



# Survey of Architectures

---

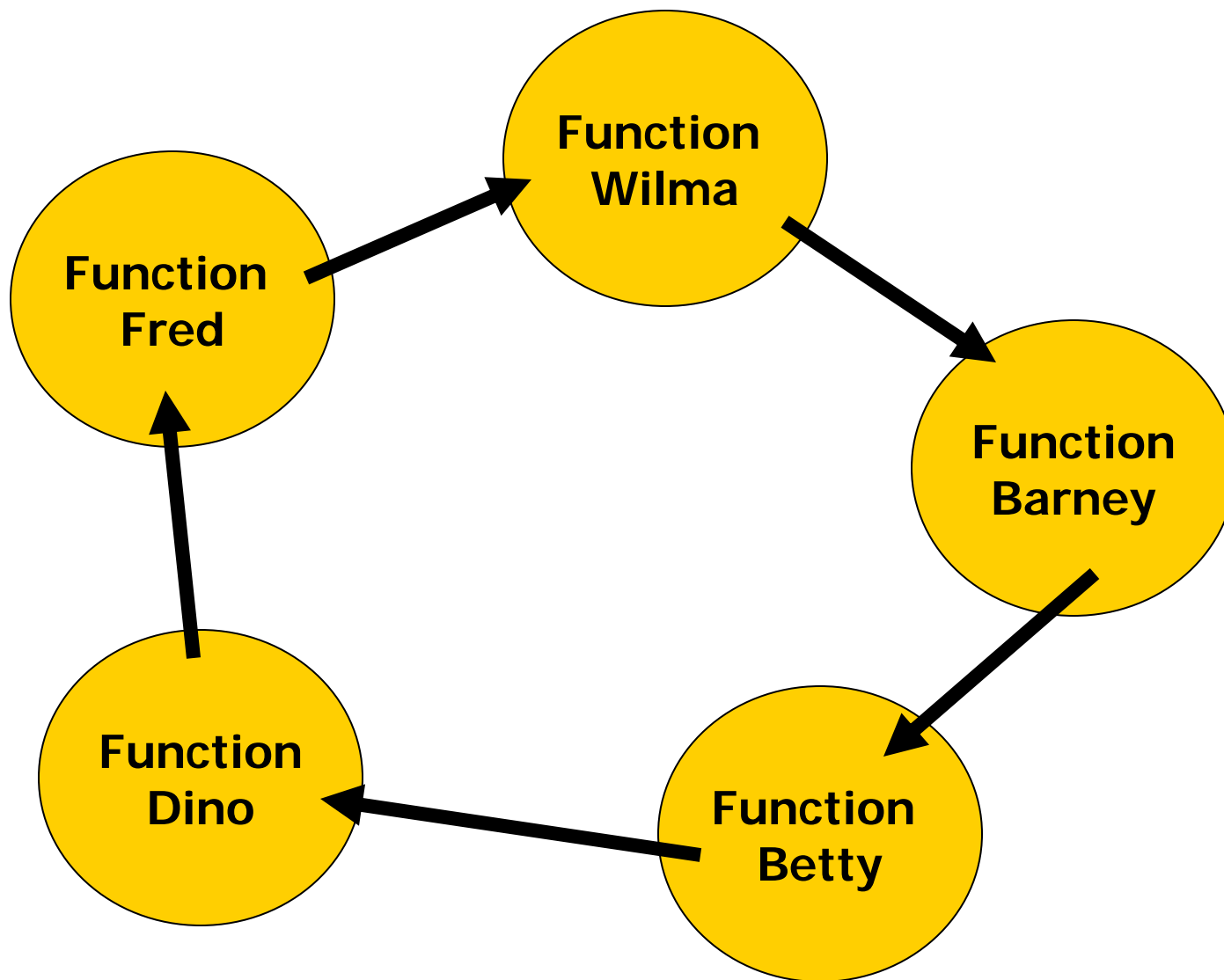
Simon Chapter 5

Round robin, RR with interrupts, Function-  
queue scheduling, RTOS

# Choosing an Architecture

- The best architecture depends on several factors:
  - Real-time requirements of the application (absolute response time)
  - Available hardware (speed, features)
  - Number and complexity of different software features
  - Number and complexity of different peripherals
  - Relative priority of features
- Architecture Selection: Tradeoff between complexity and control over response and priority

# Round Robin



# Round Robin

---

- Simplest architecture
- No interrupts
- Main loop checks each device one at a time, and service whichever needs to be serviced.
- Service order depends on position in the loop.
- No priorities
- No shared data
- No latency issues (other than waiting for other devices to be serviced)

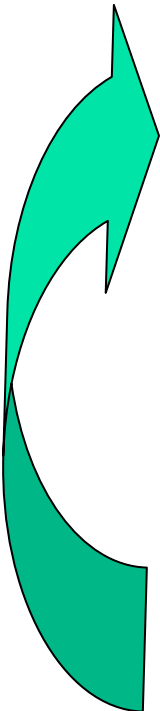
# Round Robin Architecture

```
void main (void)
{
    while (TRUE)
    {
        if (!! I/O Device A needs service)
            !! Service A

        if (!! I/O Device B needs service)
            !! Service B

        if (!! I/O Device C needs service)
            !! Service C

        !! etc.
    }
}
```



# Round Robin

- Pros
  - Simple, no shared data, no interrupts
- Cons
  - Max delay is max time to traverse the loop if all devices need to be serviced
    - Architecture fails if any one device requires a shorter response time
    - Most I/O needs fast response time (buttons, serial ports, etc.)
  - Lengthy processing adversely affects even soft time deadlines
  - Architecture is fragile to added functionality
    - Adding one more device to the loop may break everything
- Uses
  - ????

# Round Robin

- Simple devices
  - Watches
  - Possibly microwave ovens
- Devices where operations are all user initiated and process quickly

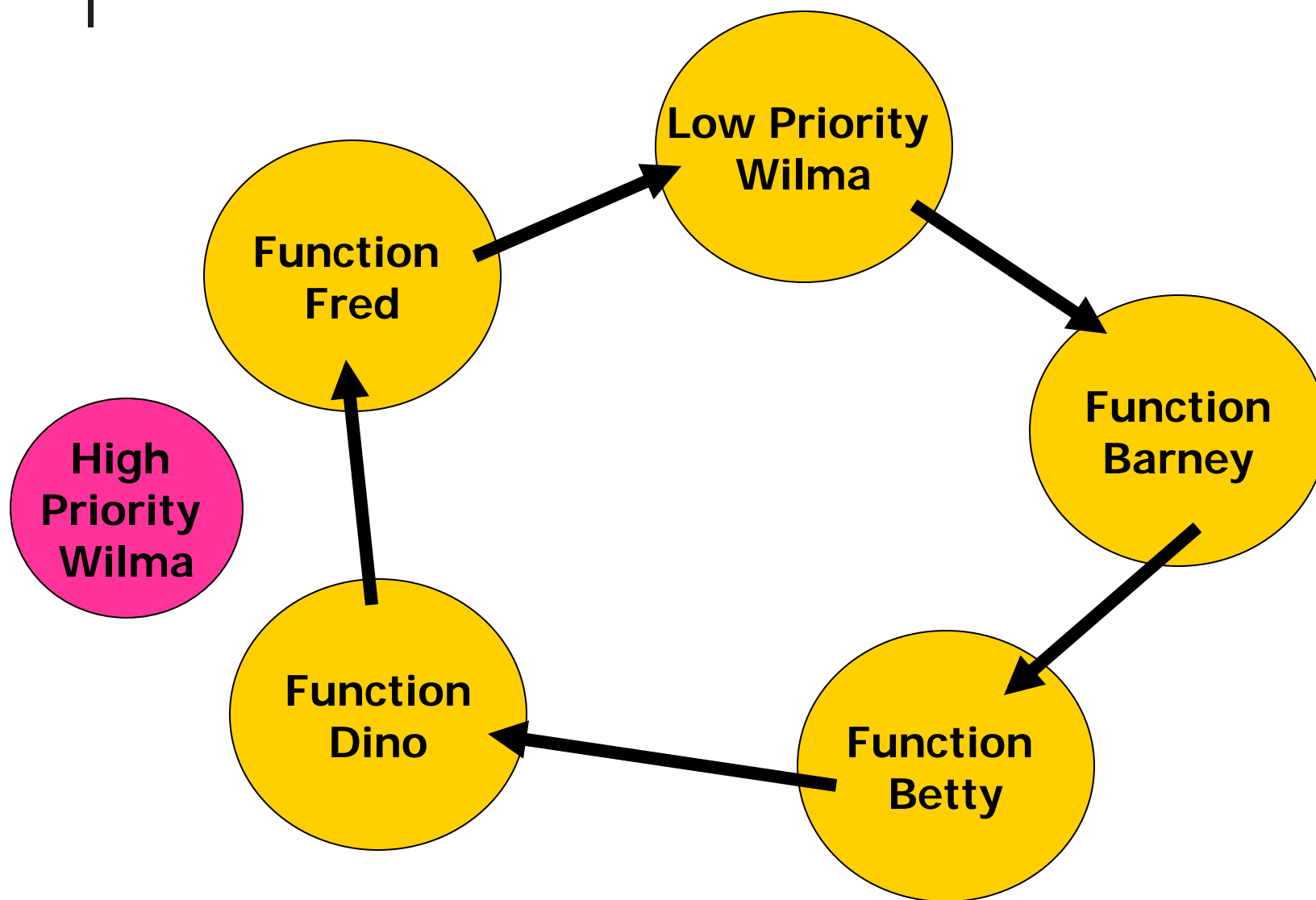
# Multimeter

```
void main (void)
{
    while (TRUE)
    {
        eSwitchPosition = !! read position of switch
        switch(eSwitchPosition)
        {
            case OHMS_1:
                !! Read hardware ohms, display result
                break;
            case VOLTS_100:
                !! Read hardware volts, display result
                break;
            !! etc
        }
    }
}
```

**User will not expect faster response than they can move their hands and the probes!**

**Each operation fast.**

# Round Robin with Interrupts



# Round Robin with Interrupts

---

- Based on Round Robin, but interrupts deal with urgent timing requirements.
  - Interrupts a) service hardware and b) set flags
  - Main routine checks flags and does any lower priority follow-up processing.
- Why? Gives more control over priorities.

```

unsigned char fDeviceA = FALSE;
unsigned char fDeviceB = FALSE;
void interrupt vHandleDeviceA(void)
{
    !! Take care of I/O Device A
    fDeviceA = TRUE;
}
void interrupt vHandleDeviceB(void)
{
    !! Take care of I/O Device B
    fDeviceB = TRUE;
}
void main (void)
{
    while (TRUE)
    {
        if (fDeviceA == TRUE)
        {
            fDeviceA = FALSE;
            !! Handle data to/from Device A
        }
        if (fDeviceB == TRUE)
        {
            fDeviceB = FALSE;
            !! Handle data to/from Device B
        }
    }
}

```

Where is potential problem?

Define shared data flags

Write interrupt routines to service I/O Interrupts. For example, read and save temperature of thermister. Then set flag new temperature received.

Use data flags to communicate with main routine.

For example, use new temperature value to update thermal control routine.

Reset flag.

# Round Robin Priorities

High-priority  
processing

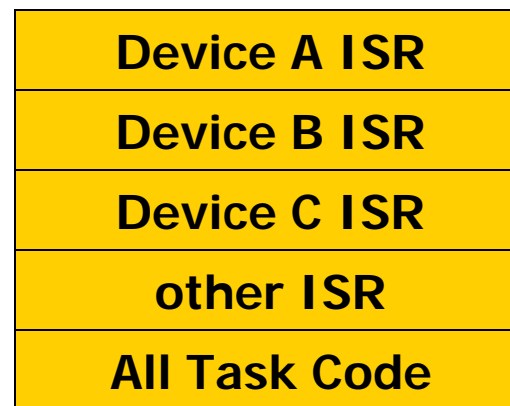


Low-priority  
processing

Round Robin



Round Robin with  
Interrupts



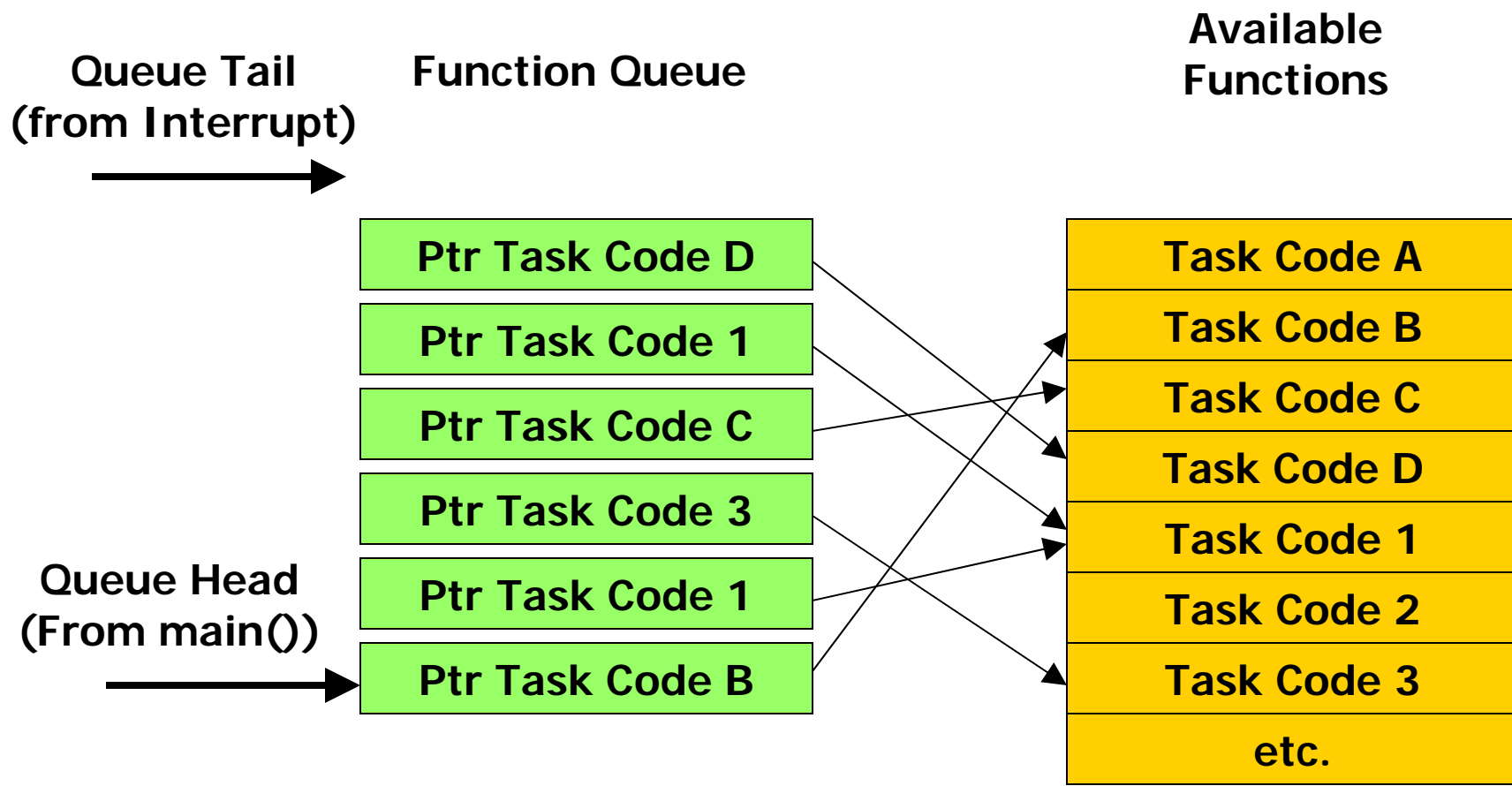
# Round Robin with Interrupts

- Pros
  - Still relatively simple
  - Hardware timing requirements better met
- Cons
  - All task code still executes at same priority
  - Maximum delay unchanged
  - Worst case response time = sum all other execution times + execution times of any other interrupts that occur
- How could you fix?

# Round Robin with Interrupts

- Adjustments
  - Change order flags are checked (e.g., A,B,A,B,A,D)
    - Improves response of A
    - Increases latency of other tasks
  - Move some task code to interrupt
    - Decreases response time of lower priority interrupts
    - May not be able to ensure lower priority interrupt code executes fast enough

# Function Queue Scheduling Architecture



# Function Queue Scheduling Architecture

- Interrupts add function pointers to a queue
- Main routine reads queue and executes calls

```
void main (void)
{
    while (TRUE)
    {
        while (!!queue of function pointers is empty)
            ;
        !! Call functions on the queue
    }
}
```

# Function Queue Scheduling

- Pros:

- Main routine can use any algorithm to choose what order to execute functions (not necessarily FIFO)
- Better response time for highest priority task = length of longest function code
- Can improve best response time by cutting long functions into several pieces

- Cons

- Worse response time for lower priority code (no guarantee it will actually run!)

# Real Time Operating System Architecture

---

- Most complex
- Interrupts handle urgent operations, then signal that there is more work to do for task code
- Differences with previous architectures
  - We don't write signaling flags (RTOS takes care of it)
  - No loop in our code decides what is executed next (RTOS does this)
  - RTOS knows relative task priorities and controls what is executed next
  - RTOS can suspend a task in the middle to execute code of higher priority
- !!! Now we can control task response AND interrupt response!

# Priorities

High-priority processing

Round Robin

Round Robin with Interrupts

Real-time Operating System



Low-priority processing

All Code
----------

Device A ISR
Device B ISR
Device C ISR
other ISR
All Task Code

Device A ISR
Device B ISR
Device C ISR
other ISR
Task Code 1
Task Code 2
Task Code 3
etc.

# Real Time Operating Systems

## ■ Pros

- Worst case response time for highest priority function is zero
- System's high priority response time relatively stable when extra functionality added
- Useful functionality pre-written
- Generally come with vendor tools

## ■ Cons

- RTOS has cost
- Added processing time
- Code out of your control, may contain bugs