Data Structures (8A)

Young Won Lim
7/3/20

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice.

Based on

ARM System-on-Chip Architecture, 2nd ed, Steve Furber

Introduction to ARM Cortex-M Microcontrollers
– Embedded Systems, Jonathan W. Valvano

Digital Design and Computer Architecture,
D. M. Harris and S. L. Harris

Computer Organization and Design ARM Edition:
The Hardware Software Interface
By David A. Patterson, John L. Hennessy

https://thinkingeek.com/arm-assembler-raspberry-pi/

Heap

1. dynamically allocated by the program
    when it creates the data structure
2. used by the program to store information
3. dynamically released by the program
    when the structure is no longer needed

Heap manager

pt = **malloc**(size);   // returns a pointer to a block of size bytes
free(pt);   // deallocates the block at pt

simple version to be considered

pt = **Heap_Allocate**();   // returns a pointer to a block of fixed size bytes
Heap_Release(pt);   // deallocates the block at pt

Assembly Programming (8A)
Data Structures

Young Won Lim
7/3/20

Heap

```
#define    SIZE      4
#define    NUM       5
#define    NULL      0      // empty pointer

int32_t    *FreePt;
int32_t    Heap[SIZE*NUM];
```
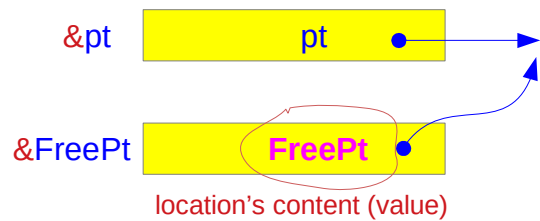
```
SIZE      EQU      4
NUM       EQU      5
NULL      EQU      0
FreePT    SPACE    4
Heap      SPACE    SIZE*NUM*4
```
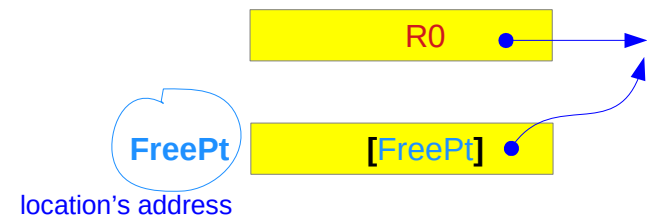
Pointer and array declarations – C and ARM assembly

C code

```
int32_t *  pt;
int32_t *  FreePt;
int32_t    Heap[SIZE*NUM];
```

ARM assembly code

```
FreePT    SPACE   4
Heap      SPACE   SIZE*NUM*4
```

&pt | pt |

&FreePt | **FreePt** |

location's content (value)

Heap	Heap[0]
Heap+1	Heap[1]
Heap+2	Heap[2]
Heap+3	Heap[3]
Heap+4	Heap[4]
Heap+5	Heap[5]

| | R0 |

FreePt | [FreePt] |

location's address

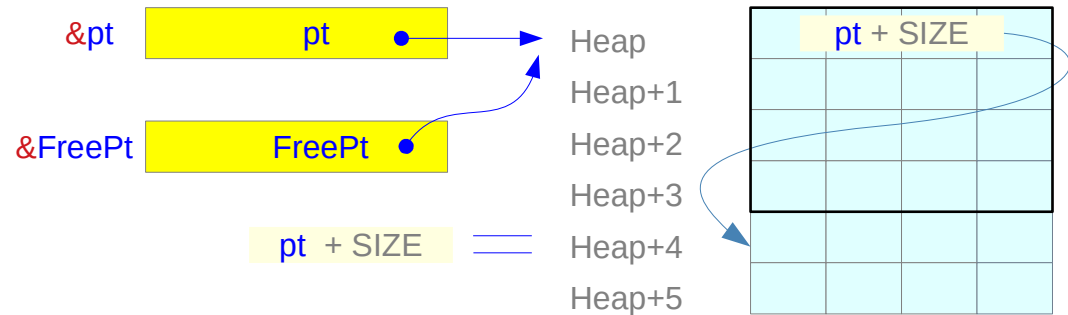| **Heap** | |
Heap+1	
Heap+2	
Heap+3	
Heap+4	
Heap+5	

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

Heap_Init (1)

```
void Heap_Init(void) {
      int          i
      int32_t *   pt;

      pt = FreePt = &Heap[0];

      for (i=1; i<NUM; i++) {
            *pt = (int32_t) (pt +SIZE);
            pt = pt+SIZE;
      }

      * (int32_t *) pt = NULL;
}
```

```
#define      SIZE  4
#define      NUM  5
#define      NULL  0      // empty pointer

int32_t       *FreePt;
int32_t       Heap[SIZE*NUM];
```

&pt   pt   •   → Heap
                            Heap+1
&FreePt   FreePt •   Heap+2
                            Heap+3
      pt + SIZE ──   Heap+4
                            Heap+5

pt + SIZE

```
int32_t *       pt;
int32_t *       FreePt;
```

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

Heap_Init (2)

&pt | pt
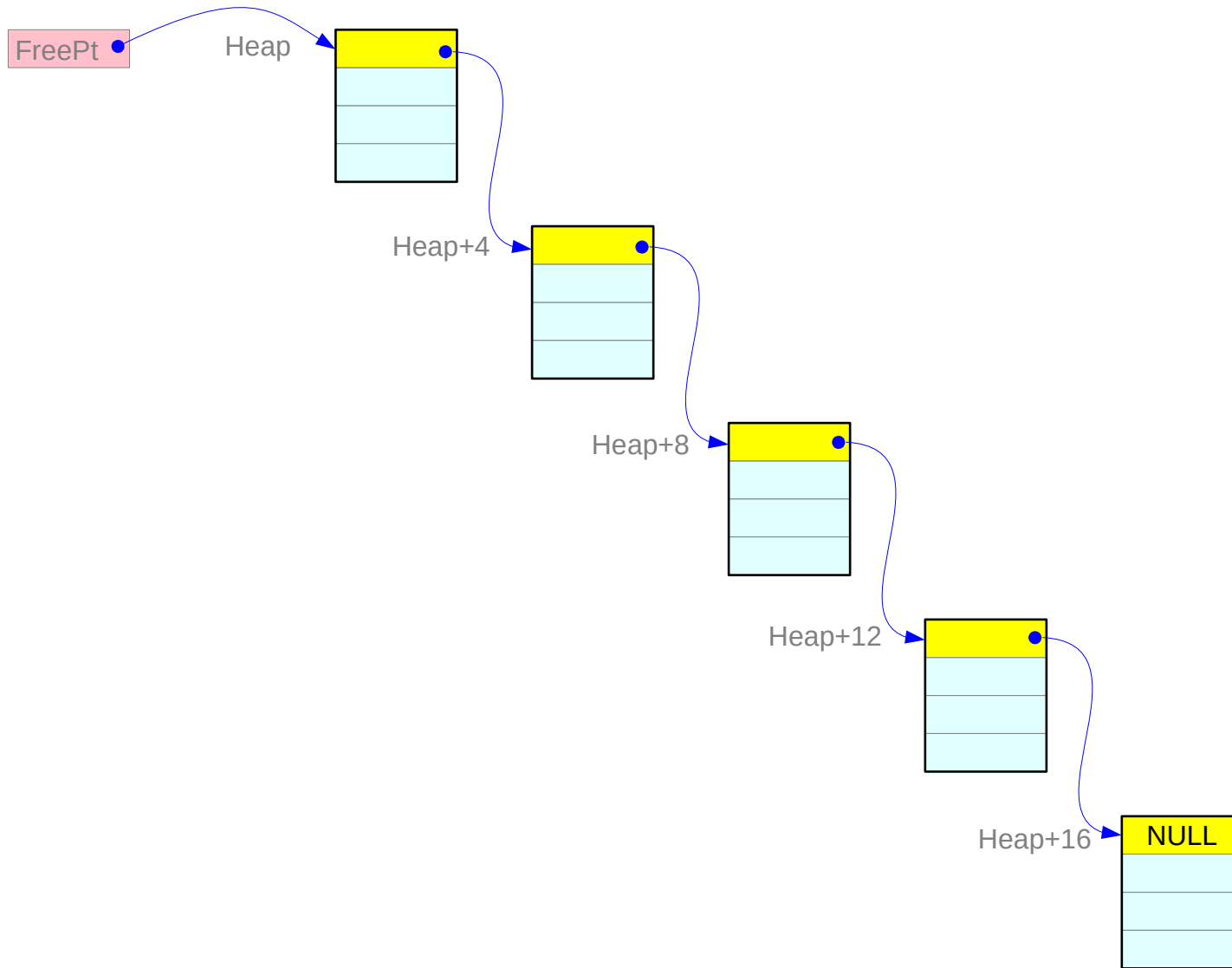&FreePt | FreePt

Heap
Heap+1
Heap+2
Heap+3
Heap+4
Heap+5
Heap+6
Heap+7
Heap+8
Heap+9
Heap+10
Heap+11
Heap+12
Heap+13
Heap+14
Heap+15
Heap+16
Heap+17
Heap+18
Heap+19

pt = pt+SIZE;

Heap+4
Heap+8
Heap+12
Heap+16
NULL

*pt = (int32_t) (pt +SIZE);

*pt = (int32_t) (pt +SIZE);

*pt = (int32_t) (pt +SIZE);

*pt = (int32_t) (pt +SIZE);

* (int32_t *) pt = NULL

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

**Assembly Programming (8A)
Data Structures**

9

Heap_Init (3)

FreePt ● → Heap

●

Heap+4

●

Heap+8

●

Heap+12

●

Heap+16 →

NULL

**Assembly Programming (8A)
Data Structures**

10
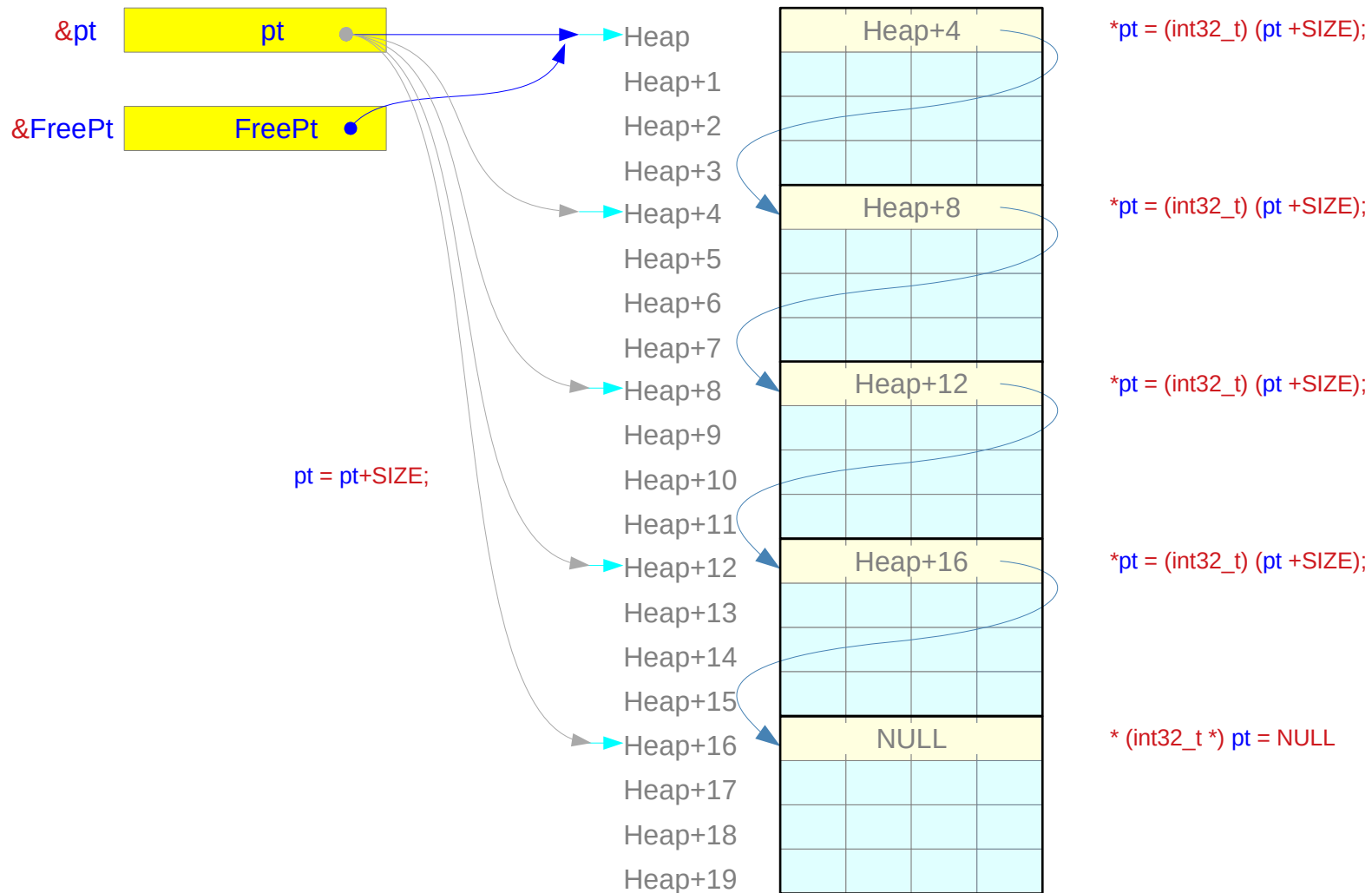
Heap_Init (1)

```
Heap_Init    LDR     R0, =Heap              ; pt              ; R0 ← &Heap
             LDR     R1, =FreePt            ;                 ; R1 ← FreePt
             STR     R0, [R1]               ; *FreePT = Heap  ; [FreePt] ← Heap
             MOV     R2, #SIZE                               ; R2 ← #4
             MOV     R3, #NUM-1                              ; R3 ← #3
imLoop       ADD     R1, R0, R2, LSL #2     ;  pt + SIZE      ; R1 ← R0 + R2*4
             STR     R1, [R0]               ; *pt = pt + SIZE ; [R0] ← R0 + R2*4
             MOV     R0, R1                 ;  pt = pt + SIZE ; R0 ← R1
             SUBS    R3, R3, #1                              ; R3 ← R3 - 1
             BNE     imLoop
             MOV     R1, #NULL                              ; R1 ← #0
             STR     R1, [R0]               ; last ptr is NULL ; [R0] ← NULL
             BX      LR
```

Heap_Init (2)
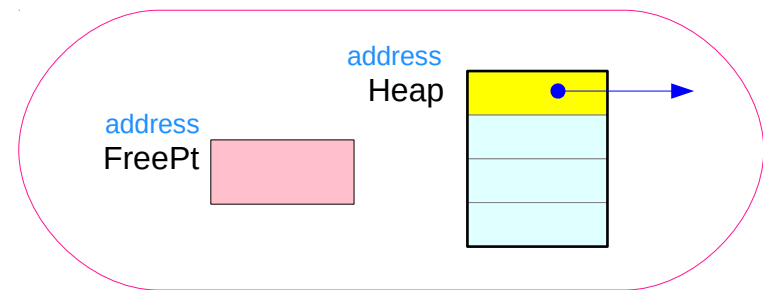
```
Heap_Init    LDR    R0, =Heap        ; pt             ; R0 ← Heap
             LDR    R1, =FreePt      ;                ; R1 ← FreePt
             STR    R0, [R1]         ; *FreePT = Heap ; [FreePt] ← Heap
```
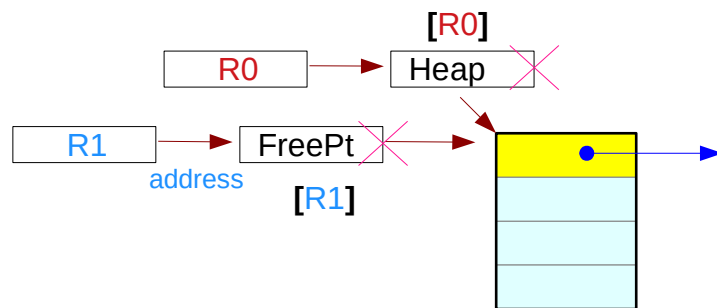
```
LDR  R0, =Heap      ; R0 ← &Heap         [R0]  ≡  Heap
LDR  R1, =FreePt    ; R1 ← &FreePt        [R1]  ≡  FreePt
```

▼ Heap, FreePt : all address labels
of memory locations

Heap_Init (3)

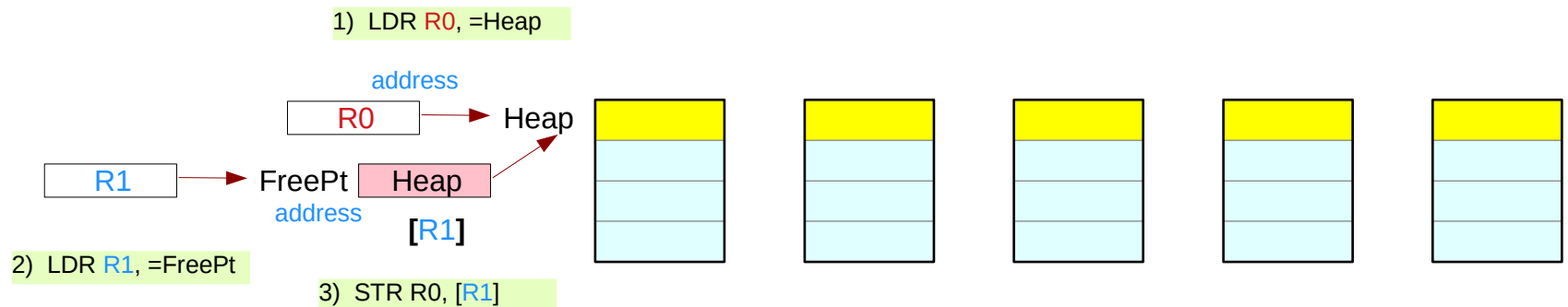Heap_Init   LDR   R0, =Heap   ; pt   ; R0 ← Heap
                 LDR   R1, =FreePt   ;   ; R1 ← FreePt
                 STR   R0, [R1]   ; *FreePT = Heap   ; [FreePt] ← Heap

LDR R0, =Heap   ; R0 ← Heap
LDR R1, =FreePt   ; R1 ← FreePt

R0 ≡   Heap   address
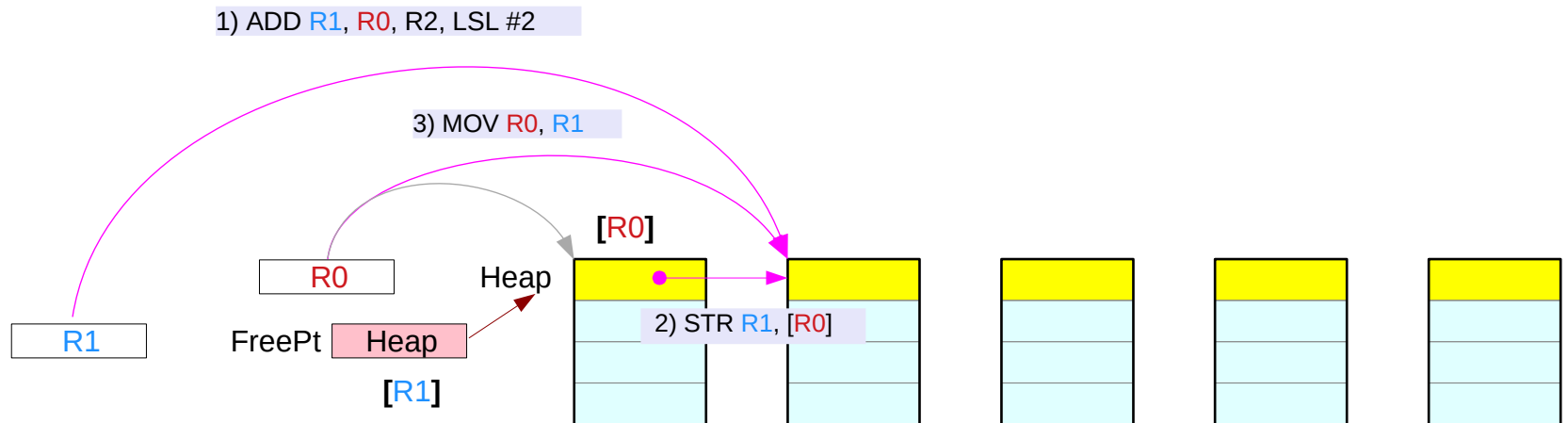R1 ≡   FreePt
              address

1) LDR R0, =Heap

2) LDR R1, =FreePt

3) STR R0, [R1]

Heap_Init (4)

```
              MOV     R2, #SIZE                                    ; R2 ← #4
              MOV     R3, #NUM-1                                   ; R3 ← #3
imLoop        ADD     R1, R0, R2, LSL #2    ;  pt + SIZE          ; R1 ← R0 + R2*4
              STR     R1, [R0]              ; *pt = pt + SIZE     ; [R0] ← R0 + R2*4
              MOV     R0, R1                ;  pt = pt + SIZE     ; R0 ← R0 + R2*4
```
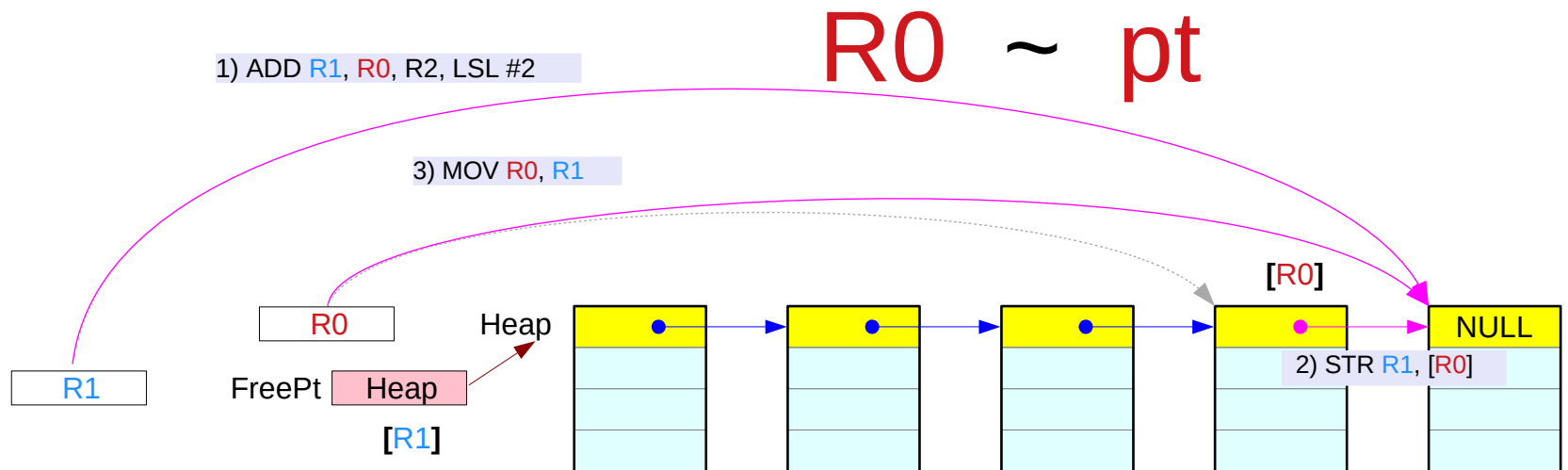
R0 ~ pt

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

Heap_Init (5)

imLoop	ADD	R1, R0, R2, LSL #2	; pt + SIZE	; R1 ← R0 + R2*4
	STR	R1, [R0]	; *pt = pt + SIZE	; [R0] ← R0 + R2*4
	MOV	R0, R1	; pt = pt + SIZE	; R0 ← R0 + R2*4
	SUBS	R3, R3, #1		; R3 ← R3 - 1
	BNE	imLoop		
	MOV	R1, #NULL		; R1 ← #0
	STR	R1, [R0]	; last ptr is NULL	; [R0] ← NULL

R0 ~ pt

1) ADD R1, R0, R2, LSL #2

3) MOV R0, R1

[R0]

R0

Heap

R1

FreePt   Heap

[R1]

2) STR R1, [R0]

NULL

Heap_Allocate

```
int32_t *Heap_Allocate(void) {
    int32_t *   pt;

    pt = FreePt;
    if (pt != NULL) {
        FreePt = (int32_t *) *pt;
    }
    return(pt);
}
```



```
int32_t        *FreePt;
int32_t        Heap[SIZE*NUM];

void Heap_Init(void) {
    …
    int32_t *      pt;
    …
    pt = FreePt = &Heap[0];
    …
}
```
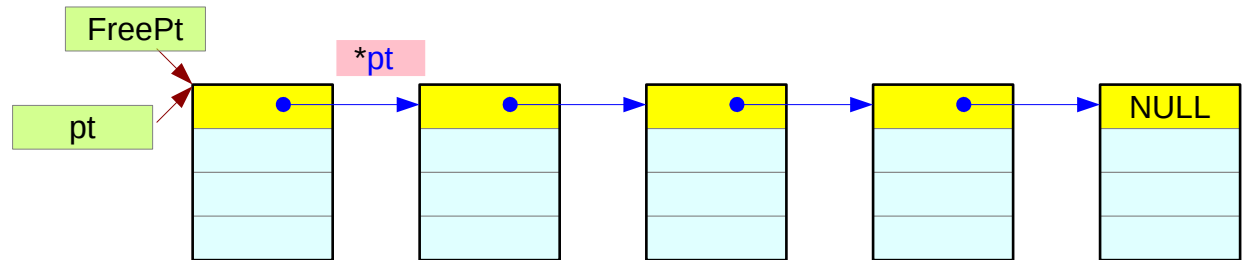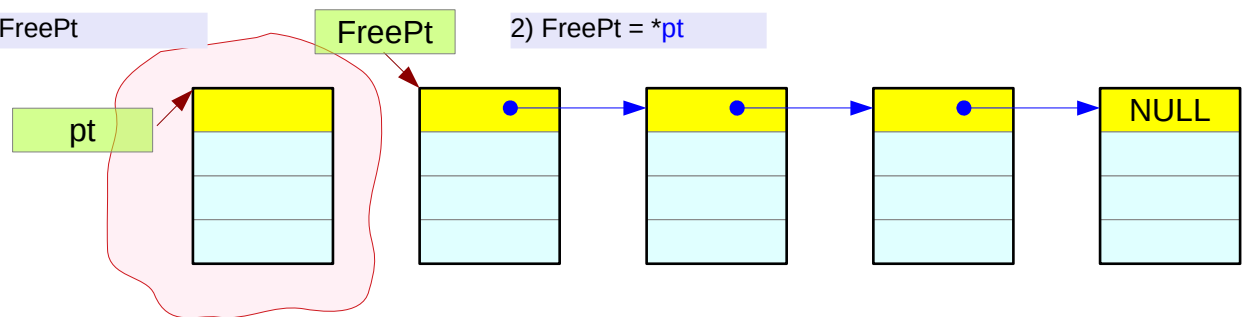
Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

Heap_Allocate

```
Heap_Allocate                              ; R0 points to new block
        LDR     R1, =FreePt                ; R1 = FreePt        ; R1 ← &FreePt
        LDR     R0, [R1]                   ; R0 = [FreePt]       ; R0 ← [R1]
        CMP     R0, #NULL                  ; if (pt != NULL)
        LDR     R2, [R0]                   ; link next           ; R2 ← [R0]
        STR     R2, [R1]                   ; FreePt = *pt        ; [R1] ← R2
Adone   BX      LR
```

pt = Heap_Allocate();
Heap_Release(pt);

3) LDR R2, [R0]
    STR R2, [R1]   Old [R1] : Heap → New [R1] : Heap+4

[R1]   [R0]

R1 → FreePt Heap → Heap
       address   address

1) LDR R1, =FreePt

R0

2) LDR R0, [R1]
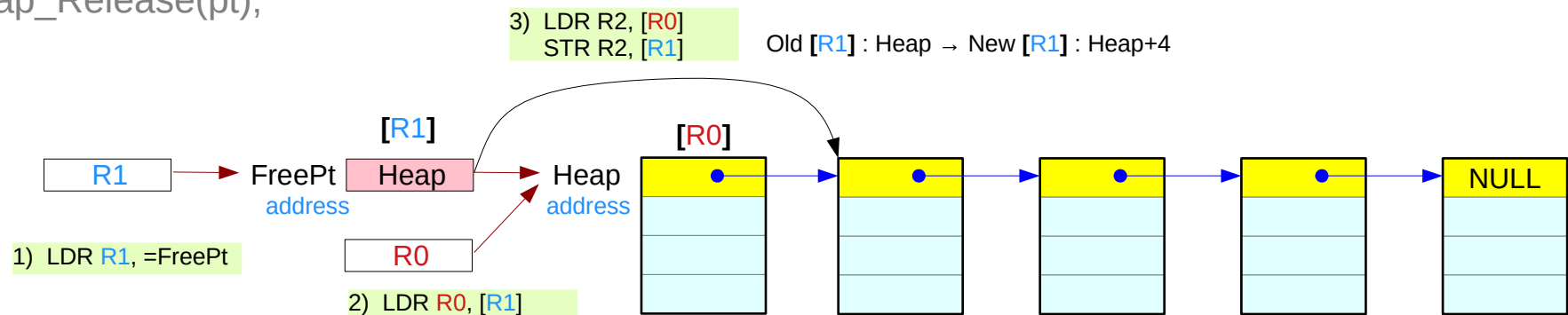
Heap_Release

int32_t *Heap_Release(int32_t *pt) {
    int32_t *   oldFreePt;

    oldFreePt = FreePt;
    FreePt = pt;
    *pt = (int32_t) oldFreePt;
}
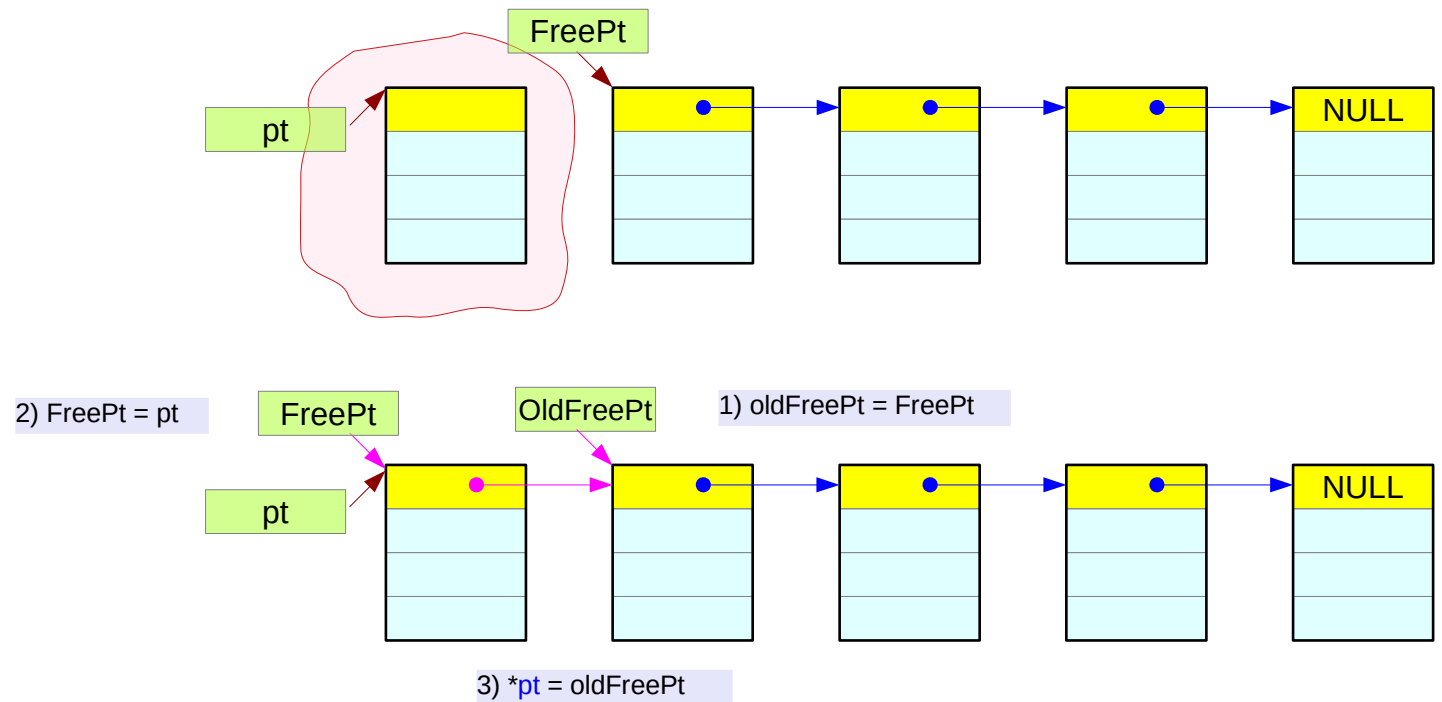
Heap_Release

Heap_Release: ; R0 : block being released pt
     LDR   R1, =FreePt
     LDR   R2, [R1]   ; R2 : oldFreePt
     STR   R0, [R1]   ; FreePt = pt
     STR   R2, [R0]   ; *pt=oldFreePt
     BX   LR

Assembly Programming (8A)
Data Structures

19
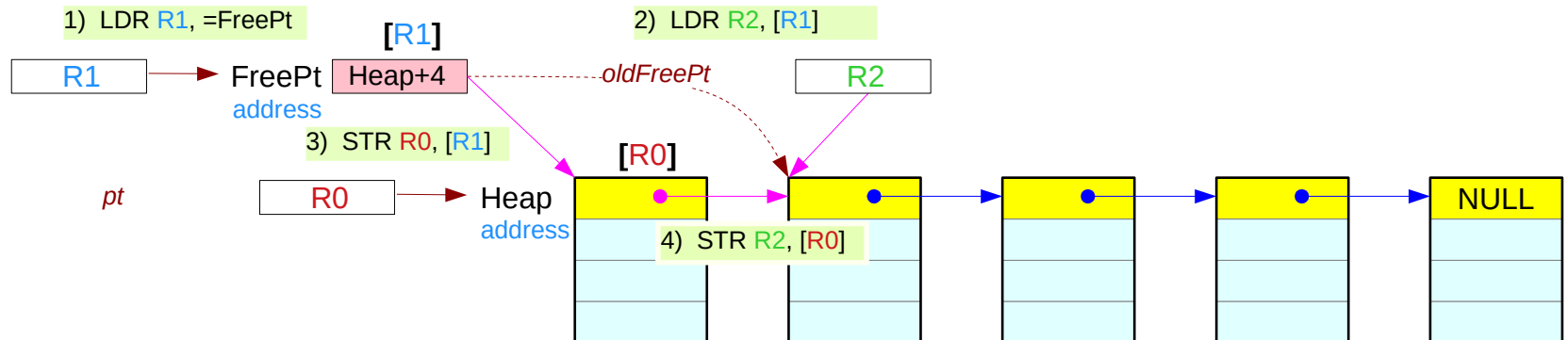
Fifo

```
struct Node {
    struct Node      *Next;
    int32_t          Data;
};
typedef struct Node NodeType;


NodeType      *PutPt;        // place to put
NodeType      *GetPt;        // place to get



; put in RAM
NEXT     EQU      0     ; next
Data     EQU      4     ; 32-bit data for node


GetPt    SPACE    4     ; pointer to oldest node
PutPt    SPACE    4     ; pointer to newest node
```

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

```c
void Fifo_Init(void) {
    GetPt = NULL;                    // Empty when null
    PutPt = NULL;
    Heap_Init();
}
```

GetPt
= NULL

PutPt
= NULL
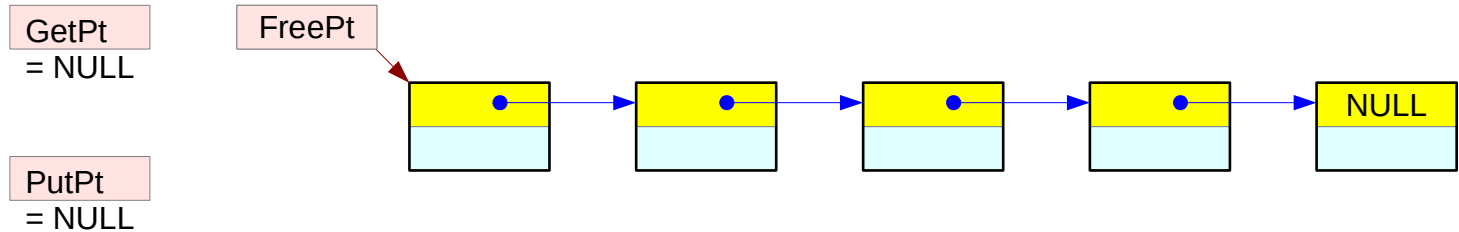
FreePt

NULL

In Heap_Init

```
#define    SIZE  2              SIZE    EQU      2
#define    NUM  5              NUM     EQU      5
#define    NULL  0    // empty pointer    NULL    EQU      0
                               FreePT  SPACE    4
int32_t    *FreePt;            Heap    SPACE    SIZE*NUM*4
int32_t    Heap[SIZE*NUM];
```

```
Fifo_Init:    PUSH    {R4, LR}
              MOV     R1, #NULL;
              LDR     R0, =GetPt                              ; R0 ← GetPt
              STR     R1, [R0]      ; GetPt = NULL    ; [GetPt] ← NULL
              LDR     R0, =PutPt                              ; R0 ← PutPt
              STR     R1, [R0]      ; PutPt = NULL    ; [PutPt] ← NULL
              BL      Heap_Init
              POP     {R4, PC}
```

3) LDR R2, [R0]
   STR R2, [R1]   Old [R1] : Heap → New [R1] : Heap+4

[R1]   [R0]

FreePt Heap → Heap
address   address

GetPt   0
address
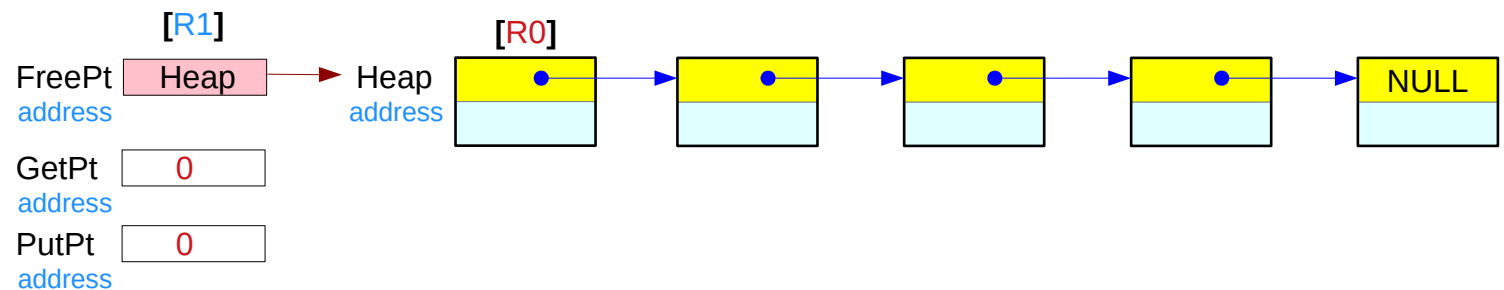
PutPt   0
address

```c
int Fifo_Put(int32_t theData) {
    NodeType *pt;                    // Empty when null

    pt = (nodeType*) Heap_Allocate();
    if (!pt) {
        return(0);                   // full
    }

    pt->Data = theData;              // store
    pt->Next = NULL;

    if (PutPt) {
        PutPt->Next = pt;            // Link
    } else {
        GetPt = pt;                  // first one
    }

    PutPt = pt;
    return(1);                       // successful
}
```

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

Heap_Init()

GetPt
= NULL

PutPt
= NULL

FreePt

NULL

Heap_Allocate()

GetPt

pt

NULL

PutPt

FreePt

NULL

```
int Fifo_Put(int32_t theData) {
    NodeType *pt;
    pt = (nodeType*) Heap_Allocate();
    if (!pt) return(0);
    pt->Data = theData;
    pt->Next = NULL;

    if (PutPt) PutPt->Next = pt;
    else GetPt = pt;
    PutPt = pt;
    return(1);
}
```

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano
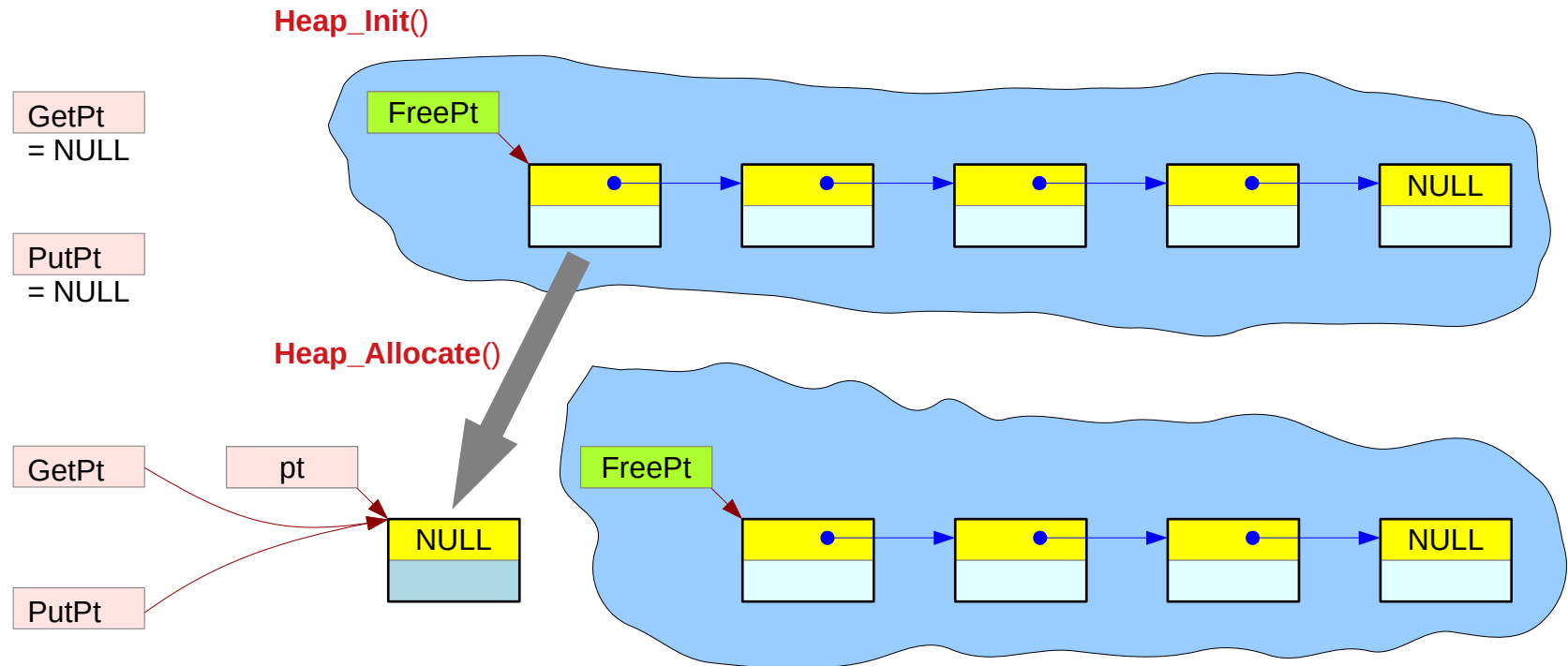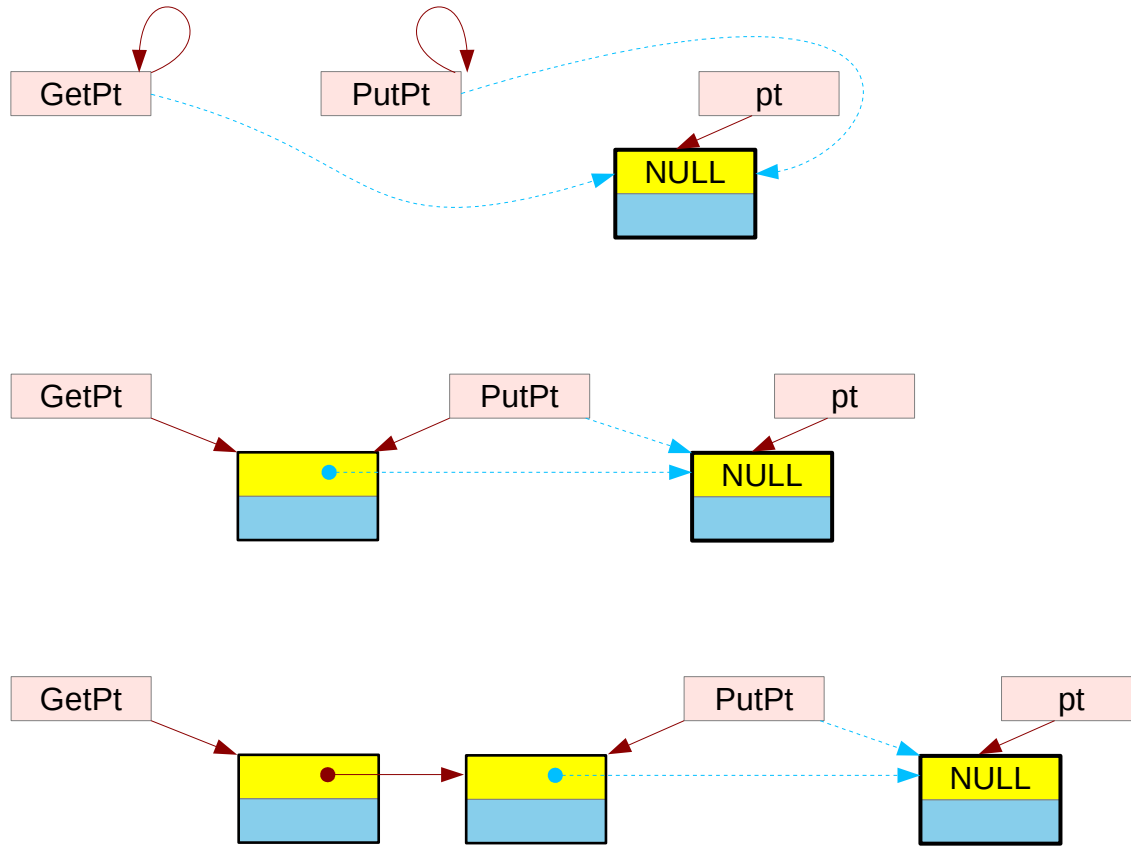
```c
int Fifo_Put(int32_t theData) {
    NodeType *pt;
    pt = (nodeType*) Heap_Allocate();
    if (!pt)  return(0);
    pt->Data = theData;
    pt->Next = NULL;
    if (PutPt) PutPt->Next = pt;
    else GetPt = pt;
    PutPt = pt;
    return(1);
}
```

Fifo_Put – overview

```
; Inputs:    R0 value, data to put
; Outputs:   R0=1 if successful
;            R0=0 in unsuccessful
Fifo_Put

        PUSH {R4, LR}
        MOV  R4, R0          ; copy the data to put into R4
        BL   Heap_Allocate   ; returns the free node address in R0
        CMP  R0, #NULL        ; R0 return value      ; pt node address
        BEQ  Pful            ; skip if full
        STR  R4, [R0, #Data]  ; store 'data'         ; R4 → pt's data
        MOV  R1, #NULL        ; last node            ; R1 ← #NULL
        STR  R1, [R0, #Next]  ; store 'next'         ; R1 → pt's next
        LDR  R2, =PutPt      ; address PutPt        ; R2 ← PutPt (address)
        LDR  R3, [R2]        ; Put node address ; R3 ← [PutPt] (content)
        CMP  R3, #NULL        ; previously empty? ; R3 == #NULL
        BEQ  PMT            ; Put eMpTy routine – no link
        STR  R0, [R3, #Next]  ; link previous        ; R3 → [PutPt]'s next
        B    Pcon
PMT     LDR  R1, =GetPt      ; address GetPt        ; R1 ← GetPt (address)
        STR  R0, [R1]        ; Get node address ; R0 → [GetPt] (content)
PCon    STR  R0, [R2]        ; Points to newst      ; R0 → [PutPt]
                             ; Now one entry
        MOV  R0, #1          ; success
        B    Pdon
PFul    MOV  R0,    #0        ; failure, full
PDon    POP  {R4, PC}
```

```c
int Fifo_Put(int32_t theData) {
    NodeType *pt;
    pt = (nodeType*) Heap_Allocate();
    if (!pt) return(0);
    pt->Data = theData;
    pt->Next = NULL;
    if (PutPt) PutPt->Next = pt;
    else GetPt = pt;
    PutPt = pt;
    return(1);
}
```
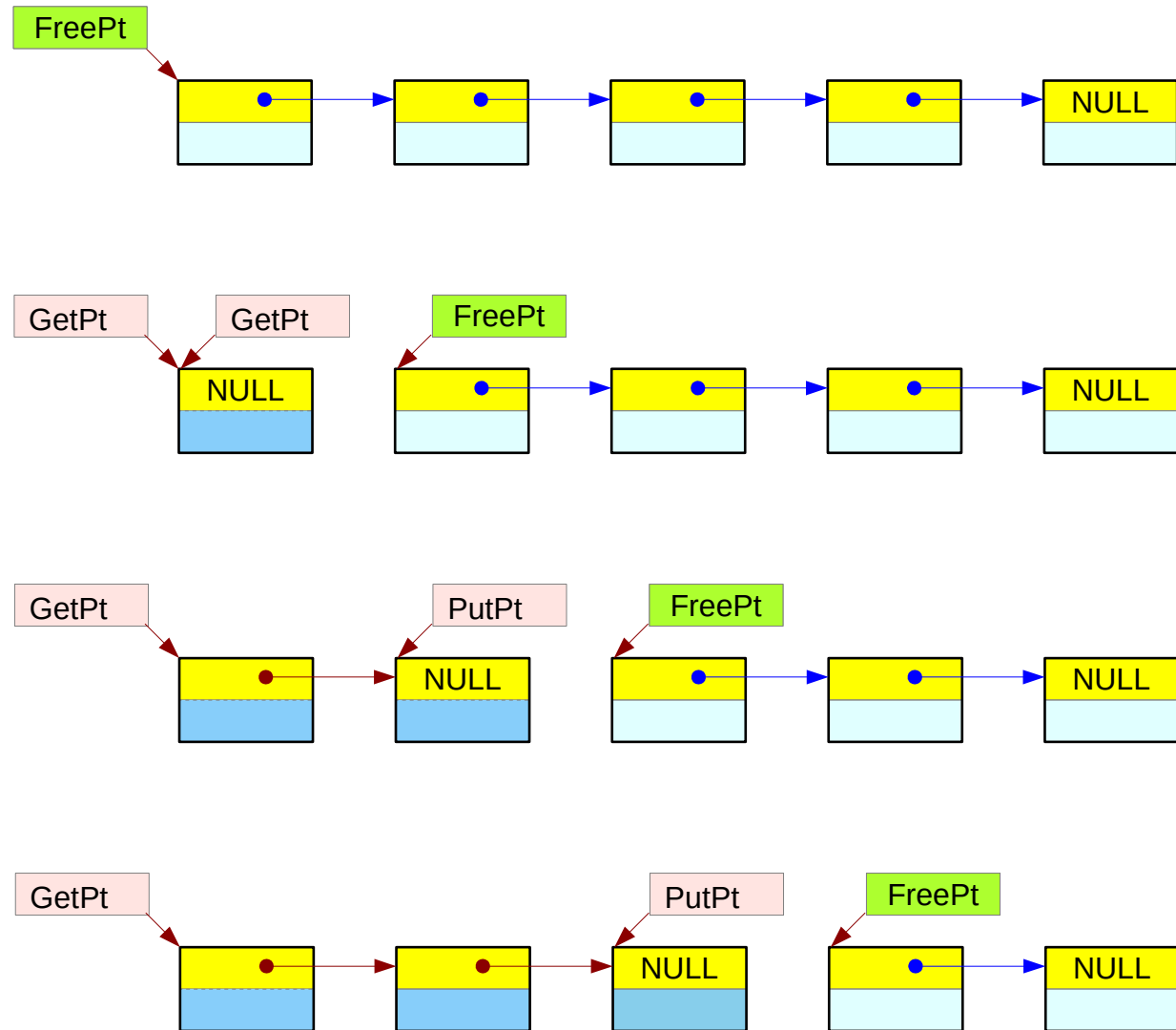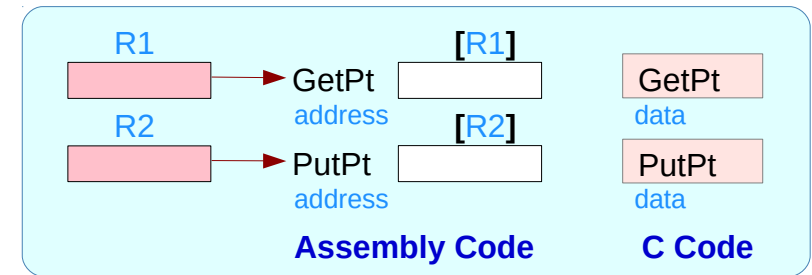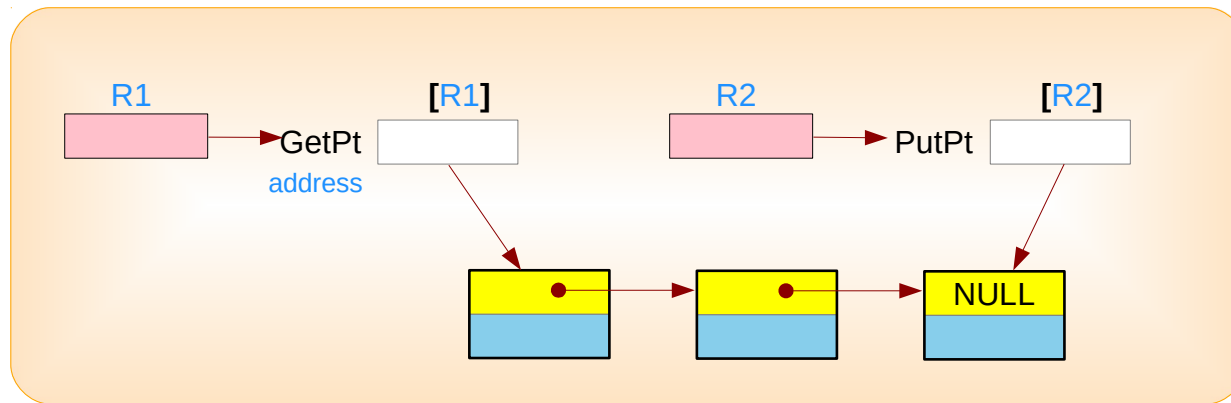
R1 → GetPt [R1]   GetPt
address   data
R2 → PutPt [R2]   PutPt
address   data
Assembly Code   **C Code**

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

Fifo_Put – put operation

Heap_Allocate

returns a free node address
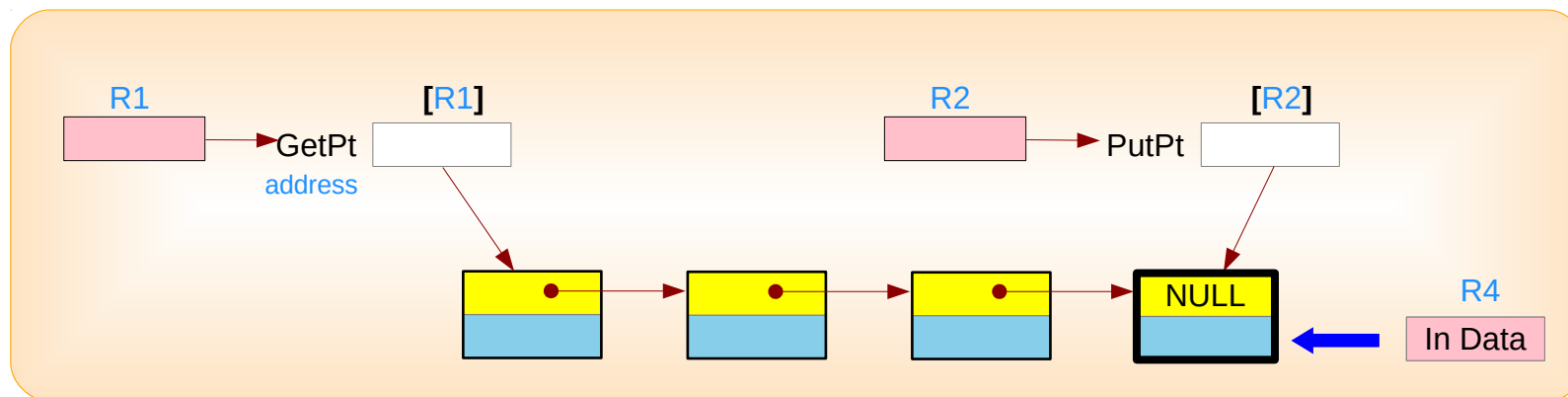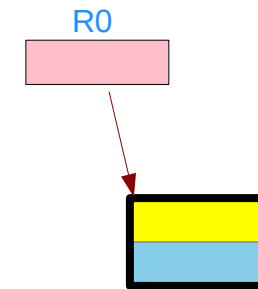
R1

[R1]

GetPt
address

R2

[R2]

PutPt

NULL

R0

R1

[R1]

GetPt
address

R2

[R2]

PutPt

NULL

R4

In Data

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

Fifo_Put

```
        PUSH {R4, LR}
        MOV  R4, R0
BL      Heap_Allocate
        CMP  R0, #NULL
        BEQ  Pful                    Put_Full
        STR  R4, [R0, #Data]
        MOV  R1, #NULL
        STR  R1, [R0, #Next]
        LDR  R2, =PutPt
        LDR  R3, [R2]
        CMP  R3, #NULL
        BEQ  PMT                     Put_Empty
        STR  R0, [R3, #Next]
        B    PCon                    Put_Conclude
PMT     LDR  R1, =GetPt
        STR  R0, [R1]
PCon    STR  R0, [R2]
        MOV  R0, #1
        B    Pdon                    Put_Done
PFul    MOV  R0, #0
PDon    POP  {R4, PC}
```


Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

Fifo_Put – load and store view

Fifo_Put

```
        PUSH {R4, LR}
        MOV  R4, R0
BL      Heap_Allocate
        CMP  R0, #NULL
        BEQ  Pful
        STR  R4, [R0, #Data]
        MOV  R1, #NULL
        STR  R1, [R0, #Next]
        LDR  R2, =PutPt
        LDR  R3, [R2]
        CMP  R3, #NULL
        BEQ  PMT
        STR  R0, [R3, #Next]
        B    Pcon
PMT     LDR  R1, =GetPt
        STR  R0, [R1]
PCon    STR  R0, [R2]
        MOV  R0, #1
        B    Pdon
PFul    MOV  R0, #0
PDon    POP  {R4, PC}
```

R0

R0+0		← R1 (NULL)	returned free node
R0+4		← R4 (input)	by **Heap_Allocate**
address			

R2 — PutPt → PutPt address, [R2] → R3 — Backup the value at PutPt into R3

R3 — [R2] → R3+0 ← R0, R3+4 address — Update the node pointed by PutPt

R1 — GetPt → GetPt address, [R1] ← R0 — Update the value at GetPt

R2 — PutPt → PutPt address, [R2] ← R0 — Update the value at PutPt

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

Fifo_Put – example cases

```
Fifo_Put
        PUSH {R4, LR}
        MOV  R4, R0
BL      Heap_Allocate
        CMP  R0, #NULL
        BEQ  Pful
        STR  R4, [R0, #Data]
        MOV  R1, #NULL
        STR  R1, [R0, #Next]
        LDR  R2, =PutPt
        LDR  R3, [R2]
        CMP  R3, #NULL
        BEQ  PMT
        STR  R0, [R3, #Next]
        B    Pcon
PMT     LDR  R1, =GetPt
        STR  R0, [R1]
PCon    STR  R0, [R2]
        MOV  R0, #1
        B    Pdon
PFul    MOV  R0, #0
PDon    POP  {R4, PC}
```
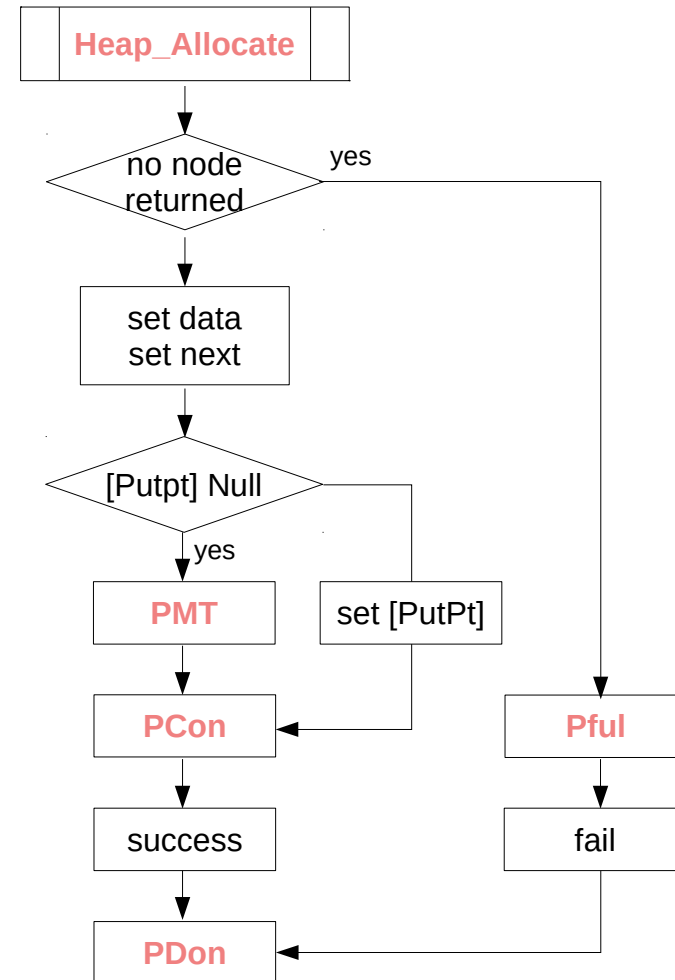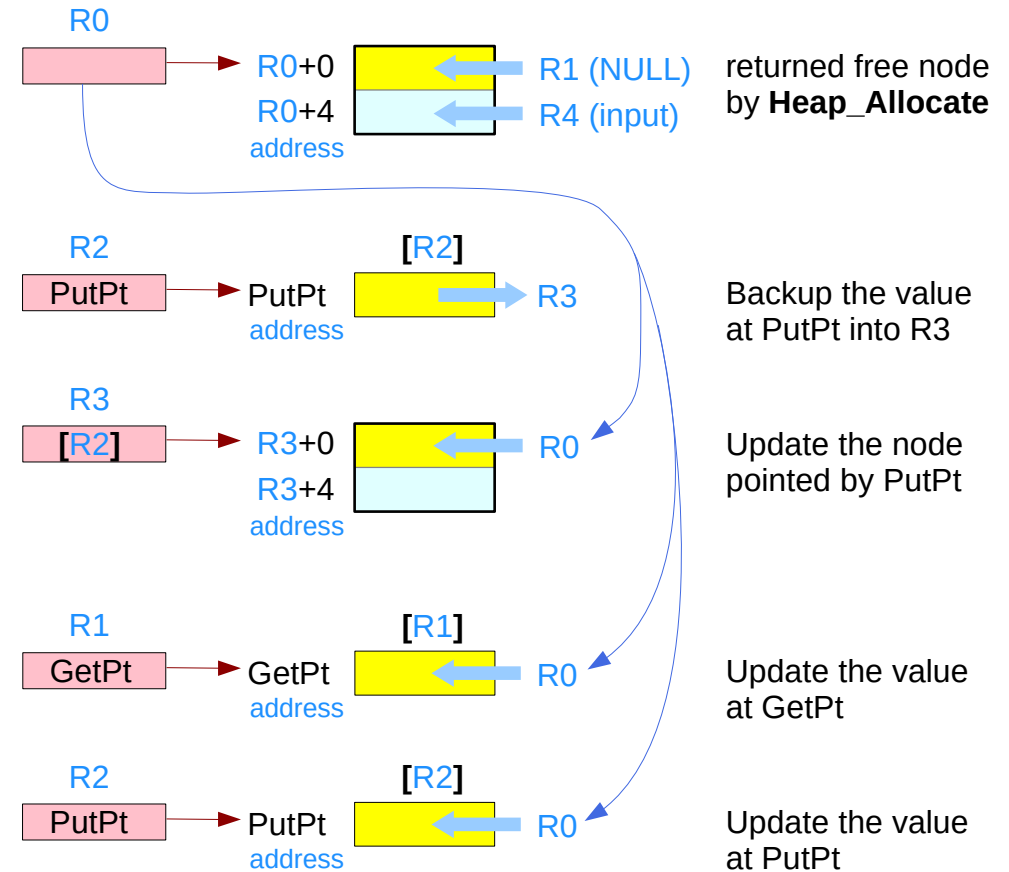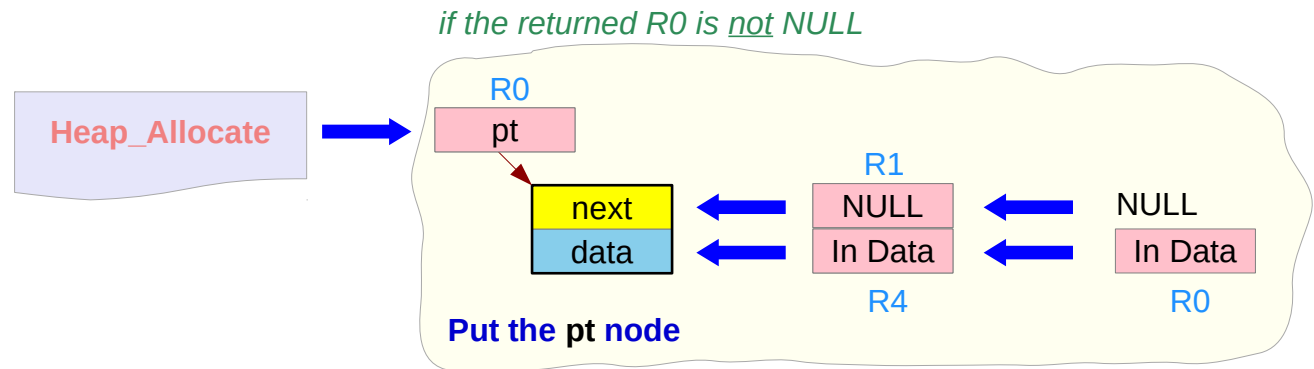

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

Fifo_Put – detailed view (1)

```
; Inputs:        R0 value, data to put
; Outputs:       R0=1 if successful
;                R0=0 in unsuccessful
Fifo_Put
```

R0 → R4
In Data → In Data

```
        PUSH    {R4, LR}
        MOV     R4, R0          ; copy the data to put into R4
        BL      Heap_Allocate   ; returns the free node address in R0
        CMP     R0, #NULL       ; R0 return value    ; pt node address
        BEQ     Pful            ; skip if full
        STR     R4, [R0, #Data] ; store 'data'       ; R4 → pt's data
        MOV     R1, #NULL       ; last node          ; R1 ← #NULL
        STR     R1, [R0, #Next] ; store 'next'       ; R1 → pt's next
```

Heap_Allocate

no node returned — yes → Pful

no → set data / set next

if the returned R0 is not NULL

Heap_Allocate →

R0 pt
R1
next ← NULL ← NULL
data ← In Data ← In Data
R4   R0

Put the pt node

R4 : In Data

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

Fifo_Put – detailed view (2)

LDR	R2, =PutPt	; address PutPt ; R2 ← PutPt (address)
LDR	R3, [R2]	; Put node address ; R3 ← [PutPt] (content)
CMP	R3, #NULL	; previously empty? ; R3 == #NULL
BEQ	PMT	; Put eMpTy routine – no link
STR	R0, [R3, #Next]	; link previous ; R3 → [PutPt]'s next
B	Pcon	

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

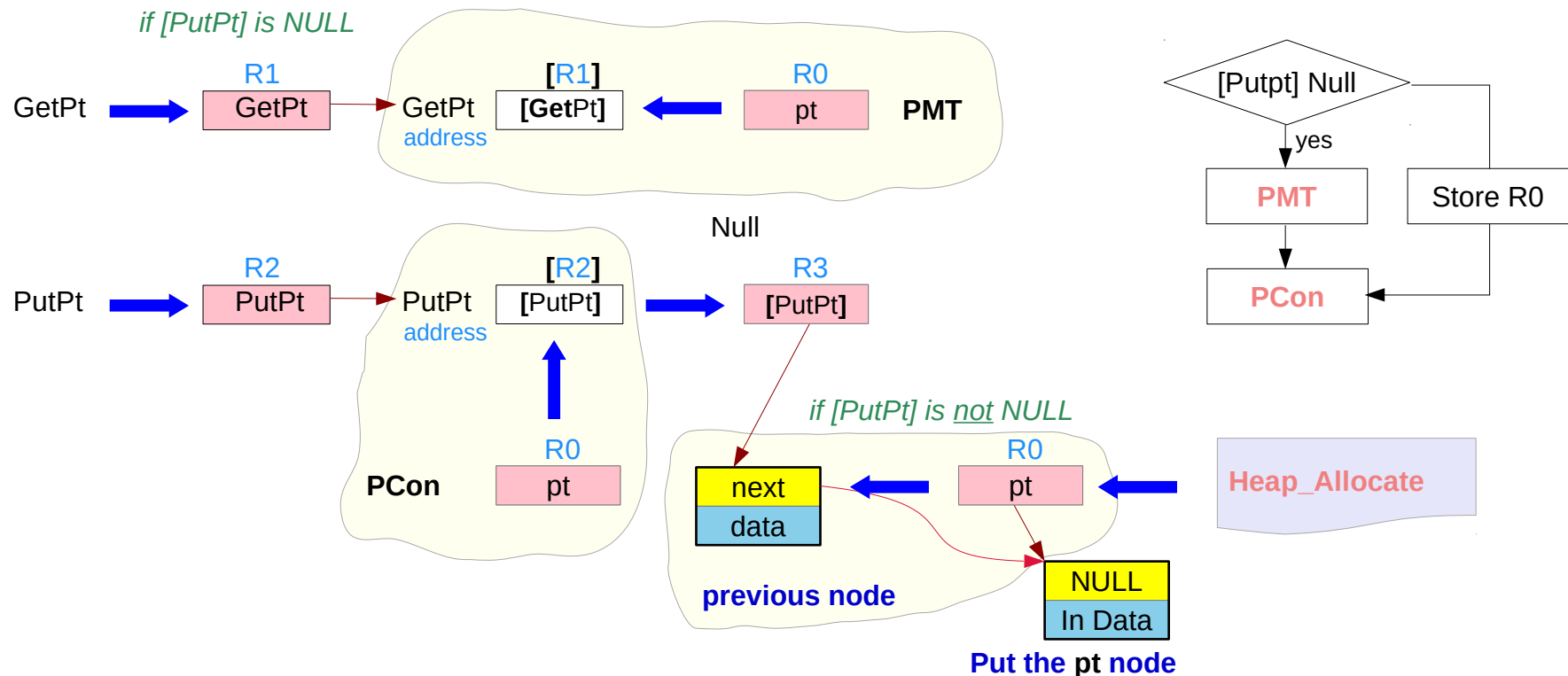# Fifo_Put – detailed view (3)

ARM assembly

PMT	LDR	R1, =GetPt	; address GetPt	; R1 ← GetPt (address)
	STR	R0, [R1]	; Get node address	; R0 → [GetPt] (content)
PCon	STR	R0, [R2]	; Points to newst	; R0 → [PutPt]
			; Now one entry	

if [PutPt] is NULL

R1
GetPt → **R1** GetPt → GetPt → **[R1]** **[Get**Pt**]** ← **R0** pt **PMT**
GetPt address

Null

R2
PutPt → **R2** PutPt → PutPt → **[R2]** **[PutPt]** → **R3** **[PutPt]**
PutPt address

PCon **R0** pt

if [PutPt] is not NULL

R0 pt ← **Heap_Allocate**

next
data

previous node

NULL
In Data

Put the pt node

[Putpt] Null
yes
PMT | Store R0
PCon

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

**Assembly Programming (8A)
Data Structures**

34

Young Won Lim
7/3/20

```
                MOV     R0, #1              ; success
                B       Pdon
PFul            MOV     R0, #0              ; failure, full
PDon            POP     {R4, PC}
```

Heap_Allocate → R0
                   NULL

if [PutPt] is not NULL

R0
Out Data ← #1   PCon

if the returned R0 is NULL

R0
Out Data ← #0   PFul

Heap_Allocate

no node returned → yes → Pful

set data
set next

[Putpt] Null → yes → PMT → PCon

Store R0 → PCon

```
Int Fifo_Get(int32_t *datapt) {
    NodeType *pt;

    if (!GetPt) {
        return (0);                    // empty
    }

    *datapt = GetPt->Data;
    pt = GetPt;
    GetPt = GetPt->Next;

    if (GetPt==NULL) {                // one entry
        PutPt = NULL:
    }

    Heap_Release((int32_t *) pt);

    return (1);                       // success
}
```
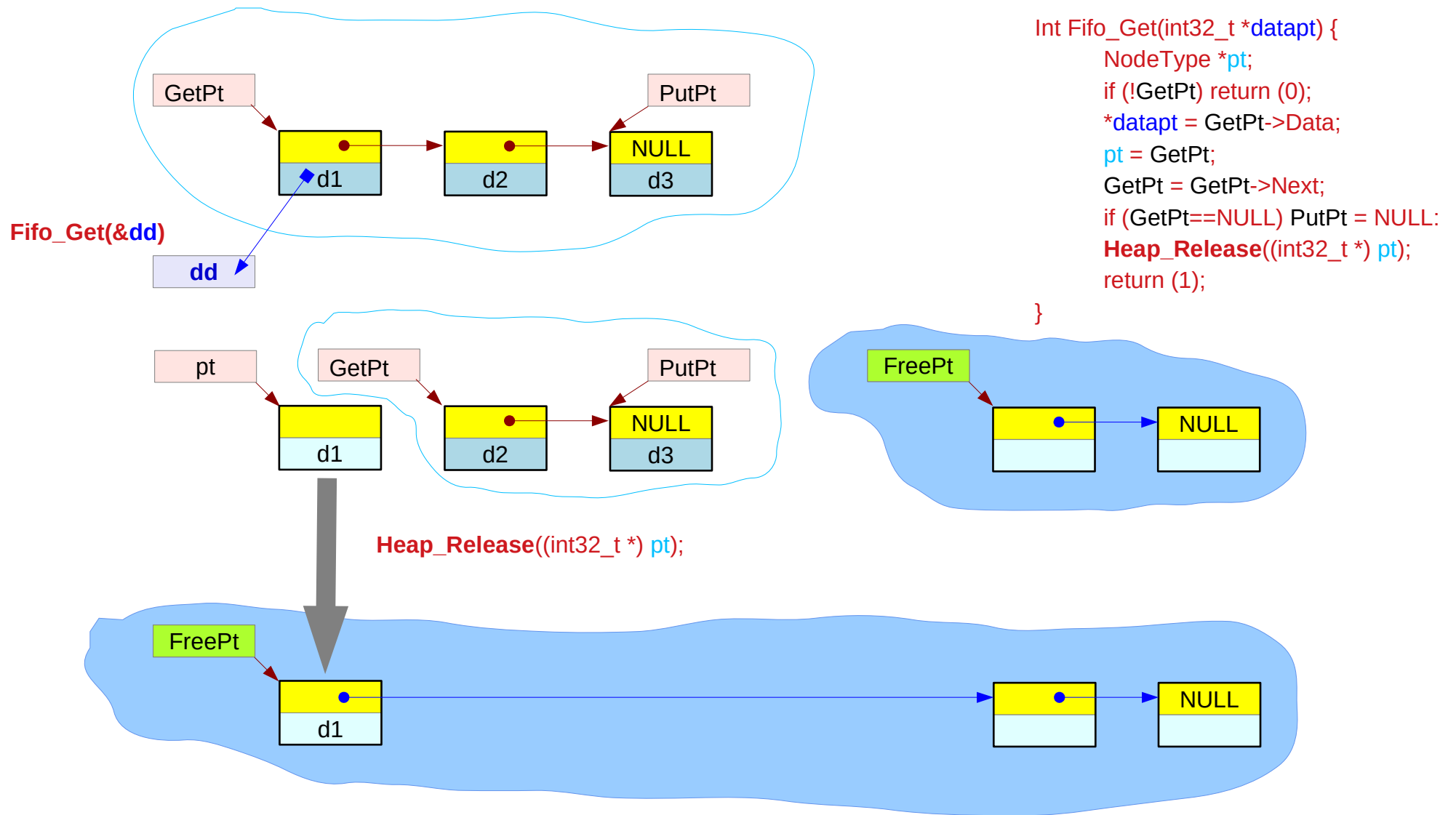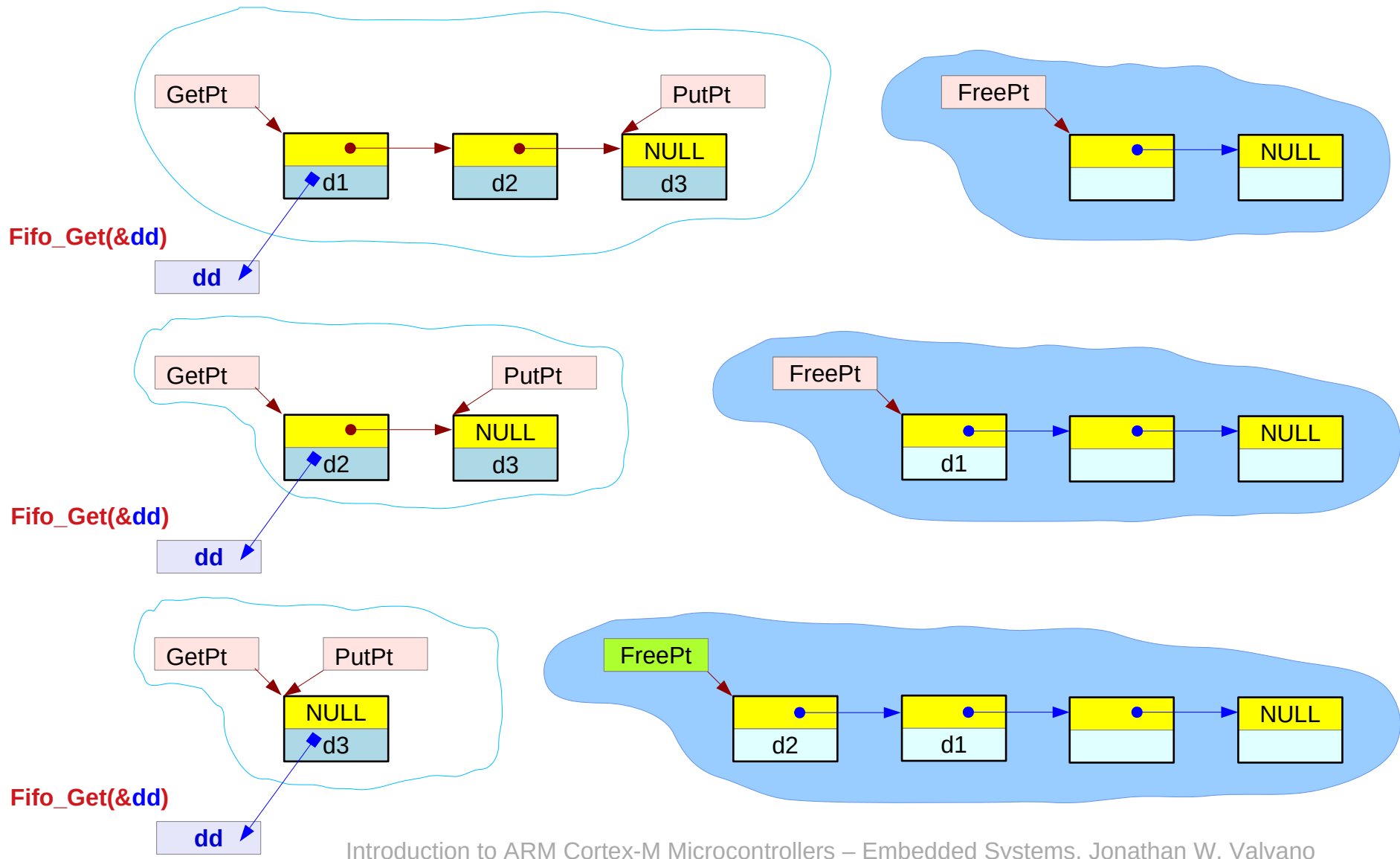
Fifo_Get Examples (1)

```
Int Fifo_Get(int32_t *datapt) {
        NodeType *pt;
        if (!GetPt) return (0);
        *datapt = GetPt->Data;
        pt = GetPt;
        GetPt = GetPt->Next;
        if (GetPt==NULL) PutPt = NULL:
        Heap_Release((int32_t *) pt);
        return (1);
}
```

Fifo_Get(&dd)

dd

Heap_Release((int32_t *) pt);

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

GetPt PutPt

NULL
d1 | d2 | d3

FreePt

NULL

Fifo_Get(&dd)

dd

GetPt PutPt

NULL
d2 | d3

FreePt

d1 | | NULL

Fifo_Get(&dd)

dd

GetPt PutPt

NULL
d3

FreePt

d2 | d1 | | NULL

Fifo_Get(&dd)

dd

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

GetPt   PutPt

NULL

d3

FreePt

d2   d1   NULL

Fifo_Get(&dd)

dd

GetPt
= NULL

PutPt
= NULL

FreePt

d3   d2   d1   NULL

```
Int Fifo_Get(int32_t *datapt) {
    NodeType *pt;
    if (!GetPt) return (0);
    *datapt = GetPt->Data;
    pt = GetPt;
    GetPt = GetPt->Next;
    if (GetPt==NULL) PutPt = NULL:
    Heap_Release((int32_t *) pt);
    return (1);
}
```
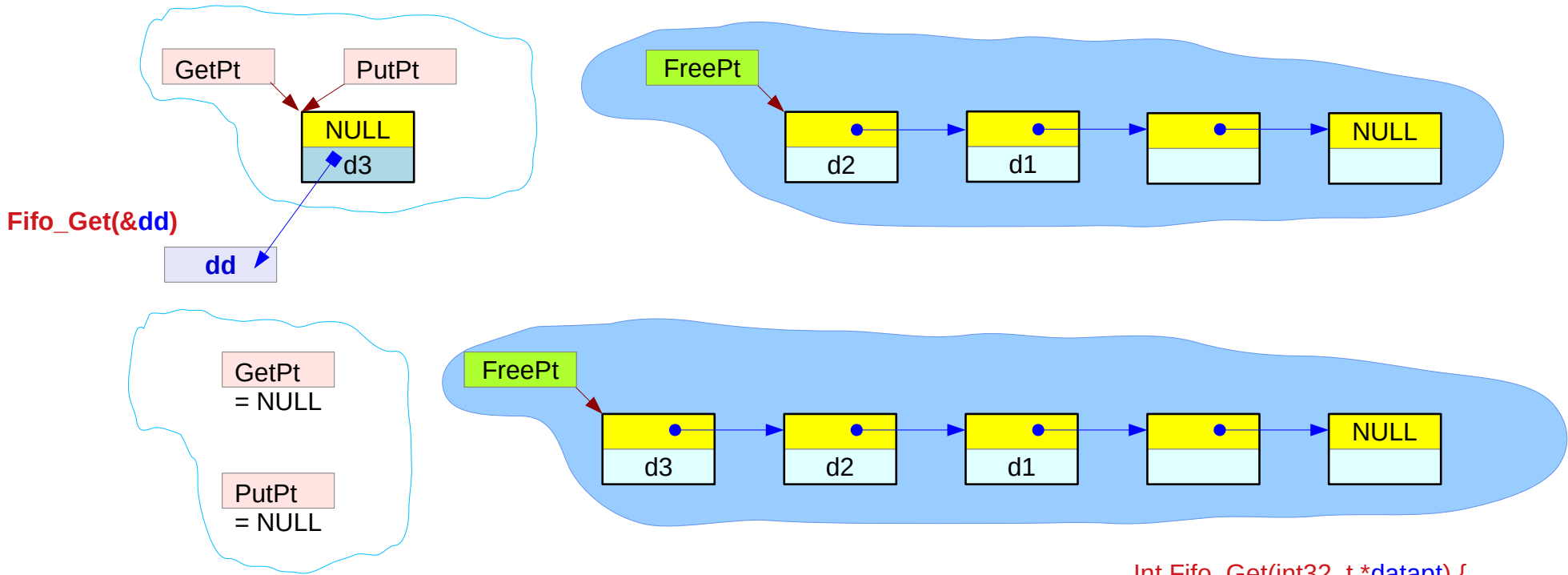
Assembly Programming (8A)
Data Structures

Young Won Lim
7/3/20

GetPt
PutPt
FreePt
NULL

FreePt
GetPt
PutPt
NULL
NULL

FreePt
GetPt
PutPt
NULL
NULL

FreePt
NULL

GetPt
= NULL

PutPt
= NULL

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

Fifo_Get – overview

```
; Inputs:     R0    points to an empty place
; Outputs:    data removed to place
             R0=0  if successful
             R0=1  if empty
```

R1 → GetPt   [R1]
address   GetPt data

R1 → PutPt   [R1]
address   PutPt data

Assembly Code   **C Code**
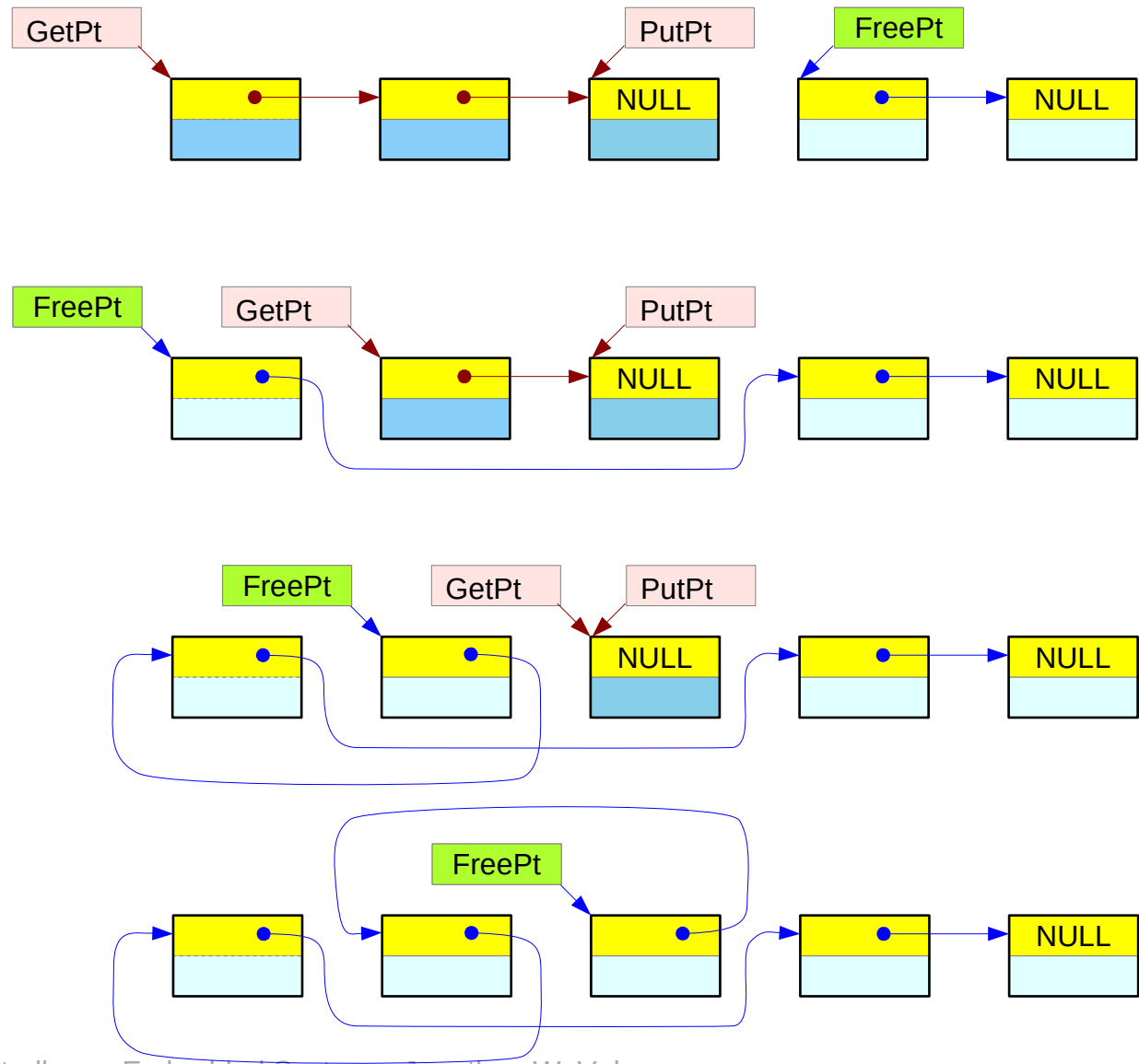
```
Fifo_Get

        PUSH {R4, LR}
        LDR   R1, =GetPt        ; address GetPt      ; R1 ← GetPt (address)
        LDR   R2, [R1]          ; Get node address ; R2 ← [GetPt] (content)
        CMP   R2, #NULL
        BEQ   GMT               ; empty if NULL
        LDR   R3, [R2, #Data]   ; read 'data'        ; R3 ← pt's data
        STR   R3, [R0]          ; store 'data'       ; int Fifo_Get(int32_t *datapt)
        LDR   R3, [R2, #Next]   ; read 'next'        ; R3 ← pt's next
        STR   R3, [R1]          ; store 'next'       ; R3 → [GetPt]
        CMP   R3, #NULL         ; the last node           ; pt node
        BNE   GCon              ; if not empty            ; pt's next  <> NULL
        LDR   R1, =PutPt        ; address PutPt      ; R1 ← PutPt (address)
        STR   R3, [R1]          ; Put node address ; NULL → [PutPt] (content)
GCon    MOV   R0, R2           ; old data          ; R0 ← [GetPt]  ; input arg
        BL    Heap_Release
        MOV   R0, #1            ; success
        B     GDon
GMT     MOV   R0, #0            ; failure, empty
GDon    POP   {R4, PC}
```

```c
int Fifo_Get(int32_t *datapt) {
    NodeType *pt;

    if (!GetPt) return (0);
    *datapt = GetPt->Data;
    pt = GetPt;
    GetPt = GetPt→Next;
    if (GetPt==NULL) PutPt = NULL:
    Heap_Release((int32_t *) pt);
    return (1);
}
```
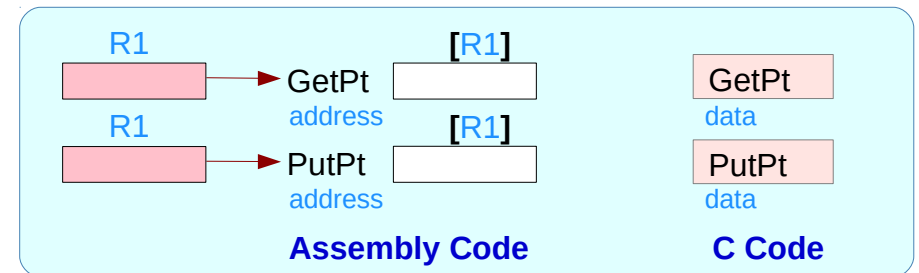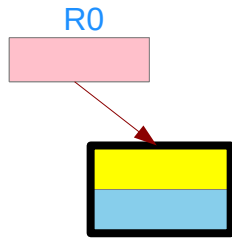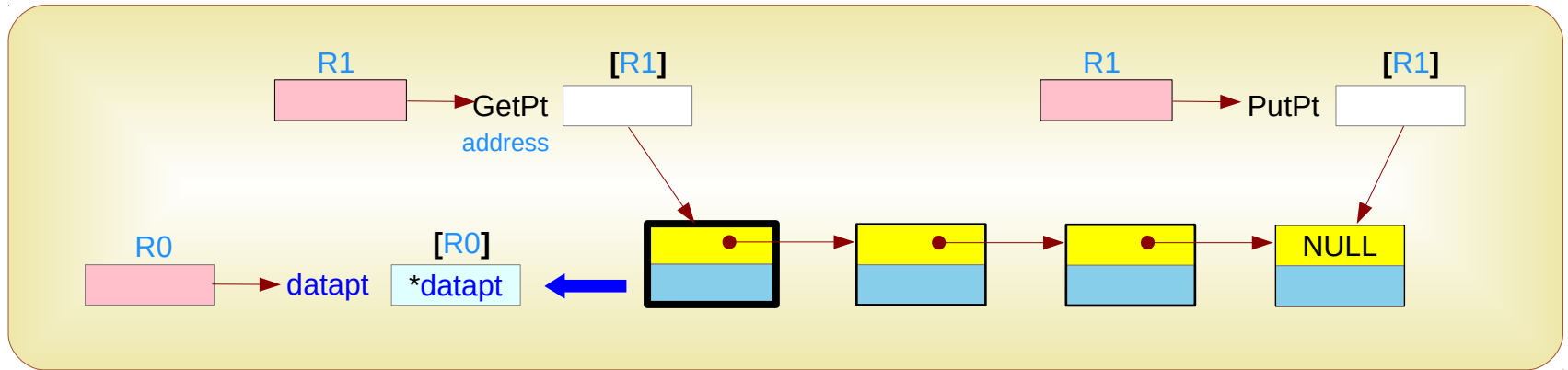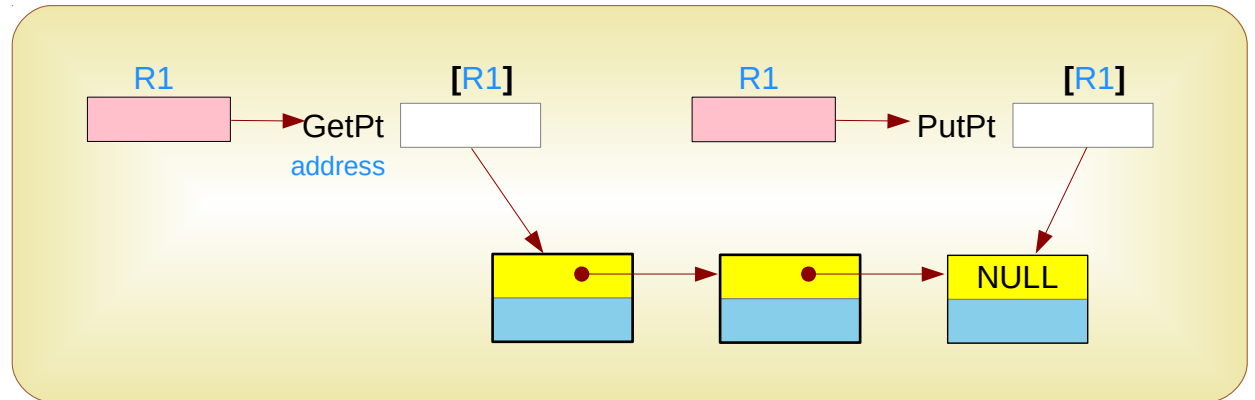
Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

Fifo_Get – get operation

ARM assembly

R1 → GetPt   [R1] address
R0 → datapt   [R0] *datapt

R1 → PutPt   [R1]

NULL

R0

argument : a release node address

Heap_Release

R1 → GetPt   [R1] address
R1 → PutPt   [R1]

NULL

Fifo_Get – flow char view

```
Fifo_Get
        PUSH {R4, LR}
        LDR    R1, =GetPt
        LDR    R2, [R1]
        CMP    R2, #NULL
        BEQ    GMT                    Get_Empty
        LDR    R3, [R2, #Data]
        STR    R3, [R0]
        LDR    R3, [R2, #Next]
        STR    R3, [R1]
        CMP    R3, #NULL
        BNE    GCon                   Get_Conclude
        LDR    R1, =PutPt
        STR    R3, [R1]
Gcon    MOV    R0, R2
        BL     Heap_Release
        MOV    R0, #1
        B      GDon                   Get_Done
GMT     MOV    R0, #0
GDon    POP    {R4, PC}
```


Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

Fifo_Get

```
        PUSH {R4, LR}
        LDR   R1, =GetPt
        LDR   R2, [R1]
        CMP   R2, #NULL
        BEQ   GMT
        LDR   R3, [R2, #Data]
        STR   R3, [R0]
        LDR   R3, [R2, #Next]
        STR   R3, [R1]
        CMP   R3, #NULL
        BNE   GCon
        LDR   R1, =PutPt
        STR   R3, [R1]
Gcon    MOV   R0, R2
        BL    Heap_Release
        MOV   R0, #1
        B     GDon
GMT     MOV   R0, #0
GDon    POP   {R4, PC}
```
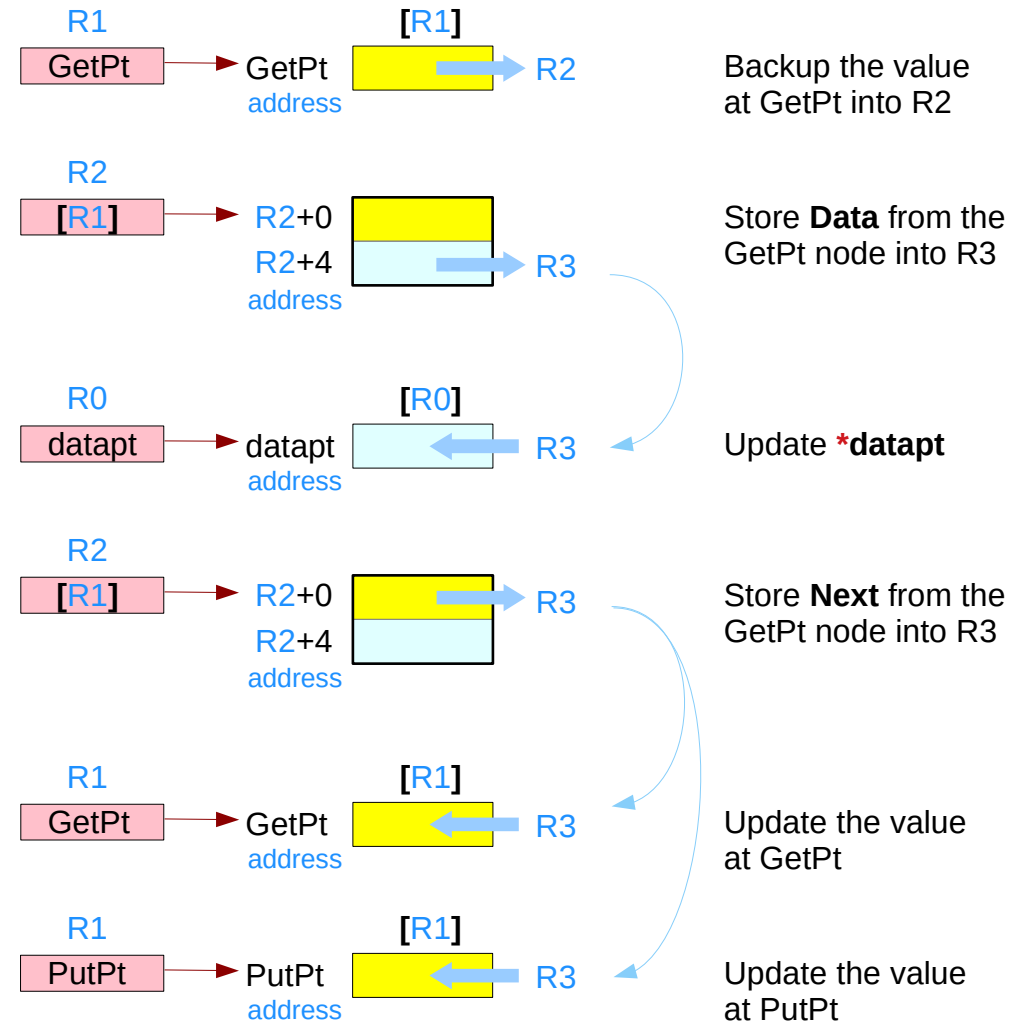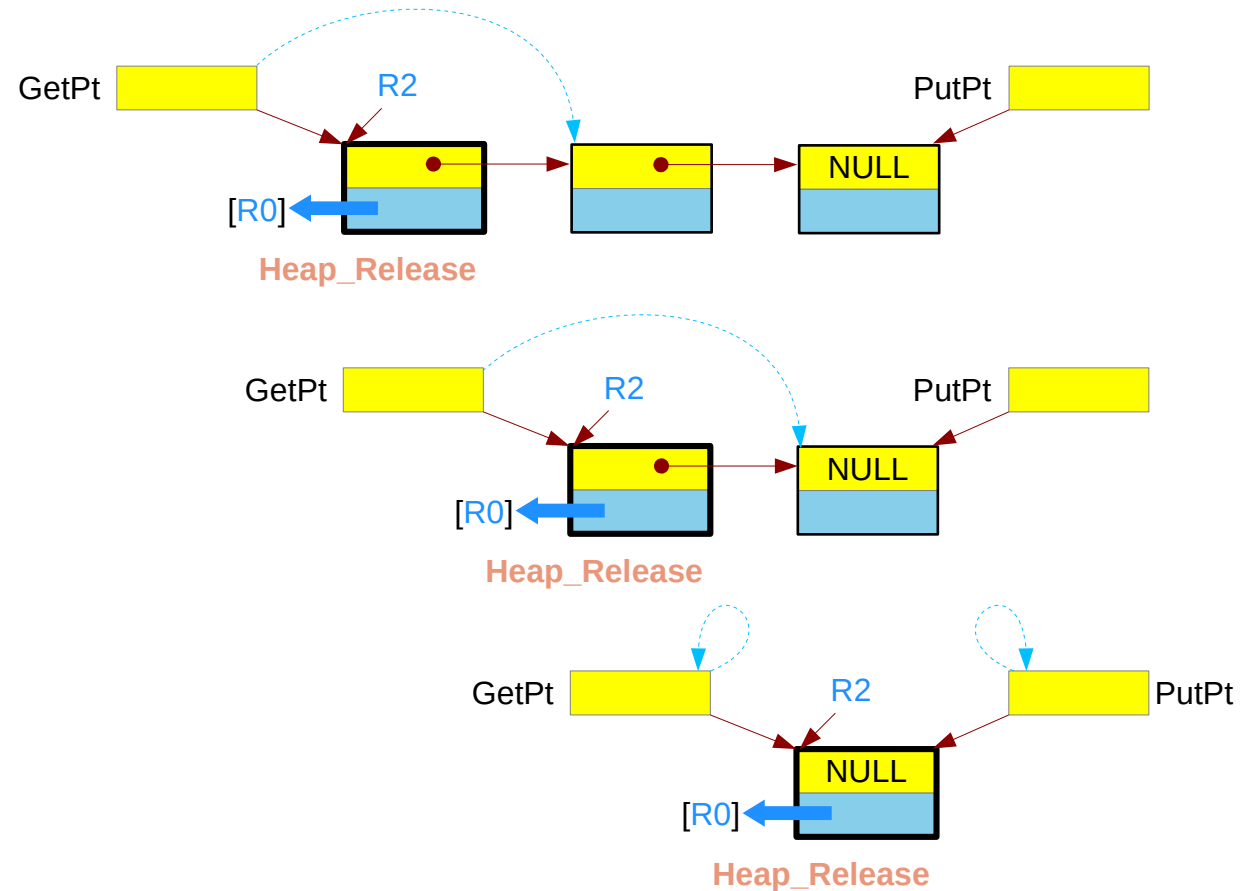
R1   [R1]
GetPt → GetPt   → R2   Backup the value at GetPt into R2
address

R2
[R1] → R2+0
R2+4 → R3   Store **Data** from the GetPt node into R3
address

R0   [R0]
datapt → datapt ← R3   Update *datapt
address

R2
[R1] → R2+0 → R3
R2+4   Store **Next** from the GetPt node into R3
address

R1   [R1]
GetPt → GetPt ← R3   Update the value at GetPt
address

R1   [R1]
PutPt → PutPt ← R3   Update the value at PutPt
address

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

Fifo_Get – example cases

```
Fifo_Get
        PUSH  {R4, LR}
        LDR   R1, =GetPt
        LDR   R2, [R1]
        CMP   R2, #NULL
        BEQ   GMT
        LDR   R3, [R2, #Data]
        STR   R3, [R0]
        LDR   R3, [R2, #Next]
        STR   R3, [R1]
        CMP   R3, #NULL
        BNE   GCon
        LDR   R1, =PutPt
        STR   R3, [R1]
GCon    MOV   R0, R2
        BL    Heap_Release
        MOV   R0, #1
        B     GDon
GMT     MOV   R0, #0
GDon    POP   {R4, PC}
```
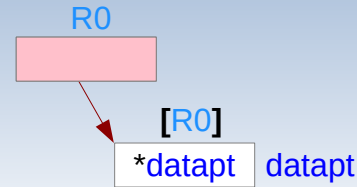

Heap_Release

Heap_Release

Heap_Release

Fifo_Get – detailed view (1)

R0

[R0]

*datapt   datapt

```
; Inputs:      R0   points to an empty place
; Outputs:     data removed to place
              R0=0     if successful
              R0=1     if empty

Fifo_Get

              PUSH     {R4, LR}
              LDR      R1, =GetPt        ; address GetPt      ; R1 ← GetPt (address)
              LDR      R2, [R1]          ; Get node address ; R2 ← [GetPt] (content)
```
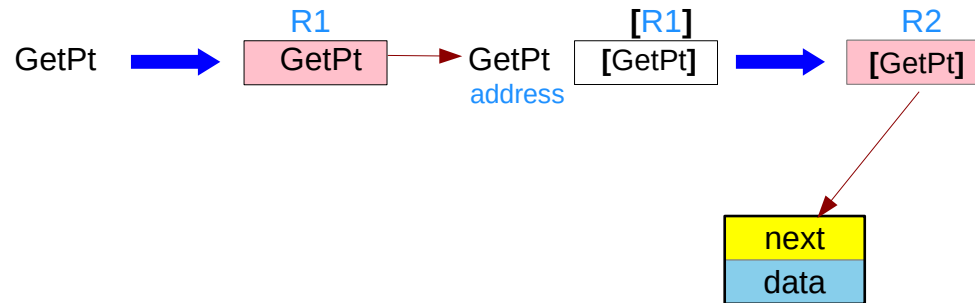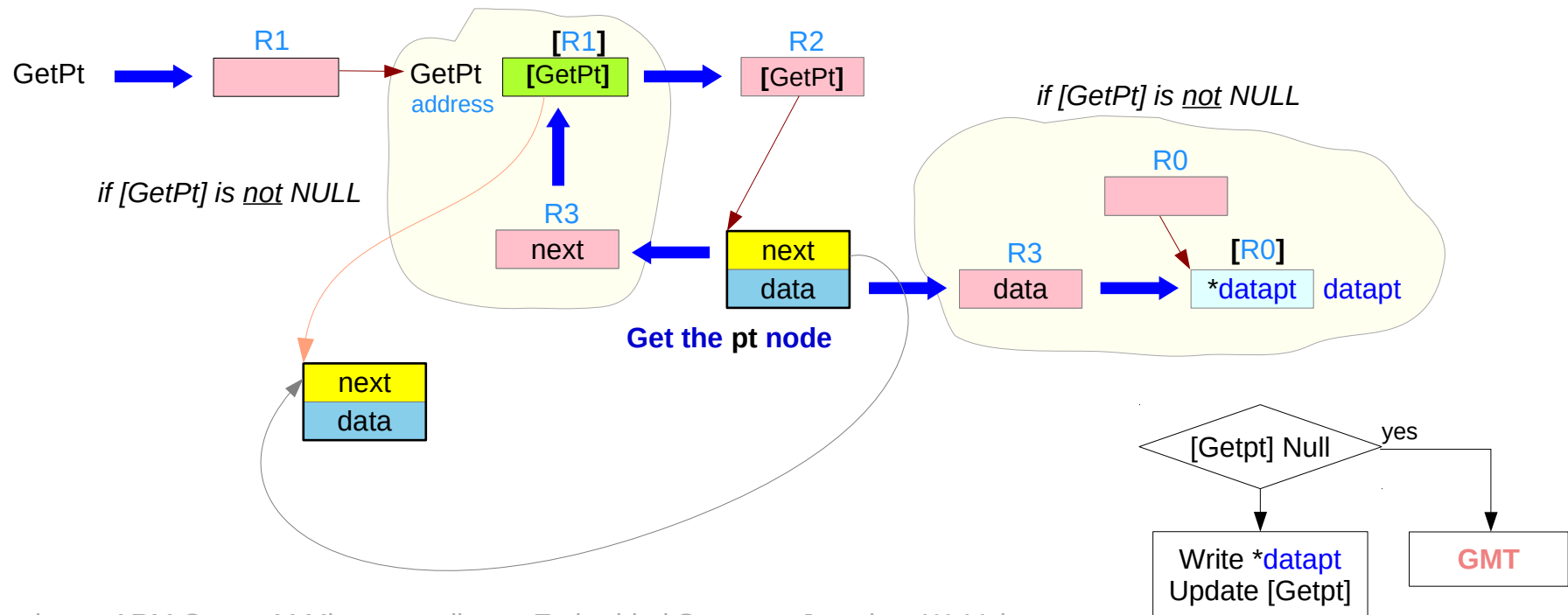
R1
GetPt

[R1]
[GetPt]

R2
[GetPt]

GetPt   GetPt   GetPt   [GetPt]
                  address

next
data

Get the pt node

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano
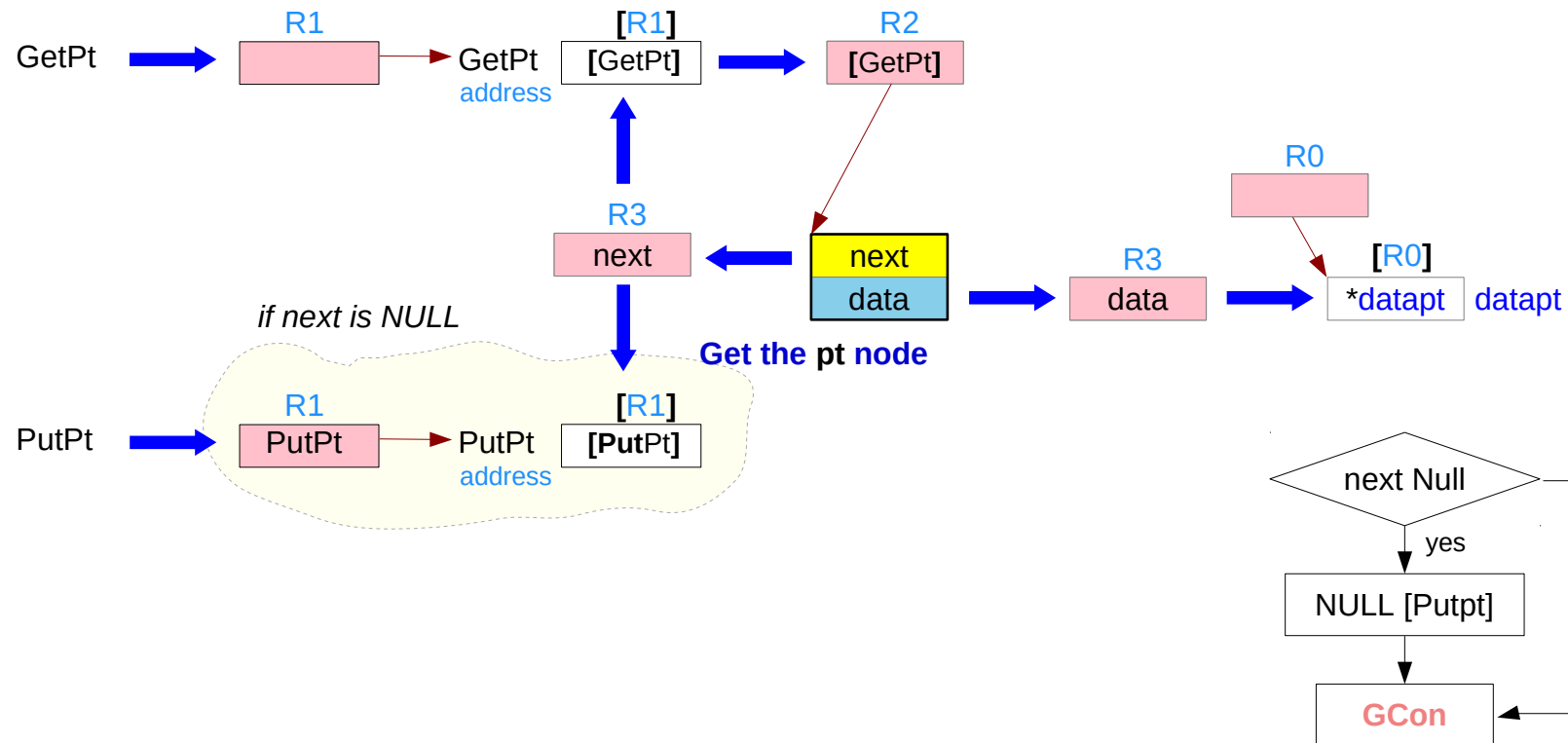
```
CMP      R2, #NULL
BEQ      GMT              ; empty if NULL
LDR      R3, [R2, #Data]  ; read 'data'       ; R3 ← pt's data
STR      R3, [R0]         ; store 'data'      ; int Fifo_Get(int32_t *datapt)
LDR      R3, [R2, #Next]  ; read 'next'       ; R3 ← pt's next
STR      R3, [R1]         ; store 'next'      ; R3 → [GetPt]
```


if [GetPt] is not NULL

Get the pt node

if [GetPt] is not NULL

[Getpt] Null — yes → GMT

Write *datapt
Update [Getpt]

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

Fifo_Get – detailed view (3)

```
CMP     R3, #NULL      ; the last node      ; pt node
BNE     GCon           ; if not empty       ; pt's next  <> NULL
LDR     R1, =PutPt     ; address PutPt      ; R1 ← PutPt (address)
STR     R3, [R1]       ; Put node address   ; NULL → [PutPt] (content)
```
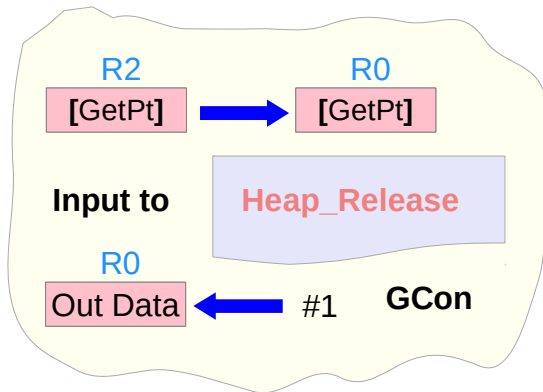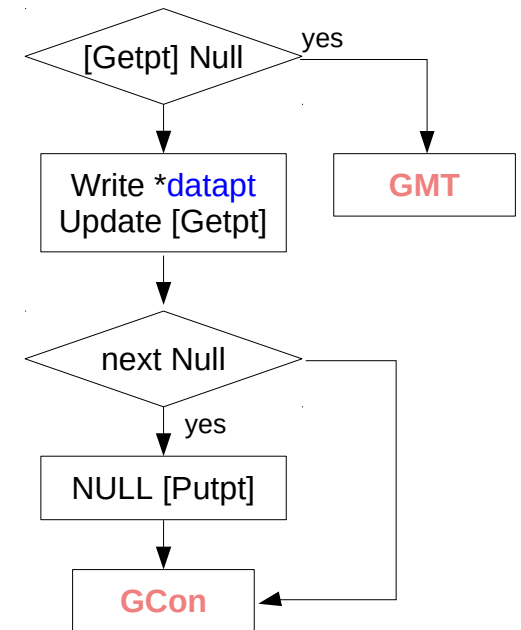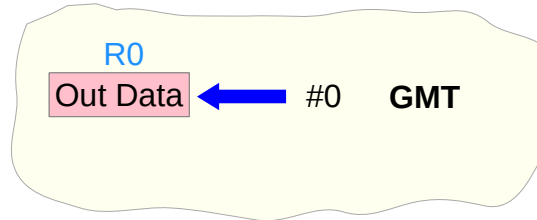
Fifo_Get – detailed view (4)

GCon	MOV	R0, R2	; old data	; R0 ← [GetPt] ; input arg
	BL	**Heap_Release**		
	MOV	R0, #1	; success	
	B	**GDon**		
GMT	MOV	R0, #0	; failure, empty	
GDon	POP	{R4, PC}		

R1   GetPt → [R1] [GetPt] → R2 [GetPt]

GetPt →   GetPt address

if [GetPt] is not NULL

R2 [GetPt] → R0 [GetPt]

Input to   **Heap_Release**

R0 Out Data ← #1   **GCon**

if [GetPt] is NULL

R0 Out Data ← #0   **GMT**

[Getpt] Null — yes → **GMT**

Write *datapt
Update [Getpt]

next Null — yes

NULL [Putpt]

GCon

Matrix read

uint8_t   M[2][3];   // byte matrix with 2 rows and 3 columns

// **Base + n*i +j**

// Read an 8-bit value from (i, j)
// row or column major by compiler
// Input: i is the row index
//   j is the column index
// Output: retrieved value
// Assume: (0 <= I <= 1) and
//   (0 <= j <= 2)

uint8_t **Matrix_read**(uint8_t base[][], uint8_t i, uint8_t j) {
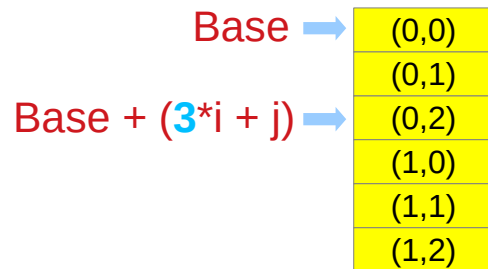   return base[i][j];
}

Accessing and storing matrices

Store in row major order	Store in column major order
byte matrix uint8_t M[2][3]; Base + (3*i + j)	**byte matrix** uint8_t M[2][3]; Base + (2*j + i)
halfword matrix uint16_t M[2][3]; Base + 2*(3*i + j)	**halfword matrix** uint16_t M[2][3]; Base + 2*(2*j + i)
word matrix uint32_t M[2][3]; Base + 4*(3*i + j)	**word matrix** uint32_t M[2][3]; Base + 4*(2*j + i)

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano
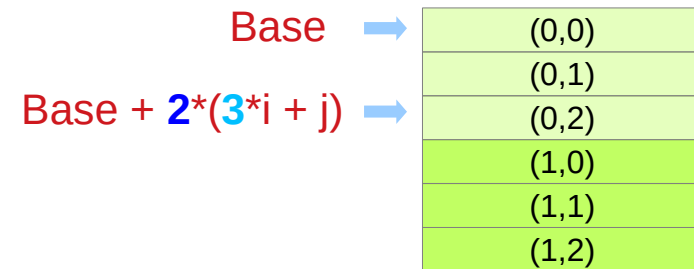
Matrix M[m][n] in row major order

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)

byte matrix   uint8_t   M[2][3];

halfword matrix   uint16_t   M[2][3];

Base ➡
(0,0)
(0,1)

Base + (**3***i + j) ➡
(0,2)
(1,0)
(1,1)
(1,2)

Base ➡
(0,0)
(0,1)

Base + **2***(**3***i + j) ➡
(0,2)
(1,0)
(1,1)
(1,2)

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)

word matrix   uint32_t   M[2][3];

Base ➡
(0,0)
(0,1)

Base + **4***(**3***i + j) ➡
(0,2)
(1,0)
(1,1)
(1,2)

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano
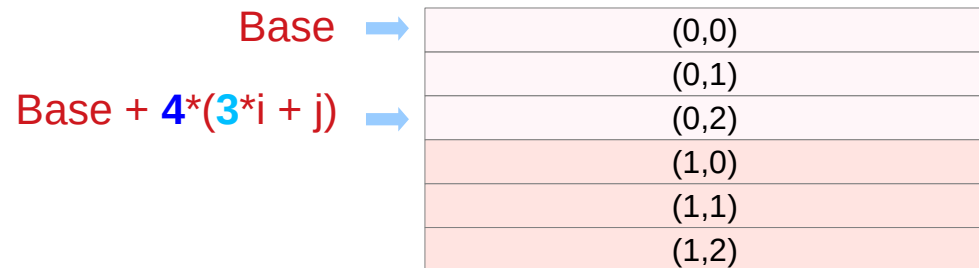
row major order vs column major order

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
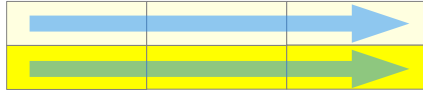
(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)

uint8_t   M[2][3];

uint8_t   M[2][3];

byte matrix – <u>row</u> major order
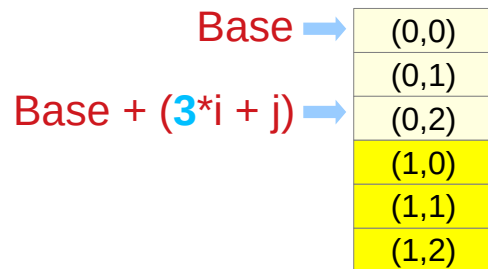
byte matrix – <u>column</u> major order

Base ➡ (0,0)
          (0,1)
Base + (3*i + j) ➡ (0,2)
          (1,0)
          (1,1)
          (1,2)

Base ➡ (0,0)
          (1,0)
          (0,1)
          (1,1)
Base + (2*j + i) ➡ (0,2)
          (1,2)

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano
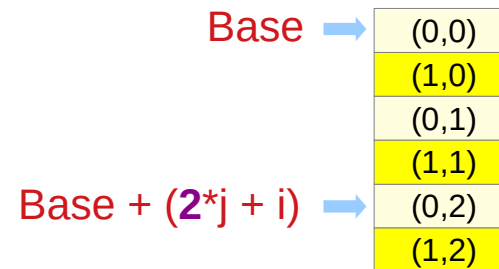
Matrix

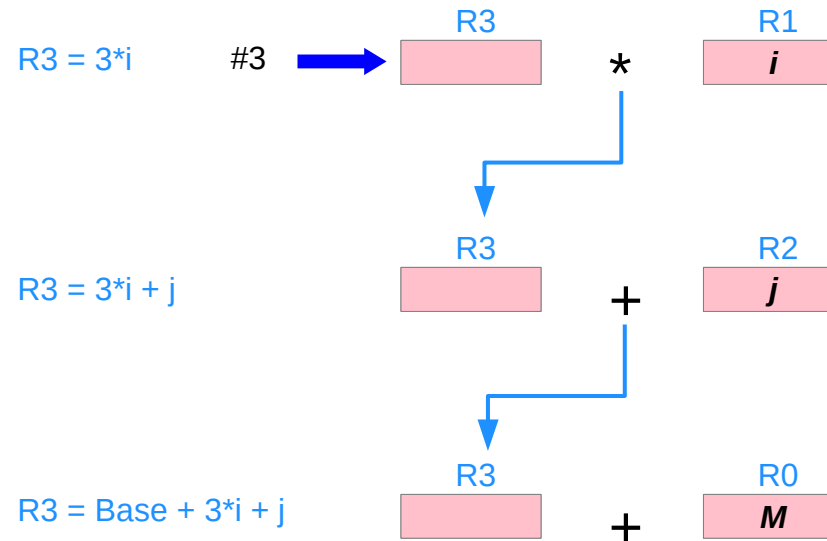```
; Read an 8-bit value from (i, j)   ;; Base + n*i +j
; Input: Base   (R0)   pointer to matrix   ;; R0 + n*R1 + R2
;   i   (R1)   is the row index
;   j   (R2)   is the column index
; Assume: (0 <= R1 <= 1)
;   (0 <= R2 <= 2)
Matrix_read
          MOV   R3, #3   ; R3 = 3 columns M[2][3]
          MUL   R3, R3, R1   ; R3 = 3*i
          ADD   R3, R3, R2   ; R3 = 3*i + j
          ADD   R3, R3, R0   ; R3 = Base + 3*i + j
          LDRB   R0, [R3]   ; R0 = M[i, j]
          BX   LR   ; return
```

| R0 | R1 | R2 |
|----|----|----|
| Base | *i* | *j* |
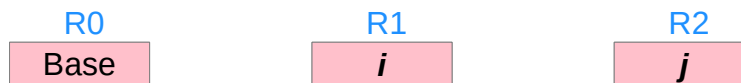
# Matrix

```
Matrix_read
        MOV   R3, #3   ; R3 = 3 columns M[2][3]
        MUL   R3, R3, R1   ; R3 = 3*i
        ADD   R3, R3, R2   ; R3 = 3*i + j
        ADD   R3, R3, R0   ; R3 = Base + 3*i + j
        LDRB R0, [R3]   ; R0 = M[i, j]
        BX   LR   ; return
```
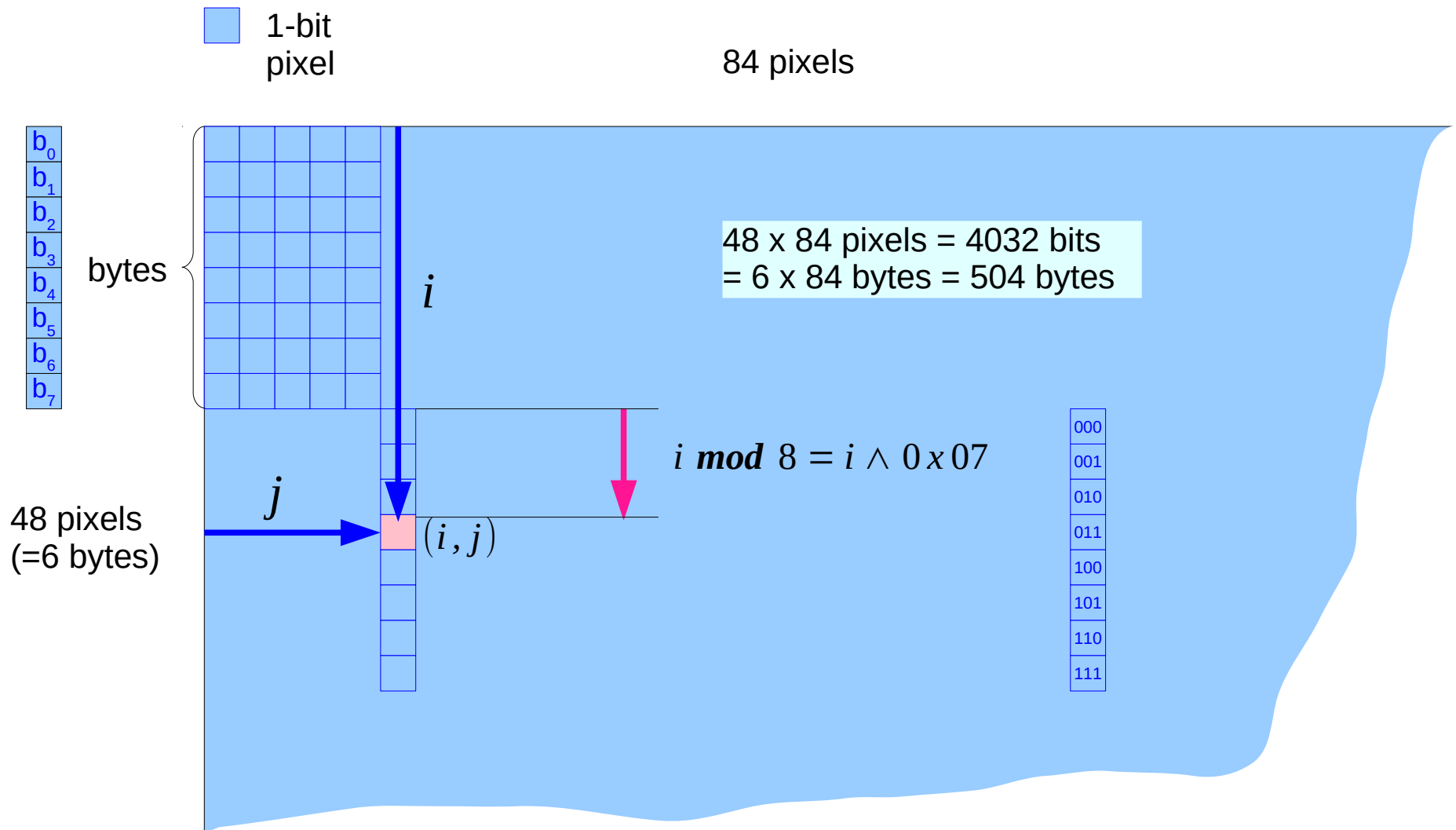
R3 = 3*i       #3 →

R3 = 3*i + j

R3 = Base + 3*i + j

**Assembly Programming (8A)**
**Data Structures**

Young Won Lim
7/3/20

# 48 x 84 B/W Image

1-bit pixel

84 pixels

$b_0$
$b_1$
$b_2$
$b_3$
$b_4$
$b_5$
$b_6$
$b_7$

bytes

$i$

48 x 84 pixels = 4032 bits
= 6 x 84 bytes = 504 bytes

$i \bmod 8 = i \wedge 0x07$

$j$

48 pixels
(=6 bytes)

$(i, j)$

000
001
010
011
100
101
110
111

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

# Matrix

uint8_t Screen[504]                  // stores the next image to be printed on the screen

Screen + 84*(i>>3)                   // Screen + 84*(*i* **div** 8)
Screen + 84*(i>>3) +j;               // Screen + 84*(*i* **div** 8) + j

k = i & 0x07;                        // i **mod** 8

Masks FCB     0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80

Masks ➡

| | | |
|---|---|---|
| 0x01 | 0000_0001 | $b_0$ |
| 0x02 | 0000_0010 | $b_1$ |
| 0x04 | 0000_0100 | $b_2$ |
| 0x08 | 0000_1000 | $b_3$ |
| 0x10 | 0001_0000 | $b_4$ |
| 0x20 | 0010_0000 | $b_5$ |
| 0x40 | 0100_0000 | $b_6$ |
| 0x80 | 1000_0000 | $b_7$ |

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano
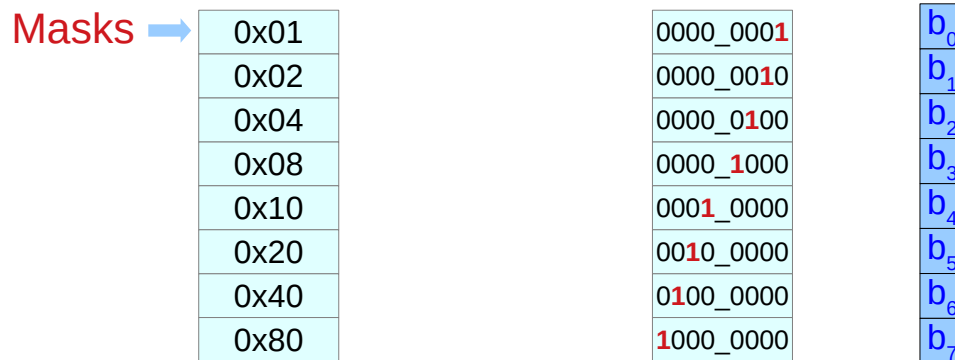
# FCB Directive

fcb Form Constant Byte

    (<label>) fcb <expr>(,<expr>,...,<expr>) (<comment>)
    (<label>) dc.b <expr>(,<expr>,...,<expr>) (<comment>)
    (<label>) db <expr>(,<expr>,...,<expr>) (<comment>)
    (<label>) .byte <expr>(,<expr>,...,<expr>) (<comment>)

The FCB directive may have one or more operands separated by commas. The value of each operand is truncated to eight bits, and is stored in a single byte of the object program. Multiple operands are stored in successive bytes. The operand may be a numeric constant, a character constant, a symbol, or an expression. If multiple operands are present, one or more of them can be null (two adjacent commas), in which case a single byte of zero will be assigned for that operand. An error will occur if the upper eight bits of the evaluated operands' values are not all ones or all zeros.

    A string can be included, which is stored as a sequence of ASCII characters. The delimitors supported by TExaS are " ' and \. The string is not terminated, so the programmer must explicitly terminate it. For example:

str1 fcb "Hello World",0

# Matrix

; Input :    R0 the row index *i* (0 to 47 in this case), Y-coordinate
;                 R1 the column index *j* (0 to 83 in this case), X-coordinate
; Output:  **R0** points to the <u>byte</u> of interest
                 **R1** the Mask to access that <u>1-bit</u> pixel

| R0 | R1 |
|----|----|
| *i* | *j* |

BL         Access

byte addr      mask
R0              R1

Access           ; Access the Image pixel at (*i*, *j*)

```
LDR   R3, =Masks   ; R3 = Masks (pointer)
AND   R2, R0, #0x07   ; R2 = k = i & 0x07
LDRB   R4, [R3, R2]   ; R2 = Mask[k]
LDR   R3, =Screen   ; R3 = Screen (pointer)
LSR   R0, R0, #3   ; R0 = R0 >> 3 (i div 8)
MOV   R2, #84
MUL   R2, R0, R2   ; R2 = 84 * (i >> 3)
ADD   R0, R2, R3   ; R0 = Screen + 84 * (i >> 3)   (pointer)
ADD   R0, R0, R1   ; R0 = Screen + 84 * (i >> 3) + j   (pointer)
MOV   R1, R4   ; R1 is Masks[i mod 8]
BX   LR
```
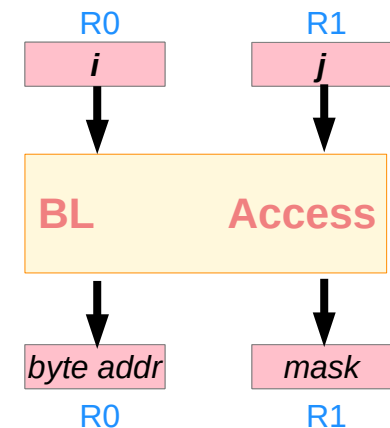
# Matrix

```
Access   ; Access the Image pixel at (i, j)
    LDR   R3, =Masks   ; R3 = Masks (pointer)
    AND   R2, R0, #0x07   ; R2 = k = i & 0x07
    LDRB R4, [R3, R2]   ; R2 = Mask[k]
    LDR   R3, =Screen   ; R3 = Screen (pointer)
    LSR   R0, R0, #3   ; R0 = R0 >> 3 (i div 8)
    MOV   R2, #84
    MUL   R2, R0, R2   ; R2 = 84 * (i >> 3)
    ADD   R0, R2, R3   ; R0 = Screen + 84 * (i >> 3)   (pointer)
    ADD   R0, R0, R1   ; R0 = Screen + 84 * (i >> 3) + j (pointer)
    MOV   R1, R4   ; R1 is Mask[i mod 8]
    BX   LR
```
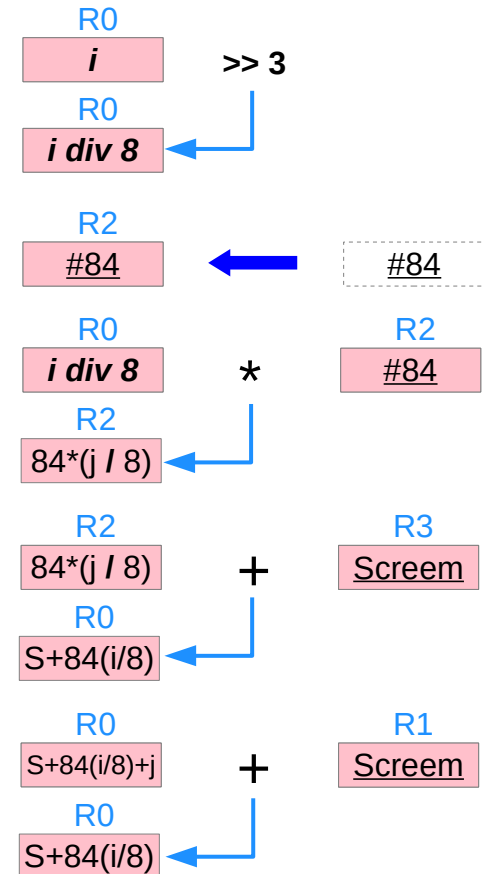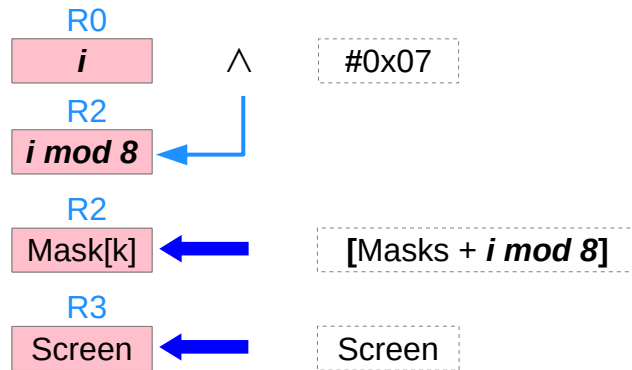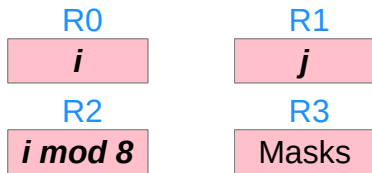
# Clear a pixel

; Clear the Image pixel at (*i*, *j*), turning it dark
; Input:     R0 the row index *i* (0 to 47 in this case)          Y-coordinate
              R1 the column index *j* (0 to 83 in this case)      X-coordinate
; Output:  none        modifies R0, R1, R2, R3

ClrPxl
              PUSH      {LR}
              **BL**        **Access**                        ; get pointer to pixel to change
              LDRB      R3, [**R0**]                       ; R3 = [R0] = read 8 pixels
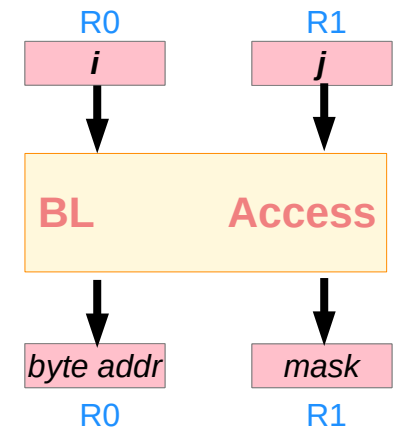              BIC        R3, R3, **R1**                    ; R3 = R3 & ~R1     clear proper pixel
              STRB      R3, [**R0**]                       ; [R0] = R3 = write 8 pixels
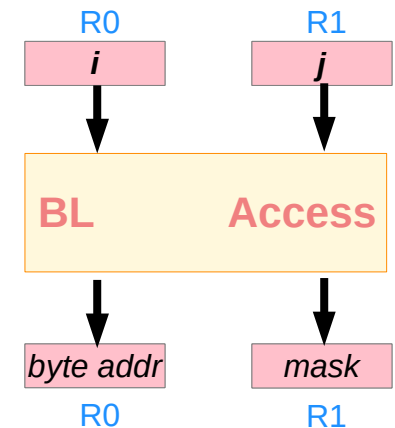              POP        {PC}

| R0 | R1 |
|---|---|
| *i* | *j* |

**BL          Access**

| byte addr | mask |
|---|---|
| R0 | R1 |

# Set a pixel

; Set the Image pixel at (I, j) to the given value
; Input:    R0 the row index I (0 to 47 in this case)          Y-coordinate
            R1 the column index j (0 to 83 in this case)       X-coordinate
; Output:  none        modifies R0, R1, R2, R3

SetPxl

        PUSH     {LR}
        **BL**       **Access**          ; get pointer to pixel to change
        LDRB     R3, [**R0**]         ; R3 = [R0] = read 8 pixels
        ORR      R3, R3, **R1**      ; R3 = R3 | R1        set proper pixel
        STRB     R3, [**R0**]        ; [R0] = R3 = write 8 pixels
        POP      {PC}

| R0 | R1 |
|----|----|
| *i* | *j* |

BL       Access

| byte addr | mask |
|-----------|------|
| R0 | R1 |

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

# Table

```
const int32_t IxTabl[22] = {
      0,13,26,38,51,64,77,90,102,
      115,128,141,154,166,179,192,205,
      218,230,243,255, 256};

const int32_t IyTabl[22] = {
      0,39,75,103,121,127,121,103,
      75,39,0,-39,-75,-103,-121,-127,
      -121,-103,-75,-39,0,0};
```

```
IxTabl    DCD       0,13,26,38,51,64,77,90,102
          DCD       115,128,141,154,166,179,192,205
          DCD       218,230,243,255, 256

IyTabl    DCD       0,39,75,103,121,127,121,103
          DCD       75,39,0,-39,-75,-103,-121,-127
          DCD       -121,-103,-75,-39,0,0
```

Table

```
// Ix is 0 to 255 (pi/128)
// Iy is -127 to +127 (1/256)

int32_t Sin(int32_t Ix) {
     int32_t x1, x2, y1, y2;
     int i=0;

     while (Ix >= IxTab(i+1)) {
          i++;
     }
     x1 = IxTab[i];
     x2 = IxTab[i+1];
     y1 = IyTab[i];
     y2 = IyTab[i+1];

     return ((y2-y1)*(Ix-x1))/(x2-x1) + y1;
}
```

Table

```
; Input :   R0 is 0 to 255, Ix
; Output:  R1 is -127 to +127, Iy

Sin             PUSH    {R4-R6, LR}
                LDR     R1, =IxTab              ; find x1 <= Ix < x2
                LDR     R2, =IyTab
Lookx1          LDR     R6, [R1, #4]        ; x2
                CMP     R0, R6              ; check Ix < x2
                BLO     found               ; R1 >= x1
                ADD     R1, #4
                ADD     R2, #4
                B       lookx1
```

Table

```
Found        LDR     R4, [R1]              ; x1
             SUB     R4, R0, R4            ; Ix-x1
             LDR     R5, [R2, #4]          ; y2
```

Assembly Programming (8A)
Data Structures

Young Won Lim
7/3/20

Pointer access to an array

References

[1] ftp://ftp.geoinfo.tuwien.ac.at/navratil/HaskellTutorial.pdf
[2] https://www.umiacs.umd.edu/~hal/docs/daume02yaht.pdf