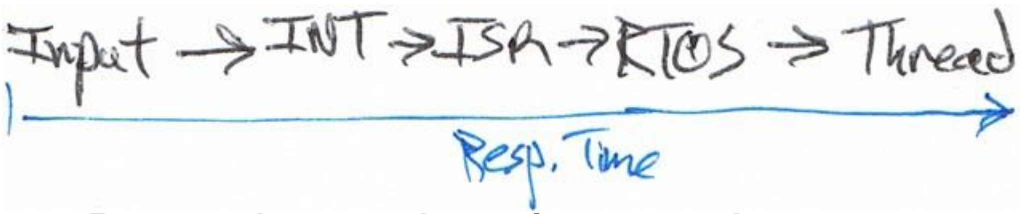
Analyzing Responsiveness for Real-Time Systems

Overview

- In these slides
  - Examining Response Time for Shield Audio Software
  - Periodic Task Model and Scheduling
  - Numerical Response Time Analysis
  - Deadlines, Priority Assignment and Schedulability Tests
- Next slides
  - Estimating Task Execution Time
  - Analyzing Priority Inversion

Response Time Matters to **Real-Time** Systems

- Response time
  - Delay from input (release) to output (completion)

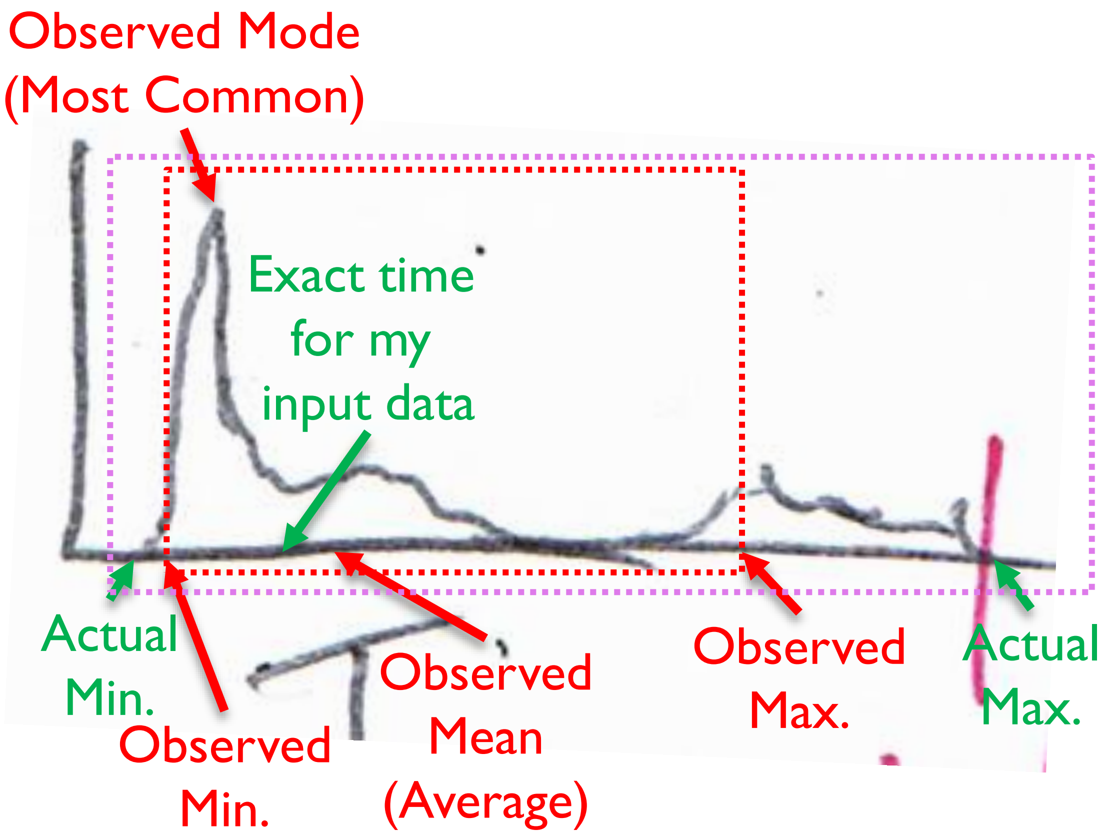


  - Depends on what else is in the system
- Response time is important to all programs, but more important to some than others
  - Antilock brakes in car, truck, aircraft
  - Lawn irrigation system
- Response time is a range of values

  - Different paths through program
  - Different machine state, not architecturally visible (pipelines, predictors, caches, etc.)
  - Interference from rest of system
  - Sampling asynchronous inputs
- If timing is critical, must understand it
  - Average
  - Extreme cases: best and worst cases (minimum and maximum)

Real-Time Methods

- Want to predict timing of system
  - Ideal:
    - Exact response time, given inputs and state
    - Extremes, and inputs and states which cause them
  - Wrong (but maybe useful):
    - Measure times experimentally and note extremes
    - Analytical bounds on extremes
- Bounds and tightness
  - Tighter bounds are better,* but harder to determine
    - * actually "*not worse*". Would knowing the 0 to 20.83 ns value from sampling with the 48 MHz system I/O clock matter for this system?
- **Real-Time Methods** make it easier to build a system with predictable timing
  - Design methods: how to build a system
  - Analysis methods: how to analyze a system
    - Usually include simplifying assumptions.
    - Some design methods are easier to analyze than others



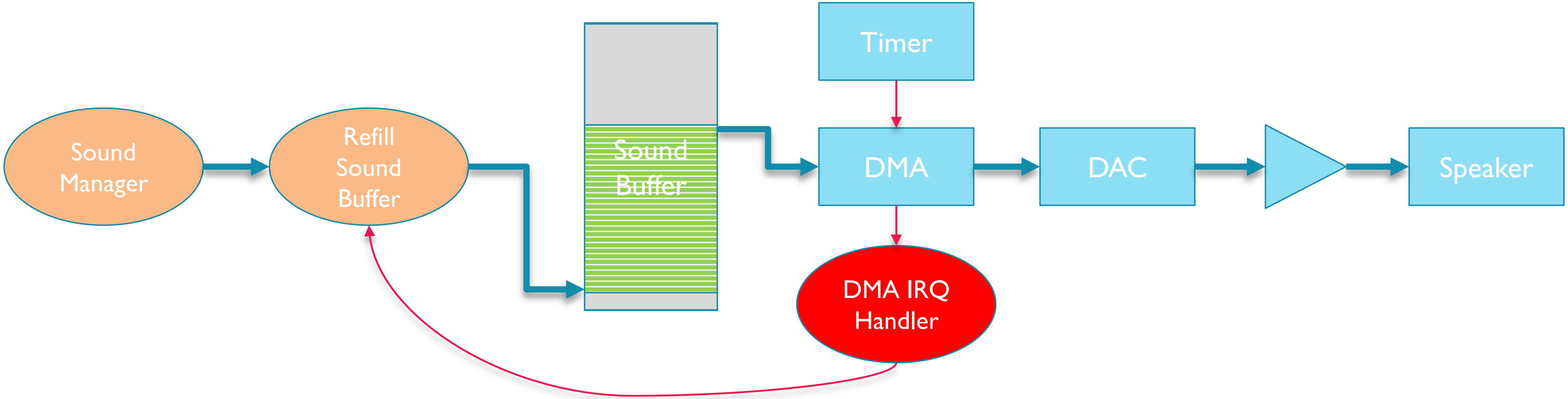
Big Picture of Real-Time Systems

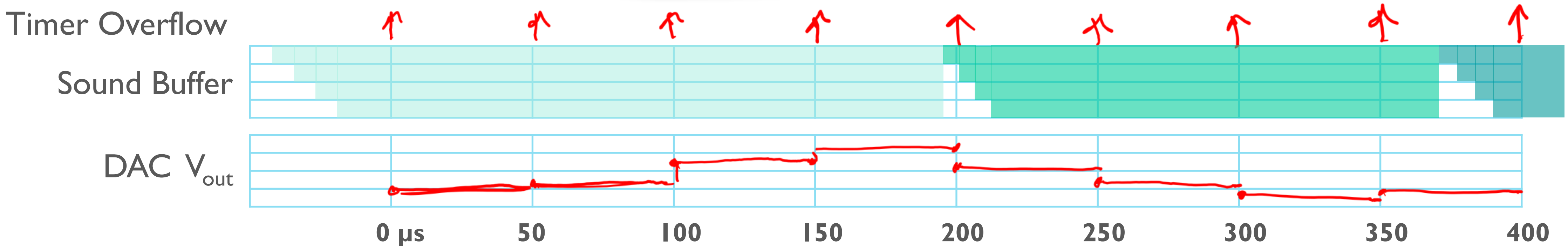
Concepts — *Basic Ideas* — *Details and Improvements*

- **Task Model**
  - When does task run?
    - Periodically
    - Everything else
  - How long task takes: Execution time analysis
    - Analytical – safe
    - Experimental – unsafe
- **Scheduling**
  - Preemption vs. run-to-completion
  - Priorities
    - Fixed
    - Dynamic
  - Overheads
    - Hardware: Interrupts
    - OS: Context Switching, IPC
- **Response Time Analysis**
  - Basic methods
  - Blocking
  - Handling priority inversion
- **Deadlines**
  - Using response time analysis
  - Optimal scheduling approaches
  - Schedulability tests
  - Comparing impact of preemption and priority

EXAMPLE: REAL-TIME AUDIO GENERATION ON THE SHIELD

Shield Audio System Architecture (Single Buffer)





- f_{sample} = 20 kHz, T_{sample} = 50 μs
- Sample deadline every 50 μs

- Sound buffer (waveform)
  - Array of 512 halfwords
  - Holds 512 * 50 μs = 25.6 ms of data
  - Refill buffer deadline every 25.600 ms

Questions

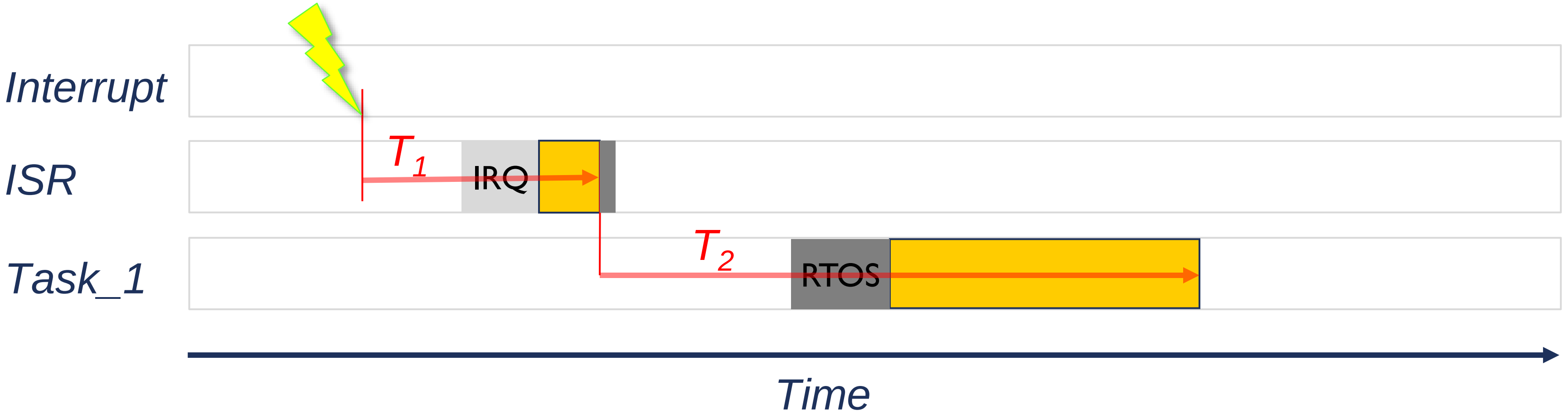
- Audio generation
  - Can we miss any audio deadlines?
    - If not, how close can we get to missing one?
  - How much of CPU's time is used on audio generation?
  - How slowly can we run the CPU while keeping audio working?
  - How fast of an audio sample rate can we manage with f_{CPU} = 48 MHz?
- Generalized
  - On average, how much of the CPU's time is used, and how much is free?
  - What is the worst-case response time for each task?
  - If we have deadlines, can we miss any? How close can we get to missing one?

Assumptions and Definitions

- Assumptions
  - Single CPU
  - Context switch takes no time
  - No data dependencies between tasks unless explicitly specified and modeled

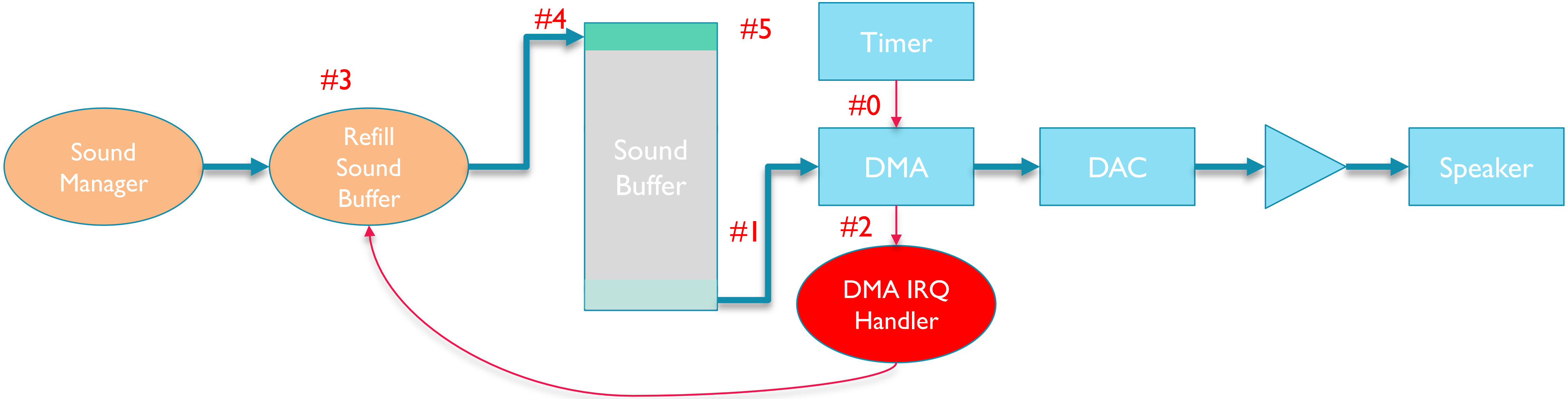
- Definitions
  - Release Time = time at which event occurs or when task is released (depends on context)
    - e.g. timer overflow
  - Completion Time = time at which task finishes
  - Response time = completion time – release time
  - Deadline = time at which task must have completed
  - “Schedulable” = a schedule exists which allows each task to meet its deadline

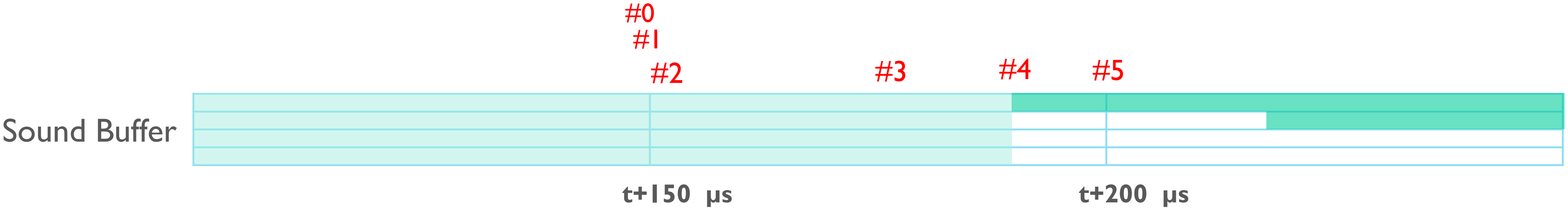
Evaluating Responsiveness



- DMA ISR and one dependent task
- Assumption
  - ISR or task signals next task after its critical work is completed
- Two important components in the critical path
  - T_1: From interrupt request to ISR running and completing critical work. Uses MCU interrupt hardware.
  - T_2: From ISR to user task running and completing critical work. Uses OS signaling.

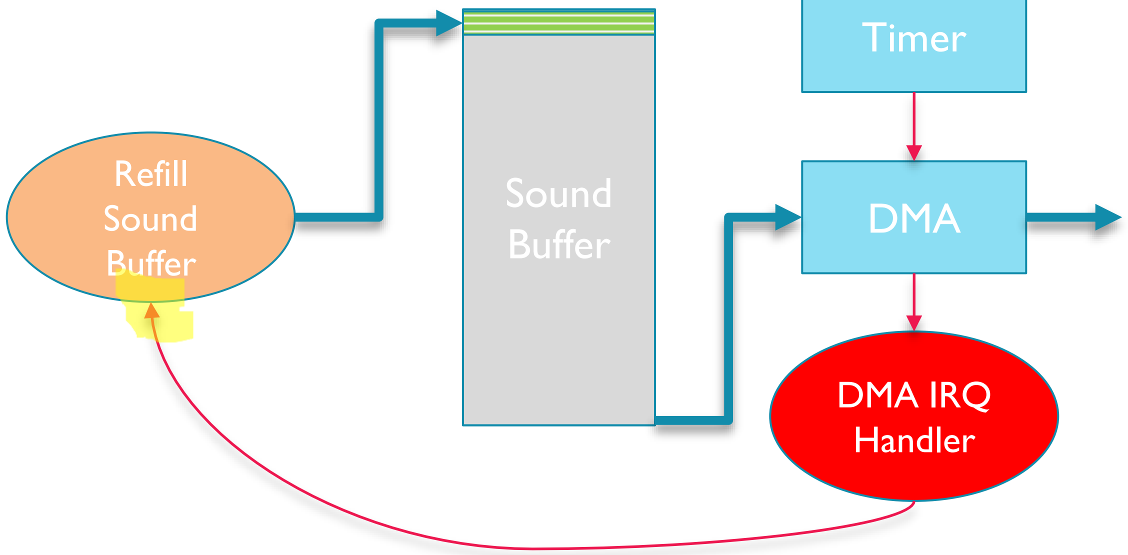
What Happens at the Last Sample (Single Buffer)?

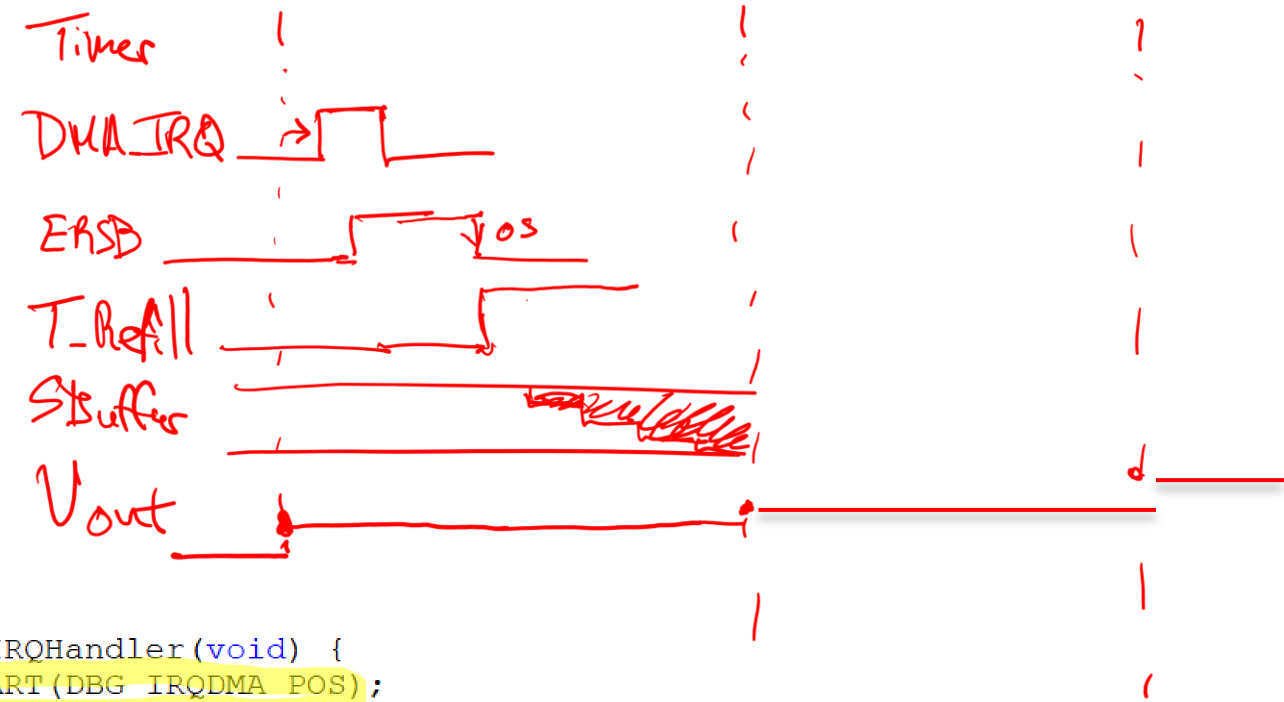




- Need to load first new sample into buffer before DMA reads it. Have <50 μs.
- Need to load second sample before 100 μs
- Sample[n] is needed before n*50 μs

DMA0_IRQHandler

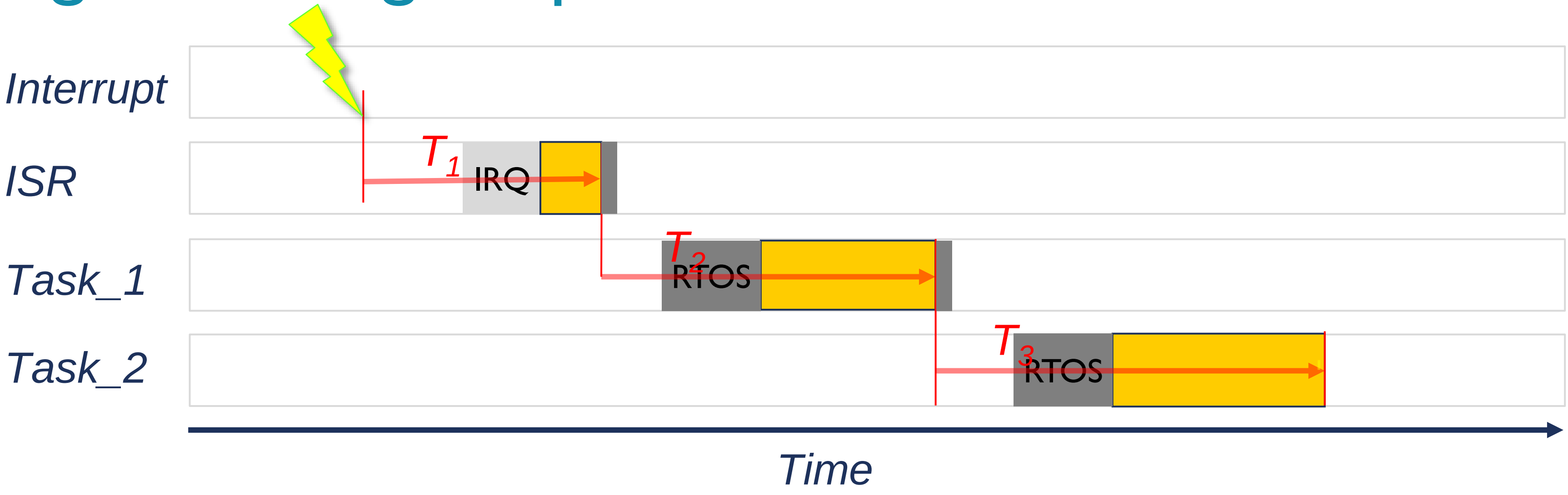




- Handler needs to…
  - Tell hardware it is handling the DMA interrupt
  - Tell someone to refill sound buffer
  - Tell the DMA hardware to play the buffer again, starting with the next timer trigger

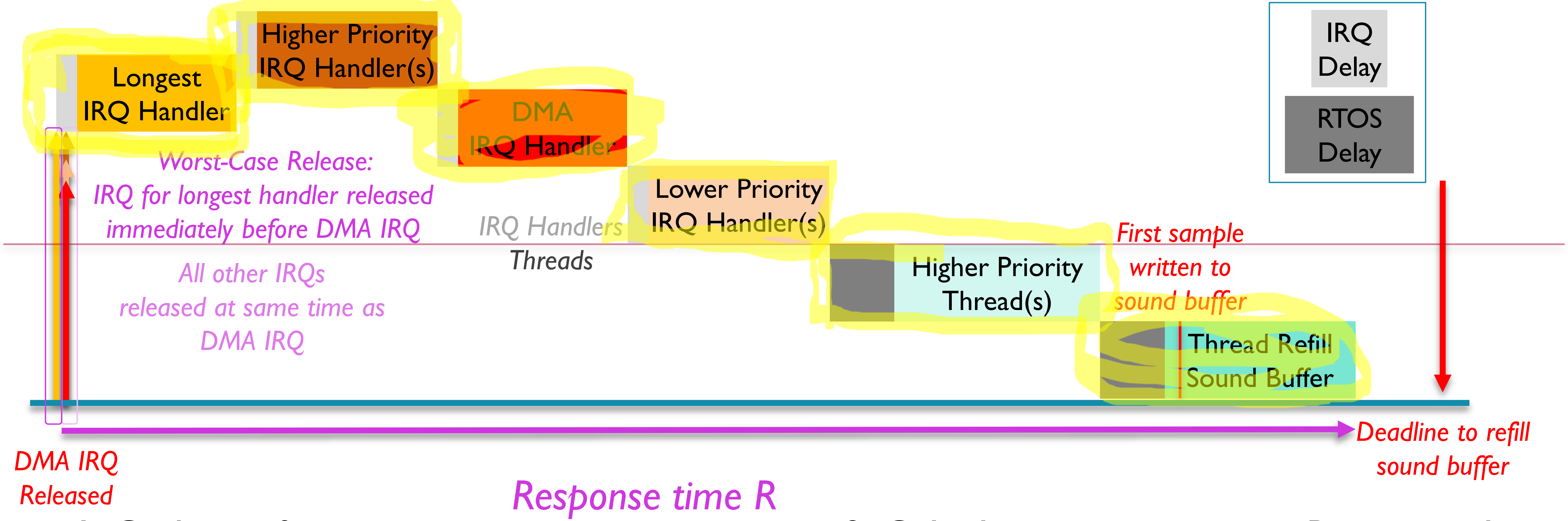
```
void DMA0_IRQHandler(void) {
  DEBUG_START(DBG_IRQDMA_POS);

  // Clear done flag
  DMA0->DMA[0].DSR_BCR |= DMA_DSR_BCR_DONE_MASK;

  if (--DMA_Playback_Count > 0) {
    // Signal event requesting source buffer refill
    osThreadFlagsSet(t_Refill_Sound_Buffer, EV_REFILL_SOUND_BUFFER);
#if USE_DOUBLE_BUFFER
    // switch to other buffer
    read_buffer_num = 1 - read_buffer_num;
#endif
    Control_RGB_LEDs(0,0,read_buffer_num);
    // Start playback again
    Start_DMA_Playback();
  }
  DEBUG_STOP(DBG_IRQDMA_POS);
}
```

Generalizing Evaluating Responsiveness



- May have multiple sequential dependent tasks
- Assumption
  - ISR or task signals next task after its critical work is completed
- Three important components in the critical path
  - T_1: From interrupt request to ISR running and completing critical work. Uses MCU interrupt hardware.
  - T_2: From ISR to user task running and completing critical work. Uses OS signaling.
  - T_3: From one user task to another user task running and completing critical work. Uses OS signaling.

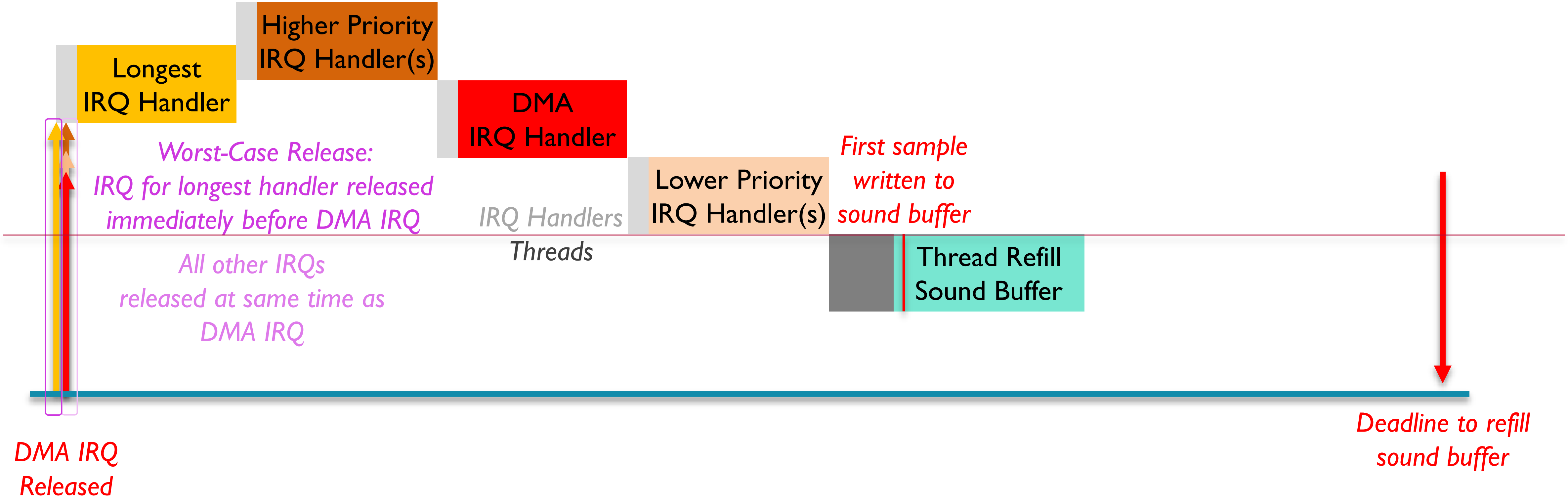
Critical Path Analysis for Sound Buffer Refill Sequence



- 1. Gather information
  - Which activities are in critical path
  - How long each activity takes
- 2. Calculate response time R iteratively
  - Estimate of R assuming everything released simultaneously (*critical instant analysis*)
  - More work may have arrived during R, delaying our thread, so update R
  - Repeat until R stabilizes or exceeds deadline

NC STATE UNIVERSITY

Improvement: Give Refill Sound Buffer Thread the Highest Priority



- Now no threads have higher priority than Thread Refill Sound Buffer

Big Picture of Real-Time Systems Analysis and "Optimization"

- Development Cycle
  - Think
  - Modify
  - Test and measure
- Measurement is critical
  - Expectations ≠ Reality
  - Want to measure to find biggest problem, attack that first
  - Want to see if our changes help or not
- How to measure?
  - Use embedded instruction trace capability – many Cortex M MCUs have ETM or MTB
  - Instrument program: add instructions for visibility
    - Send out trace information (e.g. debug signals) to view with oscilloscope or logic analyzer
    - Will do this in lab

Big Picture of Real-Time Systems

Concepts | *Basic Ideas* → *Details and Improvements*

- **Task Model**
  - When does task run?
    - Periodically
    - Everything else
  - How long task takes: Execution time analysis
    - Analytical – safe
    - Experimental – unsafe
- **Scheduling**
  - Preemption vs. run-to-completion
  - Priorities
    - Fixed
    - Dynamic
  - Overheads
    - Hardware: Interrupts
    - OS: Context Switching, IPC
- **Response Time Analysis**
  - Basic methods
  - Blocking
  - Handling priority inversion
- **Deadlines**
  - Using response time analysis
  - Optimal scheduling approaches
  - Schedulability tests
  - Comparing impact of preemption and priority

Basic Design Choices for Scheduler

- Break up design space into categories based on choices in scheduling approach
  - Can **tasks preempt other tasks**?
    - Enabled ISRs can always preempt tasks
  - Is **task priority fixed** or **dynamic**?
    - Does a task have a single priority, or can the priority change (e.g. based on time until deadline)
- For a given category we want to know…
  - How to we get the best priority assignment?
  - How much of the processor's time does it let us use?
  - What is the worst-case response time for each task?

	Preemptive	**Non-Preemptive**
Fixed Priority		
Dynamic Priority		

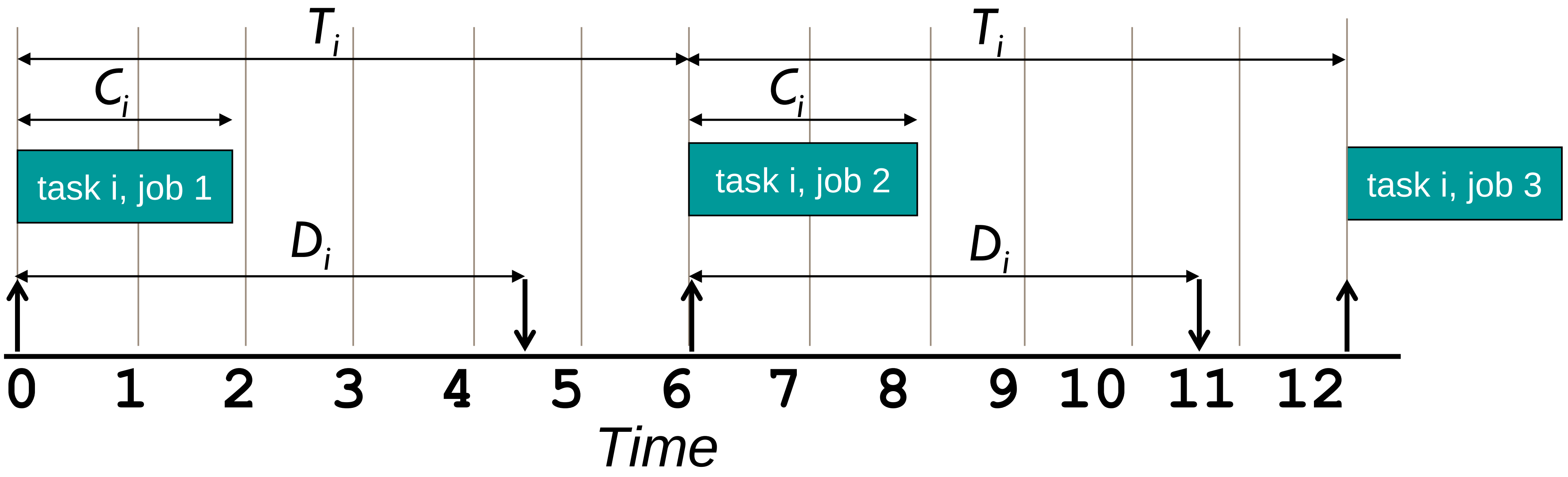
Scheduling – Selecting a Ready Task to Run

- What if multiple tasks are ready to run?
  - Non-prioritized
    - Give each ready task a chance to run (round robin, taking turns).
    - A task's responsiveness depends on the run time of ***all other tasks in the system.***
    - *Timing is unstable and fragile.*
  - Prioritized:
    - Some ready tasks have precedence over others. Scheduler runs them preferentially.
    - A task's responsiveness becomes (more) ***independent of lower priority tasks***.
    - *Timing is much more stable.*

- Assign priorities to urgent tasks to improve their responsiveness
  - Implicit: OK to delay less urgent tasks
  - We'll see different approaches to *Priority Assignment*
  - Tasks may have deadlines
  - Scheduler may or may not know about deadlines

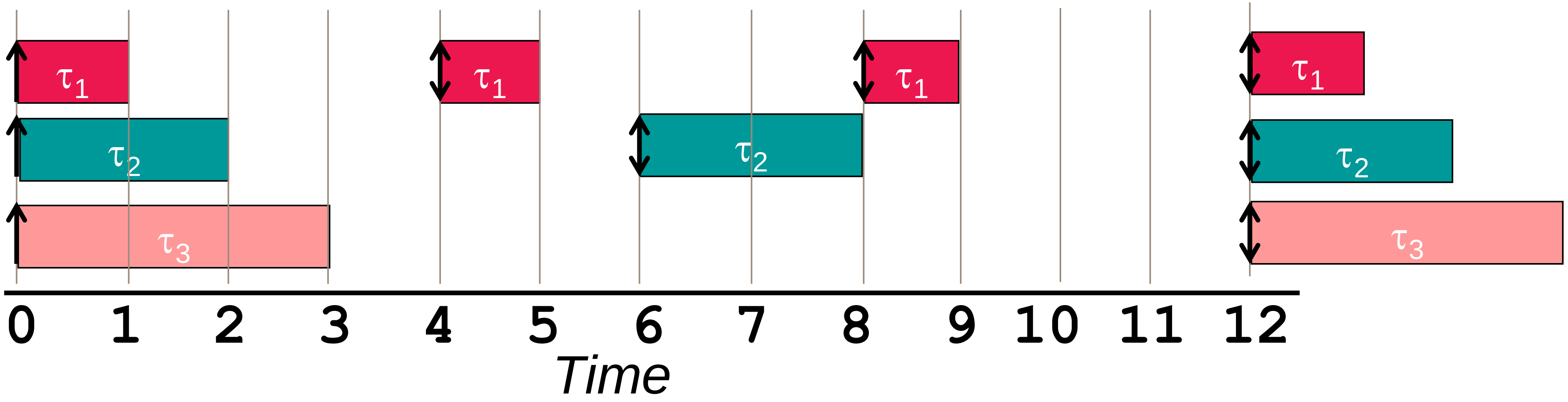
PERIODIC TASK MODEL

Periodic Task Model of Computational Requirements



- Periodic Task Model describes characteristics for each task τ_i
  - Job = a specific instance of that task running
  - Task releases job so scheduler can run it
- ↑ A periodic task *i* releases a job every T_i time units
- ↓ Job may have an absolute deadline D_i after its release
- Job takes a constant time C_i to execute
- Simplifying assumptions include
  - no time needed for scheduler, task switching, ISR response/return

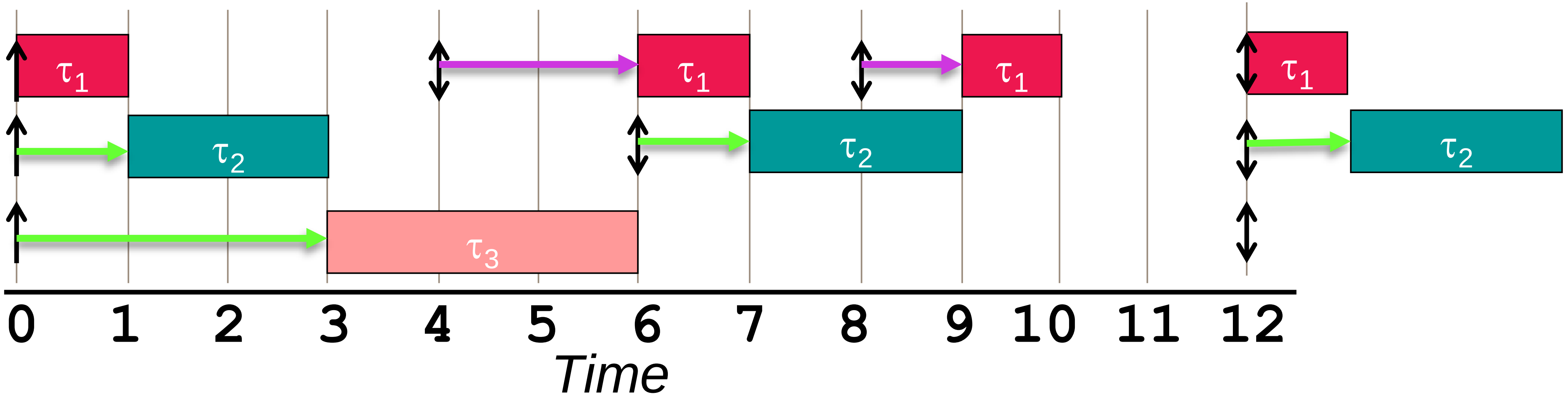
Example Workload: What We Ask For



- Set of tasks with real-time requirements
- What gets executed when?
  - Depends on scheduler and task priorities

Task	Exec. Time C_i	Period T_i	Deadline D_i
τ_1	1	4	4
τ_2	2	6	6
τ_3	3	12	12

Scheduled Workload: What We Get



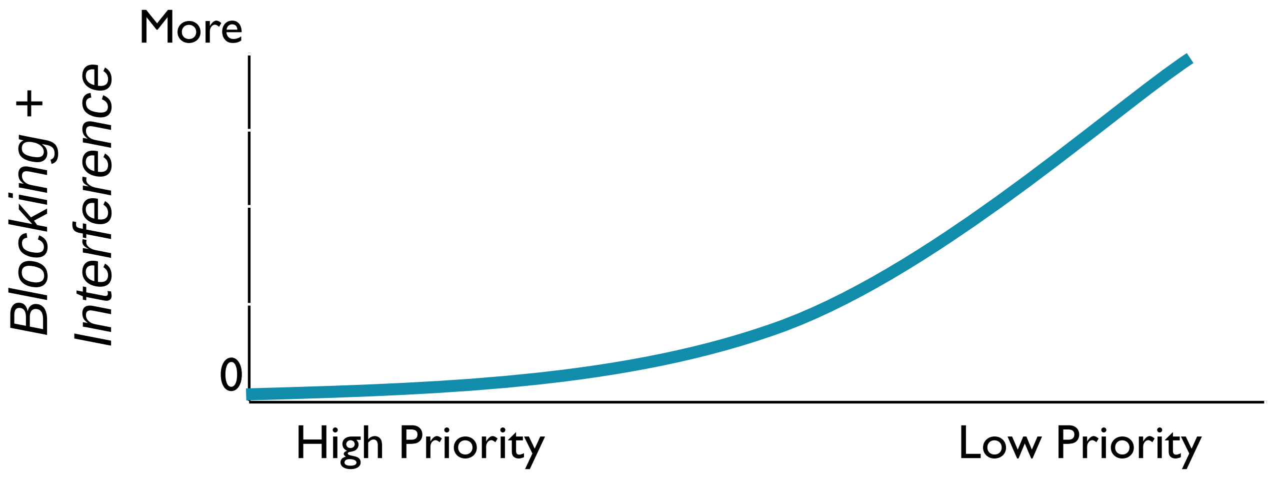
- Example: Scheduler and task fixed priorities
  - Assign priorities as shown
  - Use a non-preemptive scheduler
- What can delay a task?
  - I: Interference caused by higher priority tasks
  - B: Blocking caused by lower priority tasks
- Response time = Computation + Blocking + Interference

Task	Exec. Time C_i	Period T_i	Deadline D_i	Priority
τ_1	1	4	4	*High*
τ_2	2	6	6	*Medium*
τ_3	3	12	12	*Low*

$$R_i = C_i + B_i + I_i$$

Graphically Evaluating Response Times with Different Schedulers

- Goal: Higher priority should result in less blocking and interference

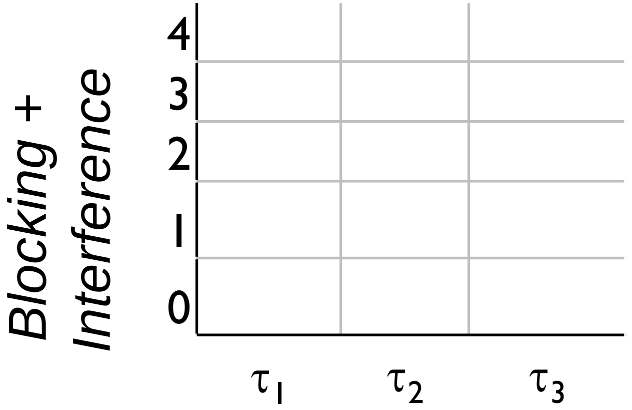


- Evaluate three schedulers
  - Non-preemptive
  - Non-preemptive
    - With τ_3 split into two-state FSM (C_i = 1.5 each), as is largest C_i
  - Preemptive
    - With original τ_3

Task	**Exec. Time C_i**	**Period T_i**	**Priority**
τ_1	1	4	*High*
τ_2	1	5	*Medium*
τ_3	3	7	*Low*

Non-Preemptive Scheduling

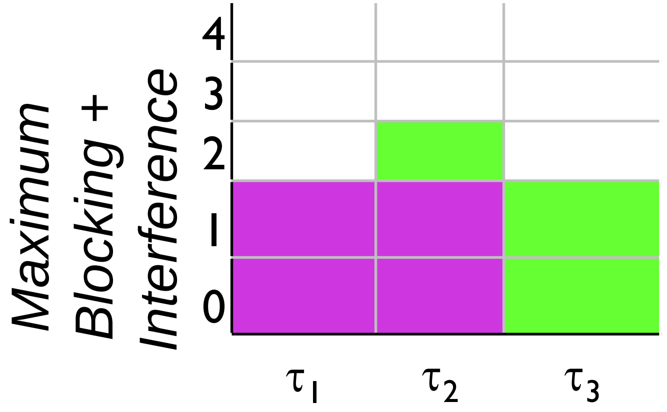
Task	Exec. Time C_i	Period T_i	Priority
τ_1	1	4	*High*
τ_2	1	5	*Medium*
τ_3	3	7	*Low*

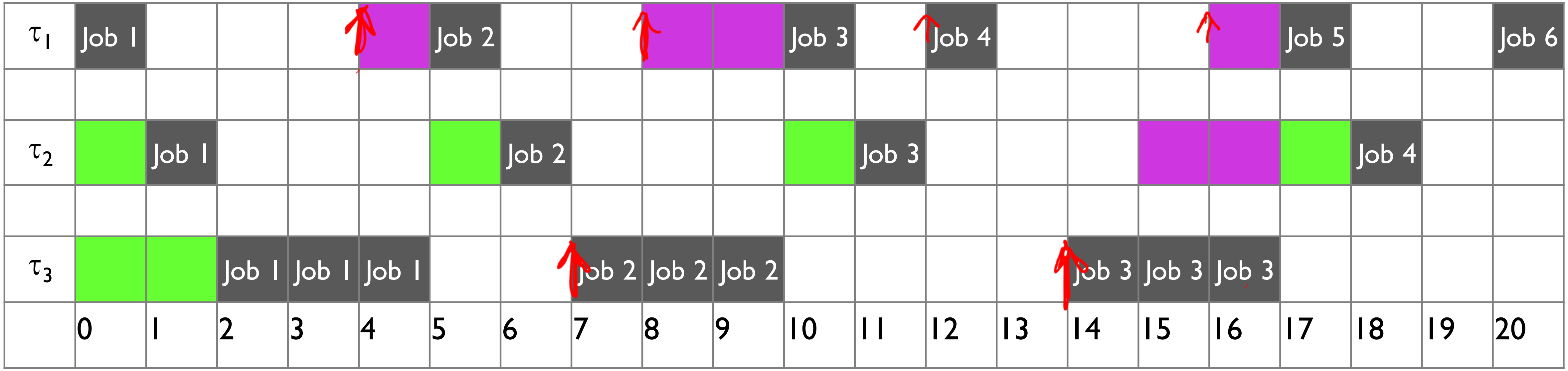


τ_1																					
τ_2																					
τ_3																					
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Non-Preemptive Scheduling

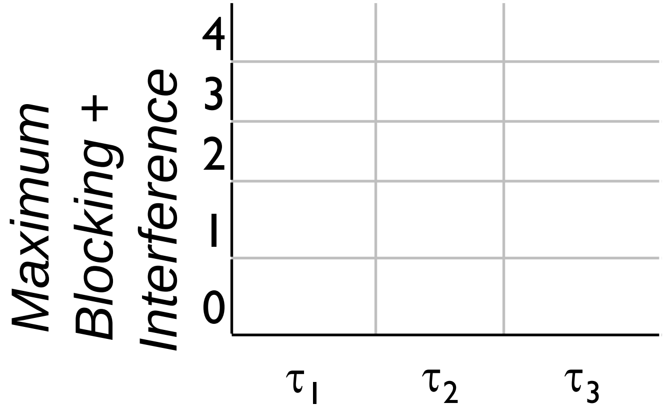
Task	Exec. Time C_i	Period T_i	Priority
τ_1	1	4	*High*
τ_2	1	5	*Medium*
τ_3	3	7	*Low*





Non-Preemptive Scheduling with FSM for τ_3

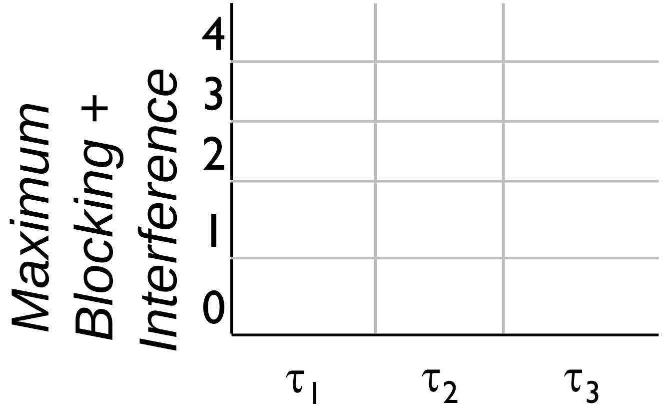
Task	**Exec. Time C_i**	**Period T_i**	**Priority**
τ_1	1	4	*High*
τ_2	1	5	*Medium*
τ_3	1.5	7	*Low*



τ_1																					
τ_2																					
τ_3																					
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Sidebar: Non-Preemptive Scheduling with FSM for τ_3 (with shorter period)

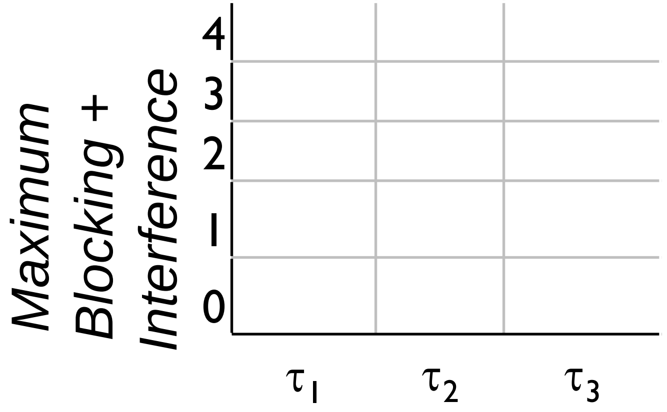
Task	Exec. Time C_i	Period T_i	Priority
τ_1	1	4	*High*
τ_2	1	5	*Medium*
τ_3	1.5	3.5	*Low*



τ_1																					
τ_2																					
τ_3																					
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Preemptive Scheduling

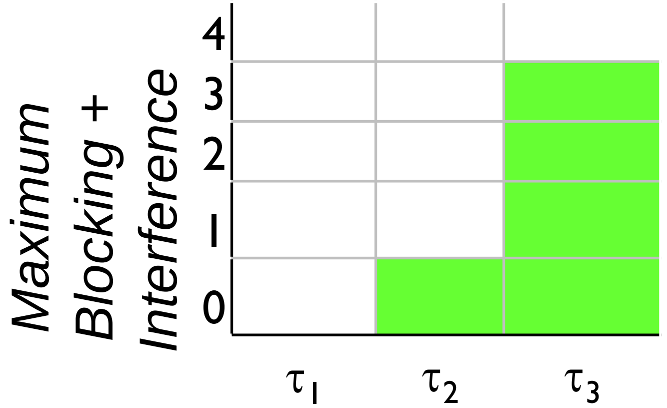
Task	Exec. Time C_i	Period T_i	Priority
τ_1	1	4	*High*
τ_2	1	5	*Medium*
τ_3	3	7	*Low*

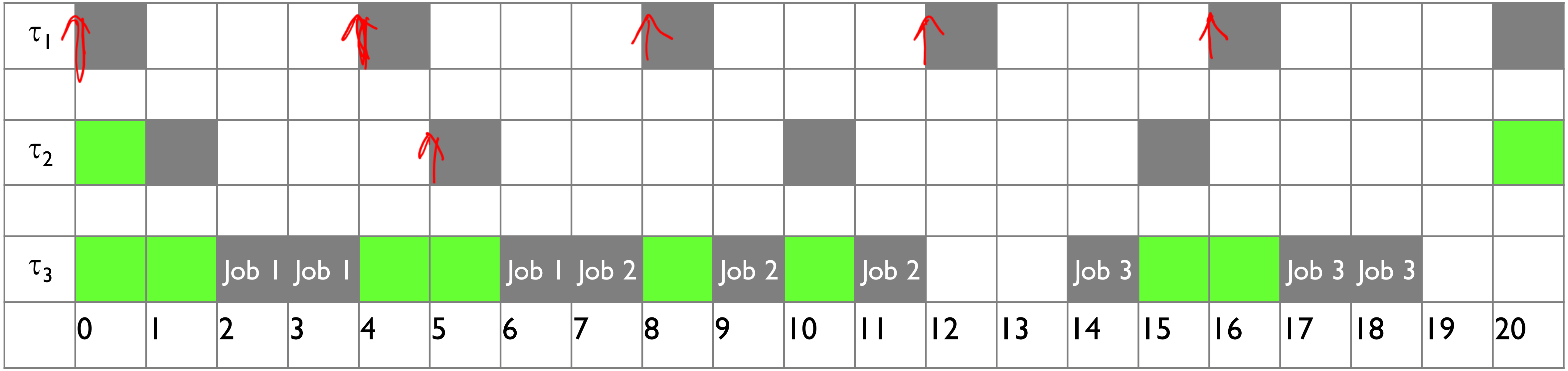


τ_1																					
τ_2																					
τ_3																					
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Preemptive Scheduling

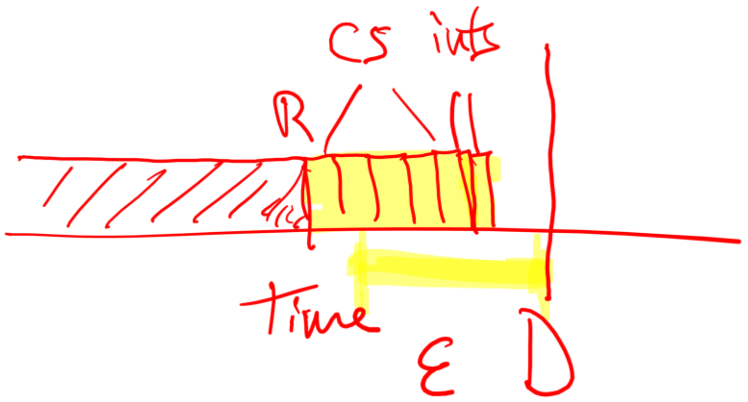
Task	Exec. Time C_i	Period T_i	Priority
τ_1	1	4	*High*
τ_2	1	5	*Medium*
τ_3	3	7	*Low*





Building and Using a Periodic Task Model

- Information per task and ISR – use spreadsheet
  - Period
  - Worst-case execution time,
  - Deadline (if present)
  - Critical sections (duration and communicating tasks)
    - Ignore because initially we assume all tasks are independent
- Don't forget overheads from OS and interrupt handling
  - For now, leave a margin of error **ε**. Stay away from the edge!
- Can now apply scheduling policy and assign priorities
  - Preemptive or non-preemptive?
  - Fixed or dynamic priority?
  - What priority assignment approach?



NUMERICAL RESPONSE TIME ANALYSIS

Response Time Analysis, Step 1

- How long could it take for task *i* to complete? What is its *response time* R_i?
- Initial estimate based on worst case:

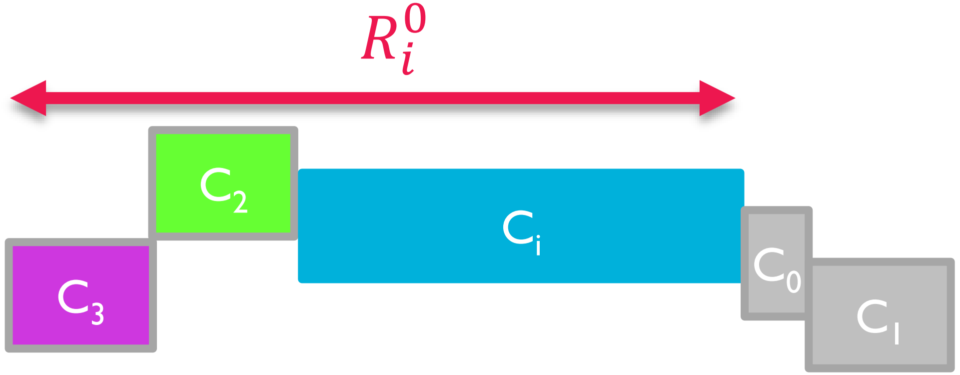
  R_i^0 = computation time for task *i* + computation time for other tasks.

  - Non-prioritized scheduling: Every other task can run once

```
while (1) {
  for (j=0; j<NUM_TASKS; j++) {
    if (Tasks[j].RP > 0) {
      Tasks[j].RP--;
      Tasks[j].Task();
} } }
```

$$R_i^0 = \boxed{C_i} + \boxed{\sum_{j\neq i} C_j}$$

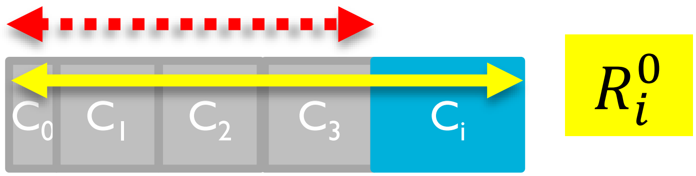
  - Prioritized scheduling: All higher-priority tasks (+ longest lower-priority task if non-preemptive) can run once



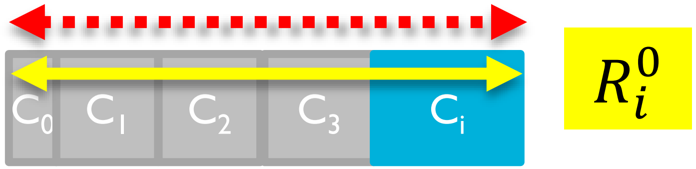
$$R_i^0 = \boxed{C_i} + \boxed{\max_{j\in lp(i)}(C_j)} + \boxed{\sum_{j\in hp(i)} C_j}$$

Additional Timing Interference

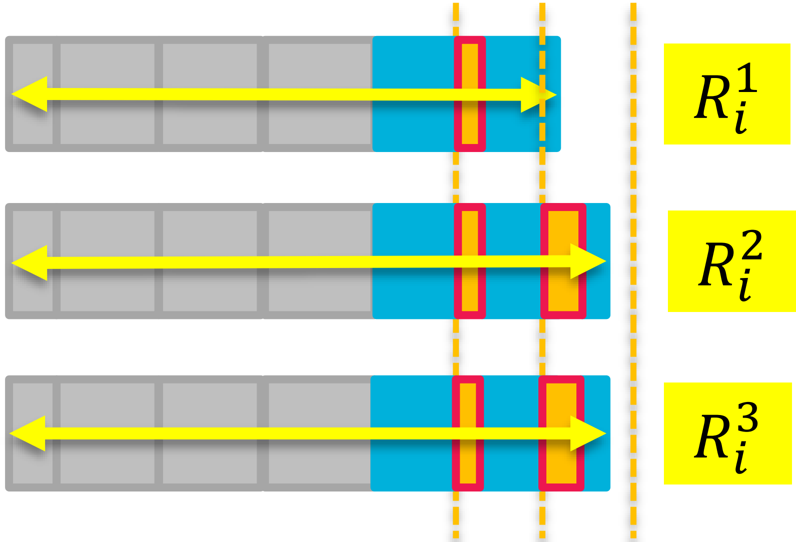
- Task *i* is vulnerable to delays from *new job releases* during vulnerable time
  - Non-preemptive: 0 to $R_i^n - C_i$ since task *i* can't be preempted after it starts



  - Preemptive: 0 to R_i^n since higher-priority task can preempt task *i*



- Consider new releases to update completion time estimate R_i^{n+1}
- Repeat until no new releases, or any deadline (if present) is missed



How Many T_i Releases Possible During Vulnerable Time?

- Initial estimate was one release, so task's time is one job: $1*C_j$
- Remaining estimates must consider all job releases possible during vulnerable time: Ceiling(vulnerable time / T_j)*C_j

$$R_i^0 = \boxed{C_i} + \boxed{\sum_{j \neq i} C_j}$$

$$R_i^0 = \boxed{C_i} + \boxed{\max_{j \in lp(i)}(C_j)} + \boxed{\sum_{j \in hp(i)} C_j}$$

	Non-prioritized	Prioritized
Non-preemptive	$\sum_{j \neq i} \left\lceil \frac{R_i^n - C_i}{T_j} \right\rceil C_j$	$\sum_{j \in hp(i)} \left\lceil \frac{R_i^n - C_i}{T_j} \right\rceil C_j$
Preemptive	$\sum_{j \neq i} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j$	$\sum_{j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j$

Response Time: Indep. Tasks with Task Preemption + Prioritization

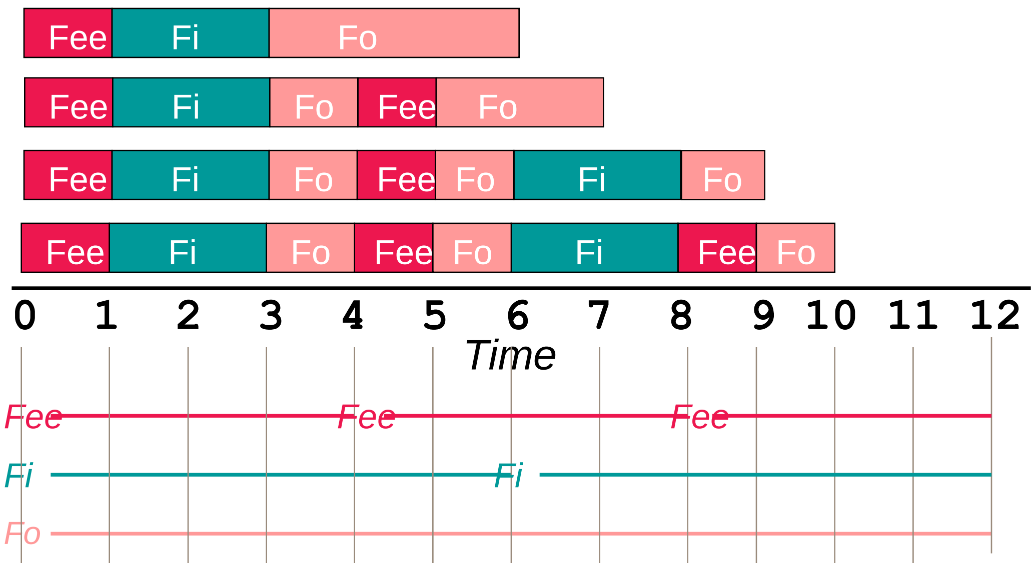
- Preemption …
  - Eliminates blocking of task *i* by lower-priority **independent** tasks.
  - Allows higher-priority tasks to preempt task *i*

$$R_i^{n+1} = \boxed{C_i} + \boxed{\sum_{j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j}$$

Processor Activity for Independent Tasks with Task Preemption

Since I started numbering tasks at 1 in this example, we start j at 1 here

j	Task	Exec. Time C	Period T	Priority
1	Fee	1	4	High
2	Fi	2	6	Medium
3	Fo	3	12	Low



$$R_3^0 = 3 + \sum_{j=1}^{i-1} C_j = 3 + 1 + 2 = 6$$

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

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

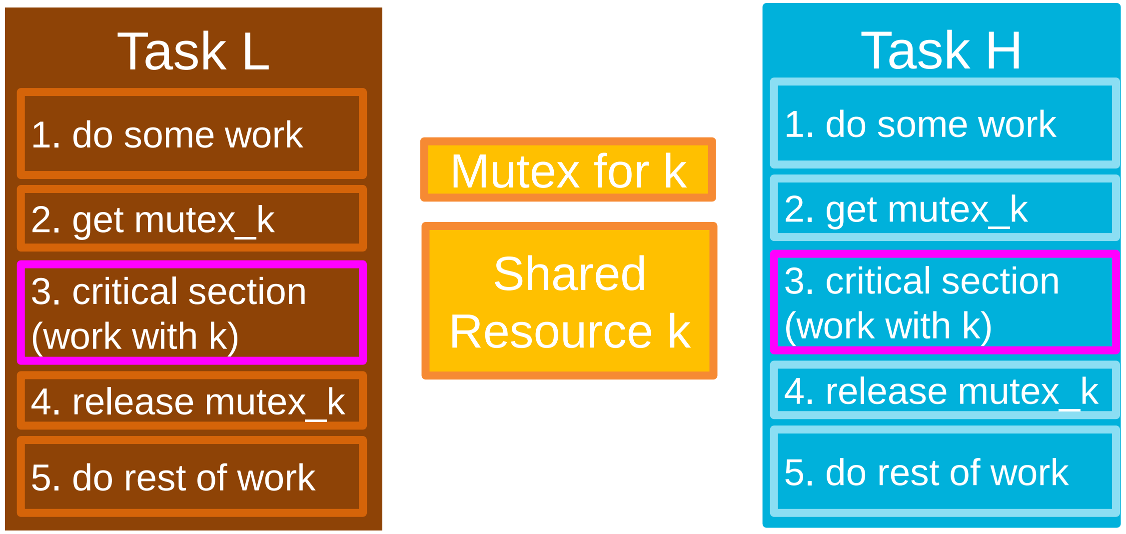
$$R_3^3 = 3 + \sum_{j=1}^{i-1} \left\lceil \frac{9}{T_j} \right\rceil C_j = 3 + \left\lceil \frac{9}{4} \right\rceil * 1 + \left\lceil \frac{9}{6} \right\rceil * 2 = 3 + 3 + 4 = 10$$

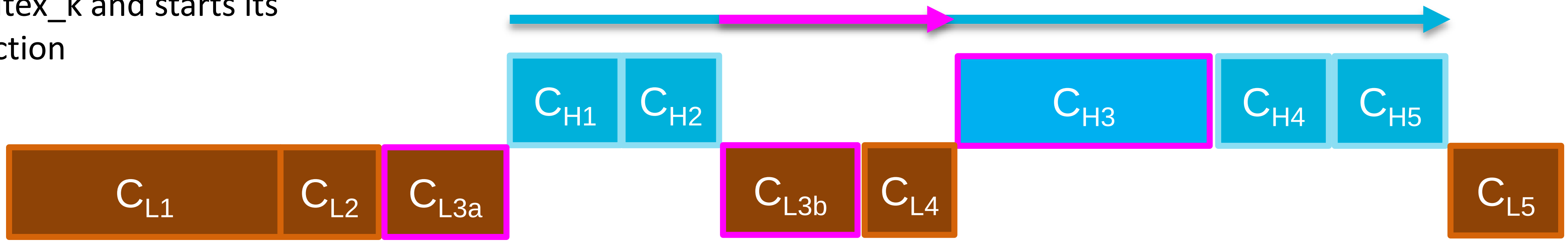
$$R_3^4 = 3 + \sum_{j=1}^{i-1} \left\lceil \frac{10}{T_j} \right\rceil C_j = 3 + \left\lceil \frac{10}{4} \right\rceil * 1 + \left\lceil \frac{10}{6} \right\rceil * 2 = 3 + 3 + 4 = 10$$

Iterate until R_3 stops changing

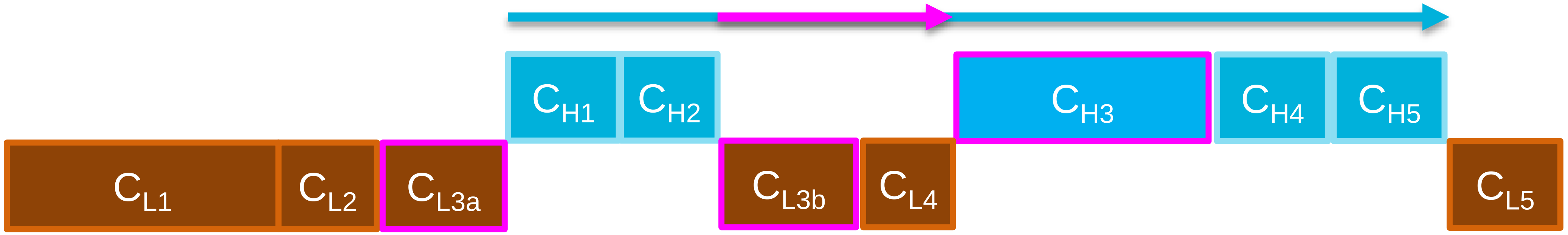
Response Time: Dependent Tasks with Task Preempt. + Prioritization

- Tasks H, L share a resource *k*
  - Tasks may use resource *k,* but not concurrently. Must take turns using mutex.
  - Code between getting, releasing mutex is a *critical section*
- Scenario where L can **block** H
  - L starts, gets mutex_k, starts executing critical section
  - H is released, preempting L
  - H runs but blocks when trying to get mutex_k
  - L resumes running and completes critical section
  - L releases mutex_k
  - H gets mutex_k and starts its critical section





Model for Blocking

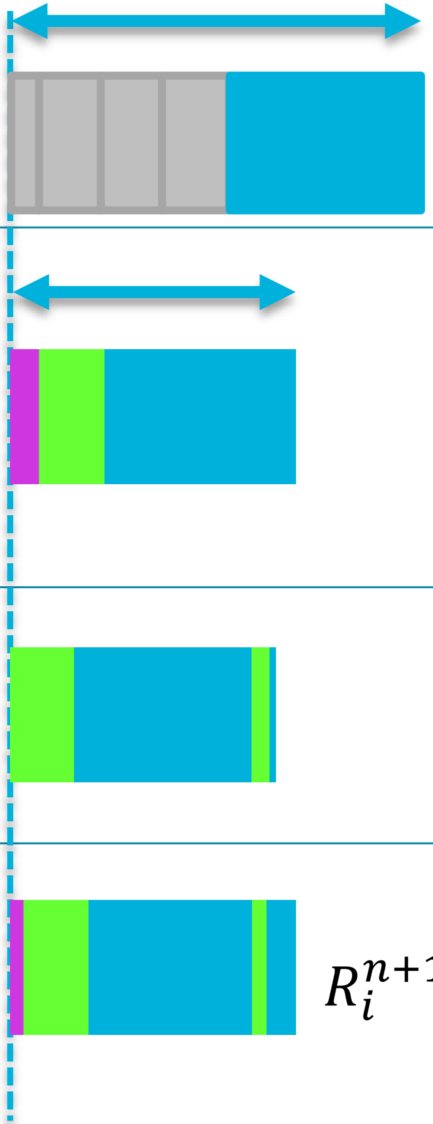


$$R_i^{n+1} = \boxed{C_i} + \boxed{\max_{\forall k}\left(usage(k,i)\ C_{j\ CritSect\ k}\right)} + \boxed{\sum_{j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j}$$

- Equation and terms
  - *i*: task being analyzed for response time
  - *j*: higher priority task
  - *k*: shared resource
  - $C_{j\ CritSect\ k}$: duration of task j's critical section for resource k
  - *usage(k,i):* 1 if task i uses resource k, else 0
- Next set of slides (Advanced Responsiveness)
  - Covers this case
  - Covers Priority Inversion: what happens if medium-priority task M gets caught here?

Summary of Response Time Equations

- Base case: no priority or preemption



$$R_i^{n+1} = \boxed{C_i} + \boxed{\sum_{j\neq i} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j}$$

- +Task prioritization
  - Blocking,
  - interference

$$R_i^{n+1} = \boxed{C_i} + \boxed{B_i(R_i^n)} + \boxed{I_i(R_i^n)}$$

$$R_i^{n+1} = \boxed{C_i} + \boxed{\max_{j\in lp(i)}(C_j)} + \boxed{\sum_{j\in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j}$$

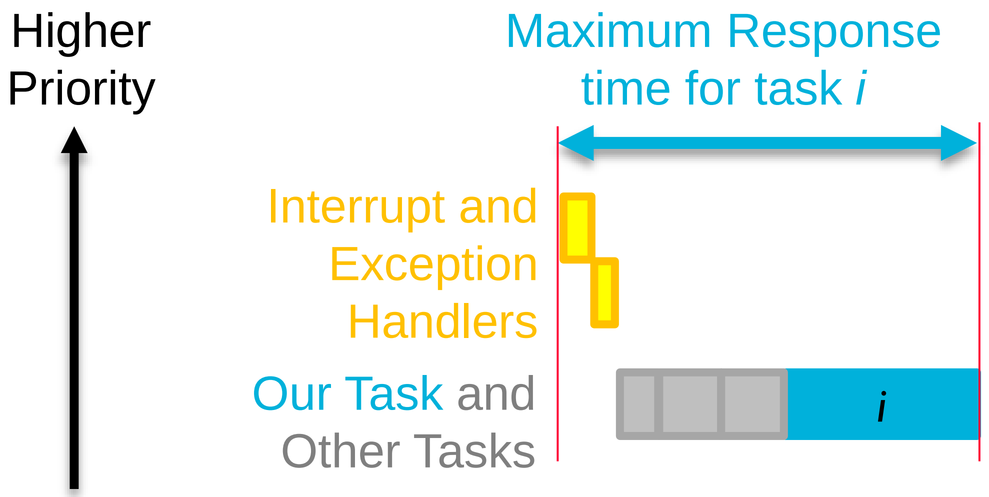
- +Task preemption
  - Independent tasks

$$R_i^{n+1} = \boxed{C_i} + \boxed{\sum_{j\in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j}$$

  - Dependent tasks

$$R_i^{n+1} = \boxed{C_i} + \boxed{\max_{\forall k}\left(usage(k,i)\ C_{j\ CritSect\ k}\right)} + \boxed{\sum_{j\in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j}$$

Task Response Time with Interrupts, no Task Priority or Preemption

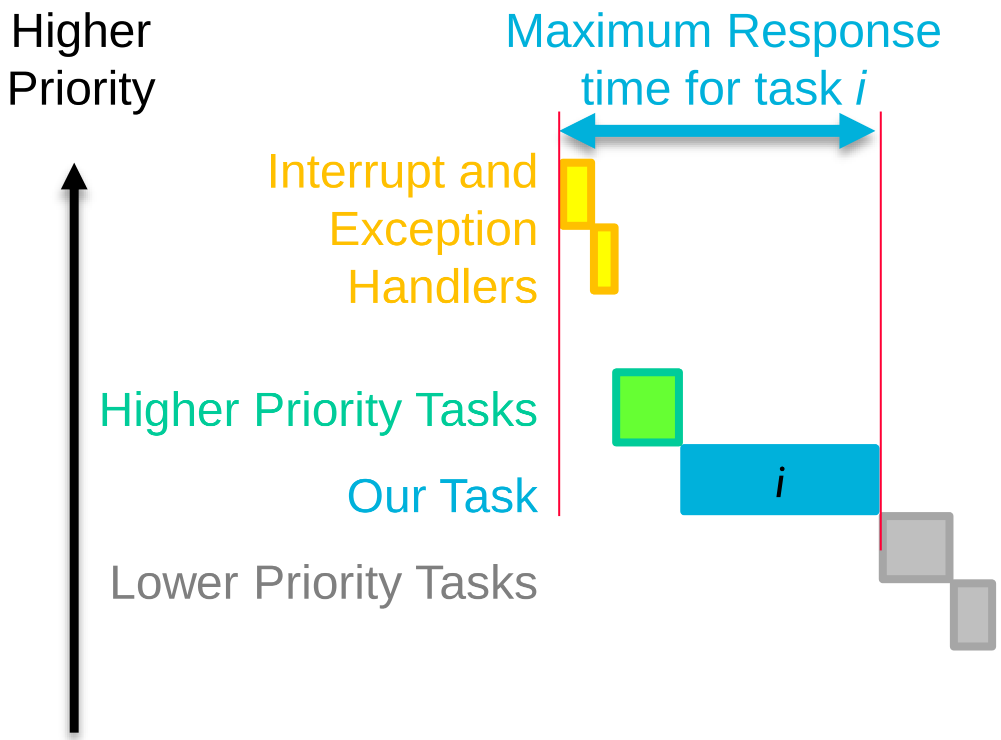


$$R_i^0 = \boxed{C_i} + \boxed{\sum_{j \in handlers} C_j} + \boxed{\sum_{j \in tasks \cap j \neq i} C_j}$$

$$R_i^{n+1} = \boxed{C_i} + \boxed{\sum_{j \in handlers} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j} + \boxed{\sum_{j \in tasks \cap j \neq i} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j}$$

- No task priority or task preemption
  - All other tasks can delay this task
- All handlers can preempt tasks

Task Response Time with Interrupts, Task Priority and Preemption

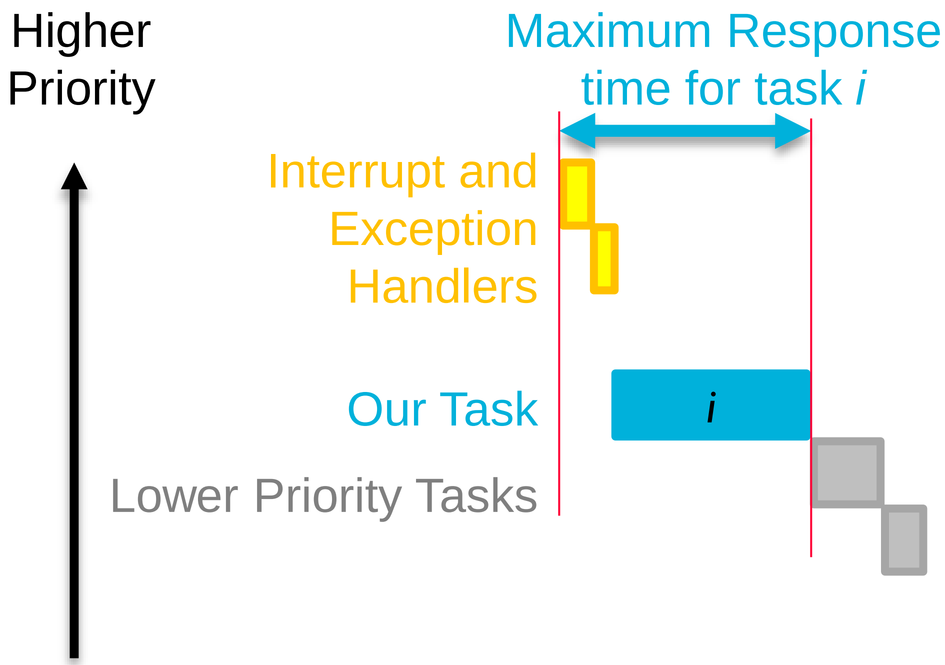


$$R_i^0 = \boxed{C_i} + \boxed{\sum_{j \in handlers} C_j} + \boxed{\sum_{j \in hp(i)} C_j}$$

$$R_i^{n+1} = \boxed{C_i} + \boxed{\sum_{j \in handlers} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j} + \boxed{\sum_{j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j}$$

- All handlers can preempt tasks
- Tasks have priority and preemption
  - Only higher-priority tasks can delay this task

Our Task is Highest Priority Task

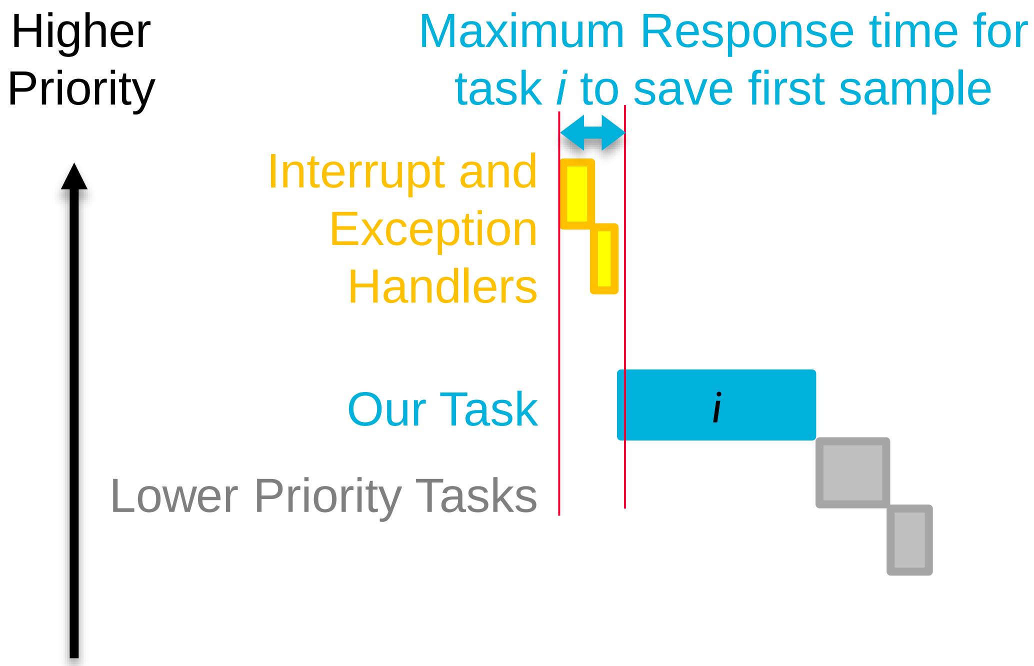


$$R_i^0 = \boxed{C_i} + \boxed{\sum_{j \in handlers} C_j}$$

$$R_i^{n+1} = \boxed{C_i} + \boxed{\sum_{j \in handlers} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j}$$

- All handlers can preempt tasks
- Tasks have priority and preemption
  - Only higher-priority tasks can delay this task

Our Task is Highest Priority Task, First Sample



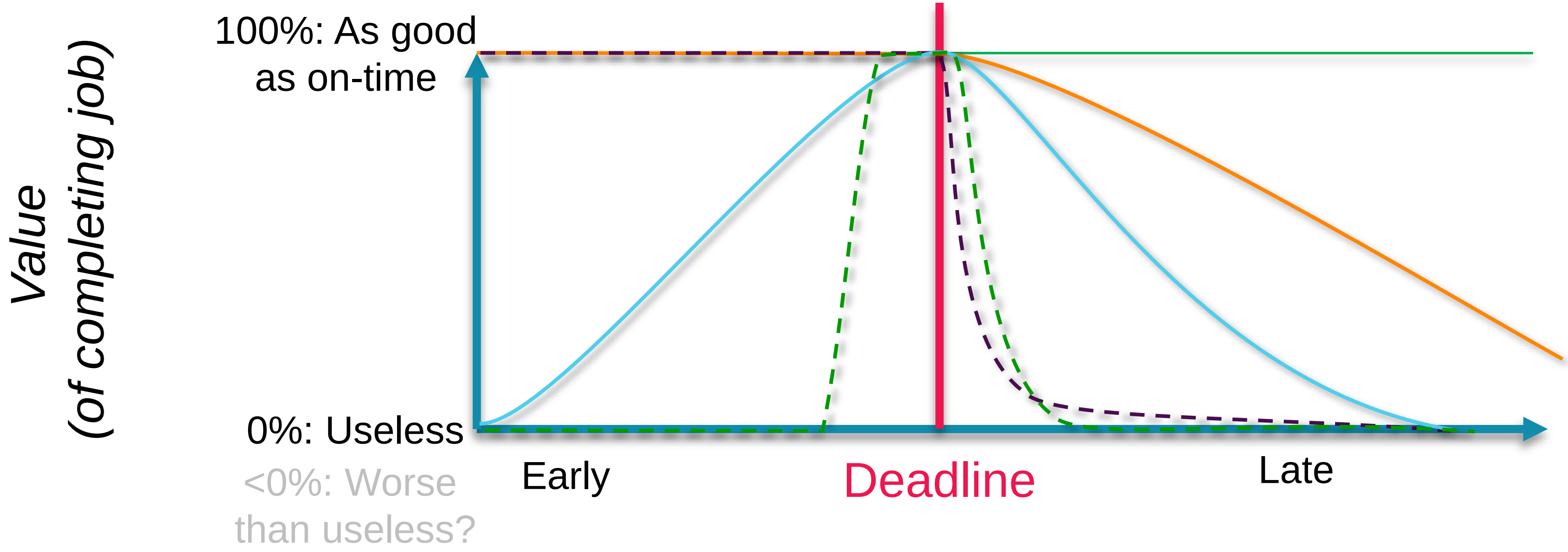
$$R_i^0 = \boxed{\frac{C_i}{512}} + \boxed{\sum_{j \in handlers} C_j}$$

$$R_i^{n+1} = \boxed{\frac{C_i}{512}} + \boxed{\sum_{j \in handlers} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j}$$

- All handlers can preempt tasks
- Tasks have priority and preemption
  - Only higher-priority tasks can delay this task

DEADLINES, PRIORITY ASSIGNMENT AND SCHEDULABILITY TESTS

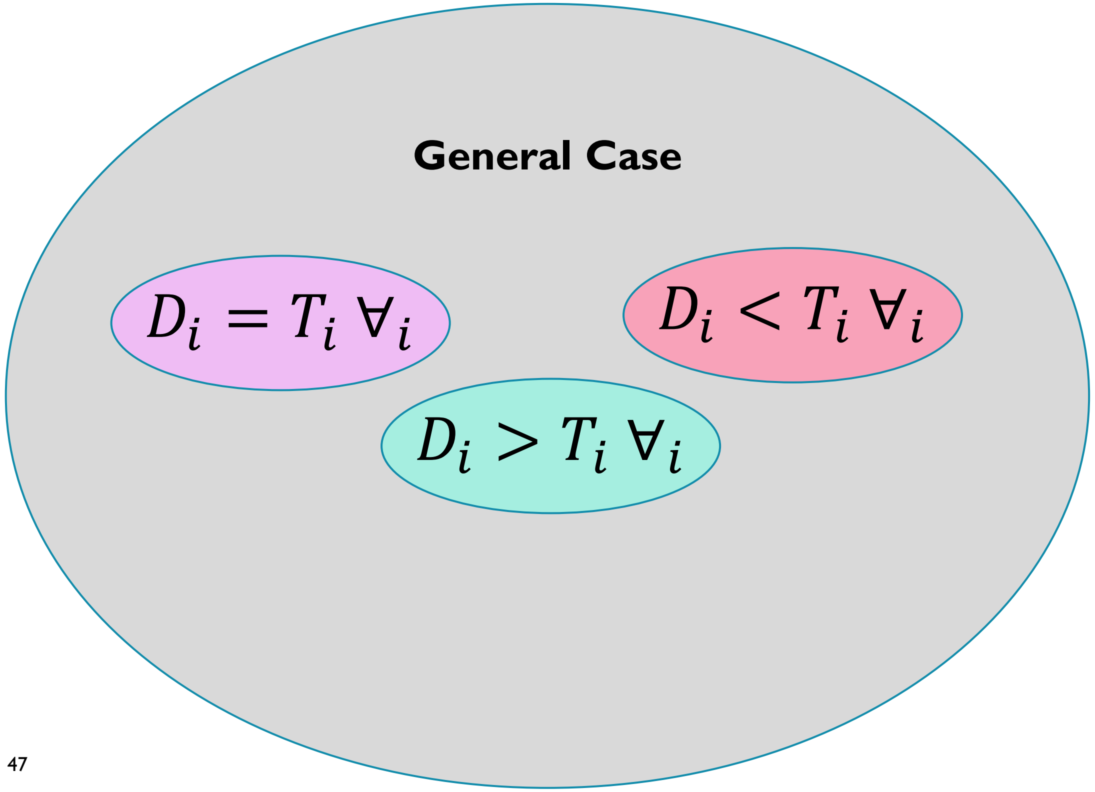
Deadlines



- Real-life activities
  - Juggling, cooking, catching the bus/airplane, paying bills, watering the plans, submitting a class project
- Embedded systems activities
  - Reading touchscreen
  - Displaying JPEG on LCD
  - Measuring output in switch-mode power converter control – should be synchronous (locked to phase)
- Types of deadlines
  - Hard: Critical to complete job by deadline
  - Soft: OK to miss by a little bit, but value decreases with increased lateness
  - M out of N: Must complete at least M out of N successive jobs by deadline (e.g. video frame update)
  - Others too…

Design Space for Workload with Deadlines and Scheduler

- Now consider task deadlines
- Break up design space further
  - Does any relationship between deadline $\boldsymbol{D_i}$ vs. period $\boldsymbol{T_i}$ hold true for **all tasks** ($\forall_i$)?



	Preemptive		Non-Preemptive	
	Fixed Priority	Dynamic Priority	Fixed Priority	Dynamic Priority
General Case				
$D_i < T_i$				
$D_i > T_i$				
$D_i = T_i$				

Questions

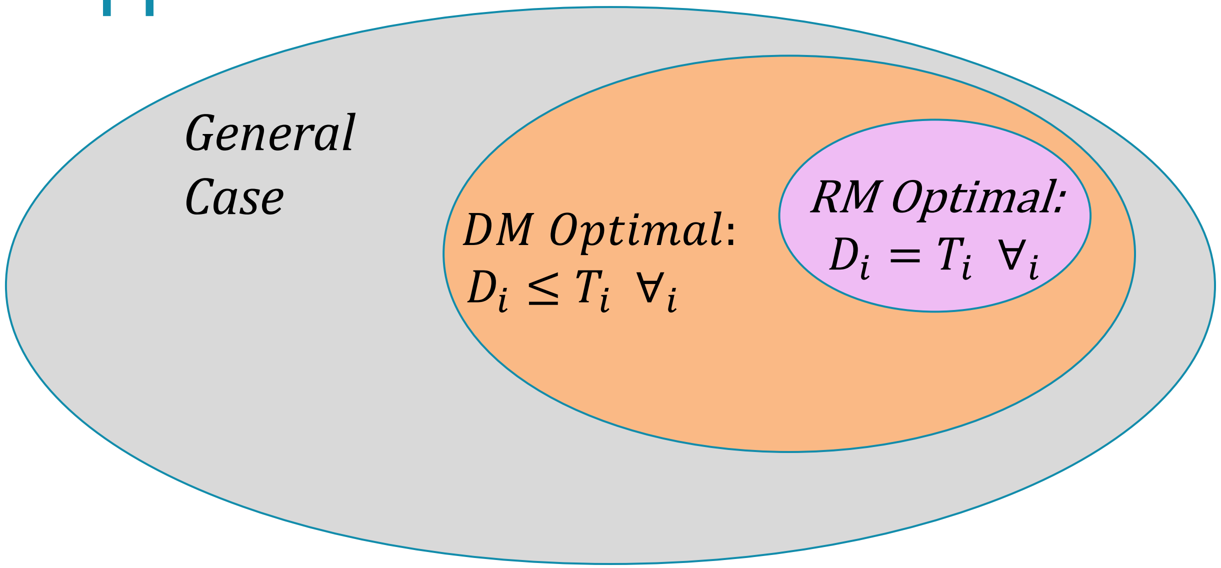
- For each category, we want to know…
  - What is the **optimal priority assignment?**
    - Use to assign priorities to tasks
  - Can we **calculate the exact worst-case response time for each task?**
    - Good for design analysis, including timing margins
    - Can determine schedulability – prove that deadlines can never be missed
  - Is there an easy **utilization-based schedulability test?**
    - Utilization U = fraction of CPU time used by tasks
    - Will a given priority assignment always create a schedule which misses no deadlines?

	Preemptive		**Non-Preemptive**	
	Fixed Priority	**Dynamic Priority**	**Fixed Priority**	**Dynamic Priority**
General Case				
$D_i < T_i$				
$D_i > T_i$				
$D_i = T_i$				

$$U = \sum_{i=1}^{m} \frac{C_i}{T_i} \leq U_{Magical\ Bound}$$

Common Fixed-Priority Assignment Approaches

- Audsley's priority assignment method
  - Is optimal for **all workloads** (general case)
    - No relationship needs to hold between all D_i and T_i
  - Complexity is $O(n^2)$. Number of steps depends on **square** of number of tasks
  - Audsley, N. C., (1991). "Optimal Priority Assignment And Feasibility Of Static Priority Tasks with Arbitrary Start Times", Technical Report YCS 164, Dept. Computer Science, University of York, UK, Dec. 1991



- Rate Monotonic (RM)
  - Priority based on release rate (1/period)
    - Higher release rate => higher priority
    - Complexity is $O(n)$
  - Optimal for workloads where deadline is end of period: $\boldsymbol{D_i = T_i} \ \forall_i$
  - Has easy utilization-based schedulability test
  - C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", Journal of the ACM 20(1), pp. 40-61 (1973)

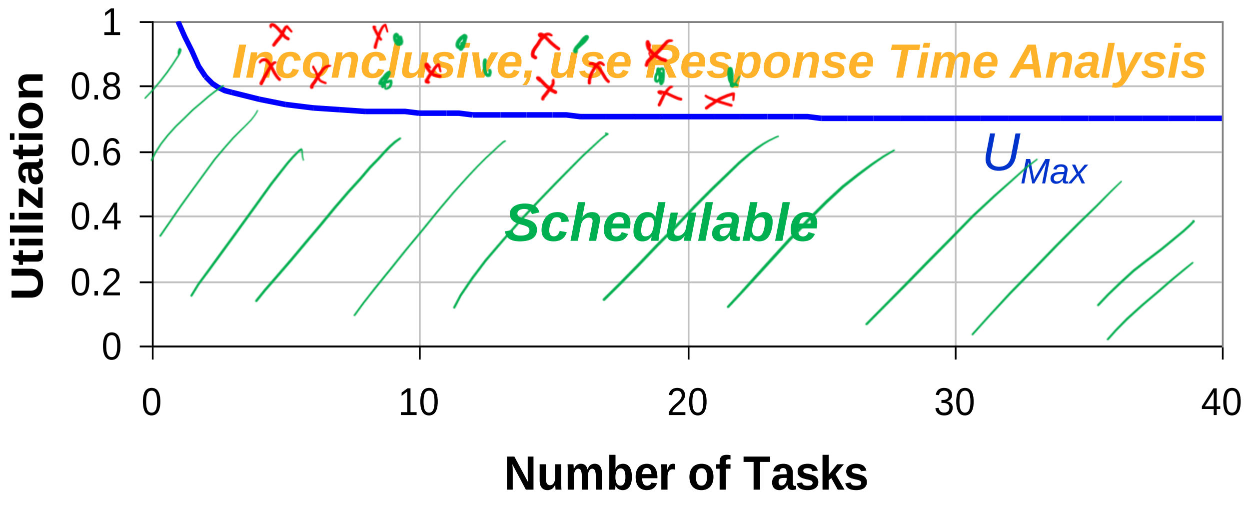
- Deadline Monotonic (DM)
  - Priority is based on time from release to deadline
    - Shorter deadline => higher priority
    - Complexity is $O(n)$
  - Optimal for workloads where deadline is no later than end of period: $\boldsymbol{D_i \leq T_i} \ \forall_i$
  - DM **includes** RM
  - Has easy utilization-based schedulability test
  - M. Joseph, P. Pandya, "Finding response times in a real-time system", BCS Comp. Jour., 29(5), pp. 390-395, 1986.

Utilization Bound for RM ($D_i = T_i \ \forall_i$)

- Calculate total utilization *U* for the system's *m* tasks
  - Fraction of time spent running tasks
- Calculate utilization bound U_{Max} for *m* tasks
  - Maximum value of *U* for which RM is guaranteed to work
  - Converges to → ln 2 ≈ 0.7
- Compare U with U_{Max}
  - $U < U_{Max}$: always schedulable with RMS
  - $U_{Max} < U < 1.0$: inconclusive
  - $U > 1.0$: Not schedulable
- Why is U_{Max} so small?
  - Conservative, simplifies math
- Can use for DM if $D_i < T_i$
  - Use D_i instead of T_i. Makes estimate of U larger
  - Gets more inconclusive (pessimistic) as D_i gets smaller than T_i

$$U = \sum_{i=1}^{m} \frac{C_i}{T_i}$$

$$U_{Max} = m\left(2^{1/m} - 1\right)$$



$$U = \sum_{i=1}^{m} \frac{C_i}{D_i}$$

Examples of Utilization Bound Values

m	U_max
1	1.000
2	0.828
3	0.780
4	0.757
5	0.743
6	0.735
7	0.729
8	0.724
9	0.721
10	0.718
15	0.709
20	0.705

Evaluating Schedulability with RM and UB

Task	Exec. Time C	Period T	Priority
τ_1	1	4	High
τ_2	2	6	Medium
τ_3	1	12	Low

$$U = \frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} = \frac{1}{4} + \frac{2}{6} + \frac{1}{12} = 0.6667$$

$$U_{Max} = m(2^{1/m} - 1) = 3(2^{1/3} - 1) = 0.780$$

Utilization Bound test shows task set is schedulable

Evaluating Schedulability with RM and UB

task	Exec. Time C	Period T	Priority
τ_1	1	4	High
τ_2	2	6	Medium
τ_3	3	12	Low

$$U = \frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} = \frac{1}{4} + \frac{2}{6} + \frac{3}{12} = 0.833$$

$$U_{Max} = m(2^{1/m} - 1) = 3(2^{1/3} - 1) = 0.780$$

Utilization Bound test is inconclusive!

Need a more accurate measurement: calculate worst-case response times of all tasks

More Examples of Using UB Test

Task	Exec. Time C	Period T	Total U	U_{Max}	Sched. w/ RMA?
τ_1	1	4			
τ_2	2	8			
τ_3	2	12			
τ_1, τ_2, τ_3			0.667	0.780	Yes
τ_{4A}	*1*	*15*	0.733	0.757	Yes
τ_{4B}	*2*	*10*	0.866	0.757	Maybe
τ_{4C}	*3*	*8*	1.041	0.757	No
τ_{4D}	*3*	*17*	0.843	0.757	Maybe
τ_{4E}	*5*	*20*	0.916	0.757	Maybe
τ_{4F}	*2*	*15*	0.8	0.757	Maybe

Harmonic Rate Monotonic

- Special case of RM
  - Every task period must evenly divide every longer task period
    - e.g. task periods of 10, 20, 40, 120
  - May be able to shorten task periods make them harmonic, but monitor increase in utilization
- Can still use utilization-based test (easy)
- Utilization bound $U_{Max\ HRM}$ is now "1" (really 1-ε)
- Example: Start with RM
  - 4 tasks, so $U_{Max\ RM}$ = 0.757
  - Utilization is 0.761 > $U_{Max\ RM}$, so schedulability test is inconclusive

		Original: RM		Adjusted: HRM	
Task	**Exec. Time**	**Period**	**Utilization**	**Period**	**Utilization**
t_1	4	13	0.308	10	0.400
t_2	8	35	0.229	30	0.267
t_3	7	60	0.117	60	0.117
t_4	12	111	0.108	60	0.200
		Total	**0.761**		**0.983**

- Apply HRM for this workload
  - Shorten periods to meet HRM requirement
  - $U_{Max\ HRM}$ = 1.000 regardless of task count
  - Utilization is 0.983 < 1.000, so workload is schedulable

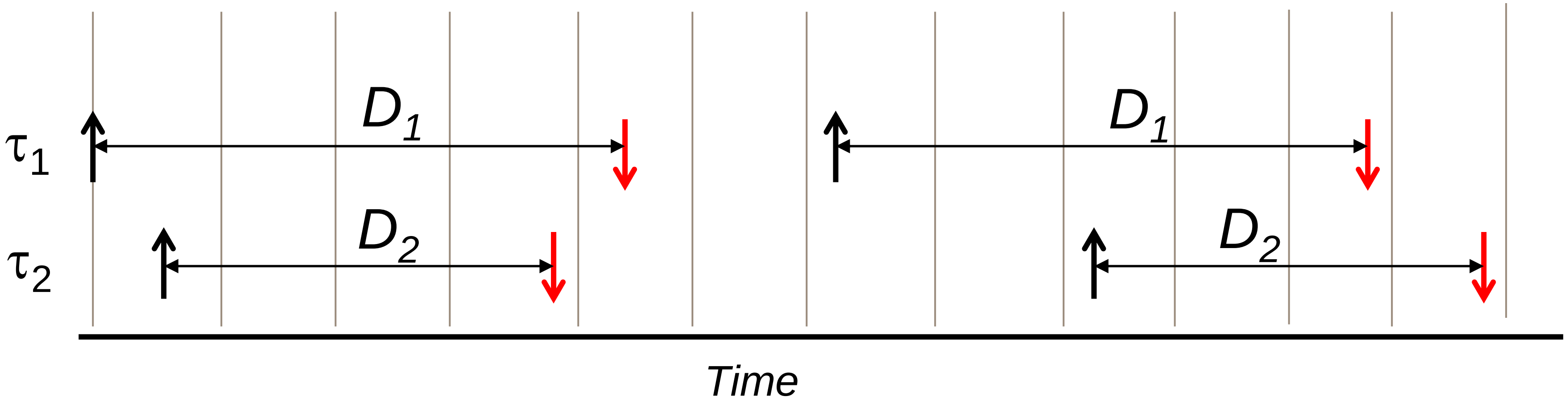
DYNAMIC PRIORITY PREEMPTIVE SYSTEMS

Dynamic Priority

- Earliest Deadline First (EDF)
  - Priority based on amount of time currently left until deadline
  - Closer deadline => higher priority
  - M. Dertouzos, "Control Robotics: the procedural control of physical processors", Proceedings of the IFIP congress, p 807-813, 1974.

- Least Laxity First (LLF)
  - Priority based on amount of laxity: absolute deadline minus current time minus remaining execution time)
  - A.K. Mok, "Fundamental Design Problems for the Hard Real-Time Environments", May 1983, MIT Ph.D. Dissertation

Earliest Deadline First

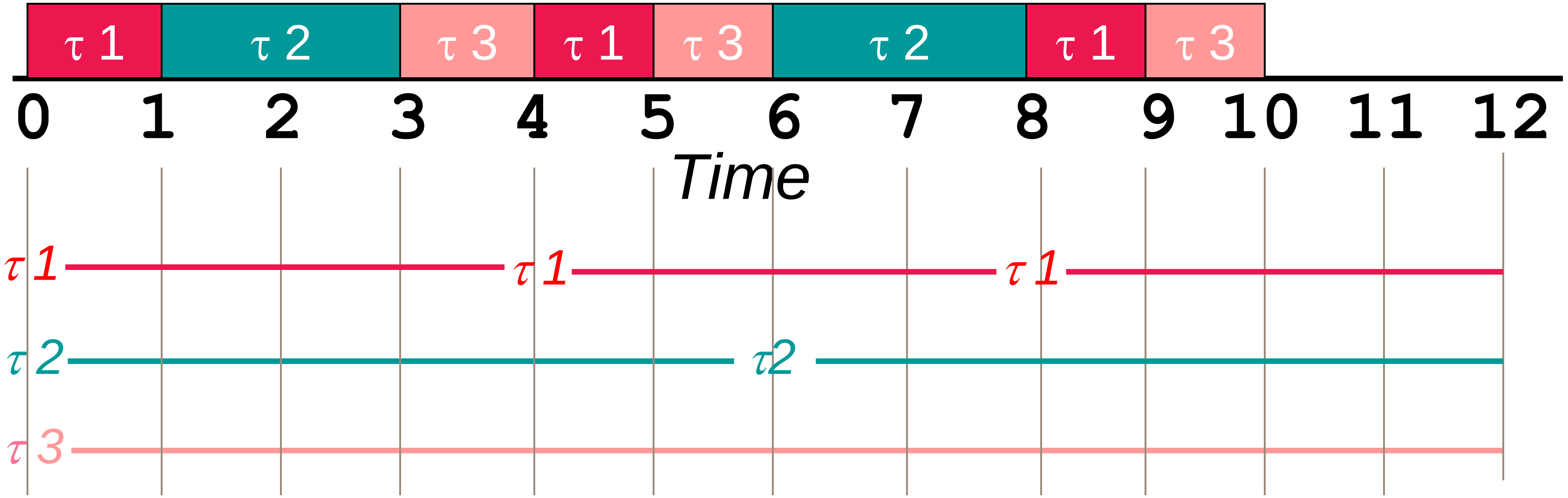


- Run the job with the earliest deadline first!
  - First releases: τ_2 runs before τ_1
  - Second releases: τ_1 runs before τ_2
- Implementation
  - Scheduler tracks each job's deadline, which depends on its release time
  - Jobs must be sorted by deadline
    - General case sorting complexity is $O(n^2)$
    - Optimizations for scheduler reduce complexity
      - Keep ready queue sorted, use bit masks for groups of tasks
- Utilization-based schedulability test depends on deadline constraints
  - $D_i = T_i$: Schedulable if utilization ≤ 1-ε
  - $D_i > T_i$: Schedulable if utilization ≤ 1-ε
  - $D_i < T_i$: Have to use a more complicated test

$$U = \sum_{i=1}^{m} \frac{C_i}{T_i}$$

EDF Processor Activity

Task	Execution Time C	Period T	Deadline D
τ1	1	4	4
τ2	2	6	6
τ3	3	12	12



Response Time Analysis for EDF

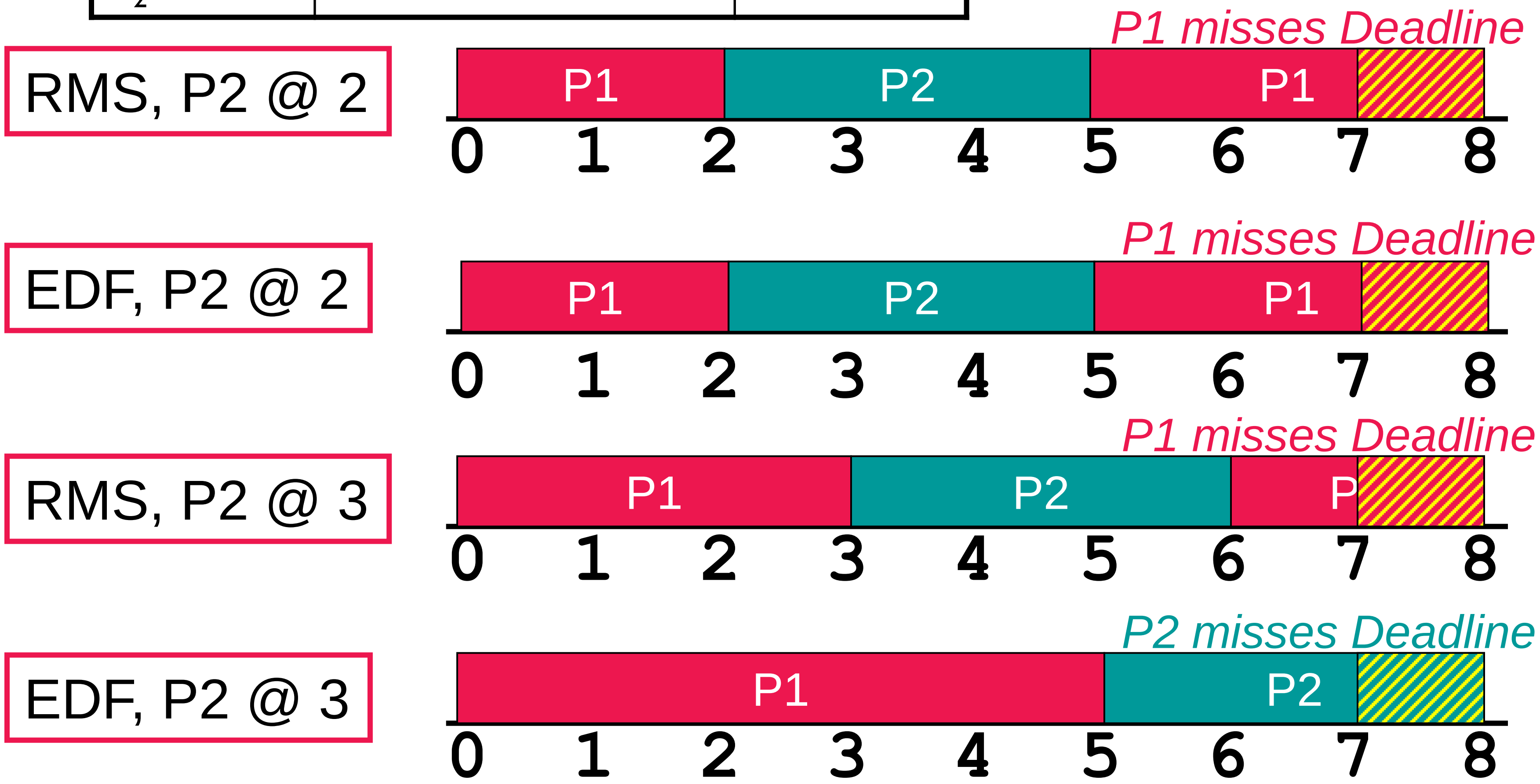
- "Non-trivial" to calculate for EDF and other dynamic priority schemes
  - Sum up impact from all possible higher priority tasks, but priority depends on how soon the deadlines are
  - This depends on when tasks are released
- Assume all tasks are periodic
  - Execution schedule will repeat every *hyper-period*
  - Hyper-period = least common multiple (LCM) of all task periods
    - LCM is smallest positive integer which is a multiple of all inputs
    - For tasks of periods 5, 20, 31 and 47, hyper-period is 29140
- Need to analyze the response time for ***each release*** within the hyper-period.
  - How many releases? At least 29140/5 + 29140/20 + 29140/31 + 29140/47 = 8845

System Performance During Transient Overload

- RM, DM – Each task has fixed priority. *So?*
  - This priority determines that tasks will be scheduled consistently
    - Task A will always preempt task B if needed
    - Task B will be forced to miss its deadline to help task A meet its deadline
- EDF – Each task has varying priority. *So?*
  - This priority depends upon when the task's deadline is, and hence when the task becomes ready to run (*release time*)
    - Task B may have higher priority than A depending on release times
    - To determine whether task A or B will miss its deadline we need to know their release times

Comparison of RM and EDF During Overload

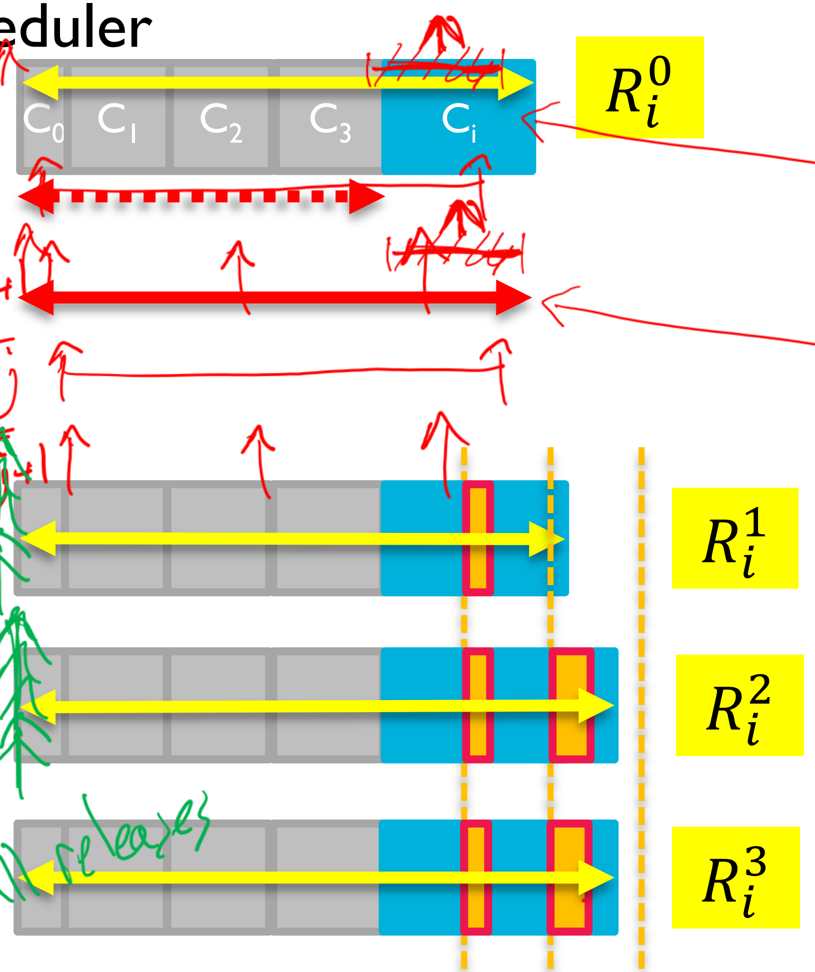
Task	Execution Time C	Period T
τ_1	5	7
τ_2	3	4



Old

Response Time Analysis, Steps 2, 3, 4, 5, 6 …

- Task i is vulnerable to *new job releases* during vulnerable time, which depends on scheduler
  - Non-preemptive: 0 to $R_i^n - C_i$ since task *i* can't be preempted after it starts
  - Preemptive: 0 to R_i^n since higher priority task can preempt task *i*
- Update completion time estimate R_i^{n+1}
- Repeat until no new releases, or deadline is missed



	Non-prioritized	Prioritized
Non-preemptive	$\sum_{j\neq i}\left\lceil \frac{R_i^n - C_i}{T_j}\right\rceil C_j$	$\sum_{j\in hp(i)}\left\lceil \frac{R_i^n - C_i}{T_j}\right\rceil C_j$
Preemptive	$\sum_{j\neq i}\left\lceil \frac{R_i^n}{T_j}\right\rceil C_j$	$\sum_{j\in hp(i)}\left\lceil \frac{R_i^n}{T_j}\right\rceil C_j$

# of jobs * C_j

WC: All releases