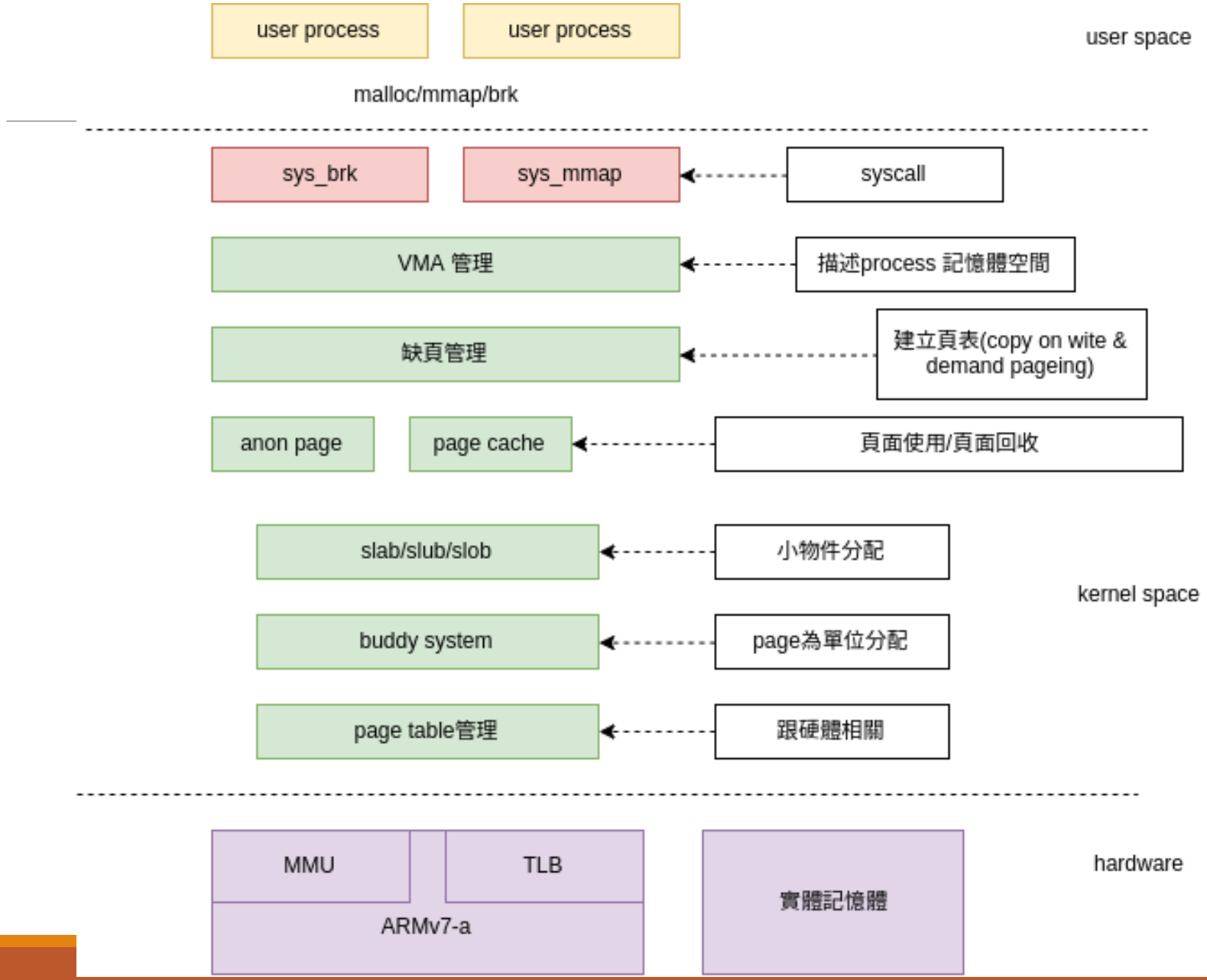
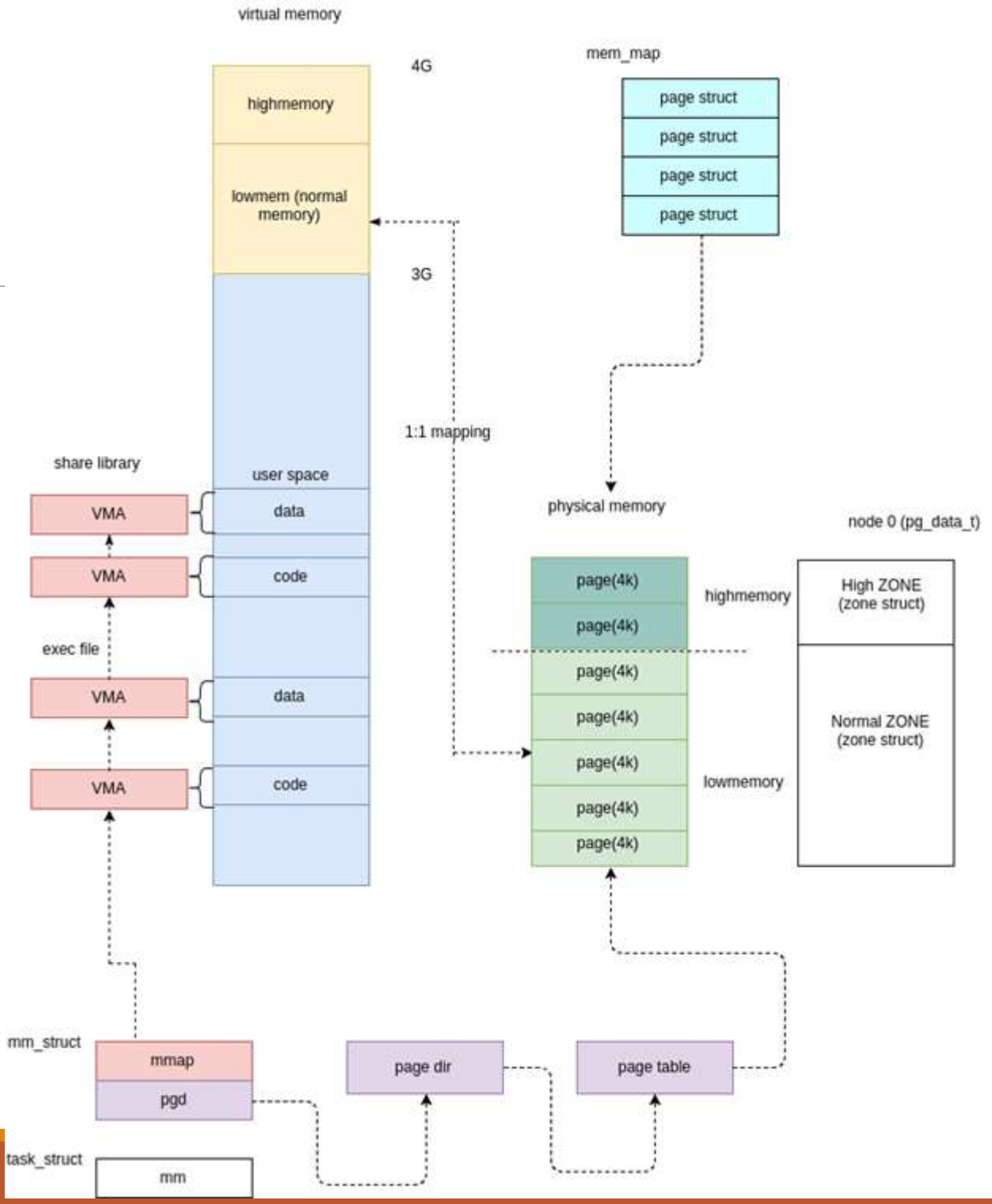
ch12 Memory Management

wayling

2017/12/19

Process 視角


user process
user process
user space
malloc/mmap/brk
sys_brk
sys_mmap
syscall
VMA 管理
描述process 記憶體空間
缺頁管理
建立頁表(copy on wite & demand pageing)
anon page
page cache
頁面使用/頁面回收
slab/slub/slob
小物件分配
kernel space
buddy system
page為單位分配
page table管理
跟硬體相關
MMU
TLB
ARMv7-a
實體記憶體
hardware


virtual memory
4G
highmemory
lowmem (normal memory)
3G
user space
share library
VMA
data
VMA
code
exec file
VMA
data
VMA
code
1:1 mapping
mem_map
page struct
page struct
page struct
page struct
physical memory
node 0 (pg_data_t)
page(4k)
page(4k)
page(4k)
page(4k)
page(4k)
page(4k)
page(4k)
highmemory
lowmemory
High ZONE (zone struct)
Normal ZONE (zone struct)
mm_struct
mmap
pgd
page dir
page table
task_struct
mm

地址轉換 ARMv7-a

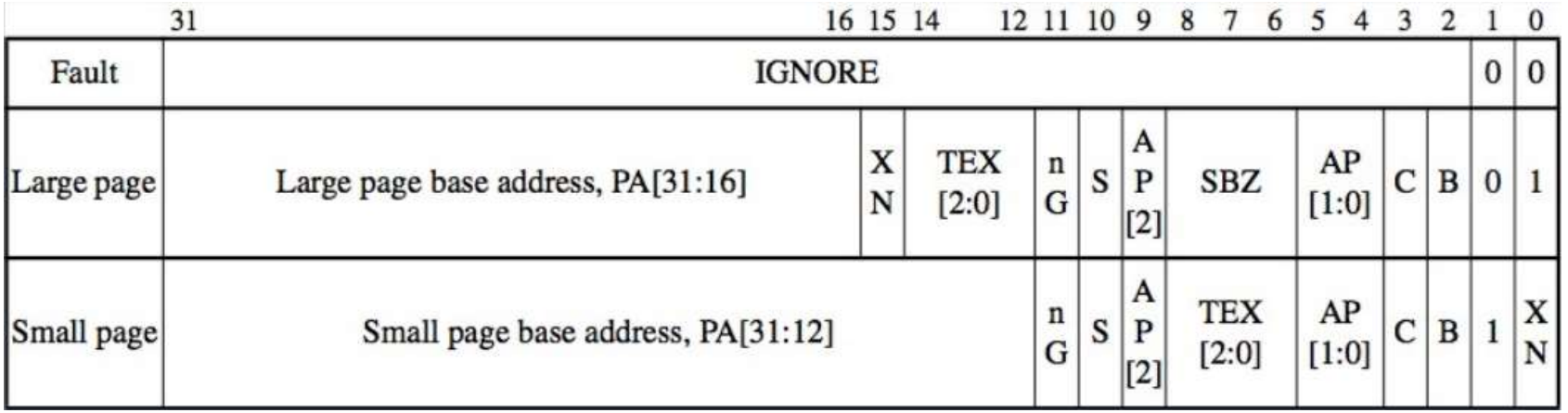
- 這邊只探討ARMv7a沒開LPAE
- 目前linux ARMv7-a是kernel(TTBR1) / user space(TTBR0)的page table獨立所以需要分別探討
- ARMv7-a Kernel space的page table是不會隨著context switch而切換的(跟x86不同,x64沒研究…),也就是所有process看到一樣的kernel space
- Linux ARMv7-a software跟hardware的地址轉換"過程"是不一樣的,原因是hardware table(h/w pt)所提供欄位無法滿足linux kernel的設計所以需要software table (linux pt)

ARMv7-a level 1 table

- Kernel space使用 Section(10)
- User space 是用Page table(01)
- 引發page fault(00)

	31	24	23	20	19	18	17	16	15	14	12	11	10	9	8	5	4	3	2	1	0
Fault	IGNORE																			0	0
Page table	Page table base address, bits [31:10]													IMP	Domain		SBZ	NS	SBZ	0	1
Section	Section base address, PA[31:20]				NS	0	nG	S	AP[2]	TEX [2:0]		AP [1:0]		IMP	Domain		XN	C	B	1	0
Supersection	Supersection base address PA[31:24]		Extended base address PA[35:32]		NS	1	nG	S	AP[2]	TEX [2:0]		AP [1:0]		IMP	Extended base address PA[39:36]		XN	C	B	1	0
Reserved	Reserved																			1	1

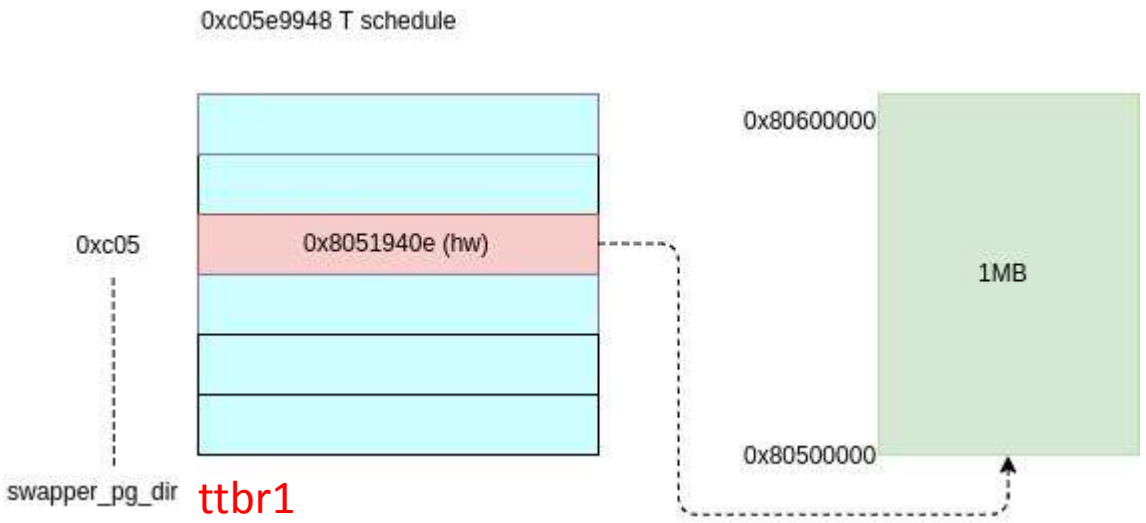
ARMv7-a level 2 table



	31 … 16	15	14 … 12	11	10	9	8 … 7	6 … 5	4 … 3	2	1	0
Fault	IGNORE										0	0
Large page	Large page base address, PA[31:16]	XN	TEX [2:0]	nG	S	AP [2]	SBZ	AP [1:0]	C	B	0	1
Small page	Small page base address, PA[31:12]			nG	S	AP [2]	TEX [2:0]	AP [1:0]	C	B	1	XN

kernel地址轉換 (hardware 視角)

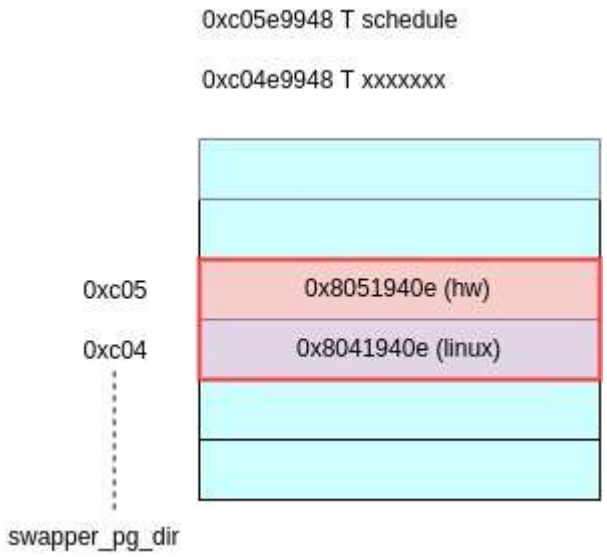
- section mapping所以每個entry對應1MB 地址空間



  - 0xc05e9948 => 0x805e9948

kernel地址轉換 (software(kernel) 視角)

- 對kernel來說看到的是2MBmapping,只需要table的欄位而已
- Kernel space 的page table地址可以直接參考swapper_pg_dir



- typedef u32 pmdval_t;
- typedef pmdval_t pgd_t[2];
- pgd_t *pgd;
- pgd = pgd_offset_k(0xc05e9948);
- pgd_val(*pgd);

- 0xc05e9948 => 0x8041940e
- 0xc04e9948 => 0x8041940e

user地址轉換 (hardware 視角)

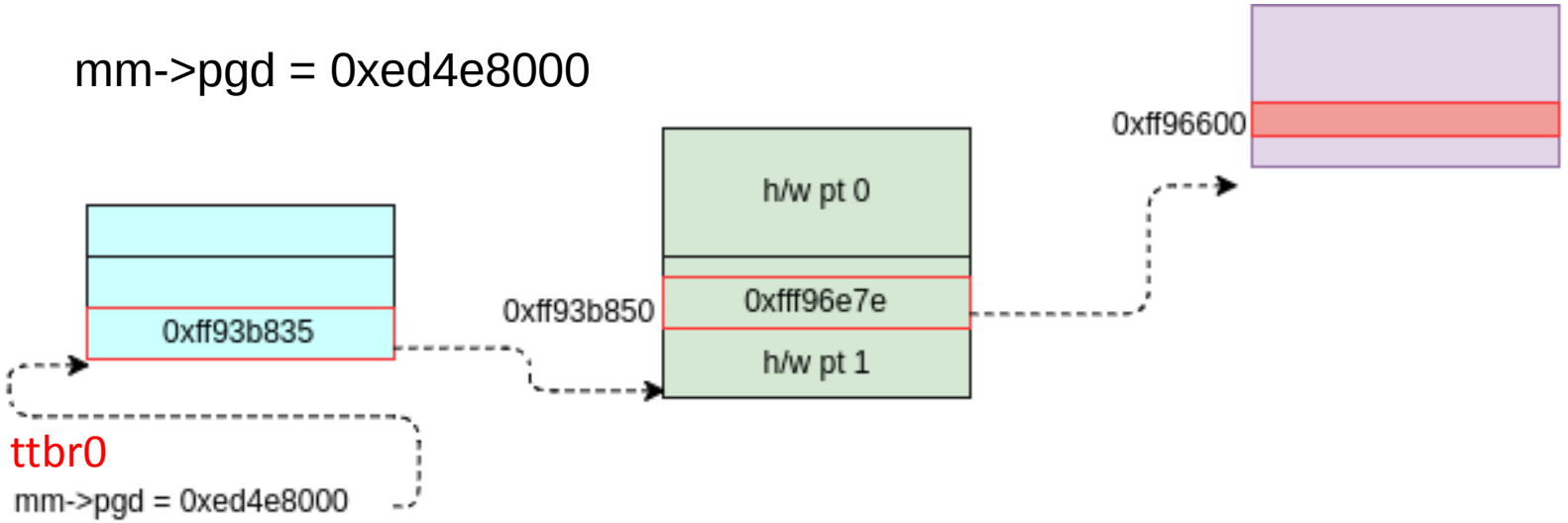
- process 的page table地址可以參考mm->pgd

Example : busybox(init) code addr : 0x00014600

0b0000 0000 0000 0001 0100 0110 0000 0000

ARMv7a的hw地址轉換: 12/8/12
0x0 / 0x14 / 0x600

mm->pgd = 0xed4e8000



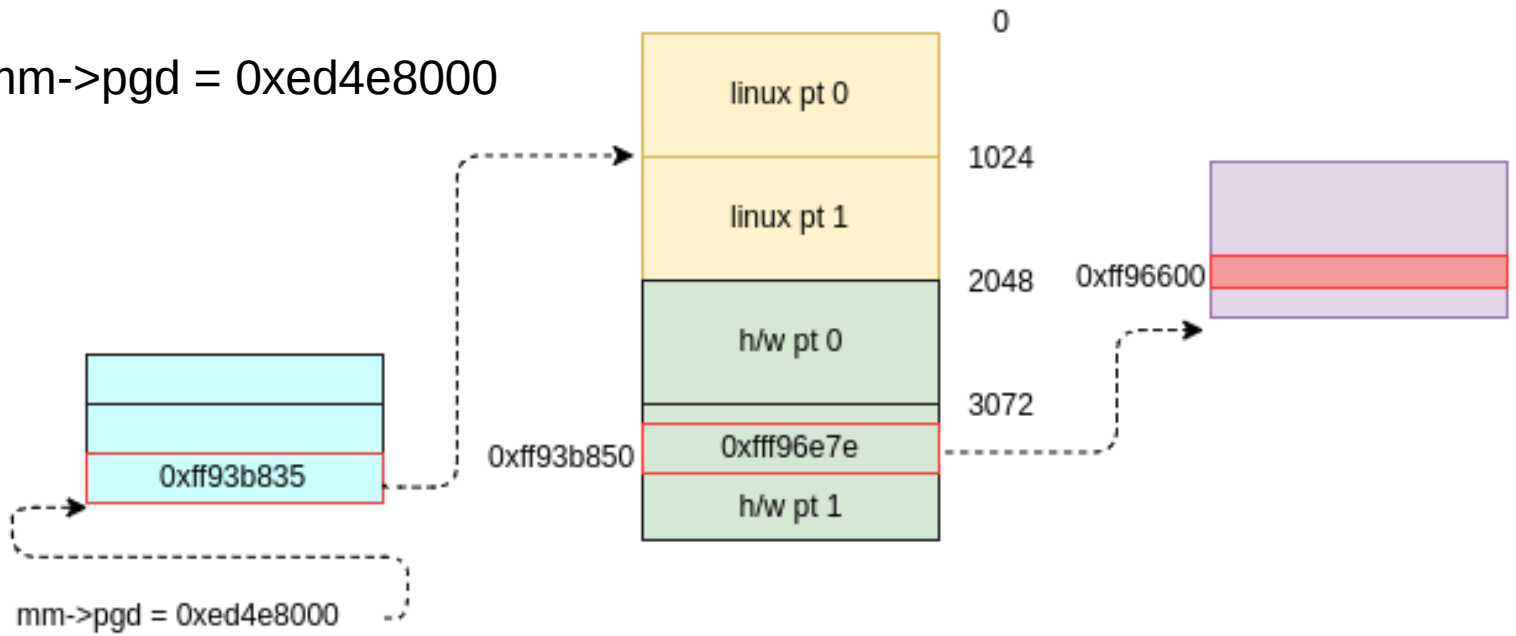
user地址轉換 (software(kernel) 視角)

busybox(init) code addr : 0x00014600

0b0000 0000 0000 0001 0100 0110 0000 0000

ARMv7a的sw地址轉換:11/9/12
0x0 / 0x14 / 0x600

mm->pgd = 0xed4e8000



Linux 地址轉換參考

```
/*
* This is useful to dump out the page tables associated with
* 'addr' in mm 'mm'.
*/
void show_pte(struct mm_struct *mm, unsigned long addr);
```

- virt_to_phys(virt_addr);
- phys_to_virt(phys_addr);

- arch/arm/mm/fault.c
- 可以參考kernel及user space的address轉換過程

Linux memory map

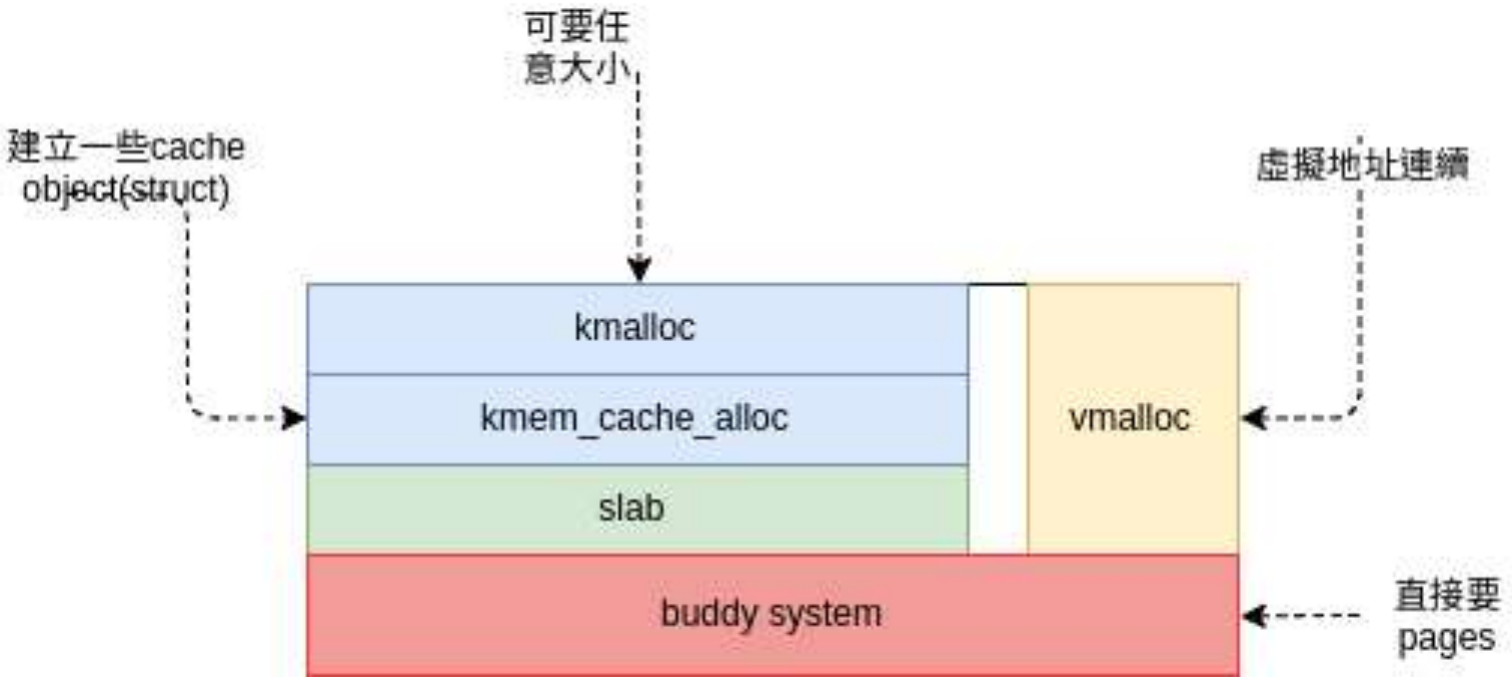
- 如果想要dump kernel space的page table可以打開”CONFIG_ARM_PTDUMP”
  - mount -t debugfs none /sys/kernel/debug/
  - cat /sys/kernel/debug/kernel_page_tables

```
---[ Modules ]---
0xbfe01000-0xbfe3c000         236K     RW NX SHD MEM/CACHED/WBWA
---[ Kernel Mapping ]---
0xc0000000-0xc0100000           1M     RW NX SHD
0xc0100000-0xc0600000           5M     ro x  SHD
0xc0600000-0xc0800000           2M     ro NX SHD
0xc0800000-0xf0000000         760M     RW NX SHD
---[ vmalloc() Area ]---
0xf0800000-0xf0801000           4K     RW NX SHD DEV/SHARED
0xf0802000-0xf0804000           8K     RW NX SHD DEV/SHARED
0xf0805000-0xf0806000           4K     RW NX SHD DEV/SHARED
0xf0807000-0xf0808000           4K     RW NX SHD DEV/SHARED
0xf0809000-0xf080a000           4K     RW NX SHD DEV/SHARED
0xf0811000-0xf0812000           4K     RW NX SHD DEV/SHARED
0xf0813000-0xf0814000           4K     RW NX SHD DEV/SHARED
0xf0815000-0xf0816000           4K     RW NX SHD DEV/SHARED
0xf0817000-0xf0857000         256K     RW NX SHD MEM/BUFFERABLE/WC
```

在開機過程也會有kernel space memory map輸出

```
Memory: 2056788K/2097148K available (5120K kernel code, 172K rwdata, 1220K rodata, 1024K init, 4304K bss, 40360K reserved, 0K cma-res
erved, 1310716K highmem)
Virtual kernel memory layout:
    vector  : 0xffff0000 - 0xffff1000   (   4 kB)
    fixmap  : 0xffc00000 - 0xfff00000   (3072 kB)
    vmalloc : 0xf0800000 - 0xff800000   ( 240 MB)
    lowmem  : 0xc0000000 - 0xf0000000   ( 768 MB)
    pkmap   : 0xbfe00000 - 0xc0000000   (   2 MB)
    modules : 0xbf000000 - 0xbfe00000   (  14 MB)
      .text : 0xc0008000 - 0xc0600000   (6112 kB)
      .init : 0xc0800000 - 0xc0900000   (1024 kB)
      .data : 0xc0900000 - 0xc092b2c0   ( 173 kB)
       .bss : 0xc092d000 - 0xc0d613f4   (4305 kB)
```

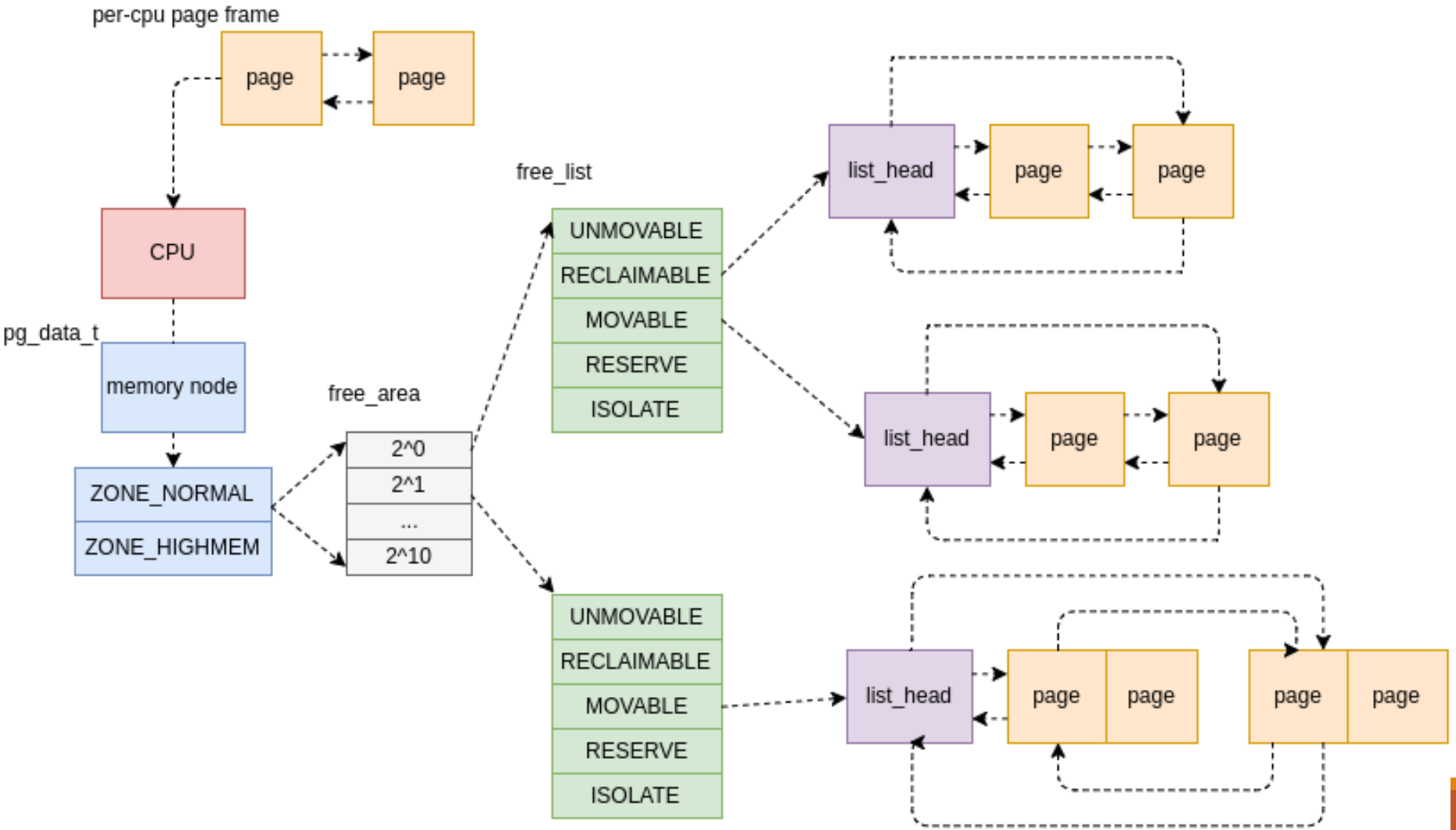
常見分配記憶體的方式


可要任
意大小
建立一些cache
object(struct)
虛擬地址連續
kmalloc
kmem_cache_alloc
slab
vmalloc
buddy system
直接要
pages

zone

- 實體記憶體的使用有時候需要屈就於外部裝置因此kernel對實體記憶體作區域(zone)的劃分
- 典型x86
  - ZONE_DMA
  - ZONE_NORMAL
  - ZONE_HIGHMEM
- 典型ARMv7a
  - ZONE_NORMAL 768 MB
  - ZONE_HIGHMEM > 768 MB
- 再傳統32bit (1GB/3GB)的劃分下我們會面臨一個問題,1GB的kernel space無法1:1的去映射超過1GB的實體記憶體因此有了ZONE_HIGHMEM的出現.
- 在64bit機器下ZONE_HIGHMEM是不需要的
- zone struct會定義一些跟記憶體回收相關的watermark (minimum,low,high)
- 可以參考 cat /proc/zoneinfo

zone&buddy system



binary buddy system allocation(1)

- 記憶體分配
  - 尋找一個合適大小的記憶體(大於 requested memory, 同時也就是分配一個滿足要求的最小記憶體2^n)
  - 如果找到了直接分配
  - 如果沒有找到
    - 1.拆分一個比 requested memory 更大的記憶體塊(2^n+1, 分成兩半)
    - 2.如果拆分出來的一半滿足requested memory, 並且不能再分了, 已經是最小的了,就分配該塊.
    - 3.重複1, 尋找合適大小的內存塊.

binary buddy system allocation(2)

- 記憶體釋放
  - 1.釋放2^n記憶體塊
  - 2.查看記憶體塊的夥伴也就是分配之後的另一半2^n塊是否也free了
  - 3.如果是，則會回到2並且重複執行直到所有記憶體被釋放或者有一個夥伴沒有被free掉, 無法合併.

buddy system

```
# cat /proc/buddyinfo
Node 0, zone   Normal    130    103     34      9      5      6      6      4      4      2    176
Node 0, zone  HighMem      2     10      7      8      4      1      1      2      1      1    318
```

```
# cat /proc/pagetypeinfo
Page block order: 10
Pages per block:  1024

Free pages count per migrate type at order       0      1      2      3      4      5      6      7      8      9     10
Node    0, zone   Normal, type    Unmovable    150     44      5      1      1      1      1      1      1      0      0
Node    0, zone   Normal, type      Movable      2      1      2      4      5      3      4      3      2      2    176
Node    0, zone   Normal, type  Reclaimable      1      1      0      2      0      1      1      0      1      0      0
Node    0, zone   Normal, type   HighAtomic      0      0      0      0      0      0      0      0      0      0      0
Node    0, zone   Normal, type          CMA      0      0      0      0      0      0      0      0      0      0      0
Node    0, zone   Normal, type      Isolate      0      0      0      0      0      0      0      0      0      0      0
Node    0, zone  HighMem, type    Unmovable     10      3      2      1      0      0      0      1      0      1      0
```

page

- struct page{
  - unsigned long flags;
  - unsigned counters;
  - atomic_t _mapcount;
  - ….
- };
- 此結構用於描述實體頁面,該結構對所描述的實體頁面可能是暫時性的,因為實體頁面可能被swap out.
- 因為每個實體頁面都需要一個struct page描述,所以對於此結構的大小需要很精準的控制,有些變數還是共用的(不同情況意義不同),避免浪費一點空間
- pagemap.txt這份文件有些資訊可以參考, kernel把page的資訊都有丟給user space去觀察(/proc/pid/pagemap , /proc/kpagecount , /proc/kpageflags)
  - https://www.kernel.org/doc/Documentation/vm/page

分配/釋放page的API

- 對於於linux 來說請求連續的記憶體空間的大小是有限制的,如有特殊應用需用其他方式保留
- 實體記憶體連續的分配API
  - struct page *alloc_pages(gfp_t gfp_mask,unsigned int order);
  - void *page_address(struct page *page);
  - unsigned long __get_free_pages(gfp_t gfp_mask,unsigned int order);
  - void free_pages(unsigned long addr,unsinged int order);
  -
  - void *kmalloc(size_t size,gfp_t flags);
  - void kfree(const void *ptr);

gfp_mask flag

- 參考 (include\linux\gfp.h)
- 動作修飾符
  - 如何分配所請求的記憶體,例如不能睡眠
- 分區修飾符
  - 從哪個zone中取得
- 類型
  - 完成特定類型的分配

gfp_mask flag

旗標	說明
GFP_ATOMIC	The allocation is high priority and must not sleep
GFP_NOWAIT	Like GFP_ATOMIC, except that the call will not fallback on emergency memory pools.
GFP_NOIO	This allocation can block, but must not initiate disk I/O.
GFP_NOFS	This allocation can block and can initiate disk I/O, if it must, but it will not initiate a filesystem operation.

gfp_mask flag

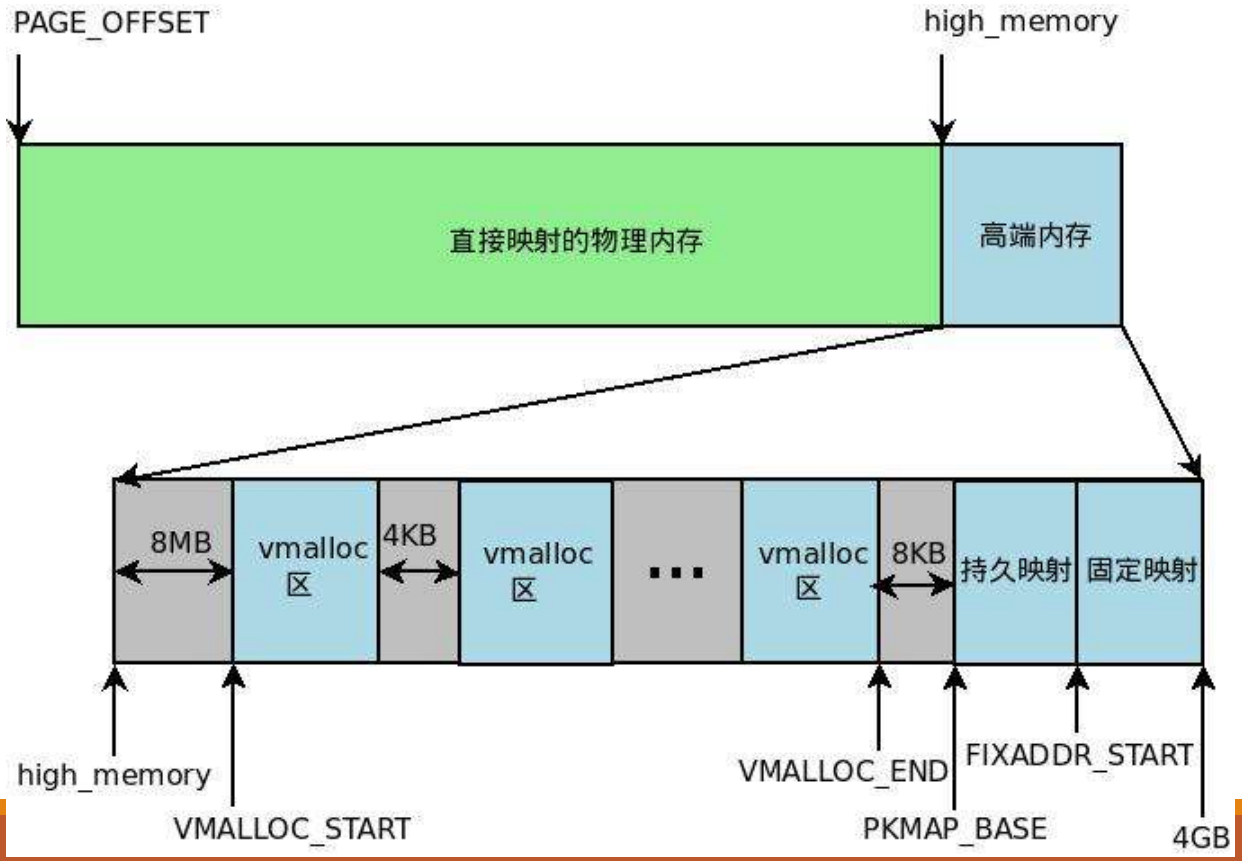
旗標	說明
GFP_KERNEL	This is a normal allocation and might block. This is the flag to use in process context code when it is safe to sleep
GFP_USER	This is a normal allocation and might block. This flag is used to allocate memory for user-space processes.
GFP_HIGHUSER	This is an allocation from ZONE_HIGHMEM and might block.
GFP_DMA	This is an allocation from ZONE_DMA.

非連續記憶體配置

- void *vmalloc(unsigned long size);
  - 分配的記憶體空間不保證實體上連續,虛擬是連續的
  - 通常硬體所需的記憶體空間都續要實體連續,所以不適合使用這個API
  - 把實體頁面做成虛擬的連續會對page table做操作有可能造成TLB效能損失
- void vfree(const void *addr);

vmalloc如何實作

- vmalloc使用類似user space的 VMA 來描述非連續區間,在這邊使用 struct vm_struct

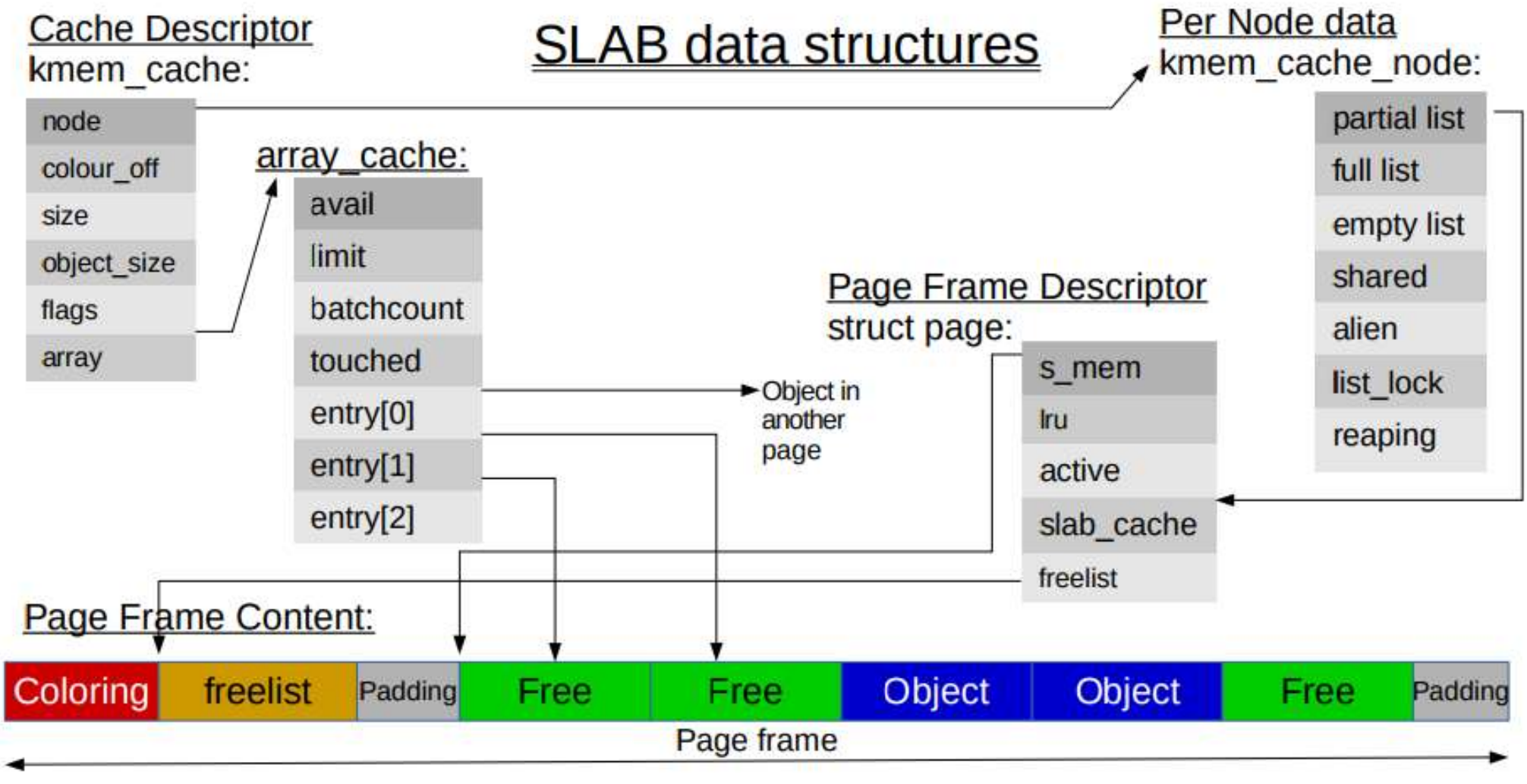


The SLAB allocator

- slab/slub/slob目前在linux kernel中是可選的
  - SLOB: As compact as possible
  - SLAB: As cache friendly as possible. Benchmark friendly
  - SLUB: Simple and instruction cost counts. Superior debugging. Defragmentation. Execution time friendly.
  - kmalloc() -> kmem_cache_create(), kmem_cache_alloc()
- 對於一個系統效能來說我們需要避免分配與釋放struct 物件頻繁發生
  - cache 常用資料結構,必要時再釋放
  - Per CPU cache
  - Colored object提升CPU cache命中

SLAB data structures

- 目前網路上就這張圖比較符合4.1x的 code.(維護者的投影片)



Slab 分配範例

- struct kmem_cache *task_struct_cachep;
- task_struct_cachep = kmem_cache_create(“task_struct”,
- sizeof(struct task_struct),
- ARCH_MIN_TASKALIGN,
- SLAB_PANIC | SLAB_NOTRACK,
- NULL);
- struct task_struct *tsk;
- tsk = kmem_cache_alloc(task_struct_cachep, GFP_KERNEL);

Statically Allocating on the Stack

- 每個process皆有一個kernel stack因為大小有限制所以儘量不要使用太多
- function call chain也會用到stack

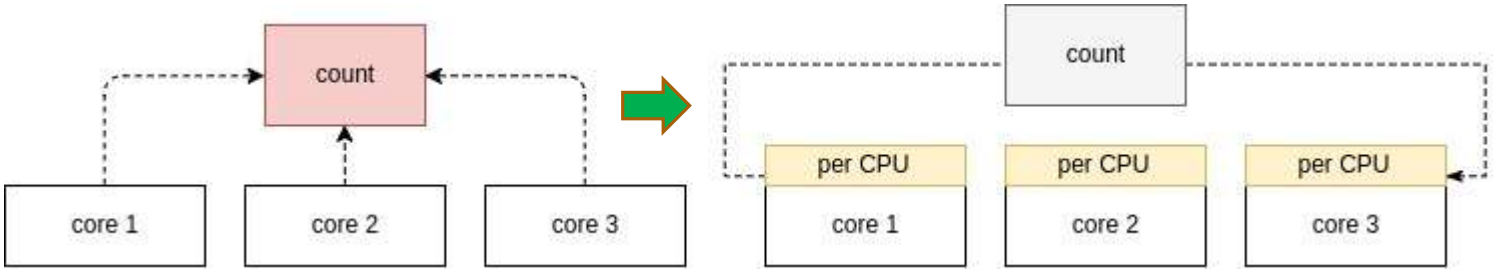
High Memory Mappings

- High Memory 區域只有實體記憶體可是沒有虛擬地址所以 kernel 要用的話需要mapping 到kernel space
- 永久
  - void *kmap(struct page *page)
  - void kunmap(struct page *page)
- 暫時
  - void *kmap_atomic(struct page *page, enum km_type type)
  - void kunmap_atomic(void *kvaddr, enum km_type type)
  -

Per-CPU Allocations

- 降低鎖的使用,再SMP系統上鎖的競爭很有可能是效能瓶頸
- 降低CPU cache miss,也減少cache 同步的時間

- Ex:如果要統計SMP系統上的packet收到數量



Future

- Linux mm 還有非常大的主題,之後可以分主題來深入研究
- 看有沒有人要入坑(加入奔跑吧讀書會)....Orz

Reference

- Linux Kernel Development 3rd
- 奔跑吧 Linux内核 - 张天飞 (作者)
- buddy-system-內核物理頁管理的實現
  - https://ggaaooppeenngg.github.io/zh-CN/2016/08/31/buddy-system-%E5%86%85%E6%A0%B8%E7%89%A9%E7%90%86%E9%A1%B5%E5%88%86%E9%85%8D%E7%9A%84%E5%AE%9E%E7%8E%B0/
- buddy-system-struct
  - http://guojing.me/linux-kernel-architecture/posts/buddy-system-struct/

Reference

- 内存管理（一）node & zone
  http://blog.chinaunix.net/uid-30282771-id-5171166.html
- 内存-内核空间
  https://jin-yang.github.io/post/kernel-memory-management-from-kernel-view.html
- Virtual Memory and Linux
  https://events.linuxfoundation.org/sites/events/files/slides/elc_2016_mem.pdf
- Slab allocators in the Linux Kernel: SLAB, SLOB, SLUB
  https://events.linuxfoundation.org/sites/events/files/slides/slaballocators.pdf