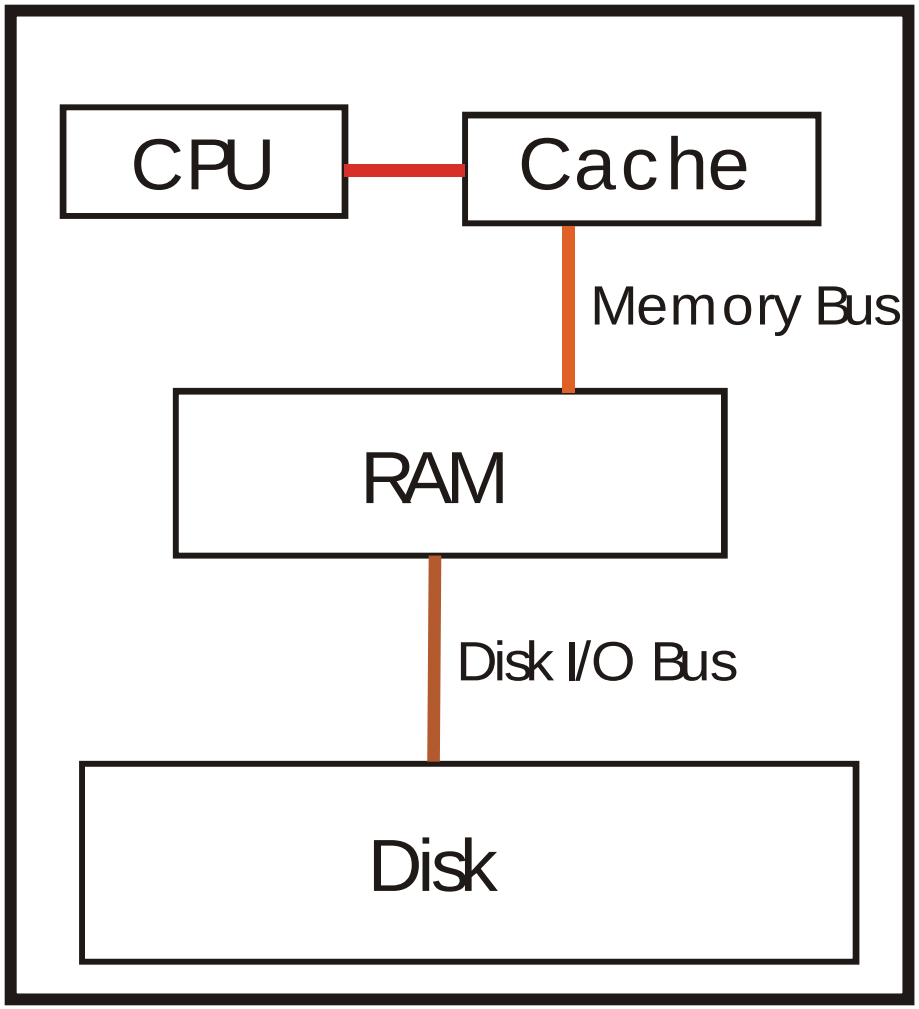
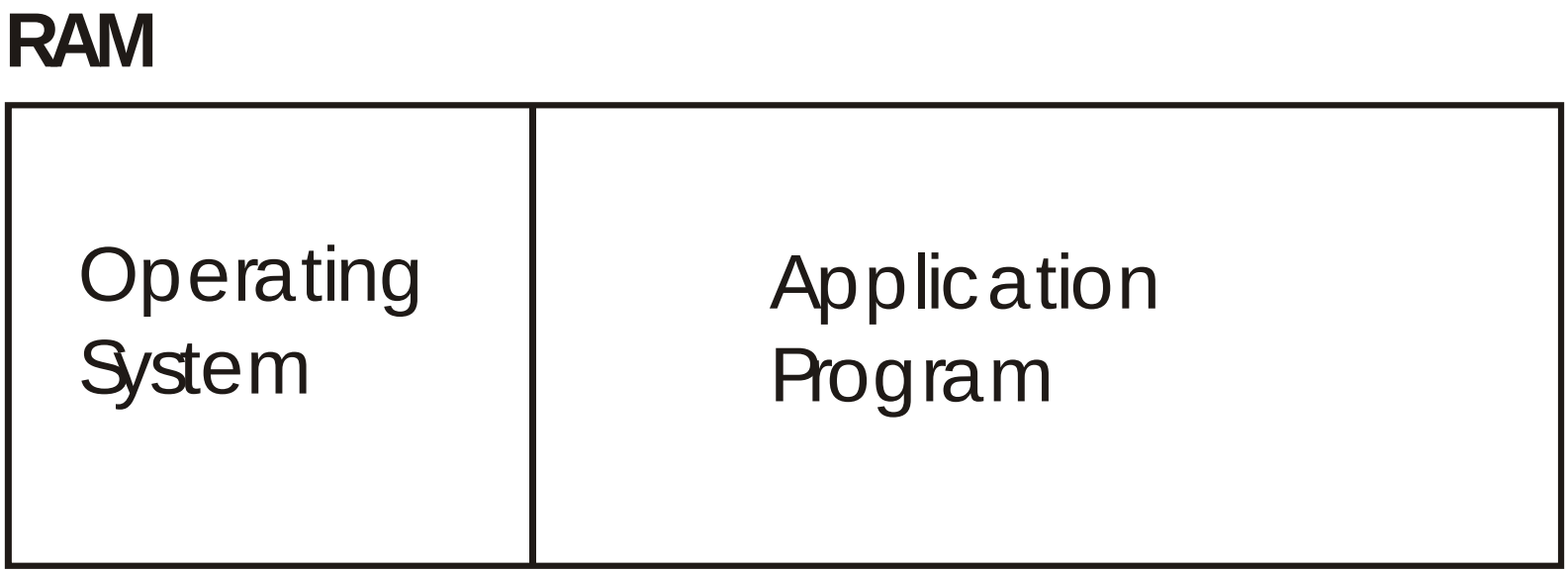
CEE432/CEE532/MAE541
Developing Software for Engineering Applications

Lecture 8: Pointers and All the Good Stuff
(Chapter 8)

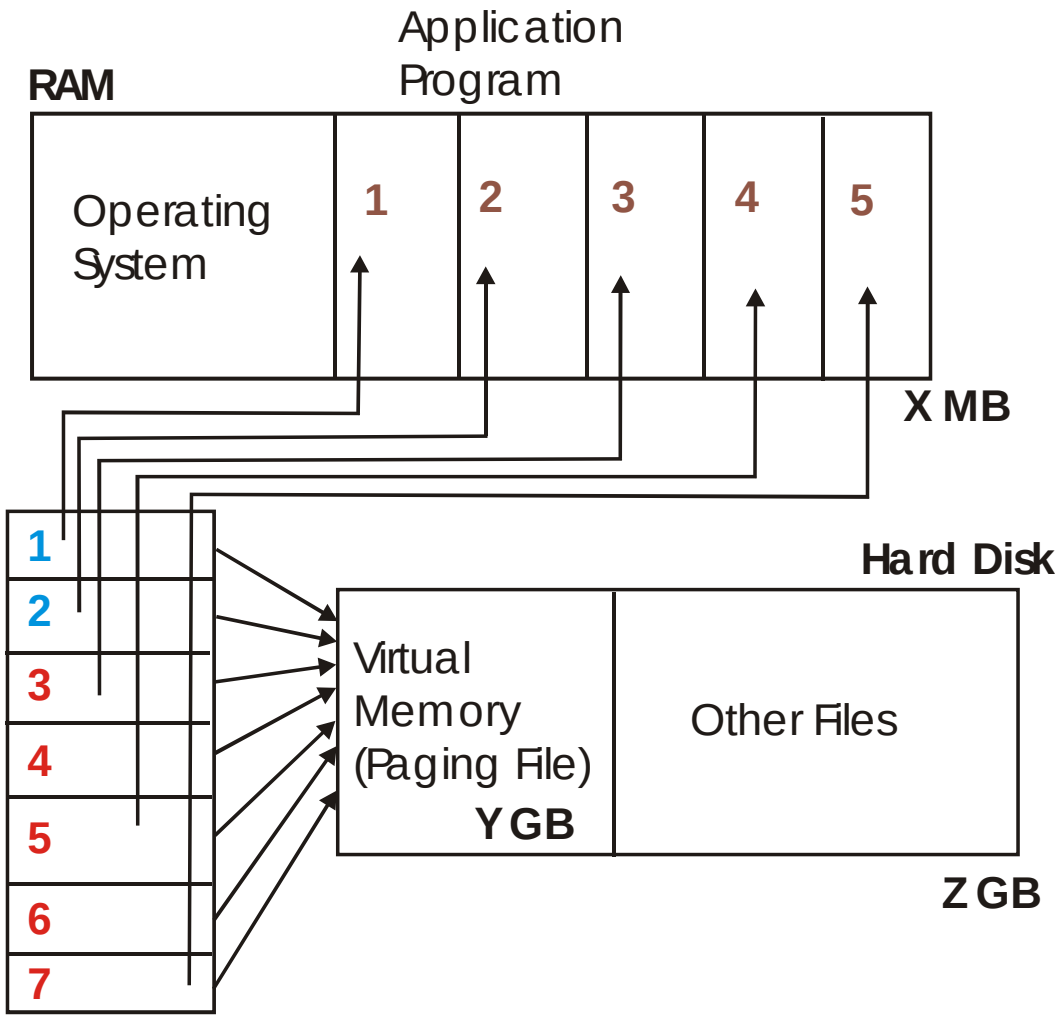
Memory Hierarchy



Random Access Memory



Memory Management



Application Virtual Memory Page Table

Memory Management

- Stack
  - The stack is used to store the automatic variables. The objects are created as soon as memory is allocated and destroyed immediately before memory is deallocated.
- Freestore (or Heap)
  - This memory area is used for dynamic memory allocation and is affected by **new** and **delete** operators. Memory for objects is allocated but this memory may not be immediately initialized. This memory may be accessed and manipulated outside of the object's lifetime but while the memory is still allocated.

Pointer Data Type

- Pointer data types represent a reference to an object or a location.
- Typically they hold the memory address.

```
int *pnX;
```

- Do **not** use this style

```
int* pnX;
```

Pointer Data Type

```
int *pnX, nX;
nX = 10;
pnX = &nX;
```

Referencing and Dereferening

- **&** is the reference operator
- ***** is the dereference operator

Calling with Pointer Variables

- Function prototype

```
void ABC (const int *nA, int *nB)
{ *nB = 2* (*nA);}
```

- Calling the function

```
int nA = 10, nB;
ABC (&nA, &nB);
```

- This is a poor case for using pointers. We can pass by reference

Dynamic Memory Allocation

- The **new** operator

```
pointer_variable = new typename;
```

- The **delete** operator

```
delete pointer_variable;
```

Dynamic Memory Allocation

```
float *pfX, fY;
pfX = new float;
*pfX = 43.5f;
fY = *pfX;
….
delete pfX;
```

- No particular advantage here

(Deprecated) NULL operator

- NULL is a special pointer with a special value (typically zero)
- Needs **`#include <stdlib.h>`**

```
pfX = new float;
if (pfX == NULL)
....
```

Exception Handling

```
try
{
   // statements including at least one of the following
    throw expression;
    …
}
catch (datatype_1 identifier)
{
   // exception handling statements
}
…
catch (datatype_n identifier)
{
   // exception handling statements
}
catch (…) // catch all block
{
   // exception handling statements
}
```

try ..throw

```
#include <stdexcept>  // exception handling
try
{
   fVX = new float[4];
}
catch (std::bad_alloc)
{
   std::cout << "Unable to allocate memory.\n";
   exit (1);
}
```

Using dynamic vector data type

```
float fVX[4];
```

is similar to

```
float *pfVX;
pfVX = new float[4];
…
delete [] pfVX;
```

- HUGE advantage here

Pointer Arithmetic

- Note

`*(pfVX+i)`

is the same as

`pfVX[i]`

- This is ‘exactly’ what we want!

Dynamically Allocated Vector

- We now are ready to look at implementing a class to handle dynamically allocated vector
- Poor man's vector class: **`CMyVector`**

CMyVector class

```
class CMyVector
{
    public:
        CMyVector ();           // default constructor
        CMyVector (int nRows);  // constructor
        ~CMyVector ();          // destructor

        // helper functions
        int GetSize () const { return m_nRows; };
        float At (int) const;
        float& At (int);
        void Display (const std::string&) const;
    public:
        float *m_pData;   // where the vector data are
    private:
        int m_nRows;      // # of rows
};
```

Client Code

```
#include <iostream>
#include "myvector.h"

int main ()
{
    // dynamically allocated vectors
    CMyVector fVX(3), fVY(3);
    // populate the two vectors
    for (int i=0; i < 3; i++)
    {
        fVX.m_pData[i] = static_cast<float>(i+1);
        fVY.At(i) = static_cast<float>((i+1)*(i+1));
    }
    float fDotP = fVX.DotProduct (fVY);
    std::cout << "Dot product of ...\n";
    fVX.Display ("    Vector X ");
    fVY.Display ("    Vector Y ");
    std::cout << "    is equal to " << fDotP << "\n";

    return 0;
}
```

-> and . operators

```
CPoint *pPoint12;
pPoint12 = new CPoint;
…
pPoint12->SetValues (1.2f,-17.65f);
or
(*pPoint12).SetValues (1.2f,-17.65f);
…
delete pPoint12;
```

Summary

- Pointers are variables and store an address rather than a value.
- They are bound to the data type that they are associated with.
- Address is obtained via the `&` operator and values are obtained via the `*` operator.
- Dynamic memory allocation and deallocation take place using the `new` and `delete` operators.

Summary

- Dynamic memory allocation can take place for C++ AND user-defined data types.
- For vector variables, memory is allocated contiguously.
- For pointers associated with objects, member functions are accessed via the `->` or the `.` operators.
- Indiscriminate use of pointers can lead to programming problems such as dangling pointers, memory leaks, or access violations etc.