How Different *are* High, Medium, and Low Pool Processors?

Scott Chapman

Enterprise Performance Strategies, Inc.

Scott.chapman@EPStrategies.com

Contact, Copyright, and Trademarks

Questions?

Send email to performance.questions@EPStrategies.com, or visit our website at https://www.epstrategies.com or http://www.pivotor.com.

Abstract (why you're here!)

HiperDispatch has been around for a number of years now, but there is still a misunderstanding of the true differentials and effectiveness of logical processors designated as high, medium, and low. In addition, there is the seemingly never ending questions of how HiperDispatch determines the number of high, medium, and low pool processors for an LPAR. A common practice is to optimize LPAR configuration such that the most important LPARs have at least one high pool processor. But how much does this matter in real life? How much benefit can you expect to gain for your most-loved LPARs if you can give them an extra high-pool processor? How much might that hurt other LPARs?

During this webinar, Scott Chapman will dive deeper into HiperDispatch and help the attendees better understand the true meaning and effectiveness of each pool of processors.

Agenda

- Brief overview of HiperDispatch
- Medium pool rules
- Common HiperDispatch expectations & measurements
- What do we see in real life measurements?
- Conclusion: how much should you worry about this?

HiperDispatch Overview

HiperDispatch History

- HiperDispatch was introduced on the z10 in 2008
- Goal was to improve performance through improved cache coherency
  - Basically: don't needlessly split work across lots of CPs if you can keep like work on a smaller number of CPs
  - Mitigates the "short CP" problem
    - Caused by having high ratio of logical to physical CPs
- Changed both PR/SM and z/OS dispatching
- Was originally optional, but default and expectation is now "On"
  - Required in some configurations and if using SMT

Vertical CP Management

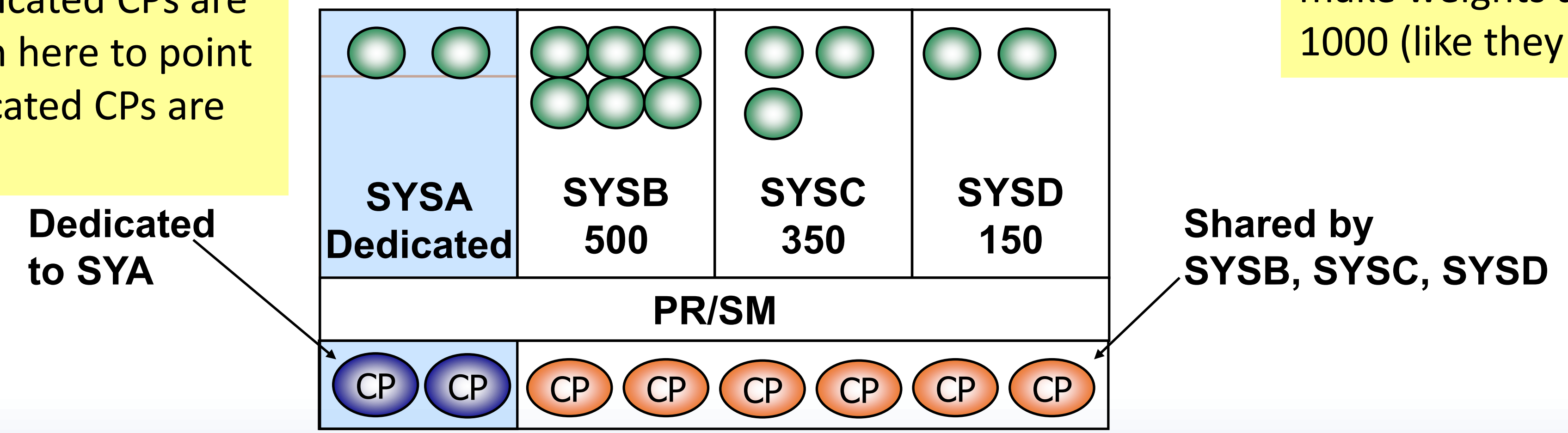
- HiperDispatch manages CPs "vertically", meaning it endeavors to make the logical CPs a larger percentage of a physical
- Logical processors classified as:
  - High – The processor is quasi-dedicated to the LPAR (100% share) (VH)
  - Medium – Share between 0% and 100% (VM)
  - Low – Unneeded to satisfy LPAR's weight (VL)
- This processor classification is sometimes referred to as "vertical" or "polarity" or "pool"
  - E.G. Vertical High = VH = High Polarity = High Pool = HP
- Parked / Unparked
  - Initially, VL processors are "parked": work is not dispatched to them
  - VL processors may become unparked (eligible for work) if there is demand and available capacity

Guaranteed Share as Processors

- Each LPAR's share can be translated into a number of processors
  - $LPAR\ Guaranteed\ Processors = LPAR\ Share\ * Shared\ Processor\ Count$
- In below example, there are 6 shared processors so:
  - SYSB = 500/1000 * 6 = 3 processors
  - SYSC = 350/1000 * 6 = 2.1 processors
  - SYSD = 150/1000 * 6 = 0.9 processors

For ease of use, try to make weights add up to 1000 (like they do here).

LPARs with dedicated CPs are rare, but shown here to point out those dedicated CPs are separate

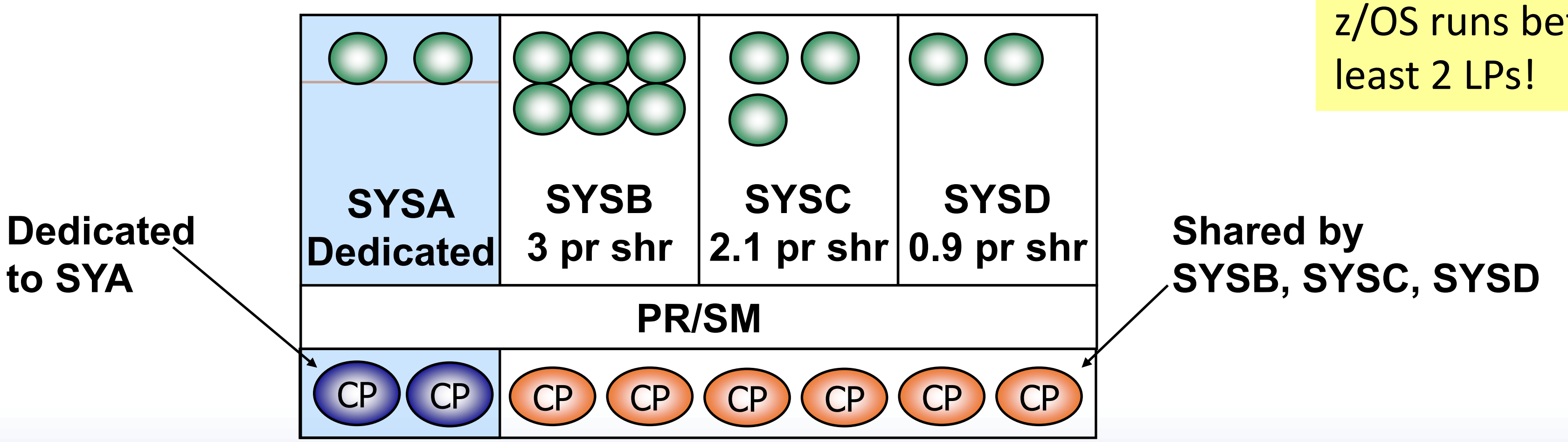


Horizontal CP Management

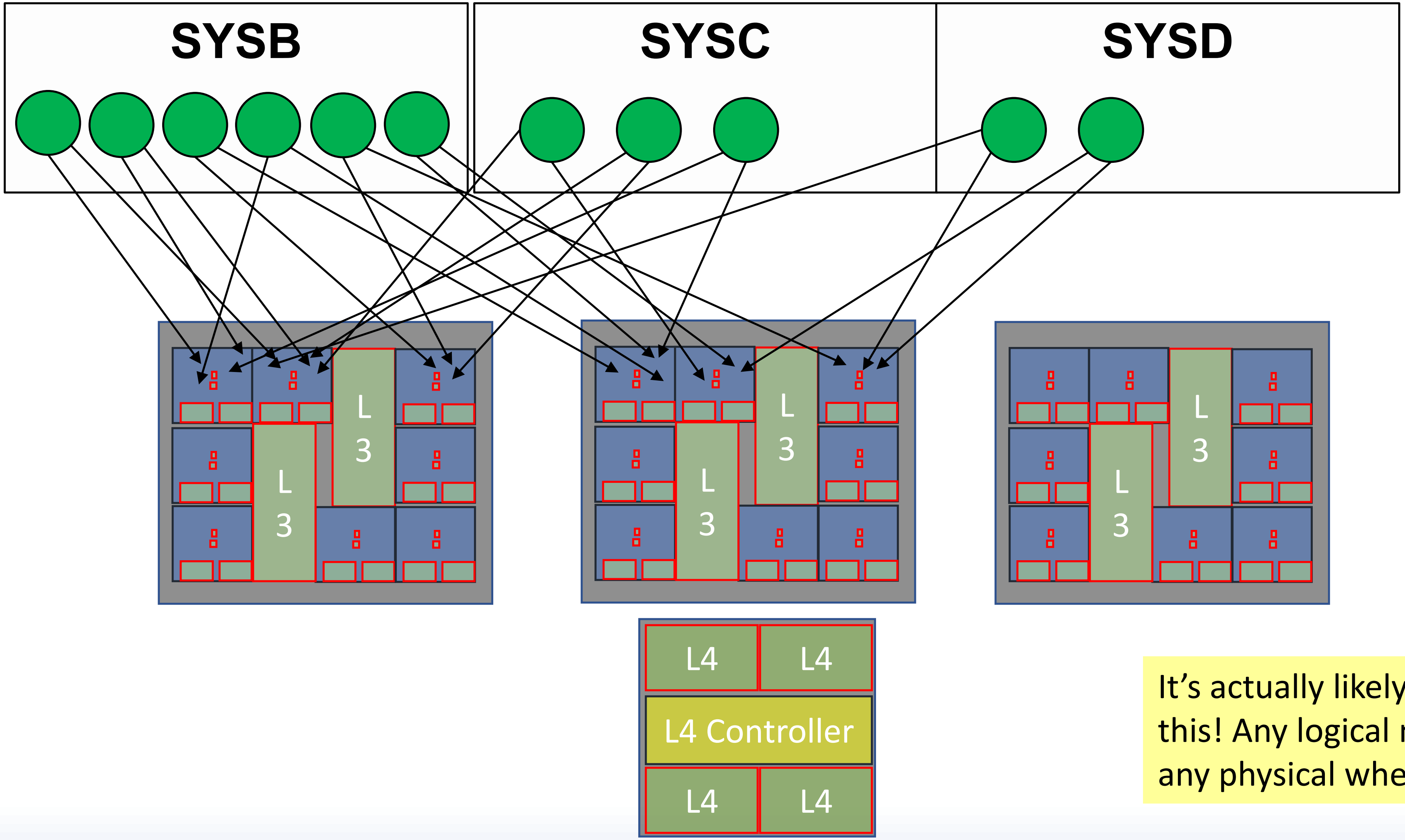
- Prior to HiperDispatch, PR/SM would split each logical CPU evenly based on its average share of a processor
  - SYSB gets 6 LPs, each effectively 50% of a physical (3 / 6)
  - SYSC gets 3 LPs, each effectively 70% of a physical (2.1 / 3)
  - SYSD gets 2 LPs, each effectively 45% of a physical (0.9 / 2)

Can lead to what's called "short CPs": note SYSB's logicals spend less time dispatched to a physical than SYSC's!

z/OS runs better with at least 2 LPs!

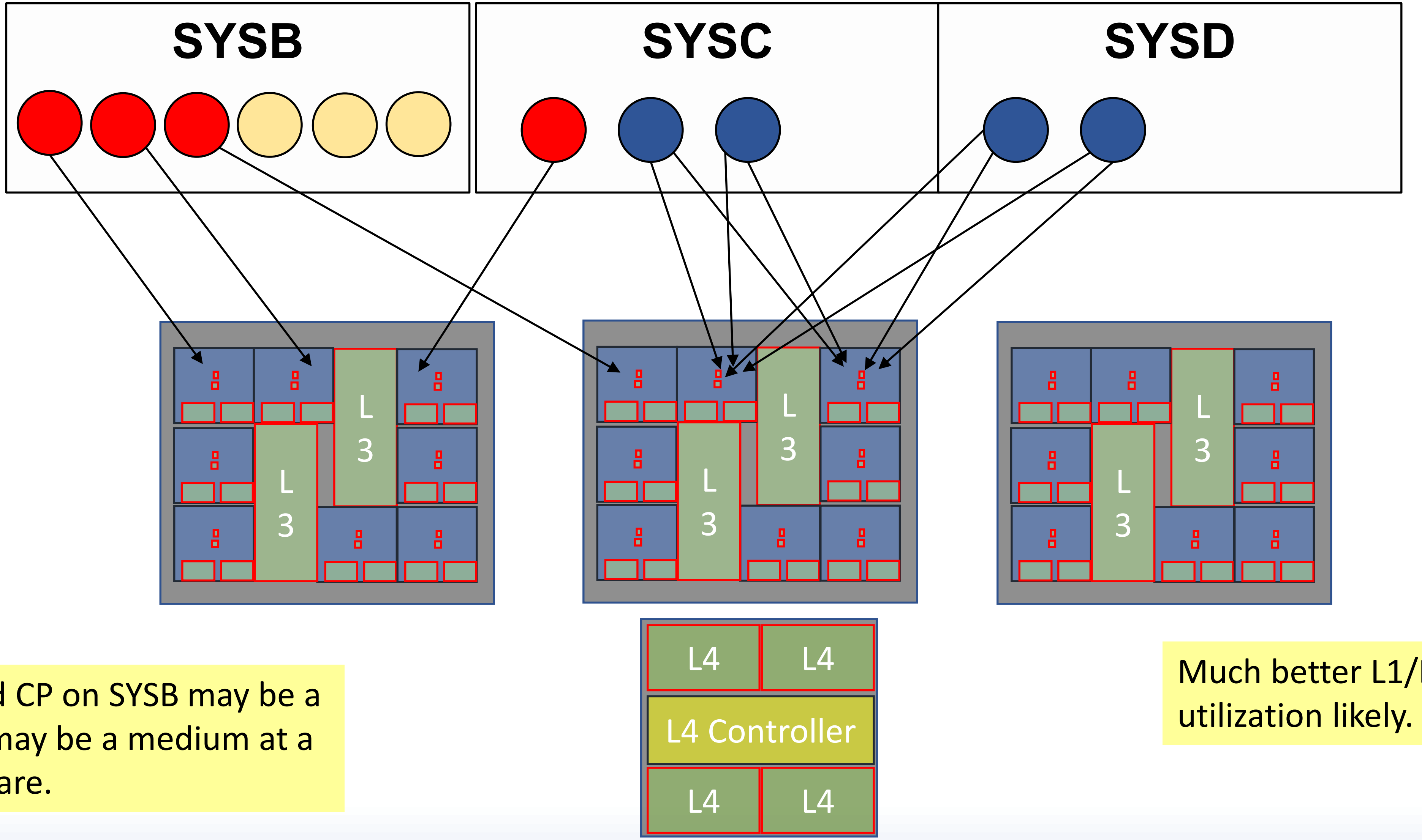


HiperDispatch Off



It's actually likely worse than this! Any logical might end up on any physical when redispatched.

HiperDispatch On



The third CP on SYSB may be a high or may be a medium at a 100% share.

Much better L1/L2 cache utilization likely.

Medium Pool Rules

Medium Pool Processors

- HiperDispatch prefers not to have VMs with low weight
  - Instead a VH will be taken as a second VM and the two VMs sharing the weight of those two engines
- E.G. an LPAR with weight giving it access to 2.4 CP’s worth of capacity:
  - ~~2 VH (100% each) + 1 VM (40%)~~ <- PR/SM will not do this
  - 1 VH (100%) + 2 VM (70% each) <- PR/SM will do this
- Basically PR/SM wants a single medium pool CP to get at least a 50% share of a physical CP
  - If the weight of an LPAR is just under n.5 CPs of capacity getting it to n.5 should result in an extra VH

Potential VM Confusion

- z13 has different rules for when the weight is between 1.5 and 2.0 CPs
  - Instead of 1 VH and 1 VM, gets 2 VMs
- Can only have a VM if it’s weight would be at least 0.5 CPs
  - Otherwise a VH is demoted and the combined weight is divided between the two VMs
  - E.G. and LPAR with a weight of 2.1 CPs would have 1 VH, and 2 VMs at 0.55 each
- So if there’s 2 VMs, they will always have a weight between 0.5 and 0.75
  - I.E. ((1 + .01) / 2) <= x <= ((1 + .49) / 2)
  - Except the z13 scenario above, where both will be > 0.75
- But if the VM would have had a weight > 0.5 it can stand on its own
  - And such a solo VM could have a weight approaching 1

HiperDispatch Expectations

High-pool love, Low-pool hate

- Common belief / expectation:
  - VH processors perform better 🥰
  - VL processors perform worse 😠
  - HiperDispatch is geared towards machines with many processors
- It is common to hear recommendations to tweak LPAR weights to get an extra VH processor for a loved LPAR
- Also common is the recommendation to not use low-pool processors
  - IBM recommendation to not have more than 2 VL processors
  - Note we're only talking about z/OS running under PR/SM in this presentation: impacts to z/VM and z/Linux may be different

How can we measure efficiency?

- Commonly cited:
  - CPI – Cycles Per Instruction – lower is better
    - Can be broken down into
      - Instruction Complexity CPI – CPI influenced by the instruction mix
      - Finite Cache CPI – CPI influenced by cache contention (because caches are finite)
  - RNI – Relative Nest Intensity – lower is better
    - Calculates a number that is workload-related and should remain somewhat stable when moving between processor generations
    - Can be useful for showing the relative impact of cache misses at each level
- More directly: if you make a change and the CPU consumption for the workload goes down, that was a good change
  - Note you can't take single measurements though—you have to look over multiple executions to account for normal variations and cross-workload contentions

Can we justify the love/hate?

- Probably the easiest way to show this is to look at the Estimated Finite CPI for each processor, with the expectation:
  - VH will show lower Est Finite CPI
  - VL will show higher Est Finite CPI
  - VM will be in the middle

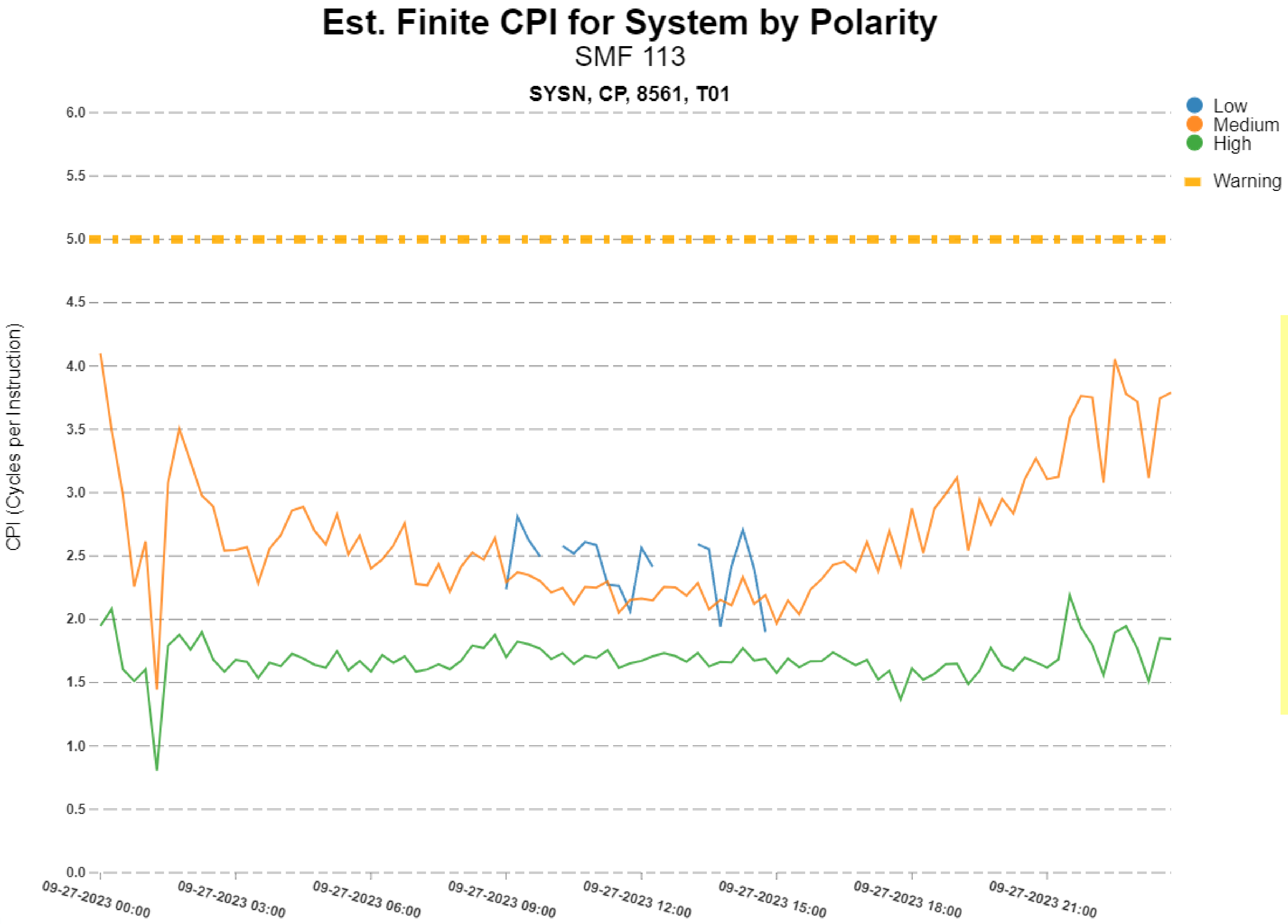
- But do we see this?

Maybe

Sometimes

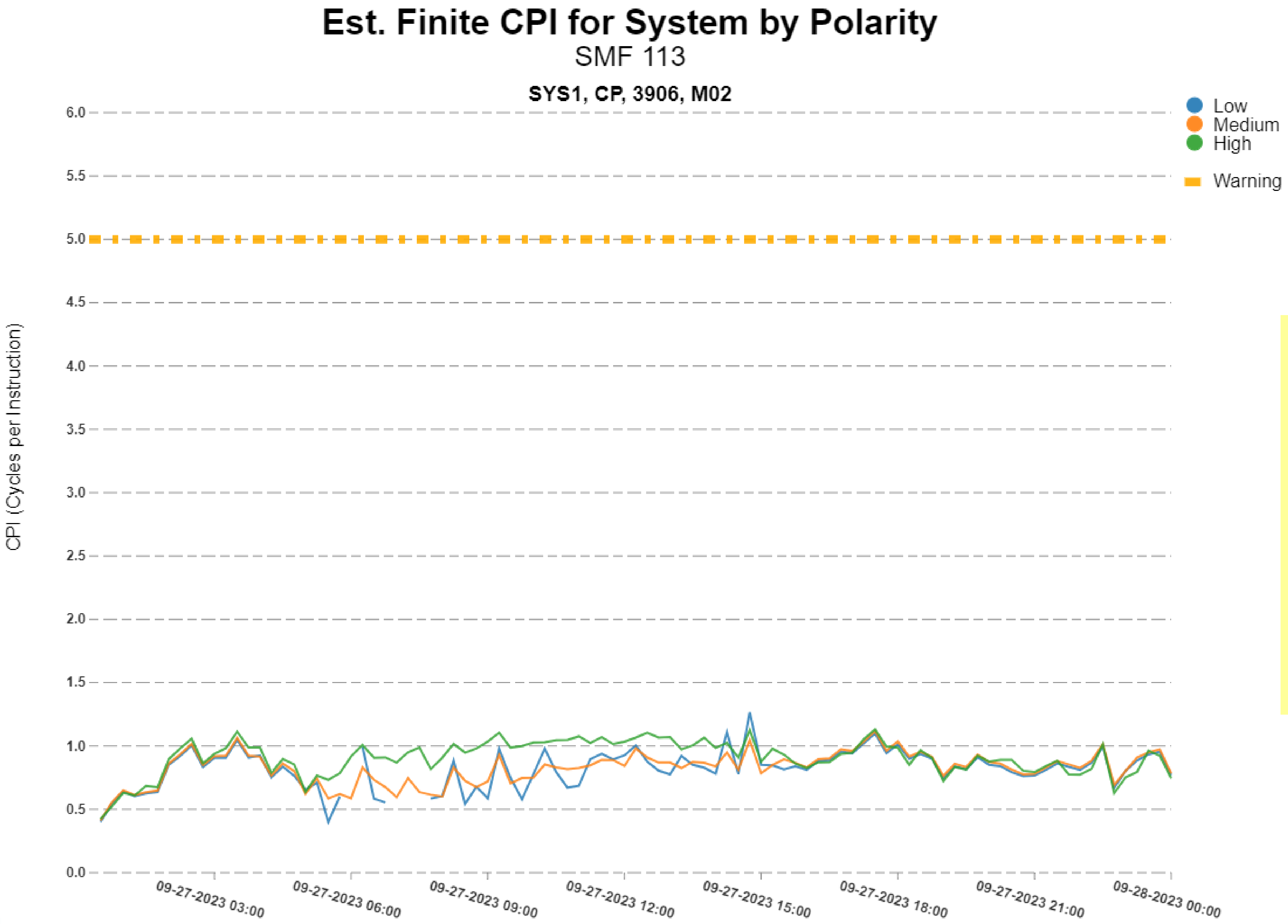
It Depends

Real life measurements

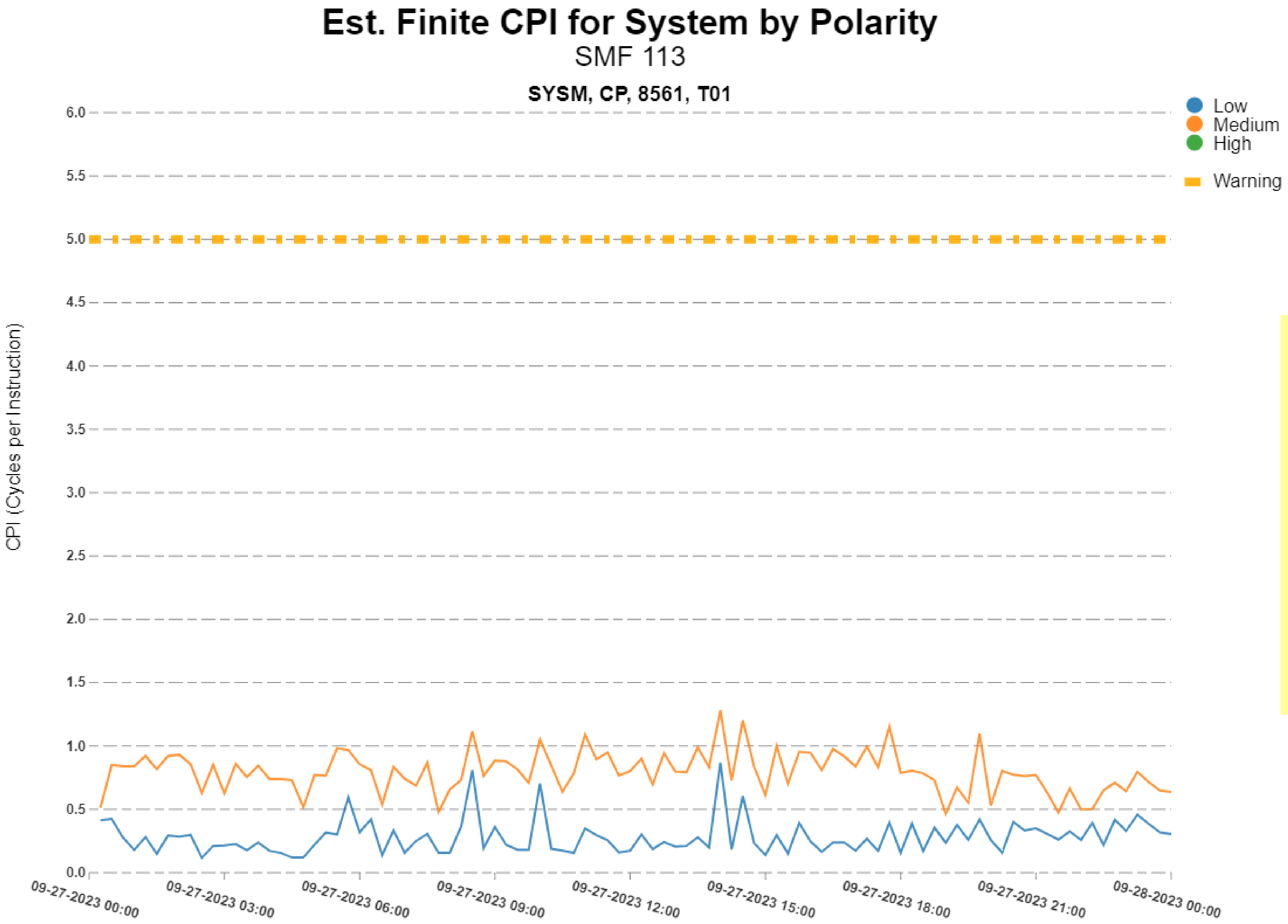


So the assumption is that if we looked at estimated finite CPI by engine polarity, we'd see patterns that usually looked like this.

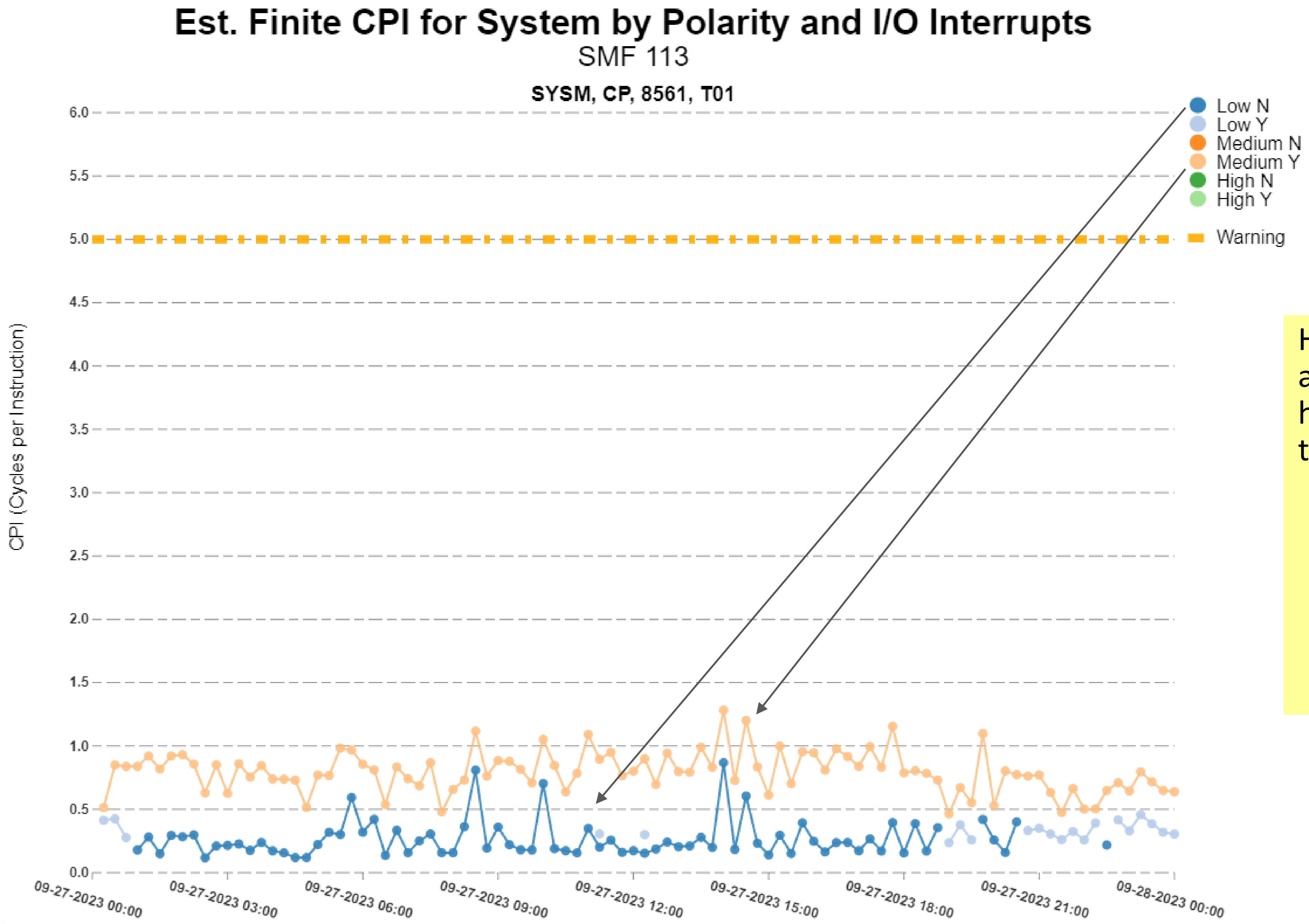
Reality is rarely this nice!



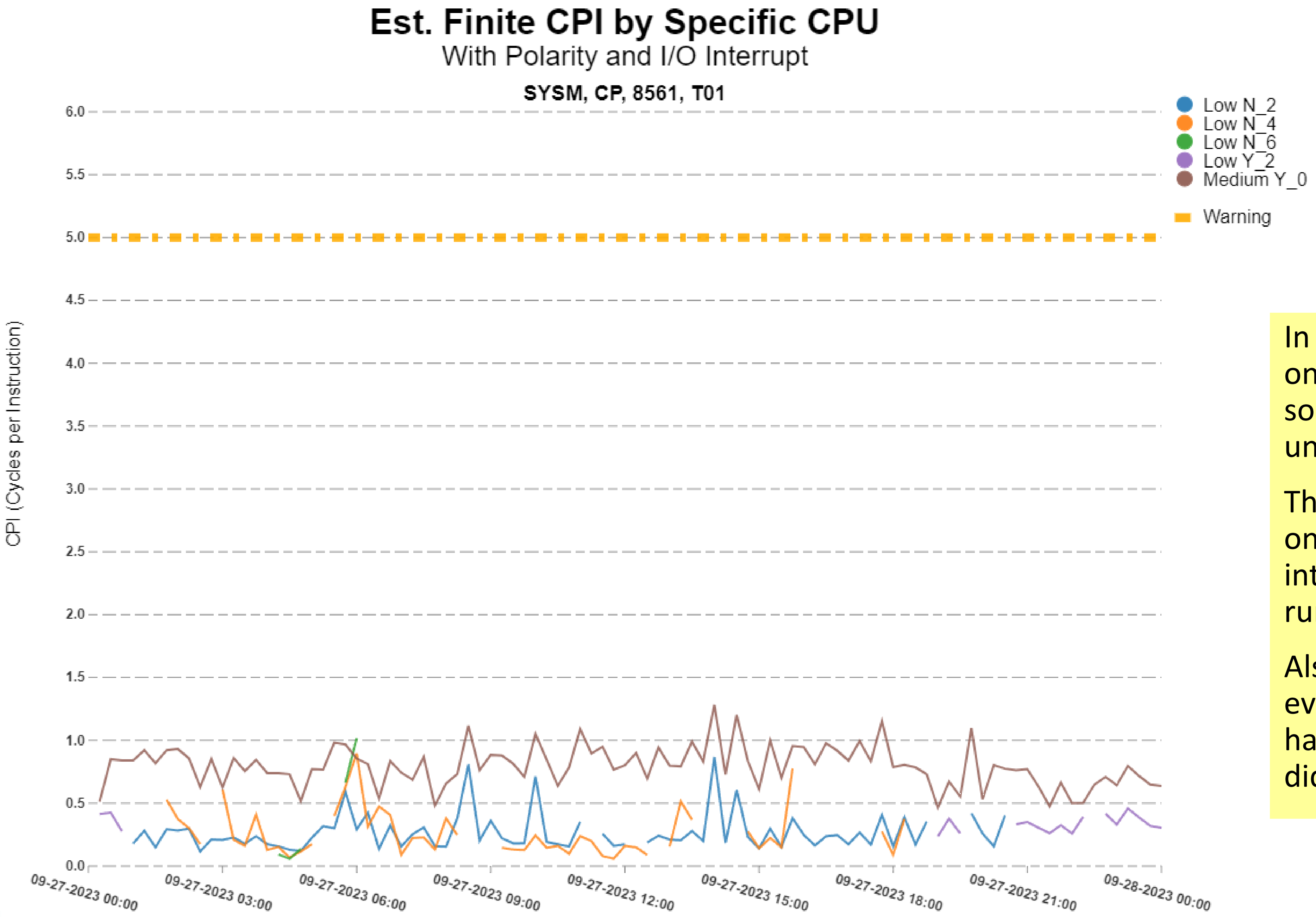
Instead we see systems like this where there is either no difference or maybe even the high pool processor shows running worse than the medium/low!



Or how about this, where there is no high, but the low(s) always seem to be more efficient than the medium(s)??



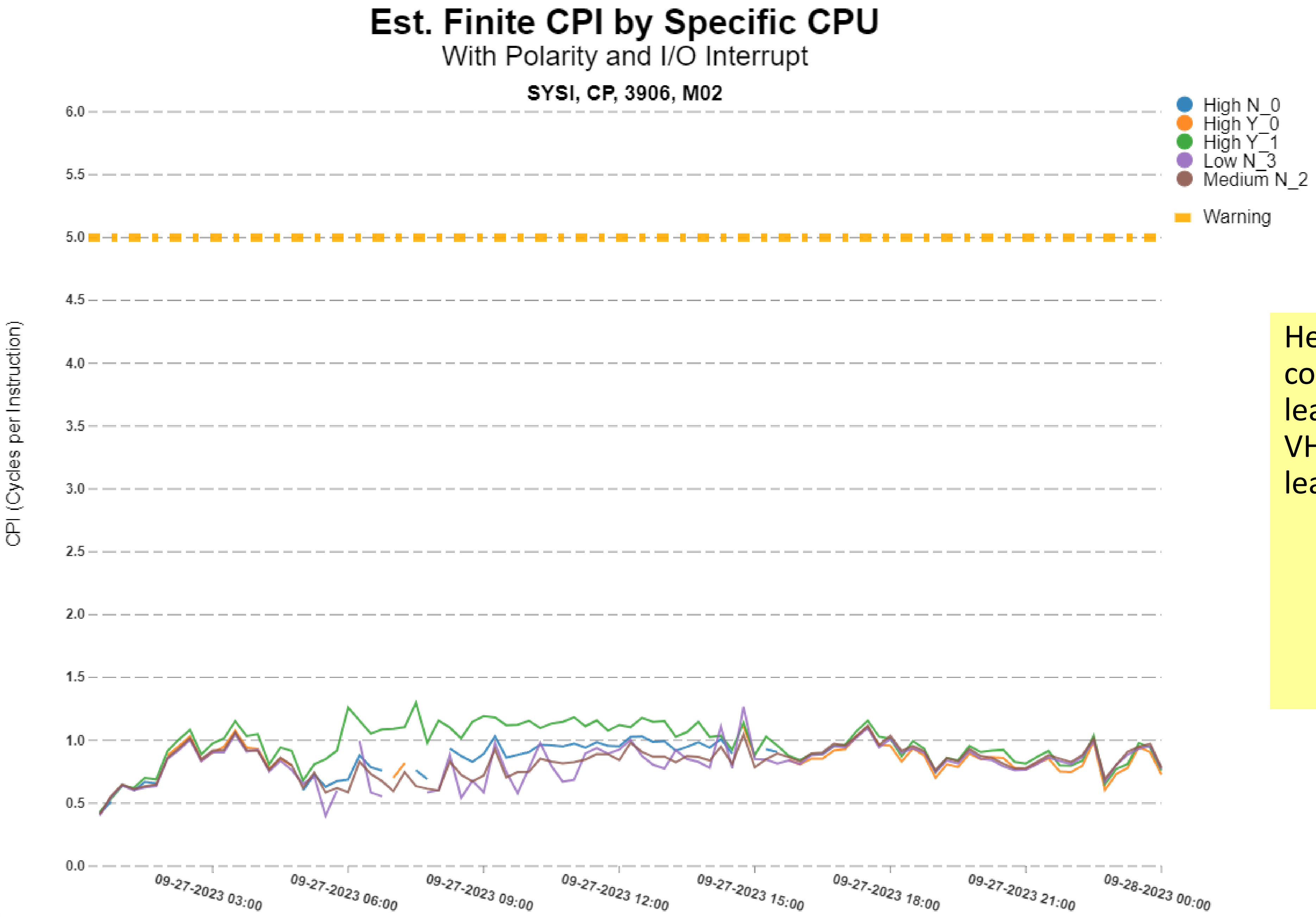
Here's part of the answer: CPs which handle I/O interrupts tend to be less efficient.



In this case, the LPAR only has 1 VM and 3 VL, so one VL will always be unparked.

The VM is usually the only CP enabled for interrupts, and it so it runs less efficiently.

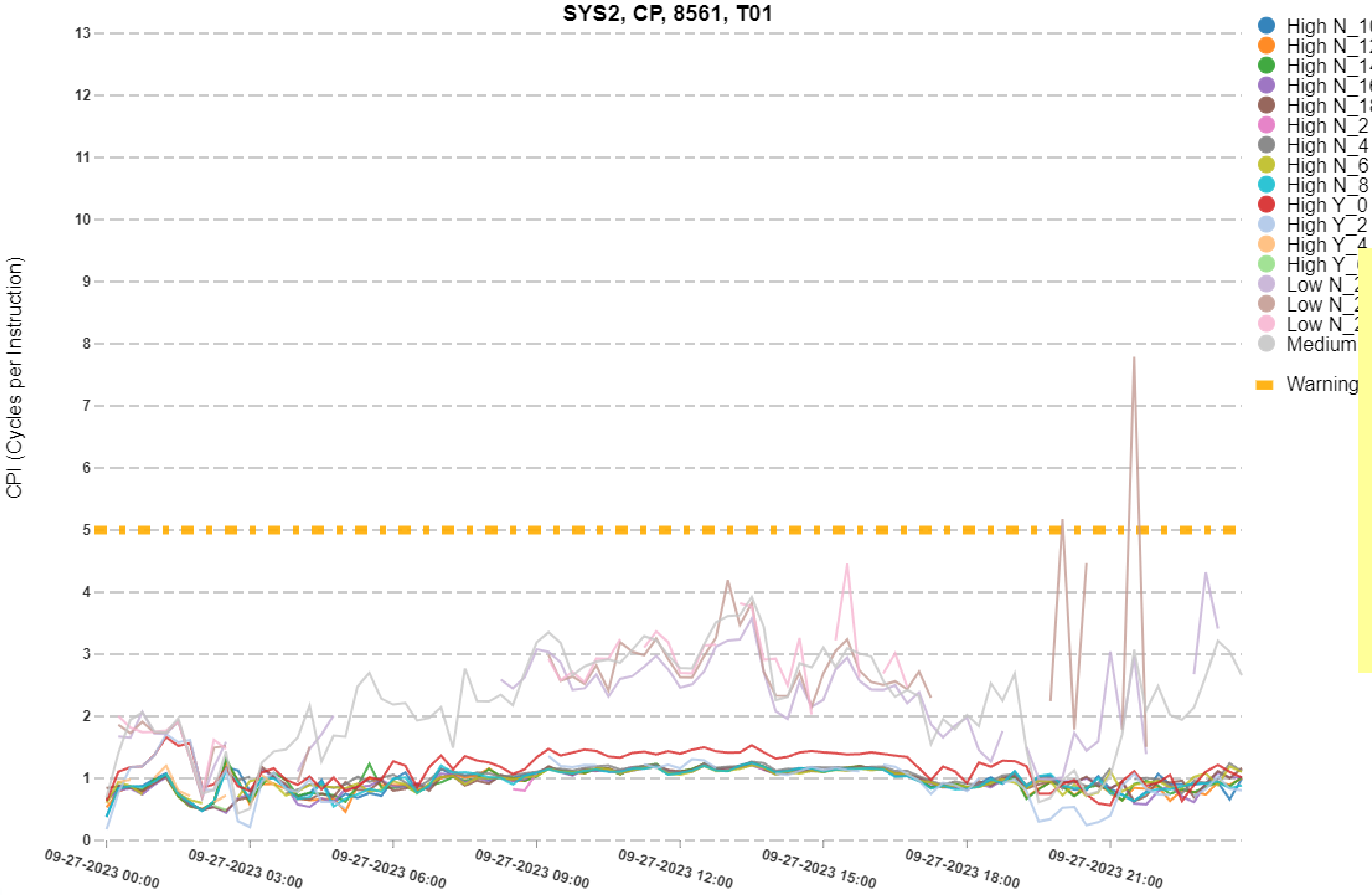
Also note in this case even when the VL did handle interrupts, it didn't handle many.



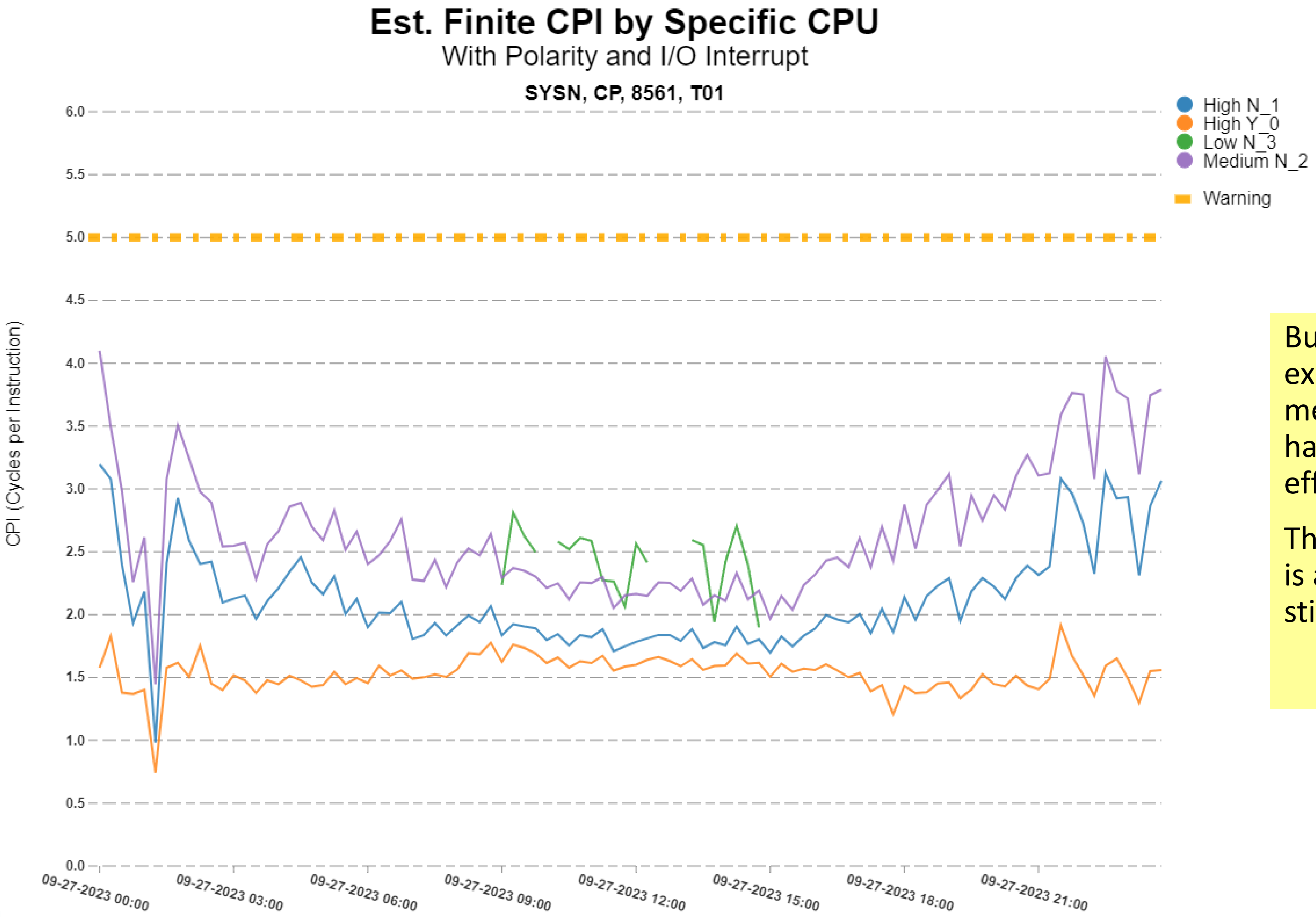
Here's a system with a couple of VH and for at least part of the day, the VH that does I/O is the least efficient CP.

Est. Finite CPI by Specific CPU

With Polarity and I/O Interrupt

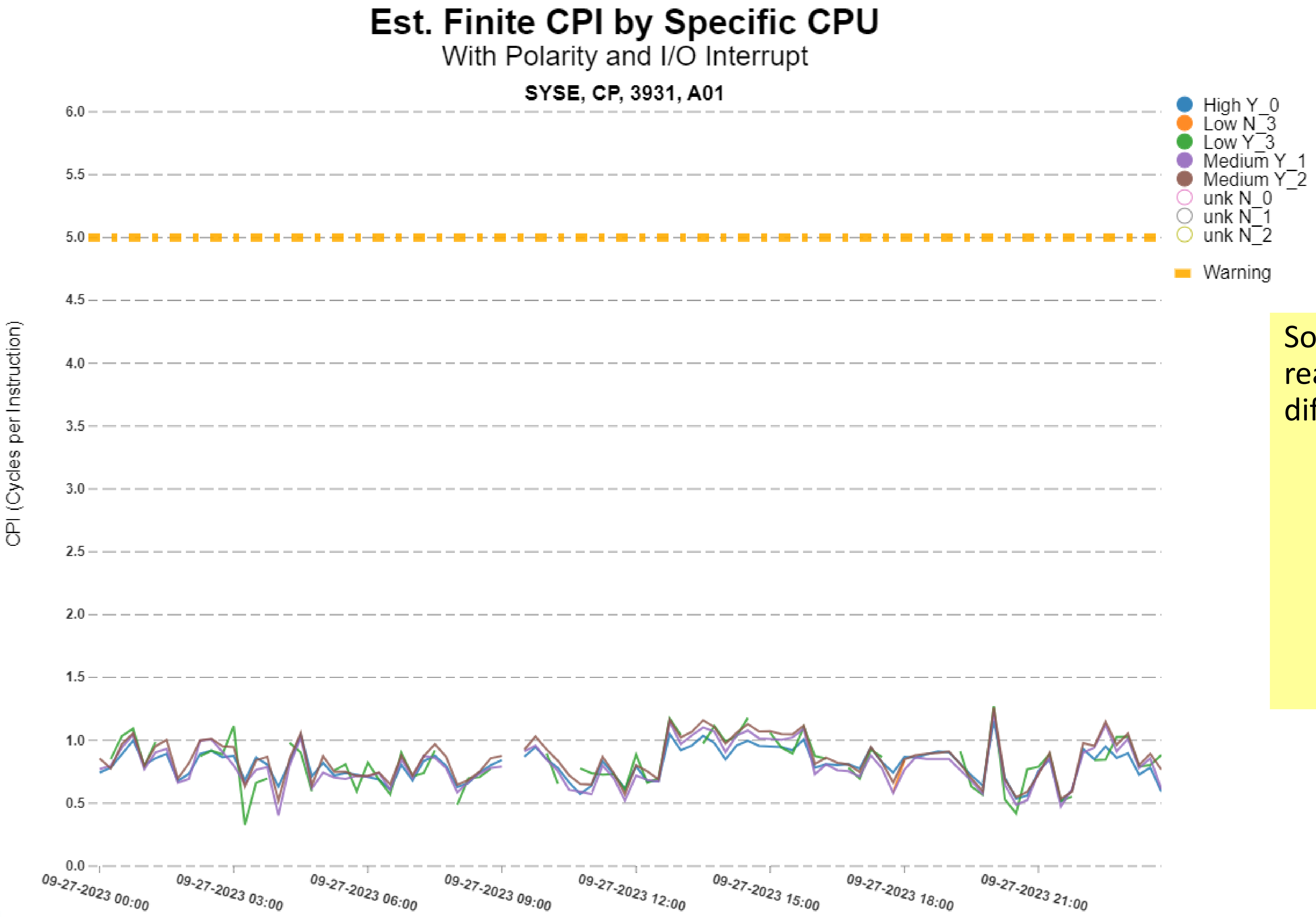


Much larger system with several VH, all of which generally outperform the VM and VLs. And the VH handling the I/O interrupts is generally a bit less efficient.



But sometimes our expectations are not met. Here the VH handling I/O is the more efficient.

This may be because this is a less busy system, but still does significant I/O.



Sometimes there just really isn't that much difference at all!

General Conclusions

- Effects of HiperDispatch most obvious on LPARs with several CPs
  - But still has value on LPARs with fewer CPs too
- Efficiency by polarity can be confusing
  - Especially when there's relatively few CPs
- On LPARs with a VM and VL, the unparked VL is effectively a VM
- On larger LPARs, VLs that are regularly used may be similar to VMs
  - But as the CEC gets busier, they will suffer more and become less efficient
- *Usually* the CPs handling I/O interrupts will be a bit less efficient
  - The VH handling I/O interrupts may be less efficient than the VM that's not
  - But if the CP has little to do other than I/O, it might appear more efficient

Less I/O = more efficient CPU

Summary: How much should you care?

- Probably not much: HiperDispatch is generally a good and helpful thing
- I/O interrupts being handled by the processors with more assigned weight is a good thing because it helps ensure interrupts aren't delayed
- A VM -> VH conversion might not result in any significant gain
  - I'll even say: probably won't in most cases
- Correcting weights to avoid using VL is still good & beneficial practice
  - Avoid risk of interference from the other LPARs
  - But some minor sporadic use of VL is fine, although "flapping" VLs can be bad
- Avoiding I/O means avoiding I/O interrupts and means reducing the efficiency impacts of handling the I/O interrupts

Questions?