# Operating Systems
# Lecture

## RTOS and microkernel

# Outline

- Real-time Operating Systems (RTOS)
  - Definition and scenarios of RTOS
  - Why Linux is not real-time?
  - Design and implementation of a RTOS
- Microkernel
  - Microkernel vs. Monolithic Kernel
  - Why Microkernel
  - Typical Microkernels and their designs

# What's RTOS?

- Real-time Operating Systems (RTOS)
  - RTOS:
    - ❑Definition: RTOS is a OS which can support Realtime tasks
    - ❑Realtime tasks[1]
      - Soft real-time tasks: if the deadline is missed, it's ok and may still have value
      - Hard real-time tasks: the deadline must not be missed, otherwise the system breaks
      - Firmed real-time tasks: if missed, the system will be fine, but the value is zero
    - ❑Categories
      - Soft real-time OS can support soft real-time tasks
      - Hard real-time OS can support hard real-time tasks
      - Firmed real-time OS can support firmed real-time tasks
    - ❑Features
      - Determinacy: the execute time of a function is determinate
      - Worst execution time: focus on the worst case but not the best/average situation
      - Performance: Not care about metrics like throughput, task execution time

[1] https://stackoverflow.com/questions/17308956/differences-between-hard-real-time-soft-real-time-and-firm-real-time

# What's the scenarios?

- Real-time Operating Systems (RTOS)
  - Scenarios
    - ❑Industry
    - ❑Medical instruments
    - ❑Flight
    - ❑Car
    - ❑Satellite


# The RTOS examples

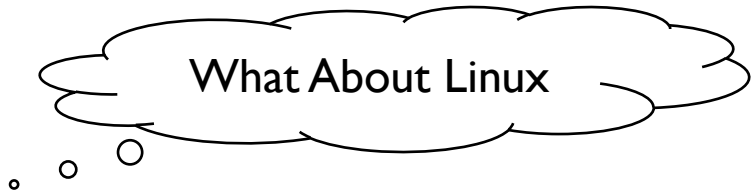
- Real-time Operating Systems (RTOS)
  - Examples
    - ❑Vxworks
    - ❑Uc/os-II/III
    - ❑FreeRTOS
    - ❑RTEMS
    - ❑Nutt
    - ❑QNX
    - ❑……





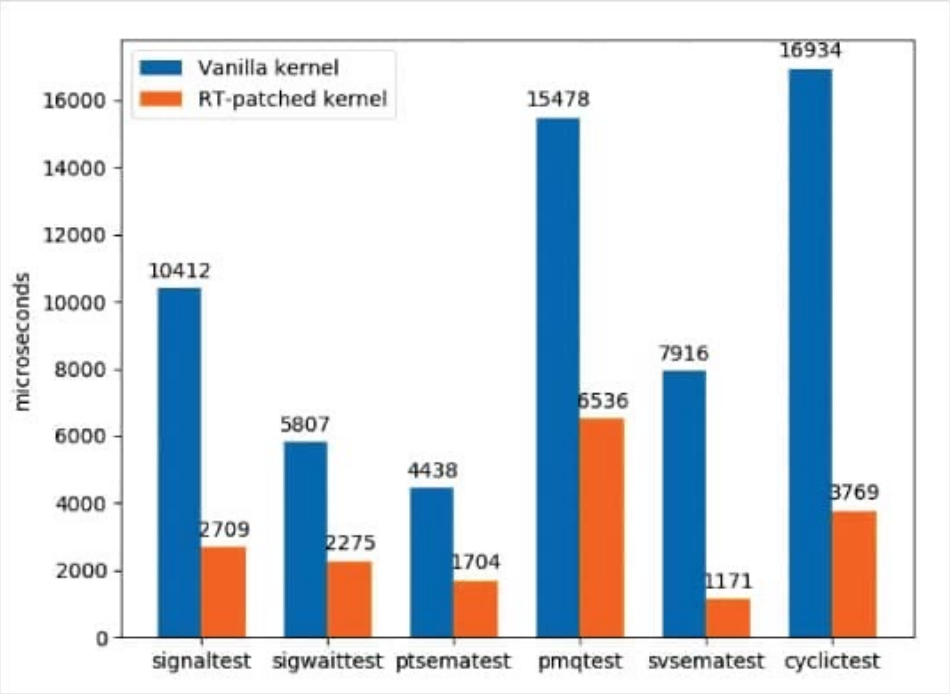
# The RTOS examples

- Real-time Operating Systems (RTOS)
  - Examples
    - ❑Vxworks
    - ❑Uc/os-II/III
    - ❑FreeRTOS
    - ❑RTEMS
    - ❑Nutt
    - ❑QNX
    - ❑……



# Is Linux realtime?

- Real-time Operating Systems (RTOS)
  - Linux is a General Purpose Operating System (GPOS)
  - GPOS VS RTOS
    - ❑GPOS focuses on finishing the tasks effective and efficient
    - ❑RTOS focuses on removing all the factors influencing the determinacy



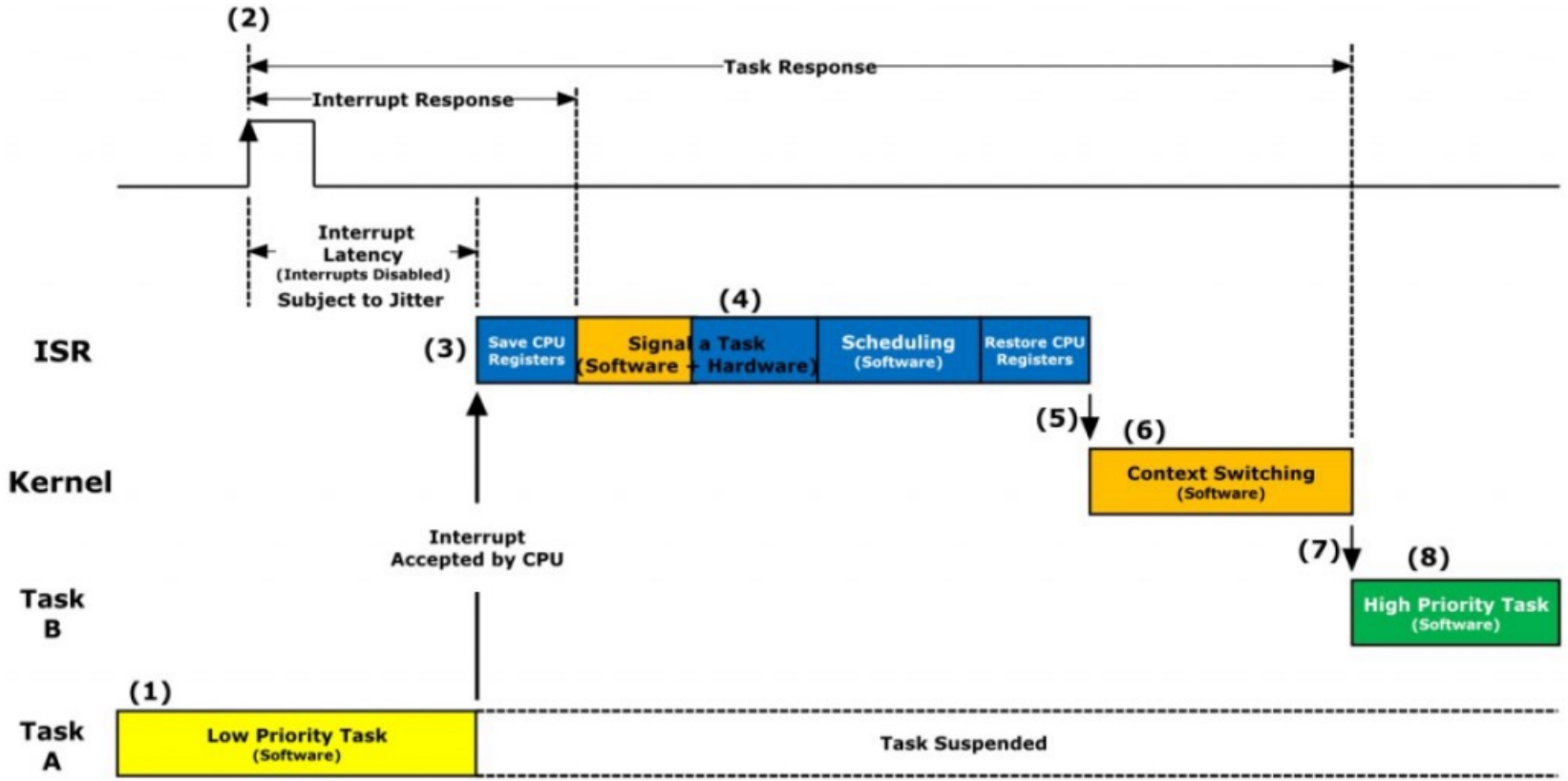

| Measured delay and jitter without rescheduling | | | | |
|---|---|---|---|---|
| | Linux | RTAI | Xenomai | VxWorks |
| Jitter(us) | 0.4 | 0.15 | 0.25 | 0.5 |
| Delay(us) | 72.8 | 71.8 | 73.2 | 69.2 |
| **Measured delay and jitter including rescheduling** | | | | |
| | Linux | RTAI | Xenomai | VxWorks |
| Jitter(us) | 1.5 | 0.4 | 0.45 | 1.5 |
| Rescheduling(us) | 5.6 | 0.2 | 2.7 | 1.2 |
| Delay(us) | 72.8 | 71.8 | 73.2 | 69.2 |
| **Measured delays with real-time network communication** | | | | |
| | Linux | RTAI | Xenomai | VxWorks |
| Jitter(us) | 11.1 | 3 | 3.3 | 20.4 |
| Delay(us) | 113 | 101 | 104.5 | 156.6 |

*Table 1*

# What makes Linux not realtime

- What's the key factor preventing Linux (or a GPOS) to be a RTOS

"Uncertainty"



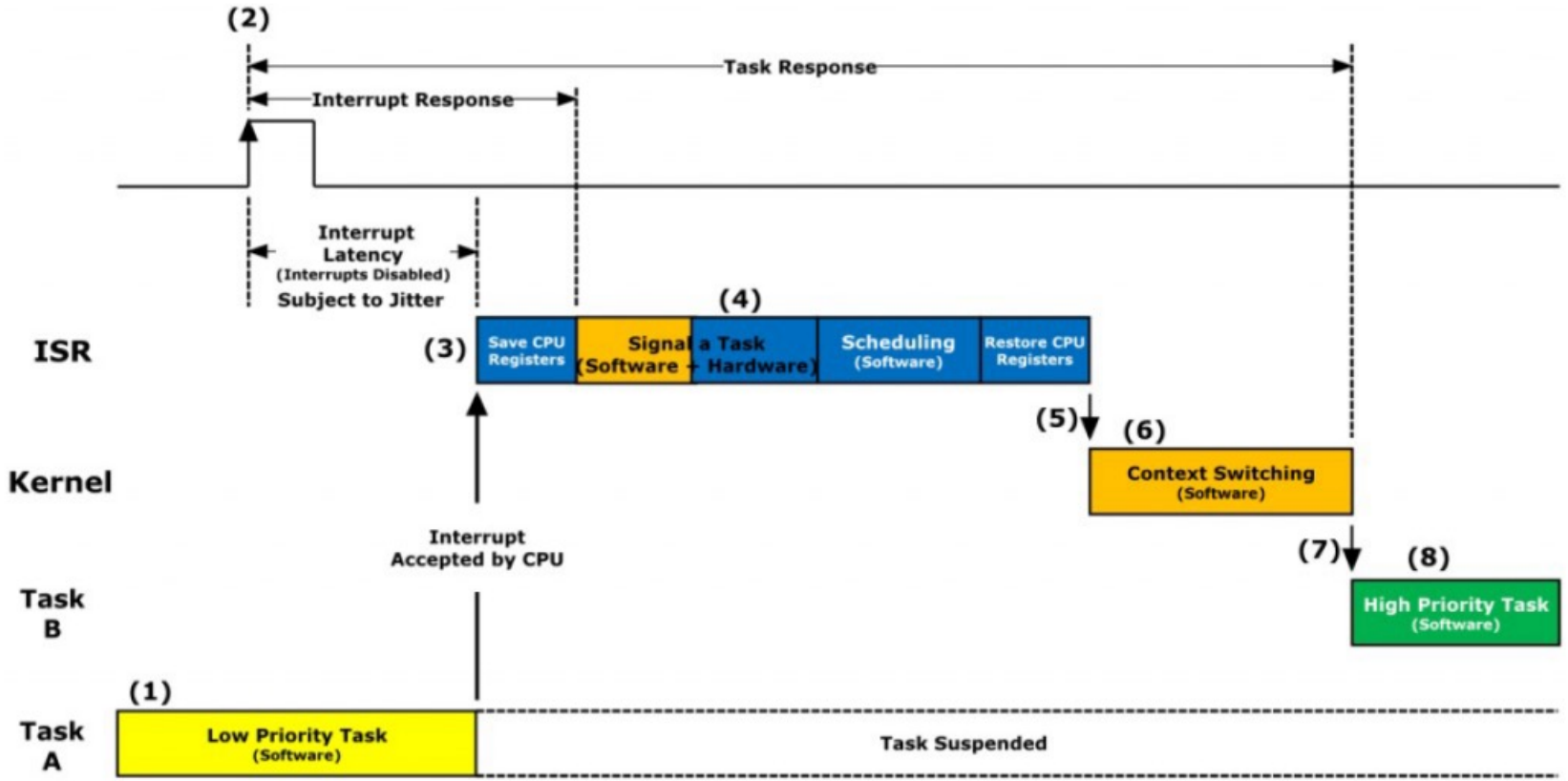


[1] https://www.eet-china.com/mp/a111433.html

[2] https://ntrs.nasa.gov/api/citations/20200002390/downloads/20200002390.pdf

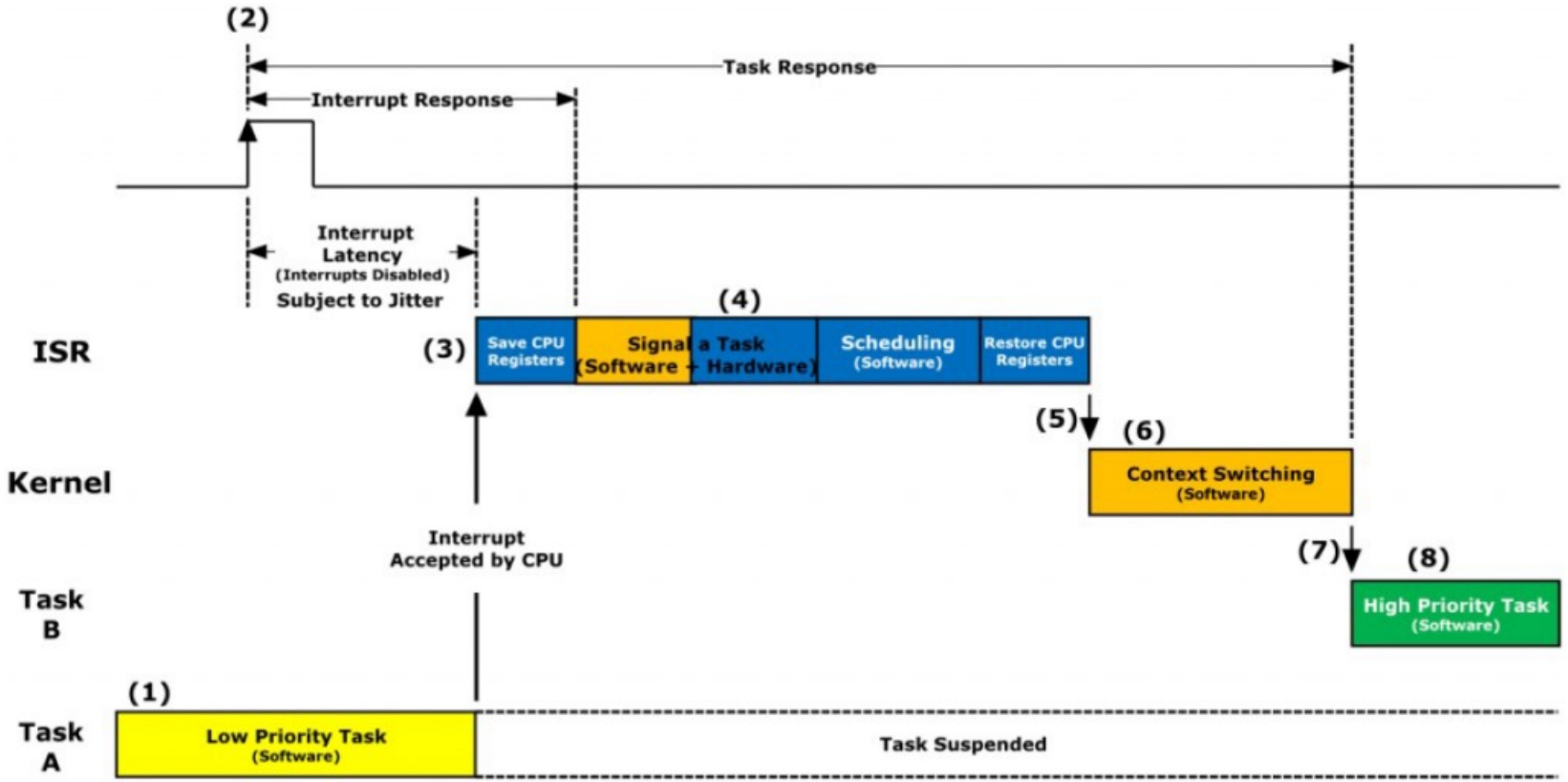
# What makes Linux not realtime

- What's the key factor preventing Linux (or a GPOS) to be a RTOS
  - Interrupt
    - ❑ Interrupt disabled (why?)
  - Scheduler
  - Sync
  - Timer
  - Memory



# What makes Linux not realtime

- What's the key factor preventing Linux (or a GPOS) to be a RTOS
  - Interrupt
  - Scheduler
    - ❑Linux is non-preemptible, which means it will execute one progress util it finishes
  - Sync
  - Timer
  - Memory



# What makes Linux not realtime

- What's the key factor preventing Linux (or a GPOS) to be a RTOS
  - Interrupt
  - Scheduler
  - Sync
    - ❑Spinlock prevents Interrupt
    - ❑Priority Inversion
  - Timer
  - Memory

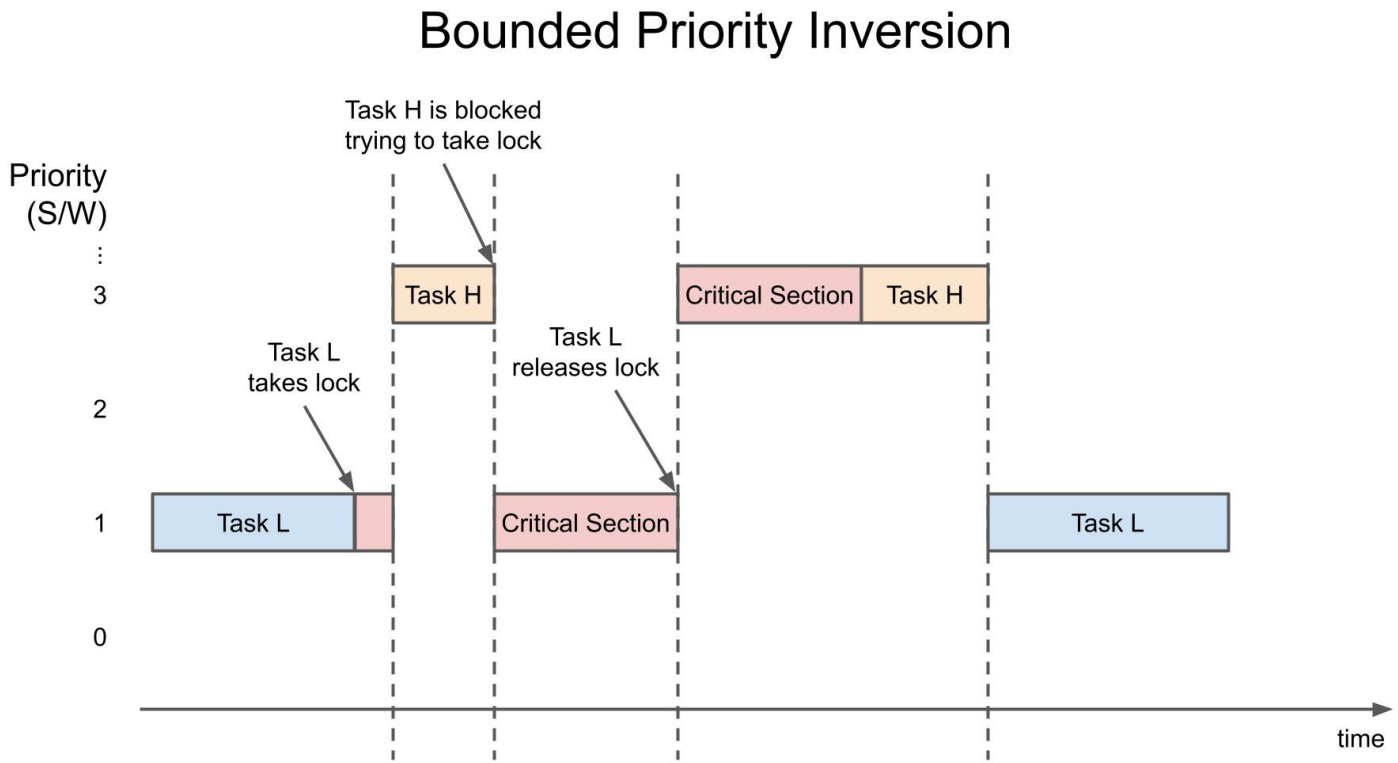
- Spinlock (**自旋锁**): another flawed, but simple solution:

```
int value = 0; // Free

Acquire() {
   while (test&set(value)); // while busy
}

Release() {
   value = 0;
}
```

- Simple explanation:
  - If lock is free, test&set reads 0 and sets value=1, so lock is now busy
    It returns 0 so while exits
  - If lock is busy, test&set reads 1 and sets value=1 (no change)
    It returns 1, so while loop continues
  - When we set value = 0, someone else can get lock
- Busy-Waiting: thread consumes cycles while waiting

# What makes Linux not realtime

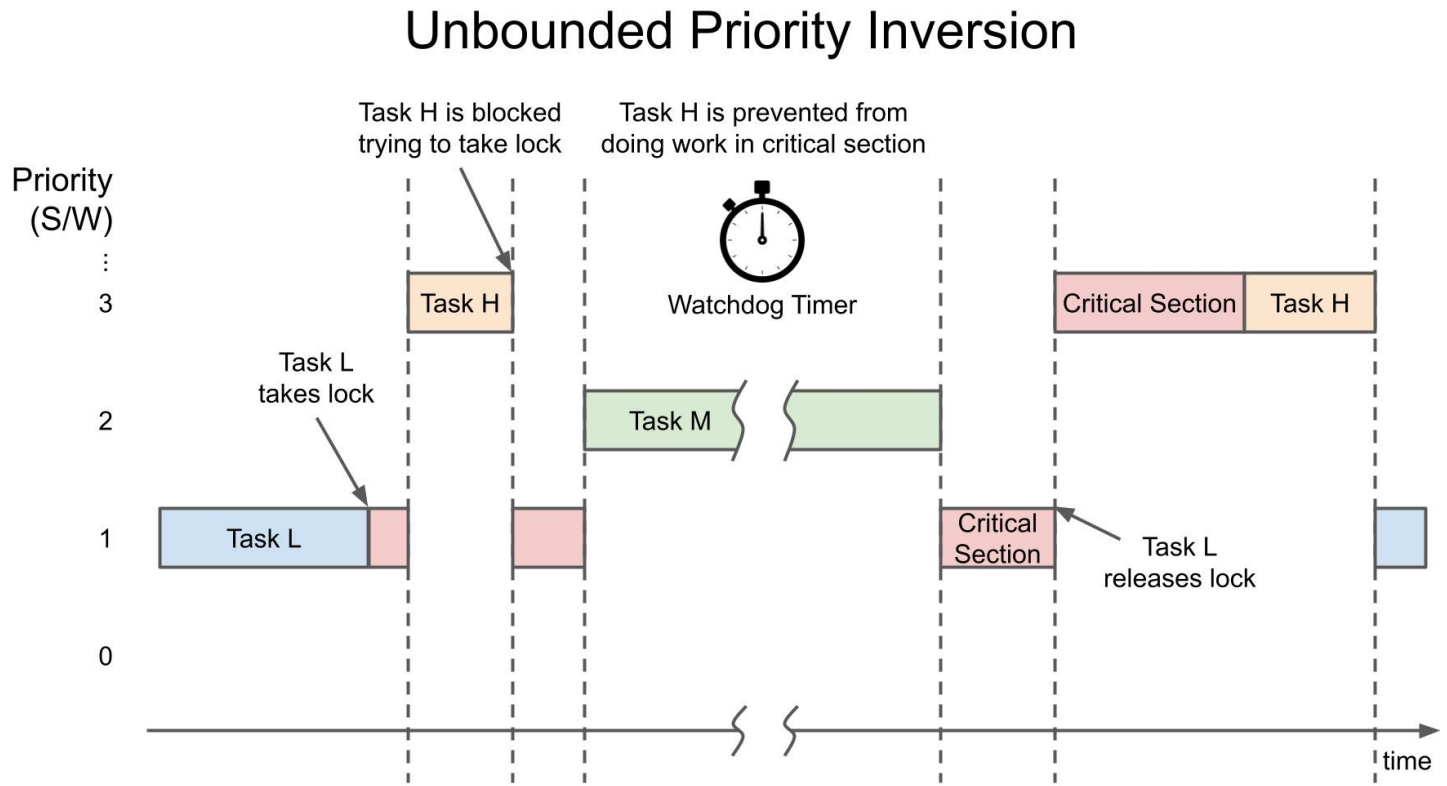
- What's the key factor preventing Linux (or a GPOS) to be a RTOS
  - Interrupt
  - Scheduler
  - Sync
    - ❑Spinlock prevents Interrupt
    - ❑Priority Inversion
  - Timer
  - Memory




[1] https://www.digikey.com/en/maker/projects/introduction-to-rtos-solution-to-part-11-priority-inversion/abf4b8f7cd4a4c70bece35678d178321

# What makes Linux not realtime

- What's the key factor preventing Linux (or a GPOS) to be a RTOS
  - Interrupt
  - Scheduler
  - Sync
    - ❑Spinlock prevents Interrupt
    - ❑Priority Inversion
  - Timer
  - Memory




[1] https://www.digikey.com/en/maker/projects/introduction-to-rtos-solution-to-part-11-priority-inversion/abf4b8f7cd4a4c70bece35678d178321

# What makes Linux not realtime

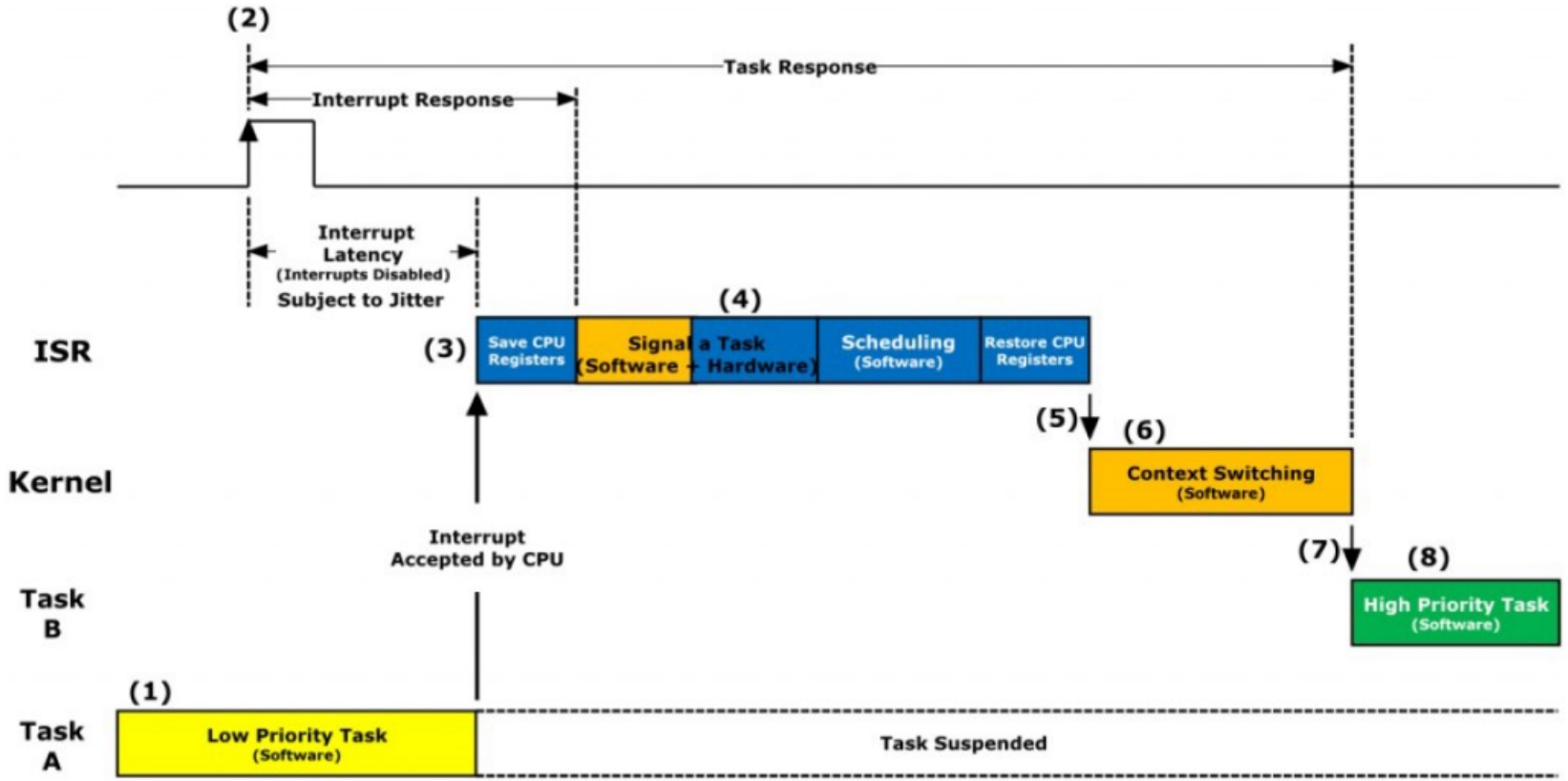
- What's the key factor preventing Linux (or a GPOS) to be a RTOS
  - Interrupt
  - Scheduler
  - Sync
  - Timer
    - ❑The timer resolution is low, which enlarge the latency
  - Memory

| Operating System | Typical Timer Interval | Notes |
|---|---|---|
| Windows (pre-Windows 8) | 15.6 ms | Older versions of Windows used a default tick rate of 64 Hz. |
| Windows (Windows 8 and later) | 0.5 ms to 15.6 ms | Uses a dynamic tick rate for improved energy efficiency. |
| Linux (Standard Kernel) | 4 ms to 10 ms | Default is 250 Hz (4 ms) or 100 Hz (10 ms), but it's configurable. |
| Linux (Real-Time Kernel) | 1 ms or lower | Real-time kernels may have higher frequencies for better responsiveness. |
| macOS | 10 ms | macOS typically uses a 100 Hz tick rate. |
| UNIX (General) | 10 ms to 100 ms | Varies significantly with the specific UNIX variant and configuration. |

```
❯ sysctl -a | grep hz
kern.clockrate: { hz = 100, tick = 10000, tickadj = 0, profhz = 1
00, stathz = 100 }
```

# What makes Linux not realtime

- What's the key factor preventing Linux (or a GPOS) to be a RTOS
  - Interrupt
  - Scheduler
  - Sync
  - Timer
  - Memory
    - ❑Soft/hard Page fault
    - ❑TLB/Cache miss
    - ❑Dynamic alloc



# Two ways to make RTOS

1. Write a RTOS from scratch

2. Make GPOS (like Linux) realtime

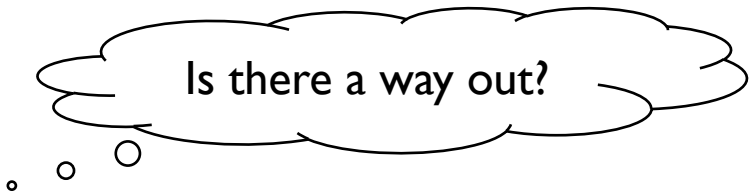
# Why make Linux real-time

- Real-time Operating Systems (RTOS)
  - There are some scenarios where we need both real-time and generalization
  - SpaceX recycled rocket [1]
    - ❑The control algorithm of Falcon 9 rocket
    - ❑Need to be computed in 100ms
    - ❑The algorithm is complex and need Convexification Toolbox which is not provided in the RTOS

[1] SpaceX火箭姿控算法：Lossless Convexification of Nonconvex Control Bound and Pointing Constraints of the Soft Landing Optimal Control Problem

# Why make Linux real-time

- Real-time Operating Systems (RTOS)
  - There are some scenarios where we need both real-time and generalization
  - SpaceX recycled rocket [1]
    - ❑The control algorithm of Falcon 9 rocket
    - ❑Need to be computed in 100ms
    - ❑The algorithm is complex and need Convexification Toolbox which is not provided in the RTOS



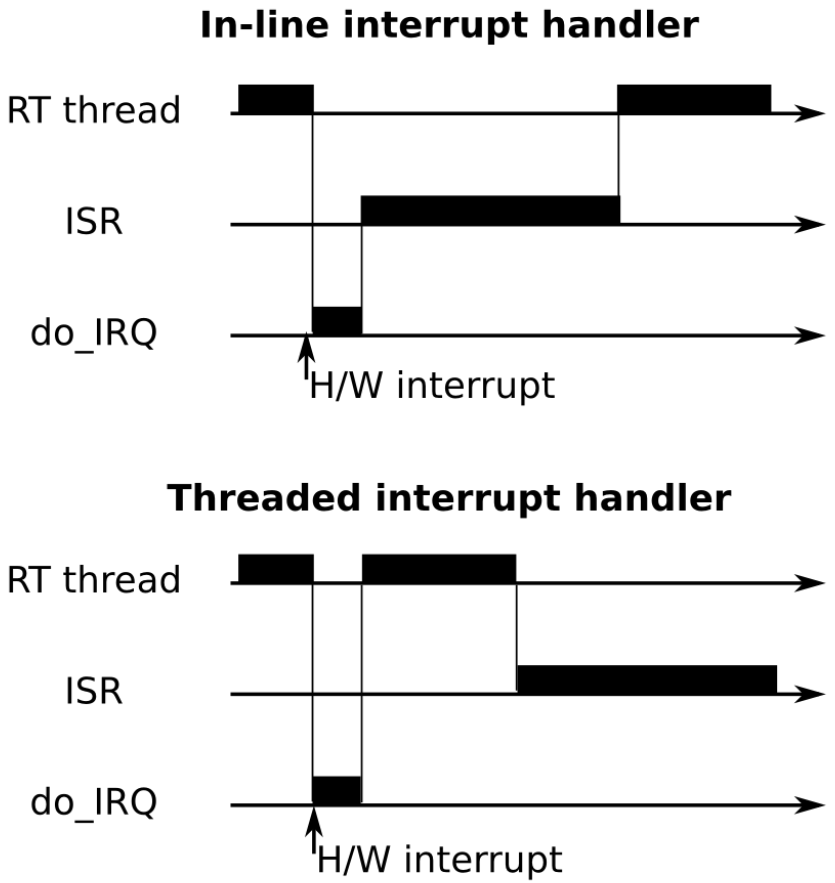

[1] SpaceX火箭姿控算法：Lossless Convexification of Nonconvex Control Bound and Pointing Constraints of the Soft Landing Optimal Control Problem

# How to make Linux real-time

- Real-time Operating Systems (RTOS)
  - Some ways
    - ❑Preempt_RT
    - ❑Dual-kernel
      - RTLinux
      - Xenomai
      - RTAI
      - RROS
    - ❑Hypervisor: RTOS+Linux

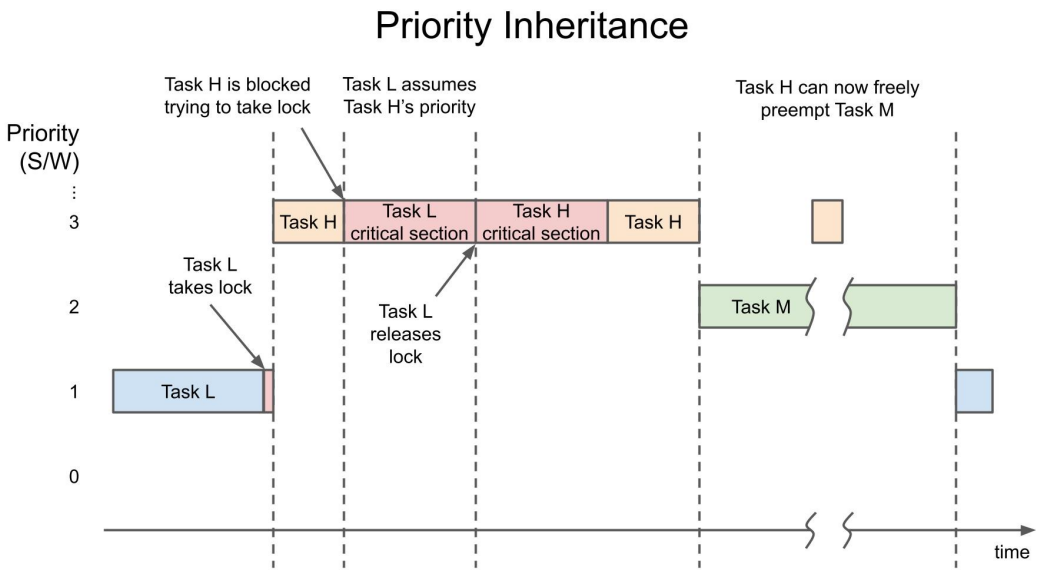
# Preempt_RT

- Real-time Operating Systems (RTOS)
  - Preempt_RT
    - ❑Modify all the elements that are prevent Linux to be a RTOS
    - ❑Interrupt
      - Interrupt disabled
      - T1: Reduce critical zone that disables the irq
      - T2: Split isr into the half-top and half-bottom (interrupt threading)
    - ❑Scheduler
    - ❑Sync
    - ❑Timer
    - ❑Memory



# Preempt_RT

- Real-time Operating Systems (RTOS)
  - Preempt_RT
    - ❑Modify all the elements that are prevent Linux to be a RTOS
    - ❑Interrupt
    - ❑Scheduler
      - Linux is non-preemptible, which means it will execute one progress util it finishes
      - T1: Add more preempted points (yield),
      - priority reverse
      - T2: priority inherit
    - ❑Sync
    - ❑Timer
    - ❑Memory



# Preempt_RT

- Real-time Operating Systems (RTOS)
  - Preempt_RT
    - ❑Modify all the elements that are prevent Linux to be a RTOS
    - ❑Interrupt
    - ❑Scheduler
    - ❑Sync
      - Spinlock prevents interrupt
      - T1: make the spinlock preemptible (How?)
    - ❑Timer
    - ❑Memory

# Preempt_RT

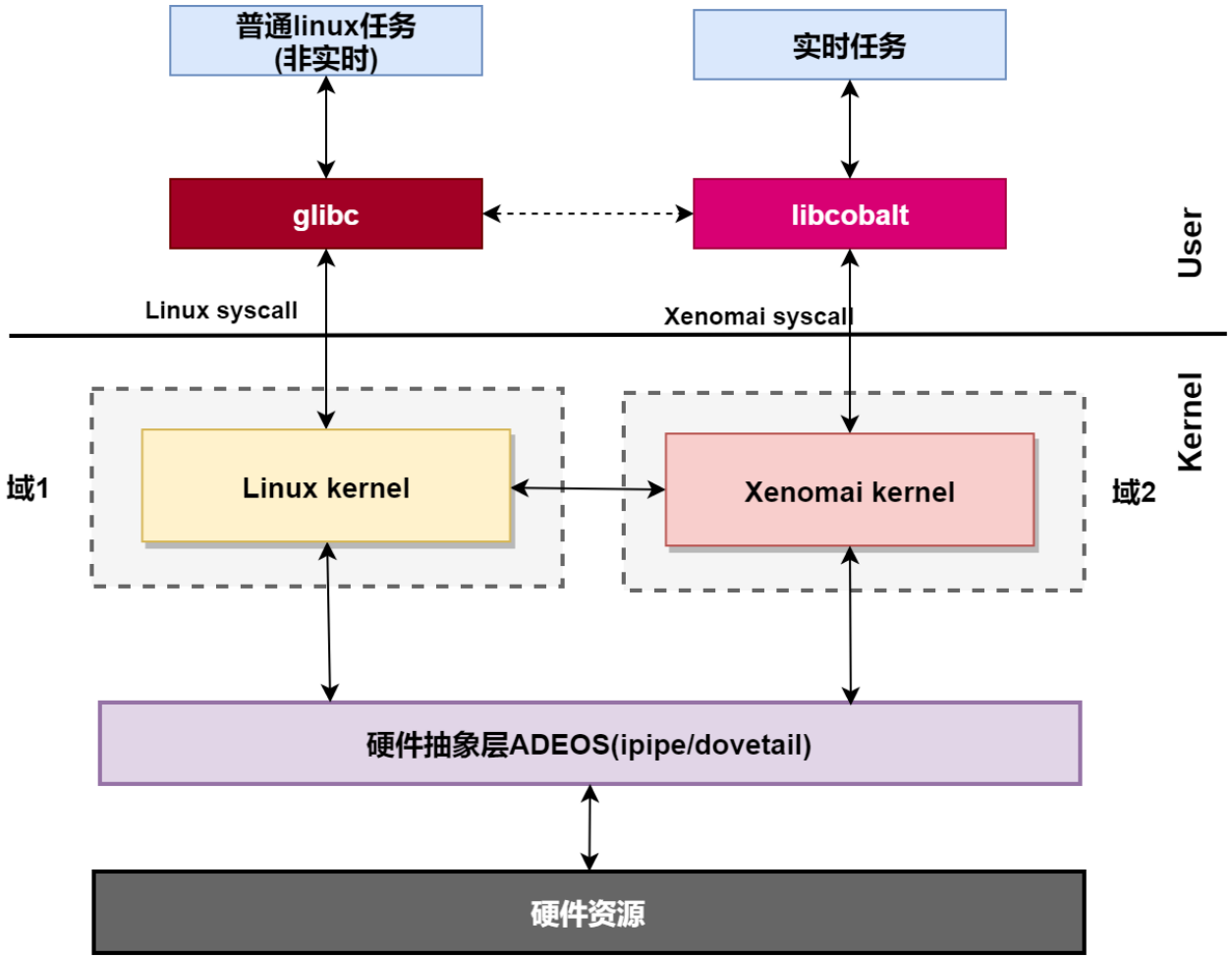
- Real-time Operating Systems (RTOS)
  - Some ways
    - ❑Preempt_RT
      - Modify all the elements that are prevent Linux to be a RTOS
      - Interrupt
      - Scheduler
      - Sync
      - Timer
        - The timer resolution is low, which enlarge the latency
        - T1: High-Resolution Timers (hrtimers)
        - T2: no_hz Model (Tickless Kernel)
      - Memory

# Preempt_RT

- Real-time Operating Systems (RTOS)
  - Preempt_RT
    - ❑Modify all the elements that are prevent Linux to be a RTOS
    - ❑Interrupt
    - ❑Scheduler
    - ❑Sync
    - ❑Timer
    - ❑Memory
      - Soft/hard Page fault
      - T1: alloc the memory in the kernel space
      - TLB/Cache miss
      - T2: no
      - Dynamic alloc
      - T3: pre alloc

# Dual-kernel

- Real-time Operating Systems (RTOS)
  - Dual-kernel [1]
    - ❑Interrupt virtualization layer
      - Dovetail distributes the interrupt
    - ❑Realtime kernel
      - A real-time kernel
    - ❑General kernel
      - Linux kernel
    - ❑The real-time kernel can handle the interrupt prior to the general kernel




[1] https://www.cnblogs.com/wsg1100/p/12833126.html

# Write your own RTOS--design

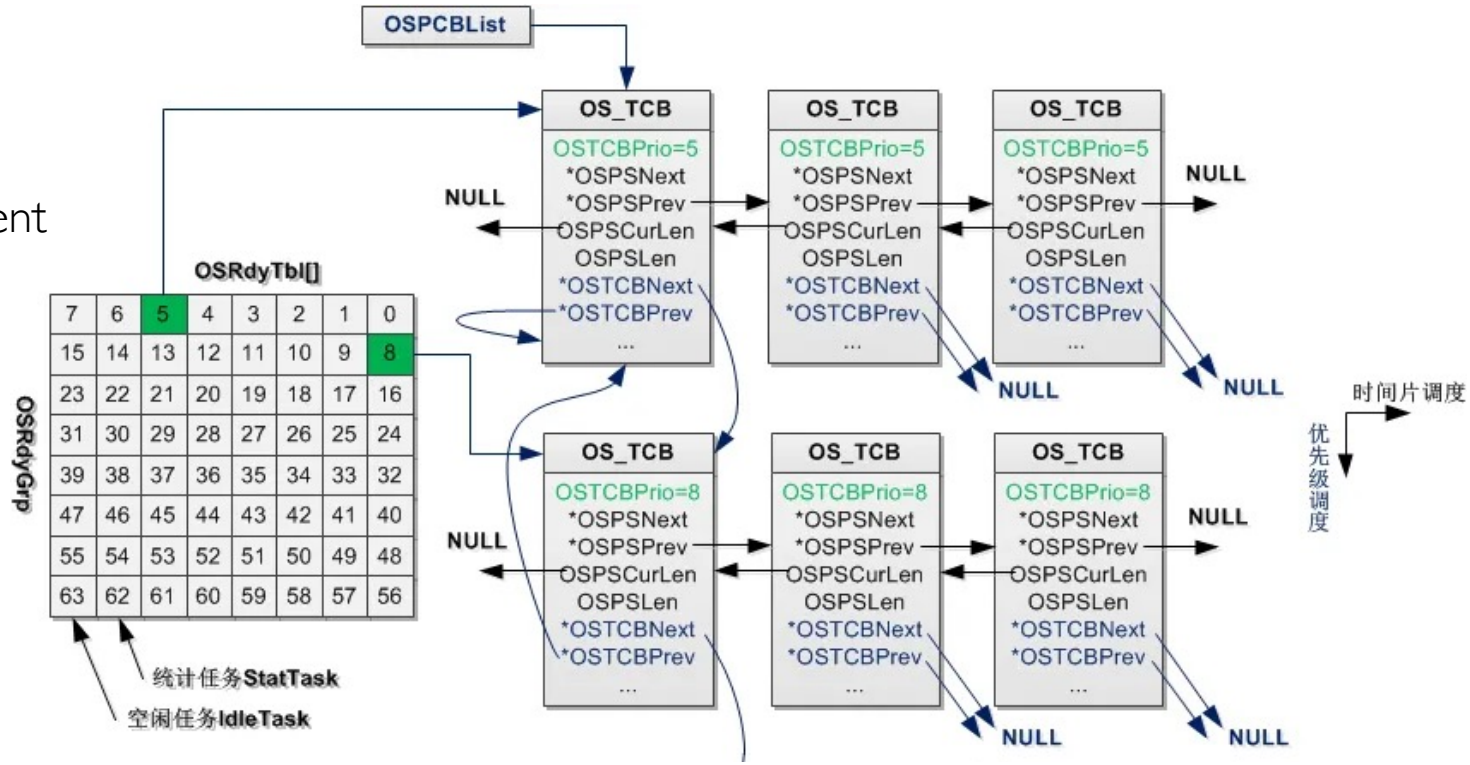
- Real-time Operating Systems (RTOS)
  - How to write a RTOS
    - ❑ Schedule
      - FIFO
      - RR
    - ❑ Sync
      - priority inverse: priority ceil, priority inherit
    - ❑ Interrupt
      - Less critical zone
      - More preempted points
    - ❑ Timer
      - High resolution
      - Less tick
    - ❑ Memory
      - mmu
        - No MMU
        - MMU: soft/hard Page fault, cache/tlb miss
      - Dynamic memory allocation
        - Pre alloc memory
    - ❑ Network
      - TSN

# Write your own RTOS

- Real-time Operating Systems (RTOS)
  - How to write a RTOS
    - ❑Print
      - Print can alloc memory
      - Dynamically alloc memory may introduce latency
      - It's unacceptable to use print in the realtime
      - Make print in a non realtime thread **but this may lose information**
      - **Print is a trick function in all the RTOSes**

# Write your own RTOS

- Real-time Operating Systems (RTOS)
  - How to write a RTOS (Based on the uc/os)
    - ❑Schedule
      - FIFO+multi priorities
      - The threads on the same priority use FIFO, on the different priorities use multi priorities




[1] https://blog.51cto.com/u_15064630/4649712

# Write your own RTOS

- Real-time Operating Systems (RTOS)
  - How to write a RTOS
    - ❑Interrupt
      - Make the interrupt handler nested to decrease the latency time[1]
      - Supports up to 250 nested interrupts

os_core.c

```
263 *********************************************************************************************************
264 */
265
266 void  OSIntEnter (void)
267 {
268     if (OSRunning != OS_STATE_OS_RUNNING) {              /* Is OS running? 判断UCOSIII是否运行
269         return;                                          /* No
270     }
271
272     if (OSIntNestingCtr >= (OS_NESTING_CTR)250u) {       /* Have we nested past 250 levels?
273         return;                                          /* Yes判断中断嵌套次数是否大于250
274     }
275
276     OSIntNestingCtr++;                                   /* Increment ISR nesting level
277 }                                                            中断次数加一
278
```

CSDN @特轮飞

```
void USART1_IRQHandler(void)                    //串口1中断服务程序
{
    uint8_t d=0;

    //进入中断
    OSIntEnter();

    if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)   //接收中断
    {
        //接收串口数据
        d=USART_ReceiveData(USART1);

        //清空串口接收中断标志位
        USART_ClearITPendingBit(USART1, USART_IT_RXNE);
    }

    //退出中断
    OSIntExit();
```

CSDN @Alnert

[1] https://blog.csdn.net/ttdffmm/article/details/133811245

# Write your own RTOS

- Real-time Operating Systems (RTOS)
  - How to write a RTOS
    - ❑Interrupt
      - Make the interrupt handler nested to decrease the latency time
      - Provide the interface to disable the irq[1]
        - The critical area should be minimal

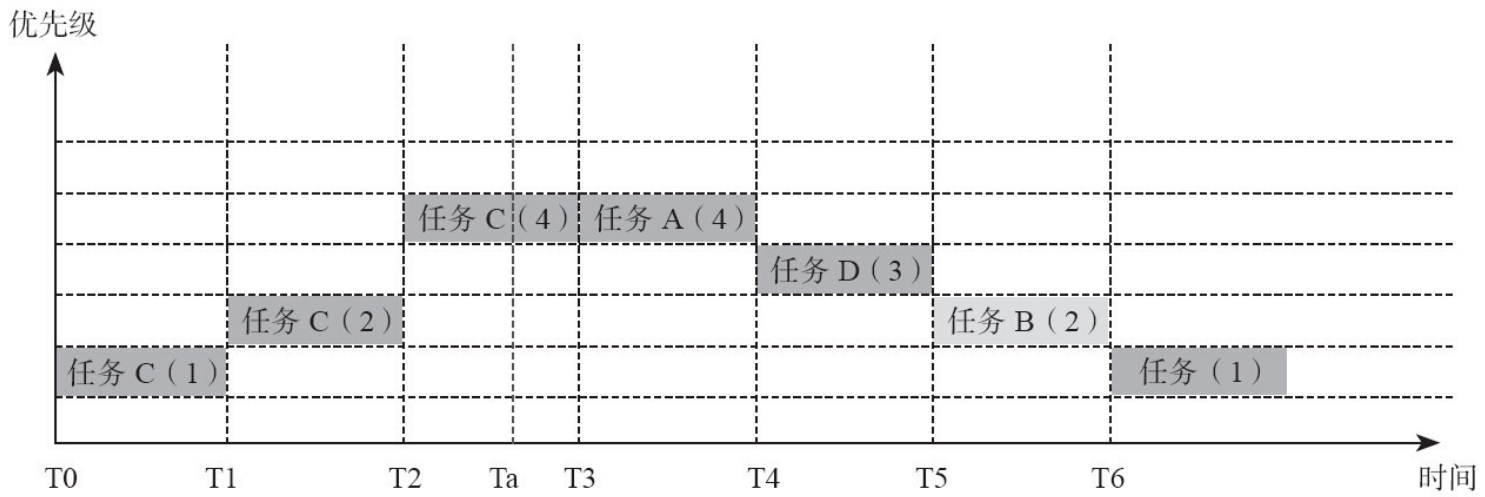
```
void task1(void *parg)
{
    OS_ERR err;

    CPU_SR_ALLOC();

    printf("task1 is create ok\r\n");

    while(1)
    {
        //进入临界区，关中断，其他任务都停止，独占共享资源
        OS_CRITICAL_ENTER();

        printf("task1 is running ...\r\n");

        delay();

        //............


        //............

        //立即退出临界区，开中断，允许任务调度，释放共享资源
        OS_CRITICAL_EXIT();

        delay_ms(1000);

    }
}
```

CSDN @Alnert

[1] https://blog.csdn.net/weixin_72589081/article/details/129746305

# Write your own RTOS

- Real-time Operating Systems (RTOS)
  - How to write a RTOS
    - ❑Sync
      - priority inverse
      - priority inherit[1]




[1]https://currenwong.github.io/Mkdoc/%E8%AF%BE%E7%A8%8B%E7%AC%94%E8%AE%B0/%E5%B5%8C%E5%85%A5%E5%BC%8F%E7%B3%BB%E7%BB%9F/2.%E4%BC%98%E5%85%88%E7%BA%A7%E5%8F%8D%E8%BD%AC%E4%B8%8E%E4%B8%AD%E6%96%AD%E6%9C%BA%E5%88%B6/#21

# Write your own RTOS

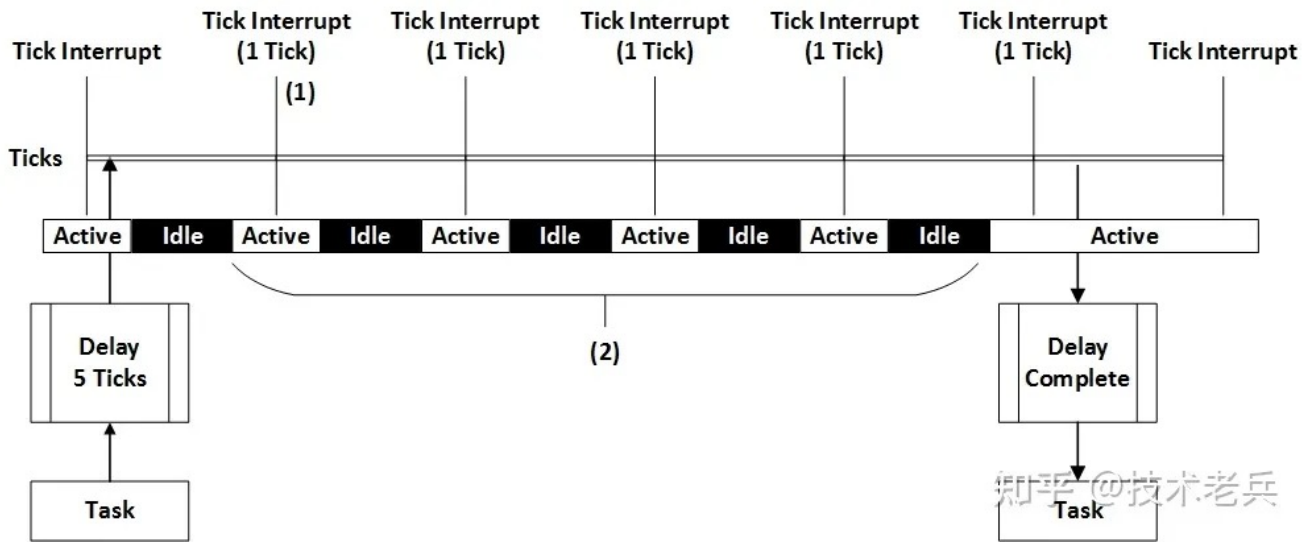
- Real-time Operating Sys
  - How to write a RTOS
    - ❑Sync
      - priority inverse
      - priority inherit[1]

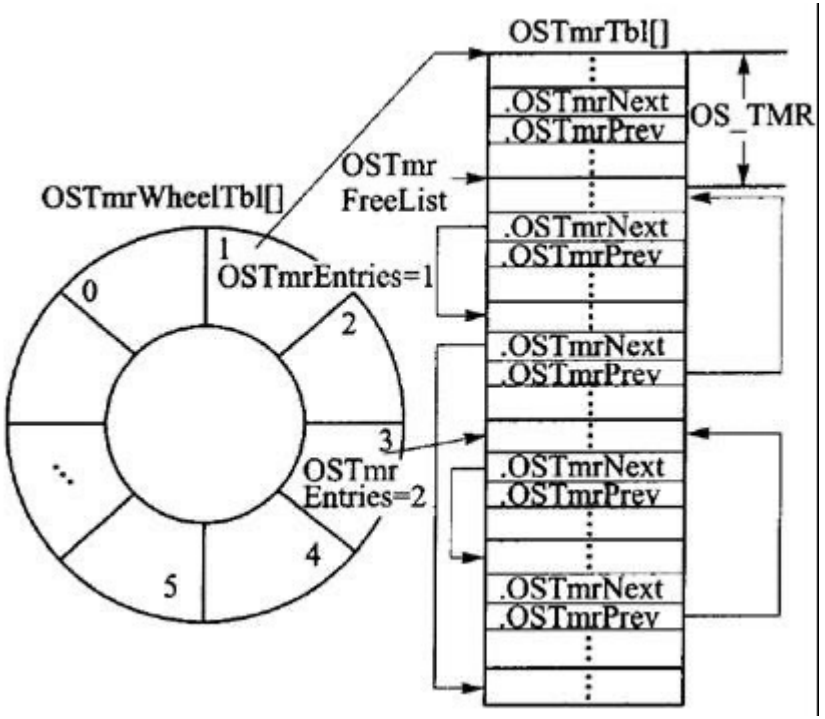
```
if (pcp != OS_PRIO_MUTEX_CEIL_DIS) {
    mprio = (INT8U)(pevent->OSEventCnt & OS_MUTEX_KEEP_LOWER_8); /*  Get priority of mutex owner   */
    ptcb  = (OS_TCB *)(pevent->OSEventPtr);                      /*     Point to TCB of mutex owner   */
    if (ptcb->OSTCBPrio > pcp) {                                 /*     Need to promote prio of owner?*/
        if (mprio > OSTCBCur->OSTCBPrio) {
            y = ptcb->OSTCBY;
            if ((OSRdyTbl[y] & ptcb->OSTCBBitX) != 0u) {         /*     See if mutex owner is ready   */
                OSRdyTbl[y] &= (OS_PRIO)~ptcb->OSTCBBitX;        /*     Yes, Remove owner from Rdy ...*/
                if (OSRdyTbl[y] == 0u) {                         /*          ... list at current prio */
                    OSRdyGrp &= (OS_PRIO)~ptcb->OSTCBBitY;
                }
                rdy = OS_TRUE;
            } else {
                pevent2 = ptcb->OSTCBEventPtr;
                if (pevent2 != (OS_EVENT *)0) {                  /* Remove from event wait list      */
                    y = ptcb->OSTCBY;
                    pevent2->OSEventTbl[y] &= (OS_PRIO)~ptcb->OSTCBBitX;
                    if (pevent2->OSEventTbl[y] == 0u) {
                        pevent2->OSEventGrp &= (OS_PRIO)~ptcb->OSTCBBitY;
                    }
                }
                rdy = OS_FALSE;                                  /* No                               */
            }
            ptcb->OSTCBPrio = pcp;                               /* Change owner task prio to PCP    */
```

[1] https://github.com/jcdubois/uCOS-II/blob/master/uCOS-II/Source/os_mutex.c#L469

# Write your own RTOS

- Real-time Operating Systems (RTOS)
  - How to write a RTOS
    - ❑Timer
      - Resolution is according to the HZ of the timer device
      - Timer control
        - Software
          - **Period (best resolution is the tick)**

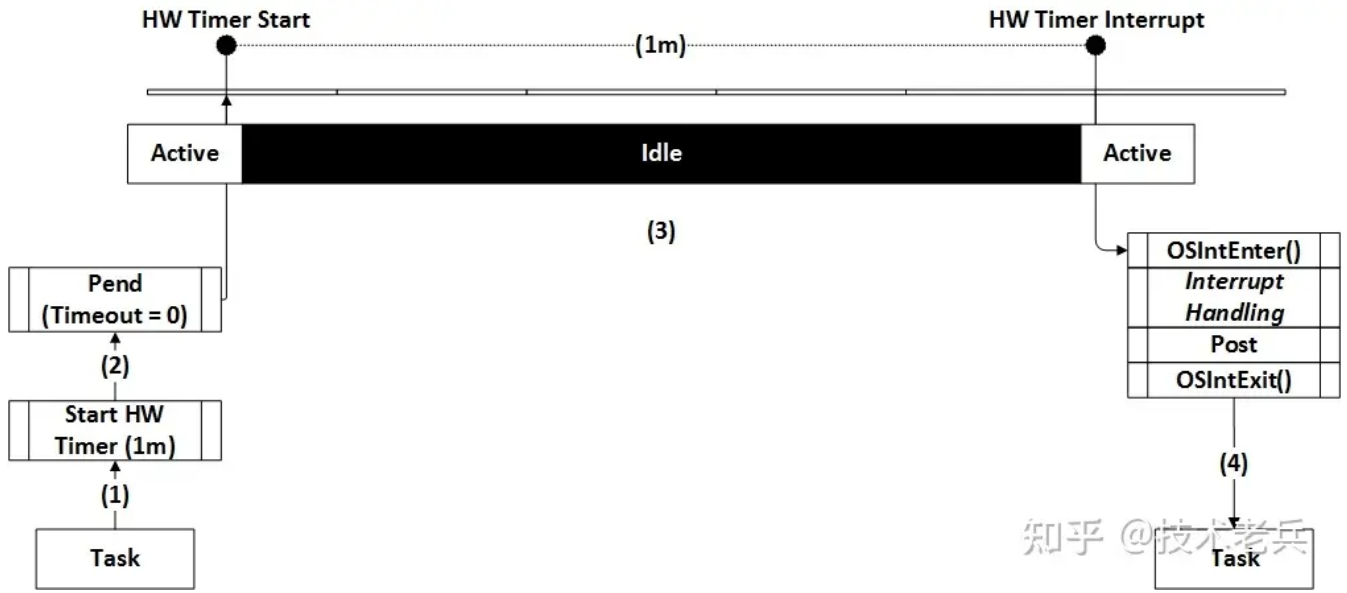



[1] https://zhuanlan.zhihu.com/p/461831300?utm_id=0

# Write your own RTOS

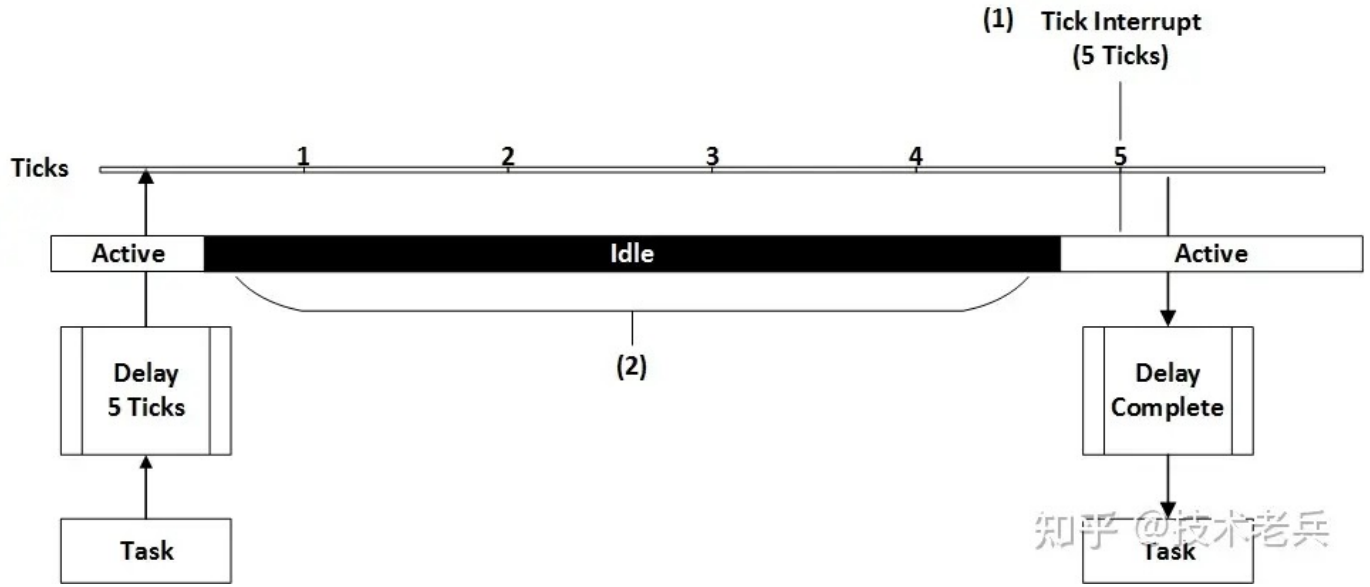
- Real-time Operating Systems (RTOS)
  - How to write a RTOS
    - ❑Timer
      - Resolution is according to the HZ of the timer device
      - Timer control
        - Hardware (best resolution is up to the device)
          - **No_tick**
          - **Dyn_tick**




[1] https://zhuanlan.zhihu.com/p/461831300?utm_id=0

# Write your own RTOS

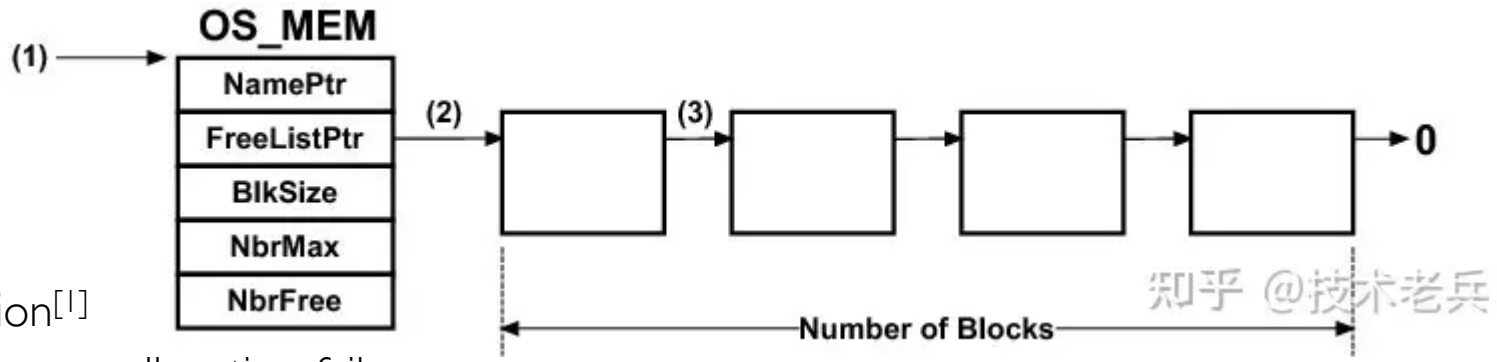
- Real-time Operating Systems (RTOS)
  - How to write a RTOS
    - ❑Timer
      - Resolution is according to the HZ of the timer device
      - Timer control
        - Hardware (best resolution is up to the device)
          - **No_tick**
          - **Dyn_tick**




[1] https://zhuanlan.zhihu.com/p/461831300?utm_id=0

# Write your own RTOS

- Real-time Operating Systems (RTOS)
  - How to write a RTOS
    - ❑Memory
      - MMU
        - no MMU
      - Dynamic memory allocation[1]
        - Let user to avoid memory allocation fail
      - Memory allocation algorithm[2, 3]
        - O(1)





[1] http://zhangzhenyuan163.blog.163.com/blog/static/858193892011518025249 41/

[2] https://zhuanlan.zhihu.com/p/465108575?utm_id=0

[3] https://github.com/aleen42/ucos/blob/master/ucos/OS_MEM.c

# Write your own RTOS

- Real-time Operating S
  - How to write a RTOS
    - ❑Memory
      - Mmu
        - no mmu
      - Dynamic memory a
        - Let user to avo
      - Memory allocation
        - O(1)

```
void  *OSMemGet (OS_MEM *pmem, INT8U *err)
{
#if OS_CRITICAL_METHOD == 3                          /* Allocate storage for CPU status register      */
    OS_CPU_SR  cpu_sr;
#endif
    void      *pblk;


#if OS_ARG_CHK_EN > 0
    if (pmem == (OS_MEM *)0) {                       /* Must point to a valid memory partition        */
        *err = OS_MEM_INVALID_PMEM;
        return ((OS_MEM *)0);
    }
#endif
    OS_ENTER_CRITICAL();
    if (pmem->OSMemNFree > 0) {                      /* See if there are any free memory blocks       */
        pblk                = pmem->OSMemFreeList;   /* Yes, point to next free memory block          */
        pmem->OSMemFreeList = *(void **)pblk;        /*      Adjust pointer to new free list          */
        pmem->OSMemNFree--;                          /*      One less memory block in this partition  */
        OS_EXIT_CRITICAL();
        *err = OS_NO_ERR;                            /*      No error                                 */
        return (pblk);                               /*      Return memory block to caller            */
    }
    OS_EXIT_CRITICAL();
    *err = OS_MEM_NO_FREE_BLKS;                      /* No,  Notify caller of empty memory partition  */
    return ((void *)0);                              /*      Return NULL pointer to caller            */
}
```


[1] http://zhangzhenyuan163.blog.163.com/blog/st

[2] https://zhuanlan.zhihu.com/p/465108575?utm_

[3] https://github.com/aleen42/ucos/blob/master/u

# Write your own RTOS

- Real-time Operating Systems (RTOS)
  - How to write a RTOS
    - ❑Print
      - No print support in the kernel
      - User program can use print in non-critical area

# What's Microkernel

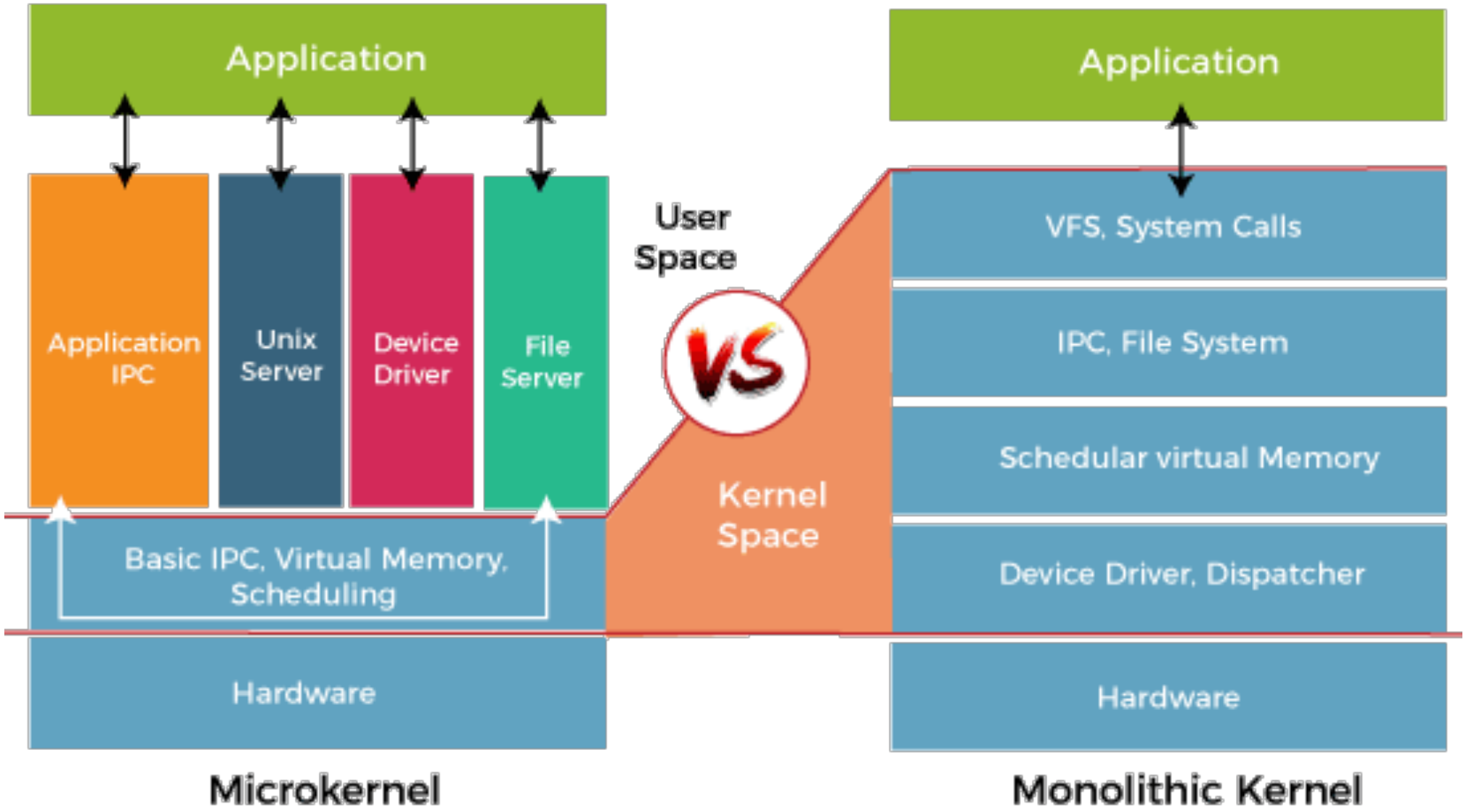
- Microkernel
  - Microkernel vs. Monolithic Kernel
    - From requirements: GPOS, Embedded OS(RTOS),
    - From architecture: Microkernel, Monolithic Kernel, ……
  - Definition
    - Monolithic kernel: the kernel subsystems are in the kernel space
    - Microkernel kernel: only key subsystems are in the kernel space; other subsystems are in the user space and communicate with IPC mechanism
  - Pros/Cons
    - (From academic aspect. Because there is never a pure popular microkernel.)
    - Pros
      - Stability
      - **Simplicity**
      - **Maintainability**
    - Cons
      - Performance
      - Debug

[1] https://yarchive.net/comp/microkernels.html

# Why Microkernel

- Microkernel
  - Pros
    - ❑Stability
      - If some of the user space components crush, the whole kernel is still safe.
    - ❑Simplicity
      - The kernel only have much fewer few codes than Monolithic kernel
    - ❑Maintainability
      - The kernel space code is really small.
    - ❑Modularity
      - It's easy to share the code between multi OSes.

# Microkernel Architecture



# Why not Microkernel

- Microkernel
  - Cons
    - ❑There are always attempts to use microkernel, why microkernel is not really popular?
    - ❑Performance: IPC rather than function call
    - ❑Yes, there is only one obvious disadvantage, but it's fatal.
    - ❑Example[1]
      - Mach (used by apple, the ancestry of MacOS), ending with Mach3.0, is a pure microkernel.
      - Mach3 is 67% slower than Unix, which is not acceptable. Then the projects stops.
      - A fun story:
        - Mach2.0 is successful because it still have some components in the user space and can achieve 75% performance of Unix.

[1] https://en.wikipedia.org/wiki/Mach_(kernel)

# IPC performance problem

- Microkernel
  - IPC Performance[1, 2]
    - ❑Cache line
      - IPC will break the cache line and cause cache miss
    - ❑Memory copy
      - Especially for network and file subsystem
    - ❑Privilege change
      - Use IPC to communicate needs privilege change
    - ❑Sync
      - Two subsystems need sync with each other
  - Design is also hard!
    - ❑Designing subsystems that totally depends IPC is hard

[1] https://www.zhihu.com/question/392807700

[2] https://dl.acm.org/doi/10.1145/3302424.3303946 EuroSys'19

# What's the truth

- Microkernel
  - Pros/Cons
    - ❑Pros
      - Stability
      - **Simplicity**
      - **Maintainability**
    - ❑Cons
      - Performance
      - Debug
  - Linus: I have some words to say[1].
  - There are always debates about the pros/cons of the Microkernel /Monolithic[2]
    - ❑Linus vs Andrew Tanenbaum (the author of Minix)
    - ❑Engineer vs Academic

"Maintainability" and "ease of porting" of microkernels are both pipe dreams that have absolutely no basis in reality. They are widespread

Actually, it seems to be that _current_ academic thoughts have started moving away from microkernels. They are much more varied than they were 5 years ago, with people still being MK proponents (although even the MK proponents tend to call them "nanokernels" in order to distance themselves from horrors like Mach), but they've had their own backlash

Umm. Microkernels don't have any advantages I'm aware of. All the much-touted "advantages" have either been outright lies ("simplicity") or things that have very little to do with microkernels ("modularity"). Some of them are just ridiculous and obviously dishonest ("performance").

[1] https://yarchive.net/comp/microkernels.html

[2] https://zh.wikipedia.org/wiki/%E5%A1%94%E8%83%BD%E9%AE%91%E5%A7%86-%E6%89%98%E7%93%A6%E8%8C%B2%E8%BE%AF%E8%AB%96

# Typical Microkernels

- Microkernel
  - Examples: 1.0[1]
    - ❑Mach[2]
      - Developed by CMU
      - Mach 1.0, compatible with UNIX, but in the spirit of Microkernels
      - Mach 2.0, not pure Microkernels, 25% performance loss
        - A fun story: Mach 2.0 uses too much code, thus it's larger than the Linux kernel. The academics creates a term nanokernel that means the kernels are small to represent Microkernels other than Mach.
      - Mach 3.0, pure Microkernels, 67% performance loss. The last version.

[1] https://www.zhihu.com/question/436729706/answer/1651950496

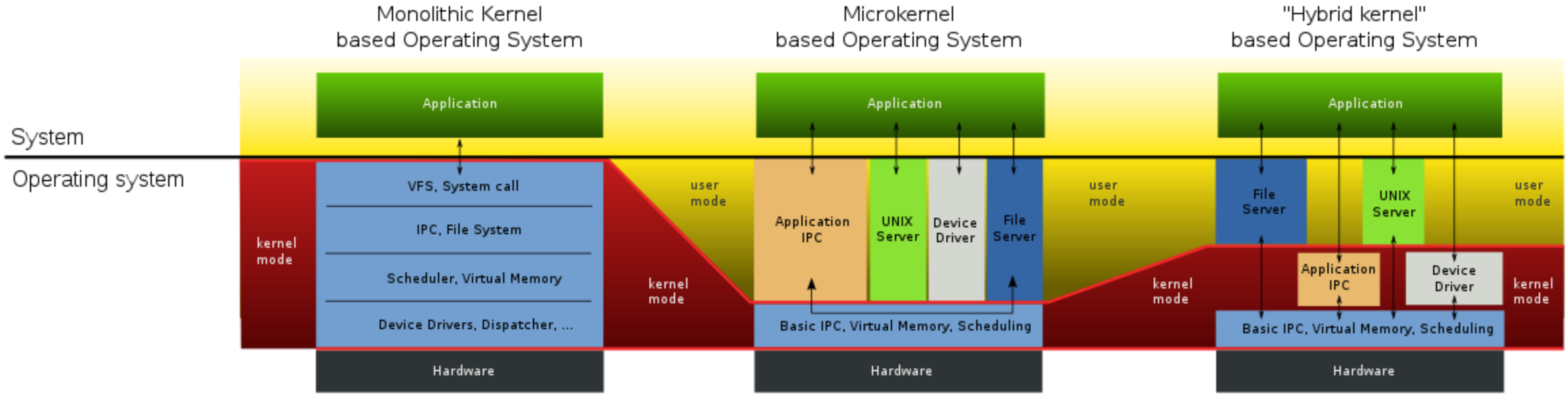
[2] https://www.gnu.org/software/hurd/history.html

# Hybrid kernel

- Microkernel
  - Things are changing……
    - ❑Maybe the extremal concept is more popular in the debate. But the mixed method is more practical.
  - There are kernels mixing these two concepts currently: Hybrid
    - ❑They put something in the kernel and some in the user space
    - ❑This is due to they first want to put all components in the user space, but the performance is hard to improve. Thus they put some in the kernel space to improve the performance.
    - ❑Examples
      - Windows NT
      - MacOS(absorbs the lessons from the Mach)

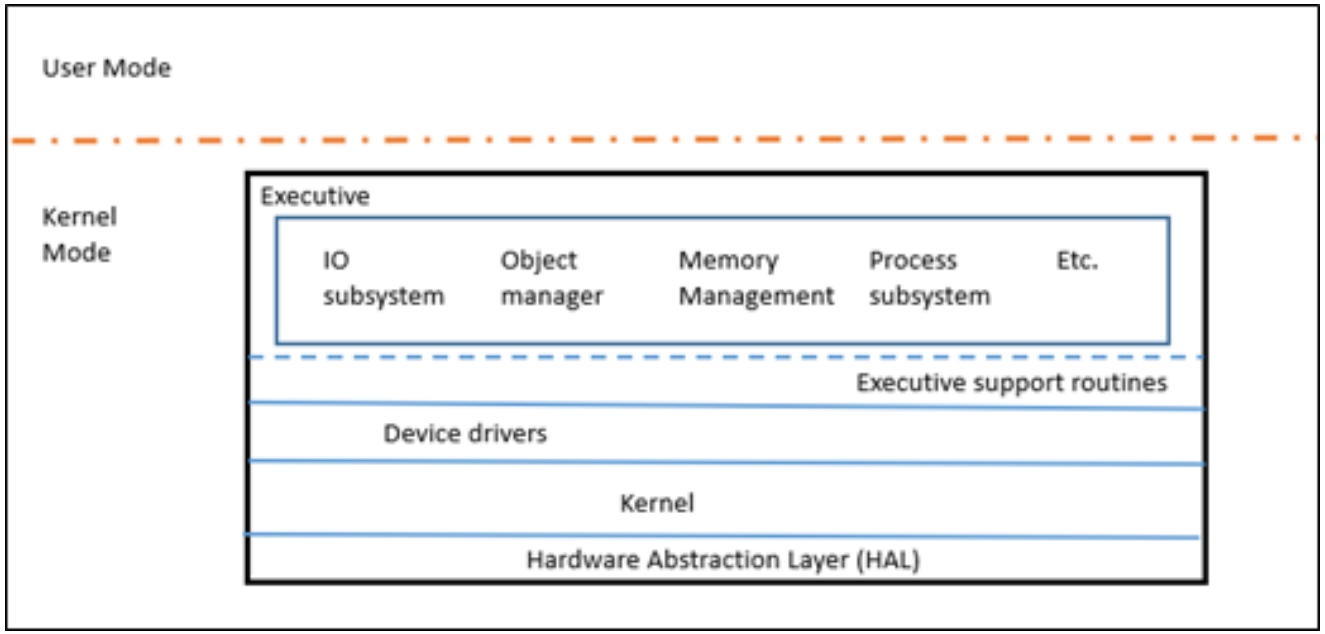
# Hybrid kernel

- Microkernel
  - Things are changing……
    - ❑Maybe the extremal concept is more popular in the debate. But the mixed method is more practical.
  - There are kernels mixing these two concepts currently: Hybrid
    - ❑They put something in the kernel and some in the user space



# Hybrid kernel

- Microkernel
  - Windows NT
    - ❑Creator: David Cutler
      - Dislike UNIX
      - Maybe he is not famous as Linus, but he is definitely a talent.
      - It takes five years to develop Windows NT.
    - ❑Windows OS kernel history
      - DOS (Before XP)
      - Windows NT(After XP)
    - ❑Windows NT architecture
      - The components are not in the Kernel but in the kernel mode
      - The kernel itself is still small



# Hybrid kernel

- Microkernel
  - Windows NT
    - ❑Creator: David Cutler
      - Dislike UNIX
      - Maybe he is not famous as Linus, but he is definitely a talent. He Is still work now.
      - It takes five years to develop Windows NT.
    - ❑Windows OS kernel history
      - DOS (Before XP)
      - Windows NT(After XP)
    - ❑Windows NT architecture
      - The components are not in the Kernel but in the kernel mode
      - The kernel itself is still small

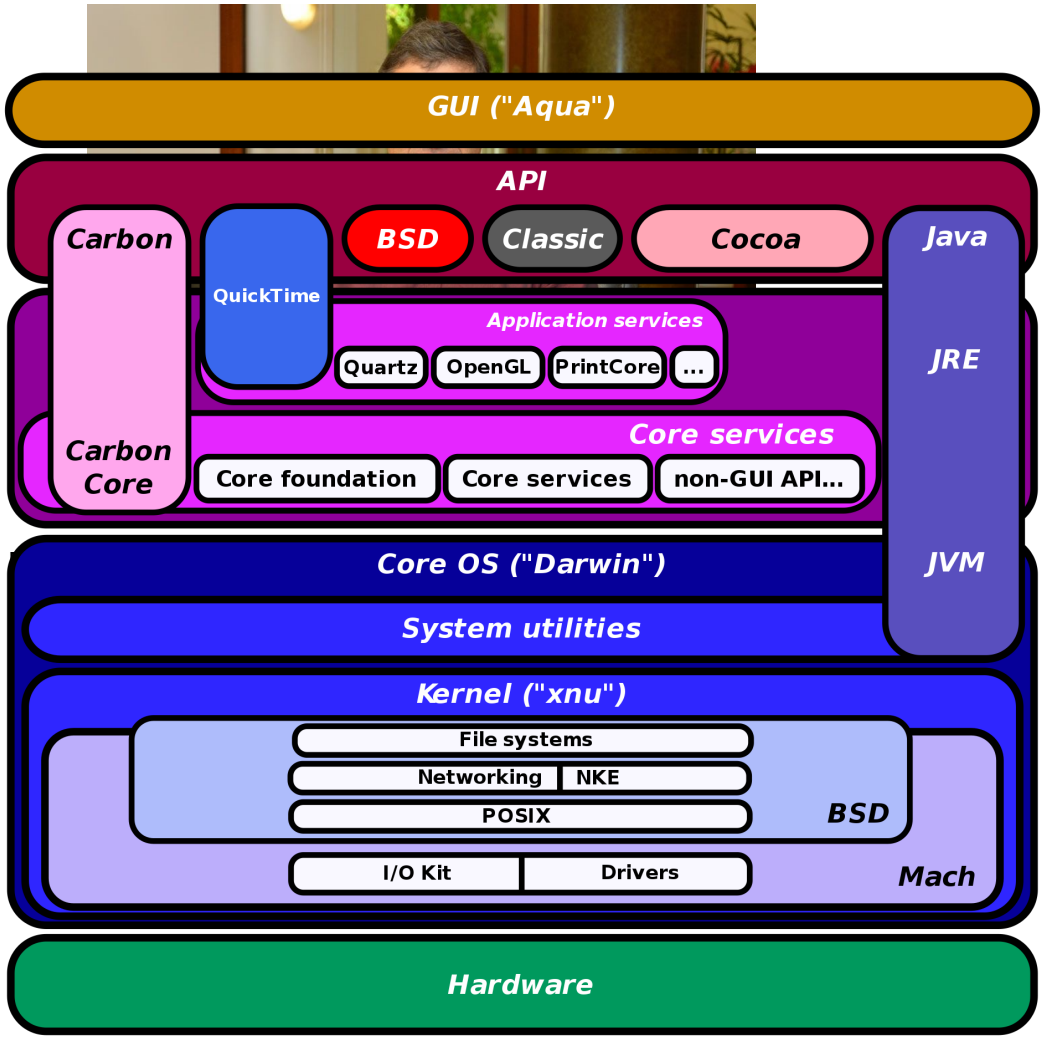
| Kernel subsystems | Lines of code |
|---|---|
| Memory Manager | 501, 000 |
| Registry | 211,000 |
| Power | 238,000 |
| Executive | 157,000 |
| Security | 135,000 |
| Kernel | 339,000 |
| Process sub-system | 116,000 |

# Hybrid kernel

- Microkernel
  - MacOS
    - ❑Darwin/MacOS Creator:
      - Tevanian: The main developer of Mach
      - IOS is also based on the MacOS
    - ❑MacOS history
      - Classical MacOS(before **Mac OS 10**)
      - MacOS/IOS(based on the NeXTSTEP, evoles to Darwin distribution, the kernel: XNU)
    - ❑XNU architecture
      - XNU is a microkernel
      - Mach: osfmk branch
      - BSD: In the user space, it uses FreeBSD(Linux denies Jobs) to be compatible with other programs(POSIX)
      - I/O Kit: user driver subsystem

# Hybrid kernel

- Microkernel
  - MacOS
    - ❑Darwin/MacOS Creator:
      - Tevanian: The main developer of Mach
      - IOS is also based on the MacOS
    - ❑MacOS history
      - Classical MacOS(before Mac OS 10)
      - MacOS/IOS(based on the NeXTSTEP, evoles to
    - ❑XNU architecture
      - XNU is a microkernel
      - Mach: osfmk branch
      - BSD: In the user space, it uses FreeBSD (Linux denies Jobs) to be compatible with other programs(POSIX)
      - I/O Kit: user driver subsystem



# Hybrid kernel

- Microkernel
  - There are kernels mixing these two concepts currently: Hybrid
    - ❑They put something in the kernel and some in the user space
    - ❑This is due to they first want to put all components in the user space, but the performance is hard to improve. Thus they put some in the kernel space to improve the performance.
    - ❑Examples
      - Windows NT
      - MacOS(aborts the lessons from the Mach)
  - Linus: I have some words to say again[1]

> As to the whole "hybrid kernel" thing - it's just marketing. It's "oh, those microkernels had good PR, how can we try to get good PR for our *working* kernel? Oh, I know, let's use a cool name and try to imply that it has all the PR advantages that that other system has"

[1] https://www.realworldtech.com/forum/?threadid=65915&curpostid=65936

# Typical Microkernels 2.0

- Microkernel 2.0
  - Improve the microkernels can be used in some performance critical areas.
  - Examples[1]
    - ❑L3/L4
      - Jochen Liedtke
      - Two techniques
        - T1: Reduce the checks in the IPC of the microkernel
        - T2: Use the register rather than the memory to pass the message
      - There are twenties of varieties of L4 due to the good performance
    - ❑QNX Neutrino

[1] https://www.zhihu.com/question/436729706/answer/1651950496

# Typical Microkernels 2.0

- Microkernel 2.0
  - Improve the microkernels can be used in some performance critical areas.
  - Examples[1]
    - ❑L3/L4
    - ❑QNX Neutrino
      - QNX microkernel builds from Unix
      - QNX itself supports POSIX interface
      - Schedule fragment makes sure the program can get the time slice

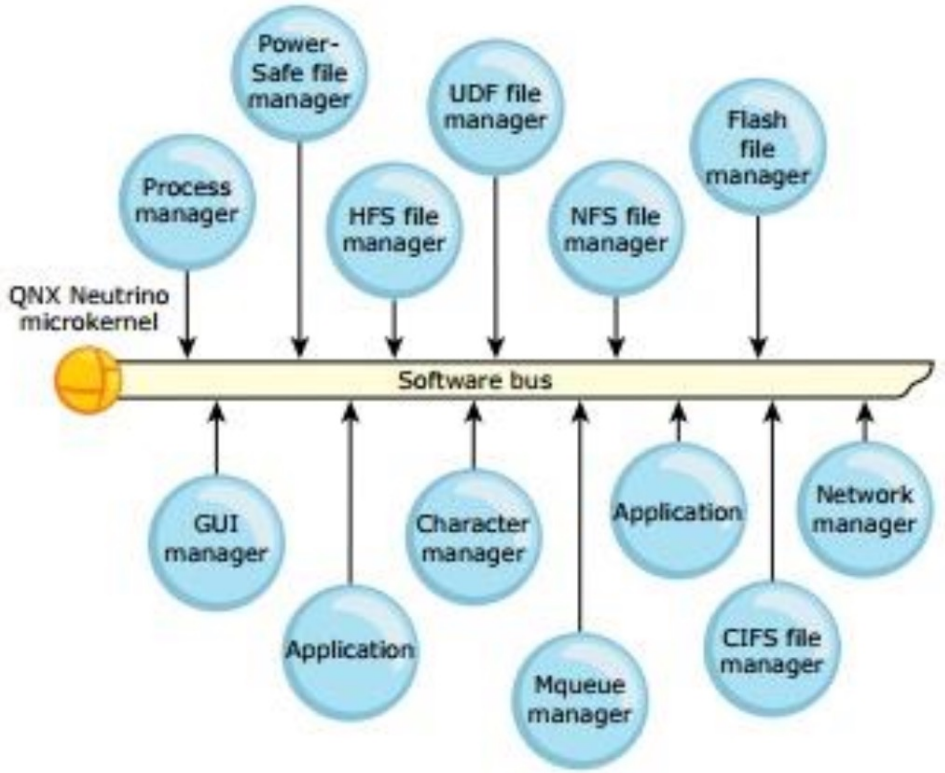



Figure 4: The QNX Neutrino RTOS architecture.

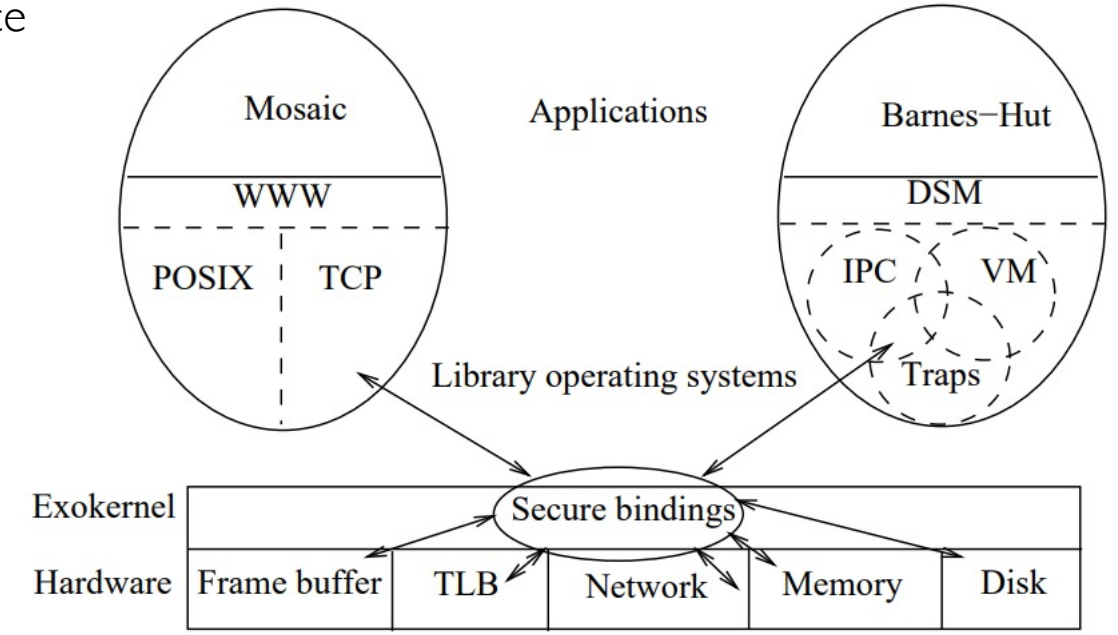
[1] https://zhuanlan.zhihu.com/p/305576924

# Typical Microkernels 3.0

- Microkernel 3.0
  - While microkernels 2.0 cares about performance, it moves the safety relevant components from kernel space to the user space.
  - The microkernels 3.0 aims to solve this and can be used in some safety critical areas.
  - Examples
    - ❑seL4
      - seL4 is a microkernel which is used in safety-critical scenarios.
      - It use a math tool called formal vertifaction to prove its code correctness.
    - ❑Zircon(Fuchsia's kernel)
      - Google wants to use it in the smart devices to replace android.
      - It's hard to say Zircon is successful. Google puts lots of resource in this system and wants to use it replace android. But the progress is slow.
    - ❑OpenHarmony
      - In development, can not judge it with source code

# There are more types of kernels

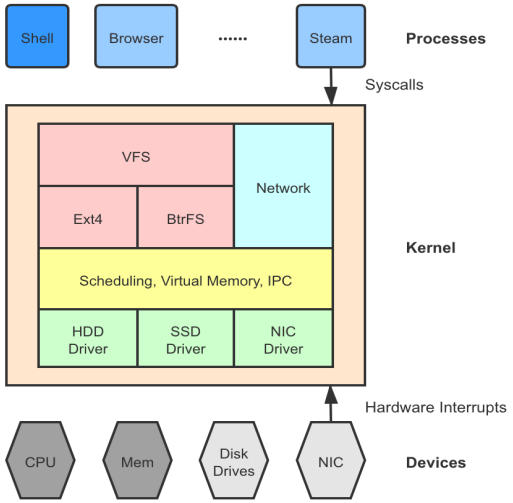
- Microkernel: One more thing
  - From architecture: Microkernel, Monolithic Kernel, Exokernel && LibOS
    - ❑Exokernel Kernel manages the hardware and exposes the interface to the upper layer
    - ❑LibOS runs on the top of the exokernel
      - Unikernel, which can runs on the top of Xen, is an image of the libos
    - ❑Microkernel vs Exokernel
      - They both want subsystems in the user space
      - Microkernel use IPC to communicate
      - Exokernel use library interface instead
      - Actually, exokernel extends the microkernel from the timeline aspect

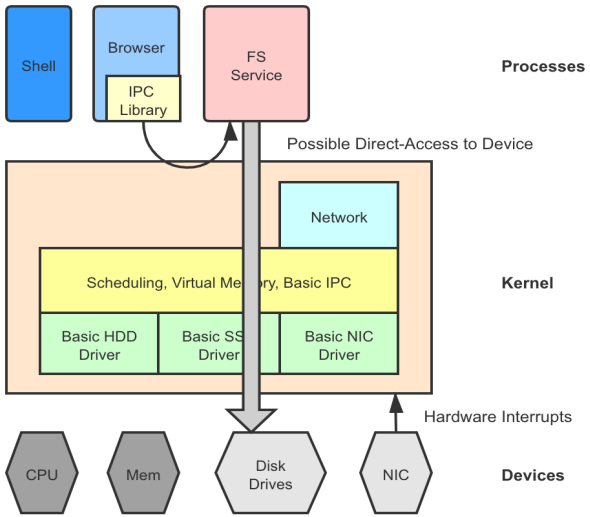


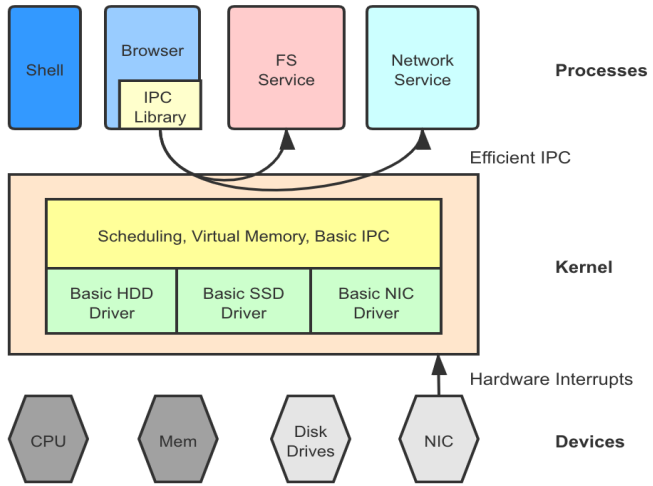

[1] https://gaocegege.com/Blog/csp/unikernel

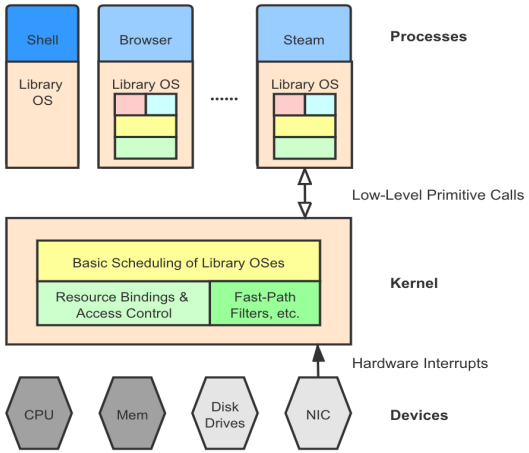
[2] https://pdos.csail.mit.edu/6.828/2008/readings/engler95exokernel.pdf

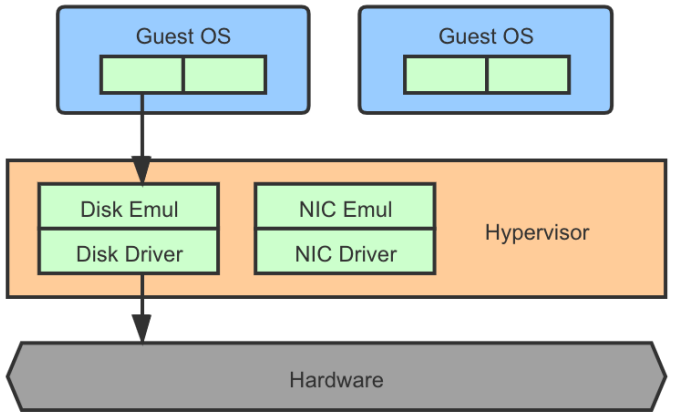
# Guess who are they

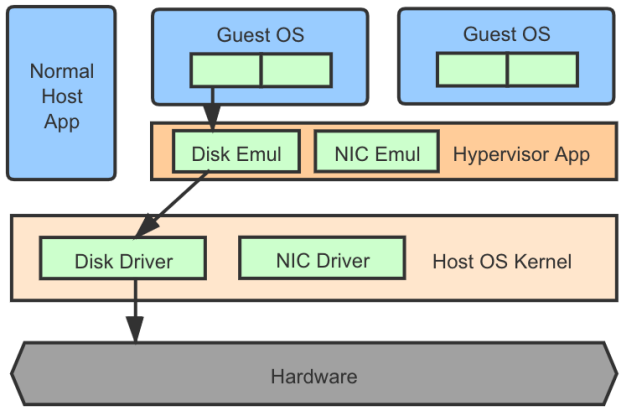



https://www.josehu.com/technical/2021/05/24/os-kernel-models.html