Integração da modelagem com Sistemas Operacionais de Tempo Real e Drivers

Prof. Dr. Giovani Gracioli
giovani@lisha.ufsc.br

ROTA2030
FUNDEP

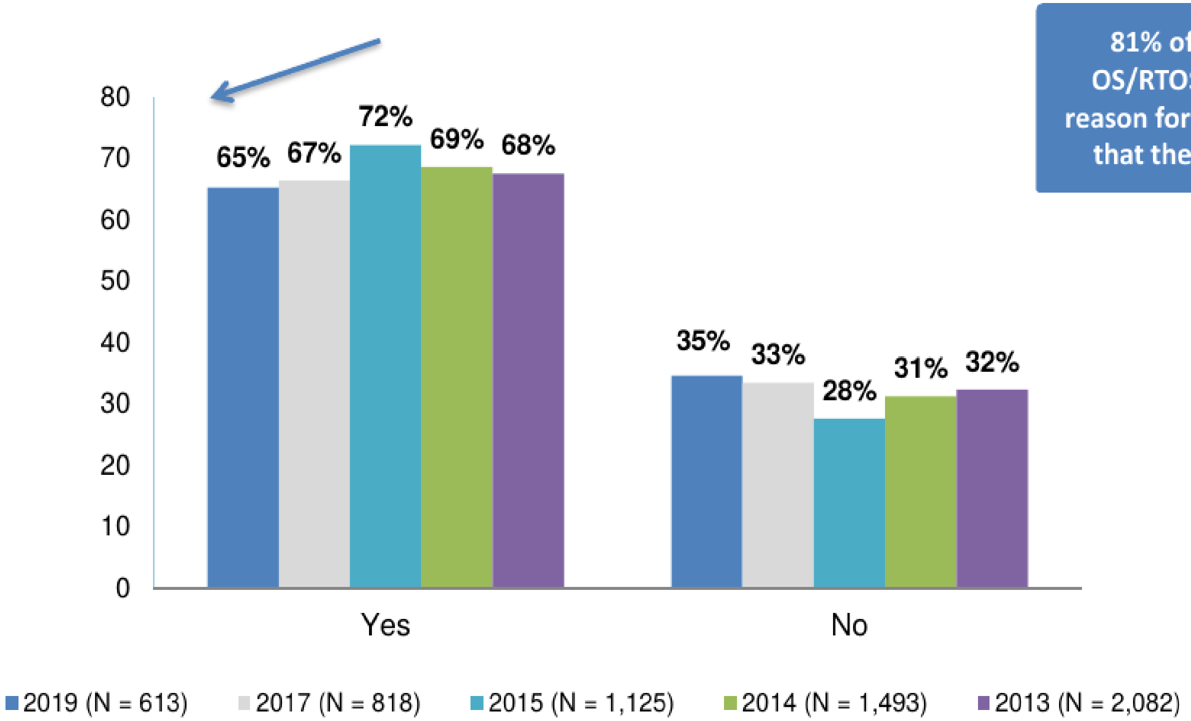
Agenda

- Integração com SOTR e Drivers
  - Visão geral
  - Exemplo

Uso de SOTR

- EETimes Embedded Markets Study 2019

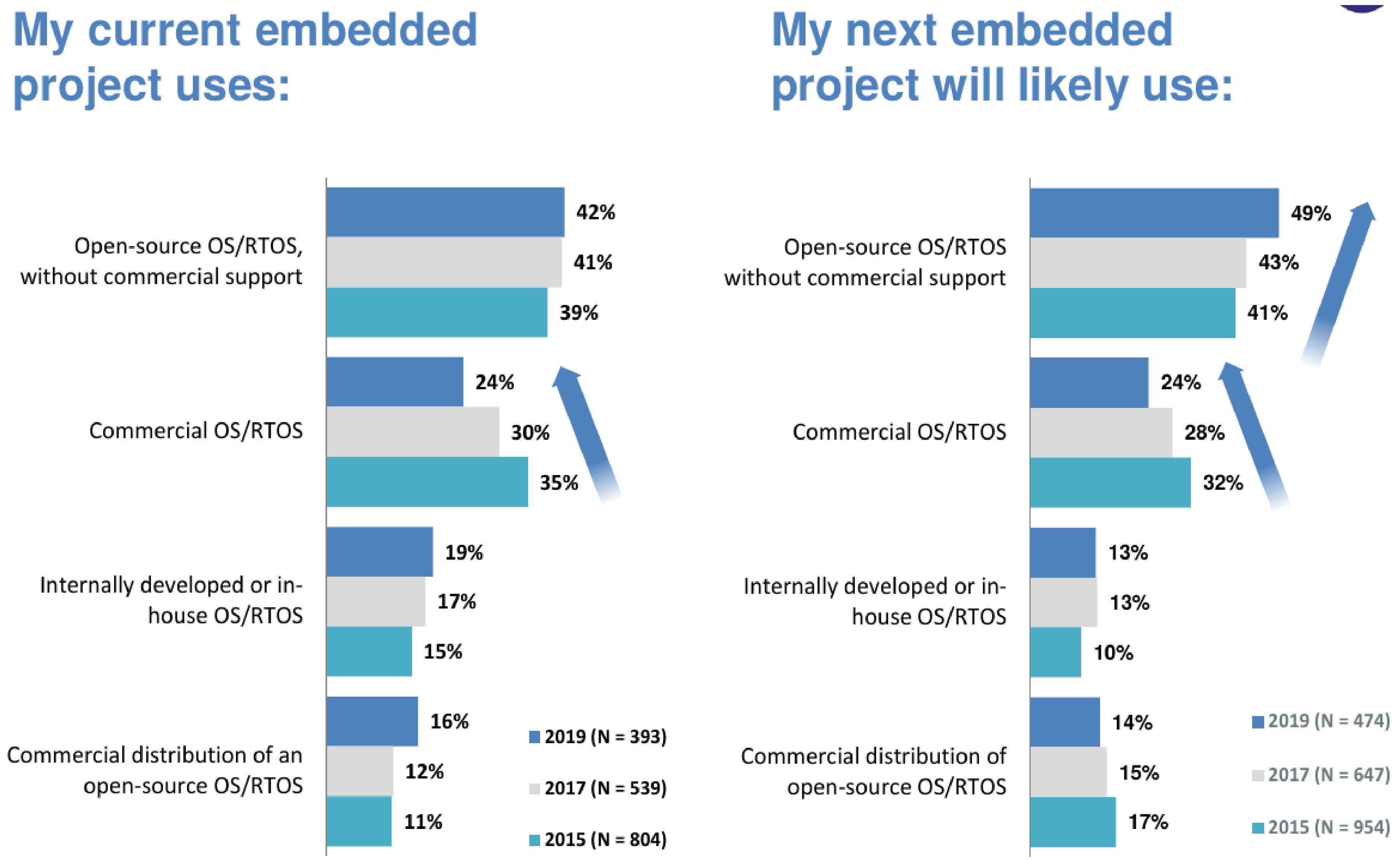
Does your current embedded project use an operating system, RTOS, kernel, software executive, or scheduler of any kind?



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

Uso de SOTR

- EETimes Embedded Markets Study 2019



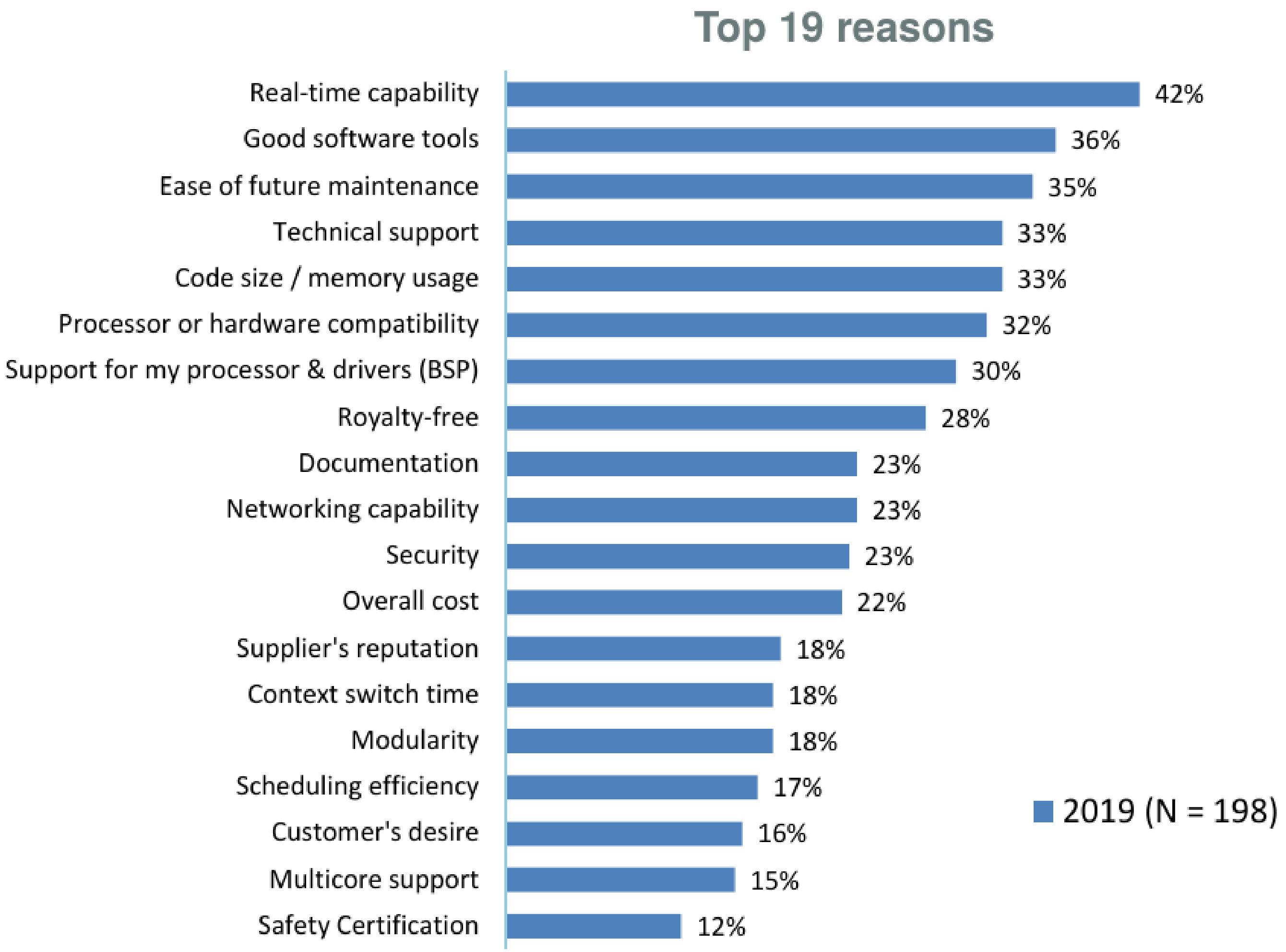
My current embedded project uses:

My next embedded project will likely use:

Uso de SOTR

- EETimes Embedded Markets Study 2019

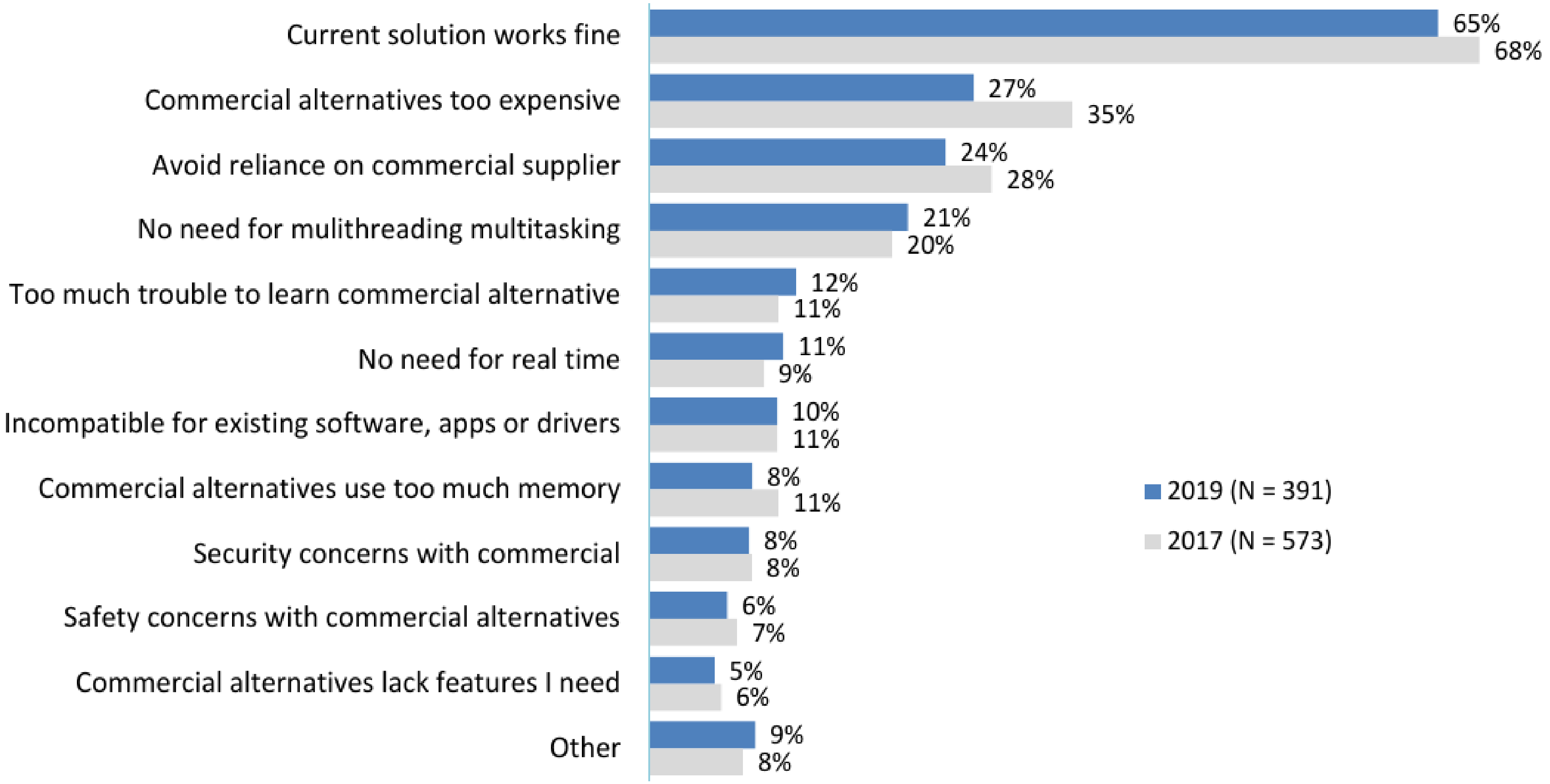
Which factors most influenced your decision to use a commercial operating system?



Uso de SOTR

- EETimes Embedded Markets Study 2019

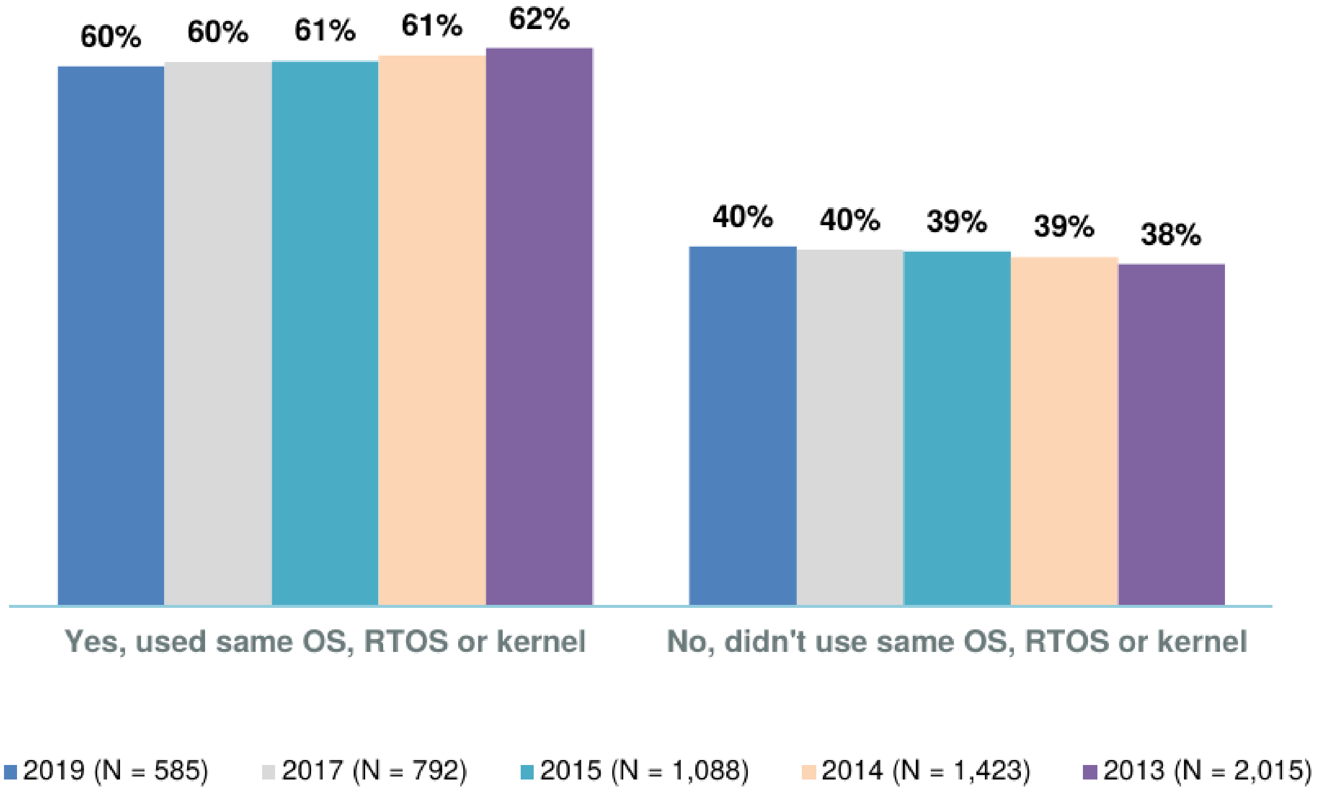
What are your reasons for not using a commercial operating system?



Uso de SOTR

- EETimes Embedded Markets Study 2019

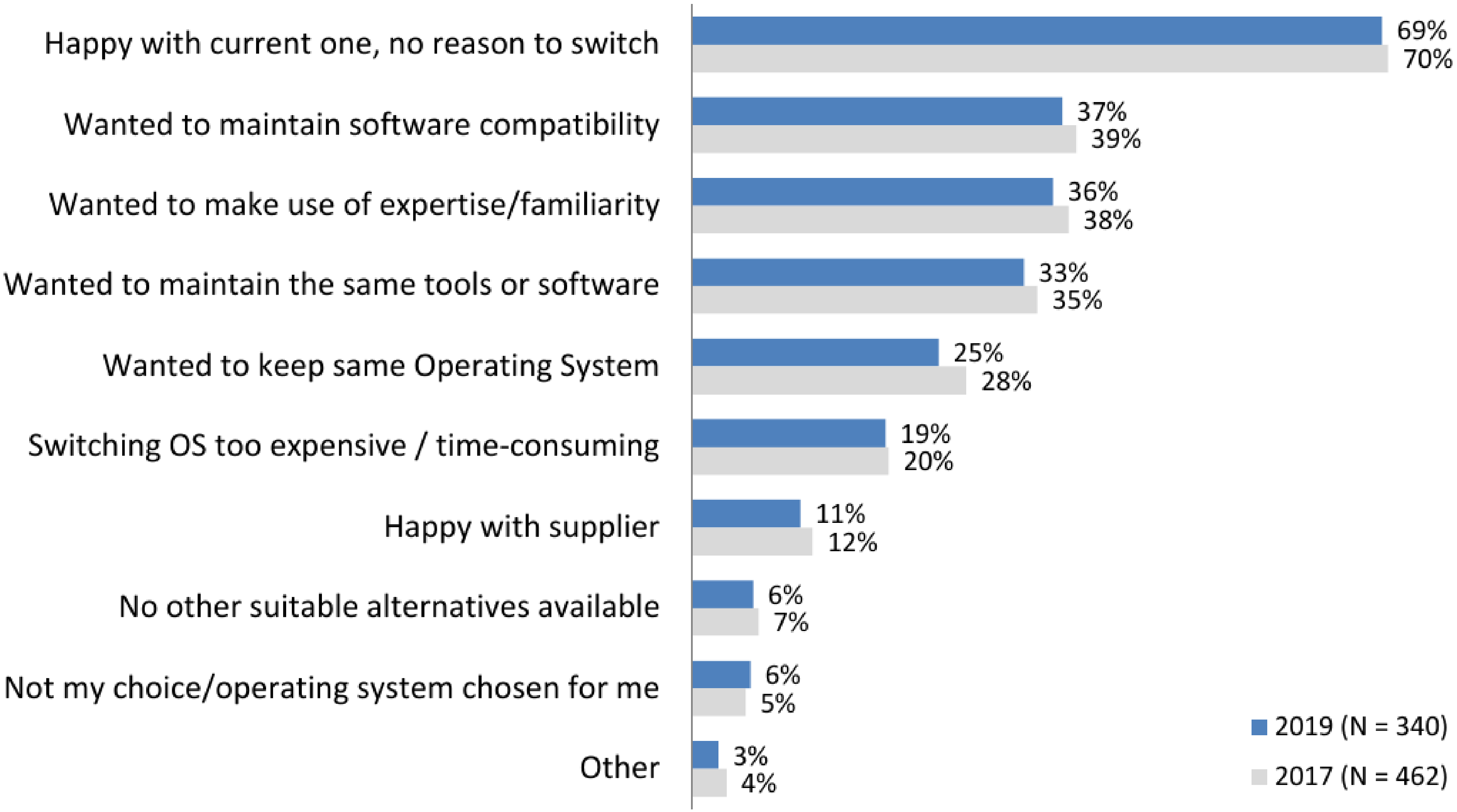
Did you use the same operating system, RTOS, or kernel as in your previous project?



Uso de SOTR

- EETimes Embedded Markets Study 2019

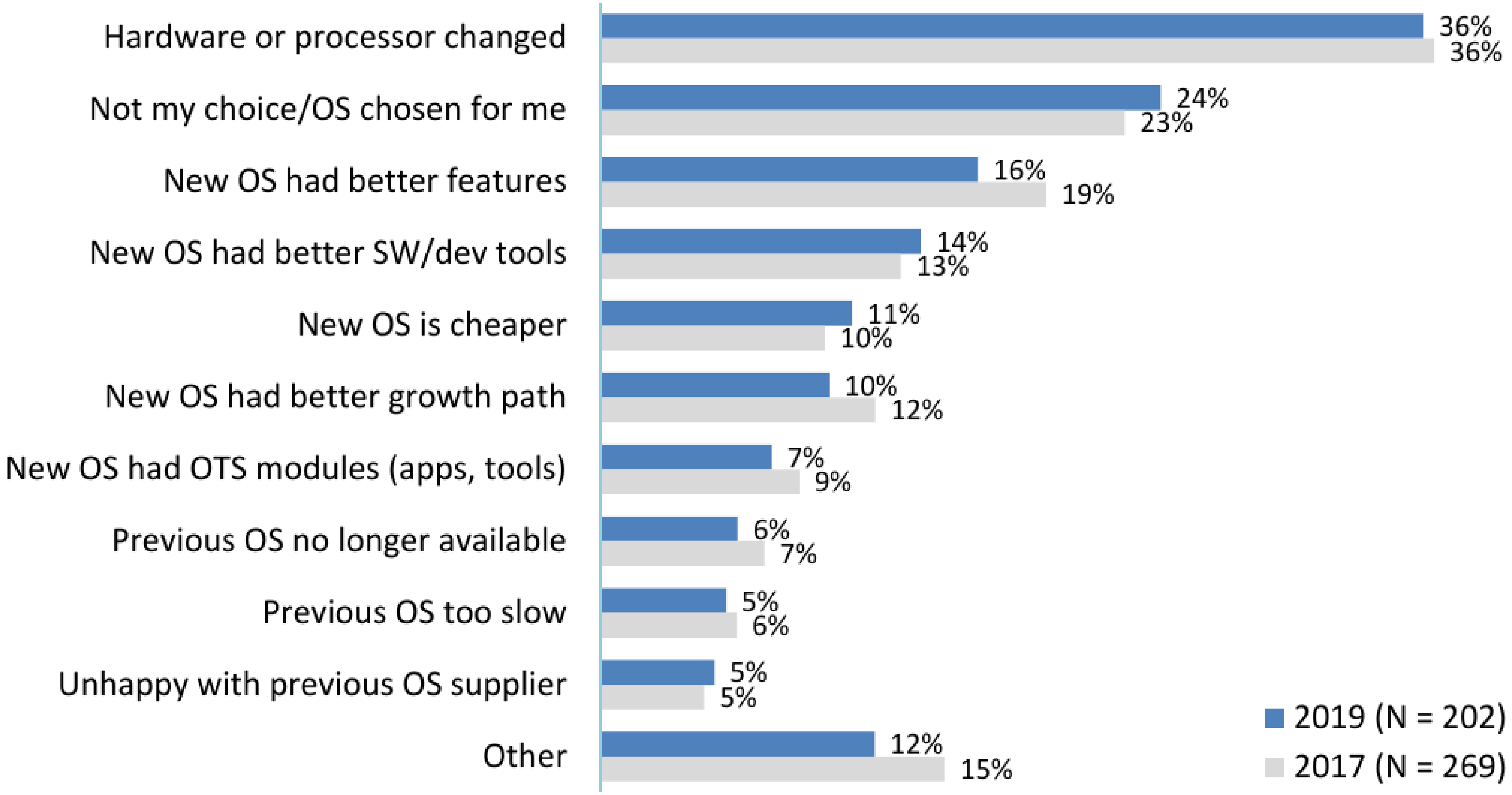
Why did you use the same operating system?



Uso de SOTR

- EETimes Embedded Markets Study 2019

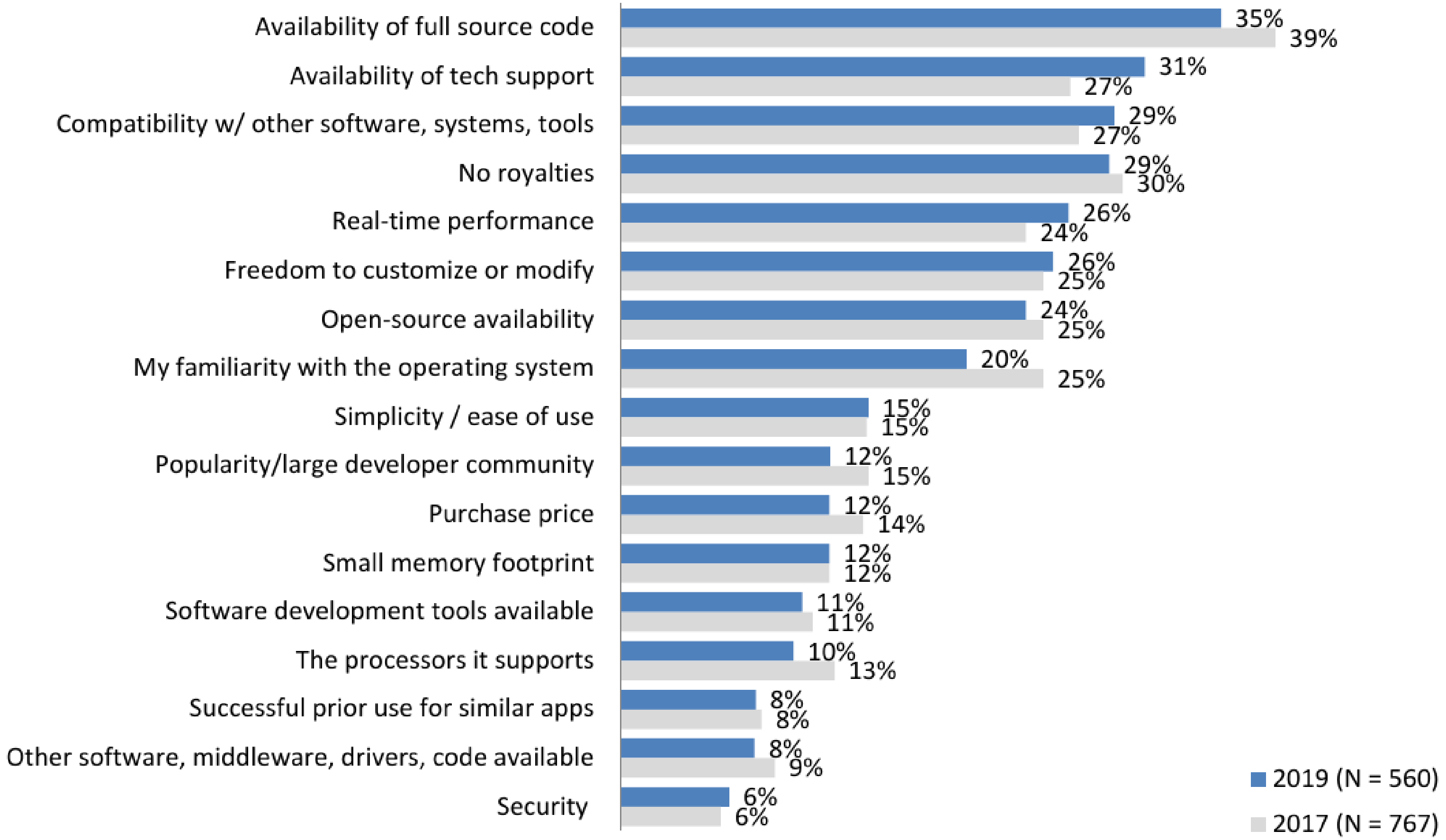
Why did you switch operating systems?



Uso de SOTR

- EETimes Embedded Markets Study 2019

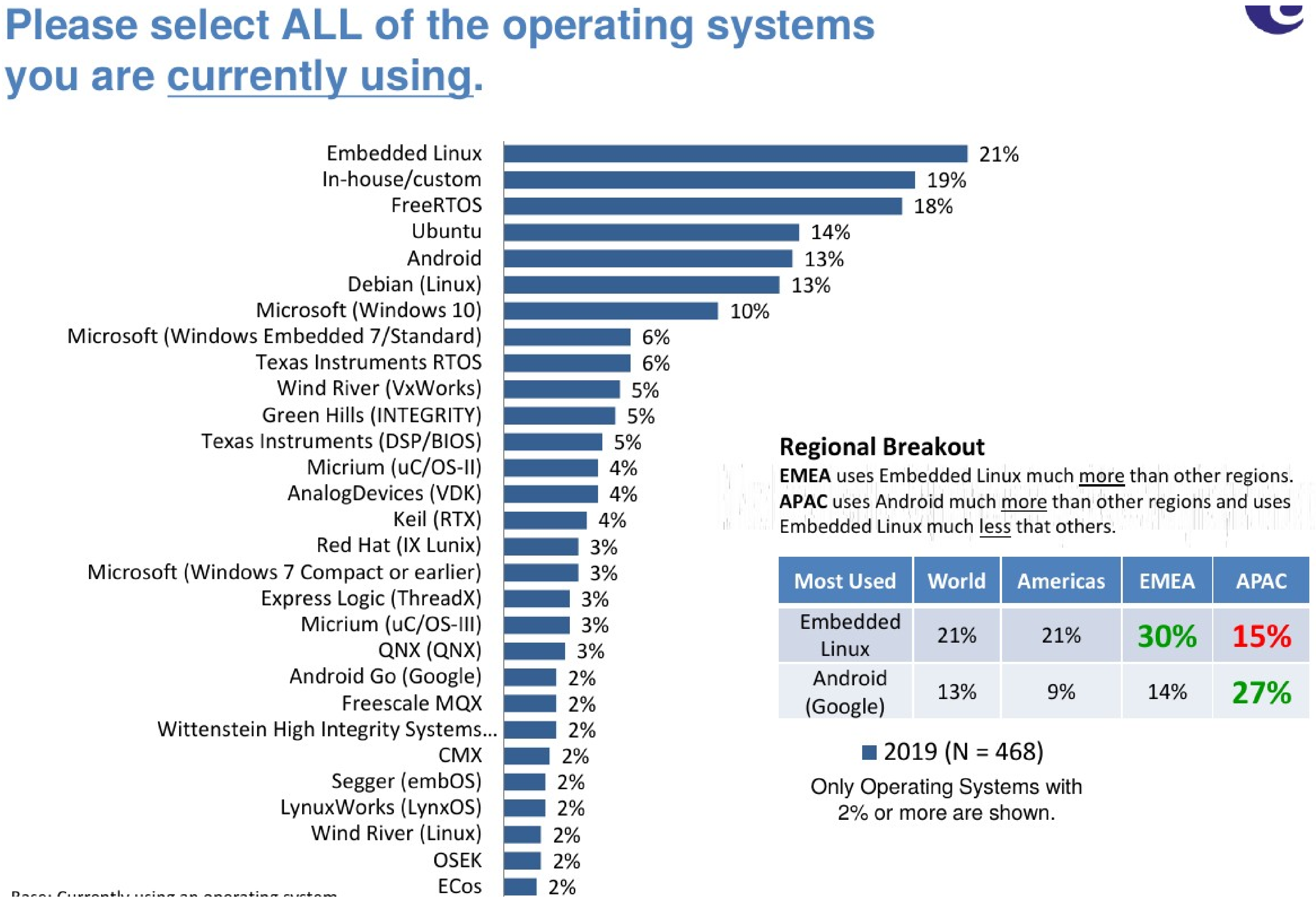
What are the most important factors in choosing an operating system?



Base: Currently using an operating system

Uso de SOTR

- EETimes Embedded Markets Study 2019



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

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%

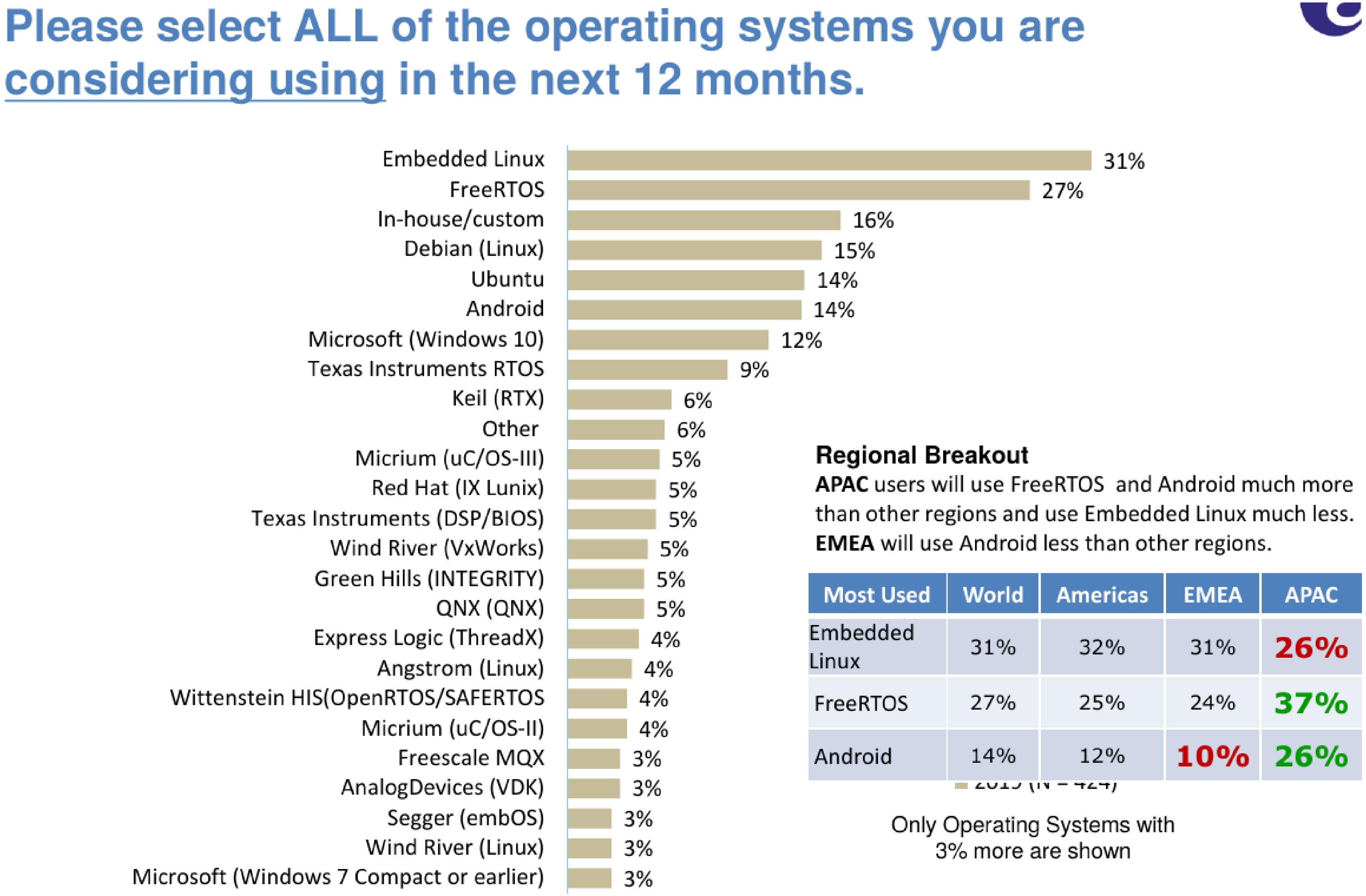
■ 2019 (N = 468)

Only Operating Systems with 2% or more are shown.

Base: Currently using an operating system

Uso de SOTR

- EETimes Embedded Markets Study 2019



Please select ALL of the operating systems you are considering using in the next 12 months.

Regional Breakout

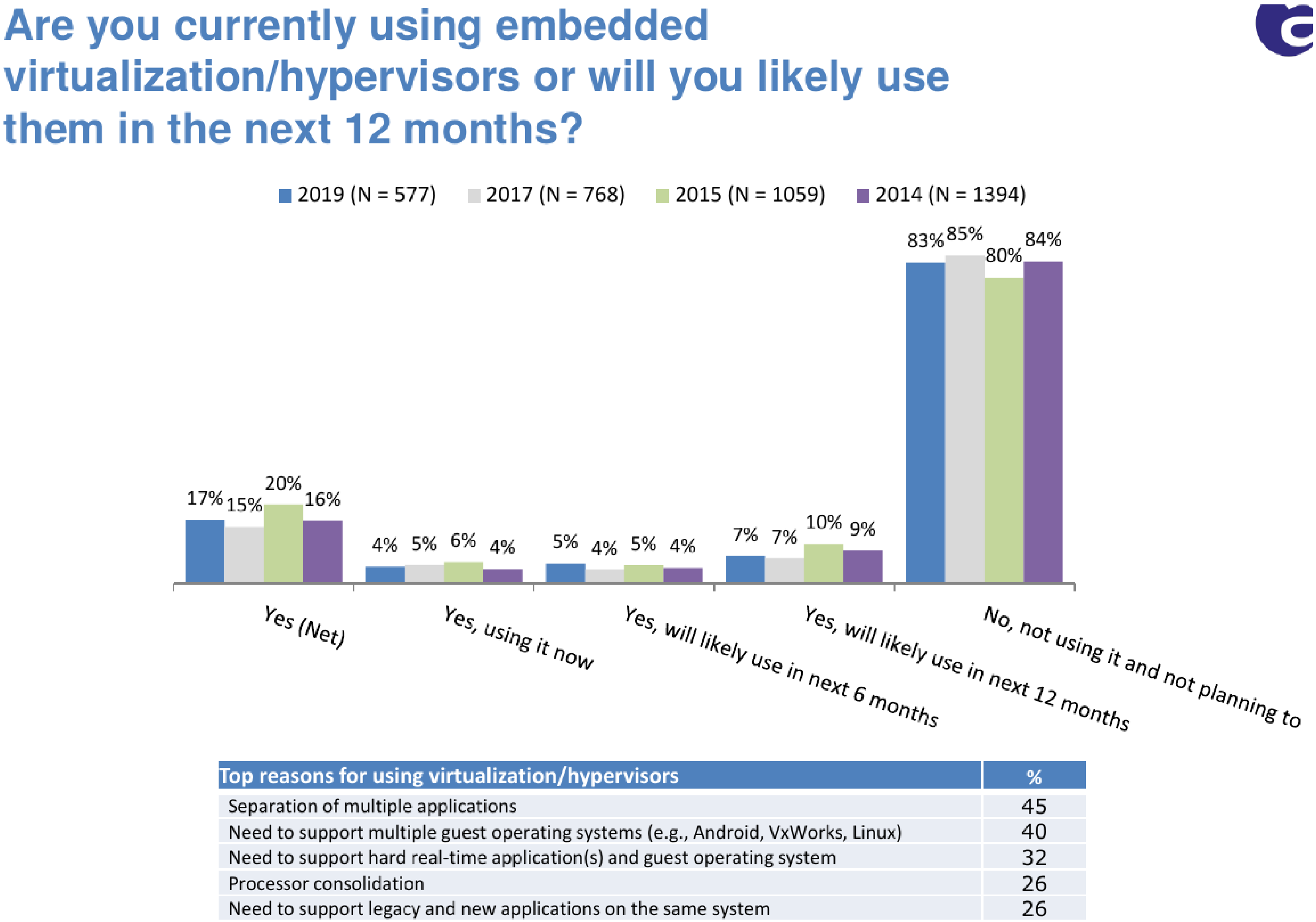
APAC users will use FreeRTOS and Android much more than other regions and use Embedded Linux much less. **EMEA** will use Android less than other regions.

Most Used	World	Americas	EMEA	APAC
Embedded Linux	31%	32%	31%	26%
FreeRTOS	27%	25%	24%	37%
Android	14%	12%	10%	26%

Only Operating Systems with 3% more are shown

Uso de SOTR

- EETimes Embedded Markets Study 2019

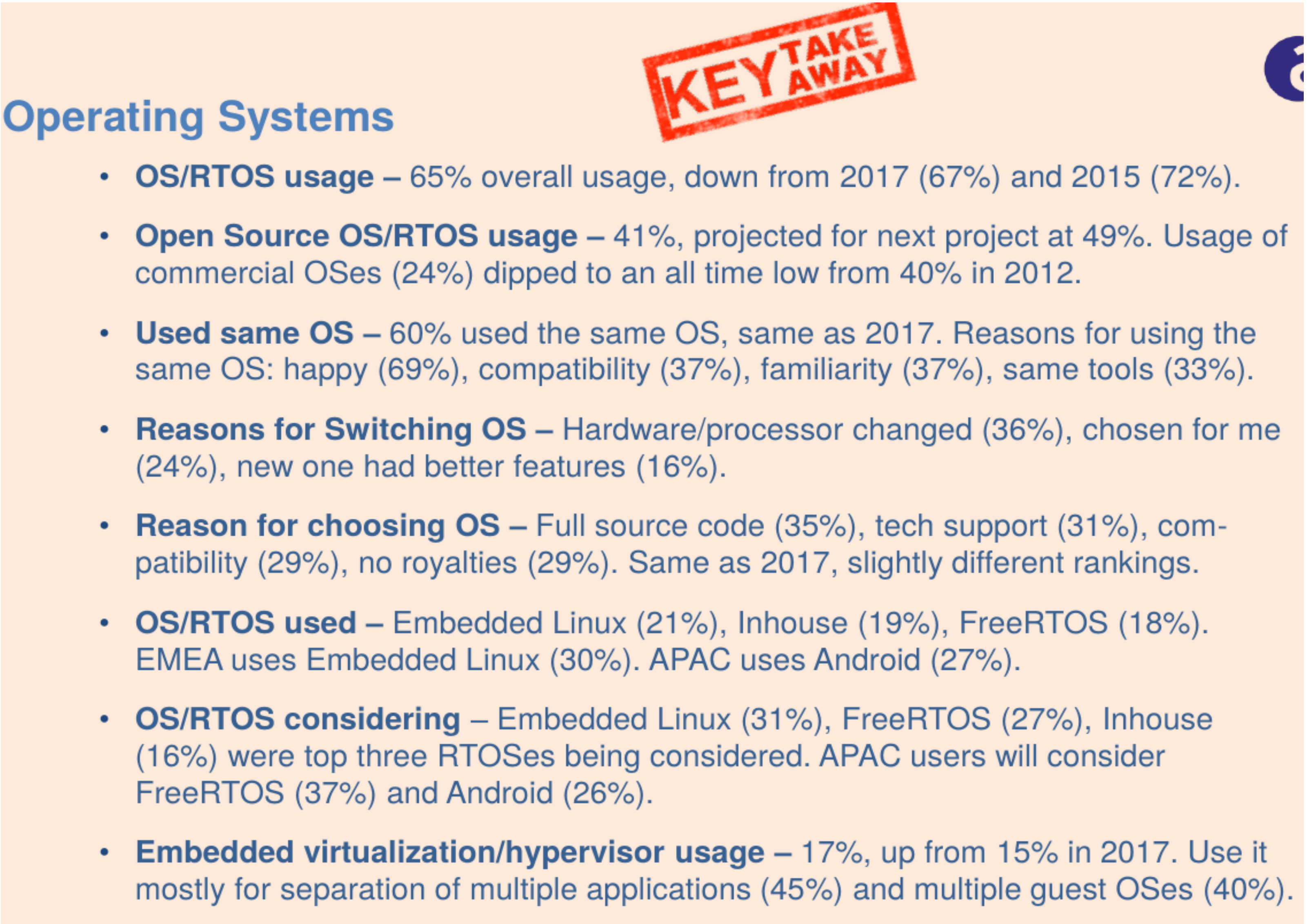


Are you currently using embedded virtualization/hypervisors or will you likely use them in the next 12 months?

Top reasons for using virtualization/hypervisors	%
Separation of multiple applications	45
Need to support multiple guest operating systems (e.g., Android, VxWorks, Linux)	40
Need to support hard real-time application(s) and guest operating system	32
Processor consolidation	26
Need to support legacy and new applications on the same system	26

Uso de SOTR

- EETimes Embedded Markets Study 2019



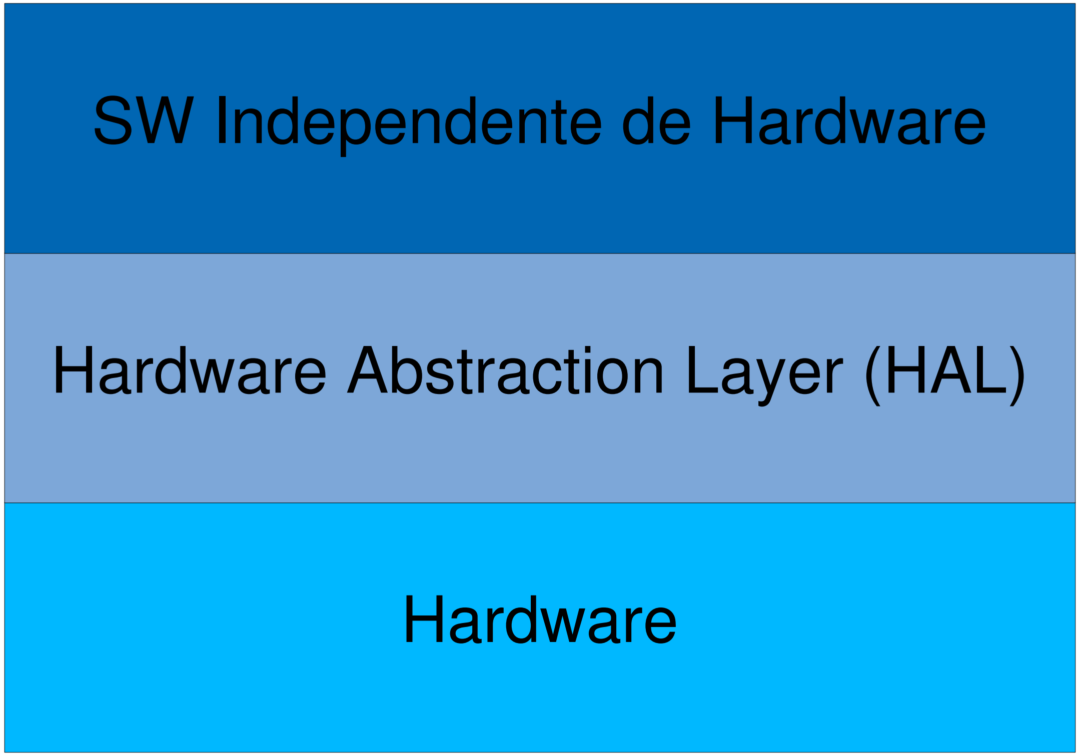
KEY TAKE AWAY

Operating Systems

- **OS/RTOS usage –** 65% overall usage, down from 2017 (67%) and 2015 (72%).
- **Open Source OS/RTOS usage –** 41%, projected for next project at 49%. Usage of commercial OSes (24%) dipped to an all time low from 40% in 2012.
- **Used same OS –** 60% used the same OS, same as 2017. Reasons for using the same OS: happy (69%), compatibility (37%), familiarity (37%), same tools (33%).
- **Reasons for Switching OS –** Hardware/processor changed (36%), chosen for me (24%), new one had better features (16%).
- **Reason for choosing OS –** Full source code (35%), tech support (31%), compatibility (29%), no royalties (29%). Same as 2017, slightly different rankings.
- **OS/RTOS used –** Embedded Linux (21%), Inhouse (19%), FreeRTOS (18%). EMEA uses Embedded Linux (30%). APAC uses Android (27%).
- **OS/RTOS considering** – Embedded Linux (31%), FreeRTOS (27%), Inhouse (16%) were top three RTOSes being considered. APAC users will consider FreeRTOS (37%) and Android (26%).
- **Embedded virtualization/hypervisor usage –** 17%, up from 15% in 2017. Use it mostly for separation of multiple applications (45%) and multiple guest OSes (40%).

Integração de SOTR no Modelo

- O SOTR provê dois tipos de abstração para as aplicações



SW Independente de Hardware

Hardware Abstraction Layer (HAL)

Hardware

Integração de SOTR no Modelo

- A HAL implementa os drivers
- A interface para o mesmo tipo de dispositivo é obrigatoriamente a mesma
  - Se mudar um dispositivo, o software que está acima não muda
- A camada independente de hardware provê os conceitos usados pela aplicação
  - Tarefas/Threads periódica/aperiódica
  - Sincronização entre tarefas
  - Impressão
  - Comunicação, etc

Alguns exemplos – Erika RTOS

```
TaskType task1_id;
int main(void) {
    StatusType s = E_OK;
    s |= CreateTask( &task1_id, OSEE_TASK_TYPE_EXTENDED,
                  TASK_FUNC(Task1), 1U, 1U, 1U, 1024 );

    s |= CreateTask( &isr2_clock_id, EE_TASK_TYPE_ISR2, clock_handler,
                  1U, 1U, 1U, SYSTEM_STACK);

    /* Tie ISR2 With IRQ */
    s = SetISR2Source(isr2_clock_id, OSEE_GTIMER_IRQ);
    StartOS(OSDEFAULTAPPMODE);
    printk("MAIN | Initializing the timer...\n");
    ticks_per_beat = osEE_aarch64_gtimer_get_freq();
    ticks_per_beat /= BEATS_PER_SEC;
    expected_ticks = osEE_aarch64_gtimer_get_ticks() + ticks_per_beat;
    osEE_aarch64_gtimer_start(ticks_per_beat,
                  OSEE_AARCH64_GTIMER_COUNTDOWN);
    ActivateTask(task1_id);
    return 0;
}
```

Alguns exemplos – Erika RTOS

```
void clock_handler(void) {
    SetEvent(task1_id, 0x8);
    osEE_aarch64_gtimer_start(ticks_per_beat,
                OSEE_AARCH64_GTIMER_COUNTDOWN);
}
```

Alguns exemplos – Erika RTOS

```
DeclareTask(Task1);
TASK(Task1) {
    while(counter < MAX_EXECS) {
        WaitEvent(0x8);
        ClearEvent(0x8);
        do_work();
    }
    TerminateTask();
}
```

Alguns exemplos - EPOS

```
 int main() {
    cout << "\nThis test consists in creating three periodic threads as follows:" << endl;
    thread_a = new Periodic_Thread(RTConf(period_a * 1000, 0, 0, 0, iterations), &func_a);
    thread_b = new Periodic_Thread(RTConf(period_b * 1000, 0, 0, 0, iterations), &func_b);
    thread_c = new Periodic_Thread(RTConf(period_c * 1000, 0, 0, 0, iterations), &func_c);

    exec('M');
    chrono.start();
    int status_a = thread_a->join();
    int status_b = thread_b->join();
    int status_c = thread_c→join();
    return 0;
}
```

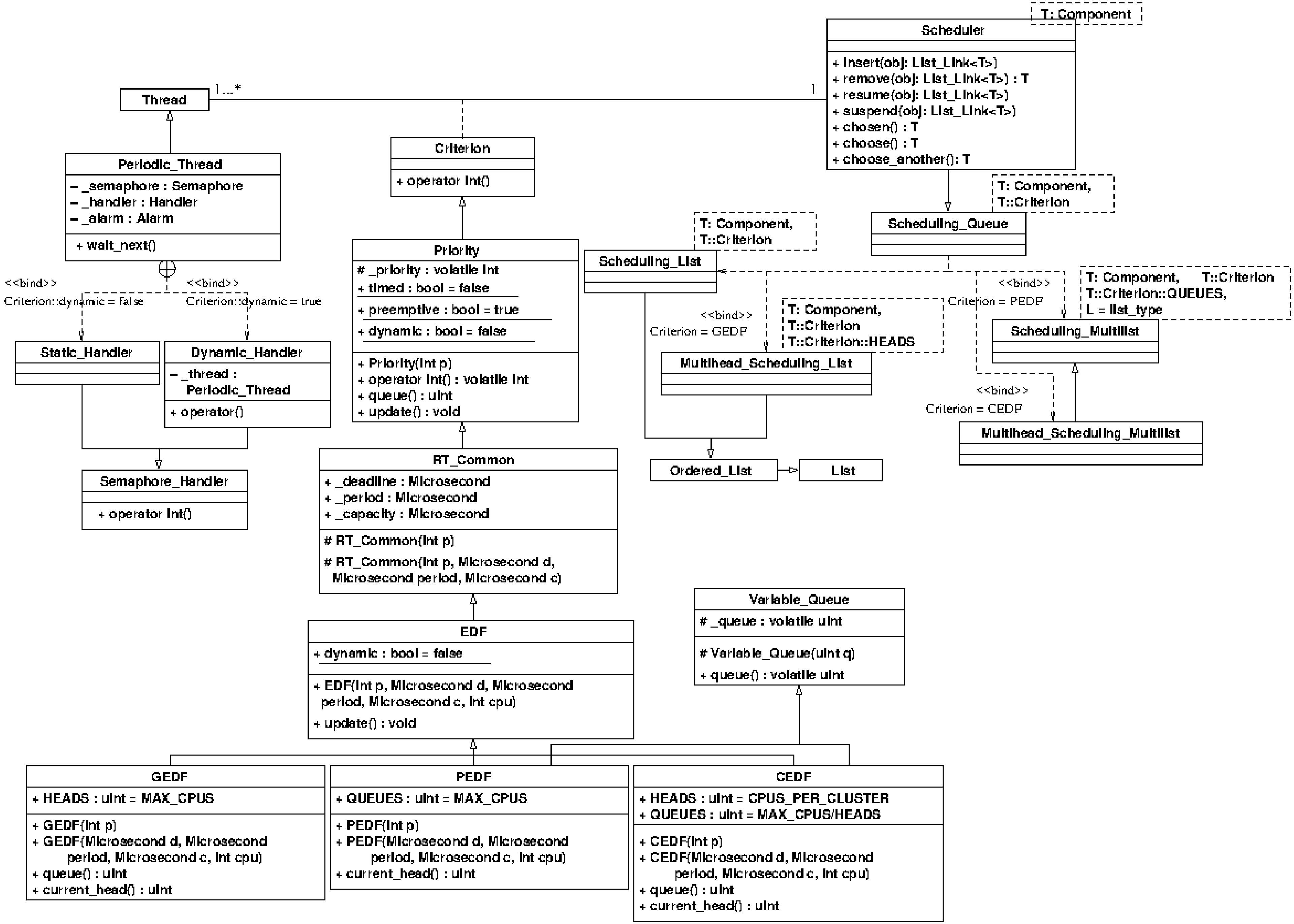
Alguns exemplos - EPOS

```
 int func_a() {
    exec('A');

    do {
        exec('a', wcet_a);
    } while (Periodic_Thread::wait_next());

    exec('A');

    return 'A';
}
```

Configuração

```
template<> struct Traits<Build>: public Traits_Tokens {
   // Basic configuration
   static const unsigned int MODE = LIBRARY;
   static const unsigned int ARCHITECTURE = ARMv7;
   static const unsigned int MACHINE = Cortex;
   static const unsigned int MODEL = Zynq;
   static const unsigned int CPUS = 1;
}
template<> struct Traits<Thread>: public Traits<Build> {
   …..
   typedef Scheduling_Criteria::RM Criterion;
   ...
};
```

Alguns exemplos - EPOS



T: Component

Scheduler

+ insert(obj: List_Link<T>)
+ remove(obj: List_Link<T>) : T
+ resume(obj: List_Link<T>)
+ suspend(obj: List_Link<T>)
+ chosen() : T
+ choose() : T
+ choose_another(): T

Thread

1...*

1

Periodic_Thread

- _semaphore : Semaphore
- _handler : Handler
- _alarm : Alarm

+ wait_next()

<<bind>>
Criterion::dynamic = false

<<bind>>
Criterion::dynamic = true

Static_Handler

Dynamic_Handler

- _thread :
  Periodic_Thread

+ operator()

Semaphore_Handler

+ operator int()

Criterion

+ operator int()

Priority

_priority : volatile int
+ timed : bool = false
+ preemptive : bool = true
+ dynamic : bool = false

+ Priority(int p)
+ operator int() : volatile int
+ queue() : uint
+ update() : void

RT_Common

+ _deadline : Microsecond
+ _period : Microsecond
+ _capacity : Microsecond

# RT_Common(int p)
# RT_Common(int p, Microsecond d,
  Microsecond period, Microsecond c)

EDF

+ dynamic : bool = false

+ EDF(int p, Microsecond d, Microsecond
  period, Microsecond c, int cpu)
+ update() : void

GEDF

+ HEADS : uint = MAX_CPUS

+ GEDF(int p)
+ GEDF(Microsecond d, Microsecond
  period, Microsecond c, int cpu)
+ queue() : uint
+ current_head() : uint

PEDF

+ QUEUES : uint = MAX_CPUS

+ PEDF(int p)
+ PEDF(Microsecond d, Microsecond
  period, Microsecond c, int cpu)
+ current_head() : uint

CEDF

+ HEADS : uint = CPUS_PER_CLUSTER
+ QUEUES : uint = MAX_CPUS/HEADS

+ CEDF(int p)
+ CEDF(Microsecond d, Microsecond
  period, Microsecond c, int cpu)
+ queue() : uint
+ current_head() : uint

Variable_Queue

_queue : volatile uint

Variable_Queue(uint q)
+ queue() : volatile uint

T: Component,
T::Criterion

Scheduling_Queue

T: Component,
T::Criterion

Scheduling_List

<<bind>>
Criterion = GEDF

T: Component,
T::Criterion,
T::Criterion::HEADS

Multihead_Scheduling_List

<<bind>>
Criterion = PEDF

T: Component, T::Criterion
T::Criterion::QUEUES,
L = list_type

Scheduling_Multilist

<<bind>>
Criterion = CEDF

Multihead_Scheduling_Multilist

Ordered_List

List

Referências

- http://www.erika-enterprise.com/

- https://epos.lisha.ufsc.br/

Obrigado!