Parallel Computing with GPUs

Memory
Part 1 – Pointers

The University Of Sheffield.

Dr Paul Richmond

http://paulrichmond.shef.ac.uk/teaching/COM4521/

This Lecture (learning objectives)

- Pointers
  - Identify and use pointers and differentiate pointers from variables
- Pointers and arrays
  - Recognise the relationship between arrays and pointers
- Pointer arithmetic
  - Operate on pointers using simple arithmetic and predict how arithmetic operators make a pointers value change

Pointers

- A pointer is a variable that contains the address of a variable
- Pointers and arrays are closely related
  - We have already seen some of the syntax with * and & operators
- The * operator can be used to define a pointer variable
- The operator & gives the address of a variable
  - Can not be applied to expressions or constants

```
#include <stdio.h>

void main()
{
    int a;
    int *p;

    a = 8;
    p = &a;
}
```

Pointer example

```
int a = 8;
int *p = &a;
```

```
printf("a = %d, p = %d\n", a, p);
printf("a = %d, p = 0x%08X\n", a, p);
```

```
a = 8, p = 2750532
a = 8, p = 0x0045FCE0
```

- Same example using a `char`

```
char b;
char *p;
b = 8;
p = &b;
printf("sizeof(b) = %d, sizeof(p) = %d\n", sizeof(b), sizeof(p));
printf("b = %d, p = 0x%08X\n", b, p);
```

- What is the size of `p`?

Pointer example

```
printf("a = %d, p = %d\n", a, p);
printf("a = %d, p = 0x%08X\n", a, p);
```

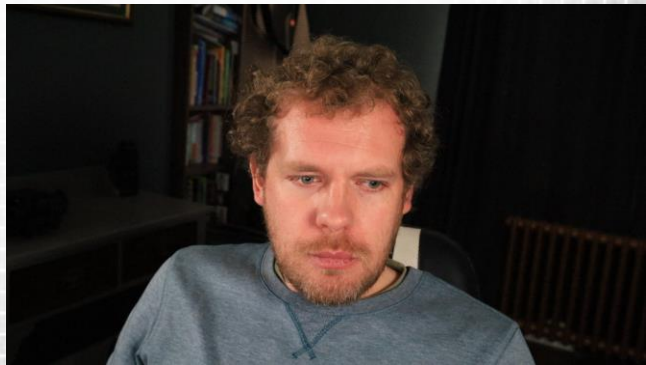
```
a = 8, p = 2750532
a = 8, p = 0x0045FCE0
```

- Same example using a `char`

```
char b;
char *p;
b = 8;
p = &b;
printf("sizeof(b) = %d, sizeof(p) = %d\n", sizeof(b), sizeof(p));
printf("b = %d, p = 0x%08X\n", b, p);
```

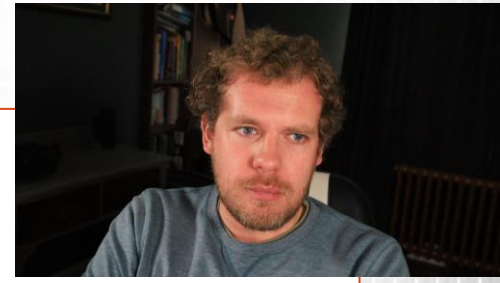
- What is the size of `p`?

```
sizeof(b) = 1, sizeof(p) = 4
b = 8, p = 0x003BF9A7
```



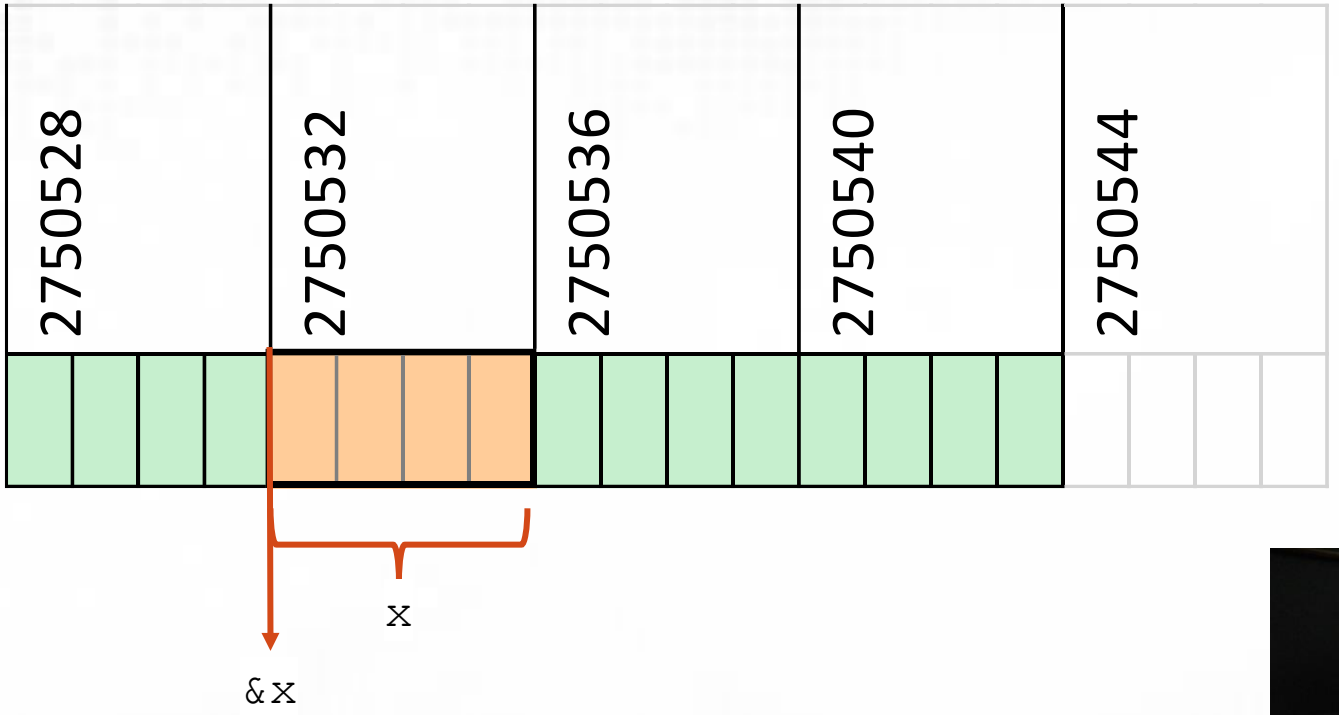
Pointers

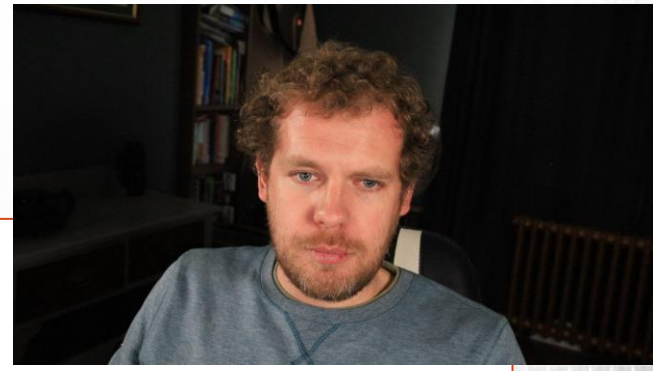
- Pointer size does not change regardless of what it points to
  - The size of a pointer on a 32 bit machine is always 4 bytes
  - The size of a pointer on a 64 bit machine is always 8 bytes
- The operator * is the indirection operator and can be used to dereference a pointer
  - I.e. it accesses the value that a pointer points to…
- The macro NULL can be assigned to a pointer to give it a value 0
  - This is useful in checking if a pointer has been assigned



```
int x = 1; int y = 0;
int *p;
p = &x; // p now points to x (value is address of x)
y = *p; // y is now equal to the value of what p points to (i.e. x)
x++;    // x is now 2 (y is still 1)
(*p)++; // x is now 3 (y is still 1)
p = NULL// p is now 0
```

Pointers





```
int x = 1; int y = 0;
int *p;
p = &x; // p now points to x (value is address of x)
y = *p; // y is now equal to the value of what p points to (i.e. x)
x++;    // x is now 2 (y is still 1)
(*p)++; // x is now 3 (y is still 1)
p = NULL// p is now 0
```

Pointers and arguments

- C passes function arguments by value
  - They can therefore only be modified locally

```
void swap (int x, int y){
    int temp;
    temp = x;
    x = y;
    y = temp;
}
```

- This is ineffective
  - Local copies of `x` and `y` are exchanged and then discarded

Pointers and arguments

- C passes function arguments by value
  - They can therefore only be modified locally

```
void swap (int *x, int *y){
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
```

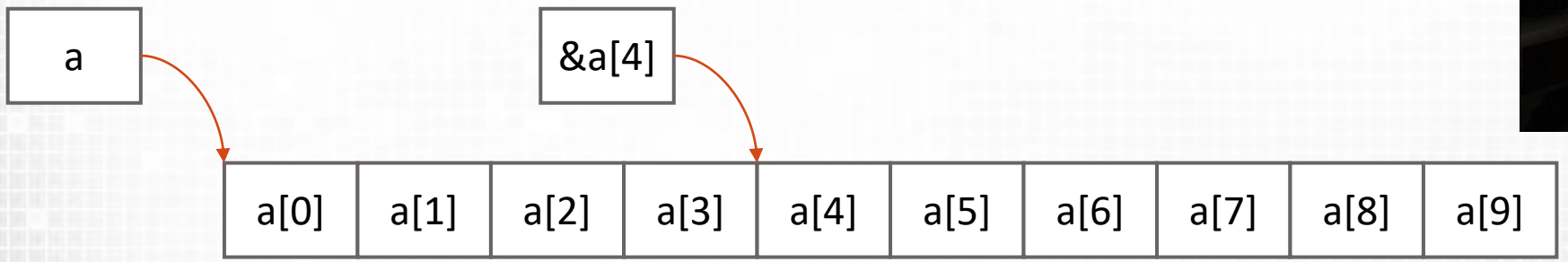
- This swaps the values which x and y point to
- Called by using the & operator

```
swap(&x, &y);
```

Pointers and Arrays

- In the last lecture we saw pointer being used for arrays
  - `char *name` is equivalent to `char name []`
- When we declare an array at compile time the variable **is a pointer** to the starting address of the array
  - E.g. `int a[10];`

```
int a[10] = {1,2,3,4,5,6,7,8,9,10};
int *p;
p = &a[4];
printf("*p=%d, p[0]=%d\n", *p, p[0]);
```

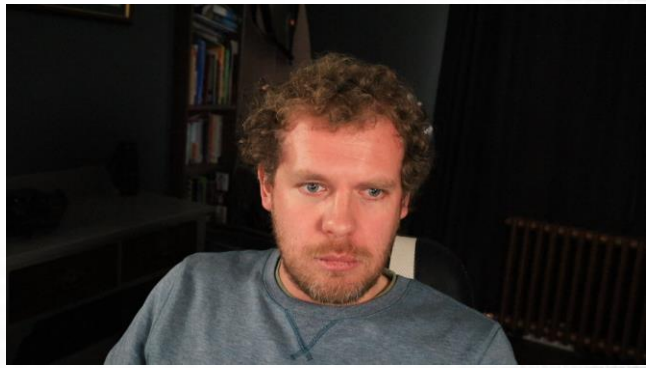


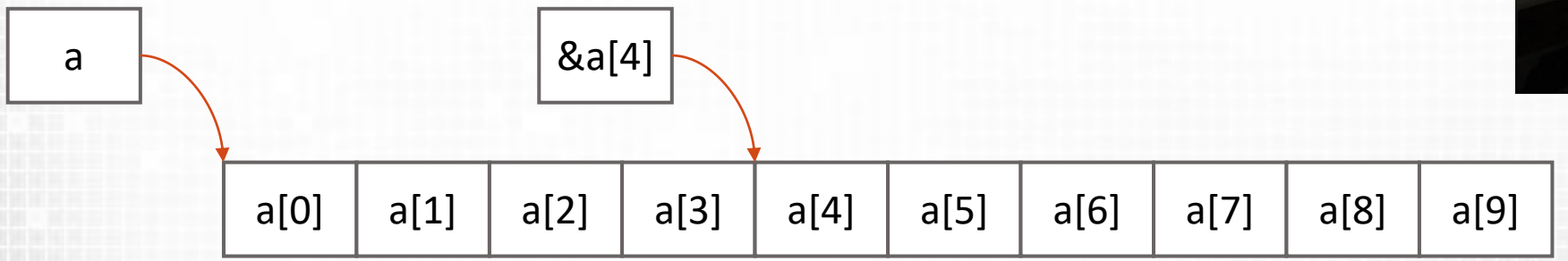
What is the output?

Pointers and Arrays

- In the last lecture we saw pointer being used for arrays
  - `char *name` is equivalent to `char name []`
- When we declare an array at compile time the variable **is a pointer** to the starting address of the array
  - E.g. `int a[10];`

```
int a[10] = {1,2,3,4,5,6,7,8,9,10};
int *p;
p = &a[4];
printf("*p=%d, p[0]=%d\n", *p, p[0]);
```






```
*p=5, p[0]=5
```

Pointer and Arrays

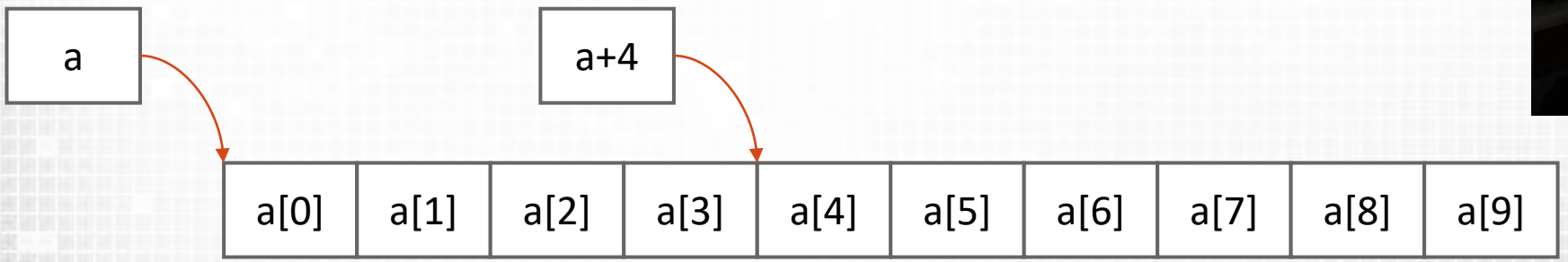
- There is however an important distinction between `char *name` and `char name []`
- Consider the following
  - The pointer may be modified
  - The array can only refer to the same storage

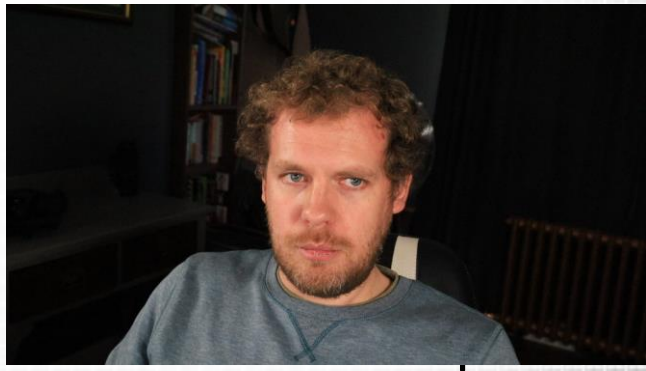
```
char a[] = "hello world 1";
char *b = "hello world 2";
char *temp;
temp = b;
b = a;
a = temp; //ERROR
```

Pointer arithmetic

- Pointer can be manipulated like any other value
  - `p++:` advances the pointer the next element
  - Pointer arithmetic must not go beyond the bounds of an array
- Incrementing a pointer increments the memory location depending on the pointer type
  - An single integer *pointer* will increment 4 bytes to the next integer

```
int a[10] = {10,9,8,7,6,5,4,3,2,1};
int *p = a;
p+=4;
printf("*p=%d, p[0]=%d\n", *p, p[0]);
```



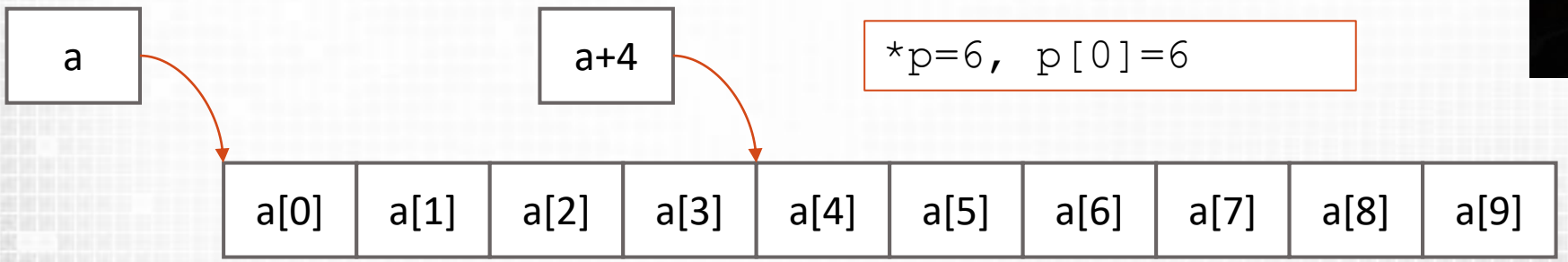


The University Of Sheffield.

Pointer arithmetic

- Pointer can be manipulated like any other value
  - `p++:` advances the pointer the next element
  - Pointer arithmetic must not go beyond the bounds of an array
- Incrementing a pointer increments the memory location depending on the pointer type
  - An single integer *pointer* will increment 4 bytes to the next integer

```
int a[10] = {10,9,8,7,6,5,4,3,2,1};
int *p = a;
p+=4;
printf("*p=%d, p[0]=%d\n", *p, p[0]);
```




```
*p=6, p[0]=6
```

This Lecture (learning objectives)

- Pointers
  - Identify and use pointers and differentiate pointers from variables
- Pointers and arrays
  - Recognise the relationship between arrays and pointers
- Pointer arithmetic
  - Operate on pointers using simple arithmetic and predict how arithmetic operators make a pointers value change

- Next Lecture: Advanced use of Pointers