OPTR: Order-Preserving Translation and Recovery Design for SSDs with a Standard Block Device Interface

Yun-Sheng Chang and Ren-Shuo Liu

System and Storage Design Lab
Department of Electrical Engineering
National Tsing Hua University, Taiwan

國立清華大學
NATIONAL TSING HUA UNIVERSITY

USENIX
ATC '19

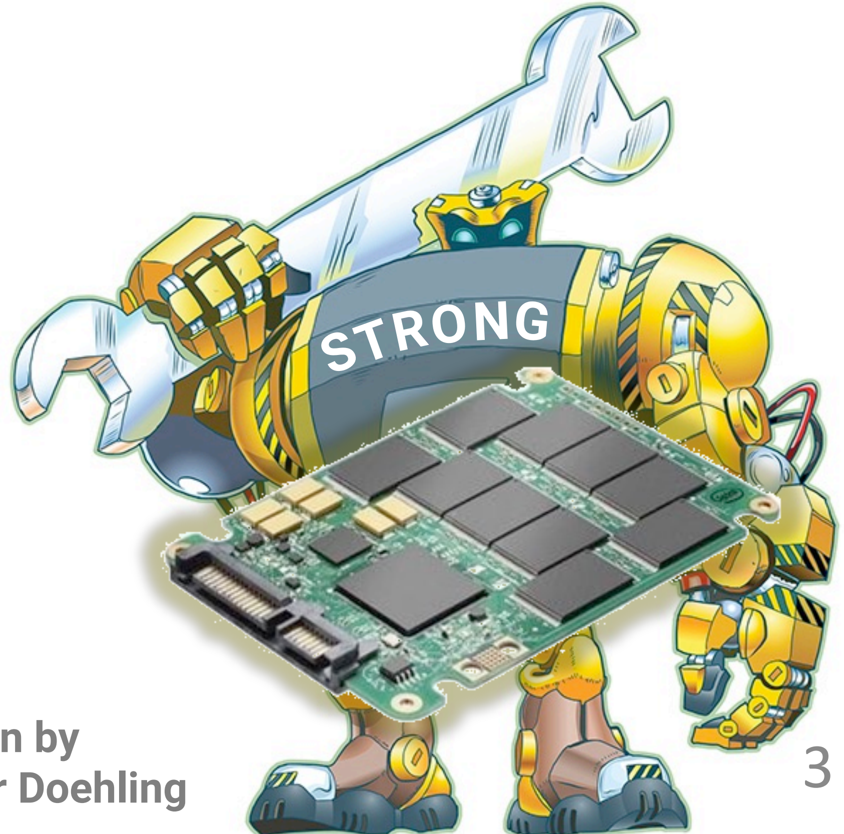
Solid-State Drives (SSDs)

- Inherit the **interface** and a **weak guarantee** from HDDs
  - **Permit** persisting write requests in an **arbitrary order**
- Implication to FS and DBS
  - Need to frequently **flush** SSDs to ensure order
  - At the cost of performance degradation

1989 1999 2009 2019

Order-Preserving SSDs (OP-SSDs)

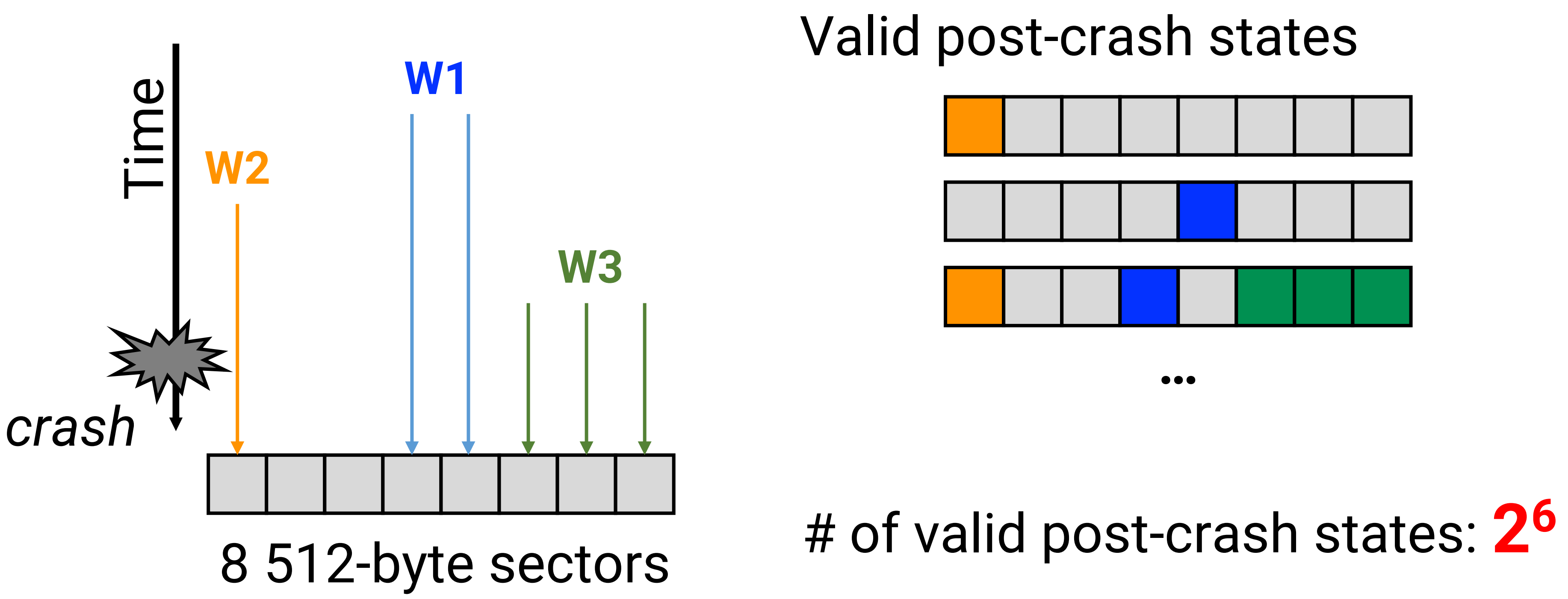
- **Strong** request-level guarantees
  - Persist all write requests **in order**
  - Persist each write request **atomically** (a bonus)
- Invariants
  - **Identical** interface to existing software, i.e., read, write, and flush
  - **Comparable** performance to traditional SSDs



Robot drawn by Christopher Doehling

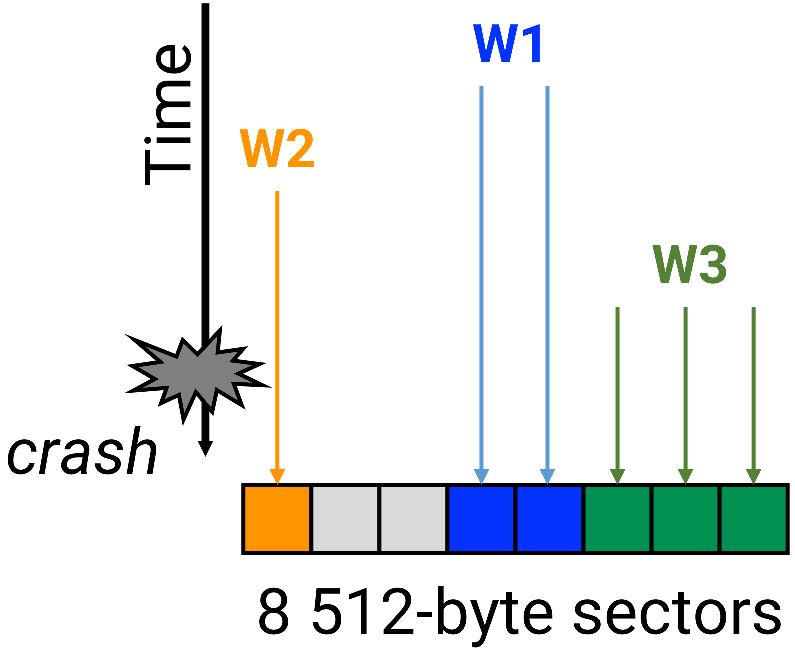
Traditional SSD: Weak Crash Guarantees

- Write requests can be persisted **out-of-order**
- Each write request can be **partially complete**

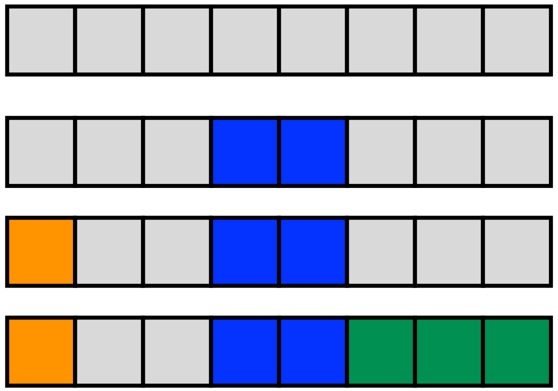


OP-SSD: Strong Crash Guarantees

- Write requests are persisted **in-order**
- Each write request is **atomic**, regardless of its size



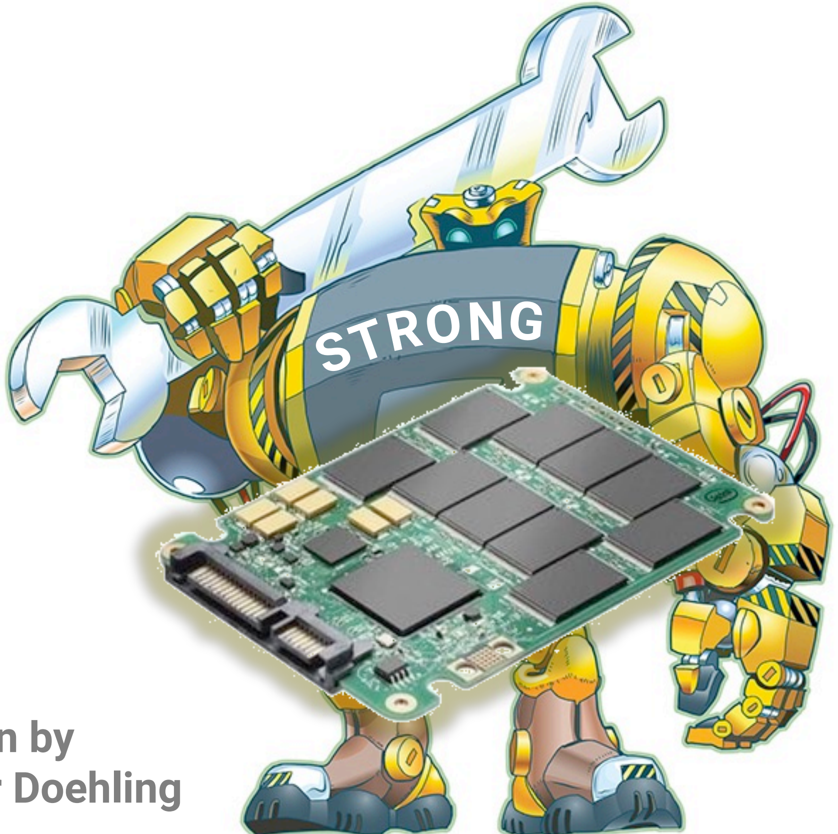
Valid post-crash states



# of valid post-crash states: **4**

OP-SSDs in Computer Systems

- Optimize **existing FS and DBS**
  - Remove unnecessary flushes
  - Practical and manageable because OP-SSDs keep the interface intact
- Inspire **new FS and DBS**
  - Exploit the strong crash guarantees
- New SSD **industrial standard**
- New SSD **research area**
  - Flash-translation layers (FTLs)



Robot drawn by
Christopher Doehling

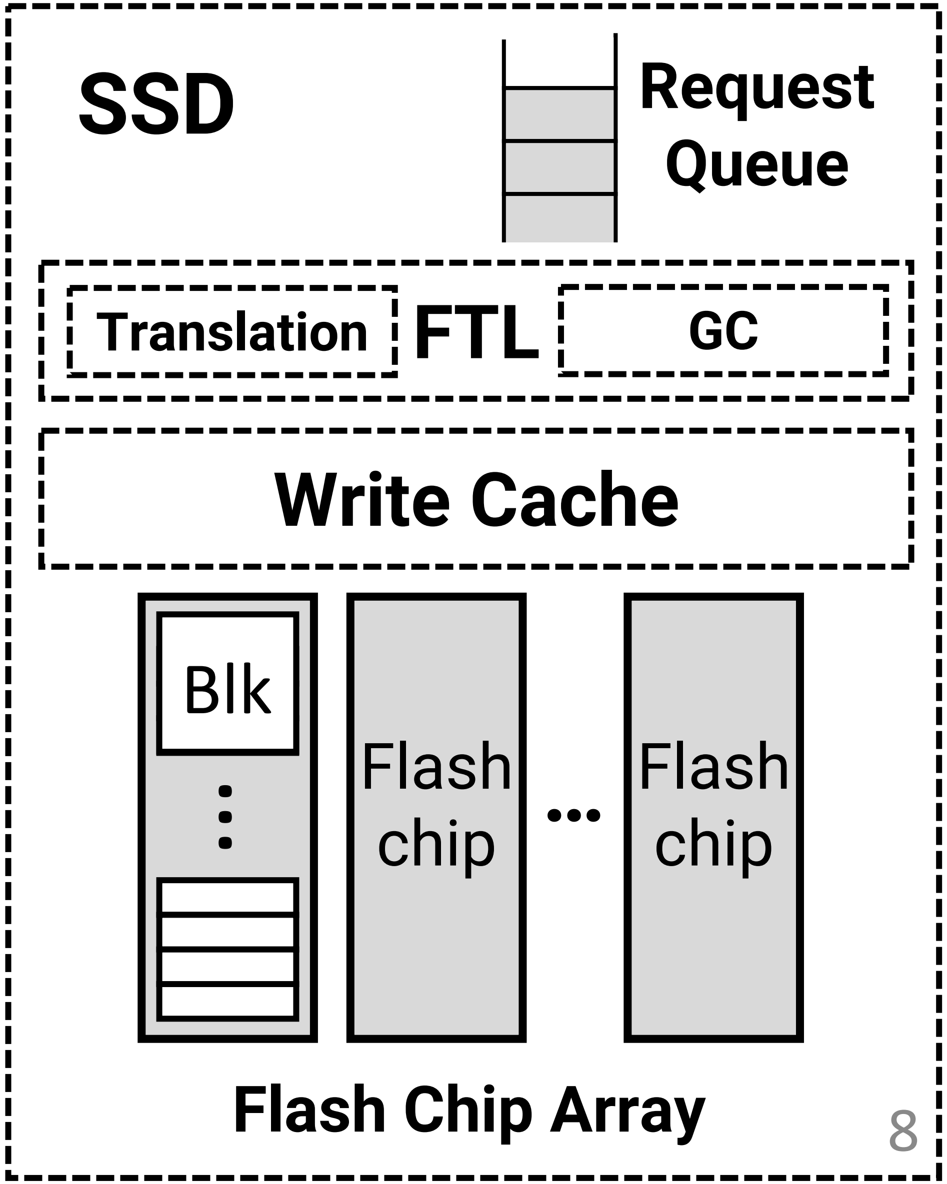
Outline

- Order-preserving SSDs
- Background
- Order-preserving design
- System optimizations and evaluation
- Conclusion

Background: A Simple SSD Model

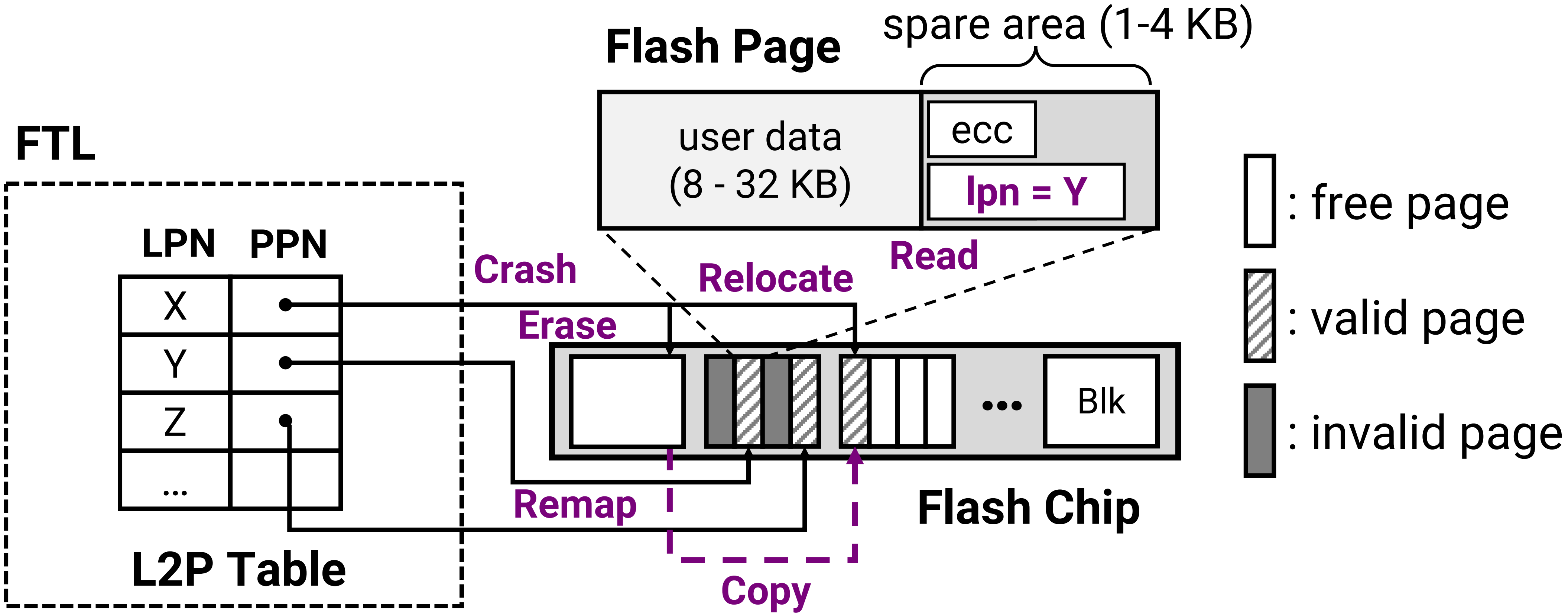
- FTL (flash translation layer) performs logical-to-physical address mapping
  - Constraint of flash: No in-place update
- High performance schemes
  - Flash parallelism
  - Request reordering
  - Write cache

  Breaking the order!
- Garbage collection
- Crash recovery



Background: GC and SSD Recovery

- **GC** is required to reclaim space for future writes
- **Crash recovery**: Since L2P table is kept in RAM, FTL has to reconstruct the L2P table after a crash

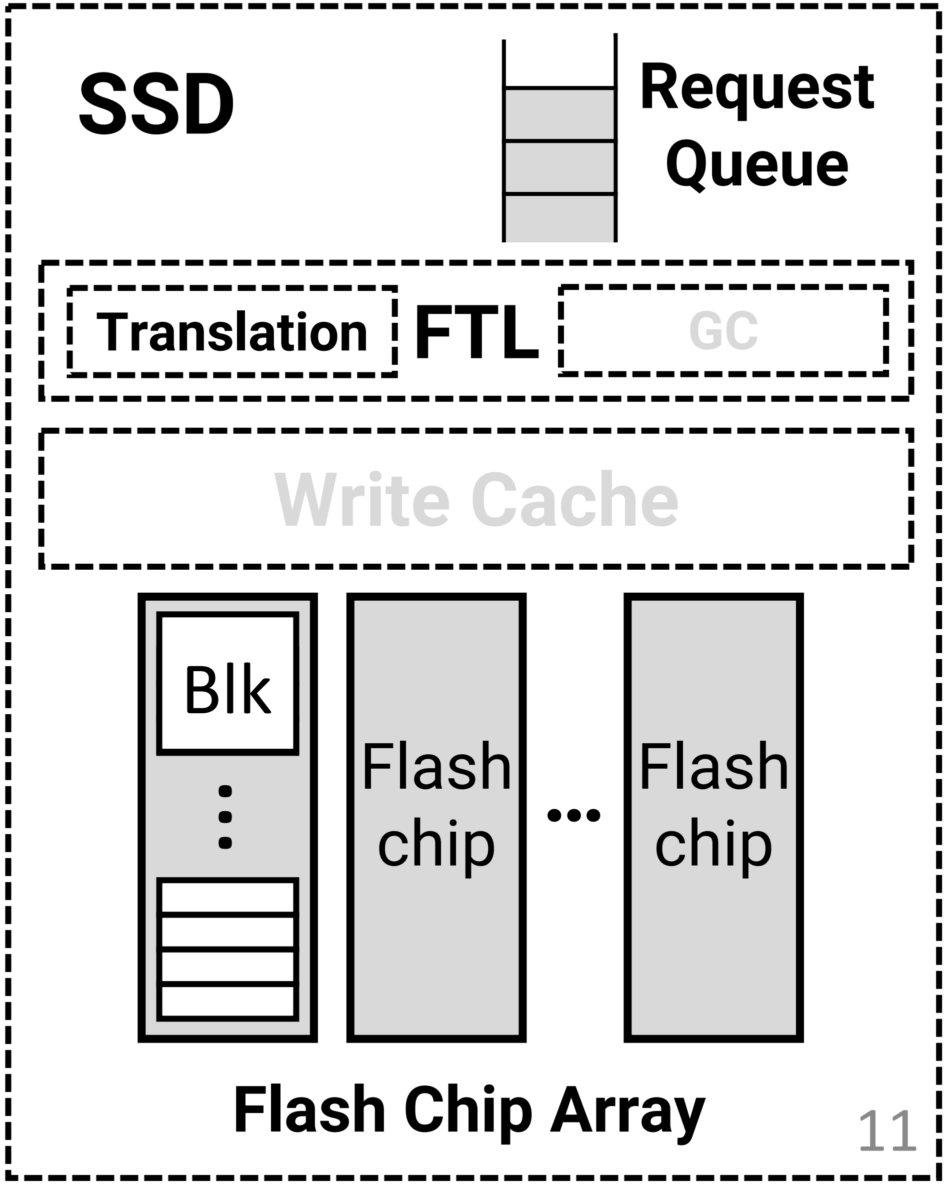


Goal of Order-Preserving Design

- High performance schemes are still kept
  - Flash parallelism
  - Request reordering
  - Write cache (coalescing)
- Write requests are not necessarily processed in order
- **Recovery procedure** of FTL is extended
  - Rollback SSD to a desired state
  - Create an order-preserving **illusion**

An Incomplete SSD Model

- Let's first assume an SSD without a write cache and GC
- We'll remove these (impractical) assumptions in a minute

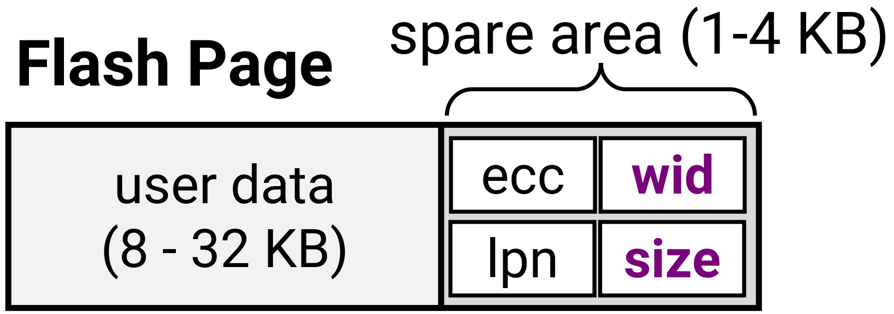


Order-Preserving Recovery

- **Idea**: During recovery, if we know exactly which writes are complete, we can recover until the first incomplete write
  - E.g., if the 1^{st}, 2^{nd}, 3^{rd}, 5^{th} writes are complete, then we can simply recover the first three writes, but not any other write
- **Write completion tracking**: If a write contains N pages, and during recovery, we find N pages for the write, then the write is indeed complete; otherwise, the write is incomplete

Order-Preserving Recovery

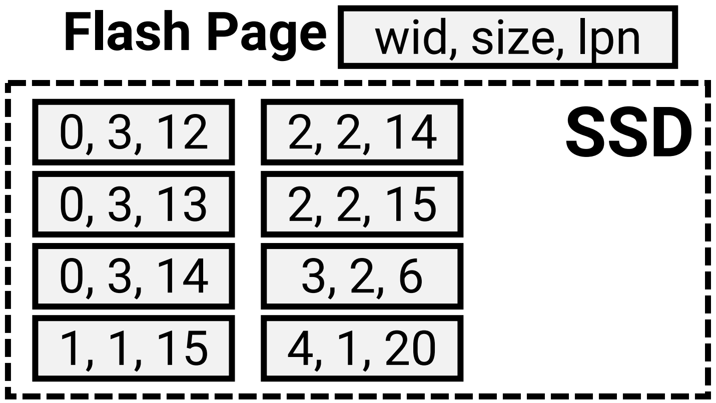
- **wid** (8 B): a sequence number assigned to a write according to the order in which writes are received by the SSD
- **size** (4 B): the number of pages the write contains

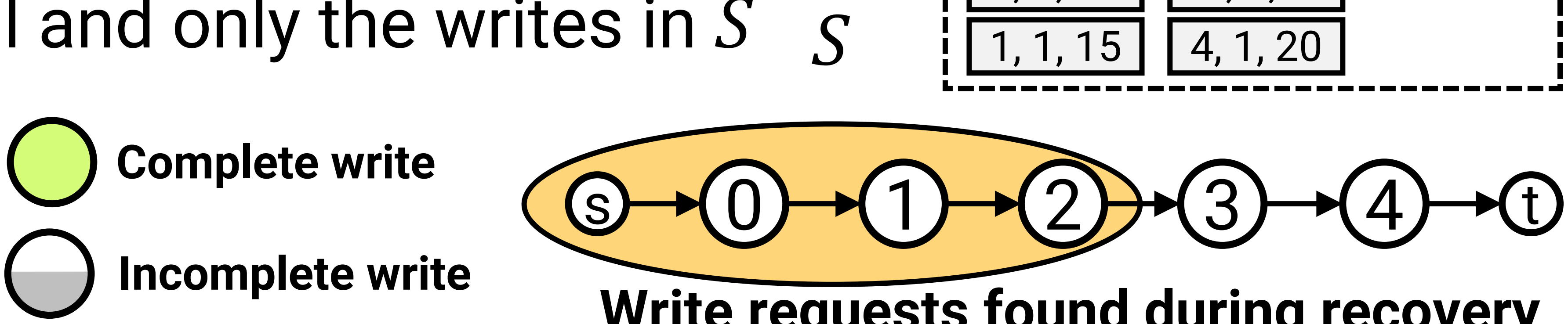


Status	Condition
Complete	# pages found = **size**
Incomplete	# pages found < **size**

Recovery Procedure (without write cache and GC)

- Read out all the programmed pages
- Determine whether each write is complete or incomplete
- Construct a flow network with each node representing a write request and each edge pointing from W_i to W_{i+1}
- Find a s-t cut $C = (S, T)$ such that
  - Every write in S is complete
  - $|S|$ is maximized
- Recover all and only the writes in S





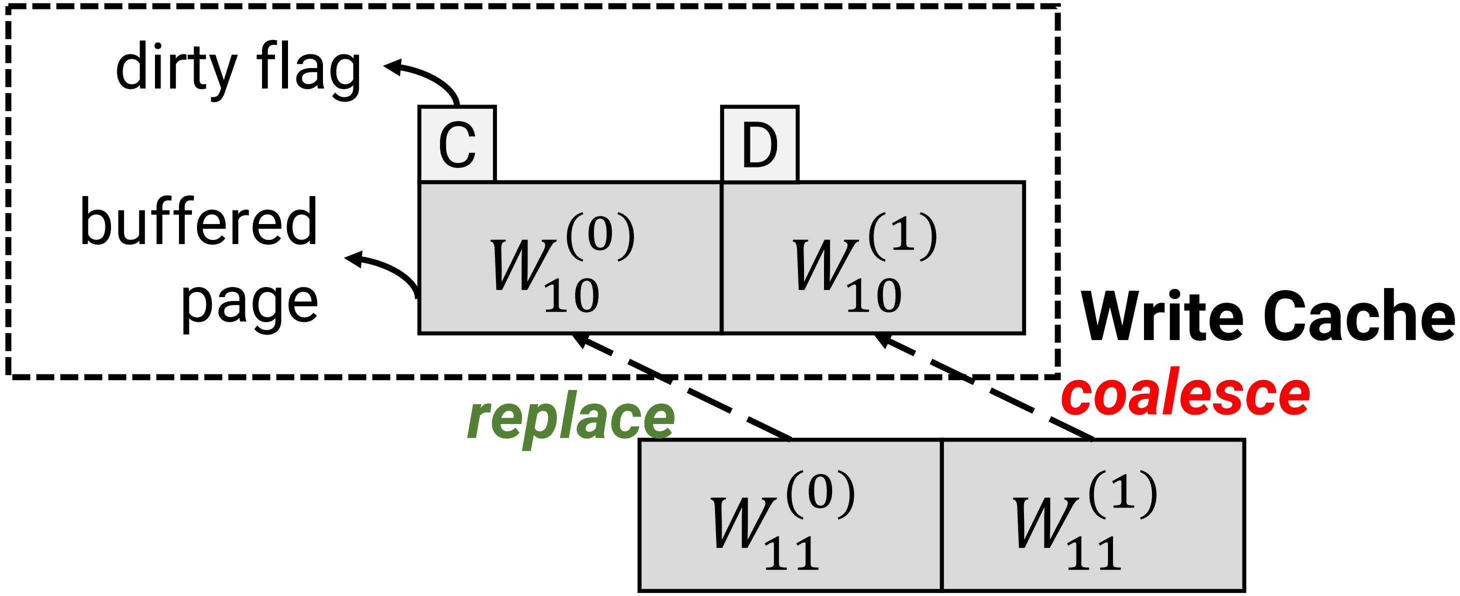
Write requests found during recovery

Support for Write Coalescing

- Write coalescing improves performance and lifetime
- **Challenge**: The number of pages found during recovery can no longer match the number of pages the write contains
- Naïve solution: Forbid write coalescing

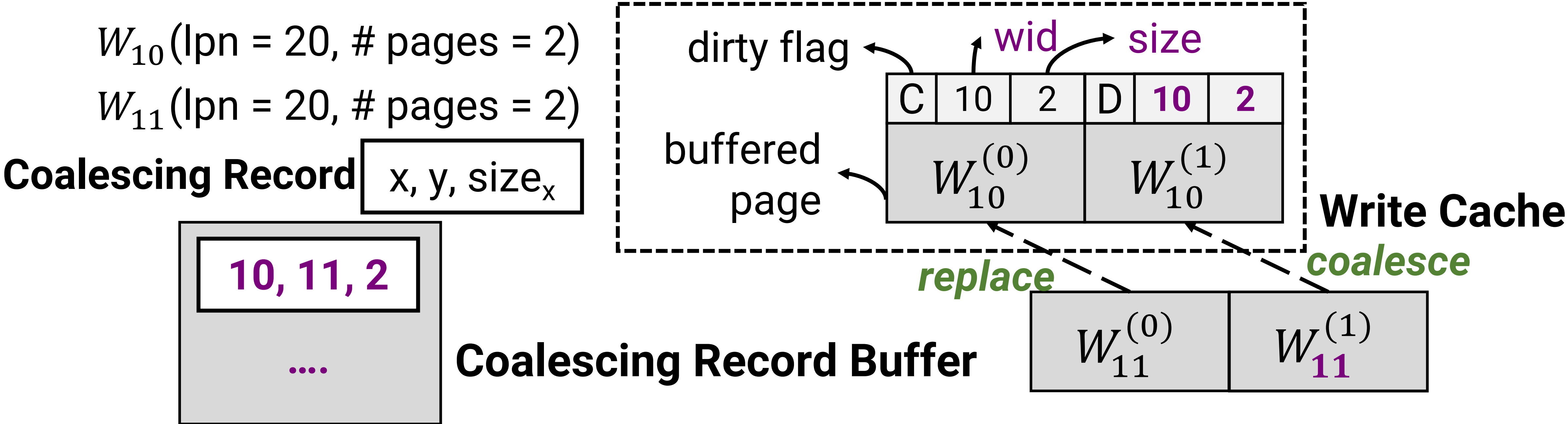
W_{10}(lpn = 20, # pages = 2)

W_{11}(lpn = 20, # pages = 2)



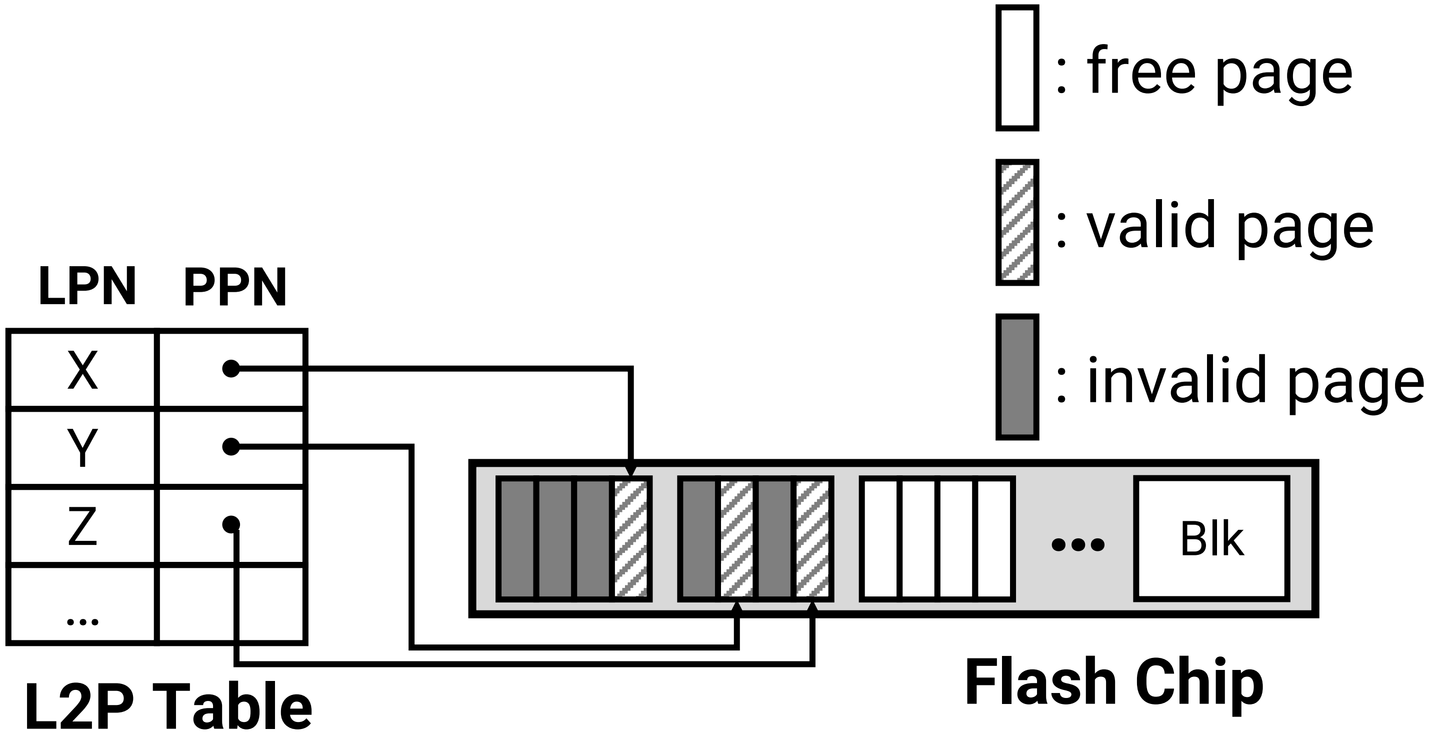
Write Coalescing Tracking

- **Coalescing records** keep track of coalescing events
  - Recovery procedure expect one less page for each record
  - Write requests that coalesce are atomic as a whole
- A batch of coalescing records are written to flash when the buffer is full or upon receiving a flush request



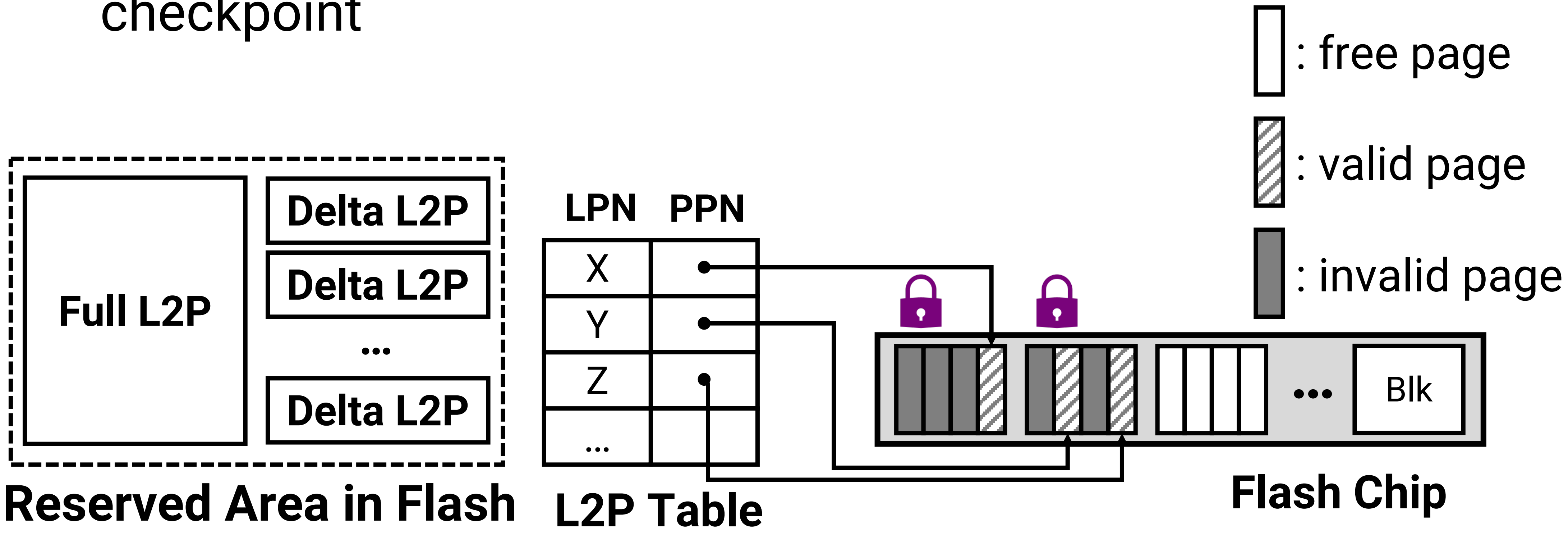
Support for Garbage Collection

- The job of a garbage collector is to reclaim invalid pages
- However, our recovery procedure relies on these invalid pages to determine whether each write is complete
- **Solution**: Mapping table checkpoint



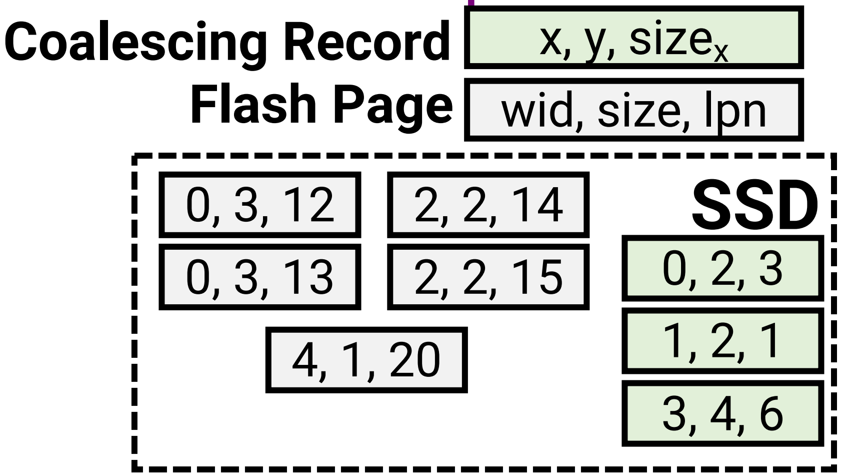
Mapping Table Checkpointing

- Perform incremental and full checkpoint
- Once a checkpoint is successfully created, all write requests prior to the checkpoint is guaranteed recoverable
- Restrict GC to only reclaim pages programmed **before** a checkpoint

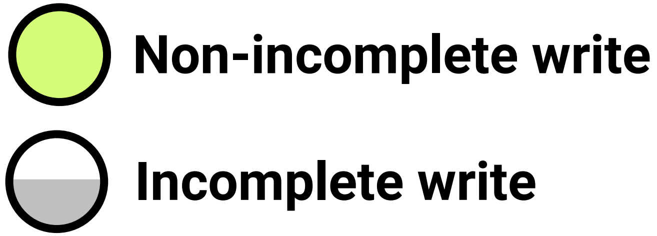


Recovery Procedure (with write cache and GC)

- Sequentially apply all checkpoints
- Read out all the pages programmed after the latest chkpt
- Read out all the coalescing records created after the latest chkpt
- Determine whether each write is incomplete or non-incomplete



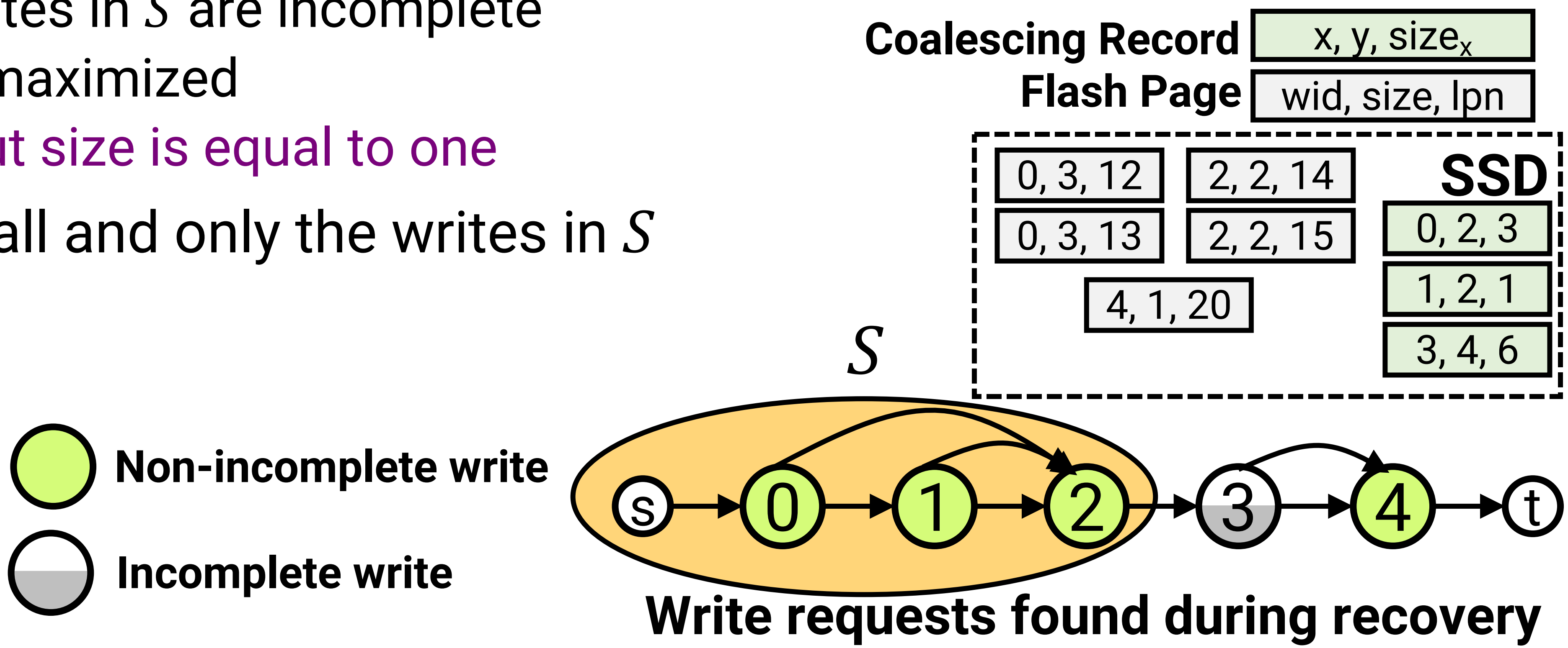
Status	Condition
Incomplete	# pages found + # coalescing records < **size**



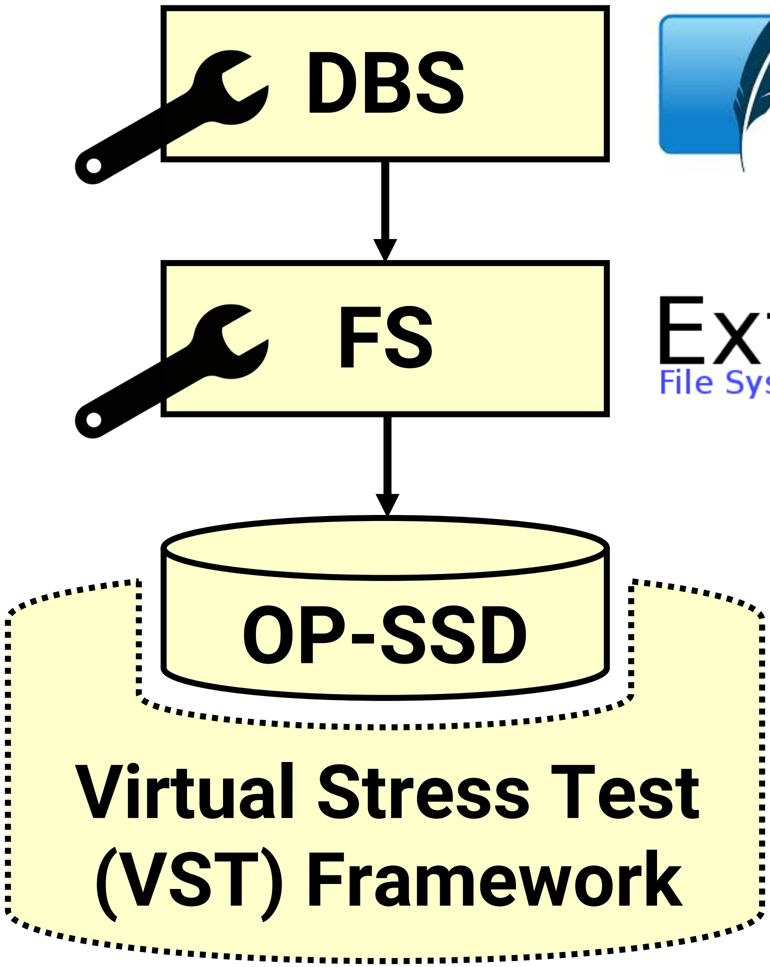
Write requests found during recovery

Recovery Procedure (with write cache and GC)

- Construct a flow network with each node representing a write request, each directed edge pointing from W_i to W_{i+1}, and each bent edge pointing from x to y for each coalescing record <x, y, size$_x$>
- Find a s-t cut $C = (S, T)$ such that
  - No writes in S are incomplete
  - $|S|$ is maximized
  - The cut size is equal to one
- Recover all and only the writes in S



System Optimizations and Evaluation



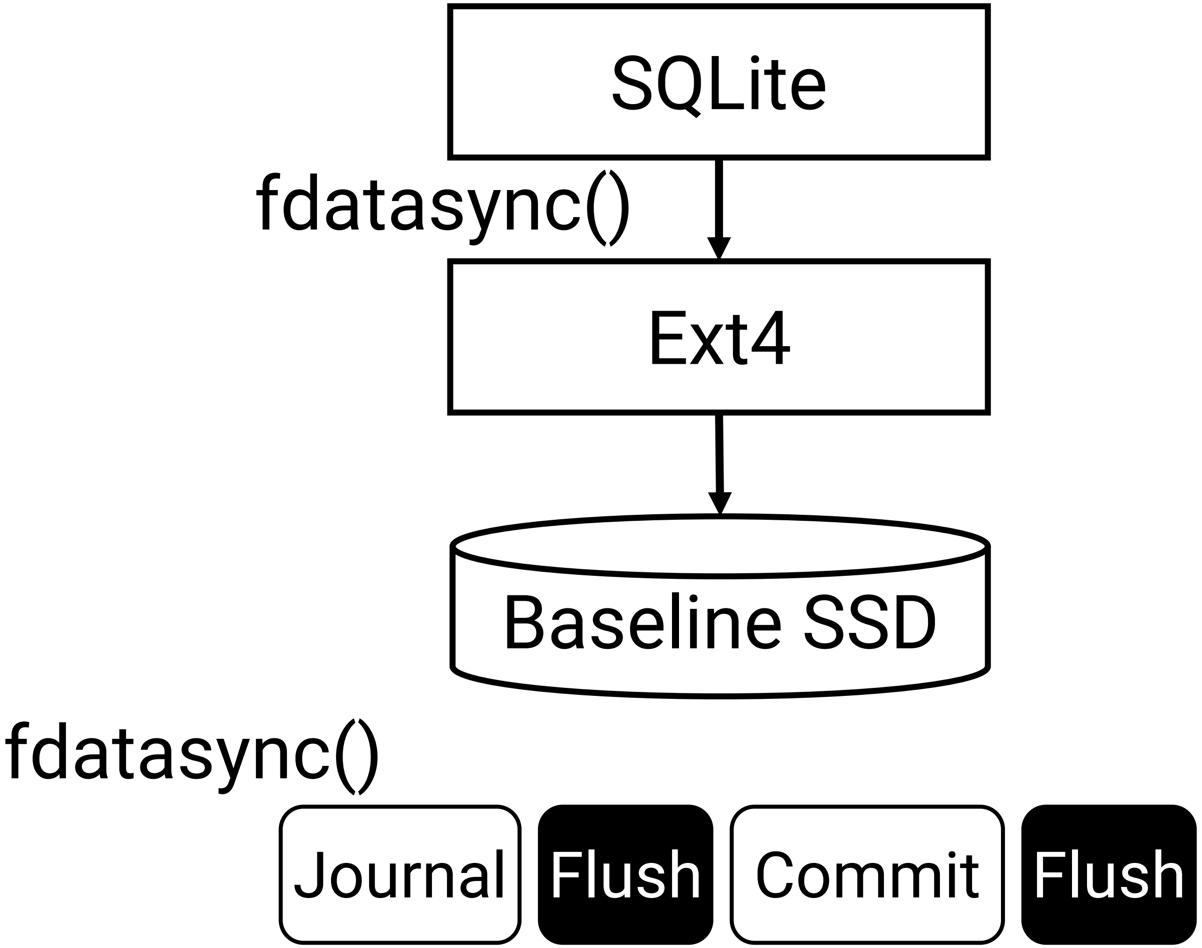
SQLite

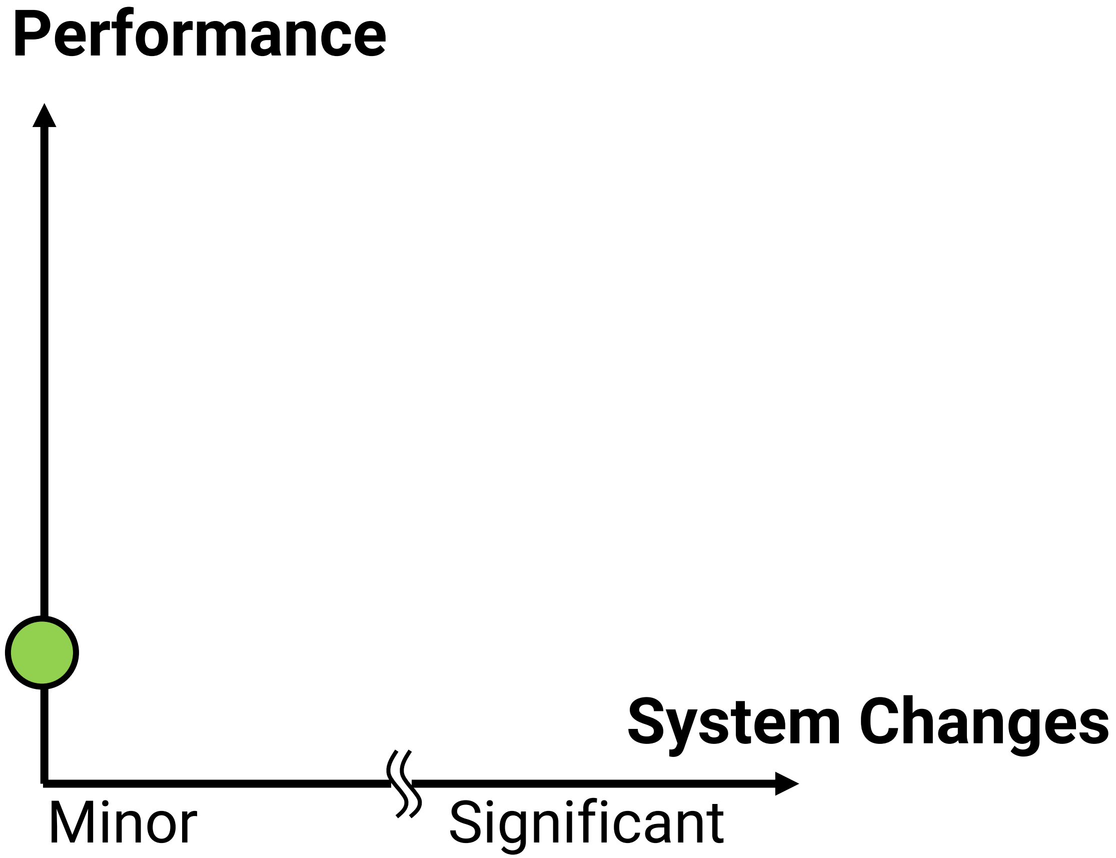
Ext4
File System

System Optimizations and Evaluation

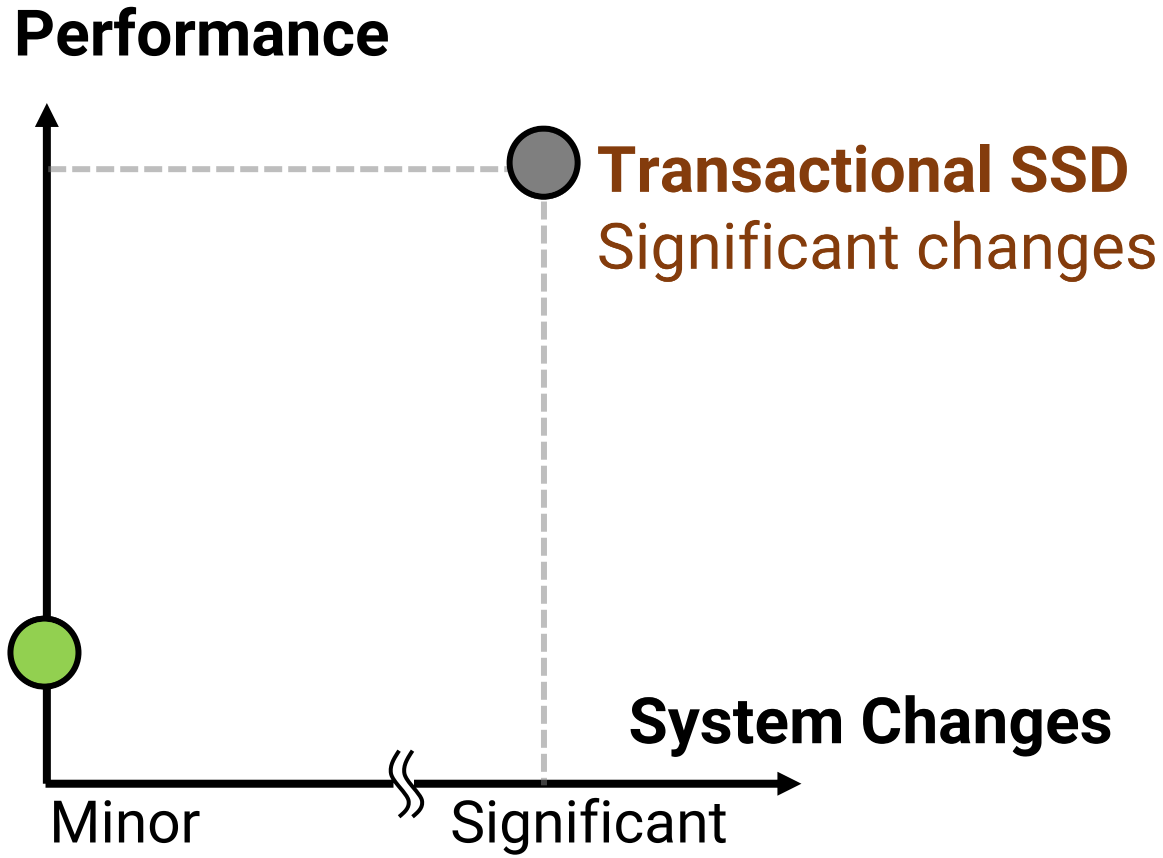
B	*fdatasync*	*writes* *flush* *writes* *flush*
1	*fdatasync*	*writes* *writes* *flush*
2	*fdatasync*	*writes* *writes* *flush*
	fdatafence	*writes* *writes*
3	*fdatasync*	*writes* *writes*

Baseline Systems

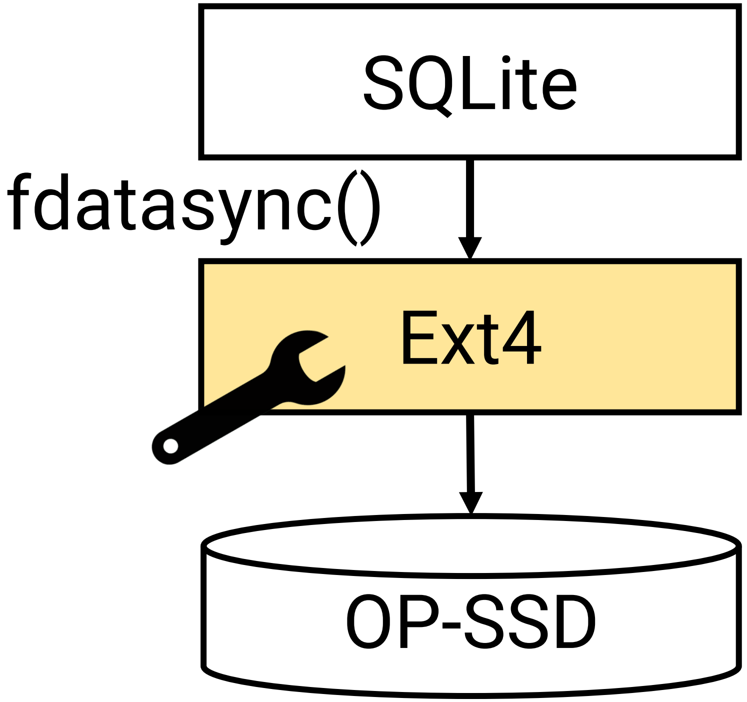




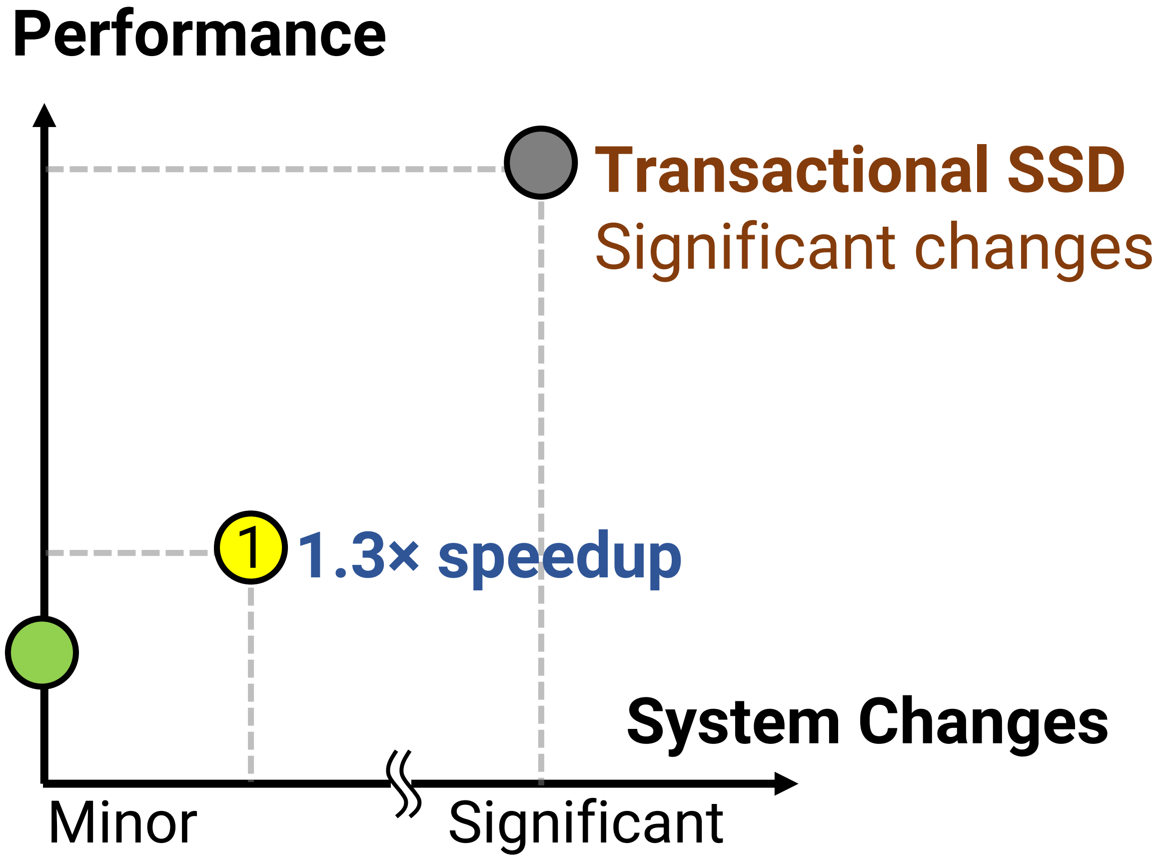
Systems Using Transactional SSDs



1st System Optimization with OP-SSDs

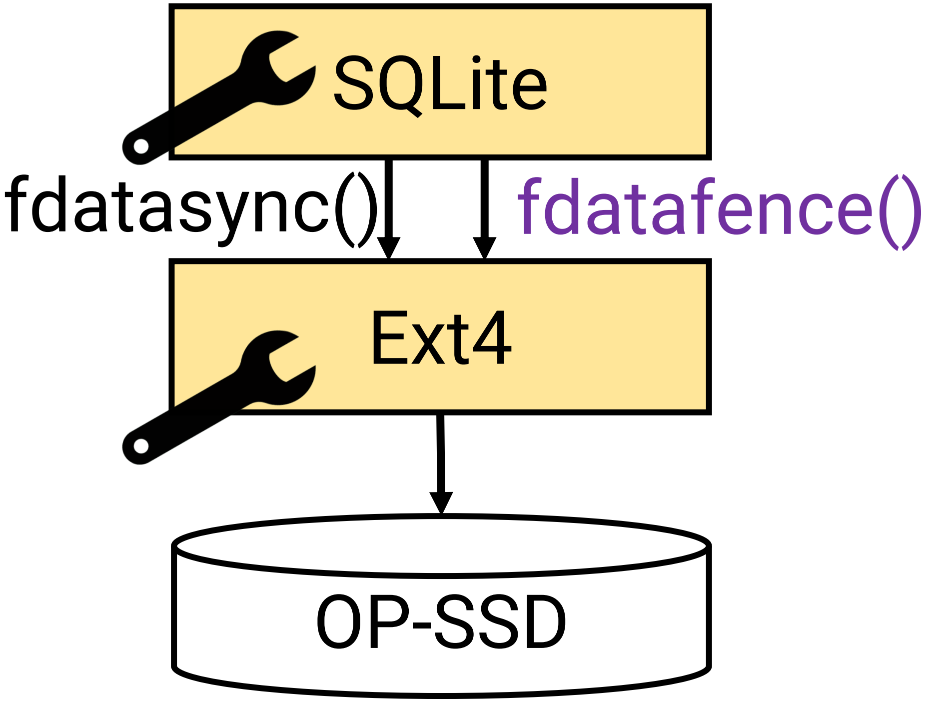


fdatasync()



Only one flush per fdatasync()

2nd System Optimization with OP-SSDs

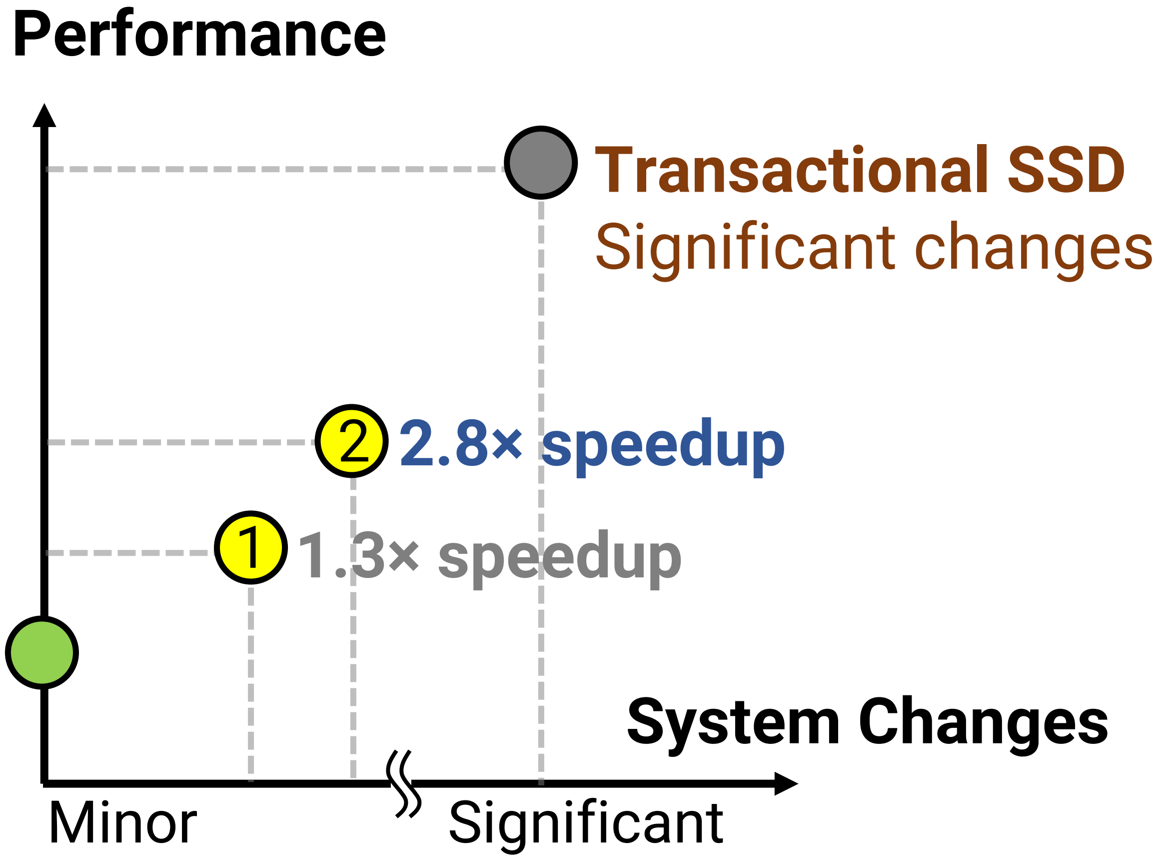


fdatasync()

Journal | Commit | Flush

fdatafence()

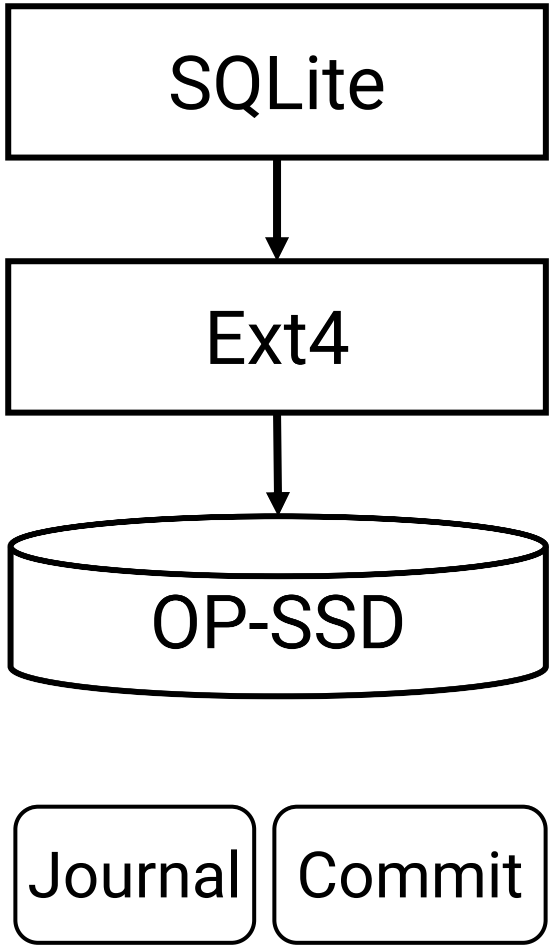
Journal | Commit

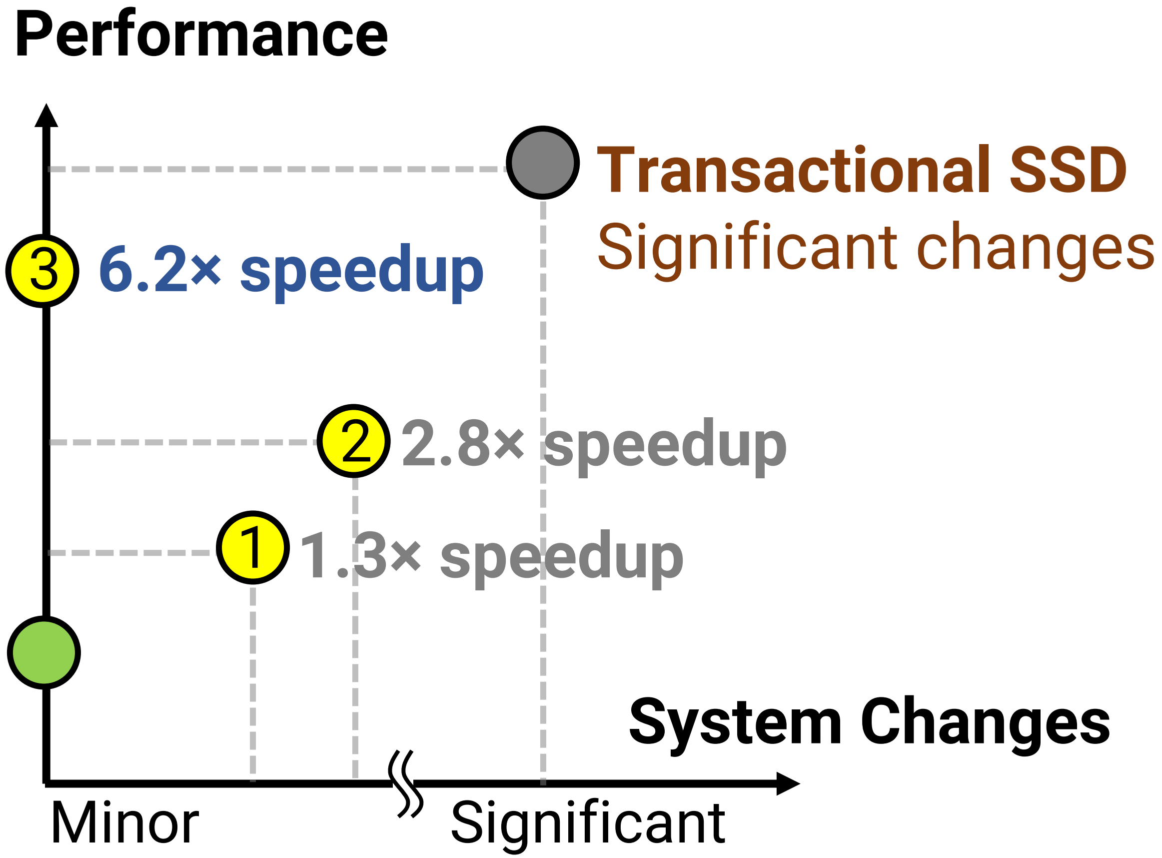


3rd System Optimization with OP-SSDs

Mount with *-o nobarrier*

fdatasync()





Relaxed durability + guaranteed consistency

Conclusion

- We propose **order-preserving SSDs**
- **Strong** request-level guarantees
  - Persist all write requests **in order**
  - Persist each write request **atomically**
- Impacts of OP-SSDs to computer systems
  - Optimize **existing FS and DBS** → Show three optimizations
  - Inspire **new FS and DBS**
  - New SSD **industrial standard**
  - New SSD **research area** → Realize a prototype

} Future work

Order-Preserving SSDs

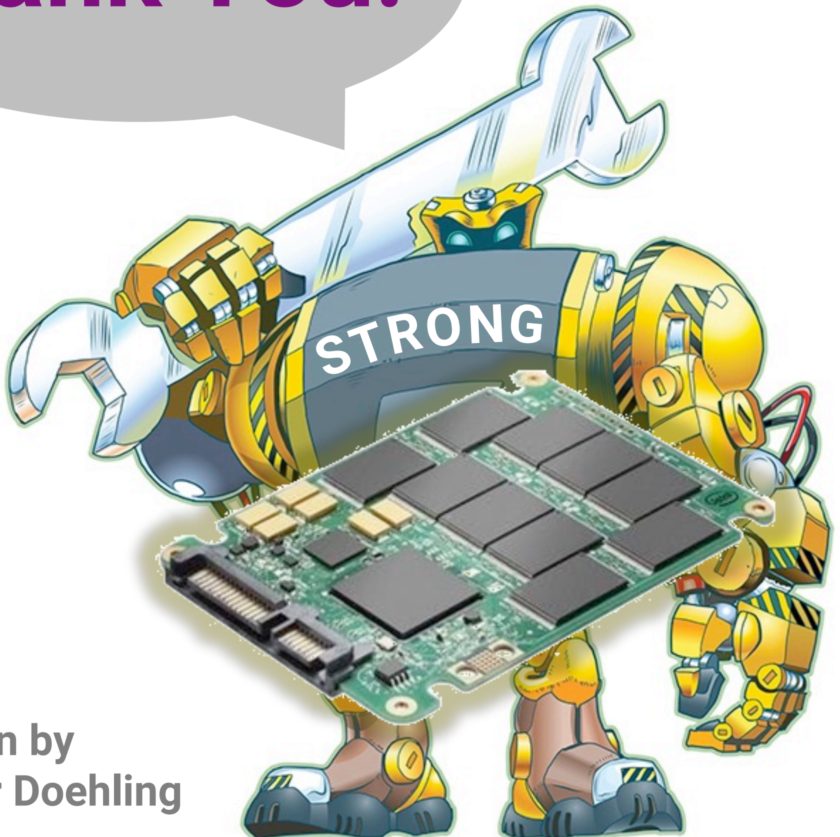
Yun-Sheng Chang and **Ren-Shuo Liu**

System and Storage Design Lab
Department of Electrical Engineering
National Tsing Hua University, Taiwan

國立清華大學
NATIONAL TSING HUA UNIVERSITY

USENIX ATC '19

Thank You!



Robot drawn by
Christopher Doehling

ssdlab.ee.nthu.edu.tw/optr
(Available before Aug 15)