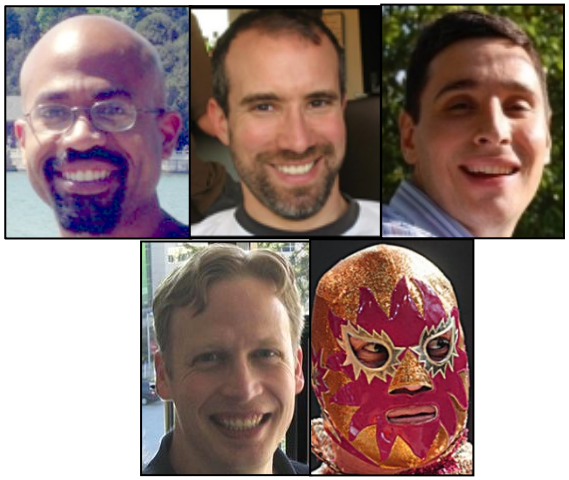
Blizzard: Fast, Cloud-scale Block Storage for Cloud-oblivious Applications

Microsoft Research



James Mickens, Jeremy Elson, Edmund B. Nightingale, Darren Gehring, Krishna Nareddy

THE UNIVERSITY of WISCONSIN MADISON

Asim Kadav
Vijay Chidambaram

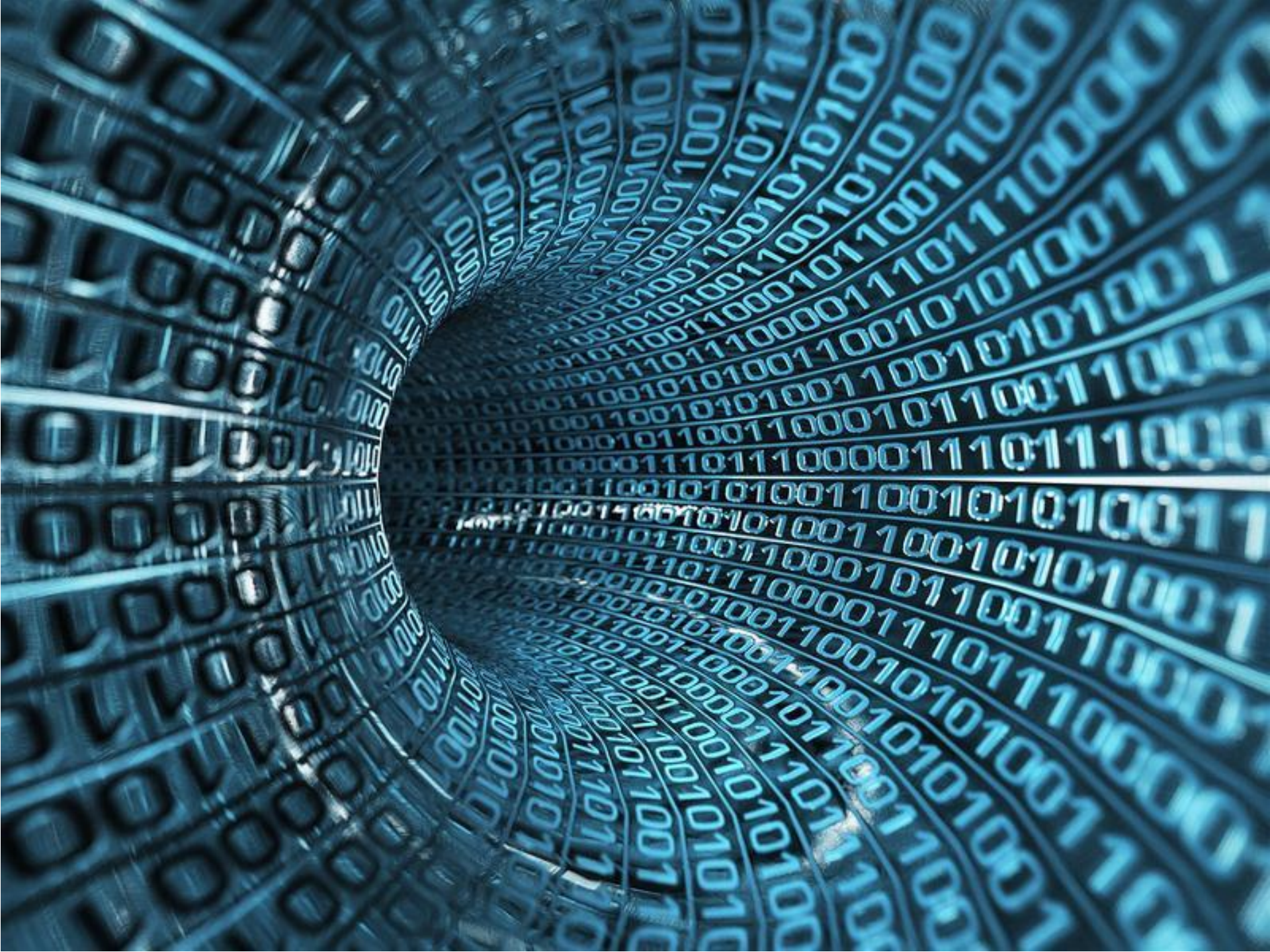
Carnegie Mellon University

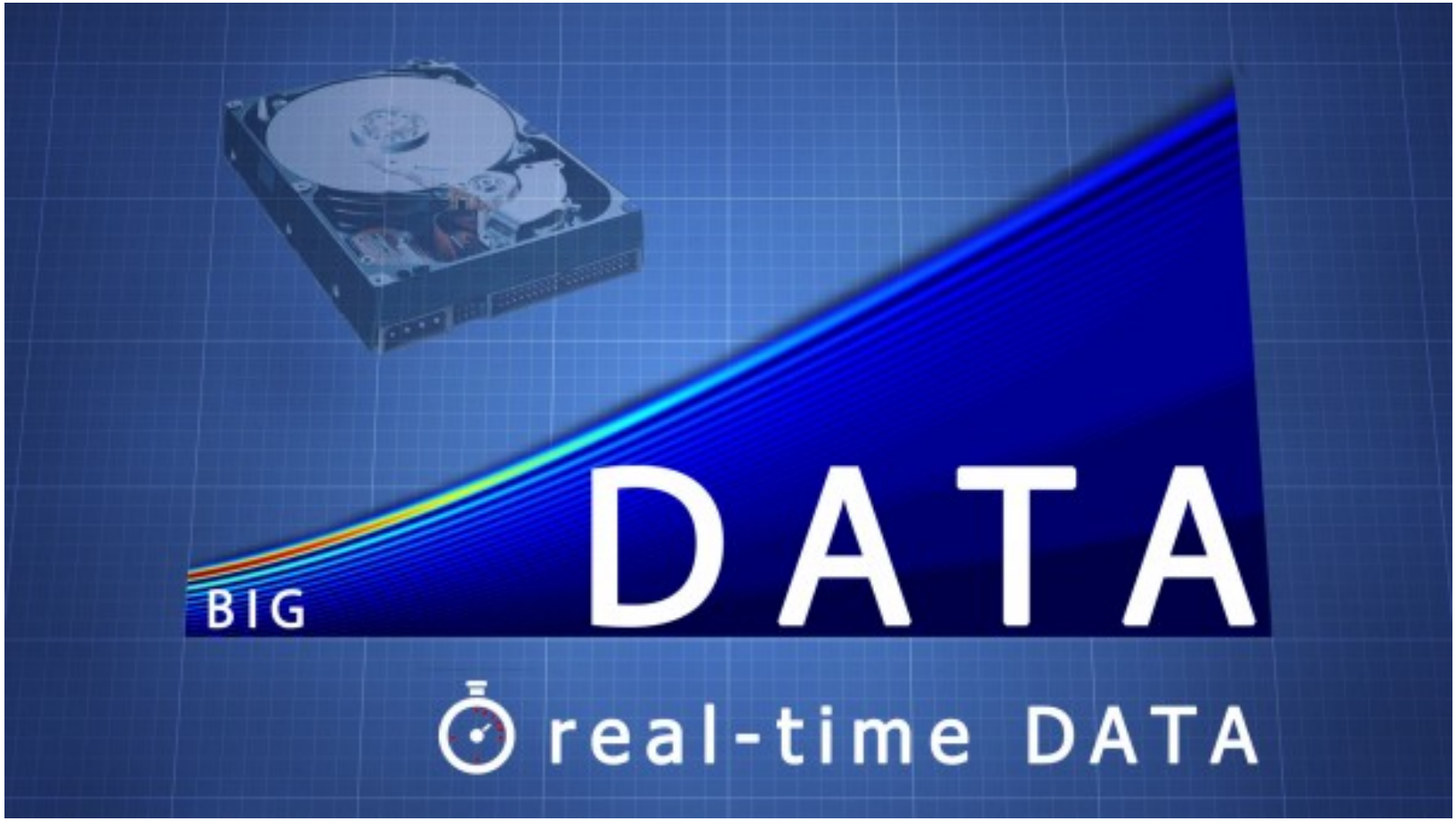
Bin Fan

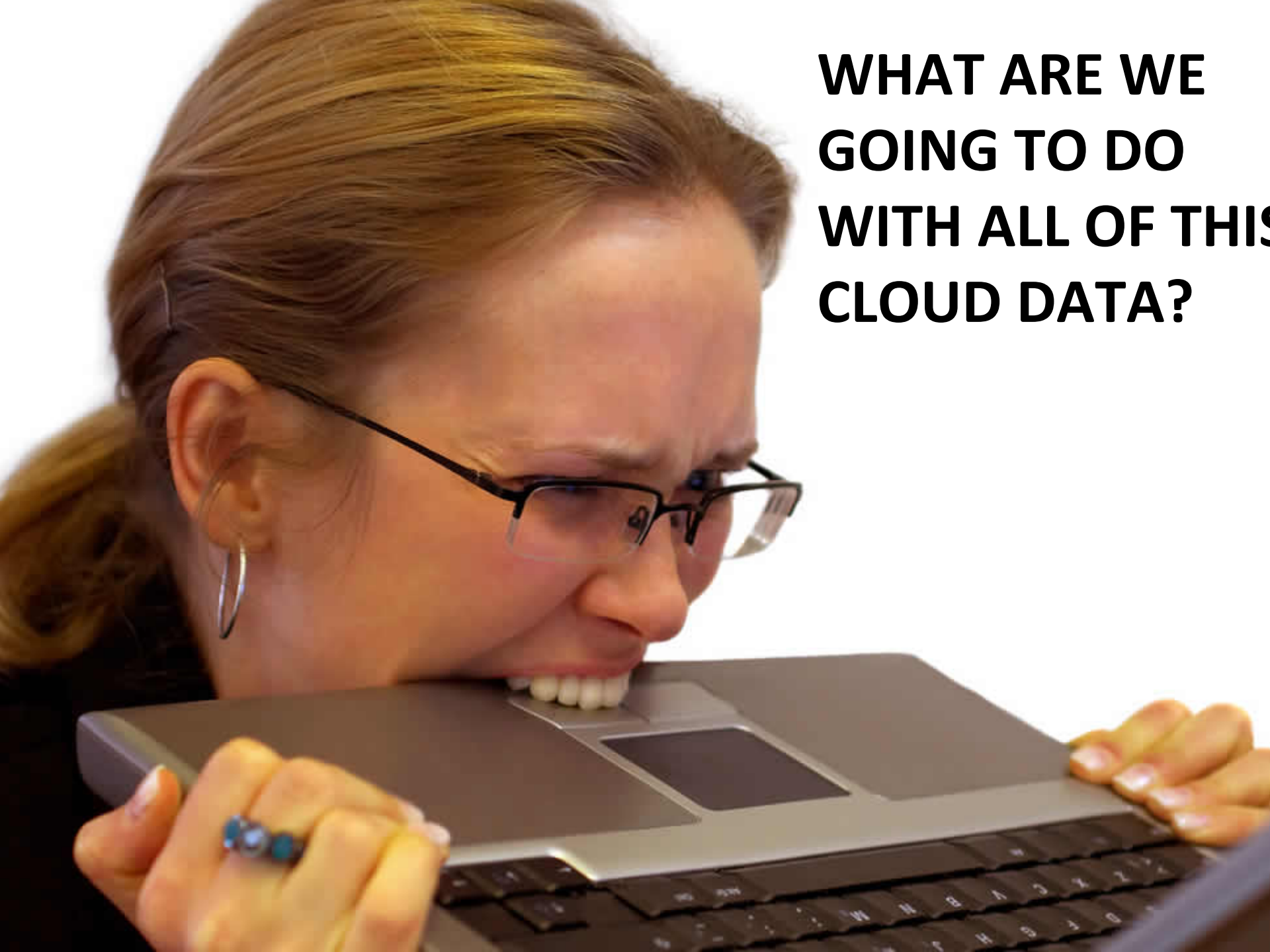
JOHNS HOPKINS

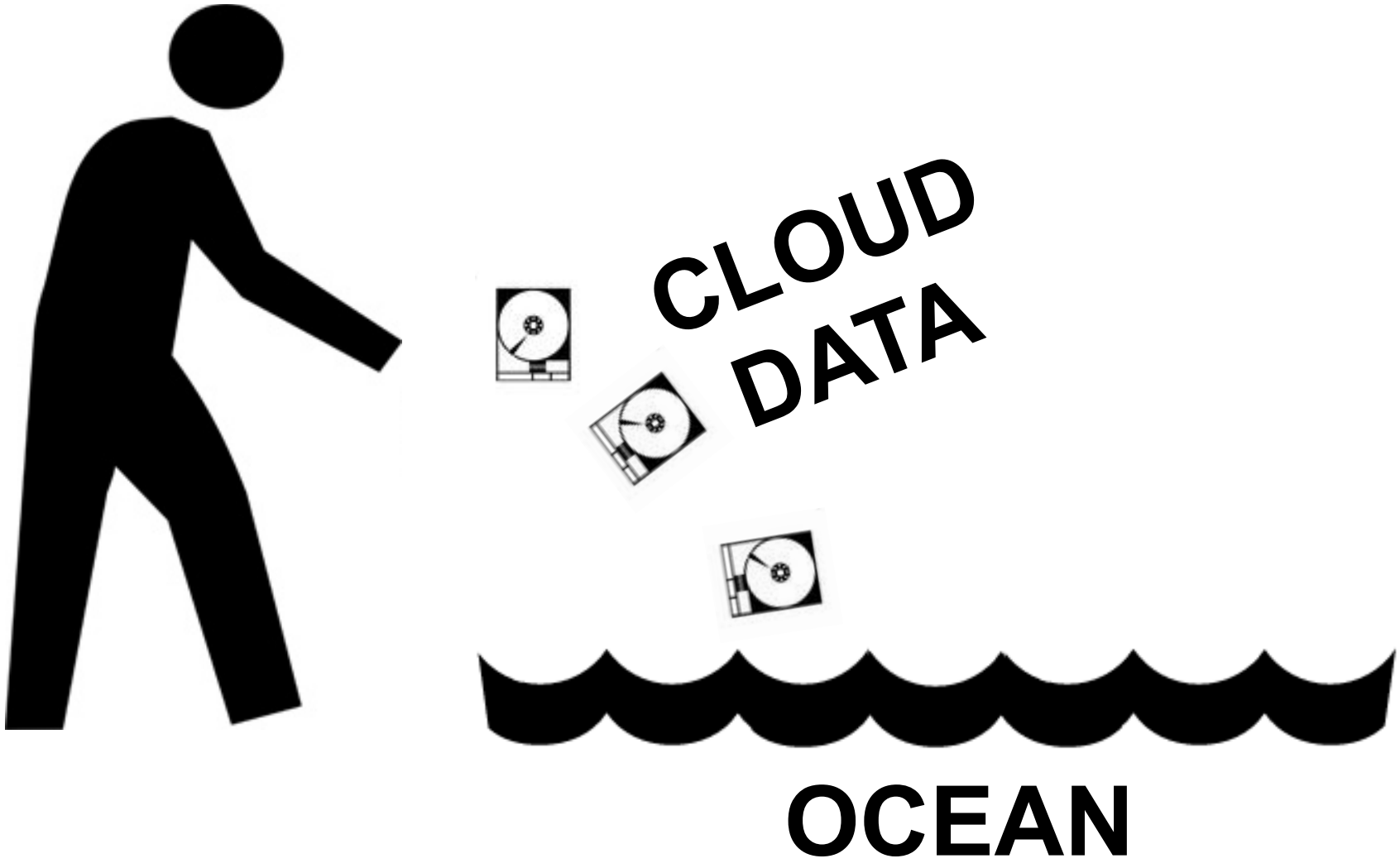
Osama Khan

IT'S, UH, THE FUTURE




BIG
DATA
real-time DATA


WHAT ARE WE
GOING TO DO
WITH ALL OF THIS
CLOUD DATA?


CLOUD
DATA
OCEAN

My Goal

- 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!

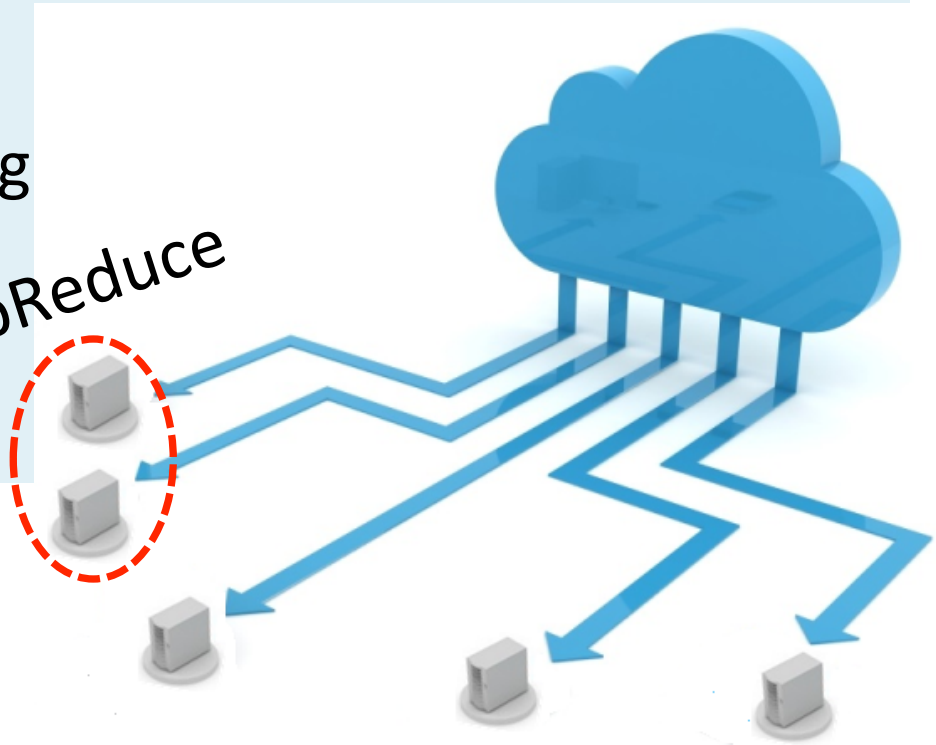
My Goal

- 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!
  - Throughput > 1000 MB/s
  - Scale-out architecture using commodity parts
  - Transparent failure recovery

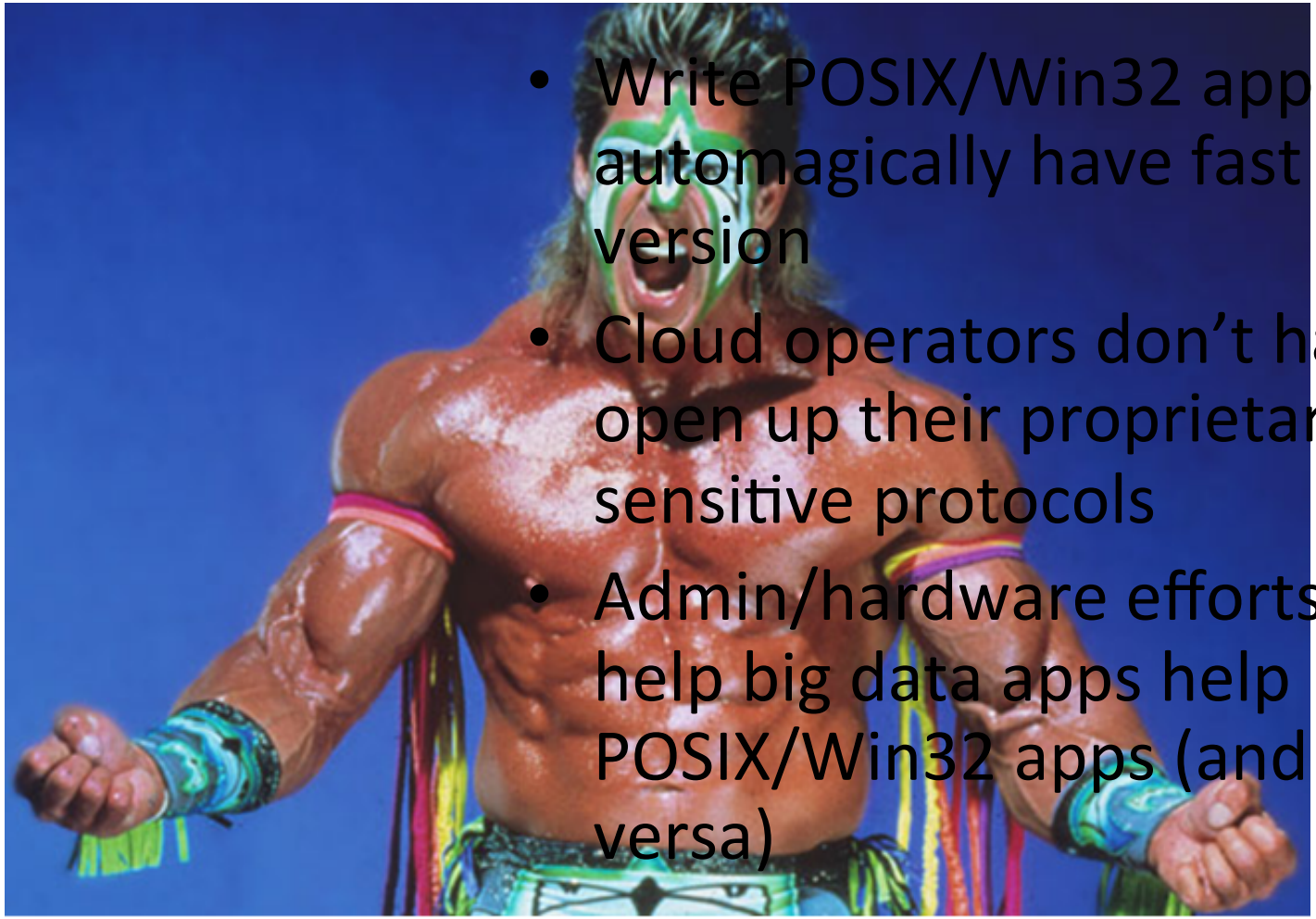
PostgreSQL

eclipse

Microsoft Exchange 2013



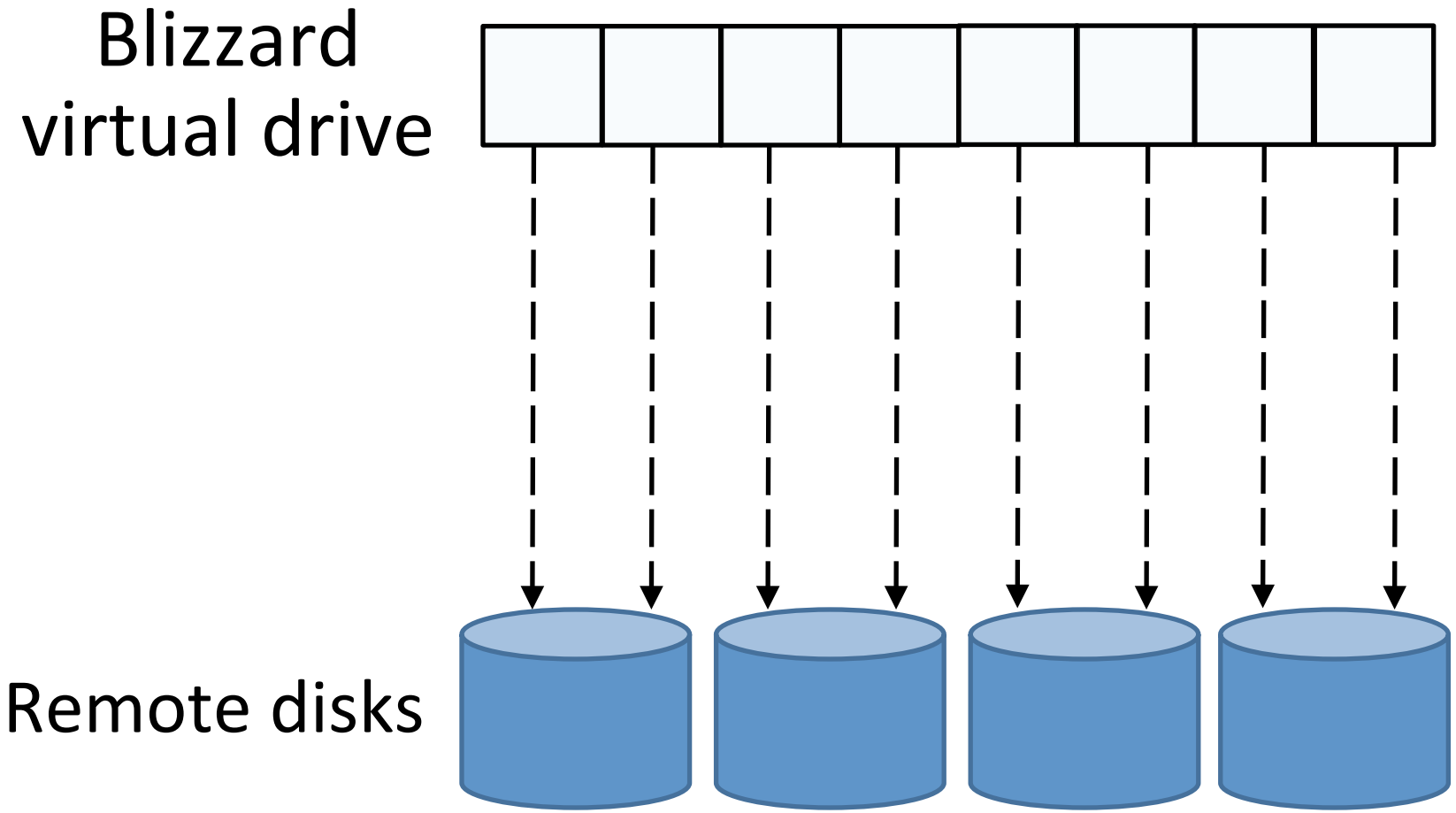
Why Do I Want To Do This?



- 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)

BECAUSE THIS WOULD BE AMAZING

Our Solution: Blizzard



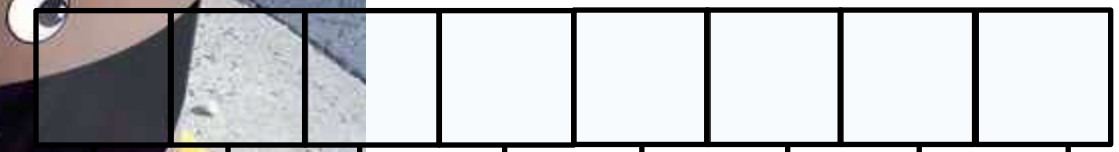
Our Solution: Blizzard

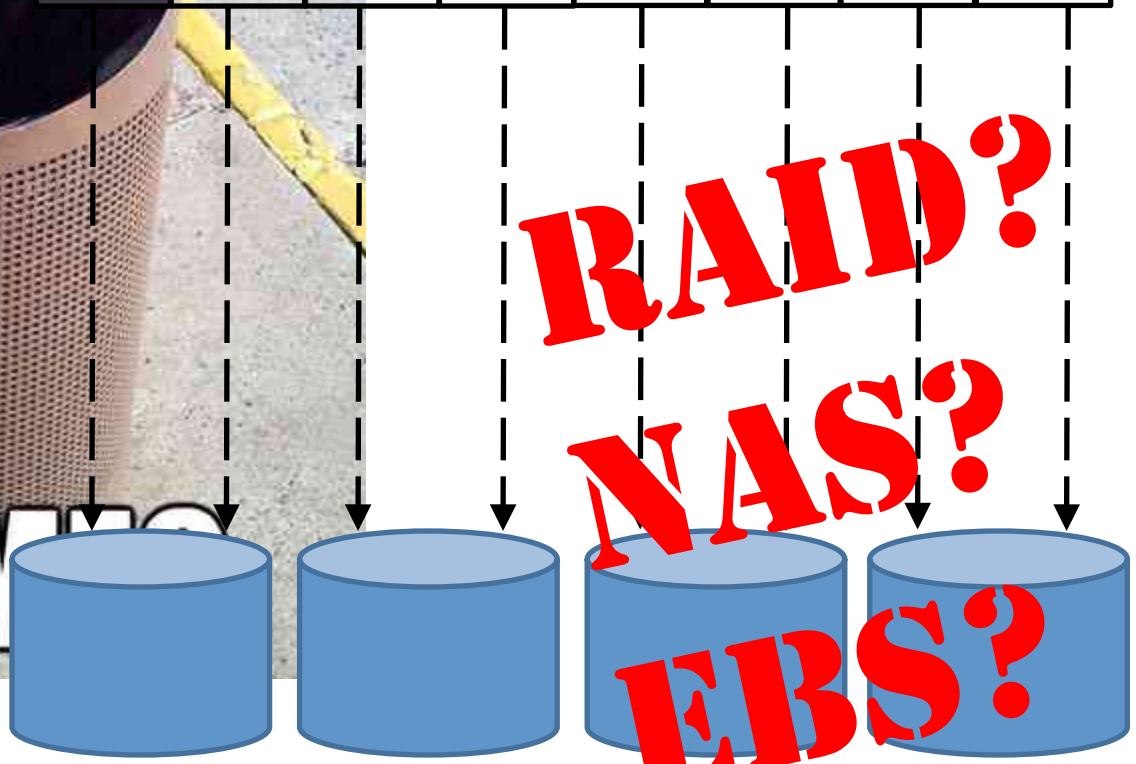
PostgreSQL

eclipse

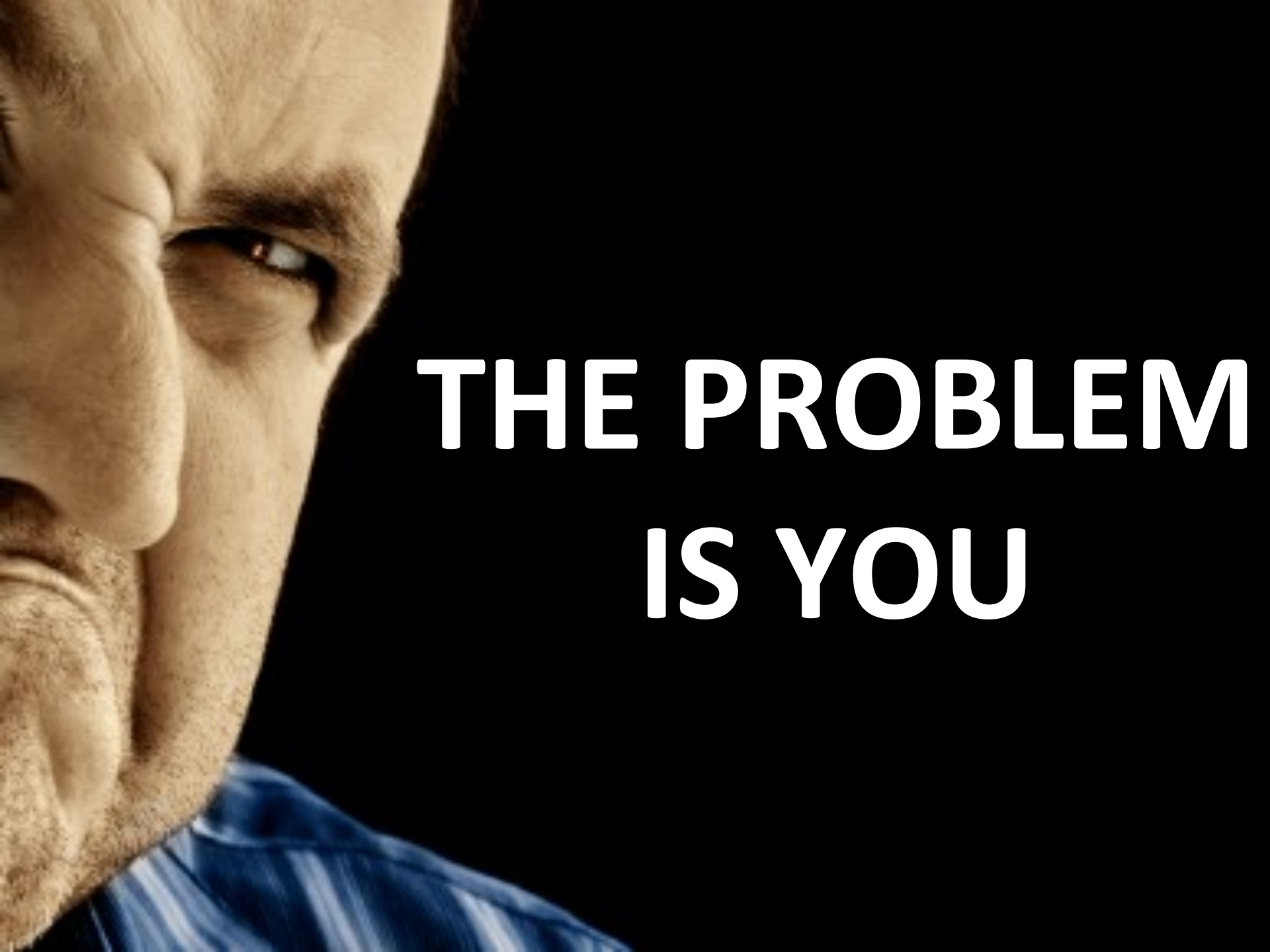
Microsoft® Exchange 2013

Blizzard virtual drive





Remote disks


THE PROBLEM
IS YOU

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

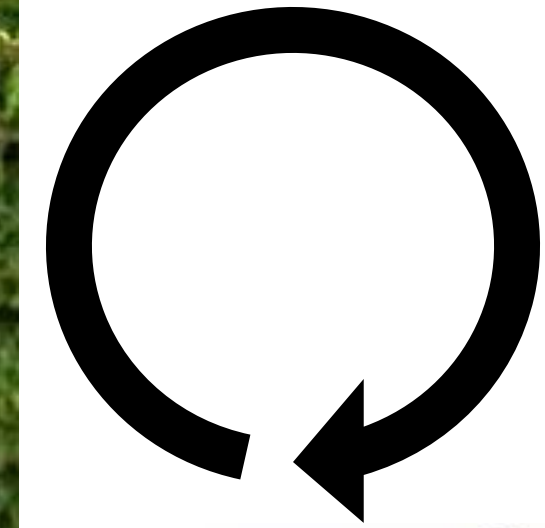
The naïve approach for implementing virtual disks does not **maximize spindle parallelism** for POSIX/Win32 applications which **frequently issue fsync() operations** to maintain consistency.

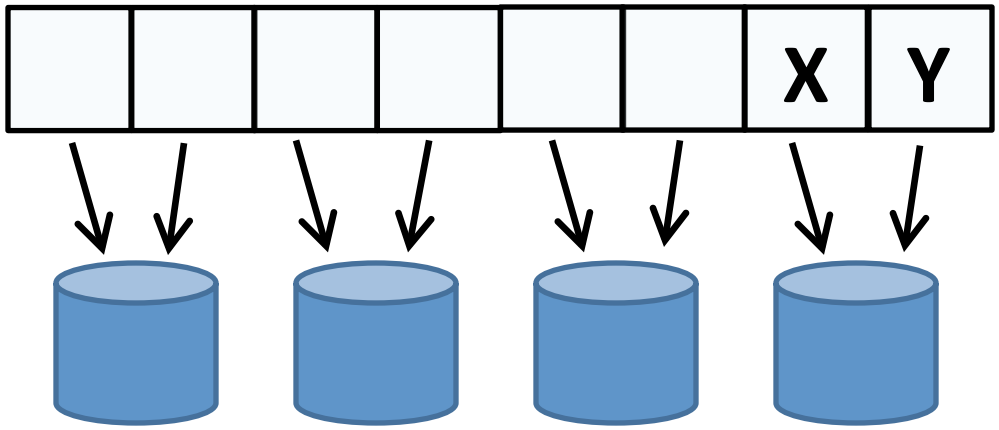
LISTEN

IOp dilation:
Nested striping

Rack locality:
Locality-oblivious
storage

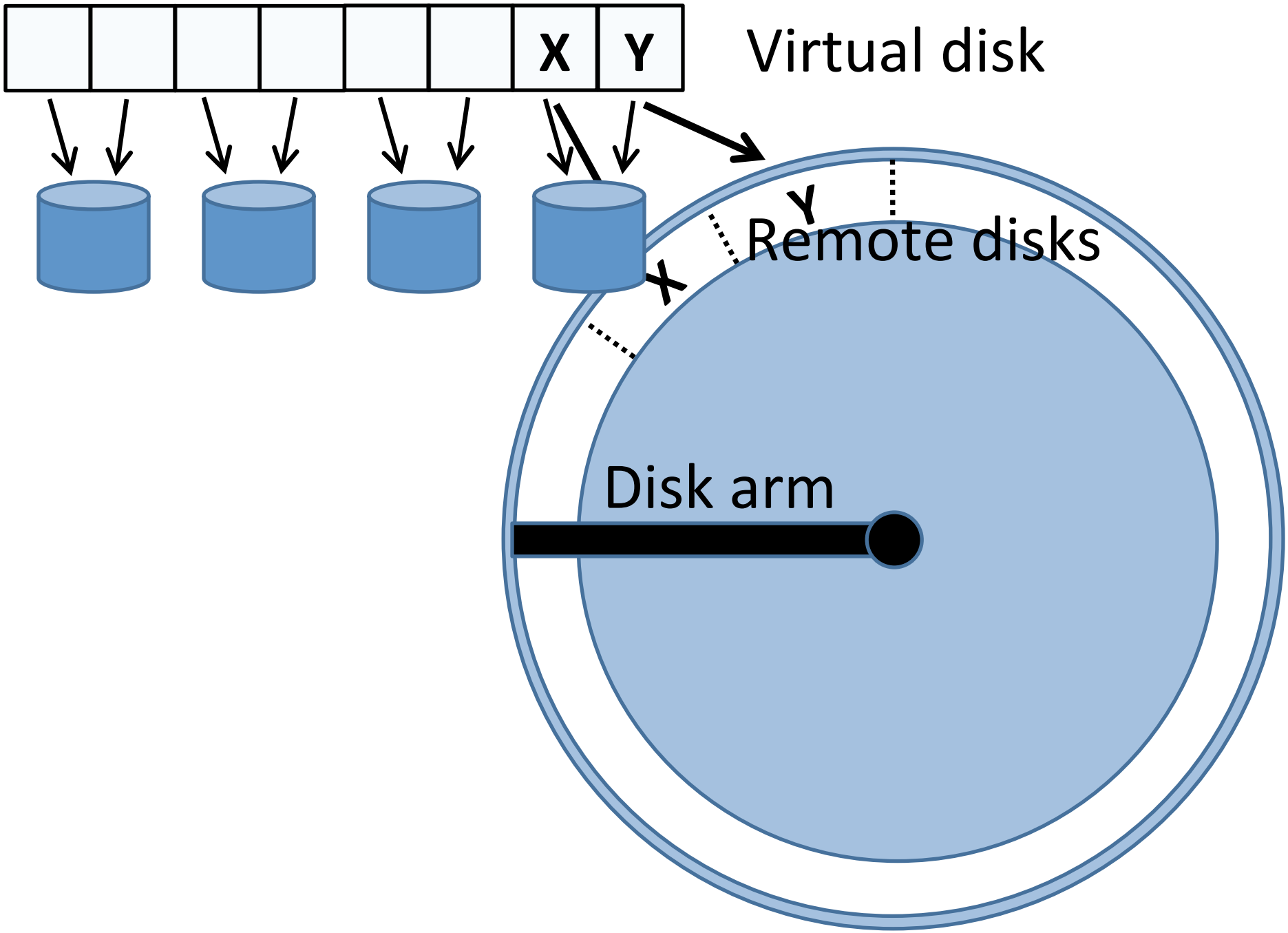
fsync() write barriers:
Delayed durability
semantics

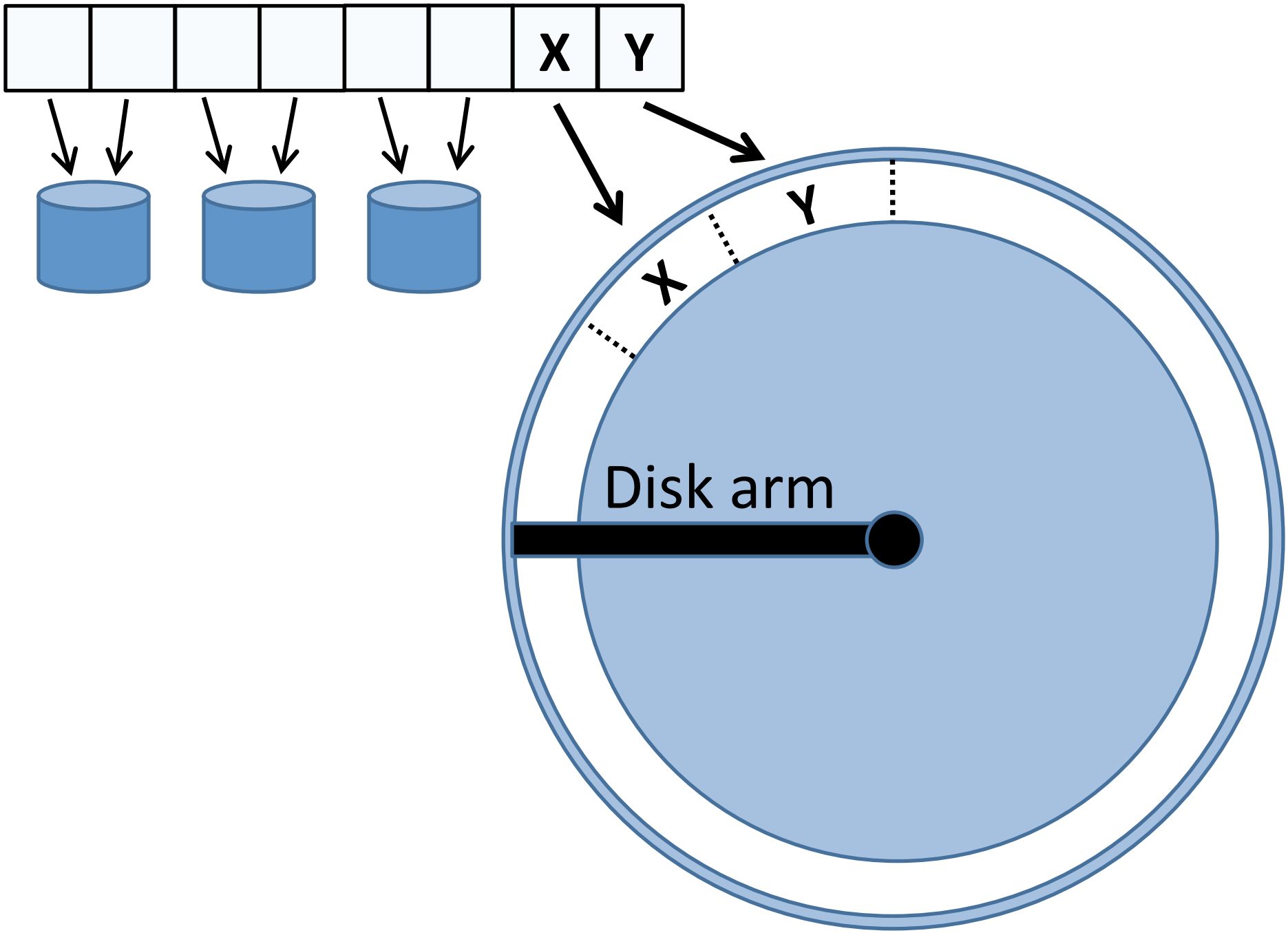


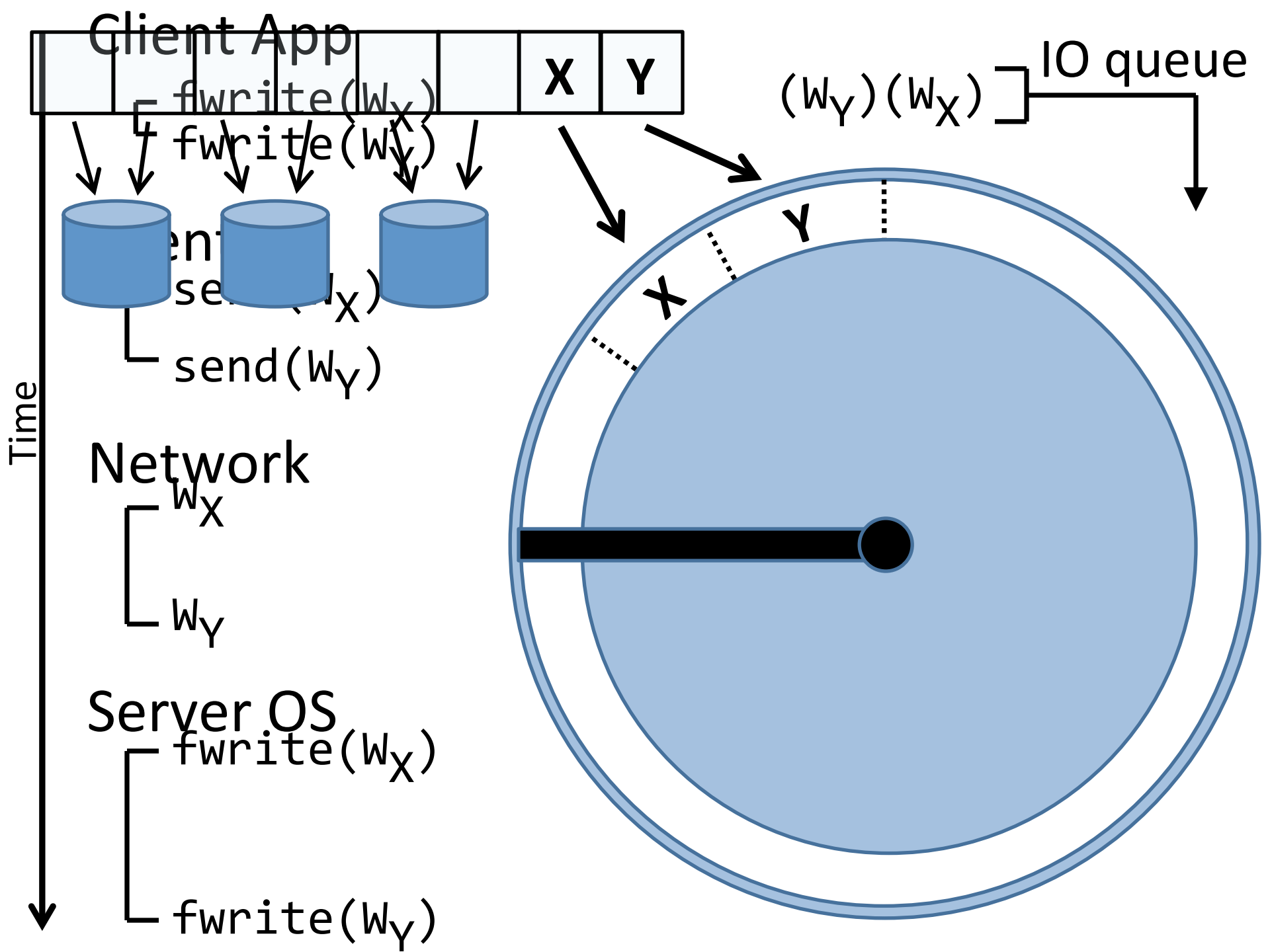


Virtual disk

Remote disks


X
Y
Virtual disk
Y
Remote disks
X
Disk arm


X
Y
Y
X
Disk arm


Client App
fwrite(W_X)
fwrite(W_Y)
X
Y
IO queue
(W_Y)(W_X)
send(W_X)
send(W_Y)
Time
Network
W_X
W_Y
Server OS
fwrite(W_X)
fwrite(W_Y)

IOp Convoy Dilation

Client App

fwrite(W_X)

fwrite(W_Y)

Client OS

fwrite(W_X)

fwrite(W_Y)

Network

fwrite(W_X)

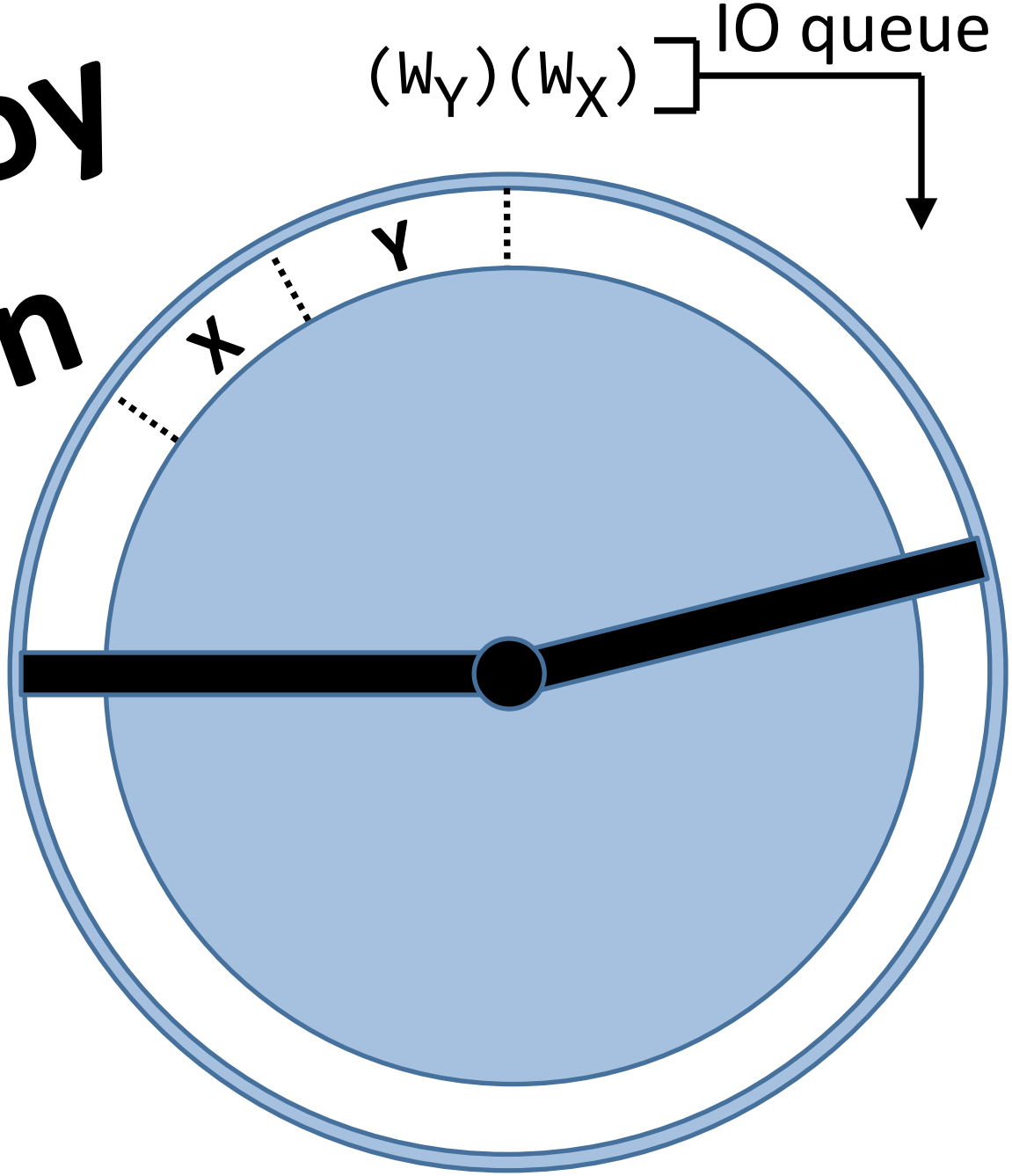
fwrite(W_Y)

Server OS

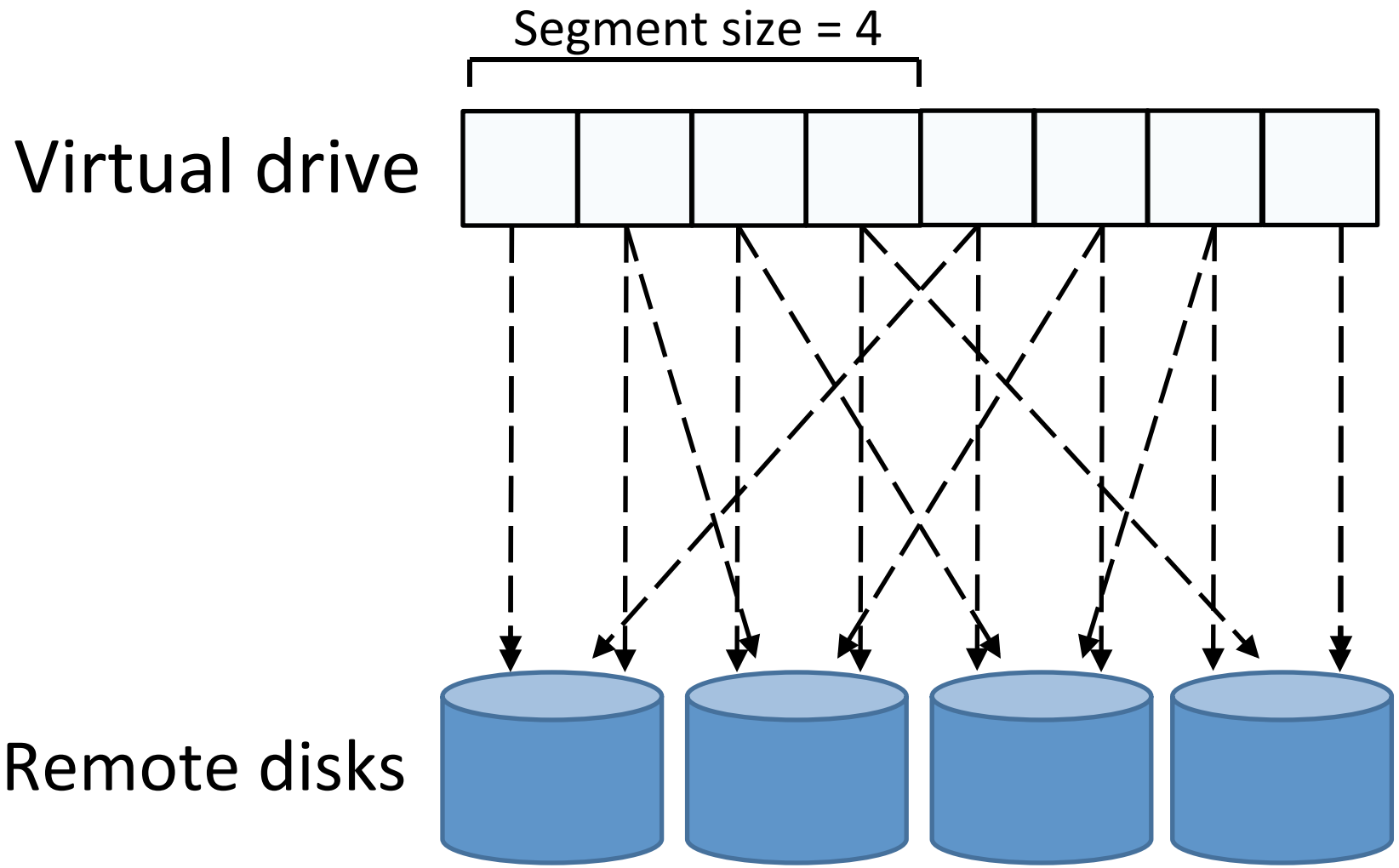
fwrite(W_X)

fwrite(W_Y)

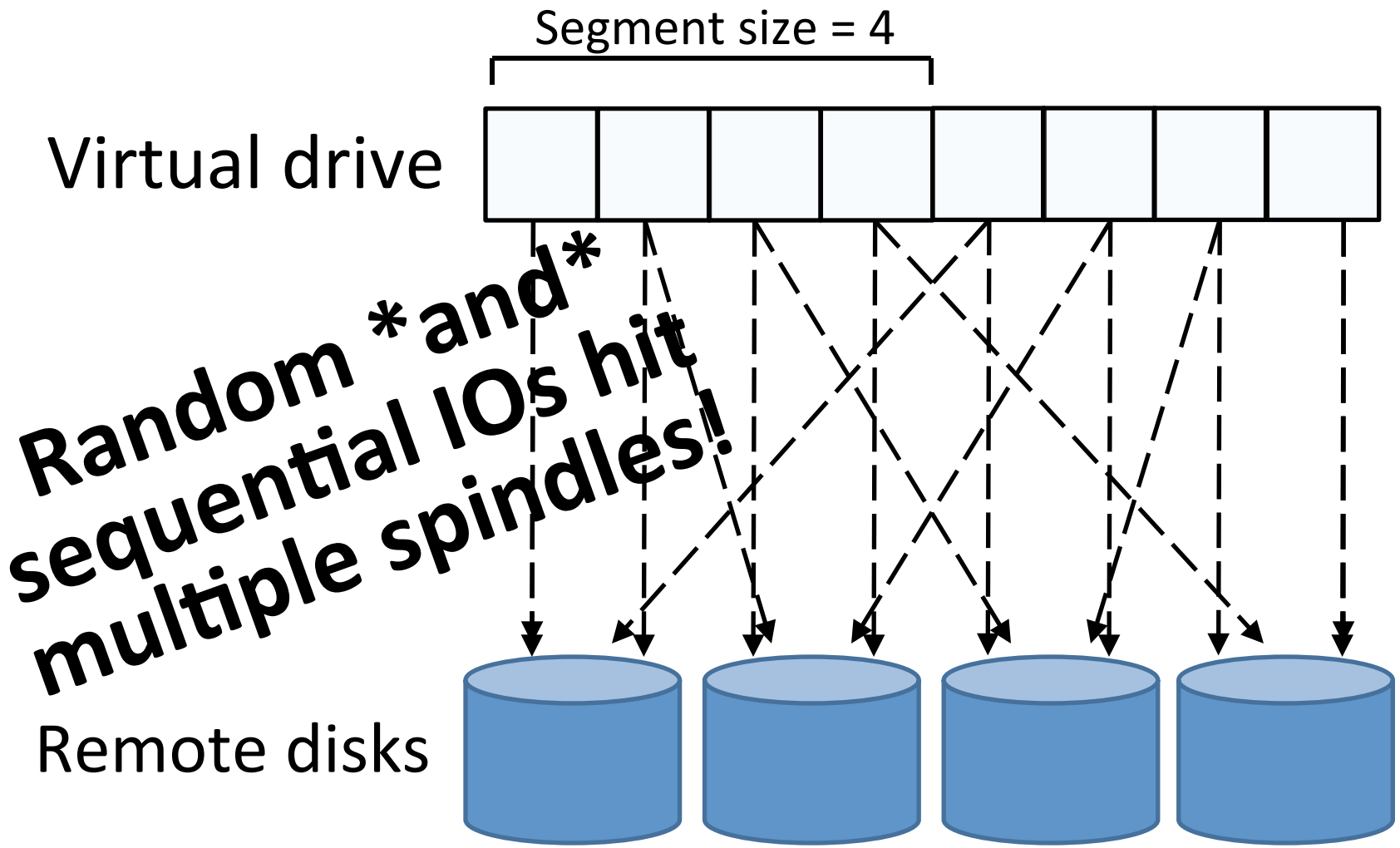
Time



Fixing IOp Convoy Dilation



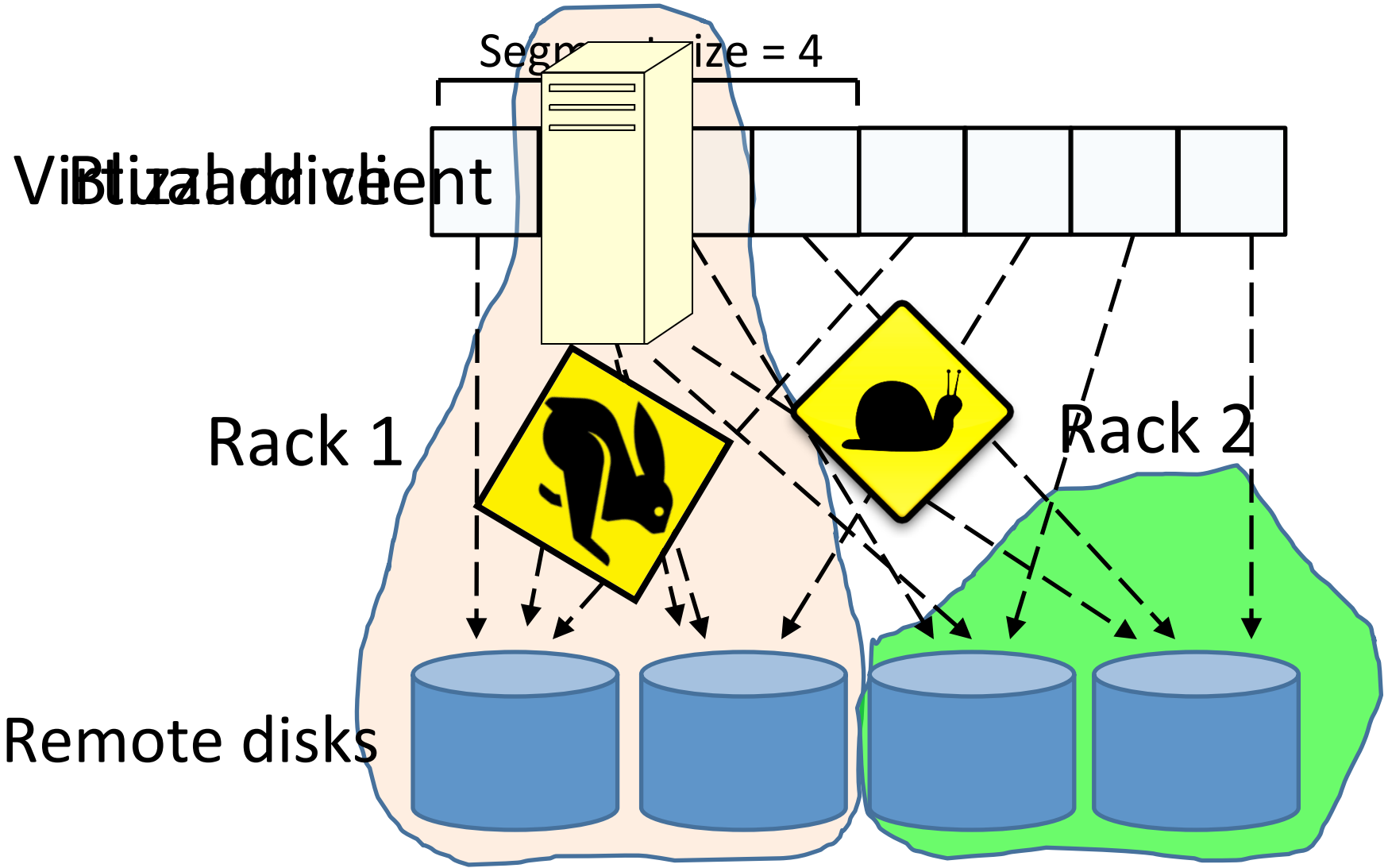
Fixing IOp Convoy Dilation




TOTAL
VICTORY

Rack Locality

Rack Locality

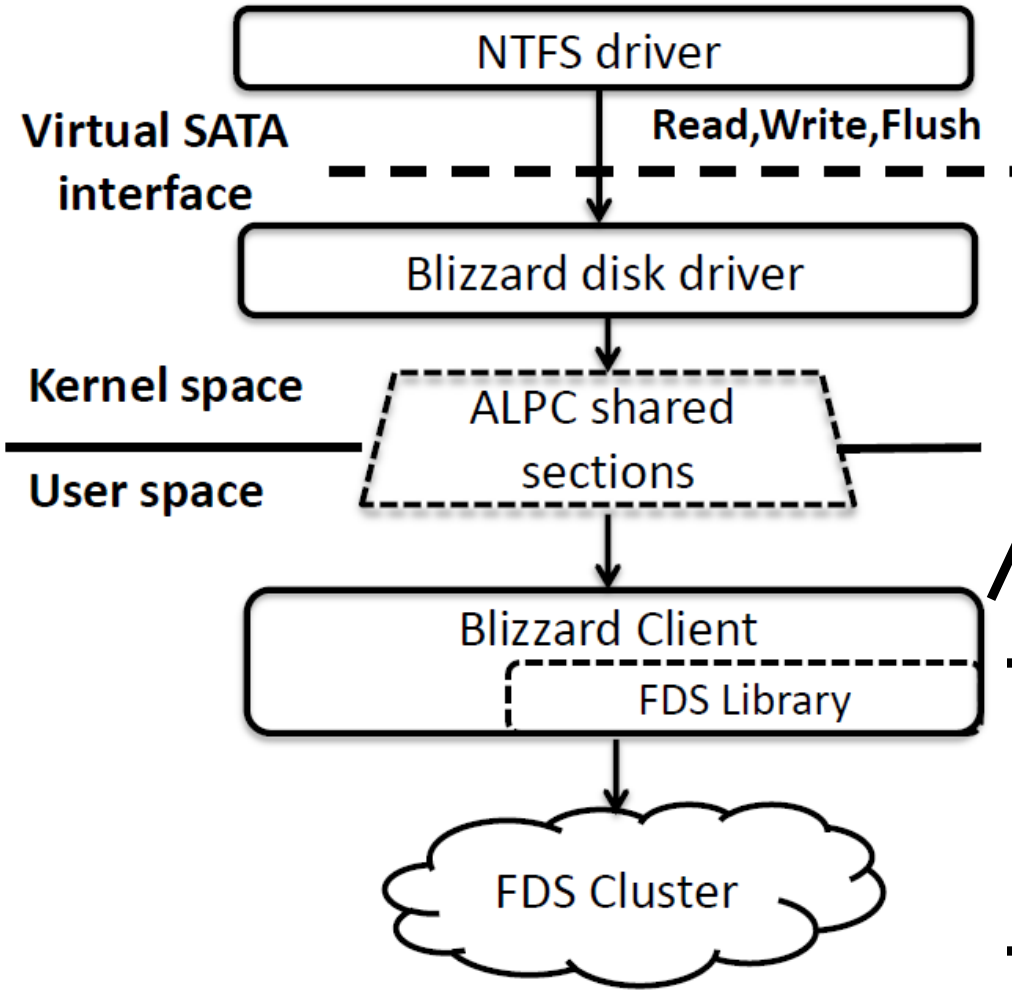


AIN’T NOBODY GOT TIME FOR THAT

FDS To The Rescue (OSDI 2012)

- Hardware architecture
  - Full bisection bandwidth network (no oversubscription)
  - Allocate each disk enough network bandwidth to drive disk at full sequential speed (1 disk ≈ 128 MB/s ≈ 1Gbps)
- Result: locality-oblivious storage
  - Any client can access any disk as fast as local
  - **Enables aggressive striping**

Blizzard as FDS Client



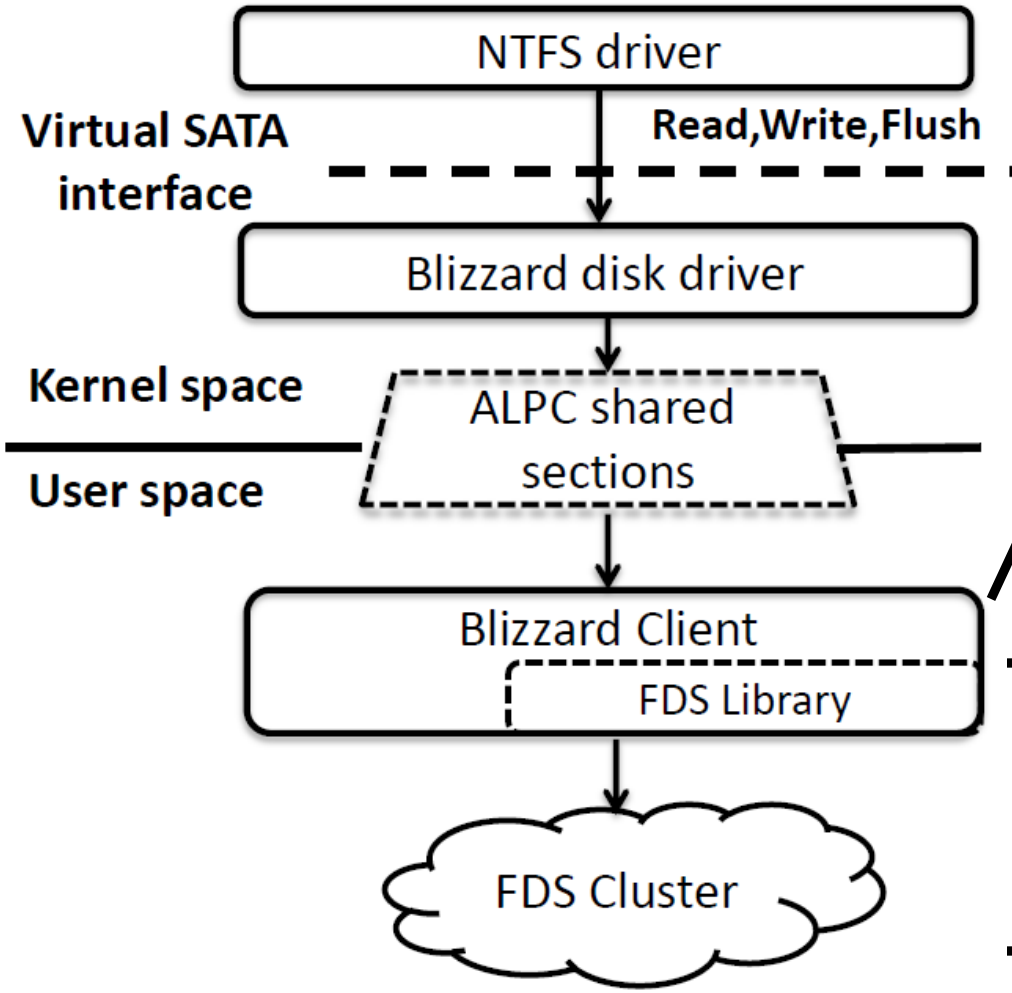
Blizzard client handles:

- Nested striping
- Delayed durability semantics

FDS provides:

- Locality-oblivious storage hardware
- Server-side failure recovery
- RTS/CTS to avoid edge congestion

Blizzard as FDS Client



Blizzard client handles:

- Nested striping
- Delayed durability semantics

FDS provides:

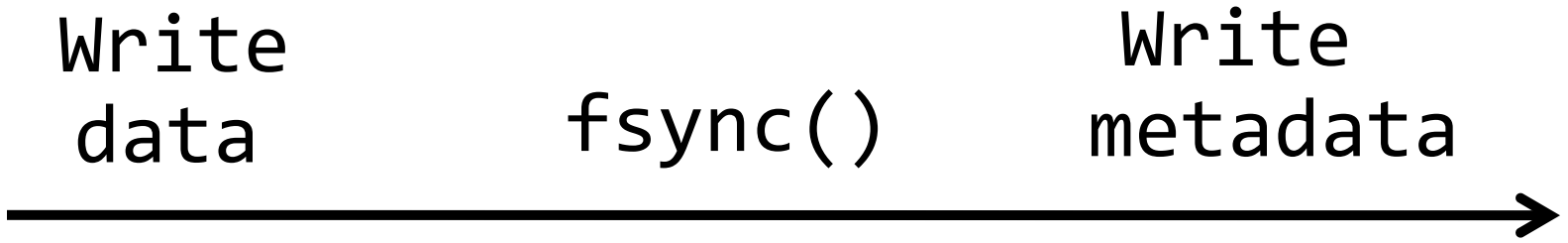
- Locality-oblivious storage hardware
- Server-side failure recovery
- RTS/CTS to avoid edge congestion


POSIX/Win32
apps
Traditional
big-data apps

ARE WE THERE YET?

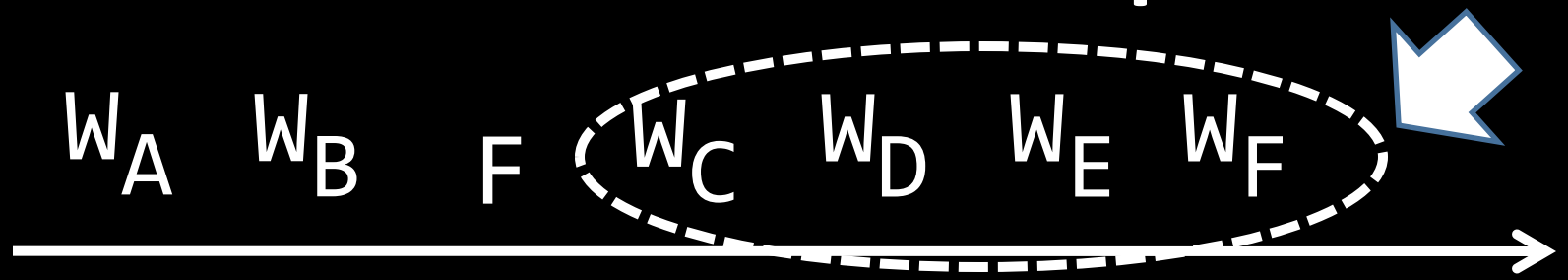
The problem with `fsync()`

- Used by POSIX/Win32 file systems and applications to implement crash consistency
  - Disk only returns from `fsync()` when all prior writes have become durable
  - Ex: ensure data is written before metadata



WRITE BARRIERS RUIN BIRTHDAYS

Stalled operations limit parallelism!

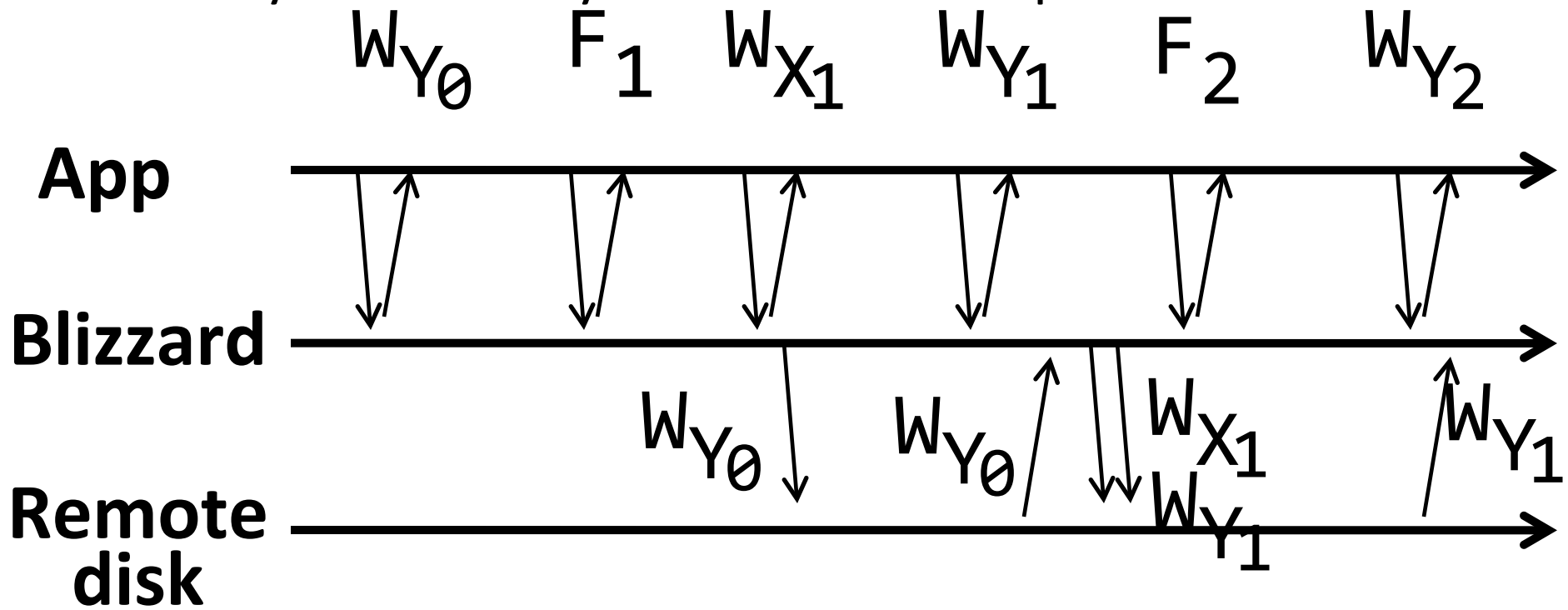


Delayed Durability

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

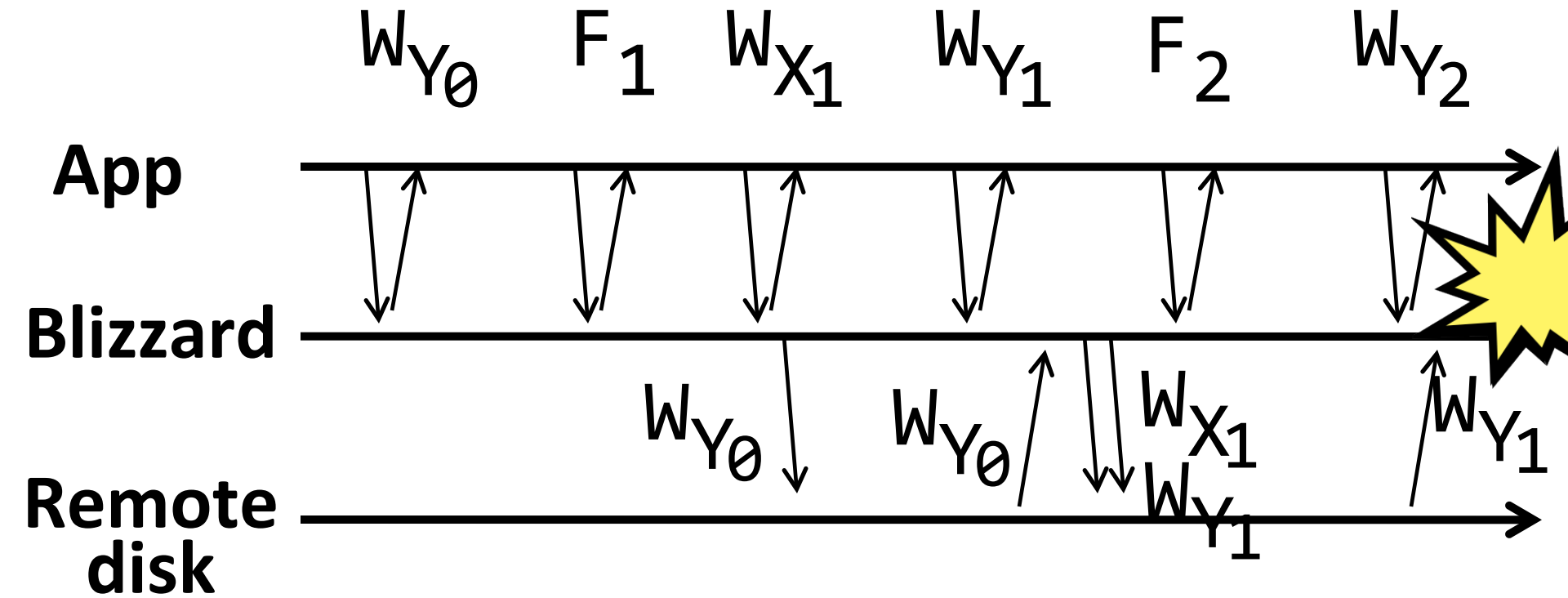
Delayed Durability

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

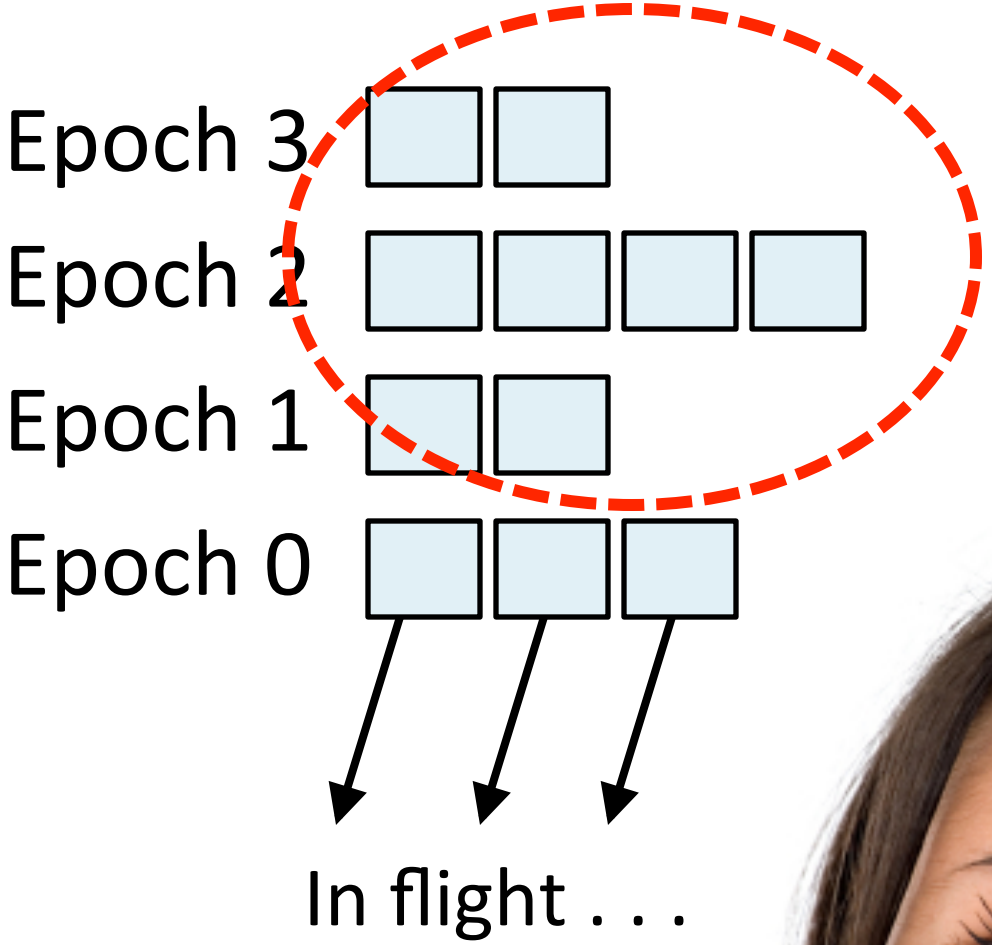


Delayed Durability

- All writes are acknowledged . . .
- . . . but only W_{Y_0} and W_{Y_1} are durable!
- Decouple durability from ordering
  - Acknowledge fsync() immediately . . .
  - . . . but tag writes with their epoch number
  - . . . and asynchronously retire writes in epoch order
- Prefix consistency good enough for most apps, provides much better performance!



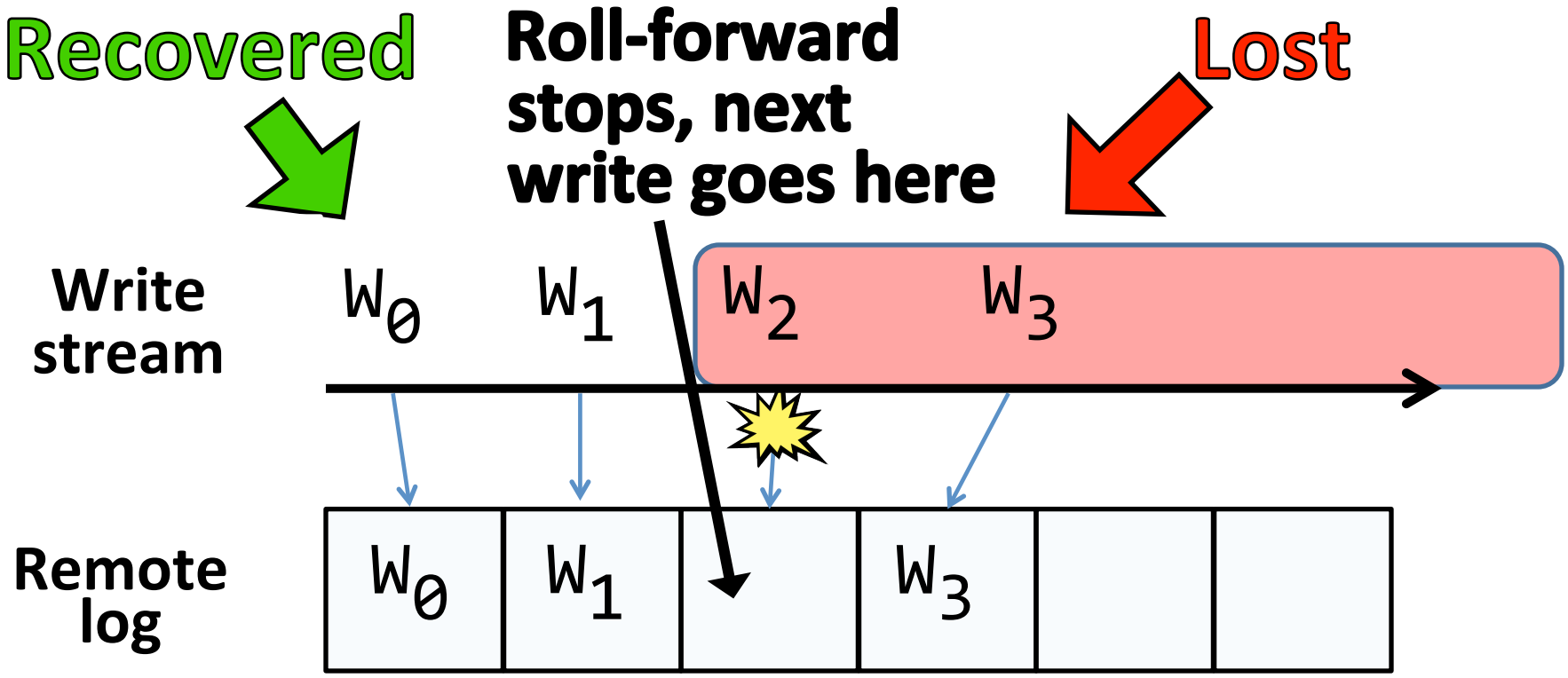
Isn’t Blizzard buffering a lot of data?



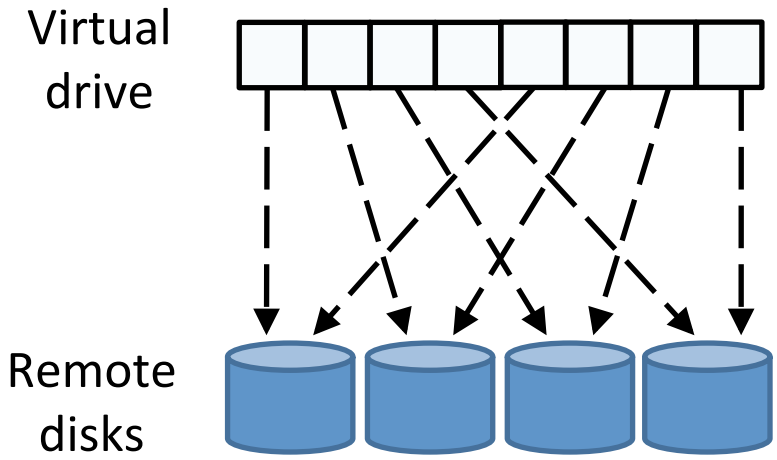

THERE ARE NO NEW TECHNIQUES
SPECULATIVE CACHING?
SPECULATIVE EXECUTION?

Log-based Writes

- Treat backing storage as a distributed log
  - Issue writes to log **immediately** and **in order**
  - On failure, roll forward from last checkpoint and stop when you find torn write, unallocated log block with old epoch number



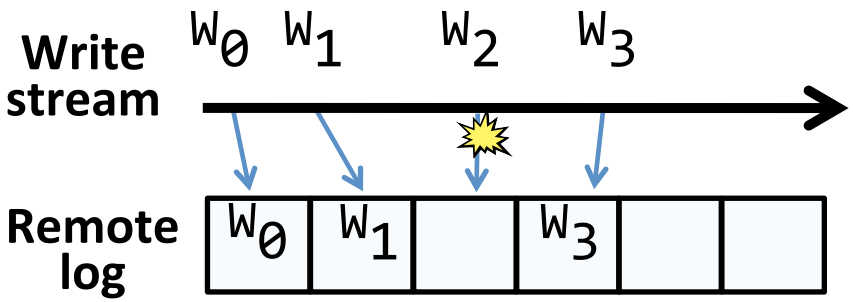
Summary of Blizzard's Design



- Problem: IOp Dilation
- Solution: Nested striping

FDS

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

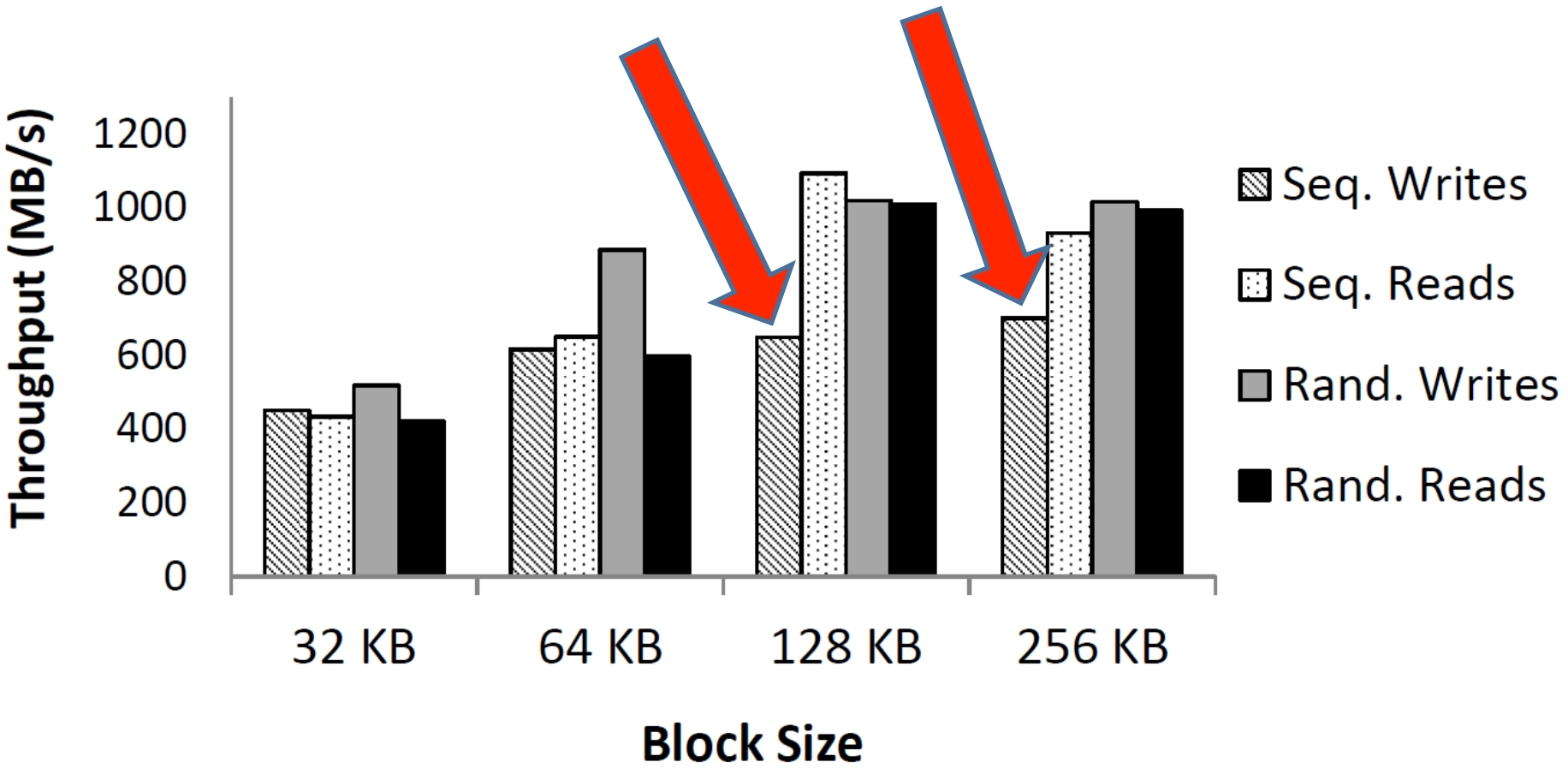


- Problem: Evil fsync()s
- Solution: Delayed durability

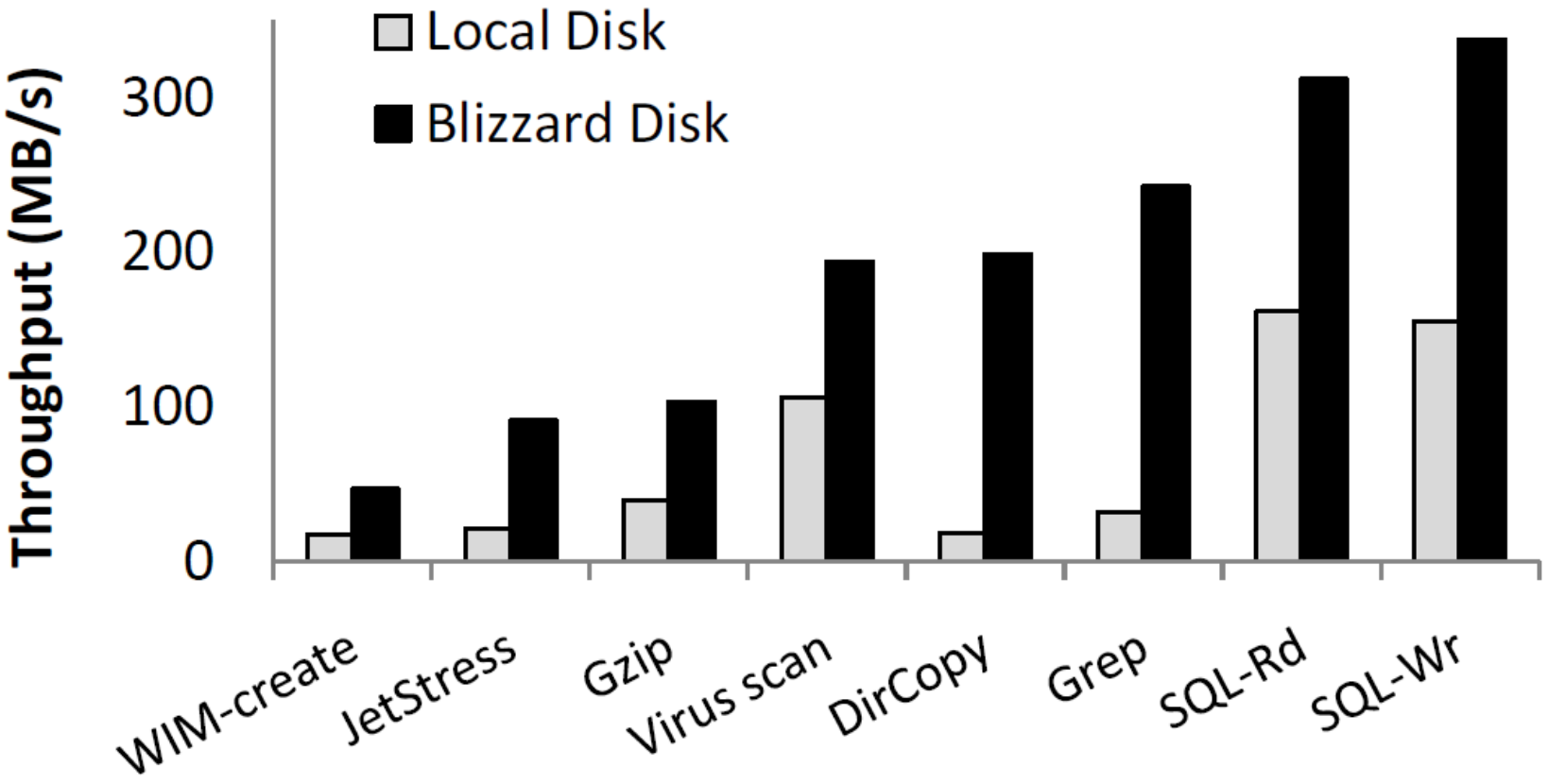
PROOF OF YOUR EXCELLENCE

I DO NOT SEE IT

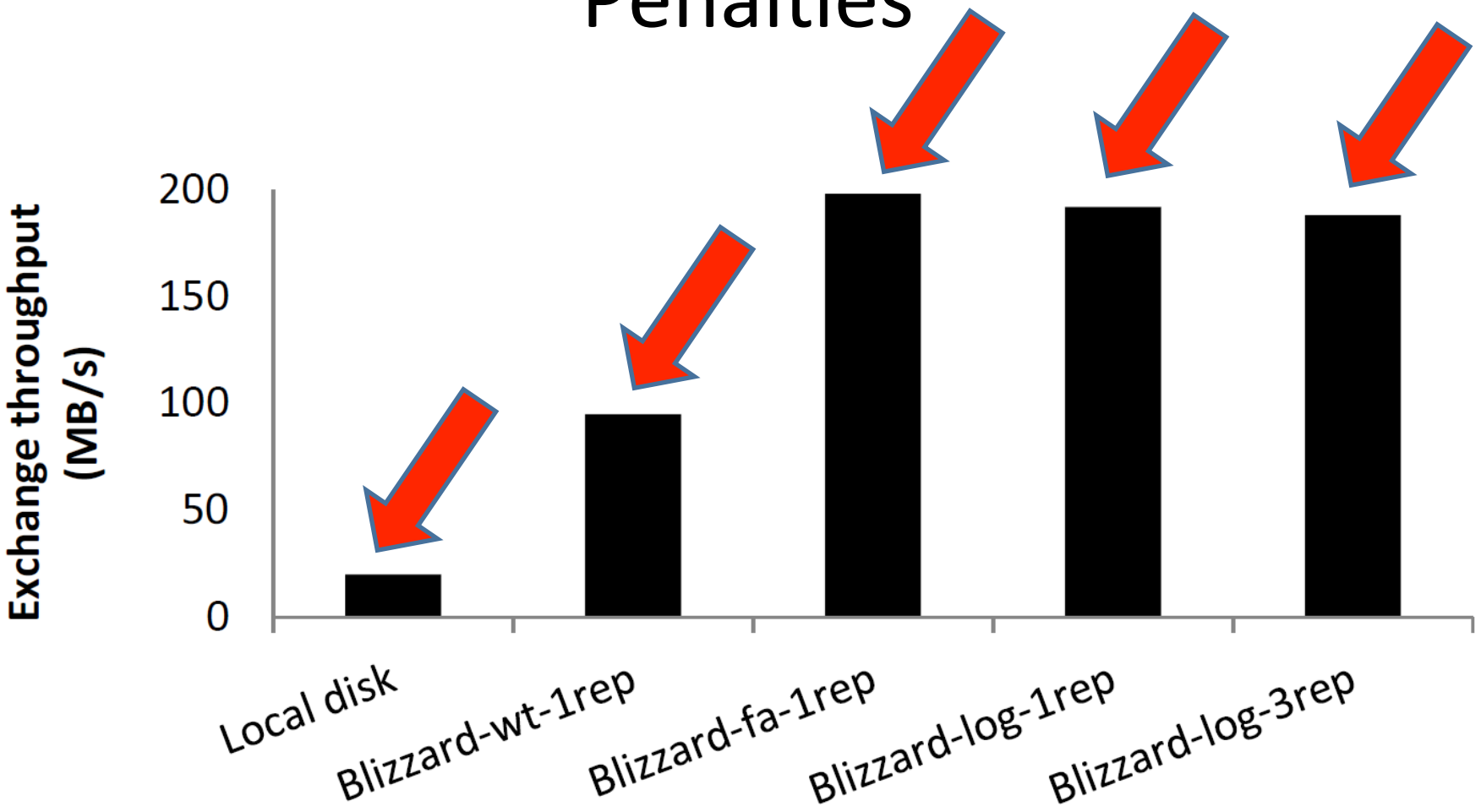
Throughput Microbenchmark



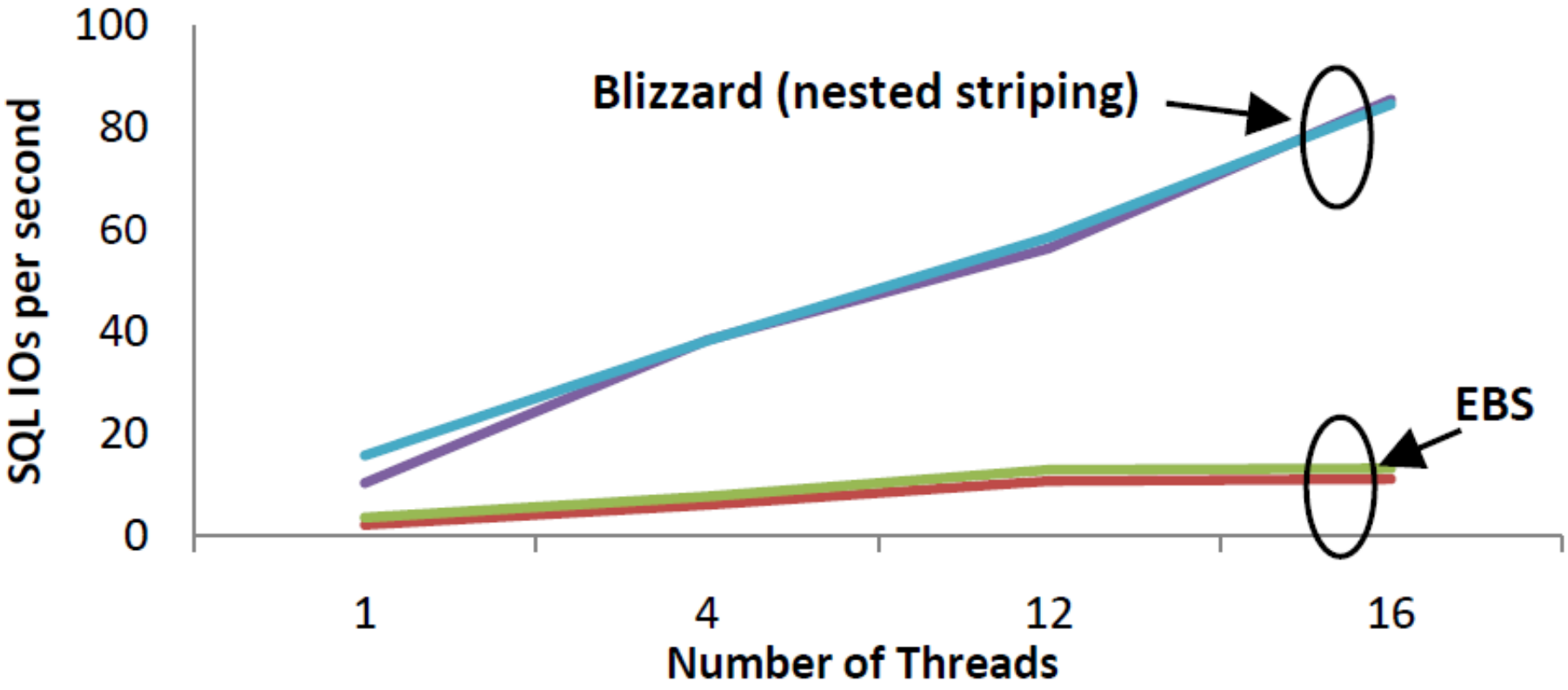
Application Macrobenchmarks



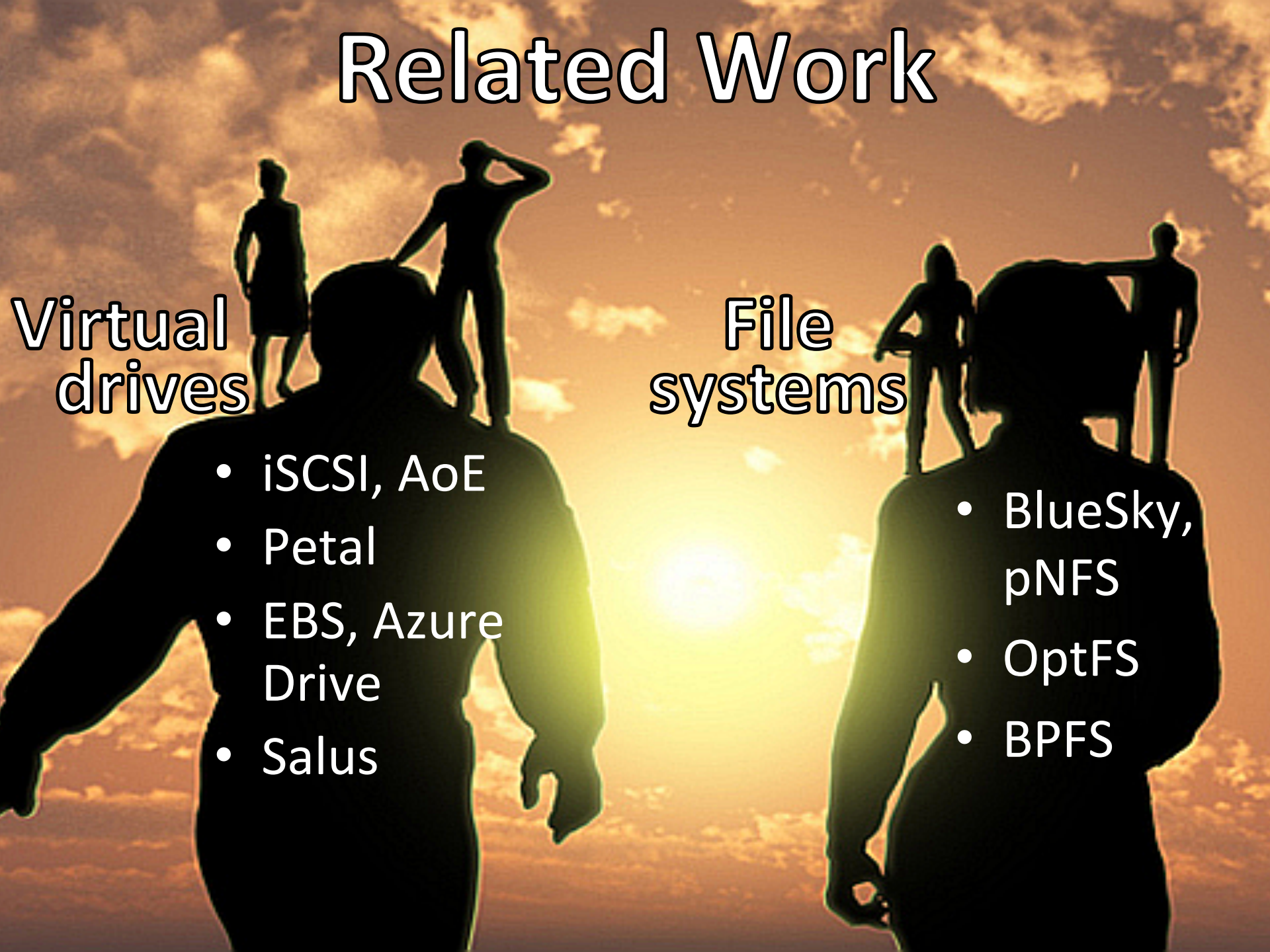
Delayed Durability: Hiding Replication Penalties



Blizzard vs EBS:
Write IOps and Read IOps



Related Work

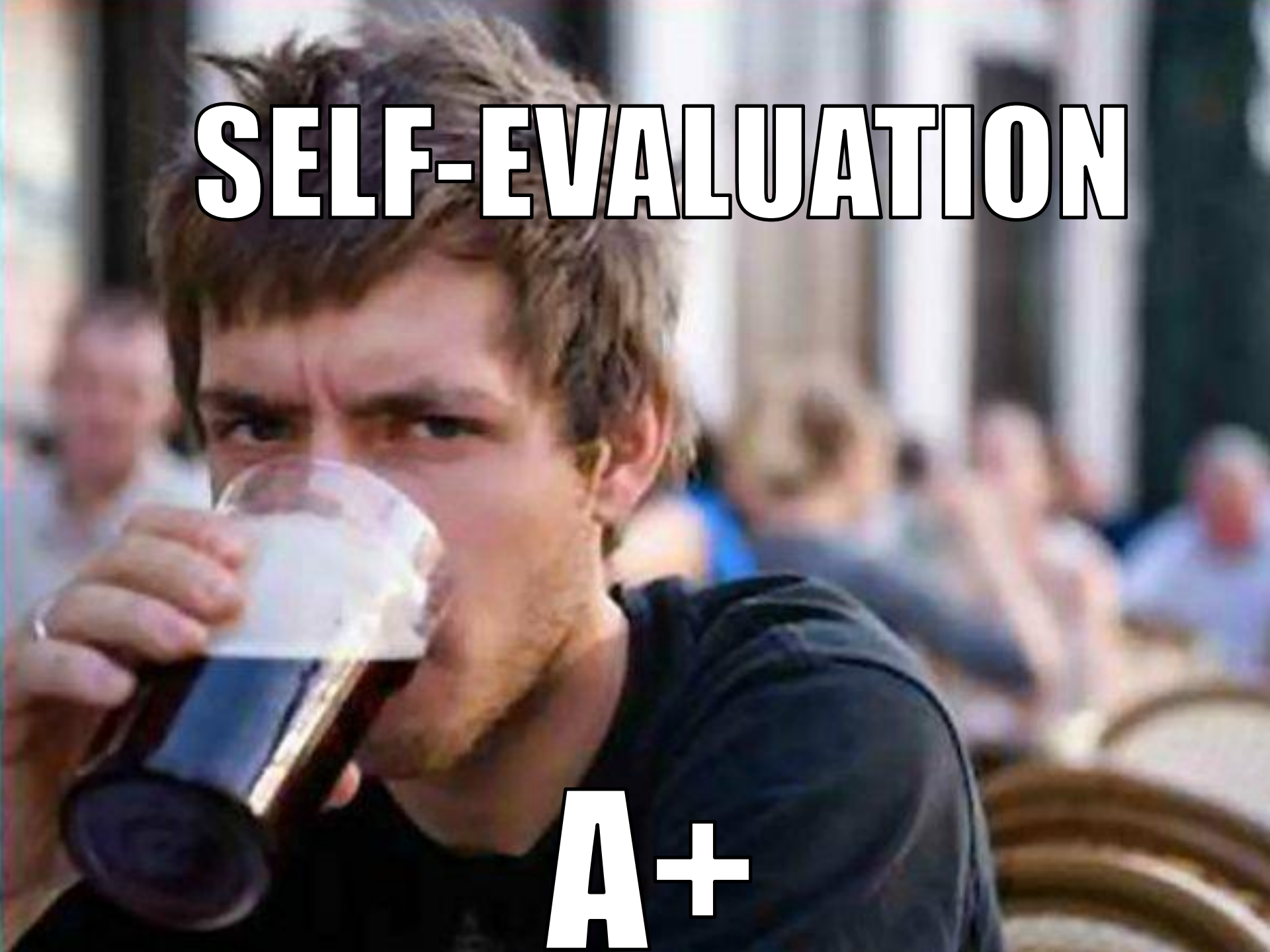


Virtual drives

- iSCSI, AoE
- Petal
- EBS, Azure Drive
- Salus

File systems

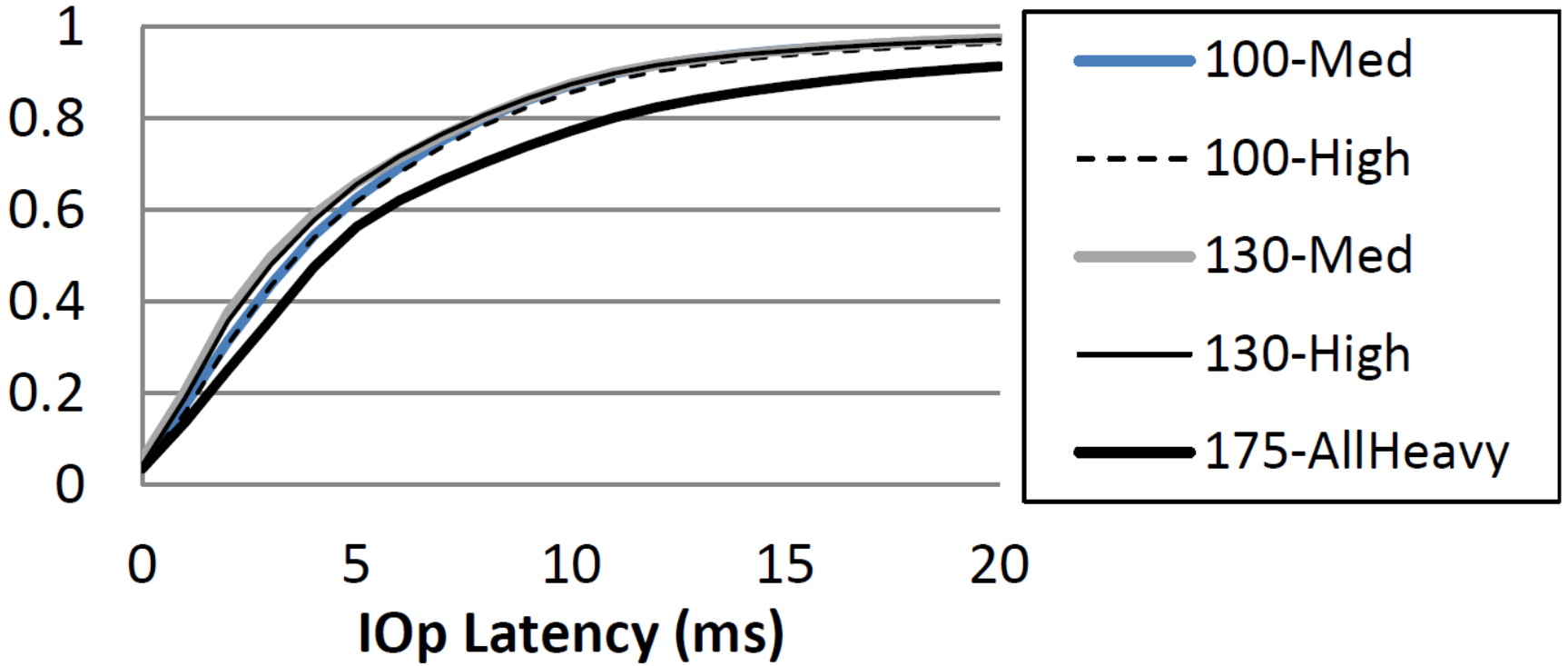
- BlueSky, pNFS
- OptFS
- BPFS


SELF-EVALUATION
A+

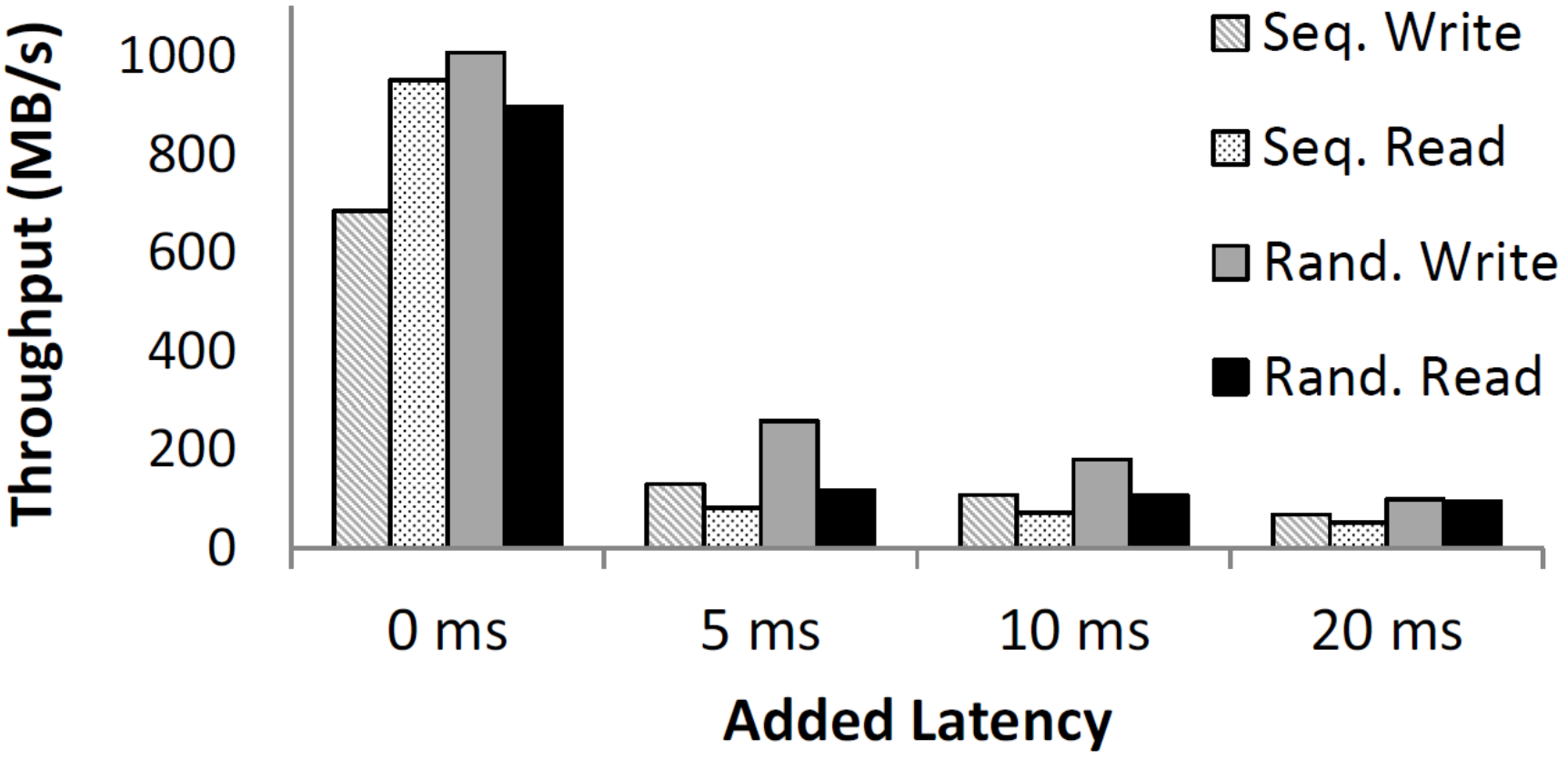
Conclusions

- Unmodified POSIX/Win32 apps can have cloud-scale IO!
  - Nested striping
  - FDS-style hardware substrate
  - Delayed durability semantics
- Raw perf: 1000+ MB/s
- 2x—10x app-level speedups

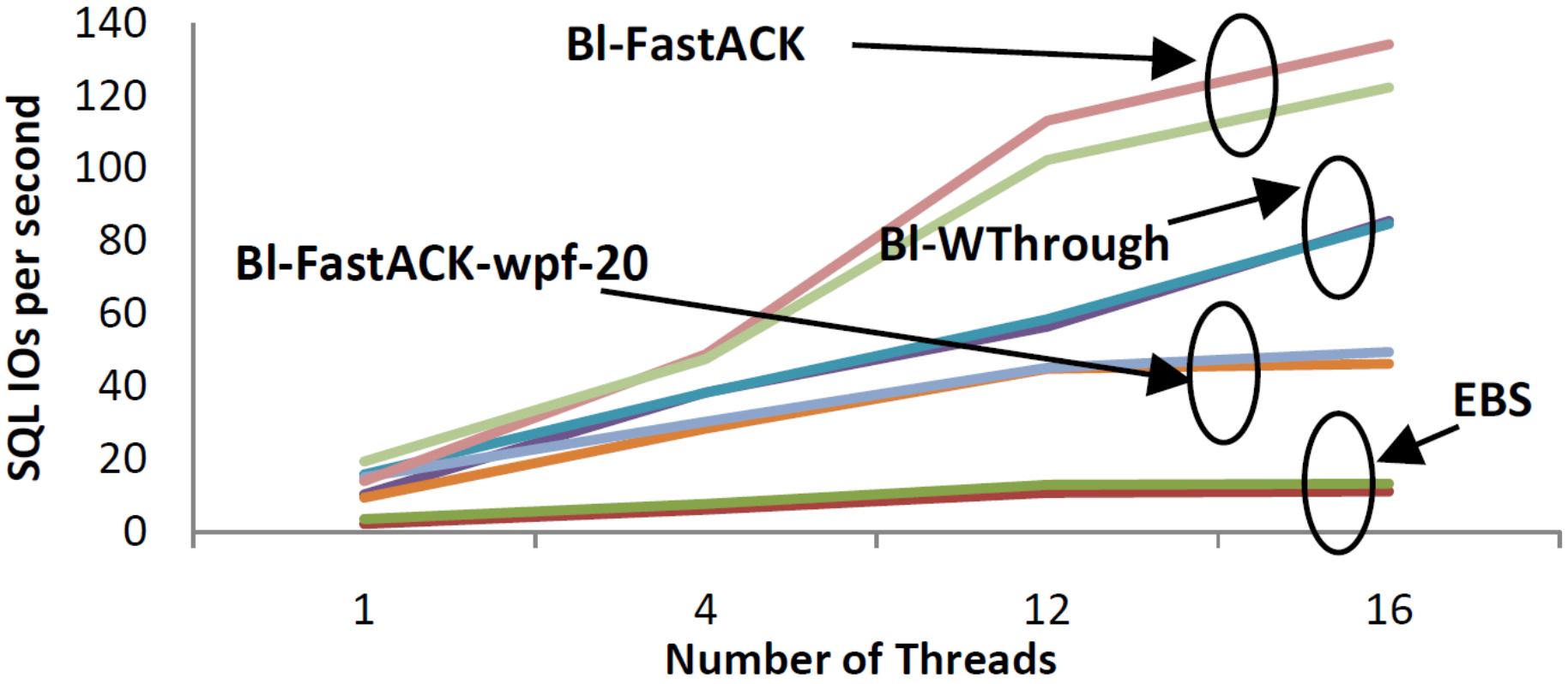
IOp Latency


1
0.8
0.6
0.4
0.2
0
0
5
10
15
20
IOp Latency (ms)
100-Med
100-High
130-Med
130-High
175-AllHeavy

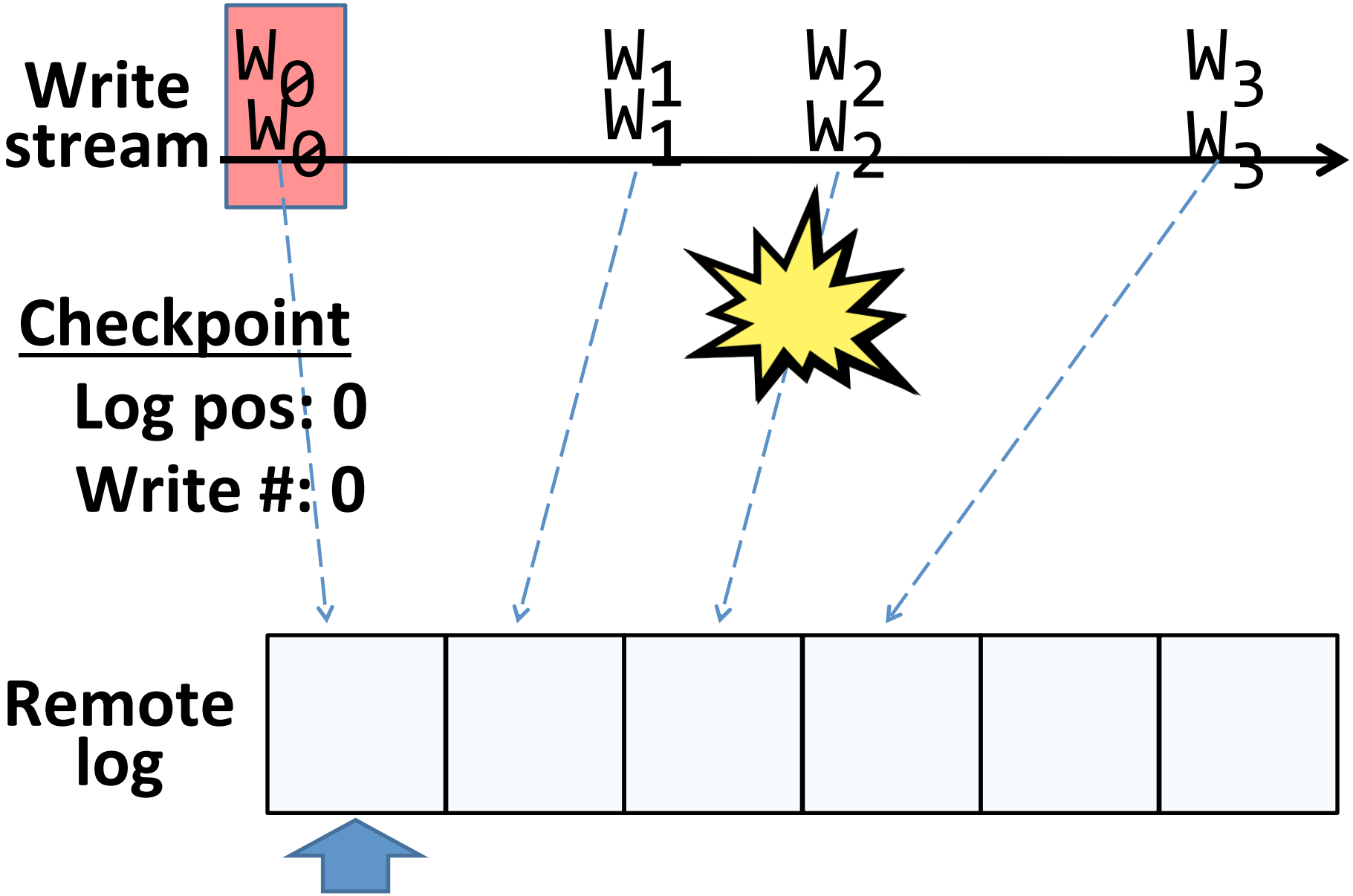
RTT Sensitivity



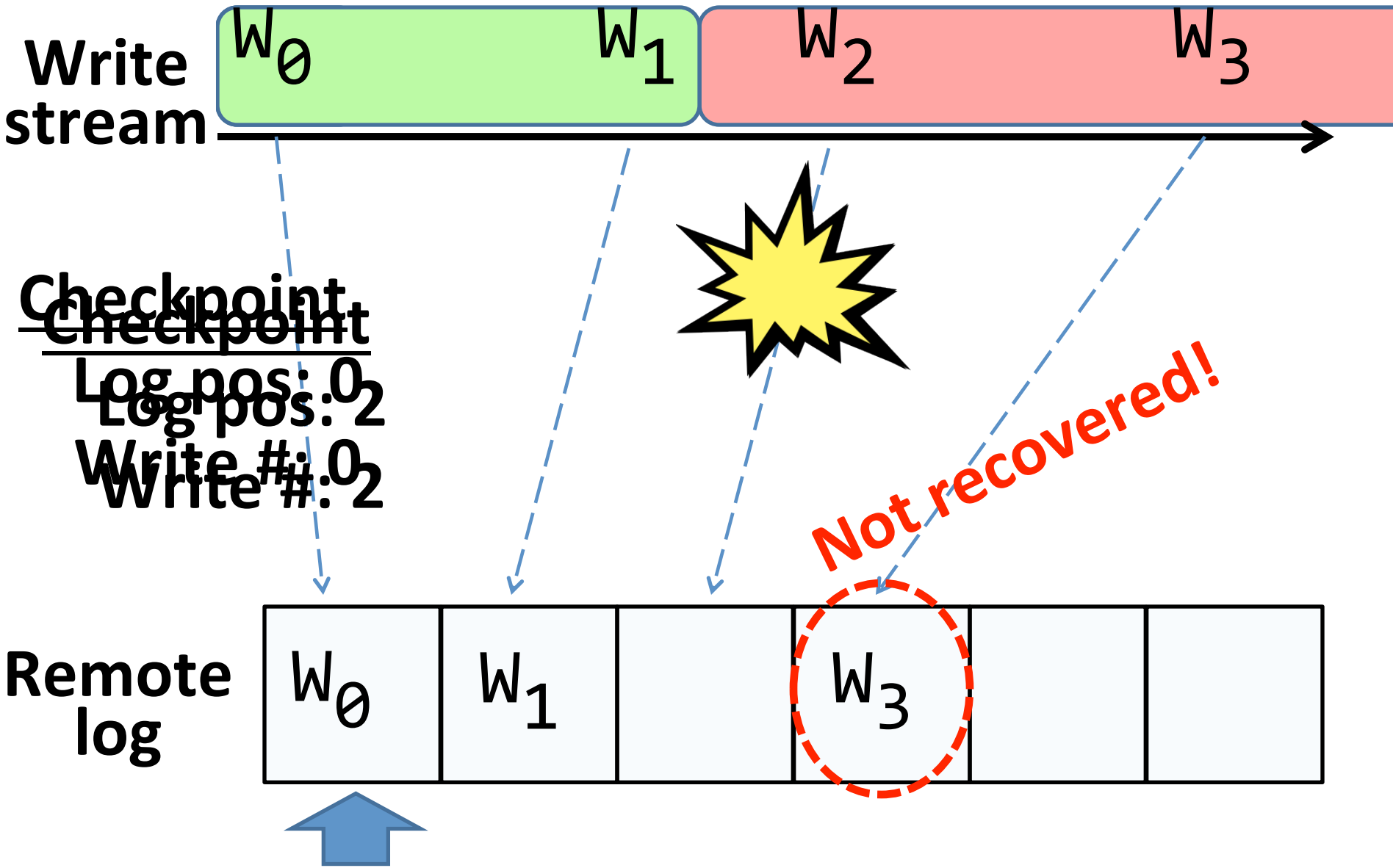
Blizzard vs. EBS



Log-Based Recovery Writes



Recovery


Write stream
W_0
W_1
W_2
W_3
Checkpoint
Log pos: 0
Write #: 0
Checkpoint
Log pos: 2
Write #: 2
Not recovered!
Remote log
W_0
W_1
W_3

Additional Details

- Blizzard maps each virtual block to backing physical block in the log
  - Allocation map included in checkpoints
- To avoid IOp dilation, use random permutation to determine next log position!

Prior example: 0, 1, 2, 3, 4, …

What Blizzard really does: $X_{n+1} = (aX_n + c) \bmod m$

(Checkpoint a, c, m, X_n)