Nelms Institute for the Connected World
UNIVERSITY of FLORIDA

Lab 4

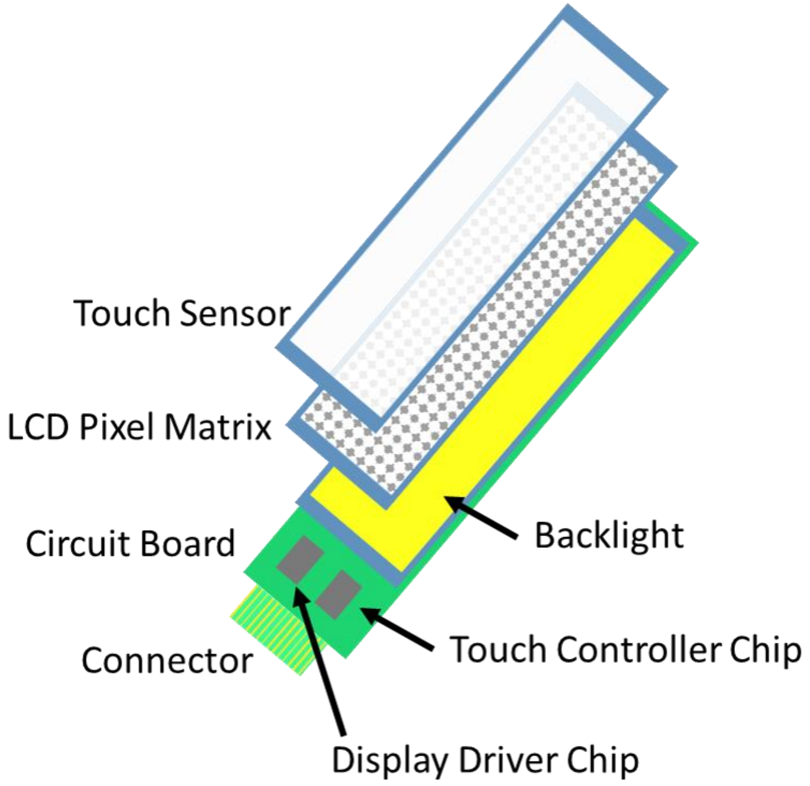
THREAD PRIORITY, DYNAMIC THREAD CREATION AND DESTRUCTION, APERIODIC EVENTS, AND INTERFACING WITH AN LCD

OBJECTIVES

- Write a extended library to interact with LCD touchscreen.
- Write functions that allow to dynamically create and destroy threads.
- Incorporate aperiodic event threads in previous RTOS.
- Convert the round-robin scheduler into a priority schedulers.

REQUIRED

- More Hardware
  - Sensors Booster Pack
  - HY28B Colorful LCD Touchscreen
- Software
  - Lab 3 G8RTOS
  - Board Support Package



PART A

- HY28B Resistive Touchscreen.
  - ILI9325 LCD controller. (Embedded in your LCD screen)
  - XPT2046 Touchscreen controller. (Embedded in your LCD screen)
  - Library file template provided on Canvas.

https://os.mbed.com/components/HY28B-28-Touch-Screen-TFT-LCD-SPI-8-16-b/

https://www.arduino.cc/en/Guide/TFT

https://www.buydisplay.com/download/ic/XPT2046.pdf

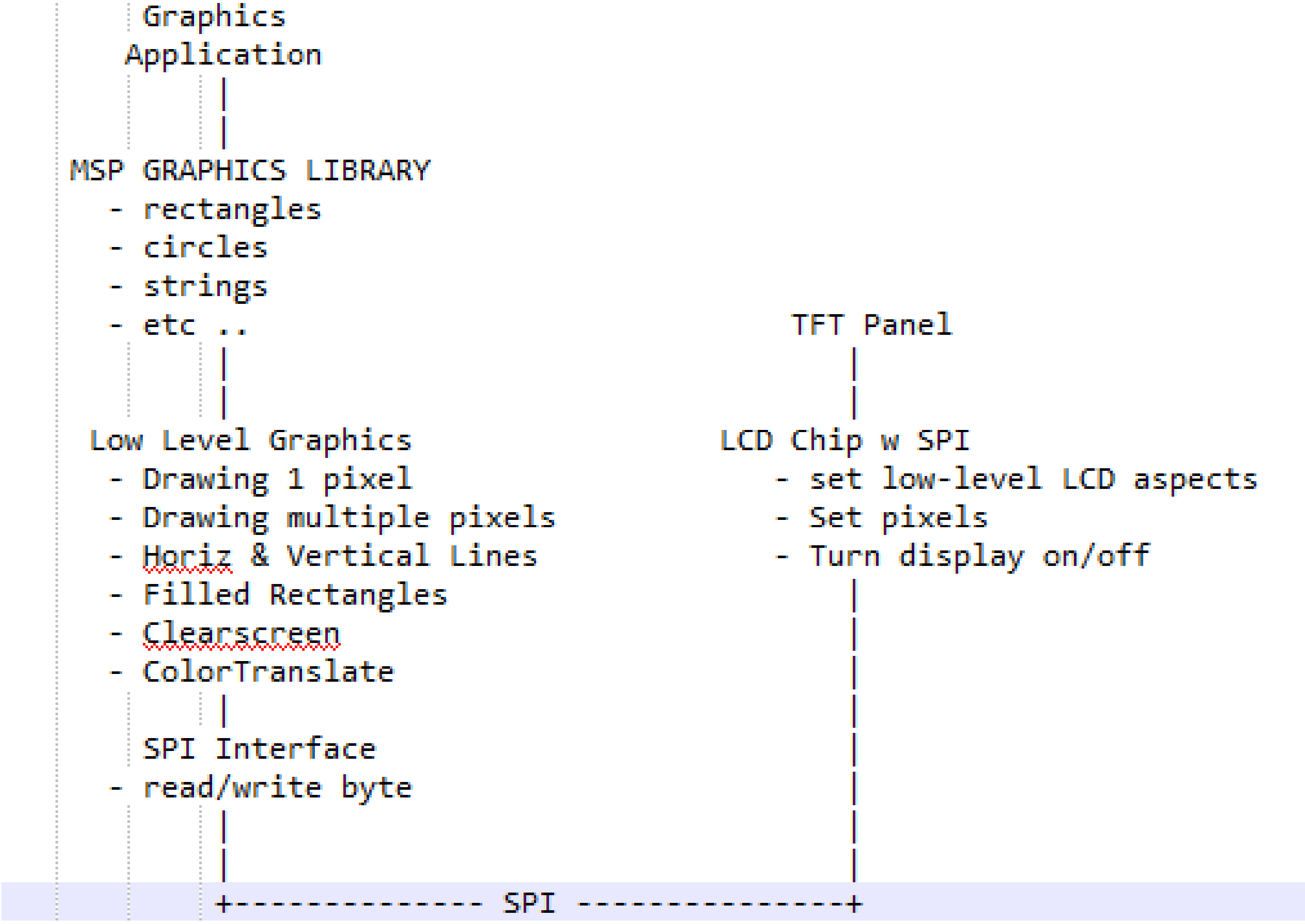
http://www.haoyuelectronics.com/Attachment/HY28B/ILI9325C%20datasheet.pdf

PART A

- SPI Configuration/Connection
  - Use `P10SEL` register to configure the SPI function
  - P10.1 CLK
  - P10.2 MOSI
  - P10.3 MISO
  - SPI configuration
    - 3 Pin, 8 bit SPI master, high polarity for inactive state, 12MHz
  - P10.4 LCD CS
  - P10.5 TP CS

PART A

- Software design model



```
    Graphics
   Application
        |
        |
MSP GRAPHICS LIBRARY
  - rectangles
  - circles
  - strings
  - etc ..                              TFT Panel
        |                                  |
        |                                  |
 Low Level Graphics                 LCD Chip w SPI
  - Drawing 1 pixel                    - set low-level LCD aspects
  - Drawing multiple pixels            - Set pixels
  - Horiz & Vertical Lines             - Turn display on/off
  - Filled Rectangles                      |
  - Clearscreen                            |
  - ColorTranslate                         |
        |                                  |
    SPI Interface                          |
  - read/write byte                        |
        |                                  |
        |                                  |
        +-------------- SPI ---------------+
```

PART A

- Function provided
  - `LCD_Init`
    - Initializes the LCD hardware, remember to initializes the SPI peripheral.
  - `PutChar`
    - Put a character to specified location/coordinate.
  - `LCD_Text`
    - Put a string to specified location/coordinate.
  - `LCD_WriteIndex`
    - Set the address of register we want to write to

- Function provided
  - `LCD_WriteData`
    - Write 16 bit data to the register which specified by `LCD_WriteIndex`
  - `LCD_ReadData`
    - Read 16 bit data to the register which specified by `LCD_WriteIndex`
  - `LCD_Write_Data_Start`
    - Send out the starting condition of continuous data

PART A

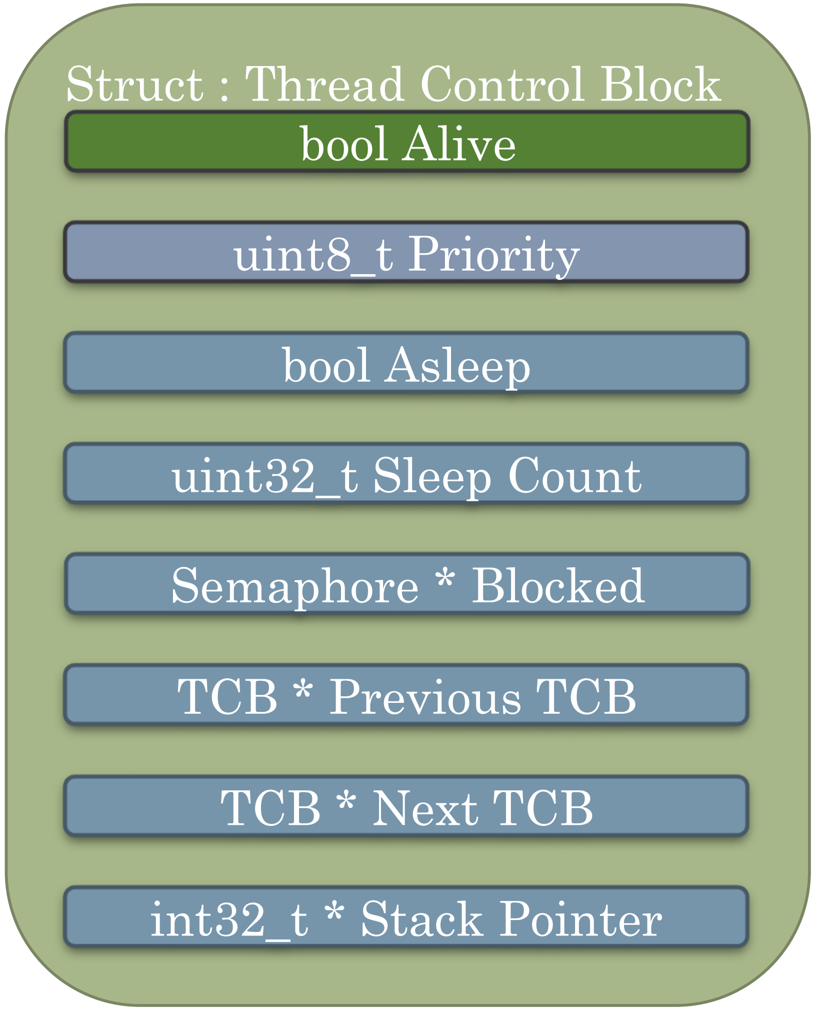
- Function you write
  - **LCD_initSPI**
  - **SPISendRecvByte**
  - **TP_ReadXY**
  - LCD_DrawRectangle
  - LCD_Clear
  - LCD_SetPoint
  - LCD_Write_Data_Only
  - LCD_ReadReg
  - LCD_WriteReg
  - LCD_SetCursor

PART A

- `LCD_initSPI`
  - Initialize the SPI peripheral with predefined parameters
  - 3 Pins, 8bit SPI master, and 12MHz
- `SPISendRecvByte`
  - Interface to send and receive data with SPI
  - You can use `SPI_transmitData` and `SPI_receiveData` from DriveLib
- `TP_ReadXY (XPT2046 Page 22, Differential Mode)`
  - TP_ReadX: SPI Command `CHX`
  - TP_ReadY: SPI Command `CHY`

PART B

- Priority Scheduler
  - Bool Alive
  - Uint8_t Priority
- Guarantee 30fps LCD refresh



Struct : Thread Control Block

bool Alive

uint8_t Priority

bool Asleep

uint32_t Sleep Count

Semaphore * Blocked

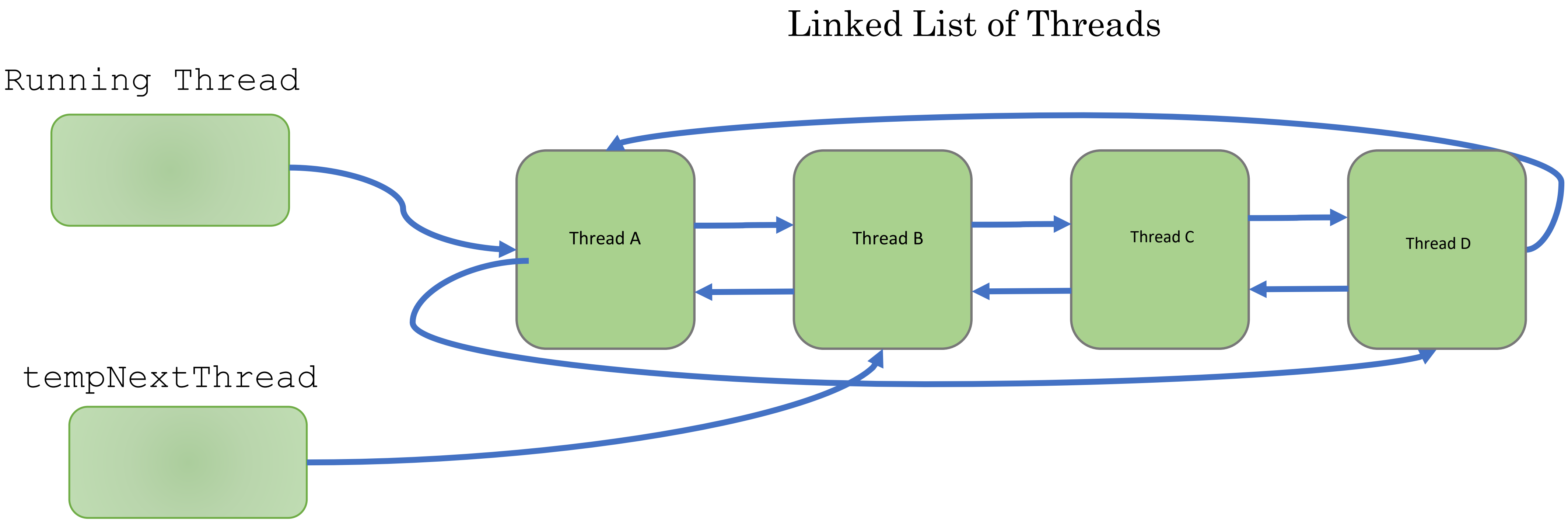
TCB * Previous TCB

TCB * Next TCB

int32_t * Stack Pointer

PART B

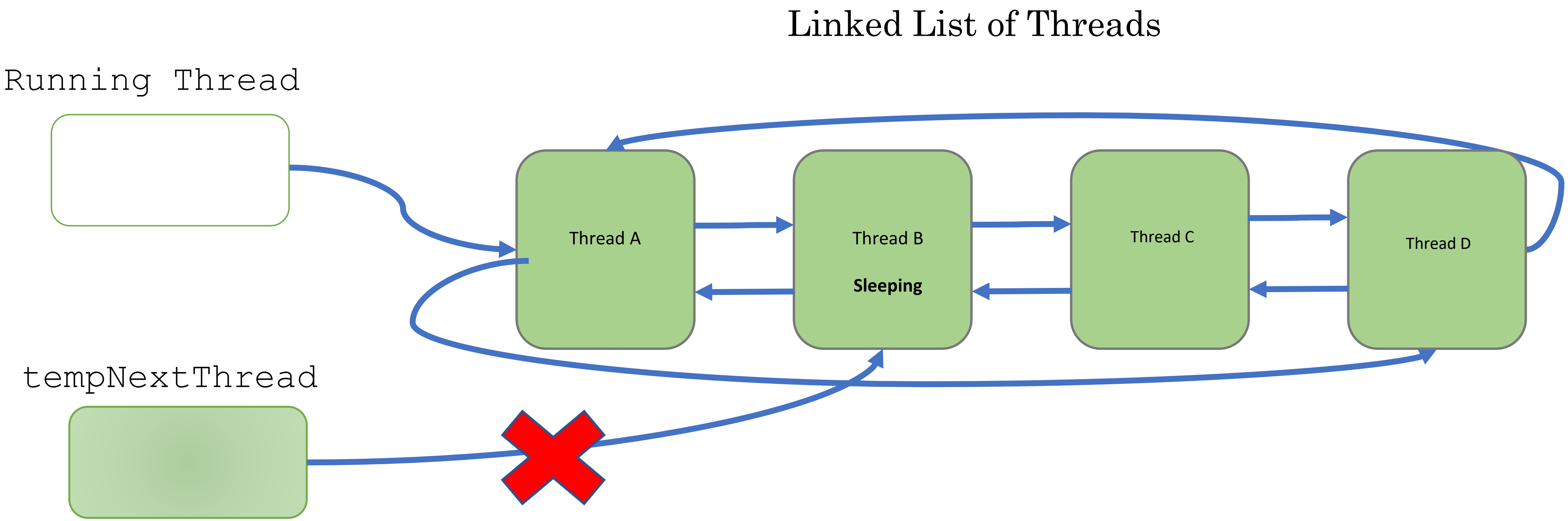
- Priority Scheduler



UF Nelms Institute for the Connected World
UNIVERSITY of FLORIDA

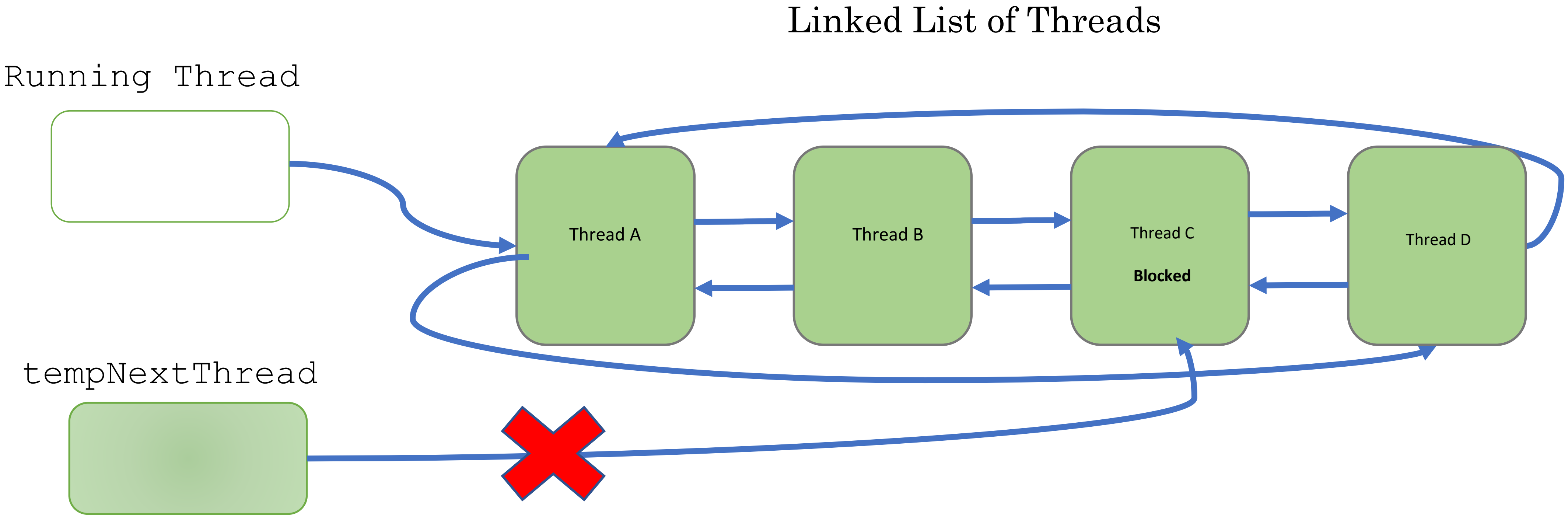
PART B

- Priority Scheduler
  - Not sleeping



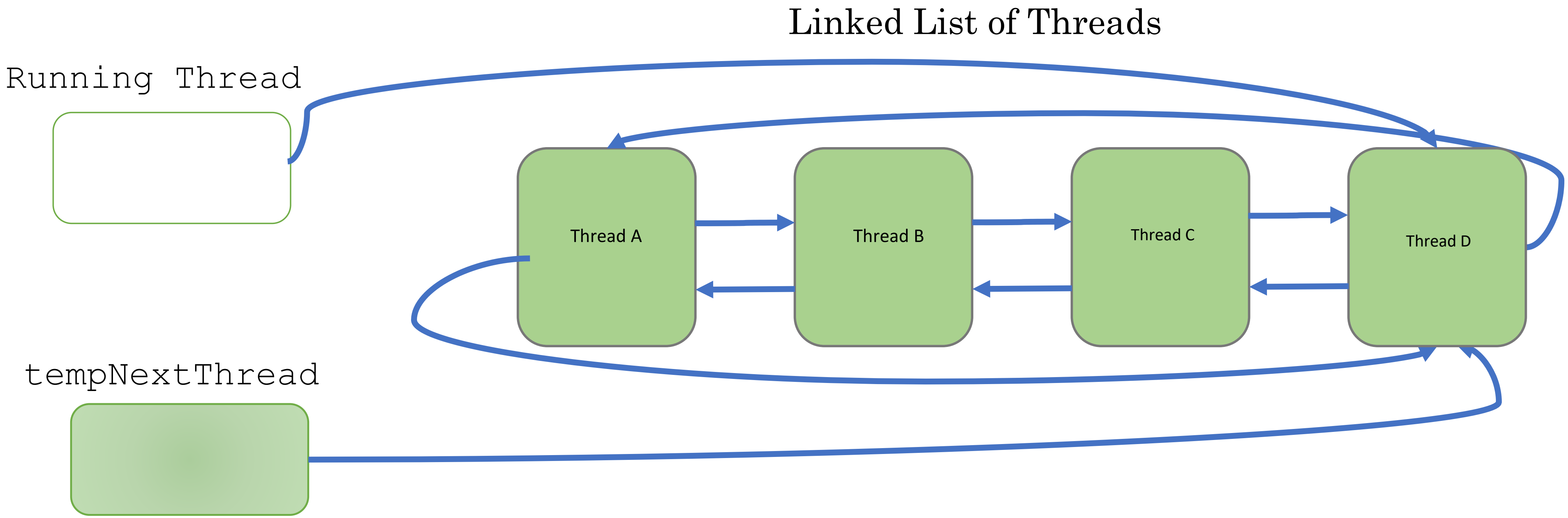
PART B

- Priority Scheduler
  - Not blocked



PART B

- Priority Scheduler
    - Check Priority



PART B

- Priority check pseudo code

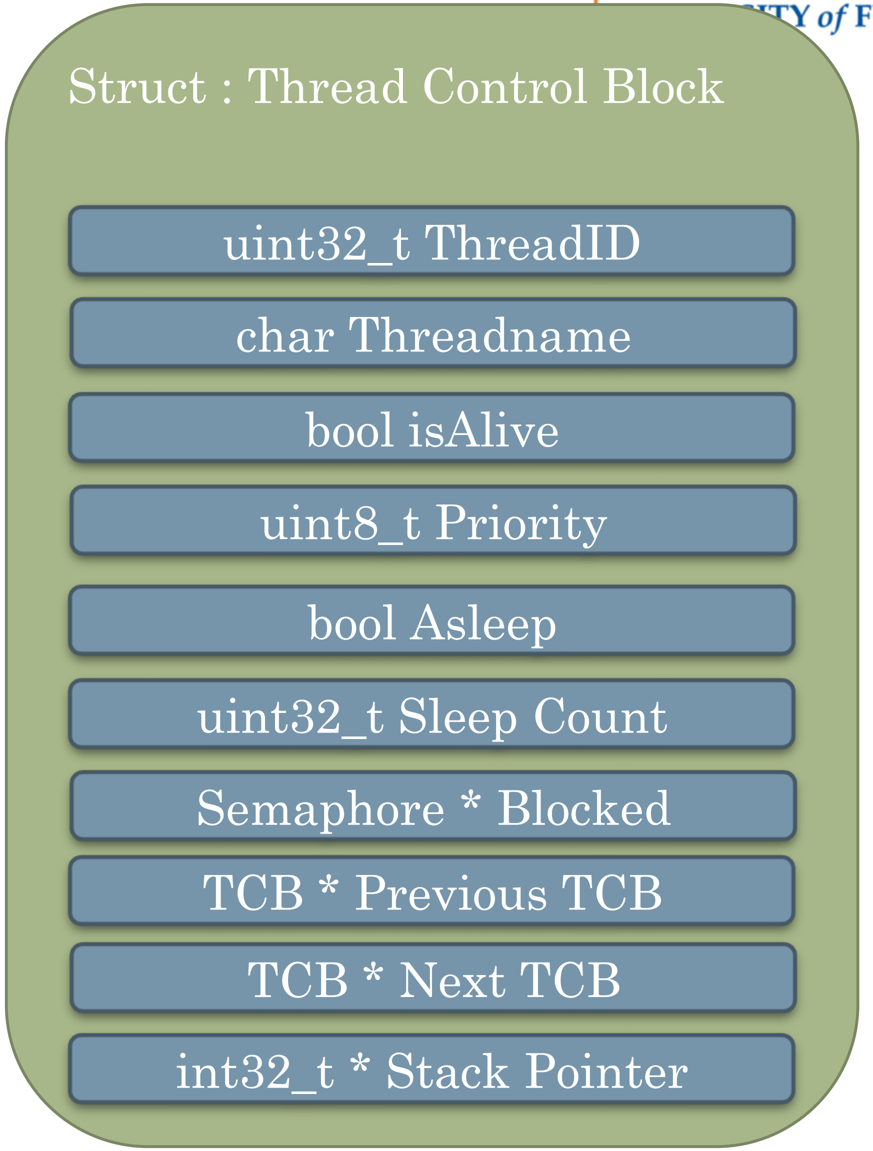
```
/* Priority of potential next thread to run */
uint8_t nextThreadPriority = UINT8_MAX;

for(loop)
{
    /* Check if Thread is blocked or asleep */
    if !nextThread.issleep() && !nextThread.isblocked()
    {
        /* Check if priority is higher than current max */
        if nextThread.Priority less than nextThreadPriority
        {
            /* Set CurrentlyRunning thread to the next thread to run */
            CurrentlyRunningThread = nextThread
            nextThreadPriority = CurrentlyRunningThread.Priority;
        }
    }

    nextThread = nextThread.nextTCB;
}
```

PART C

- Thread related improvement
  - Dynamic thread creation and destruction
  - Modification of `AddThread`
  - New function `KillThread`
  - New function `GetThreadId`
  - New function `KillSelf`



Struct : Thread Control Block

uint32_t ThreadID

char Threadname

bool isAlive

uint8_t Priority

bool Asleep

uint32_t Sleep Count

Semaphore * Blocked

TCB * Previous TCB

TCB * Next TCB

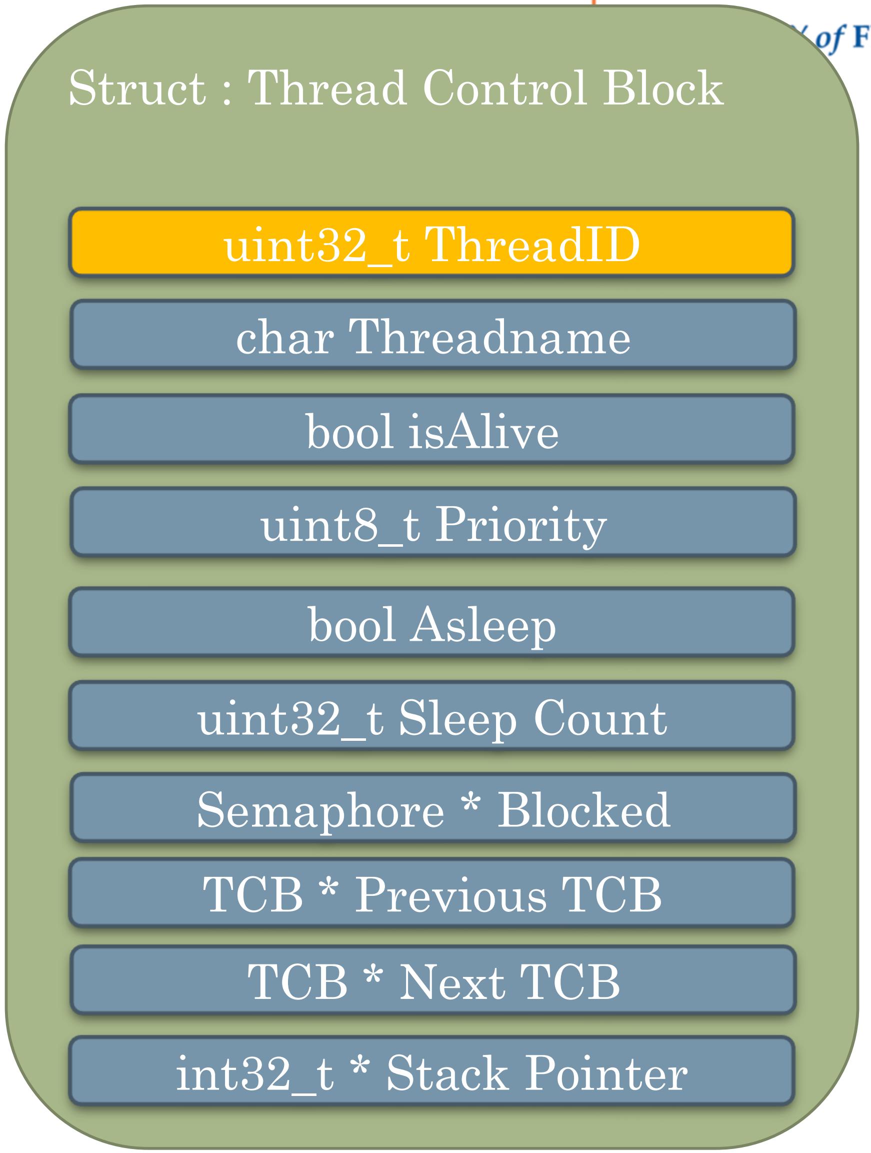
int32_t * Stack Pointer

PART C

- Modification of `AddThread()`
- Parameters
  - `void (*threadToAdd)(void), uint8_t priority, char * name`
- Routine
  - Enter critical section
  - Is there any more available slot for new thread? `NumberOfThreads`
  - Is there any dead thread so we can replace it? `isAlive`
  - Initialize the thread control block
    - Stack pointer, blocked, sleep, isalive, threadID, threadName, priority, etc.
    - threadID? `((IDCounter++) << 16) | tcbToInitialize;`
  - Leave critical section

PART C

- New function `GetThreadId`
  - Returns the CurrentlyRunningThread’s thread ID.
  - Easy to do.
  - `CurrentRunningThread->ThreadID`



Struct : Thread Control Block

uint32_t ThreadID

char Threadname

bool isAlive

uint8_t Priority

bool Asleep

uint32_t Sleep Count

Semaphore * Blocked

TCB * Previous TCB

TCB * Next TCB

int32_t * Stack Pointer

PART C

- New function `KillThread`
  - Take in a threadId, indicating the thread to kill.
- Parameters
  - `threadId_t threadId`
- Routine
  - Enter a critical section
  - Return appropriate error code if there’s only one thread running
  - Search for thread with the same threadId
  - Return error code if the thread does not exist
  - Set the threads isAlive bit to false
  - Update thread pointers
  - If thread being killed is the currently running thread, we need to context switch once critical section is ended
  - Decrement number of threads
  - End critical section

PART C

- New function `KillSelf`
    - Simply kill the currently running thread
- Routine
    - Enter a critical section
    - If only 1 thread running, return appropriate error code
    - Change isAlive bit to false
    - Update thread pointers
    - Start context switch
    - Decrement number of threads
    - End critical section

PART D

- Aperiodic Event Threads
- Definition
  - An event thread with an arrival pattern that lacks a bounded minimum interval between subsequent instances.
- How do we implement it?
  - Essentially be an interrupt routine
  - Nested Vectored Interrupt Controller (NVIC)
  - Initialize the appropriate NVIC registers accordingly

PART D

- Aperiodic Event Threads
- Parameters
  - `void (*AthreadToAdd)(void), uint8_t priority, IRQn_Type IRQn`
- Routine
  - Verify the IRQn is less than the last exception (PSS_IRQn) and greater than last acceptable user IRQn (PORT6_IRQn), or else return appropriate error
  - Verify priority is not greater than 6, the greatest user priority number, or else return appropriate error
  - Use the following core_cm4 library functions to initialize the NVIC registers
    - `__NVIC_SetVector`
    - `__NVIC_SetPriority`
    - `NVIC_EnableIRQ`

PART D

- Aperiodic Event Threads
- Attention
  - To relocate the ISR interrupt vector, the interrupt vector table should be relocated into SRAM. Thus, you should put the following code snippet into the RTOS initialization function.

```
// Relocate vector table to SRAM to use aperiodic events
    uint32_t newVTORTable = 0x20000000;
    memcpy((uint32_t *)newVTORTable, (uint32_t *)SCB->VTOR, 57*4);
// 57 interrupt vectors to copy
    SCB->VTOR = newVTORTable;
```

Demonstration

- Program will launch with nothing on the screen, waiting for a touch on the screen.
- Once touched, a ball (4x4 rectangle in our case) should be drawn on the screen with a random color .
- Depending on the accelerometer x and y values, the ball will move accordingly.
- Every new ball created should have a random speed.
- If one of the balls is touched, you should delete the ball.
- There will be a max number of 20 balls allowed at one time.
- If a ball hits an edge, it should wrap around to the other side.