

# Systemy operacyjne

Wykład 9: Pamięć wirtualna

# Po co nam pamięć wirtualna?

Ilość pamięci wirtualnej zużywanej przez wszystkie procesy?

```
ps -A -o vsz | awk '{s+=$1} END {print s/1024}'
```

Ilość faktycznie zużywanej pamięci fizycznej → **vmstat -h**

- programy nie potrzebują całego swojego kodu w pamięci  
(rzadko korzystamy z całej funkcjonalności, w tym kodu obsługi błędów)
- ... i wszystkich swoich danych  
(przydział tablic dla przypadków pesymistycznych)
- prawie cały kod możemy współdzielić  
(sekcja **text** programów jest tylko do odczytu → biblioteki współdzielone)
- ... dane też da się współdzielić, tak długo jak ich nie zmieniamy  
(dzięki temu można implementować efektywnie **fork**)

# Przestrzeń adresowa i stronicowanie

Procesy posługują się **adresami wirtualnymi** i żyją w **wirtualnej przestrzeni adresowej**. Podobnie jest z jądrem, przy czym ono zarządza również zasobami w **fizycznej przestrzeni adresowej**.

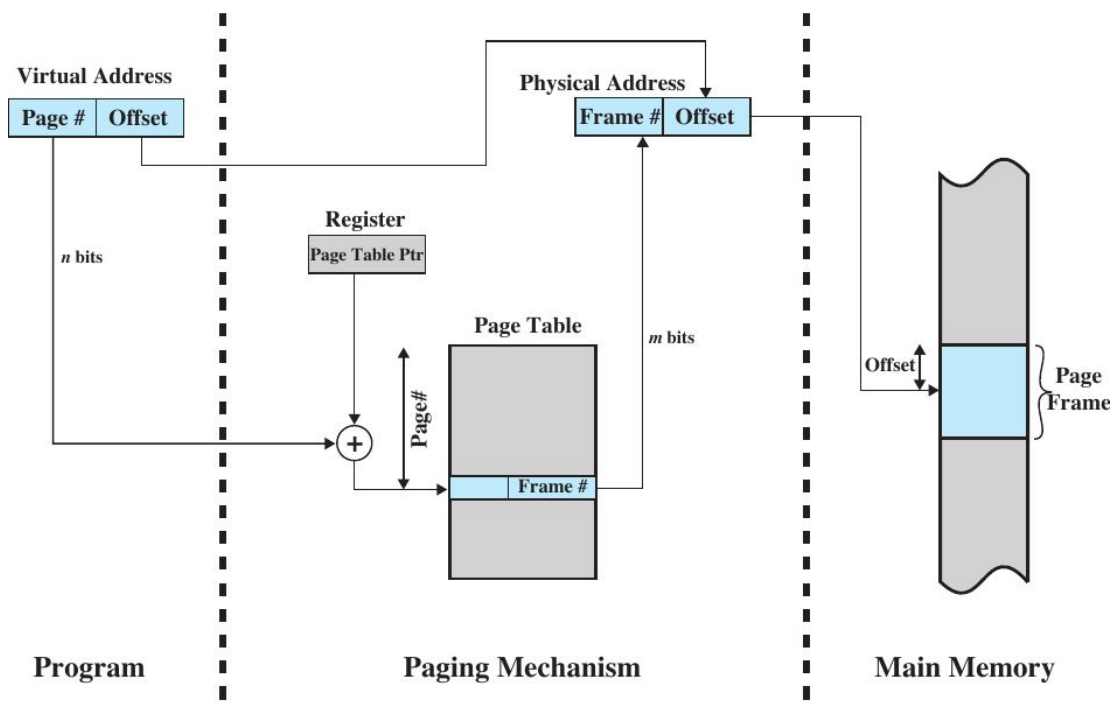
Procesor przy każdym dostępie do pamięci wykonuje **translację adresów** z wirtualnych na fizyczne z użyciem zintegrowanej **jednostki zarządzania pamięcią**.

W **stronicowaniu** cała przestrzeń adresowa jest podzielona na **strony**. Są to bloki pamięci ustalonego rozmiaru  $2^k$ , które zaczynają się zawsze pod adresem podzielnym przez  $2^k$ . Strony leżą w **pamięci wirtualnej**, a **ramki stron** w **pamięci fizycznej**.

# Stronicowanie i translacja adresów

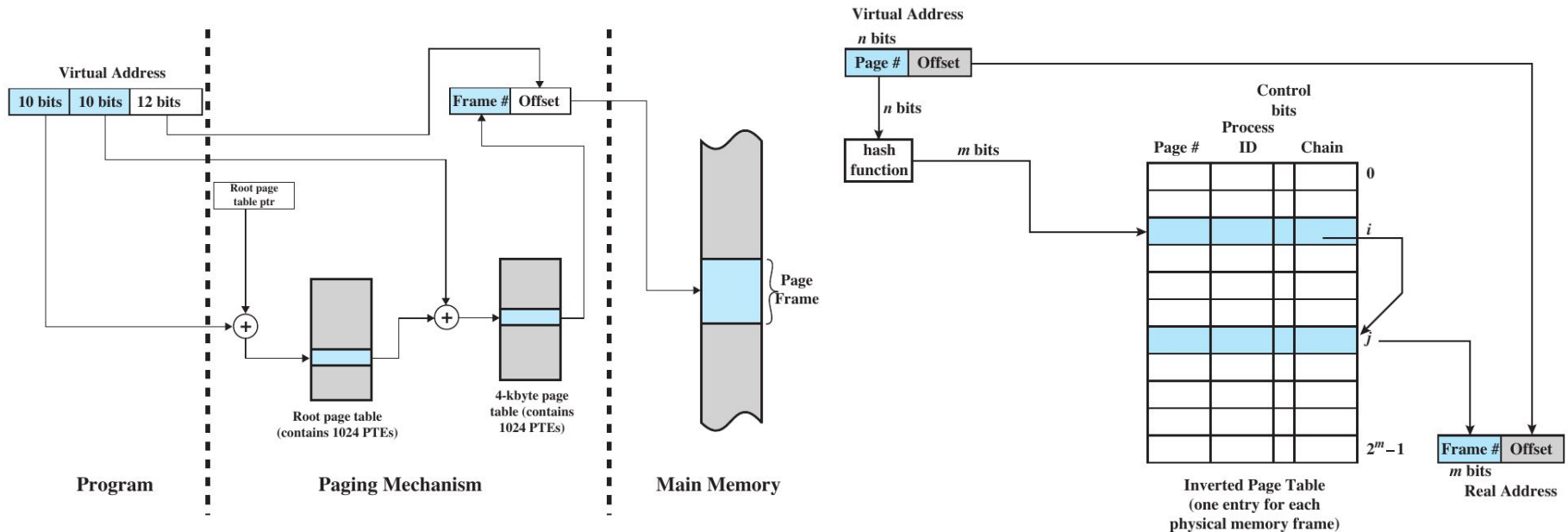
Opis wirtualnej przestrzeni adresowej dla procesora znajduje się w **tabeli stron**. Na podstawie adresu wirtualnego stron procesor wyznacza **wirtualny numer strony**, a następnie **przechodzi tablicę stron** szukając **wpisu strony**, z której wyciąga **numer ramki strony**. Mniej znaczące bity adresu wirtualnego nie podlegają translacji.

Wpisy stron przechowują dodatkowe informacje.



# Tablica stron

Tablice stron mogą być różnie zorganizowane: **wielopoziomowa** (Intel, ARM), **odwrócona** (PowerPC), **wirtualna płaska** (MIPS).  
Chcemy by opis przestrzeni adresowej był jak najefektywniejszy!



# Wpisy tablicy stron

Wpisy tablicy stron oprócz fizycznego numeru strony zawierają:

- **bity uprawnień** → czy procesor może odczytywać, zapisywać, wykonywać zawartość strony?
- **bit obecności** → czy strona ma przypisaną ramkę?
- **bit trybu pracy procesora** → czy procesor będąc w trybie użytkownika albo nadzorcy może korzystać ze strony?
- **bity polityki buforowania** → czy i jak procesor może przechowywać zawartość strony w pamięci podręcznej?  
(disable cache, write-through, copy-back, etc.)
- **bity monitorowania dostępu** → czy ostatnio procesor czytał tą stronę (**referenced**) albo modyfikował (**modified**)?  
(będą nam potrzebne do implementacji polityk zastępowania stron)

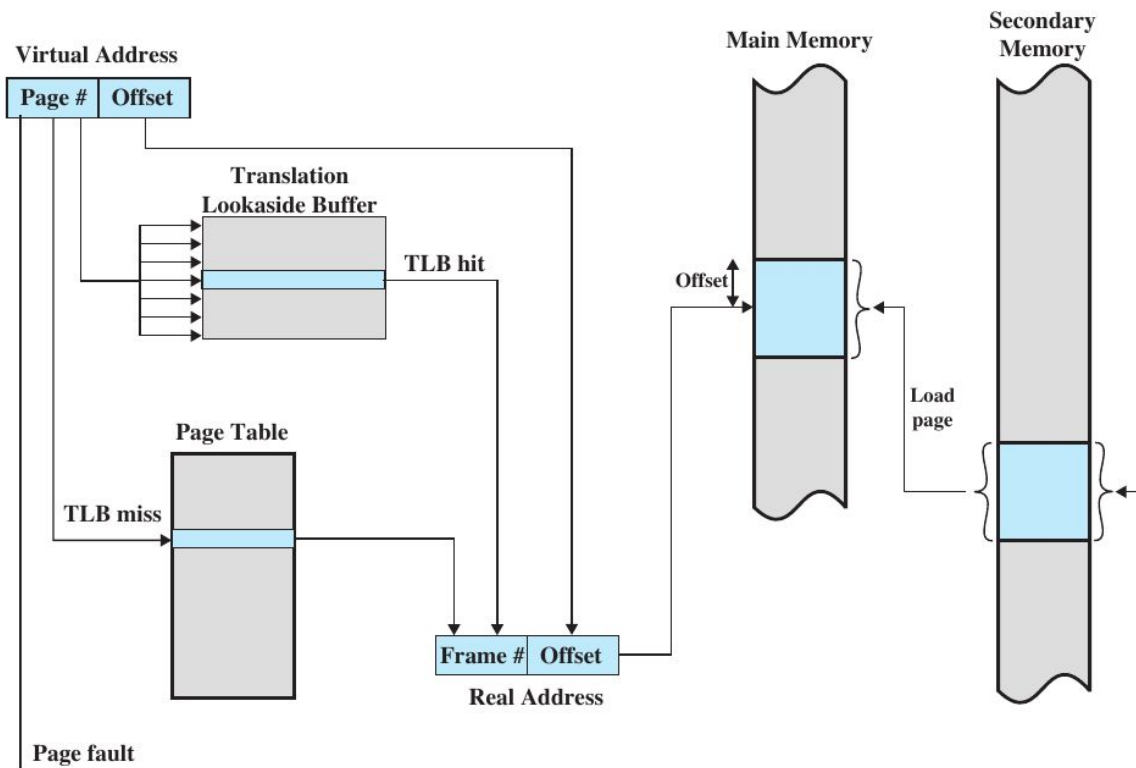
# Translation Lookaside Buffer

By translacja adresów przebiegała szybko procesory używają **TLB** – b. szybkiej pamięci podręcznej przechowującej wpisy stron.

## Chybiecie w TLB

wymaga zajrzenia do tablicy stron.

**Zasięg TLB** to rozmiar pamięci, którą możemy zaadresować bez generowania chybiecia.



Slajdy z CSAPP3e (18)



# Unix: wywołania systemowe (1)

<code>sysconf(_SC_PAGESIZE)</code>	Podaje rozmiar strony, to samo co <code>getpagesize()</code> .
<code>mmap(addr, length, prot, flags, fd, offset)</code>	Odwzoruj <b>length</b> bajtów zasobu <b>fd</b> od pozycji <b>offset</b> lub pamięci anonimowej pod <b>addr</b> . Uprawnienia obszaru to <b>prot</b> .
<code>munmap(addr, length)</code>	Usuń fragment lub całość odwzorowania.
<code>mprotect(addr, length, prot)</code>	Zmień uprawnienia wybranego obszaru.
<code>msync(addr, length, flags)</code>	Synchronizuj odwzorowanie z obiektem. <b>flags</b> ∈ {MS_ASYNC, MS_SYNC, MS_INVALIDATE}

- **length** i **offset** podzielne przez rozmiar strony
- **addr** jest wskazówką dla **mmap** chyba, że **MAP\_FIXED** ∈ **flags**
- **prot** złożeniem flag **PROT\_EXEC**, **PROT\_READ**, **PROT\_WRITE**
- mapowanie może być prywatne (**MAP\_SHARED**) lub dzielone (**MAP\_PRIVATE**)
- pamięć anonimową przydzielamy flagą **MAP\_ANONYMOUS**
- obszar może automatycznie rosnąć w dół **MAP\_STACK**
- **MS\_INVALIDATE** unieważnia zawartość pozostałych odwzorowań zasobu

## Unix: wywołania systemowe (2)

<code>mlock(addr, length)</code>	Przypnij strony do pamięci RAM.
<code>munlock(addr, length)</code>	Odepnij strony od pamięci RAM.
<code>mincore(addr, length, vec)</code>	Sprawdź czy strony są w <a href="#">rdzeniu</a> (pradawne określenie RAM).
<code>madvise(addr, length, advice)</code>	Poinformuj system o zamiarach w stosunku do obszaru. <b>flags</b> $\in \{\text{MADV\_NORMAL}, \text{MADV\_RANDOM}, \text{MADV\_SEQUENTIAL}, \text{MADV\_WILLNEED}, \text{MADV\_DONTNEED}\}$

- **vec** to adres miejsca na wektor bitów per strona
- przypięte strony → zabraniamy wymiany do pamięci drugorzędnej
- **MADV\_SEQUENTIAL** → strony będą sprowadzane sekwencyjne, możesz włączyć gorliwe sprowadzanie (ang. *read-ahead*)
- **MADV\_WILLNEED** → niedługo będę potrzebować, możesz asynchronicznie sprowadzić do pamięci RAM
- dużo flag *Linux-specific*, np. **MADV\_MERGEABLE** (przydatne dla monitorów maszyn wirtualnych)

# Zbiór roboczy i zbiór rezydentny

**Zbiór roboczy** to zbiór stron, których proces potrzebował w chwili  $t$  do działania w ciągu  $\Delta$  ostatnich tyknięć wirtualnego zegara.

**Zbiór rezydentny** to zbiór wszystkich stron procesu rezydujących w pamięci operacyjnej w chwili  $t$  wirtualnego zegara procesu.

System operacyjny zarządza zbiorem rezydentnym i stara się przybliżać nim zbiór roboczy  $\rightarrow$  chcemy by  $W(t, \Delta) \subseteq R(t)$  !

Znów będziemy polegać na **lokalności odwołań**, bo wiemy, że programy wykazują pewne wzorce w dostęпах do pamięci!

# Pomniejszy i poważny błąd strony

**pomniejszy błąd strony** (ang. *minor page fault*) → strona w pamięci operacyjnej, ale złe uprawnienia bądź nie podczepiona

**poważny błąd strony** (ang. *major page fault*) → strona w pamięci drugorzędnej, wymaga potencjalnie długiej operacji czytania, czas przełączyć się na inny proces

**częstość błędów stron** (ang. *page fault frequency*) → błędy stron można zliczać i analizować; można szybko stwierdzić czy rozmiar zbioru rezydentnego jest dopasowany do potrzeb procesu

```
ps -A -o pid,rss,vsz,min_flt,maj_flt,command
```

**PAMIĘĆ WIRTUALNA MÓWISZ?**



**... JA TO CHYBA JUŻ WIDZIAŁEM**

imgflip.com

# Zarządzanie cache'm vs. pamięcią wirtualną

Jest wiele podobieństw i słusznie! Realizujemy te same cele, ale między innymi **poziomami hierarchii pamięci** tylko, że tym razem robimy to programowo, a nie sprzętowo!

blok → strona

metadane bloku → wpis tablicy stron

pamięć asocjacyjna → tablica stron

zastępowanie bloków → polityka zastępowania stron

cache miss → major page fault

pojemność pamięci podręcznej → rozmiar pamięci RAM

pobranie bloku z RAM → pobranie strony z dysku

polityka write-back → buforowanie stron

# Polityka sprowadzania

**polityka sprowadzania** (ang. *fetch*) → uruchamiamy nowy program, najpierw musimy go wczytać... ale czy będziemy potrzebować całego jego kodu i wszystkich danych w pamięci operacyjnej od razu?

- leniwie → **stronicowanie na żądanie** (ang. *demand paging*)  
(błąd strony → ładujemy; a jeśli błędów jest dużo na krótkim odcinku czasu?)
- gorliwie → **stronicowanie wstępne** (ang. *prepaging*)  
(zauważamy, że sprowadzenie całej ścieżki z dysku lub jednego bloku nie różni się czasem → **czytanie z wyprzedzeniem** (ang. *read-ahead*))

Może dać programiście jakieś narzędzie do wyrażania intencji?

# Polityka sprzątanania

**polityka sprzątanania** (ang. *clean*) → możemy wyznaczyć podzbiór ramek stron, z których programy nie korzystają od dłuższego czasu, ale są zmodyfikowane i wymagają uspoźnienia z pamięcią drugorzędną. Kiedy je wyczyścić?

- leniwie → **czyszczenie na żądanie** (ang. *demand cleaning*)  
(jak musimy ponownie użyć danej strony to dopiero ją uspoźniamy)
- gorliwie → **czyszczenie wstępne** (ang. *precleaning*)  
(zbieramy strony w grupy i wyrzucamy na dysk, może już nie będą potrzebne)

Lepsze podejście? **Buforowanie stron**, o którym będzie później.



# Polityki przydziału

**polityka przydziału** (ang. *placement*) → sprowadzamy strony z pamięci drugorzędnej do RAM, czy to pod jakim adresem je umieścimy może mieć wpływ na wydajność programów? Tak!

- systemy z **niejednorodnym dostępem do pamięci** (ang. *Non-Uniform Memory Architecture*)  
(procesor ma “dalej” do pewnych fragmentów pamięci fizycznej)
- **kolorowanie stron** (ang. *page colouring*)  
(ułożenie danych w RAM może wpływać na liczbę chybień w pamięć cache)
- **duże strony** (ang. *huge pages*)  
(zwiększanie zasięgu TLB → mniej chybień w pamięciożernych programach)

# Polityka zastępowania

**polityka zastępowania** (ang. *replacement*) → chcemy zmniejszyć zbiór roboczy programu, potrzebujemy wyznaczyć ramki stron, których proces nie będzie już potrzebować. Będziemy wybierać strony ofiary (ang. *victim*), a następnie wyrzucać je lub uspójniać z pamięcią drugorzędną. Oczywiście nie znamy przyszłości (idealny algorytm **OPT**) więc będzie trzeba sobie radzić inaczej.

- możemy badać czy proces odnosił się do stron w przeszłości (wsparcie sprzętowe → bity monitorowania dostępu referenced / modified)
- ... i kojarzyć te informacje z wirtualnym czasomierzem procesu

... problem w tym, że będziemy popełniać błędy.

# Podstawowe polityki zastępowania: **OPT**, **LRU**, **FIFO**

Page address  
stream

2 3 2 1 5 2 4 5 3 2 5 2

**OPT**

2	2	2	2	2	2	4	4	4	2	2	2
	3	3	3	3	3	3	3	3	3	3	3
			1	5	5	5	5	5	5	5	5
				F		F			F		

**LRU**

2	2	2	2	2	2	2	2	3	3	3	3
	3	3	3	5	5	5	5	5	5	5	5
			1	1	1	4	4	4	2	2	2
				F		F		F	F		

**FIFO**

2	2	2	2	5	5	5	5	3	3	3	3
	3	3	3	3	2	2	2	2	2	5	5
			1	1	1	4	4	4	4	4	2
				F	F	F		F		F	F

F → błąd strony zapoczątkowujący wybór ofiary do zastąpienia

# Not Recently Used

Co każde tyknięcie wirtualnego zegara, np. 20ms, przeglądamy zbiór rezydentny procesu i na podstawie bitów monitorowania dostępu przypisujemy ramki do jednej z klas:

1. `referenced = 0, modified = 0` (ostatnio nieużywane)
2. `referenced = 1, modified = 0` (ostatnio używane)
3. `referenced = 0, modified = 1` (ostatnio nieużywane, brudne)
4. `referenced = 1, modified = 1` (ostatnio używane, brudne)

Szukamy ofiar do zastąpienia? Oczywiście nie chcemy stron z klasy 2 i 4. Najłatwiej wziąć strony z klasy 1. Strony z klasy 3 / 4 po **wyczyszczeniu** przechodzą do klasy 1 / 2.

# Algorytm drugiej szansy i buforowanie stron

Algorytm FIFO nie patrzy na bit **referenced** wyrzucanej strony. Mógł być ustawiony na 1 → więc nie chcemy jej usuwać. Dajmy jej drugą szansę → wrzucimy ją do **bufora stron** zamiast czyścić.

**demon stronicowania** (ang. *paging daemon*) wątek jądra (**kswapd** w Linuksie) przegląda regularnie bufory stron i:

- za mało wolnych ramek → zacznij usuwać strony
- strona długo brudna (10s?) i dysk się nudzi → uspójnij
- strona czysta i nie używana bardzo długo → zapomnij
- strona czysta i bez właściciela → zapomnij

**Q:** Czemu w buforze stron mogą być ramki nienależące do żadnego procesu?

**A:** Przyspieszenie ładowania często wykorzystywanych programów.

# Algorytm zegarowy (CLOCK)

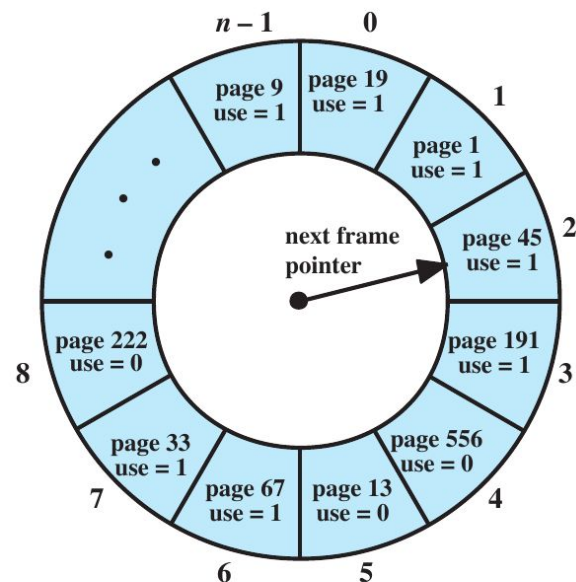
Ramię zegara wskazuje na stronę kandydata na **ofiara** do zastąpienia.

Występuje błąd strony:

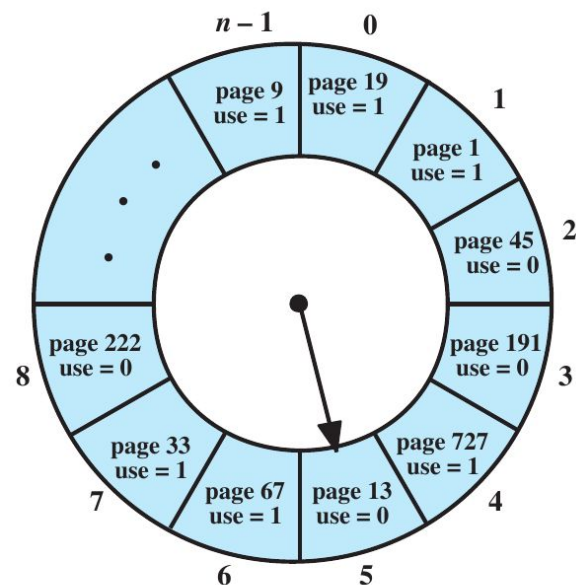
1. referenced=0 → usuwamy stronę
2. referenced=1 → wyczyść bit, idź do następnej strony, powtórz od (1)

Wszystkie strony miały R=1? Algorytm degeneruje się do FIFO po okrążeniu!

**Rozszerzenie:** Można dodać drugie ramię, które będzie wyznaczać strony do wyczyszczenia.



(a) State of buffer just prior to a page replacement



(b) State of buffer just after the next page replacement

# Najrzadziej używana strona

Żeby zrobić dokładnie **LRU** (ang. Least Recently Used) trzeba wsparcia ze strony sprzętu. Do wpisów tablicy stron dodajemy licznik  $C$  odwołań (32-bit?, więcej?). Każdy dostęp  $\rightarrow +1$ .

Obsługa błędu strony  $\rightarrow$  bierzemy strony zbioru rezydentnego, sortujemy po  $C$  i wybieramy najmniejszy.

**Wady:** Sprzęt musi aktualizować liczniki. Większa tablica stron.  $O(n \log n)$  na sortowanie przy każdym błędzie strony. Co jeśli liczniki się przepełniają?

Wsparcie sprzętowe nie istnieje. Koszt dokładnego algorytmu zbyt duży. Czy nie możemy jakoś przybliżać LRU?

# Postarzanie stron (ang. *ageing*)

	R bits for pages 0-5, clock tick 0	R bits for pages 0-5, clock tick 1	R bits for pages 0-5, clock tick 2	R bits for pages 0-5, clock tick 3	R bits for pages 0-5, clock tick 4
	1 0 1 0 1 1	1 1 0 0 1 0	1 1 0 1 0 1	1 0 0 0 1 0	0 1 1 0 0 0
Page					
0	10000000	11000000	11100000	11110000	01111000
1	00000000	10000000	11000000	01100000	10110000
2	10000000	01000000	00100000	00010000	10001000
3	00000000	00000000	10000000	01000000	00100000
4	10000000	11000000	01100000	10110000	01011000
5	10000000	01000000	10100000	01010000	00101000

Stronom zbioru rezydentnego dajemy rejestr przesuwany. Bity dostępu wsuwamy od lewej co cykl wirtualnego zegara. Usuwamy stronę o najmniejszej wartości.



# Modelowanie zbioru roboczego

Chcemy zastępować te strony zbioru rezydentnego, które nie są częścią zbioru roboczego. Sprzęt tego nie umożliwia!

**Q:** Jak wyznaczyć przybliżoną zawartość zbioru roboczego?

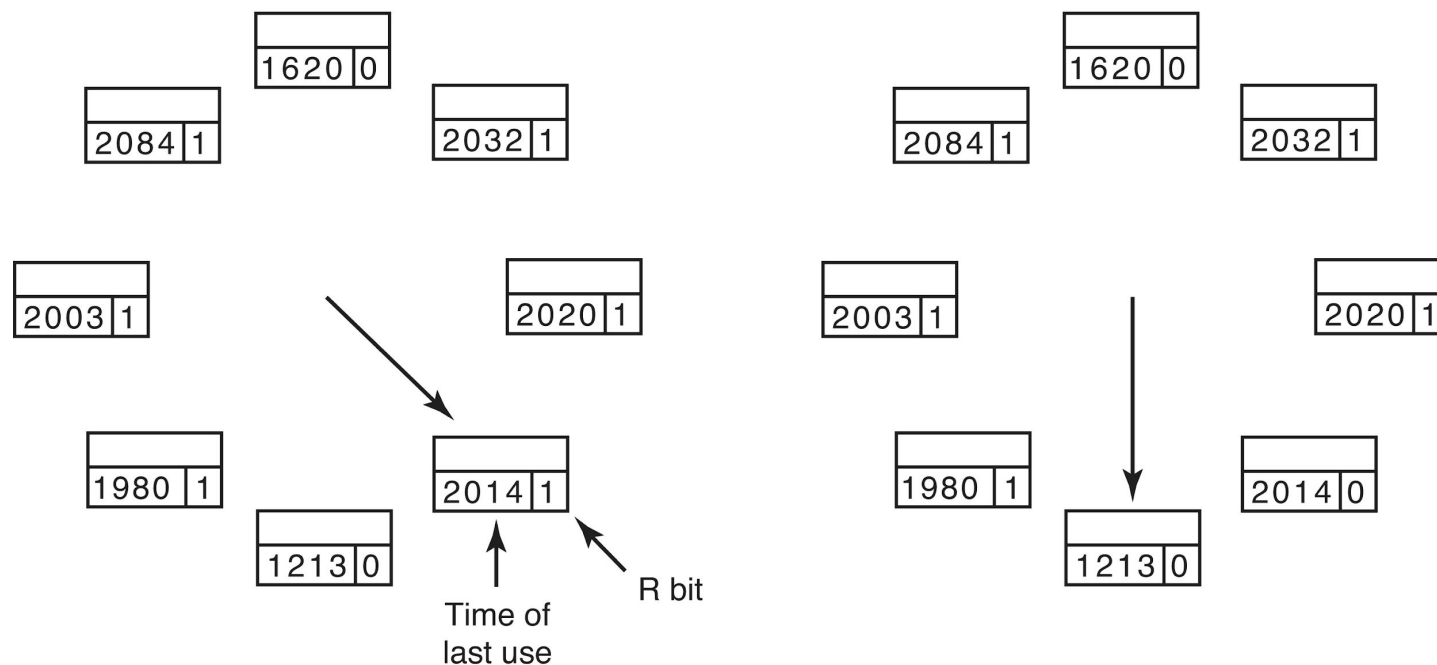
**A:** Niech zbiór roboczy to ramki używane w ciągu ostatnich  $\tau$  milisekund.

Planista będzie mierzył na ile taktów uruchomiliśmy proces zanim znów znalazł się w jądrze → **bieżący czas wirtualny procesu**.

Ze stronami zbioru rezydentnego kojarzymy **czas ostatniego użycia** (ang. *timestamp*). Co tyknięcie zegara przeglądamy strony:

- $\text{ref} = 0$  i  $\text{age} > \tau \rightarrow$  usuwamy stronę
- $\text{ref} = 1 \rightarrow \text{ref} \leftarrow 0$  i ustawiamy znacznik na obecny czas.

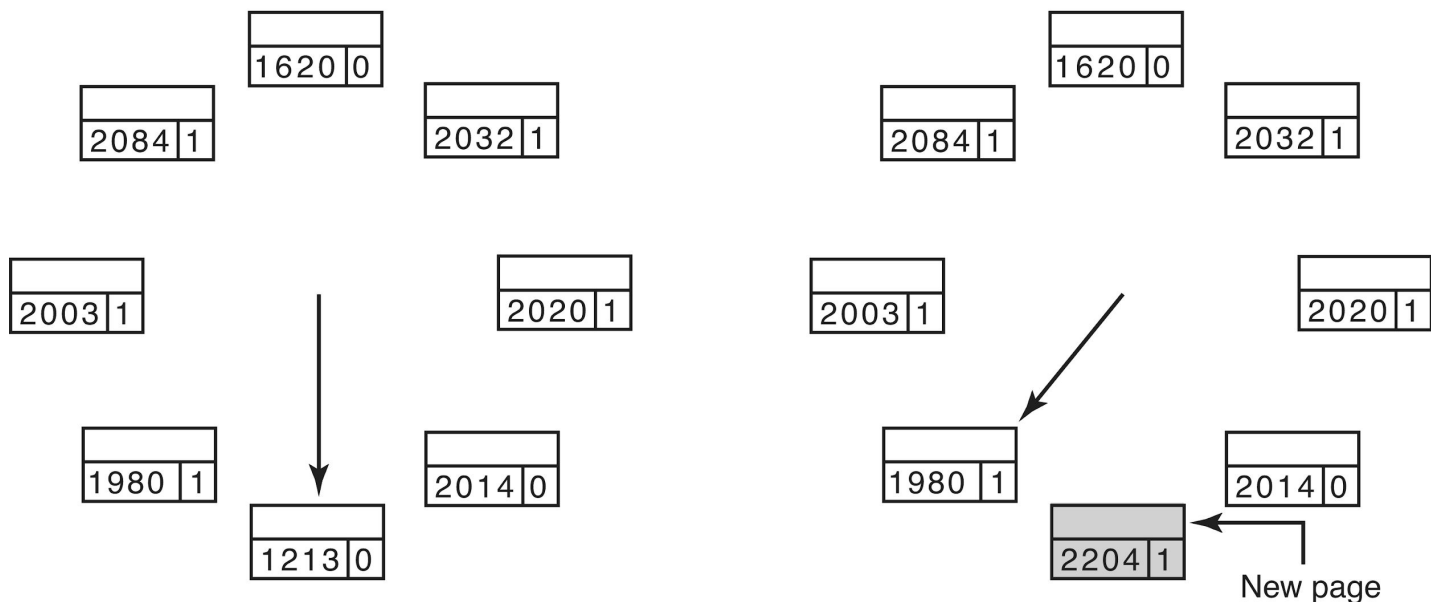
# WSClock (1)



Szybki i stosowany w praktyce algorytm modelujący zbiór roboczy.

Bieżący czas wirtualny 2204. Bit modified ukryto. Niech  $\tau = 250$ .

## WSClock (2)



Wskazana strona jest czysta i stara → usuwamy! Jeśli była brudna planujemy zapis na dysk i lecimy dalej – czy ograniczyć liczbę współbieżnych zapisów? Jeśli zbiór roboczy zbyt mały → należy go powiększyć. Możemy regulować wartość  $\tau$ !

# Zarządzanie rozmiarem zbioru rezydentnego

Czy procesowi przydzielić **stałą** (ang. *fixed-allocation*) czy **zmienną liczbę ramek** (ang. *variable-allocation*)?

- Procesy mają różne potrzeby, które można badać mierząc **PFF!**
- A gdyby tak chcieć sprawiedliwie traktować grupy procesów?
- Czy błędy stron wątków zliczać osobno czy na koszt procesu?

**zakres zastępowania stron** (ang. *replacement scope*) → rozpatrywać wybór ofiar osobno w każdym procesie czy globalnie?

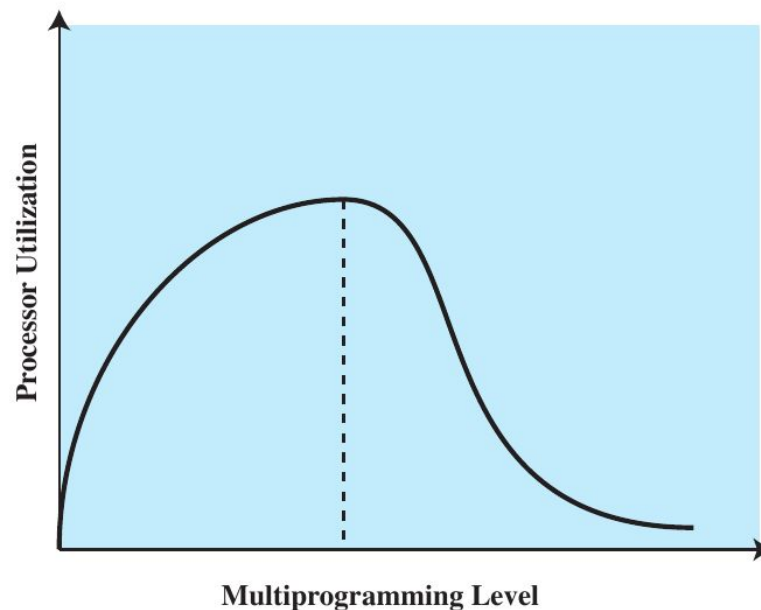
Zliczanie PFF trzeba zrobić mądrze... w trakcie życia procesu naturalnie istnieją pewne momenty, w który potrzebuje więcej stron.

# Zarządzanie obciążeniem i wieloprogramowość

**szamotanie** (ang. thrashing) → zbiory rezydentne wszystkich procesów są zbyt małe i system nie ma wolnych ramek, zatem spędza dużo czasu na **wymianie stron** (ang. swapping).

Jak wybrnąć z szamotania?

- wyrzucić procesy o niskim priorytecie z RAM
- zamrozić proces z najszybciej rosnącym zbiorem rezydentnym
- wybrać proces ofiarę i ją zabić ([Linux OOM killer](#))



# Przypinanie stron (*pinned / wired / locked pages*)

Po co zabraniać usuwania stron procesów z pamięci operacyjnej?

- chcemy zapewnić, że proces zawsze może się wykonywać  
(menadżer zadań do zabicia pamięciożernego procesu)
- proces musi szybko reagować na zdarzenia  
(system dostarcza pewnych gwarancji czasu rzeczywistego)

A co ze stronami jądra?

- nie możemy usunąć stron, na których leżą struktury danych  
(proces nie był używany od godziny → jego PCB musi być w RAM)
- transferujemy dane z użyciem DMA do dysku / karty sieciowej  
(urządzenia używają adresów fizycznych! nie możemy ani usunąć tych stron z RAM ani wykonywać ich relokacji → *locked pages*)

# Pamięć drugorzędna

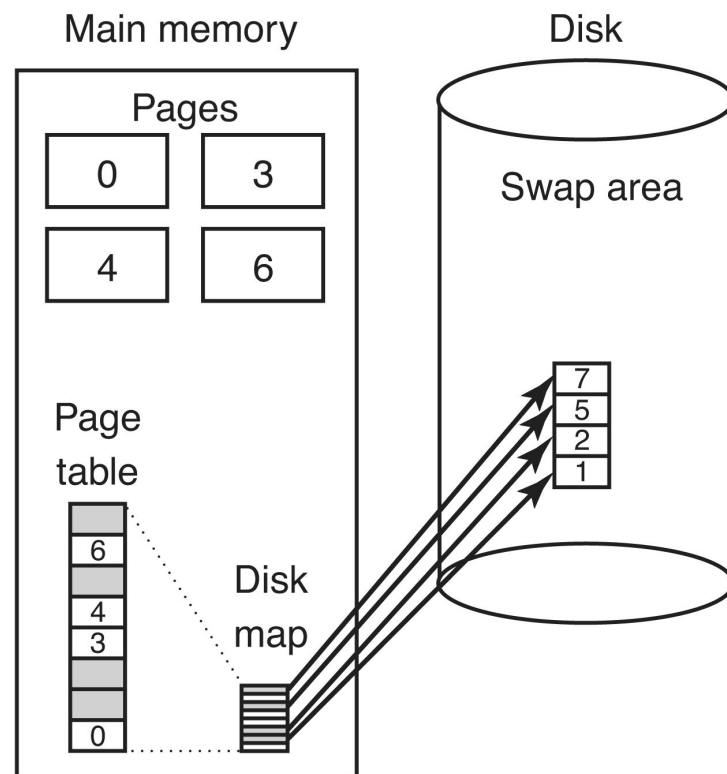
Strony zapisujemy w **obszarze wymiany** (ang. *swap area*).

**Ciągły plik** lub dowolne **urządzenie blokowe** → partycja dyskowa, dysk USB, pamięć karty graficznej, [zRAM](#)!

Służy do zachowania stanu systemu i procesów przy **hibernacji** na dysk!

**Q:** Co jeśli w pamięci były dane wrażliwe → hasła, klucze prywatne?

**A:** Obszar wymiany trzeba szyfrować!



Pytania?