

NuttX RTOS

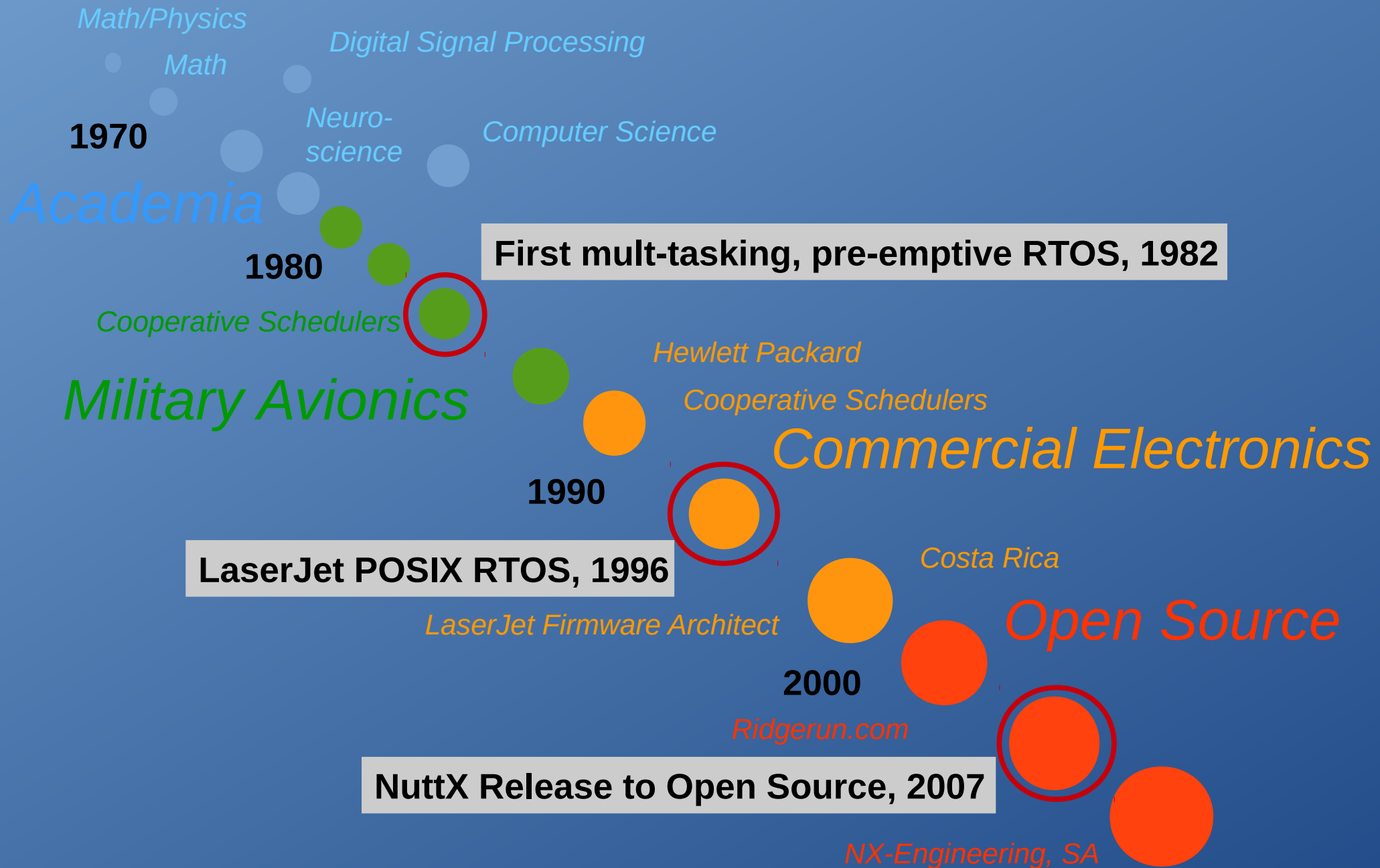
Beginnings



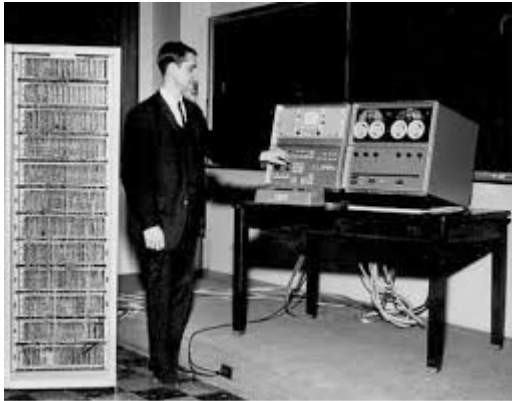
Gregory Nutt



About Me



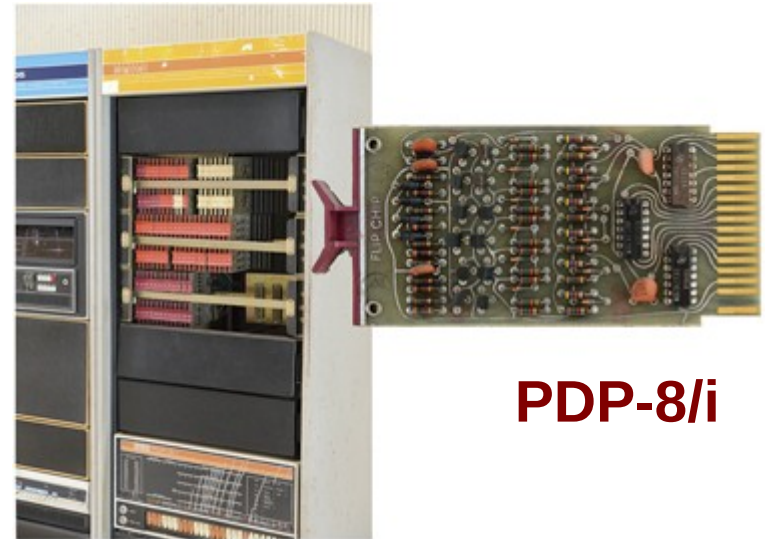
Graduate School Days



LINC



LINC-8



PDP-8/i

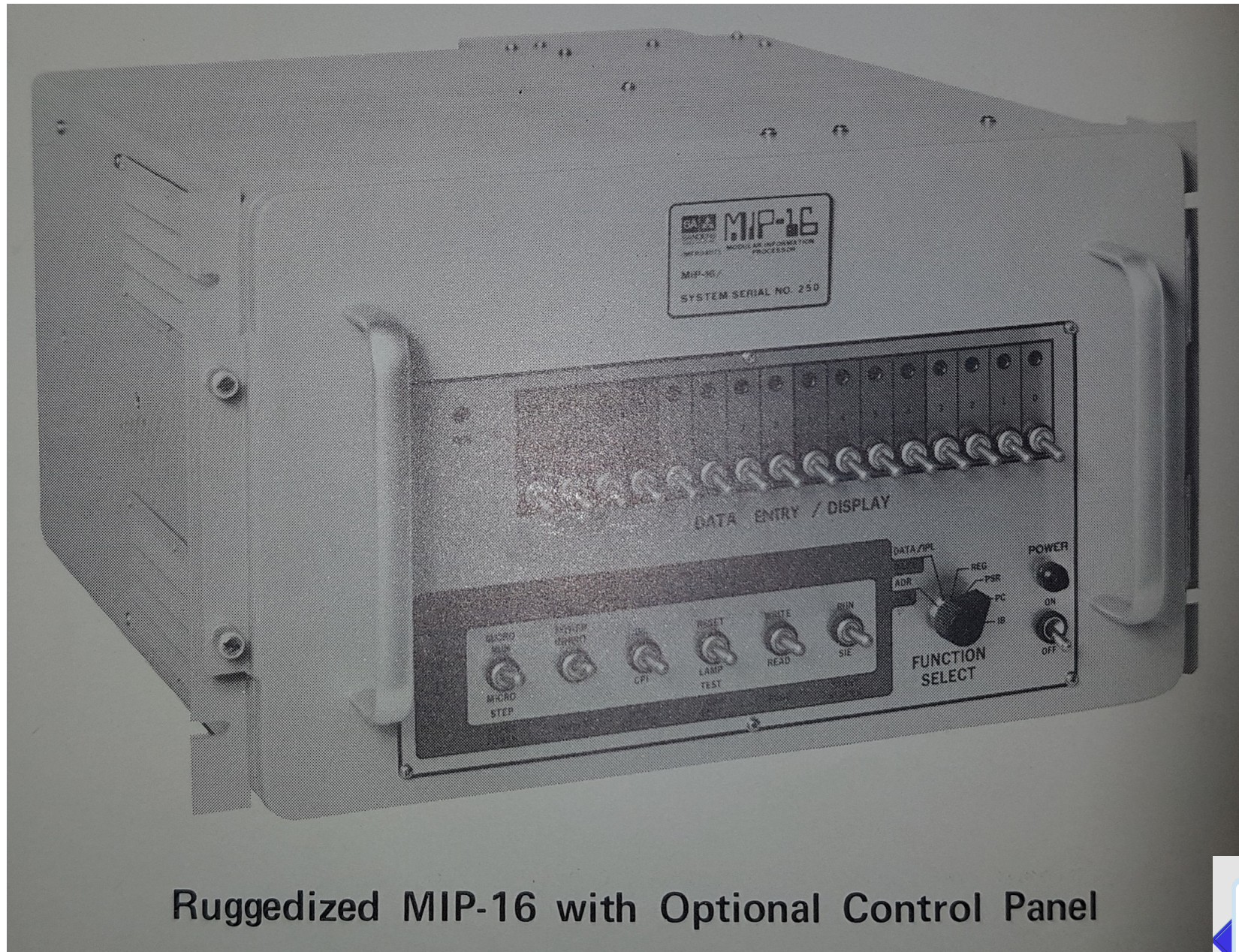


PDP-8/e

**TRS-80
Model 1**



Hardware of First RTOS

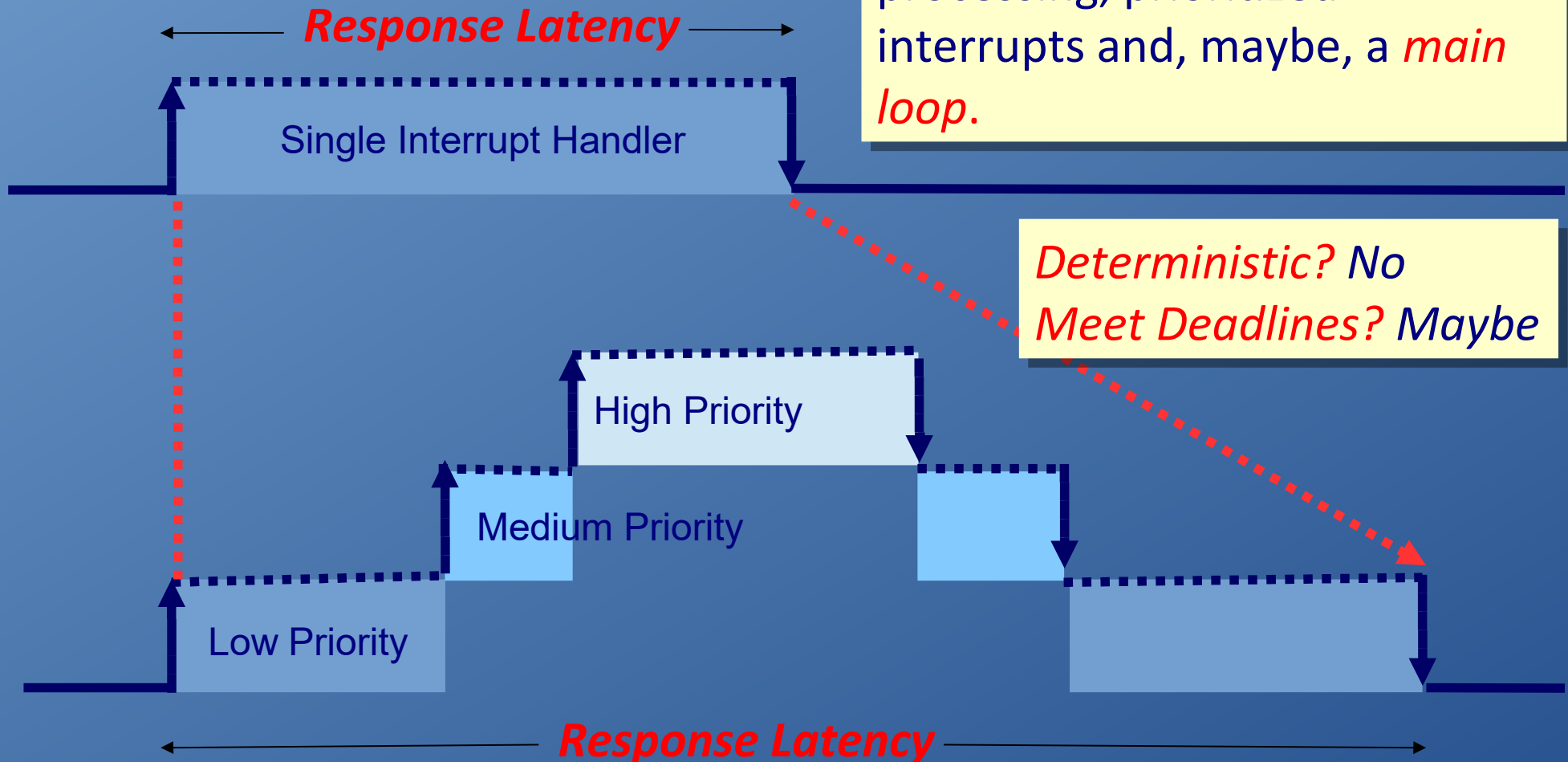


Ruggedized MIP-16 with Optional Control Panel



Interrupt Driven – OS #1 (Bare Metal)

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

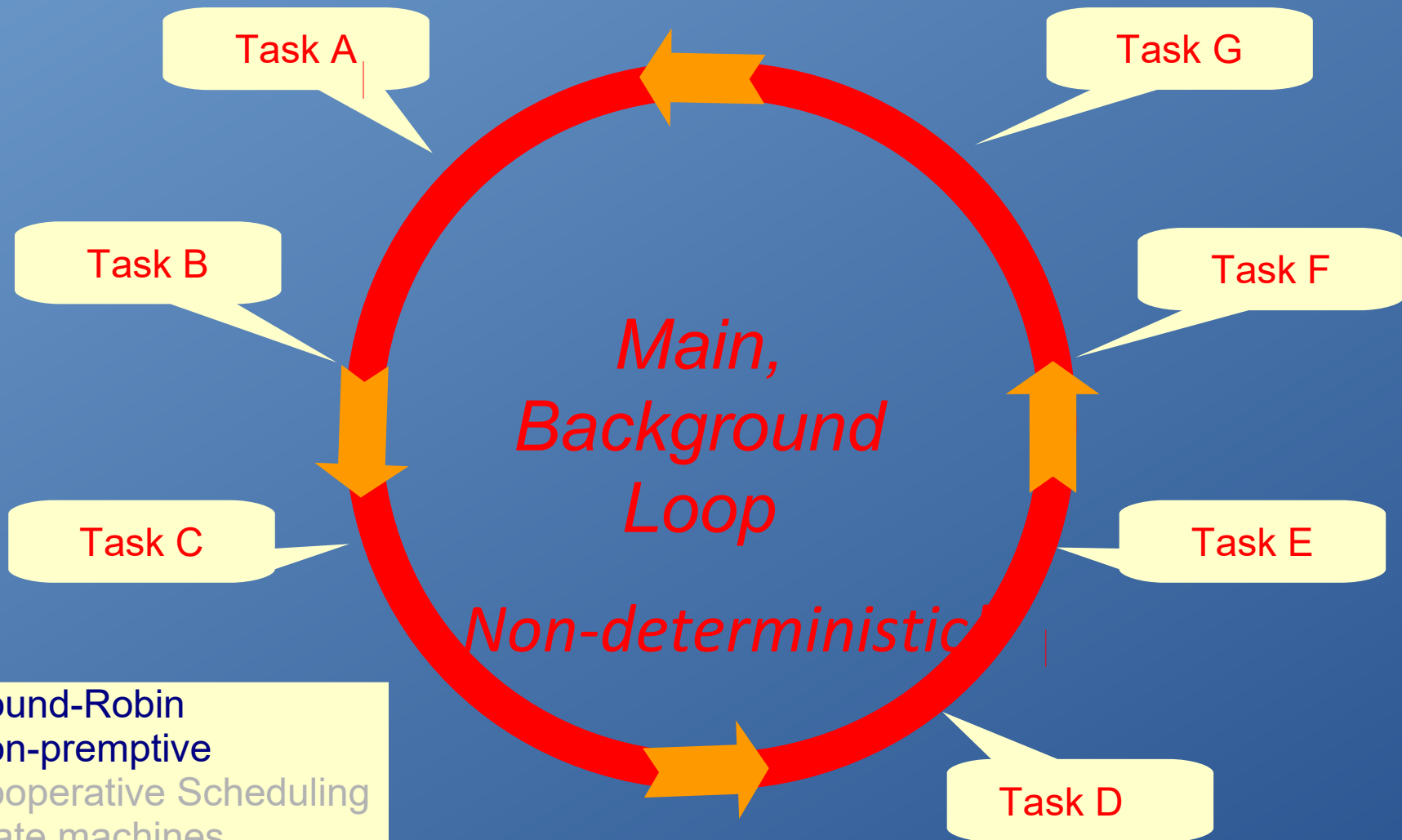


Deterministic? No
Meet Deadlines? Maybe

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



Main Loop – OS #1 (Cont'd)

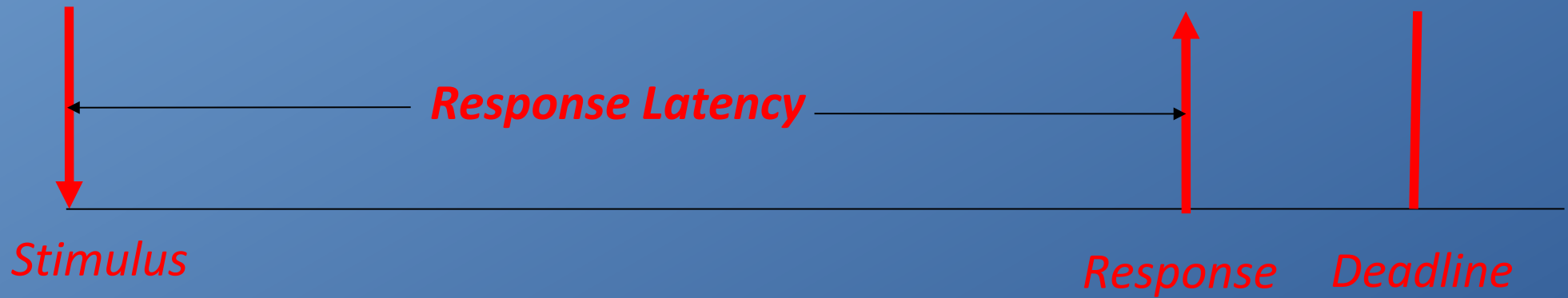


- Round-Robin
- Non-preemptive
- Cooperative Scheduling
- State machines
- *Ad hoc* strategies

Non-deterministic!



Real Time == Deterministic



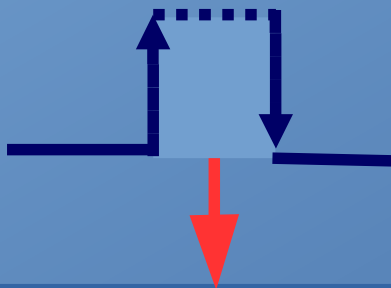
Real time does not mean “fast”

Real time systems have ***Deadlines***



Main Loop with Priority Queue – OS #2

Interrupt



Brief interrupt processing,
only queues work



*Main,
Background
Loop*

Still Non-deterministic!

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

Task X



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-preemptive
- Cooperative Scheduling

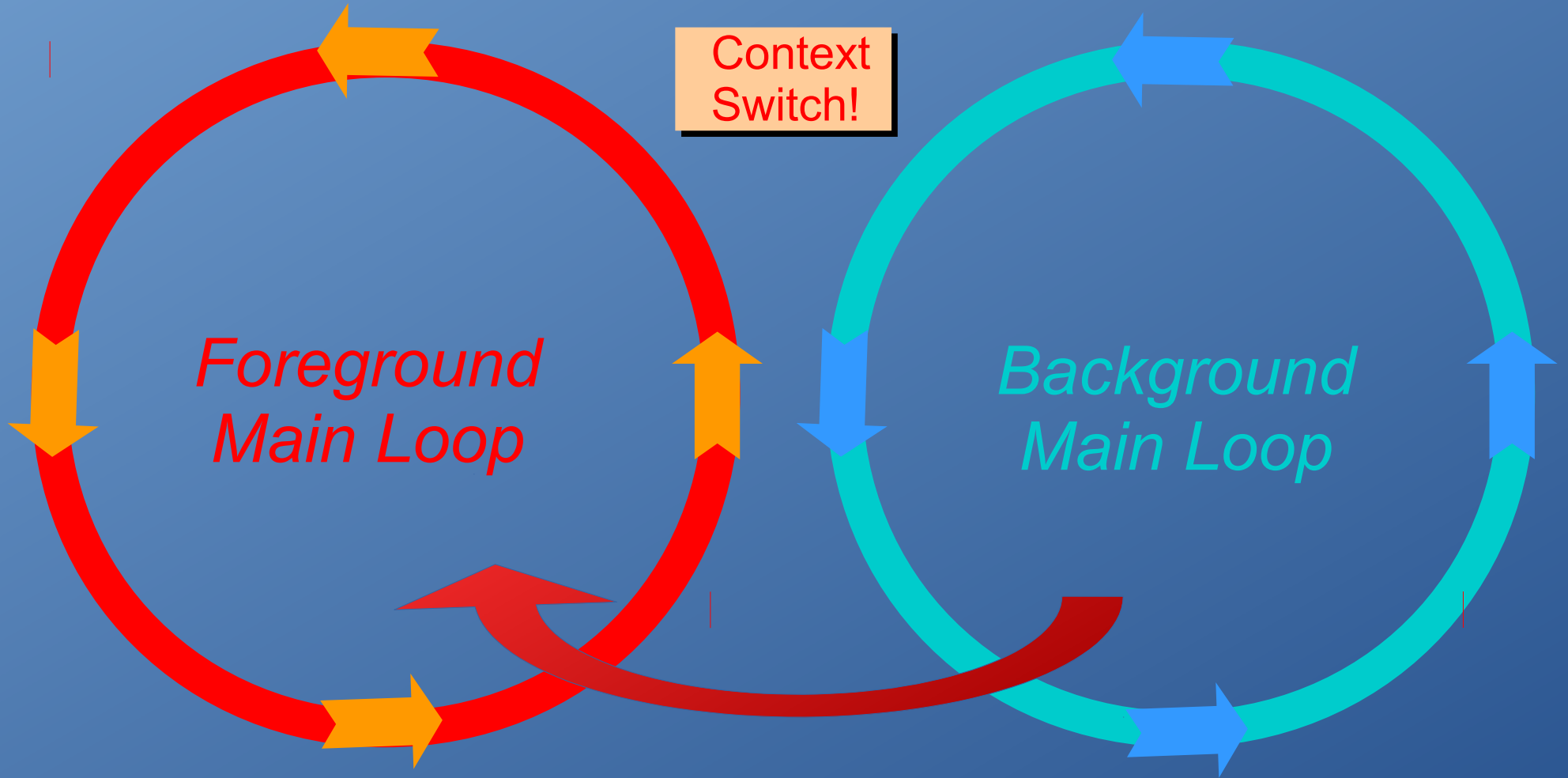
- Divide event processing up into pieces
- Manage with a state machine
- Reschedule to allow higher priority tasks
- Other *ad hoc* strategies

Still Non-deterministic!

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



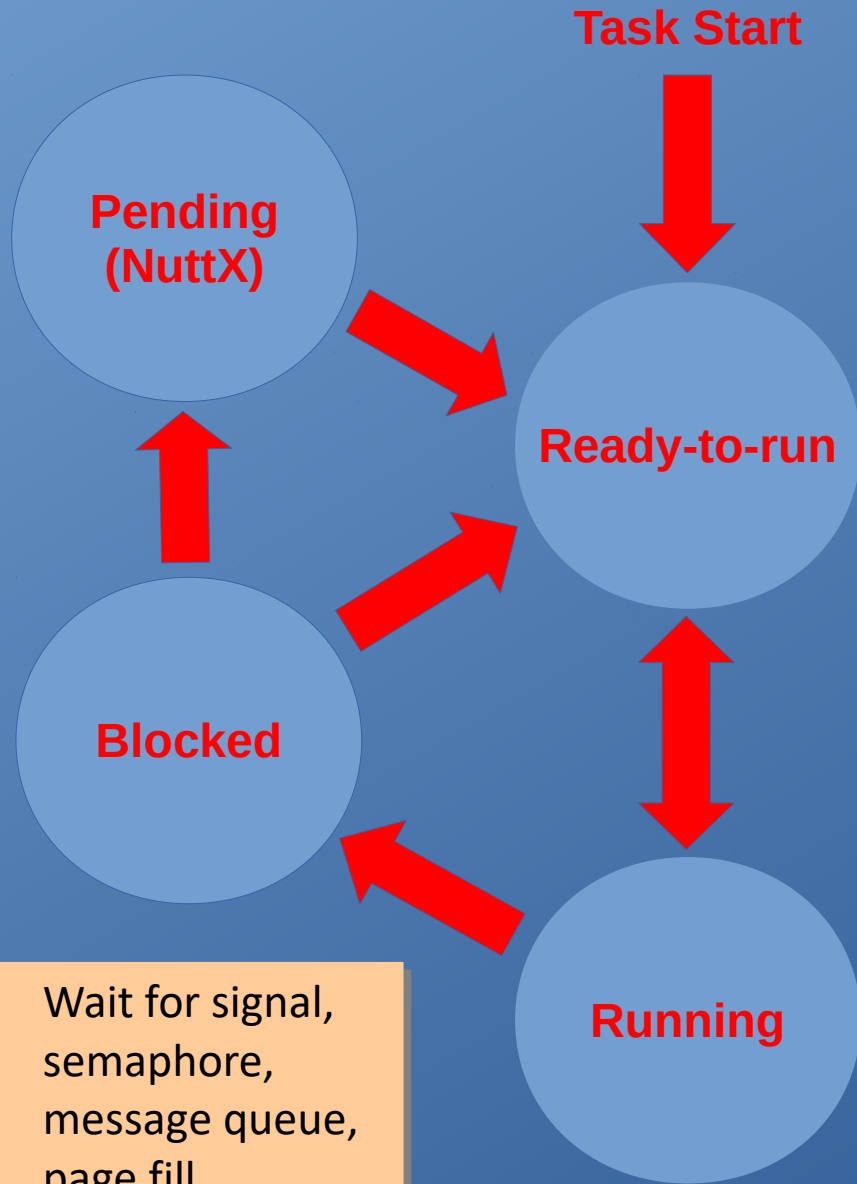
Foreground / Background Main Loops – OS #4



Partially Deterministic



Pre-emptive OS – OS #5



The DEC connection

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

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

Stimulus



*Interrupt
Handler*



RTOS Scheduler
Reassess next
ready-to-run thread

Signals thread via IPC



Resumes thread if highest
priority, ready-to-run

Task
Suspended,
Waiting for
event

Task awakened,
Processes interrupt related event

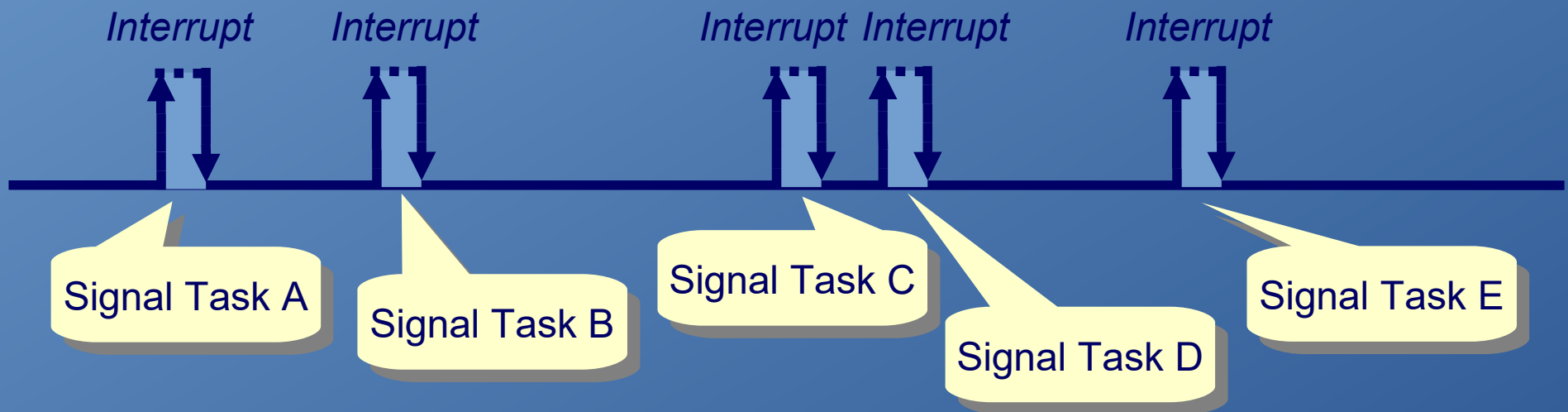
Task
Suspended,
Waiting for
Next event

Response



RTOS Interrupts

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

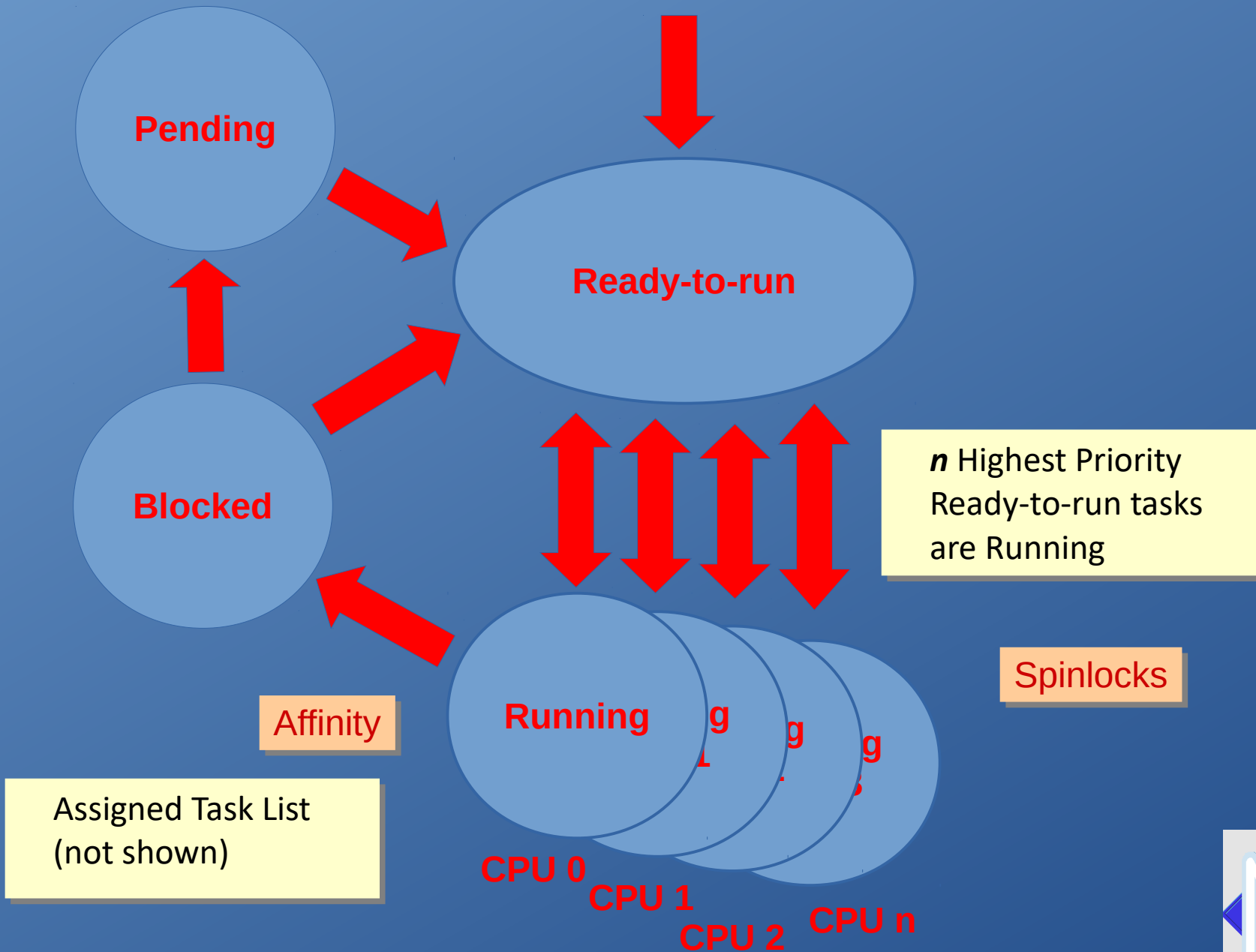


RTOS way:

- 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



Rate Monotonic 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

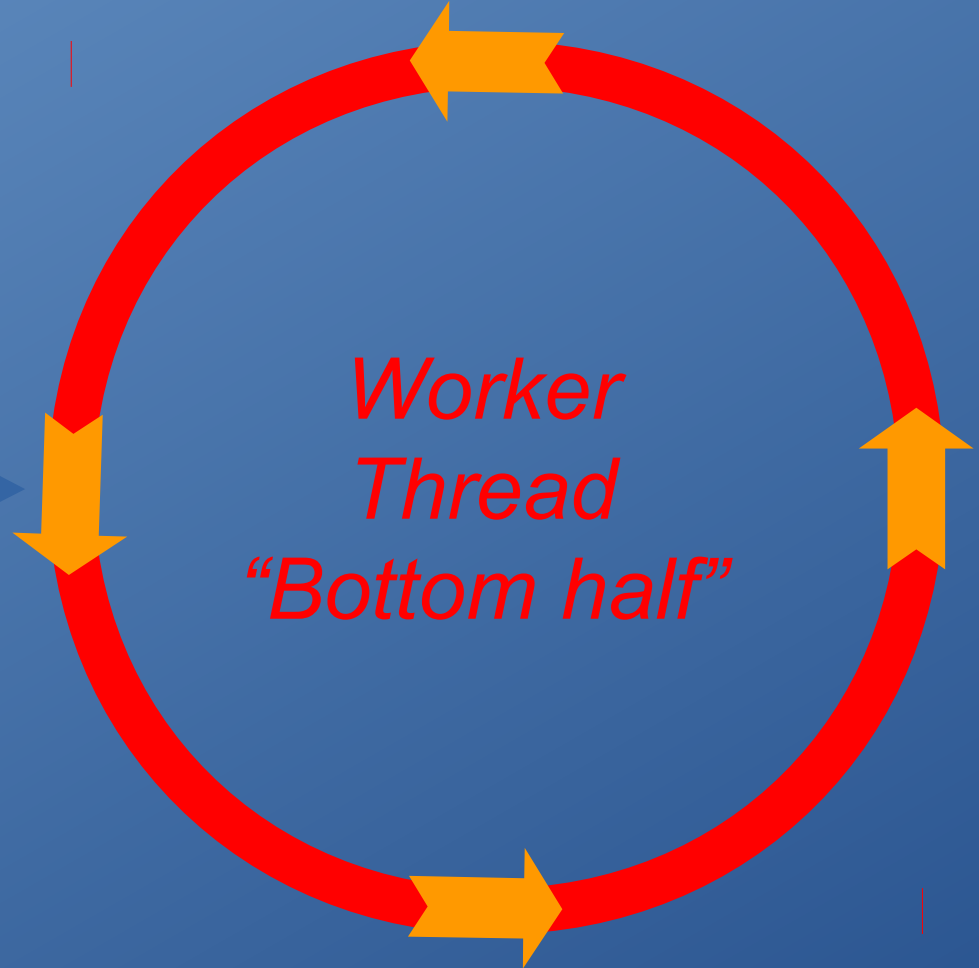
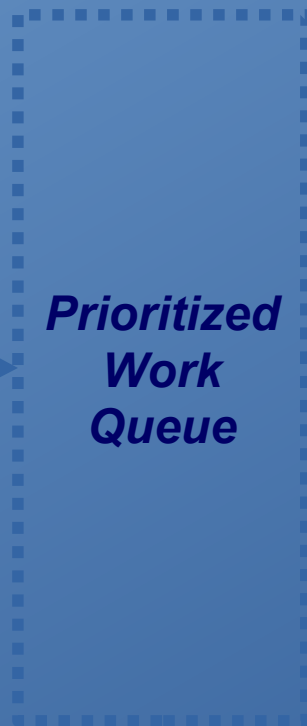
- Portability
- Linux compatibility
- Complex build models: PROTECTED and KERNEL builds



Work Queues

*Interrupt
Handler
"Top Half"*

Defer more
extended
interrupt
processing to
Worker Thread



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

Non-deterministic!

Use with care!



Multiple Work Queues

