Advanced Pointer Use

Embedded Software Essentials

C2 M2 V5

Advanced Pointers [S2]

- Memories of an Embedded System
  - Generic Pointer (void)
  - Double Pointer
  - Restrict Pointer

```
void * ptr1 = NULL;
void ** ptr2  = &ptr1;
uint32_t * restrict ptr3;
uint32_t ** ptr4;
```

```
sizeof( uint8_t* ) = sizeof( void* )
                   = sizeof( void** )
                   = sizeof( uint32_t** )
                   = sizeof( uint32_t* restrict )
                   = 32-Bits![1]
```

```
sizeof( ptr1 ) = sizeof( ptr2 )
               = sizeof( ptr3 )
               = sizeof( ptr4 )
               = 32-Bits! [1]
```

[1] On our 32-bit ARM Architecture

Void Pointer [S3a]

- Void pointers are Generic Pointers, they point to a memory address
  - void = Lack of type, dereferencing does not make sense!

[1] On our 32-bit ARM Architecture

Void Pointer [S3b]

- Void pointers are Generic Pointers, they point to a memory address
  - void = Lack of type, dereferencing does not make sense!

```
sizeof( void* ) = sizeof( uint8_t* )
                = sizeof( float* )
                = sizeof( uint32_t* )
                = 32-Bits![1]
```

Void Pointers are NOT NULL Pointers, but a NULL Pointer is a Void Pointer:

```
#define NULL (void*(0))
void * ptr1 = NULL;
```

[1] On our 32-bit ARM Architecture

Void Pointer [S3c]

- Void pointers are Generic Pointers, they point to a memory address
  - void = Lack of type, dereferencing does not make sense!

```
sizeof( void* ) = sizeof( uint8_t* )
                = sizeof( float* )
                = sizeof( uint32_t* )
                = 32-Bits![1]
```

Void Pointers are NOT NULL Pointers, but a NULL Pointer is a Void Pointer:

```
#define NULL (void*(0))
void * ptr1 = NULL;
```

- Must cast before using
- No dereferencing on a void *
- No pointer arithmetic on a void *

```
void * ptr1 = (void*)0x40000000;
*((uint16_t*)ptr1) = 0x0202;
```

Equivalent to:

```
TA0CTL = 0x0202;
```

[1] On our 32-bit ARM Architecture

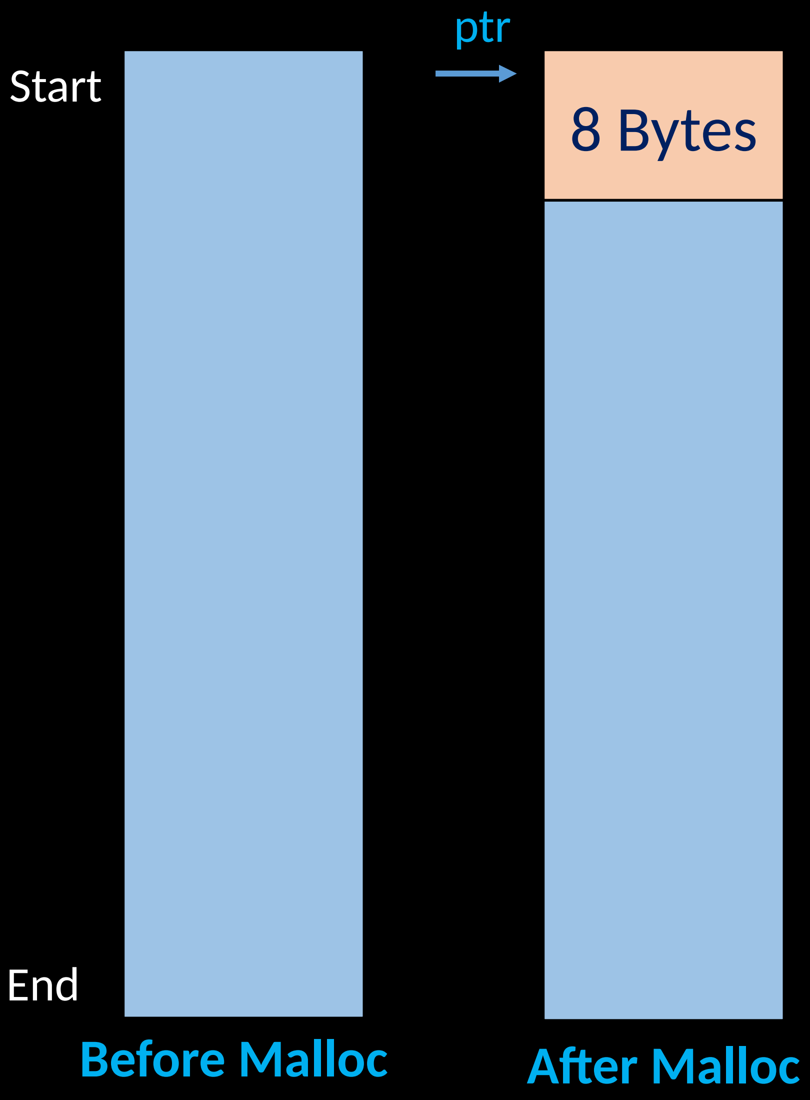
Malloc and Void *[S4]

- Malloc reserves blocks of data, it does not care how it is used
  - Returns a void pointer, you cast this pointer for the intended use

```
char * ptr;
ptr = (char *)malloc(8*sizeof(char));

if (ptr == NULL) {
  /* Allocation Failed!!! */
  /* …Handle Failure */
}
 /* Other Code */
free((void *)ptr);
```

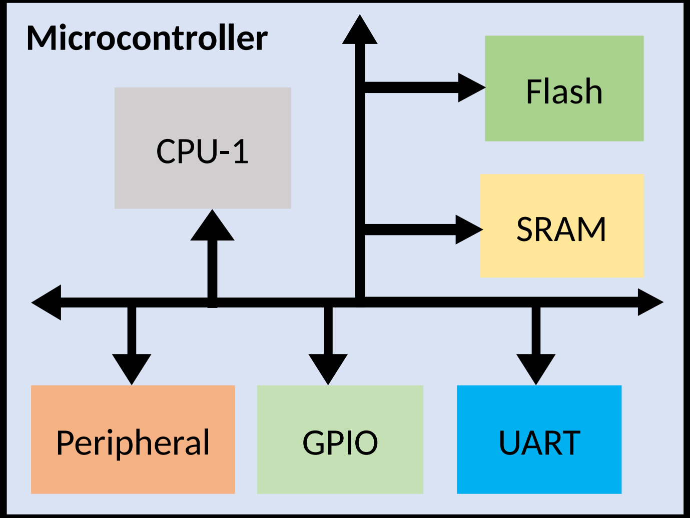
Heap States



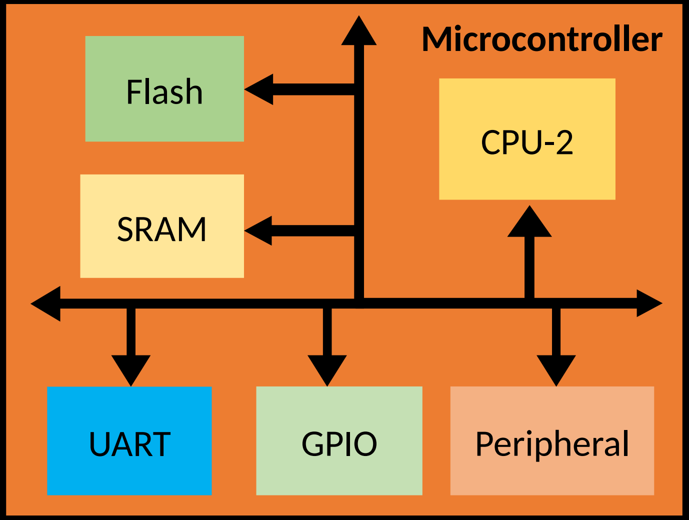
Void Pointer Example [S5a]

- You might not know the underlying type without some processing
  - Sequence of bytes being sent, first byte is type indicator

Platform 1

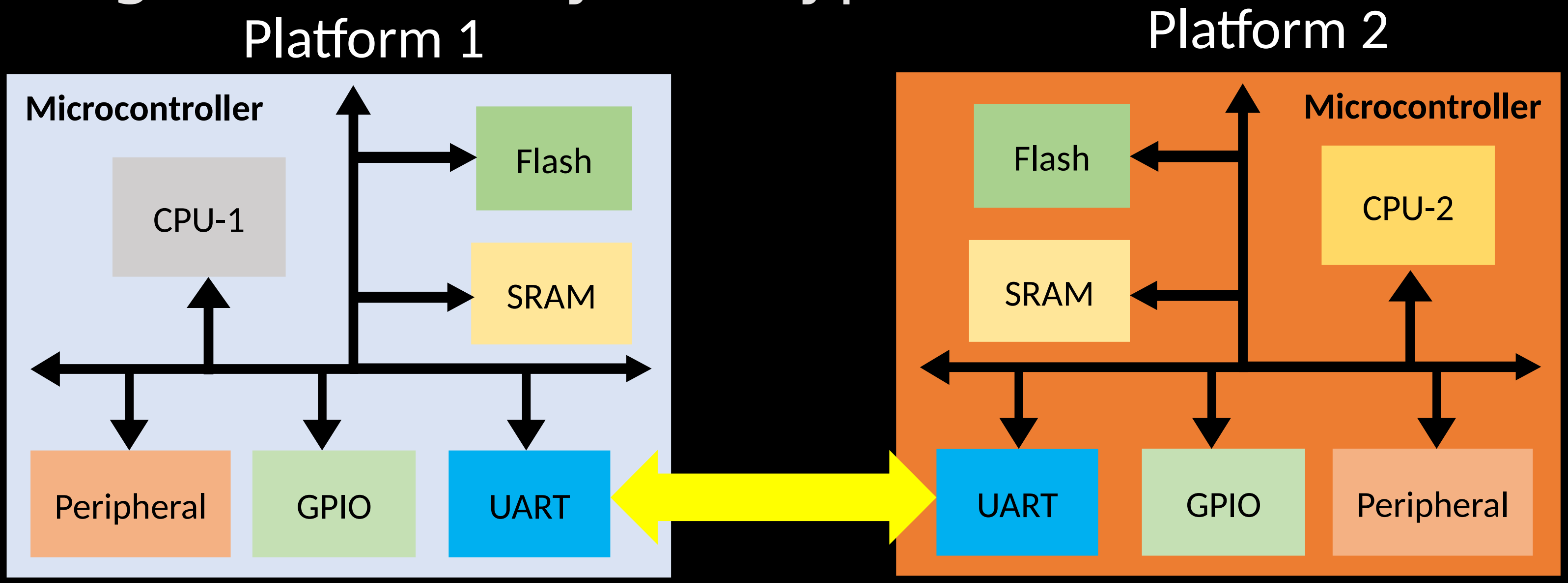


Platform 2



Void Pointer Example [S5b]

- You might not know the underlying type without some processing
  - Sequence of bytes being sent, first byte is type indicator



Two embedded systems sending command and responses to each other

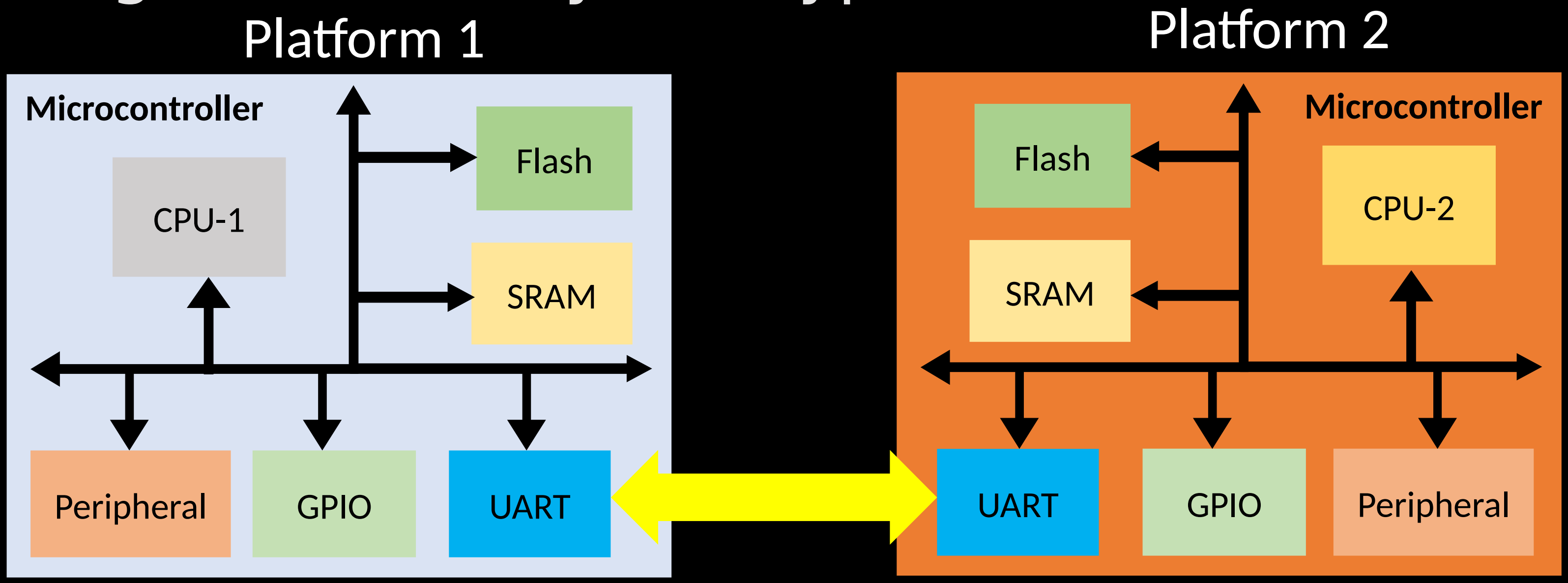
Void Pointer Example [S5c]

- You might not know the underlying type without some processing
  - Sequence of bytes being sent, first byte is type indicator

```
typedef enum {
  RSP_TYPE_1 = 0,
  RSP_TYPE_2 = 1,
} RSP_e;

typedef struct {
   RSP_e rsp_type;
   uint8_t data[4];
} rsp1;

typedef struct {
   RSP_e rsp_type;
   uint32_t data;
} rsp2;
```



Two embedded systems sending command and responses to each other

Void Pointer Example [S5d]

- You might not know the underlying type without some processing
  - Sequence of bytes being sent, first byte is type indicator

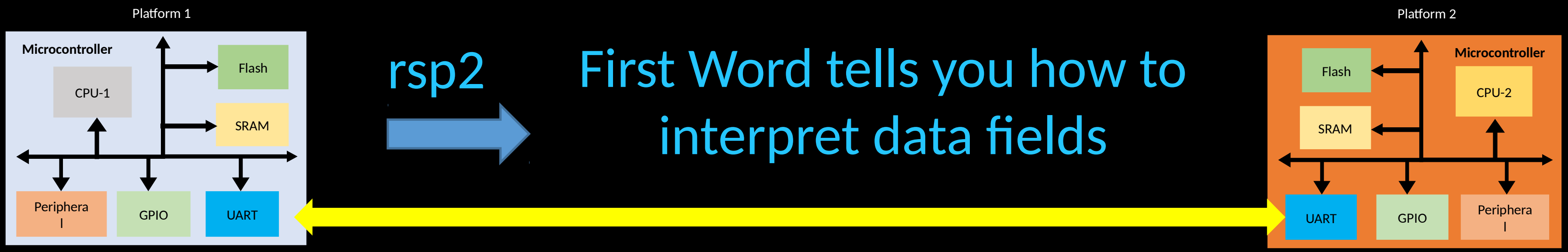
```
typedef enum {
  RSP_TYPE_1 = 0,
  RSP_TYPE_2 = 1,
} RSP_e;

typedef struct {
   RSP_e rsp_type;
   uint8_t data[4];
} rsp1;

typedef struct {
   RSP_e rsp_type;
   uint32_t data;
} rsp2;
```

Assume Packed: sizeof(rsp1) = sizeof(rsp2) = 8 Bytes to transmit

rsp1	Data[3]	Data[2]	Data[1]	Data[0]	RSP_e = RSP_TYPE_1
rsp2	Data				RSP_e = RSP_TYPE_2



rsp2

First Word tells you how to interpret data fields

Double Pointer [S6a]

- Double pointers are a pointer to a pointer
- Must use the ** in declarations

```
sizeof( float** ) = sizeof( uint8_t** )
                  = sizeof( void** )
                  = sizeof( uint32_t** )
                  = 32-Bits![1]
```

```
uint32_t var = 0x1234ABCD;
uint32_t * ptr3 = &var;
uint32_t ** ptr4 = &ptr3;
```

[1] On our 32-bit ARM Architecture

Double Pointer [S6b]

- Double pointers are a pointer to a pointer
- Must use the ** in declarations

sizeof(float**) = sizeof(uint8_t**)
= sizeof(void**)
= sizeof(uint32_t**)
= 32-Bits![1]

- Used to set value of a pointer (address)
  - Single dereference accesses pointer address
  - Double dereference accesses pointer

```
uint32_t var = 0x1234ABCD;
uint32_t * ptr3 = &var;
uint32_t ** ptr4 = &ptr3;
```

[1] On our 32-bit ARM Architecture

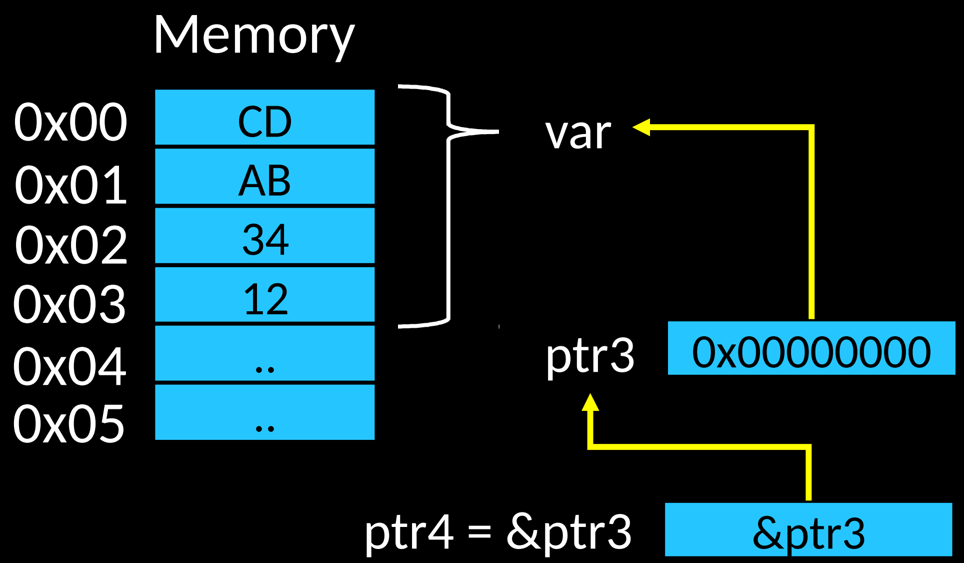
Double Pointer [S6c]

- Double pointers are a pointer to a pointer
- Must use the ** in declarations

```
sizeof( float** ) = sizeof( uint8_t** )
                  = sizeof( void** )
                  = sizeof( uint32_t** )
                  = 32-Bits!¹
```

- Used to set value of a pointer (address)
  - Single dereference accesses pointer address
  - Double dereference accesses pointer

```
uint32_t var = 0x1234ABCD;
uint32_t * ptr3 = &var;
uint32_t ** ptr4 = &ptr3;
```



[1] On our 32-bit ARM Architecture

Double Pointer Example [S7]

- Copies of pointers are made when passed into a function
  - Original pointer address cannot be altered!

```
typedef enum {
  RSP_TYPE_1 = 0,
  RSP_TYPE_2 = 1,
} RSP_e;

typedef struct {
   RSP_e rsp_type;
   uint8_t data[4];
} rsp1;
```

```
int8_t create_rsp1 (rsp1 ** r_p){
  *r_p = (rsp1 *)malloc(sizeof(rsp1));

  if (*r_p == NULL) {
    /* Allocation Failed!!! */
    return -1;
  }
  (*r_p)->rsp_type = RSP_TYPE_1;
  return 0;
}
```

Restrict Qualified Pointer [S8a]

- Restrict type qualifier helps compiler to optimize memory interactions
  - Must use the restrict qualifier AFTER the * in declarations

```
uint32_t * restrict ptr4;
```

```
sizeof( float* ) = sizeof( uint8_t* )
                 = sizeof( void* )
                 = sizeof( uint32_t* restrict )
                 = 32-Bits![1]
```

- Introduced in C99 Standard

[1] On our 32-bit ARM Architecture

Restrict Qualified Pointer [S8b]

- Restrict type qualifier helps compiler to optimize memory interactions
  - Must use the restrict qualifier AFTER the * in declarations

```
uint32_t * restrict ptr4;
```

- Only the data at this location or data near is accessed by this pointer

- Largest speedup comes from iterative memory interaction
  - Compiler removes unneeded assembly instructions
  - Couple assembly instructions per loop

[1] On our 32-bit ARM Architecture