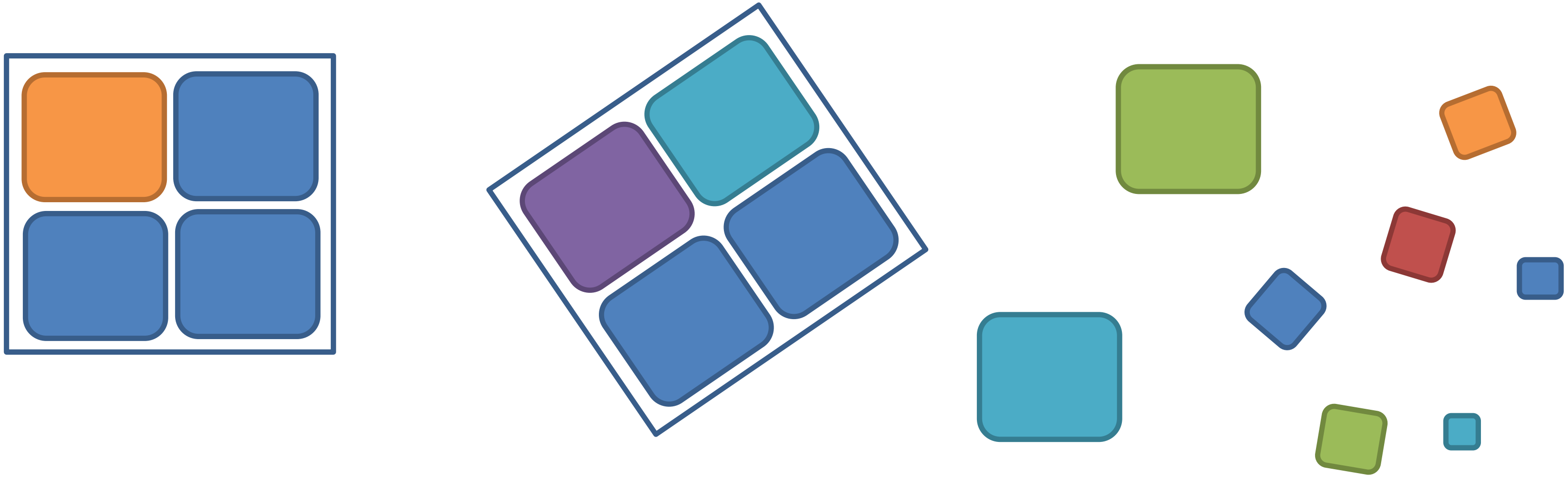
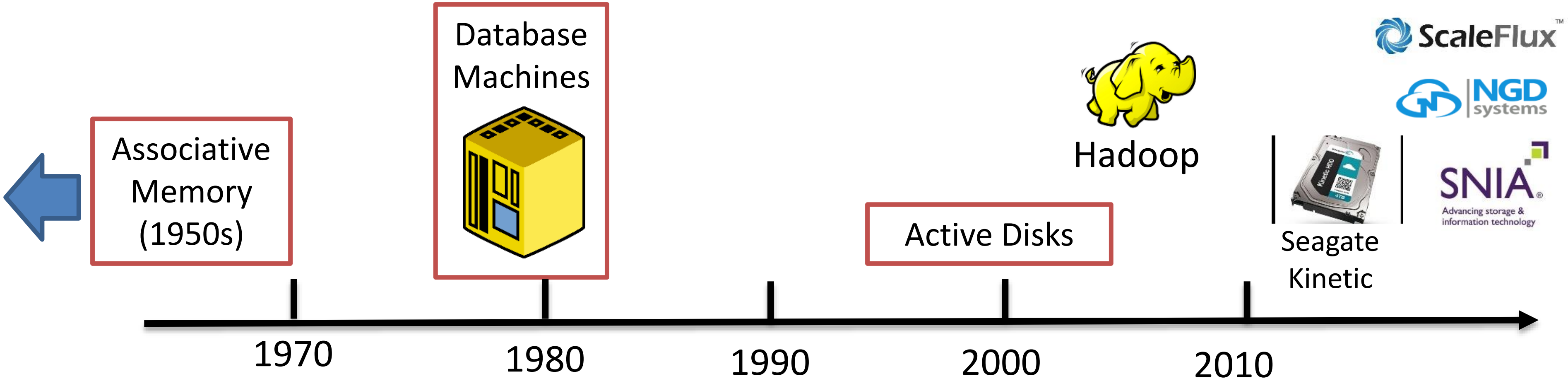
Respecting the block interface – computational storage using *virtual* objects



Ian F. Adams, John Keys, Michael P. Mesnier

A brief history of computational storage



Associative Memory (1950s)

Database Machines

Active Disks

Hadoop

Seagate Kinetic

ScaleFlux

NGD systems

SNIA Advancing storage & information technology

1970

1980

1990

2000

2010

Simple concept with a long history

- Move the compute to the data
- Associative memory, database machines, active disks, key-value HDD…

Why didn’t it gain widespread adoption?

- Short version: wasn’t quite worth it… ***until now***

What's changed?

Very high density, high-performance storage is here

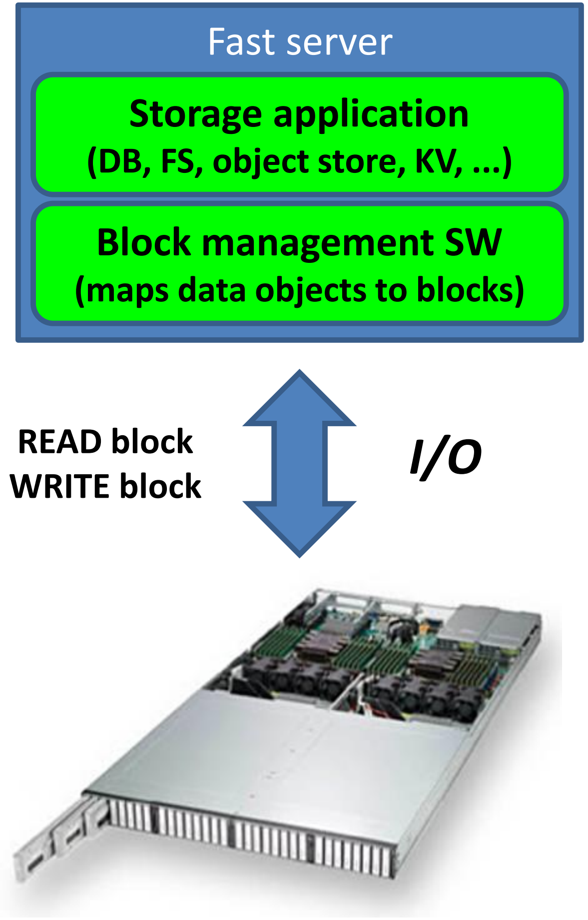
- 16-32 TB drives are here, 100+TB SSDs are coming
  - 1PB in a 1U server
- All this behind NICs, I/O controllers, devices, etc.

Large scale disaggregated ***block*** storage is here (NVMeoF)

- Enables "diskless" storage stacks
- Greater flexibility, but yet more I/O traffic

Devices and targets are more powerful

- More flexibility and headroom to work with
  - (also, we're Intel and like hardware ☺)



Fast server

Storage application
(DB, FS, object store, KV, …)

Block management SW
(maps data objects to blocks)

READ block
WRITE block

I/O

Moving compute into storage

(to avoid an I/O bottleneck)

Moving compute into storage

Step 1. Teach the storage about data objects

- Files, objects, DB records, key-value pairs, …

Step 2. Provide a way to program storage (API)

Step 3. Implement compute methods in storage

- E.g., search, compress, checksum, resize, …

Object or file-based storage makes this process straightforward

BUT, storage is fundamentally ****still**** *built on blocks!*

Challenge 1:

Moving compute into storage

^

block

Object *Awareness*

Recall Step 1: Teach storage about objects

- Constraint: we need to talk block storage

Prior experience makes us leery of changing low-level storage interfaces

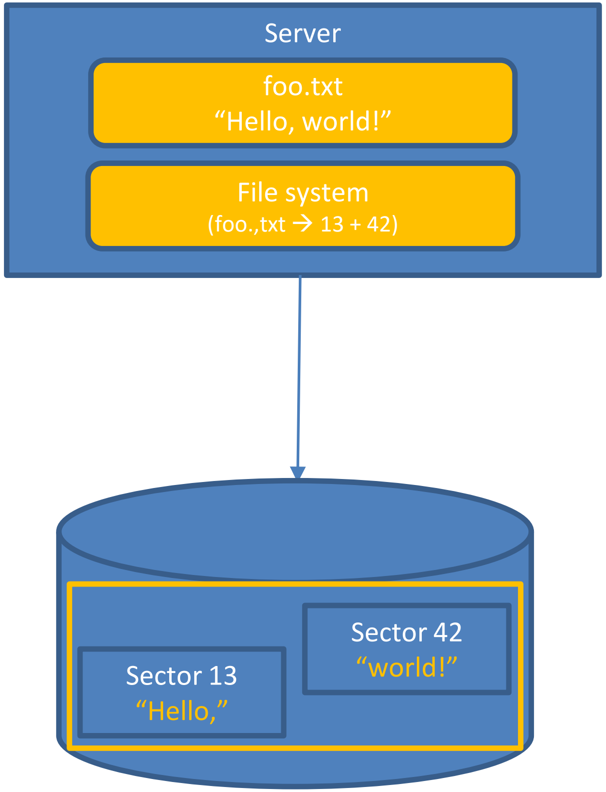
- E.g., uphill battle for KV drives

Can we make block storage *object aware* without…

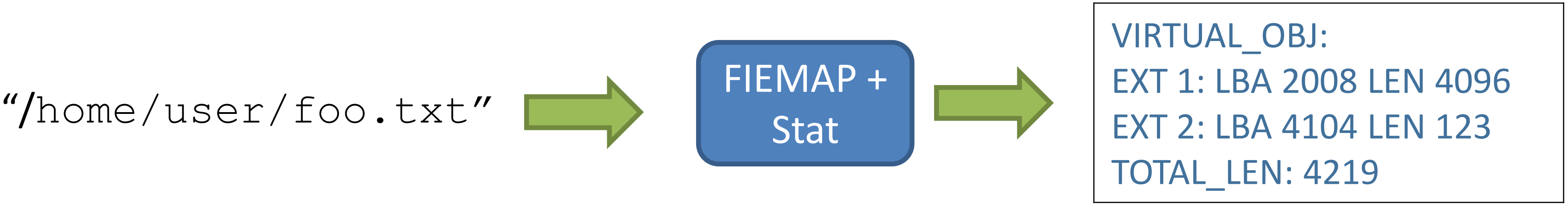
- Changing the interface
- Adding a lot of state and complexity

We need to consider

- Host and target data consistency, input vs output, non-sector aligned data, transport considerations (bidirectional transfers), chained operations, permissions…



Introducing virtual objects (step 1 of 3)



“`/home/user/foo.txt`”

FIEMAP + Stat

VIRTUAL_OBJ:
EXT 1: LBA 2008 LEN 4096
EXT 2: LBA 4104 LEN 123
TOTAL_LEN: 4219

Virtual object:

- *An ephemeral* mapping of blocks to make block storage *object aware*
  - Don’t have to turn block storage into object storage
  - Stateless: mapping is only valid for duration of an operation
  - Can be used for both input and output
- Complementary to existing stacks built on block storage
  - Object, KV store, file, etc.

This is step 1: teach the block storage about objects

Programmability (step 2 of 3)

Compute Descriptor

VIRTUAL_OBJ:
EXT 1: LBA 2008 LEN 4096
EXT 2: LBA 4104 LEN 123
TOTAL_LEN: 4219

OPCODE: “search” ARG: “baz”

Virtual objects are embedded in **compute descriptors**

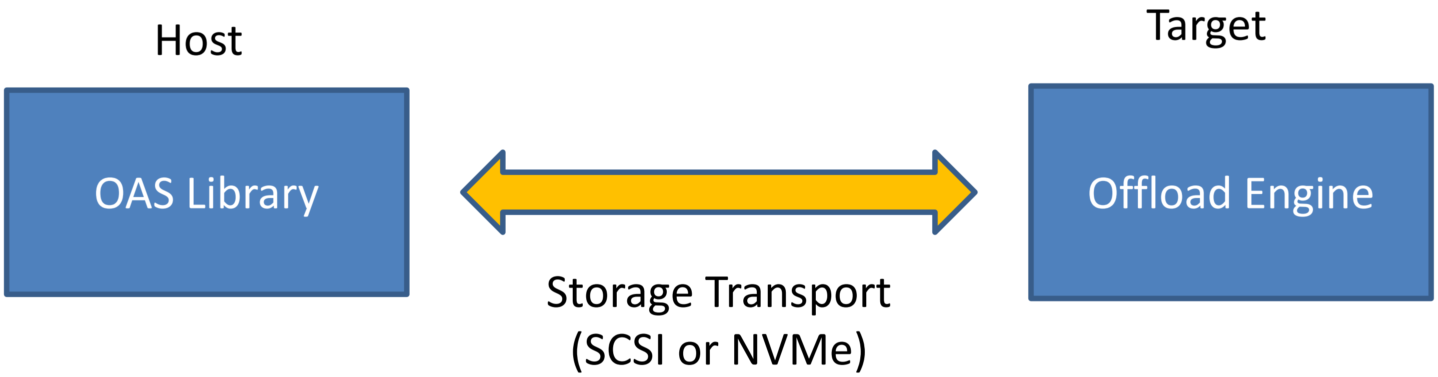
- Add arguments and operations for computing inside block storage
- Can have multiple input and output virtual objects

Descriptors are ***block-protocol compatible***!

- For SCSI and NVME, works as a vendor specific **EXEC** command
- Small results can be returned as a payload, larger results written to **output objects**

This is step 2: provides a way to program storage

Implementing offloads (step 3of 3)



Storage Transport
(SCSI or NVMe)

Object Aware Storage (OAS) Library handles host/app interactions

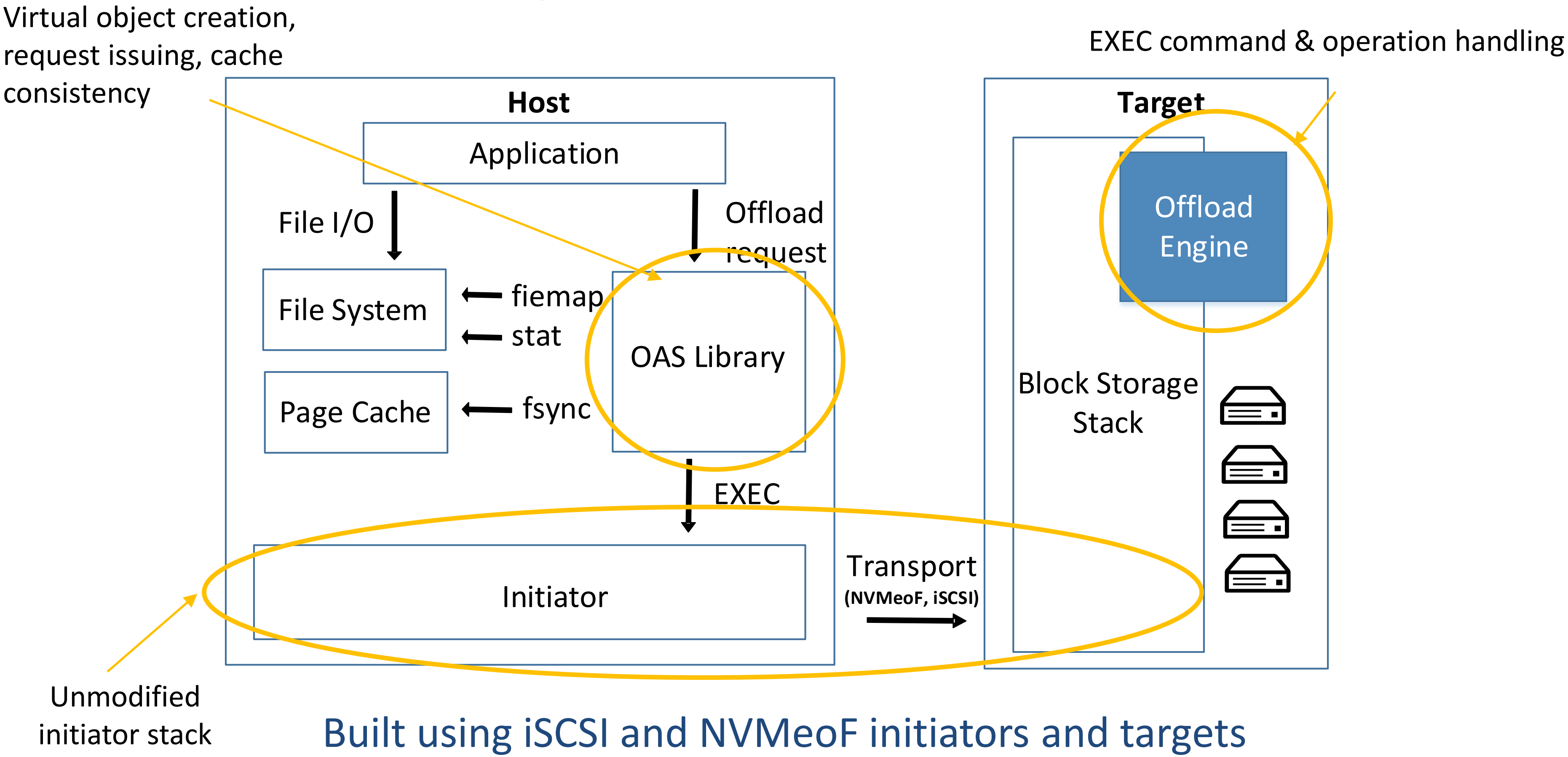
- Cache consistency
- Creating and allocating virtual objects
- Building and transporting compute descriptors

Offload Engine: interprets EXEC command an descriptors

- Implement our methods like checksum, search, etc.

This is step 3: provides a way to implement operations

Prototype Architecture + Flow



Built using iSCSI and NVMeoF initiators and targets

Evaluation

Experimental setup

2 servers connected via 40 GbE

- Target and Host: Dual Xeon Gold 6140s, Dual Xeon E5-2699 v3s
  - Runs NVMeoF stack, handles offloads
- 8 P4600 NVMe SSDs (~3 GB/s per drive)
- Benchmark:
  - OASBench (in-house benchmarking utility)
  - 100 16 MB files per SSD, 48 worker threads

Focused on checksum offload

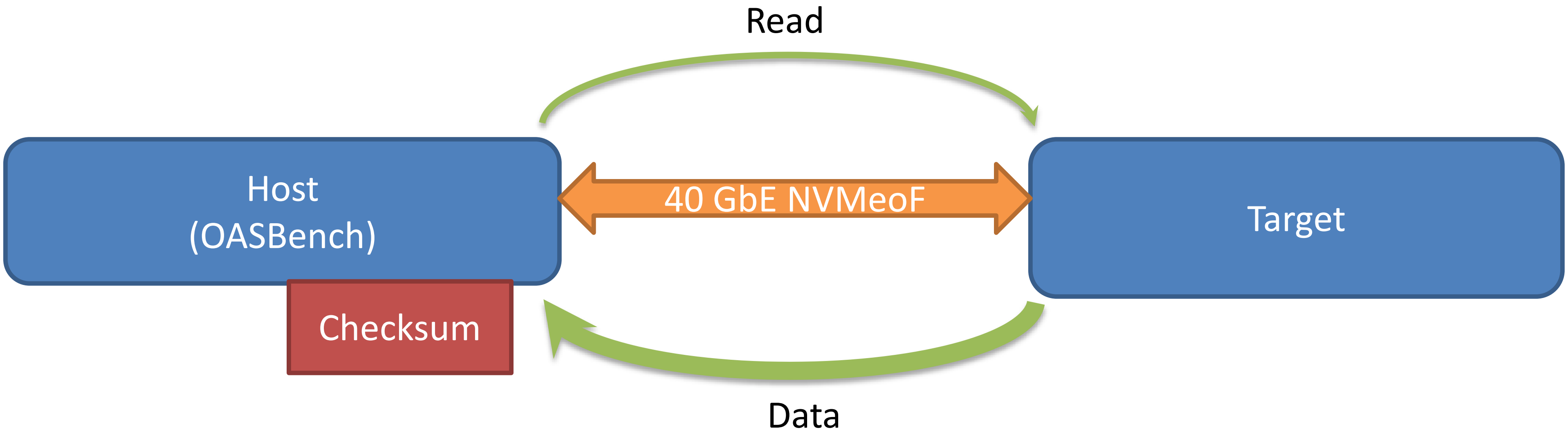
- "Bitrot" detection for object storage
- Modern hashes are I/O bound

Host
(OASBench)

40 GbE NVMeoF

Target

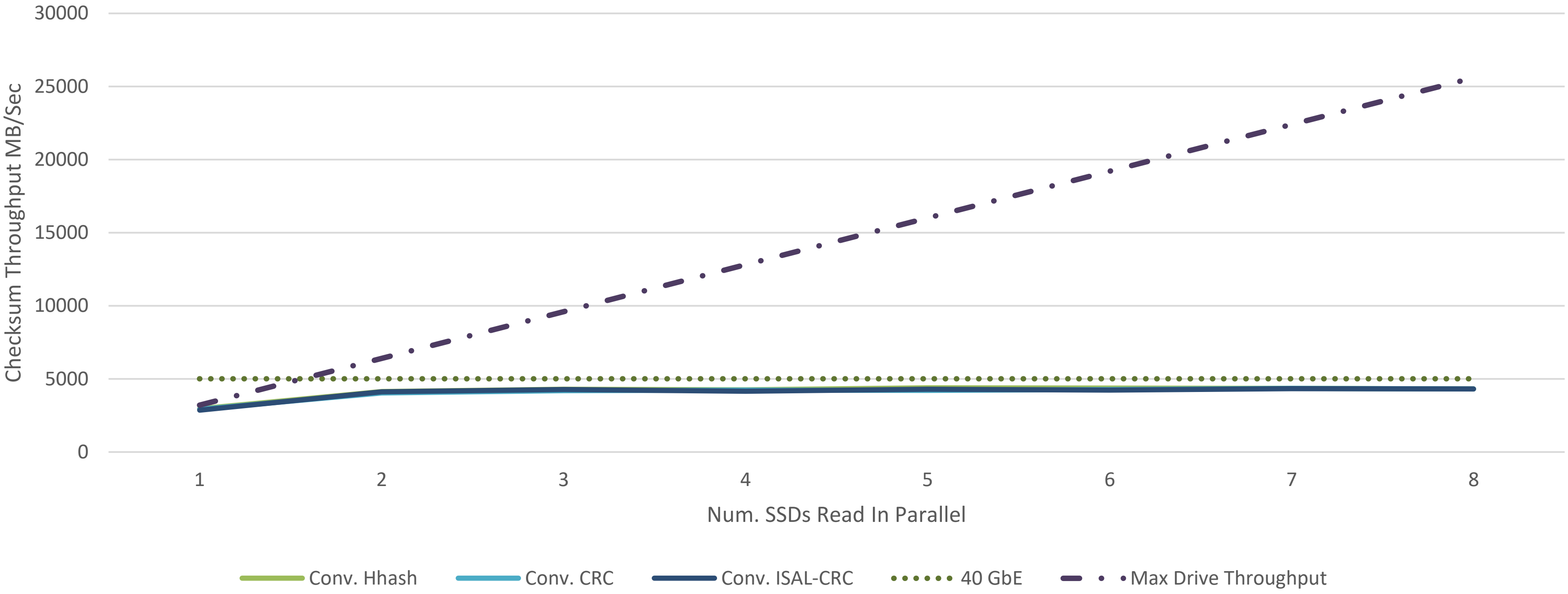
Experiment 1: Conventional Access



Read file/object data from target to host, and compute checksum

- Expect to be bottlenecked by the 40 GbE link

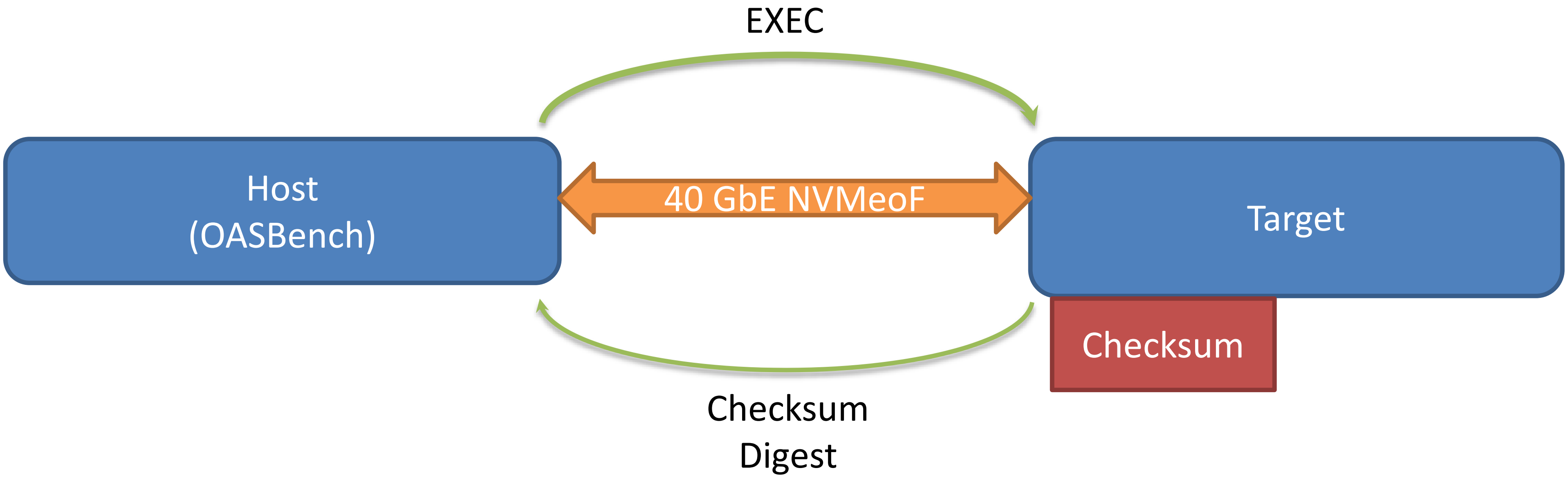
Conventional operations



Conventional operations: data is pulled to the host before computation

- Quickly bottlenecked by 40 GbE network
- <2 SSDs worth of throughput

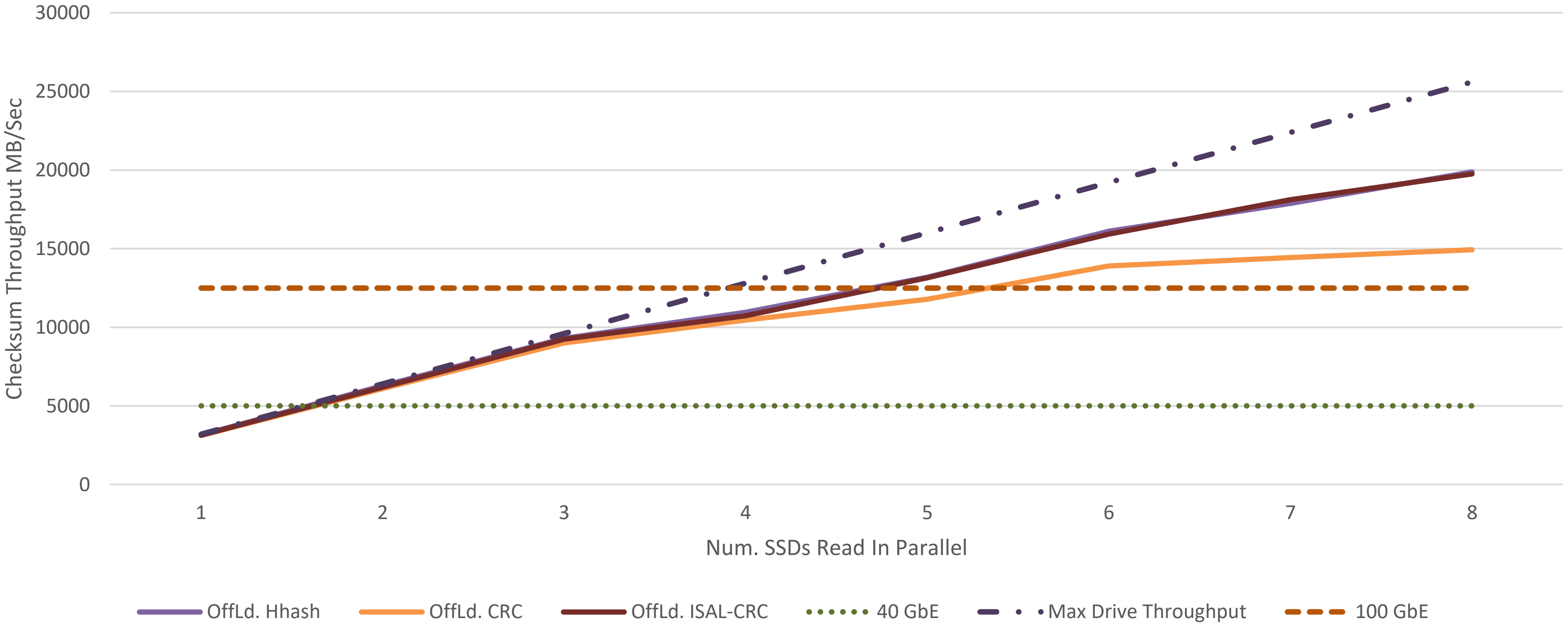
Experiment 2: Offloaded Access



Issue EXEC command with virtual objects

- Target computes checksum *in-situ* and returns digest
- Network bottlenecks should go away

Offloaded operations



Offloaded operations are run in the storage target

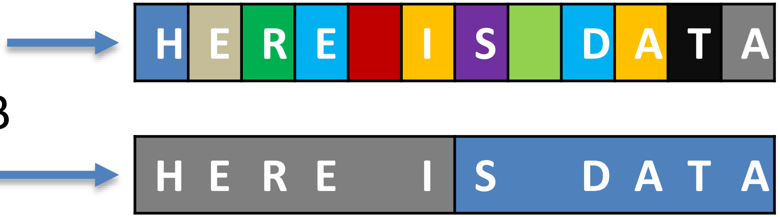
- Bypasses the 40 GbE bottleneck and scales with the number of SSDs being hit
- 40 GbE link bypass even what could be provided from 100 GbE!
  - No longer transport bound!
- >99% reduction in network traffic, along with up to 3x speedups (Not shown)
  - Implemented in Ceph, Swift and MinIO

Challenge 2:
Handling Distributed, Striped Data

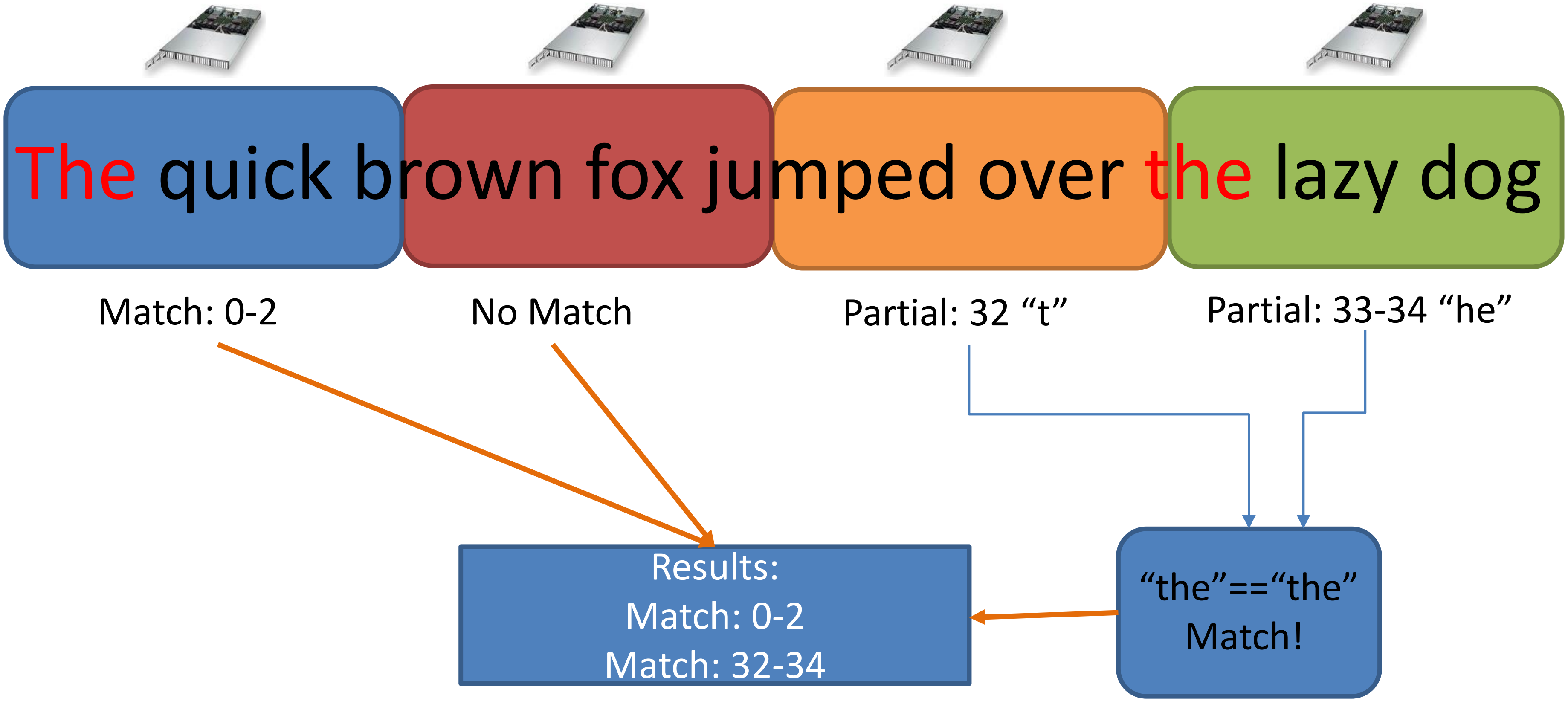
Computational Storage and EC

Trends in Data Striping

- Erasure coded (EC) deployments have exploded beyond traditional RAID
  - RAID chunks in low bytes to KiB ranges
    - Very difficult to offload computations
  - EC chunks in hundreds of KiB to low MiB
    - Individual elements easily found
- Large volumes of data have well defined structure and elements
  - E.g., CSVs, JSONs, dense matrices, etc.



Our Solution



The quick brown fox jumped over the lazy dog

Match: 0-2

No Match

Partial: 32 “t”

Partial: 33-34 “he”

Results:
Match: 0-2
Match: 32-34

“the”==“the”
Match!

Our solution is to leverage data structure and large stripe pieces

- Most work still done inside target
- Ambiguous “border” elements returned as “residuals” handled host-side

Ongoing and Future Work

Lots of other offloads (not enough time to cover)

- Image preprocessing for ML pipelines
  - >90% data movement reduction
- Merge, Sort, Search, LSM Compaction, CSV queries, microclassifiers…

We're not just for fabrics targets

- Methodology is compatible with devices as well

Industry involvement and engagement

Wrapping it Up!

Introduced virtual objects for computational *block* storage

- Prototypes in iSCSI and NVMeoF with a variety of offloads

Showed that handling distributed, striped data can be straightforward with large EC shards and (semi) structured data

We want collaborators!

- Working on open sourcing

Stay tuned for more updates from Intel ☺

Thanks for your attention!

Questions? Comments?

ian.f.adams@intel.com

john.keys@intel.com

michael.mesnier@intel.com

Extras/Backups

Applications are easy to adapt and enable

- Application integration isn't difficult
  - Example with our Golang bindings using iSCSI
- Client library is small
  - (< 500 LOC)
- New offloads are straightforward
  - Currently a combination of C libraries and kernel modules
  - Currently porting to full userspace implementations

```
/*path to talk to the scsi device*/
sgpath := "/dev/bsg/20:0:0:0"

/*Target file for operating on*/
fpath := "/mnt/oas_dev/test.txt"

/*Create the OAS Context*/
ctx := oas_client.OasCtx{sgpath}

/*Call MD5  method*/
oas_md5_resp := ctx.MD5(fpath)
```