Real Time Operating systems (RTOS) concepts

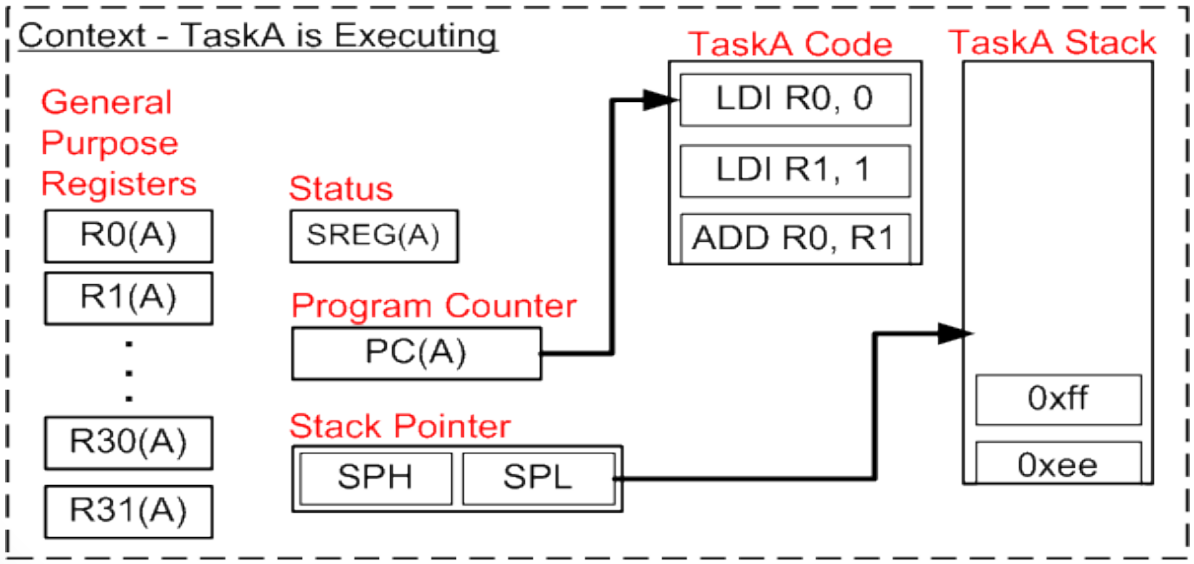
Abu Bakr Mohamed Ramadan
Eng.abubakr88@gmail.com

Content:

- Task Context.
- Context Switching between tasks.
- Shared Data Problem.
- *Non Reentrant Function.*
- Reentrant Function.
- Gray area of reentrancy.
- How to protect Shared Data?
- References and Read more

Task Context

- Every task has it's own context (it's own Data).
- Every Task Created has it's own data structure called Task Control Block (TCB).
- Task saves it's data like: tasks status, ID , priority, stack pointer, Pointer to function(task itself) ……, in it's **TCB**.
- Task context also saved in it's own stack and CPU registers.

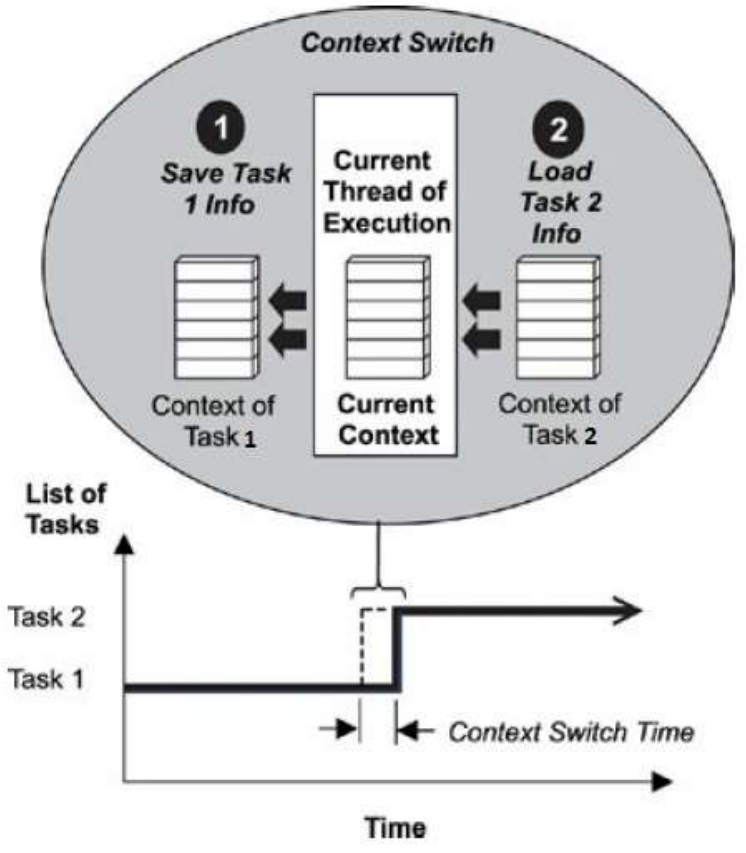


Context Switching between tasks

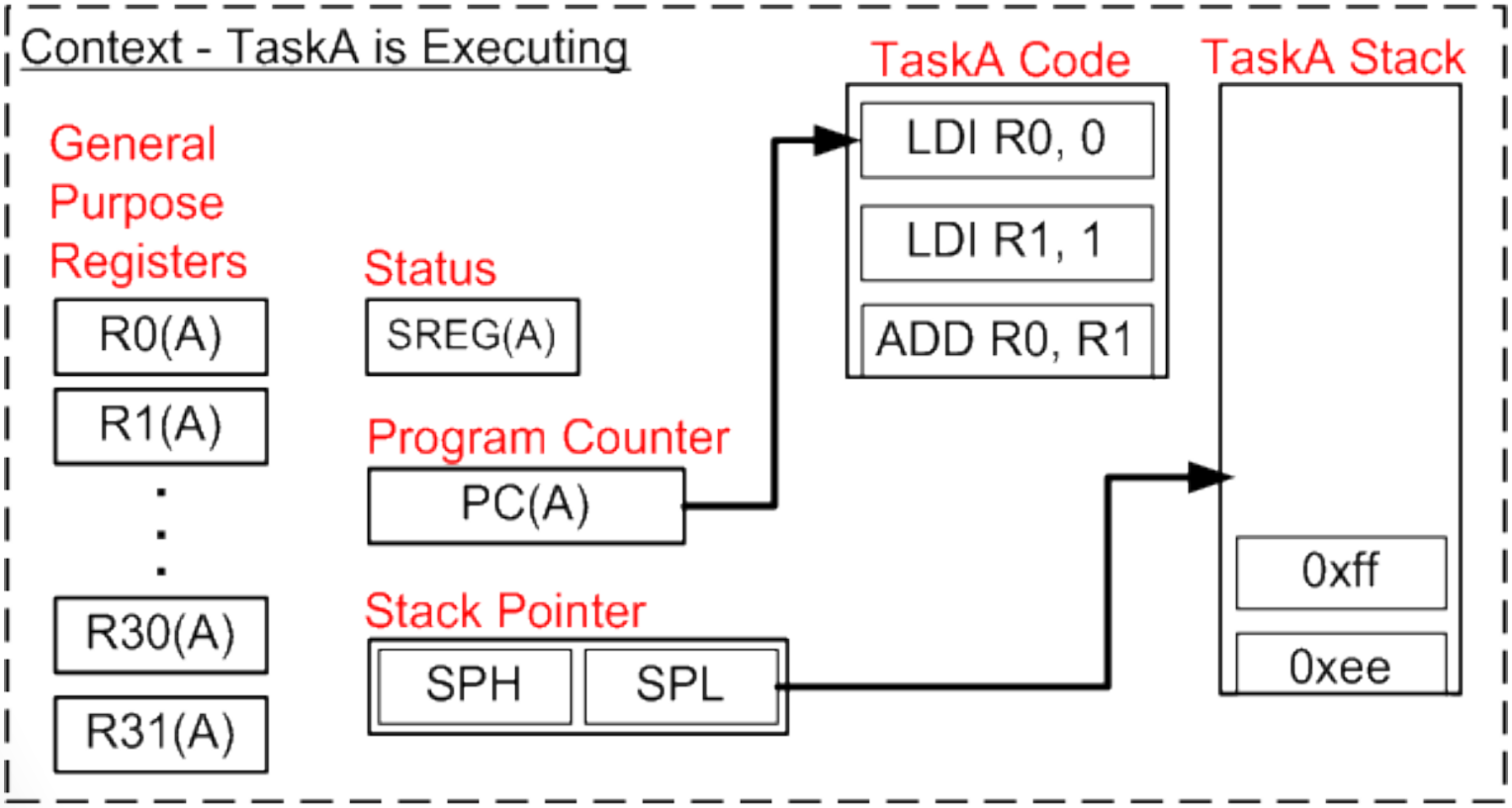
- **Context switching :** Is how to switch the processor between the context of one task to the another, so the system must:
  - Save the state of the old process,
  - Then load the saved state for the new process,
  - The new process continues from where it left off just before the context switch.
- When the task is not running, its context is frozen within the TCB, to be restored the next time the task runs.
- **The Dispatcher :**
  - Is the part of the scheduler that performs context switching.

Context Switching between tasks Cont.

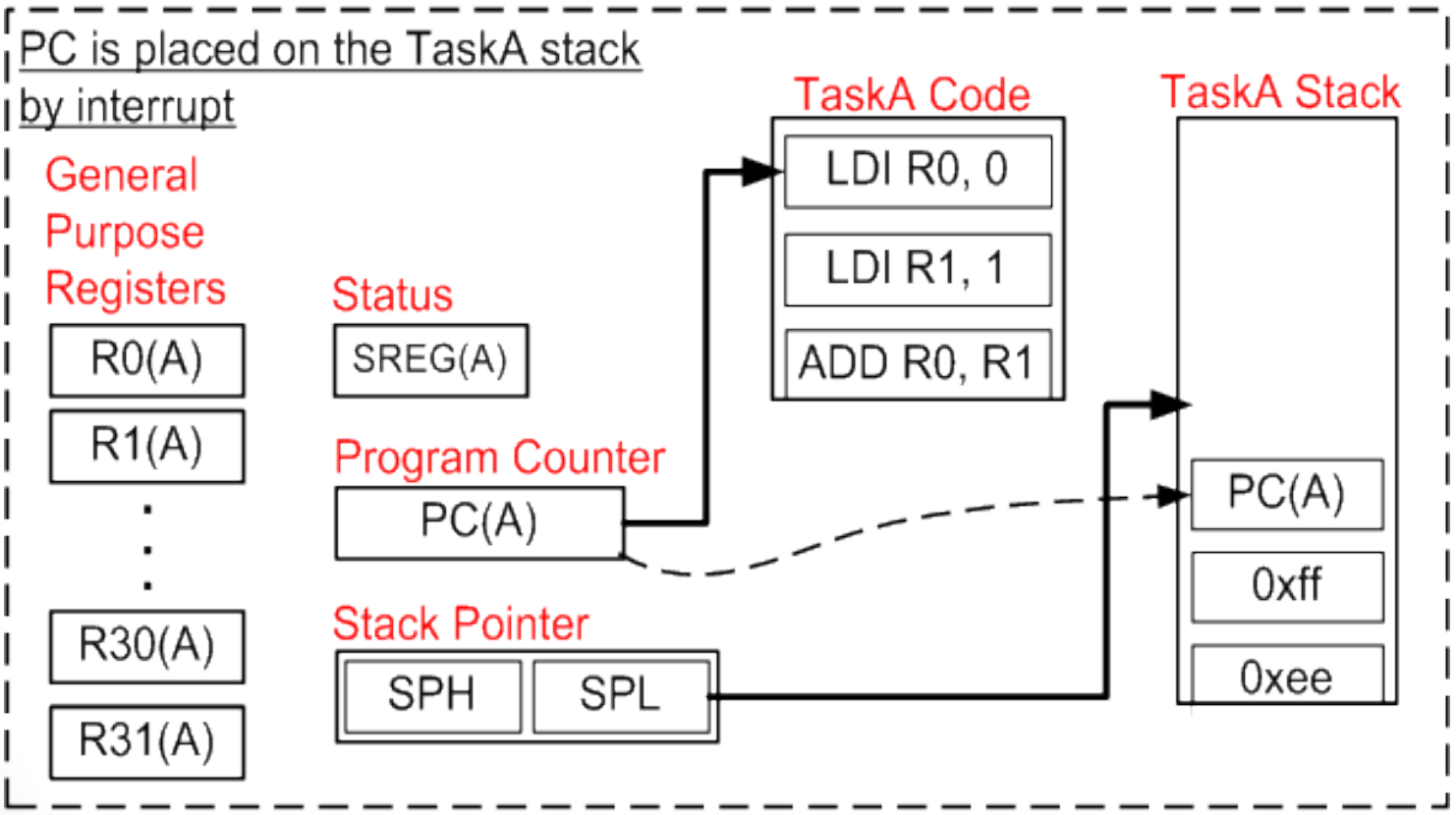
- **Context Switch Time:** Is the time it takes for the scheduler to switch from one task to another.
- frequent context switching makes a performance overhead.



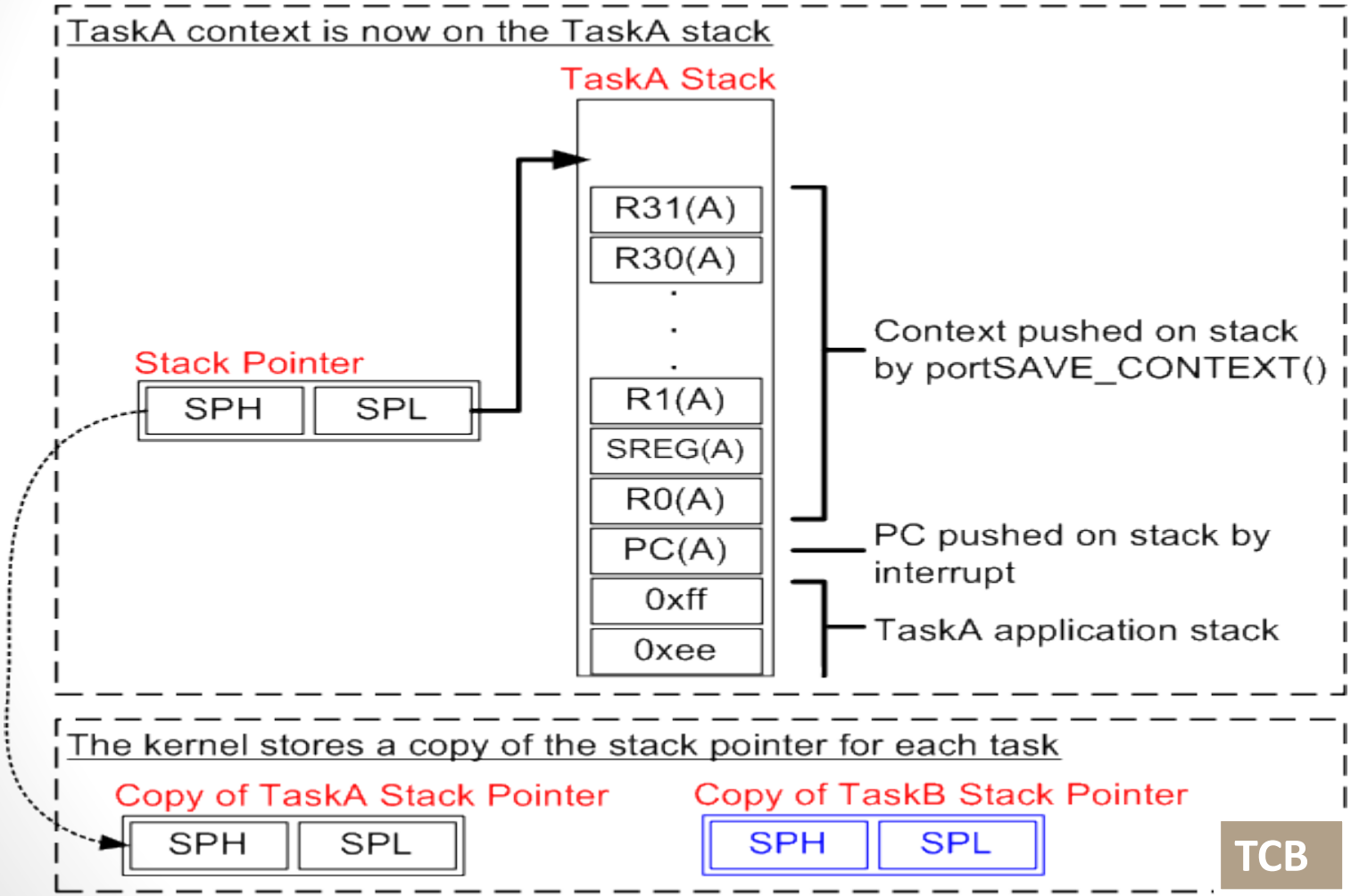
Context Switching between tasks Cont.



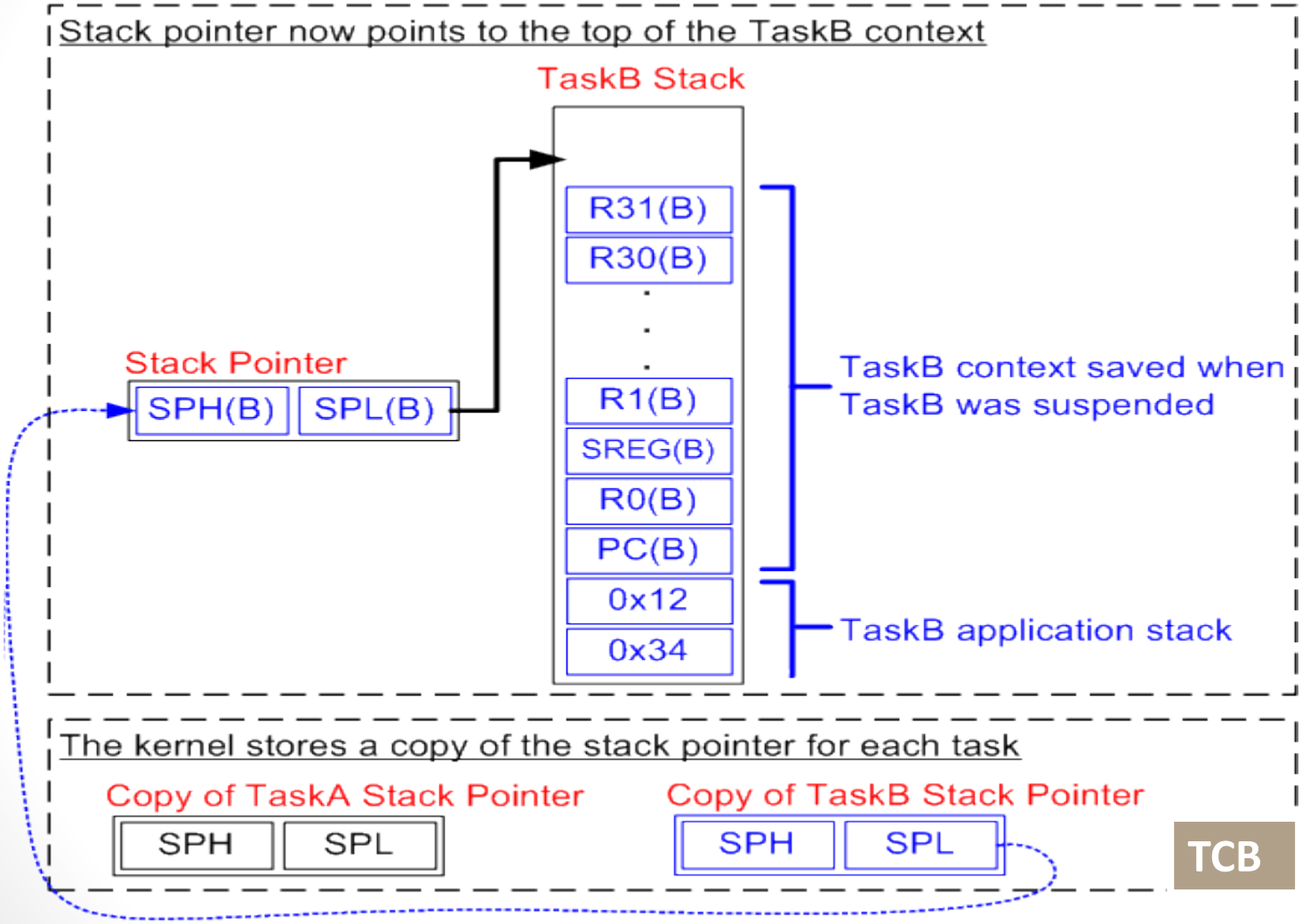
Context Switching between tasks Cont.



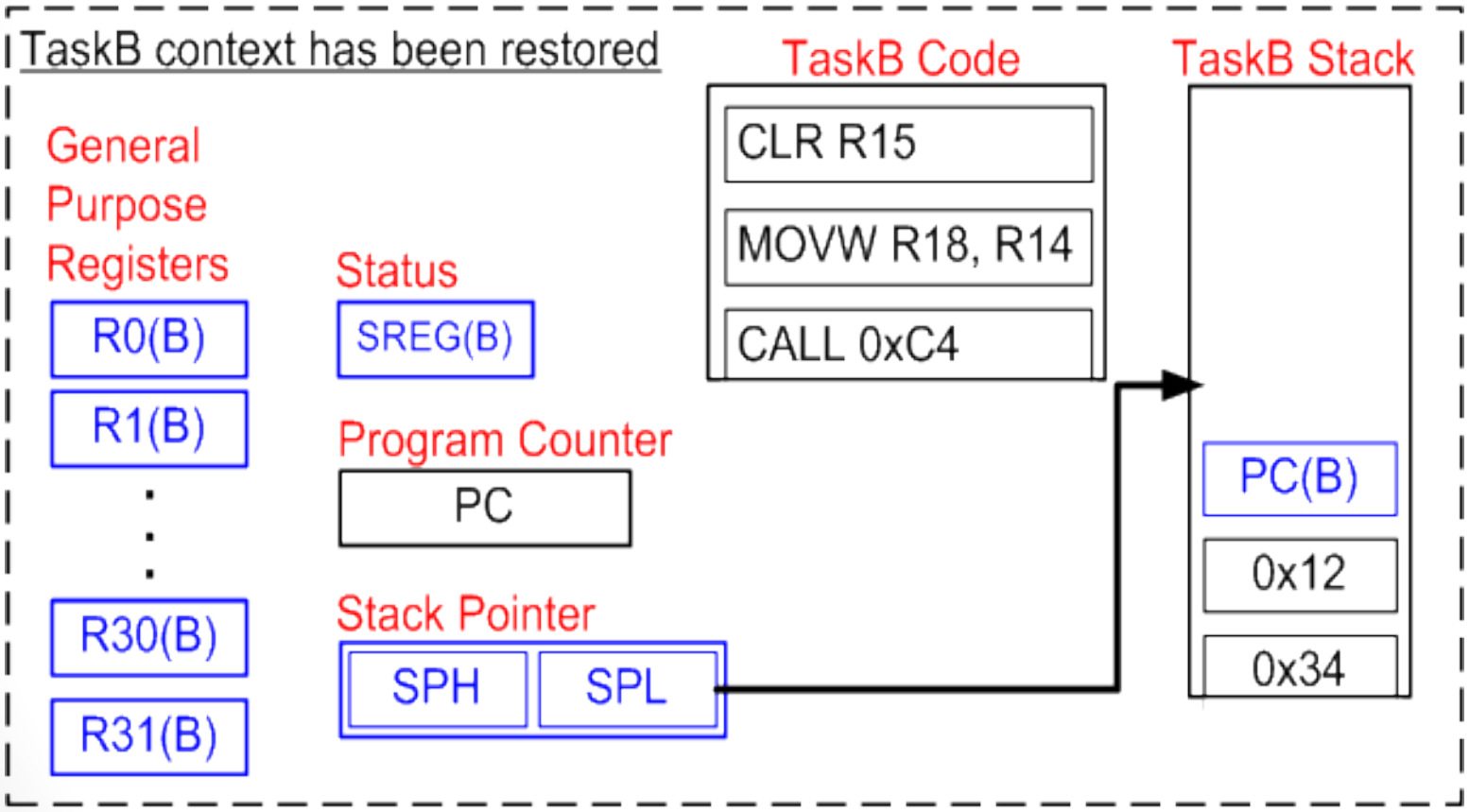
Context Switching between tasks Cont.



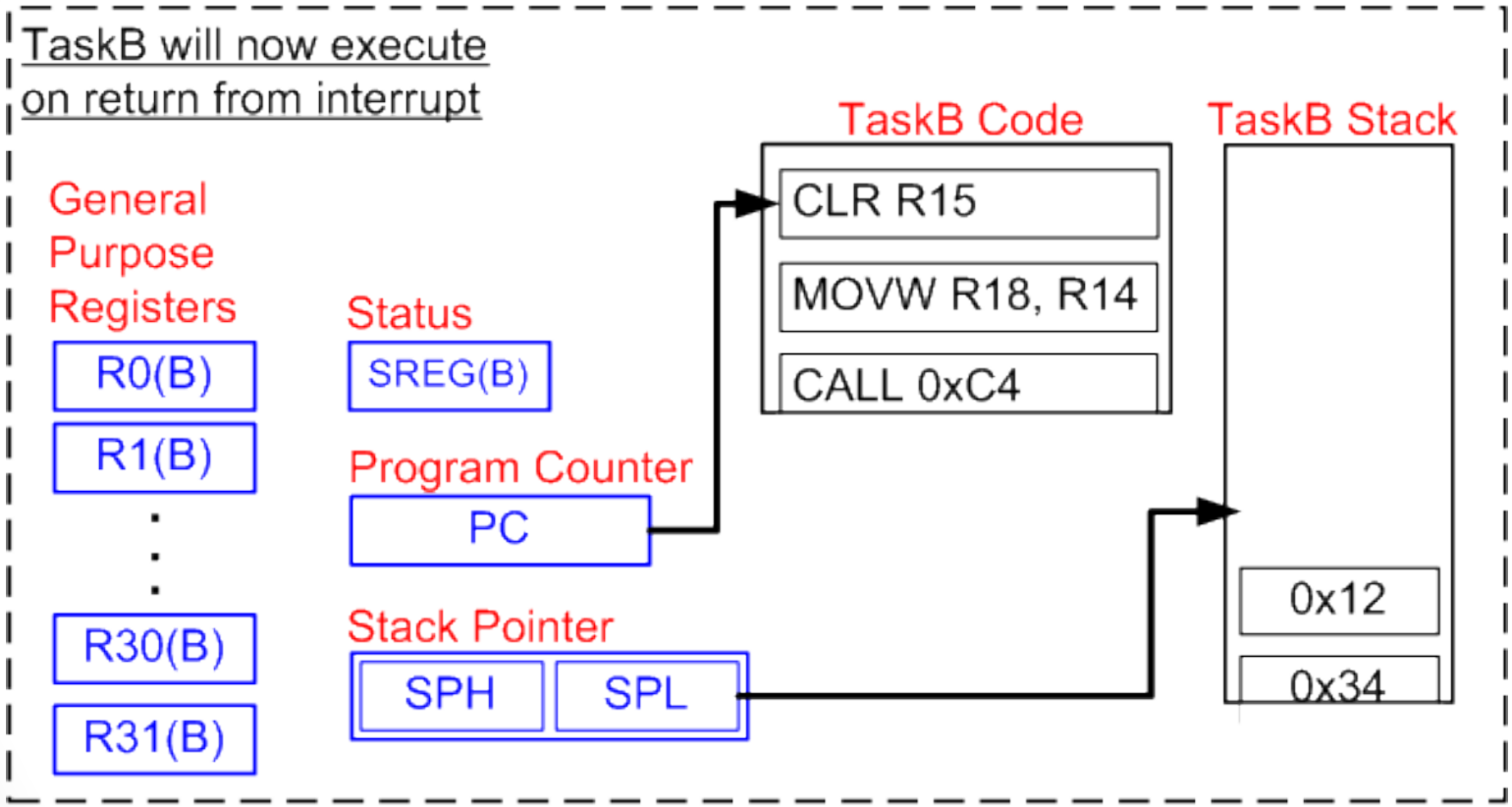
Context Switching between tasks Cont.



Context Switching between tasks Cont.



Context Switching between tasks Cont.



Shared Data Problem.

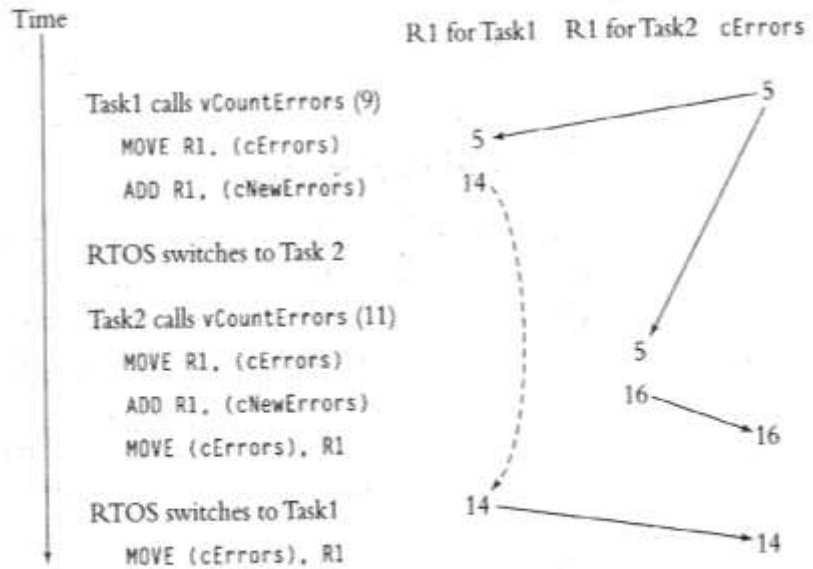
```
void Task1 (void)
{
    .
    .
    .
    vCountErrors (9);
    .
    .
    .
}

void Task2 (void)
{
    .
    .
    .
    vCountErrors (11);
    .
    .
    .
}

static int cErrors;

void vCountErrors (int cNewErrors)
{
    cErrors += cNewErrors;
}
```

```
; Assembly code for vCountErrors
;
; void vCountErrors (int cNewErrors)
;{
;       cErrors += cNewErrors;
        MOVE R1, (cErrors)
        ADD R1, (cNewErrors)
        Move (cErrors), R1
        RETURN
;}
```



Non Reentrant Function.

- Is a function that can't be used between more than one task.
- Can't be interrupted or a data loss will happen.
- Example:

```
static int cErrors;

void vCountErrors (int cNewErrors)
{
    cErrors += cNewErrors;
}
```

Reentrant Function.

- Is a function that can be used between more than one task with out fear of data corruption.
- Can be interrupted at any time and resumed without loss of data.
- Reentrant functions either:
  - Use local variables, Or use protected global variables.
  - Can’t Call other non reentrant function.

Gray area of reentrancy

- **Some Functions and some operation on a shared data are** processor and compiler dependent.

- **Example:**
  - **printf() Function Is reentrant or non reentrant?**
    - The answer is it’s depend on the processor and on the compiler.
  - **A++**

How to protect Shared Data?

- **Using Mutual Exclusion access.**
- **Examples on Mutual Exclusion methods are:**
  - Disable and enable interrupts,
  - Disabling Scheduling, and
  - Using Semaphores.

References and Read more:

- **Real-Time Concepts for Embedded Systems book** by Qing Li and Carolyn.
  - http://www.e-reading.club/book.php?book=102147
- **An Embedded Software Primer** by David E. Simon.
  - http://www.amazon.com/Embedded-Software-Primer-David-Simon/dp/020161569X
- **Linux Kernel Embedded Systems Building Blocks 2e** by Jean J. Labrosse.
  - http://www.amazon.com/Embedded-Systems-Building-Blocks-Ready/dp/0879306041
- **FreeRTOS website.**
  - http://www.freertos.org