Memory Resource Controller

Edition:Oct/2009

Japan Linux Symposium 22/Oct/2009

Kame
kamezawa.hiroyu@jp.fujitsu.com

Contents

- Background
- Memory Resource Controller
  - Basic Concepts
  - Charge/Uncharge
  - LRU
  - Performance
  - TODO List

Background

- In 90's, many studies for OS-level resource control on big servers and slow/small machines.
- In 00's, fast PC/network + cluster control
- In these years
  - Multi-core CPUs.
  - Memory is getting less expensive. 64Bit systems allow us to use more memory.
  - Virtual Machine is now popular.
  - Hmm....OS-level resource controls for Linux ?

    There will be users.
    - OpenVZ, Linux Vserver etc...

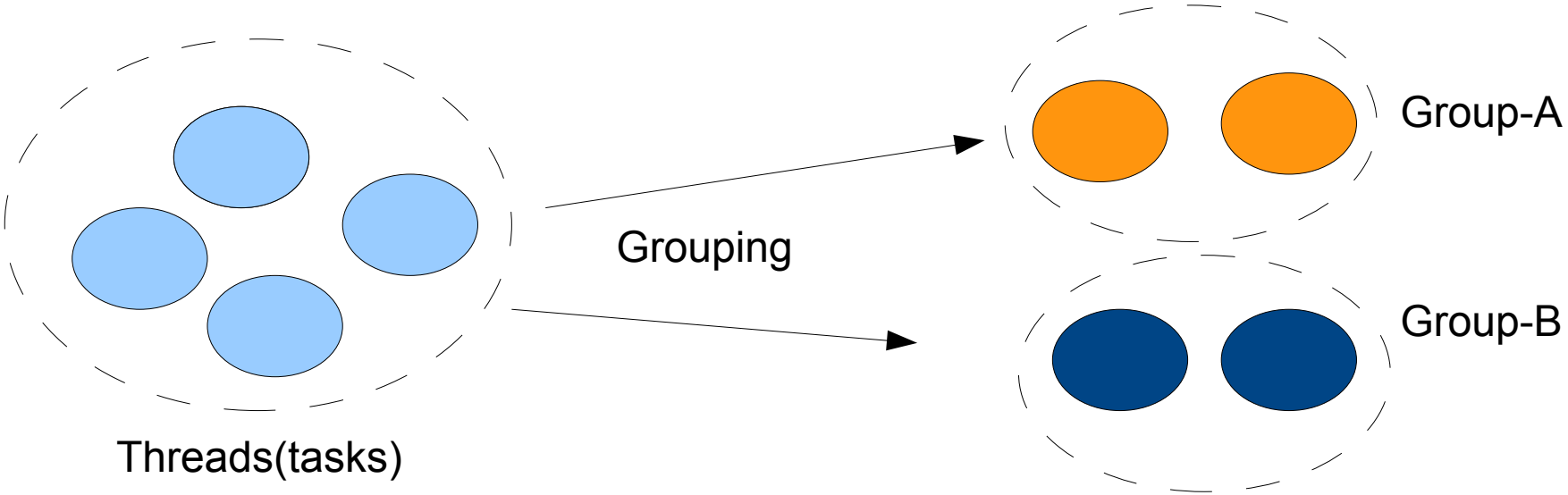
Cgroup

Several proposals were done....

Paul Menage(@google) finally implemented "Cgroup" as base technology for control.

Cgroup

- Cgroup is for putting processes into groups.
- Characteristics
  - Implemented as pseudo filesystem.
  - Grouping can be done by a unit of thread.
  - Functions are implemented as selectable options, “subsystem”.



Cgroup interface

- mount

```
# mount -t cgroup none /cgroup -o subsystem
```

- mkdir (create a group)

```
# mkdir /cgroup/group-A
```

- rmdir (destroy a group)

```
# rmdir /cgroup/group-A
```

- attach a task

```
# echo <PID> > /cgroup/group-A/tasks
```

libcgroup provides automatic configuration based on user defined rules and sophisticated interface. But not shown in this slide.

Cgroup Subsystems(1)

- Can be specified as mount option of cgroupfs.

  ex) #mount -t cgroup none /cgroup -o cpu

- 2 types of subsystem in general

  A) Resource control … cpu, memory, I/O,

  B) Isolation and special controls

  cpuset, namespace, freezer, device, checkpoint/restart

Cgroup subsystems(2)

- Ex) mount each subsystem independently

```
# mount -t cgroup none /cpu -o cpu
# mount -t cgroup none /memory -o memory
```

- Ex) mount at once

```
# mount -t cgroup none /cgroups -o cpu, memory,
```

Cgroup's feature is determined how it equips subsystems.

Contents

- Background
- Memory Resource Controller
  - Basics
  - Charge/Uncharge
  - LRUs
  - Performance
  - TODO List

Memory resource control

Basic concept is...

- Accounting memory usage under cgroup
  - Memory here is physical memory.
- Limit memory usage under user specified value.
- If necessary, cull(pageout) memory under it.

Memory Cgroup is often called as memcg.
It's been almost 2 years since the first patch is merged.
Config is CONFIG_CGROUP_MEM_RES_CTRL.
See mm/memcontrol.c.

Features of memory cgroup

- Limiting memory
  - anonymous(anon) and file-caches, swap-cache
  - When hit limit, cull memory.
- Limiting usage of memory+swap.
- Memory statistics per cgroup.
- SoftLimit per cgroup(hint for kswapd)

How to use.

Scenario: A user wants to get a big file but doesn't want unnecessary memory pressure to other process, file cache for copied file is not necessary.

```
# mount -t cgroup none /memory -o memory
# mkdir /memory/group01
# echo 128M > group01/memory.limit_in_bytes
# echo $$ > (...)/tasks
# wget  http://..... veryverybigfile
```

The amount of file cache doesn't exceed 128M.

How to use(memory+swap).

```
# mount -t cgroup none /memory -o memory
# mkdir /memory/group01
# echo 128M > (...)/memory.memsw. limit_in_bytes
```

Same to memory cgroup. Has memsw prefix. This limits the sum of usage of memory and swap.

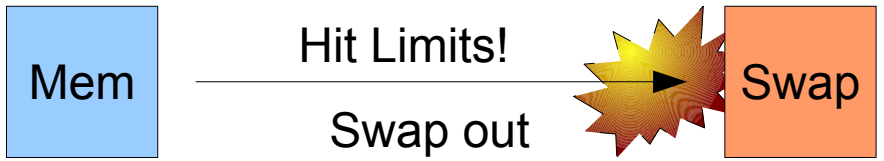
Use case)

Run a process with 10G of anonymous memory under 100MB memory limit can generate 9.9GBytes of swap. With Memory+Swap control, an administrator can prevent too much swap use.

Memory+Swap ?

Why Memory+Swap not swap-limit-controller ? Assume that kswapd tries to pageout a page at system memory shortage.

SwapUsage += PAGE_SIZE



Swap Limit controller

When swap usage hit limit, kswapd cannot free memory. This is just a brutal mlock().

No changes in accounting

Memory+Swap

Memory Usage -= PAGE_SIZE
Swap Usage += PAGE_SIZE
No change in total usage.

Kswapd will not be disturbed.

Contents

- Background
- Memory Resource Controller
  - Basics
  - Charge/Uncharge
  - LRU
  - Performance
  - TODO List

Charge and Uncharge

Memory cgroup accounts usage of memory. There are roughly 2 operations, charge/uncharge.

- Charge
  - (Memory) Usage += PAGE_SIZE
  - Free/cull memory if usage hit limits
  - Check a page as “This page is charged”

- Uncharge
  - (Memory) Usage -= PAGE_SIZE
  - Remove the check

mm owner

There is a gap.

- Cgroup is based on thread.
- Memory is maintained per process, not thread.

When **CONFIG_CGROUP_MEM_RES_CTLR=y**

mm_struct->owner (points to one of threads in a process)

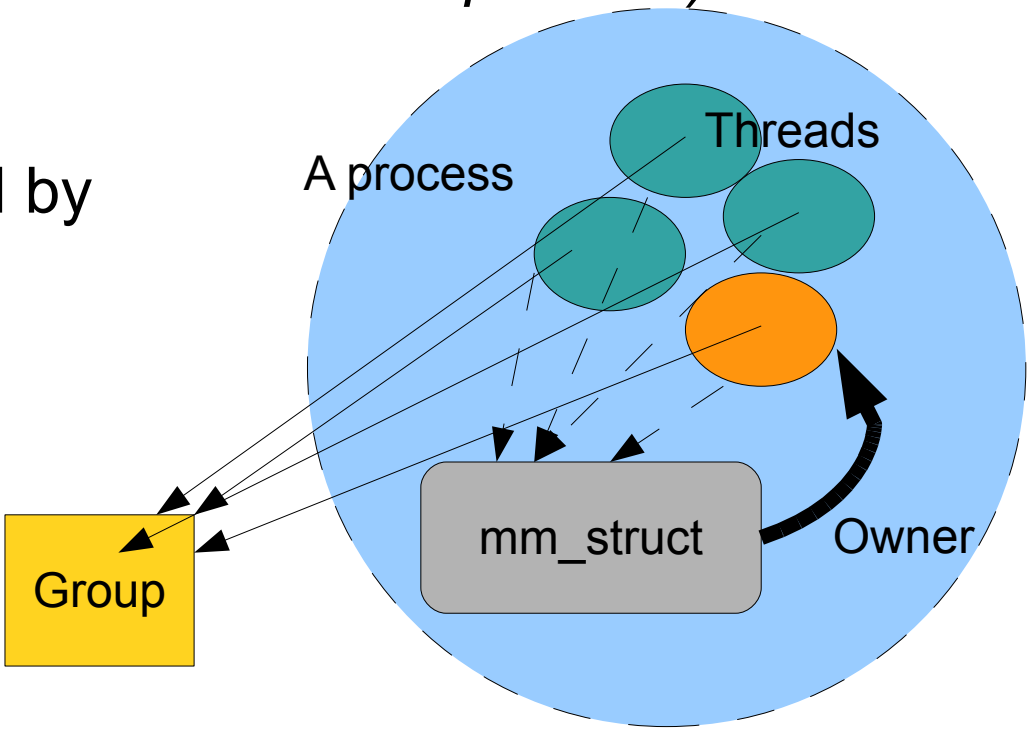
is added to mm_struct.

Memcg of a thread can be found by

thread->mm->owner->cgroup

In usual, mm->owner is

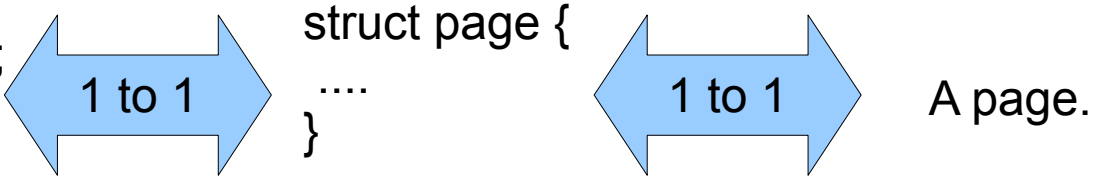
the thread group leader.



struct page_cgroup

Memcg uses page_cgroup for tracking all pages. It's allocated per page like struct page.

```
struct page_cgroup {
        unsigned long flags;
        struct mem_cgroup *mem_cgroup;
        struct page *page;
        struct list_head head;
};
```



struct page_cgroup occupies 40bytes/4096bytes(x86-64), 1% of memory.
Even if CONFIG_CGROUP_MEM_RES_CTRL=y,
this can be turned off by boot option.

In flags field

PCG_LOCK. for lock_page_cgroup()

PCG_USED bit in page_cgroup->flags indicates a page_cgroup is charged.

Types of charges.

For explanation, classify charges into 3 types.

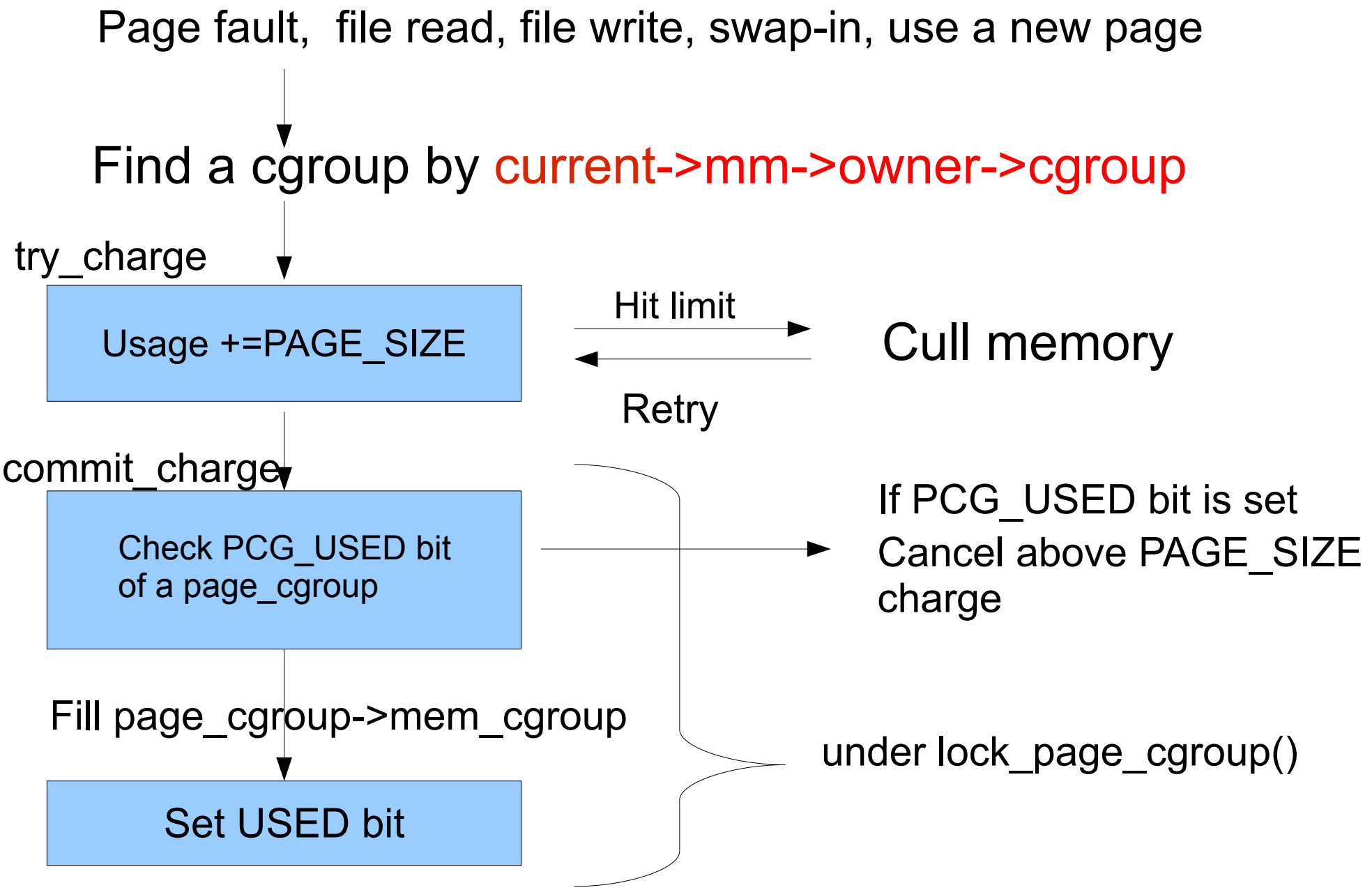
- Anonymous page.
- File Cache
- SwapCache

We track only pages on LRU, which can be reclaimed.

Then, slab,hugepage, etc...are not handled.

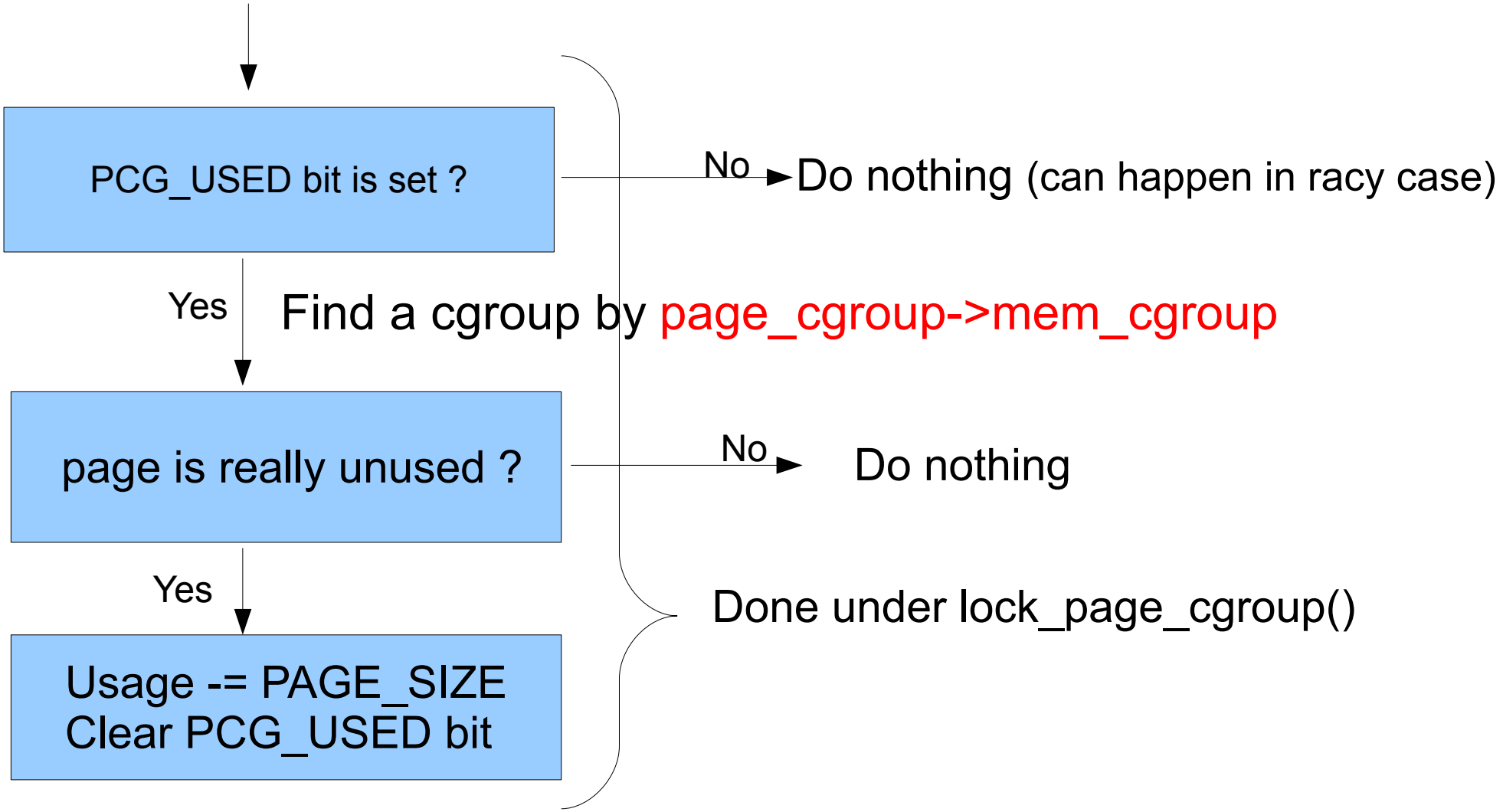
(I wonder pages not on LRU should be handled in other cgroups....if necessary. But no idea, yet.)

Charge


Page fault, file read, file write, swap-in, use a new page
Find a cgroup by current->mm->owner->cgroup
try_charge
Usage +=PAGE_SIZE
Hit limit
Retry
Cull memory
commit_charge
Check PCG_USED bit
of a page_cgroup
If PCG_USED bit is set
Cancel above PAGE_SIZE
charge
Fill page_cgroup->mem_cgroup
under lock_page_cgroup()
Set USED bit

Uncharge

Unmap, exit, truncate file,drop cache, kswapd.....freeing a page



Charge for anon.

After a new page allocation.

```
page = alloc_page(gfp)

ret = mem_cgroup_newpage_charge(page);
if (ret == -ENOMEM)
        ..........
```

You can see this in page allocation pass in page fault.

This means an anon page is charged at its first mapping.

i.e. only when map_count changes from 0 to 1.

..... Nothing happens when a page is shared

Uncharge(anon)

An anon page is uncharged when its fully unmapped.
page_remove_rmap() is called when a page is unmapped.

```
page_remove_rmap()
{
    if (decrement page->mapcount ...the result is 0 ?) {
        .........
         if (PageAnon(page))
                mem_cgroup_uncharge_page(page);
    }
```

Uncharge when map_count changes from 1 to 0.
(*)If the page is SwapCache, it will not be uncharged here.

Charge (file cache)

At inserting a new page into page cache

```
add_to_page_cache_locked(mapping, page, gfp)
{
    ret = mem_cgroup_cache_charge(page);
    if (ret == -ENOMEM)
        .....
```

Accounted against the first user. Nothing happens when this page cache is accessed/mapped/unmapped.

Now,

- shmem requires special handling
- hugemem is ignored.
- No hooks in swapcache

Uncharge(File Cache)

- A file cache page is uncharged when it's removed from page cache
- Comparing “charge”, there are several callers.
  - remove_from_page_cache()
  - truncate()
  - remove_mapping()

    etc....

SwapCache(1)

When the kernel tries to swap out an anon page, make it as a cache-of-swap-entry. It's called as SwapCache.

[swap-out]

Make a page as swapcache → unmap → write out → free

[swap-in]

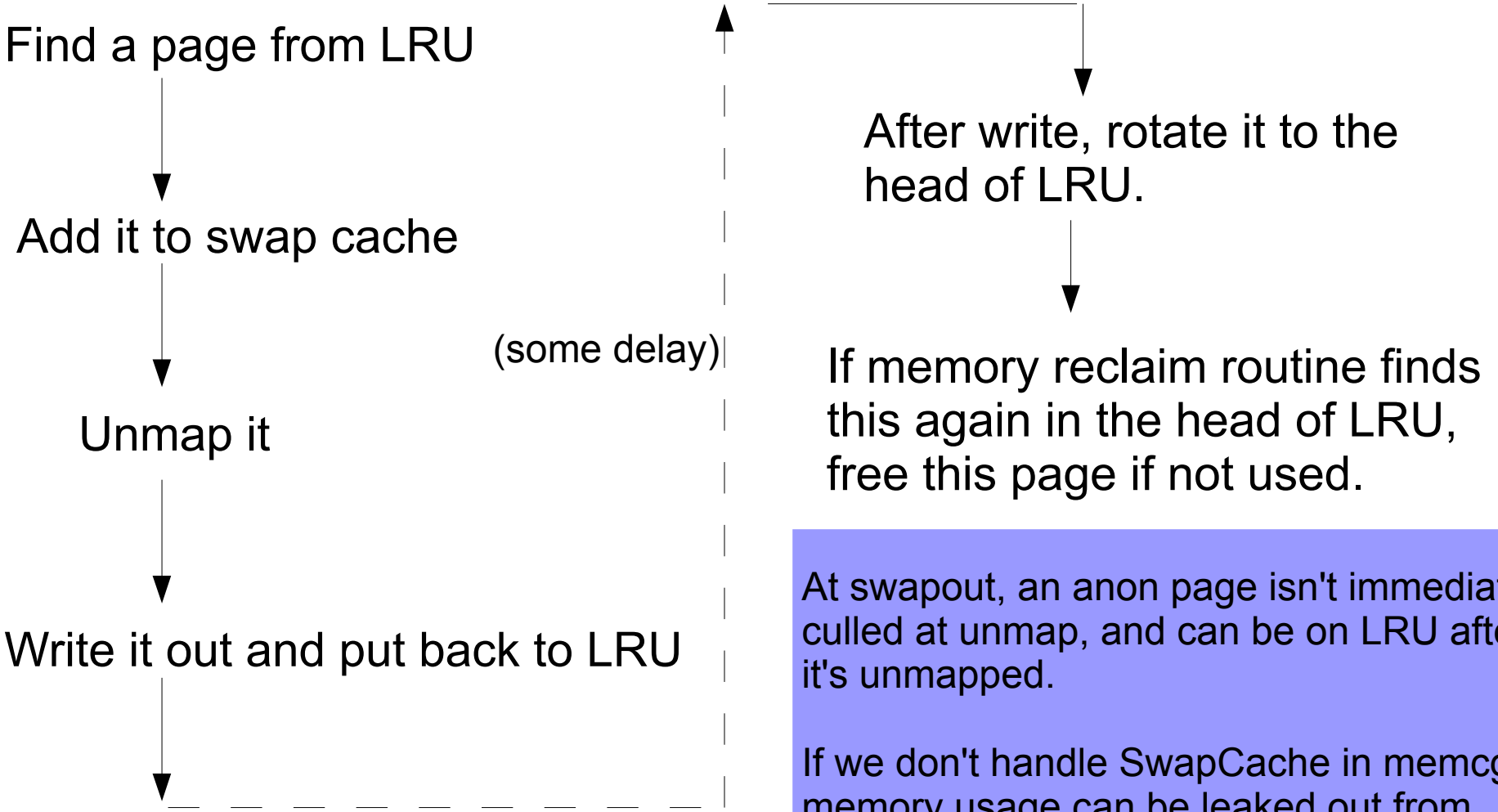
Alloc page → make it as swapcache → read from disk → map it.

Basic design

- Swapcache is uncharged when it is freed.
- Swapcache is charged when it's mapped.

SwapCache(2)

swap-out(pageout to swap) works as following.



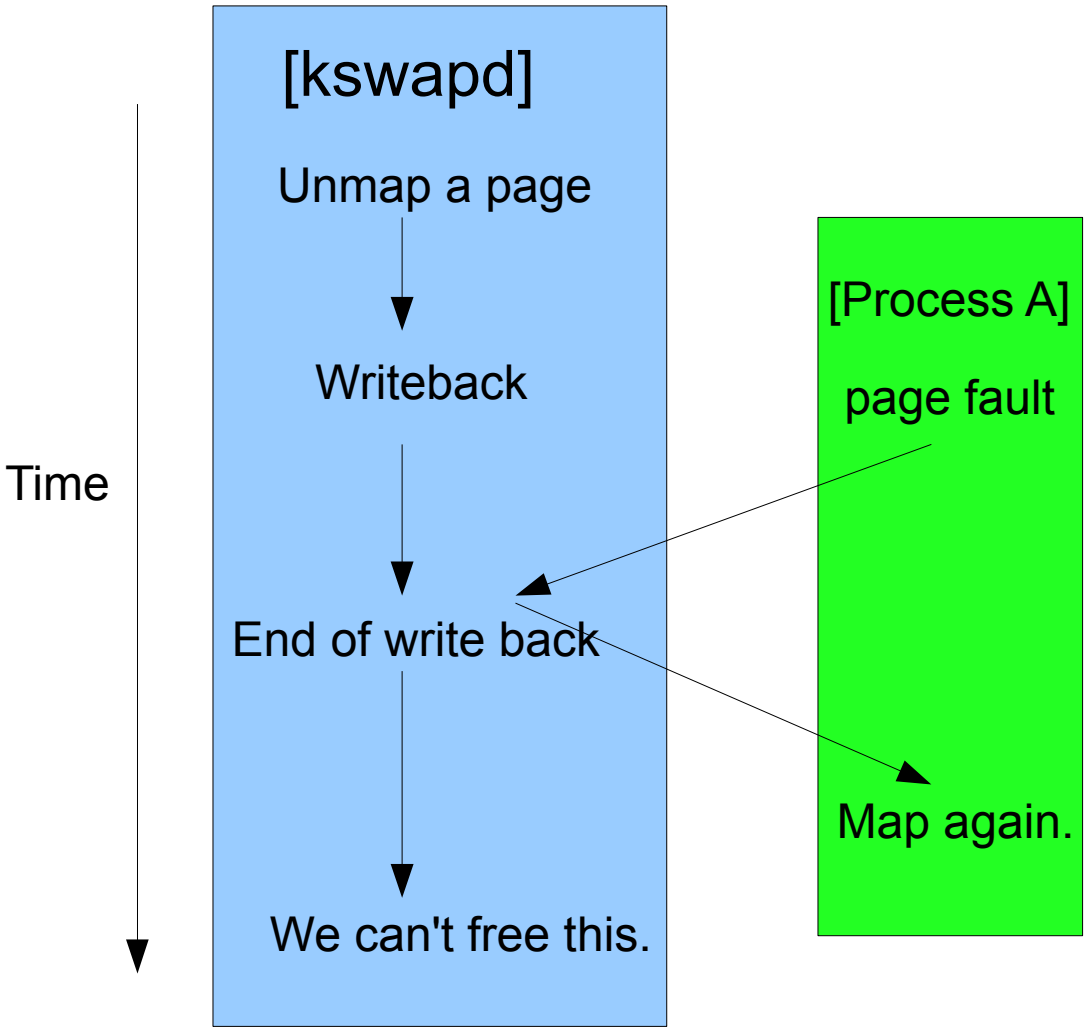
At swapout, an anon page isn't immediately culled at unmap, and can be on LRU after it's unmapped.

If we don't handle SwapCache in memcg, memory usage can be leaked out from memcg, very easily.

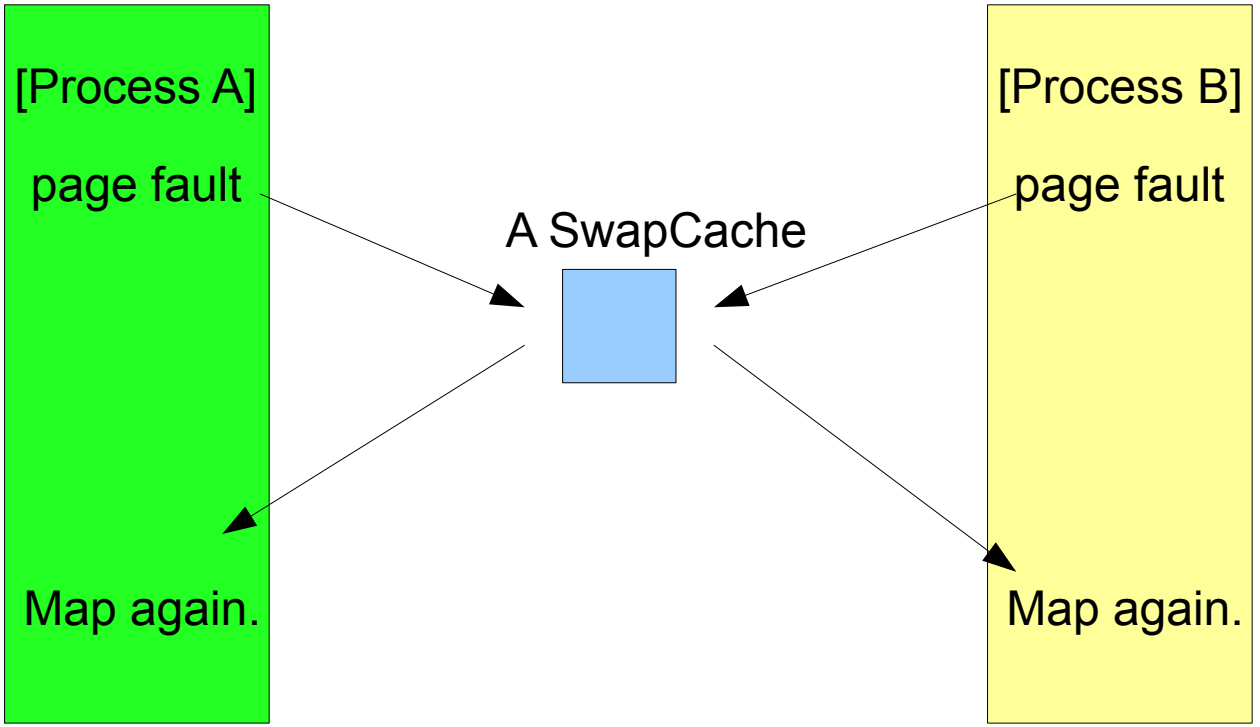
SwapCache(3)

When we account SwapCache.....there are some complicated cases.

Assume that a page is culled by kswapd but mapped again soon via page fault. In this case, we'll recharge against an “used” page.



SwapCache(4)



Many kinds of racy situation can be considered. We'll have to charge carefully against SwapCache. PCG_USED bit works well for us.

Charge(swapcache)

Following 2-phase call is used for avoiding race at mapping a page from SwapCache.

```
do_swap_page()
{
   Read swap and find swap cache.
    .......
    ret = mem_cgroup_try_charge_swapin(page);
    if (ret == -ENOMEM)
    ......
   page-table-lock.
   .......
   mem_cgroup_commit_charge_swapin(page);
```

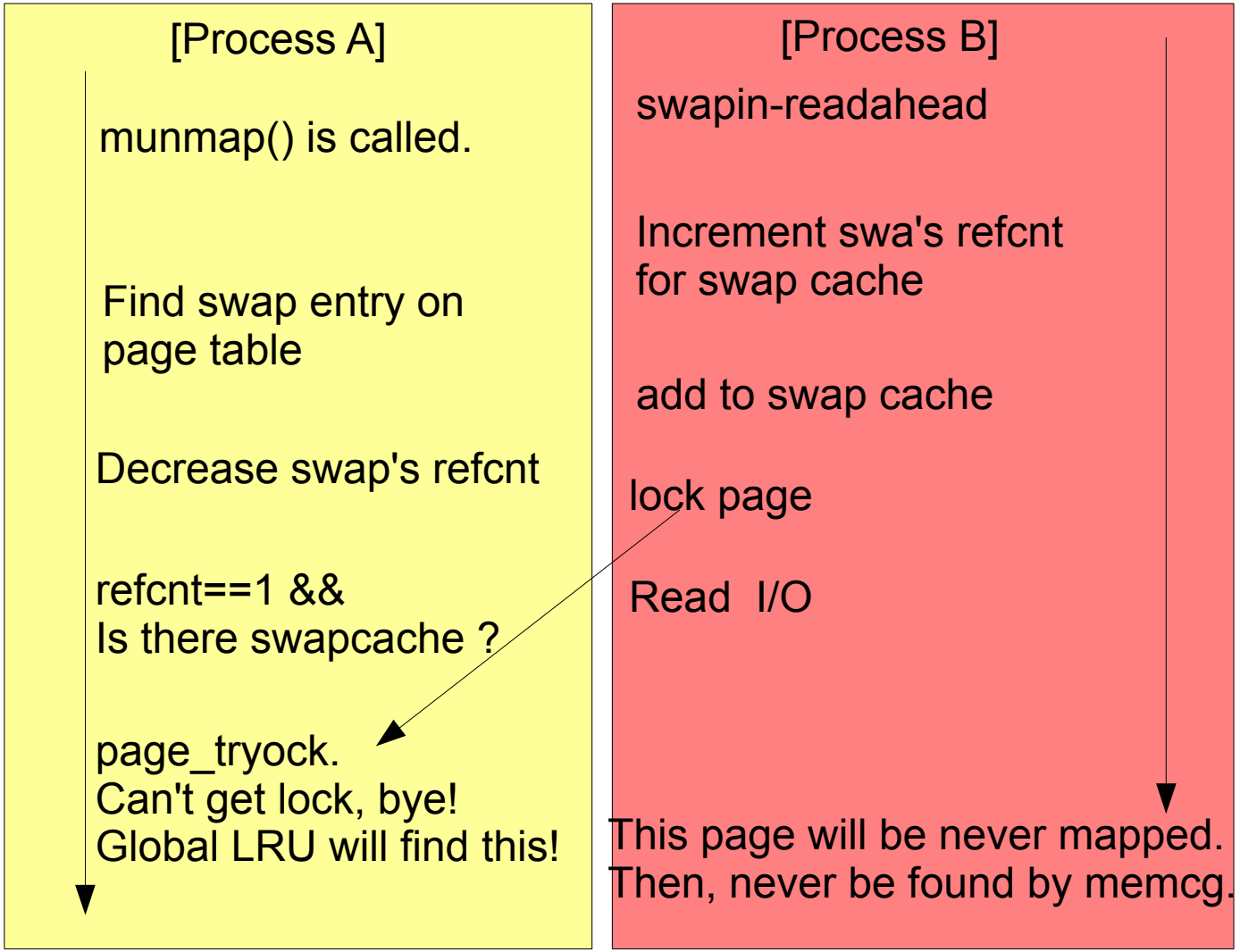
Charge PAGE_SIZE

Check PCG_USED

A SwapCache is charged when
its map_count changes from 0 to 1.
&&
it's not charged. i.e. PCG_USED bit is not set.

Freeing SwapCache

Because try_lock was used, there were races. (example)



If memcg is well used, global memory reclaim will not work often.

Then, pages on LRU but not caught by memcg Is hard to be freed.

In this example, swap entry is also leaked until global LRU runs...

(*) swapin readahead can read swap entry at random

Uncharge(SwapCache) (1)

- We can catch “Freeing SwapCache” at delete_from_swap_cache(). But we don't.
- There are try_lock() around swap cache freeing. This makes some races.
- We focus on swap entry itself.

Uncharge(SwapCache)(2)

We change swap entry's reference counting. Swap keeps refcnt of swap entry in swap_map[].

[Before]

swap_map[entry] = # of references + a ref from SwapCache

[After]

swap_map[entry] = # of references | SWAP_HAS_CACHE flag

Now, we can know swap entry is really used or not.

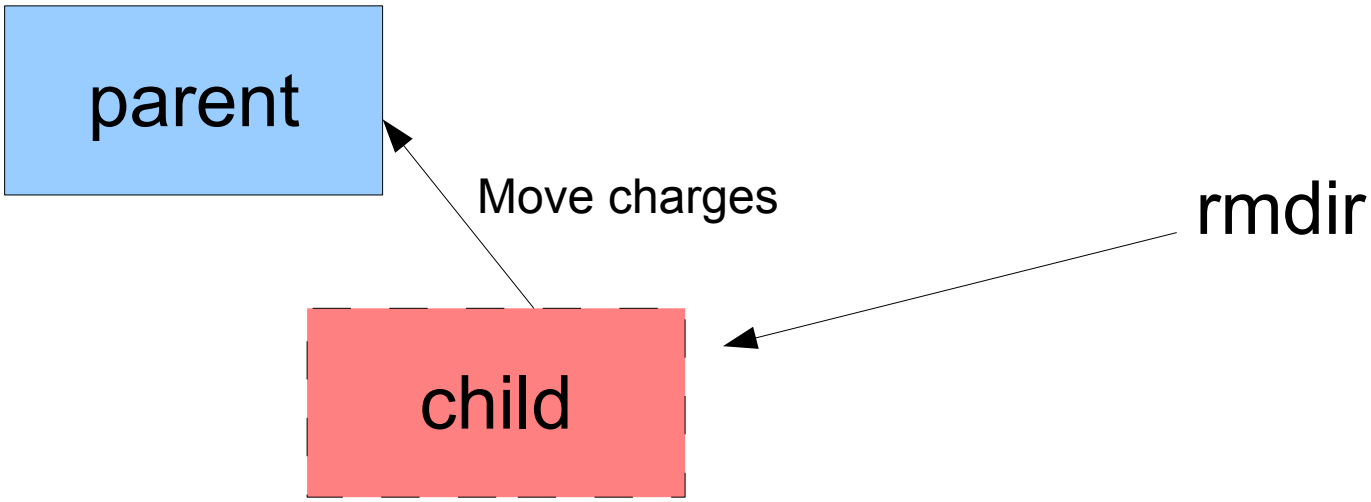
Uncharge(SwapCache)(3)

- do following at swap refcnt decrement.
  - No SwapCache, no swap refs.
    - Memory= no change, Mem+Swap -= PAGE_SIZE
  - SwapCache is not mapped, no swap-refs
    - Memory -= PAGE_SIZE, Mem+Swap -= PAGE_SIZE
  - SwapCache is not mapped, but swap-refs.
    - Memory -= PAGE_SIZE, Mem+Swap= no change.

No-swap-refs but has swap-cache, it's not mapped... page will be culled when the kernel searches available swap entries, because it's obvious that swapcache is of-no-use. Owner of swap entry(cgroup) is recored..but I don't explain it in this slide.

What happens at rmdir

- At rmdir(), file caches can exist as charge even if no tasks.
- All charges under a child will be moved to its parent at rmdir. If it seems the parent is full, pages are culled for fitting parent's limit.

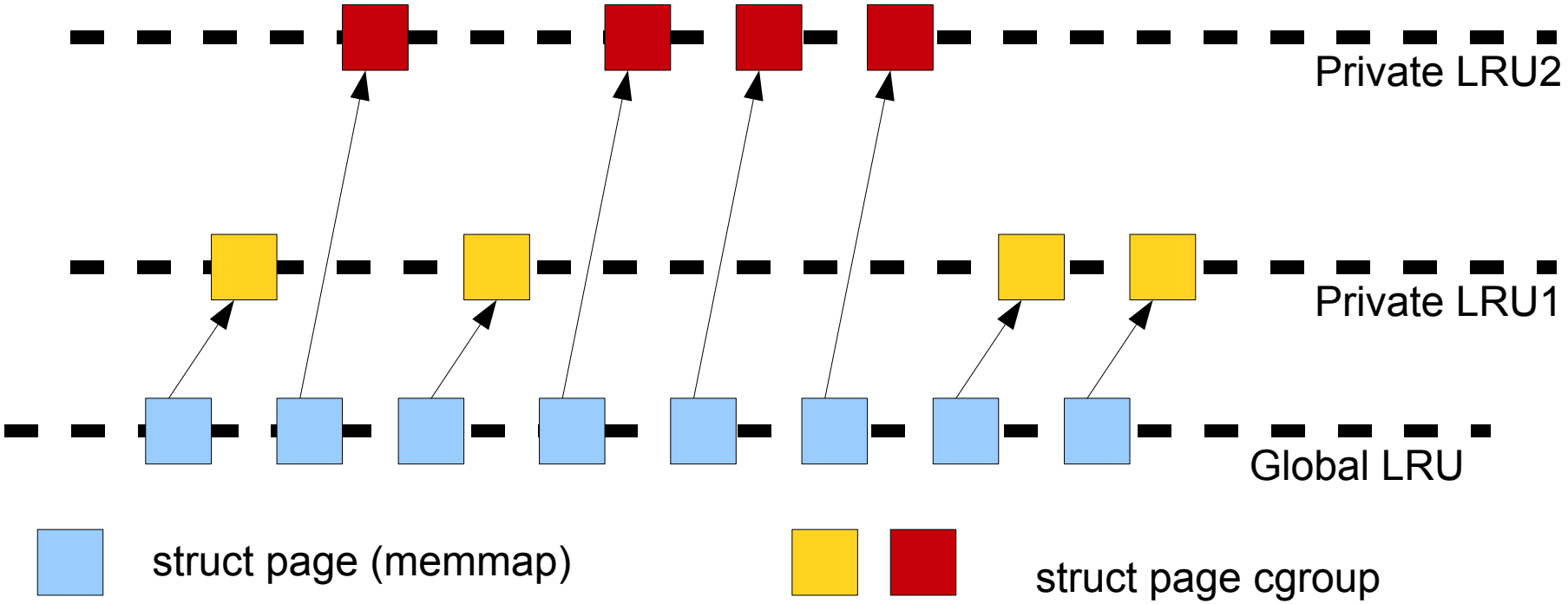


Contents

- Background
- Memory Resource Controller
  - Basics
  - Charege/Uncharge
  - LRU
  - Performance
  - TODO List

Per-memcg LRU

Memory controller has its own LRU.



These LRUs are maintained per-zone

LRU handling

- Memcg's LRU is handled under zone->lru_lock as global LRU's one.
- Memcg's LRU tracks behavior of global LRU.
- Memcg's LRU add hooks to global LRU's code and tries to do similar things.

Memcg's LRU list works asynchronously with memcg's charge/uncharge.

Seems a bit complicated but reduces maintenance cost.

Difference between memcg's reclaim and global's

- Global VM's memory reclaim logic is triggered at memory shortage in a zone.
  - It's important where we reclaim memory from and the kernel may have to reclaim continuous pages.
  - Kswapd works.
  - Slabs are dropped.
- Memcg's memory reclaim is triggered at memory usage hits its limit.
  - Just reducing memory is important. No care about placement of pages.
  - No kswapd help (now)
  - No slab drop.

Out-Of-Memory(OOM)

Memcg may hit OOM if usage cannot be reduced under limit.

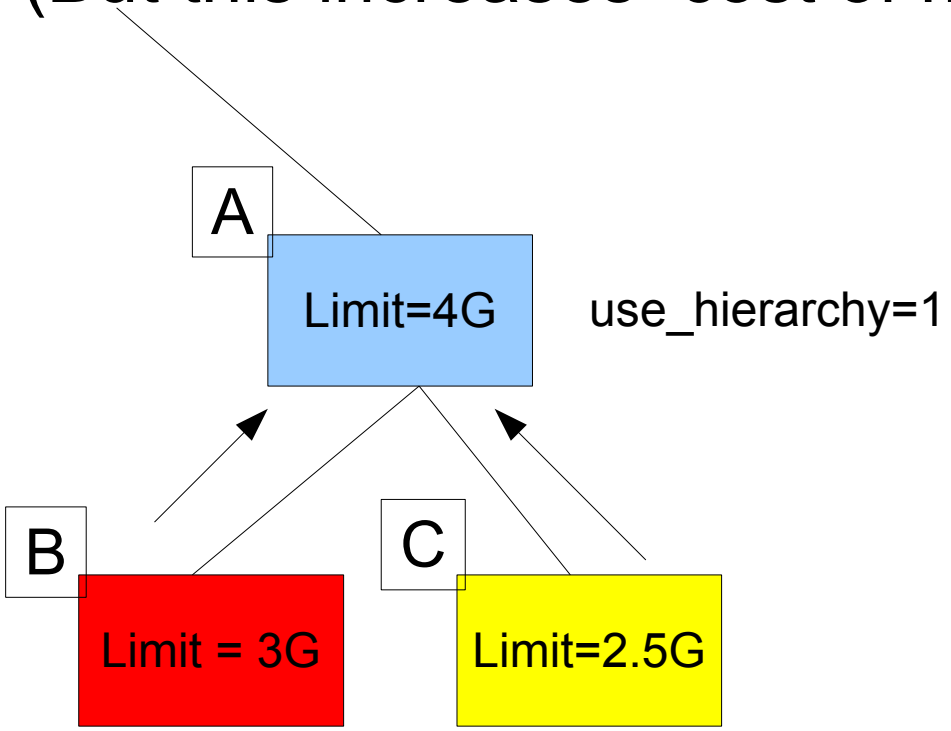
- At OOM in a cgroup, a process in it will be killed by oom-killer.

If global LRU hits OOM, usual OOM killer is invoked.

Special OOM handler has been discussed but ..... not implemented yet.

Hierarchical accounting

Cgroup has filesystem hierarchy.
In memcg, if use_hierarchy=1, hierarchical accounting is used. Charges are accounted to all ancestors.
(But this increases cost of memcg.)



usage(A) =

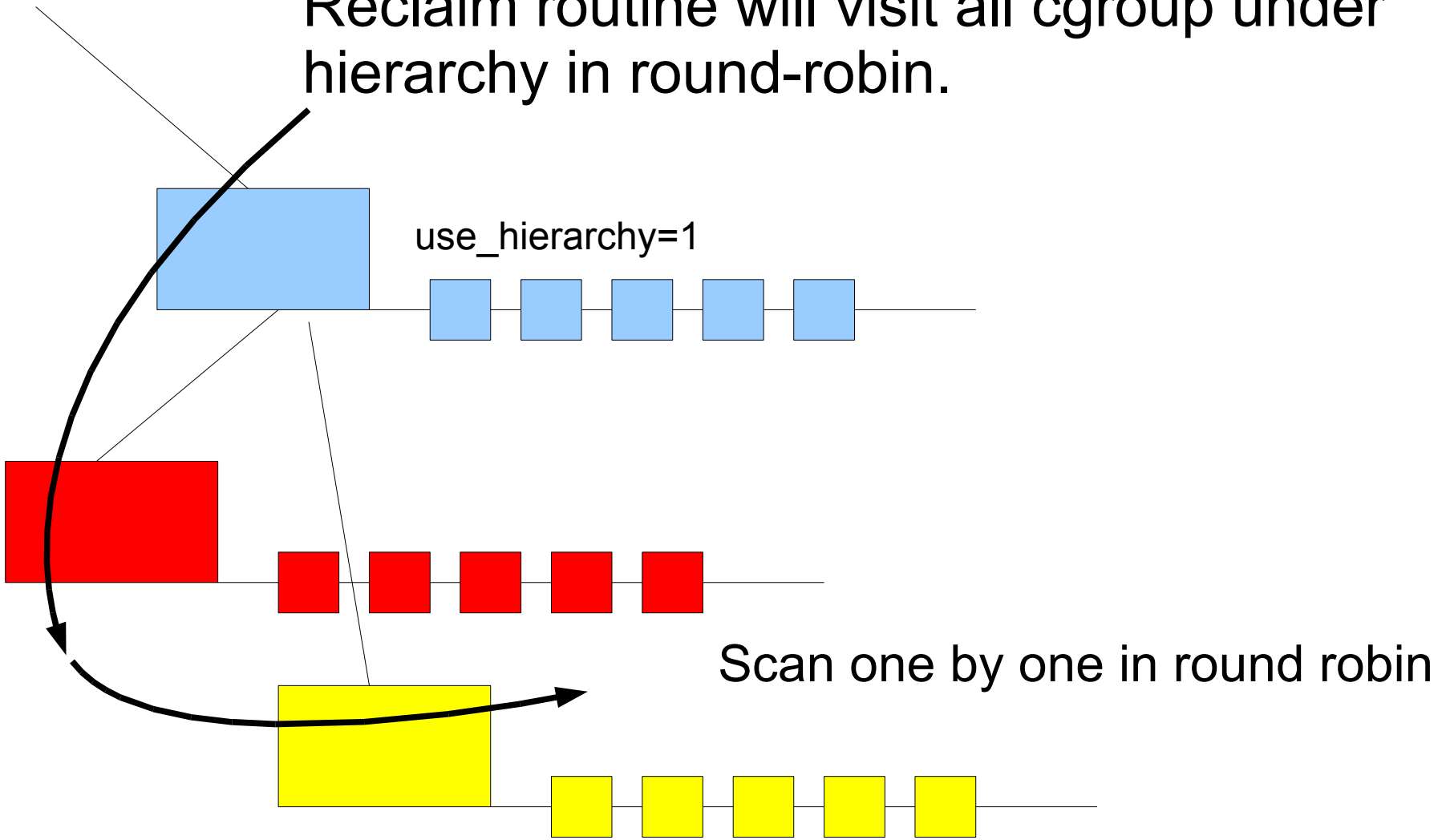
+ usage under A

+ usage(B)

+ usage(C)

Hierarchical accounting and LRU

LRU is maintained under each cgroup.
Reclaim routine will visit all cgroup under hierarchy in round-robin.



SoftLimit

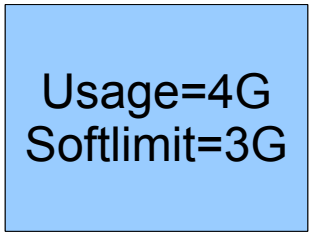
- SoftLimit is a feature for showing a hint to kswapd.
- If a memcg exceeds its softlimit, it will be notified to kswapd as candidates for victim.
- Victim group information are sorted by usage in excess of softlimit.

Very new feature in 2.6.31-rc series.

SoftLimit(Cont.)

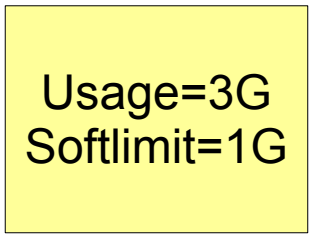
Example) Assume 3 groups of A, B, C.
Excess order is B->A->C

A



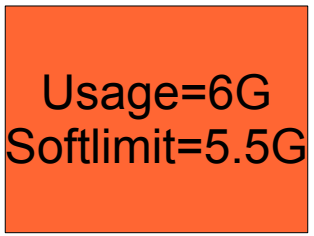
Excess = 4G-3G = 1G

B



Excess = 3G-1G = 2G

C



Excess = 6G-5.5G = 0.5G

With this hint, kswapd will cull memory from memcgs in order of B->A->C.

But kswapd has to check location of memory, this order is just a hint.

Contents

- Background
- Cgroup and subsystems
- Memory Resource Controller
  - Basics
  - Charge/Uncharge
  - LRUs
  - Performance
  - TODO Lists

2 aspects of performance impact of memcg.

a) Costs to call additional code path, handling page_cgroup.

b) Costs to modify system-wide shared counter for accounting usage.

b) has been a big problem in these months.

Benchmark

- Kernel make on tmpfs
  - make -j 8 + defconfig.
  - No I/O layer influence. (just for seeing cgroup costs)
- Parallel page fault microbench.
  - Do mmap/fault/unmap 1Mbytes of range repeatedly.
  - Run in parallel on 8cpus.

Environment

- x86-64 (3GHz/2socket/8core) box. SMP
- kernel version: 2.6.32-rc4
- Used following configuration
  a) 2.6.32-rc4 disable memcg by config
  b) 2.6.32-rc4 + under root cgroup
  c) 2.6.32-rc4 + under a child cgroup
  d) 2.6.32-rc4 + under a 2nd-level child cgroup
- mem+swap cgroup is enabled.

2.6.32-rc series includes performance fix for root cgroup.
After this fix, you can't set limit to root cgroup.

Result(-rc4/make)

Results of “time” on make -j 8 kernel/tmpfs

Config	sys(sec)	user(sec)	real(sec)
No config	36.7	79.5	20.7
Root cgroup	39.3	79.3	21.2
1st level child	44.1	79.2	21.7
2nd level child	47.5	79.4	22.4

No I/O influences. “sys” indicates the cost of memory cgroup.

Result(-rc4/pagefaults)

Measured sum of page faults on 8processes/8cpus.
Each process does mmap/fault/munmap 1Mbytes of anonymous pages.

config	Throughput (M/sec)	cache-miss/fault (smaller is better)
No config	0.143	8.02
Root cgroup	0.123	9.87
1st level child	0.039	37.8
2nd level child	0.038	42.1

Environment(-mm)

- Trying to reduce overheads in child cgroup.
- x86-64 (3GHz/2socket/8core) box. SMP
- kernel version: 2.6.32-rc3-mm
- Used following configuration
  a) 2.6.32-rc3-mm disable memcg by config
  b) 2.6.32-rc3-mm + under root cgroup
  c) 2.6.32-rc3-mm + under a child cgroup
  d) 2.6.32-rc3-mm + under a 2nd-level child cgroup
- mem+swap cgroup is enabled.

Result(-mm/make)

Results of “time” on make -j 8 kernel/tmpfs

Config	sys(sec)	user(sec)	real(sec)
No config	39.1	79.5	20.9
Root cgroup	40.0	79.3	21.2
1st level child	40.1	79.2	21.2
2nd level child	40.3	79.3	21.2

No I/O influences. “sys” indicates the cost of cgroup.

Result(-mm/pagefaults)

Measured sum of page faults on 8processes/8cpus.
Each process does mmap/fault/munmap 1Mbytes of anonymous pages.

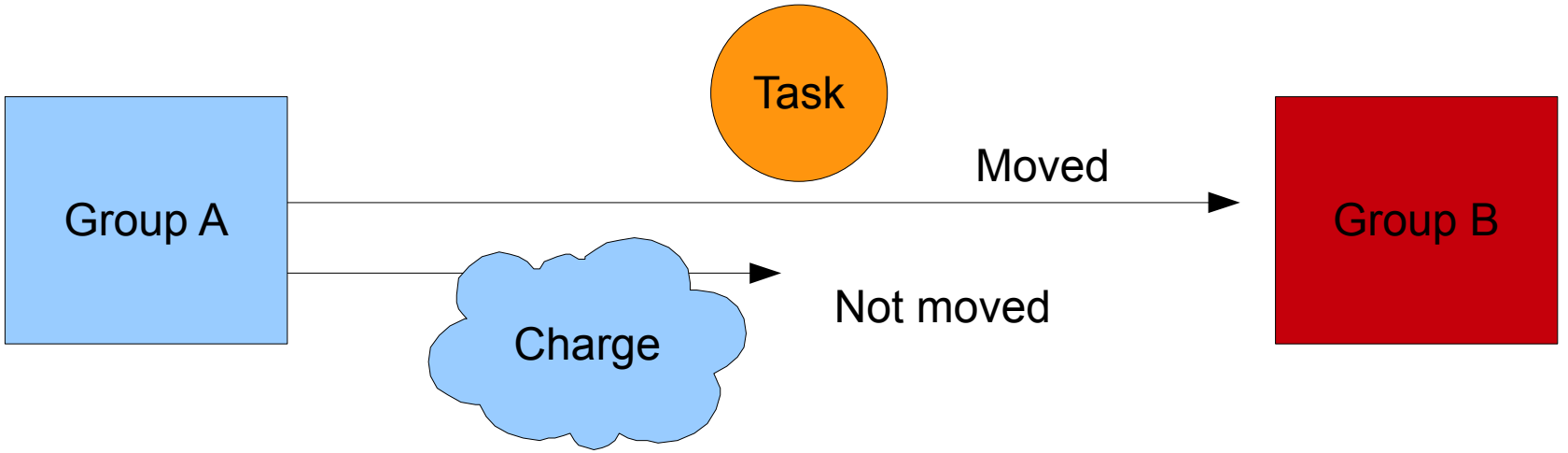
config	Throughput (M/sec)	cachemiss/fault (smaller is better)
No config	0.142	7.98
Root cgroup	0.142	8.54
1st level child	0.133(*)	8.90
2nd level child	0.138	10.15

(*)# of page faults can be affected by cpu utilization.
Cache-miss / fault shows the cost per page fault.

Contents

- Background
- Memory Resource Controller
  - Basics
  - Charge/Uncharge
  - LRUs
  - Performance
  - TODO Lists

Moving task



Users can move tasks between cgroup.
Now, even if tasks are moved, charges obtained by them will not move.

Feature(s) for moving “charges” is now under development.

Dirty page accounting

- Now, the number of dirty pages per cgroup is not accounted.
- Then, we cannot kick pdflush(flusher threads) to do write back fast.
- So, when “write a big file” hits the limit, performance is not good.

Event notifier

- An interface for waiting until the usage exceeds a specified value.

  like.....

  # echo 200M > memory.notify_excess

  # wait_for_ready < memory.notify_excess
- OOM-notifier.
  - Stop OOM-Killer under memcg ???

Others...

- Remove EXPERIMENTAL.
- Scalability / Clean up
- User Land Tools!
- Disk I/O controller must be necessary.....
- Large pages, mlock amounts, guarantee ?

- Consider a counter which can scale more....

Acknowledgements

Special thanks to

Balbir Singh (IBM)

Nishimura Daisuke (NEC)

KOSAKI Motohiro (Fujitsu)

and maintainers

Paul Menage (Google)

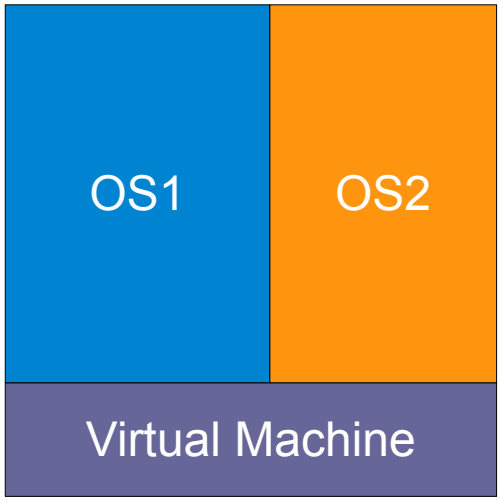
Andrew Morton (Google)

All my works are supported by Fujitsu Ltd.

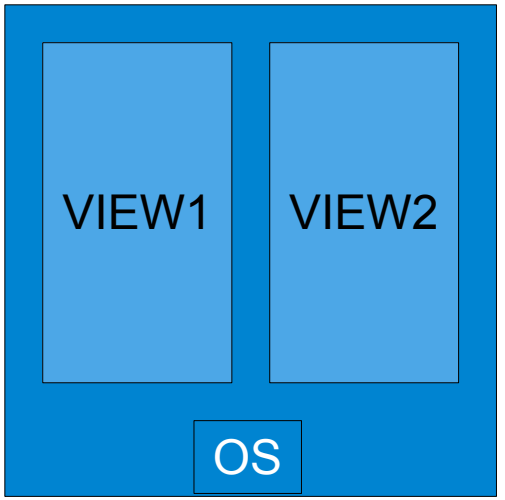
Back Up

3Levels of resource control

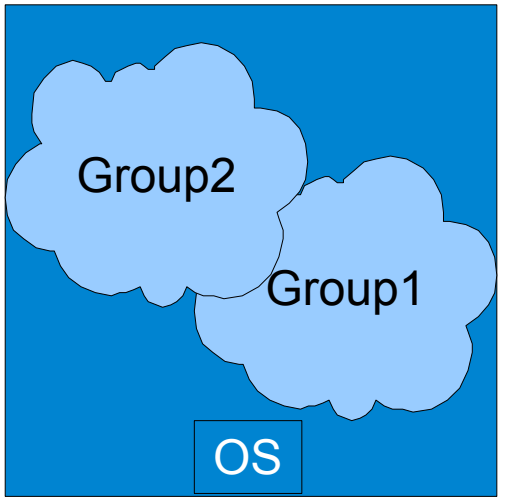
Isolation by
Virtual Machine



Isolation by OS(Virtual OS)
(Container/Jail)



Flexible Resource Control



	Virtual Machine	Container	RC
Performance	Not good	Very good	Good
Isolation/Security	Very good	Good	Not good
Runtime Flexibility	Not good	Good	Very good
Maintenance	Not good	Good	Good