

A MULTI-SOCKET FERRARI FOR NFV



ERICSSON



GOALS



GOING VIRTUALIZED
ENVIRONMENT-INDEPENDENCY
FLEXIBLE PLATFORM
BACKWARD COMPATIBILITY
HIGH AND DETERMINISTIC PERFORMANCE
HIGH AVAILABILITY
ROBUSTNESS
INTACT DPDK COMMON LIBRARIES

GOALS

GOING VIRTUALIZED



THAT'S LIFE :)

GOALS

ENVIRONMENT-INDEPENDENCY



RUNNING IN CLOUD
HYPERVISOR-AGNOSTIC SOLUTION
RUNNING ALSO NATIVE OR BARE METAL
REUSING EXISTING BLADES

GOALS

FLEXIBLE PLATFORM



FLEXIBLE YET SIMPLE CONFIGURATION
SUPPORT OF MULTIPLE APPLICATION MODELS
SCALABILITY
PERFORMANCE TUNING

(ADOPTING TO THE ENVIRONMENT WITHOUT
HARDCODING)

GOALS

BACKWARD COMPATIBILITY



EASY MIGRATION OF EXISTING APPLICATIONS

GOALS

HIGH AND DETERMINISTIC PERFORMANCE



HIGH THROUGHPUT

LOW PACKET DELAY VARIATION

SAME PERFORMANCE AFTER VM INSTANTIATIONS

EQUAL PER-INSTANCE PERFORMANCE

EQUALLY DISTRIBUTED RESOURCES

GOALS

HIGH AVAILABILITY



TELCO REQUIREMENT
FAST RECOVERY
AVOID SINGLE POINT OF FAILURE
REDUNDANCY



PROTECT AGAINST ACCIDENTAL MEMORY WRITES
MULTI-PROCESS SUPPORT
INCREASED DEBUGGABILITY

GOALS

INTACT DPDK COMMON LIBRARIES



EASY INTEGRATION OF NEW DPDK RELEASES

CHALLENGE



OUR TYPICAL APPLICATIONS HAVE LARGE MEMORY
FOOTPRINTS AND SIGNIFICANT AMOUNT OF RANDOM
READS/WRITES

SUPPORTING GIANT VIRTUAL MACHINES
NUMA-AWARENESS IN XEN HVM



IN REALITY, WE PRESENT
A VIRTUAL WORLD TO
DPDK

SOLUTION

OUR NEW DPDK EAL



SPLIT EAL INTO PUBLISHER AND CONSUMER
DECOUPLED ENVIRONMENT DETECTION
TOPOLOGY PUBLISHING
(CPU, MEMORY, DEVICE TOPOLOGY)
MEMORY CONFIGURATION PUBLISHING
DEVICE PUBLISHING
(REGISTERED DPDK PORTS)

SOLUTION

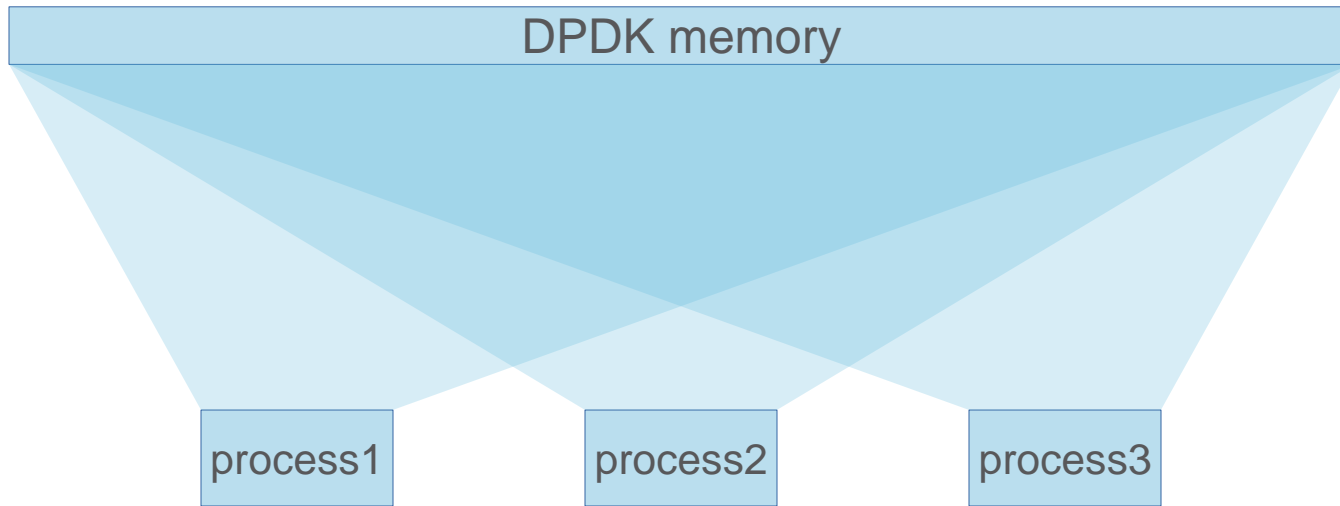
MULTI PROCESS SUPPORT



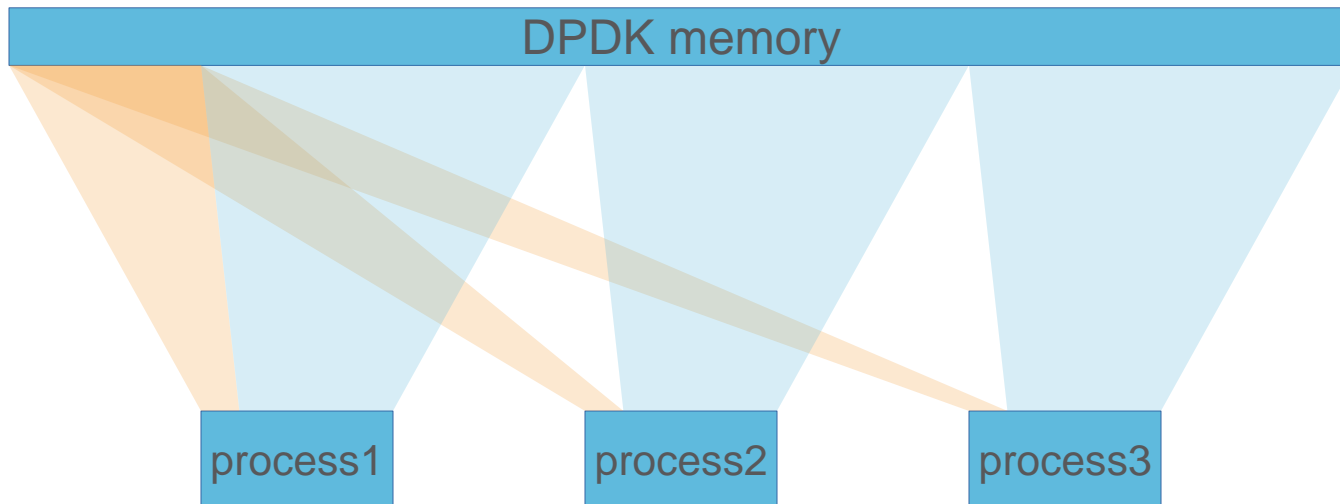
SELECTIVE MEMORY MAPPING
MEMORY ACCESS CONTROL
(AVOID MAPPING EVERYTHING AS R/W)



DPDK mapping



Our mapping



R/W

R/O

SOLUTION

MULTI PROCESS SUPPORT CONT'D



VIRTUAL ADDRESS SPACE ALLOCATOR
PRIMARY ROLE PASSING
(DISTRIBUTED RESOURCE MANAGEMENT)
DEVICE (RE-)INITIALIZATION

SOLUTION

IMPROVED MEMORY MANAGEMENT



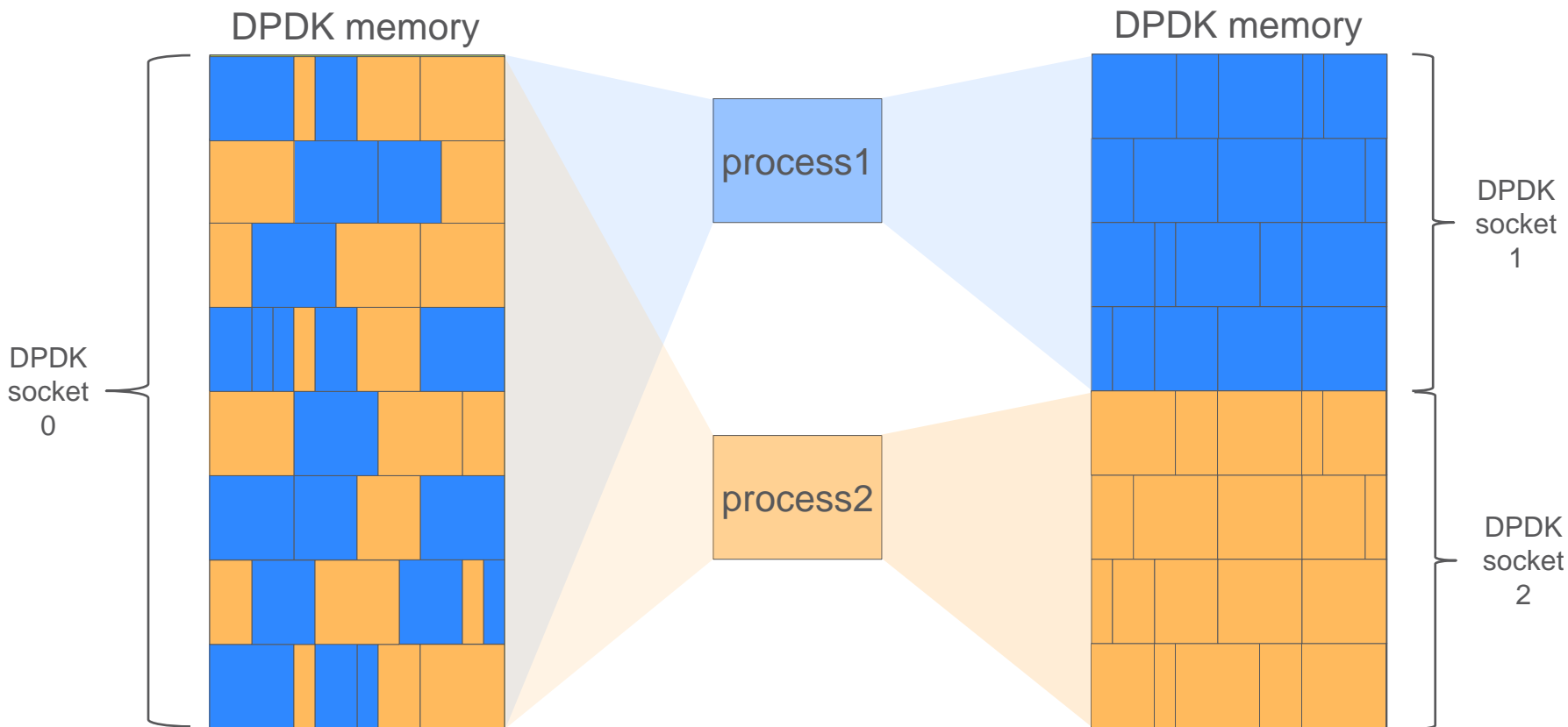
SIMPLE MEMORY MANAGEMENT API FOR APPS
(NAMED MEMORY PARTITION POOLS)



FLEXIBLE CONFIGURATION / RE-CONFIGURATION
(WITHOUT CODE CHANGE)

MAKING DPDK NUMA AWARE IN A NUMA FLAT OS
MORE GRANULAR WAY OF PLACING OBJECTS
(CONTROL TLB ENTRY USAGE => ZERO TLB MISS)

SOLUTION

IMPROVED MEMORY MANAGEMENT CONT'D



-  memzones allocated/used by process1
-  memzones allocated/used by process2

SOLUTION

IMPROVED MEMORY MANAGEMENT CONT'D



PARTITIONING MEMORY
(MULTIPLE DPDK NUMA SOCKETS)

FRAGMENTATION SUPPORT

PER-LCORE PRIVATE MEMORY
(DEDICATED DPDK SOCKET ID)

WHILE LEAVING DPDK COMMON LIBRARIES INTACT

SOLUTION

RESOURCE MANAGER



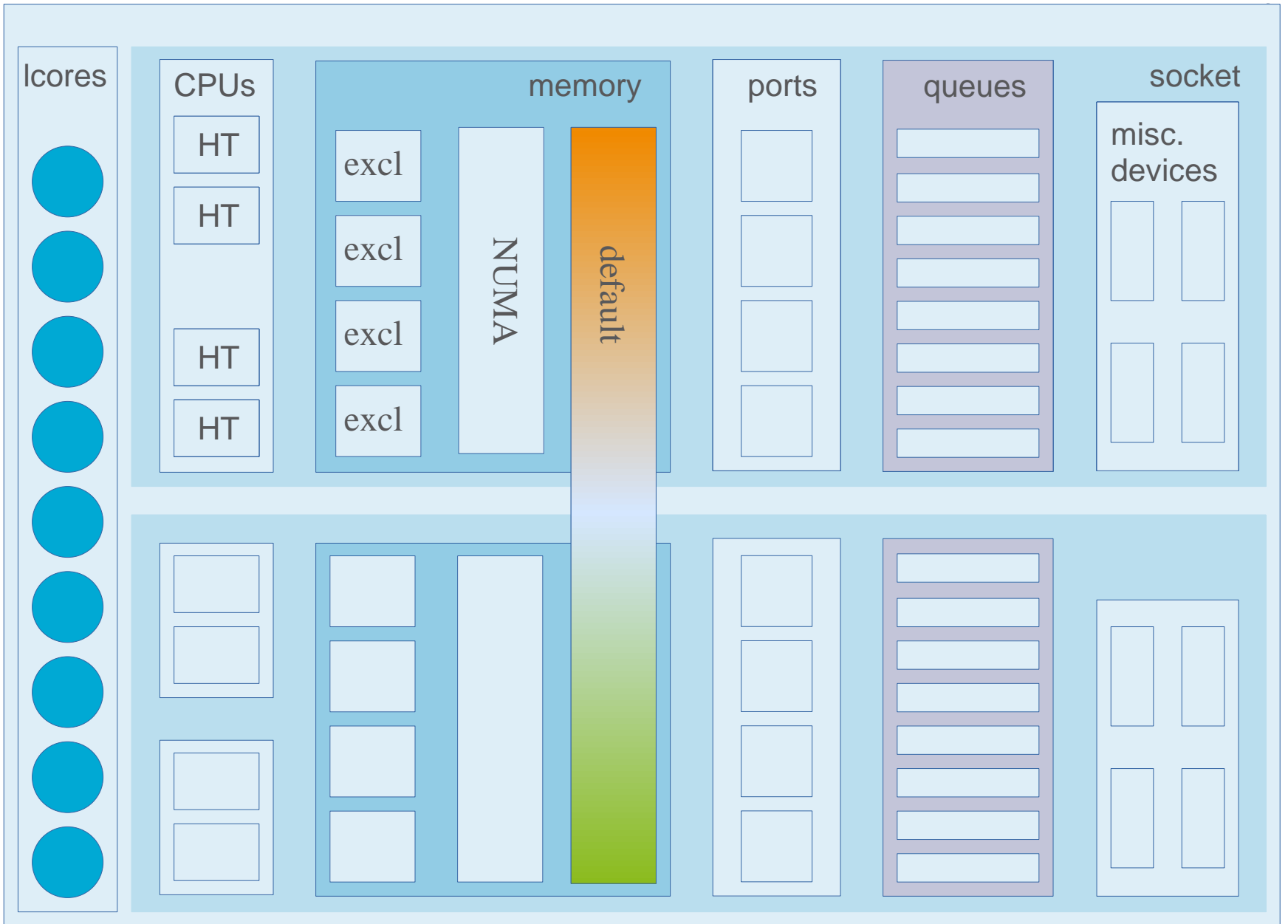
KEEP TRACK OF AVAILABLE RESOURCES:
LCORE, CPU, MEMORY, PORT, QUEUE, DEVICE
ON-DEMAND RESOURCES
RECLAIM RESOURCES

SOLUTION

READY TO USE CONFIGURATIONS

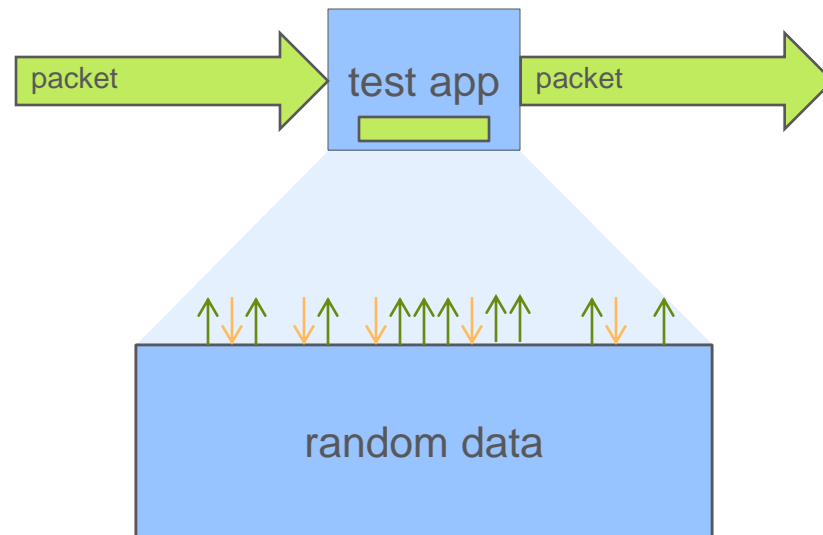


PUBLISHING CONFIGURATION
MEMZONES (MEMDOMAINS FOR APP)
NAMED PACKET POOLS
NAMED VIRTUAL PORTS / QUEUES



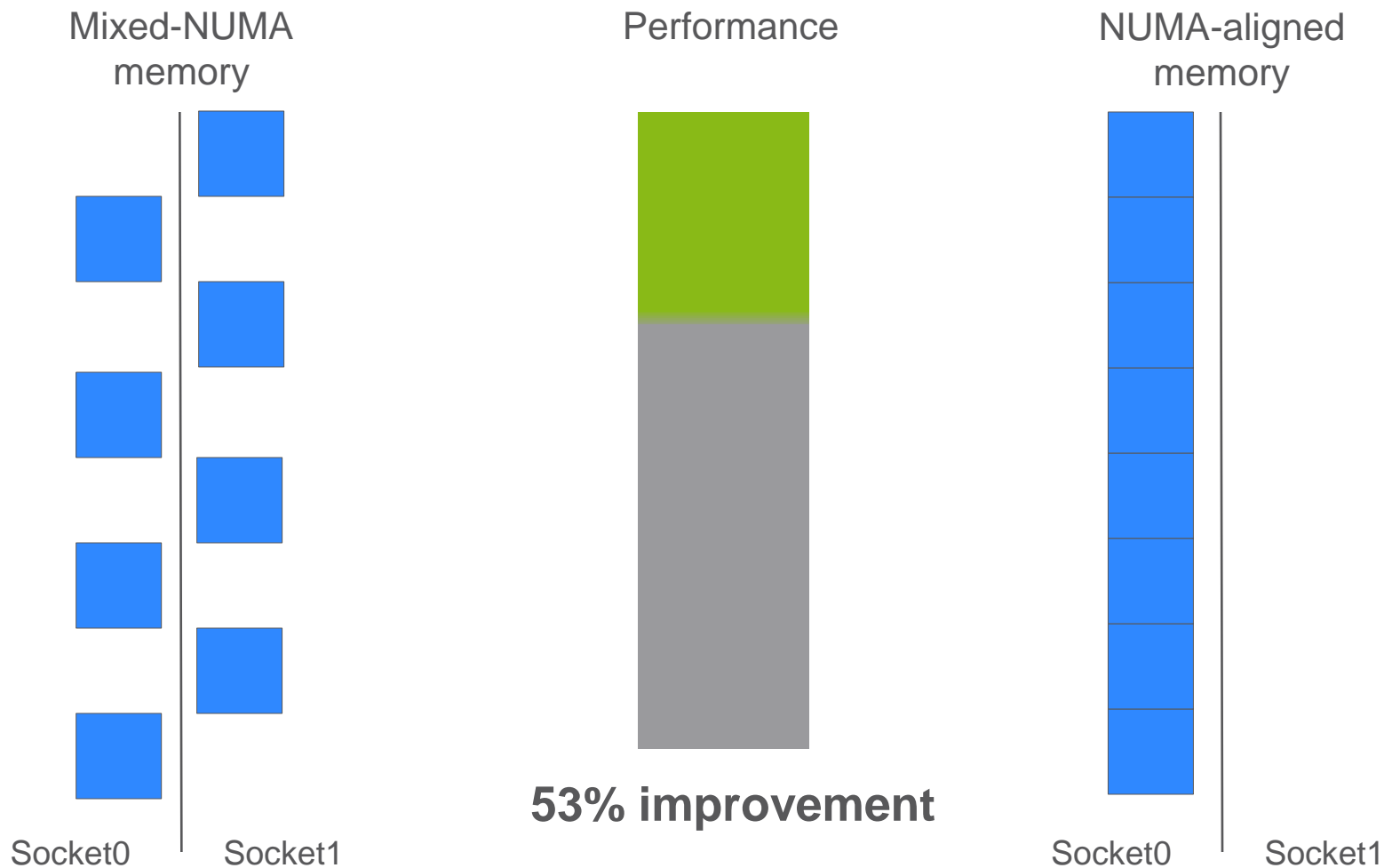
RESULTS

(SAMPLE TEST APP BRIEF)



RESULTS

(DUAL SOCKET - NUMA ALIGNMENT)

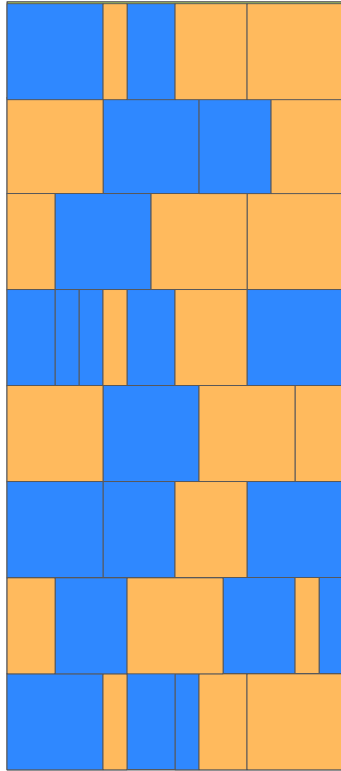


RESULTS

(SINGLE SOCKET - ZERO TLB MISS)



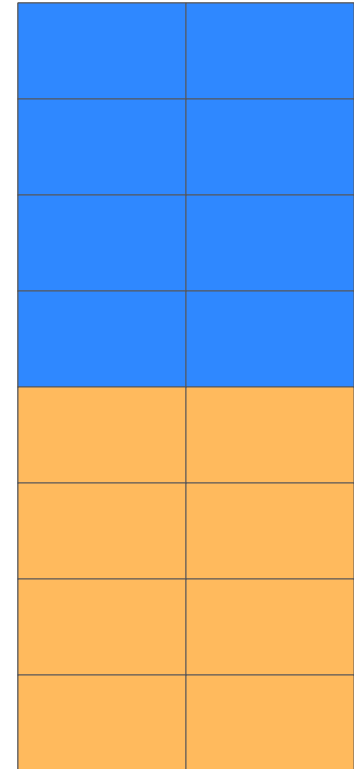
Fragmented allocation



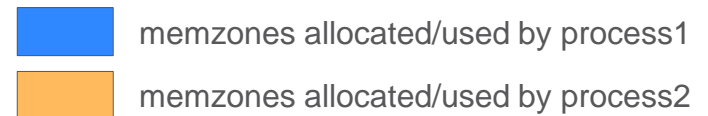
Performance



Unfragmented allocation

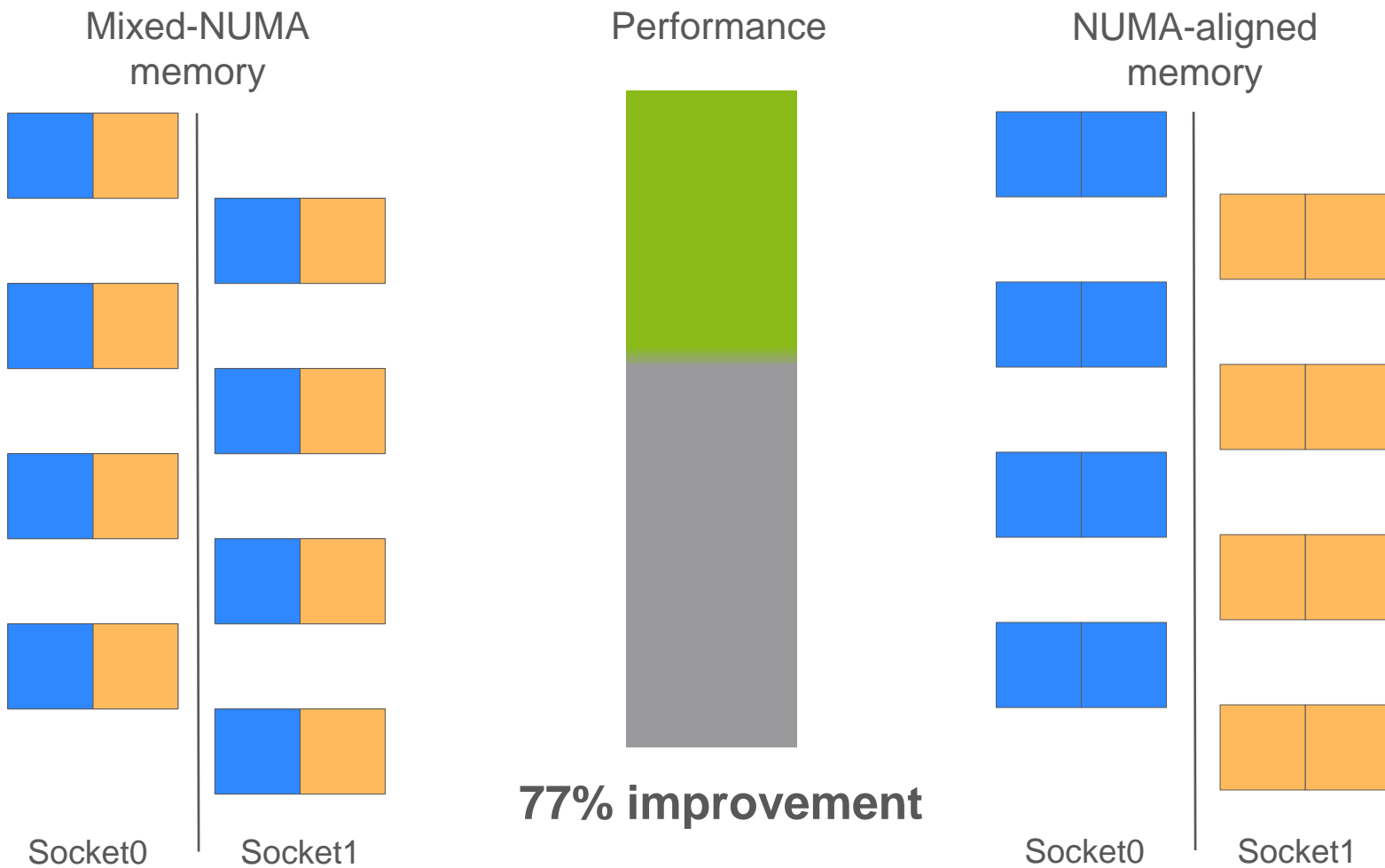


16% improvement



RESULTS

(NUMA ALIGNMENT PLUS ZERO TLB MISS)



CONCLUSION



BE(A)WARE OF THE NUMA
HUGEPAGE BACKING (HYPERVISOR CONTROL)
TLB MISSES ARE COSTLY IN VM
PROPER MEMORY ALIGNMENT IS VITAL

IDEAS



IMPROVE MEMZONE ALLOCATOR
FURTHER IMPROVE MULTI-PROCESS SUPPORT
PER-LCORE RTE_MALLOC AREA
DPDK DOMAINS
GENERIC RESOURCE MANAGER FOR DPDK
ADDITIONAL CONFIGURATION LAYER TO EAL
ADD CACHE-QOS TO MEMDOMAINS



ERICSSON

BACKUP SLIDES



CONFIGURATION EXAMPLE

CPUALIAS



```
cpualias all {
    cpumask = "0-31"    # all available CPU
}
cpualias foreground {
    cpumask = "2-31:2"  # even cpus excluding cpu0
}
```

CONFIGURATION EXAMPLE

NUMA TYPE



```
# -----  
# Per NUMA node shared memory.  
# App instances tied to the same NUMA  
# are sharing the same memory partition.  
# -----  
memdomain App_NUMA_Shared {  
    type = numa  
    cpualias = "all"  
    alloc_memzone = true  
    size {  
        is_per_numa = true  
        huge_2M = 0  
        huge_1G = 1G  
    }  
}
```


CONFIGURATION EXAMPLE

EXCLUSIVE TYPE



```
# -----  
# Per App instance private memory.  
# Every instance has its own memory  
# partition.  
# -----  
memdomain App_Thread_Local {  
    type = excl  
    cpualias = "foreground"  
    alloc_memzone = true  
    size {  
        is_per_cpu = true  
        huge_2M = 0  
        huge_1G = 1G  
    }  
}
```