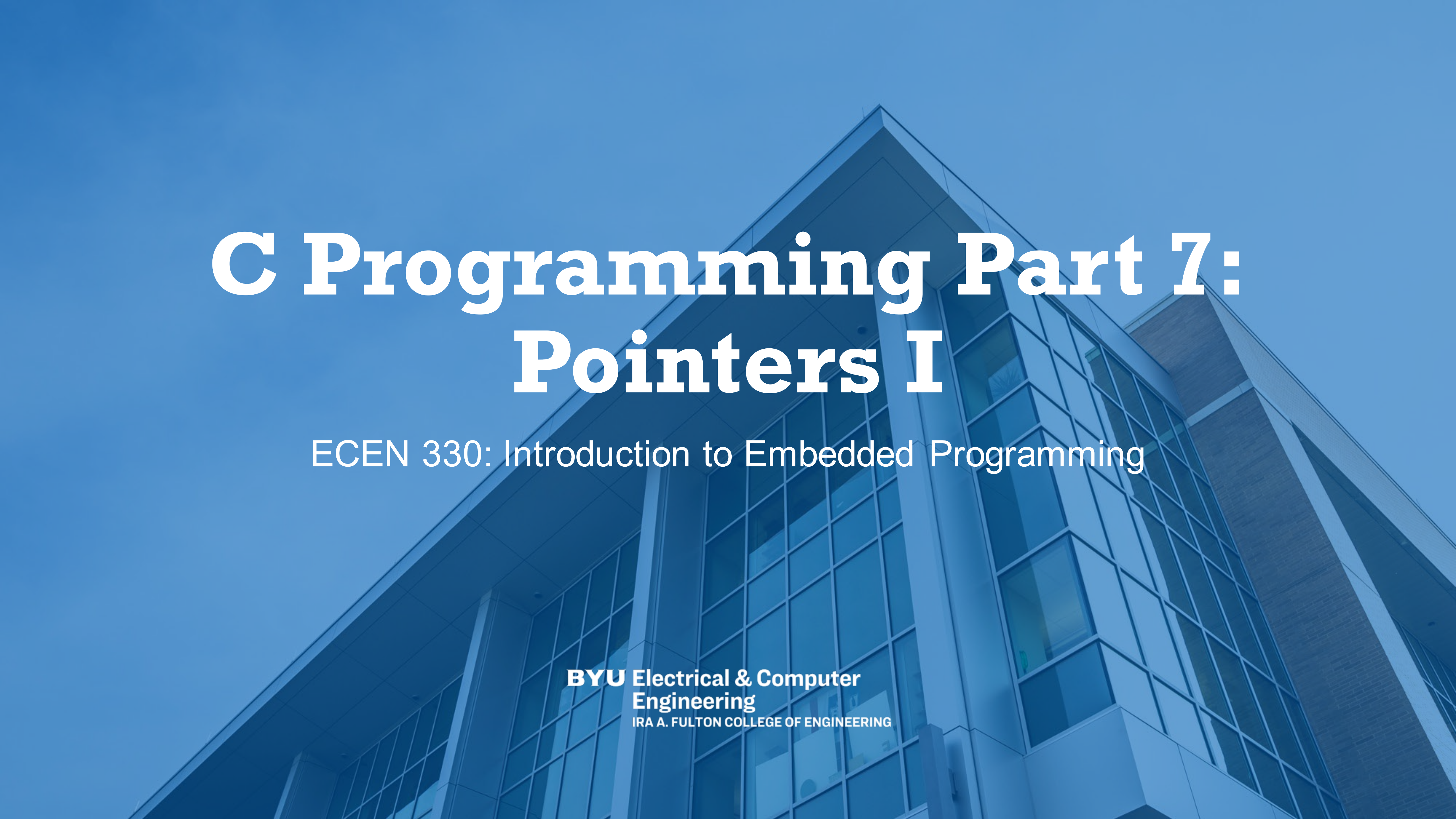


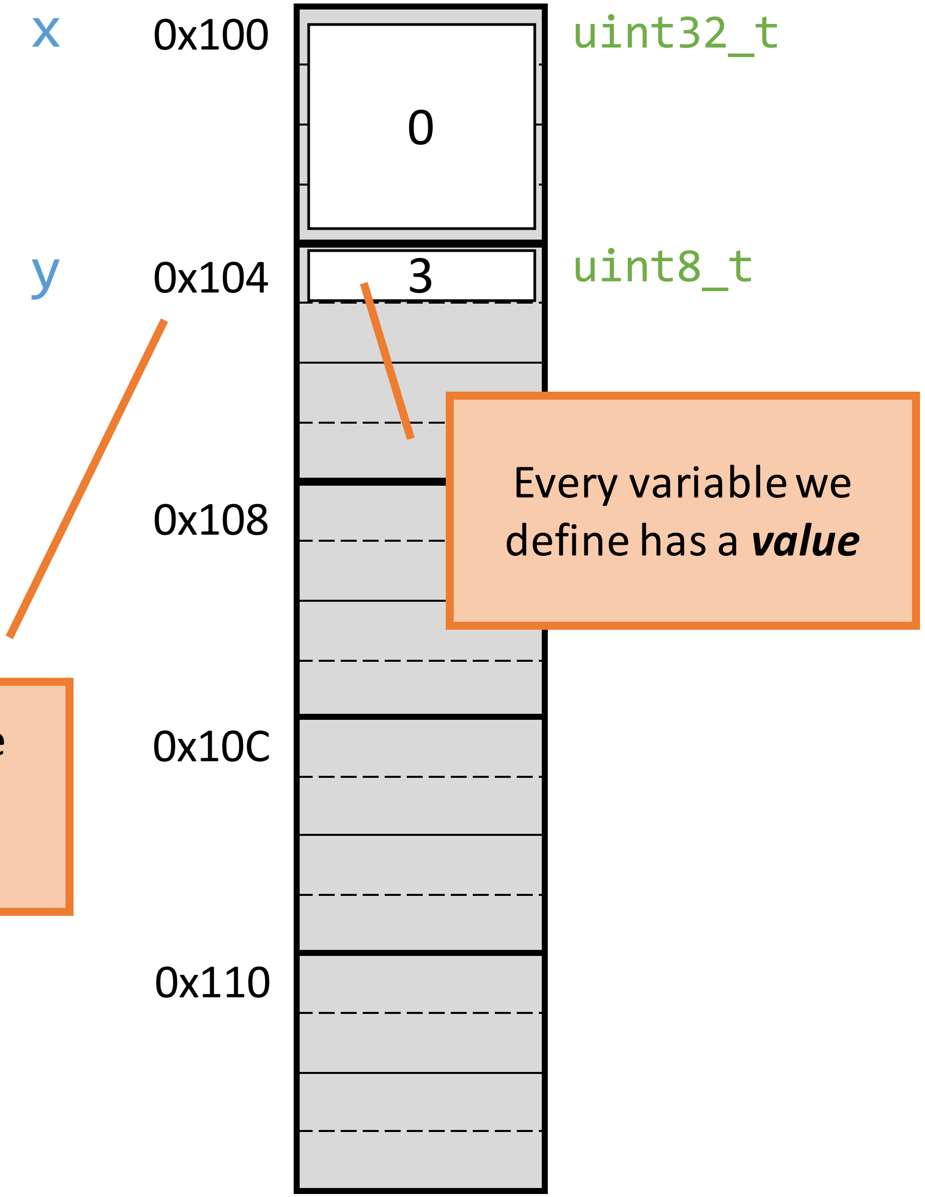
C Programming Part 7: Pointers I

ECEN 330: Introduction to Embedded Programming

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

Let’s back up…

```
uint32_t x = 0;
uint8_t y = 3;
```



Every variable we define has a ***value***

Every variable we define lives at an ***address***

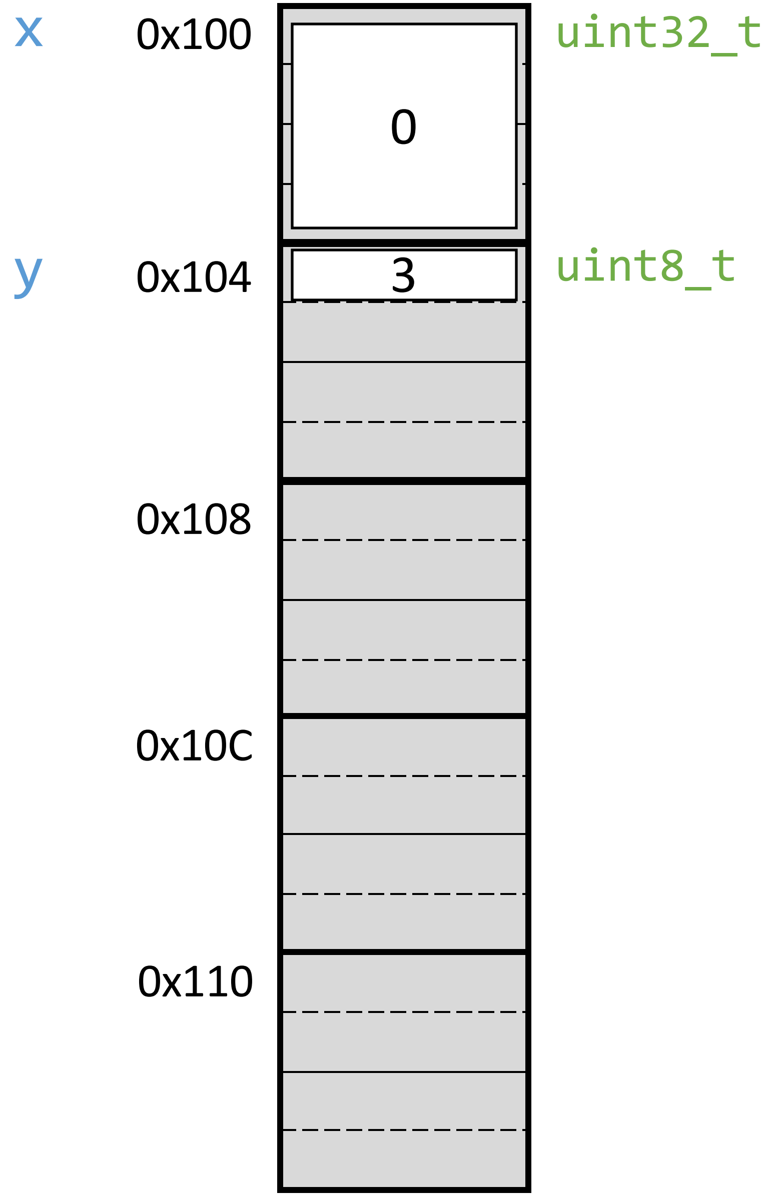
Let's back up...

```
uint32_t x = 0;
uint8_t y = 3;

y = x + 5;
```

We can assign to a variable to *update its value.*

We can read from a variable to **get its value.**



Pointers - &

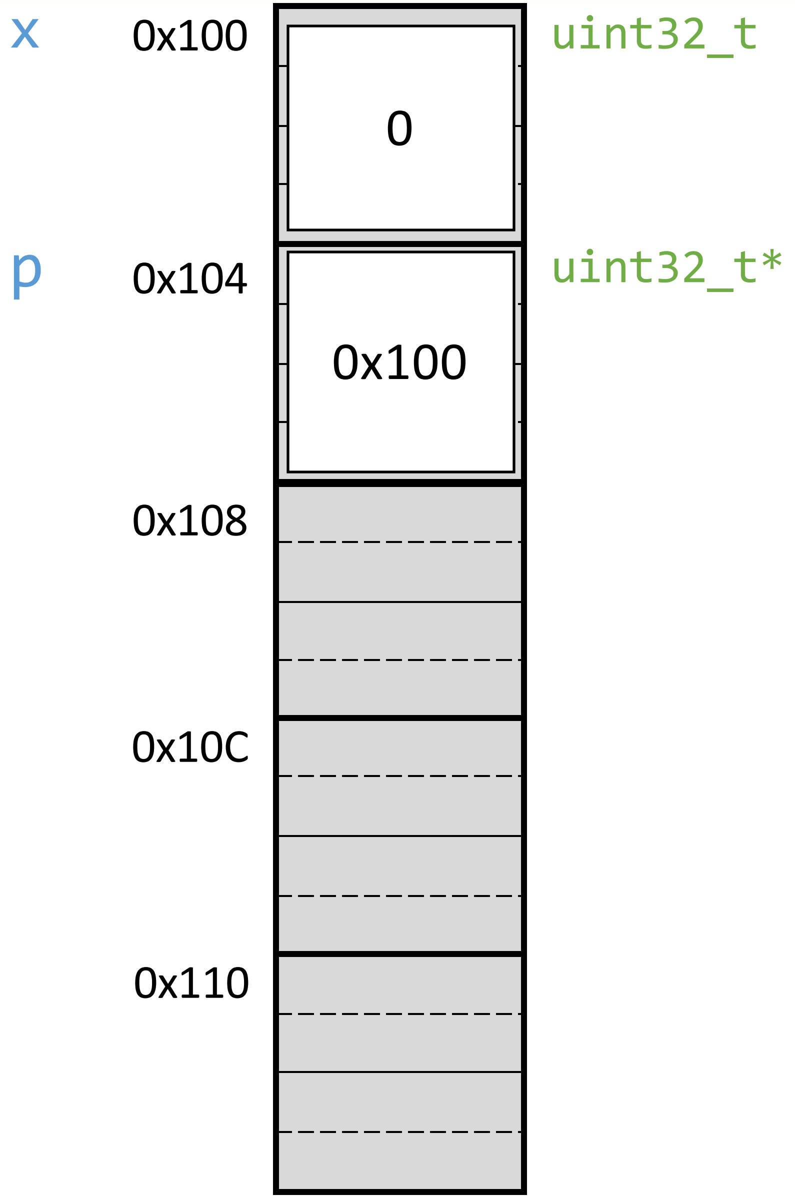
```
uint32_t x = 0;
uint32_t* p;


p = &x;
```

We can use ‘*’ to declare pointers **(address types).**

We can use ‘&’ on a variable to **get its address.**

We can assign to a variable to ***update its value.***



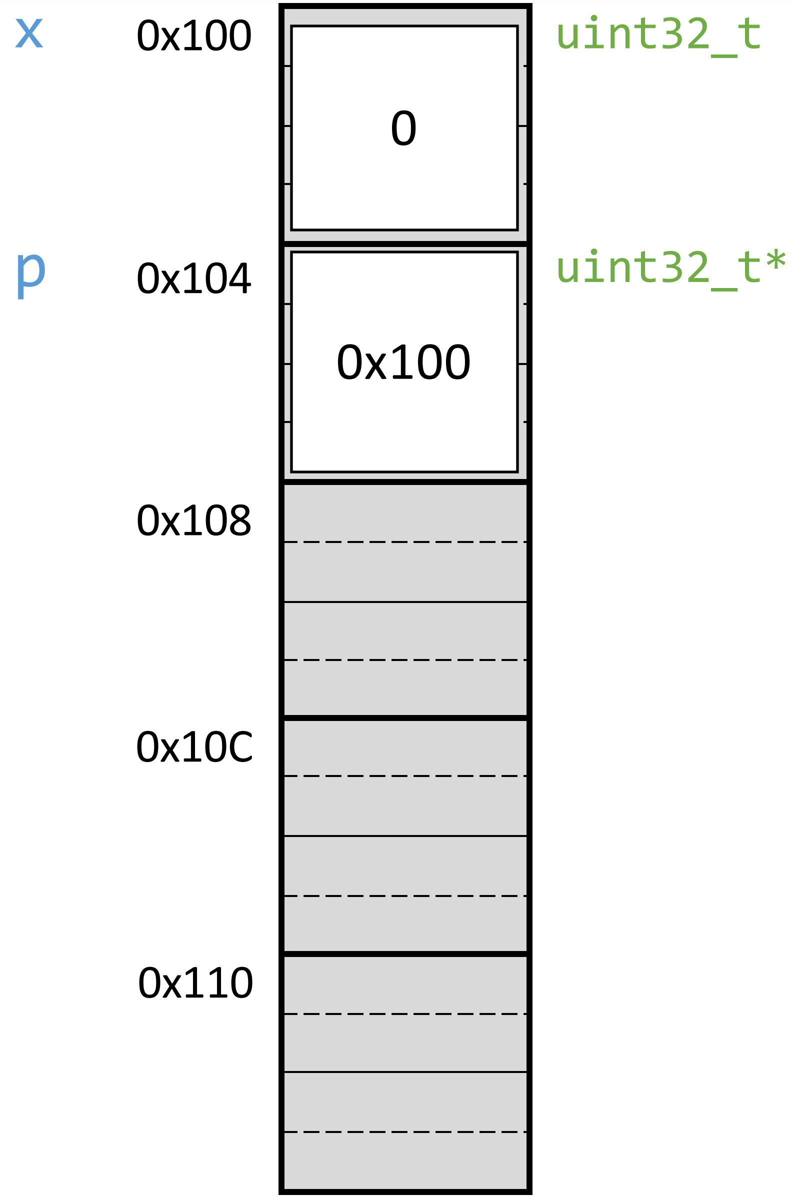
Pointers - *

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

p = &x;

*p = 13;
```

The * operator accesses the **value this address points to.**



Pointers - *

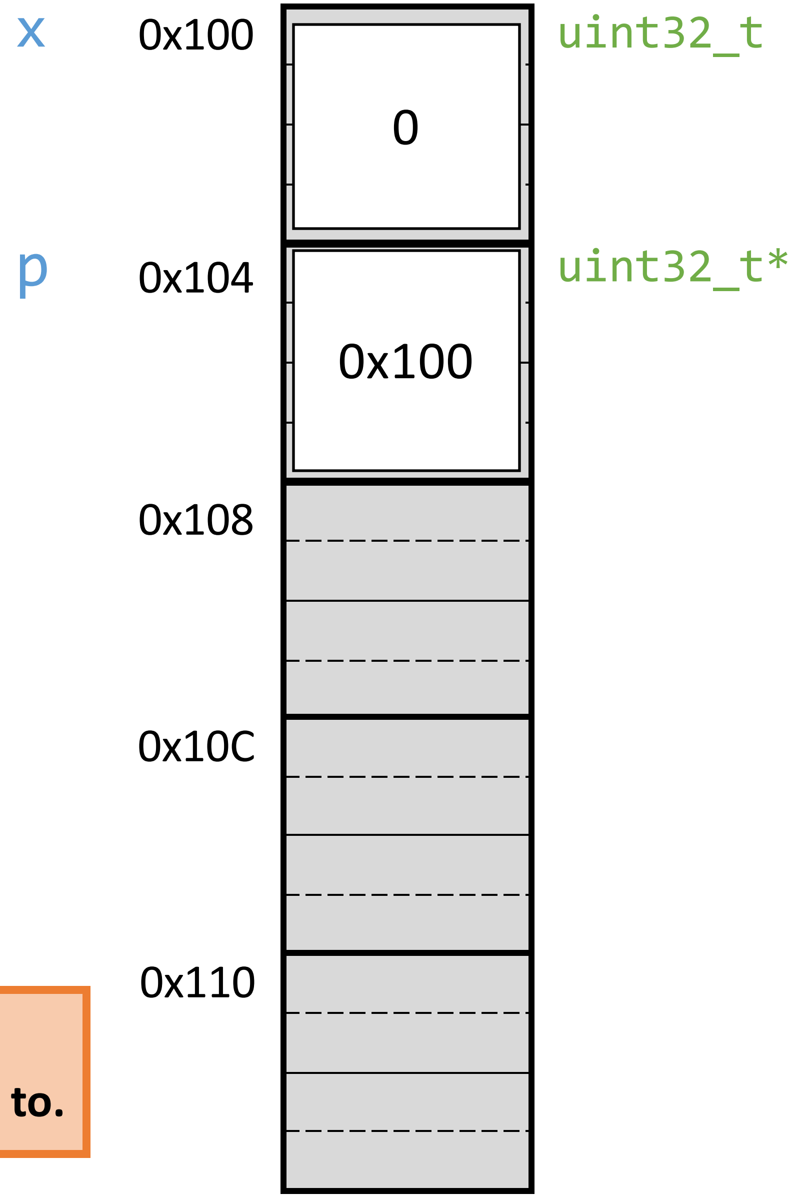
```
uint32_t x = 0;
uint32_t* p;

p = &x;

*p = 13;
printf("%d\n", *p);
```

We can ***assign*** the
value this address points to.

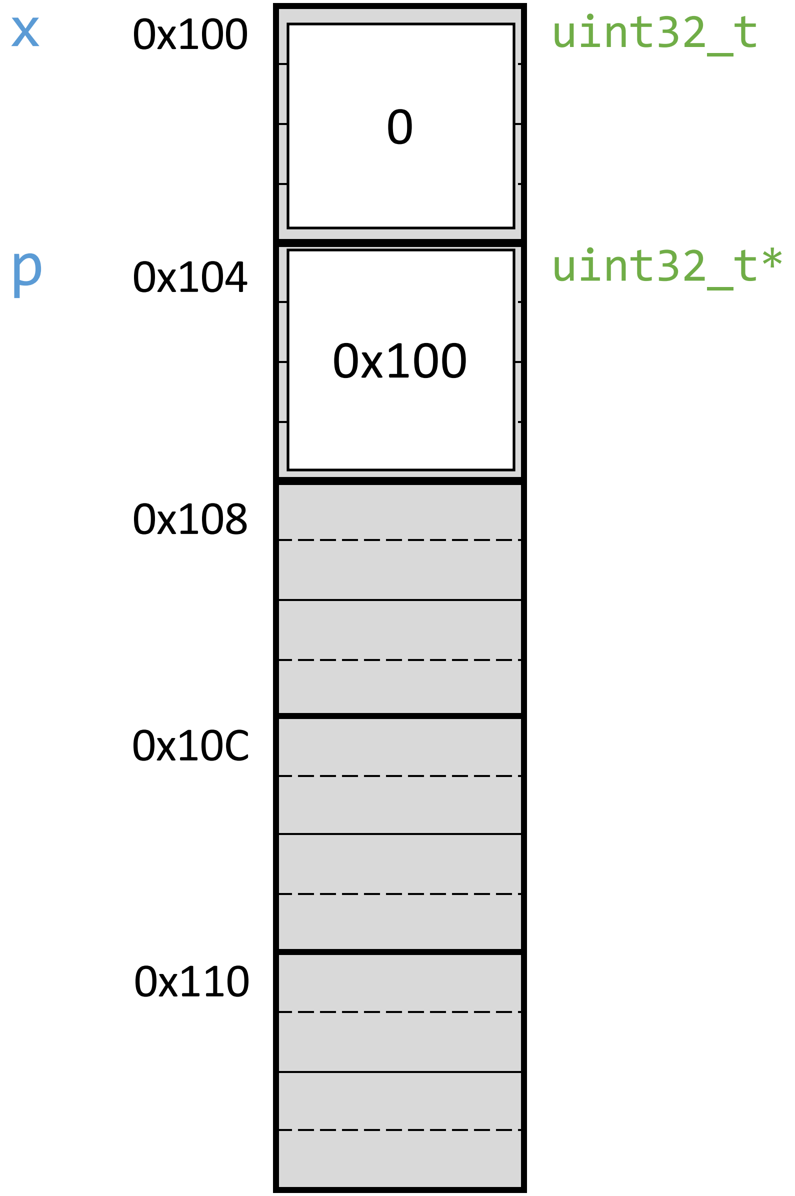
We can ***read*** the
value this address points to.



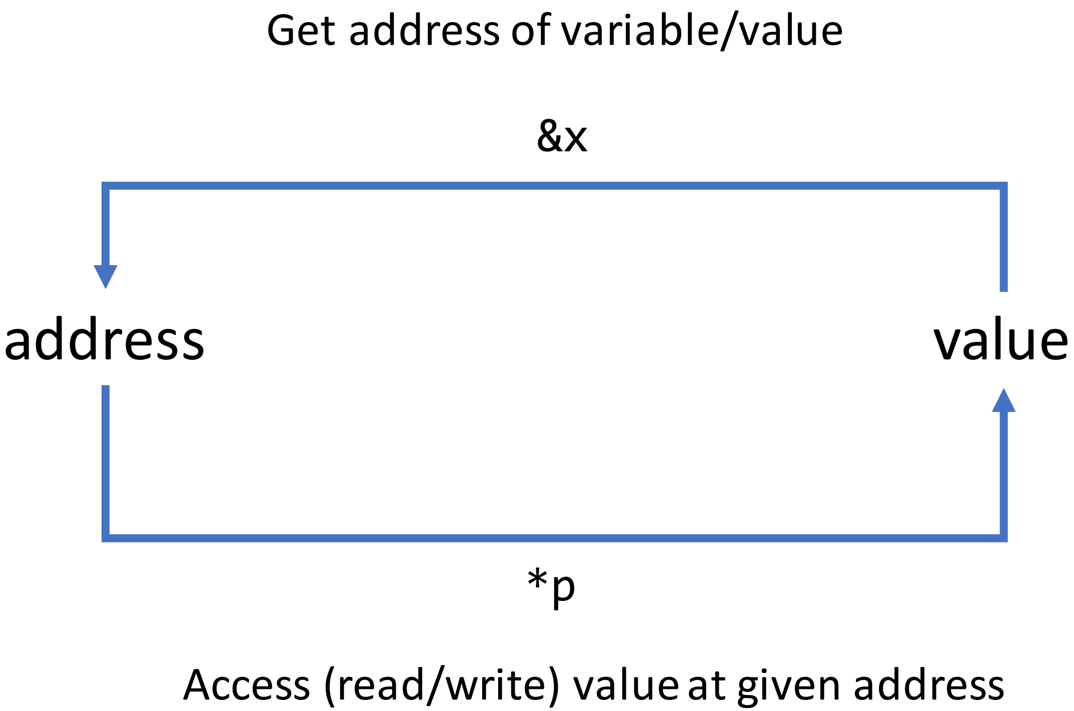
```
uint32_t x = 0;
uint32_t* p;

p = &x;

*p = 13;
printf("%x\n", x);
printf("%x\n", &x);
printf("%x\n", p);
printf("%x\n", *p);
printf("%x\n", &p);
```



& and * are opposites



Get address of variable/value

&x

address

value

*p

Access (read/write) value at given address

```
int x;


   x  = 7;
*(&x) = 7;
```

These do the same thing

Rule: Pointers should point to something valid or be NULL

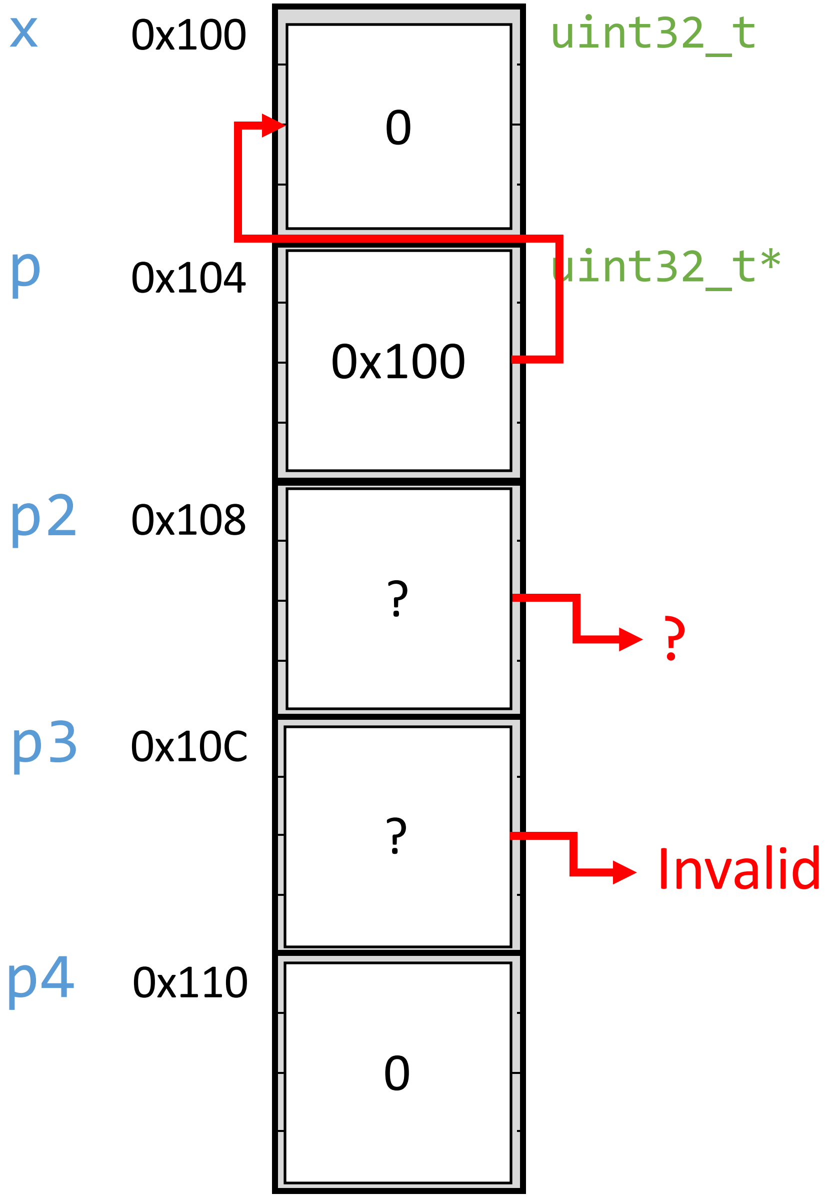
```
uint32_t x = 0;
uint32_t* p;

p = &x;
*p = 13;

uint16_t* p2;
*p2 = 13;

uint8_t* p3 = 10;
*p3 = 13;

uint32_t p4 = NULL;
```



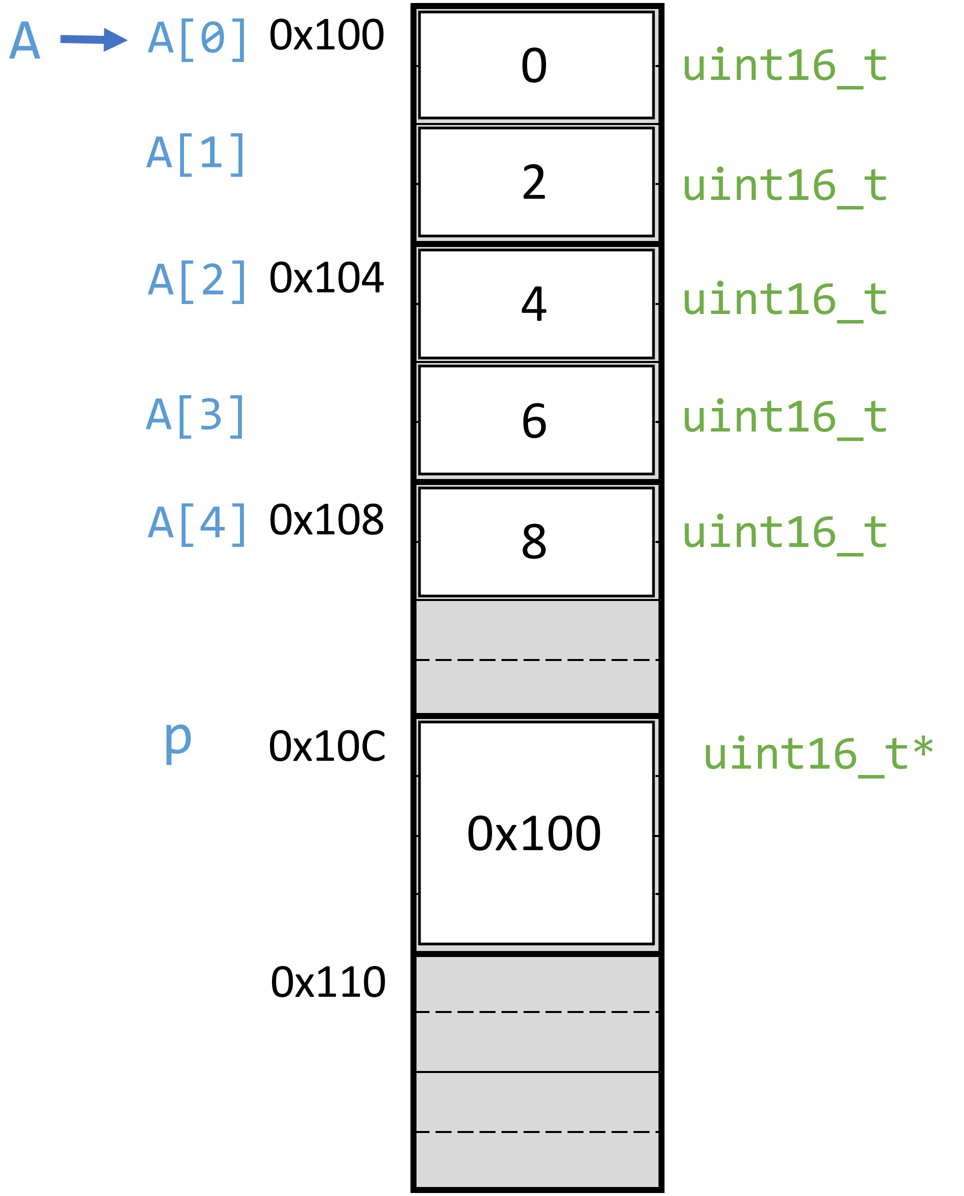
Pointers to Arrays

```
uint16_t A[5]={0,2,4,6,8};
uint16_t* p;
p = A;

A++;

p++;
p = p + 2;
(*p)++;

printf("%x\n", p);
printf("%x\n", *p);
printf("%x\n", A[3]);
printf("%x\n", p[1]);
```



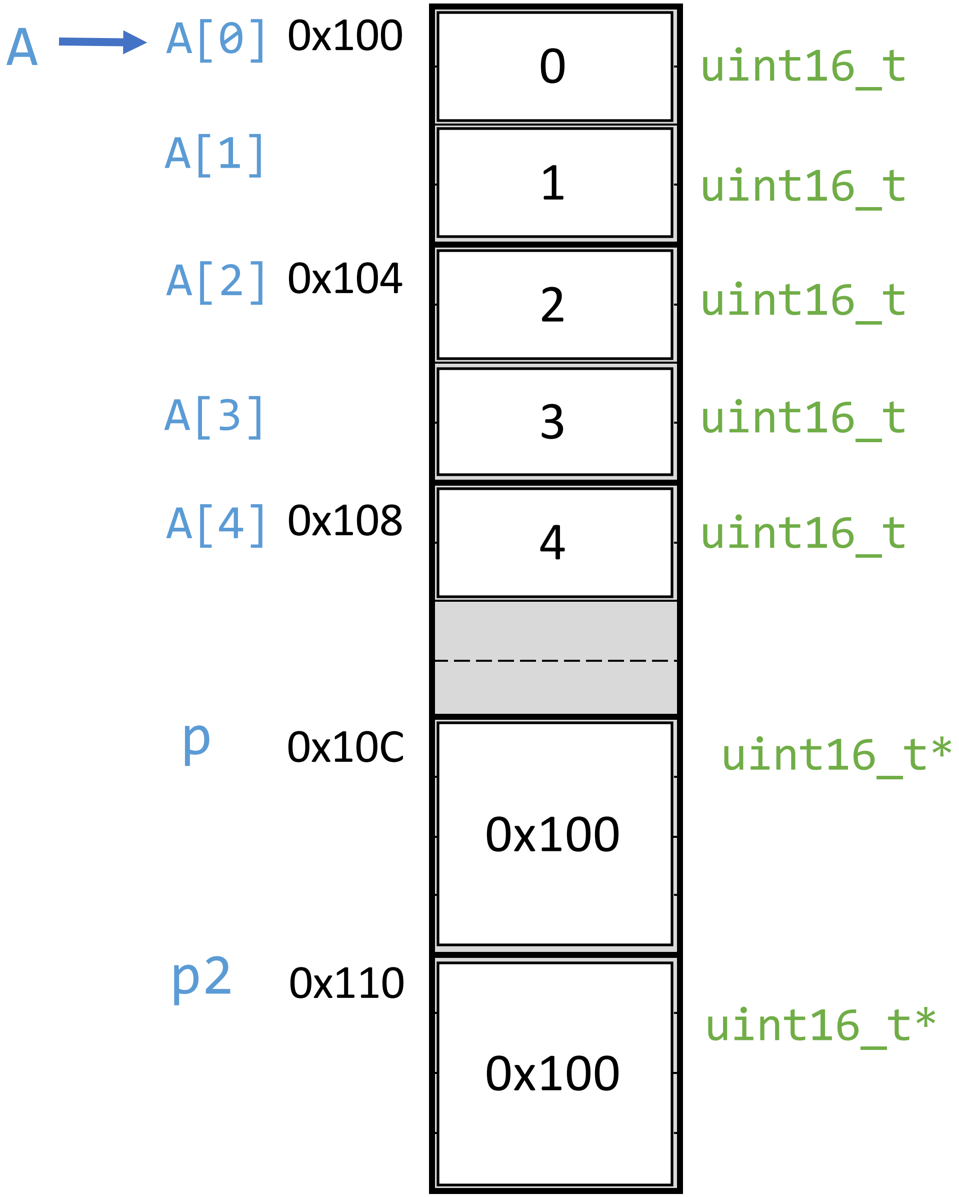
Pointers to Arrays

```
uint16_t A[5]={0,1,2,3,4};
uint16_t* p;
uint16_t* p2;

p = A;
p2 = &(A[0]);

(*p)++;
p2++;
(*(p+2))++;

printf("%d\n", p[2]    );
printf("%d\n", *(p+2)  );
printf("%d\n", *(p2+1) );
```



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)