Methods to achieve low latency and consistent performance

Alan Wu, Architect, Memblaze

zhongjie.wu@memblaze.com

2015/8/13

Software Defined Flash Storage System

Software defined flash storage system

NVMe SSD

SATA SSD

Commodity components / hardware

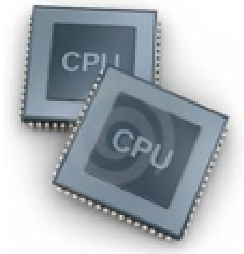
Commodity server / platform

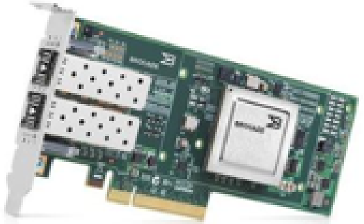
Memblaze provides software defined flash system – memOS

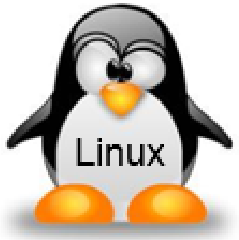
Design Challenges

- Low latency challenges
  - Write request with low latency
  - Interaction between read & write requests
  - Balance between bandwidth and latency
- Consistent performance challenges
  - Linux OS makes performance inconsistent
  - Multi-cores / NUMA affect performance consistency
  - How to keep low latency within high IOPS

Where's the Bottleneck of Flash System?



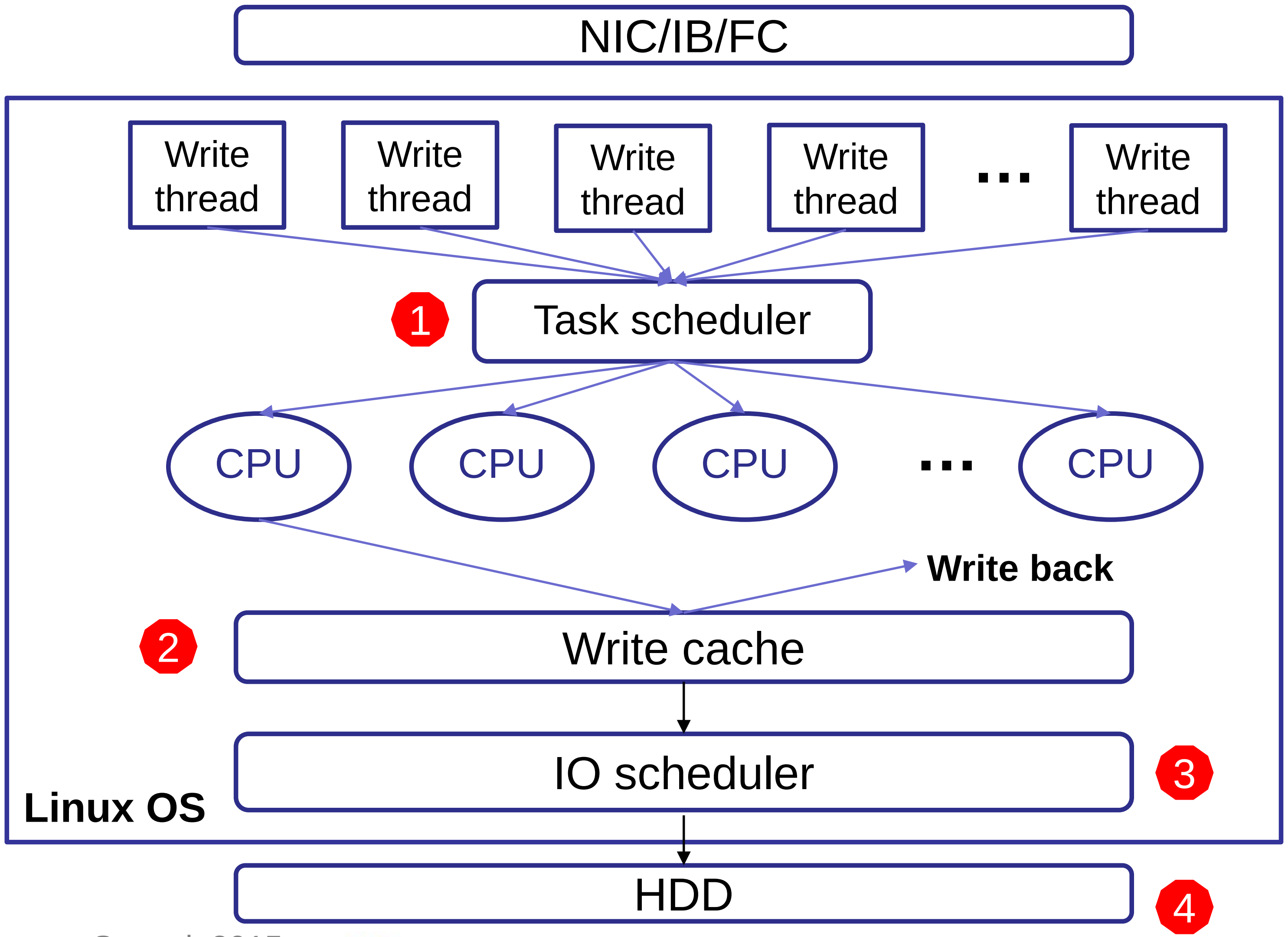




Who is bottleneck?

NUMA	Task Scheduler	IO scheduler	FC port
Interrupt	Memory copy	Disk interface	Iscsi
Multi-cores	Context switch	FTL GC	RDMA

Traditional Write Path Analysis



Initiator and exportation interface has performance bottleneck

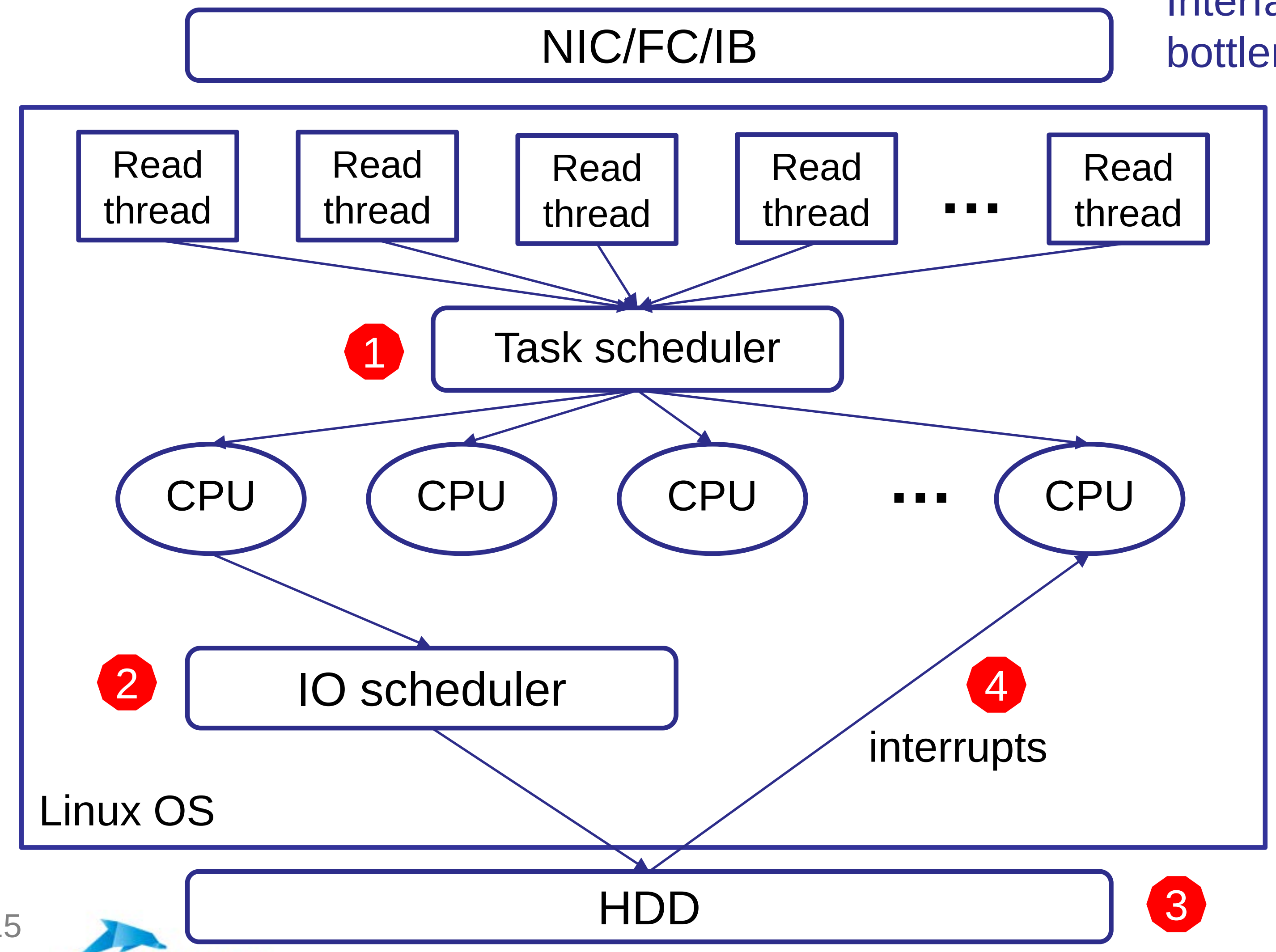
Introduce IO latency and jitter

Cache can reduce IO latency

Introduce IO latency and jitter

GC/Random write affect performance

Traditional Read Path Analysis



Interface has performance bottleneck

Introduce IO latency and jitter

Introduce IO latency and jitter

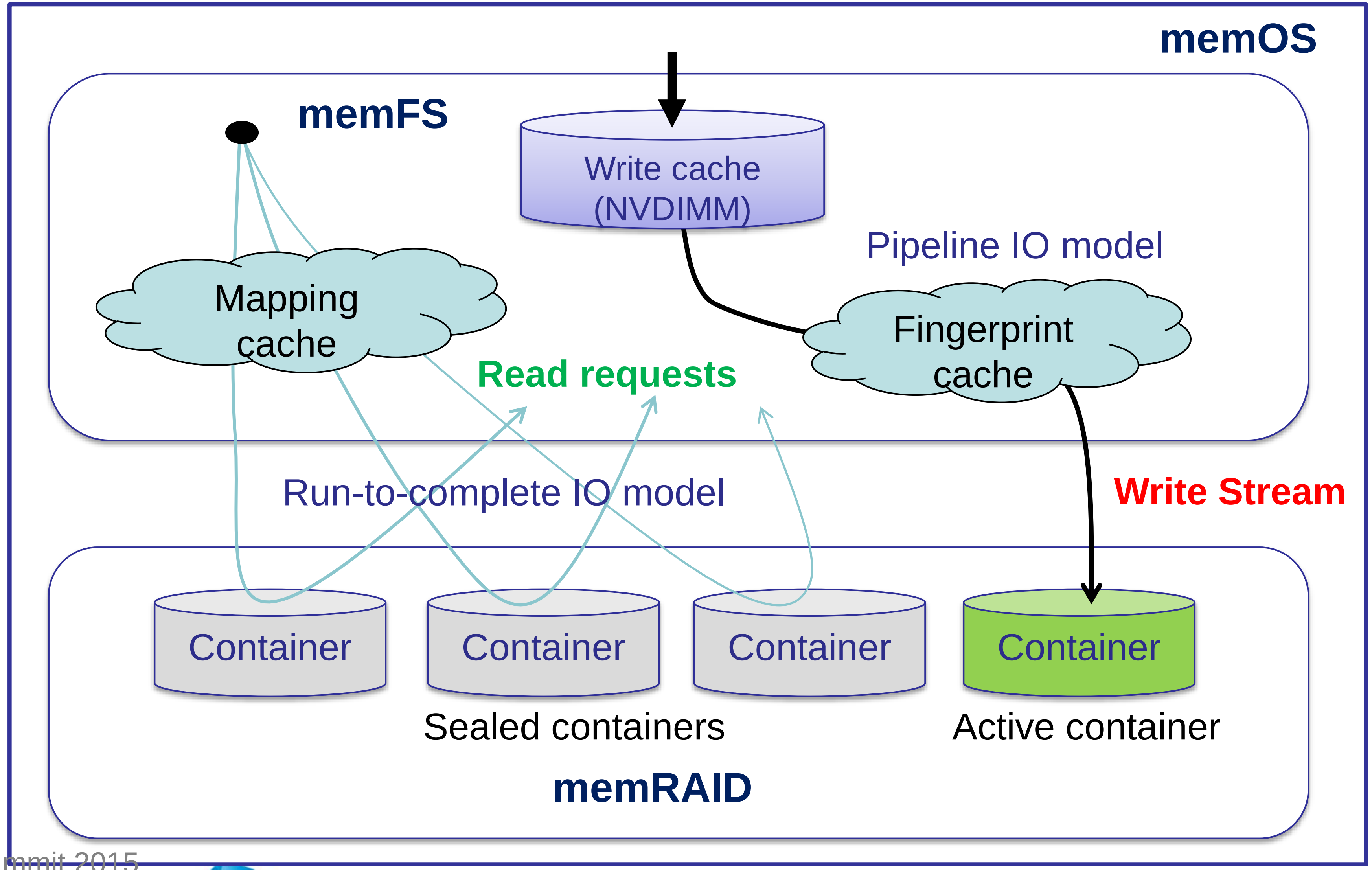
Interrupts affect performance

GC/write affect read performance

New Approach: RISL Software Architecture

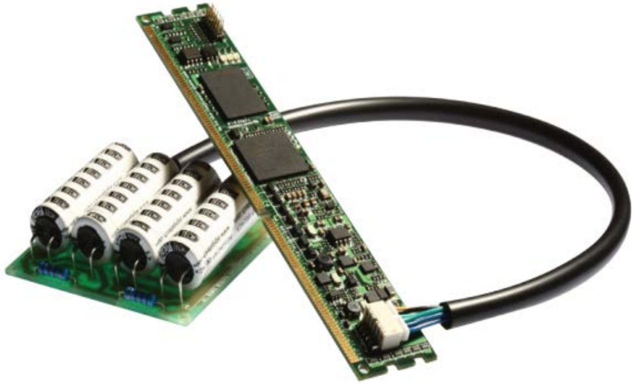
- SSD characteristics
  - Random write generates lots of mapping information and make GC busy
    - Sequential write can make FTL works in best condition
  - High random read performance
  - Write / erase operation affects read performance
- **Memblaze answer: RISL (patent filed by Memblaze)**
  - **R**andom **I**nput **S**tream **L**ayout
    - Whatever input IO patterns, **data layout on SSD is always sequential**
  - RISL Includes:
    - Non-volatile write cache: converts any write pattern into sequential
    - Separate read and write requests into different container (storage object)
    - Pipeline and run-to-complete IO model is used to handle write request
    - Run-to-complete IO model is used to handle read request

RISL Architecture



Introduce NVDIMM to Reduce Write Latency

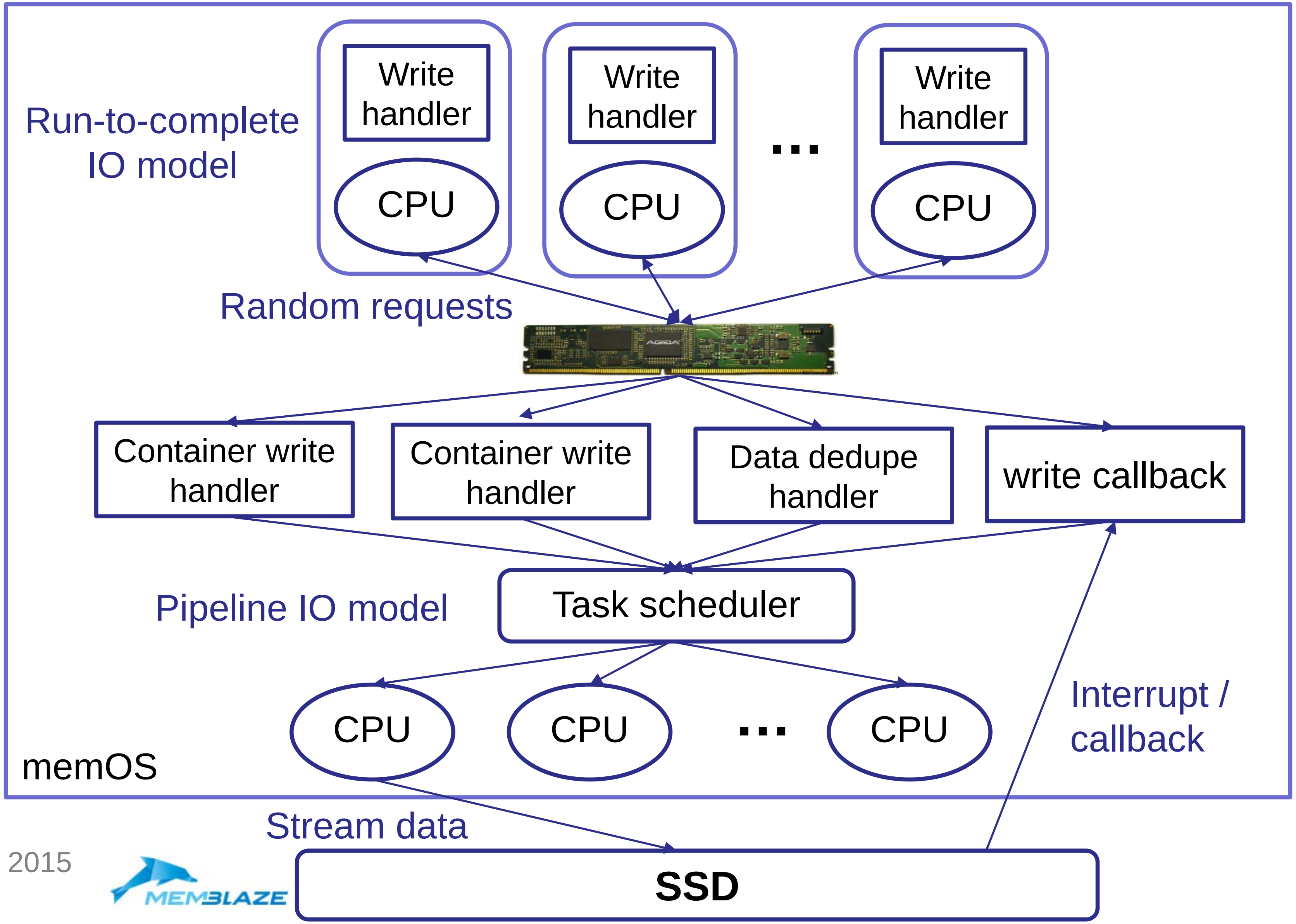
- NVDIMM vs. SSD
  - NVDIMM has higher IOPS and lower latency
    - 10 ~ 100ns latency
  - SSD has higher capacity
    - 10 ~ 100us latency
- Benefits from NVDIMM
  - Avoid updating metadata on SSD frequently
  - Used as write cache to reduce latency for write request
  - Convert all kinds of requests' pattern into sequential
    - Convert IOPS issue into bandwidth
  - Enable to adopt pipeline IO handling model to deal with write request



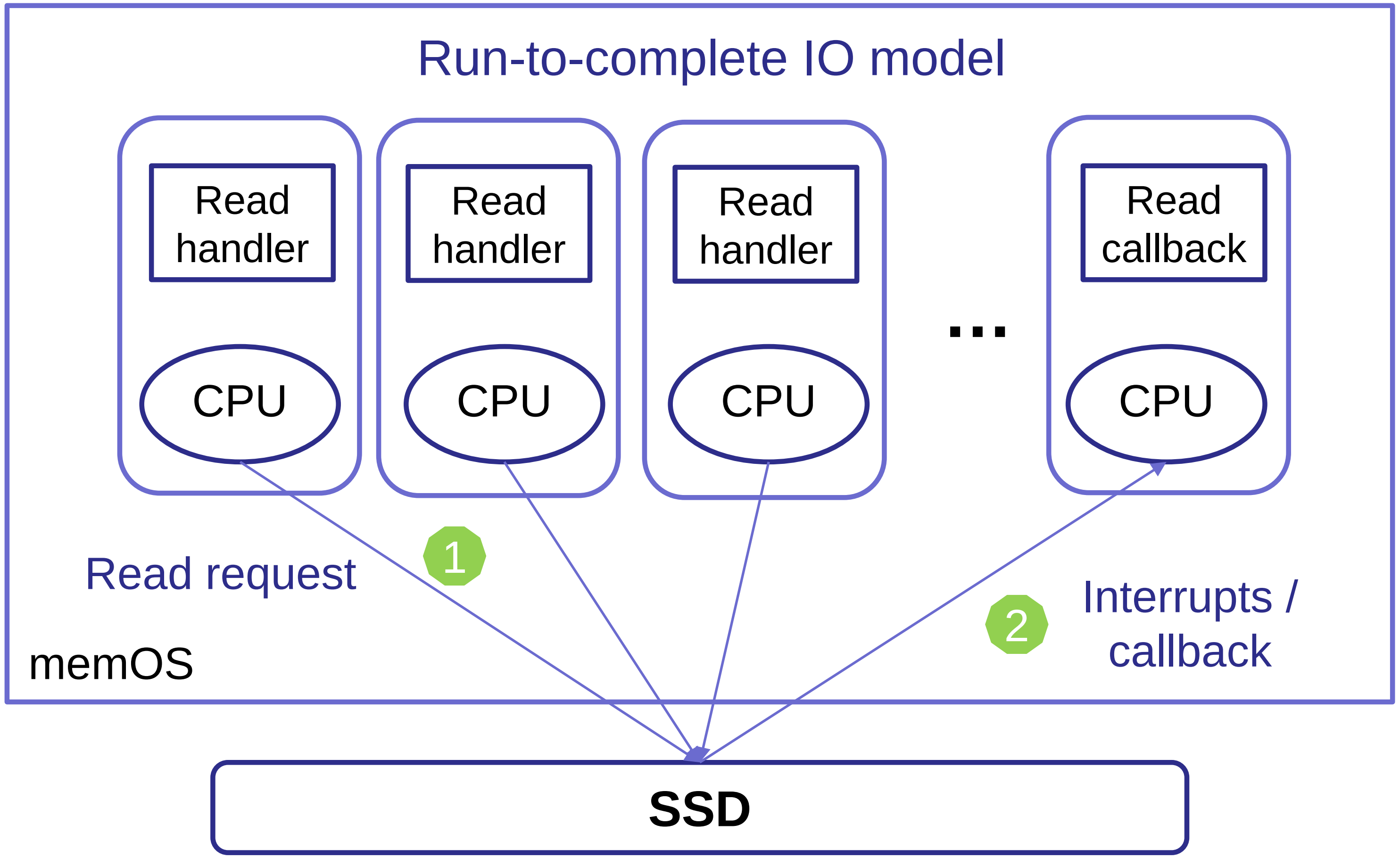
IO Handling Model in RISL

- **Design conflicts: bandwidth & latency**
- **IO handling model**
  - Pipeline
    - Aggregate bandwidth but introduce latency
  - Run-to-complete
    - Reduce latency but affect bandwidth
- **Combine pipeline and run-to-complete**
  - Separate write and read handling processes
  - Write uses both run-to-complete and pipeline model
    - Adopt NVDIMM to reduce latency
  - Read uses run-to-complete model
    - Expand CPU to increase bandwidth

Write Data Path with RISL

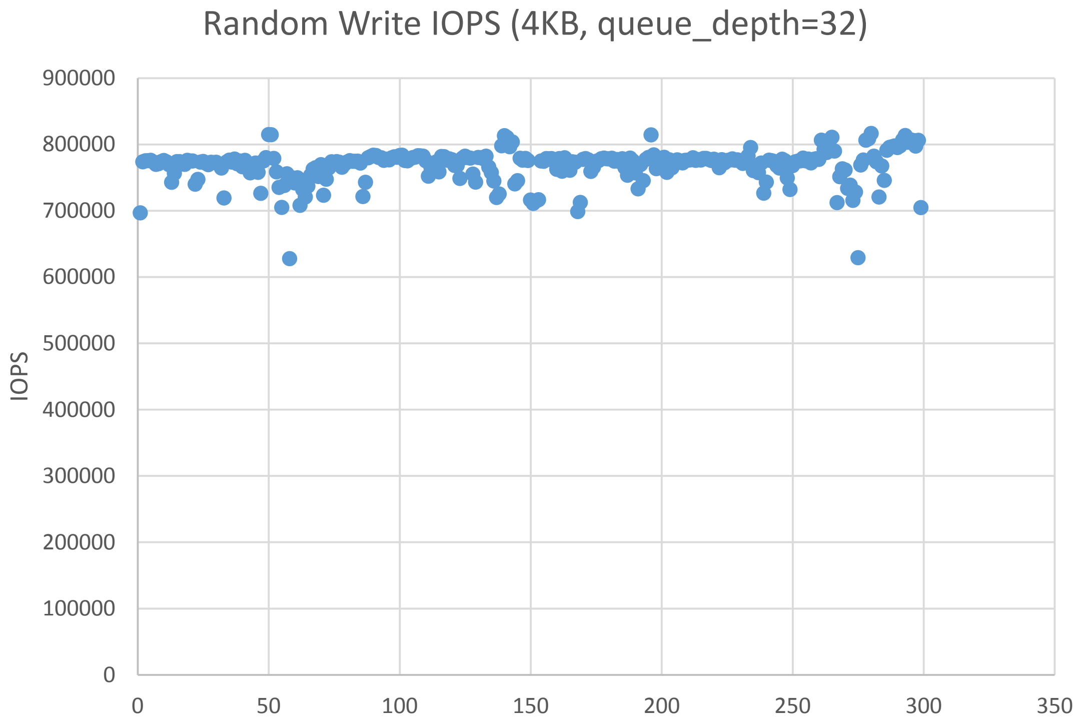


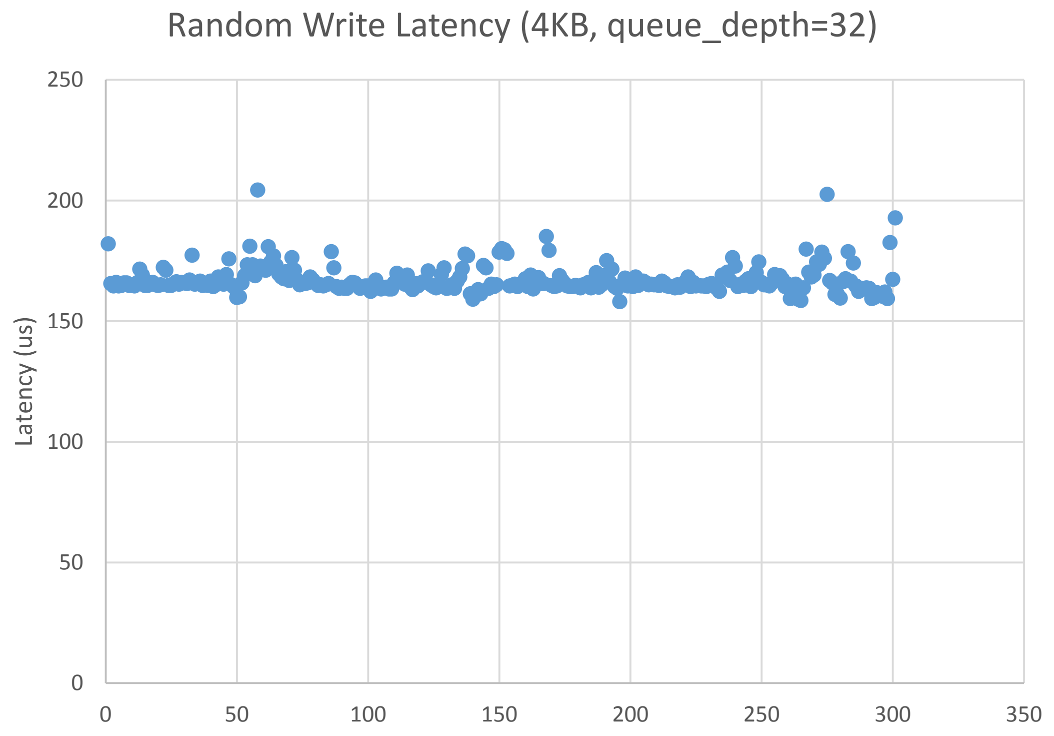
Read Data Patch with RISL



Write Latency Evaluation with RISL

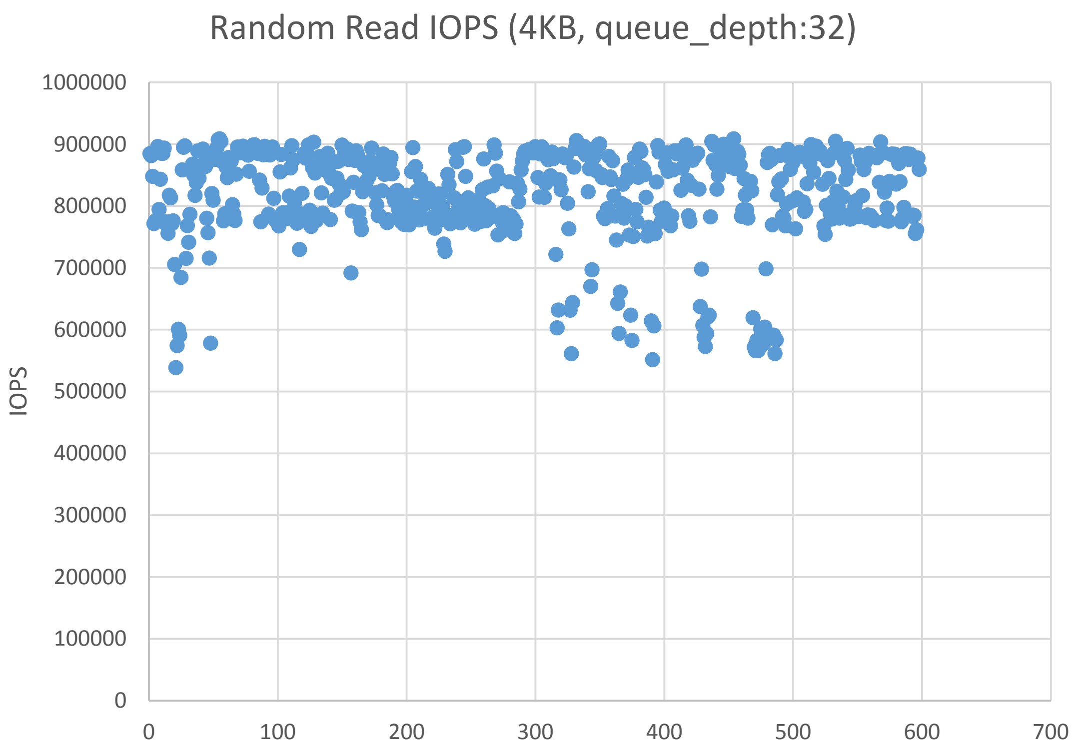
- Write latency is about 160us (8 NVMe SSD, RAID6, 4GB NVDIMM)

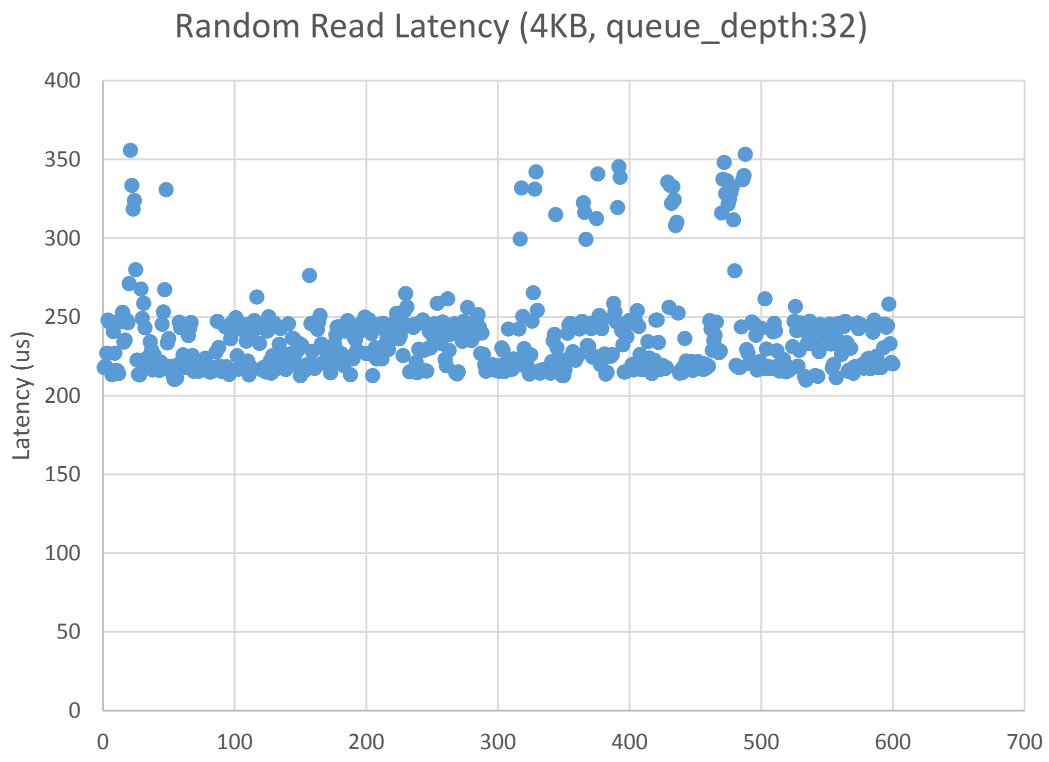




Read Latency Evaluation with RISL

- With 820,000 IOPS, read latency is about 230us (8 NVMe SSDs)



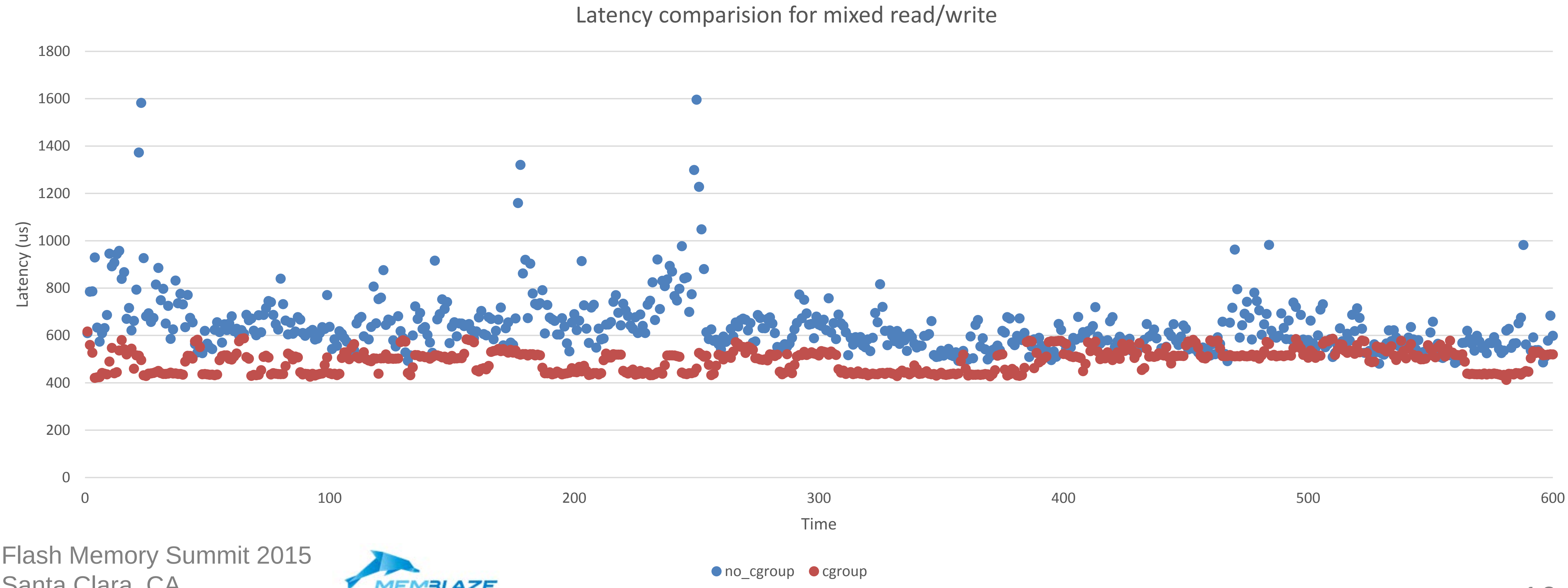


How to Make Consistent Performance?

- Mixed type of IO requests
  - Separate read & write handling threads
  - Write request is dispatched into active containers and read request is distributed on sealed containers
- Linux OS affects performance consistency
  - Linux task scheduler
    - Use cgroup to separate CPU resources
  - Interrupt
    - Interrupt affinity and balance on multi-cores platform

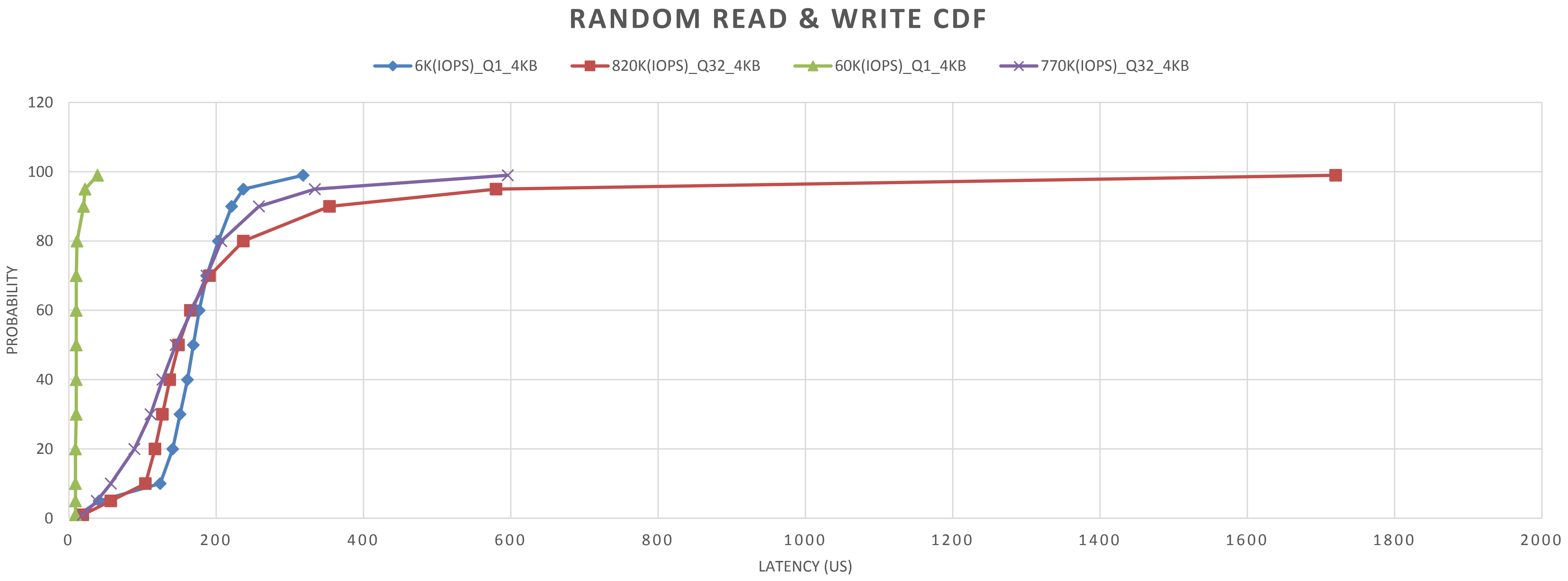
Isolate CPU to make performance consistent

- Cgroup makes performance more consistent



Sustained Latency Evaluation

- Cumulative distribution function (8 NVMe SSDs, RAID6)



Conclusion

- RISL (Random Input Stream Layout) architecture is used to ensure low latency and consistent performance
  - Uses NVDIMM as write cache
  - Separates read & write requests
  - Combines pipeline and run-to-complete IO handling model
  - Converts all kinds of IO pattern into sequential stream on SSD
  - Optimizes data layout on SSD
- Optimize Linux to achieve consistent performance
  - Cgroup / Interrupt affinity / request affinity

Thank You!

http://www.memblaze.com