Advanced RTOS Issues: Responsiveness

Thread Priority

- Scheduler runs highest priority ready thread
- Can set thread priority upon creation
  - Initialize attributes (attr) first
  - Then call osThreadNew(func, arg, &attr)
- Can set thread priority during execution
  - osThreadGetPriority(threadID)
  - osThreadSetPriority(threadID, priority)


```
int main (void) {
  osThreadAttr_t attr;
  . . .
  //Set initial thread
  //  priority to high
  attr.priority = osPriorityHigh;
  osThreadNew(thread1, NULL, &attr);
  . . .
}
```

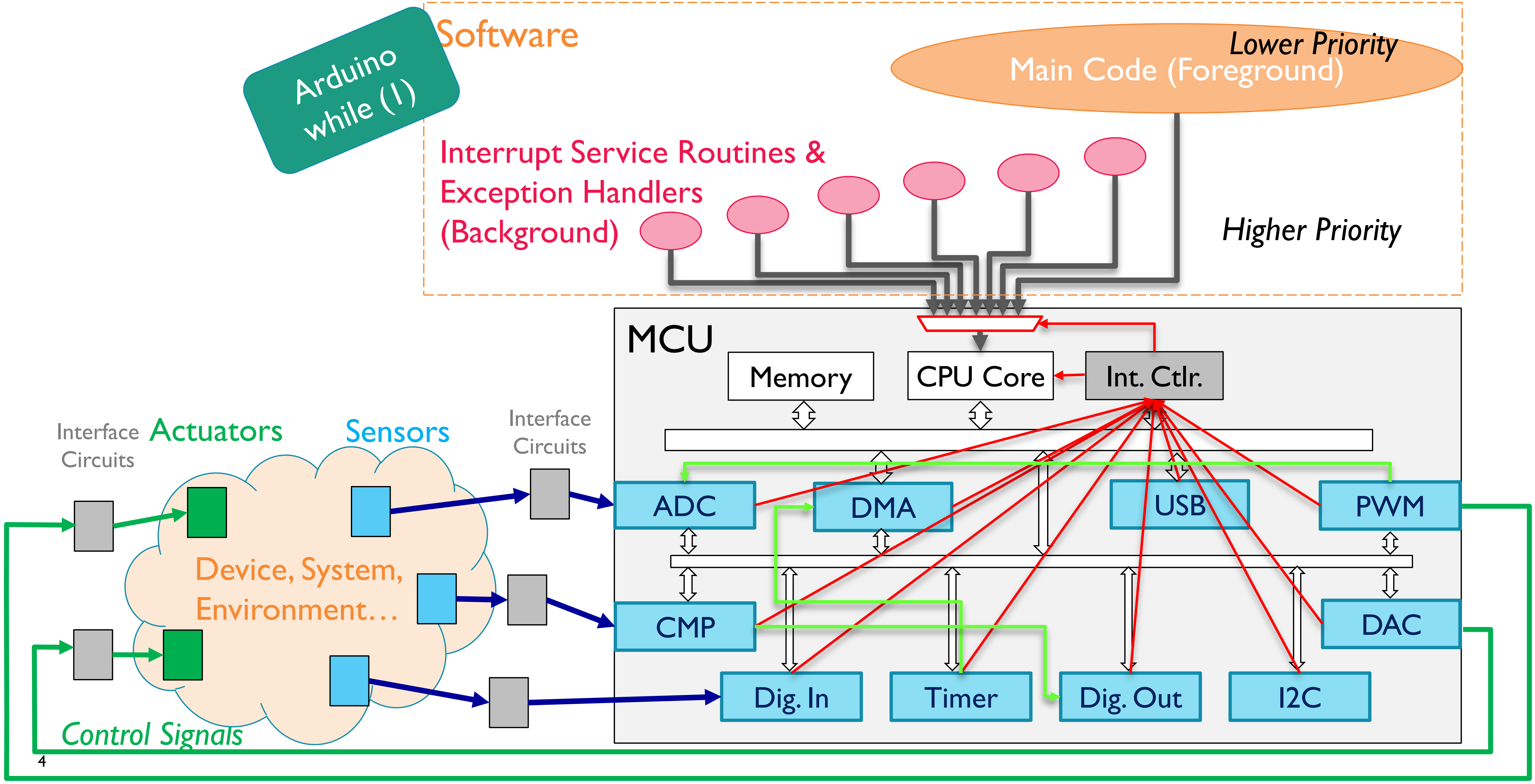
Priority Level
osPriorityRealtime7
. . .
osPriorityRealtime1
osPriorityRealtime
osPriorityHigh7
. . .
osPriorityHigh1
osPriorityHigh
osPriorityAboveNormal7
. . .
osPriorityAboveNormal1
osPriorityAboveNormal
osPriorityNormal7
. . .
osPriorityNormal1
osPriorityNormal
osPriorityBelowNormal7
. . .
osPriorityBelowNormal1
osPriorityBelowNormal
osPriorityLow7
. . .
osPriorityLow1
osPriorityLow
osPriorityIdle

Which Priorities to Assign to Threads?

- Field of *real-time systems* examines answers to this question
- Covered in ECE 461/561
- In general, **priority** rises with **urgency**
  - The sooner the thread must finish, the higher its priority

- Fixed vs. dynamic priority
  - Fixed:
    - Thread is assigned a fixed (static) priority
      - Typically at design time
    - Priority does not change…
      - Except for special cases with handling mutexes and priority inversion
      - Or when it is explicitly changed by program
  - Dynamic:
    - Thread has a deadline
    - Scheduler uses deadline when running to determine priority
- RTXv5 uses static thread priority assignment
  - But with some exceptions

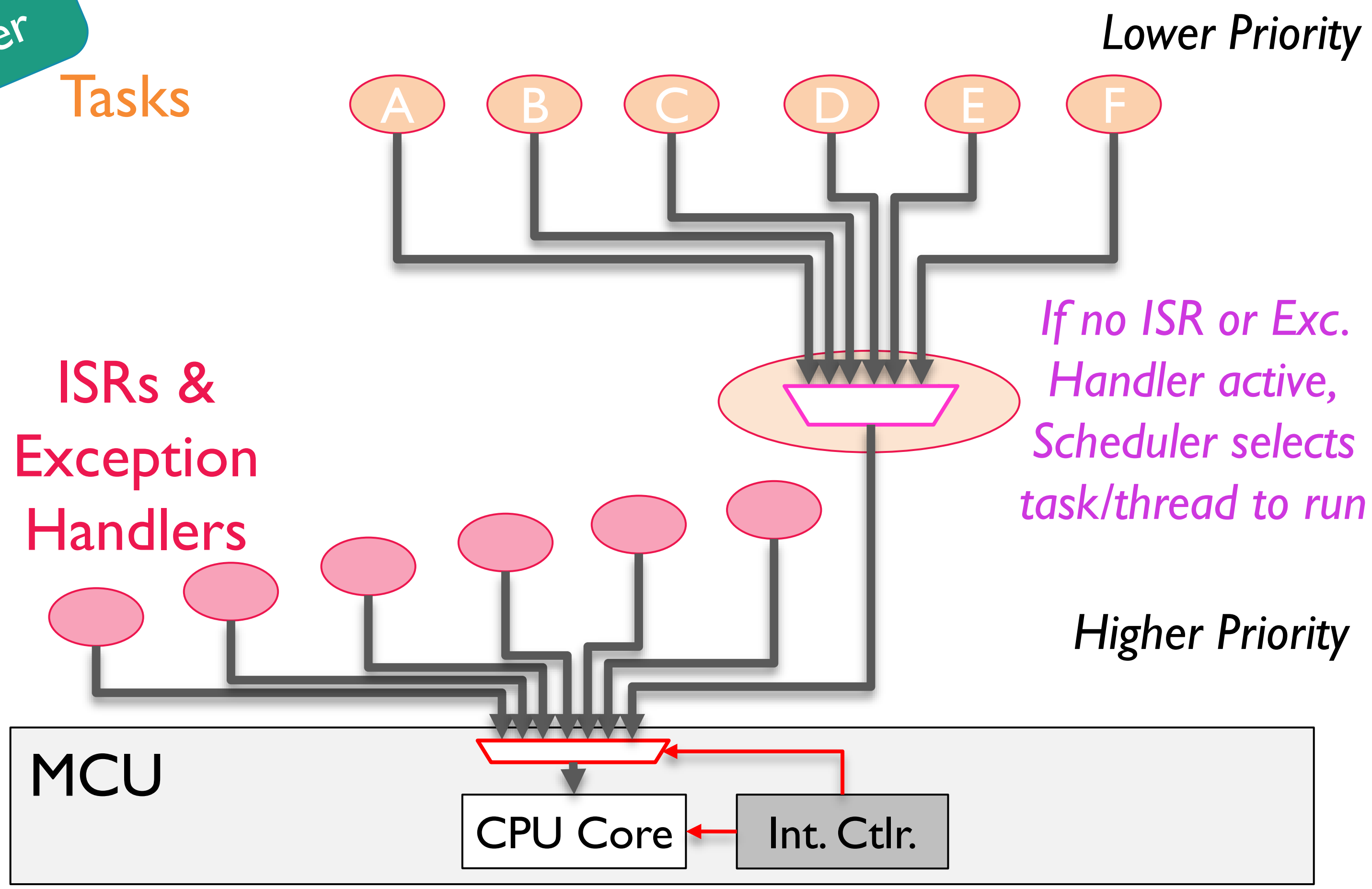
Parallel Hardware … but Serialized Software



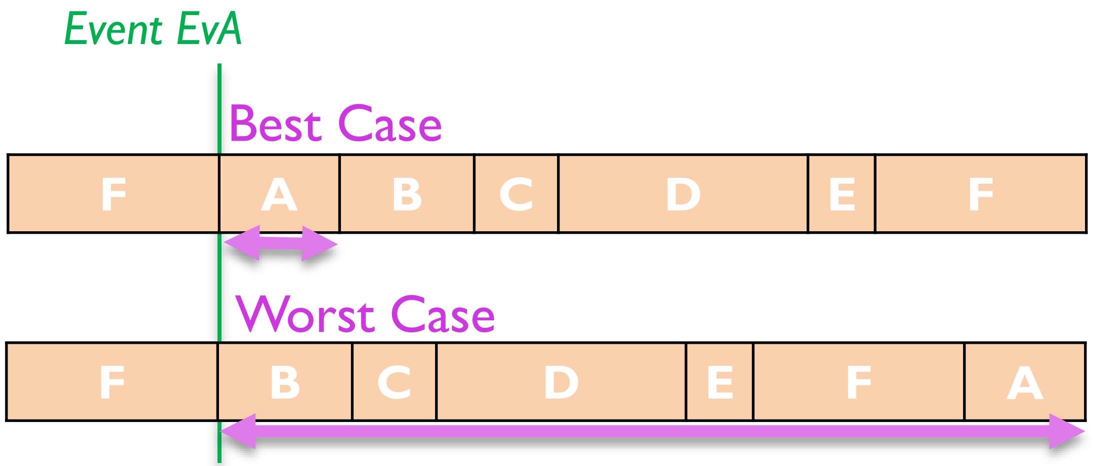
Schedulers: Helping Software Share the CPU Better

Arduino Scheduler

- Build modular program
  - Separate tasks/threads and ISRs, each running (mostly) independently
  - Easier to develop, maintain, debug
- What code does CPU run?
  - **Interrupt controller** decides code to run next.
    - Checks interrupt/exception requests **before** starting each instruction
  - If not, **task scheduler** decides which task/thread to run next

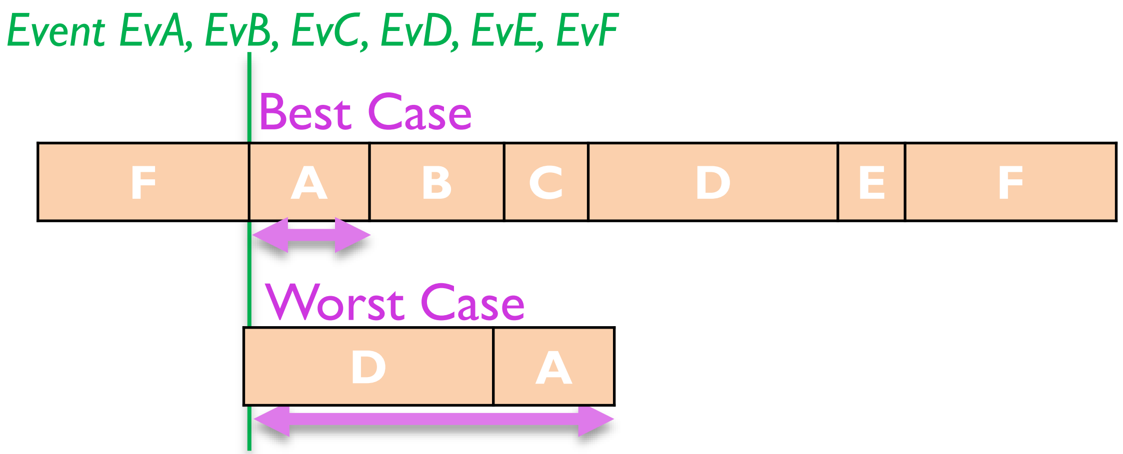


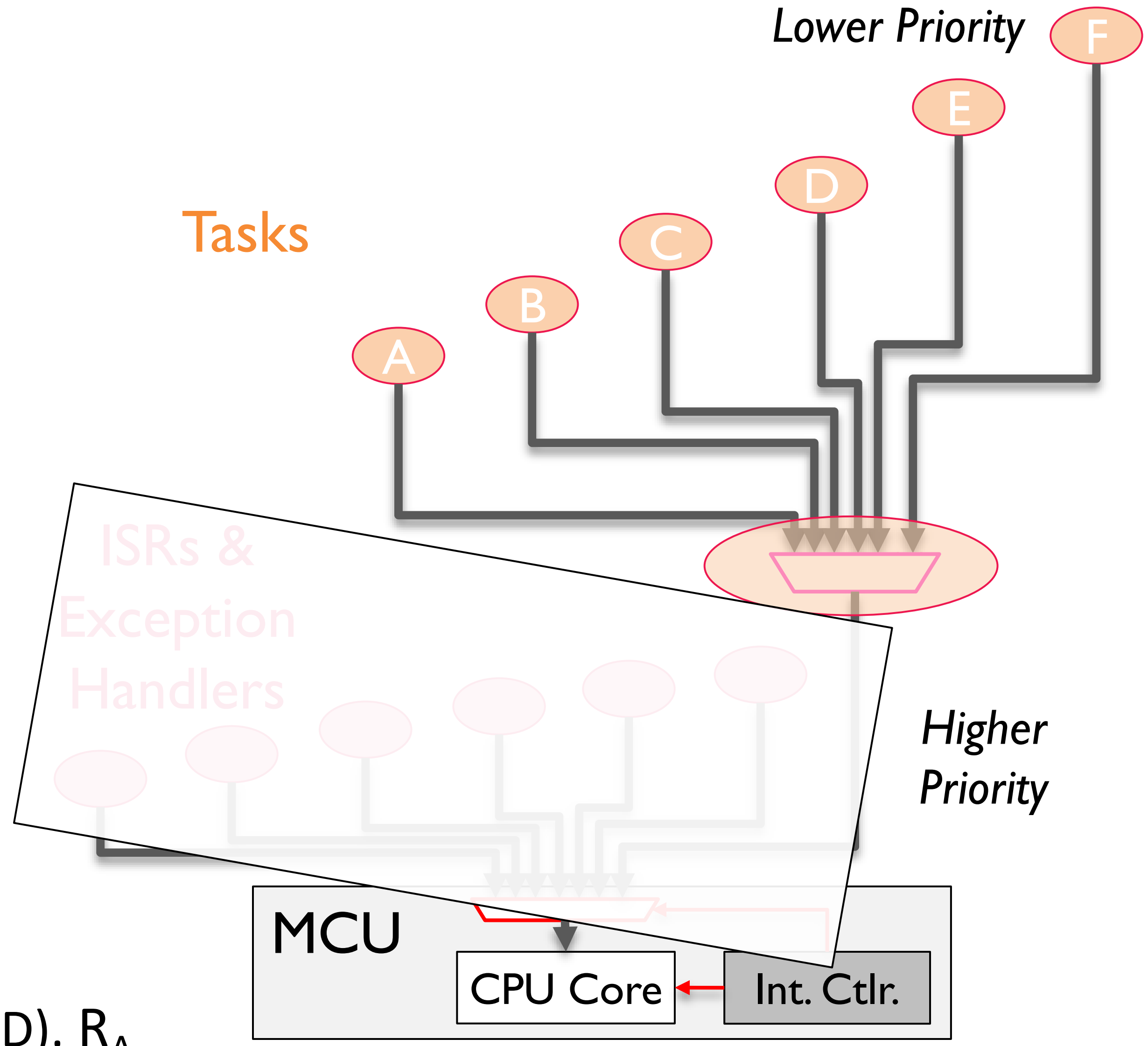
Responsiveness



- Task A must run to service event EvA
  - EvA makes scheduler ***release*** task A
- Scheduler behavior:
  - EvA happened? Release A, run A until done.
  - EvB happened? Release B, run B until done. Et cetera…
- Terms: No task preemption, round-robin task ordering. Task i takes C_i time of computation
- Task A's **response time** (R_A): How long from event EvA until task A **finishes** servicing it? Ignore time for handlers, scheduler for now.
  - Best case: EvA happens just before scheduler checks it. $R_A = C_A$
  - Worst case: EvA happened just after scheduler checked, every other event (EvB – EvF) happens before scheduler checks EvA again. $R_A = C_A + C_B + C_C + C_D + C_E + C_F$

Improvement: Prioritized Tasks

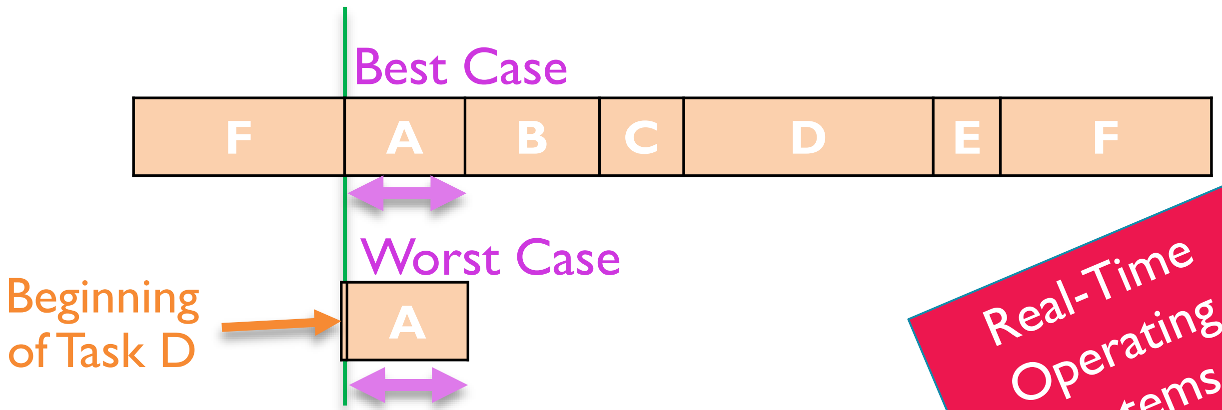




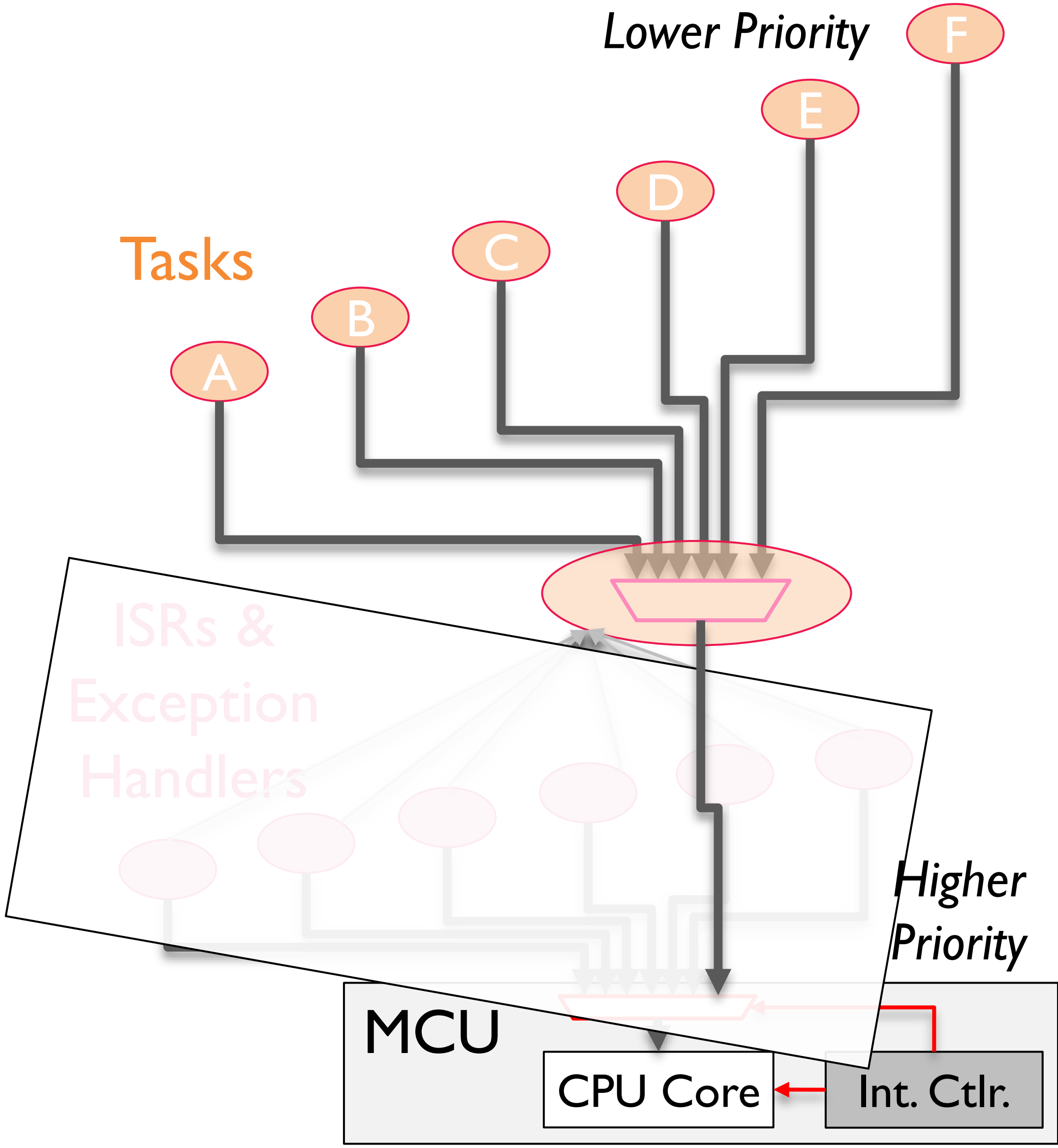
- Change scheduler to prioritize A > B > C etc.
- New behavior:
  - If EvA happened, run A until done.
    - Else if EvB happened, run B until done.
      - Else if EvC happened, run C until done.
        - Et cetera
- Best case: Same as before. $R_A = C_A$
- Worst case: Delayed only by longest other task (D). $R_A = C_A + \mathrm{Max}(C_B, C_C, C_D, C_E, C_F)$

Improvement: Preemptive, Prioritized Tasks

Event EvA, EvB, EvC, EvD, EvE, EvF

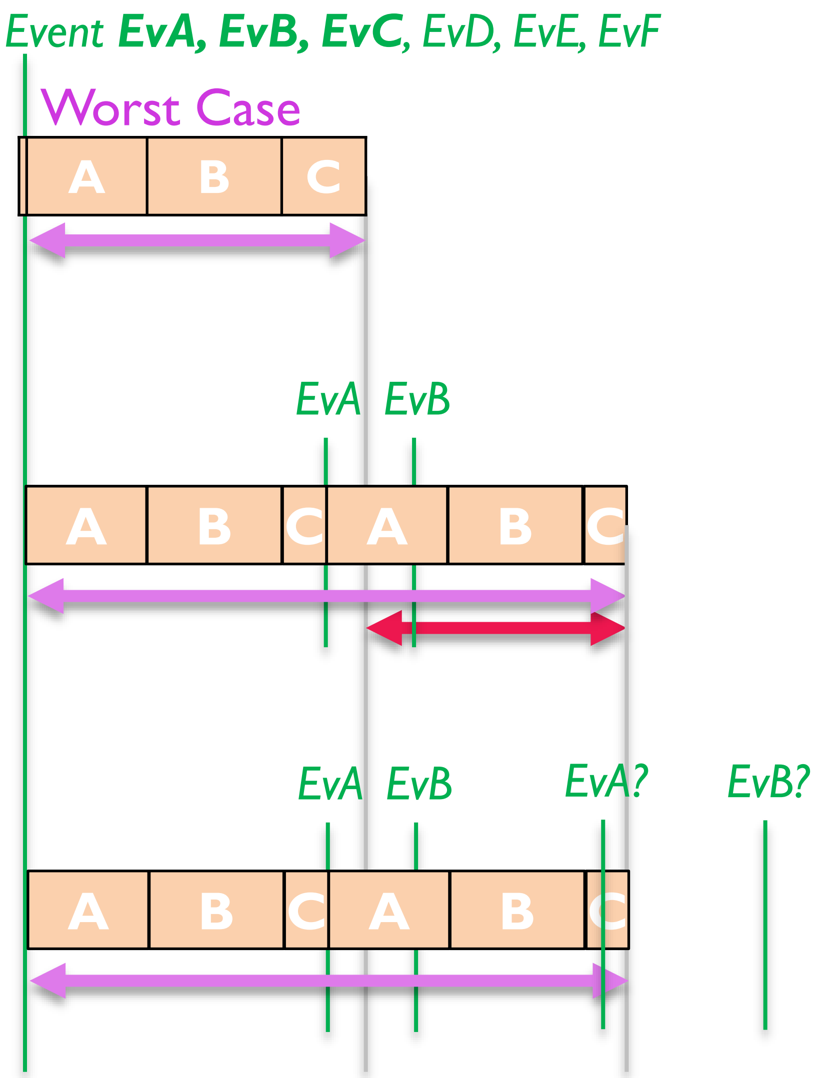


Real-Time Operating Systems



- Improvements
  - Let scheduler preempt lower priority task to run higher priority task when ready
  - And tell scheduler about events sooner, using interrupts and paying attention to clues from other tasks
- Delay doesn't depend on other tasks now, just task A.
  - $R_A = C_A$

What about Lower Priority Tasks?



- Task C's finish can be delayed **only** by higher priority tasks(A, B)
  - Initial worst-case estimate: Assume A, B will each delay C's finish once
  - $R_C = C_A + C_B + C_C$
- What if C still hasn't finished when A or B is released again?
  - A, B preempt C
  - $R_C = 2*C_A + 2*C_B + C_C$
- Must ***repeat*** to see if A or B are released again during this additional vulnerable time
  - Depends on minimum time between releases (A->A, B->B) in the worst case (burst)

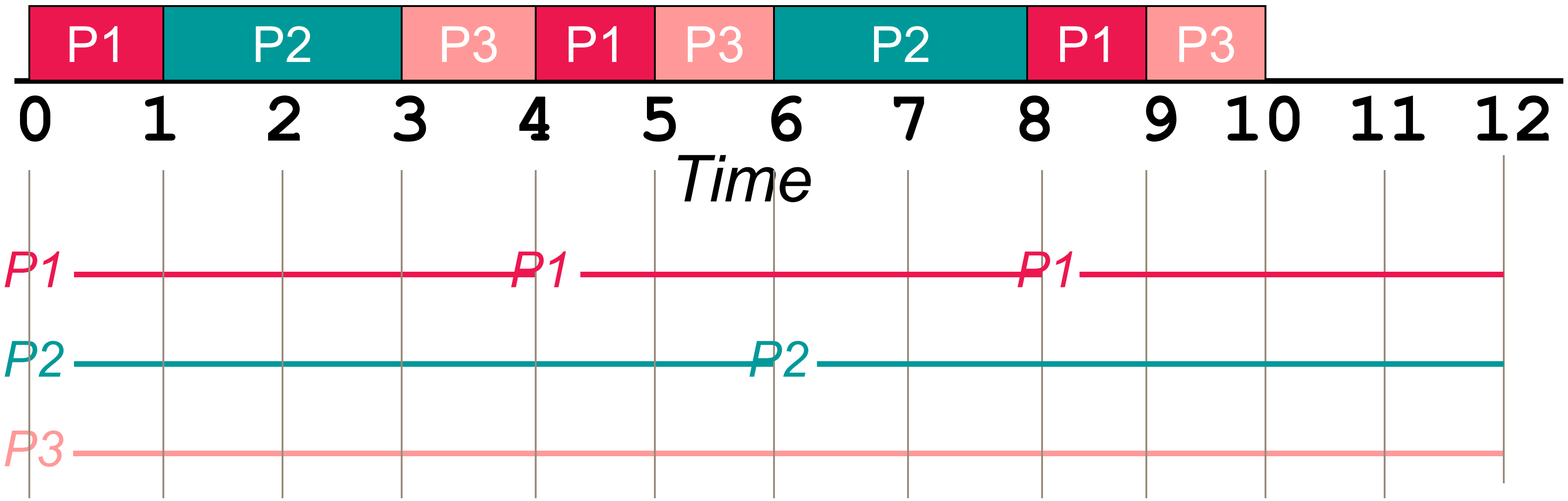
Periodic Task Model

- Periodic task model specifies
  - How often task is released to run: A minimum period T_i or maximum frequency per task
  - How long task takes to run once: Constant compute time requirement C_i per task

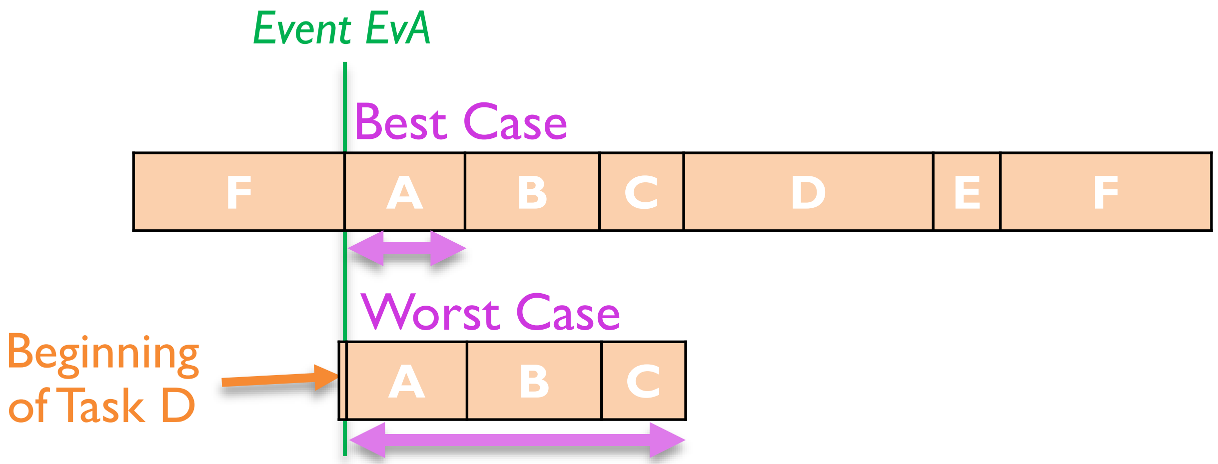
Diagram of Processor Activity

Task	Exec. Time C	Period T	Priority
P1	1	4	High
P2	2	6	Medium
P3	3	12	Low

Revise to match EvA, etc. from introduction

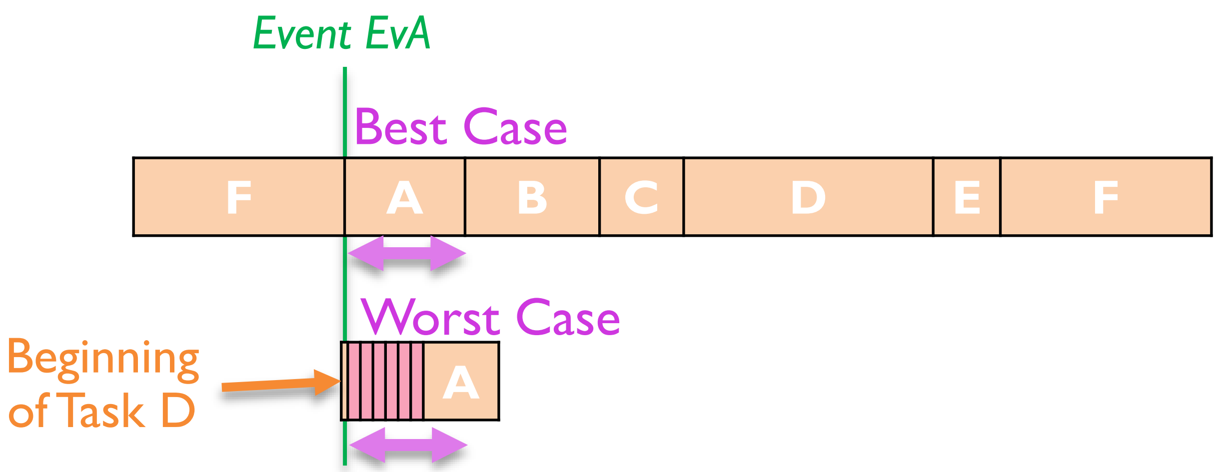


Repeated Calculation/Iteration until Stable

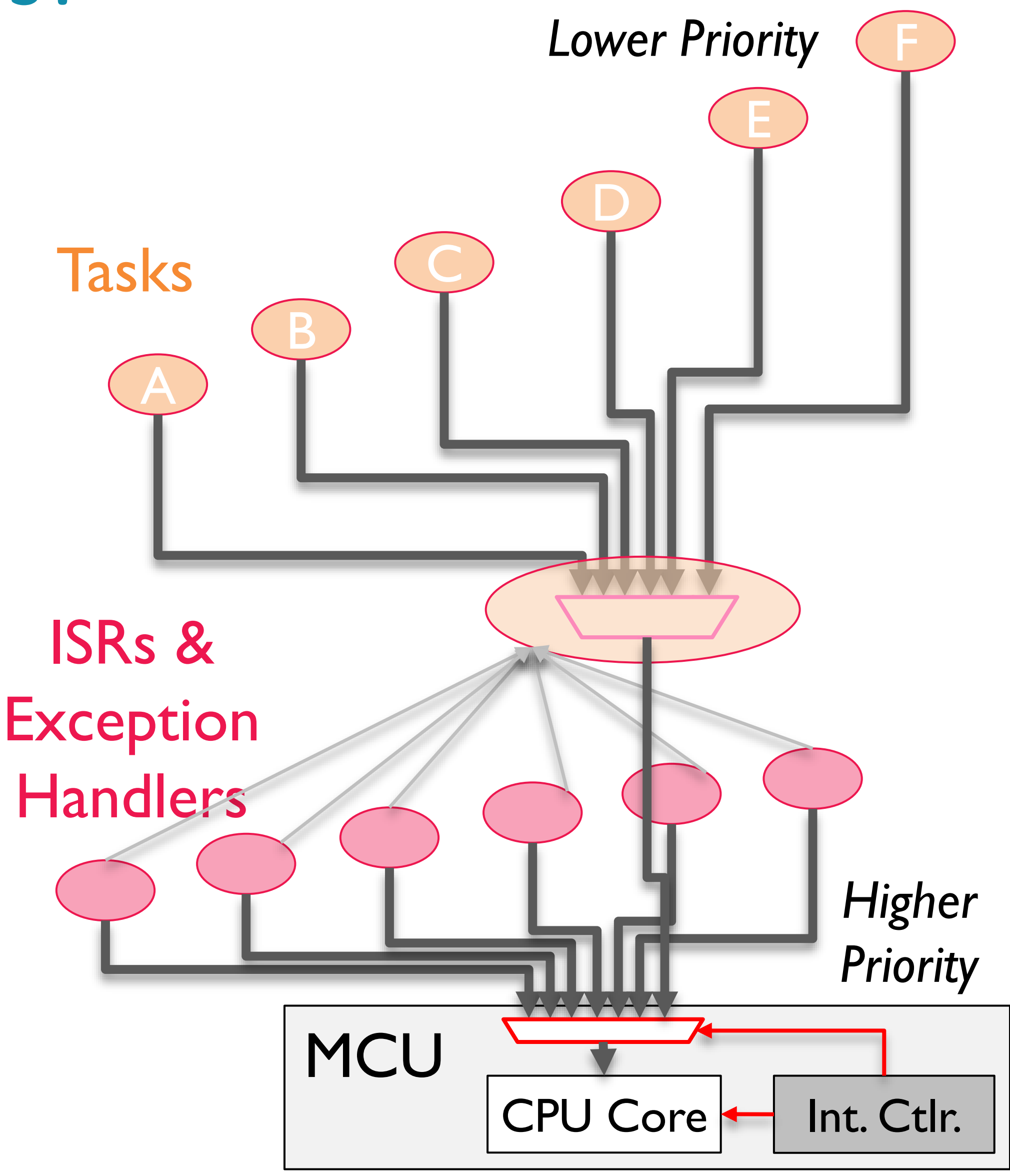


Complete

What about ISRs and Exception Handlers?



- Let's stop ignoring the ISRs/exception handlers, as they can preempt tasks too



Generic Numerical Response Time Analysis

- How long could it take for task *i* to complete?
- Response time = Computation time + Delays
- Start with estimate R_i of task *i*'s response time at *critical instant*
- Then compute new completion time R_i' based on *blocking* and *interference* from *new job arrivals*
- Repeat until R_i' stops changing

$$R_i = C_i + Initial\ Delays_i$$

$$R_i' = C_i + Delay_i(R_i)$$

Reducing the Delay

$$R_i' = C_i + Delay_i(R_i)$$

- Base case: no priority or preemption

$$Delay_i(R_i) = \sum_{j \neq i} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

- Add task prioritization
  - Splits delays into **blocking** and **interference**
  - Eliminates **most** effects from lower priority tasks

$$Delay(R_i') = B_i(R_i) + I_i(R_i)$$

$$Delay(R_i') = \max_{j \in lp(i)}(C_j) + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

- Add task preemption
  - Eliminates blocking of task *i* by lower priority tasks if they don't communicate with it.
  - Unrealistic, will revisit this.

$$Delay(R_i') = \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Fixed Priority, Preemptive Response Time Analysis

- How long could it take task *i* being released to completing?
  - Response time = Task i Computation + Blocking + Interference
  - Can use this to evaluate **schedulability**: Will tasks with deadlines meet them?

$$R_i = C_i + B_i + I_i$$

- Details
  - Start with estimate R_i of task *i*'s response time at *Critical Instant* (all tasks released simultaneously)
  - Then determine possible *interference* from *additional job arrivals* and update completion time estimate
  - Recompute based on any new arrivals until no change in R_i'

$$R_i = C_i + B_i + \sum_{j \neq i} C_j$$

$$R_i' = C_i + B_i + \boxed{\sum_{j \neq i} \left\lceil \frac{R_i}{T_j} \right\rceil C_j}$$

Response Time Example (Without Blocking)

Task Name	Index i	Exec. Time C_i	Period T_i	Priority
A	1	1	4	High
B	2	2	6	Medium
C	3	3	12	Low

Iterate until R_3 stops changing

$$R_i = C_i + \sum_{j=1}^{i-1} C_j$$

$$R_3 = 3 + \sum_{j=1}^{i-1} C_j = 3 + 1 * 1 + 1 * 2 = 6$$

$$R_3 = 3 + \sum_{j=1}^{i-1} \left\lceil \frac{6}{T_j} \right\rceil C_j = 3 + \left\lceil \frac{6}{4} \right\rceil * 1 + \left\lceil \frac{6}{6} \right\rceil * 2 = 3 + 2 * 1 + 1 * 2 = 7$$

$$R_3 = 3 + \sum_{j=1}^{i-1} \left\lceil \frac{7}{T_j} \right\rceil C_j = 3 + \left\lceil \frac{7}{4} \right\rceil * 1 + \left\lceil \frac{7}{6} \right\rceil * 2 = 3 + 2 * 1 + 2 * 2 = 9$$

$$R_3 = 3 + \sum_{j=1}^{i-1} \left\lceil \frac{9}{T_j} \right\rceil C_j = 3 + \left\lceil \frac{9}{4} \right\rceil * 1 + \left\lceil \frac{9}{6} \right\rceil * 2 = 3 + 3 * 1 + 2 * 2 = 10$$

$$R_3 = 3 + \sum_{j=1}^{i-1} \left\lceil \frac{10}{T_j} \right\rceil C_j = 3 + \left\lceil \frac{10}{4} \right\rceil * 1 + \left\lceil \frac{10}{6} \right\rceil * 2 = 3 + 3 * 1 + 2 * 2 = 10$$