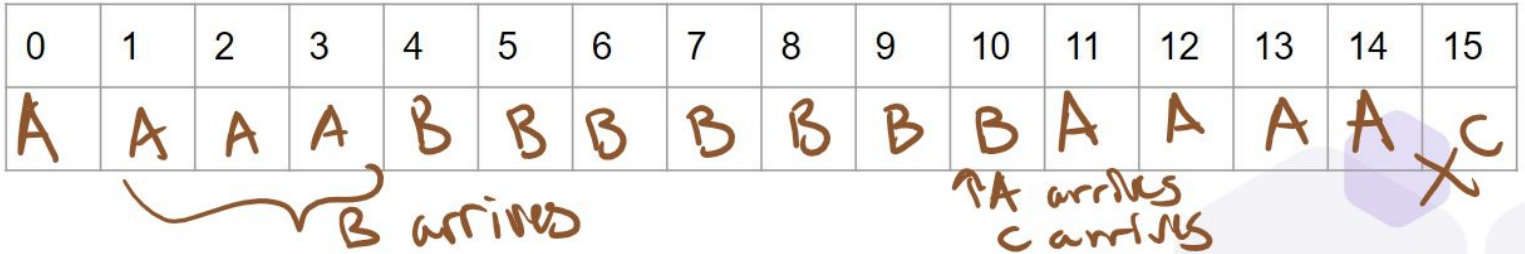
13: Scheduling and RTOS

Latency and priority

High priority interrupt: A (4 ms every 10 ms)

Lower priority interrupts: B (7 ms every 100ms), C (1ms every 15 ms)

Can C fail to execute within 15 ms?



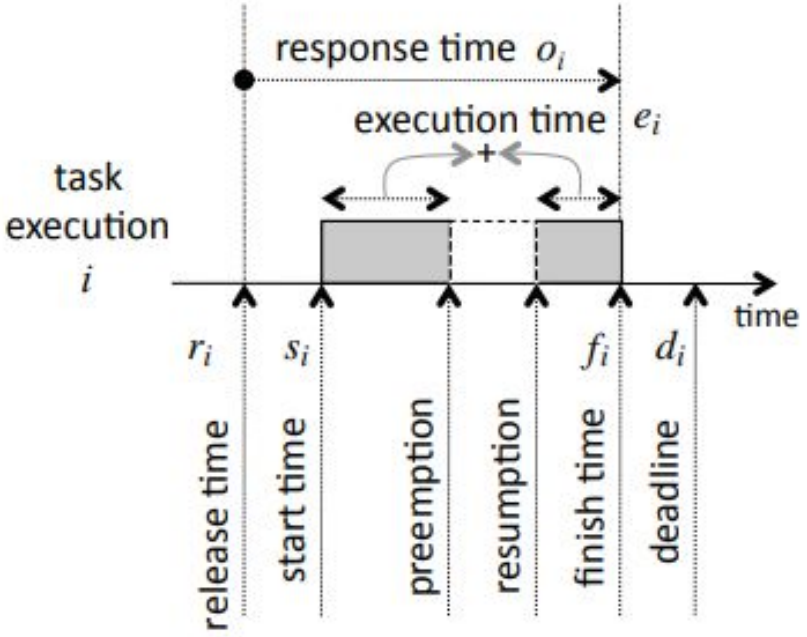
Different scheduling strategies

Static - figure it out ahead of time, CPU follows the set schedule

Dynamic:

Earliest deadline first (EDF)

Least laxity first (LLF) (**laxity** = d_i - e_i)



Feasibility of scheduling periodic tasks

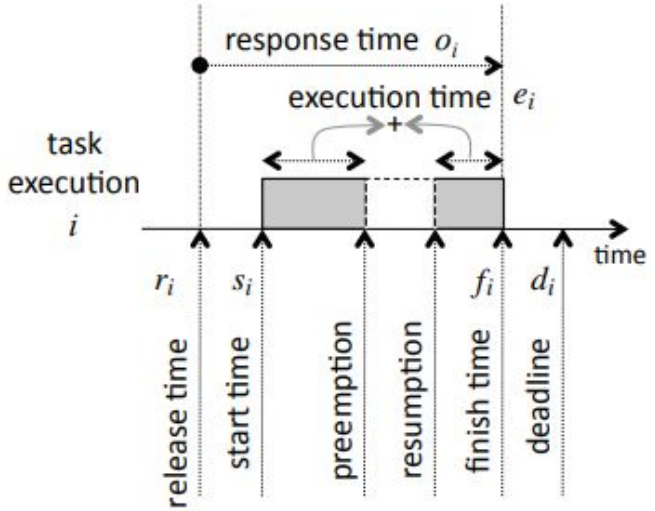
Feasibility: feasible if $f_i \leq d_i$ *for all i*

Utilization: % of time CPU spends executing tasks (vs idle)

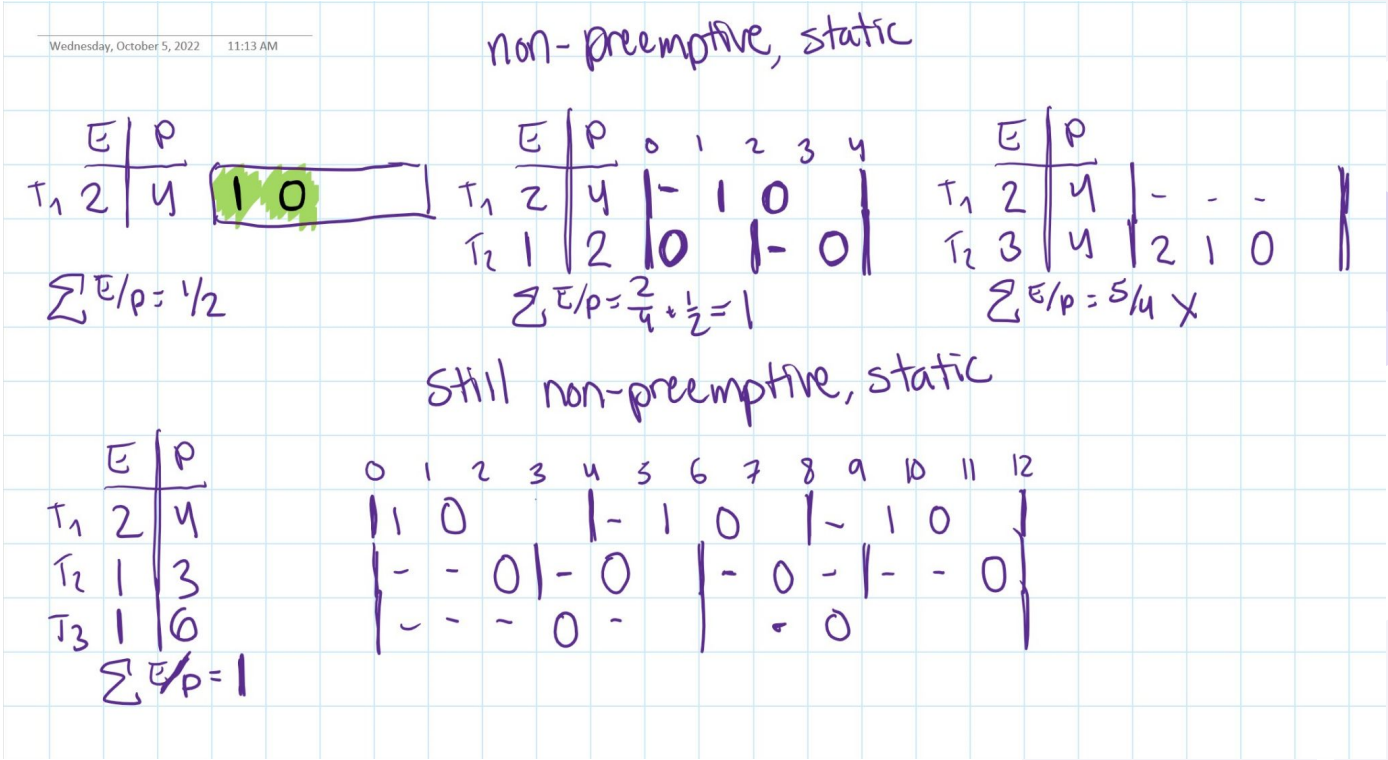
Necessary but not **Sufficient** condition for feasibility:

Sum of e_i/p_i (aka e_i/d_i) for all *i* is at most 1

Aka utilization <= 100%



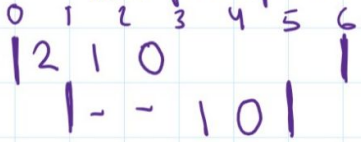
Scheduling examples on the board



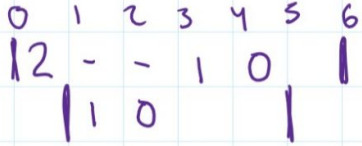
non-periodic EDF (dynamic)

	Release	Exec.	Deadline
T_1	0	3	6
T_2	1	2	5

non-preemptive:

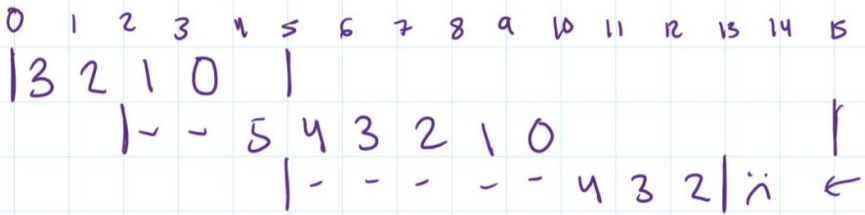


preemptive:



	Release	Exec.	Deadline
T_1	0	4	5
T_2	2	6	15
T_3	5	4	13

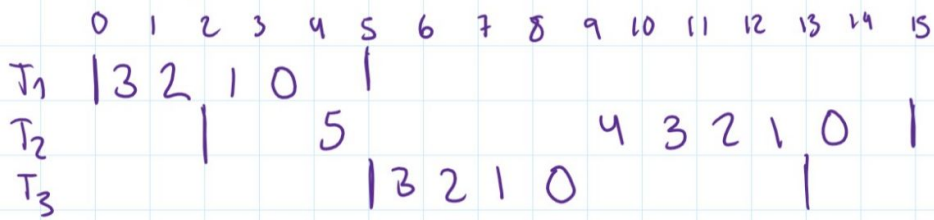
non-preemptive:



Note that we could have scheduled this statically so that the tasks meet their deadlines, but a dynamic scheduler may have no idea what tasks are arriving when and therefore relies on a rule like "earliest deadline first" to schedule

(still according to EDF)

preemptive:



Rate Monotonic Scheduling (RMS)

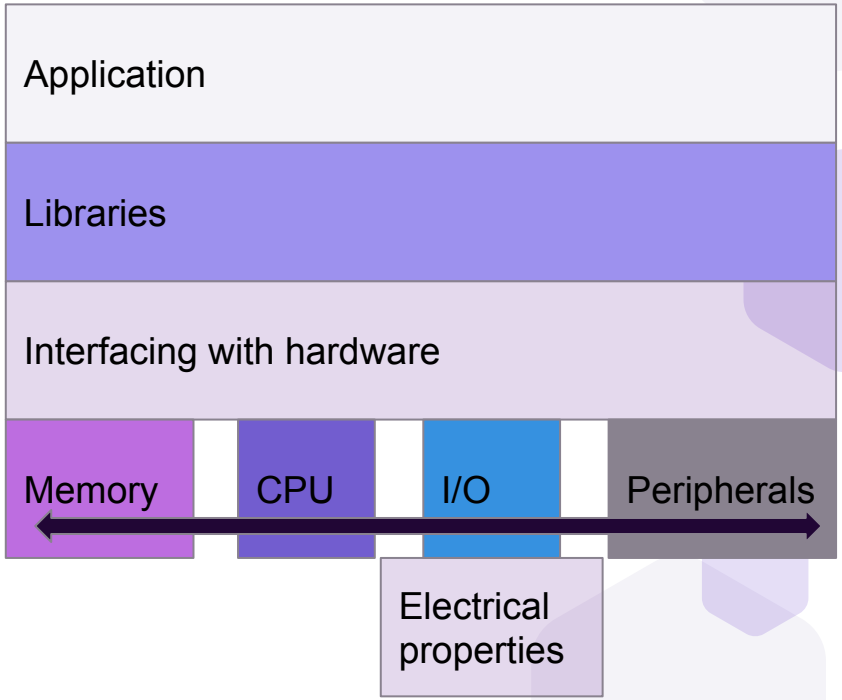
Fixed-priority, determined ahead of time

- Each task has its own priority
- Task with smallest period = highest priority
- Pre-emptive (higher priority tasks interrupt lower-priority tasks)

Guarantee of scheduling when utilization < 69.3%

$$\mu \leq n(2^{1/n} - 1), \qquad (12.2)$$

Stepping back – Embedded systems as systems

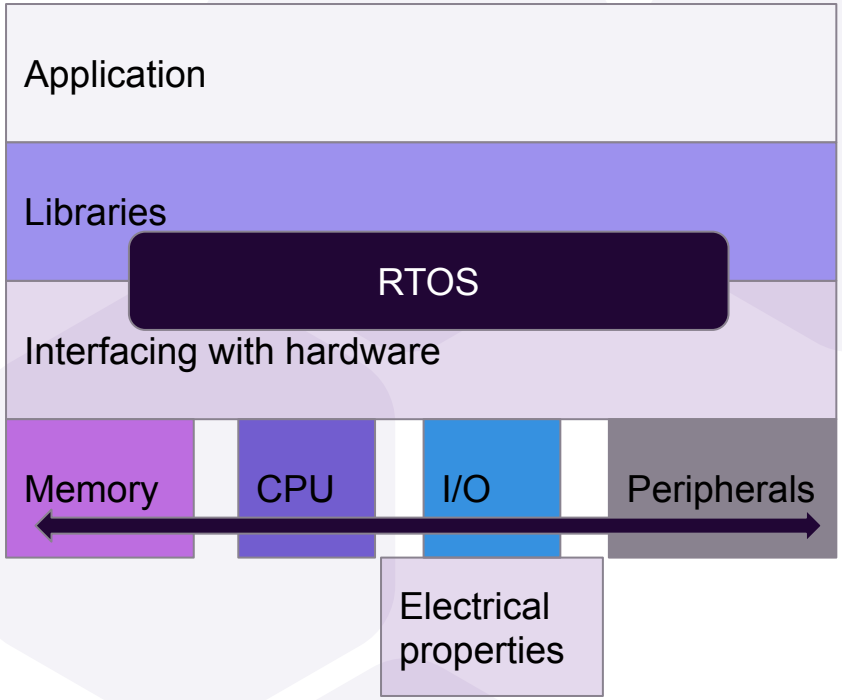


Real-Time Operating Systems

OS - manages system resources and provides services to programs/processes/threads

RTOS - an OS with real-time constraints

- Scheduling policies
- Often support for prioritization
- Libraries for mutexes/semaphores
- Memory management


Application
Libraries
RTOS
Interfacing with hardware
Memory
CPU
I/O
Peripherals
Electrical properties

Pros/cons to using an RTOS?

Would you want to write your own RTOS?

“Free” RTOS considerations

Expertise for being versed in RTOS use isn’t free

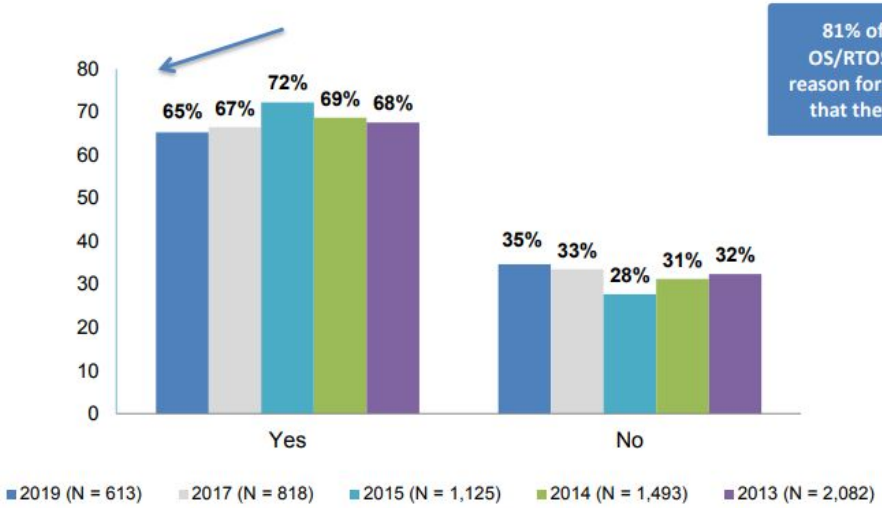
- Usually when you buy software you also buy support

Patching in updates isn’t free

Industry use of open-source is tricky

- License may require release of code

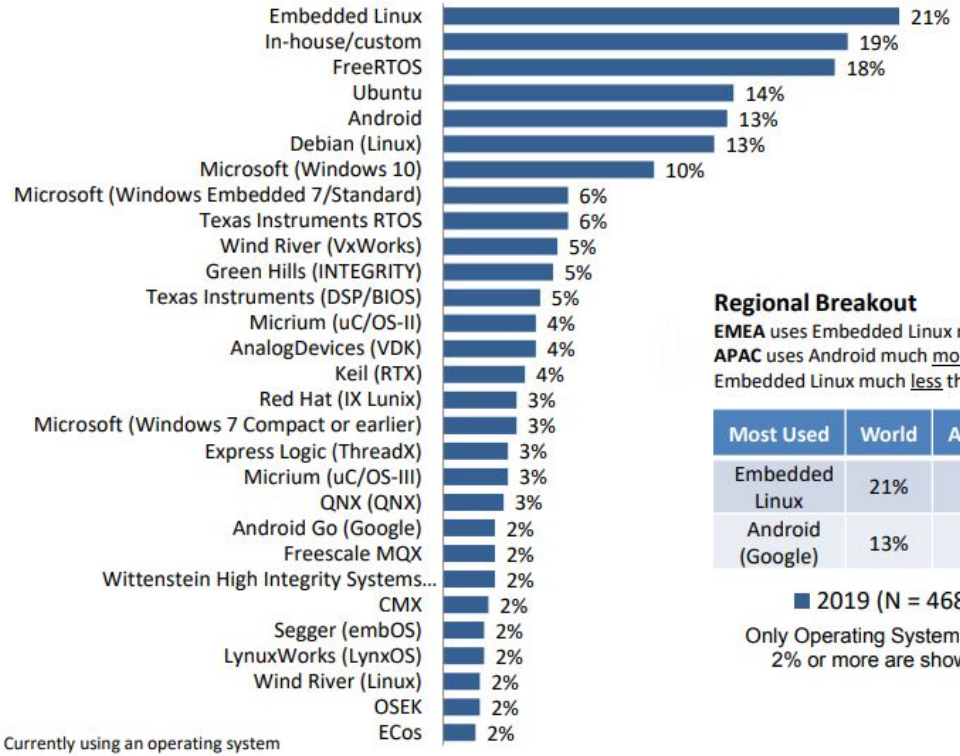
Does your current embedded project use an operating system, RTOS, kernel, software executive, or scheduler of any kind?



81% of those not using OS/RTOSes, said the main reason for NOT using is simply that they are not needed.

Image source

Please select ALL of the operating systems you are currently using.



Base: Currently using an operating system

Regional Breakout

EMEA uses Embedded Linux much more than other regions.
APAC uses Android much more than other regions and uses Embedded Linux much less that others.

Most Used	World	Americas	EMEA	APAC
Embedded Linux	21%	21%	30%	15%
Android (Google)	13%	9%	14%	27%

■ 2019 (N = 468)

Only Operating Systems with 2% or more are shown.

EETimes embedded

2019 Embedded Markets Study

Image source