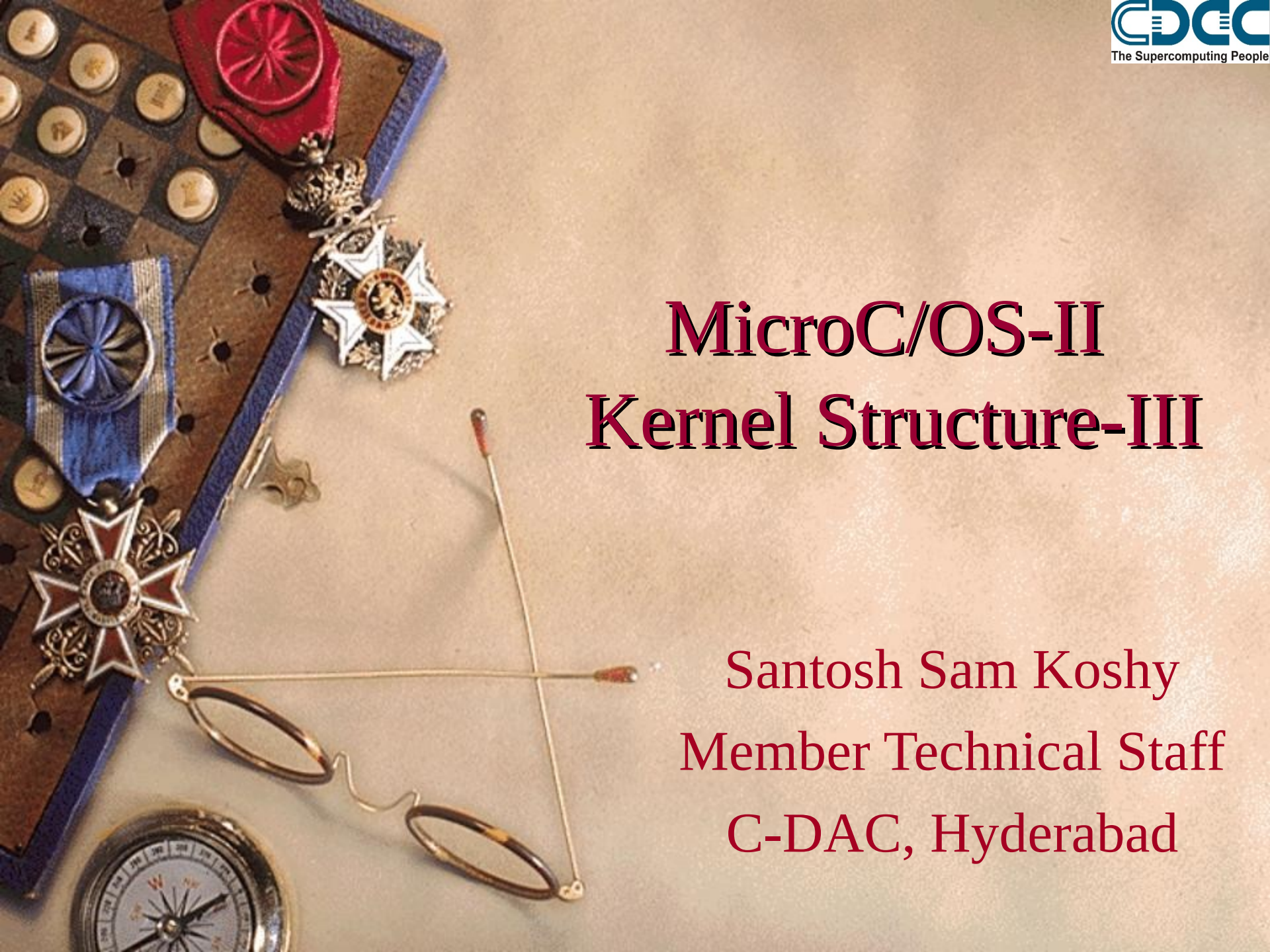


MicroC/OS-II
Kernel Structure-III

Santosh Sam Koshy
Member Technical Staff
C-DAC, Hyderabad

Task Scheduling

- ISR level scheduling
  - Explain later …
- Task level scheduling
  - `OSSched()`
    - Execute the highest priority task ready to run.
    - It will exit if
      - Called from an ISR.
        - *`OSIntNesting > 0`*
      - Scheduling has been disabled.
  - `OSSchedLock()`
    - To disable task scheduling
  - `OSSchedUnlock()`


```
void OSSched (void)
{
        INT8U y;
        OS_ENTER_CRITICAL();
        if ((OSLockNesting | OSIntNesting) == 0) {
            y  = OSUnMapTbl[OSRdyGrp];
            OSPrioHighRdy  =  (INT8U)((y  <<  3)  +
                                    OSUnMapTbl[OSRdyTbl[y]]);
          if (OSPrioHighRdy != OSPrioCur) {
            OSTCBHighRdy=OSTCBPrioTbl[OSPrioHighRdy];
            OSCtxSwCtr++;
            OS_TASK_SW();
            }
        }
        OS_EXIT_CRITICAL();
}
```

OS_TASK_SW()

OSCtxSw

- void OSCtxSw(void)
- {
  - PUSH Register set onto the current stack
  - OSTCBCur -> OSTCBStkPtr = SP
  - OSTCBCur = OSTCBHighRdy;
  - SP = OSTCBHighRdy -> OSTCBStkPtr
  - POP Register set from the stack
  - RET from Interrupt
- }

Locking Scheduler

```
void OSSchedLock (void)
   {
      if (OSRunning == TRUE) {
         OS_ENTER_CRITICAL();
         OSLockNesting++;
         OS_EXIT_CRITICAL();
      }
   }
```

UnLocking Scheduler

```
void OSSchedUnlock (void)
    {
       if (OSRunning == TRUE) {
          OS_ENTER_CRITICAL();
          if (OSLockNesting > 0) {
             OSLockNesting--;
             if ((OSLockNesting | OSIntNesting) == 0) {
                OS_EXIT_CRITICAL();
                OSSched();
             } else {
                OS_EXIT_CRITICAL();
             }
          } else {
             OS_EXIT_CRITICAL();
          }
       }
    }
```

Scheduling

- Caution
  - OSSchedLock() and OSSchedUnlock() must be used in pairs.
  - The task that calls OSSchedLock() keeps control of CPU forever!
  - After calling OSSchedLock(), your task must not make any kernel service that will suspend execution of the current task.
    - No other task will be allowed to run!
- Reference
  - OS_CORE.C

ISR

- Internal
  - A piece of code in assembly language.
  - If in line assembly is supported then it may be used to implement ISRs
  - Pseudo Code
    - ***ourISR:***

      Save all CPU registers;

      Call OSIntEnter();

      Execute user assembly to service ISR;

      Call OSIntExit();

      Restore all CPU registers;

      Execute a return from interrupt instruction;

ISR

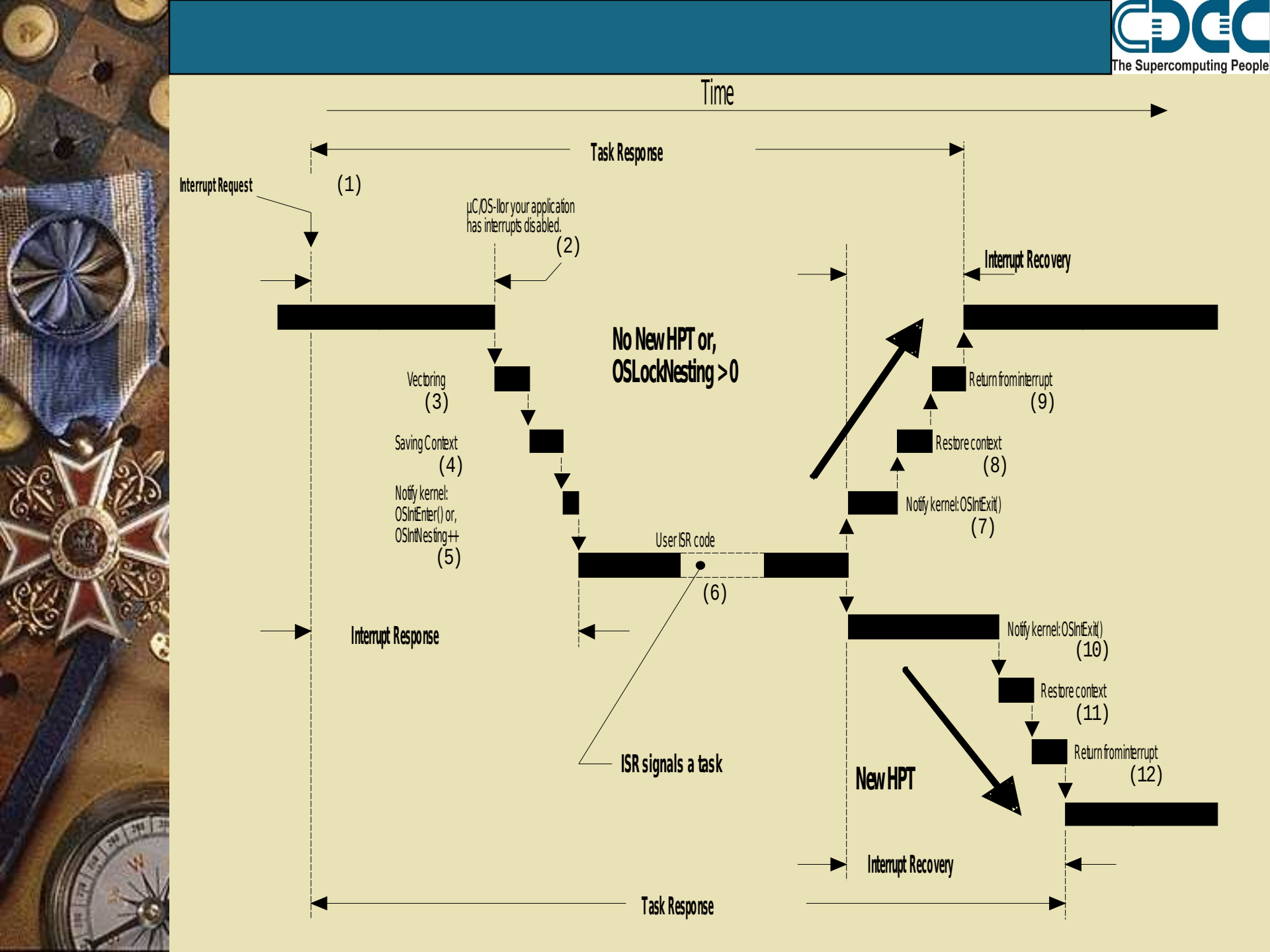
- OSIntEnter()
  - Notify µC/OS-II about beginning of ISR.
- OSIntExit()
  - Notify µC/OS-II about leaving of ISR.

ISR Entry

```
void OSIntEnter (void)
{
    OS_ENTER_CRITICAL();
    OSIntNesting++;
    OS_EXIT_CRITICAL();
}
```

ISR Exit

```
void OSIntExit (void)
  {
     OS_ENTER_CRITICAL();
     if ((--OSIntNesting | OSLockNesting) == 0) {
        OSIntExitY    = OSUnMapTbl[OSRdyGrp];
        OSPrioHighRdy = (INT8U)((OSIntExitY << 3) +
                        OSUnMapTbl[OSRdyTbl[OSIntExitY]]);
        if (OSPrioHighRdy != OSPrioCur) {
           OSTCBHighRdy  = OSTCBPrioTbl[OSPrioHighRdy];
           OSCtxSwCtr++;
           OSIntCtxSw();
        }
     }
     OS_EXIT_CRITICAL();
  }
```


Time
Task Response
Interrupt Request
(1)
µC/OS-II or your application has interrupts disabled.
(2)
Interrupt Recovery
No New HPT or, OSLockNesting > 0
Vectoring
(3)
Return from interrupt
(9)
Saving Context
(4)
Restore context
(8)
Notify kernel: OSIntEnter() or, OSIntNesting++
(5)
Notify kernel: OSIntExit()
(7)
User ISR code
(6)
Interrupt Response
Notify kernel: OSIntExit()
(10)
Restore context
(11)
Return from interrupt
(12)
ISR signals a task
New HPT
Interrupt Recovery
Task Response

Clock Ticker

- Periodic time source to keep track of time delay and timeouts.
  - OSTCBDly in TCB
- Must be enabled after multitasking has started.
  - Enable ticker interrupts at first task!
- Source
  - Hardware timer
  - AC power line (50/60 Hz)

OSTickISR

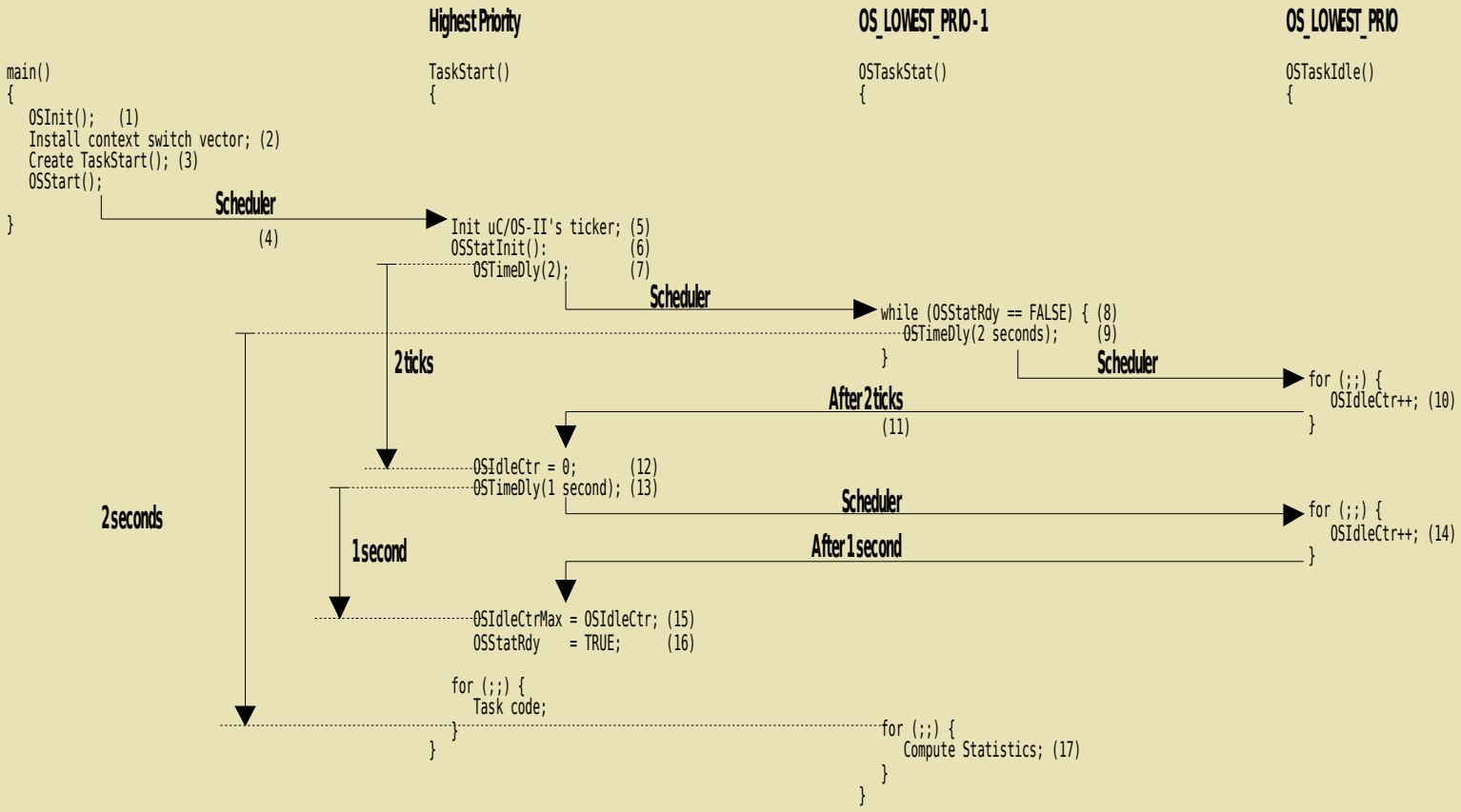
- Pseudo Code

```
void OSTickISR(void)
{
  Save processor registers;
  Call OSIntEnter();
  Call OSTimeTick();
  Call OSIntExit();
  {
        Restore processor registers;
        Execute a return from interrupt instruction;
  }
}
```

Clock Ticker

```
void OSTimeTick (void)
    {
      OS_TCB *ptcb;
      OSTimeTickHook();
      ptcb = OSTCBList;
      while (ptcb->OSTCBPrio != OS_IDLE_PRIO) {
        OS_ENTER_CRITICAL();
        if (ptcb->OSTCBDly != 0) {
          if (--ptcb->OSTCBDly == 0) {
            if (!(ptcb->OSTCBStat & OS_STAT_SUSPEND)) {
              OSRdyGrp               |= ptcb->OSTCBBitY;
              OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
            } else {
              ptcb->OSTCBDly = 1;
            }
          }
        }
        ptcb = ptcb->OSTCBNext;
        OS_EXIT_CRITICAL();
      }
      OS_ENTER_CRITICAL();
      OSTime++;
      OS_EXIT_CRITICAL();
    }
```

Ticker


Highest Priority
OS_LOWEST_PRIO-1
OS_LOWEST_PRIO
main()
{
OSInit(); (1)
Install context switch vector; (2)
Create TaskStart(); (3)
OSStart();
}
Scheduler
(4)
TaskStart()
{
Init uC/OS-II's ticker; (5)
OSStatInit(); (6)
OSTimeDly(2); (7)
Scheduler
OSTaskStat()
{
while (OSStatRdy == FALSE) { (8)
OSTimeDly(2 seconds); (9)
}
Scheduler
OSTaskIdle()
{
for (;;) {
OSIdleCtr++; (10)
}
2ticks
After2ticks
(11)
OSIdleCtr = 0; (12)
OSTimeDly(1 second); (13)
Scheduler
for (;;) {
OSIdleCtr++; (14)
}
2seconds
1second
After1second
OSIdleCtrMax = OSIdleCtr; (15)
OSStatRdy = TRUE; (16)
for (;;) {
Task code;
}
}
for (;;) {
Compute Statistics; (17)
}
}

Multitasking

- Initializing and starting µC/OS-II
  - ```
    void main (void)
    {
        OSInit();          // Initialize uC/OS-II
        .
        .
        Create at least 1 task;
        .
        .
        OSStart();         // Start multitasking!
    }
    ```
- `OSInit()`
  - Initialize all variables and data structures.
  - Create the idle task (`OSTaskIdle()`) which is always ready-to-run.
  - Create the statistic task (`OSTaskStat()`) optionally.
- `OSStart()`
  - Find the highest priority task that have created.
  - Never return to the caller!
    ```

Multitasking (Cont 1)

- Source Code
  - Start Multitasking
    - ```
      void OSStart (void)
      {
          INT8U y;
          INT8U x;

          if (OSRunning == FALSE) {
              y             = OSUnMapTbl[OSRdyGrp];
              x             = OSUnMapTbl[OSRdyTbl[y]];
              OSPrioHighRdy = (INT8U)((y << 3) + x);
              OSPrioCur     = OSPrioHighRdy;
              OSTCBHighRdy  = OSTCBPrioTbl[OSPrioHighRdy];
              OSTCBCur      = OSTCBHighRdy;
              OSStartHighRdy();
          }
      }
      ```
  - Obtaining µC/OS-II's version
    - ```
      INT16U OSVersion (void)
      {
          return (OS_VERSION);
      }
      ```

The Supercomputing People

The End

Of Kernel
Structure
PART-III