



# Introduction to "ivy"

Block storage synthetic workload generator with real time dynamic feedback control and workflow engine.

<https://github.com/Hitachi-Data-Systems/ivy>

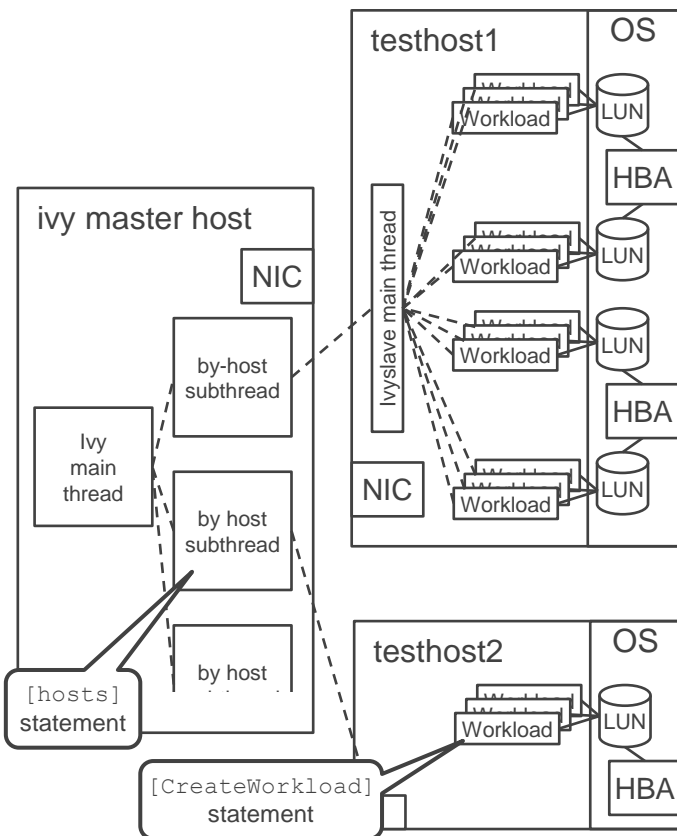
**Hitachi Data Systems** - 2016-08-13

Allart Ian Vogelesang [ian.vogelesang@hds.com](mailto:ian.vogelesang@hds.com)

- To automate an entire test suite workflow, performing tests in the theoretically minimum time possible
  - To reduce cost
  - To speed time-to-market
  - To improve quality
  - To automatically calibrate sales sizing tools
- To simplify manual test setup
  - Just say you want to test selecting `pool_ID = { 0, 2 }`, and let ivy figure out which LUNs on which hosts.
  - Easily verify actual test configuration – say `[CreateRollup] "port" [quantity] 64 [MaxDroop] 20%`; and ivy will automatically validate that there are 64 ports reporting and that none of them is reporting an IOPS more than 20% below the IOPS of the fastest port.
- To offer Dynamic Feedback Control measurements like "find the IOPS to obtain 1.0 ms service time"
- To secure customer trust in measurement results made using open source software

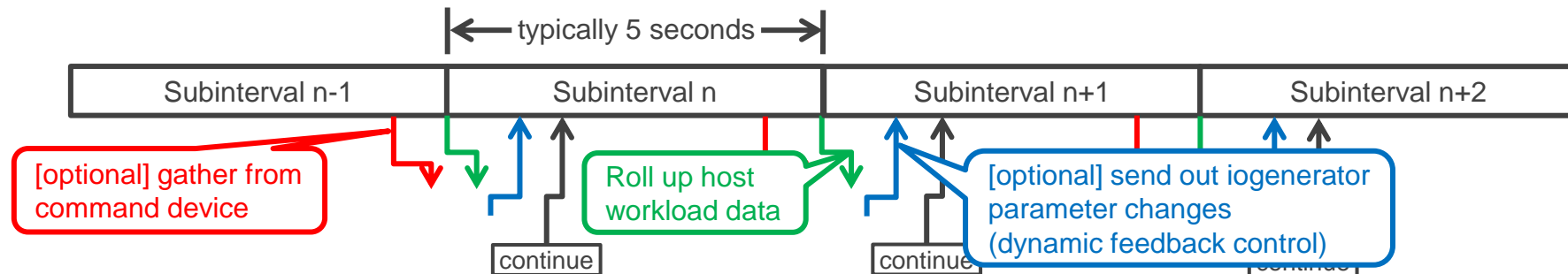
- Valid results are *repeatable*
  - If you run the test again, within the specified +/- experimental error, you will get the same result again.
  - Valid, repeatable, results are for steady-state conditions
- If the workload / subsystem are not steady-state, you can't make a valid measurement.
  - After imposing a workload on the subsystem
    1. we need to wait for the behaviour to settle down into a steady state, waiting for initial transient conditions to settle down, then
    2. depending on how much "noise" there is in what is being measured, measure for long enough to obtain a valid measurement to a specified plus/minus accuracy.

# Scalable: master host => test host => LUN => workload



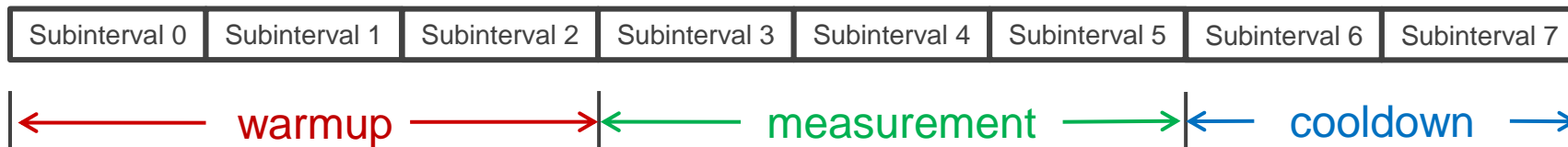
- In ivy, you can layer multiple named “workload” I/O generator threads on LUNs on test hosts.
- Each workload thread uses an I/O sequencer plug-in that generates I/Os in scheduled I/O start time sequence.
  - We try to keep a few I/Os “pre-computed” and ready to go at all times.
  - Generalized interface allows use of any I/O pattern generator, e.g. synthetic or in future, playback.
- Use of C++ with Linux kernel Asynchronous I/O interface means each workload thread can drive any queue depth up to limits of the OS / HBA with state of the art minimal CPU overhead.

# An ivy test is a sequence of "subintervals"



- The default is `subinterval_seconds = 5`
- If a command device connector is being used, the performance data for the subinterval arrive at ivymaster and are posted using configuration filters by rollup instance.
- At end of subinterval, workload thread measurement data are sent to ivymaster to be rolled up.
- If dynamic feedback control (dfc=PID) is being used, DFC parameter updates at the granularity of the rollup instance are sent out
- ivymaster decides to continue the test for another subinterval, or to stop.

# A "successful" test step (subinterval sequence) – 3 phases



- During the **warmup** phase, we are either allowing time for, or expressly waiting for, initial transient conditions to settle down.
  - `warmup_seconds` specifies the minimum warmup period, which the `measure` feature may extend
- When the test step is declared a "success", the results of the test step are defined as the average/totals over the **measurement** phase.
  - `measurement_seconds` specifies the minimum measurement period, which the `measure` feature may extend
  - A test step may be declared a "failure" by the `measure` feature upon reaching the `timeout_seconds` value without observing a valid measurement. In this case, all subintervals are posted as "warmup" subintervals.
- During the **cooldown** phase, we are running at least one extra subinterval continuing to drive the workload, so that even with some time jitter across test hosts, the full workload will have been running throughout the measurement.
  - If a "command device connector" is being used, `cooldown_by_wp=on` controls whether to subsequently keep running even more subintervals with IOPS=0 until we see that Write Pending in the subsystem has emptied.

- `accuracy_plus_minus = 5%`
  - Default is 5%.
  - Extend the warmup as necessary until initial transients die down and extend the measurement period as necessary until we have "seen enough" to be sufficiently confident we have made a measurement to the specified accuracy.
- `confidence = 95%`
  - How confident you need to be that your measurement falls within the specified plus or minus range around the long term average that you would get measuring forever.
  - Default is 95%, meaning measurement falls within `accuracy_plus_minus` 19 times out of 20.
  - Ivy has a menu of 11 specific pre-loaded confidence values that you pick from.
    - 50%, 60%, 70%, 80%, 90%, 95%, 98%, 99%, 99.5%, 99.8%, and 99.9%

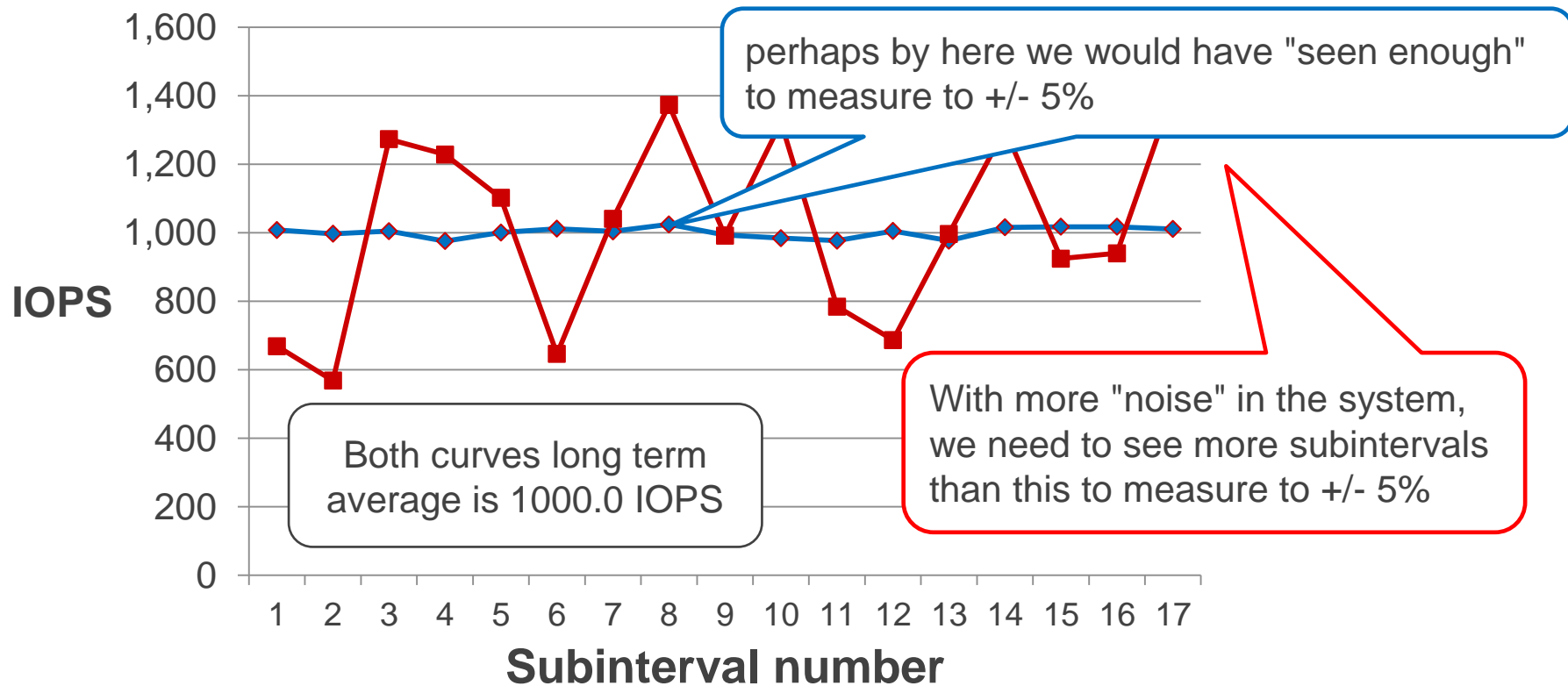
- It is both convenient and effective to use the standard statistical formulas using the `accuracy_plus_minus` and confidence parameters
  - [http://en.wikipedia.org/wiki/Student%27s\\_t-distribution](http://en.wikipedia.org/wiki/Student%27s_t-distribution) (for the math literate)
- Limitation – the standard formulas don't exactly apply to ivy
  - The formulas are for samples drawn at random from a large population.
  - The behaviour / measurement for one subinterval is related to those for preceding / succeeding subintervals, thus "gaming the system".
- Rule of thumb to correct for this
  - Specify `accuracy_plus_minus` with a 2x safety factor, and you'll be OK.
    - Say `accuracy_plus_minus` = 5% to get repeatability within 10%.



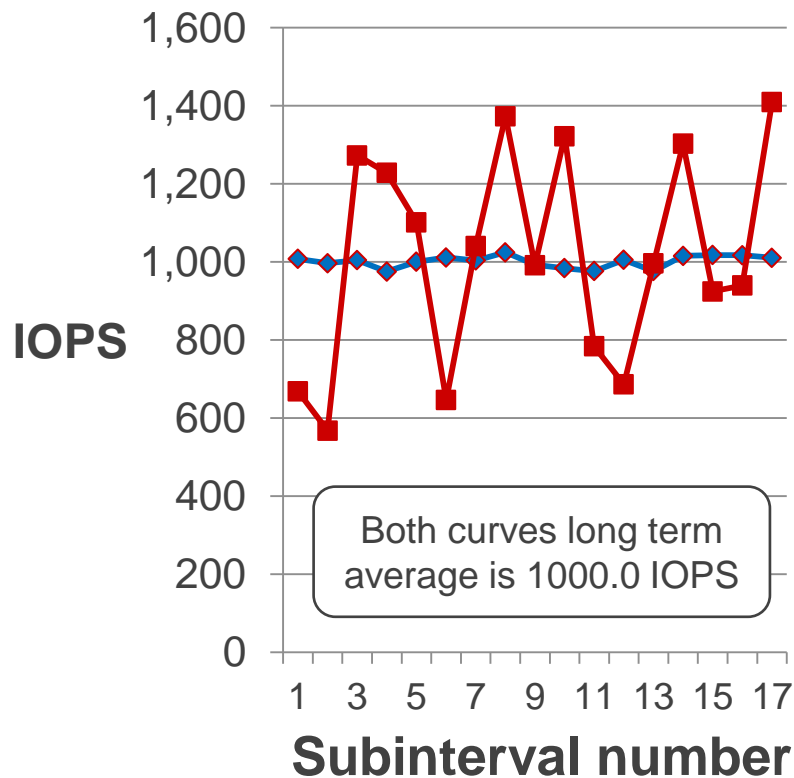
# measure parameter "shorthand" presets

- `measure = IOPS`  
`measure = MB_per_second`  
`measure = service_time_seconds`  
`measure = response_time_seconds`
- These are "shorthand" for a particular combination of more detailed parameter settings
  - See "ivy programmer's reference, part two" for a description of the full set of parameter settings.

measure=IOPS, accuracy\_plus\_minus=5%

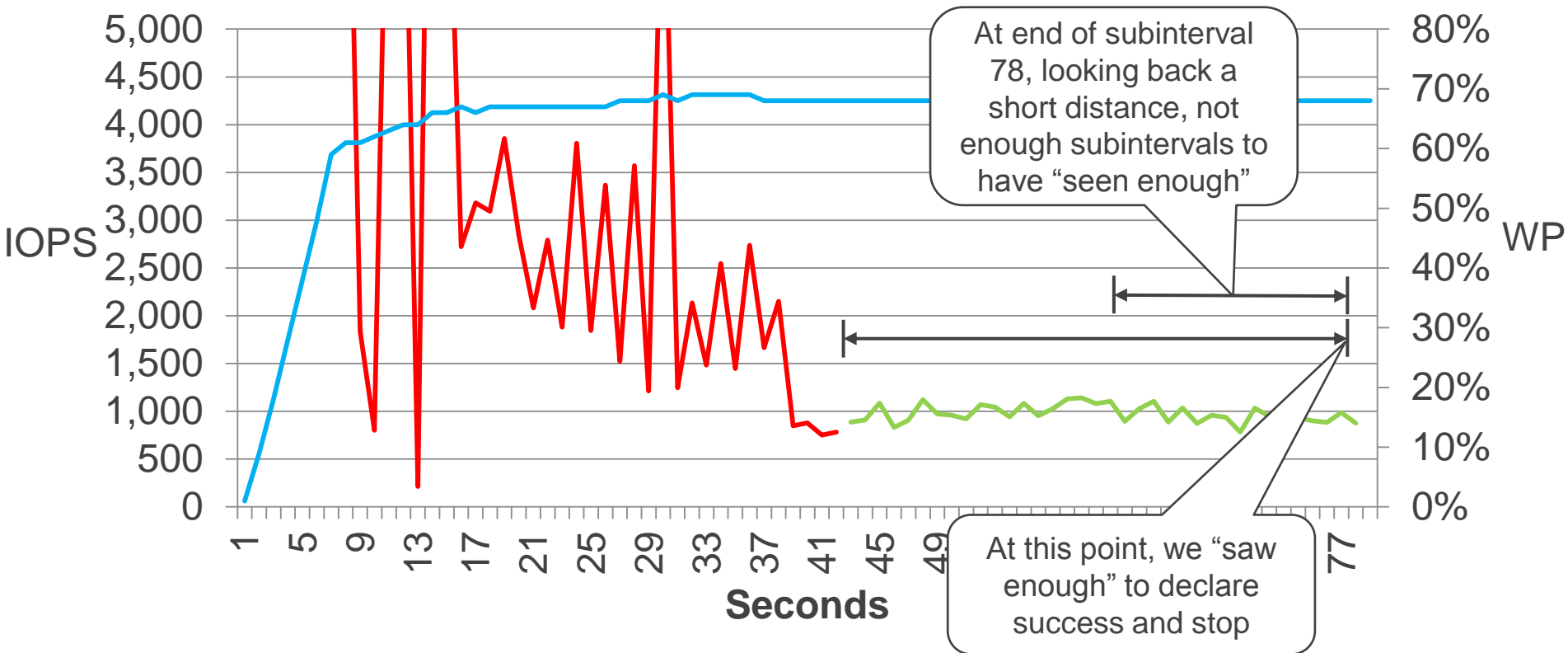


measure=IOPS, accuracy\_plus\_minus=5%



- The ivy "measure" feature extends the test until enough measurements have been made to be sufficiently statistically confident that the average of the by-subinterval values is within a specified accuracy\_plus\_minus variation from what the long term average would be continuing to test indefinitely.

# measure effective to automatically reject initial transients



- Instead of running for a fixed 6 minutes for each step of the several hundred combinations / permutations of a standard scalability test sequence, when a variable step run time using `measure` was used
  - Overall test run time decreased by 60%
    - The vast majority of the time workloads settled down quickly and were then stable.
  - Quality was improved
    - In some rare cases, there were test steps that needed to run for 20 minutes to get an accurate measurement.
- *The use of the `measure` feature in ivy substantially reduces test time, while at the same time improving quality.*

1. An external vendor proprietary simple SCSI Inquiry-based LUN discovery tool is used.
  - The external tool provides a "discovered LUNs" csv file that has a header line with column titles
    - `host,LUN name,HDS product,port,LDEV,PG,CLPR, ...`  
`testy1,/dev/sdxy,VSP,1A,00:00,1-1,CLPR0, ...`  
`testy1,/dev/sdyz,VSP,2A,00:01,1-2,CLPR0, ...`
  - To support another vendor's architecture or terminology, all you need is the external LUN discovery tool that produces such a csv file.
  - Hitachi LUN discovery tool is open source - [https://github.com/Hitachi-Data-Systems/LUN\\_discovery](https://github.com/Hitachi-Data-Systems/LUN_discovery)
2. In ivy, the LUN discovery csv file header line column titles become selectable
  - `[select] "port" is { "1A", "3A", "5A", "7A"}, "CLPR" is "CLPR0"`

## "command device connector" is proprietary extension to ivy

- Available only for authorized use in Hitachi labs, with license key mechanism.
- Transparently / automatically connects to a command device if one is available.
- Retrieves subsystem configuration data
  - Enables selection of test configuration by subsystem configuration attributes such as `drive_type`.
- Retrieves real time subsystem performance data, aligned with test subintervals.
  - Records what is happening inside the subsystem correlated with what the hosts are seeing.
  - Enables real-time dynamic feedback control of host workload based on subsystem data
    - *"measure IOPS at 50% owning MP core % busy", or "measure IOPS at 50% parity group percent busy"*
- Checks that subsystem does not have any failed components.

- Vendors are encouraged to prepare their own proprietary "subsystem connector" interface tools to be used with ivy.
  - Collect subsystem configuration data to augment SCSI Inquiry attribute data
  - Collect real time subsystem performance data, synchronized and aligned with test host workload data.
    - Facilitates development of modeling tools.
  - Use dynamic feedback control to find the IOPS to reach a target value for
    - subsystem MP % busy, or
    - drive % busy, or
    - cache dirty data % full, etc.

Output from demo runs includes examples of output using a Hitachi command device connector.

This is to show you that with ivy, you too can do dynamic feedback control on your own real-time metrics if you write your own ivy connector.



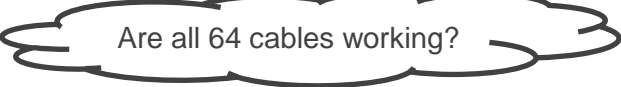
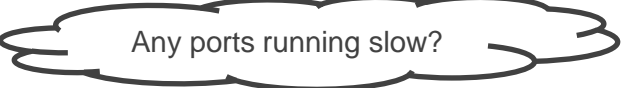
# [CreateRollup] "port";

- Every workload thread is represented in exactly one instance of each rollup.
- There is by default always a rollup called "all" which has one rollup instance, "all=all" which consists of all workloads on all LUNs on all hosts.
- A user-defined rollup name must be combination of LUN attribute names.
  - If you have multiple subsystems under test, to get data by subsystem by port say

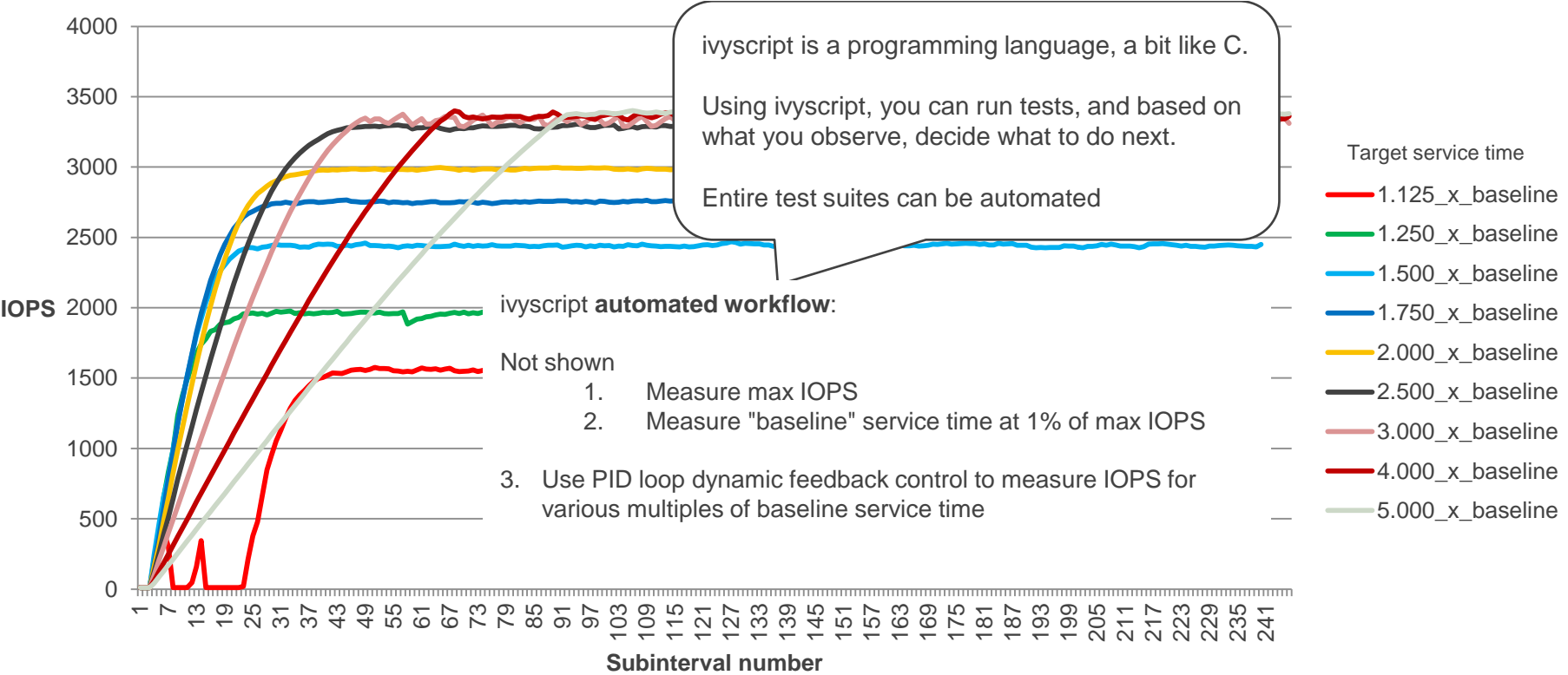
```
[CreateRollup] "serial_number+port";
```

and get instances like "serial\_number+port=410034+1A".

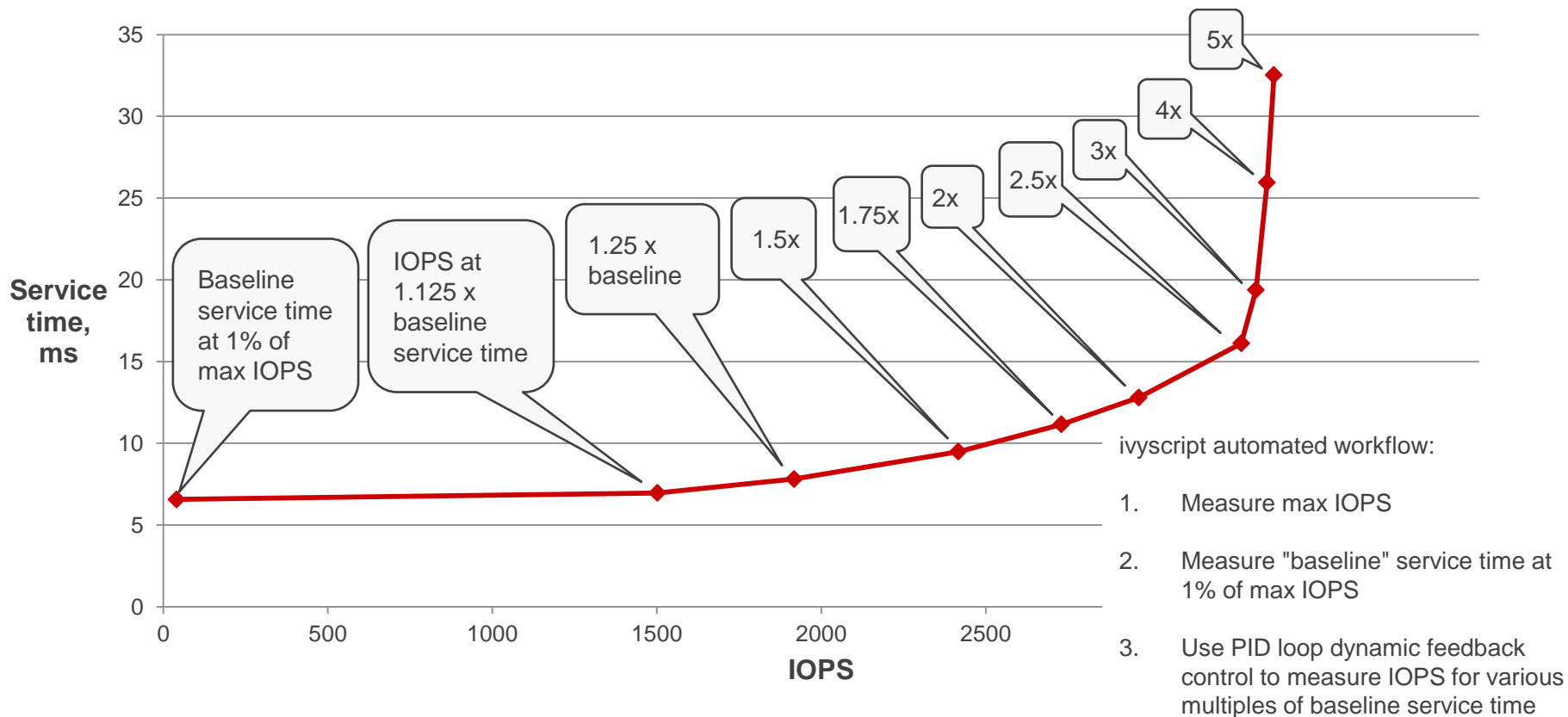
# "Everything" in ivy works using rollups

- You get csv files by rollup instance
  - Say [nocsv] to suppress.
- Both the "measure" functionality, and the dynamic feedback control functionality (dfc=PID) operate at the granularity of the rollup instance (fine grained)
  - To do this on subsystem data, with a command device connector, a "configuration filter" is built in advance for each rollup instance showing which underlying hardware instances to filter on for that rollup instance, e.g. port={1A, 3A}, MPU=03, ... This includes indirectly associated hardware inferred by examining subsystem configuration data.
- [CreateRollup] "port" [quantity] 64; •••  Are all 64 cables working?
  - Will invalidate a measurement if we don't get data for exactly 64 subsystem ports.
- [CreateRollup] "port" [MaxDroop] 20%; •••  Any ports running slow?
  - Invalidates if there is a port whose IOPS is more than 20% slower than the fastest port.

# industry standard "PID Loop" controller

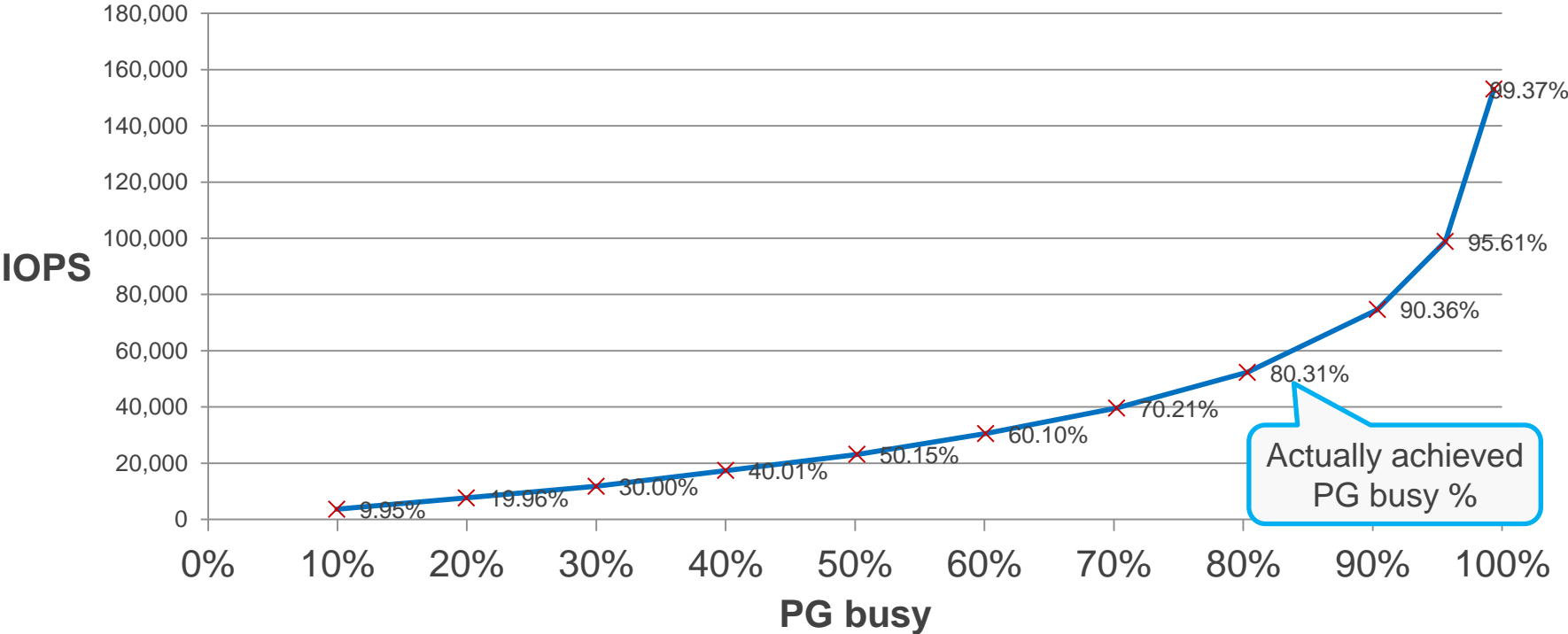


# Workflow plots auto-scaling IOPS vs. service time curve



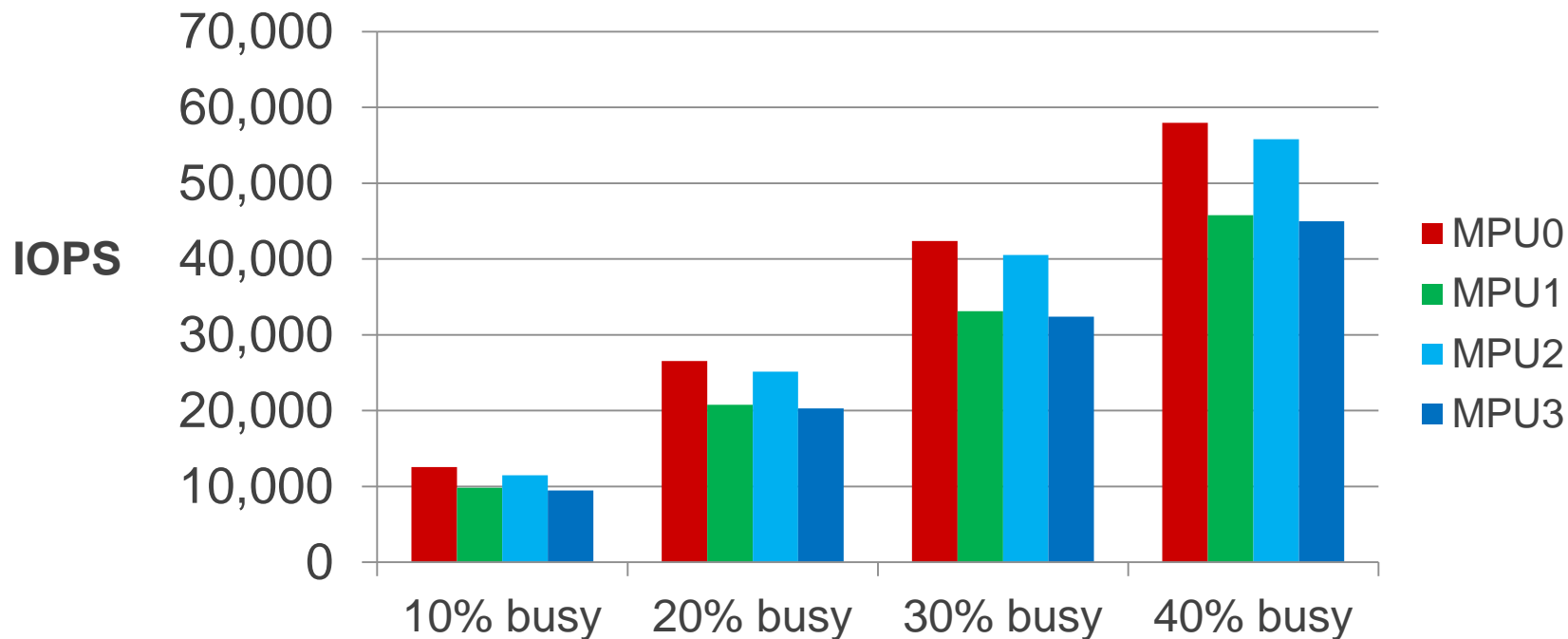
# DFC effective on command device connector data

For parity group busy % = 10%, 20%, 30%, ...



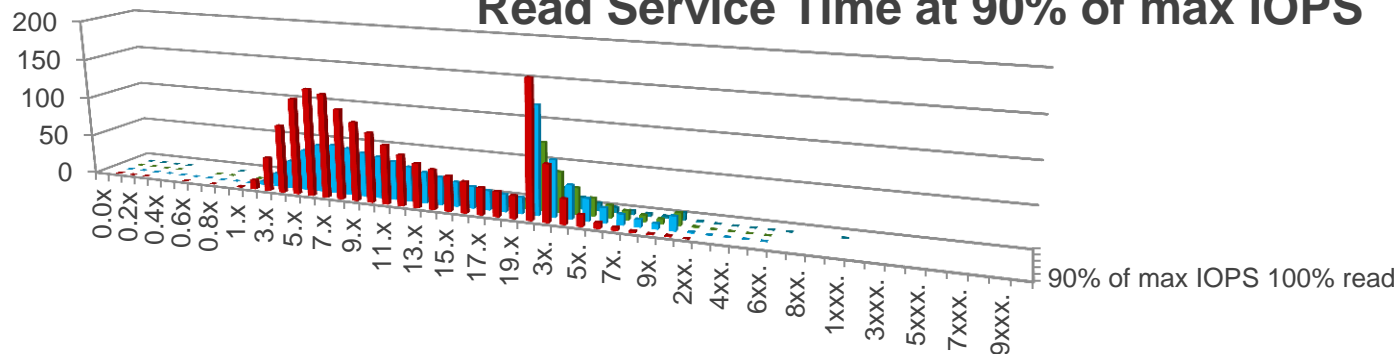
Actually achieved  
PG busy %

# Fine-grained dynamic feedback control



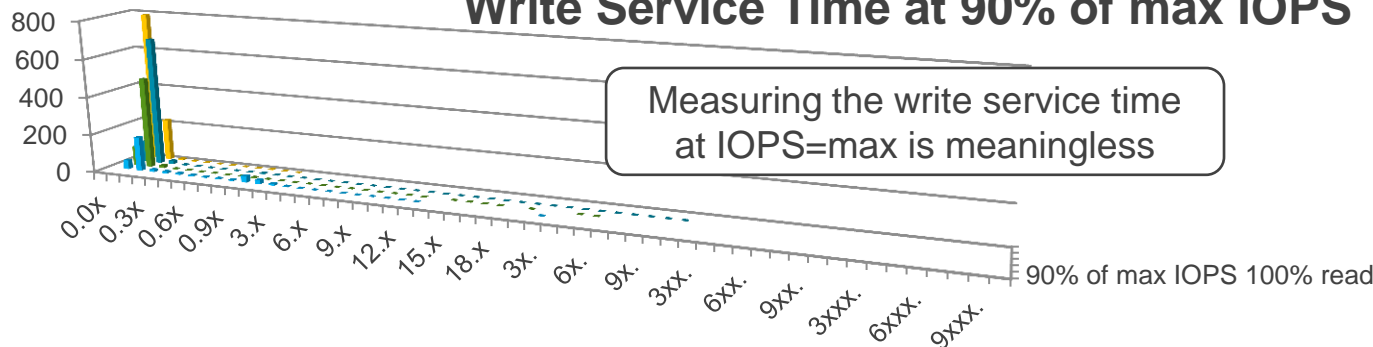
# Service time histograms

## Read Service Time at 90% of max IOPS



- 90% of max IOPS 100% read
- 90% of max IOPS 75% read
- 90% of max IOPS 50% read
- 90% of max IOPS 25% read
- 90% of max IOPS 0% read

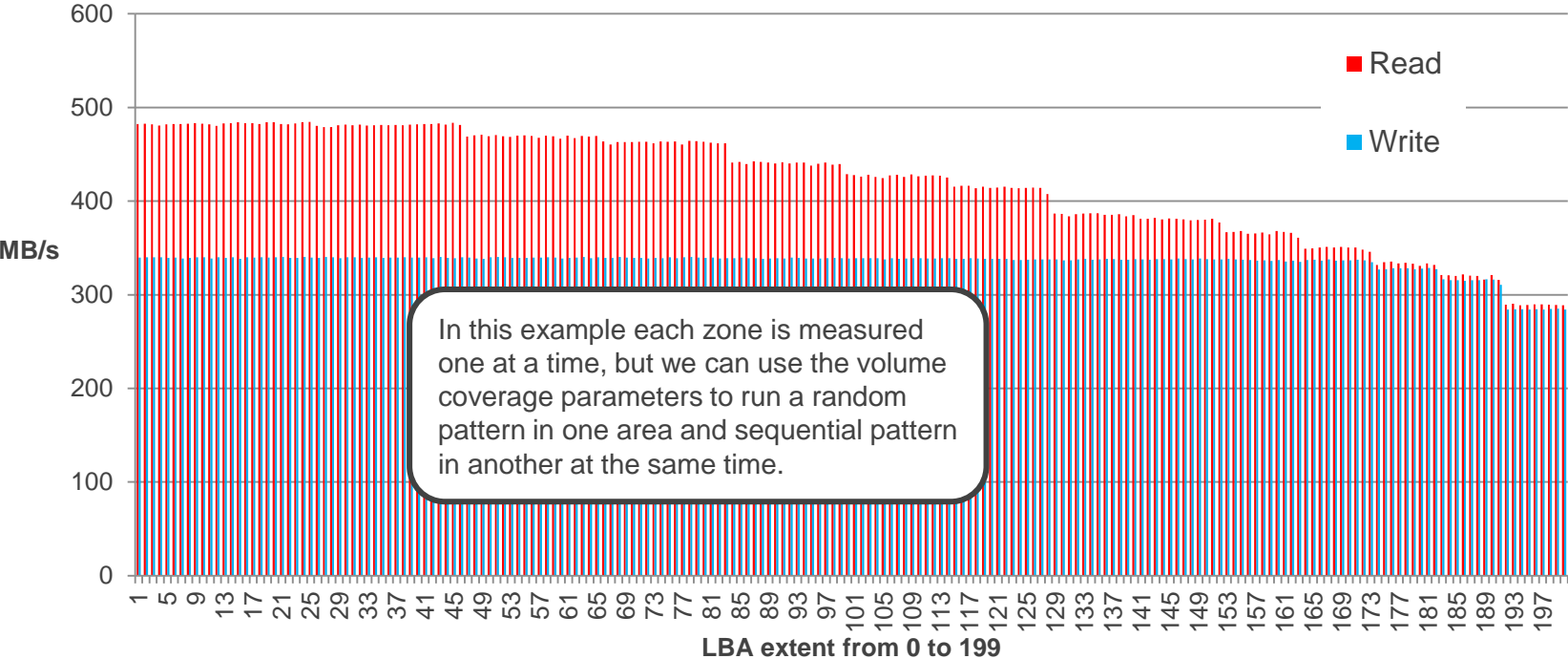
## Write Service Time at 90% of max IOPS



- 90% of max IOPS 100% read
- 90% of max IOPS 75% read
- 90% of max IOPS 50% read
- 90% of max IOPS 25% read
- 90% of max IOPS 0% read

# Layered workloads can be placed within the LUN

## Sequential MB/s by zone





- Written in C++ using Linux native kernel Asynchronous I/O interface
  - Lowest CPU overhead – minimizes test host hardware cost & maximizes measurement accuracy
- Designed for open source / vendor independence.
- Super easy test setup – select by LDEV, by Port, by PG, Pool ID
- Tests for minimum time to make a valid measurement to specified +/- % accuracy
- Real-time dynamic feedback control (DFC) of workload
  - "find what IOPS is needed to obtain 1.0 ms service time"
- Records only valid measurement data
  - Can check number of resources reporting (ports, PGs, ...) and that none running too slowly.
- Workflow automation using ivyscript programming language wrapper around ivy engine.

- August, 2016
  - Internal HDS performance team user beta test in progress, including test of dedupe & compression support
  - Released as open source.
  - Demos available for all functionality.
  - Support for HUS VM, VSP Gx00.
  - Support for HUS100 series.

- Some ideas:
  - Repackage access to ivy engine via a RESTful API.
    - Build CLI on RESTful API to enable programming ivy in Python or any other language
    - Remove outer ivyscript programming layer wrapper, exposing ivy engine control statements as CLI commands.
  - Develop VSP G1000 support
  - Extend VSP Gx00 support
  - Develop "auto gain control" feature for PID loop
  - Transition to POSIX asynchronous I/O and port to other OS platforms.



# Thank You

**HITACHI**  
Inspire the Next 