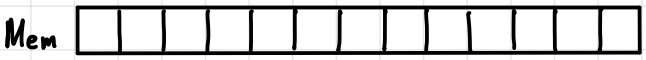
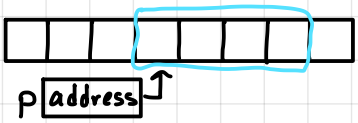
C Memory Model



- each byte has an address
- A pointer is a variable holding an address
- *p treats memory starting at p as a variable
  - size depends on type

```
int *p   //typically 4 bytes (machine dependent)
```




```
int x = *p
*p = x
```

- #include <stdint.h>

Uint8_T, Uint16_T, Uint32_T, Uint64_T

Unsigned

bits

```
Uint16_T *p

Unit x = *p
```

int8_T... int64_T

signed

- Use int for general integer
- Use Uint for data manipulation
  - DON'T USE SIGNED, OR BUGS OCCUR

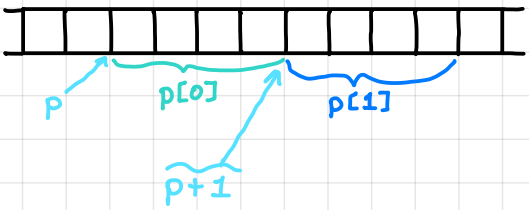
- Raw pointers: void *

```
int foo(void *p)
```

- p is a pointer with no associated type
- Think of p as pointer to buffer
- Can be assigned to/from any pointer type

- Pointers can be treated like arrays

```
Uint32_T *p
  p[0]
  p[1]
```



- pointer arithmetic

p + x evaluates to the address of P(x)

Sometimes: p = p+1

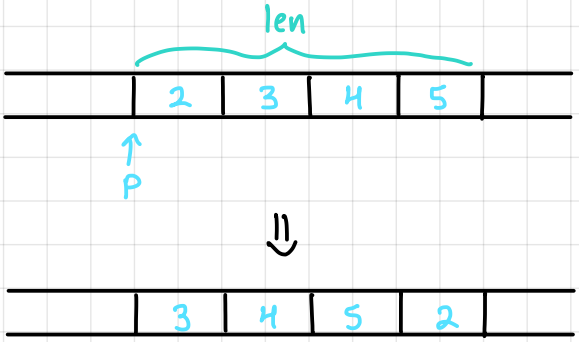
* Use to traverse the buffer

p_orig = p

Loop: p=p+1

Examples:

Circular Shift:



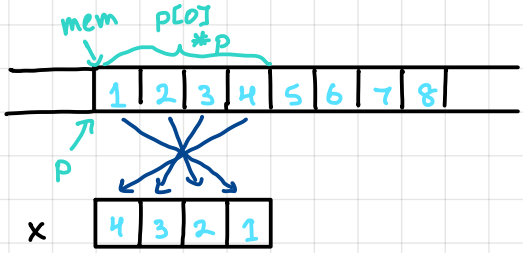
Method:

1) Save first byte
2) Move remaining bytes to left (loop)
3) Store saved first byte to end

Endianess

Intel reads are "little endian"

```
uint8_t mem[0] = {1, 2, 3, 4, 5, 6, 7, 8};
uint32_t *p = (uint32_t *) mem;
uint32_t x = *p
```



x = 4321

* Note: Big Endian machine: No Reverse of data.

Little Endian machine: data is reversed.