Pointer Arithmetic

Pointer Arithmetic

- All programming languages support using binary operators such as addition and subtraction for the purpose of standard computer arithmetic, such as:

```
double a = 2, b=4;
double y = a + b;
```

- C/C++ in addition to standard arithmetic operations supports pointer arithmetic operations. It means you can use operators + (addition) and – (subtraction) to perform arithmetic operations on pointers.
    - Pointer arithmetic is generally useful only to refer to the elements of an array.
    - Adding an integer to or subtracting an integer from a pointer yields a pointer with the same type.

Pointer Arithmetic

- Legal pointer arithmetic in C++

  Pointer + Integer

  Integer + Pointer

  Pointer – Integer

  Pointer – Pointer

  Pointer++

  ++Pointer

  Pointer--

  --Pointer
- Other arithmetic operations are illegal.
- Examples of Illegal pointer arithmetic

  Integer – Pointer

  Pointer + Pointer.

  Pointer * Integer

  Pointer / Integer

  Etc…

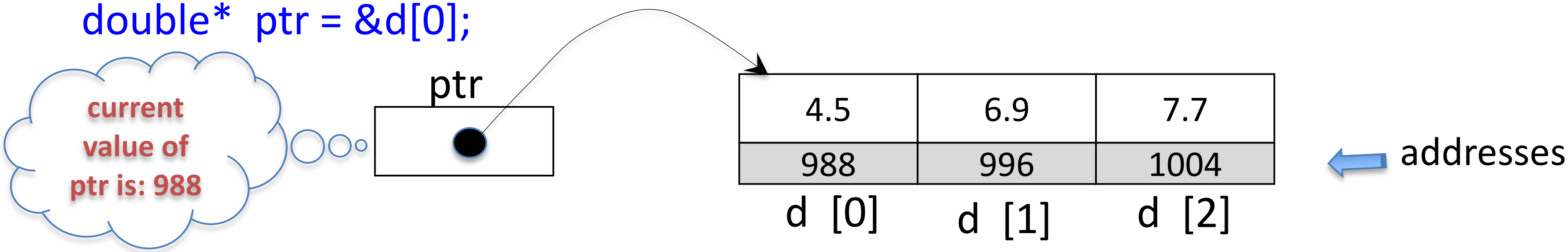
Pointer Arithmetic

- "pointer + n" refers to the address of nth element , from the current address.
- Assuming **n** is an integer and the **pointer** has a valid address value:

  pointer + n = = address_value + n * sizeof (type)

  Example:

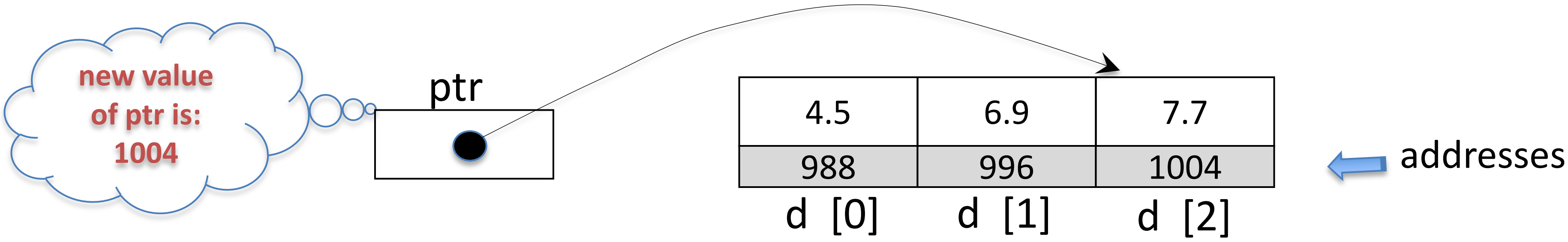
```
double d[3] = {4.5, 6.9, 7.7};
double*  ptr = &d[0];
```



current value of ptr is: 988

d [0]	d [1]	d [2]
4.5	6.9	7.7
988	996	1004

```
ptr = ptr + 2;   // The new value of ptr is 988 + 2 * 8 = 1004
```



new value of ptr is: 1004

d [0]	d [1]	d [2]
4.5	6.9	7.7
988	996	1004

- The new value of ***ptr** is 7.7

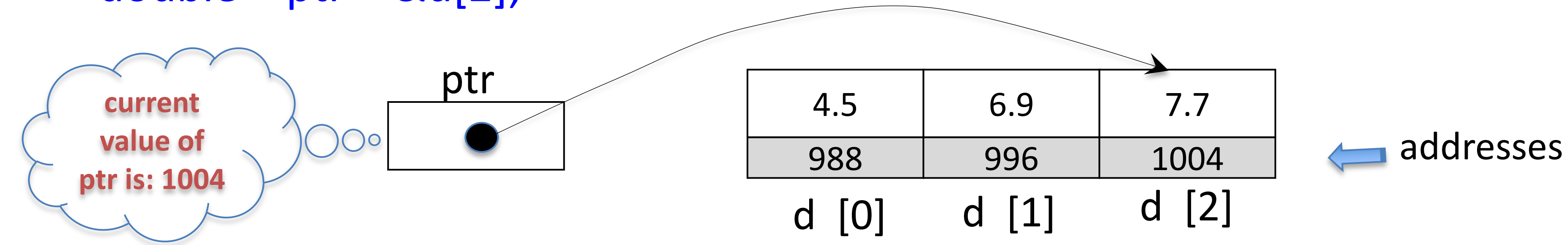
Pointer Arithmetic

- Assuming **n** is an integer and the **pointer** has a valid address value:

  pointer - n = = address_value - n * sizeof (type)

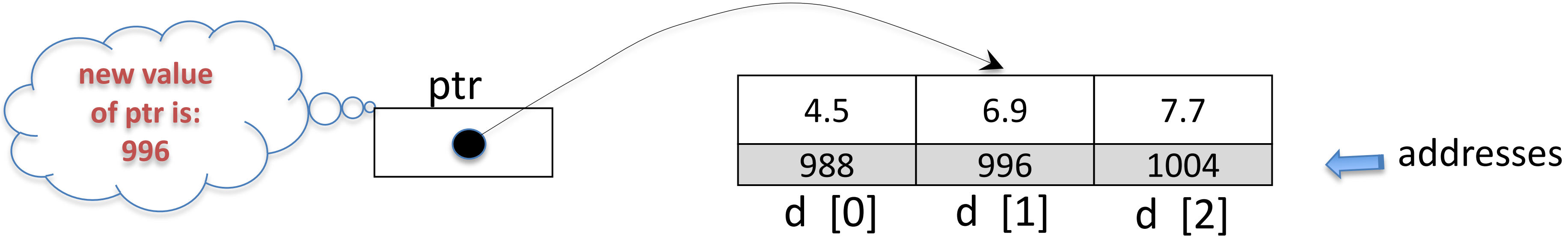
  Example:

```
double d[3] = {4.5, 6.9, 7.7};
double*  ptr = &d[2];
```



4.5	6.9	7.7
988	996	1004
d [0]	d [1]	d [2]

```
ptr--;   // The new value of ptr is 1004 – (1 * 8) = 996
```



4.5	6.9	7.7
988	996	1004
d [0]	d [1]	d [2]

- The new value of ***ptr** is 6.9

Pointer Arithmetic

- "Pointer1 – Pointer2", results in an integer value that represents the number of elements between the two pointers:

```
int arr[5] = {2, 6, 4, 7, 9};
int* ptr;
int diff;
ptr = arr + 5;    // ptr points to arr[5] after the last element
// Allowed to write:  ptr = 5 + arr;
diff = ptr – arr;
```

  - In this example the value of diff will be 5. Why?
    - If the address of first element of **arr** is **1000**, the value of **ptr** will be **1020**, assuming that size of int is 4 bytes, the value of **diff** is calculated as follows:

      diff = (1020 – 1000)/sizeof(int) = 20/4 = 5

More on Arrays and Pointers Notations

- Array notations and pointer notations are interchangeable.
- Based on pointer arithmetic rules explained in previous slides, you can replace a square bracket notation that refers to an element of the array with a pointer notation.
- Consider the following declarations:

```
int myArray[5] = { 31, 41, 22, 66, 90};
int* ptr = myArray + 2;
```

- The following statements are all true:

```
myArray == &myArray[0]
myArray[0] == *myArray
myArray[2] == *(myArray+2)
myArray + 2 == &myArray[2]
2 + myArray == &myArray[2]
ptr  + 2 == &ptr[2]
ptr + 2 == &myArray[4]
ptr – 2 == &ptr[-2];
*(ptr – 2) == ptr [-2]
```

Pointer Arithmetic

- To learn some of the applications of pointer arithmetic, lets take a look at different versions of a small c-string function that calculates the length of its c-string argument.
- The next few slides shows:
  - How array notations and pointer notations are interchangeable
  - How the same problem can be solved, using different ways
    - In terms of performance efficiency they are all almost the same and their possible differences are negligible.

Pointers Arithmetic

- **Version 1 – Using Array Notation**

```
int main ()
{
    int length;

    const char *s = “xyz”;

    length = my_strlen ( s );

    printf (“The string length is %d.”, length);

     return 0;
}
```

```
int my_strlen (const char* string)
 {

   int i = 0;

    while (string [i] !=  ‘\0’ )
    {
        i++;
    }
    // Draw AR diagram at this point
    return i;
}
```

- **Now, lets write a different version of my_strlen that uses pointer arithmetic.**

Pointers Arithmetic

- **Version 2 – Using Pointer Notation and Pointer Arithmetic**

```
int main ()
{
    int length;

    const char *s = “xyz”;

    length = my_strlen ( s );

    printf (“The string length is %d.”, length);

     return 0;
}
```

```
int my_strlen (const char* string)
 {

    int i = 0;

     while (*(string + i) !=  ‘\0’ )
     {
         i++;
     }
    // Draw AR diagram at this point
     return i;
}
```

- **Now, lets write a different version of my_strlen that uses pointer arithmetic.**

Pointers Arithmetic

- **Version 3 - This is another possible way, but not a better way?**

```
int main ()
{
    int length;

    const char *s = “xyz”;

    length = my_strlen ( s );

    printf (“The string length is %d.”,  length);

     return 0;
}
```

```
int my_strlen (const char* string)
 {

   int i = 0;

    while (*string !=  ‘\0’ )
    {
        string++;
        i++;
    }

    // Draw AR diagram at this point
     return i;
}
```

- **Can we write another version using “pointer – pointer” arithmetic?**

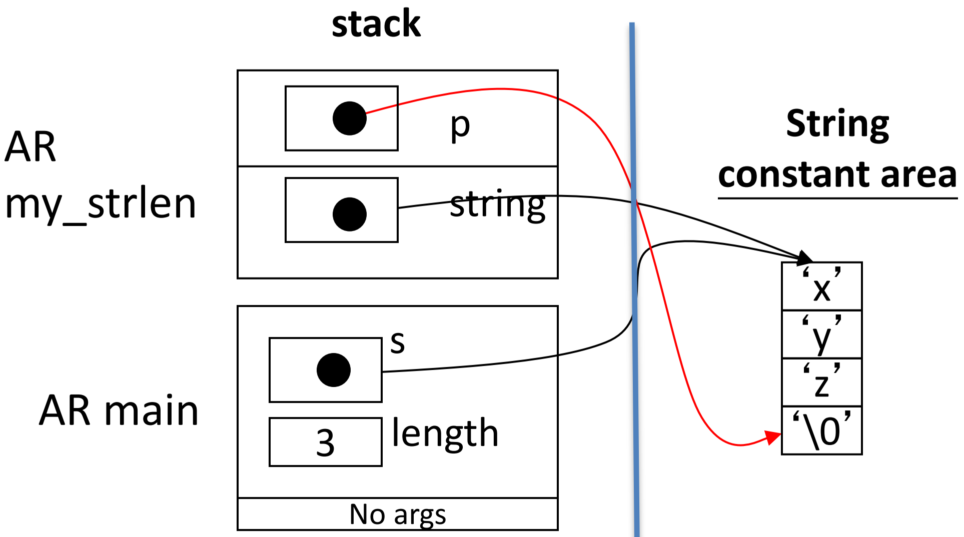
Pointers Arithmetic

- **Version 4 - This is another possible way, but not a better way?**

```
int main () {
    int length;
    const char *s = “xyz”;

    length = my_strlen ( s )
    printf (“The string length is %d.”, length);

     return 0;
}
```

```
int my_strlen (const char* string)  {
    const char *p = string

   while (*p != '\0')
         p++;
    // Draw AR diagram at this point

    return (int) (p - string);
}
```

AR at POINT ONE:


stack
AR
my_strlen
p
string
String
constant area
‘x’
‘y’
‘z’
‘\0’
s
3
length
AR main
No args