

# SMB Direct in Linux SMB kernel client

Long Li  
Microsoft

# Agenda

- Introduction to SMB Direct
- Transferring data with RDMA
- SMB Direct credit system
- Memory registration
- RDMA failure recovery
- Direct I/O
- Benchmarks
- Future work

# SMB Direct

- Transferring SMB packets over RDMA
  - Infiniband
  - RoCE (RDMA over Converged Ethernet)
  - iWARP (IETF RDMA over TCP)
- Introduced in SMB 3.0 with Windows 2012

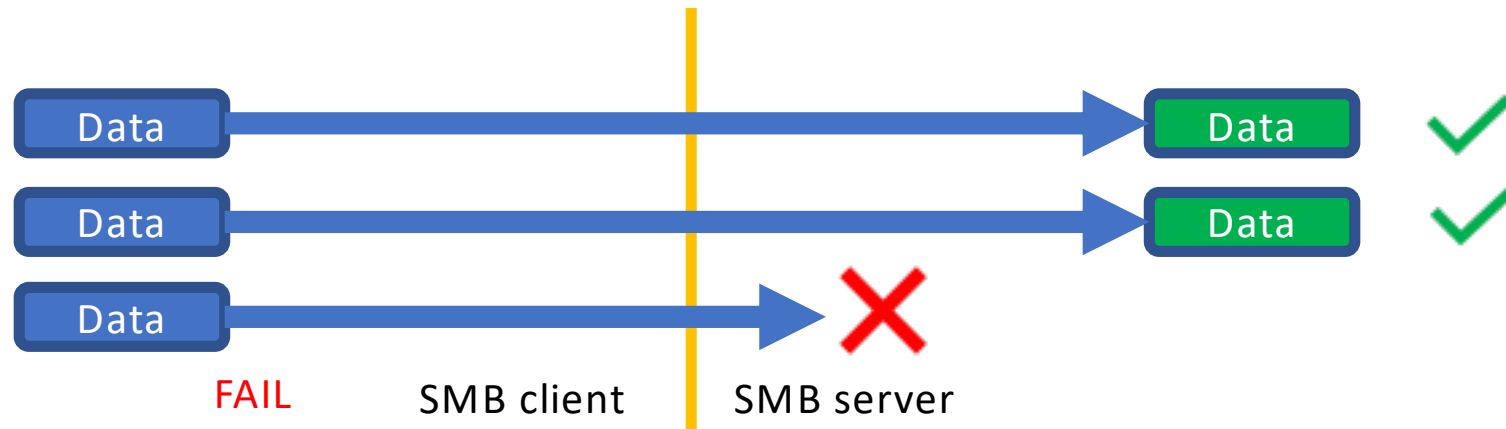
		New features
Windows Server 2012	SMB 3.0	SMB Direct
Windows Server 2012 R2	SMB 3.02	Remote invalidation
Windows Server 2016	SMB 3.1.1	

# Transfer data with SMB Direct

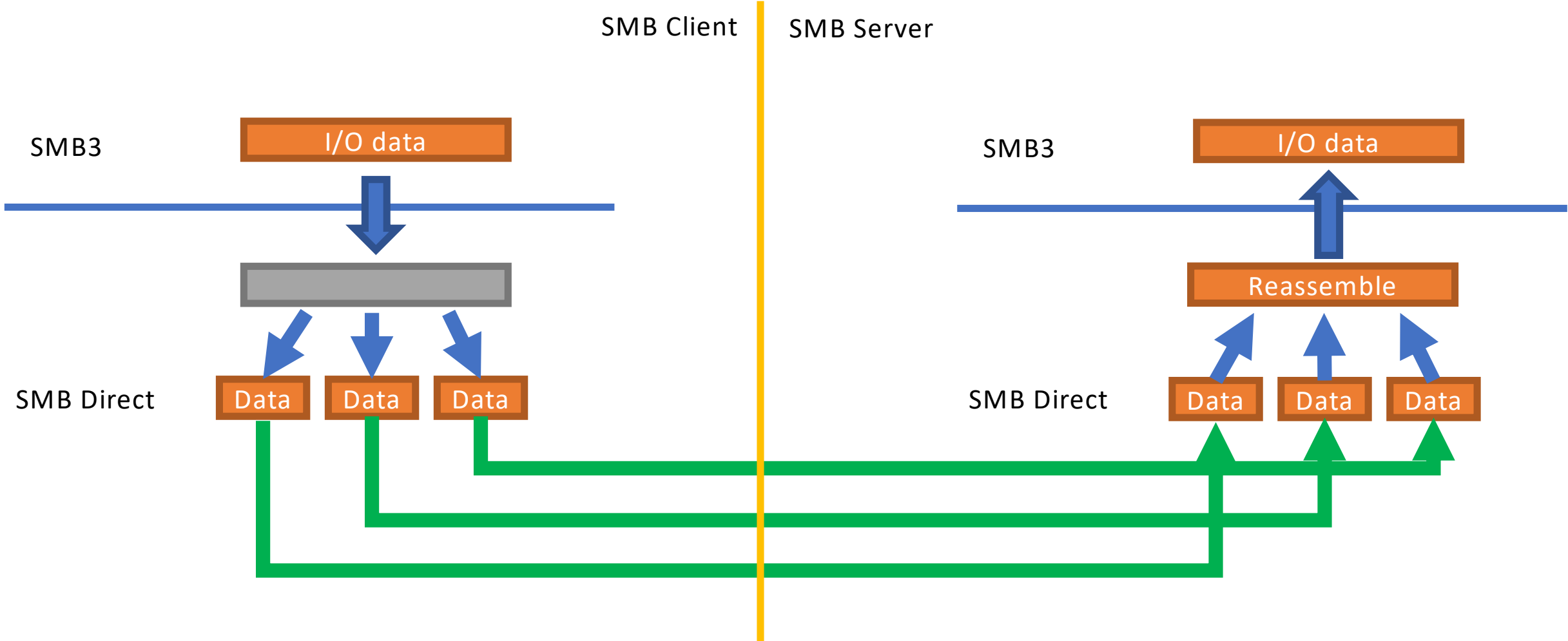
- Remote Direct Memory Access
- RDMA send/receive
  - Similar to socket interface, with no data copy in software stack
- RDMA read/write
  - Overlap local CPU and communication
  - Reduce CPU overhead on send sider
- Talking to RDMA hardware
  - RC (Reliable Connection) Queue Pair For SMB Direct
    - RDMA also supports UD (Unreliable Datagram) and UC (Unreliable Connection)
    - RC guarantees packet in order delivery and without corruption
  - Completion Queue is used to signaling I/O complete

# Data buffers in RDMA

- Nobody in the software stack will buffer the data
- RDMA
  - There is only one copy of the data buffer
    - Send -> no receive?
  - Application needs to do flow control
    - SMB Direct uses a credit system
    - No send-credits? Can't send data.



# RDMA Send/Receive



# SMB Direct credit system

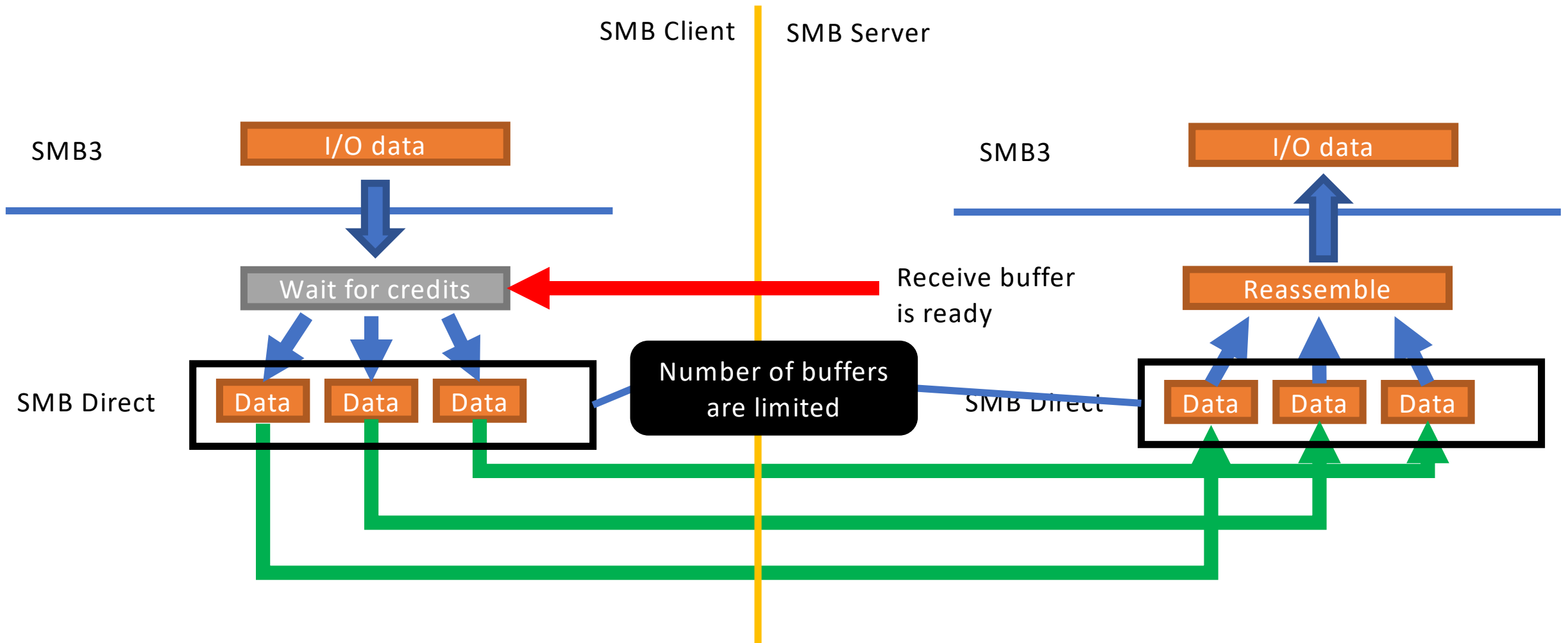
- Send credits
  - Decreased on each RDMA send
  - Receiving peer guarantees a RDMA recv buffer is posted for this send
- Credits are requested and granted in SMB Direct packet

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
CreditsRequested																CreditsGranted															
Flags																Reserved															
RemainingDataLength																															
DataOffset																															
DataLength																															

# SMB Direct credit system

- Running out of credits?
  - Some SMB commands send or receive lots of packet
  - One side keeps sending to the other side, no response is needed
  - Eventually the send runs out of send credits
- SMB Direct packet without payload
  - Extend credits to peer
  - Keep transport flowing
  - Should send as soon as new buffers are make available to post receive

# SMB Direct credit system

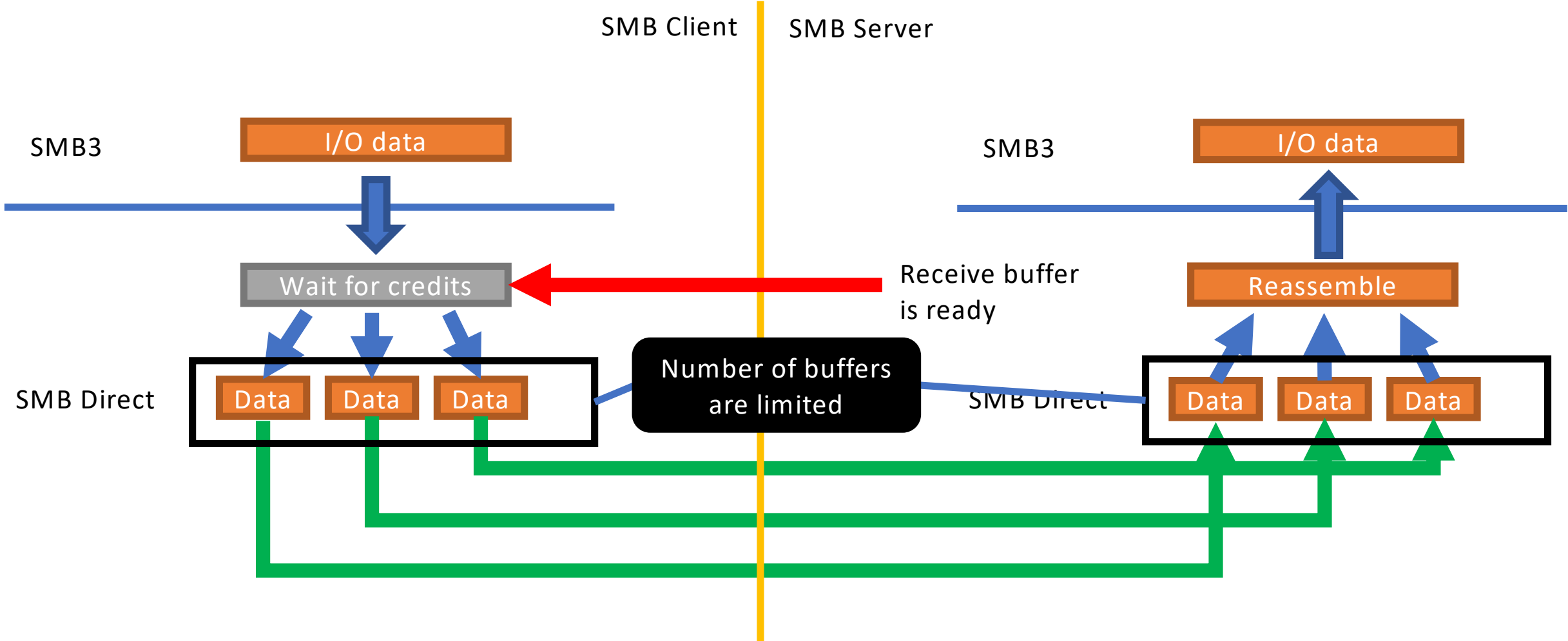


# RDMA Send/Receive

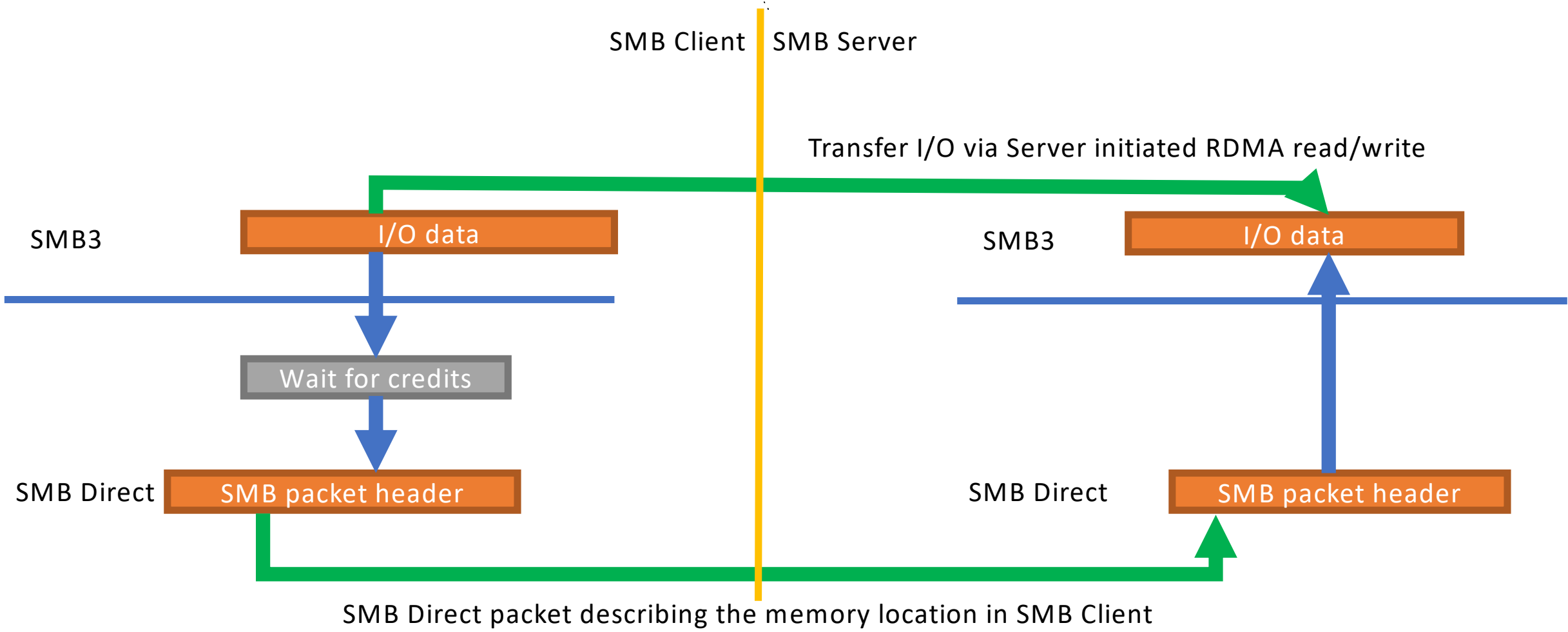
- CPU is doing all the hard work of packet segmentation and reassembly
- Not the best way to send or receive a large packet
  - Slower than most TCP hardware
  - Today most of TCP based NIC support hardware offloading
- SMB Direct uses RDMA send/receive for smaller packets
  - Default for packet size less than 4k bytes

# RDMA Send/Receive

How about large packets for file I/O?



# RDMA Read/Write



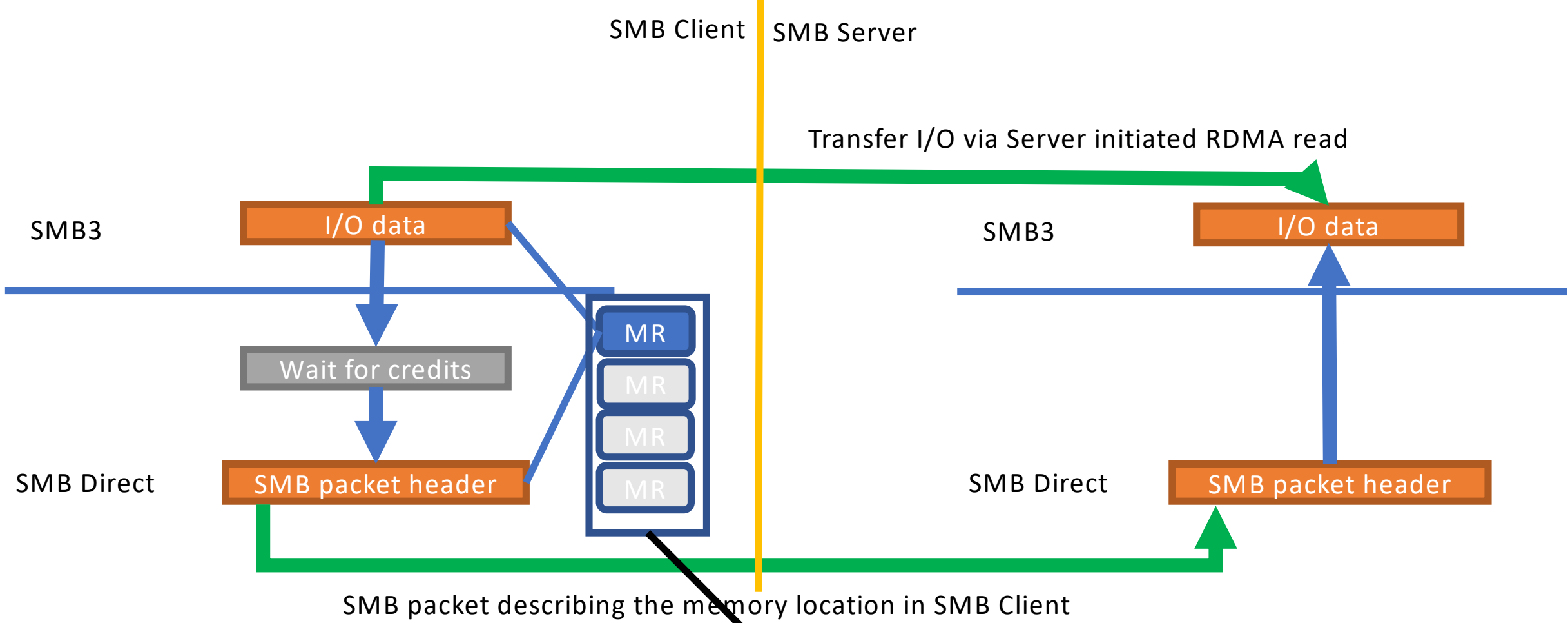
# Memory registration

- Client needs to tell Server where to write or read the data from its memory
- Memory is registered for RDMA
  - May not always be mapped to virtual address
  - I/O data are described as pages
- Correct permission is set on the memory registration
- SMB Client asks the SMB Server to do a RDMA I/O on this memory registration

# Memory registration order enforcement

- Need to make sure memory is registered before posting the request for SMB server to initiate RDMA I/O
  - Need to wait for completion for this request
  - If not, SMB server can't find where to look for data
  - A potential CPU context switch
- FRWR (Fast Registration Work Requests)
  - Send `IB_WR_REG_MR` through `ib_post_send`
  - No need to wait for completion if I/O is issued on the same CPU
  - Acts like a barrier in QP, guarantees it finishes before the following WR
  - Supported by almost all the modern RDMA hardware

# Memory registration



Limited number of memory registration pending I/O available per QP  
– determined by responder resources in CM.

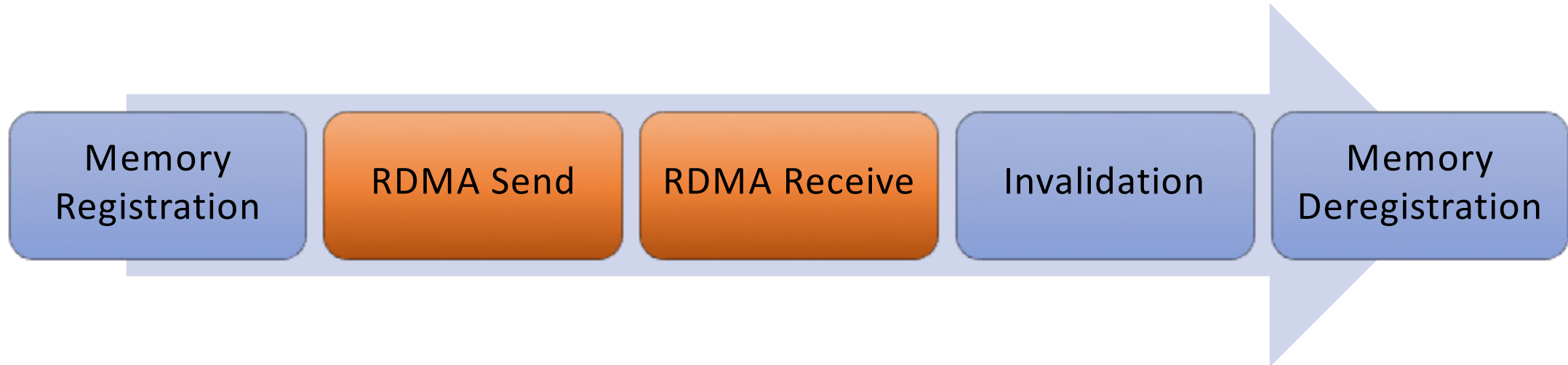
# Memory registration invalidation

- What to do when I/O is finished
  - Make sure SMB server no long has access to the memory region
  - Otherwise it can be messy since this is a hardware address and can be potentially changed by the server without client knowing it
- Client invalidates memory registration after I/O is done
  - IB\_WR\_LOCAL\_INV
  - After it completes, server no longer has access to this memory
  - Client has to wait for completion before buffer is consumed by upper layer
- Starting with SMB 3.02, SMB server supports remote invalidation
  - SMB2\_CHANNEL\_RDMA\_V1\_INVALIDATE

# Memory Deregistration

- Need to deregister memory after it's used for RDMA
- It's a time consuming process
  - In practice, it's even slower than memory registration and local invalidation combined
- Defer to a background kernel thread to do memory deregistration
  - It doesn't block the I/O returning path
  - Locking?

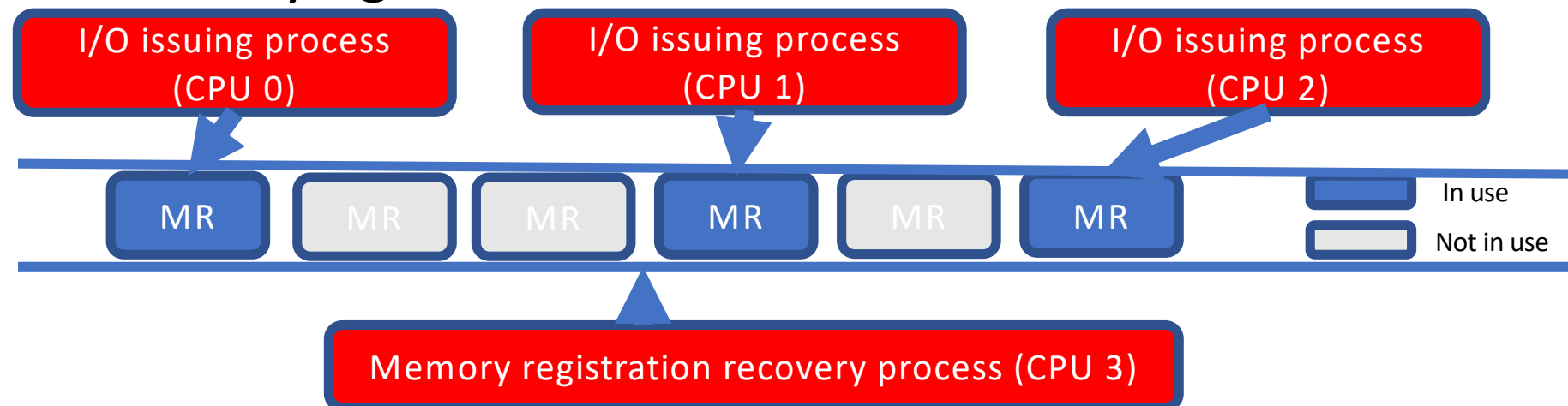
# RDMA Read/Write



- There are **three extra steps** compared to **RDMA Send/Receive**
- The last thing we want is locking for those 3 steps

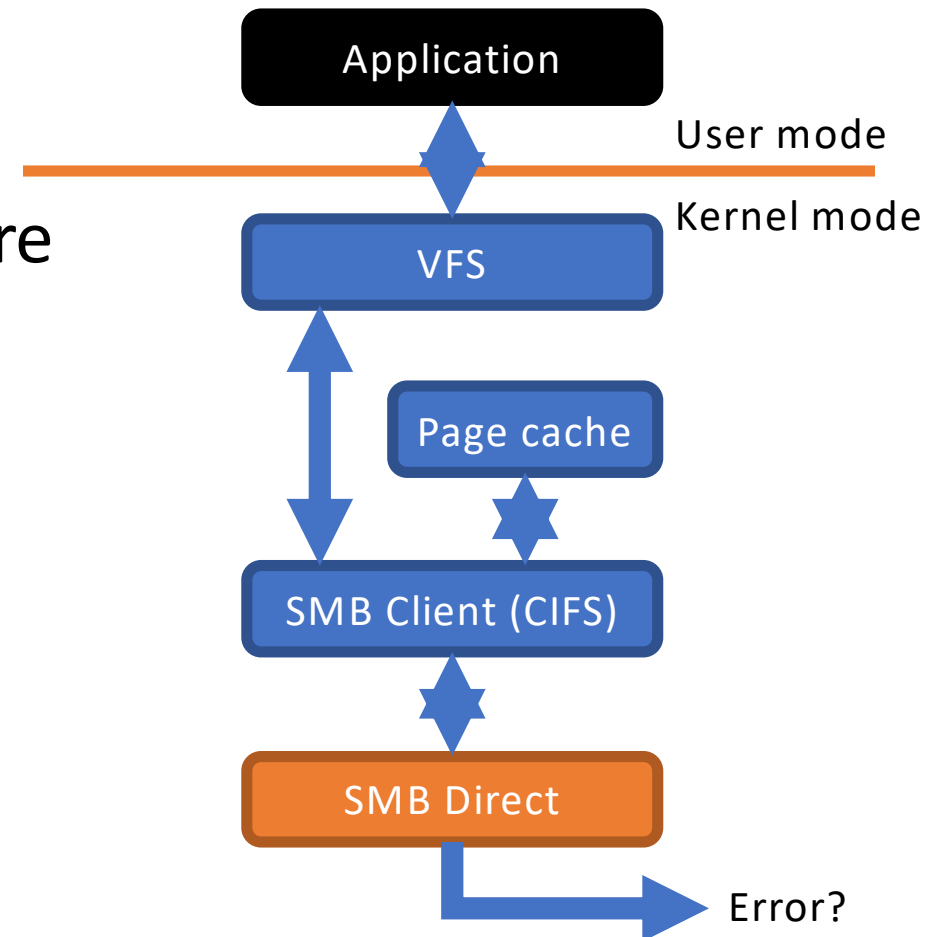
# Memory registration/deregistration

- Maintain a list of pre-allocated memory registration slots
- Defer to a background thread to recover MR while other I/Os are in progress
  - Return I/O as soon as the MR is invalidated
  - How about recovery process being blocked?
  - No lock needed since there is one only recovery process modifying the list



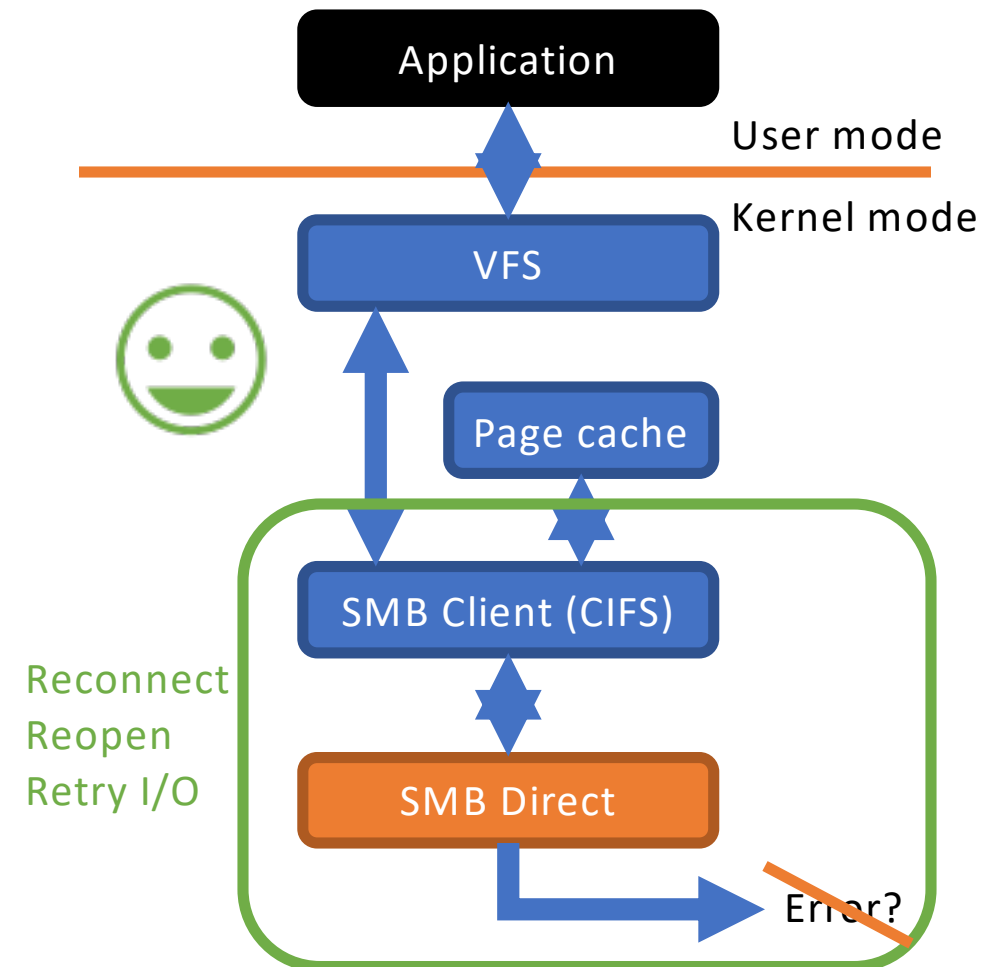
# RDMA failure

- It's possible hardware can sometimes return error
  - Even on a RC QP
  - In most cases can be reset and recovered
- SMB Direct will disconnect on any RDMA failure
- Return failure to upper layer?
  - Application may give up
  - Even worse for page cache write back

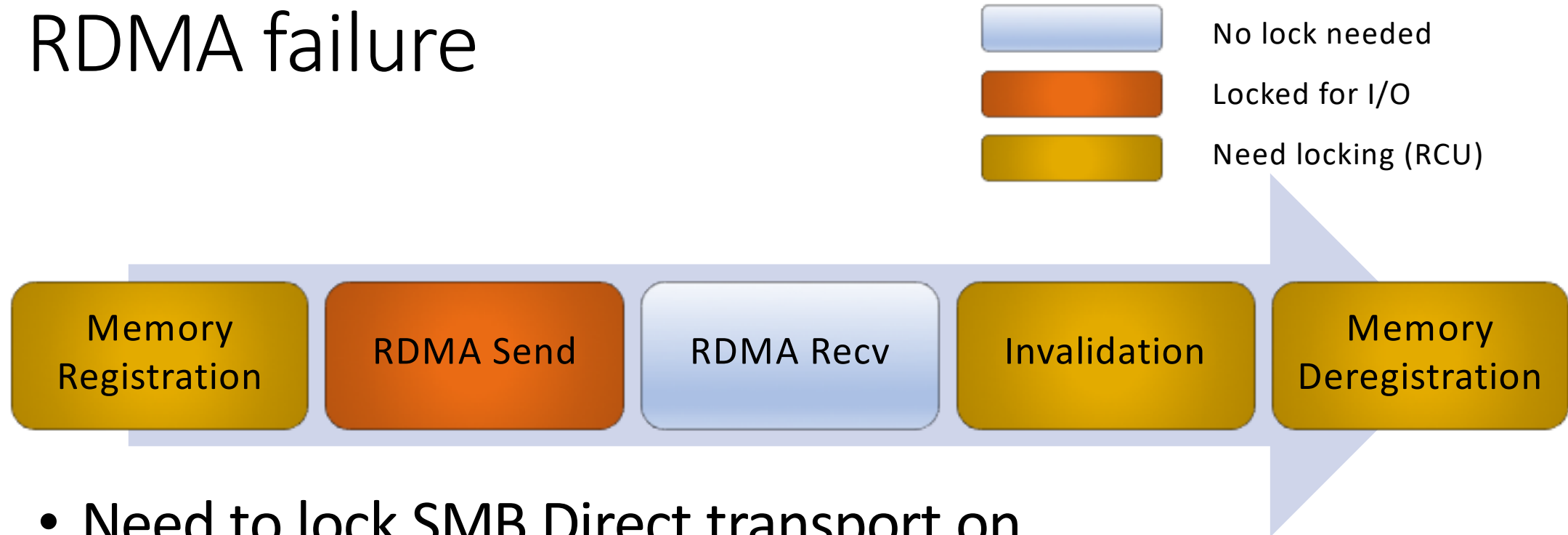


# RDMA failure

- SMB Direct recovery
  - Reestablish RDMA connection
  - Reinitialize resources and data buffers
- SMB layer recovery
  - Reopen session
  - Reopen file
- I/O recovery
  - Rebuild SMB I/O request
  - Requeue to RDMA transport
- Upper layer proceeds as if nothing happens
  - Application is happy
  - Kernel page cache is happy



# RDMA failure

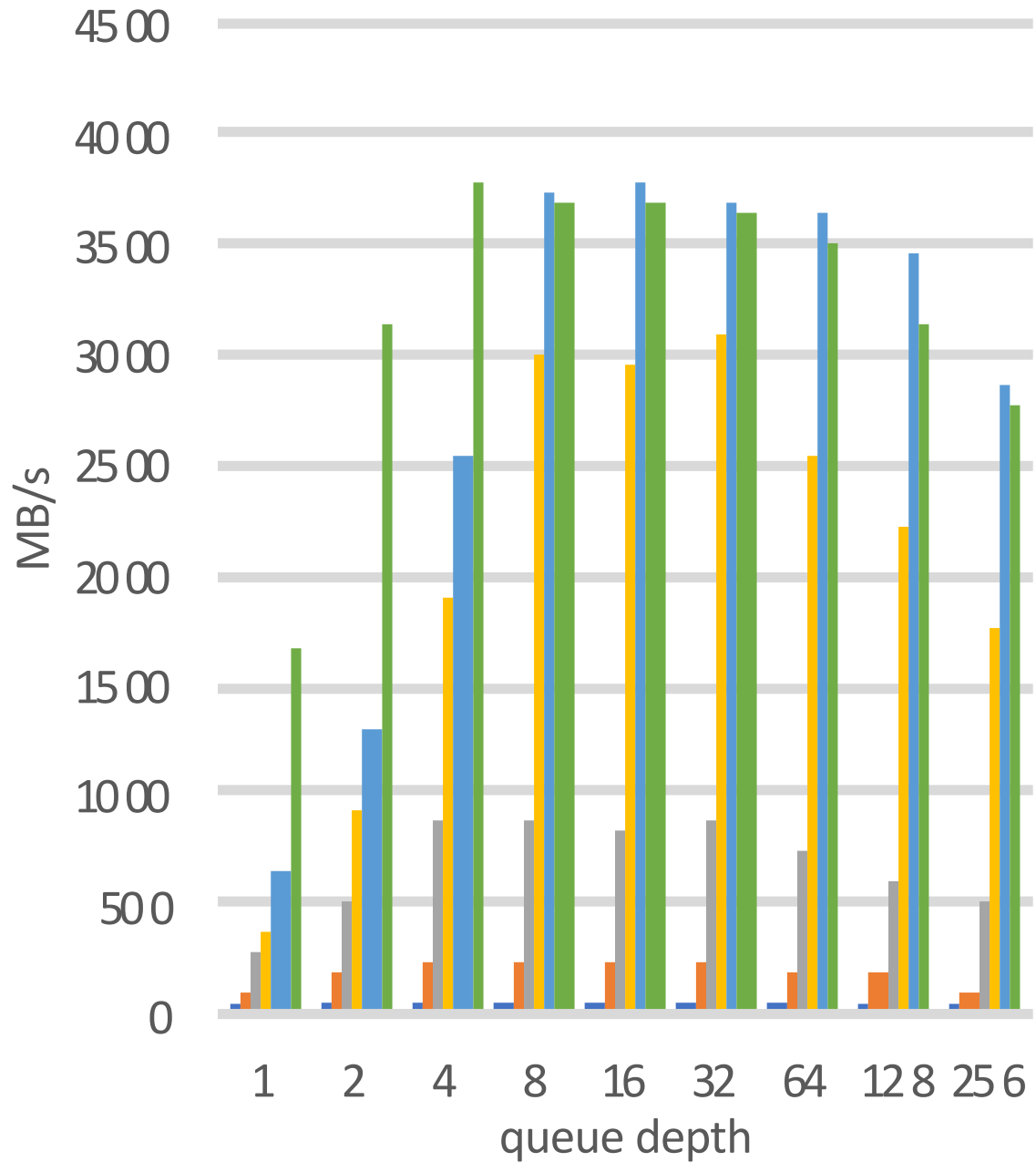


- Need to lock SMB Direct transport on disconnect/connect
- Use separate RCU to protect registrations
  - Rely on CPU context switch
  - Extreme lightweight on R side
  - CU takes all the locking overhead

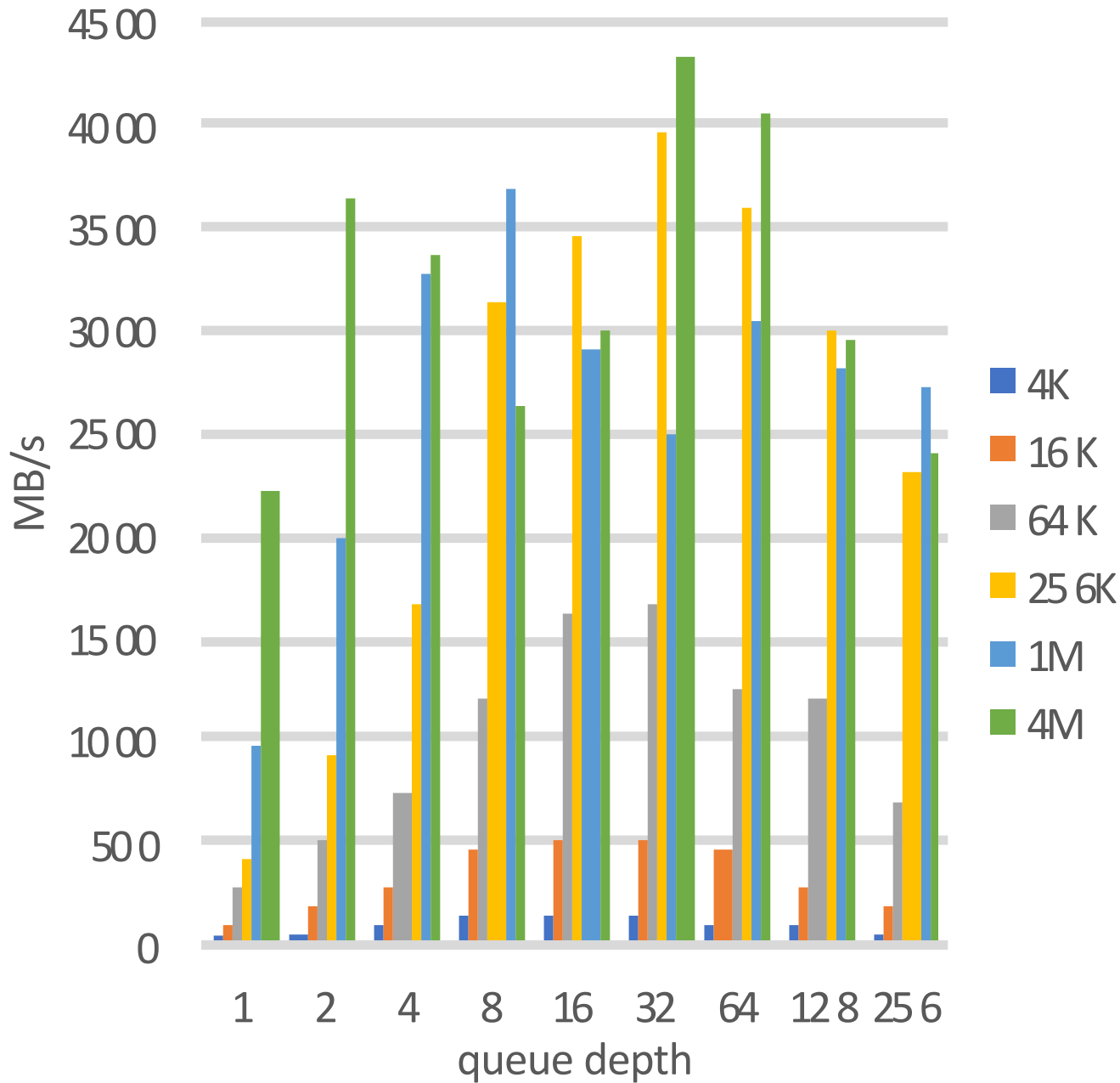
# Benchmark – test setup

- Linux SMB Client kernel 4.17-rc6
  - 2 x Intel E5-2650 v3 @ 2.30GHz
  - 128 GB RAM
- Windows SMB Server 2016
  - 2 x Intel E5-2695 v2 @ 2.40GHz
  - 128 GB RAM
  - SMB share on RAM disk
- Switch
  - Mellanox SX6036 40G VPI switch
- NIC
  - Mellanox ConnectX-3 Pro 40G Infiniband (32G effective data rate)
  - Chelsio T580-LP-CR 40G iWARP
- `mount.cifs -o rdma,vers=3.02`
- `FIO direct=1`

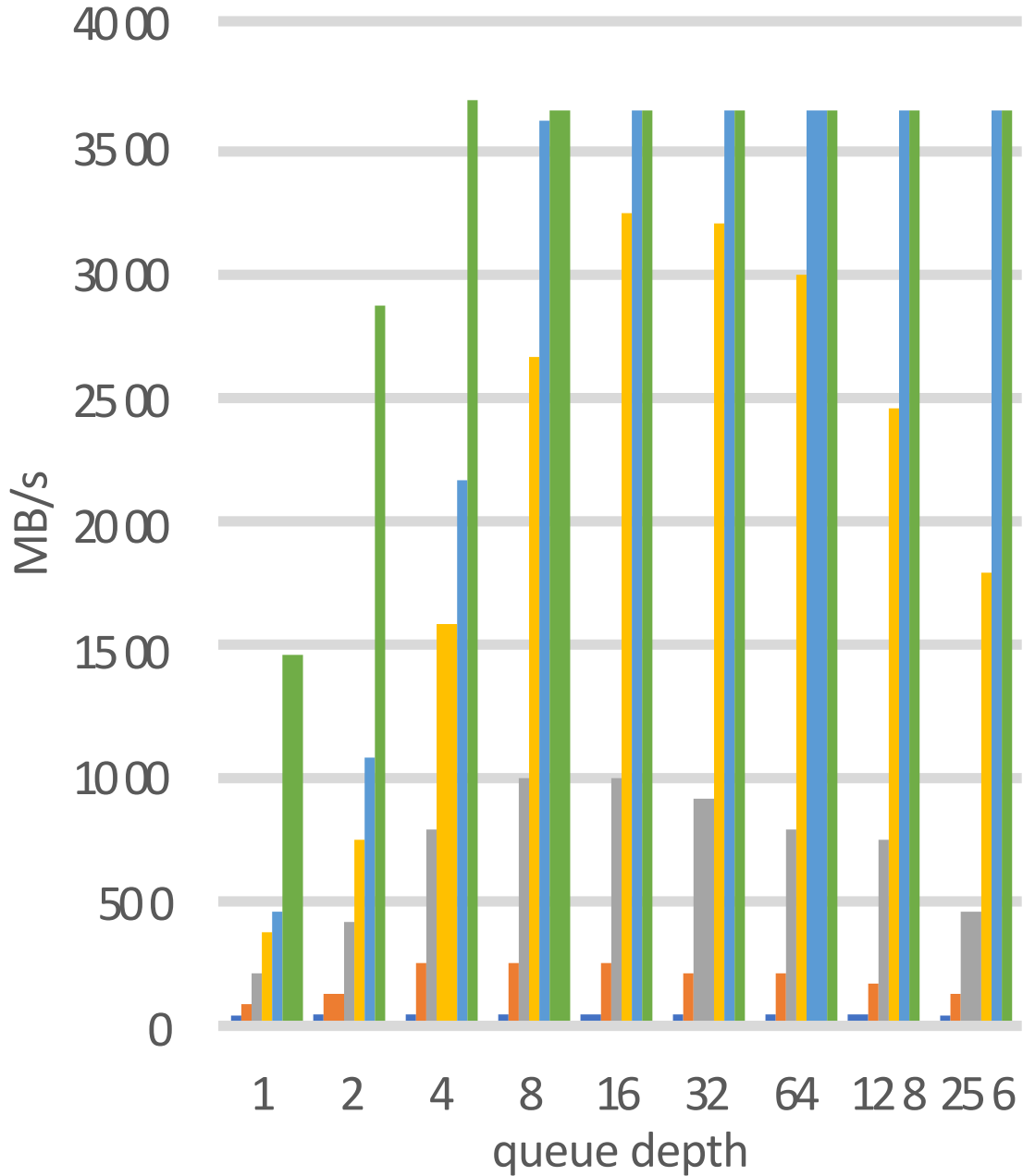
### SMB Read - Mellanox



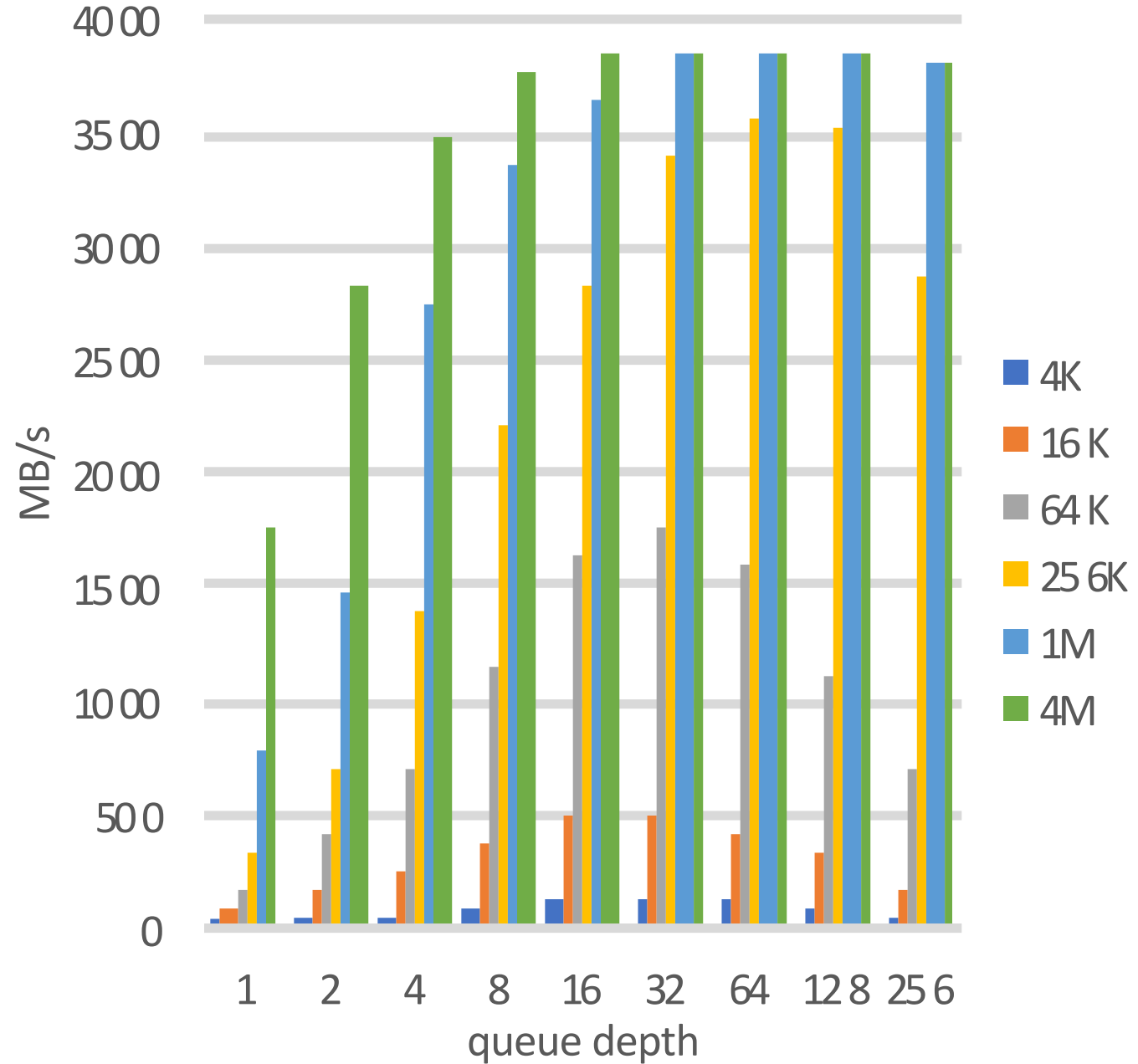
### SMB Read - Chelsio



### SMB Write - Mellanox

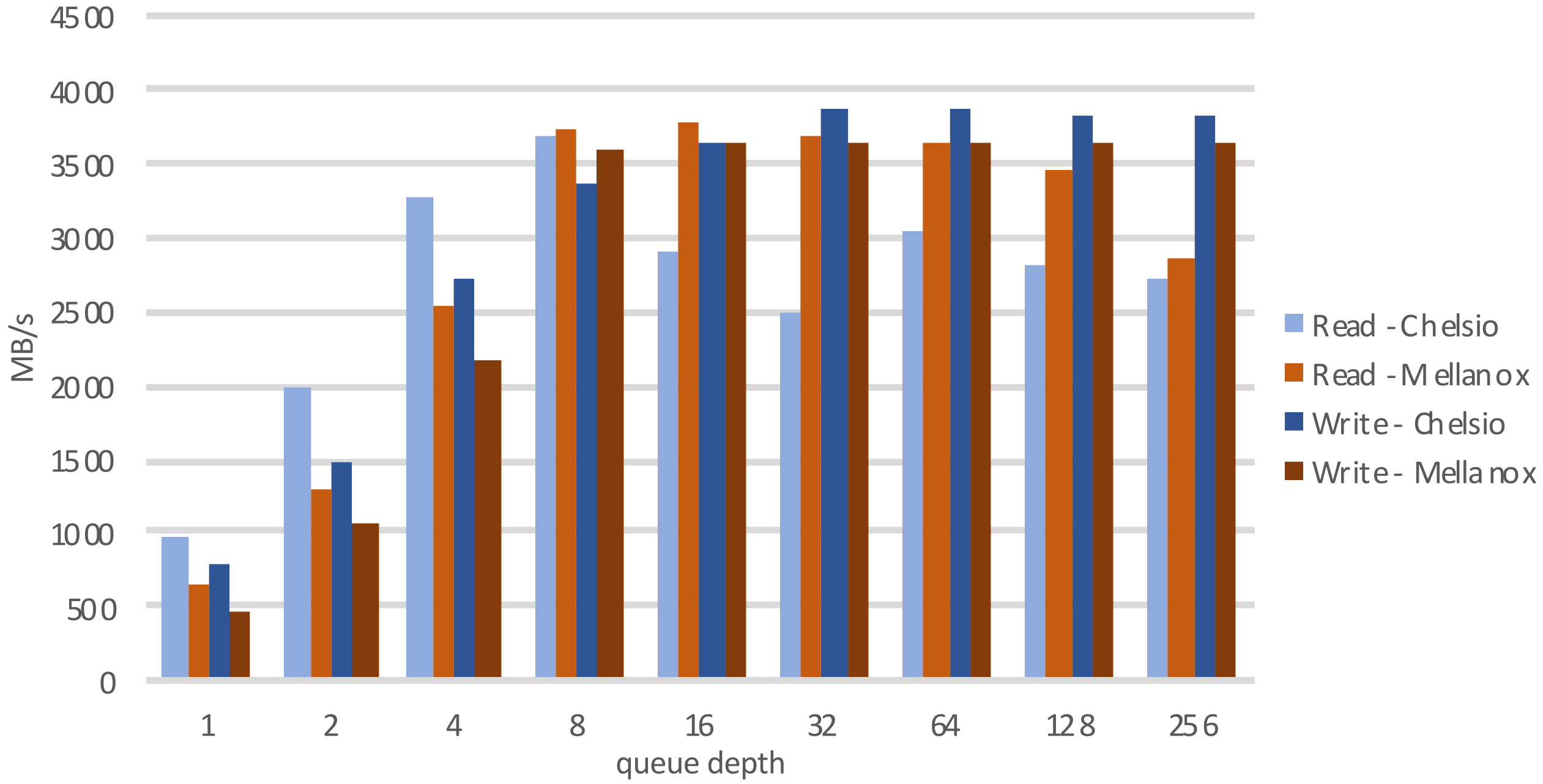


### SMB Write - Chelsio

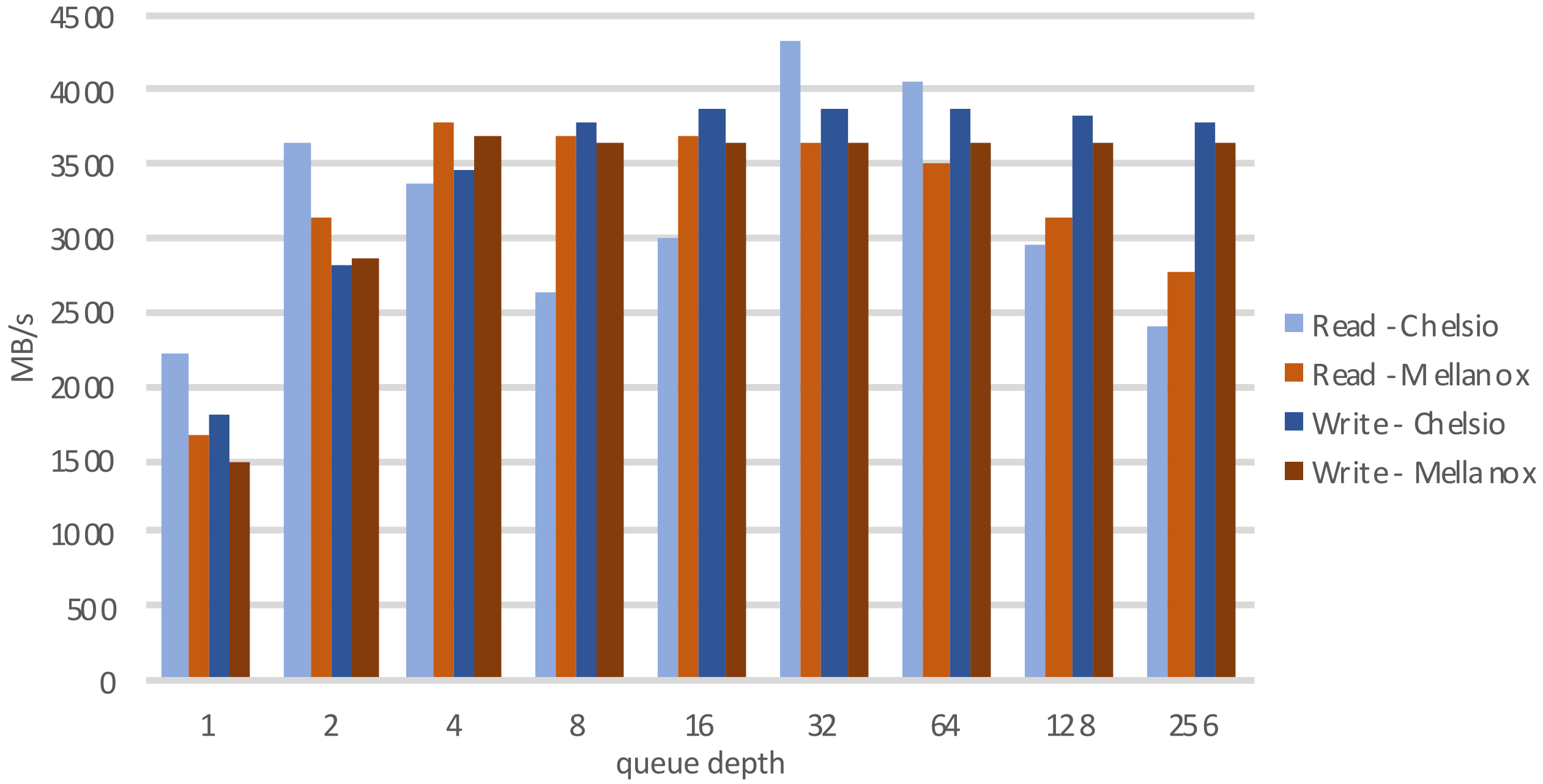


- 4K
- 16K
- 64K
- 256K
- 1M
- 4M

# Infiniband vs iWARP - 1M I/O

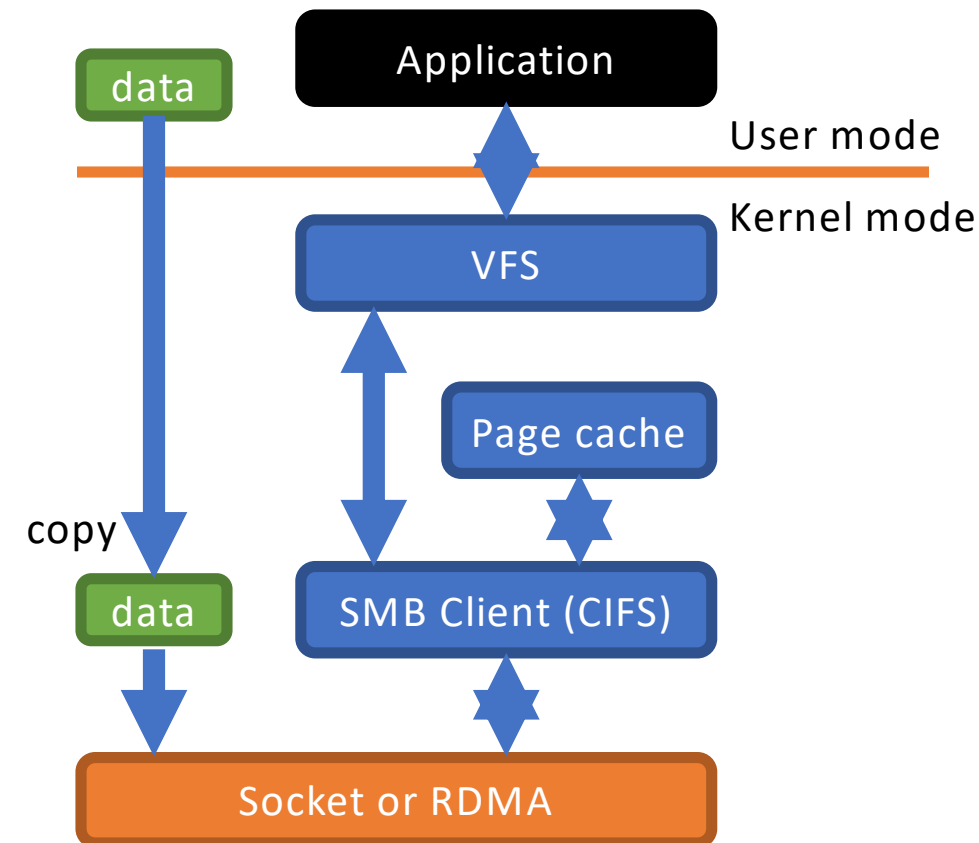


# Infiniband vs iWARP - 4M I/O



# Buffered I/O

- Copy the data from user space to kernel space
  - CIFS always doing this
  - User data can't be trusted
  - May use data for signing and encryption
    - User application modifies data?
- It's good for caching
  - Page cache speeds up I/O
- There is a cost
  - CIFS needs to allocate buffers for I/O
  - Memory copy uses CPU and takes time

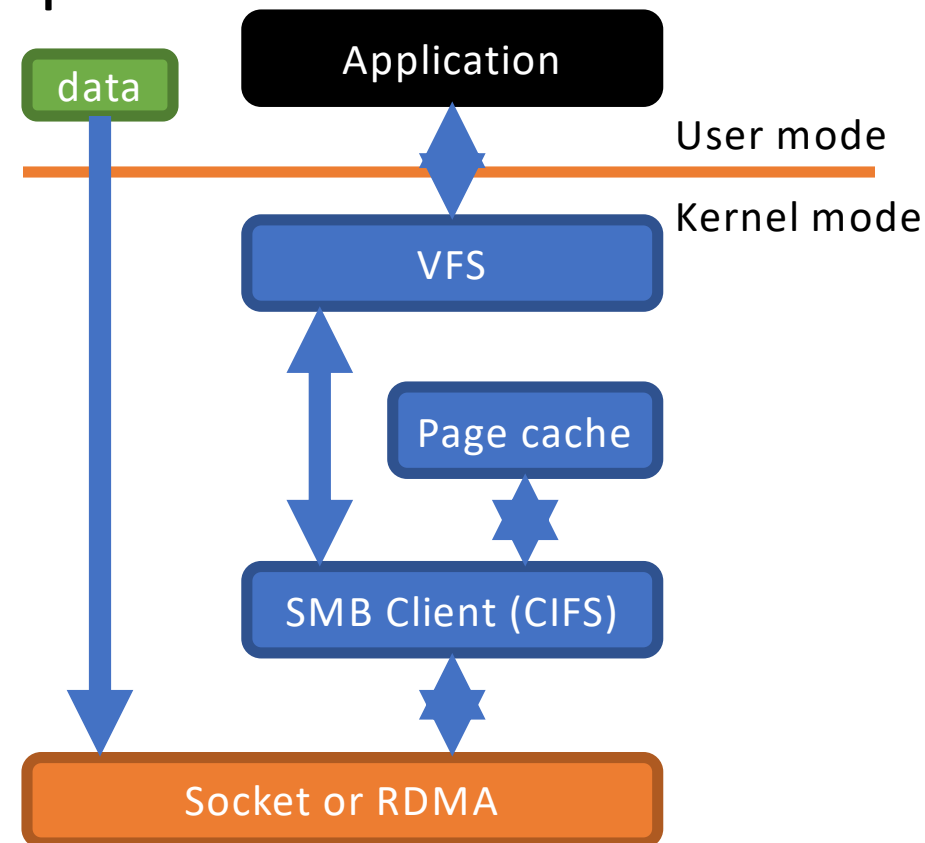


# SMB Read 1M

```
root@43f13-31: /md1/cifs-2.6/tools/perf
PerfTop: 3786 irqs/sec kernel:99.1% exact: 100.0% [2750Hz cycles:ppp], (all, 40 CPUs)
-----
17.72% [kernel] [k] memcpy_erms ←
 8.65% [kernel] [k] acpi_os_write_port
 8.20% [kernel] [k] __lock_acquire.isra.32
 5.31% [kernel] [k] get_page_from_freelist ←
 3.43% [kernel] [k] gup_pgd_range
 3.17% [kernel] [k] lock_release
 2.51% [kernel] [k] free_pcpages_bulk
 2.38% [kernel] [k] lock_acquire
 1.57% [kernel] [k] _copy_to_iter
 1.40% [kernel] [k] copy_page_to_iter
 1.32% [kernel] [k] __alloc_pages_nodemask
 0.93% [kernel] [k] page_mapping
 0.82% [kernel] [k] find_held_lock
 0.80% [ib_core] [k] ib_sg_to_pages
 0.73% [kernel] [k] debug_check_no_locks_freed
 0.71% [kernel] [k] free_unref_page
 0.69% [kernel] [k] __lock_is_held
 0.68% [kernel] [k] free_unref_page_commit
 0.68% [kernel] [k] update_load_avg
 0.67% [kernel] [k] lock_is_held_type
 0.62% [cifs] [k] 0x0000000000000283f5
 0.62% [kernel] [k] __sched_text_start
 0.62% [kernel] [k] menu_select
 0.58% [kernel] [k] __page_cache_release
 0.58% [cifs] [k] 0x000000000000028403
 0.55% [kernel] [k] free_unref_page_prepare.part.58
 0.51% [kernel] [k] free_pcp_prepare
```

# Direct I/O

- Data is passed directly from user-space to transport
- I/O data are not cached
  - Useful in situations don't need page cache
- No page allocation, deallocation, memcpy
- Security concern?
  - More surface for user space attack
- Patch set in review
- Use direct I/O
  - `mount.cifs -o cache=none`
  - open file with `O_DIRECT`

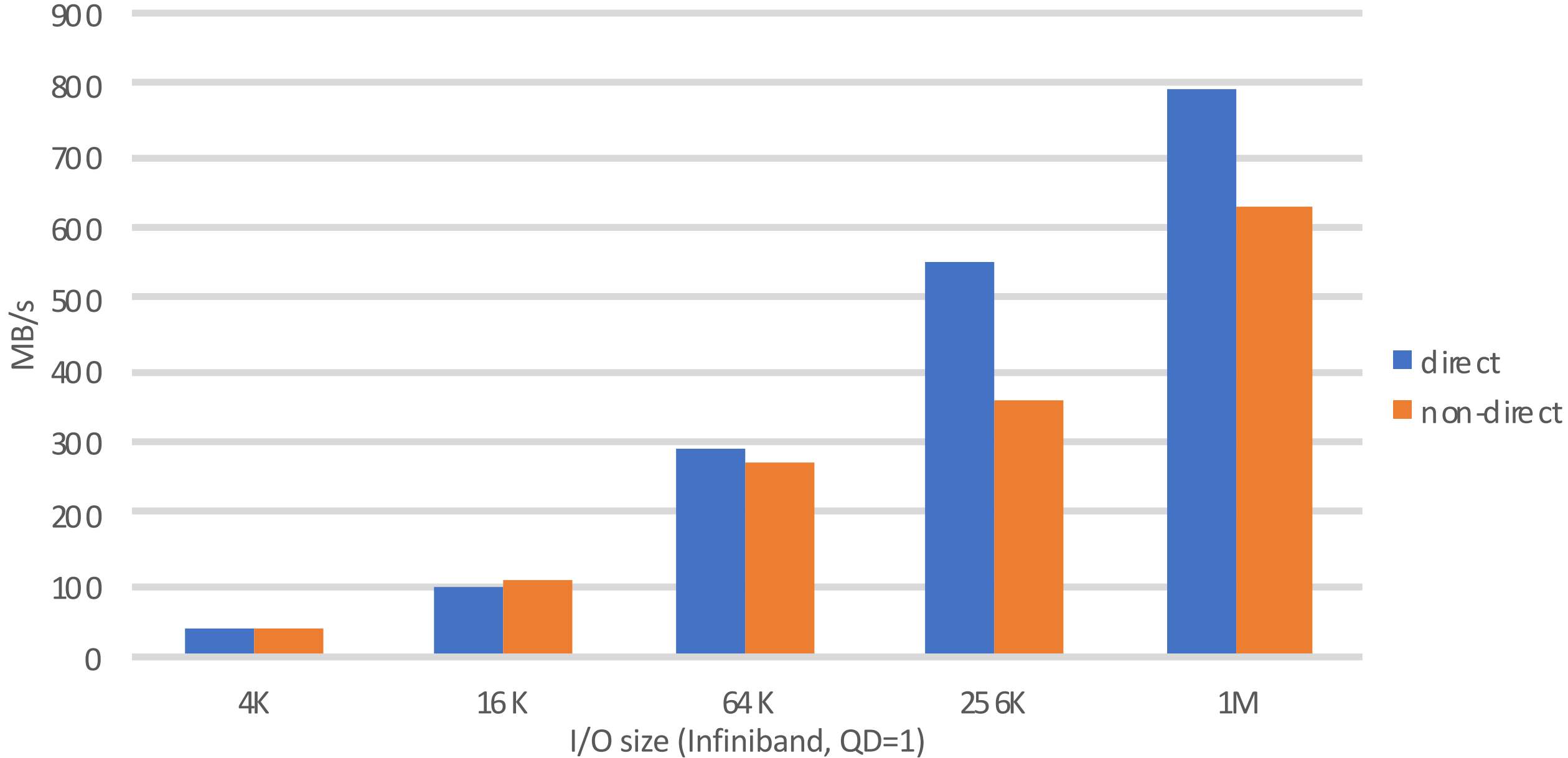


# SMB Read 1M

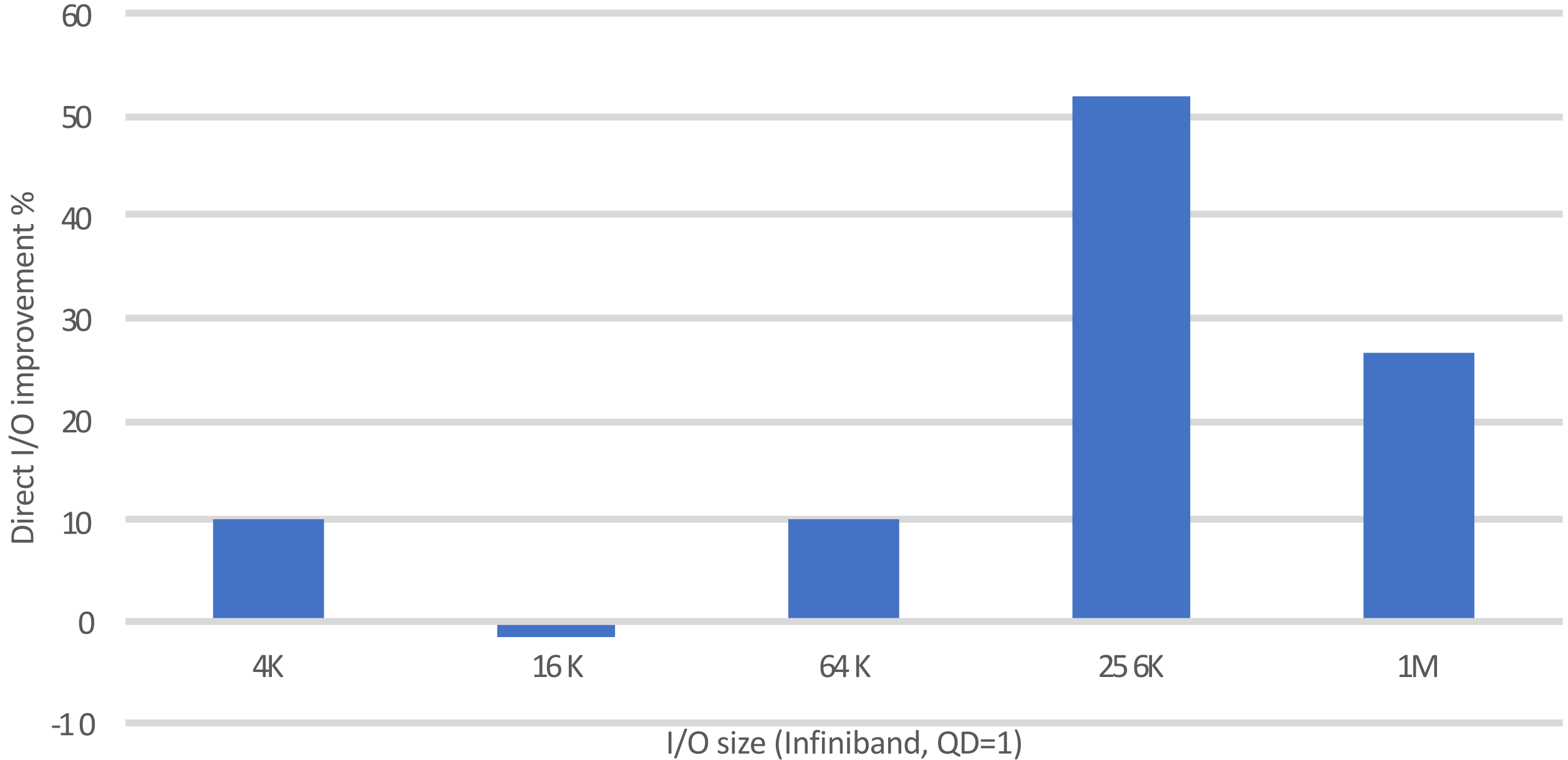
```
root@43f13-31: /md1/cifs-2.6/tools/perf
PerfTop: 3786 irqs/sec kernel:99.1% exact: 100.0% [27]
-----
17.72% [kernel] [k] memcpy_erms
8.65% [kernel] [k] acpi_os_write_port
8.20% [kernel] [k] __lock_acquire.isra.32
5.31% [kernel] [k] get_page_from_freelist
3.43% [kernel] [k] gup_pgd_range
3.17% [kernel] [k] lock_release
2.51% [kernel] [k] free_pcppages_bulk
2.38% [kernel] [k] lock_acquire
1.57% [kernel] [k] __copy_to_iter
1.40% [kernel] [k] copy_page_to_iter
1.32% [kernel] [k] __alloc_pages_nodemask
0.93% [kernel] [k] page_mapping
0.82% [kernel] [k] find_held_lock
0.80% [ib_core] [k] ib_sg_to_pages
0.73% [kernel] [k] debug_check_no_locks_freed
0.71% [kernel] [k] free_unref_page
0.69% [kernel] [k] __lock_is_held
0.68% [kernel] [k] free_unref_page_commit
0.68% [kernel] [k] update_load_avg
0.67% [kernel] [k] lock_is_held_type
0.62% [cifs] [k] 0x000000000000283f5
0.62% [kernel] [k] __sched_text_start
0.62% [kernel] [k] menu_select
0.58% [kernel] [k] __page_cache_release
0.58% [cifs] [k] 0x00000000000028403
0.55% [kernel] [k] free_unref_page_prepare.par
```

```
root@43f13-31: /md1/cifs-2.6/tools/perf
PerfTop: 3601 irqs/sec kernel:98.9% exact: 100.0%
-----
11.62% [kernel] [k] __lock_acquire.isra.32
8.72% [kernel] [k] acpi_os_write_port
3.92% [kernel] [k] lock_release
3.53% [kernel] [k] lock_acquire
2.61% [kernel] [k] gup_pgd_range
1.90% [kernel] [k] try_to_wake_up
1.59% [kernel] [k] lock_is_held_type
1.59% [cifs] [k] 0x00000000000028343
1.56% [ib_core] [k] ib_sg_to_pages
1.50% [kernel] [k] __sched_text_start
1.46% [kernel] [k] menu_select
1.42% [kernel] [k] update_load_avg
1.40% [kernel] [k] __lock_is_held
1.36% [kernel] [k] find_busiest_group
1.12% [kernel] [k] find_held_lock
1.08% [kernel] [k] do_raw_spin_lock
0.73% [kernel] [k] cpuidle_enter_state
0.72% [kernel] [k] __mutex_unlock_slowpath
0.70% [cifs] [k] 0x00000000000028351
0.69% [kernel] [k] update_blocked_averages
0.64% [kernel] [k] pick_next_task_fair
0.64% [kernel] [k] __update_load_avg_se.isra
0.63% [kernel] [k] __x86_indirect_thunk_rax
0.61% [kernel] [k] find_next_bit
```

# SMB Read with direct I/O



# SMB Read with direct I/O



# Future research

- Multiple channels
- CQ polling choices
  - Polling from SOFTIRQ
    - Polling from interrupts or softirq threads
  - Take some time if SMB server sends large amount of packets
  - Hung the CPU in interrupt mode
- NUMA aware
  - If we have to do memcpy, it's better on the same NUMA node

# About your speaker

- Enabling Linux VM for Azure at Microsoft
- Infiniband for Azure HPC
  - User-mode RDMA only, kernel by-pass
  - QP, CP, door bell directly to hardware, with little virtualization overhead
- Storage for Azure
  - Currently on SCSI
  - Support for block multiple queues
- GPU compute for Azure HPC
  - Nvidia Tesla
  - PCI express passthrough



Questions?

Thank you