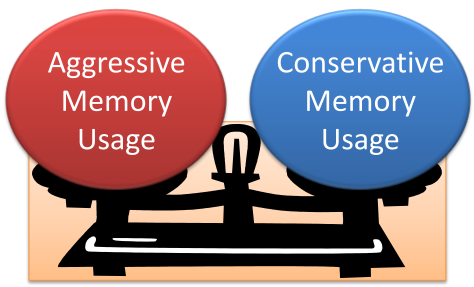
# Dynamic and Efficient Memory Sharing for Cloud Computing Environments

Eiji Kawai
eiji-ka@nict.go.jp
NICT

## Motivation

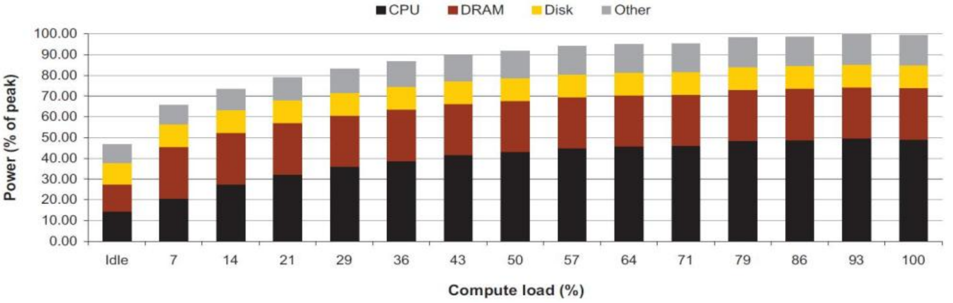
- Aggressive memory usage for higher performance
  - Ex. #1: Data object cache in distributed computing environments
  - Ex. #2: Advanced page sharing among virtual hosts




- Conservative memory resource provisioning for stable performance
  - Memory management is highly virtualized and opaque
    - Virtual memory mechanisms (paging in OS)
    - Multistage virtualization (paging in VMM)
  - Memory shortage causes severe performance degradation (slashing)

# Motivation (cont'd)

- More memory is not a perfect solution
  - Higher TCO (capital cost, energy cost)
  - A possibility of system-wide catastrophe still remains




From Barroso and Hölzle, "The Datacenter as a Computer"



# Memory Paging Semantics

- Paging *evenly* preserves page data in swap when the pages are reclaimed
  - Data in a virtual memory space should be consistent and persistent
- Accesses to swap are highly optimized
  - Do not page-out the data consistent with that already in a secondary storage (e.g., non-dirty pages, text-area, and file system cache)

**What can we do further?**

## Approach

- Relaxing the memory paging semantics
  - Define a programming model for a memory area without data preservation semantics in the traditional paging mechanism
    - Data in such pages can be destroyed asynchronously
  - An asynchronous memory management mechanism
    - In memory shortage, a system can just reclaim pages without page-outs
- Aggressive memory users are forced to change their behavior in memory shortage to avoid system-wide severe performance penalty



## Related Technique: Weak pointer

- Weak pointer is a reference to a memory area which does not affect the behavior of garbage collection (GC)
  - To access a memory area that is reachable only by a weak pointer, a strong pointer must be generated from the weak pointer
  - Generation of a strong pointer fails if the memory area is already collected by GC
- Adopt the semantics of weak pointers into non-GC memory management

## Implementation

**Loose consistency model**

- State management and sharing mechanism

**Asynchronous memory management**

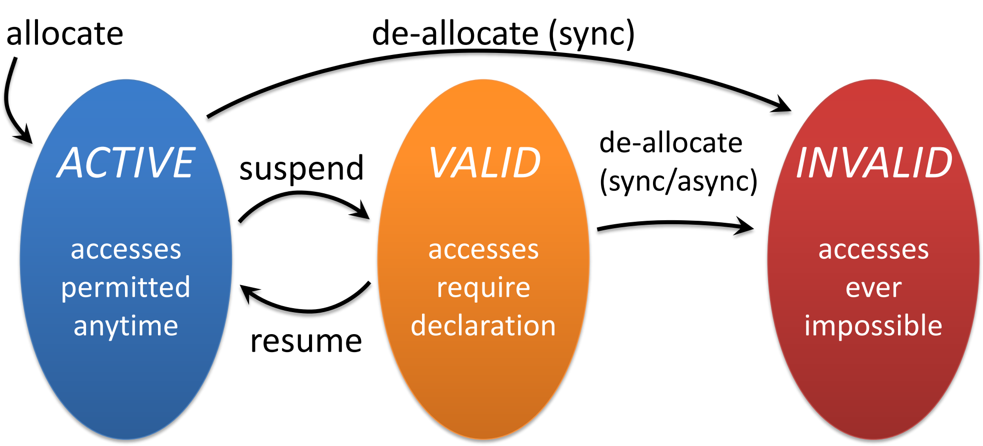
- Independent memory manager thread

**Extensions in low-level paging**

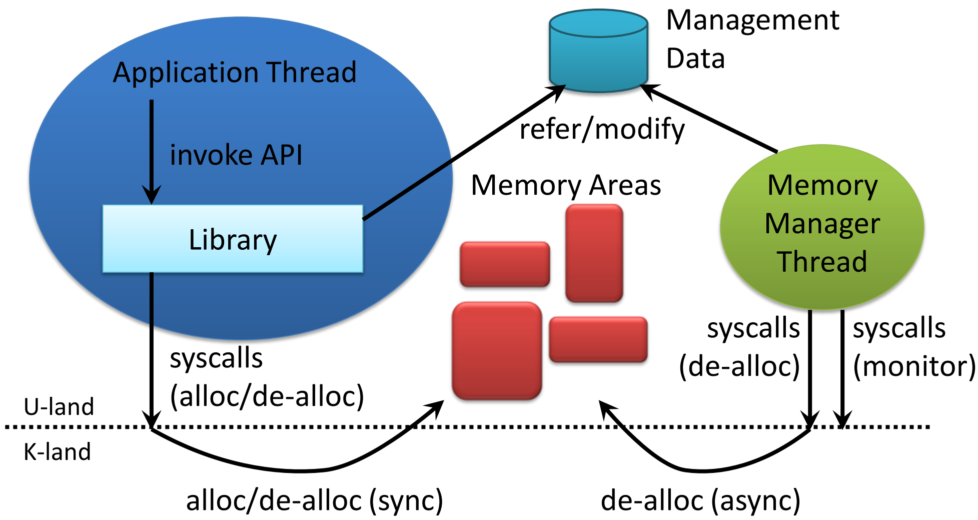
- Full control of page management and paging activities



## State Management



## Modules and Controls






## Low-level Paging

- Cannot depend on heap memory
  - `free()` does not always release pages
    - Heap is a single continuous area
- Utilizing memory mapping
  - Anonymous pages allow explicit allocation and de-allocation of pages
    - `mmap()` with `MAP_ANONYMOUS` option
    - Some `malloc()` implementations use `mmap()` to allocate a large area
  - Locked pages do not allow paging activities
    - `Mmap()` with `MAP_LOCKED` option

## Application Program Interfaces

- Library Initialization
  - Specify a replacement algorithm
- Memory allocation/de-allocation
  - Synchronous operations by user programs
- Memory state management
  - Explicitly suspend memory usage
- Memory access
  - Copy/move/compare/scan/set
    - `memcpy()/memmove()/memcmp()/memchr()/memset()`
  - Implicitly resume memory usage
  - Direct access with a virtual address is prohibited
    - Specify an offset



## Conclusion

- Relaxing the memory paging semantics can make aggressive and conservative memory usages compatible
  - Reducing the risk of system-wide performance catastrophe caused by slashing
- Future work
  - More consideration on implementation issues
    - System interface (system calls)
    - Middleware architecture
  - Multi-process/Multi-threading fairness issues
    - Synchronization among the memory manager threads
  - Performance Evaluation
    - Micro-benchmarks