CMRX

realtime microkernel

RTOS for microcontrollers with memory protection

Eduard Drusa

https://cmrxrtos.org/

Embedded Software Engineering Kongress

Who would win?

DEC PDP-7

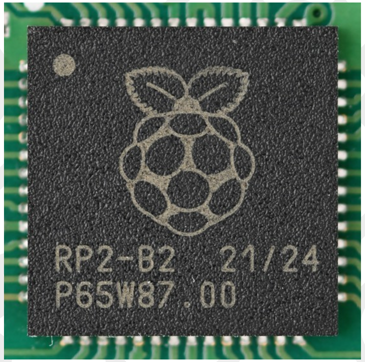
1964

$72,000

Raspberry Pi RP2040

2021

1.16 €



nopisonnope@gmail.com

Who am I?

- 17+ years - Software developer
- Web, Desktop, Server, Embedded, Automotive
- Applying methods from one industry in others
- Recent interest: „Can we make embedded software more secure?“

nopisonnope@gmail.com

MS DOS

Mac OS

Zephyr

freeRTOS

ThreadX

NuttX RTOS

nopisonnope@gmail.com

The Problem

- Bulk of IoT systems runs unprotected kernels
- Existing solutions are too complicated to be understood
- Developers generally avoid the topic of memory isolation: „Its complicated“

```c
void vStartMPUDemo( void )
{
/**
 * Since stack of a task is protected using MPU, it must satisfy MPU
 * requirements as mentioned at the top of this file.
 */
static StackType_t xROAccessTaskStack[ configMINIMAL_STACK_SIZE ]
__attribute__( ( aligned( configMINIMAL_STACK_SIZE * sizeof( StackType_t ) ) )
);
static StackType_t xRWAccessTaskStack[ configMINIMAL_STACK_SIZE ]
__attribute__( ( aligned( configMINIMAL_STACK_SIZE * sizeof( StackType_t ) ) )
);
TaskParameters_t xROAccessTaskParameters =
{
    .pvTaskCode     = prvROAccessTask,
    .pcName         = "ROAccess",
    .usStackDepth   = configMINIMAL_STACK_SIZE,
    .pvParameters   = NULL,
    .uxPriority     = tskIDLE_PRIORITY,
    .puxStackBuffer = xROAccessTaskStack,
    .xRegions       = {
                        { ucSharedMemory,
                        SHARED_MEMORY_SIZE,
                        portMPU_REGION_PRIVILEGED_READ_WRITE_UNPRIV_READ_ON
                        LY | portMPU_REGION_EXECUTE_NEVER    },
                        { ( void * ) ucROTaskFaultTracker,
                        SHARED_MEMORY_SIZE, portMPU_REGION_READ_WRITE |
                        portMPU_REGION_EXECUTE_NEVER
                            },
                        { 0,                    0,
                          0
                                                },
                      }
};
TaskParameters_t xRWAccessTaskParameters =
{
    .pvTaskCode     = prvRWAccessTask,
    .pcName         = "RWAccess",
    .usStackDepth   = configMINIMAL_STACK_SIZE,
    .pvParameters   = NULL,
    .uxPriority     = tskIDLE_PRIORITY,
    .puxStackBuffer = xRWAccessTaskStack,
    .xRegions       = {
                        { ucSharedMemory,  SHARED_MEMORY_SIZE,
                        portMPU_REGION_READ_WRITE |
                        portMPU_REGION_EXECUTE_NEVER},
                        { 0,          0,          0
                                                    },
                        { 0,          0,          0
                                                    },
                      }
};

    /* Create an unprivileged task with RO access to ucSharedMemory. */
    xTaskCreateRestricted( &( xROAccessTaskParameters ), NULL );

    /* Create an unprivileged task with RW access to ucSharedMemory. */
    xTaskCreateRestricted( &( xRWAccessTaskParameters ), NULL );
}
```

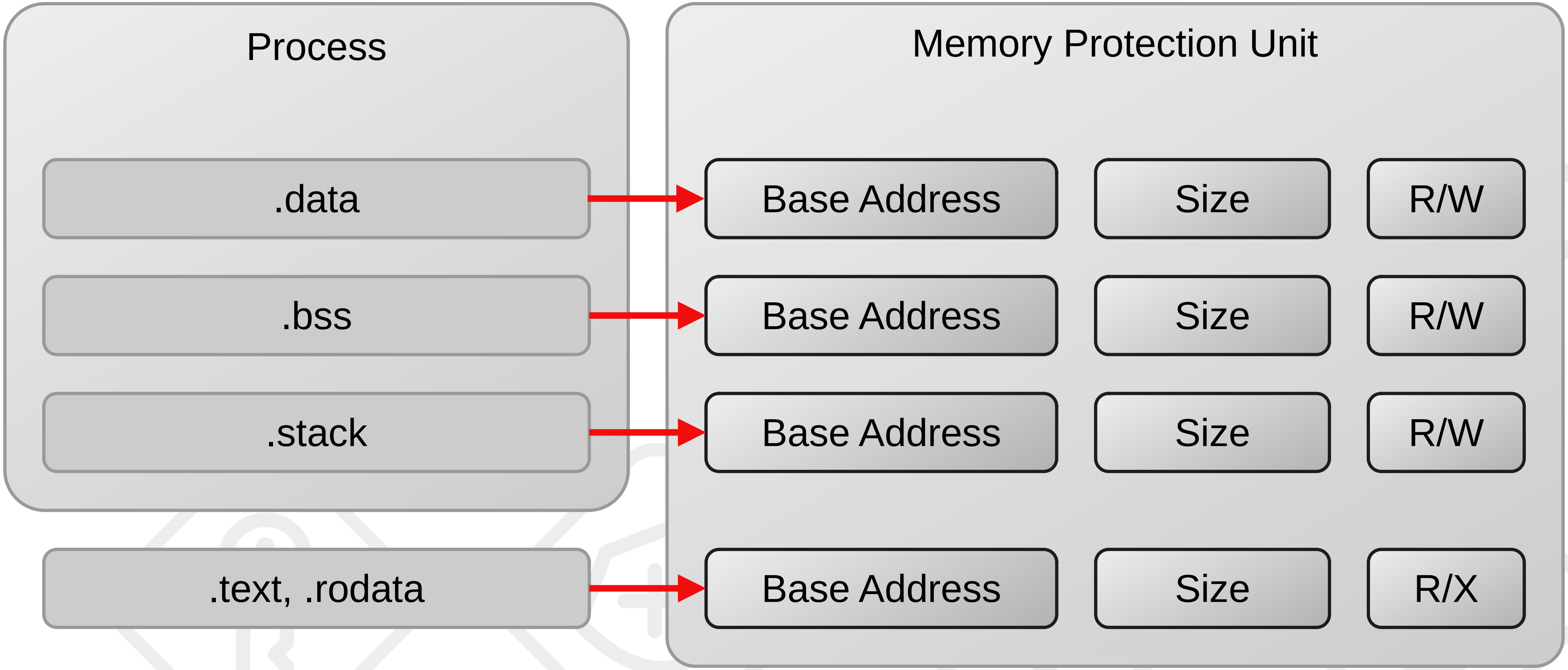
```c
#include <zephyr/kernel.h>
#include <zephyr/app_memory/app_memdomain.h>
#include <zephyr/sys/libc-hooks.h>

/* Memory partitions definitions */
K_APPMEM_PARTITION_DEFINE(partition1);
K_APPMEM_PARTITION_DEFINE(partition2);
K_APPMEM_PARTITION_DEFINE(partition_shared);

/* Memory domains declarations */
struct k_mem_domain domain_a;
struct k_mem_domain domain_b;

/* Variables in specific memory partitions */
K_APP_DMEM(partition1) int var_1 = 11;
K_APP_DMEM(partition2) int var_2 = 22;
K_APP_DMEM(partition_shared) int var_shared = 99;

/* Thread functions for application A */
void app_a_threads(void *arg1, void *arg2, void *arg3) {
    printk("App A, Thread %d: can access var_1 = %d and var_shared = %d\n", \
                (int) arg1, var_1, var_shared);
    printk("App A, Thread %d: cannot access var_2\n", (int) arg1);
}

/* Thread functions for application B */
void app_b_threads(void *arg1, void *arg2, void *arg3) {
    printk("App B, Thread %d: can access var_2 = %d and var_shared = %d\n", \
                (int) arg1, var_2, var_shared);
    printk("App B, Thread %d: cannot access var_1\n", (int) arg1);
}


/* Threads definition for different applications */
K_THREAD_DEFINE(tid_app_a1, 1024, app_a_threads, (void*) 1, NULL, NULL, 10, K_USER, 0);
K_THREAD_DEFINE(tid_app_a2, 1024, app_a_threads, (void*) 2, NULL, NULL, 10, K_USER, 0);
K_THREAD_DEFINE(tid_app_b1, 1024, app_b_threads, (void*) 1, NULL, NULL, 10, K_USER, 0);
K_THREAD_DEFINE(tid_app_b2, 1024, app_b_threads, (void*) 2, NULL, NULL, 10, K_USER, 0);
/* Kernel thread function */
void kernel_thread(void *arg1, void *arg2, void *arg3) {
    printk("Kernel Thread: can access var_1 = %d, var_2 = %d, and var_shared = %d\n", \
                var_1, var_2, var_shared);
}

/* Kernel thread */
K_THREAD_DEFINE(tid_kernel, 1024, kernel_thread, NULL, NULL, NULL, 10, 0, 0);

/* Memory partition configuration arrays */
struct k_mem_partition *app_a_partitions[] = {
#ifdef CONFIG_CPU_CORTEX_M
    &z_libc_partition,
#endif
    &partition1,
    &partition_shared
};

struct k_mem_partition *app_b_partitions[] = {
#ifdef CONFIG_CPU_CORTEX_M
    &z_libc_partition,
#endif
    &partition2,
    &partition_shared
};

/* Main function to initialize domains and add threads to them */
int main(void)
{
    /* Initialize and assign partitions to domains */
    k_mem_domain_init(&domain_a, ARRAY_SIZE(app_a_partitions), app_a_partitions);
    k_mem_domain_init(&domain_b, ARRAY_SIZE(app_b_partitions), app_b_partitions);

    /* Add app_a threads to domain a */
    k_mem_domain_add_thread(&domain_a, tid_app_a1);
    k_mem_domain_add_thread(&domain_a, tid_app_a2);

    /* Add app_b threads to domain b */
    k_mem_domain_add_thread(&domain_b, tid_app_b1);
    k_mem_domain_add_thread(&domain_b, tid_app_b2);

    return 0;
}
```

What if...

...we could simply write this?

```c
/* Variables that are only accessible by the process */
#define BUFSIZE 512
static char buffer[BUFSIZE];

static int blinky_main()
{
    /* Do something interesting */
    return 0;
}

OS_APPLICATION(blinky);
OS_THREAD_CREATE(blinky, blinky_main, NULL, 32);
```

Isolation Using Processes



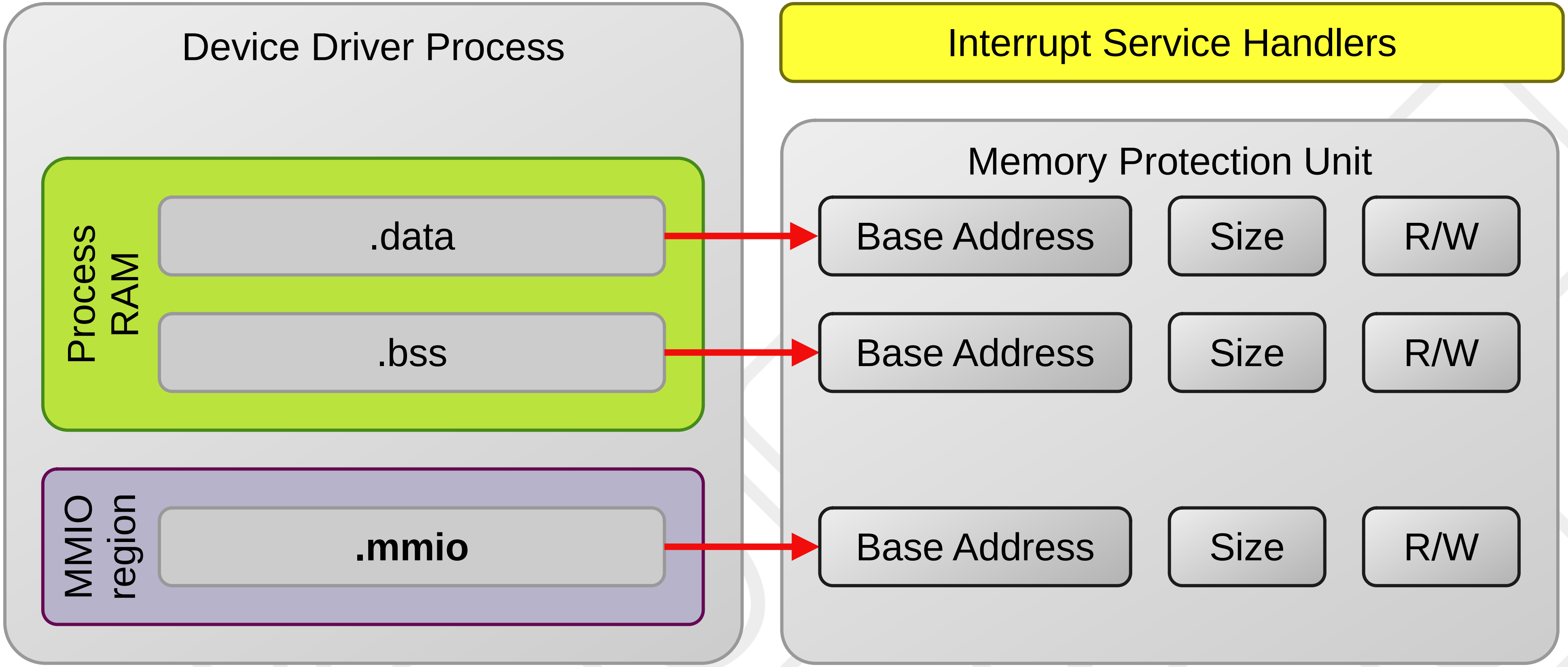
nopisonnope@gmail.com

OK, but...

...manual approach can access peripherals. This, can't!

```
/* Variables that are only accessible by the process */
#define BUFSIZE 512
static char buffer[BUFSIZE];

static int blinky_main()
{
    /* Do something interesting */
    return 0;
}

/* Accidentally, these are where RP2040 GPIO is mapped to */
OS_APPLICATION_MMIO_RANGES(blinky, 0x40000000, 0x50000000, 0xd0000000, 0xe0000000);
OS_APPLICATION(blinky);
OS_THREAD_CREATE(blinky, blinky_main, NULL, 32);
```

Drivers in Userspace



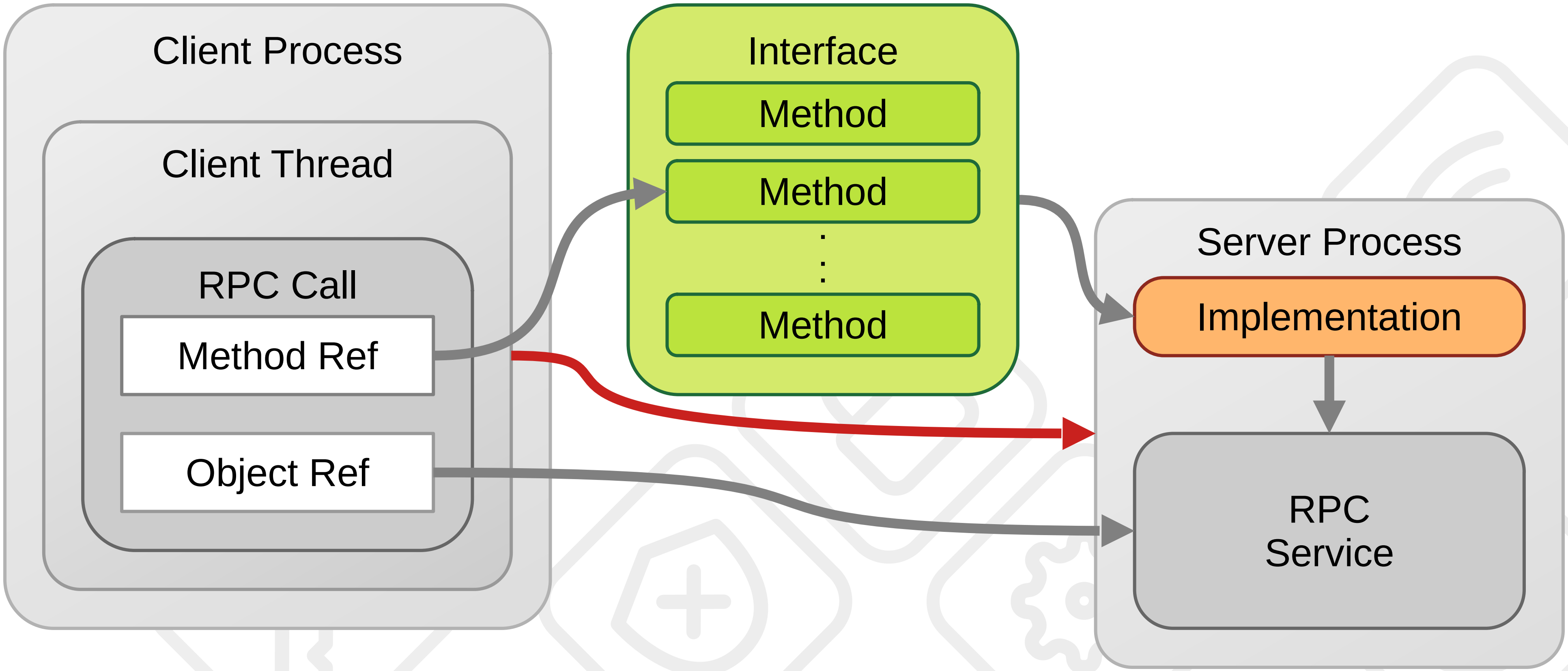
nopisonnope@gmail.com

OK, but...

...manual approach can create shared memory. This, can't!

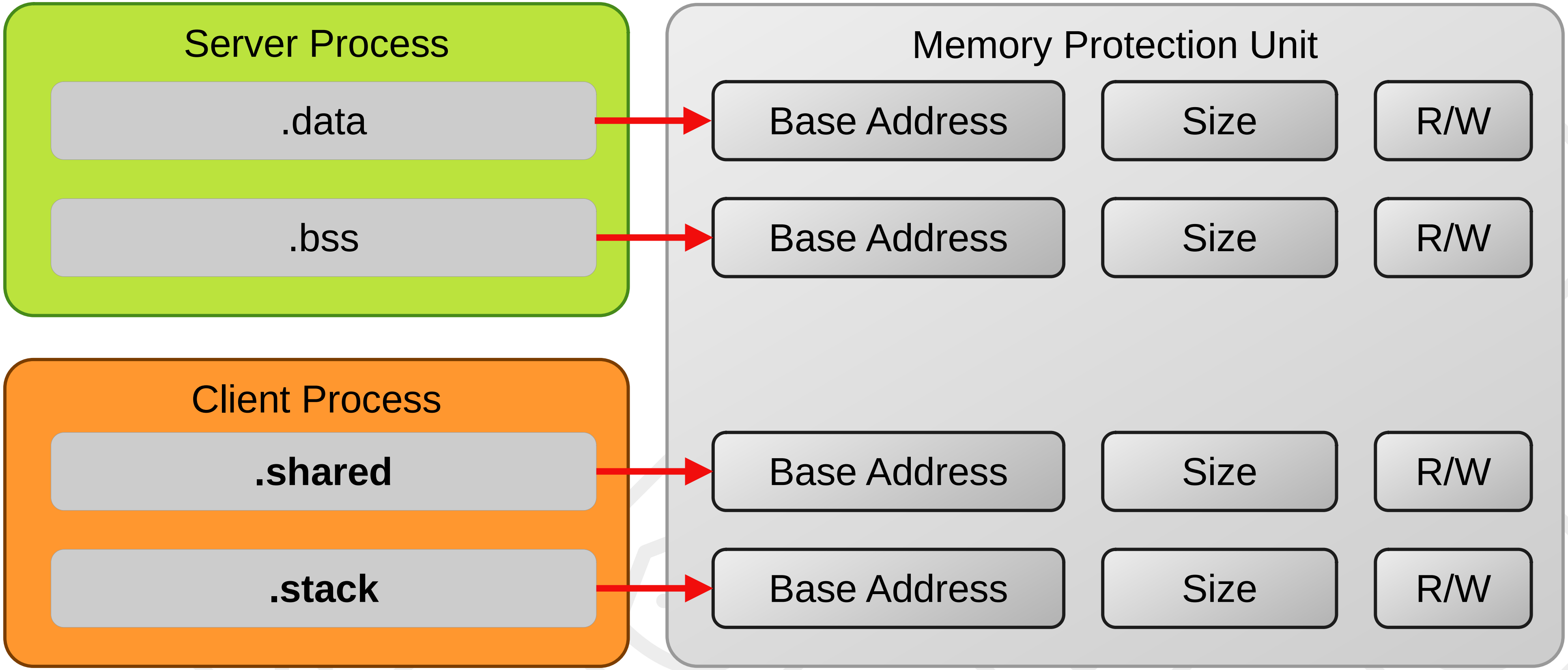
```c
/* Variables that are only accessible by the process */
#define BUFSIZE 512
SHARED static char buffer[BUFSIZE];

static int blinky_main()
{
    /* Do something interesting */
    return 0;
}

OS_APPLICATION(blinky);
OS_THREAD_CREATE(blinky, blinky_main, NULL, 32);
```

Remote Procedure Calling



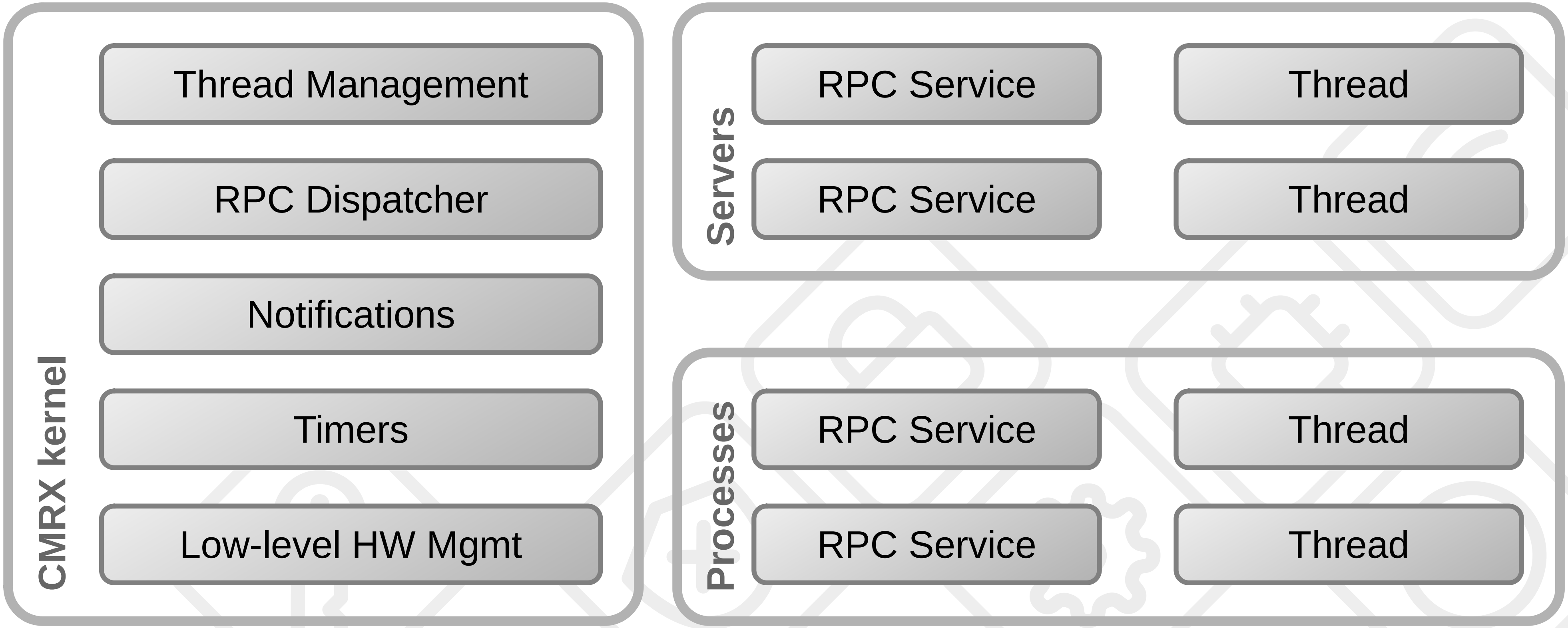
nopisonnope@gmail.com

RPC Enables Shared Memory



nopisonnope@gmail.com

CMRX microkernel



Why should I care?

- CRA recognizes memory isolation as technical requirement (ETSI EN 304 626 draft, TR-MISO)
- Memory isolation may* be required for certain devices
- **Device manufacturers** must *„Quickly inform users about incidents and provide mitigation measures.“*
- Stating that your product is not affected due component integration specifics **is valid mitigation measure**.
- Fixing vulnerability still mandatory.

nopisonnope@gmail.com

Is It All Roses?

- HALs, SDKs, drivers, stacks: Memory isolation? Huh?
- Portability: a curse word
- ARMv6/7M and RV32E MPU (possibly others) have limitations
- „We are doing it like this for 15 years“

nopisonnope@gmail.com

Demos

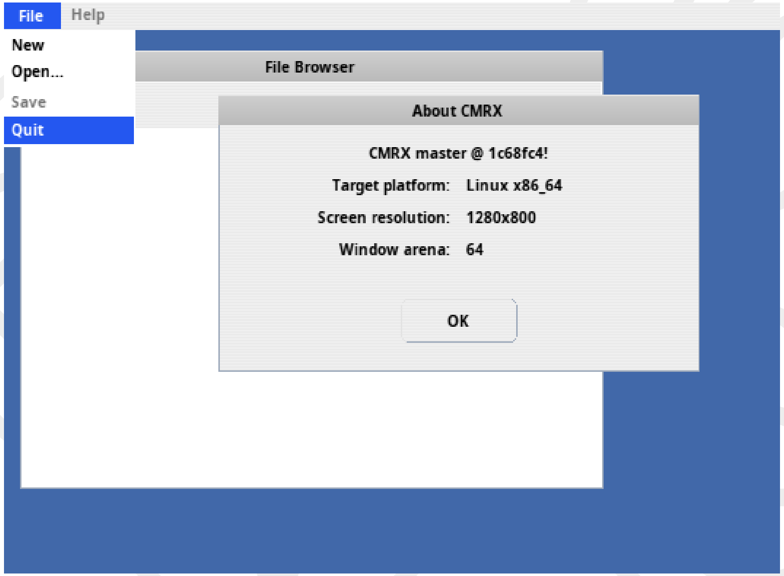
- Lean peripheral emulation

https://github.com/ventzl/cmrx-examples

- Linux-hello subdirectory
  - Emulated UART peripheral
  - Exported as UNIX socket

- Graphic desktop demo

https://github.com/ventzl/cmrx-linux-target



nopisonnope@gmail.com

Thank you!

Q&A

https://cmrxrtos.org

https://github.com/ventzl/cmrx

https://linkedin.com/in/eduard-drusa

nopisonnope@gmail.com