ERICSSON

GENERIC RESOURCE MANAGER

ANDRÁS KOVÁCS (ANDRAS.KOVACS@ERICSSON.COM)

LÁSZLÓ VADKERTI (LASZLO.VADKERTI@ERICSSON.COM)

A manager we would like :)

LAST YEAR...

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

WHAT IS A RESOURCE?

by *merriam-webster*:

- something that a country has and can use to increase its wealth
- a supply of something (such as money) that someone has and can use when it is needed
- a place or thing that provides something useful

by *wikipedia (computing)*:

- A resource, or system resource, is **any physical or virtual component of limited availability** within a computer system. [...] Every internal system component is a resource. […].

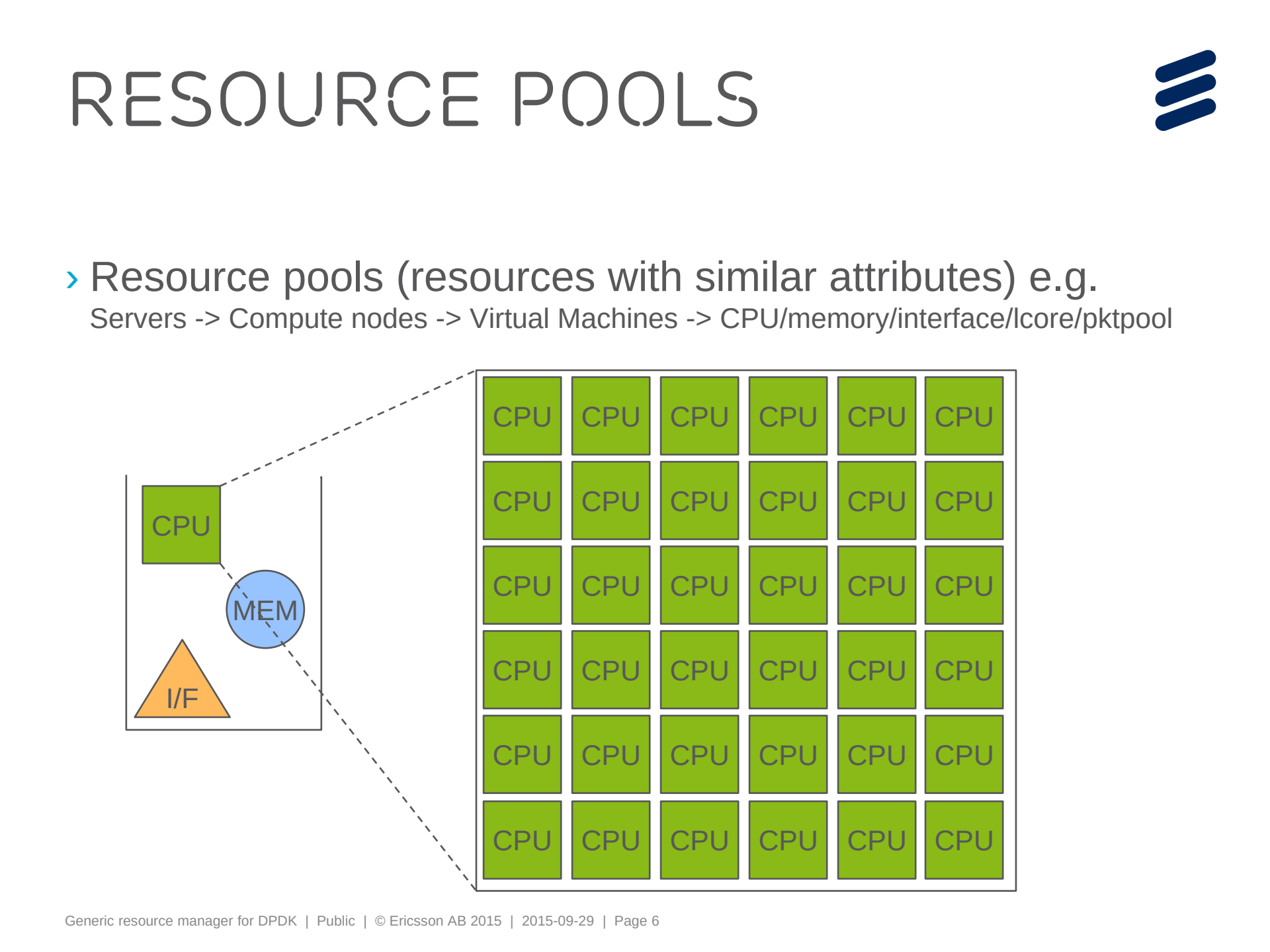
TYPICAL DPDK RESOURCES

- CPU
- Memory (hugepage, cache, ivshm, memzone)
- Virtual address
- Packet pool
- Network/Virtual Device (port/queue)
- Lcore
- Instance
- HW accelerator
- And there are lots of other resources coming… (threads, protocol stacks, etc)

WHY?

- Better control over resources
- Hide “real” environment from applications
- Support migration
- Keep track of resources/usage (High Availability)
- Prioritize resources
- Access control
- Avoid code duplication

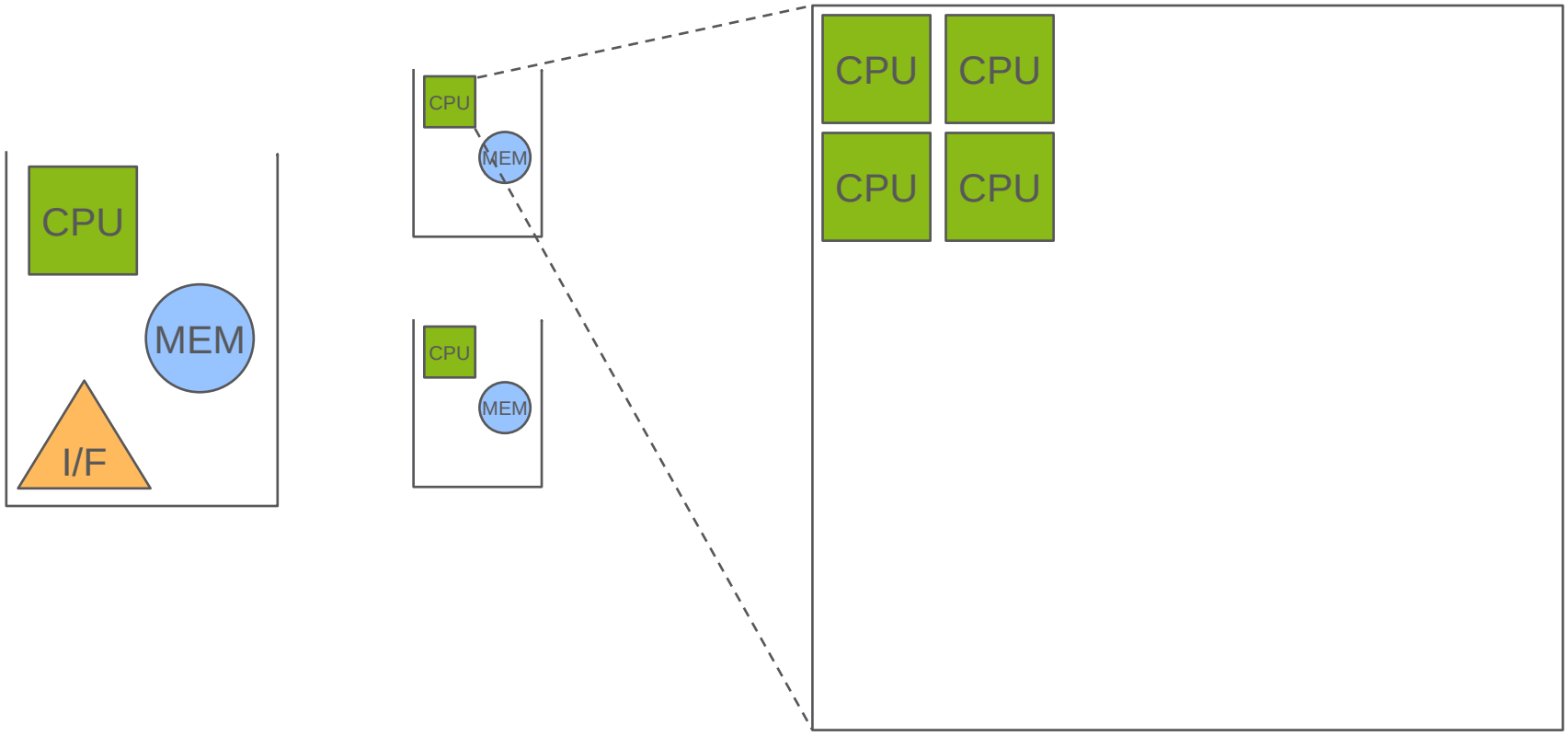
RESOURCE POOLS



- Resource pools (resources with similar attributes) e.g.
  Servers -> Compute nodes -> Virtual Machines -> CPU/memory/interface/lcore/pktpool

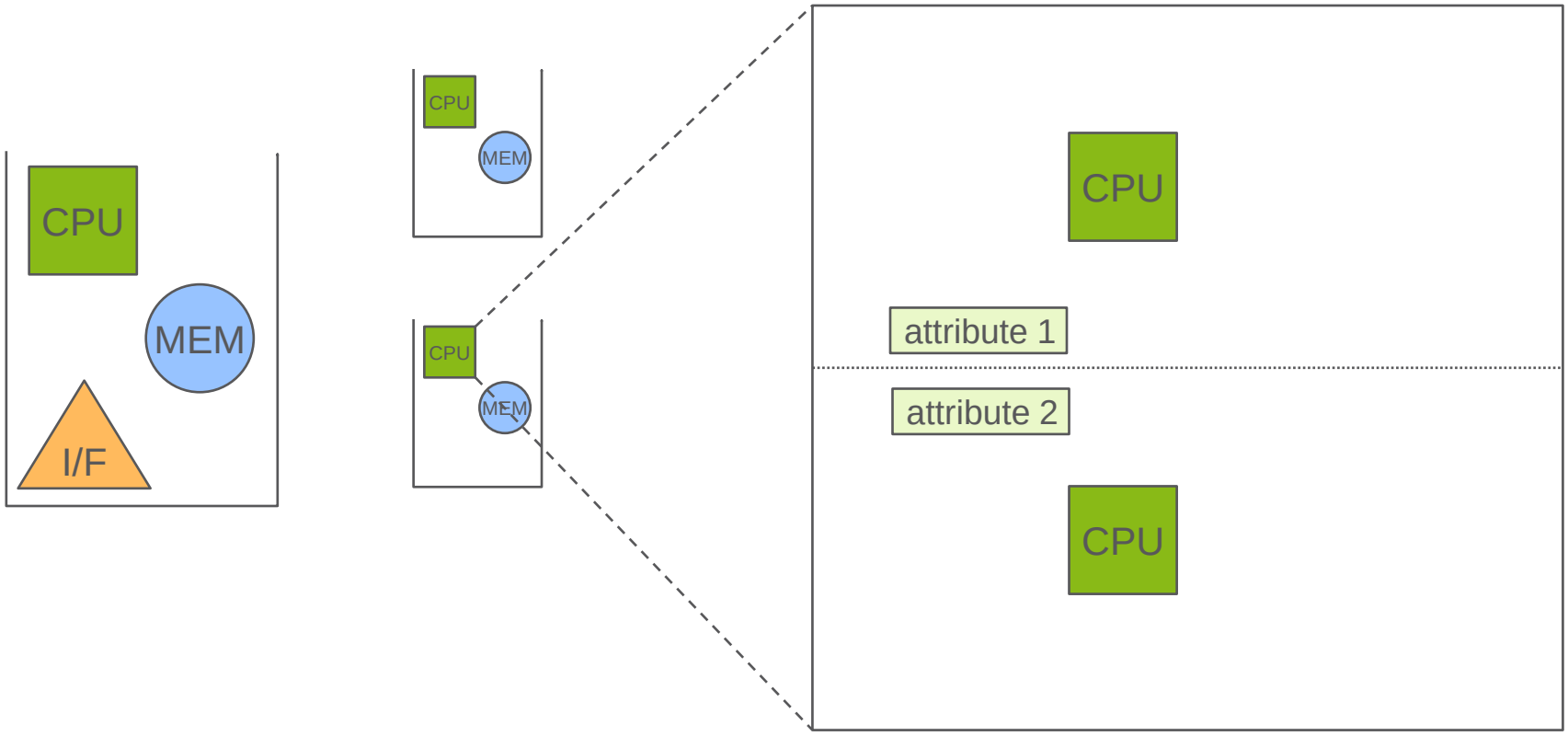
RESOURCE POOLS

- Resource pools (resources with similar attributes) e.g.
  Servers -> Compute nodes -> Virtual Machines -> CPU/memory/interface/lcore/pktpool



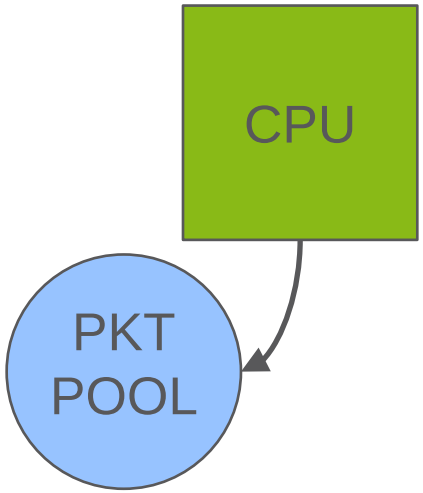
RESOURCE POOLS

- Resource pools (resources with similar attributes) e.g.
  Servers -> Compute nodes -> Virtual Machines -> CPU/memory/interface/lcore/pktpool

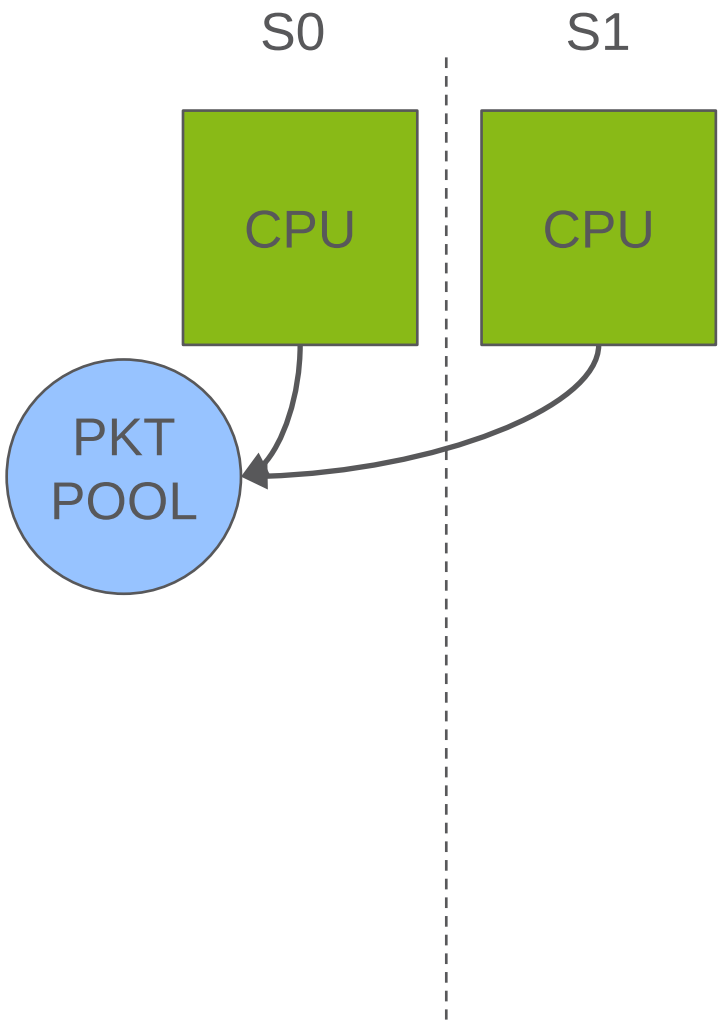


CONFIG EXAMPLE

```
cpualias foreground {
      cpumask = "2"
}


pktpool pktpool_fg {
      cpualias = foreground
      num_pkts = 8K
      num_cached_pkts = 64
}
```

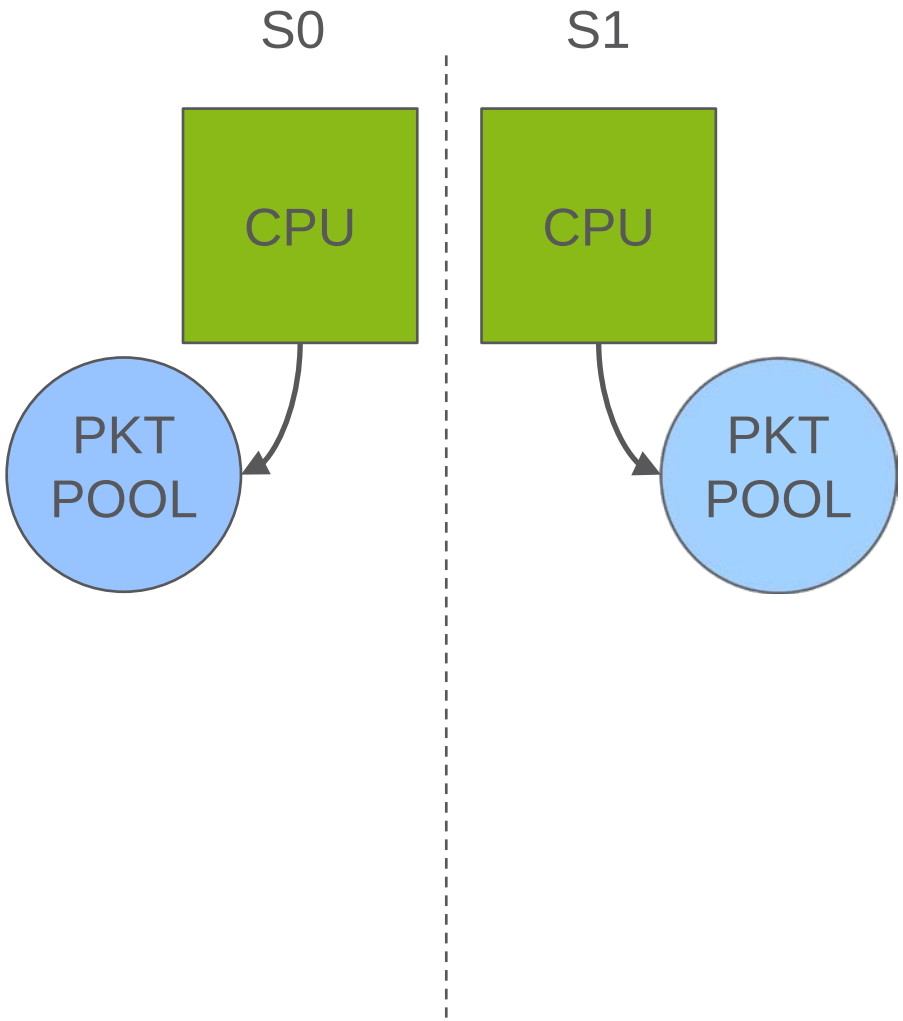


CONFIG EXAMPLE




```
cpualias foreground {
      cpumask = "2,12"
}


pktpool pktpool_fg {
      cpualias = foreground
      num_pkts = 8K
      num_cached_pkts = 64
}
```

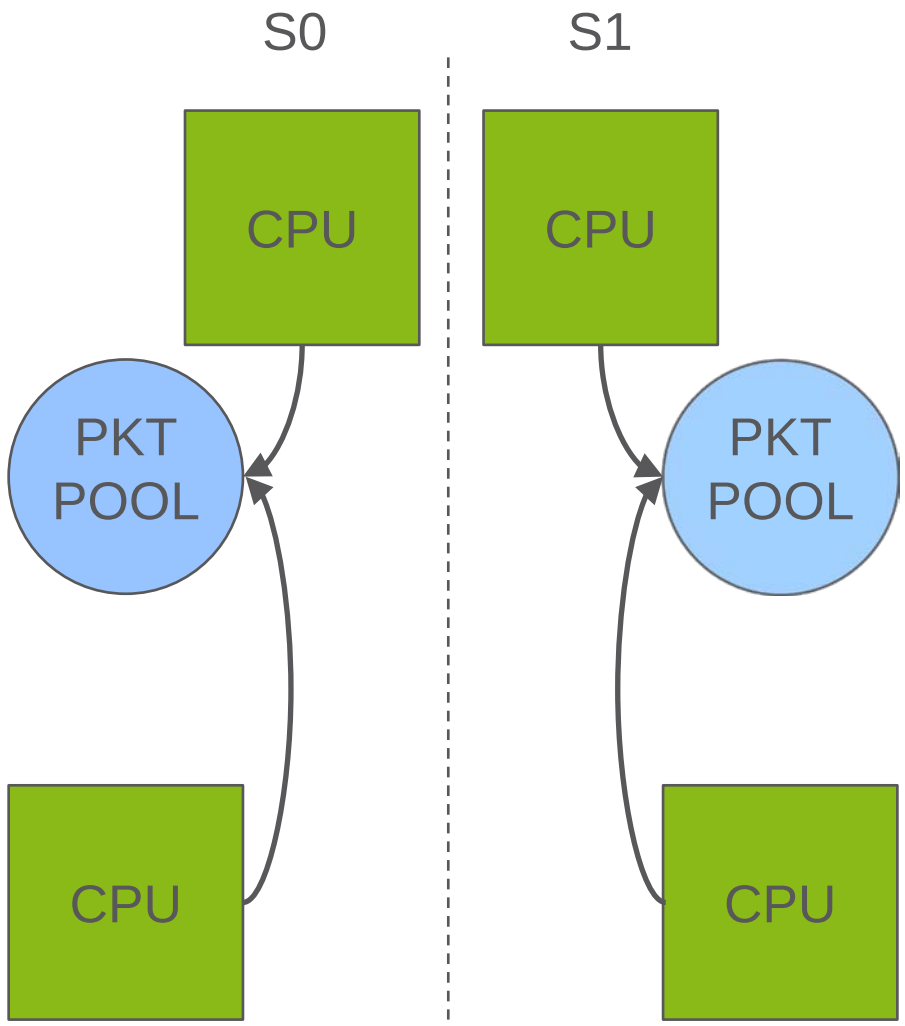
CONFIG EXAMPLE




```
cpualias foreground {
      cpumask = "2,12"
}


pktpool pktpool_fg {
      cpualias = foreground
      num_pkts = 8K
      num_cached_pkts = 64
      type = numa
}
```

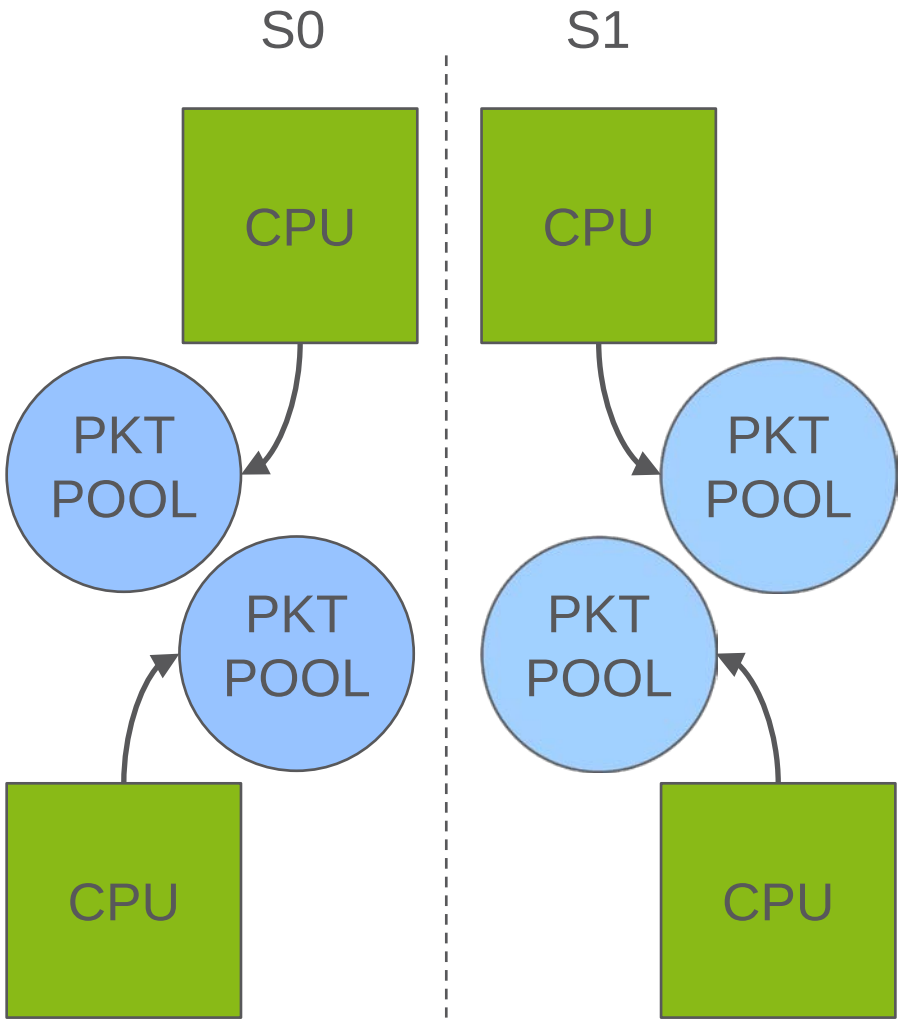
CONFIG EXAMPLE

```
cpualias foreground {
      cpumask = "2,4,12,14"
}


pktpool pktpool_fg {
      cpualias = foreground
      num_pkts = 8K
      num_cached_pkts = 64
      type = numa
}
```



CONFIG EXAMPLE

```
cpualias foreground {
      cpumask = "2,4,12,14"
}


pktpool pktpool_fg {
      cpualias = foreground
      num_pkts = 8K
      num_cached_pkts = 64
      type = exclusive
}
```



RESOURCE ALLOCATION

- resource allocation (static or on-demand)
- chain of pools in case of dynamic allocation (example)

RESMGR AT WORK

- e.g. a native workstation with 4 sockets, memory, interfaces.
- resmgr starts, autodetects (CPUs, HT siblings, cache sizes, etc) every resources we can use.
- resmgr: daemon/binary/library
- reading configuration and creating a free resource database (our choice is xattr)
- supporting popup cpu, memory, interface, etc
- linking/referencing resources (cpu table, HT table, NUMA, L3 cache (CAT/cache QoS), etc)

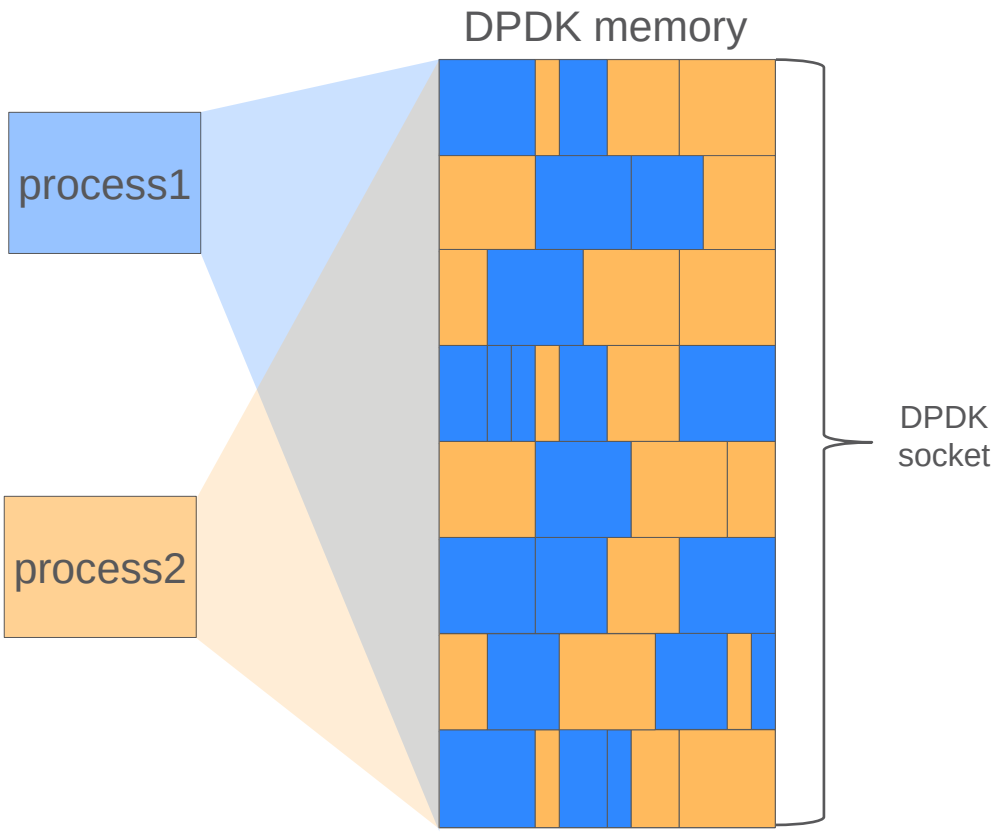
PACKET DOMAIN

- Is the relation between virtual ports, virtual queues and named packet pools. Somewhat similar to pipeline model configuration.

DPDK DOMAIN (APPLICATION DOMAIN)

- Allocating resources
- Per-application configuration
- Application instances see only assigned resources (like a VM)

EXISTING MEMORY MANAGEMENT EXAMPLE



MEMORY MANAGEMENT GOALS

- Simple memory management API
- Transparent NUMA awareness for applications
  - Even in a non-NUMA aware guest VM environment
- Flexible, configuration, re-configuration
- Provide a more granular way of placing objects in memory
  - e.g. to control the number of TLB cache entry usage
- Memory partitioning
  - Physically contiguous memory, hugepages etc.
  - Partitioning across application instances
  - Shared memory support

WHAT IS A MEMDOMAIN?

- Named and indexed memory partition (shared memory) with defined properties such as
  - Location, e.g. NUMA node(s)
  - Size of partition, e.g. 4GB/1G, 512MB/2M hugepages
  - Physical contiguity
  - Type (future improvement), e.g.
    - Hugepage memory (only possible type today)
    - Mmap-ped file
    - Inter-vm shared memory
    - Distributed memory
    - etc.

WHAT IS A MEMDOMAIN POOL?

- Is a pool of shared memory resources (memdomains)
- Is a named group of memory partitions with different location attributes
- Three main types
  - DEFAULT
    - No location preference, only a single memdomain in the pool
  - NUMA
    - One memdomain per involved NUMA node
  - EXCLUSIVE
    - One memdomain per application instance

MEMDOMAIN CONFIGURATION

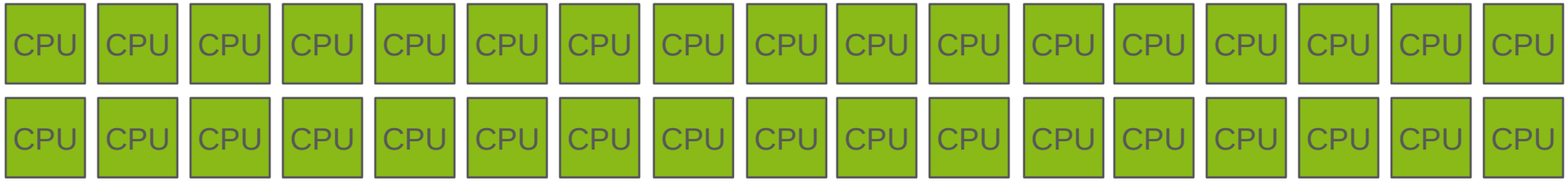
- Can be compared to partitioning a hard drive where
  - Available hugepage memory is the hard drive, e.g. one drive per NUMA node
  - Memdomains can be compared to disk partitions
  - Memzones can be compared to files
- Should be a system engineering task to find the best model working for the specific application model

MEMDOMAIN SIMPLE APPLICATION API

- Initialize library (oai_memcfg_libinit)
  - Register a unique name to the process (High Availability, debug)
  - Initializes DPDK memory, grabs hugepages etc.
- Attach to named memory partition (oai_memcfg_attach)
  - Using pre-configured memdomain pool name
  - Automatic memdomain selection from pool (per instance, NUMA)
  - Maps memdomain (memzones) into virtual address space

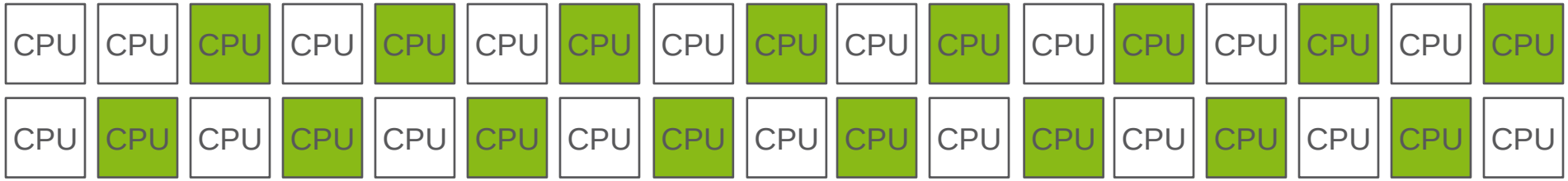
MEMDOMAIN CONFIGURATION EXAMPLE
CPUALIAS

```
cpualias all {
    cpumask = "0-31"    # all available CPU
}
```



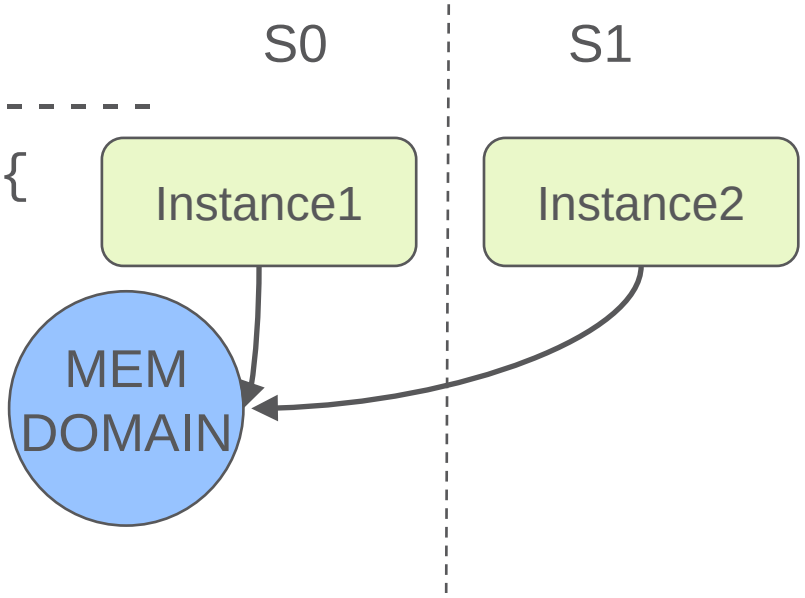
MEMDOMAIN CONFIGURATION EXAMPLE
CPUALIAS

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



MEMDOMAIN CONFIGURATION EXAMPLE
DEFAULT TYPE

```
# -------------------------------
#  Shared memory accessed by
#  all instances
# -------------------------------
memdomain App_Shared_Memory {
      type = default
      cpualias = "all"
      alloc_memzone = true
      size {
            huge_2M = 512M
            huge_1G = 0
      }
}
```



MEMDOMAIN API ATTACH EXAMPLE
DEFAULT TYPE

Instance1:

```
- oai_memcfg_libinit("AppProcess1");
- addr1 = oai_memcfg_attach("App_Shared_Memory",
                            OAI_MEMCFG_MD_INDEX_AUTO,
                            &len,
                            OAI_MEMCFG_MZ_FLAGS_NONE);
```

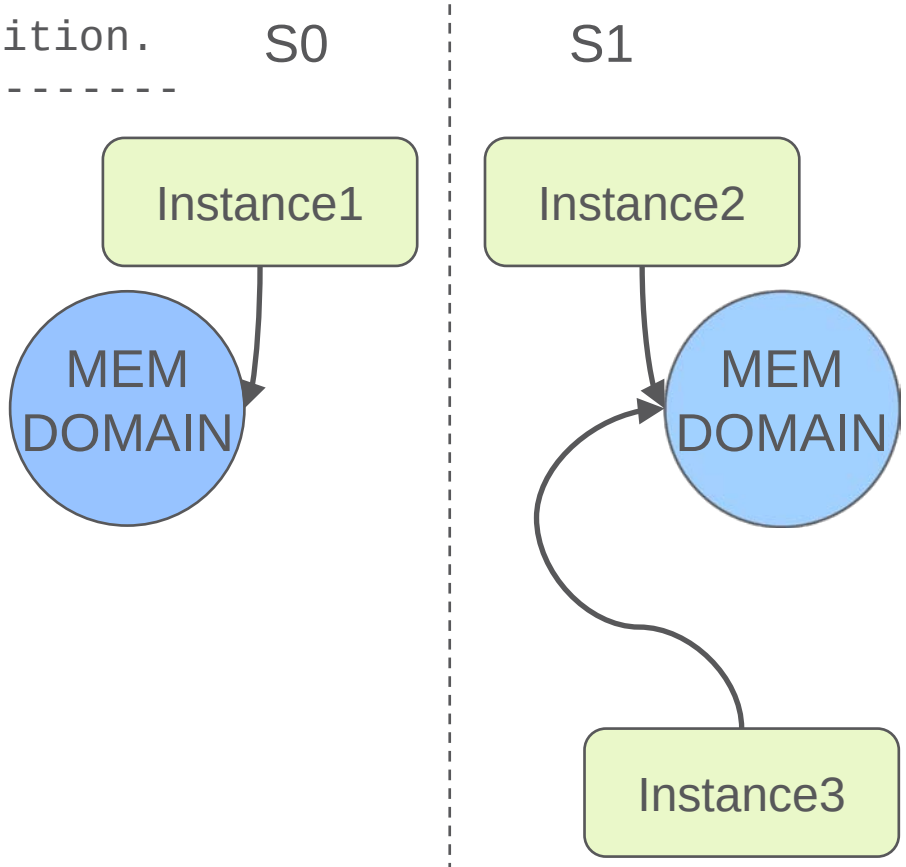
Instance2:

```
- oai_memcfg_libinit("AppProcess2");
- addr2 = oai_memcfg_attach("App_Shared_Memory",
                            OAI_MEMCFG_MD_INDEX_AUTO,
                            &len,
                            OAI_MEMCFG_MZ_FLAGS_NONE);
```

- Explanation:
  - addr1 in process1 will point to the same memory as addr2 in process2

MEMDOMAIN 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
        }
}
```



MEMDOMAIN API ATTACH EXAMPLE
NUMA TYPE

Instance1:

```
- oai_memcfg_libinit("AppProcess1");
- pin to vcpu on numa node0 (lcore_assign, pktpthread_setaffinity_np)
- addr1 = oai_memcfg_attach("App_NUMA_Shared",
                            OAI_MEMCFG_MD_INDEX_AUTO,
                            &len,
                            OAI_MEMCFG_MZ_FLAGS_NONE);
```

Instance2:

```
- oai_memcfg_libinit("AppProcess2");
- pin to vcpu on numa node1 (lcore_assign, pktpthread_setaffinity_np)
- addr2 = oai_memcfg_attach("App_NUMA_Shared",
                            OAI_MEMCFG_MD_INDEX_AUTO,
                            &len,
                            OAI_MEMCFG_MZ_FLAGS_NONE);
```

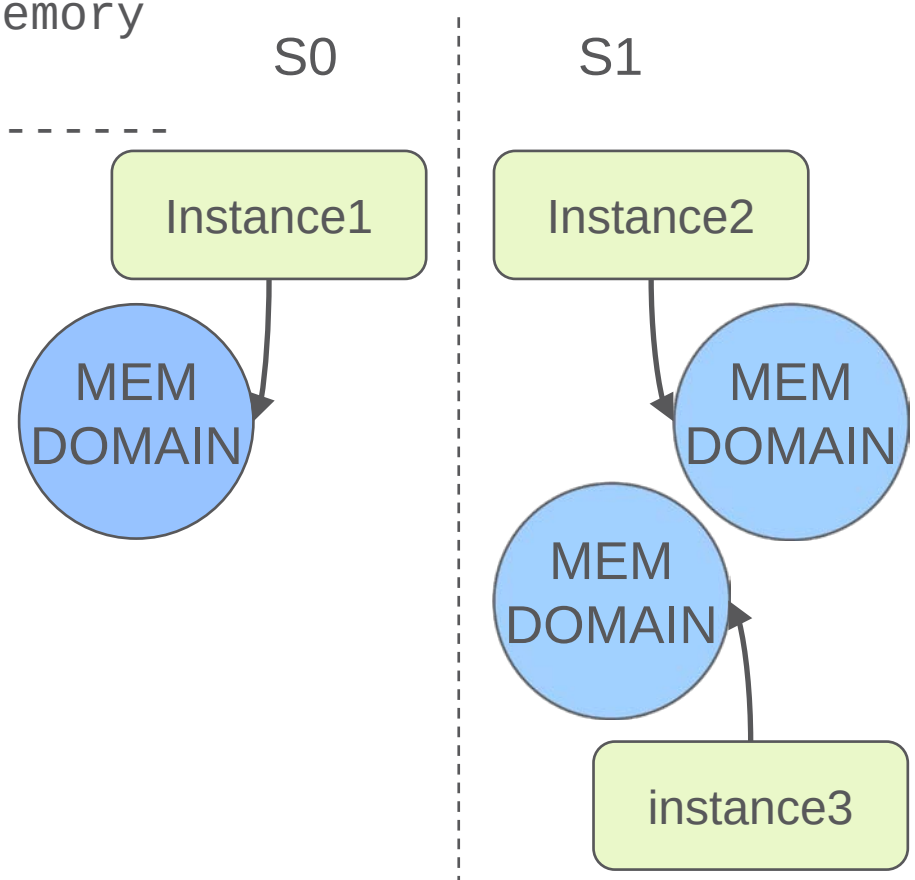
Instance3:

```
- oai_memcfg_libinit("AppProcess3");
- pin to vcpu on numa node1 (lcore_assign, pktpthread_setaffinity_np)
- addr3 = oai_memcfg_attach("App_NUMA_Shared",
                            OAI_MEMCFG_MD_INDEX_AUTO,
                            &len,
                            OAI_MEMCFG_MZ_FLAGS_NONE);
```

- Explanation:
  - addr1 in instance1 will point to the partition allocated in NUMA node0 while addr2 in instance2 and addr3 in instance3 will point to the same partition allocated in NUMA node1

MEMDOMAIN 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
        }
}
```



MEMDOMAIN API ATTACH EXAMPLE
EXCLUSIVE TYPE

Instance1:

```
- oai_memcfg_libinit("AppProcess1");
- pin to vcpu2 on numa node0 (lcore_assign, pktpthread_setaffinity_np)
- addr1 = oai_memcfg_attach("App_Thread_Local",
                            OAI_MEMCFG_MD_INDEX_AUTO,
                            &len,
                            OAI_MEMCFG_MZ_FLAGS_NONE);
```

Instance2:

```
- oai_memcfg_libinit("AppProcess2");
- pin to vcpu16 on numa node1 (lcore_assign, pktpthread_setaffinity_np)
- addr2 = oai_memcfg_attach("App_Thread_Local",
                            OAI_MEMCFG_MD_INDEX_AUTO,
                            &len,
                            OAI_MEMCFG_MZ_FLAGS_NONE);
```

Instance3:

```
- oai_memcfg_libinit("AppProcess3");
- pin to vcpu18 on numa node1 (lcore_assign, pktpthread_setaffinity_np)
- addr3 = oai_memcfg_attach("App_Thread_Local",
                            OAI_MEMCFG_MD_INDEX_AUTO,
                            &len,
                            OAI_MEMCFG_MZ_FLAGS_NONE);
```

- Explanation:
  - Addr1, addr2 and addr3 will point to different memory addresses
  - addr1 allocated on NUMA node0 while addr2 and addr3 allocated on NUMA node1

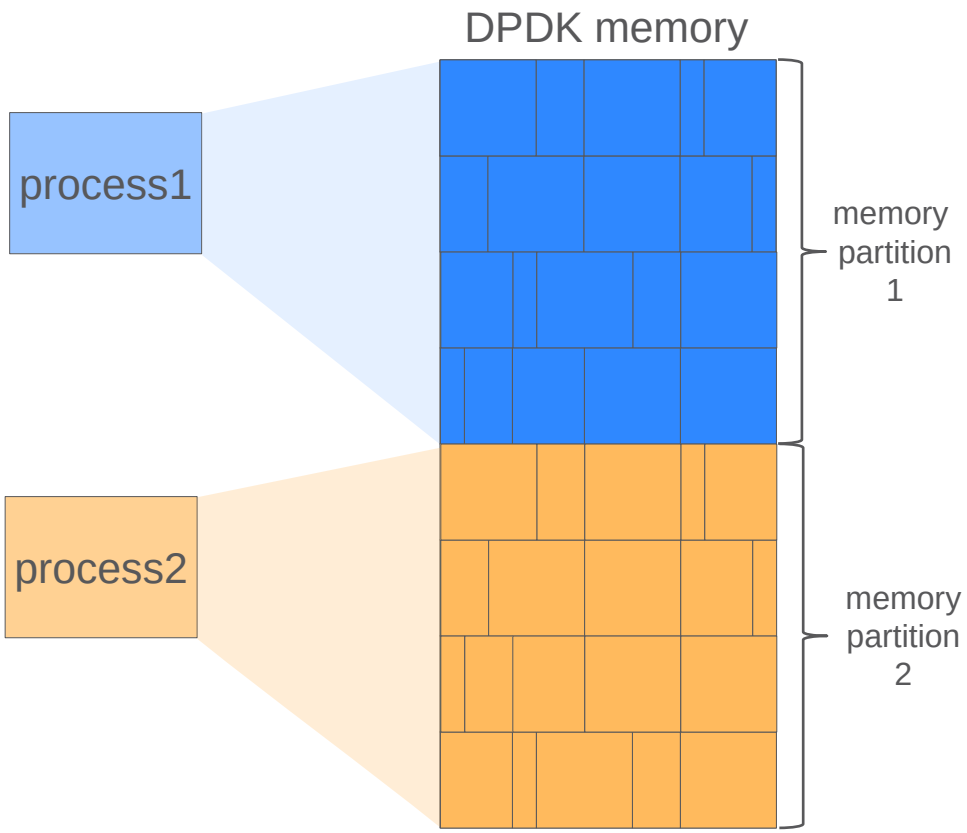
ADVANCED MEMCFG API

- Create memdomain pools in a transaction
  - Creating a pool of memory partitions
    (partitions are auto aligned based on user's location or requested layout)
  - Creating multiple memdomain pools in a transaction allows better placement (e.g. prioritizing physically contiguous memory)
- Attach to memdomain (memory partition)
  - Attach by memdomain name and index, returning md handle
    (index could be auto if instance location is known e.g. by lcoreid)
  - Could also MMAP whole partition into virtual address space
- Get number of memory partitions in a memdomain pool
  - Allows attaching to all partitions by index within a pool
    e.g. for per NUMA table replication

ADVANCED MEMCFG API

- Reserve named memzones by memdomain handle
  - Memzone name must only be unique within the memdomain, same name can be used in another memdomain
    e.g. it allows attaching to replicas using the same name
  - supports fragments if physical contiguity is not requested (virtually cont)
- Lookup memzone by memdomain handle
  - Looks up memzone within a memdomain returning mz handle
- Attach to memzone by memzone handle
  - Attaches to a memzone, e.g. mapping into requested/reserved virtual address space as readonly/readwrite

MEMDOMAIN EXAMPLE



FUTURE IDEAS

- SUPPORT NON-DPDK APPLICATIONS
- MEMORY TYPES FOR RTE_MALLOC
- DPDK CERTIFICATION (FOR VM OR HARDWARE)
- GUI OR CONFIG FILE FOR DPDK STARTUP PARAMETERS (RESMGR CONFIG ALTERNATIVE)
- VISUAL RESOURCE ALLOCATOR (GPARTED STYLE)
- DPDK OS / DPDK DISTRO
- STANDBY INSTANCES
- NEW EMULATED QEMU DEVICE

ERICSSON