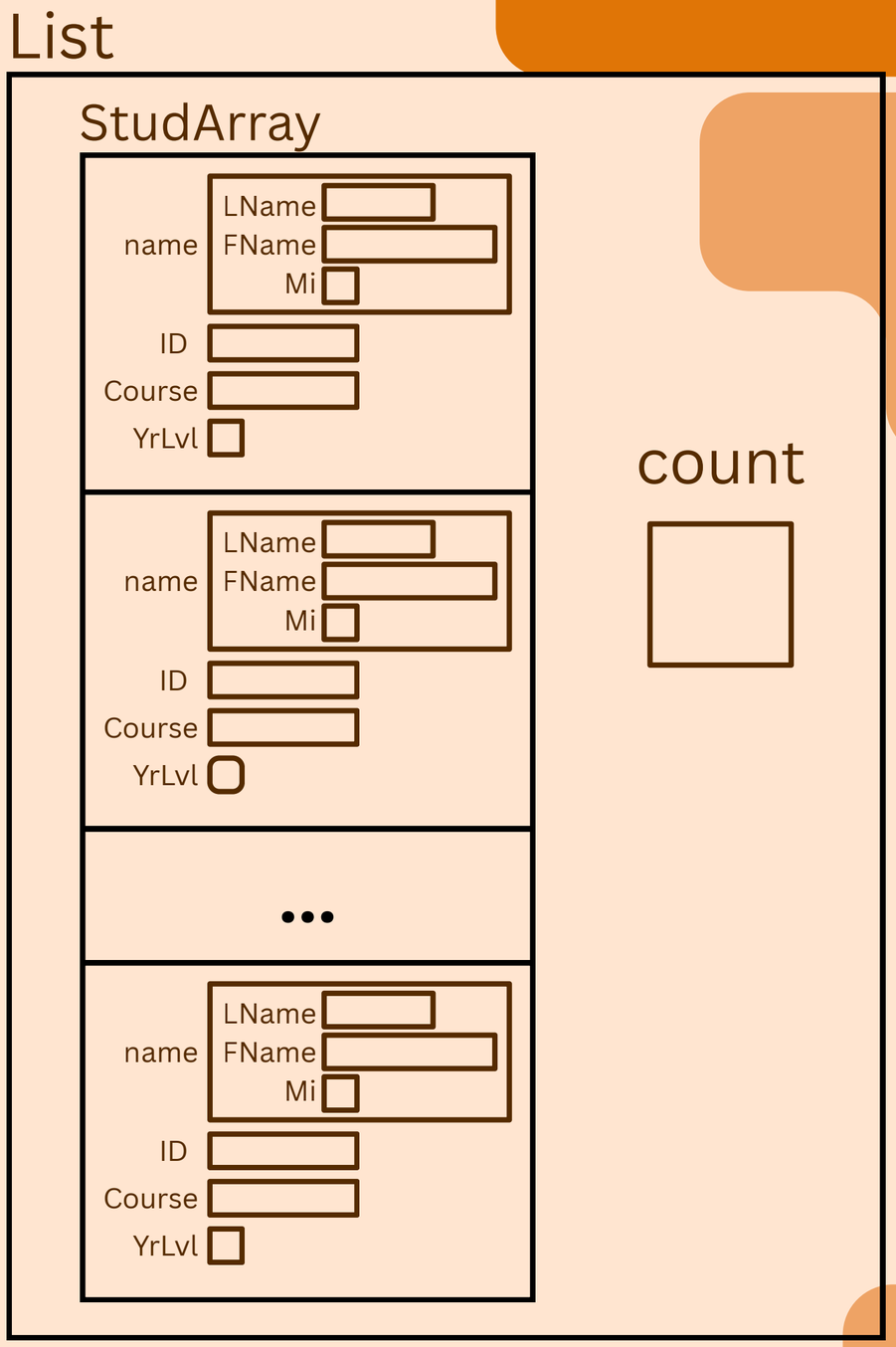
ARRAY OF STRUCTURES

```
typedef struct{
   char LName[16];
   char FName[24];
   char Mi;
}Nametype;
```

```
typedef struct{
   Nametype name;
   unsigned int ID;
   char Course[8];
   int YrLvl;
}Studtype;
```

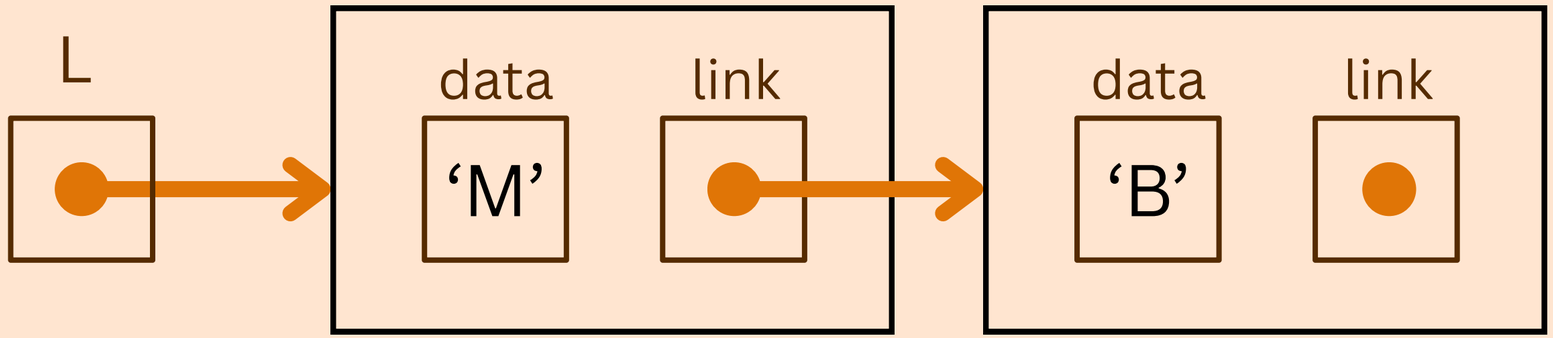
```
typedef struct{
   Studtype StudArray[size];
   int count;
}StudList;
```

```
StudList List;
```



LINKED-LIST

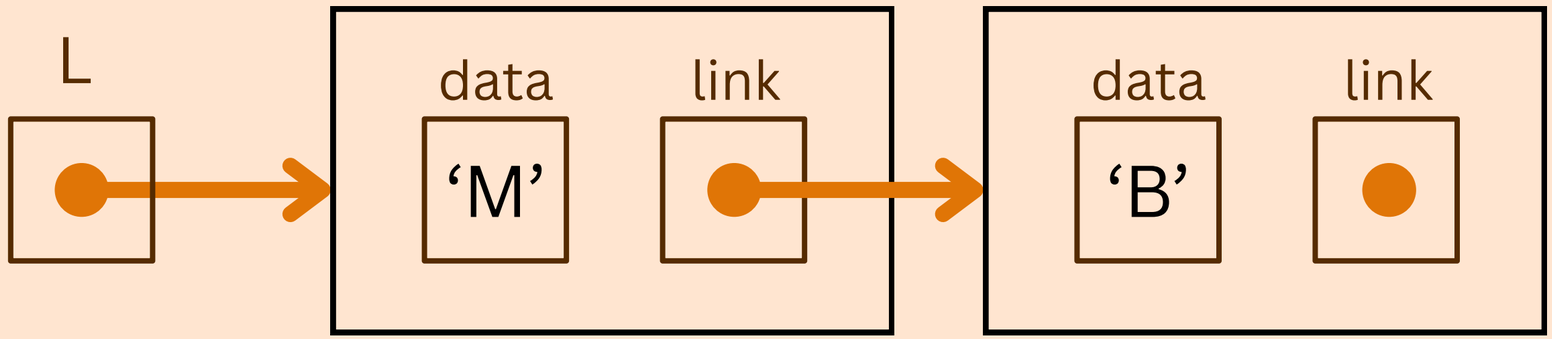
- a linear data structure that isn’t stored in a contiguous location
- linked with pointers
- self referencing structure



LINKED-LIST

```
typedef struct node{
    char data;
    struct node *link;
}celltype,*LIST;
```

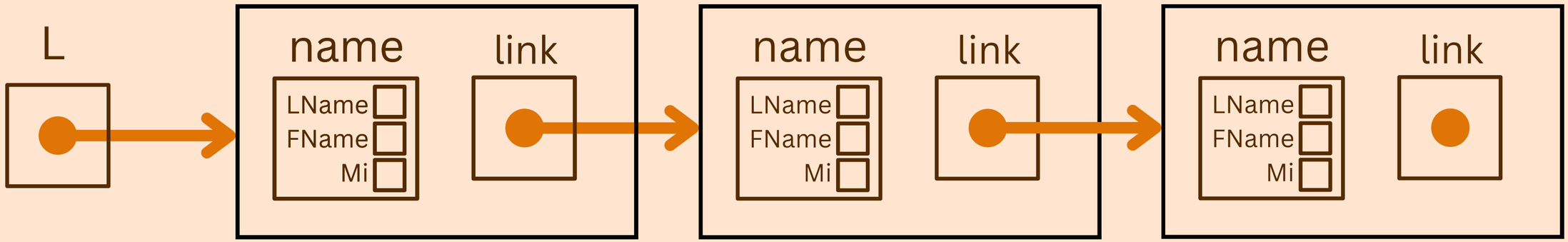
```
List L;
L=(LIST)malloc(sizeof(celltype));
L->data='M';
L->link=(LIST)malloc(sizeof(celltype));
L->link->data='B';
L->link->link=NULL;
```



LINKED-LIST

```
typedef struct{
   char LName[16];
   char FName[24];
   char Mi;
}Nametype;
```

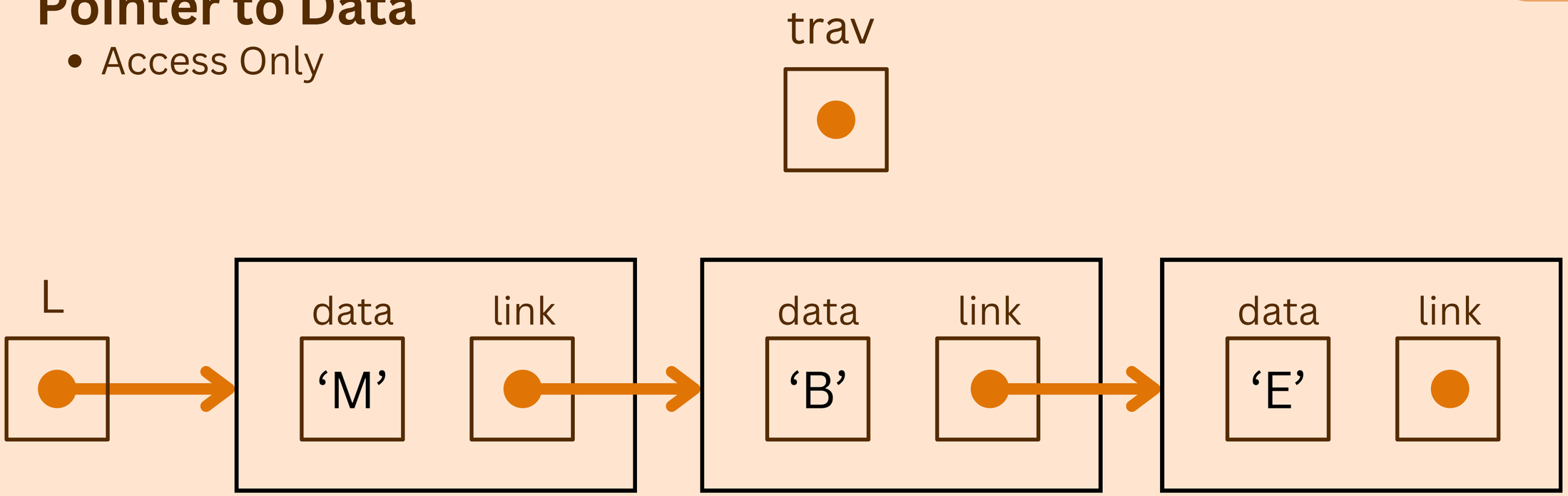
```
typedef struct node{
   Nametype name;
   struct node *link;
}celltype, *LIST;
```



LINKED-LIST TRAVERSAL

Pointer to Data

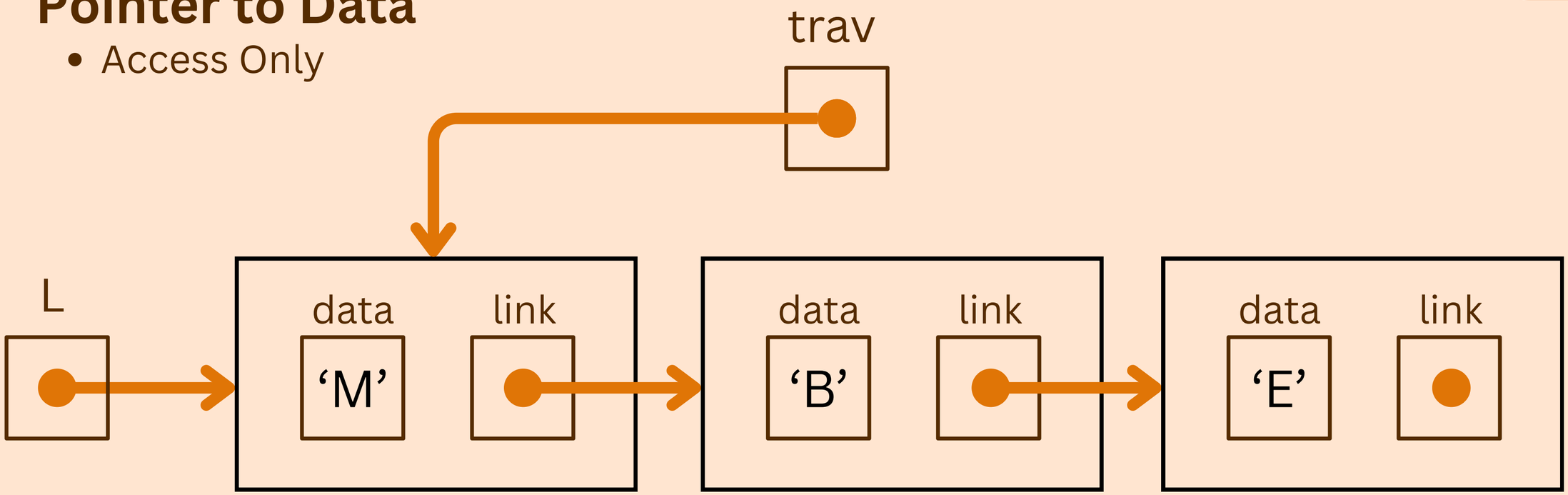
- Access Only



LINKED-LIST TRAVERSAL

Pointer to Data

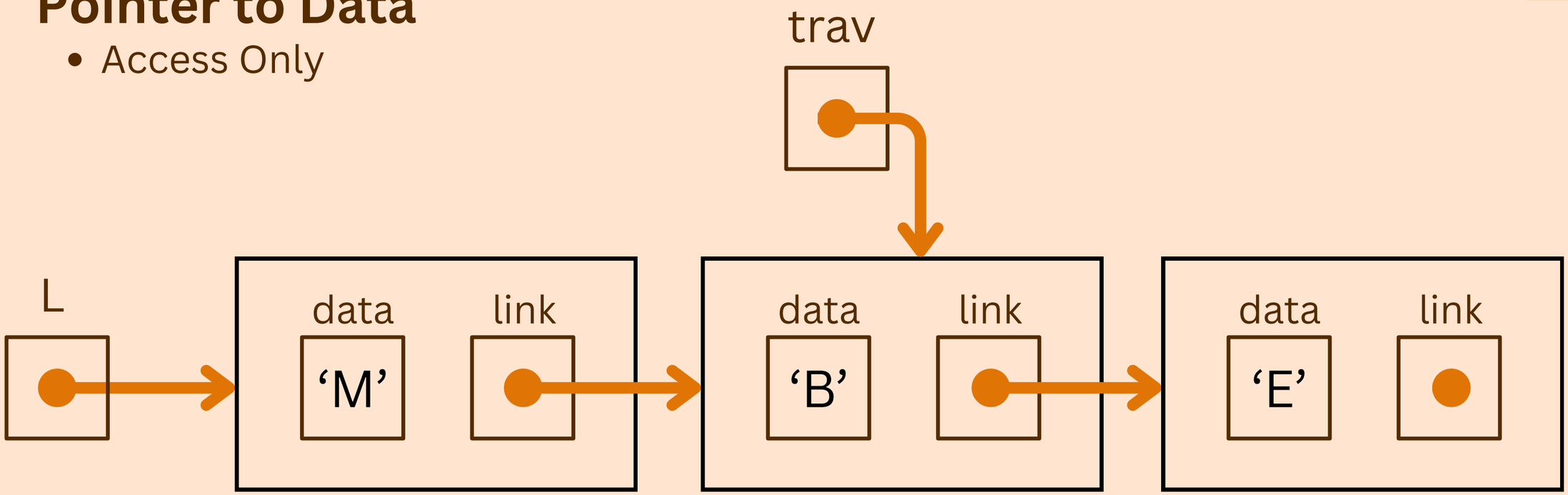
- Access Only



LINKED-LIST TRAVERSAL

Pointer to Data

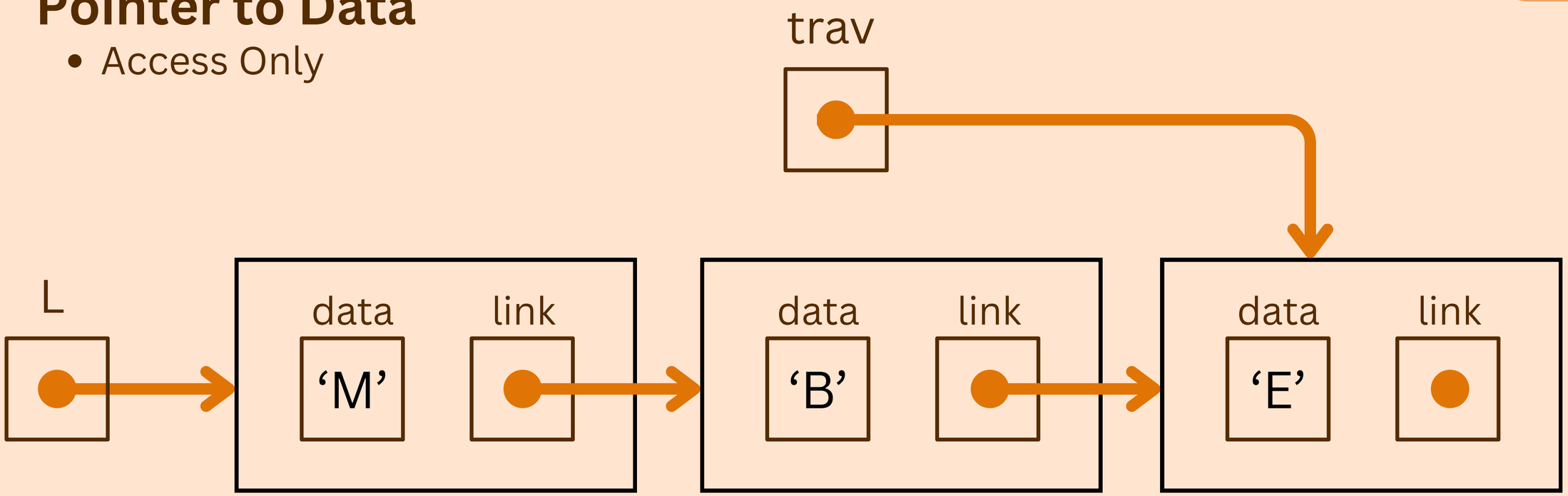
- Access Only



LINKED-LIST TRAVERSAL

Pointer to Data

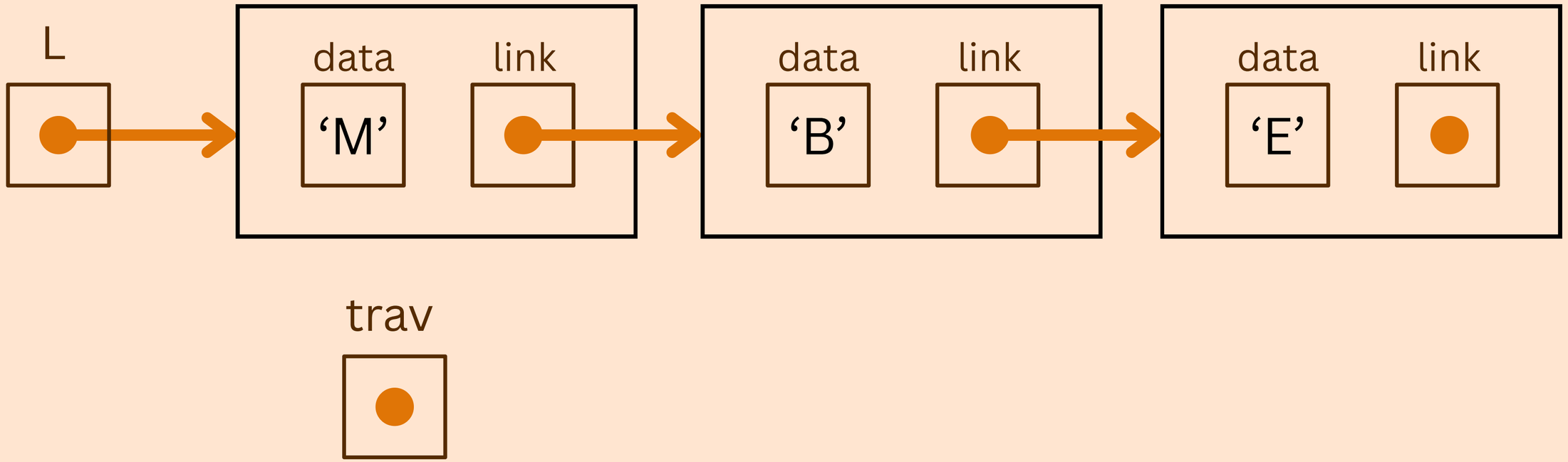
- Access Only



LINKED-LIST TRAVERSAL

Pointer to Data Pointer

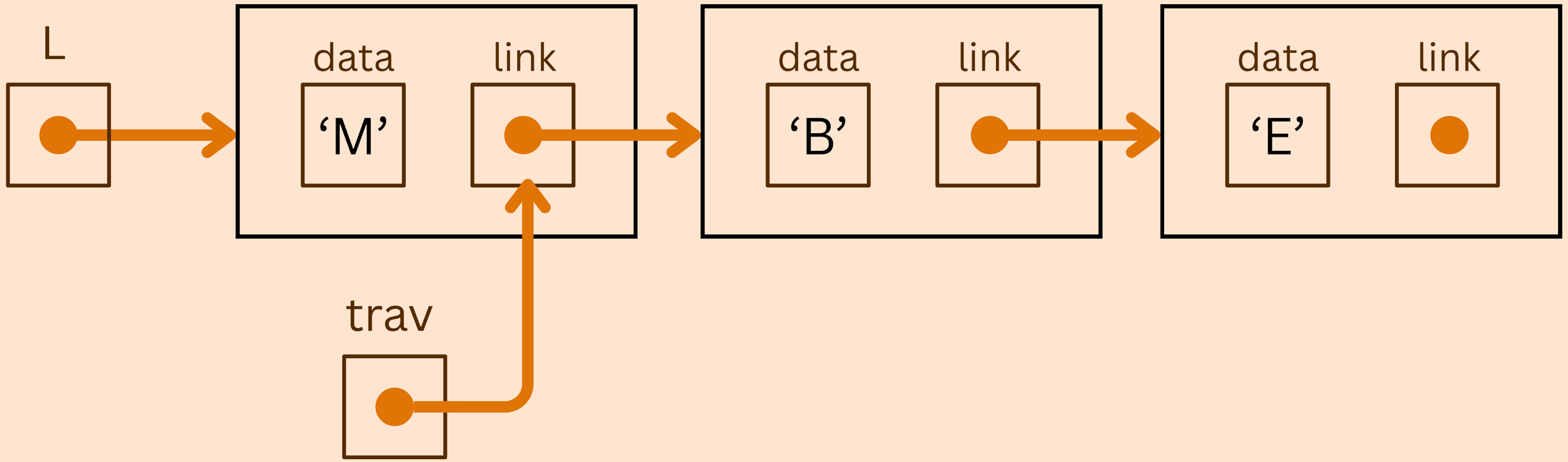
- Modifying pointer list



LINKED-LIST TRAVERSAL

Pointer to Data Pointer

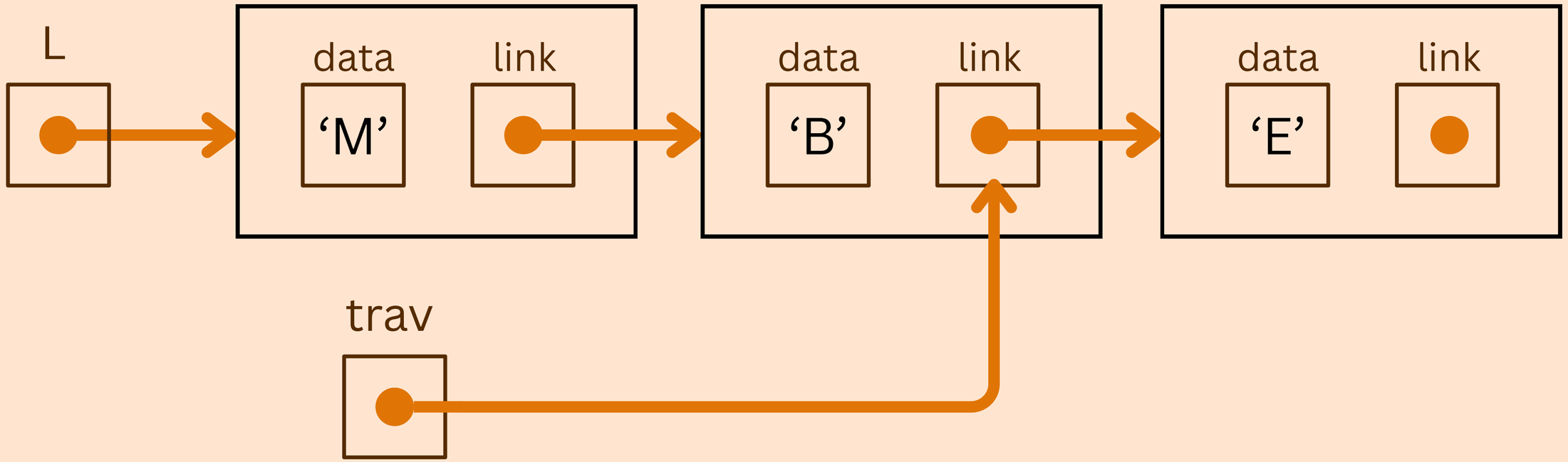
- Modifying pointer list



LINKED-LIST TRAVERSAL

Pointer to Data Pointer

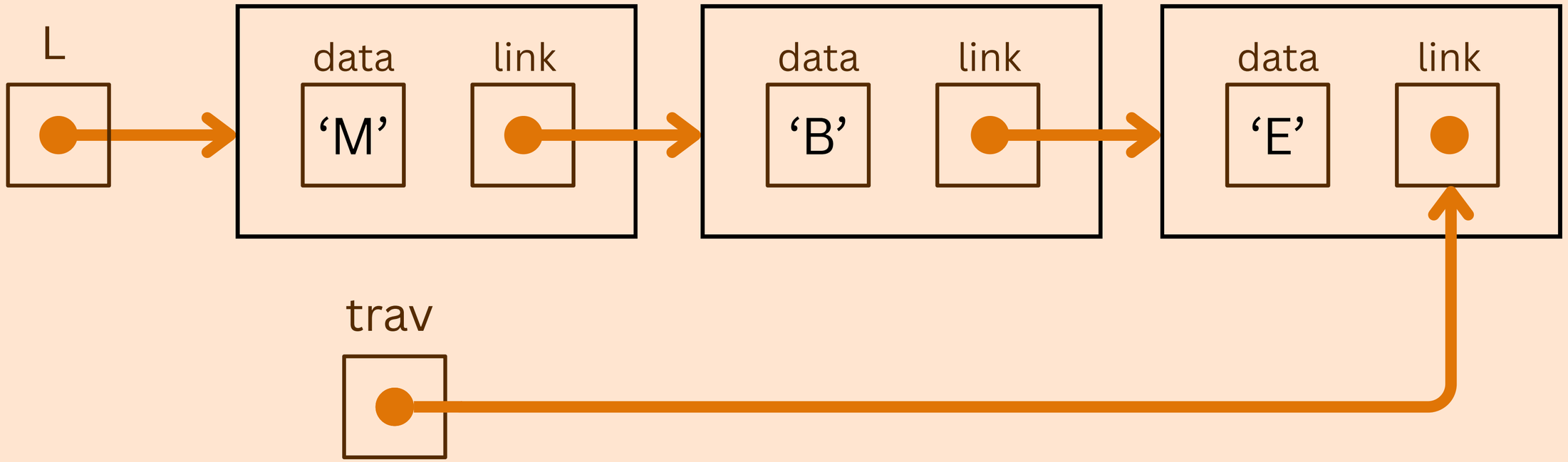
- Modifying pointer list



LINKED-LIST TRAVERSAL

Pointer to Data Pointer

- Modifying pointer list



THANK YOU