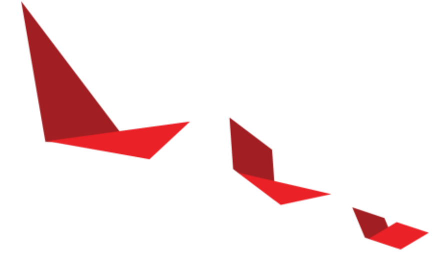
Xen Cache Coloring & Real-Time

Stefano Stabellini
Marco Solieri
Luca Miccio
Giulio Corradi

© Copyright 2020 Xilinx

Xen in Embedded



- Consolidation and Componentization
- Isolation
  - Security
  - Reliability
  - Safety
  - **Interference & Real-Time**

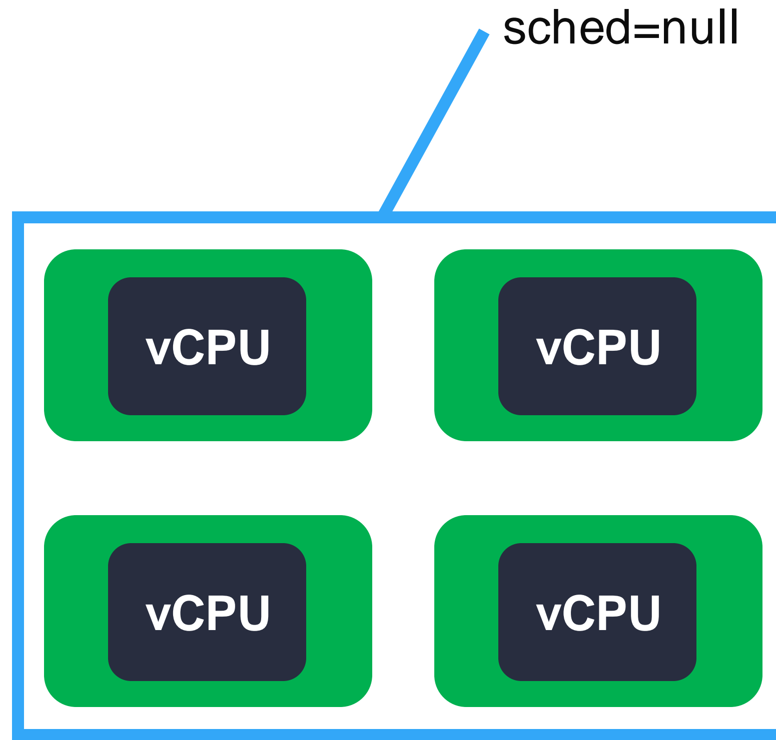
© Copyright 2020 Xilinx

Reference Example

- Static Configuration – 2-4 VMs
- No multitenancy – All VMs owned by the same user
- Direct assignment of resources
- No PV drivers
  - Maybe except for PV Console
  - A shared page to exchange data between VMs
- Baremetal DomU driving a soft-logic block with latency requirements

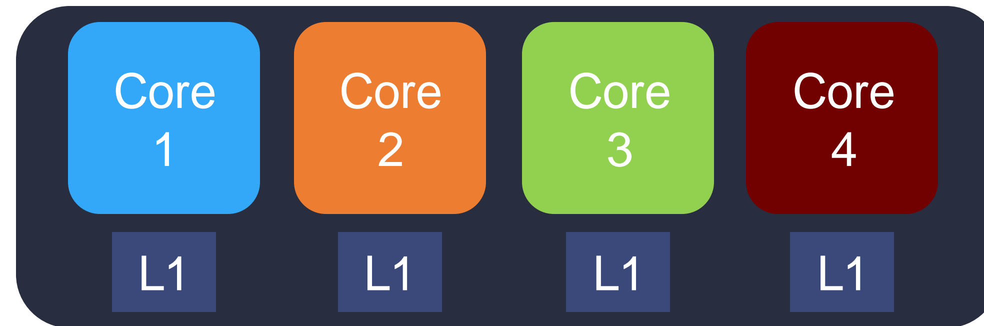
© Copyright 2020 Xilinx

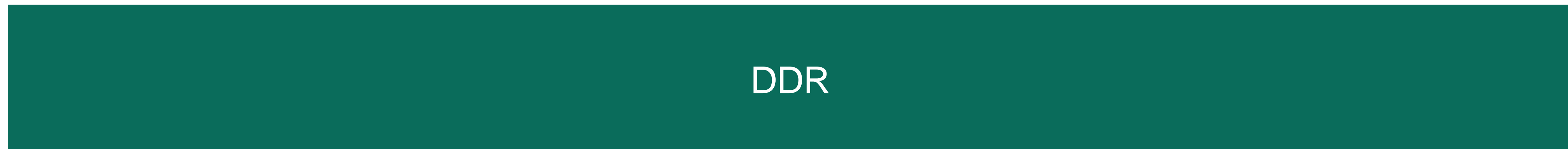
sched=null vwfi=native



© Copyright 2020 Xilinx

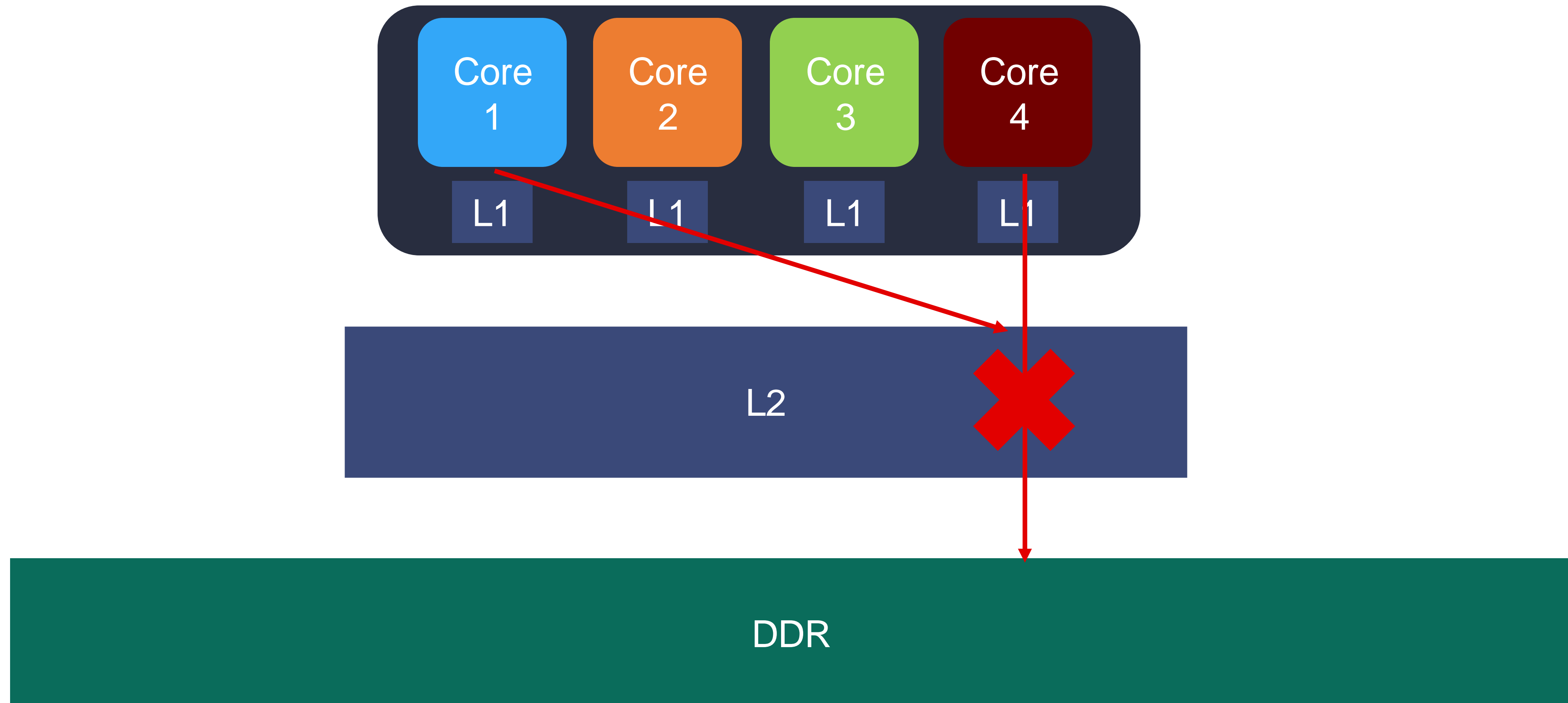
Shared L2 cache





© Copyright 2020 Xilinx

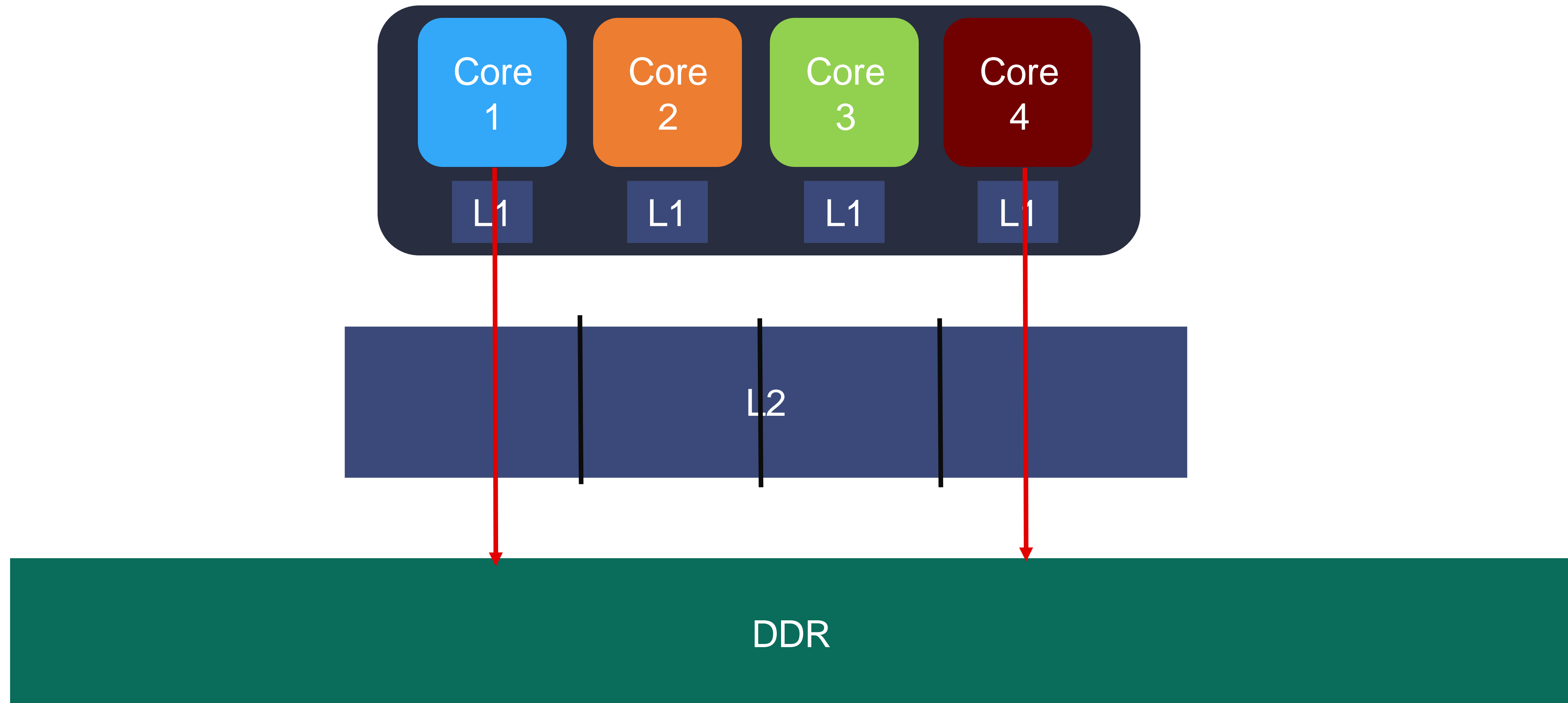
Shared L2 cache



© Copyright 2020 Xilinx

XILINX

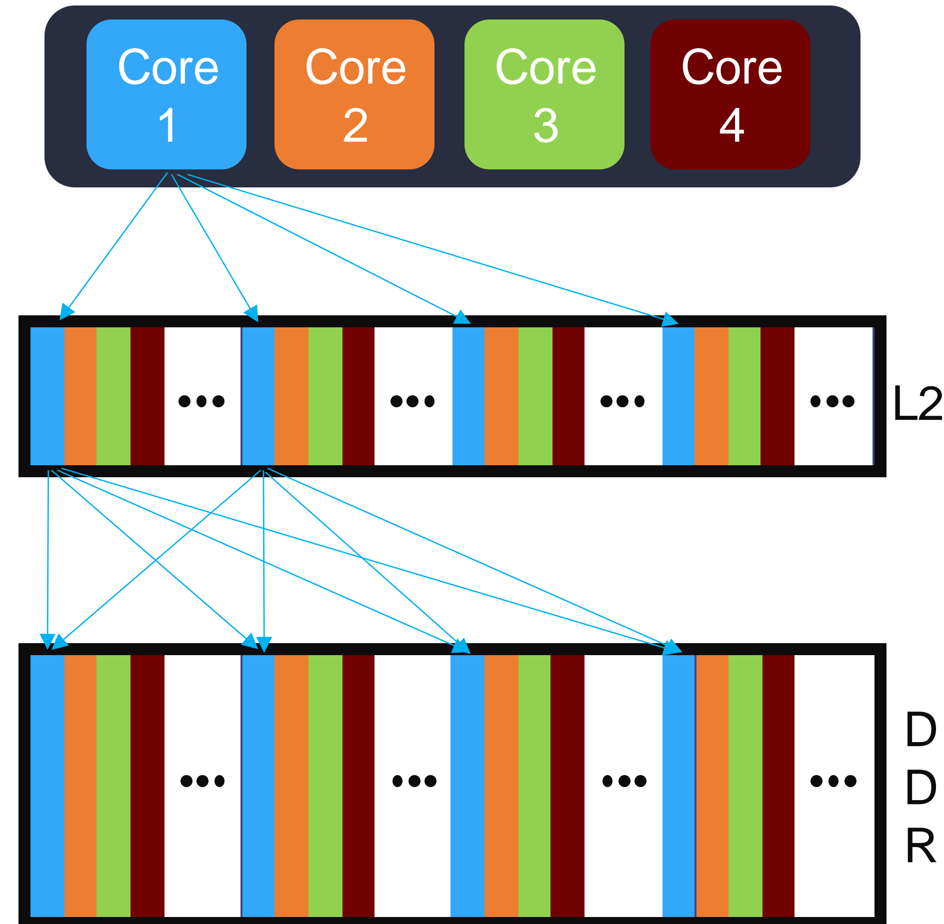
Shared L2 cache



© Copyright 2020 Xilinx

Xen Cache Coloring

- CPUs clusters often share L2 cache
- Interference via L2 cache affects performance
  - App0 running on CPU0 can cause cache entries evictions, which affect App1 running on CPU1
  - App1 running on CPU1 could miss a deadline due to App0's behavior
  - It can happen between Apps running on the same OS & between VMs on the same hypervisor
- Hypervisor Solution: Cache Partitioning, AKA **Cache Coloring**
  - Each VM gets its own allocation of cache entries
  - No shared cache entries between VMs
  - Allows real-time apps to run with deterministic IRQ latency



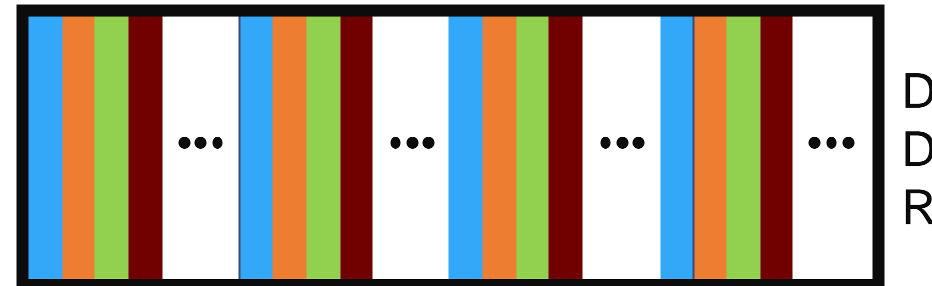
© Copyright 2020 Xilinx

XILINX.

Colors Calculation

- Detect Way Size
  - via registers
  - via xen command line

- Calculate colors masks and total number of available colors
  - Xilinx ZynqMP: way size is 65536
  - Order 16 -> color bitmask 0x**F**000 -> 16 colors



© Copyright 2020 Xilinx

Colored Allocation

- New colored memory allocator
  - In alternative to Buddy
  - Pages are stored by color in separate lists
- Can be used for Xen memory as well as Domain memory
- Xen relocates itself in colored memory

© Copyright 2020 Xilinx

Coloring Configuration

- Xen command line

```
way_size=65536 xen_colors=0 dom0_colors=1-6
```

- Dom0-less DomUs

```
fdt set /chosen/domU0 colors <0x0 0x80>
```

- xl domU config files

```
colors=["10-11"]
```

© Copyright 2020 Xilinx

Benchmarking

- Hardware: Xilinx MPSoC
- Workload: Bare Metal application
- Stress: Bare Metal application
- Test configuration:
  - Measured load: 1500 sums
  - Internal load: 8000 sums every 250us
  - Interrupt triggered every 8ms
  - Samples: 1000
- Colored configuration:
  - Xen: ["0,1"]
  - Motor Control DomU: ["4,7"]
  - Stress DomU: ["8,11"]
  - Stress2 DomU: ["12,15"]
  - Linux stress: ["2,3"]
- Linux stress command: stress -m 1 --vm-bytes 8M --vm-stride 64 --vm-keep -t 3600 &

© Copyright 2020 Xilinx

L1 vs L2 cache

- L1 cache can hide L2 cache interference effects in simple test scenarios
- Make sure not use the L1 cache by accident during the measurements

© Copyright 2020 Xilinx

Benchmark #1: Motor Control Execution Time

- Measure the time it takes to run a typical motor control execution routine, with and without interference

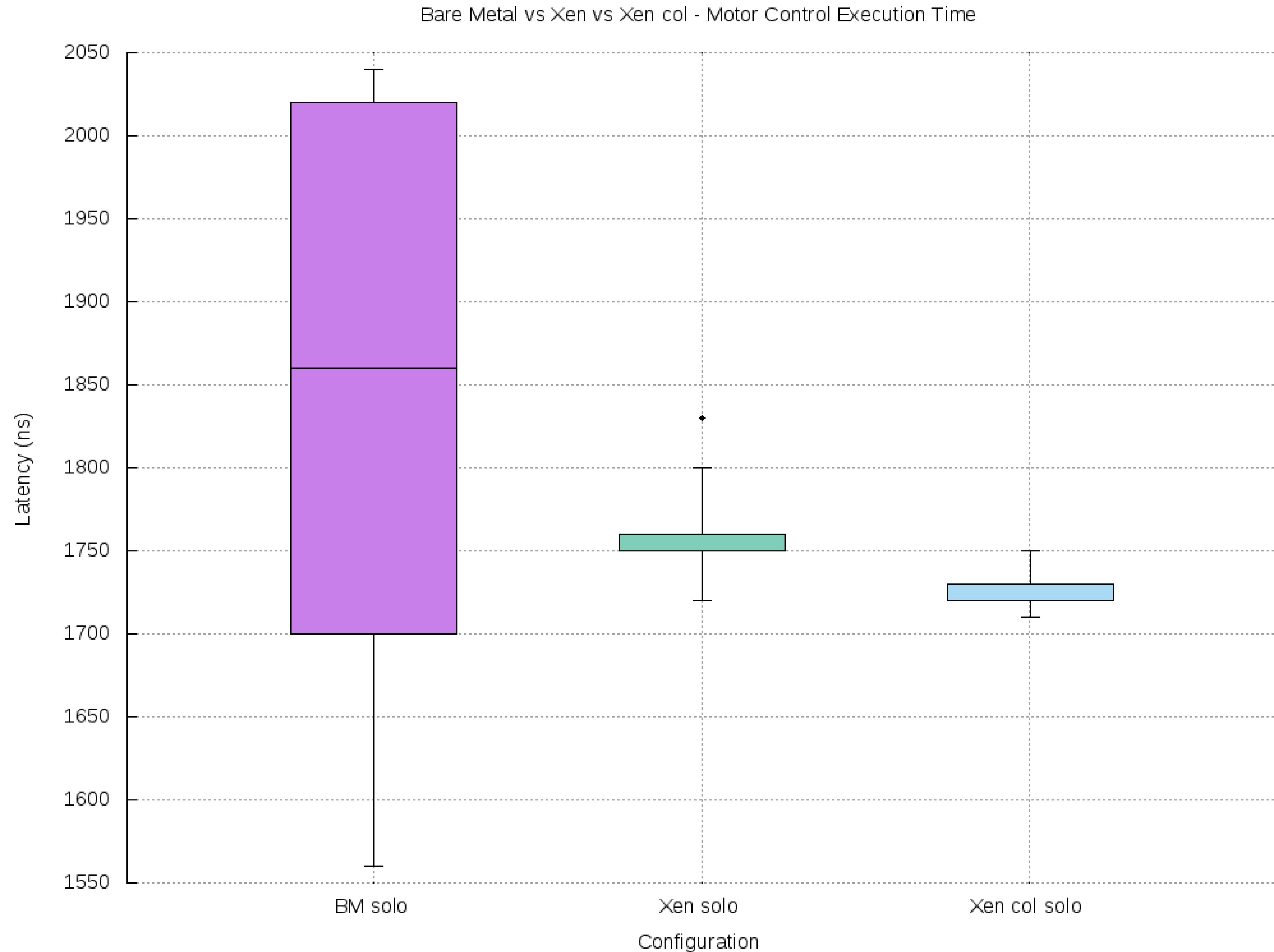
© Copyright 2020 Xilinx

Results

No interference:
results for reference

Lower is better

cpu0 for dom0
cpu1 for the benchmark
cpu2-3 sleep
No other VMs



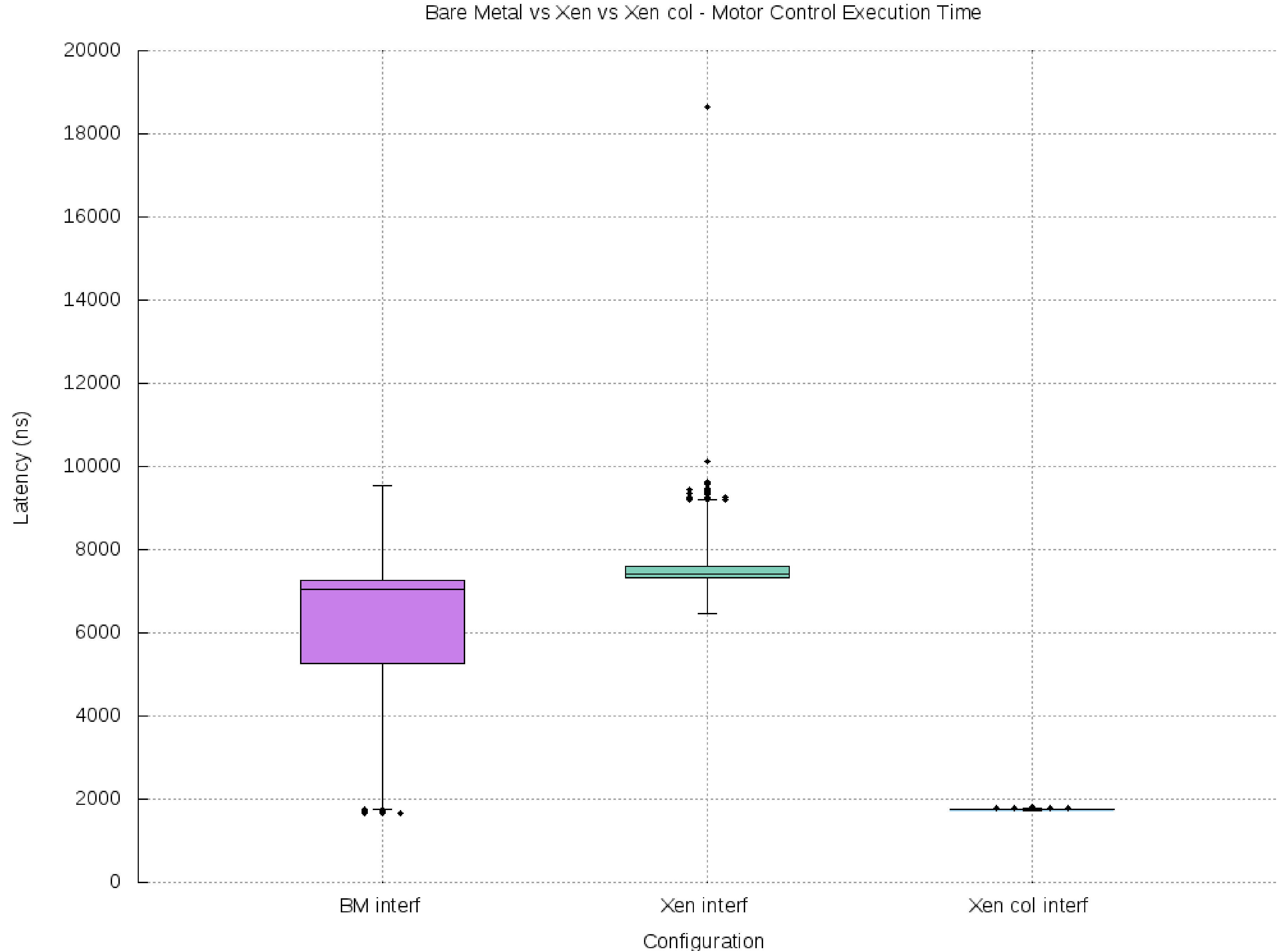
Results

Adding interference:

Xen Coloring
has the best execution time

cpu0 for dom0
cpu1 for the benchmark
cpu2 for interference
- memcpy loop 2MB

cpu3 sleep

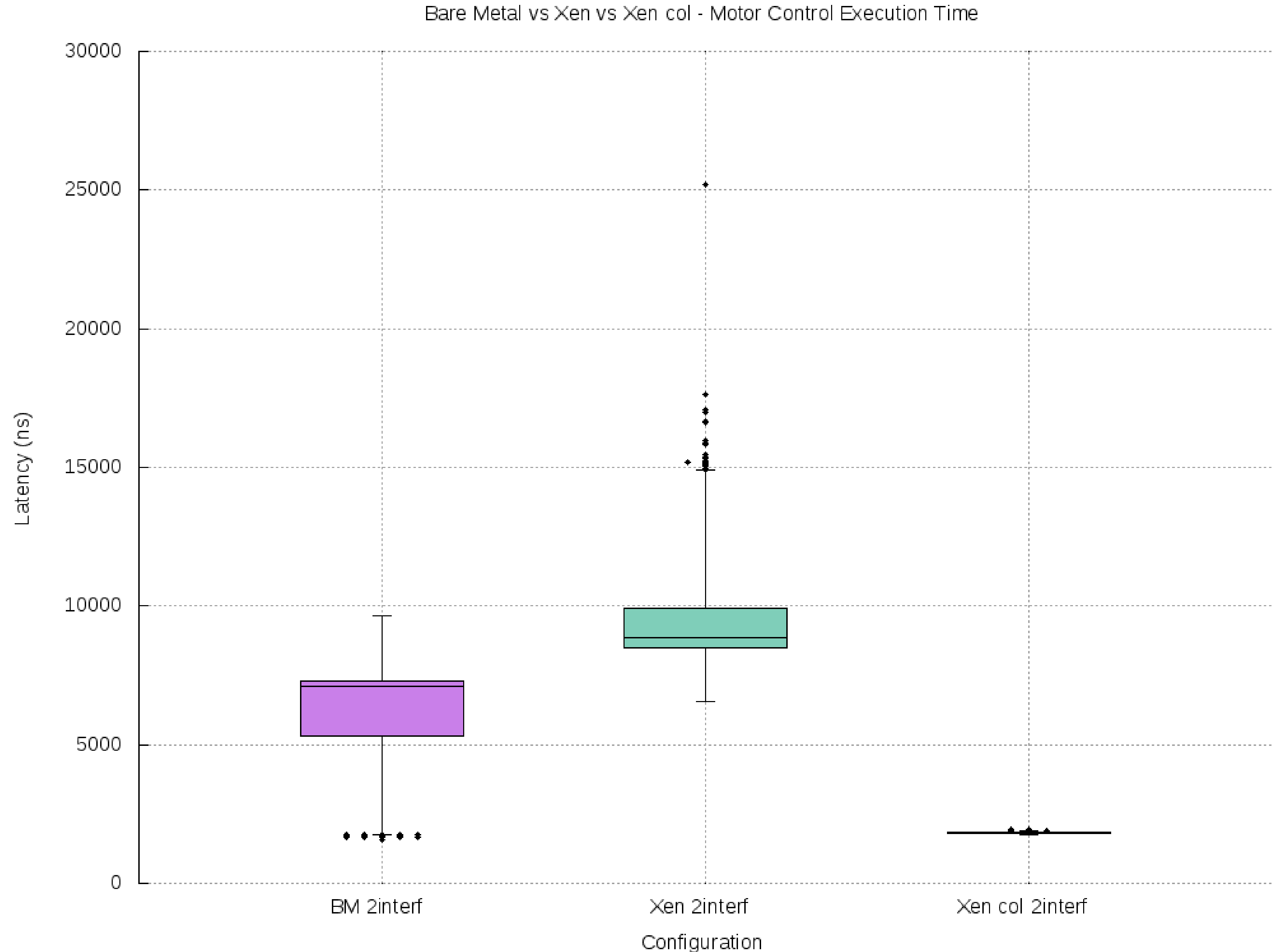


Results

Increasing interference:

- Xen Coloring is stable
- Other results worsen

cpu0 for dom0
cpu1 for the benchmark
cpu2-3 for interference
- memcpy loop 2MB



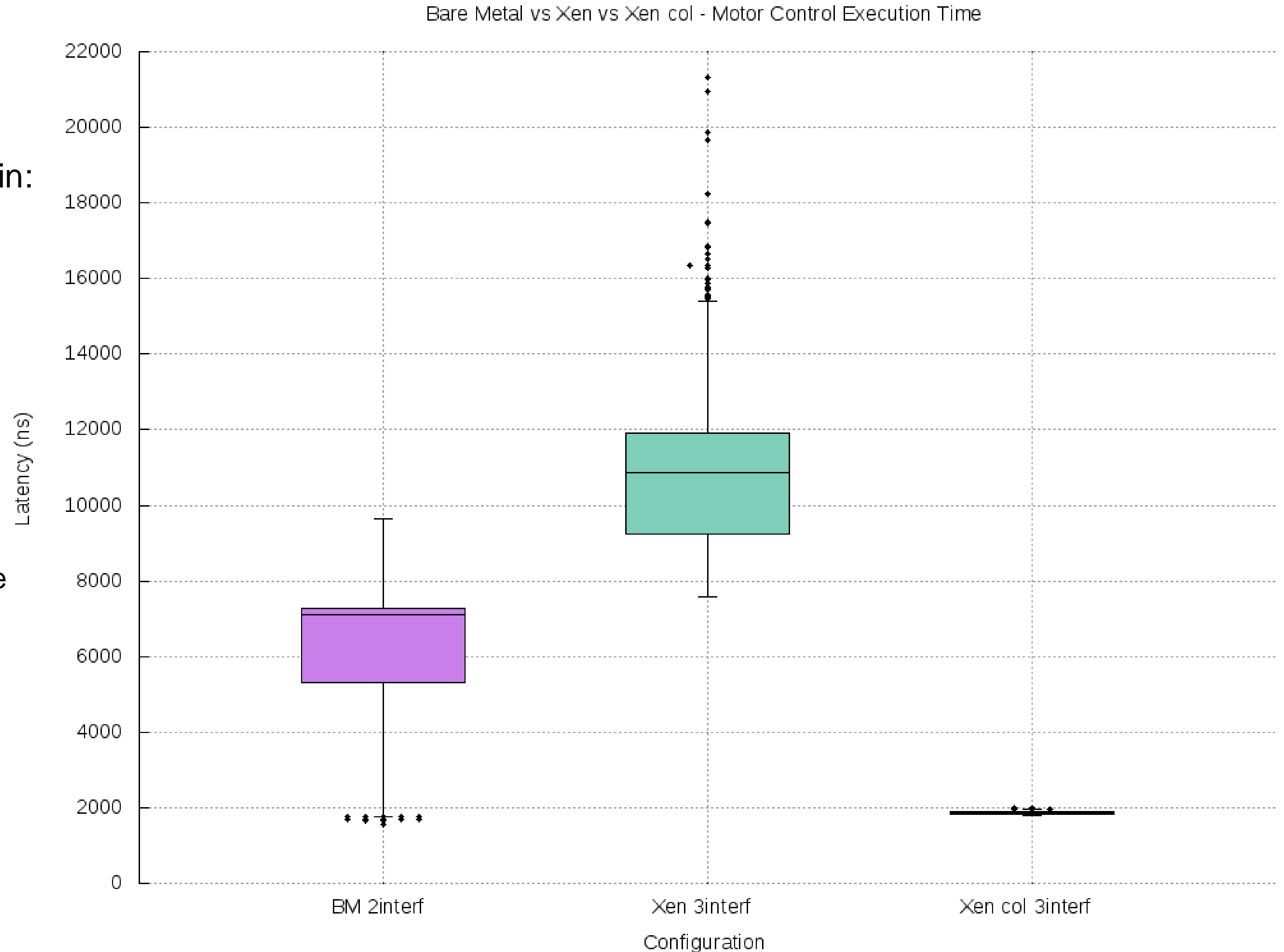
Results

Increasing interference again:

- Xen Coloring is stable
- Other results worsen still

cpu0 for dom0 with interference
  - Memcpy loop 2MB

cpu1 for the benchmark
cpu2-3 for interference
  - memcpy loop 2MB

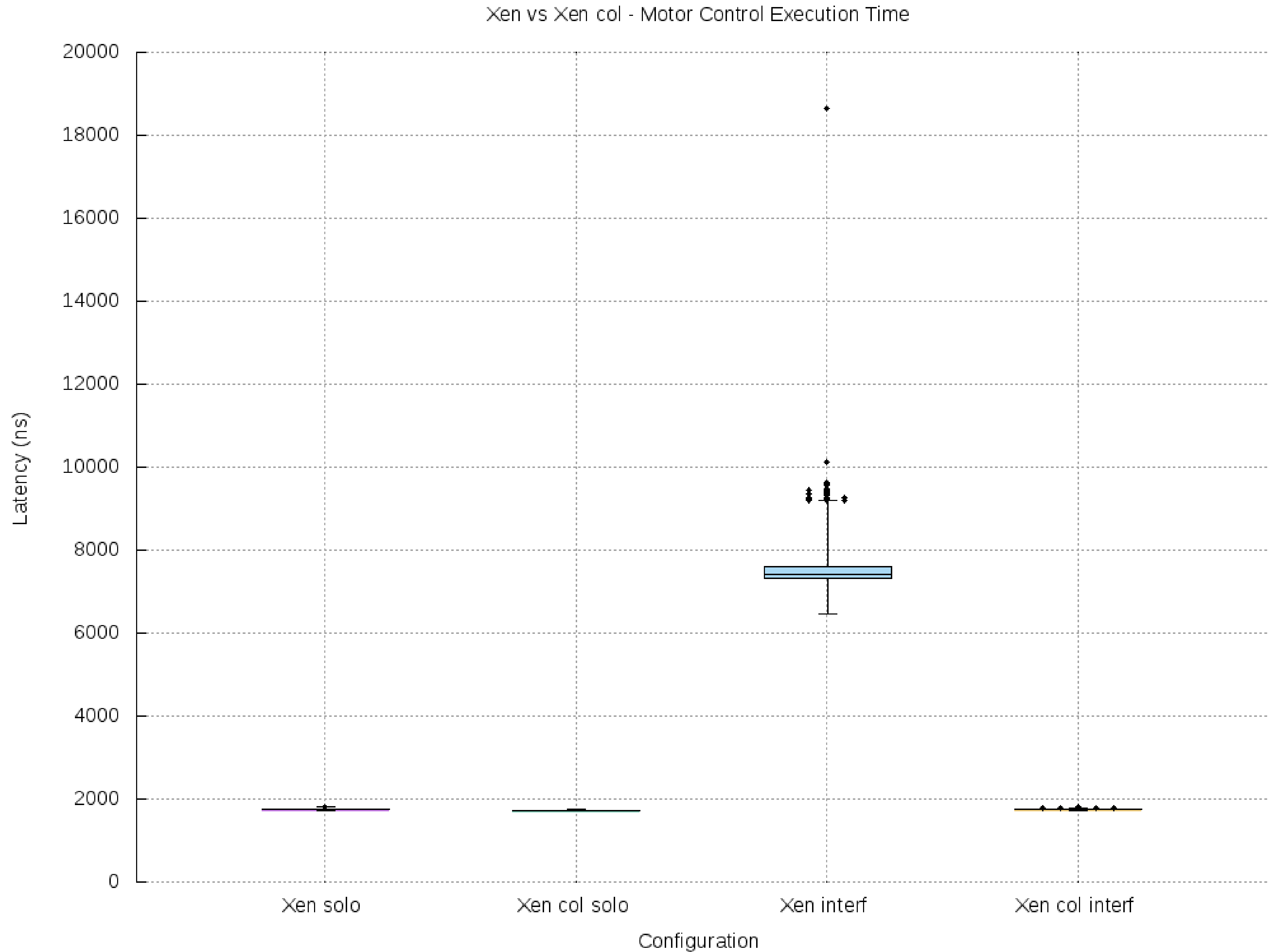


Results

Comparing
Xen vs. Xen Colored

Lower is better

All results on one slide



Benchmark #2: interrupt response time

- Measure the difference between the time when the interrupt service routine runs and the time the interrupt is set to fire, with and without interference
- Timer in Programmable Logic
  - to be independent from A53 cluster
  - to have a fully dedicated physical timer dedicated to the VM
    - arch_timer is partially virtualized by Xen

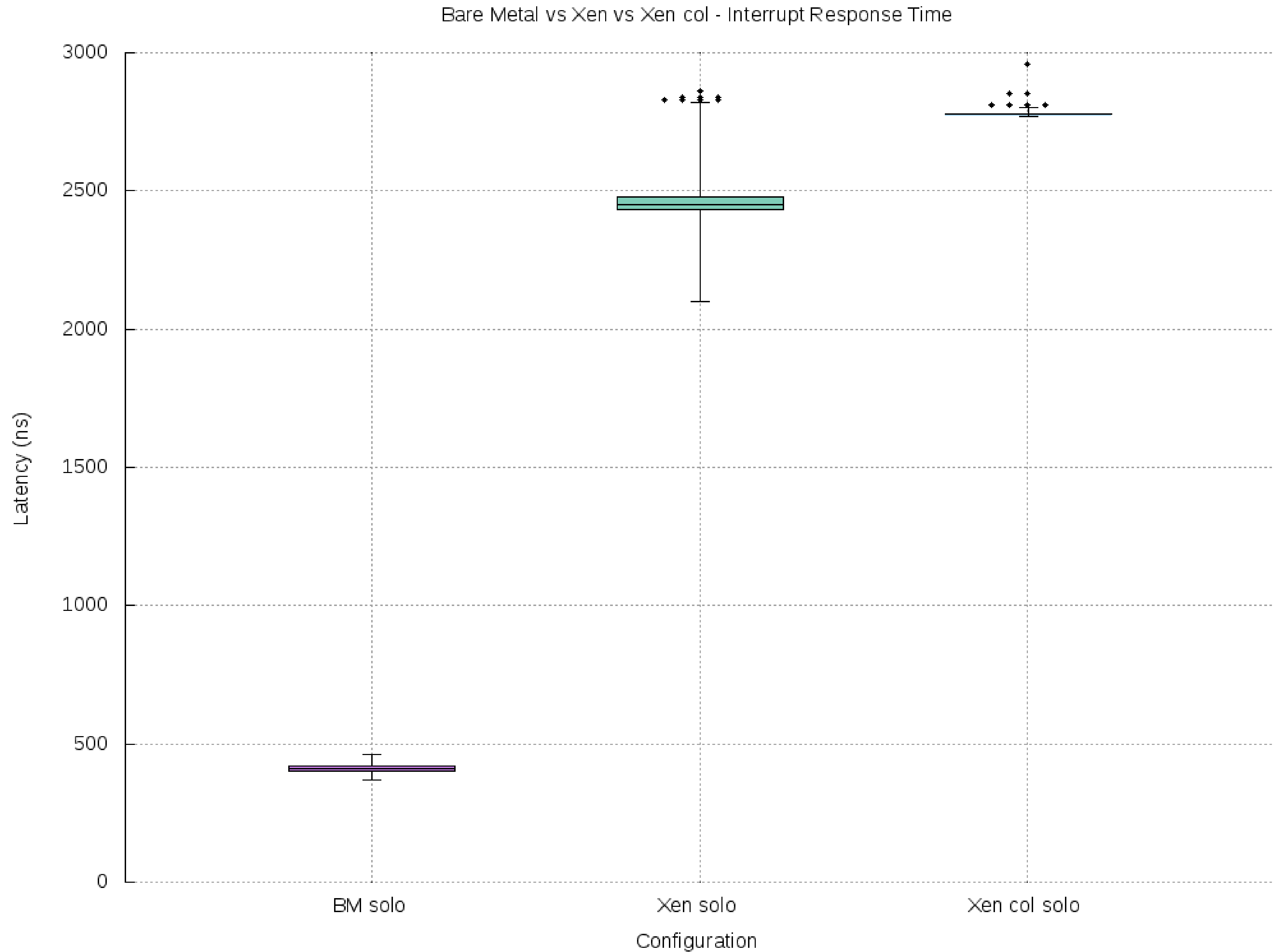
© Copyright 2020 Xilinx

Results

No interference:
results for reference

Lower is better

cpu0 for dom0
cpu1 for the benchmark
cpu2-3 sleep



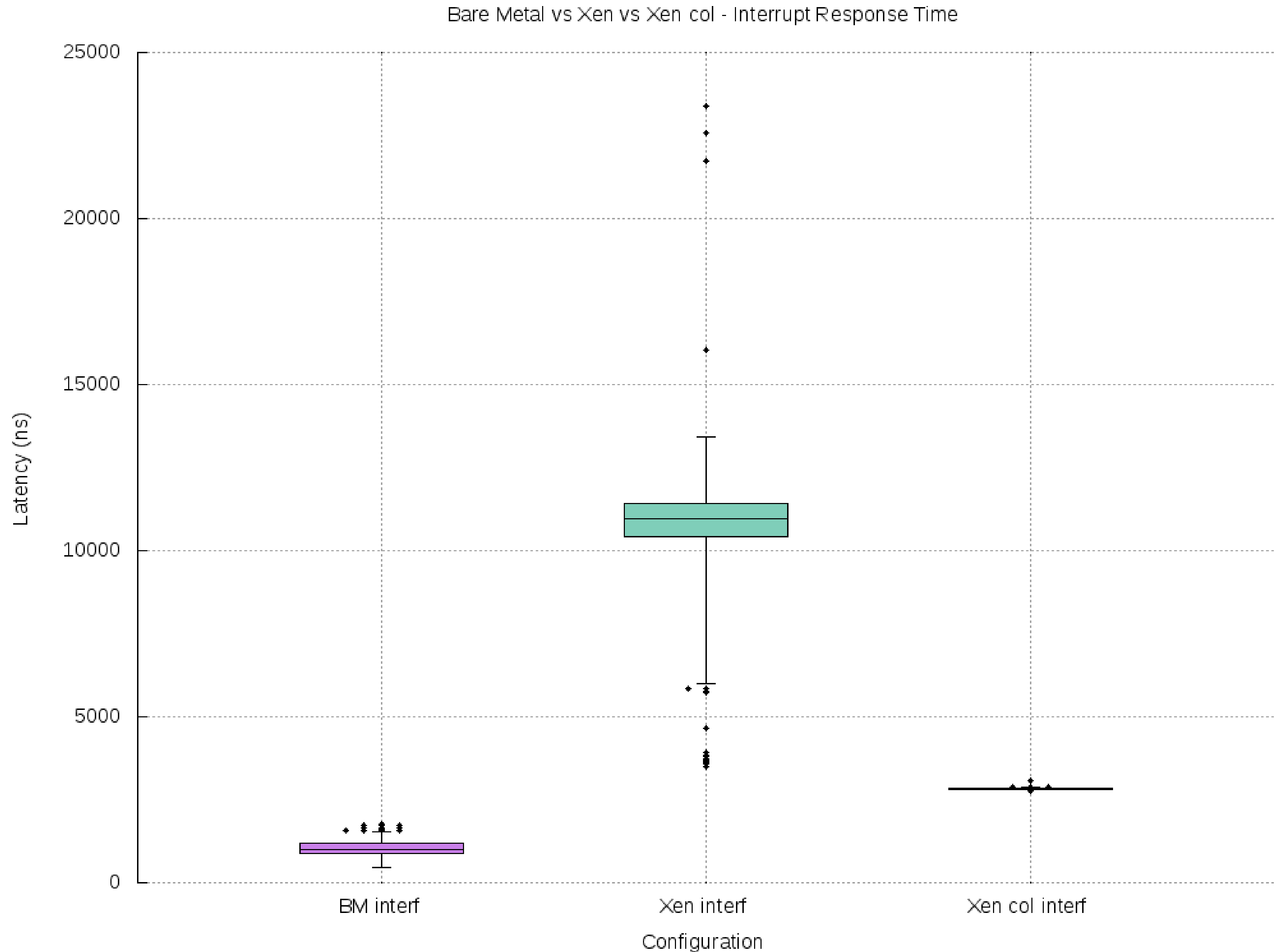
Results

Adding interference:

- Xen without Coloring worsen
- Xen Coloring is stable

cpu0 for dom0
cpu1 for the benchmark
cpu2 for interference
  - mempcy loop 2M

cpu3 sleeps



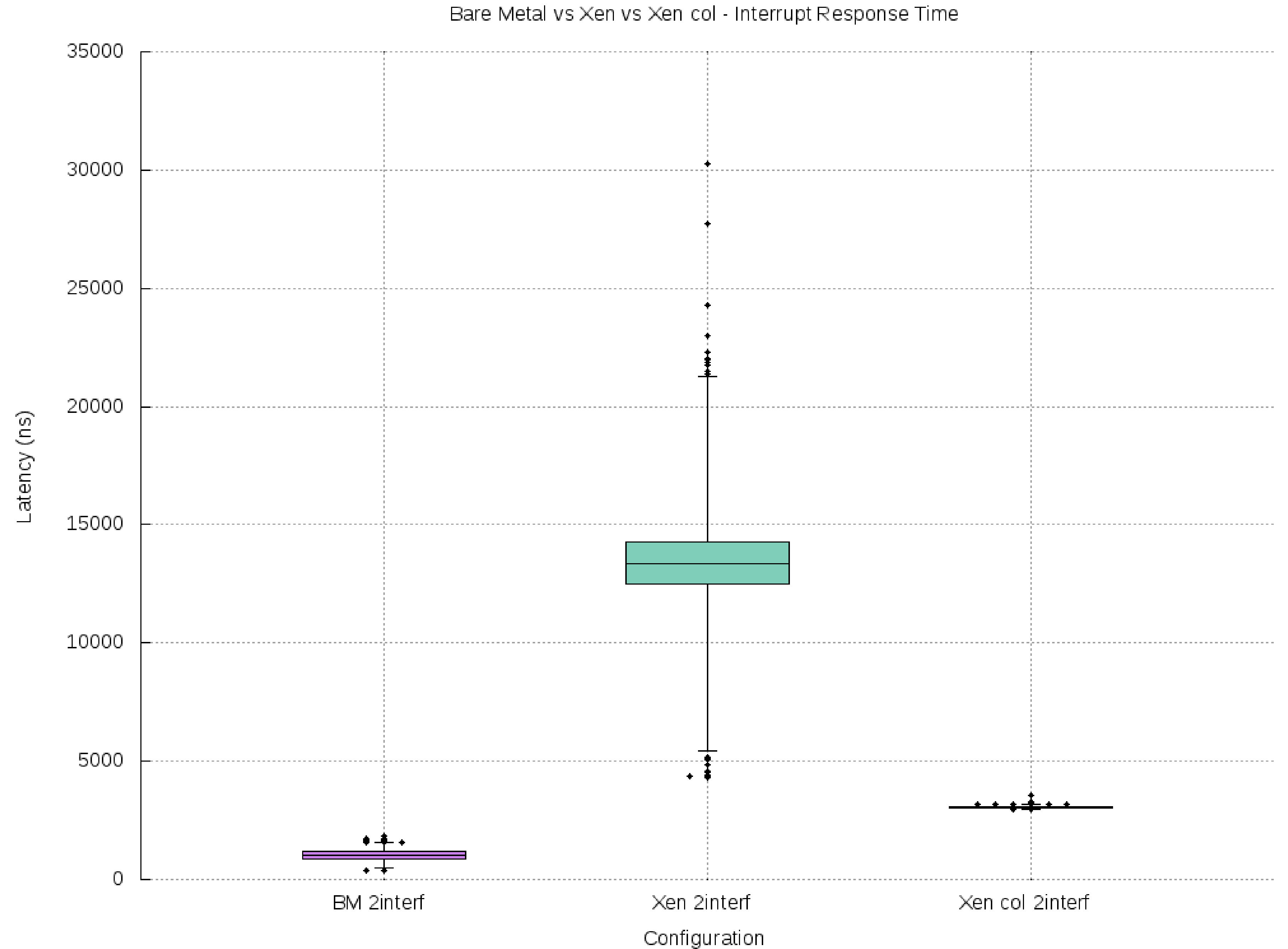
Results

Adding more interference:

- Xen without Coloring worsen again
- Xen Coloring is stable

cpu0 for dom0
cpu1 for the benchmark
cpu2-3 for interference

- memcpy loop 2M



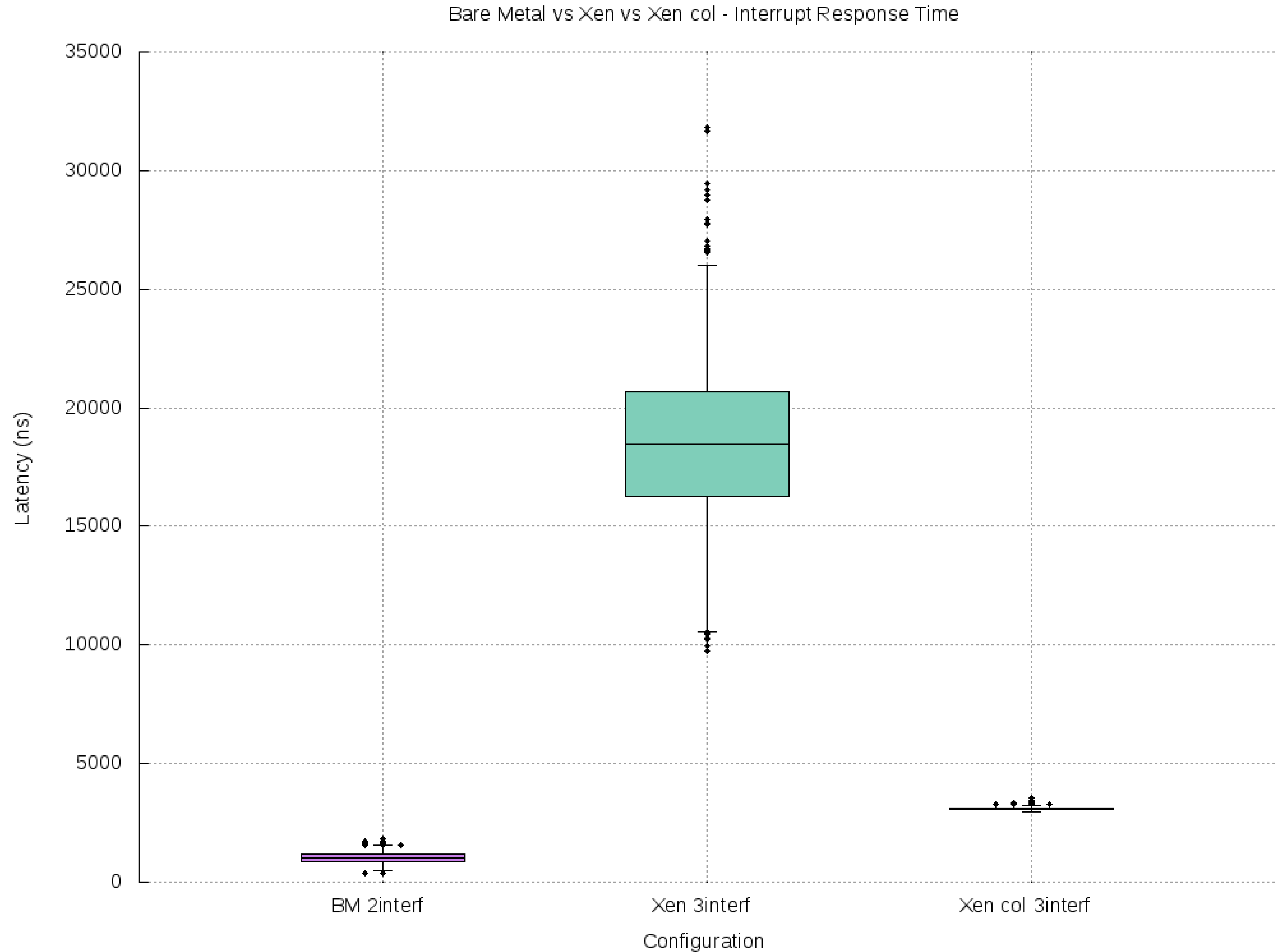
Results

Adding even more interference:

- Xen without Coloring worsen still
- Xen Coloring is stable

cpu0 for dom0 with interference
- memcpy loop 2M

cpu1 for the benchmark
cpu2-3 for interference
- memcpy loop 2M

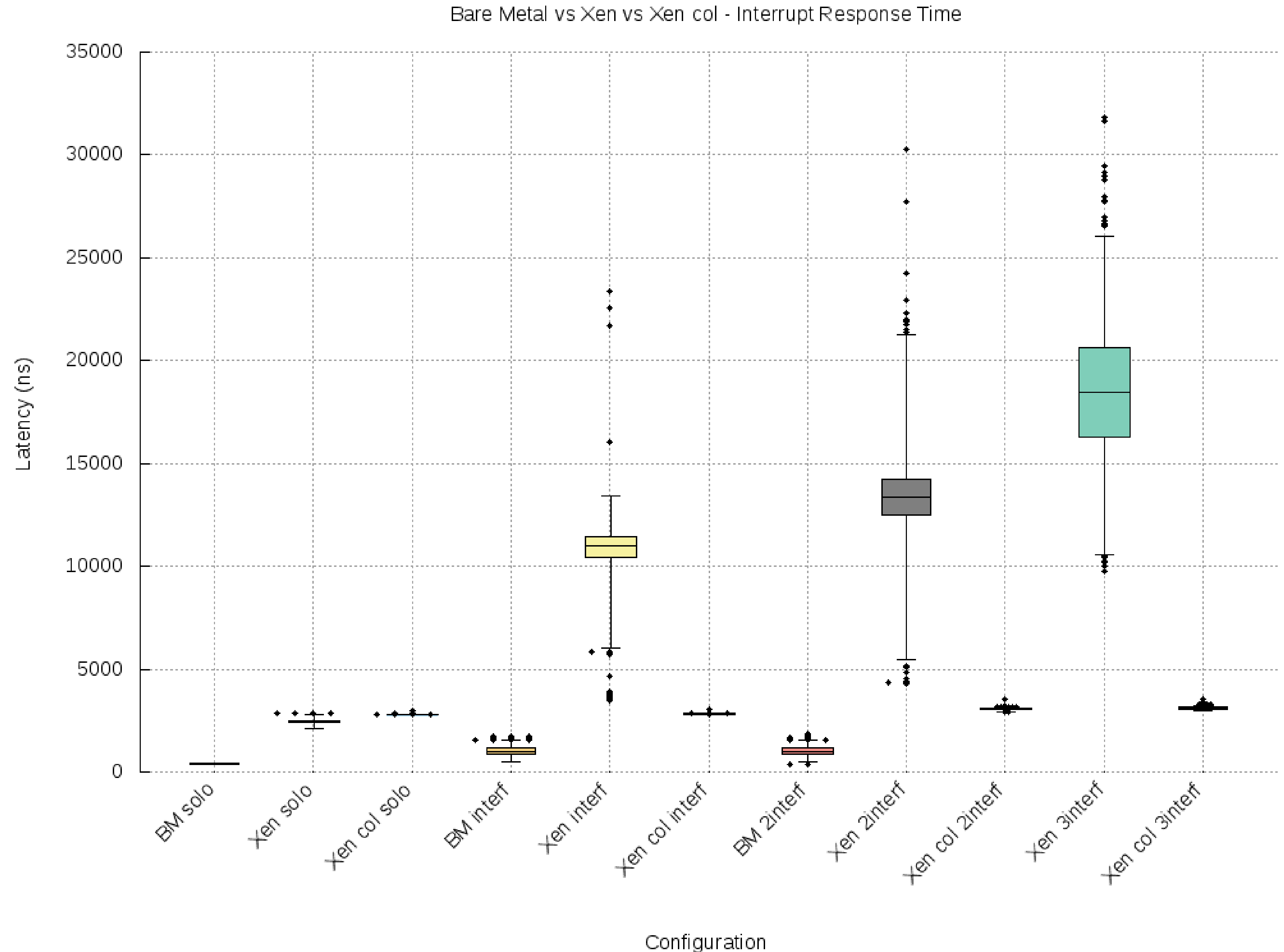


Results

All results together

Xen Coloring provides stable IRQ latency even under severe levels of interference

All results on one slide



Conclusions

- Xen Cache Coloring offers
  - Much lower execution times under stress
  - Much lower IRQ latency under stress
- Greatly improves Determinism -- measurements have far lower variance

- Xen Cache Coloring effectively reduces the effects of cache interference
- Enable deployments of real-time and non-real-time workloads on a single SoC

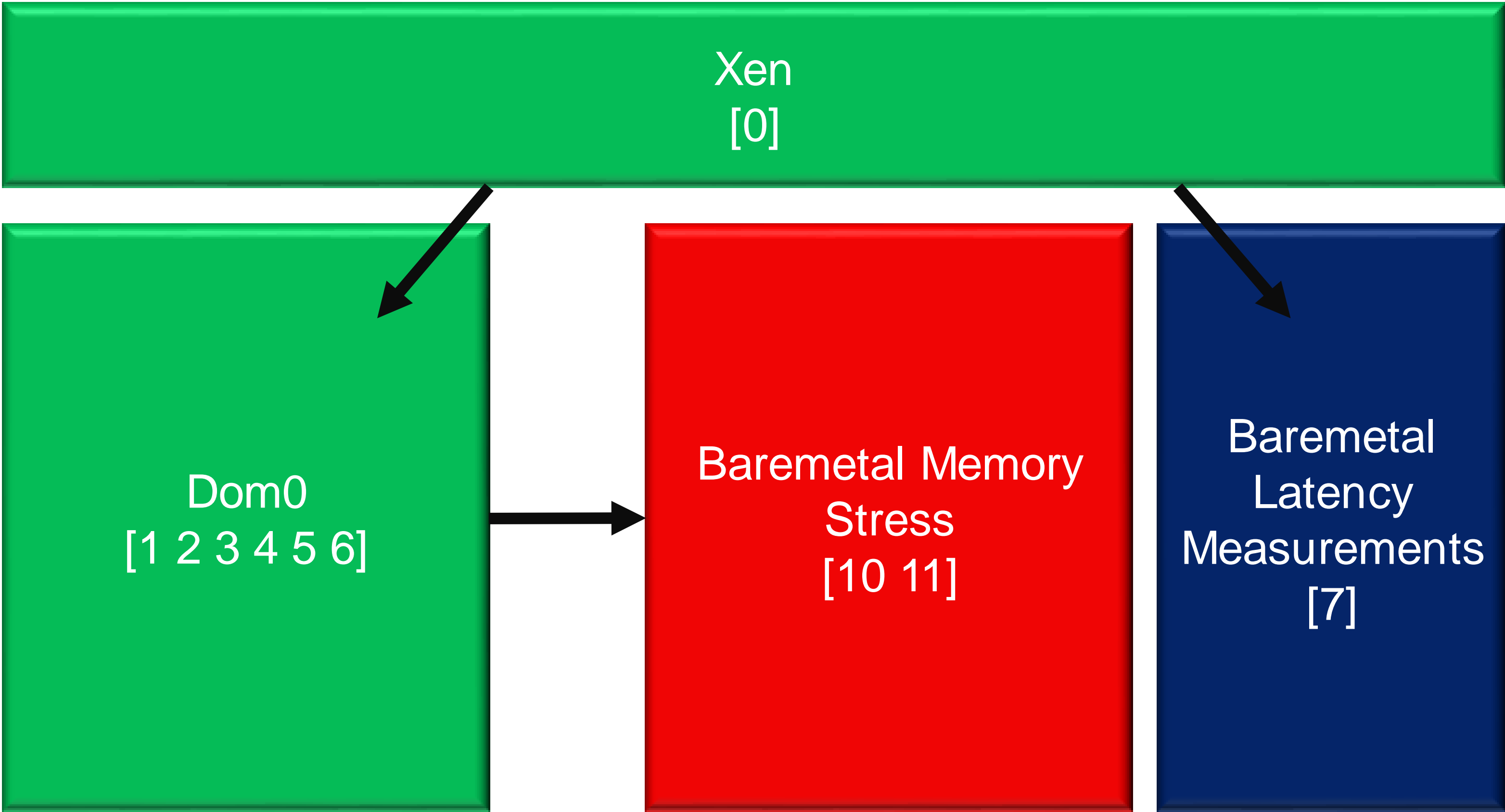
© Copyright 2020 Xilinx

Status

- All patches online at:
  - https://github.com/Xilinx/xen.git xilinx/release-2020.1
- TODO
  - Upstreaming
  - Linux Dom0 not 1:1 mapped, i.e. PV drivers support
    - Likely to be released in Petalinux 2021.1 – will work on any distro
- Limitations
  - Contiguous memory needs to be exposed as SRAM/MMIO region to VMs
  - SMMU is required if the VMs have any DMA-capable device assigned

© Copyright 2020 Xilinx

Xen Cache Coloring: Demo



© Copyright 2020 Xilinx

Baremetal Latency Measurements

sstabellini : vi — Konsole

File Edit View Bookmarks Settings Help

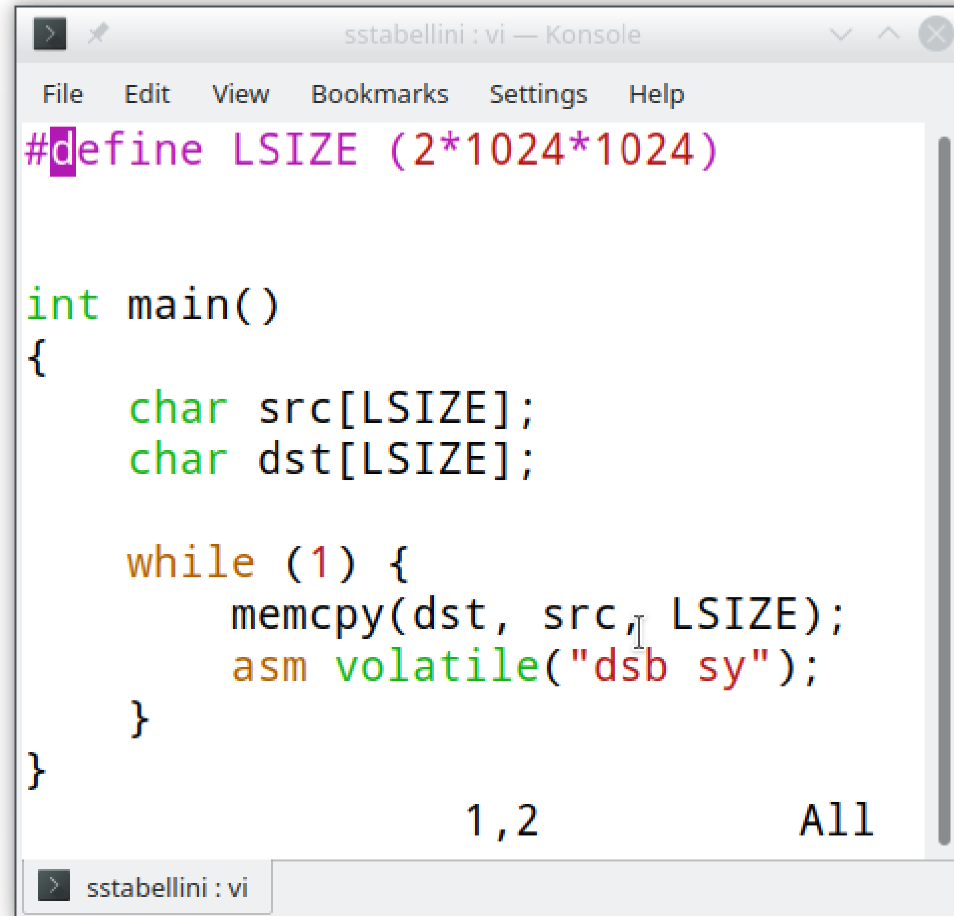
```
static void TickHandler(void *CallBackRef)
{
    u32 StatusEvent;

    XTime_GetTime(&real);
    Latencies[TickCount] = (real - expected)*(NS/COUNTS_PER_SECOND);
    /*
     * Read the interrupt status, then write it back to clear the interrupt.
     */
    StatusEvent = XTtcPs_GetInterruptStatus((XTtcPs *)CallBackRef);
    XTtcPs_ClearInterruptStatus((XTtcPs *)CallBackRef, StatusEvent);
    TickCount++;

    // Reset expected based on current time
    XTime_GetTime(&expected);
    expected += (INTERVAL_NS/COUNTS_PER_SECOND);
}
                                                    14,0-1        Bot
```

sstabellini : vi

© Copyright 2020 Xilinx

XILINX.

Baremetal Memory Stress



© Copyright 2020 Xilinx

XILINX.

Thank You

© Copyright 2020 Xilinx