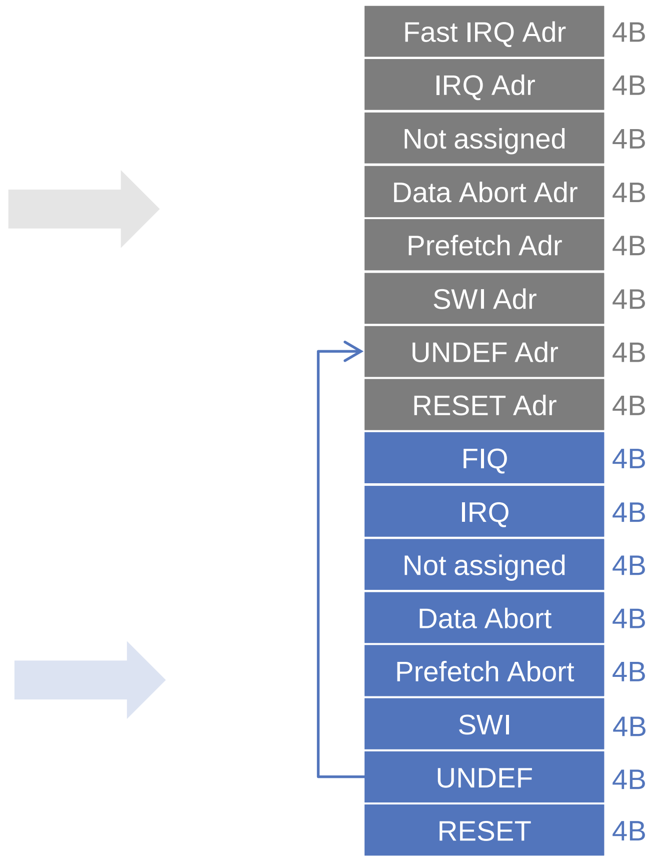
Initialization of Exceptions

```
InstallHandler(SWITrap, Platform.SWI);
InstallHandler( .... );
....
```

```
FOR i := 0 TO 7 DO
   SYSTEM.PUT32(ExceptionVectorBase + 4*i,
      0E59FF018H);
END;
```



Enable IRQs

```
VAR cpsr: LONGINT;
..
SYSTEM.STPSR(0, cpsr);
cpsr := SYSTEM.VAL(LONGINT, SYSTEM.VAL(SET, cpsr) - {7});
SYSTEM.LDPSR(0, cpsr)
```

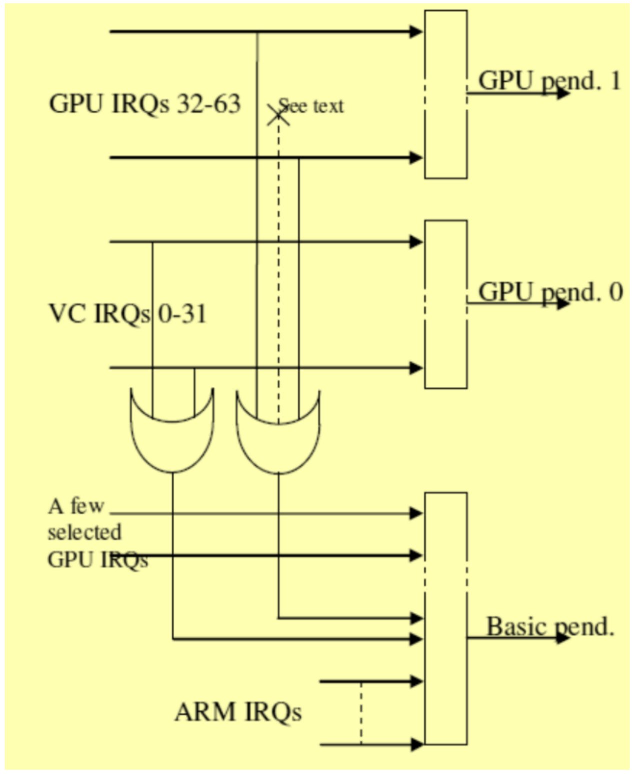
Install Timer

```
Platform.WriteWord(Platform.STC1,
      Platform.ReadWord(Platform.STCLO)+Platform.TimerInterval);

Platform.WriteBits (Platform.STCS, {1});

nextTimerInterrupt := Platform.ReadWord(Platform.STC1);

EnableIRQ(Platform.SystemTimerIRQ, TRUE);
```

Sets bit in IRQEnable registers
cf. BCM2835 ARM Peripherals document, chapter 7.

IRQ Trap Handler

```
PROCEDURE {INTERRUPT, PCOFFSET=4} IRQTrap;
VAR i, j, spsr: LONGINT;  basicPending, pending1, pending2: SET;
BEGIN
  SYSTEM.STPSR( 1, spsr );   (* store SPSR *)
  (* read pending bits *)
  ...
  (* disable corresponding device interrupts *)
  ...
  (* cf BCM2835 Manual, Section 7.5 *)
  (* process pending bits and call irq handler*)
  ...
  SYSTEM.LDPSR( 1, spsr ); (* SPSR := old *)
END IRQTrap;
```

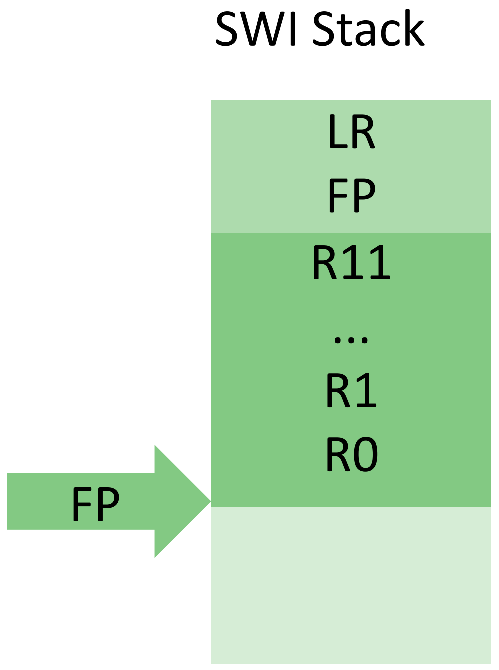


DataAbort handler

```
(*page fault*)
PROCEDURE {INTERRUPT, PCOFFSET=8} DataAbort;
VAR lnk, fp: LONGINT;
BEGIN
  (* The location that trapped was lnk - 8 *)
  lnk := SYSTEM.LNK - 8;
  fp := SYSTEM.FP;
  IF trapHandler # NIL THEN
    trapHandler(Platform.DataAbort, lnk, fp)
  ELSE
   (* diagnostics output and halt *)
  END
END DataAbort;
```

SWITrap handler

```
PROCEDURE {INTERRUPT, PCOFFSET=0} SWITrap;
(* software interrupt (e.g. failed ASSERT) *)
  VAR lnk, fp: LONGINT;
BEGIN
  (* The location that trapped was lnk - 4 *)
  lnk := SYSTEM.LNK - 4;
  fp := SYSTEM.FP;
  IF trapHandler # NIL THEN
    trapHandler(Platform.SWI, lnk, fp) (* stack trace *)
  END
END SWITrap;
```



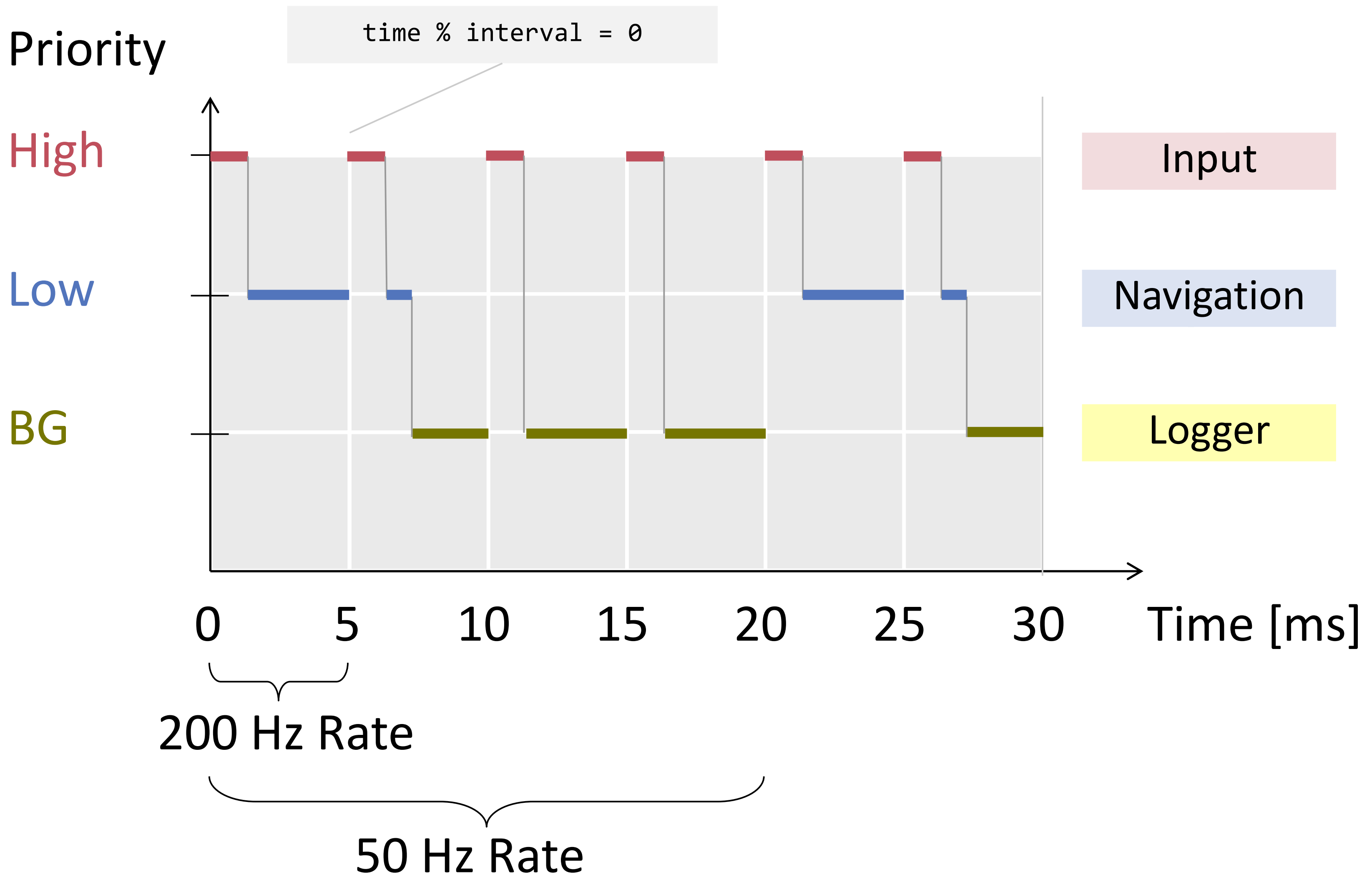
All registers are saved during entering the trap.
Get the original FP (reg12) from the local stack and traverse the stack.

1.4. TASK SCHEDULING

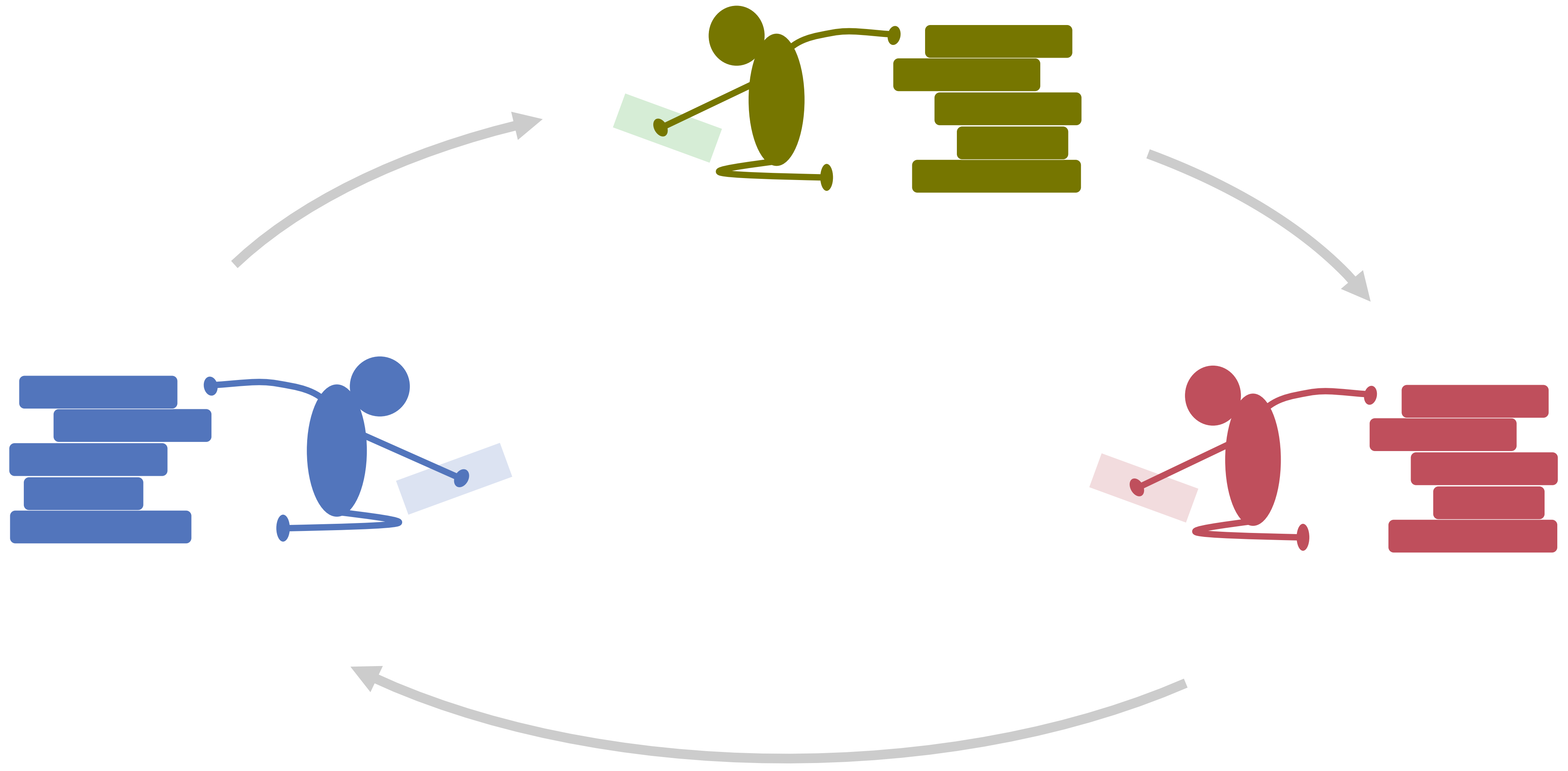
Scheduling Strategy

- Task types
  - High priority synchronous tasks
    (scheduled each 5 ms)
  - Low priority synchronous tasks
    (scheduled each 20 ms)
  - Background tasks
- Rules of preemption
  - High priority tasks preempt all others
  - Low priority tasks preempt background tasks
  - Background tasks don‘t preempt

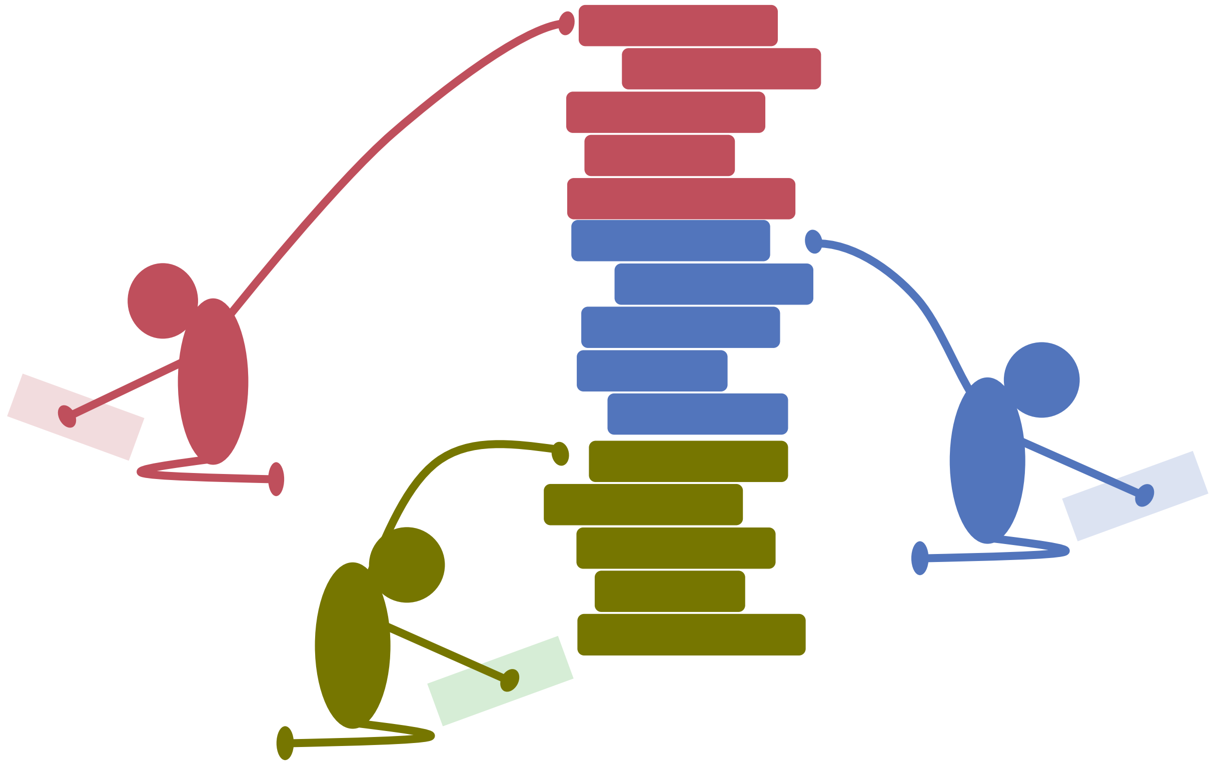
Scheduling Example



A Stack for each Process?

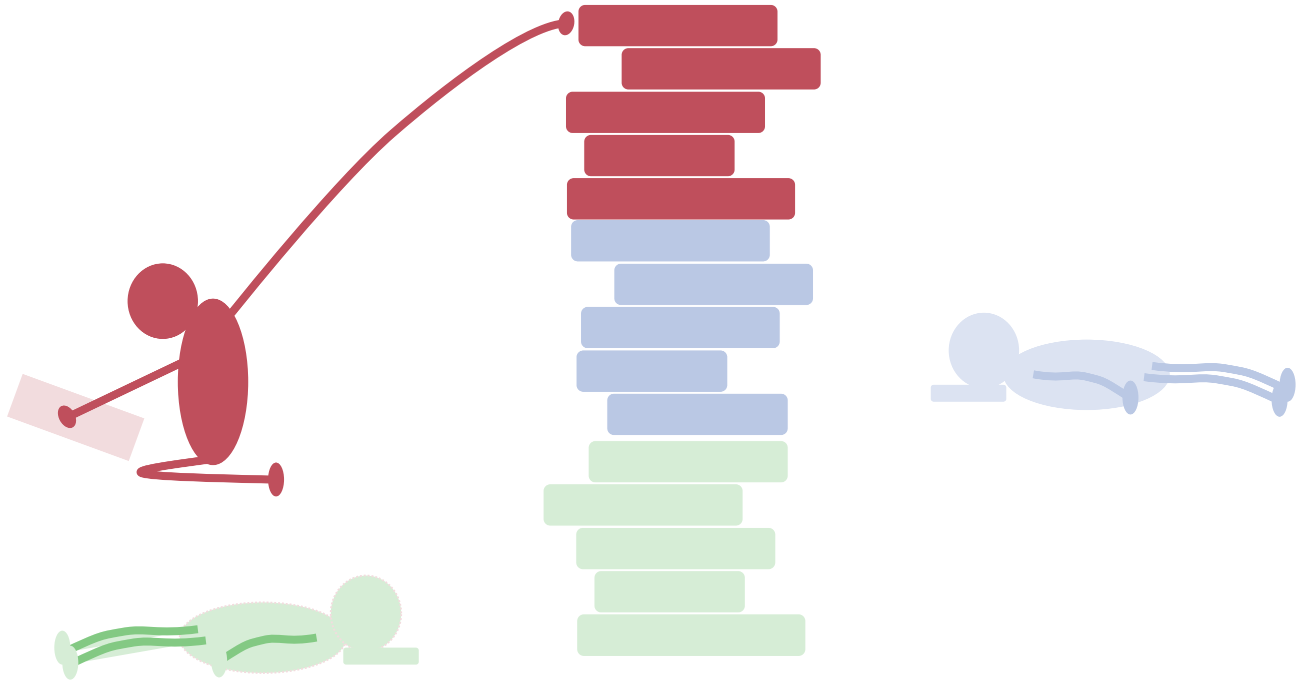


One Stack for All?

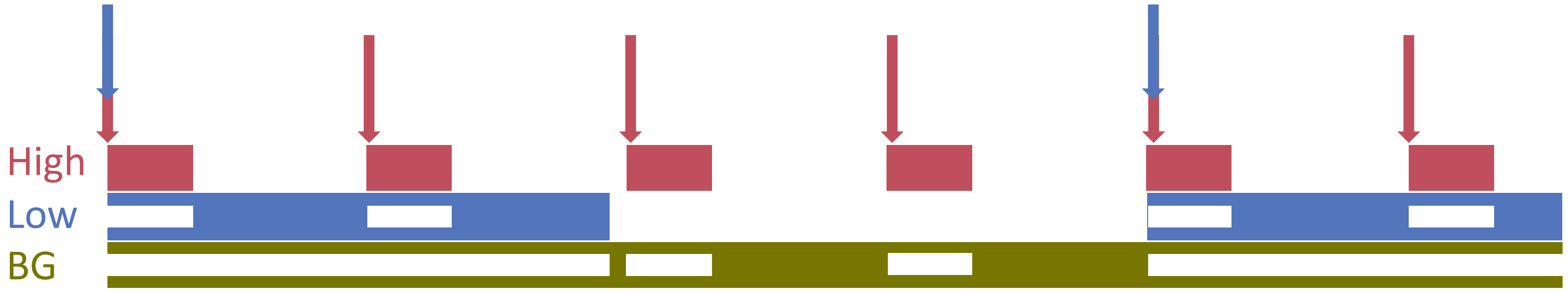


When, How ?

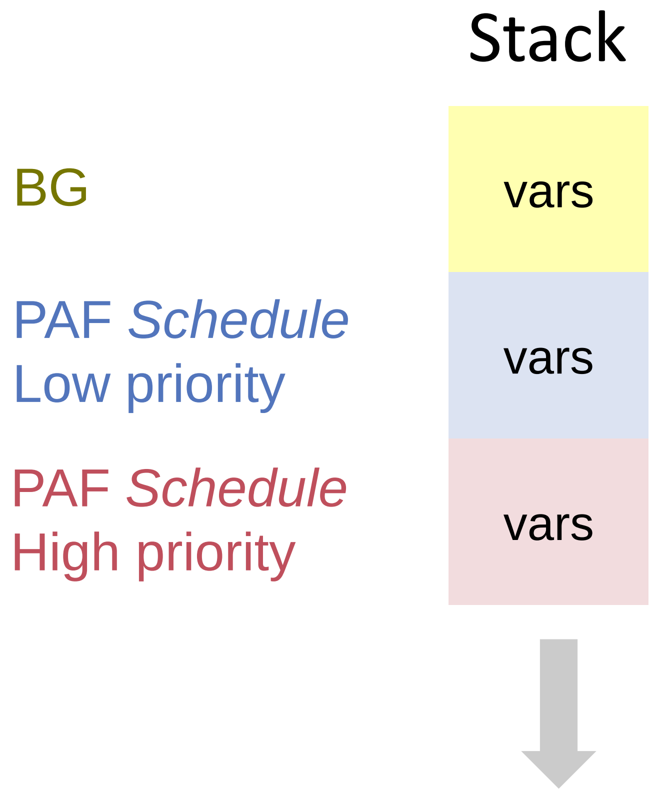
Run to completion!



Preemption



Stack Organisation



Where is the process context ?

Tasks

- Descriptors for *asynchronous* (background) tasks

```
Task*= POINTER TO TaskDesc;
TaskDesc* = RECORD
   next: Task;
   proc: TaskCode;
   name: ARRAY 32 OF CHAR;
END;
```

PROCEDURE (me: Task)

- Descriptors for *synchronous* (periodic) tasks

```
PeriodicTaskDesc* = RECORD (TaskDesc)
    interval: LONGINT;
    subPriority: LONGINT;
    nextTime: LONGINT;
END;
```

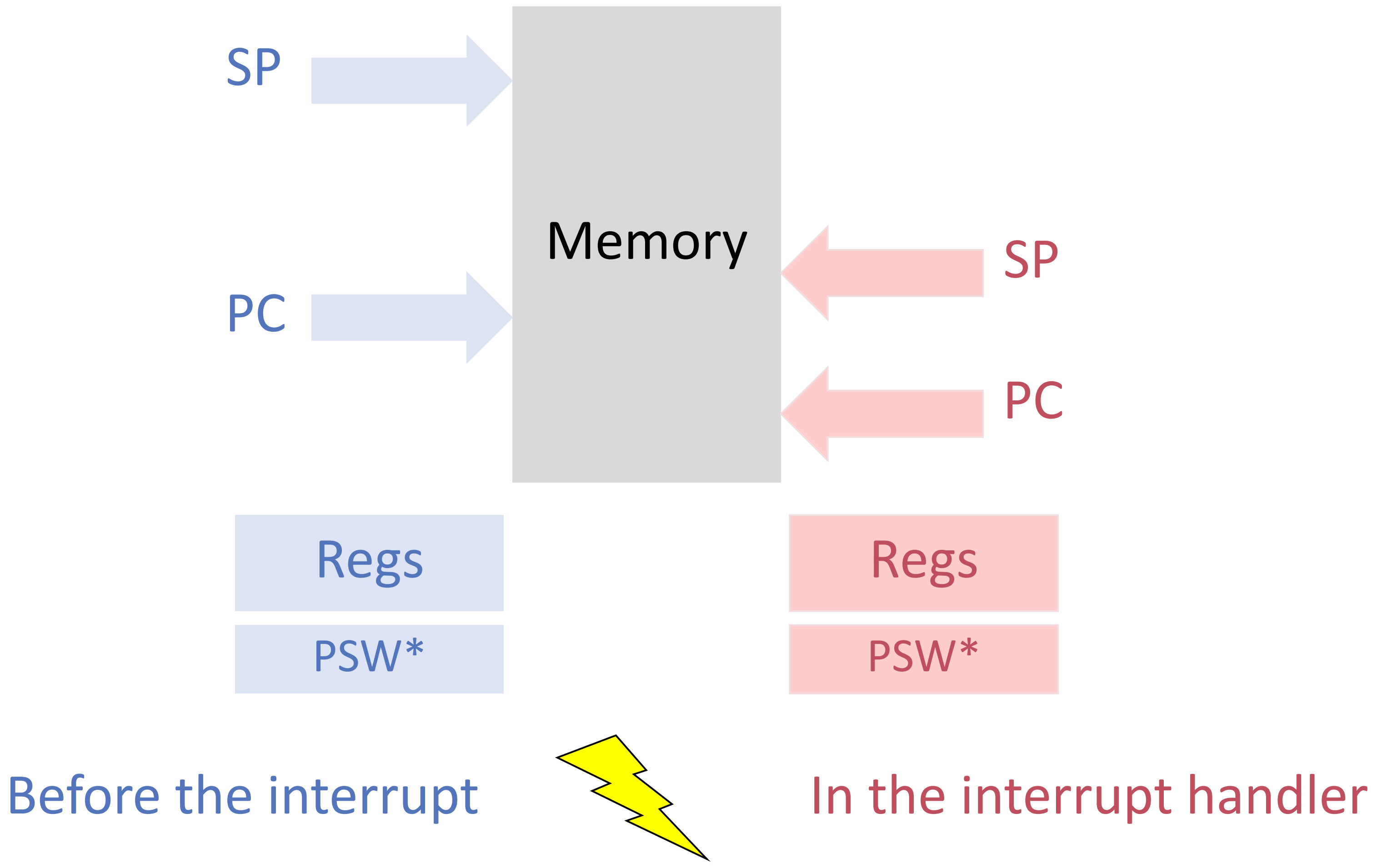
Scheduler

- Recursive interrupt procedure

```
Prolog (Interrupts masked)
Scheduling (Interrupts allowed)
Epilog(Interrupts masked)
```

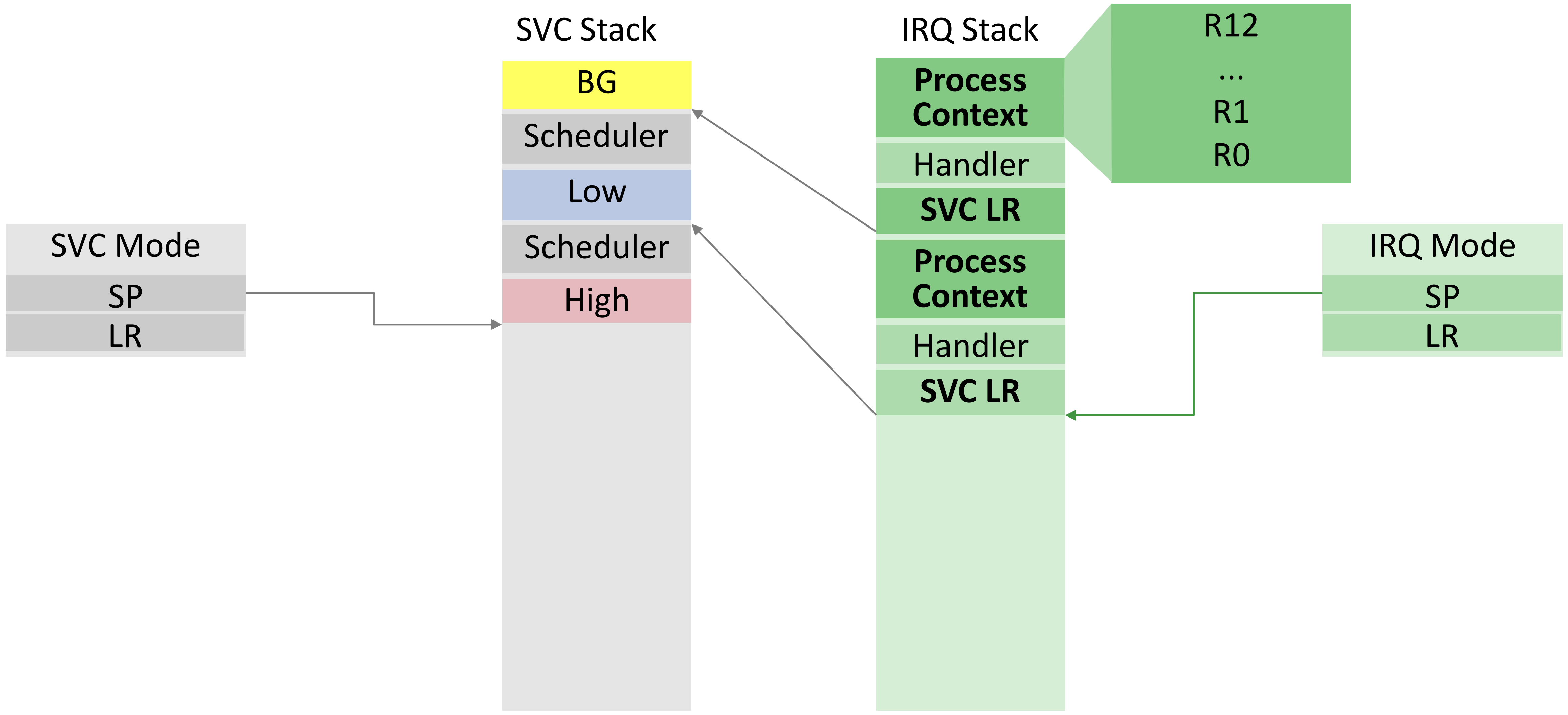
- Must be reentrant
  - Register values on stack
  - Private variables
- Assume that Interval(low) is a multiple of Interval(high)

Context change, schematic



*Processor Status Word

Process Context

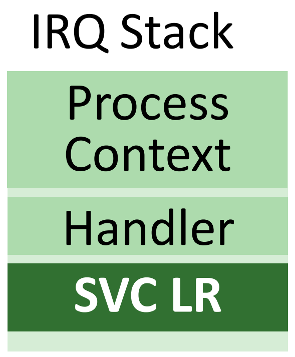


Some tricks required …

Kernel.TimerIrqHandler

```
VAR lr: INTEGER;
BEGIN
   INC( timer, Platform.UNIT );
   IF timerHandler # NIL THEN
      SYSTEM.LDPSR( 0, SVCMode + IRQDisabled );
      globalLR := SYSTEM.LNK();
      SYSTEM.LDPSR( 0, IRQMode + IRQDisabled );
      lr := globalLR;
      SYSTEM.LDPSR( 0, Platform.SVCMode );
      timerHandler;
      SYSTEM.LDPSR( 0, IRQMode + IRQDisabled );
      globalLR := lr;
      SYSTEM.LDPSR( 0, SVCMode + IRQDisabled );
      SYSTEM.SETLNK(globalLR);
      SYSTEM.LDPSR( 0, IRQMode + IRQDisabled );
END;
```

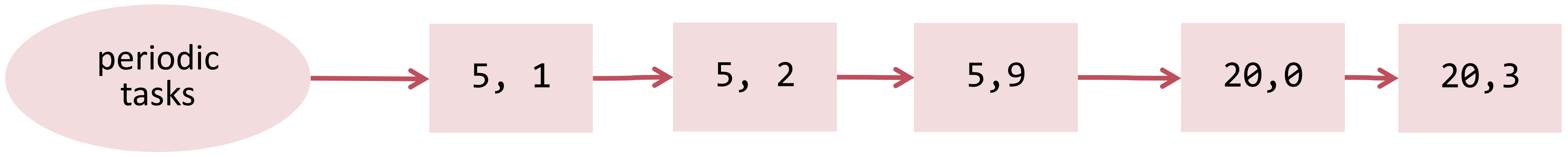
Switch to SVC mode, no IRQs



Scheduler Code

Assumptions:

- linked list stores tasks sorted by period / priority
- tasks run to completion within given period



Rate Monotonic Scheduling

Minos.Scheduler

```
currentTime := Kernel.GetTime();
current := periodicTasks;
WHILE current # NIL DO
   IF currentTime MOD current.interval = 0 THEN
      current.proc( current )
   END;
   current := current.next(PeriodicTask);
END;
```