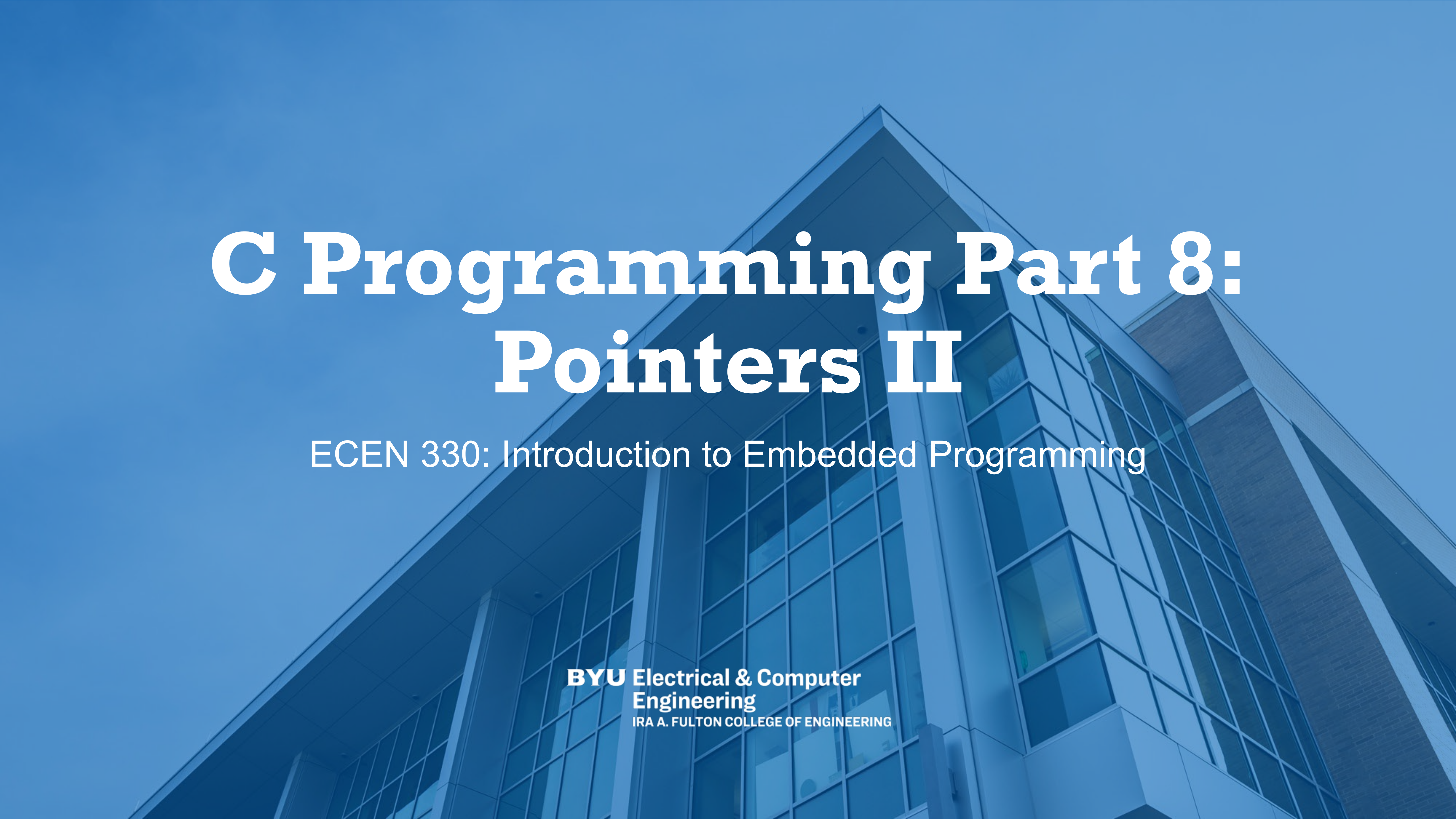


C Programming Part 8: Pointers II

ECEN 330: Introduction to Embedded Programming

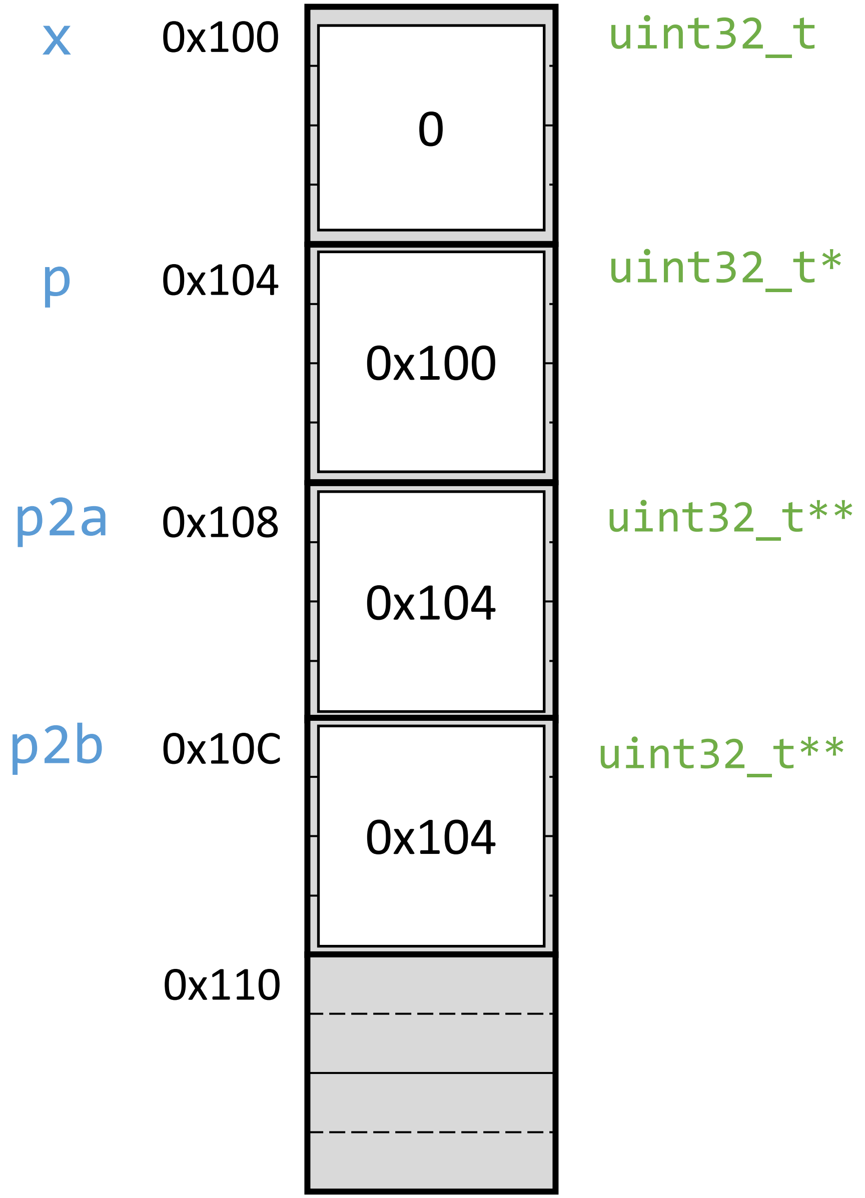
BYU Electrical & Computer Engineering
IRA A. FULTON COLLEGE OF ENGINEERING

Double Pointers

BYU Electrical & Computer Engineering
IRA A. FULTON COLLEGE OF ENGINEERING

```
uint32_t x = 0;
uint32_t* p;
uint32_t** p2a;
uint32_t** p2b;

p = &x;
p2a = &p;
p2b = &(*p2a);
```



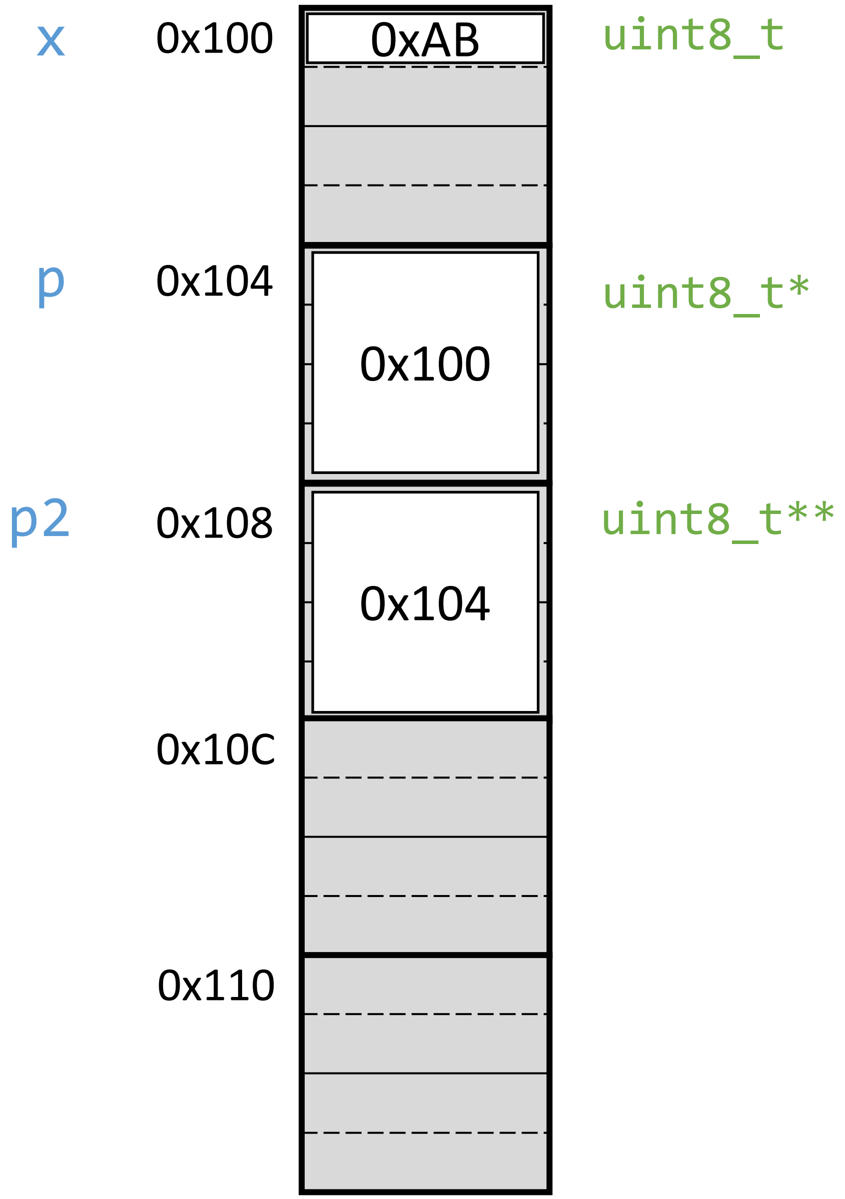
Rule: Dereference can only be done on pointer types (Compiler will check this)

BYU Electrical & Computer Engineering
IRA A. FULTON COLLEGE OF ENGINEERING

```
uint8_t x = 0xAB;
uint8_t* p = &x;
uint8_t** p2 = &p;

printf("%x\n", p);
printf("%x\n", *p);
printf("%x\n", **p);

printf("%x\n", p2);
printf("%x\n", *p2);
printf("%x\n", **p2);
printf("%x\n", ***p2);
```



Pointers To Structs

BYU Electrical & Computer Engineering
IRA A. FULTON COLLEGE OF ENGINEERING

We often have pointers to structs.

To access struct members you can dereference the pointer (*) and access (.)

Or you can do both at once with ->

```
struct point {
  int x;
  int y;
};

struct point s1 = {1,2};
struct point *p;

p = &s1;
(*p).x = 3;
p->y = 4;
printf("%d, %d\n", p->x, p->y);
```

So Why Use Pointers?

1. Change data in caller function
   - Using this you can pass data back to caller (ie have multiple return values)

2. Passing large pieces of data to function
   - In minimax, we passed the board by pointer

3. Enables many types of data structures (lists, trees)