Struktura jądra UNIX

Wykład 12: Zarządzanie pamięcią jądra

FreeBSD: Elementy zarządzania pamięcią

1. Rozruch jądra i mapa pamięci.
2. Przydział ramek pamięci fizycznej.
3. Zarządzanie tablicą stron.
4. Zarządzanie wirtualną przestrzenią adresową.
5. Przydział stron pamięci wirtualnej.
6. Przydział bloków pamięci.
7. Obsługa błędów stron ***vm_fault.***
8. Zarządzanie odwzorowaniami pamięci ***vm_object.***
9. Programy stronicujące ***pagers.***
10. Zastępowanie stron i demon stronicowania ***pageout.***

Rodzaje stron pamięci

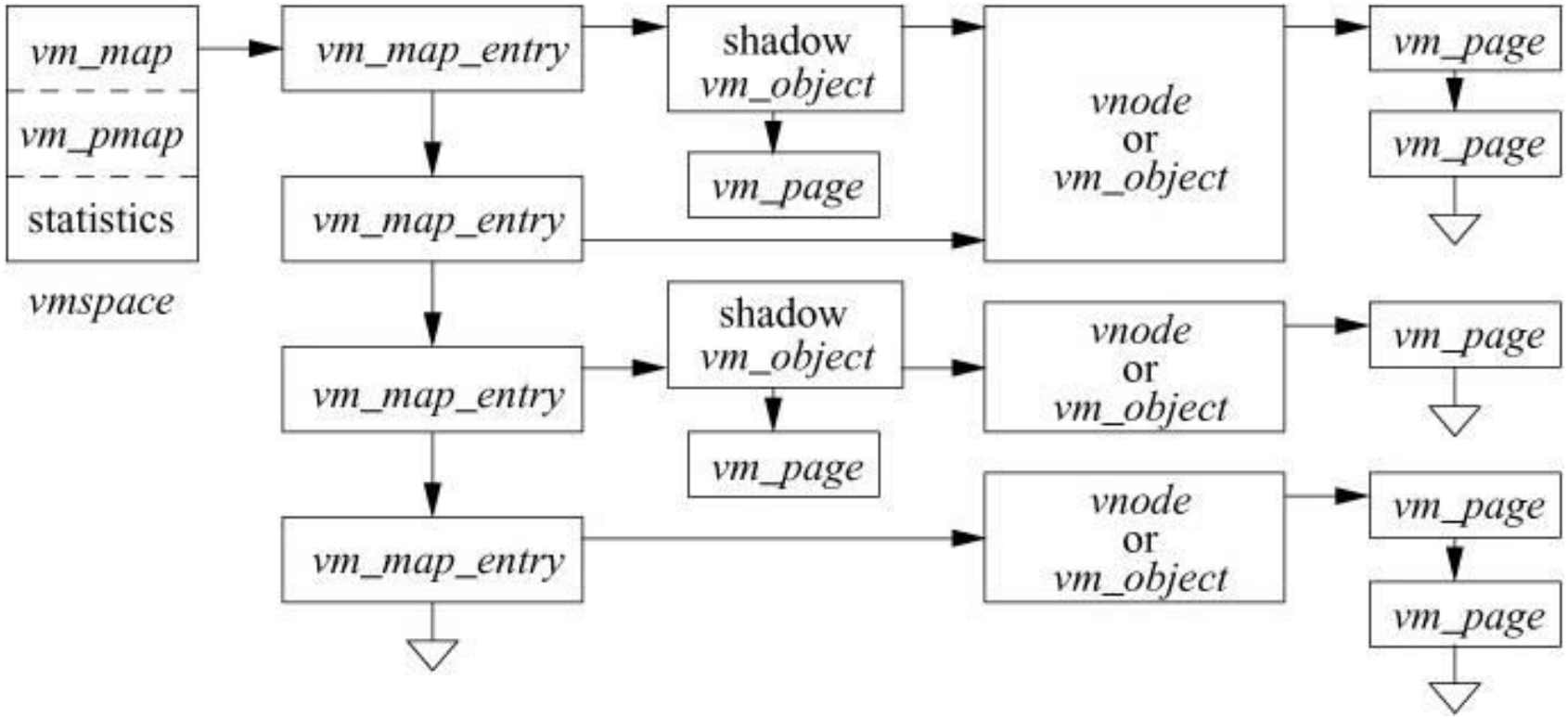
Pamięć zadrutowana (ang. *wired*) podlega przydziałowi w trakcie wywołania i nigdy nie generuje błędów strony. Dostęp nigdy nie spowoduje zmiany kontekstu.

Pamięć stronicowalna (ang. *pageable*) podlega wymianie i może zostać odczepiona przed demon ***`pageout`***. Przydzielana głównie procesom.

Pamięć fikcyjna (ang. *fictitious*) nie podlegającą stronicowaniu, a odnoszącą się do pamięci urządzeń.

Dla jądra z reguły przydzielamy pamięć zadrutowaną!

Reprezentacja przestrzeni adresowej



Składowe reprezentacji przestrzeni adresowej

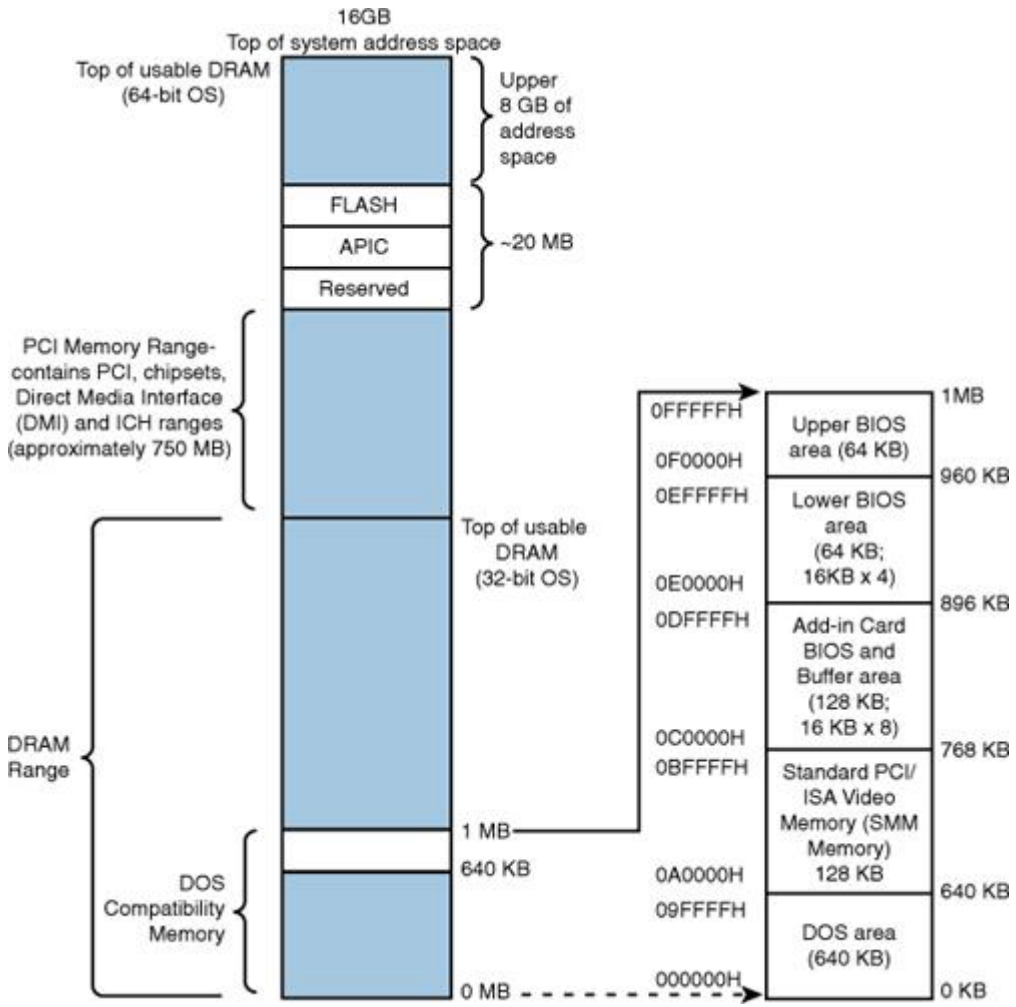
`vmspace`	zbiorcza struktura przestrzeni adresowej
`vm_map`	mapa pamięci
`vm_map_entry`	ciągły obszar adresów wirtualnych
`vm_object`	przechowuje pamięć fizyczną
`vm_page`	strona pamięci fizycznej
`pmap`	sprzętowa reprezentacja translacji adresów
`pagerops`	procedury stronicujące dla obiektu

Rozruch jądra i mapa pamięci

Brak abstrakcji pamięci

Zanim jądro systemu włączy pamięć wirtualną widzi gołą pamięć fizyczną (o ile nie jest uruchomione w maszynie wirtualnej)

```
cat /proc/iomem
```



Mapa pamięci fizycznej

Jądro musi wykryć pamięć operacyjną w maszynie (gdzie i ile?).
Dostajemy (BIOS, UEFI, FDT) opis maszyny do sparsowania,
po czym trzeba przydzielić tablice opisów wszystkich ramek stron.

Komunikaty jądra o detekcji pamięci: « `dmesg | grep e820` »

Część fizycznych adresów jest przeznaczona na podczepienie pamięci wykrytych urządzeń (karty graficzne, muzyczne, itd.)

Skoro mamy już spis całej dostępnej pamięci fizycznej, to możemy zacząć przydzielać i zwalniać ramki stron… oczywiście trzeba je jeszcze odwzorować w adresy wirtualne!

Start jądra systemu

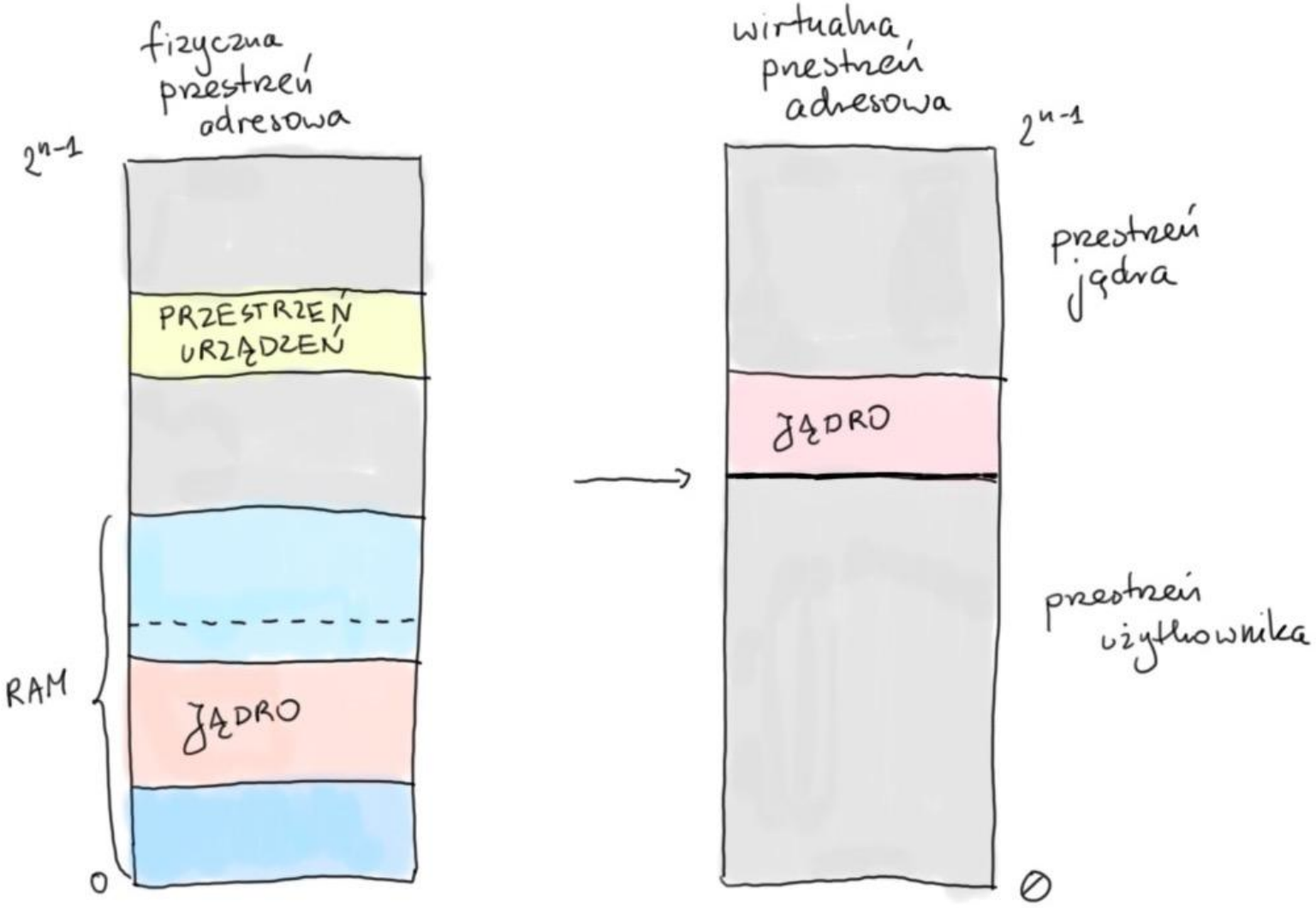
```
# mipsel-mimiker-elf-readelf -l sys/mimiker.elf
...
Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  LOAD           0x001000 0x80100000 0x00100000 0x013fc 0x013fc RWE 0x1000
  LOAD           0x003000 0xc0102000 0x00102000 0x37564 0x37564 R E 0x1000
  LOAD           0x03a564 0xc0139564 0x00139564 0x08ff8 0x08ff8 R   0x1000
  LOAD           0x044000 0xc0143000 0x00143000 0x02a7c 0x2309c RW  0x1000
```

Jądro ładowane przez program rozruchowy (ang. *boot loader*) do fizycznej przestrzeni adresowej, a działa w wirtualnej!

Procedura startowa ustala pierwotną tablicę stron, która przenosi liniowo adresy fizyczne do wysokich adresów wirtualnych, tak by niskich adresów używać do konstrukcji przestrzeni użytkownika.

UWAGA! Przeszczepiamy również stos.

Przestrzeń adresowa po rozruchu jądra

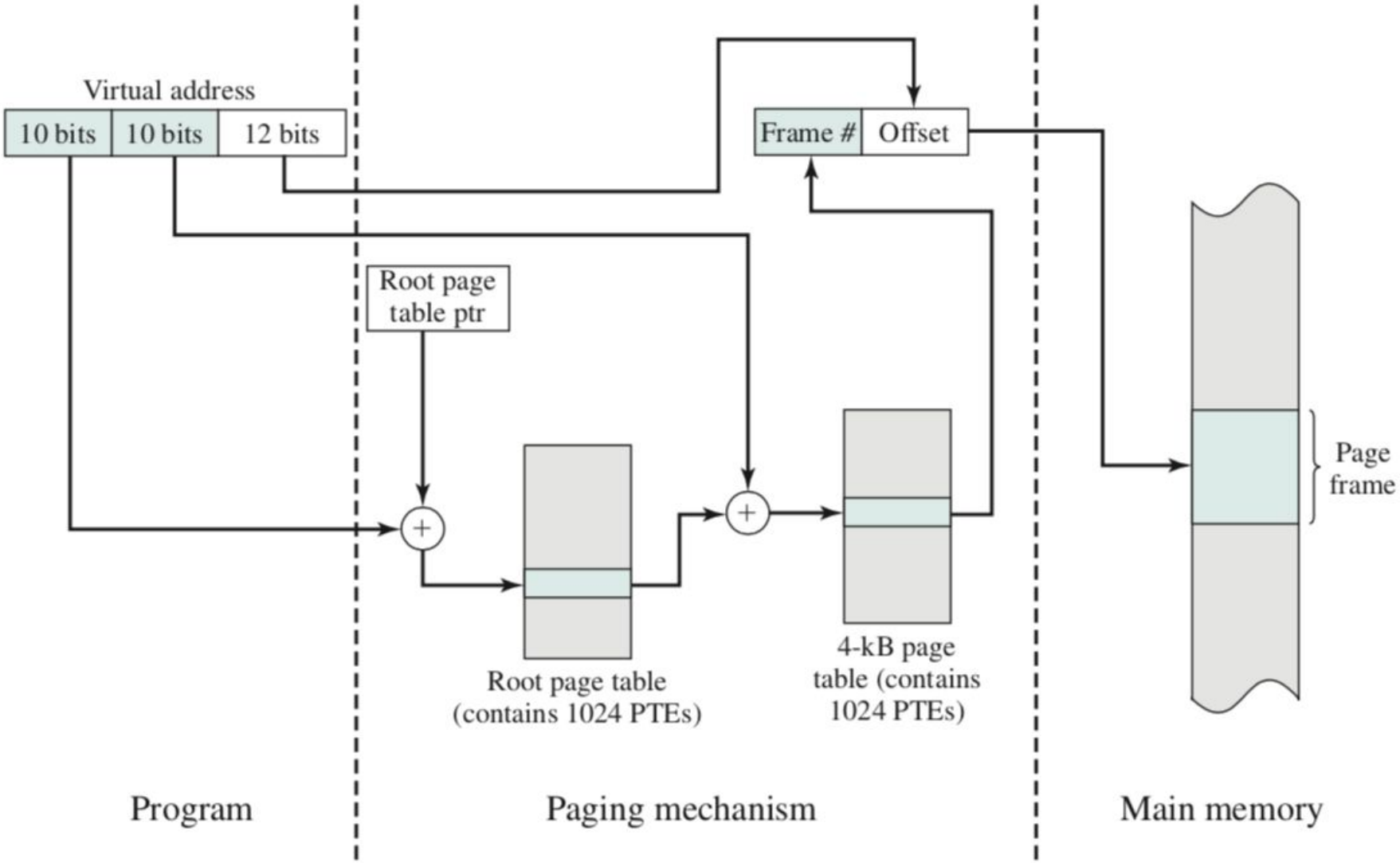


Skąd jądro bierze pamięć po rozruchu?

1. Mamy statycznie przydzielone pule pamięci w sekcji `.bss`, gdyby było trzeba przydzielić pierwsze rekordy.
2. Tablica stron jest w stanie pomieścić dodatkowe odwzorowania za końcem sekcji `.bss`
3. Możemy modyfikować skonstruowaną mapę pamięci.

Da się zrobić alokator stosowy! Wystarczy powiększyć segment pamięci fizycznej używany przez jądro i dodać nowe odwzorowania za sekcją `.bss` w jądrze.

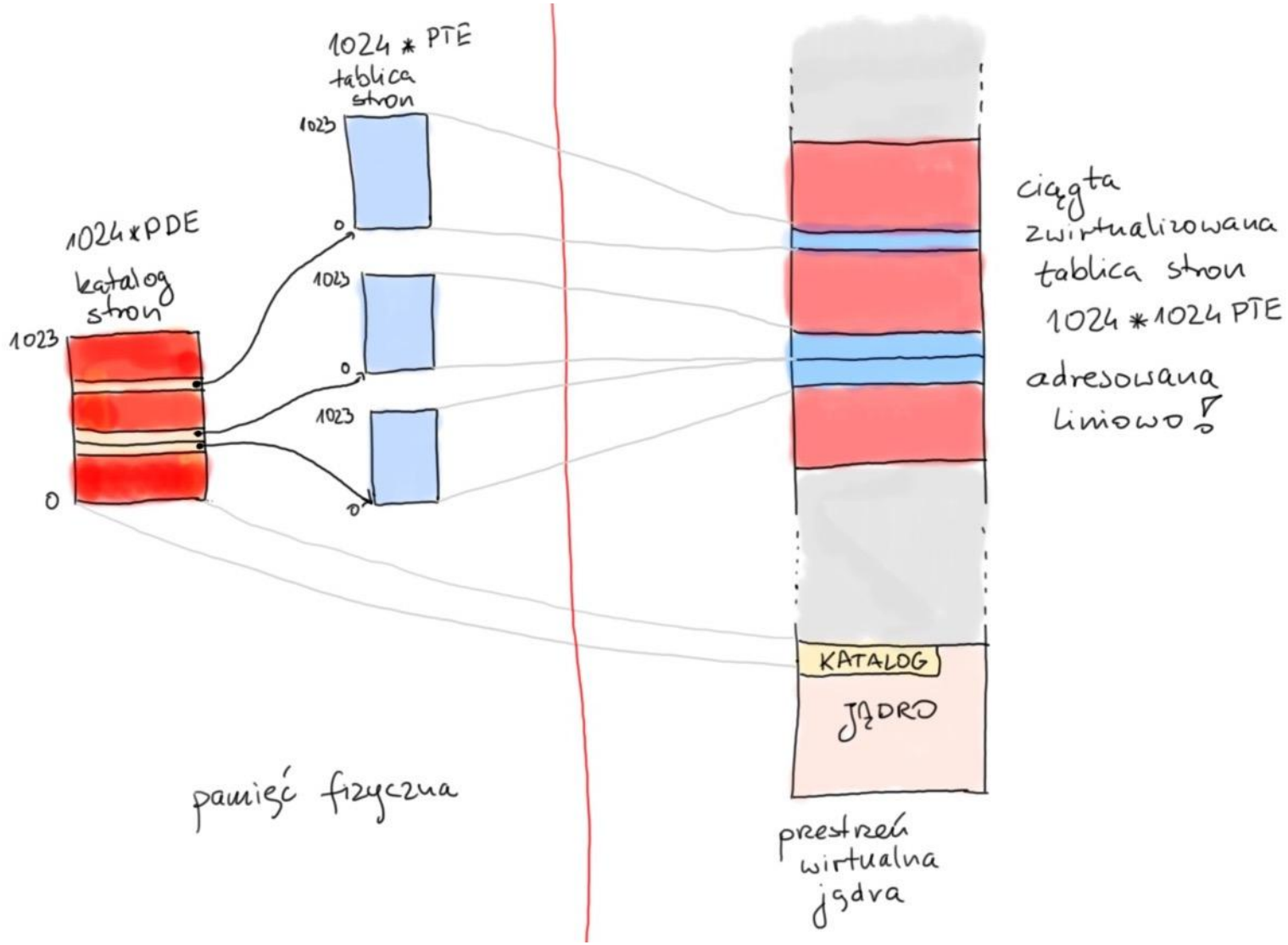
Przypomnienie: dwupoziomowa tablica stron



Kilka faktów na temat tablicy stron

1. W trakcie chybienia TLB procesor robi do niej dostępy z użyciem adresów fizycznych.
2. Katalog zawsze musi być w pamięci fizycznej.
3. Tablice stron rozmiaru 4KiB muszą pokrywać tylko odwzorowane obszary.
4. Żeby jądro mogło zarządzać wpisami, katalog i tablice stron muszą być odwzorowane w adresy wirtualne!
5. Z reguły mamy osobną hierarchiczną tablicę stron dla jądra i po jednej dla każdej przestrzeni użytkownika.

Liniowa zwirtualizowana tablica stron



Zarządzanie pamięcią fizyczną

`vm_phys`: zarządzanie pamięcią fizyczną

Wykorzystujemy algorytmem bliźniaków (ang. *buddy systems*).

- przydział i zwalnianie tylko blokami 2^k stron zaczynającymi się od adresu podzielnego przez (2^k * `PAGESIZE`)
- szybkie operacje w $O(h)$ ($h \rightarrow$ wysokość drzewa binarnego)
- struktury dla algorytmu przydzielone w trakcie rozruchu jądra
- nie potrzebuje żadnej dodatkowej pamięci

Strony pobrane przy użyciu `vm_phys` należy odwzorować w adresy wirtualne. Fragmentację zewnętrzną niwelujemy translacją adresów! Tzn. nieciągłe obszary pamięci fizycznej odwzorowujemy w ciągły obszar adresów wirtualnych.

vm_phys_seg: segmenty pamięci fizycznej

W trakcie rozruchu tworzony jest opis segmentów pamięci fizycznej. Ramki są opisywane przez strukturę vm_page.

```
#define VM_NFREEORDER 10      // Liczba list wolnych bloków stron

typedef TAILQ_HEAD(pglist, vm_page) pglist_t;

struct vm_phys_seg {
  paddr_t  start, end;           // Przedział adresów fizycznych stron
  pglist_t freeq[VM_NFREEORDER]; // Lista bloków stron rozmiaru 2^k
};
```

Wszystkie ramki są przechowywane w globalnej tablicy.

```
vm_page_t vm_page_array[];
```

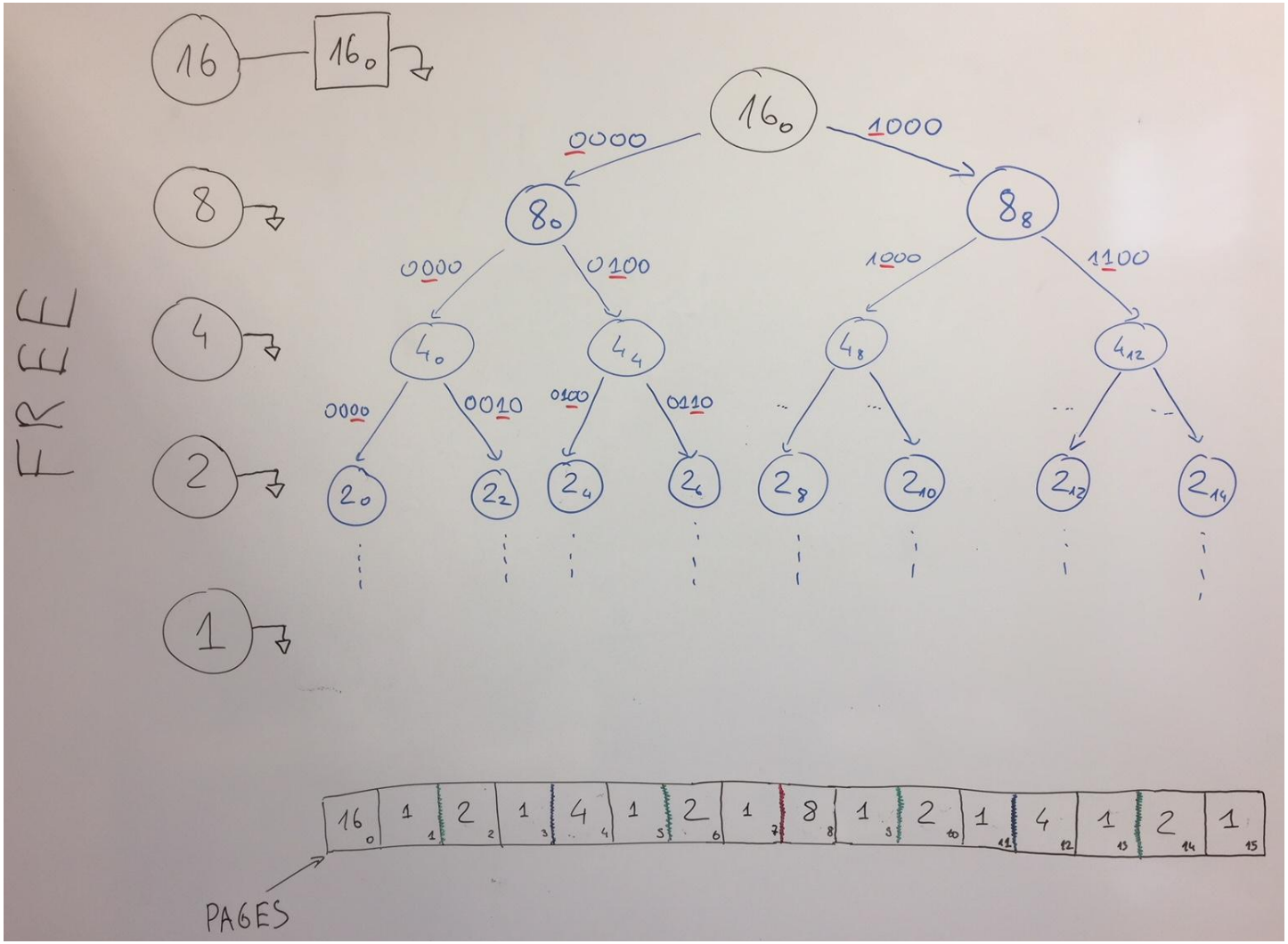
vm_page: opis strony

Założenie → vm_page opisuje zawsze stronę rozmiaru 4KiB !
Spójny ciąg stron będziemy nazywać **blokiem stron**.

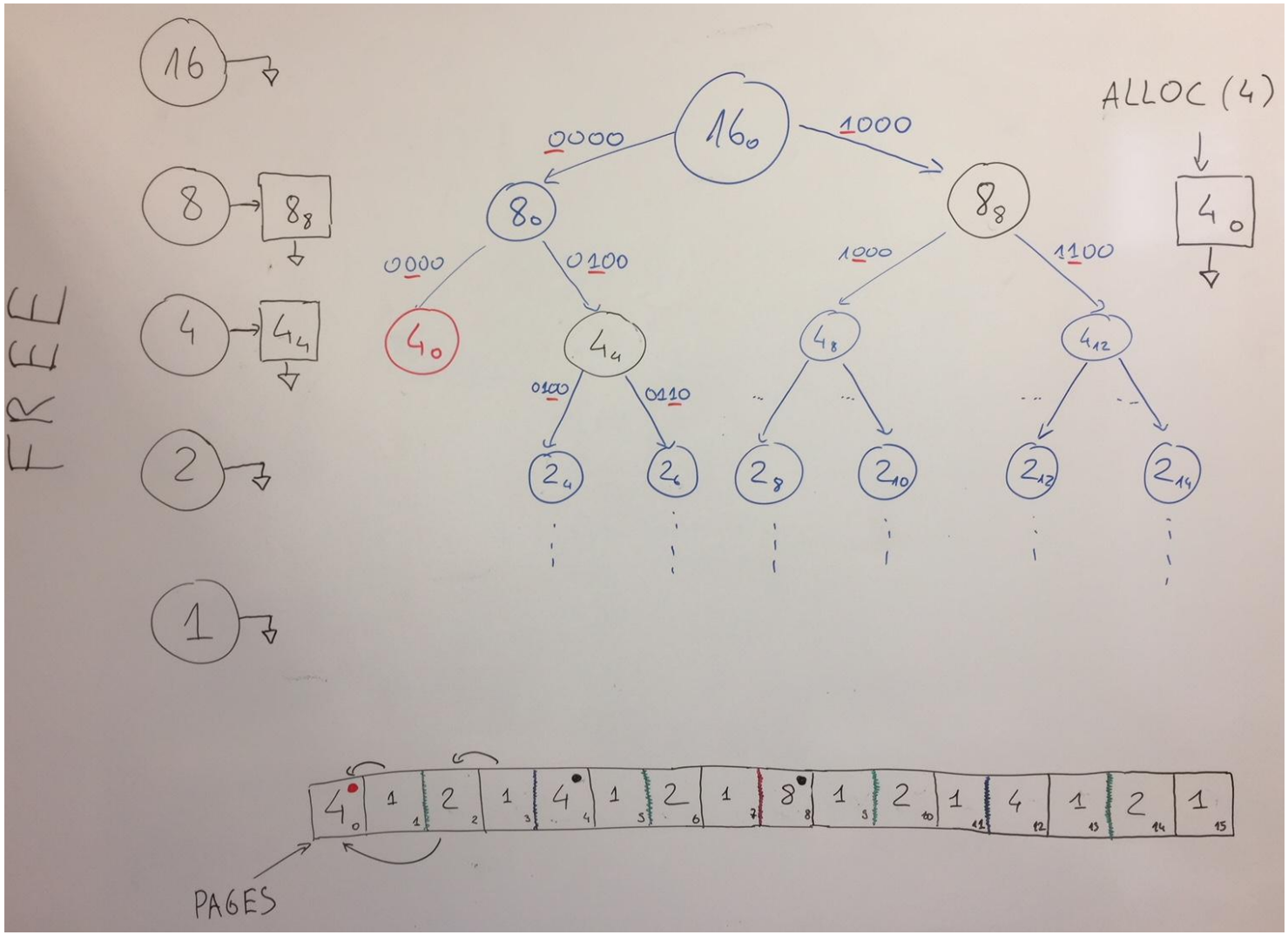
```
struct vm_page {
  TAILQ_ENTRY(vm_page) link; // Węzeł na liście wolnych bloków
  paddr_t phys_addr;         // Adres fizyczny strony
  uint8_t order;             // Rozmiar bloku w stronach (2^k)
  ...
};
```

Pole order wyznacza numer listy vm_phys_seg::freeq wolnych bloków, na którą wpięta jest strona reprezentująca dany blok.

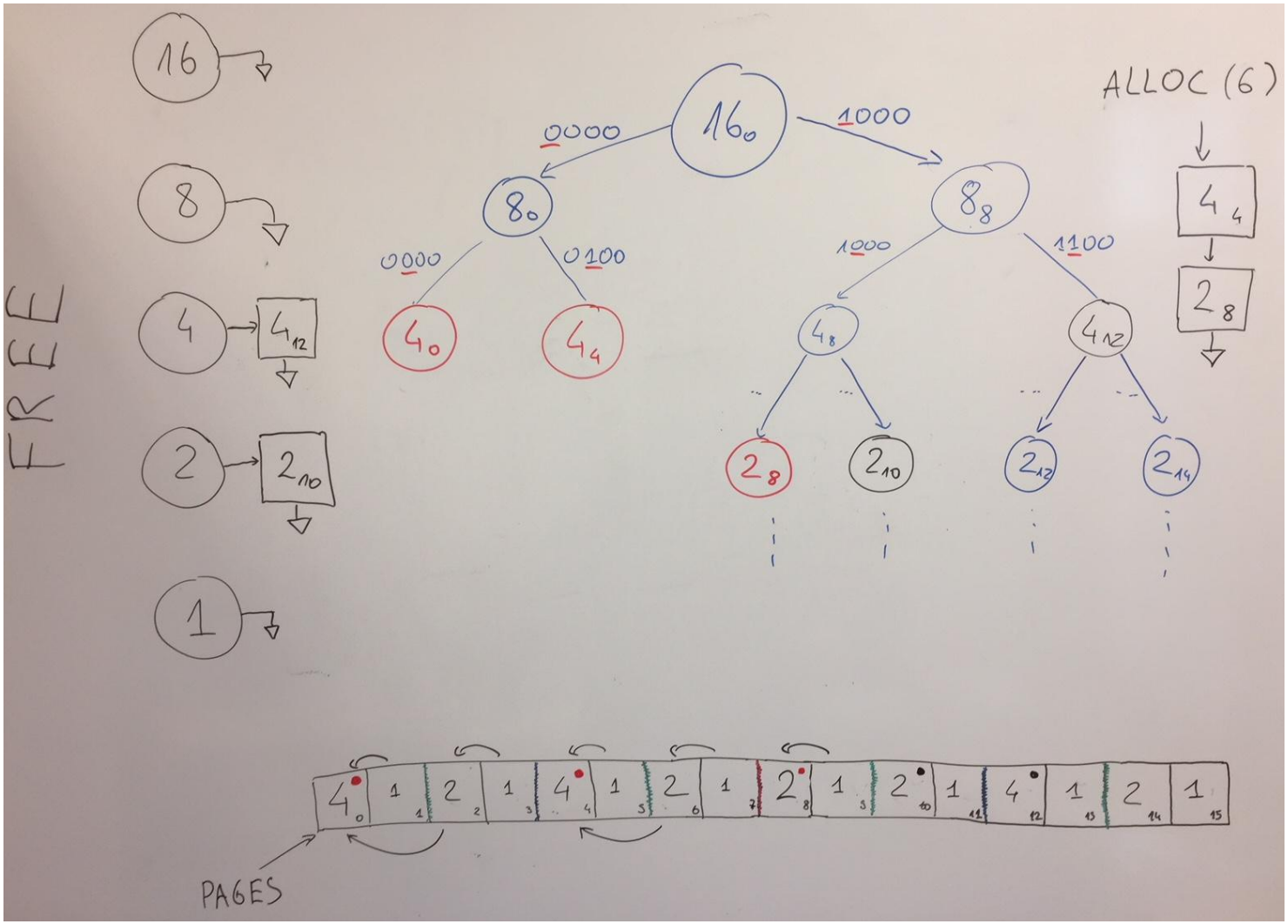
Algorytm bliźniaków: stan początkowy



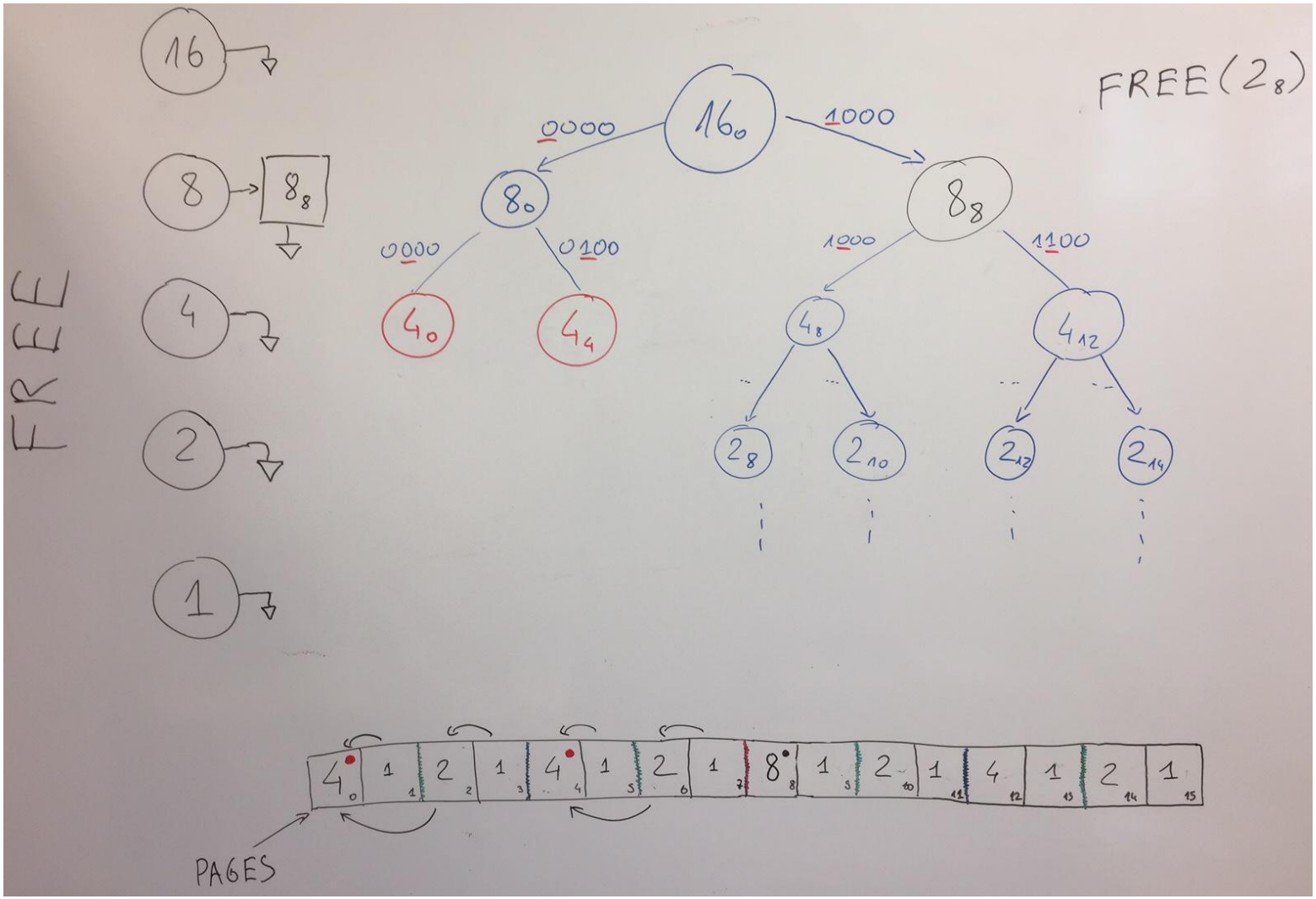
Algorytm bliźniaków: ALLOC(4)



Algorytm bliźniaków: ALLOC(6)



Algorytm bliźniaków: FREE(2_8)



Linux: Algorytm bliźniaków

/proc/buddyinfo

Order	O0	O1	O2	O3	O4	O5	O6	O7	O8	O9	O10
DMA	2	3	3	0	3	2	0	0	1	1	3
DMA32	8	9	9	9	10	7	5	10	7	7	444
Normal	6017	5582	2165	1278	575	166	71	18	7	7	2112

/proc/pagetypeinfo

	Unmovable	Movable	Reclaimable
DMA	1	7	0
DMA32	2	950	0
Normal	114	6558	416

Jądro może kompaktować jeśli zbyt dużo bloków małego stopnia!

Zarządzanie tablicą stron

Przełączanie przestrzeni adresowych

Zmieniamy wskaźnik na tablicę stron – to wszystko? **NIE!**

W **TLB** wpisy ze starej tablicy stron… Pod tymi adresami w innym procesie jest coś innego → opróżnić? **NIEDOBRZE!**

Co z pamięcią podręczną? Tagowana adresami fizycznymi → ok!
Tagowana wirtualnymi (często dla L1) → opróżnić cache? **KOSZT!**

Sprzęt oferuje pulę (2^n gdzie n małe) **identyfikatorów przestrzeni adresowych** (**`ASID`**). Każdy wpis w TLB i cache ma pole ASID, które sprzęt porównuje z zawartością uprzywilejowanego rejestru.

`Mach3:` Zarządzanie tablicą stron

Implementacja translacji adresów i tablicy stron mogą się znacząco różnić między architekturami (`Intel` vs. `PowerPC` vs. `MIPS`).

Potrzebujemy abstrakcyjnego interfejsu do zarządzania translacją adresów, uprawnieniami stron, bitami monitorowania dostępu, itp.

Moduł pmap (ang. *physical map*) zaprojektowany dla jądra Mach! Początek lat ‘90. Używany obiecnie w systemach BSD i MacOS X.

Zarządza pamięcią niezbędną do zbudowania struktur danych dla sprzętowego lub programowego przeglądania tablicy stron. Emuluje funkcje niedostępne w sprzęcie. Zna format wpisów tablicy stron. Zarządza sprzętowymi numerami przestrzeni adresowych (ASID). Unieważnia wpisy w TLB i pamięci podręcznej.

pmap: relacja między ramkami, a stronami

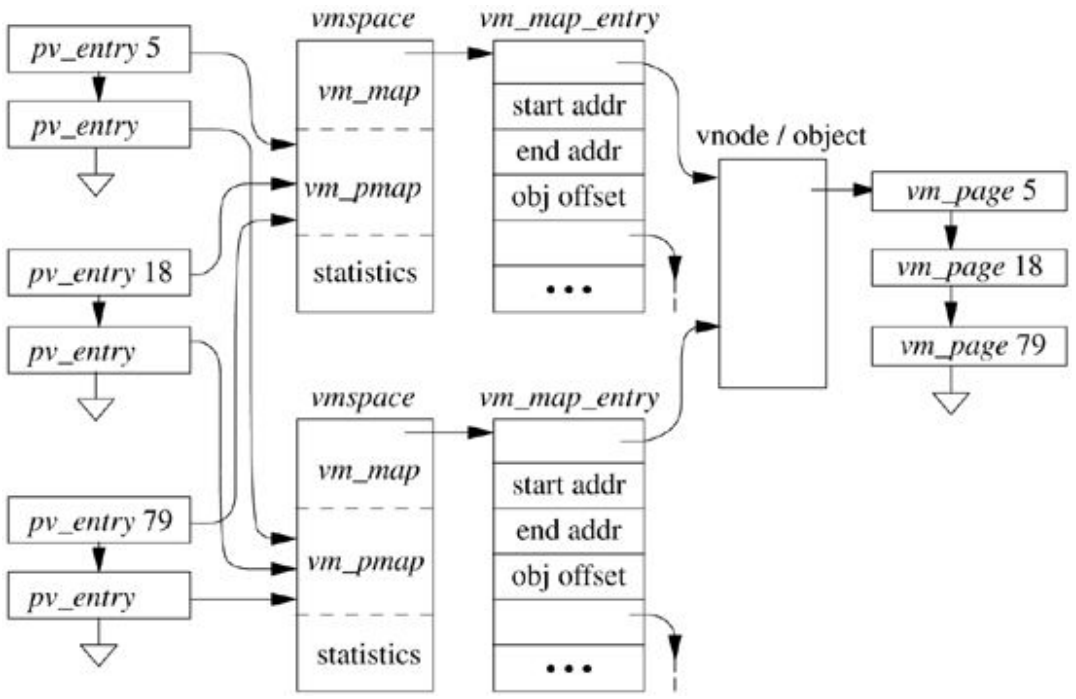
UWAGA! `vm_page` to opis ramki → nie ma adresu wirtualnego!

Jak znaleźć **vmspace** do których została podczepiona ramka?

md_page czyli część **vm_page** zależna od architektury sprzętowej przechowuje głowę listy **pv_entry**

```
struct md_page {
    TAILQ_HEAD(, pv_entry) pv_list;
};

struct pv_entry {
  pmap_t pv_pmap;
  vm_offset_t pv_va;
  TAILQ_ENTRY(pv_entry) pv_link;
};
```



pmap: podczepianie ramek i zmiana uprawnień

```
int pmap_enter(pmap_t, vm_addr_t va, vm_page_t pg, vm_prot_t, ...);
                    void pmap_zero_page(vm_page_t);
                void pmap_copy_page(vm_page_t, vm_page_t);
        void pmap_protect(pmap_t, vm_addr_t, vm_addr_t, vm_prot_t);
```

Możliwe uprawnienia: **VM_PROT_{NONE,READ,WRITE,EXECUTE,COPY}**

pmap_enter wprowadza mapowanie ramki pg pod adresem va z odpowiednimi prawami dostępu, wołane przy błędzie strony

pmap_zero_page zanim udostępnimy anonimową stronę, należy ją podpiąć do adresów wirtualnych jądra (KVA) i wyzerować
pmap_copy_page j.w, używane przy klonowaniu strony przy błędzie strony spowodowanym działaniem mechanizmu *copy-on-write*

pmap_protect zmienia uprawnienia dostępu do przedziału stron

pmap: odczepianie ramek

```
void pmap_remove(pmap_t, vm_addr_t, vm_addr_t);
void pmap_remove_all(vm_page_t);
void pmap_remove_write(vm_page_t);
```

pmap_remove odczepia strony z podanego przedziału,
np. w wyniku wywołania systemowego **munmap**

pmap_remove_all przegląda listę **pv_entry** związanych z daną ramką i odczepia je od odpowiednich przestrzeni wirtualnych, używane przez algorytm wymiany ramek

pmap_remove_write używane wewnętrznie przez podsystem pamięci wirtualnej do skonfigurowania ramki do użycia jako *copy-on-write* przy wywołaniu fork

pmap: bity dostępu i programowa translacja adresu

```
boolean_t pmap_is_modified(vm_page_t);
  void pmap_clear_modify(vm_page_t);
  int pmap_ts_referenced(vm_page_t);
```

pmap_is_modified sprawdza bit `modified` dla danej ramki
pmap_clear_modified czyści bit `modified` dla danej ramki
pmap_ts_referenced zwraca wartość licznika dostępów (jeśli sprzęt to udostępnia) dla danej ramki i czyści go

```
vm_paddr_t pmap_extract(pmap_t, vm_addr_t);
```

pmap_extract tłumaczy podany adres wirtualny na fizyczny według bieżącej zawartości sprzętowej tablicy stron

`pmap:` przełączanie przestrzeni adresowych

```
void pmap_activate(thread_t *);
```

Wołane jeśli jądro chce coś skopiować do przestrzeni adresowej wątku lub przełączyć na jego kontekst.

Sprzęt oferuje identyfikatory przestrzeni adresowych?

Tak → jądro utrzymuje listę aktywnych procesów i przypisuje im dostępne ASID. Za dużo procesów? Któryś wypada ze zbioru i jest zastępowany! Czyścimy wpisy TLB i linie cache z ustalonym ASID.

Nie → czyścimy wszystko jak idzie.

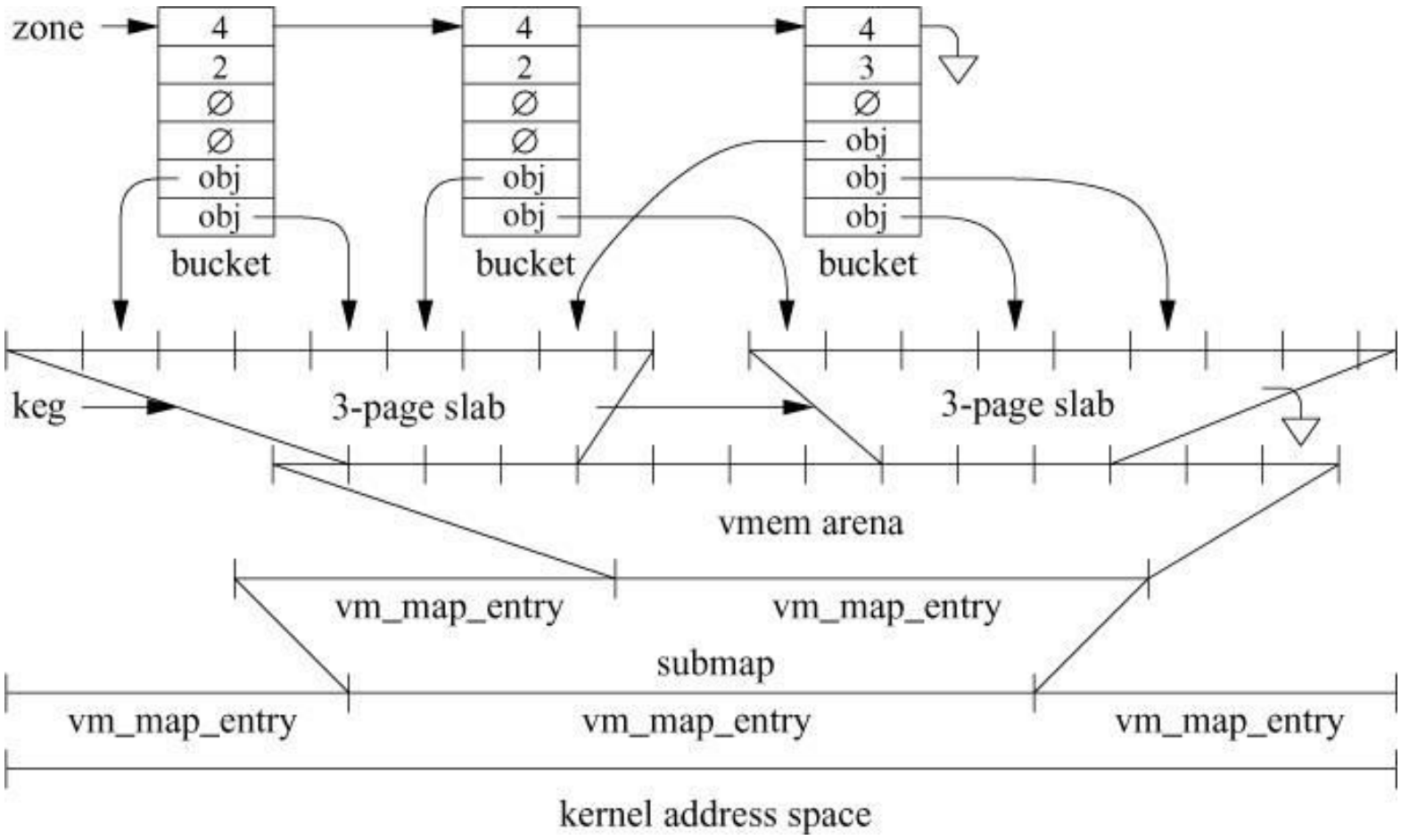
Zarządzanie pamięcią jądra

Obiekty do zarządzania pamięcią jądra

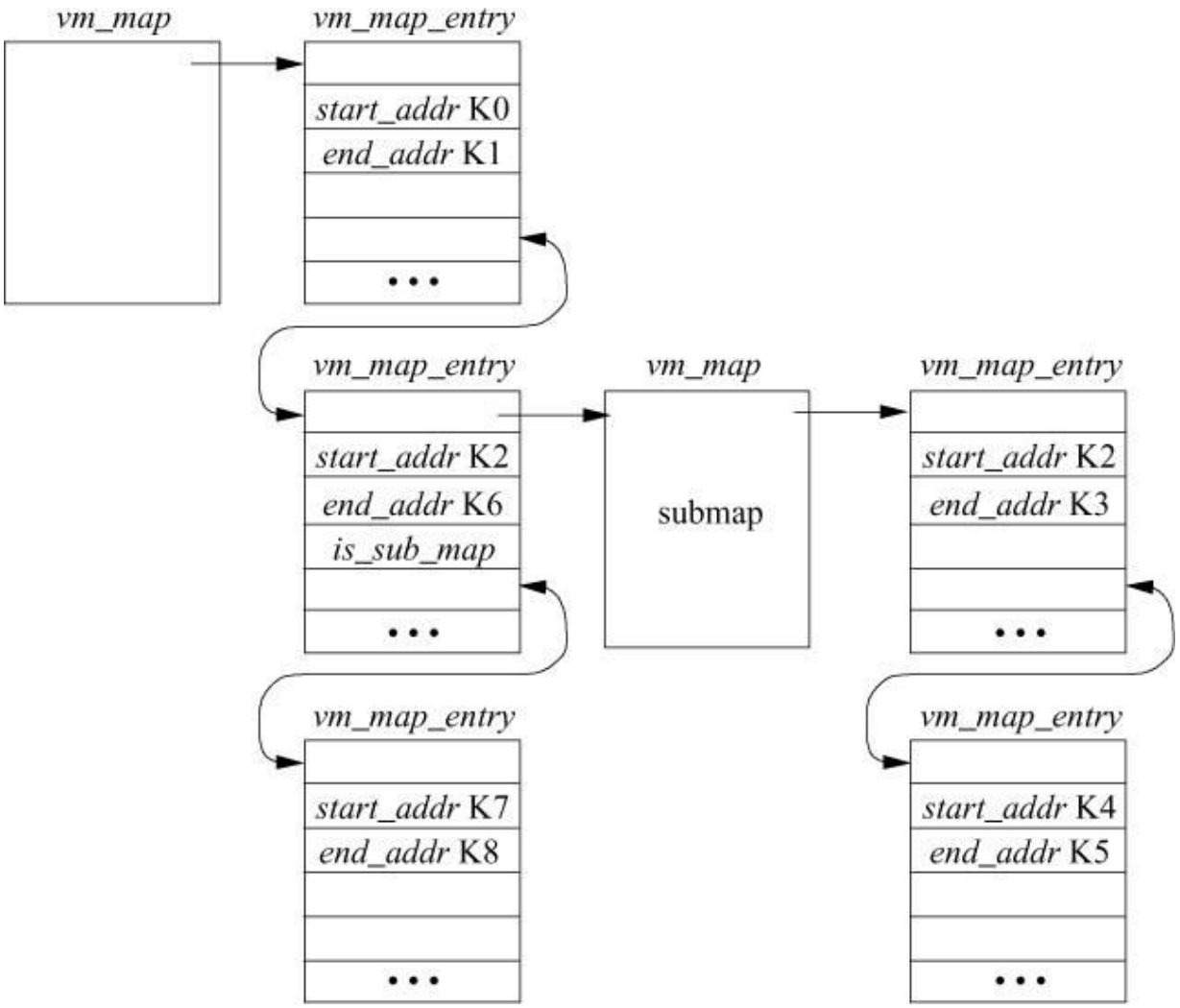
buckets	przydział obiektów per CPU
zones	przydział obiektów z ***kegs*** do ***buckets***
kegs	zbiory slabs przechowujące obiekty danego typu
slabs	przydział zbiorów obiektów z areny ***vmem***
vmem	przydział bloków stron w obrębie ***vm_map***
vm_map	partycjonowanie przestrzeni adresowej

Czemu aż tyle warstw?

Hierarchia obiektów zarządzania pamięcią jądra



Organizacja mapy wirtualnej pamięci jądra



K0-K1
obraz jądra

K2-K6
pamięć jądra

K2-K3
pamięć dla ***execve***

K4-K5
pamięć dla ***pipe***

K7-K8
pamięć buforów podlegających I/O

Motywacja: kernel maps & submaps

1. Podmapy do izolacji zarządzania pamięcią komponentów:
   a. zapobiega fragmentacji: podobny rozmiar i czas życia
2. Górne ograniczenie na wielkość pamięci dla podsystemu.
3. Rozróżnienie na pamięć stronicowalną i zadrutowaną.
4. Spełnienie specyficznych wymogów: wyrównanie adresu wirtualnego, odwzorowanie ciągłej pamięci fizycznej.

vmem: zarządzanie zasobami ?

Magazines and Vmem:
Extending the Slab Allocator to Many CPUs and Arbitrary Resources
Jeff Bonwick, Jonathan Adams; USENIX ATC 2001

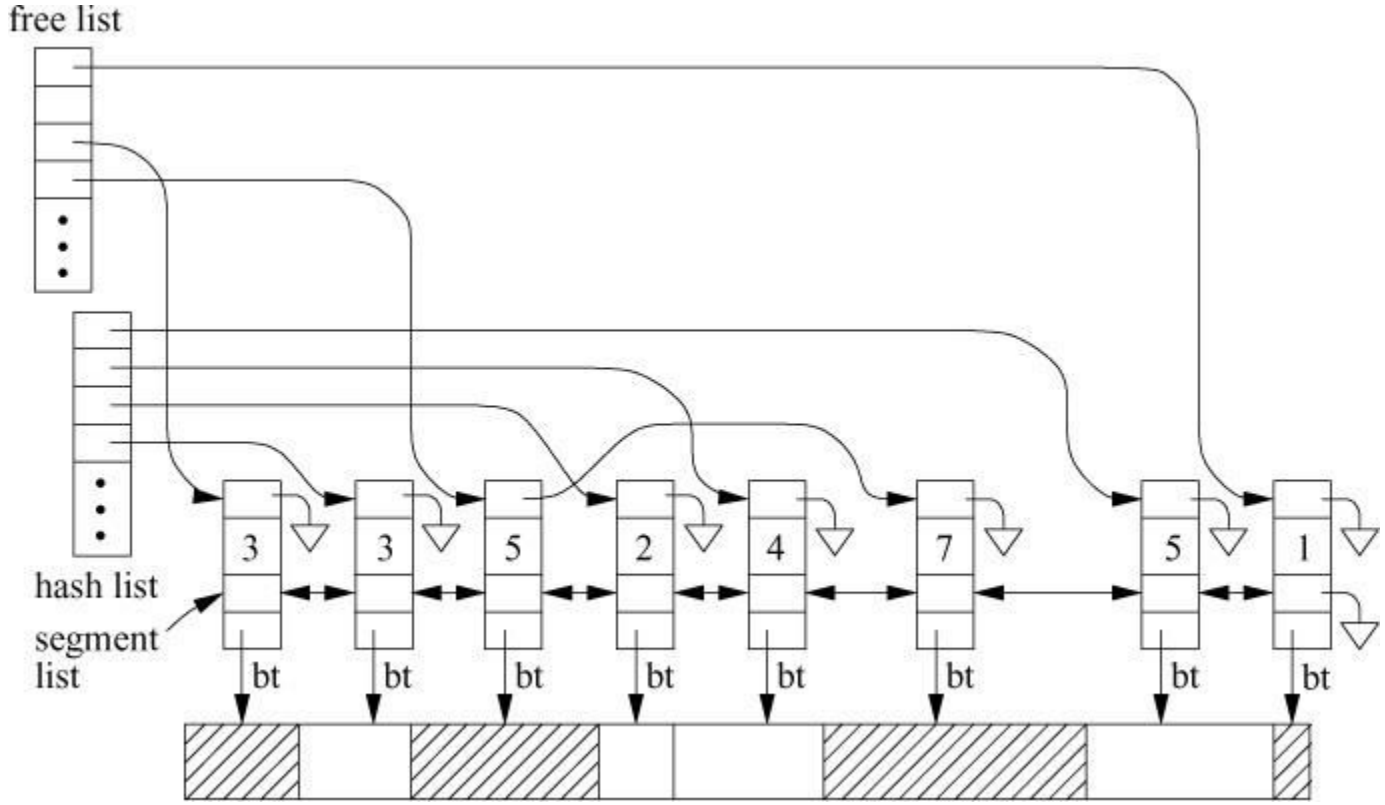
Algorytm przydziału dla ***vm_map*** mało efektywny:

1. Działa w czasie liniowym (*first-fit*), ew. logarytmicznym: szuka wolnej przestrzeni między dwoma ***vm_map_entry***.
2. W dłuższym okresie działania mocno fragmentuje.

vmem tylko i wyłącznie zarządza obszarami adresów!

Należy jeszcze przydzielić ***vm_phys_alloc_pages*** i podpiąć pod adresy wirtualne ***pmap_enter***.

vmem: efektywne zarządzanie przedziałami liczb całkowitych



Nie zarządzamy pamięcią więc ***bt*** (boundary tags) przechowujemy w wydzielonej pamięci, a nie w nagłówku bloku.

vmem: przebieg operacji

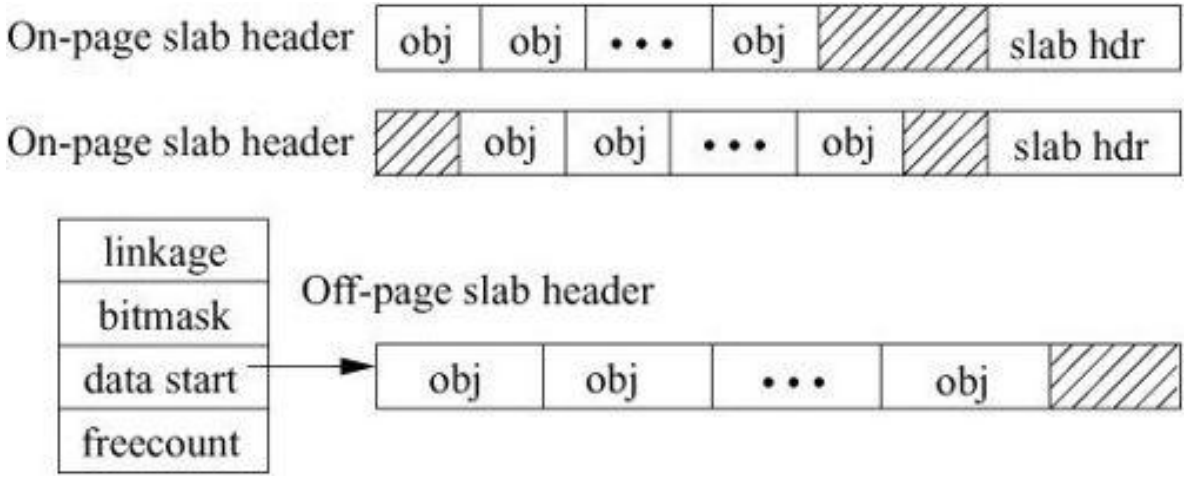
Znaczniki oznaczające wolne obszary są trzymane w kubełkach o wykładniczo rosnących rozmiarach → aproksymacja best-fit przy pomocy segregated-fit aka good-fit. Strategie przydziału:

- ***M_FIRSTFIT***: pierwszy, który jest większy lub równy,
- ***M_BESTFIT***: minimalizacja fragmentacji,
- ***M_NEXTFIT***: zapobiega fragmentacji w przypadku regionów o podobnym czasie życia.

Przy zwalnianiu haszujemy adres początkowy obszaru, szukamy w hash list i sprawdzamy czy zgadza się rozmiar.

Po drodze robimy standardowo lazy-split i eager-merge.

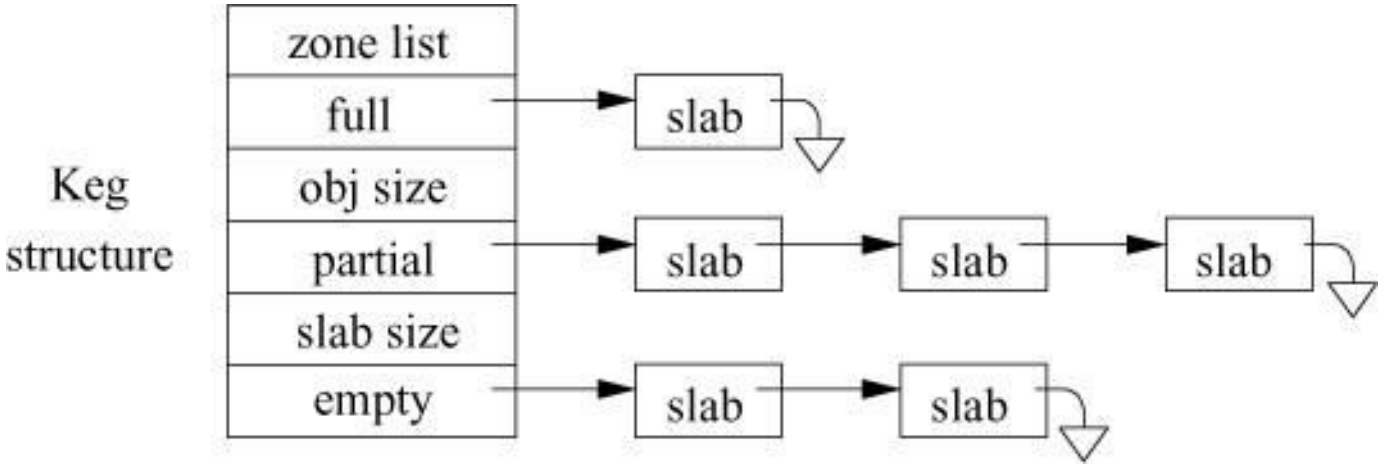
Alokator płytowy (ang. slab allocator)



Przechowuje obiekty tego samego rozmiaru i bitmapę wolnych obiektów. Płyta ma rozmiar wielokrotności strony. Zmienny offset obiektów względem początku płyty, żeby ograniczyć konflikty w zbiorach pamięci podręcznych.

Jak wyznaczyć szybko slab w trakcie zwalniania bloku?
Trik ze wskaźnikiem na `slab` w `vm_page` i `pmap_kextract`.

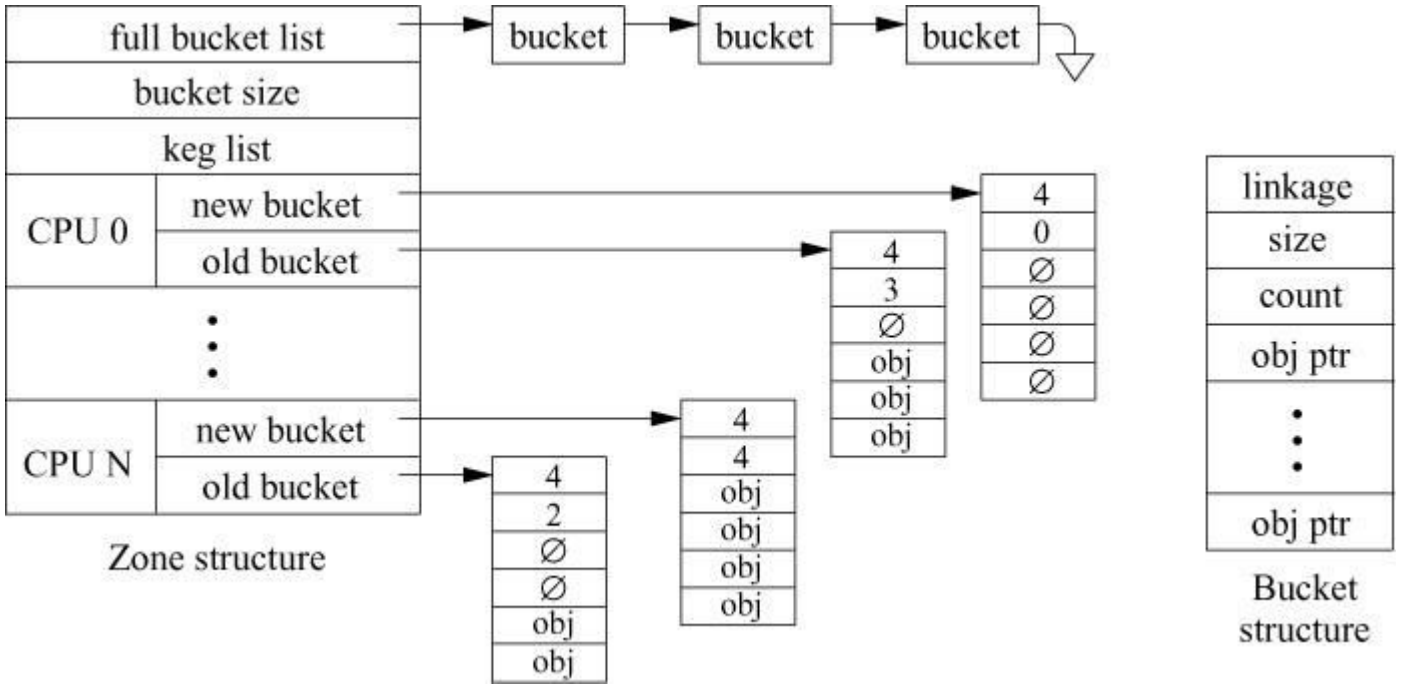
kegs: zbiory płyt



Antałek (ang. keg) zawiera płyty przechowujące obiekty tego samego typu i w dodatku częściowo zainicjowane, żeby przyspieszyć działanie konstruktora obiektu.

`obj size`: rozmiar elementów składowanych na płytach
`slab size`: rozmiar płyty (wielokrotność rozmiaru strony)
`full/partial/empty`: w pełni / częściowo zajęte płyty, wolne płyty

Alokator strefowy (ang. zone allocator)



Efektywny alokator pamięci dla systemów SMP. Alokacje ze stref mogą przebiegać współbieżnie. Wiaderko (ang. bucket) trzyma wskaźniki na wolne obiekty z antałków. Każdy procesor ma swój zestaw wiaderek.

zones: zarządzanie obiektami w ***buckets***

Idea: procesor może przydzielić lub zwolnić do M obiektów bez zakładania blokady na listę wiaderek albo strefę.

W obrębie strefy każdy procesor może przydzielać i zwalniać niezależnie od innych. Jeśli nie może spełnić żądania przydziału, to musi iść do listy wiaderek, a potem antałków. Jeśli tam nie ma miejsca, to trzeba dodać płyty. Im wyżej w hierarchii tym blokady mniej drobnoziarniste.

Demon stronicowania może zabierać wolne płyty z antałków.

`malloc(9)` korzysta ze stref zawierających bloki o wykładniczo rosnących rozmiarach → znowu good-fit. Duże bloki idą do `vmem`.

Zarządzanie pamięcią stronicowalną

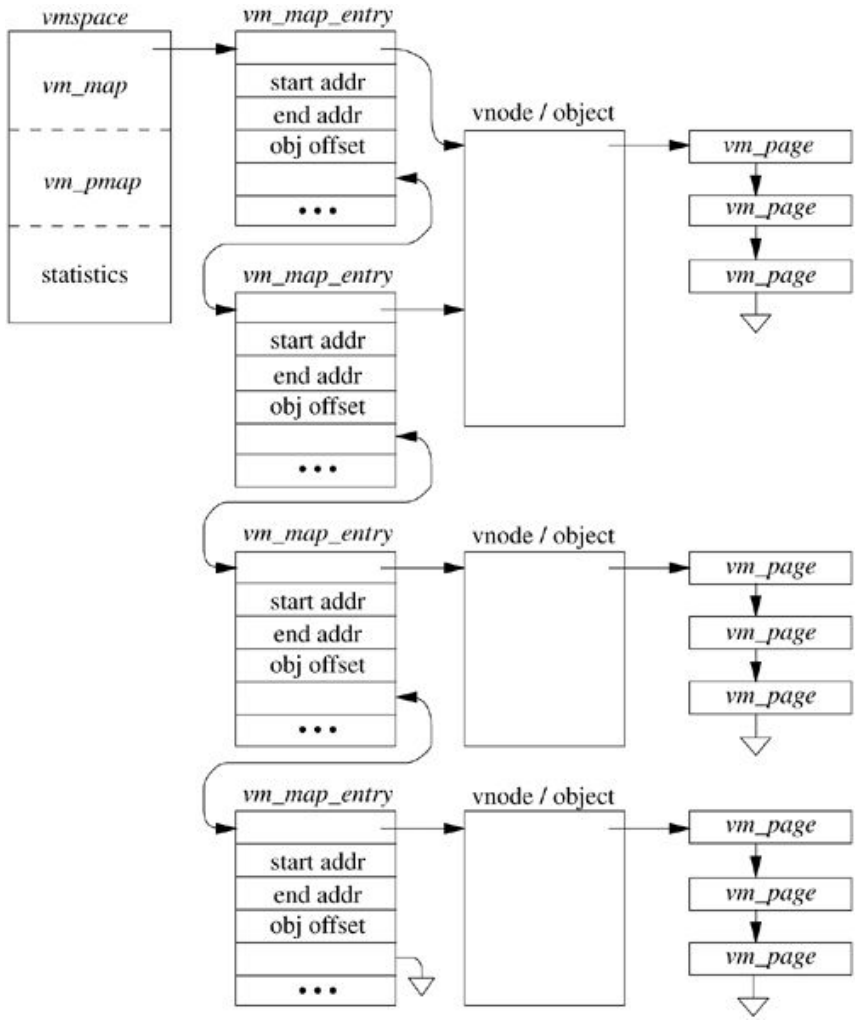
FreeBSD: Zarządzanie przestrzenią adresową

`vmspace` przechowuje tablicę stron (`pmap`), statystyki, wskaźniki do segmentów `text`, `data`, `bss`, oraz listę opisów obszarów adresów wirtualnych → **`vm_map_entry`**

`vm_object` dostarcza stron **`vm_page`**, które widać w danym przedziale adresów wirtualnych

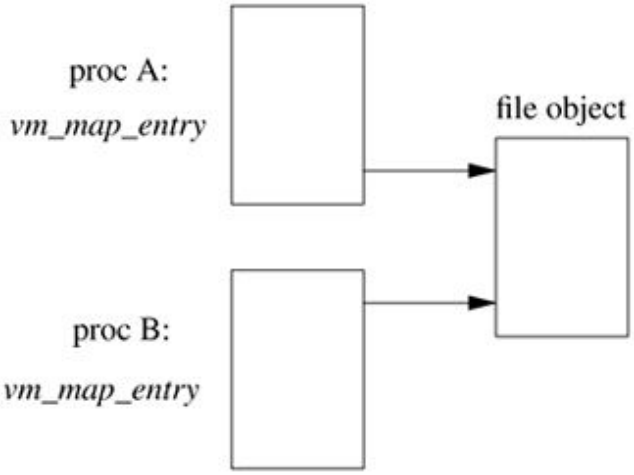
Linuksowy opis przestrzeni adresowej: **`cat /proc/$pid/maps`**

(`start`, `end`, `prot`, `offset`, `dev`, `inode`, `path`)



FreeBSD: Obiekt i procedury stronicujące

Operation	Description
pgo_init()	initialize pager
pgo_alloc()	allocate pager
pgo_dealloc()	deallocate pager
pgo_getpages()	read page(s) from backing store
pgo_putpages()	write page(s) to backing store
pgo_haspage()	check whether backing store has a page
pgo_pageunswapped()	remove a page from backing store (swap pager only)

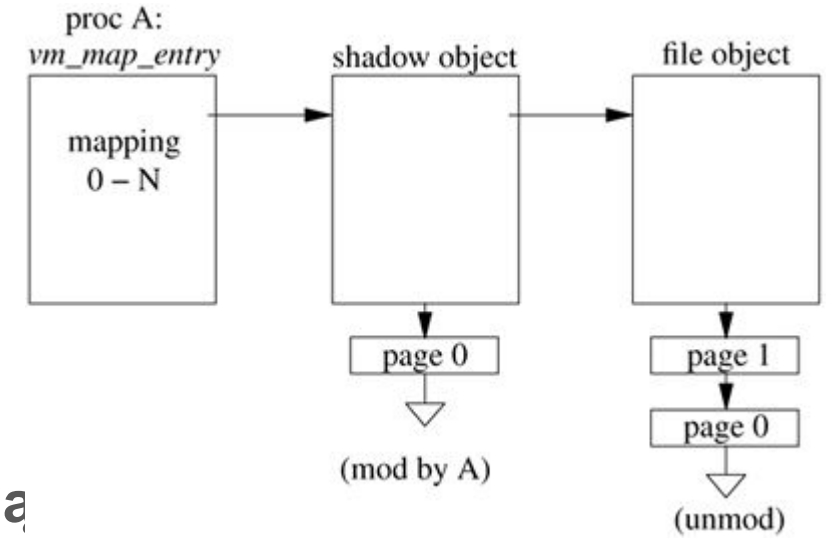


Z każdym obiektem skojarzona lista stron, procedury stronicujące, licznik referencji, itp. Obiekt może odpowiadać pamięci anonimowej, plikowi, urządzeniu. Można go też współdzielić między procesy!

`swap pager` → **`getpages`** zwraca wyzerowaną stronę anonimową
`vnode pager` → **`getpages`** przydziela stronę i ładuje do niej kawałek pliku

FreeBSD: Mapowanie prywatne plików

`MAP_PRIVATE` dla pliku tworzy odwzorowanie z kopią przy zapisie. Zmiany nie są zapisywane do pliku i nie są widziane w pozostałych procesach, które mapują ten zasób.



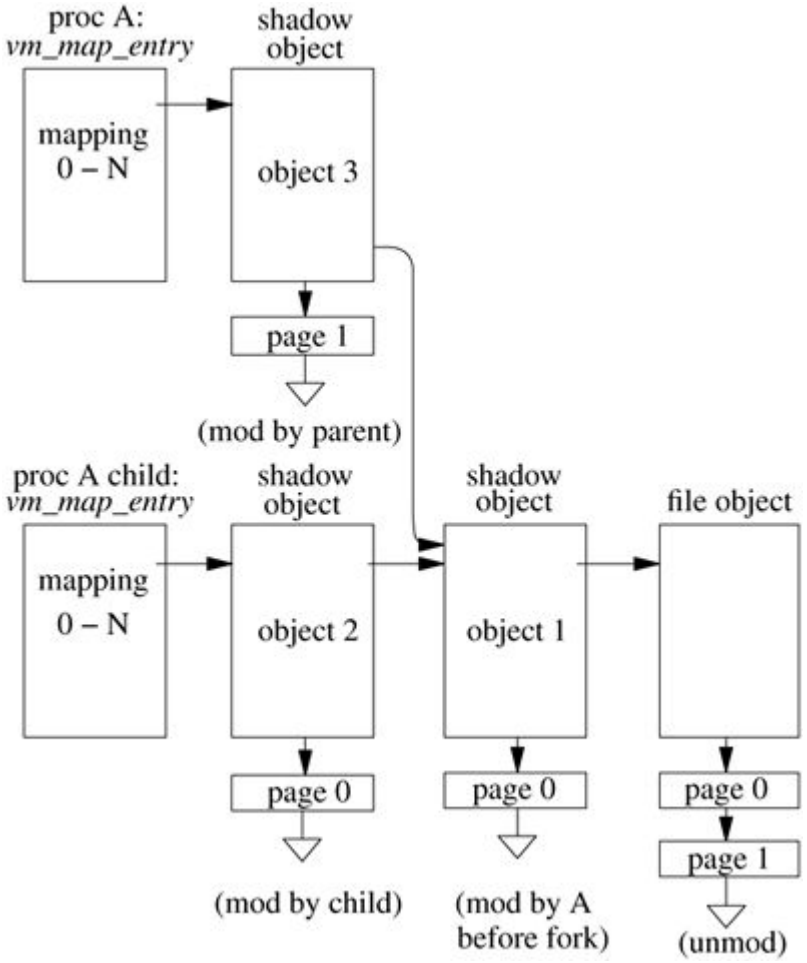
Potrzebujemy **obiektów przesłaniaja**
(ang. *shadow object*). Oryginalne strony są tylko do odczytu! Kiedy zapisujemy → błąd strony! Przydzielamy stronę anonimową, kopiujemy zawartość oryginału (ang. *copy-on-write*), podczepiamy do obiektu przesłaniającego i wpisujemy do tablicy stron (pmap_enter) w miejsce oryginału.

FreeBSD: Klonowanie przestrzeni adresowych

W momencie klonowania (**`fork`**) tworzymy rodzicowi i dziecku obiekty przesłaniające oryginalną zawartość przestrzeni adresowej, a oryginalne strony ustawiamy tylko do odczytu.

Q: Co jeśli proces **`A`** się zakończy?
A: Zwalniamy obiekt 3 i jego stronę 1. Zostaje nam ciąg obiektów cieni, które trzeba złożyć (ang. *collapse*).

Q: Co jeśli obiekt 1 zawiera stronę 1?
A: Składamy obiekty w kolejności 2 → 1, i przenosimy do 2 tylko najświeższe kopie.



FreeBSD: Obsługa błędu strony

Obsługujemy wyjątek CPU wstrzymując wątek procesu, odczytujemy rejestry sprzętowe, znajdujemy bieżącą przestrzeń adresową i wołamy:

```
int vm_fault(vm_map_t map, vm_offset_t vaddr, vm_prot_t type)
```

Uproszczona wersja bez optymalizacji i blokad (FreeBSD, §6.11):

1. Przeszukaj listę w poszukiwaniu **`vm_map_entry`**, do którego przynależy **`vaddr`**. Nie → wyślij **`SIGSEGV`** (**`SEGV_MAPERR`**)!
2. Czy obszar posiada stronę, na której leży **`vaddr`**?
   Nie → zawołaj **`pgo_getpages(object, page)`** i podczep stronę!
3. Uprawnienia się nie zgadzają?
   a. `shadow object` → znajdź stronę głębiej, skopiuj i podczep!
   b. `pager` → wyślij **`SIGSEGV`** (**`SEGV_ACCERR`**)

FreeBSD: Klasy stron w jądrze

Podejrzyjmy statystyki pamięci wirtualnej: **`vmstat -s`**

- **`WIRED`** strony przyczepione do pamięci operacyjnej, używane przez jądro lub przypięte wywołaniem **`mlock`**
- **`ACTIVE`** prawdopodobnie należą do zbiorów roboczych procesów, jądro bada użycie tych stron i przenosi do listy **`INACTIVE`**
- **`INACTIVE`** nieużywane i potencjalnie brudne, po wyczyszczeniu trafiają do listy **`CACHE`**, przy błędzie strony wracają do **`ACTIVE`**
- **`CACHE`** nieużywane i czyste, licznik referencji ustawiony na zero
- **`FREE`** strony gotowe do przydzielenia, być może wyzerowane

Demon stronicowania dąży do tego, by na liście **`FREE`** + **`CACHE`** oraz **`INACTIVE`** znajdowało się odpowiednia ilość (procentowo) pamięci.

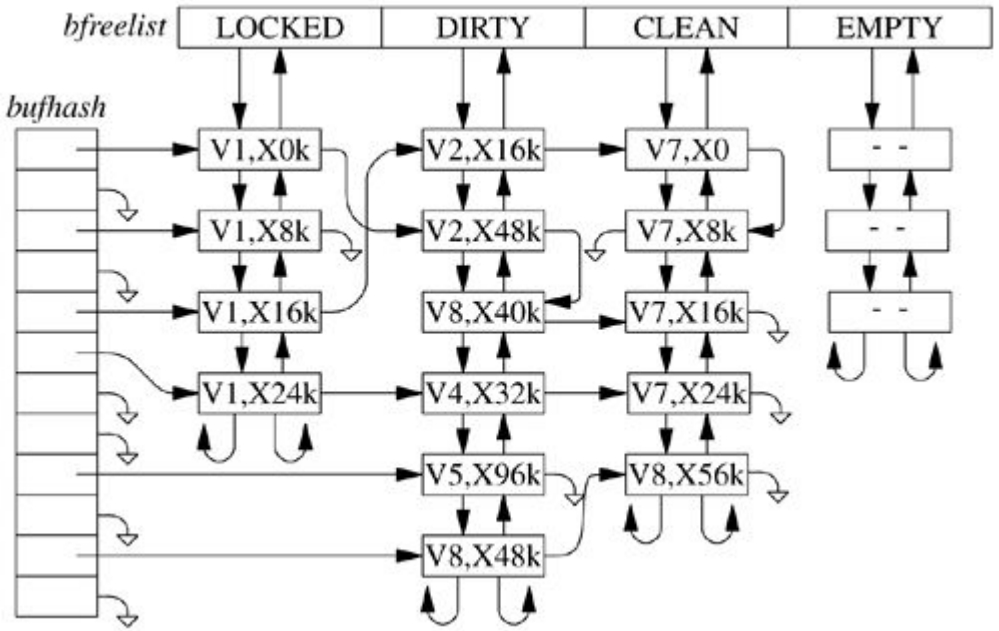
FreeBSD: Buforowanie stron

System posiada osobne bufory dla stron anonimowych oraz stron należących do plików (w tym urządzeń blokowych). Strony i bloki dyskowe są traktowane tak samo → page cache.

Q: Jak wyznaczyć położenie strony należącej do pliku?
A: Potrzebujemy identyfikator niezależny od systemu plików (vnode) i pozycję strony w pliku.

`bufhash` kubełki adresowane parą (**`vnode`**, **`offset`**)

`LOCKED` → na zawartości wykonywane operacje wej.-wyj.



Pytania?