StorScore system for SSD qualification

Dr. Laura Caulfield
Mark Santaniello

Outline

Introduction

Who we are
Unique needs & opportunities
Motivation

Background

SSD testing "gotchas"

How it works

Recipes: define the test suite
Scores: compare devices and summarize performance
Endurance: measurement of write amplification

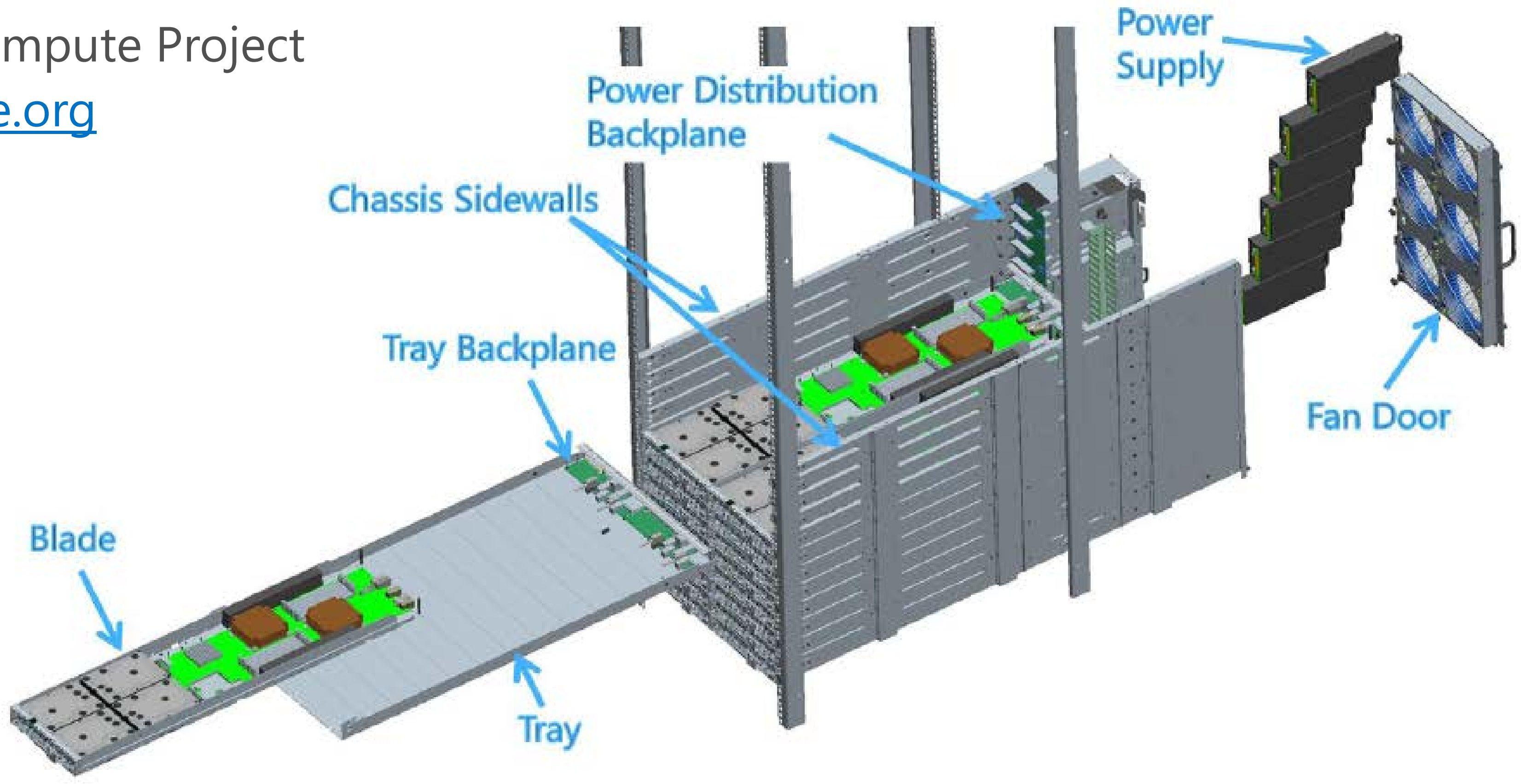
Demo

Introduction

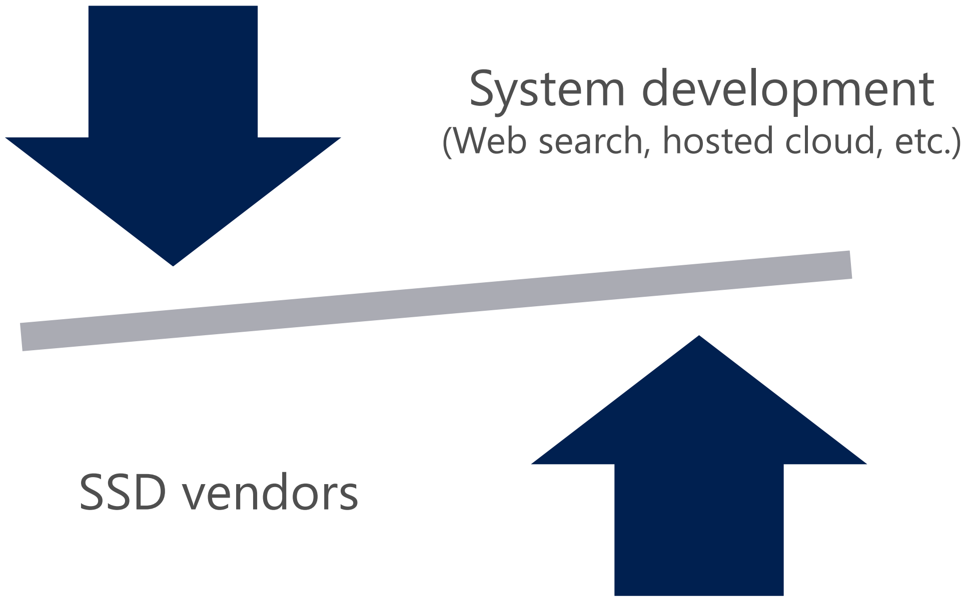
Who we are

Cloud server infrastructure engineering

Designers of Open Cloud Server
Part of the Open Compute Project
http://opencompute.org



Unique needs & opportunities



Microsoft's platform

Variety / quantity of workloads

Flexibility to modify stack

Co-design SSD & apps

Wide variety of expertise

Additional metrics

Motivation

Need for a component-level storage test (AVL/QCL)

Problems with 3rd-party testing services

Want test results under Windows, not Linux

Need results in our server, not a “reference platform”

Desire to share results with component vendors

Existing tools are insufficient (Iometer, etc.)

GUI = difficult to automate properly, therefore error-prone

Microsoft-specific tests (SMART injection)

Cloud metrics (QoS via 5-nines percentile latency)

Methodology is critical when testing SSDs

Background

SSD gotcha – initial performance

NAND must be erased before it can be programmed

When fresh-out-of-box, all NAND is already erased

Drive contains more NAND than the user-visible space

Overprovisioned (OP) space, typically 7% or 28%

This is an unnatural and ephemeral state

Reads short-circuit if block state is erased (never go to media)

Sustained writes will eventually require garbage collection (GC)

StorScore initializes by writing the entire SSD 2x

Every user-exposed LBA is written twice

Virtually guaranteed to cover any OP space

SSD gotcha – history effect

Previous workload affects *current* workload

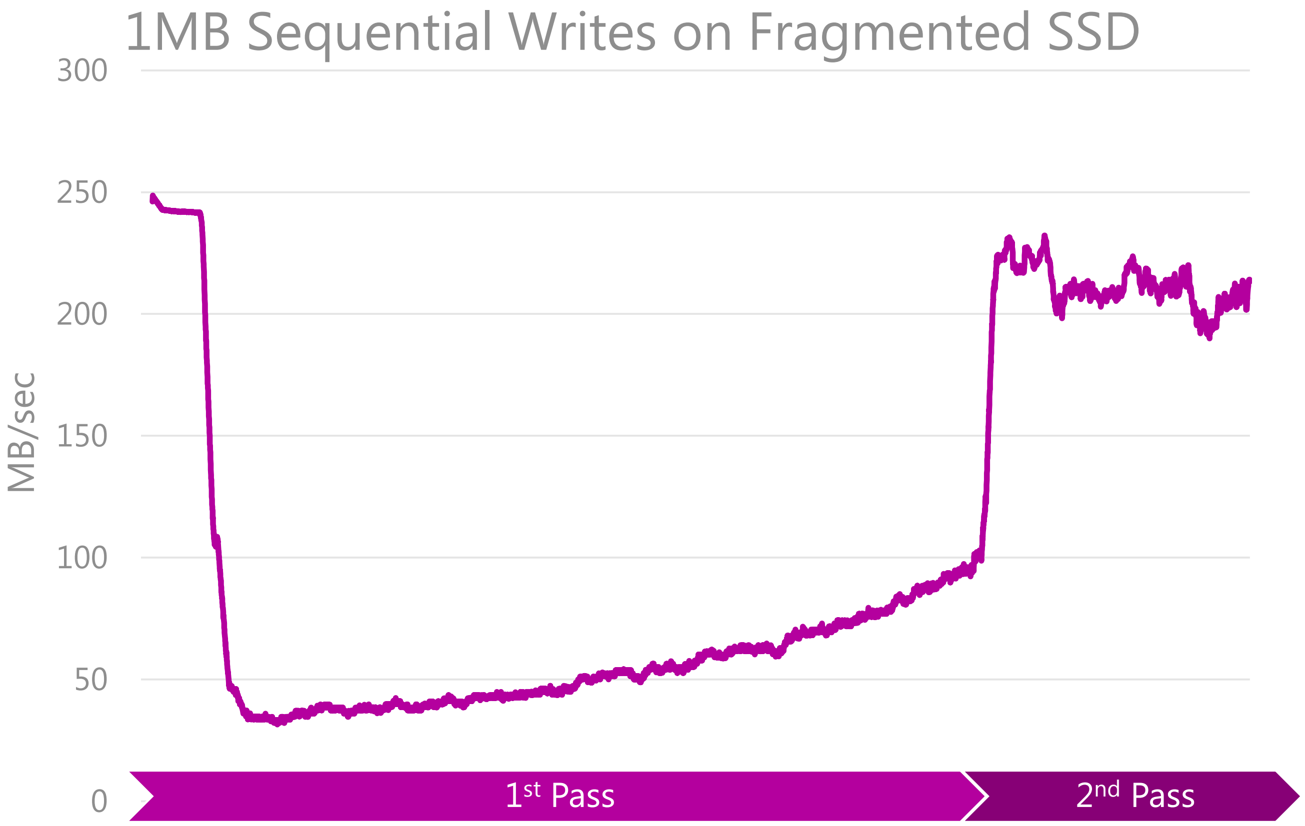
Transition phase can take hours

Main causes

FTL map fragmentation (ex: random → sequential)

Concurrent garbage collection activity (ex: 100% write → 0% write)

SSD gotcha – history effect, continued



Consistent workload will eventually reach steady-state

1st Pass, 3.5 hours

30 – 100 MB/sec

2nd Pass, 30 minutes

~220 MB/sec

SSD gotcha – detecting steady-state

StorScore includes precondition.exe

Always drives to steady-state *before* measuring performance

Method: rolling linear regression, detect near-zero slope

StorScore performs all sequential tests before any random tests

Minimizes fragmentation and therefore time required to achieve steady-state

SSD gotcha – entropy of written data

Some controllers can compress on-the-fly

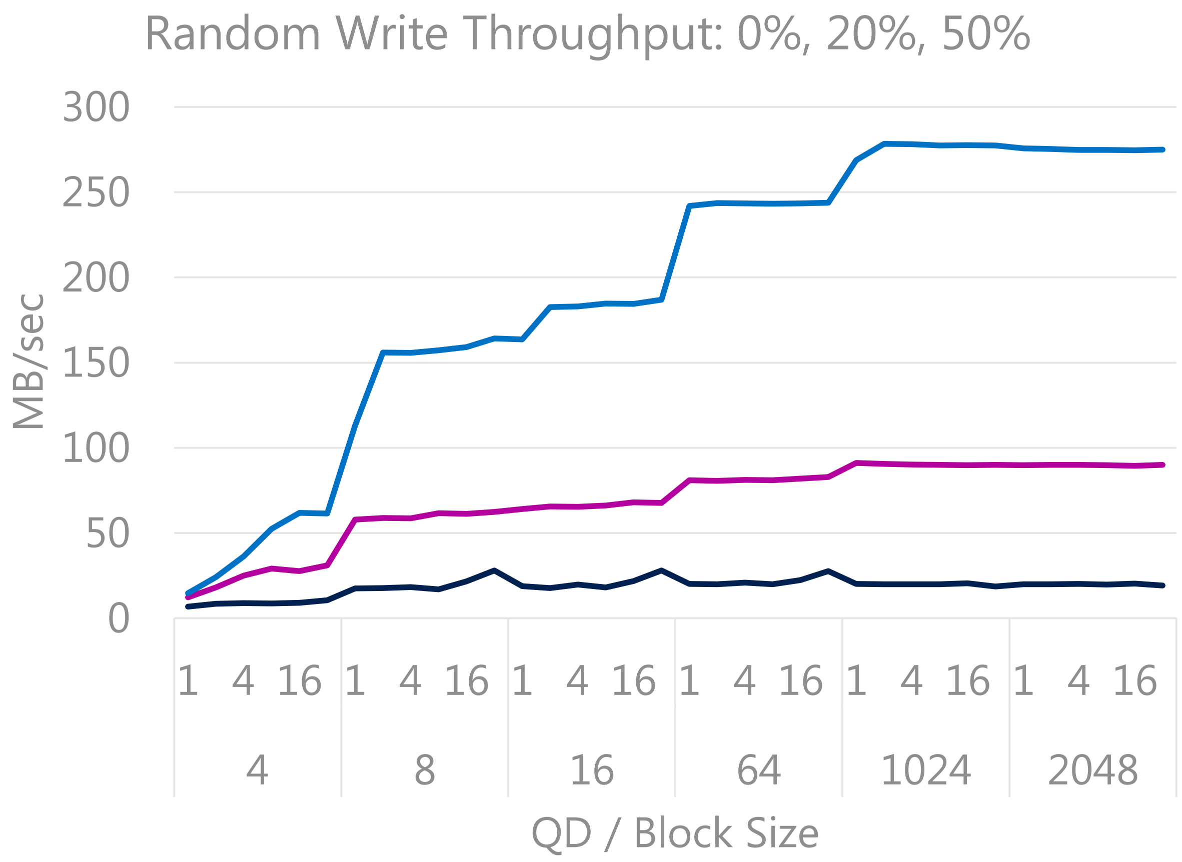
Customer data is often compressible

Entropy has a big impact on performance and endurance

StorScore supports variable compressibility

Uses incompressible data by default

Allows use of compressible data in 10% increments



How it works

Recipes – a single test

```
test(
    name_string         => 'foo',
    write_percentage    => 0,
    access_pattern      => 'random',
    block_size          => '8K',
    queue_depth         => 32,
    warmup_time         => 60,
    run_time            => 3600
);
```

The entire contents of single.rcp

Reference the file from the cmd line:

C:\>StorScore --recipe=single.rcp

Reads like English

Recipes – a matrix of tests

```
# vim: set filetype=perl:
require 'matrix.rpm';

do_matrix(
    access_patterns     => [qw( sequential random )],
    write_percentages   => [qw( 100 30 0 )],
    block_sizes         => [qw( 2M 1M 512K 64K 16K 8K 4K 1K )],
    queue_depths        => [qw( 256 64 16 4 1 )],
    warmup_time         => 60,
    run_time            => 3600
);

include 'targeted_tests.rcp';
```

Mimics test designer's whiteboard sketch

Include statements combine test files

Full functionality of Perl

```
do_workload( "Targeted Test Read Baseline" );

bg_exec( "smart_loop.cmd $gc{'target_physicaldrive'}" );
do_workload( "Targeted Test SMART Read Data " );
bg_killall();
```

Results parser

Raw output files
→ one Excel file

24 SSDs
x 218 workloads
5,232 files

Display Name	Write Mix	Access Size (kB)	Access Type	Queue Depth	Bandwidth (MB/s)	Average Latency (ms)
Device A	100%	16	random	1	54.32	1.04
Device B	100%	16	random	1	15.05	0.29
Device A	30%	16	random	1	20.01	1.39

Detects, highlights outliers

Easy pivot tables & graphs

Example policy:
Bandwidth matters a lot, latency matters a little

Device A scores 72/100
Device B scores 65/100

Still too much data

5,232 files x 23 metrics = 120k data points

Putting the “score” in StorScore

Goal

Enable data-driven decisions throughout the company

Reduce data to 1 score / drive

Method

A weighted average of all the metrics for each workload

Step 1:
Convert each value to z-score

Display Name	Write Mix	Access Size (kB)	Access Type	Queue Depth	Bandwidth (MB/s)	Average Latency (ms)
Device A	100%	16	random	1	Z_AX0	Z_AX1
Device B	100%	16	random	1	Z_BX0	Z_BX1
Device A	30%	16	random	1	Z_AY0	Z_AY1

Calculating z-scores

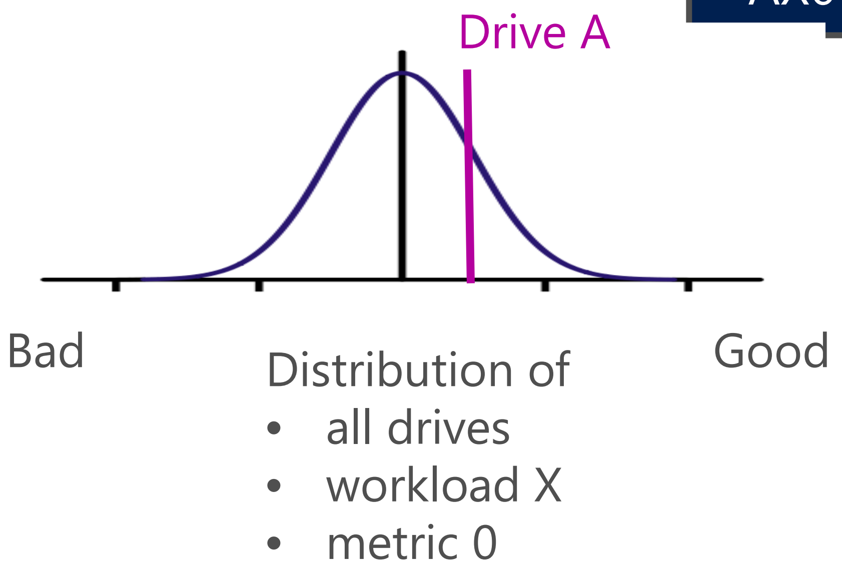
A z-score (or standard score) is the number of standard deviations from the mean

Drive: A

Workload: X (4k, rand, QD1, 100% writes)

Metric: 0 (Read Latency)

z_{AX0}



One z-score per data point

Positive = better than average

Negative = worse than average

Based on cohort of drives

Applying scoring policies

General Policy:

Throughput Metrics

Latency Metrics

50% x $\boxed{Z_{A(n+m)i}}$ + **50% x** $\boxed{Z_{A(n+m)j}}$ = **70 / 100**

Drive A
Wkld range 0 to (n+m)
Metric range 0 to i

Score for Drive A

- Can apply multiple policies at once
- Can use any kind of weight system (stay consistent within policy)

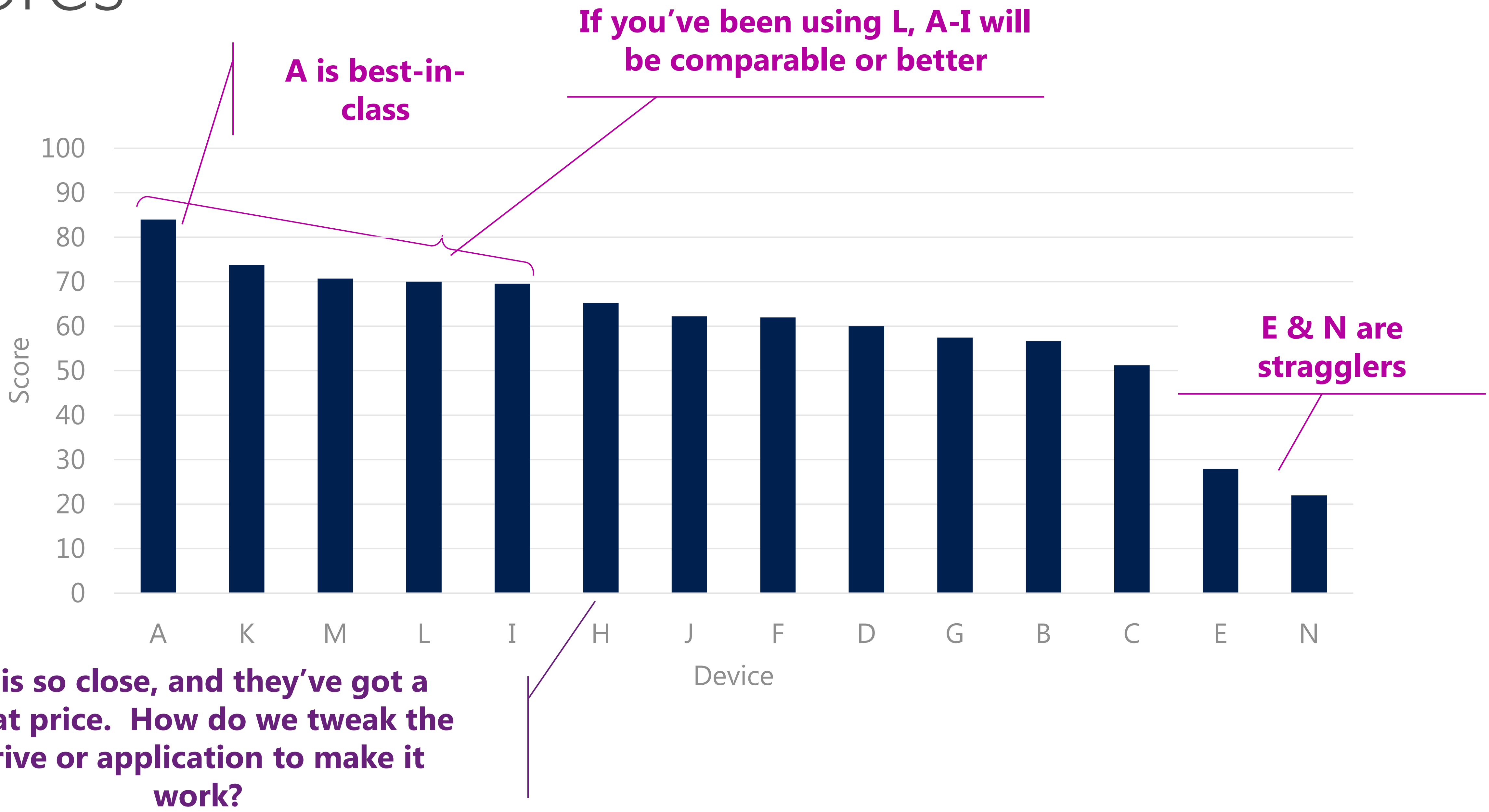
Policy to Favor Mixed Workloads:

70/30 Read/Write Mix Workloads

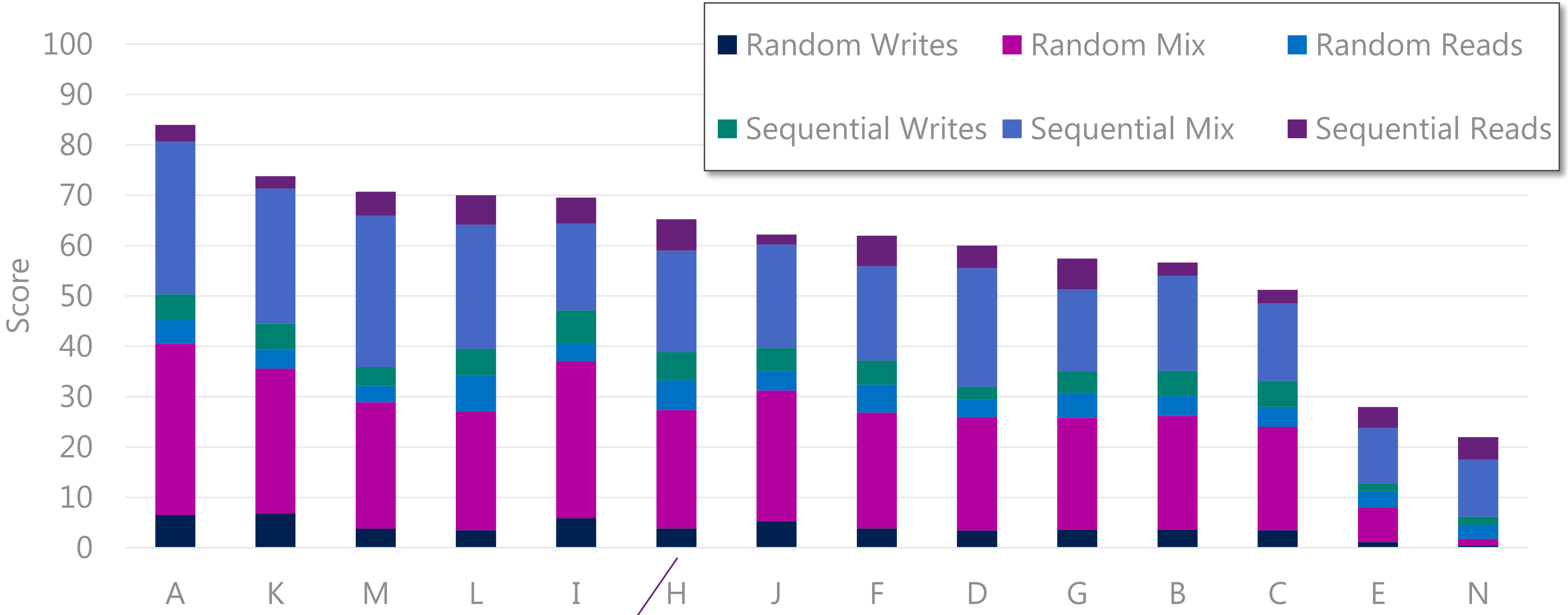
100% Read & 100% Write Workloads

5 x $\boxed{Z_{An(i+j)}}$ + **1 x** $\boxed{Z_{Am(i+j)}}$ = **65 / 100**

Scores


A is best-in-class
If you've been using L, A-I will be comparable or better
E & N are stragglers
H is so close, and they've got a great price. How do we tweak the drive or application to make it work?
Score
100
90
80
70
60
50
40
30
20
10
0
A
K
M
L
I
H
J
F
D
G
B
C
E
N
Device

Scores' breakdown



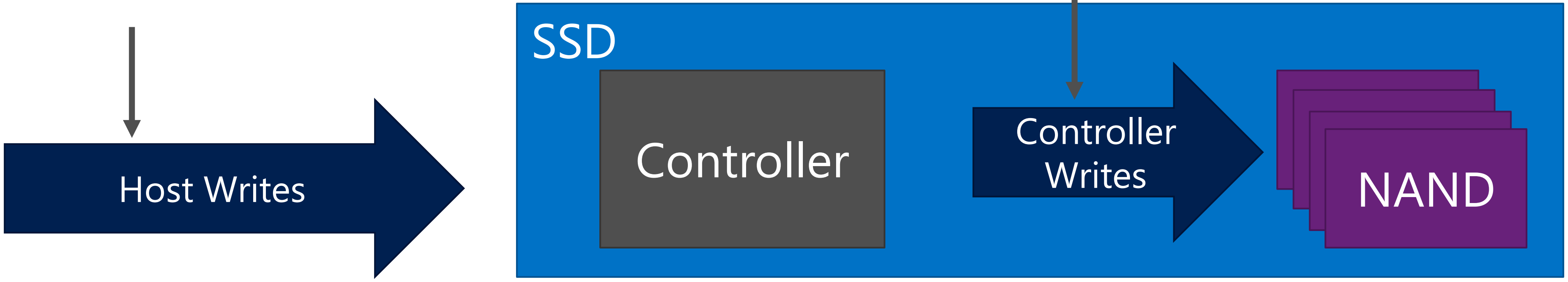
H is so close, and they've got a great price. How do we tweak the drive or application to make it work?

Answer:
Drive should improve random mix (not seq. mix), or
App should favor sequential mix (not random mix)

SSD failure mechanism: writes

Drive Writes Per Day (DWPD)
Total Bytes Writes (TBW)
Drive Writes (DW)

Program / Erase Cycles (P/E Cycles, or PEC)
Write / Erase Cycles (W/E Cycles)

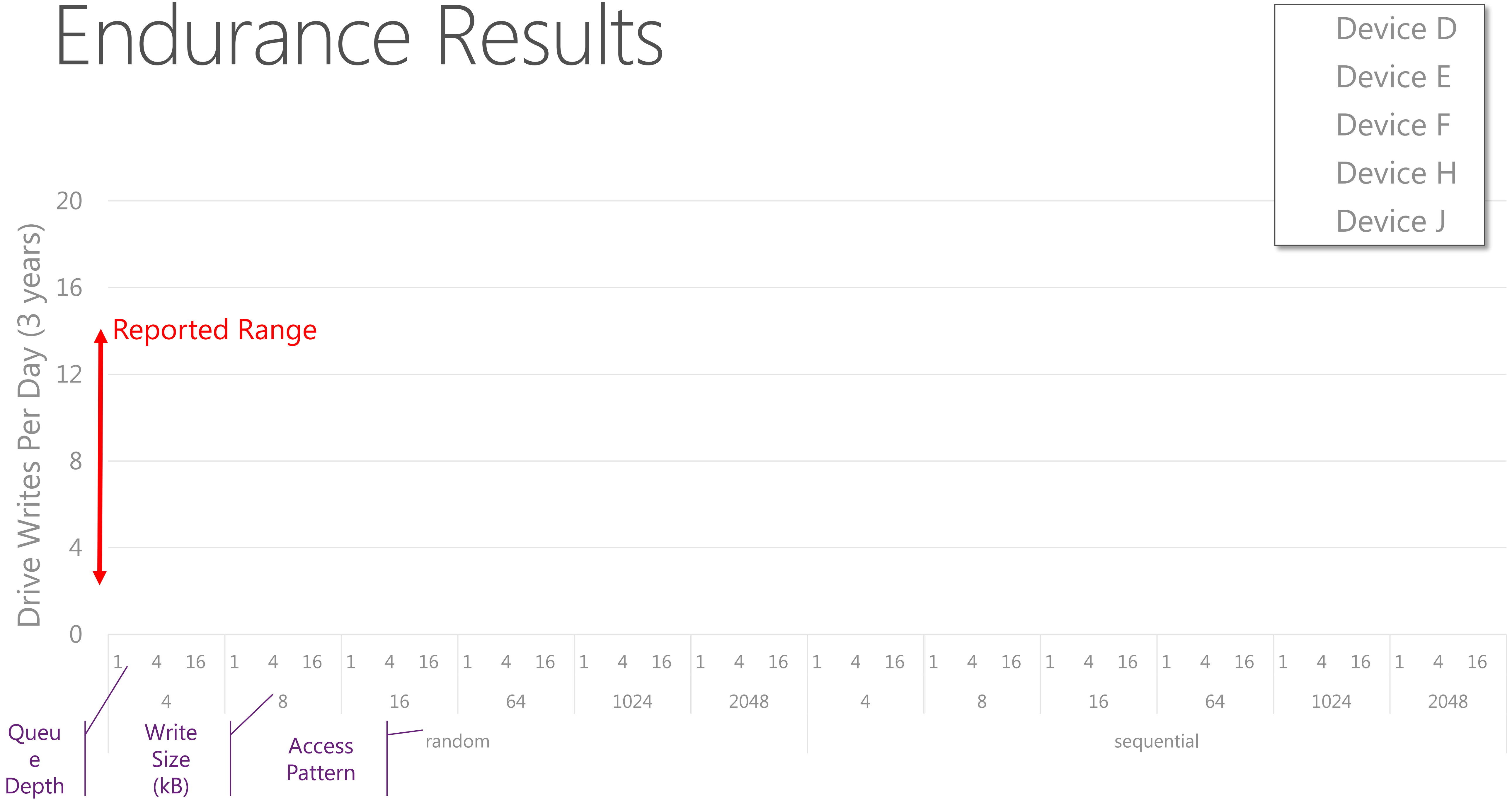


Total Drive Writes **x Write Amplification Factor= P/E Cycles**

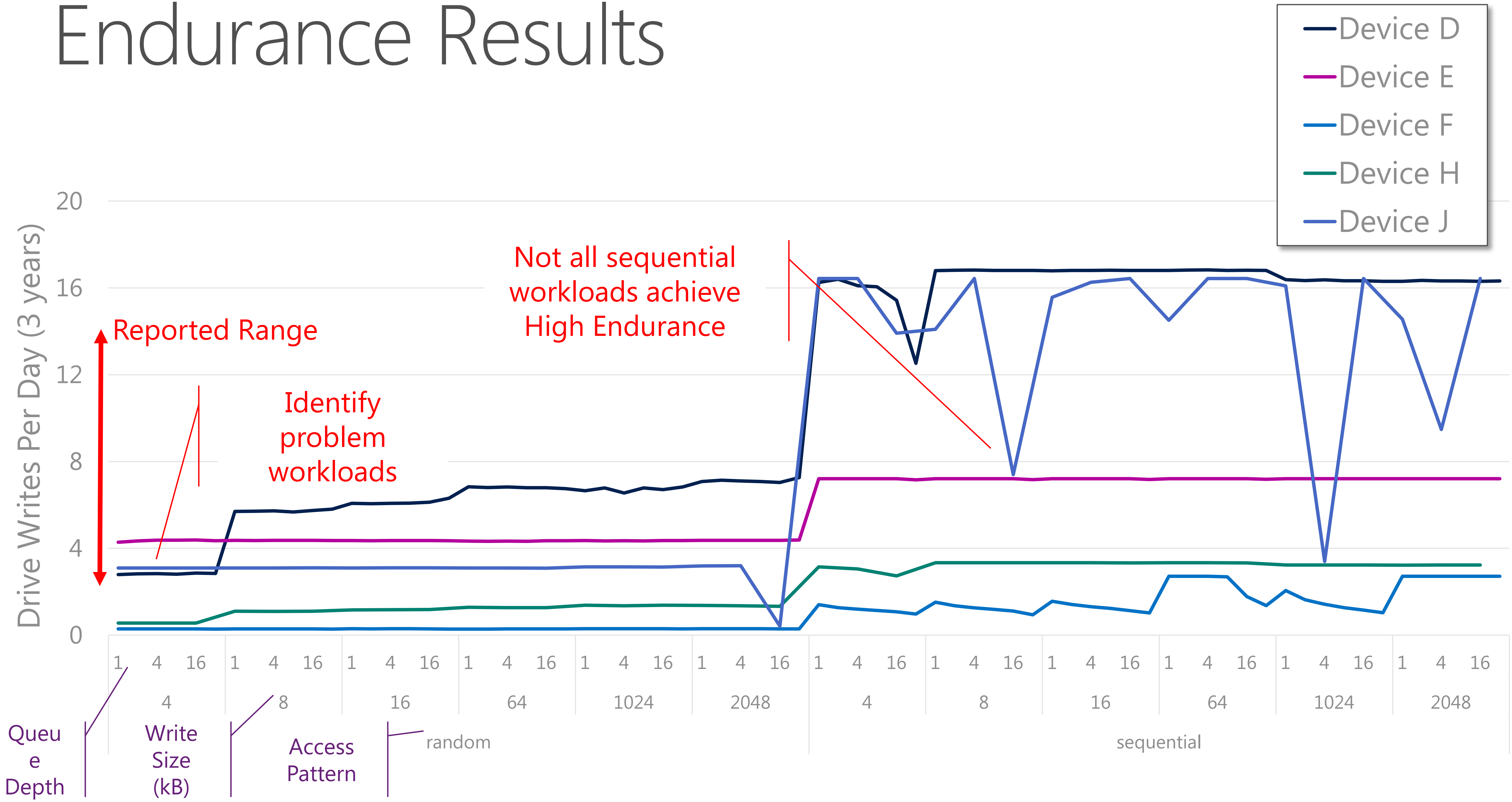
Workload Dependent,
Vendor Reported, Implementation Specific

Previously Available	**New Telemetry**
SMART "Media Wear Indicator"	SMART "Controller Writes"
Reported in **units of 1%** (300 TB for 30k, 1TB drive)	Reported in units of **sectors or GB**
4.7 months for 1 workload	1,700 workloads in 4.7 months

Endurance Results



Endurance Results



Demo

Thanks!

- Download StorScore
  - http://aka.ms/storscore
- Questions?