WEKA.IO
World's Fastest File System

Gartner
Cool Vendor
2018
™

WekaIO: Extracting Performance from Modern HPC Hardware

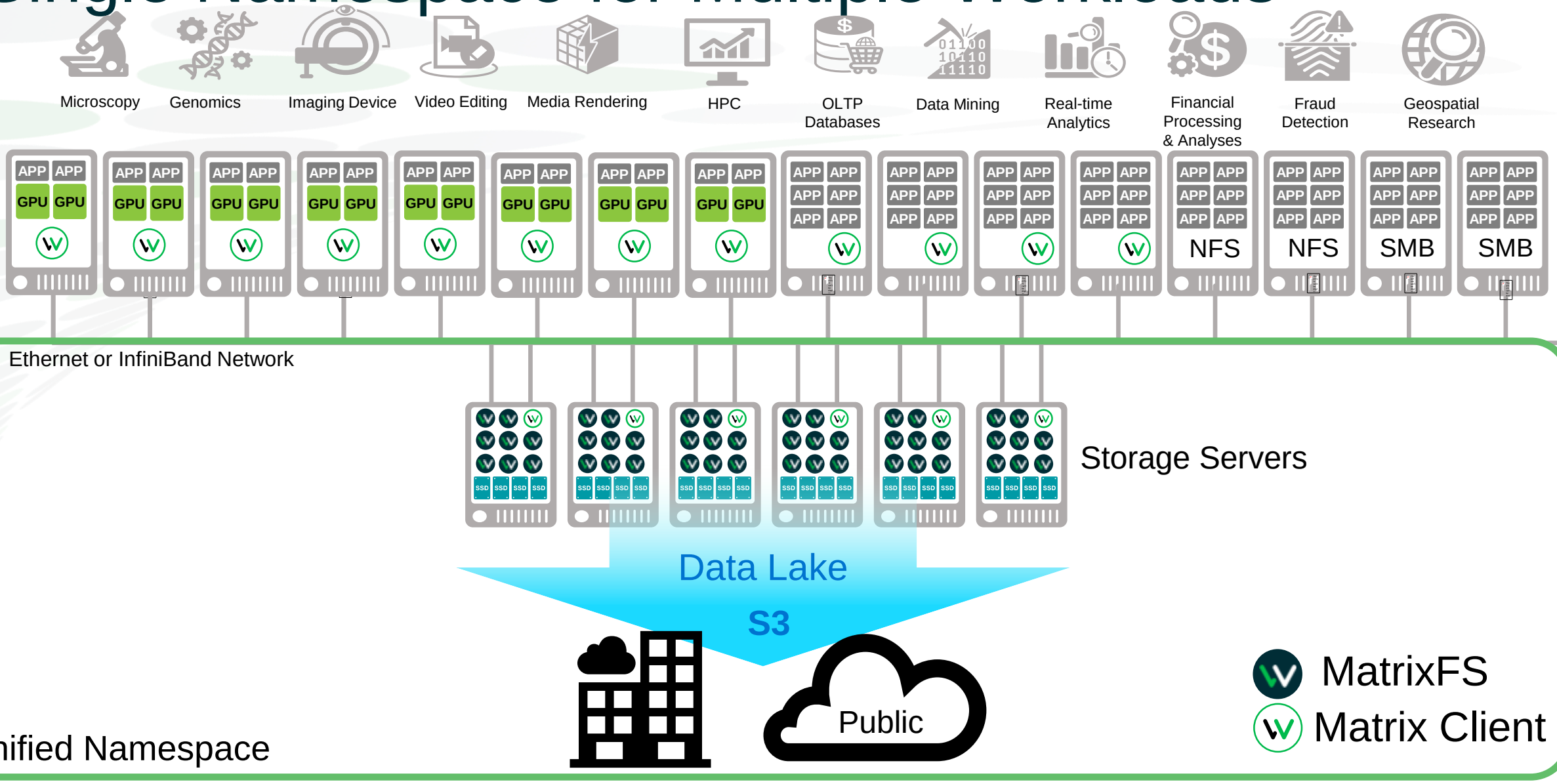
Derek Burke, Regional Account Manager
Chris Weeden, Senior Systems Engineer

Company Overview

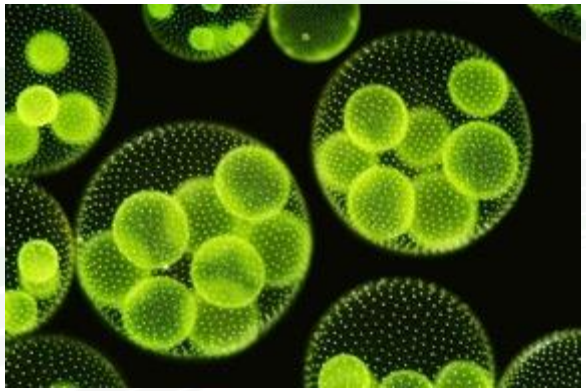
WHAT WE DO

WekaIO Matrix is the fastest, most scalable parallel file system for AI and technical compute workloads, on premises and in the public cloud

Single Namespace for Multiple Workloads



Focused On Performance Use Cases



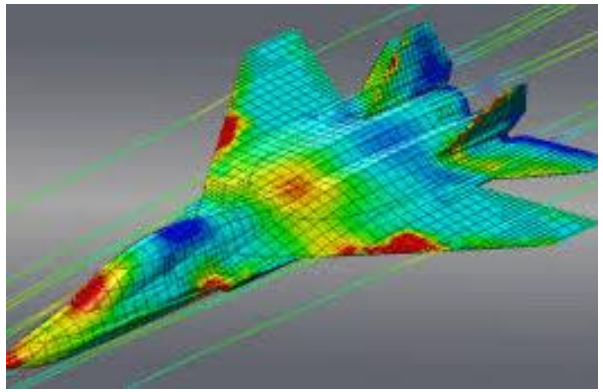
- Genomics sequencing and analytics
- Drug discovery
- Microscopy

- Machine Learning\ AI
- Real-time analytics
- IOT

- Algorithmic trading
- Business analytics (SAS Grid)
- Risk analysis (Monte Carlo simulation)

- Seismic
- Reservoir simulation
- Analytics

- Media rendering
- Transcoding
- Visual Effects (VFX)



- CFD, Simulations
- EDA

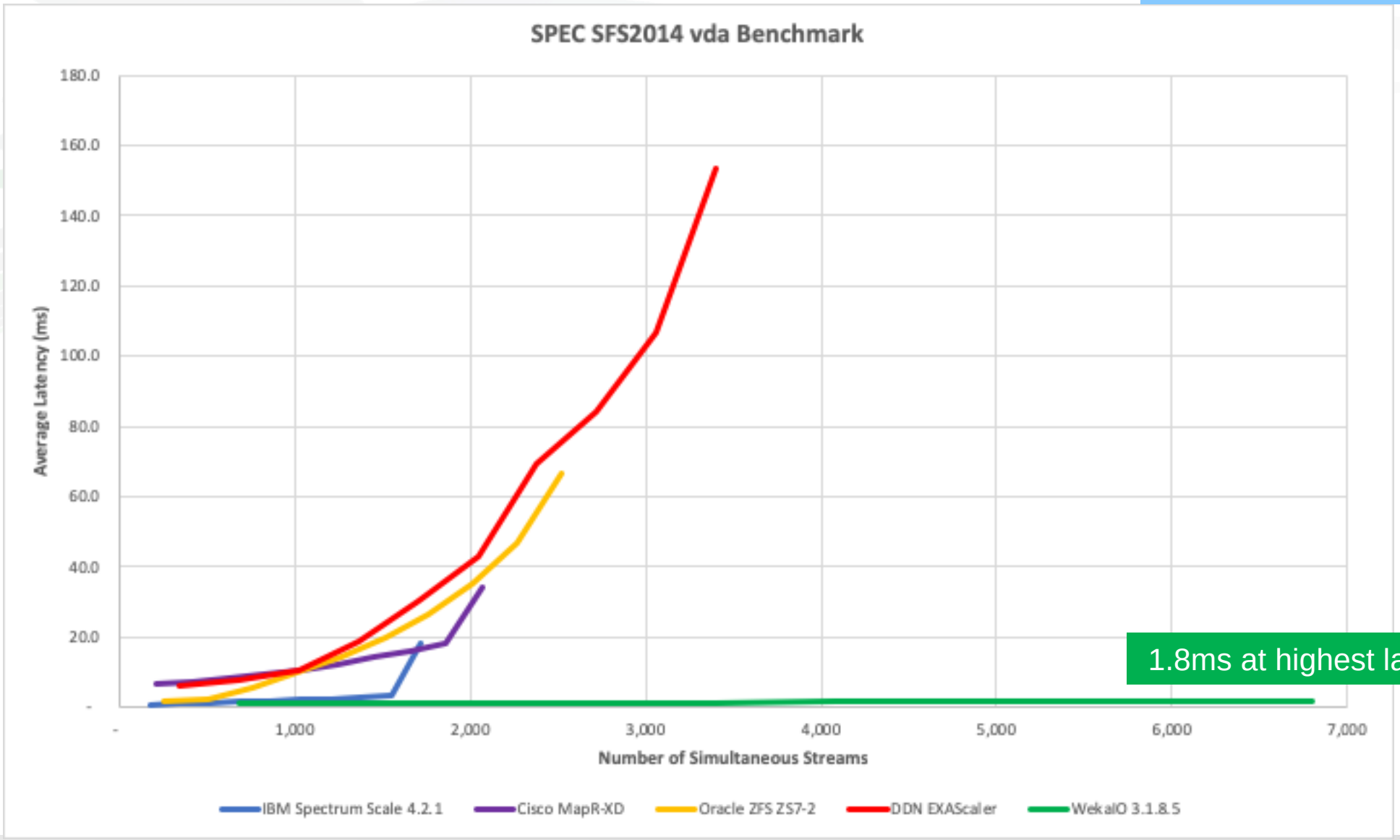
We Make a Bold Claim…

From one rack of commodity hardware with 100GbE Network Approximately:

- 400GB/s
- 30 Million IOPS

SPEC SFS2014 vda Results

Test is 90% write intensive

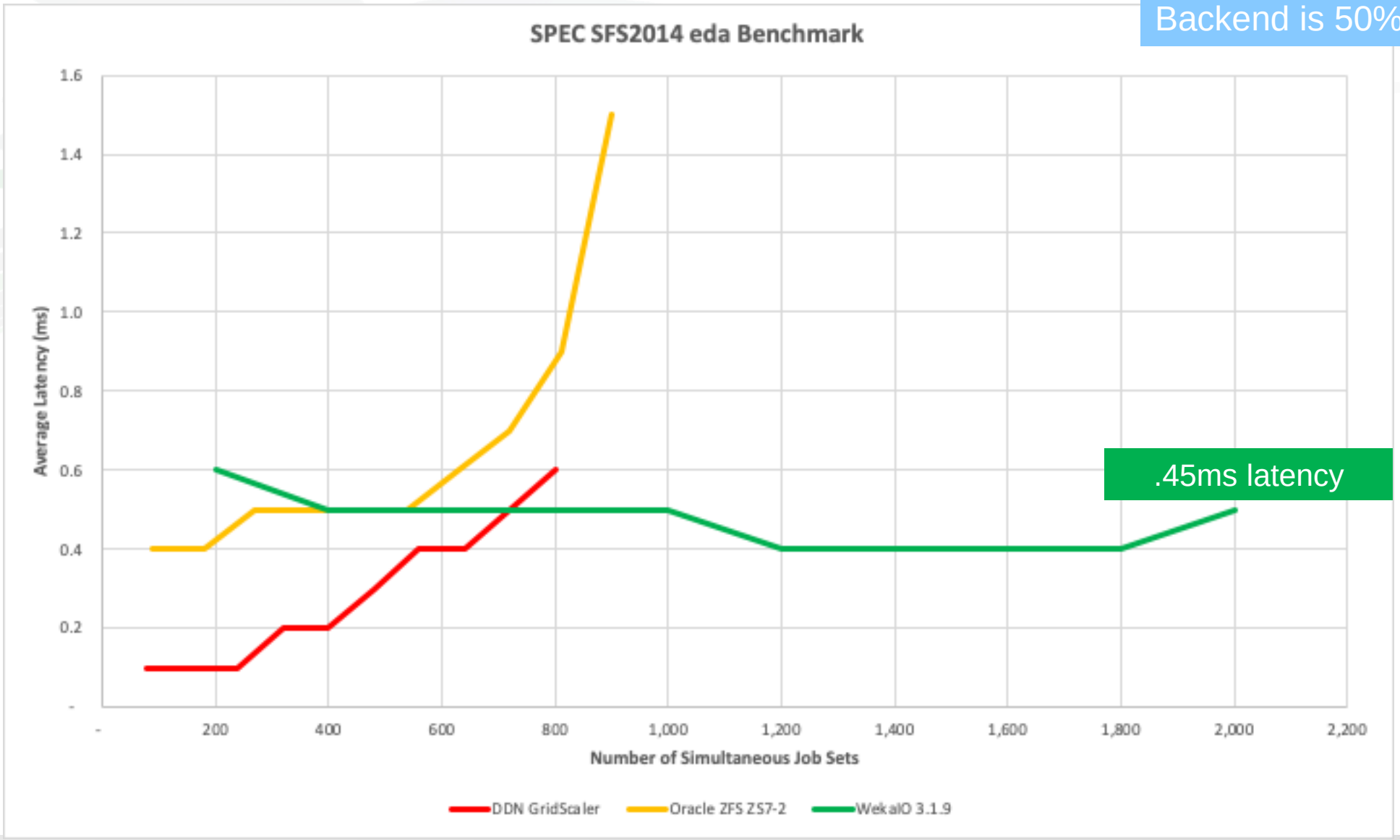


1.8ms at highest latency

Lower and Longer Are Better

SPEC SFS2014 eda Results

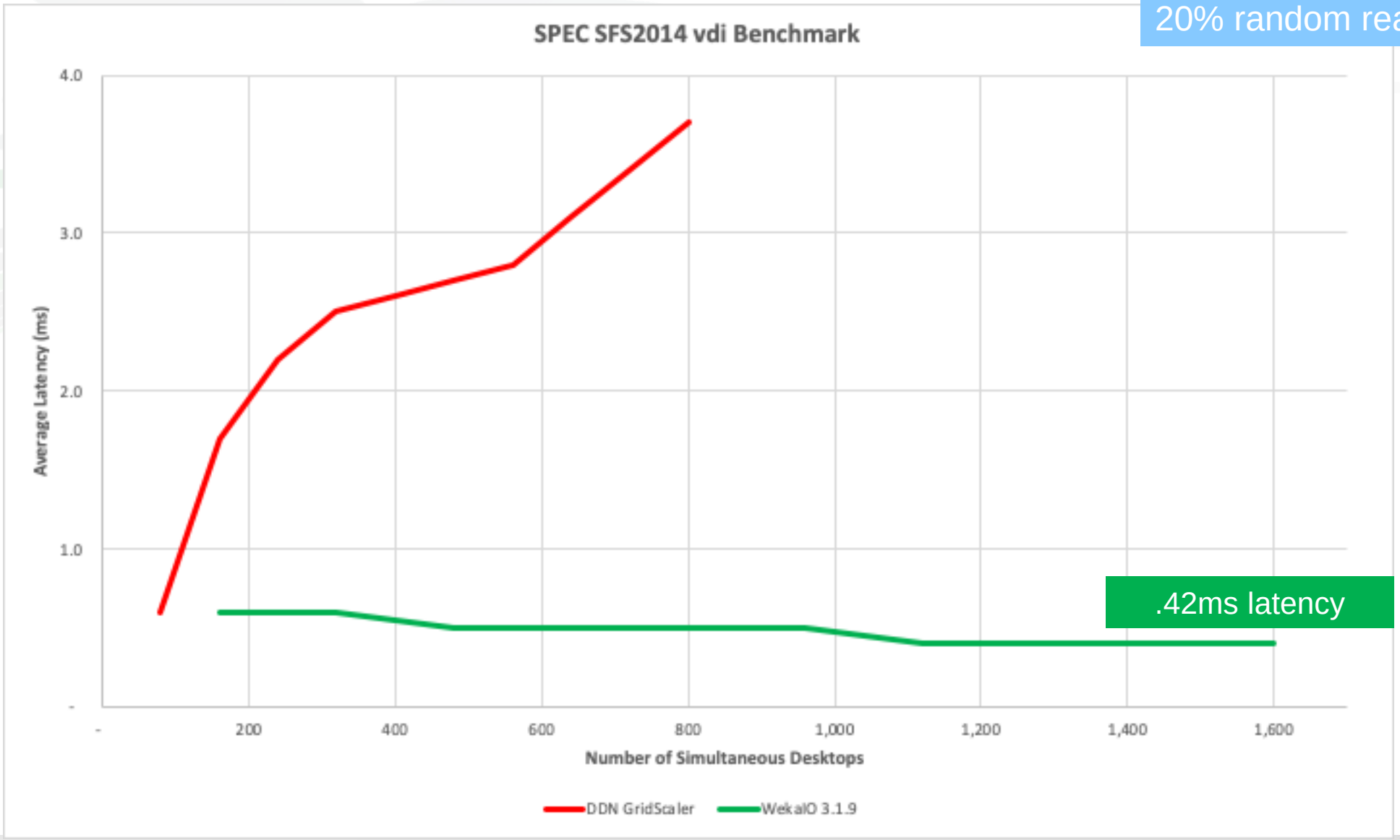
Frontend is 50% stats test
Backend is 50% R and W



.45ms latency

SPEC SFS2014 vdi Results

Test is 60% random write
20% random read

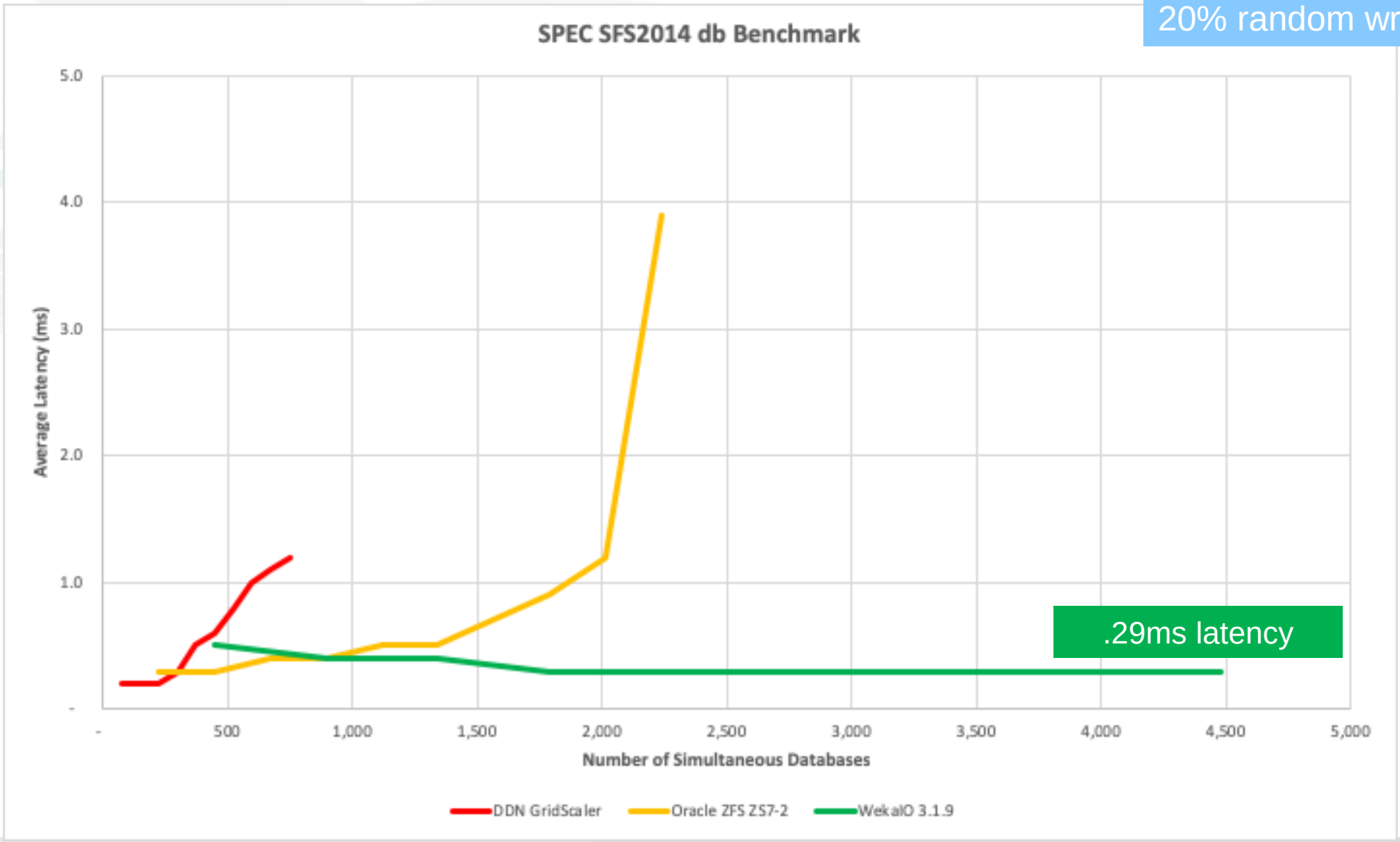


.42ms latency

Lower and Longer Are Better

SPEC SFS2014 db Results

Test is 80% random read
20% random write



Lower and Longer Are Better

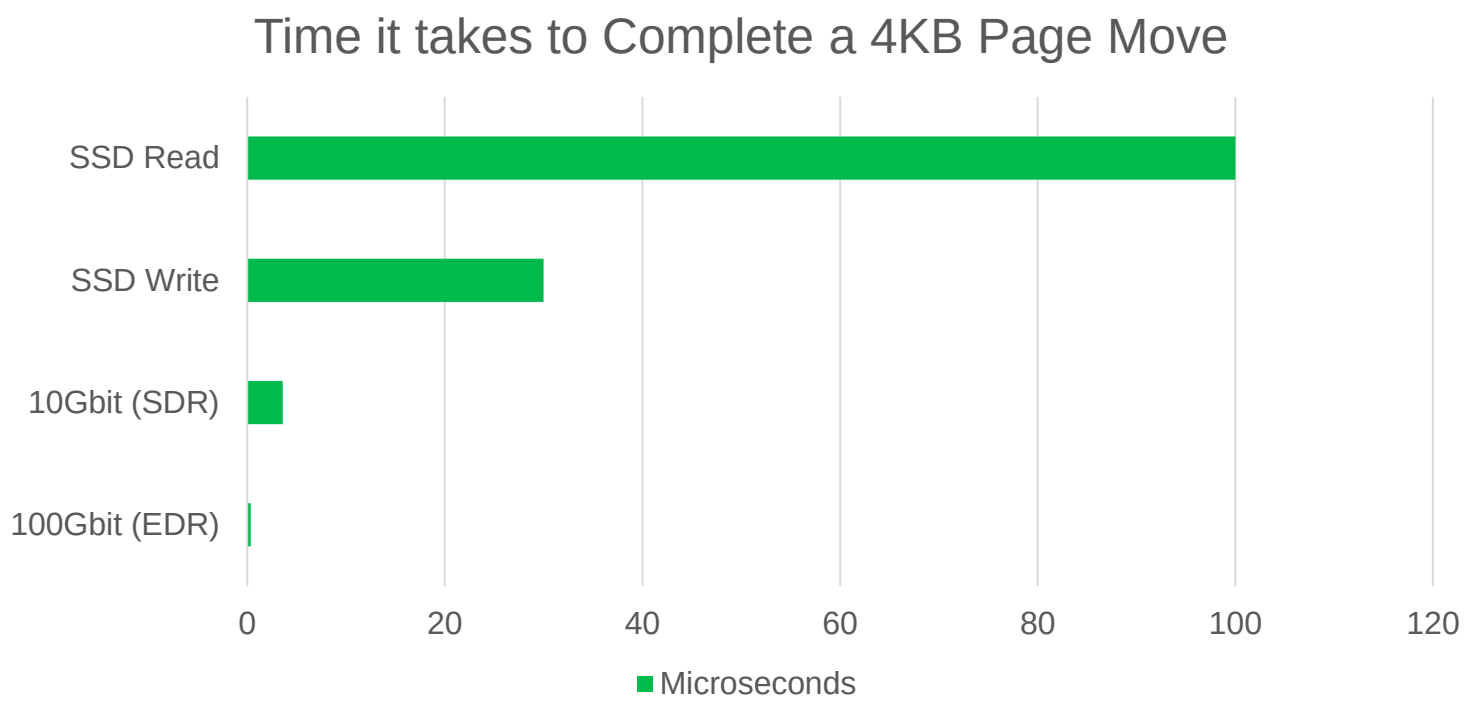
How do we do it…

WEKA.IO

World's Fastest File System™

Why Data Locality is Irrelevant

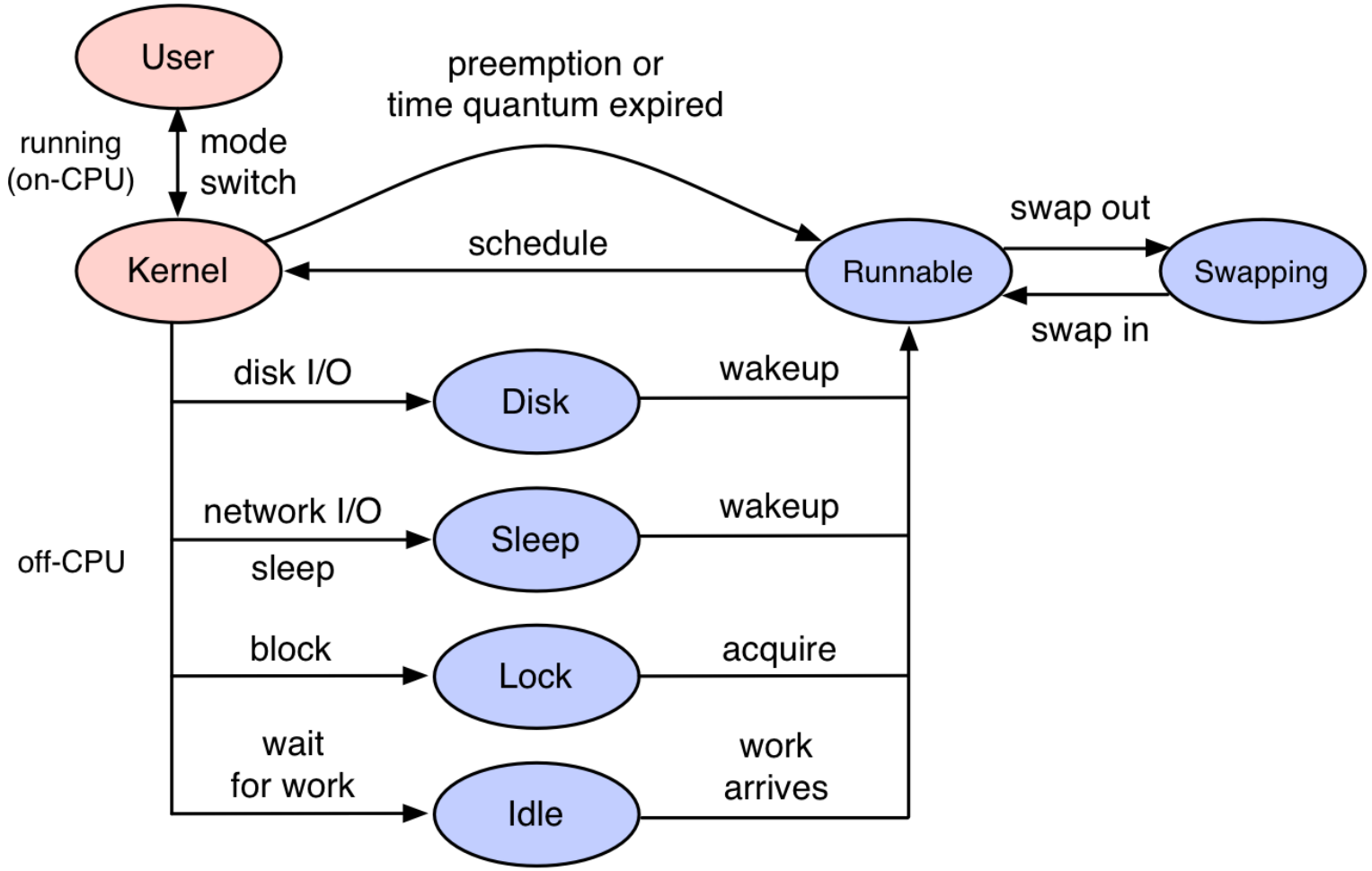
- Modern networks on 10Gbit Ethernet are 30 times faster than SSD for reads and 10 times faster than SSD for writes
- With right networking stack, shared storage is faster than local storage



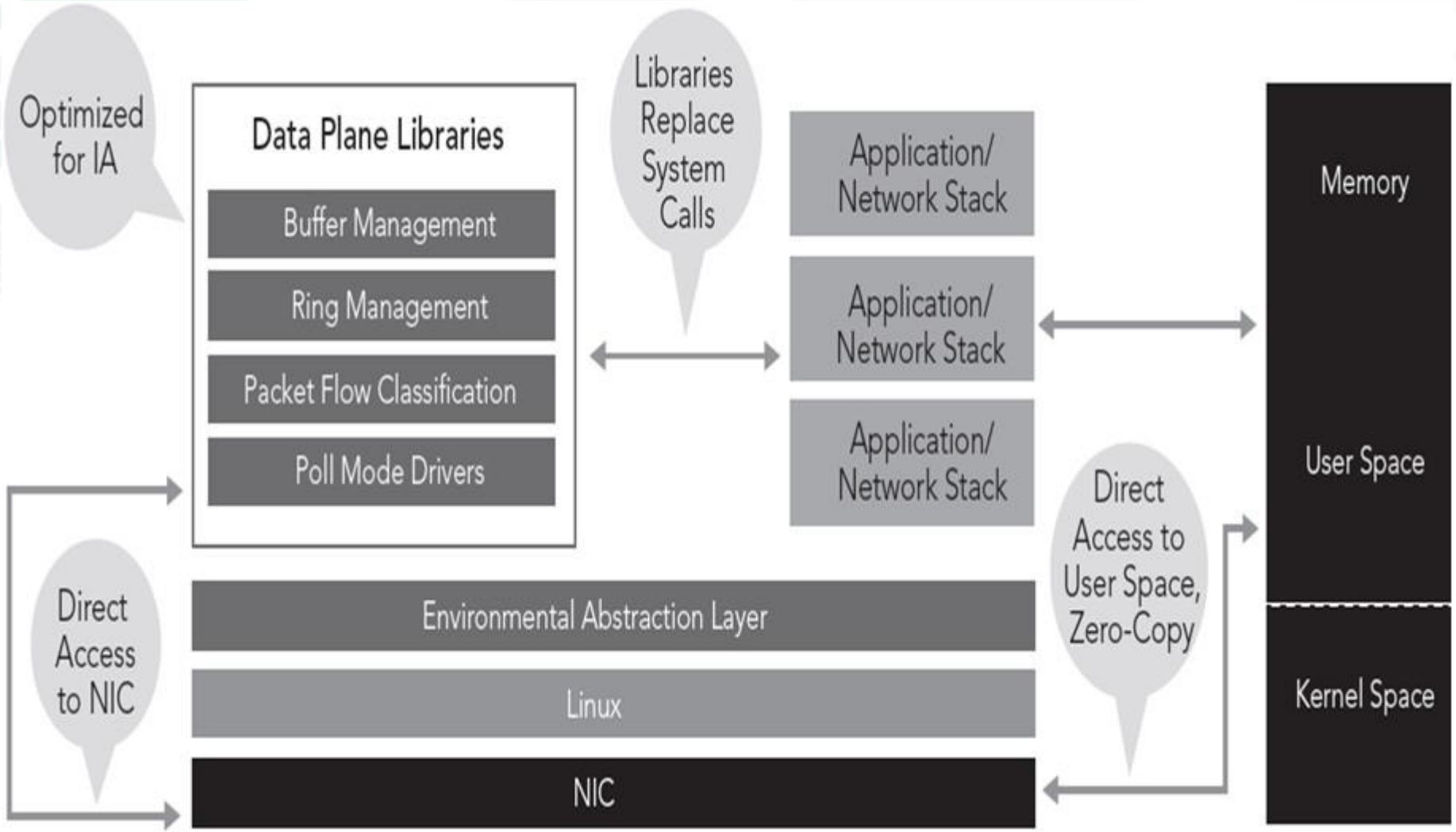
The Kernel is designed for multi-tasking…

- The system needs to be notified of the new packet and pass the data onto a specially allocated buffer sk_buff struct (Linux allocates these buffers for every packet).
- To do this, Linux uses an interrupt mechanism: an interrupt is generated several times when a new packet enters the system. The packet then needs to be transferred to the user space.
- As more packets have to be processed, more resources are consumed negatively impacting the overall system performance.
- sk_buff struct: the Linux network stack was originally designed to be compatible with as many protocols as possible. As such, metadata for all of these protocols is included in the sk_buff struct, but that's simply not necessary for processing specific packets. Because of this overly complicated struct, processing is slower than it could be.
- When an application in the user space needs to send or receive a packet, it executes a system call. The context is switched to kernel mode and then back to user mode.
  This consumes a significant amount of system resources

Context Switching is wasteful…



DPDK by-passes the Kernel…



Components of DPDK:

•**Environment Abstraction Layer (EAL)** : It is responsible for gaining access to low-level resources such as hardware and memory space.

•**Memory Manager**: Responsible for allocating pools of objects in memory. A pool is created in huge page memory space and uses a ring to store free objects.

•**Buffer Manager**: Reduces by a significant amount the time the operating system spends allocating and de-allocating buffers using advanced techniques such as Bulk Allocation, Buffer Chains, Per Core Buffer Caches etc.

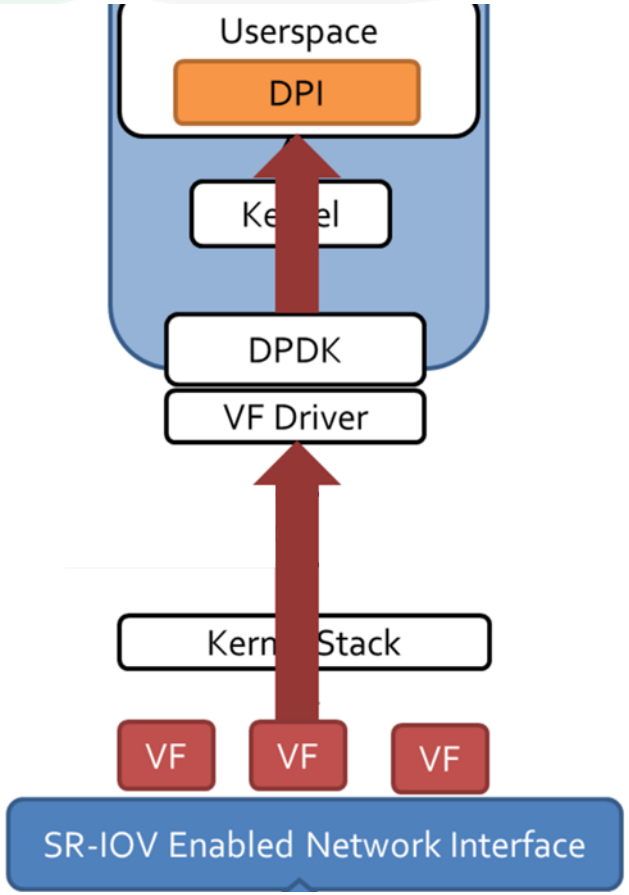
•**Queue Manager**: Implements safe lockless queues, instead of using spinlocks, that allow different software components to process packets, while avoiding unnecessary wait times.

•**Packet Flow Classification:** DPDK Flow Classifier implements hash based flow classification to quickly place packets into flows for processing.

•**Poll Mode Drivers**: Instead of using Interrupts and wasting CPU attention, PMD uses polling (scanning the NIC whether packets arrived or not), and doesn't disturb the CPU at all!

We also use SR-IOV and SPDK

Multiple Virtual NICs



Benefits of SPDK

SPDK
more performance from CPUs, non-volatile media, and networking

- Up to **10X MORE** IOPS/core for NVMe-oF* vs. Linux kernel
- Up to **8X MORE** IOPS/core for NVMe vs. Linux kernel
- Up to **50% BETTER** Tail Latency for RocksDB workloads
- **FASTER** TTM/ **LESS** RESOURCES than developing components from scratch
- Provides Future Proofing as NVM technologies increase in performance

http://www.intel.com/performance

Software Architecture

Runs in LXC Container for isolation

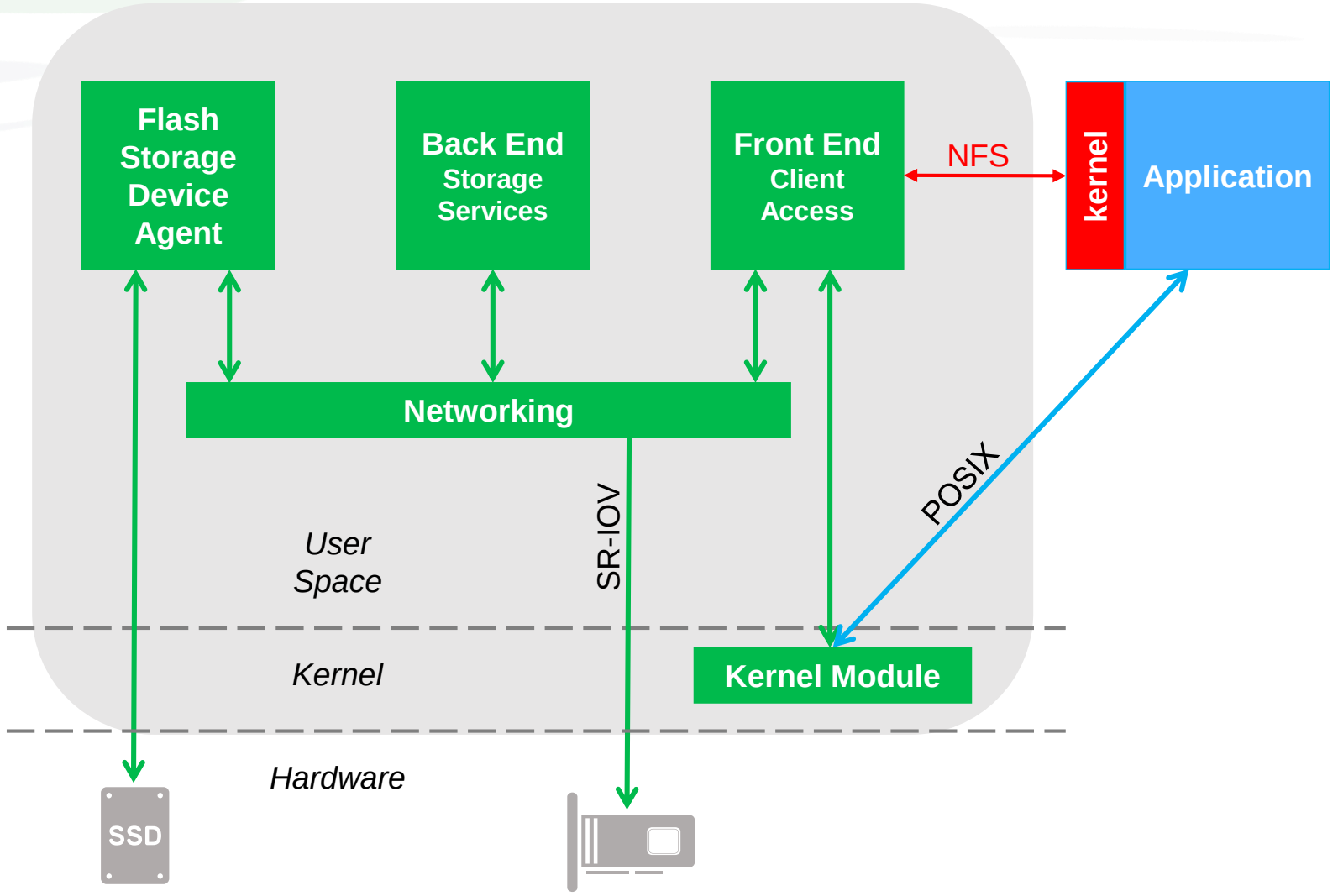
Kernel Module
- VFS Interface

Front End
- POSIX Client
- Other protocol access

Back End
- Data placement
- Data protection
- File system metadata
- Tiering

Networking
- SRIOV for network stack
- I/O bypasses network stack

Storage Agent
- I/O bypasses the kernel

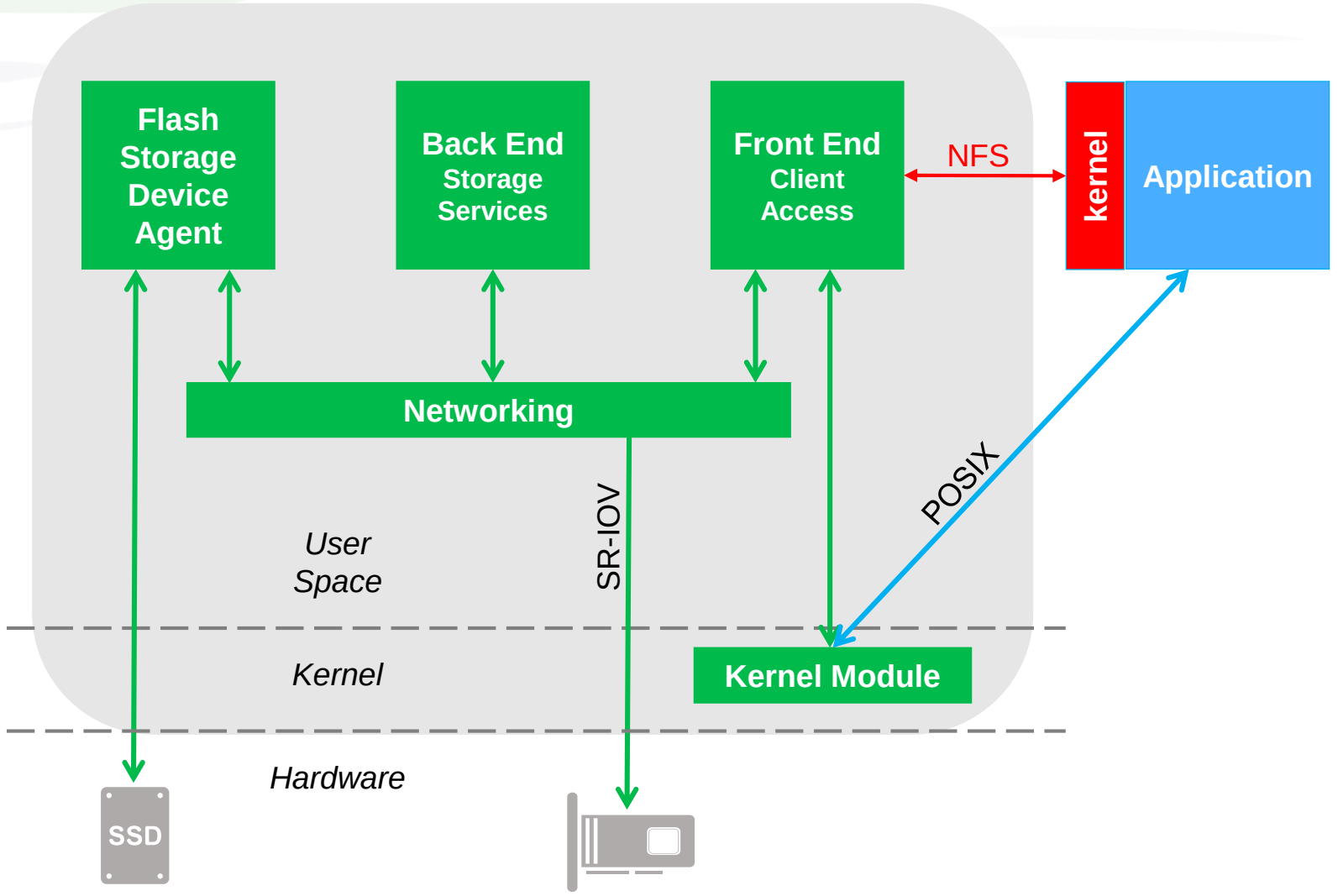


WekaIO Data Path

- Application IO (file operations)
  - Access WekaIO Client *as* Local FS
  - User-Space, Low-Latency
  - POSIX-complete, high-perf
  - Kernel Module for VFS integration

OR

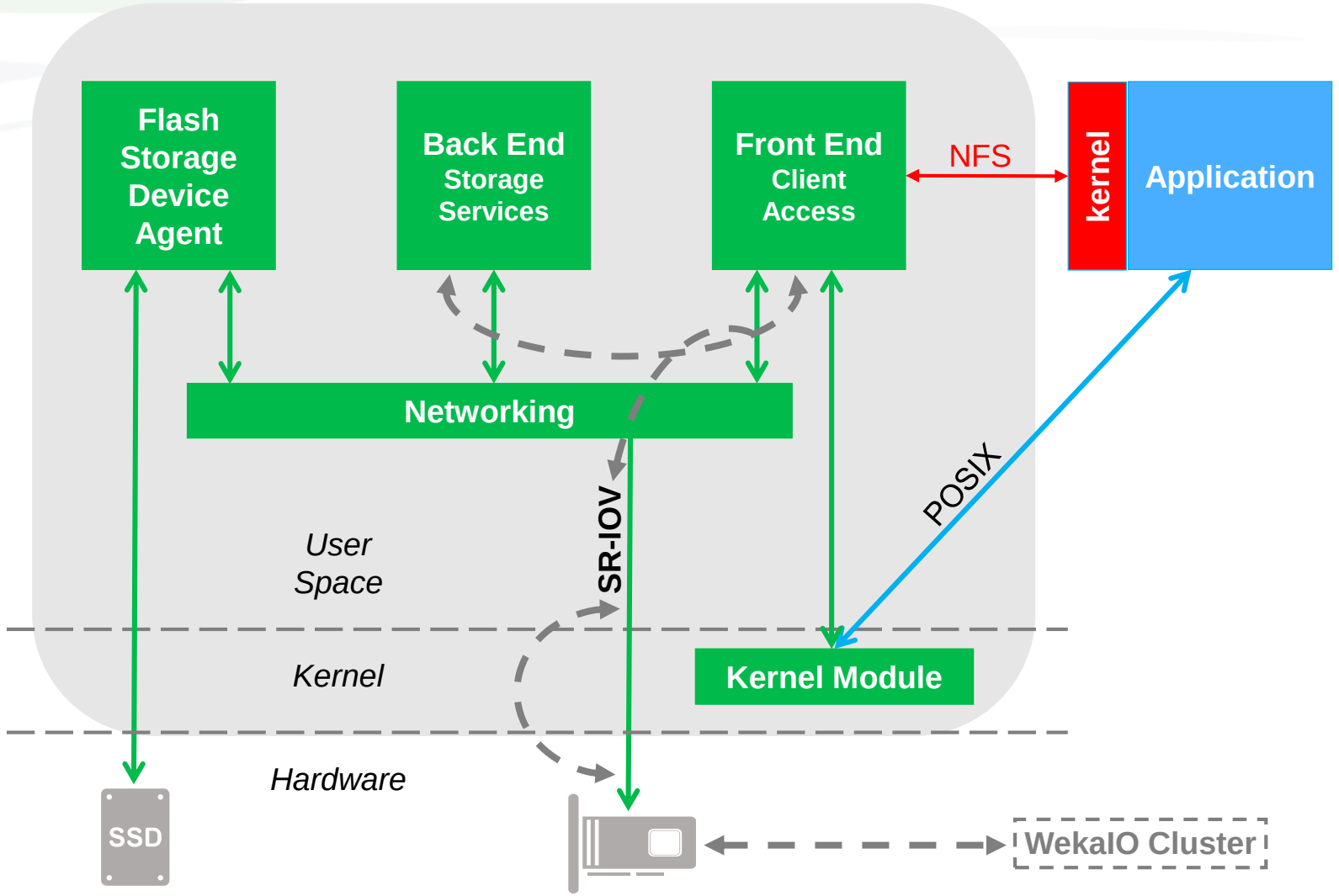
  - Client-side NFS
  - Bottlenecked by Kernel
  - Handled by WekaIO's Front End



WekaIO Data Path

- Application IO (file operations)
  - Access WekaIO Client *as* Local FS
  - User-Space, Low-Latency
  - POSIX-complete, high-perf
  - Kernel Module for VFS integration

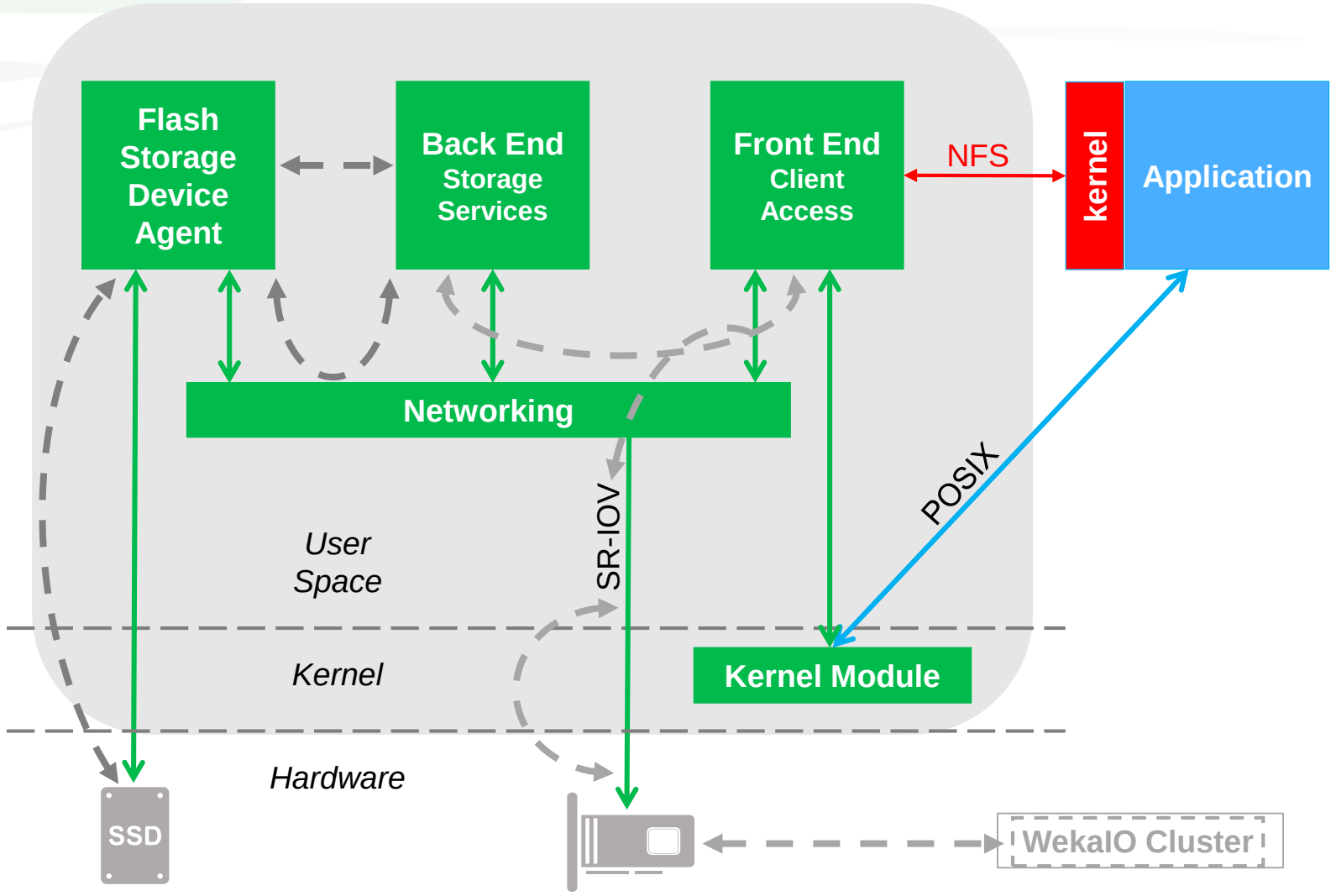
  OR
  - Client-side NFS
  - Bottlenecked by kernel
  - Handled by WekaIO's Front End
- WekaIO Front-Ends are Cluster-Aware
  - Incoming Read Requests optimized re Location & Loading Conditions
  - Incoming Writes can go anywhere
  - Metadata fully distributed
  - No redirects required
- SR-IOV optimizes Network access



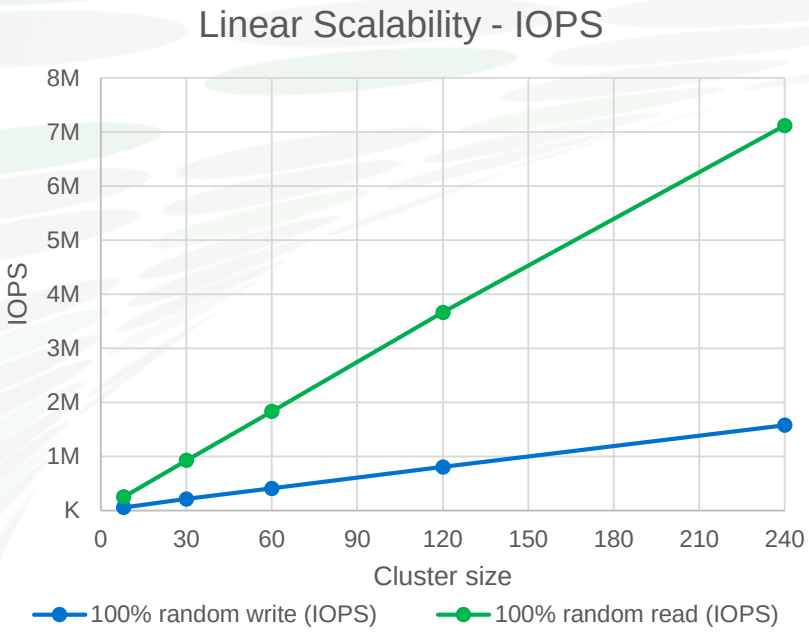
WekaIO Data Path

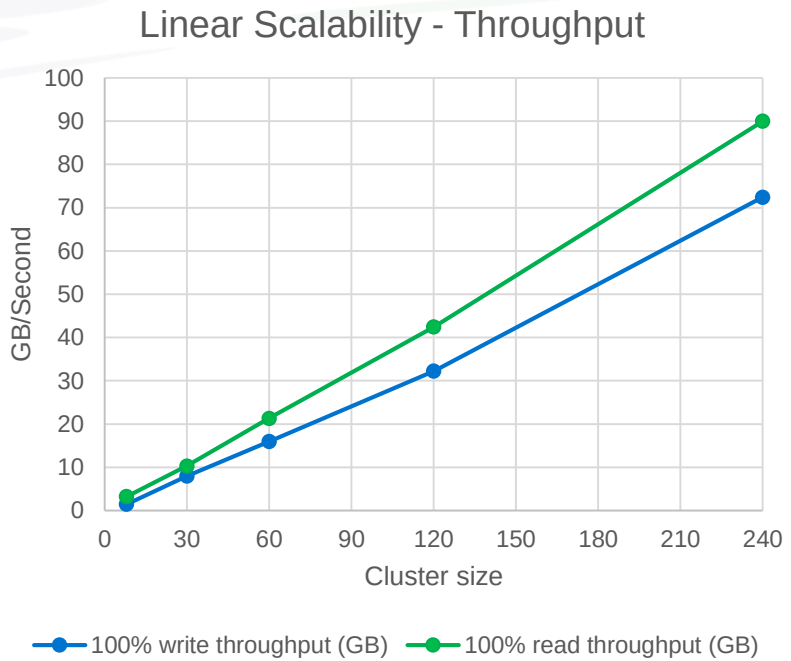
- Application IO (file operations)
  - Access WekaIO Client *as* Local FS
  - User-Space, Low-Latency
  - POSIX-complete, high-perf
  - Kernel Module for VFS integration

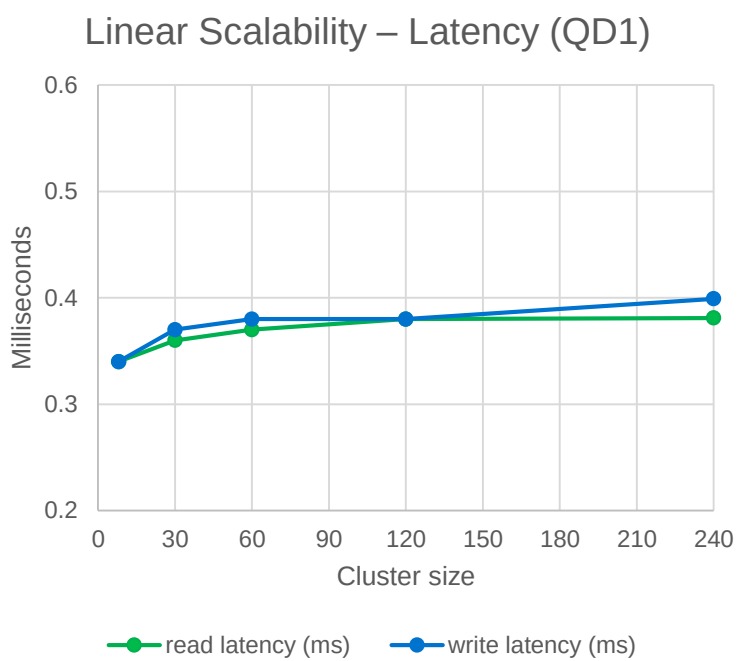
  OR
  - Client-side NFS
  - Bottlenecked by kernel
  - Handled by WekaIO's Front End
- WekaIO Front-Ends are Cluster-Aware
  - Incoming Read Requests optimized re Location & Loading Conditions
  - Incoming Writes can go anywhere
  - Metadata fully distributed
  - No redirects required
- SR-IOV optimizes Network access
- WekaIO directly accesses NVMe flash
  - Bypassing kernel, better perf



File System Scales **Linearly** with Cluster Size



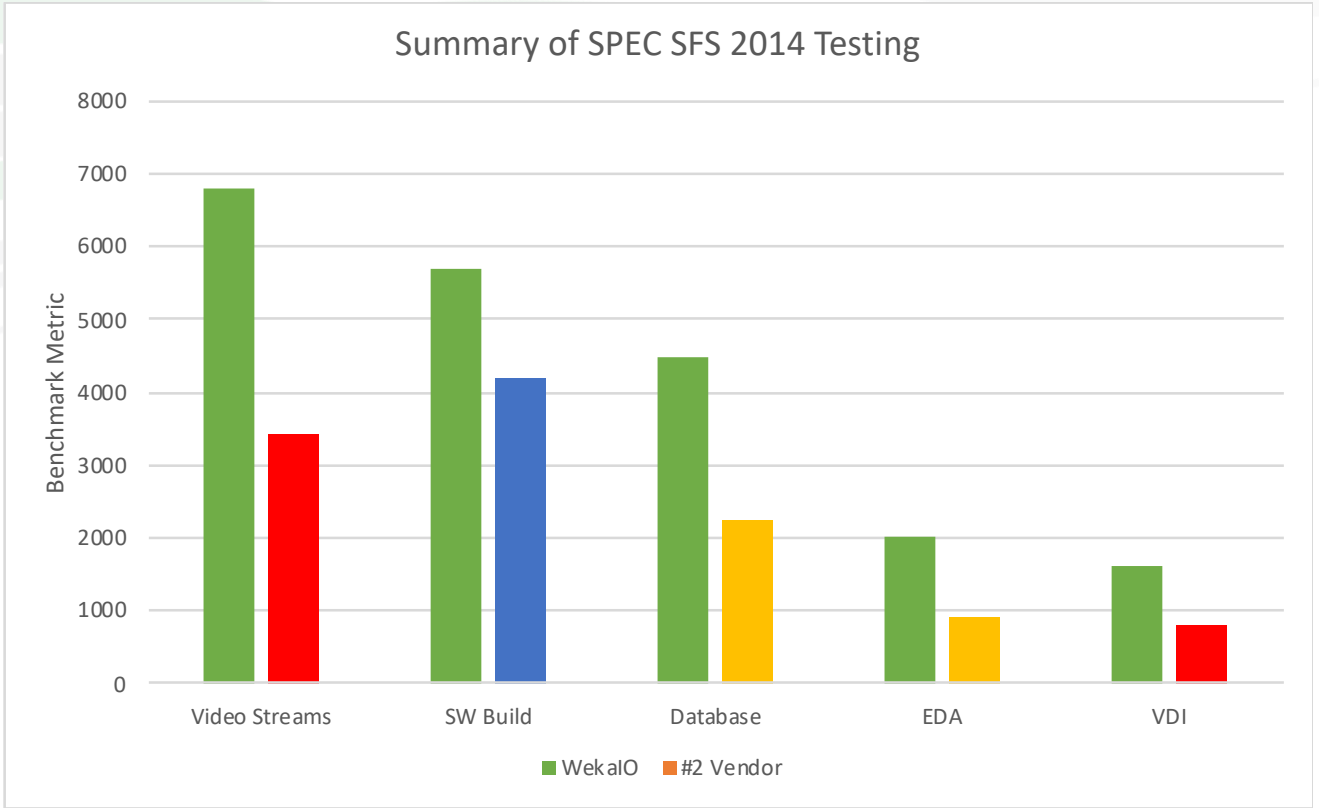




~30K OPS/AWS Instance
~375MB/sec/AWS Instance
<400 microsecond latency

Test Environment – 30-240 R3.8xlarge cluster, 1 AZ, utilizing 2 cores, 2 local SSD drives & 10GB of RAM on each instance. About 5% of CPU/RAM.

WekaIO Matrix #1 File System on SPEC



WekaIO
Oracle
NetApp
DDN

Benchmark	#1 Position	Score	ORT (ms)	#2 Position	Score	ORT (ms)
Software build	WekaIO	5700	0.26	NetApp	4200	0.78
Database	WekaIO	4480	0.34	Oracle	2240	0.78
Engineering design	WekaIO	2000	0.48	Oracle	900	0.61
Video streams	WekaIO	6800	1.56	DDN	3400	50.07
Virtual desktop	WekaIO	1600	0.48	DDN	800	2.58

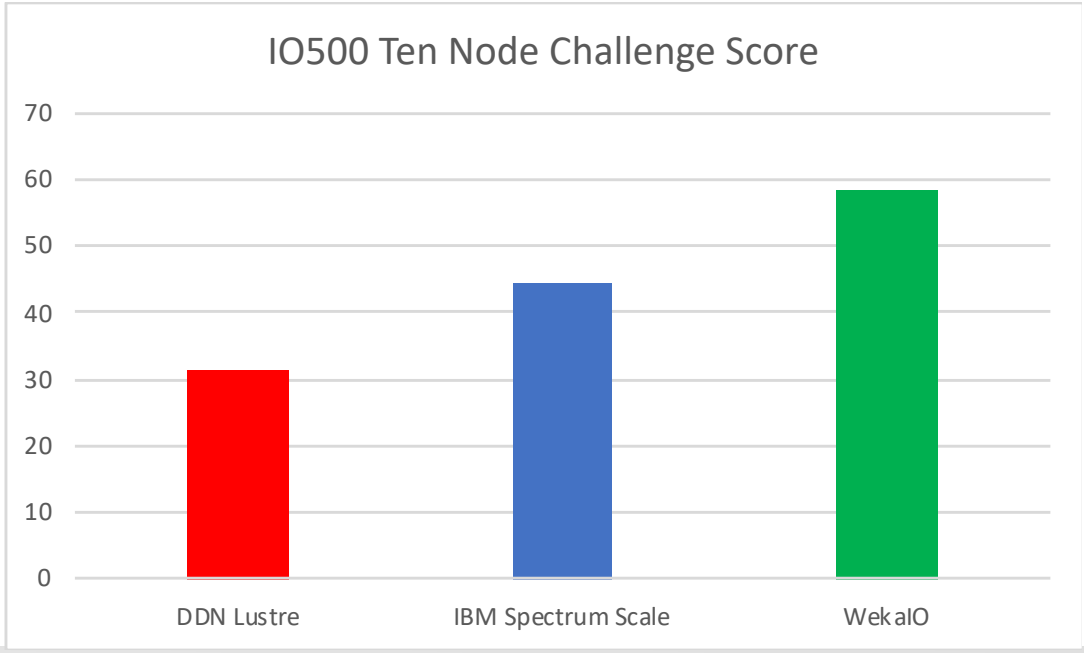
#1 File System on the IO500 Test

Test measures bandwidth and metadata

https://www.vi4io.org/io500/list/19-01/10node

#	information							io500		
	institution	system	storage vendor	filesystem type	client nodes	client total procs	data	score	bw	md
									GiB/s	kIOP/s
1	WekaIO		WekaIO		10	700	zip	58.25	27.05	125.43
2	Oak Ridge National Laboratory	Summit	IBM	Spectrum Scale	10	160	zip	44.30	9.84	199.48
3	DDN	Bancholab	DDN	Lustre	10	240	zip	31.50	6.33	156.69

- 31% better than World's largest Supercomputer
- 85% better than Lustre



Thank you.

WEKA.IO

https://docs.weka.io

https://start.weka.io