DRAMmalloc(): API and Design Documents for a Shared Global Memory Manager

Yuqing (Ivy) Wang, Ahsan Parvaiz, Swann Perarnau, and Andrew A Chien

July 2025

Collected design documents for the DRAMmalloc() library, and an application interface to the shared global memory on the UpDown System. The API exposes control of memory layout across the machine for dynamic allocation. This allows applications to have efficient control of data layout – for high performance, without changing the program structure (no change to virtual addressing).

More information can be found in:

Y. Wang, S. Perarnau and A. A. Chien, "UpDown: Combining Scalable Address Translation with Locality Control," *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Atlanta, GA, USA, 2024, pp. 1014-1024, doi: 10.1109/SCW63240.2024.00141.

DRAMalloc v1.0

Yuqing Wang, Andrew A Chien
November 11, 2024

Introduction

DRAMalloc is the application-level UpDown library responsible for managing and allocating the global shared segments to UpDown programs. Figure 1 shows the view of UpDown's distributed node memory. Each UpDown node creates a linear physical address space for its DRAM and is partitioned into two parts 1) the private memory can only be accessed by the top CPU cores and UpDown lanes on the same node, and 2) the global memory can be accessed by all the UpDown lanes in the machine provided the physical address. The global segment manager is responsible for allocating segment-based memory to each application, which will be managed internally by the DRAMalloc library in each application.

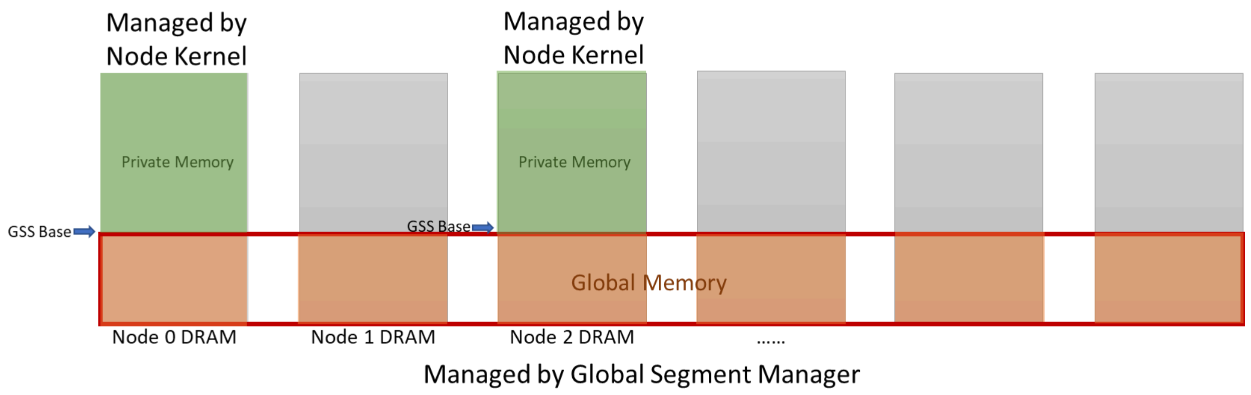


FIGURE 1. Node DRAM is partitioned into private and global memory. Each node's Global Shared Segment (GSS) base points to the start of the Global memory partition.

DRAMalloc allocates segment memory in the distributed global memory. The segment is divided into blocks and distributed across a number of node DRAM memories. At the allocation time, users can specify the size of the block and the number of blocks the segment spans to control the data layout.

Interface

DRAMalloc consists of two interfaces: the top interfaces, which initiate the allocator, make allocations and translate UpDown addresses, and the UpDown interfaces, which allocate/free memories for UpDown programs and terminate the allocator.

Constructor

```C/C++
DramAllocator(UpDown::UDRuntime_t *runtime, uint64_t eventlabel);
```

Constructs a *DramAllocator* object.

Parameters

- **runtime** A pointer to the UDRuntime_t object.
- **eventLabel** The event label from the linker output header file ended with *DRAMalloc__global_broadcast*.

Return Null

```C/C++
DramAllocator(UpDown::UDRuntime_t *runtime, uint64_t eventlabel, uint64_t
defaultBlockSize);
```

Constructs a *DramAllocator* object. **Note this constructor is for GEM5 only.**

Parameters

- **runtime** A pointer to the UDRuntime_t object.
- **eventLabel** The event label from the linker output header file ended with *DRAMalloc__global_broadcast*.
- **defaultBlockSize** The default block size for all allocations. This **must** be equal to the GEM5 parameter *gseg-block-size*, otherwise the simulator will produce the wrong address when accessing the global memory from the UpDown.

Return Null

Top

Below are functions that can be called via the *DramAllocator* object on the top program.

```C/C++
void run();
```

Runs the allocator to handle calls from UpDown. Optional if only to allocate memory from the top.

```C/C++
void *mm_malloc_global(size_t size, size_t blockSize, uint64_t numNodes,
                      uint64_t startNode);
```

Allocates memory in global virtual memory.

Parameters

- **size** The size of the memory to allocate (in Bytes).
- **blockSize** The size of the block for DRAM allocations (in Bytes). Note that this number should be a power of 2.
- **numNodes** The number of nodes that the segment spans. Note that this number should be a power of 2.
- **startNode** The starting node ID where the segment to be allocated is located.

Return A pointer to the allocated memory. Null if the allocation fails.

UpDown

Below are the events that can be called by the UpDown program.

```
None
event DRAMalloc::dramalloc(long requestType, long continuation, long size, long
                           blockSize, long numNodes, long startNodes)
```

Allocate global memory from the UpDown.

Parameters

- **requestType** Set to 1 for allocation.
- **continuation** User continuation event. Activate when the allocation finishes.
- **size** The size of the memory to allocate (in Bytes).
- **blockSize** The size of the block for DRAM allocations (in Bytes). Note that this number should be a power of 2.
- **numNodes** The number of nodes that the segment spans. Note that this number should be a power of 2.
- **startNode** The starting node ID where the segment to be allocated is located.

Return A pointer to the allocated memory. Null if the allocation fails.

```
None
event DRAMalloc::dramalloc(long requestType, long continuation, long *ptr)
```

Free a global memory allocation.

Parameters

- **requestType** Set to 2 for free.
- **continuation** User continuation event. Activate when the free finishes.
- **ptr** Pointer to the memory segment to be freed

Return 1 if succeeds else 0

```
None
event DRAMalloc::dramalloc(long requestType)
```

Terminate the allocator on Top.

Parameters

- **requestType** Set to 3.

Return Null

Expected behavior on different simulators

FastSim2

The physical memory allocated on fastsim2 correctly reflects the block-cyclic pattern specified by the call parameters. That is, the allocated physical memory has size `size` and spans node memory for nodes [`startNode`, `startNode`+`numNodes`) in a block-cyclic manner with a block size equal to `blockSize` in the machine.

GEM5

The physical memory allocated on GEM5 **does not** reflect the block-cyclic pattern specified by the call parameters, because the GEM5 sets up the VA-to-PA address mapping statically. That means the physical memory allocated on GEM5 for every DRAMalloc call always spans all the nodes in the machine with block size equal to the *gseg-block-size* of the GEM5 parameter regardless of the call parameter.

FastSim1

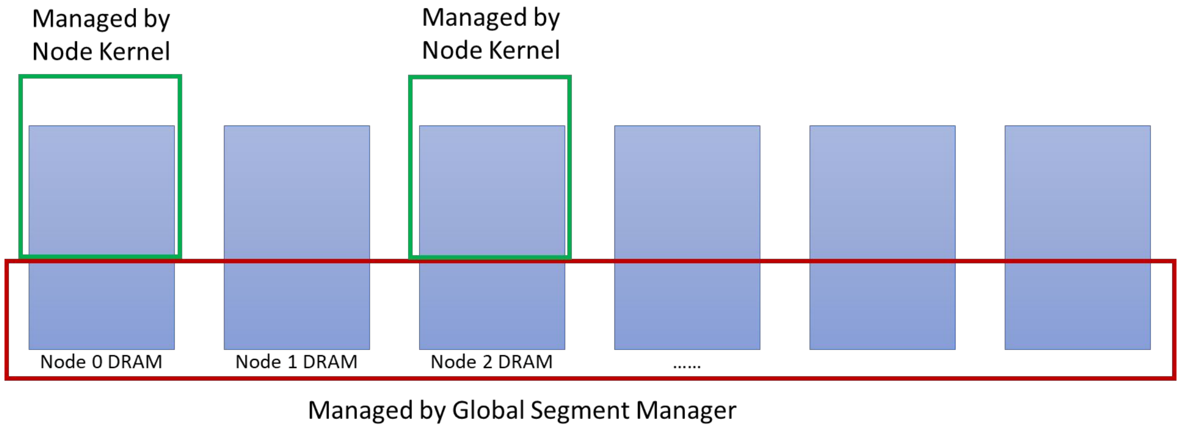
The physical memory allocated on gem5 **does not** reflect the block-cyclic pattern specified by the call parameters, because the simulator does not model the distributed physical memory for UpDown. Instead, all the nodes on fastsim1 share a single contiguous address space. Therefore, the memory allocated by DRAMalloc has the same layout as the *mmalloc* function provided by the simulator.

DRAMalloc

November 11, 2024
Ivy Wang

UpDown Memory Model

- DRAM is partitioned into two parts:
  - private memory (managed by the Node OS)
  - global DRAM (managed by the Global Segment Manager).
- Global segment manager (GSM)
  - trusted authority managing the distributed global memory and allocating the global segment to processes.

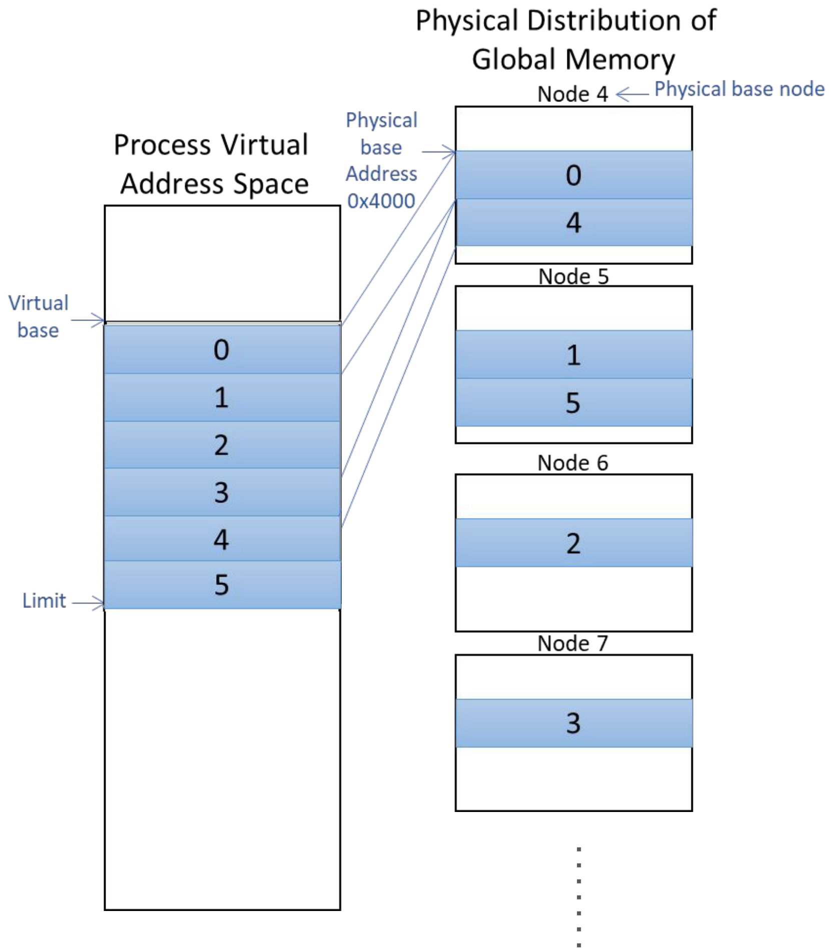


Swizzle Translation

UpDown deploys a compressed translation mechanism, called swizzle translation, for VA-PA translation.

Swizzle mask describe the block-cyclic distribution of the memory segment across the node DRAMs.

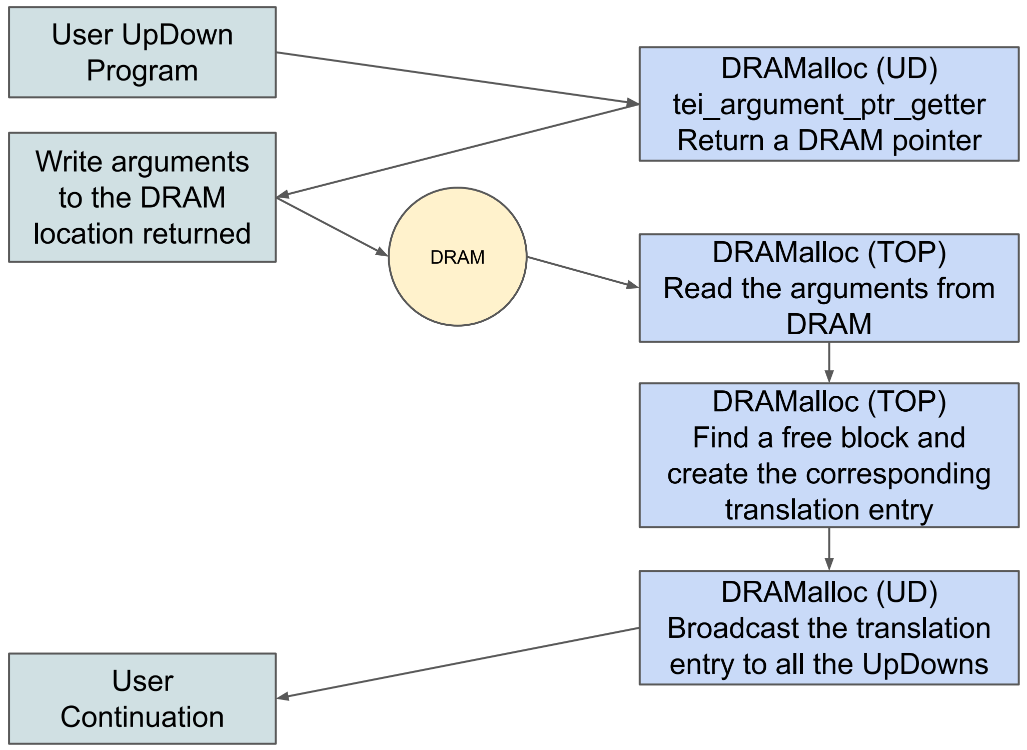
- Specify the block size and number of nodes to stride across.


Physical Distribution of Global Memory
Node 4
Physical base node
Physical base Address 0x4000
Process Virtual Address Space
Virtual base
Limit
0
1
2
3
4
5
Node 5
Node 6
Node 7

DRAMalloc

- A memory allocation library allocating the global shared memory within a process.
- The goal of DRAMalloc include:
    - Simulate the functionality of global segment manager on our simulation frameworks (gem5, fastsim, ASST)
    - Enable UpDown to allocate memory independently
    - Allocated memory
- Files are under libraries/dramalloc branch perarnau/dramalloc

High-level View


User UpDown Program
DRAMalloc (UD)
tei_argument_ptr_getter
Return a DRAM pointer
Write arguments to the DRAM location returned
DRAM
DRAMalloc (TOP)
Read the arguments from DRAM
DRAMalloc (TOP)
Find a free block and create the corresponding translation entry
DRAMalloc (UD)
Broadcast the translation entry to all the UpDowns
User Continuation

Usage

1. Initialize the allocator on Top
   a. `new dramalloc::DramAllocator(rt, machine /*machineConfig*/, event_label /*event label for the translation installation event */);`
2. Calling DRAMalloc within the UpDown program
   a. Get the DRAM location to write the call argument
      i. `long evw = evw_new(DRAMalloc_LANE0, tei_argument_ptr_getter);`
      ii. `send_event(evw, dumb_op, cont);`
   b. Write the argument to call DRAMalloc
      i. `send_dram_write(parameter_ptr, parameters, 7, continuation);`
3. DRAMalloc return
   a. OB_0 is a pointer to the allocated memory

DRAMalloc call arguments

1. Request type
   a. 1 = allocate, 2 = free
2. Entry installer nwid
3. Return event
   a. User-defined continuation event
4. Block size
   a. Block size for the allocation (ref block-cyclic)
5. Segment size
   a. Size of the requested allocation (in Bytes)
6. Number of nodes
   a. Number of nodes the allocation spans
7. Start node
   a. Base node id where the allocation starts

Note

- DRAMalloc is single threaded right now
  - Only 1 allocation request can be served at a time
- DRAMalloc is
- Gem5 only support 1 block size and need to be specified in the gem5 run command
  - `--gseg-block-size`
- Fastsim doesn’t have a accurate model of the DRAM
  - No distinction between local and global
  - Physical addresses of simulated program do not follow the per-node DRAM design of the system memory

DRAMalloc()

(see slides in
https://docs.google.com/presentation/d/1JltTCRdaZBGkWJEeJVYJv0_7Uh1cPPYU/edit#slide=id.g137760bca89_0_0)

Purpose:

- Manage all Global shared DRAM in the machine
- Provide application with a Node DRAM Pool, and Allnode DRAM Pool [no control over distribution]
- Provide application ability to allocate distributed DRAM, controlling start, chunk, striping, nr_nodes (see slides for detailed restrictions)

Interface:

DRAMalloc(long size, long blocksize=64, long nrnodes=allnodes, long startnode=0)

-> Pointer to the DRAM in virtual address space, which is start of allocation of size bytes

-> if NRNODES=1, see #1

-> if BLOCKSIZE=0, and NRNODES != 1, see #2

-> Else, see #3

Standard Translations:

- #1 Each node already has a translation that maps its node private address space, included in this is the heap
    - DRAMalloc that specifies NRNODES=1 should be allocated on this heapl
    - !!! there is no way to control how this segment maps across the HBM stacks !!!
- #2 Each application has EXACTLY one translation that spans all of the application's nodes (Allnode DRAM pool)
    - DRAMalloc that specifies BLOCKSIZE=0 should be allocated on the Allnode DRAM pool
- Idea: these two types of segments can be managed with a traditional malloc() threadsafe of course

- #3 Create a new translation that supports the requested DRAMalloc() parameters
    - Make the translation visible across the application
    - Return the pointer to the address of the start of the allocation to the caller
    - Note: we expect these calls to be rare (<10 times in an application)
        - Need not use DRAM efficiently (packing)
        - Need not respond quickly (latency)
-

Global Data Structures on UpDown

Andrew A Chien

All-team, 2/9/2023

THE UNIVERSITY OF CHICAGO | UpDown Project

Programmer's View

- Idea: Create data structures that are spread over the machine
  - Exploit the large global dram capacity
  - Exploit the large global dram bandwidth

- Enable really large graphs
- Enable really high parallelism

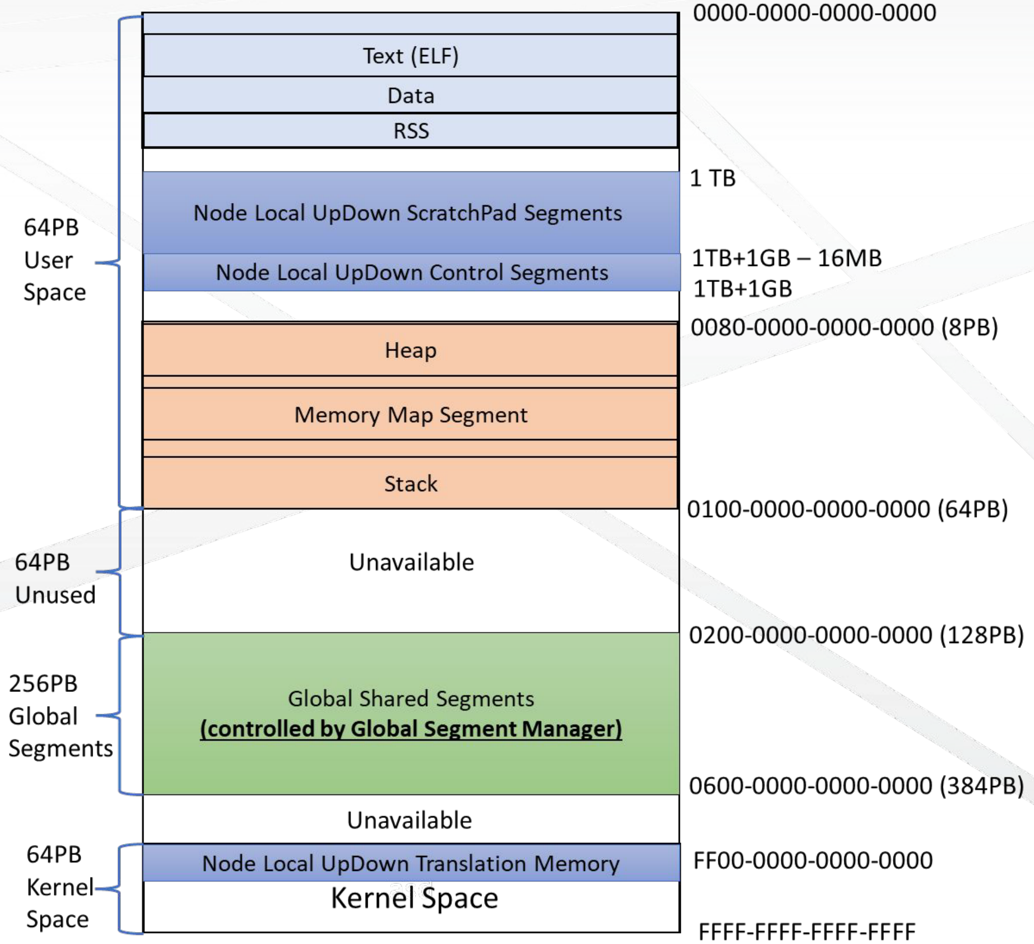
- [Inspired by block, cyclic but much simpler]

Allocating Global DRAM

- DRAMmalloc(long size, long blocksize=64, long nrnodes=allnodes, long startnode=0)
  - Size: allocation request size in bytes
  - Blocksize: size of blocks that will be interleaved, must be an integral power of 2, and at least 64; size in bytes
  - Nrnodes: number of nodes that the allocation will be interleaved across, it will be a contiguous set of nodes [startnode:startnode+nrnodes-1]
  - Startnode: the start node for the memory allocation

- DRAMmalloc(1073741824,1048576,allnodes,0) // 1GB in 1MB chunks
- DRAMmalloc(2^40,2^30,allnodes,0) // 1TB in 1GB chunks
- DRAMmalloc(2^50,2^36,allnodes,0) // 1PB in 64GB chunks

How it looks in the Virtual Address space

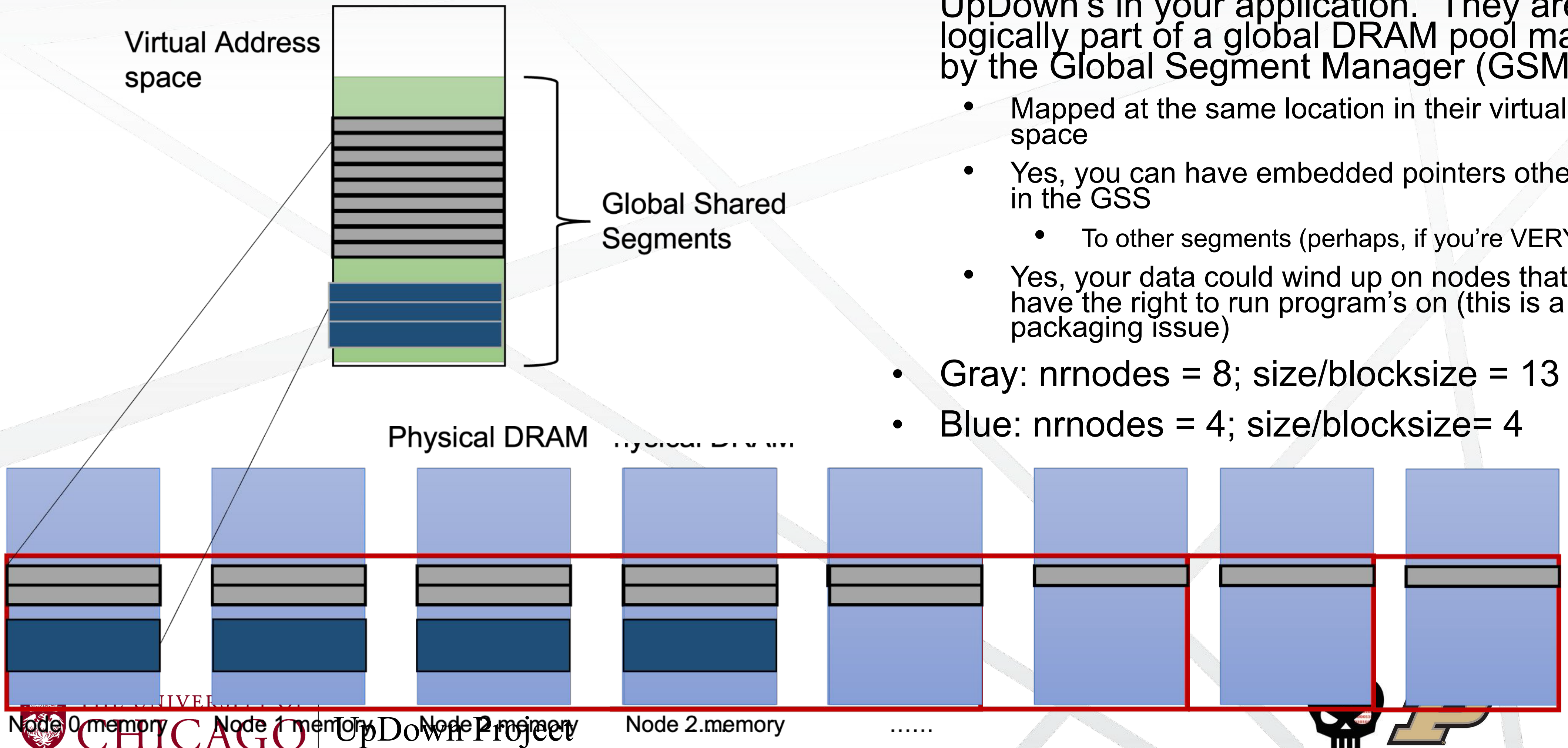


0000-0000-0000-0000
Text (ELF)
Data
RSS
1 TB
Node Local UpDown ScratchPad Segments
Node Local UpDown Control Segments
1TB+1GB – 16MB
1TB+1GB
0080-0000-0000-0000 (8PB)
64PB User Space
Heap
Memory Map Segment
Stack
0100-0000-0000-0000 (64PB)
64PB Unused
Unavailable
0200-0000-0000-0000 (128PB)
256PB Global Segments
Global Shared Segments
(controlled by Global Segment Manager)
0600-0000-0000-0000 (384PB)
Unavailable
64PB Kernel Space
Node Local UpDown Translation Memory
FF00-0000-0000-0000
Kernel Space
FFFF-FFFF-FFFF-FFFF

- Each Node's Top and UpDown have the same VA view
- Global shared segments look like a linear segment
- Green segment is what we're talking about today
  - => spread across the nodes in the machine
- Capabilities are >= what GPU folks call UVM
- Other segments are local
  - See "UpDown Process Isolation v0.9"

THE UNIVERSITY OF CHICAGO | UpDown Project

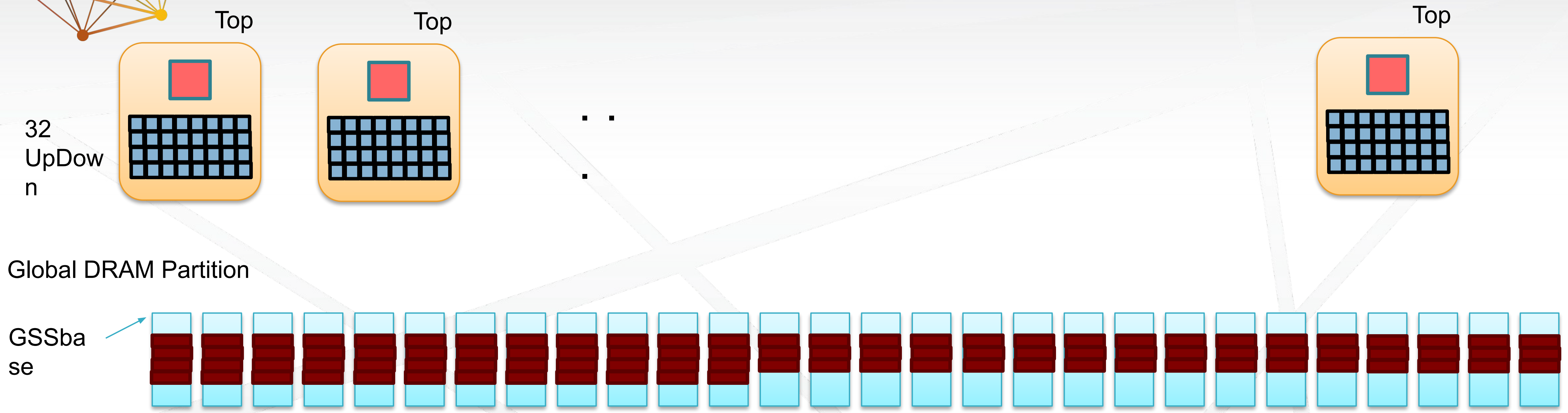
How DRAMmalloc() lays the data out across the Machine



- The globally shared segments are visible to all UpDown's in your application. They are logically part of a global DRAM pool managed by the Global Segment Manager (GSM).
  - Mapped at the same location in their virtual address space
  - Yes, you can have embedded pointers other objects in the GSS
    - To other segments (perhaps, if you're VERY careful)
  - Yes, your data could wind up on nodes that you don't have the right to run program's on (this is a hardware packaging issue)
- Gray: nrnodes = 8; size/blocksize = 13
- Blue: nrnodes = 4; size/blocksize= 4

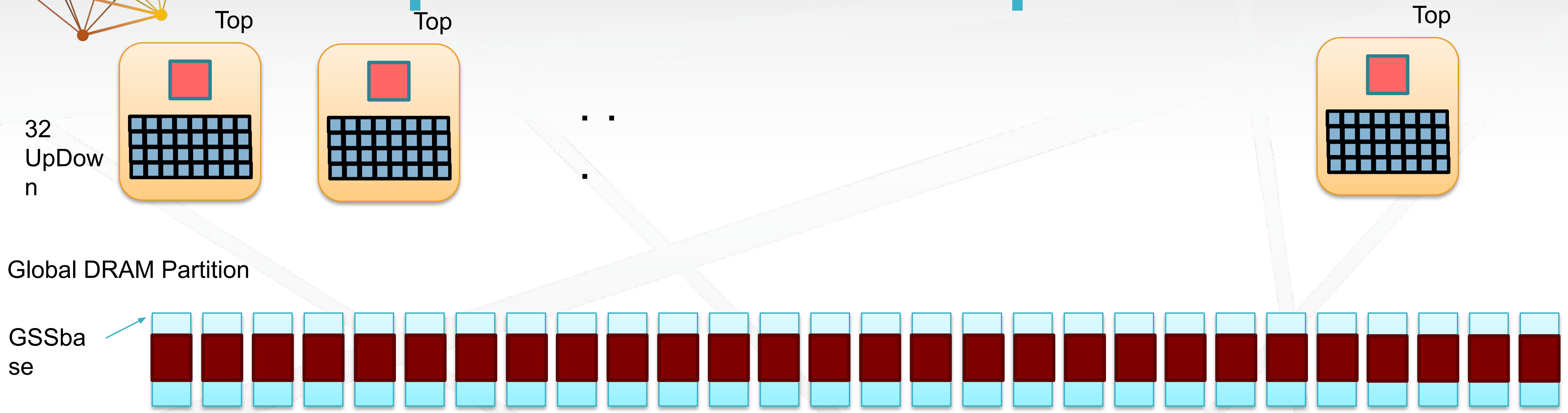
Global shared segments

Example 1: Global capacity and Bandwidth



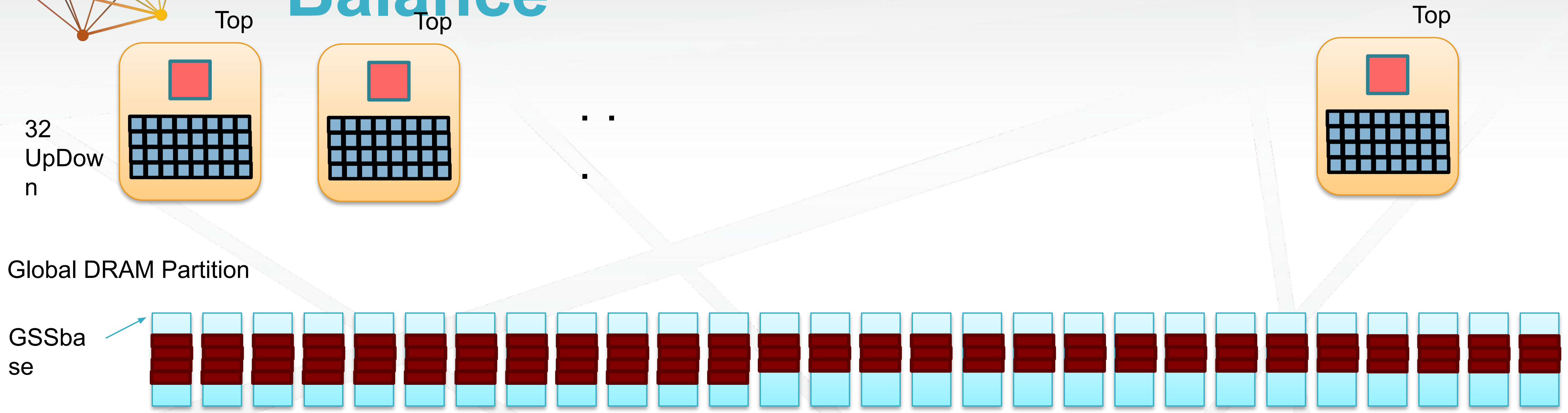
- Logical View of UpDown system and Global DRAM partition
- DRAMmalloc(2^40,64,allnodes,0) // 1TB in 64B blocks

Example 2: one Block per Node



- DRAMmalloc(2^40,2^30,2^10,0) // 1TB in 1GB blocks
- Could stop part way across

Example 3: Blocked cyclic for Load Balance



- Same as example #1, but can increase the block size if useful
- Logical View of UpDown system and Global DRAM partition
- DRAMmalloc(2^40,2^20,allnodes,0) // 1TB in 1MB blocks
  - If 1K nodes, 1024 times wraparound (1K blocks/node)
  - If 16K nodes, 64 times wraparound (64 blocks/node)

FAQ

- UpDown Translation hardware does the magic to find the right nodes (swizzling)
  - Top TLB hardware will need to be enhanced
- No internal node memory structure is exposed (each node has a linear DRAM address space, that is spread across the stacks… global addressing doesn't dictate any internal structure)
  - See sapphire rapids, gpu's
- How much should I worry about this layout? Not much! (allocate for capacity and parallelism)

Discussion

- What applications need this?
- What was unclear?
- Others?
- We should build a quick implementation to enable apps (1 week), and a smart allocator as a research project!