NuttX RTOS ***Beginnings***

Gregory Nutt

Made in Costa Rica

About Me

Math/Physics

Digital Signal Processing

Math

1970

Neuro-science

Computer Science

Academia

First mult-tasking, pre-emptive RTOS, 1982

1980

Cooperative Schedulers

Hewlett Packard

Military Avionics

Cooperative Schedulers

Commercial Electronics

1990

LaserJet POSIX RTOS, 1996

Costa Rica

LaserJet Firmware Architect

Open Source

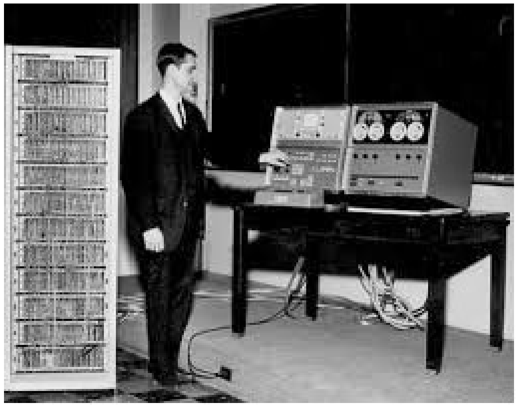
2000

Ridgerun.com

NuttX Release to Open Source, 2007

NX-Engineering, SA

Graduate School Days



LINC

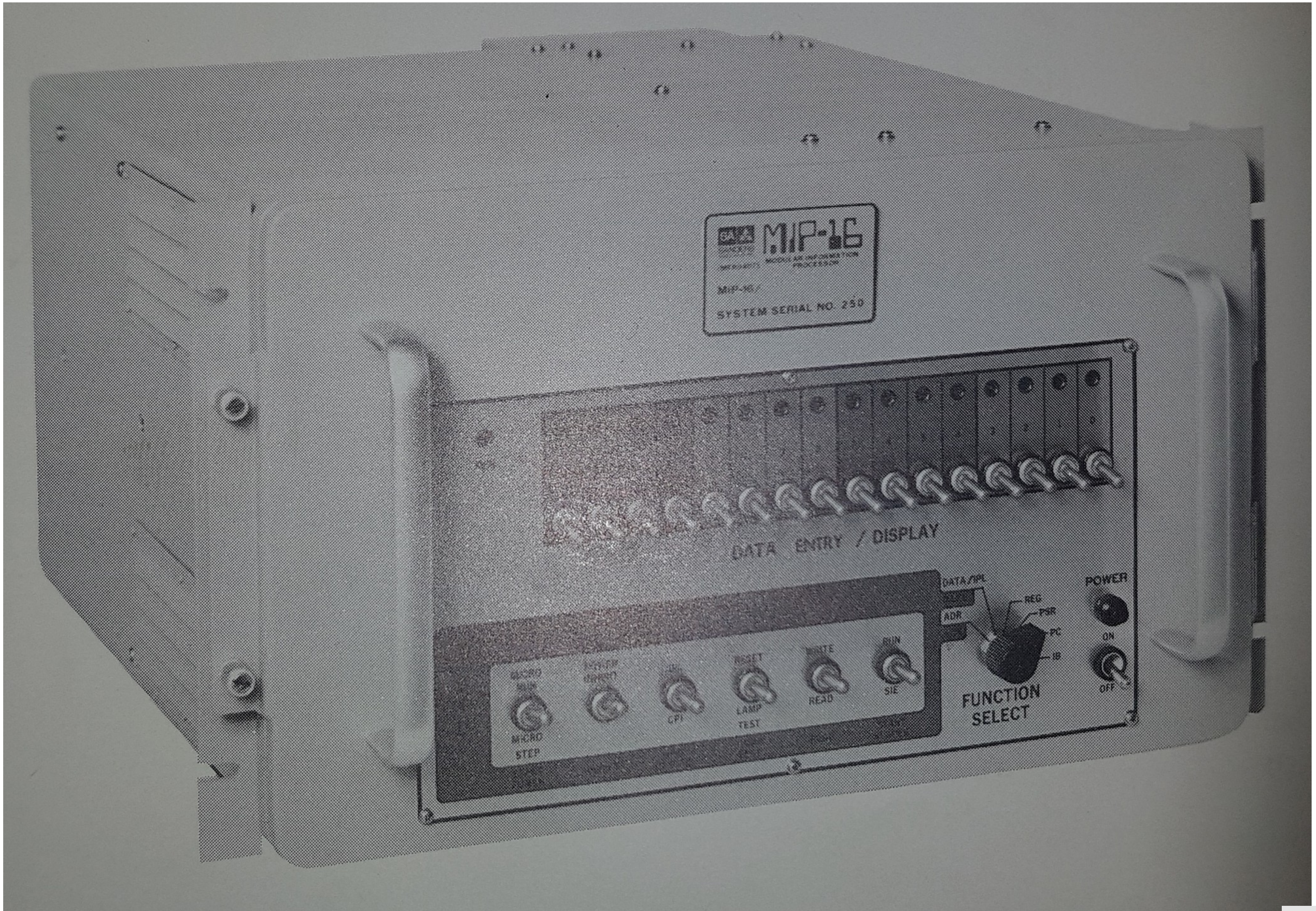
LINC-8

PDP-8/i

PDP-8/e

TRS-80
Model 1

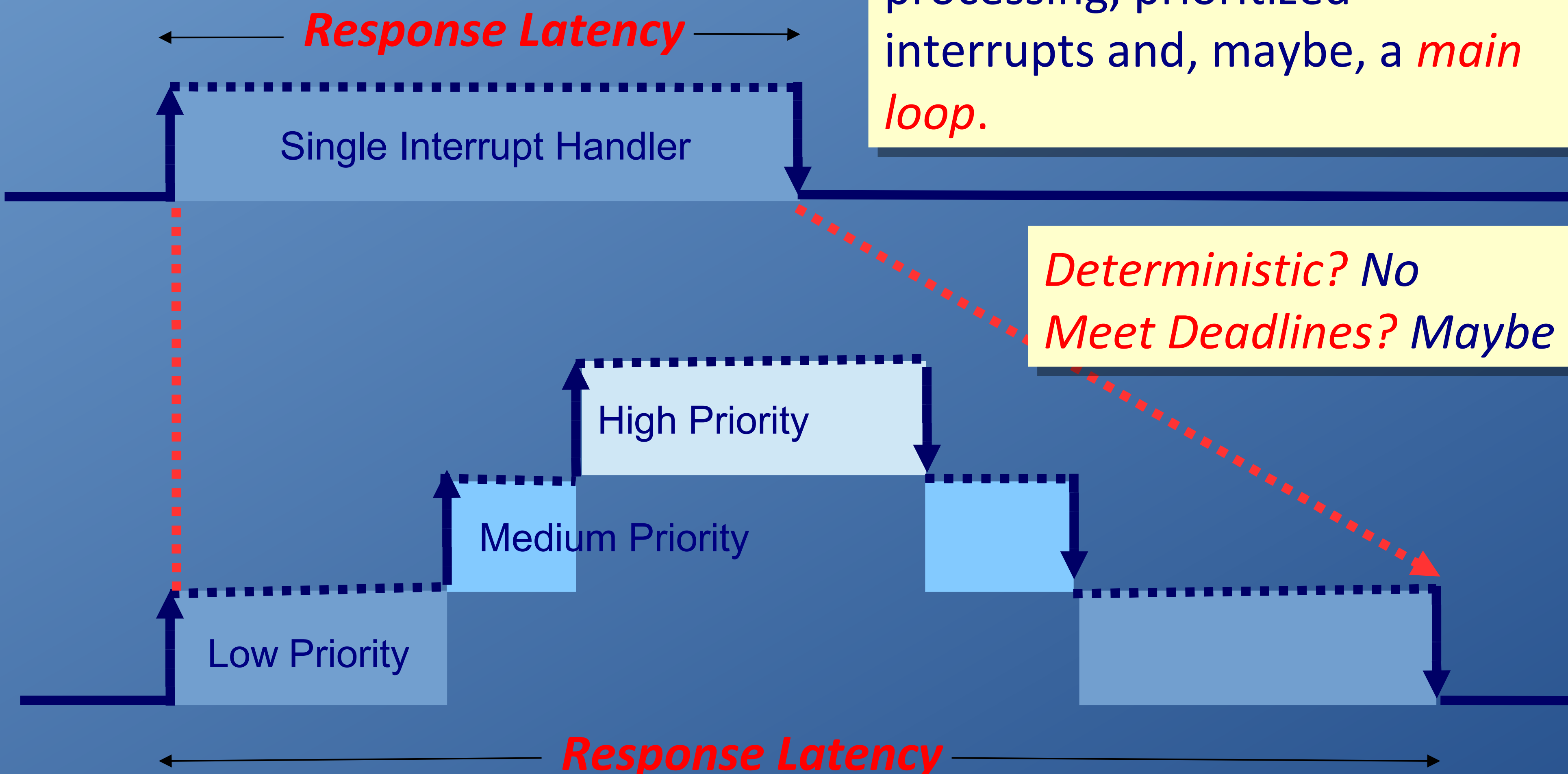
Hardware of First RTOS



Ruggedized MIP-16 with Optional Control Panel

Interrupt Driven – OS #1 (Bare Metal)

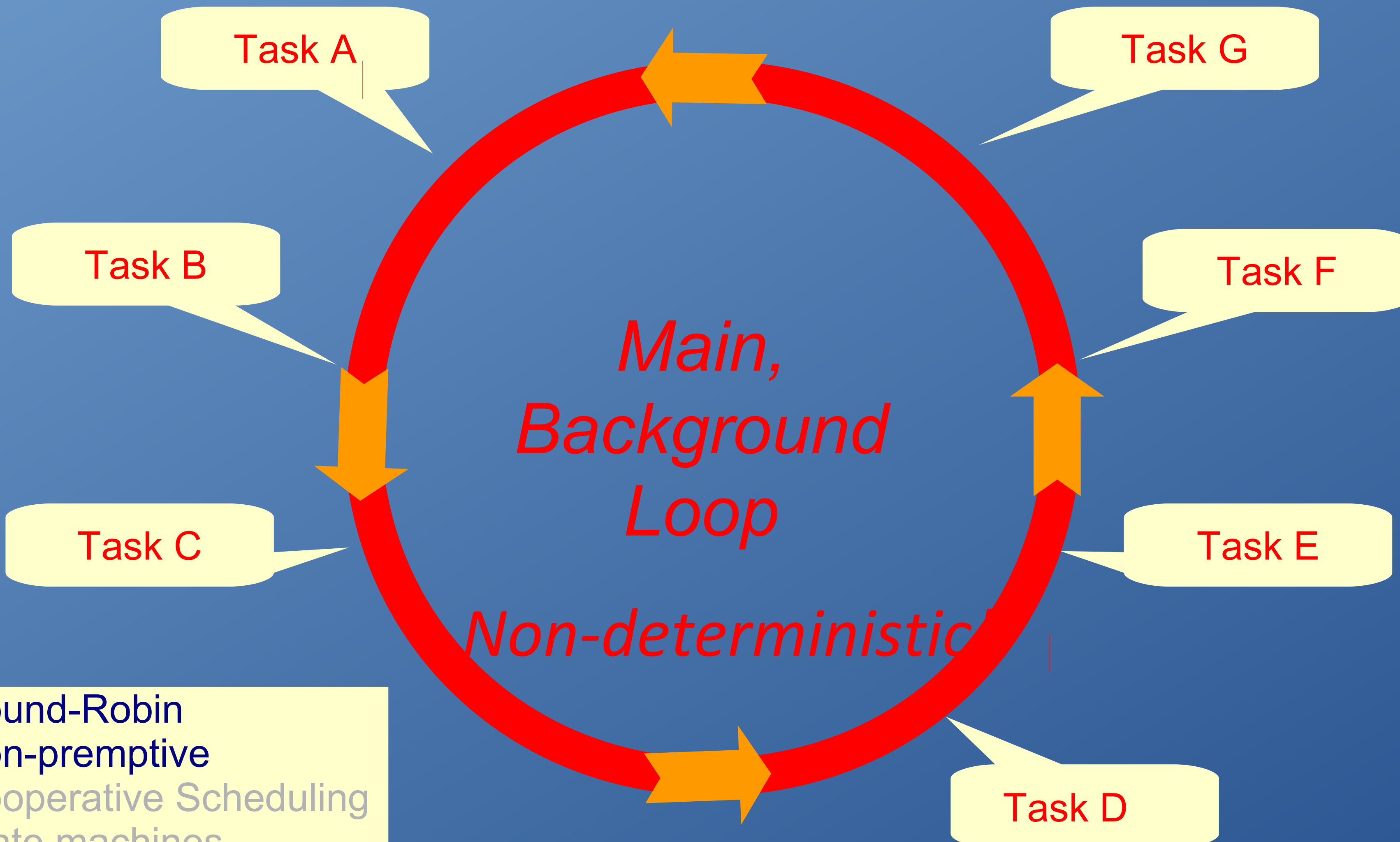
No OS: Extensive interrupt processing, prioritized interrupts and, maybe, a *main loop*.



Deterministic? No
Meet Deadlines? Maybe

Problems: Stacked, Can lose interrupts.
No waiting, all run to completion.

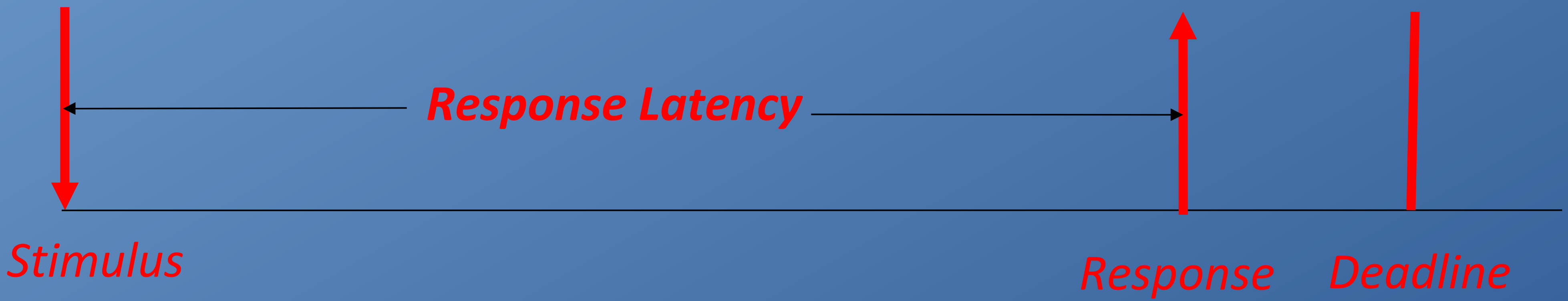
Main Loop – OS #1 (Cont'd)



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

Non-deterministic!

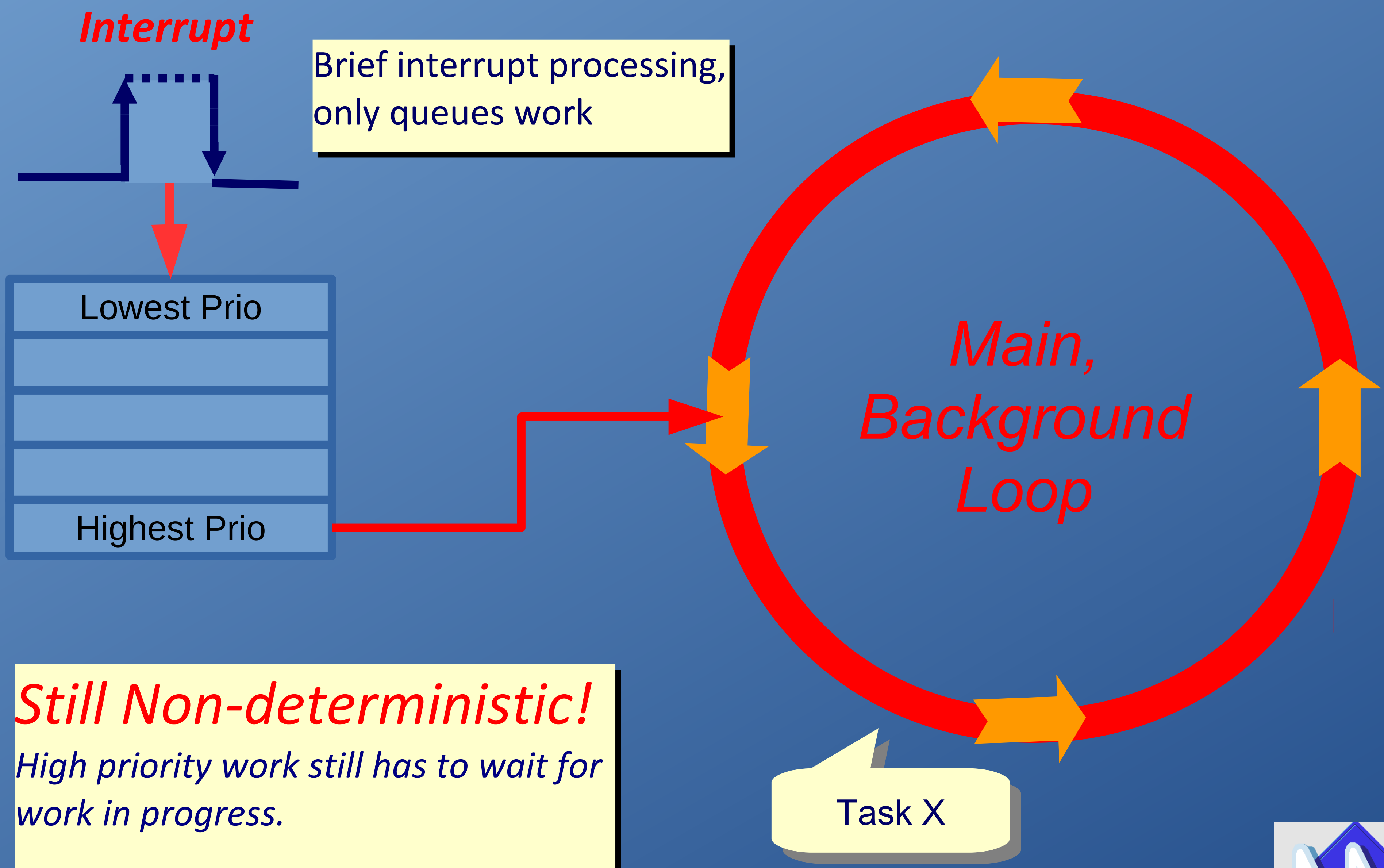
Real Time == Deterministic



Real time does not mean "fast"

Real time systems have ***Deadlines***

Main Loop with Priority Queue – OS #2



Brief interrupt processing, only queues work

Still Non-deterministic!

High priority work still has to wait for work in progress.

Main Loop with Cooperative Scheduler– OS #3

Task X

```
switch (state)
 {
     case state A:
       Start event processing;
       state = state B;
       Reschedule;
       Break;

     Case state B:
       Continue event processing;
       State = state C;
       Reschedule;
       Break;

    Case state C:
       Finish event processing;
       State = state X;
       Break;

     Case state X:
        Break;
 }
```

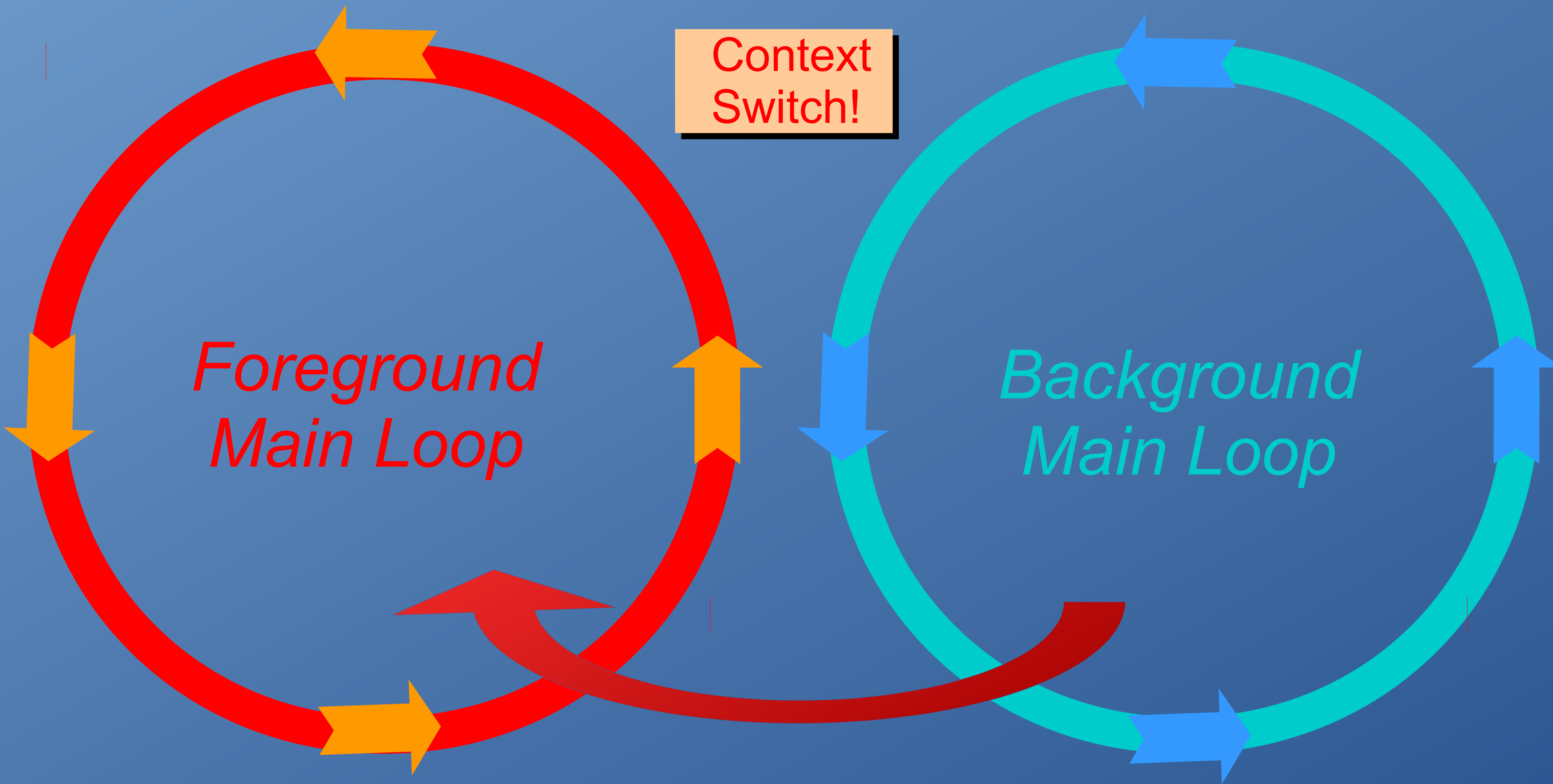
- Non-premptive
- Cooperative Scheduling

- Divide event processing up into pieces
- Manage with a state machine
- Reschedule to allow higher priority tasks
- Other *ad hoc* strategies

Still Non-deterministic!

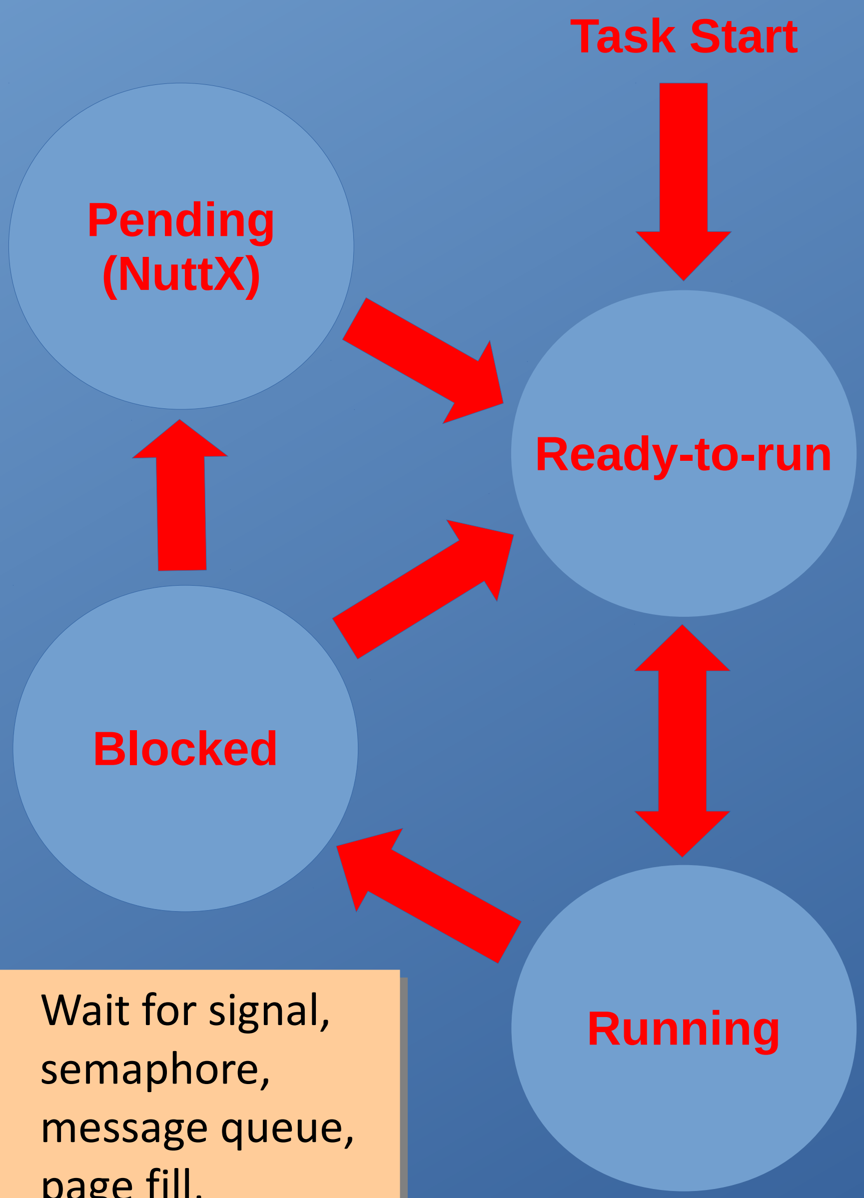
High priority work still has to wait for work in progress.

Foreground / Backgound Main Loops – OS #4



Paritially Deterministic

Pre-emptive OS – OS #5



The DEC connection

Fully pre-emptible
Context switch:
Think setjmp/longjmp on steriods

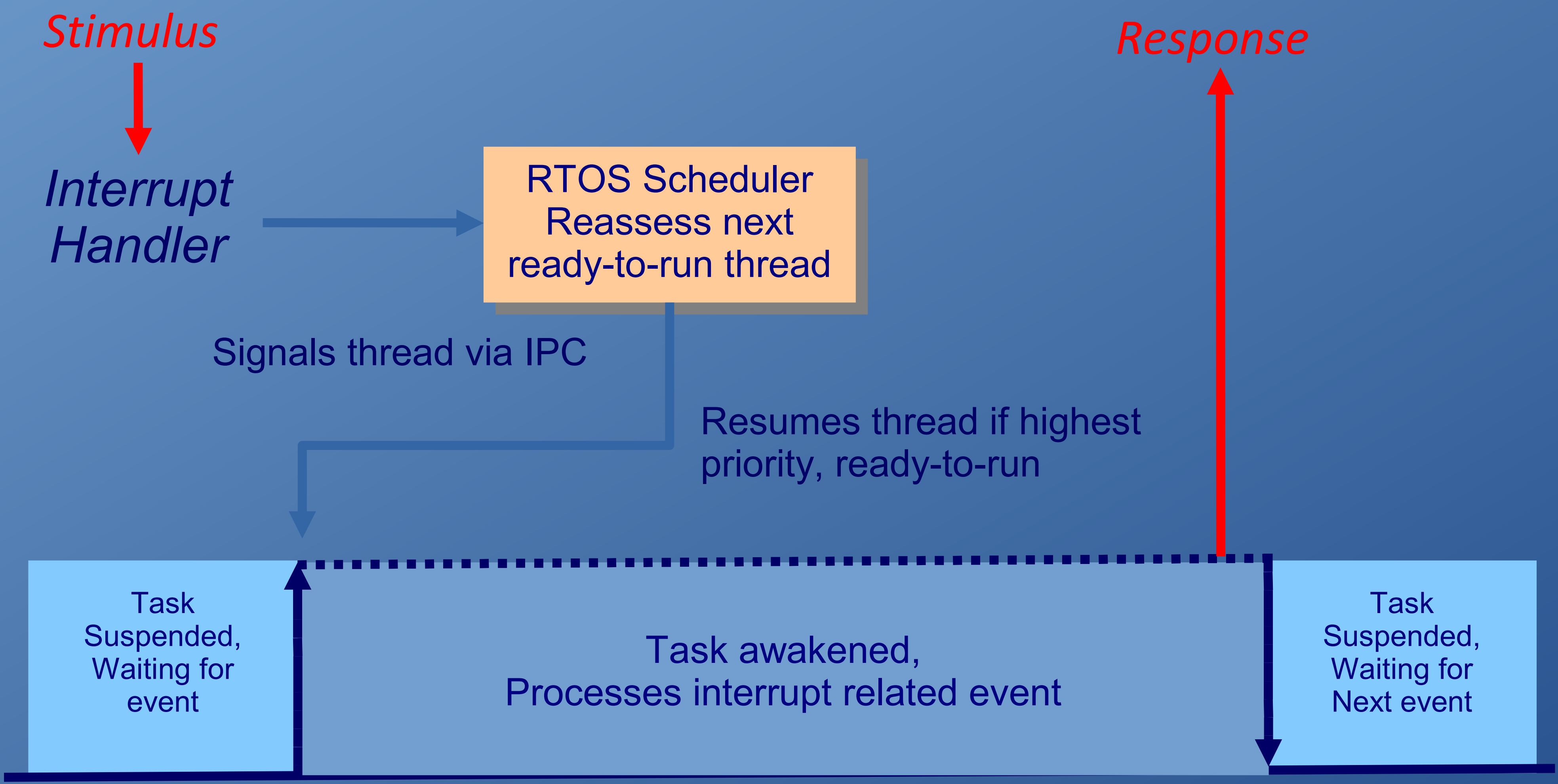
Task Control Block (TCB)

States represented by lists of TCBs

Wait for signal,
semaphore,
message queue,
page fill,
stopped, etc.

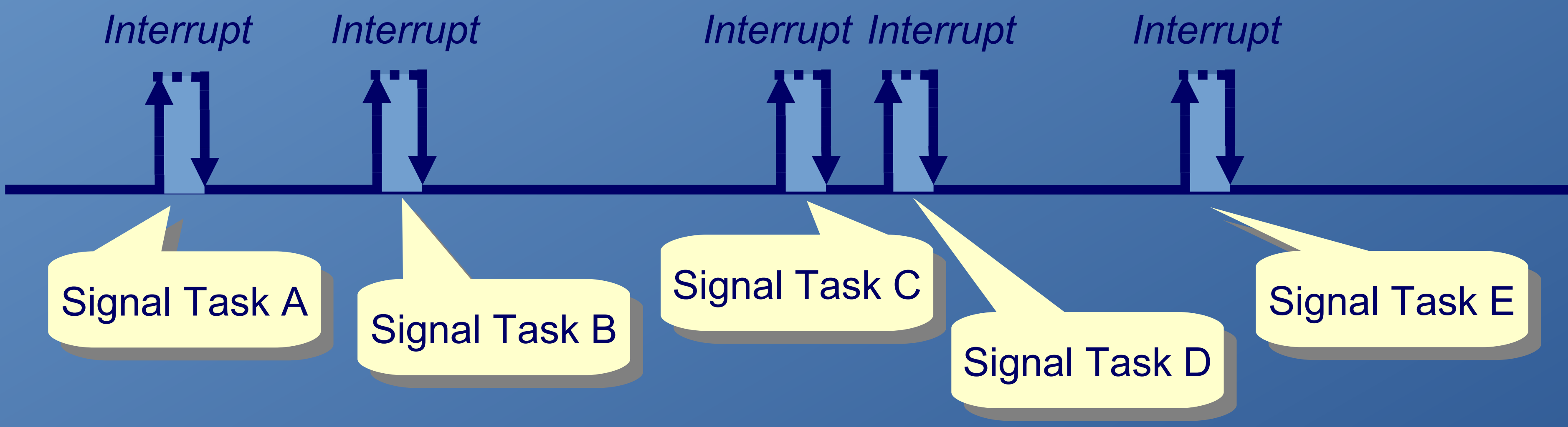
Highest Priority
Ready-to-run task
is Running

RTOS Interrupt Processing



RTOS Interrupts

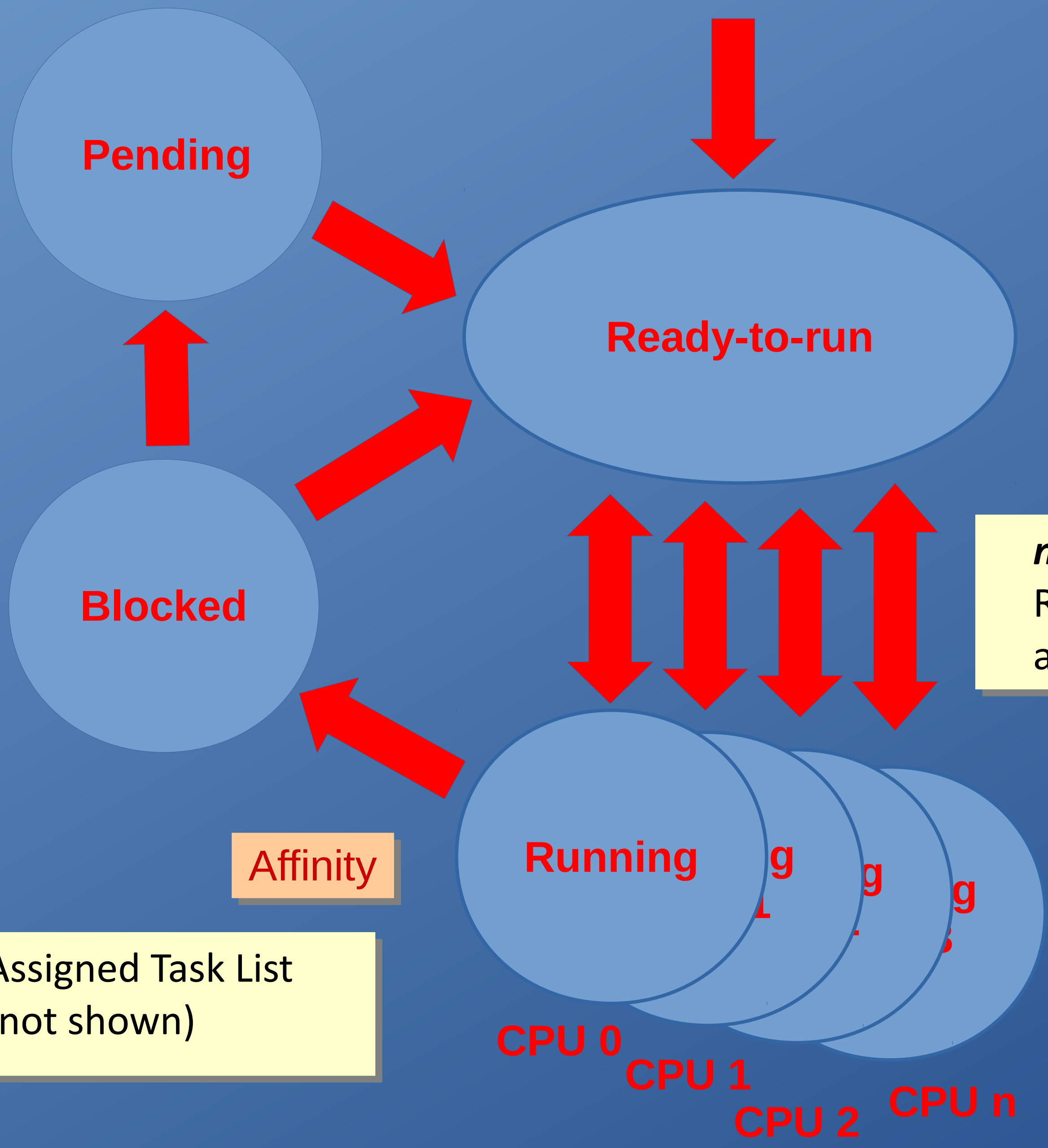
No OS way: Extensive interrupt processing, prioritized interrupts and, maybe, a *main loop*.



RTOS way:

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

SMP



n Highest Priority Ready-to-run tasks are Running

Spinlocks

Affinity

Assigned Task List (not shown)

Rate Montonic Scheduling

Can achieve Real-Time behavior under certain circumstances

- Strict priority scheduling
- Static priorities
- Priorities assigned according to
- Rate Monotonic conventions

Threads with shorter periods/ deadlines are assigned the highest priorities.

And this ***unrealistic*** assumption:

- No resource sharing
- No waiting for resources
- No semaphores or locks
- No critical sections
- No disabling pre-emption
- No disabling interrupts

Why POSIX?

Why not...

- Versus custom *ad hoc* OS interface
- POSIX device model vs HAL
- Like simpler FreeRTOS, ChibiOS, Zephyr, mbed, RIOT, etc.

At this point POSIX is the NuttX identity

- Portability
- Linux compatibility
- Complex build models: PROTECTED and KERNEL builds

Work Queues

Interrupt Handler “Top Half”

Prioritized Work Queue

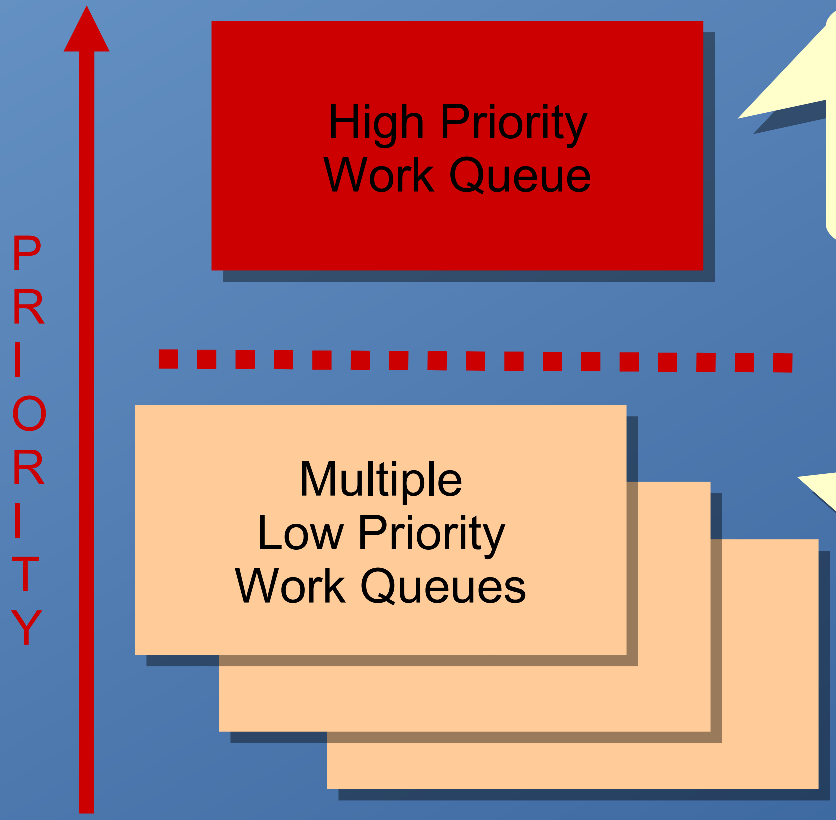
Defer more extended interrupt processing to Worker Thread

- Priority Qeue
- 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)

Thread pool