A Framework for Application Guidance in Virtual Memory Systems

Presented by Michael Jantz

Contributions from Carl Strickland, Kshitij Doshi, Martin Dimitrov, and Karthik Kumar

Executive Summary

- Memory has become a significant player in power and performance of server systems
- Memory power management is challenging
- Propose a collaborative approach between applications, operating system, and hardware:
  - Applications inform OS about memory usage
  - Expose hardware power-manageable domains to the OS
- Implement our framework by re-architecting a recent Linux kernel
- Evaluate by classifying memory usage in an industrial-grade Java virtual machine (Oracle/Sun's HotSpot)

Why 

- CPU and Memory are most significant players for power and performance
  - In servers, memory power == 40% of total power

- Applications can direct CPU usage
  - threads may be affinitized to individual cores or migrated b/w cores
  - prioritize threads for task deadlines (with nice)
  - individual cores may be turned off when unused

- ***Surprisingly, no such controls exist for memory***

Example Scenario

- System with database workload with 256GB DRAM
  - All memory in use, but only 2% of pages are accessed frequently
  - CPU utilization is low
- **How to reduce power consumption?**

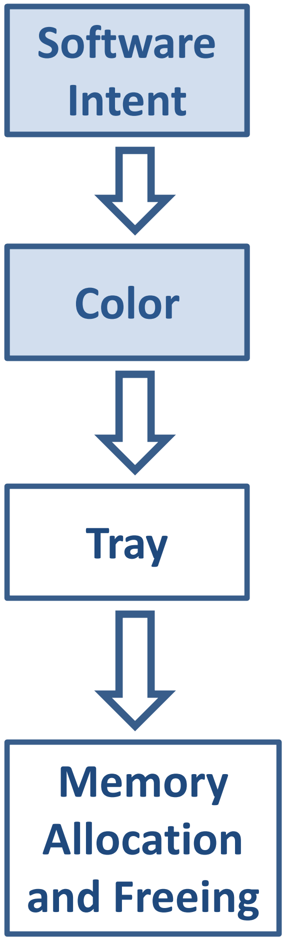
Challenges in Managing Memory Power

- At least two levels of virtualization:
  - Virtual memory abstracts away application-level info
  - Physical memory viewed as single, contiguous array of storage
- No way for agents to cooperate with the OS and with each other
- Lack of a tuning methodology

A Collaborative Approach

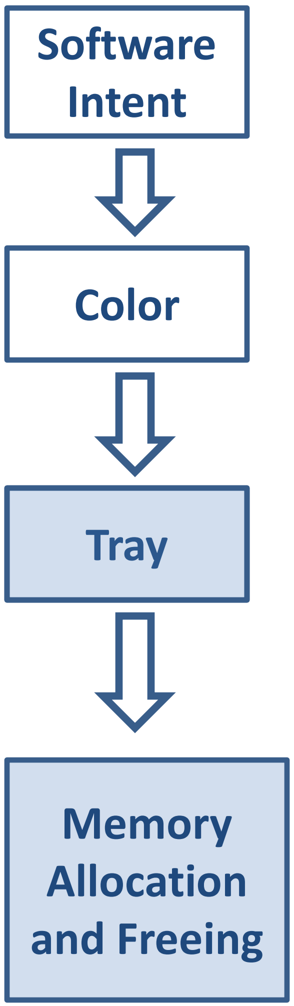
- Our approach: enable applications to guide mem. mgmt.
- Requires collaboration between the application, OS, and hardware:
  - Interface for communicating application intent to OS
  - Ability to keep track of which memory hardware units host which physical pages during memory mgmt.
- To achieve this, we propose the following abstractions:
  - Colors
  - Trays

Communicating Application Intent with Colors

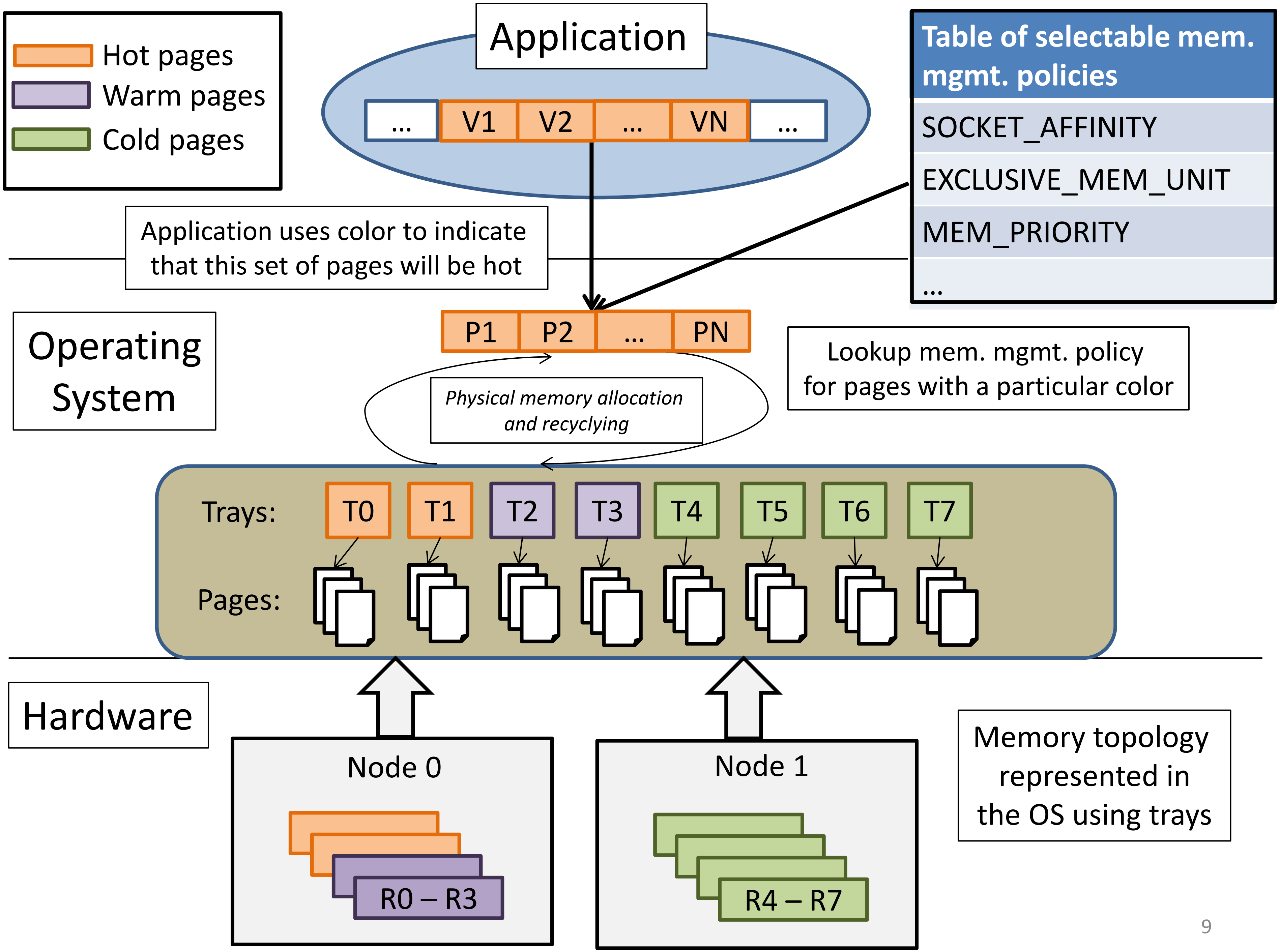


- Color = a hint for how pages will be used
  - Colors applied to sets of virtual pages that are alike
  - Attributes associated with each color
- Attributes express different types of distinctions:
  - Hot and cold pages (frequency of access)
  - Pages belonging to data structures with different usage patterns
- Allow applications to remain agnostic to lower level details of mem. mgmt.

Power-Manageable Units Represented as Trays



- Tray = software structure containing sets of pages that constitute a power-manageable unit
- Requires mapping from physical addresses to power-manageable units
  - ACPI 5.0 defines memory power state table (MPST)
- Re-architect a recent Linux Kernel to perform memory management over trays



Hot pages
Warm pages
Cold pages
Application
...
V1
V2
...
VN
...
Table of selectable mem. mgmt. policies
SOCKET_AFFINITY
EXCLUSIVE_MEM_UNIT
MEM_PRIORITY
...
Application uses color to indicate that this set of pages will be hot
Operating System
P1
P2
...
PN
Lookup mem. mgmt. policy for pages with a particular color
Physical memory allocation and recycling
Trays:
T0
T1
T2
T3
T4
T5
T6
T7
Pages:
Hardware
Node 0
R0 – R3
Node 1
R4 – R7
Memory topology represented in the OS using trays

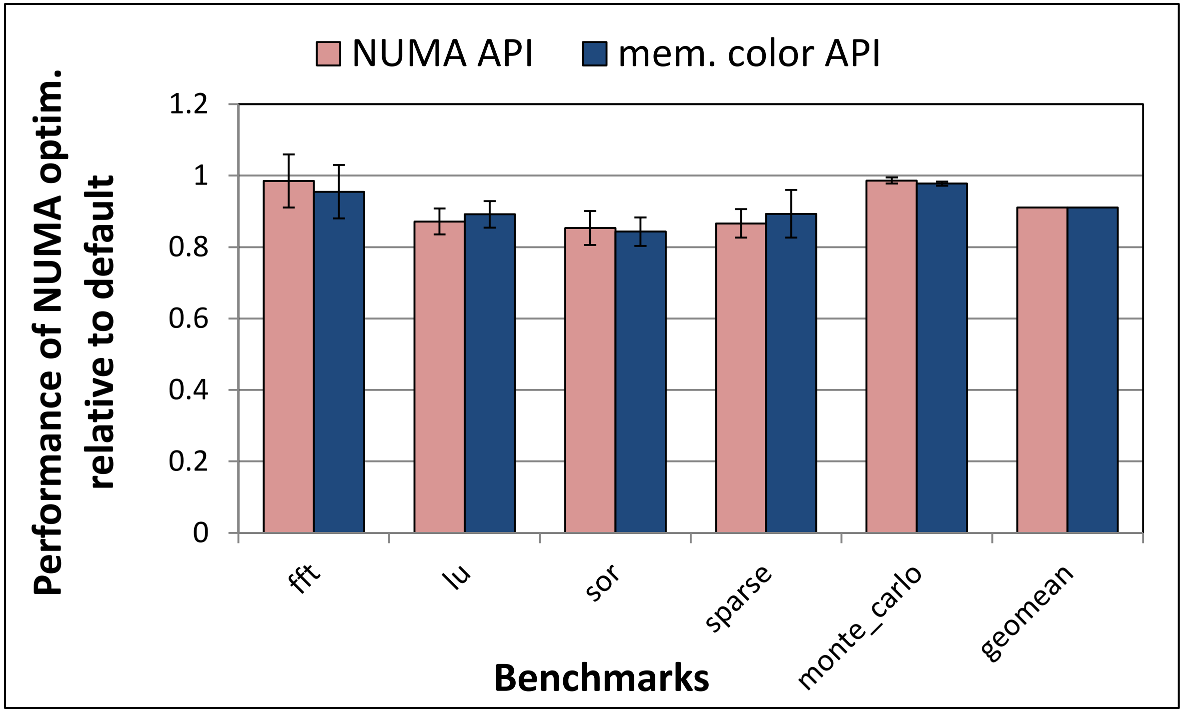
Experimental Evaluation

- Emulating NUMA API’s
- Memory prioritization
- Enabling power consumption proportional to the active footprint
- Exploiting generational garbage collection to reduce DRAM power

Emulating NUMA API's

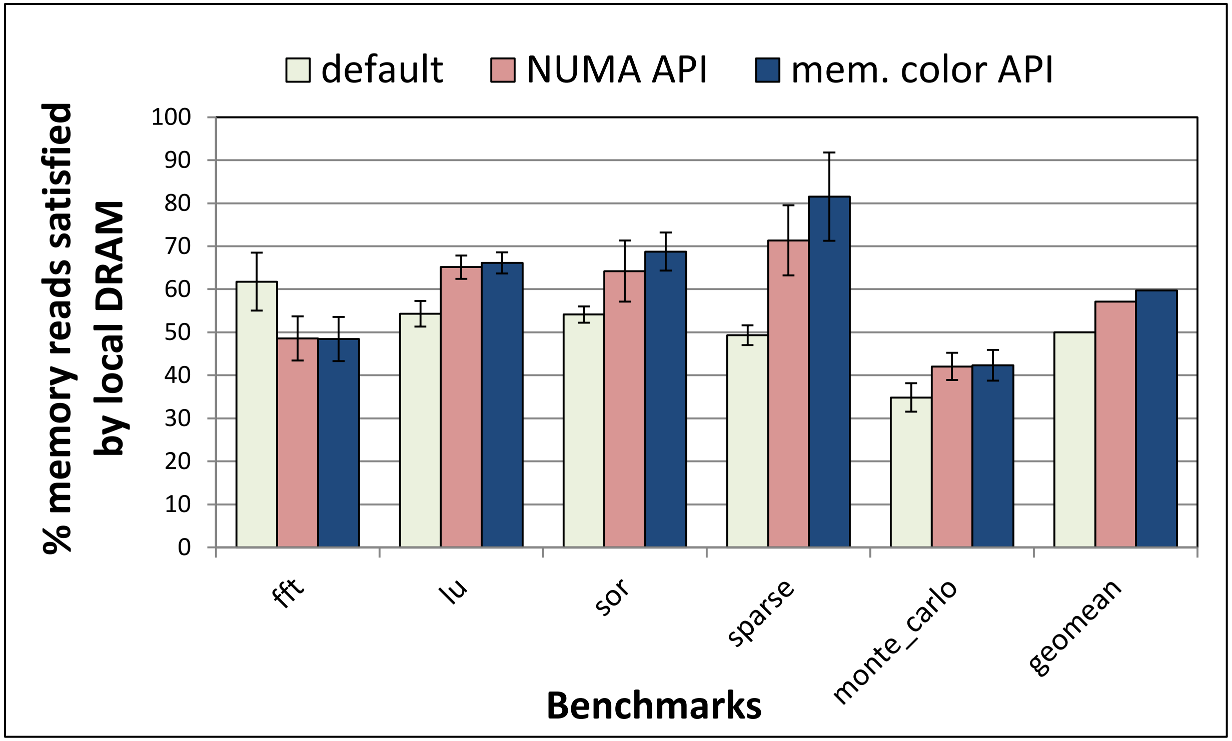
- Modern server systems enable memory mgmt. at level of NUMA node
- API's control memory placement on NUMA nodes
- Our framework is *flexible* enough to emulate NUMA API functionality
- Oracle/Sun's HotSpot JVM uses NUMA API to improve DRAM access locality for several workloads
- Modified HotSpot to control memory placement using memory coloring framework

Memory Coloring Emulates the NUMA API



- Performance of SciMark 2.0 benchmarks with “NUMA-optimized” HotSpot implemented with (1) NUMA API’s and (2) memory coloring framework
- Performance is similar for both implementations

Memory Coloring Emulates the NUMA API



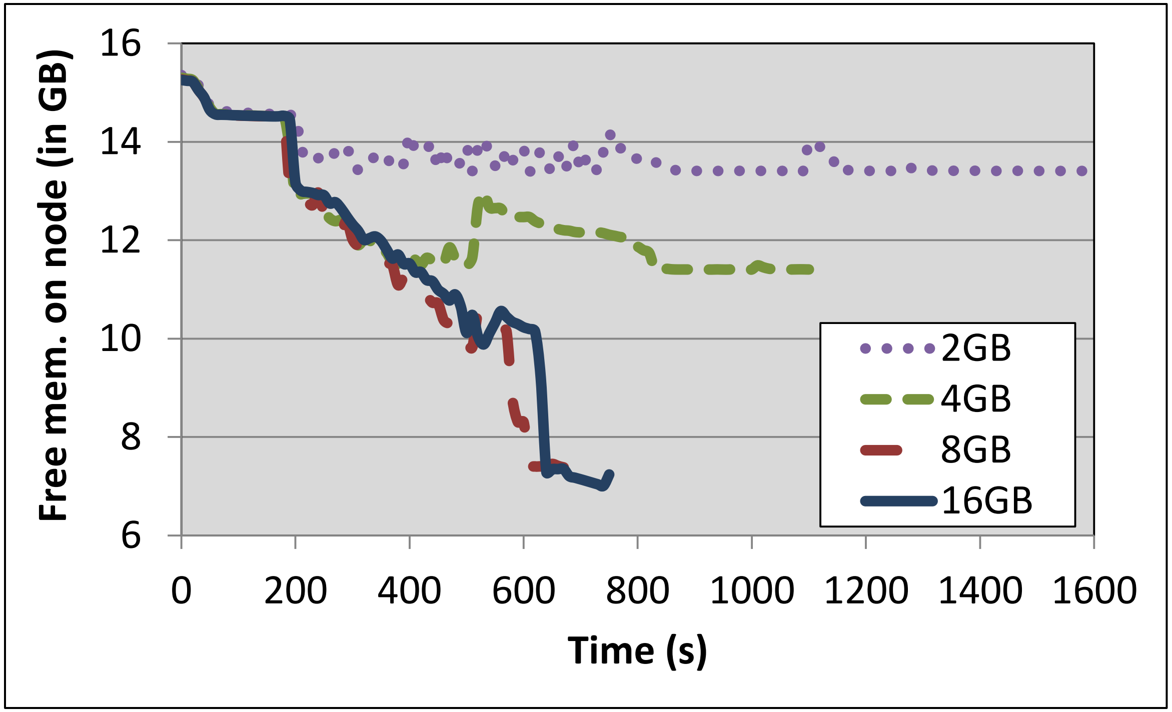
- % of memory reads satisfied by NUMA-local DRAM for SciMark 2.0 benchmarks with each HotSpot configuration.
- Performance with each implementation is (again) roughly the same

Memory Prioritization

- Example scenario:
  - Several applications compete over same memory pool
  - Allocations for low-priority apps. contend with high priority app.

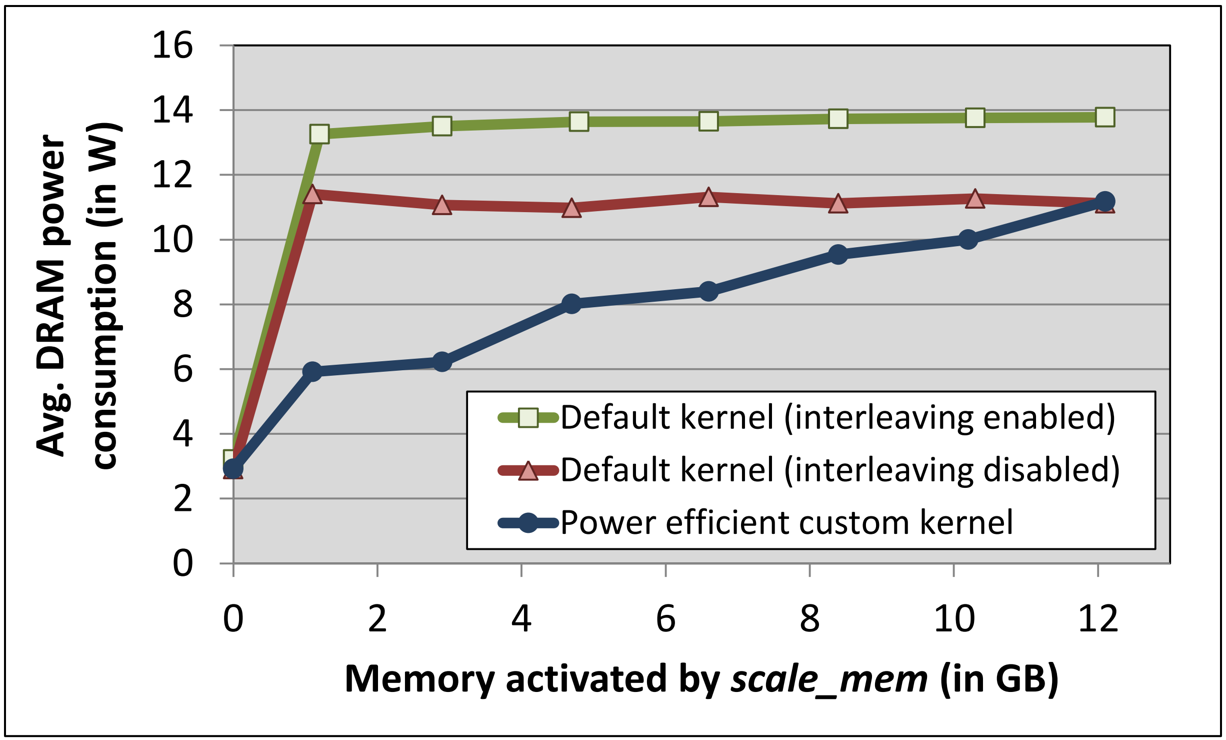
- No way for systems to directly prioritize memory
- *memnice*: restrict low-priority apps. from using the entire pool of memory
  - Use colors to set priorities for virtual pages
  - Low-priority allocations restricted to a subset trays

Memory Prioritization



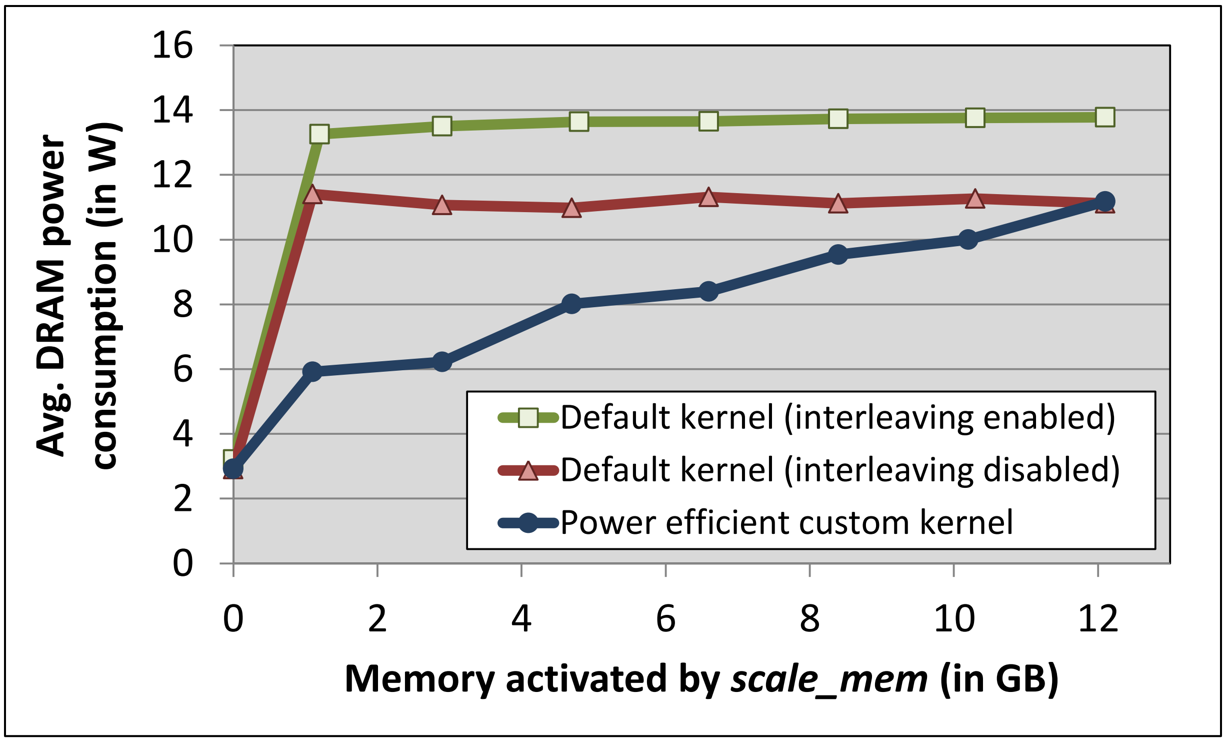
- Run kernel compilation with different memory priority configurations.
- Compute free memory on node during each compilation using /proc
- Restricted compilations stop expanding their memory footprints – but take longer to complete

Enabling Power Consumption Proportional to the Active Footprint



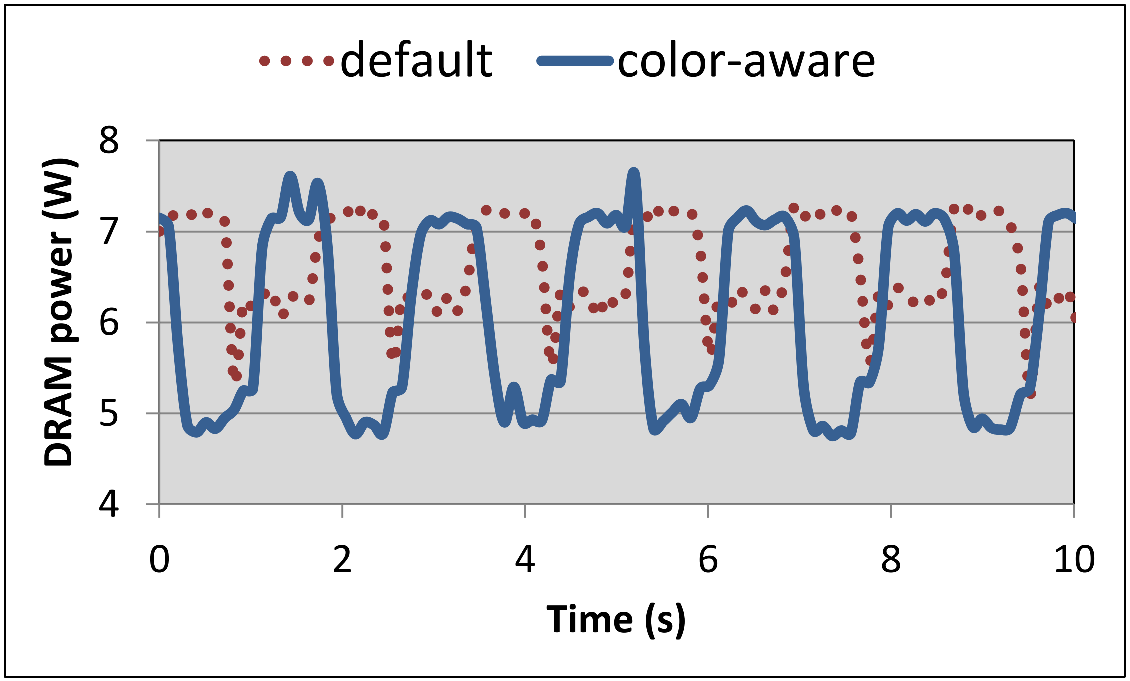
- Even a small memory footprint can prevent memory hardware units from transitioning to low-power states
- Custom workload incrementally increases memory usage in 2GB steps

Enabling Power Consumption Proportional to the Active Footprint



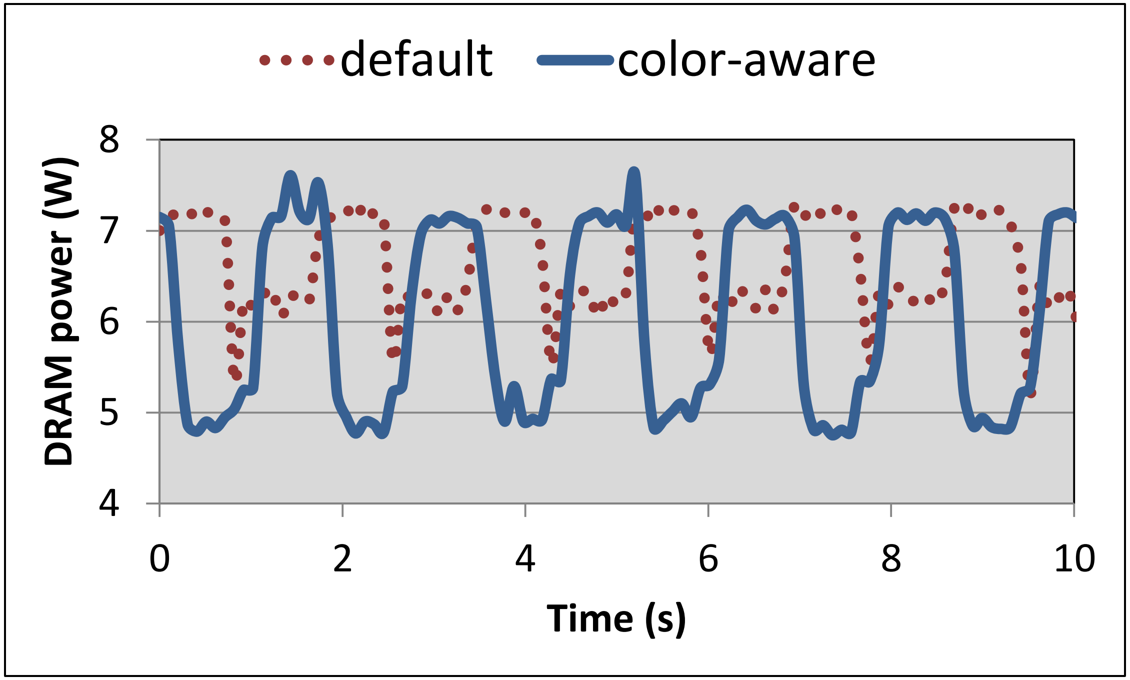
- Default kernel yields high power consumption even with small footprint
- Custom kernel – tray-based allocation enables power consumption proportional to the active footprint

Coloring Generational Garbage Collection to Reduce DRAM Power



- Memory power optimization with generational garbage collection
  - Isolate older generation on its own power-manageable unit
  - Older generation powers down during young generation GC

Coloring Generational Garbage Collection to Reduce DRAM Power



- DRAM power consumption for *derby* benchmark
- Dips correspond to garbage collection
- Isolating the older generation saves about 9% power over the entire run.

Conclusion

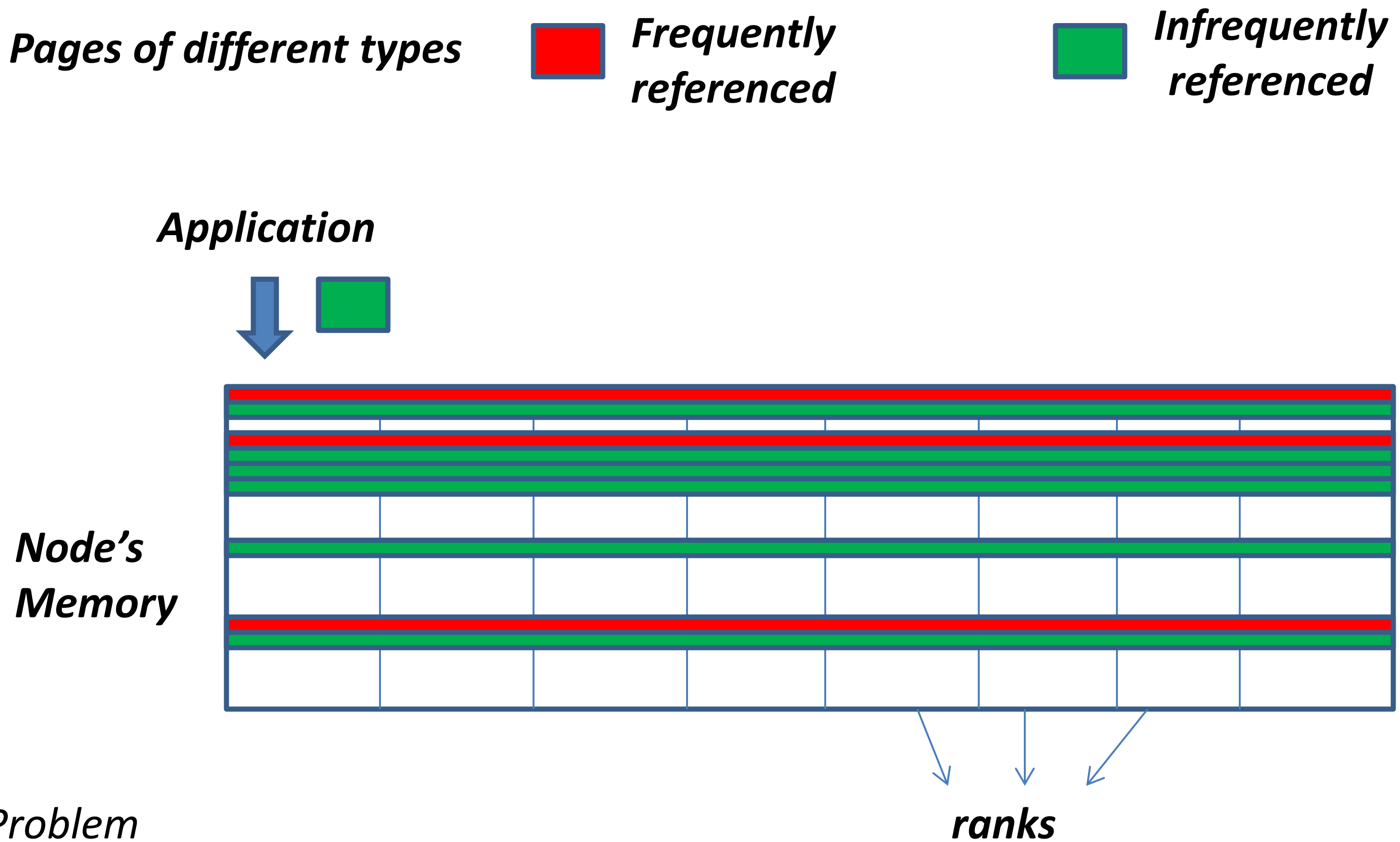
- A critical first step in meeting the need for a fine-grained, power-aware flexible provisioning of memory.
- Initial implementation demonstrates value
  - But there is much more to be done
- Questions?

Backup

Future Work

- MPST
- Select server-scale software package to implement color awareness
- Other optimizations
  - Maximize performance
  - Application-guided read-ahead and/or fault-ahead
- Page recycling policies
  - Minimum residency time, capacity allocation, etc.
- Development of tools for instrumentation, analysis, and control

Default Linux Kernel

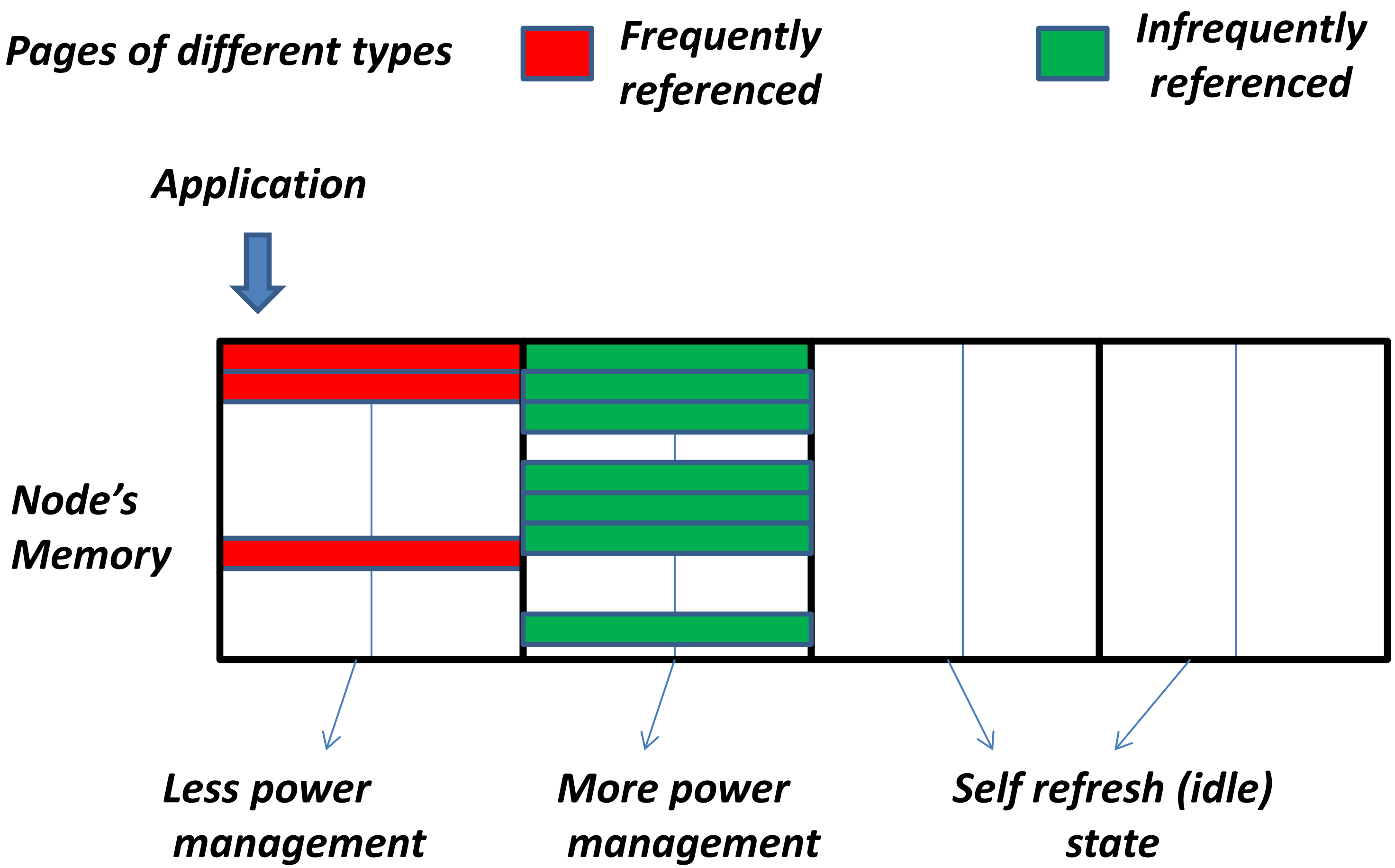


Problem

Operating system does not see a distinction between:

- ***different types of pages*** *from the application*
- ***different units of memory*** *that can be independently power managed*

Custom Kernel with Memory Containerization



Note: not drawn to scale- 10^6 4kB pages can be contained in a 4GB DIMM

Controlling Memory

- Difficult to control distribution of memory bandwidth, capacity, or power
  - Temporal and spatial variations in application memory usage
  - Depend on how virt. mem. binds to phys. mem.
- Layout of each application's hot pages affects DRAM power and performance:
  - Low activity: condense hot pages onto a small set of ranks (reduce power)
  - High activity: spread pages across as many ranks as possible (maximize perf.)

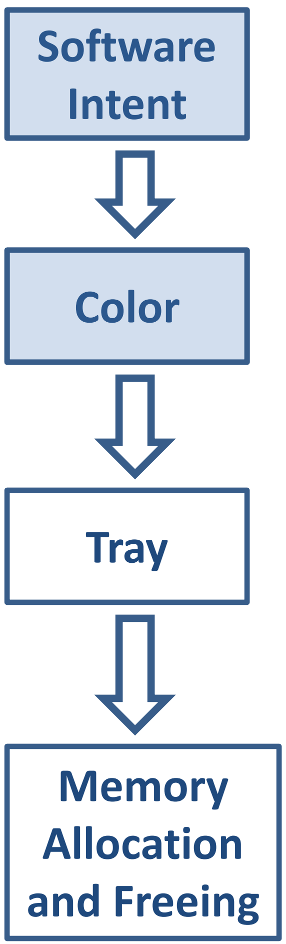
Our Approach

- *Our approach: Enable apps. to guide physical memory mgmt.*

Application Guided Memory Management

- Scenarios for application-guided, fine-grained memory management
  - An application wants to manage different sets of pages differently
  - An application wants to reduce the size of its memory footprint
  - An application wants to control its page eviction
- All the above scenarios can greatly make memory usage more efficient (equivalent thread level scenarios are currently possible for CPUs)
- Why is this not done for memory systems?
  - no mechanism for the application ***to pass information about its memory references*** to the OS and hardware
  - no mechanism for the OS to use this information to ***confine pages to different subsets of the total spatial capacity*** of memory

Mapping Intent to Color



- Example: Application with pages that are expected to be frequently accessed (hot) and pages with relatively infrequent references (cold)
- Intent: co-locate hot pages on separately power-managed memory units than cold pages
- Alignment to a set of "standard" intents:

  INTENT MEM-INTENSITY

- Mapping intent to colors is done with a configuration file:

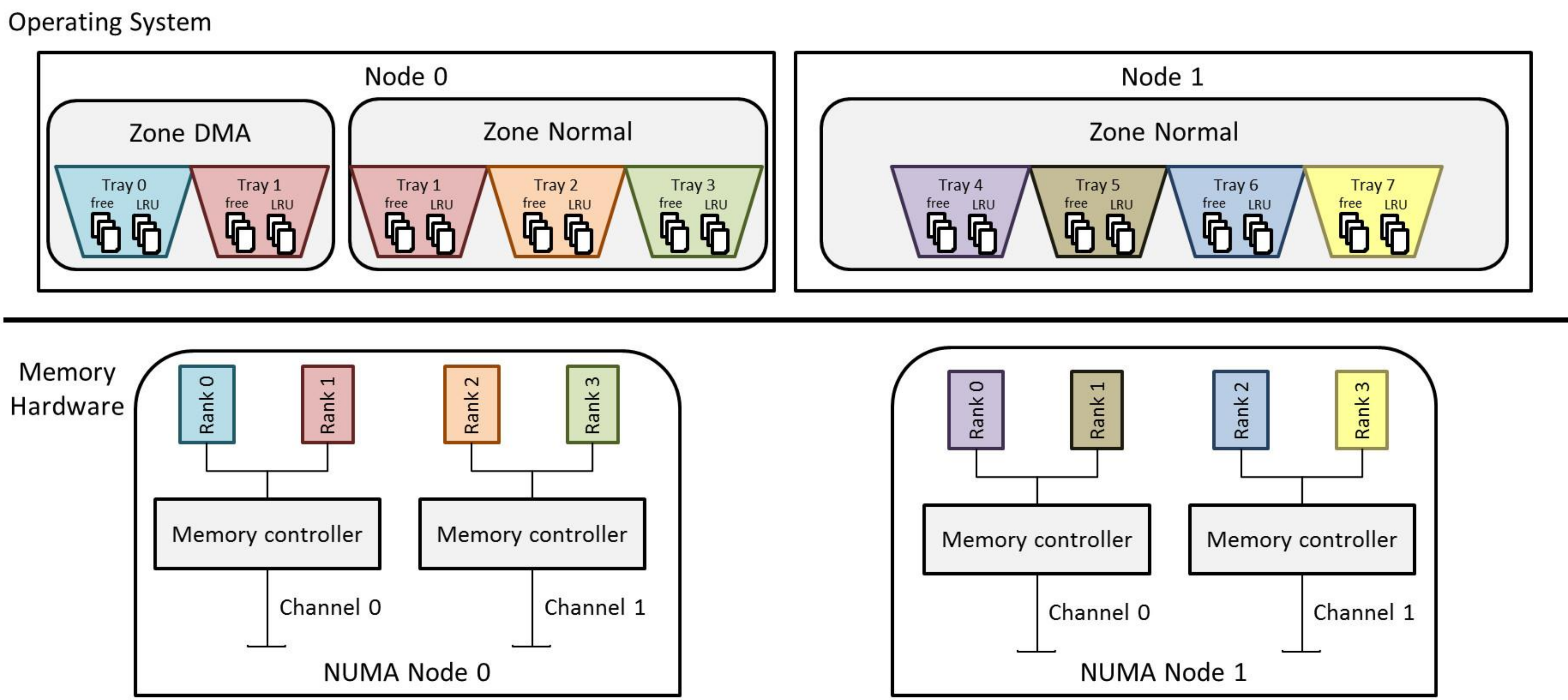
MEM-INTENSITY	RED	0	// hot pages
MEM-INTENSITY	BLUE	1	// cold pages

- In the application source code, color hot pages **RED** and cold pages **BLUE** with the *mcolor* system call:

  addr = malloc (hot_object_size);

  mcolor(addr, hot_object_size, **RED**);

Trays in the Linux Kernel



Emulating the NUMA API

- Modern server systems enable memory management at the level of NUMA nodes
  - Systems include an API and toolkit for controlling memory placement on NUMA nodes

- Our framework manages resources at the more fine-grained level of power-manageable units, but is *flexible* enough to emulate the functionality of the NUMA.

- NUMA API as colors:
  - Intent: Restrict some virtual range to physical allocations from node 1, some other virtual range to nodes 2 and 3
  - Example mapping:

```
SOCKET_AFFINITY_ABSOLUTE    RED      1      /* allocate only from node 1 */
SOCKET_AFFINITY_ABSOLUTE    BLUE     2,3    /* allocate only from nodes 2&3 */
SOCKET_AFFINITY_RELATIVE    WHITE    1      /* allocate node local */
SOCKET_AFFINITY_RELATIVE    YELLOW   0      /* allocate anywhere */
```

NUMA Optimization in the HotSpot JVM

- Oracle/Sun’s HotSpot JVM uses the NUMA API to improve DRAM access locality.
- Hypothesis: threads that allocate an object are the most likely to use the object.
- NUMA optimization in HotSpot JVM:
  - “Eden” space is divided into different regions per NUMA node and the physical memory corresponding to each eden region is bound to a particular NUMA node (via the NUMA API)
  - Application’s newly allocated objects are placed into the eden space local to the allocating thread.
- *To emulate NUMA with our framework, we color each eden space region with the appropriate SOCKET_AFFINITIZATION color*

Reducing DRAM Power Consumption

- Memory ranks transition to low-power states (such as “self refresh”) during periods of low activity.
- Mixing frequently accessed pages with pages that are not accessed very often on the same rank will increase the number of ranks that need to stay powered up.
- Our framework has the potential to reduce DRAM power consumption by allowing users to more consciously bring about periods of low activity at the memory rank level

Reducing DRAM Power Consumption

- To demonstrate the power saving potential of our framework, we designed a "power-efficient" memory management configuration
- When allocating each page, we opt to choose a tray that has already furnished a page for similar use.
  - In this way, total number of additional ranks that need to stay powered up is reduced

- Experiment with simple workload:
  - Allocate increasing amounts of memory in stages
  - Each stage allocates enough additional memory to fit in exactly one rank (2GB, in our case)
  - Each stage continuously reads and writes the allocated memory and lasts for 100 seconds.