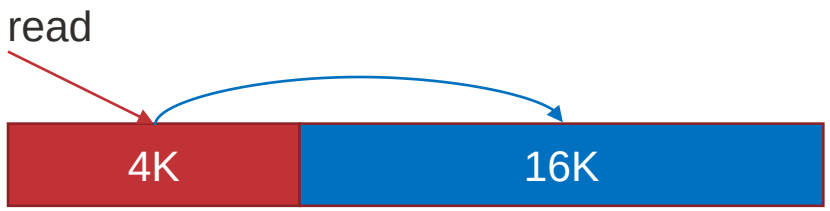
I/O Prefetch Cache as QEMU Block Filter Driver

Pavel Butsykin
Virtuozzo, inc.
KVM Forum 2016

Prefetch cache

- Idea – Fetch the data before it is needed
- There are different types of prefetch cache that focus on different read patterns
- Prefetching sequential data read (read-ahead) is one of the types of such cache

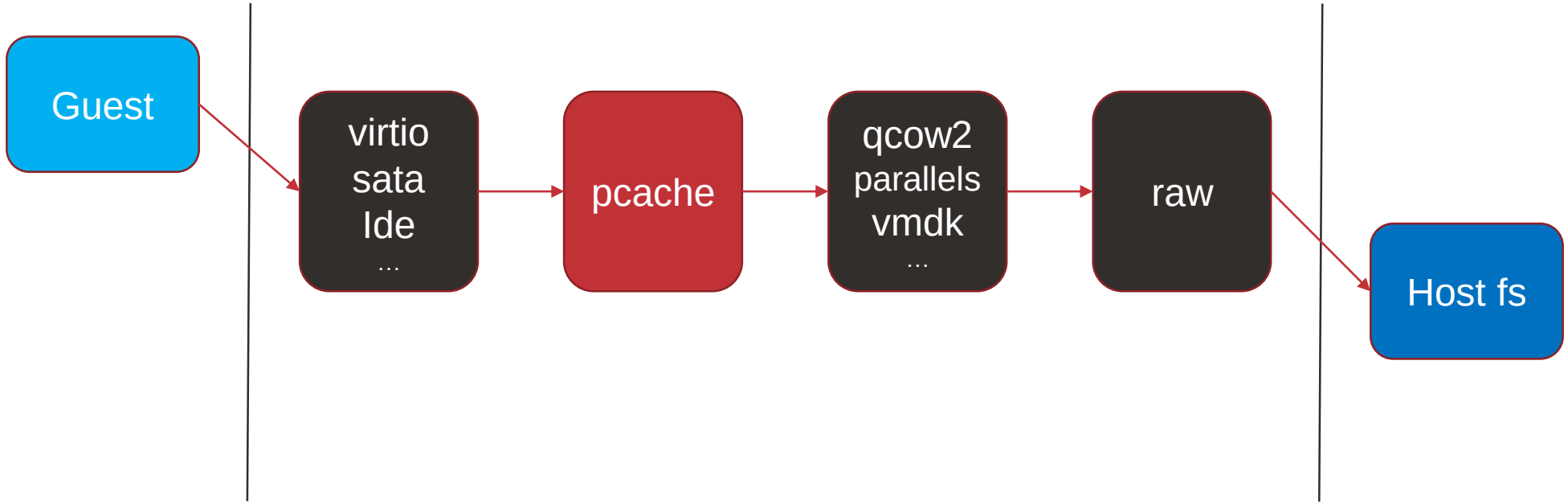
Read-ahead



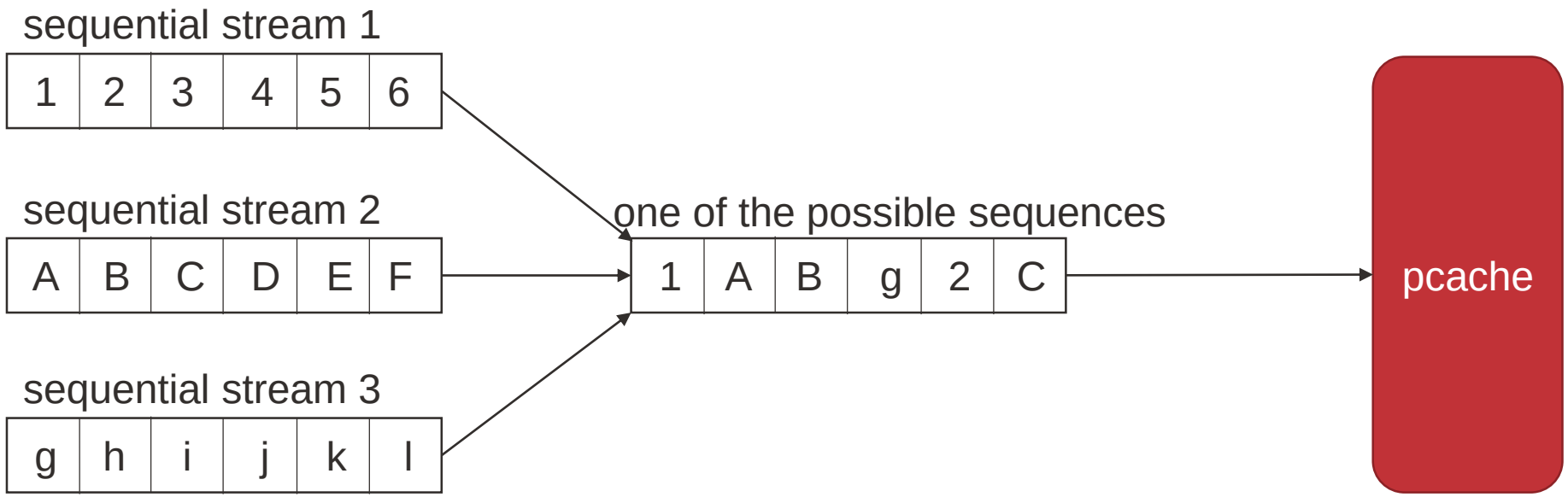
- Random access
  - can displace a lot of cache
  - can lead to performance degradation

- Sequential read detection?

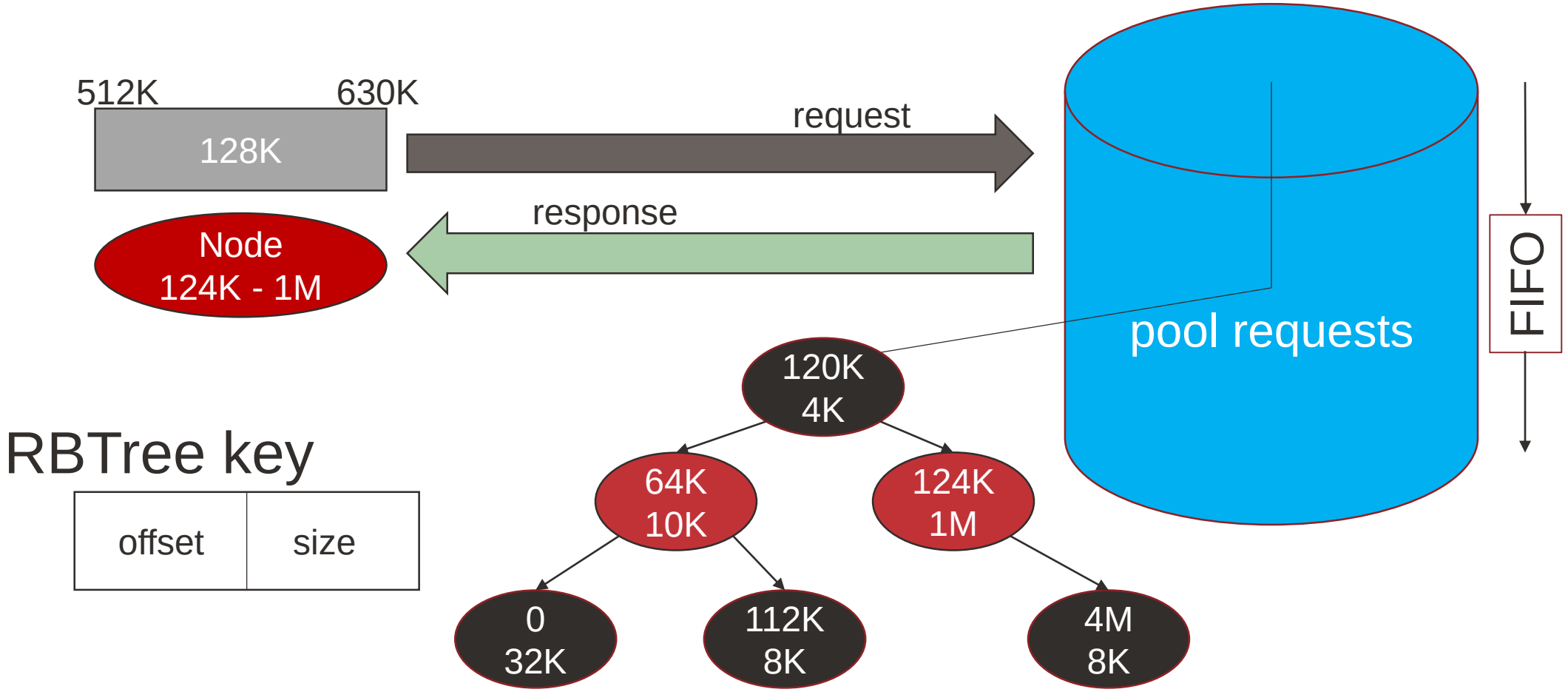
QEMU Block Filter Driver



Parallel and sequential read



Sequential read detection

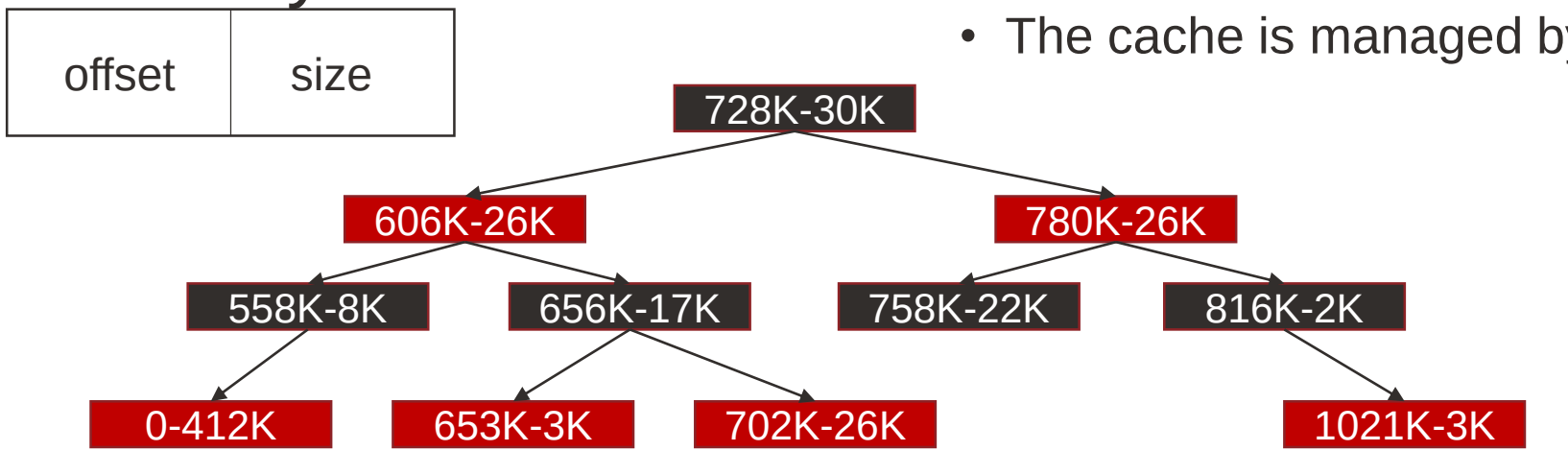


PCache AIO read overview

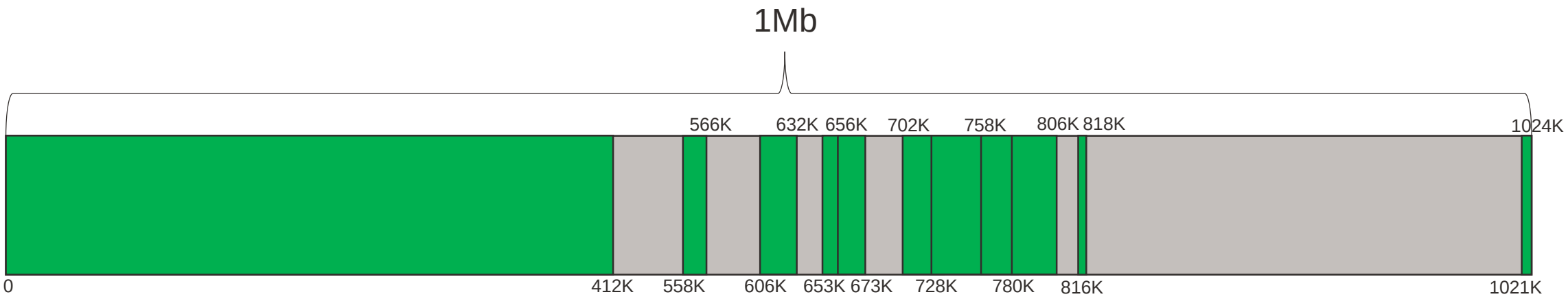
- Skip large requests (by default larger then 64Kb)
- Update request statistics
- Cache lookup
  - hit
  - partial hit
  - miss
- Read-ahead
  - check request sequence
  - read into cache a chunk of data form the end of the current request

Cache memory

RBTree key

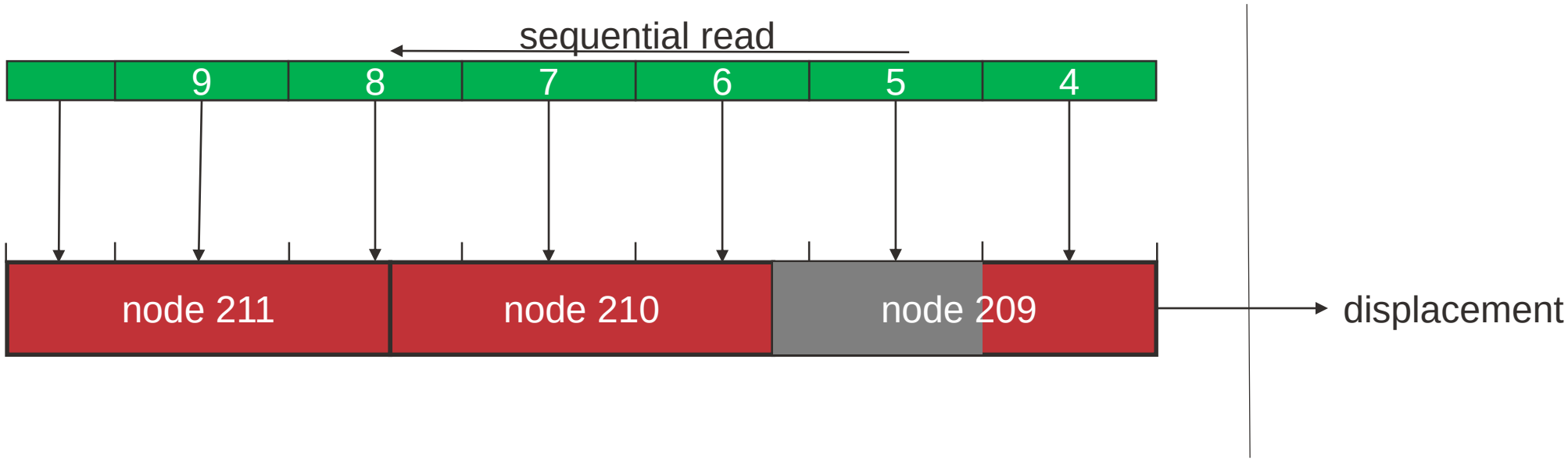


- The cache memory has limited size (4Mb by default)
- The cache is managed by LRU algorithm

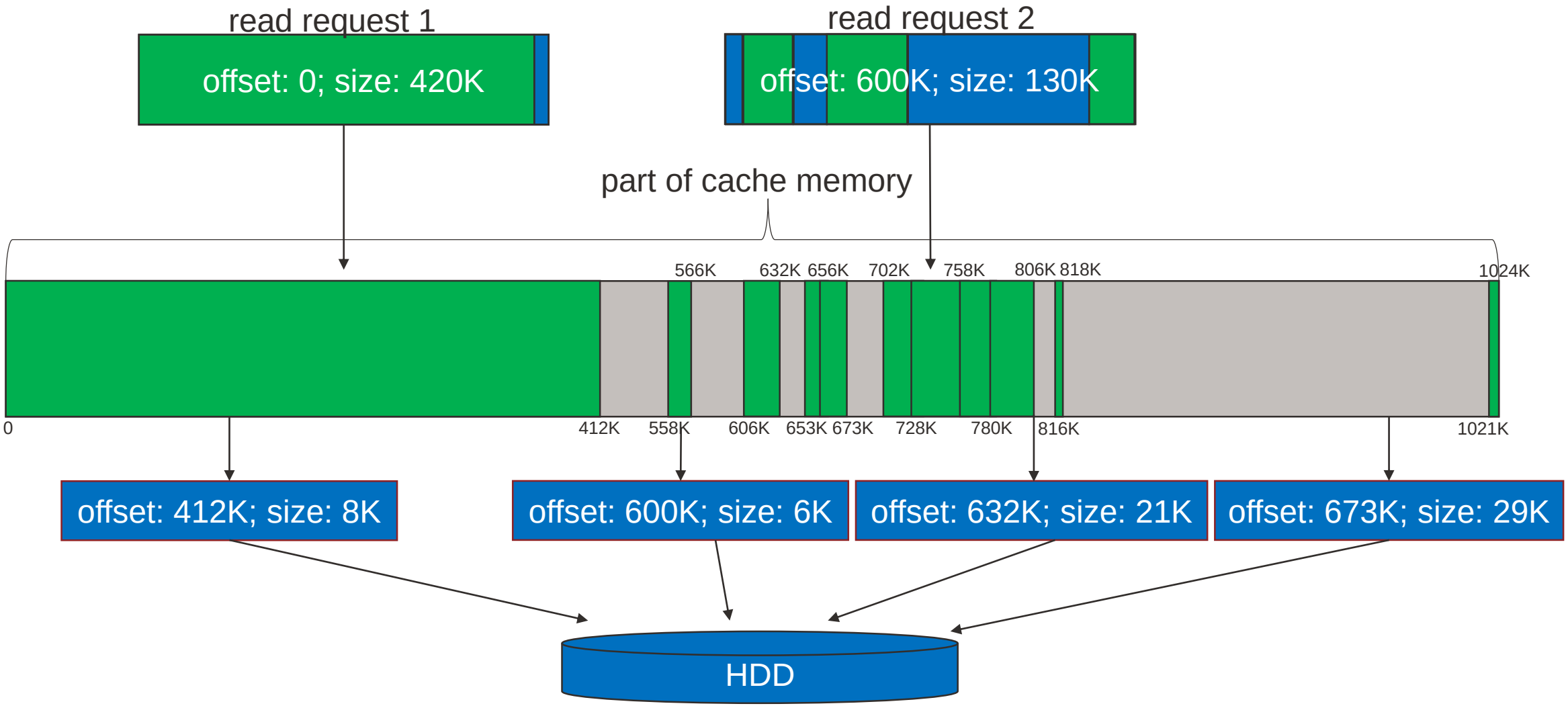


Why you need LRU for the prefetch cache?

If you have read one part of the node, then there is a high probability that you will soon read the remaining parts of the node.



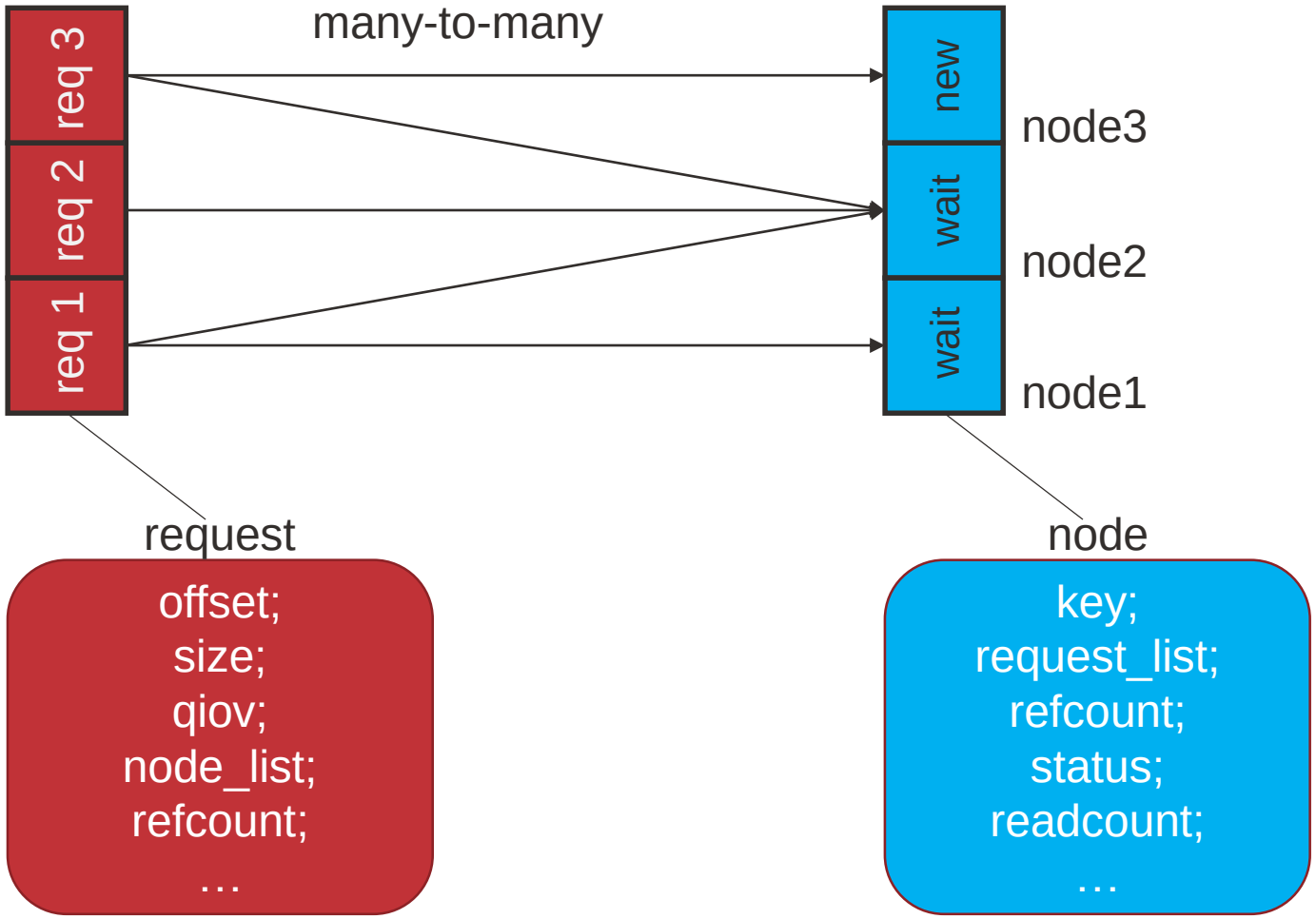
Partial cache hit



PCache AIO write overview

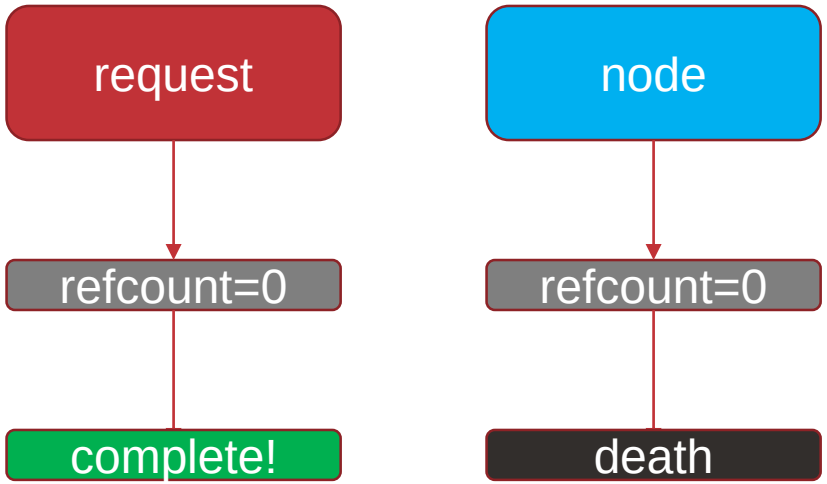
- Drop all nodes intersecting with request
- Write-through

Rescheduling AIO requests

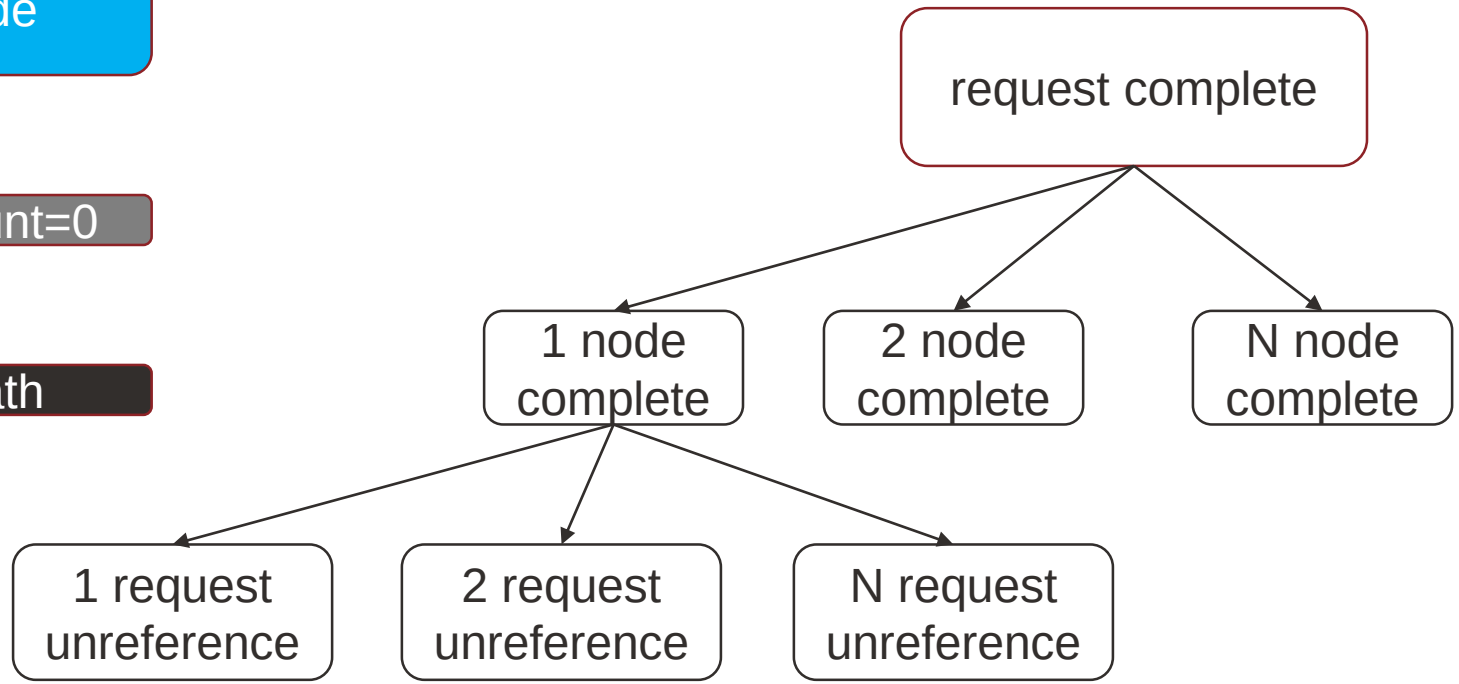


What to do if the requested node is in-flight?

PCache AIO request complete

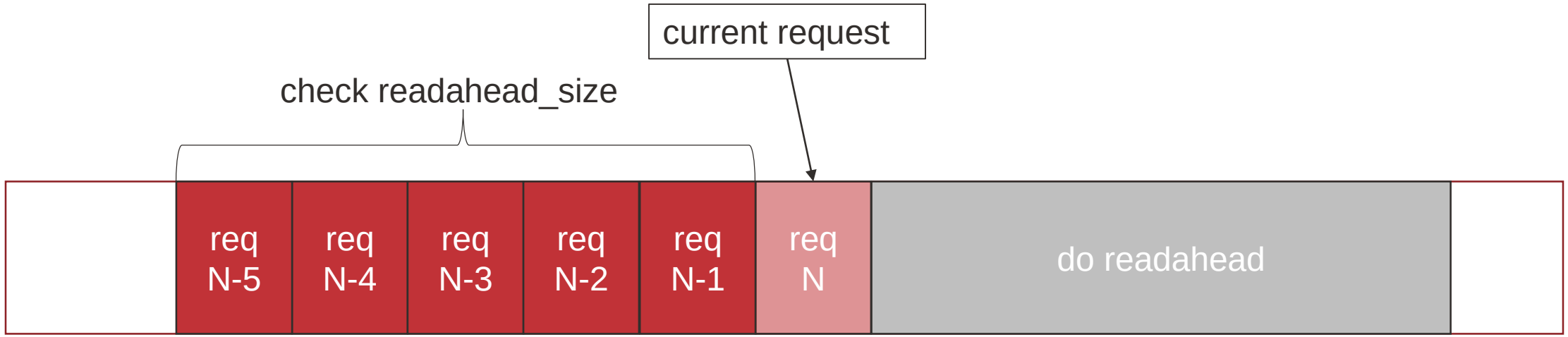


How to complete an request which expects other requests ?



Read-ahead policy

- Original requests are not written to the cache and only serve to update statistics
- Filtering of large requests helps to detect sequential read
- If part of the readahead is already in the cache then only the missing pieces will be fetched from disk



	SSDSC2BW120A4 EXT4	QEMU 2.6.50 VirtIO qcow2 Linux 4.4.0 Fedora-22 2SMP 2GB VM	QEMU 2.6.50 VirtIO qcow2 Linux 4.4.0 Fedora-22 2SMP 2GB VM + dataplane	QEMU 2.6.50 VirtIO qcow2 Linux 4.4.0 Fedora-22 2SMP 2GB VM + pcache		QEMU 2.6.50 VirtIO qcow2 Linux 4.4.0 Fedora-22 2SMP 2GB VM + dataplane + pcache		
# ↓	I/O Test	#1	vs #1	vs #1	vs #2	vs #1	vs #2	vs #3
1	2G-read-seq-4K(01)	100%	+10.7%	+341%	+298%	+490%	+433%	+33.7%
2	2G-read-seq-4K(04)	100%	+5.0%	+231%	+215%	+225%	+210%	-1.7%
3	2G-read-seq-4K(16)	100%	+0.1%	+80.3%	+80.1%	+72.1%	+71.9%	-4.6%
4	2G-read-seq-4K-AIO4(01)	100%	+43.3%	+190%	+102%	+191%	+103%	+0.2%
5	2G-read-seq-4K-AIO4(04)	100%	+1.4%	+79.0%	+76.6%	+78.2%	+75.8%	-0.4%
6	2G-read-seq-4K-AIO4(16)	100%	+0.6%	+77.6%	+76.5%	+70.3%	+69.3%	-4.1%
7	2G-read-seq-4K-AIO32(01)	100%	-7.8%	-0.8%	+7.6%	+1.2%	+9.7%	+2.0%
8	2G-read-seq-4K-AIO32(04)	100%	-4.4%	+21.8%	+27.4%	+24.9%	+30.7%	+2.6%
9	2G-read-seq-4K-AIO32(16)	100%	-3.0%	+46.4%	+50.8%	+46.7%	+51.1%	+0.2%
10	2G-read-rnd-4K(01)	100%	+1.1%	+3.0%	+1.9%	+1.9%	+0.9%	-1.0%
11	2G-read-rnd-4K(04)	100%	+2.5%	+0.4%	-2.0%	+1.7%	-0.8%	+1.3%
12	2G-read-rnd-4K(16)	100%	+7.1%	+5.4%	-1.6%	+5.7%	-1.3%	+0.3%
13	2G-read-rnd-4K-AIO4(01)	100%	+1.2%	-0.8%	-1.9%	+1.5%	+0.3%	+2.3%
14	2G-read-rnd-4K-AIO4(04)	100%	+2.4%	+6.9%	+4.3%	+7.2%	+4.7%	+0.3%
15	2G-read-rnd-4K-AIO4(16)	100%	-0.5%	+4.9%	+5.4%	+4.1%	+4.6%	-0.8%
16	2G-read-rnd-4K-AIO32(01)	100%	+1.4%	+0.1%	-1.3%	+1.5%	+0.1%	+1.4%
17	2G-read-rnd-4K-AIO32(04)	100%	+2.9%	+0.4%	-2.5%	+3.1%	+0.1%	+2.7%
18	2G-read-rnd-4K-AIO32(16)	100%	-4.6%	-0.5%	+4.3%	+0.6%	+5.4%	+1.1%

Virtuozzo

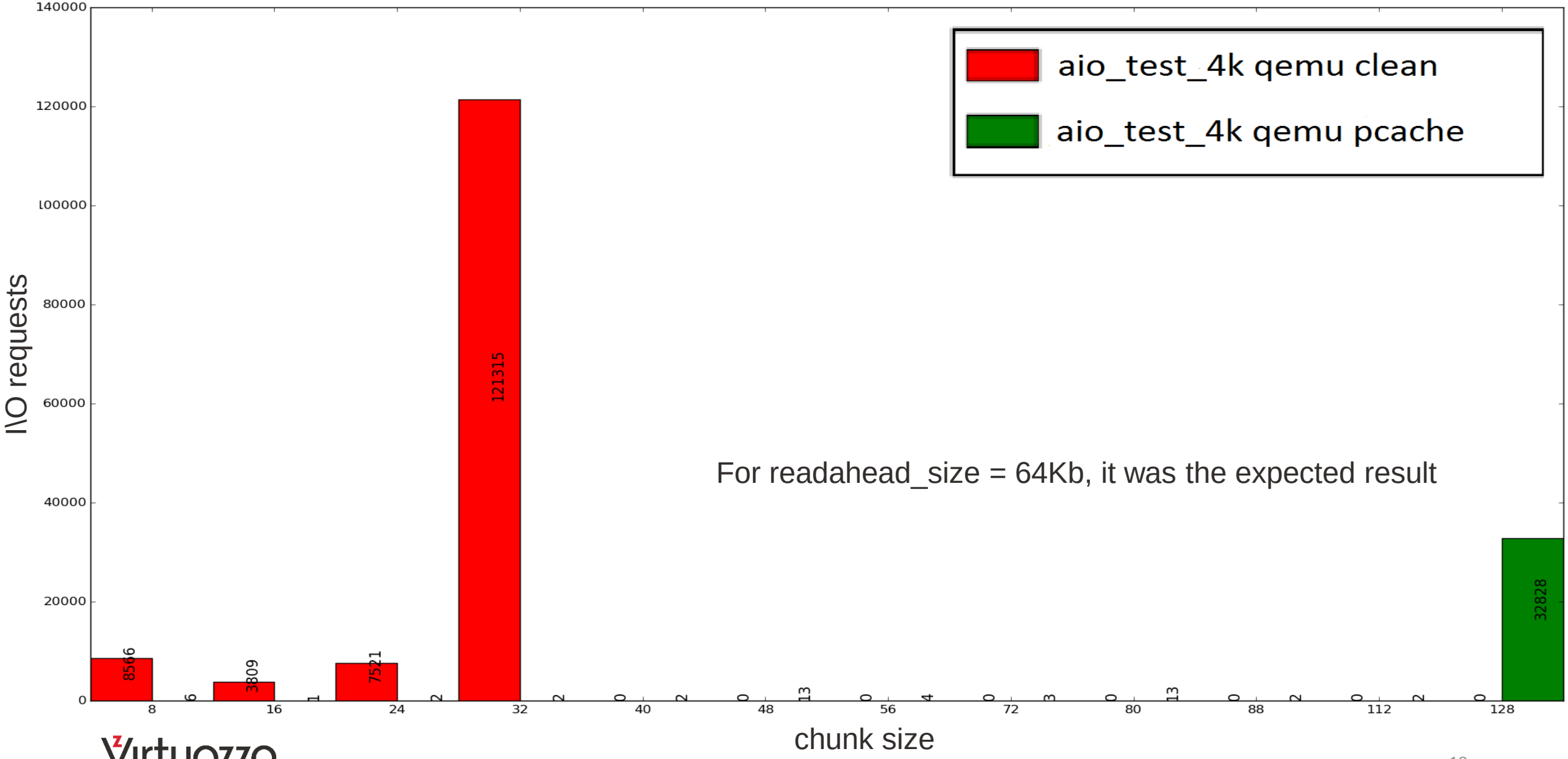
Read directory

	SSDSC2BW120A4 EXT4	QEMU 2.6.50 VirtIO qcow2 Linux 4.4.0 Fedora-22 2SMP 2GB VM	QEMU 2.6.50 VirtIO qcow2 Linux 4.4.0 Fedora-22 2SMP 2GB VM + pcache	
# ↓	Test	Scores	Scores	vs #1
1	dir_readdir Testcase: create a directory and populate it with 10 subdirs and 10 files with max depth 3 once before test. Total: there are 10^3 dirs and 10^3 files.: All files are empty: (1) open() root dir, then readdir() recursively, close():	134	137	+1.9%

Qemu bench

```
$ ./qemu-img bench -d 1 -c 262144 -f qcow2 -s 4096 -S 4096 -t none ./image.qcow2
Sending 262144 read requests, 4096 bytes each, 1 in parallel (starting at offset 0, step size 4096)
Run completed in 19.594 seconds.
$ ./qemu-img bench -d 1 -c 262144 -f pcache -s 4096 -S 4096 -t none ./image.qcow2
Sending 262144 read requests, 4096 bytes each, 1 in parallel (starting at offset 0, step size 4096)
Run completed in 4.378 seconds.
$ ./qemu-img bench -d 8 -c 262144 -f qcow2 -s 4096 -S 4096 -t none ./image.qcow2
Sending 262144 read requests, 4096 bytes each, 8 in parallel (starting at offset 0, step size 4096)
Run completed in 5.933 seconds.
$ ./qemu-img bench -d 8 -c 262144 -f pcache -s 4096 -S 4096 -t none ./image.qcow2
Sending 262144 read requests, 4096 bytes each, 8 in parallel (starting at offset 0, step size 4096)
Run completed in 4.356 seconds.
$ ./qemu-img bench -d 64 -c 262144 -f qcow2 -s 4096 -S 4096 -t none ./image.qcow2
Sending 262144 read requests, 4096 bytes each, 64 in parallel (starting at offset 0, step size 4096)
Run completed in 4.659 seconds.
$ ./qemu-img bench -d 64 -c 262144 -f pcache -s 4096 -S 4096 -t none ./image.qcow2
Sending 262144 read requests, 4096 bytes each, 64 in parallel (starting at offset 0, step size 4096)
Run completed in 4.204 seconds.
```

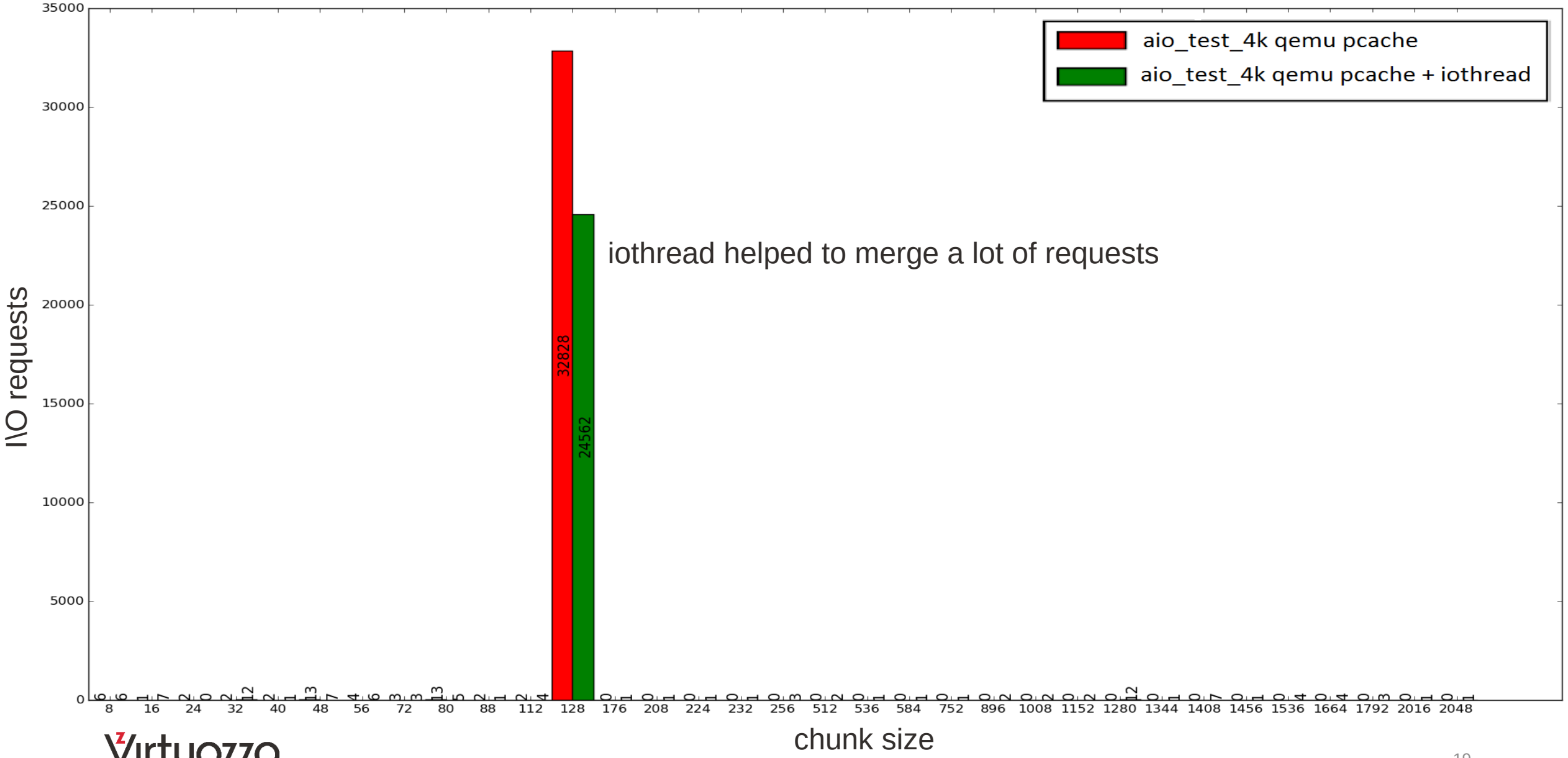
4K AIO read requests (pcache)



For readahead_size = 64Kb, it was the expected result

Virtuozzo

4K AIO read requests (pcache + iothread)



Virtuozzo

Conclusions

- PCache can optimize certain I/O patterns without pessimizing others
- PCache implementation in the form of the driver filter is unintrusive
- PCache is not universally useful, benchmark your patterns before enabling

Questions?