

Taming upmap sprawl at Proton

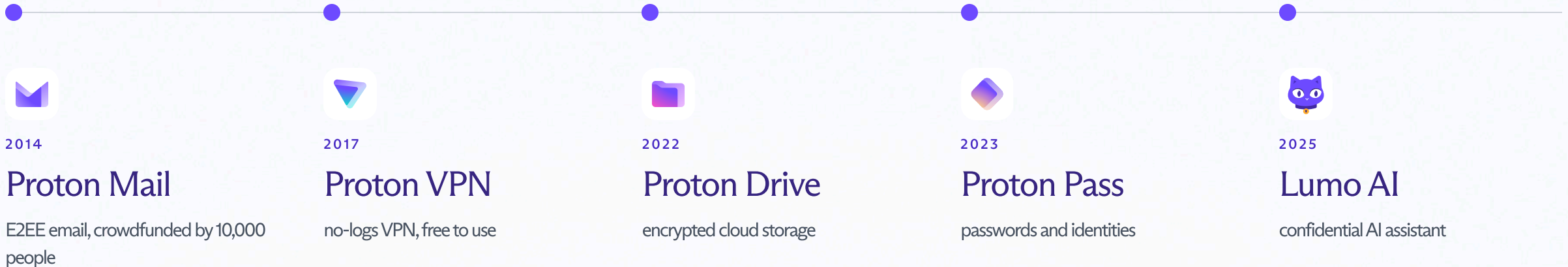
Graph-driven PG rebalancing at petabyte scale

Dr. Jan Veverka

Ceph Days Zürich 2026

Proton: privacy by default

100M+ accounts · 600+ staff, 50+ nationalities · Geneva HQ, office in Zürich · 100,000+ organisations



From 1 PB to 500+ PB in eight years

2018

1 PB

8 servers, EC 4+2, off MySQL

2024

100+ PB

3 main clusters, 6,000+ OSDs

2026

500+ PB

7 main clusters, 15,000+ OSDs

The three worst ways upmaps hurt operations

01.

Slow ops

They become more frequent and precipitated by less stress.

02.

Blast radius

Greatly amplified by upmap invalidation: up to 5x and more.

03.

Monitor disk space

Pressure on the monitors, from large OSD maps and more frequent epochs.

Upmaps only pile up

All of them add to `pg_upmap_items`; none looks at the mappings already there as a whole.



Ceph's balancer

In upmap mode.



JJ's Ceph Balancer

The external one, by Jonas Jelten.



Cancelled backfills

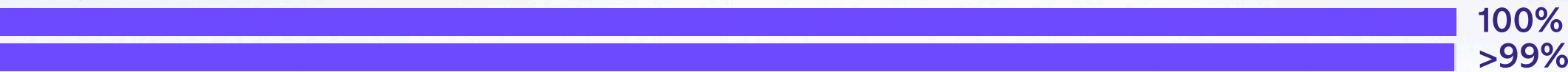
After PG splits, cluster growth and disk failures.

How many PGs have upmaps?

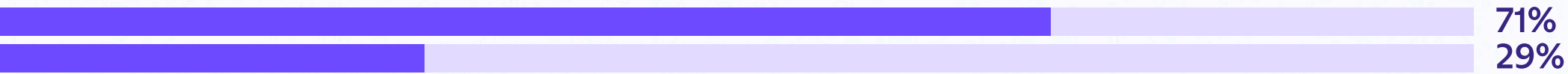
HDD, EC 8+3



HDD, EC 4+2



SSD, 3x



How many PGs have upmaps?

HDD, EC 8+3

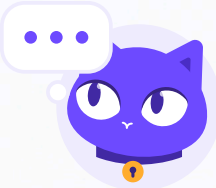


HDD, EC 4+2



SSD, 3x

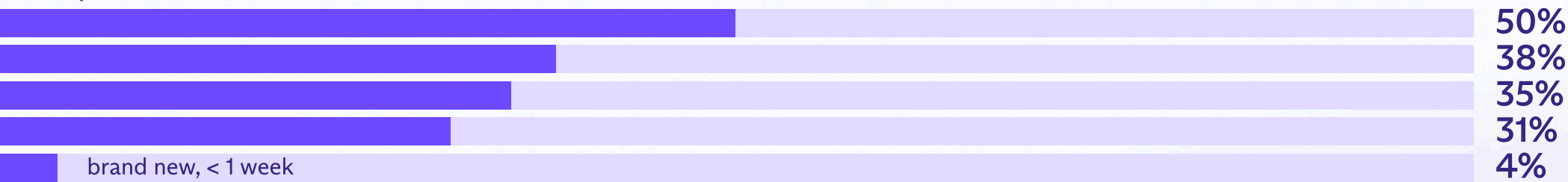


90%+  in 6 of Proton's 9 largest pools

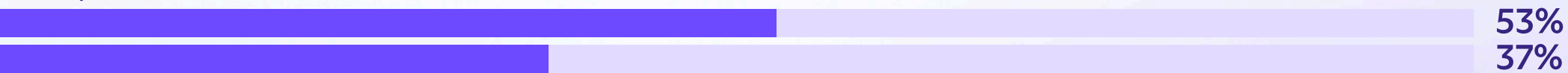
Source: 9 largest Ceph pools, September–October 2026

How many PG shards and replicas are mapped?

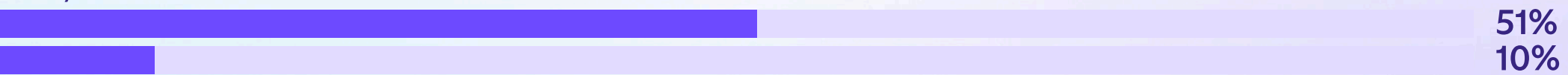
HDD, EC 8+3



HDD, EC 4+2

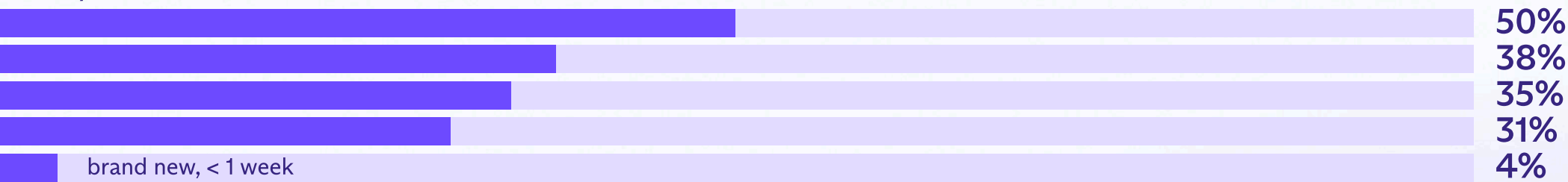


SSD, 3x

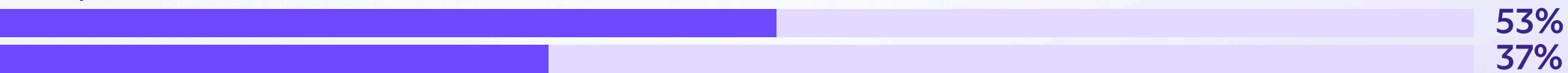


How many PG shards and replicas are mapped?

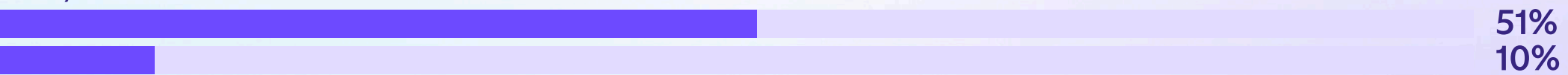
HDD, EC 8+3



HDD, EC 4+2



SSD, 3x



Up to 53% 

Remove upmaps, but gracefully

Removing upmaps takes one command per PG. The challenge is to keep the rest running as it was:



OSD balance

Utilization stays as even as the balancers left it.



Fragmentation

No more free-space fragmentation on the OSDs.



Production workloads

Clients don't notice the data moving.

The breakthrough

Upmaps as a directed multigraph

```
$ ceph osd dump | grep upmap | head -n3  
pg_upmap_items 1.0 [1128,475,2013,179]  
pg_upmap_items 2.1 [1780,1334,2598,2306,2819,930,2164,2998]  
pg_upmap_items 2.2 [1213,2943,2048,2992,230,435]
```

Upmaps as a directed multigraph

```
$ ceph osd dump | grep upmap | head -n3  
pg_upmap_items 1.0 [1128,475,2013,179]  
pg_upmap_items 2.1 [1780,1334,2598,2306,2819,930,2164,2998]  
pg_upmap_items 2.2 [1213,2943,2048,2992,230,435]
```



PG

Upmaps as a directed multigraph

```
$ ceph osd dump | grep upmap | head -n3
```

```
pg_upmap_items 1.0 [1128,475,2013,179]
```

```
pg_upmap_items 2.1 [1780,1334,2598,2306,2819,930,2164,2998]
```

```
pg_upmap_items 2.2 [1213,2943,2048,2992,230,435]
```

PG

mapping from

Upmaps as a directed multigraph

```
$ ceph osd dump | grep upmap | head -n3  
pg_upmap_items 1.0 [1128, 475, 2013, 179]  
pg_upmap_items 2.1 [1780, 1334, 2598, 2306, 2819, 930, 2164, 2998]  
pg_upmap_items 2.2 [1213, 2943, 2048, 2992, 230, 435]
```

PG

mapping to

Upmaps as a directed multigraph

```
$ ceph osd dump | grep upmap | head -n3  
pg_upmap_items 1.0 [1128, 475, 2013, 179]  
pg_upmap_items 2.1 [1780, 1334, 2598, 2306, 2819, 930, 2164, 2998]  
pg_upmap_items 2.2 [1213, 2943, 2048, 2992, 230, 435]
```

1.0: [1128] → 475 · [2013] → 179

2.1: [1780] → 1334 · [2598] → 2306 · [2819] → 930 · [2164] → 2998

2.2: [1213] → 2943 · [2048] → 2992 · [230] → 435

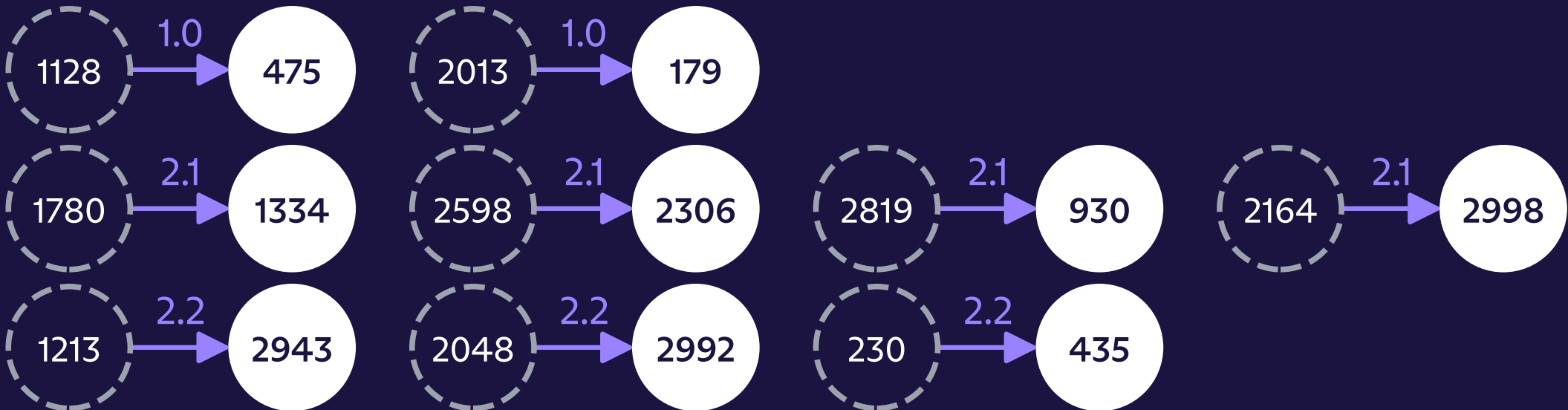
Upmaps as a directed multigraph

```
$ ceph osd dump | grep upmap | head -n3
```

```
pg_upmap_items 1.0 [1128, 475, 2013, 179]
```

```
pg_upmap_items 2.1 [1780, 1334, 2598, 2306, 2819, 930, 2164, 2998]
```

```
pg_upmap_items 2.2 [1213, 2943, 2048, 2992, 230, 435]
```



The graph in seven lines

graph.py

```
1 import json, sys, networkx as nx
2 g = nx.MultiDiGraph()
3 for item in json.load(sys.stdin)["pg_upmap_items"]:
4     if item["pgid"].startswith("2."): # one pool
5         for m in item["mappings"]:
6             g.add_edge(m["from"], m["to"], key=item["pgid"])
7 print(g)
```

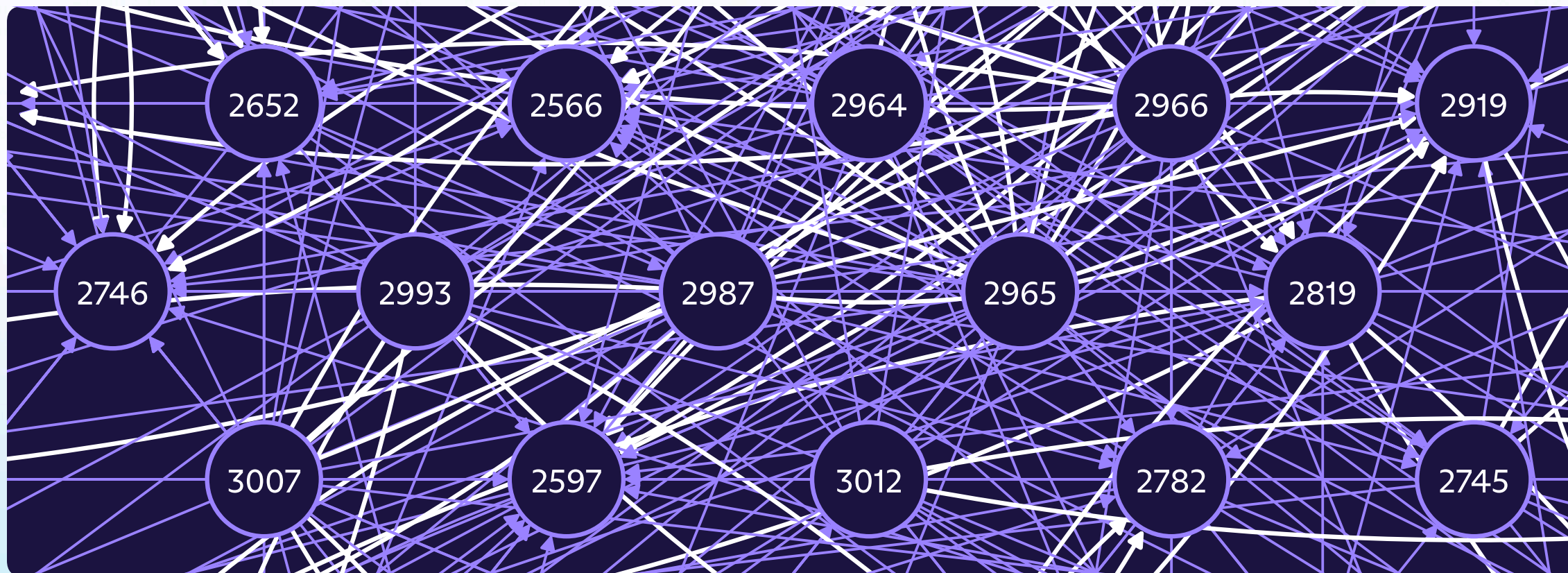
The graph in seven lines

graph.py

```
1 import json, sys, networkx as nx
2 g = nx.MultiDiGraph()
3 for item in json.load(sys.stdin)["pg_upmap_items"]:
4     if item["pgid"].startswith("2."): # one pool
5         for m in item["mappings"]:
6             g.add_edge(m["from"], m["to"], key=item["pgid"])
7 print(g)
```

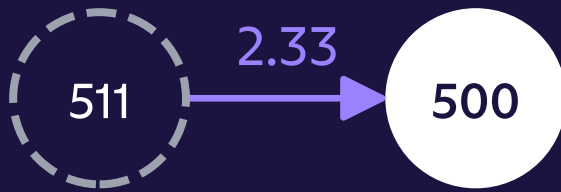
```
$ ceph osd dump -f json | python3 graph.py
MultiDiGraph with 2996 nodes and 89890 edges
```

Look for useful patterns



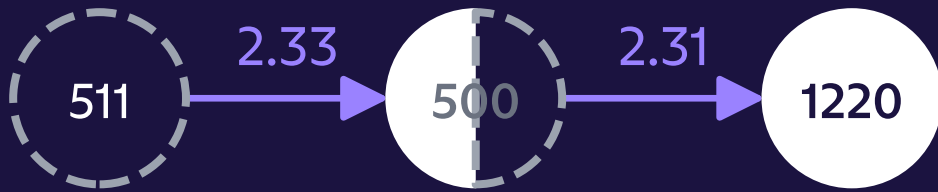
15 OSDs of a pool and their 293 mappings; the parallel ones in white

There are CHAINS! 🐱



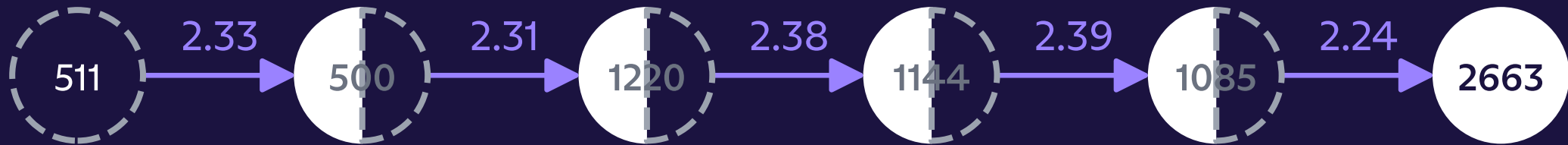
A chain: each OSD in its middle gains a PG and loses one

There are CHAINS! 🐱



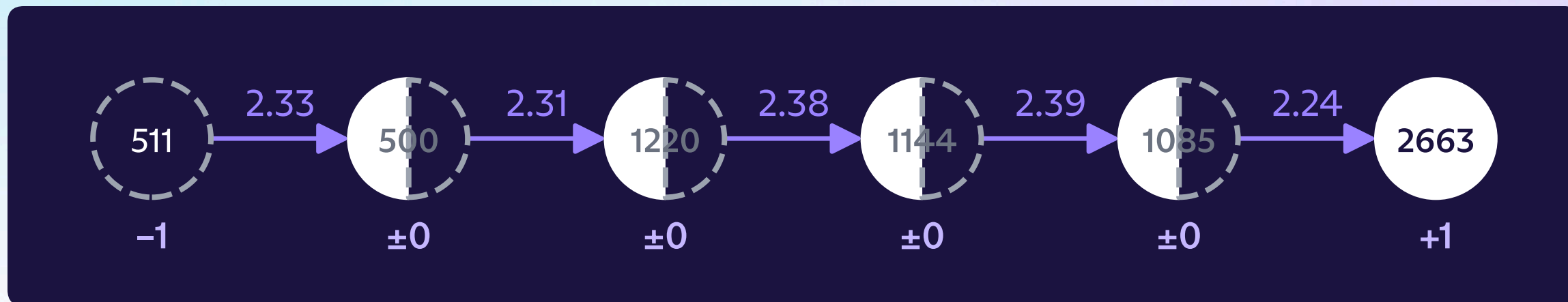
A chain: each OSD in its middle gains a PG and loses one

There are CHAINS! 🐙



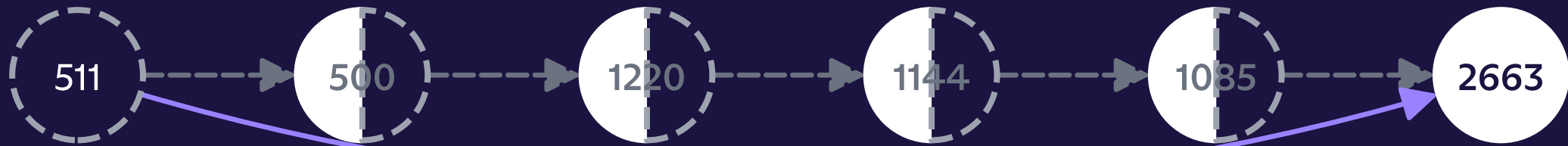
A chain: each OSD in its middle gains a PG and loses one

There are CHAINS! 🐱



Only its ends change PG counts, by -1 and +1

There are CHAINS! 🐱



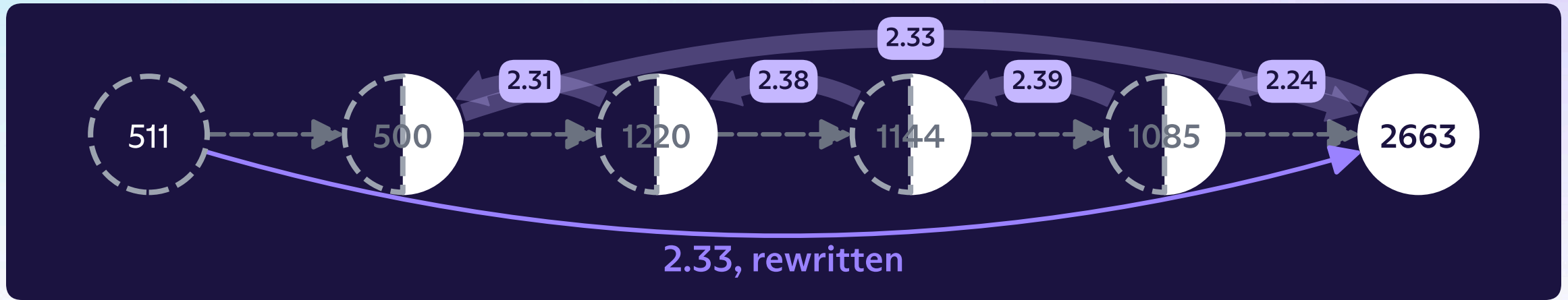
2.33, rewritten

Collapse it: rewrite the first mapping, remove the other 4

```

$ ceph osd dump -f json | python3 graph.py | tail -n 6
# nx.dag_longest_path: a chain of 5 mappings
pgremapper remap 2.33 500 2663
pgremapper remap 2.31 1220 500
pgremapper remap 2.38 1144 1220
pgremapper remap 2.39 1085 1144
pgremapper remap 2.24 2663 1085
  
```

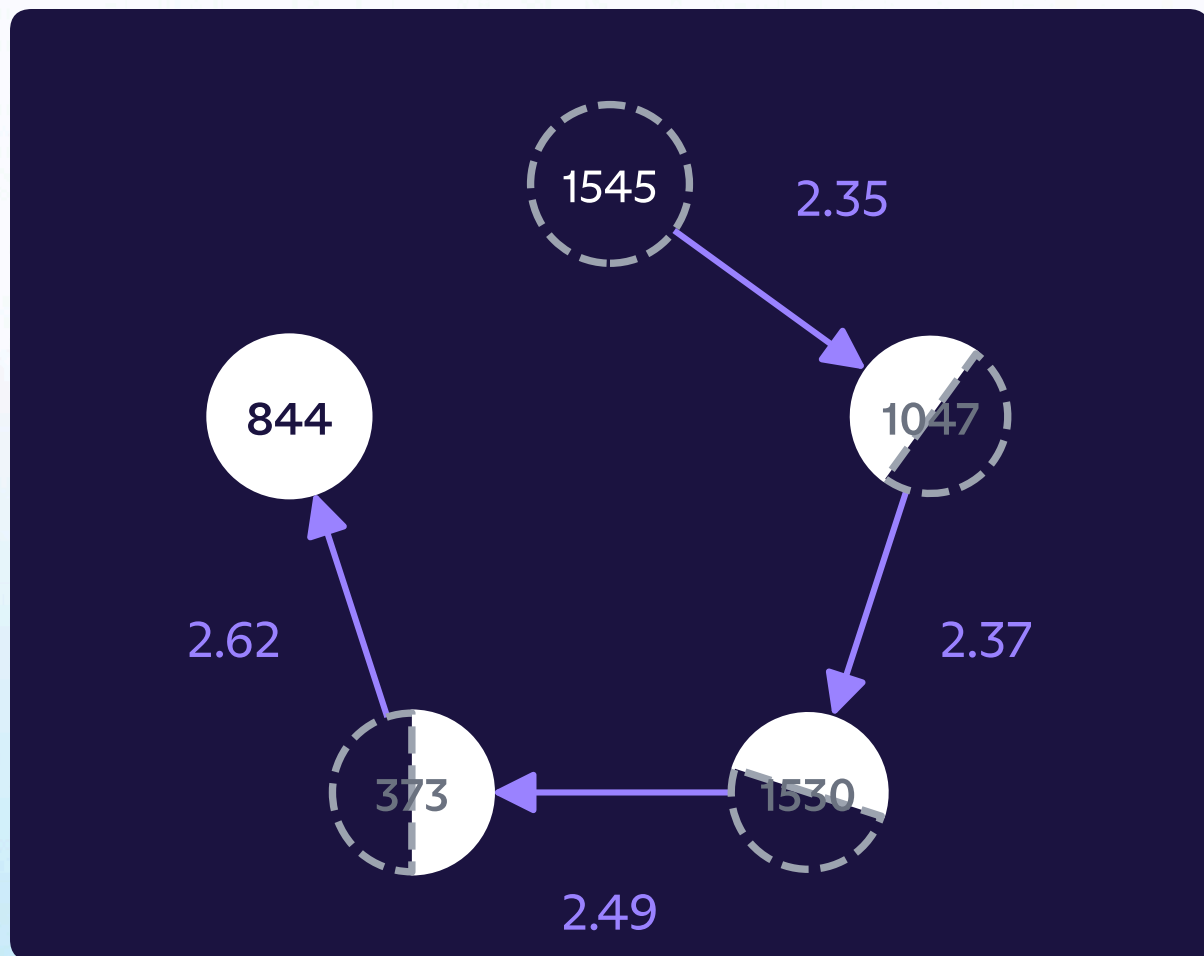
There are CHAINS! 🐱



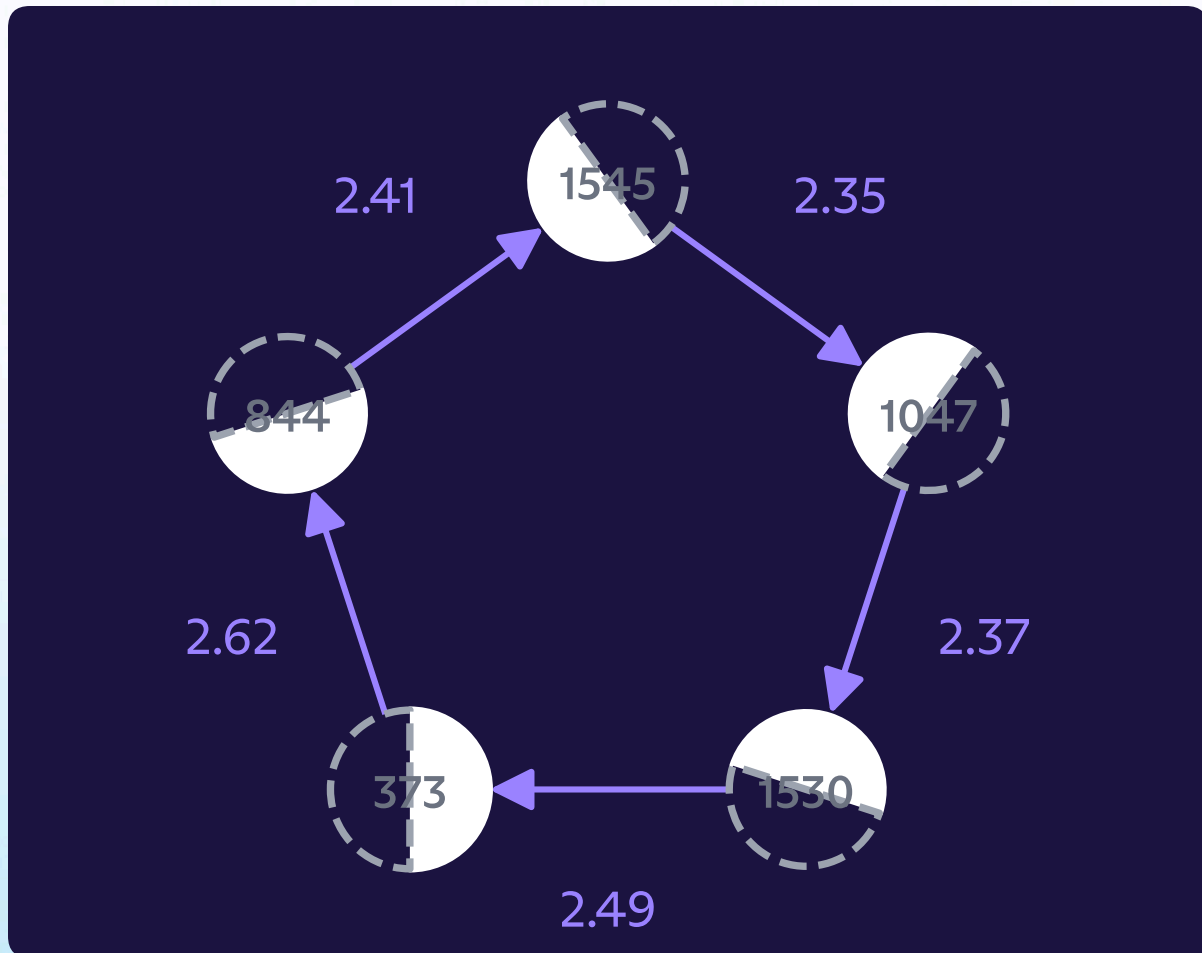
5 PGs move; the mapping count changes by $-5 + 1 = -4$

```
$ ceph osd dump -f json | python3 graph.py | tail -n 6
# nx.dag_longest_path: a chain of 5 mappings
pgremapper remap 2.33 500 2663
pgremapper remap 2.31 1220 500
pgremapper remap 2.38 1144 1220
pgremapper remap 2.39 1085 1144
pgremapper remap 2.24 2663 1085
```

There are CYCLES! 💡🐙

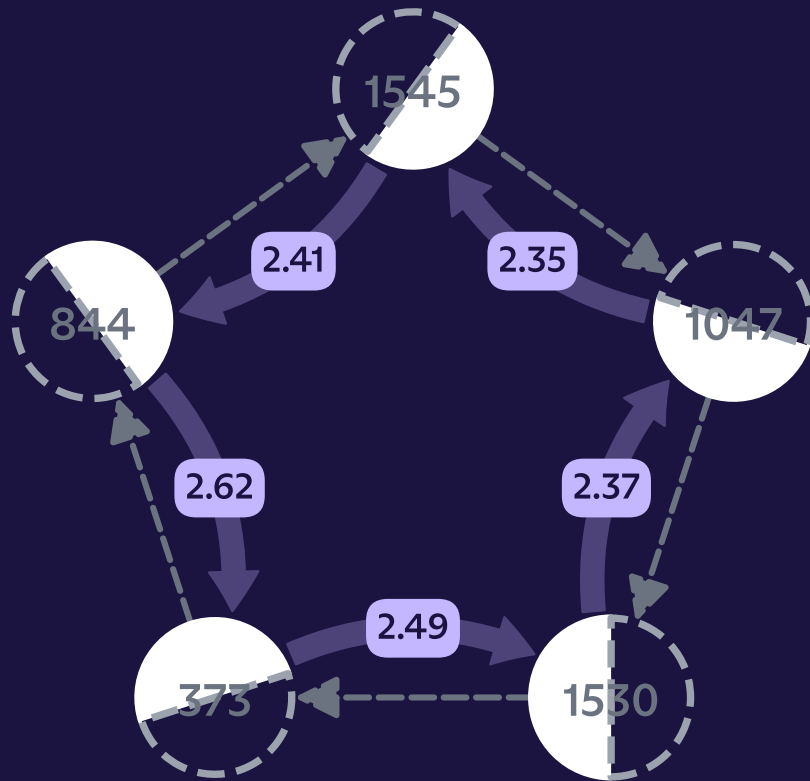


There are CYCLES!



- A chain that closes: every OSD gains a PG and loses one

There are CYCLES!

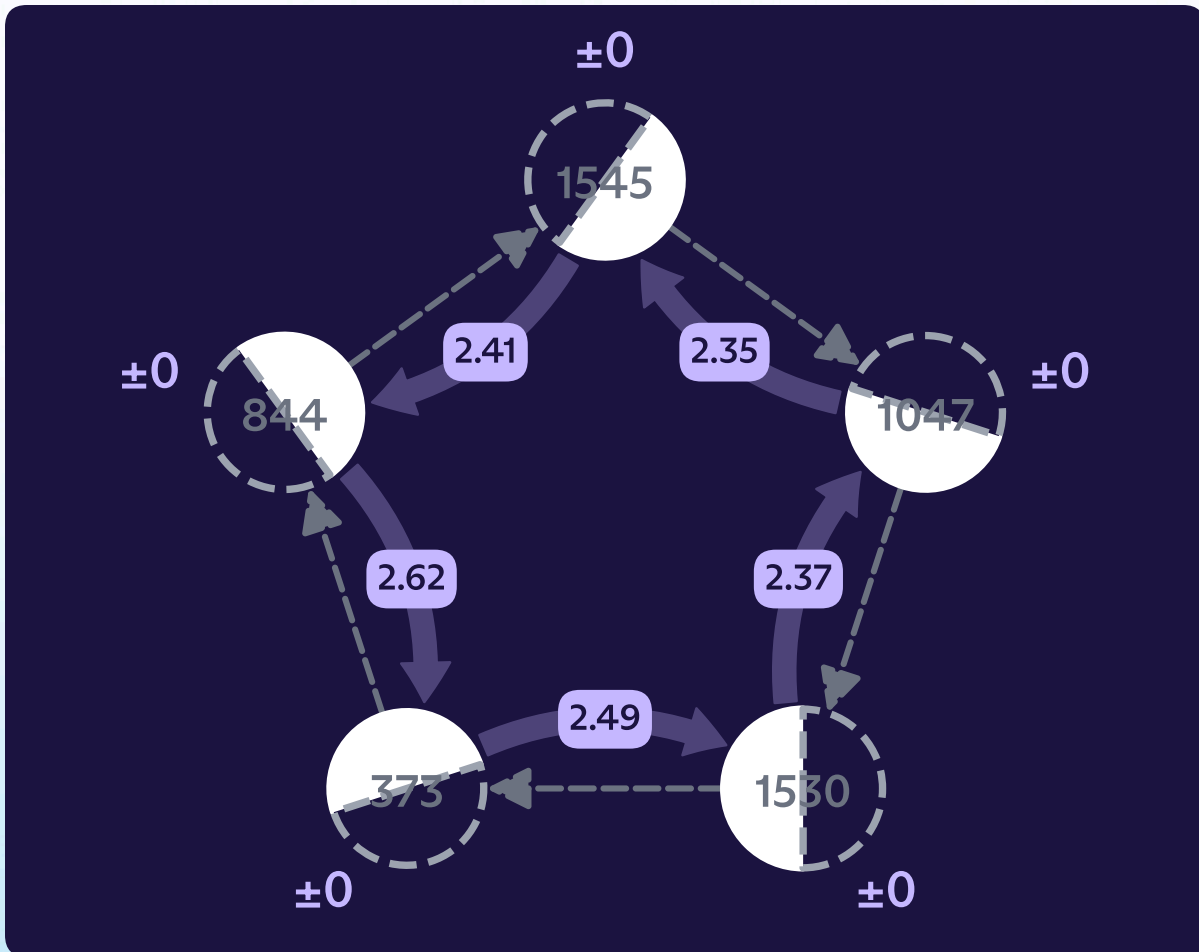


- A chain that closes: every OSD gains a PG and loses one
- Remove all 5 mappings: 5 PGs move back

```

$ ceph osd dump -f json | python3 graph.py | tail -n 6
# nx.find_cycle: a cycle of 5 mappings
pgremapper remap 2.35 1047 1545
pgremapper remap 2.37 1530 1047
pgremapper remap 2.49 373 1530
pgremapper remap 2.62 844 373
pgremapper remap 2.41 1545 844
  
```

There are CYCLES! 💡🐙



- A chain that closes: every OSD gains a PG and loses one
- Remove all 5 mappings: 5 PGs move back
- No OSD's PG count changes: the balance holds

```

$ ceph osd dump -f json | python3 graph.py | tail -n 6
# nx.find_cycle: a cycle of 5 mappings
pgremapper remap 2.35 1047 1545
pgremapper remap 2.37 1530 1047
pgremapper remap 2.49 373 1530
pgremapper remap 2.62 844 373
pgremapper remap 2.41 1545 844
  
```

75–92% of the mappings could go, in 7 of 9 pools

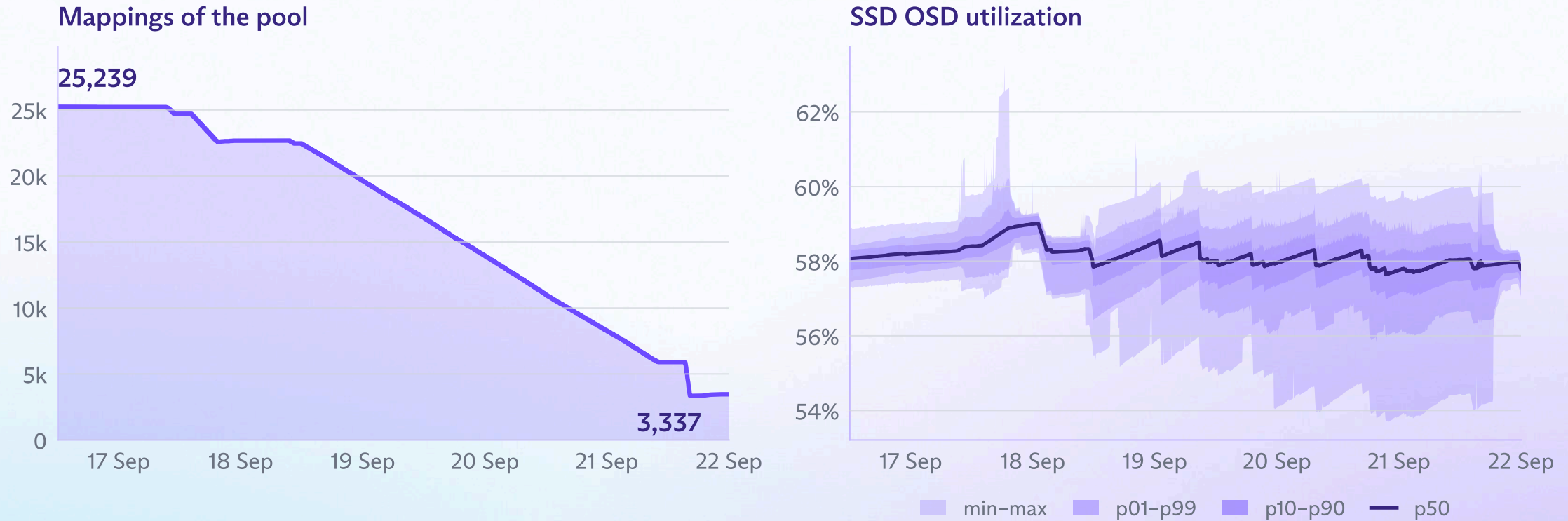
Mappings in Proton’s 9 largest pools



Source: 9 largest Ceph pools, September–October 2026

Removing cycles on SSD: –87% of mappings in 5 days

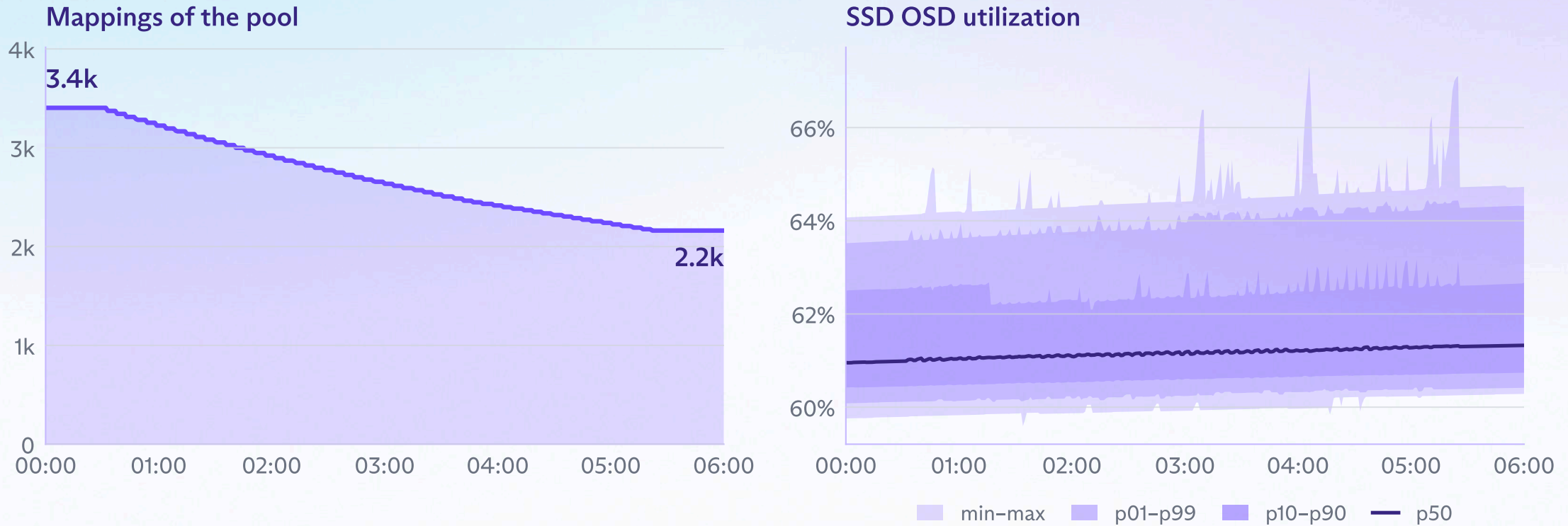
CASE 1



SSD pool, 3x: 25,239 → 3,337 mappings, as calculated 🐱

No failure-domain logic yet: utilization spread widened, until JJ's balancer narrowed it

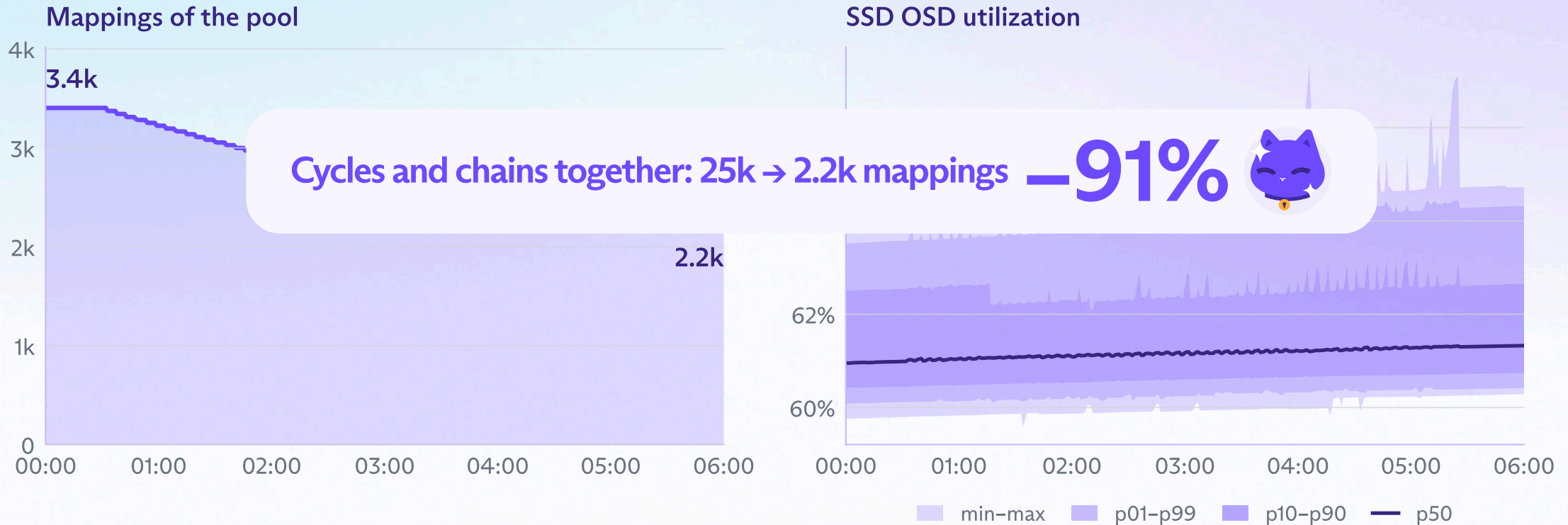
Collapsing chains on SSD: another -4.9% in 6 hours



Same pool: 3.4k → 2.2k, another 1.2k of the 25k at the start

Failure domains respected: the spread kept, without the balancer! 🐱

Collapsing chains on SSD: another -4.9% in 6 hours

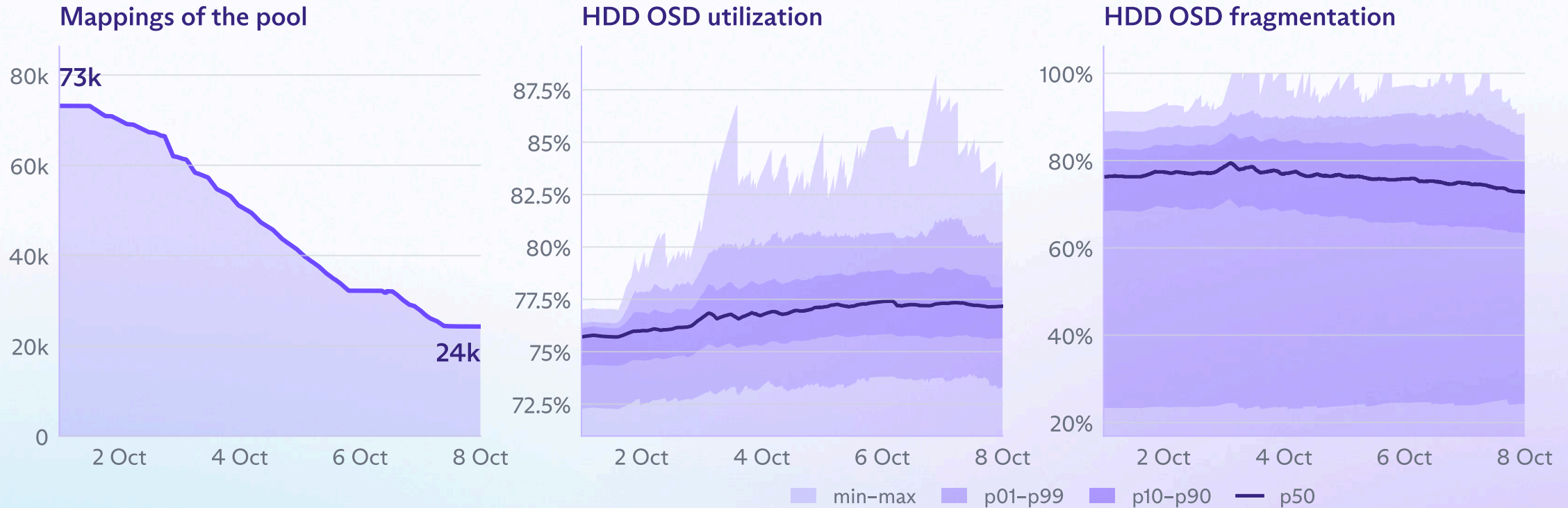


Same pool: 3.4k → 2.2k, another 1.2k of the 25k at the start

Failure domains respected: the spread kept, without the balancer! 🐱

Removing cycles on HDD, EC 4+2: -67% in 7 days, in flight

CASE 3

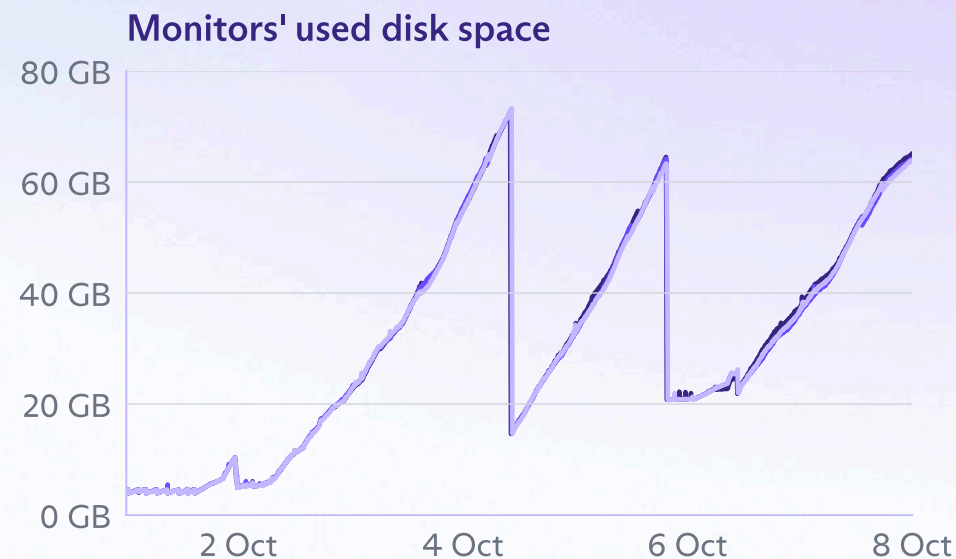


A backfill takes 8 hours here, not 2 minutes: thousands of PGs in flight

Utilization spread and fragmentation to watch: back to baseline? At the next Ceph Days... 🐙

Five caveats to plan for

- ⚠ Removing mappings can invalidate others, across failure domains
- ⚠ Mappings can block each other's one-by-one removal when avoiding their invalidation
- ⚠ Some HDD OSDs may get overwhelmed with too many backfills in and out in parallel
- ⚠ In-flight backfills need free space in the cluster
- ⚠ Monitors keep more OSD maps while PGs move



The monitors' disk while the HDD pool's cycles moved: OSD maps pile up until the PGs are clean, and drop at each compaction.

Upmaps are a graph. Most of them *can go*.

BEFORE

90+%

of the PGs had upmaps, in 6 of 9 pools, a burden on operations

SSD POOL

−91%

of the mappings, by cycles and chains, balance kept

HDD POOL

−67%

in a week, still in flight, without hurting production

Where we go next



Failure domains

Respect them everywhere.



In production

Plan, execute and monitor upmap cleanups.



The fleet

Deploy to all clusters, all pools.



This talk
Ceph Days Zürich 2026



More on Ceph at Proton
Cephalocon 2024
Luis Domingues and Ján Senko

Thank you

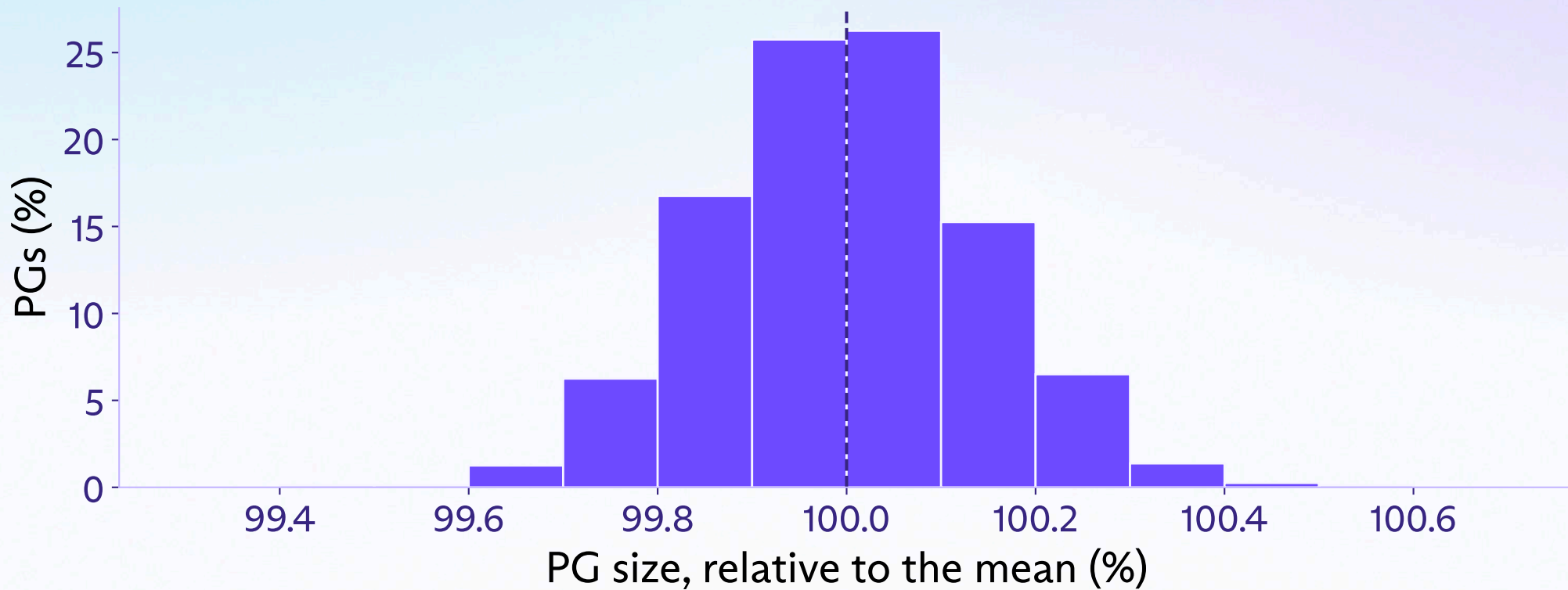
Backup

Cycles first, then chains

graph.py

```
8  # Cycles first: each mapping removed moves one PG.
9  cycle = nx.find_cycle(g)
10 print(f"# nx.find_cycle: a cycle of {len(cycle)} mappings")
11 for u, v, pg in cycle:
12     print(f"pgremapper remap {pg} {v} {u}")
13     g.remove_edge(u, v, pg) # the cycle found, removed
14 remove_all_cycles(g) # likewise, until none is left
15 # Then chains, the longest first.
16 path = nx.dag_longest_path(g)
17 print(f"# nx.dag_longest_path: a chain of {len(path) - 1} mappings")
18 pgs = [min(g[u][v]) for u, v in zip(path, path[1:])]
19 print(f"pgremapper remap {pgs[0]} {path[1]} {path[-1]}")
20 for pg, u, v in zip(pgs[1:], path[1:], path[2:]):
21     print(f"pgremapper remap {pg} {v} {u}")
```

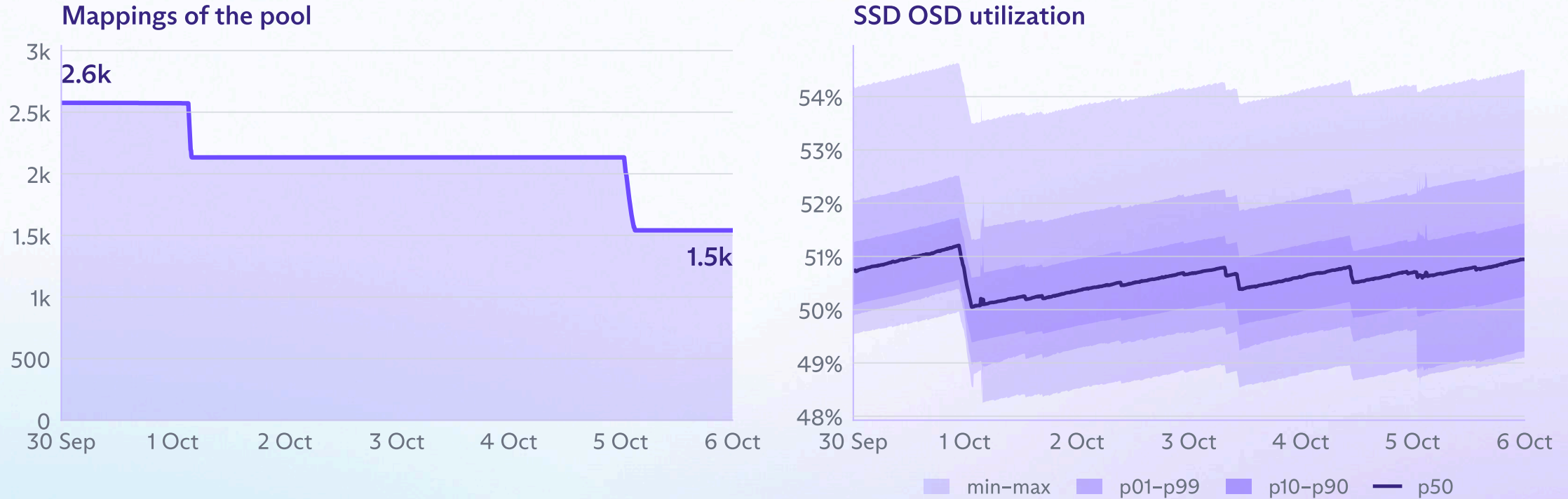
PGs are all one size: same PG count, same usage! (almost) 



90% of a pool's PGs within $\pm 0.2\%$ of their mean size

Source: a production pool, a random sample of its PGs

A second SSD pool: -40% in 6 days



SSD pool, 3x: 2.6k → 1.5k, as calculated, cycles first, then chains; failure domains respected