Hierarchical NUMA

Anshuman Khandual

khandual@linux.vnet.ibm.com
Linux Technology Centre
IBM

Contents

- NUMA Systems
- NUMA in Memory Management
- Coherent Device Memory
- Existing CDM Solutions
- Existing NUMA Design Limitations
- Need for Attribute based NUMA
- Hierarchical NUMA Proposal
- Conclusion

Why NUMA

- Some workloads benefit from a single application image running on a big NUMA system
- Accessing an extensive set of processor cores and large shared memory for it's working set
- Require a **large working memory set** to be processed in **multiple chunks** through **multiple threads**
- Primarily **enterprise** and **HPC** workloads fall in this category

  For example, the following class of applications fall in that category

  - In Memory DB
  - ERP
  - CRM
  - Business Intelligence
  - Scientific Research
  - Virtualization

  Example of some commercially available NUMA systems

  - IBM POWER8 E880
  - Dell PowerEdge R930
  - HP DL980
  - Supermicro SYS-7088B-TR4FT

- NUMA Systems
- NUMA in Memory Management
- Coherent Device Memory
- Existing CDM Solutions
- Existing NUMA Design Limitations
- Need for Attribute based NUMA
- Hierarchical NUMA Proposal
- Conclusion

NUMA Properties

- Systems implement NUMA through various types of NUMA interconnects
- NUMA interconnects can be with or without any data routers
- From CPU's perspective, distance is just access latency
- Latency depends on NUMA interconnect, number of hops, system bus speed etc
- Represented as a **distance** factor, how far is the memory from a given CPU
- Platform firmware must provide **node_distance**[MAX_NUMNODES][MAX_NUMNODES]
- Arch code processes ACPI (X86) or DT (PowerPC) to fetch distance information from firmware
- Arch code needs to export **node_distance(a, b)** for the core kernel to use every where else

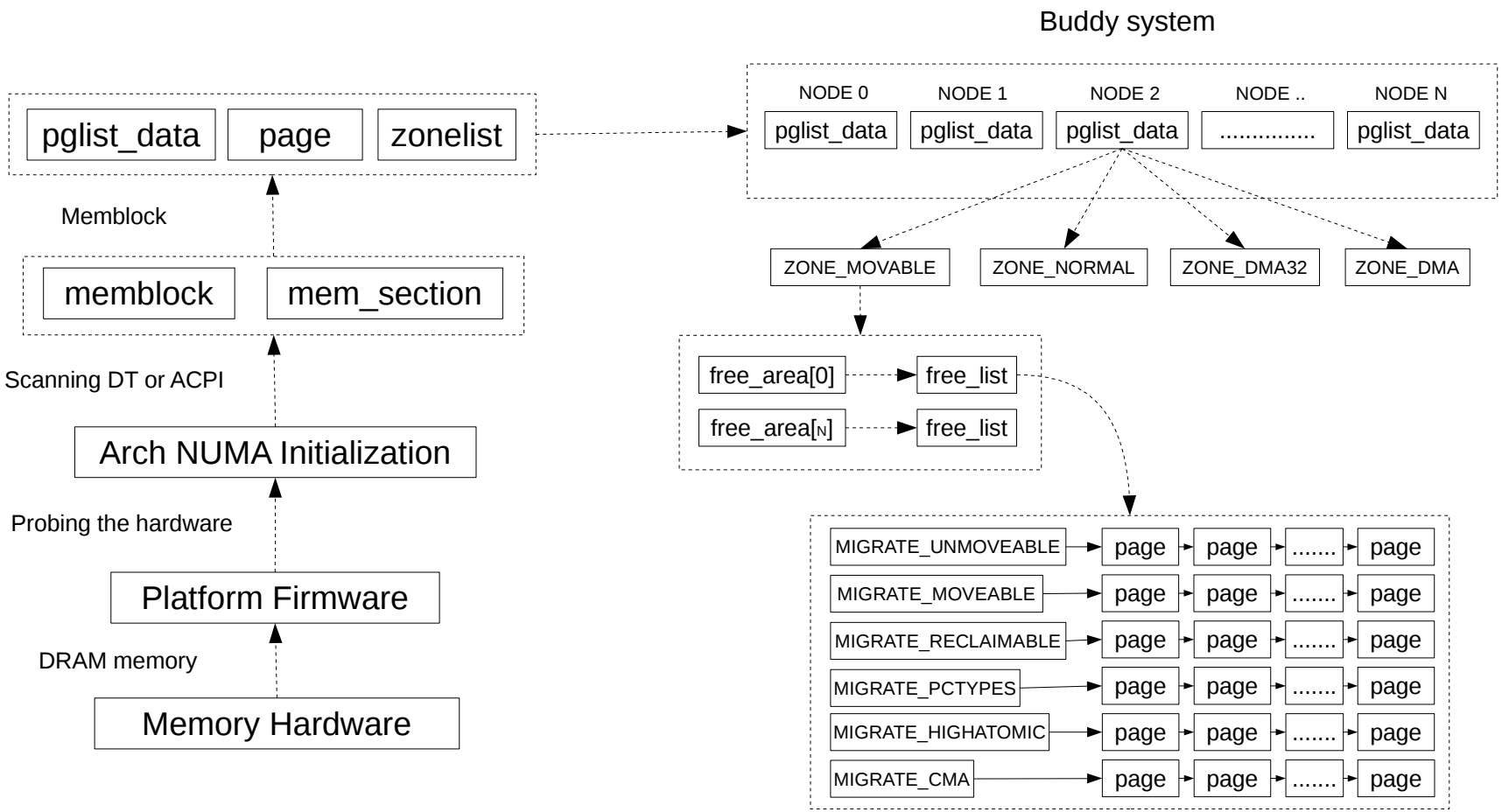
```
NUMA initialization on POWER

node_distance()
     __node_distance()
          distance_lookup_table[MAX_NUMNODES][MAX_DISTANCE_REF_POINTS]
               associativity(distance_ref_points[])

NUMA initialization on X86

node_distance()
     __node_distance()
          numa_distance[]
```

NUMA Memory Representation



- NUMA Systems
- NUMA in Memory Management
- Coherent Device Memory
- Existing CDM Solutions
- Existing NUMA Design Limitations
- Need for Attribute based NUMA
- Hierarchical NUMA Proposal
- Conclusion

Coherent Device Memory (CDM)

- Non system RAM memory which can be accessed coherently from the CPU
- They are similar to system RAM in many characteristics but are still different
- They are comparable in size with system RAM
- They offer specialized functions for applications and drivers to use
- They may be accessed by other forms of device compute

Existing CDM devices

- POWER9 processor accessing NVIDIA Volta GPU over NVLink2
- Persistent Memory (NVDIMM) on both PowerPC and X86 systems
- Intel MCDRAM is a High Bandwidth Memory (HBM) on the chip

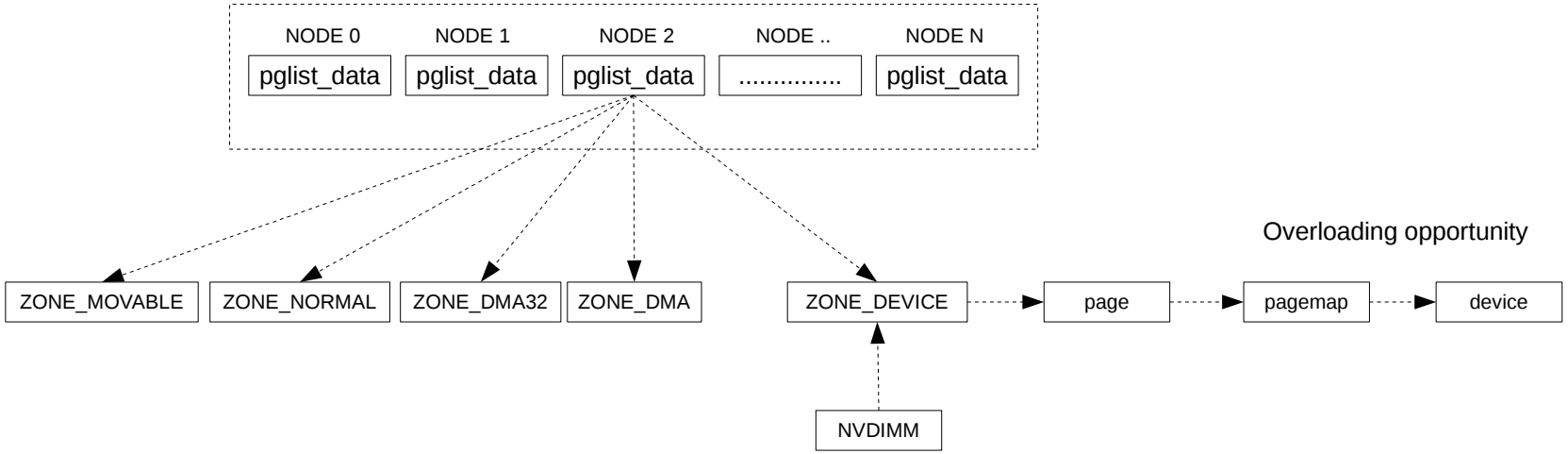
Different DRAM

- Existing systems can have DDR2, DDR3 and DDR4, some times in the same system
- DDR5 will be coming in two forms (low power format and standard one)
- They can have different latency, bandwidth and power consumption properties

- NUMA Systems
- NUMA in Memory Management
- Coherent Device Memory
- **Existing CDM Solutions**
- Existing NUMA Design Limitations
- Need for Attribute based NUMA
- Hierarchical NUMA Proposal
- Conclusion

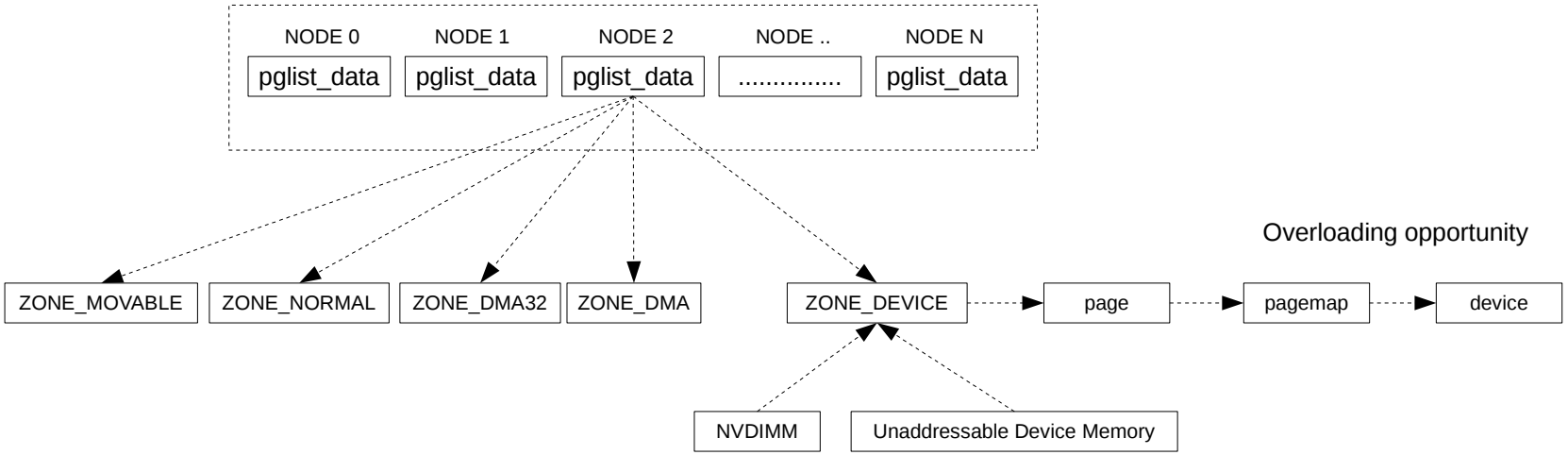
ZONE_DEVICE

- Created a brand new zone for persistent memory or NVDIMM
- Developed to represent NVDIMM or persistent memory inside the kernel
- It helps support DAX file system on those devices
- Persistent memory is fast enough, page cache layer is not required
- These memory can have their own **struct pages**, giving it access to some core MM features
- It attempts to integrate NVDIMM in the kernel while still keeping its own specializations



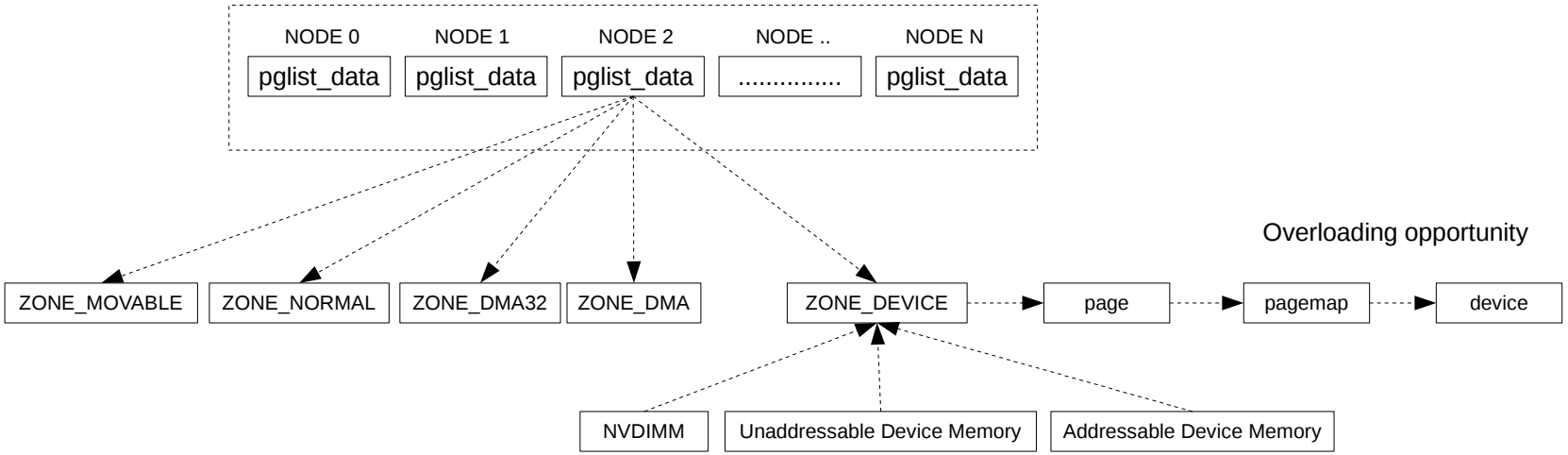
Heterogeneous Memory Management (HMM)

- Developed to represent **unaddressable** device memory as coherent
- Creates a new memory type in ZONE_DEVICE for **unaddressable** memory
- Applicable to device memory which does not support coherency, hence kernel provides coherency
- Kernel achieves the coherency with mirrored page tables and on demand migrations during access
- It helps map completely **unaddressable** memory in user space as well



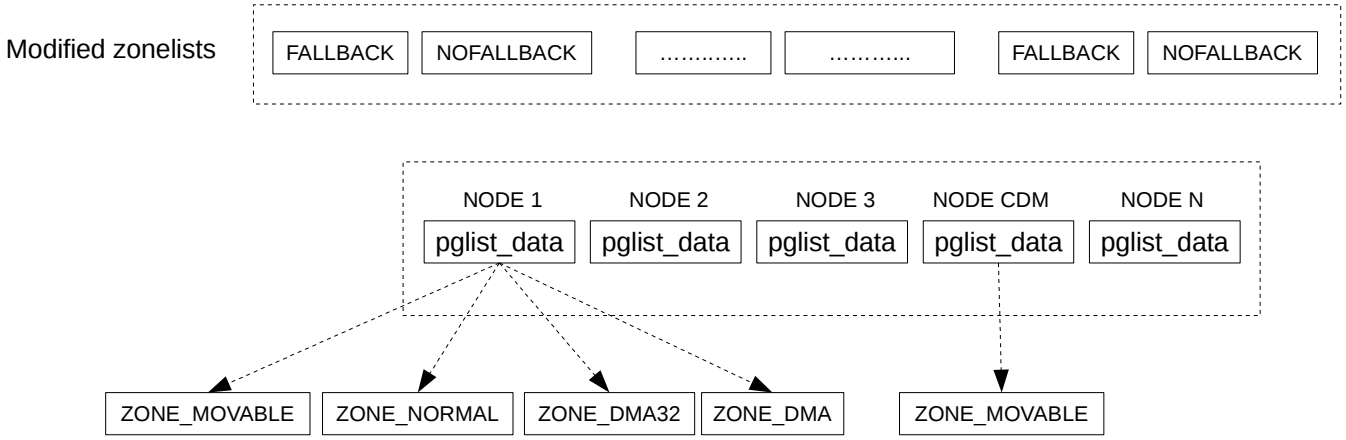
HMM CDM (Addressable Coherent Memory)

- Modified version of HMM to support devices with coherent memory
- Creates yet another memory type for coherent device memory in ZONE_DEVICE
- Drops the mirrored page table as the hardware already has the coherency support
- Migration interfaces changed to accommodate this new coherent ZONE_DEVICE memory type



CDM NUMA

- Represents CDM as NUMA node to achieve seamless integration
- Device memory remains in ZONE_MOVABLE to avoid kernel allocation into CDM
- Implicit allocation from user space should be avoided to the CDM
- Zonelists (FALLBACK and NOFALLBACK) format changed to achieve the desired isolation
- Applications can have CDM memory through mmap(MPOL_BIND, ……) interface explicitly
- mbind() implementation got changed to walk new zonelists containing CDM zones



Intel MCDRAM

- Multi Channel DRAM (MCDRAM)
- High Bandwidth Memory (~4x compared to DRAM) but low capacity (Upto 16GB)
- Packaged inside Xeon Phi X200 chip (Knights Landing)
- Can be configured as a third level cache or as a **distinct NUMA node**
- Data placement on MCDRAM happens through numactl (mbind), memkind and AutoHBW libraries
- This signifies how application can use **specialized memory** for it's advantage

Further Reading

- https://software.intel.com/en-us/blogs/2016/01/20/an-intro-to-mcdram-high-bandwidth-memory-on-knights-landing
- https://software.intel.com/sites/default/files/managed/5f/5e/MCDRAM_Tutorial.pdf

- NUMA Systems
- NUMA in Memory Management
- Coherent Device Memory
- Existing CDM Solutions
- Existing NUMA Design Limitations
- Need for Attribute based NUMA
- Hierarchical NUMA Proposal
- Conclusion

Existing NUMA Limitations (Representing CDM)

- Distance, hence latency is not the only memory attribute
- Does not recognize other properties like bandwidth, reliability, power consumption etc
- There can be different types of memory at the same distance (may be connected by different interfaces)
- So what should be the properties of memory ?
    - Latency (Speed of random access)
    - Bandwidth (Speed of contiguous data stream processing)
    - Reliability (Probable rate of memory failure)
    - Power consumption (Power consumption in holding the data in memory)
    - Density (Compact allocation, power saving, anything else)

- NUMA Systems
- NUMA in Memory Management
- Coherent Device Memory
- Existing CDM Solutions
- Existing NUMA Design Limitations
- Need for Attribute based NUMA
- Hierarchical NUMA Proposal
- Conclusion

Need for Attribute Based NUMA

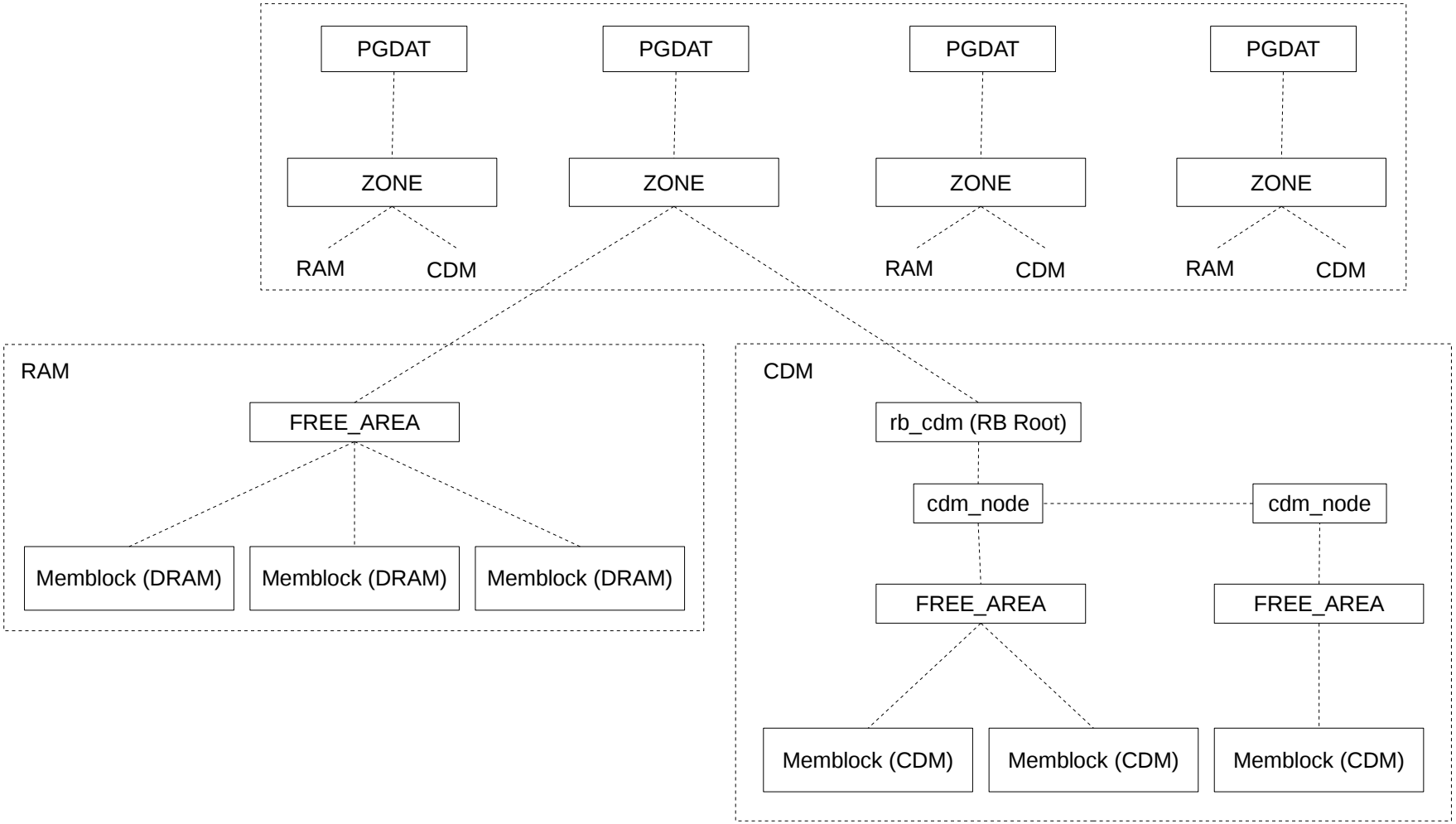
- All existing solutions attempt to make core MM understand the very concept of special memory
- ZONE_DEVICE, HMM, HMM CDM
    - Does not provide explicit user interface for CDM allocation
    - Driver managed and hidden from userspace
    - Being on ZONE_DEVICE, it is already isolated from standard allocation paths
- NUMA CDM
    - Provides explicit user interface for CDM allocation
    - Does invasive changes in kernel to achieve isolation, reclaim and compaction
- For any future CDM memory, attempts will be made to change core kernel to suit its particular needs
- Core MM does not differentiate between **various types** of memory with **different attributes**
- Hence, there is a need for unified representation of different kinds of memory in the kernel
- Traditional distance based NUMA has to change to accommodate these new kinds of memory

- NUMA Systems
- NUMA in Memory Management
- Coherent Device Memory
- Existing CDM Solutions
- Existing NUMA Design Limitations
- Need for Attribute based NUMA
- Hierarchical NUMA Proposal
- Conclusion

Hierarchical NUMA Proposal

- Changes to physical memory representation
  - Changes to memblock
  - Changes to buddy System
- Changes to virtual memory representation
  - New system calls
  - Changes to VMA
  - Changes to memory policy
- Changes to memory allocation
  - New GFP flags
  - Changes to __alloc_pages_nodemask()
  - Changes to get_pages_from_freelist()
  - Changes to __rmqueue_smallest()
  - New zone_to_area_mattr() function
- Changes to other memory functions

Physical Memory Representation



Cumulative mattr value is the key for the RB Tree

Memblock

```
struct memblock_region {
    phys_addr_t base;
    phys_addr_t size;
    unsigned long flags;                /* VMA flags */
    unsigned long mattr;                /* Memory attribute flags */
#ifdef CONFIG_HAVE_MEMBLOCK_NODE_MAP
    int nid;
#endif
};
```

```
MEM_LATENCY_MASK      0x000000000000000F    /* Memory latency */
MEM_BANDWIDTH_MASK    0x00000000000000F0    /* Memory bandwidth */
MEM_RELIABILITY_MASK  0x0000000000000F00    /* Memory reliability */
MEM_POWERVALUE_MASK   0x000000000000F000    /* Memory power consumption */
MEM_DEVICECOMP_MASK   0x00000000000F0000    /* Memory device compute */
MEM_DENSITY_MASK      0x0000000000F00000    /* Memory density */

MEM_LATENCY_SHIFT         0
MEM_BANDWIDTH_SHIFT       4
MEM_RELIABILITY_SHIFT     8
MEM_POWERVALUE_SHIFT      12
MEM_DEVICECOMPUTE_MASK    16
MEM_DENSITY_MASK          20
```

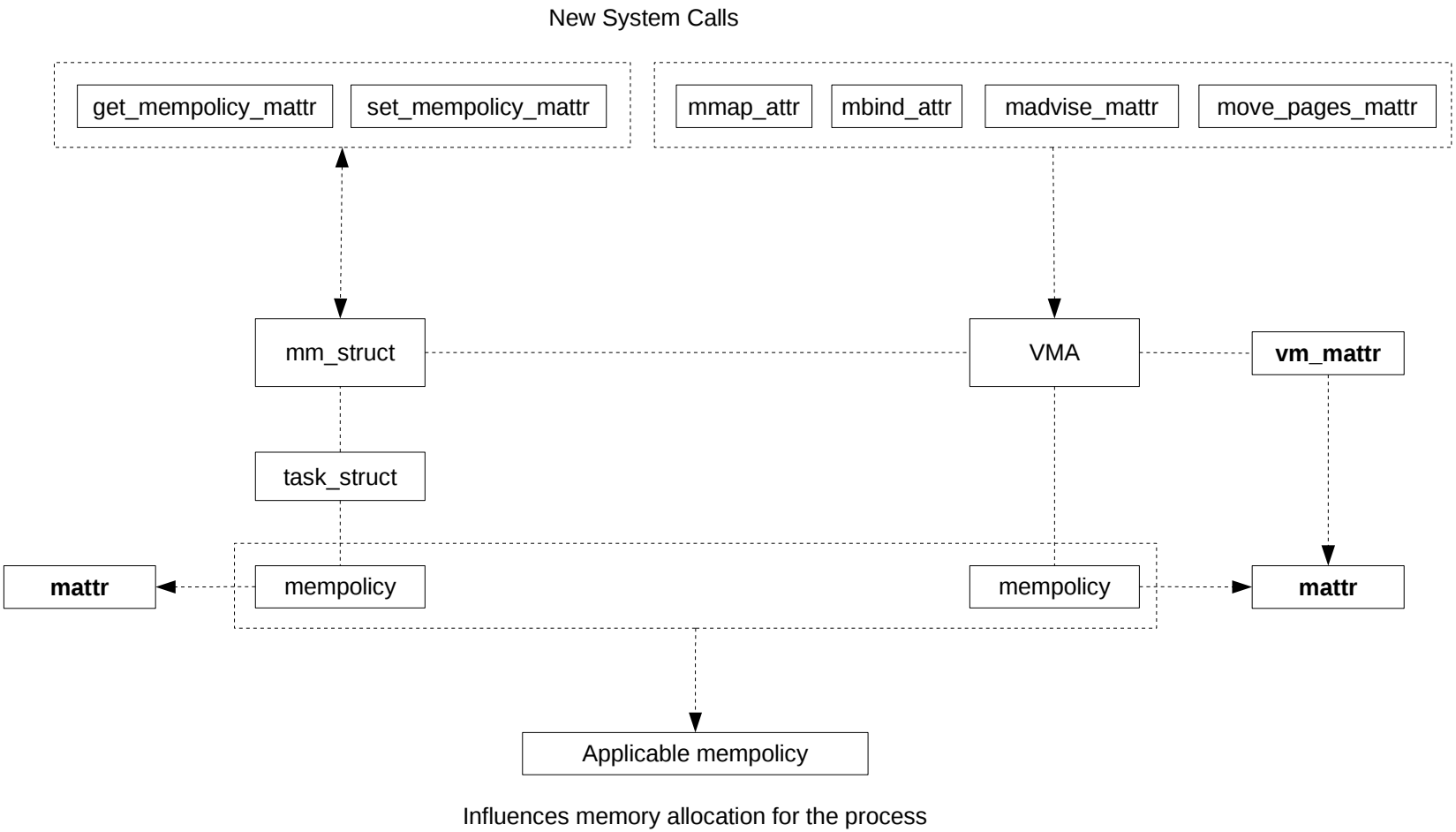
Buddy System (PGLIST_DATA)

```
struct cdm_node {
    struct rb_node node;                    /* Attached to rb_cdm */
    unsigned long mattr;                    /* Memblock attributes */
    struct free_area free_area[MAX_ORDER]; /* CDM buddy system */
  }
```

```
struct zone {
    ……————————
    struct free_area free_area[MAX_ORDER}; /* Contains DRAM */
    struct rb_root rb_cdm;                  /* Contains CDM */
    unsigned long nr_cdm;                   /* Total CDM nodes on the zone */
    struct cdm_node *cdm_ind[MAX_PROPERTY][MAX_CDM];    /* Search cache */
    struct cdm_node *cdm_cum[MAX_CDM];                  /* Search cache */
}
```

- zone→(rb_cdm, nr_cdm, cdm_ind, cdm_cum) gets updated during boot and memory hotplug
- Contiguous memory_regions need to have same memory attribute to be a single CDM node
- A zone can contain multiple CDM nodes with different properties forming a RB tree
- Cumulative value of ‘mattr’ is the key while inserting the node into RB tree
- Cumulative value adds up all memory attribute values for the CDM
- CDM nodes remain balanced in the RB tree and forms a 'hierarchical' system

Virtual Memory Representation



Virtual Memory Area (VMA)

```
struct vma_area_struct {
    …...
    unsigned long vm_flags;   /* Existing VMA flags */
    unsigned long vm_mattr;   /* Contains memory attribute flags */
    …...
}
```

```
VM_LATENCY_MASK       0x000000000000000F   /* Memory latency */
VM_BANDWIDTH_MASK     0x00000000000000F0   /* Memory bandwidth */
VM_RELIABILITY_MASK   0x0000000000000F00   /* Memory reliability */
VM_POWERVALUE_MASK    0x000000000000F000   /* Memory power consumption */
VM_DEVICECOMP_MASK    0x00000000000F0000   /* Memory device compute */
VM_DENSITY_MASK       0x0000000000F00000   /* Memory density */

VM_LATENCY_SHIFT          0
VM_BANDWIDTH_SHIFT        4
VM_RELIABILITY_SHIFT      8
VM_POWERVALUE_SHIFT       12
VM_DEVICECOMPUTE_MASK     16
VM_DENSITY_MASK           20
```

- VMA memory attributes look similar to that of memblock ? They are but
- VMA memory attributes will be ABI where as memblock attributes can change later

Memory Policy

```
struct mempolicy {
    atomic_t refcnt;
    unsigned short mode;
    unsigned short flags;
    unsigned long mattr;                    /* Contains VMA mattr flags */
    union {
        short         preferred_node;
        nodemask_t    nodes;
    } v;
    union {
        nodemask_t cpuset_mems_allowed;
        nodemask_t user_nodemask;
    } w;
};
```

- Memory policy structure can belong to either to a VMA or to the entire task
- Why copy VMA flags into memory policy ? It improves performance during page fault
- During each page fault, mmap_sem need not be held to read vma→vm_mattr

GFP Flags

LATENCY

```
__GFP_LATENCY_1
__GFP_LATENCY_2
__GFP_LATENCY_3
__GFP_LATENCY_4
__GFP_LATENCY_5
__GFP_LATENCY_6
__GFP_LATENCY_7
__GFP_LATENCY_8
__GFP_LATENCY_9
__GFP_LATENCY_10
__GFP_LATENCY_11
__GFP_LATENCY_12
__GFP_LATENCY_13
__GFP_LATENCY_14
__GFP_LATENCY_15
__GFP_LATENCY_16
```

BANDWIDTH

```
__GFP_BANDWIDTH_1
__GFP_BANDWIDTH_2
__GFP_BANDWIDTH_3
__GFP_BANDWIDTH_4
__GFP_BANDWIDTH_5
__GFP_BANDWIDTH_6
__GFP_BANDWIDTH_7
__GFP_BANDWIDTH_8
__GFP_BANDWIDTH_9
__GFP_BANDWIDTH_10
__GFP_BANDWIDTH_11
__GFP_BANDWIDTH_12
__GFP_BANDWIDTH_13
__GFP_BANDWIDTH_14
__GFP_BANDWIDTH_15
__GFP_BANDWIDTH_16
```

RELIABILITY

```
__GFP_RELIABILITY_1
__GFP_RELIABILITY_2
__GFP_RELIABILITY_3
__GFP_RELIABILITY_4
__GFP_RELIABILITY_5
__GFP_RELIABILITY_6
__GFP_RELIABILITY_7
__GFP_RELIABILITY_8
__GFP_RELIABILITY_9
__GFP_RELIABILITY_10
__GFP_RELIABILITY_11
__GFP_RELIABILITY_12
__GFP_RELIABILITY_13
__GFP_RELIABILITY_14
__GFP_RELIABILITY_15
__GFP_RELIABILITY_16
```

GFP Flags (Cont..)

POWER VALUE

```
__GFP_POWERVALUE_1
__GFP_POWERVALUE_2
__GFP_POWERVALUE_3
__GFP_POWERVALUE_4
__GFP_POWERVALUE_5
__GFP_POWERVALUE_6
__GFP_POWERVALUE_7
__GFP_POWERVALUE_8
__GFP_POWERVALUE_9
__GFP_POWERVALUE_10
__GFP_POWERVALUE_11
__GFP_POWERVALUE_12
__GFP_POWERVALUE_13
__GFP_POWERVALUE_14
__GFP_POWERVALUE_15
__GFP_POWERVALUE_16
```

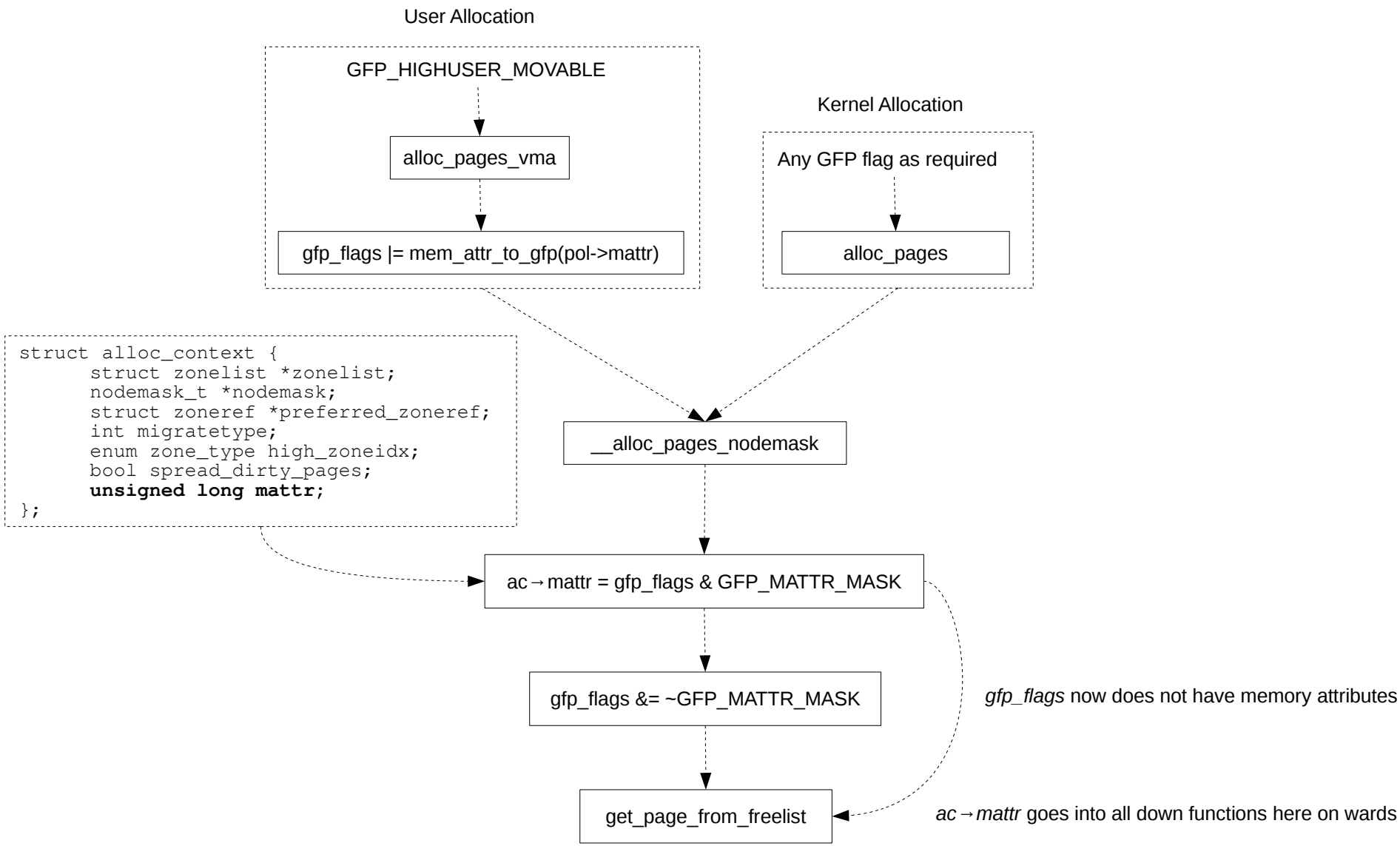
DEVICE COMPUTE

```
__GFP_DEVICECOMP_1
__GFP_DEVICECOMP_2
__GFP_DEVICECOMP_3
__GFP_DEVICECOMP_4
__GFP_DEVICECOMP_5
__GFP_DEVICECOMP_6
__GFP_DEVICECOMP_7
__GFP_DEVICECOMP_8
__GFP_DEVICECOMP_9
__GFP_DEVICECOMP_10
__GFP_DEVICECOMP_11
__GFP_DEVICECOMP_12
__GFP_DEVICECOMP_13
__GFP_DEVICECOMP_14
__GFP_DEVICECOMP_15
__GFP_DEVICECOMP_16
```

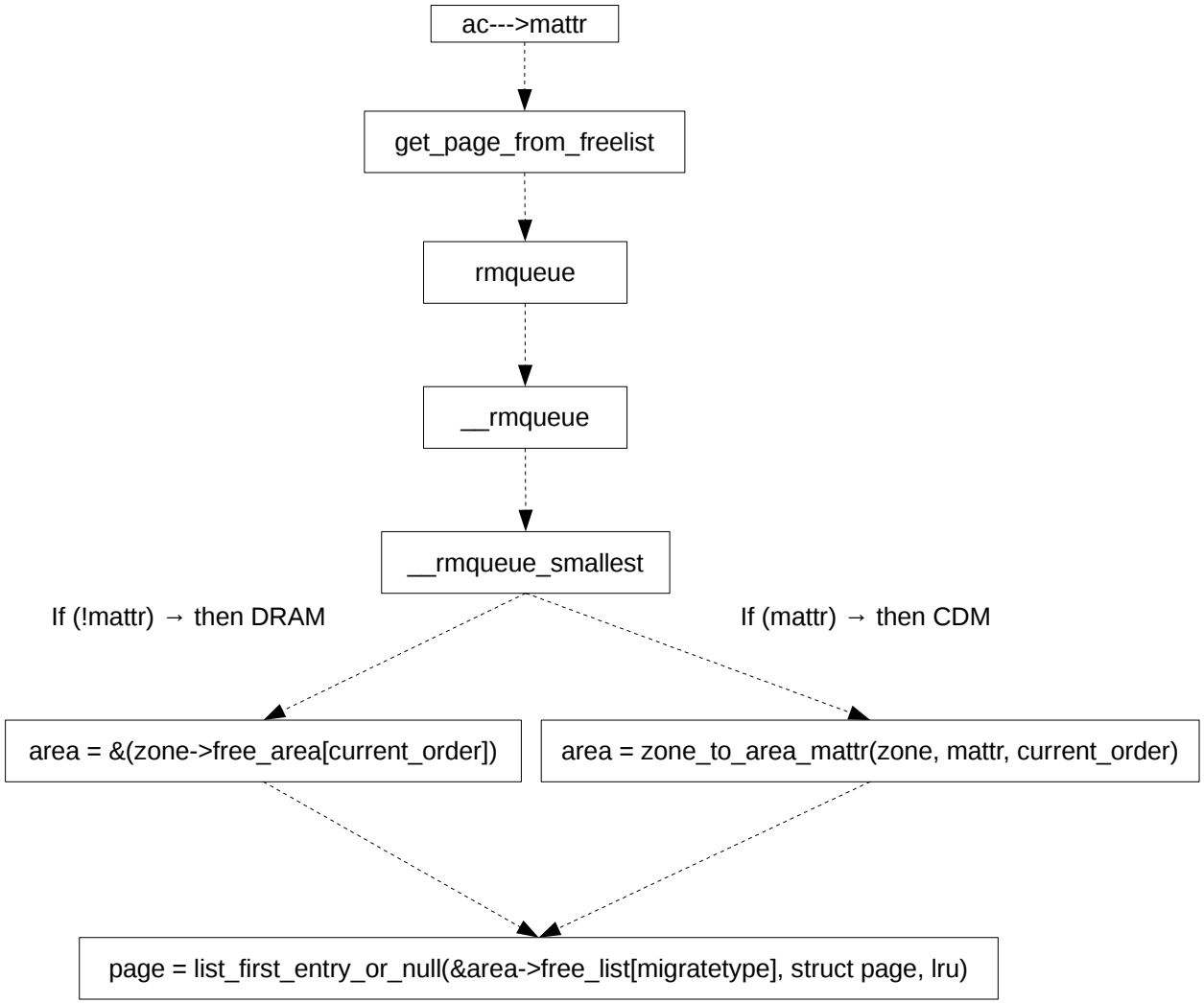
DENSITY

```
__GFP_DENSITY_1
__GFP_DENSITY_2
__GFP_DENSITY_3
__GFP_DENSITY_4
__GFP_DENSITY_5
__GFP_DENSITY_6
__GFP_DENSITY_7
__GFP_DENSITY_8
__GFP_DENSITY_9
__GFP_DENSITY_10
__GFP_DENSITY_11
__GFP_DENSITY_12
__GFP_DENSITY_13
__GFP_DENSITY_14
__GFP_DENSITY_15
__GFP_DENSITY_16
```

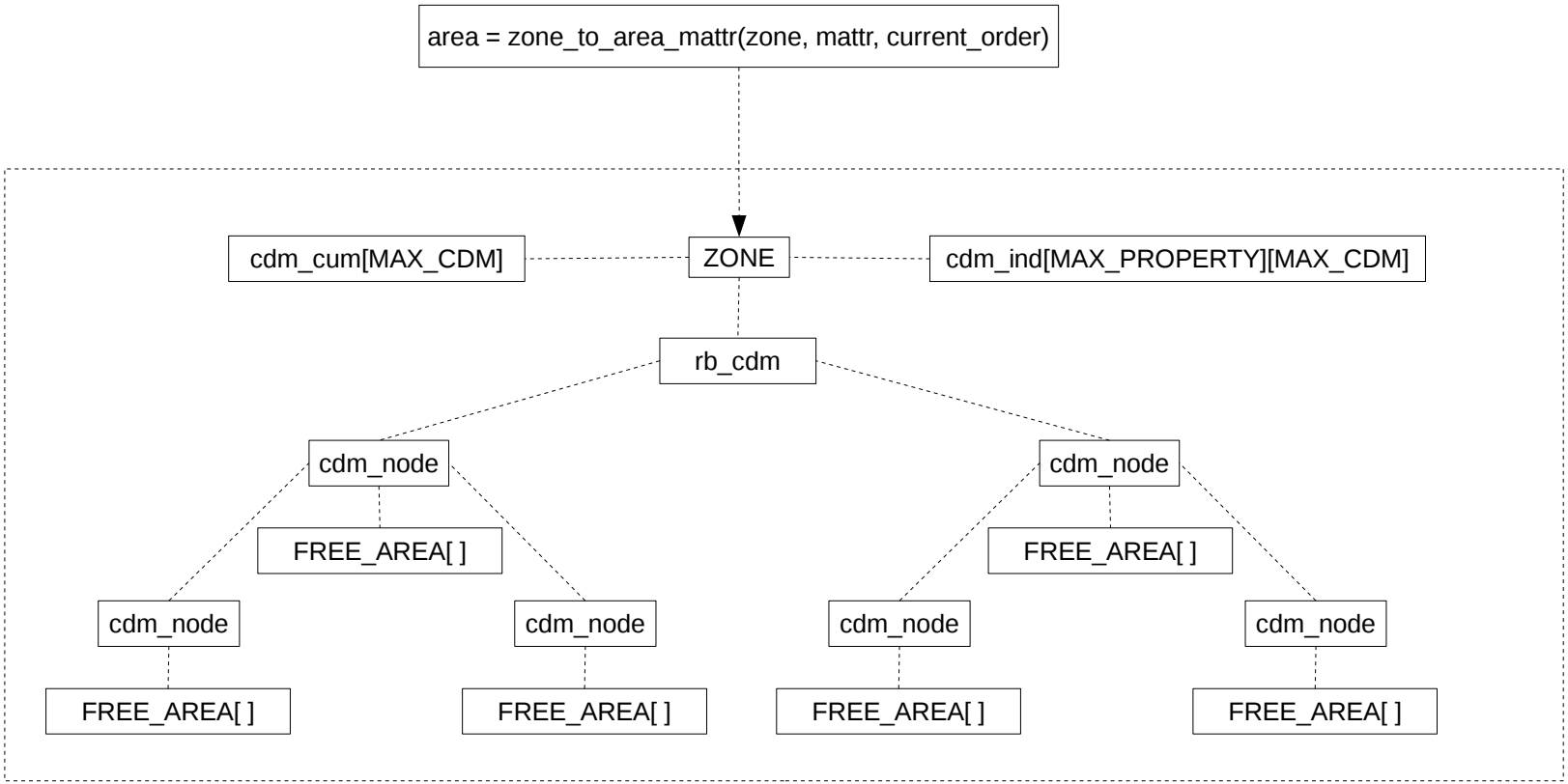
Memory Allocation



Memory Allocation (Cont..)

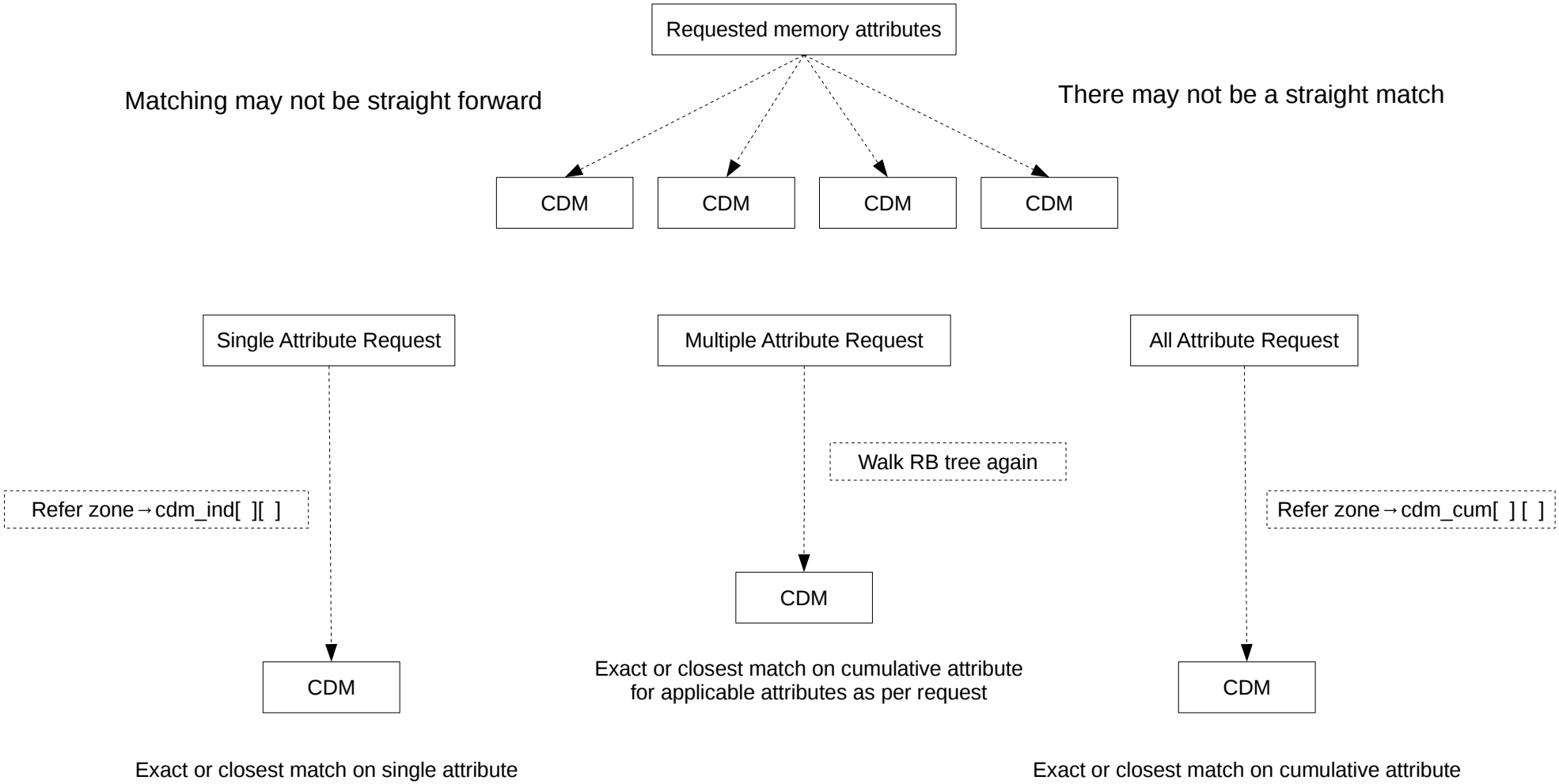


Memory Allocation (Cont..)



- If mattr contains a single attribute, match the exact or closest of the same attribute
- If mattr contains multiple attributes, match the exact or closest match for cumulative value

Memory Attribute Matching



Other Memory Functions

Memory allocation sub functions

- Compaction
  - struct compact_control must contain memory attributes
  - zone_to_area_mattr() is called to find appropriate free_area[order] in a given zone
- Reclaim
  - struct scan_control must contain memory attributes
  - zone_to_area_mattr() is called to find appropriate free_area[order] in a given zone

- NUMA Systems
- NUMA in Memory Management
- Coherent Device Memory
- Existing CDM Solutions
- Existing NUMA Design Limitations
- Need for Attribute based NUMA
- Hierarchical NUMA Proposal
- Conclusion

Conclusion

- There may be other possible solutions with some pros and cons
  - CDM and DRAM zones with memory attributes
  - CDM and DRAM nodes (NODE_DATA) with memory attributes
  - CDM and DRAM pageblocks (memory attribute types in line with migration types)

- The proposed idea here may not be the best possible solution
- The community should debate this proposal to reach at an optimal solution
- Industry trends suggest, going forward CDM will be provided my multiple vendors and OEMs
- Looking forward to build interest in the community to work towards finding a solution
- Linux kernel should be ready ! Memory HW technologies are coming in fast :)

Legal Statement

- This work represents the view of the authors and does not necessarily represent the view of the employers (IBM Corporation).
- IBM and IBM (Logo) are trademarks or registered trademarks of International Business Machines in United States and/or other countries.
- Linux is a registered trademark of Linux Torvalds.
- Other company, product and service names may be trademarks or service marks of others.

References

- http://www-03.ibm.com/systems/in/power/hardware/e880/
- http://www.dell.com/in/business/p/poweredge-r930/pd
- http://h20564.www2.hpe.com/hpsc/doc/public/display?docId=emr_na-c02497469
- https://www.supermicro.com.tw/products/system/7U/7088/SYS-7088B-TR4FT.cfm

- http://www-03.ibm.com/systems/in/resources/systems_power_hardware_e880_580x326.png
- http://www.dell.com/in/business/p/poweredge-r930/pd
- http://www.rasburytech.com/wp-content/uploads/2017/02/dl980.png
- https://www.supermicro.com/a_images/products/views/7088B-TR4FT_angle.jpg
- https://www.nextplatform.com/2015/05/04/ibm-scales-up-power8-iron-targets-in-memory/
- http://www.qdpma.com/systemarchitecture/StrategyShift.html

Acronyms

VMA	Virtual Memory Area
PFN	Page Frame Number
PGDAT	PGLIST_DATA
SMP	Symmetric Multi Processing
NUMA	Non Uniform Memory Access
HNUMA	Hierarchical NUMA
DRAM	Dynamic Random Access Memory
CDM	Coherent Device Memory
HPC	High Performance Computing
ERP	Enterprise Resource Planning
CRM	Customer Resource Management
RB	Red Black (Tree)

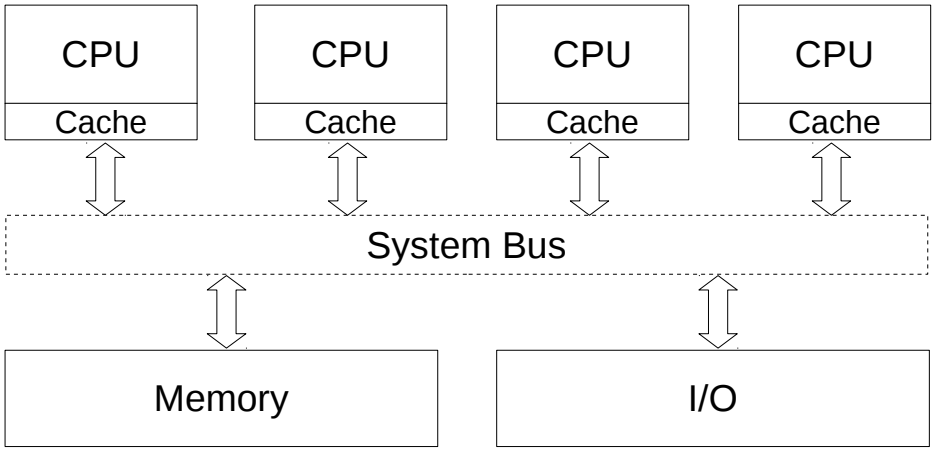
Thank You

Backup

NUMA Evolution

Symmetric Multi Processing (SMP)

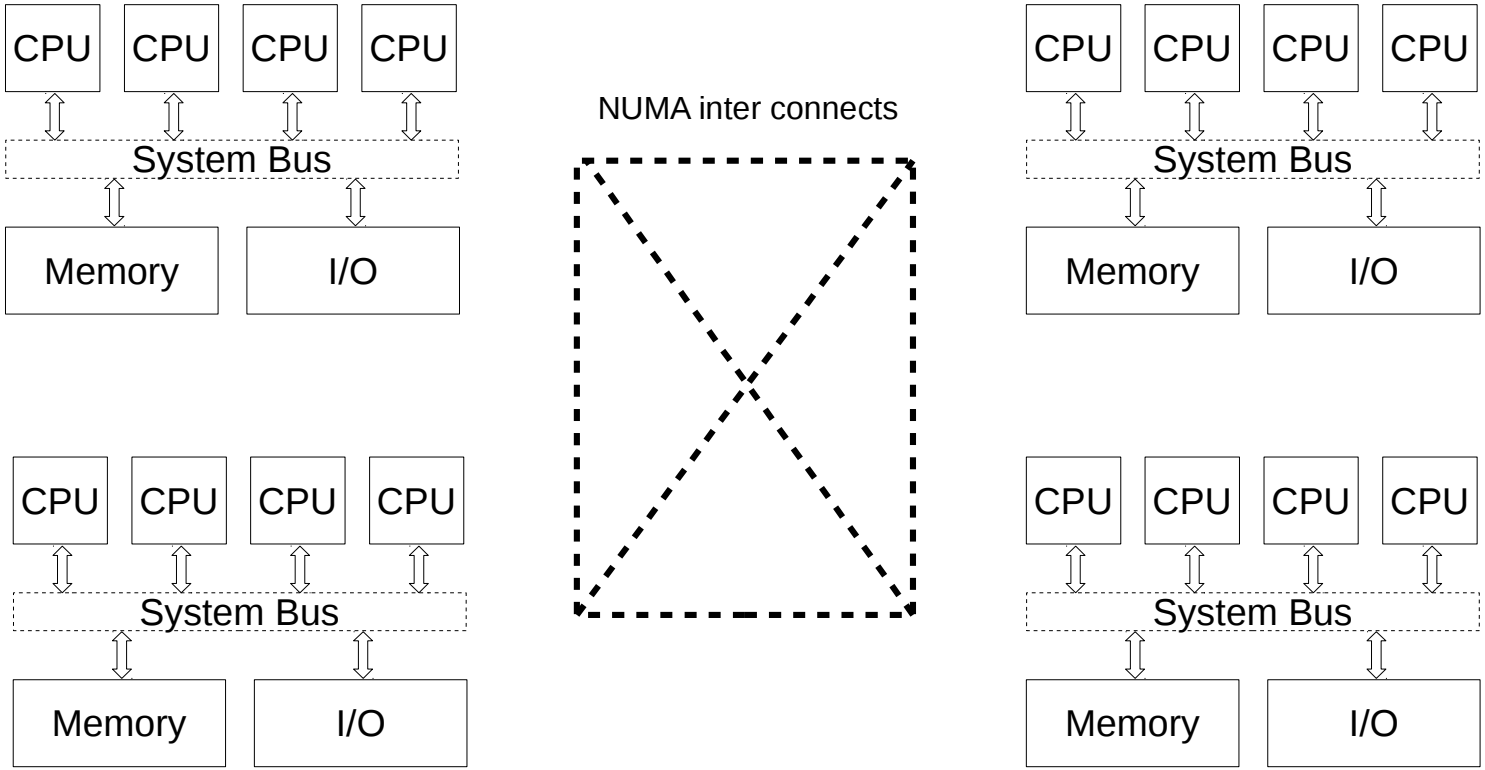
- Memory & compute intensive workloads benefit with **single OS image** on **big SMP systems**
- They utilize all cores & memory bandwidth with **shared data structures**
- They include enterprise apps like DB, CRM and HPC
- There has always been a need for bigger SMP systems for these kind of applications
- Building bigger SMP system with **Single System Bus** is complex and expensive
- **Single System Bus** design also has scalability limitations
- Then how computer system can scale up ?



Standard SMP system

Non Uniform Memory Access (NUMA)

- **NUMA** is a **design choice** to avoid further **complicating** (expensive) **Single System Bus** design
- **Single System Bus** SMP design always had **scalability problems**
- **NUMA** is a **logical follow up** from Single Bus SMP design to accommodate more cores and memory
- **NUMA** has **more than one system bus** connected through high speed **NUMA inter-connects**



Existing NUMA Data Structures

Memblock

- CONFIG_HAVE_MEMBLOCK_NODE_MAP decides if memblock has **node information**
- Memblocks are **basic blocks of memory regions** received from platform firmware
- memblock_add() adds a memory region into the system
- memblock_add_node() or memblock_set_node() adds **node information** into the memblock
- From memblock, NUMA information flows up the stack in the kernel

```
struct memblock_region {
    phys_addr_t base;
    phys_addr_t size;
    unsigned long flags;
#ifdef CONFIG_HAVE_MEMBLOCK_NODE_MAP
    int nid;
#endif
};
```

Memory Section

- Contains mem_map information along with pageblock flags
- mem_section→section_mem_map contains memmap along with pfn
- mem_section→pageblock_flags contain flags for pageblock_nr_pages blocks
- Pageblock flag information is stored in zone→pageblock_flags (CONFIG_SPARSEMEM)

```
struct mem_section {
    /*
     * This is, logically, a pointer to an array of struct
     * pages.  However, it is stored with some other magic.
     * (see sparse.c::sparse_init_one_section())
     *
     * Additionally during early boot we encode node id of
     * the location of the section here to guide allocation.
     * (see sparse.c::memory_present())
     *
     * Making it a UL at least makes someone do a cast
     * before using it wrong.
     */
    unsigned long section_mem_map;

    /* See declaration of similar field in struct zone */
    unsigned long *pageblock_flags;
#ifdef CONFIG_PAGE_EXTENSION
    /*
     * If SPARSEMEM, pgdat doesn't have page_ext pointer. We use
     * section. (see page_ext.h about this.)
     */
    struct page_ext *page_ext;
    unsigned long pad;
#endif
    /*
     * WARNING: mem_section must be a power-of-2 in size for the
     * calculation and use of SECTION_ROOT_MASK to make sense.
     */
};
```

Page

- page→flags contains NUMA information as well
- page_to_pfn() and pfn_to_page() mapping has to be fast (used in hot paths)
- If NODE_NOT_IN_PAGE_FLAGS is defined, page flags will not have NUMA information
- Instead it will be fetched from mem_section based table (which is always initialized)
- But page flags based page → nid direct mapping is always faster

```
#ifdef NODE_NOT_IN_PAGE_FLAGS
extern int page_to_nid(const struct page *page);
#else
static inline int page_to_nid(const struct page *page)
{
    return (page->flags >> NODES_PGSHIFT) & NODES_MASK;
}
#endif
```

```
int page_to_nid(const struct page *page)
{
    return section_to_node_table[page_to_section(page)];
}
EXPORT_SYMBOL(page_to_nid);
```

Pglist_data

- NUMA organization of memory starts at **struct pglist_data** (every NUMA node as got one)
- Always fetched by NODE_DATA(nid) from the core MM
- Contains all zones **struct zone node_zones[MAX_NR_ZONES]**
- Contains both the zonelists **struct zonelist node_zonelists[MAX_ZONELISTS]**
- On POWER
  - struct pglist_data *node_data[MAX_NUMNODES] → NODE_DATA(node)
  - initmem_init() → setup_node_data()
- On X86
  - struct pglist_data *node_data[MAX_NUMNODES] → NODE_DATA(node)
  - numa_register_memblks() → alloc_node_data()

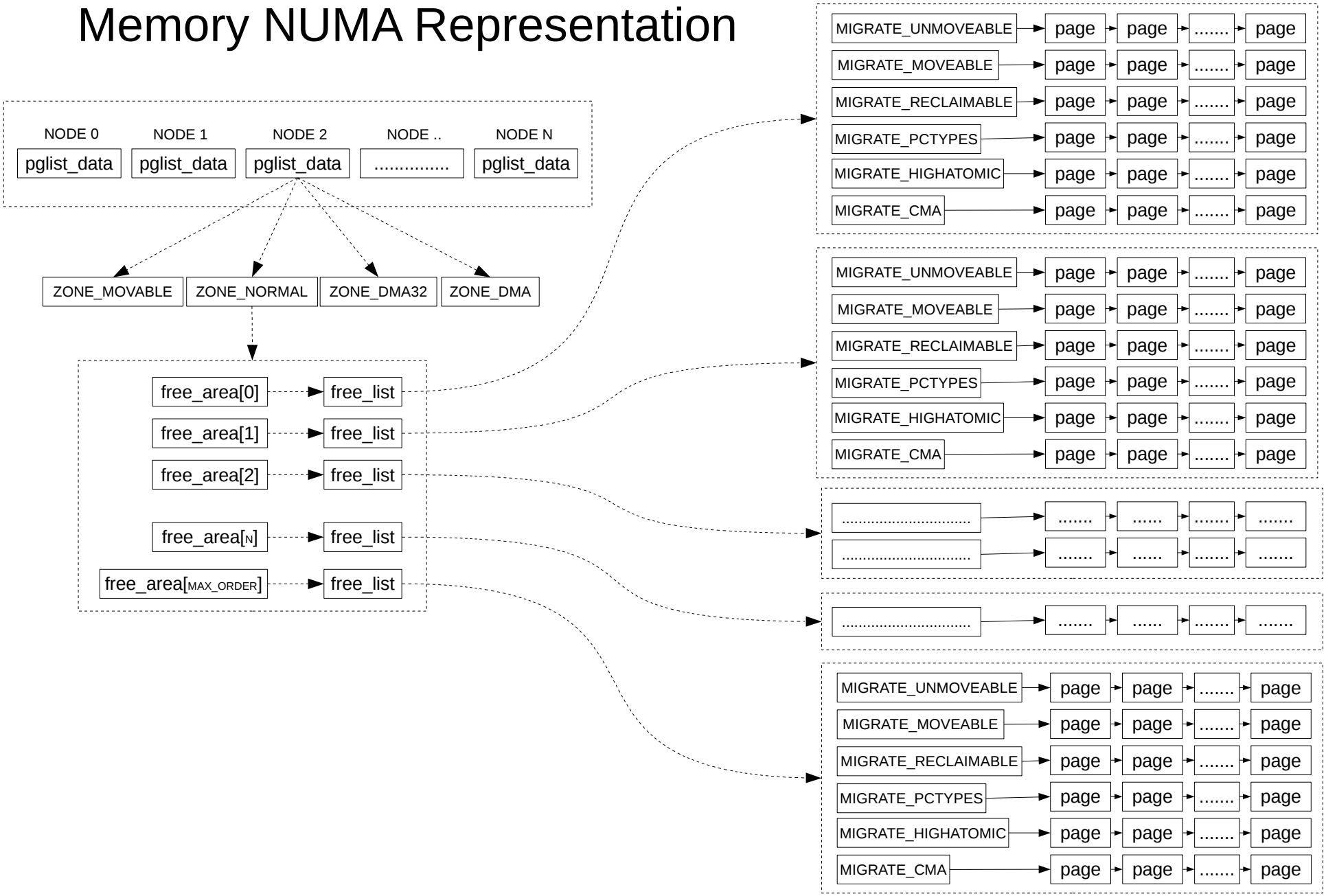
Zonelist

- Zonelists are used during memory allocation
- Helps in navigating NUMA for any allocation, should happen either for a node strictly or with fall backs
- Part of struct pglist_data (pgdat→node_zonelists[MAX_ZONELISTS])
- ZONELIST_FALLBACK contains zones of all nodes with a fallback order
- ZONELIST_NOFALLBACK contains zones of the same node, accessed only through __GFP_THISNODE

```
/*
 * This struct contains information about a zone in a zonelist. It is stored
 * here to avoid dereferences into large structures and lookups of tables
 */
struct zoneref {
    struct zone *zone;  /* Pointer to actual zone */
    int zone_idx;       /* zone_idx(zoneref->zone) */
};

/*
 * One allocation request operates on a zonelist. A zonelist
 * is a list of zones, the first one is the 'goal' of the
 * allocation, the other zones are fallback zones, in decreasing
 * priority.
 *
 * To speed the reading of the zonelist, the zonerefs contain the zone index
 * of the entry being read. Helper functions to access information given
 * a struct zoneref are
 *
 * zonelist_zone()  - Return the struct zone * for an entry in _zonerefs
 * zonelist_zone_idx()  - Return the index of the zone for an entry
 * zonelist_node_idx()  - Return the index of the node for an entry
 */
struct zonelist {
    struct zoneref _zonerefs[MAX_ZONES_PER_ZONELIST + 1];
};
```

Memory NUMA Representation



Sample Commercial NUMA Systems

IBM POWER8 E880

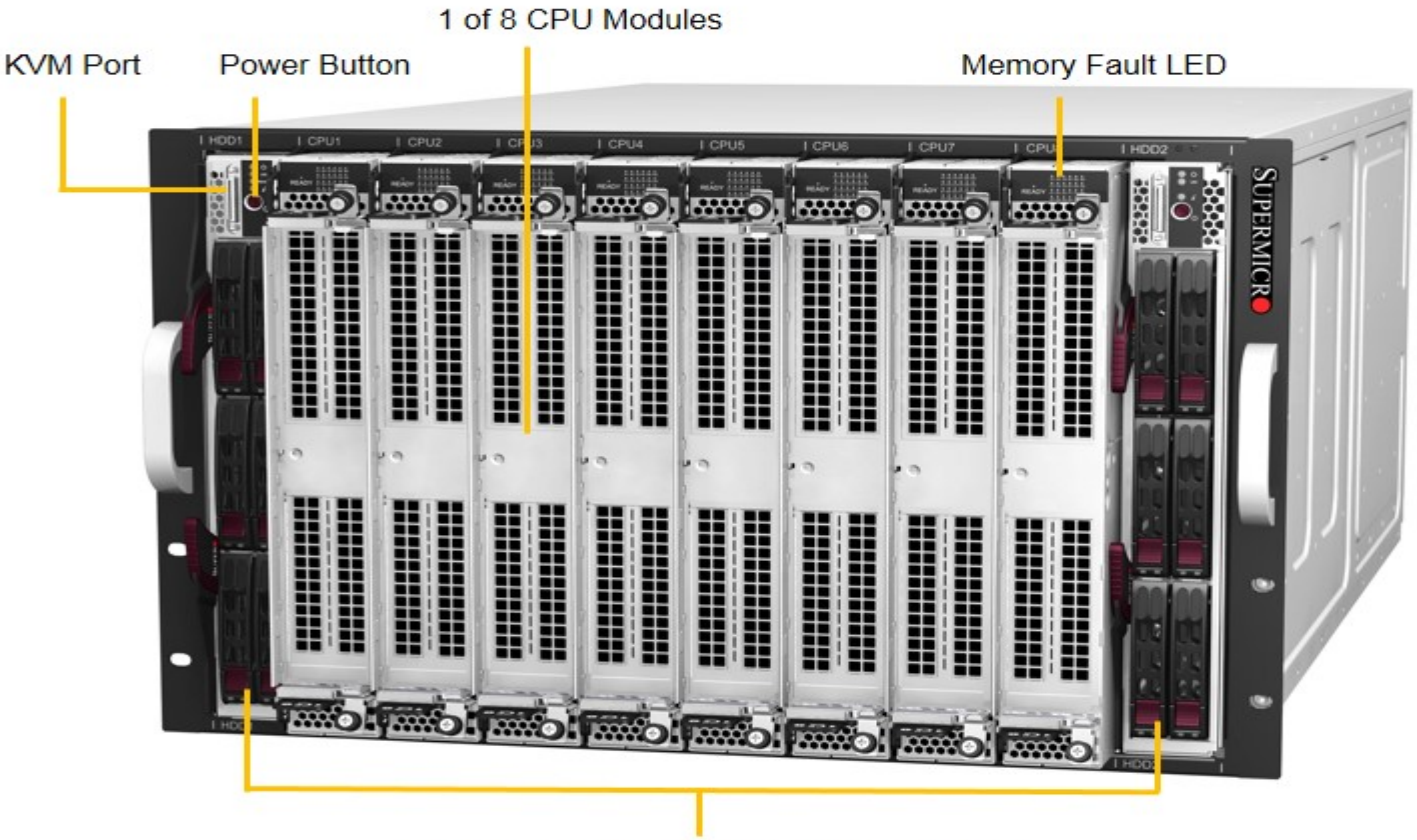
DELL Power Edge R930

HP DL980

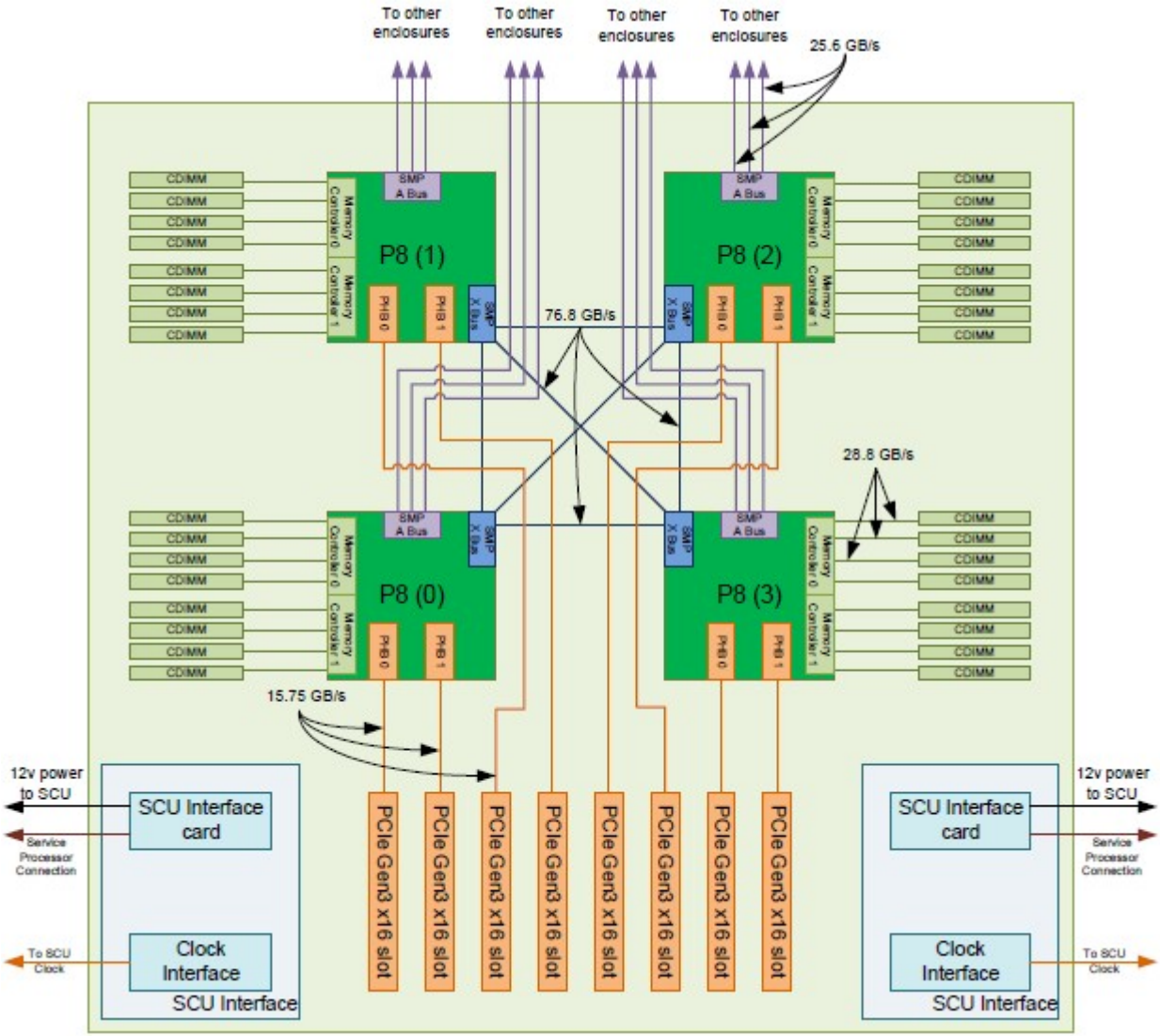
Supermicro 7088B-TR4FT

SuperServer SYS-7088B-TR4FT

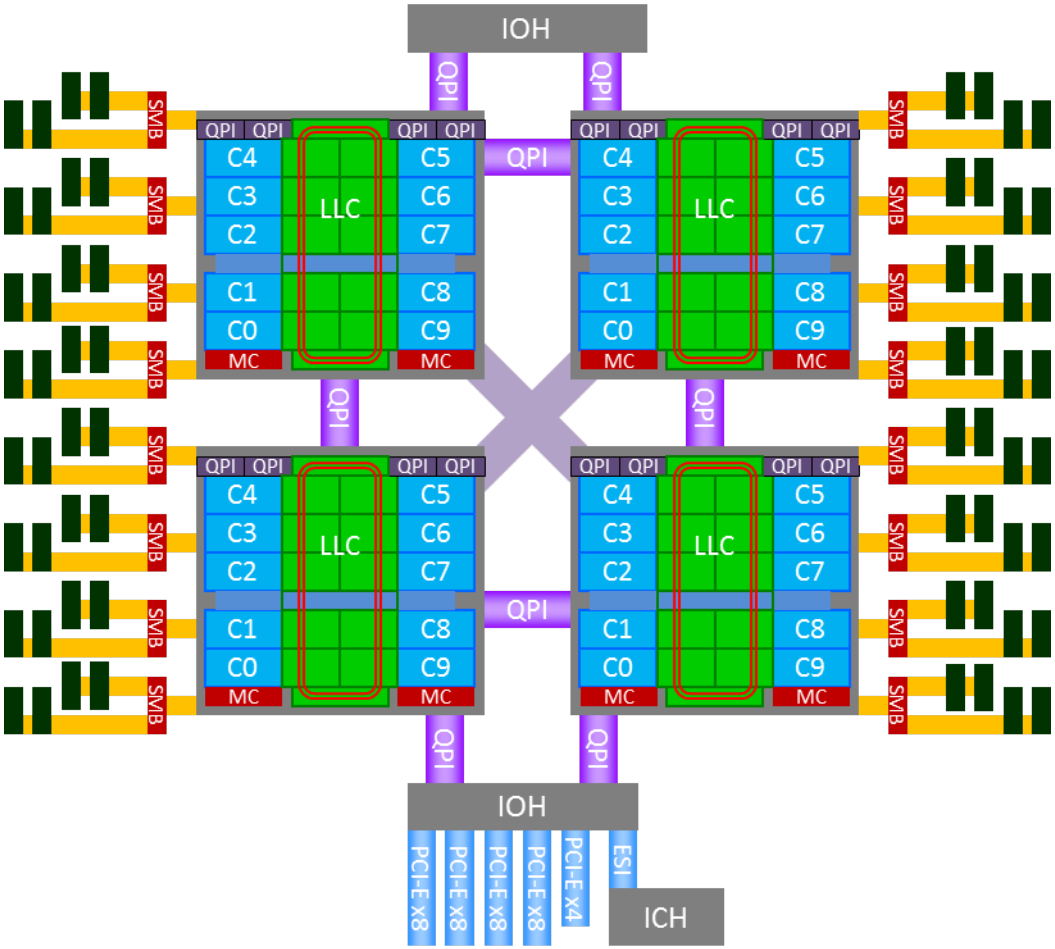
(Angled View – System)



IBM POWER8 NUMA Diagram



Intel Xeon E7400 NUMA Diagram



Relevant Community Discussions

NUMA CDM

- https://lkml.org/lkml/2016/10/24/19 (RFC V1)
- https://lkml.org/lkml/2016/11/22/339 (RFC V1.2)
- https://lkml.org/lkml/2017/1/29/198 (RFC V2)
- https://lkml.org/lkml/2017/2/8/329 (PATCH V1)
- https://lwn.net/Articles/720380/ (RFC from Balbir Singh)

LSFMM 2017

- https://lwn.net/Articles/717601/ (HMM and CDM)

HMM CDM

- https://lkml.org/lkml/2017/4/7/638 (RFC V1)
- https://lwn.net/Articles/725412/ (PATCH V2)
- https://lwn.net/Articles/727114/ (PATCH V3)
- https://lwn.net/Articles/727692/ (PATCH V4)

HMM

- https://lwn.net/Articles/726691/ (PATCH V24)
- https://lwn.net/Articles/731259/ (PATCH V25)