NuttX RTOS

NuttX Realtime Programming

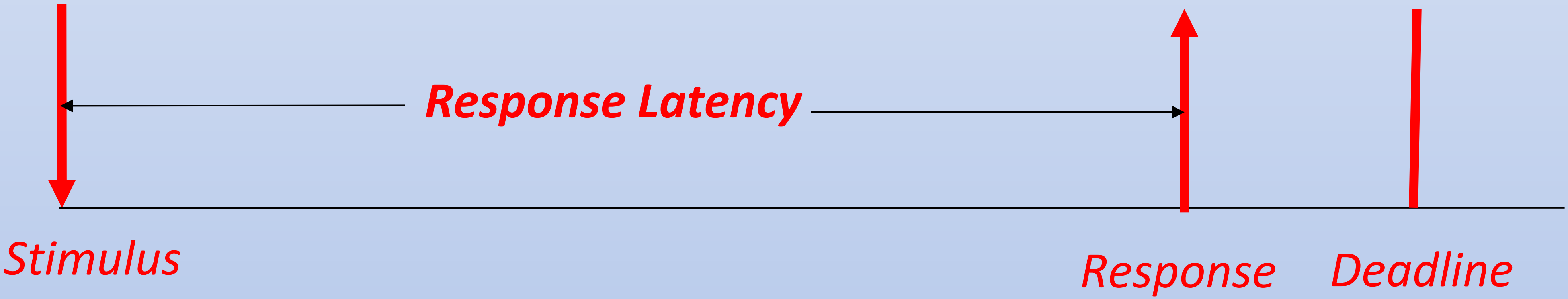
Gregory Nutt

Made in Costa Rica

Overview

- Interrupts
- Cooperative Scheduling
- Tasks
- Work Queues
- Realtime Schedulers

Real Time == Deterministic

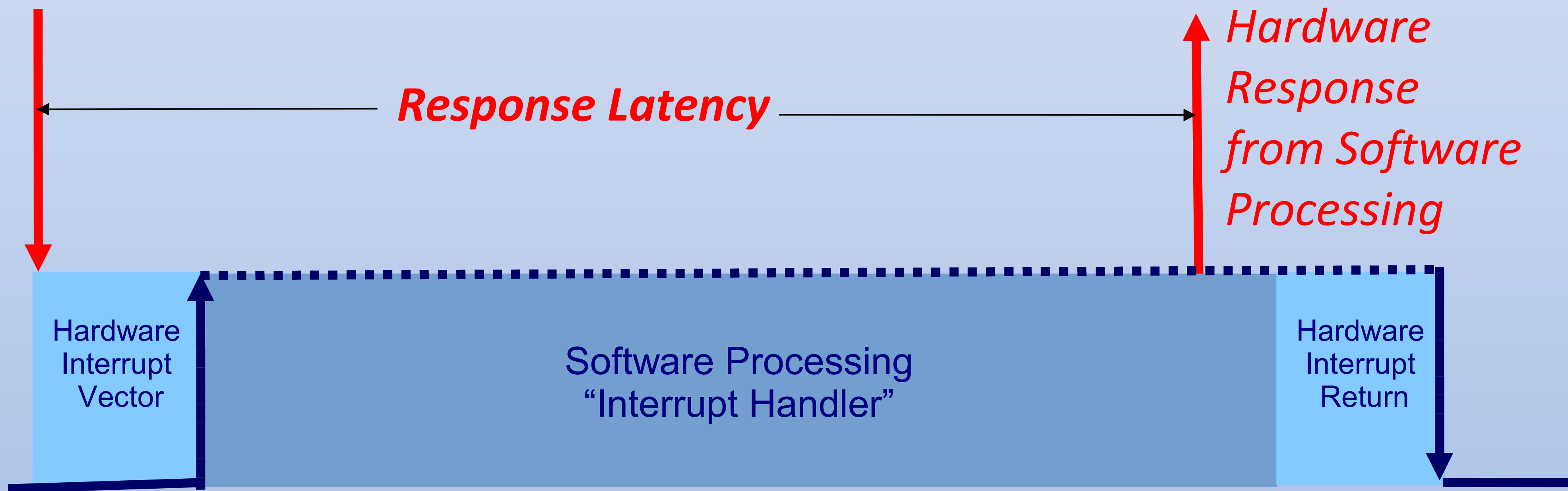


Real time does not mean "fast"

Real time systems have ***Deadlines***

Bare Metal / No-OS
Single Interrupt

External Stimulus generates Hardware Interrupt Request (IRQ)

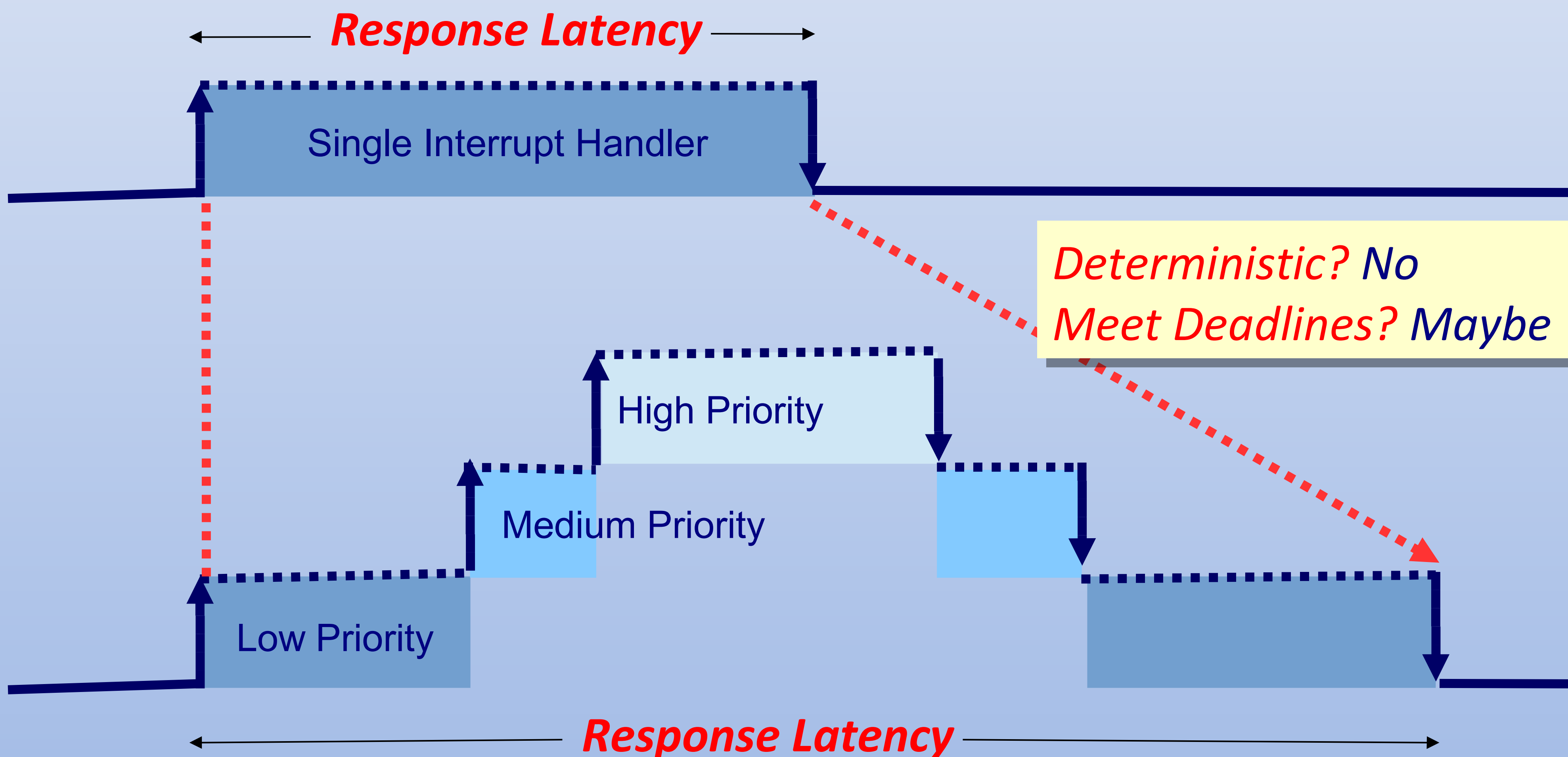


Hardware Interrupt Processing may be delayed if interrupts disabled.

Very Simple!

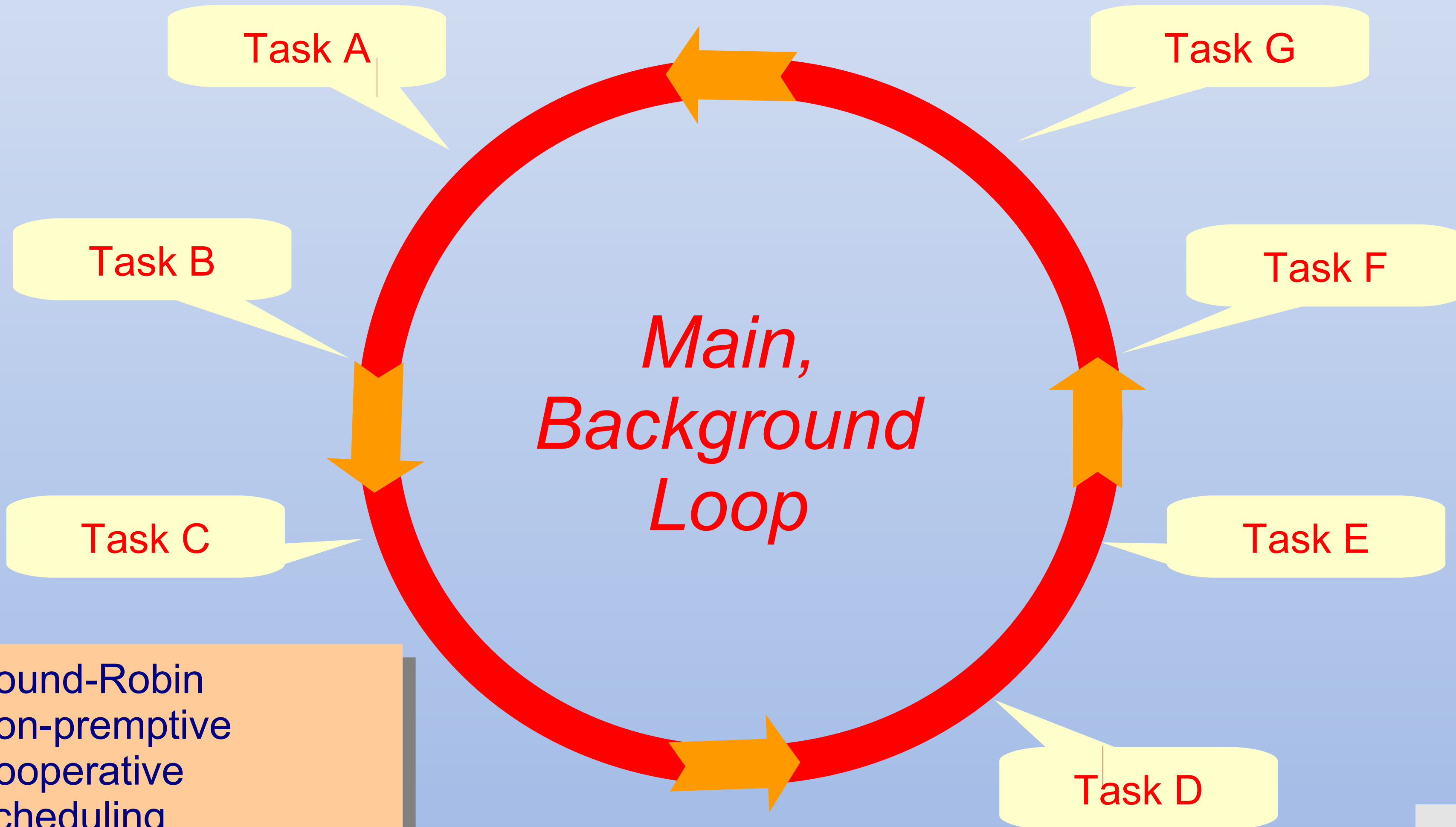
Deterministic if interrupts *never* disabled.

Bare Metal / No-OS
Nested Interrupts



No OS way: Extensive interrupt processing, prioritized interrupts and, maybe, a *main loop*.

Bare Metal / No-OS
Cooperative Scheduling

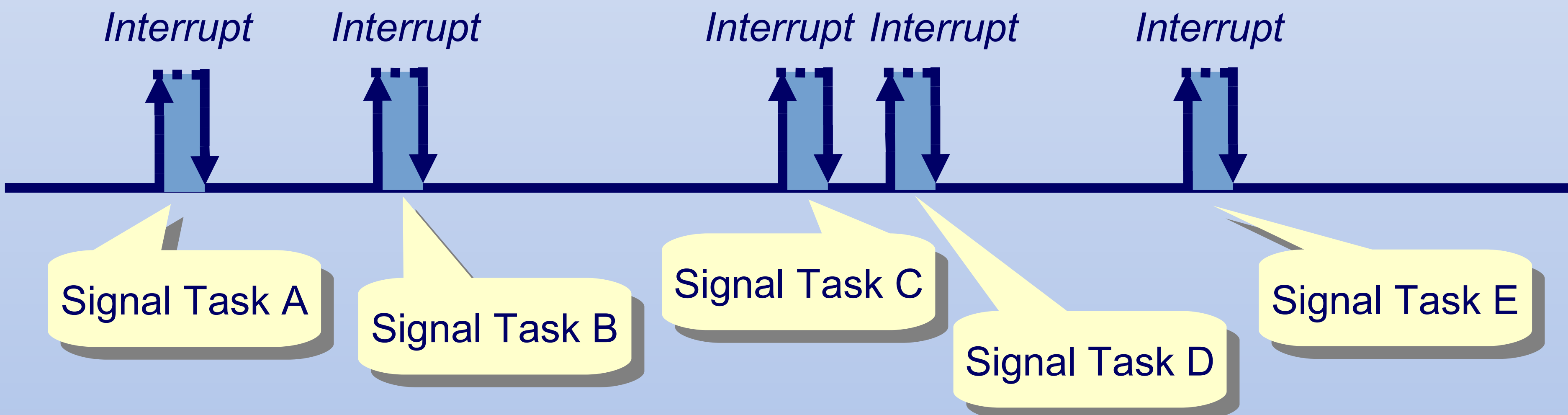


- Round-Robin
- Non-premptive
- Cooperative Scheduling
- State machines
- *ad hoc* strategies

Non-deterministic!

RTOS Interrupts

No OS way: Extensive interrupt processing, prioritized interrupts and, maybe, a *main loop*.



RTOS way:

- Minimal work performed in interrupt handlers
- Interrupt handlers only signal events to tasks
- RTOS scheduler manages realtime behavior
- Prioritized interrupts replaced with prioritized tasks
- No benefit in nesting interrupts

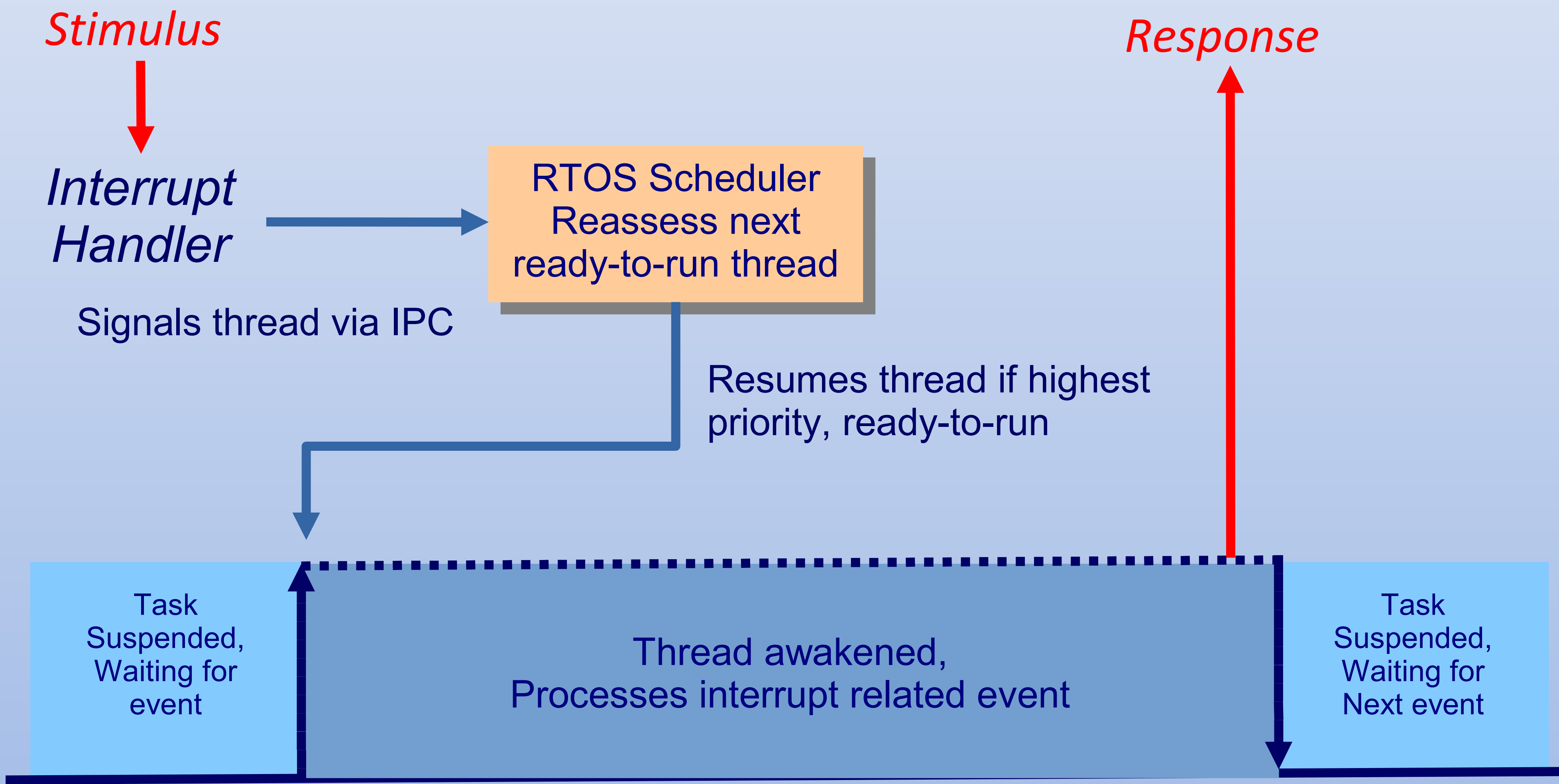
Properties of NuttX Tasks

- Task = A thread within an environment (like a Linux process)
- Thread = “Normal” sequence of instruction execution

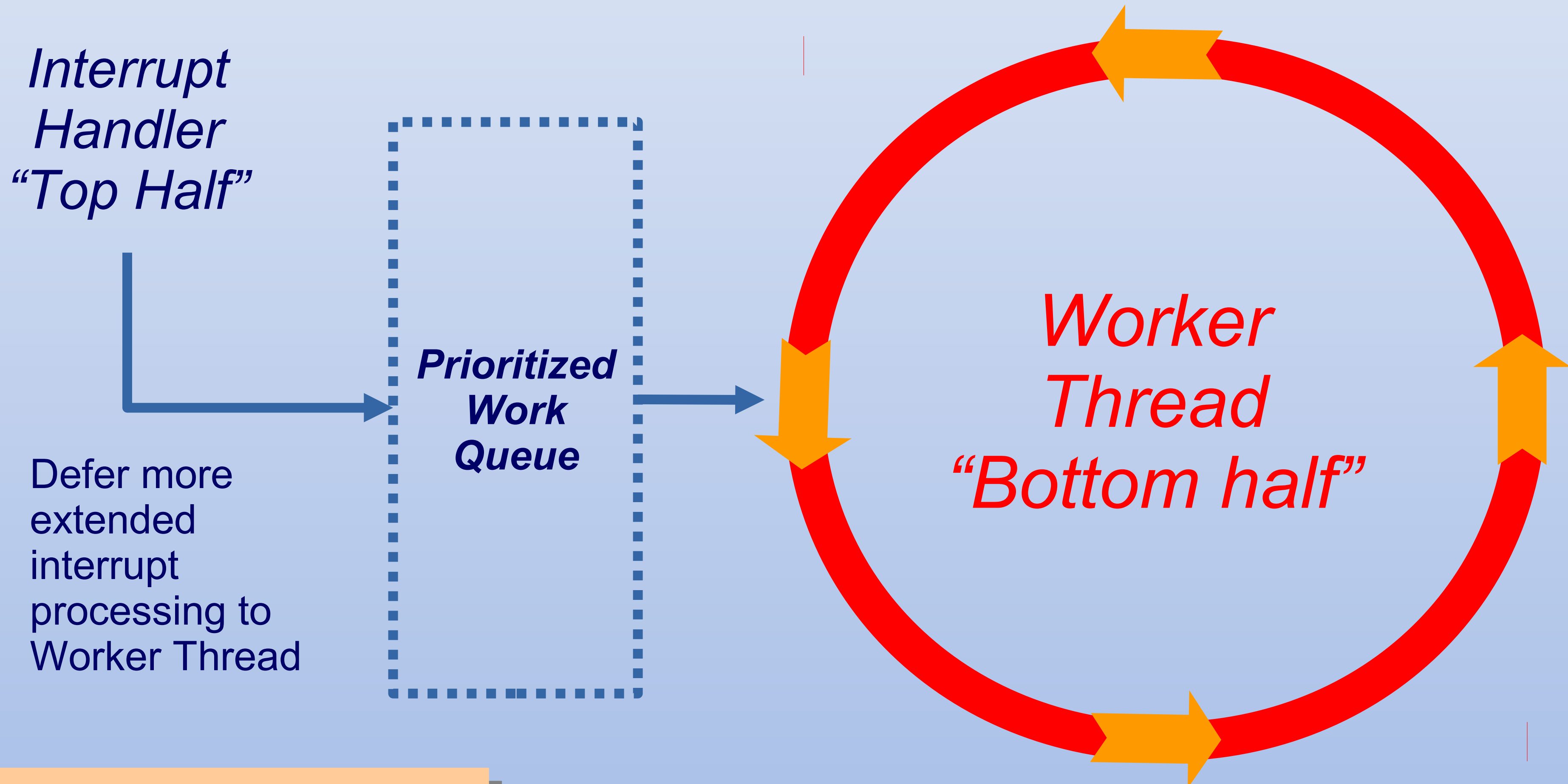
- Each thread has its own stack
- Each thread has an execution priority managed by the OS
- Each thread is a member of a “task group”
- Share resources (like a Linux process)
- Can wait for events or resource availability

- Threads communicate via Interprocess Communications (IPC):
- POSIX Message Queues, Signals, Counting semaphores, etc.
- Standard / Linux compatible
- NuttX supports use of standard IPCs from interrupt handlers

RTOS Interrupt Processing



Work Queues

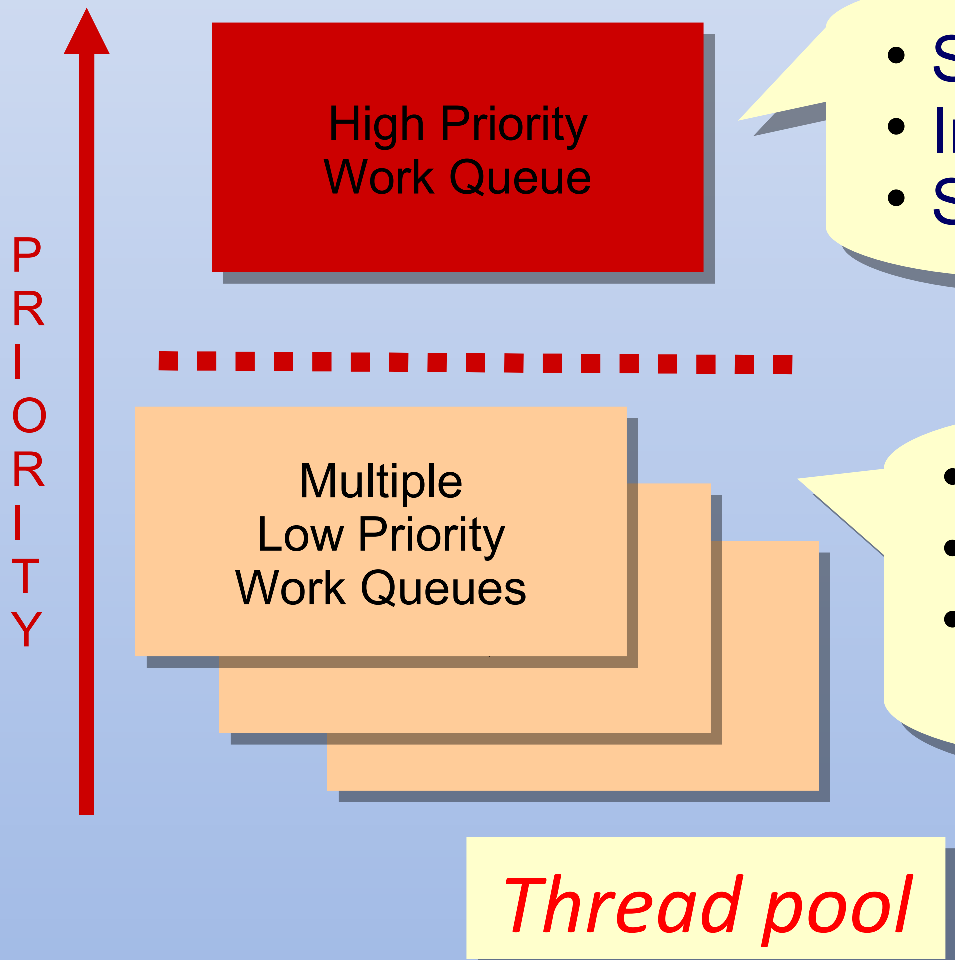


- Priority Queue
- Non-premptive
- Very high priority
- Inappropriate for extended processing

Non-deterministic!

Use with care!

Multiple Work Queues



- Single *high priority work queue*
- Intended for interrupt *"bottom half"*
- Should be highest priority

- Multiple *low priority work queues*
- Support priority inheritance
- Use to implement asynchronous I/O (AIO)

Components of Response Latency

- Stimulus Event
  - Hardware interrupt processing.
  - Delay may be extended if interrupts disabled
- Software interrupt processing
  - Thread state save (for Context Switch)
- Interrupt handler processing
  - IPC
  - Task execution may delayed if it does not have priority
- Interrupt return
  - State restore *OR* Interrupt Context Switch
- Thread processing
  - Output response

Synchronous vs Asynchronous Context Switch

Asynchronous Context Switch == Interrupt Context Switch
Critical part of realtime response
VERY efficient in NuttX… Near zero additional overhead

Synchronous Context Switch
Thread relinquishes CPU by waiting for event
NOT a critical part of realtime response
But may be important to overall performance and throughput

High Priority, Zero Latency Interrupts

- Software interrupt processing overhead
  - Thread state save and restore (for interrupt Context Switch)

- ARM Cortex-M*
  - Can support direct vector to C-code
  - Zero (software) latency

- NuttX implements with:
  - Higher interrupt priority
  - Direct vector to C code
  - Indirect interrupt context switches via PENDSV

- Important to support:
  - Very high rate interrupts
  - Interrupts with very low latency requirements

Realtime Schedulers

Realtime behavior realized via *OS scheduler*

RTOS provides tools only *enable* realtime designs
But a bad application design may still not be realtime

Scheduling Disciplines:
- Traditional / POSIX Schedulers
- Deadline Scheduler (and other modern schedulers)

Rate Monotonic Scheduling (RMS)

Deadline Schedulers

- Example: Linux SCHED_DEADLINE
  - Earliest Deadline First (EDF)

- ✓ Highly managed
- ✓ High processing overhead
- ✓ Complex
- ✓ Difficult to configure correctly
- ✓ Non-standard
- ✓ Not commonly used in a small RTOS
- ✓ *Not currently supported by NuttX*

Standard / POSIX Schedulers

Primary NuttX Specification: *OpenGroup.org*

Standard Schedulers specified at *OpenGroup.org*:

`SCHED_FIFO`

- For Managed latency
- Supports *Rate Monotonic Scheduling* (RMS)

`SCHED_RR`

- Not realtime
- Time-slicing
- Balanced throughput

`SCHED_SPORADIC`

- Dynamic prioritization to achieve processing *budget*
- For *background* tasks with guaranteed bandwidth

Response latency vs. Throughput trade-offs

Rate Monotonic Scheduling

Can achieve realtime behavior under certain circumstances:

- Strict priority scheduling
- Static priorities
- Priorities assigned according to the rate monotonic conventions

Threads with shorter periods/deadlines are given higher priorities

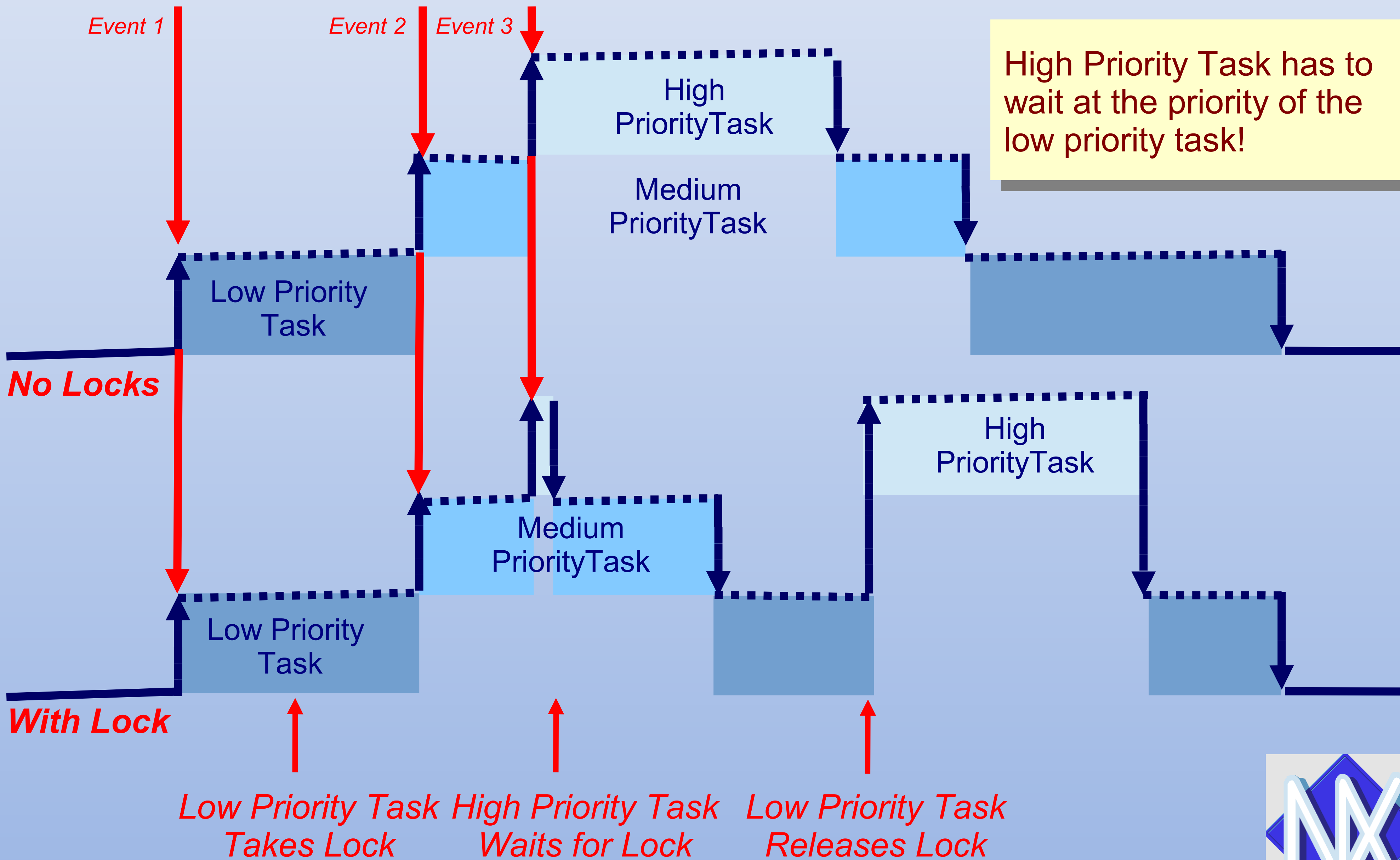
And this ***unrealistic*** assumption:

- No resource sharing
- No waiting for resources
- Example: hardware, queue, etc.
- No semaphores or locks.
- No disabling pre-emption
- No disabling interrupts
- No critical sections

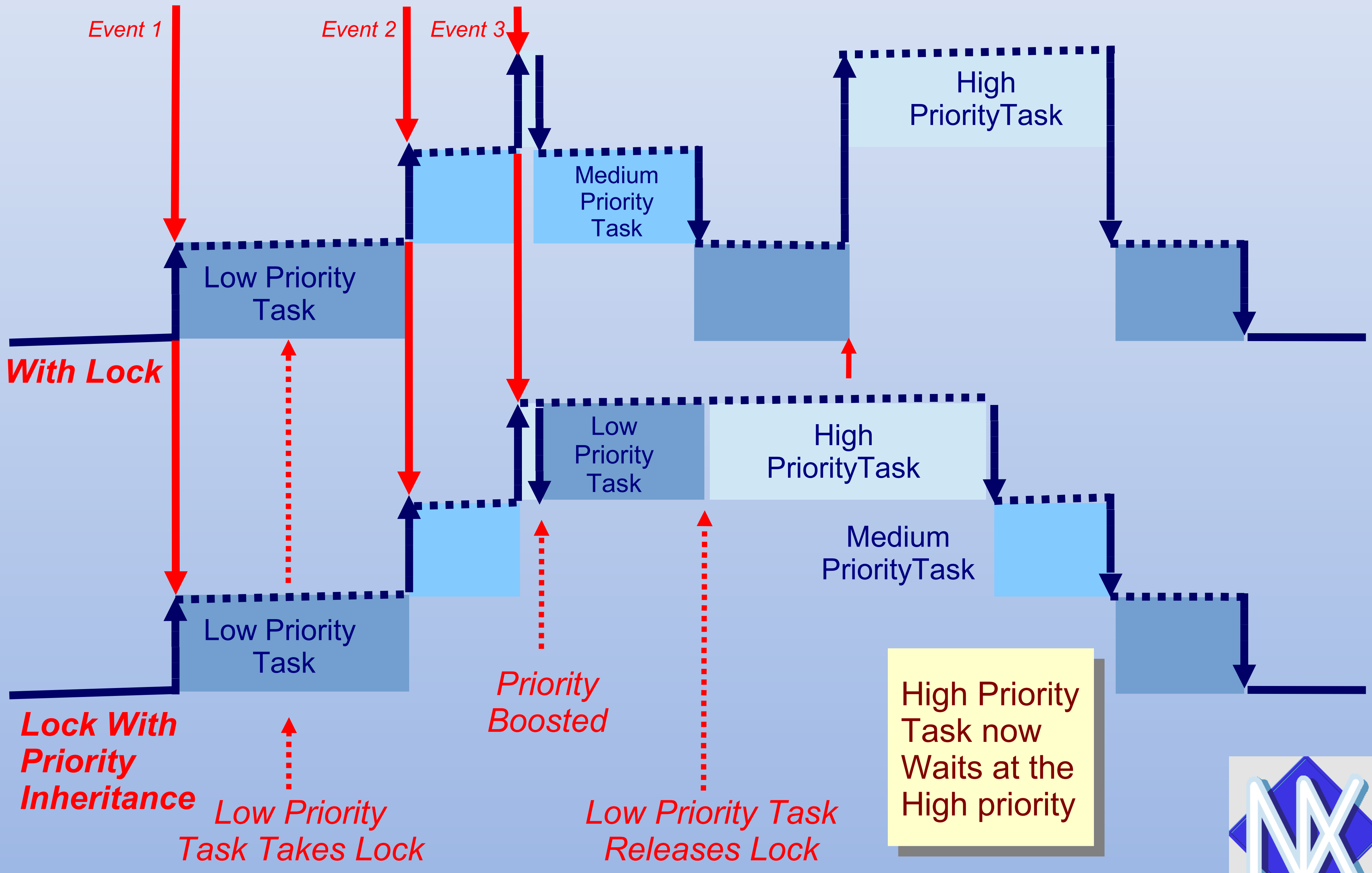
Priority Inversion / Priority Inheritance

Priority Inversion

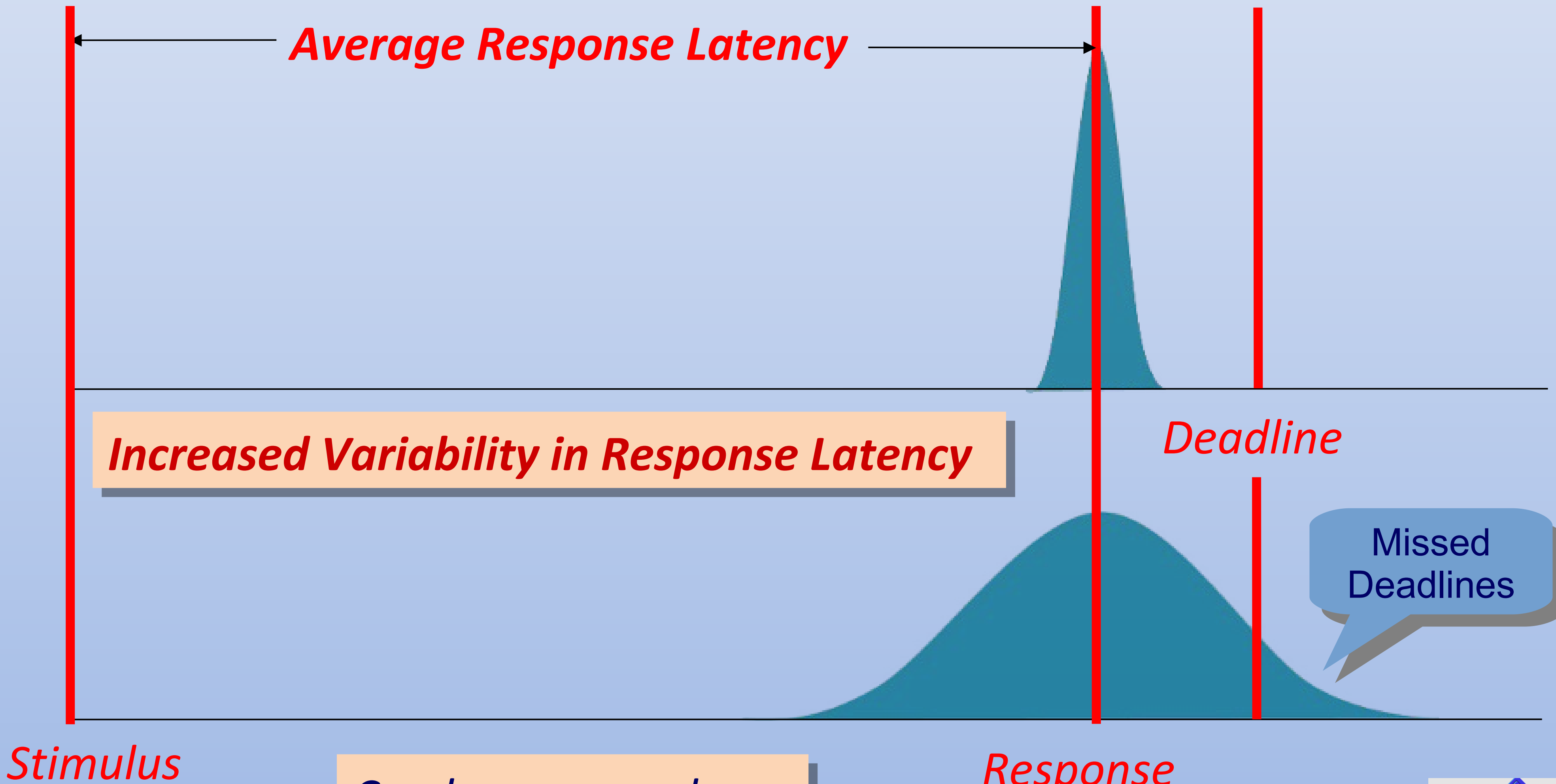
High Priority Task has to wait at the priority of the low priority task!



Priority Inheritance



Effect of Violations of Assumptions



Can be managed with good designs

Mixing Real-Time and non-Real-Time Tasks