StRAID: Stripe-threaded Architecture for Parity-based RAIDs with Ultra-fast SSDs

Shucheng Wang[1], Qiang Cao[1], Ziyi Lu[1], **Hong Jiang**[2],
Jie Yao[1] and Yuanyuan Dong[3]

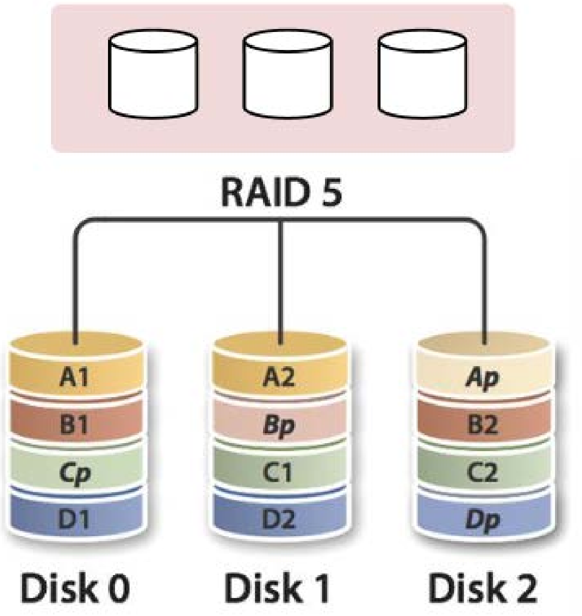
[1] Huazhong University of Science and Technology

[2] University of Texas Arlington

[3] Alibaba Group

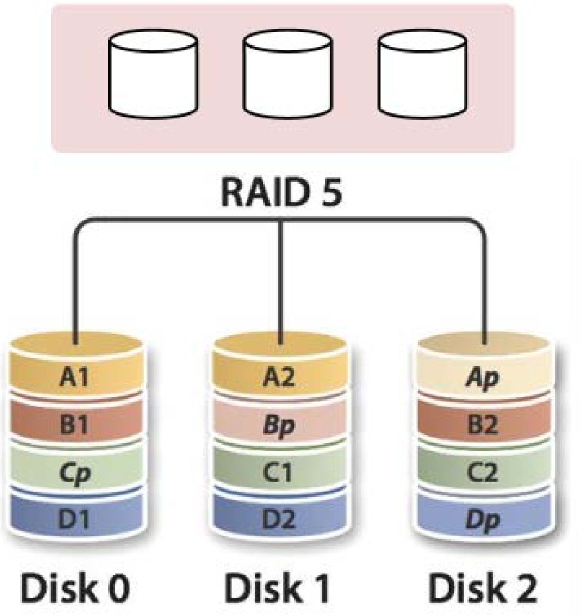
RAID Systems

- RAID (Redundant Array of Independent Disks) is widely used
  - Non-parity RAID:
    - ➢ RAID-0 (striping) and RAID-1 (mirroring)
  - **Parity-based RAID:**
    - ➢ RAID-4/5/6
    - ➢ Balancing performance and reliability
    - ➢ Read-modify-write nature



RAID Systems

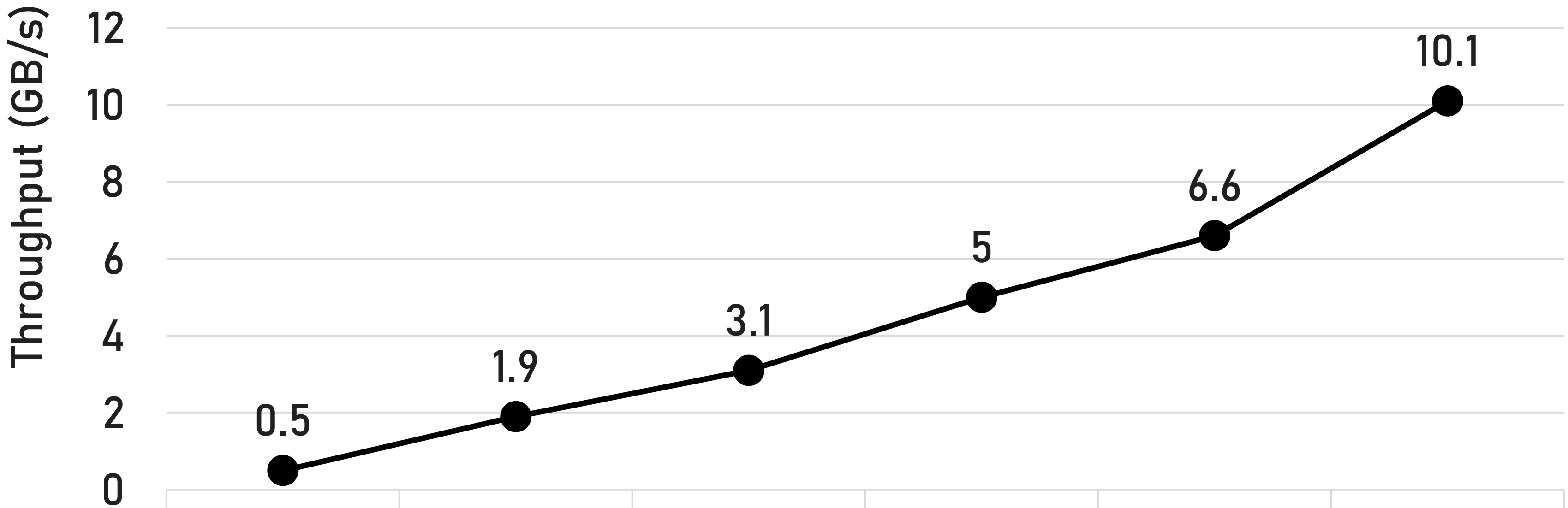
- RAID (Redundant Array of Independent Disks) is widely used
  - Non-parity RAID:
    - ➢ RAID-0 (striping) and RAID-1 (mirroring)
  - **Parity-based RAID:**
    - ➢ RAID-4/5/6
    - ➢ Balancing performance and reliability
    - ➢ Read-modify-write nature



- Linux MD: popular software RAID component
  - Linux kernel module
  - No need for extra hardware
  - Compatible with various storages

SSD Storage Trend

- Modern SSD hardware delivers higher write throughput



Products	Samsung 860 PRO	Intel DC P3700	Intel DC P4610	Samsung 980 PRO	Samsung PM1743	Phison
Interface	SATA 3.0	PCIe 3.0	PCIe 3.1	PCIe 4.0	PCIe 5.0	PCIe 5.0
Protocol	SATA	NVMe	NVMe	NVMe	NVMe	NVMe
Release Date	Q1'18	Q2'14	Q2'18	Q3'20	Q1'22	Unknown

Linux MD upon SSDs

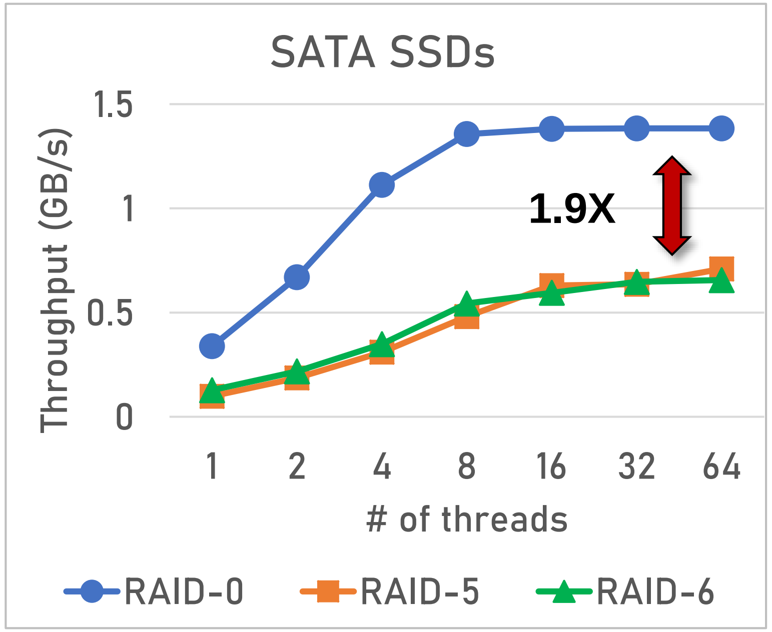
- Motivational Test
  - RAIDs setup
    - ➢ Non-parity: RAID-0 level
    - ➢ **Parity-based:** RAID-5 (5+1) and RAID-6 (4+2) level
      - – Enable the multi-worker mechanism
  - SSD products

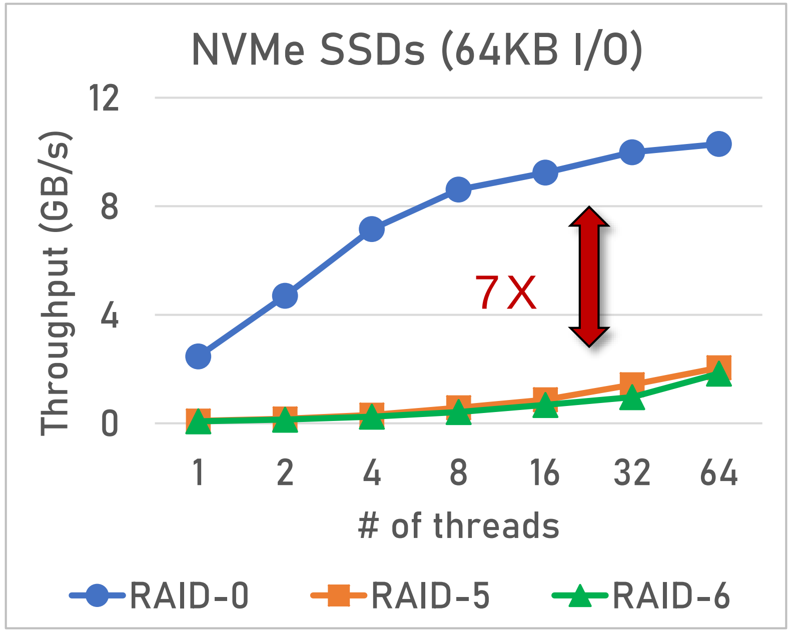
Device Types	Products	Capacity	Stable Write Throughput (MB/s)	Stable Read Throughput (MB/s)
SATA SSD	Samsung 860 PRO	512GB	500	510
NVMe SSD	Samsung 970 PRO	512GB	2200	3200
NVMe SSD	Samsung 980 PRO	1TB	**2600**	6900

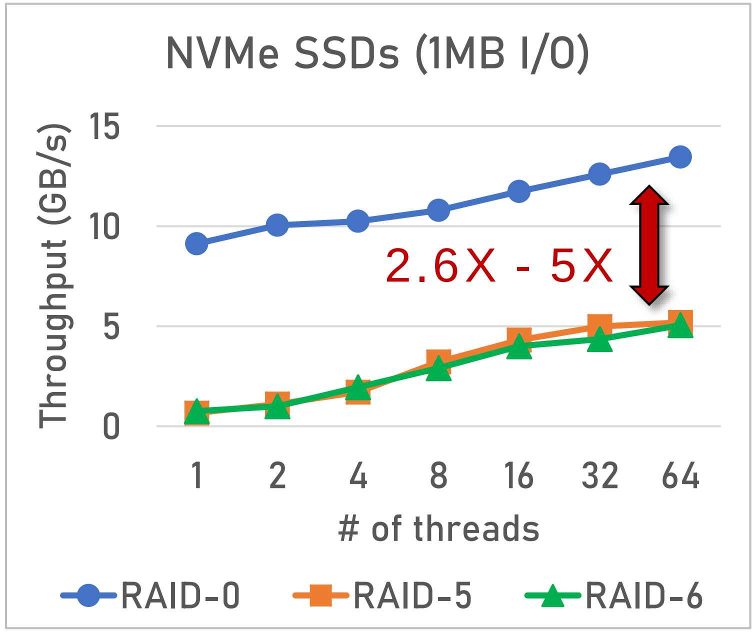
> 14 GB/s total write bandwidth on six SSDs

Multi-thread Write Scalability

- Parity-based RAIDs fail to scale for high-performance SSDs
  - ➢ Larger performance gap on fast SSDs
  - ➢ Full-stripe writes (1MB, without read-modify-write) still suffers

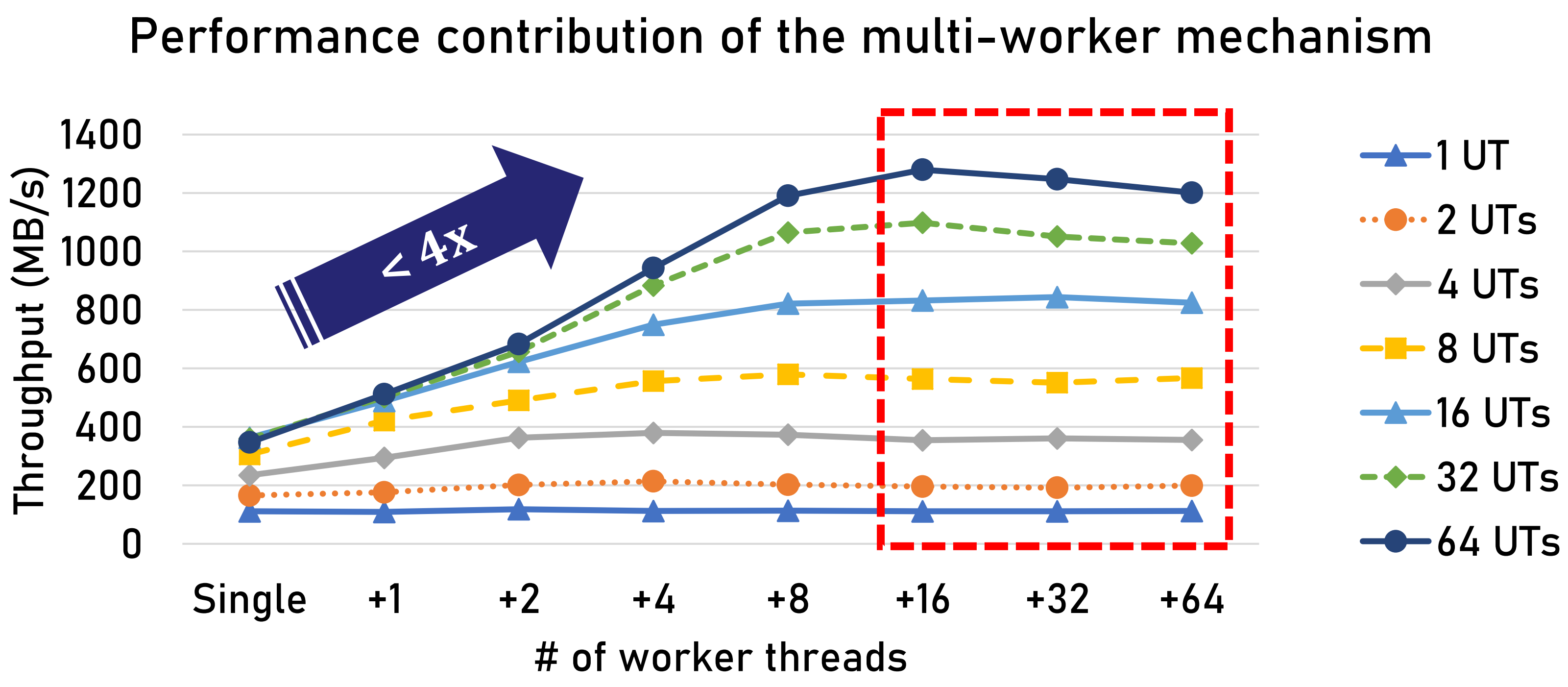






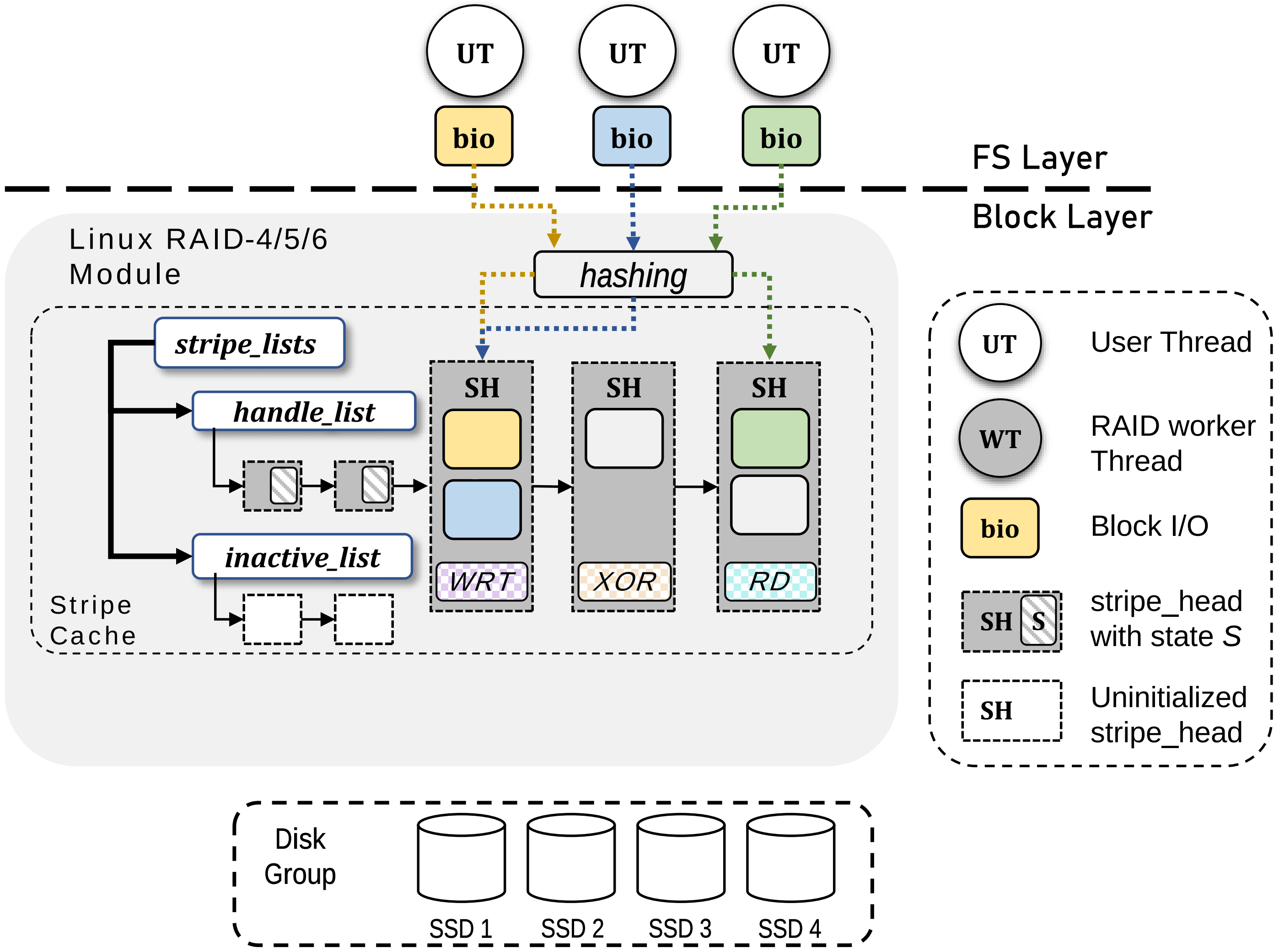
Multi-thread Write Scalability

- Parity-based RAIDs fail to scale for high-performance SSDs
  - A diminishing return in performance of the multi-worker mechanism
    - ➢ Throughput gains peak at +16 worker threads (WTs)
    - ➢ 5% decline with more WTs



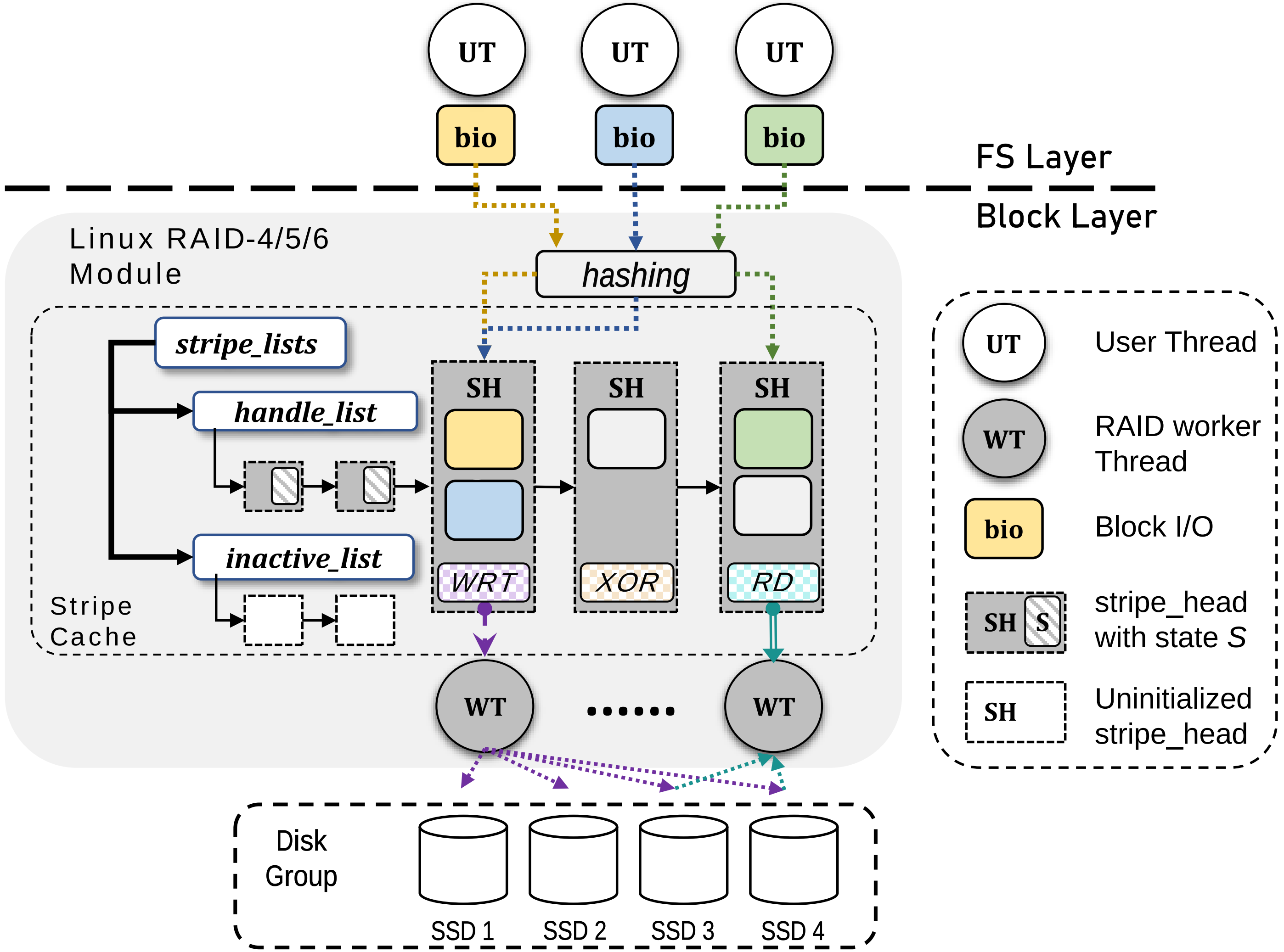
Analysis of MD

- "N-for-all" processing model
  - Incoming block I/Os are temporarily stored in the Stripe Cache
    - Aggregate bios at the granularity of stripes
    - Use stripe_heads (**SH**) to maintain stripe states
    - Store SHs in stripe_lists



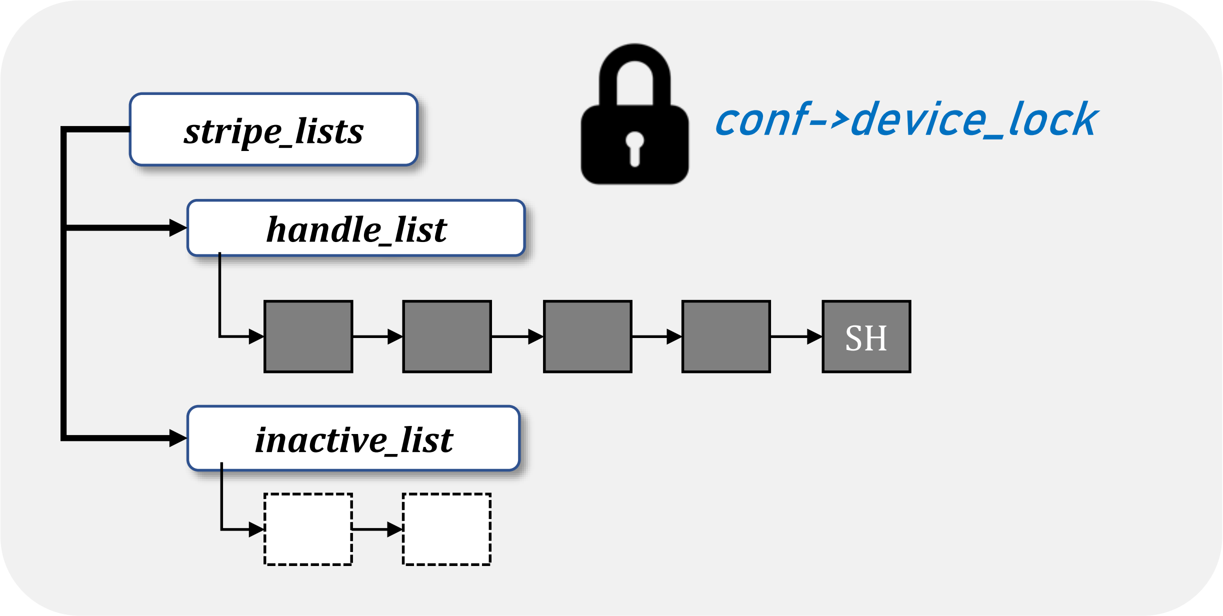
Analysis of MD

- "N-for-all" processing model
  - Incoming block I/Os are temporarily stored in the Stripe Cache
  - A set number of WTs asynchronously and non-exclusively handle stripe-write tasks



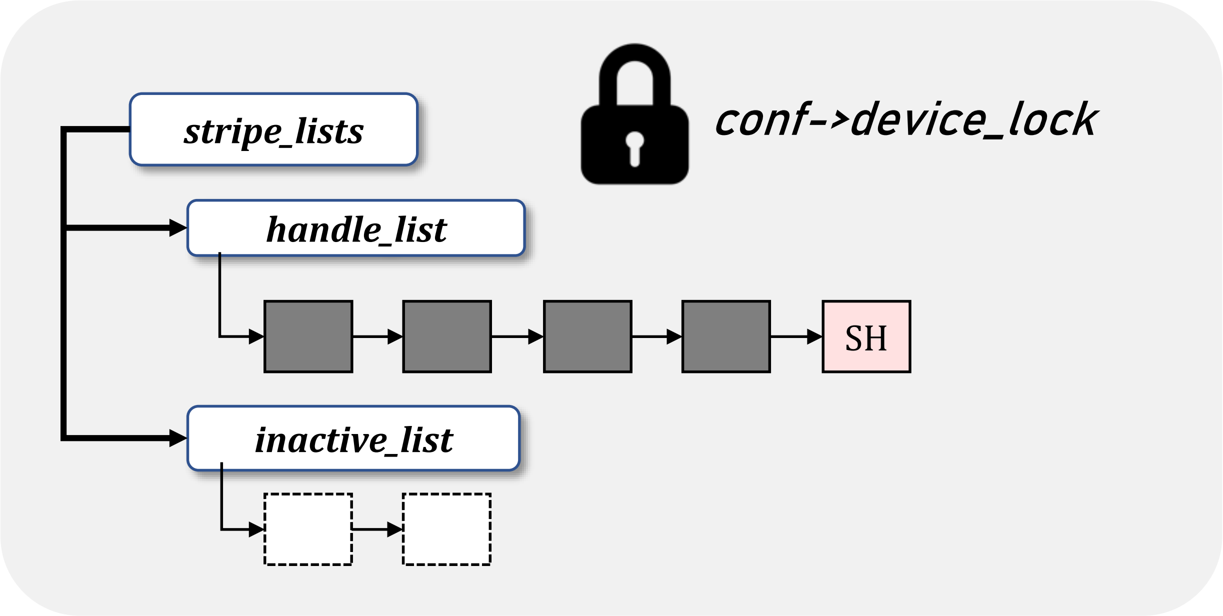
Analysis of MD

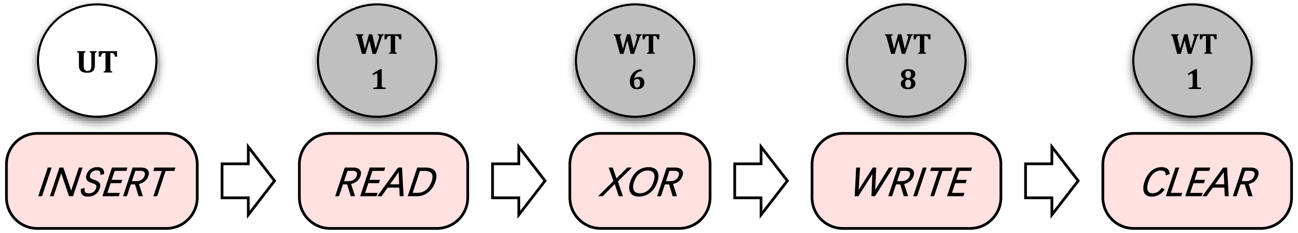
- MD's concurrency control
  - ➢ The device_lock in MD
    - A **spin-lock** shared between WTs
    - For updating shared structures (stripe_lists and metadata, etc.)



Analysis of MD

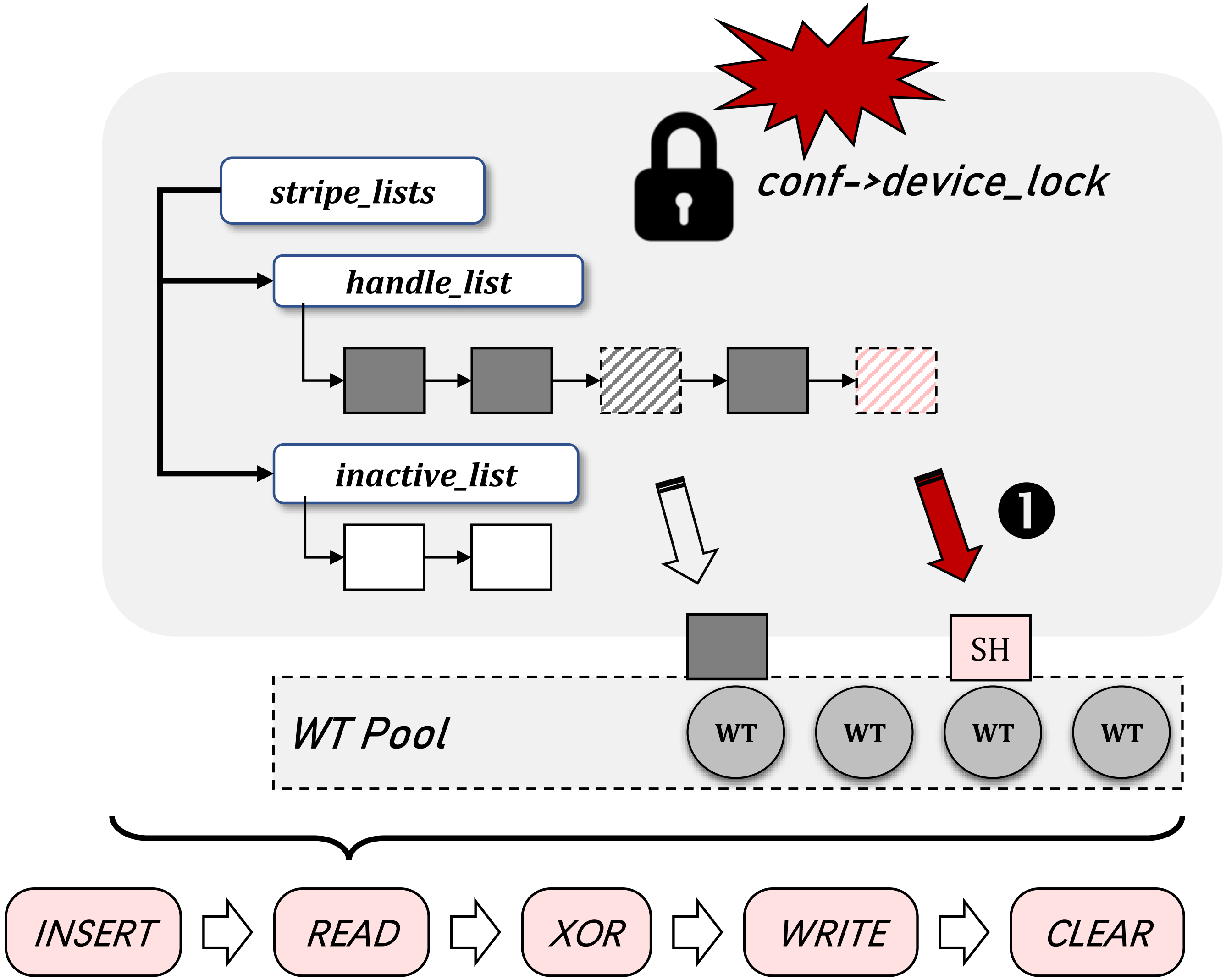
- MD's concurrency control
  - ➢ MD device lock
    - A spin-lock shared between WTs
    - For updating shared structures (stripe_lists and metadata, etc)
  - ➢ Stripe-write workflow:
    - Multi-stage stripe processing





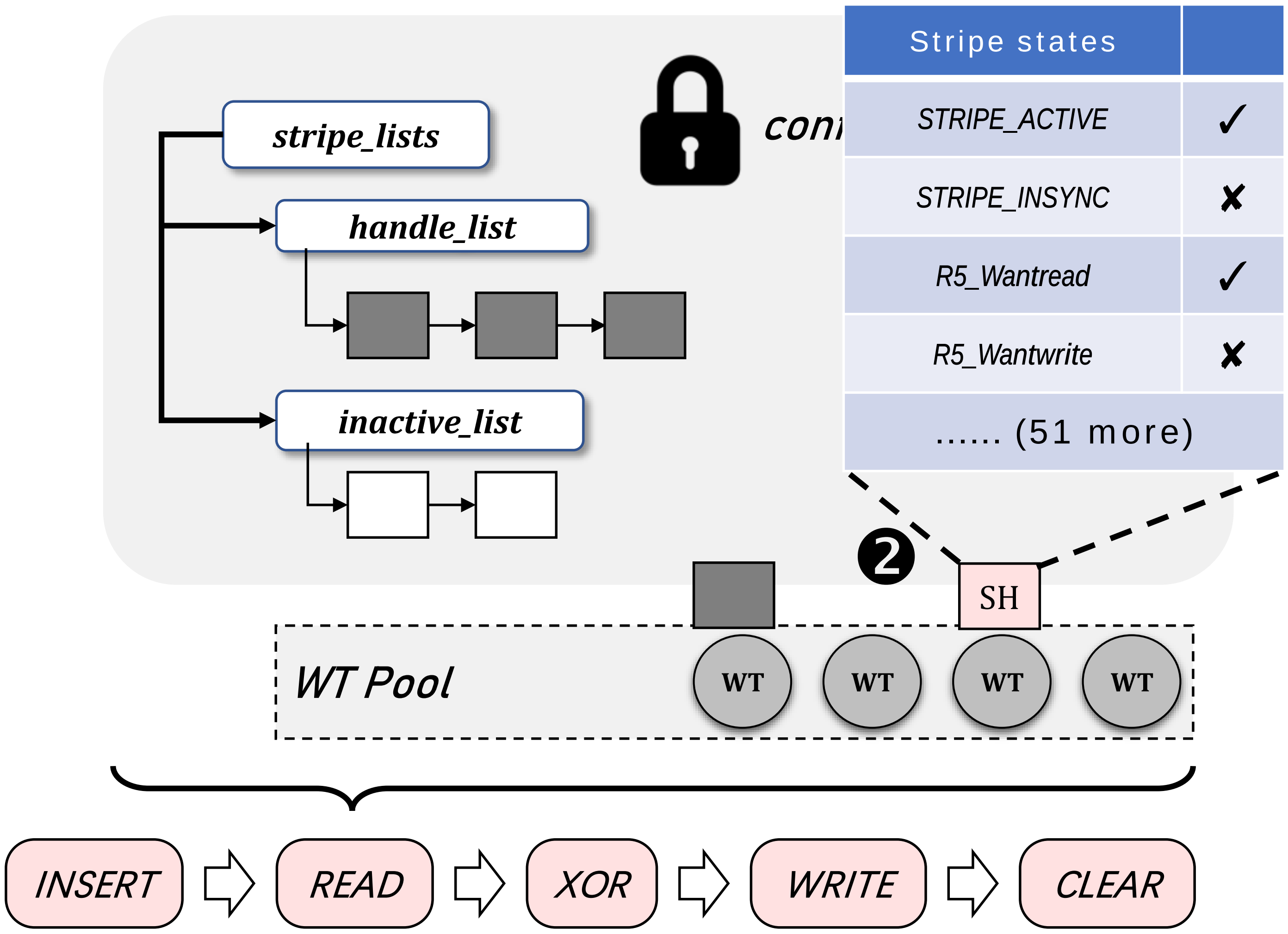
Analysis of MD

- MD's concurrency control
  - ➢ MD device lock
    - A spin-lock shared between WTs
    - For updating shared structures (stripe_lists and metadata, etc)
  - ➢ Stripe-write workflow:
    - Multi-stage stripe processing
    - Four handling steps in each stage
      1. ***Fetch*** *a SH from handle_list*



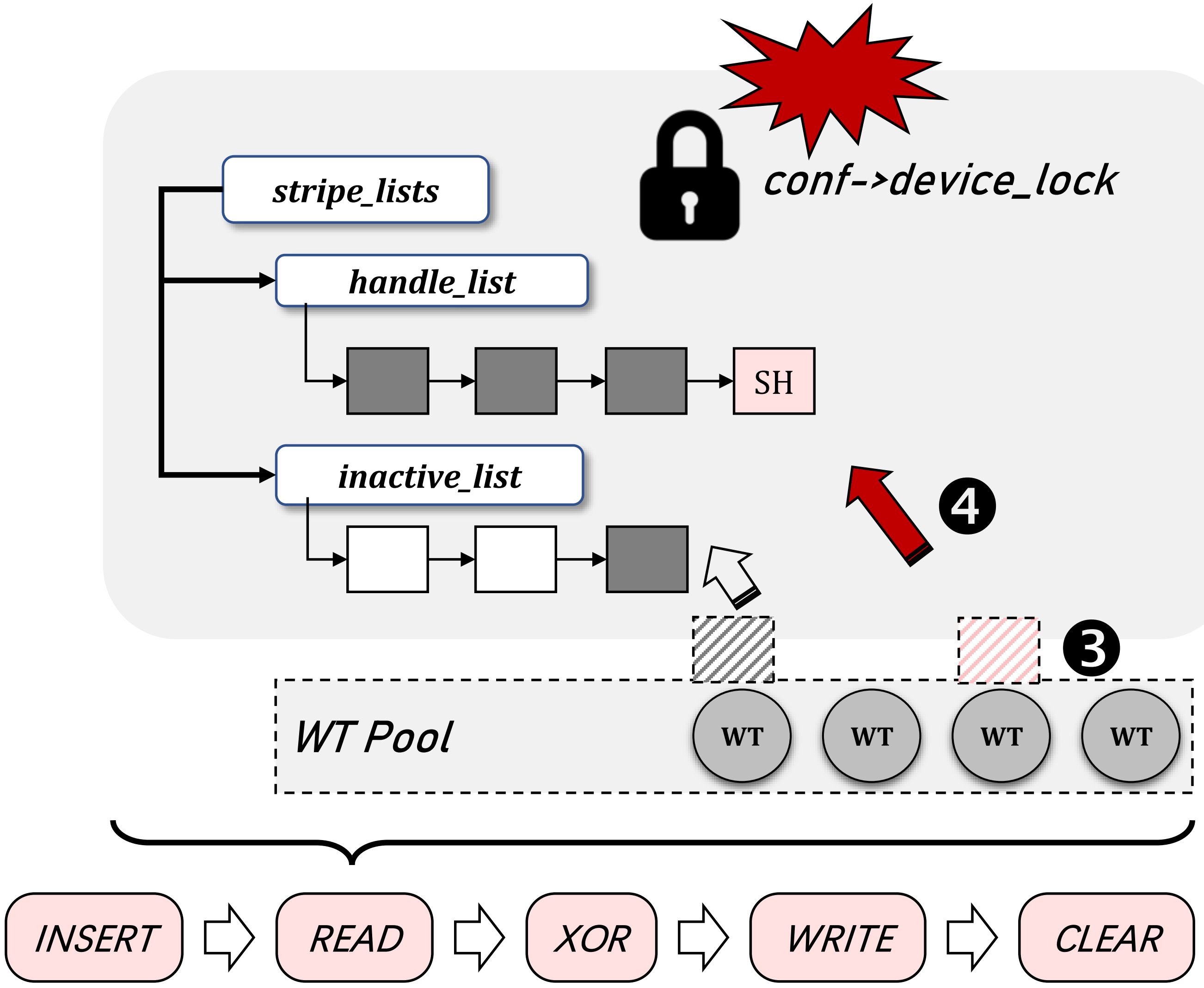
Analysis of MD

- MD's concurrency control
  - ➢ MD device lock
    - A spin-lock shared between WTs
    - For updating shared structures (stripe_lists and metadata, etc)
  - ➢ Stripe-write workflow:
    - Multi-stage stripe processing
    - Four handling steps in each stage
      1. *Fetch a SH from handle_list*
      2. ***Analyze** stripe & device states*
         - *Use semaphores*
         - *Need rcu_read_lock*



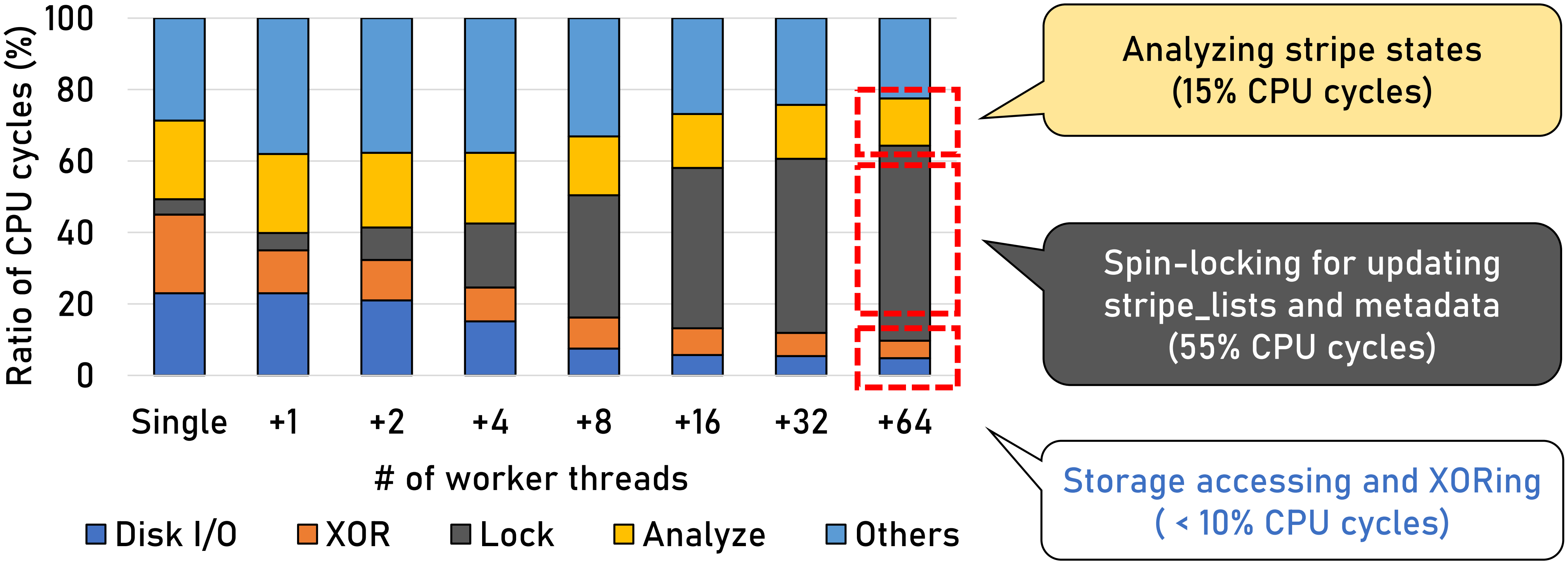
Analysis of MD

- MD's concurrency control
  - MD device lock
    - A spin-lock shared between WTs
    - For updating shared structures (stripe_lists and metadata, etc)
  - Stripe-write workflow:
    - Multi-stage stripe processing
    - Four handling steps in each stage
      1. *Fetch a SH from handle_list*
      2. *Analyze stripe & device states*
      3. *Operations for handling stripe*
      4. ***Release** and insert the SH into a stripe_list*



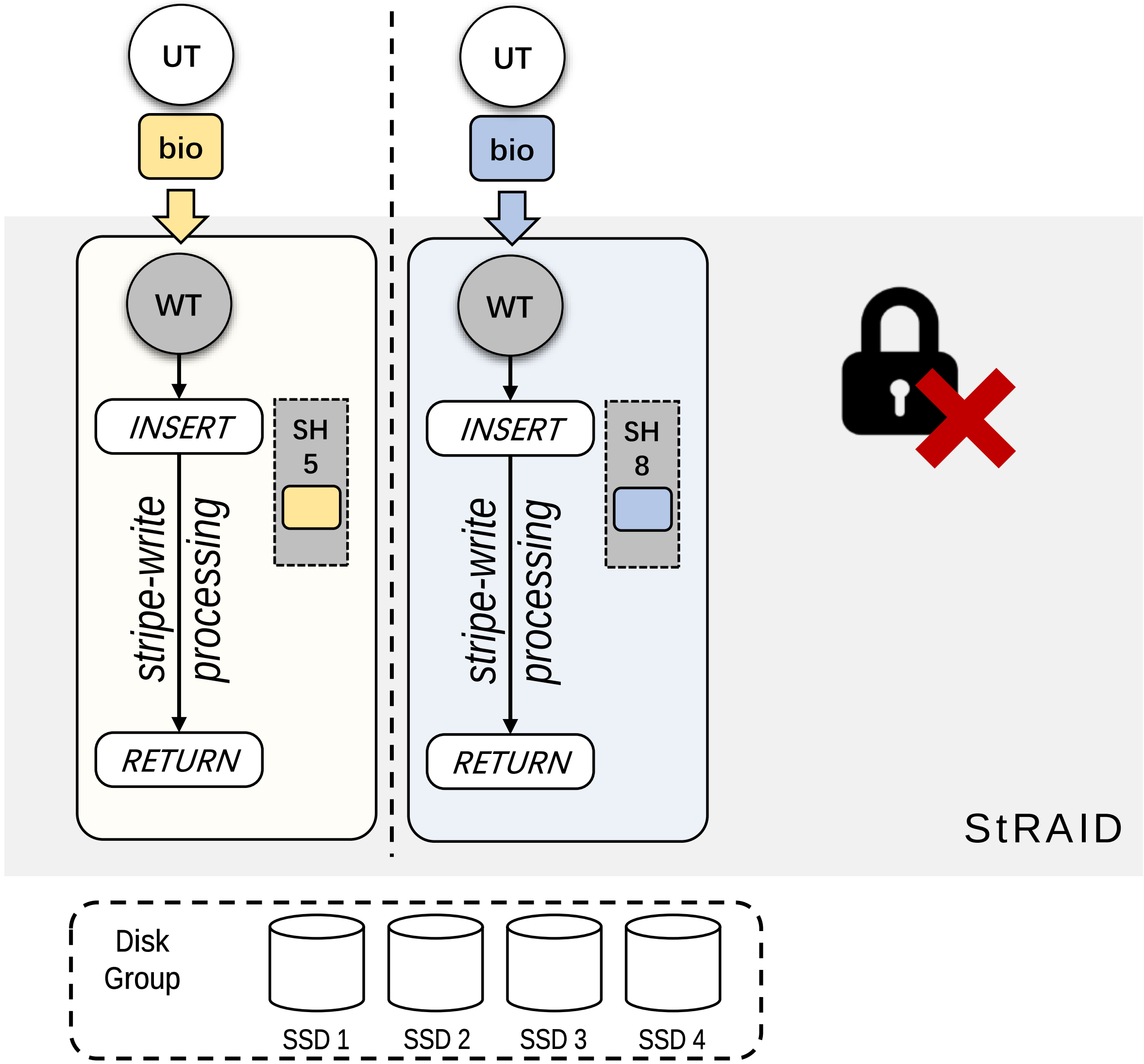
Analysis of MD

- Breakdown of CPU cycles on critical functions and locks in WTs
  - ➢ CPU becomes the bottleneck on concurrency control
  - ➢ Few CPU cycles are used to drive I/Os → storage devices are underutilized



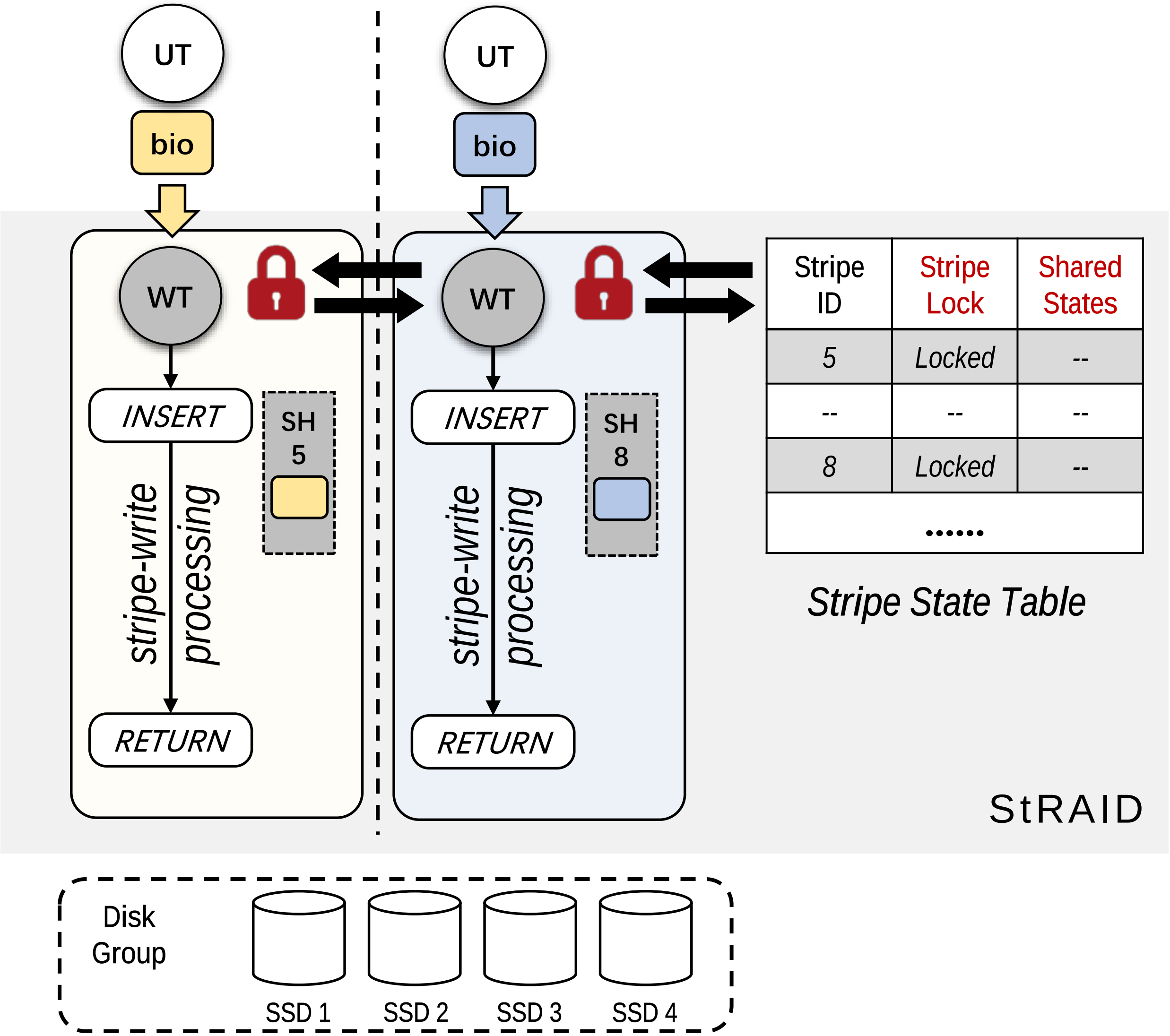
StRAID overview

- "One-for-one" processing model
  - Goals:
    - ➢ Efficient CPU utilization
    - ➢ Reduce partial-stripe-write penalty
  - Stripe-threaded architecture
    - ➢ **Dedicated WT** for each stripe-write
      - Eliminate global lock
      - Reduce stripe state checking



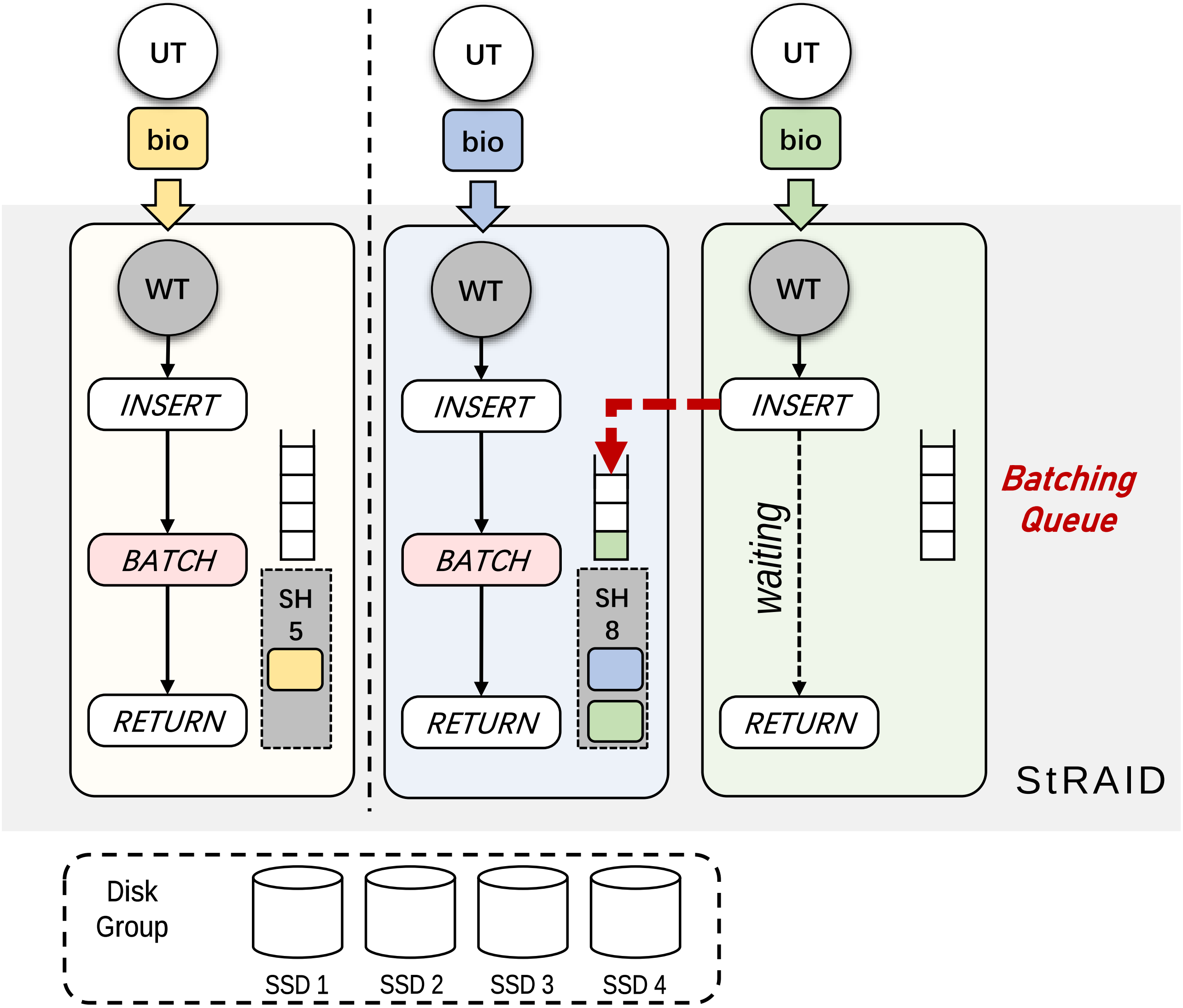
StRAID overview

- "One-for-one" processing model
  - Goals:
    - ➢ Efficient CPU utilization
    - ➢ Address partial-stripe-write penalty
  - Stripe-threaded architecture
    - ➢ Dedicated WT for each stripe-write
      - Eliminate global lock contention
      - Reduce stripe state checking
    - ➢ Stripe State Table
      - Conduct **thread collaboration**
      - Maintain indispensable shared stripe states and per-stripe locks



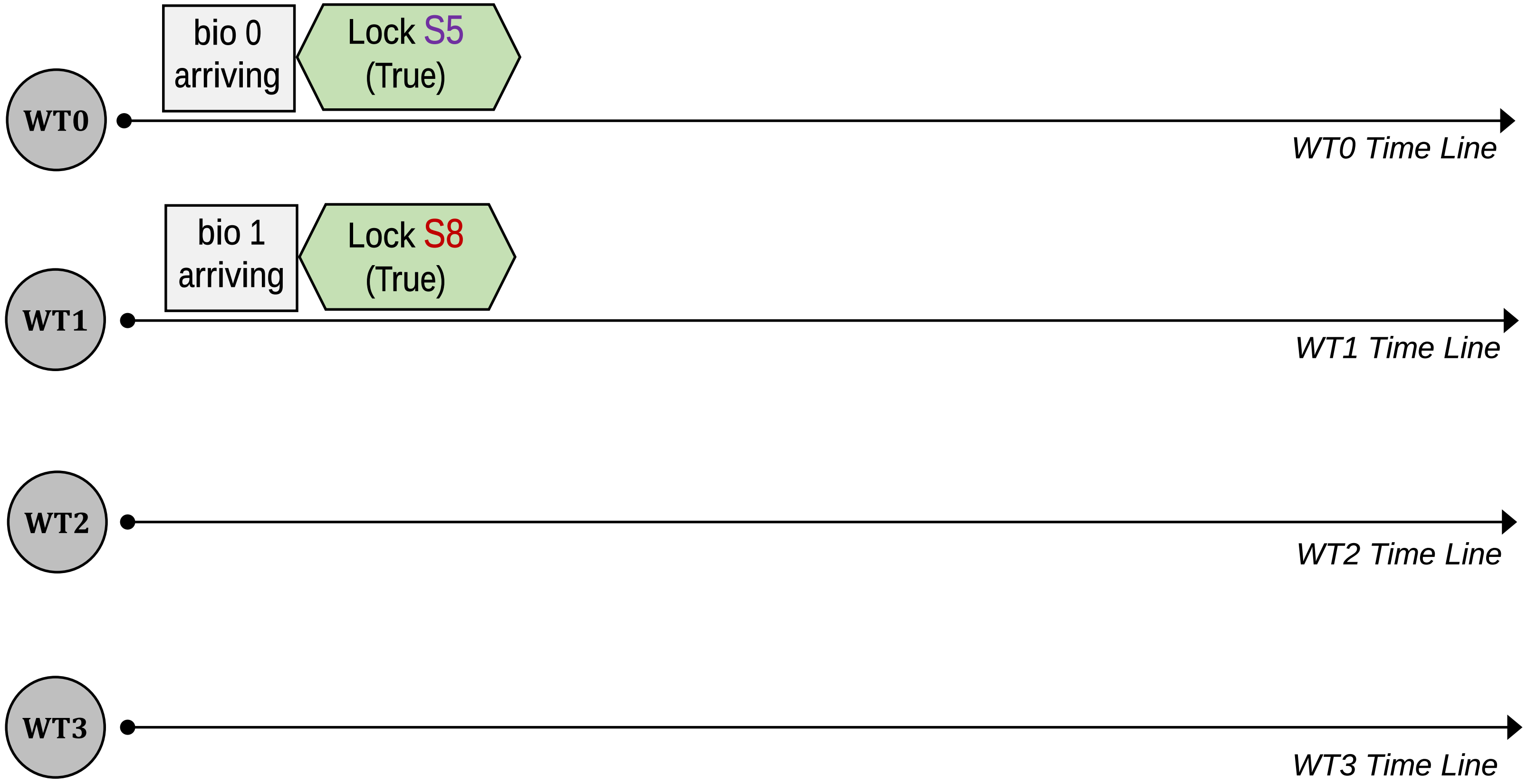
StRAID overview

- "One-for-one" processing model
  - Goals:
    - ➢ Efficient CPU utilization
    - ➢ Address partial-stripe-write penalty
  - Stripe-threaded architecture
    - ➢ Dedicated WT for each stripe-write
      - Eliminate global lock contention
      - Reduce stripe state checking
    - ➢ Stripe State Table
      - Conduct thread collaboration
      - Maintain indispensable shared stripe states and per-stripe locks
    - ➢ Two-phase stripe submission
      - Opportunistic **write batching**
      - Per-stripe batching queue



StRAID's Concurrency Control

- **An example:** four I/O threads issue block I/Os
  - *bio* 0 → *stripe* 5
  - *bio* 1 *to bio* 3 → *stripe* 8

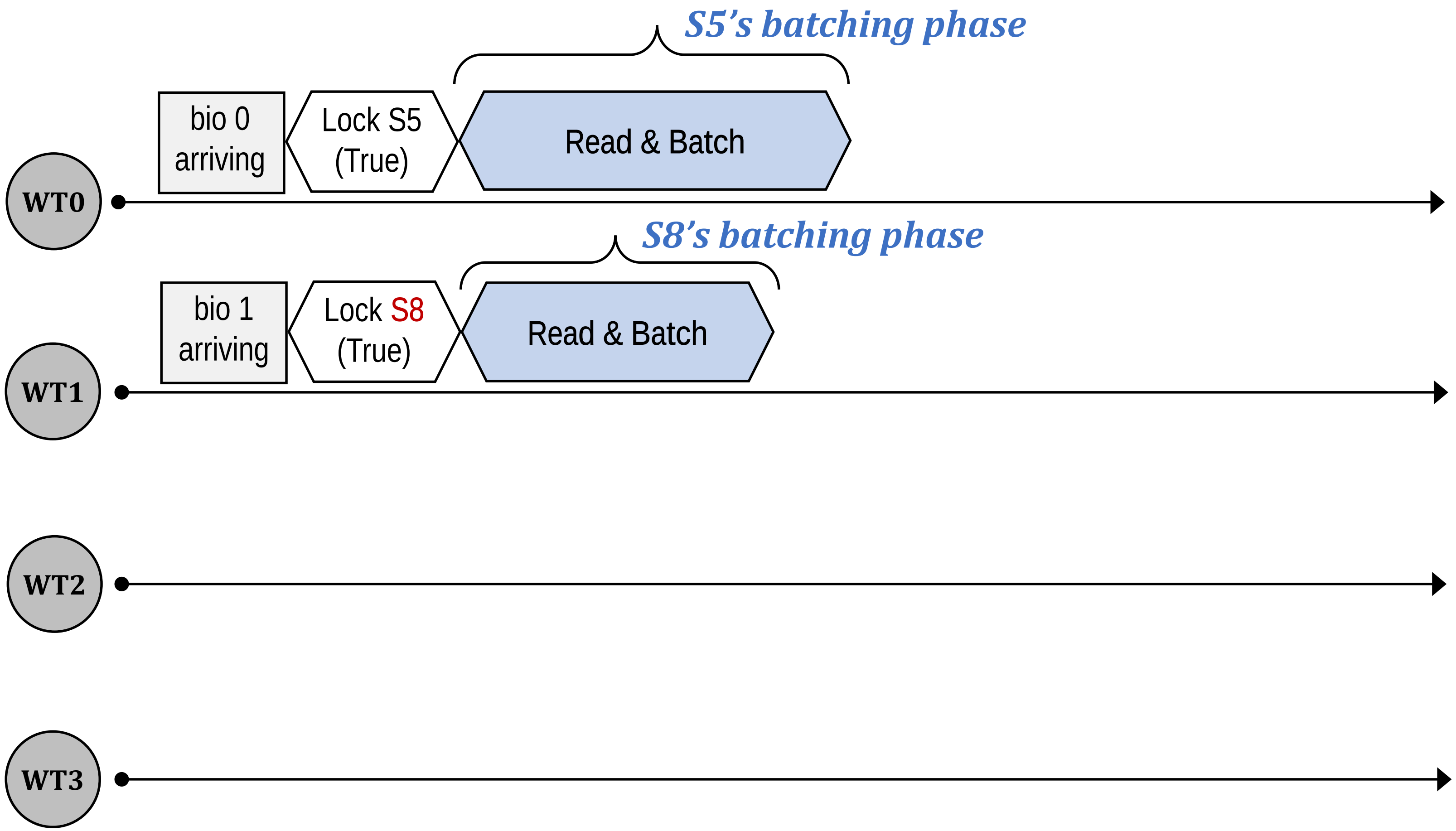


Stripe State Table

Stripe ID	Stripe Lock	TID	is_batching
5	*Locked*	*0*	*True*
--	--	--	--
8	*Locked*	*1*	*True*
......			

StRAID's Concurrency Control

- Dedicated WT aggregates requests targeting the same stripe in the batching phase

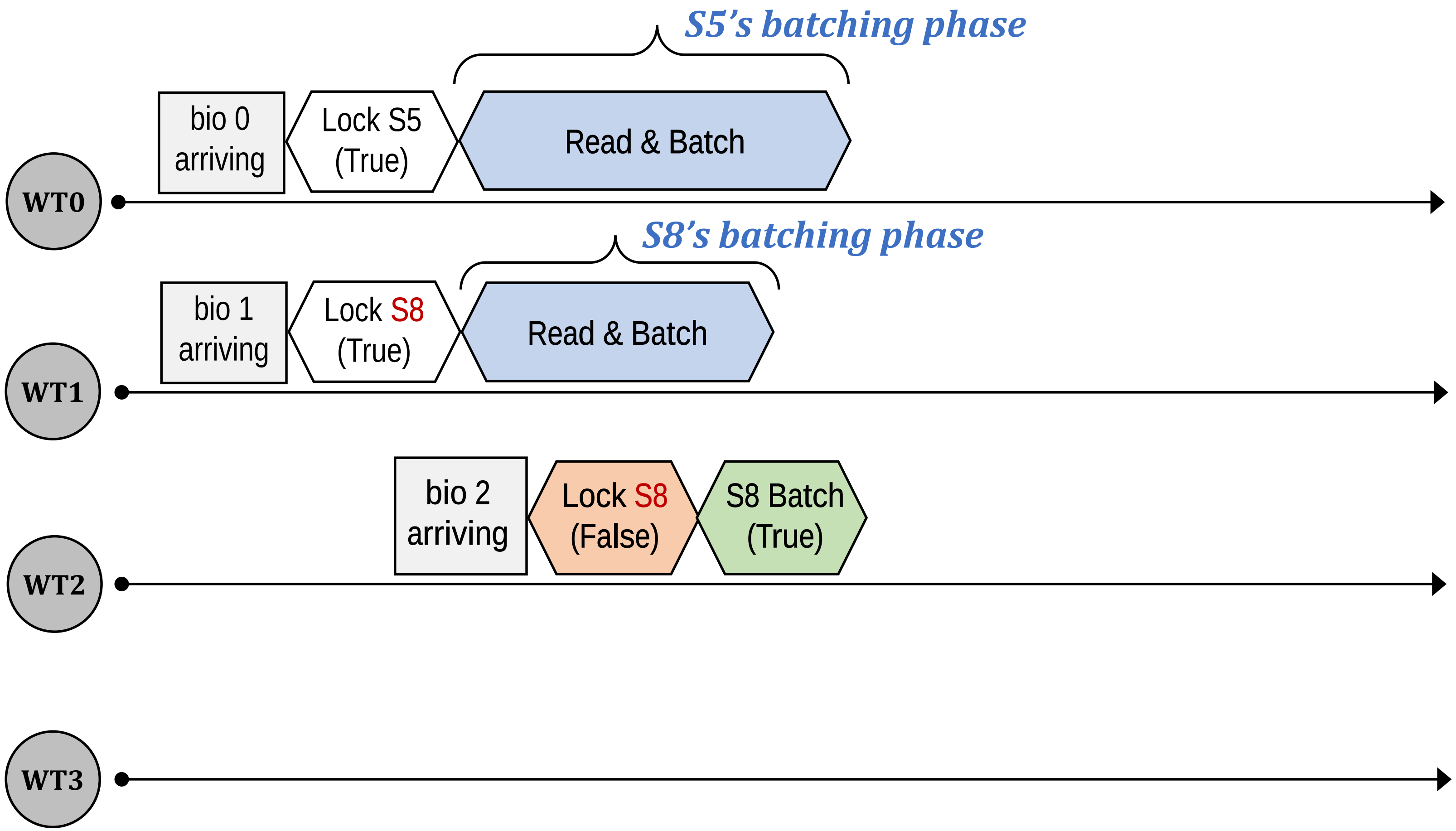


Stripe State Table

Stripe ID	Stripe Lock	TID	is_batching
5	*Locked*	*0*	*True*
--	--	--	--
8	*Locked*	*1*	*True*
......			

StRAID's Concurrency Control

- Dedicated WT aggregates requests targeting the same stripe in the batching phase

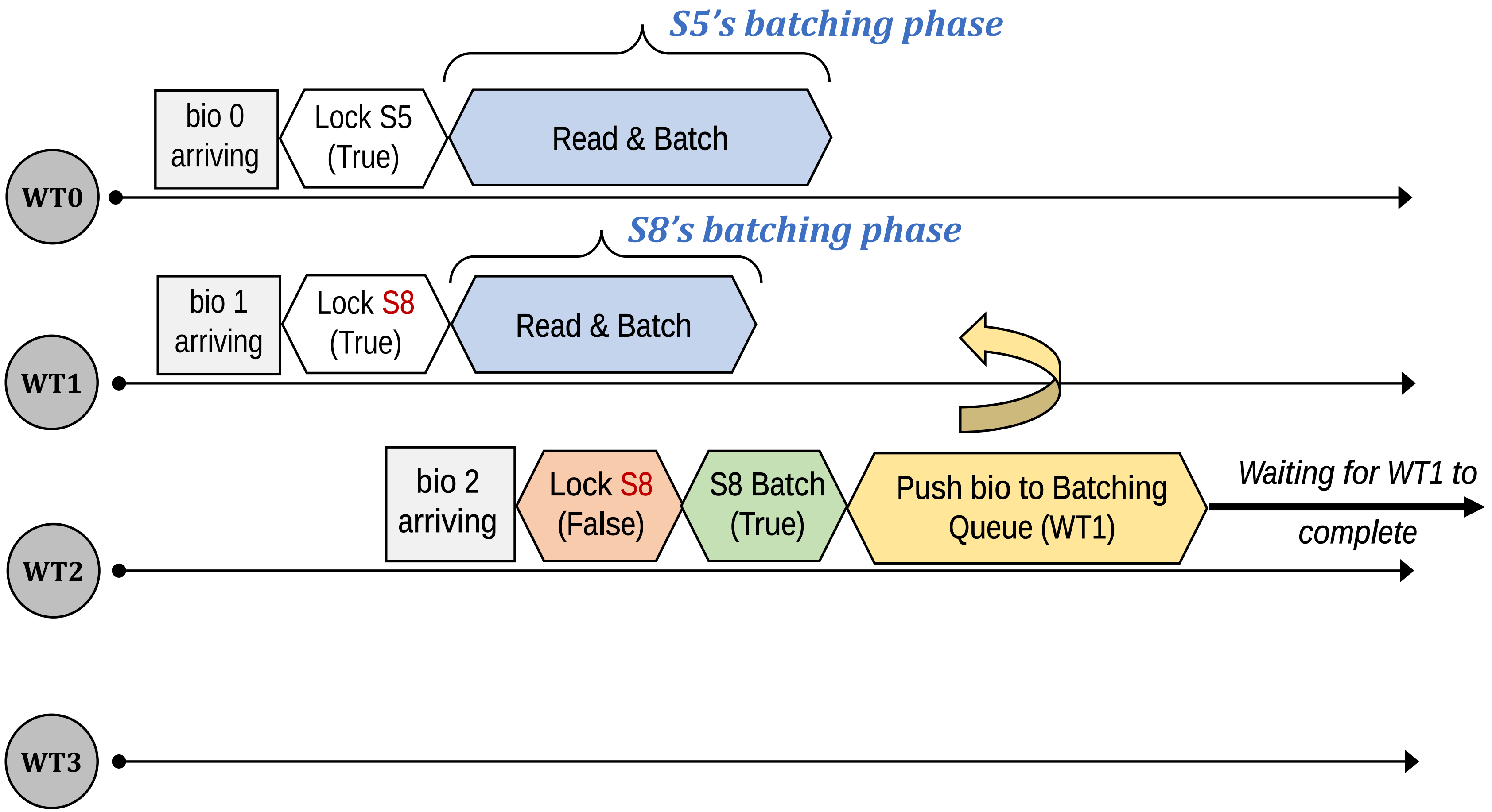


Stripe State Table

Stripe ID	Stripe Lock	TID	is_batching
5	*Locked*	*0*	*True*
--	--	--	--
8	*Locked*	*1*	*True*
......			

StRAID's Concurrency Control

- Dedicated WT aggregates requests targeting the same stripe in the batching phase

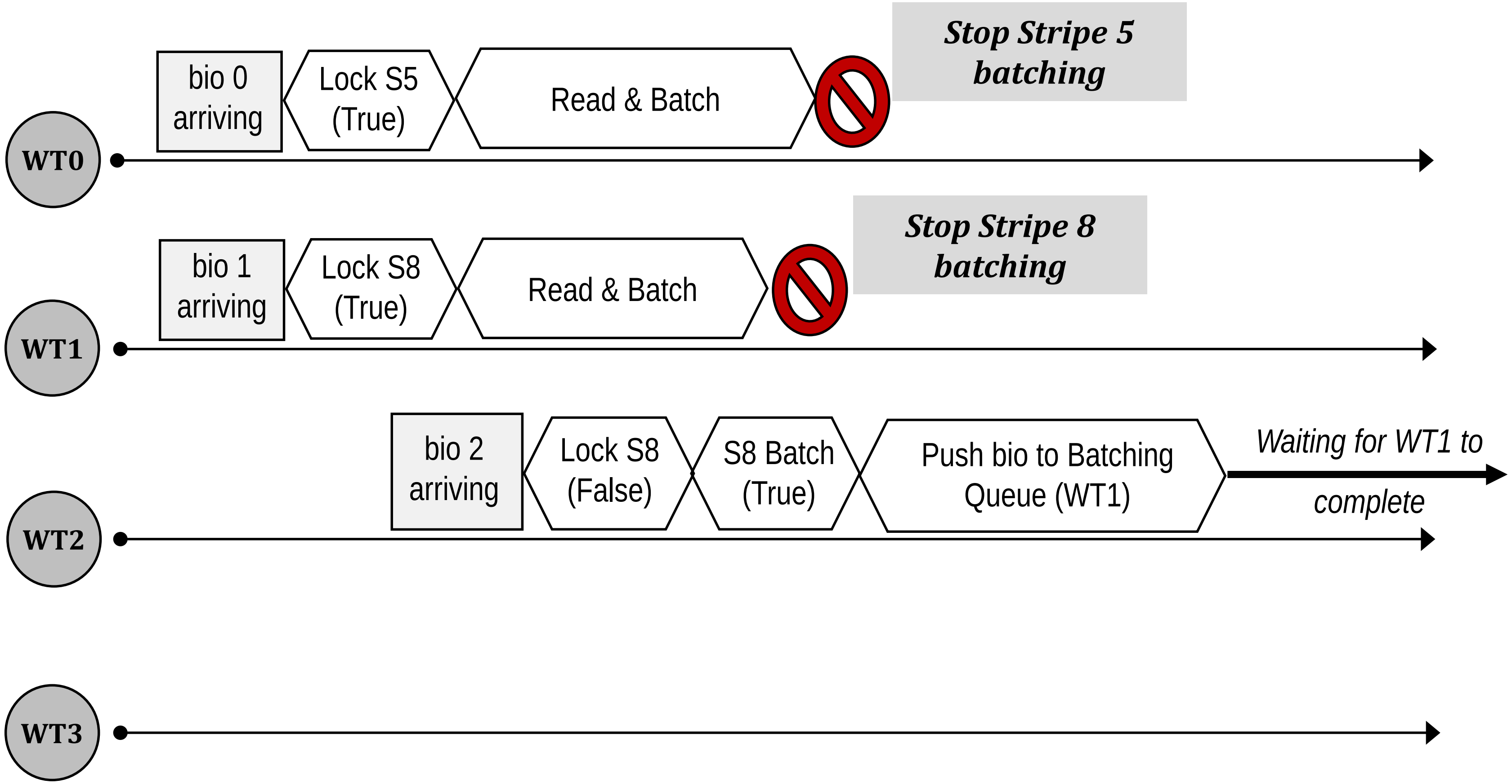


Stripe State Table

Stripe ID	Stripe Lock	TID	is_batching
5	*Locked*	*0*	*True*
--	--	--	--
8	*Locked*	*1*	*True*
......			

StRAID's Concurrency Control

- Dedicated WT stops batching phase after reading complete

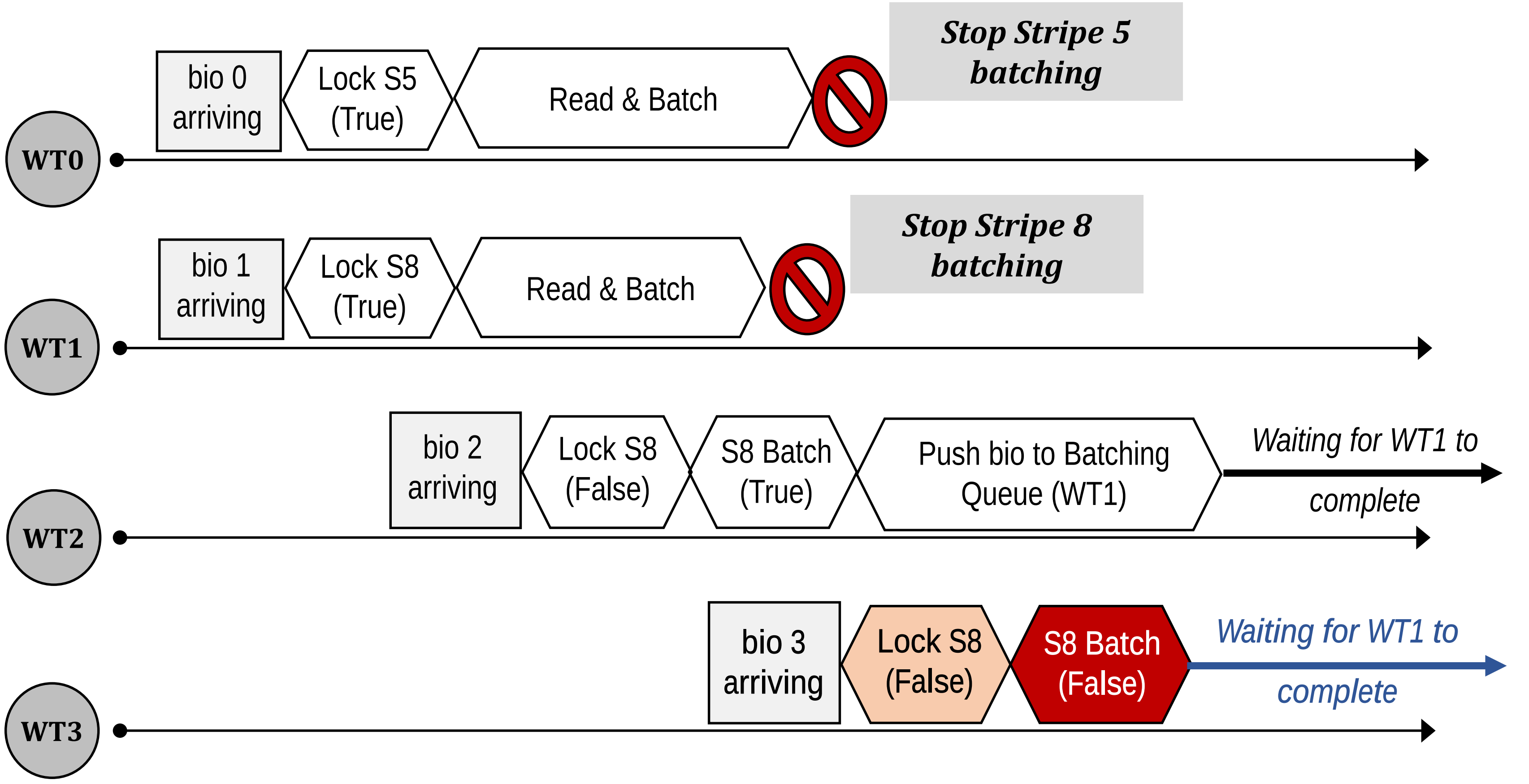


Stripe State Table

Stripe ID	Stripe Lock	TID	is_batching
5	*Locked*	*0*	*False*
--	--	--	--
8	*Locked*	*1*	*False*
......			

StRAID's Concurrency Control

- Dedicated WT stops batching phase after reading complete
- Requests failed to batch must **wait** for the dedicated WT to complete

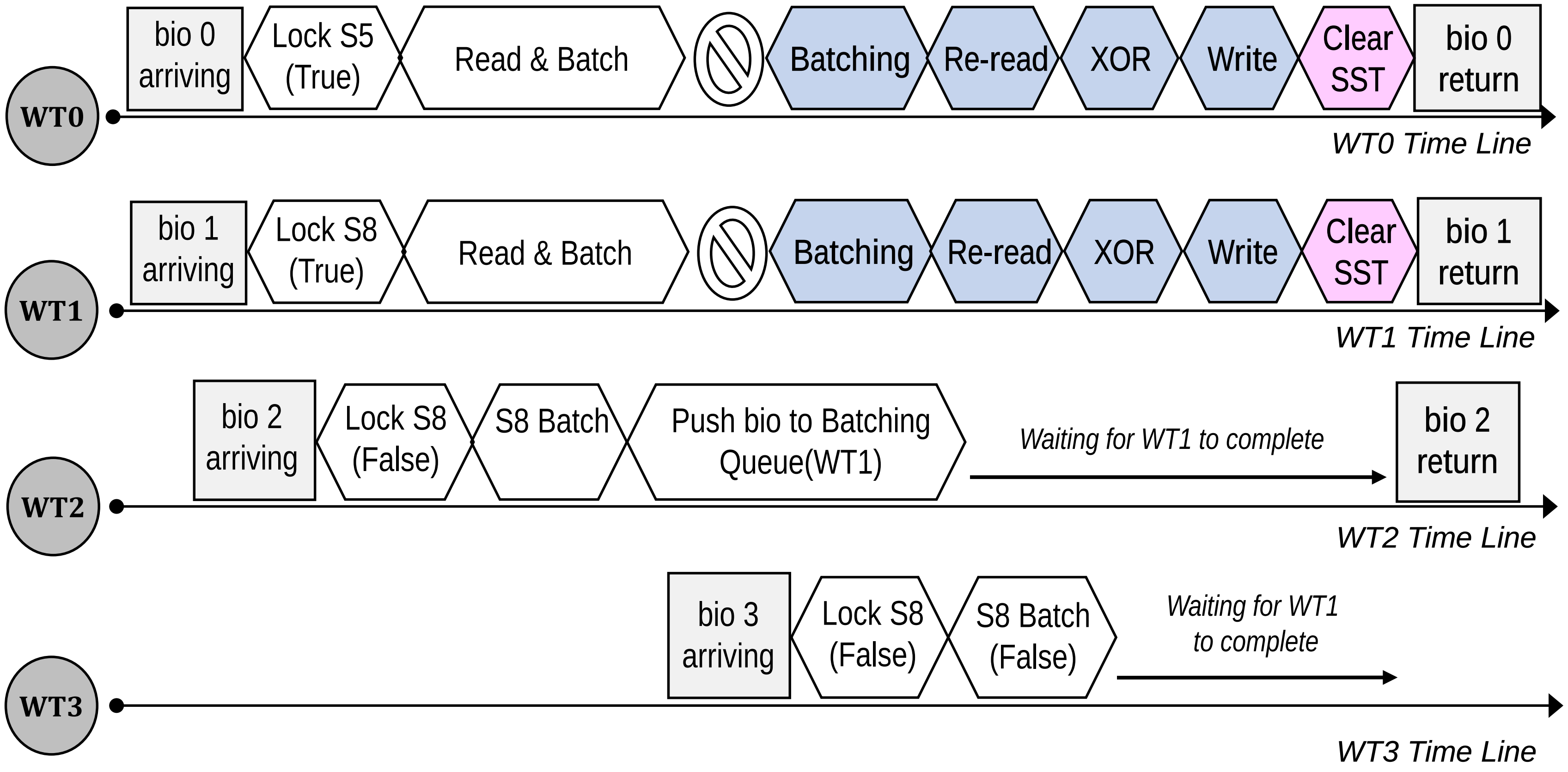


Stripe State Table

Stripe ID	Stripe Lock	TID	is_batching
5	*Locked*	*0*	*False*
--	--	--	--
8	*Locked*	*1*	*False*
......			

StRAID's Concurrency Control

- After completing stripe processing, WT cleans up SST-entry and returns I/O

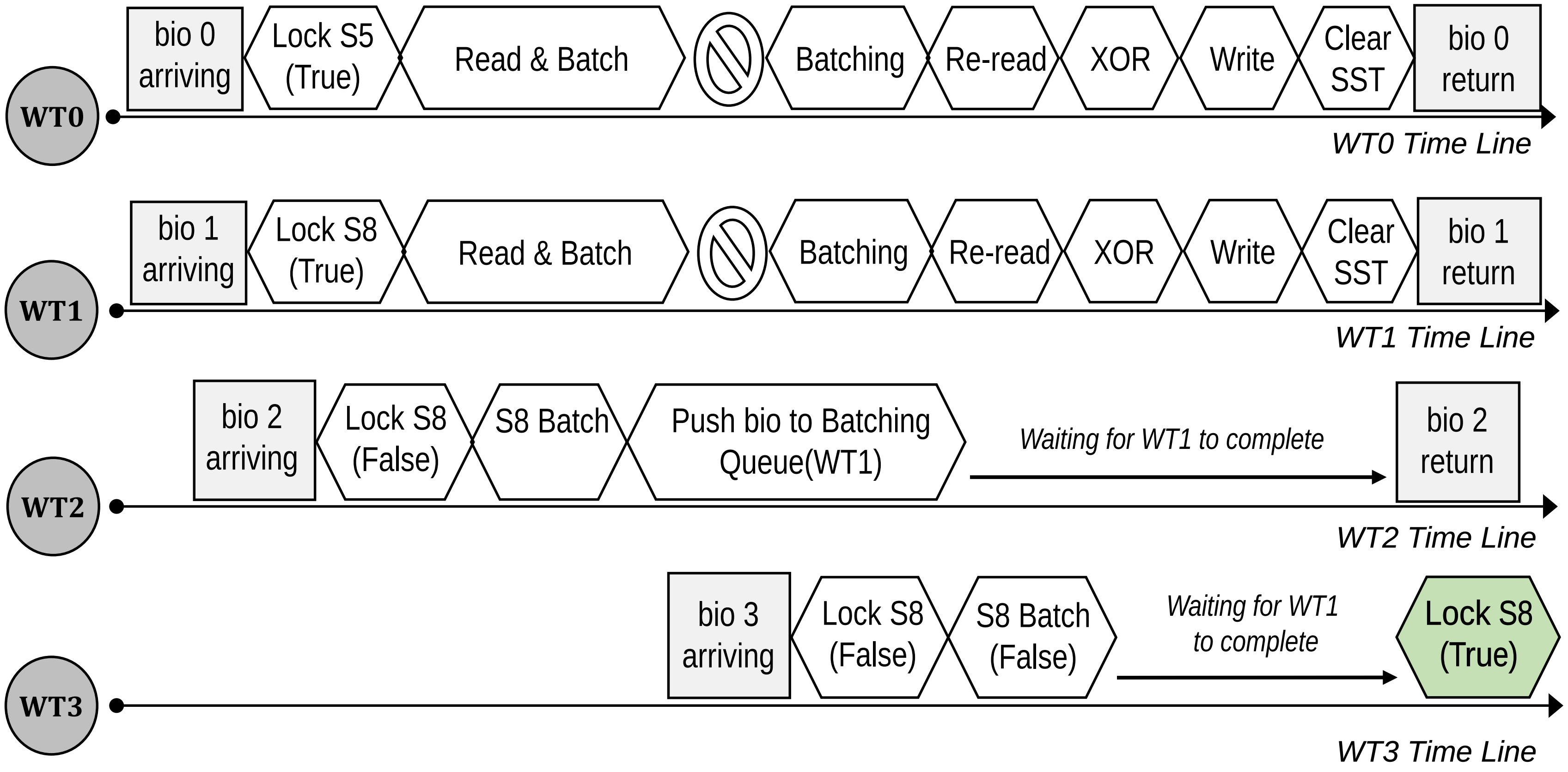


Stripe State Table

Stripe ID	Stripe Lock	TID	is_batching
5	*Unlocked*	--	--
--	--	--	--
8	*Unlocked*	--	--
......			

StRAID's Concurrency Control

- The waiting WT will try to re-acquire the stripe lock



Stripe State Table

Stripe ID	Stripe Lock	TID	is_batching
5	*Unlocked*	--	--
--	--	--	--
8	*Locked*	*3*	*True*
......			

Evaluation Setup

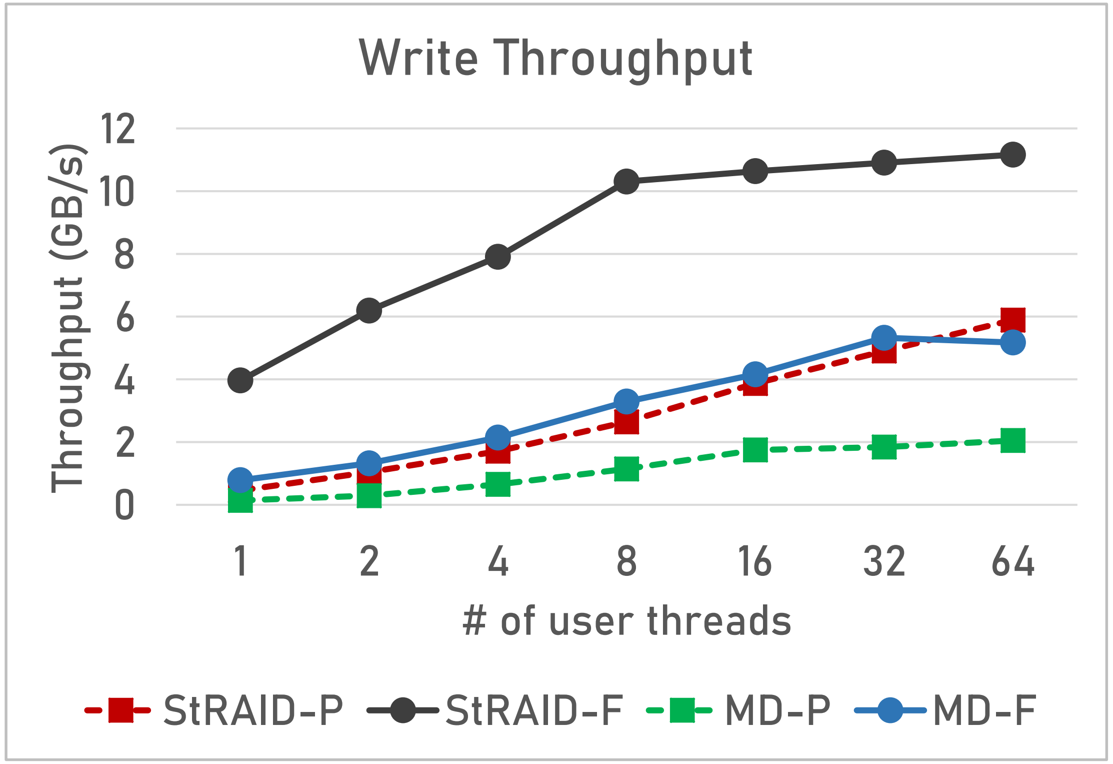
- Platform

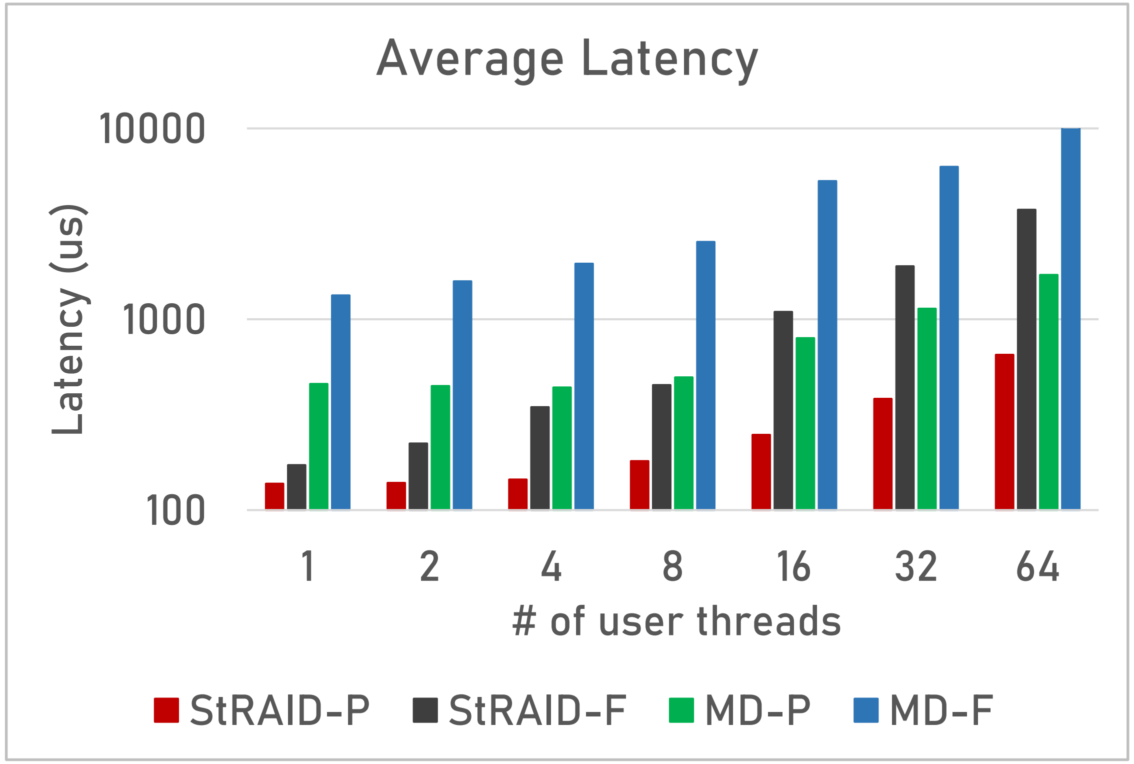
System	Linux kernel version 5.13
CPU	Duel Intel Xeon Gold 6328 CPU (totally 56 physical cores)
Memory	256 GB
SSD	6 x Samsung 970PRO (NVMe, 2.2GB/s stable write)
	6 x Samsung 980PRO (NVMe, 2.6GB/s stable write)
PM	6 x Intel Optane PM 128GB (2.3GB/s stable write)

- RAID setup
  - ➢ RAID-5 (5+1) and RAID-6 (4+2) with 64KB chunk size
- Workloads
  - ➢ Micro-benchmarks: partial-stripe writes (64KB) and full-stripe writes (1MB)
  - ➢ Macro-benchmarks: traces from Microsoft, Ali-Pangu, and Filebench

Micro-benchmark Results

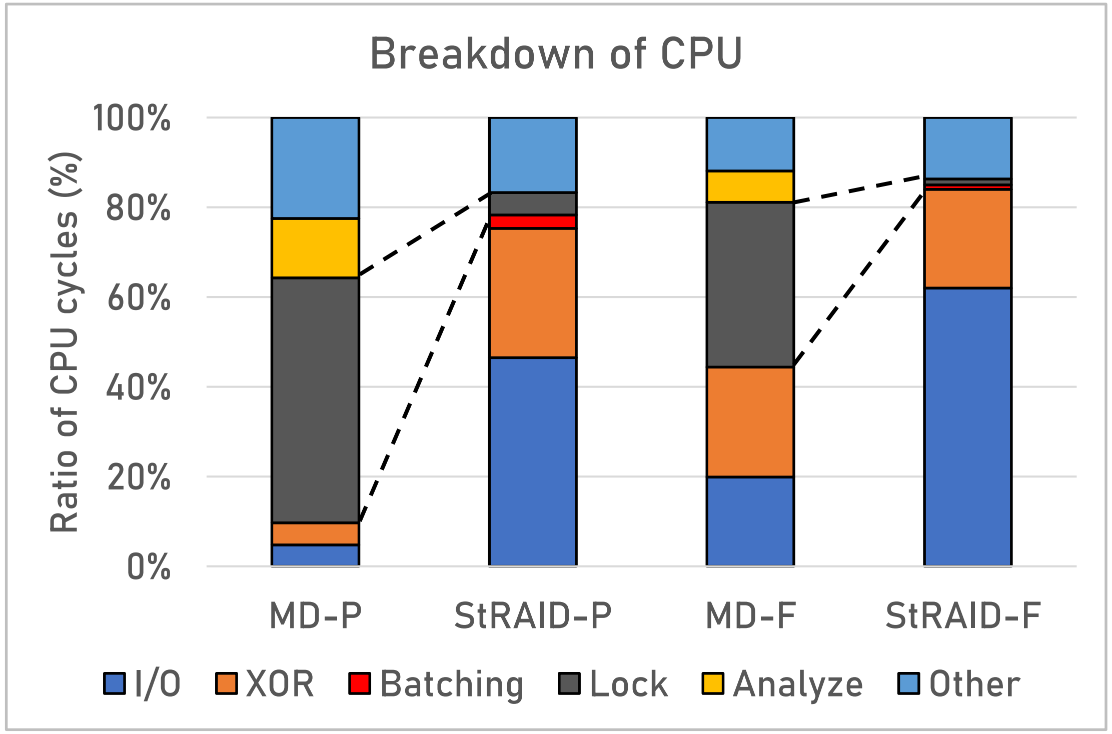
- StRAID archives 2.4x - 3.1x higher write throughput than MD
- StRAID reduces 76% - 90% average write latency from MD

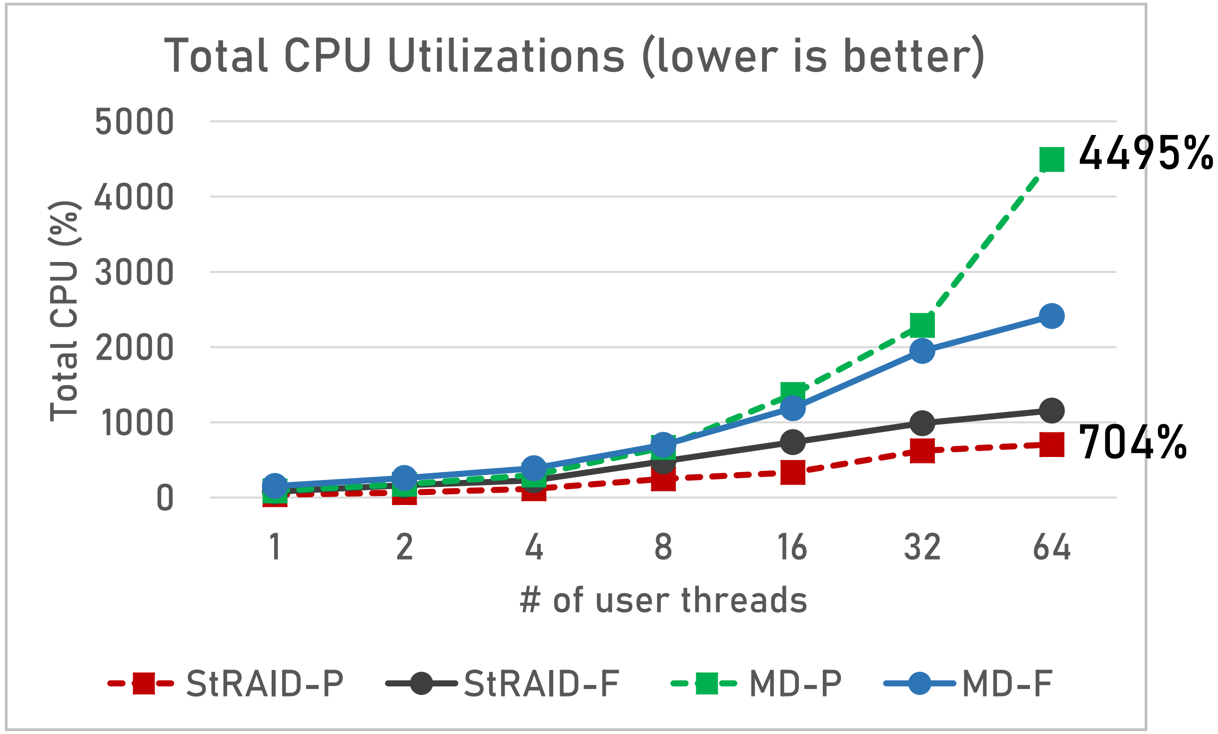




Breakdown of CPU cycles

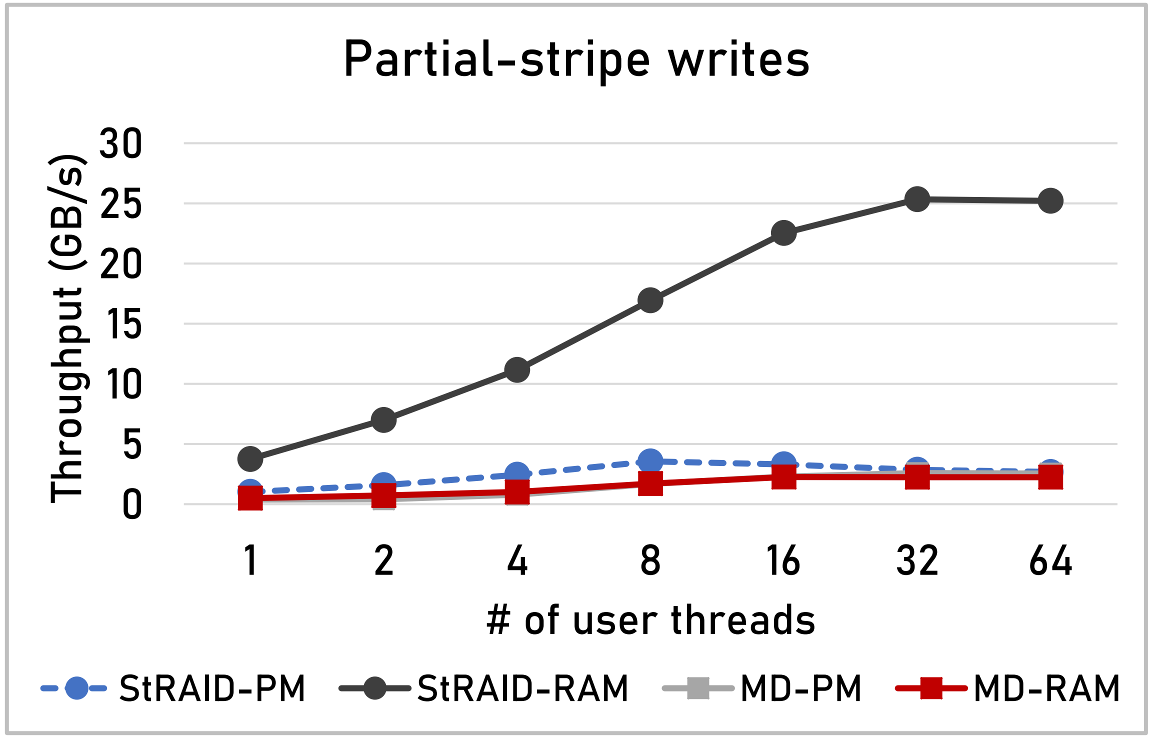
- StRAID reduces up to 90% lock overhead
  - < 5% CPU overhead on the two-phase submission
- The total CPU utilization of StRAID is up to 6.3x lower than MD

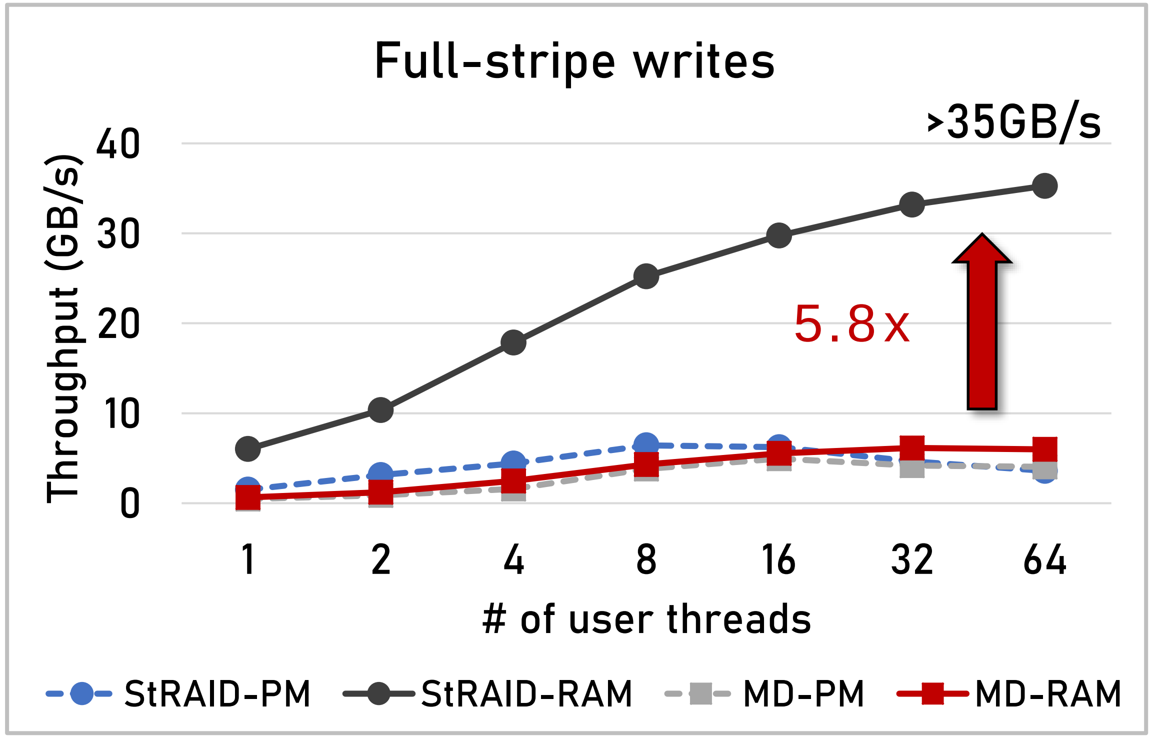




Upper bound Evaluation

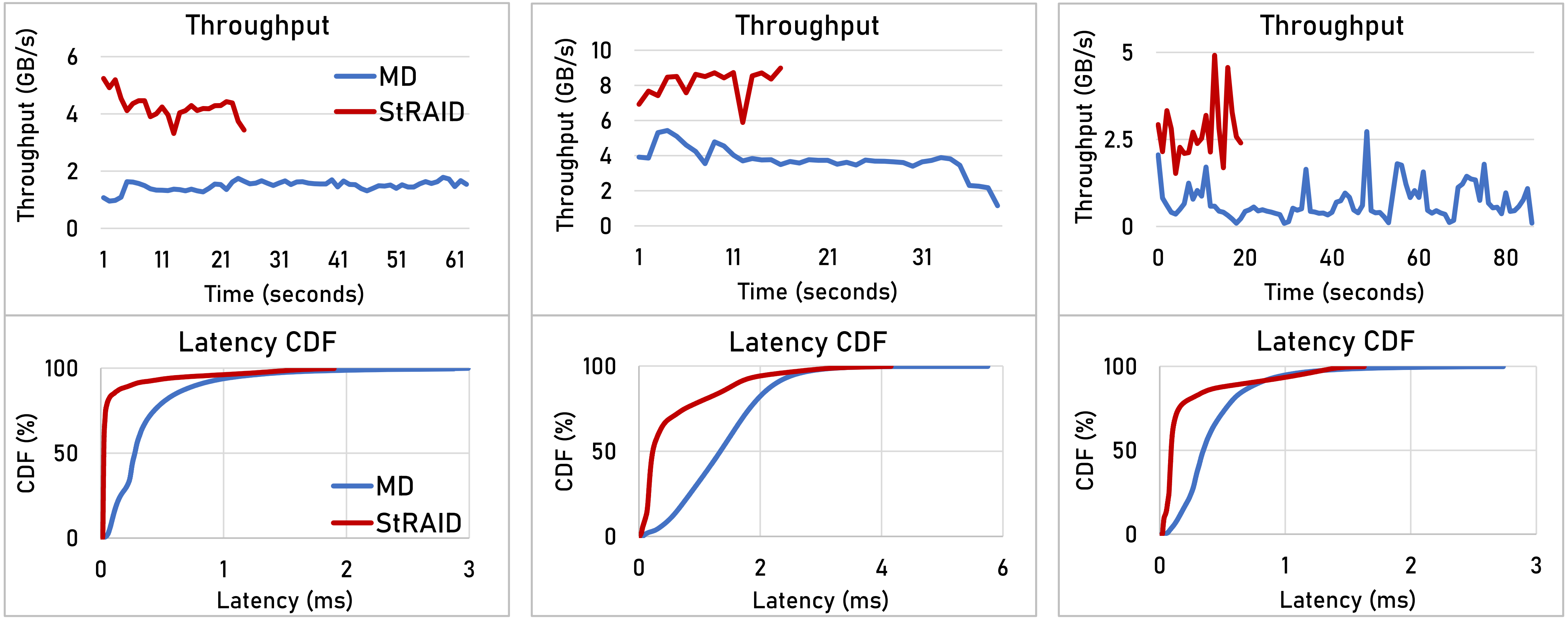
- Run StRAID over six ramdisks (**-RAM*) and Intel Optane PMs (**-PM*)
- StRAID on RAMs archives up to 5.8x higher throughput than MD





Macro-benchmark Results

- Average throughput: 2x – 2.8x higher than MD
- Mean, average, and 99th-percentile latency: 10.3x, 49%, and 25% lower than MD



Ali-Pangu **Filebench (fileserver)** **Microsoft (prn0)**

Conclusion

- StRAID: a new architecture for parity-based RAID on SSDs
  - ➢ Stripe-threaded architecture to efficiently parallelize stripe-write tasks
  - ➢ Two-phase stripe submission to address partial-stripe-write penalty
  - ➢ Performs significantly better than existing Linux MD

- See paper for more details

- Source code: https://github.com/wsczq1/straid

Thanks