Scalable High Performance Flash Systems

Jeff Bonwick
Co-founder and CTO, DSSD / Dell EMC

Yeah, yeah, yeah — but how do I process it all?

All computation is the same

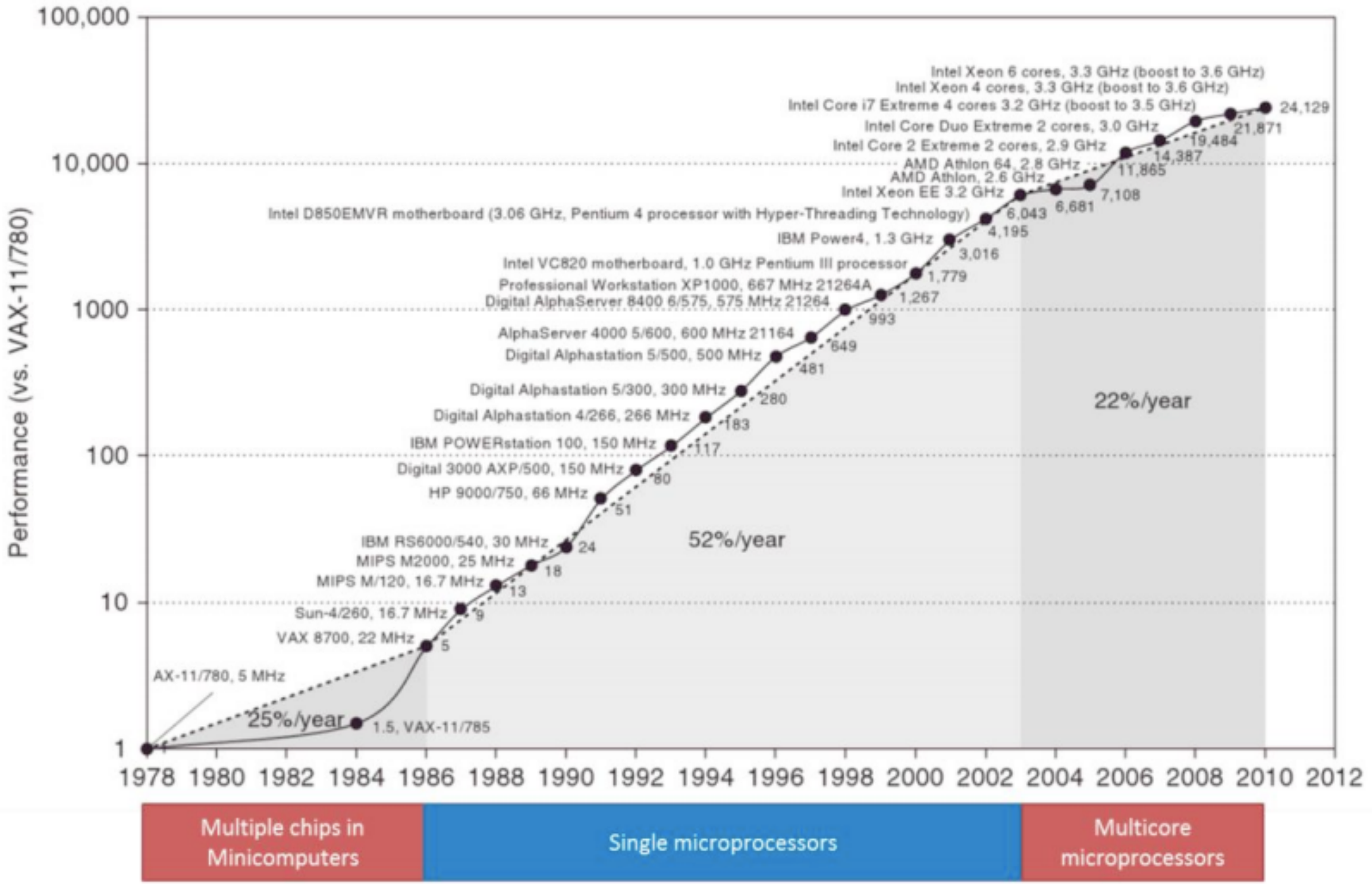
result = math(data);

To get results faster, both need to improve

- faster math
- faster data

So… how's that going?

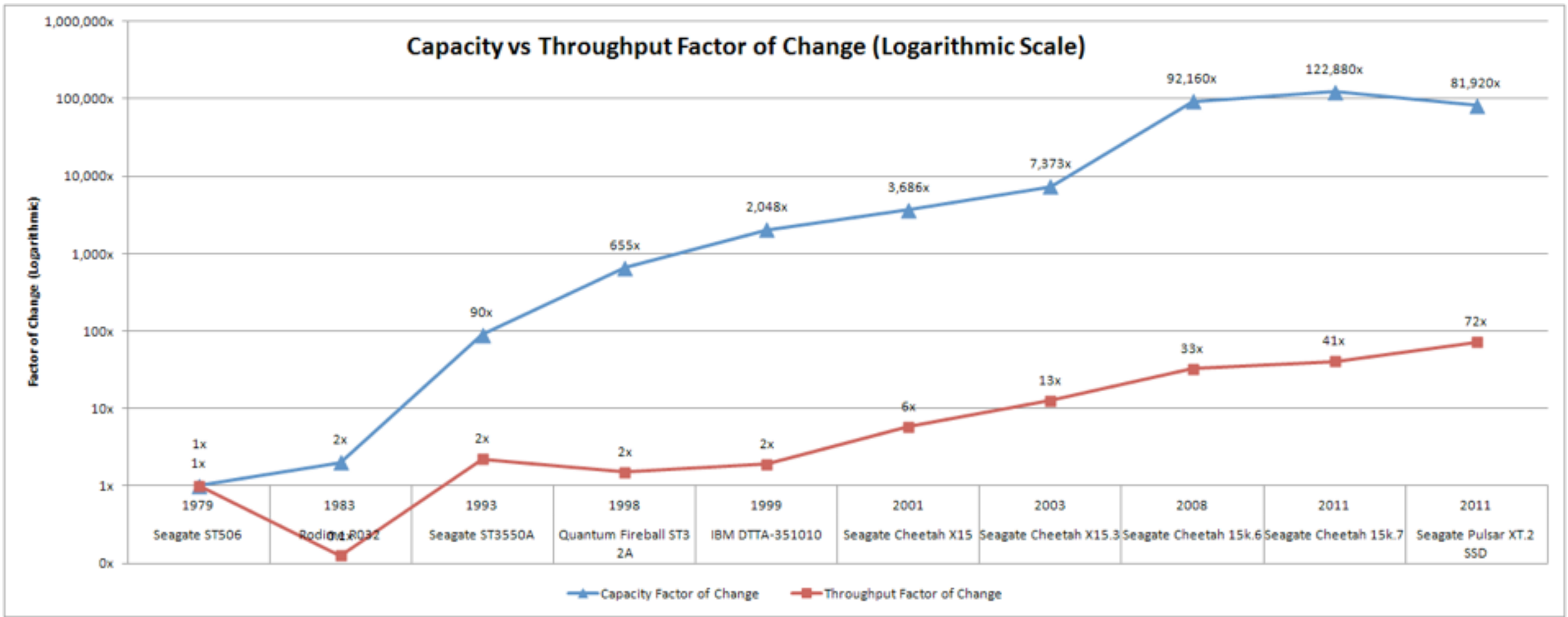
CPUs are 20,000x faster than 1980…



Source: Computer Architecture, A Quantitative Approach by Hennessy and Patterson

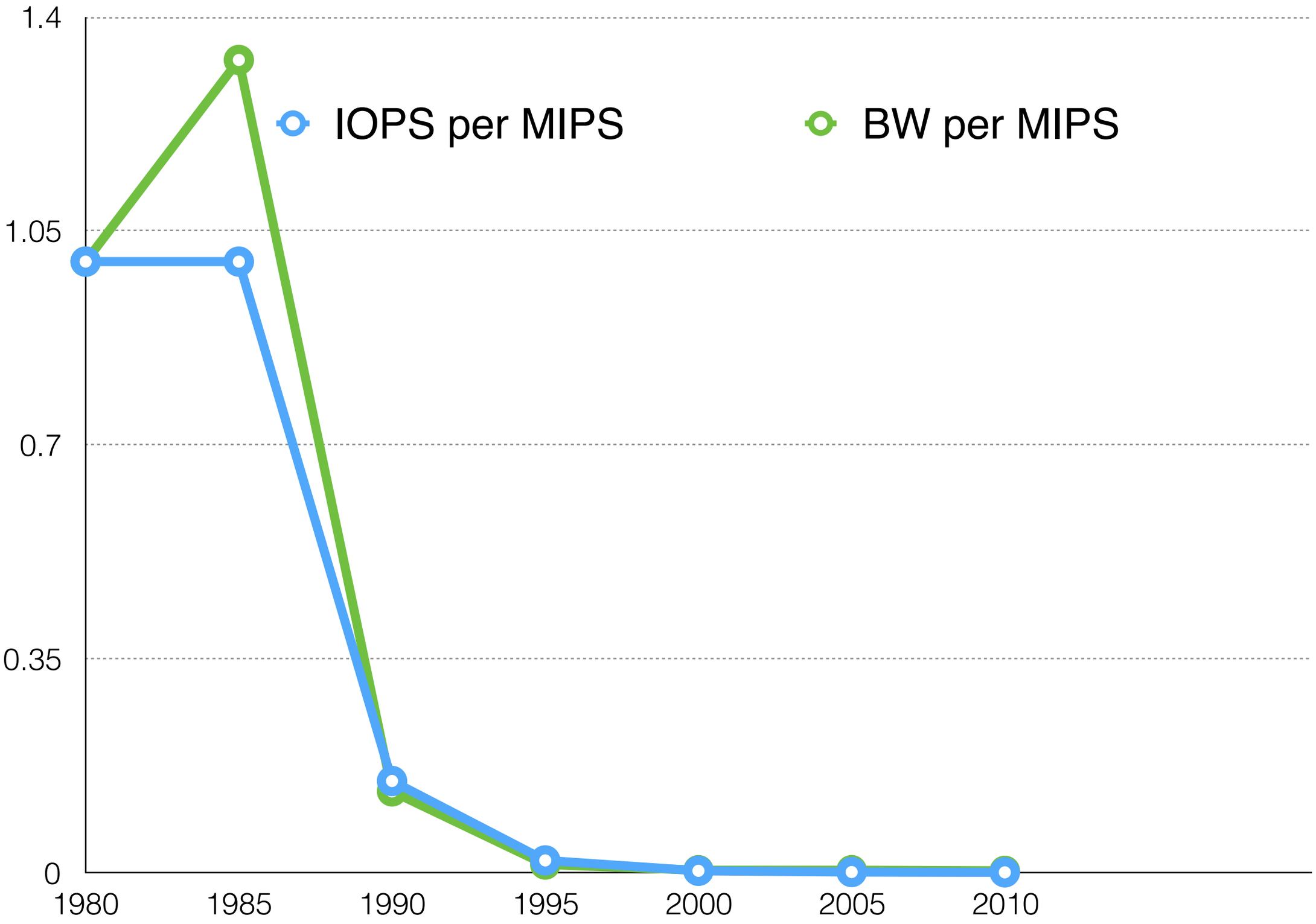
… but disk performance hasn't kept up

- Capacity has grown ~100,000x
- Bandwidth has grown ~100x
- IOPS has grown ~10x

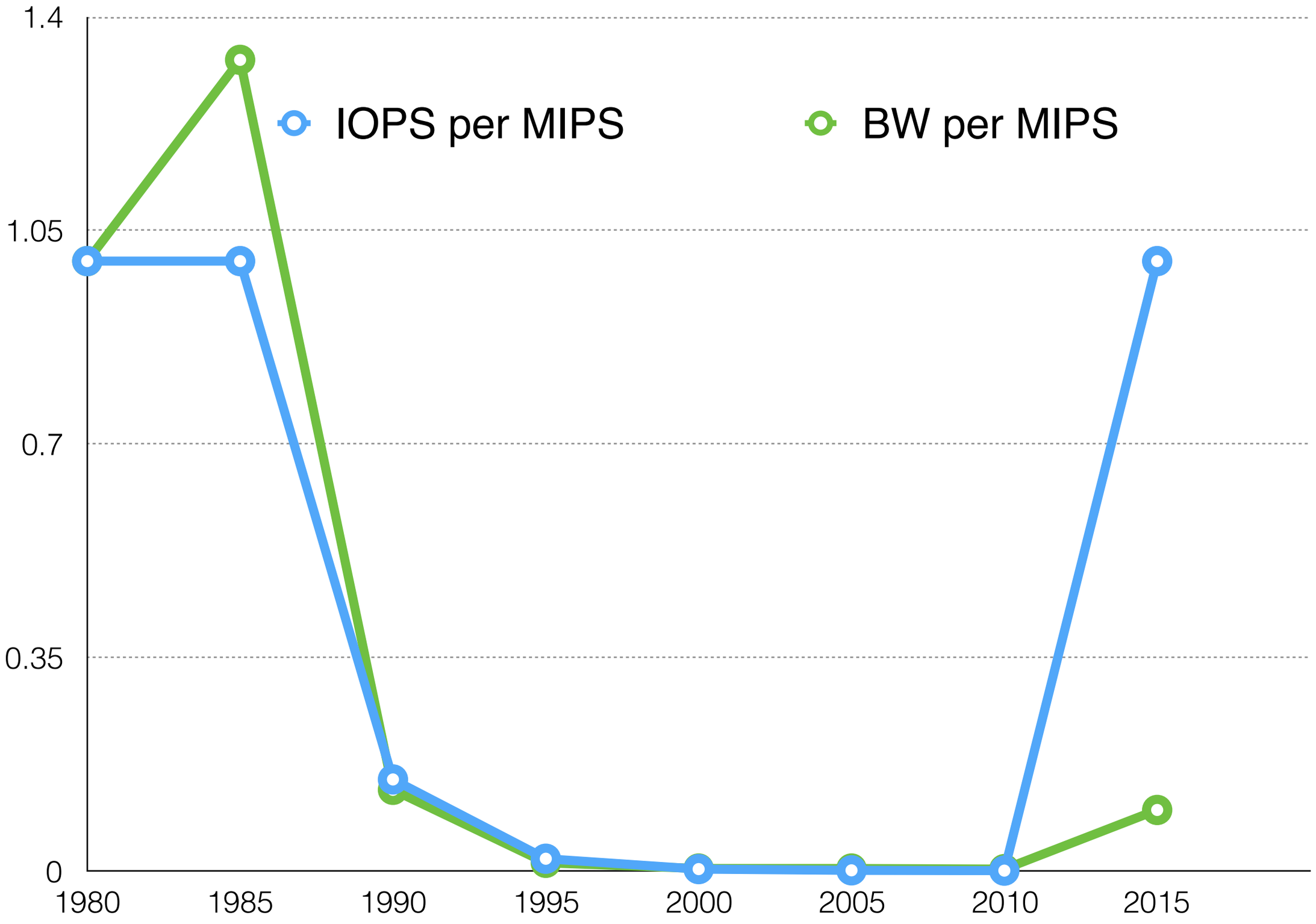


Source: https://tylermuth.files.wordpress.com/2011/11/capacity-vs-throughput-lo.png

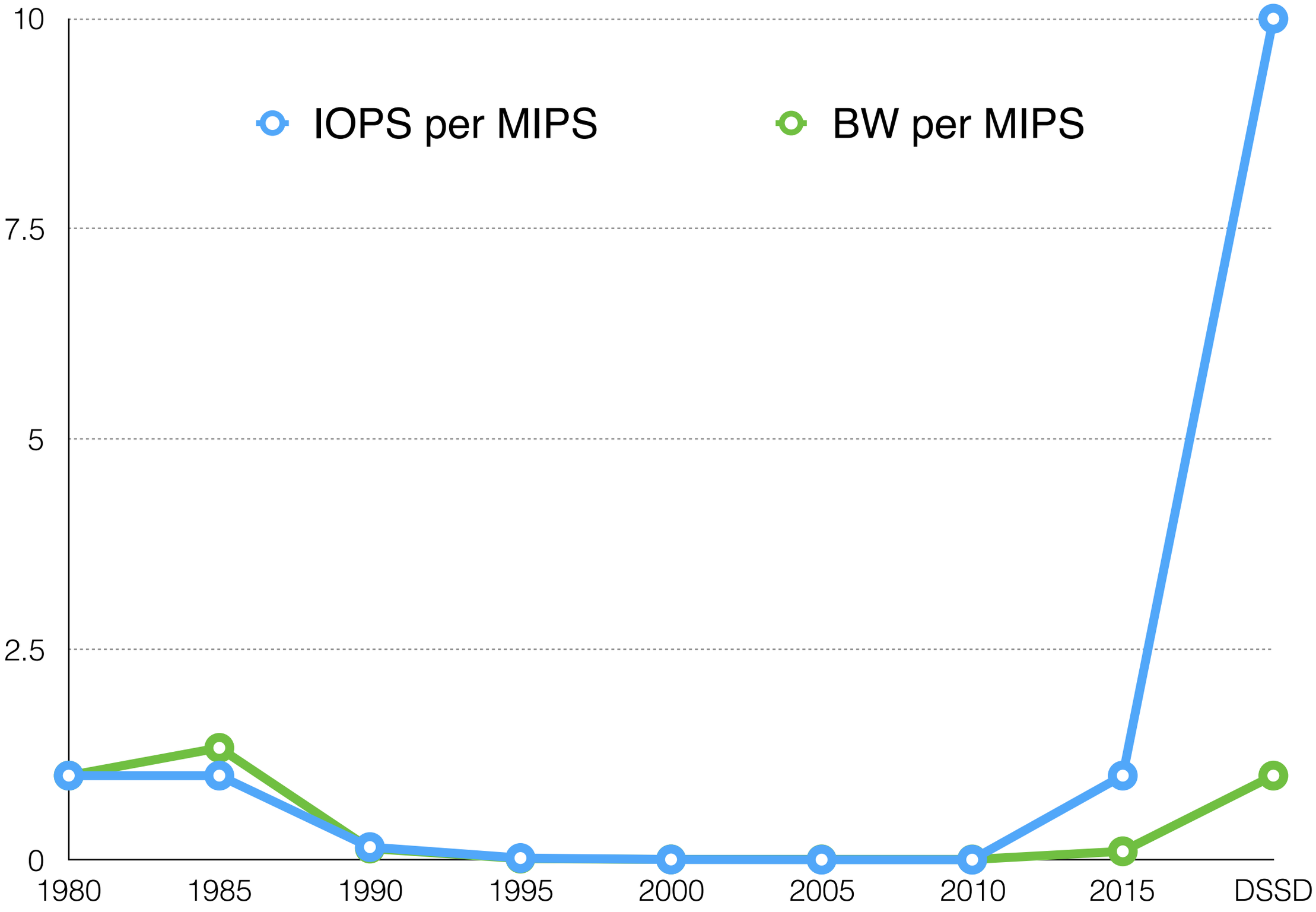
IOPS and BW per MIPS over time



IOPS and BW per MIPS, with flash arrays



IOPS and BW per MIPS, with DSSD



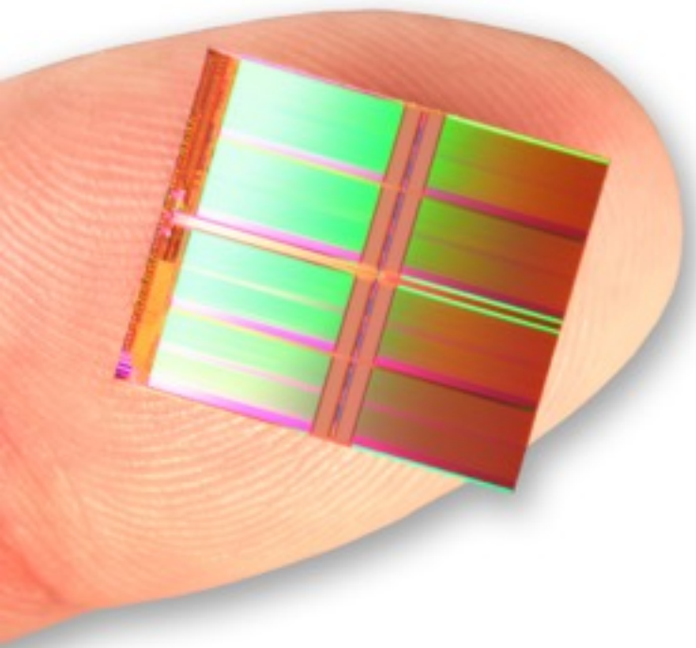
Big Data needs Big CPU and Big I/O

	1980 (HDD)	Today (DSSD)	Improvement
CPU	1	20,000	20,000x
Capacity	5 MB	100 TB	20,000,000x
Bandwidth	5 MB/s	100 GB/s	20,000x
IOPS	50	10M	200,000x

But how?

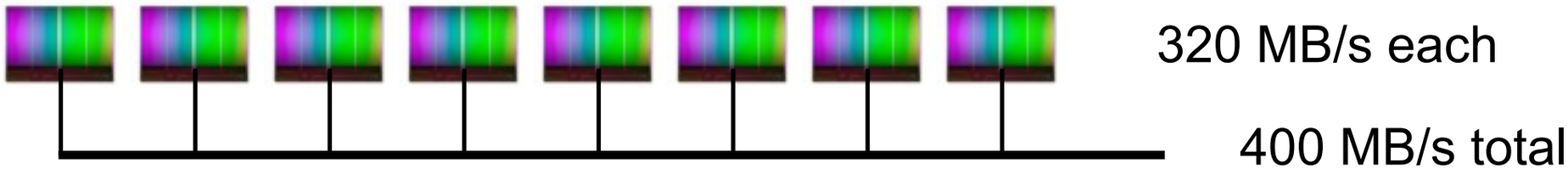
What can a single flash chip do?

- Media access time <100 us
- Implies >10,000 IOPS
- If doing 32K reads, implies >320 MB/s
- So, a single SSD containing 512 NAND die *could* deliver 160 GB/s
- Reality is just **1%** of that — why?



What limits flash performance?

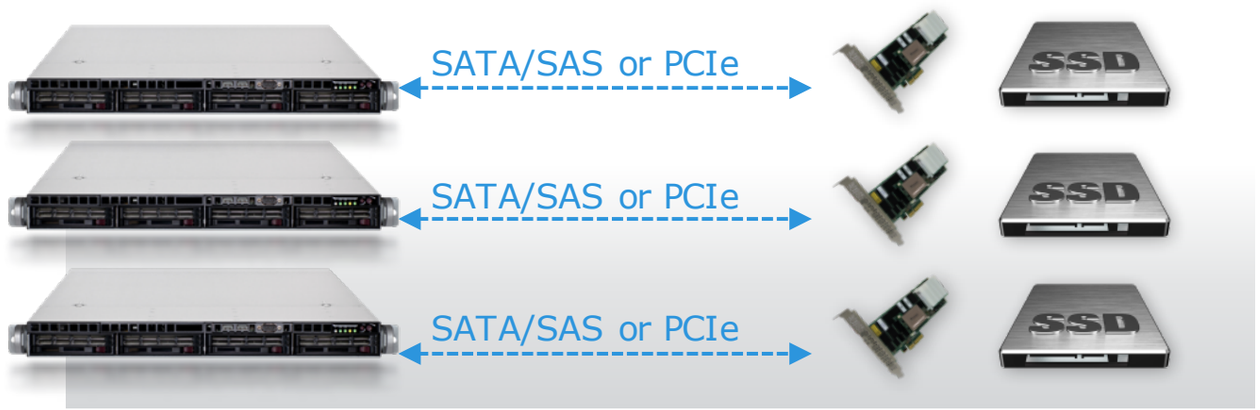
- peak power and cooling
- 8-16 NAND die share one flash channel



- 8-32 flash channels share one 12Gb SAS interface

- legacy SW stack adds 100µs or more

Two main approaches to flash thus far



Hybrid or All Flash Arrays

- Enterprise storage features

BUT

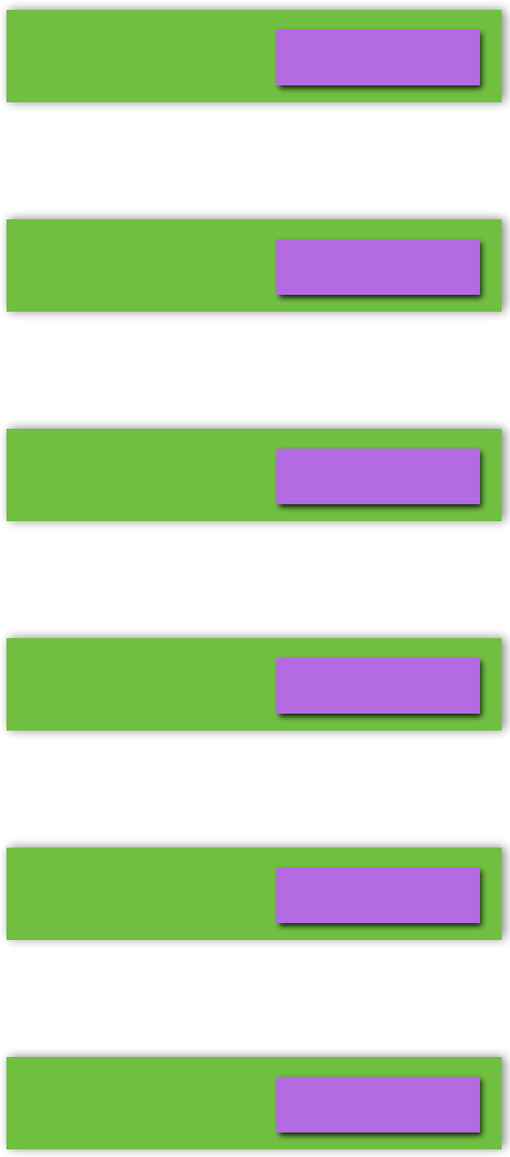
- Traditional network latencies & I/O stack bottlenecks

Server Attached Flash

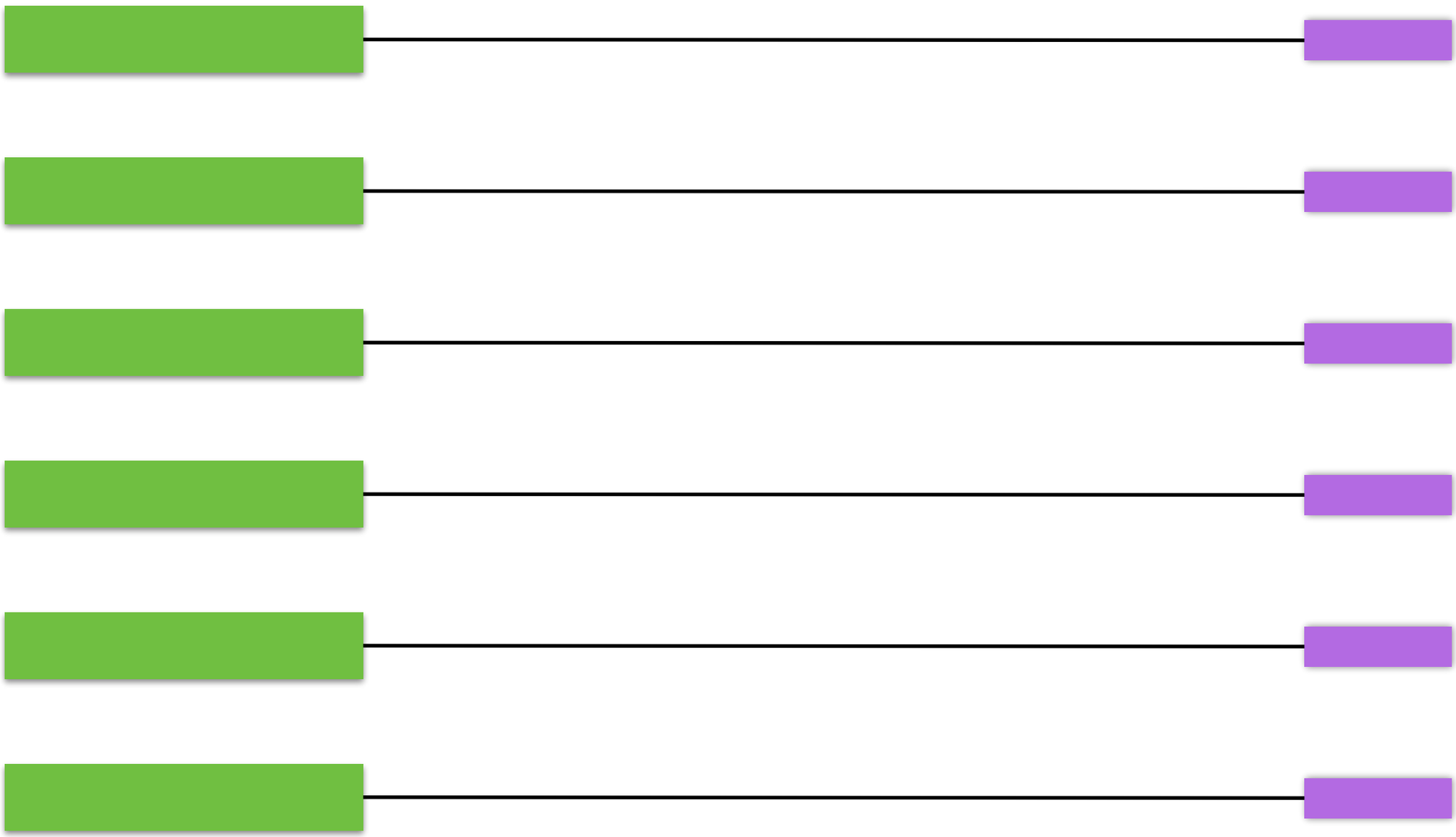
- Flash hostage to individual servers
- Stranded storage and data shuffling among servers
- No enterprise storage features
- Limited capacity

Is there some way to get the best of both?
Without the limitations of either?

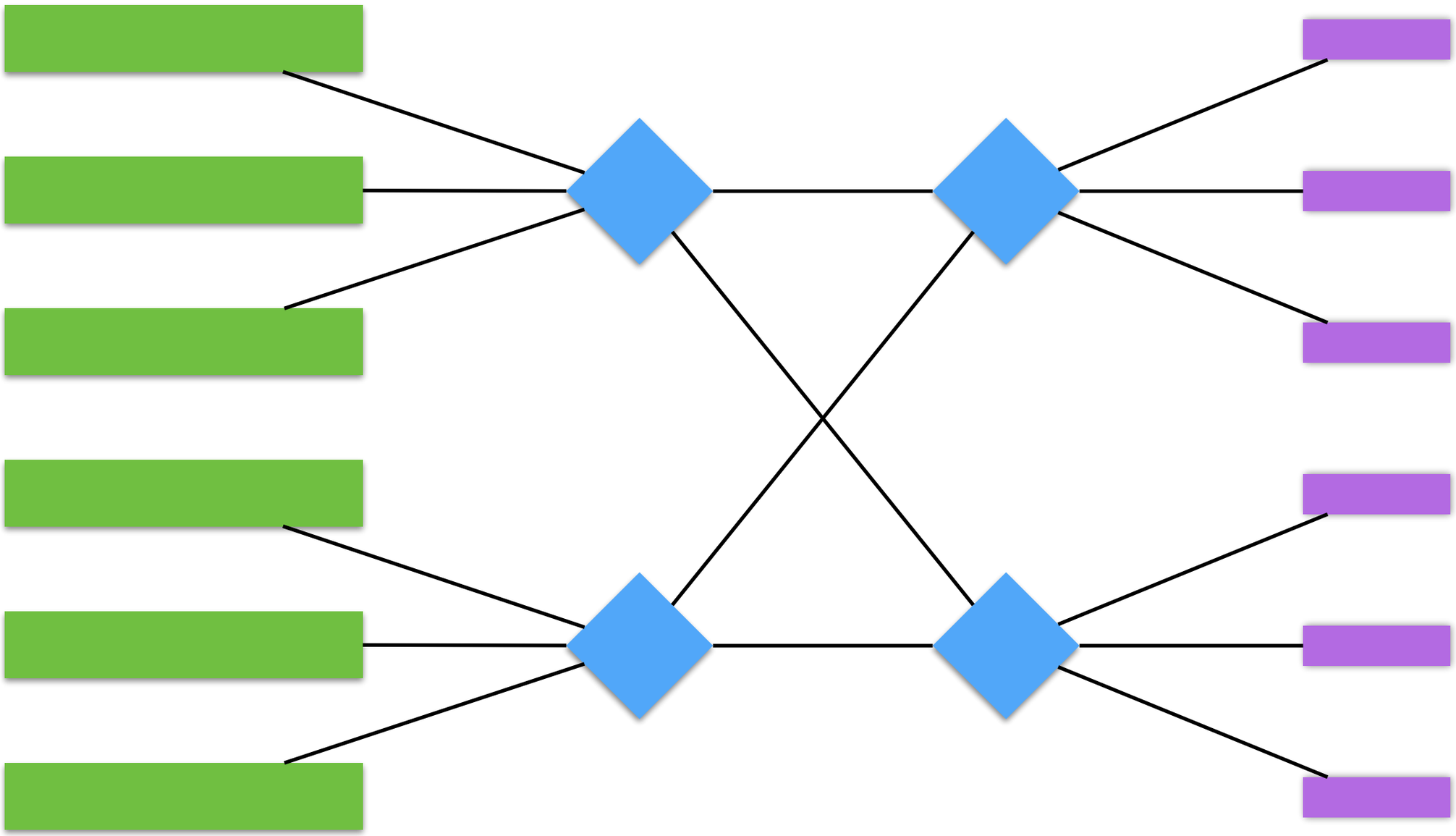
Start with server-attached flash model…



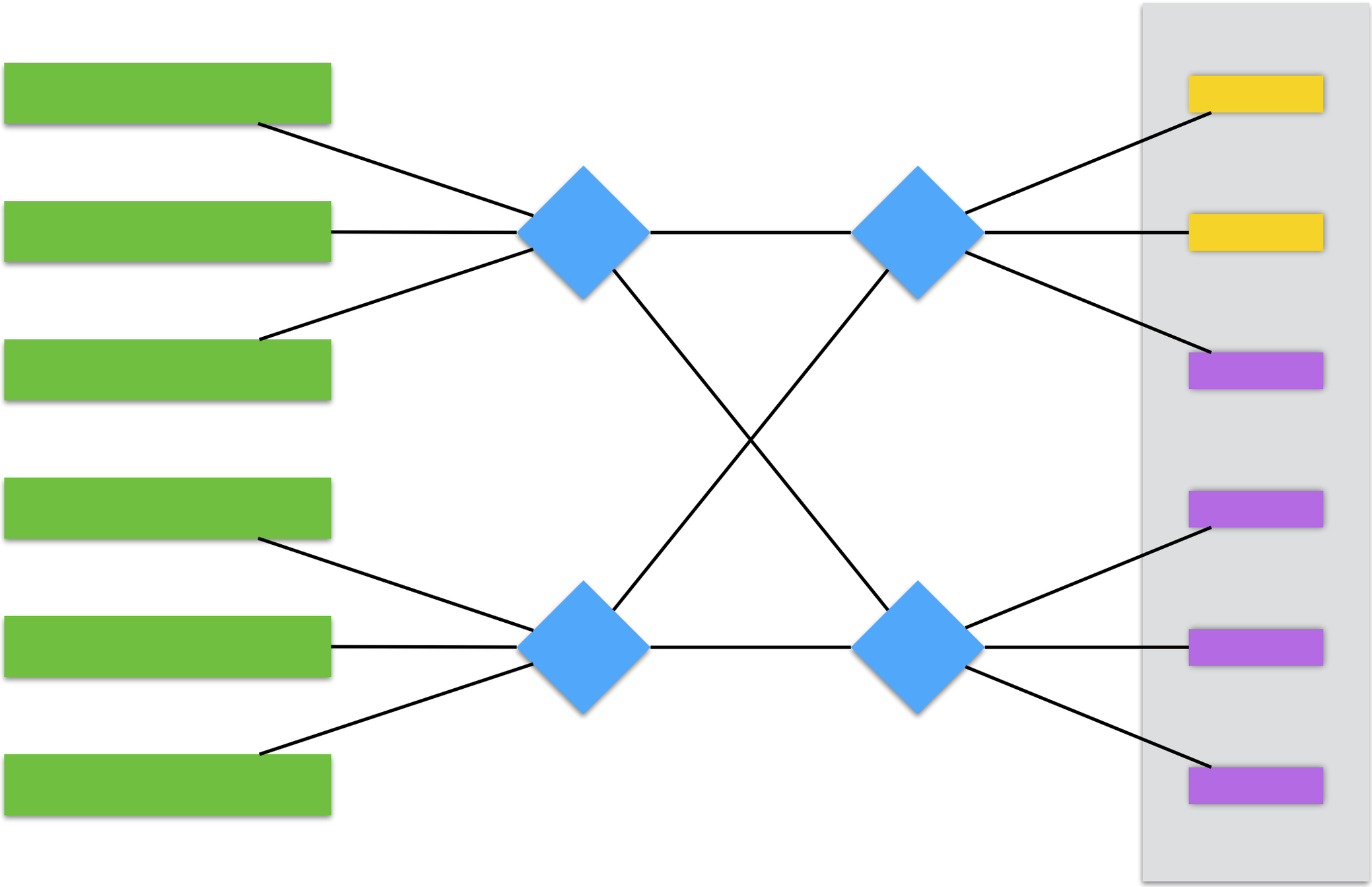
Pull the flash out…



Direct-connect all compute to all storage with a PCIe fabric…



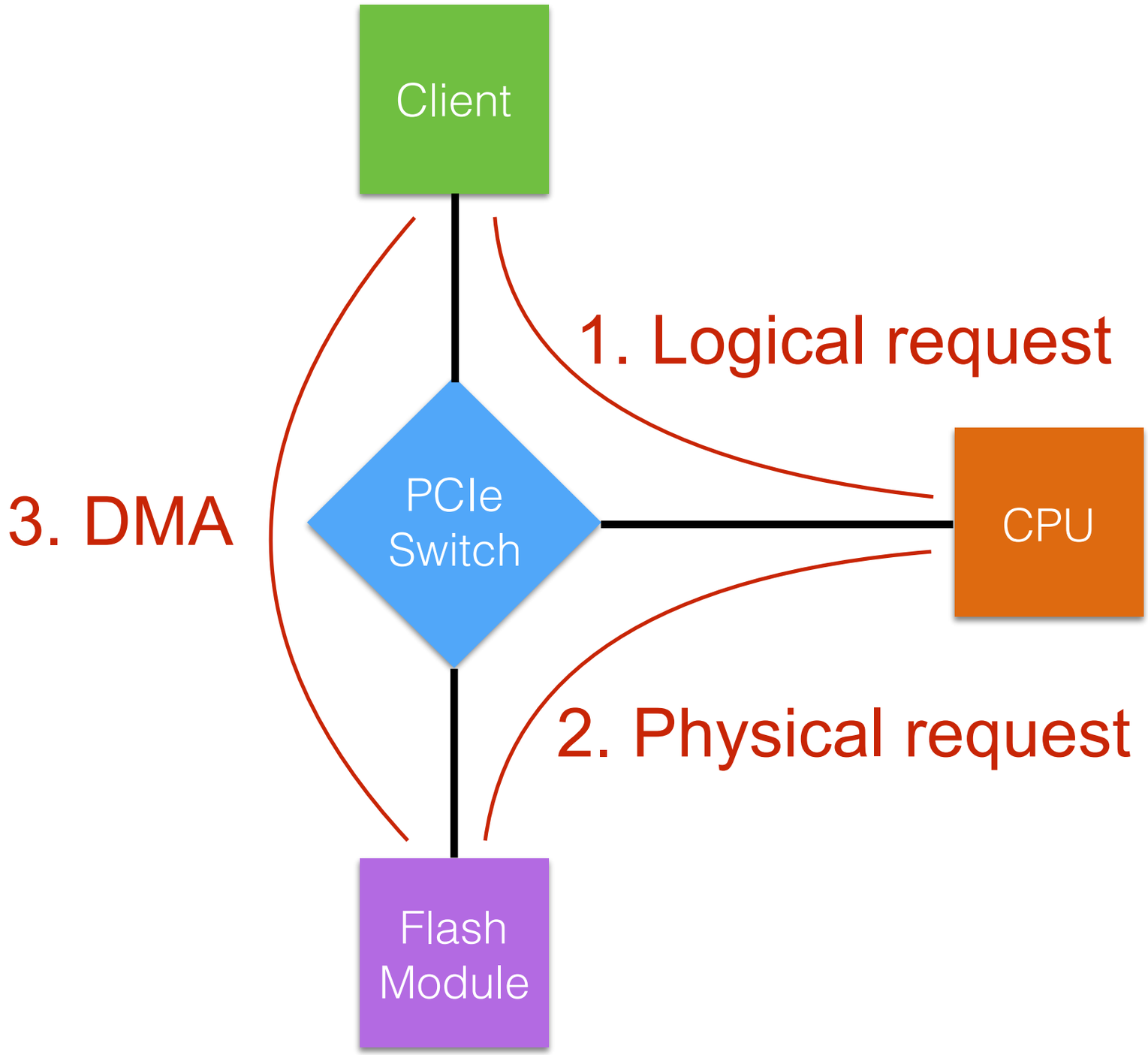
Add enterprise RAID and hot-swap…



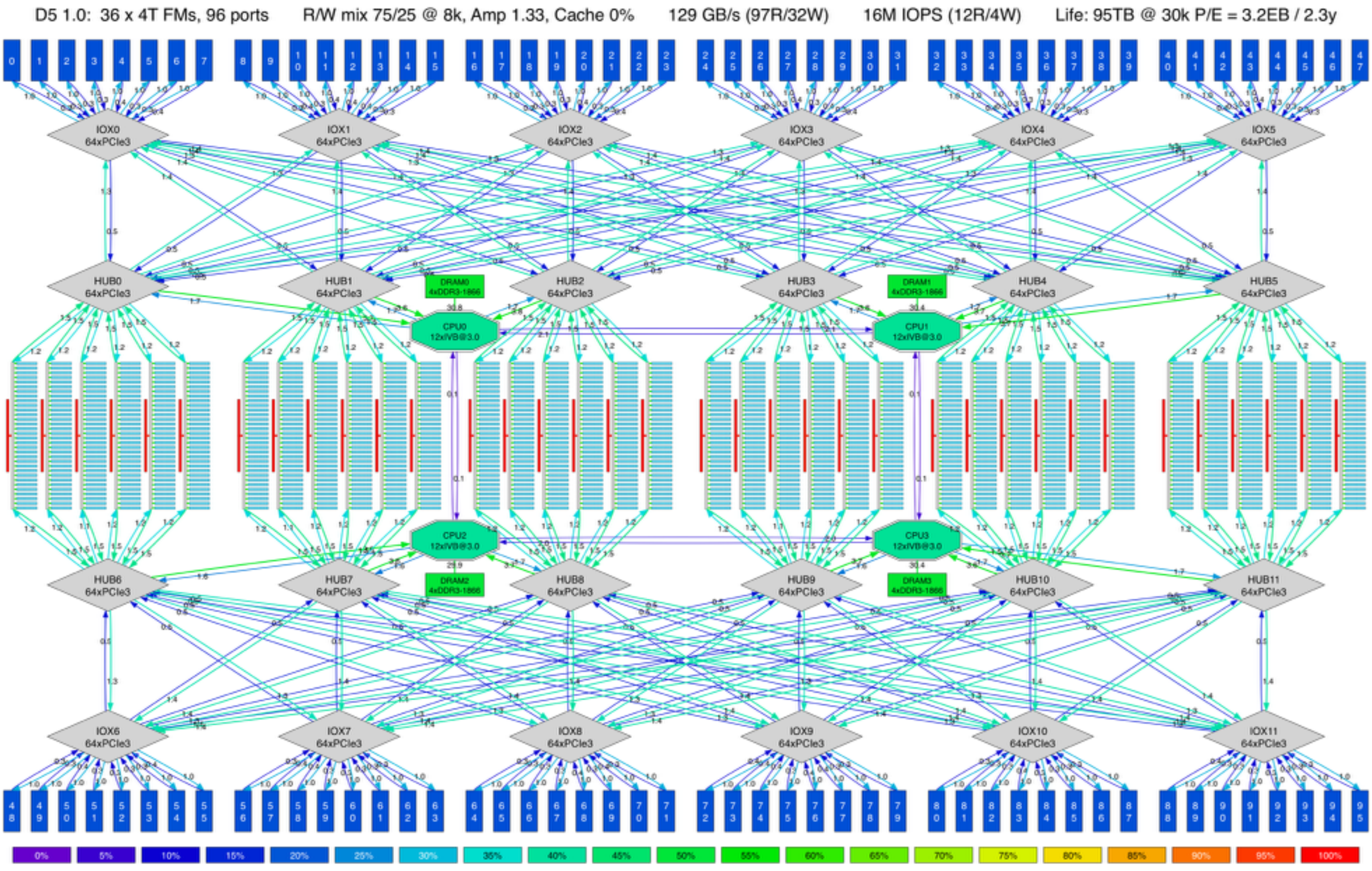
Rack-Scale Flash

- Move the flash out of servers (like AFA)
- Support RAID, dual-porting, etc (like AFA)
- Allow all clients to see all data (like AFA)
- Abandon legacy SCSI stack (like PCIe card)
- Provide much stronger data protection (new)
- Exploit massive NAND concurrency (new)
- Move smarts out of individual FMs (new)

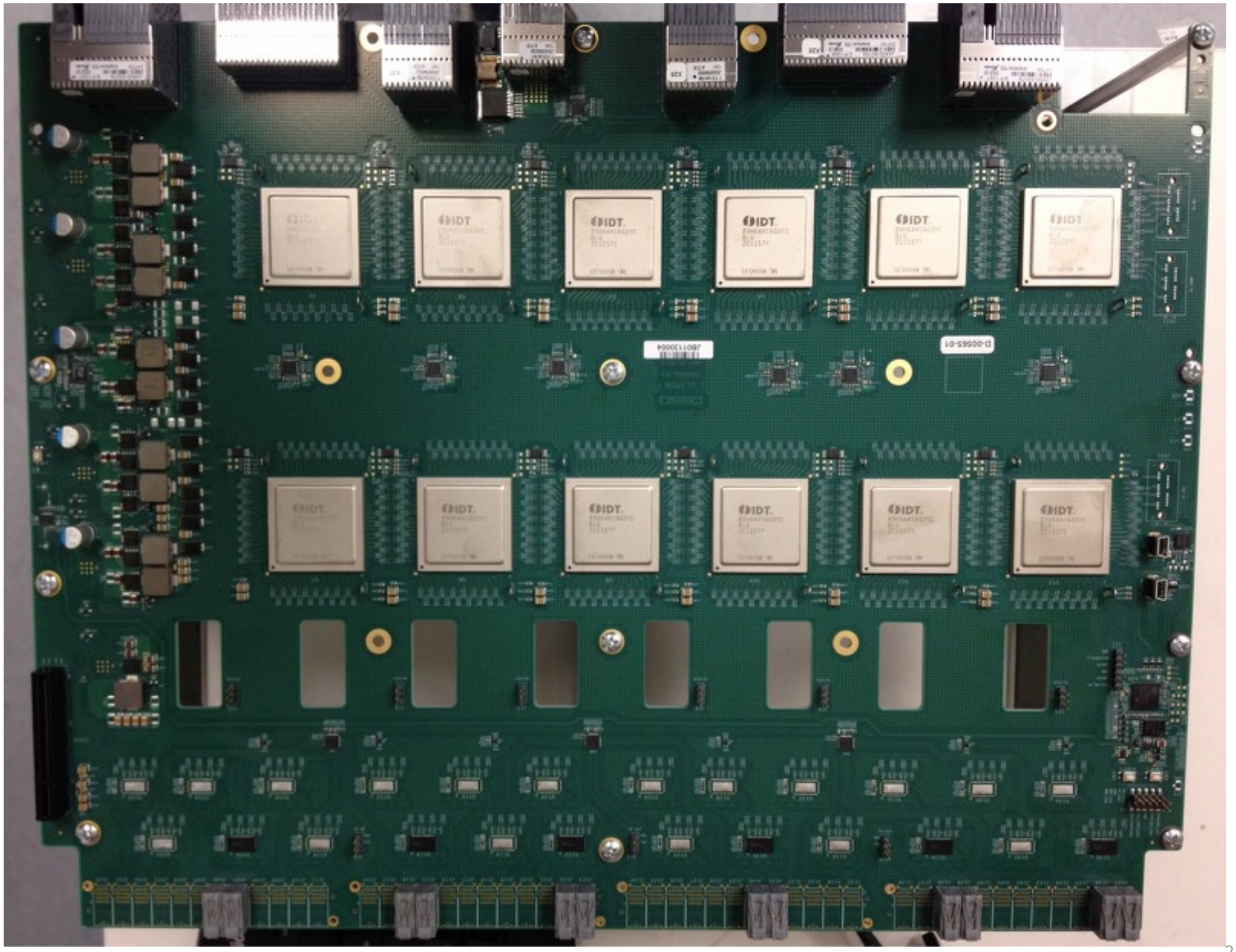
The Basic Design Element



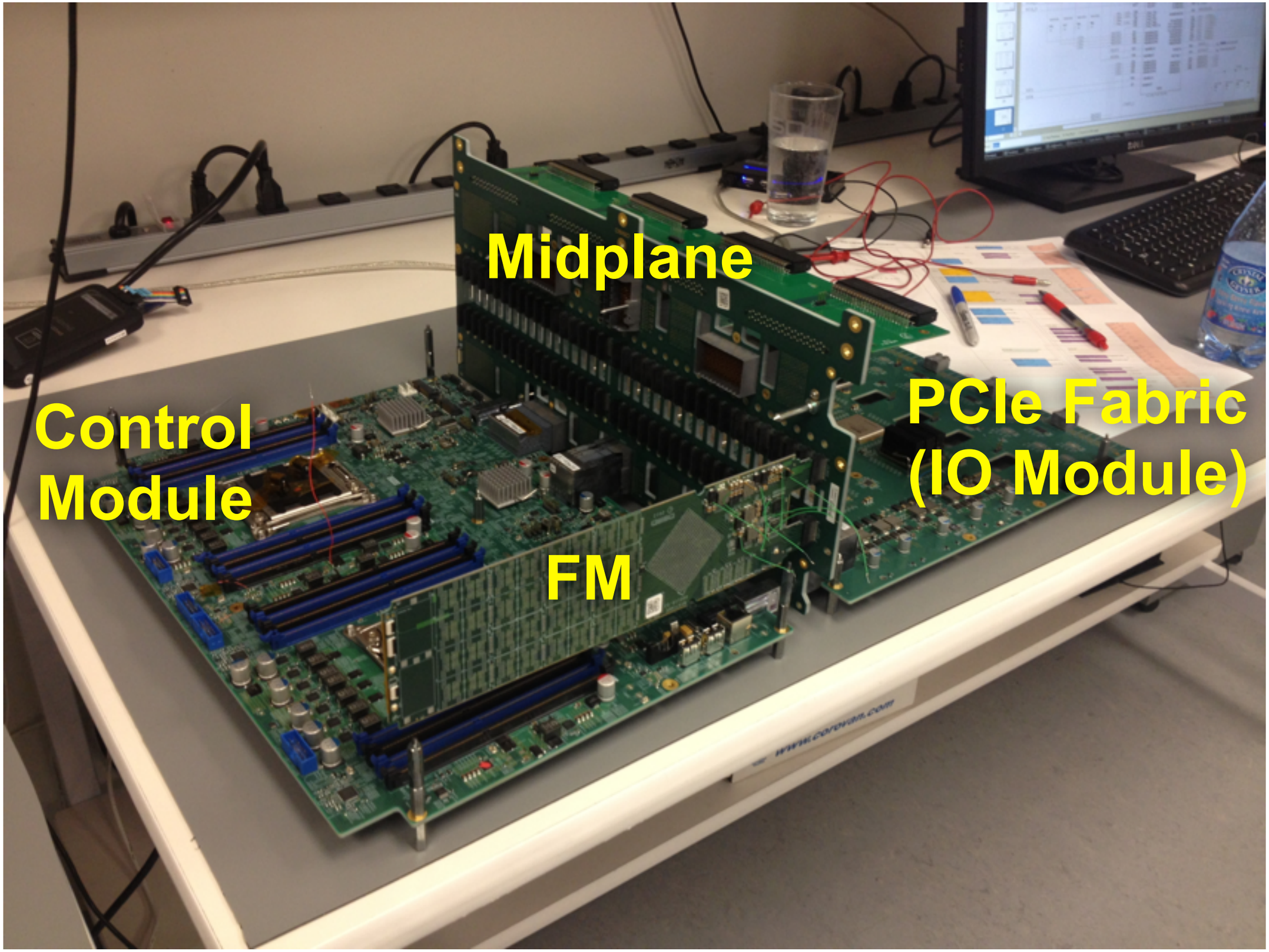
… highly replicated to make D5



D5’s full-mesh PCIe fabric



Anatomy of the D5



D5 Summary

10M IOPS
100 GB/s
100 µsec
100 TB

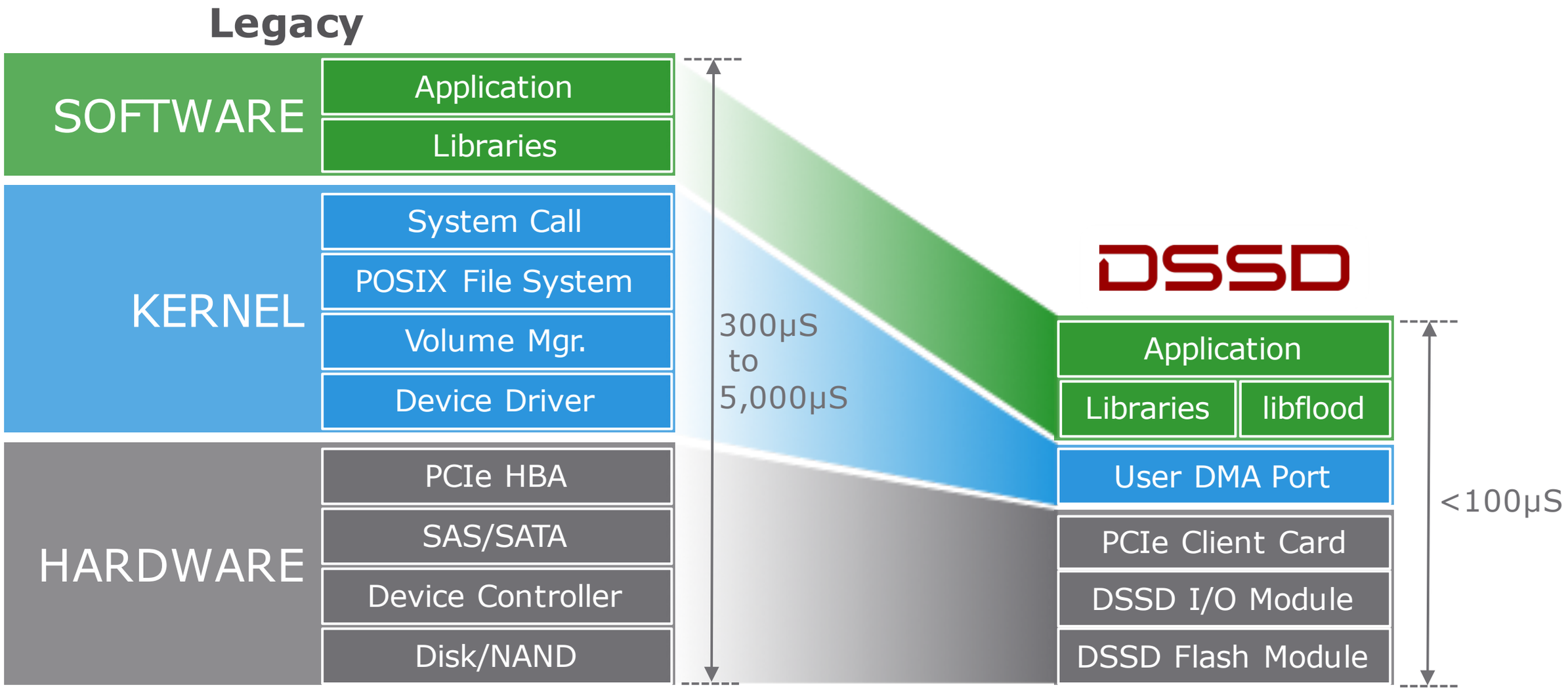
36 flash modules
96 client ports (48 x 2)
Dual hot-swap CM, IOM

Software has to be radically different

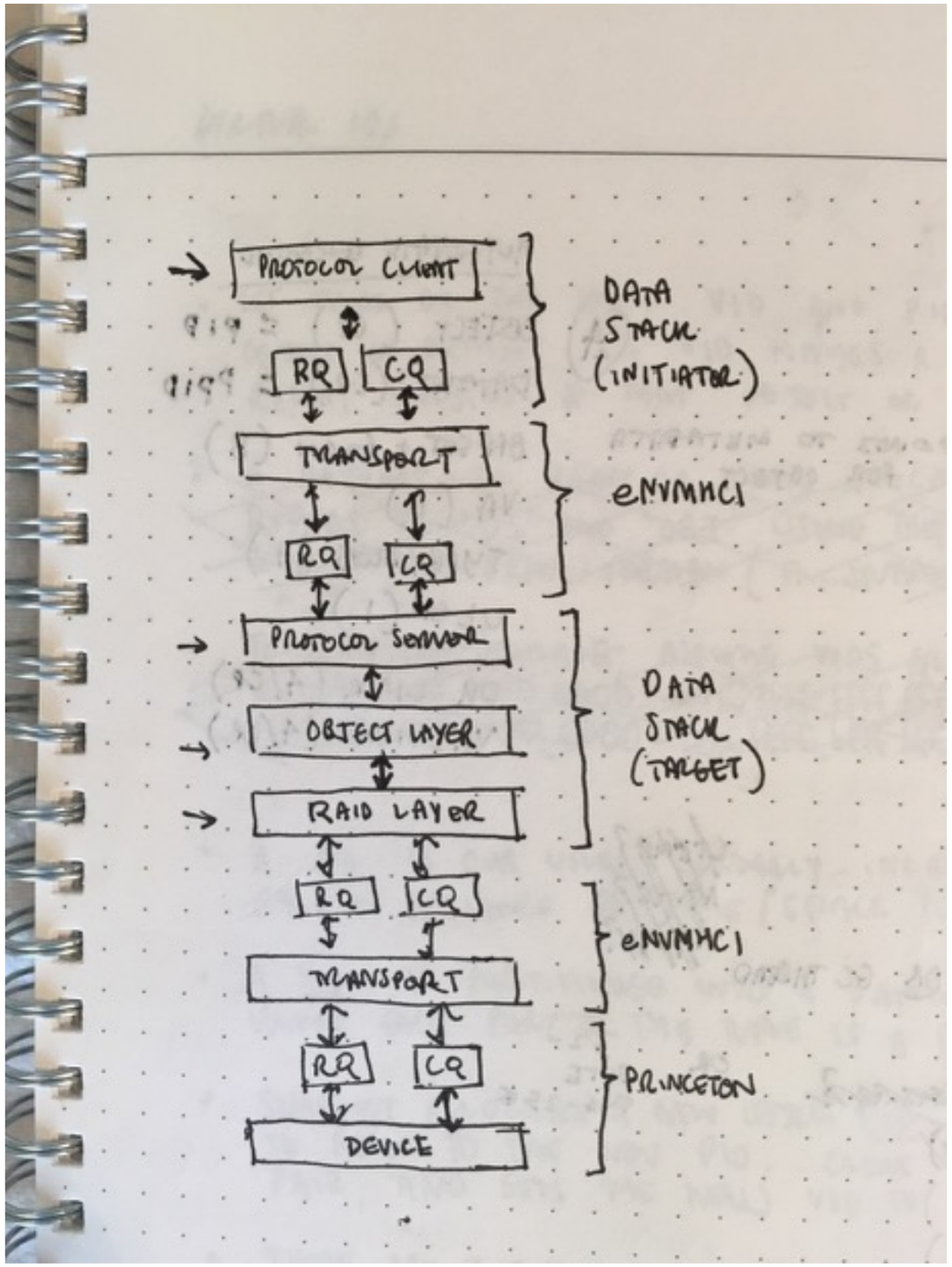
- D5 has 32 CPU cores running I/O
- Those 32 cores drive 10M IOPS
- That's 300k IOPS per core, or
- 3 microseconds per IOP
  - a few thousand instructions,
  - a dozen cache misses,
  - … time's up!
- How can we do an IOP in 3 us?

Eliminate every layer possible

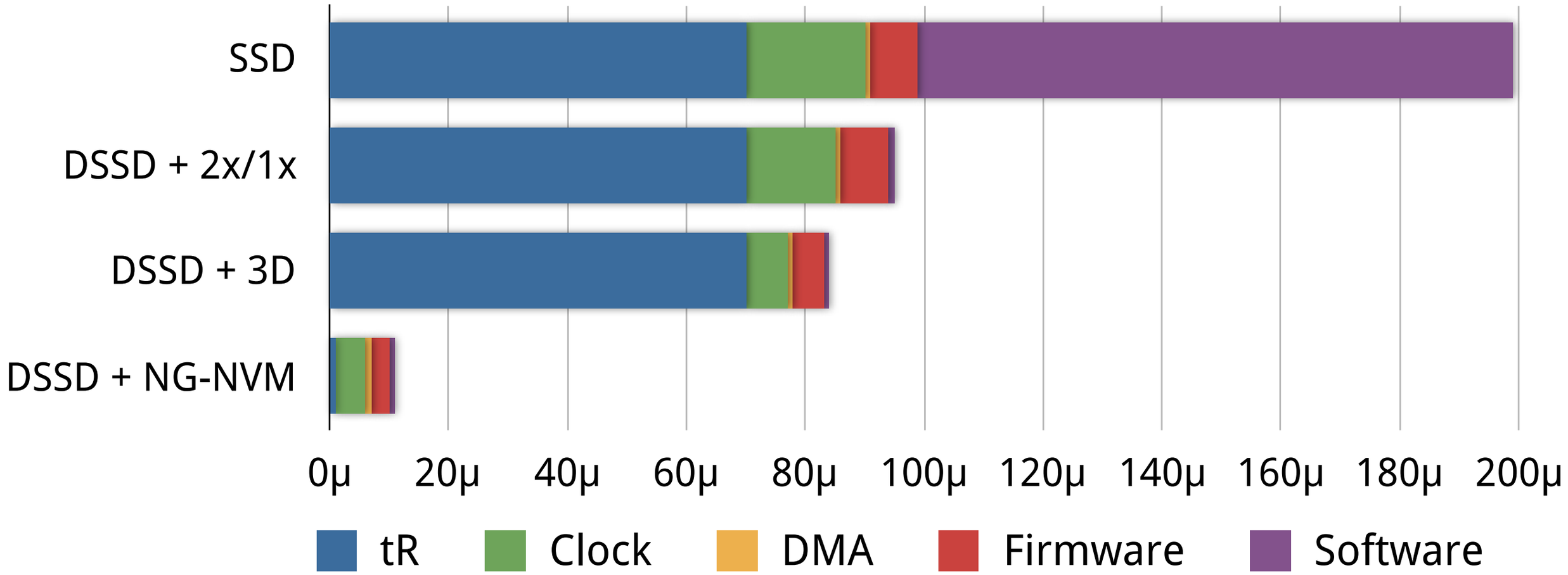
- NVMe SQ/CQ model is a key enabler
- I/O = 64-byte posted write from user space



DSSD software stack, November 2010



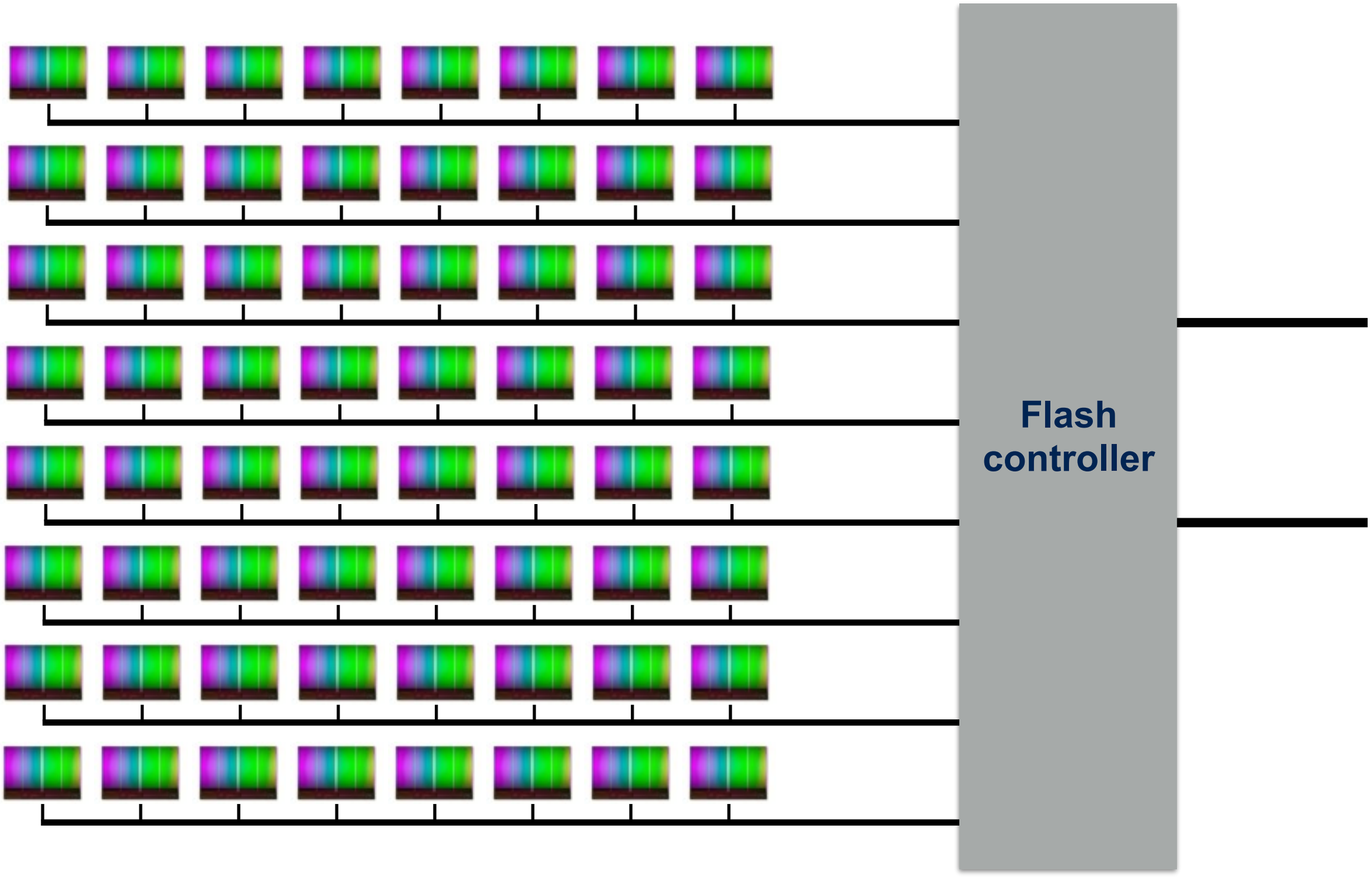
Components of I/O latency



Flash failure modes

- Disk drives fail catastrophically
  - typically the whole drive fails
- Flash degrades slowly over time
  - bit error rate eventually exceeds ECC capacity
  - individual cells fail randomly
- Vastly more discrete devices
  - 18,000 NAND die vs. array of 25 SSDs
  - enables new data protection model

This is SO NOT LIKE a disk drive



Multi-Dimensional RAID

- RAID-6 across channels within each FM
  - Survives channel failure, random page failure
  - Most page-level errors fixed here, inside the FM, so that RAID reconstruction does not involve CM
- RAID-6 across FMs
  - Survives whole-FM failure and field upgrade
  - Repairs errors that channel-level RAID could not
- Layered RAID-of-RAID is pretty good, but we can do much better

1D RAID Reconstruction

- Normal RAID-6 recovers from 1-2 bad blocks

	D0	D1	D2	D3	D4	D5	P6	Q7
D0	0, 0	0, 1	0, 2	0, 3	0, 4	0, 5	0, 6	0, 7
D1	1, 0	1, 1	1, 2	1, 3	1, 4	1, 5	1, 6	1, 7
D2	2, 0	2, 1	2, 2	2, 3	2, 4	2, 5	2, 6	2, 7
P3	3, 0	3, 1	3, 2	3, 3	3, 4	3, 5	3, 6	3, 7
Q4	4, 0	4, 1	4, 2	4, 3	4, 4	4, 5	4, 6	4, 7

1D RAID Reconstruction

- Read of <2, 5> fails.

	D0	D1	D2	D3	D4	D5	P6	Q7
D0	0, 0	0, 1	0, 2	0, 3	0, 4	0, 5	0, 6	0, 7
D1	1, 0	1, 1	1, 2	1, 3	1, 4	1, 5	1, 6	1, 7
D2	2, 0	2, 1	2, 2	2, 3	2, 4	2, 5	2, 6	2, 7
P3	3, 0	3, 1	3, 2	3, 3	3, 4	3, 5	3, 6	3, 7
Q4	4, 0	4, 1	4, 2	4, 3	4, 4	4, 5	4, 6	4, 7

1D RAID Reconstruction

- Read rest of row or column. We'll try row D2.

	D0	D1	D2	D3	D4	D5	P6	Q7
D0	0, 0	0, 1	0, 2	0, 3	0, 4	0, 5	0, 6	0, 7
D1	1, 0	1, 1	1, 2	1, 3	1, 4	1, 5	1, 6	1, 7
D2	2, 0	2, 1	2, 2	2, 3	2, 4	2, 5	2, 6	2, 7
P3	3, 0	3, 1	3, 2	3, 3	3, 4	3, 5	3, 6	3, 7
Q4	4, 0	4, 1	4, 2	4, 3	4, 4	4, 5	4, 6	4, 7

1D RAID Reconstruction

- Fix <2, 5> using double parity. Done.

	D0	D1	D2	D3	D4	D5	P6	Q7
D0	0, 0	0, 1	0, 2	0, 3	0, 4	0, 5	0, 6	0, 7
D1	1, 0	1, 1	1, 2	1, 3	1, 4	1, 5	1, 6	1, 7
D2	2, 0	2, 1	2, 2	2, 3	2, 4	2, 5	2, 6	2, 7
P3	3, 0	3, 1	3, 2	3, 3	3, 4	3, 5	3, 6	3, 7
Q4	4, 0	4, 1	4, 2	4, 3	4, 4	4, 5	4, 6	4, 7

1D RAID Reconstruction

- Normal RAID-6 cannot handle triple failure

	D0	D1	D2	D3	D4	D5	P6	Q7
D0	0, 0	0, 1	0, 2	0, 3	0, 4	0, 5	0, 6	0, 7
D1	1, 0	1, 1	1, 2	1, 3	1, 4	1, 5	1, 6	1, 7
D2	2, 0	2, 1	2, 2	2, 3	2, 4	2, 5	2, 6	2, 7
P3	3, 0	3, 1	3, 2	3, 3	3, 4	3, 5	3, 6	3, 7
Q4	4, 0	4, 1	4, 2	4, 3	4, 4	4, 5	4, 6	4, 7

1D RAID Reconstruction

- Can’t recover <2, 5> using row parity.

	D0	D1	D2	D3	D4	D5	P6	Q7
D0	0, 0	0, 1	0, 2	0, 3	0, 4	0, 5	0, 6	0, 7
D1	1, 0	1, 1	1, 2	1, 3	1, 4	1, 5	1, 6	1, 7
D2	2, 0	2, 1	2, 2	2, 3	2, 4	2, 5	2, 6	2, 7
P3	3, 0	3, 1	3, 2	3, 3	3, 4	3, 5	3, 6	3, 7
Q4	4, 0	4, 1	4, 2	4, 3	4, 4	4, 5	4, 6	4, 7

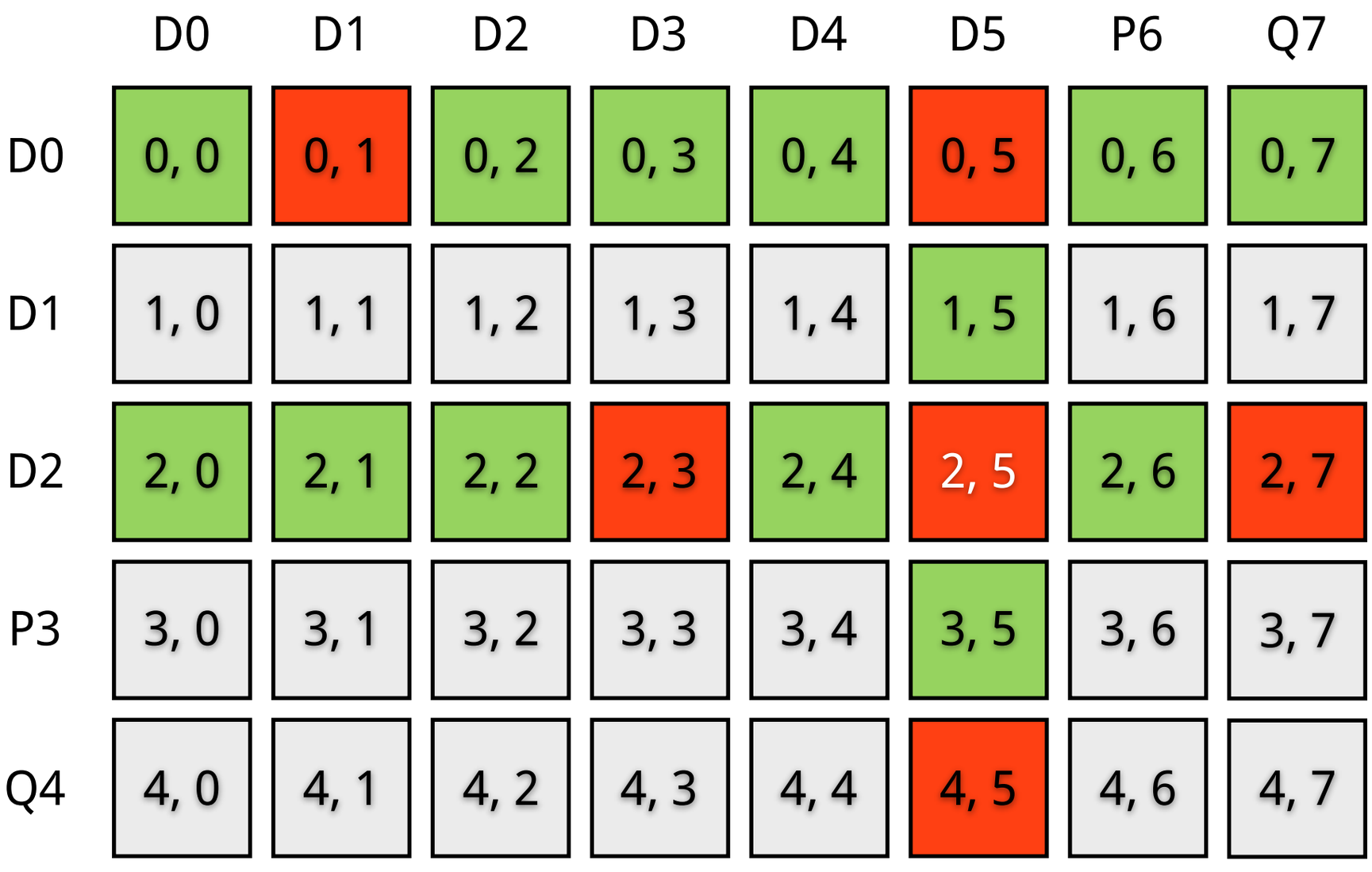
1D RAID Reconstruction

- Can’t recover <2, 5> using column parity.

	D0	D1	D2	D3	D4	D5	P6	Q7
D0	0, 0	0, 1	0, 2	0, 3	0, 4	0, 5	0, 6	0, 7
D1	1, 0	1, 1	1, 2	1, 3	1, 4	1, 5	1, 6	1, 7
D2	2, 0	2, 1	2, 2	2, 3	2, 4	2, 5	2, 6	2, 7
P3	3, 0	3, 1	3, 2	3, 3	3, 4	3, 5	3, 6	3, 7
Q4	4, 0	4, 1	4, 2	4, 3	4, 4	4, 5	4, 6	4, 7

2D RAID Grid Reconstruction

- 2D RAID **can** recover <2, 5>. First, get row D0.



2D RAID Grid Reconstruction

- Fix row D0 using row parity.

	D0	D1	D2	D3	D4	D5	P6	Q7
D0	0, 0	0, 1	0, 2	0, 3	0, 4	0, 5	0, 6	0, 7
D1	1, 0	1, 1	1, 2	1, 3	1, 4	1, 5	1, 6	1, 7
D2	2, 0	2, 1	2, 2	2, 3	2, 4	2, 5	2, 6	2, 7
P3	3, 0	3, 1	3, 2	3, 3	3, 4	3, 5	3, 6	3, 7
Q4	4, 0	4, 1	4, 2	4, 3	4, 4	4, 5	4, 6	4, 7

2D RAID Grid Reconstruction

- Fix column D5. We now have <2, 5>. Done.

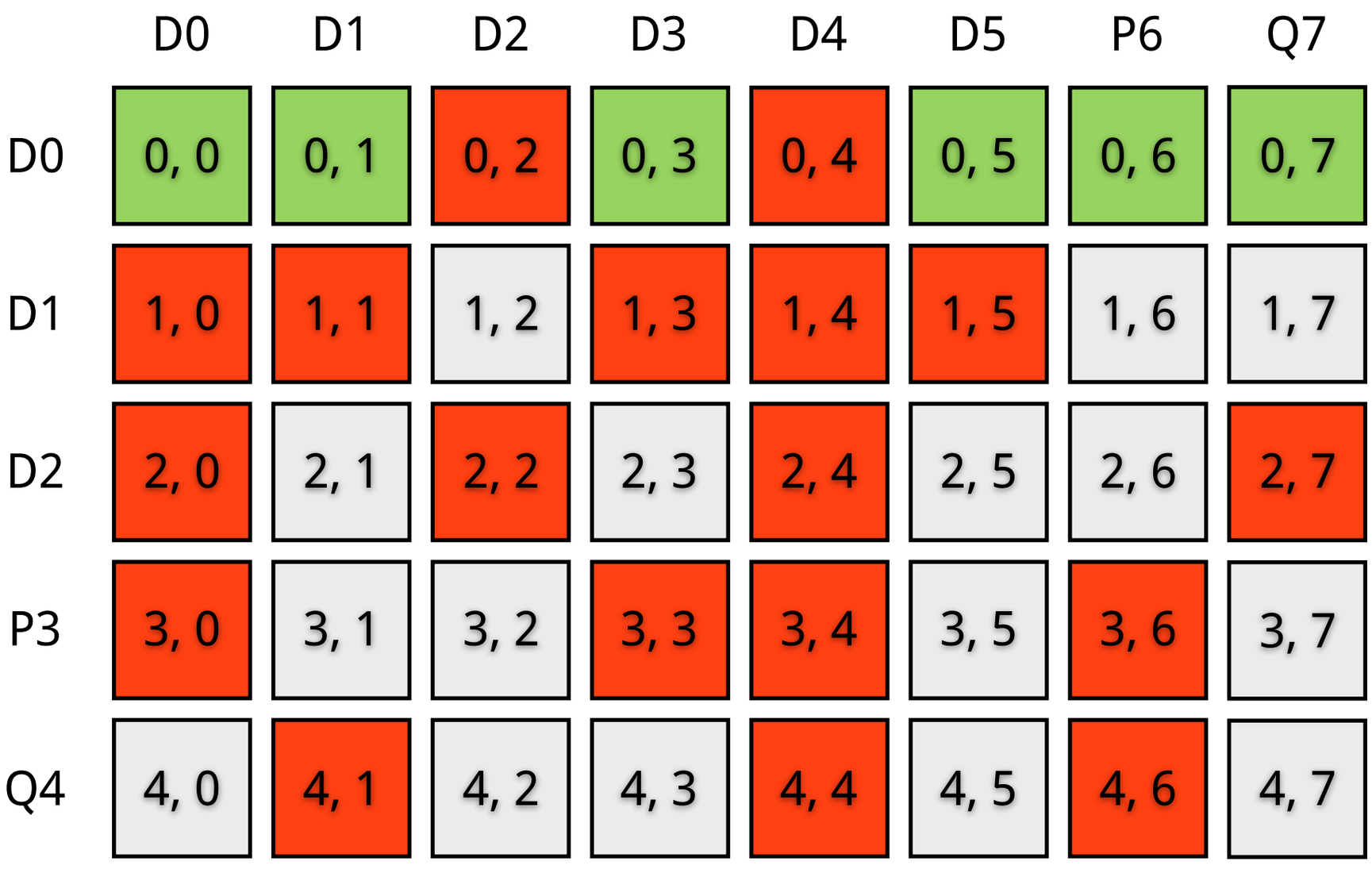
2D RAID Grid Reconstruction

- Incredibly powerful. This is recoverable:

	D0	D1	D2	D3	D4	D5	P6	Q7
D0	0, 0	0, 1	0, 2	0, 3	0, 4	0, 5	0, 6	0, 7
D1	1, 0	1, 1	1, 2	1, 3	1, 4	1, 5	1, 6	1, 7
D2	2, 0	2, 1	2, 2	2, 3	2, 4	2, 5	2, 6	2, 7
P3	3, 0	3, 1	3, 2	3, 3	3, 4	3, 5	3, 6	3, 7
Q4	4, 0	4, 1	4, 2	4, 3	4, 4	4, 5	4, 6	4, 7

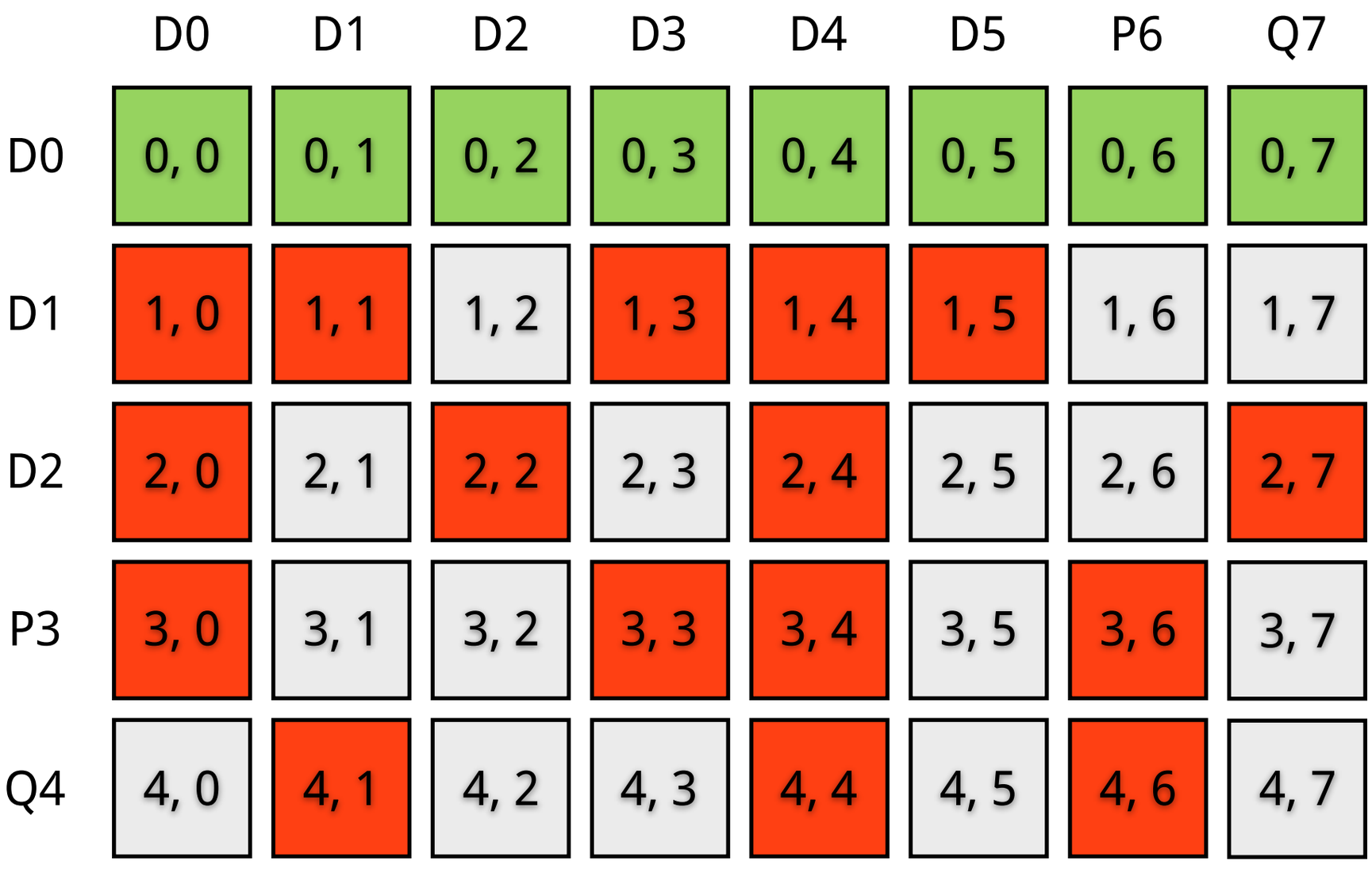
2D RAID Grid Reconstruction

- Read row D0.



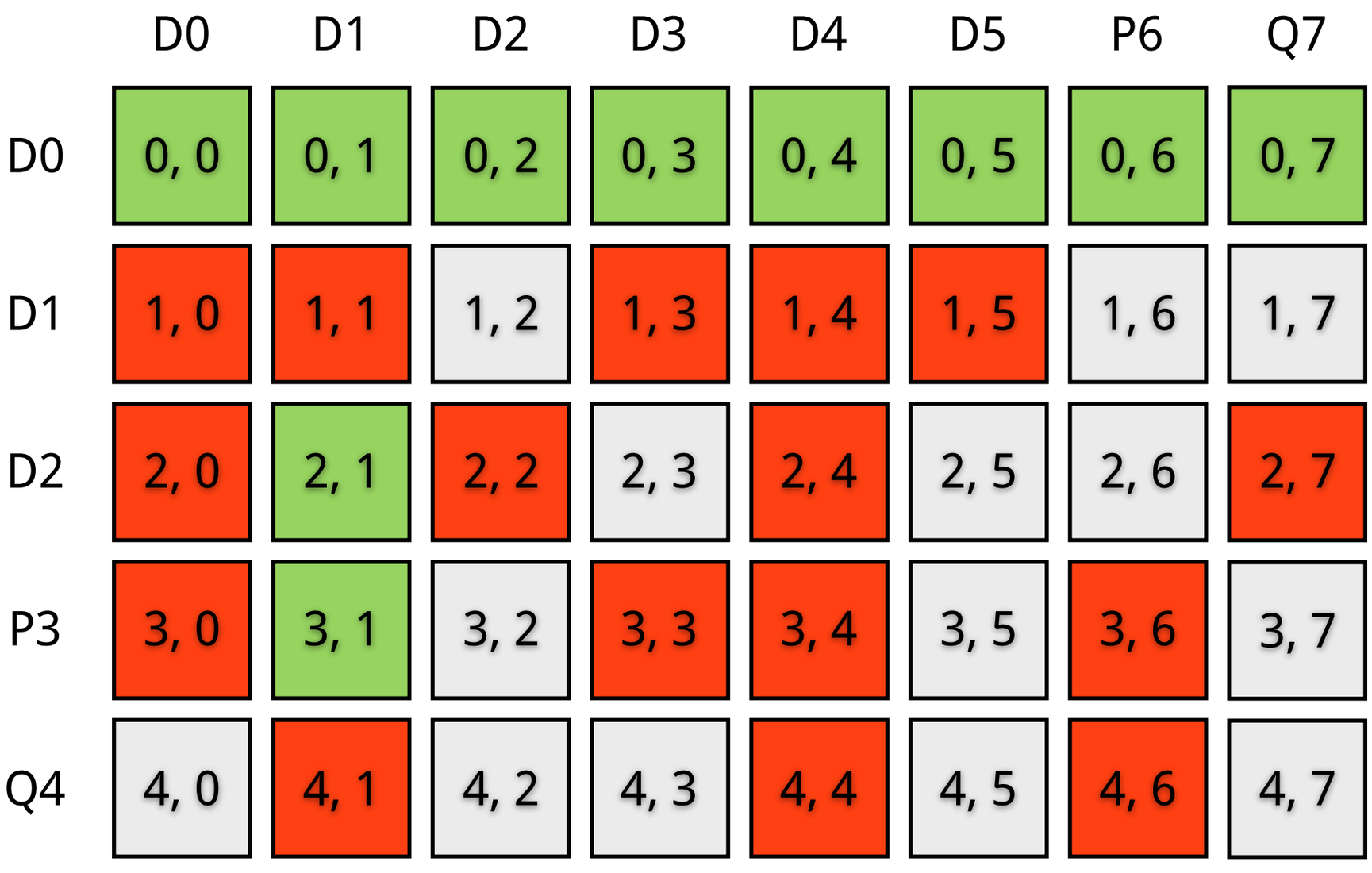
2D RAID Grid Reconstruction

- Fix row D0. No other row can be fixed.



2D RAID Grid Reconstruction

- Read column D1.



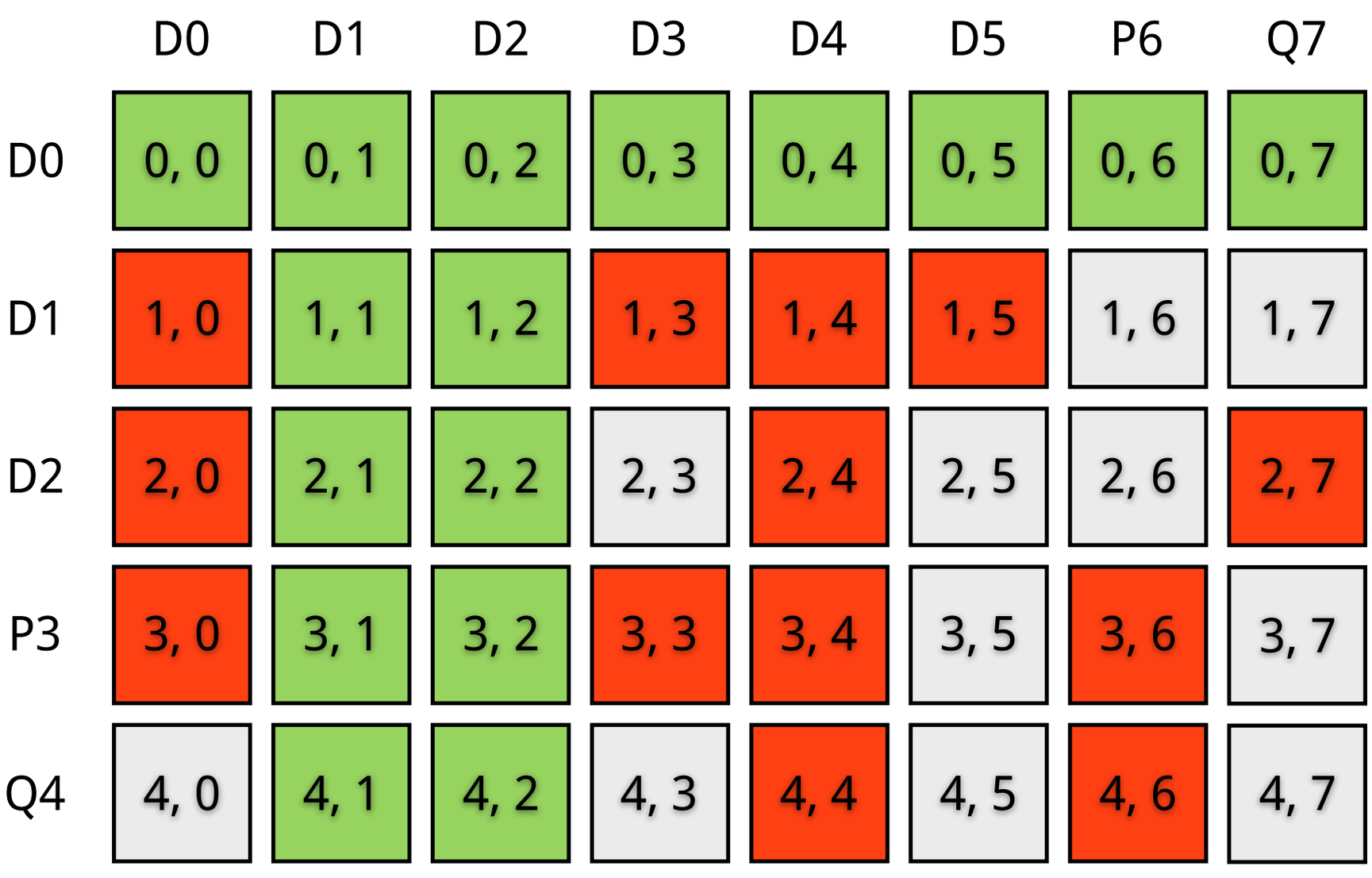
2D RAID Grid Reconstruction

- Fix column D1.

	D0	D1	D2	D3	D4	D5	P6	Q7
D0	0, 0	0, 1	0, 2	0, 3	0, 4	0, 5	0, 6	0, 7
D1	1, 0	1, 1	1, 2	1, 3	1, 4	1, 5	1, 6	1, 7
D2	2, 0	2, 1	2, 2	2, 3	2, 4	2, 5	2, 6	2, 7
P3	3, 0	3, 1	3, 2	3, 3	3, 4	3, 5	3, 6	3, 7
Q4	4, 0	4, 1	4, 2	4, 3	4, 4	4, 5	4, 6	4, 7

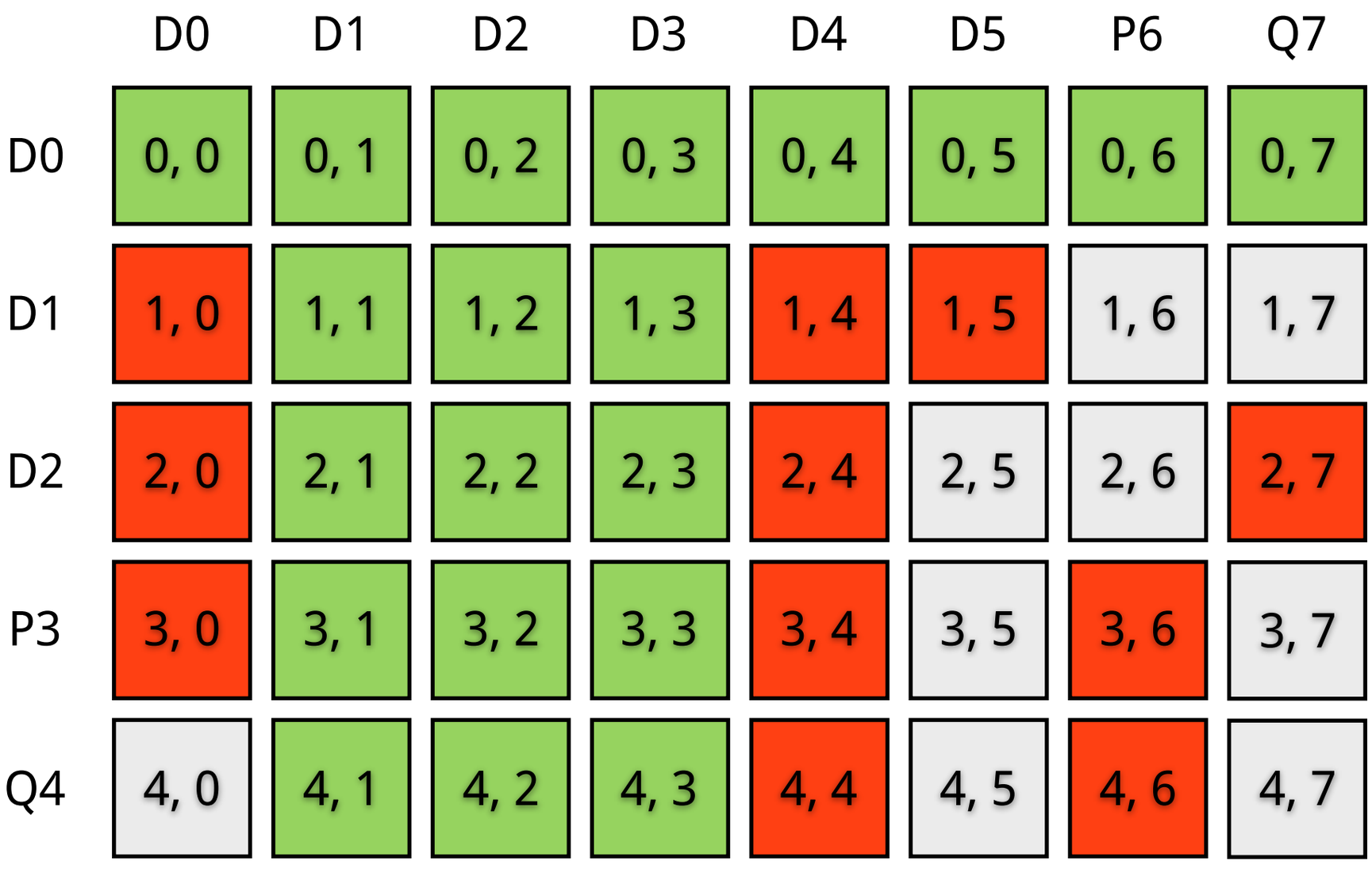
2D RAID Grid Reconstruction

- Read and fix column D2.



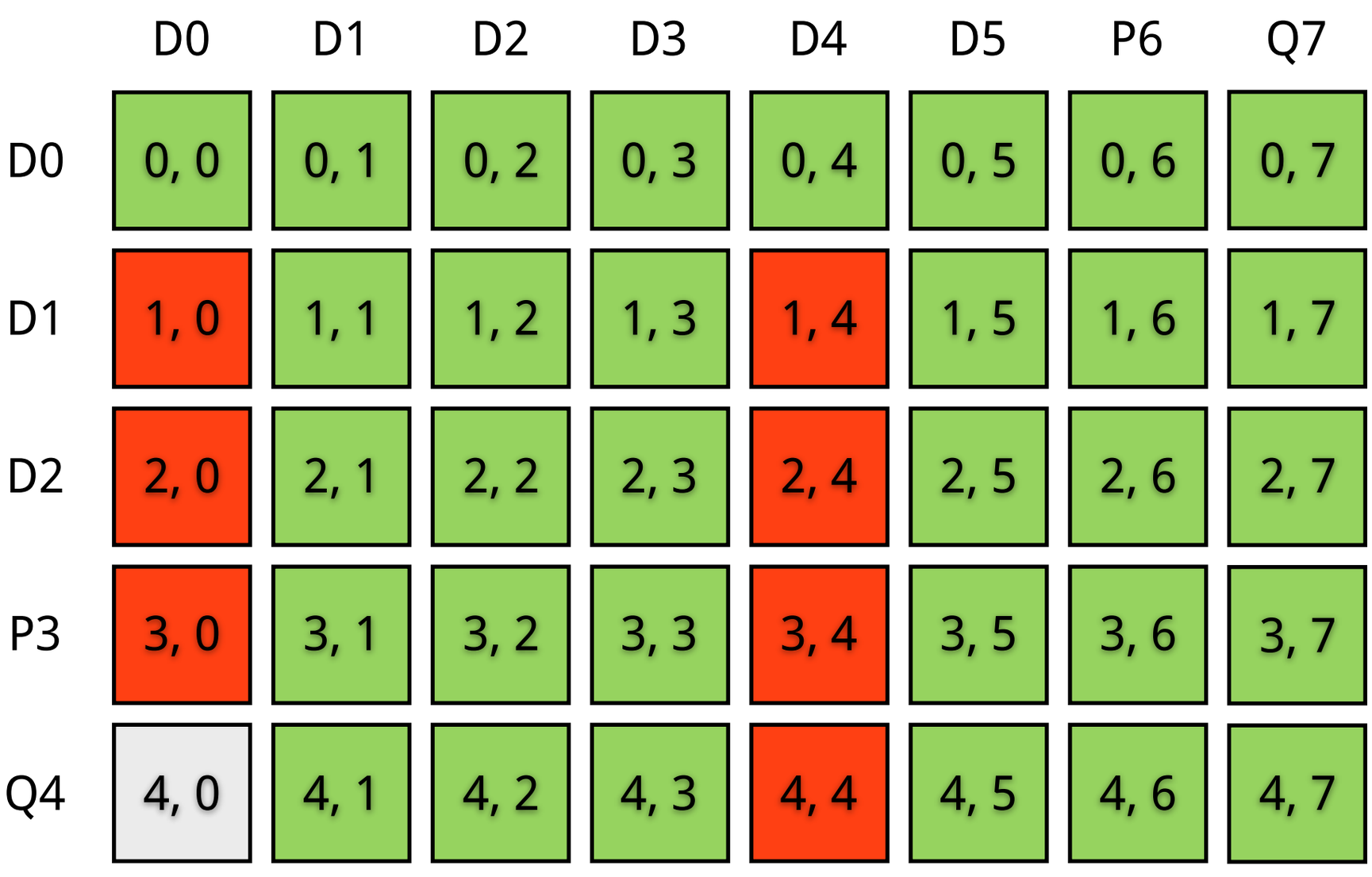
2D RAID Grid Reconstruction

- Read and fix column D3.



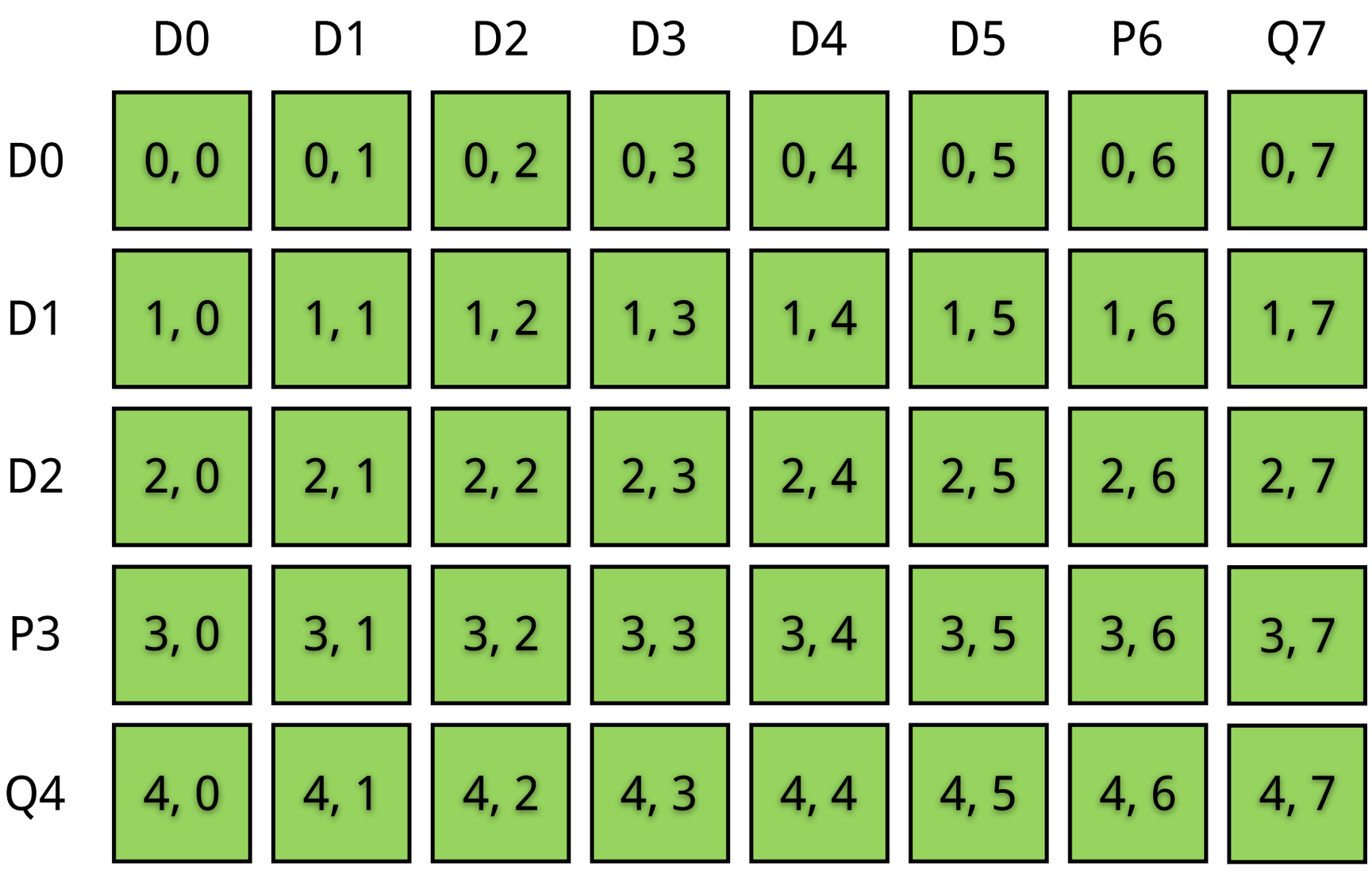
2D RAID Grid Reconstruction

- Read and fix columns D5, P6, Q7.



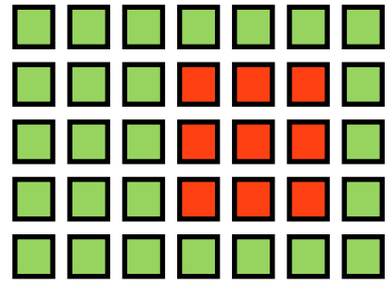
2D RAID Grid Reconstruction

- Fix rows D1, D2, D3, Q4.



Multi-Dimensional RAID Properties

- For D-dimensional RAID with M-way parity in each dimension, the smallest unsolvable failure is a perfect grid of $(M+1)^D$ points



  - Much larger asymmetric failures are solvable

- By contrast, linear RAID with the same space overhead (M•D) cannot solve anything > M•D

- Multi-dimensional RAID takes advantage of the inherent sparseness of higher-dimensional space

- Algorithm must be explicitly multi-dimensional: naive RAID-of-RAID layering can only solve M•D

2D RAID Mathematical Properties

Parity equations:

$$P = \sum_x D_x \qquad Q = \sum_x D_x \bullet g^x \qquad R = \sum_x D_x \bullet g^{2x}$$

More generally,

$$P_d = \sum_x D_x \bullet g^{dx}$$

In an xy grid,

$$\text{Row y } P_d = \sum_x D_{xy} \bullet g^{dx}$$

$$\text{Col x } P_e = \sum_y D_{xy} \bullet g^{ey}$$

Parity of parity:

$$\text{Row } P_d \text{ of Col } P_e = \sum_x \left(\sum_y D_{xy} \bullet g^{ey}\right) \bullet g^{dx}$$

$$\text{Col } P_e \text{ of Row } P_d = \sum_y \left(\sum_x D_{xy} \bullet g^{dx}\right) \bullet g^{ey}$$

Same answer either way!

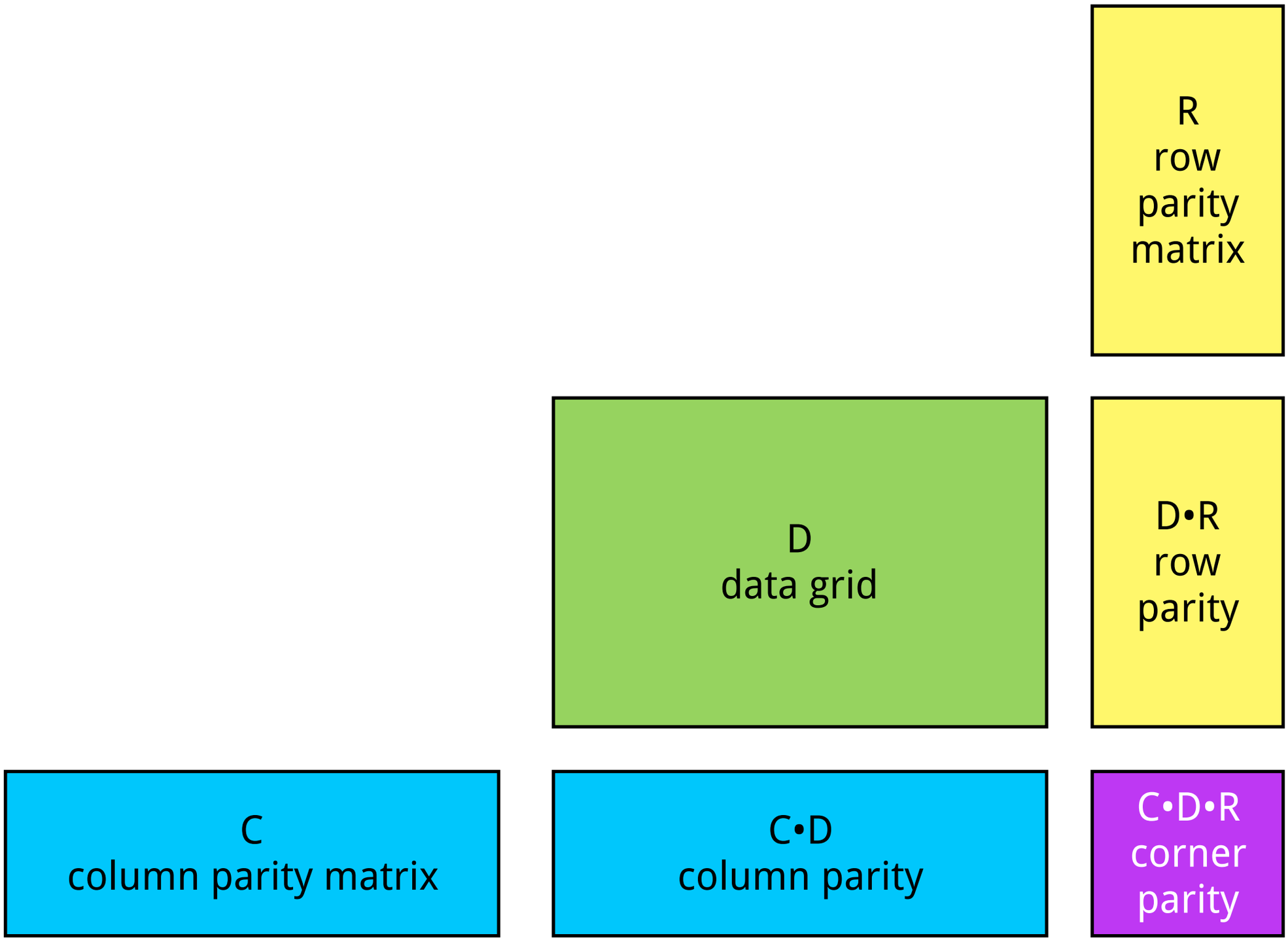
$$P_{de} = \sum_{x,y} D_{xy} \bullet g^{dx+ey}$$

Therefore, P_{de} can be used to solve both rows and columns

2D RAID Parity Grid

D_{00}	D_{10}	D_{20}	$\sum_x D_{x0}$	$\sum_x D_{x0} \cdot g^x$
D_{01}	D_{11}	D_{21}	$\sum_x D_{x1}$	$\sum_x D_{x1} \cdot g^x$
D_{02}	D_{12}	D_{22}	$\sum_x D_{x2}$	$\sum_x D_{x2} \cdot g^x$
D_{03}	D_{13}	D_{23}	$\sum_x D_{x3}$	$\sum_x D_{x3} \cdot g^x$
$\sum_y D_{0y}$	$\sum_y D_{1y}$	$\sum_y D_{2y}$	$\sum_{x,y} D_{xy}$	$\sum_{x,y} D_{xy} \cdot g^x$
$\sum_y D_{0y} \cdot g^y$	$\sum_y D_{1y} \cdot g^y$	$\sum_y D_{2y} \cdot g^y$	$\sum_{x,y} D_{xy} \cdot g^y$	$\sum_{x,y} D_{xy} \cdot g^{x+y}$

2D RAID Parity Linear Algebra



2D Parity Space: 1 - (16/18)•(30/32) = 17%