Managing Non-Volatile Memory in Database Systems

연세대학교 컴퓨터과학과 김도영

2019년 12월

과제명: IoT 환경을 위한 고성능 플래시 메모리 스토리지 기반 인메모리 분산 DBMS 연구개발

과제번호: 2017-0-00477

과학기술정보통신부
Ministry of Science and ICT

연세대학교
YONSEI UNIVERSITY

정보통신기술진흥센터
Institute for Information & communications Technology Promotion

목차

- Introduction
- Background
- NVM Buffer Management
- Three-tier Buffer Management
- Evaluation
- Related Work

Introduction

NVM

- A.k.a Storage Class Memory (SCM) or NVRAM
- DRAM의 byte address 형식과 SSD의 memory persistence를 계승
- 아직 상업적으로 활발하게 개발된 것은 아님
- 이후 **DRAM보다 느리고 큰 용량으로, 그리고 SSD보다 빠르고 작은 용량으로** 발전 예상
- "NVM이 DRAM과 SSD를 대체할 수 있을까?"
- **"어떻게 NVM을 사용할 것인가?"** Logging/recovery or storage
- 최근의 연구들은 NVM을 logging과 recovery에 사용하려는 경향
- 위 연구에서는 NVM을 **storage/cache**의 용도로 사용하고자 함

NVM into Storage Layer

- NVM을 **primary storage**로 이용하는 방법

  \+ Byte addressability를 최대한 활용할 수 있다

  \- Relations와 indexes를 main memory에 저장하여 DRAM의 lower latency를 이용한 main memory database system보다 속도가 다소 느릴 수 있다.

  (NVM의 높은 latency를 숨기기 위해 DRAM을 NVM 앞단에 cache로 이용하려는 시도)

- NVM의 용량이 SSD보다 상대적으로 작으므로 큰 데이터셋에 대한 대비가 미흡하다

Approach in this Work

- NVM을 **additional caching layer**로 사용
- "Main memory를 대체하기에는 빠르지 않지만 disk를 대체할 만큼 싸지도 않다."
- Tradeoffs and performance cliffs
- NVM의 byte addressability를 활용하면서

  DRAM, NVM, flash를 동시에 지원하는

  Storage engine을 구현하고자 함

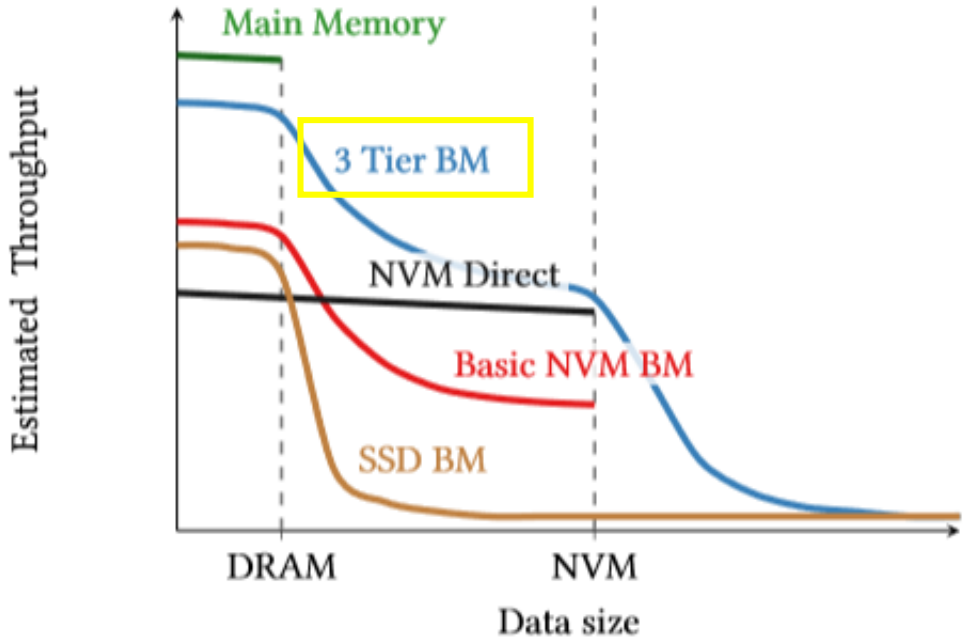


Figure 1: System designs under varying data sizes.

3 Tier Buffer Manager

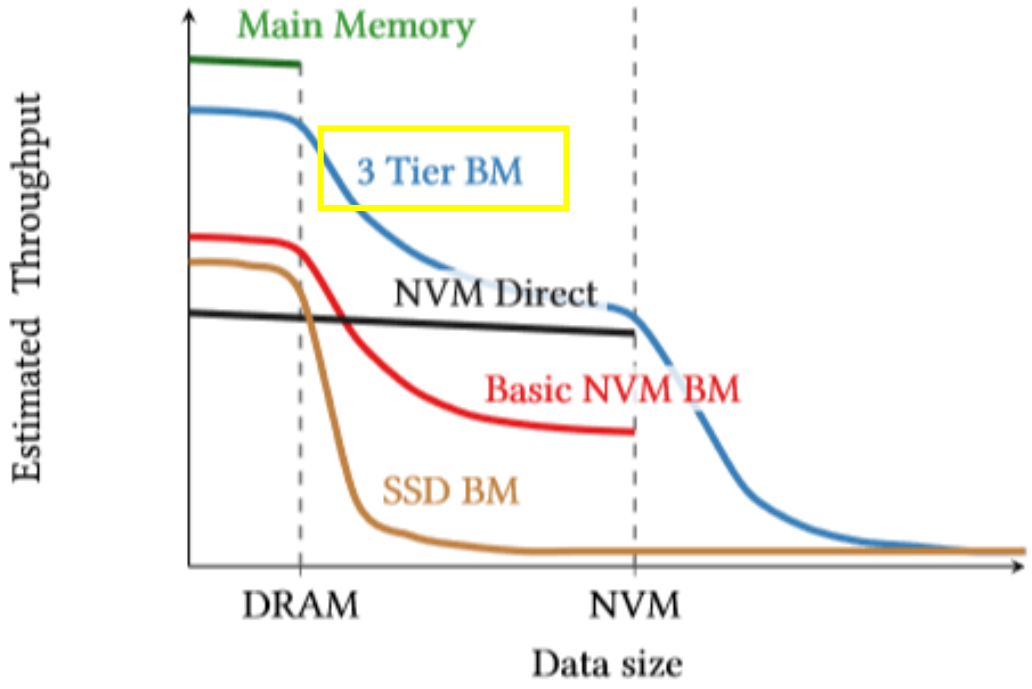


Figure 1: System designs under varying data sizes.

- Performance cliff를 없애고 specialized systems와 유사하거나 그보다 나은 성능을 보이는 3 Tier Buffer Manager를 개발
- NVM은 DRAM과 SSD를 보완하는 additional layer로 활용된다.
- SSD를 지원함으로써 **큰 데이터셋을 수용**할 수 있고 다른 접근들보다 **경제적**이다.

3 Tier Buffer Manager - Techniques

- DRAM에 NVM 액세스를 **cache-line granularity**로 캐싱하는 방식으로 NVM의 byte-addressability 이익을 최대화한다.
- 버퍼 풀은 일시적으로 **작은 page size**를 사용하여 한정된 DRAM cache를 효율적으로 사용한다.
- 동시에, **SSD에 data를 저장할 때에는 큰 page size를 채택**하여 큰 데이터셋을 받는다.
- In-memory access의 오버헤드를 줄이기 위해 **가벼운 buffer management**를 추구한다.
- NVM에 바로 업데이트하기보다 main memory에서 업데이트를 수행하는 방식으로 endurance를 올리고 write latency를 감춘다.

Background

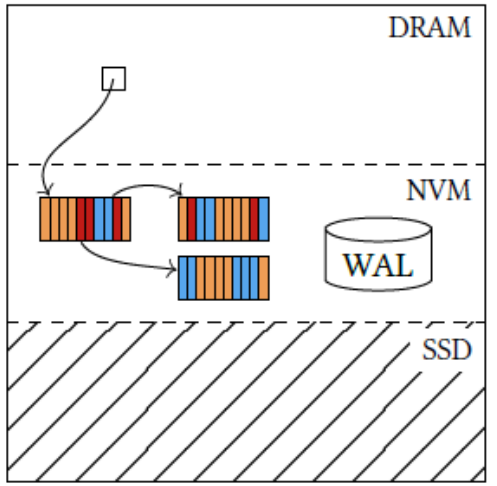
NVM Storage : NVM Direct (Arulraj et al.)

- NVM을 **primary storage layer**로 사용
- In-place update를 레퍼런스로 NVM에 직접 업데이트하는 방식을 채택
- 그러나 업데이트 시 Volatile CPU cache를 거친다.
- ID와 이전/이후 이미지를 저장하는 write-ahead log 엔트리 작성을 통해 logging 엔트리를 작성한 뒤 해당 cache line을 evict하는 방식으로 NVM에 유지된다.
- 모든 Data를 NVM에 저장하기 때문에 몇 가지 단점이 있다.

  e.i.) higher latency에 따른 lower throughput,

  한정된 NVM endurance로 인해 HW failure의 위험,

  eviction이 어렵고 모든 수정이 잠재적으로 persist하므로 프로그래밍의 어려움 등

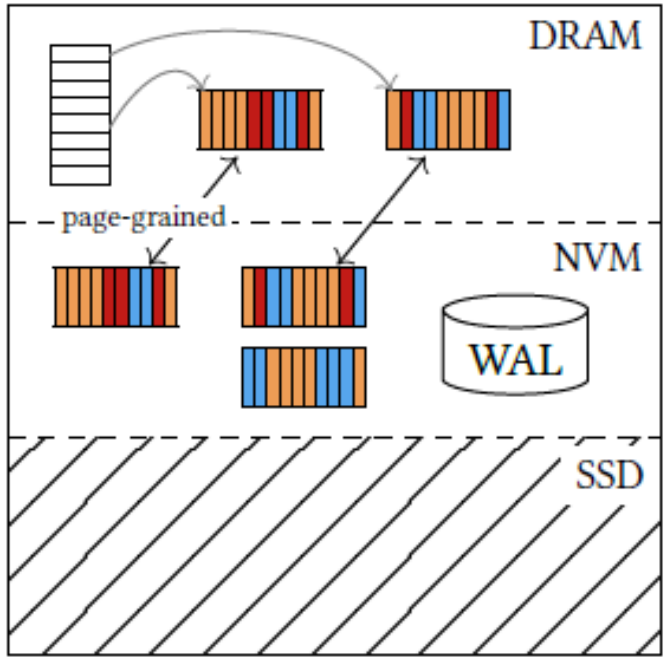


(a) NVM Direct

NVM Storage : Basic NVM Buffer Manager

- **DRAM을 NVM 앞의 cache**로 사용하는 방법
- Volatile / persistent 사이를 관리하기 위해 buffer manager를 채택
- 모든 page를 NVM에 저장, DRAM은 buffer pool로 사용
- DRAM에서만 transaction이 일어난다.
- Buffer pool에 page를 lock하기 위해 fix/unfix functions를 사용
- Page identifier대신 두 개의 pointers를 저장하는 Research prototype

  FOEDUS가 variant로서 고안되었다. (Asynchronous process)



(b) Basic NVM BM

Recovery

- 로그 엔트리는 SSD보다 NVM에 **더 빨리 write**할 수 있다. (NVM logging의 이점)
- Write ahead logging with redo & undo information을 채택
- Undo entries : Rollback을 가능케하고 recovery동안 loser의 transaction을 undo한다.
- Redo entries : commit된 transactions가 recover동안 아직 persistent하지 않을 경우 이를 반복한다.
- Log-sequence number per page

NVM Buffer Management

Goals

- 작은 데이터에서는 main-memory database와 동등한 성능을 가지지만 데이터가 커짐에 따라 NVM, SSD storage로 스케일을 넓혀가며 **완곡하고 자연스러운 성능상의 degrading**을 가지는 시스템
- DRAM과 NVM 사이에서 cache-line-grained data objects를 자리바꿈해주는 DRAM-resident buffer manager를 고안

  이를 통해 cold data가 많이 포함된 page의 hot data object를 **작은 메모리 프레임으로 추출**할 수 있다.
- Pointer swizzling scheme

Cache-Line-Grained Pages

- **NVM의 low latency**는 flash에 비해 cache line을 전송하기에 유리하다.
- Cold page에서 hot data를 뽑아 bandwidth를 보존하고 성능을 올리는 테크닉
- NVM으로부터 페이지를 **부분적으로** 가져와 DRAM에 배치한다.
- 이미 로드된 페이지들에 대한 정보를 저장하기 위해 bitmask (resident)를 기록한다.

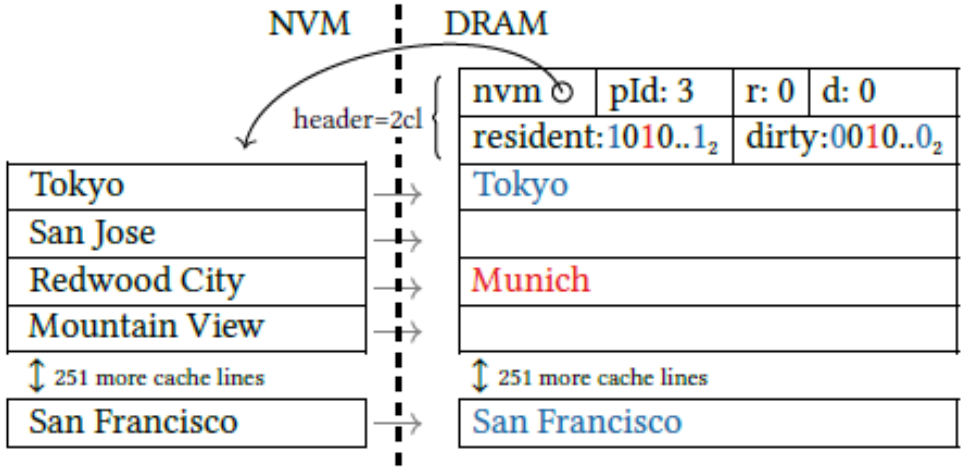
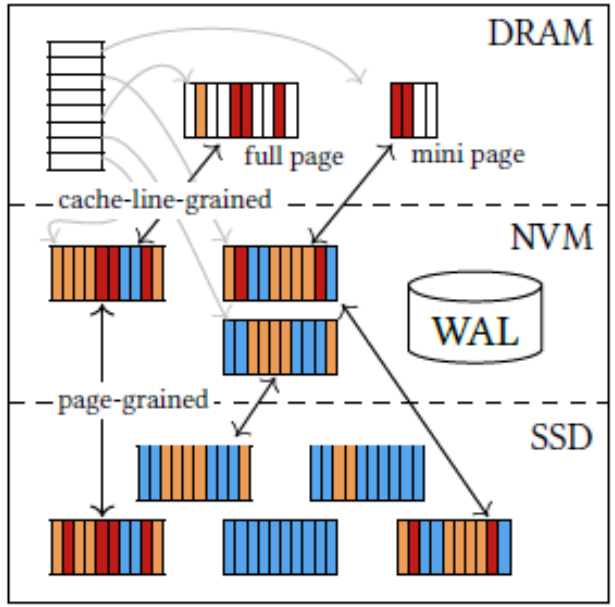


Figure 3: Cache-Line-Grained Pages – The bit masks indicate which cache lines are resident and which are dirty.



(c) Our NVM-Opt Three-Tier BM

Cache-Line-Grained Pages (ctnd.)

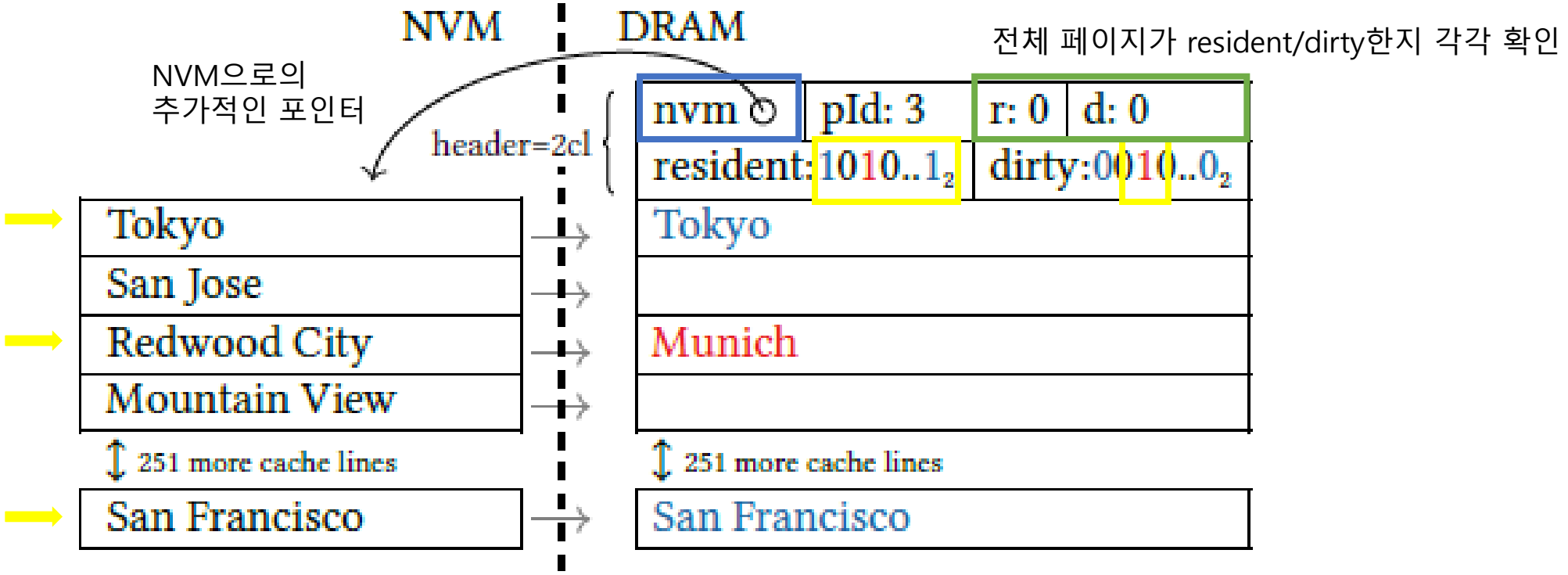
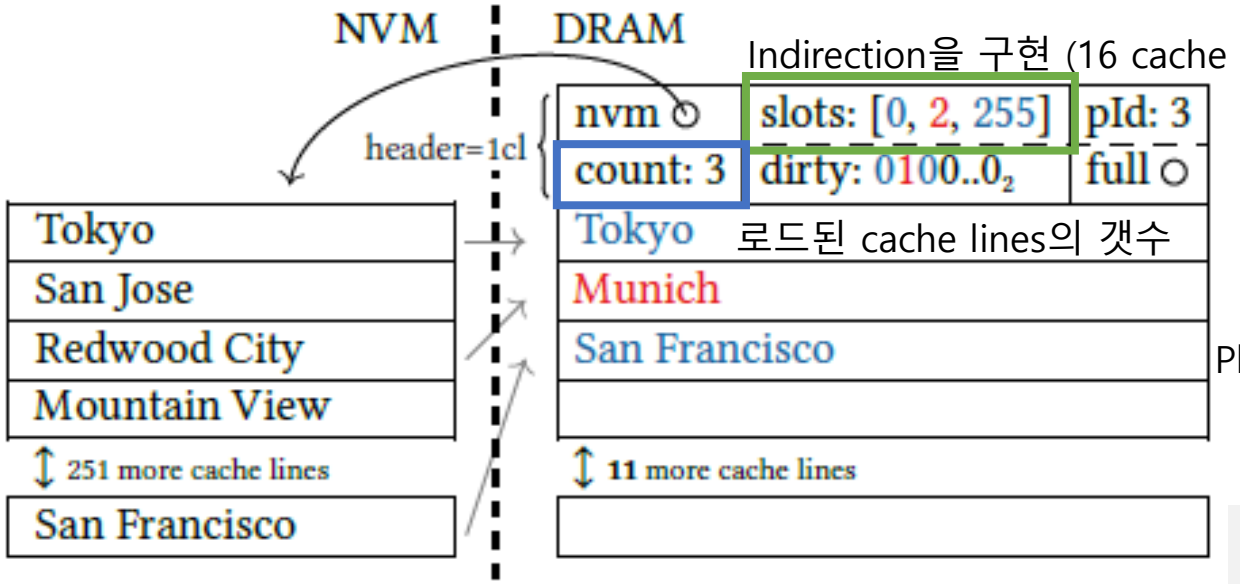


Figure 3: Cache-Line-Grained Pages – The bit masks indicate which cache lines are resident and which are dirty.

Mini Pages

- 기존의 page layout은 필요 이상으로 DRAM을 소모한다.
- 낭비되는 메모리를 줄여주는 보다 작은 2차 페이지 'mini page'



Indirection을 구현 (16 cache line slots에 각각 physical cache line id를 저장)

로드된 cache lines의 갯수

Physical index 255에 저장되어있지만 mini page slot 2에 로드됨

-> Cache line 'Redwood City'가 'Munich'로 바뀌었고 write back되어야 한다.

Figure 4: Mini Pages – The `slots` array indicates which cache lines are loaded (max 16). If promoted, `full` points to the full page.

Mini Pages – Mini to full page

- Mini page가 더이상 요청을 수행할 memory를 가지고 있지 않으면 이는 **full page로 변환된다**.
- Buffer manager에 빈 full page를 배정하고 mini page의 full member에 이 페이지를 저장한다.
- 그 뒤, mini page의 현재 status가 새로 배정된 full page에 복사된다.
- 마지막으로, buffer manager 안의 **page mapping table**은 full page를 point하기 위해 업데이트된다.
- 그 이후로 mini page는 partially promote라고 불리우게 되고, mini page로의 모든 요청은 full page로 전해진다.
- Garbage collection은 오직 마지막 transaction이 이를 unfix하는 순간에만 안전하게("safely") 이루어진다.

  (page mapping table이 full page를 가리키게되고 mini page에 더이상의 new reference가 이루어지지 않을 때)

Pointer Swizzling

- 동적인 방법으로 **page id를 physical pointers**와 대체함으로써 **overhead를 줄이는 테크닉**
- Overhead를 줄이는 것이 이전보다 중요해진 이유
- Page의 주소를 인코딩하여 DRAM-resident pages의 page identifier에 넣는다.

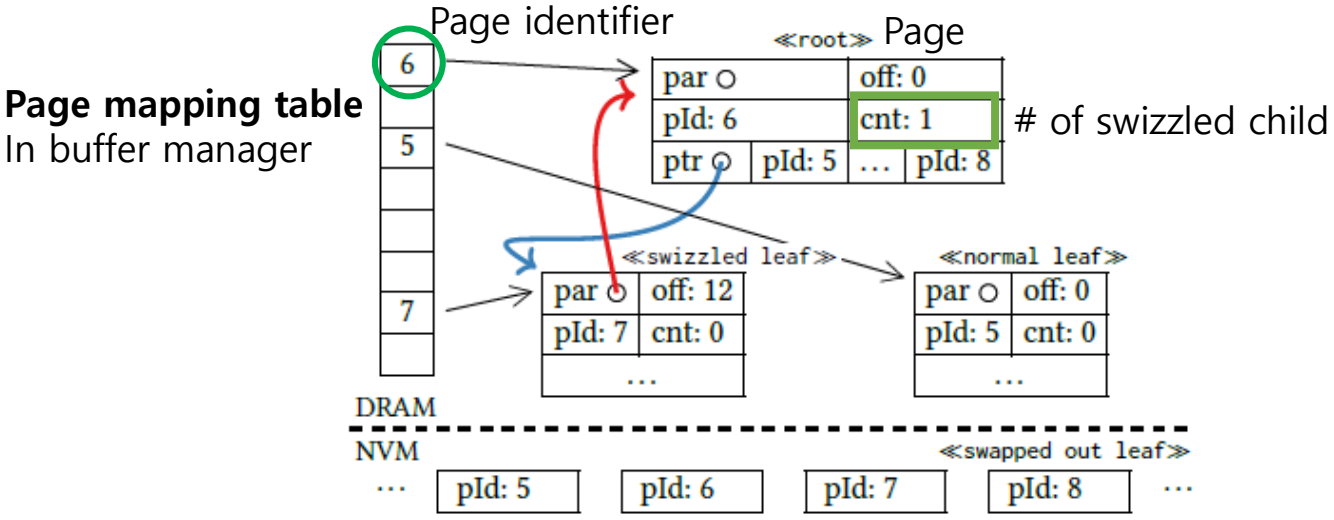


Figure 5: Pointer Swizzling – A B-tree with a root (pId: 6) and three child pages: A swizzled page (pId: 7), a normal DRAM page (pId: 5) and a page currently not in DRAM (pId: 8).

Pointer Swizzling (ctnd.)

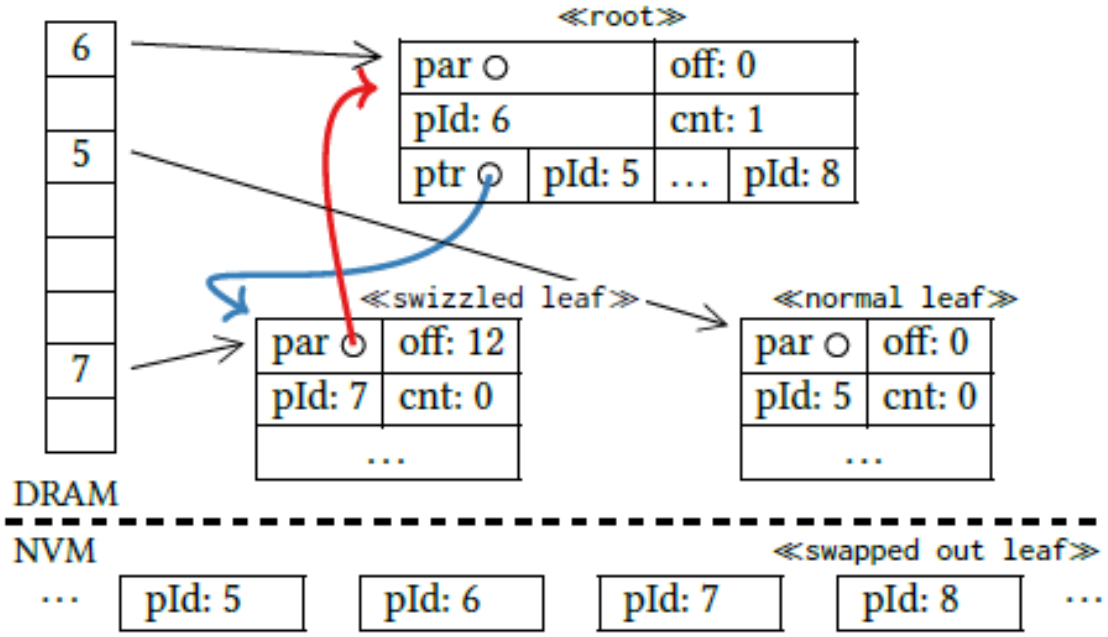


Figure 5: Pointer Swizzling – A B-tree with a root (pId: 6) and three child pages: A swizzled page (pId: 7), a normal DRAM page (pId: 5) and a page currently not in DRAM (pId: 8).

Swizzled page를 swap할 때
1. Par를 통해 그 부모 페이지를 수정한다
2. 스스로를 가리키던 offset pointer(off)를 normal page identifier로 변환한다.

Pointer swizzling은 다른 자료구조에도 가능
고정된 조건은 **fixed-size pages/header field**

Swizzled mini page가 promote될 때,
Swizzling information도 함께 업데이트되어야한다.

Mini page가 partially promote되고 unfix되기 전까지는 full page의 wrapper로 기능하다가 unfix될 때
Parent pointer를 full page로 redirect한다.
(이 때 par와 off pointer는 full page로 복사된다.)

Three-tier Buffer Management

Goals

- Flash(SSD)를 3차 레이어로 추가하되, low overhead를 가진다.
- 이를 통해 in-memory/NVM-only system에 비해 큰 maximum workload를 가진다.
- 큰 데이터셋을 처리할 수 있고, 보다 경제적일 수 있도록 SSD를 지원한다.

Design Outline

- 3중 구조에서, NVM과 DRAM을 SSD storage layer에 대해 **선택적으로 캐싱에 사용하여** 버퍼 관리를 한다.
- Page는 오직 DRAM에서 액세스되고 (read or write) Write Ahead Log(WAL)는 durability를 위해 기록된다.
- Recovery를 위해 textbook-style WAL과 ARIES-based restart 과정을 거친다.

Replacement Strategies

- 일반적인 구조와 달리, **두 개의 buffer pools**를 관리해야한다.
- Three necessary replacement decisions **: DRAM eviction, NVM eviction and NVM admission**

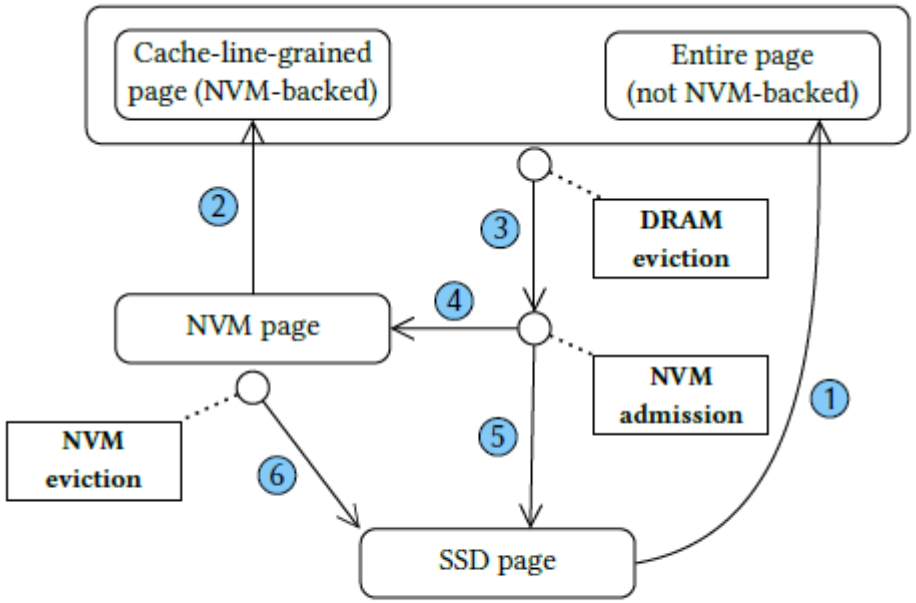


Figure 6: Page Life Cycle – There are five possible page transitions and the three critical decisions (DRAM eviction, NVM admission, and NVM eviction).

Replacement Strategies (ctnd.)

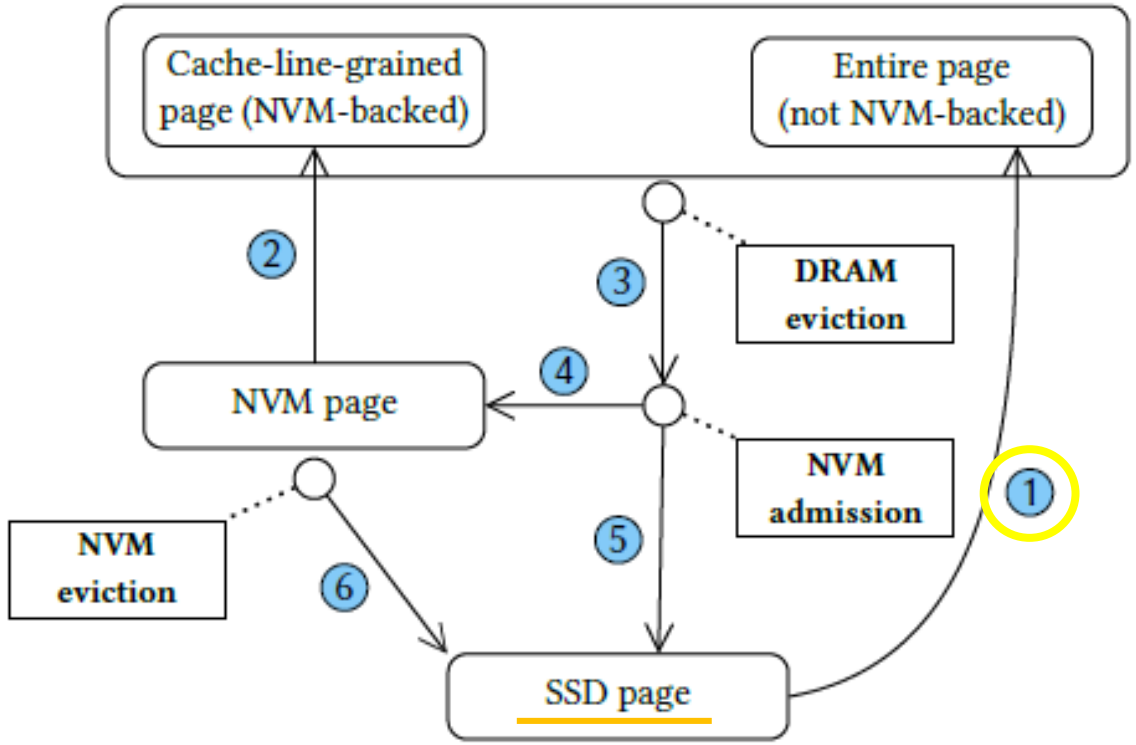


Figure 6: Page Life Cycle – There are five possible page transitions and the three critical decisions (DRAM eviction, NVM admission, and NVM eviction).

(1)

- 모든 새롭게 배정된 page는 SSD에서 시작한다.
- Transaction이 페이지를 요청하면 이는 DRAM으로 직접, 완성체로 (completely) load된다.
- Page가 completely load되는 이유
- Not NVM-backed : SSD로부터 로드될 때

Replacement Strategies (ctnd.)

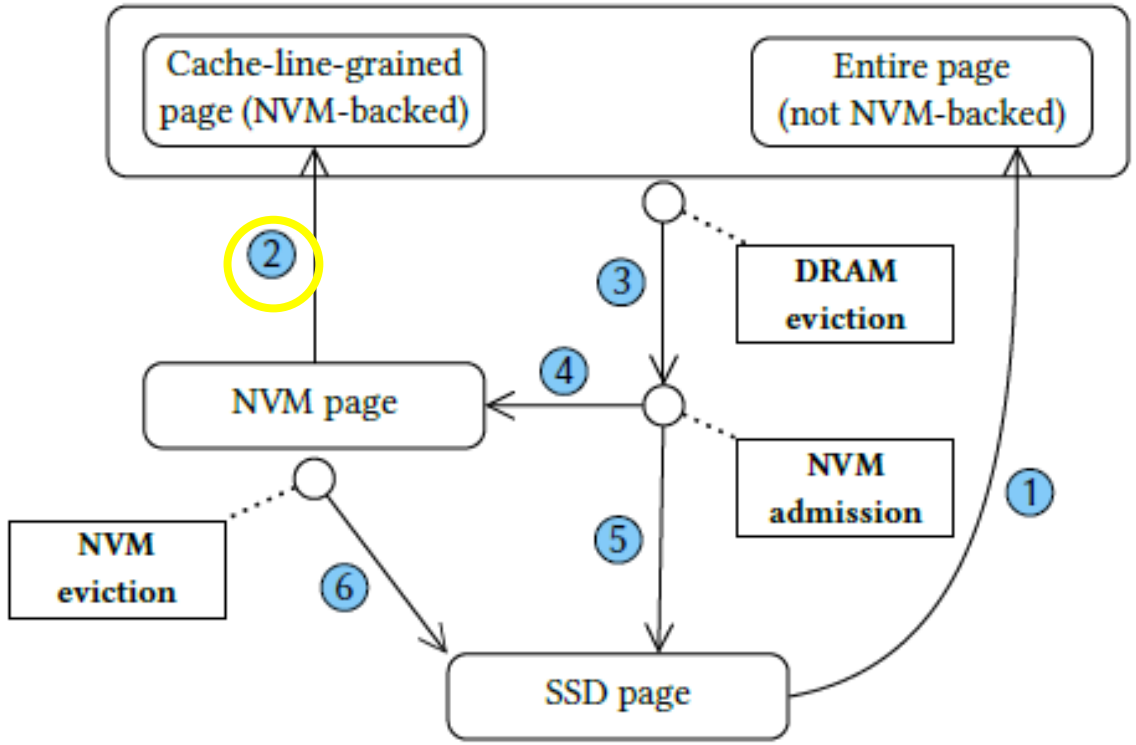


Figure 6: Page Life Cycle – There are five possible page transitions and the three critical decisions (DRAM eviction, NVM admission, and NVM eviction).

(2)

- NVM-backed
- Cache-line-grained page를 취함

(NVM backed일 때만 가능)

Replacement Strategies (ctnd.)

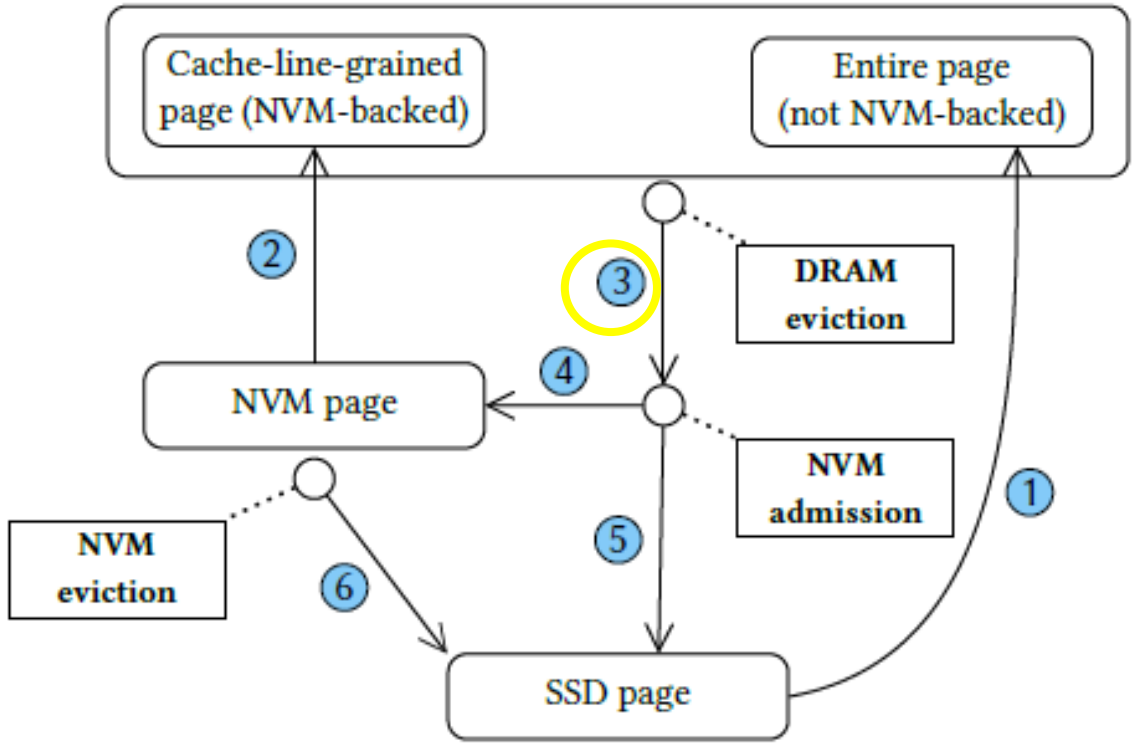


Figure 6: Page Life Cycle – There are five possible page transitions and the three critical decisions (DRAM eviction, NVM admission, and NVM eviction).

(3)

- DRAM eviction이 일어나는 상황
- DRAM eviction의 목표
- Clock/second chance algorithm

Replacement Strategies (ctnd.)

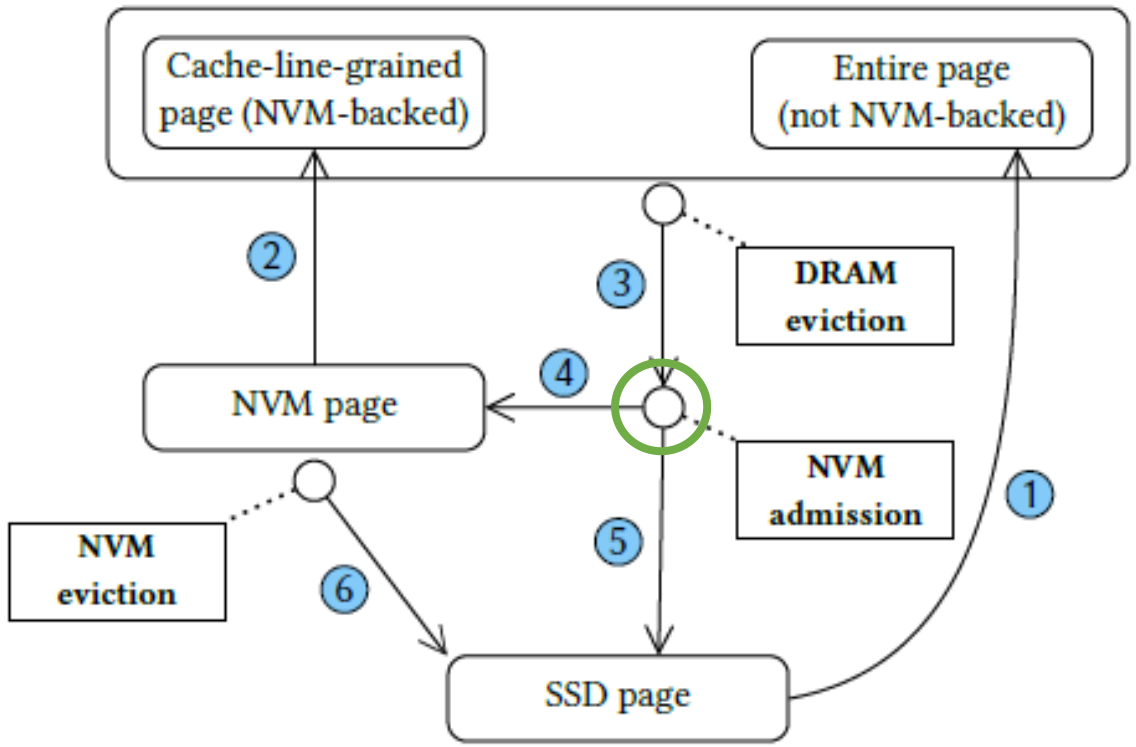


Figure 6: Page Life Cycle – There are five possible page transitions and the three critical decisions (DRAM eviction, NVM admission, and NVM eviction).

DRAM eviction을 통해 page가 선택되고 아직 NVM에 저장되지 않았을 때,

NVM admission이 수행된다.

- NVM admission의 목표
- 2개의 queue를 사용해 2-layer system에 hot data를 최적화된 방법으로 정렬하여 이후 replacement 때를 대비
- Admission set 사용

Replacement Strategies (ctnd.)

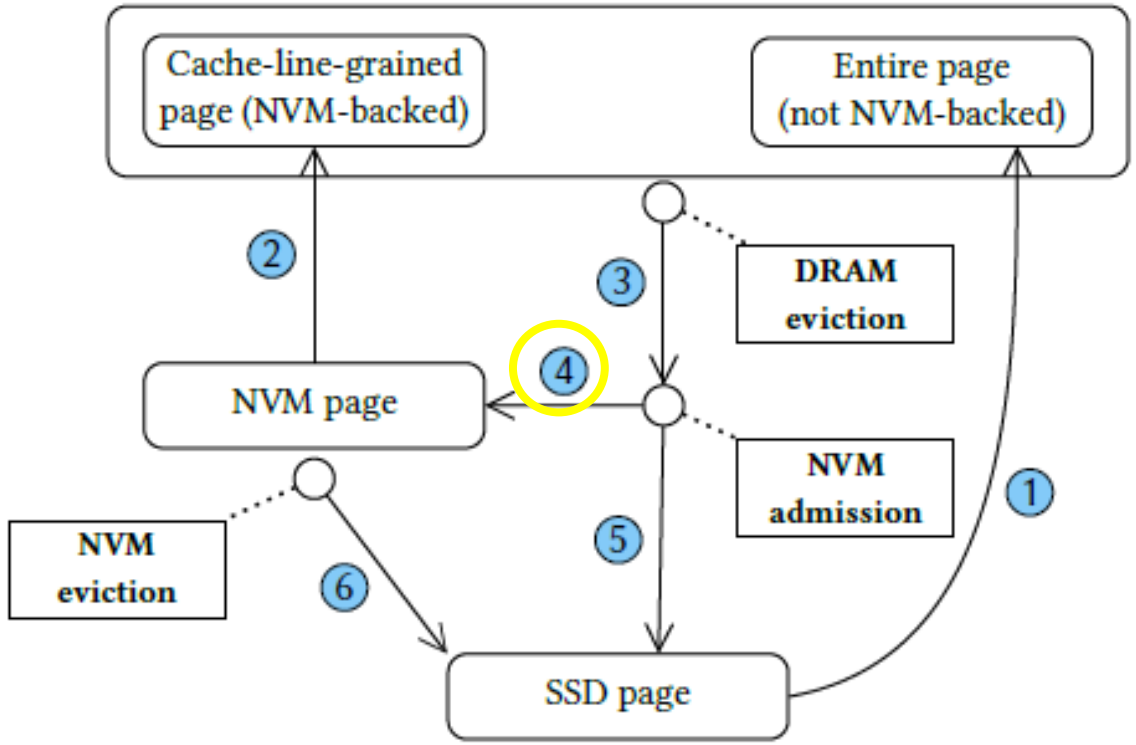


Figure 6: Page Life Cycle – There are five possible page transitions and the three critical decisions (DRAM eviction, NVM admission, and NVM eviction).

(4)

- Admission set
- Admission set의 용도
- Page가 admission queue에 있다면?
- Page가 admission queue에 없다면?

Replacement Strategies (ctnd.)

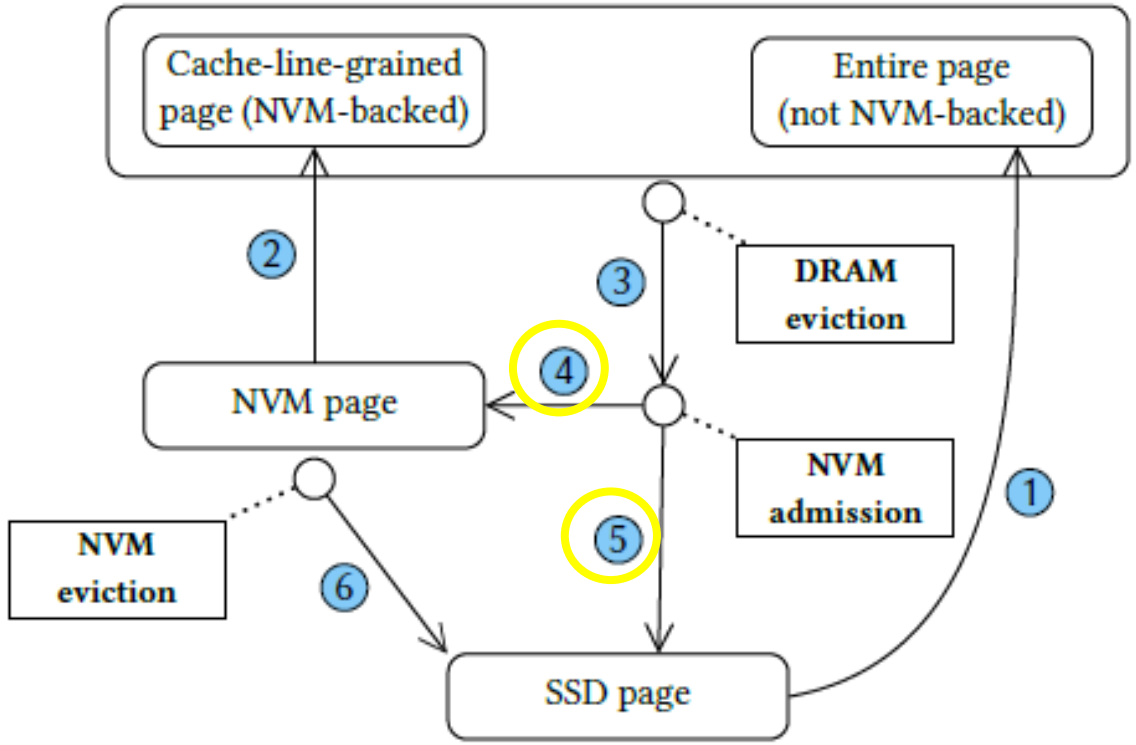


Figure 6: Page Life Cycle – There are five possible page transitions and the three critical decisions (DRAM eviction, NVM admission, and NVM eviction).

(4) (5)

- Admission set
- Admission set의 용도
- Page가 admission queue에 있다면?
- Page가 admission queue에 없다면?
- Limited size of admission set

Replacement Strategies (ctnd.)

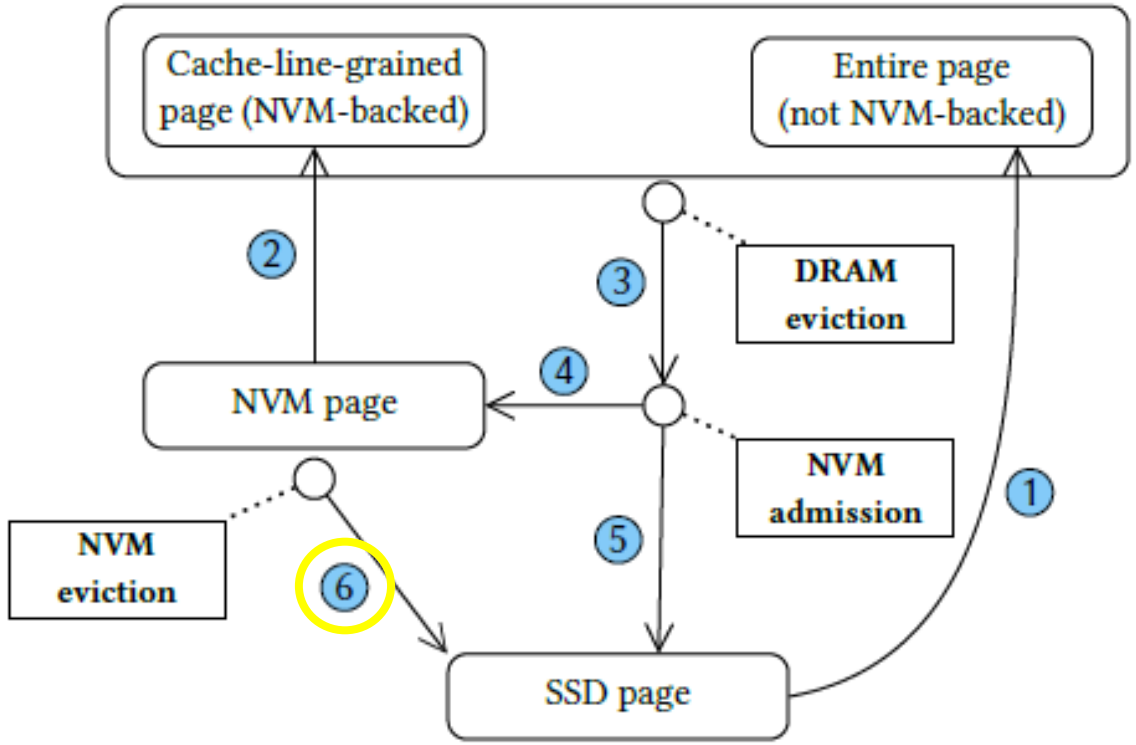


Figure 6: Page Life Cycle – There are five possible page transitions and the three critical decisions (DRAM eviction, NVM admission, and NVM eviction).

(6)

- NVM eviction
- i.e. 새 페이지가 admit되었을 때 기존의 NVM에서 swap out될 page 선정
- Clock algorithm

Combined Page Table

- NVM에 fit된 workload의 경우 3차 레이어 (SSD)는 사용되지 않고 overhead를 만들지도 않는다.
- 이를 하나의 hash table 안에 page identifier->DRAM과 page identifier->NVM location의 mapping을 함께 저장하는 Combined page table을 통해 구현한다.

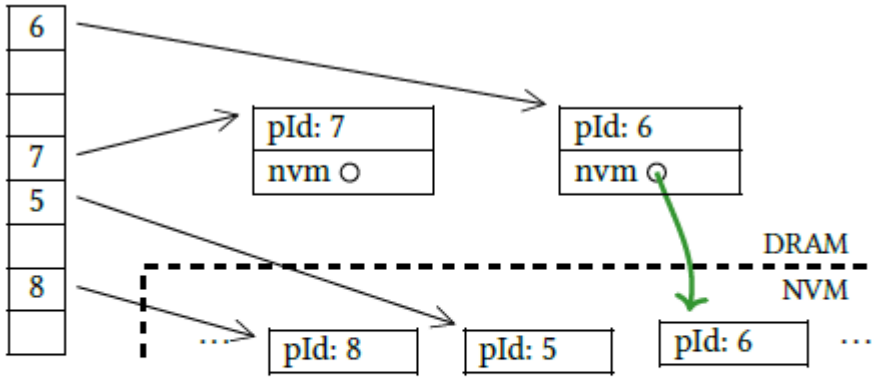


Figure 7: Single-Table Mapping – Using one hash table for DRAM and NVM-resident pages eliminates most overhead for managing the SSD layer. The hash table entries are identified by their location in memory (DRAM or NVM).

System Restart

- Page mapping table은 성능에 크리티컬한 영향을 주기 때문에 DRAM에 저장된다.
- Restart 이후, table은 rebuilt되어야 한다.
- 단점 1) SSD에 저장된 page들의 버전이 NVM 버전보다 오래되었으므로 이를 커버하기 위해 첫 transaction에 드는 시간이 오래걸린다.
- 2) 시스템이 재시작되면, DRAM cache 뿐만 아니라 NVM cache 역시 비었으므로 pre-crash throughput에 도달하는데 더 오랜 시간이 걸린다.
- 이에 대한 대책 : Reconstructing the PMT requires scanning over all NVM pages, reading their page identifiers and adding them to the DRAM-resident page table
- 100GB의 NVM에 대한 page identifiers를 읽는 것은 1초도 걸리지 않지만, 보다 빠른 restart를 보장한다.

Evaluation

Architecture Comparison

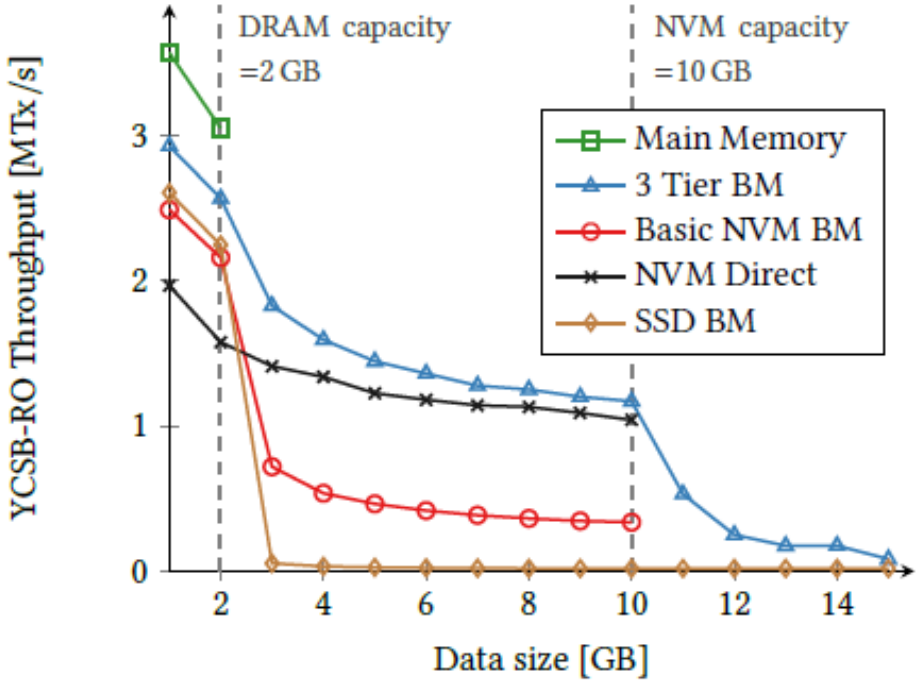


Figure 8: YCSB-RO – Performance for varying data sizes on read-only YCSB workload. The capacity of DRAM, NVM, and SSD is set to 2 GB, 10 GB, and 50 GB, respectively.

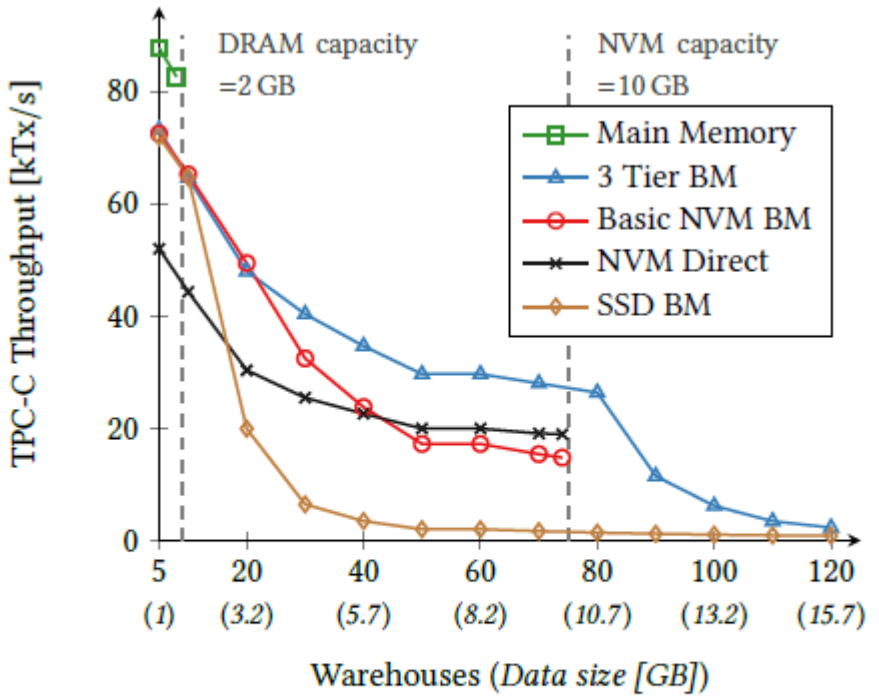
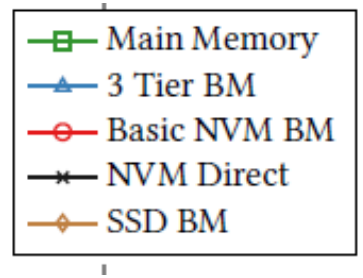


Figure 9: TPC-C – Performance in TPC-C for an increasing number of warehouses. The capacity of DRAM, NVM, and SSD is set to 2 GB, 10 GB, and 50 GB, respectively.

Architecture Comparison (ctnd.)



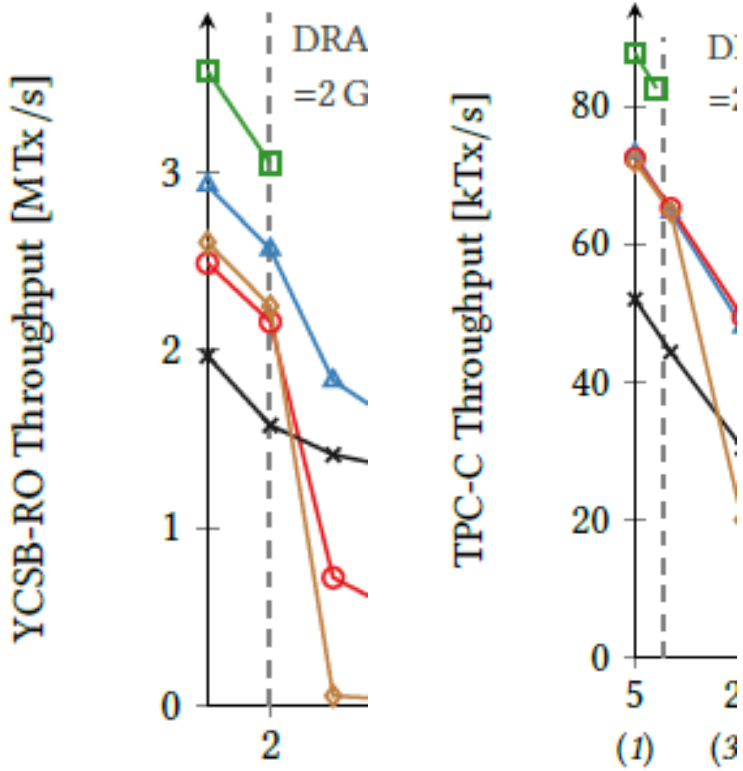
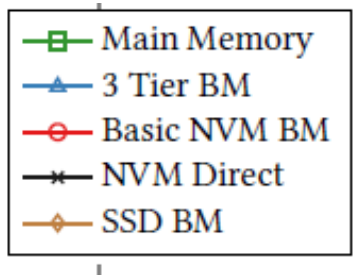


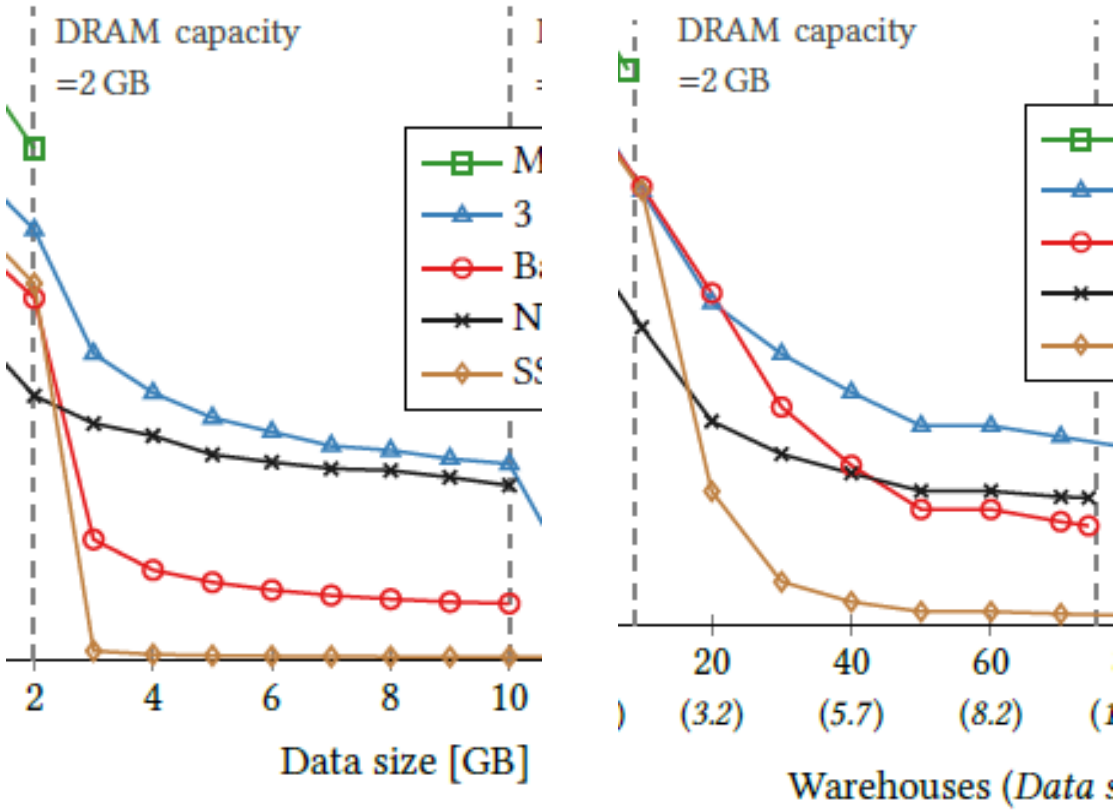
Figure 8: YCSB-RO - Figure 9: TPC-C -

DRAM Area

- Main memory variant의 우월한 성능
- Buffer-managed architectures가 이를 따라잡는 것은 불가능
- 3 tier BM의 경우 individual cache line의 적재 유무를 확인하는 작업은 필수적이므로 완전히 없애지는 못한다.
- Pointer swizzling의 speed up & cache-line-grained access의 speed down 효과
- DRAM이 아닌 느린 NVM을 썼으므로 NVM Direct가 최하위

Architecture Comparison (ctnd.)

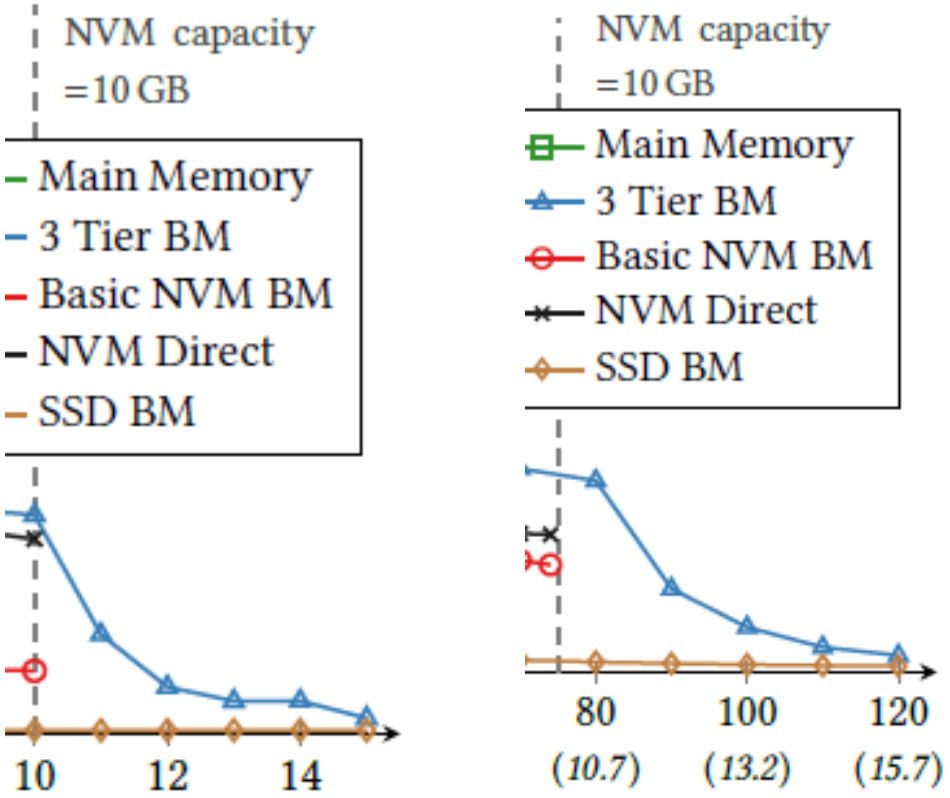




NVM Area

- 크기가 커짐에 따라 performance drop, 그러나 DRAM caching system으로 인해 NVM Direct의 성능을 앞선다.
- YBCB-RO보다 TPC-C에서 감소세가 완만한데, 그 이유는 TPC-C에서 working set (hot data)이 전체 데이터의 일부만을 차지하여 buffer-managed syste에 의해 cache된 것의 효과로 추정된다.

Architecture Comparison (ctnd.)



SSD Area

- SSD에서 더 많은 data를 끌어와야 하므로 performance drop이 관찰된다.
- TPC-C에서 이 감소세가 늦게 일어나는 이유는 NVM에 hot data가 당분간 계속 들어있기 때문이다.
- Unavoidable performance drop

Performance Drill Down

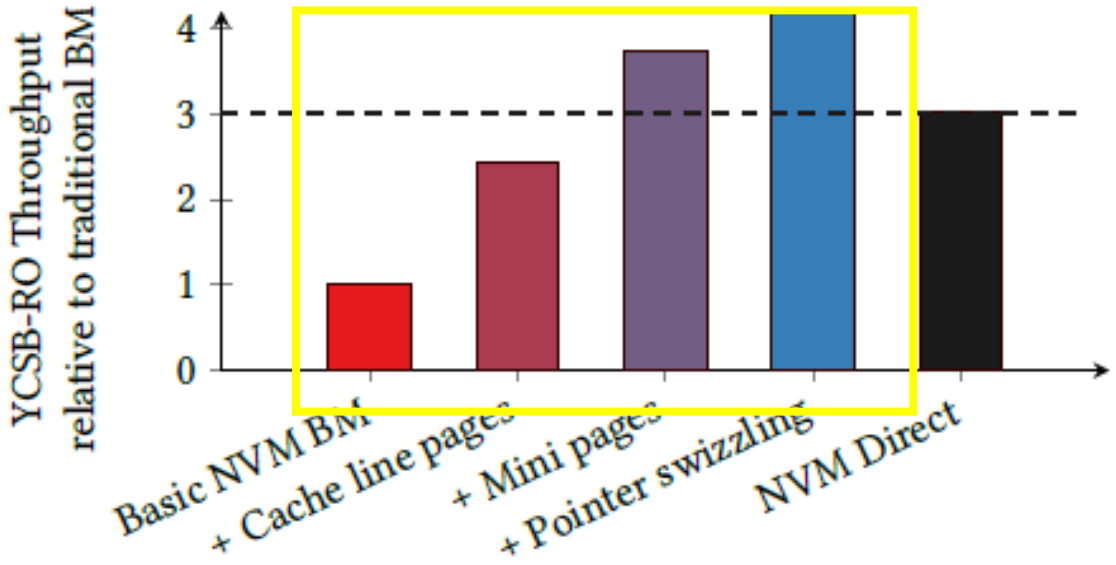


Figure 10: Performance Drill Down – Effect of proposed optimizations relative to a traditional buffer manager on NVM (YCSB-RO with 10 GB of data, read only, 2 GB DRAM, and 10 GB NVM).

	Small Scan (range = 100)	Full Scan (range = |table|)
Basic NVM BM (100%)	50 000 scans/s	0.34 scans/s
+ Cache-Line-Grained	104.2 %	91.3 %
+ Mini Pages	93.1 %	90.8 %
+ Pointer Swizzling	93.8 %	90.9 %

Overhead analysis on YCSB-SCAN

Hybrid Structures

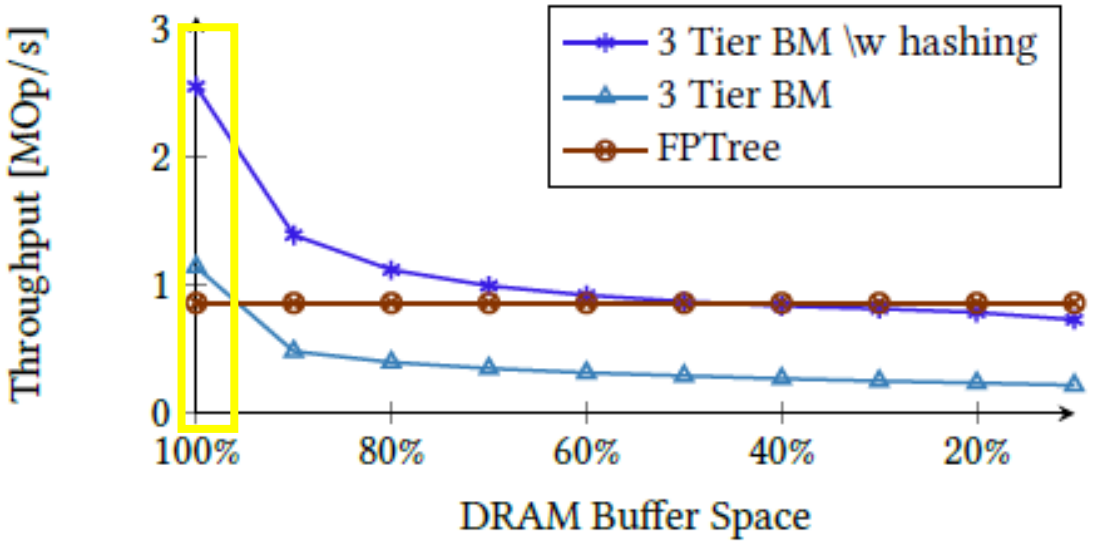


Figure 11: Hybrid DRAM-NVM Systems – Uniformly distributed lookup keys in tree with 100 M 8 byte keys values pairs.

FPTree : leaf node는 NVM에, inner node는 DRAM에 저장하여 빠른 탐색과 높은 duaibility을 가능케하는 B+tree

- Leaf node design에 있어 3-tier는 다소 agnostic.

  따라서 hashing structure로 leaf point lookup을 보완한 결과, 더 작은 DRAM cache size에도 불구하고 competitive한 성과를 보였다.

- Buffer Managed Approach의 장점

Impact of NVM Characteristics

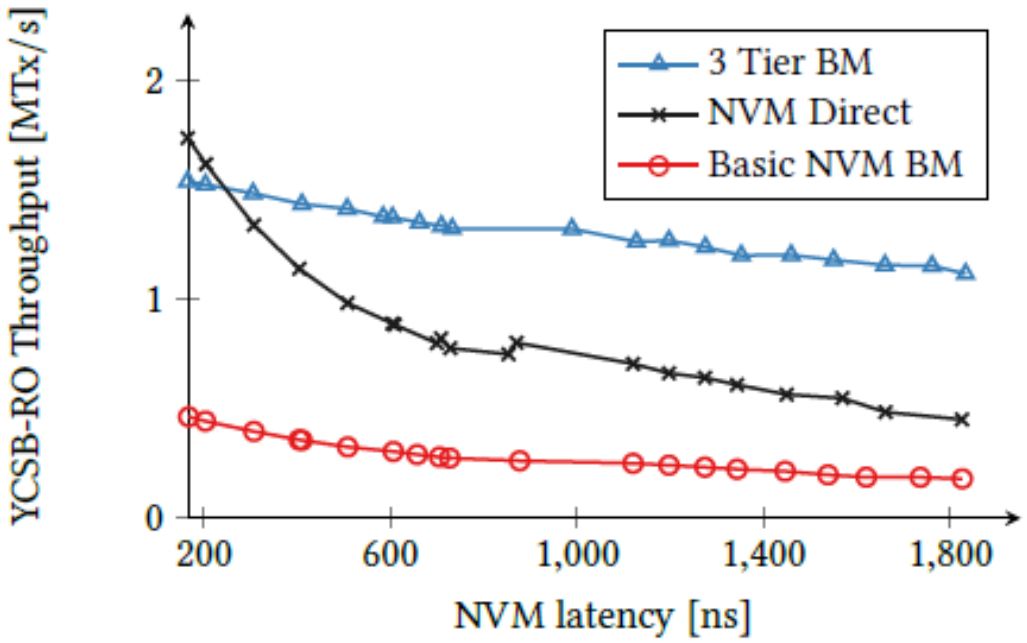


Figure 12: NVM Latency – The impact of varying NVM latencies on the YCSB-RO performance (YCSB with 10 GB of data, read only, 2 GB DRAM, and 10 GB NVM).

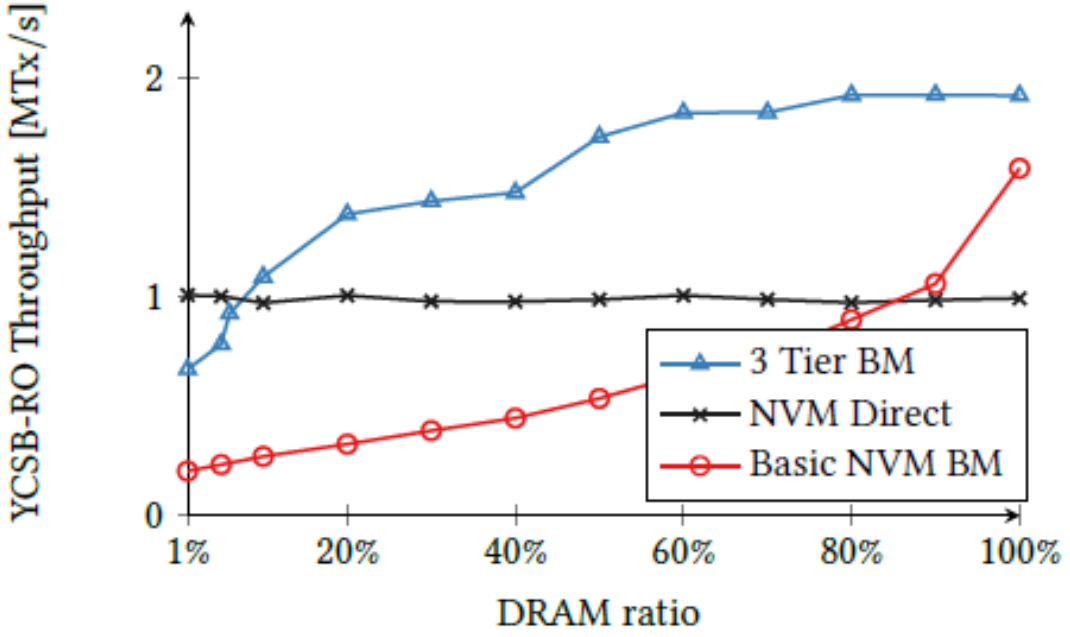


Figure 13: DRAM Buffer Size – YCSB-RO performance for varying amounts of DRAM and a fixed NVM capacity (YCSB with 10 GB of data, read only and 10 GB NVM).

Related Work

Related Work

Name	Characteristics
SOFORT database engine	Copy-on-write architecture for NVM 모든 primary data는 NVM에서 바로 저장되고 수정된다. Secondary data는 main memory에 저장된다.
FOEDUS	NVM에 직접 접근하여 restart time을 줄이지만, higher access latency를 가진다. 2-layered approach : either on NVM or DRAM
SAP HANA in-memory database system	"Delta" and "main" storage separation NVM : immutable, compressed bulk (main) Main memory : recent changes (delta)
Microsoft Siberia	Main memory database system의 capacity를 넓히는 접근 Cold storage를 도입하여 자주 액세스되지 않는 tuple을 옮긴다.
Pmem.io library	NVM management의 de facto standard 다양한 추상화 레벨을 제공한다.

End of Contents