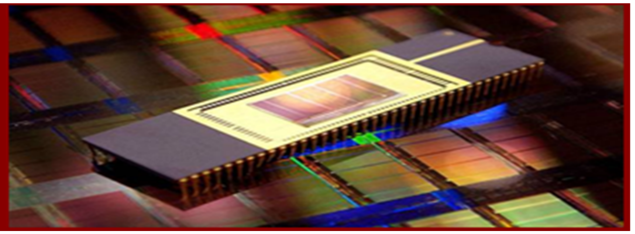
Operating System Supports for SCM as Main Memory Systems (Focusing on iBuddy)

2011. 4. 19
Jongmoo Choi
http://embedded.dankook.ac.kr/~choijm

NVRAMOS 2011 Spring
Operating System Support for
Next Generation Large Scale NVRAM
Organized by KIISE, April 18 - 20, 2011, Jeju, Korea

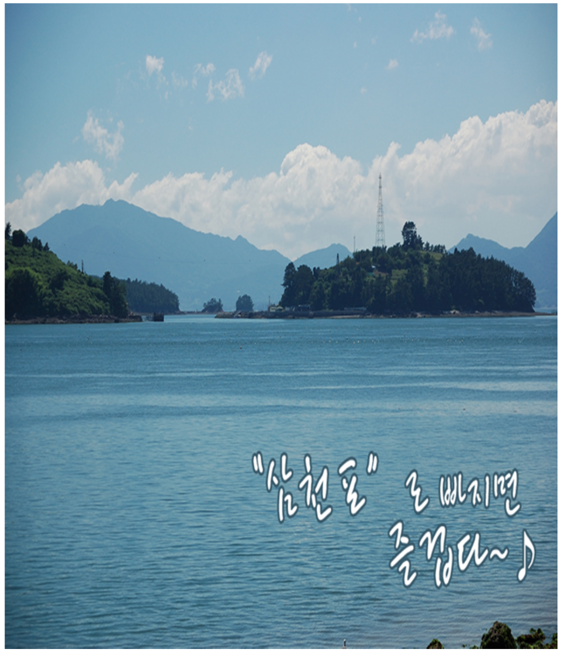


Contents

- Overview
- Motivation
- Observations
- Proposal: iBuddy (inverse Buddy)
- Performance Evaluation
- Conclusion

Overview

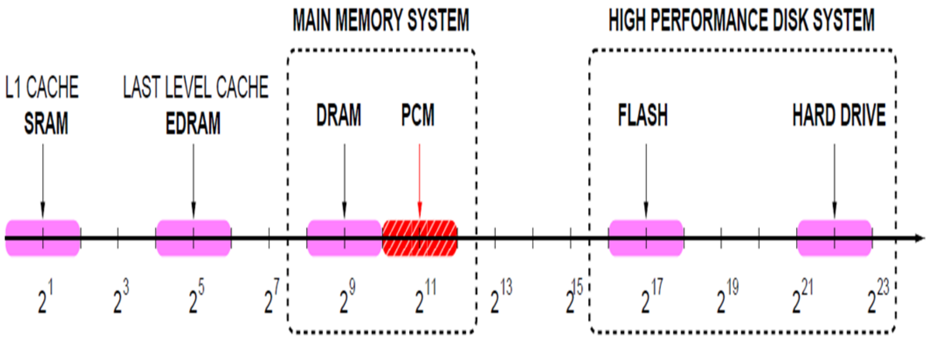
- Get sidetracked



Motivation

- SCM Introduction
  - ✓ Both DRAM and Storage Characteristics
    - Byte-addressable, Non-volatile
    - PRAM, MRAM, FRAM, RRAM, …
  - ✓ Technical Hurdles for using main memory
    - Performance
    - Endurance

NVRAM (or SCM)



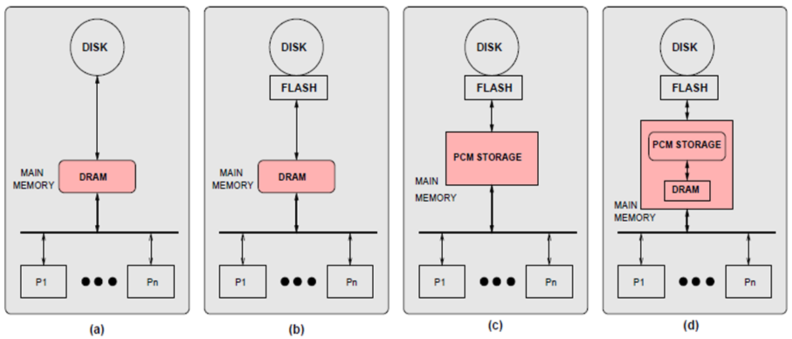
Typical Access Latency (in terms of processor cycles for a 4 GHz processor)

Parameter	DRAM	NAND Flash	NOR Flash	PCM
Density	1X	4X	0.25X	2X-4X
Read Latency	60ns	25 us	300 ns	200-300 ns
Write Speed	≈1 Gbps	2.4 MB/s	0.5 MB/s	≈100 MB/s
Endurance	N/A	10^4	10^4	10^6 to 10^8
Retention	Refresh	10yrs	10yrs	10 yrs

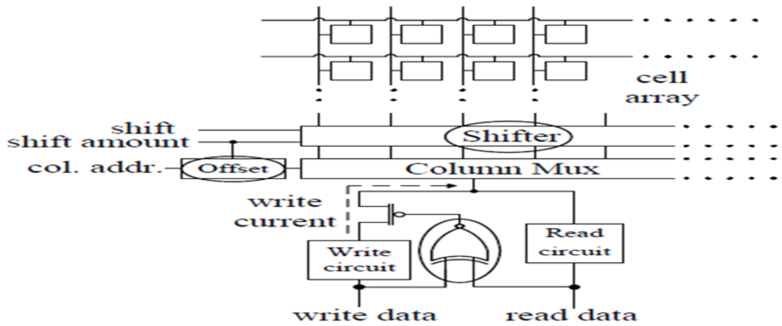
(Source: M. Qureshi et al., "Scalable High Performance Main Memory System Using Phase-Change Memory Technology", ISCA,09)

Motivation

- Previous research
  - M. Qureshi, et al., "Scalable high performance main memory system using phase-change memory technology", ISCA'09.
    - Hybrid main memory, Caching, Delayed writes, Line-level writes
  - P. Zhou et al., "A durable and energy efficient main memory using phase change memory technology", ISCA'09.
    - Removing redundant bit-writes, Row shifting and segment swap
  - B. Lee et al., "Architecturing phase change memory as a scalable dram alternative", ISCA'09.
    - Partial writes: track dirty data in CPU cache
  - A. Wang et al., "Conquest: Better performance through a disk/persistent-ram hybrid file system", USENIX'02.
  - J. Condit et al., "Better i/o through byte-addressable, persistent memory", SOSP'09.
  - A. Caulfield et al., "Moneta: A high-performance storage array architecture for next-generation, non-volatile memories", MICRO'10.



(Source: M. Qureshi's ISCA'09 paper)



(Source: P. Zhou's ISCA'09 paper)

Motivation

- Previous research
  - ✓ Mainly based on hardware-level approach

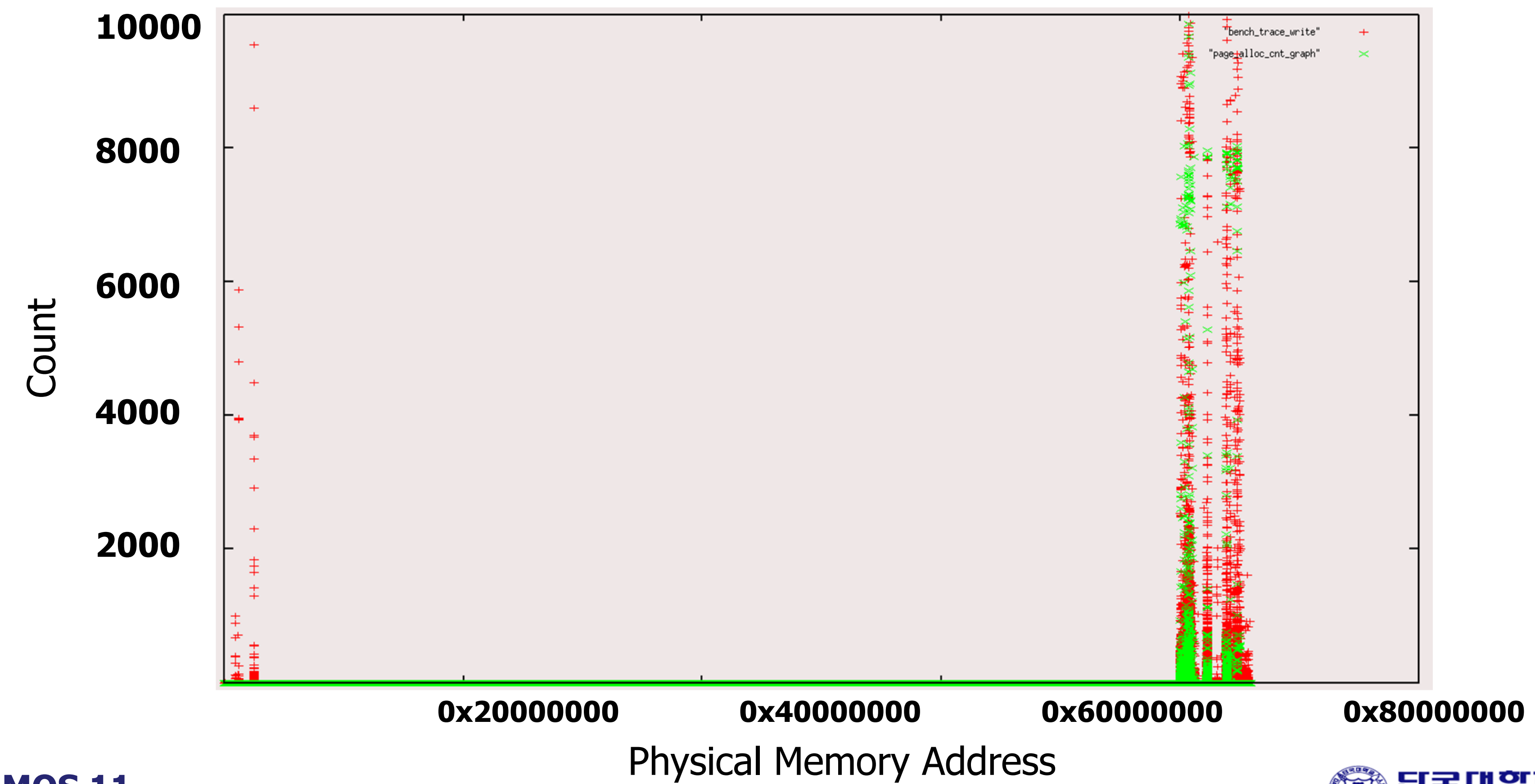
- Any feasible OS-level approach?
  - ✓ Focusing on endurance issue
  - ✓ Fair page frame allocation for wear-leveling
  - ✓ Instincts
    - Positive relation between allocation and write
    - Burst writes can be mitigated by CPU cache
    - Can obtain long term wear-leveling without keeping allocation counts per each page frame

Observations

- Page frame allocation and write distribution
  - ✓ Test environments: Intel 8 cores, 32GB DRAM, 450GB*10 Disks
  - ✓ OS: Linux 2.6.32
  - ✓ Workload: Unixbench

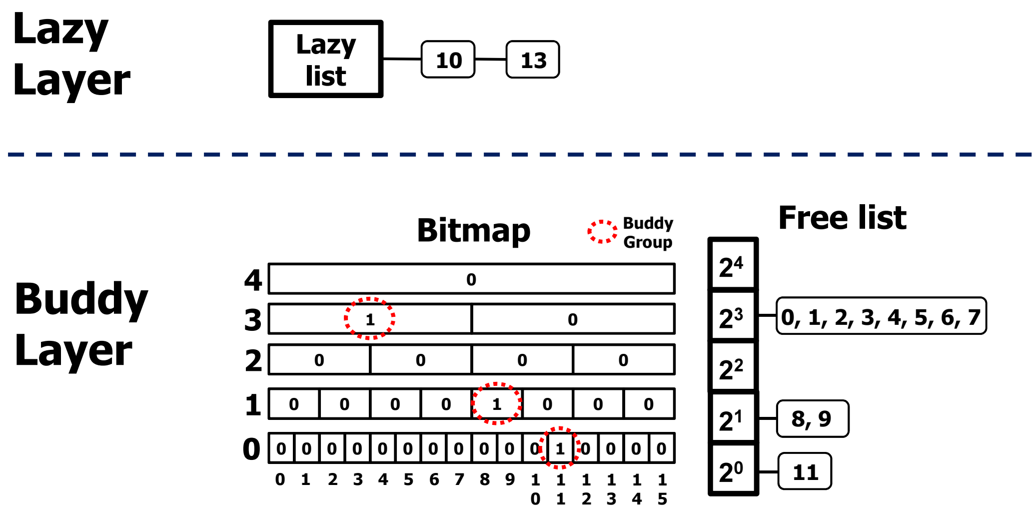
X : 페이지 할당 횟수

+ : 쓰기 연산 횟수



Observations

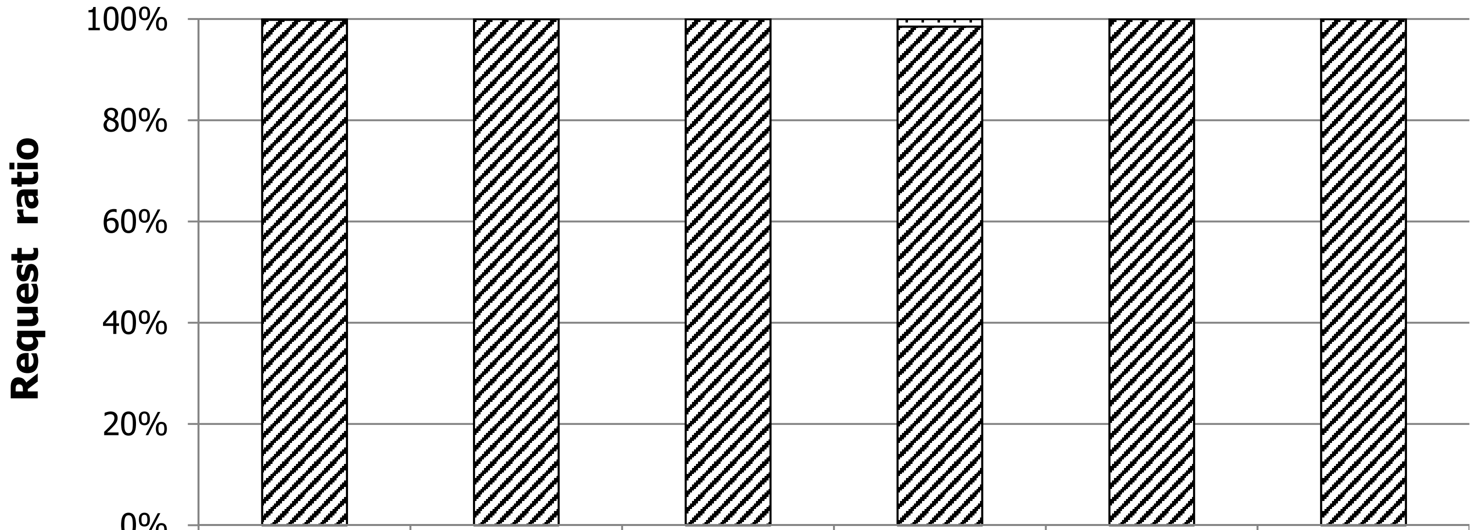
- Memory manager in Linux
  - ✓ Lazy buddy system
    - Re-allocate the recently freed page frames with higher probability
    - Lazy layer deteriorates unfairness
    - Group management makes it difficult to employ an allocation scheme based on allocation-counts of each page frame



☞ **Is it possible to manage each page frame individually for fair allocation?**

Observations

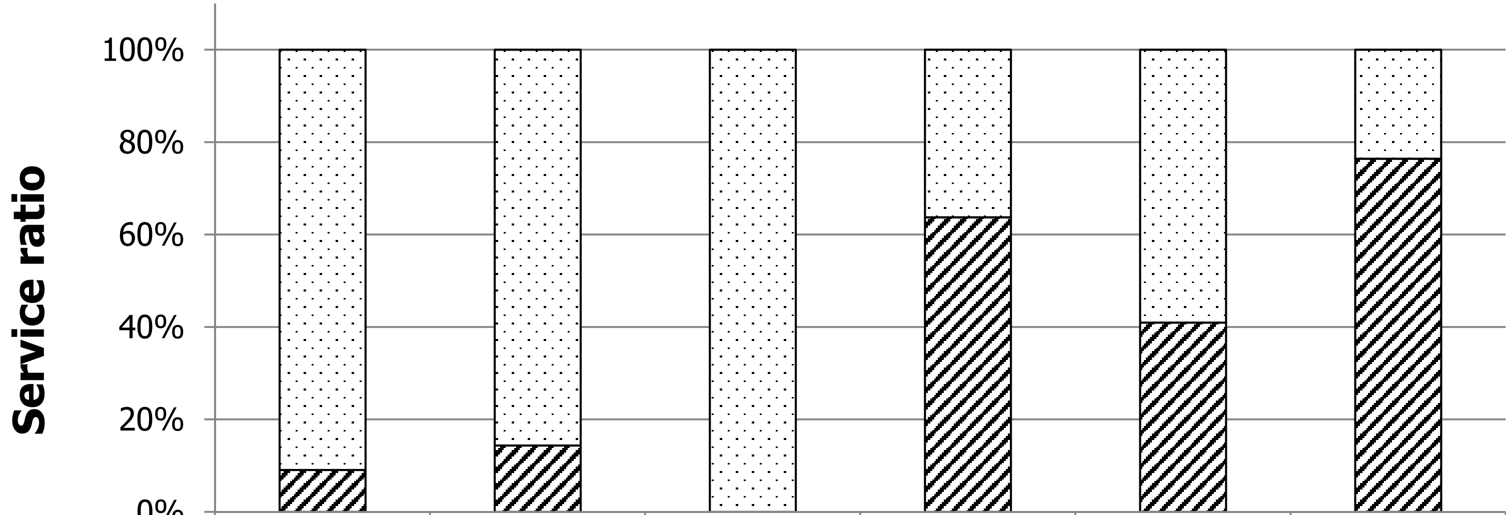
- Request types: Single vs. Multiple
  - ✓ Same test environments
  - ✓ Mainly single page frame requests



	Kernel Compile	Lmbench	Stream	Unixbench	Dbench	Tbench
Multiple page	106316	9442	25	3195113	351	366
Single page	85977206	89195105	1504996	213534514	38052300	112785688

Observations

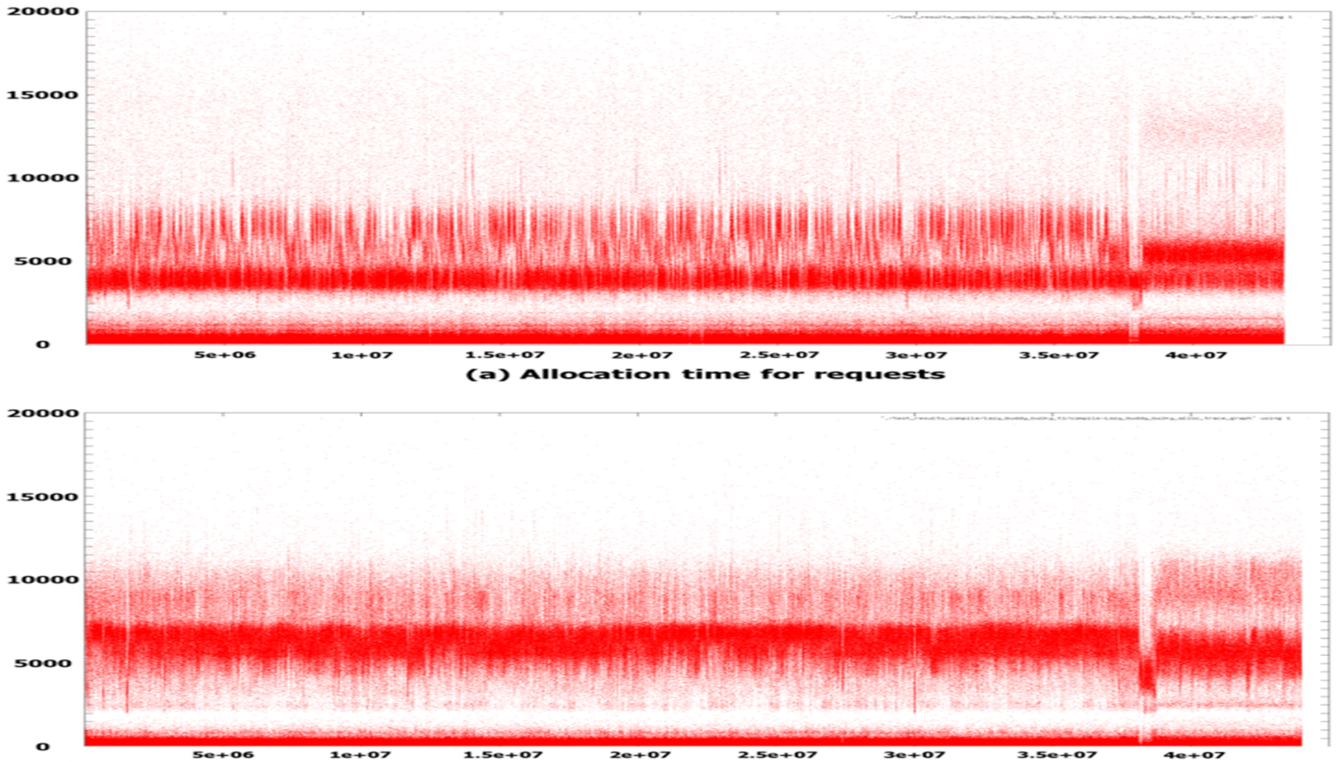
- Service layer
  - ✓ Same test environments
  - ✓ Large portion of requests are handled in the buddy layer
  - ✓ Depend on workload characteristics (burstiness)



	Kernel Compile	Lmbench	Stream	Unixbench	Dbench	Tbench
the buddy layer	78192436	76376010	1504080	77406822	22464403	26593767
the lazy layer	7784770	12819095	916	136127692	15587897	86191921

Observations

- Response time
  - ✓ Same test environments
  - ✓ Significant Buddy layer overhead (for splitting and coalescing)
  - ✓ Large response time variations



(a) Allocation time for requests

(b) Free time for requests

☞ **Get sidetracked**

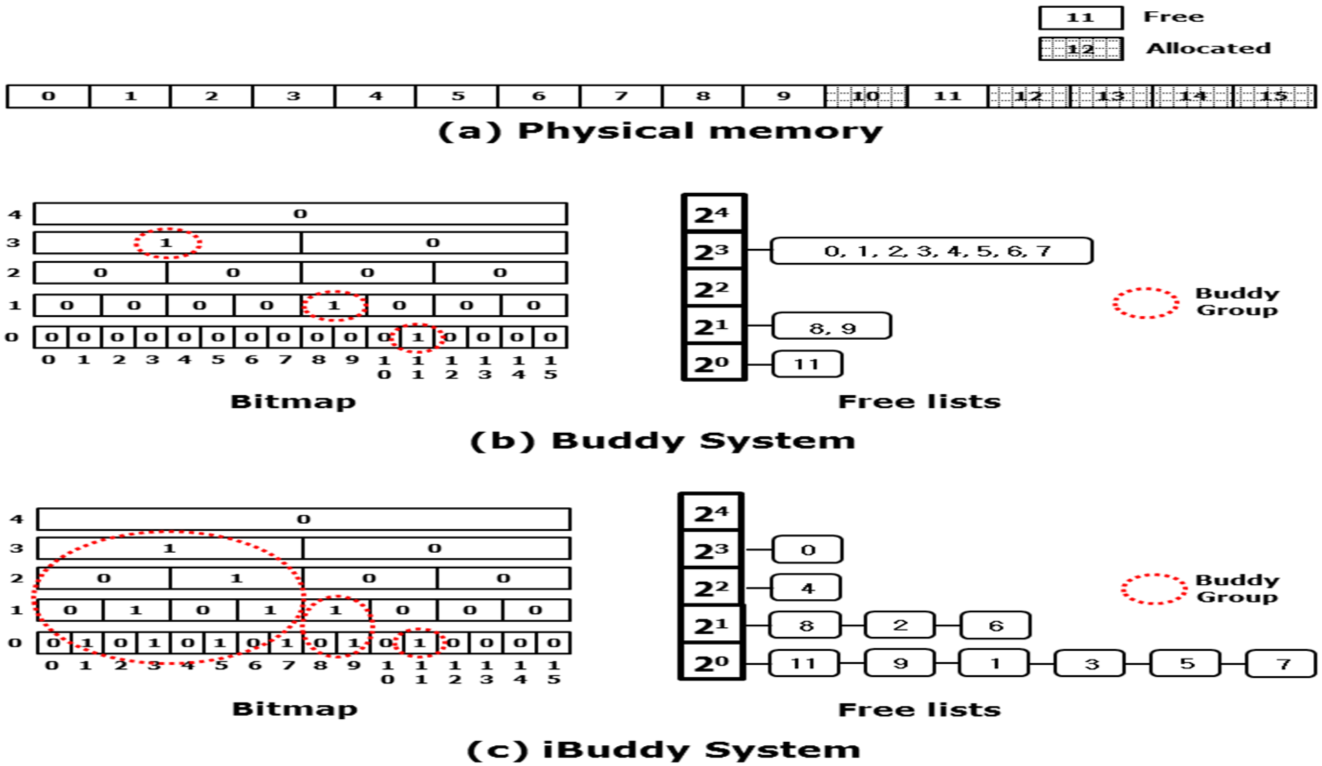
Proposal: iBuddy

- New buddy system
  - ✓ Fair allocation (based on allocation counts)
  - ✓ Overcome the unfairness problem of Lazy layer
  - ✓ Individual page frame management
  - ✓ Reducing the splitting and coalescing overheads
  - ✓ In addition, efficient handling multiple page frames requests

☞ **iBuddy: Inverse (or Individual) Buddy**

Proposal: iBuddy

- Structure
  - ✓ Individual page frame management
  - ✓ Dual meaning bitmap
  - ✓ Splitting or coalescing occurs only for multiple page frames request (laziest buddy)

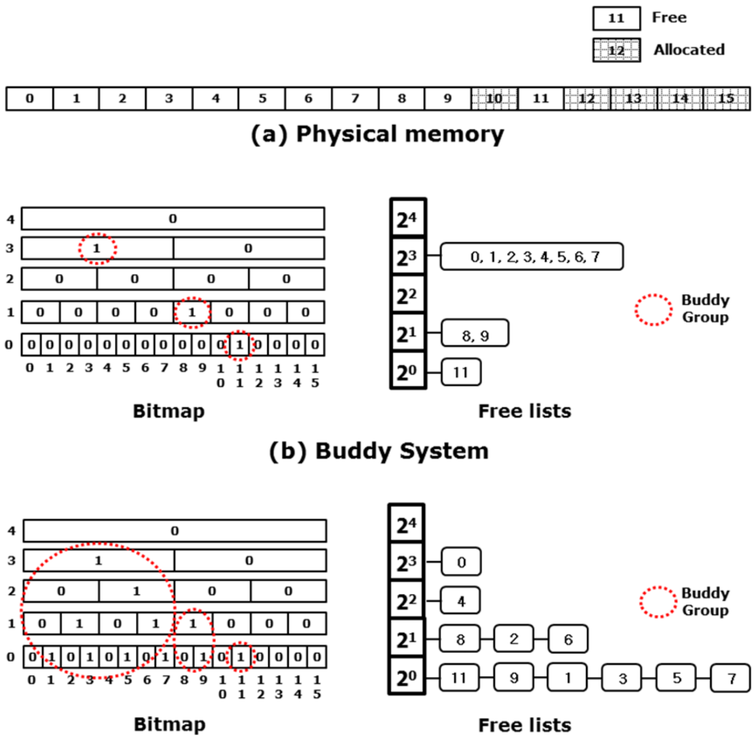


Proposal: iBuddy

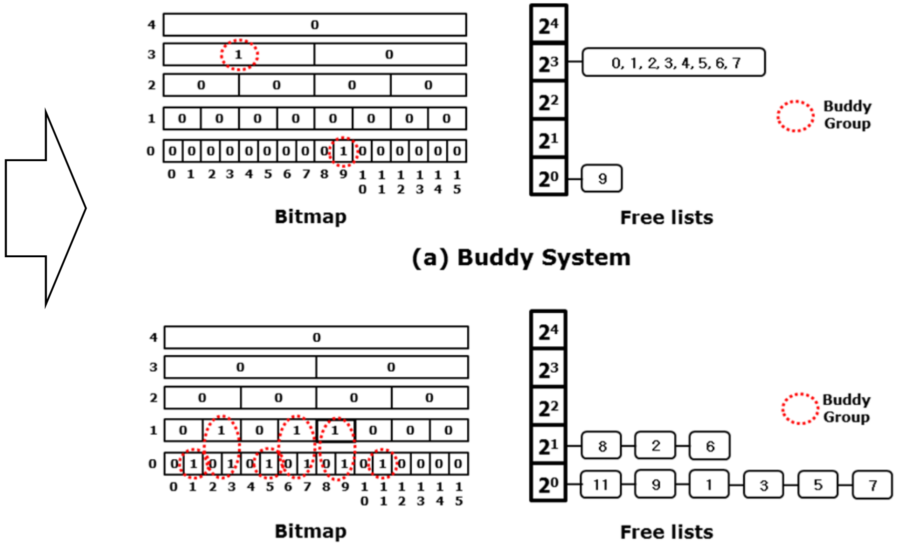
- Allocation
  - ✓ after handing two single page frame requests

Algorithm 1 Allocation for lazy iBuddy system

```
procedure _ALLOC_PAGES(sz)
    lazy_list ← get lazy_list of current core
    if sz == 4KB and lazy_list is NOT empty then
        delete page from lazy_list
        Return ptr of page
    else
        free_area ← get buddy space assigned for this core
        if no page satisfies this request then
            free_area ← get new free_area
        end if
        lock free_area
        get represent_page from next_allocation_level
        get first_page and last_page from represent_page
        while first_page to last_page do
            delete page from free_list
            clear page bit_map location bit on level
        end while
        if no page on next_allocation_level then
            adjust next_allocation_level
        end if
        unlock free_area
        Return ptr of first_page
    end if
end procedure
```



(c) iBuddy System



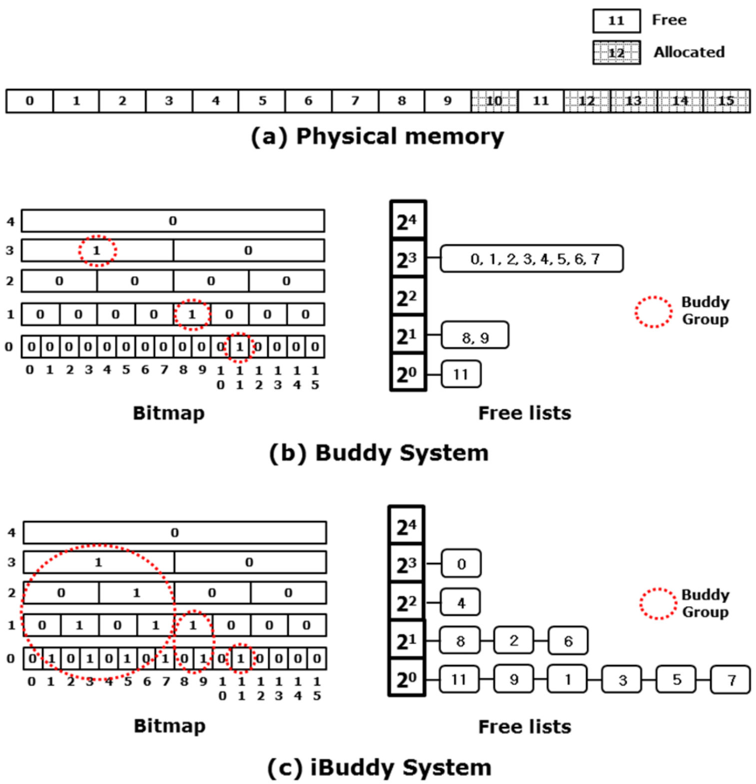
(b) iBuddy System

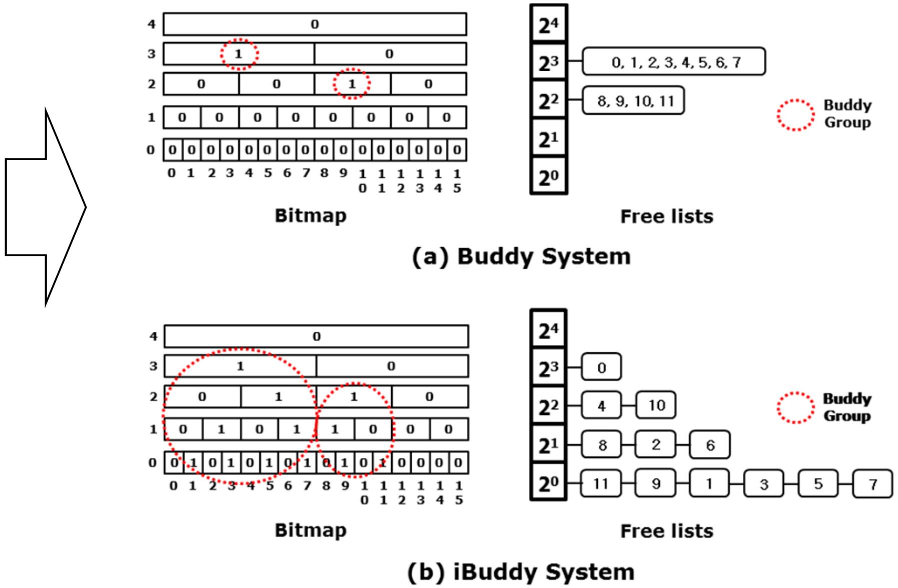
Proposal: iBuddy

- Free
  - ✓ after handing a single page frame (10) free request

Algorithm 2 Deallocation for lazy iBuddy system

```
procedure _FREE_PAGES(ptr of first_page, sz)
    lazy_list ← get lazy_list of current core
    if sz == 4KB and lazy_list is NOT full then
        insert page to lazy_list
    else
        free_area ← get buddy space from first_page
        lock free_area
        while first_page to last_page do
            find free_list level for page
            insert page to free_list
            set page bit_map location bit on level
        end while
        if last level of free_list > next_allocation_level then
            next_allocation_level ← last level of free_list
        end if
        unlock free_area
    end if
end procedure
```





Proposal: iBuddy

- Summary of iBuddy characteristics

		Lazy Buddy System	Lazy iBuddy system
When Coalescing happened		Page is freed into buddy layer	Multiple page allocation request
When Splitting happened		Page is allocated from buddy layer	Multiple page free request
Time complexity	Single page	O(logn)	O(1)
	Multiple pages	O(logn)	O(n)
Lock granularity on buddy layer		Coarse-granularity	Fine-granularity
The number of Pages Management Policy on the lazy layer		Bulky	Bypass
Performance improvement ratio (baseline : Lazy Buddy system)		-	32%
Standard deviation		1400 cycles	400 cycles

Performance Evaluation

- Allocation/Free response time
  - ✓ Test environments: Intel 8 cores, 32GB DRAM, 450GB*10 Disks
  - ✓ OS: Linux 2.6.32
  - ✓ Workload: Kernel compile, Lmbench, Stream, Unixbench, Dbench, Tbench

Table 2. Average elapsed time for memory requests (cycles)

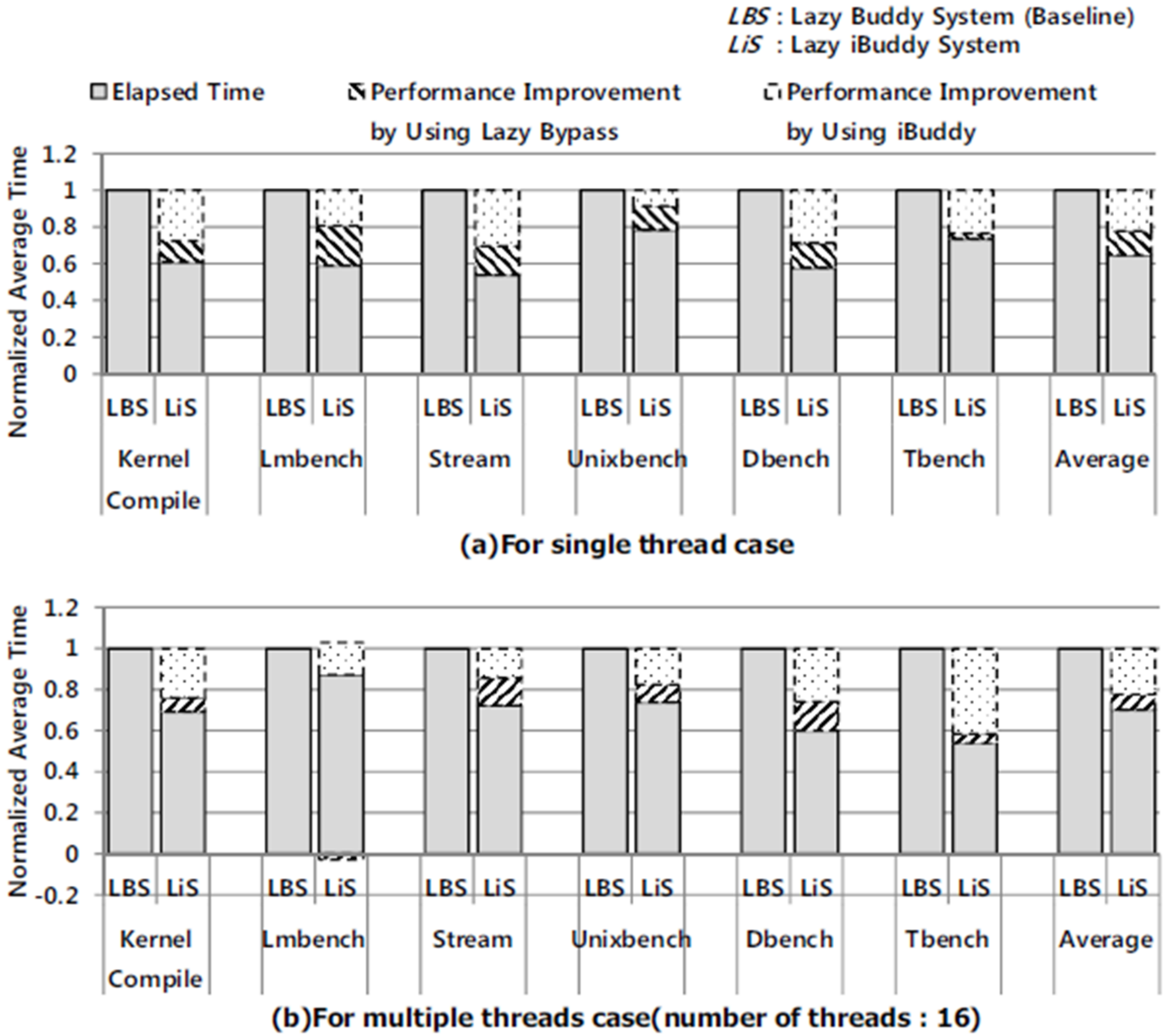
	Kernel Compile						Lmbench					
	1	2	4	8	16	32	1	2	4	8	16	32
(a) Standard Buddy System	292	269	265	325	473	487	194	221	385	668	991	1261
(b) [LBS-1] Lazy Buddy System (Batch Size = 1)	277	252	257	317	463	484	188	225	356	594	988	989
(c) [LBS-31] Lazy Buddy System (Batch Size = 31)	292	285	293	332	416	434	213	222	280	361	416	446
(d) iBuddy System	194	178	181	220	306	325	131	163	247	387	467	563
(e) [LiS-1] Lazy iBuddy System (Batch Size = 1)	178	162	166	200	288	306	126	146	193	305	363	375
(f) [LiS-31] Lazy iBuddy System (Batch Size=31)	212	212	223	258	316	326	172	181	216	296	351	357
Performance increase ratio (between (c) and (e))	39%	43%	43%	40%	31%	29%	41%	34%	31%	16%	13%	16%

	Stream						Unixbench					
	1	2	4	8	16	32	1	2	4	8	16	32
(a) Standard Buddy System	190	215	268	344	396	405	406	448	491	649	590	-
(b) [LBS-1] Lazy Buddy System (Batch Size = 1)	193	203	332	341	408	387	279	249	258	291	262	-
(c) [LBS-31] Lazy Buddy System (Batch Size = 31)	217	229	227	245	285	253	317	258	277	266	295	-
(d) iBuddy System	106	111	116	162	190	169	360	351	380	443	411	-
(e) [LiS-1] Lazy iBuddy System (Batch Size = 1)	117	121	142	152	206	170	249	210	221	239	218	-
(f) [LiS-31] Lazy iBuddy System (Batch Size=31)	151	189	176	230	244	245	290	234	246	261	243	-
Performance increase ratio (between (c) and (e))	46%	47%	37%	38%	28%	33%	21%	19%	20%	10%	26%	-

	Dbench						Tbench						Average of Six Benchmarks
	1	2	4	8	16	32	1	2	4	8	16	32	
(a) Standard Buddy System	253	261	347	364	474	526	231	254	251	291	441	446	411
(b) [LBS-1] Lazy Buddy System (Batch Size = 1)	187	215	288	308	386	428	162	161	158	161	187	193	326
(c) [LBS-31] Lazy Buddy System (Batch Size = 31)	195	248	348	370	474	459	171	181	178	180	221	226	289
(d) iBuddy System	160	197	265	292	365	393	197	200	207	227	318	336	266
(e) [LiS-1] Lazy iBuddy System (Batch Size = 1)	113	146	201	219	284	287	126	127	120	119	119	125	195
(f) [LiS-31] Lazy iBuddy System (Batch Size=31)	139	177	258	283	351	358	131	129	121	117	129	133	227
Performance increase ratio (between (c) and (e))	42%	41%	42%	41%	40%	37%	26%	30%	33%	34%	46%	45%	32%

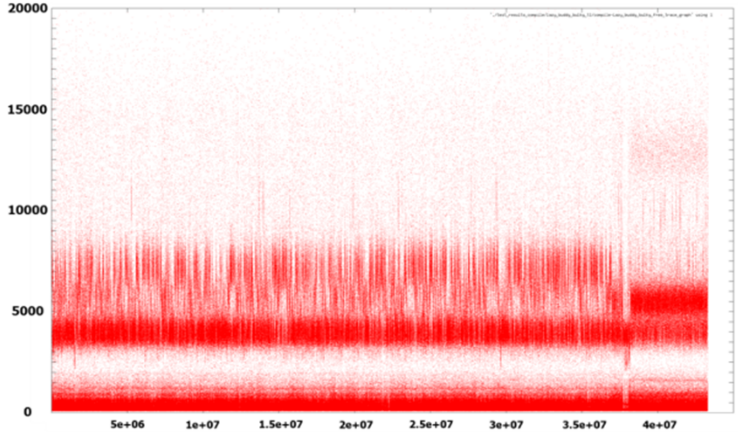
Performance Evaluation

- Performance Improvement Analysis

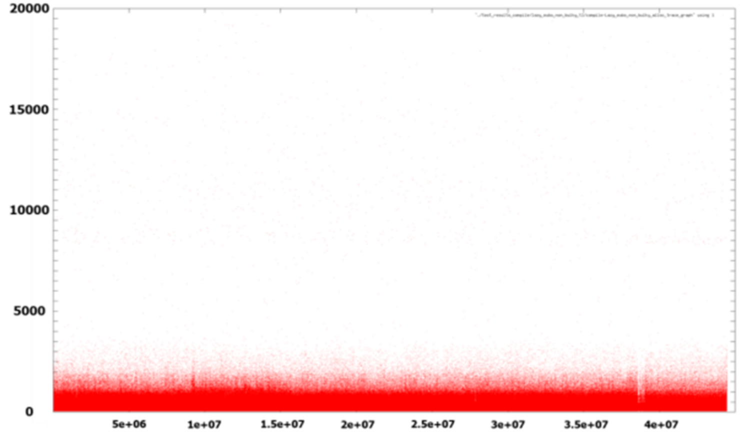


Performance Evaluation

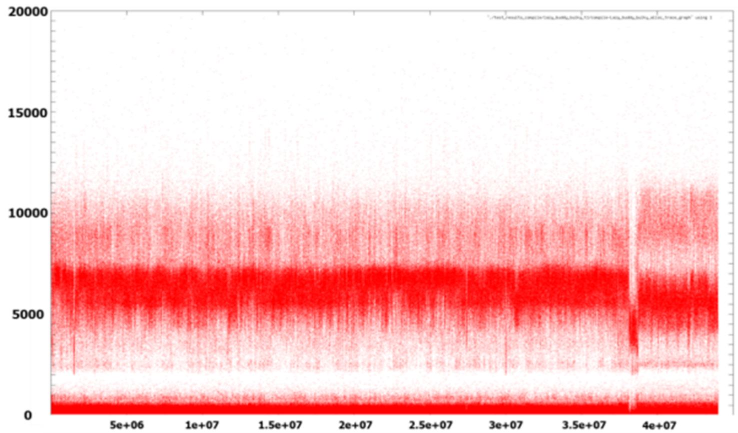
- Variation of Response time



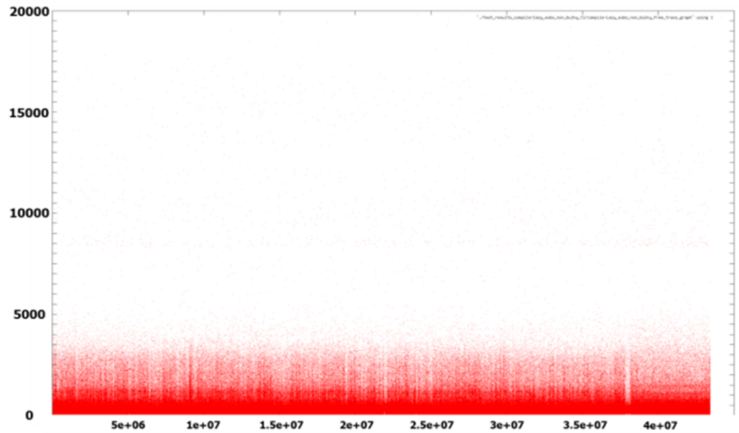
(a) Allocation time for requests



(a) Allocation time for requests



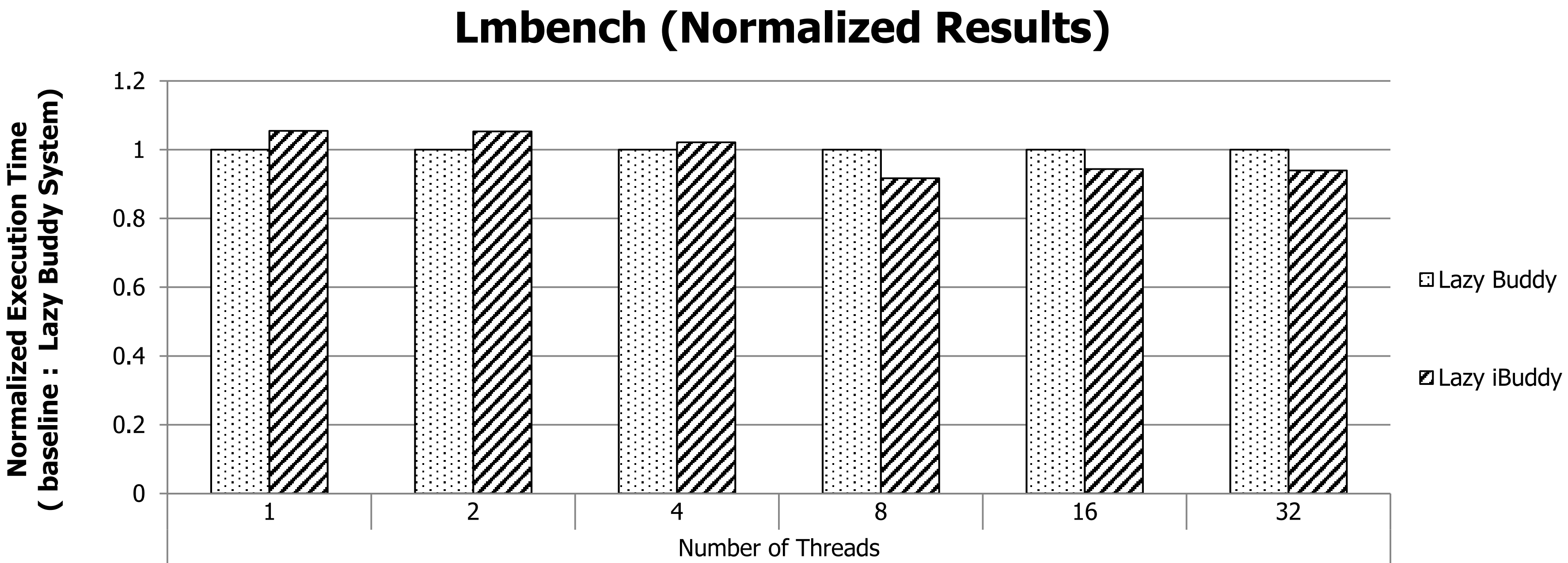
(b) Free time for requests



(b) Free time for requests

Performance Evaluation

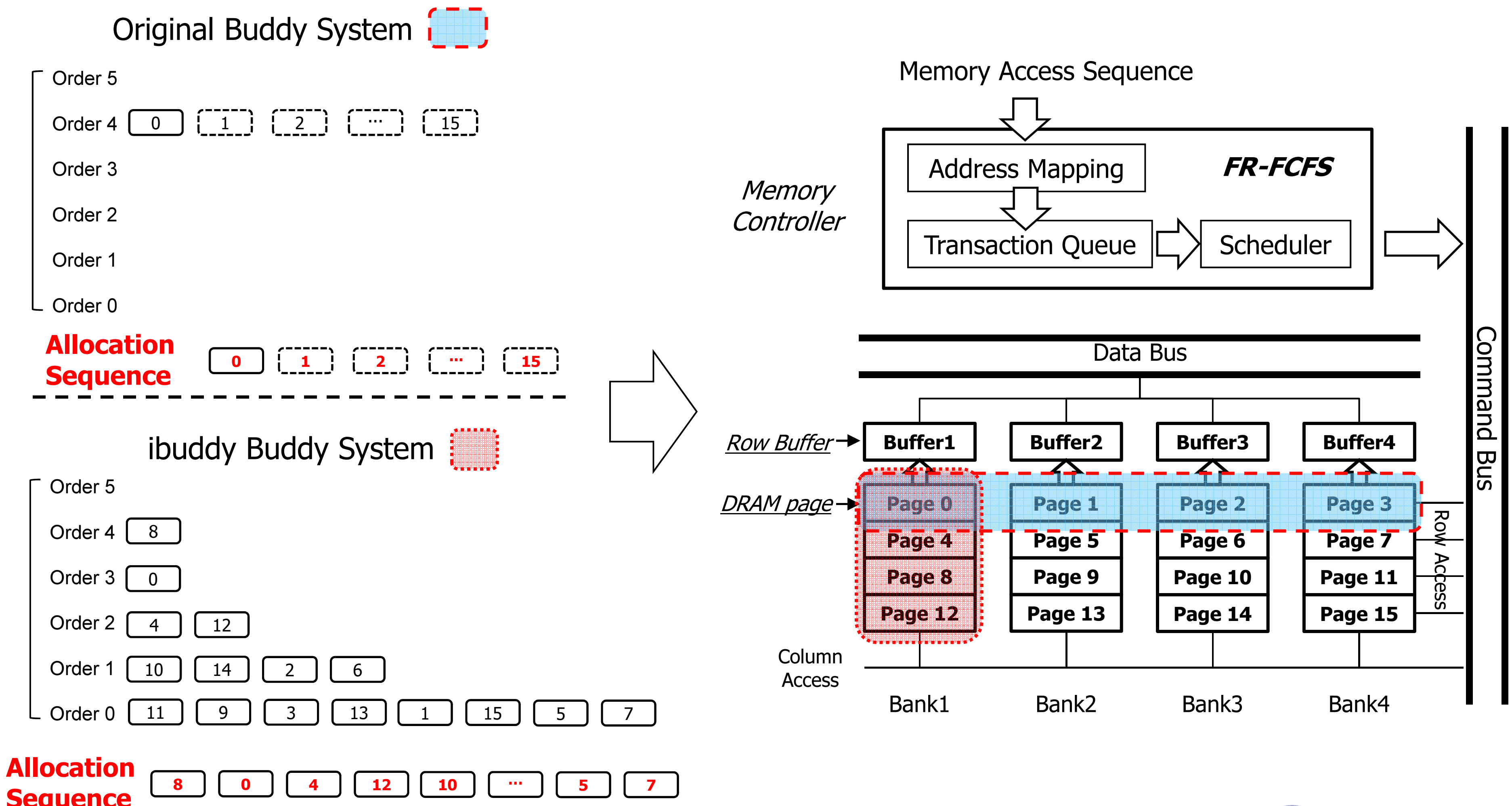
- But, …
  - ✓ Total execution time of benchmark



	1	2	4	8	16	32
Lazy Buddy	1	1	1	1	1	1
Lazy iBuddy	1.054610486	1.053342336	1.021406728	0.91680208	0.943805598	0.93977559

Performance Evaluation

- Possible causes about performance degradation for small thread cases



Conclusion

- New buddy system: iBuddy
  - ✓ Inverse thinking
    - Managing page frames individually
    - Splitting and coalescing occurs on multiple page frames request
  - ✓ But, the original lazy buddy has its own strong points
    - CPU cache, multibank
    - Can keep large consecutive page frames
  - ✓ Issues
    - Multicore/Multibank
      - Multibank parallelism
      - Multicore issues (lock issues in the buddy system)
      - NUMA issues
    - Fair-allocation for SCM
      - RB-tree
      - Performance degradation issues