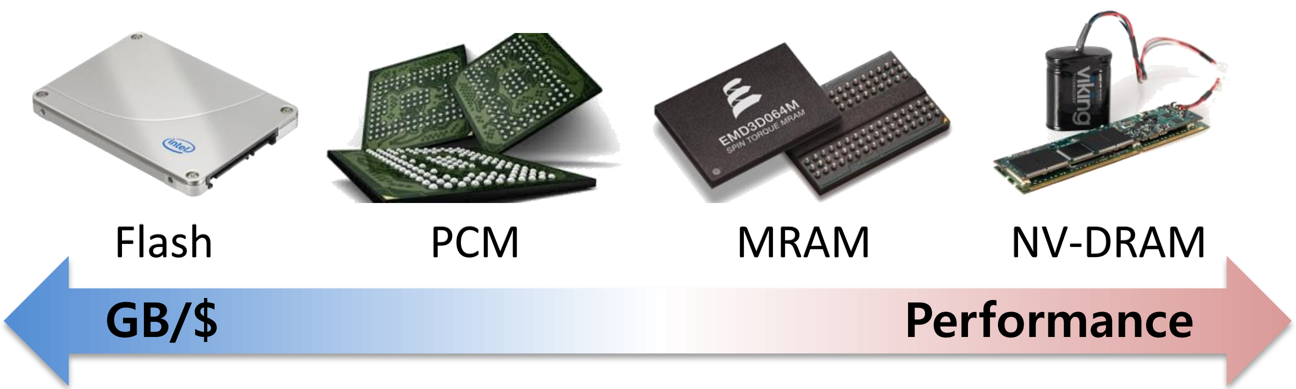
Request-Oriented Durable Write Caching for Application Performance

Sangwook Kim[1], Hwanju Kim[2], Sang-Hoon Kim[3], Joonwon Lee[1], and Jinkyu Jeong[1]

Sungkyunkwan University[1]
University of Cambridge[2]
Korea Advanced Institute of Science and Technology (KAIST)[3]

Non-volatile Write Cache

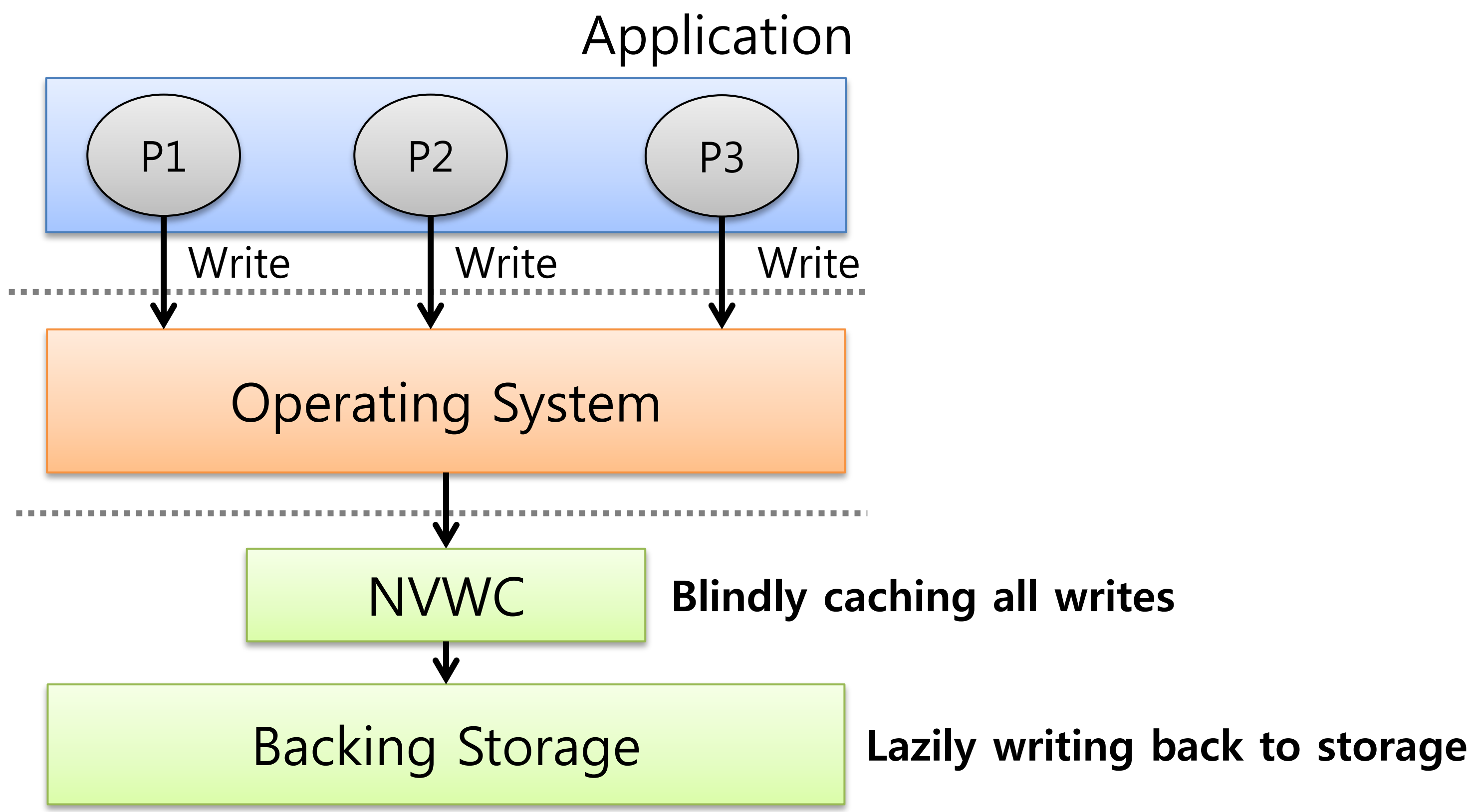
- **Volatile** DRAM cache is ineffective for write
  - Writes are dominant I/Os [FAST'09, FAST'10, FAST'14]
- Non-volatile write cache (NVWC) provides
  - **Fast response** for write w/o loss of durability
- NVWC candidates



[Bhadkamkar *et al.*, FAST'09] BORG: Block-reORGanization for self-optimizing storage systems
[Koller *et al.*, FAST'10] I/O deduplication: Utilizing content similarity to improve I/O performance
[Harter *et al.*, FAST'14] Analysis of HDFS under HBase: a Facebook messages case study

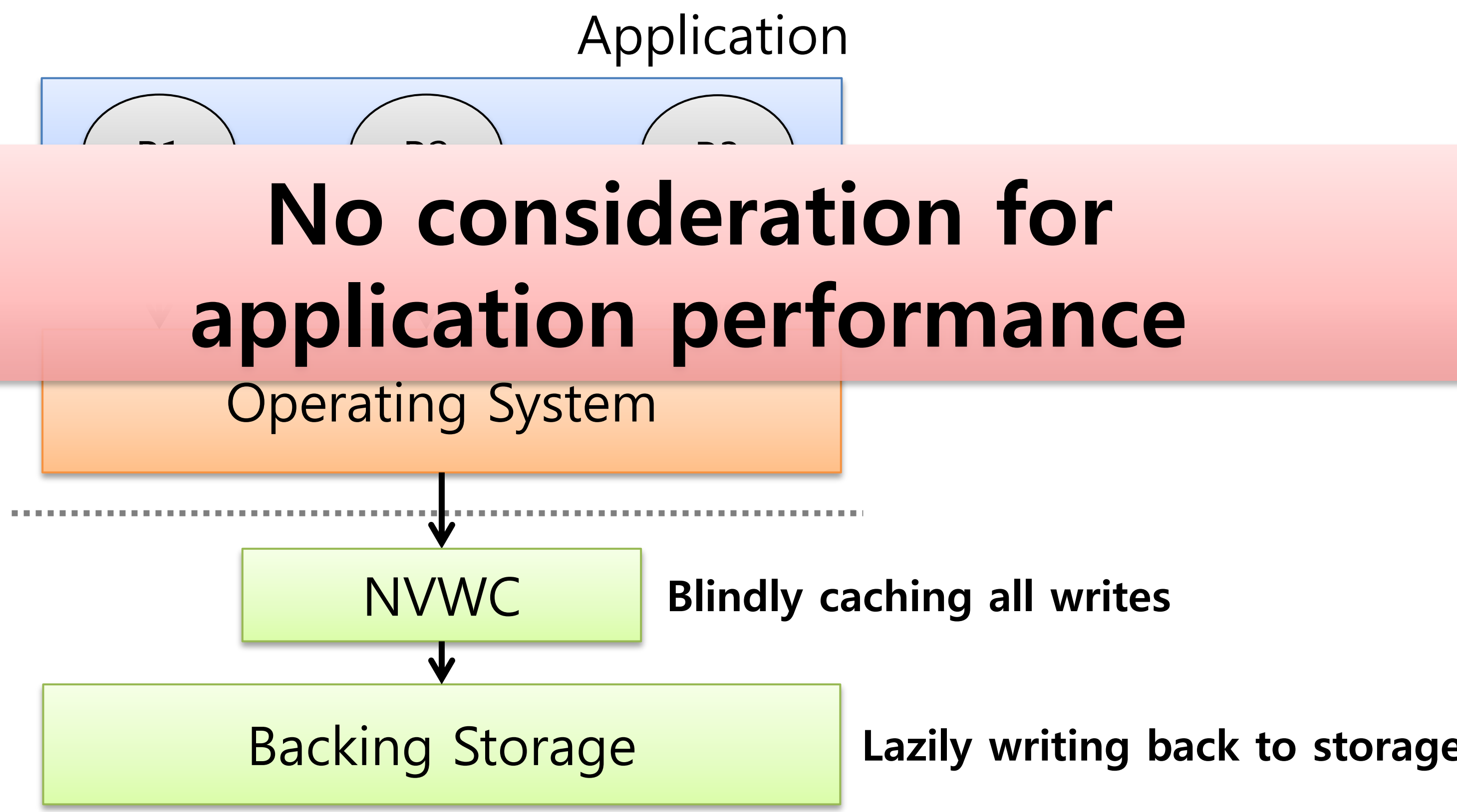
Non-volatile Write Cache

- Simple caching policy



Non-volatile Write Cache

- Simple caching policy



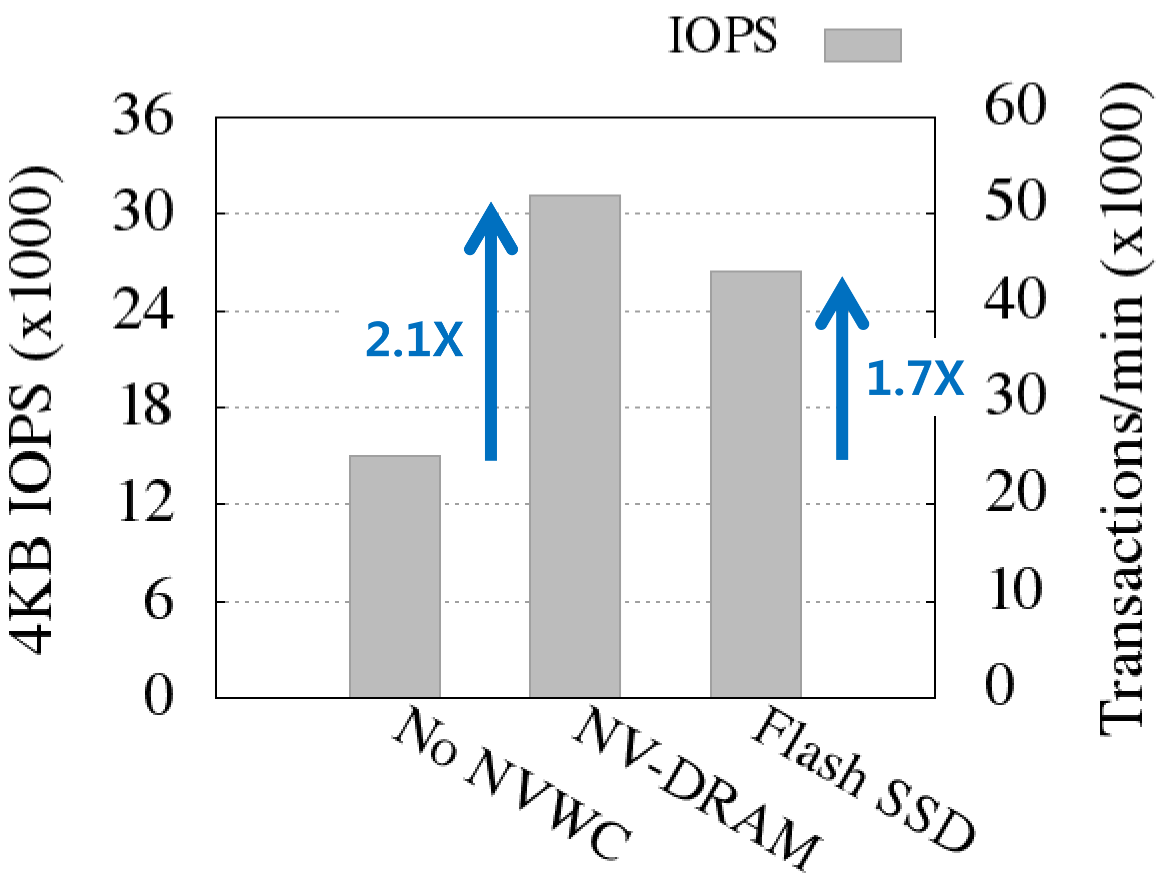
No consideration for application performance

Impact on Application Performance

- Illustrative experiment
  - TPC-C workload
    - PostgreSQL database
  - 2 NVWC devices
    - 32MB NV-DRAM (emulated via ramdisk)
    - 4GB flash SSD

Impact on Application Performance

- Experimental result

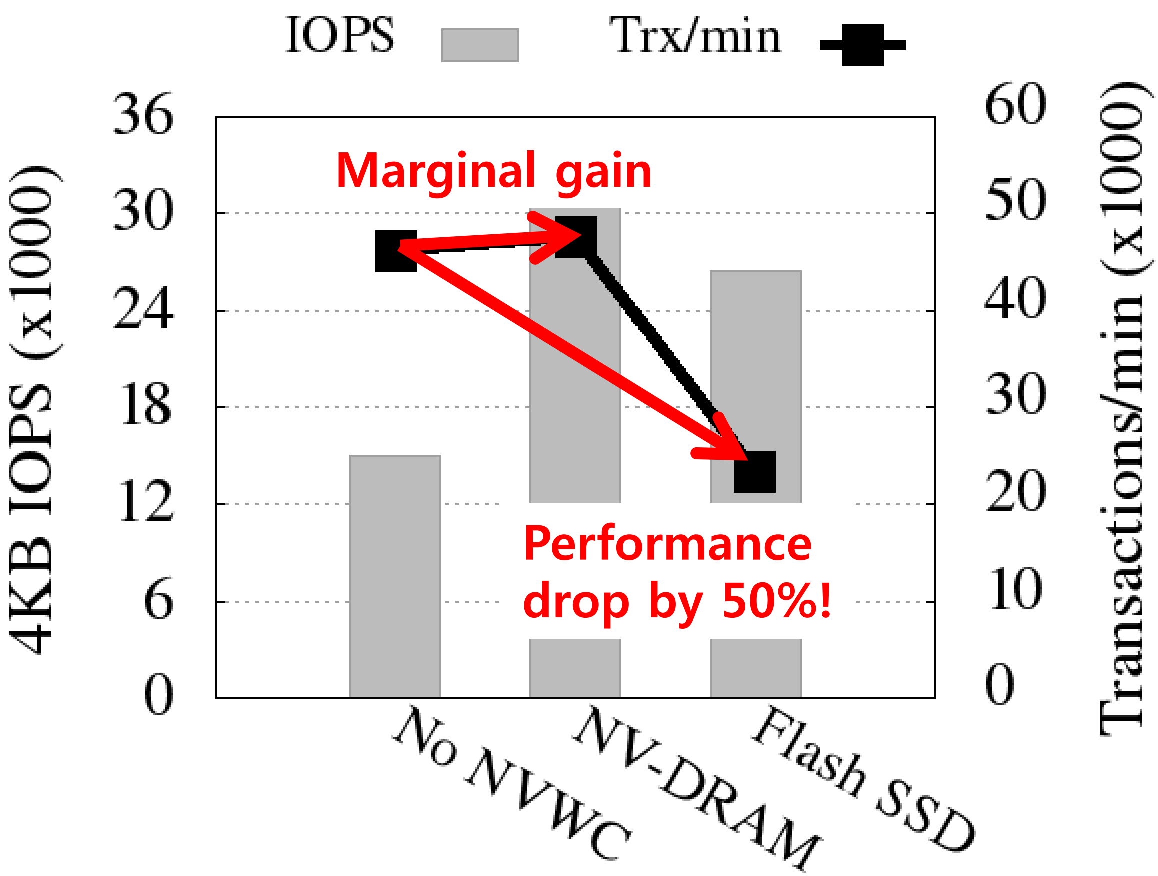


* System perf.

- **~ 2.1X** improved

Impact on Application Performance

- Experimental result



* System perf.

- ~ **2.1X** improved

* Application perf.

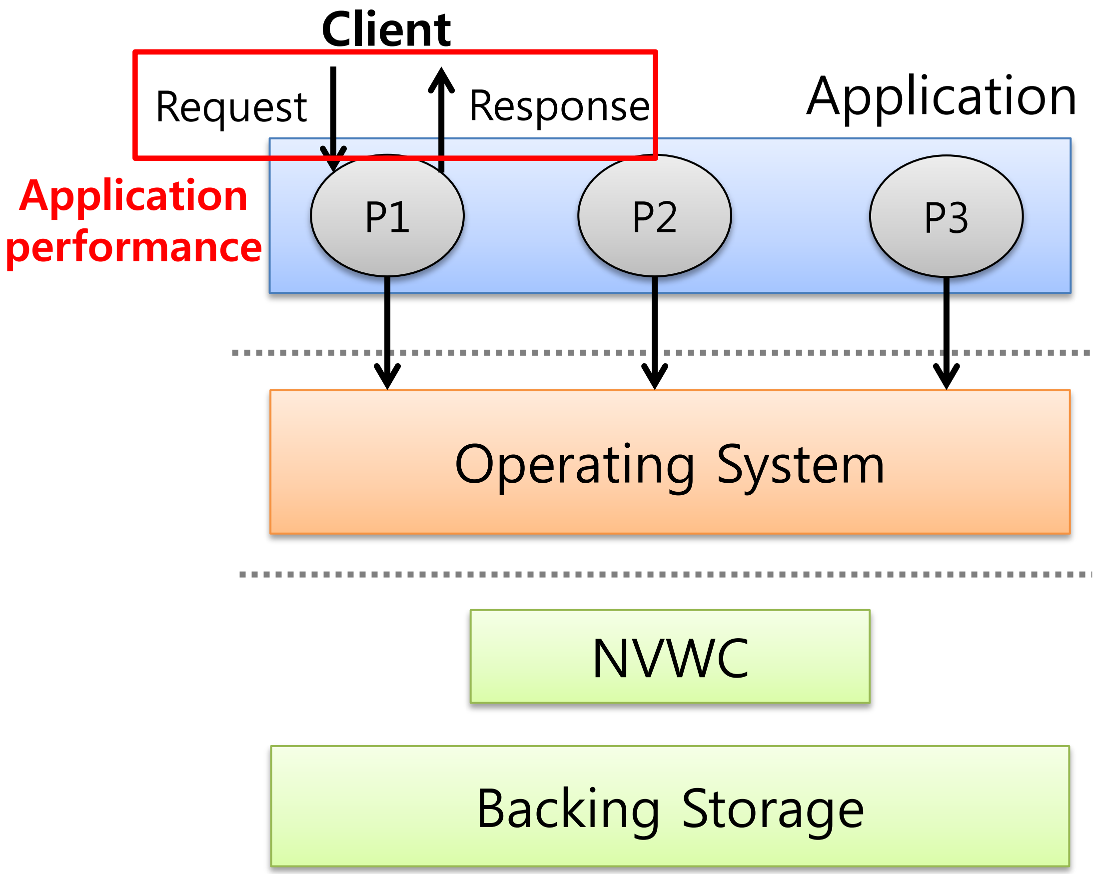
- ~ **50%** degraded

What's the Problem?

- **Criticality-agnostic** contention

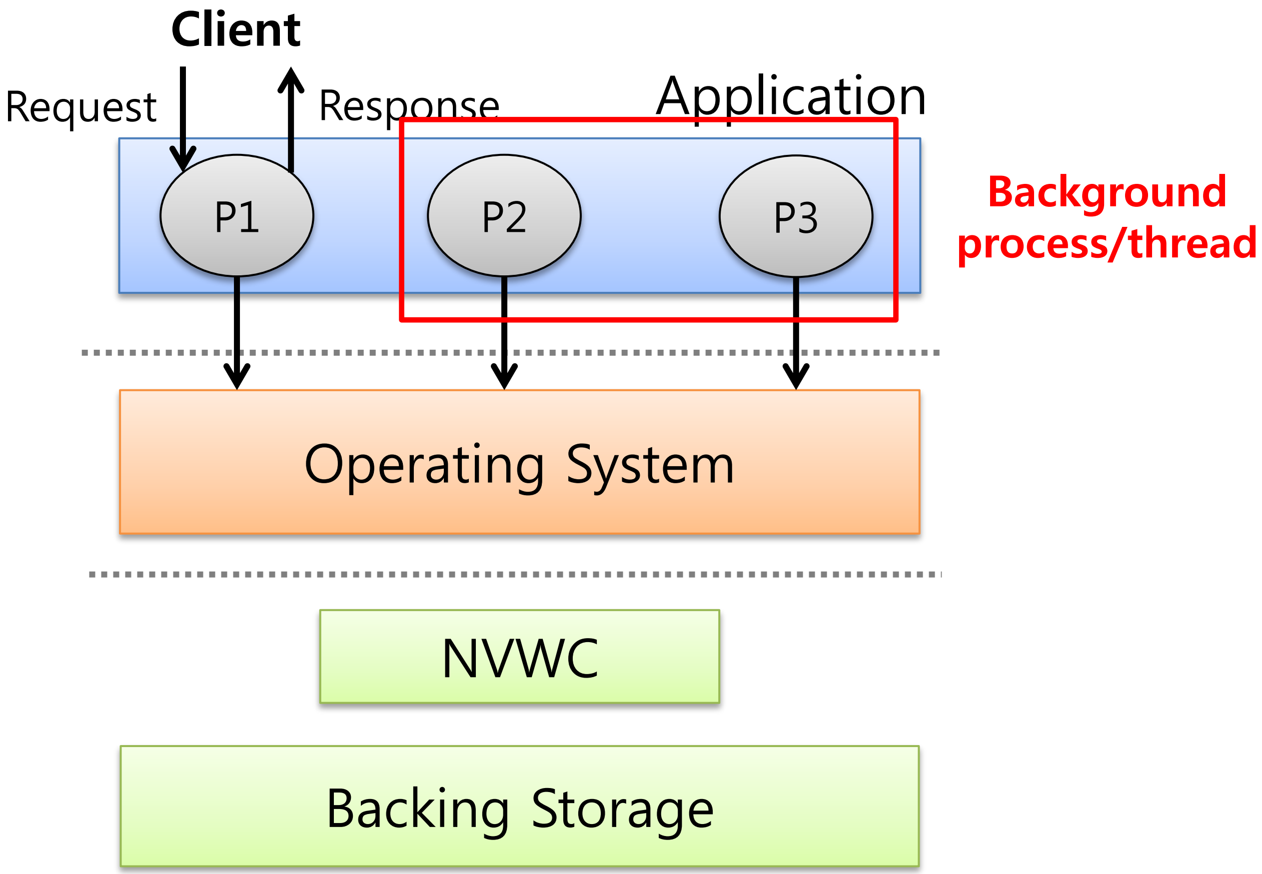
Criticality-Agnostic Contention

- Different write criticality



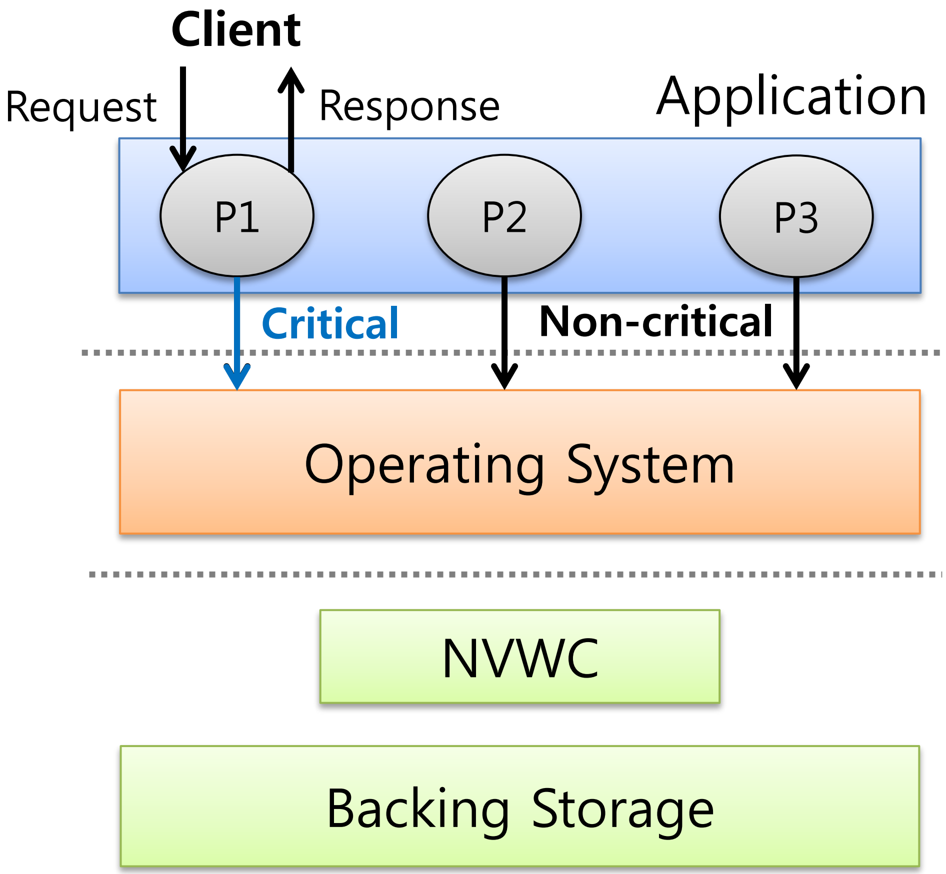
Criticality-Agnostic Contention

- Different write criticality



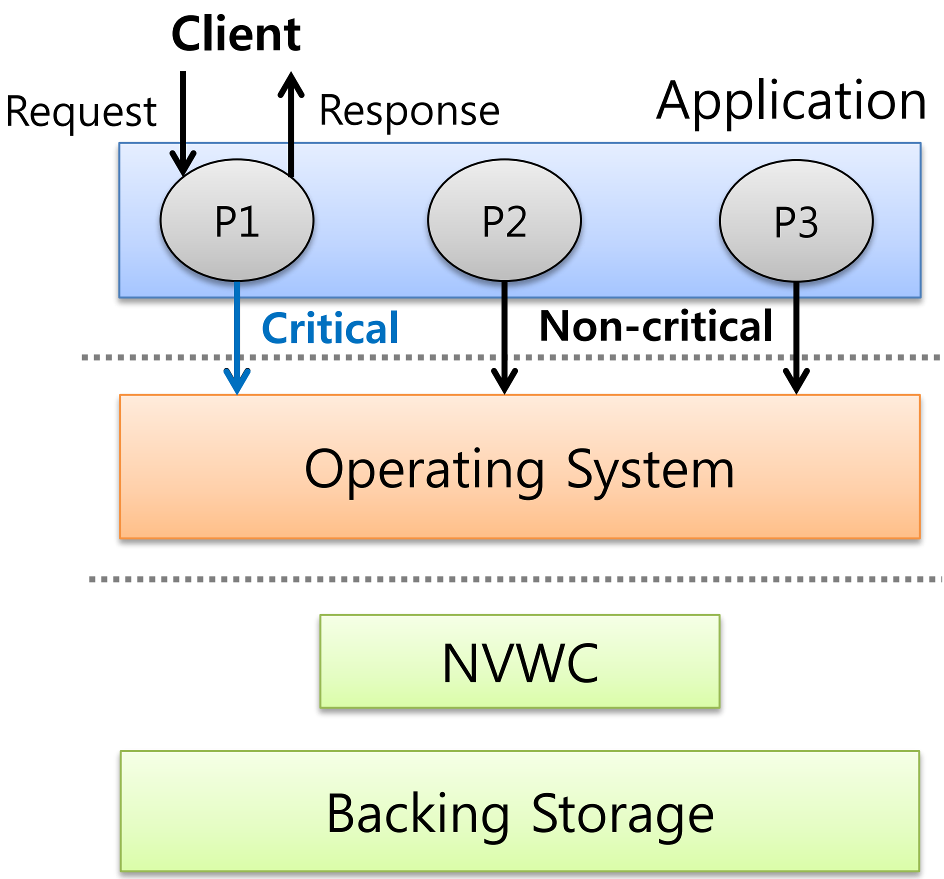
Criticality-Agnostic Contention

- Different write criticality



Criticality-Agnostic Contention

- Different write criticality

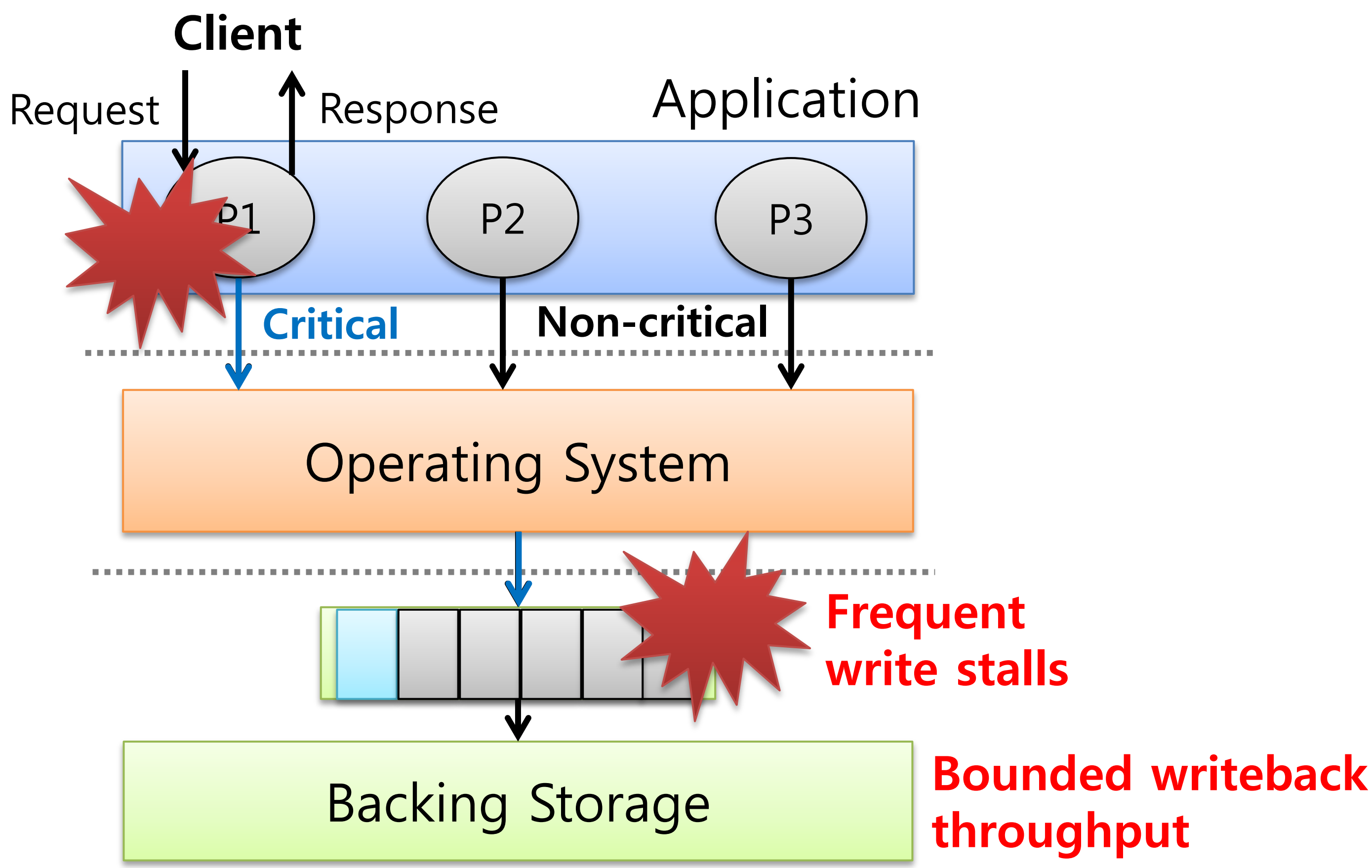


* Contentions

- Capacity contention
- Bandwidth contention

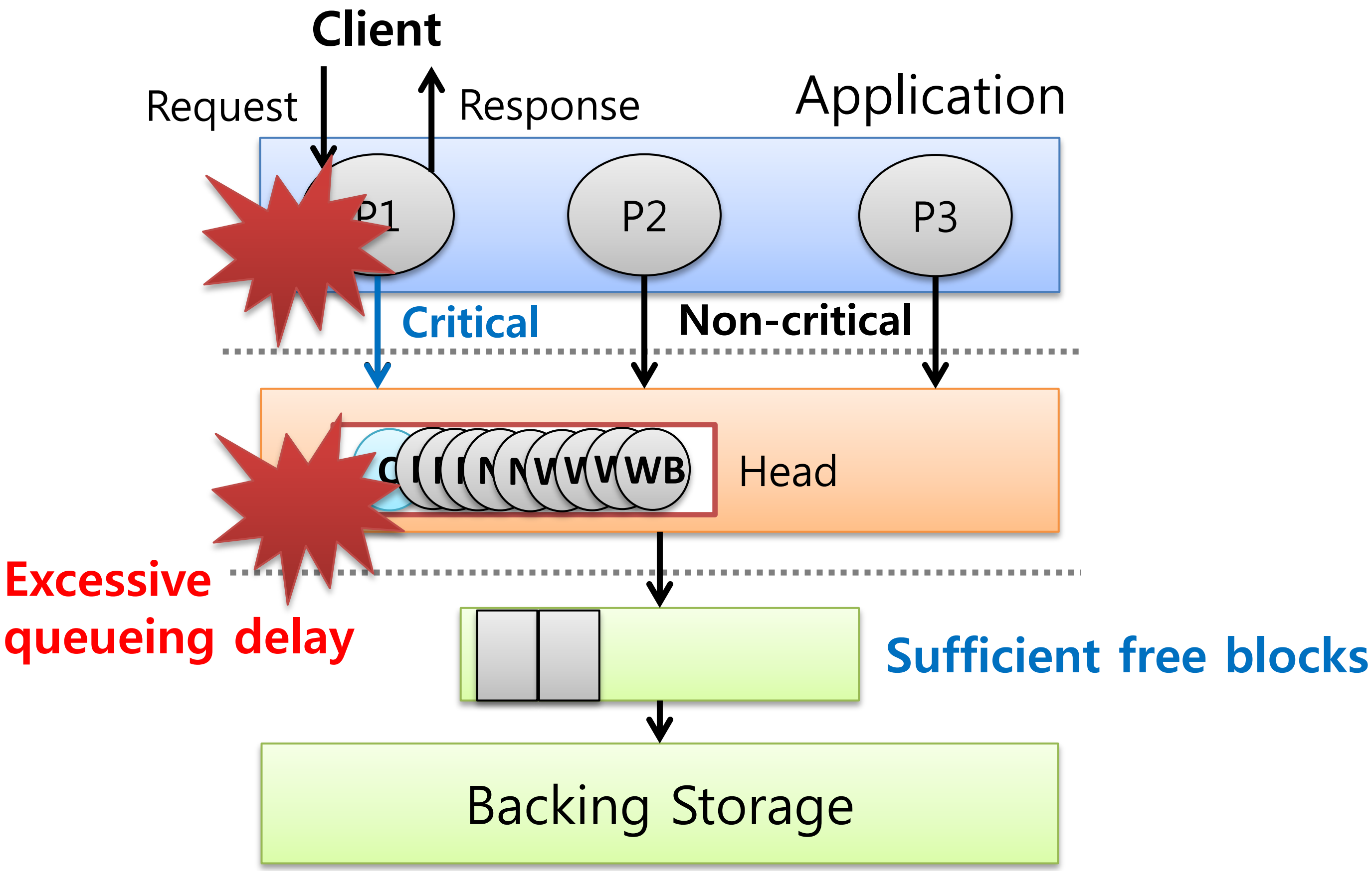
Criticality-Agnostic Contention

- Capacity contention



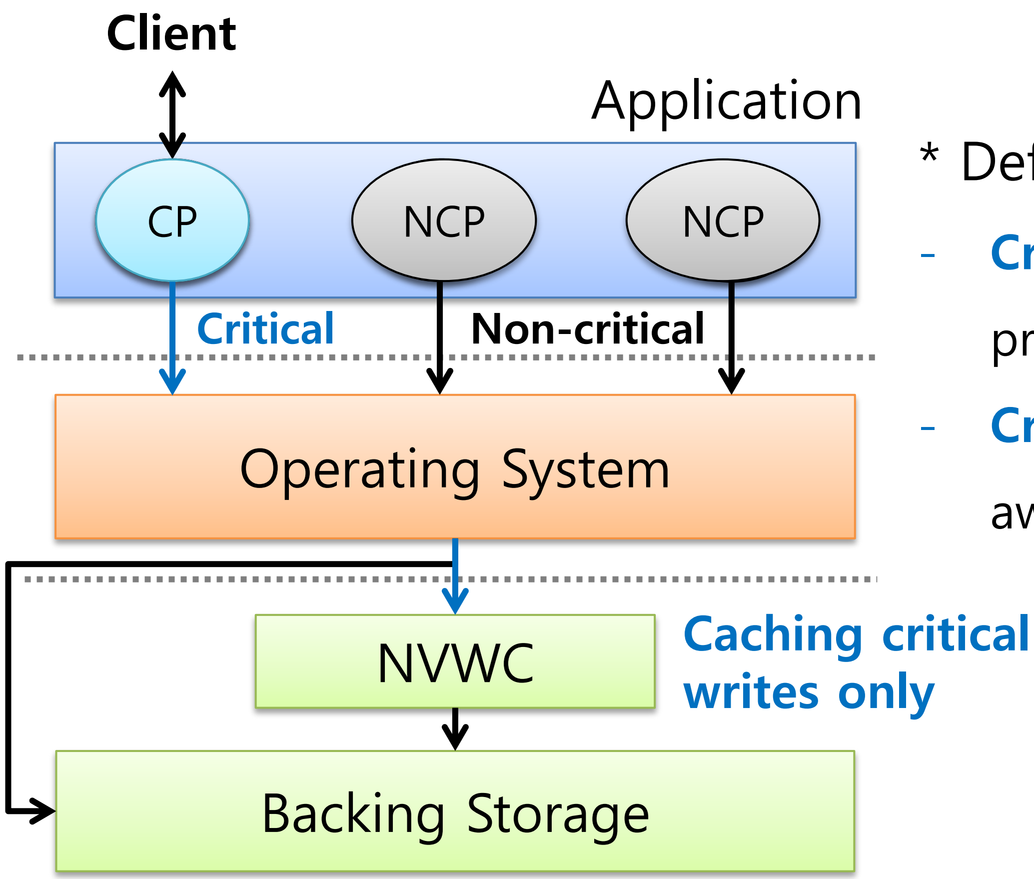
Criticality-Agnostic Contention

- Bandwidth contention



Our Approach

- Request-oriented caching policy



* Definitions

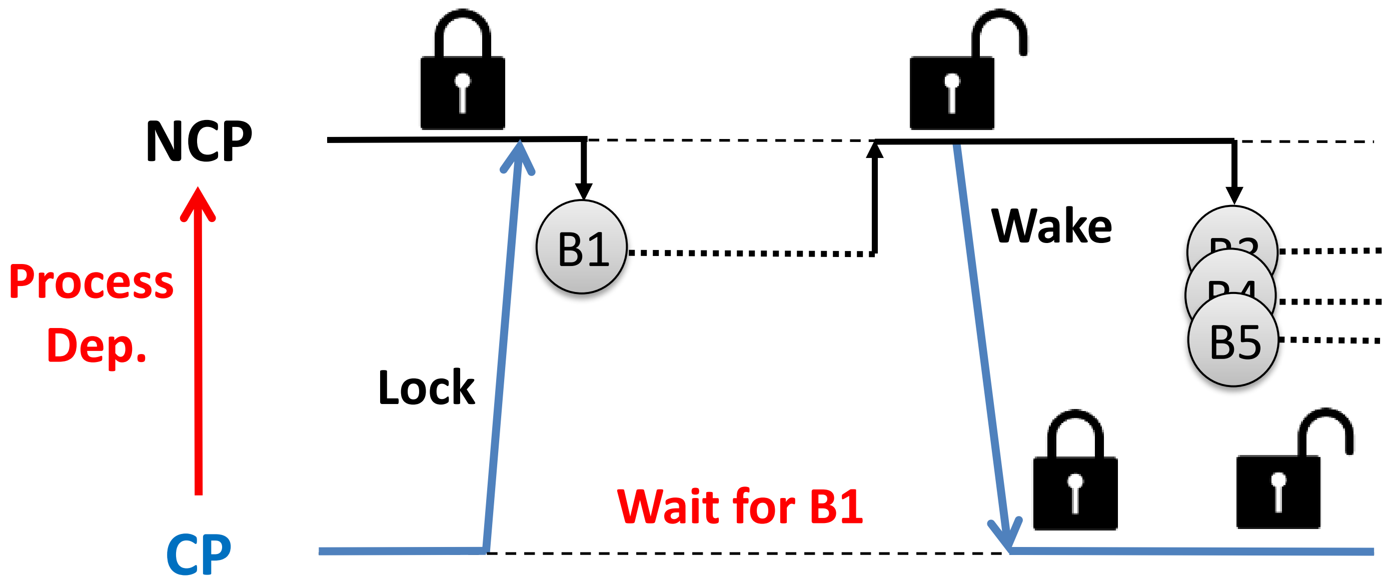
- **Critical process (CP)**: a process handling request
- **Critical write**: a write awaited by a critical proc.

Challenge

- How to accurately detect critical writes
- Types of critical write
  - Sync. writes from critical processes
  - **Dependency-induced** critical writes
    - Process dependency-induced
    - I/O dependency-induced

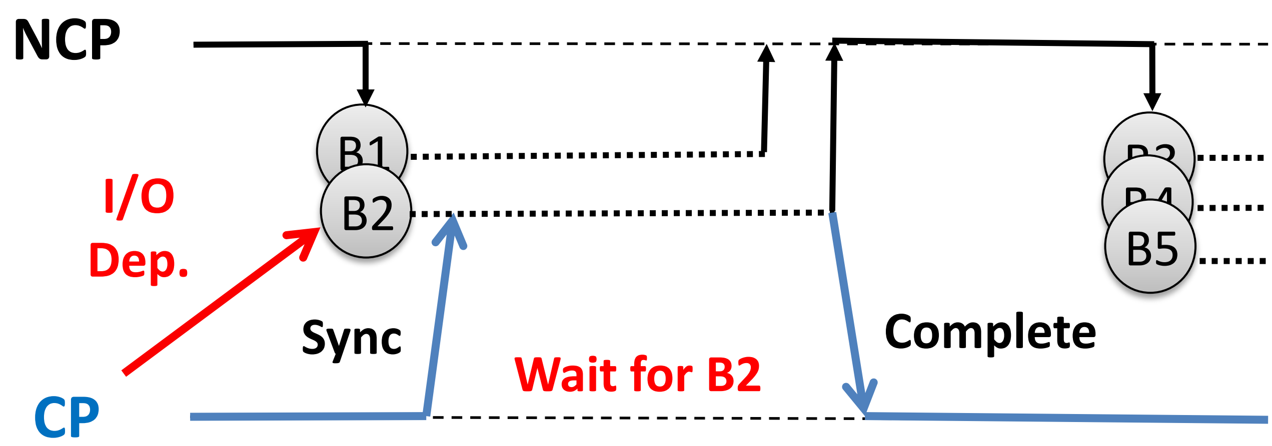
Dependency Problem

- Process dependency



Dependency Problem

- I/O dependency



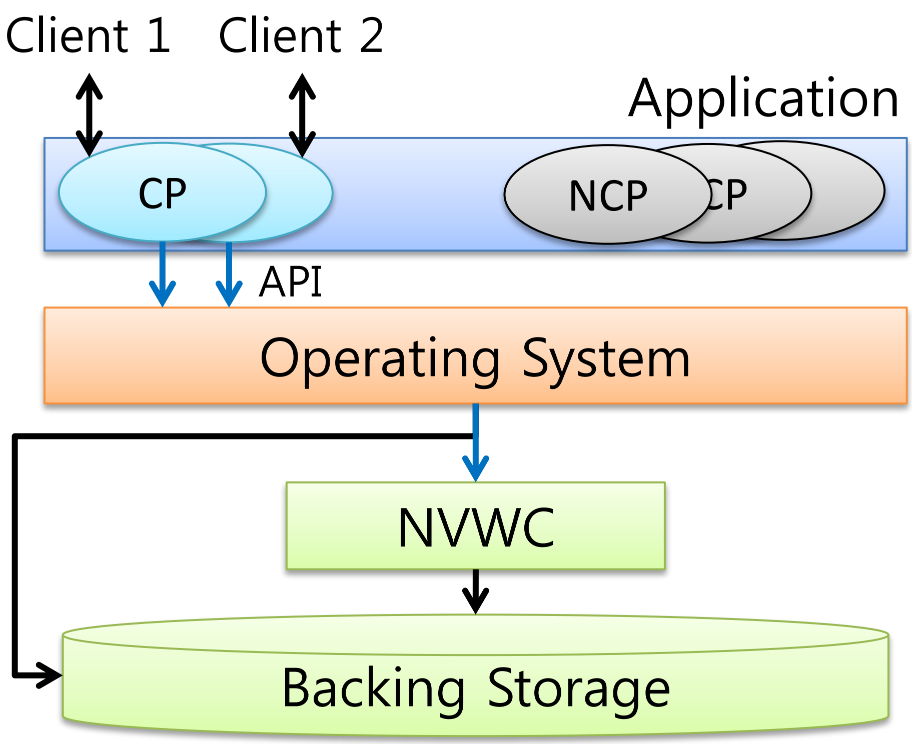
* Example scenarios:
- CP **fsync()** to a block under writeback issued by NCP
- CP tries to **overwrite** fs journal buffer under writeback

Critical Write Detection

- Critical process identification
  - Application-guided identification

Critical Process Identification

- Application-guided identification



Critical Write Detection

- Critical process identification
    - Application-guided identification

- **Dependency resolution**
    - **Criticality inheritance protocols**
        - **Process criticality inheritance**
        - **I/O criticality inheritance**
        - **Blocking object tracking**

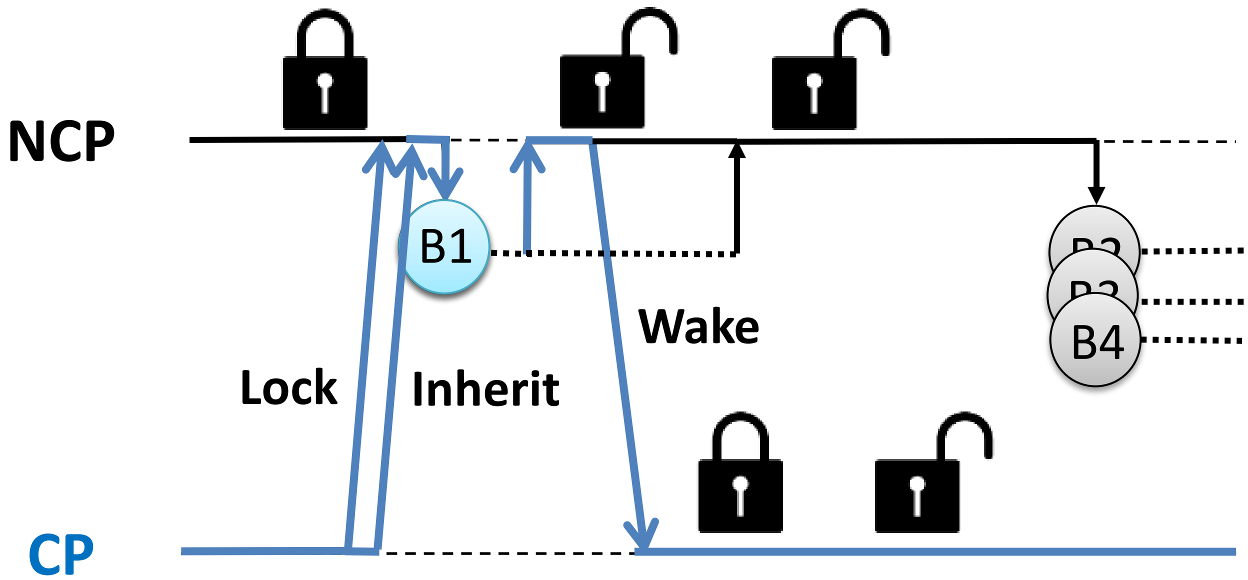
Critical Write Detection

- Critical process identification
  - Application-guided identification

- **Dependency resolution**
  - **Criticality inheritance protocols**
    - **Process criticality inheritance**
    - **I/O criticality inheritance**
    - Blocking object tracking

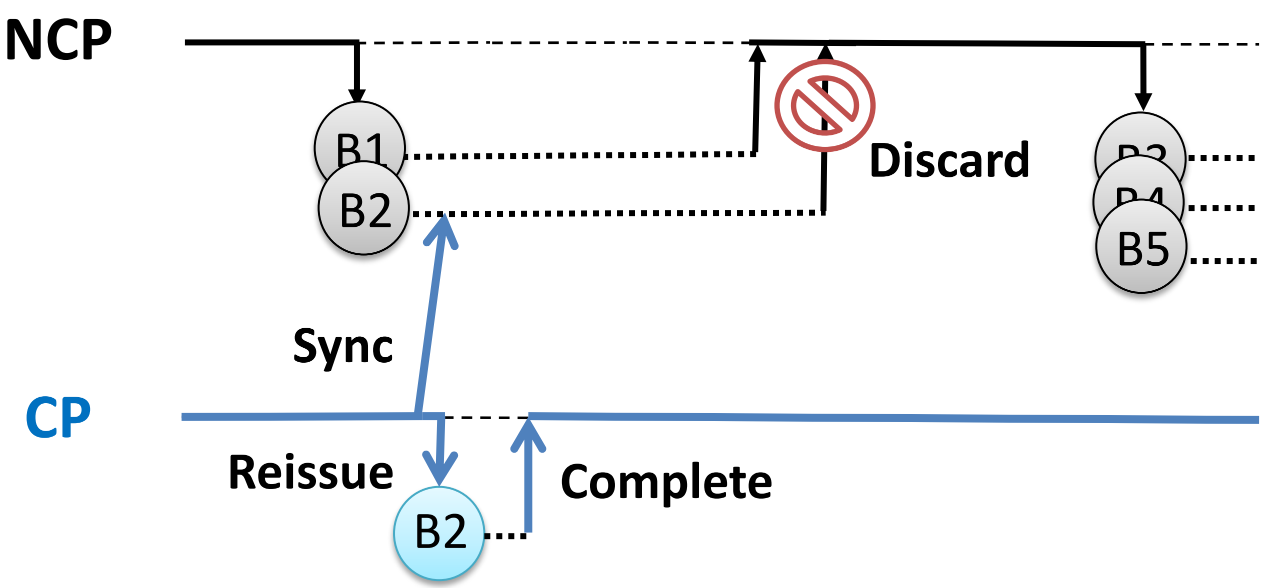
Criticality Inheritance Protocols

- Process criticality inheritance



Criticality Inheritance Protocols

- I/O criticality inheritance

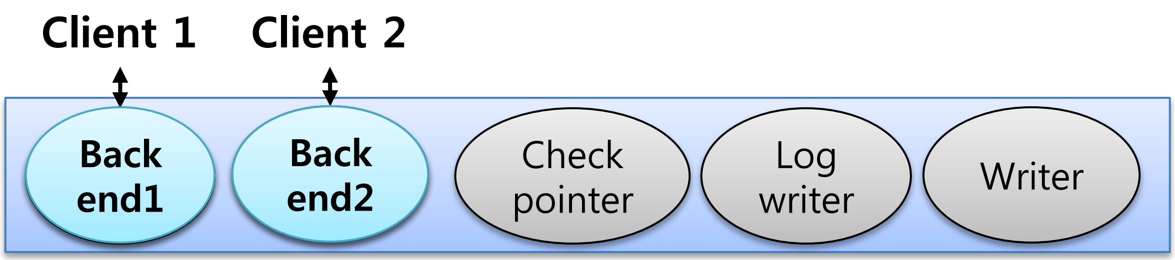


Key issue:
caching the dependent write outstanding to disk w/o side effects

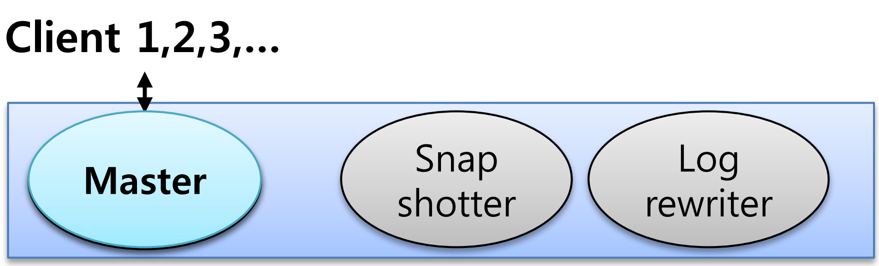
Evaluation

- Implementation on Linux 3.13 w/ FlashCache 3.1

- Application studies
  - PostgreSQL database

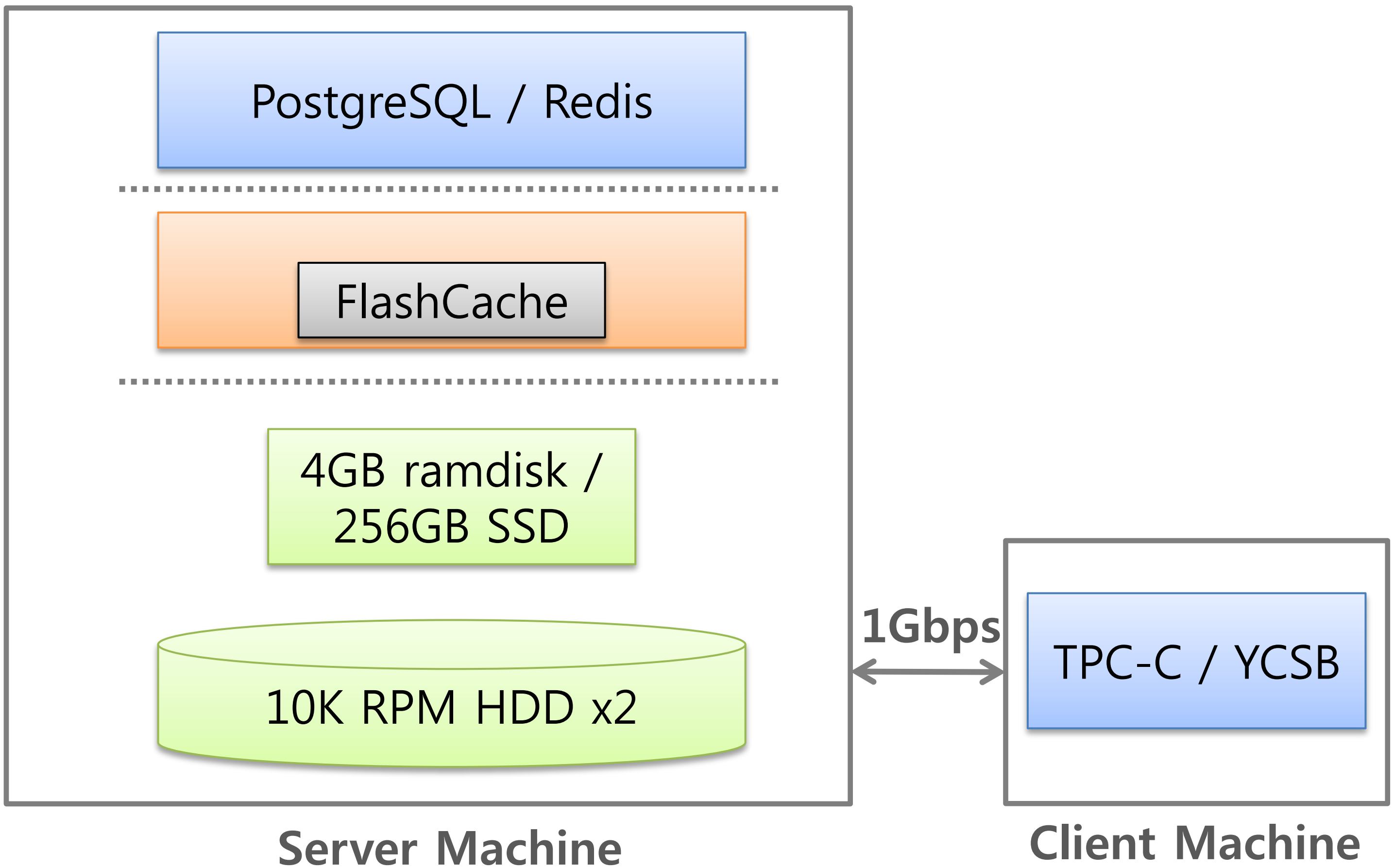


  - Redis key-value store



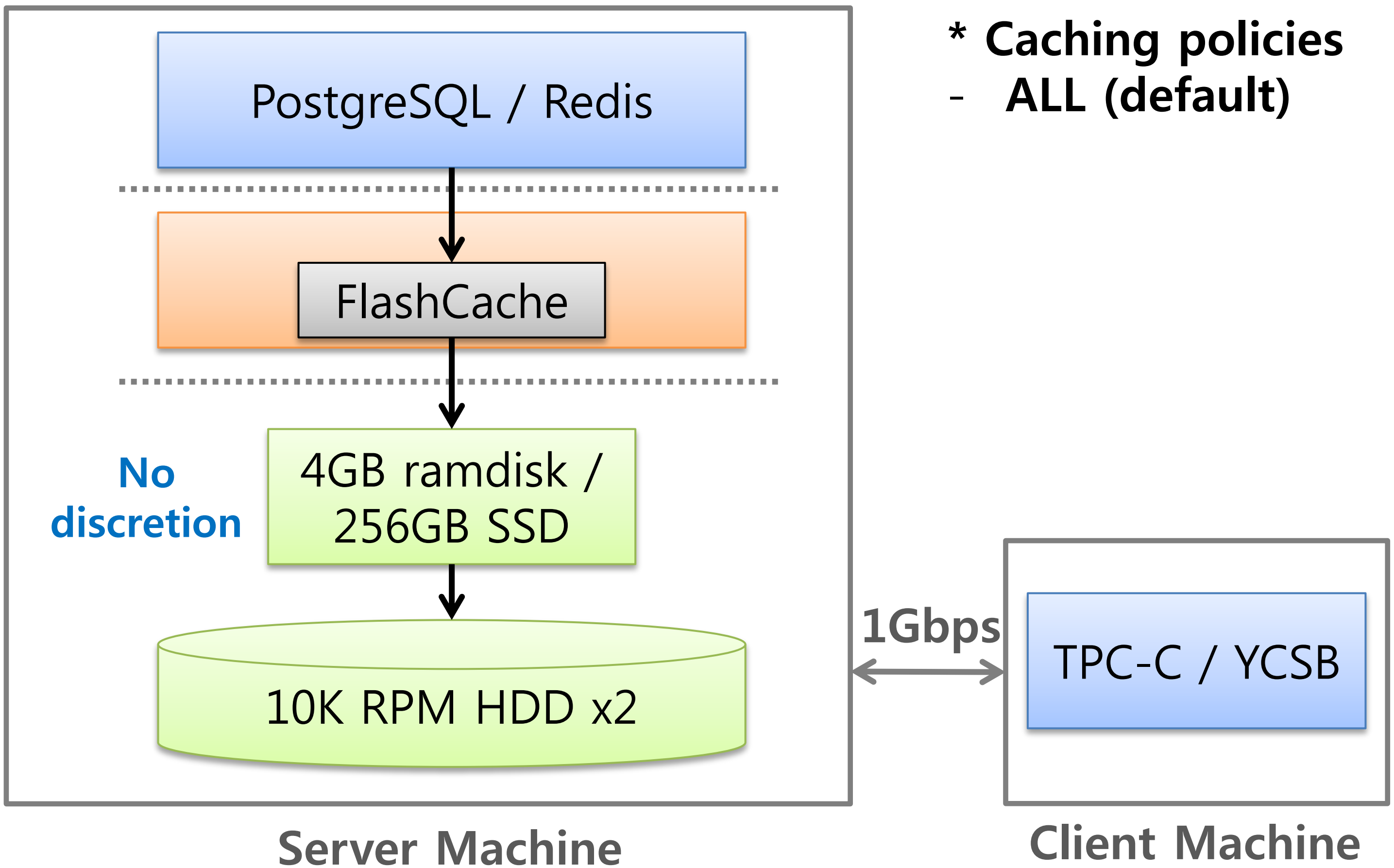
Evaluation

- Experimental setup



Evaluation

- Experimental setup

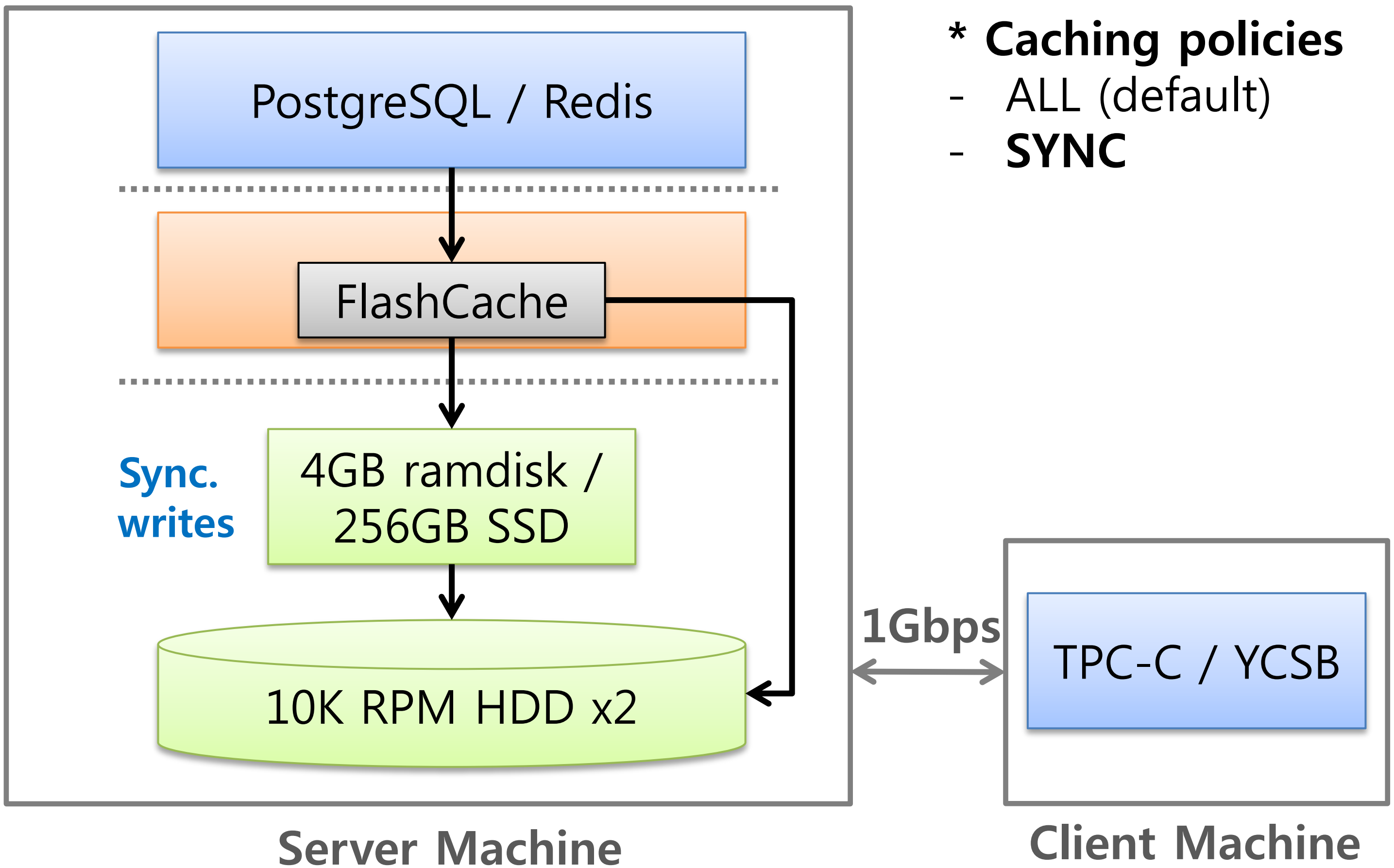


* **Caching policies**

- **ALL (default)**

Evaluation

- Experimental setup

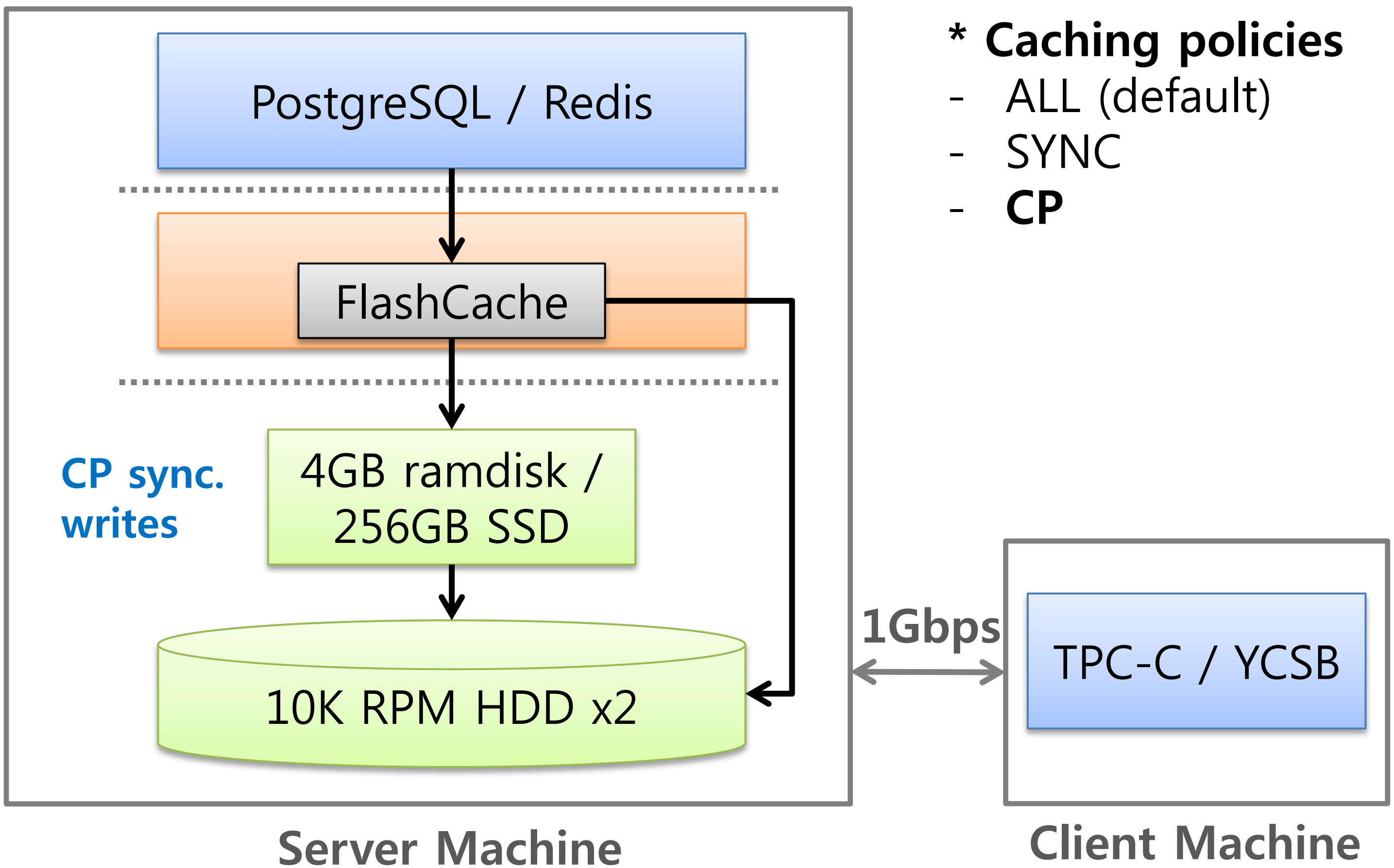


*** Caching policies**

- ALL (default)
- **SYNC**

Evaluation

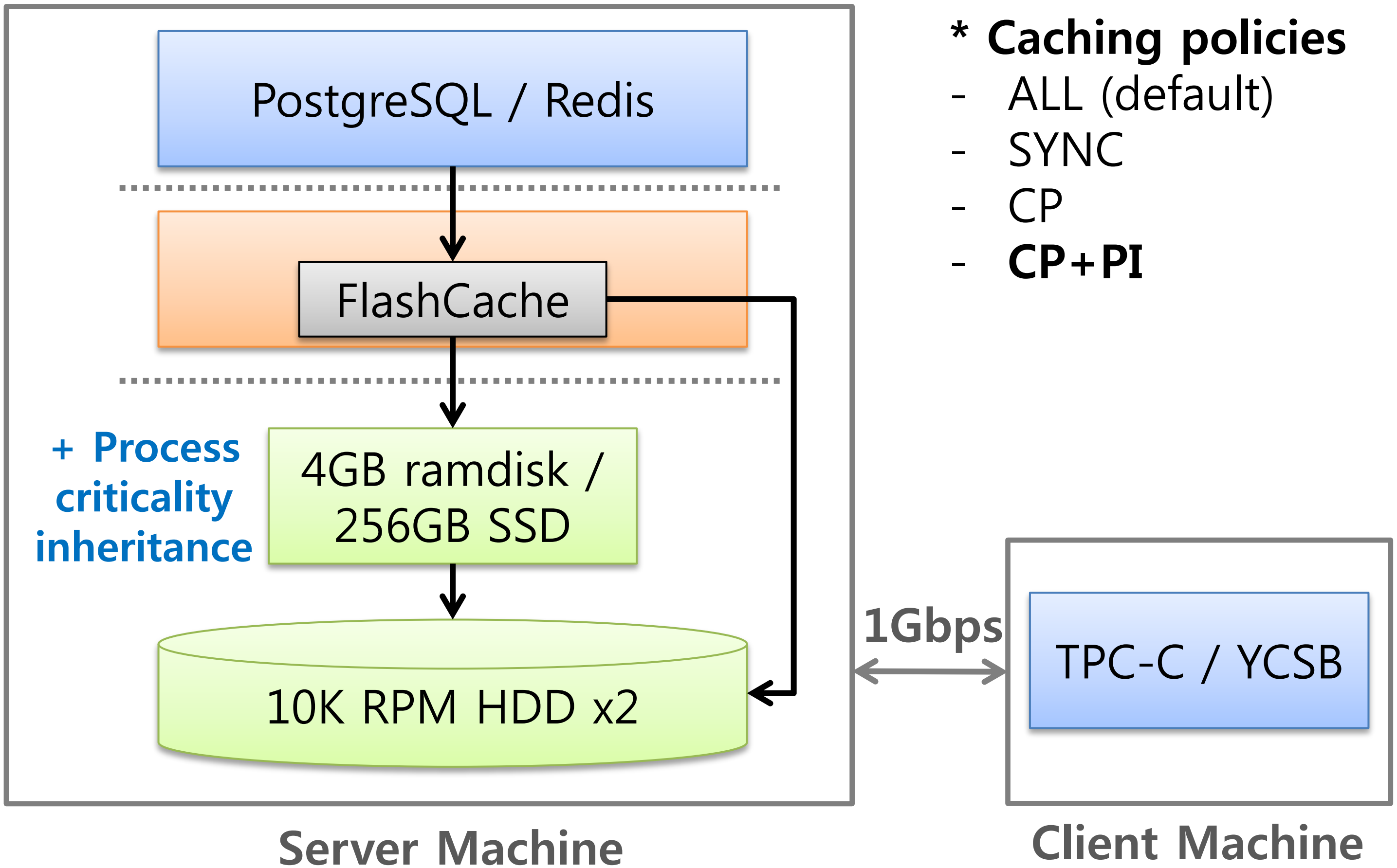
- Experimental setup



*** Caching policies**
- ALL (default)
- SYNC
- **CP**

Evaluation

- Experimental setup

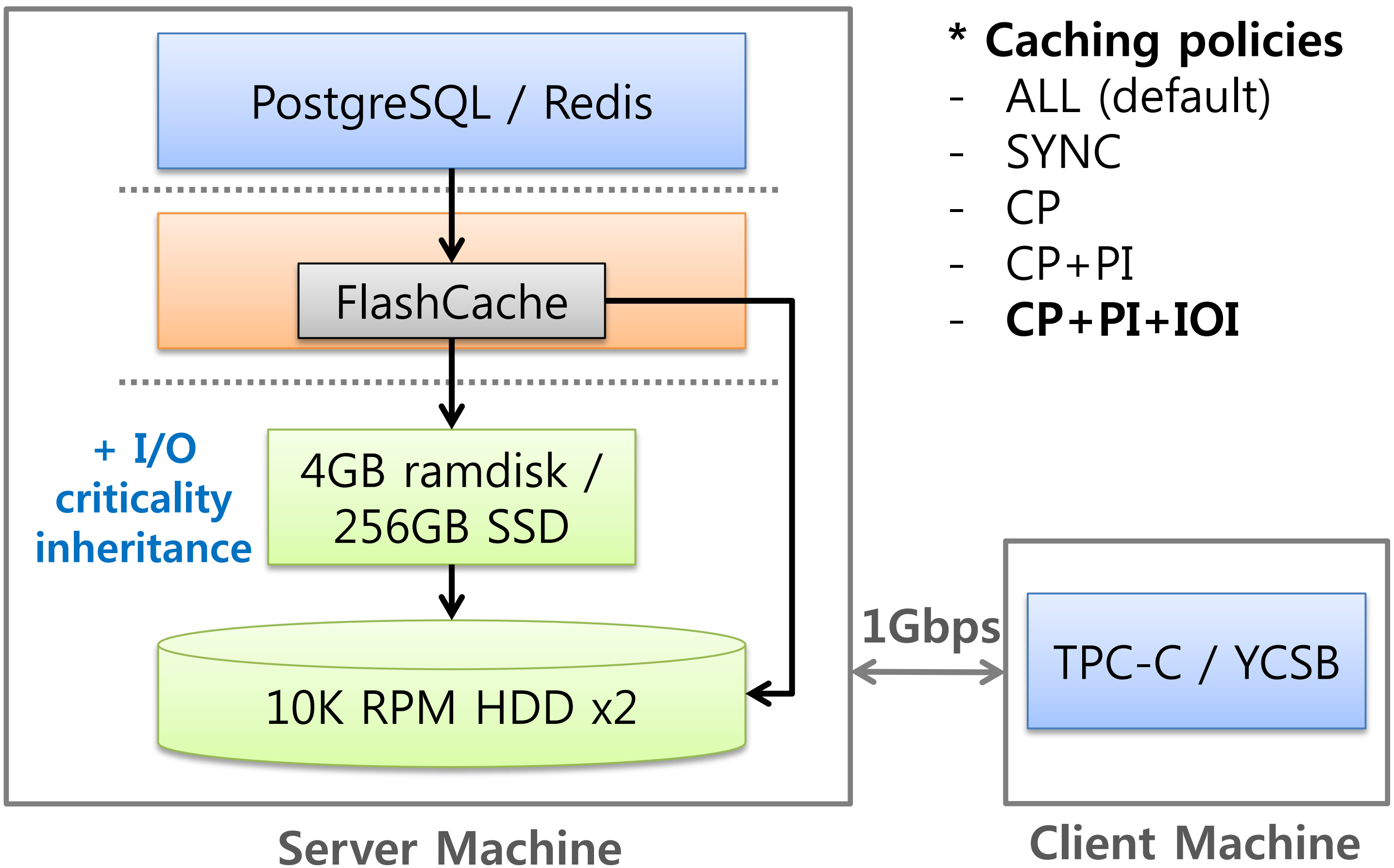


*** Caching policies**

- ALL (default)
- SYNC
- CP
- **CP+PI**

Evaluation

- Experimental setup

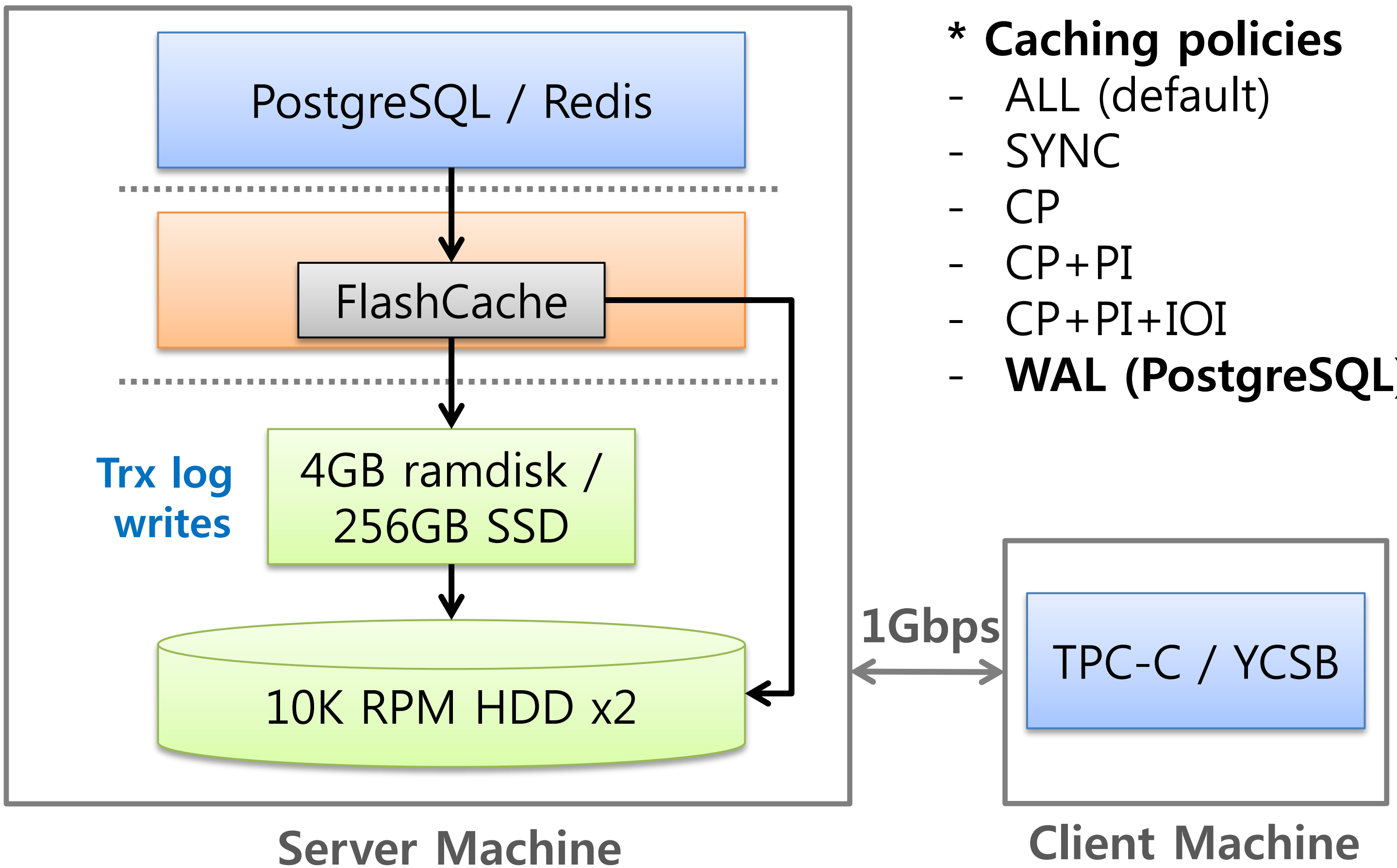


*** Caching policies**

- ALL (default)
- SYNC
- CP
- CP+PI
- **CP+PI+IOI**

Evaluation

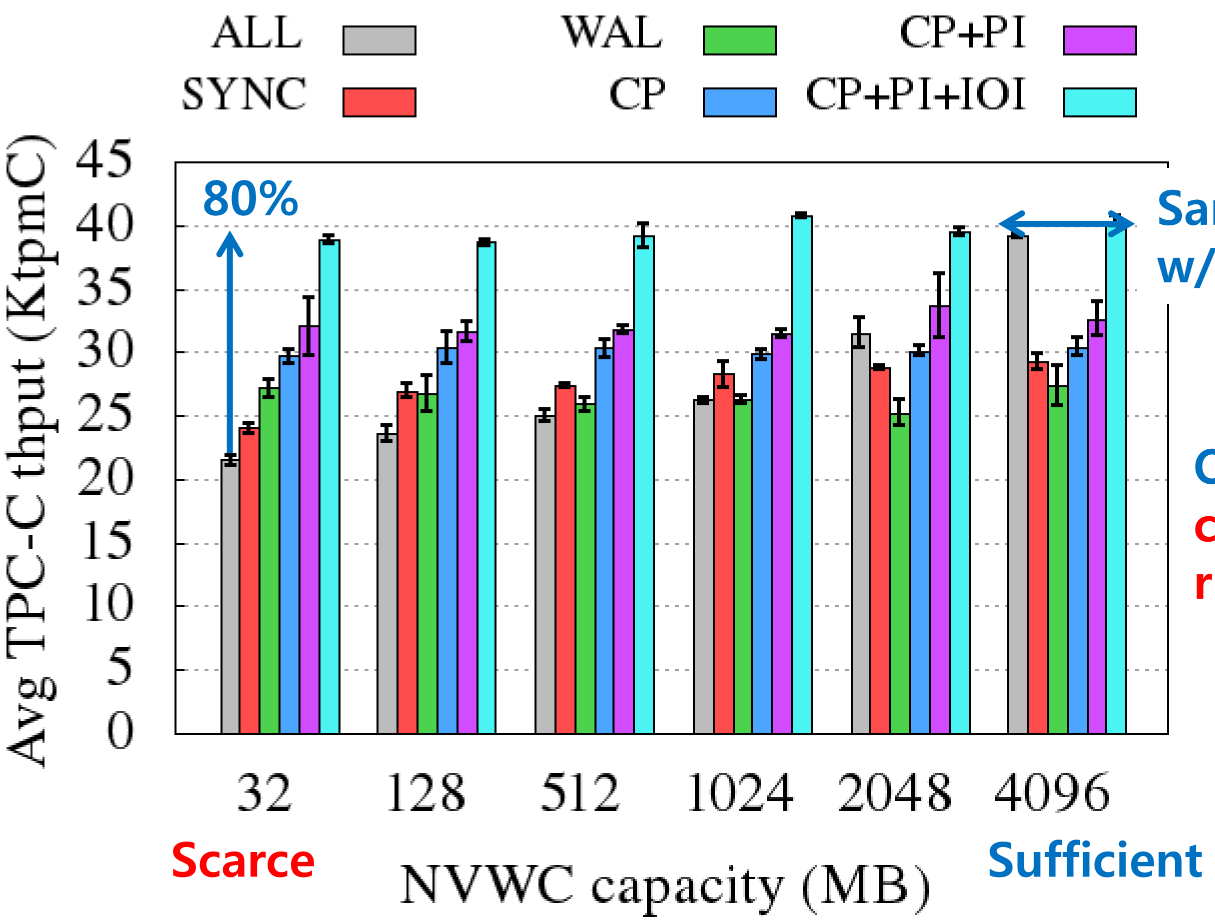
- Experimental setup



*** Caching policies**
- ALL (default)
- SYNC
- CP
- CP+PI
- CP+PI+IOI
- **WAL (PostgreSQL)**

PostgreSQL Performance

- TPC-C workload w/ ramdisk

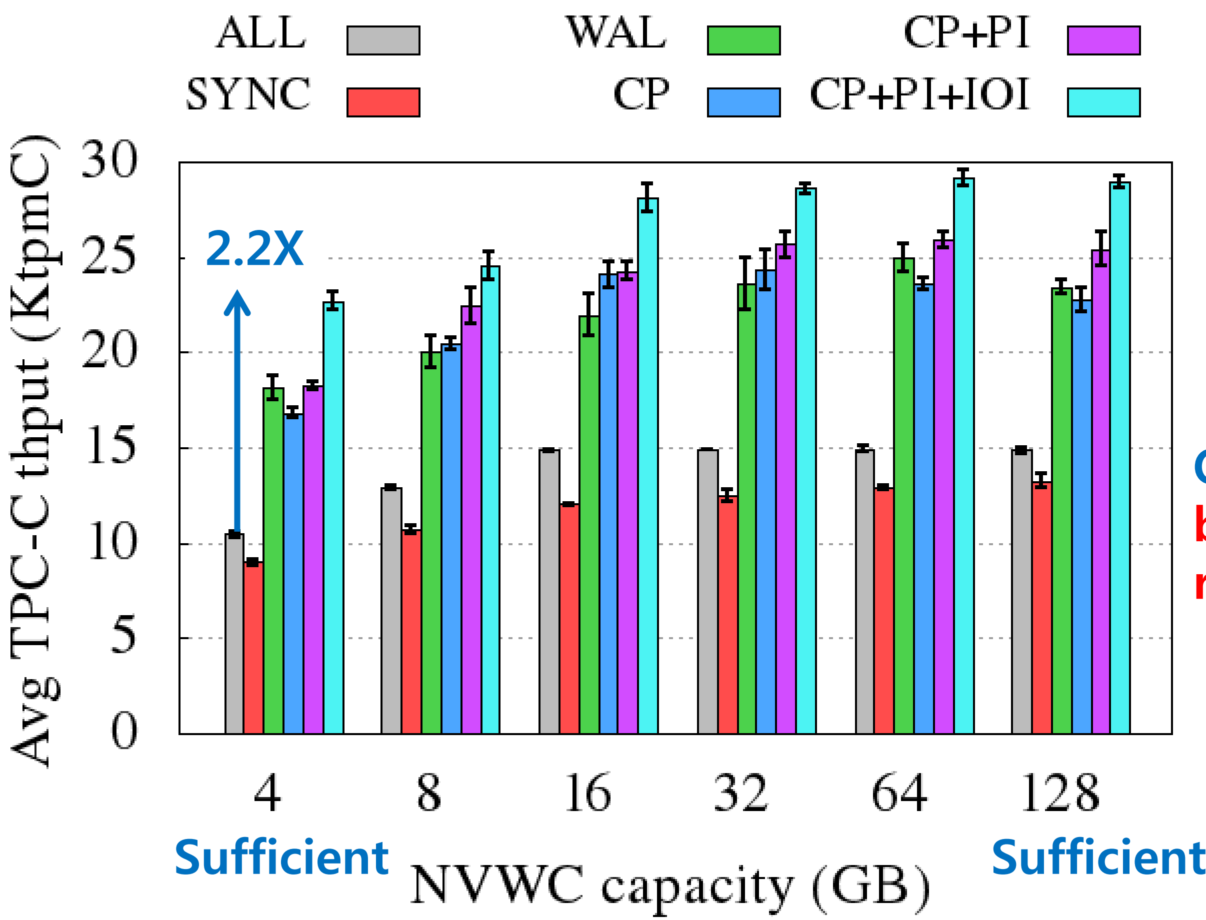


Same performance w/ 72% less cached writes

Our scheme resolves capacity contention & runtime dependencies

PostgreSQL Performance

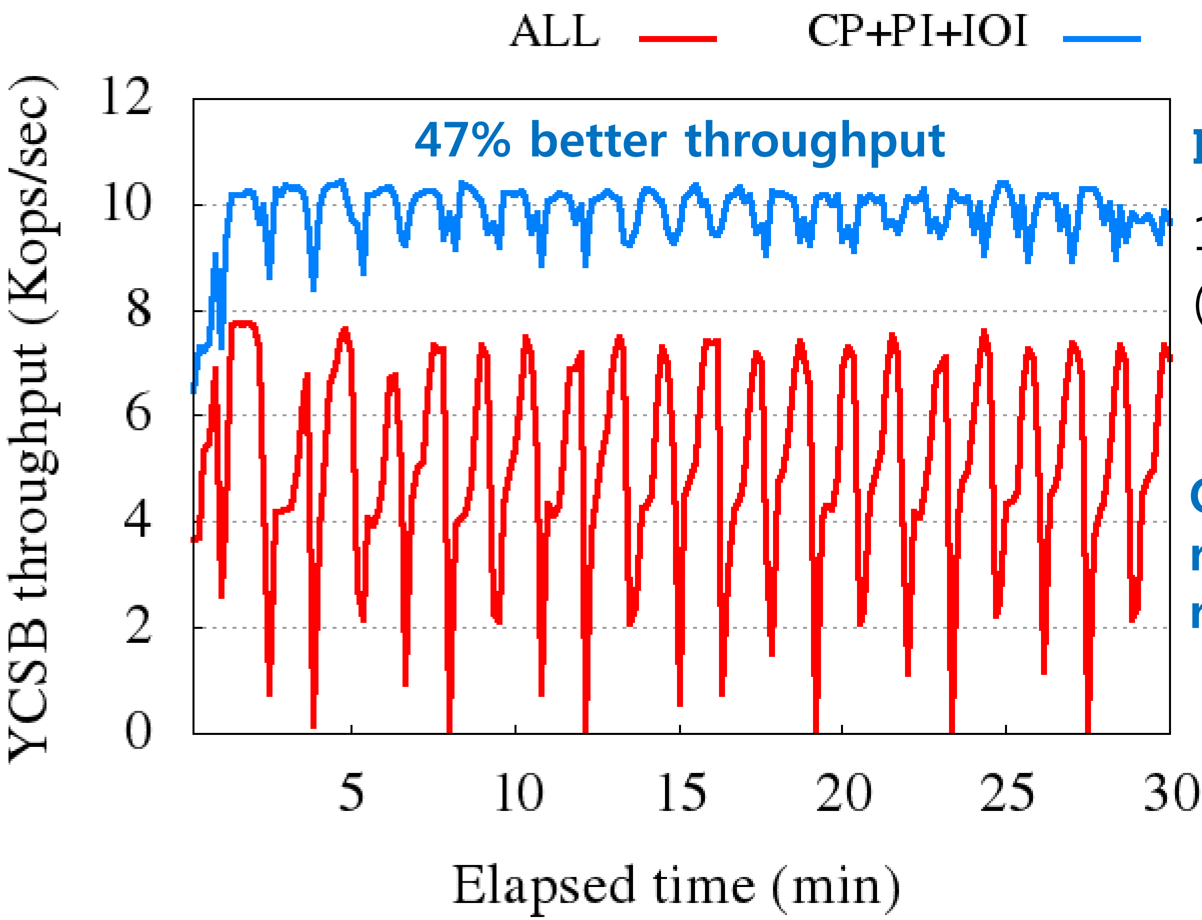
- TPC-C workload w/ SSD



Our scheme resolves **bandwidth contention** & **runtime dependencies**

Redis Performance

- Update-heavy workload w/ 16GB SSD



Improved tail latency

13X better @ 99.9th %ile

(50ms vs. 649ms)

Our scheme improves request throughput & request latency

Conclusions

- Key observation
  - Each write has different performance-criticality
- Request-oriented caching policy
  - Solely utilizes NVWC for application performance
  - **Improves** performance while **reducing** cached writes
- Future work
  - System-level critical process identification
  - Application to user-interactive environments

Thank You!

- Questions and comments

- Contact
  - sw.kim@skku.edu