Memory overcommit for overcommitted admins

Presented by Jonathan Davies

Based on the work of David Vrabel

OCTOBER 2018 | KVM FORUM

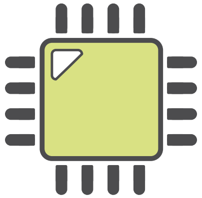
Outline

1. Memory overcommit: Why? How?
2. Design approach
3. Rapid prototyping by simulation
4. Conclusion

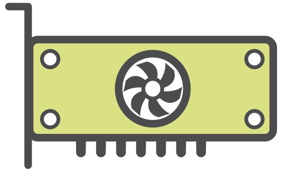
Resource overcommit

In many virtualization scenarios, resource overcommit is welcome.

Users understand that performance will be impacted when over-committing.

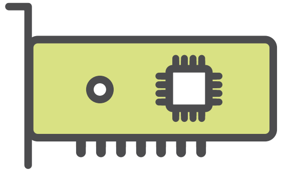


CPUs



GPUs

Storage capacity



Network bandwidth

Resource overcommit

In many virtualization scenarios, resource overcommit is welcome.

Users understand that performance will be impacted when over-committing.

CPUs

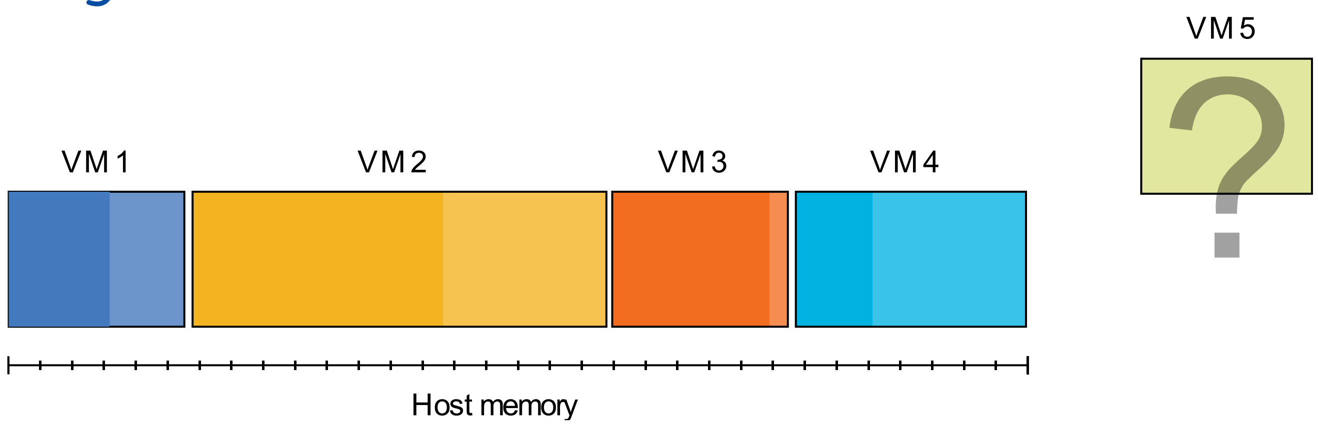
GPUs

Storage capacity

Network bandwidth

Memory

Why memory overcommit?



This host is “full”.

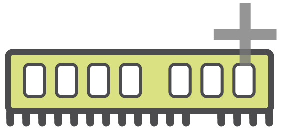
But in some cases the VMs may not need all their memory all the time.

- e.g. bursty workloads where VMs are often idle
- e.g. the user who demands a 16 GB VM but will only ever use 4 GB

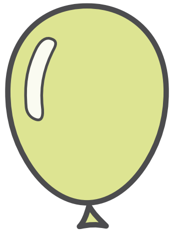
If we need to, we should be able to find enough room in which to start another VM

- e.g. to cope with unforeseen short-term capacity requirements

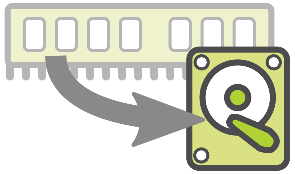
Memory overcommit techniques



Memory hotplug/unplug



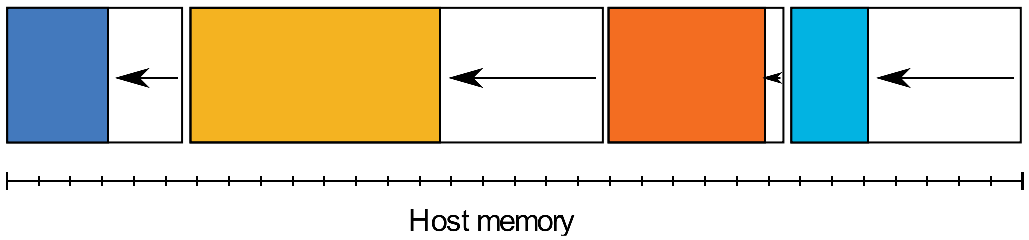
Memory ballooning



Hypervisor paging

Memory hotplug/unplug

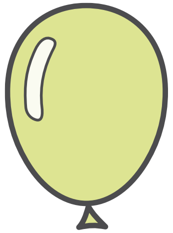
Add and remove virtual DIMMs



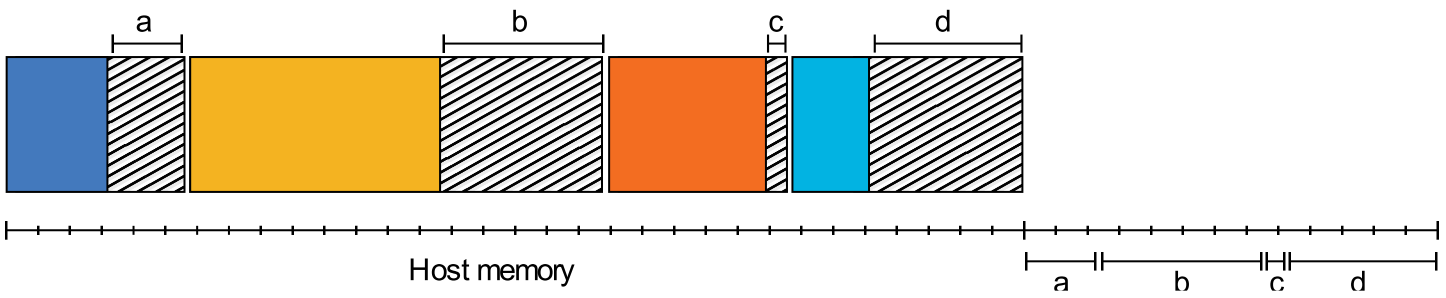
Issues

- Removing memory is hard, requiring OS support, so will not work for all VMs
- No guarantee over timely release
- Administrator needs to indicate how much memory it is “safe” to remove

Memory ballooning



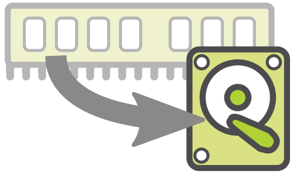
Reclaim unused pages from running VMs



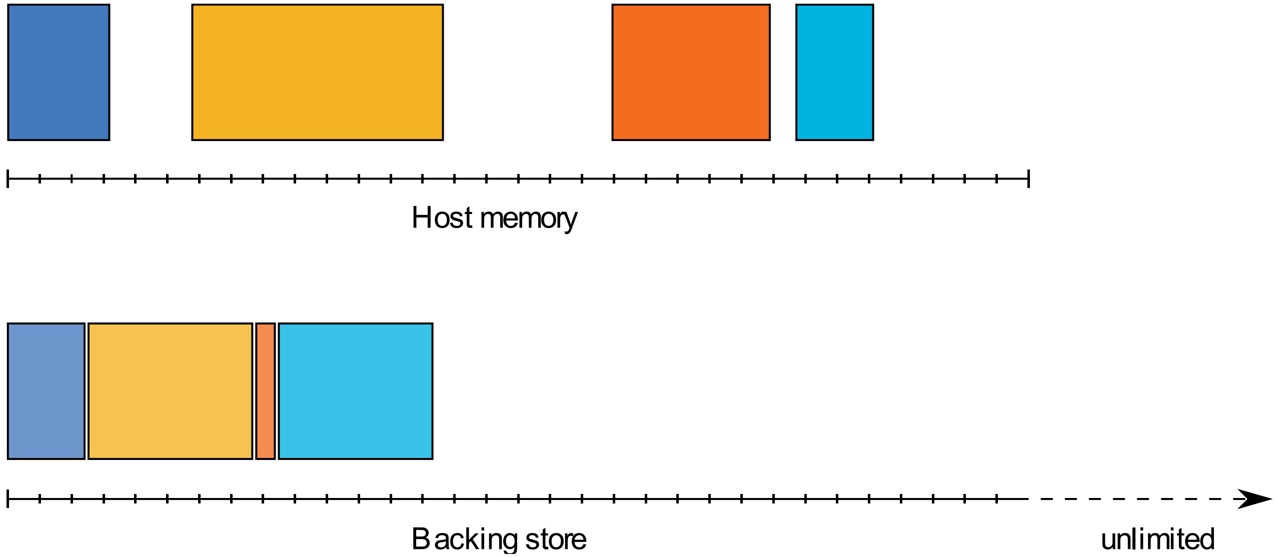
Issues

- Requires an in-guest driver, so VMs must be trusted to co-operate
- VM reboot resets balloon, so bulk reboots could cause OOM
- No guarantee over timely release
- Administrator needs to indicate how much memory it is “safe” to remove

Hypervisor paging



Shift some of a VM’s memory into backing store



Issues

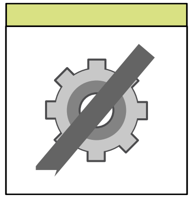
- Slow to swap pages back in when needed
- Requires paging infrastructure in hypervisor

Outline

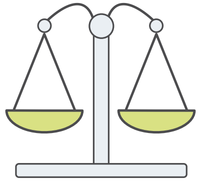
Our design goals

Automatic sizing

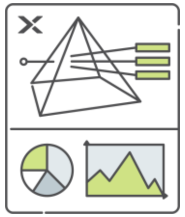
OS agnostic



Agentless

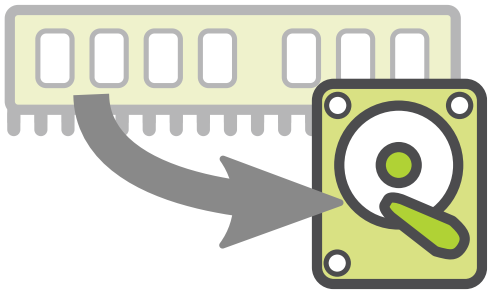


Fair



Disable-able

Chosen approach



Use hypervisor paging

- ✓ Hypervisor paging is straightforward with Linux
- ✓ No OS co-operation required
- ✓ No in-guest agent required
- ✗ Won’t work with VMs with PCI-passthrough
- ✗ Cannot use memory allocated from hugetlbfs

Use feedback from the system to automatically size VMs

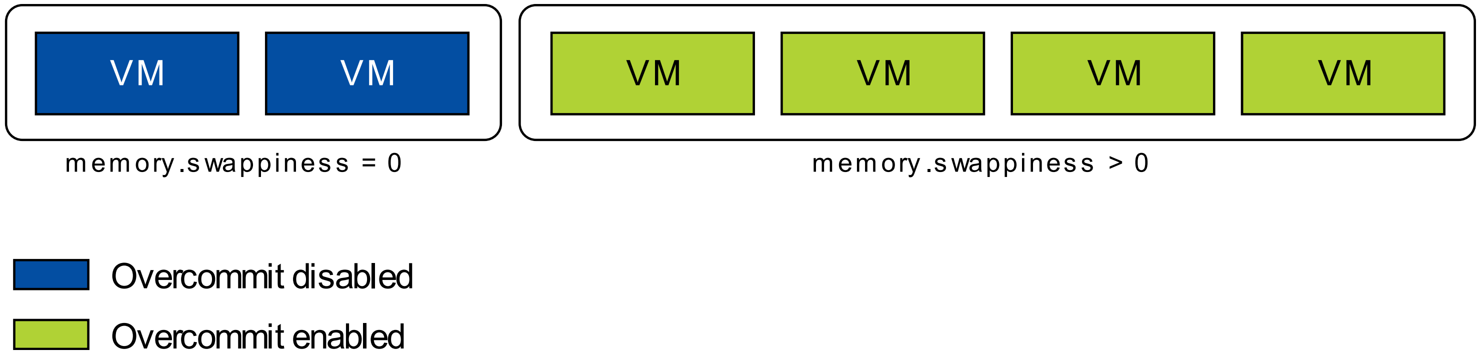
- ✓ No guesswork or manual configuration of memory sizes
- ✓ Minimise amount of swapping needed

Control paging using cgroups

Each VM sits in two cgroups:

1. A per-VM cgroup. Its memory size is controlled by adjusting memory.limit_in_bytes.
2. For VMs with overcommit enabled, a host-wide cgroup with swap enabled.
   For VMs with overcommit disabled, a host-wide cgroup with swap disabled.

Use a per-host swap device.



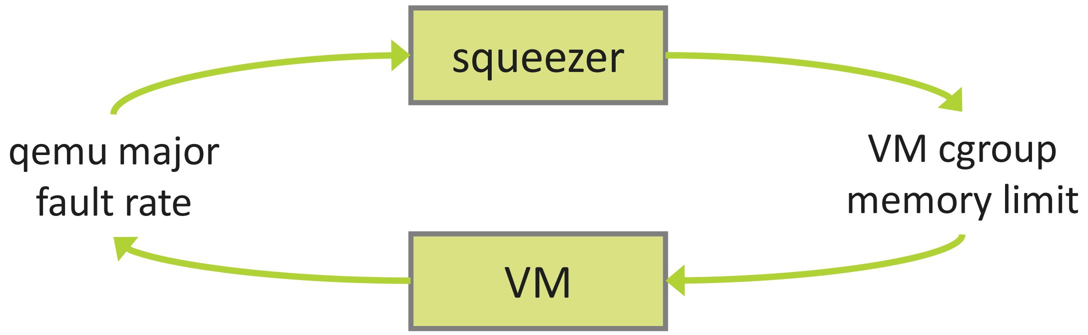
Automatic VM sizing

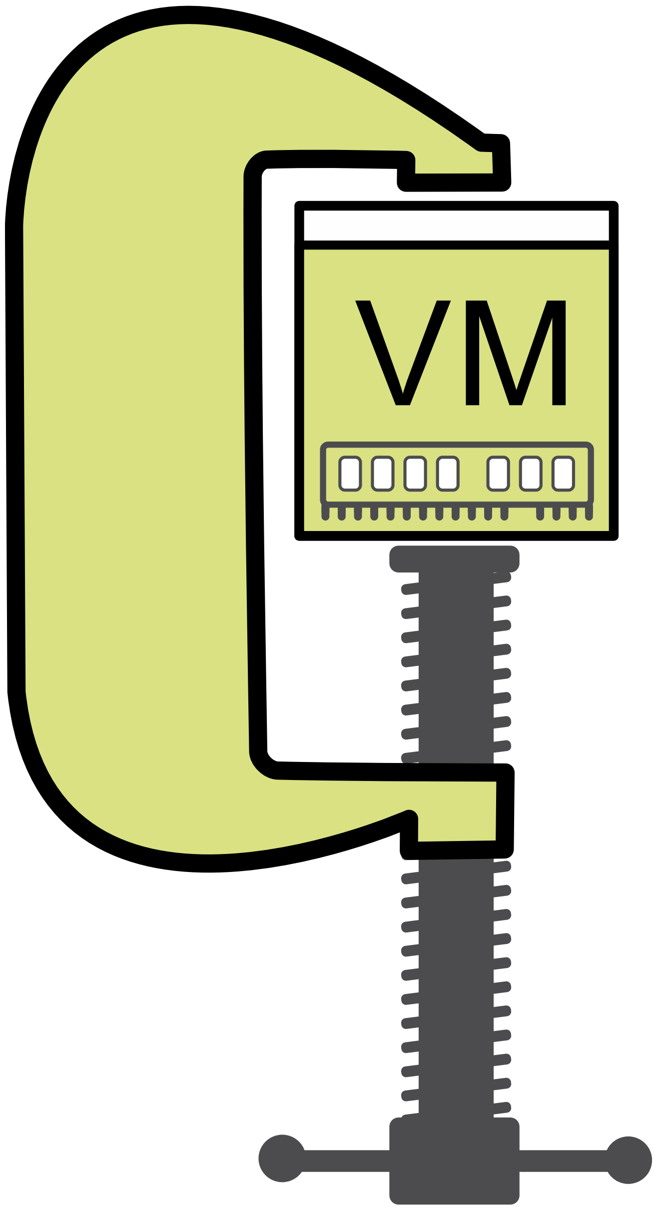
A "squeezer" daemon maintains an estimate of each VM's working set.

The squeezer estimates the VM's working set size from qemu major fault rates[1]

- Low fault rate → reduce estimate
- High fault rate → increase estimate

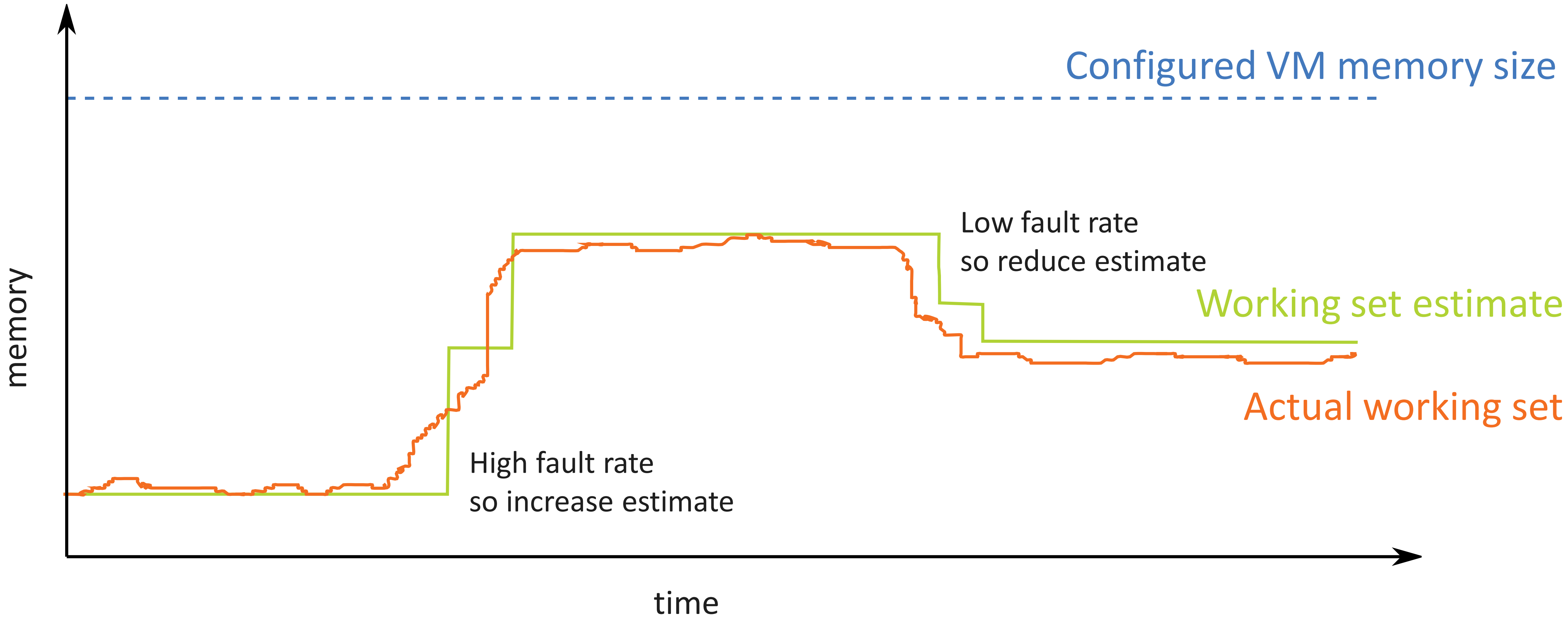
The squeezer sets the VM's cgroup memory limit to this estimate.



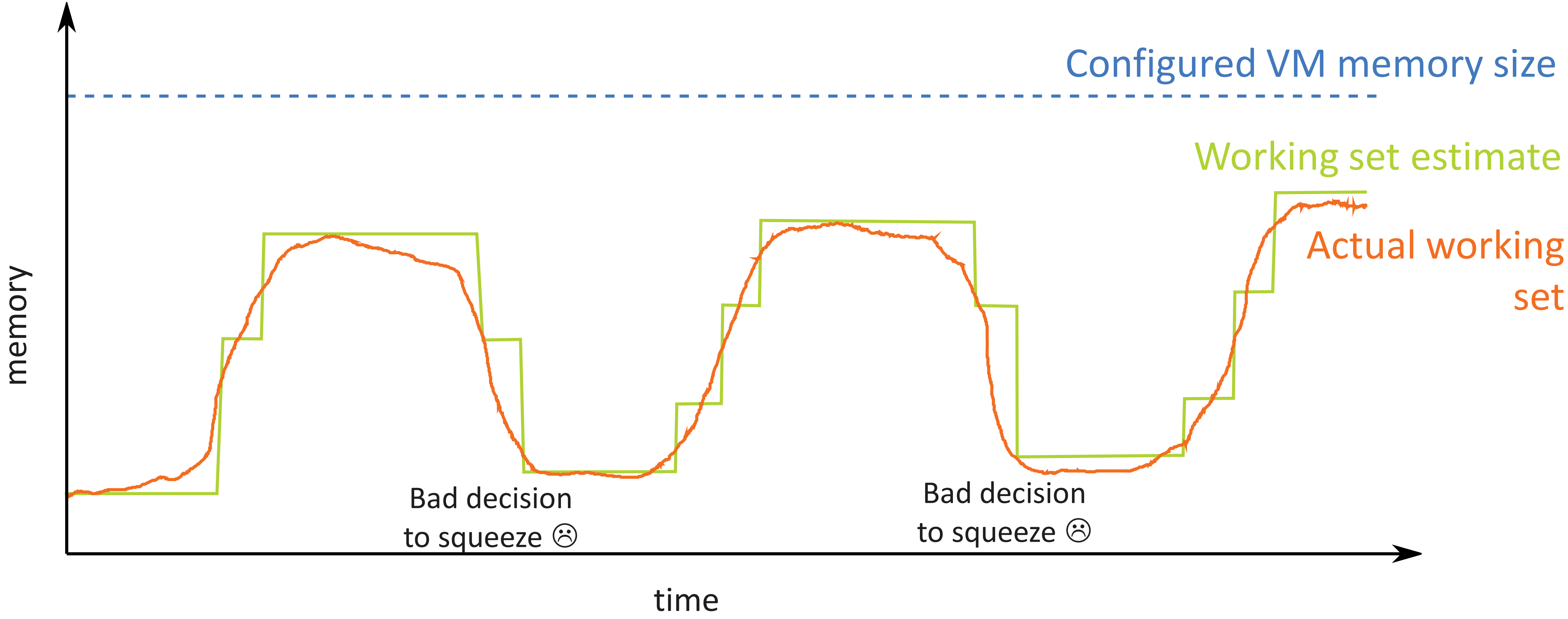


[1] by observing majflt + cmajflt from qemu process's /proc/<pid>/stat file

Automatic VM sizing

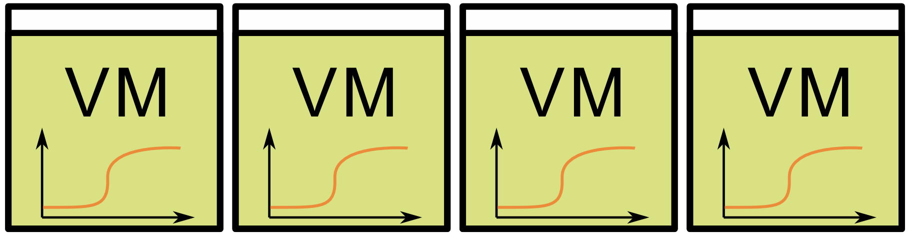


Problem: excessive paging



Problem: over-full hosts

What if several VMs on a host increase their working set at around the same time?



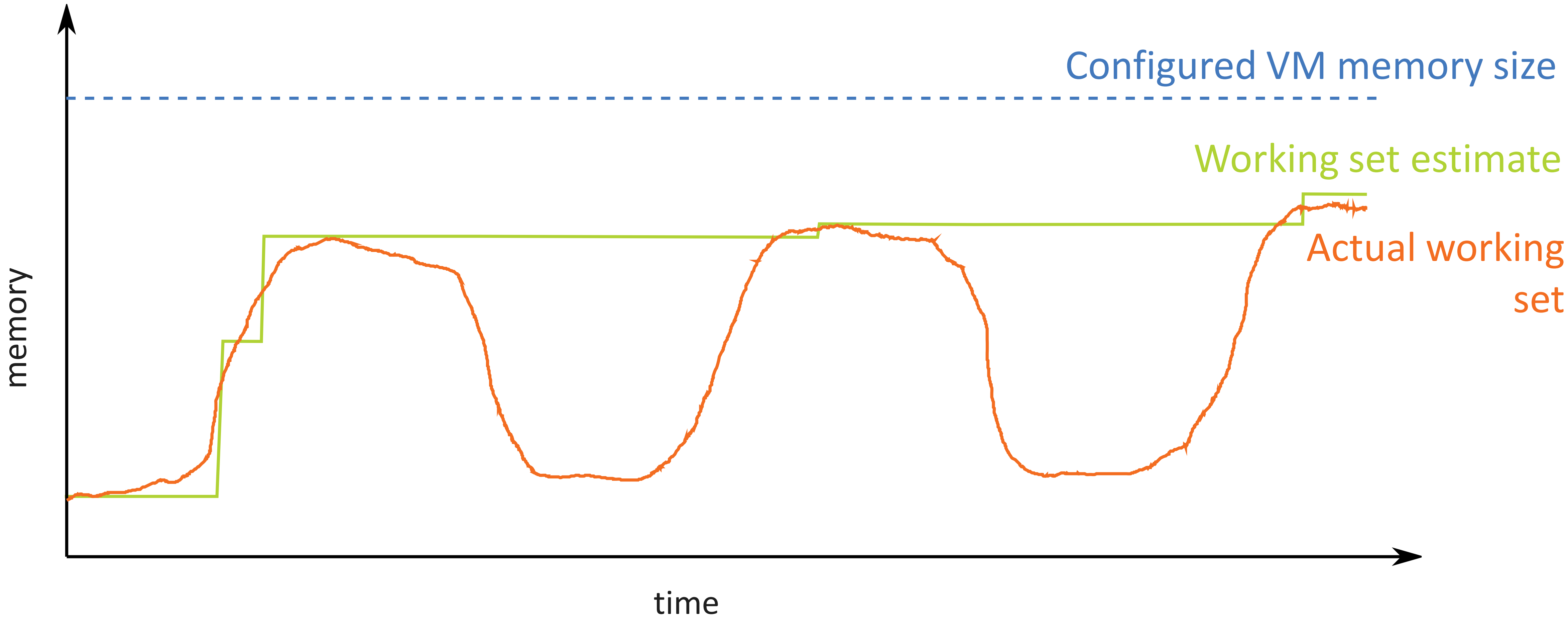
If the sum of VMs' working sets exceeds host memory, the host is overcommitted.

- VM performance will be heavily impacted.
- Computation of working set estimates cannot occur.

We may be able to eventually recover by migrating VMs to other hosts.

But we want to avoid this scenario whenever possible.

Solution: track long-term trends



Outline

1. Memory overcommit: Why? How?
2. Design approach
3. **Rapid prototyping by simulation**
4. Conclusion

How to evaluate a squeezing algorithm?

Goal: maximise squeezing while minimising performance degradation

There are several important parameters for the squeezer, including

1. How quickly should the squeezer react to an increase in fault rate?
2. When the fault rate is low, when should the squeezer try decreasing the cgroup limit?
3. By what amount should the squeezer increase or decrease the cgroup limit?
4. What length of time window should be used for the long-term estimate?

How can we find an optimal squeezing algorithm?

Use simulation to aid rapid prototyping

Approach:

1. Monitor

Monitor real guest memory usage patterns under real-world workloads

2. Model

Construct a model that can quantify the impact of a squeezer algorithm on a VM workload

3. Simulate

Rapidly evaluate possible squeezer algorithms

Monitoring: use idle page tracking

We used idle page tracking[1] to monitor the page-accessing activity of guests under real world workloads.

The file /sys/kernel/mm/page_idle/bitmap contains one bit per physical page on the host.

When a page is accessed, the corresponding bit is cleared.

We periodically sample the pages corresponding to a VM's memory, resetting the bits after each sample.

```
/proc/<pid>/maps:
...
7f7b7faf4000-7f7b7faf8000 rw-p 00000000 00:00 0
7f7b7faf8000-7f7c7faf8000 rw-s 00000000 00:1d 1048319 ...
7f7c7faf8000-7f7c7faf9000 ---p 00000000 00:00 0
...
```

```
off = 0x7f7b 7faf a000 / 4096 * 8 = 0x3f bdbf d7d0
```

```
/proc/<pid>/pagemap:          PFN            flags
...
3fbdbfd7c0: 39 40 0e 01 00 00 00 a2 e2 47 0e 01 00 00 00 a2
3fbdbfd7d0: b4 32 0e 01 00 00 00 a2 ea 14 0e 01 00 00 00 a2
3fbdbfd7e0: 72 d2 0d 01 00 00 00 a2 63 f5 0d 01 00 00 00 a2
3fbdbfd7f0: 3d 37 0e 01 00 00 00 a2 3d ef 0d 01 00 00 00 a2
...
```

```
off = (0x0000 010e 32b4 / 8) & ~0x7 = 0x21c650
bit = 0x0000 010e 32b4 & 0x3f = 52
```

```
/sys/kernel/mm/page_idle/bitmap:
...
021c640: 04 51 44 10 50 00 05 01 54 00 14 40 00 11 05 15
021c650: 00 00 40 01 44 10 00 50 30 0a 22 82 00 20 68 55
021c660: 80 28 04 80 d7 15 77 57 24 d2 08 c0 7f 5d d7 f5
021c670: 40 0a 3c 22 7f 75 57 7f 11 05 01 00 5f 14 6e 00
...
```

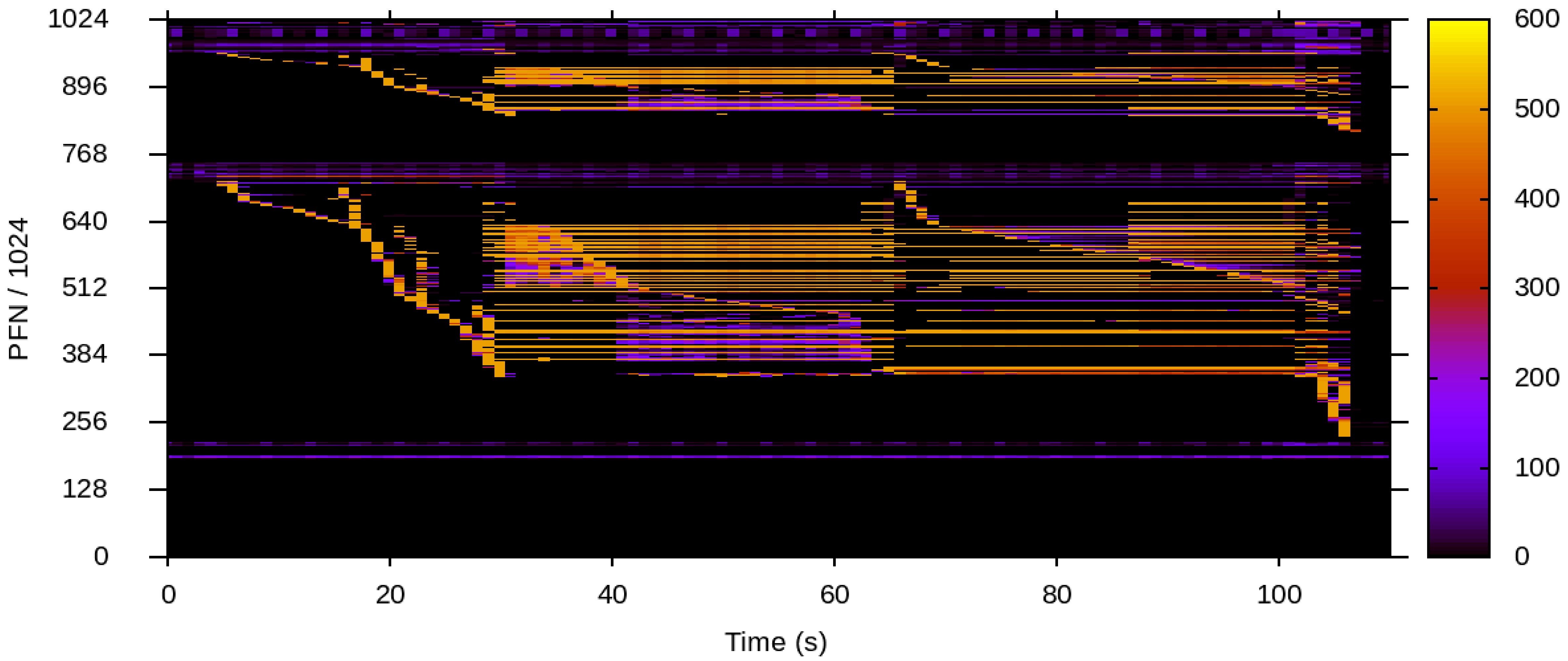
0x5000 1044 0140 0000 & (1<<52) = 0, i.e. page was accessed.

[1] available since Linux 4.3, see Documentation/mm/idle_page_tracking.txt

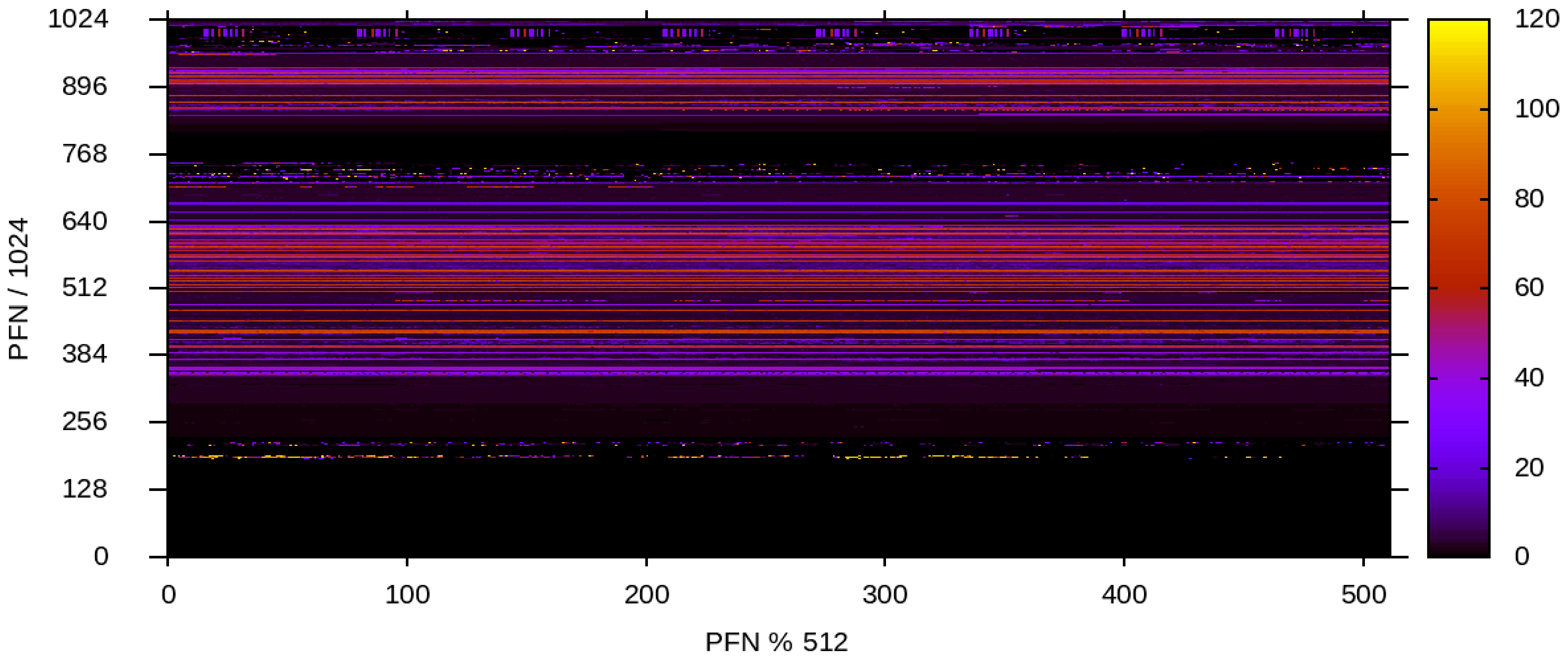
Example 1: git clone in Linux

```
[root@localhost ~]# git clone https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
Cloning into 'linux'...
remote: Counting objects: 6252839, done.
remote: Compressing objects: 100% (727/727), done.
remote: Total 6252839 (delta 479), reused 212 (delta 163)
Receiving objects: 100% (6252839/6252839), 1.05 GiB | 18.92 MiB/s, done.
Resolving deltas: 100% (5258620/5258620), done.
Checking out files: 100% (61733/61733), done.
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# ls linux
arch   COPYING  Documentation  fs       ipc      kernel    MAINTAINERS  net      scripts   tools
block  CREDITS  drivers        include  Kbuild   lib       Makefile     README   security  usr
certs  crypto   firmware       init     Kconfig  LICENSES  mm           samples  sound     virt
[root@localhost ~]# _
```

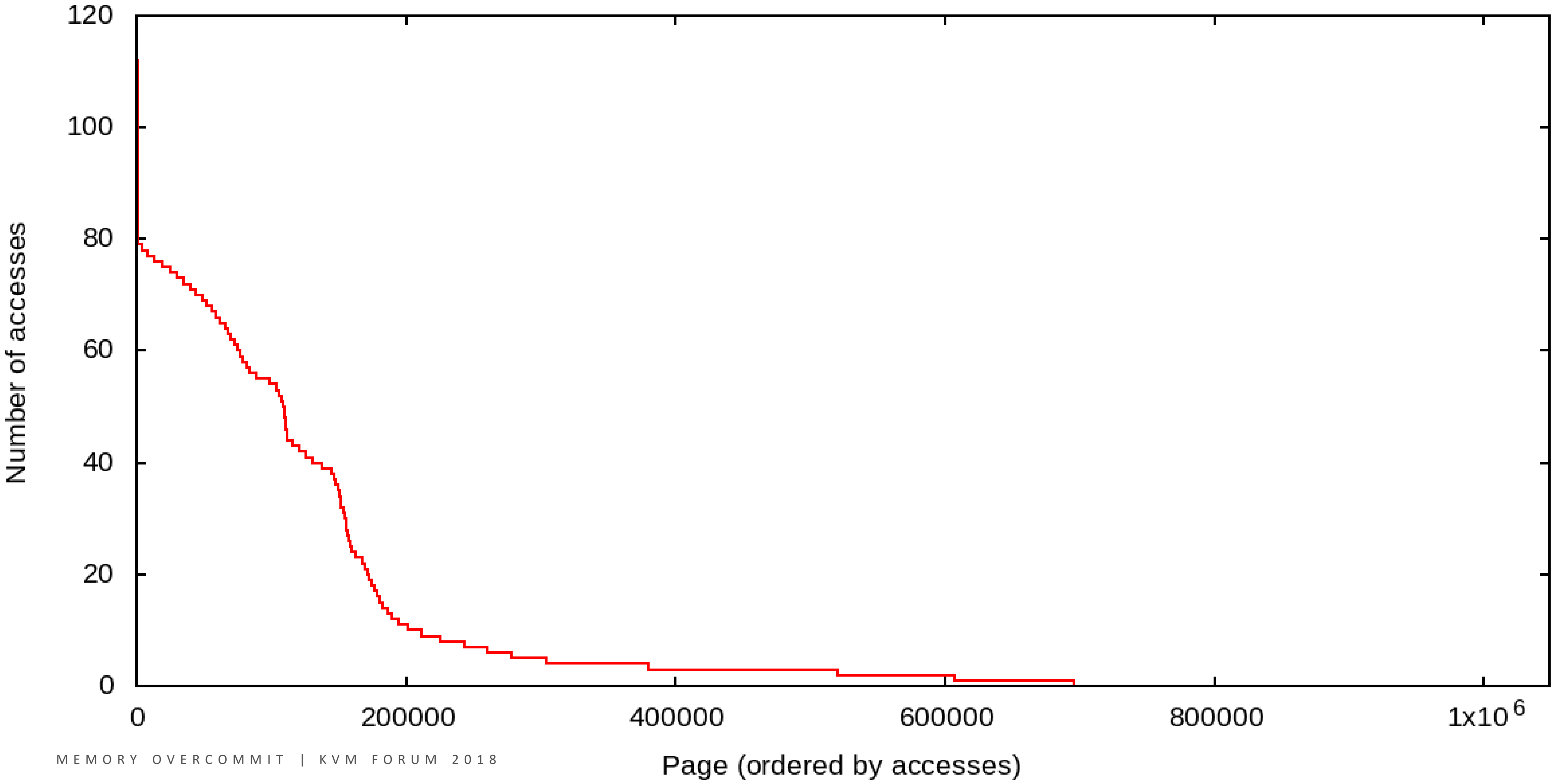
Example 1: git clone in Linux



Example 1: git clone in Linux



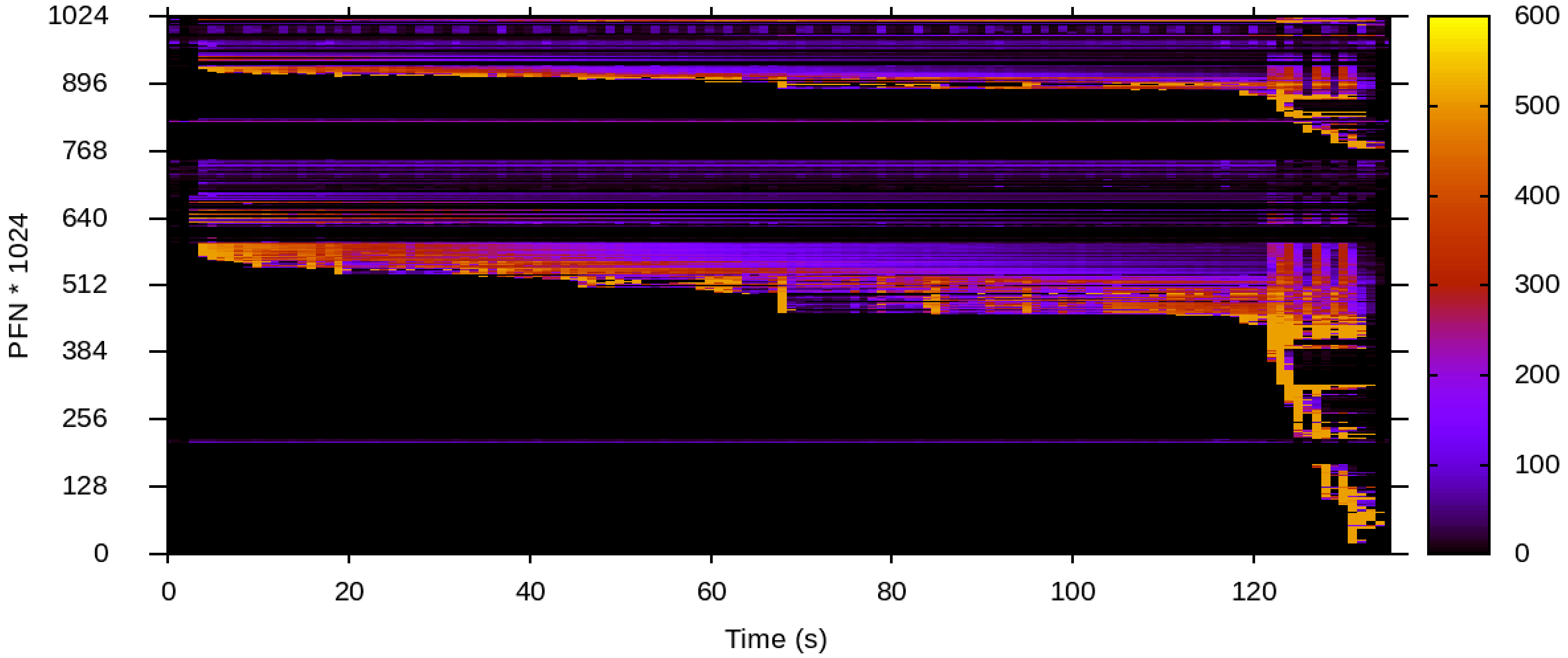
Example 1: git clone in Linux



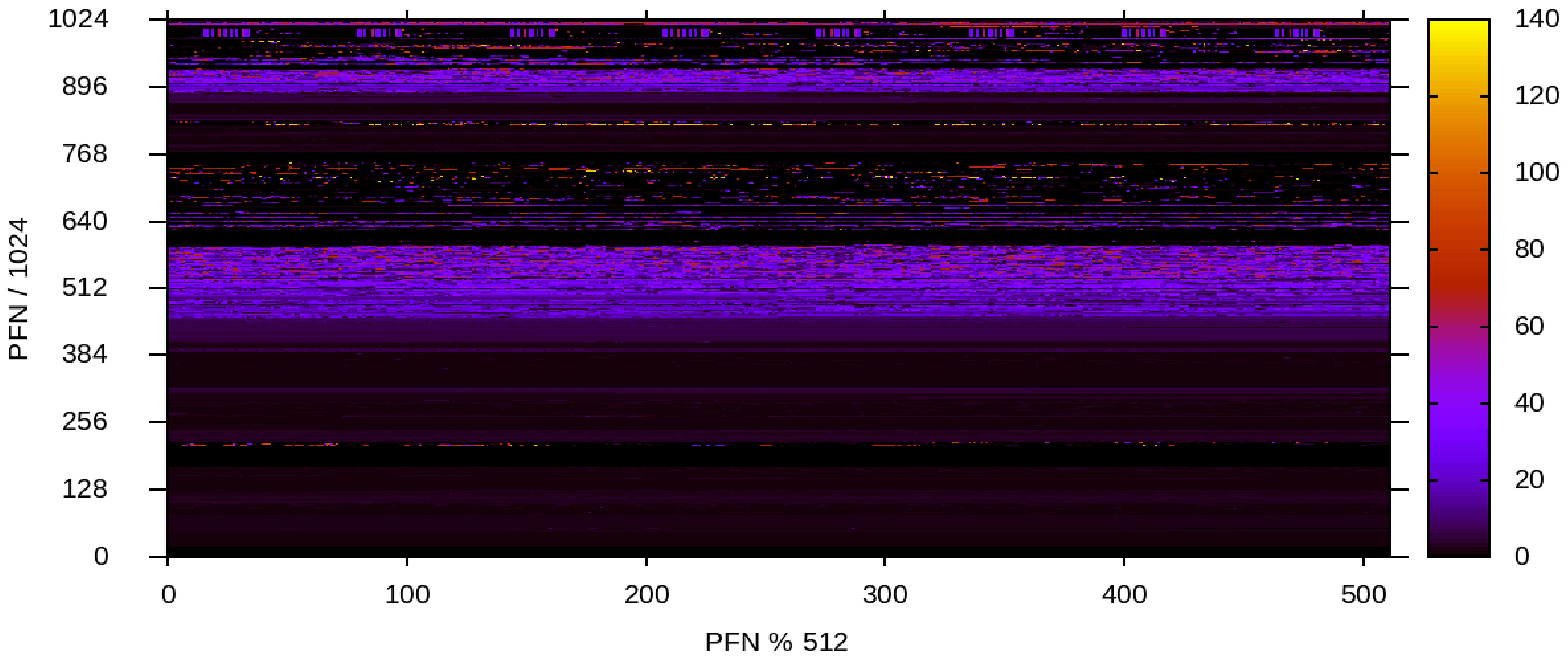
Example 2: kernel build in Linux

```
  AS      arch/x86/boot/pmjump.o
  CC      arch/x86/boot/printf.o
  CC      arch/x86/boot/regs.o
  LDS     arch/x86/boot/compressed/vmlinux.lds
  AS      arch/x86/boot/compressed/head_64.o
  CC      arch/x86/boot/string.o
  CC      arch/x86/boot/tty.o
  VOFFSET arch/x86/boot/compressed/../voffset.h
  CC      arch/x86/boot/video.o
  CC      arch/x86/boot/video-mode.o
  CC      arch/x86/boot/version.o
  CC      arch/x86/boot/video-vga.o
  CC      arch/x86/boot/video-vesa.o
  CC      arch/x86/boot/video-bios.o
  HOSTCC  arch/x86/boot/tools/build
  CPUSTR  arch/x86/boot/cpustr.h
  CC      arch/x86/boot/cpu.o
  CC      arch/x86/boot/compressed/string.o
  CC      arch/x86/boot/compressed/cmdline.o
  CC      arch/x86/boot/compressed/error.o
  OBJCOPY arch/x86/boot/compressed/vmlinux.bin
  RELOCS  arch/x86/boot/compressed/vmlinux.relocs
  HOSTCC  arch/x86/boot/compressed/mkpiggy
  CC      arch/x86/boot/compressed/cpuflags.o
  CC      arch/x86/boot/compressed/early_serial_console.o
  CC      arch/x86/boot/compressed/kaslr.o
  CC      arch/x86/boot/compressed/pagetable.o
  AS      arch/x86/boot/compressed/mem_encrypt.o
  CC      arch/x86/boot/compressed/pgtable_64.o
  CC      arch/x86/boot/compressed/eboot.o
  AS      arch/x86/boot/compressed/efi_stub_64.o
  CC      arch/x86/boot/compressed/misc.o
  GZIP    arch/x86/boot/compressed/vmlinux.bin.gz
  MKPIGGY arch/x86/boot/compressed/piggy.S
  AS      arch/x86/boot/compressed/piggy.o
  DATAREL arch/x86/boot/compressed/vmlinux
  LD      arch/x86/boot/compressed/vmlinux
  OBJCOPY arch/x86/boot/vmlinux.bin
  ZOFFSET arch/x86/boot/zoffset.h
  AS      arch/x86/boot/header.o
  LD      arch/x86/boot/setup.elf
  OBJCOPY arch/x86/boot/setup.bin
  BUILD   arch/x86/boot/bzImage
Setup is 15820 bytes (padded to 15872 bytes).
System is 6005 kB
CRC 265558d7
Kernel: arch/x86/boot/bzImage is ready  (#1)
[root@localhost linux]# _
```

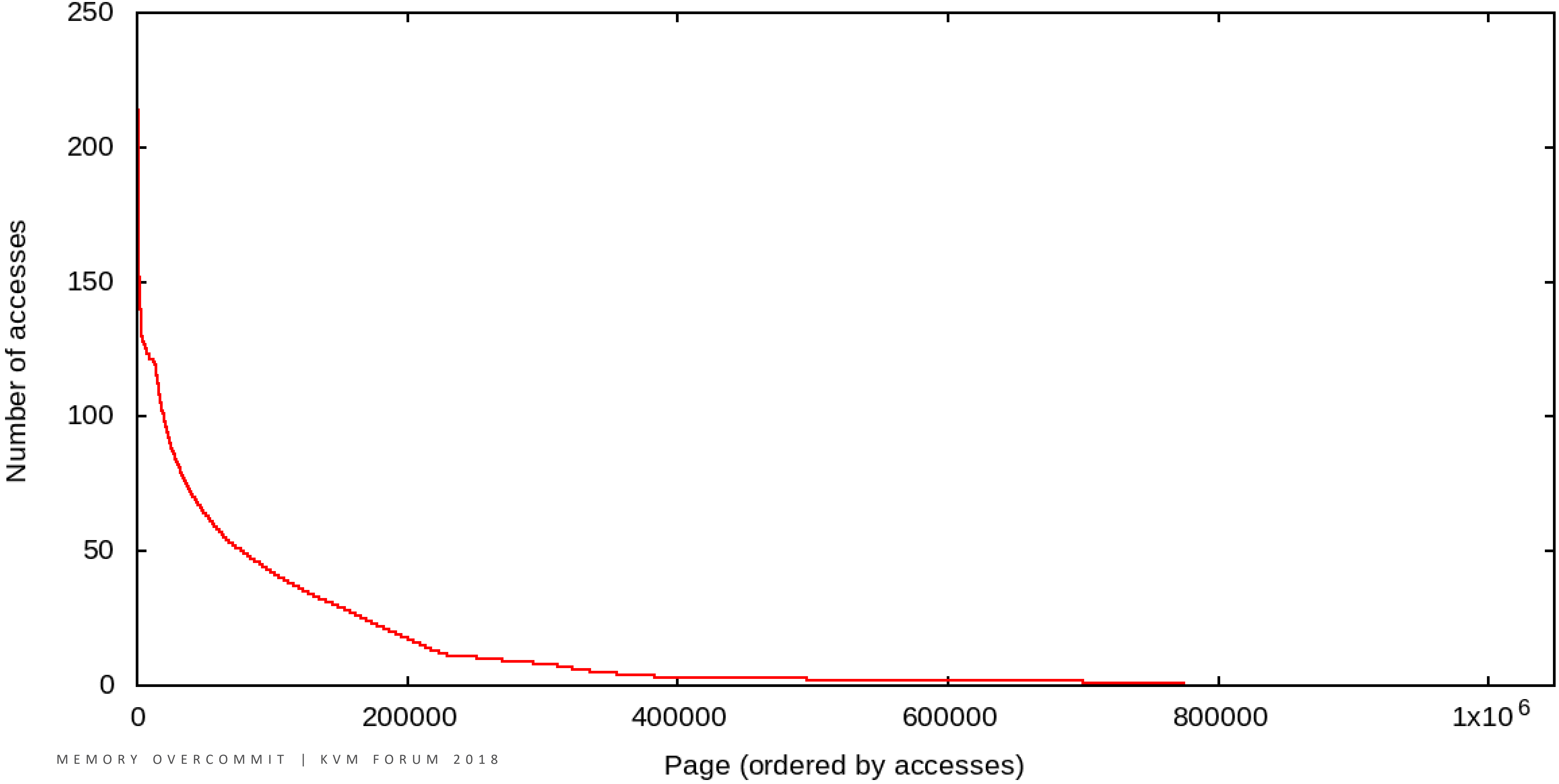
Example 2: kernel build in Linux



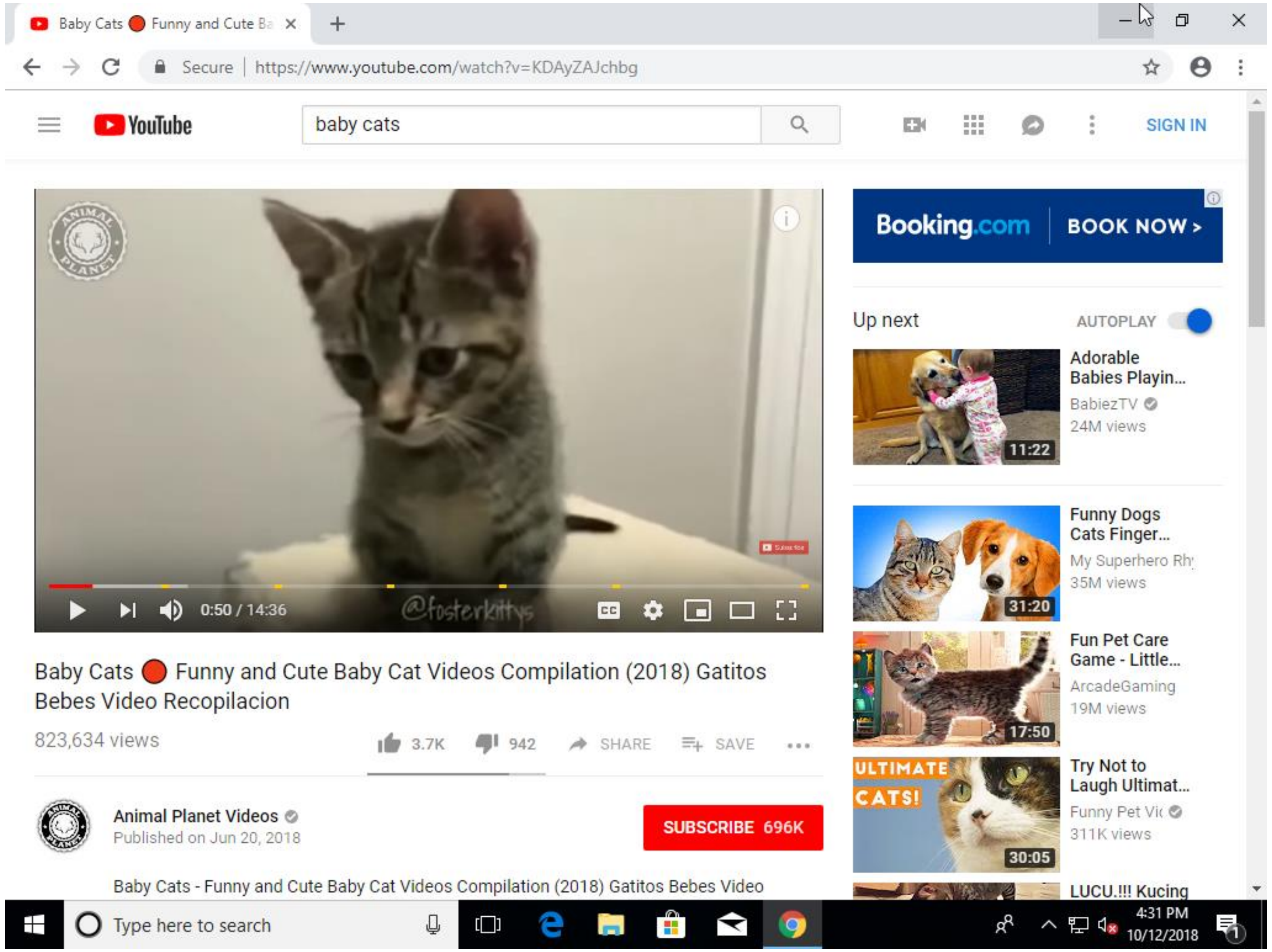
Example 2: kernel build in Linux



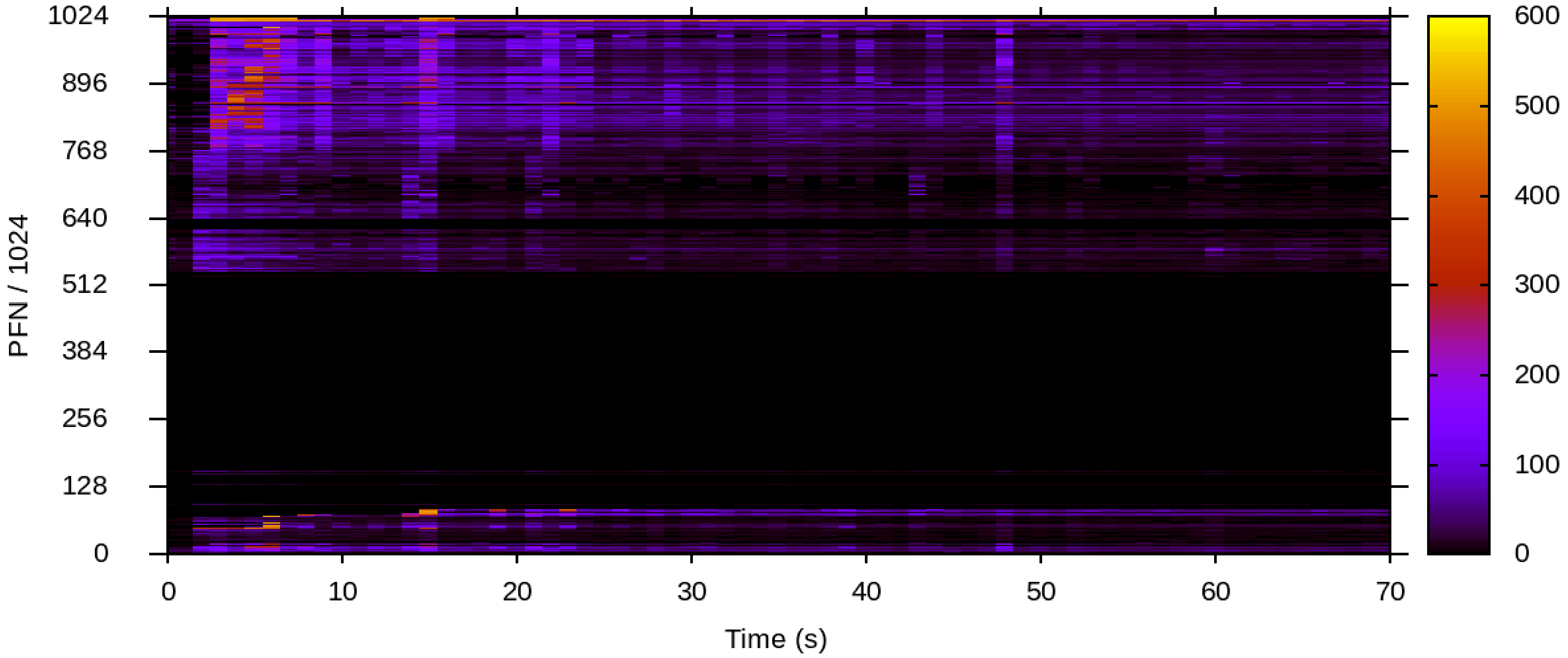
Example 2: kernel build in Linux



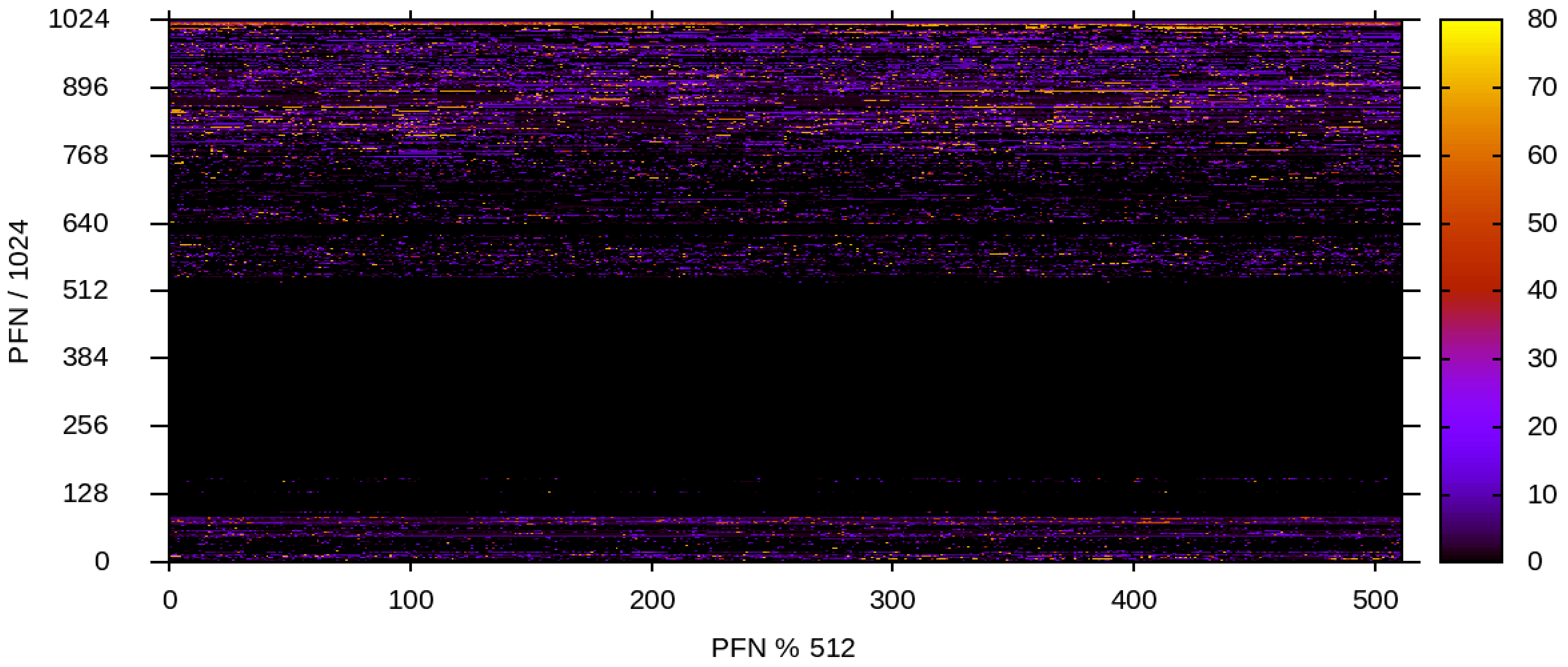
Example 3: Youtube video in Windows



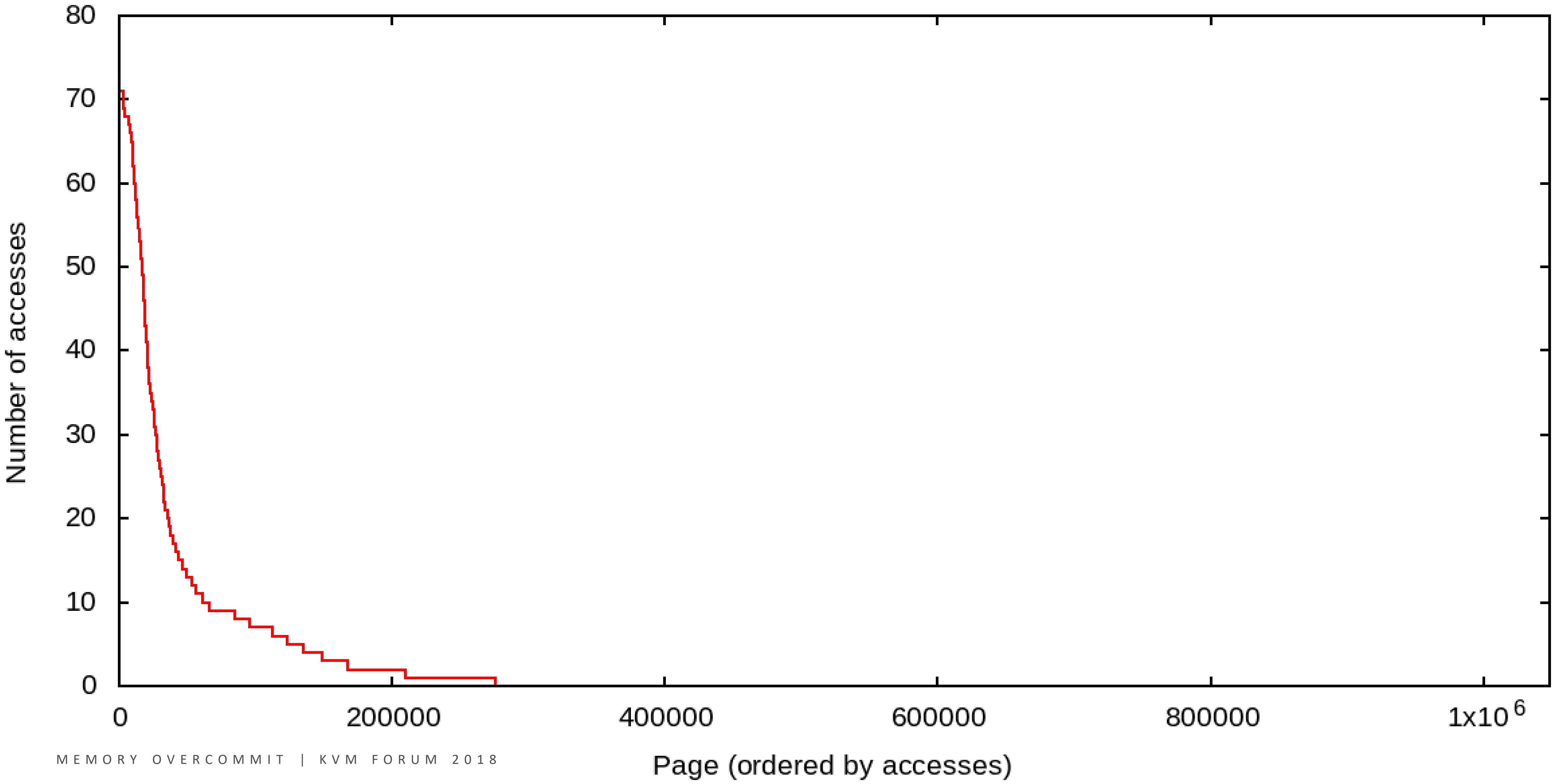
Example 3: Youtube video in Windows



Example 3: Youtube video in Windows



Example 3: Youtube video in Windows



Converting into a model

A simple model that approximates these three data sets:

- Split PFNs into two sets: used and unused
- Used pages will be accessed according to a normal distribution
- Unused pages will not be accessed

We can model variation over time by varying the model's parameters.

Linux memory subsystem simulator

We have created **psim** to simulate a VM performing a workload and measure the amount of work done in a given time

- https://github.com/jjd27/psim

For every simulator tick:

- Workload determines which page the vCPU accesses
  - Accessing a free or unmapped page incurs a minor fault, blocking the vCPU from doing work for a short time
  - Accessing a paged-out page incurs a major fault, blocking the vCPU from doing work for a prolonged time
- vCPU does a unit of work on the page

Simulate Linux memory subsystem behaviour:

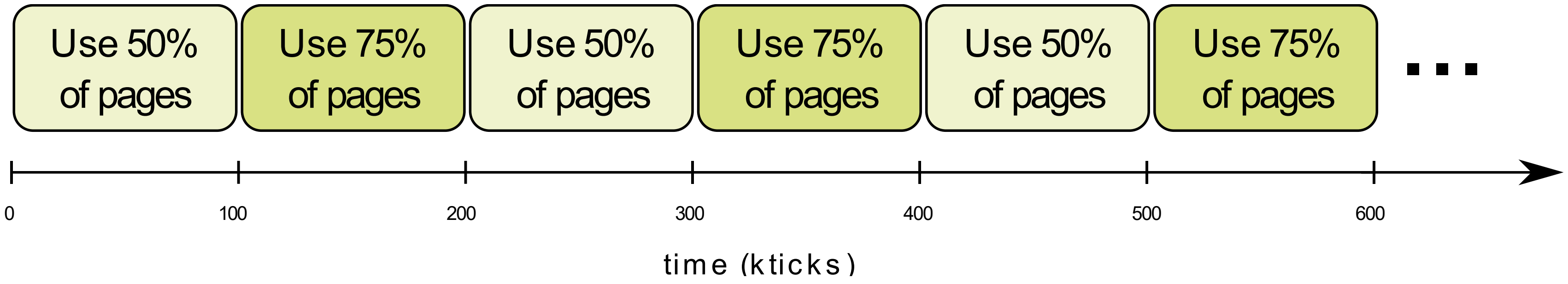
- Periodically scan memory to update active/inactive lists according to page accesses since last scan
  - Page out inactive pages if the number of allocated pages exceeds cgroup memory limit

Further reading: Mel Gorman, Understanding the Linux Virtual Memory Manager (chapter 10), http://www.kernel.org/doc/gorman/

Simulated workload for rapid prototyping

An example workload, looping forever:

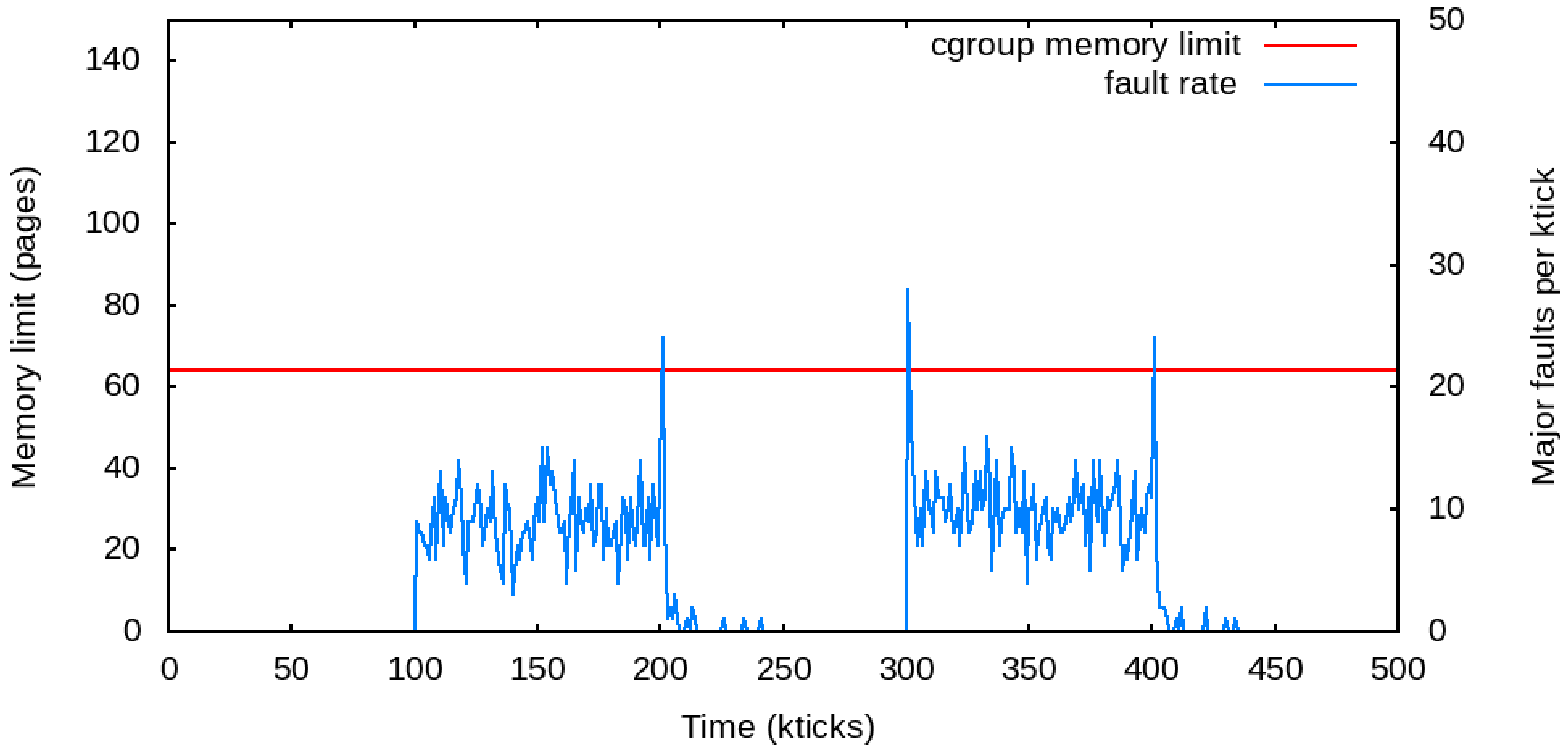
- Access 50% of pages according to a normal distribution (one page per tick for 100k ticks)
- Access 75% of pages according to a normal distribution (one page per tick for 100k ticks)



Run the squeezer algorithm every 1k ticks

Static squeezing

Algorithm: Keep page limit constant at half the total number of pages

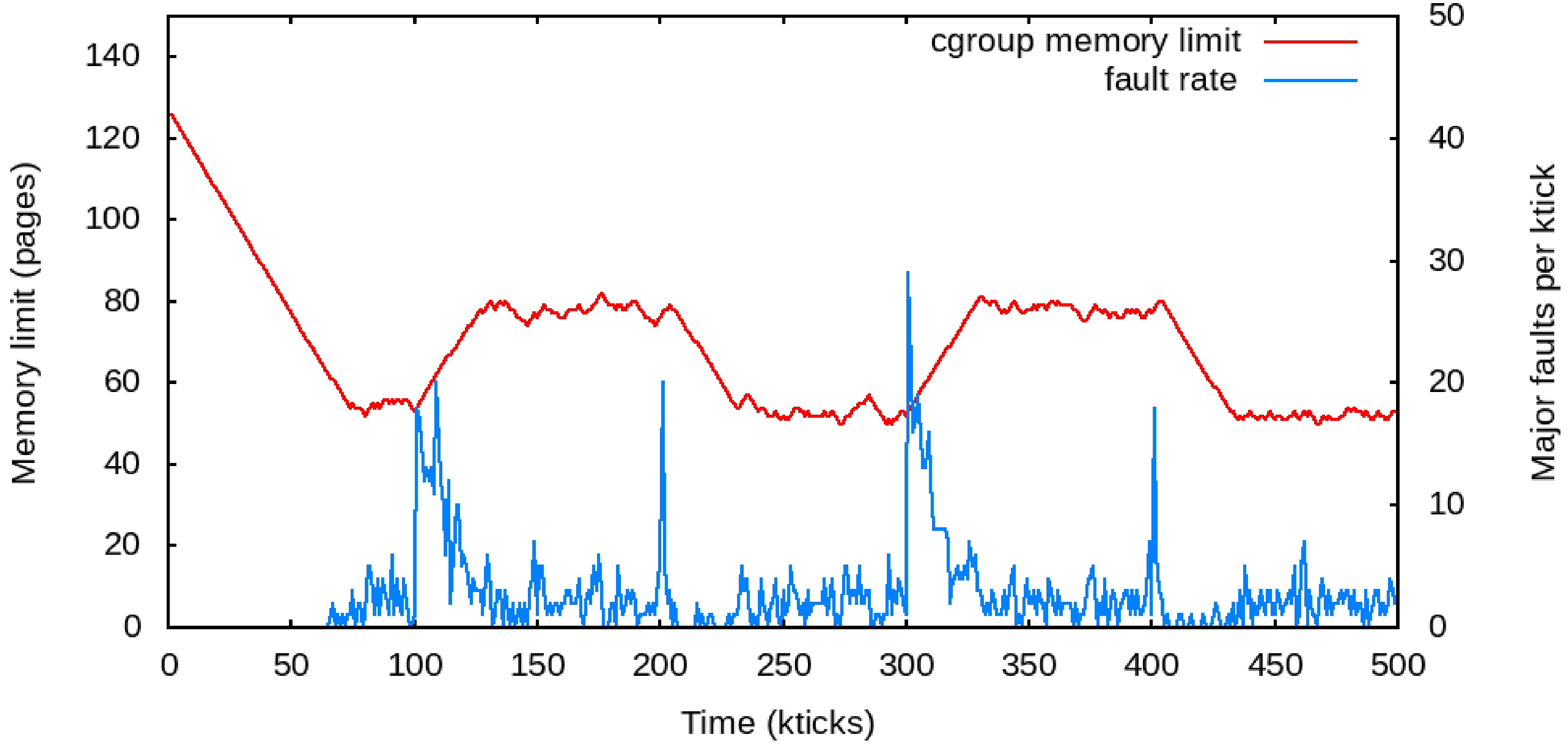


Memory limit = 64 pages, i.e. **50%** of static allocation

Total work done = 436k units of work, i.e. **87.1%** efficiency

Simple automatic squeezing

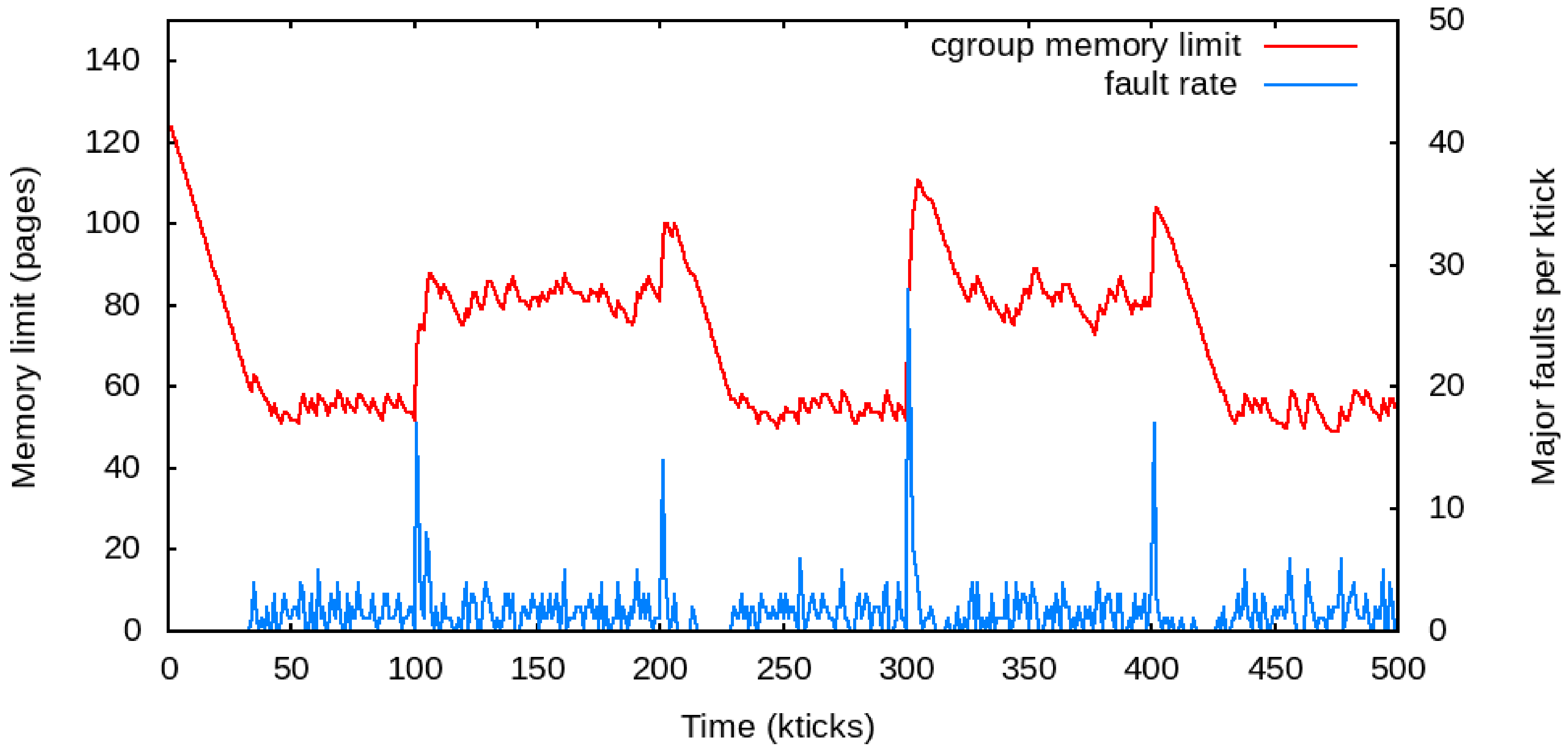
Algorithm: Reduce/increase limit by 1 page if less/more than 2 major faults in period



Average memory limit = 68.9 pages, i.e. **53.8%** of static allocation
Total work done = 460k units of work, i.e. **92.0%** performance

Proportional squeezing

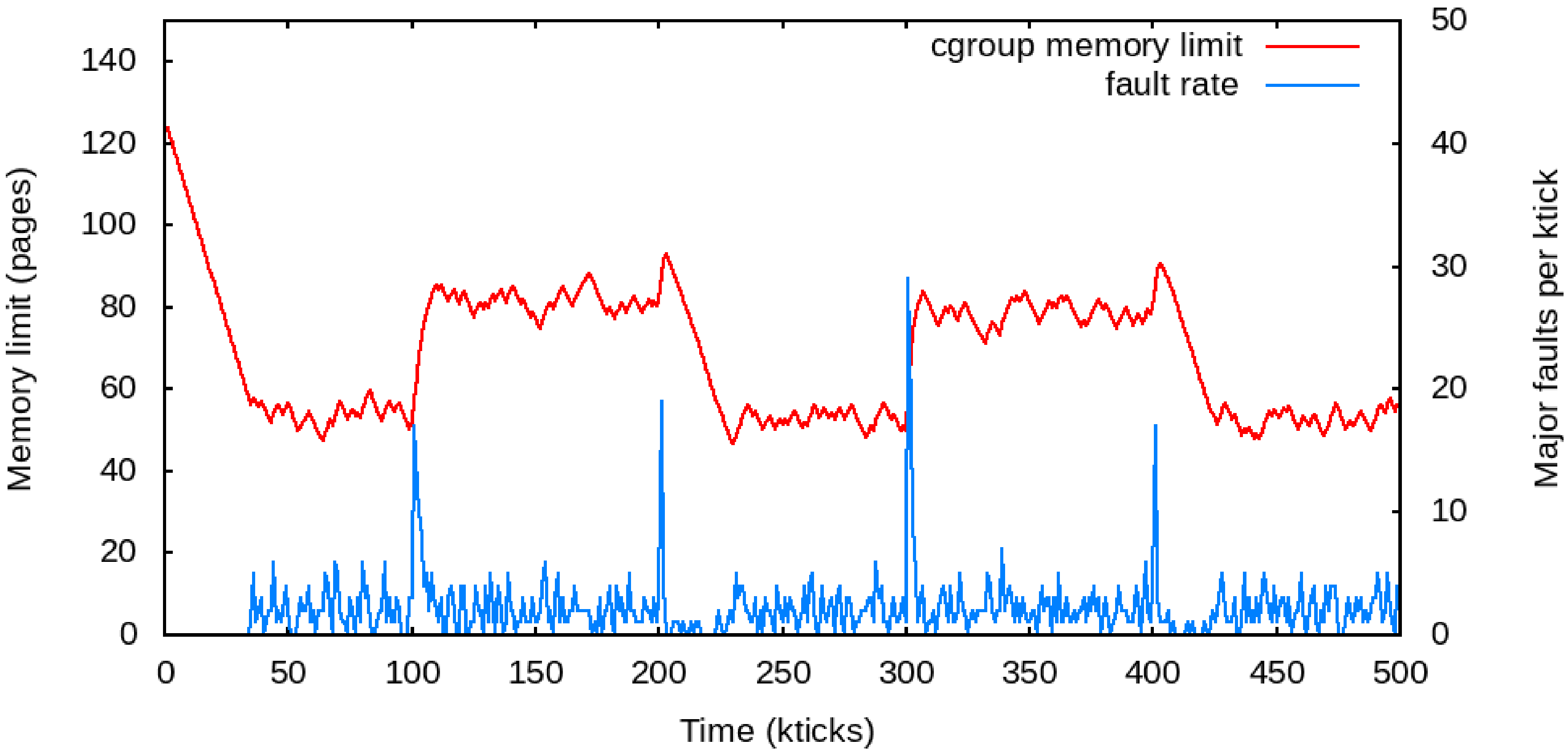
Algorithm: Reduce/increase limit by number of major faults in period



Average memory limit = 71.9 pages, i.e. **56.2%** of static allocation
Total work done = 477k units of work, i.e. **95.3%** performance

Smooth out recent fault rate

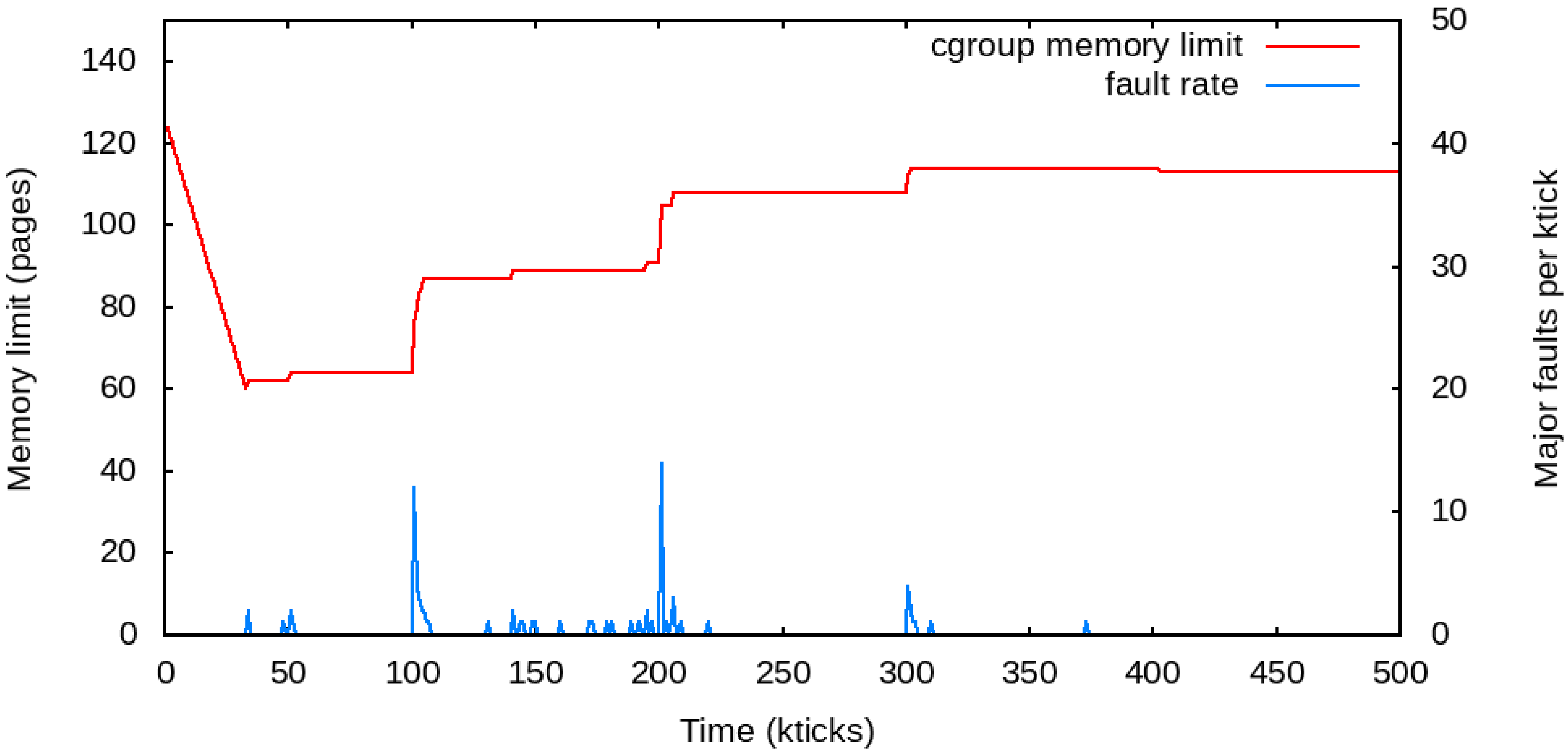
Algorithm: Reduce/increase limit by number proportional to weighted average of the number of faults since the start



Average memory limit = 68.4 pages, i.e. **53.4%** of static allocation
Total work done = 468k units of work, i.e. **93.5%** performance

Or, be more conservative

Algorithm: Hard squeeze until first page fault; soft squeeze if we get fewer than 2 faults in 100 intervals



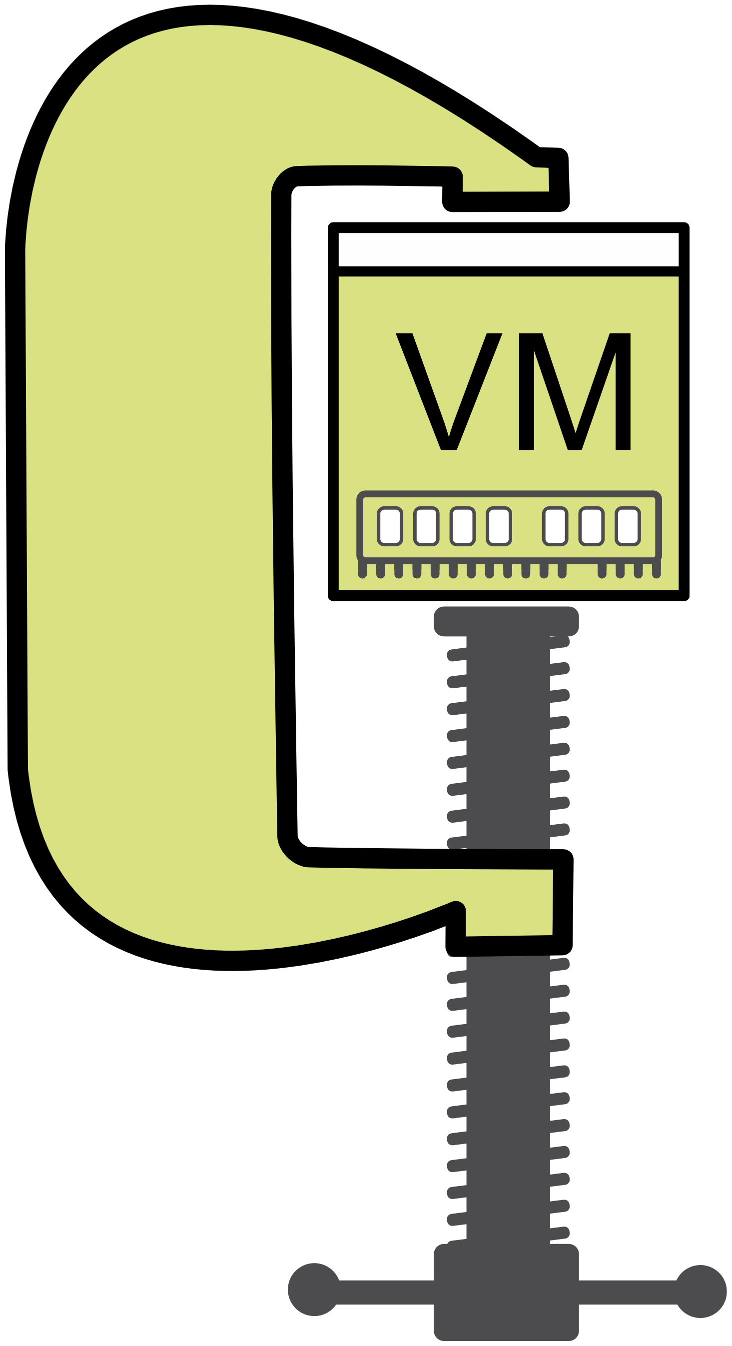
Average memory limit = 102.7 pages, i.e. **80.2%** of static allocation
Total work done = 497k units of work, i.e. **99.4%** performance

Outline

1. Memory overcommit: Why? How?
2. Design approach
3. Rapid prototyping by simulation
4. **Conclusion**

Conclusions

- Memory overcommit is beneficial for some use-cases and workloads
- Hypervisor paging does not require guest co-operation
- Automatic VM memory sizing avoids administrative effort and guesswork
- A squeezer algorithm can be evaluated using simulation based on real-world data
- A good squeezer algorithm can maintain good levels of VM performance while squeezing VMs close to their working set



Memory overcommit for overcommitted admins

Presented by Jonathan Davies

Based on the work of David Vrabel

OCTOBER 2018 | KVM FORUM