"Flash is good"
-- Jim Gray, 2006

1M IOPS systems are within reach

Page caching throttles IOPS

But why cache?...Locality

Then, F(orget) locality

Implementing locality is expensive and workloads exhibit less and less locality
- graphs (linked data and relationships)
- noSQL (big data analysis)

Redesign the page cache for concurrency rather than locality.

A less reactionary approach?

Sacrifice some cache performance to improve IOPS into the page cache.

For graph analysis workloads that exhibit little locality and benefit less from caching.

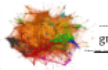
Forget Locality

(Radio Edit)

Randal Burns

Two things to debate:
- are workloads evolving to have less locality
- are there big graph problems

Why does Randal want 1M IOPS?



....to solve big graph problems

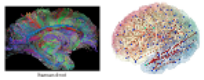
"There is no big graph problem"

-- Martin Kersten, 2010

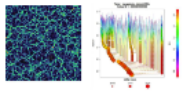
I/O Challenges for Massive Graphs

Can we build data-intensive analysis for graphs?

human brain: 10^11 vertices, 10^15 edges



n-body: 1B particles, 10k snapshots

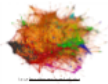


Current computing paradigms:
- supercomputing
- custom hardware (Cray XMT)

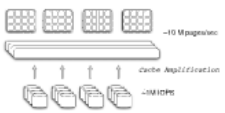
Require in-memory or semi-external memory representations of graphs
- batch computing on cloud

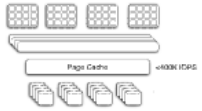
Not fast!

What's Hard about Graphs??



- small and shrinking diameter
- no good cuts
- no locality
- bad I/O lower bounds





I/O Tricks Don't Work

Partition and parallelize (no good partitions)
Localize and cache (no locality)
Stream data (no natural orderings)

I/O friendly (sort-like) algorithms
- list ranking
- connected components
- minimum spanning forest
- maximal matching

I/O unfriendly (traversal) algorithms
- breadth-first search
- shortest paths
- transitive closure
- diameter

and anything else you really want to know

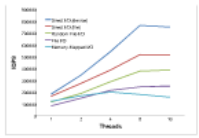
What went wrong?

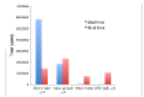
There's a lot of bookkeeping

Address space:
Lists:
alloc
chop
locked
AB Tree
page maps
Page cache:
Page hash table

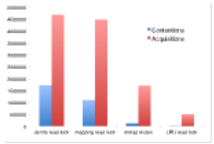
Buffer cache:
Lists:
clean
dirty
locked
cleared
and used
Buffer hash table

....and locks for all





What we think?
it's locking overhead
What we can't show?
it's locking overhead



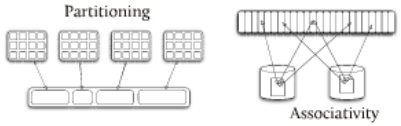
Locks contention is moderate
wait times are small

But, locking needs expensive cache coherency
when requested from multiple cores
we are performing "knock out" experiments
to verify lock overheads

Both techniques used in processor caches.

Something more interesting to do?

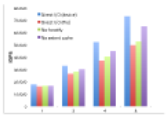
Directions
- reduce lock intersection (decluster locks)
- speculative lock elision (elimintae locks)
- move away from global data structures

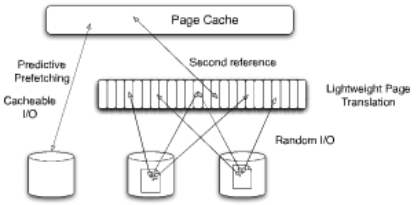


Investment Strategies

Promote pages on second hit
Old idea (2Q-LRU, LIRS, ARC)
but about investing cycles, not replacement

In this manner, we can accelerate direct file I/O
- Reduces interference (not overhead)





Thought for food
- Are there large graphs? (that you care about?)
- Is workload locality decreasing?
- Are random read IOPS a performance limiting factor for analysis engines?

Forget Locality

(Radio Edit)

Randal Burns

Department of Computer Science
Institute for Data Inensive Engineering
and Science (IDIES)
Johns Hopkins University

"Flash is good"

-- Jim Gray, 2006

M IOPS systems are within rea

"Flash is good"

-- Jim Gray, 2006

1M IOPS systems are within reach

Page caching throttles IOPS

But why cache?....Locality

But why cache?....Locality

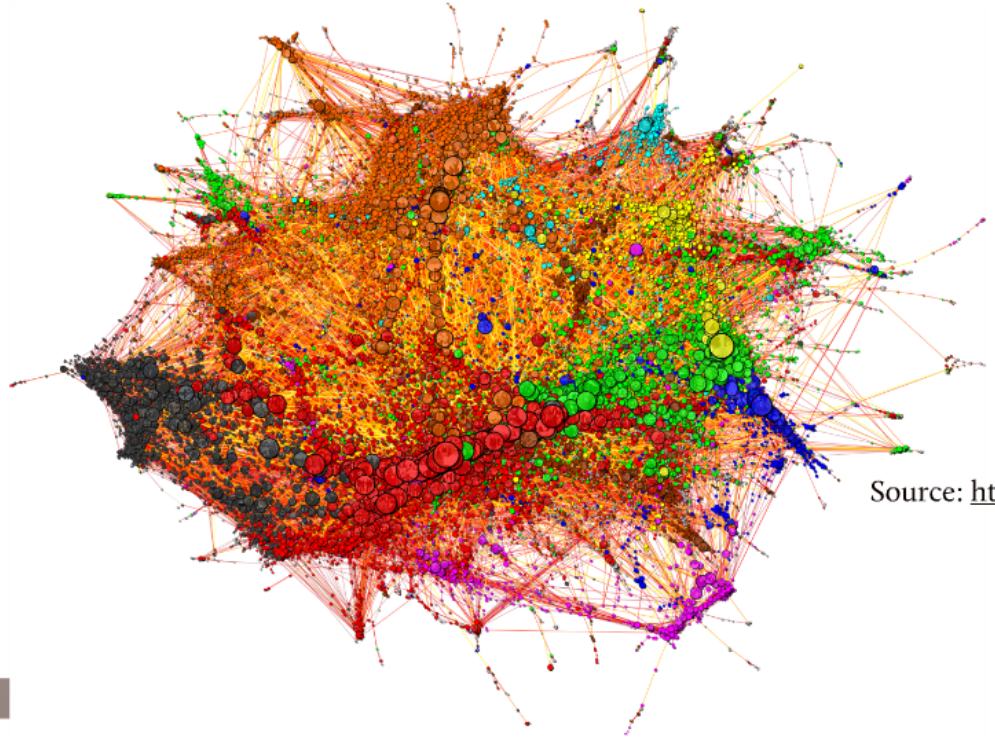
Then, F(orget) locality

Implementing locality is expensive and workloads exhibit less and less locality

- graphs (linked data and relationships)
- scans (big data analysis)

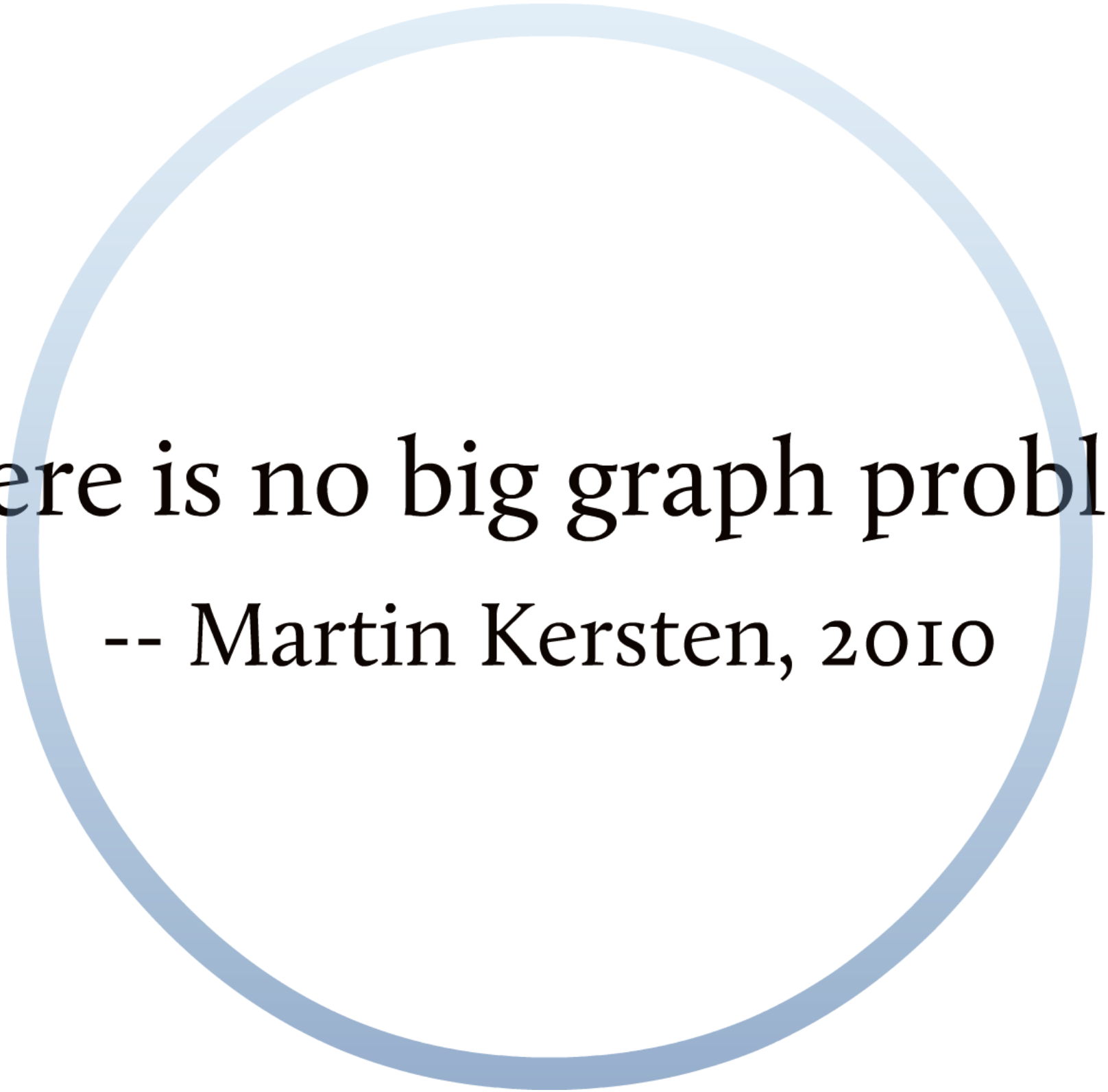
and Science (IDIES)
Johns Hopkins University

Why does Randal want 1M IOPS?



....to solve big graph problems

Source: https://http://sixdegrees.hu/last.fm/


'There is no big graph problem'
-- Martin Kersten, 2010

A less reactionary approach?

Sacrifice some cache performance to improve IOPS into the page cache.

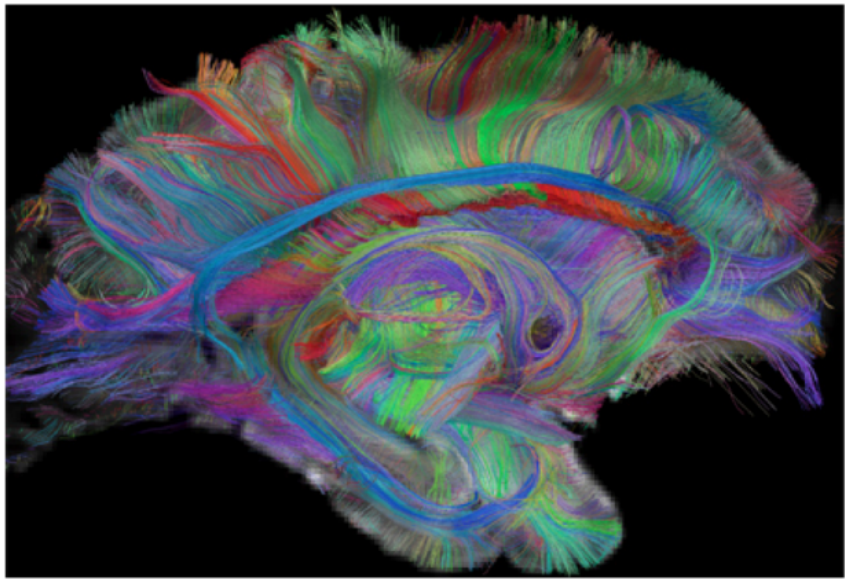
For graph analysis workloads that exhibit little locality and benefit less from caching.

Redesign the page cache for concurrency rather than locality.

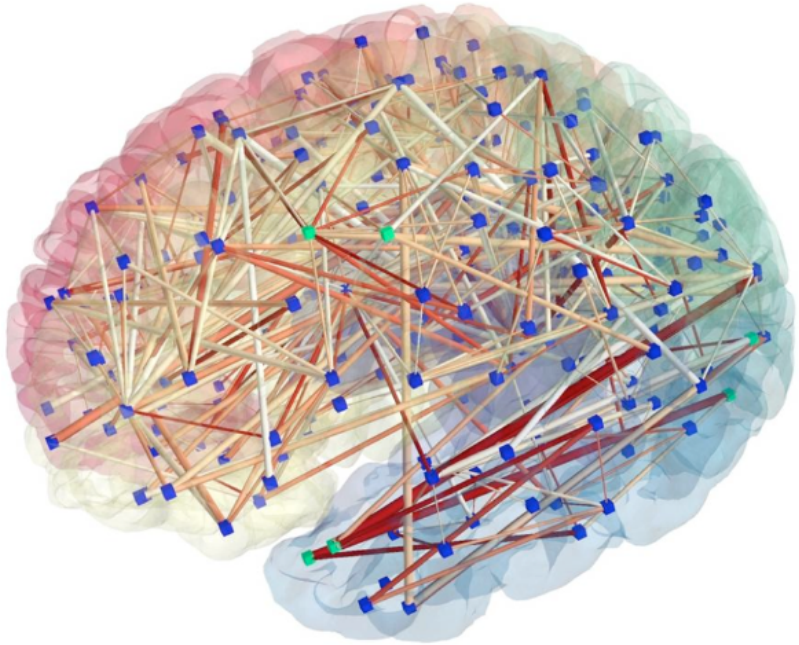
I/O Challenges for Massive Graphs

Can we build data-intensive analysis for graphs?

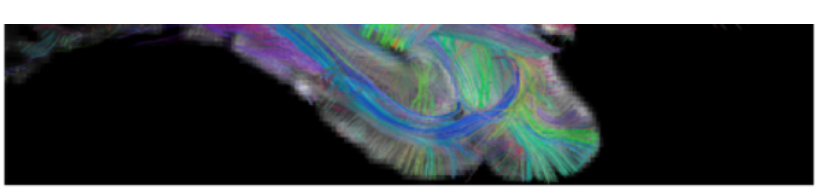
human brain: 10^{11} vertices, 10^{15} edges

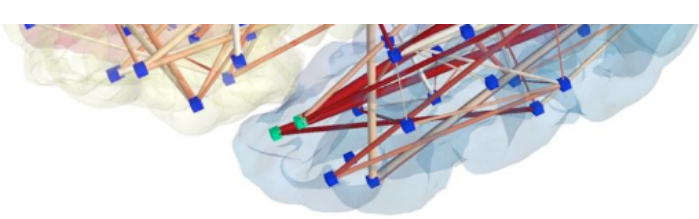


human dmri



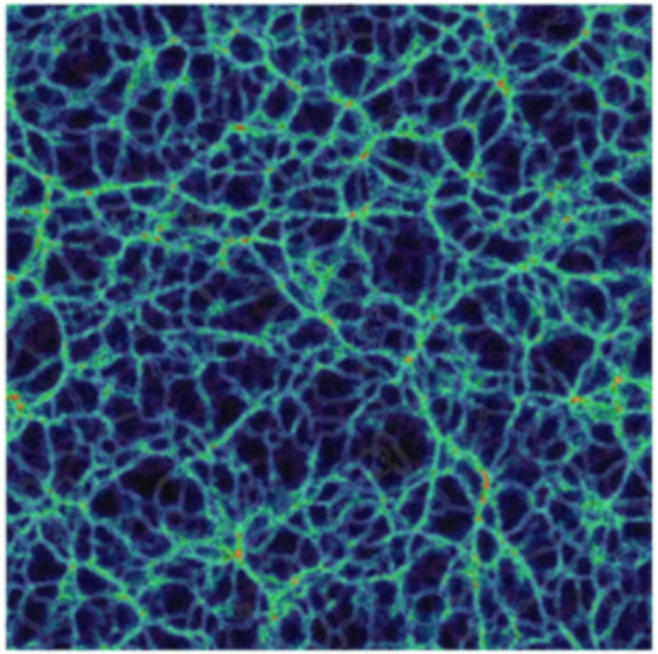
n-body: 1B particles, 10k snapshots

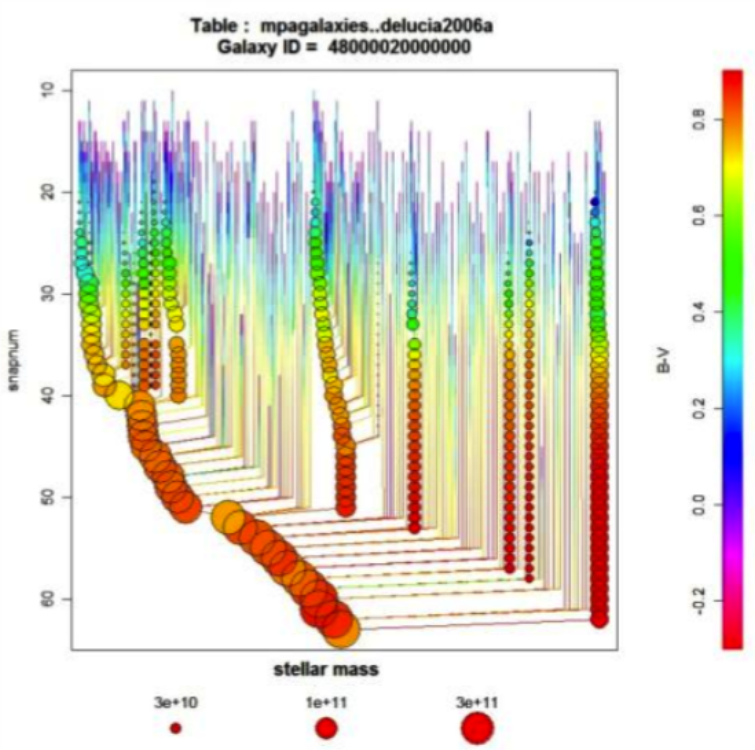




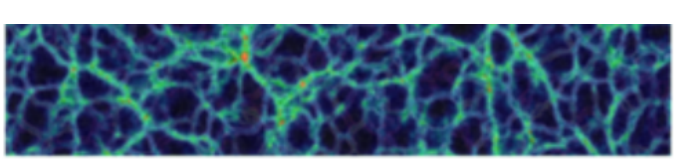
human dmri

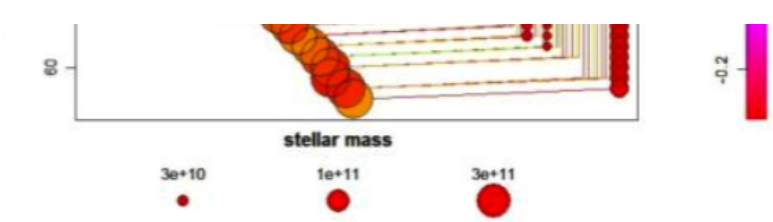
n-body: 1B particles, 10k snapshots





Current computing paradigms:





Current computing paradigms:

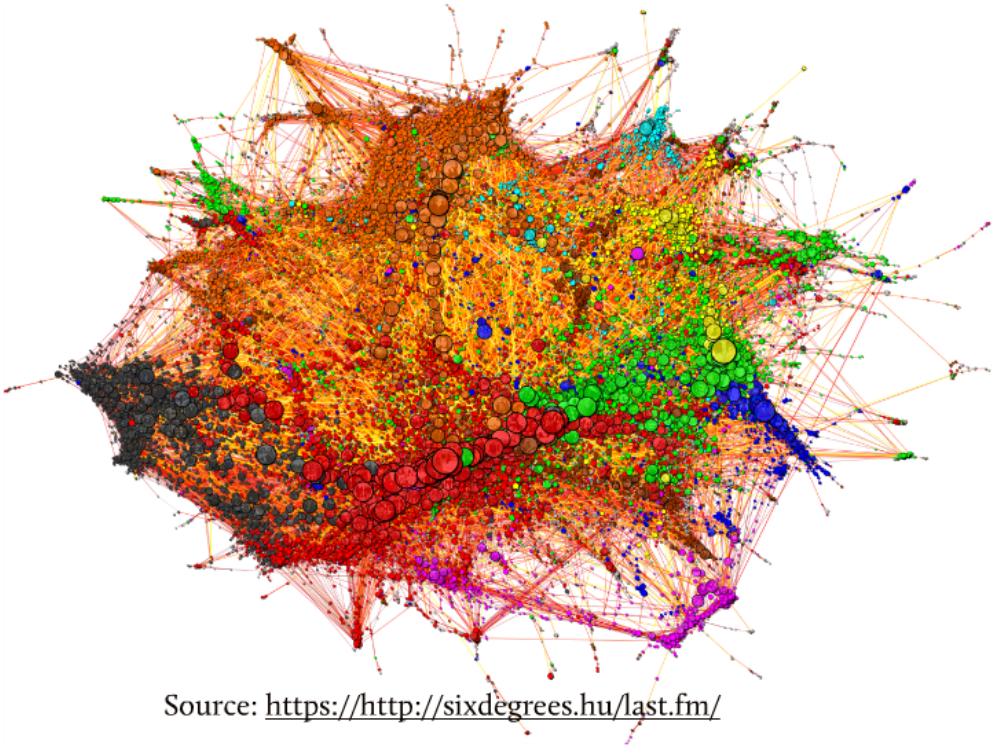
- supercomputing
- custom hardware (Cray XMT)

Require in-memory or semi-external memory representations of graphs

- batch computing on cloud

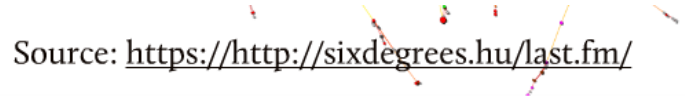
Not fast!

What's Hard about Graphs??



Source: https://http://sixdegrees.hu/last.fm/

- small and shrinking diameter
- no good cuts
- no locality
- bad I/O lower bounds



Source: https://http://sixdegrees.hu/last.fm/

I/O Tricks Don't Work

Partition and parallelize (no good partitions)
Localize and cache (no locality)
Stream data (no natural orderings)

I/O friendly (sort-like) algorithms

Localize and cache (no locality)
Stream data (no natural orderings)

I/O friendly (sort-like) algorithms

- list ranking
- connected components
- minimum spanning forest
- maximal matching

I/O unfriendly (traversal) algorithms

- breadth-first search

- connected components
- minimum spanning forest
- maximal matching

I/O unfriendly (traversal) algorithms

- breadth-first search
- shortest paths
- transitive closure
- diameter

and anything else you really want to know

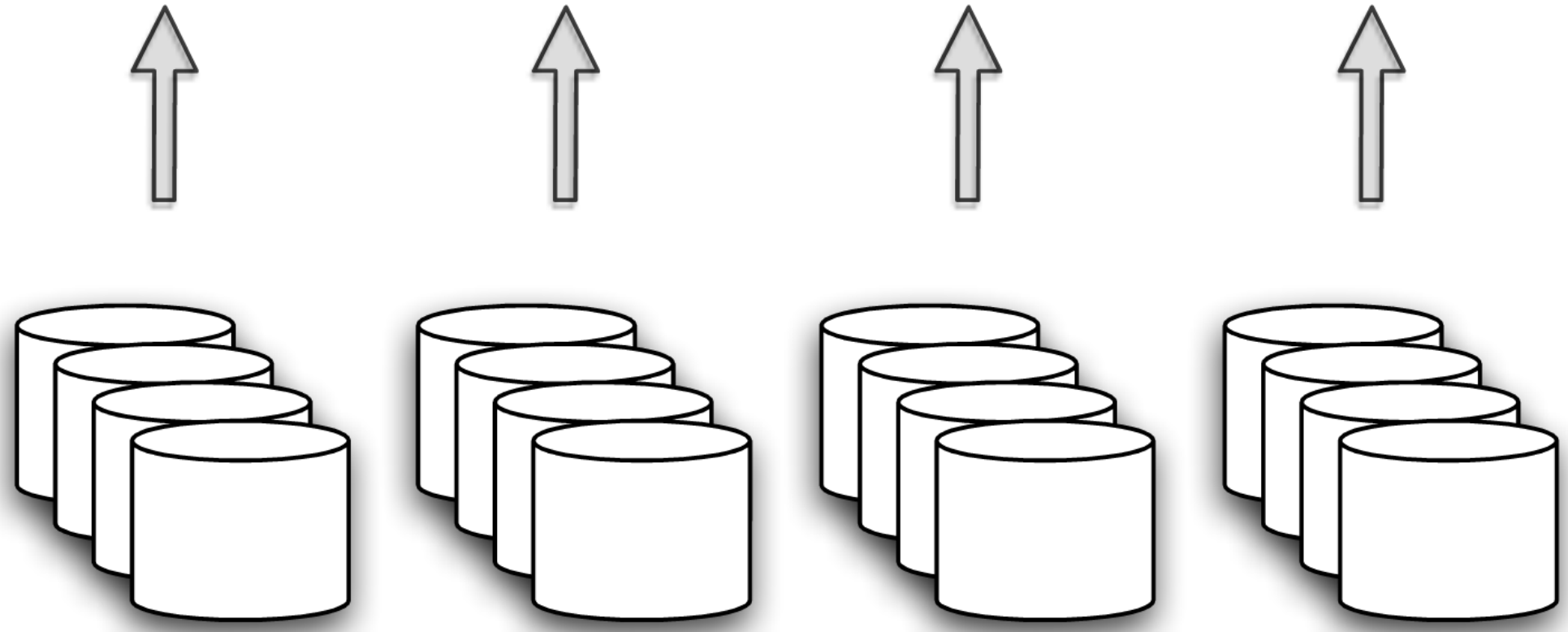
$400

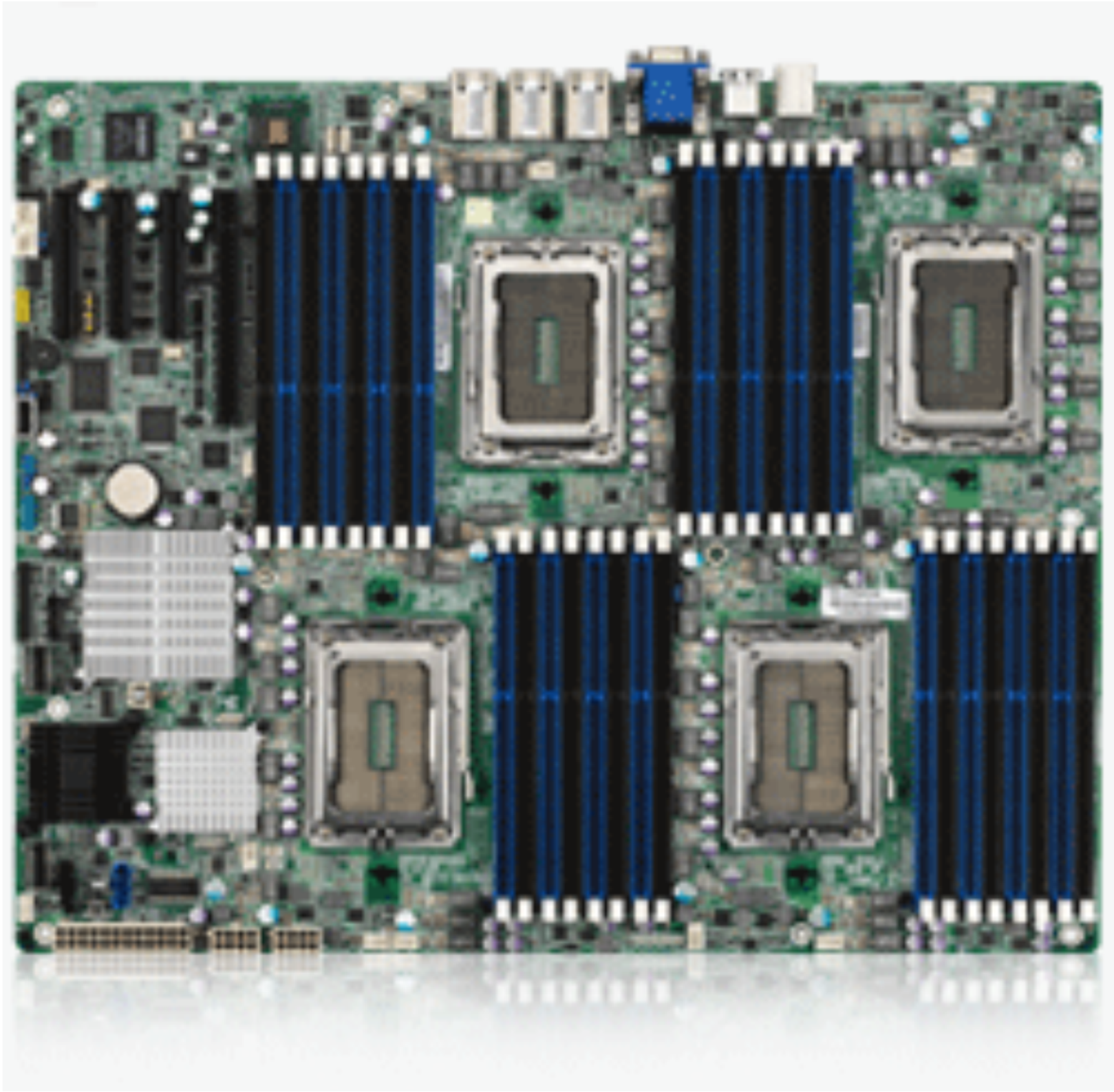

OCZ
Technology
ocztechnology.com
VERTEX 3
2.5" solid state drive

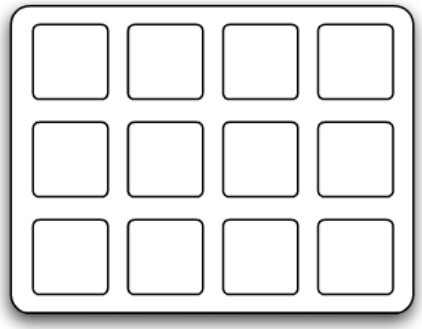
$7600

1M Random Read IOPS

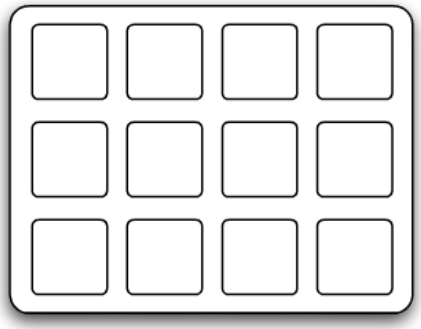
>800K IOPS
(random read)



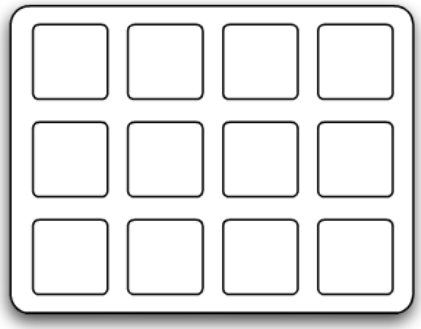




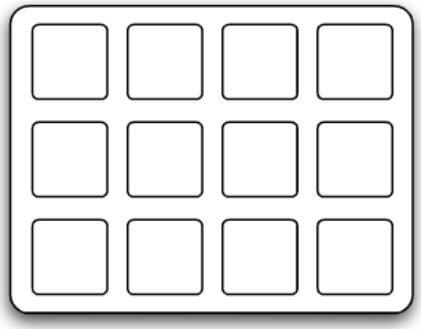
6M pages/sec



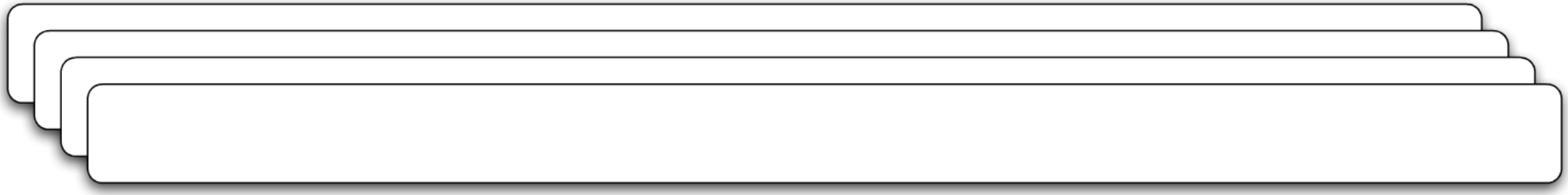
6M pages/sec

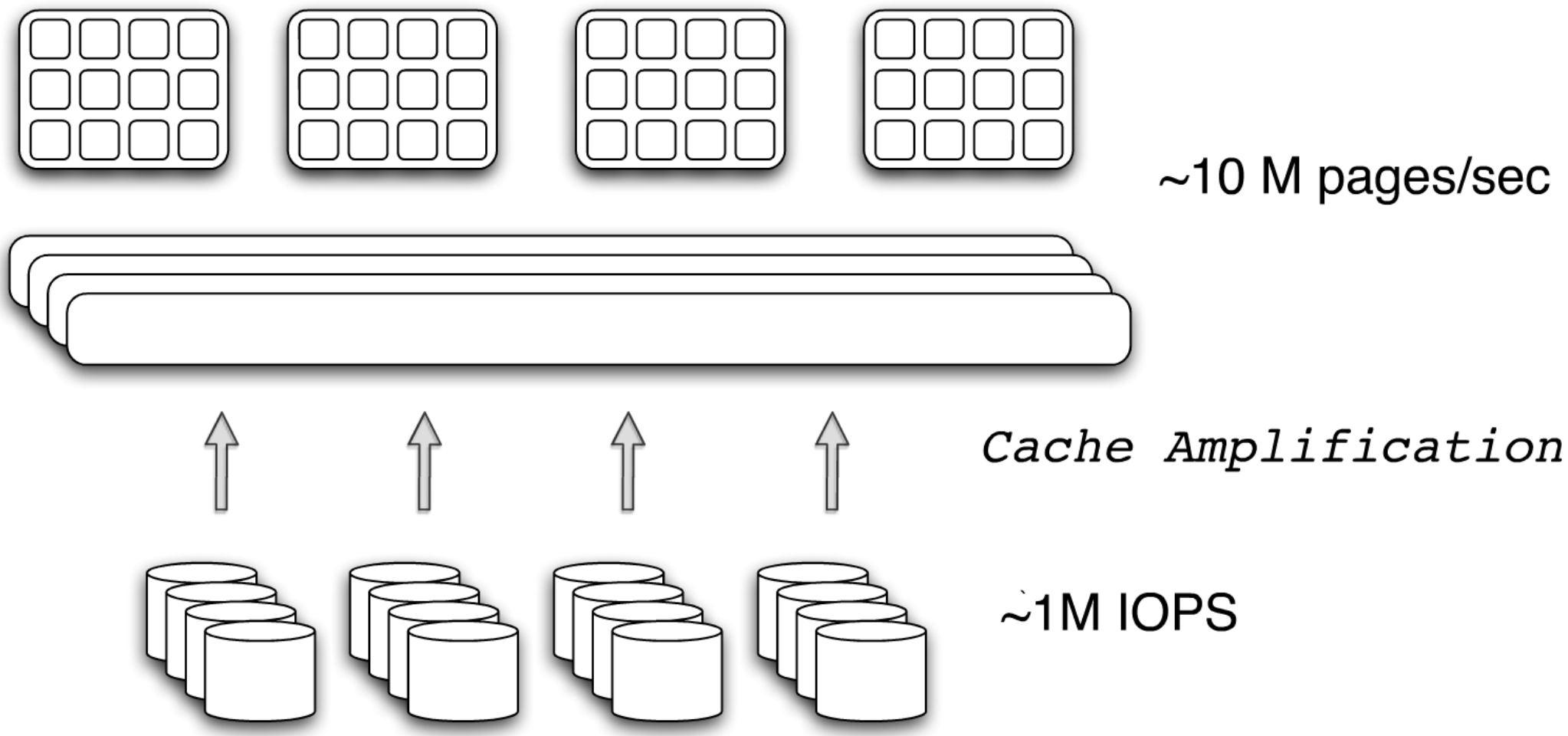


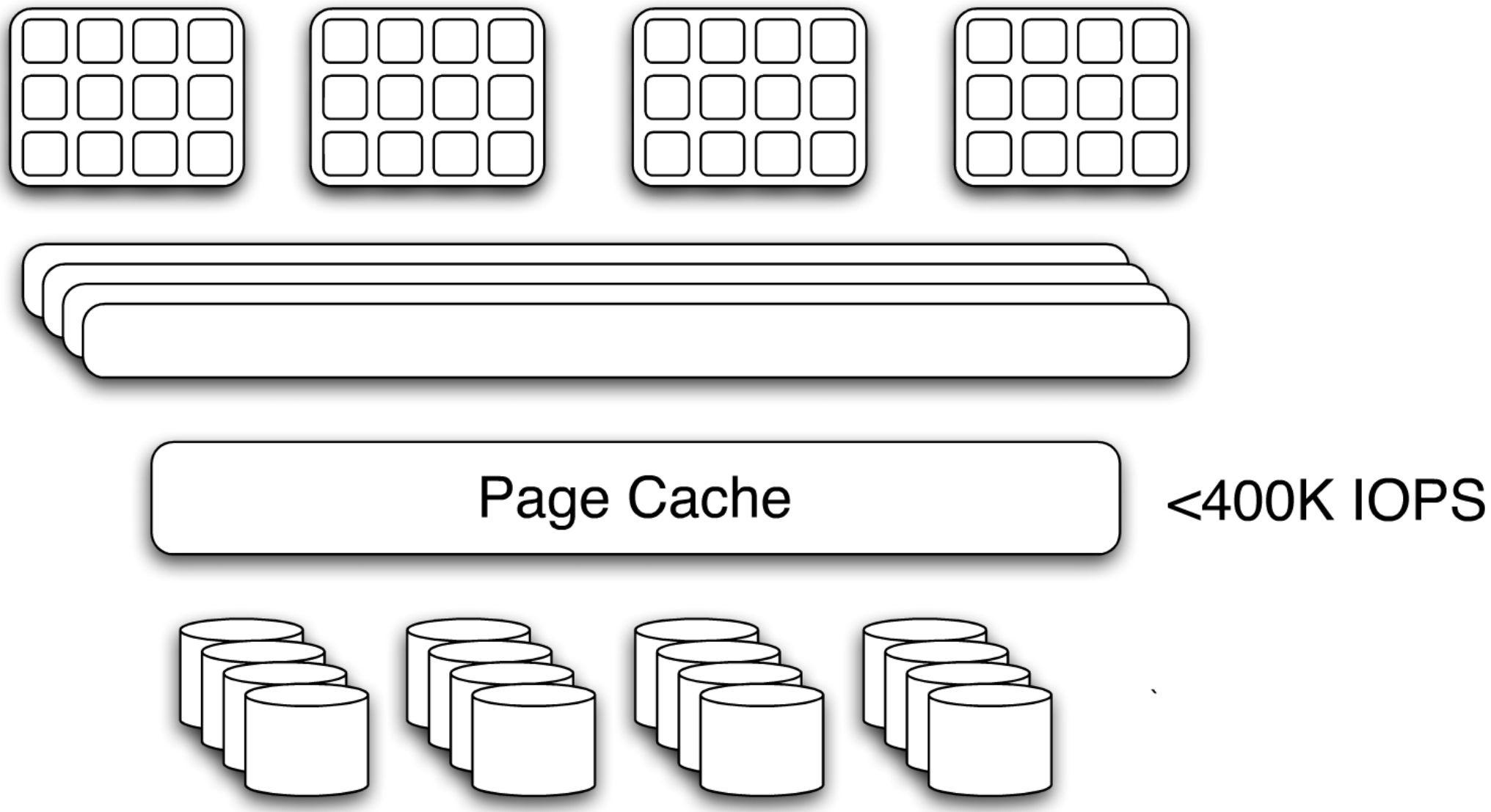
6M pages/sec



6M pages/sec




~10 M pages/sec
Cache Amplification
~1M IOPS


Page Cache
<400K IOPS

What went wrong?

There's a lot of bookkeeping

Address space:

Lists:
- clean
- dirty
- locked

RB Tree:
- page maps

Page cache:

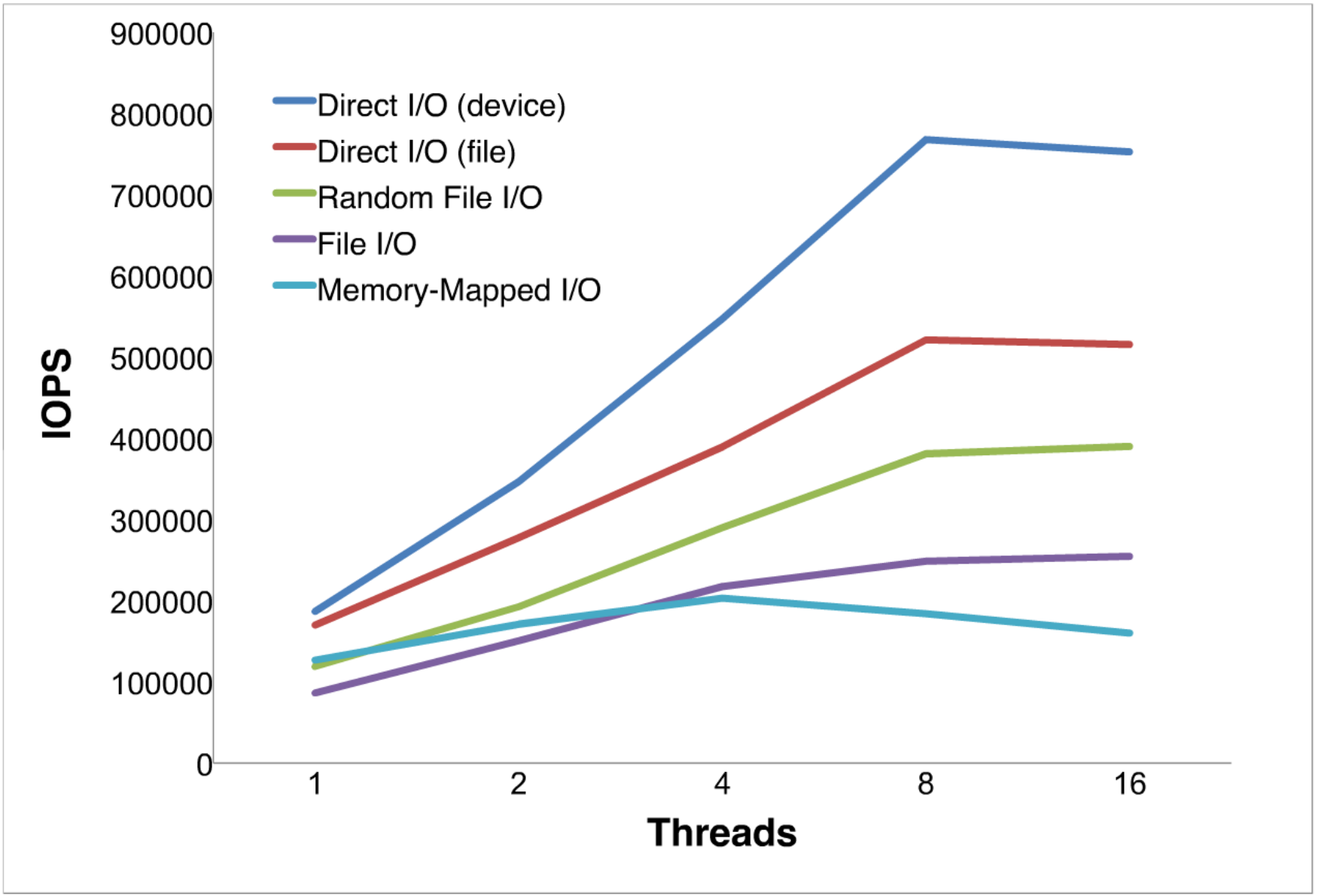
Page hash table

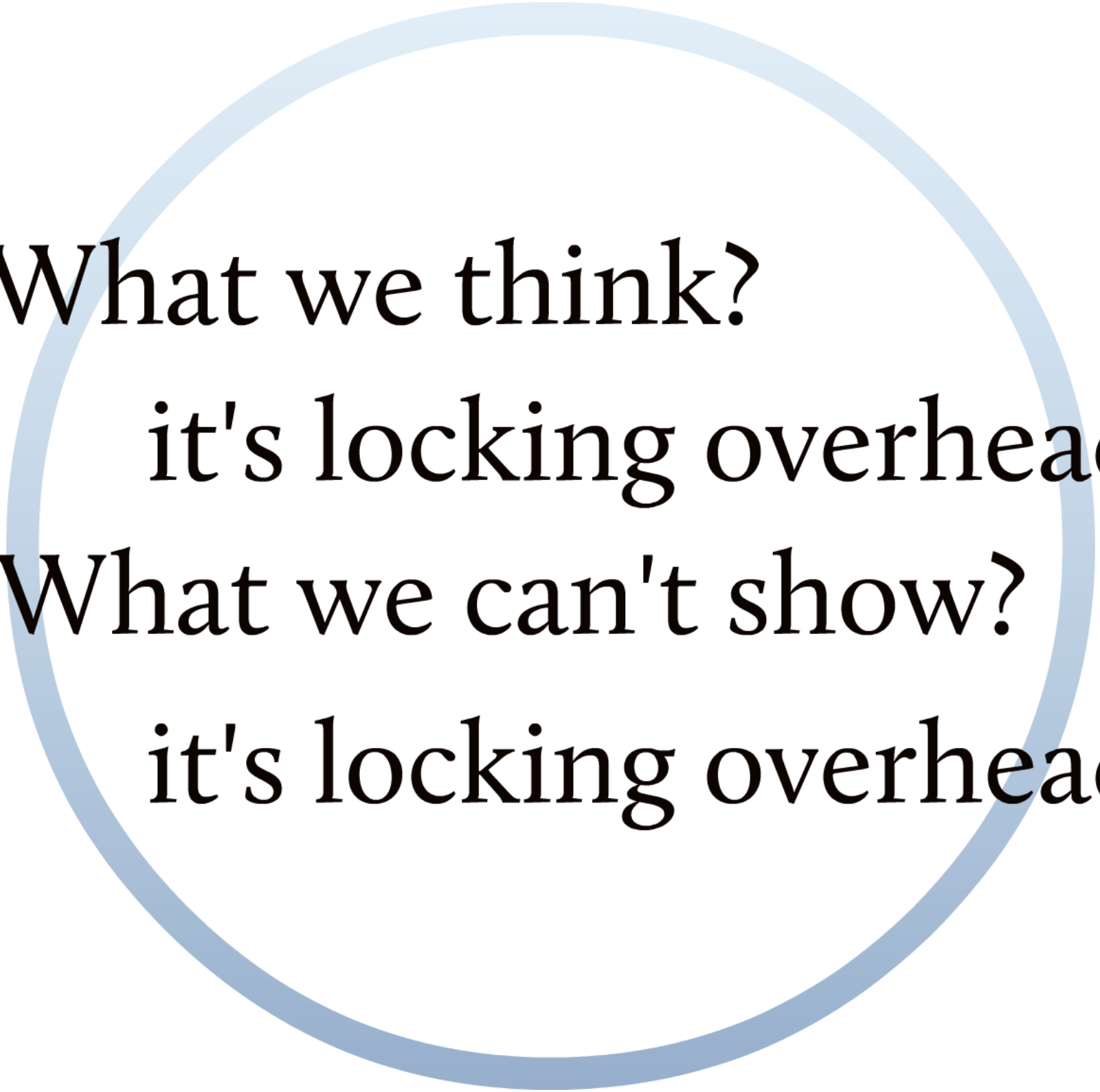
Buffer cache:

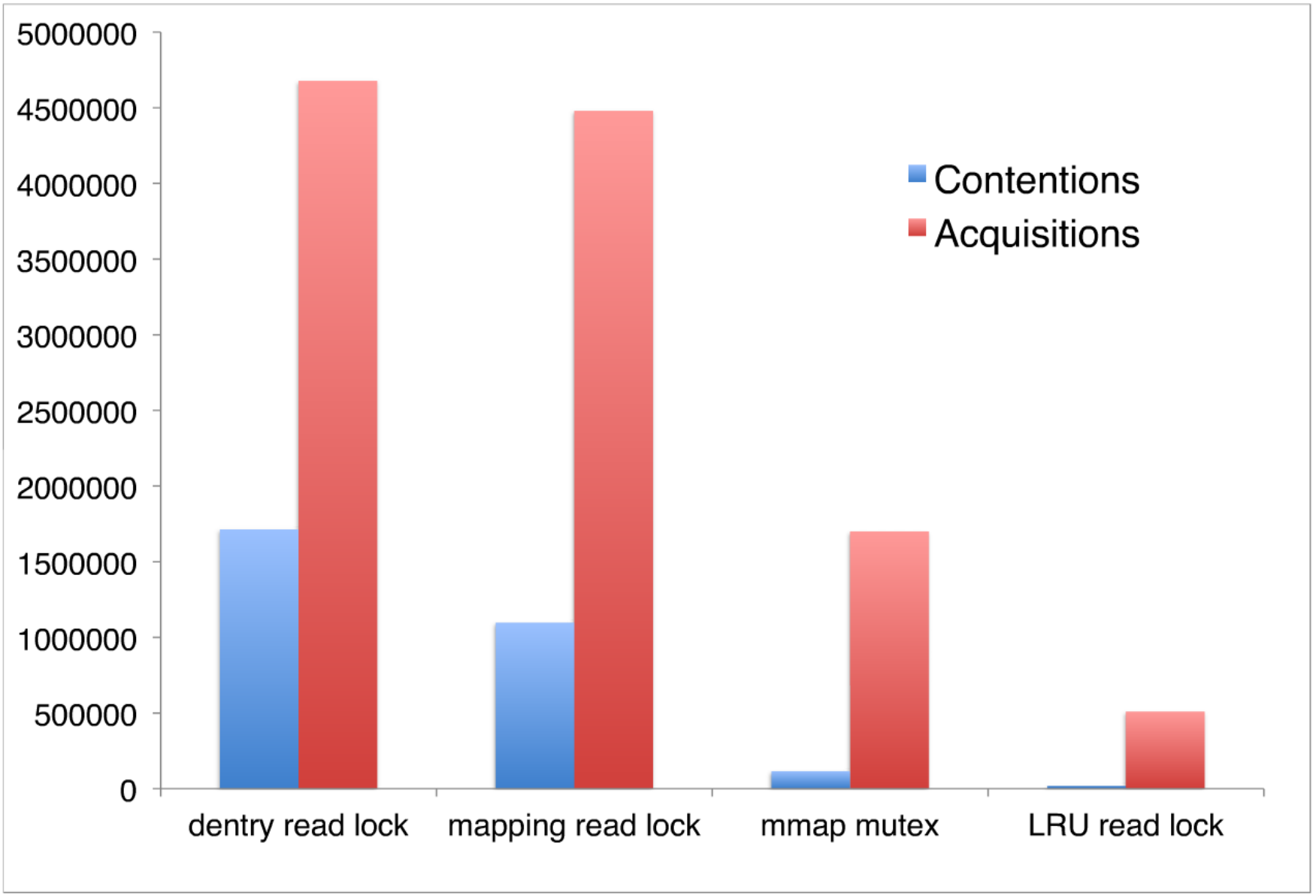
Lists:
- clean
- dirty
- locked
- shared
- unshared

Buffer hash table

....and locks for all


900000
800000
700000
600000
500000
400000
300000
200000
100000
0
IOPS
Direct I/O (device)
Direct I/O (file)
Random File I/O
File I/O
Memory-Mapped I/O
1
2
4
8
16
Threads


What we think?
it's locking overhead
What we can't show?
it's locking overhead

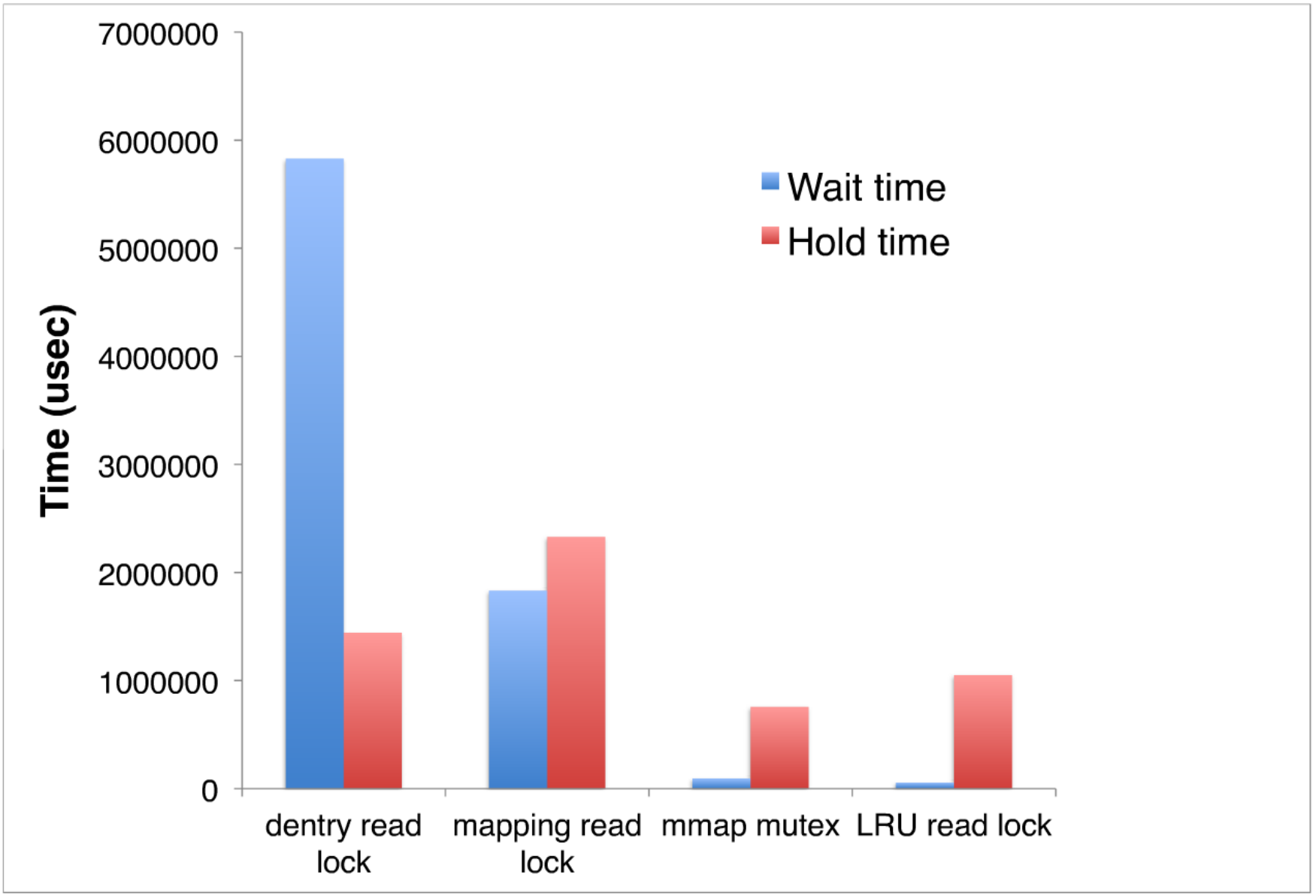

5000000
4500000
4000000
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
Contentions
Acquisitions
dentry read lock
mapping read lock
mmap mutex
LRU read lock

Locks contention is moderate

- wait times are small

But, locking needs expensive cache coherency

- when requested from multiple cores
- we are performing "knock out" experiments to verify lock overheads

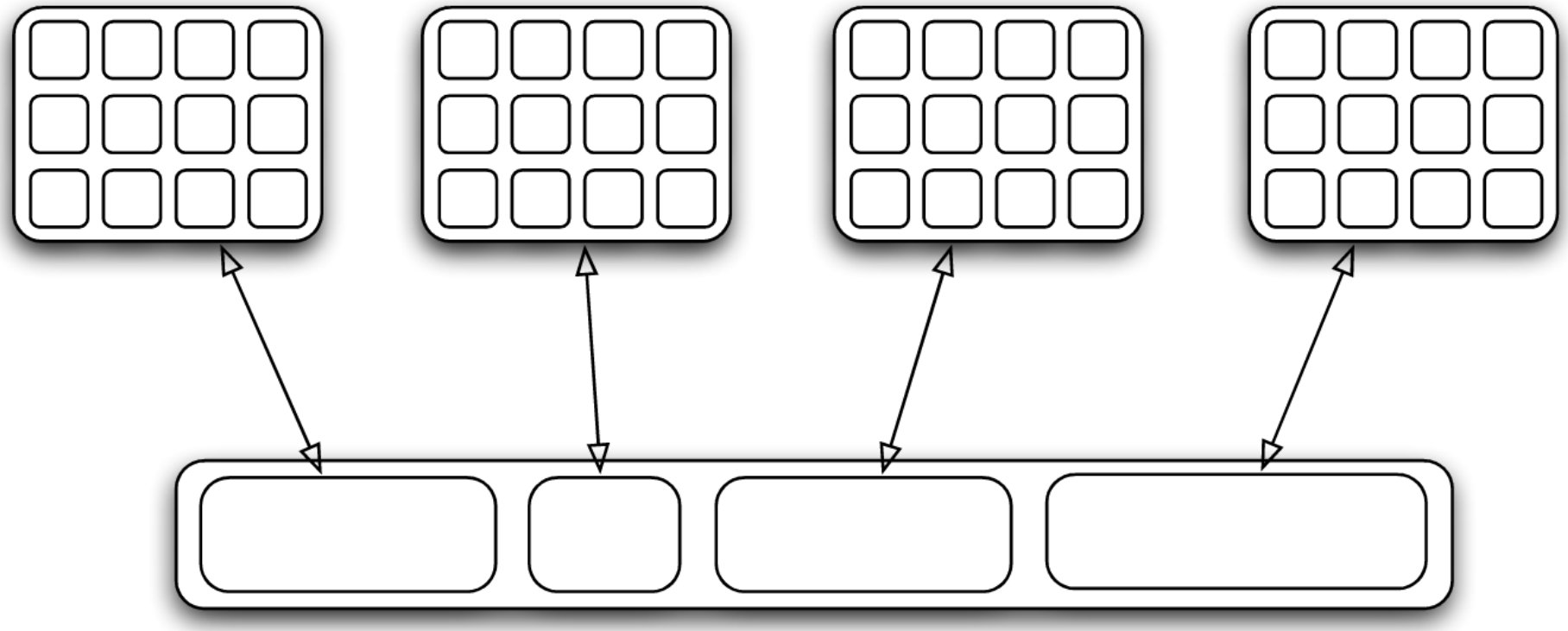

7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
Time (usec)
Wait time
Hold time
dentry read lock
mapping read lock
mmap mutex
LRU read lock

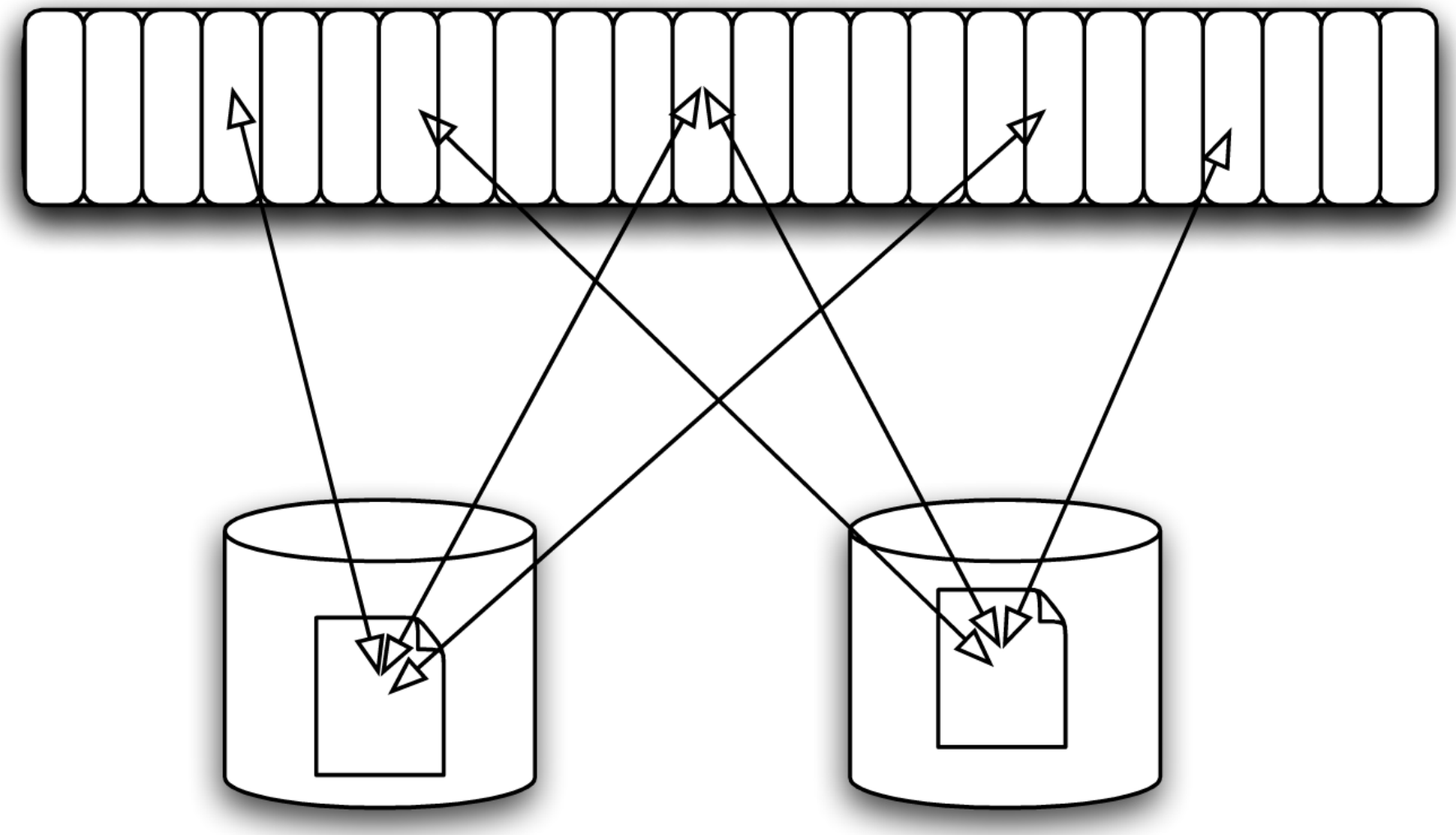
to verify lock overheads

Directions

- reduce lock intersection (decluster locks)
- speculative lock elision (elimintae locks)
- move away from global data structures

Partitioning





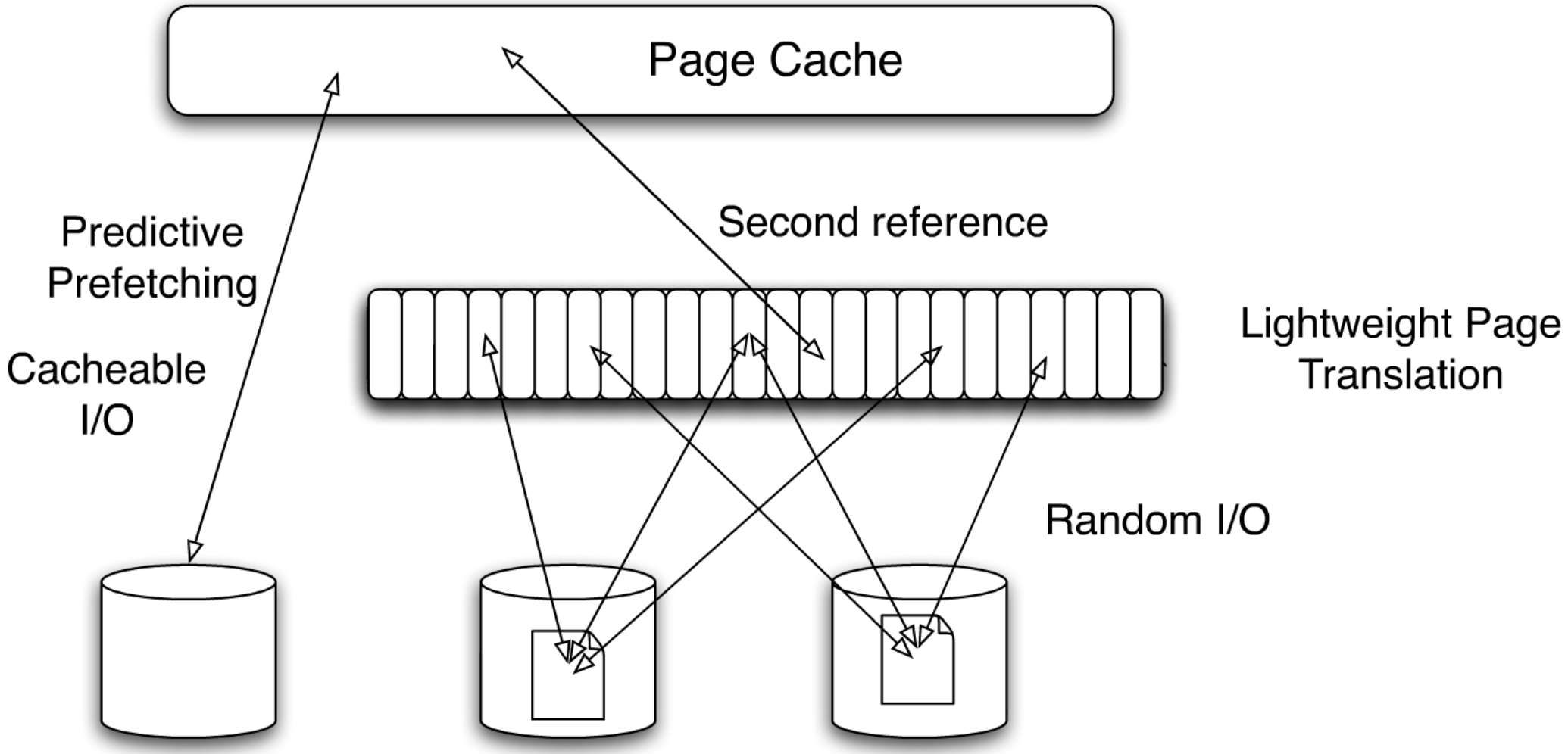
Associativity

Both techniques used in processor caches.

Something more interesting to do?

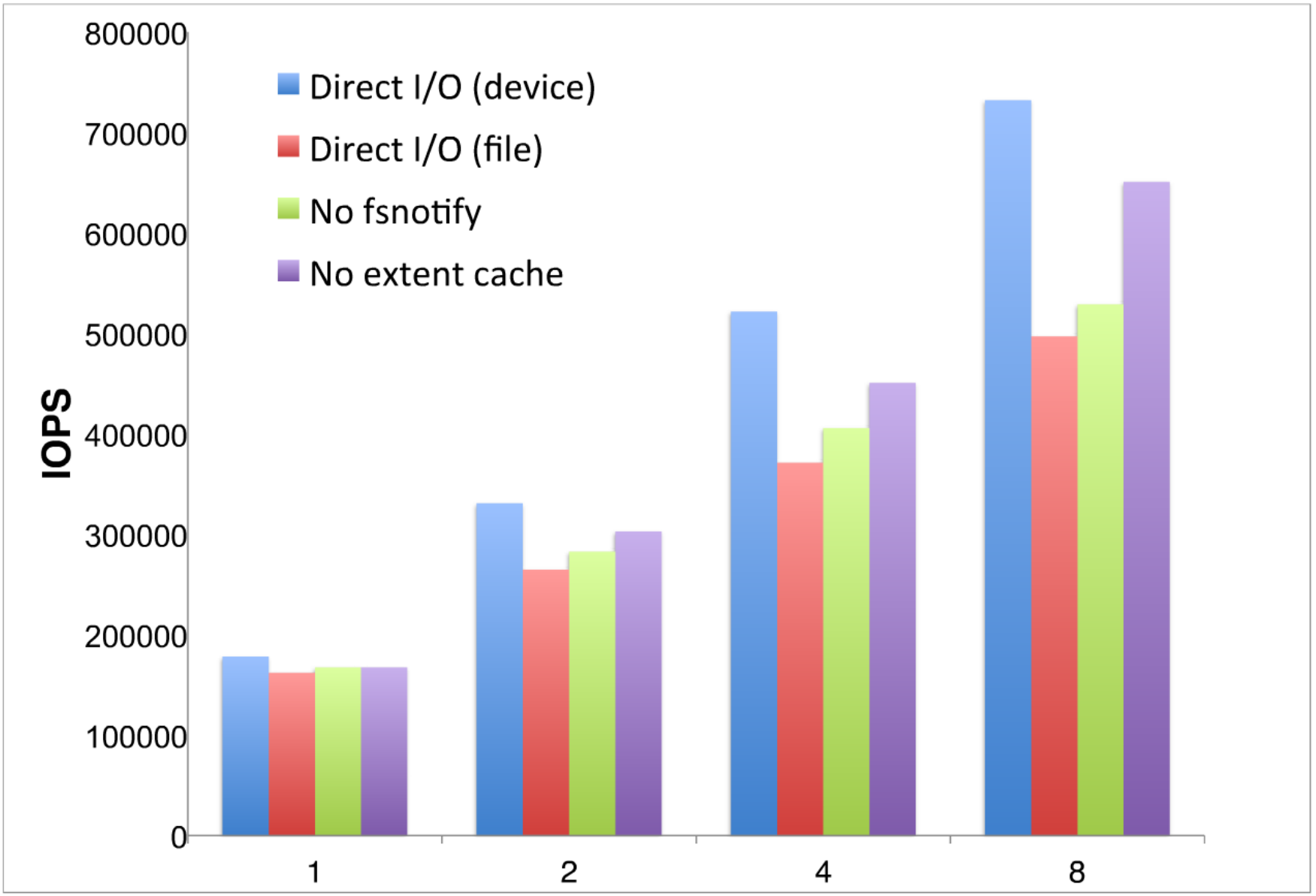
Investment Strategies

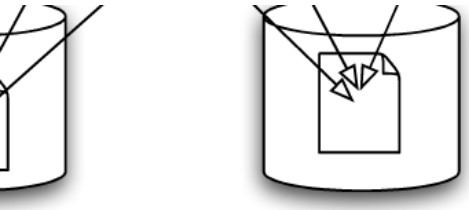
Promote pages on second hit
Old idea (2Q-LRU, LIRS, ARC)
but about investing cycles, not relpacement


Page Cache
Predictive
Prefetching
Second reference
Cacheable
I/O
Lightweight Page
Translation
Random I/O

In this manner, we can accelerate direct file I/O

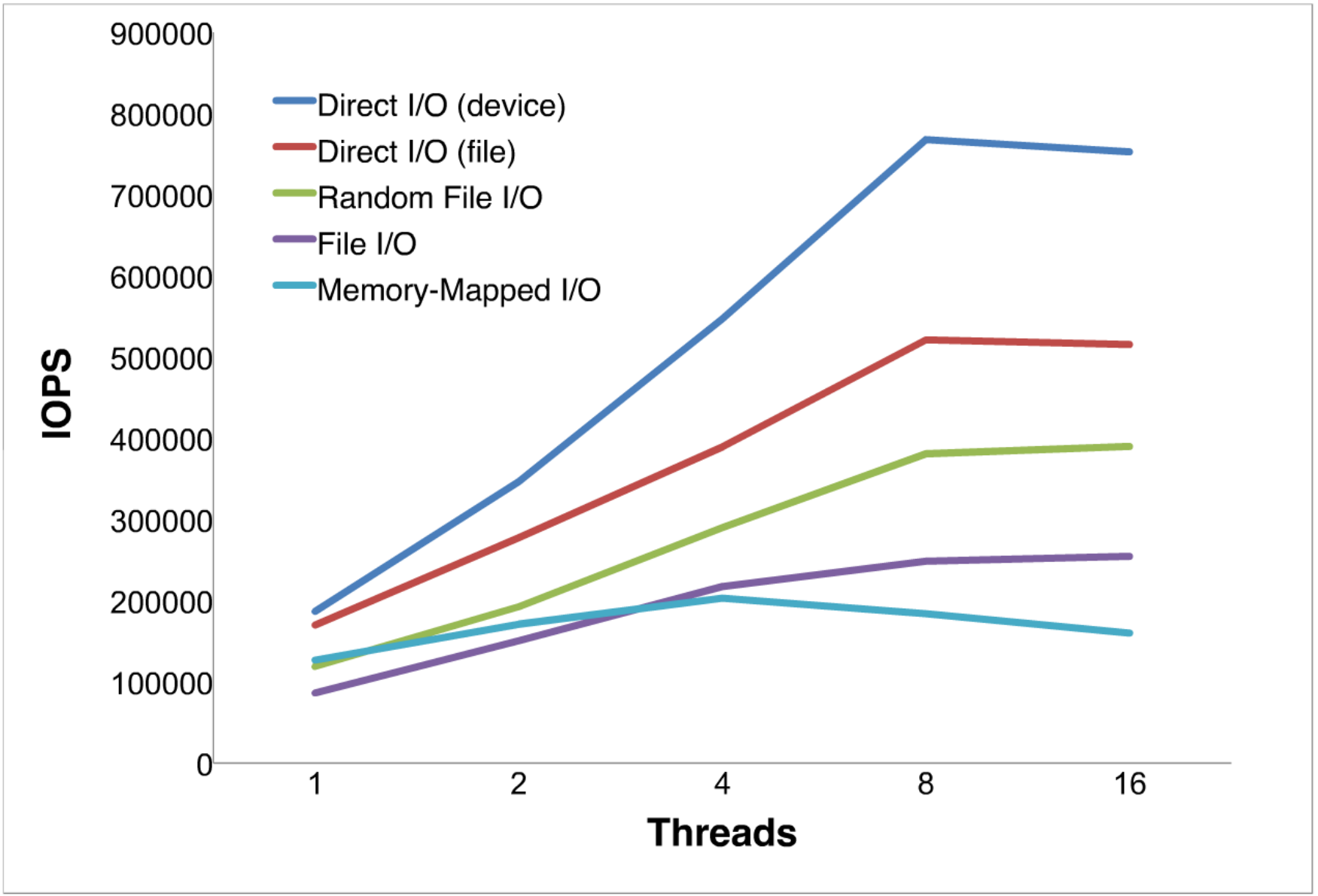
- Reduces interference (not overhead)


800000
700000
600000
500000
400000
300000
200000
100000
0
IOPS
Direct I/O (device)
Direct I/O (file)
No fsnotify
No extent cache
1
2
4
8



Thought for food

- Are there large graphs? (that you care about?)
- Is workload locality decreasing?
- Are random read IOPS a performance limiting factor for analysis engines?


900000
800000
700000
600000
500000
400000
300000
200000
100000
0
IOPS
Direct I/O (device)
Direct I/O (file)
Random File I/O
File I/O
Memory-Mapped I/O
1
2
4
8
16
Threads