Boosting OLTP Performance with Per-Page Logging on NVDIMM

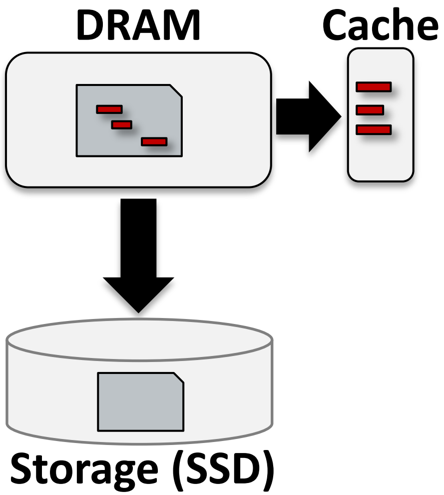
Bohyun Lee[+], **Seongjae Moon**[*], Jonghyeok Park[**], Sang-Won Lee[†]

Technical University of Munich[+] , Sungkyunkwan University[*], Korea University[**], Seoul National University[†]

Background and Motivation

Durability Cost in DBMSs

- Write durability overhead in **OLTP workloads**
  - Page-granularity writes regardless of update size
  - **Small and random updates** → excessive writes to SSDs
  - Reads stall behind slower writes
  - **Writes are a major performance bottleneck**



- How to reduce write traffic to SSDs?
  - Storing only updates in **a fast and durable cache**
  - Improving overall performance: throughput ↑, latency ↓

- Need: a **fast** and **durable cache** to store **small and random updates**

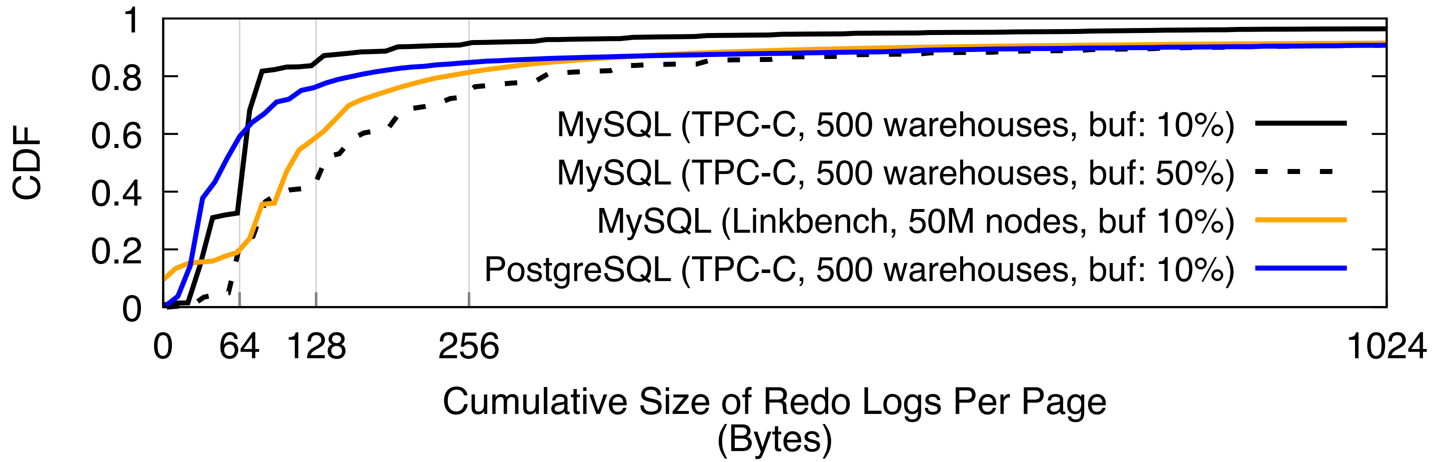
Non-volatile DIMMs

Storage Media	Random IOPS (Unit)	Read	Write	Unit Capacity	Unit Price	$/GB
DRAM [71]	-	-	-	32GB	$180	$5.6
NVDIMM [62]	(256B)	25,523K	27,827K	32GB	$540	$16.9
DCPMM [81]	(256B)	6,164K	1,905K	128GB	$695	$5.4
SSD [72]	(4KB)	555K	28K	1024GB	$310	$0.3

3x expensive!

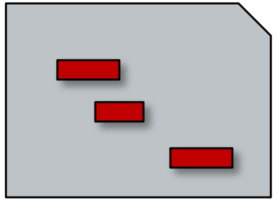
- NVDIMM-N as a fast and durable cache (vs. DCPMM)
  - **Lower** write latency and **higher write** bandwidth
  - **Finer** access granularity (64B vs. 256B)
  - **Less** read/write asymmetry
- **Challenge**: efficient utilization of limited NVDIMM space
  - Small capacity, 3x costlier $/GB than DRAM

Per-Page Update Size in OLTP Workloads



- Small update footprint per page write
  - ≤ 64B: 30% of TPC-C page writes, 20% of LinkBench page writes
  - ≤ 256B: 90% of TPC-C page writes, 80% of LinkBench page writes

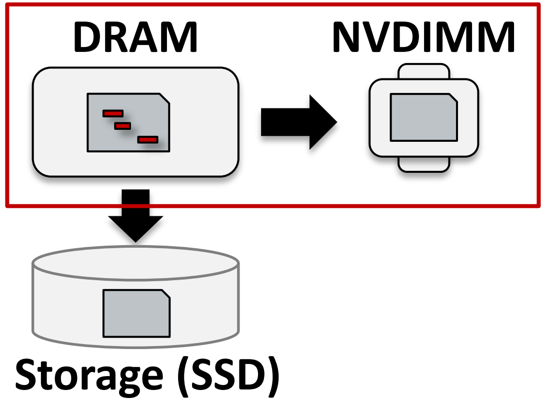
- Durability overhead with DRAM-SSD memory hierarchy
  - **16x write amplification**



4KB page
(update propagation unit)

Durability Cost: SSD vs. NVDIMM

Approach	Storage	IOPS (Write)	Durability granularity	Durability Cost
Vanilla	SSD	28K	4KB Page	$0.011
NV-SQL(VLDB 2023)	NVDIMM-N	**27,827K**	**4KB Page**	$0.000065
NV-PPL			**256B redo logs**	**$0.000004**

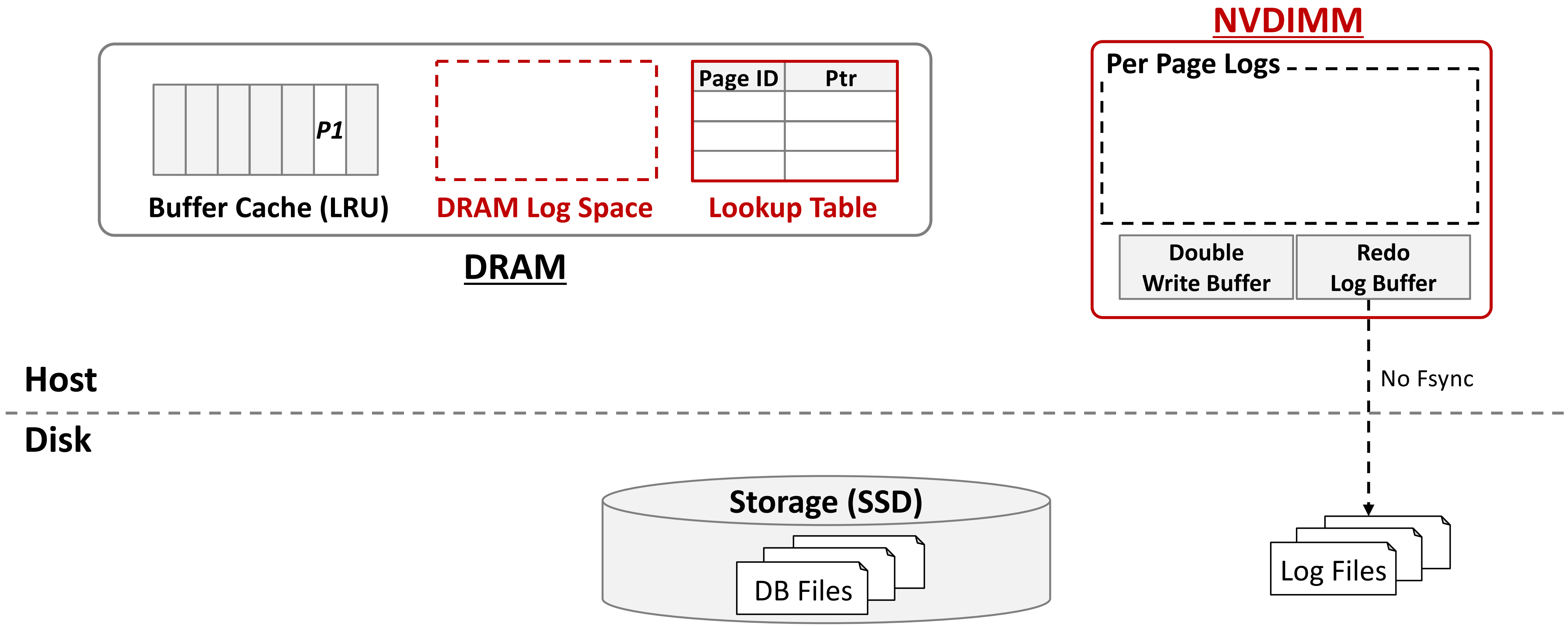


- Caching only the per-page redo logs in NVDIMM
  (based on the five-minute rule)
  - **Cost: 2,750×** and **16× cheaper** than other approaches
  - **Speed: 1,000x** faster durability speed than SSD-based approach

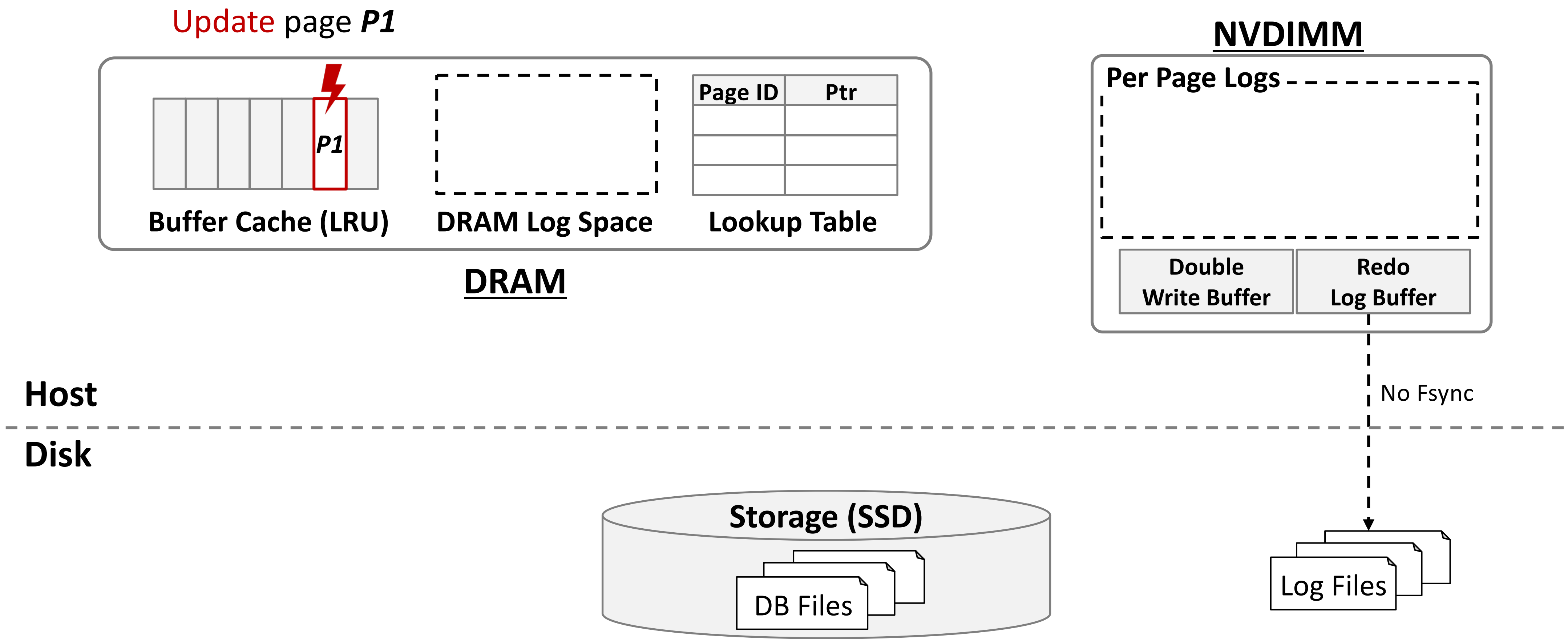
- Simply caching pages themselves in NVDIMM is not cost-effective

NV-PPL: Per-Page Logging on NVDIMM

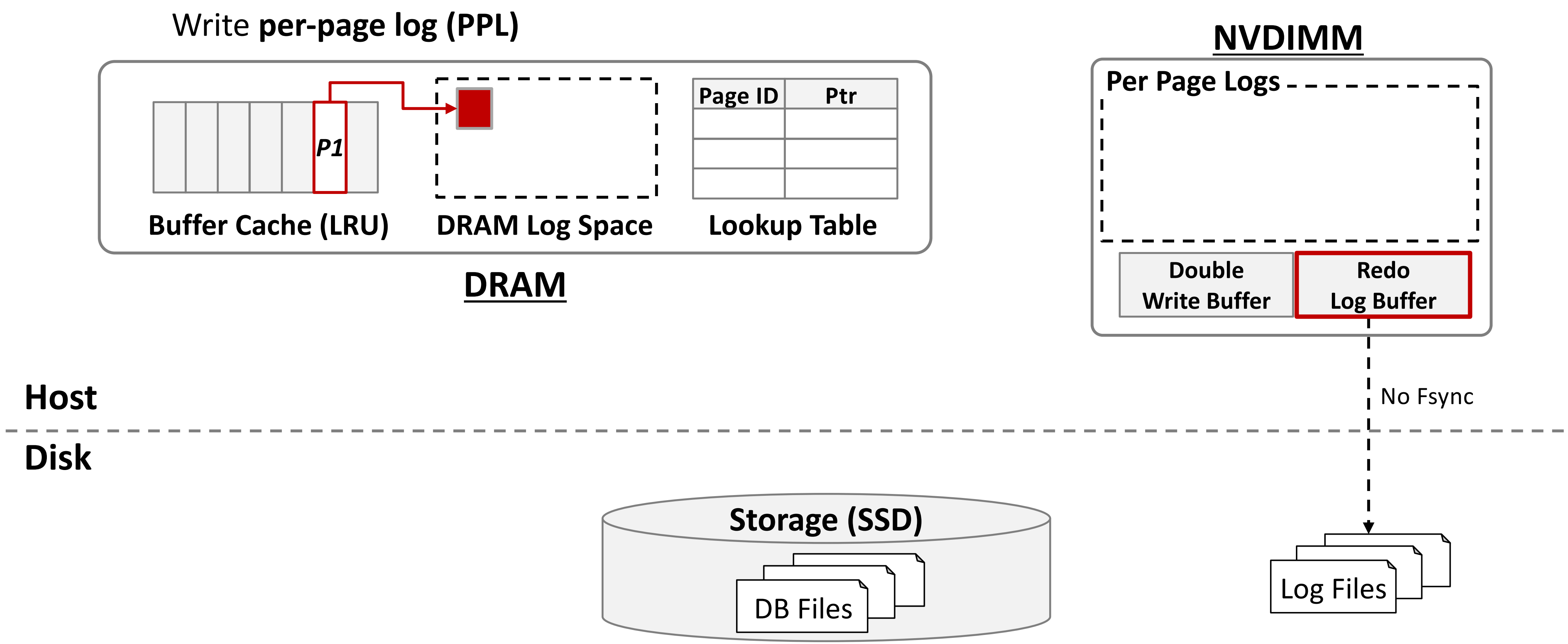
Basic Framework



Basic Framework: Normal Page Update

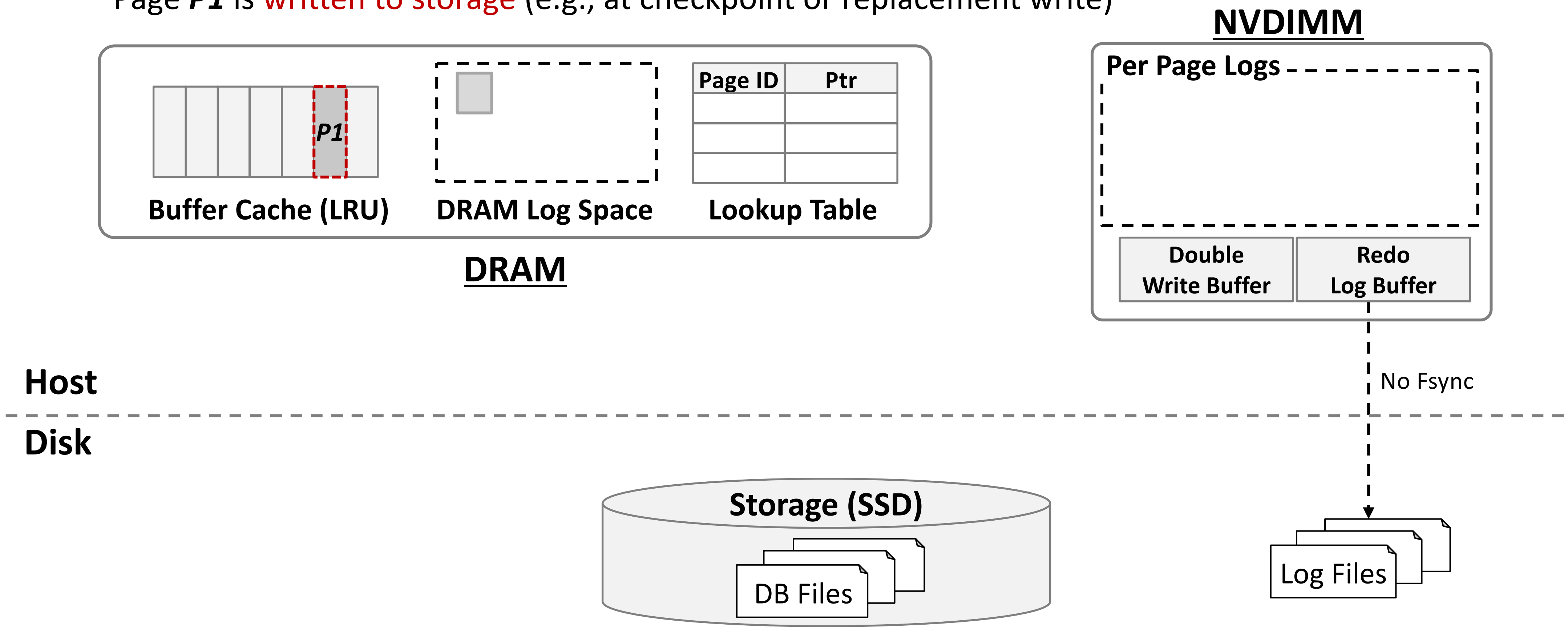


Basic Framework: Normal Page Update

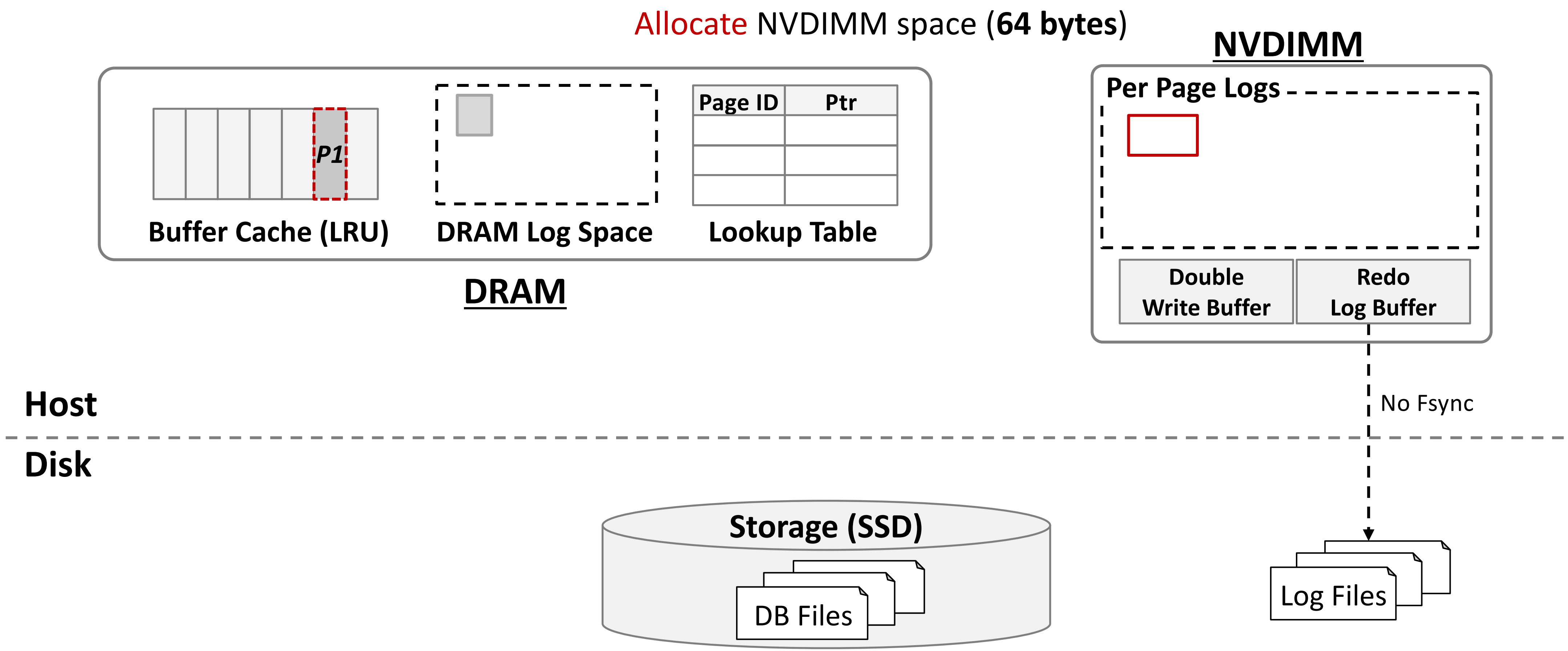


Basic Framework: Normal Page Write

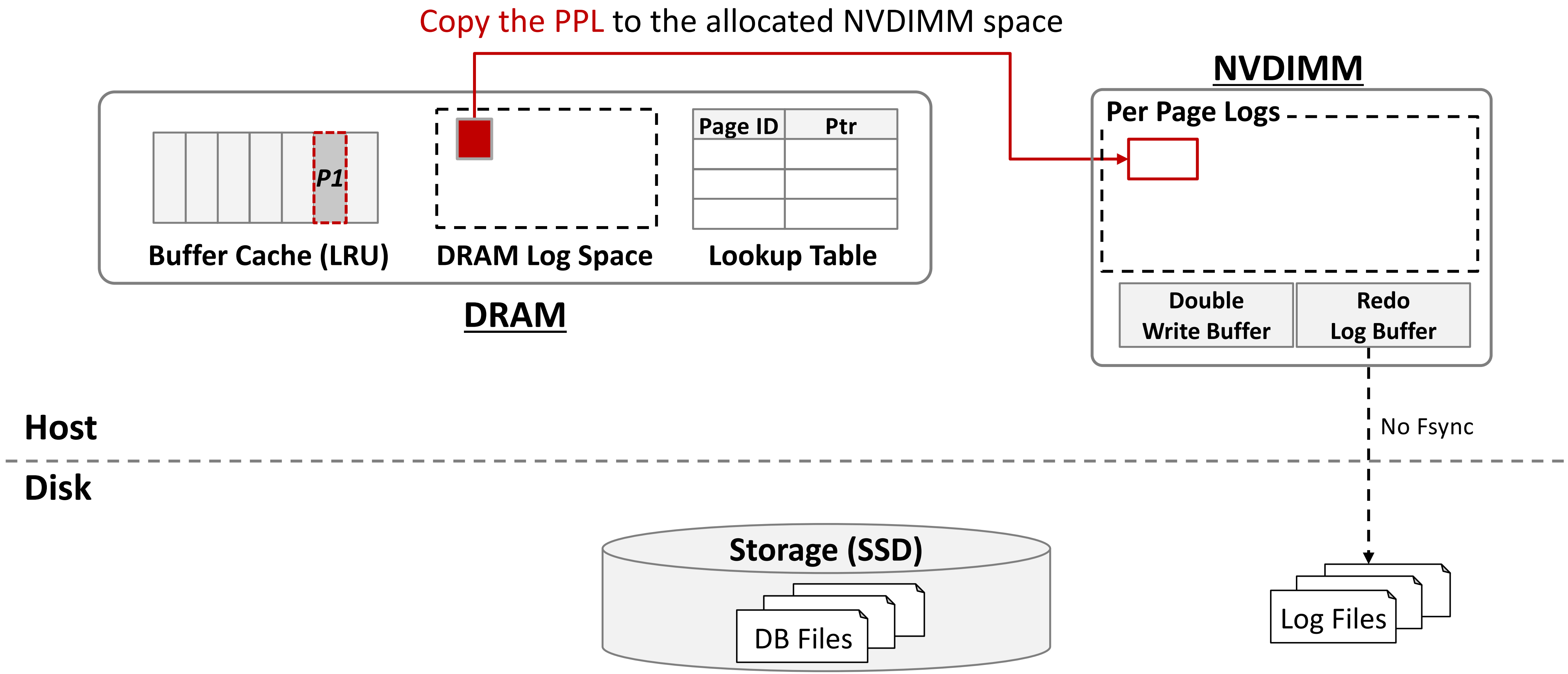
Page ***P1*** is written to storage (e.g., at checkpoint or replacement write)



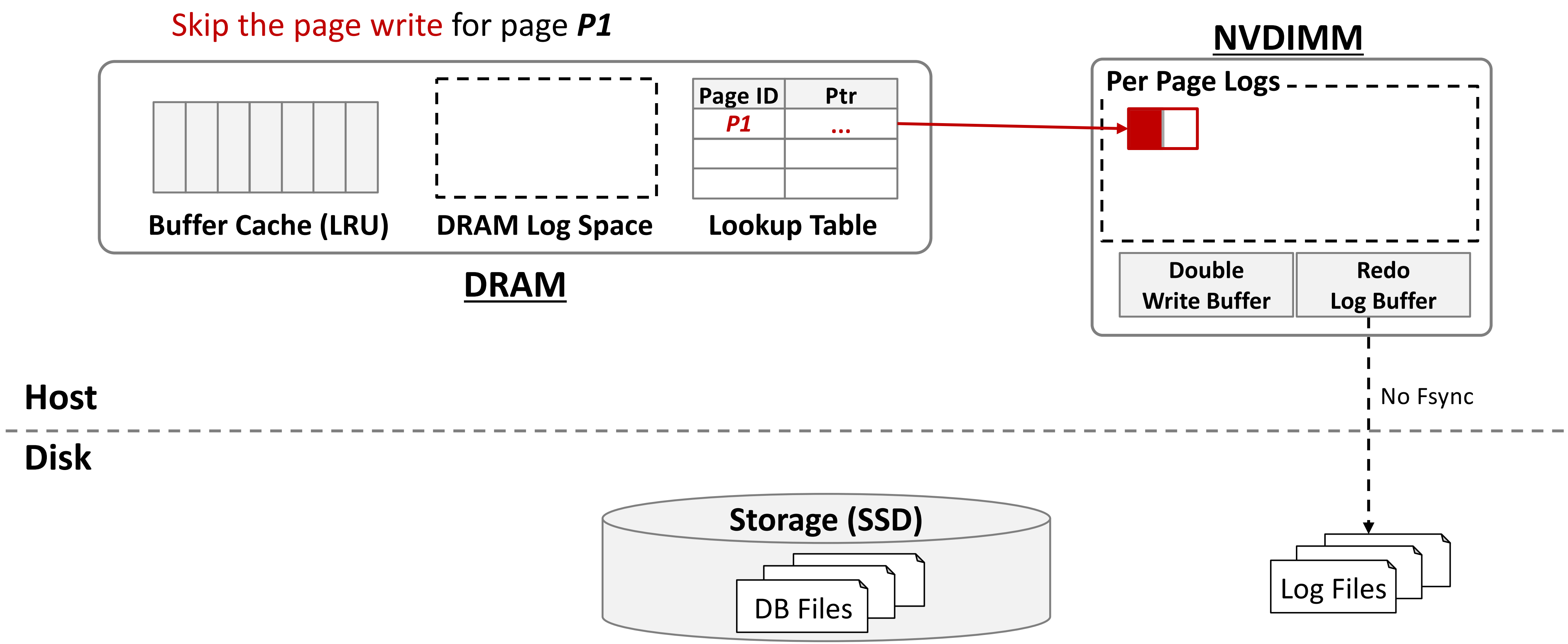
Basic Framework: Normal Page Write



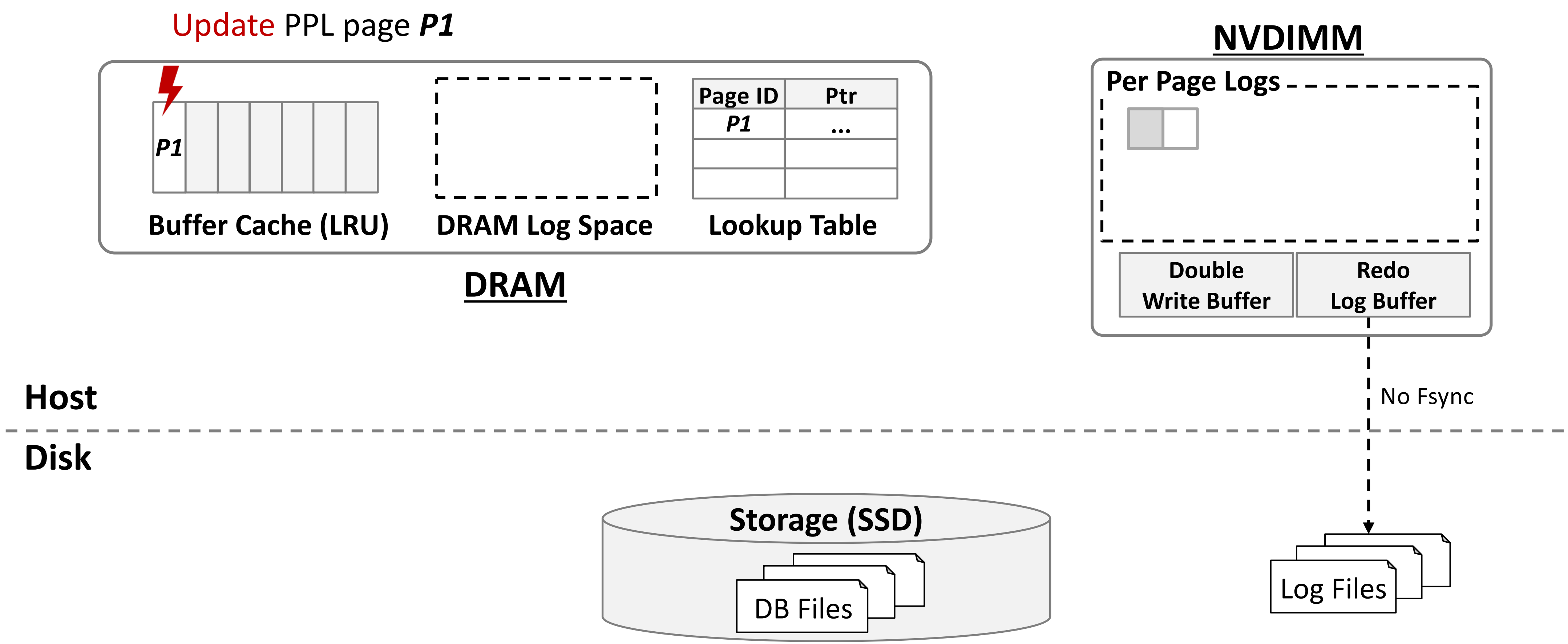
Basic Framework: Normal Page Write



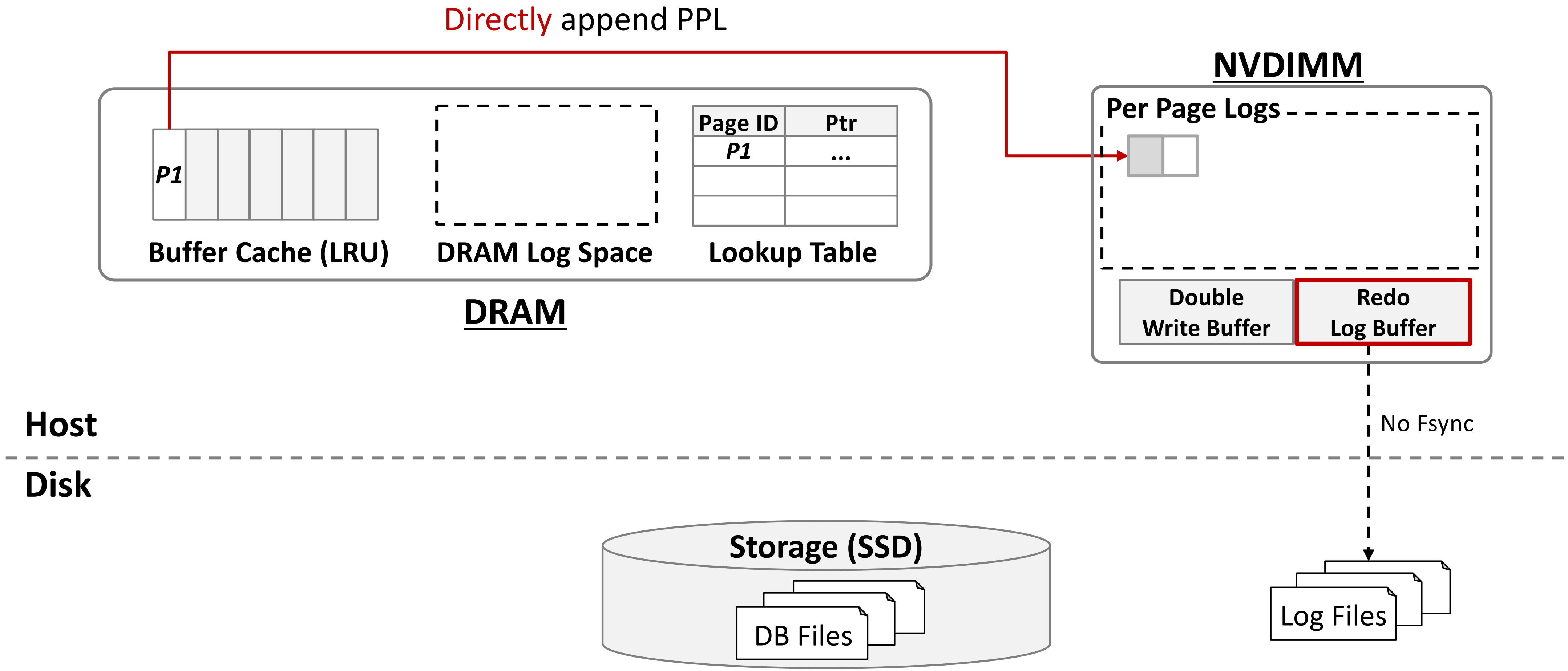
Basic Framework: Normal Page Write



Basic Framework: PPL Page Update

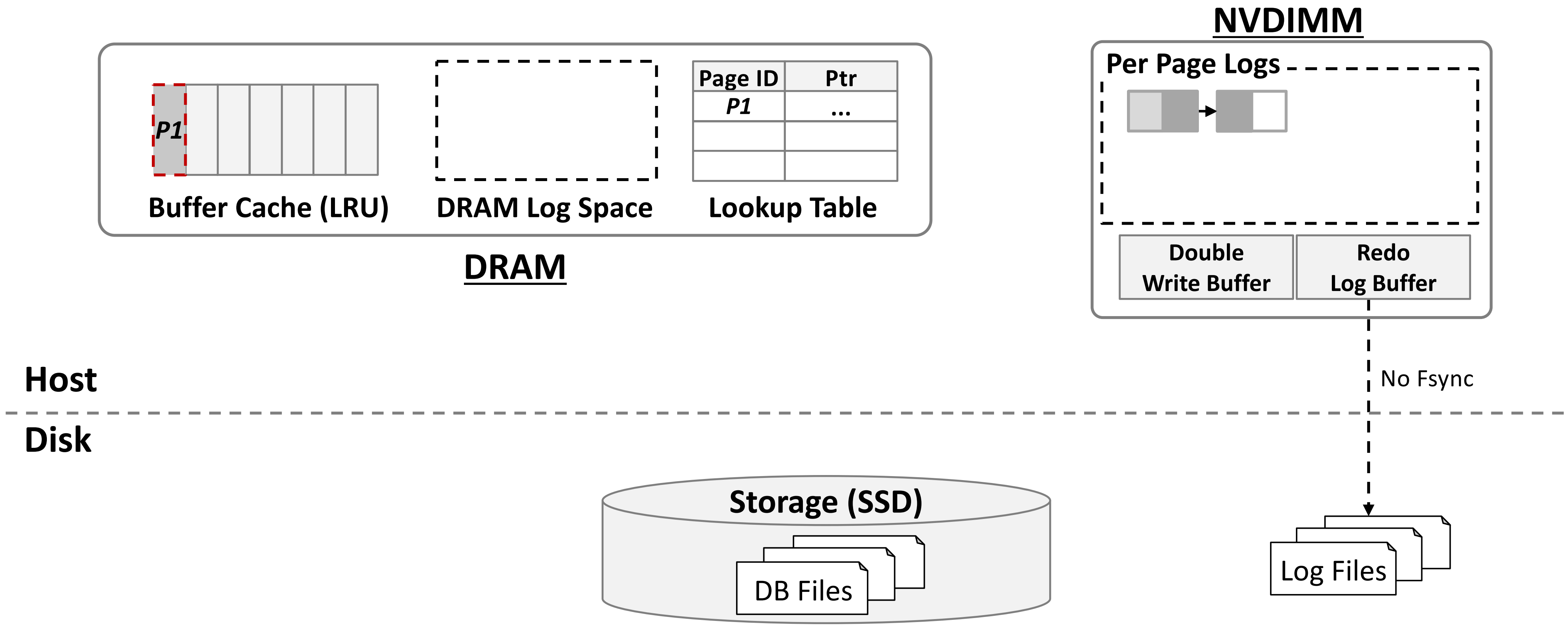


Basic Framework: PPL Page Update

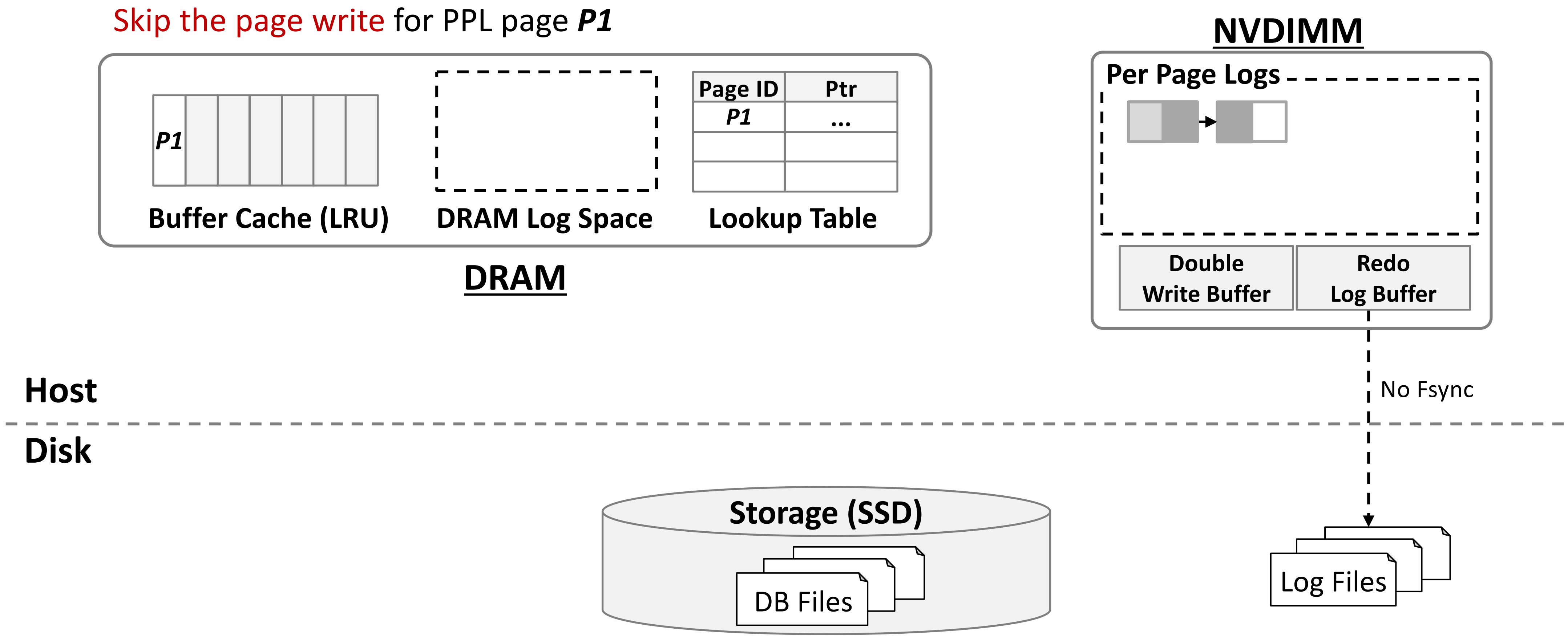


Basic Framework: PPL Page Write

PPL page ***P1*** is written to storage



Basic Framework: PPL Page Write

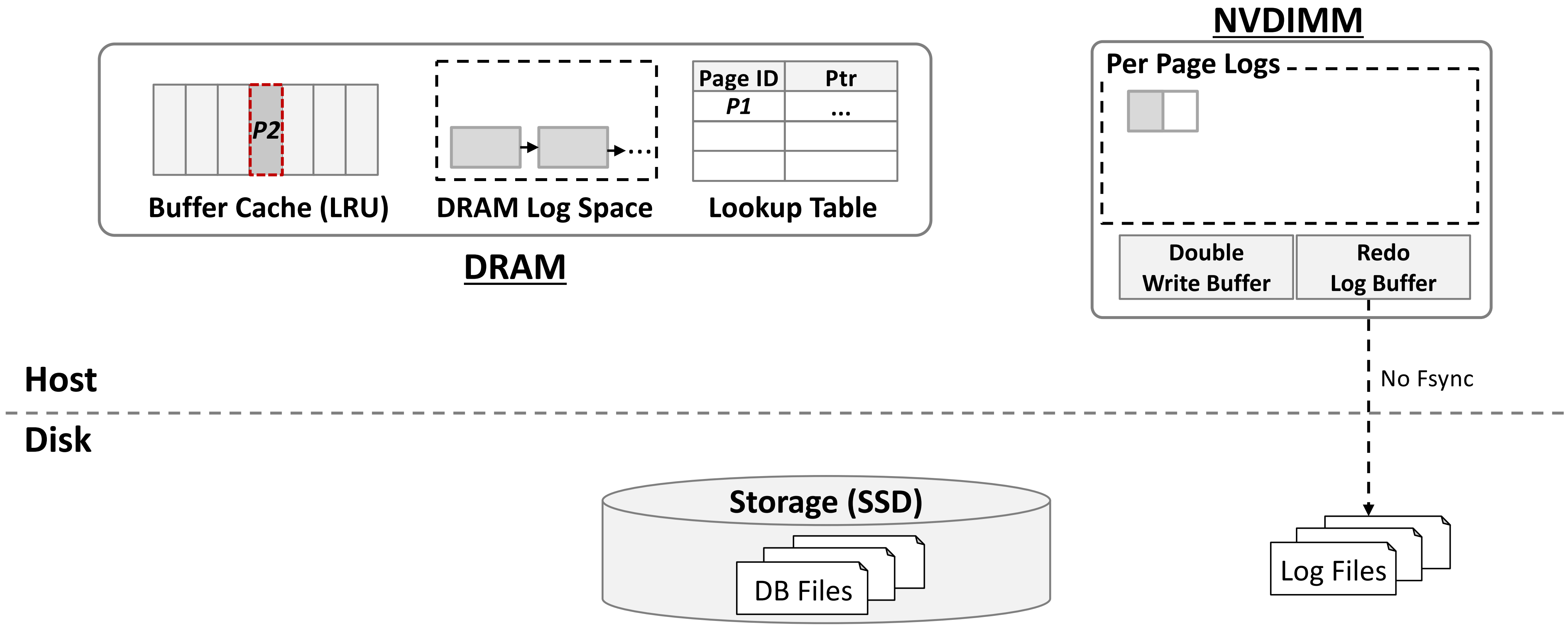


NV-PPL Issues

- **Reducing page writes** by caching PPLs in NVDIMM
- **Issues**
  - How to efficiently use expensive NVDIMM space?
    - **Solution**: Selectively allocating NVDIMM space to the page
      - **Total size of PPLs > 256 bytes** → write page to SSD

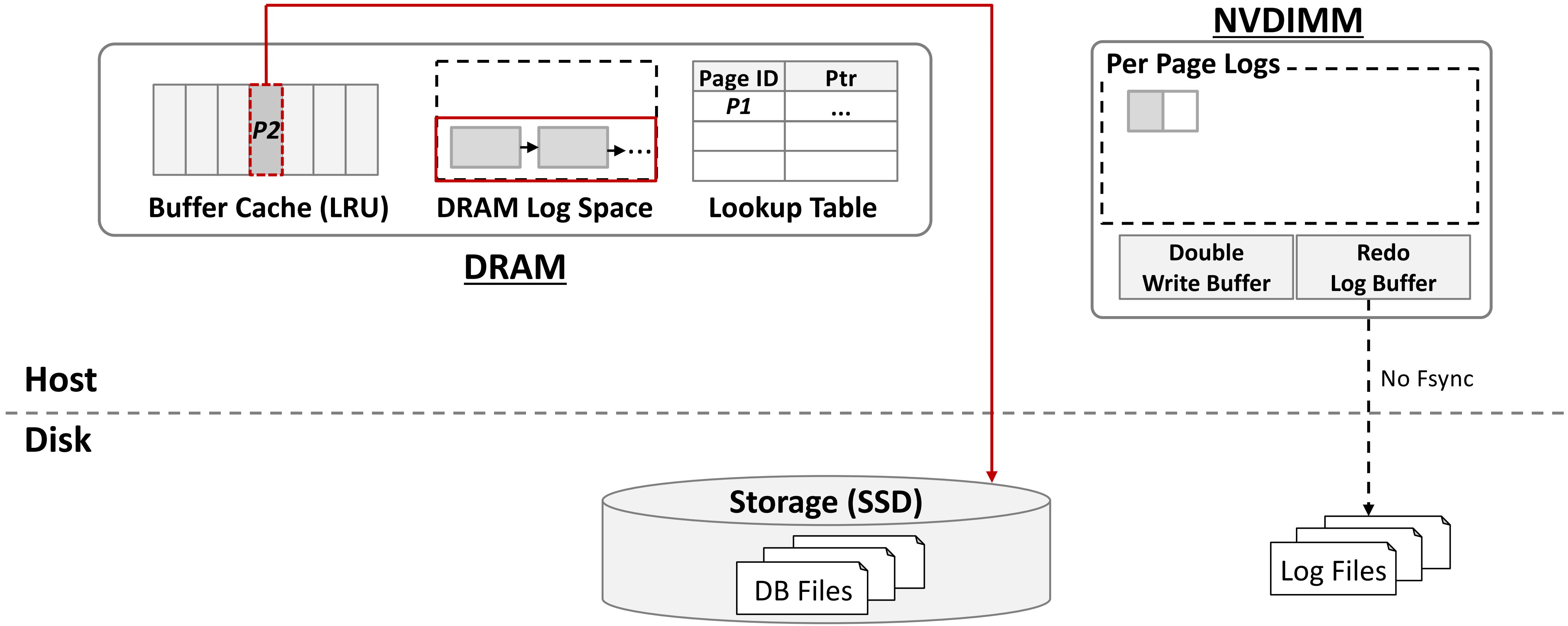
Selectively allocating NVDIMM space: Normal Page

Page ***P2*** is written to storage

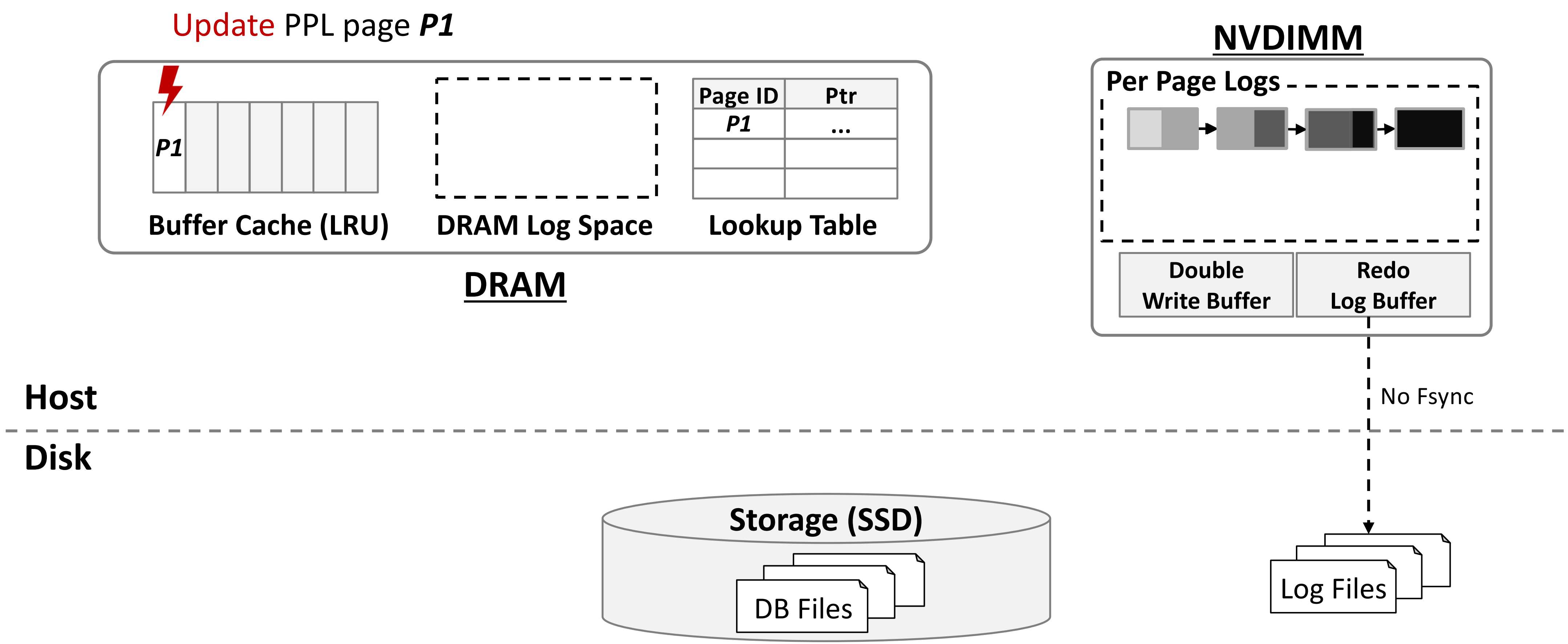


Selectively allocating NVDIMM space: Normal Page

Write page ***P2*** to storage $(PPLs > 256B)$
to avoid wasting NVDIMM space

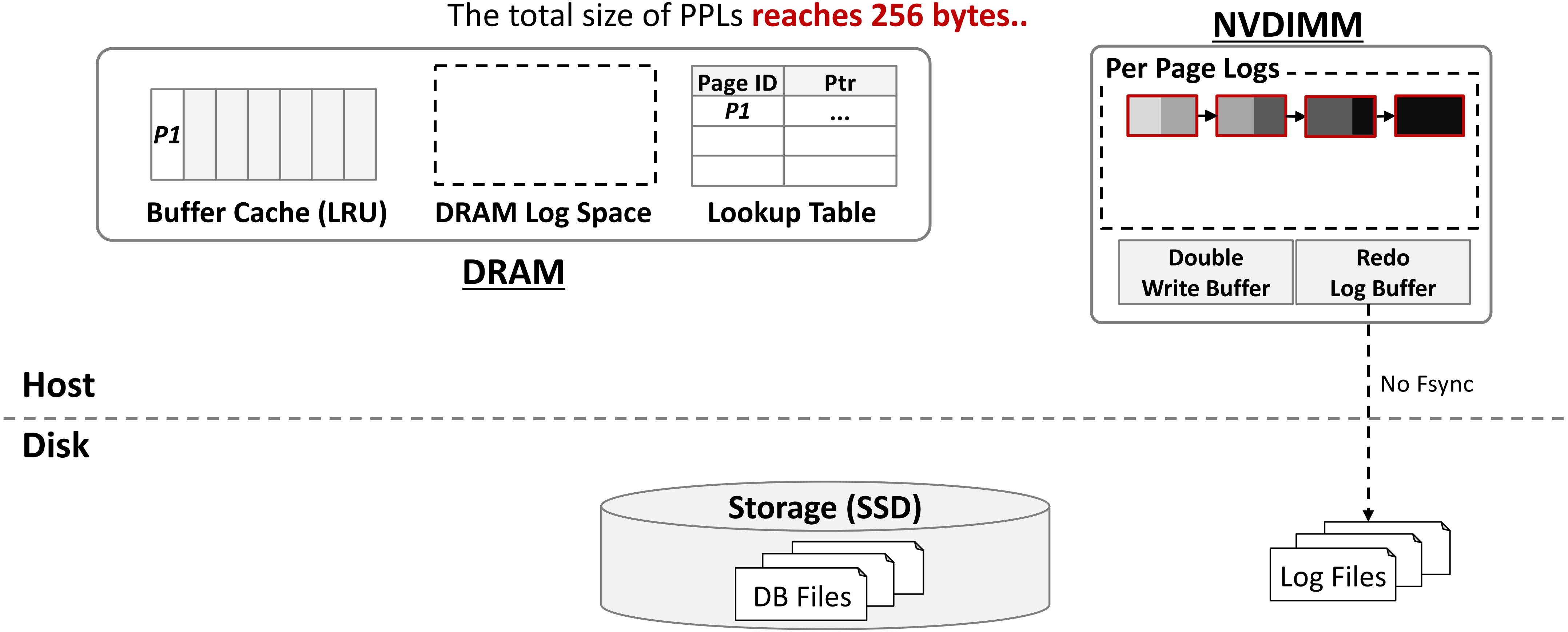


Selectively allocating NVDIMM space: PPL Page

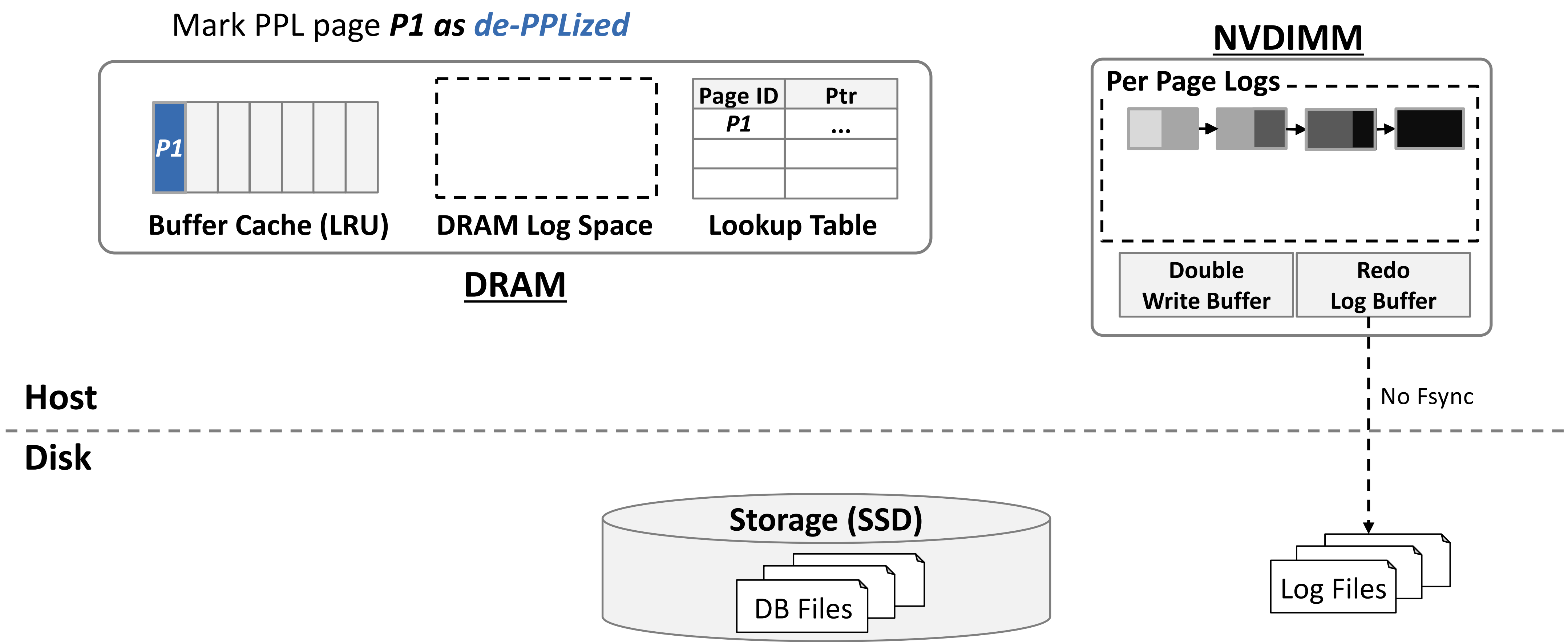


Selectively allocating NVDIMM space: PPL Page

The total size of PPLs **reaches 256 bytes..**

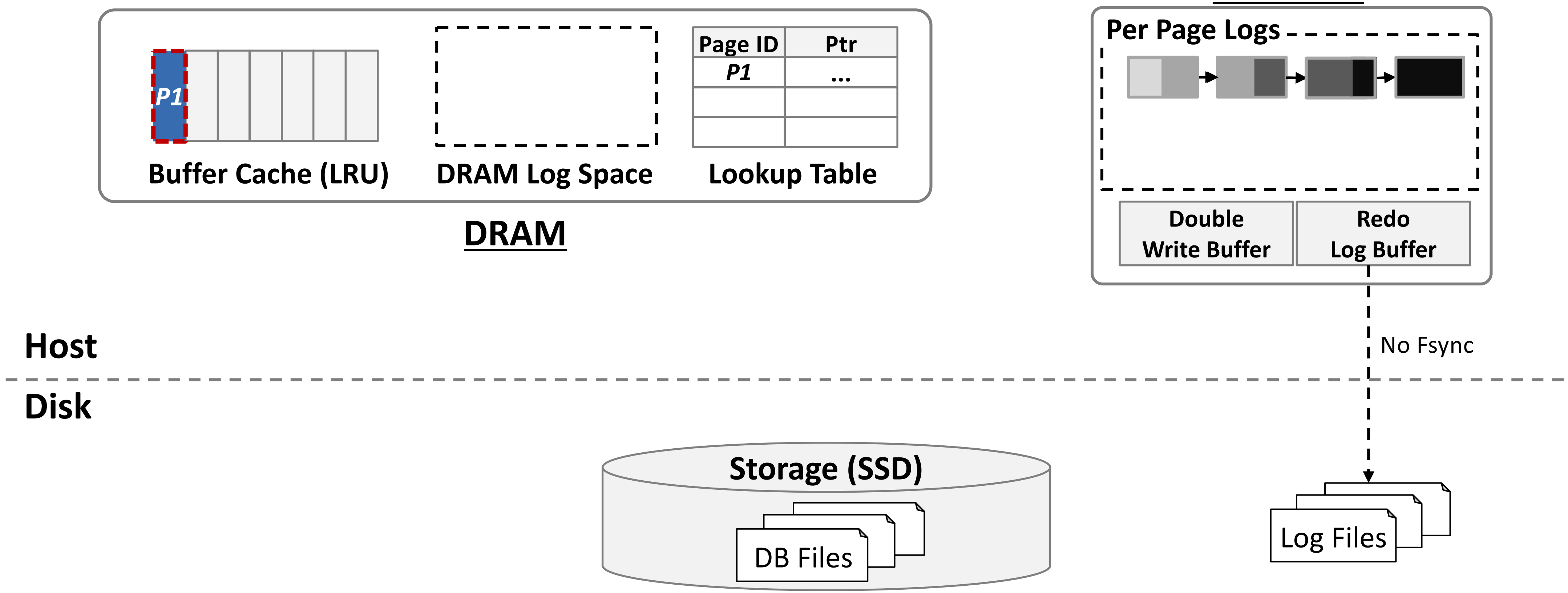


Selectively allocating NVDIMM space: PPL Page

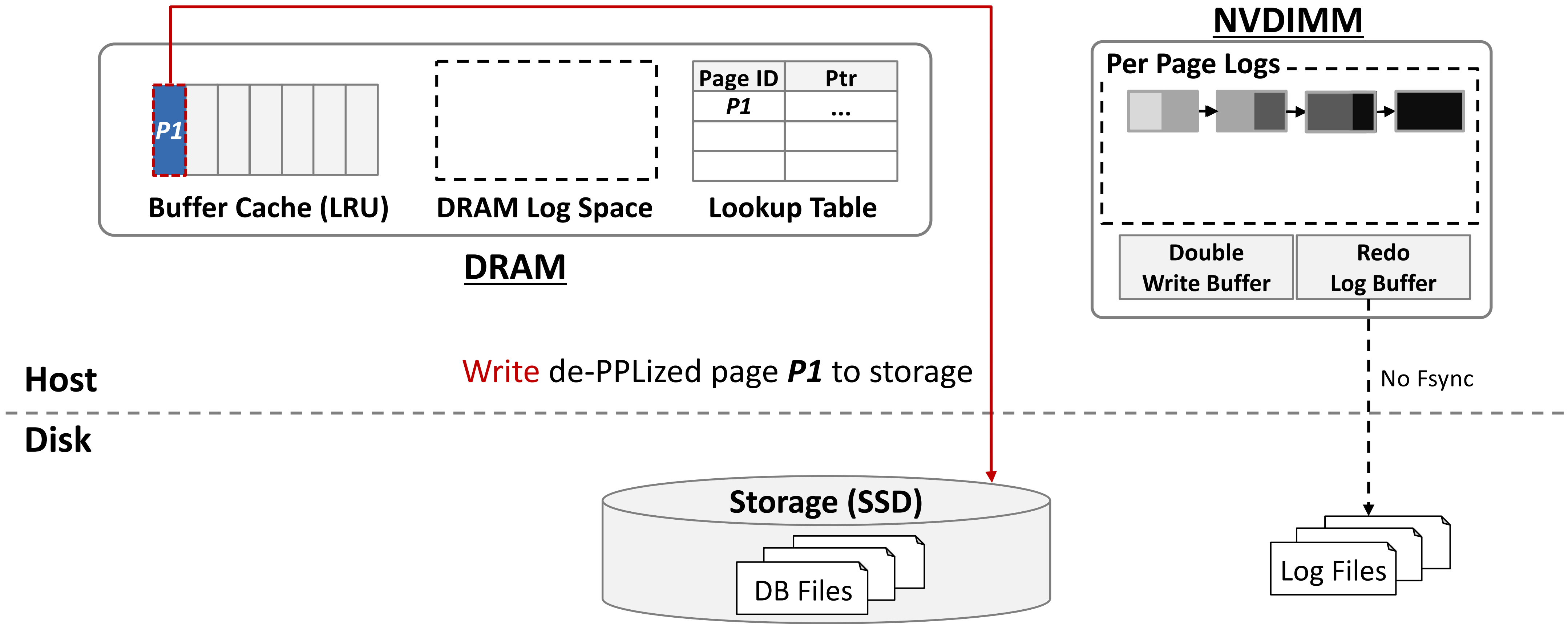


Selectively allocating NVDIMM space: PPL Page

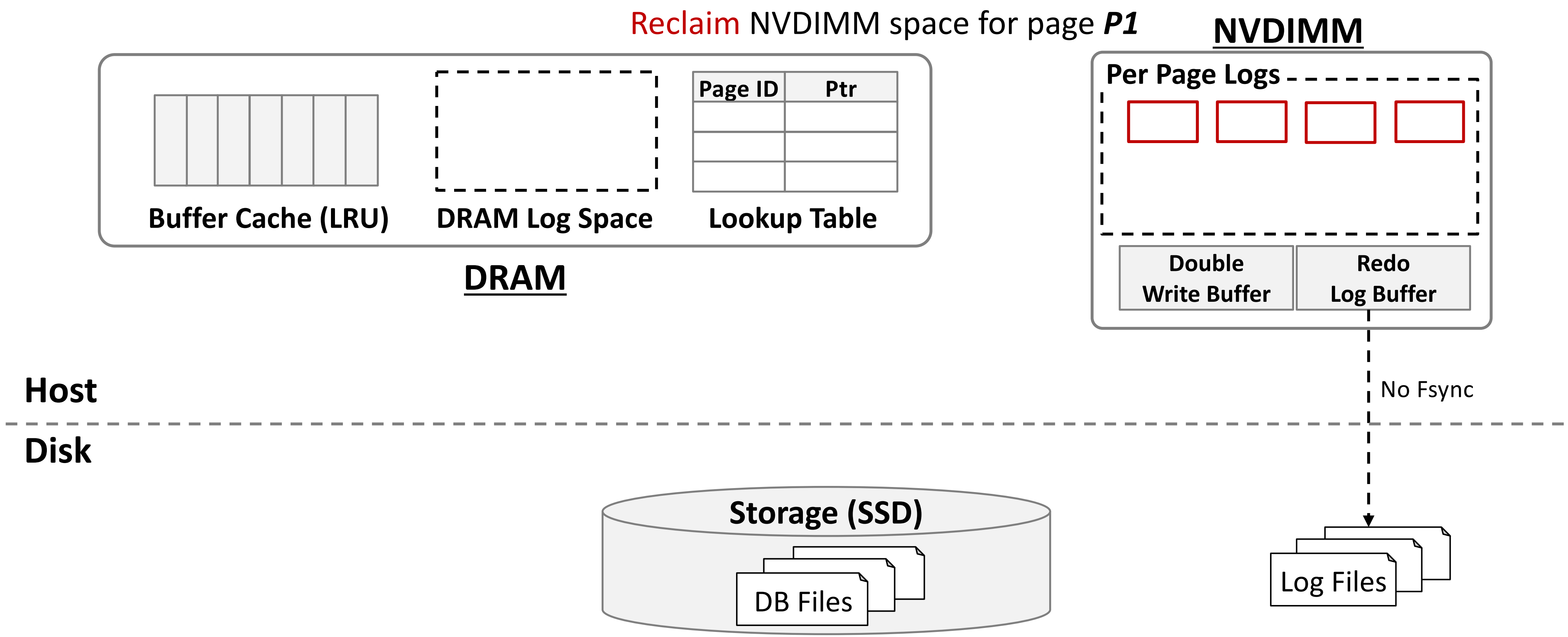
de-PPLized page ***P1*** is written to storage



Selectively allocating NVDIMM space: PPL Page



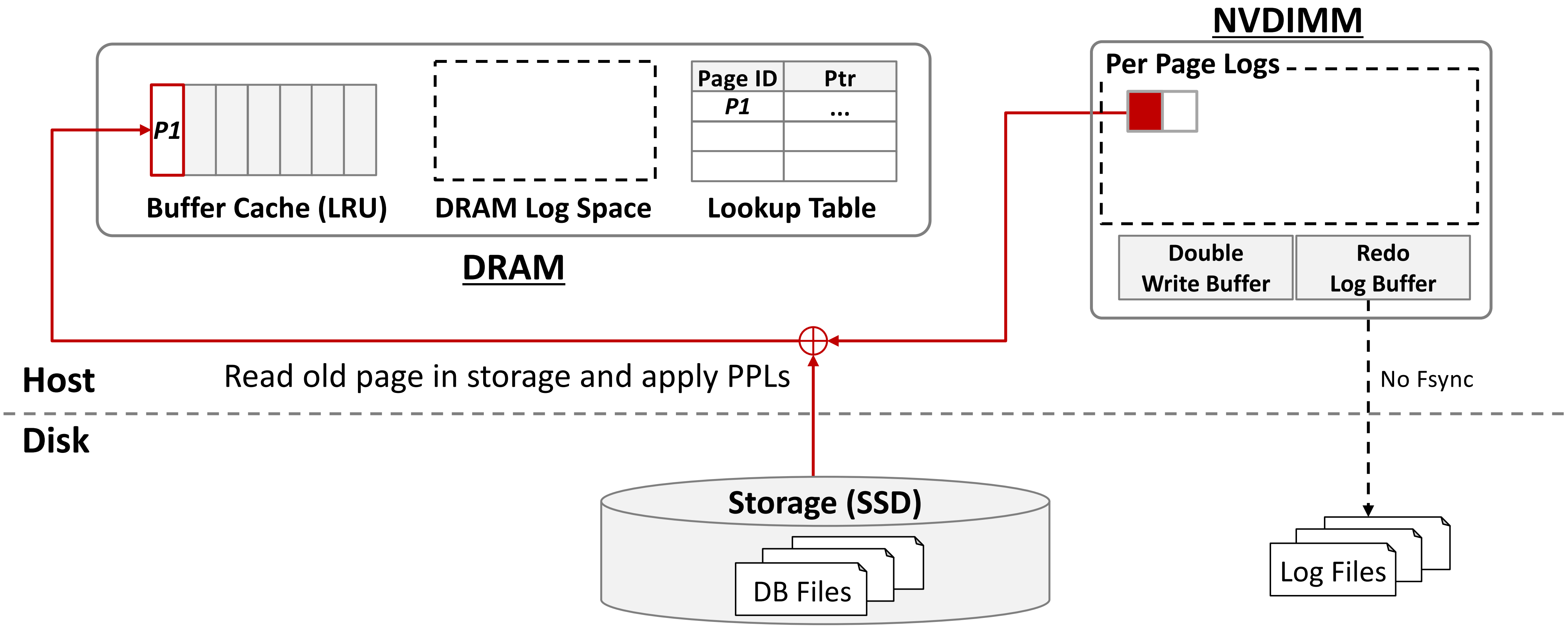
Selectively allocating NVDIMM space: PPL Page



NV-PPL Issues

- **Reducing page writes** by caching PPLs in NVDIMM
- **Issues**
  - How to efficiently use expensive NVDIMM space?
    - **Solution**: Selectively allocating NVDIMM space to the page
      - **Total size of PPLs > 256 bytes** → write page to SSD
  - Increased read latency when reading PPL page from storage
    - NV-PPL **needs to apply PPLs** to the old page from storage

PPL Page Read



NV-PPL Issues

- **Reducing page writes** by caching PPLs in NVDIMM
- **Issues**
  - How to efficiently use expensive NVDIMM space?
    - **Solution**: Selectively allocating NVDIMM space to the page
      - **Total size of PPLs > 256 bytes** → write page to SSD
  - Increased read latency when reading PPL page from storage
    - Total size of PPLs is limited ($\leq 256\ bytes$)
    - **Write reduction gains far outweigh these overheads**
      - (e.g., Read latency +7.8%, extra CPU usage +2.7%)

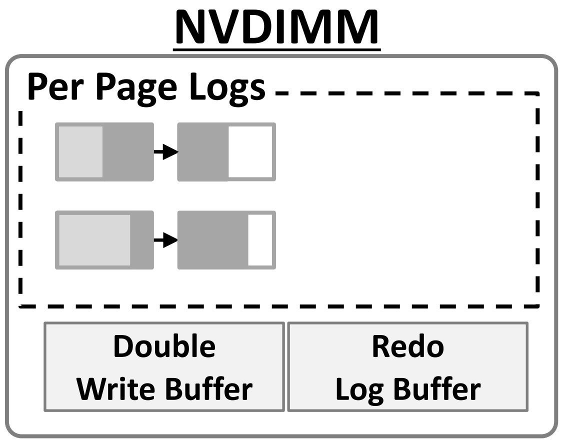
NV-PPL Issues

- **Reducing page writes** by caching PPLs in NVDIMM
- **Issues**
  - How to efficiently use expensive NVDIMM space?
    - **Solution**: Selectively allocating NVDIMM space to the page
      - **Total size of PPLs > 256 bytes** → write page to SSD
  - Increased read latency when reading PPL page from storage
    - Total size of PPLs is limited ($\leq\ 256\ bytes$)
    - **Write reduction gains far outweigh these overheads**
      - (e.g., Read latency +7.8%, extra CPU usage +2.7%)
  - Limited NVDIMM space
    - Solution: **The NVDIMM space reclamation**
    - **Foreground reclamation**: targeting PPL pages in the buffer that use a lot of NVDIMM space
    - **Background PPL cleaner**: targeting unused PPL pages not in the buffer

NVDIMM-worth Objects

- Redo log buffer
  - Eliminates overhead of *log-force-at-commit* and *write-ahead-log* protocols
  - Removes log flushing latency from the critical path of transaction execution

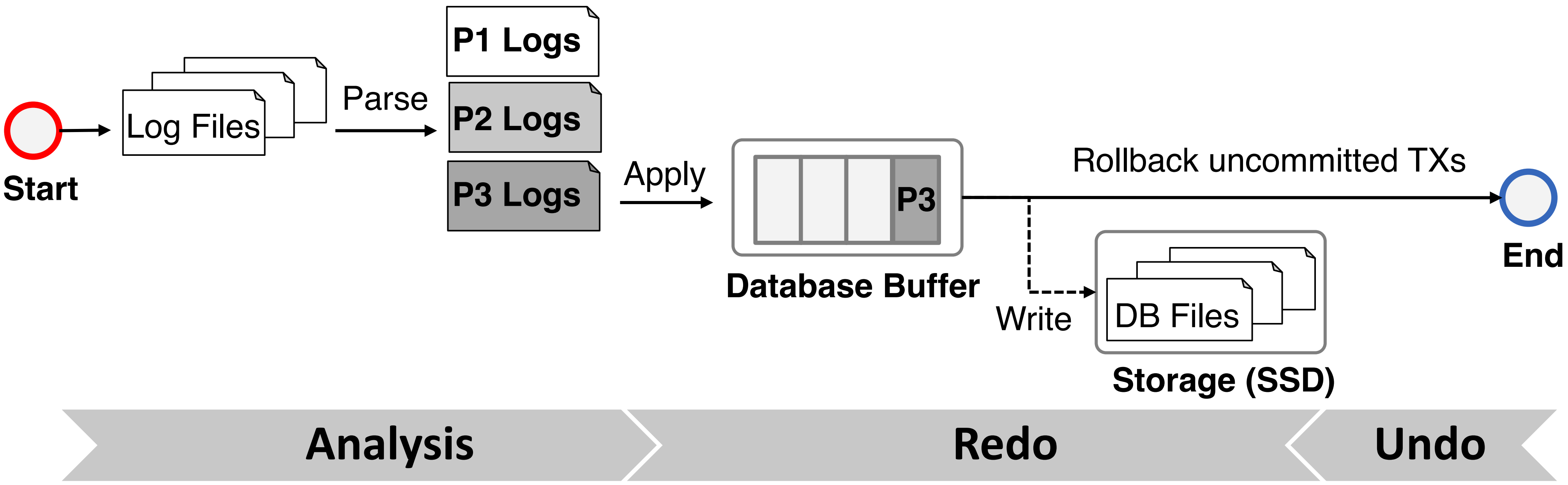
- Double write buffer (DWB)
  - Ensure atomic propagation of multiple pages using single-write journaling
  - Remove redundant write to SSD
  - Replace Disk I/O (**200 us**) → Memory I/O (**200 ns**)



Recovery

Recovery

- Based on ARIES recovery algorithm
  - Redo-less and undo-less recovery for most *PPL* pages



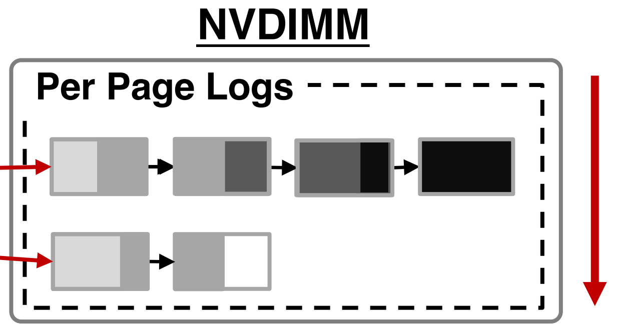
Recovery: Analysis

de-pplized == true

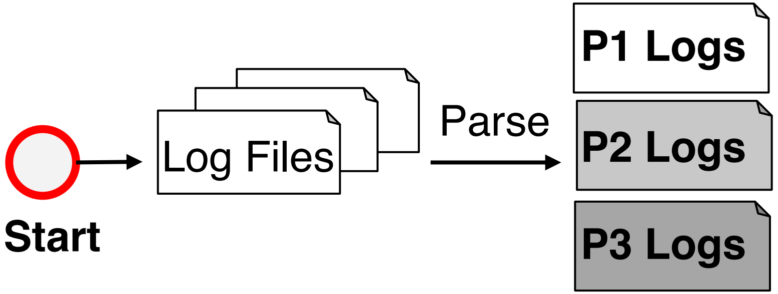
de-pplized == false

Page ID	Ptr
P1	...
P2	...

Lookup Table



By scanning the NVDIMM space,
reconstruct lookup table

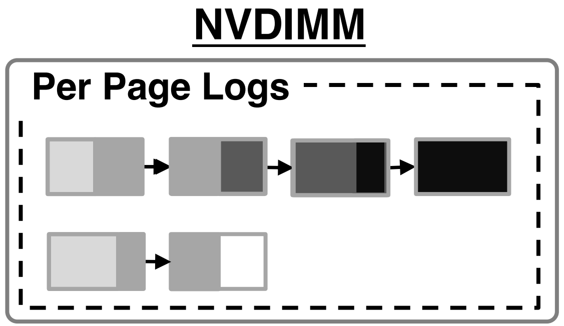


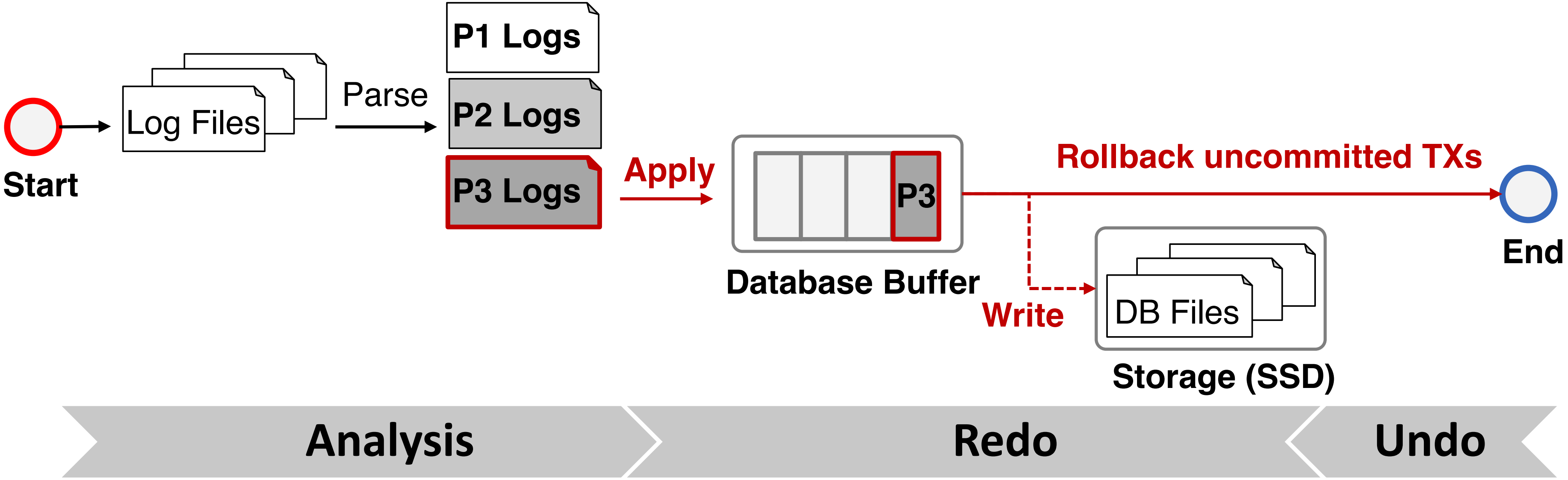
Recovery: Normal Page

de-pplized == true
de-pplized == false

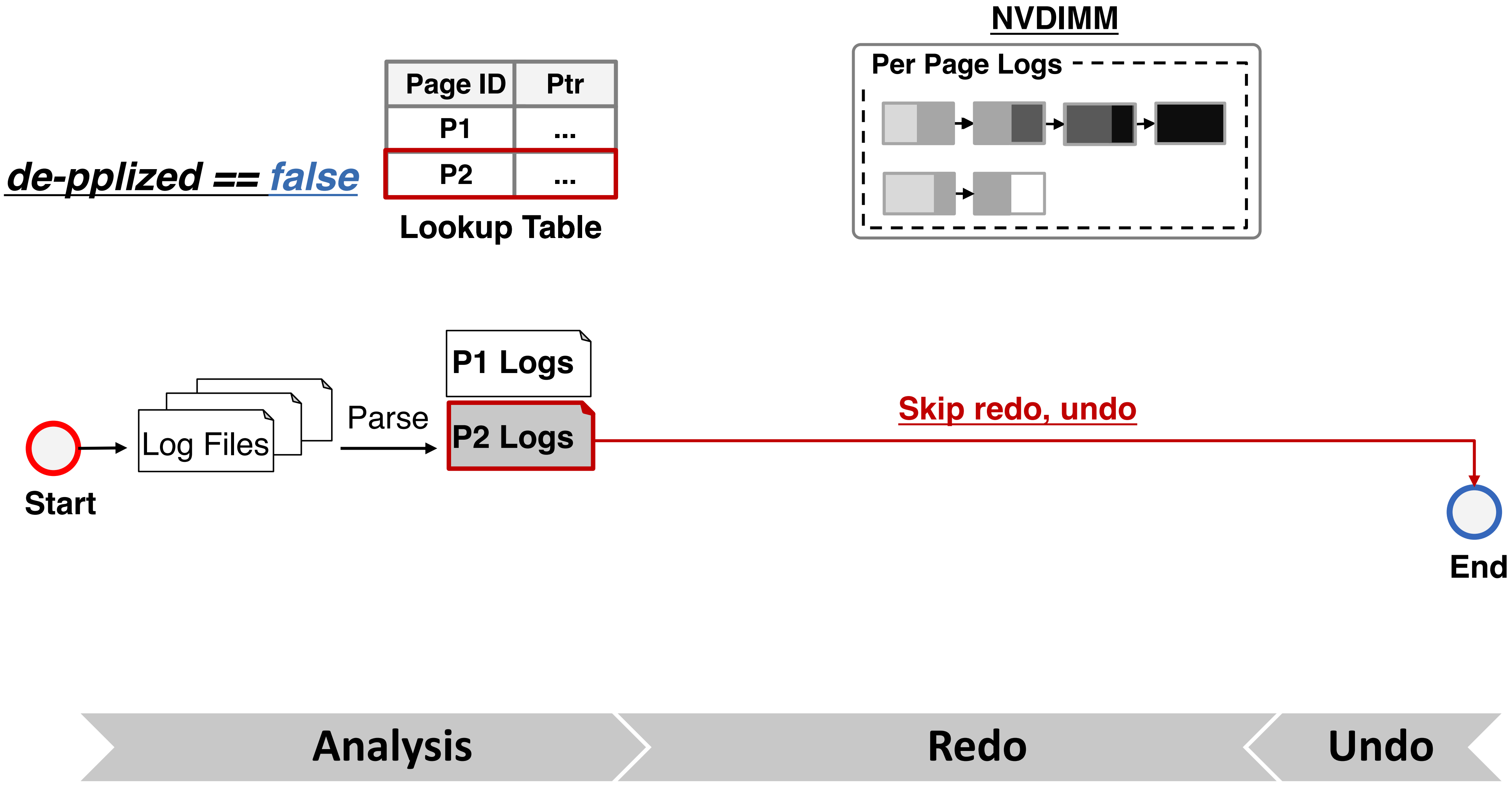
Page ID	Ptr
P1	...
P2	...

Lookup Table

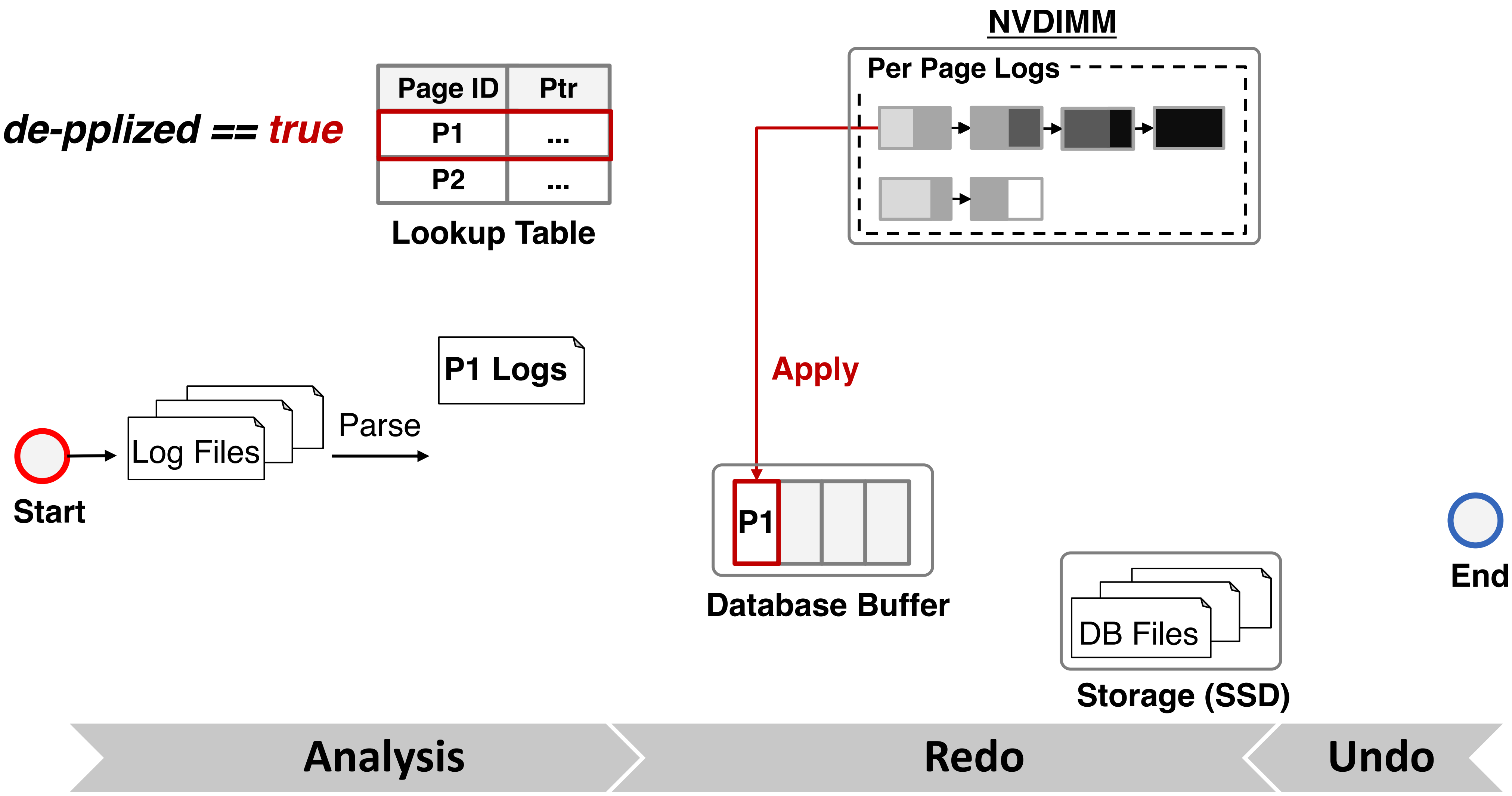




Recovery: PPL Page

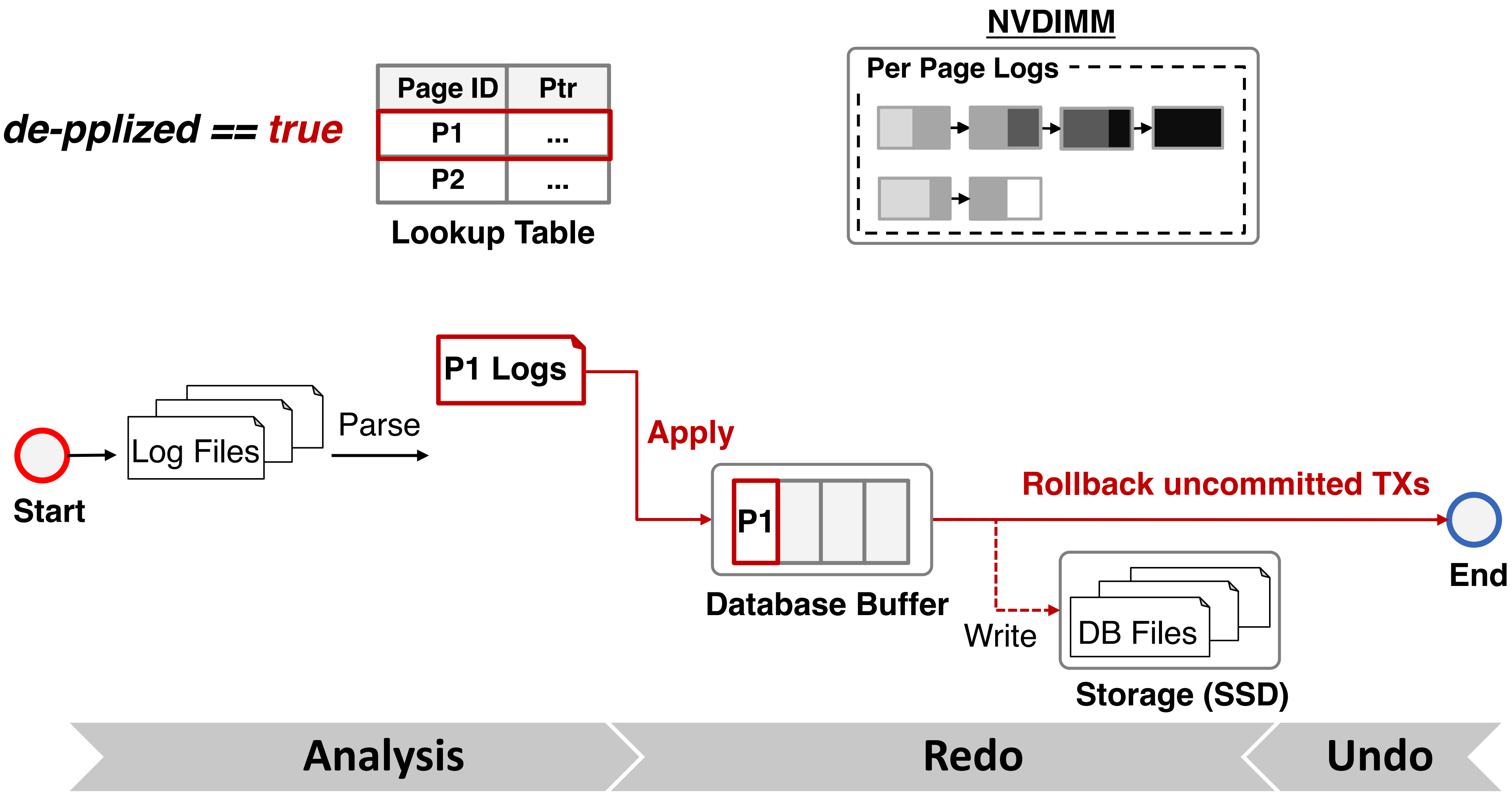


Recovery: De-PPLized Page



Recovery: De-PPLized Page

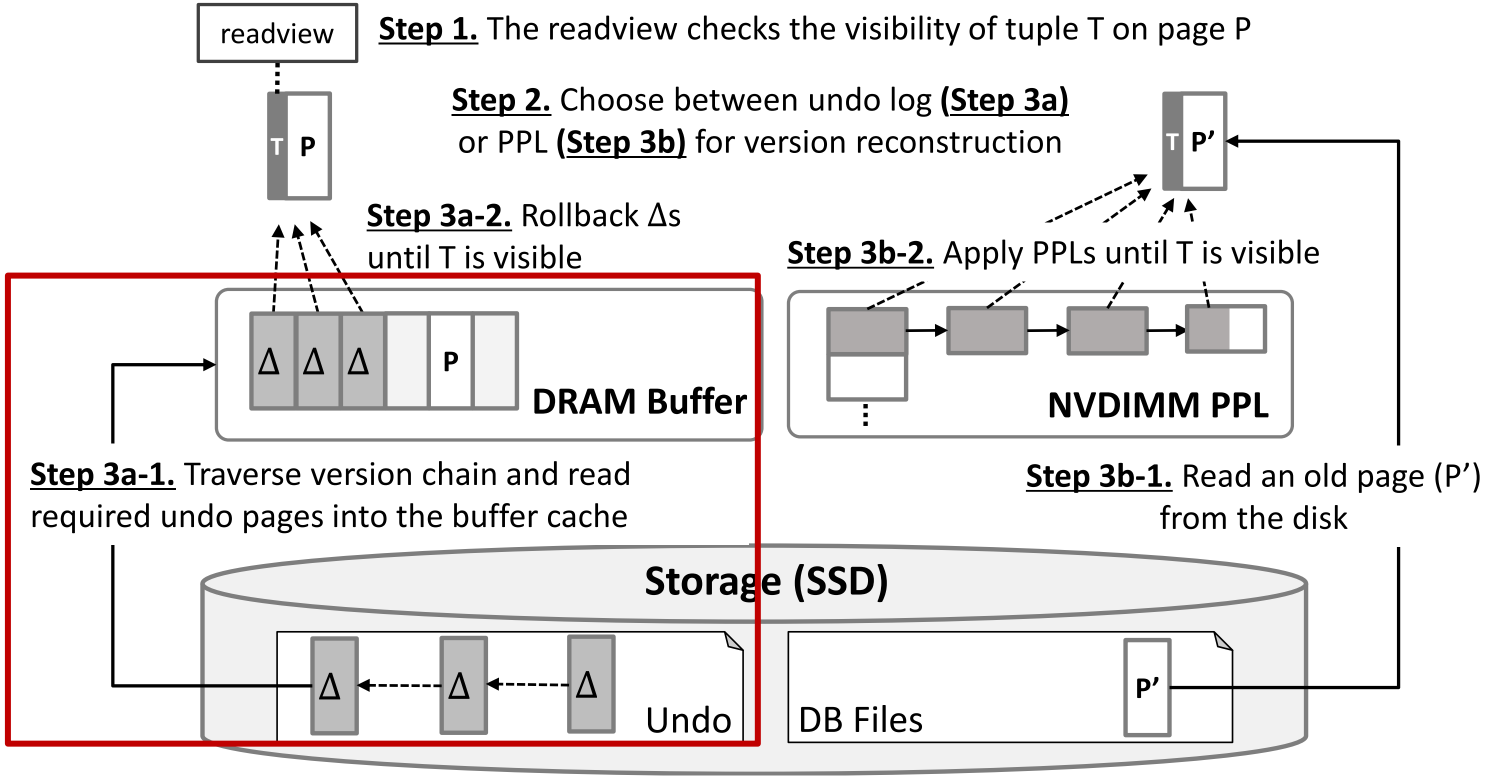
de-pplized == true



Redo-based Multi-Version Support

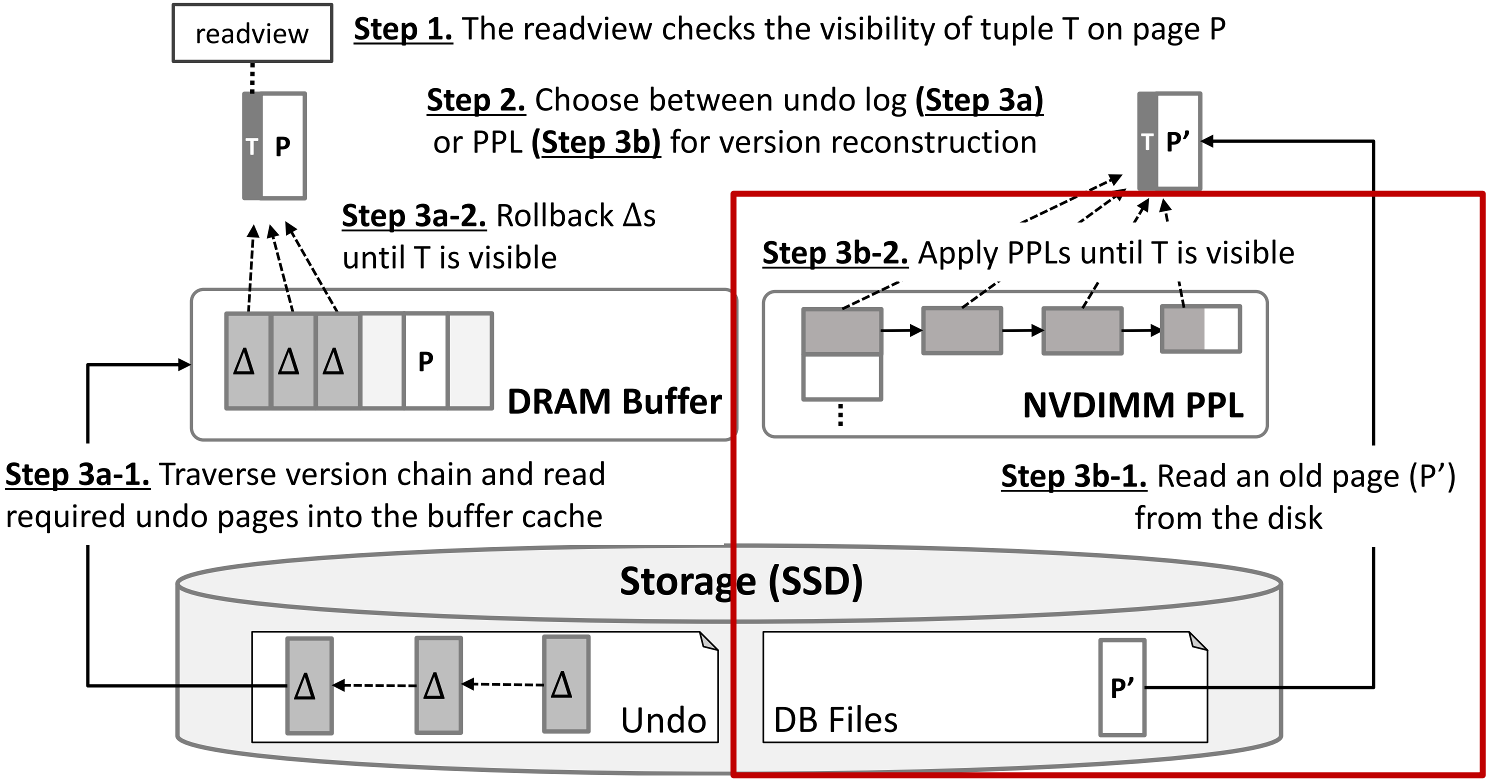
Undo-based vs. PPL-based Multi-Versioning

- Undo logs are stored in **transaction order**
  - Problems with HTAP workloads
    - Multiple undo pages read **prolongs Long-lived transaction (LLT) Latency**
    - Page misses for OLTP transaction **worsens OLTP throguhput**



Undo-based vs. PPL-based Multi-Versioning

- Supporting consistent view with PPL
  - PPLs are stored in **chronological order** for each page
  - Read: Multiple undo pages → **one old page + PPLs**



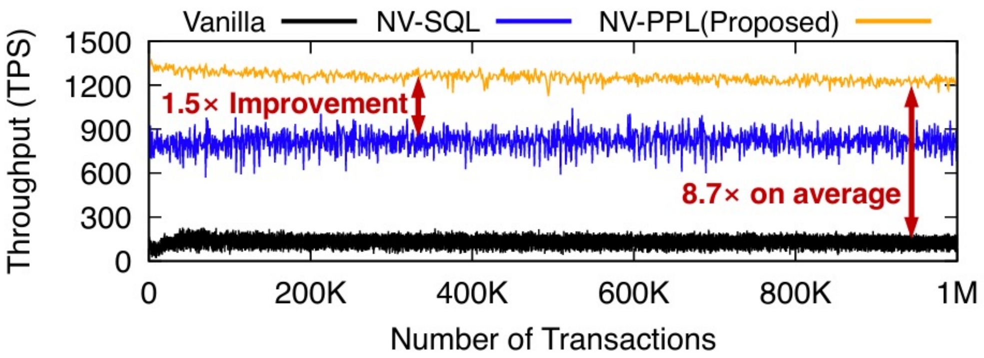
Performance Evaluation

Experimental Setup

- System setup
  - Intel Xeon ® E5-2640 (32 cores)
  - 64GB main memory
  - 16GB NVDIMM-N

- Database setup
  - Prototype: MySQL InnoDB v5.7
  - Database size: **54GB (500 warehouses)**
  - Buffer cache size: **5GB (10% of the DB size)**
    - All systems to have equal total memory cost.
    - **$ of 1GB NVDIMM = $ of 3GB DRAM**
    - Additional DRAM for Vanilla and NV-SQL (lookup table, DRAM log space, ...)
  - Concurrent clients: 32

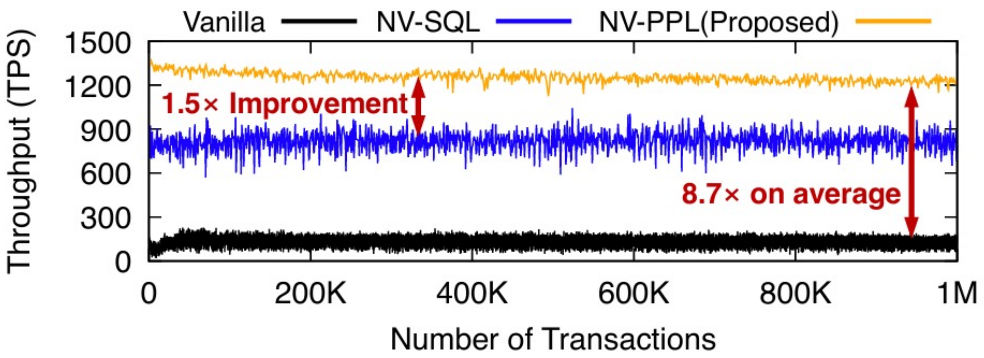
Evaluation #1. Basic Performance Analysis with TPC-C



DRAM:NVDIMM (GB)	Vanilla (5:0)	NV-SQL (2:1)	NV-PPL (2:1)
Average TPS	140	811	1,228
Avg Latency (ms)	7.8	1.2	0.8
99th Latency (ms)	399.5	135.4	24.7
Write/tx (KB)	158.4	78.5	29.7
Read/tx (KB)	98.2	139.1	187.2
Write/Sec. (MB)	21.7	62.2	35.6
Read/Sec. (MB)	13.5	110.1	224.6
Hit Ratio (%)	97.0	95.2	94.6
User CPU (%)	5.9	33.6	54.4

- Write/TX (KB)
  - **81%** lower (vs. Vanilla), **62%** lower (vs. NV-SQL)
- Transaction throughput
  - **8.7×** over Vanilla, **1.5×** over NV-SQL

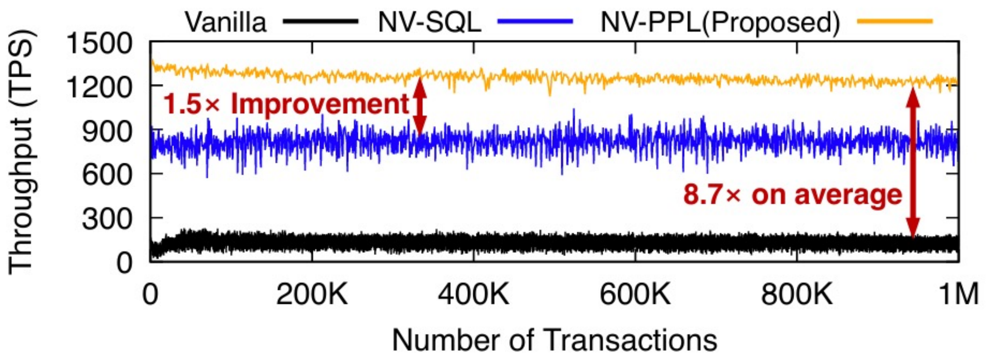
Evaluation #1. Basic Performance Analysis with TPC-C



DRAM:NVDIMM (GB)	Vanilla (5:0)	NV-SQL (2:1)	NV-PPL (2:1)
Average TPS	140	811	1,228
Avg Latency (ms)	7.8	1.2	0.8
99th Latency (ms)	399.5	135.4	24.7
Write/tx (KB)	158.4	78.5	29.7
Read/tx (KB)	98.2	139.1	187.2
Write/Sec. (MB)	21.7	62.2	35.6
Read/Sec. (MB)	13.5	110.1	224.6
Hit Ratio (%)	97.0	95.2	94.6
User CPU (%)	5.9	33.6	54.4

- Read/sec : Write/sec
  - Vanilla = 1: 1.6
  - NV-SQL = 1: 0.6
  - NV-PPL = **1: 0.2**
- I/O transformation: **Write-heavy** → **Read-heavy**
  - Leveraging fast read performance of SSD
  - Extending SSD lifespan

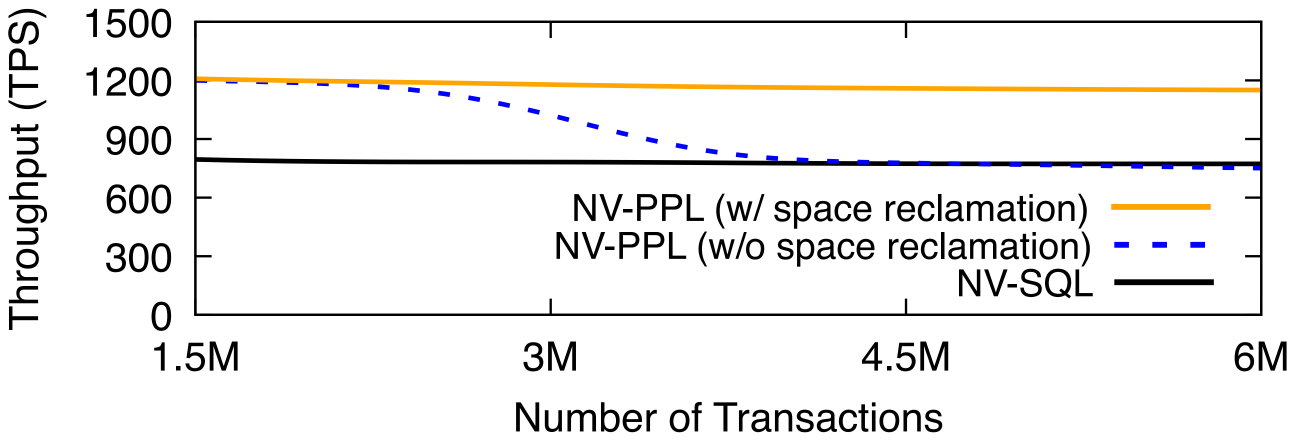
Evaluation #1. Basic Performance Analysis with TPC-C



DRAM:NVDIMM (GB)	Vanilla (5:0)	NV-SQL (2:1)	NV-PPL (2:1)
Average TPS	140	811	1,228
Avg Latency (ms)	7.8	1.2	0.8
99th Latency (ms)	399.5	135.4	24.7
Write/tx (KB)	158.4	78.5	29.7
Read/tx (KB)	98.2	139.1	187.2
Write/Sec. (MB)	21.7	62.2	35.6
Read/Sec. (MB)	13.5	110.1	224.6
Hit Ratio (%)	97.0	95.2	94.6
User CPU (%)	5.9	33.6	54.4

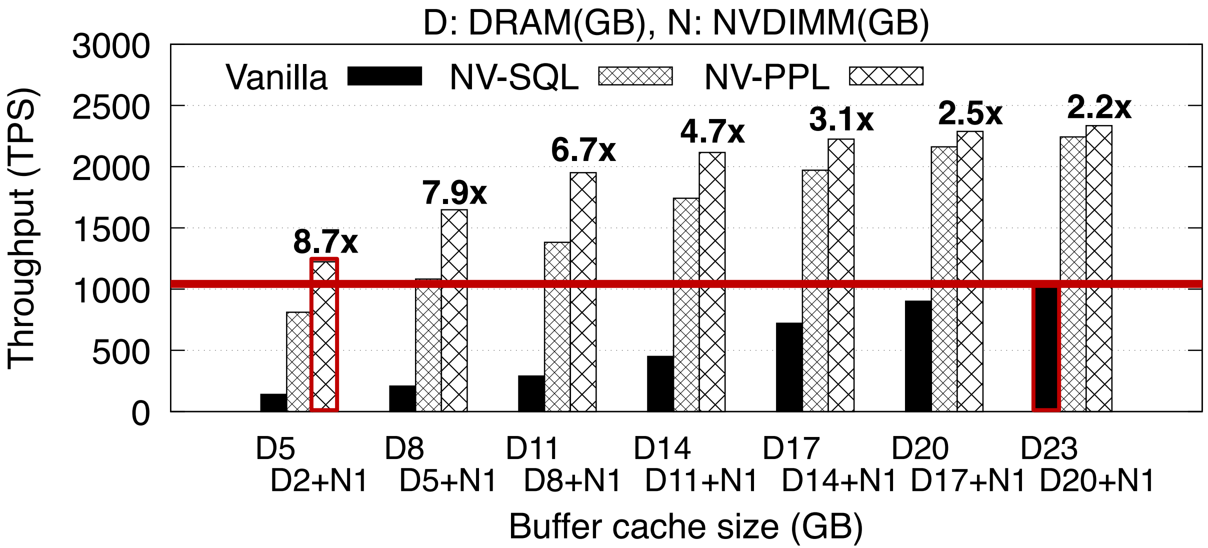
- Write reduction gains far outweigh these overheads (vs. Vanilla)
  - Additional CPU usage: 2.7%p ($CPU_{NVPPL} - CPUVanilla * TPS\ improvement$)
  - Lower hit ratio: -2.4%p

Evaluation #2. Effect of NVDIMM space reclamation



- **Without** space reclamation
  - Performance drops due to **running out of NVDIMM space**
- **With** space reclamation
  - Performance is consistently sustained

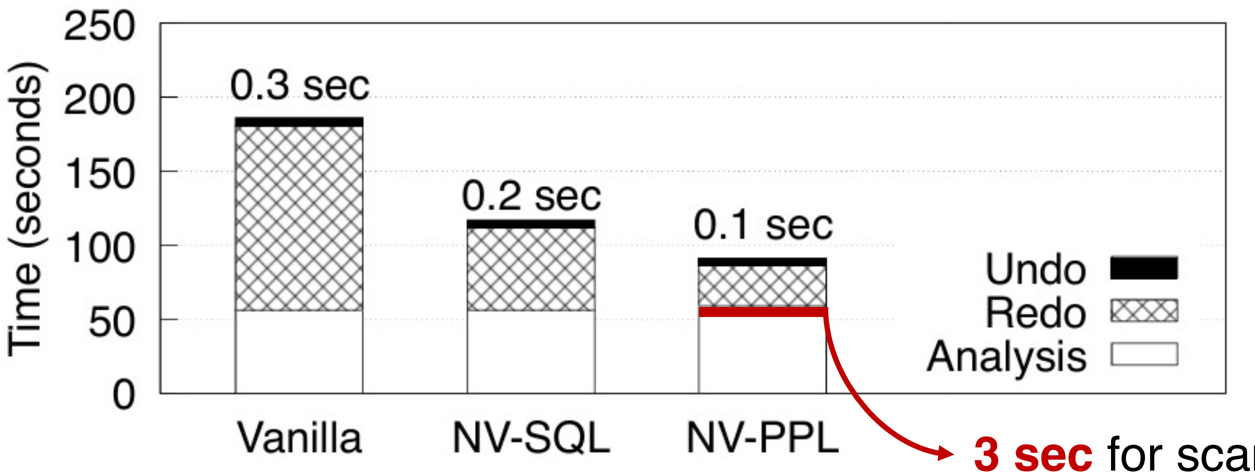
Evaluation #3. Varying Buffer Sizes



- NV-PPL consistently outperforms at all buffer sizes
  - Smaller decline in throughput with large buffer size
    - CPU-bound (i.e., more than 85% CPU utilization)
    - Higher hit ratio

- **Cost-performance improvement:**
  - NV-PPL (D2+N1) vs. Vanilla (D23): **1.17× throughput, 1/5 cost**

Evaluation #4. Recovery

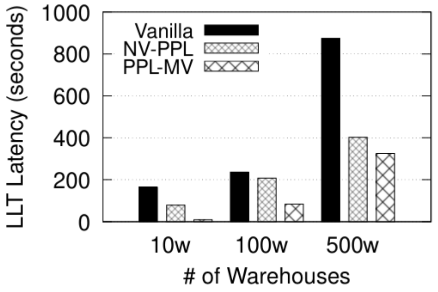


3 sec for scanning NVDIMM space

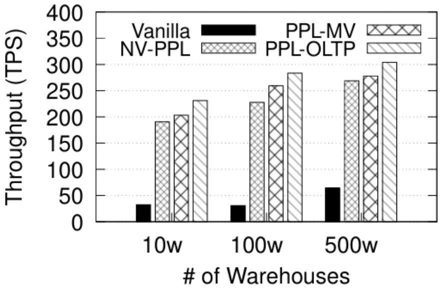
- NV-PPL spends **3 more seconds** during analysis phase
  – Due to scanning NVDIMM space
- NV-PPL can safely skip the redo for *PPL* pages
  – Redo time: **one-fifth** of Vanilla, **one-half** of NV-SQL
  – I/O from SSD: **one-half** of Vanilla

Evaluation #5. Effect of NV-PPL on HTAP Workloads

		Vanilla	NV-PPL	PPL-MV
Q1	Avg LLT Latency (s)	164	87	9
	Version Build (μs)	10,089	3,781	362
	Hit Ratio (%)	97.1	87.8	89.6
Q2	Avg LLT Latency (s)	44	23	6
	Version Build (μs)	9,851	1,872	197
	Hit Ratio (%)	97.2	89.1	90.9



(a) Average LLT Latency



(b) Average OLTP Throughput

- PPL-MV reduces average version construction time
  - Q1: 96.5% lower (vs. Vanilla), 89.6% lower (vs. NV-PPL)
- PPL-MV results in a milder drop in transaction throughput
  - Hit ratio: +1.8%p ↑ (vs. NV-PPL)

Summary

- NV-PPL: per-page logging on NVDIMM
  - Selectively write based on size of cumulative PPLs
  - Foreground reclamation & background PPL cleaner
  - Redo-less recovery based on ARIES & PPL-based multi-versioning
- Benefits
  - Basic NV-PPL
    - Reducing page writes to SSD
    - Improving throughput and latency
    - Extending SSD lifespan
  - Recovery
    - Redo-less and undo-less recovery
- **It's time to revisit NVDIMM**

Thank you

Boosting OLTP Performance with Per-Page Logging on NVDIMM

Bohyun Lee[+], **Seongjae Moon**[*], Jonghyeok Park[**], Sang-Won Lee[†]

Technical University of Munich[+] , Sungkyunkwan University[*], Korea University[**], Seoul National University[†]

E-mail: sungjae.moon@skku.edu
Github: https://github.com/JonghyeokPark/mysql-57-nvdimm-ppl

Check out more details in our paper!