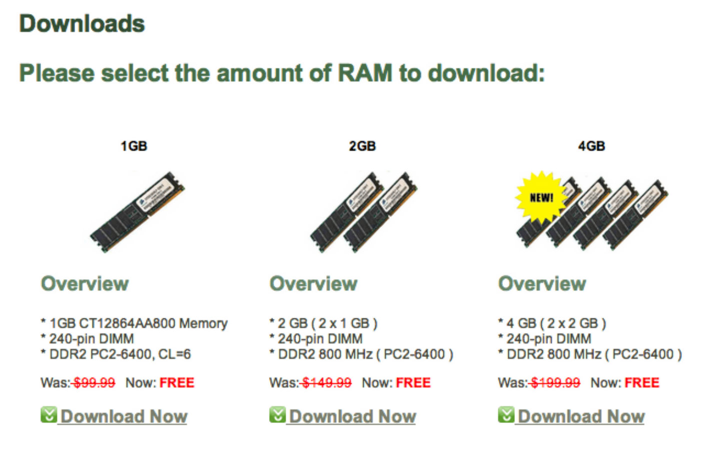
Meta

7 years of cgroup v2

The future of Linux resource control

Chris Down
Kernel, Meta
https://chrisdown.name



Downloads

Please select the amount of RAM to download:

1GB

Overview

* 1GB CT12864AA800 Memory
* 240-pin DIMM
* DDR2 PC2-6400, CL=6

Was: ~~$99.99~~ Now: **FREE**

Download Now

2GB

Overview

* 2 GB (2 x 1 GB)
* 240-pin DIMM
* DDR2 800 MHz (PC2-6400)

Was: ~~$149.99~~ Now: **FREE**

Download Now

4GB

Overview

* 4 GB (2 x 2 GB)
* 240-pin DIMM
* DDR2 800 MHz (PC2-6400)

Was: ~~$199.99~~ Now: **FREE**

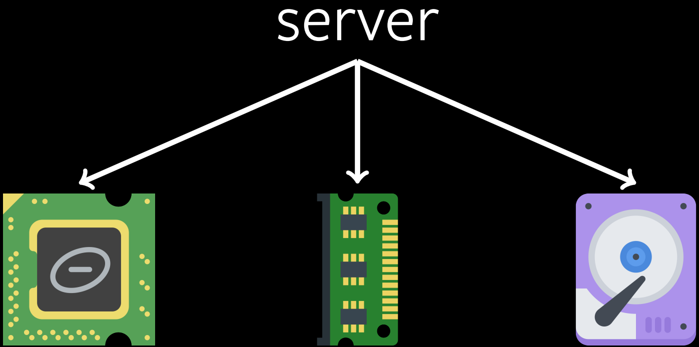
Download Now

Lance Cheung, CC BY-NC-SA: `bit.ly/sevimage`

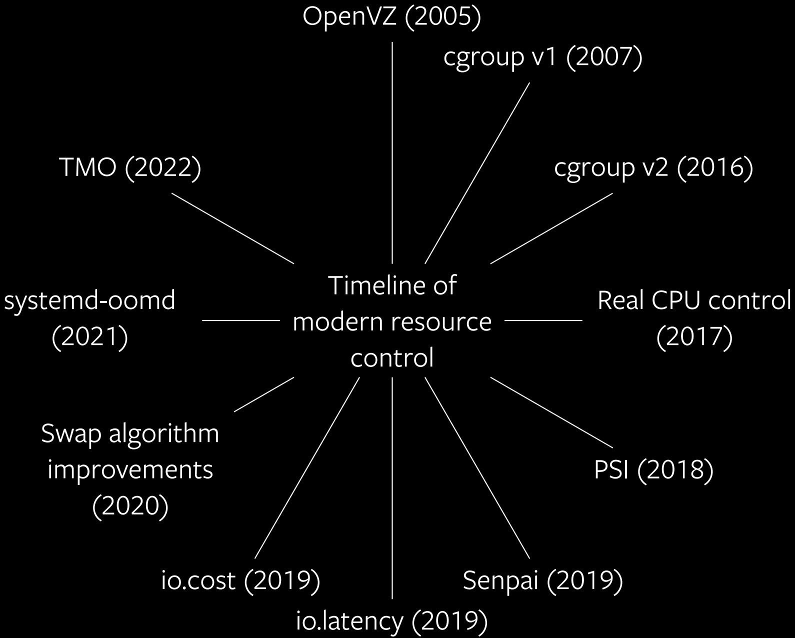
USE CGROUPS

CONTROL RESOURCES

DON'T MAKE THE WEBSITE FALL OVER



server



Timeline of modern resource control

OpenVZ (2005)

cgroup v1 (2007)

cgroup v2 (2016)

Real CPU control (2017)

PSI (2018)

Senpai (2019)

io.latency (2019)

io.cost (2019)

Swap algorithm improvements (2020)

systemd-oomd (2021)

TMO (2022)

- containerd ≥ 1.4
- Docker/Moby ≥ 20.10
- podman ≥ 1.4.4
- runc ≥ 1.0.0
- systemd ≥ 226

...and many more!

facebook

cgroupv2: Linux's new unified control group system

Chris Down (cdown@fb.com)
Production Engineer, Web Foundation

bit.ly/cgv2qcon

How did this work in cgroup v1?

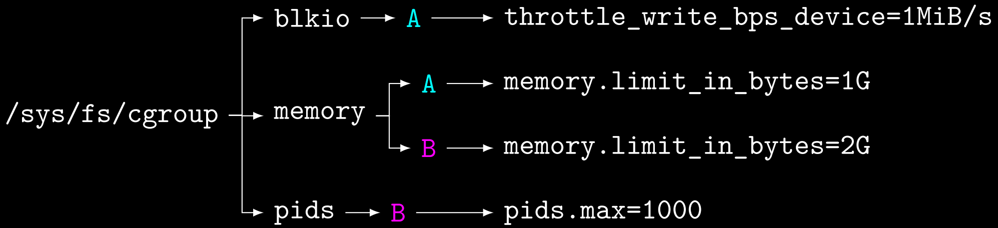
cgroup v1 has a hierarchy per-resource, for example:

```
% ls /sys/fs/cgroup
cpu/ cpuacct/  cpuset/  devices/  freezer/
memory/  net_cls/  pids/
```

Each resource hierarchy contains cgroups for this resource:

```
% find /sys/fs/cgroup/memory -type d
/sys/fs/cgroup/memory/background.slice
/sys/fs/cgroup/memory/background.slice/sshd.service
/sys/fs/cgroup/memory/workload.slice
```

Hierarchy in cgroup v1



/sys/fs/cgroup

blkio → A → throttle_write_bps_device=1MiB/s

memory → A → memory.limit_in_bytes=1G

memory → B → memory.limit_in_bytes=2G

pids → B → pids.max=1000

How does this work in cgroup v2?

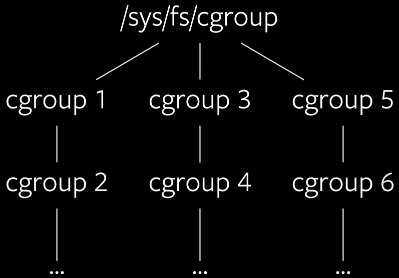
cgroup v2 has a *unified hierarchy*, for example:

```
% ls /sys/fs/cgroup
background.slice/  workload.slice/
```

Each cgroup can support multiple resource domains:

```
% ls /sys/fs/cgroup/background.slice
async.slice/  foo.mount/  cgroup.subtree_control
memory.high  memory.max  pids.current  pids.max
```

How does this work in cgroup v2?



/sys/fs/cgroup

cgroup 1

cgroup 3

cgroup 5

cgroup 2

cgroup 4

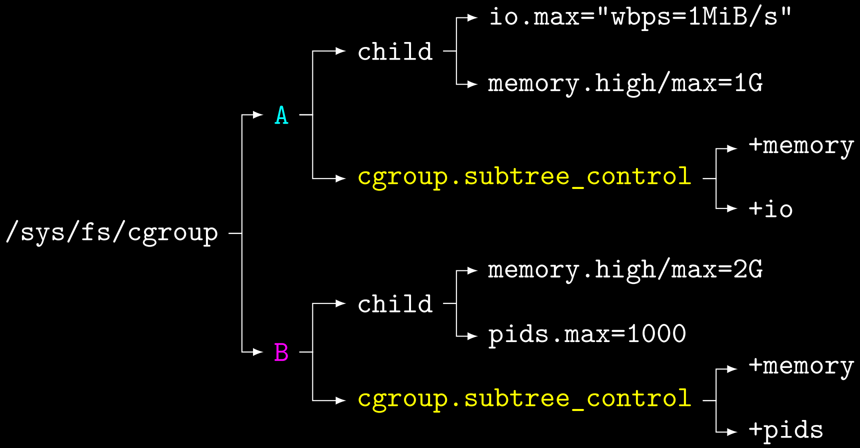
cgroup 6

...

...

...

Hierarchy in cgroup v2



/sys/fs/cgroup

A

child

io.max="wbps=1MiB/s"

memory.high/max=1G

cgroup.subtree_control

+memory

+io

B

child

memory.high/max=2G

pids.max=1000

cgroup.subtree_control

+memory

+pids

Why do we need a single resource hierarchy?

Why do we need a single resource hierarchy?

- Memory starts to run out

Why do we need a single resource hierarchy?

- Memory starts to run out
- This causes us to reclaim page caches/swap, causing disk IO

Why do we need a single resource hierarchy?

- Memory starts to run out
- This causes us to reclaim page caches/swap, causing disk IO
- This reclaim costs sometimes non-trivial CPU cycles

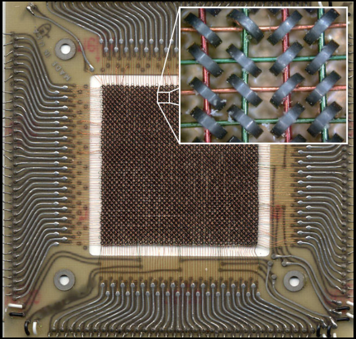
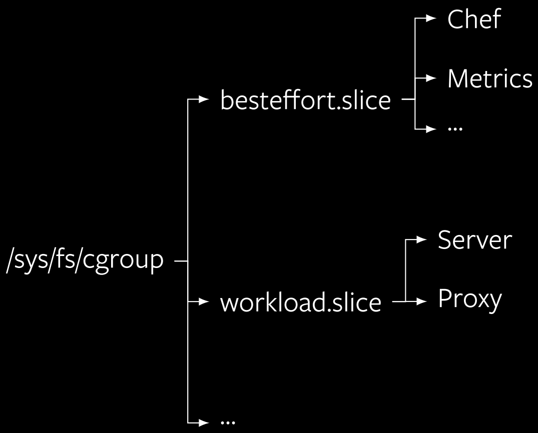


Image: Orion J on Wikimedia Commons, CC-BY

- Memory is divided in to multiple “types”: anon, cache, buffers, etc
- “Reclaimable” or “unreclaimable” is important, but not guaranteed
- RSS is kinda bullshit, sorry

```
# cgroup v2
echo 1G > /sys/fs/cgroup/foo/memory.max
```



/sys/fs/cgroup

besteffort.slice

Chef

Metrics

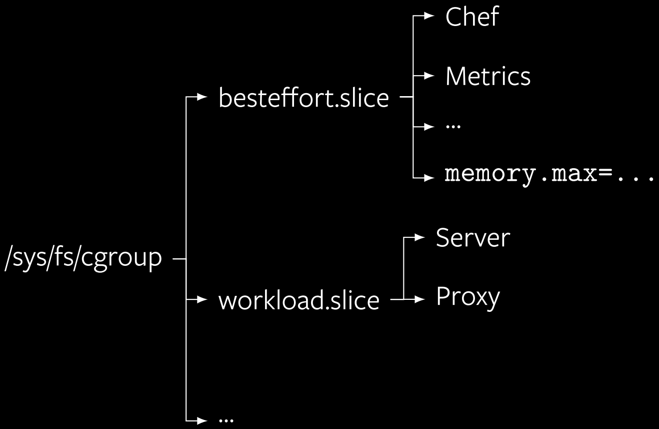
…

workload.slice

Server

Proxy

…



/sys/fs/cgroup

besteffort.slice

Chef

Metrics

...

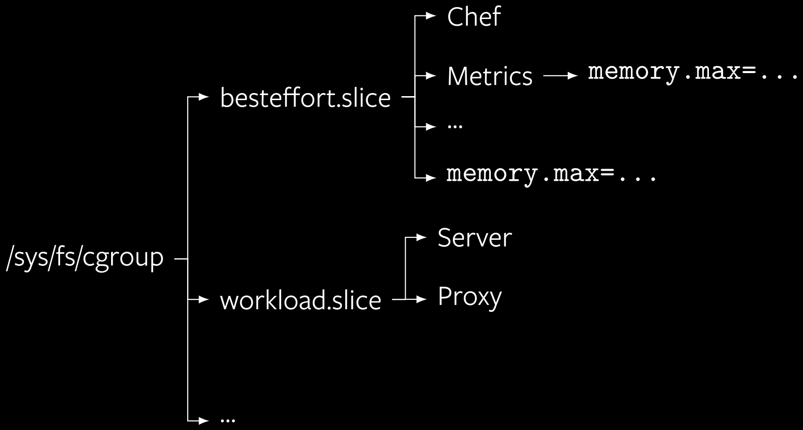
`memory.max=...`

workload.slice

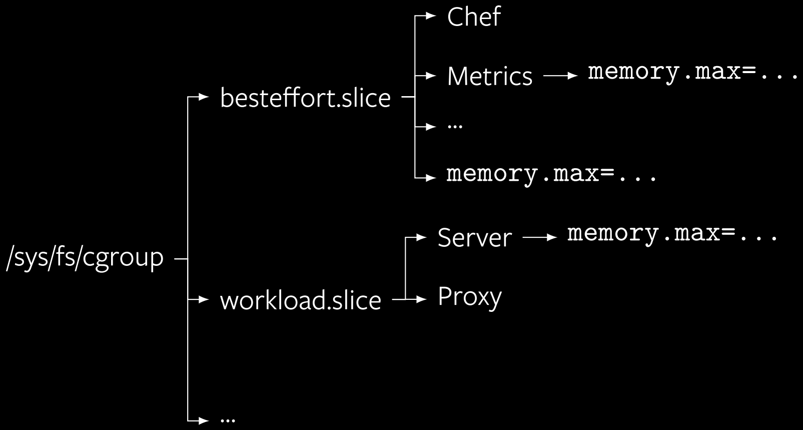
Server

Proxy

...



- /sys/fs/cgroup
  - besteffort.slice
    - Chef
    - Metrics
      - `memory.max=...`
    - …
    - `memory.max=...`
  - workload.slice
    - Server
    - Proxy
  - …



/sys/fs/cgroup

besteffort.slice

Chef

Metrics

`memory.max=...`

...

`memory.max=...`

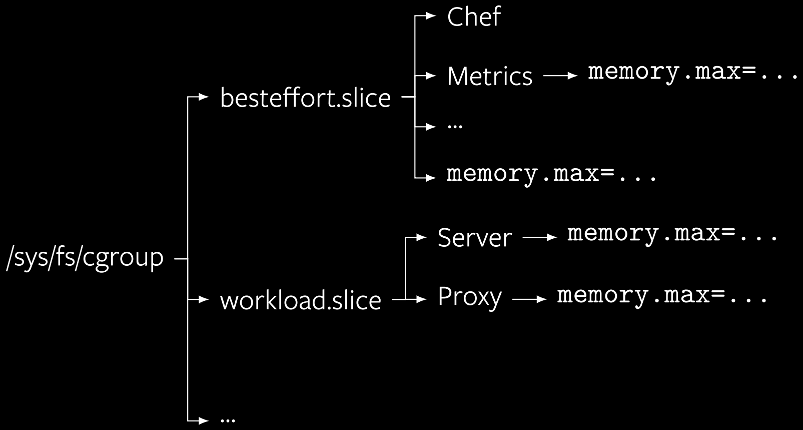
workload.slice

Server

`memory.max=...`

Proxy

...



/sys/fs/cgroup

besteffort.slice

Chef

Metrics

`memory.max=...`

...

`memory.max=...`

workload.slice

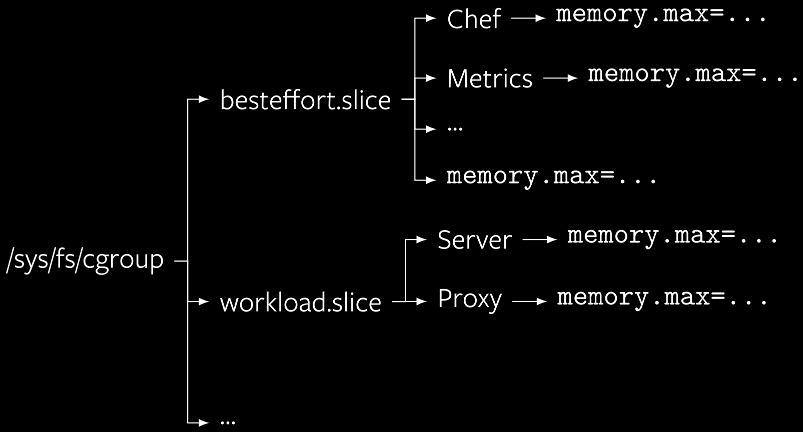
Server

`memory.max=...`

Proxy

`memory.max=...`

...



/sys/fs/cgroup

besteffort.slice

Chef

memory.max=...

Metrics

memory.max=...

...

memory.max=...

workload.slice

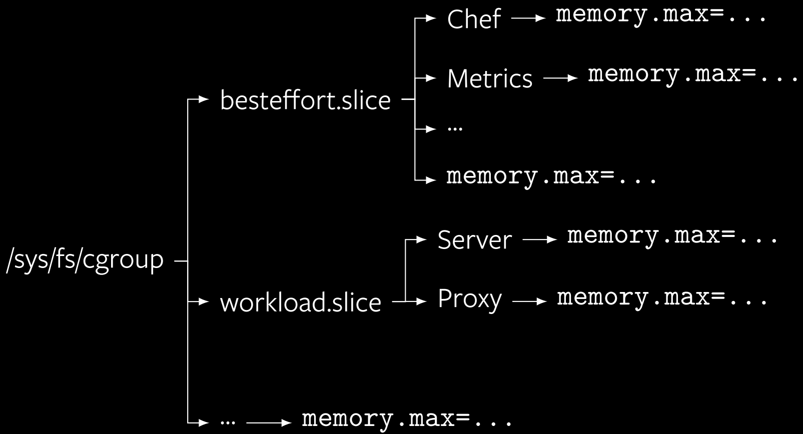
Server

memory.max=...

Proxy

memory.max=...

...



/sys/fs/cgroup

besteffort.slice

Chef → `memory.max=...`

Metrics → `memory.max=...`

...

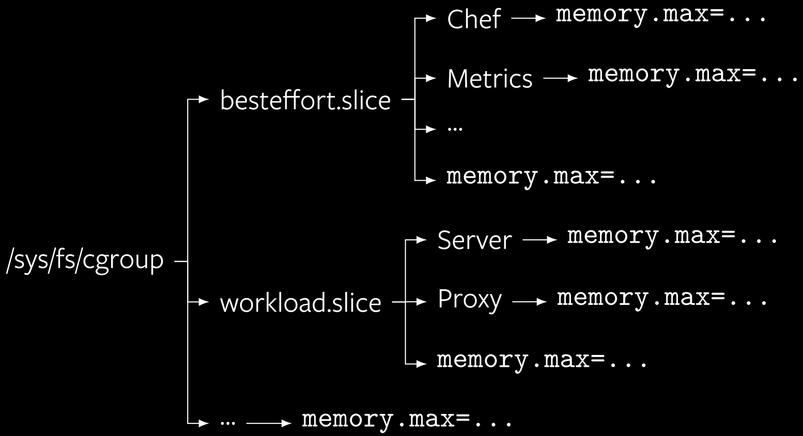
`memory.max=...`

workload.slice

Server → `memory.max=...`

Proxy → `memory.max=...`

... → `memory.max=...`



/sys/fs/cgroup

besteffort.slice

Chef → memory.max=...

Metrics → memory.max=...

...

memory.max=...

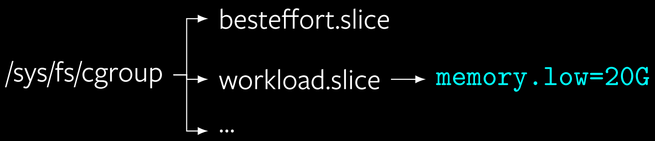
workload.slice

Server → memory.max=...

Proxy → memory.max=...

memory.max=...

... → memory.max=...

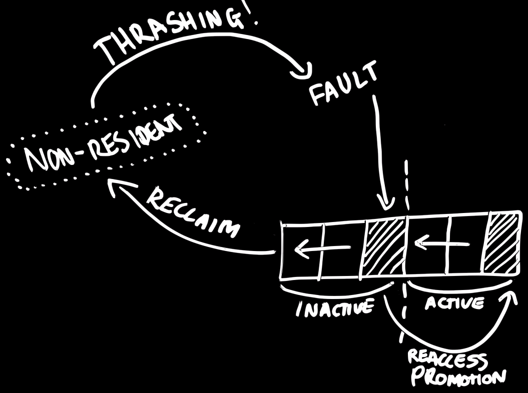


/sys/fs/cgroup

besteffort.slice

workload.slice → `memory.low=20G`

...



- memory.low and memory.min bias reclaim away from a cgroup
- Reclaim can still be triggered when protected on global memory shortage

How can you view memory usage for a process in Linux?

How can you view memory usage for a process in Linux?

- SIKE THIS SLIDE WAS A TRAP

```
% size -A chrome | awk '$1 == ".text" { print $2 }'
132394881
```

```
% cat /proc/self/cgroup
0::/system.slice/foo.service
% cat /sys/fs/cgroup/system.slice/foo.service/memory.current
3786670080
```

- `memory.current` tells the truth, but the truth is sometimes complicated
- Slack grows to fill up to cgroup limits if there’s no global pressure

psi

“If I had more of this resource, I could probably run *N*% faster”

- Find bottlenecks
- Detect workload health issues before they become severe
- Used for resource allocation, load shedding, pre-OOM detection

```
% cat /sys/fs/cgroup/system.slice/memory.pressure
some avg10=0.21 avg60=0.22 total=4760988587
full avg10=0.21 avg60=0.22 total=4681731696
```

```
% time make -j4 -s
real    3m58.050s
user    13m33.735s
sys     1m30.130s

# Peak memory.current bytes: 803934208
```

```
% sudo sh -c 'echo 600M > memory.high'
% time make -j4 -s
real    4m0.654s
user    13m28.493s
sys     1m31.509s

# Peak memory.current bytes: 629116928
```

```
% sudo sh -c 'echo 400M > memory.high'
% time make -j4 -s
real    4m3.186s
user    13m20.452s
sys     1m31.085s

# Peak memory.current bytes: 419368960
```

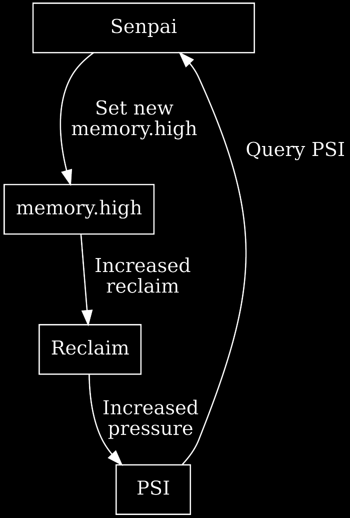
```
% sudo sh -c 'echo 300M > memory.high'
% time make -j4 -s
^C
real    9m9.974s
user    10m59.315s
sys     1m16.576s
```

```
% sudo senpai /sys/fs/cgroup/...
% make -j4 -s  # ran in the cgroup
               # senpai is operating on
```

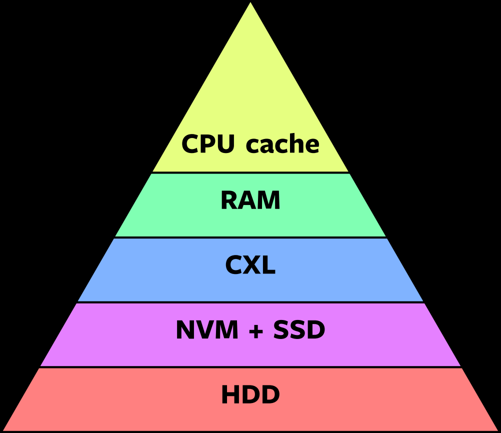
Senpai output during stabilisation:

```
2023-02-23 14:26:43
    limit=340.48M pressure=0.16
    delta=202 integral=202
2023-02-23 14:26:44
    limit=340.48M pressure=0.13
    delta=0 integral=202
```

The job still takes 4 minutes, with less than half the memory we originally used.

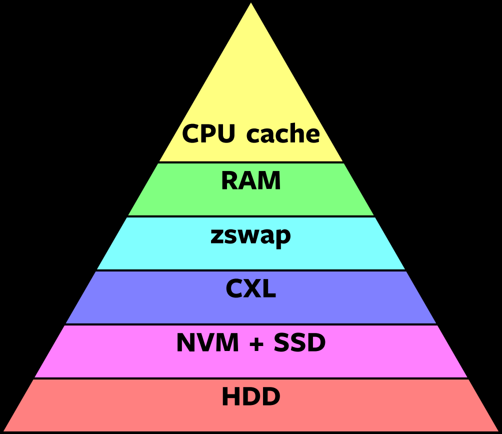


bit.ly/cgsenpai



↑ high cost, low latency

↓ low cost, high latency



↑ high cost, low latency

↓ low cost, high latency

New swap algorithm in kernel 5.8+:

- Repeadly faulting/evicting a cache page over and over? Evict a heap page instead

New swap algorithm in kernel 5.8+:

- Repeadly faulting/evicting a cache page over and over? Evict a heap page instead
- We only trade one type of paging for another: we’re not adding I/O load

Effects of swap algorithm improvements:

Effects of swap algorithm improvements:

- Decrease in heap memory

Effects of swap algorithm improvements:

- Decrease in heap memory
- Increase in cache memory

Effects of swap algorithm improvements:

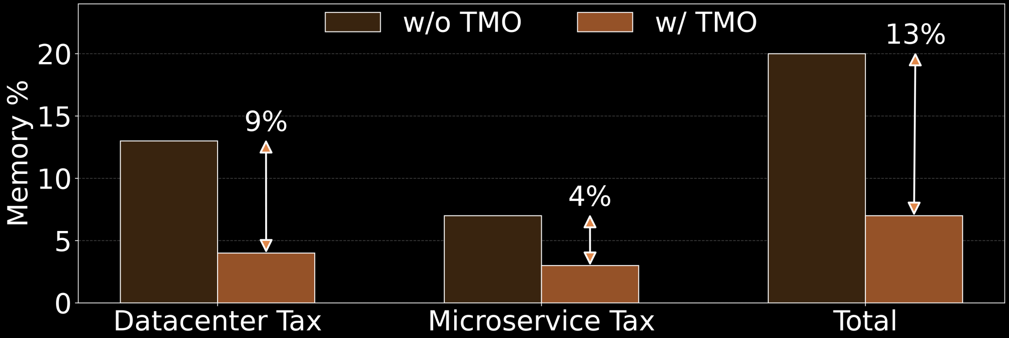
- Decrease in heap memory
- Increase in cache memory
- Increase in web server performance

Effects of swap algorithm improvements:

- Decrease in heap memory
- Increase in cache memory
- Increase in web server performance
- Decrease in disk I/O from paging activity

Effects of swap algorithm improvements:

- Decrease in heap memory
- Increase in cache memory
- Increase in web server performance
- Decrease in disk I/O from paging activity
- Increase in workload stacking opportunities

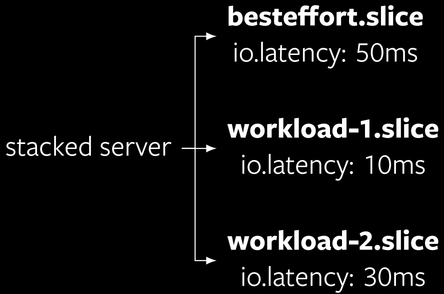


bit.ly/tmopost

- Memory starts to run out
- This causes us to reclaim page caches/swap, causing disk IO
- This reclaim costs sometimes non-trivial CPU cycles

```
% echo '8:16 wbps=1MiB wiops=120' > io.max
```

```
# target= is in milliseconds
% echo '8:16 target=10' > io.latency
```

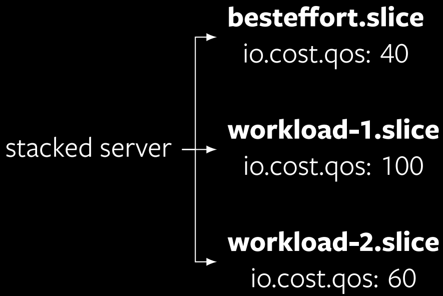


stacked server

besteffort.slice
io.latency: 50ms

workload-1.slice
io.latency: 10ms

workload-2.slice
io.latency: 30ms



stacked server

besteffort.slice
io.cost.qos: 40

workload-1.slice
io.cost.qos: 100

workload-2.slice
io.cost.qos: 60

`bit.ly/iocost + bit.ly/resctlbench`

All the cool kids are using it

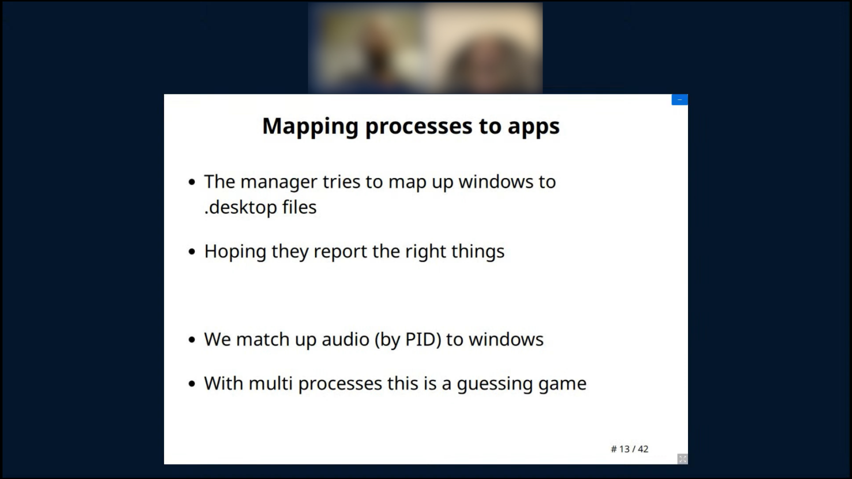
cgroup v2 users:

- containerd $\geq$ 1.4
- Docker/Moby $\geq$ 20.10
- podman $\geq$ 1.4.4
- runc $\geq$ 1.0.0
- systemd $\geq$ 226

Distributions:

- Fedora uses by default on $\geq$ 32
- Coming to other distributions by default soon^{TM}

Try it yourself: `cgroup_no_v1=all` on the kernel command line



Mapping processes to apps

- The manager tries to map up windows to .desktop files
- Hoping they report the right things

- We match up audio (by PID) to windows
- With multi processes this is a guessing game

bit.ly/kdecgv2

Try out cgroup v2 for yourself:

- `cgroup_no_v1=all` on the kernel command line
- Docs: `bit.ly/cgroupv2doc`
- Whitepaper: `bit.ly/cgroupv2wp`

Feedback:

- E-mail: `chris@chrisdown.name`
- Mastodon: `@cdown@fosstodon.org`

Meta