SSDAlloc: Hybrid SSD/RAM Memory Management Made Easy

Anirudh Badam
Vivek S. Pai
Princeton University

03/31/2011

Memory in Networked Systems

Memory in Networked Systems

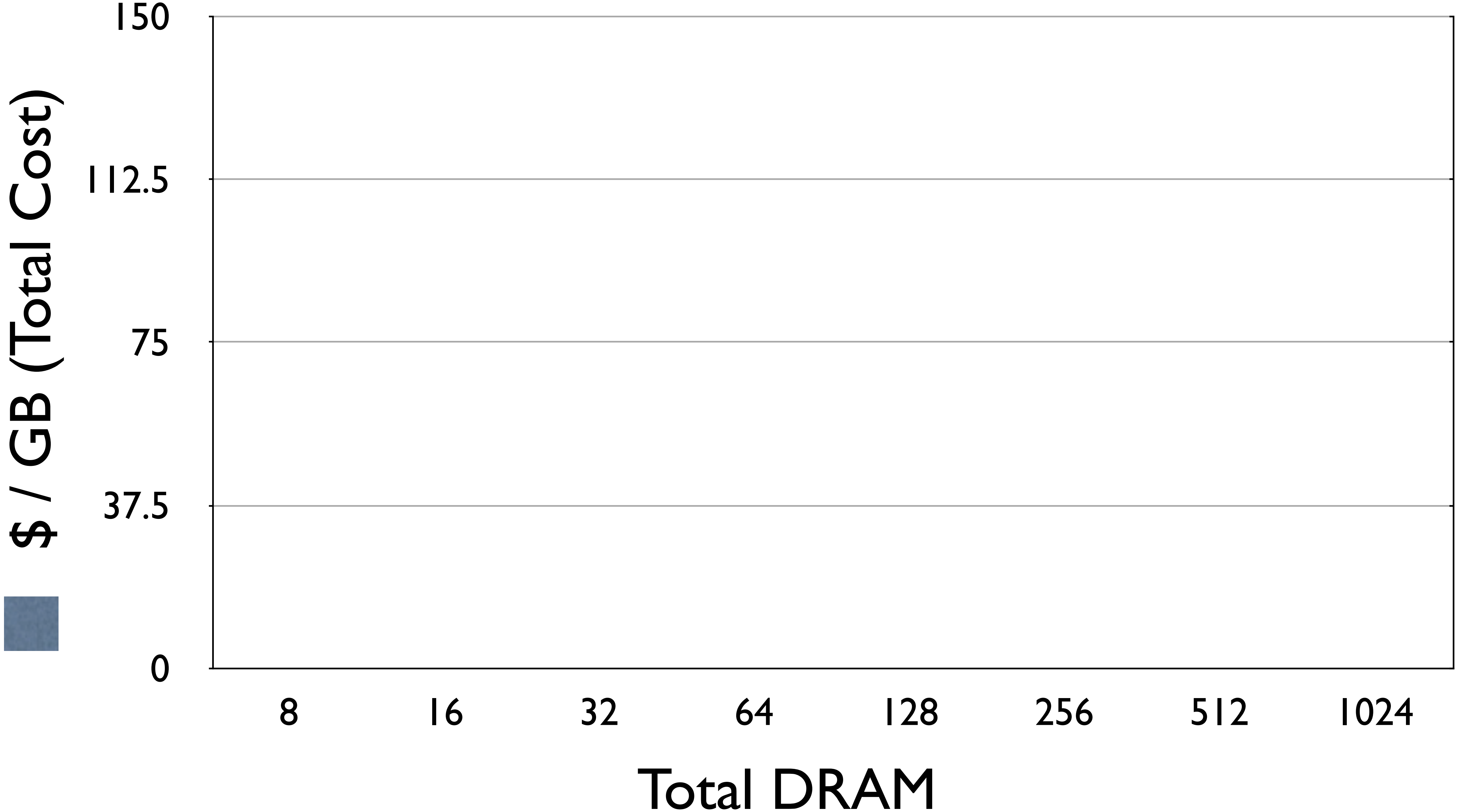
- As a cache to reduce pressure on the disk
  - Memcache like tools
  - Act as front-end caches for Web data back-end

Memory in Networked Systems

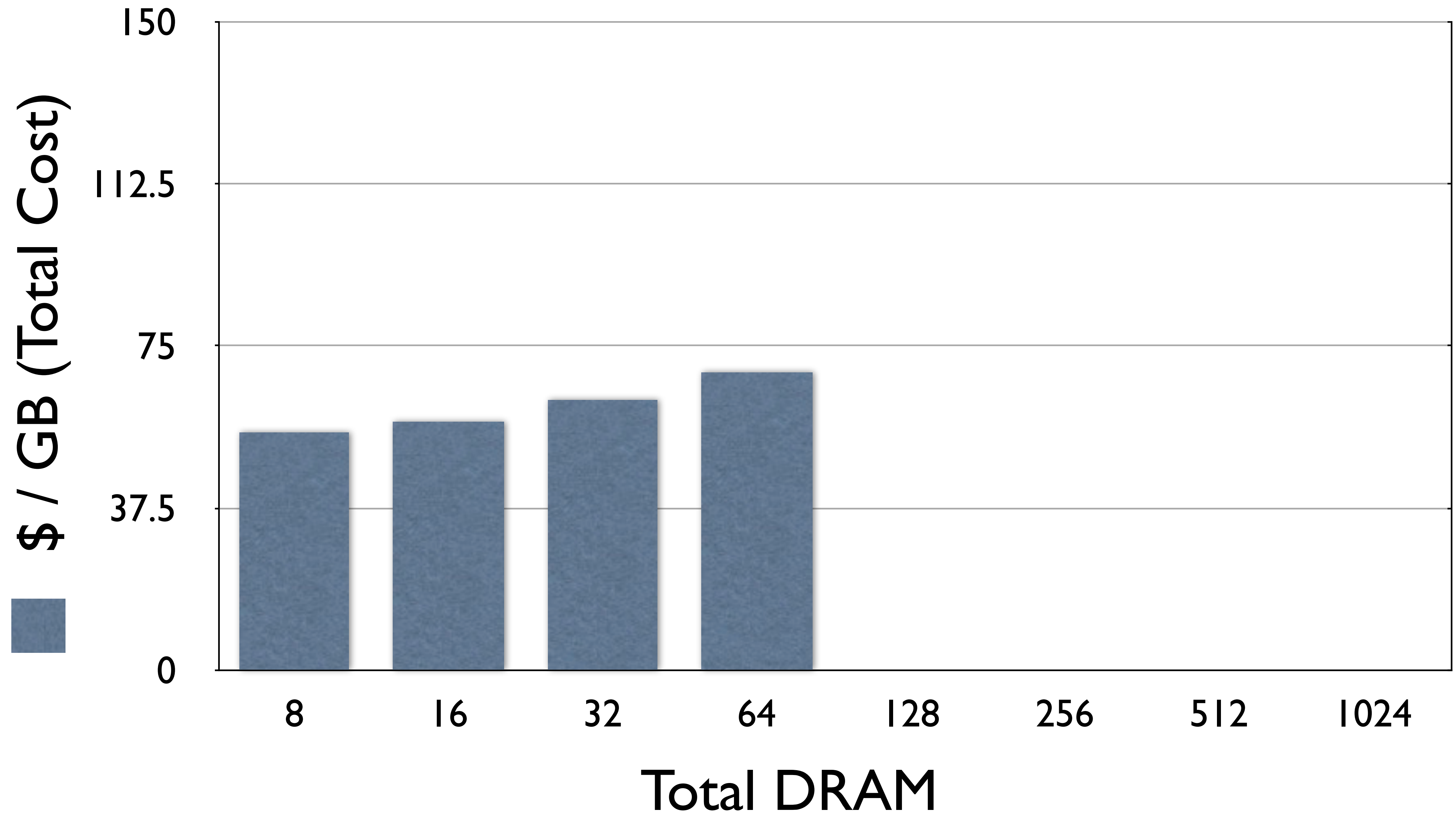
- As a cache to reduce pressure on the disk
  - Memcache like tools
  - Act as front-end caches for Web data back-end
- As an index to reduce pressure on the disk
  - Indexes for proxy caches, WAN accelerators and inline data-deduplicators
  - Help avoid false positives and use the disk effectively

Problem: Memory Density

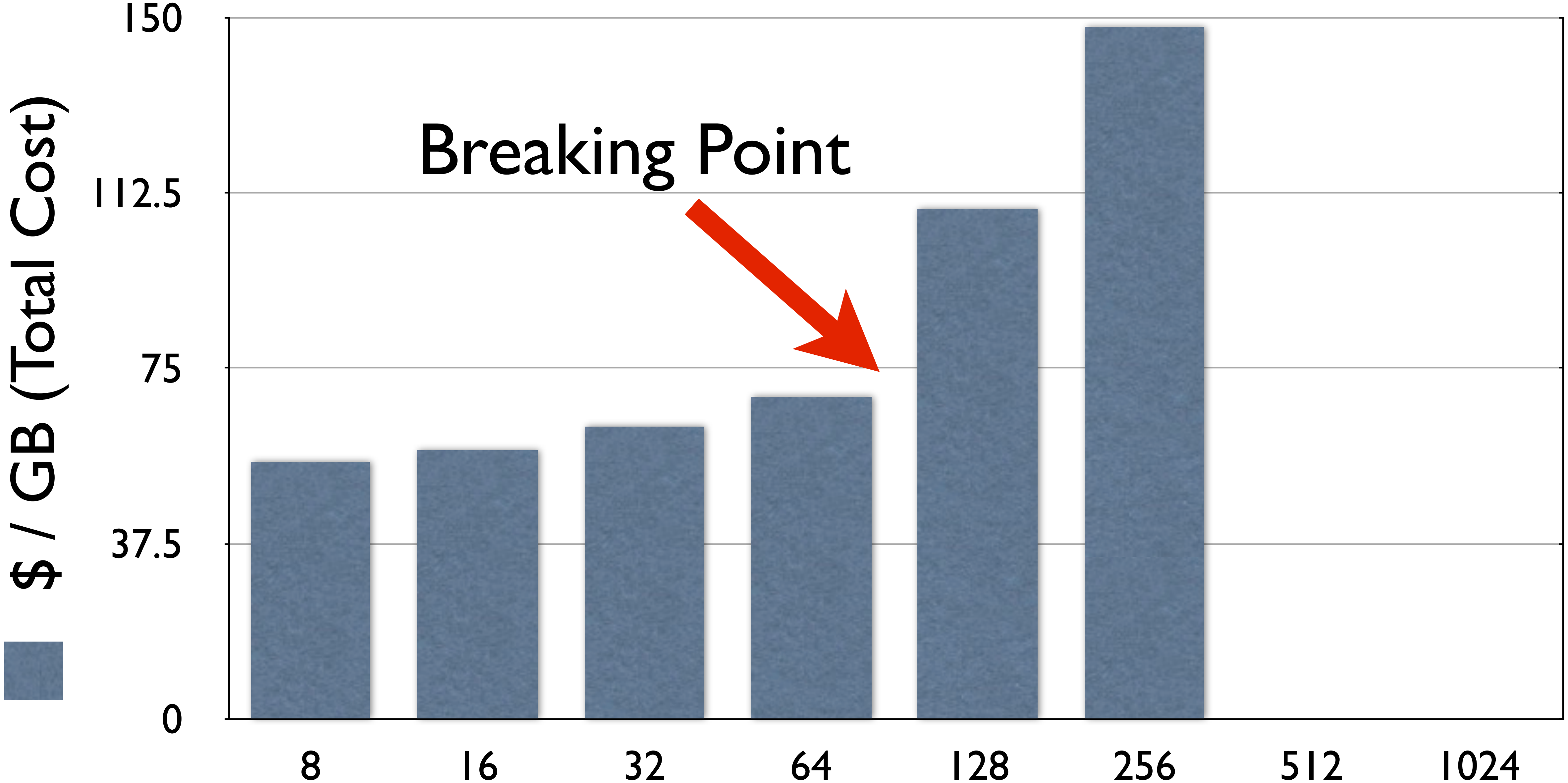
Problem: Memory Density



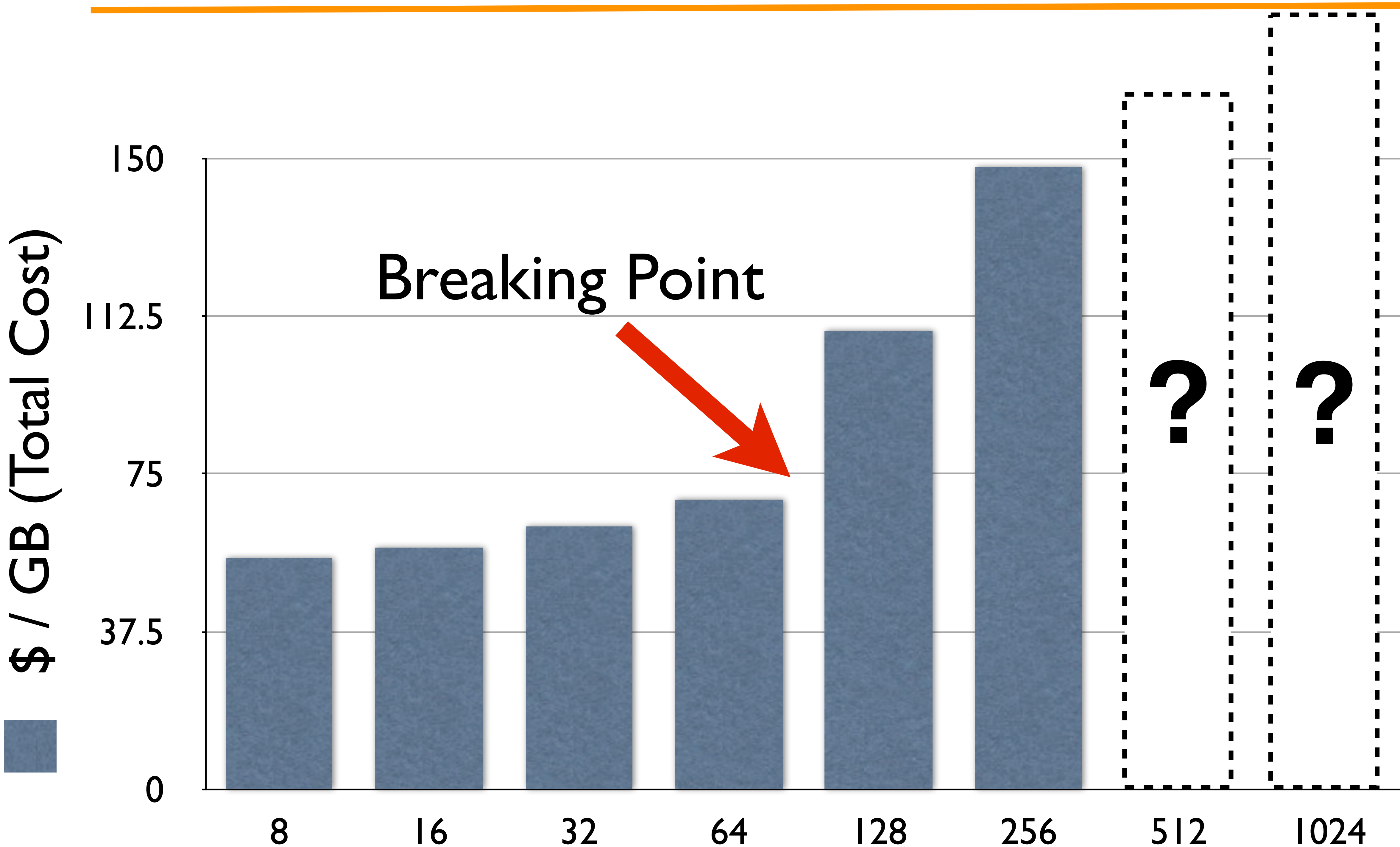
Problem: Memory Density



Problem: Memory Density



Problem: Memory Density



Problem: Disk Speed Limits

Problem: Disk Speed Limits

- Magnetic disk speed is not scaling well
  - Capacity is increasing but seek latency is not decreasing
  - About 200 seeks/disk/sec

Problem: Disk Speed Limits

- Magnetic disk speed is not scaling well
  - Capacity is increasing but seek latency is not decreasing
  - About 200 seeks/disk/sec
- High speed disk arrays: many smaller capacity drives
  - Total cost about 10X more compared to similar capacity 7200 rpm drives
  - Use more rack space per byte

Proposal: Use Flash as Memory

Proposal: Use Flash as Memory

- Address DRAM density limitation
  - Overcome per system DRAM limits via flash memory
  - Provide a choice -- more servers or a single server + flash memory

Proposal: Use Flash as Memory

- Address DRAM density limitation
  - Overcome per system DRAM limits via flash memory
  - Provide a choice -- more servers or a single server + flash memory
- Reduce total cost of ownership
  - “Long-tailed” workloads are important
  - DRAM too expensive and disk too slow
  - CPU under-utilized due to DRAM limit

Proposal: Use Flash as Memory

- Address DRAM density limitation
  - Overcome per system DRAM limits via flash memory
  - Provide a choice -- more servers or a single server + flash memory
- Reduce total cost of ownership
  - “Long-tailed” workloads are important
  - DRAM too expensive and disk too slow
  - CPU under-utilized due to DRAM limit
- **How to ease application development with flash memory?**

Flash Memory Primer

Flash Memory Primer

- Fast random reads (upto 1M IOPS per drive)

Flash Memory Primer

- Fast random reads (upto 1M IOPS per drive)
- Writes happen after an erase
  - Limited lifetime and endurance

Flash Memory Primer

- Fast random reads (upto 1M IOPS per drive)
- Writes happen after an erase
  - Limited lifetime and endurance
- No seek latency (only read/write latency)

Flash Memory Primer

- Fast random reads (upto 1M IOPS per drive)
- Writes happen after an erase
    - Limited lifetime and endurance
- No seek latency (only read/write latency)
- Large capacity (single 2.5" disk ~ 512GB)
    - PCIe 10.2 TB - Fusion-io io-octal drive

Question of Hierarchy

Question of Hierarchy

Memory	**Disk**

Question of Hierarchy

Memory	**Disk**
Byte Addressable	Block Addressable

Question of Hierarchy

Memory
Byte Addressable
Virtually Addressed

Disk
Block Addressable
Directly Addressed

Question of Hierarchy

Memory	Disk
Byte Addressable	Block Addressable
Virtually Addressed	Directly Addressed
Low Latency	High Latency

Question of Hierarchy

Memory
Byte Addressable
Virtually Addressed
Low Latency
Volatile

Disk
Block Addressable
Directly Addressed
High Latency
Non-volatile

Question of Hierarchy

Memory
Byte Addressable
Virtually Addressed
Low Latency
Volatile

Disk
Block Addressable
Directly Addressed
High Latency
Non-volatile

Question of Hierarchy

Memory
Byte Addressable
Virtually Addressed
Low Latency
Volatile

Disk
Block Addressable
Directly Addressed
High Latency
Non-volatile

SSDs

Question of Hierarchy

Memory	**Disk**
Byte Addressable	Block Addressable
Virtually Addressed	Directly Addressed
Low Latency	High Latency
Volatile	Non-volatile

SSDs

Flash has low latency ←

Question of Hierarchy

Memory		**Disk**
Byte Addressable		Block Addressable
Virtually Addressed		Directly Addressed
Low Latency		High Latency
Volatile	SSDs	Non-volatile

Flash has low latency

←

Transparent Tiering Today

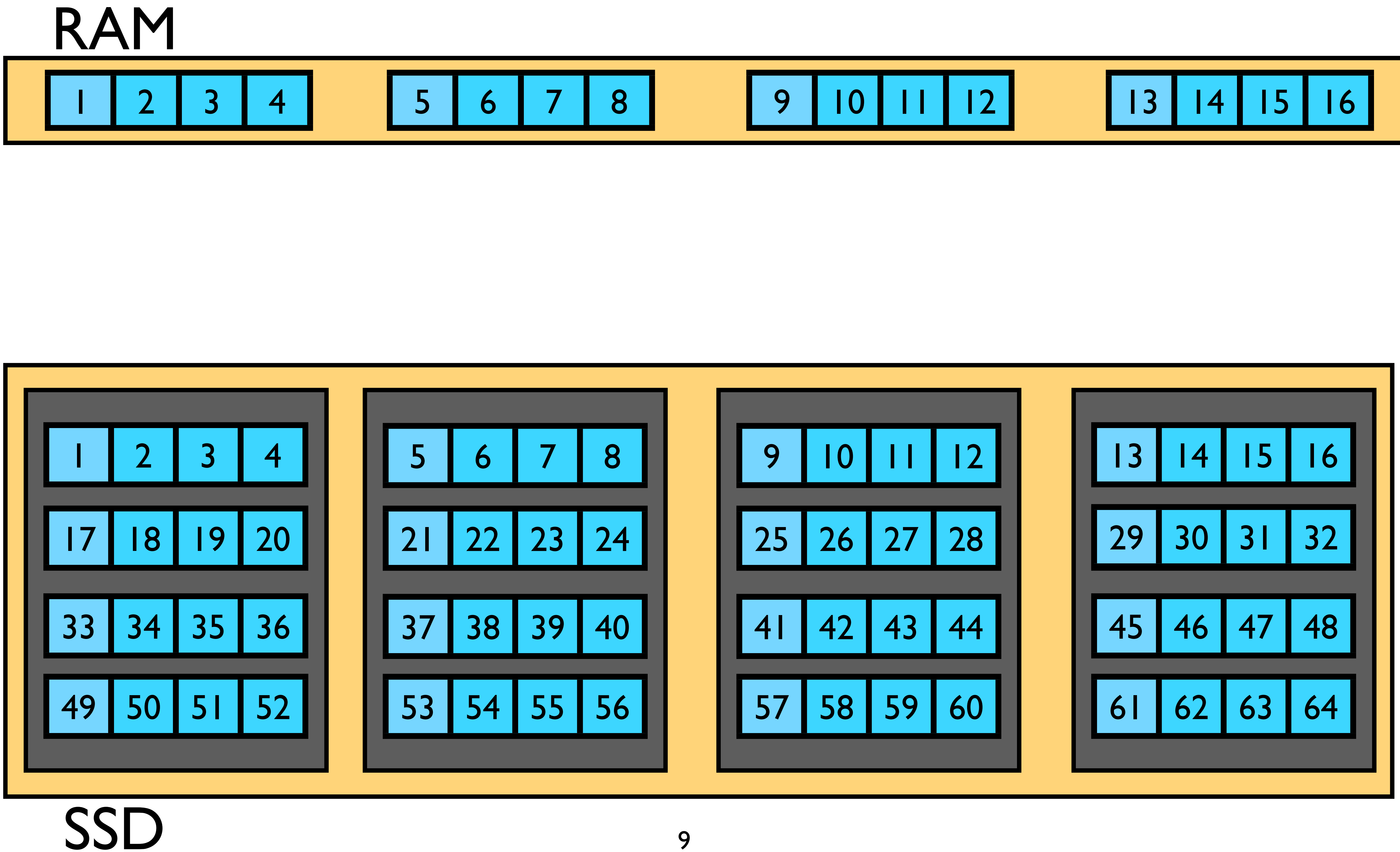
Transparent Tiering Today

- Use it as memory via swap or mmap
  - Application need not be modified
  - Pages transparently swapped in and out based on usage in DRAM

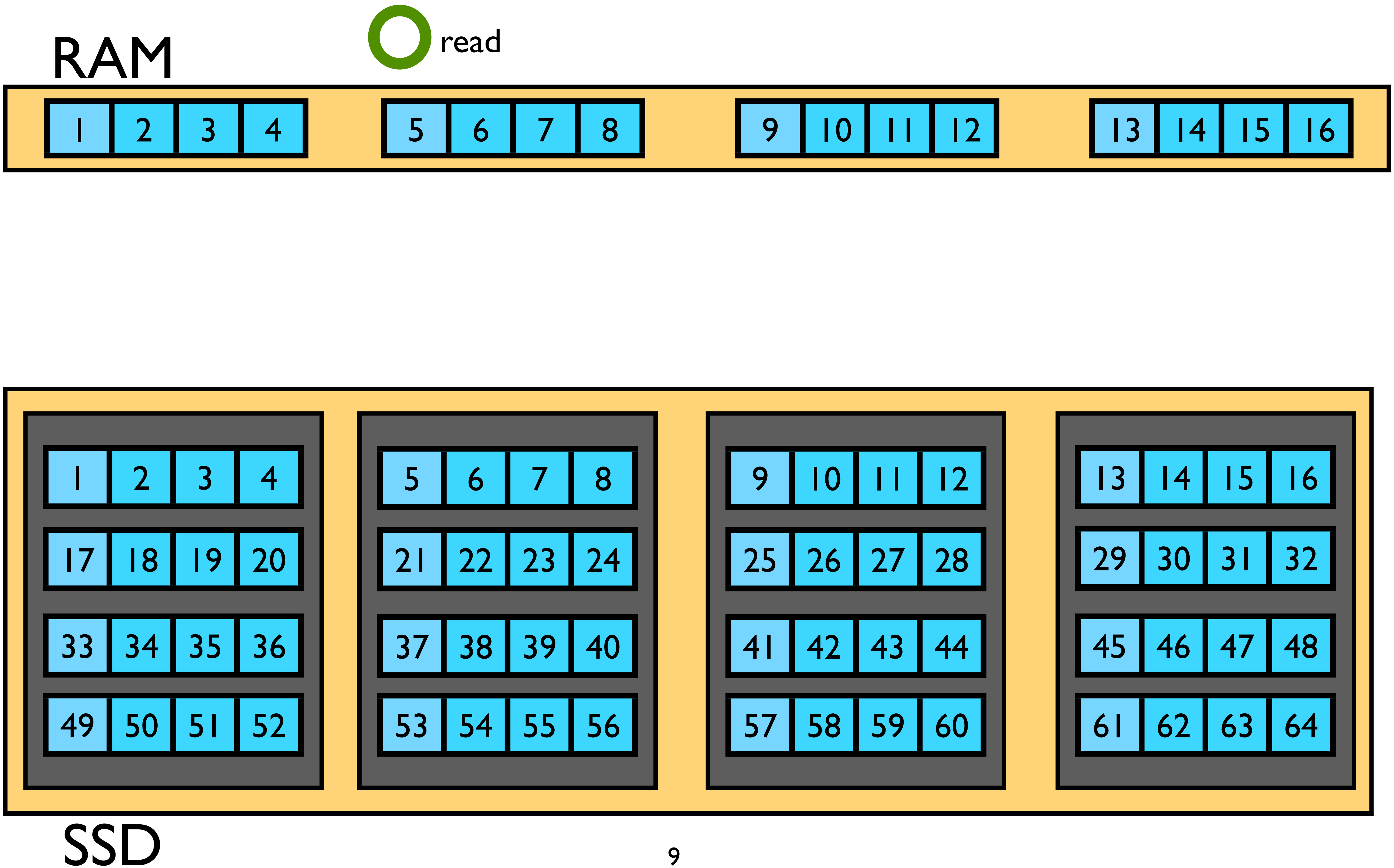
Transparent Tiering Today

- Use it as memory via swap or mmap
  - Application need not be modified
  - Pages transparently swapped in and out based on usage in DRAM
  - Native pager delivers only 10% of the SSD's performance
  - Flash aware pager delivers only 30% of the SSD's performance
  - OS pager optimized for seek limited disks and was designed as a "dead page storage"

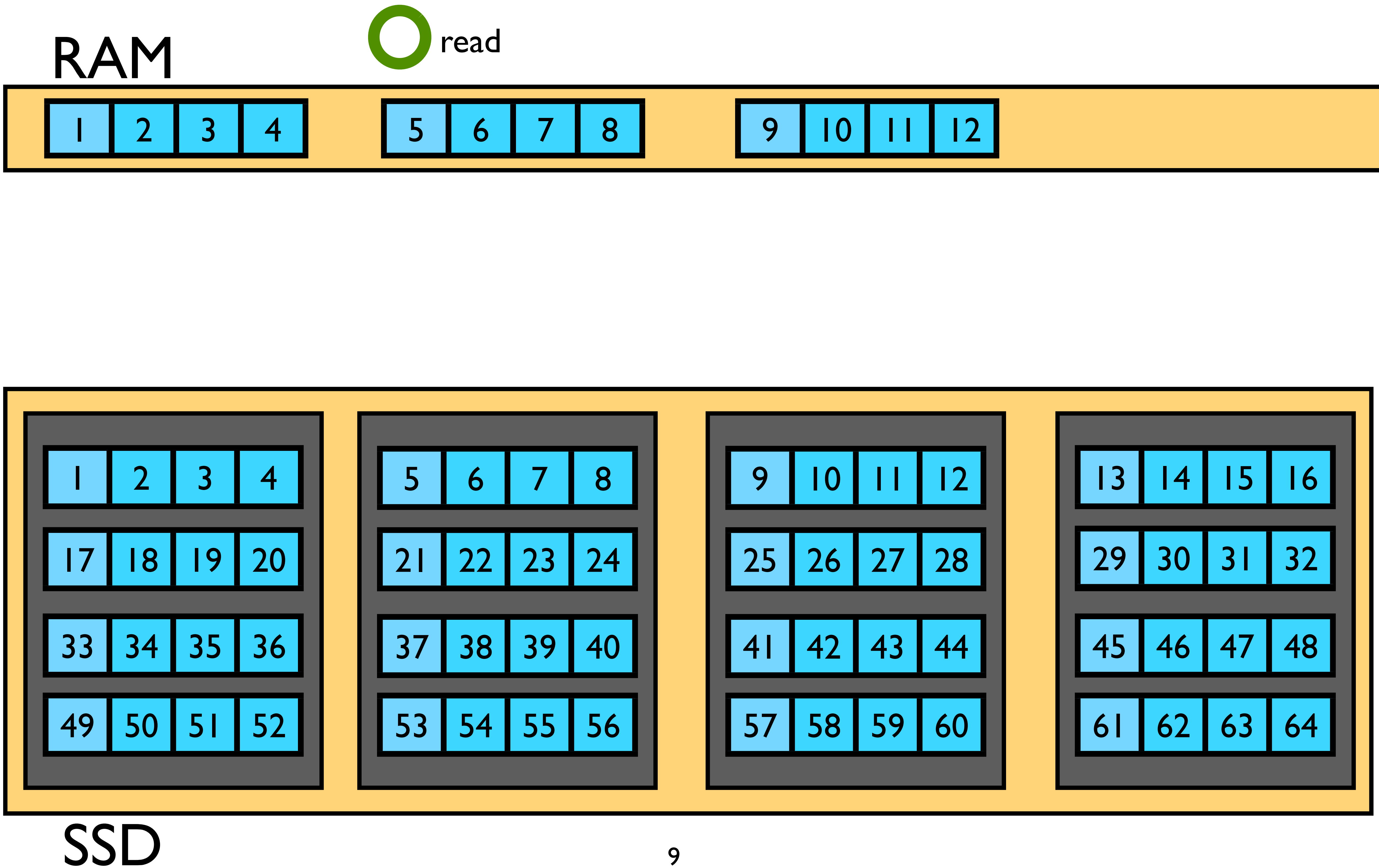
Transparent Tiering Today



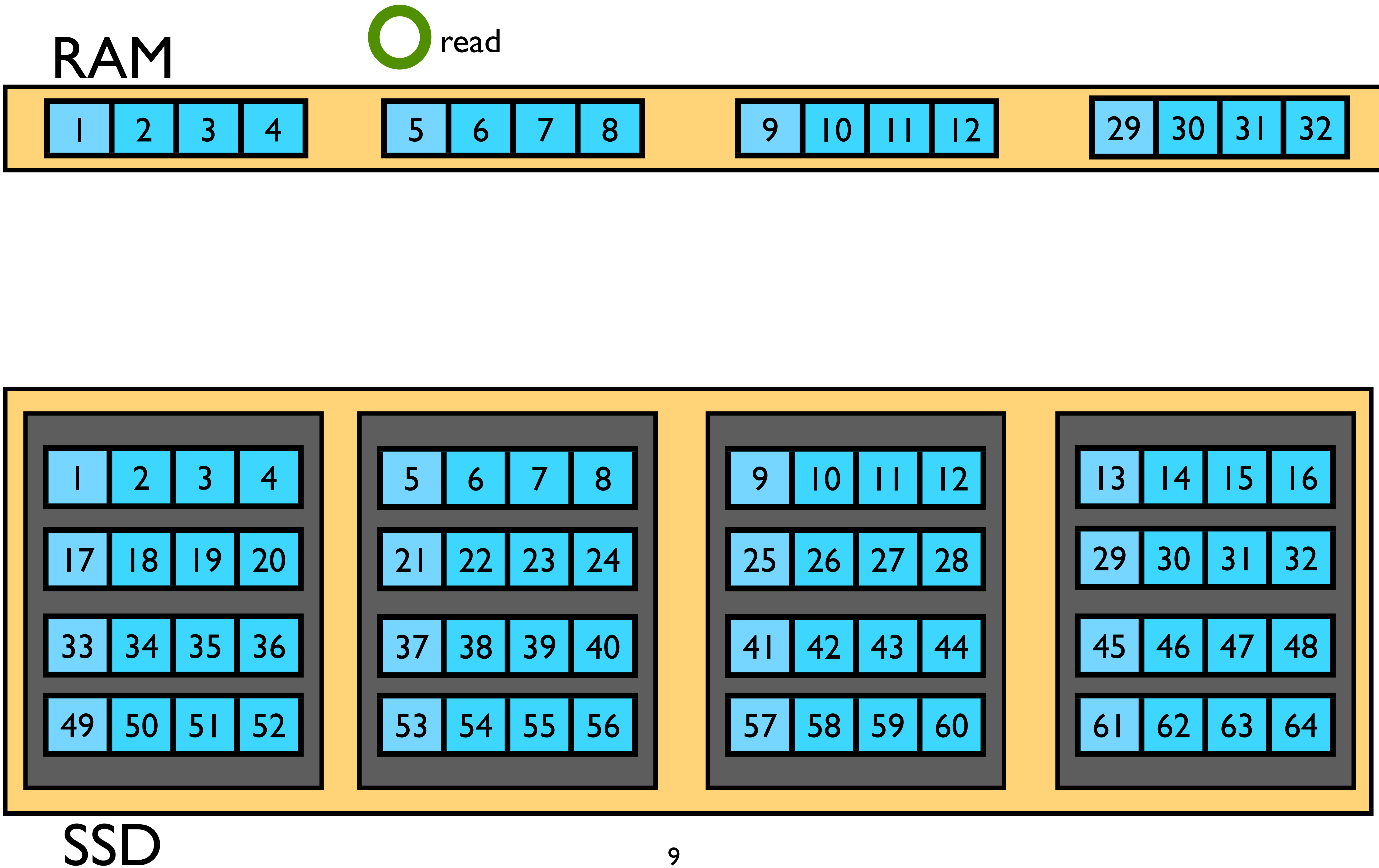
Transparent Tiering Today



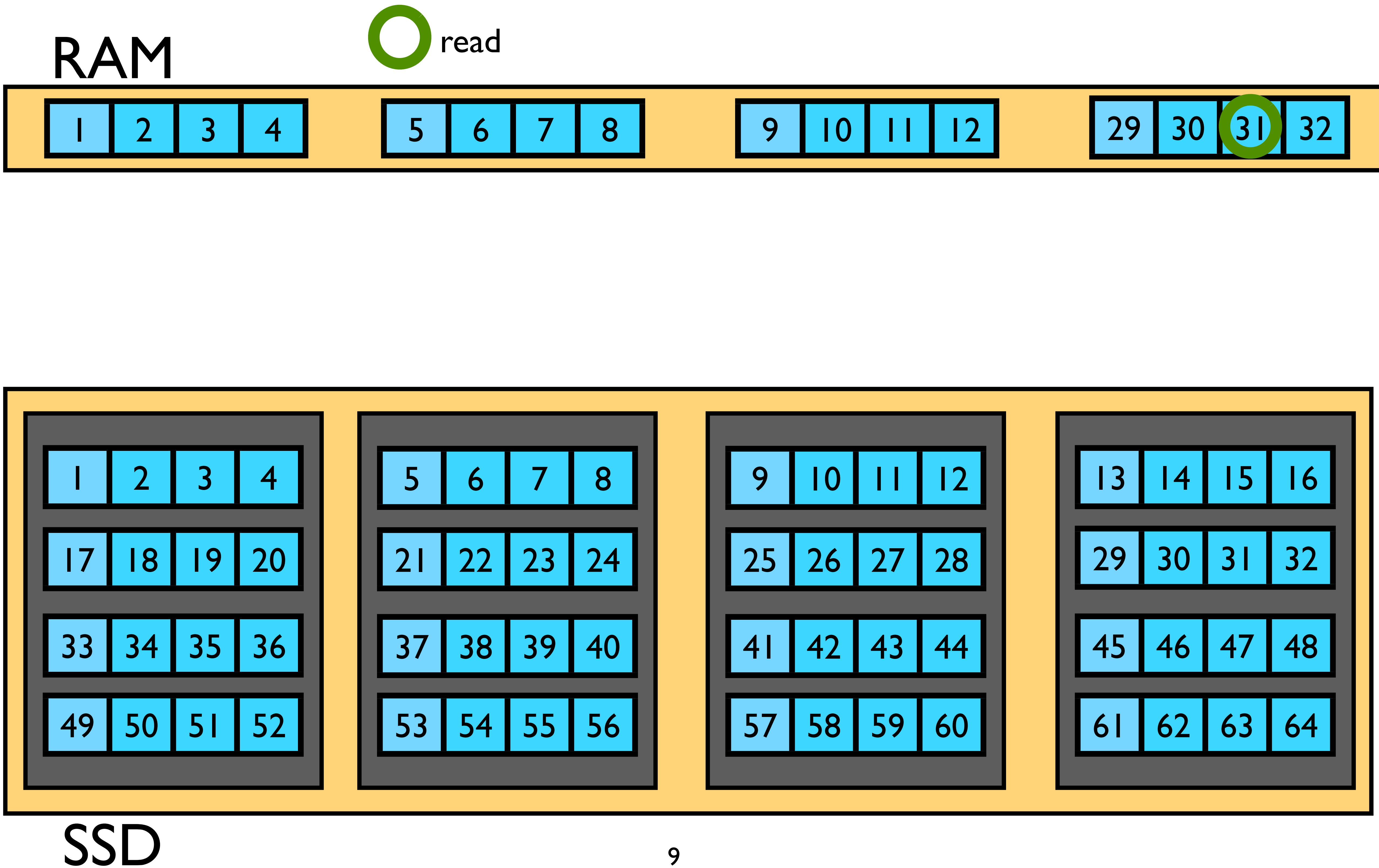
Transparent Tiering Today



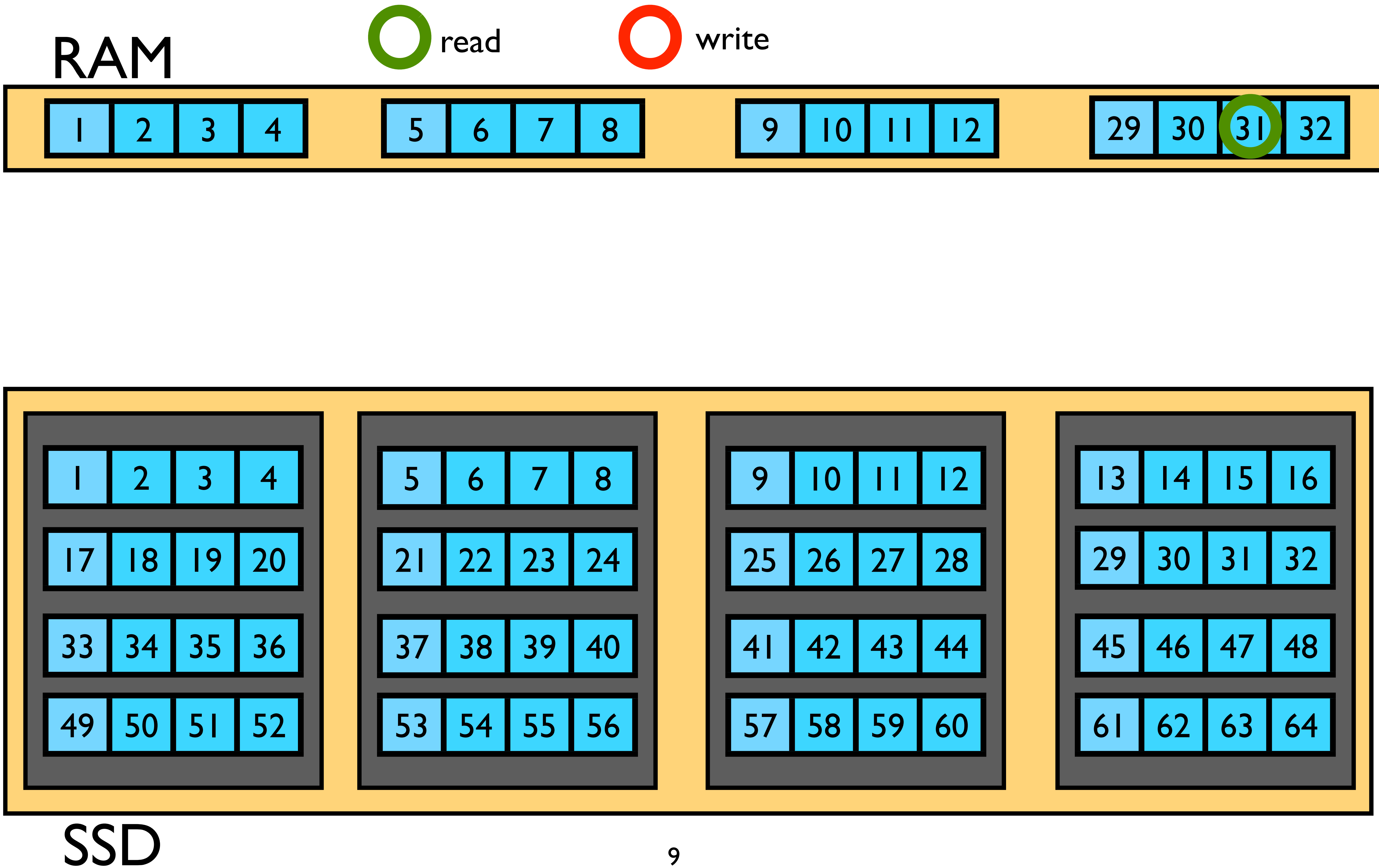
Transparent Tiering Today



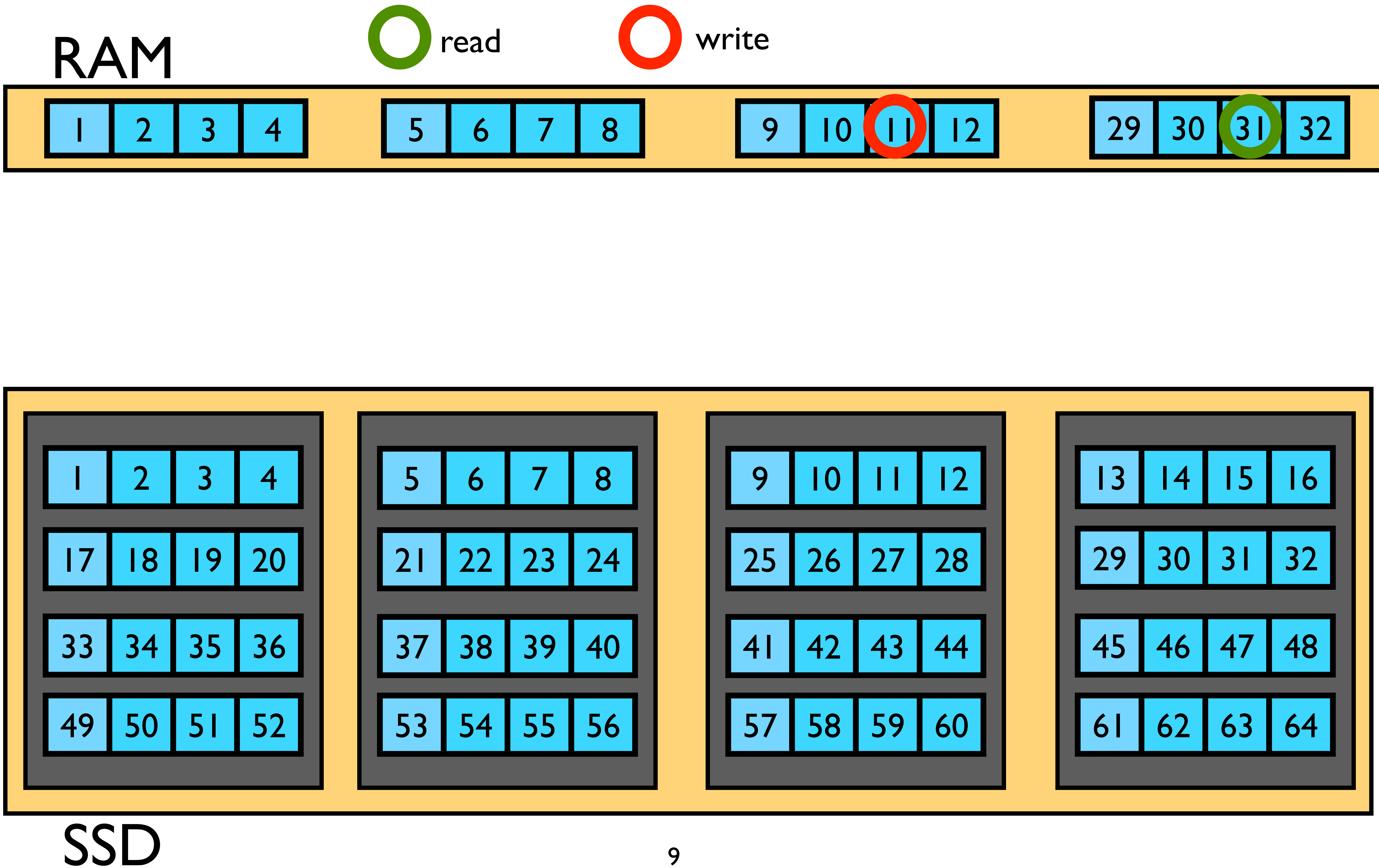
Transparent Tiering Today



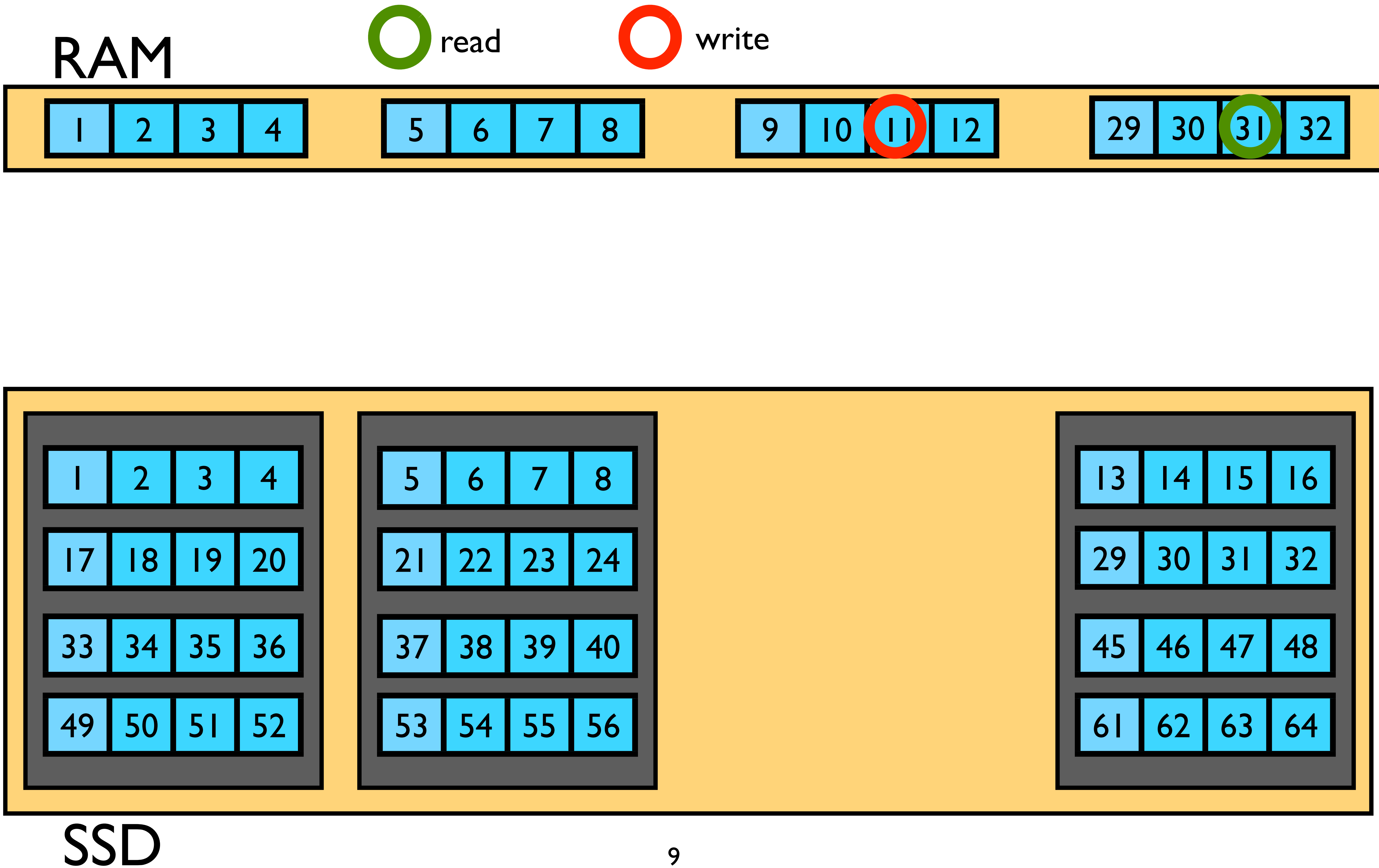
Transparent Tiering Today



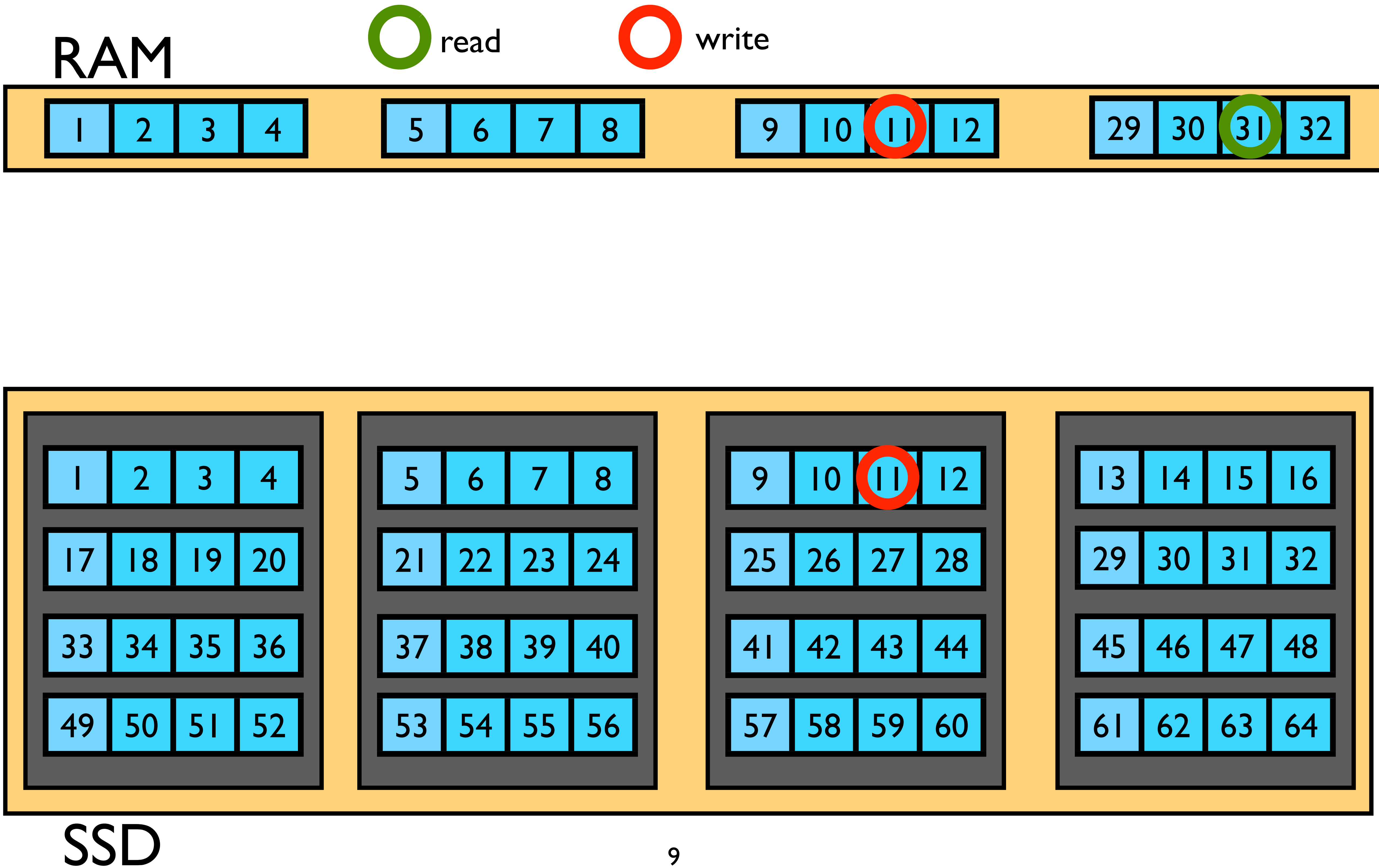
Transparent Tiering Today



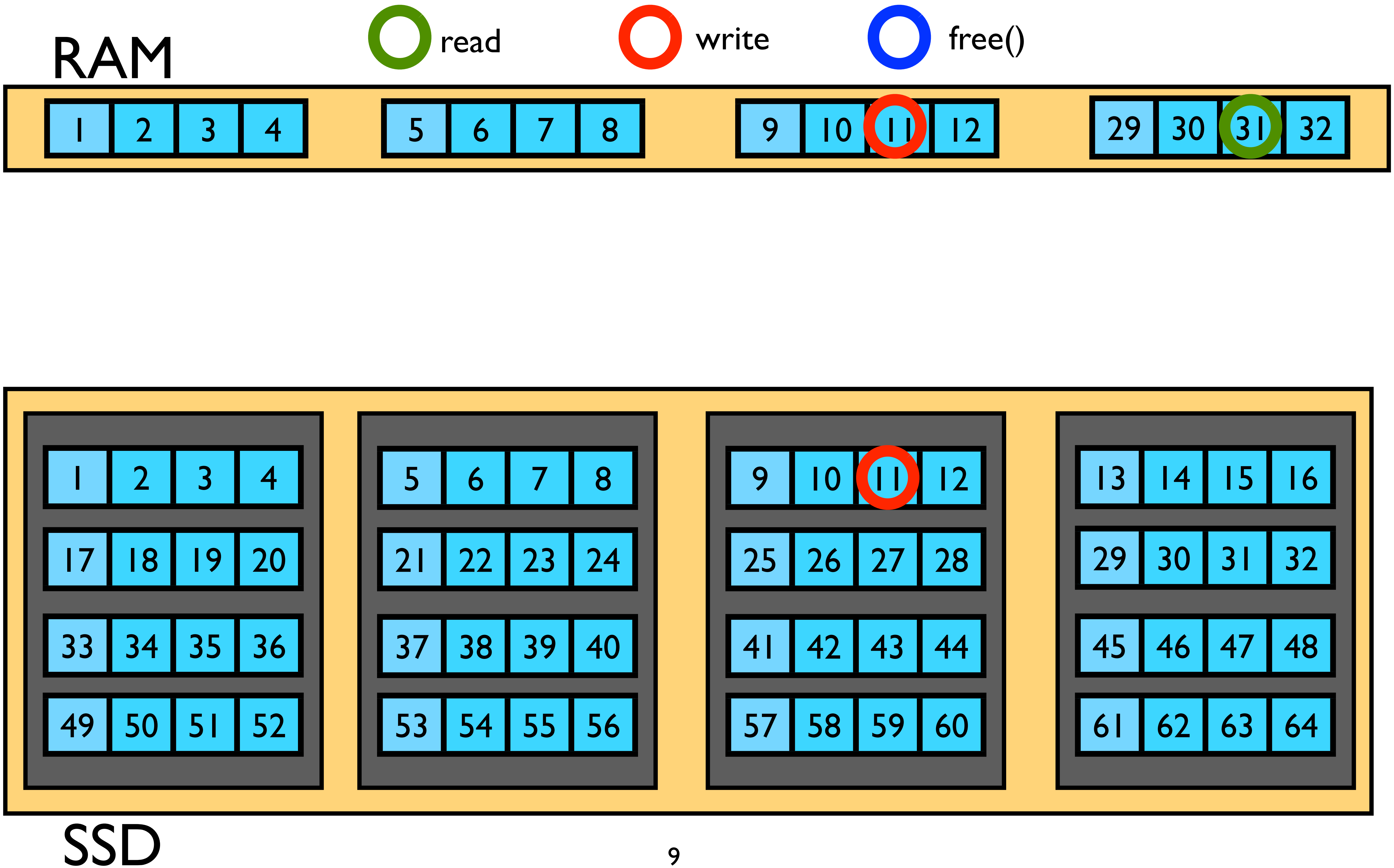
Transparent Tiering Today



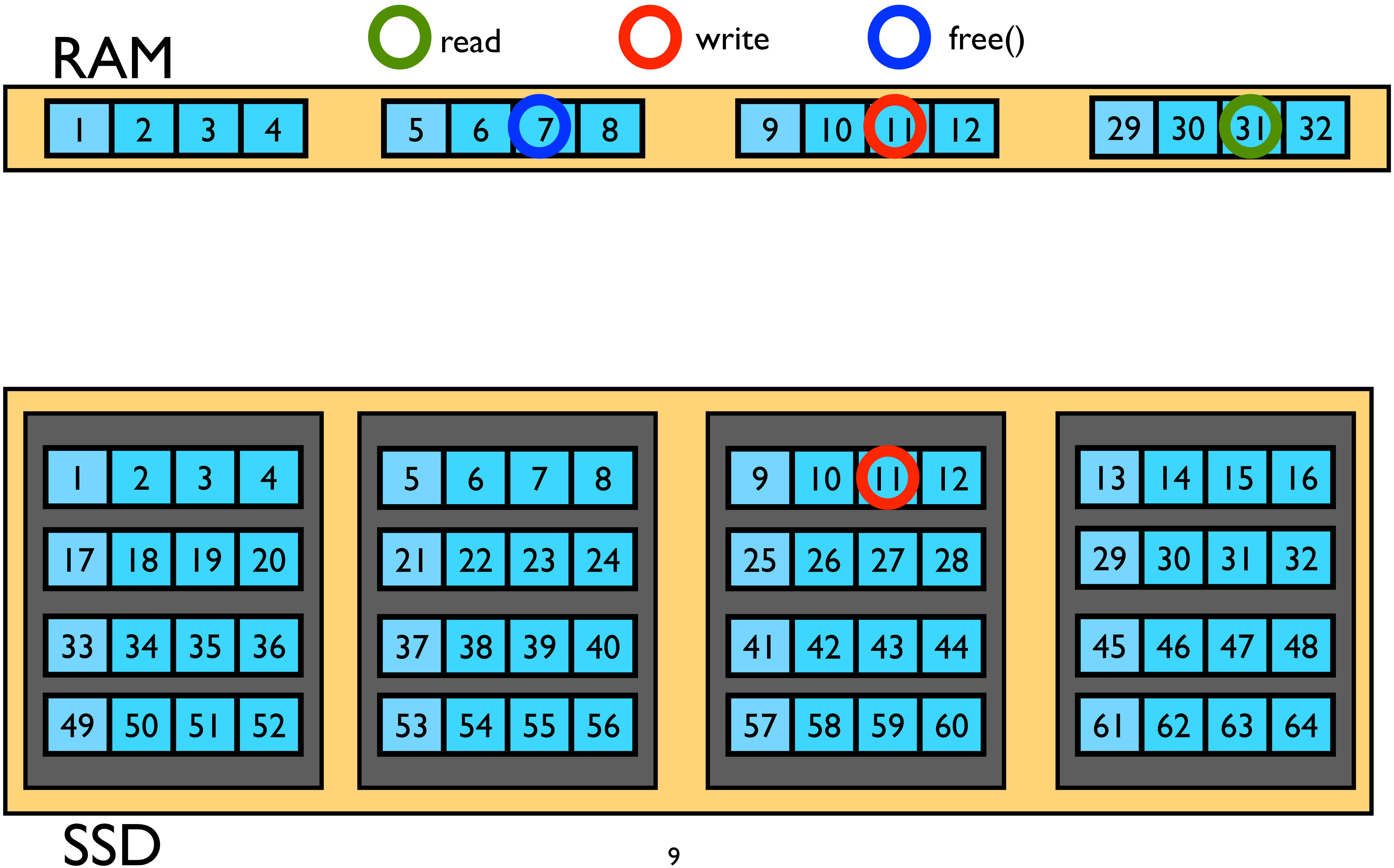
Transparent Tiering Today



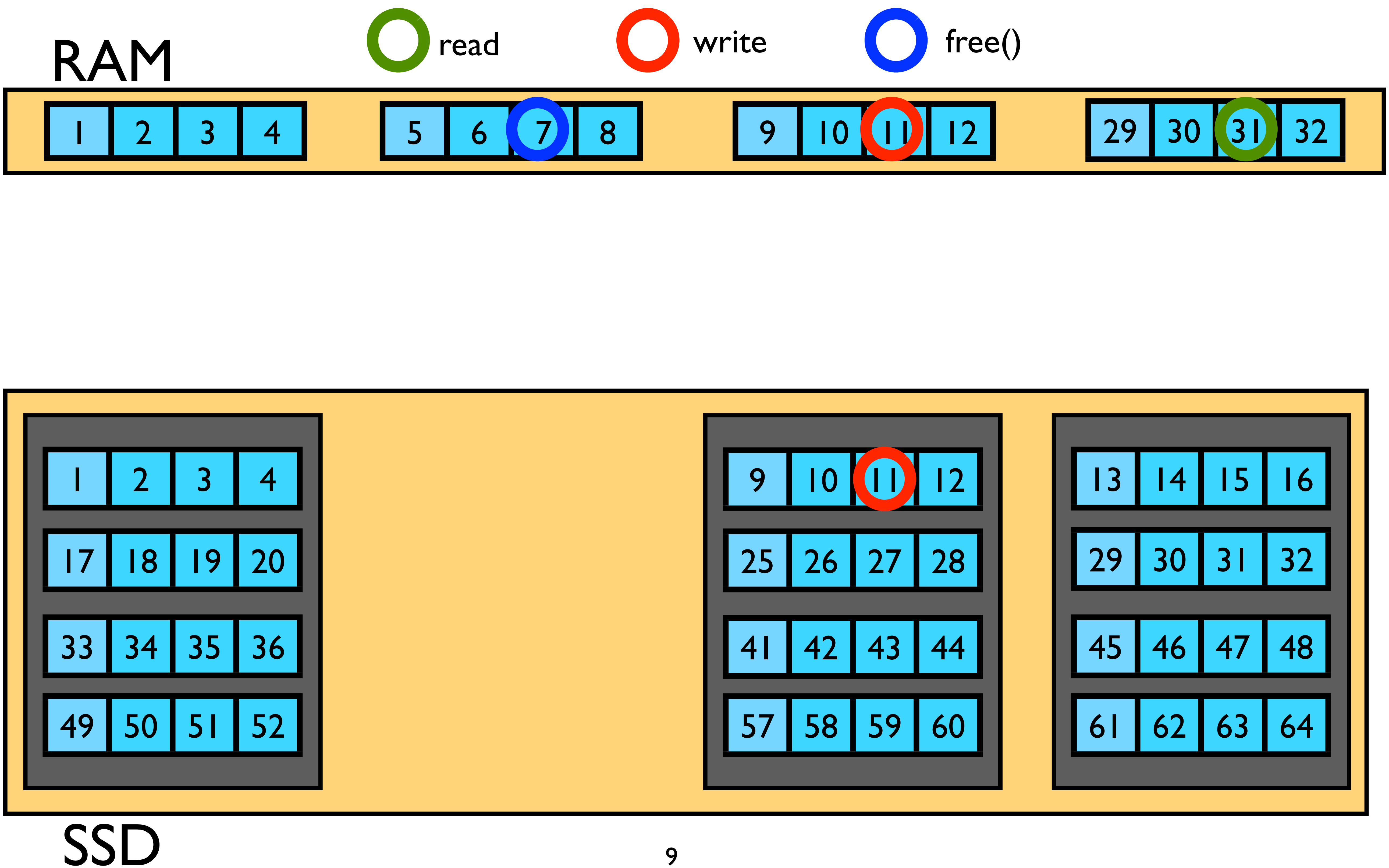
Transparent Tiering Today



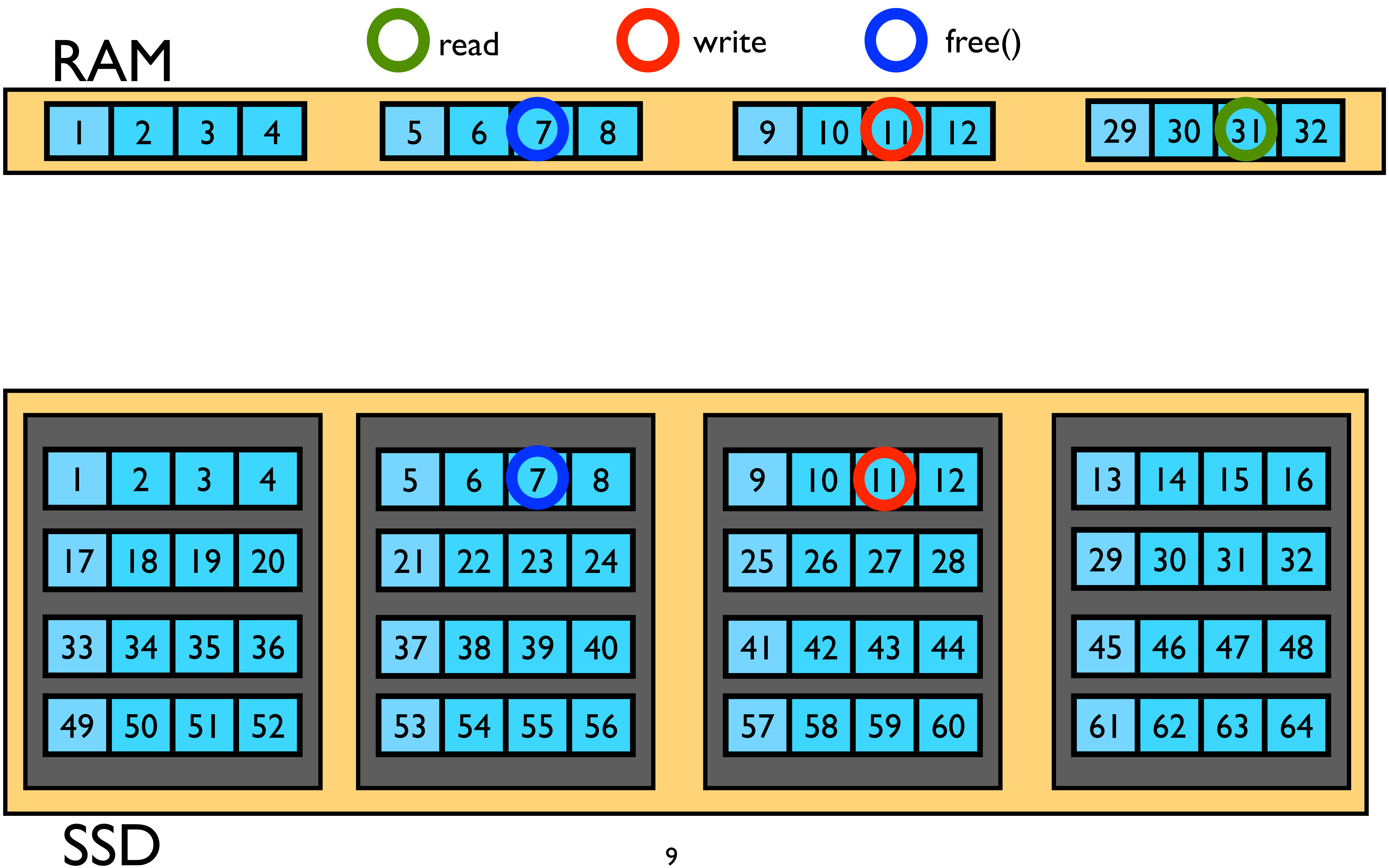
Transparent Tiering Today



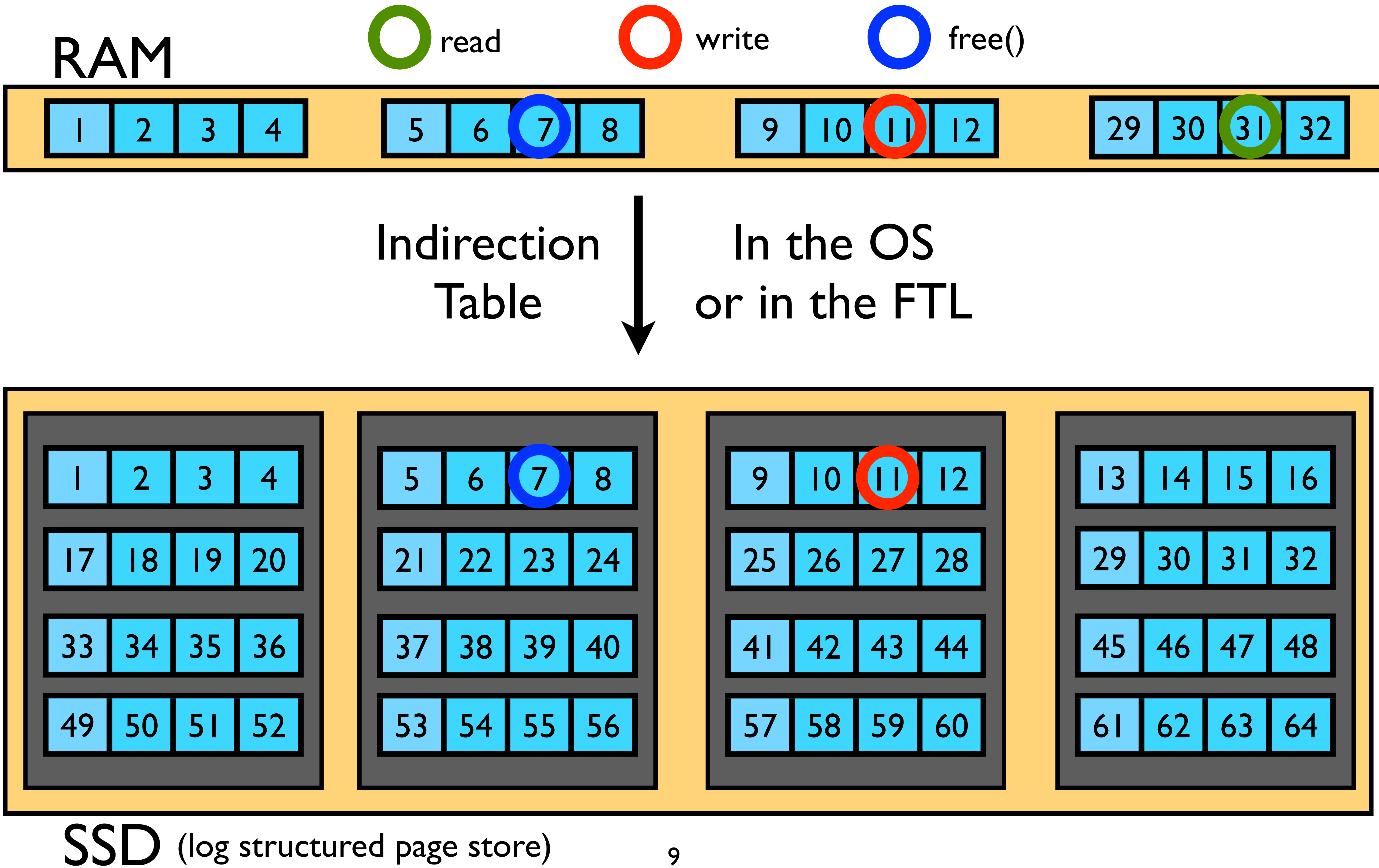
Transparent Tiering Today



Transparent Tiering Today



Transparent Tiering Today



Non-Transparent Tiering

Non-Transparent Tiering

- Redesign application to be flash aware
  - Custom object store with custom pointers
  - Reads, writes and garbage collection at an application object granularity
  - Avoid in-place writes (objects could be small)
  - Obtain the best performance and lifetime from flash memory device

Non-Transparent Tiering

- Redesign application to be flash aware
  - Custom object store with custom pointers
  - Reads, writes and garbage collection at an application object granularity
  - Avoid in-place writes (objects could be small)
  - Obtain the best performance and lifetime from flash memory device
  - Intrusive modifications needed
  - Expertise with flash memory needed

Non-Transparent Tiering

malloc
+
SSD-swap

```
MyObject* obj = malloc( sizeof( MyObject ) );

obj->x = 0;
obj->y = 1;
obj->z = 2;
free( obj );
```

Application Rewrite

```
MyObjectID oid = createObject( sizeof( MyObject ) );
MyObject* obj = malloc( sizeof( MyObject ) );

readObject( oid, obj );
obj->x = 0;
obj->y = 1;
obj->z = 2;
writeObject( oid, obj );
free( obj );
```

Our Goal

Our Goal

- Run mostly unmodified applications
  - Work via memory allocators in C-style programs

Our Goal

- Run mostly unmodified applications
  - Work via memory allocators in C-style programs
- Use the DRAM effectively
  - Use it as an object cache (not as a page cache)

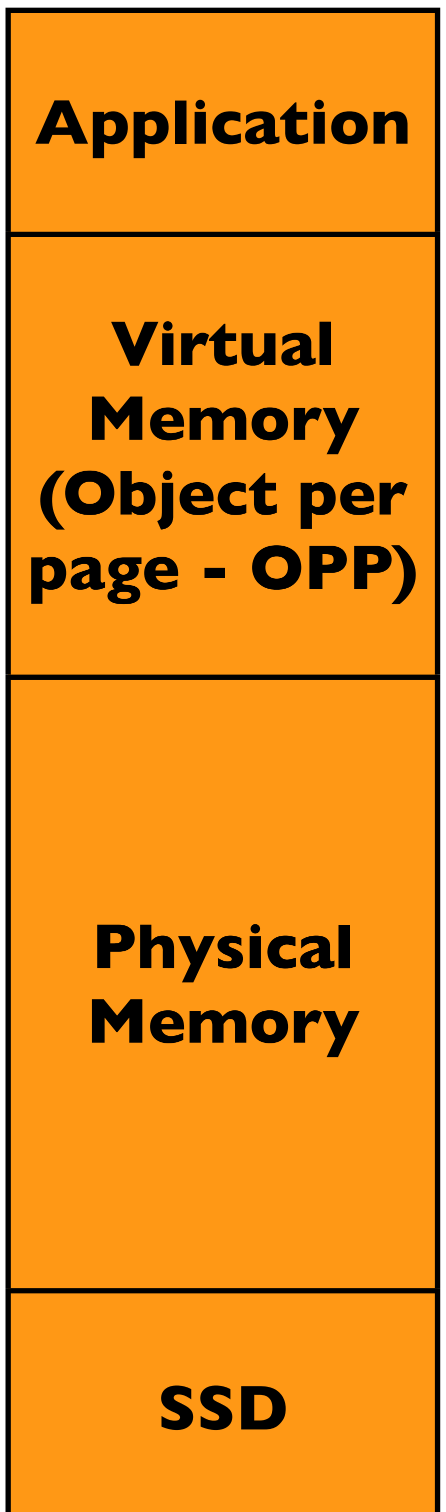
Our Goal

- Run mostly unmodified applications
  - Work via memory allocators in C-style programs
- Use the DRAM effectively
  - Use it as an object cache (not as a page cache)
- Use the SSD wisely
  - As a log-structured object store

Our Goal

- Run mostly unmodified applications
  - Work via memory allocators in C-style programs
- Use the DRAM effectively
  - Use it as an object cache (not as a page cache)
- Use the SSD wisely
  - As a log-structured object store
- Reorganize virtual memory allocation to discern object information

SSDAlloc Overview



SSDAlloc Overview

Application

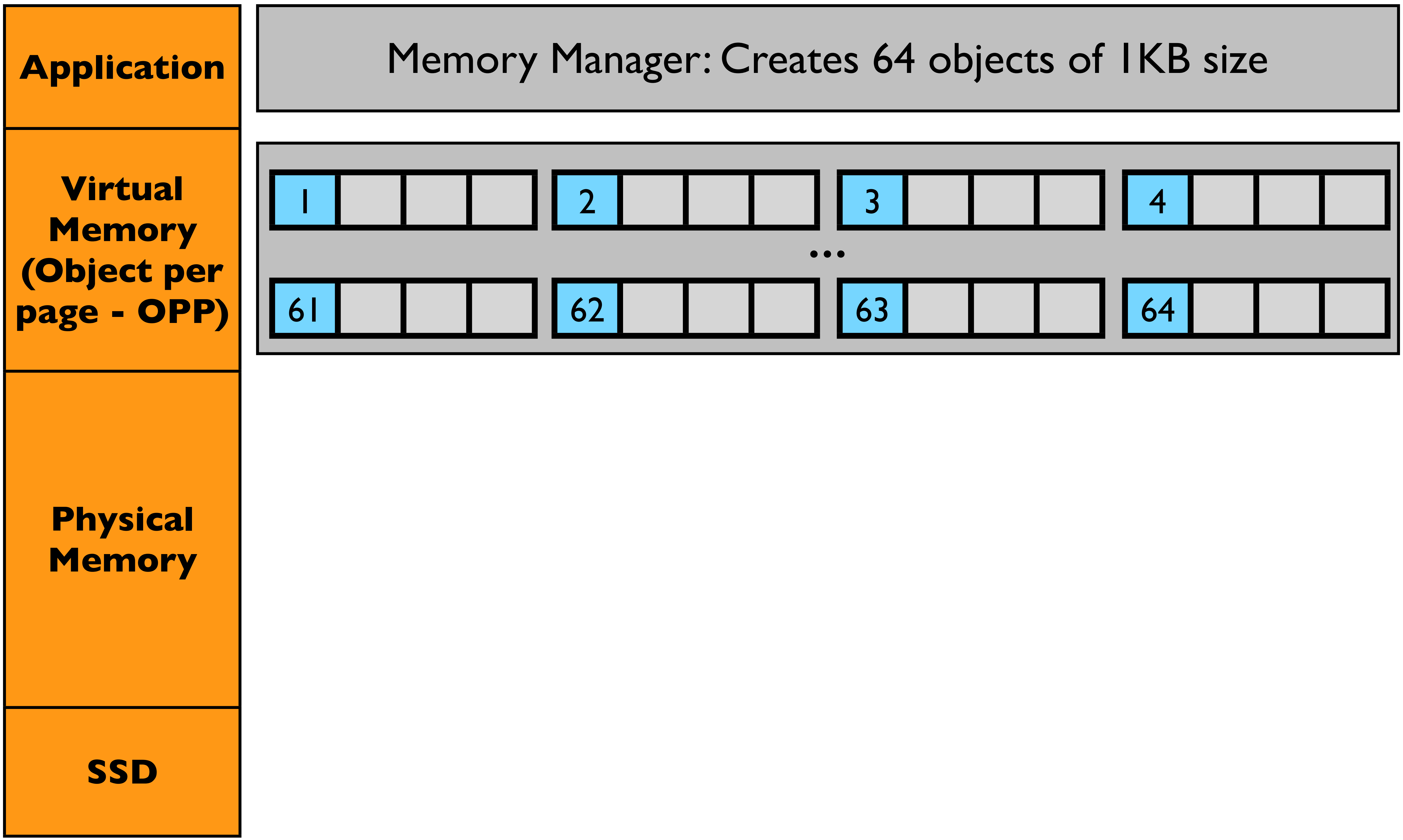
Virtual Memory (Object per page - OPP)

Physical Memory

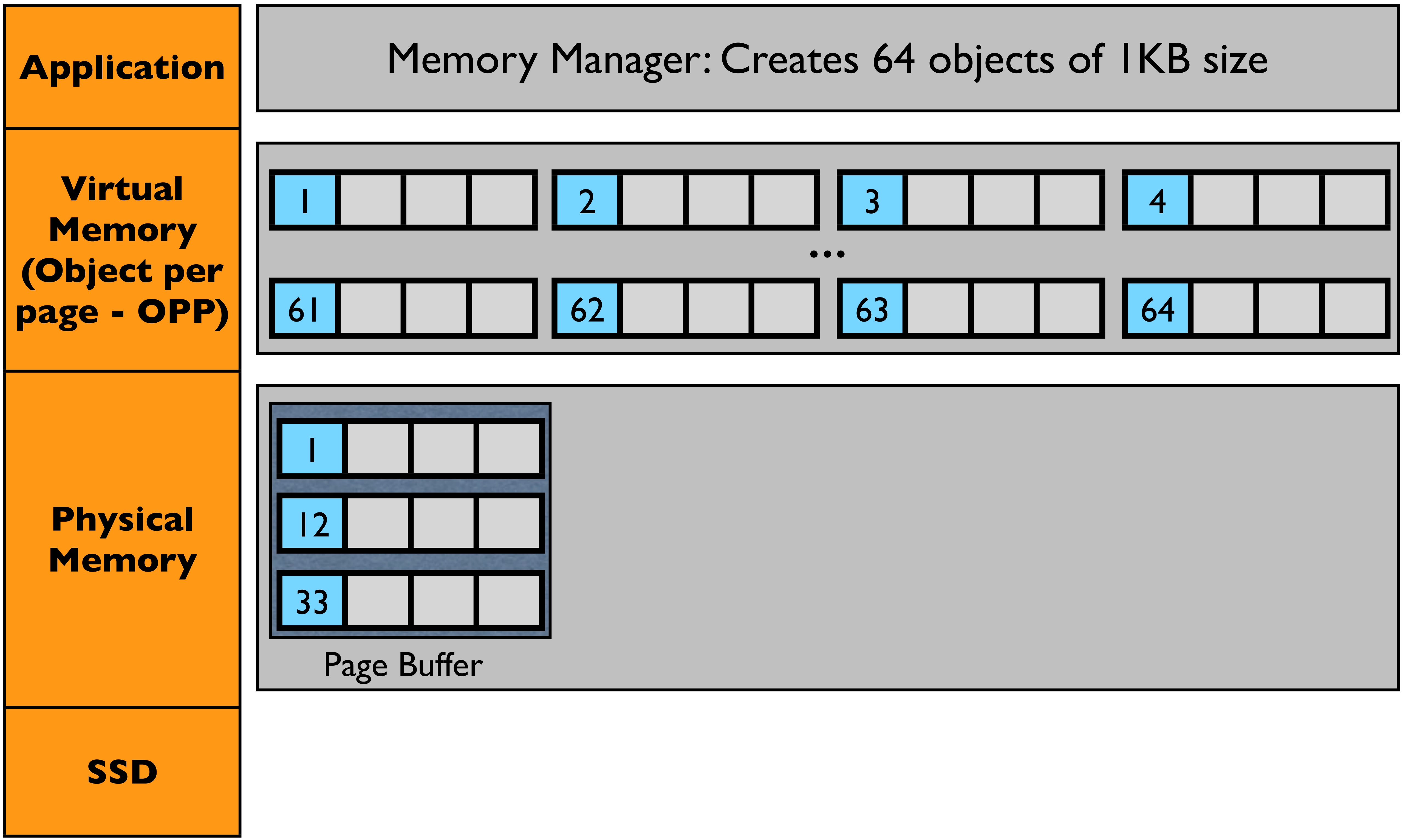
SSD

Memory Manager: Creates 64 objects of 1KB size

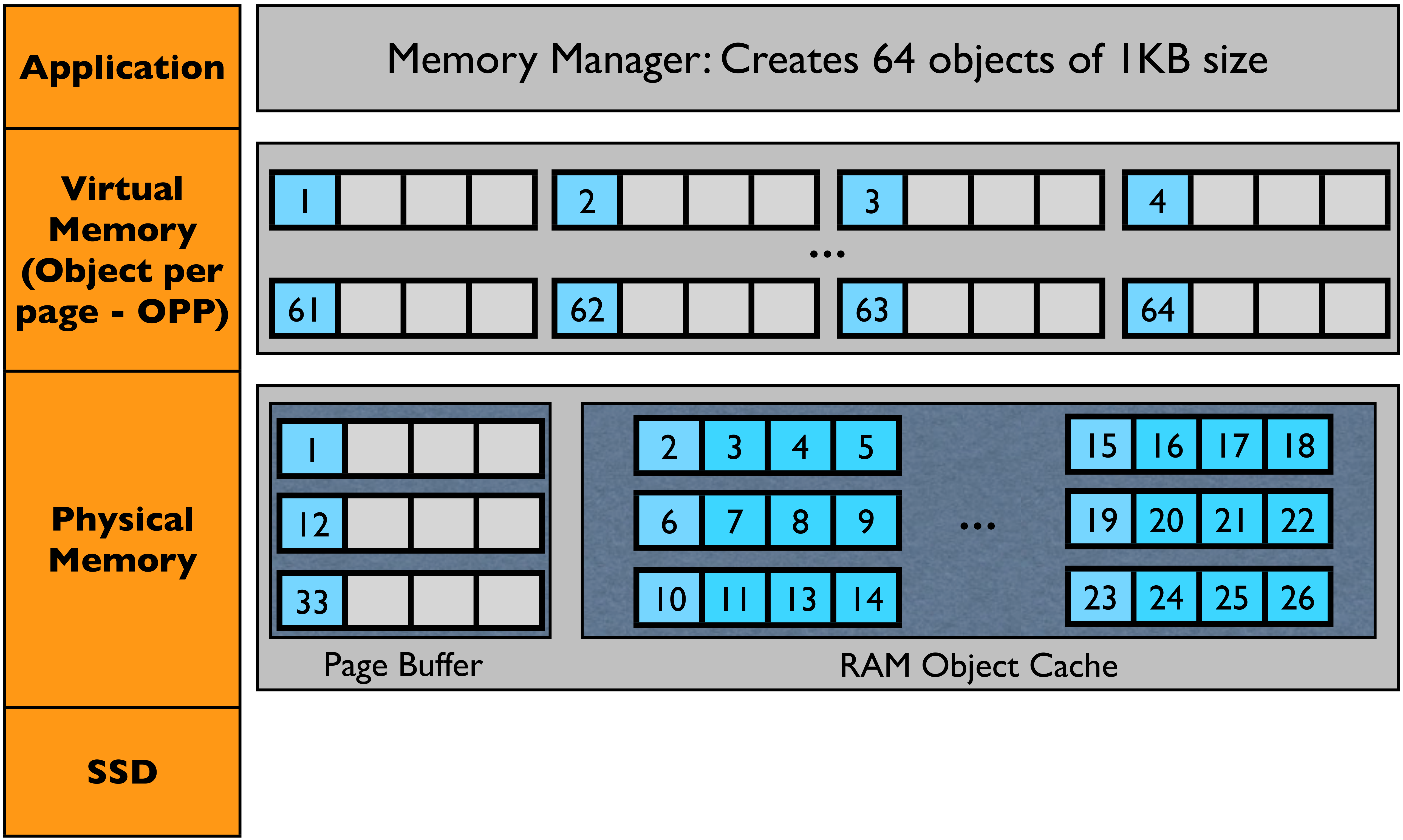
SSDAlloc Overview



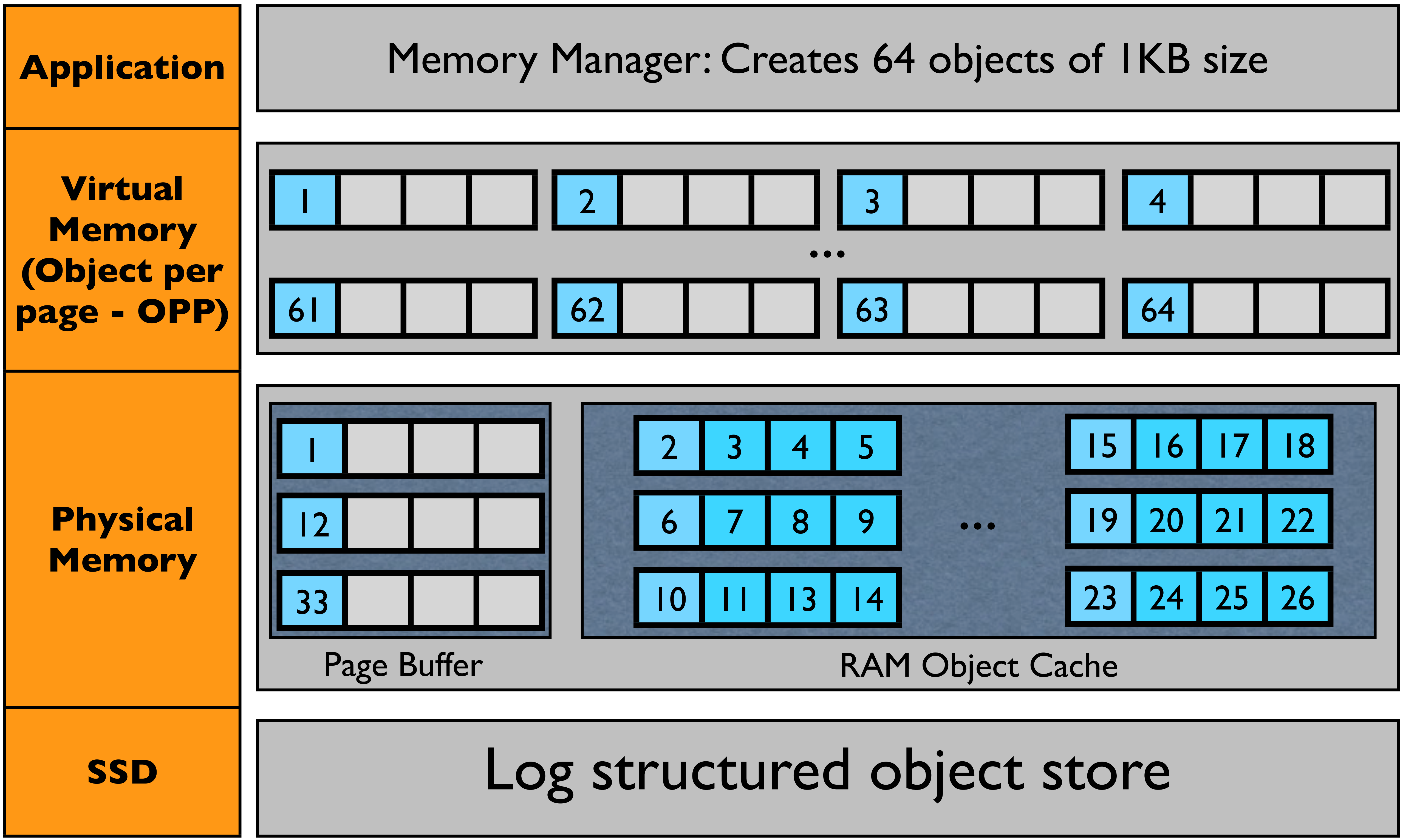
SSDAlloc Overview



SSDAlloc Overview



SSDAlloc Overview



SSDAlloc Options

SSDAlloc Options

	Object Per Page (OPP)	Memory Page (MP)
Data Entity	Application Defined Objects	4KB objects (like pages)

SSDAlloc Options

	Object Per Page (OPP)	Memory Page (MP)
Data Entity	Application Defined Objects	4KB objects (like pages)
Memory Manager	Pool Allocator	Coalescing Allocator

SSDAlloc Options

	Object Per Page (OPP)	Memory Page (MP)
Data Entity	Application Defined Objects	4KB objects (like pages)
Memory Manager	Pool Allocator	Coalescing Allocator
Virtual Memory	No. of objects * page_size	No. of pages * page_size

SSDAlloc Options

	Object Per Page (OPP)	Memory Page (MP)
Data Entity	Application Defined Objects	4KB objects (like pages)
Memory Manager	Pool Allocator	Coalescing Allocator
Virtual Memory	No. of objects * page_size	No. of pages * page_size
Physical Memory	Separate Page Buffer & RAM Object Cache	No such separation

SSDAlloc Options

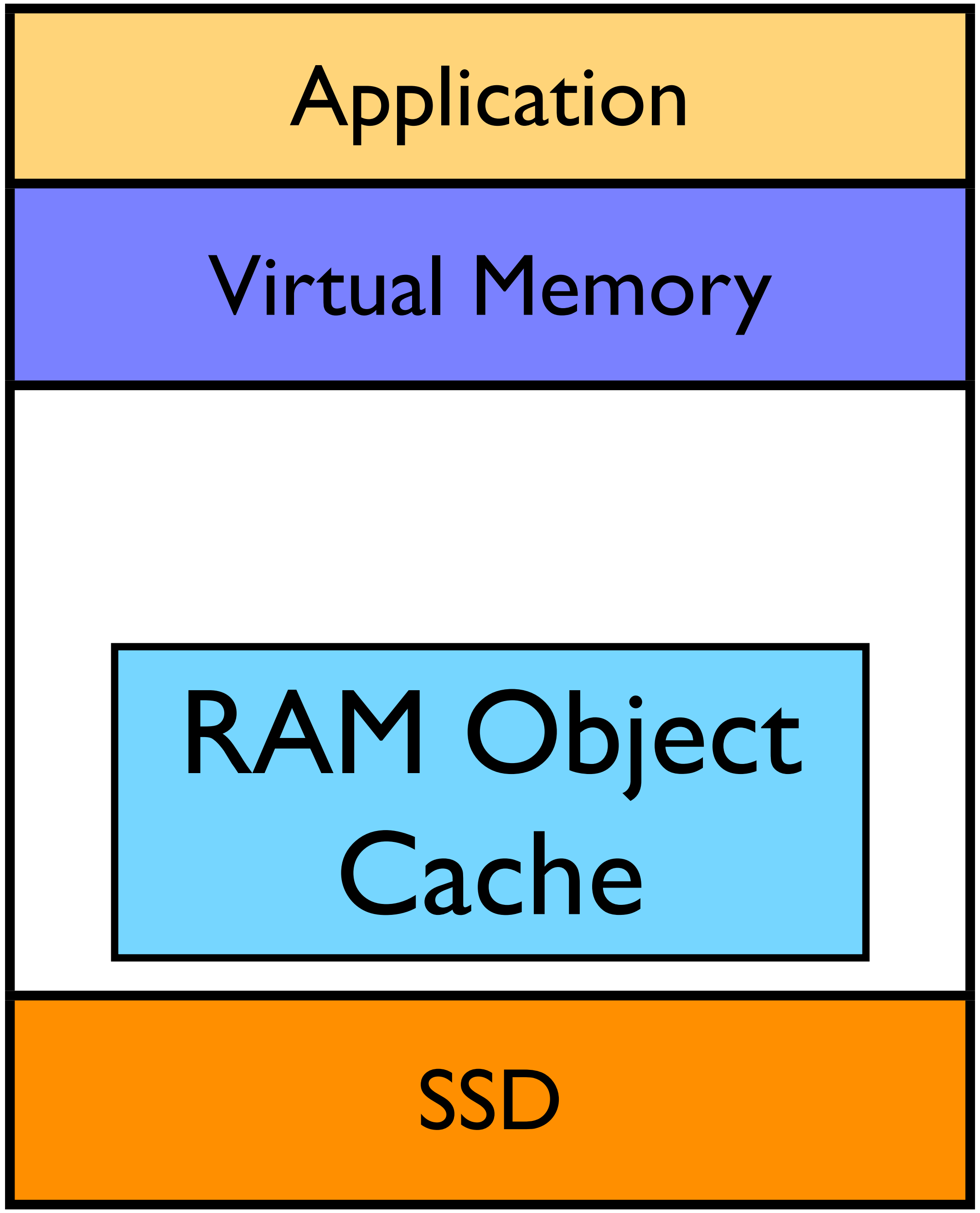
	Object Per Page (OPP)	Memory Page (MP)
Data Entity	Application Defined Objects	4KB objects (like pages)
Memory Manager	Pool Allocator	Coalescing Allocator
Virtual Memory	No. of objects * page_size	No. of pages * page_size
Physical Memory	Separate Page Buffer & RAM Object Cache	No such separation
SSD Usage	Log-structured Object Store	Log-structured Page Store

SSDAlloc Options

	Object Per Page (OPP)	Memory Page (MP)
Data Entity	Application Defined Objects	4KB objects (like pages)
Memory Manager	Pool Allocator	Coalescing Allocator
Virtual Memory	No. of objects * page_size	No. of pages * page_size
Physical Memory	Separate Page Buffer & RAM Object Cache	No such separation
SSD Usage	Log-structured Object Store	Log-structured Page Store
Code Changes	Minimal changes restricted to memory allocation	No changes needed

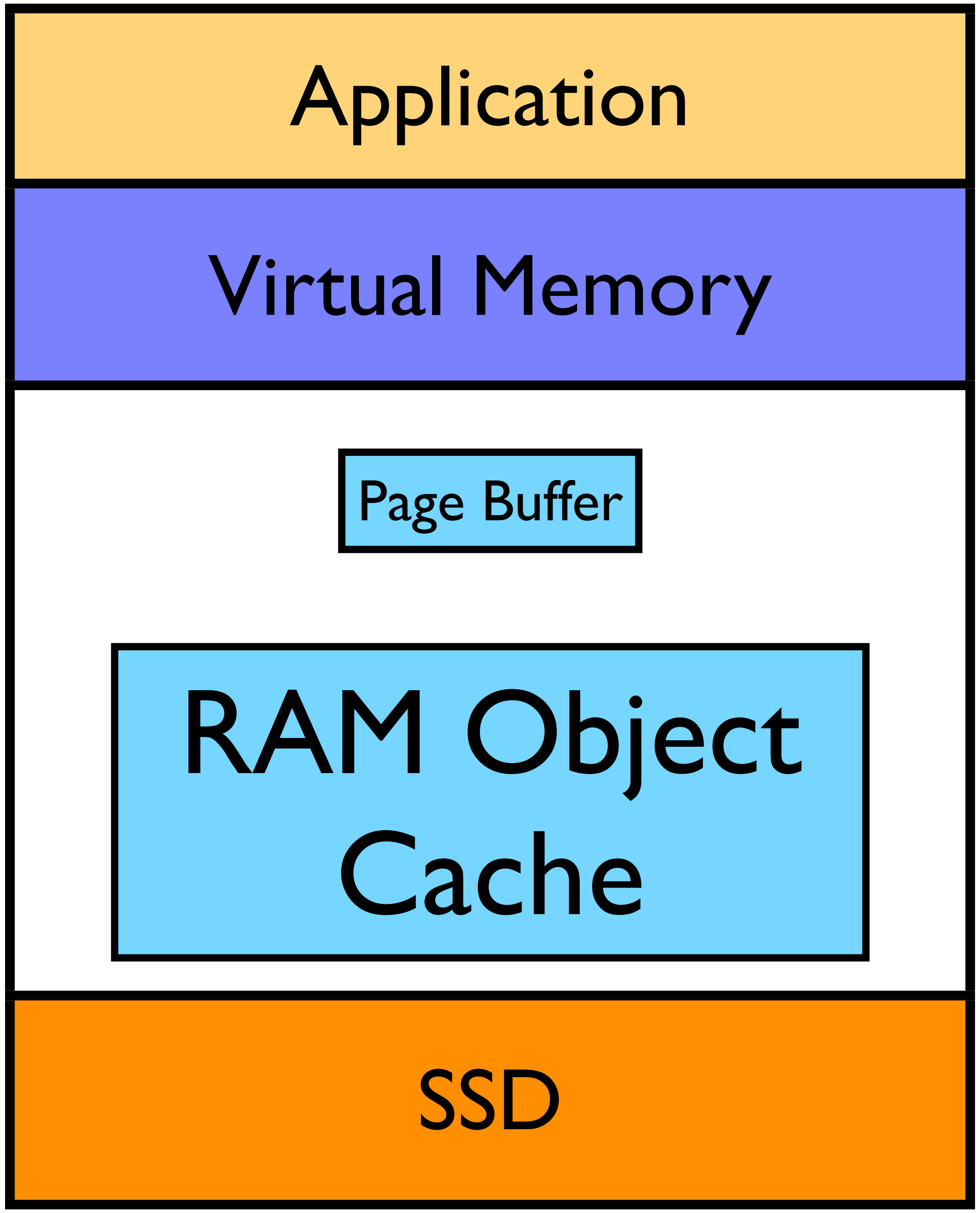
SSDAlloc Overview

SSDAlloc Overview



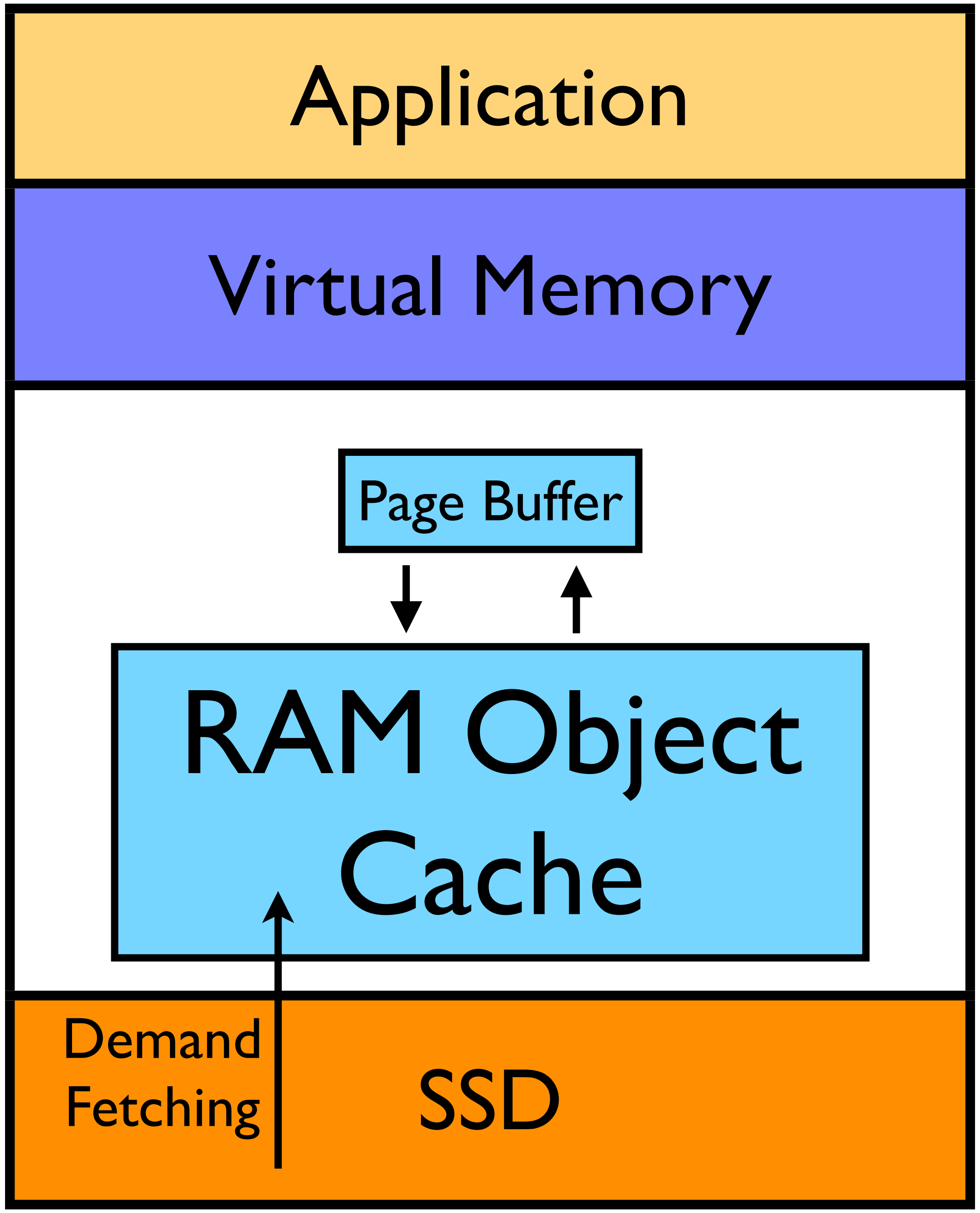
SSDAlloc Overview

- A small set of pages in core



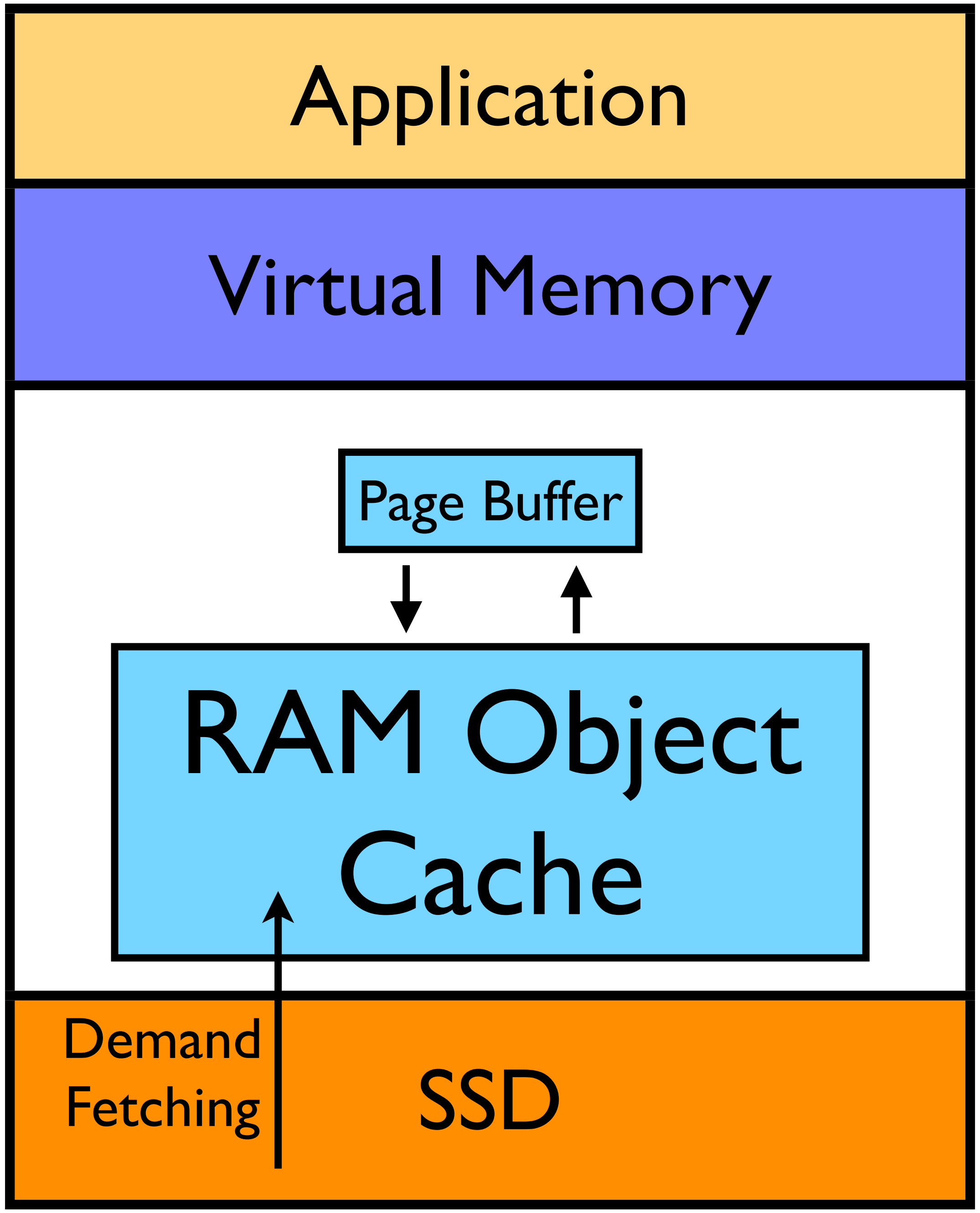
SSDAlloc Overview

- A small set of pages in core
  - Pages materialized on demand from RAM object cache/SSD
  - Restricted in size to minimize RAM wastage (from OPP)



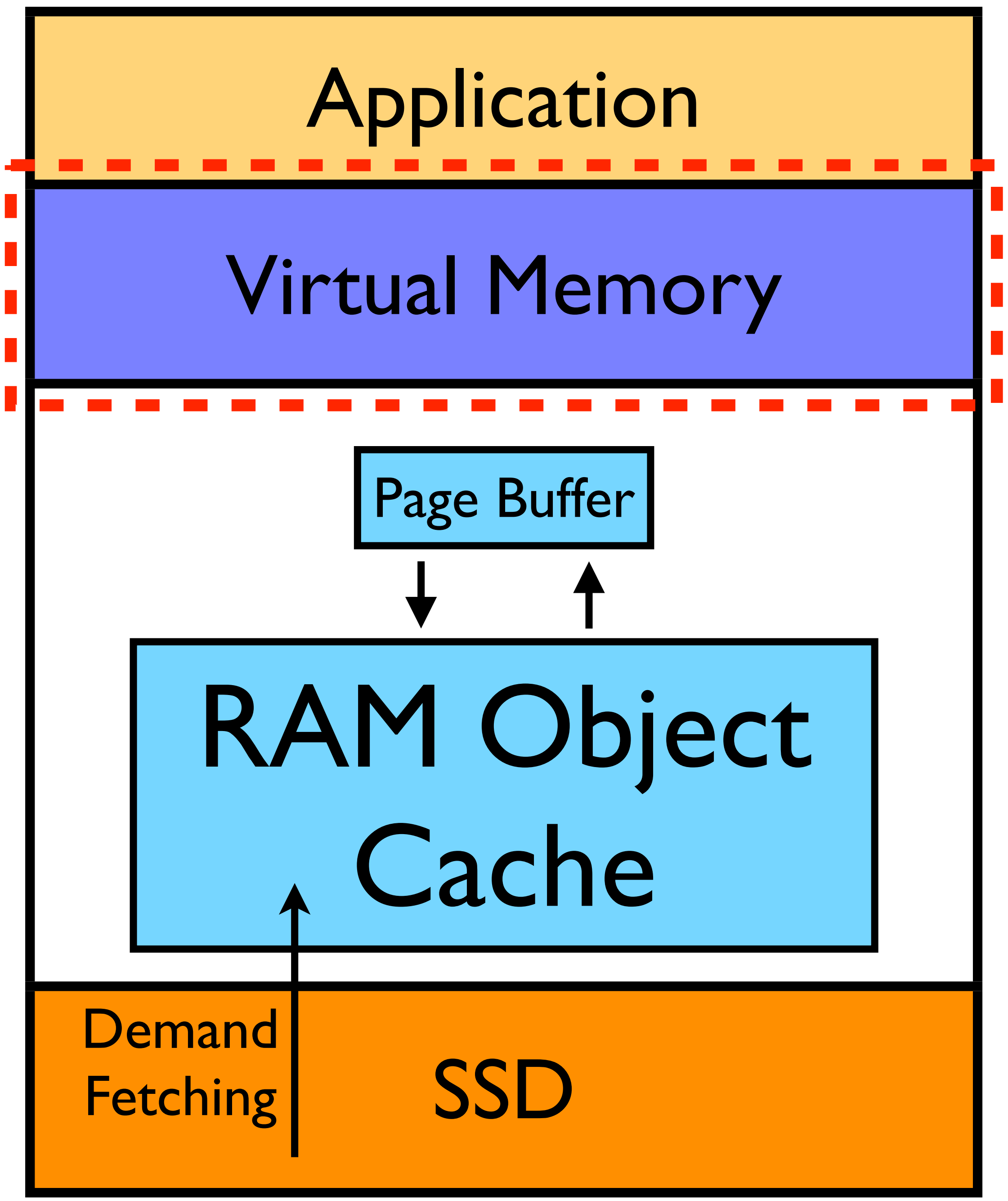
SSDAlloc Overview

- A small set of pages in core
  - Pages materialized on demand from RAM object cache/SSD
  - Restricted in size to minimize RAM wastage (from OPP)
- Implemented using *mprotect*



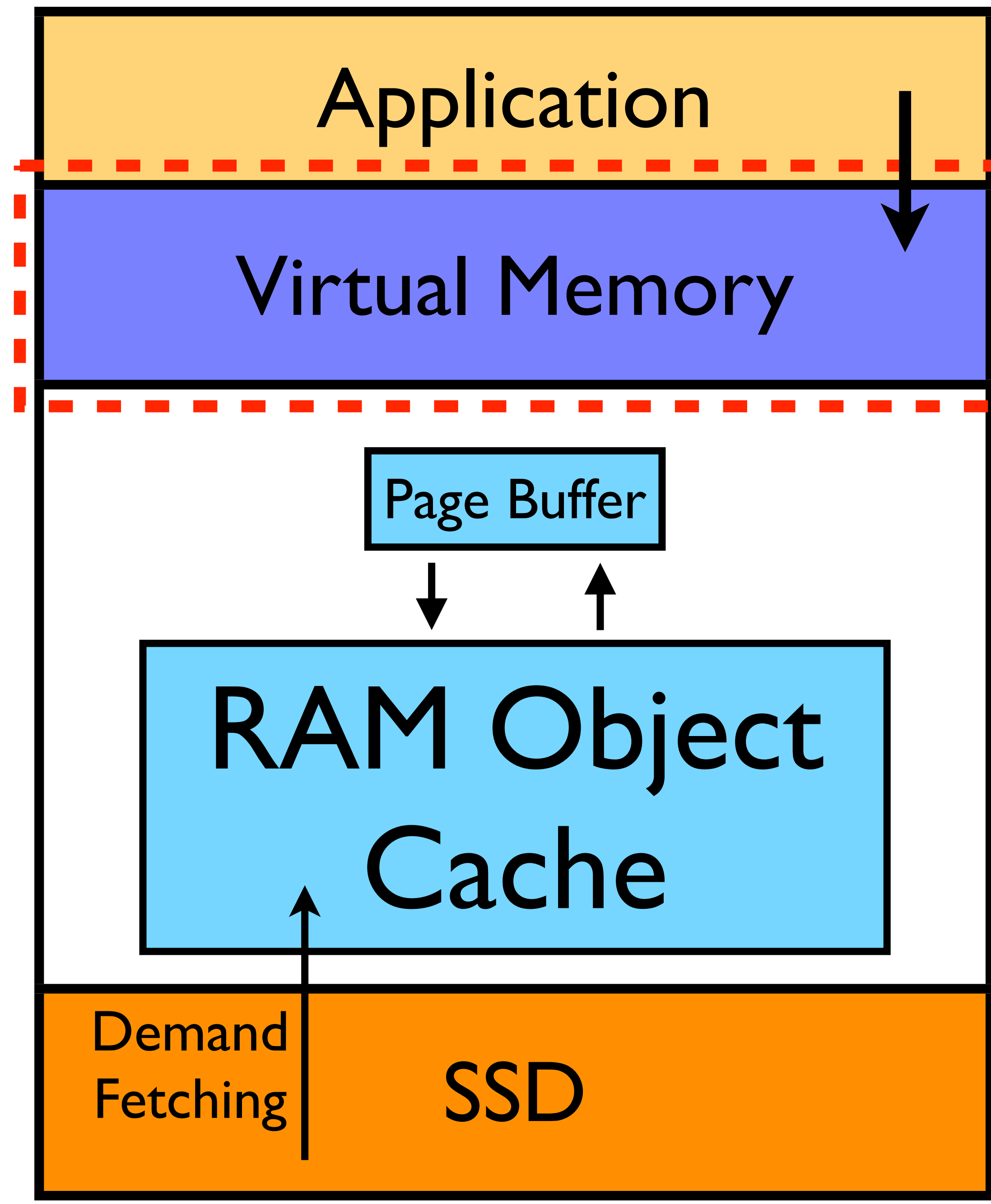
SSDAlloc Overview

- A small set of pages in core
  - Pages materialized on demand from RAM object cache/SSD
  - Restricted in size to minimize RAM wastage (from OPP)
- Implemented using *mprotect*



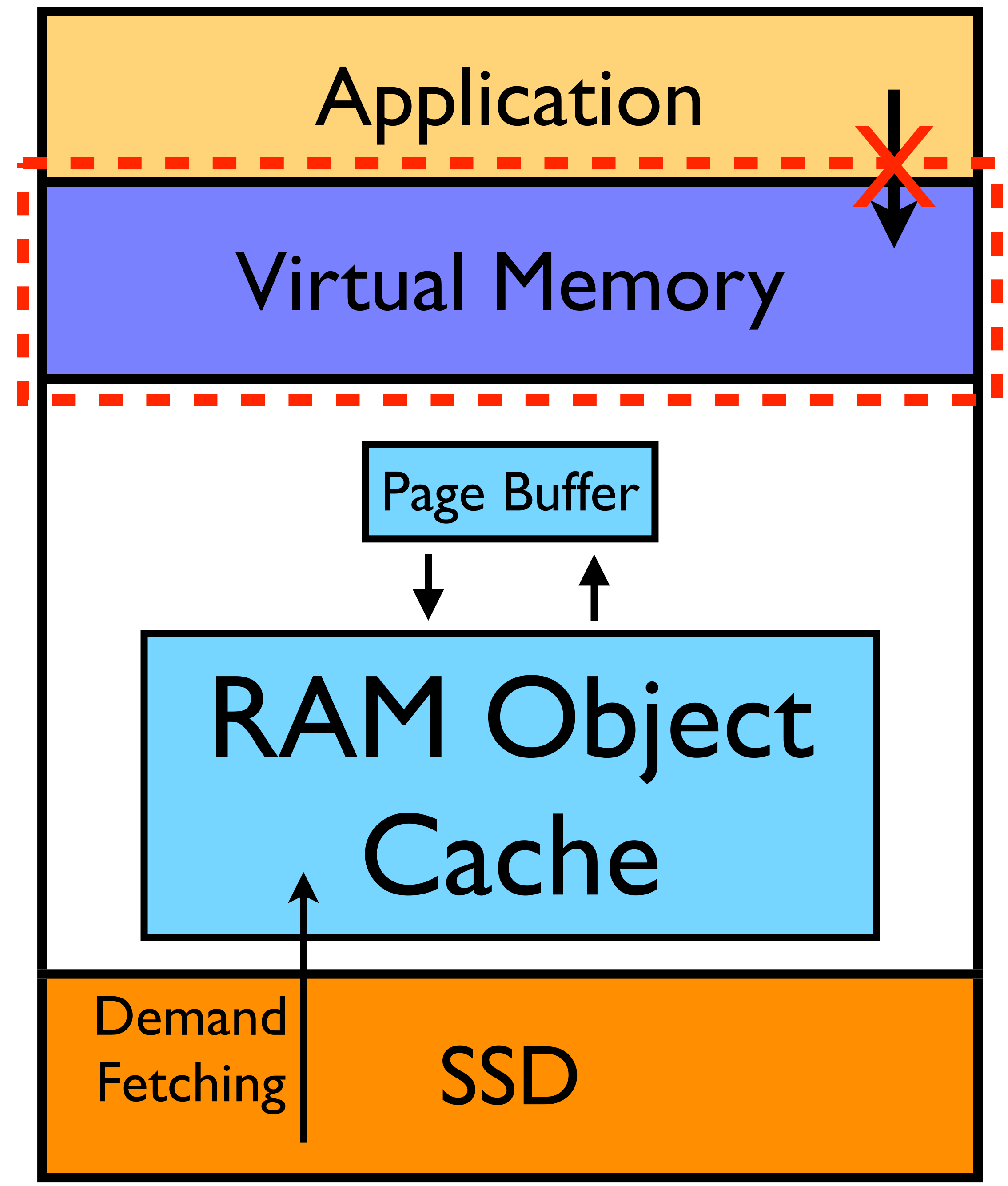
SSDAlloc Overview

- A small set of pages in core
  - Pages materialized on demand from RAM object cache/SSD
  - Restricted in size to minimize RAM wastage (from OPP)
- Implemented using *mprotect*



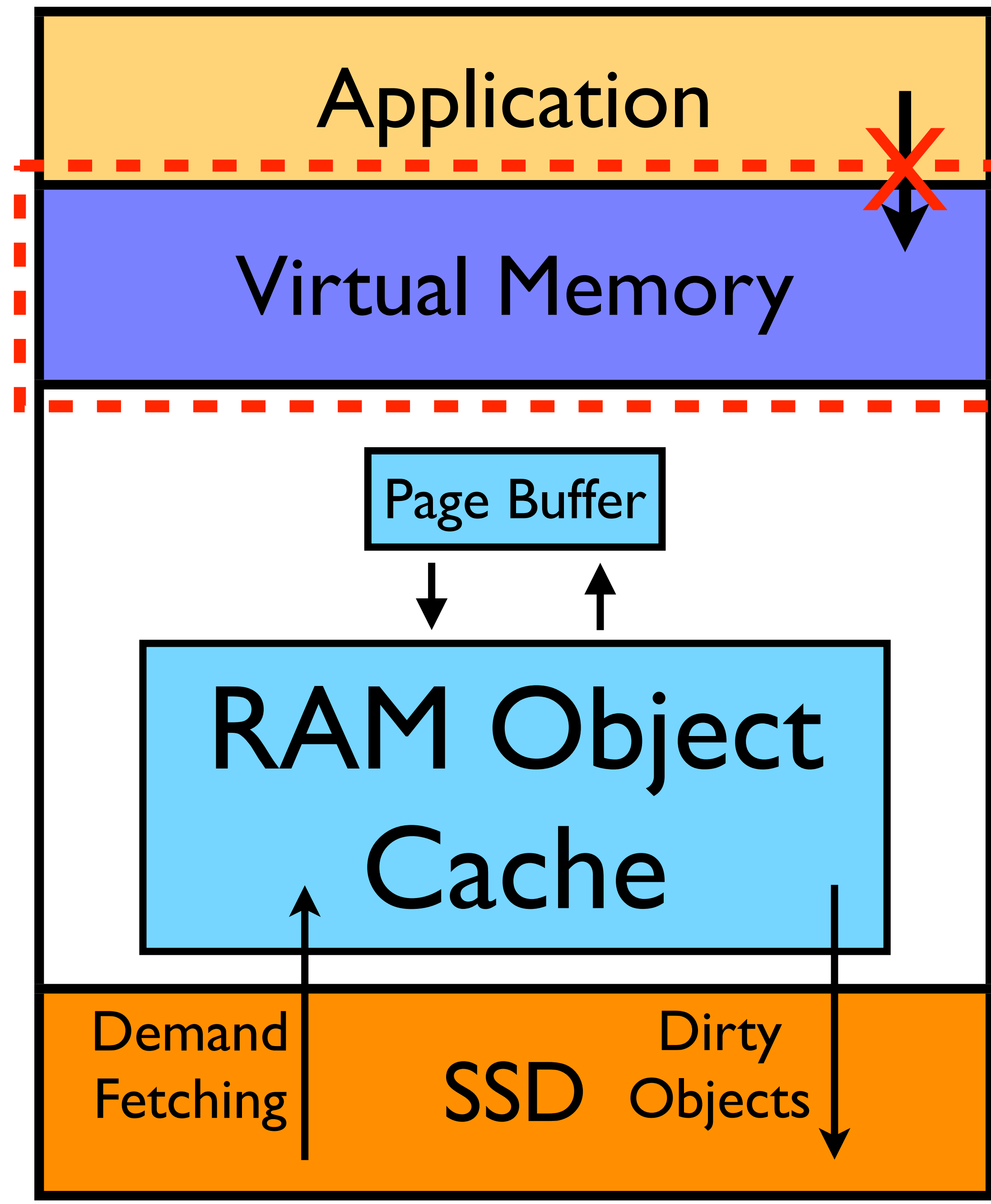
SSDAlloc Overview

- A small set of pages in core
  - Pages materialized on demand from RAM object cache/SSD
  - Restricted in size to minimize RAM wastage (from OPP)
- Implemented using *mprotect*
  - Page materialized in seg-fault handler



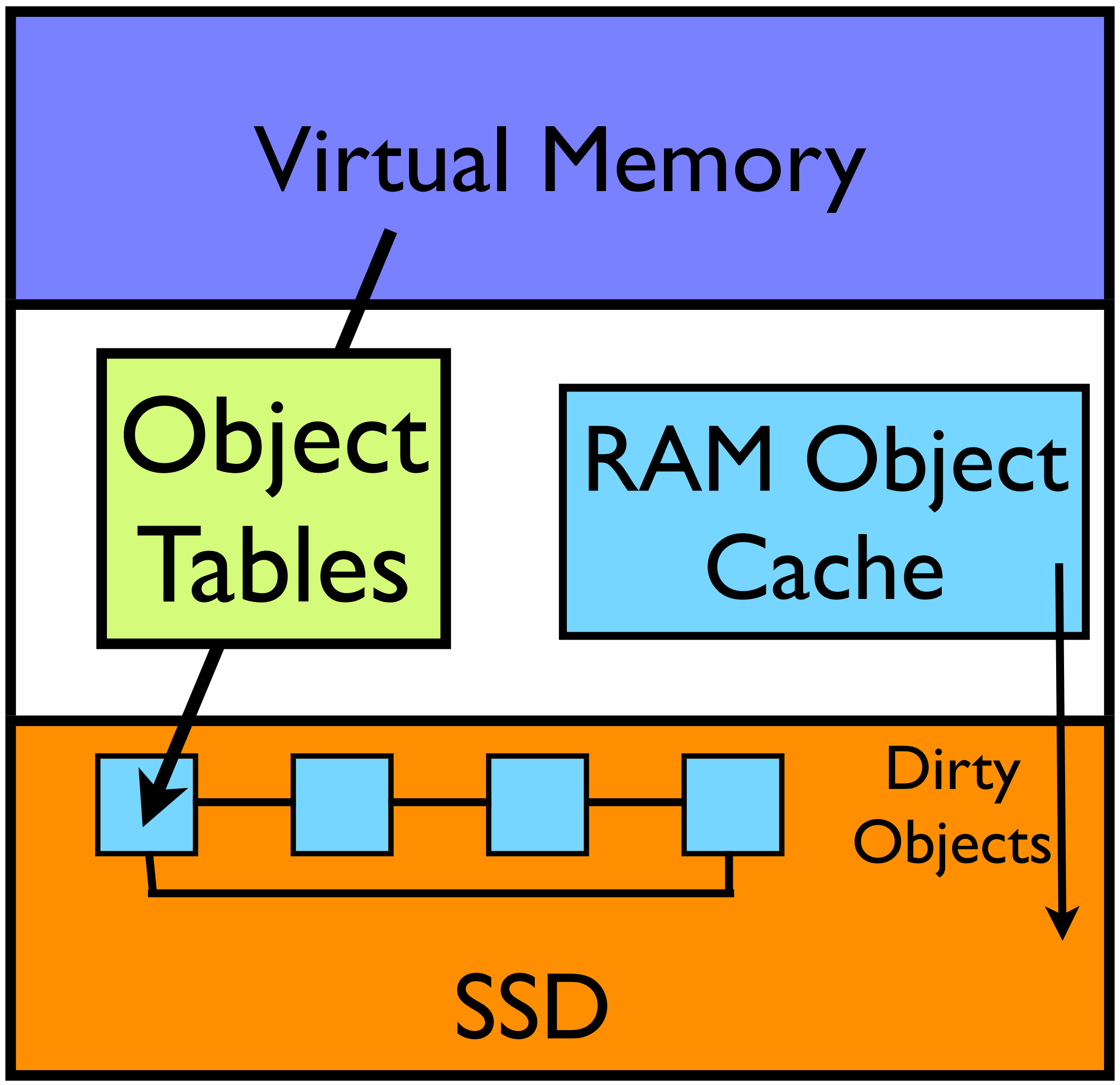
SSDAlloc Overview

- A small set of pages in core
  - Pages materialized on demand from RAM object cache/SSD
  - Restricted in size to minimize RAM wastage (from OPP)
- Implemented using *mprotect*
  - Page materialized in seg-fault handler
- RAM Object Cache continuously flushes dirty objects to the SSD in LRU order



SSD Maintenance

SSD Maintenance



SSD Maintenance

SSD Maintenance

- Copy-and-compact garbage-collector/log-writer
  - Seek optimizations not needed

SSD Maintenance

- Copy-and-compact garbage-collector/log-writer
  - Seek optimizations not needed
- Read at the head and write live and dirty objects
  - Use Object Tables to determine liveness

SSD Maintenance

- Copy-and-compact garbage-collector/log-writer
  - Seek optimizations not needed
- Read at the head and write live and dirty objects
  - Use Object Tables to determine liveness
- Garbage is disposed
  - Objects written elsewhere are garbage
  - OPP object which is "free" is garbage

Implementation

Implementation

- 11,000 lines of C++ code (runtime library)
  - Implemented using *mprotect, mmap, and madvise*
  - *SSDAlloc-OPP* pool and array allocator
  - *SSDAlloc-MP* coalescing allocator (array allocations)
  - *SSDFree* frees the allocated data
  - Can coexist with malloc pointers

SSD Usage Techniques

SSD Usage Techniques

Technique	Write Logging	Access < 4KB	Finegrained GC	Avoid DRAM Pollution	High Performance	Programming Ease
SSD Swap						✔
SSD Swap (Write Logged)	✔					✔
Application Rewrite	✔	✔	✔	✔	✔	

SSD Usage Techniques

Technique	Write Logging	Access < 4KB	Finegrained GC	Avoid DRAM Pollution	High Performance	Programming Ease
SSD Swap						✔
SSD Swap (Write Logged)	✔					✔
Application Rewrite	✔	✔	✔	✔	✔	
SSDAlloc	✔	✔	✔	✔	✔	✔

SSDAlloc Runtime Overhead

SSDAlloc Runtime Overhead

- Overhead for SSDAlloc runtime intervention

Overhead Source	Max Latency
TLB Miss (DRAM Read)	0.014 µSec
Object Table Lookup	0.046 µSec
Page Materialization	0.138 µSec
Page Dematerialization	0.172 µSec
Signal Handling	0.666 µSec
Combined Overhead	0.833 µSec

SSDAlloc Runtime Overhead

- Overhead for SSDAlloc runtime intervention

Overhead Source	Max Latency
TLB Miss (DRAM Read)	0.014 μSec
Object Table Lookup	0.046 μSec
Page Materialization	0.138 μSec
Page Dematerialization	0.172 μSec
Signal Handling	0.666 μSec
Combined Overhead	0.833 μSec

- NAND Flash latency ~ 30-50 μSec

SSDAlloc Runtime Overhead

- Overhead for SSDAlloc runtime intervention

Overhead Source	Max Latency
TLB Miss (DRAM Read)	0.014 µSec
Object Table Lookup	0.046 µSec
Page Materialization	0.138 µSec
Page Dematerialization	0.172 µSec
Signal Handling	0.666 µSec
Combined Overhead	0.833 µSec

- NAND Flash latency ~ 30-50 µSec
- Can reach 1 Million IOPS

Experiments

Experiments

- Comparing three allocation methods
  - *malloc* **replaced with** *SSDAlloc-OPP*
  - *malloc* **replaced with** *SSDAlloc-MP*
  - *Swap*

Experiments

- Comparing three allocation methods
  - *malloc* **replaced with** *SSDAlloc-OPP*
  - *malloc* **replaced with** *SSDAlloc-MP*
  - *Swap*
- 2.4Ghz Quadcore CPU with 16GB RAM
  - RiData, Kingston, Intel X25-E, Intel X25-V and Intel X25-M

Results Overview

Application	Original LOC	Modified LOC	SSDAlloc-OPP's gain vs	
			Swap	SSDAlloc-MP
Memcached	11,193	21	5.5 - 17.4x	1.4 - 3.5x
B+Tree Index	477	15	4.3 - 12.7x	1.4 - 3.2x
Packet Cache	1,540	9	4.8 - 10.1x	1.3 - 2.3x
HashCache	20,096	36	5.3 - 17.1x	1.3 - 3.3x

Results Overview

Application	Original LOC	Modified LOC	SSDAlloc-OPP's gain vs	
			Swap	SSDAlloc-MP
Memcached	11,193	21	5.5 - 17.4x	1.4 - 3.5x
B+Tree Index	477	15	4.3 - 12.7x	1.4 - 3.2x
Packet Cache	1,540	9	4.8 - 10.1x	1.3 - 2.3x
HashCache	20,096	36	5.3 - 17.1x	1.3 - 3.3x

- SSDAlloc applications write up to 32 times less data to the SSD than when compared to the traditional VM style applications

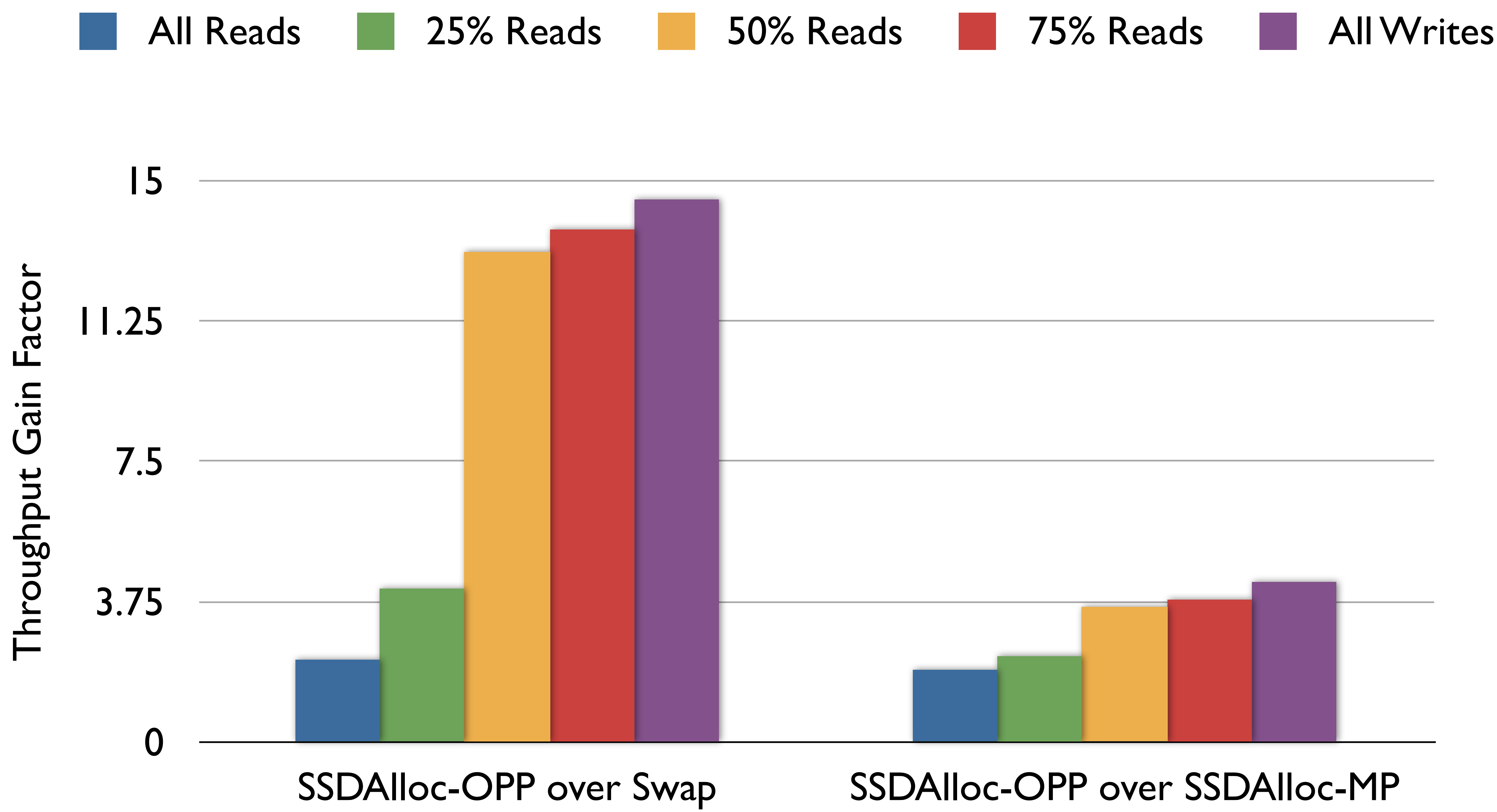
Microbenchmarks

Microbenchmarks

- 32GB array of 128 byte objects (32GB SSD, 2GB RAM)

Microbenchmarks

- 32GB array of 128 byte objects (32GB SSD, 2GB RAM)



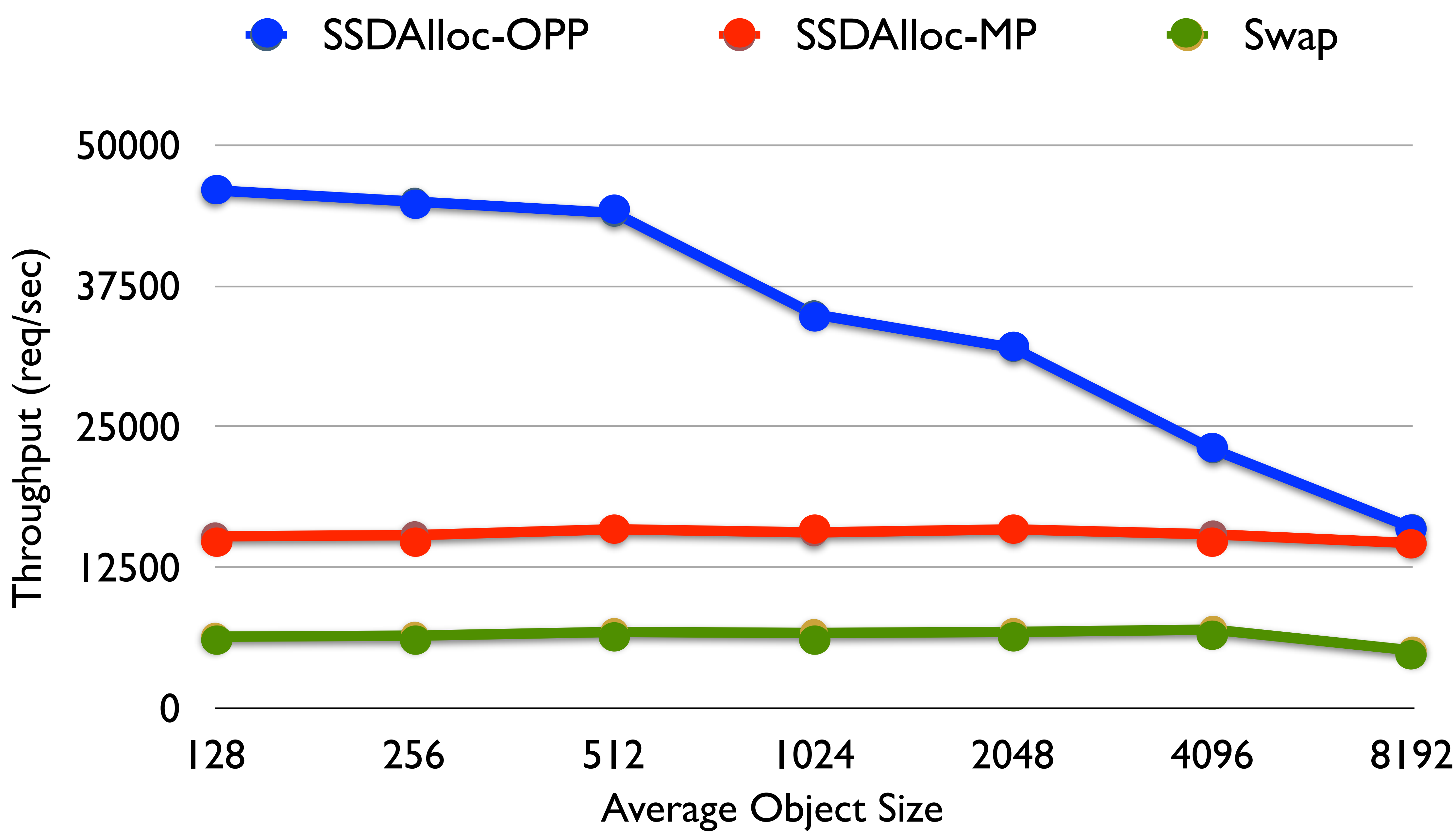
Memcached Benchmarks

Memcached Benchmarks

- 30GB SSD, 4GB RAM, 4 memcache clients
- Memcache server slab allocator modified to use SSDAlloc

Memcached Benchmarks

- 30GB SSD, 4GB RAM, 4 memcache clients
- Memcache server slab allocator modified to use SSDAlloc



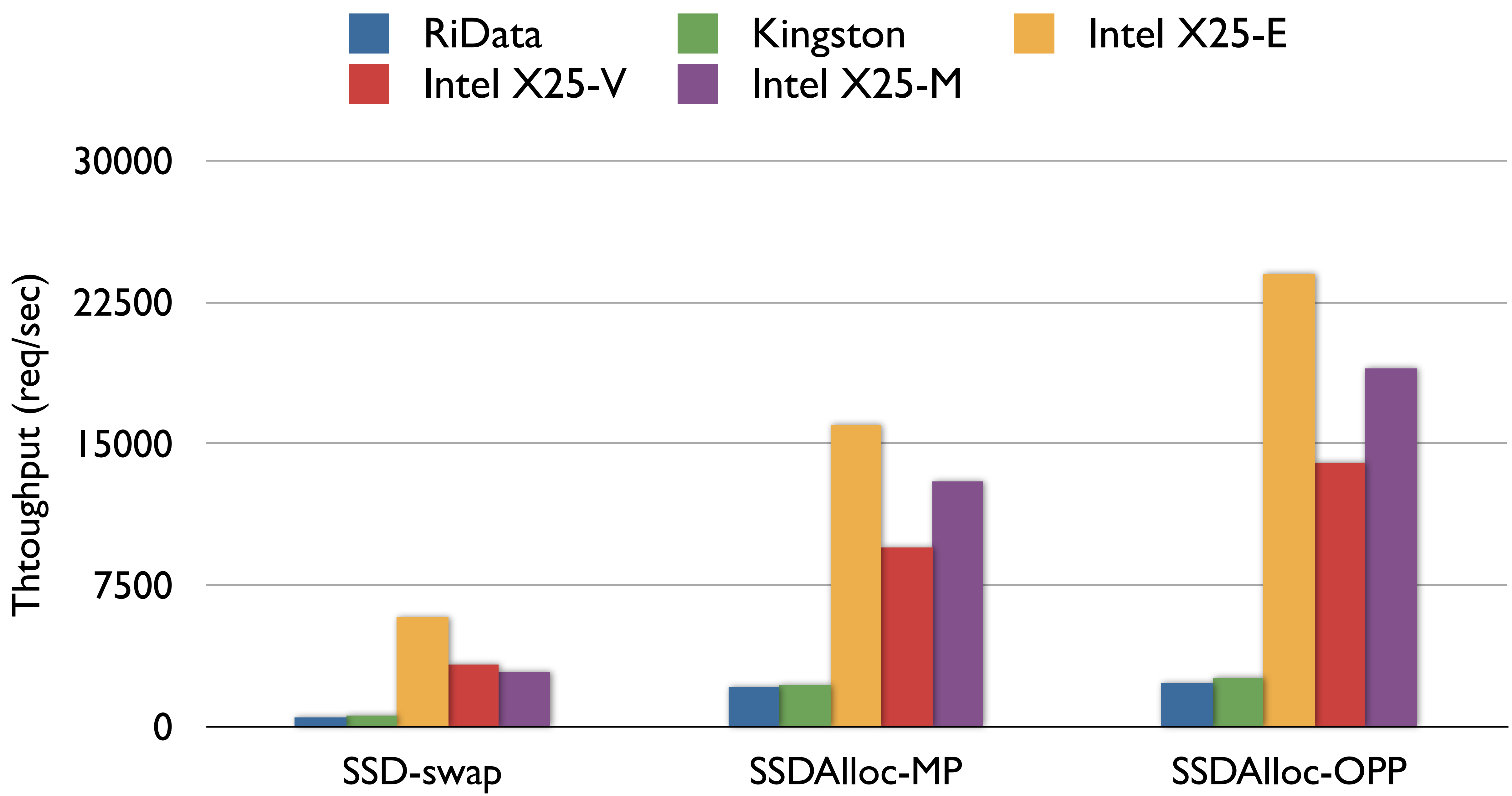
Memcached Benchmarks

Memcached Benchmarks

- Performance for 50% reads and 50% writes

Memcached Benchmarks

- Performance for 50% reads and 50% writes



Summary

Summary

- SSDAlloc migrates SSD naturally into VM system
  - RAM as a compact object cache
  - Virtual memory addresses are used
  - Only memory allocation code changes (9 to 36 LOC)
  - Other approaches need intrusive modifications

Summary

- SSDAlloc migrates SSD naturally into VM system
  - RAM as a compact object cache
  - Virtual memory addresses are used
  - Only memory allocation code changes (9 to 36 LOC)
  - Other approaches need intrusive modifications
- SSD as log-structured object store
  - Can obtain 90% raw SSD random read performance
  - Other transparent approaches deliver only 10--30%
  - Reduce write traffic by up to 32 times

Thanks

Anirudh Badam
abadam@cs.princeton.edu
http://tinyurl.com/ssdalloc