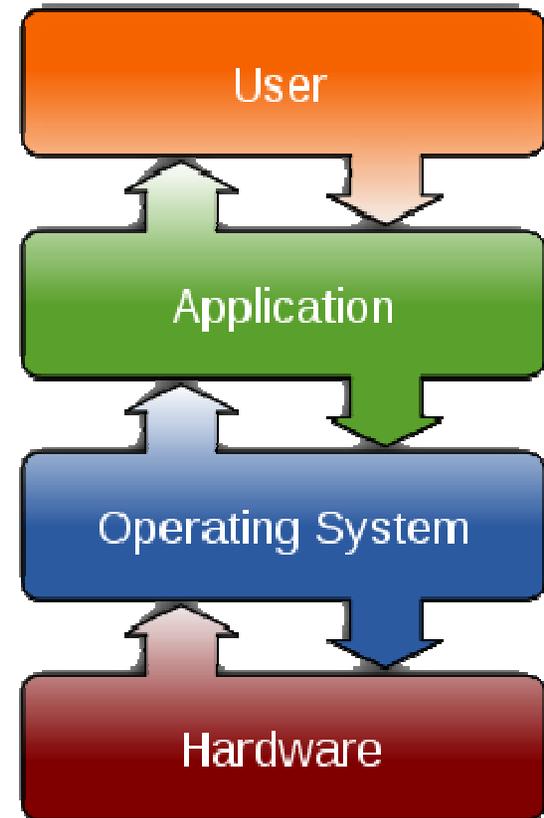


Introduction to SYS/BIOS

Outline

◆ Intro to SYS/BIOS

- ◆ Overview
 - ◆ Threads and Scheduling
 - ◆ Creating a BIOS Thread
 - ◆ System Timeline
 - ◆ Real-Time Analysis Tools
 - ◆ Create A New Project
 - ◆ BIOS Configuration (.CFG)
 - ◆ Platforms
 - ◆ For More Information
- ## ◆ BIOS Threads



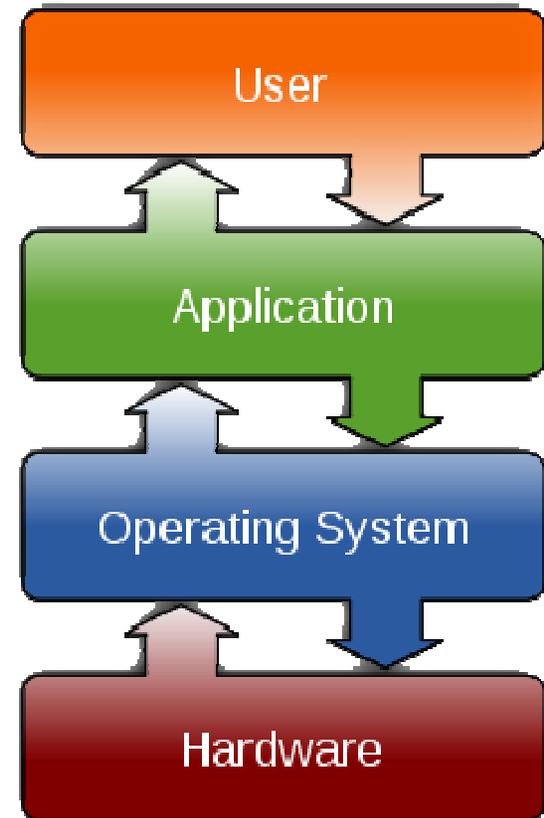
Outline

◆ Intro to SYS/BIOS

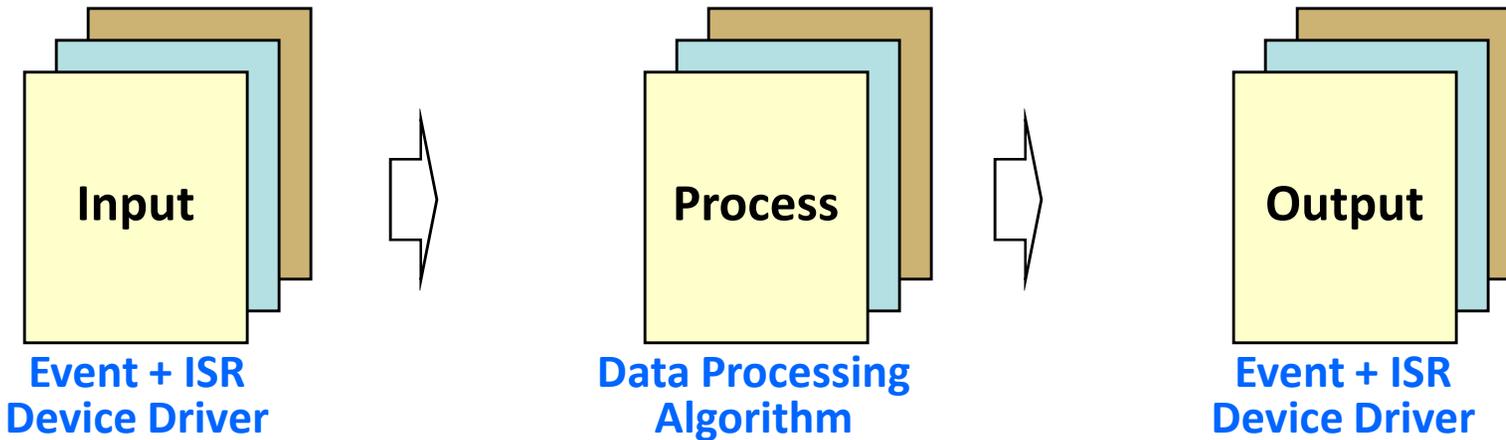
◆ Overview

- ◆ Threads and Scheduling
- ◆ Creating a BIOS Thread
- ◆ System Timeline
- ◆ Real-Time Analysis Tools
- ◆ Create A New Project
- ◆ BIOS Configuration (.CFG)
- ◆ Platforms
- ◆ For More Information

◆ BIOS Threads

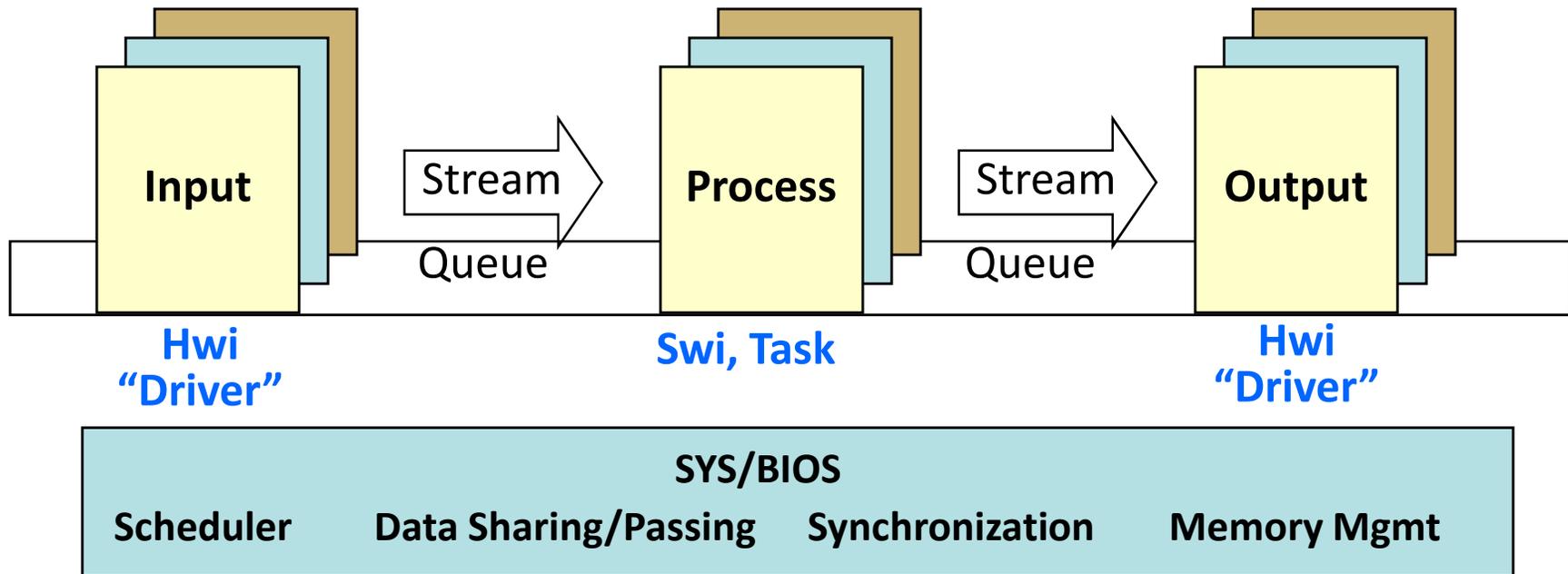


Need for an Operating System



- Simple system: single I-P-O is easy to manage
- As system complexity increases (multiple threads):
 - Can they all meet real time ?
 - Synchronization of events?
 - Priorities of threads/algos ?
 - Data sharing/passing ?
- Two options: “home-grown” or use existing (SYS/BIOS)
(either option requires overhead)
- If you choose an existing O/S, what should you consider?
 - Is it modular?
 - Is it reliable?
 - Is it easy to use?
 - Data sharing/passing?
 - How much does it cost?
 - What code overhead exists?

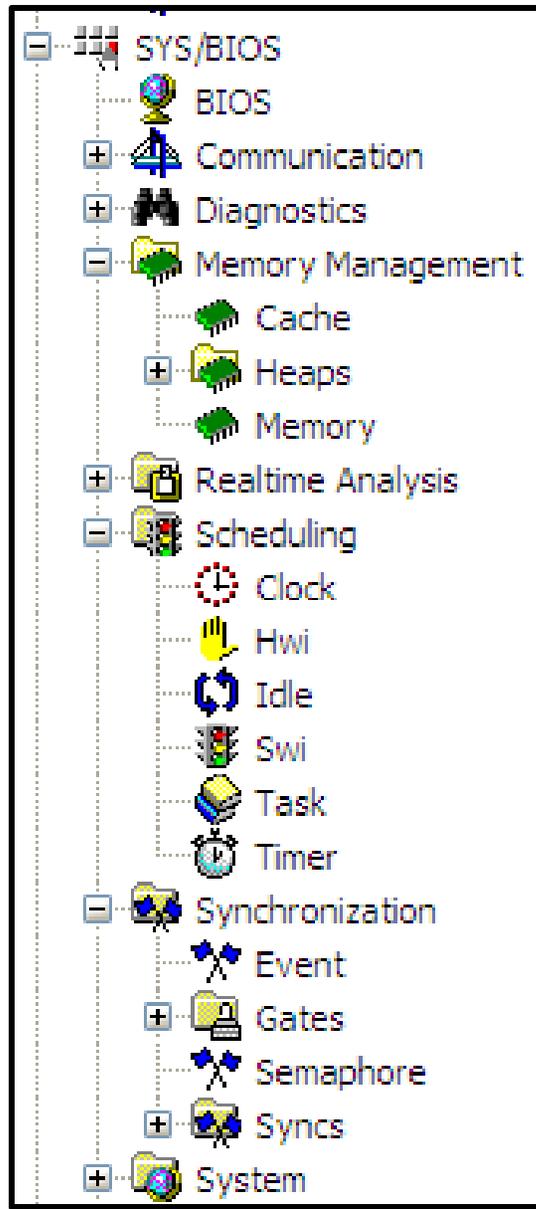
SYS/BIOS Overview



SYS/BIOS is a scalable, real-time kernel used in 1000s of systems today:

- **Pre-emptive Scheduler** to design system to meet real-time (including sync/priorities)
- **Modular** – Include only what is needed
- **API** - pre-defined interface for inter-thread communications
- **Reliable** – 1000s of applications have used it for more than 10 years
- **Footprint** – deterministic, small code size, can choose which modules you desire
- **Cost** – **free of charge**

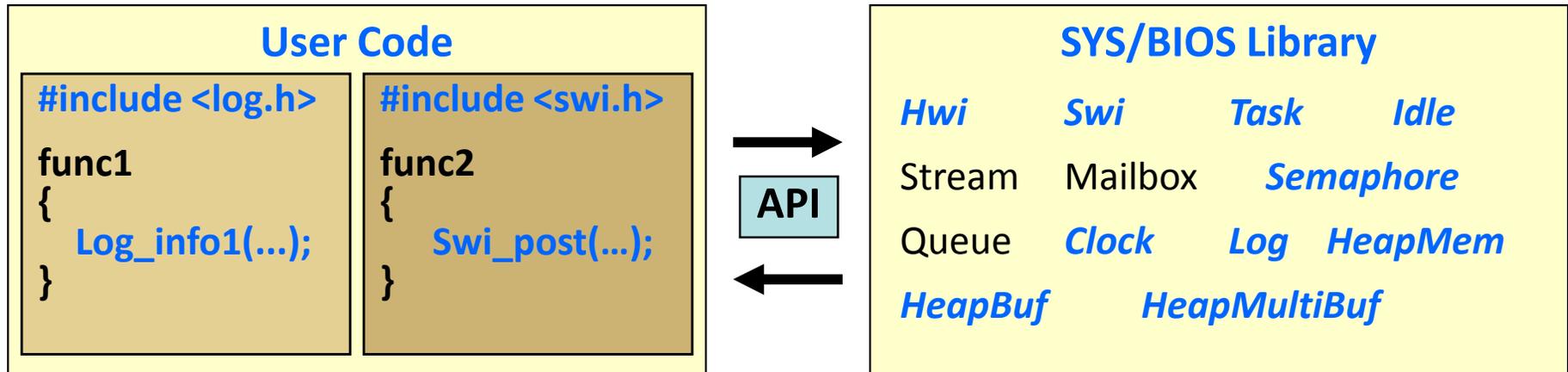
SYS/BIOS Modules & Services



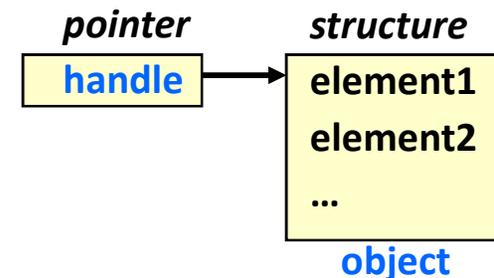
BIOS Configuration

- ◆ **Memory Management**
 - ◆ Cache
 - ◆ Heaps
- ◆ **Realtime Analysis**
 - ◆ Logs
 - ◆ Loads
 - ◆ Execution Graph
- ◆ **Scheduling**
 - ◆ All thread types
- ◆ **Synchronization**
 - ◆ Events
 - ◆ Gates
 - ◆ Semaphores

SYS/BIOS Environment



- ◆ SYS/BIOS is a library that contains modules with a particular interface and data structures.
- ◆ Application Program Interfaces (API) define the interactions (methods) with a module and data structures (objects).
- ◆ Objects are structures that define the state of a component.
 - ◆ Pointers to objects are called **handles**.
 - ◆ Object-based programming offers:
 - ◆ *Better encapsulation and abstraction*
 - ◆ *Multiple instance ability*



Definitions / Vocabulary

- ◆ In this workshop, we'll be using these terms often:

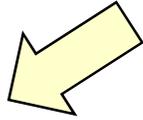
Real-time System

- *Where processing must keep up with the rate of I/O*

Function

- *Sequence of program instructions that produce a given result*

Thread



- *Function that executes within a specific context (regs, stack, PRIORITY)*

API

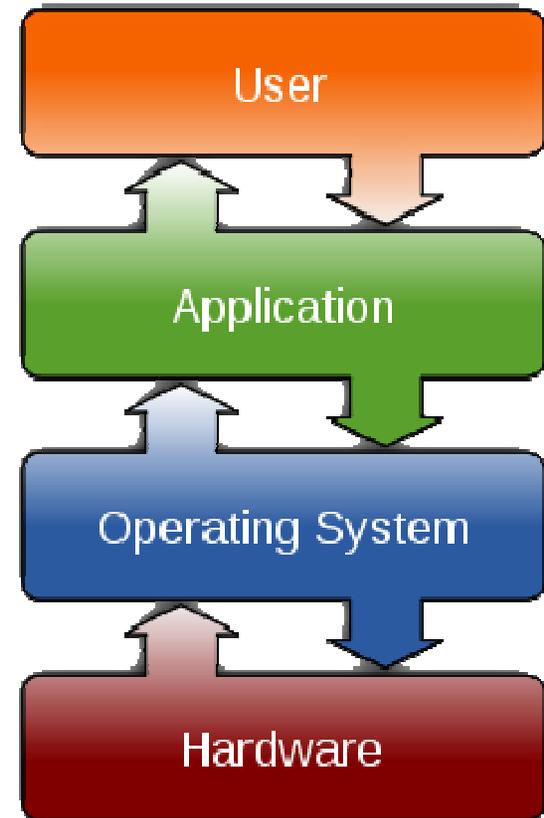
- *Application Programming Interface provides methods for interacting with library routines and data objects*

Comparing RTOS and GP/OS

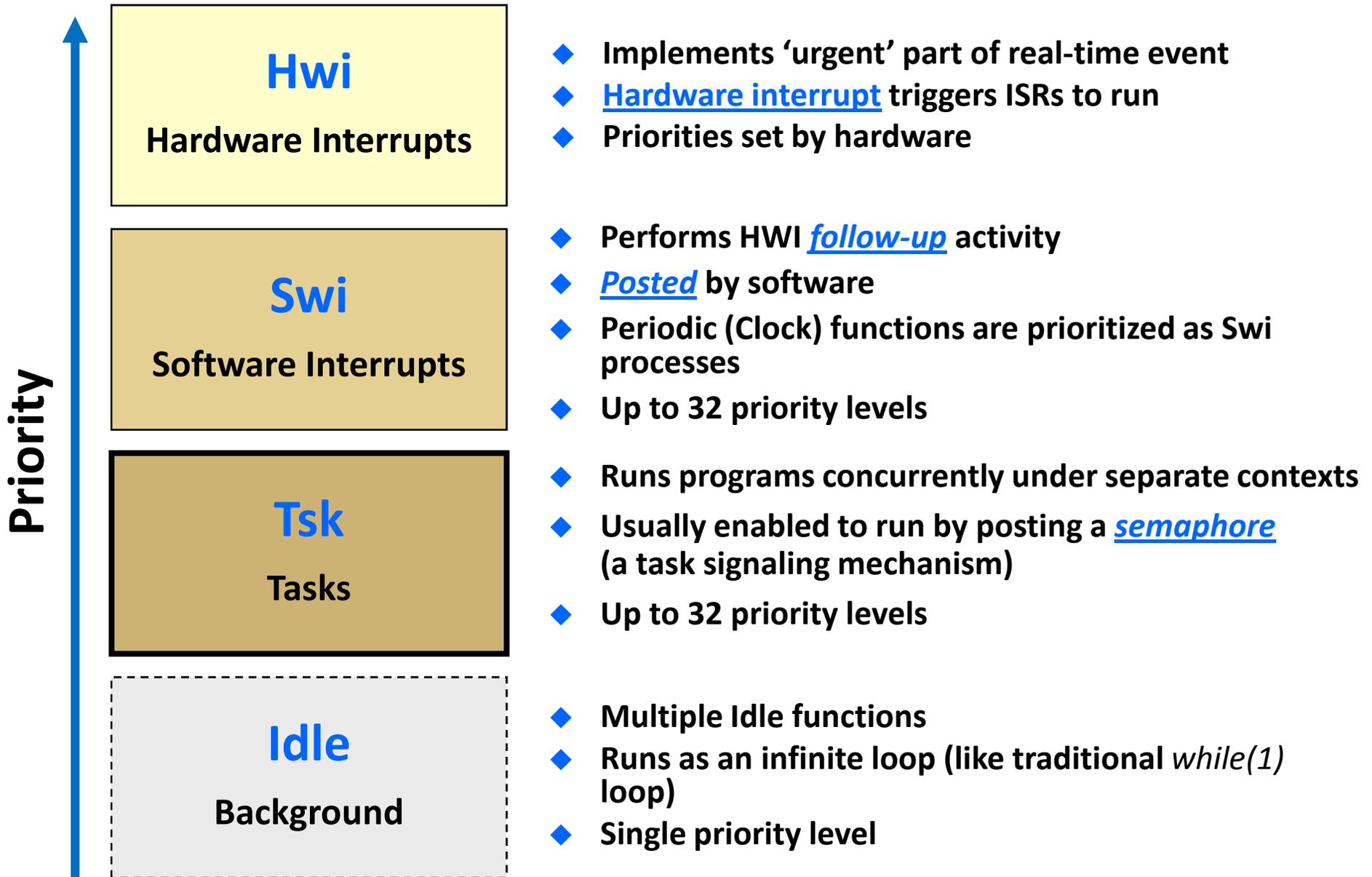
	GP/OS (e.g. Linux)	RTOS (e.g. SYS/BIOS)
Scope	General	Specific
Size	Large: 5M-50M	Small: 5K-50K
Event response	1ms to .1ms	100 – 10 ns
File management	FAT, etc.	FatFS
Dynamic Memory	Yes	Yes
Threads	Processes, pThreads, Ints	Hwi, Swi, Task, Idle
Scheduler	Time Slicing	Preemption
Host Processor	ARM, x86, Power PC	ARM, MSP430, DSP

Outline

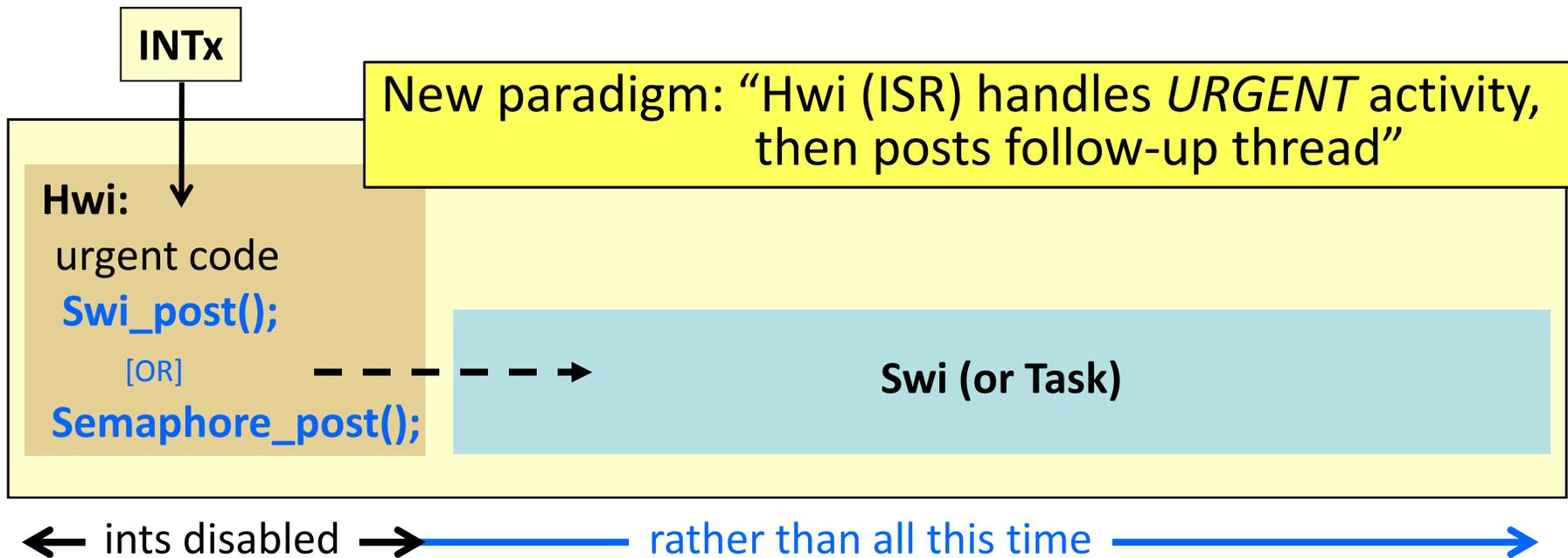
- ◆ Intro to SYS/BIOS
 - ◆ Overview
 - ◆ **Threads and Scheduling**
 - ◆ Creating a BIOS Thread
 - ◆ System Timeline
 - ◆ Real-Time Analysis Tools
 - ◆ Create A New Project
 - ◆ BIOS Configuration (.CFG)
 - ◆ Platforms
 - ◆ For More Information
- ◆ BIOS Threads



SYS/BIOS Thread Types



Hwi Signaling Swi/Task



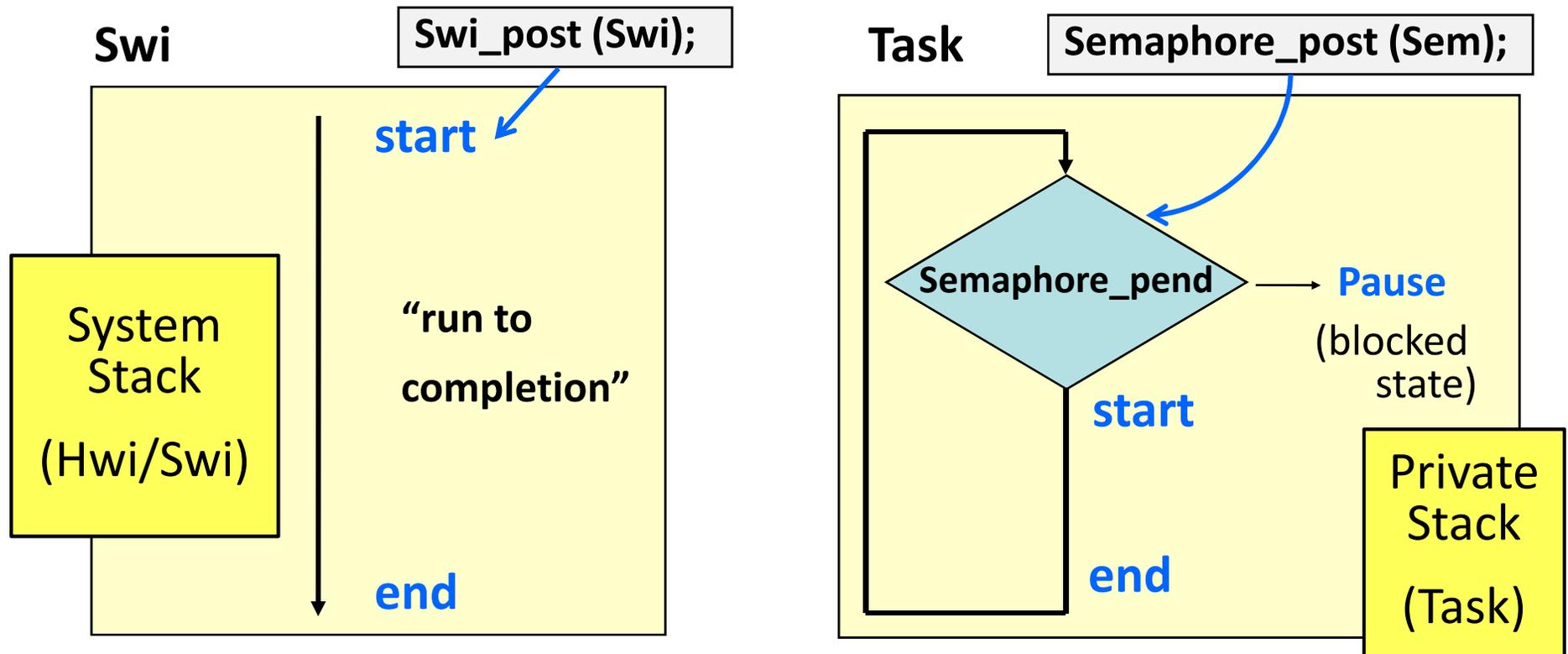
Hwi

- ◆ Fast response to interrupts
- ◆ Minimal context switching
- ◆ High priority only
- ◆ Can post Swi
- ◆ Use for urgent code only – then post follow up activity

Swi

- ◆ Latency in response time
- ◆ Context switch performed
- ◆ Selectable priority levels
- ◆ Can post another Swi
- ◆ Execution managed by scheduler

Swi and Tasks

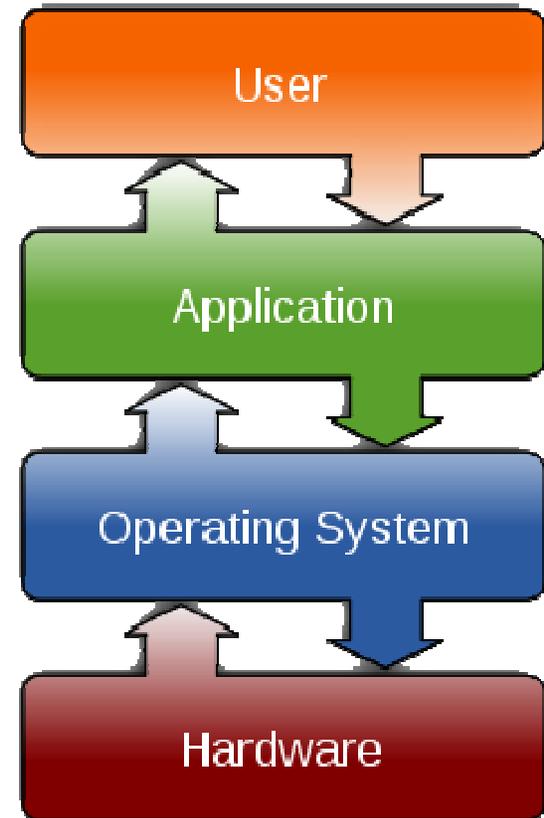


- ◆ Similar to hardware interrupt, but triggered when posted
- ◆ All Swi activities share system software stack with Hwi activities.

- ◆ Unblocking triggers execution (also could be mailbox, events, etc.)
- ◆ Each Task has its own stack, which allows them to pause (i.e. block)
- ◆ **Topology: prologue, loop, epilogue...**

Outline

- ◆ Intro to SYS/BIOS
 - ◆ Overview
 - ◆ Threads and Scheduling
 - ◆ **Creating a BIOS Thread**
 - ◆ System Timeline
 - ◆ Real-Time Analysis Tools
 - ◆ Create A New Project
 - ◆ BIOS Configuration (.CFG)
 - ◆ Platforms
 - ◆ For More Information
- ◆ BIOS Threads



Thread (Object) Creation in BIOS

Users can create threads (BIOS resources or “objects”):

- Statically (via the GUI or .cfg script)
- Dynamically (via C code)
- BIOS doesn't care – but you might...

Static (GUI or Script)

Generic Hardware Interrupt Instance

Basic | Advanced

Basic Settings

Name:

ISR function:

Interrupt Number:

Interrupt Scheduling Options

Interrupts to mask:

Priority:

Event Id:

Enabled at startup

```
var Hwi = xdc.useModule('ti.sysbios.hal.Hwi');  
var hwiParams = new Hwi.Params();  
hwiParams.eventId = 61;  
Hwi.create(5, "&isrAudio", hwiParams);
```

app.cfg

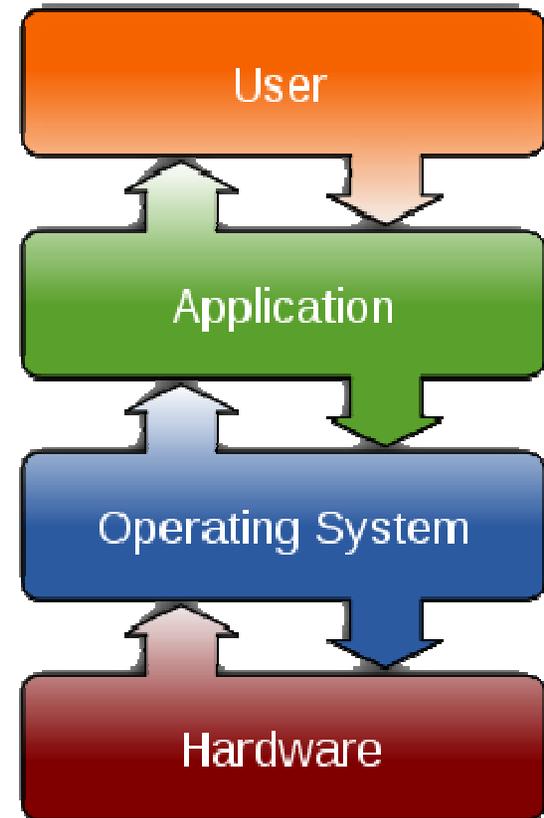
Dynamic (C Code)

```
#include <ti/sysbios/hal/Hwi.h>  
Hwi_Params hwiParams;  
Hwi_Params_init(&hwiParams);  
hwiParams.eventId = 61;  
Hwi_create(5, isrAudio, &hwiParams, NULL);
```

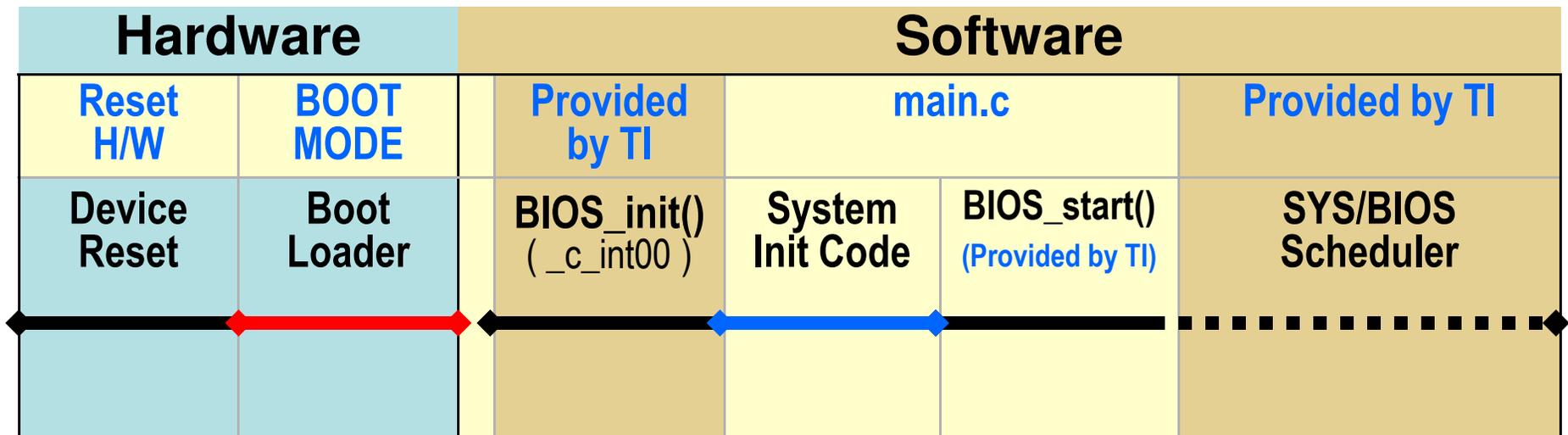
app.c

Outline

- ◆ Intro to SYS/BIOS
 - ◆ Overview
 - ◆ Threads and Scheduling
 - ◆ Creating a BIOS Thread
 - ◆ System Timeline
 - ◆ Real-Time Analysis Tools
 - ◆ Create A New Project
 - ◆ BIOS Configuration (.CFG)
 - ◆ Platforms
 - ◆ For More Information
- ◆ BIOS Threads



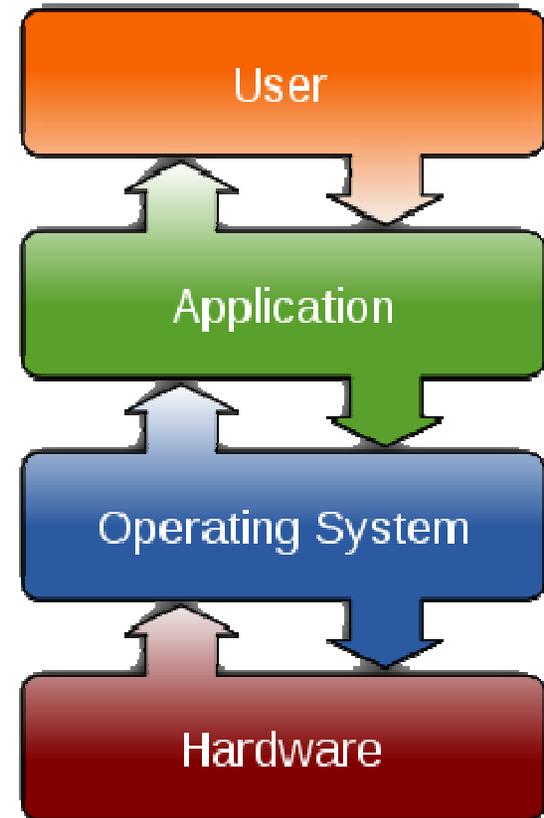
System Timeline



- ◆ **RESET:** Device is reset, then jumps to bootloader or code entry point (`c_int00`)
- ◆ **BOOT MODE** runs bootloader (if applicable)
- ◆ `BIOS_init()` configures static BIOS objects, jumps to `c_int00` to init Stack Pointer (SP), globals/statics, then calls `main()`
- ◆ `main()`
 - ◆ User initialization
 - ◆ Must execute `BIOS_start()` to enable BIOS Scheduler & INTs

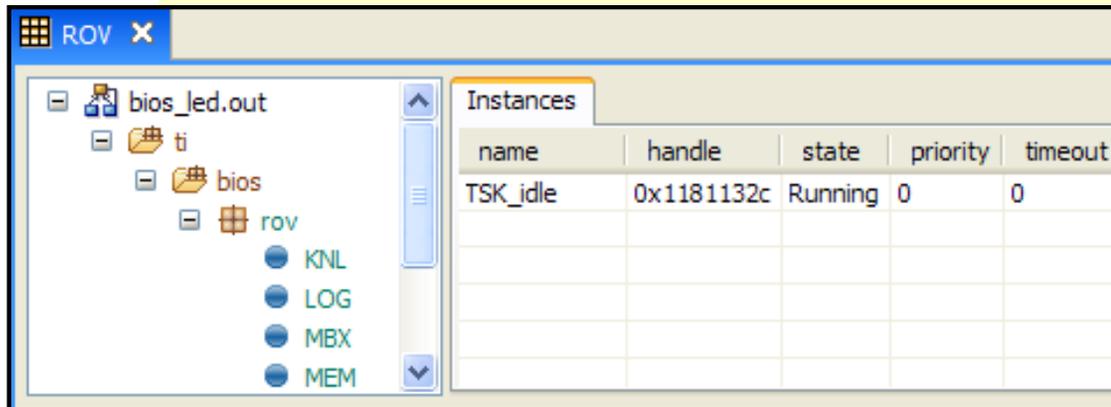
Outline

- ◆ Intro to SYS/BIOS
 - ◆ Overview
 - ◆ Threads and Scheduling
 - ◆ Creating a BIOS Thread
 - ◆ System Timeline
 - ◆ Real-Time Analysis Tools
 - ◆ Create A New Project
 - ◆ BIOS Configuration (.CFG)
 - ◆ Platforms
 - ◆ For More Information
- ◆ BIOS Threads



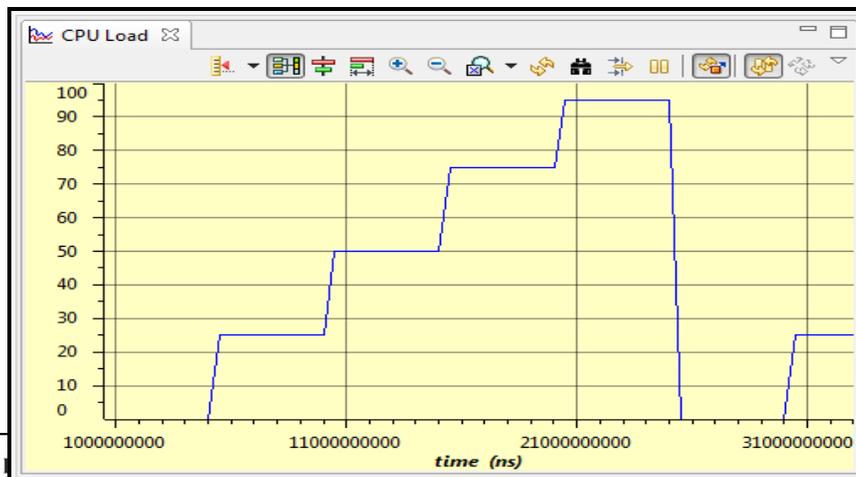
Built-in Real-Time Analysis Tools

- ◆ Gather data on target (30-40 CPU cycles)
- ◆ Format data on host (1000s of host PC cycles)
- ◆ Data gathering does NOT stop target CPU
- ◆ Halt CPU to see results (stop-time debug)



RunTime Obj View (ROV)

- ◆ Halt to see results
- ◆ Displays stats about all threads in system



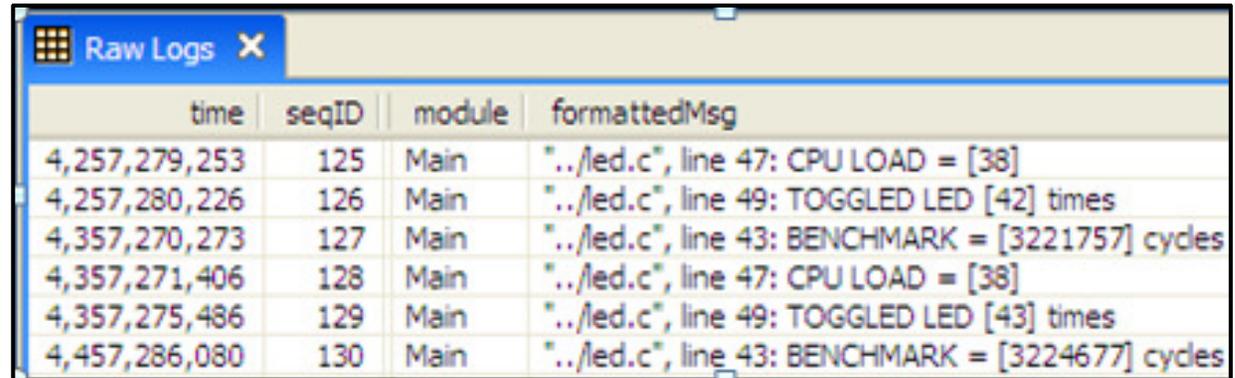
CPU/Thread Load Graph

- ◆ Analyze time NOT spent in Idle

Built-in Real-Time Analysis Tools

Logs

- ◆ Send DBG Msgs to PC
- ◆ Data displayed during stop-time
- ◆ Deterministic, low CPU cycle count
- ◆ WAY more efficient than traditional `printf()`

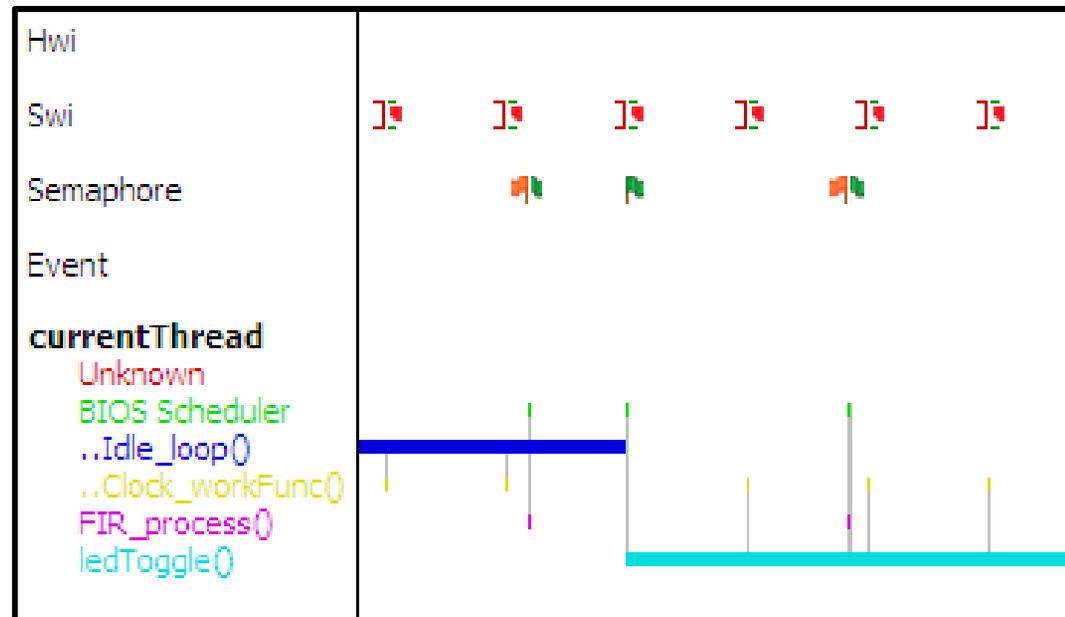


time	seqID	module	formattedMsg
4,257,279,253	125	Main	"../led.c", line 47: CPU LOAD = [38]
4,257,280,226	126	Main	"../led.c", line 49: TOGGLED LED [42] times
4,357,270,273	127	Main	"../led.c", line 43: BENCHMARK = [3221757] cycles
4,357,271,406	128	Main	"../led.c", line 47: CPU LOAD = [38]
4,357,275,486	129	Main	"../led.c", line 49: TOGGLED LED [43] times
4,457,286,080	130	Main	"../led.c", line 43: BENCHMARK = [3224677] cycles

```
Log_info1("TOGGLED LED [%u] times", count);
```

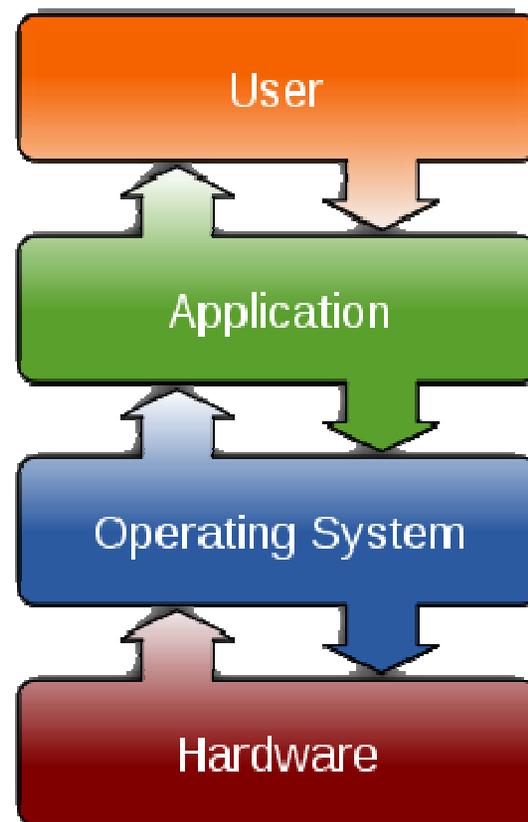
Execution Graph

- ◆ View system events down to the CPU cycle
- ◆ Calculate benchmarks



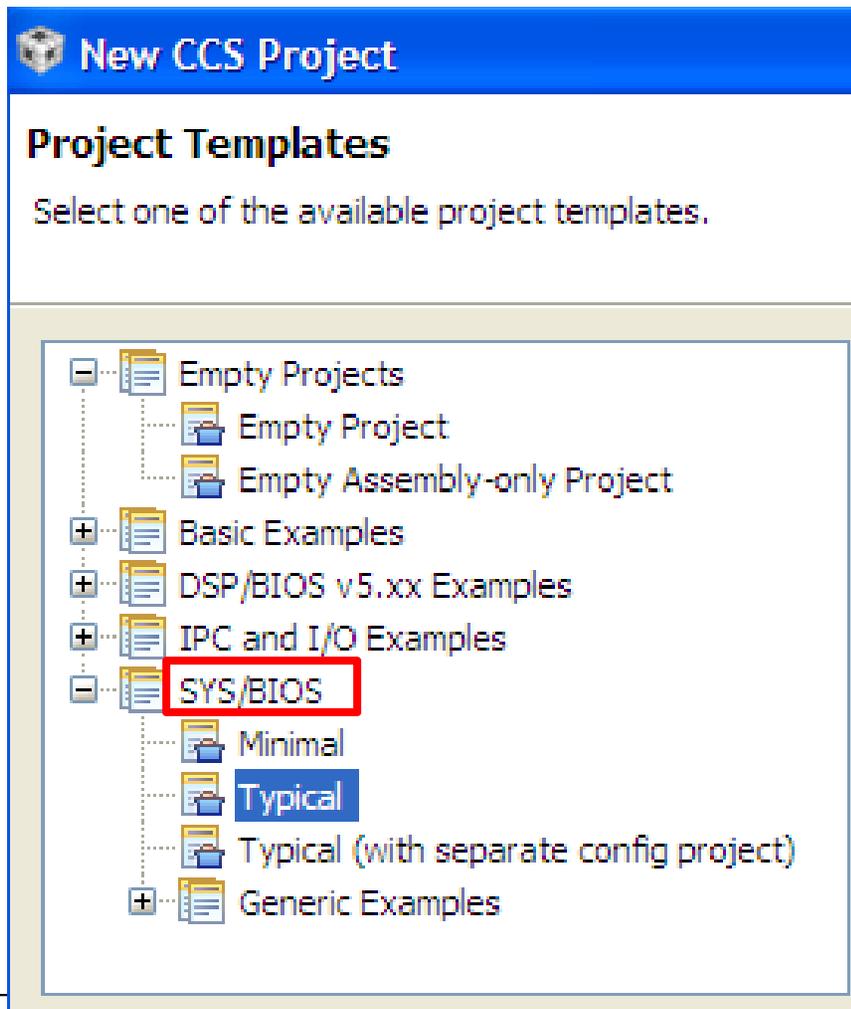
Outline

- ◆ Intro to SYS/BIOS
 - ◆ Overview
 - ◆ Threads and Scheduling
 - ◆ Creating a BIOS Thread
 - ◆ System Timeline
 - ◆ Real-Time Analysis Tools
 - ◆ **Create A New Project**
 - ◆ BIOS Configuration (.CFG)
 - ◆ Platforms
 - ◆ For More Information
- ◆ BIOS Threads



Building a NEW SYS/BIOS Project

- ◆ Create CCS Project (as normal), then click:
- ◆ Select a SYS/BIOS Example:



What is in the project created by *Typical*?

- Paths to SYS/BIOS tools
- .CFG file (app.cfg) that contains a “typical” configuration for static objects (e.g. Swi, Task).
- Source files (main.c) that contain the appropriate #includes of header files.

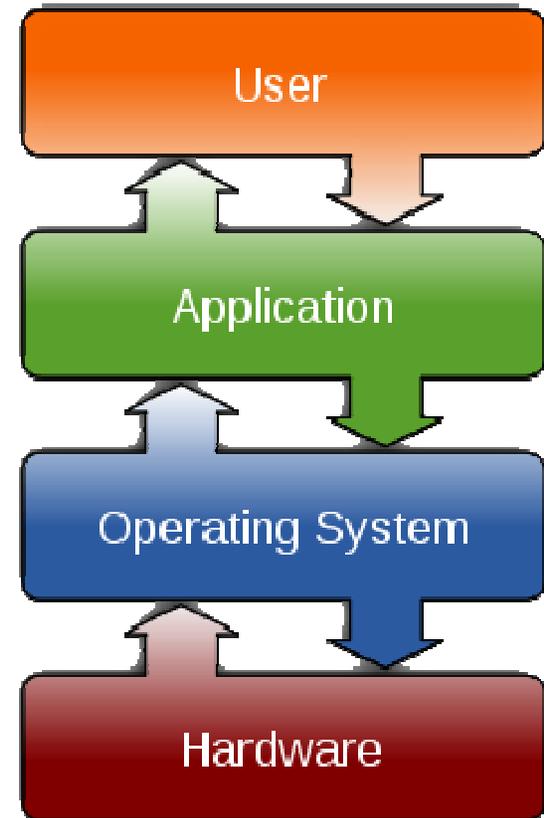
SYS/BIOS Project Settings

- ◆ Select versions for XDC, IPC, SYS/BIOS, xDAIS.
- ◆ Select the *Platform* (similar to the .tcf seed file for memory).

The screenshot displays the 'CCS Build' configuration window. At the top, the 'Build configuration' is set to 'Debug'. Below this, there are four tabs: 'General', 'RTSC', 'Link Order', and 'Dependencies'. The 'RTSC' tab is selected, and the 'XDCtools version' is set to '3.22.1.21', which is highlighted with a red box. Underneath, the 'Products and Repositories' section is visible, with an 'Order' button. A tree view shows several components: 'Inter-processor Communication' (checked), 'SYS/BIOS' (checked and highlighted with a red box), and 'XDAIS' (unchecked). Below the tree, the 'Target' is set to 'ti.targets.C674', the 'Platform' is set to 'ti.platforms.evm6748' (indicated by a blue arrow), and the 'Build-profile' is set to 'release'.

Outline

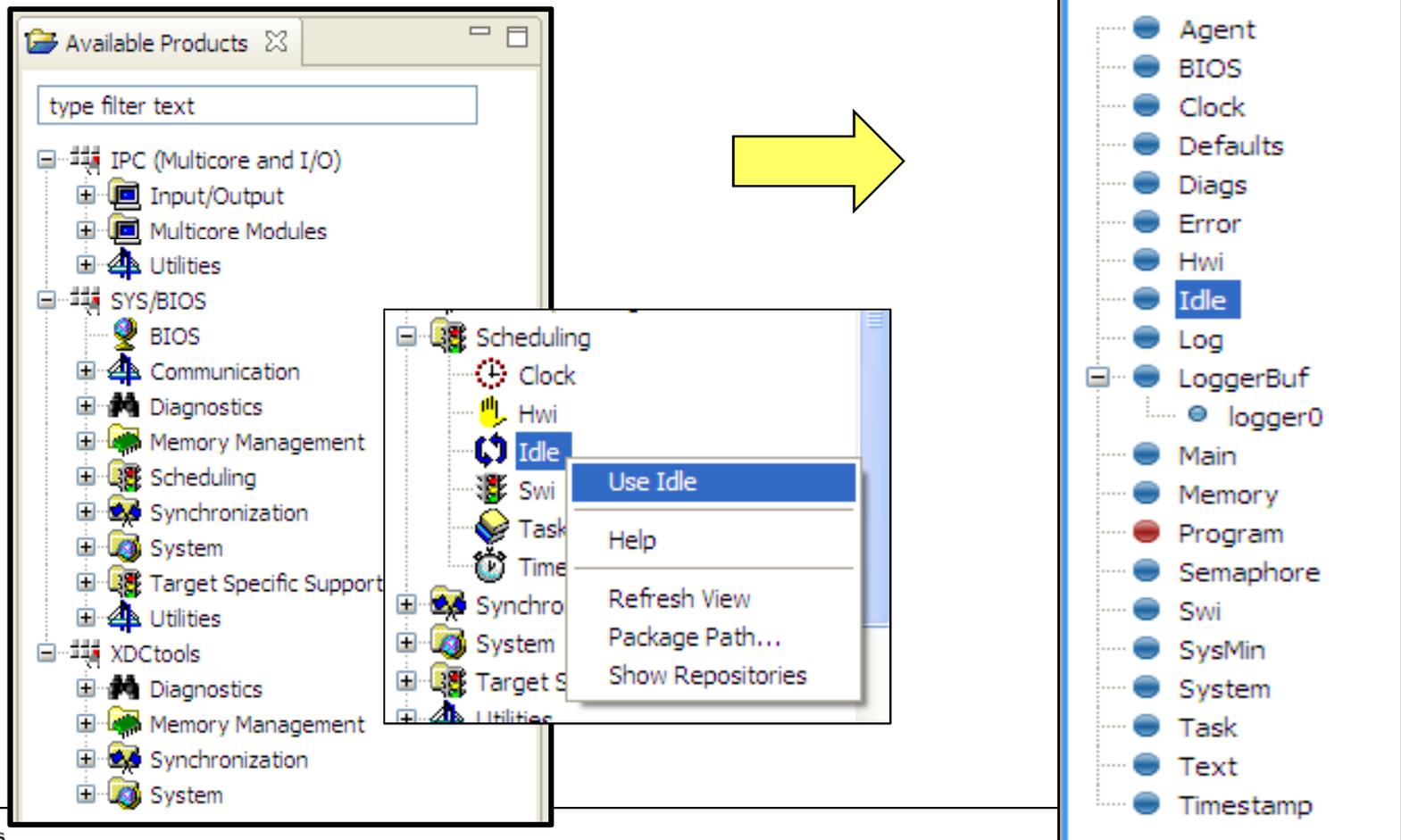
- ◆ Intro to SYS/BIOS
 - ◆ Overview
 - ◆ Threads and Scheduling
 - ◆ Creating a BIOS Thread
 - ◆ System Timeline
 - ◆ Real-Time Analysis Tools
 - ◆ Create A New Project
 - ◆ BIOS Configuration (.CFG)
 - ◆ Platforms
 - ◆ For More Information
- ◆ BIOS Threads



Static BIOS Configuration

Users interact with the CFG file via the GUI – XGCONF:

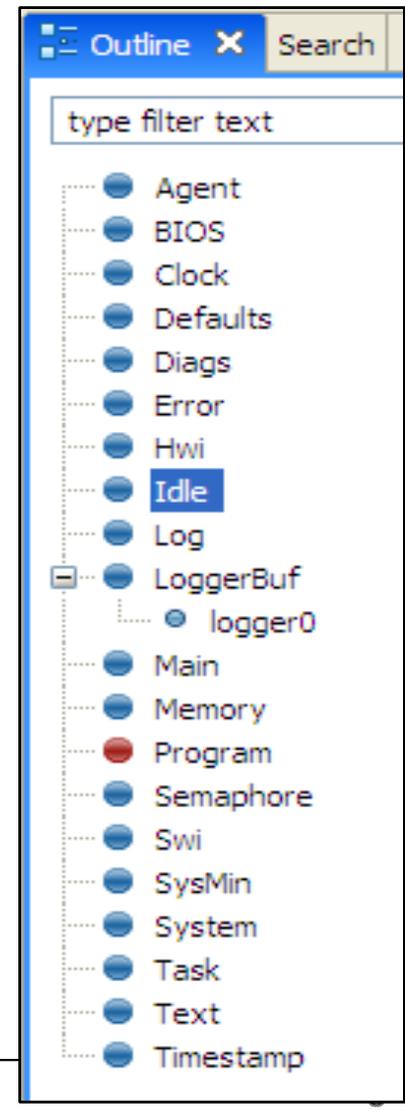
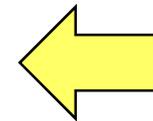
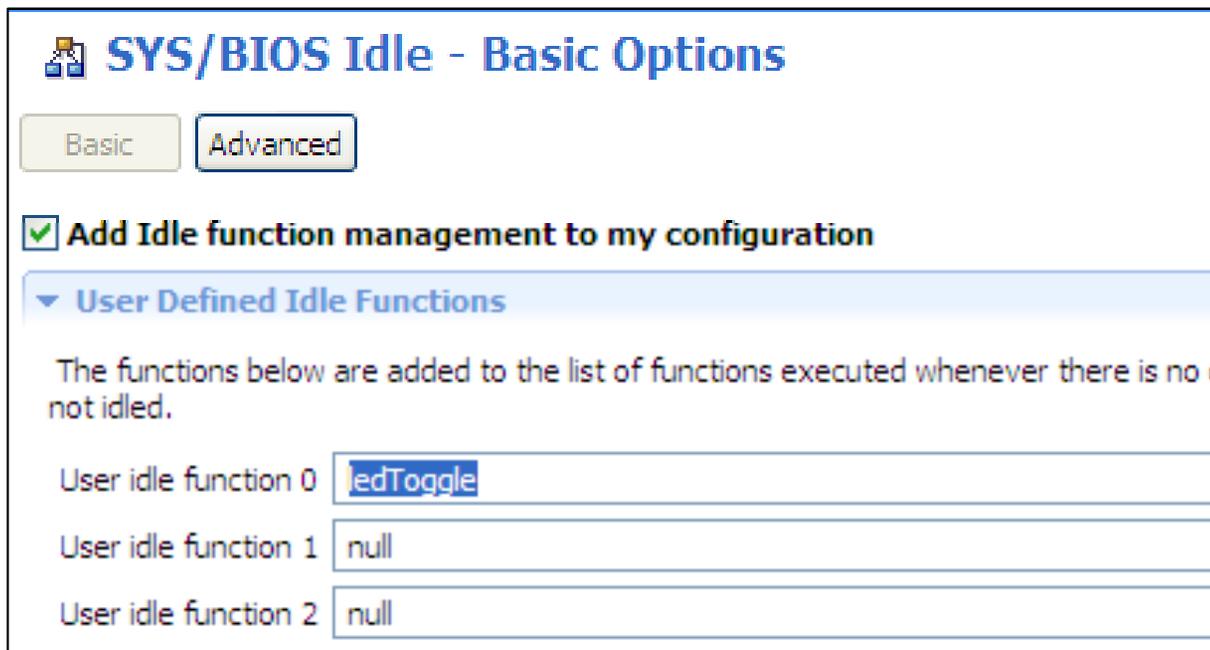
- XGCONF shows *Available Products*; Right-click and select *Use Mod*.
- *Mod* shows up in *Outline* view. Right-click and select *Add New*.
- All graphical changes in GUI are displayed in [.cfg](#) source code.



Static Config – .CFG Files

◆ Users interact with the CFG file via the GUI – XGCONF

- When you *Add New*, a dialogue box is provided to set up parameters.
- This window provides two views:
 - *Basic*
 - *Advanced*

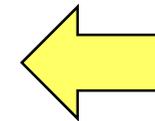


.CFG Files (XDC script)

- ◆ All changes made to the GUI are reflected with java script in the corresponding .CFG file.
- ◆ Click on a module in the *Outline* view to see the corresponding script in the *app.cfg* file.

```
app.cfg X
11
12 var BIOS = xdc.useModule('ti.sysbios.BIOS');
13 var Clock = xdc.useModule('ti.sysbios.knl.Clock');
14 var Swi = xdc.useModule('ti.sysbios.knl.Swi');
15 var Task = xdc.useModule('ti.sysbios.knl.Task');
16 var Semaphore = xdc.useModule('ti.sysbios.knl.Semaphore');
17 var Hwi = xdc.useModule('ti.sysbios.hal.Hwi');
18 var Idle = xdc.useModule('ti.sysbios.knl.Idle');
19 var Timestamp = xdc.useModule('xdc.runtime.Timestamp');
20
```

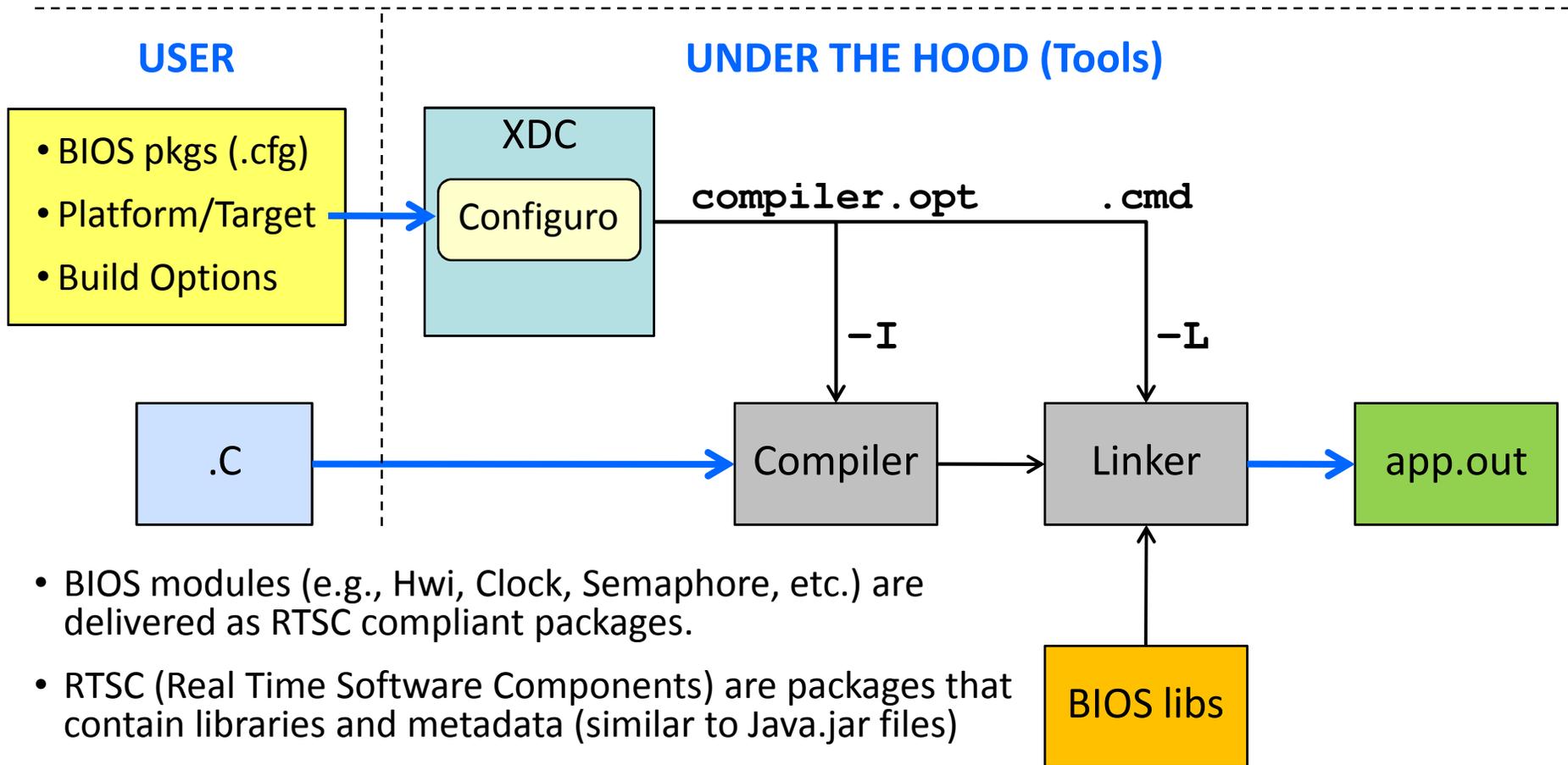
```
98 Idle.idleFxnns[0] = "&ledToggle";
```

A screenshot of the 'Outline' view in a development tool. The view shows a tree structure of modules. The 'Idle' module is selected and highlighted in blue. The tree includes modules like Agent, BIOS, Clock, Defaults, Diags, Error, Hwi, Idle, Log, LoggerBuf (with a sub-module logger0), Main, Memory, Program, Semaphore, Swi, SysMin, System, Task, Text, and Timestamp. A search bar is at the top with the text 'type filter text'.

- Agent
- BIOS
- Clock
- Defaults
- Diags
- Error
- Hwi
- Idle
- Log
- LoggerBuf
 - logger0
- Main
- Memory
- Program
- Semaphore
- Swi
- SysMin
- System
- Task
- Text
- Timestamp

Configuration Build Flow (CFG)

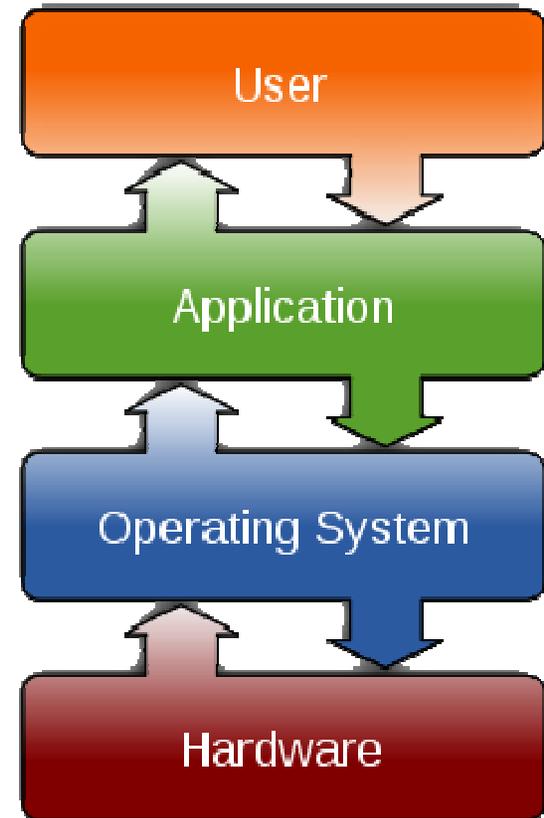
- SYS/BIOS: User configures system with CFG file
- The rest is “under the hood.”



- BIOS modules (e.g., Hwi, Clock, Semaphore, etc.) are delivered as RTSC compliant packages.
- RTSC (Real Time Software Components) are packages that contain libraries and metadata (similar to Java.jar files)
- XDC (eXpress DSP Components) is a set of tools that consume RTSC packages (knows how to read RTSC metadata).

Outline

- ◆ Intro to SYS/BIOS
 - ◆ Overview
 - ◆ Threads and Scheduling
 - ◆ Creating a BIOS Thread
 - ◆ System Timeline
 - ◆ Real-Time Analysis Tools
 - ◆ Create A New Project
 - ◆ BIOS Configuration (.CFG)
 - ◆ Platforms
 - ◆ For More Information
- ◆ BIOS Threads



Platform (Memory Config)

Memory Configuration

- ◆ Create Internal Memory Segments (e.g. IRAM)
- ◆ Configure cache
- ◆ Define External Memory Segments

Section Placement

- ◆ Can link code, data, and stack to any defined mem segment

Custom Platform

- ◆ Use *Import* button to copy “seed” platform and then customize

The screenshot shows the Platform Configuration tool interface. The 'Tools' menu is open, showing options like 'Port Connect', 'Pin Connect', 'RTSC Tools', 'Profile', and 'ROV'. The 'Platform' submenu is also open, showing 'Edit/View' and 'New' options. The main configuration area is titled 'Device Details' and shows the following settings:

- Device Name: TMS320C6748
- Device Family: c6000
- Clock Speed (MHz): 300.0
- Import... button

The 'Device Memory' section contains a table with the following data:

Name	Base	Length	Space	Access
IRAM	0x11800000	0x00040000	code/data	RWX
L3_CBA_RAM	0x80000000	0x00020000	code/data	RWX
IROM	0x11700000	0x00100000	code/data	RX
L1PSRAM	0x11E00000	0x00000000	code	RWX
L1DSRAM	0x11F00000	0x00000000	data	RW

Below the table, there are cache configuration options:

- L1D Cache: 32k
- L1P Cache: 32k
- L2 Cache: 0k

A checkbox for 'Customize Memory' is present and unchecked.

The 'External Memory' section contains a table with the following data:

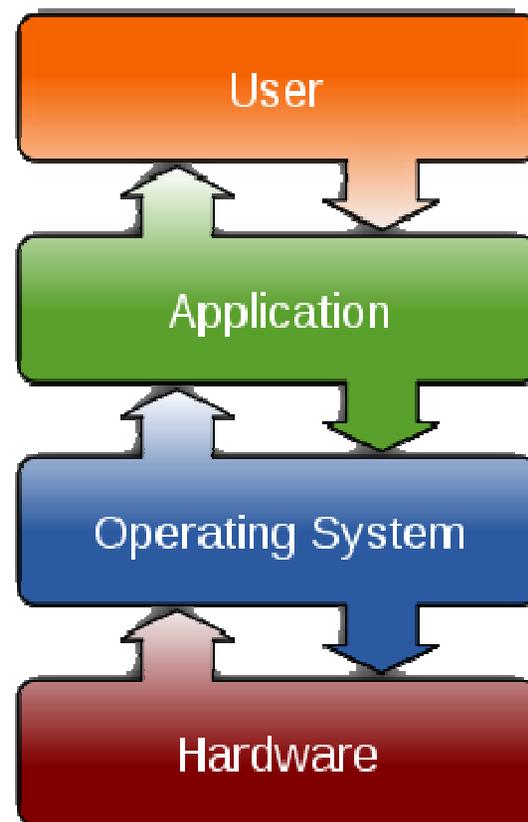
Name	Base	Length	Space	Access
DDR	0xC0000000	0x08000000	code/data	RWX

The 'Memory Sections' section shows the following configuration:

- Code Memory: IRAM
- Data Memory: IRAM
- Stack Memory: IRAM

Outline

- ◆ **Intro to SYS/BIOS**
 - ◆ Overview
 - ◆ Threads and Scheduling
 - ◆ Creating a BIOS Thread
 - ◆ System Timeline
 - ◆ Real-Time Analysis Tools
 - ◆ Create A New Project
 - ◆ BIOS Configuration (.CFG)
 - ◆ Platforms
 - ◆ For More Information
- ◆ **BIOS Threads**



For More Information (1)

- ◆ [SYS/BIOS Product Page http://www.ti.com/sysbios](http://www.ti.com/sysbios)

SYS/BIOS Real-Time Operating System (RTOS) Status

: ACTIVE
SYSBIOS

[Description/Features](#) [Technical Documents](#) [Support & Community](#)

Order Now

Part Number	Texas Instruments	Status	P
SYSBIOS6: SYS/BIOS 6.x Real-Time Operating System (previously DSP/BIOS v6)	Get Software	ACTIVE	P

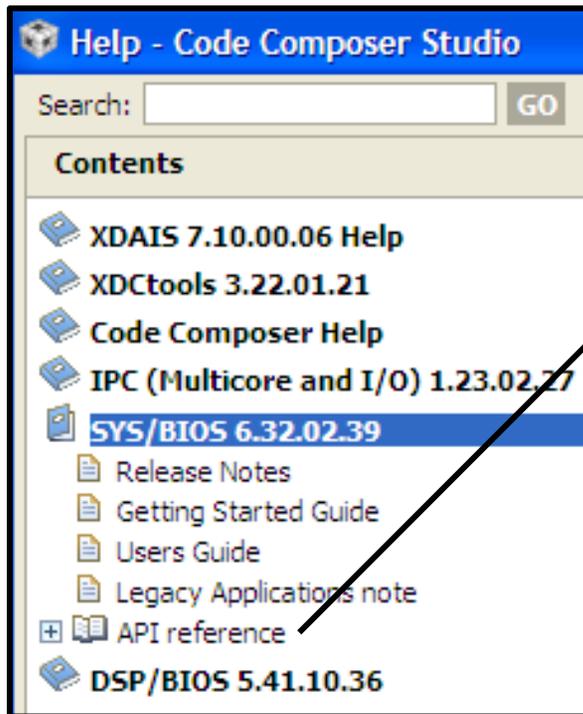
Description

Advanced RTOS Solution

SYS/BIOS™ 6.x is an advanced, real-time operating system for use in a wide range of DSPs, ARMs, and microcontrollers. It is designed for use in embedded applications that need real-time scheduling, synchronization, and instrumentation. It provides preemptive multitasking, hardware abstraction, and memory management. Compared to its predecessor, DSP/BIOS™ 5.x, it has numerous enhancements in functionality and performance.

For More Information (2)

◆ CCS Help Contents



- User Guides
- API Reference (knl)

Contents

- ti.sysbios.heaps
- ti.sysbios.interfaces
- ti.sysbios.knl**
 - Clock
 - Event
 - Idle
 - Mailbox
 - Semaphore
 - Swi
 - Task
- ti.sysbios.rta
- ti.sysbios.syncs
 - ti.sysbios.timers
- ti.sysbios.timers.dmtim
- ti.sysbios.timers.gptim
- ti.sysbios.timers.timer
- ti.sysbios.utils
 - Load
- xdc.runtime
 - Assert
 - Defaults
 - Diags
 - Error
 - Gate
 - GateNull
 - HeapMin
 - HeapStd
 - IFilterLogger
 - IGateProvider
 - IHeap
 - IInstance
 - ILogger
 - IModule

SYS/BIOS 6.32.02.39

```
package ti.sysbios.knl
```

Contains core threading modules

Many real-time applications must perform a such as the availability of data or the prese important. [[more ...](#)]

XDCspec declarations

```
requires ti.sysbios.interfaces;
requires ti.sysbios.family;

package ti.sysbios.knl [2, 0, 0, 0] {

    module Clock;
        // System Clock Manager
    module Event;
        // Event Manager
    module Idle;
        // Idle Thread Manager
    module Mailbox;
        // Mailbox Manager
    module Semaphore;
        // Semaphore Manager
    module Swi;
        // Software Interrupt Manager
    module Task;
        // Task Manager
}
```

Download Latest Tools

◆ Download Target Content

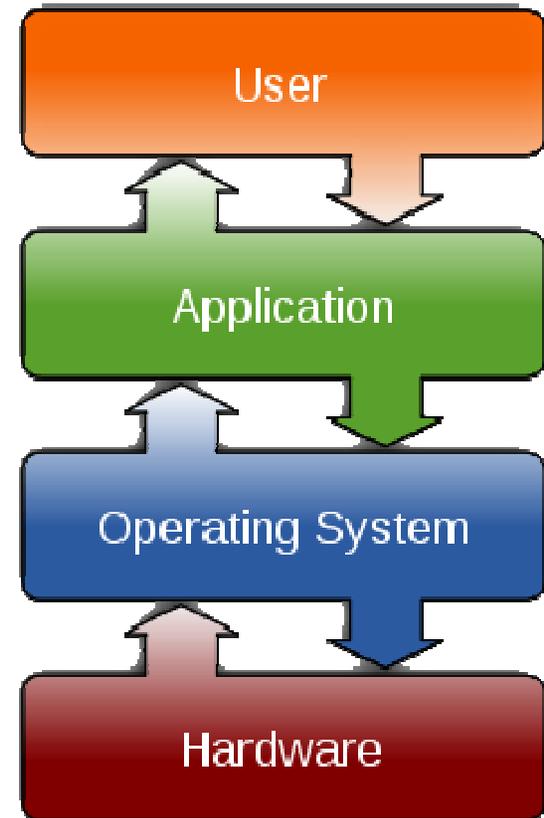
http://software-dl.ti.com/dsp/dsp_public_sw/sdo_sb/targetcontent/

Target Content Infrastructure Product Downloads
BIOS Platform Support Packages
DSP/BIOS and SYS/BIOS
DSP/BIOS BIOSUSB Product
DSP/BIOS Utilities
Digital Video Software Development Kits (DVSDK)
DSP Link and SysLink <ul style="list-style-type: none">• SysLink (BIOS 6)• DSP Link (BIOS 5)
Graphics SDK
EDMA3 Low-level Driver
Interprocessor Communication (IPC)

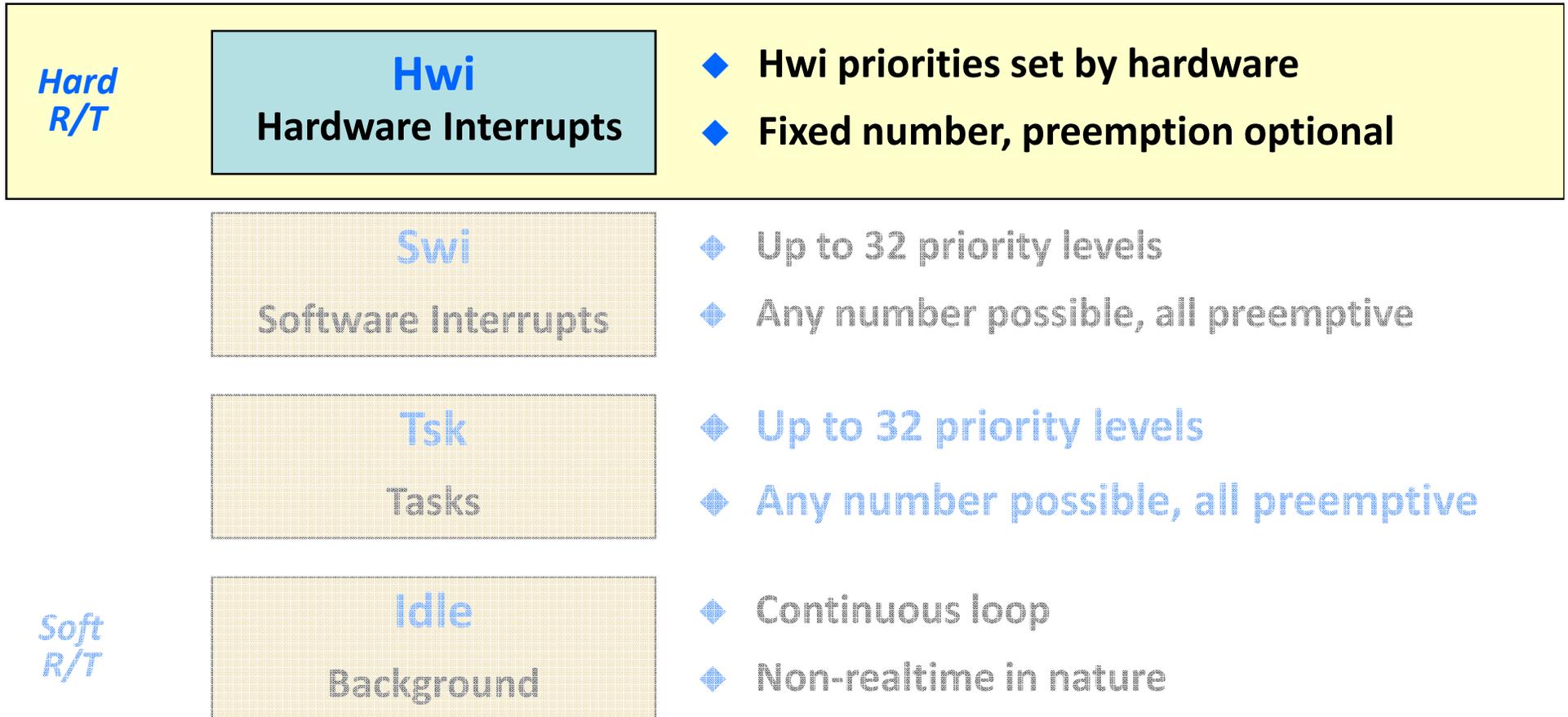
- ◆ DSP/BIOS
- ◆ SYS/BIOS
- ◆ Utilities
- ◆ SysLink
- ◆ DSP Link
- ◆ IPC
- ◆ Etc.

Outline

- ◆ Intro to SYS/BIOS
- ◆ BIOS Threads
 - ◆ Hardware Interrupts (Hwi)
 - ◆ Software Interrupts (Swi)
 - ◆ Tasks (Tsk)
 - ◆ Semaphores (Sem)

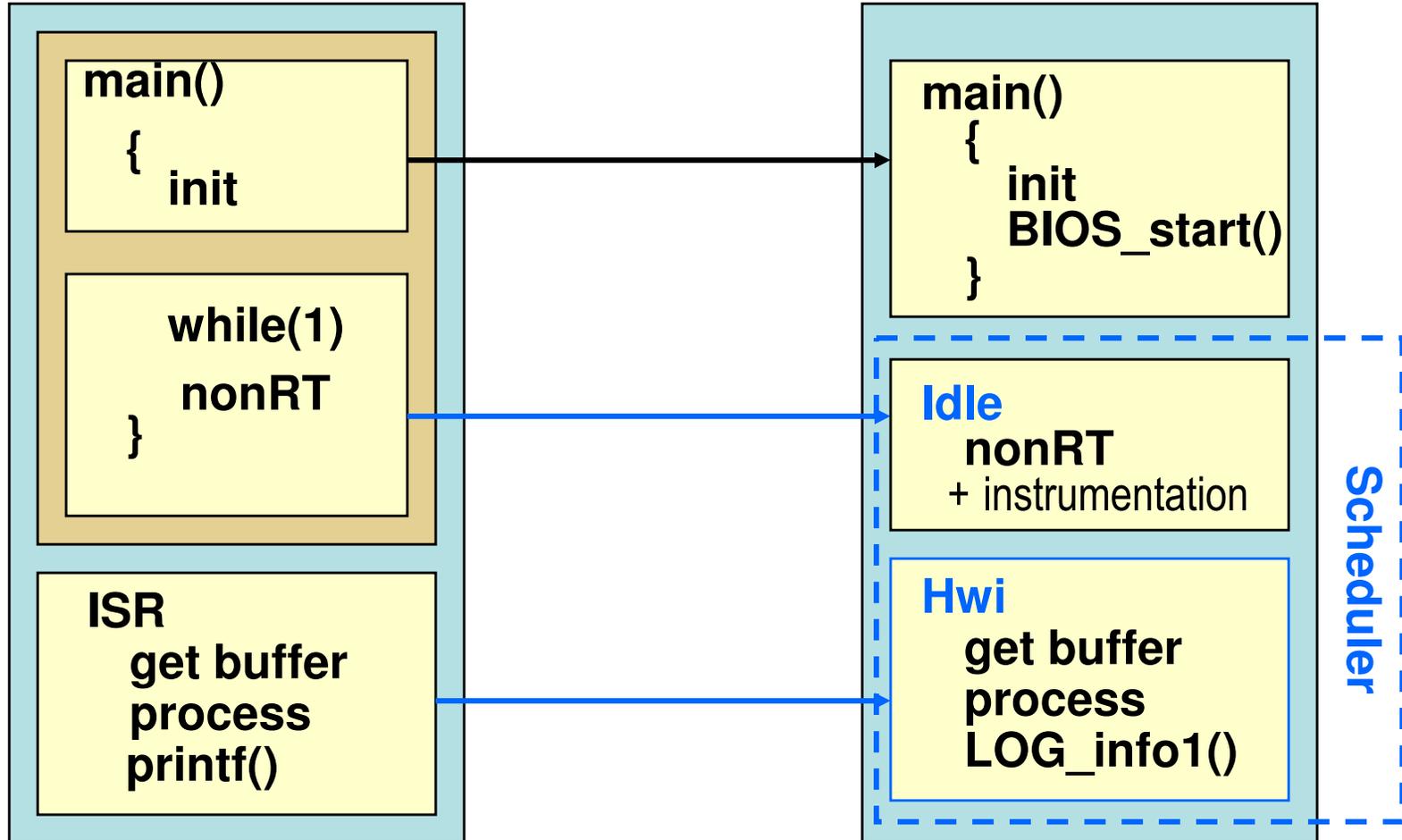


Hwi Scheduling



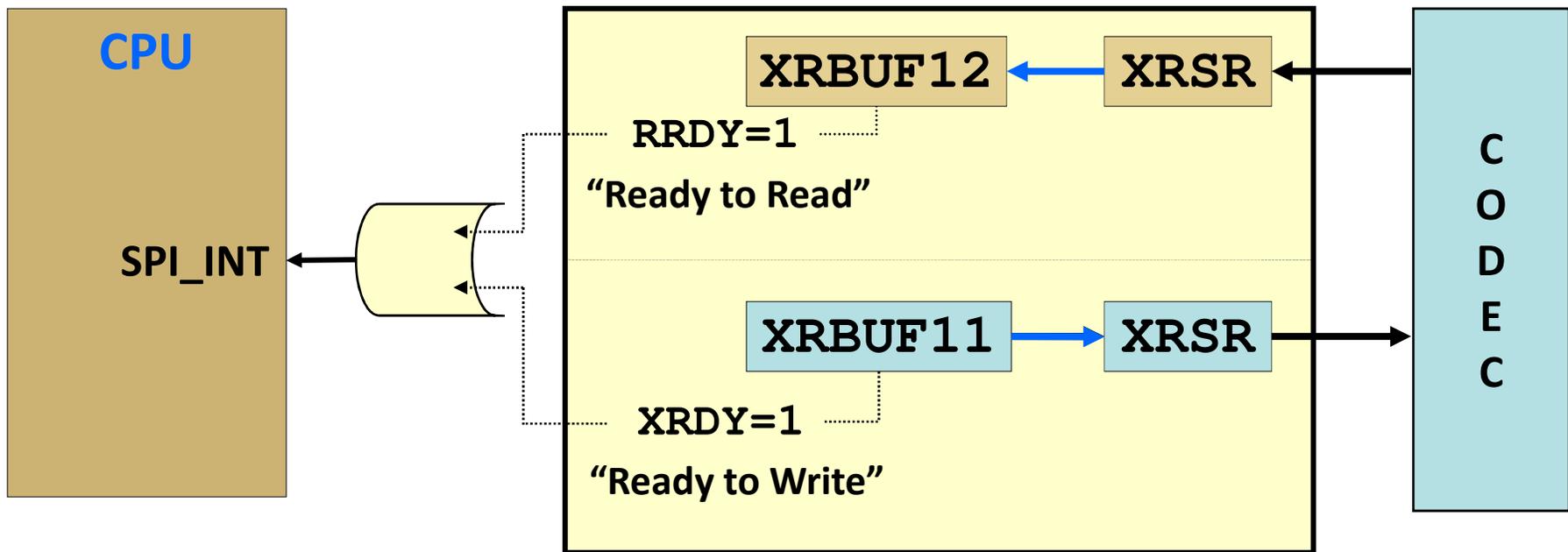
- ◆ **Idle** events run in sequence when no **Hwi** are posted.
- ◆ **Hwi** is ISR with automatic vector table generation + context save/restore.
- ◆ Any **Hwi** preempts **Idle**, **Hwi** may preempt other **Hwi** if desired.

Foreground / Background Scheduling



- ◆ **Idle** events run in sequence when no **Hwi** are posted.
- ◆ **Hwi** is ISR with automatic vector table generation + context save/restore.
- ◆ Any **Hwi** preempts **Idle**, **Hwi** may preempt other **Hwi** if desired.

CPU Interrupts from Peripheral (SPI)



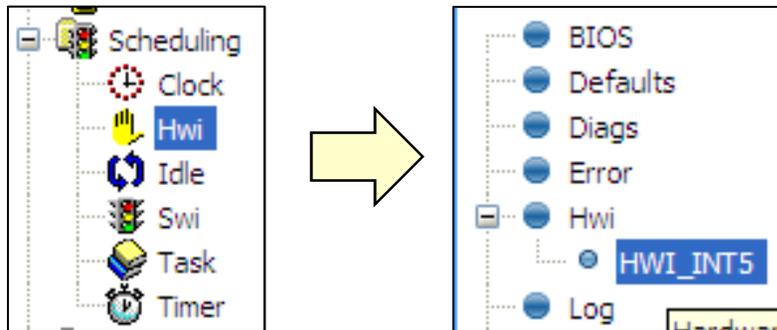
- ◆ A peripheral (e.g., SPI on C6678) causes an interrupt to the CPU to indicate “service required.”
- ◆ This “event” will have an ID (datasheet) and can be tied to a specific CPU interrupt (target specific).

How do we configure SYS/BIOS to respond to this interrupt and call the appropriate ISR?

Configuring an Hwi: Statically via GUI

Example: Tie SPI_INT to the CPU HWI₅

- 1 Use Hwi module (Available Products), insert new Hwi (Outline View)



NOTE: BIOS objects can be created via the GUI, script code, or C code (dynamic).

- 2 Configure Hwi: Event ID, CPU Int #, ISR vector:

Generic Hardware Interrupt Instance

Basic | Advanced

Basic Settings

Name: HWI_INT5
ISR function: isrAudio
Interrupt Number: 5

Interrupt Scheduling Options

Interrupts to mask: MaskingOption_SELF
Priority: 5
Event Id: 61
 Enabled at startup

To enable INT at startup, check the box

Where do you find the Event Id #?

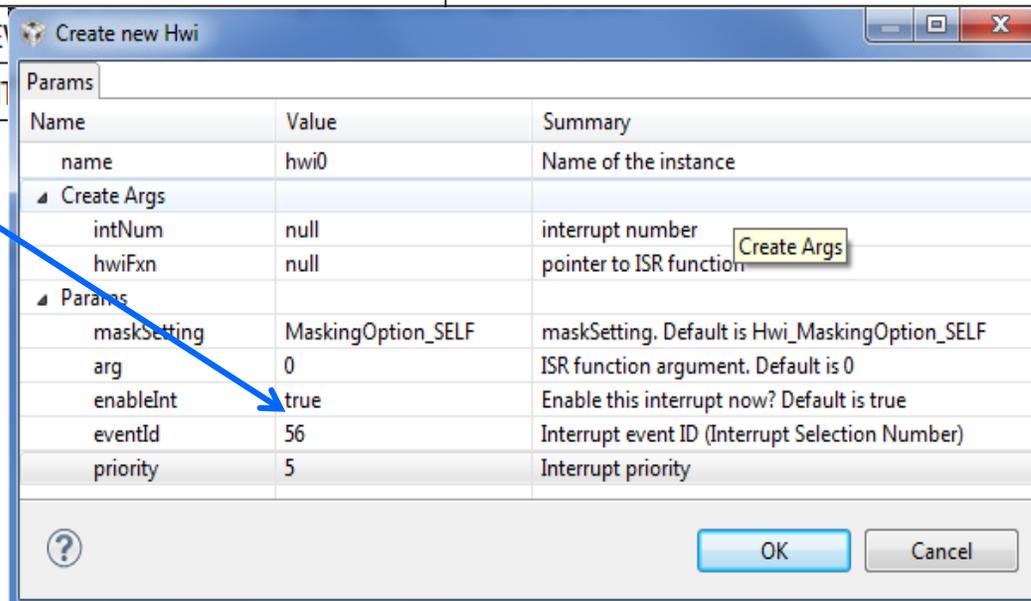
Hardware Event IDs

- ◆ How do you know the names of the interrupt events and their corresponding event numbers?

Look it up in the datasheet.

Source: TMS320C6678 datasheet

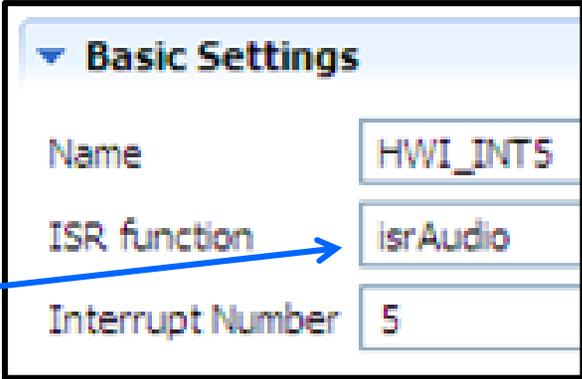
52	PCIExpress_Legacy_INTC	Legacy interrupt mode
53	PCIExpress_Legacy_INTD	Legacy interrupt mode
54	SPIINT0	SPI interrupt0
55	SPIINT1	SPI interrupt1
56	SPIXEVT	Transmit event
57	SPIRE	
58	I2CINT	



- ◆ As appropriate, refer to the datasheet for your target platform.

Example ISR (SPI)

Example ISR for SPIXEV_INT interrupt



Basic Settings	
Name	HWI_INT5
ISR function	isrAudio
Interrupt Number	5

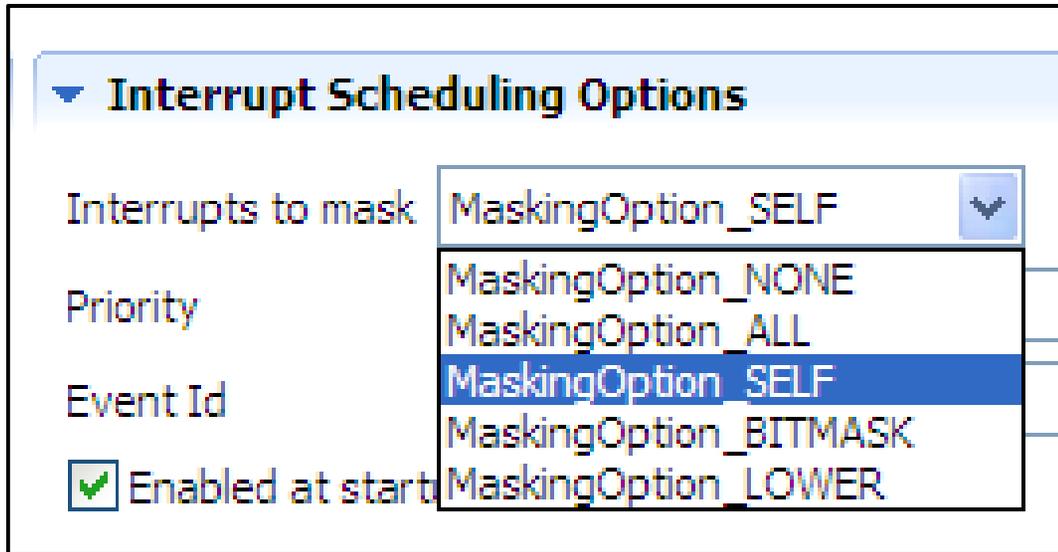
isrAudio:

```
pInBuf[blkCnt] = SPI->RCV;           // READ audio sample from SPI
SPI->XMT = pOutBuf[blkCnt]           // WRITE audio sample to SPI
blkCnt+=1;                           // increment blk counter

if( blkCnt >= BUFFSIZE )
{
    memcpy(pOut, pIn, Len);           // Copy pIn to pOut (Algo)
    blkCnt = 0;                       // reset blkCnt for new buf's
    pingPong ^= 1;                    // PING/PONG buffer boolean
}
```

Can one interrupt preempt another?

Enabling Preemption of Hwi



- ◆ **Default** mask is *SELF*, which means all other Hwi activities can pre-empt except for itself.
- ◆ Can choose other masking options as required:

ALL:	Best choice if ISR is short & fast
NONE:	Dangerous; Make sure ISR code is re-entrant.
BITMASK:	Allows custom mask
LOWER:	Masks any interrupt(s) with lower priority (ARM)

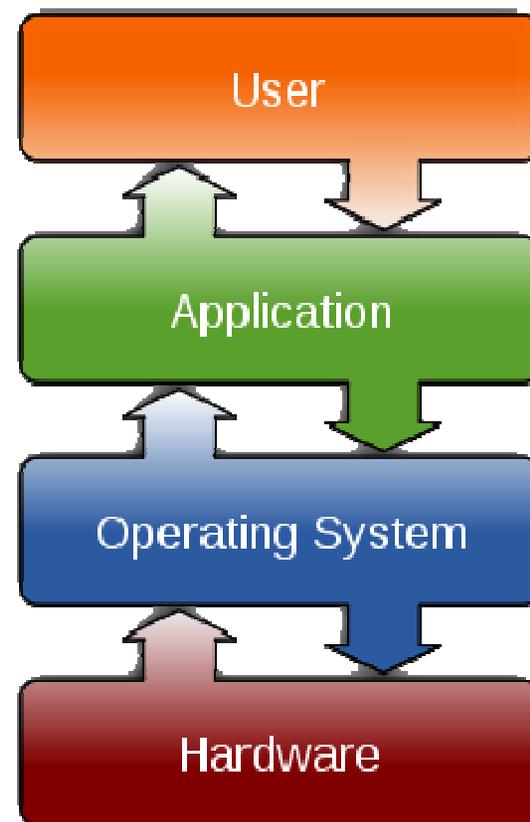
SYS/BIOS Hwi APIs

Other useful Hwi APIs:

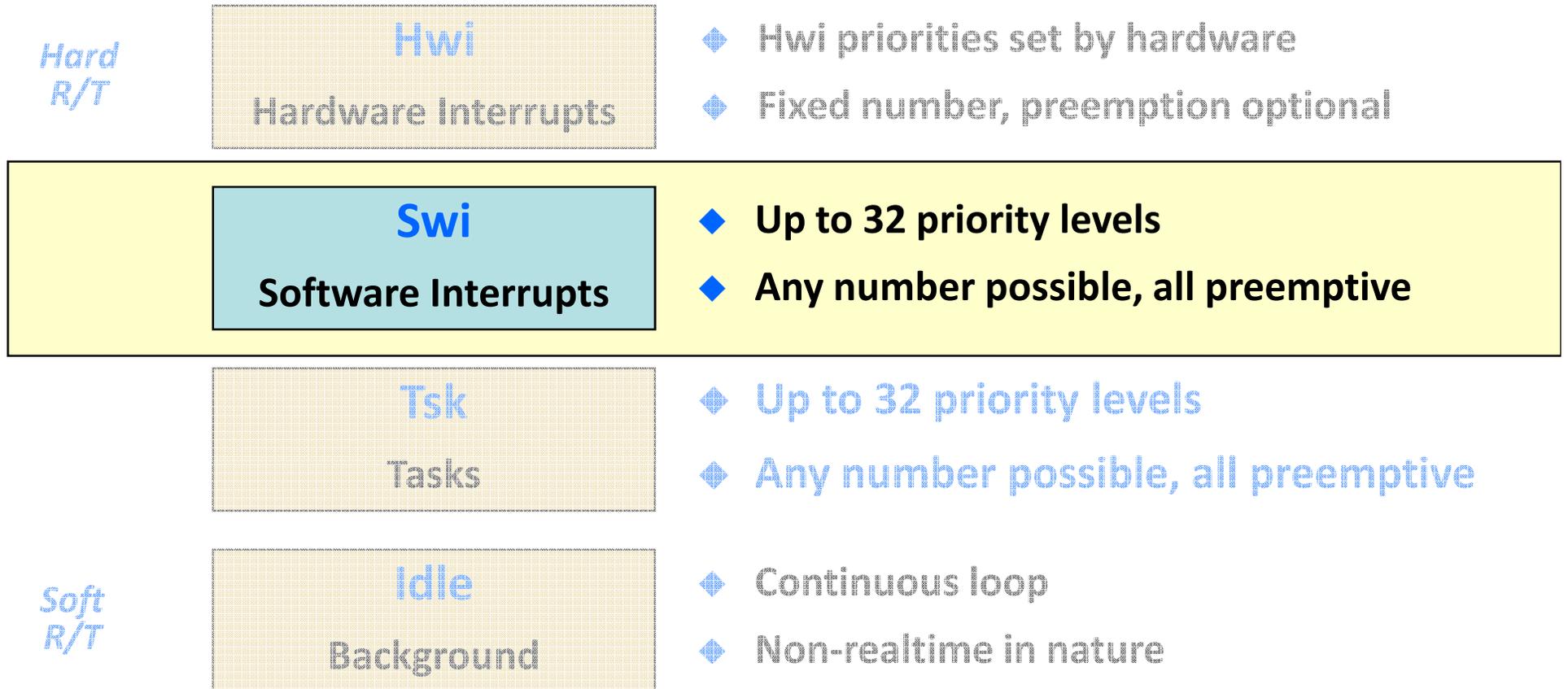
<code>Hwi_disableInterrupt ()</code> <code>Hwi_enableInterrupt ()</code> <code>Hwi_clearInterrupt ()</code>	Set enable bit = 0 Set enable bit = 1 Clear INT flag bit = 0
<code>Hwi_post ()</code> New in SYS/BIOS	Post INT # (in code)
<code>Hwi_disable ()</code> <code>Hwi_enable ()</code> <code>Hwi_restore ()</code>	Global INTs disable Global INTs enable Global INTs restore

Outline

- ◆ Intro to SYS/BIOS
- ◆ BIOS Threads
 - ◆ Hardware Interrupts (Hwi)
 - ◆ Software Interrupts (Swi)
 - ◆ Tasks (Tsk)
 - ◆ Semaphores (Sem)



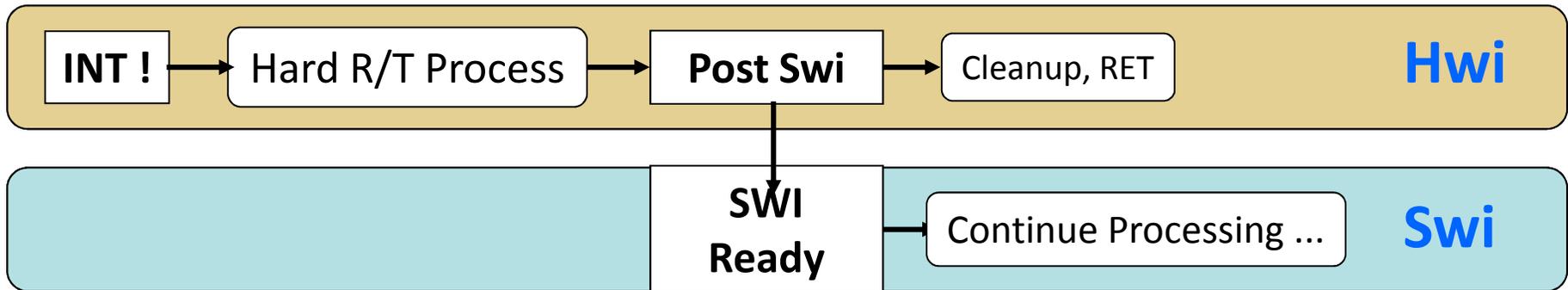
Swi Scheduling



- ◆ SYS/BIOS provides for Hwi and Swi management.
- ◆ SYS/BIOS allows the Hwi to post a Swi to the ready queue.

Hardware and Software Interrupt System

Execution flow for flexible real-time systems:



Hwi

- ◆ Fast response to INTs
- ◆ Min context switching
- ◆ High priority for CPU
- ◆ Limited # of Hwi possible

isrAudio:

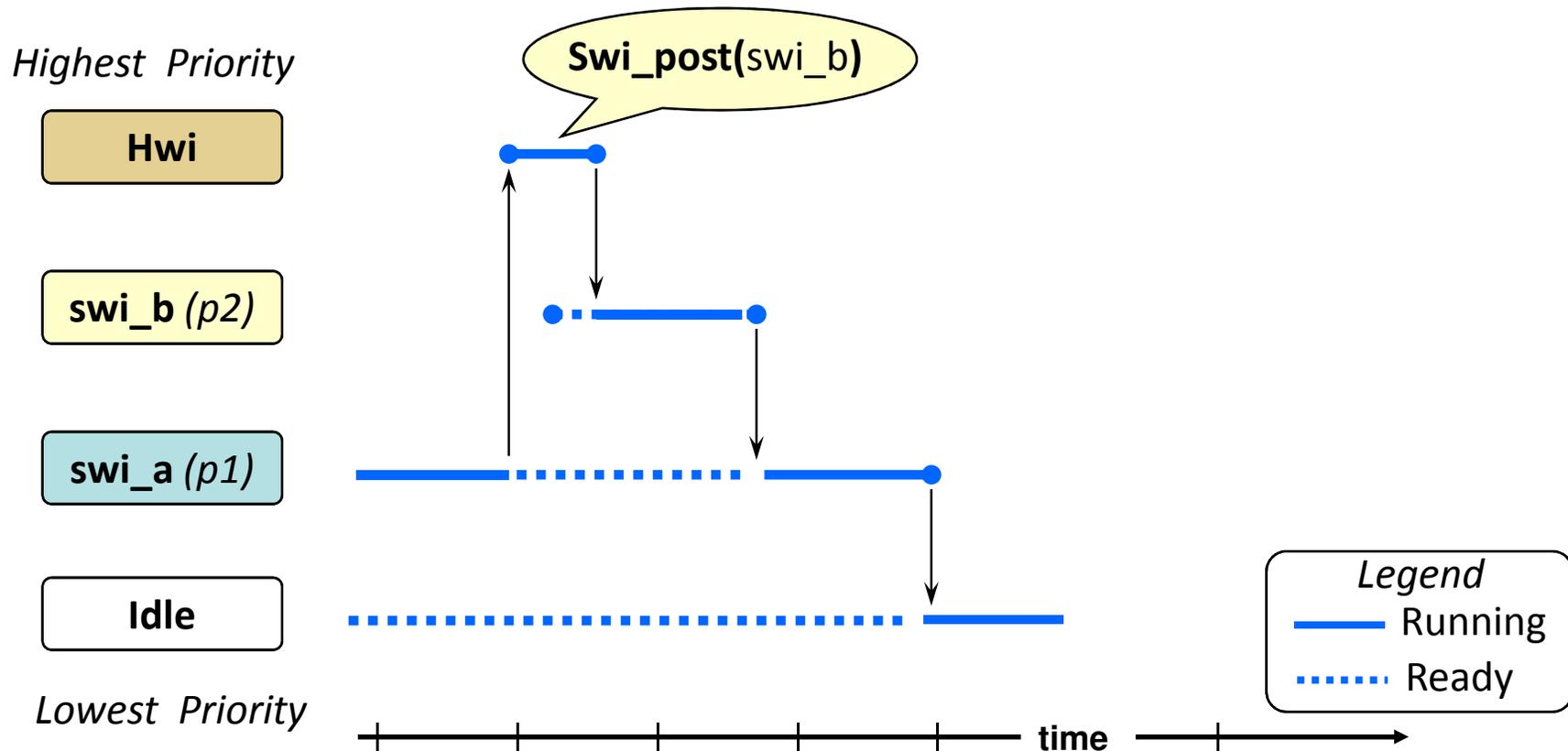
```
*buf++ = *XBUF;  
cnt++;  
if (cnt >= BLKSZ) {  
    Swi_post(swiFir);  
    count = 0;  
    pingPong ^= 1;  
}
```

Swi

- ◆ Latency in response time
- ◆ Context switch
- ◆ Selectable priority levels
- ◆ Scheduler manages execution

- ◆ SYS/BIOS provides for Hwi and Swi management.
- ◆ SYS/BIOS allows the Hwi to post a Swi to the ready queue.

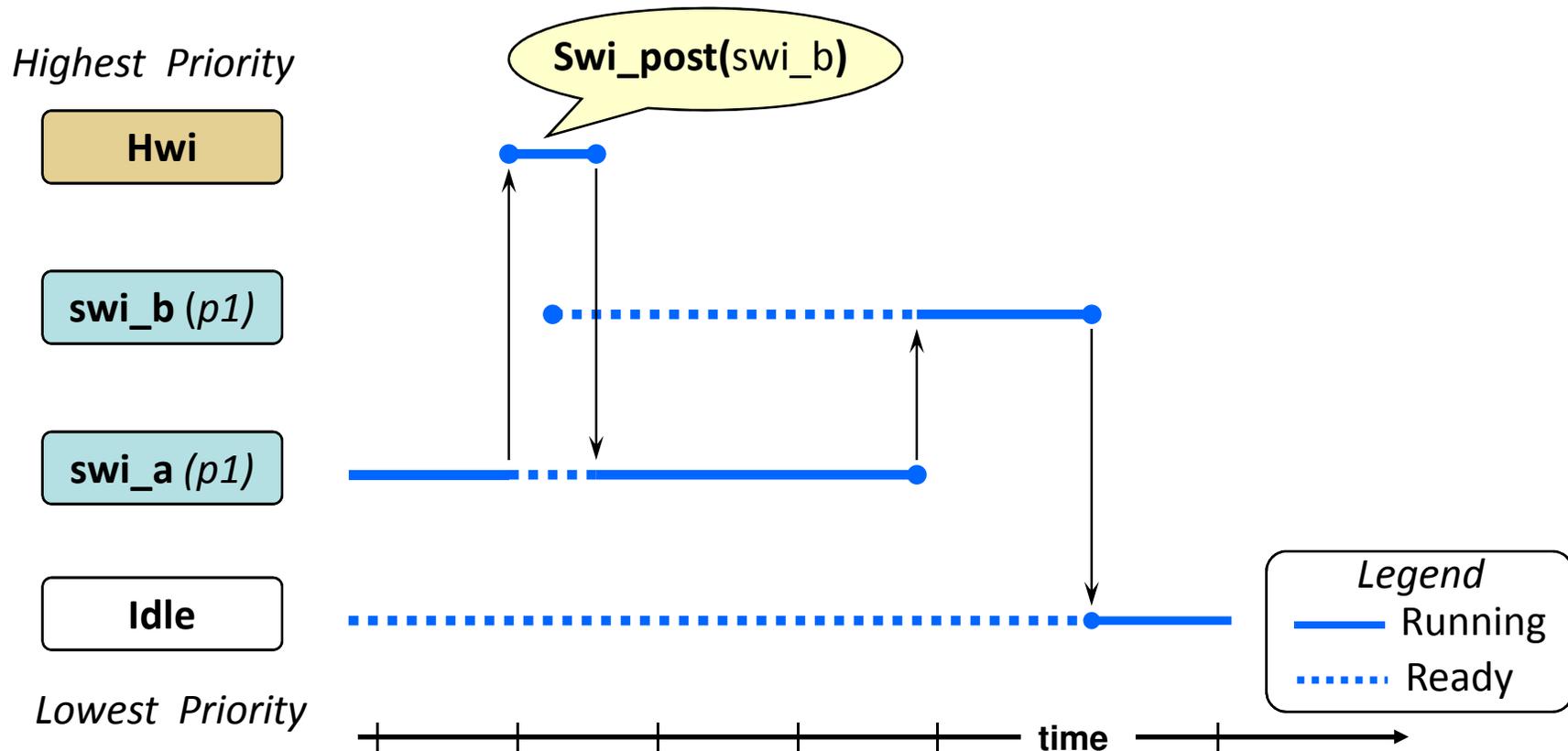
Scheduling Rules



- ◆ **Swi_post(mySwi)** : Unconditionally post a software interrupt (in the ready state).
- ◆ If a higher priority thread becomes ready, the running thread is preempted.
- ◆ **Swi** priorities range from 1 to 32.
- ◆ Automatic context switch (uses system stack)

What if more than one Swi process is set to the same priority?

Scheduling Rules

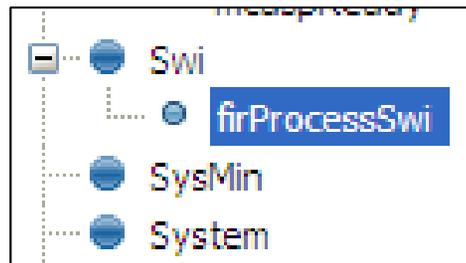
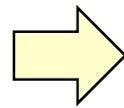
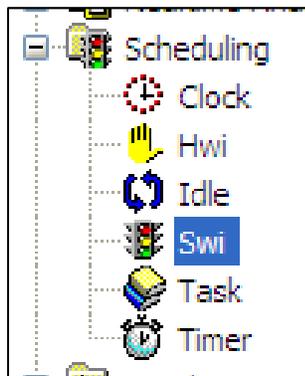


- ◆ Processes of same priority are scheduled first-in first-out (FIFO).
- ◆ Having threads at the SAME priority offers certain advantages, such as resource sharing (without conflicts).

Configuring a Swi: Statically via GUI

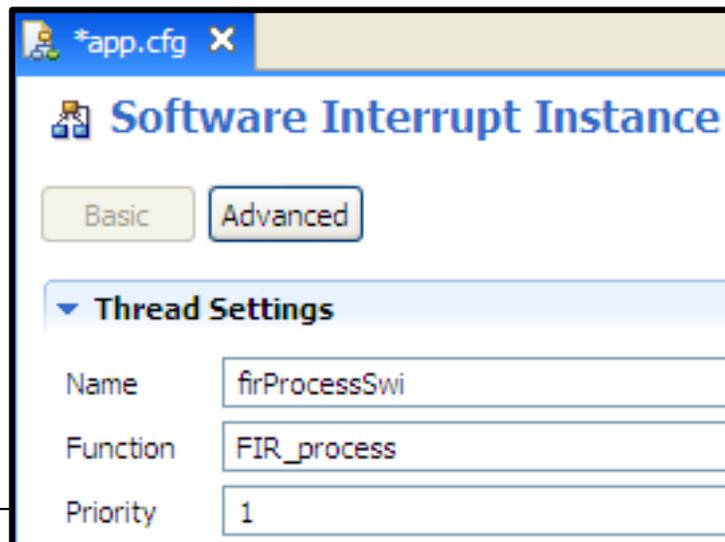
Example: Tie isrAudio() fxn to Swi; Use priority 1

- 1 Use Swi module (Available Products), insert new Hwi (Outline View)



NOTE: BIOS objects can be created via the GUI, script code, or C code (dynamic).

- 2 Configure Swi – Object name, function, priority:



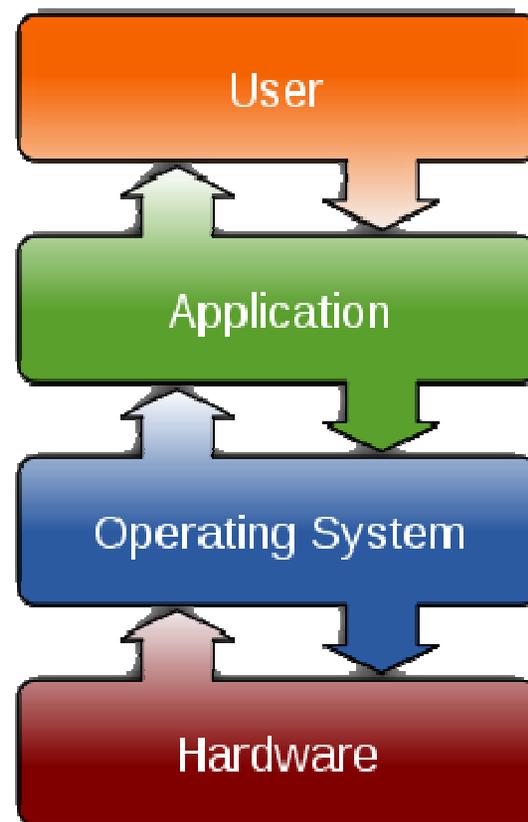
SYS/BIOS Swi APIs

Other useful Swi APIs:

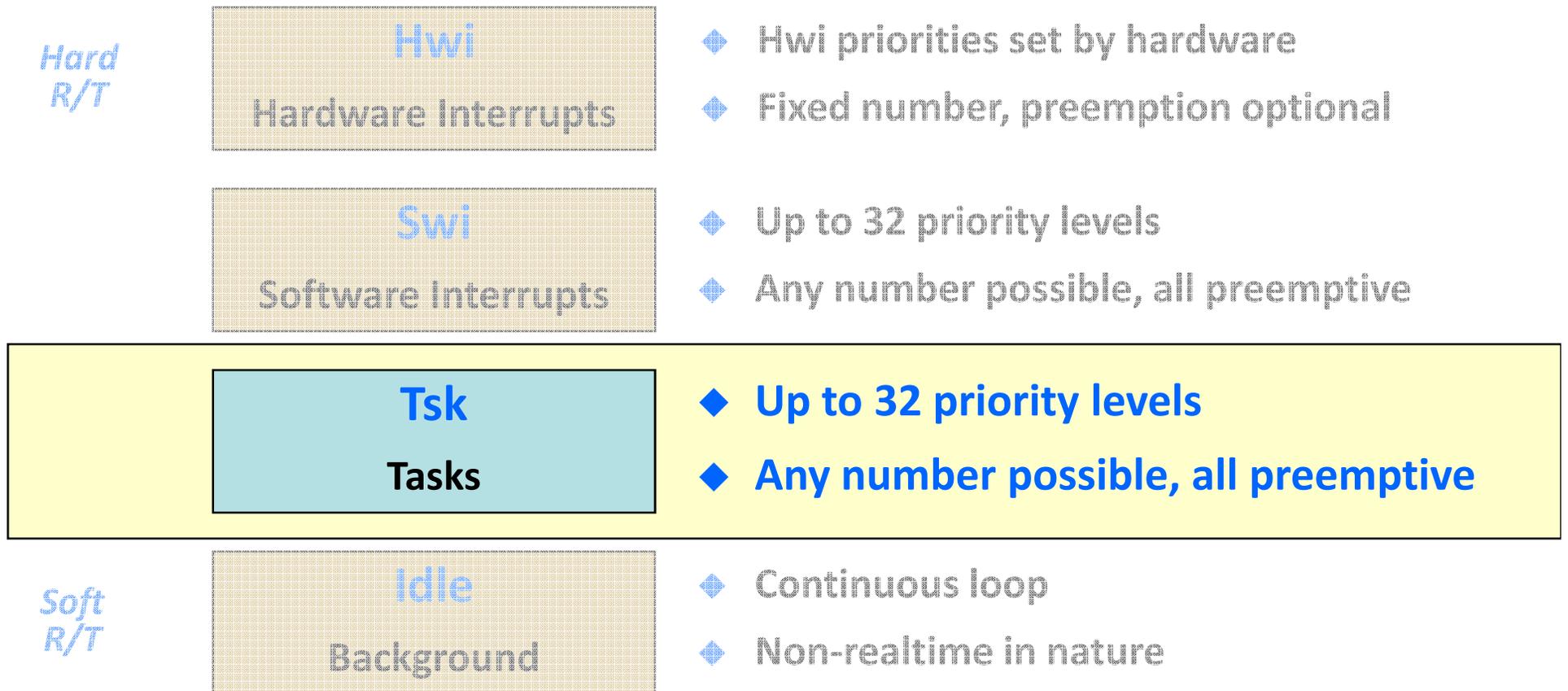
Swi_inc()	Post, increment count
Swi_dec()	Decrement count, post if 0
Swi_or()	Post, OR bit (signature)
Swi_andn()	ANDn bit, post if all posted
Swi_getPri()	Get any Swi Priority
Swi_enable	Global Swi enable
Swi_disable()	Global Swi disable
Swi_restore()	Global Swi restore

Outline

- ◆ Intro to SYS/BIOS
- ◆ BIOS Threads
 - ◆ Hardware Interrupts (Hwi)
 - ◆ Software Interrupts (Swi)
 - ◆ **Tasks (Tsk)**
 - ◆ Semaphores (Sem)

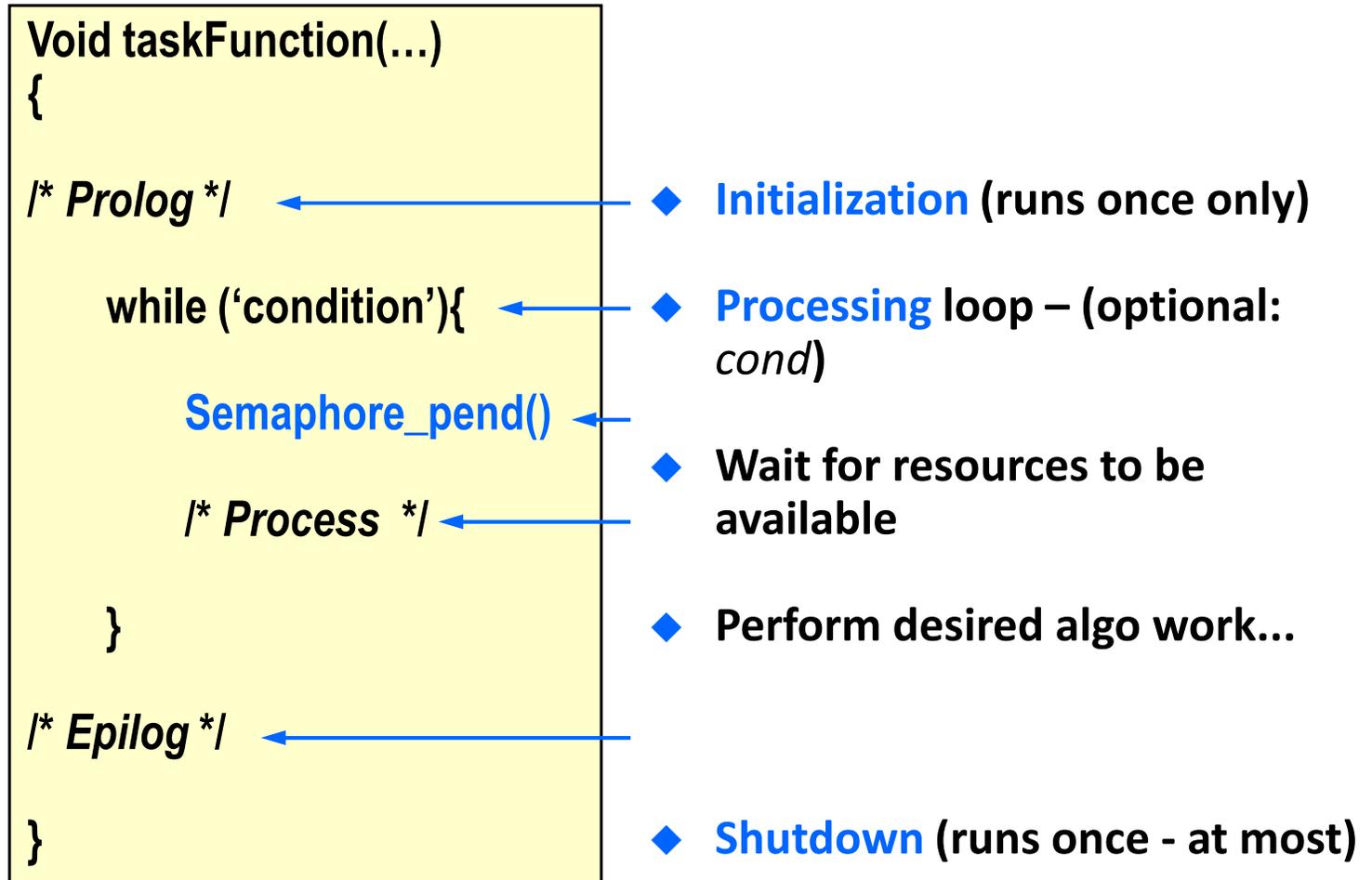


Task Scheduling



- ◆ All Tasks are preempted by all Swi and Hwi.
- ◆ All Swi are preempted by all Hwi.
- ◆ Preemption amongst Hwi is determined by user.
- ◆ In absence of Hwi, Swi, and Tsk, Idle functions run in loop.

Task Code Topology – Pending



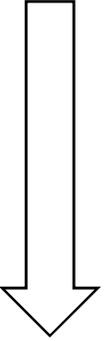
- ◆ Task can encompass *three* phases of activity.
- ◆ Semaphore can be used to signal resource availability to Task.
- ◆ **Semaphore_pend ()** *blocks* Task until semaphore (flag) is posted.

Comparing Swi and Task

Swi

`_post` →

```
void mySwi () {  
    // set local env  
  
    *** RUN ***  
}
```



- “Ready” when POSTED
- Initial state NOT preserved; Must set each time **Swi** is run
- CanNOT block (runs to completion)
- Context switch speed (~140c)
- All **Swi** share system stack w/ Hwi
- Usage: As follow-up to Hwi and/or when memory size is an absolute premium

Task

`_create` →

```
void myTask () {  
    // Prologue (set Task env)  
    while(cond) {  
        Semaphore_pend();  
        *** RUN ***  
    }  
    // Epilogue (free env)  
}
```

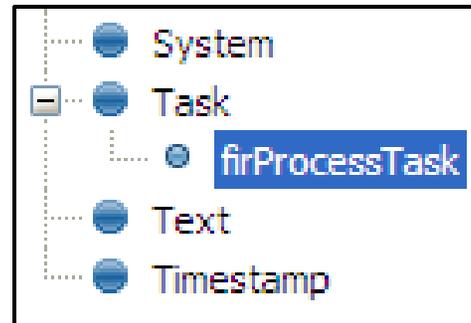
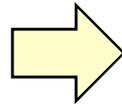
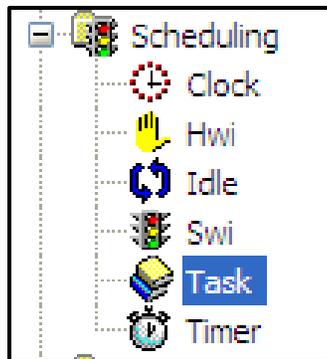


- “Ready” when CREATED (BIOS_start or dynamic)
- P-L-E structure handy for resource creation (P) and deletion (E), initial state preserved
- Can block/suspend on semaphore (flag)
- Context switch speed (~160c)
- Uses its OWN stack to store context
- Usage: Full-featured sys, CPU w/more speed/mem

Configuring a Task: Statically via the GUI

Example: Create `firProcessTask`, tie to `FIR_process()`, priority 2

- 1 Use Task module (*Available Products*), insert new Task (*Outline View*)



NOTE: BIOS objects can be created via the GUI, script code, or C code (dynamic).

- 2 Configure Task: Object name, function, priority, stack size

Thread Settings

Name:

Function:

Priority:

Use the vital flag to prevent system exit until this task completes.

Task is vital

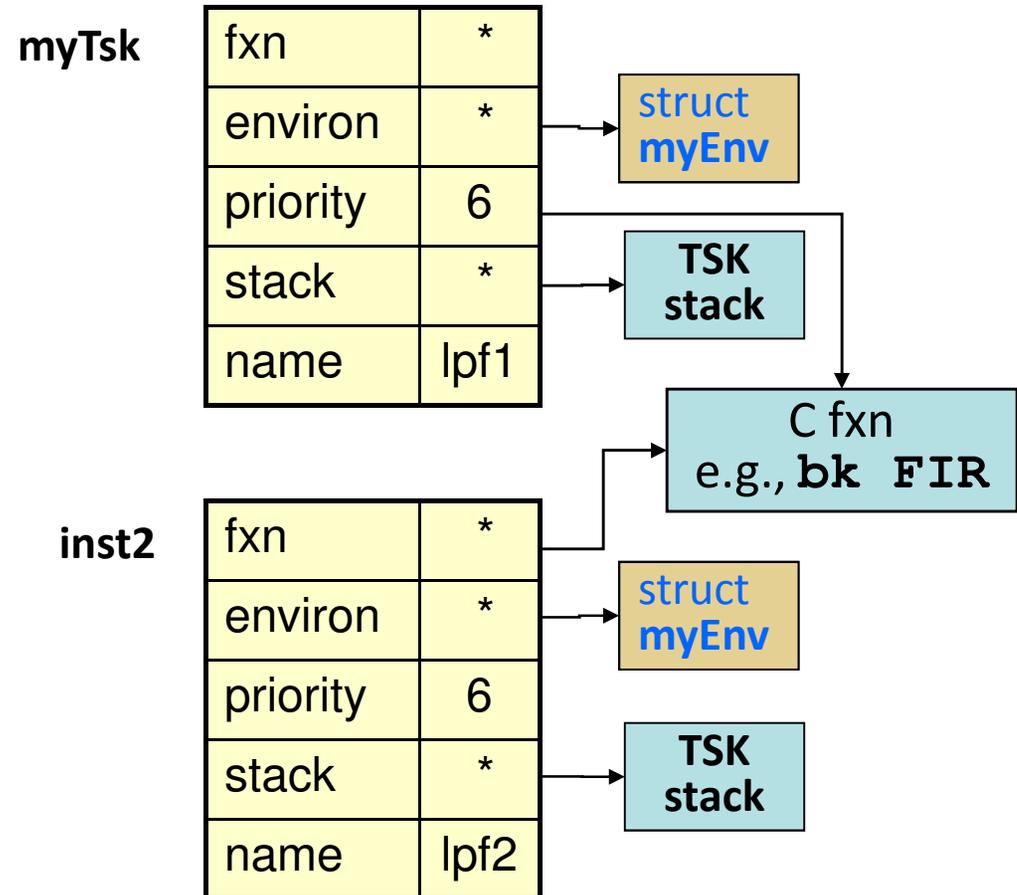
Stack Control Options

Stack size:

Task Object Concepts

Task object:

- ◆ Pointer to task function
- ◆ Priority: changable
- ◆ Pointer to task's stack
 - ◆ Stores local variables
 - ◆ Nested function calls
 - ◆ makes blocking possible
 - ◆ Interrupts run on the system stack
- ◆ Pointer to text name of TSK
- ◆ **Environment**: pointer to *user defined* structure:

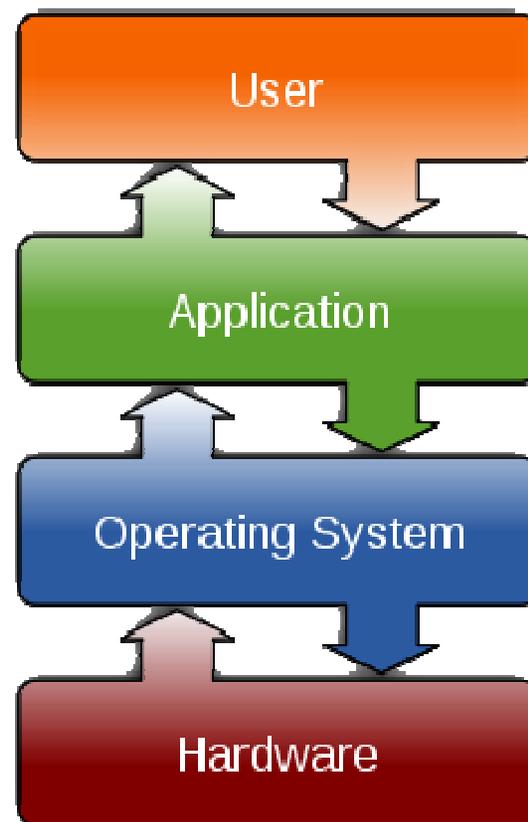


```
Task_setenv (Task_self () , &myEnv) ;
```

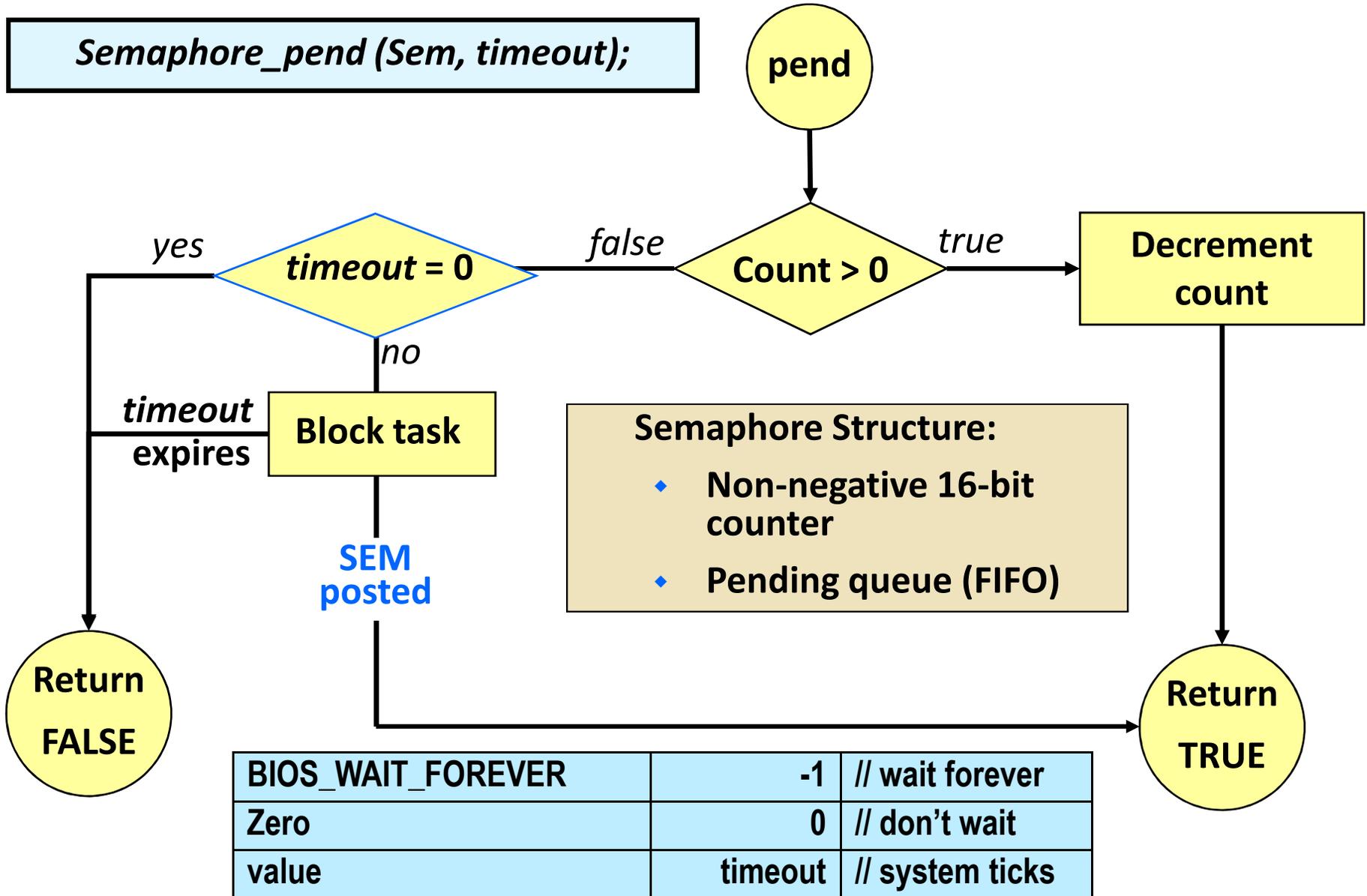
```
hMyEnv = Task_getenv (&myTsk) ;
```

Outline

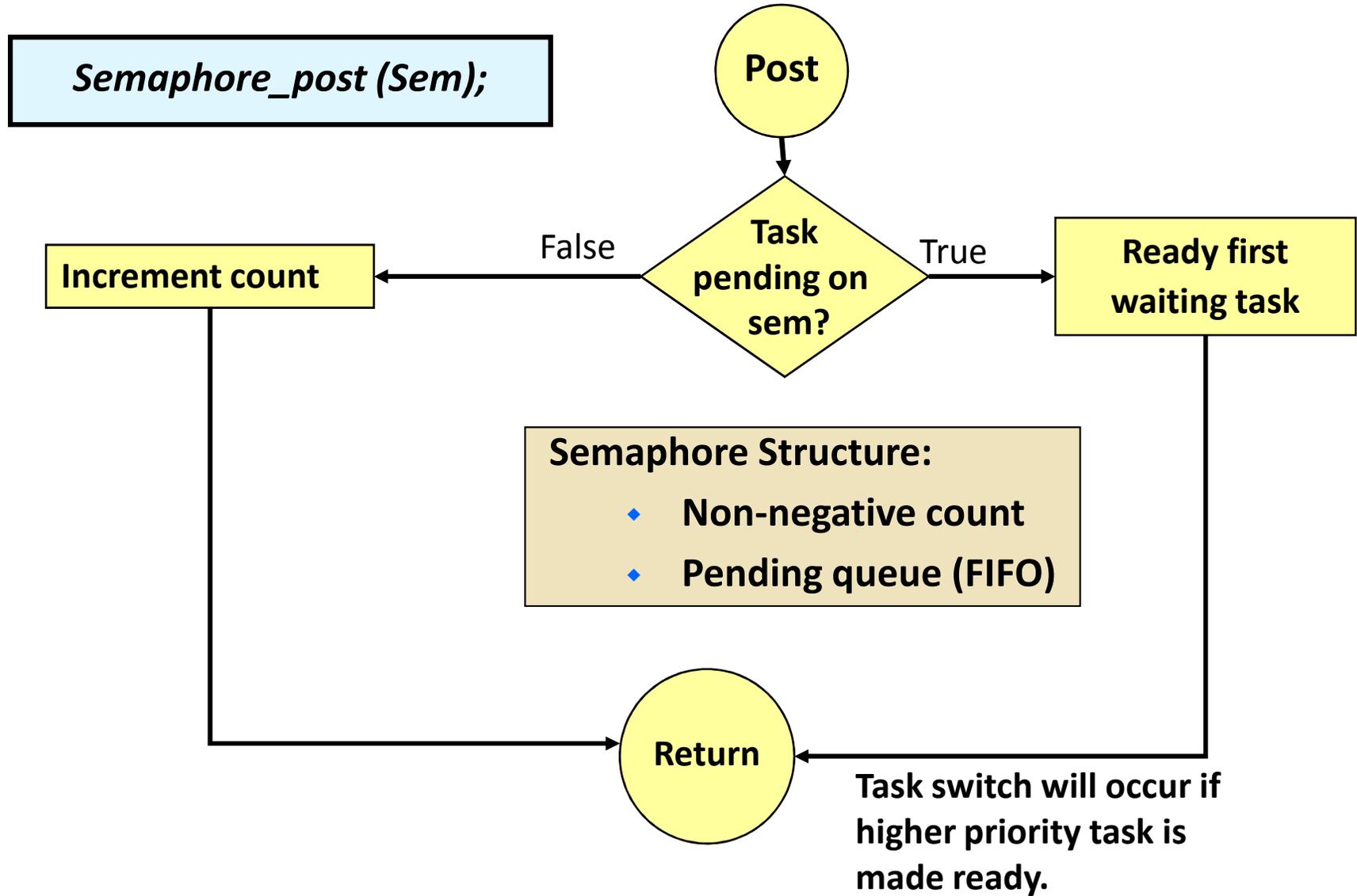
- ◆ Intro to SYS/BIOS
- ◆ BIOS Threads
 - ◆ Hardware Interrupts (Hwi)
 - ◆ Software Interrupts (Swi)
 - ◆ Tasks (Tsk)
 - ◆ Semaphores (Sem)



Semaphore Pend



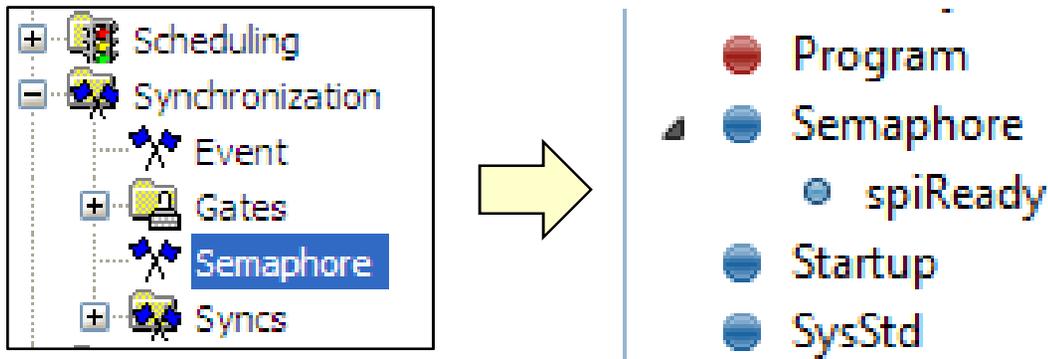
Semaphore Post



Configuring a Semaphore: Statically via GUI

Example: Create *spiReady*, counting

- 1 Use Semaphore (*Available Products*), insert new Semaphore (*Outline View*)



- 2 Configure Semaphore: Object name, initial count, type

Semaphore Instance - Basic Options

Basic Advanced

Required Settings

Name

Initial count

Semaphore type Counting semaphore Binary Semaphore

SYS/BIOS Semaphore/Task APIs

Other useful Semaphore APIs:

<code>Semaphore_getCount ()</code>	Get semaphore count
------------------------------------	---------------------

Other useful Task APIs:

<code>Task_sleep ()</code>	Sleep for N system ticks
<code>Task_yield ()</code>	Yield to same pri Task
<code>Task_setPri ()</code>	Set Task priority
<code>Task_getPri ()</code>	Get Task priority
<code>Task_get/setEnv ()</code>	Get/set Task Env
<code>Task_enable ()</code>	Enable Task Mgr
<code>Task_disable ()</code>	Disable Task Mgr
<code>Task_restore ()</code>	Restore Task Mgr

Questions?