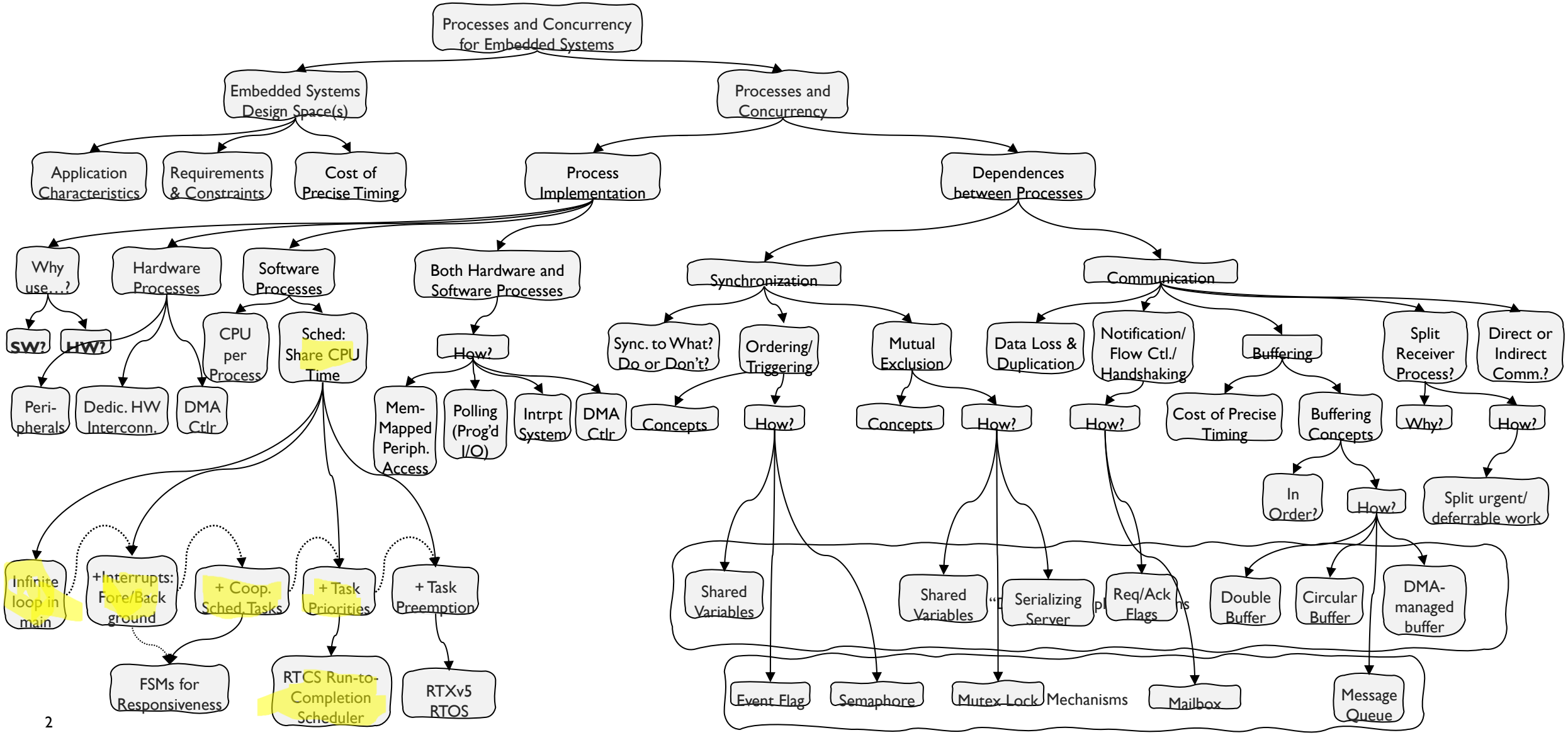
07: Turning the while(1) loop into RTCS: Run-to-Completion Non-Preemptive Prioritized Scheduler

v2

Extended Topic Map: Class 07



Applications: Functionality First, then Performance

			Blinky Control Panel	Quadrature Decoder w/Limit Switch	Waveform Generator	LCD Controller	Touch screen	Scope	Serial Comms.	I^2C Comms.	µSD via SPI Comms.	SMPS Controller
Providing Functionality	Interfacing	Simple Digital	In, Out	In			Out					PWM Out
		Complex Digital	PWM			Bus Out			In, Out	In, Out	In, Out	
		Analog	ADC In, CMP In, DAC Out		Out		ADC In	ADC In, Cmp In				Sync. ADC In Sampling
	Sync and …	Sync and Do: Coarse Triggering		Digital Edge Detection	Periodic Output Updates			Analog Edge Det., Data buffer mgt.	Tx Ready, Rx Done events. Producer & Consumer	Data Producers & Consumers. I2C Device read response.	Tx Ready, Rx Done events. Producers & Consumers	Sync. ADC In Sampling
		Internal, Fine Grain Block/Sched/Trig					ADC conversion time			Internal timing reqts. for processing data bytes, Start/Stop		
		Sync and Don't: Sharing & Races						LCD Ctlr Sharing, Data buffer mgt.	Tx, Rx Qs	Tx, Rx Msgs	Tx, Rx Qs	
	IPC	Inter-Process Comm.		Shared Position Variable				Data buffer				
Meeting Perf. Reqts.		Timing Stability	1		1			2				1
		Responsiveness	1					1			1	1
		Reducing SW Overhead			2	1- Perf. Optimiz.		2		1 – series of timed I/O events per message. FSM vs. RTOS		2
		Tolerating Timing Mismatches			2			2	2		2	

Example Applications: Key Requirements and Challenges

Application	Providing Functionality					Meeting Performance Requirements			
	Digital Interfacing	Analog Interfacing	Sync and Do: Triggering	Sync and Don't: Sharing & Races	Inter-Process Comm.	Timing Stability	Responsiveness	Reducing Software Overhead	Tolerating Timing Mismatches
Blinky Control Panel	In, Out, PWM								
Quadrature Decoder	In								
Waveform Generator		Out							
Oscilloscope		In							
Serial Comms.	In, Out								
I2C Comms.	In, Out								
LCD Controller	Out								
Touchscreen		In							
SMPS Controller	Out	In							
µSD via SPI Comms.	In, Out								

Improving the Infinite Loop Scheduler (Cyclic Executive)

- Features
  - Helps modularity by supporting tasks
  - Gives all tasks a chance to run (round-robin)
- Limitations
  - Task function name is hard-coded in scheduler function – *separation of functionality*
  - Task makes decision whether to do its work – *separation of functionality*
  - All tasks have same priority, run in fixed order
  - All tasks run at same rate
    - Each gets one chance to do its work per scheduler loop iteration
  - Tasks cannot preempt each other (sequential, not concurrent execution)

```
void Scheduler(void) {
    while (1) {
       Task_Foo();
       Task_Bar();
       Task_Service_UART();
    }

}

void Task_Service_UART(void) {
   if (UART_received_data) {
        // do task work
   }
   return;
}
```

Improving the Cyclic Executive (Infinite Loop Scheduler)

- Starting point
  - Each task returns early if it has no work to do
  - Task is making this run decision

```
void Scheduler(void) {
    while (1) {
       Task_Foo();
       Task_Bar();
       Task_Service_UART();
    }

}

void Task_Service_UART(void) {
   if (UART_received_data) {
        // do task work
   }
   return;
}
```

Basic Scheduler

Task Code
Scheduler Code
Scheduler Data

```
void Scheduler(void) {
   while (1) {
      Task_Foo();
      Task_Bar();
   }
}
```

```
void Task_Foo(void) {
   if (run_Foo > 0) {
      run_Foo--;
      // do the Foo work
   }
}
```

```
void Task_Bar(void) {
   if (run_Bar > 0) {
      run_Bar--;
      // do the Bar work
   }
}
```

```
/* Globals for
   scheduling */
int run_Foo;
int run_Bar;
```

Move Decision to Work from Task into Scheduler

```
void Scheduler(void) {
    while (1) {
        if (run_Foo) {
            run_Foo--;
            Task_Foo();
        }
        if (run_Bar) {
            run_Bar--;
            Task_Bar();
        }
    }
}
```

```
/* Globals for
   scheduling */
int run_Foo;
int run_Bar;
```

```
void Task_Foo(void) {
    // do the Foo work
}
```

```
void Task_Bar(void) {
    // do the Bar work
}
```

Generalize the Scheduler with a Table (1)

```
/* Globals for
   scheduling */
int run_Foo;
int run_Bar;
```

```
typedef struct {
   int ReleasesPending;
   void (*Task)(void);
} TaskTableEntry;

TaskTableEntry Task_Table[3];
```

TaskTable		
	ReleasesPending	Task (Function Pointer)
0	2	Task_Foo
1	1	Task_Bar
2	0	

```
void Task_Foo(void) {
   // do the Foo work
}
```

```
void Task_Bar(void) {
   // do the Bar work
}
```

```
Task_Table[0].Task = Task_Foo;
Task_Table[0].ReleasesPending = 0;
Task_Table[1].Task = Task_Bar;
Task_Table[1].ReleasesPending = 0;
```

- Change run_* to ReleasesPending
  - >0? Need to run (release) the task
  - Count how many run requests have been made but not yet started
- "Task" field is pointer to task's root function

Generalize the Scheduler with a Table (2)

```
void Scheduler(void) {
    int i; // Current entry in task table
    while (1) {
       for (i=0; i<NTASKS; i++) {
           if (Task_Table[i].ReleasesPending > 0) { // If run is requested
              Task_Table[i].ReleasesPending--;      // Decrement request count
              Task_Table[i].Task();                 // Run the task
           }
       }
    }
}
```

	ReleasesPending	Task (Function Pointer)
0	2	Task_Foo
1	1	Task_Bar
2	0	

```
void Task_Foo(void) {
   // do the Foo work
}
```

```
void Task_Bar(void) {
   // do the Bar work
}
```

Example of Scheduler Operation

```
void Scheduler(void) {
    int i; // Current entry in task table
    while (1) {
       for (i=0; i<NTASKS; i++) {
           if (Task_Table[i].ReleasesPending > 0) { // If run is requested
              Task_Table[i].ReleasesPending--;      // Decrement request count
              Task_Table[i].Task();                 // Run the task
           }
       }
    }
}
```

Foo (0) ReleasesPending	2
Bar (1) ReleasesPending	1
Scheduler Activity	
Task Executing	

Timing Interference Between Tasks

- Fair scheduling
  - Scheduler takes turns running ready tasks, giving each one run per scheduling cycle
  - Each task's run is delayed by all other ready tasks

- Replace fairness with prioritization
  - Scheduler runs highest priority ready task
  - Timing interference of lower-priority tasks is mostly eliminated
    - Exception: if lower-priority task is currently running

Add Priorities to Scheduler Operation

```
void Scheduler(void) {
   int i; // Current entry in task table
   while (1) {
      for (i=0; i<NTASKS; i++) {
         if (Task_Table[i].ReleasesPending > 0) { // If run is requested
            Task_Table[i].ReleasesPending--;      // Decrement request count
            Task_Table[i].Task();                 // Run the task
            break;                                // Exit for loop
         }
      }
   }
}
```

- After a task finishes running, have scheduler start at top of table again
- Use break statement to exit for loop

Foo (0) ReleasesPending	2
Bar (1) ReleasesPending	1
Scheduler Activity	
Task Executing	

Summary of Key Scheduler Features (so far)

- Source code in RTCS\RTCS.[ch]
- Finish the current task before you start another
  - "Run to completion" = task only yields control to scheduler at end of its root function.
  - Tasks aren't concurrent (unless we use FSMs).
  - One task can't preempt another task.
  - With these restrictions, we can still call tasks as subroutines, so only need one stack
- Interrupts operate as expected
  - Return control to previously executing function/task
- If there is more than one task to start, run the highest priority task first
  - Use an array of task descriptions (task table)
  - Place highest priority tasks at top
  - Start at *top* of task table whenever looking for a task to run
- If a task is ready to run, then scheduler will run it eventually, unless system is overloaded with work

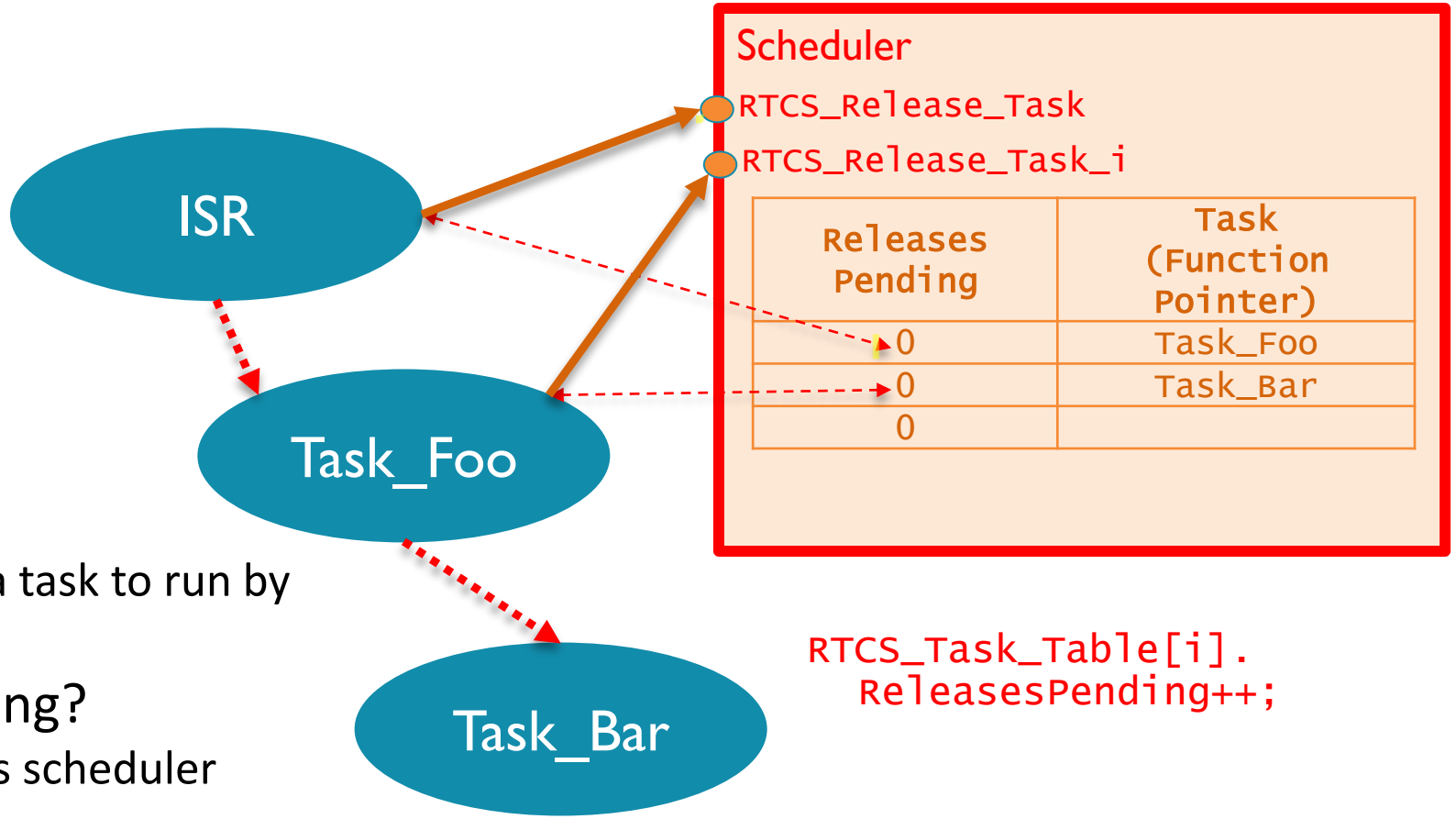
Current Code to Run the Scheduler

```
void RTCS_Run_Scheduler(void){
  int i;
  while (1) {
    // start at top of table to find highest priority ready task
    for (i=0; i<RTCS_NUM_TASKS; i++) {
       if ((RTCS_Task_Table[i].Task != 0) &&
       (RTCS_Task_Table[i].ReleasesPending > 0)) {
          RTCS_Task_Table[i].ReleasesPending--;
          RTCS_Task_Table[i].Task();
          break; // Priority! Exit from for loop, start at top of table again
       }
    }
  }
}
```

- Note: added safety check for null Task pointer

Triggering Tasks with Events

- How can we trigger tasks to run?
  - Who **increments** ReleasesPending?
- Example
  - ISR triggers Task_Foo
  - Task_Foo triggers Task_Bar
- Event-Triggering
  - Tasks and ISRs can request for a task to run by incrementing *ReleasesPending*
- How to access ReleasesPending?
  - Dangerous and sloppy to access scheduler table directly
  - Instead access through an API call (adds range checks, other benefits)
    - RTCS_Release_Task, RTCS_Release_Task_i



API for Triggering Tasks

```
int RTCS_Release_Task_i(int i) {
  if (i < RTCS_NUM_TASKS) {
    if (RTCS_Task_Table[i].Task != 0) {
        RTCS_Task_Table[i].ReleasesPending++;
        return 1;
    }
  }
  return 0;
}
```

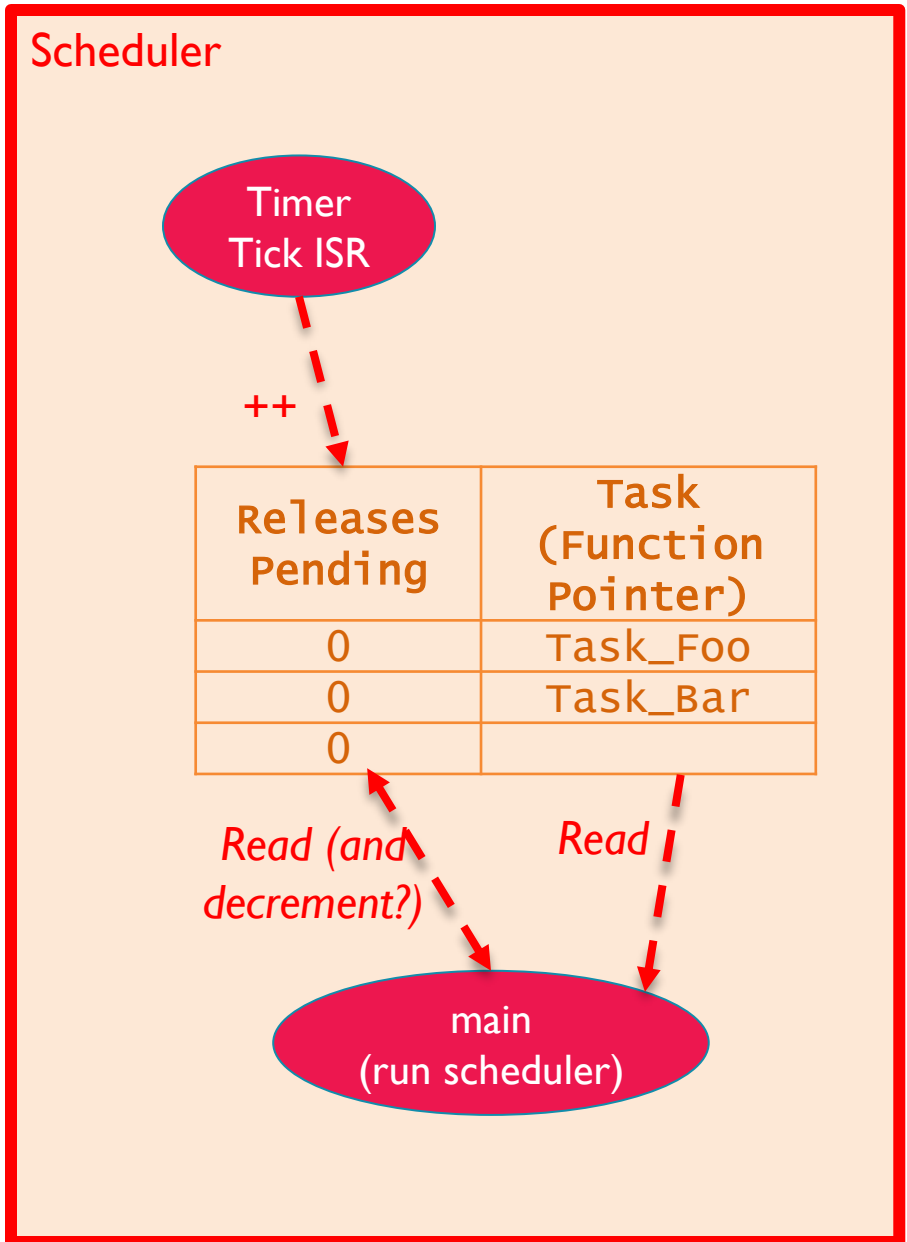
- Can specify task by table location or by function pointer

```
int RTCS_Find_Task_Priority(void (*task)(void)) {
  int i;
  for (i=0; i<RTCS_NUM_TASKS; i++) {
    if (RTCS_Task_Table[i].Task == task) {
        return i;
    }
  }
  return -1;
}
```

```
int RTCS_Release_Task(void (*task)(void)) {
  int i;
  i = RTCS_Find_Task_Priority(task);
  if ((i >= 0) && (i<RTCS_NUM_TASKS)) {
        RTCS_Task_Table[i].ReleasesPending++;
        return 1;
  }
  return 0;
}
```

Adding Periodic Triggering for Tasks

- Current solution requires task or ISR to trigger each task run (release)
  - Awkward, limited
- Let's add time-based scheduling triggers
  - "Run this task every 18 ms starting ASAP"
  - Could also do
    - "Run this task after 13 ms, and then every 13 ms."
- Use a periodic scheduler tick function
  - Use hardware timer to trigger ISR every 1 ms
  - Scheduler tick function
    - Tracks elapsed time
    - Increments a task's *ReleasesPending* count if it's time to run the task
- Remember, scheduler will run any tasks with ReleasesPending > 0

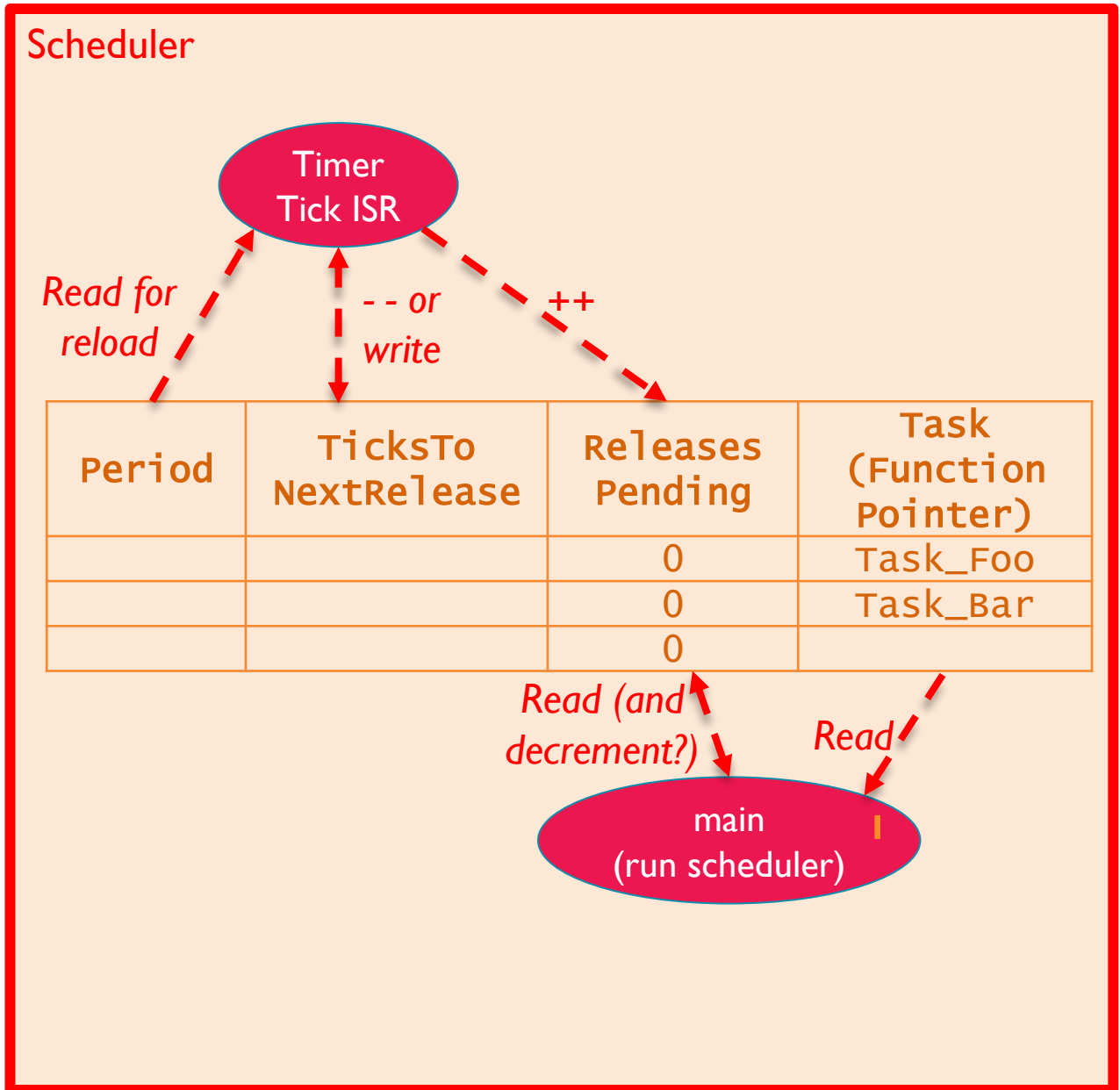


Making a Task Periodic

- On each tick, Timer Tick function decrements ticks remaining until next release
- When ticks remaining reaches zero, Timer Tick ISR increments ReleasesPending
- RTCS_Run_Scheduler can now run task

Adding Periodic Triggering for Tasks

- Task table needs more information
  - Ticks to next release (TTNR)
  - Task period, to reload TTNR value
- Timer Tick Function
  - Decrements delay *TicksToNextRelease*
  - If TTNR reaches zero,
    - Reload it with *Period*
    - Increment *RunRequestsPending*
- **Scheduler** will run tasks with *ReleasesPending* > 0



Code to Update Task Table on Each Tick

```
void RTCS_Timer_Tick(void){
  int i;

  PTB->PSOR = MASK(DEBUG_RTCS_POS); // Debug signal
  RTCS_Num_Ticks++;
  for (i=0; i<RTCS_NUM_TASKS; i++) {
   if ((RTCS_Task_Table[i].Task!=0) && (RTCS_Task_Table[i].TicksToNextRelease>0)) {
    if (--RTCS_Task_Table[i].TicksToNextRelease == 0) {
      RTCS_Task_Table[i].ReleasesPending++;
      RTCS_Task_Table[i].TicksToNextRelease = RTCS_Task_Table[i].Period;
    }
   }
  }
  PTB->PCOR = MASK(DEBUG_RTCS_POS); // Debug signal
}
```

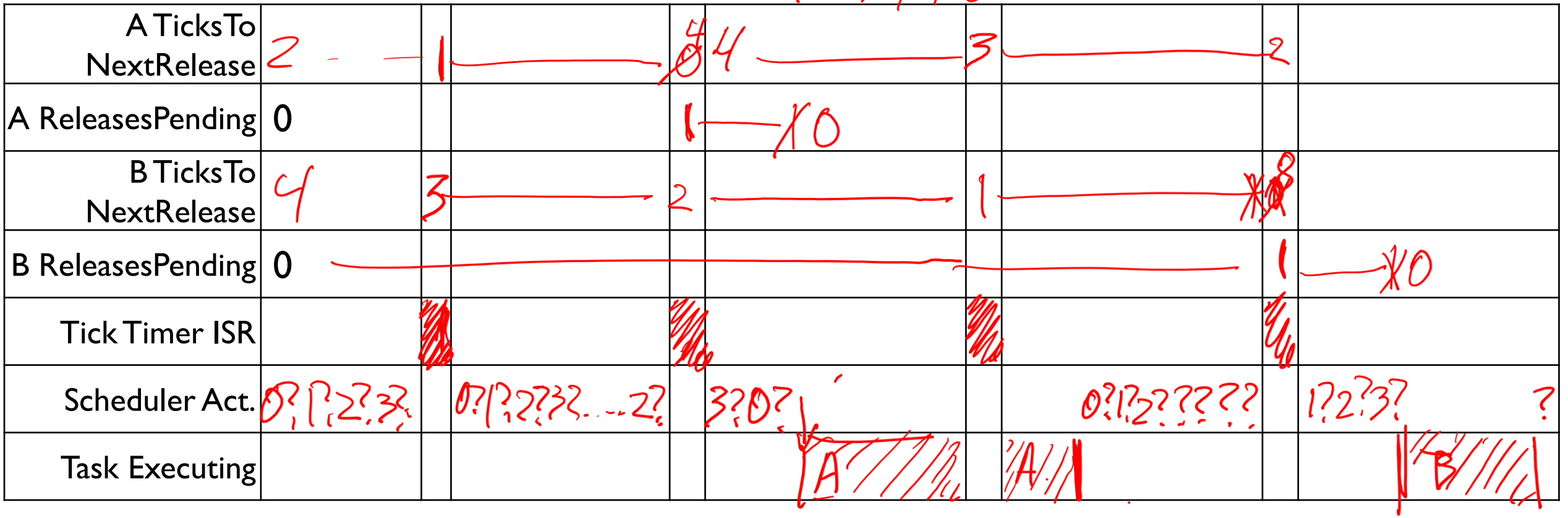
Example of Scheduler Operation with Periodic Tasks

Index	Task	TicksToNextRelease	ReleasesPending	Period
0	A	2	0	4
1	(empty)	0	0	
2	(empty)	0	0	
3	B	4	0	8

A TicksTo NextRelease									
A ReleasesPending	0								
B TicksTo NextRelease									
B ReleasesPending	0								
Tick Timer ISR									
Scheduler Act.									
Task Executing									

Example of Scheduler Operation with Periodic Tasks

Index	Task	TicksToNextRelease	ReleasesPending	Period
0	A	2	0	4
1	(empty)	0	0	
2	(empty)	0	0	
3	B	4	0	8



Enable/Disable Task

- Useful to be able to enable or disable tasks in scheduler
  - Need a new field in the scheduler table
  - Need to update tests in scheduler and timer tick functions
  - Need related API Enable/Disable functions

```
typedef struct {
  void (*Task)(void);
  uint16_t Period;
  uint16_t TicksToNextRelease;
  uint8_t RunRequestsPending;
  uint8_t Enabled;
} RTCS_TASK_ENTRY;
```

```
// RTCS_Run_Scheduler
   if ((RTCS_Task_Table[i].Task != 0) && (RTCS_Task_Table[i].Enabled) &&
      (RTCS_Task_Table[i].RunRequestsPending > 0)) {
```

```
// RTCS_Timer_Tick
   if ((RTCS_Task_Table[i].Task != 0) && (RTCS_Task_Table[i].Enabled) &&
    (RTCS_Task_Table[i].TicksToNextRelease > 0)) {
```

RTCS Scheduler Configuration

RTCS.h

```
#define RTCS_MAX_TASKS 10     // Set maximum number of tasks to be used in system
```

gpio_defs.h

```
#define DEBUG_RTCS_POS     0 // J10 pin 2
```