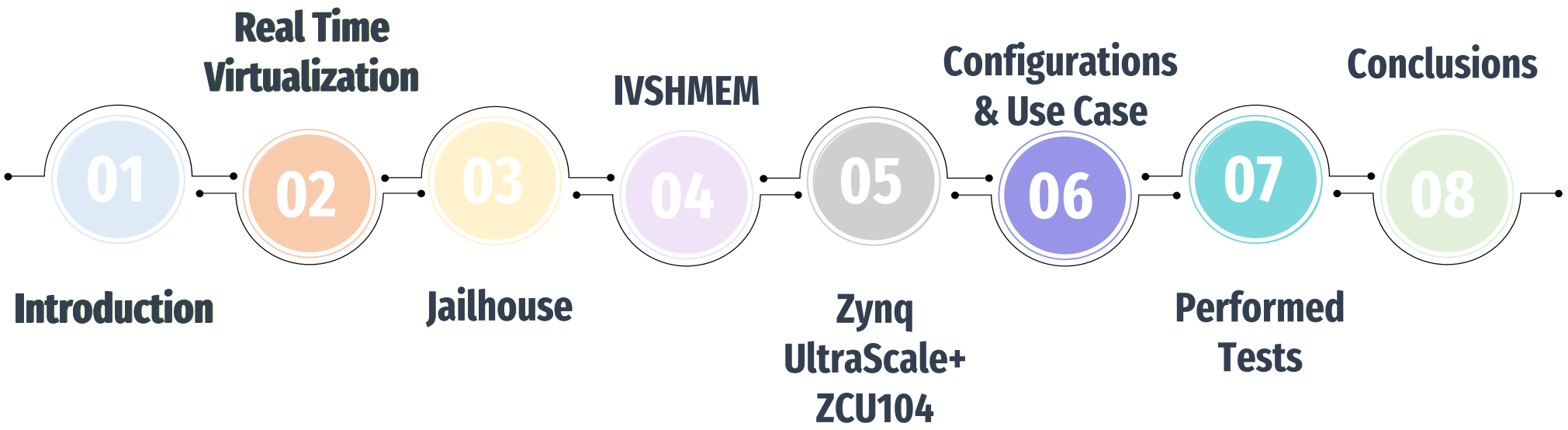
Università Degli Studi
di Napoli Federico II

IVSHMEM: Inter-VM communication mechanism on Jailhouse

Laura Naddei M63001356
Michele Spina M63001227

Roadmap

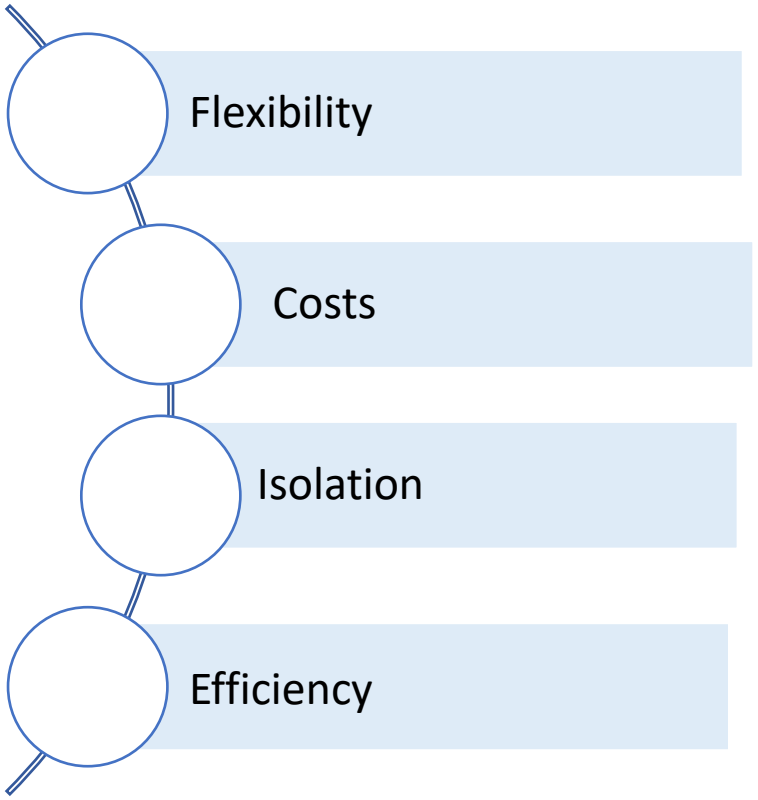

01
Introduction
02
Real Time Virtualization
03
Jailhouse
04
IVSHMEM
05
Zynq UltraScale+ ZCU104
06
Configurations & Use Case
07
Performed Tests
08
Conclusions

01 Introduction

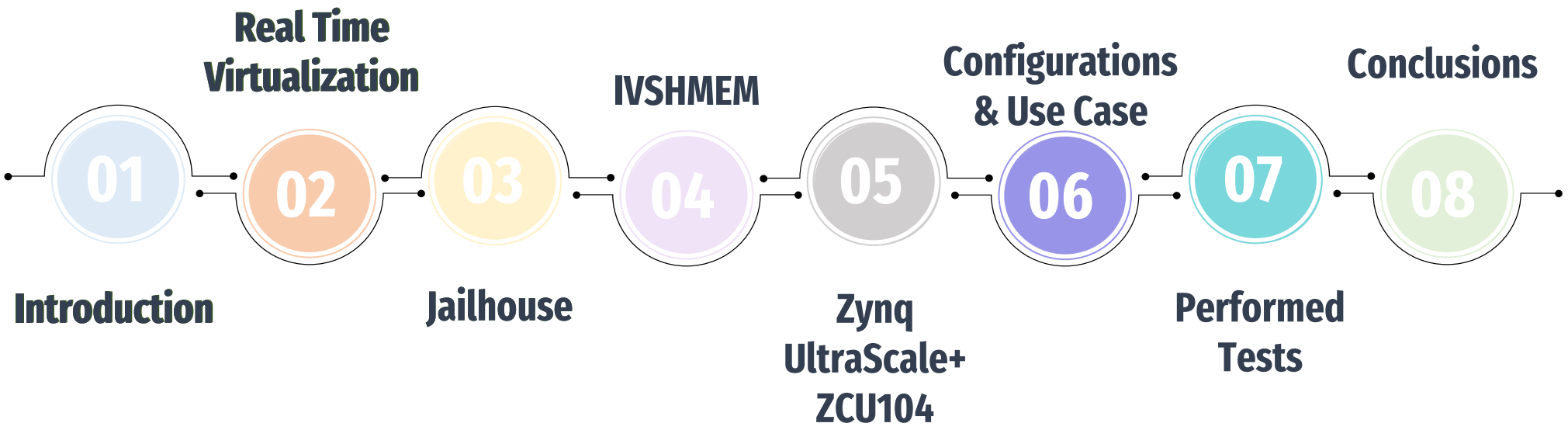
Virtualization technology is gaining space in the embedded domains and real time context thanks to the interest in mixed-criticality systems (MCS).

The power of virtualization in this context:

- the isolation of the environment of different criticality levels provides a solution to the main problem of MCS: the interferences between different parts of the system.
- it facilitates the development of software by decoupling from the real hardware reducing development cost.
- the encapsulation of high critical subsystem permits to certify this part independently of the other subsystems

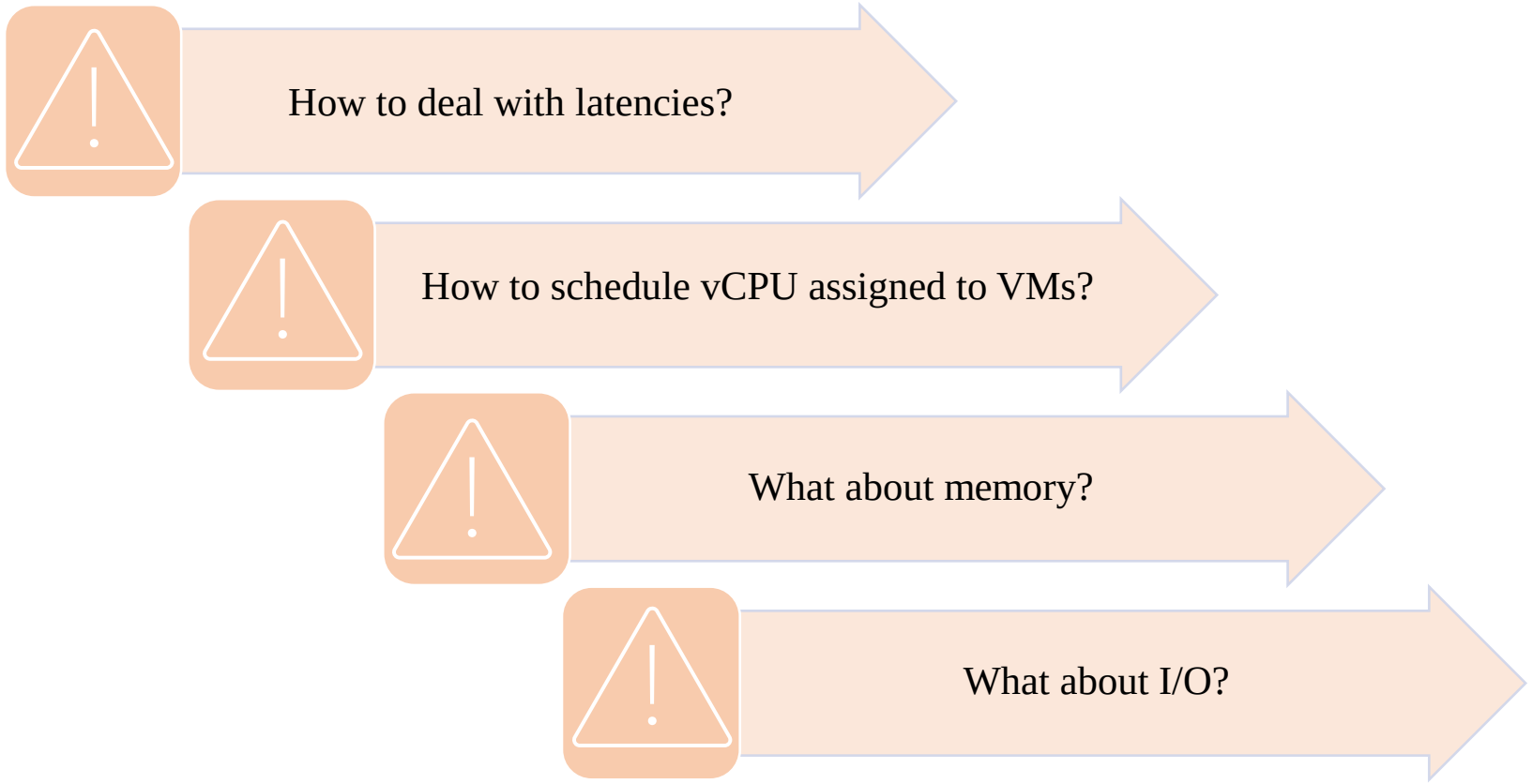


Roadmap


01
Introduction
02
Real Time Virtualization
03
Jailhouse
04
IVSHMEM
05
Zynq UltraScale+ ZCU104
06
Configurations & Use Case
07
Performed Tests
08
Conclusions

02 Real Time Virtualization

To guarantee hard deadlines constraints, both hypervisor and guest operating system must support ***real-time*** scheduling and properties but…



Real Time Virtualization

This four principal problems are differently dealt by three main hypervisor categories:

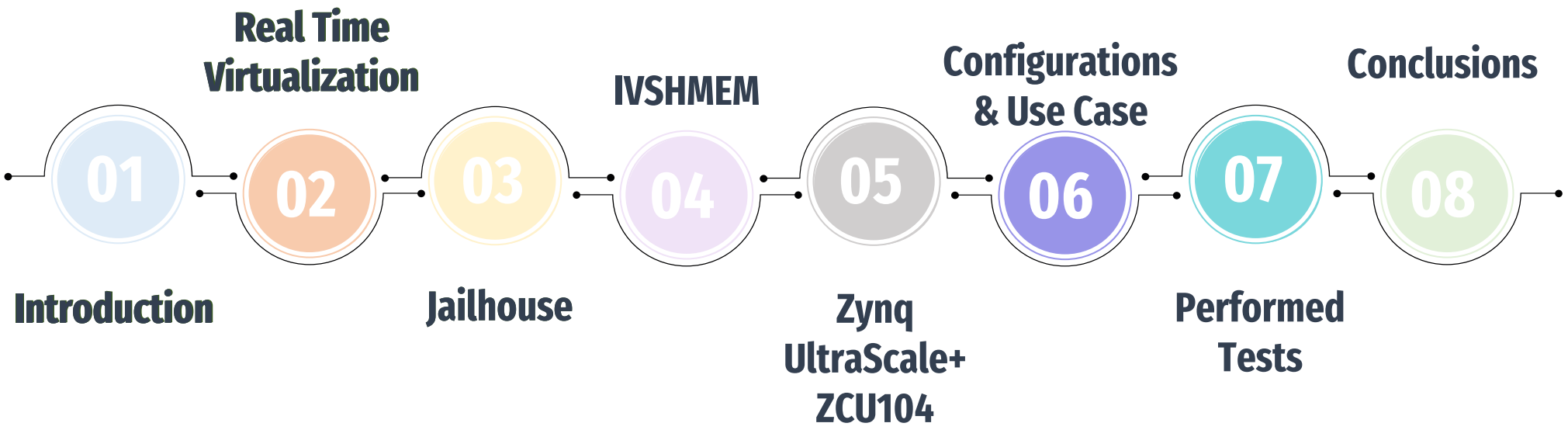
1. ***Microkernel and separation kernel*** face the problems with a static approach: Jailhouse.

2. ***Adaptation of general-purpose hypervisor***: Xen or KVM.

3. ***HV based on specific hardware features*** such as ARM TrustZone, that provides two state of execution, secure and non-secure.

Let's focus on ***Jailhouse!***

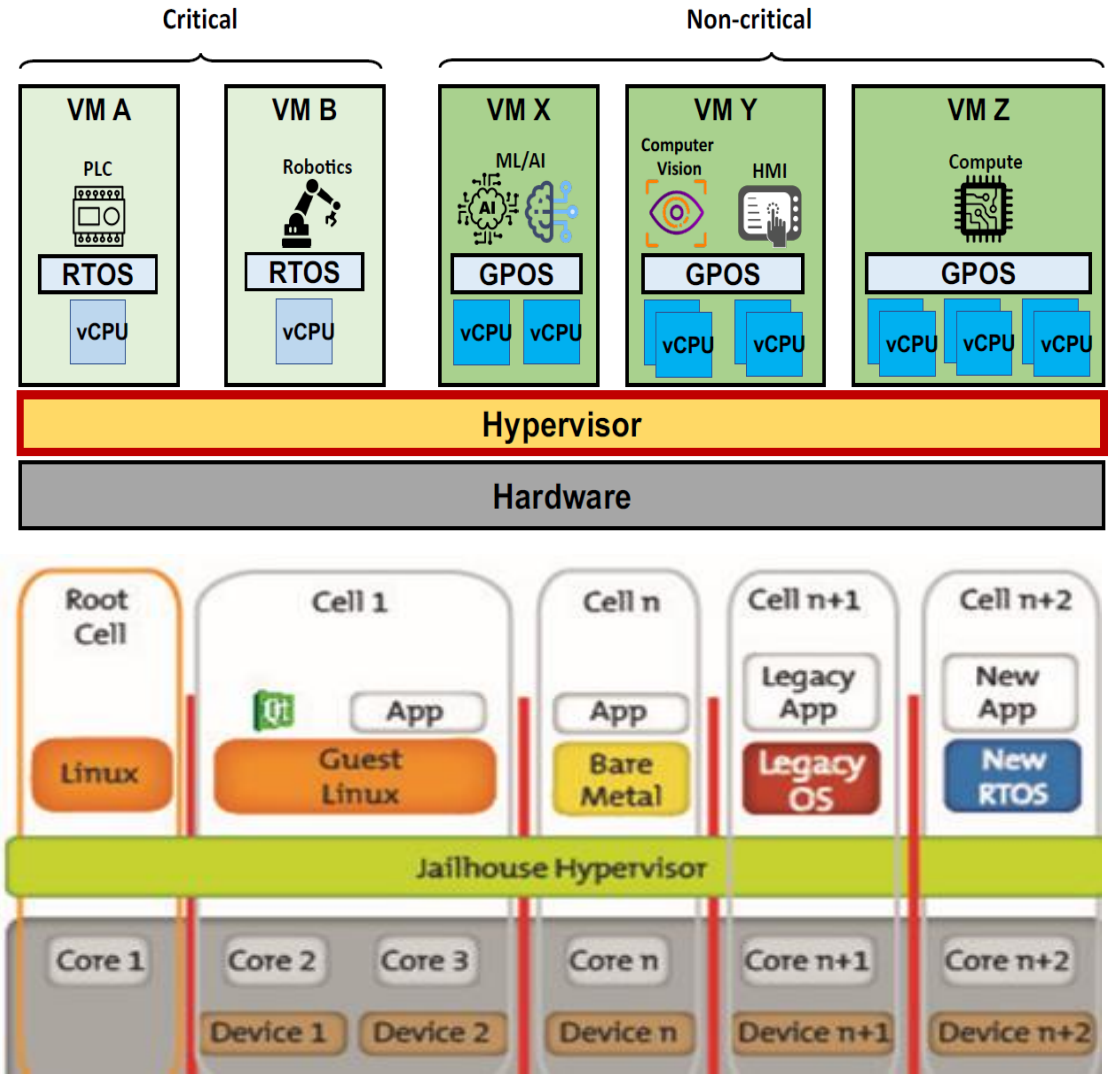
Roadmap


01
Introduction
02
Real Time Virtualization
03
Jailhouse
04
IVSHMEM
05
Zynq UltraScale+ ZCU104
06
Configurations & Use Case
07
Performed Tests
08
Conclusions

Jailhouse is a static partitioning hypervisor based on Linux that allows partitioning of a single physical platform into multiple isolated environments called "*cells*".

The hypervisor is initiated and loaded and takes the control of the system making Linux the "root cell", a cell that owns all the resources and is configured via configuration file.

All the cells non-root can be created by subtracting resources from root cell in base of configuration files.



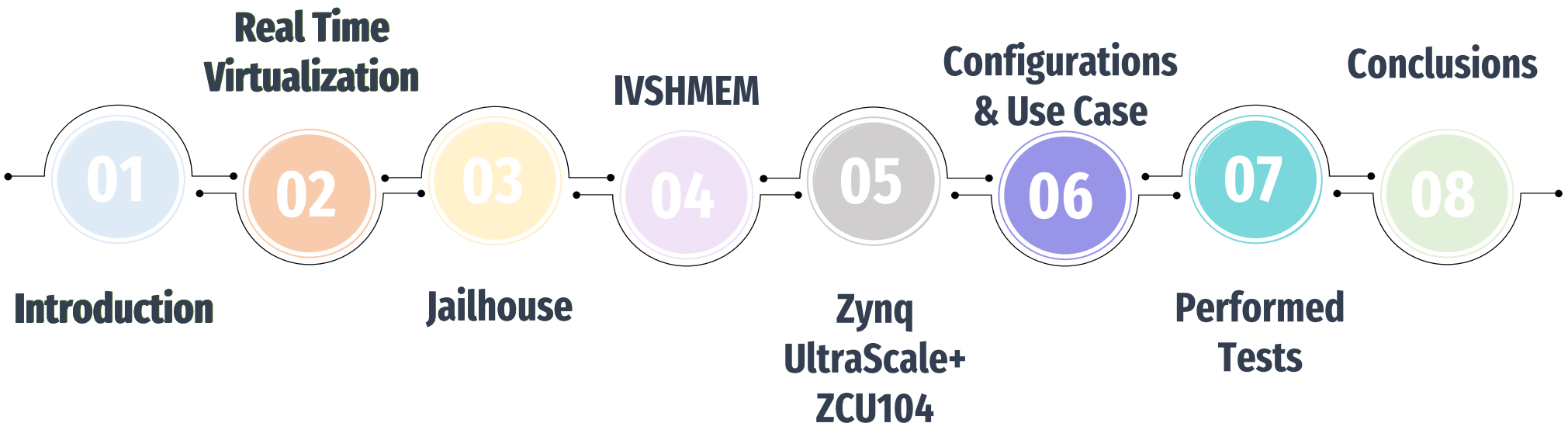
Jailhouse

Each cell with its own set of resources (CPU, memory, I/O devices), runs its own operating system (or bare metal) and applications and is isolated from the other cells and from the host system.

Jailhouse focuses on giving isolation to the applications, but this can be seen as a limitation as it may restrict communication between applications running in different partitions.

The ***IVSHMEM (Inter-VM Shared Memory)*** is a mechanism for communication between multiple virtual machines running on the same physical host.

Roadmap

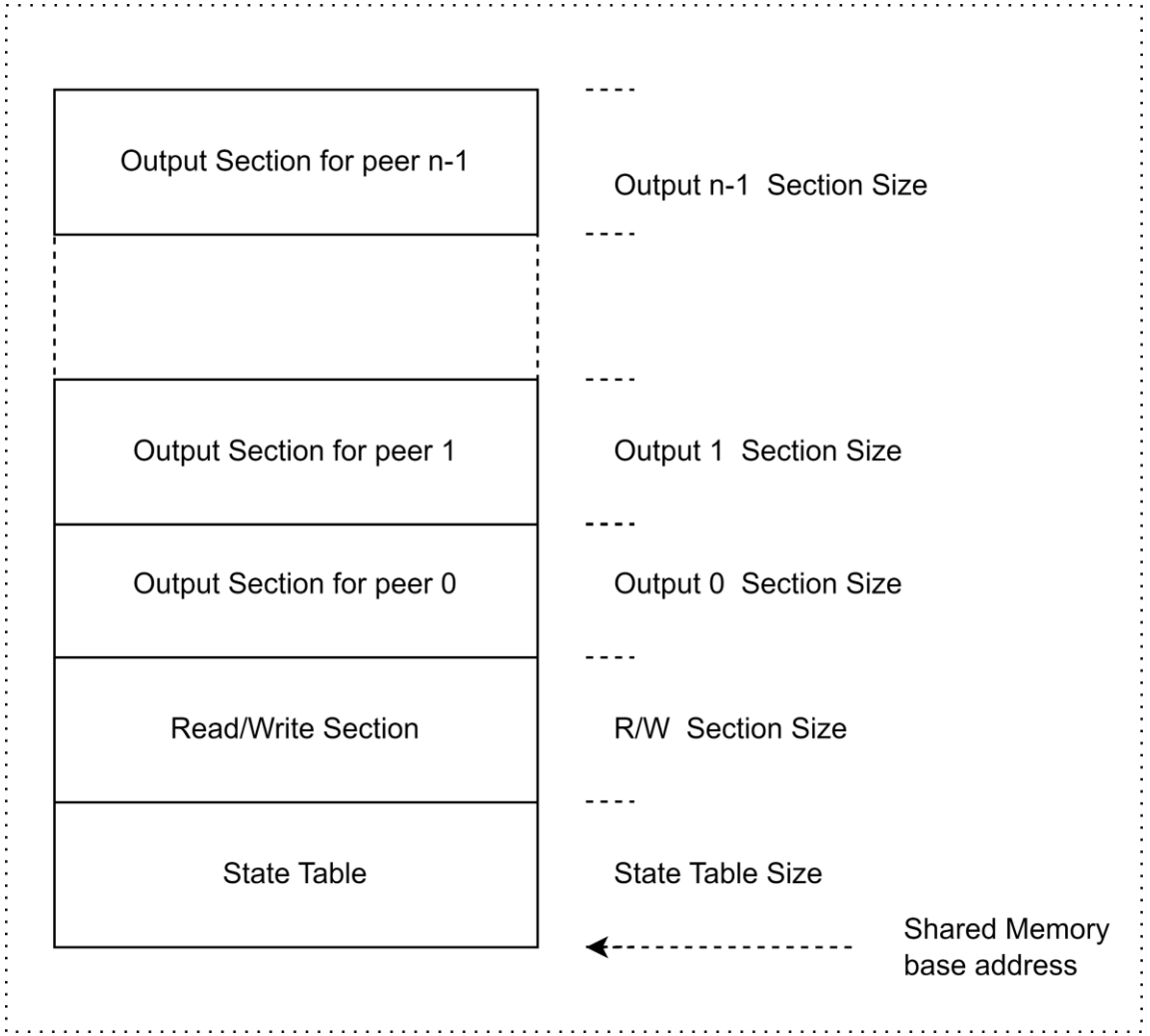

01
Introduction
02
Real Time Virtualization
03
Jailhouse
04
IVSHMEM
05
Zynq UltraScale+ ZCU104
06
Configurations & Use Case
07
Performed Tests
08
Conclusions

04 IVSHMEM

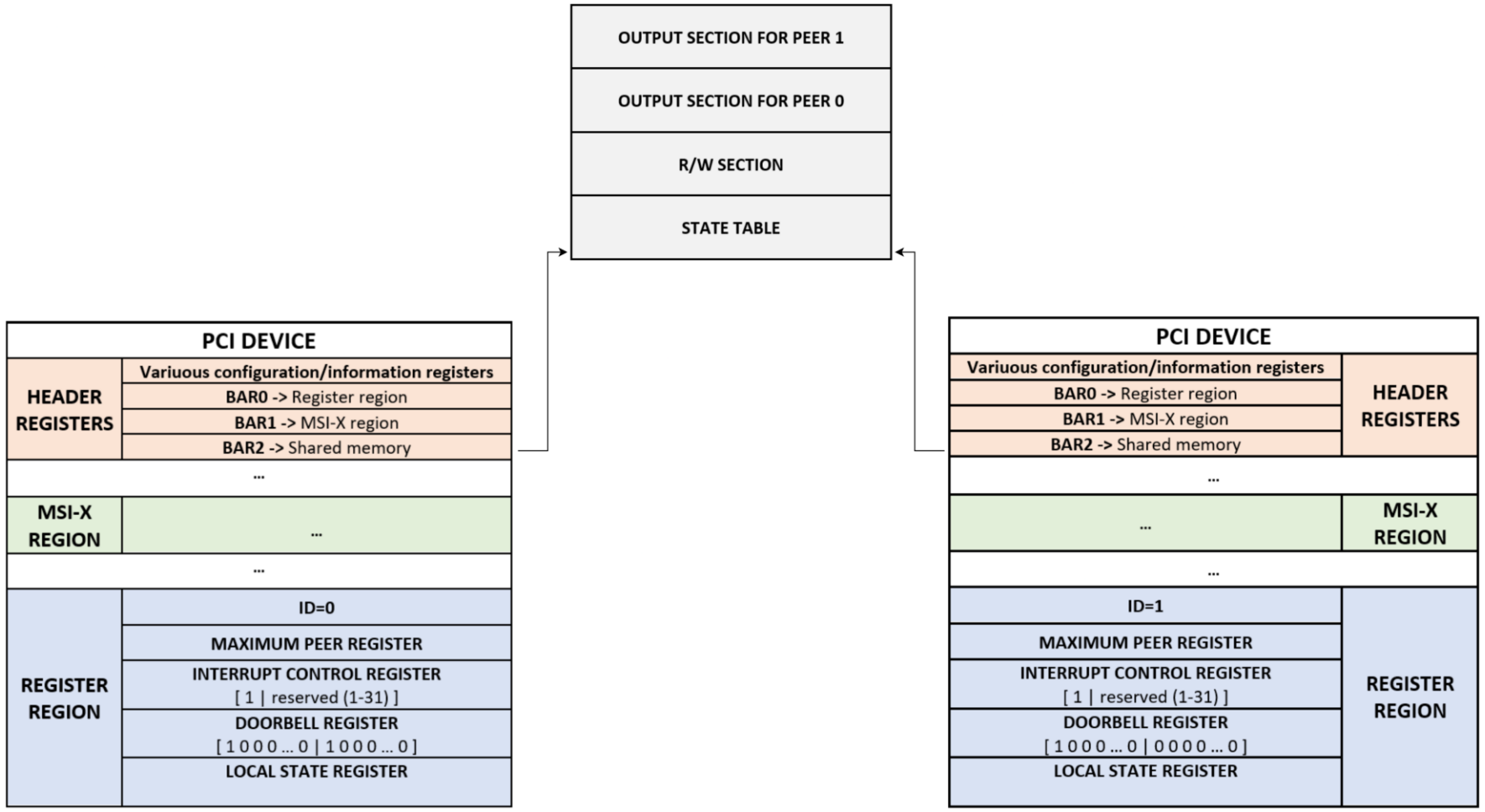
IVSHMEM's goal is to realize a memory area shared by the cells that is opaque to the hypervisor to give guests the complete control on it.

In order to set up a communication channel between two or more cells we have to add memory regions to each cell, in particular:

- Read-only region
- Common read/writable region for all peers
- Output regions



IVSHMEM


OUTPUT SECTION FOR PEER 1
OUTPUT SECTION FOR PEER 0
R/W SECTION
STATE TABLE
PCI DEVICE
HEADER REGISTERS
Variuous configuration/information registers
BAR0 -> Register region
BAR1 -> MSI-X region
BAR2 -> Shared memory
...
MSI-X REGION
...
...
REGISTER REGION
ID=0
MAXIMUM PEER REGISTER
INTERRUPT CONTROL REGISTER
[1 | reserved (1-31)]
DOORBELL REGISTER
[1 0 0 0 ... 0 | 1 0 0 0 ... 0]
LOCAL STATE REGISTER
PCI DEVICE
Variuous configuration/information registers
BAR0 -> Register region
BAR1 -> MSI-X region
BAR2 -> Shared memory
HEADER REGISTERS
...
...
MSI-X REGION
...
ID=1
MAXIMUM PEER REGISTER
INTERRUPT CONTROL REGISTER
[1 | reserved (1-31)]
DOORBELL REGISTER
[1 0 0 0 ... 0 | 0 0 0 0 ... 0]
LOCAL STATE REGISTER
REGISTER REGION

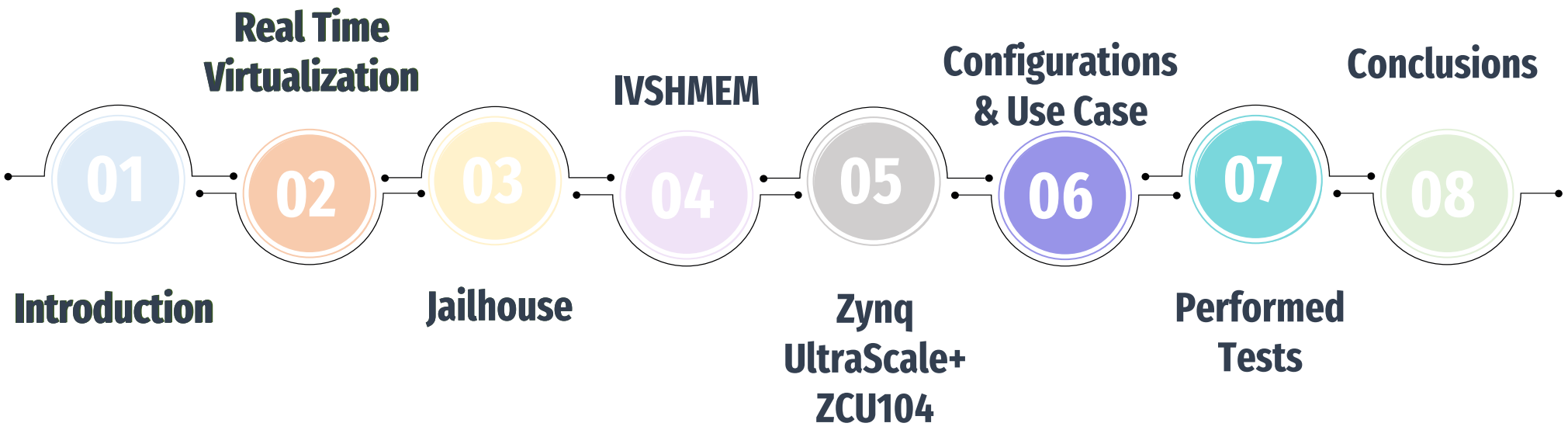
04 IVSHMEM : advantages

An important aspect of the IVSHMEM protocol is that it is based on pointers and offsets.

This approach has several ***advantages***:

- ✓ it allows for more flexible and dynamic memory allocation;
- ✓ it makes it easier to handle different memory layouts in different VMs;
- ✓ it provides a more secure mechanism for communication between VMs;

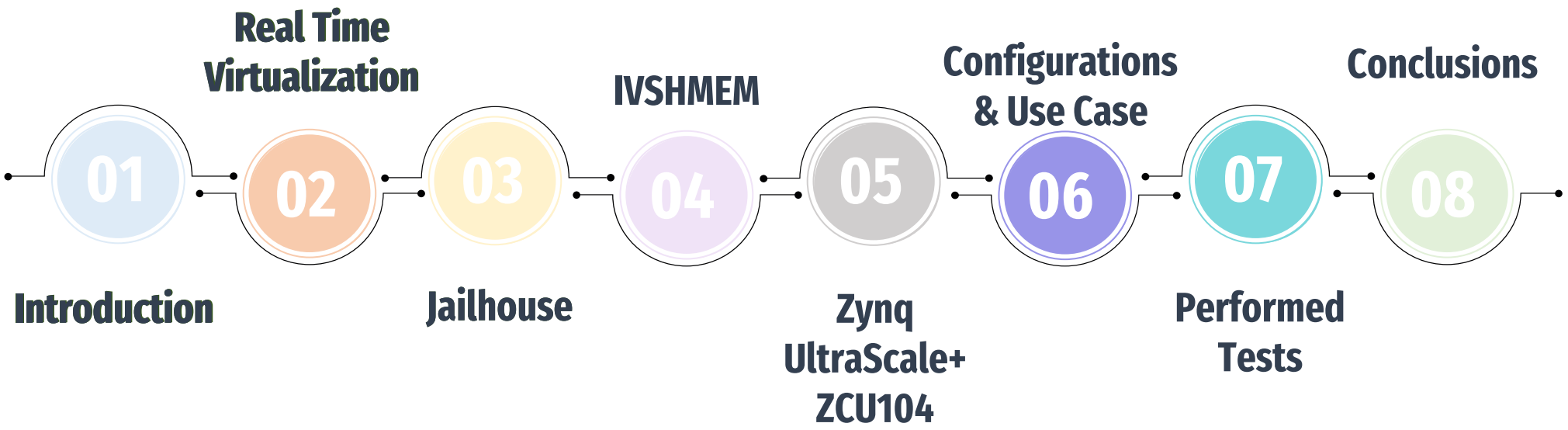
Roadmap


01
Introduction
02
Real Time Virtualization
03
Jailhouse
04
IVSHMEM
05
Zynq UltraScale+ ZCU104
06
Configurations & Use Case
07
Performed Tests
08
Conclusions

05 Zynq UltraScale+ ZCU104

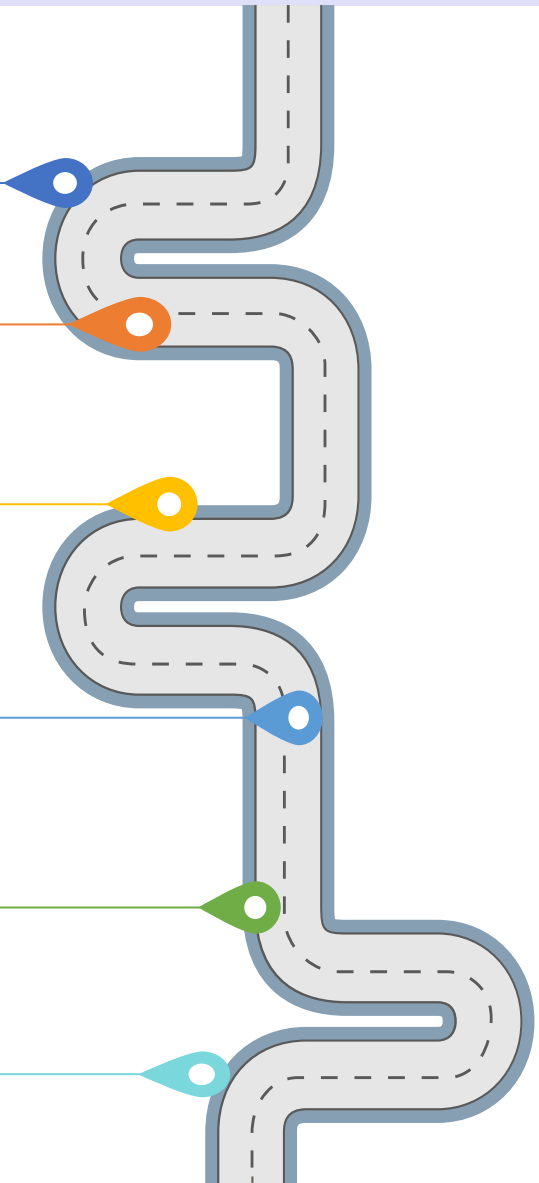
- Quad-core ARM® Cortex™-A53 applications processor
- Dual-core Cortex-R5 real-time processor
- Mali™-400 MP2 graphics processing unit
- 4KP60 capable H.264/H.265 video codec
- 16nm FinFET+ programmable logic.

Roadmap


01
Introduction
02
Real Time Virtualization
03
Jailhouse
04
IVSHMEM
05
Zynq UltraScale+ ZCU104
06
Configurations & Use Case
07
Performed Tests
08
Conclusions

06 Steps to follow

1. Build Linux image with Petalinux 2019.1 on an Ubuntu 18.04 VM
2. Cross-compile Jailhouse
3. Make SD card bootable for the *Zynq Ultrascale+ ZCU104*
4. Define cells configuration with the appropriate memory regions for using the *IVSHMEMv2* protocol
5. Implement the use case
6. Perform Tests



06 Real - Problems!

NON-root cell task with ~~FreeRTOS~~ → Bare-Metal

IVSHMEM mechanism based on ~~INTERRUPTS~~ → Pointer-based mechanism

Driver of vPCI for Linux using Kiszka's Linux kernel ~~UIO driver~~» → But even bare-metal interrupts don't work so..

Wrong address assignment to the IVSHMEM: some locations are written with ~~uncontrolled~~ values → Avoid the overlap of the root cell RAM section with the shared one

Successful detection of changes due to writing in R/W region only in singular conditions: Missed ~~Cache Coherence~~ → Flush the cache after every write

→ **One of major LIMITATIONS of IVSHMEM!!**

06 Cell Configuration


```
.mem_regions = {
    /* IVSHMEM shared memory regions*/
    // state region
    {
        .phys_start = 0x7fb00000,
        .virt_start = 0x7fb00000,
        .size = 0x1000,
        .flags = JAILHOUSE_MEM_READ,
    },
    // R/W region
    {
        .phys_start = 0x7fb00000 + 0x1000,
        .virt_start = 0x7fb00000 + 0x1000,
        .size = 0x4000,
        .flags = JAILHOUSE_MEM_READ | JAILHOUSE_MEM_WRITE,
    },
    //OUTPUT ROOT
    {
        .phys_start = 0x7fb00000 + 0x5000,
        .virt_start = 0x7fb00000 + 0x5000,
        .size = 0x4000,
        .flags = JAILHOUSE_MEM_READ | JAILHOUSE_MEM_WRITE,
     },
    //OUTPUT INMATE
    {
        .phys_start = 0x7fb00000 + 0x9000,
        .virt_start = 0x7fb00000 + 0x9000,
        .size = 0x4000,
        .flags = JAILHOUSE_MEM_READ,

     },
```

```
        /* MMIO (permissive) */ {
            .phys_start = 0xfd000000,
            .virt_start = 0xfd000000,
            .size =       0x03000000,
            .flags = JAILHOUSE_MEM_READ | JAILHOUSE_MEM_WRITE |
                JAILHOUSE_MEM_IO,
        },
        /* RAM */ {
            .phys_start = 0x00000000,
            .virt_start = 0x00000000,
            .size =       0x7fb00000,
            .flags = JAILHOUSE_MEM_READ | JAILHOUSE_MEM_WRITE |
                JAILHOUSE_MEM_EXECUTE,
        },
    },

    .pci_devices = {
        /* 0001:00:01.0 */ {
            .type = JAILHOUSE_PCI_TYPE_IVSHMEM,
            .domain = 1,
            .bdf = 1 << 3,
            .bar_mask = JAILHOUSE_IVSHMEM_BAR_MASK_INTX,
            .shmem_regions_start = 0,
            .shmem_dev_id = 0,
            .shmem_peers = 2,
            .shmem_protocol = JAILHOUSE_SHMEM_PROTO_UNDEFINED,
        },
    },
```

06 Use Case

1. The non-root cell and the Linux controller initialize the pointers to the readable/writable region;
2. Linux waits for the *keepalive* (in polling);
3. The non-root cell periodically writes in the R/W region and flushes the cache;
4. Linux reads the R/W region and lowers the keepalive signal;
5. Linux returns to polling;

Instead, if the non-root cell fails and doesn't send the *keepalive* signal, the root cell will restart the non-root cell.

```
//printk("[BM]\Start send ACK...\n");
for(int i=0; i<3; i++){
    dev.rw_section[PING] = ACK;
    arm_dcaches_flush((void*)0x7fb01000,0x1000, DCACHE_CLEAN_AND_INVALIDATE);
    delay_us(1000);
}
```

```
    printf("[LNX]\n-----Start ping-----\n");

    printf("[LNX] \nWaiting for response...\n");
    int rip=0;
//  system("./stress-ng --class cpu-cache --all 1 --metrics-brief -t6000 &");

    while(rip<5475){
        int s=0;

        while(mem[PING] != 1 && s<3){
            usleep(1000*5);
            s++;
        }
        if(s<3){
            printf("[LNX] \nResponse received! Changing flag...\n");
            mem[PING] = 0;
        } else {
            printf("\nResponse not received in time! I am about to restore the cell...\n");

            clock_gettime(CLOCK_REALTIME, &t1);
            system("sh destroy.sh"); //a simple script that perform the restore by using destroy or shutdown
            while(mem[PING] != 1);
            clock_gettime(CLOCK_REALTIME, &t2);


            timespec_diff(&t2,&t1,&t3);

            ns = (t3.tv_sec * 1000000000) + t3.tv_nsec;

            FILE *fpt = fopen("time.txt", "a");
            fprintf(fpt,"%lld\n",ns);
            fclose(fpt);
            rip++;
            while(mem[PING] != 1);
        }
    }
```

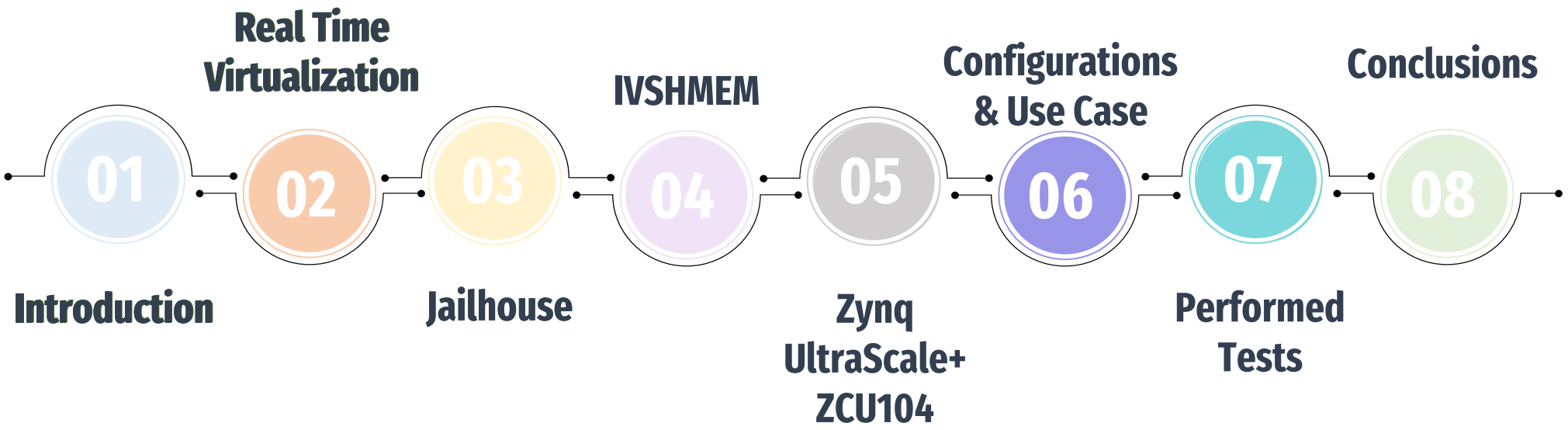
06 Demonstration


```
2 secondi
[LNX]
3 secondi
[LNX]
4 secondi
[BM]
Invio ack #2
[LNX]
5 secondi
[LNX]
Risposta ricevuta! Cambio il flag...
[LNX]
1 secondi
[LNX]
2 secondi
[LNX]
3 secondi
[LNX]
4 secondi
[BM]
Invio ack #3
[LNX]
5 secondi
[LNX]
Risposta ricevuta! Cambio il flag...
[LNX]
1 secondi
[LNX]
2 secondi
[LNX]
3 secondi
[LNX]
4 secondi
[BM]
Invio ack #4
```

```
[LNX]
5 secondi
[LNX]
6 secondi

Risposta non ricevuta in tempo! Sto per lanciare il comando destroy
Closing cell "inmate-demo"
Page pool usage after cell destruction: mem 60/993, remap 5/131072
[ 1621.175716] Detected VIPT I-cache on CPU3
[ 1621.175772] CPU3: Booted secondary processor 0x0000000003 [0x410fd034]
[ 1621.186515] Destroyed Jailhouse cell "inmate-demo"
[ 1621.211724] CPU3: shutdown
[ 1621.214420] psci: CPU3 killed.
Adding virtual PCI device 00:01.0 to cell "inmate-demo"
Shared memory connection established, peer cells:
 "ZCU104-root"
Created cell "inmate-demo"
Page pool usage after cell creation: mem 76/993, remap 5/131072
[ 1621.236411] Created Jailhouse cell "inmate-demo"
Cell "inmate-demo" can be loaded
Started cell "inmate-demo"
IVSHMEM: Found device at 00:01.0
IVSHMEM: bar0 is at 0x00000000ff000000
IVSHMEM: bar1 is at 0x00000000ff001000
IVSHMEM: ID is 276
IVSHMEM: max. peers is 32
IVSHMEM: state table is at 0x000000007fb00000
IVSHMEM: R/W section is at 0x000000007fb01000
IVSHMEM: input sections start at 0x000000007fb05000
IVSHMEM: output section is at 0x000000007ff55000
IVSHMEM: initialized device
[BM]
Inizio invio ack...
[BM]
Invio ack #0
[LNX]
1 secondi
[LNX]
Risposta ricevuta! Cambio il flag...
[LNX]
```

Roadmap


01
Introduction
02
Real Time Virtualization
03
Jailhouse
04
IVSHMEM
05
Zynq UltraScale+ ZCU104
06
Configurations & Use Case
07
Performed Tests
08
Conclusions

07 Performed Tests

In the context of a *"keepalive"* protocol between two cells, we tested the **recovery time**, which is the time between the failure of the bare-metal cell and the restoration of the keepalive protocol.

Four kind of tests has been made:

1. **Shutdown**: the bare metal cell is restored using the *"jailhouse cell shutdown 1"* command in a non-stressed environment.

2. **Shutdown with stress:** the bare metal cell is restored using the *shutdown* command in a cpu-cache stressed environment obtained using the *"stress-ng --class cpu-cache --all 1"* command .

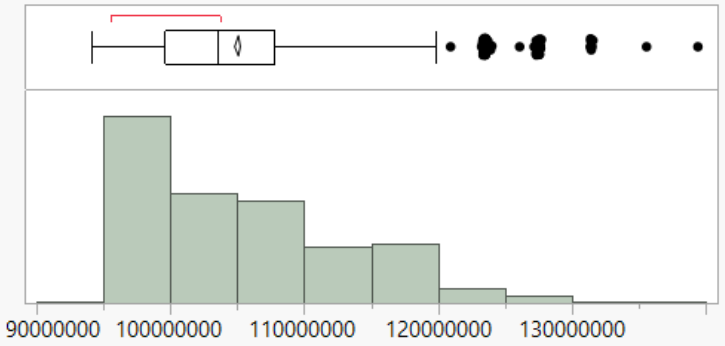
3. **Destroy:** the bare metal cell is restored using the *"jailhouse cell destroy 1"* command and then re-created in a non-stressed environment.

4. **Destroy with stress:** the bare metal cell is restored using the *destroy* command and then re-created in a cpu-cache stressed environment obtained using the *"stress-ng --class cpu-cache --all 1"* command .

07 Data distributions

Distribuzioni

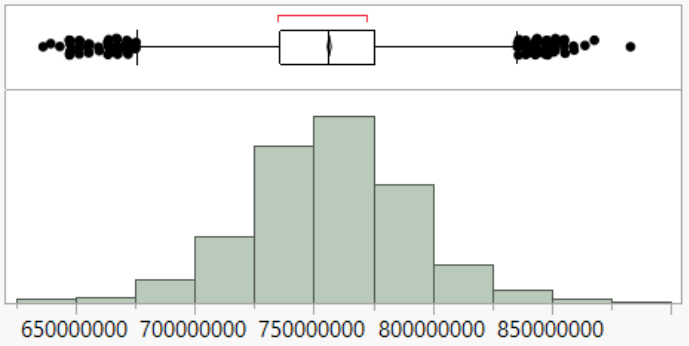
Time Destroy



Statistiche di riepilogo

Media	105059410
Dev std	7867144,3
Errore std della media	107326,93
Media superiore al 95%	105269814
Media inferiore al 95%	104849006
N	5373

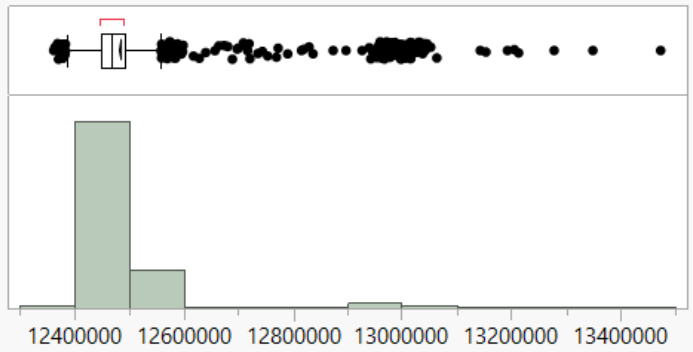
Time Destroy CPU Cache Stress



Statistiche di riepilogo

Media	756455125
Dev std	32566233
Errore std della media	444282,39
Media superiore al 95%	757326099
Media inferiore al 95%	755584152
N	5373

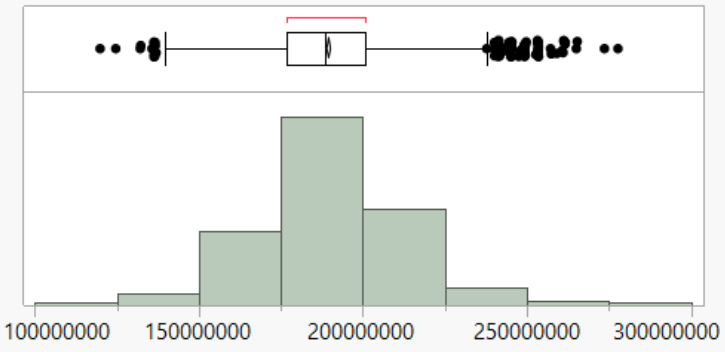
Time Shutdown



Statistiche di riepilogo

Media	12484274
Dev std	90833,931
Errore std della media	1239,4258
Media superiore al 95%	12486703
Media inferiore al 95%	12481844
N	5371

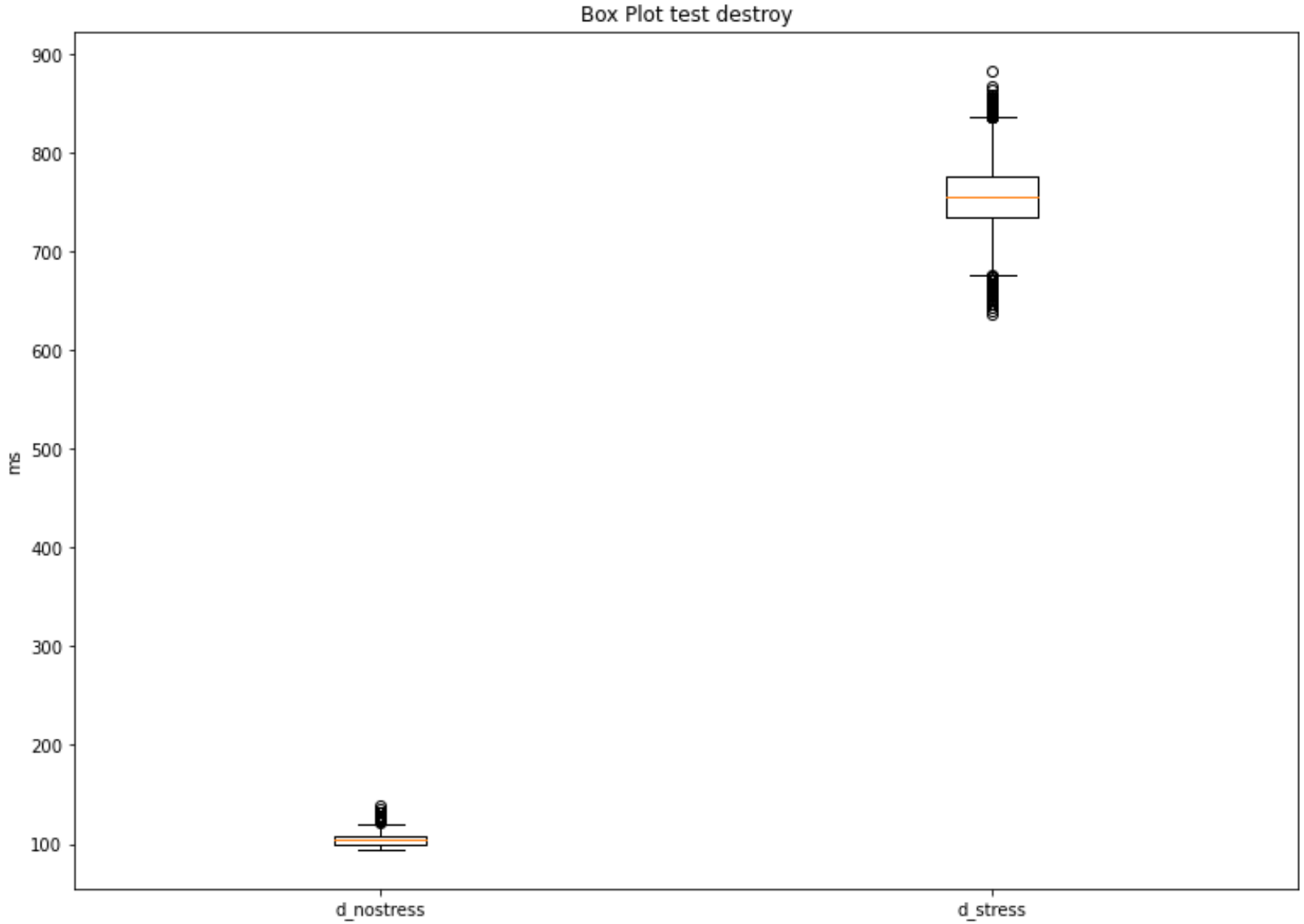
Time Shutdown CPU Cache Stress



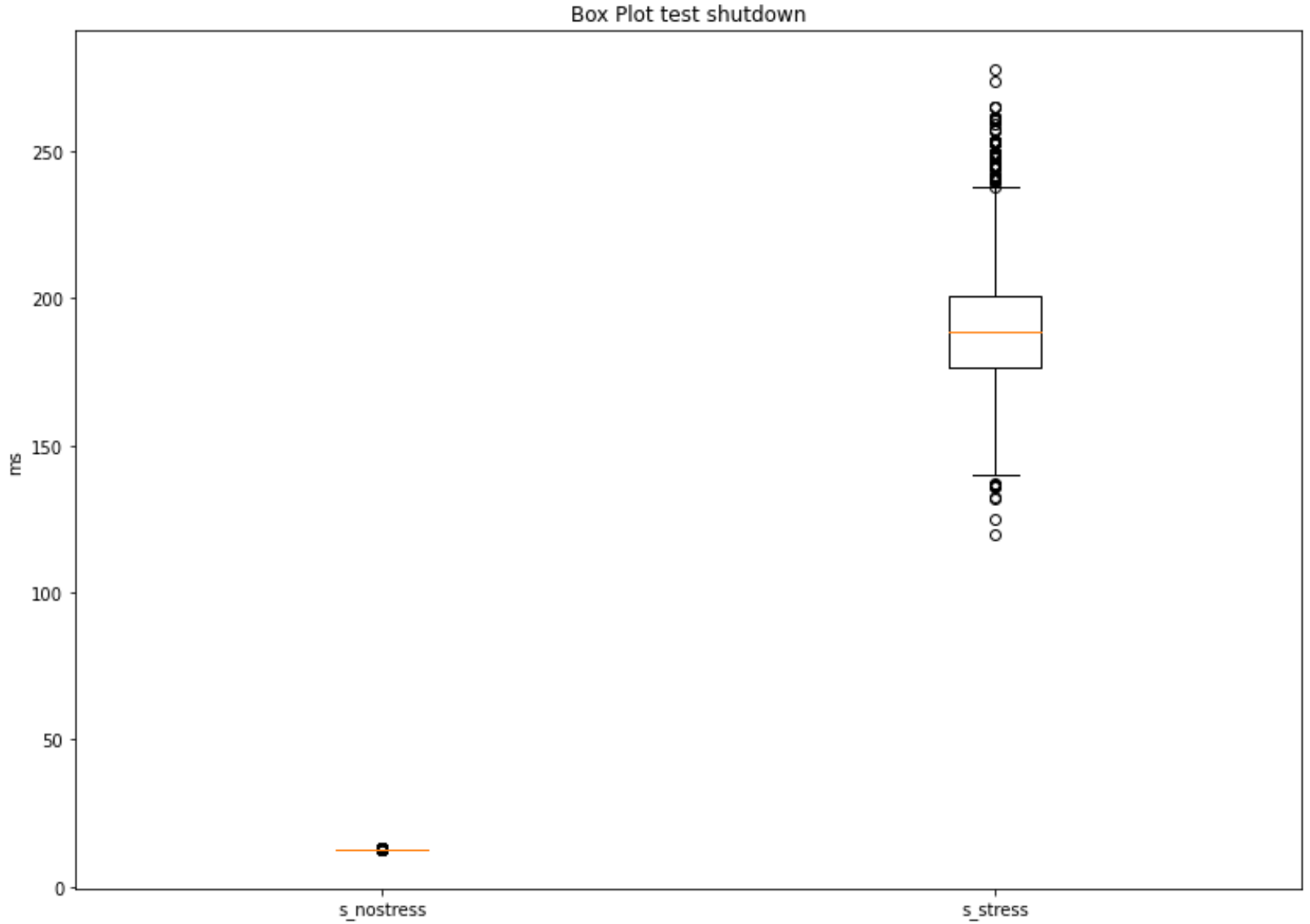
Statistiche di riepilogo

Media	189501130
Dev std	20600876
Errore std della media	281045,91
Media superiore al 95%	190052094
Media inferiore al 95%	188950166
N	5373

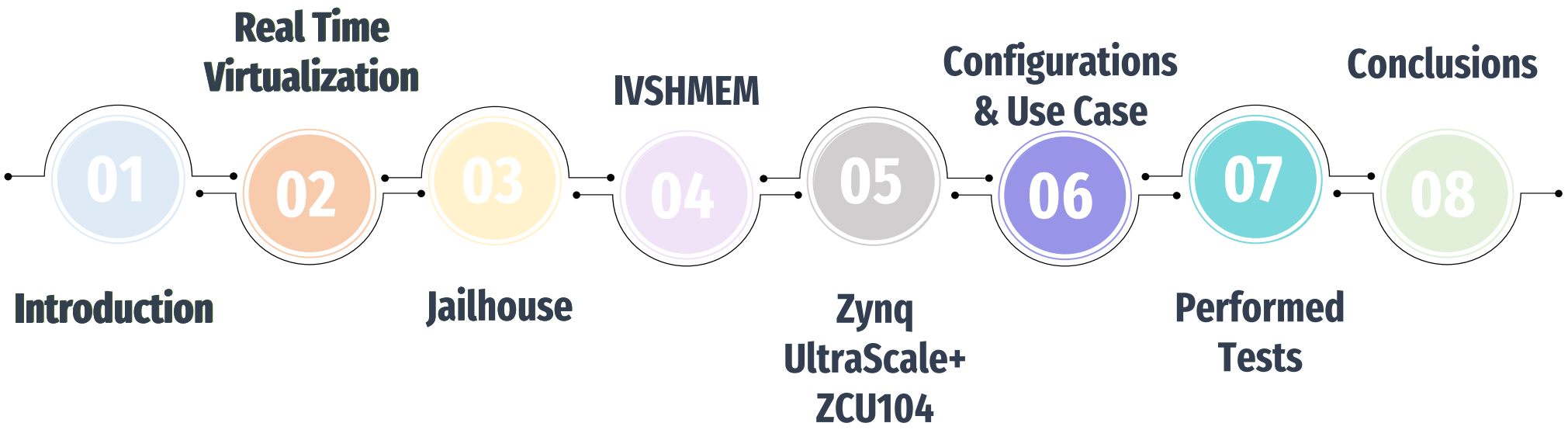
07 Result analysis



Result analysis



Roadmap


01
Introduction
02
Real Time Virtualization
03
Jailhouse
04
IVSHMEM
05
Zynq UltraScale+ ZCU104
06
Configurations & Use Case
07
Performed Tests
08
Conclusions

08 Conclusions

There're many **open issues** on this topic, in particular because of *ARM64* architecture.

The IVSHMEMv2 protocol is theoretically well founded and undoubtedly opens several possibilities.
The keepalive protocol is just one of the possibilities but has proven to be a tricky implementation because of the cache's failure to writeback. This limits the implementation of reactive protocols between two cells.

Tests showed significant slowdown by the hypervisor (on the order of x7 and x15) in cell recovery under cpu-cache stress conditions, this is a major limitation in a Mixed Criticality Context.
This can certainly be mitigated by the use of cache coloring coupled with a bandwidth control on the data bus.

However, a crucial aspect is the **low compatibility of *IVSHMEM* with *ARM64* architectures**, a problem also recognized by the Jailhouse community, and which has prevented us from using the protocol as it was designed, i.e., with interrupt mechanisms that we were unable to use.

Thank You

For The Attention!