

CS/ECE 5780/6780: Embedded System Design

John Regehr

Lecture 11: Threads

Today: Threads

We start by looking at the implementation side
Later, we look at how to use threads

Introduction to Threads

Interrupts create a concurrent environment with a single foreground thread (the main program) plus the ISRs.

In projects where modules are loosely coupled, multiple foreground threads may be necessary.

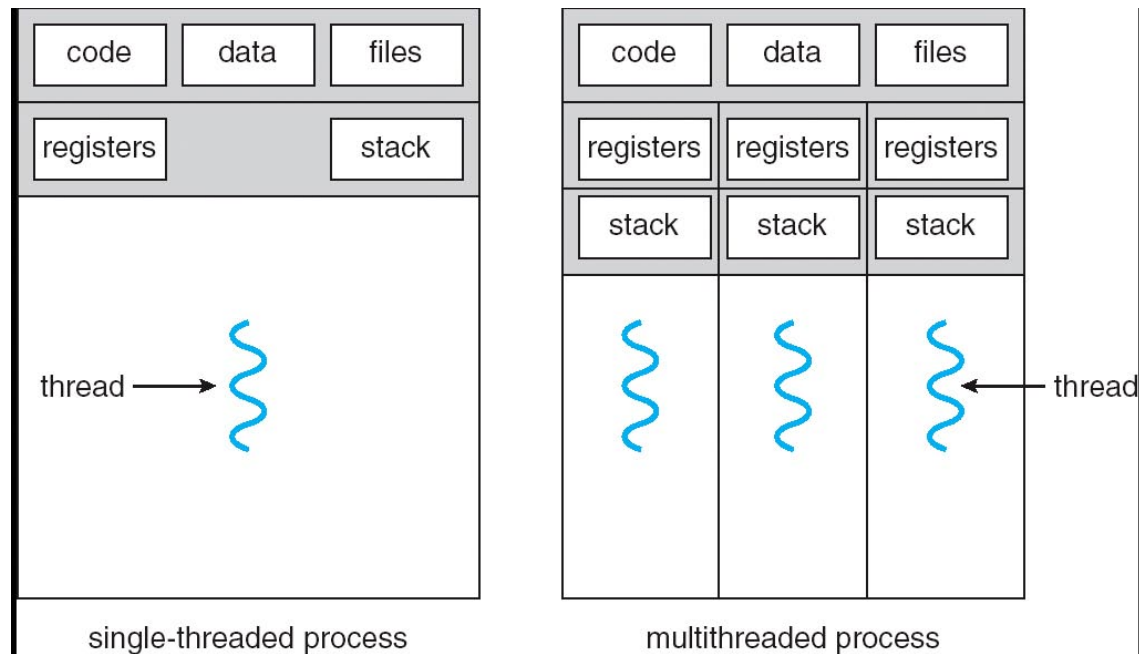


Figure is copyright Silberschatz, Galvin, and Gagne, 2005. (<http://www.os-book.com>)

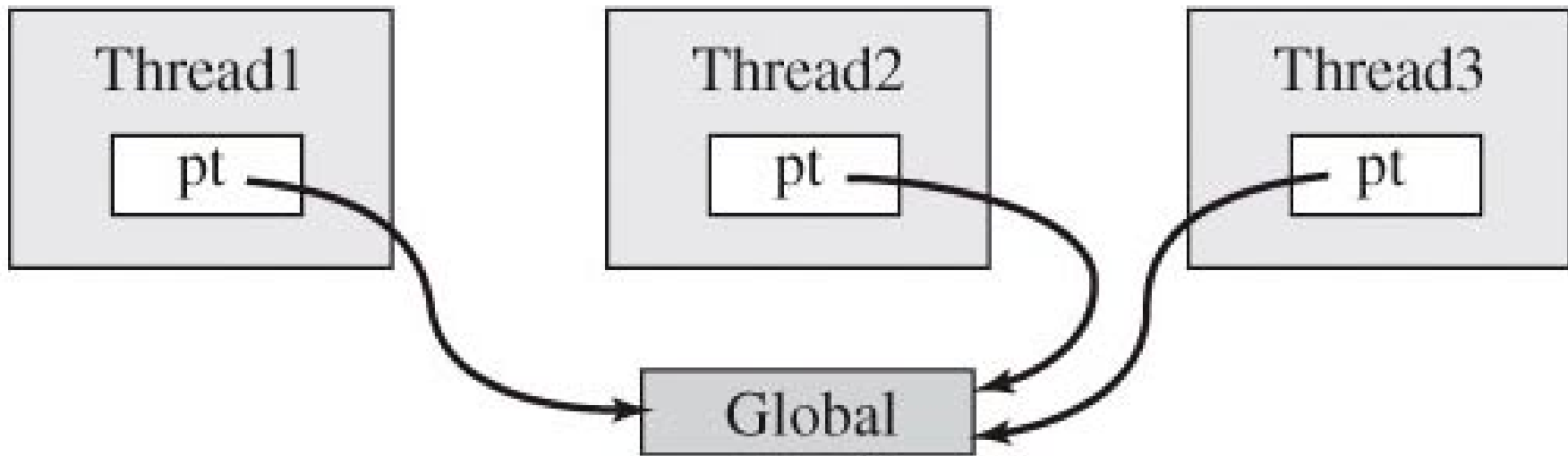
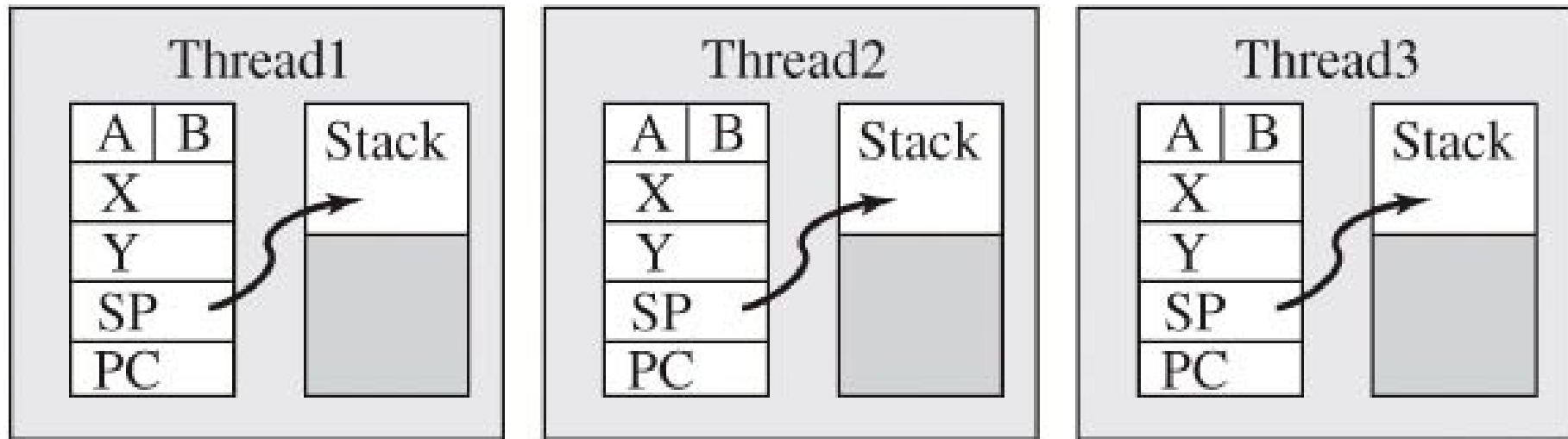
Why use threads?

Can improve program responsiveness.

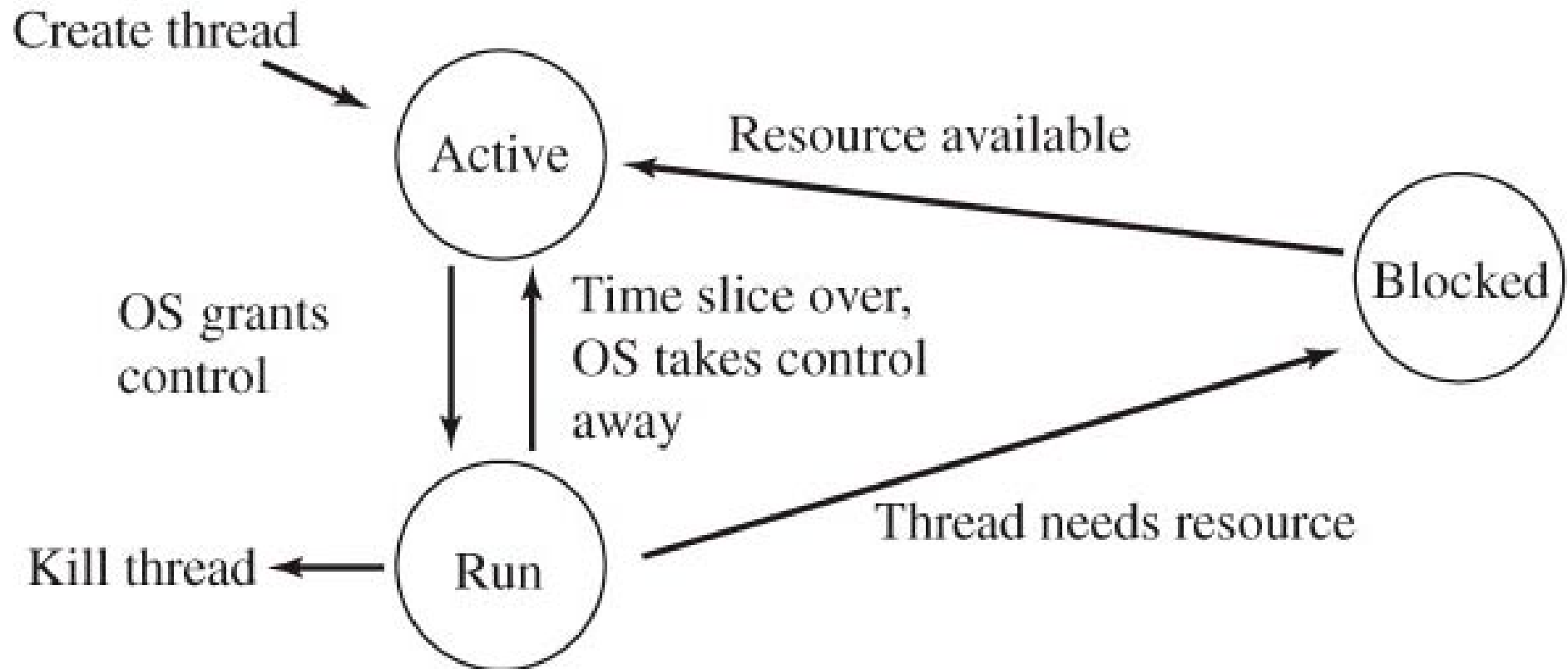
Can improve program modularity through decoupling.

Blocking is very convenient for programmers.

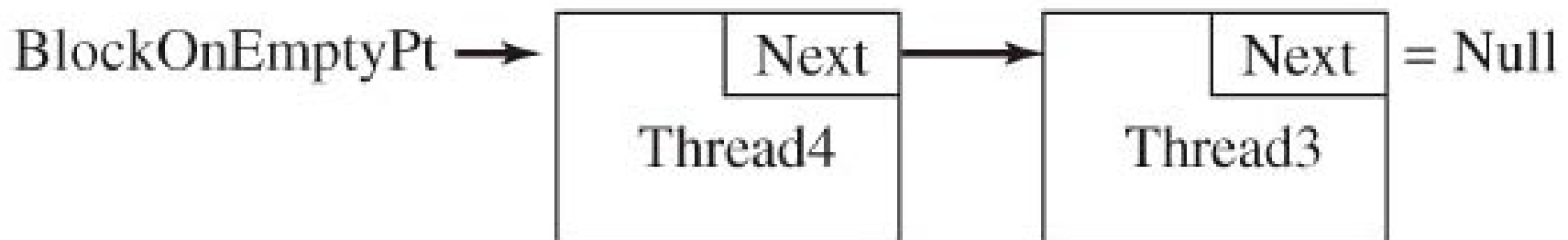
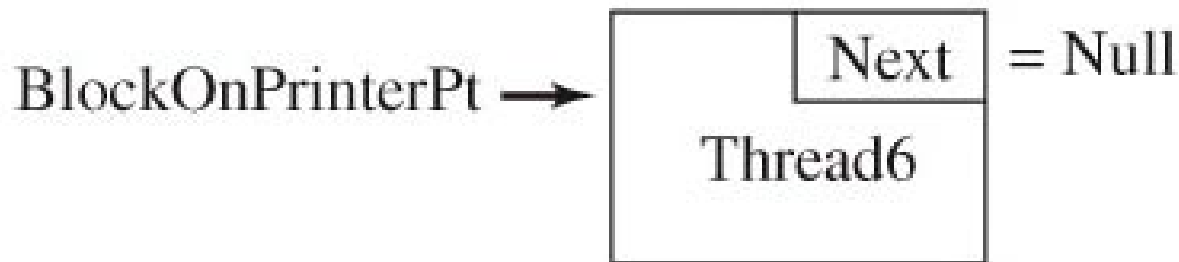
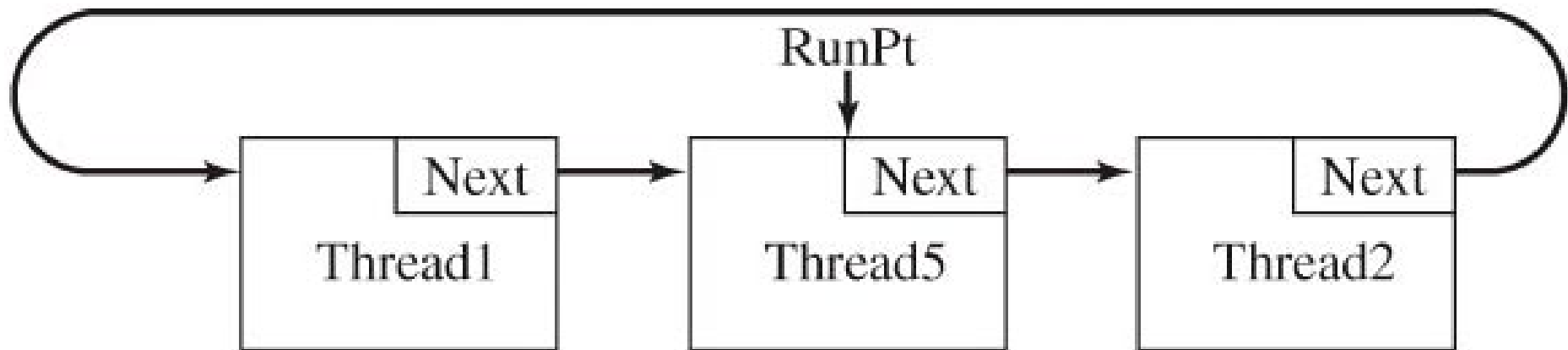
Thread Memory



State Machine for a Single Thread



Thread Lists



BlockOnFullPt = Null

Scheduling

Scheduling is the process by which the system determines which thread to run next.

Scheduling decisions happen when threads change state (Run → Blocked, etc.) or are created or deleted.

Nonpreemptive scheduling is when the scheduler only chooses a new thread/process after the current one terminates or blocks.

Preemptive scheduling is when the scheduler may choose a new thread/process even though the current one is Active.

Preemptive schedulers can result in more responsive systems but require more effort from the programmer to create a correct system.

Scheduling Metrics

CPU utilization.

Throughput.

Turnaround time.

Waiting time.

Response time.

Scheduler Types

First-Come, First-Served.

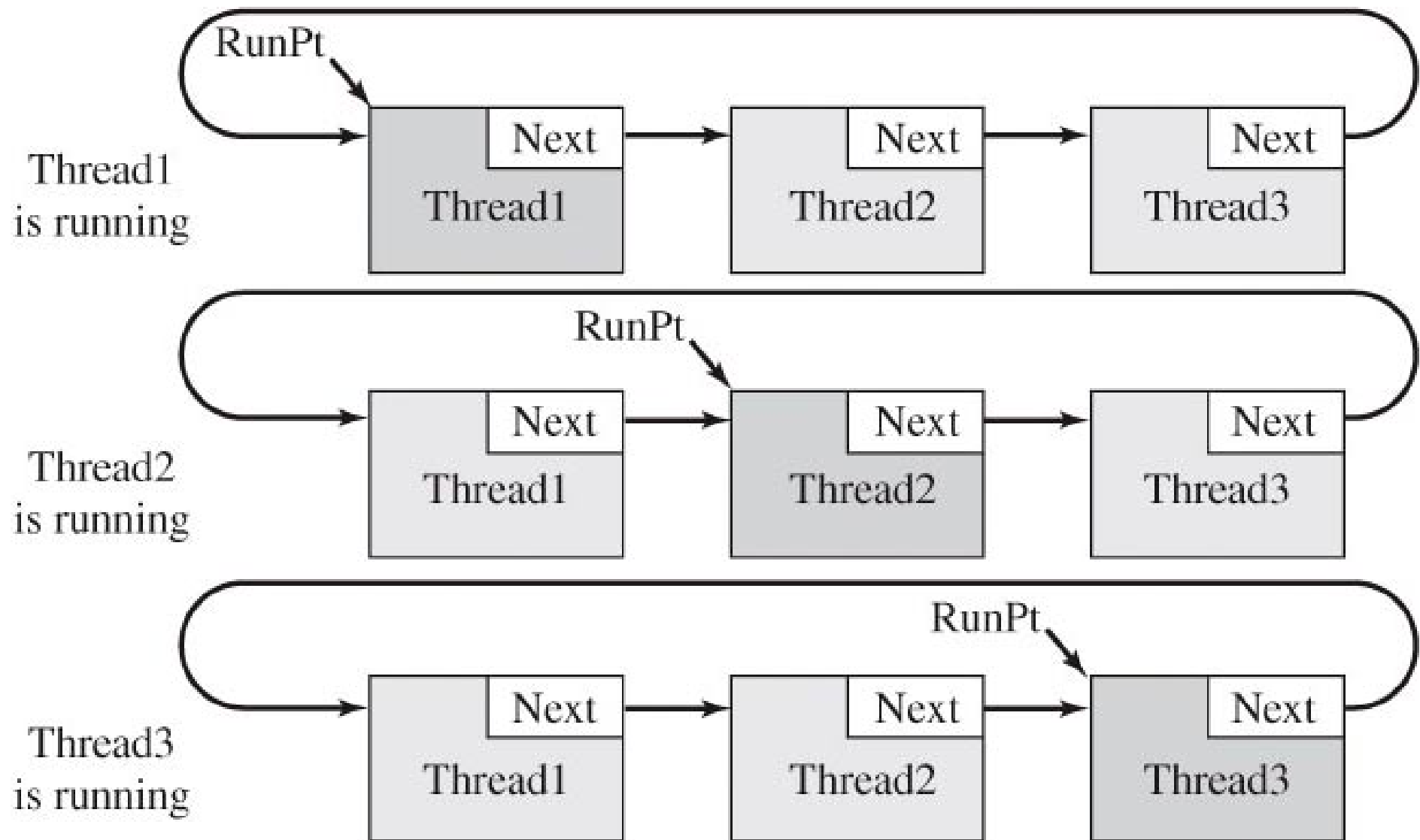
Shortest-Job-First.

Priority.

Round-Robin.

Multi-level Queue and Variants.

Round-Robin Scheduler



Thread Control Block

A *thread control block* (TCB) stores information for managing each thread, and it must contain:

- A pointer so that it can be chained into a linked list.

- The value of its stack pointer.

- A stack area for local variables and saved registers.

A TCB may also contain:

- Thread number, type, or name.

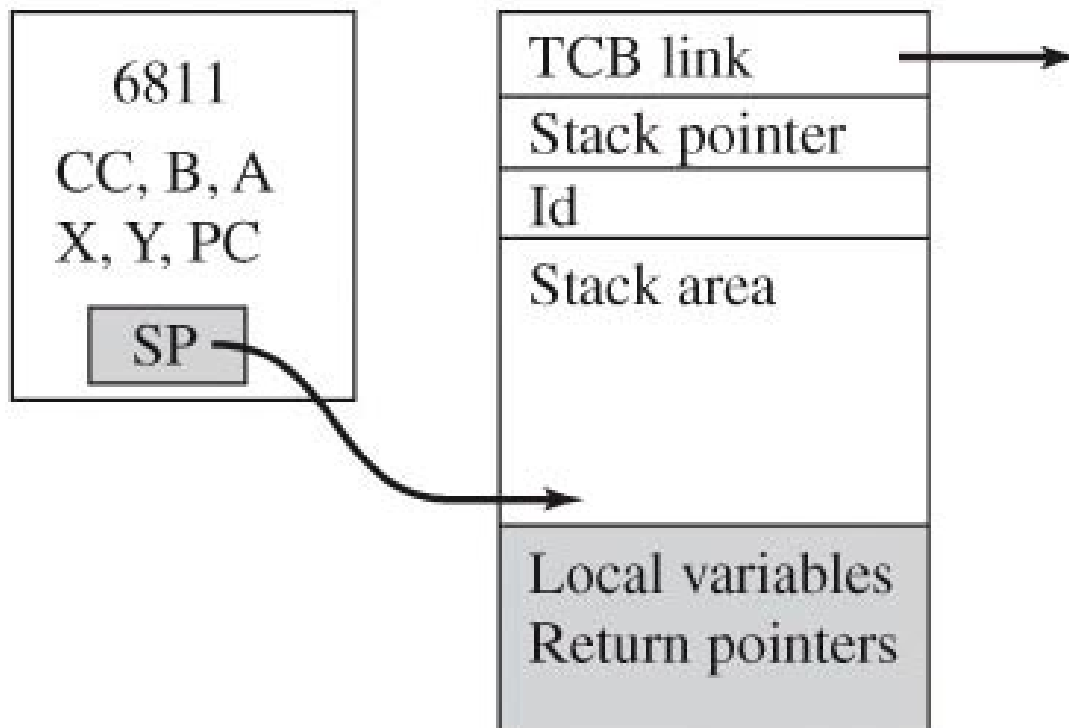
- Age, or how long this thread has been active.

- Priority.

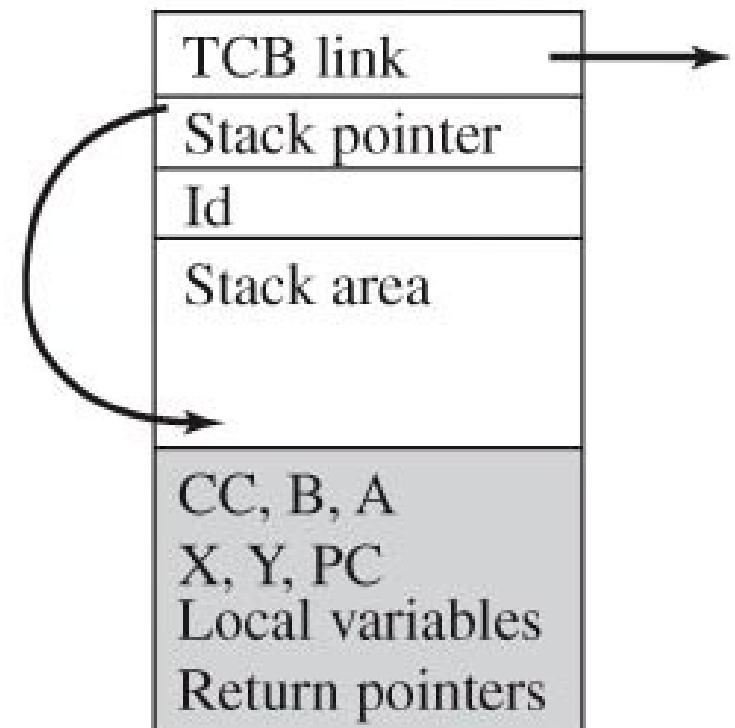
- Resources that this thread has been granted.

Thread Registers

TCB of a running thread



TCB of a thread not running



Code Running in Threads

```
int Sub(int j) { int i;  
    PTM = 1;  // Port M  
    i = j+1;  
    return(i); }  
void ProgA() { int i;  
    i=5;  
    while(1) {  
        PTM = 2;  
        i = Sub(i); }}  
void ProgB() { int i;  
    i=6;  
    while(1) {  
        PTM = 4;  
        i = Sub(i); }}
```

Thread Control Block in C

```
struct TCB
{
    struct TCB *Next;          /* Link to Next TCB */
    unsigned char *SP;         /* Stack Pointer when idle */
    unsigned short Id;        /* output to PortT */
    unsigned char MoreStack[49]; /* more stack */
    unsigned char CCR;        /* Initial CCR */
    unsigned char RegB;       /* Initial RegB */
    unsigned char RegA;       /* Initial RegA */
    unsigned short RegX;      /* Initial RegX */
    unsigned short RegY;      /* Initial RegY */
    void (*PC)(void);         /* Initial PC */
};

typedef struct TCB TCBType;
typedef TCBType * TCBPtr;
```

Thread Control Block in C

```
TCBType sys[3]={
  { &sys[1],          /* Pointer to Next */
    &sys[0].CCR,      /* Initial SP */
    1,                /* Id */
    { 0},
    0x40,0,0,0,0,    /* CCR,B,A,X,Y */
    ProgA },         /* Initial PC */
  { &sys[2],          /* Pointer to Next */
    &sys[1].CCR,      /* Initial SP */
    2,                /* Id */
    { 0},
    0x40,0,0,0,0,    /* CCR,B,A,X,Y */
    ProgA },         /* Initial PC */
  { &sys[0],          /* Pointer to Next */
    &sys[2].CCR,      /* Initial SP */
    4,                /* Id */
    { 0},
    0x40,0,0,0,0,    /* CCR,B,A,X,Y */
    ProgB } };
```

Preemptive Thread Scheduler in C

```
TCBPtr RunPt;          /* Pointer to current thread */
void main(void) {
    DDRT = 0xFF;        /* Output running thread on Port T */
    DDRM = 0xFF;        /* Output running program on Port M */
    RunPt = &sys[0];    /* Specify first thread */
    asm sei
    TFLG1 = 0x20;      /* Clear C5F */
    TIE = 0x20;        /* Arm C5F */
    TSCR1 = 0x80;      /* Enable TCNT*/
    TSCR2 = 0x01;      /* 2MHz TCNT */
    TIOS |= 0x20;      /* Output compare */
    TC5 = TCNT+20000;
    PTT = RunPt->Id;
    asm ldx RunPt
    asm lds 2,x
    asm cli
    asm rti
} /* Launch First Thread */
```

Preemptive Thread Scheduler in C (cont)

```
void interrupt 13 ThreadSwitch() {
    asm ldx RunPt
    asm sts 2,x
    RunPt = RunPt->Next;
    PTT = RunPt->Id;          /* Porth=active thread */
    asm ldx RunPt
    asm lds 2,x
    TC5 = TCNT+20000;        /* Thread runs for 10 ms */
    TFLG1 = 0x20; }        /* ack by clearing C5F */
```

Dynamic Allocation of Threads

```
int create(void (*startFunc)(void), int TheId) {
    TCBPtr NewPt;        // pointer to new thread control block
    NewPt = (TCBPtr)malloc(sizeof(TCBType)); // new TCB
    if(NewPt==0)return FAIL;
    NewPt->SP = &(NewPt->CCR); /* Stack Pointer when not running */
    NewPt->Id = TheId;        /* Visualize active thread */
    NewPt->CCR = 0x40;        /* Initial CCR, I=0 */
    NewPt->RegB = 0;          /* Initial RegB */
    NewPt->RegA = 0;          /* Initial RegA */
    NewPt->RegX = 0;          /* Initial RegX */
    NewPt->RegY = 0;          /* Initial RegY */
    NewPt->PC = startFunc;    /* Initial PC */
    if(RunPt) {
        NewPt->Next = RunPt->Next;
        RunPt->Next = NewPt;} /* will run Next */
    else
        RunPt = NewPt;        /* the first and only thread */
    return SUCCESS;
}
```

Summary

This was the implementation side of a very simple thread system

It is not that hard!

A preemptive threading system is the core of an RTOS

Designing correct embedded code that uses threads is the hard part