Introduction to Free-RTOS

Deepak D'Souza

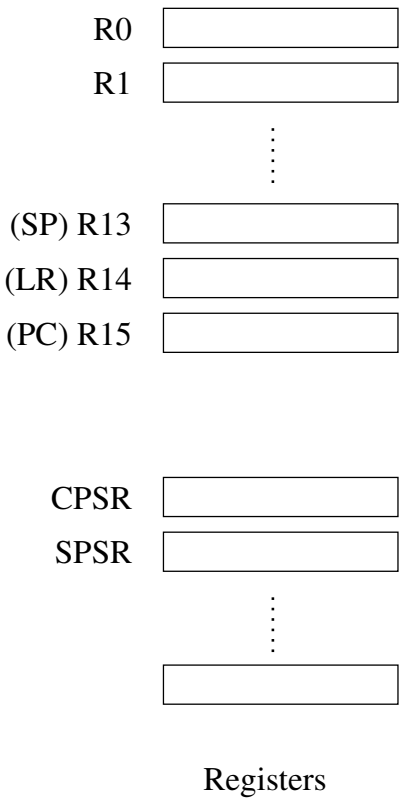
Department of Computer Science and Automation
Indian Institute of Science, Bangalore.

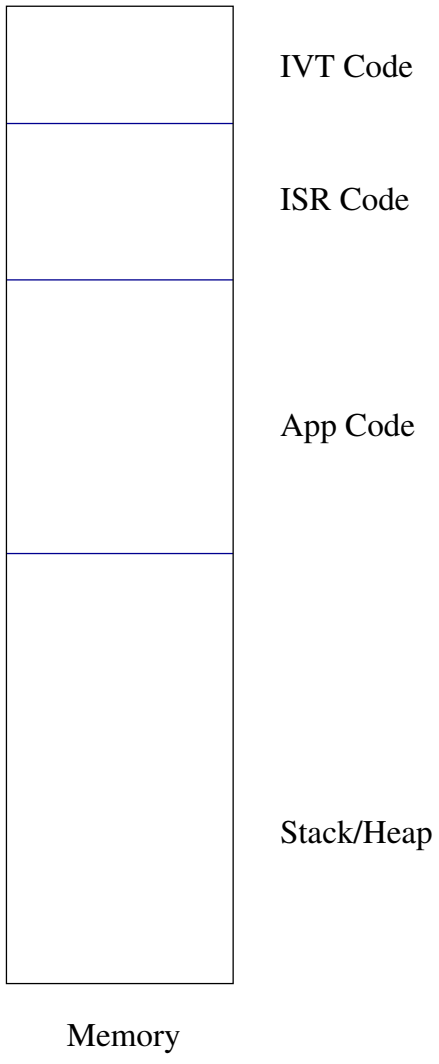
23 August 2011

Outline

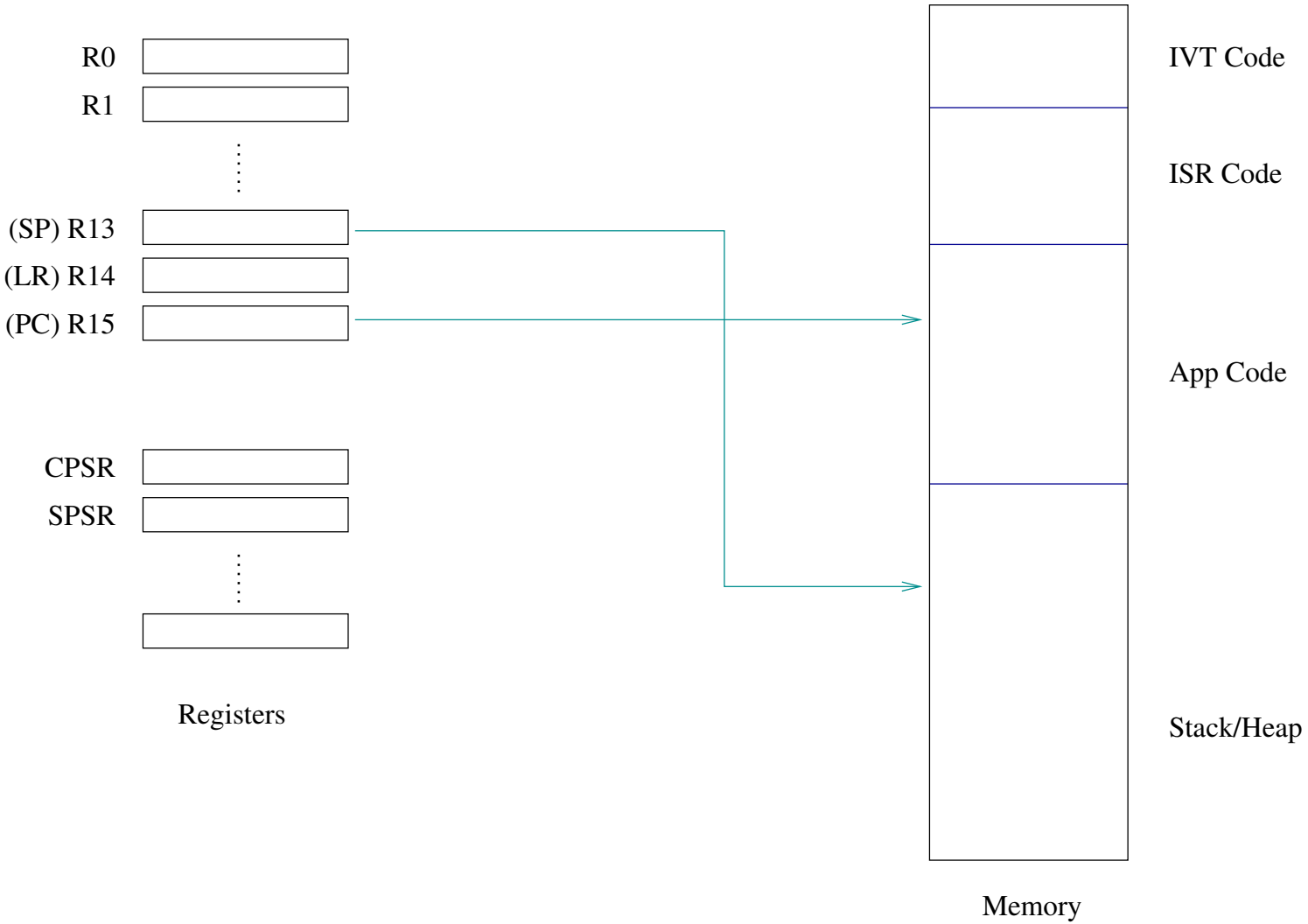
1. How a program runs on the ARM processor
2. How an application uses RTOS
3. What RTOS is expected to do

ARM processor architecture



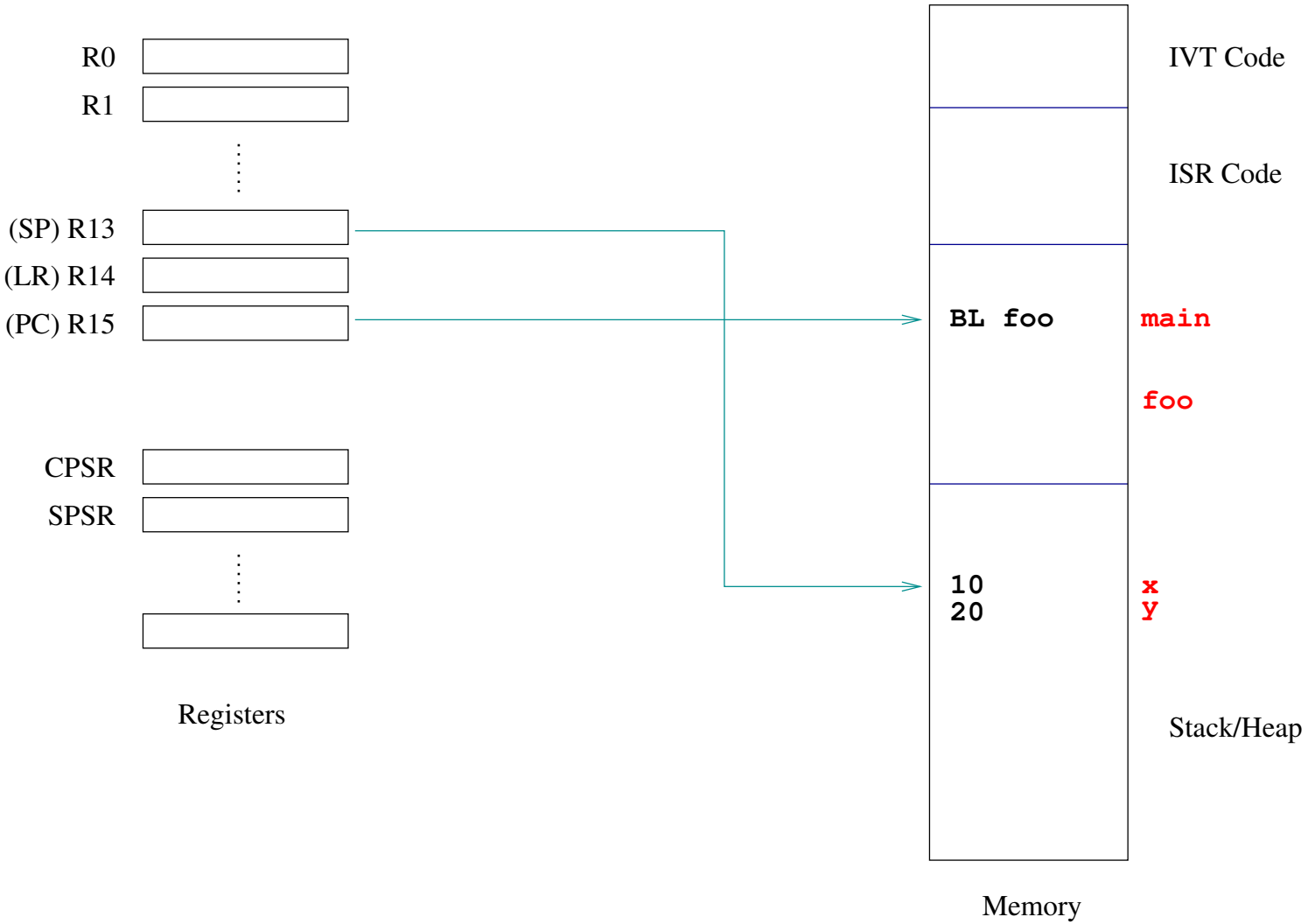


ARM processor architecture: SP and PC Registers



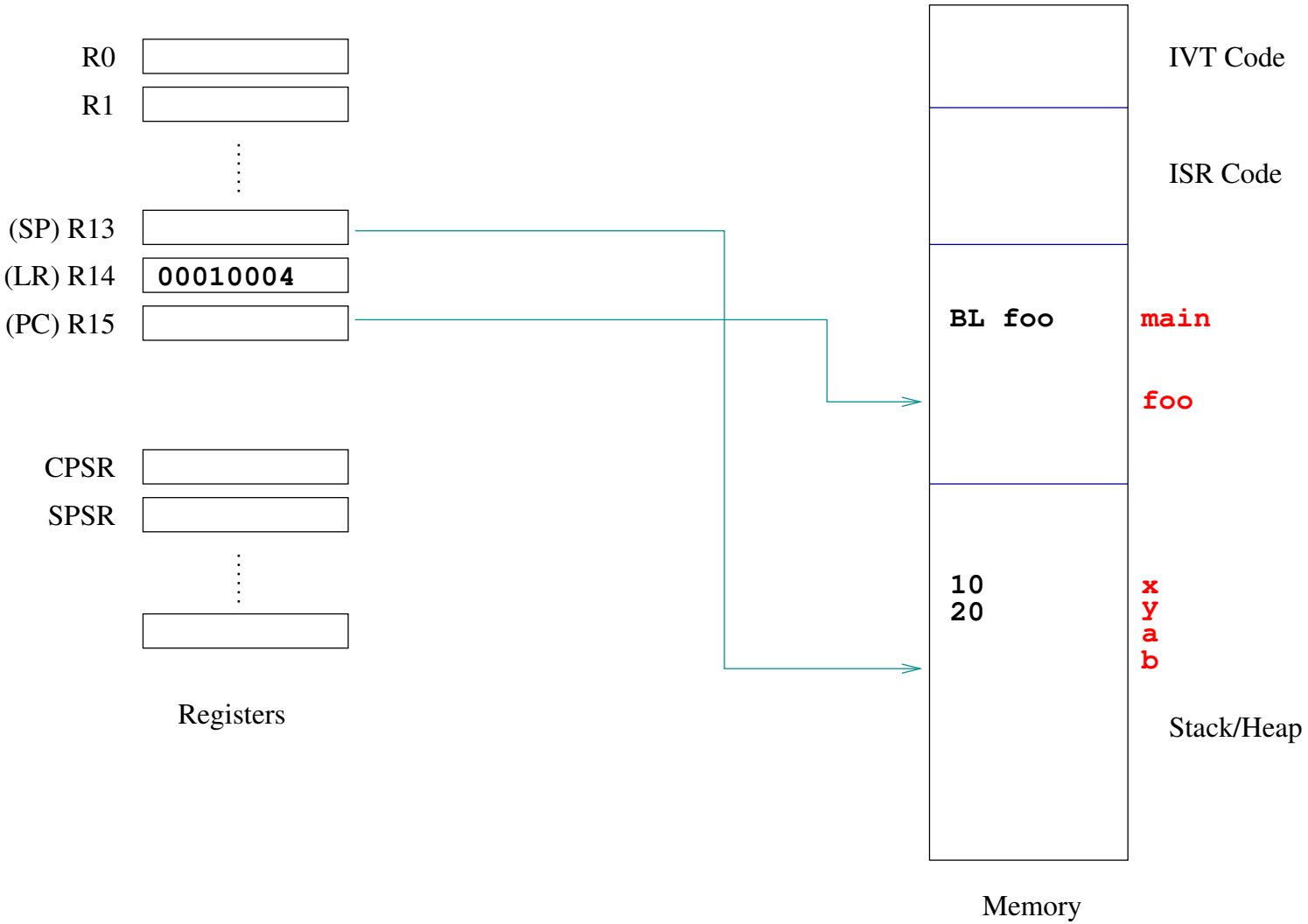
Normal execution: fetch instruction pointed to by PC and execute.

Using the stack: calling `foo`



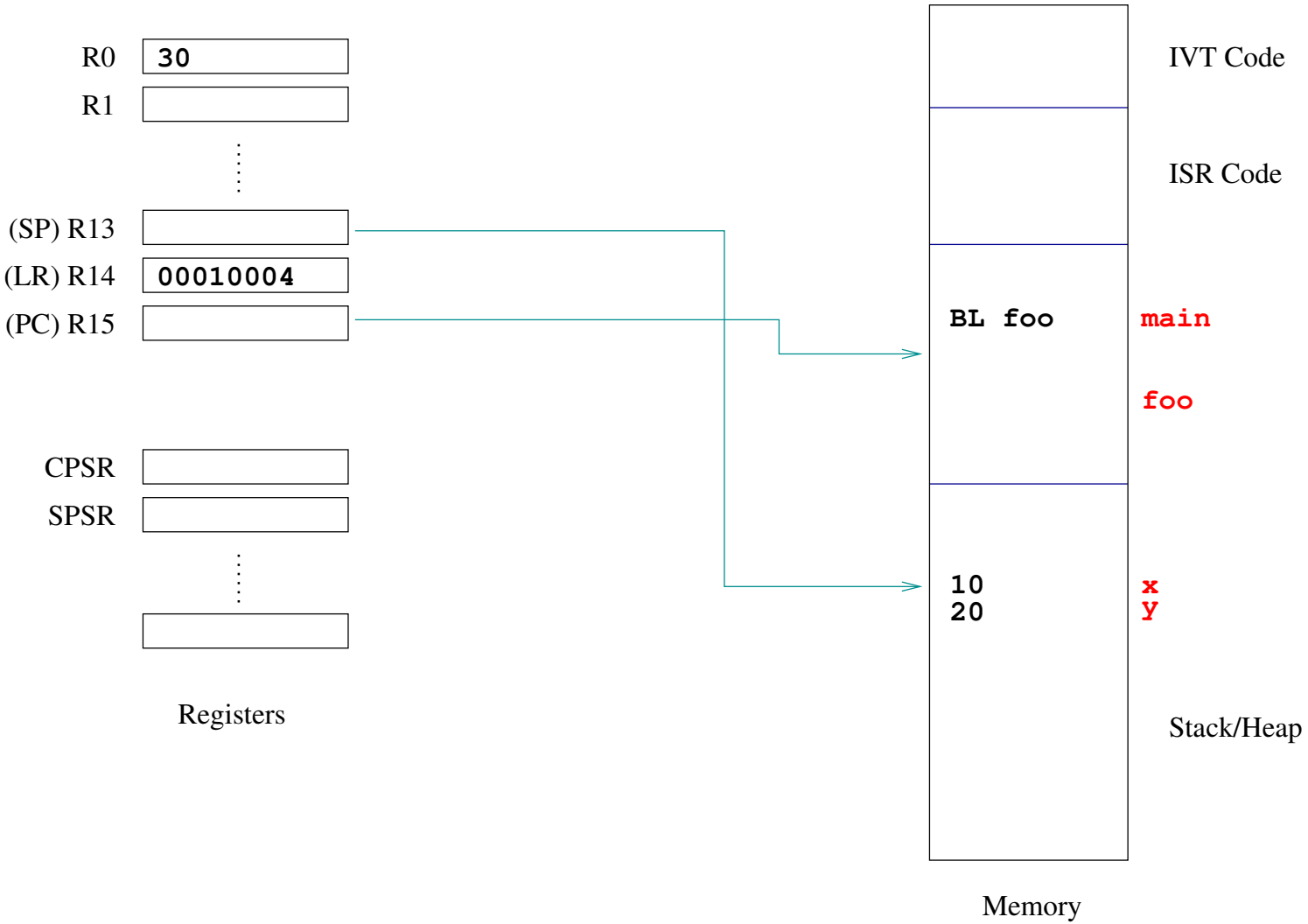
`main` pushes arguments to `foo` on the stack before calling `foo`.

Using the stack: in `foo`



`foo` creates space for local vars, saves regs it uses; accesses args.

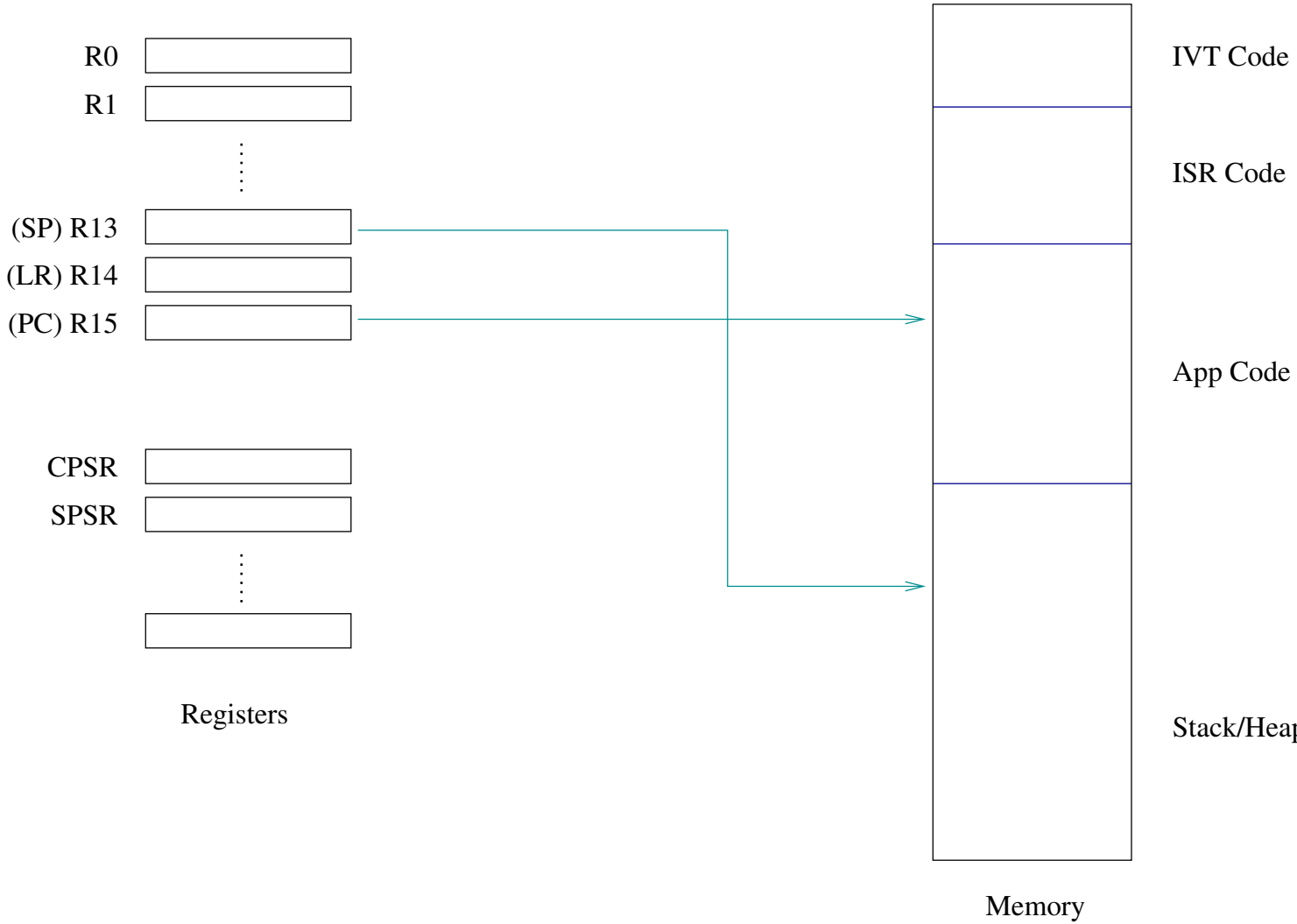
Using the stack: returning from `foo`



`foo` puts return value in register, restores regs and stack pointer.

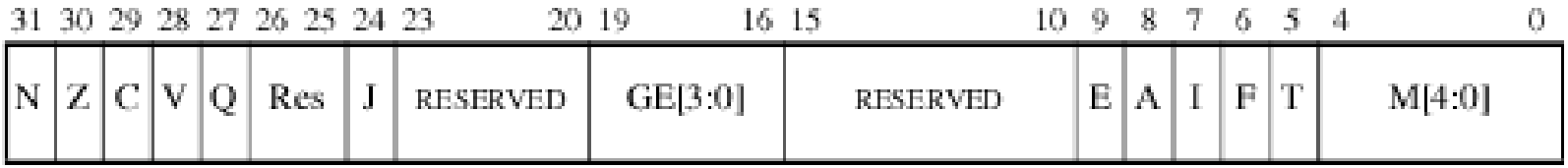
How interrupts are handled

- Save current PC in LR', CPSR in CPSR'.
- Change to "super" mode.
- Disable lower priority interrupts.
- Branch to appropriate IVT entry.



Current Program Status Register (CPSR)

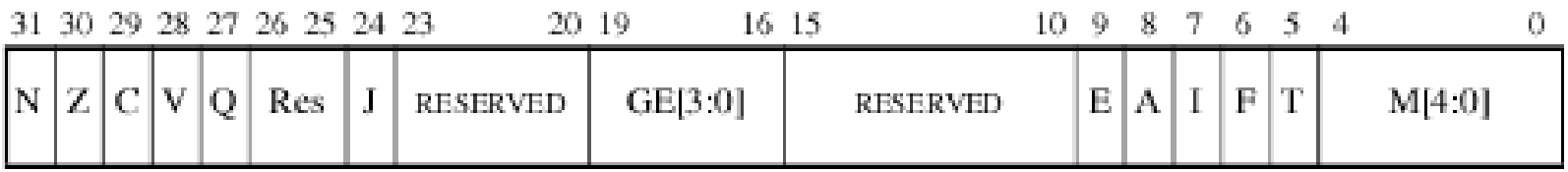
- Arithmetic instructions set bits [31:27] (overflow, carry, etc conditions).
- Normal programs execute in "User" mode.
  - Restricted access to memory and co-processor functions.
  - Cannot change "mode" (except by calling SWI).



Interrupts (Exceptions)

- Various kinds of interrupts may be generated (hardware eg. timer, software, instruction exceptions).
- Corresponding mode bits are set in CPSR[4:0] (Eg. 10011 for SWI).

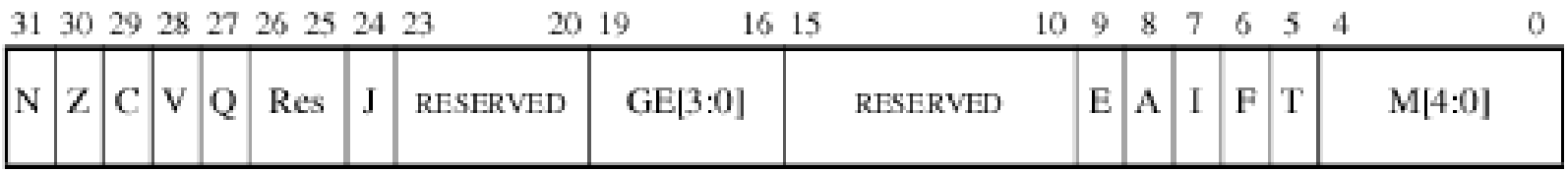
Exception	Resulting Mode	IVT address
Reset	Supervisor	0x00000000
Undefined Inst.	Undef	0x00000004
Software Interrupt	Supervisor	0x00000008
Abort prefetch	Abort	0x0000000C
Abort data	Abort	0x00000010
IRQ	IRQ	0x00000018
FIQ	FIQ	0x0000001C



Interrupts (Exceptions)

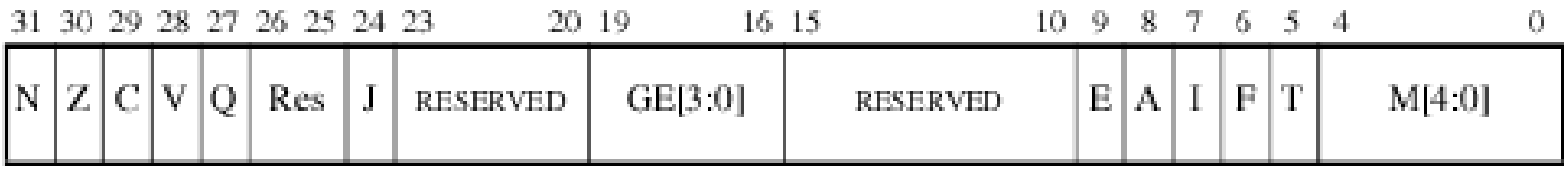
- Various kinds of interrupts may be generated (hardware eg. timer, software, instruction exceptions).
- Corresponding mode bits are set in CPSR[4:0] (Eg. 10011 for SWI).

Exception	Resulting Mode	IVT address	Priority
Reset	Supervisor	0x00000000	1
Undefined Inst.	Undef	0x00000004	6
Software Interrupt	Supervisor	0x00000008	6
Abort prefetch	Abort	0x0000000C	2
Abort data	Abort	0x00000010	2
IRQ	IRQ	0x00000018	4
FIQ	FIQ	0x0000001C	3



Disabling interrupts

- Setting 'I' bit (CPSR[7]) disables IRQ interrupts.
- Setting 'F' bit (CPSR[6]) disables FIQ interrupts.



What the ARM processor does on an interrupt

When interrupt (exception) occurs do [ARM]

```
R14_<exception_mode> = return link
SPSR_<exception_mode> = CPSR
CPSR[4:0] = exception mode number
CPSR[5] = 0                                  /* Execute in ARM state *
if <exception_mode> == Reset or FIQ then
    CPSR[6] = 1                              /* Disable fast interrupt
CPSR[7] = 1                                  /* Disable normal interru
if <exception_mode> != UNDEF or SWI then
    CPSR[8] = 1                              /* Disable imprecise abor
CPSR[9] = CP15_reg1_EEbit                    /* Endianness on exceptio
PC = exception vector address
```

ARM has many instructions that facilitate a quick save/restore for entering/exiting an ISR.

What the ARM processor does on an s/w interrupt

When SWI instruction is executed do [ARM]

```
R14_svc     = address of next instruction after the SWI instructi
SPSR_svc    = CPSR
CPSR[4:0]   = 0b10011                   /* Enter Supervisor mode */
CPSR[5]     = 0                         /* Execute in ARM state */
CPSR[7]     = 1                         /* Disable normal interrupts
CPSR[9]     = CP15_reg1_EEbit           /* Endianness on exception en
if high vectors configured then
    PC      = 0xFFFF0008
else
    PC      = 0x00000008
```

To return after handling: R14 = $R14_{svc}$, CPSR = $SPSR_{svc}$, return to the instruction following SWI.

To return after handling [ARM]

```
    MOVS PC,R14
```

What RTOS provides the programmer

Ways to:

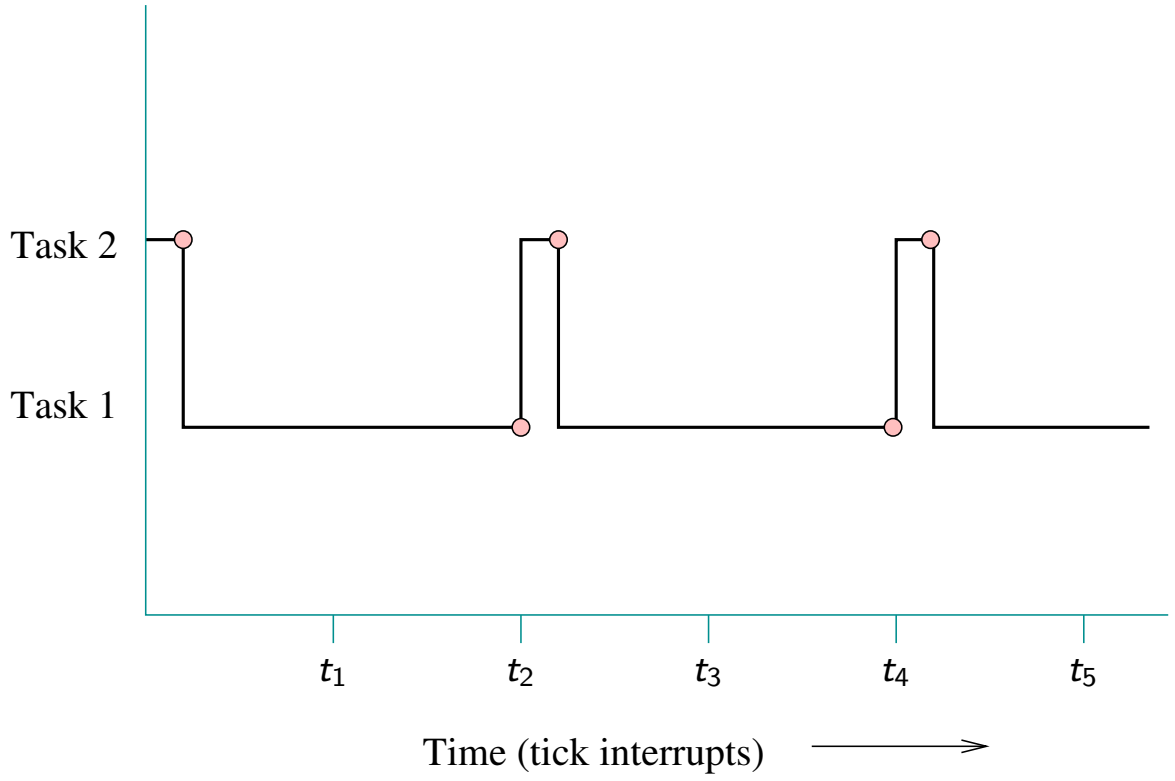
- Create and manage multiple tasks.
- Schedule tasks based on priority-based pre-emption.
- Let tasks communicate (via message queues, semaphores, etc).
- Let tasks delay and timeout on blocking operations.

Example application that uses RTOS

Sample RTOS application

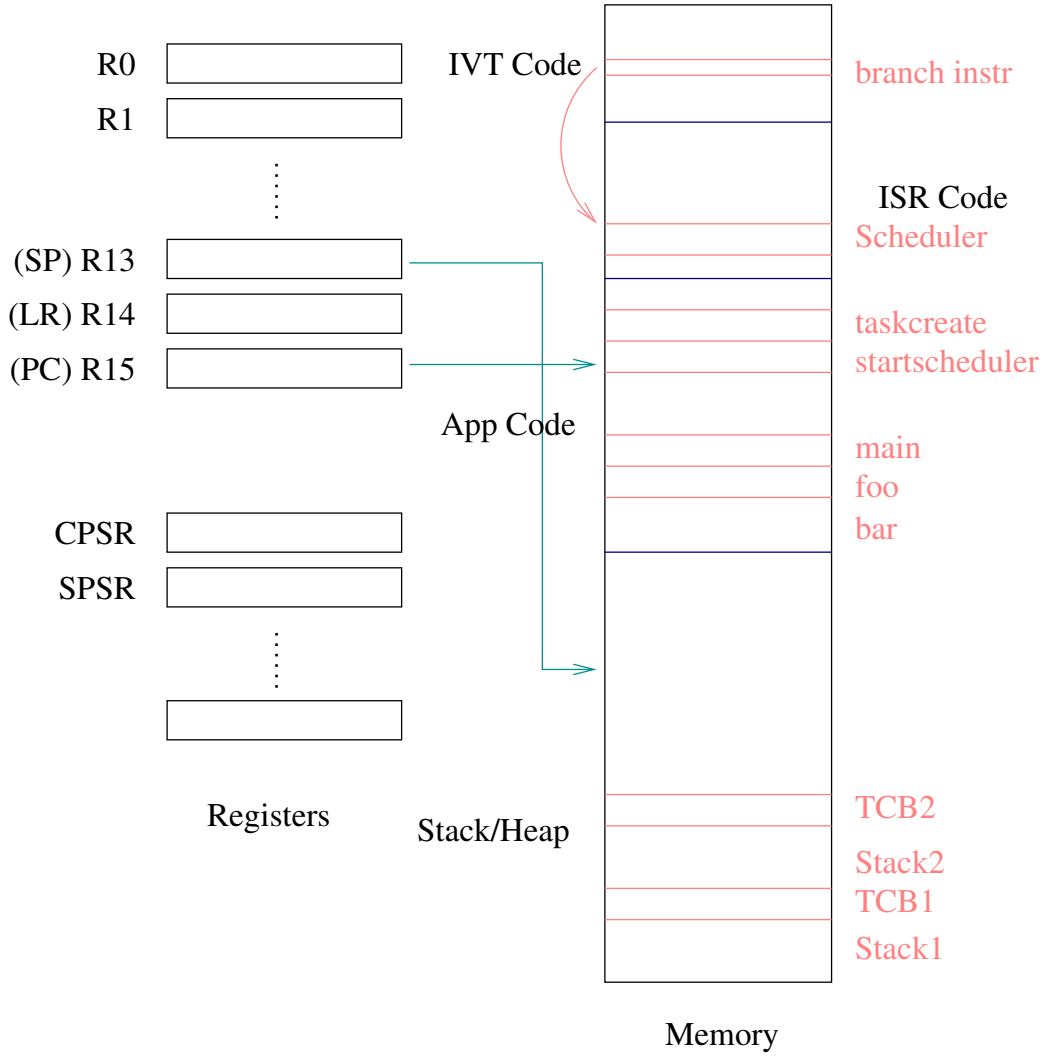
```
int main(void){
    xTaskCreate(foo, "Task 1", 1000, NULL, 1, NULL);
    xTaskCreate(bar, "Task 2", 1000, NULL, 2, NULL);
    vTaskStartScheduler();
}

void foo(void* params){
    for(;;);
}

void bar(void* params){
    for(;;){
        vTaskDelay(2);
    }
}
```

Task execution in example application



Example application: execution sequence

```
main
        vtaskCreate(1)
        vtaskCreate(2)
        vtaskStartScheduler()
                create Idle task
        task2
                vtaskdelay()
                    yield()
        task1
                timer interrupt
                timer interrupt
        task2
```

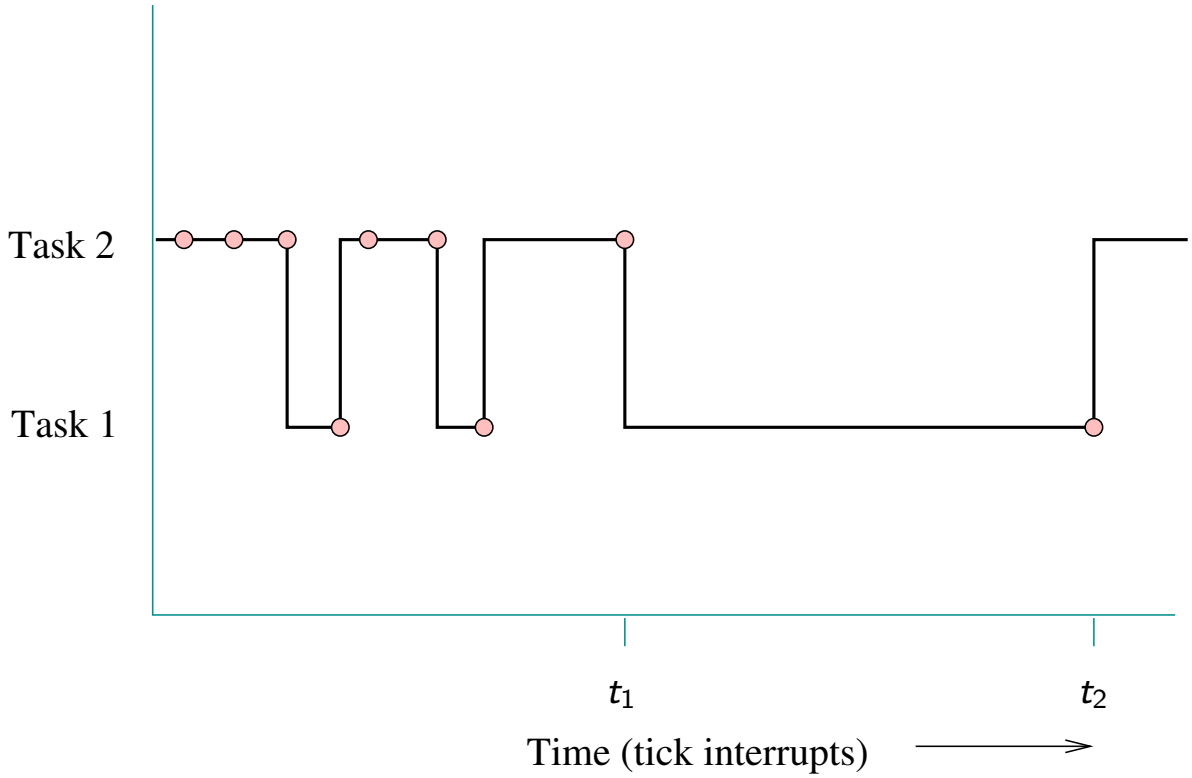


Example 2: using RTOS inter-task communication

RTOS application (Example 2)

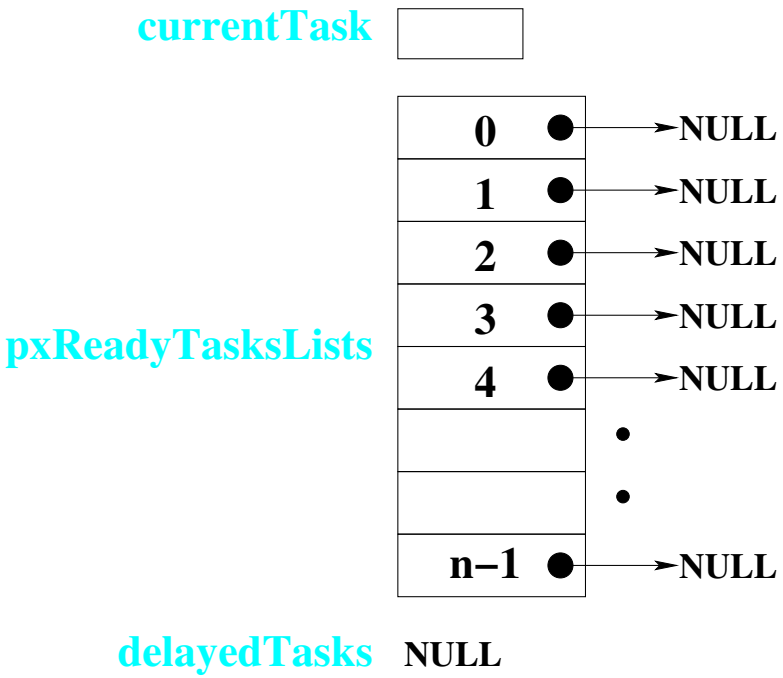
```
int main(void){
        xTaskCreate(vTask1,"Task1",configMINIMAL_STACK_SIZE,NULL,2,&xTask1Handle);
        xTaskCreate(vTask2,"Task2",configMINIMAL_STACK_SIZE,NULL,4,&xTask2Handle);
        vTaskStartScheduler();
}

void vTask1(void *pvParameters){
        xQueueReceive(xQueue,&lGlobalData,0);
        vTaskPrioritySet(NULL,0);
        for(;;);
}

void vTask2(void *pvParameters){
        long lData = 1;
        xQueue = xQueueCreate(2,sizeof(long));
        xQueueSendToBack(xQueue,&lData,0);
        lData = 2;
        xQueueSendToBack(xQueue,&lData,0);
        lData = 3;
        xQueueSendToBack(xQueue,&lData,1000);
        vTaskPrioritySet(NULL,0);
        for(;;);
}
```

Task execution in example application



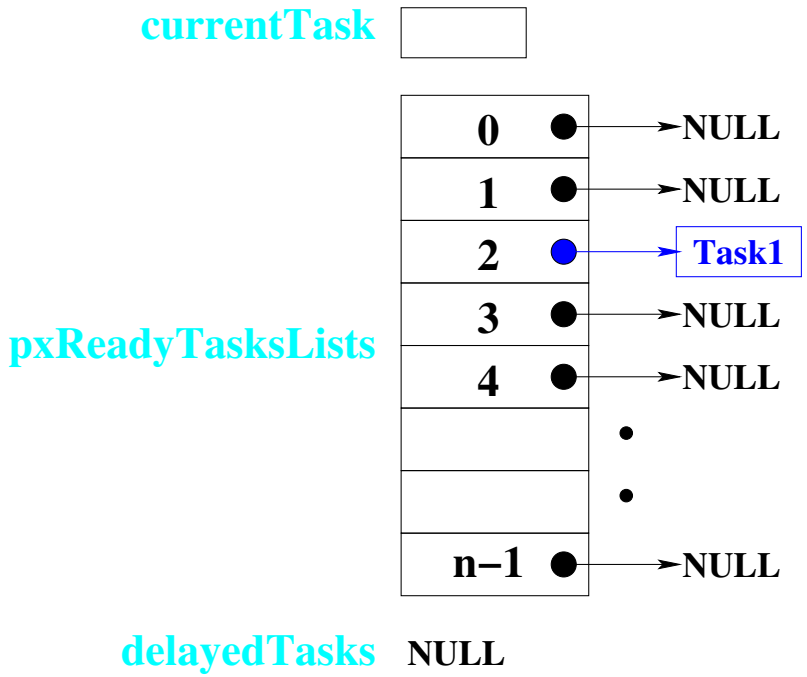
Example 2: Data-structures maintained by RTOS

Initially:



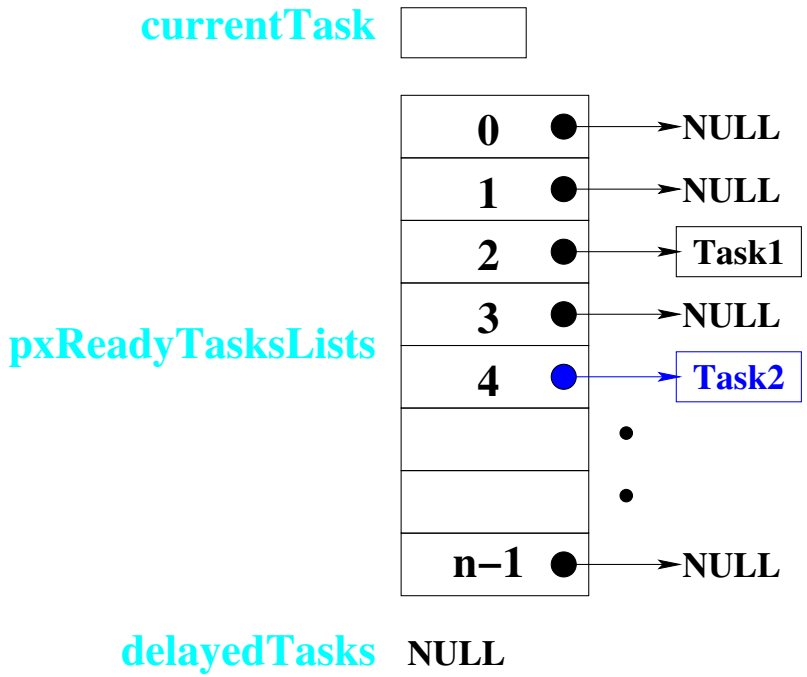
Example 2: Data-structures maintained by RTOS

After TaskCreate(1):



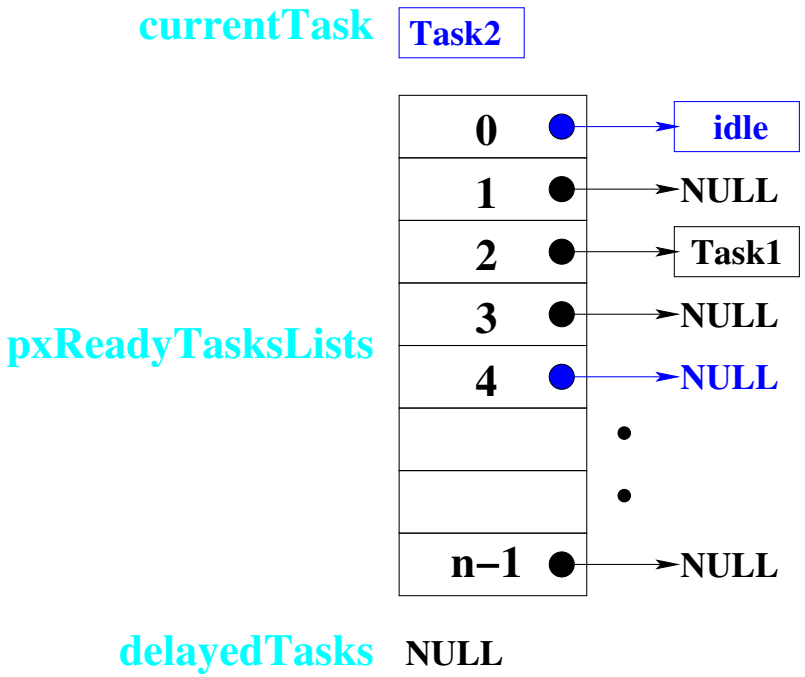
Example 2: Data-structures maintained by RTOS

After TaskCreate(2):



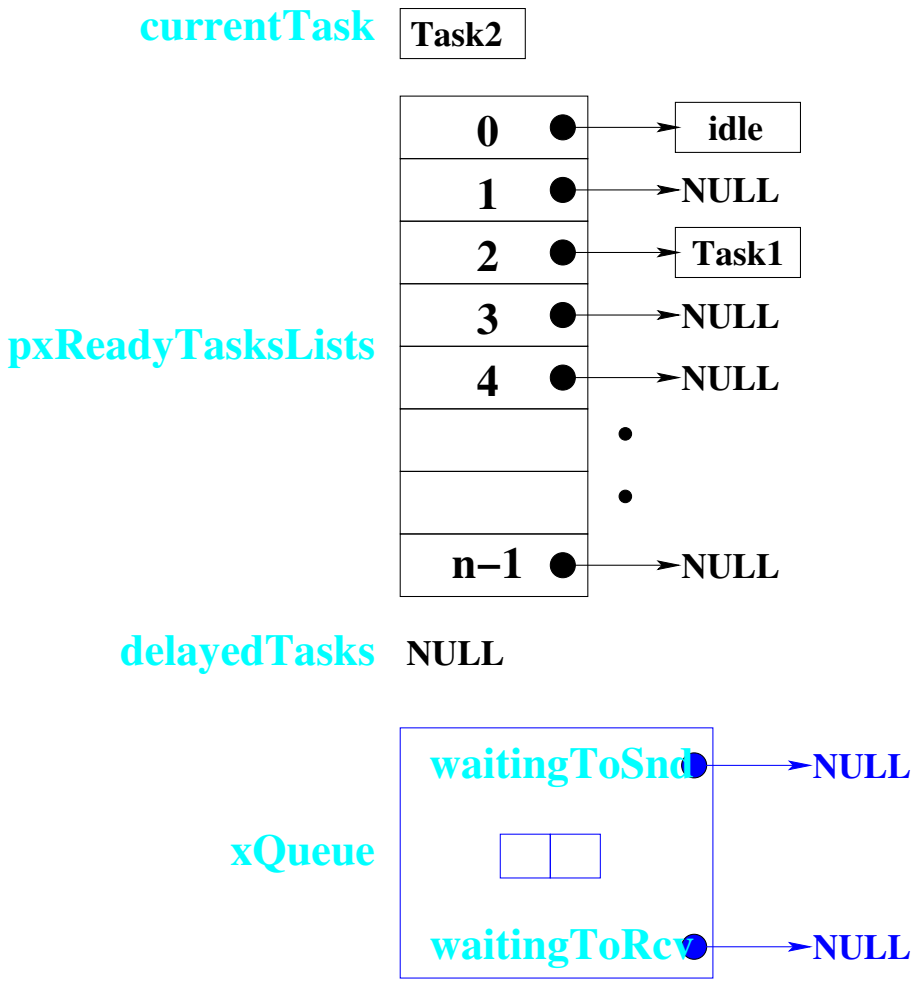
Example 2: Data-structures maintained by RTOS

After TaskStartScheduler():



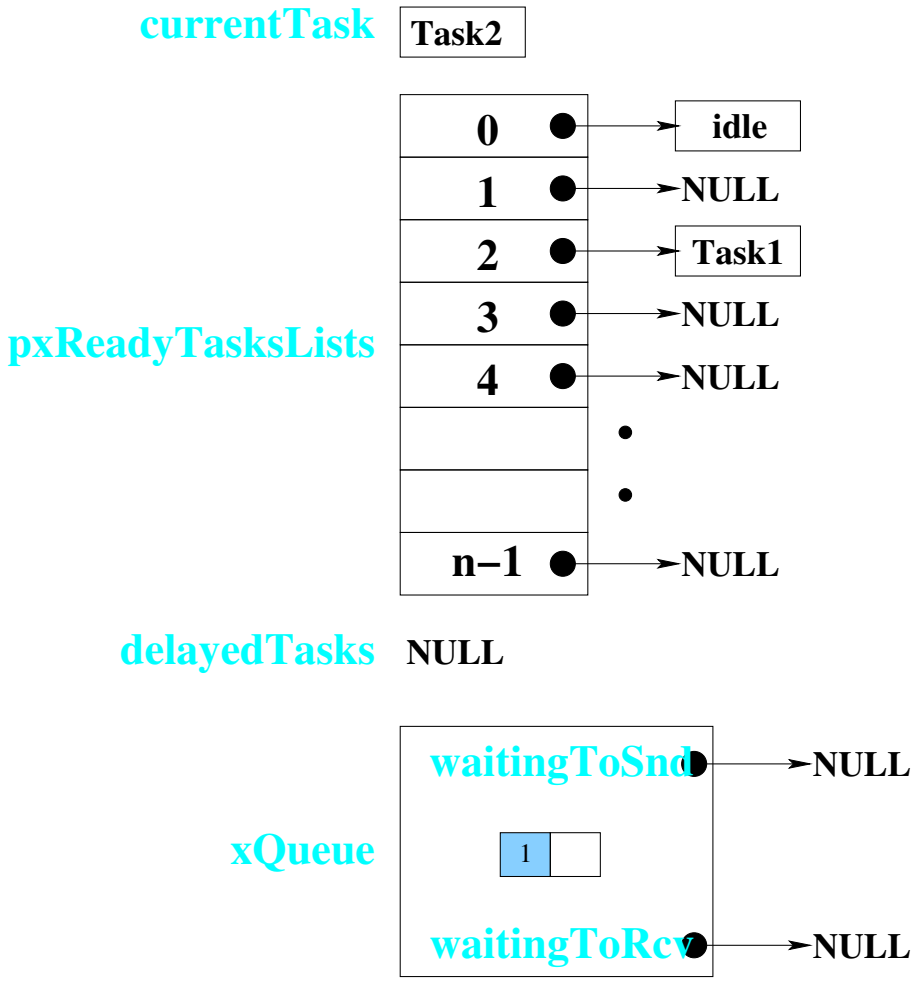
Example 2: Data-structures maintained by RTOS

After QueueCreate:



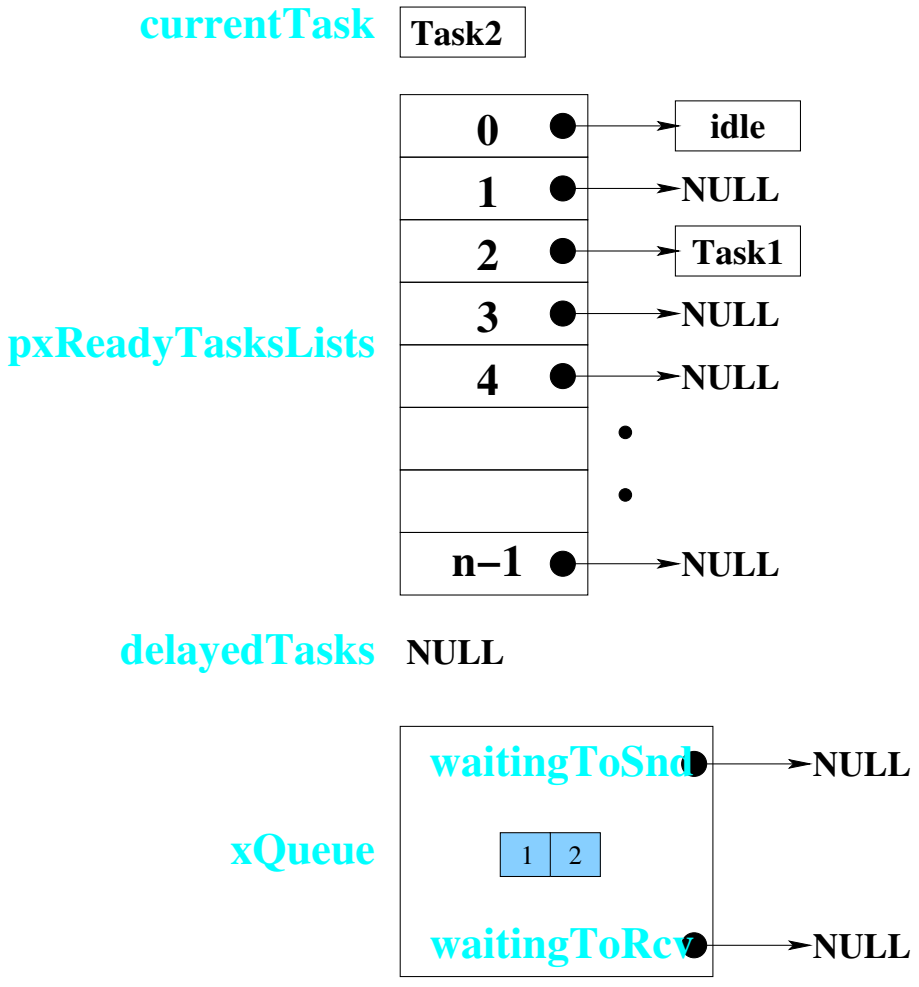
Example 2: Data-structures maintained by RTOS

After QueueSend(1):



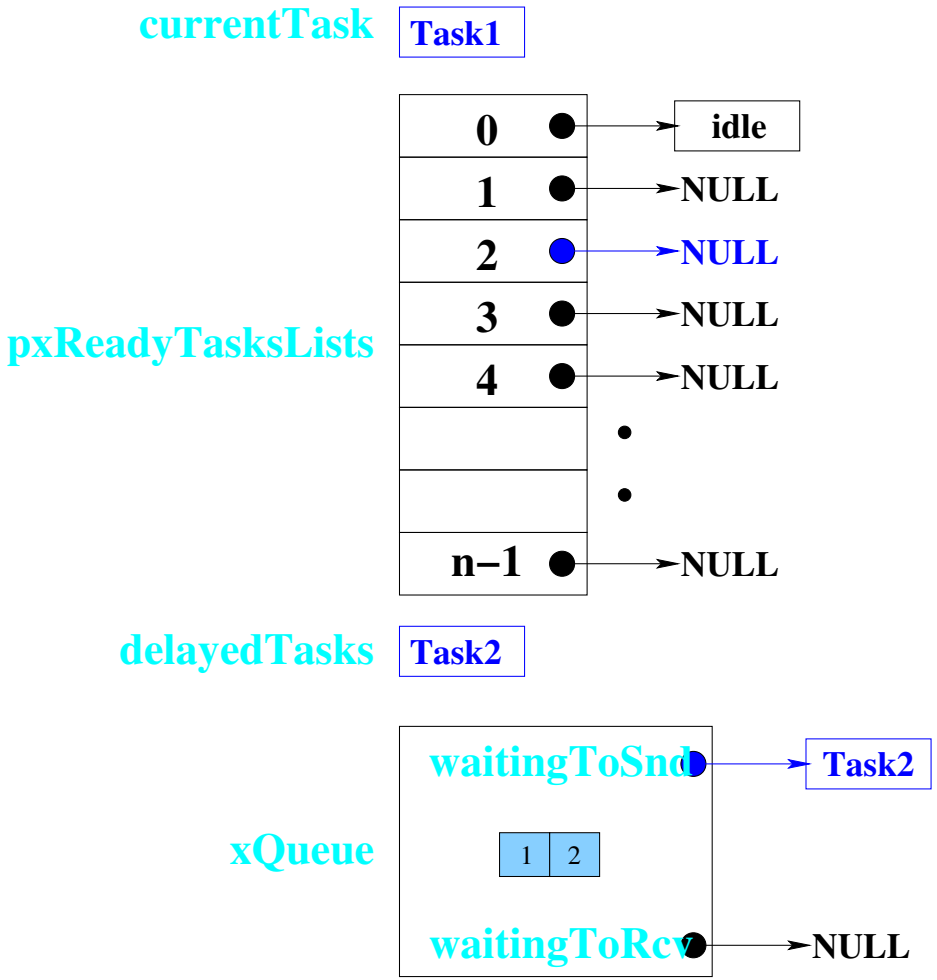
Example 2: Data-structures maintained by RTOS

After QueueSend(2):



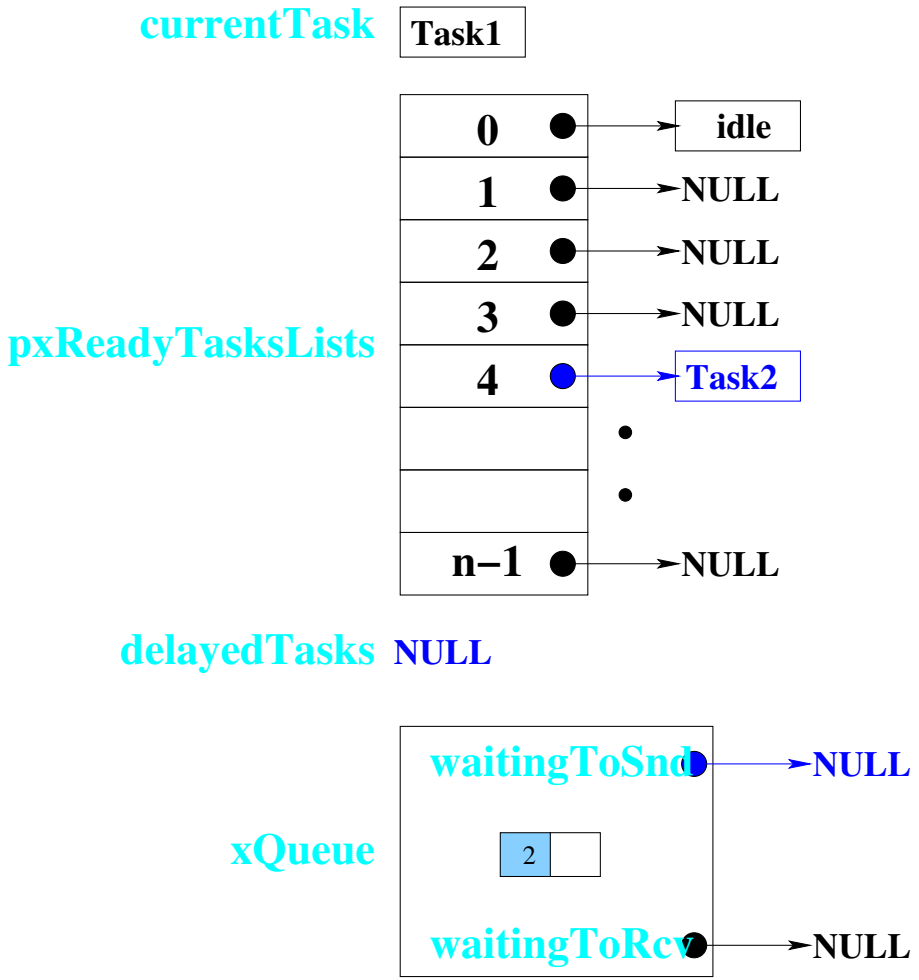
Example 2: Data-structures maintained by RTOS

After QueueSend(3)$_b$:



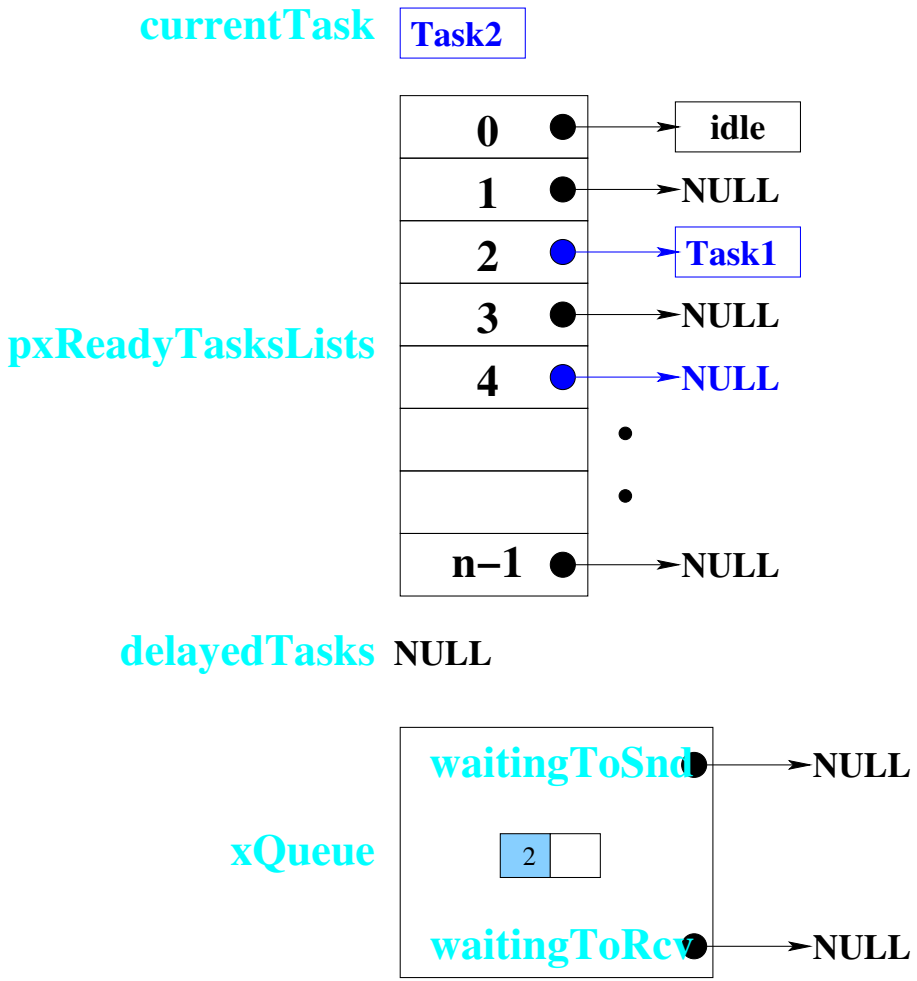
Example 2: Data-structures maintained by RTOS

After QueueReceive(1):



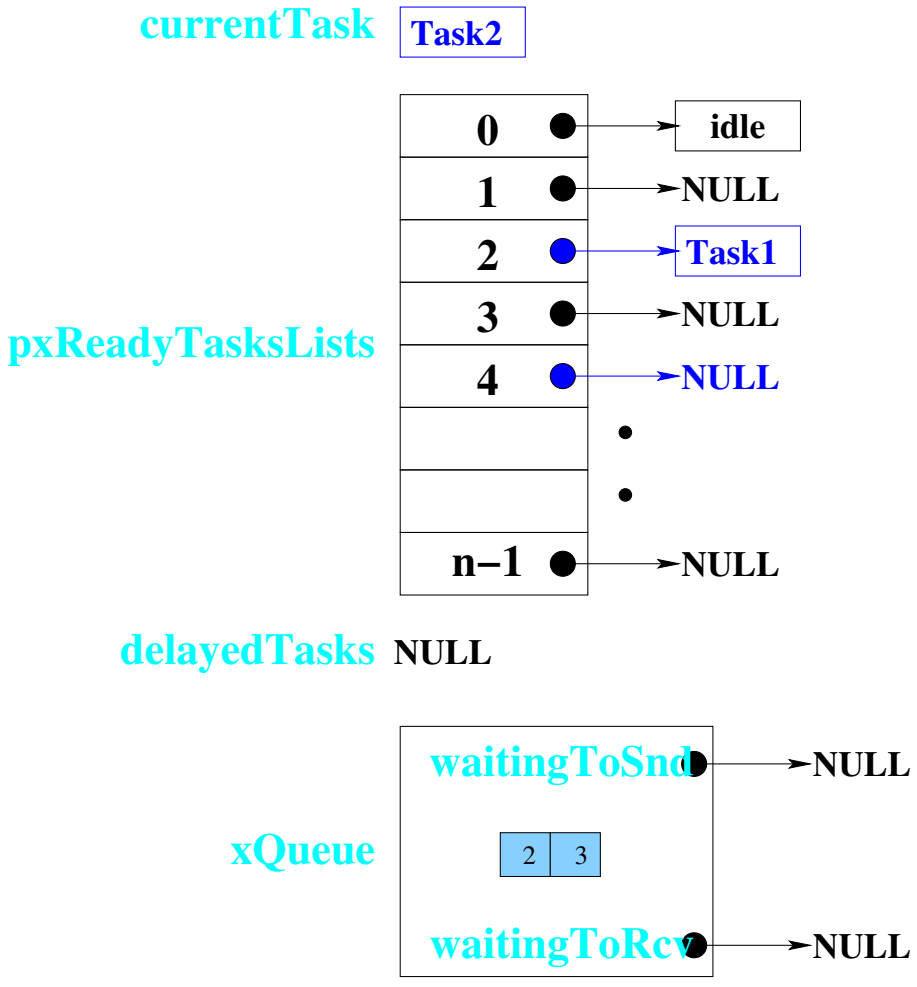
Example 2: Data-structures maintained by RTOS

After QueueReceive(1):



Example 2: Data-structures maintained by RTOS

After QueueSend(3)$_e$:



Main functionality of RTOS

What do we expect RTOS to do?

Main functionality of RTOS

What do we expect RTOS to do?

- Implement its stated scheduling policy (fixed priority pre-emptive scheduling).
- Trap SWI interrupts
  - Find highest priority ready task to run.
  - Save context of yielding task.
  - Restore context of new task.
- Trap timer IRQ interrupt
  - Update tickcount,
  - Check delayed tasks, and move to ready if required,
  - Switch context if required.
- Provide Application Programmer Interface (API's) for:
  - Task creation, deletion, set priority, etc.
  - Inter-task communication through queues, semaphores, and mutexes.
  - Heap memory management (malloc, free).

References

 ARM Architecture Reference Manual

 RTOS User Guide, `http://www.freertos.org`.