Request-aware Cooperative I/O Scheduling for Scale-out Database Applications

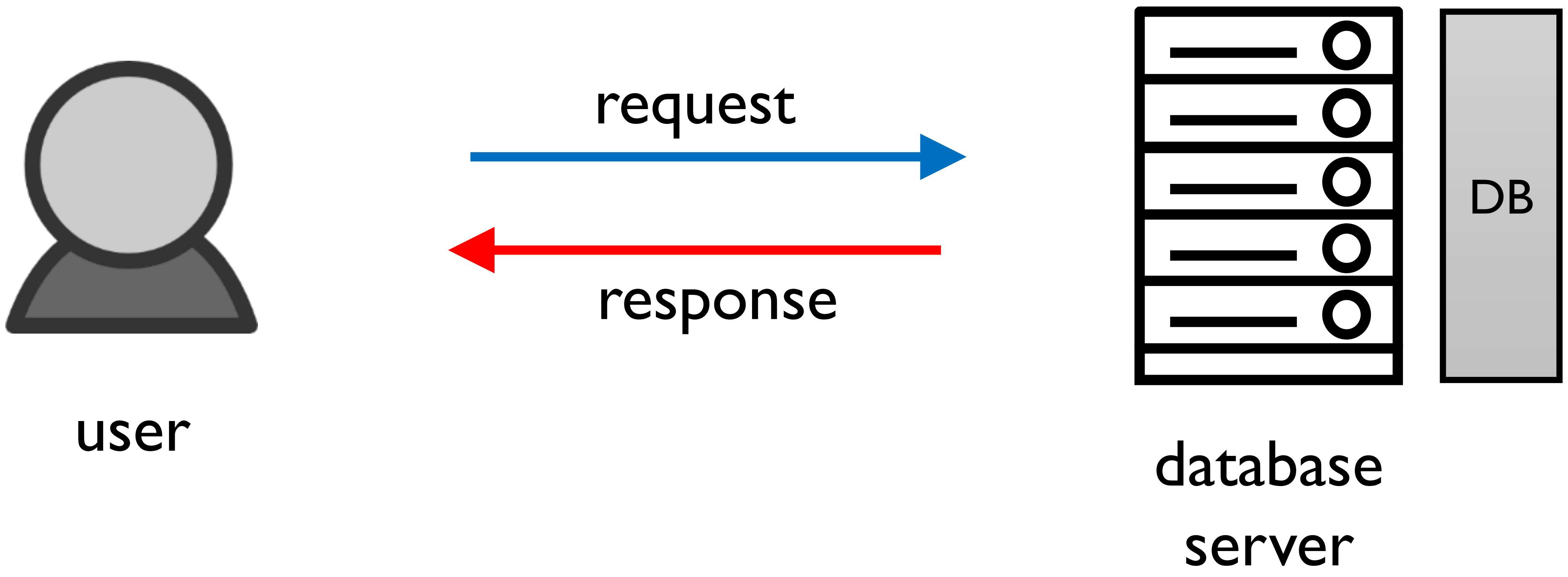
Hyungil Jo*, Sung-hun Kim*, Sangwook Kim**, Jinkyu Jeong*, Joonwon Lee*

*Computer Systems Lab. at Sungkyunkwan University

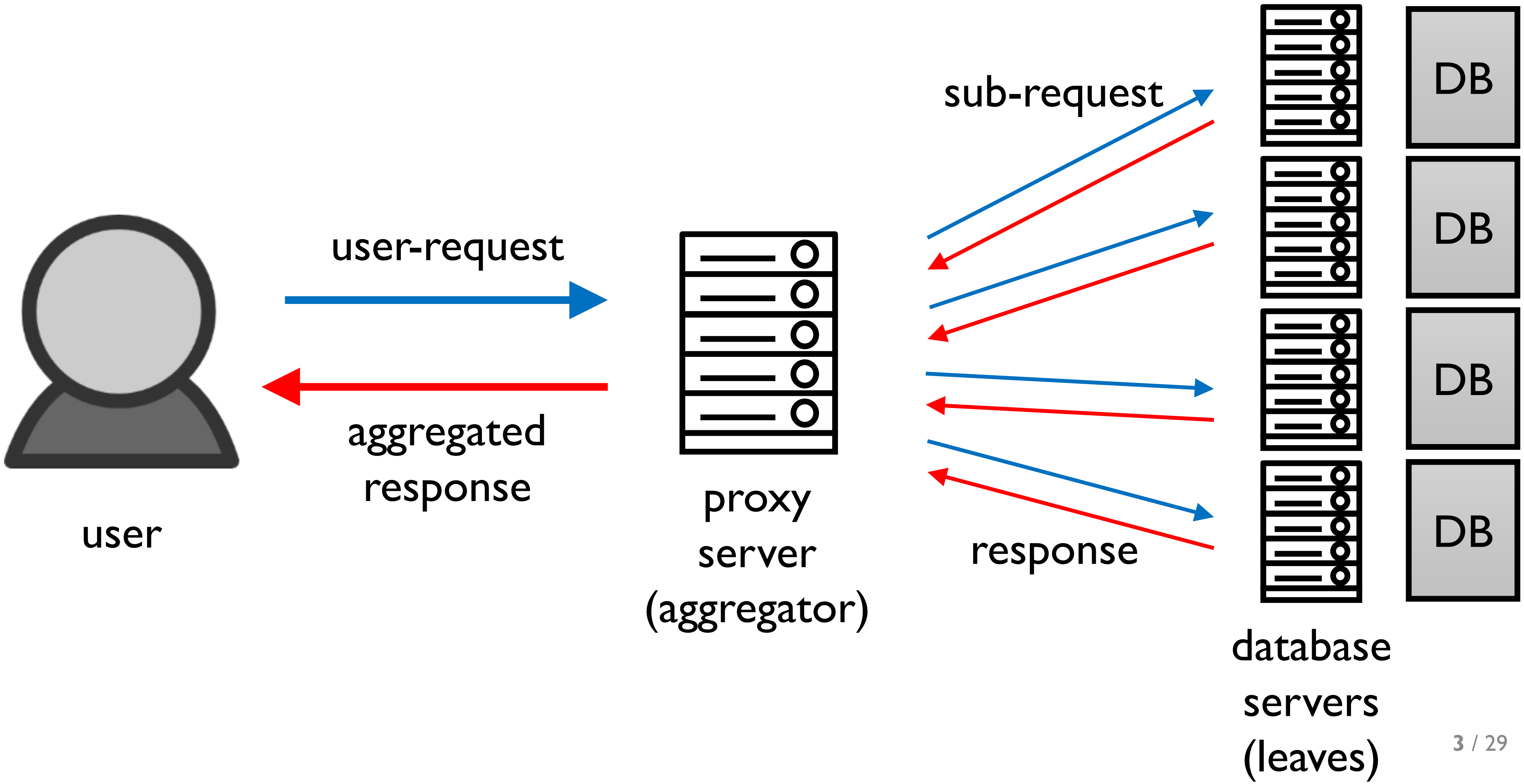
**Apposha

SUNG KYUN KWAN UNIVERSITY

Introduction

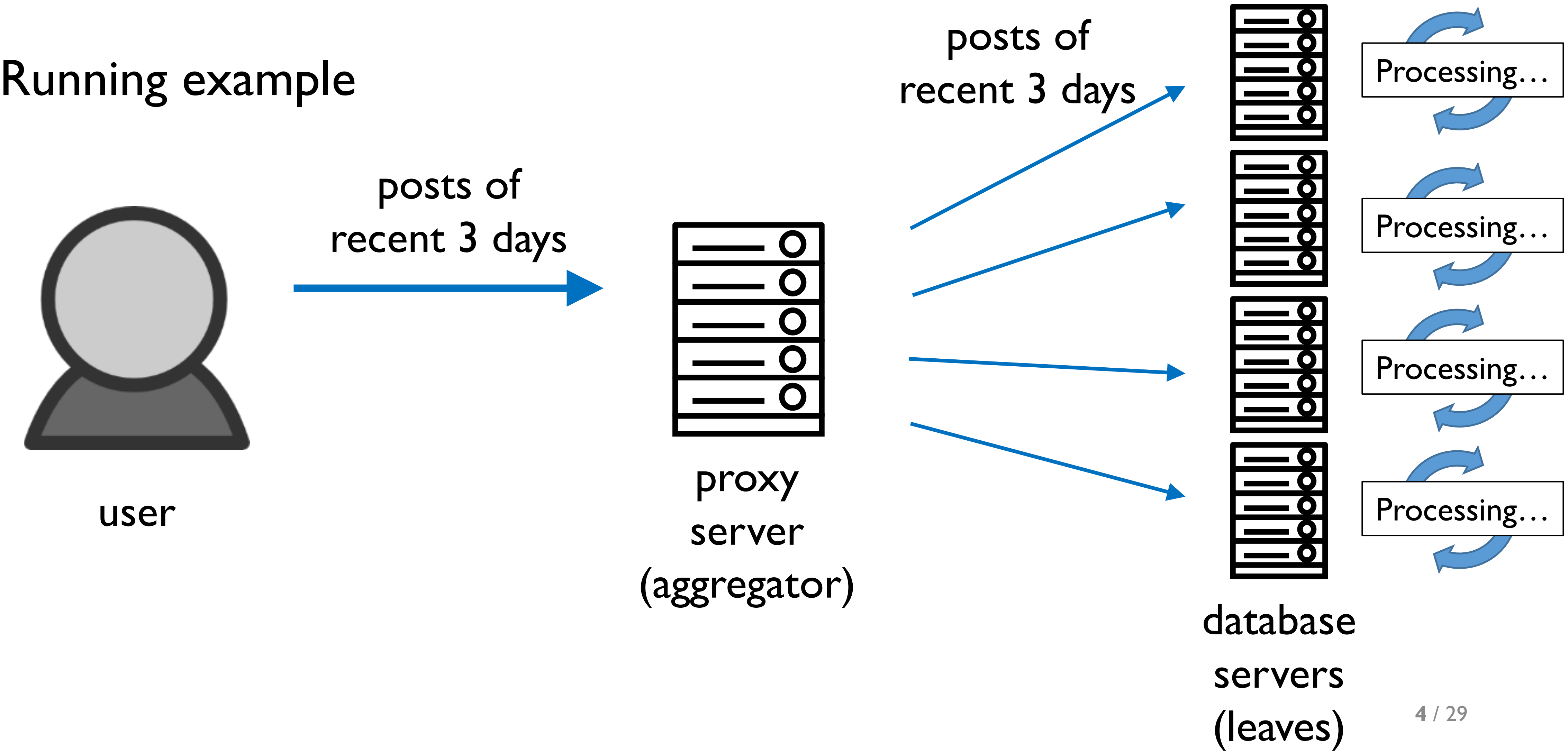


Scale-out Architecture



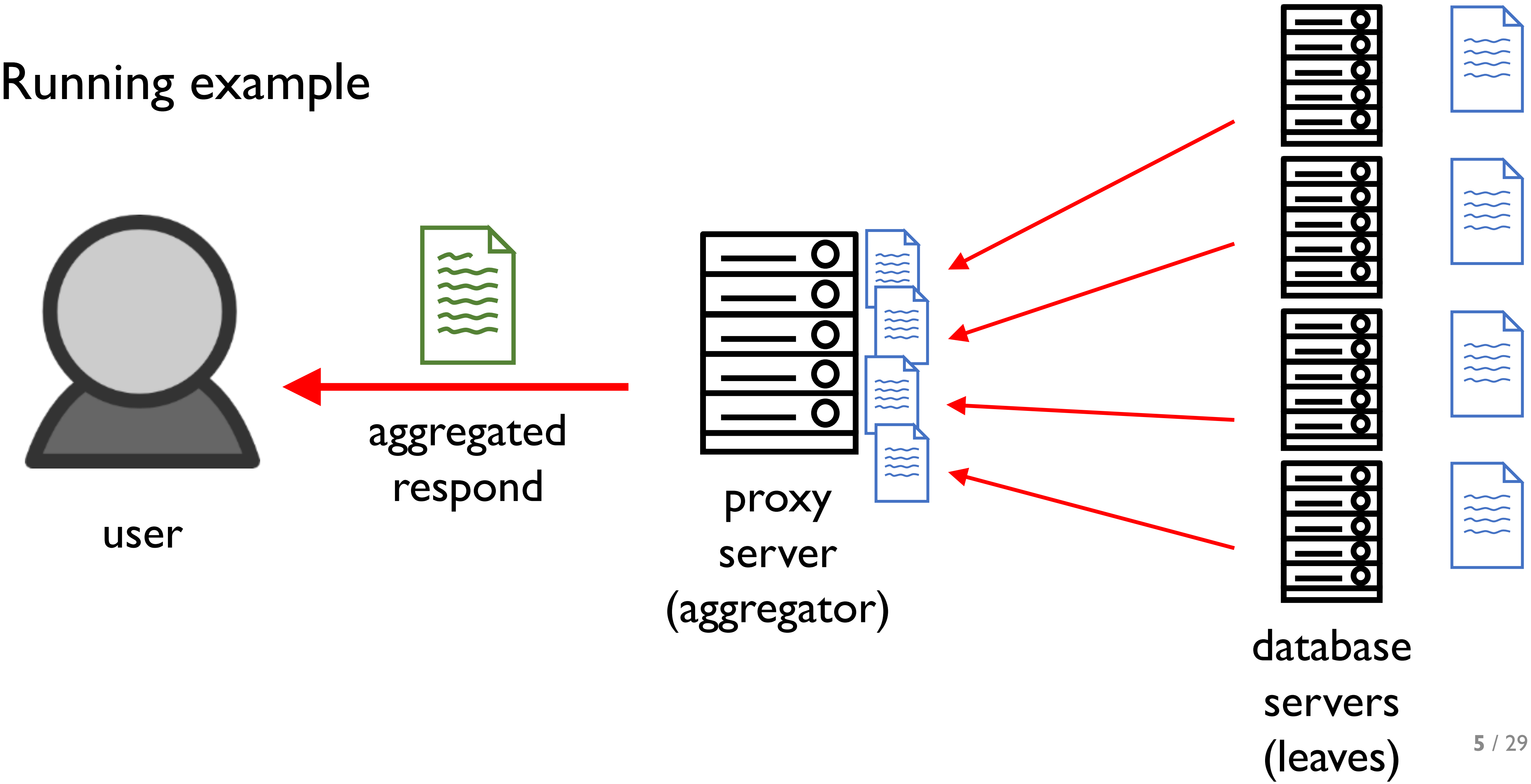
Scale-out Architecture

- Running example

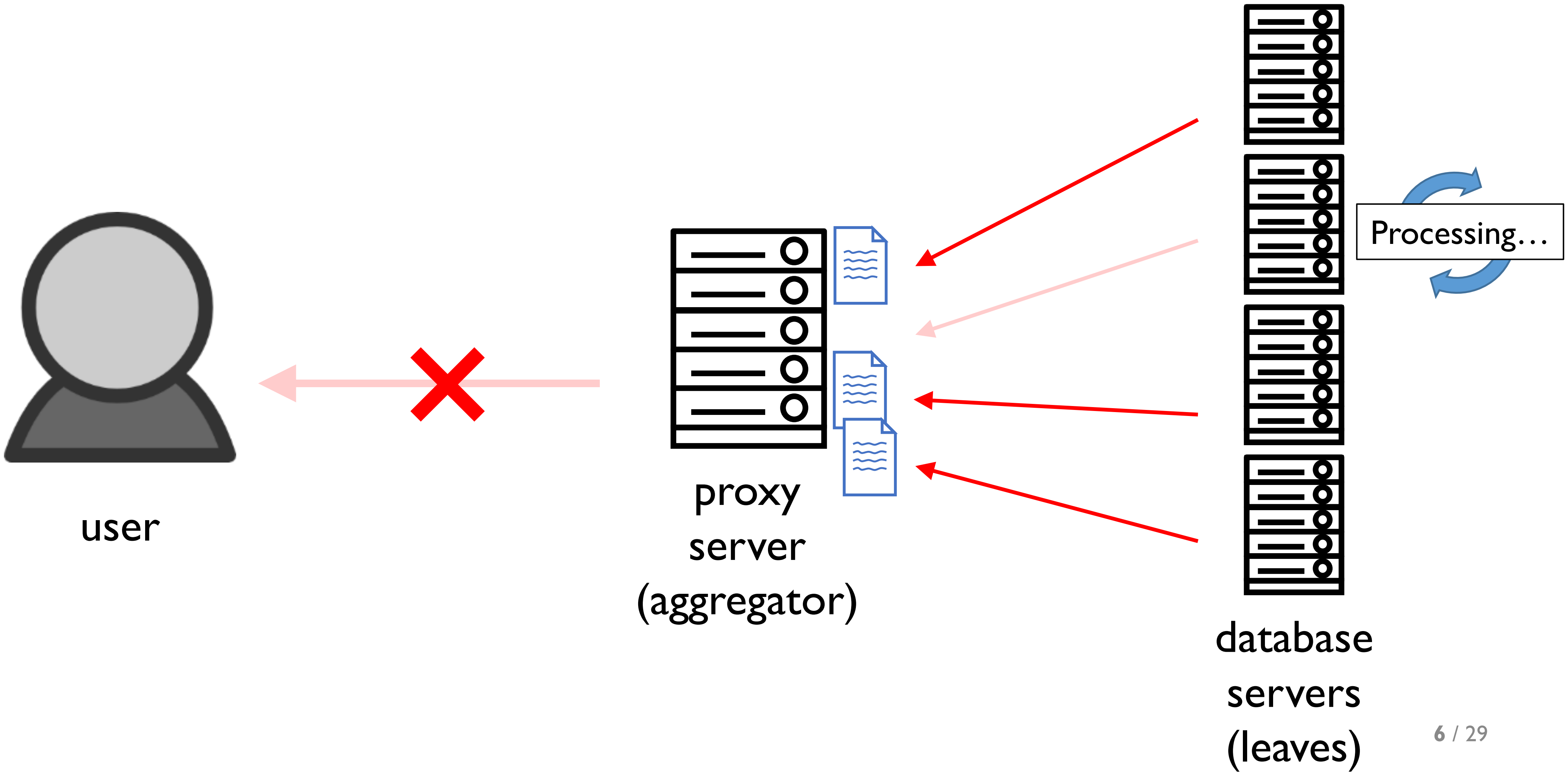


Scale-out Architecture

- Running example

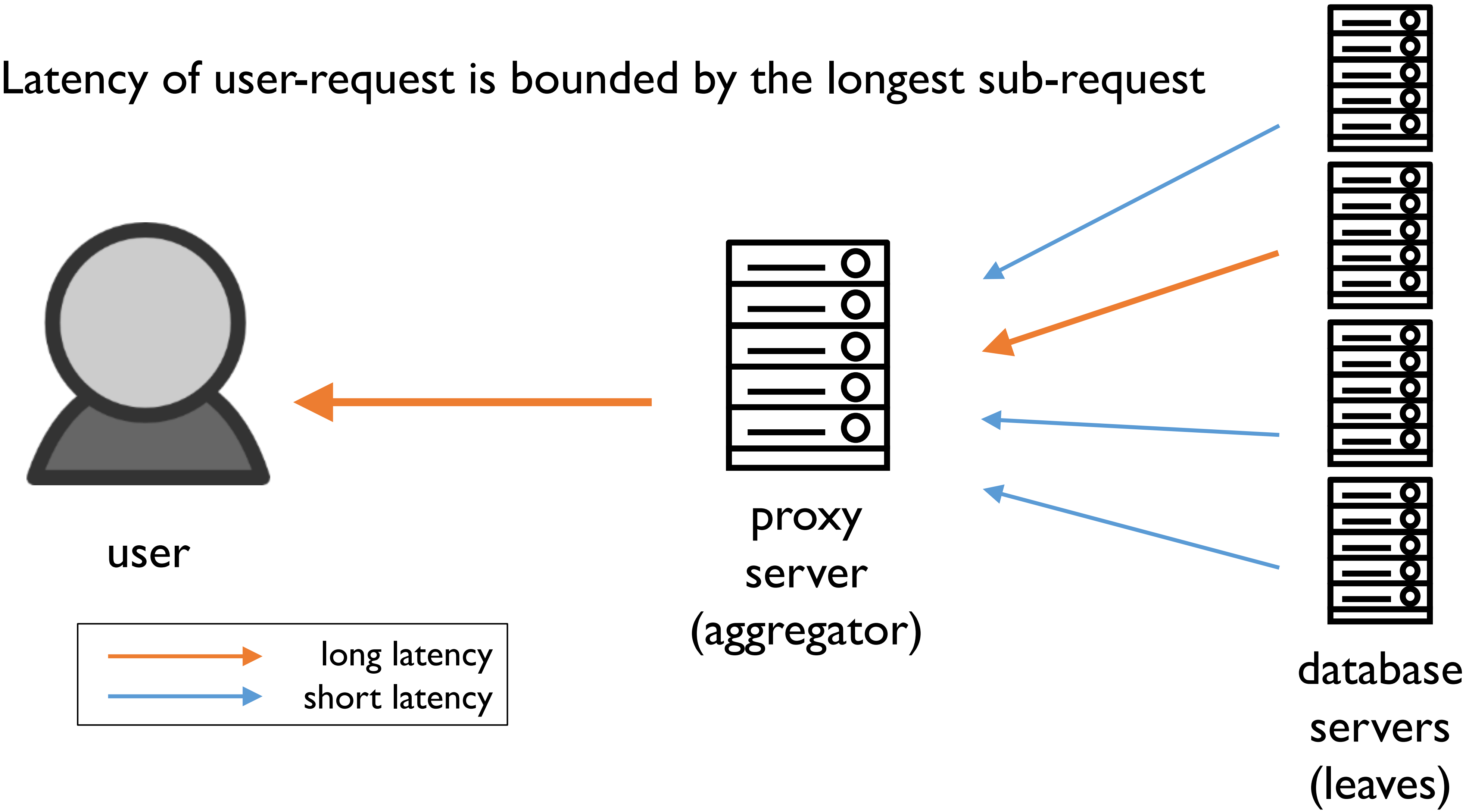


Cause of Tail Latency



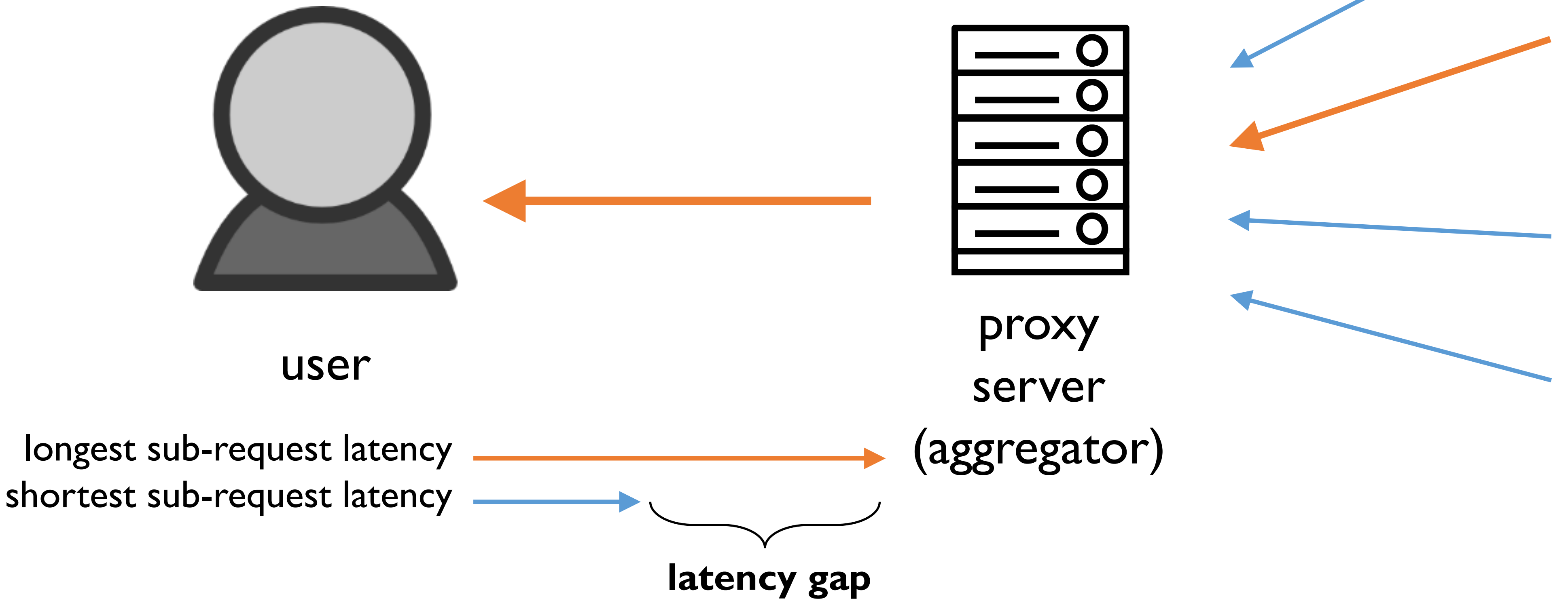
Cause of Tail Latency

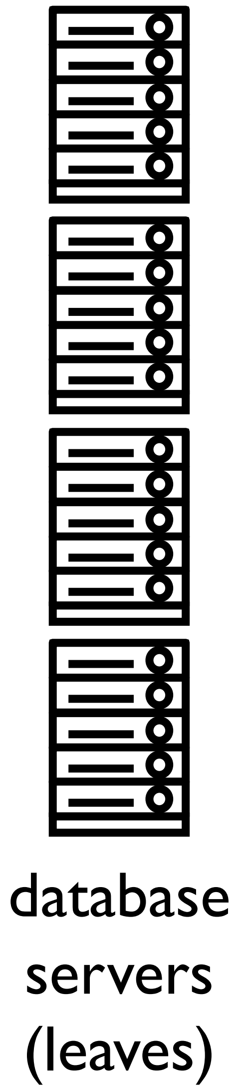
- Latency of user-request is bounded by the longest sub-request



Cause of Tail Latency

- Latency of user-request is bounded by the longest sub-request
- Tail latency is usually aggravated by long latency gap





Agenda

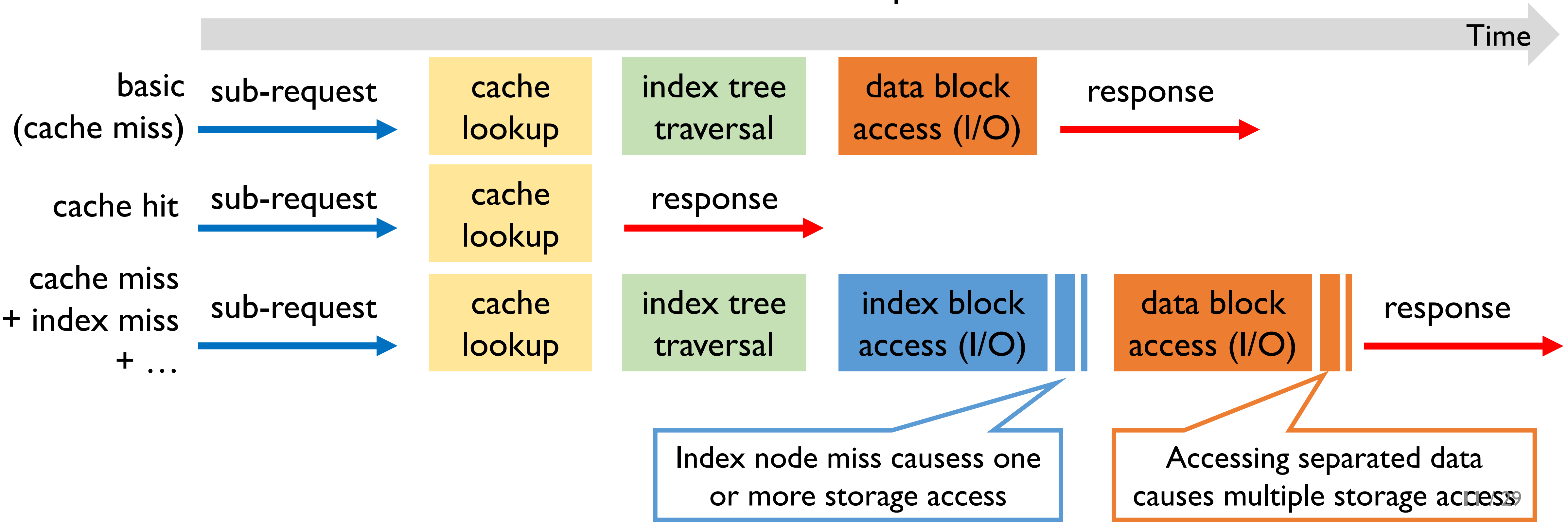
- Analysis of the factors affecting latency gap
  - Server-internal state
  - I/O interleaving
  - Unsynchronized I/O handling
- Request-aware cooperative I/O scheduling
  - Batched I/O scheduling in individual server
  - Synchronized I/O handling across multiple servers

Agenda

- Analysis of the factors affecting latency gap
  - Server-internal state
  - I/O interleaving
  - Unsynchronized I/O handling
- Request-aware cooperative I/O scheduling
  - Batched I/O scheduling in individual server
  - Synchronized I/O handling across multiple servers

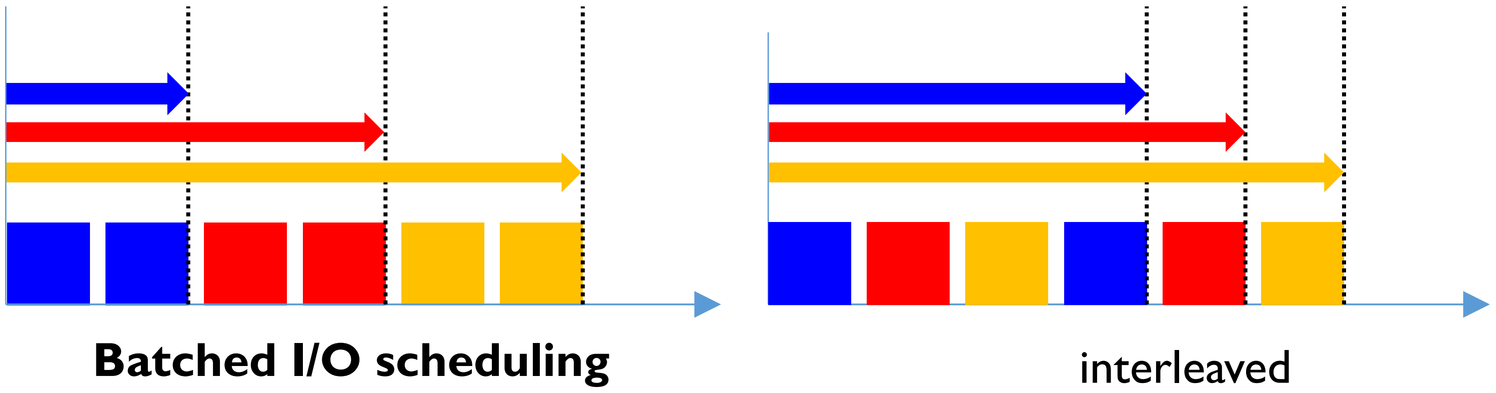
Factors Affecting Latency Gap (1)

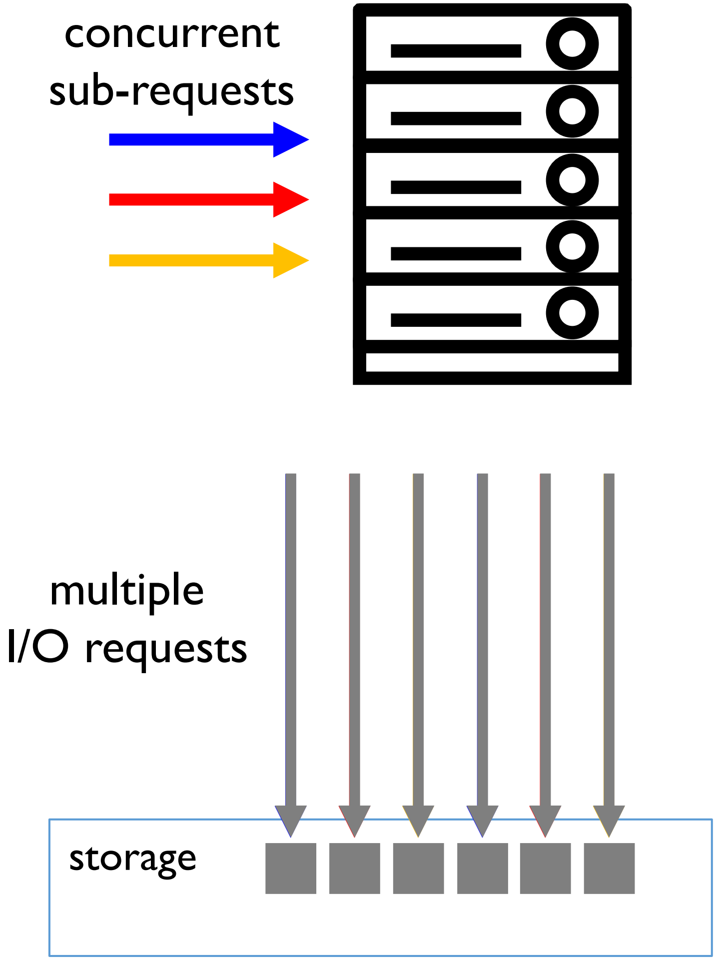
- Server-internal state
  - Different server states make various I/O patterns



Factors Affecting Latency Gap (2)

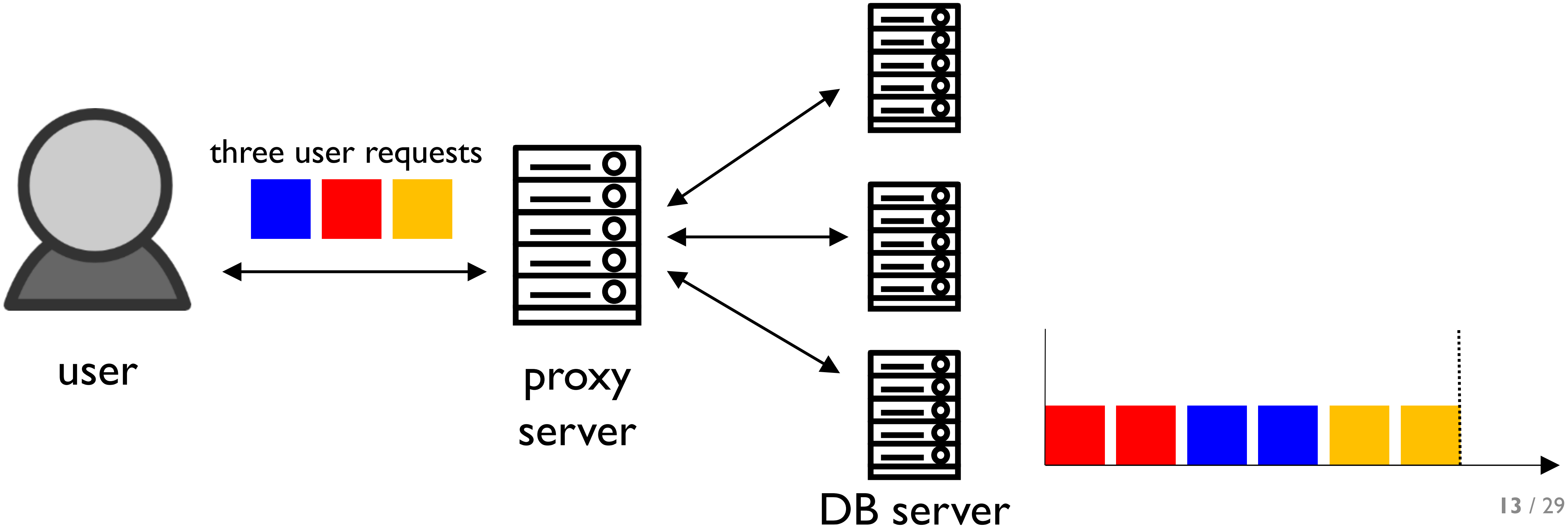
- I/O interleaving
  - Sub-request can issue multiple I/Os
    - Multiple sub-requests issue a number of I/Os
  - Kernel handle I/O request in best effort basis
    - Process I/O requests in receiving order





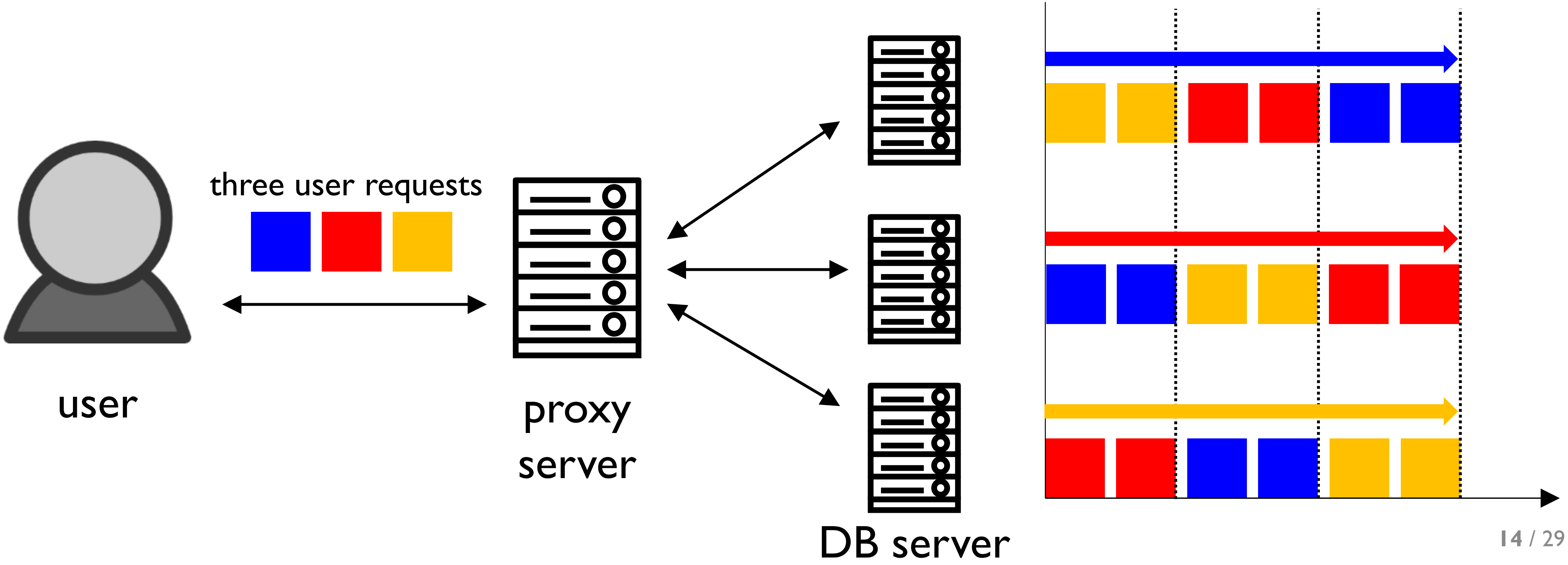
Factors Affecting Latency Gap (3)

- Unsynchronized I/O handling induced by user request
  - Batched I/O scheduling is not effective if sub-requests are unsynchronized



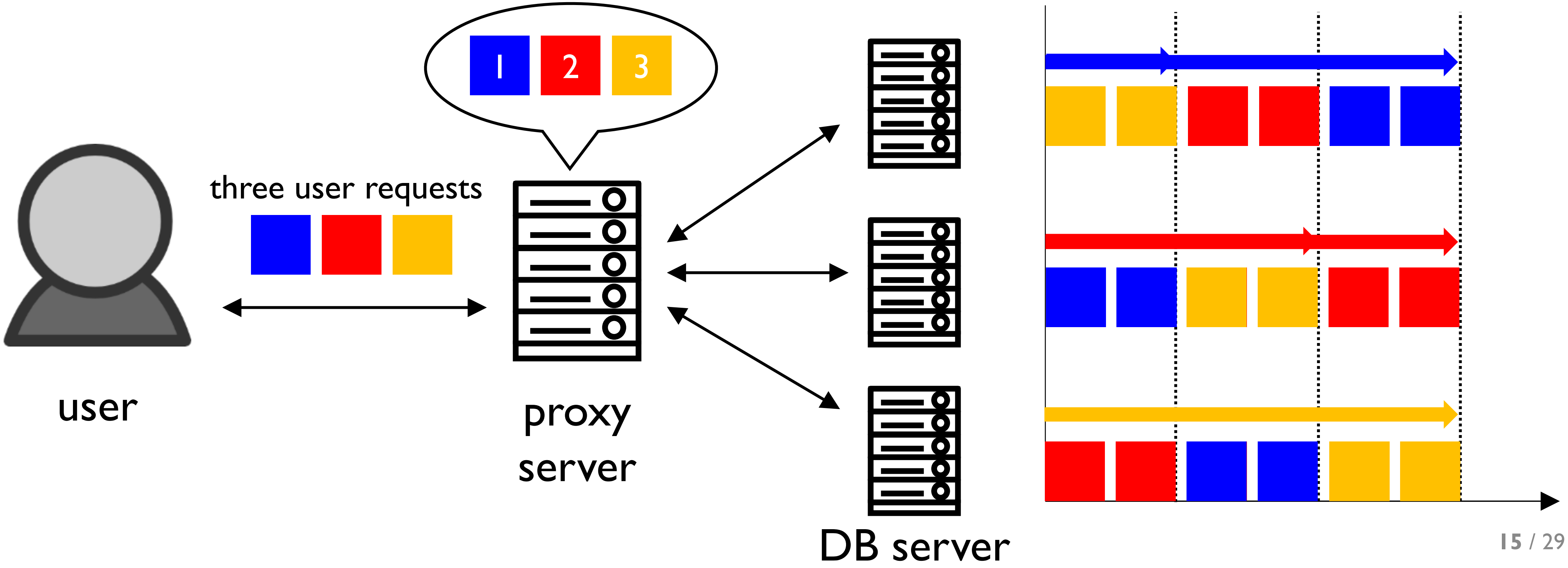
Factors Affecting Latency Gap (3)

- Unsynchronized I/O handling induced by user request
  - Batched I/O scheduling is not effective if sub-requests are unsynchronized



Factors Affecting Latency Gap (3)

- **Synchronized I/O handling across multiple servers**
  - Exploit user-request arrival order



Our Approach

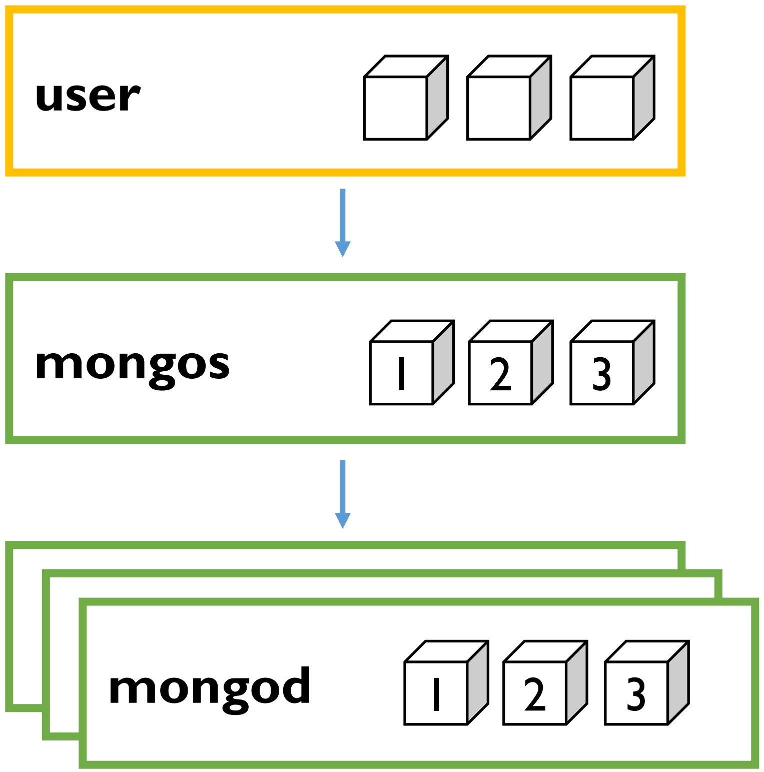
- Problems - suggested solutions
  - ~~Server-internal state~~
  - I/O interleaving - I/O scheduling in ***a batched manner***
  - Unsynchronized I/O handling - propagating ***request arrival order***

Implementation Overview

- Two components of implementation
  - Propagating request arrival order
  - Scheduling I/Os according to request arrival order
- Implemented in read path of MongoDB and Linux kernel
  - Propagating request arrival order
    - mongos instance (proxy server)
    - mongod instance (database server)
    - No changes in a core engine of database application
  - Cooperative I/O scheduler

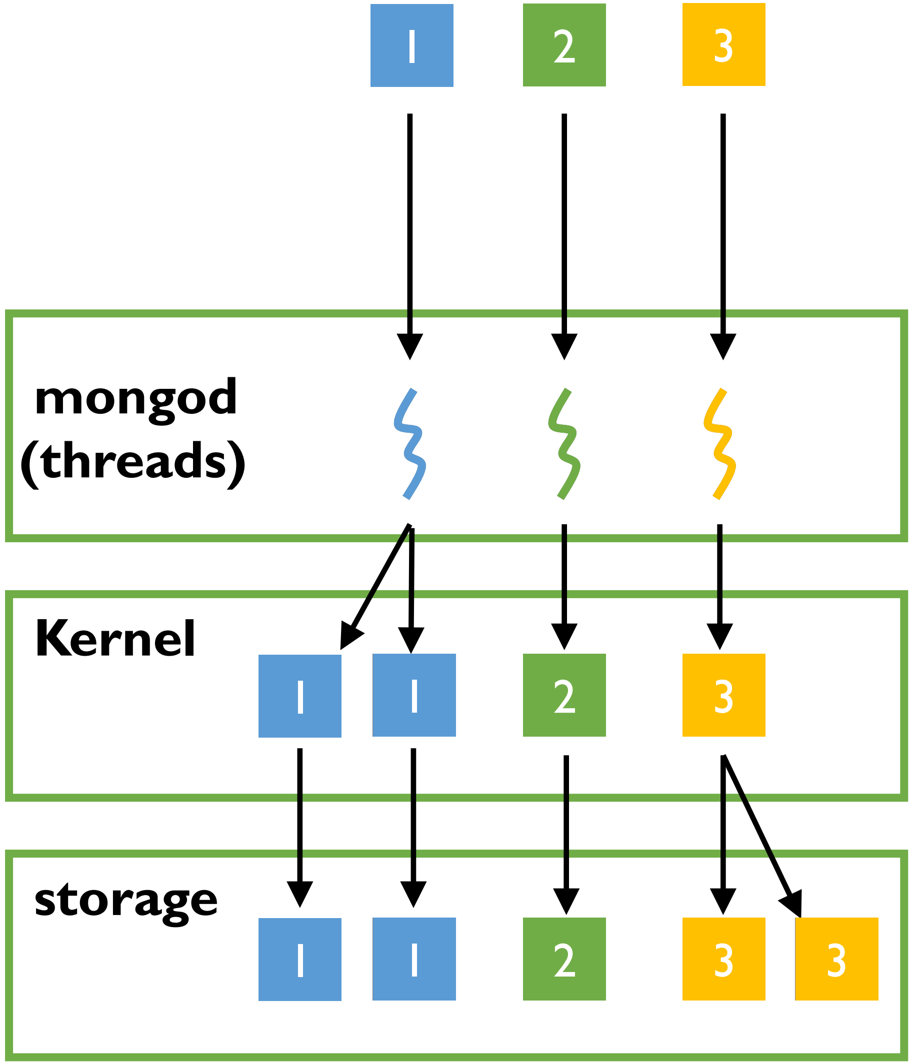
Propagating Request Arrival Order (1/2)

- Mongos (proxy)
  - Capture the user request arriving order
  - Allocate request id in request arrival order
  - Attach request id to sub-requests when a proxy sends them



Propagating Request Arrival Order (2/2)

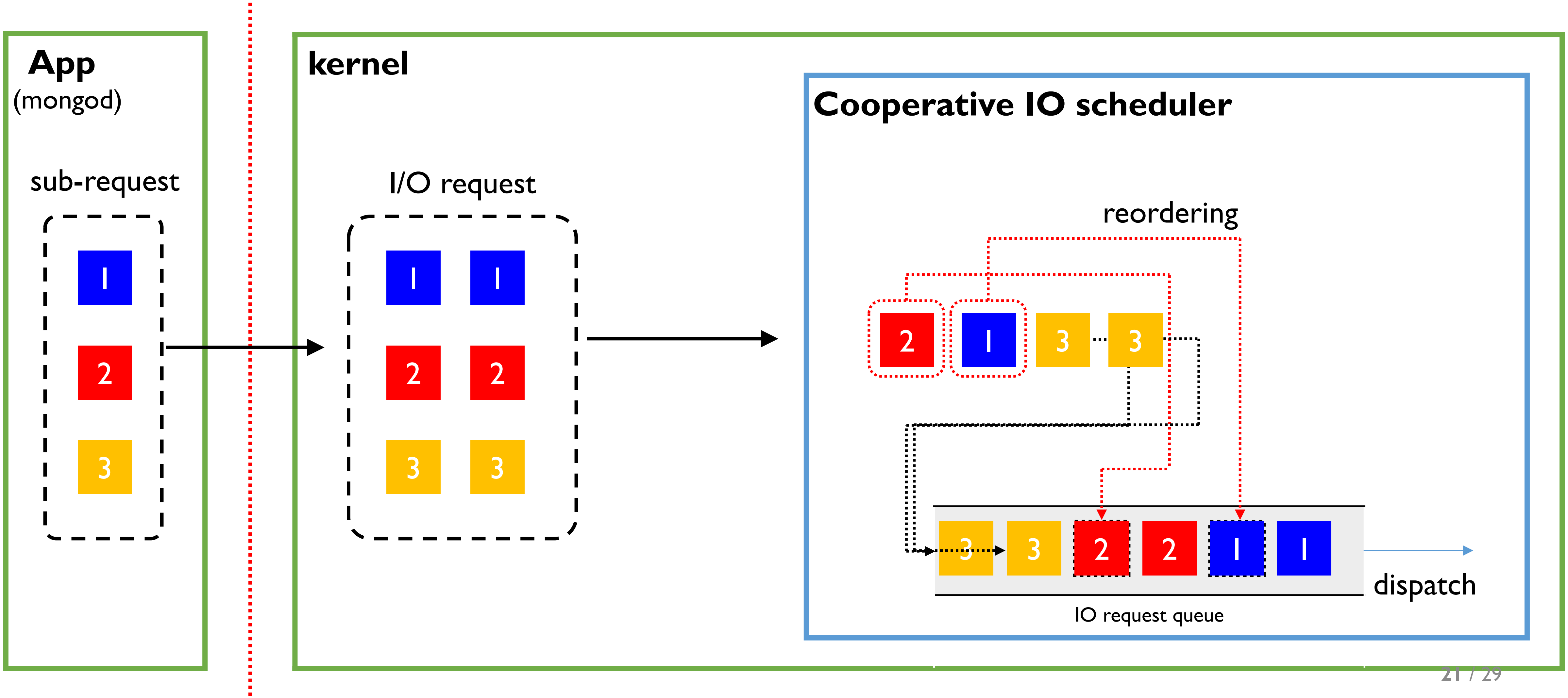
- Mongod (database)
  - Kernel provides two syscalls
    - Notify request id of an outstanding sub-request
    - CTX_BEGIN, CTX_END
  - Each mongod thread handles only one sub-request at the same time
  - Kernel tags request id when threads issue I/Os



Cooperative I/O Scheduler (1/2)

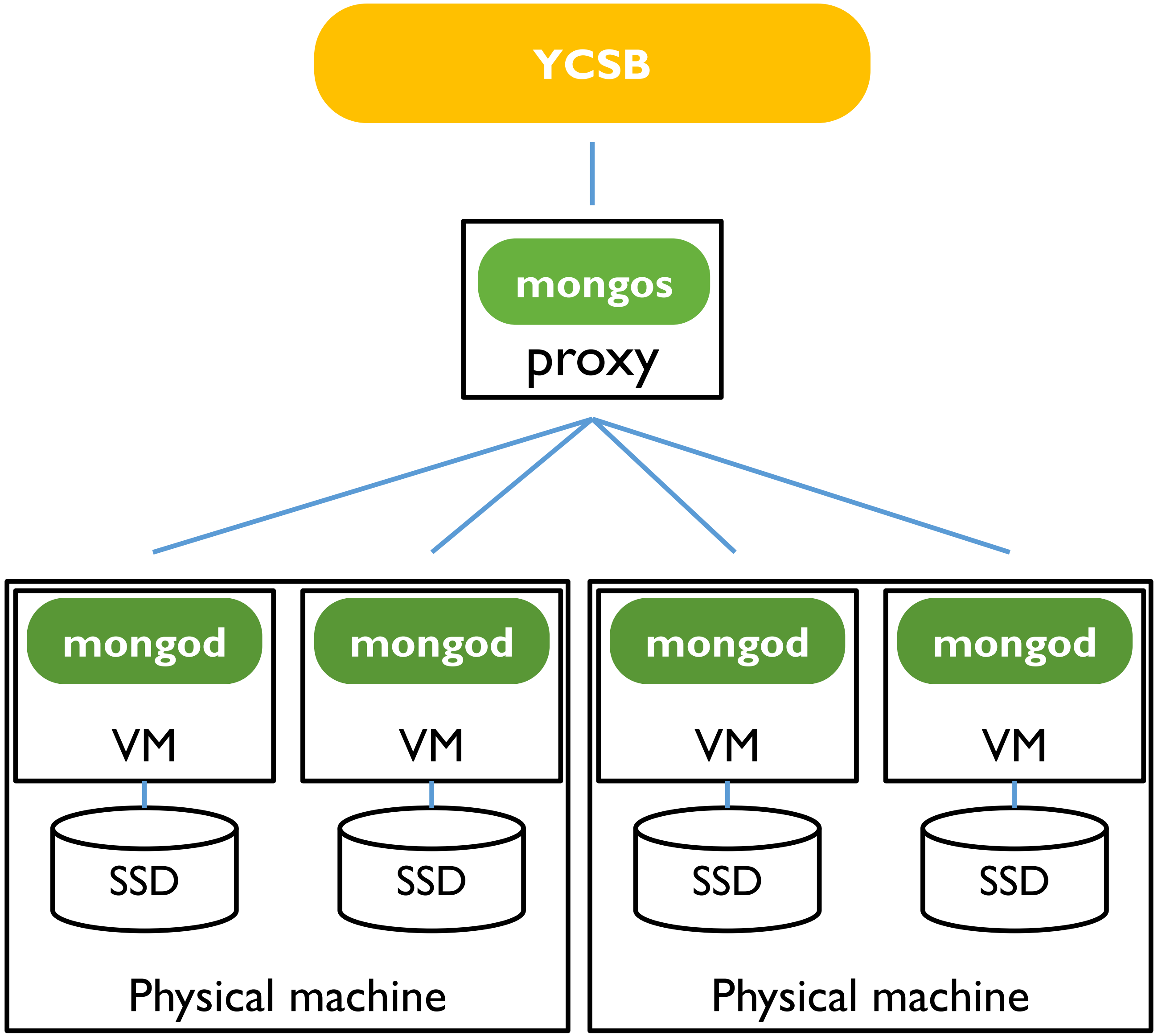
- Cooperate to achieve one goal
  - Latency of user-request is a matter
- Scale-out architecture of database application
  - I/O schedulers should follow propagated request arrival order
- Multi-threaded nature in mongod
  - Threads should endure un-fairness in I/O scheduling

Cooperative I/O Scheduler (2/2)



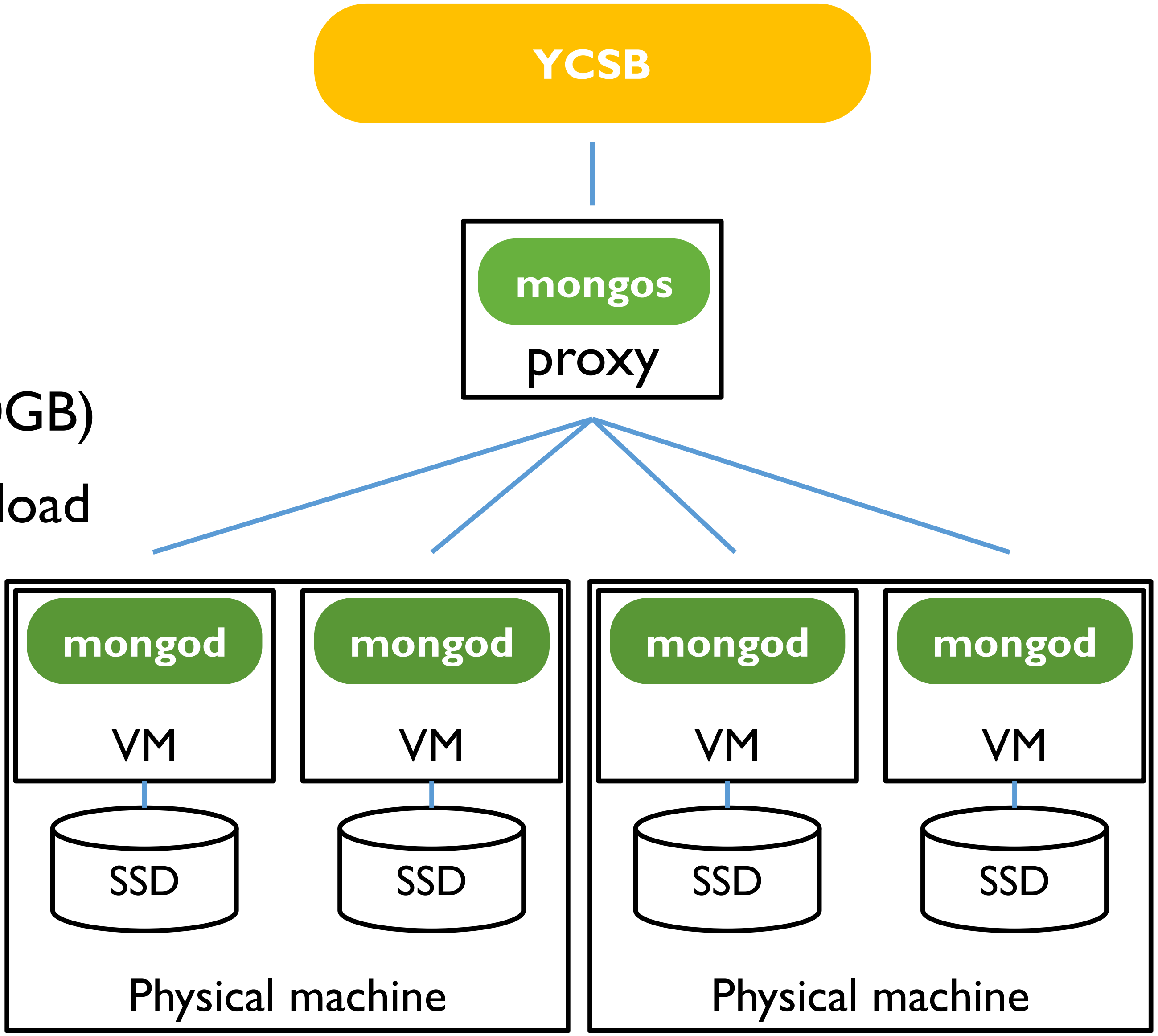
Evaluation Setup

- Physical machine
  - CPU : Intel Xeon E5-2650 x2 (64 core)
  - Memory : 32GB
  - Storage : SAS SSD x2 (dedicated to VM)
- Virtual machine
  - vCPU (x2)
  - Memory 2GB
- Network
  - 10Gbit/s ethernet



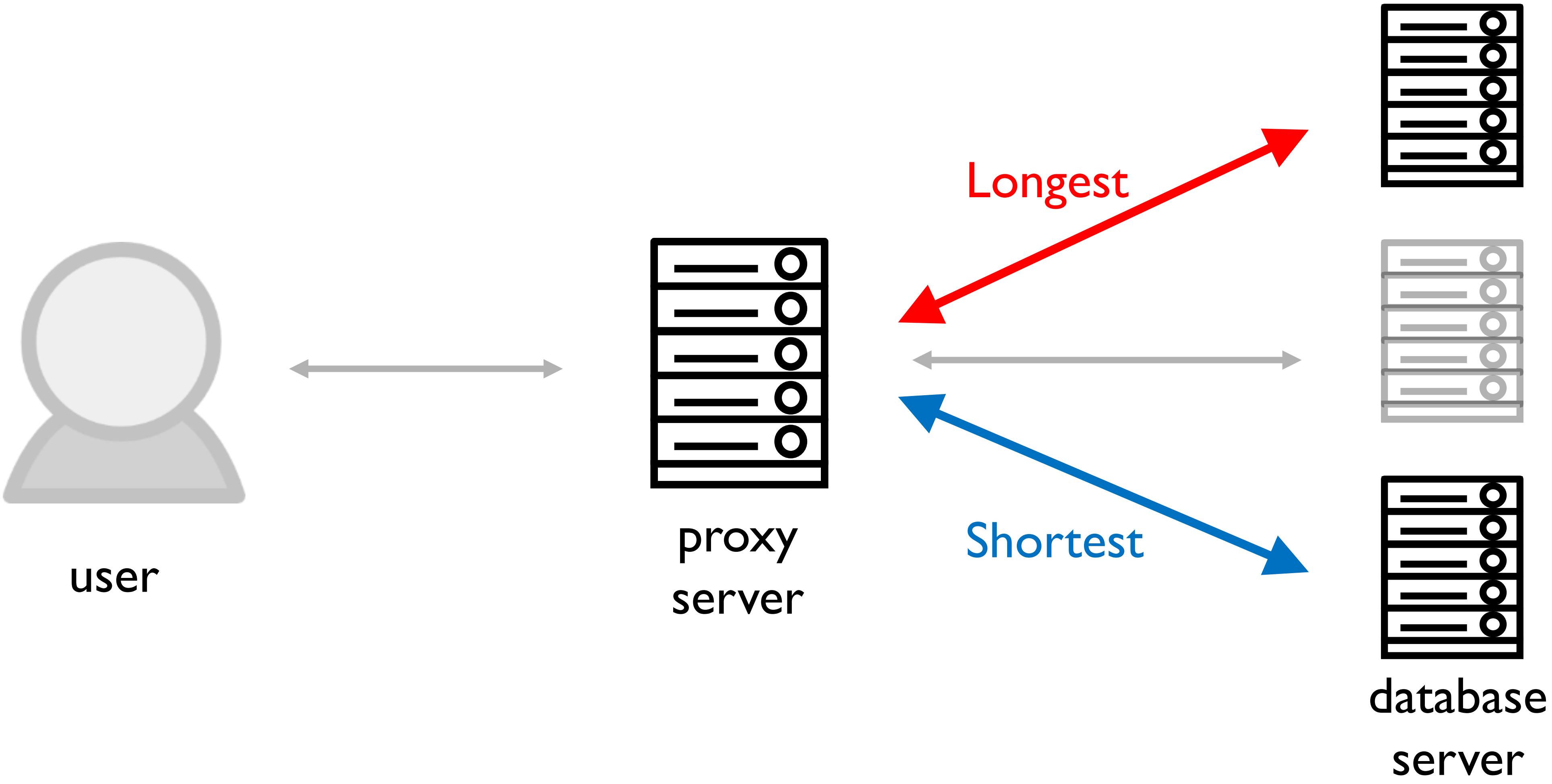
Evaluation Setup

- OS : Linux 4.8.2
- DB : MongoDB 3.2.10
- Data set : 1KB record x 40 million (40GB)
- Workload : YCSB synthetic scan workload
  - consist of scan query only
  - scan query : read 1~100 document which value is bigger than or equal to specific value
- I/O scheduler
  - noop, deadline, CFQ
  - coop (cooperative I/O scheduler)



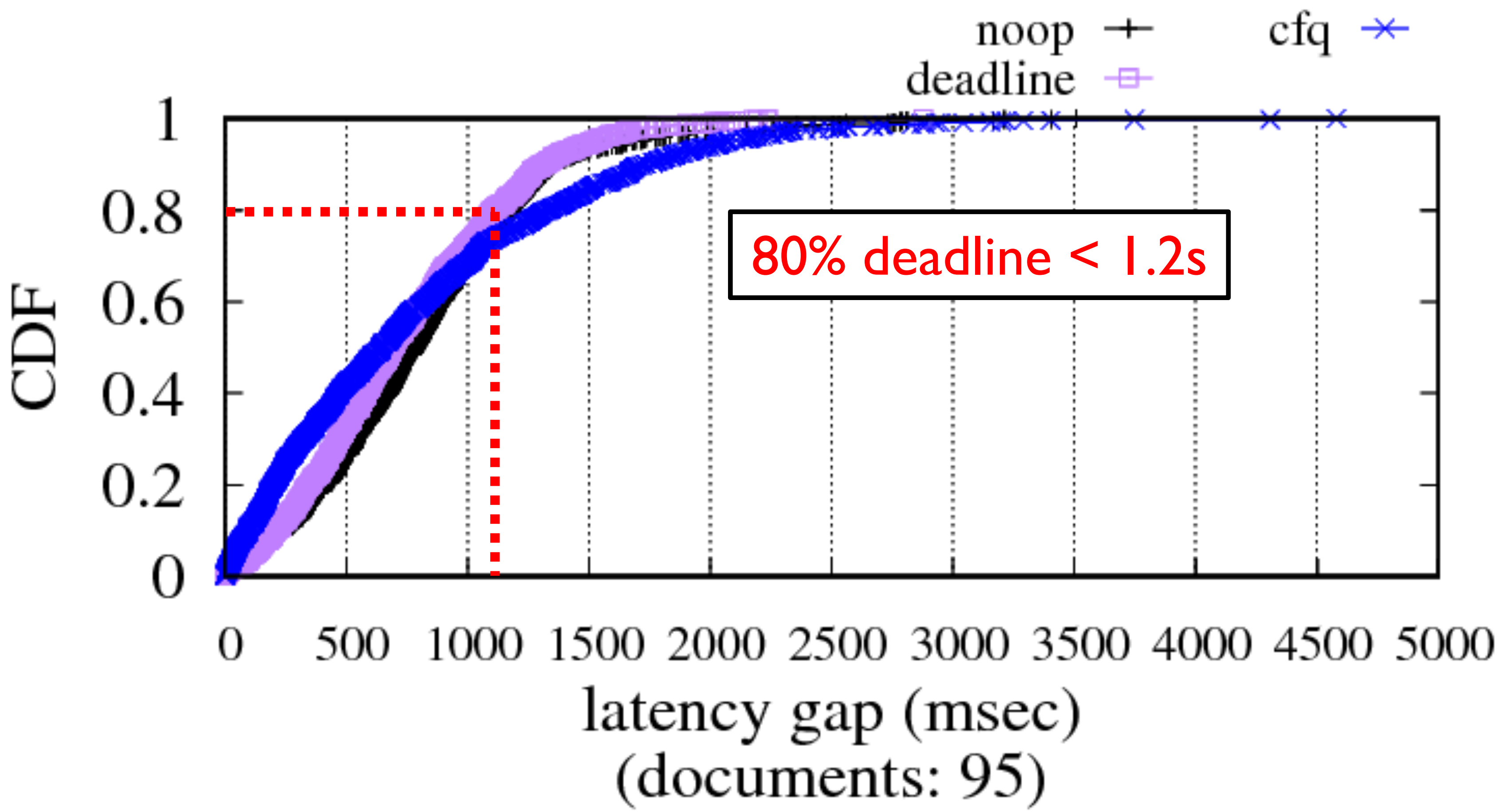
※ YCSB : Yahoo Cloud Serving Benchmark

Latency Gap in Each User-Request



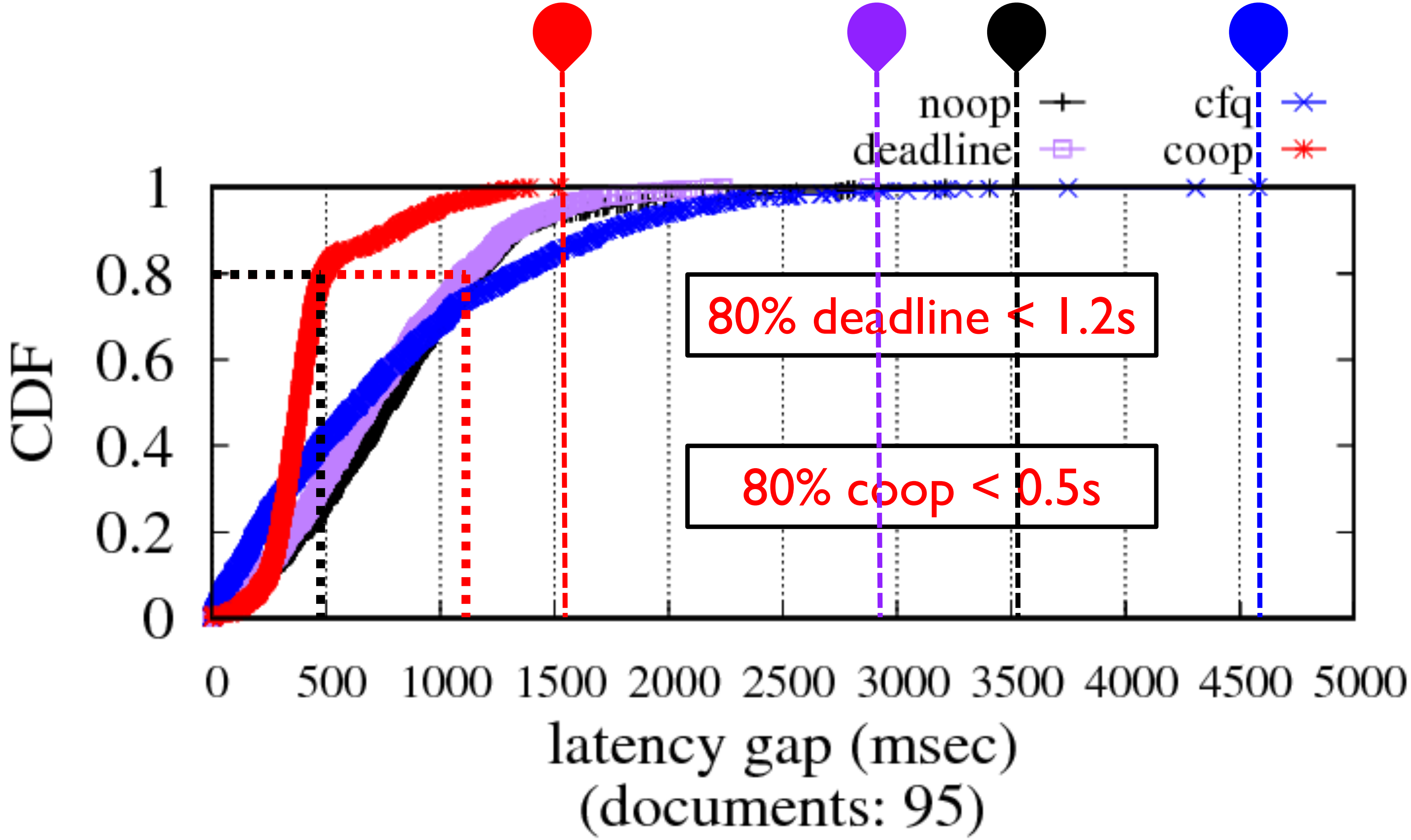
※ latency gap = difference between longest and shortest latency of sub-requests

Latency Gap in Each User-Request



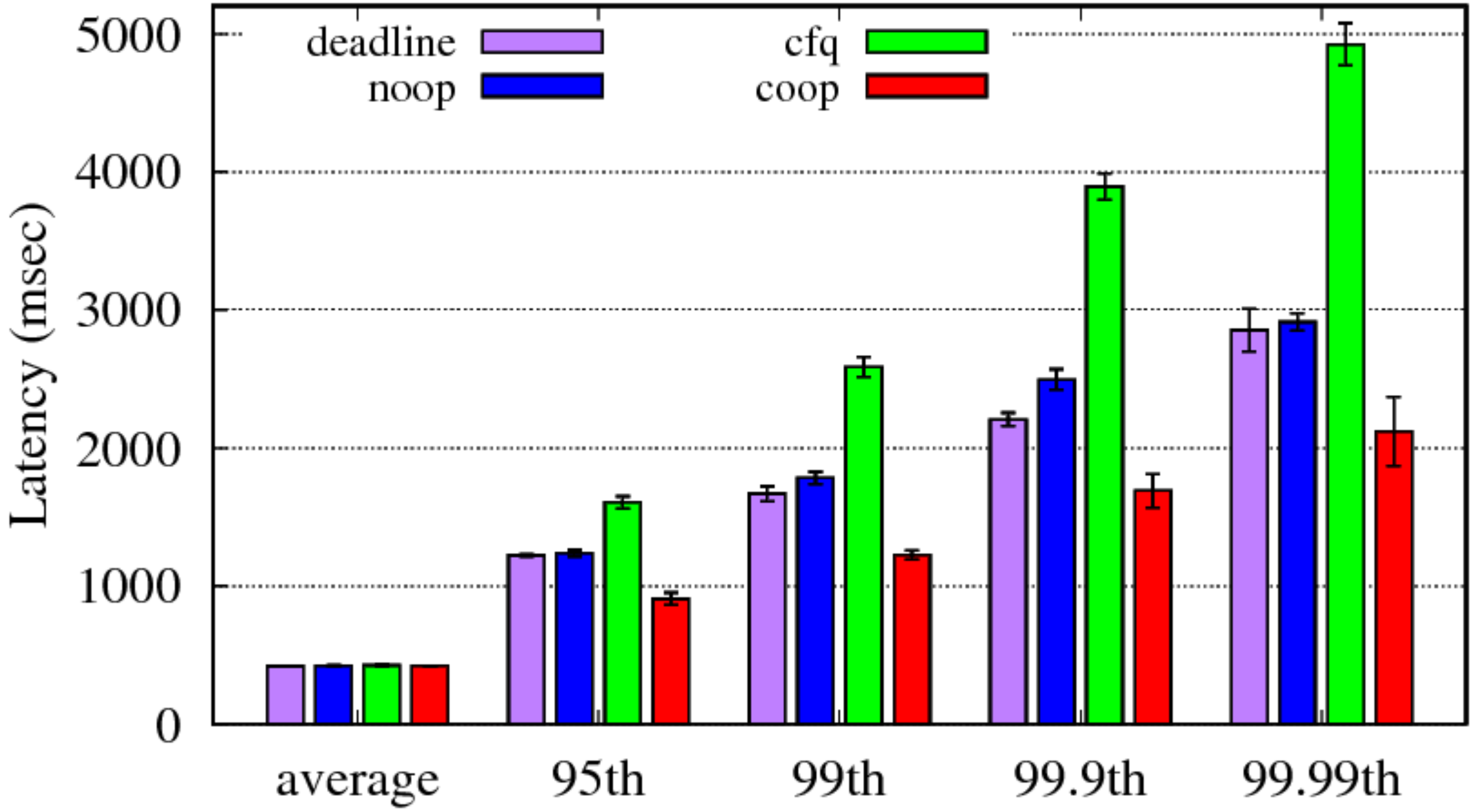
※ latency gap = difference between longest and shortest latency of sub-requests

Latency Gap in Each User-Request



※ latency gap = difference between longest and shortest latency of sub-requests

Scan Workload Result



Tail latency reduction by
23~26% compared to **deadline**
26~32% compared to **noop**
43~57% compared to **cfq**

Conclusion

- Summary
  - Latency of user request is bounded to longest sub-request
  - Suggest request-aware cooperative I/O scheduling scheme
    - Propagate request arrival order from application proxy to kernel I/O scheduler
    - Schedule I/Os in ***1)request arrival order*** and a ***2)batched manner***
    - Tail latency is reduced up to 57% than default I/O schedulers
- Future work
  - Implementing in full I/O path from application to kernel
  - Finding and applying optimal scheduling policy to reduce average latency

Q&A

Thank you for listening.

You can contact through hyungil.jo@csl.skku.edu