

# The Last Day

CS 161: Lecture 19

4/25/17

**LIE**  
DOWN



**TRY**  
NOT TO CRY



**CRY**  
A LOT



# Goals

- Take unmodified POSIX/Win32 applications . . .
- Run those applications in the cloud . . .
- On the same hardware used to run big-data apps . . .
- . . . and give them cloud-scale IO performance!

PostgreSQL



Microsoft®  
Exchange 2013



# Goals

- Take unmodified POSIX/Win32 applications . . .
- Run those applications in the cloud . . .
- On the same hardware used to run big-data apps . . .
- . . . and give them cloud-scale IO performance!

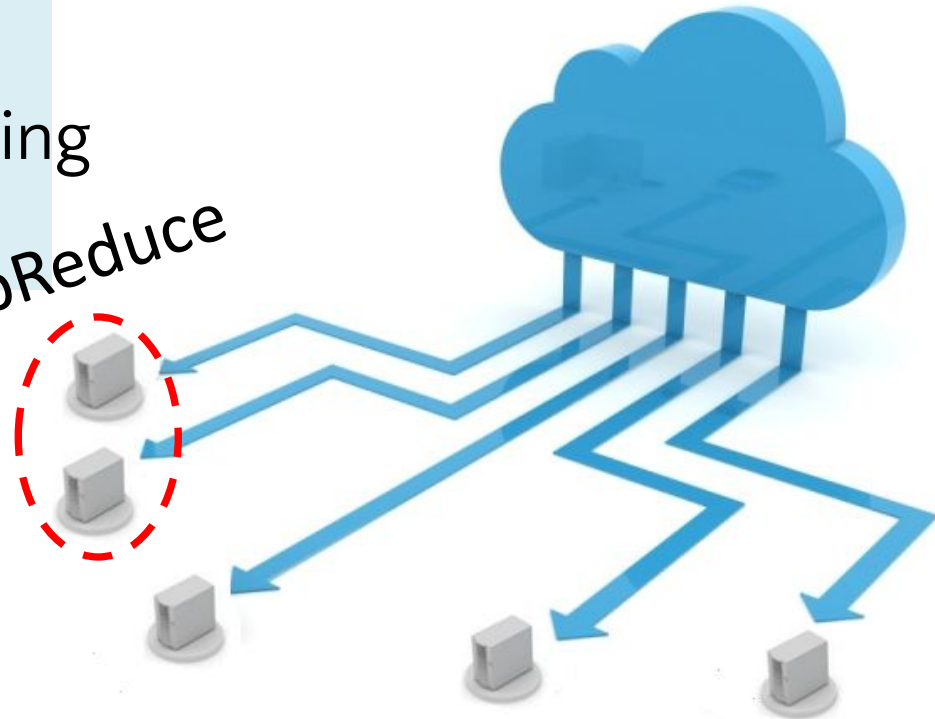
PostgreSQL Throughput > 1000 MB/s



Scale-out architecture using commodity parts



MapReduce



# Why Do I Want To Do This?

- Write POSIX/Win32 app once, automagically have fast cloud version
- Cloud operators don't have to open up their proprietary or sensitive protocols
- Admin/hardware efforts that help big data apps help POSIX/Win32 apps (and vice versa)

# Naïve Solution: Network RAID

PostgreSQL

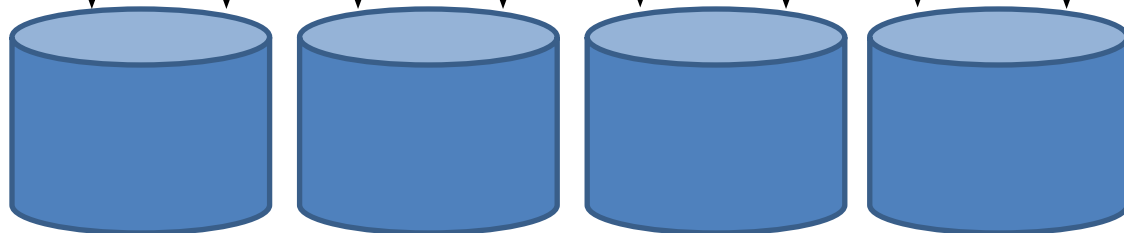


Microsoft®  
Exchange 2013


Blizzard  
virtual drive



Remote disks




# LISTEN



The naïve approach for implementing virtual disks does not **maximize spindle parallelism** for POSIX/Win32 applications which **frequently issue fsync() operations** to maintain consistency.

# LISTEN



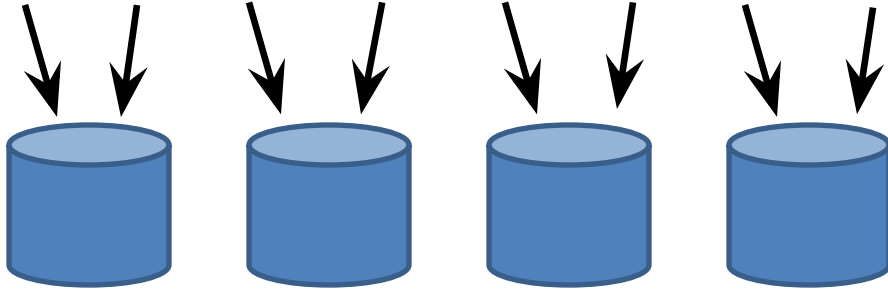
The naïve approach for implementing virtual disks does not **maximize spindle parallelism** for POSIX/Win32 applications which **frequently issue fsync() operations** to maintain consistency.



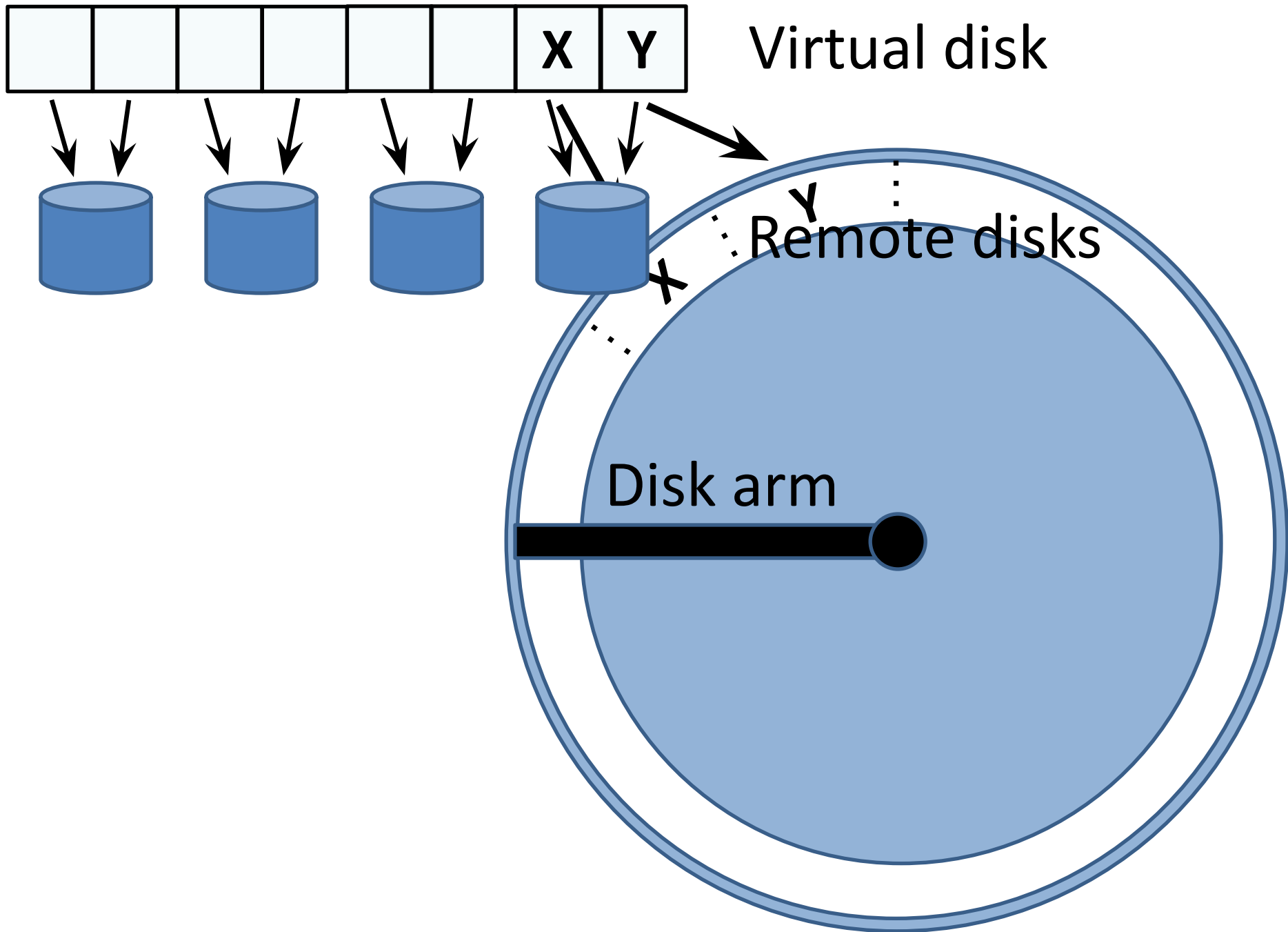


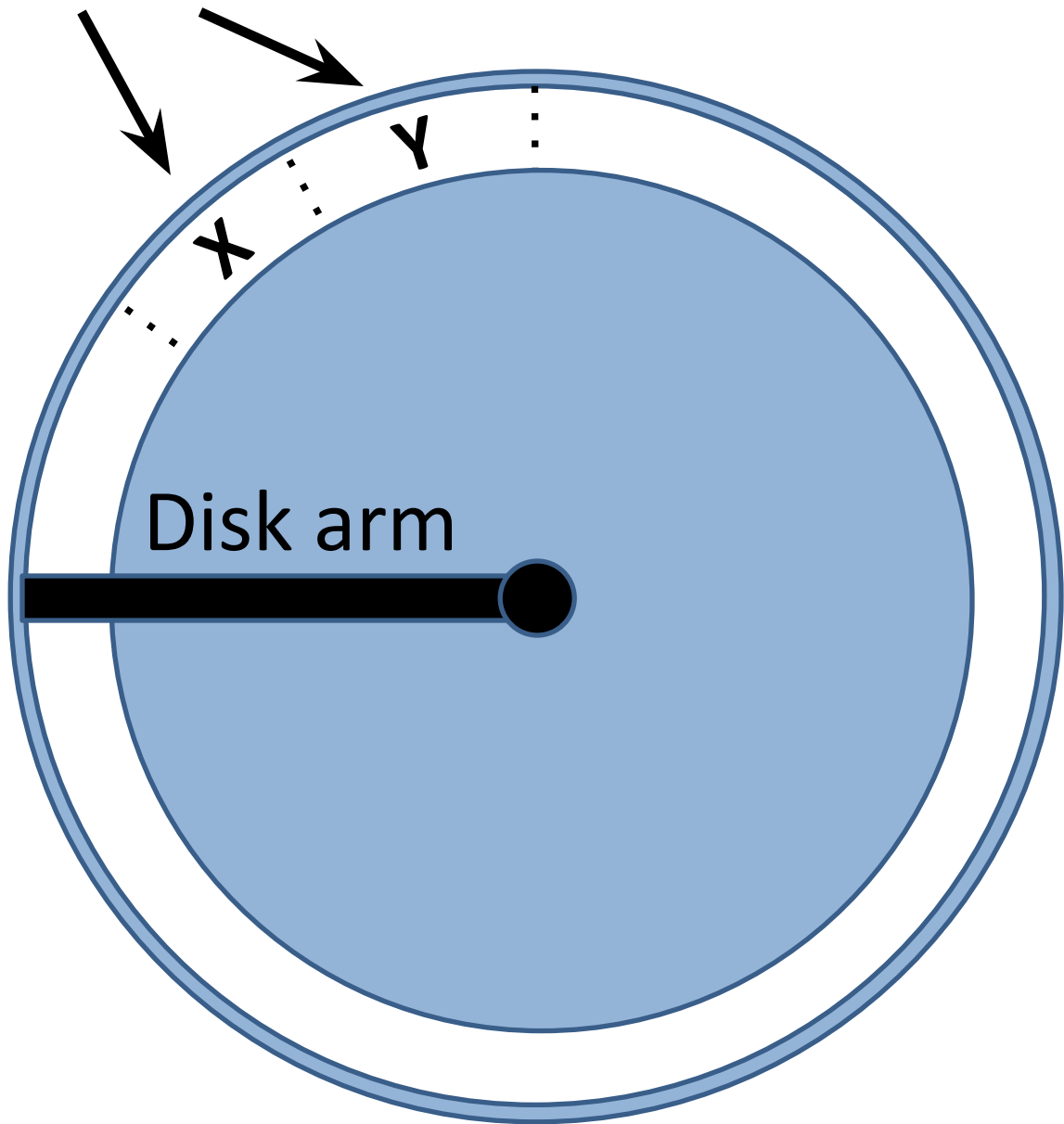
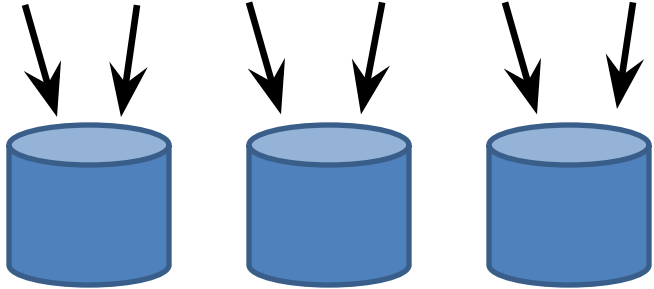


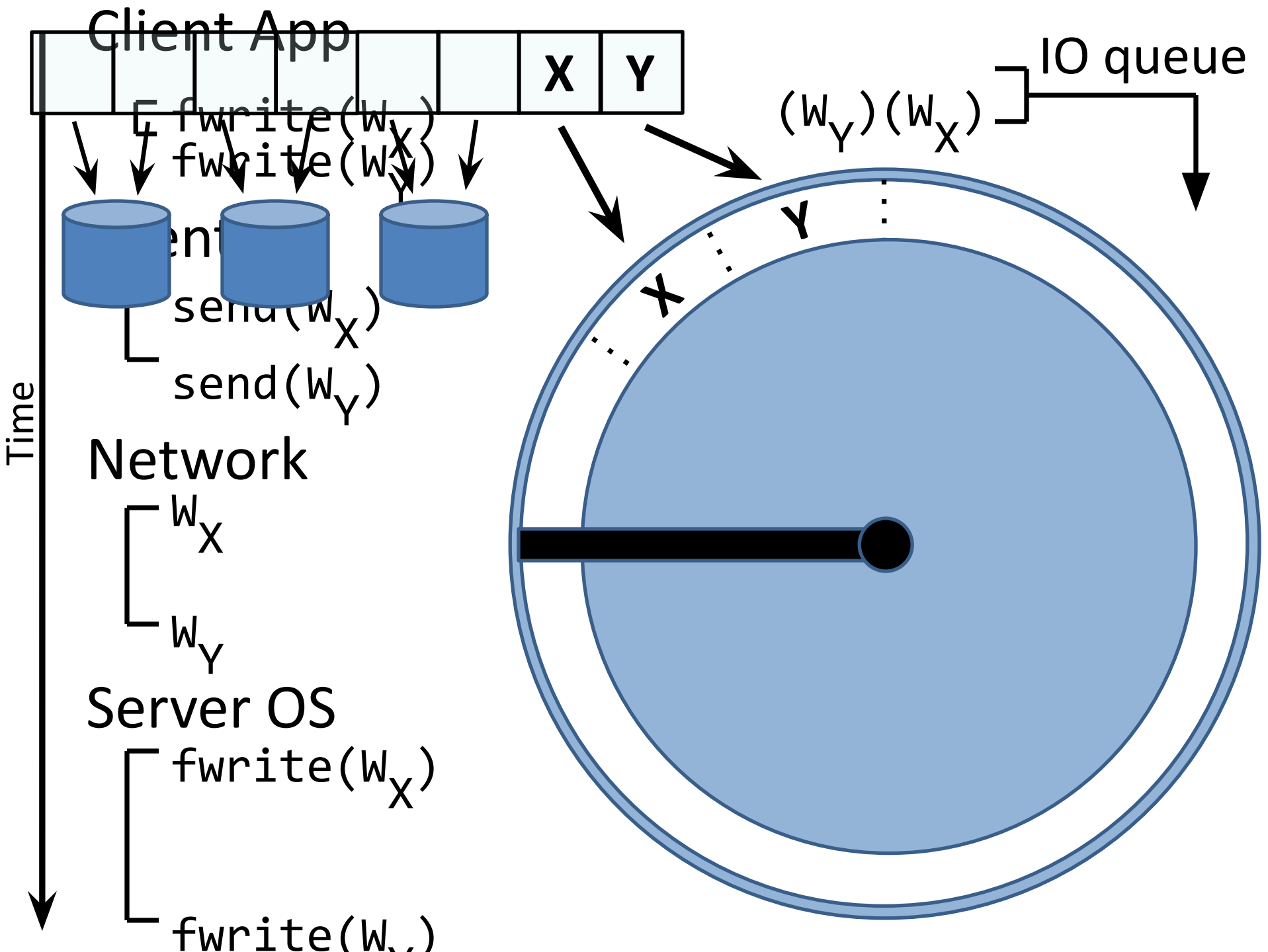
Virtual disk

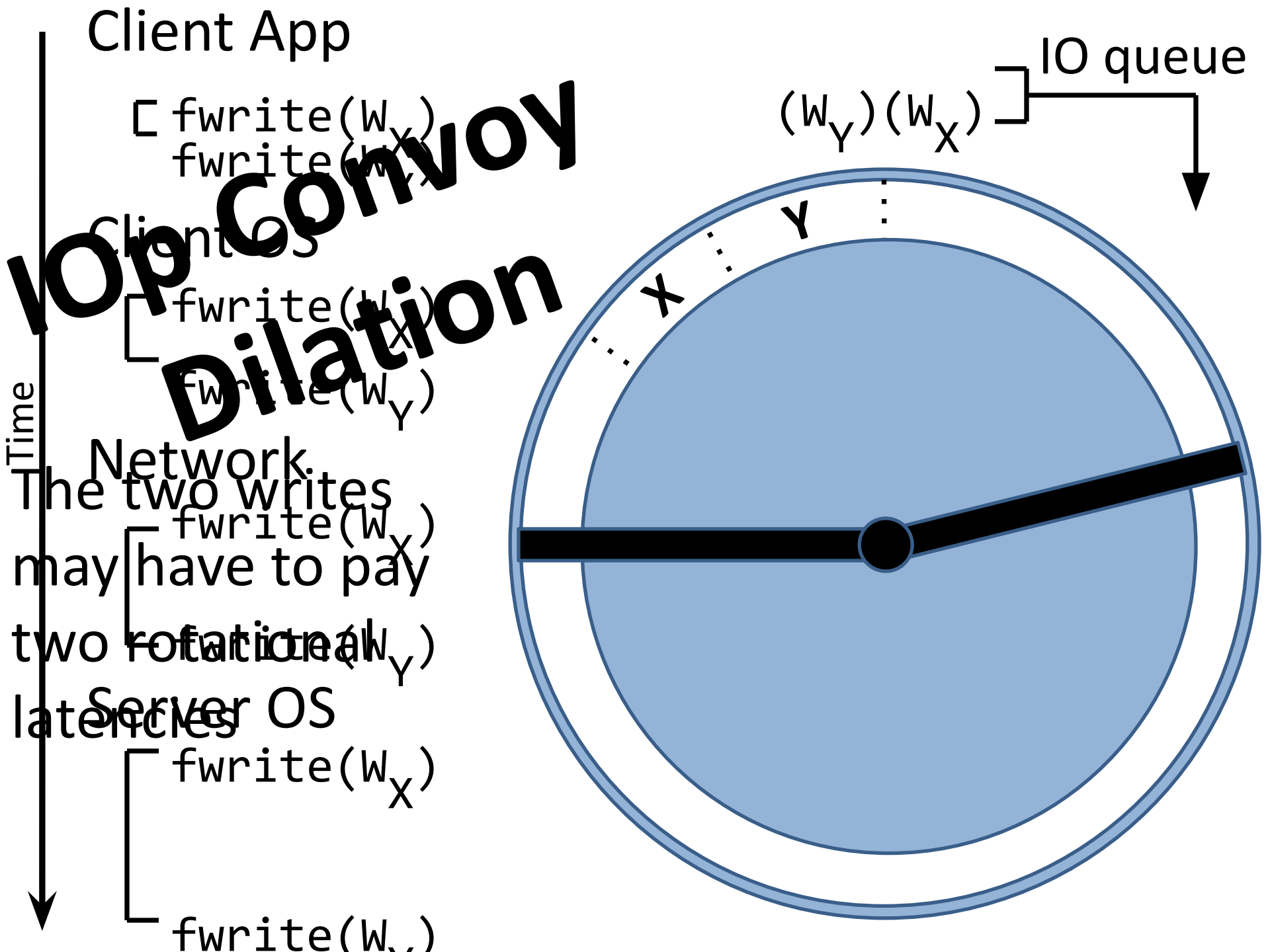


Remote disks

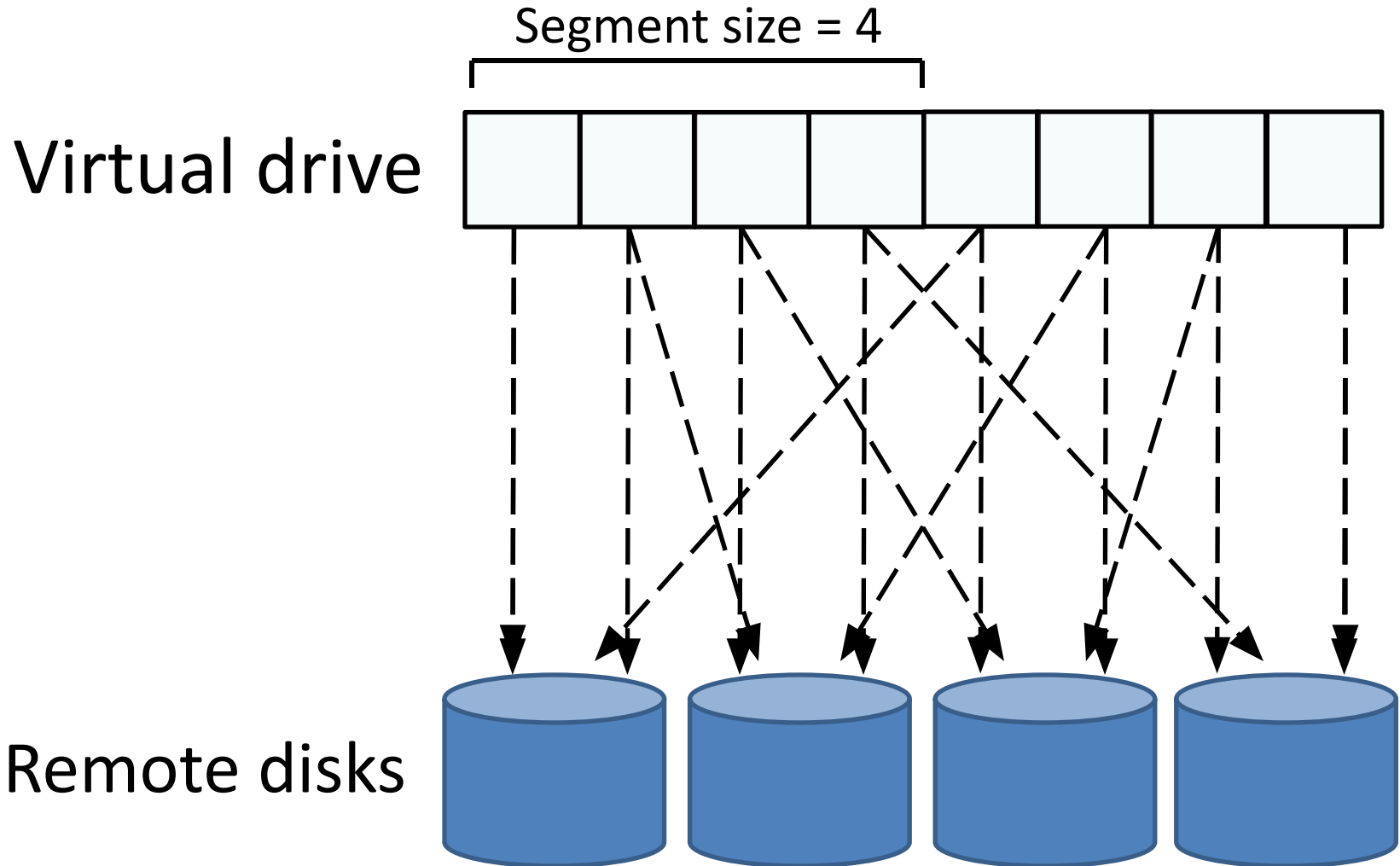




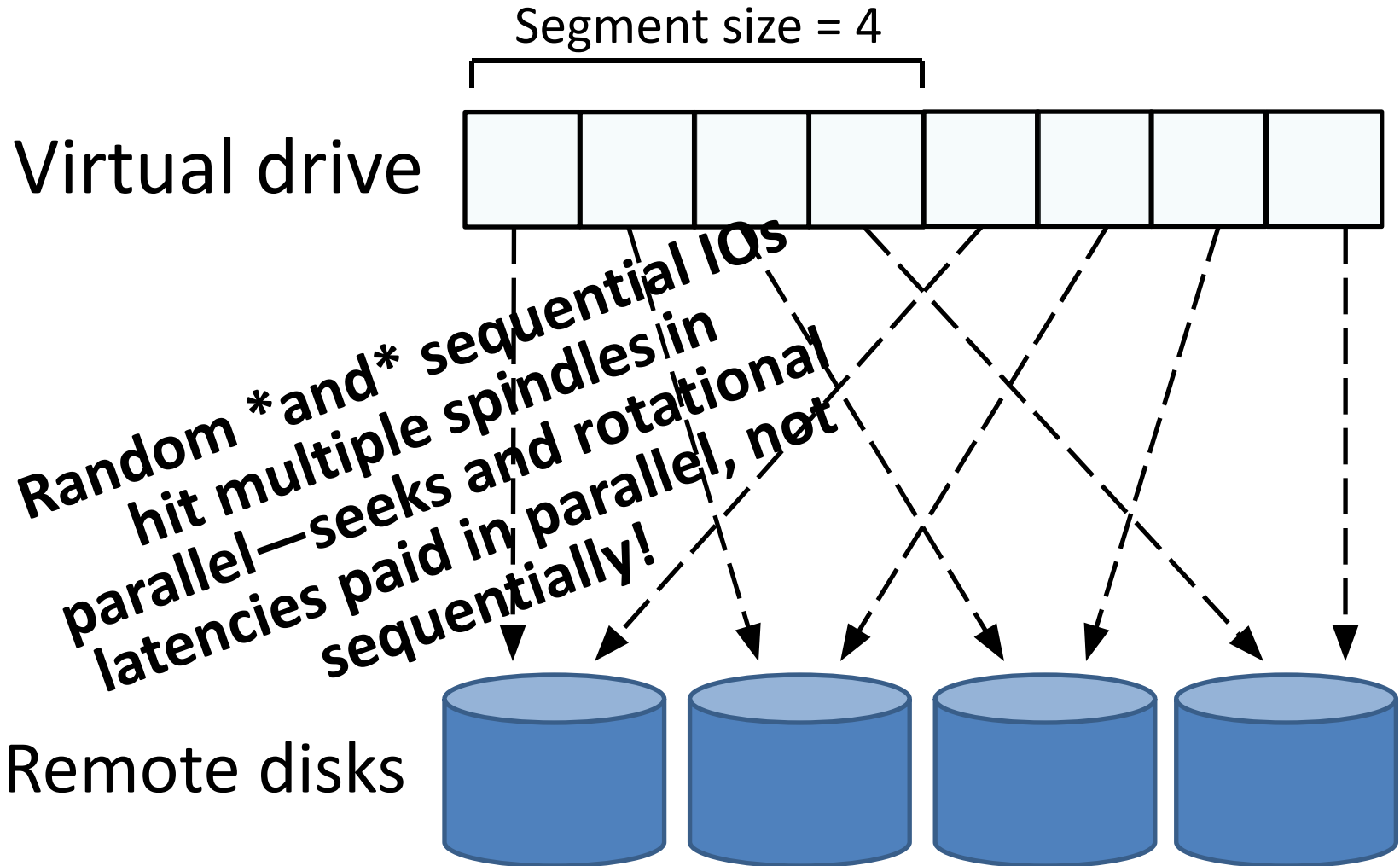




# Fixing IOp Convoy Dilation

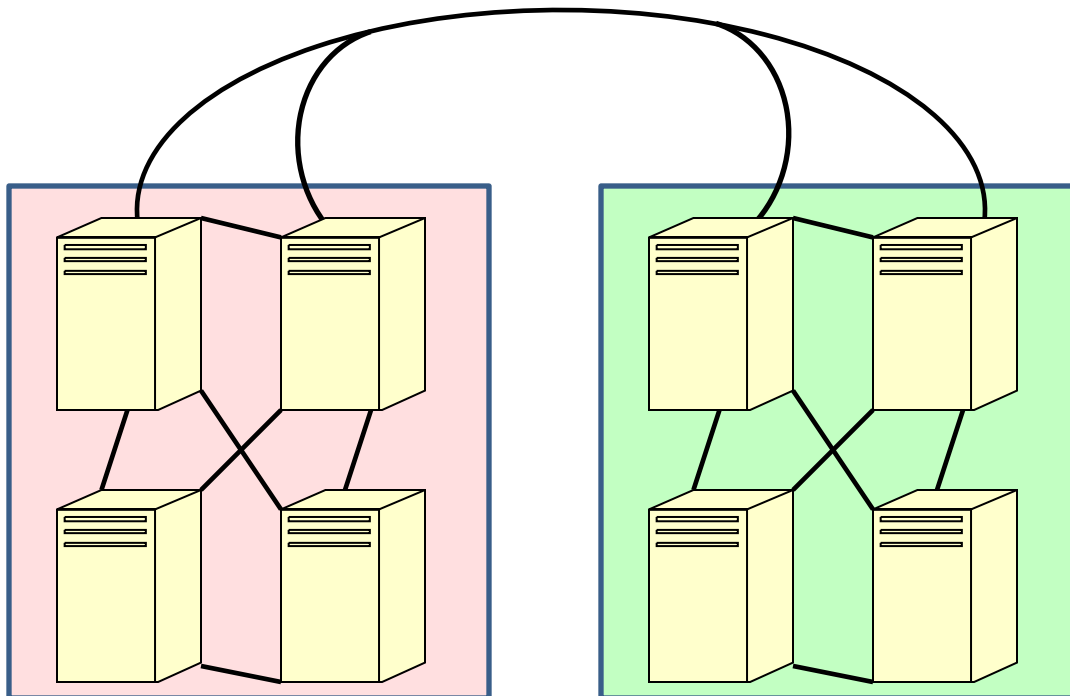


# Fixing IOp Convoy Dilation



# Rack Locality

20 Gbps  
cross-rack

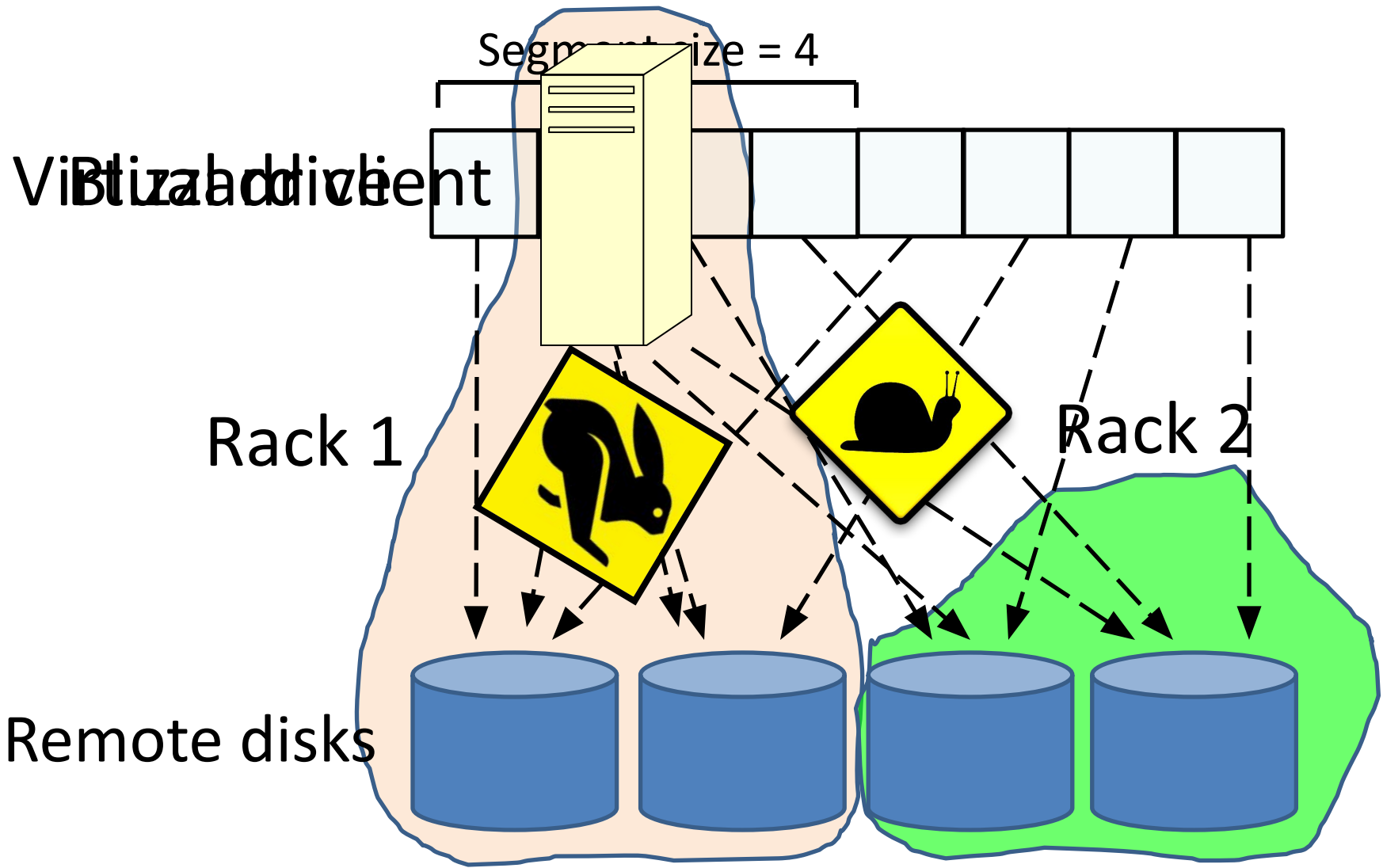


10 Gbps to all  
rack peers

10 Gbps to all  
rack peers



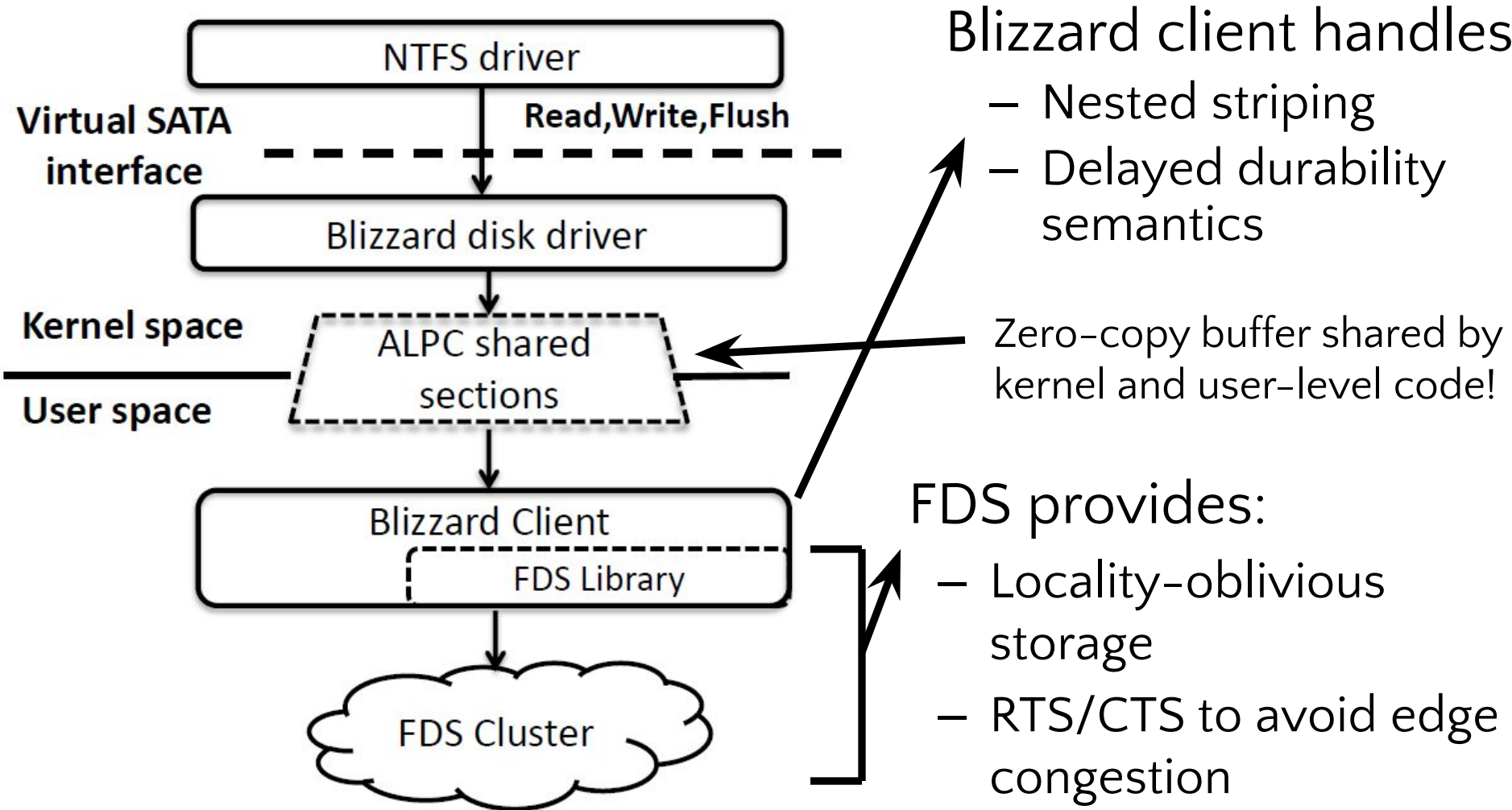
# Rack Locality In A Datacenter



# Flat Datacenter Storage (FDS)

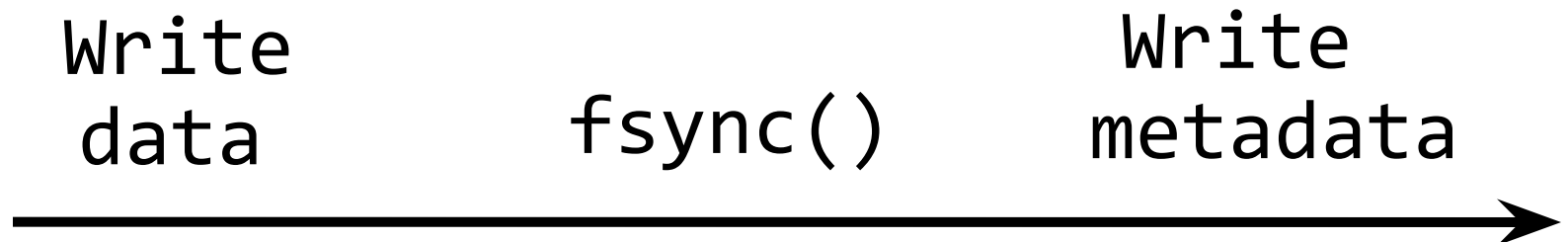
- Idea 1: Build a datacenter network with full-bisection bandwidth (i.e., no oversubscription)
  - Half of the servers can simultaneously communicate with the other half, and the network won't melt
  - In other words, the core of the network has enough bandwidth to handle  $\frac{1}{2}$  the sum of the servers' NIC speeds
- Idea 2: Give each server enough NICs to be able to read/write the server's disks at full sequential speeds
  - Ex: If one disk has sequential r/w bandwidth of 128 MB, and a server has 10 disks, give the server  $10 \times 128 \text{ MB} = 10 \text{ Gbps}$  NIC
- Result: Locality-oblivious remote storage
  - Any server can access any disk as fast as if the disk were local (assuming datacenter RTTs  $\ll$  than seek+rotational delays)
  - FDS is useful for big data applications like MapReduce too!

# Blizzard as FDS Client



# The problem with fsync()

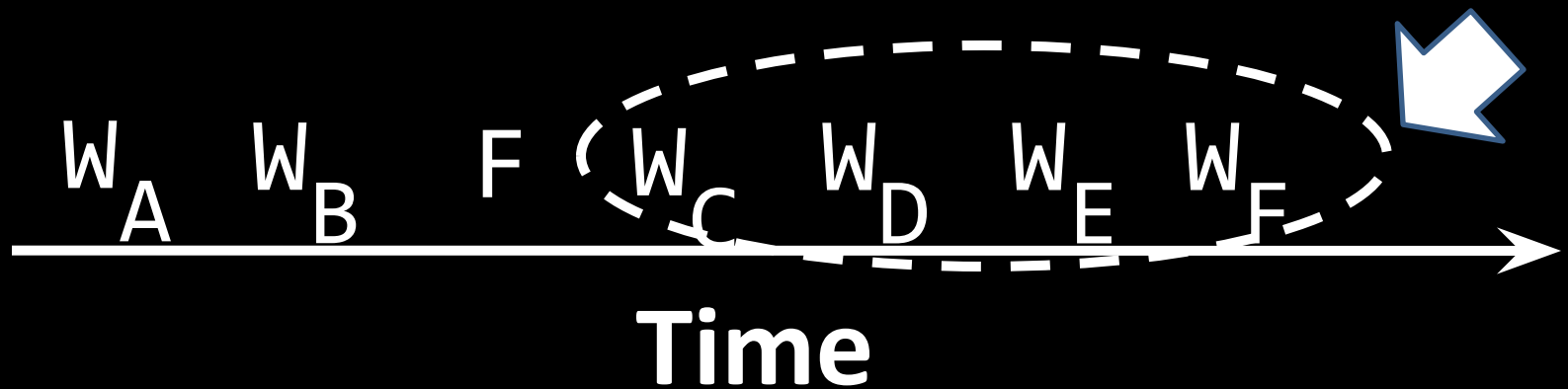
- Used by POSIX/Win32 file systems and applications to implement crash consistency
  - On-disk write buffers let the disk acknowledge a write quickly, even if the write data has not been written to a platter!
  - In addition to supporting read() and write(), the disk also implements flush()
    - The flush() command only finishes when all writes issued prior to the flush() have hit a platter
  - fsync() system call allows user-level code to ask the OS to issue a flush()
  - Ex: ensure data is written before metadata





**WRITE BARRIERS  
RUIN  
BIRTHDAYS**

**Stalled operations  
limit parallelism!**

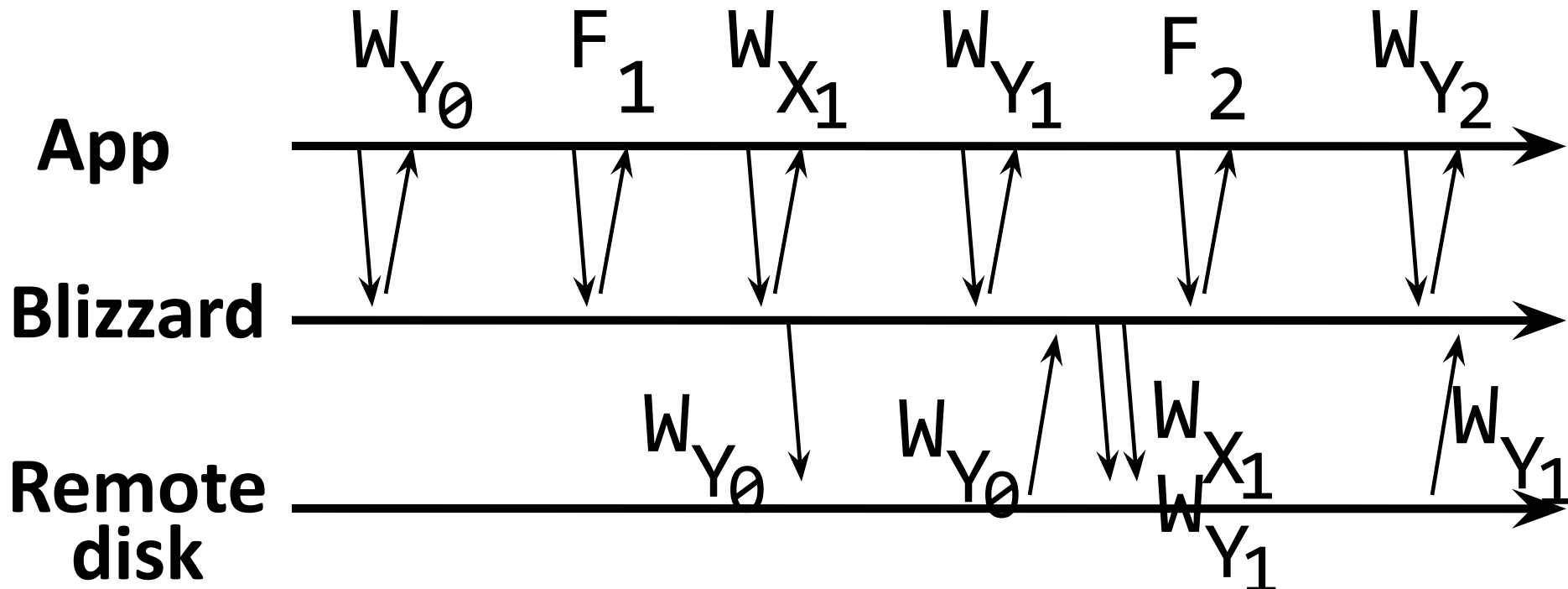


# Delayed Durability in Blizzard's Virtual Drive

- Decouple durability from ordering
- Acknowledge flush() immediately . . .
  - . . . but increment flush epoch
  - Tag writes with their epoch number, asynchronously retire writes in epoch order

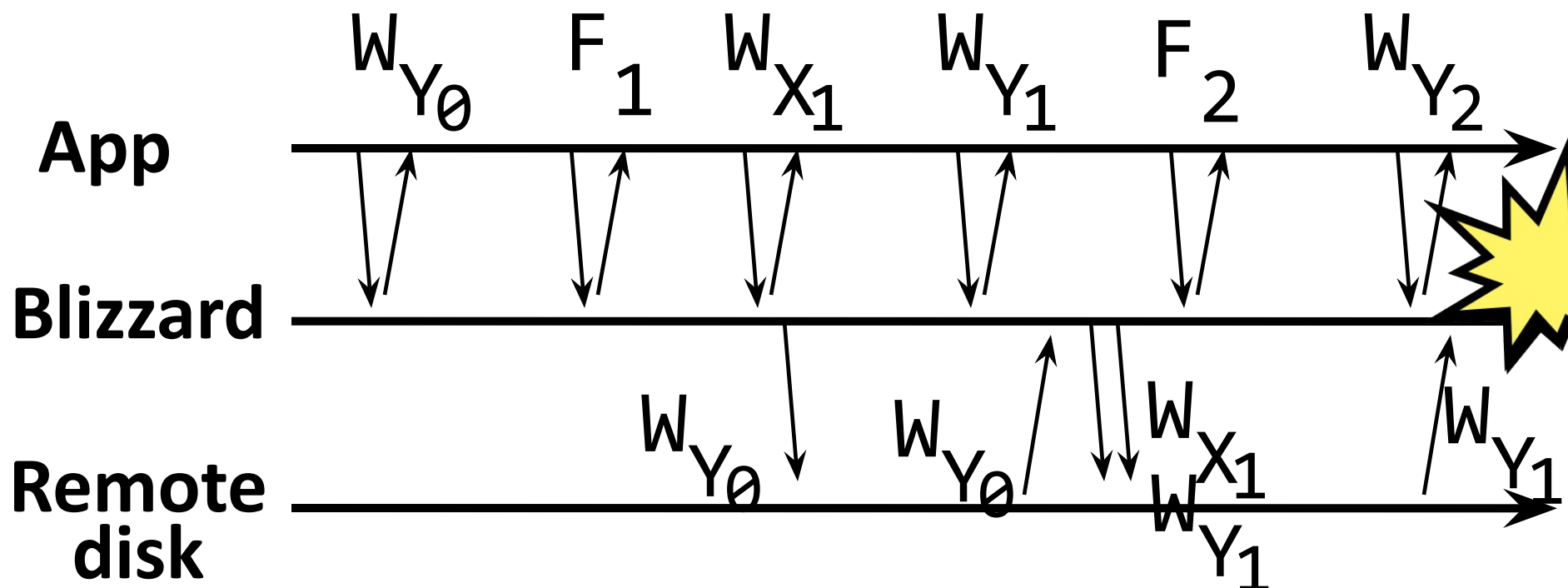
# Delayed Durability in Blizzard's Virtual Drive

- Decouple durability from ordering
- Acknowledge flush() immediately ...
  - ... but increment flush epoch
  - Tag writes with their epoch number, asynchronously retire writes in epoch order



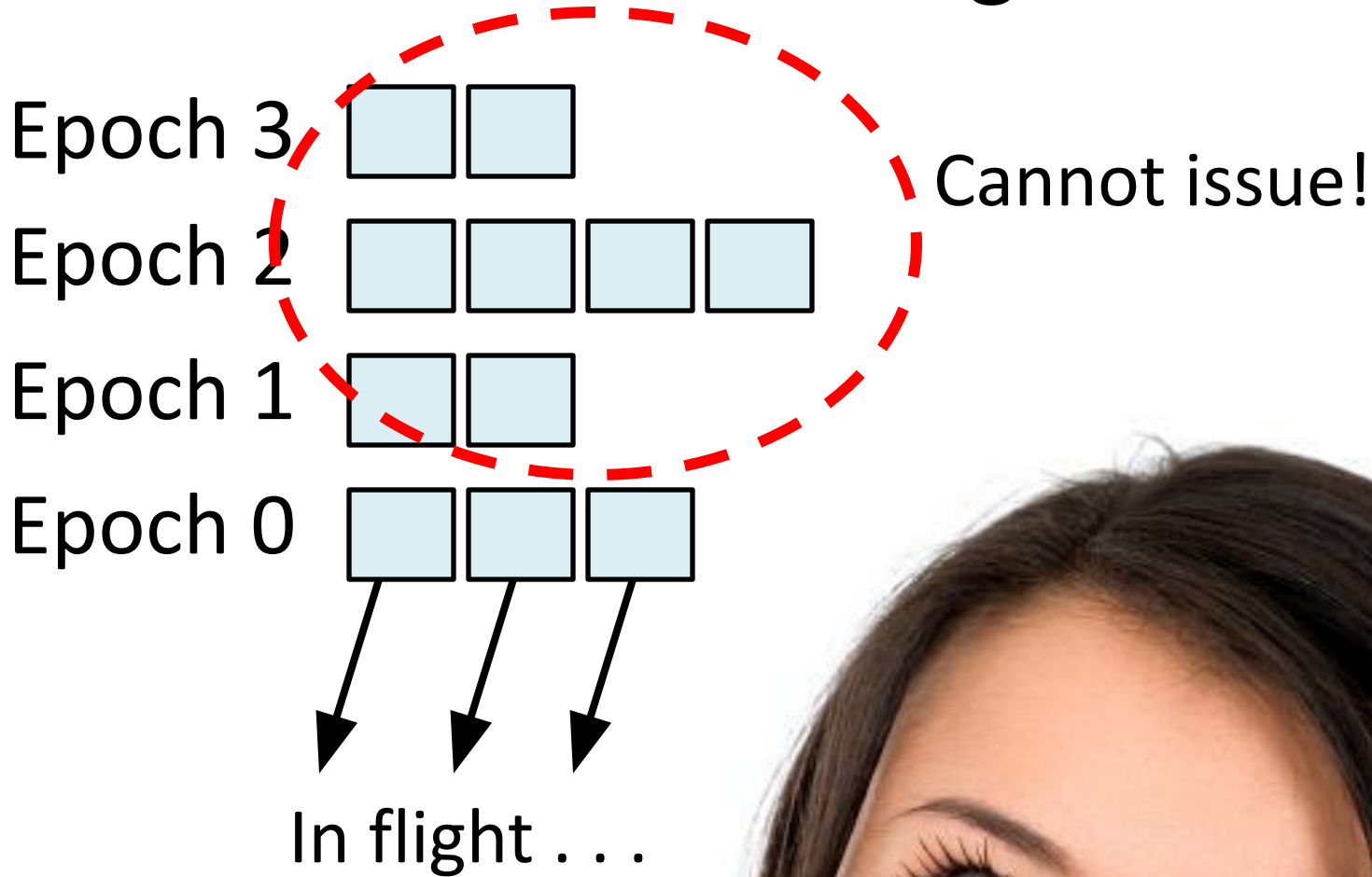
# Delayed Durability in Blizzard's Virtual Drive

- All writes are acknowledged
- ... but only  $W_Y$  and  $W_Y$  are durable!
- Satisfies prefix consistency ordering
  - All epochs up to  $N-1$  are durable.
- Acknowledge `flush()` immediately
  - Some, all, or no writes from epoch  $N$  are durable
  - No writes from later epochs are durable
- Prefix consistency is good enough for most apps, provides much better performance
  - asynchronously retire writes in epoch order



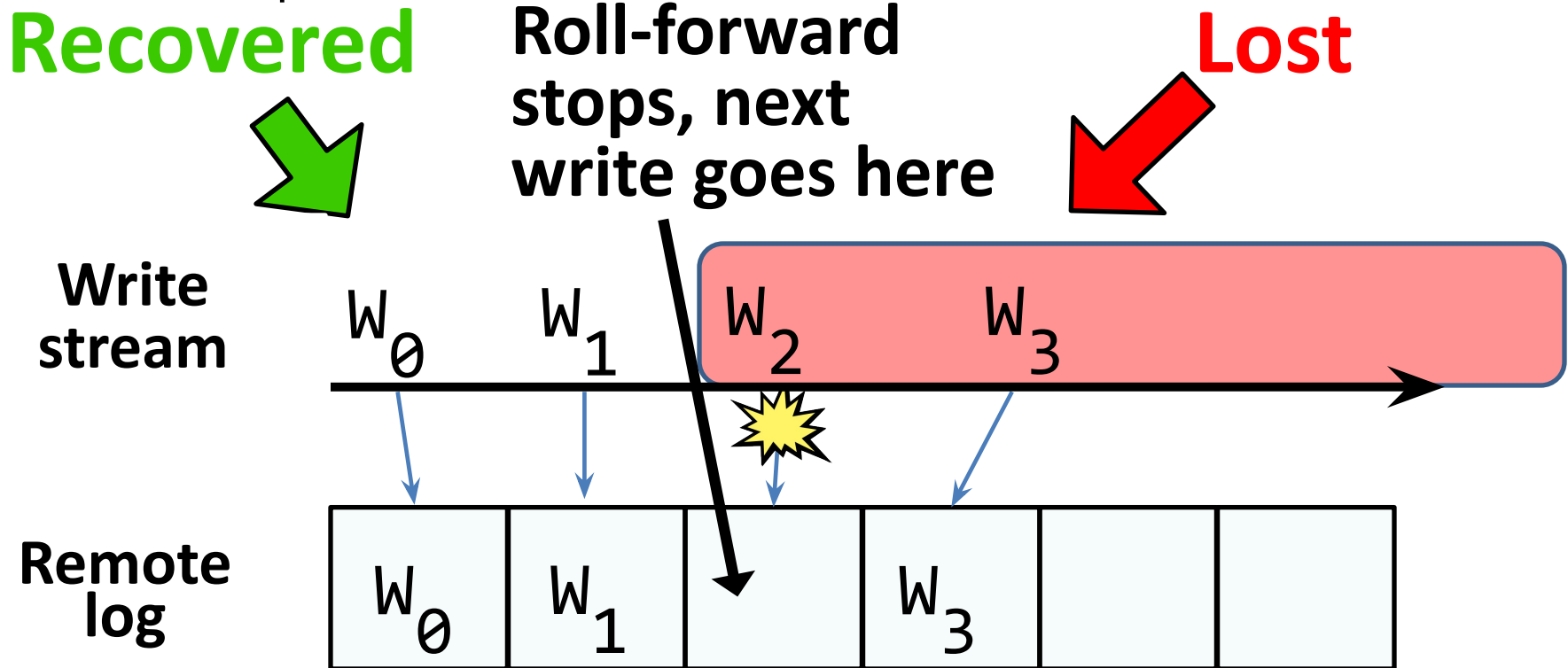


# Isn't Blizzard buffering a lot of data?

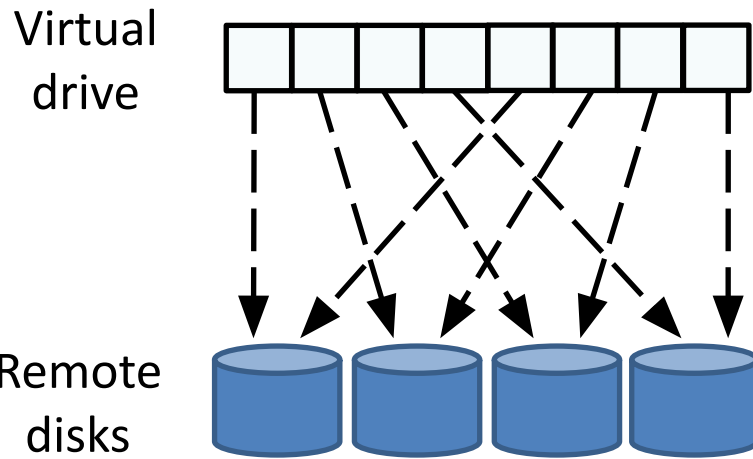


# Log-based Writes

- Treat backing FDS storage as a distributed log
  - Issue block writes to log immediately and in order
  - Blizzard maintains a mapping from logical virtual disk blocks to their physical location in the log
  - On failure, roll forward from last checkpoint and stop when you find torn write, unallocated log block with old epoch number



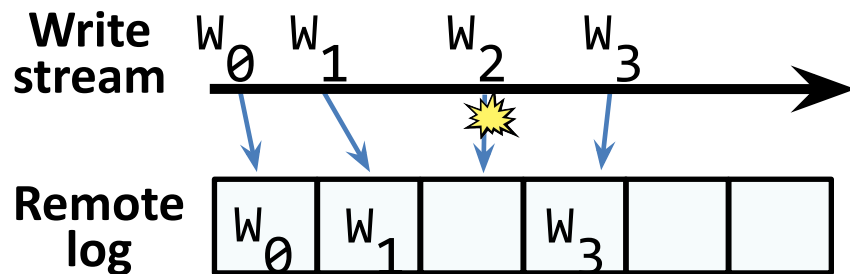
# Summary of Blizzard's Design



- Problem: I/O Dilation
- Solution: Nested striping

**FDS**

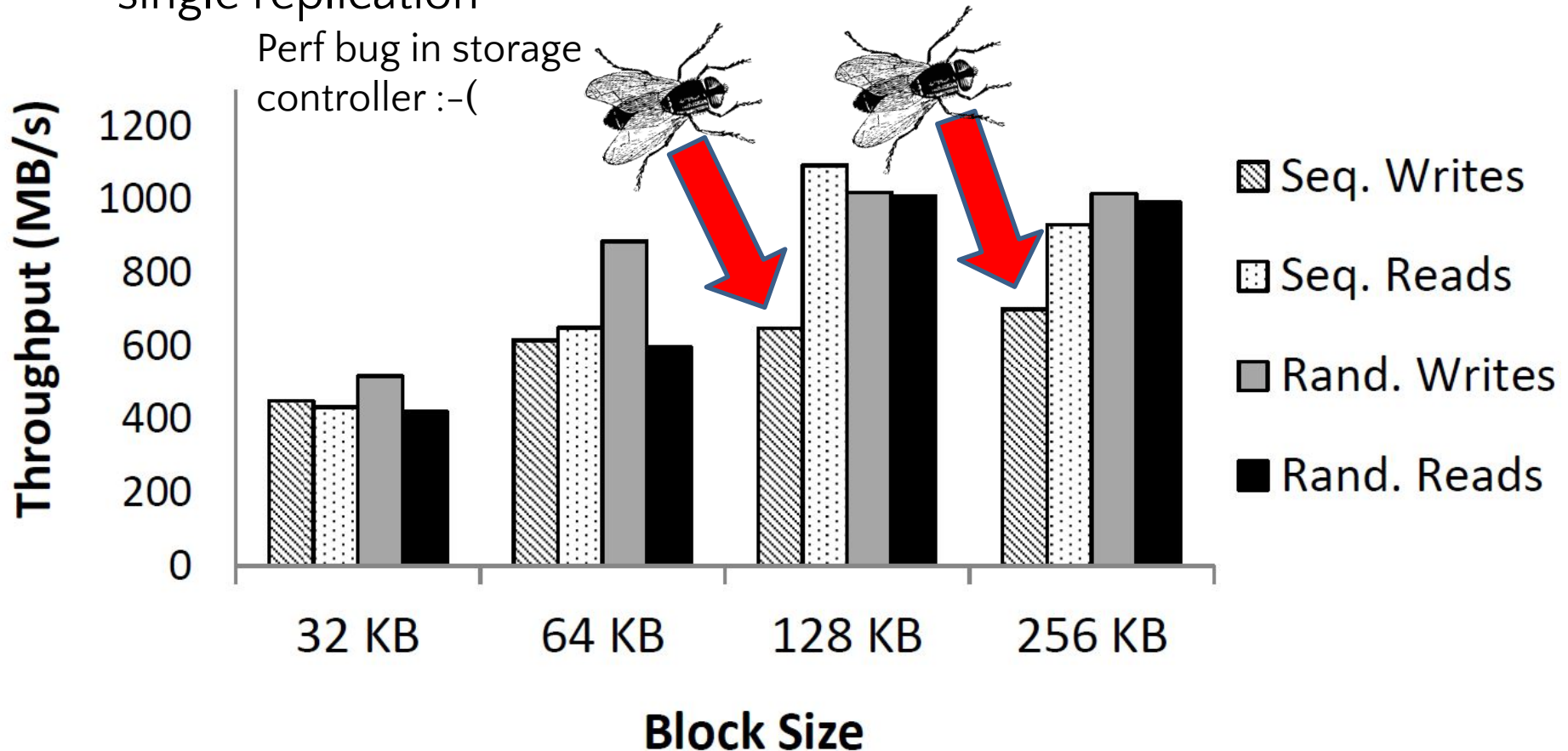
- Problem: Rack locality constrains parallelism
- Solution: Full-bisection networks, match disk and network bandwidth



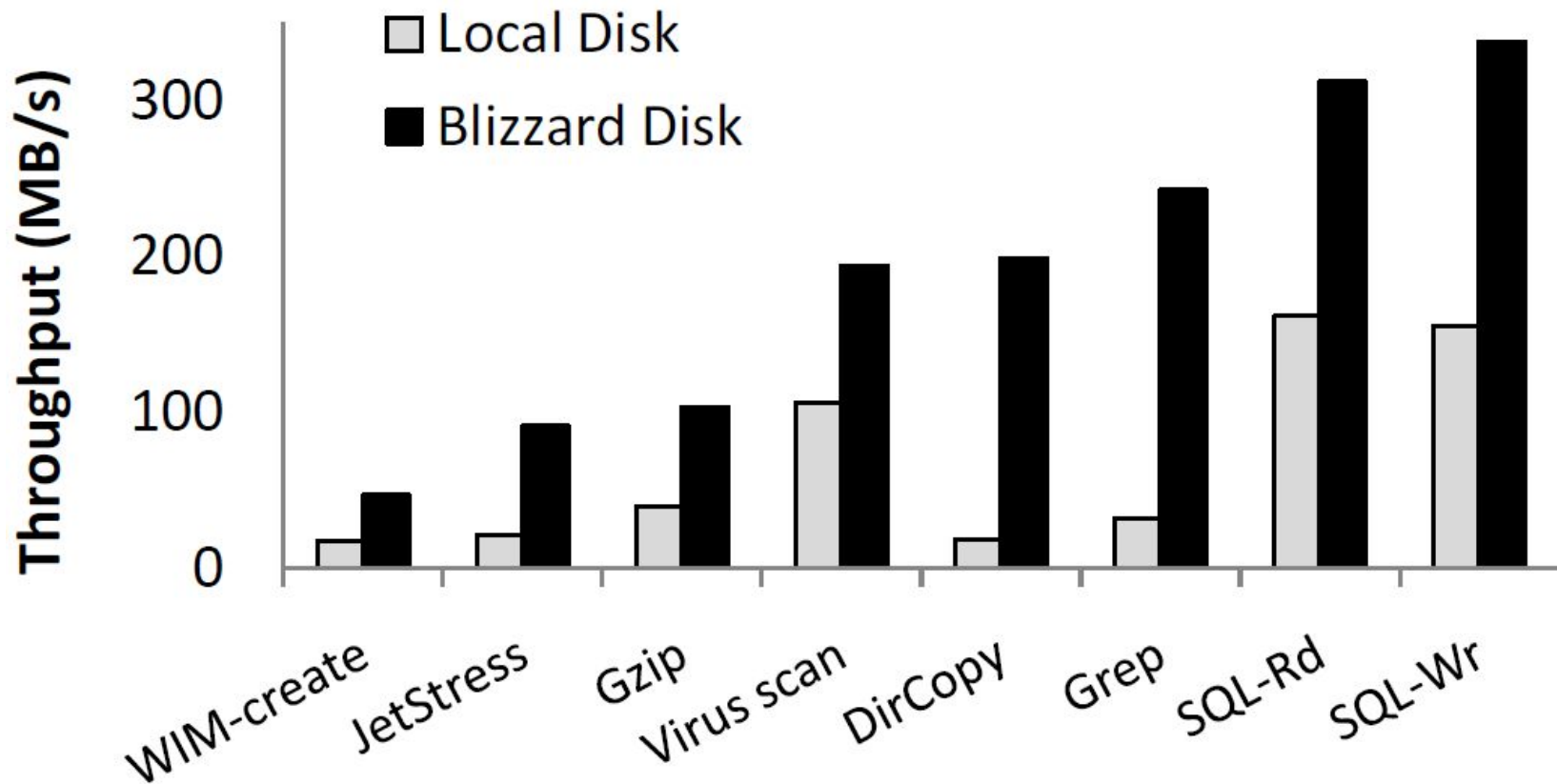
- Problem: Evil `fsync()`s
- Solution: Delayed durability (note that the log is nested-striped)

# Throughput Microbenchmark

- Application issues a bunch of parallel reads or writes
  - In this experiment, we use nested striping but synchronous write-through (i.e., no delayed durability tricks—a write does not complete until it is persistent)
  - Blizzard virtual disk backed by 128 remote physical disks, and used single replication



# Application Macrobenchmarks (Write-through, Single Replication)



# Delayed Durability: Hiding Replication Penalties

fa="fast acknowledgment"  
(i.e., delayed durability but  
no log-based writes)

Just as fast as fast ack, but has  
to buffer less data (and thus  
loses less data after crash)

