Pamięć wirtualna

Pierwsze zastosowanie – głównie historyczne:

umożliwienie programom korzystania z większego obszaru pamięci operacyjnej niż rozmiar fizycznej pamięci (RAM) zainstalowanej w systemie.

Zasada działania jest podobna do pracy pamięci *cache*.

Zmienia się „skala” – w hierachii pamięci schodzimy poziom niżej:

- główna pamięć operacyjna zawiera fragment (np. kod jednego programu…) informacji przechowywanych w pamięci masowej (wszystkie zainstalowane w systemie programy),

- pamięć masowa (np. dysk magnetyczny, nieulotna pamięć półprzewodnikowa) ma jeszcze większą pojemność i jest jeszcze wolniejsza (w stosunku do operacyjnej),

- tak jak *cache* jest podzielony na bloki-linie, tak pamięć operacyjna-wirtualna **podzielona jest na strony** (***pages***) – o rozmiarze np. 4 kB (lub większym).

• Pamięci operacyjnej (fizycznej - RAM) jest mniej niż wirtualnej.

• Strony które „nie mieszczą się” w pamięci przechowywane są w **pliku wymiany** (***swap file***), przechowywanym w pamięci masowej.

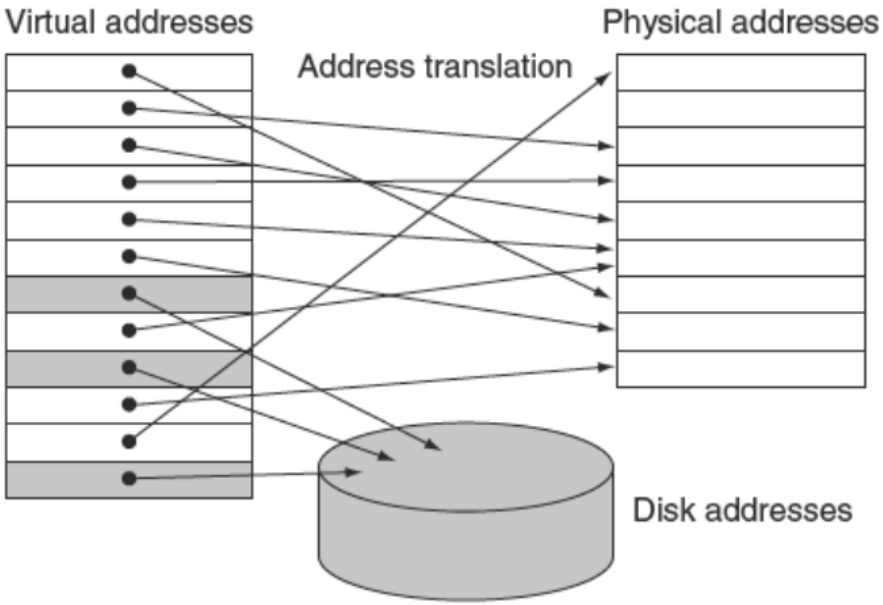
-> Procesor, operując adresami wirtualnymi, zgłasza konieczność dostępu do danych (przechowywanych w danej stronie pamięci).

• Adres wirtualny jest tłumaczony na adres fizyczny.

• Jeśli żądana strona znajduje się w pamięci operacyjnej – dane są z niej pobierane (i do *cache*, i do CPU…).

• Jeżeli danej strony nie ma w pamięci RAM – zgłaszany jest wyjątek: **błąd strony** (***page fault***) – wywoływana jest procedura obsługi wyjątku i dana strona ładowana jest do RAM z pliku wymiany.

swap file - the space on the disk reserved for the full virtual memory space of a program



Pamięć wirtualna

- Ze względu na konieczność ładowania brakujących stron z pamięci masowej i związany z tym długi czas dostępu (w przypadku mechanizmów „twardych” dysków magnetycznych rzędu kilku - kilkunastu milisekund) błędy strony obsługiwane są programowo – przez dedykowaną funkcję systemu operacyjnego.

- Narzut czasowy związany z programową obsługą jest niewielki w porównaniu z czasem dostępu do dysku. Rozwiązanie programowe pozwala również na łatwiejsze zastosowanie bardziej złożonych algorytmów wymiany stron niż implementacja sprzętowa (oraz ich ew. łatwą zmianę - aktualizację).

Możliwość powiększenia dostępnego dla programu rozmiaru pamięci poprzez umieszczenie jej części na dysku twardym była wykorzystywana **głównie dawniej**.

Rozmiary instalowanej pamięci były często niewystarczające do jednoczesnego przechowywania programu, danych i systemu operacyjnego (np. z powodów technologicznych, rzutujących na wysokie ceny pamięci).

Pamięć wirtualna

Zastosowanie obecne – w wielozadaniowych systemach operacyjnych

Mechanizm pozwalający na efektywne i bezpieczne **współdzielenie** przez wiele jednocześnie pracujących programów **jednej, wspólnej pamięci operacyjnej**:

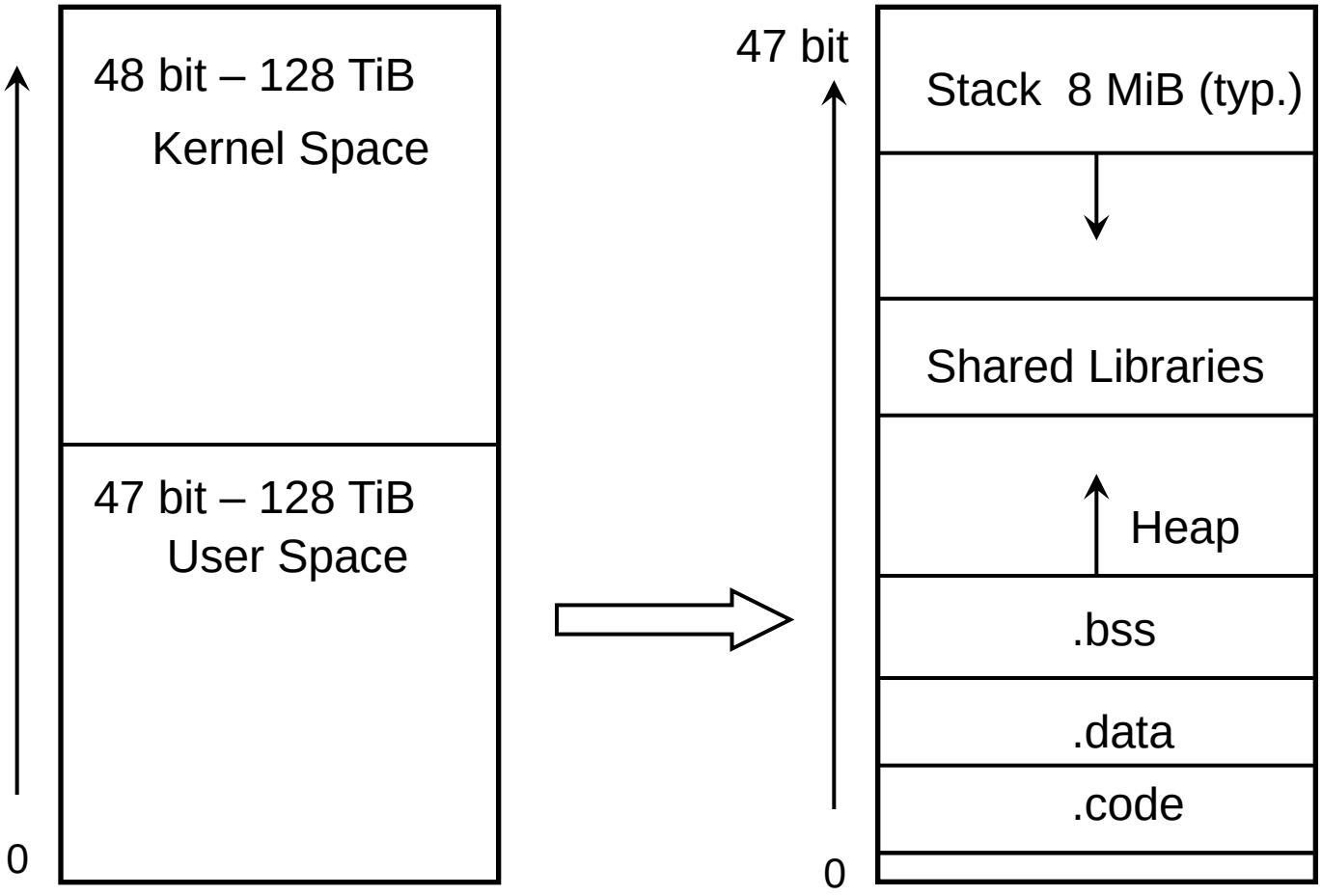
- każdy program/proces pracuje w swojej własnej wirtualnej przestrzeni adresowej (liniowej: adresowanej od adresu „0” do ustalonego maks.: np. 2^47 = 128 TiB),
- przestrzenie adresowe programów są izolowane od siebie. Typowy program użytkownika może odczytywać i zapisywać dane tylko w swoim, przydzielonym obszarze (ew. w kontrolowany sposób można utworzyć współdzielony między procesami fragment – „okno” pamięci).
- możliwa jest również implementacja **ochrony pamięci**: strony dostępne tylko dla systemu operacyjnego – niedostępne z poziomu użytkownika,
  strony tylko do odczytu itp.

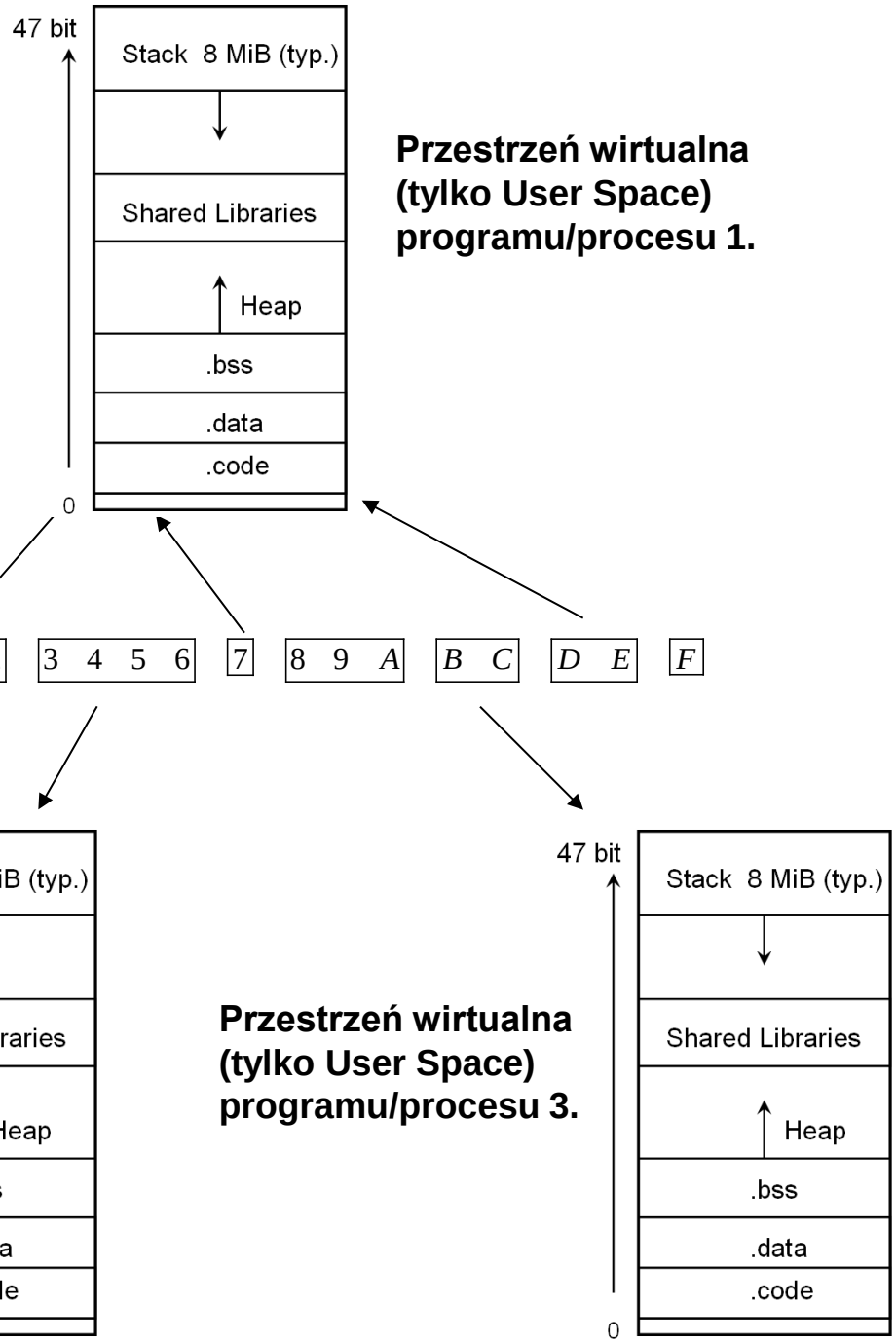
Pamięć wirtualna

W wielozadaniowych systemach operacyjnych rozmiar i miejsca zajętych i wolnych obszarów w pamięci zmieniają się w czasie: część programów / usług pracuje ciągle, inne programy rozpoczynają i kończą działanie w losowych momentach - odpowiednio alokując i zwalniając pamięć.

- mechanizm pamięci wirtualnej pozwala utworzyć **jedną, liniową wirtualną przestrzeń adresową** z **wielu nieużywanych obszarów, rozrzuconych po całej fizycznej przestrzeni adresowej**.

Pamięć wirtualna procesu - Linux 64 bit




47 bit
Stack 8 MiB (typ.)
Shared Libraries
Heap
.bss
.data
.code
0
Przestrzeń wirtualna (tylko User Space) programu/procesu 1.
Pamięć fizyczna
np. 16 GiB
0 1 2 3 4 5 6 7 8 9 A B C D E F
47 bit
Stack 8 MiB (typ.)
Shared Libraries
Heap
.bss
.data
.code
0
Przestrzeń wirtualna (tylko User Space) programu/procesu 2.
Przestrzeń wirtualna (tylko User Space) programu/procesu 3.
47 bit
Stack 8 MiB (typ.)
Shared Libraries
Heap
.bss
.data
.code
0

Pamięć wirtualna

Wirtualne przestrzenie pamięci, tworzone i przydzielane każdemu programowi w chwili jego uruchomienia, ułatwiają tworzenie programu: zarówno programiście (np. w asemblerze), jak i kompilatorowi, automatycznie tłumaczącemu wysokopoziomowy język na kod maszynowy.

Przestrzenie **wirtualne** są (pomijając wymagany rozmiar) „zawsze takie same”:

- adresowane „od zera”,

- w ramach danej sekcji (.text, .data, stos…) **nie zawierają nieciągłości**,

- proste programy* nie muszą być budowane jako „*position-independent*” - można używać stałych wirtualnych adresów absolutnych (numerów komórek pamięci – np. 32 bitowych**) i nie trzeba się martwić zmiennym położeniem sekcji kodu (.text) i danych statycznych (.data), relokacjami itp.

Przydzielone **fizyczne** adresy komórek pamięci mogą się zmieniać za każdym uruchomieniem tego samego programu.

* w przeciwieństwie do np. bibliotek dynamicznych. W przypadku „*randomizacji*” przestrzeni wirtualnej (Address Space Layout Randomization – ALSR) kompilatory domyślnie tworzą cały program jako *position-independent*.

** o ile przestrzeń 4 GiB na kod programu, ew. dane statyczne jest wystarczająca…

Pamięć wirtualna

- Zachodzi konieczność tłumaczenia adresu wirtualnego - jakim operuje CPU (w kodzie programu) - na fizyczny adres komórki (w układzie scalonym pamięci).

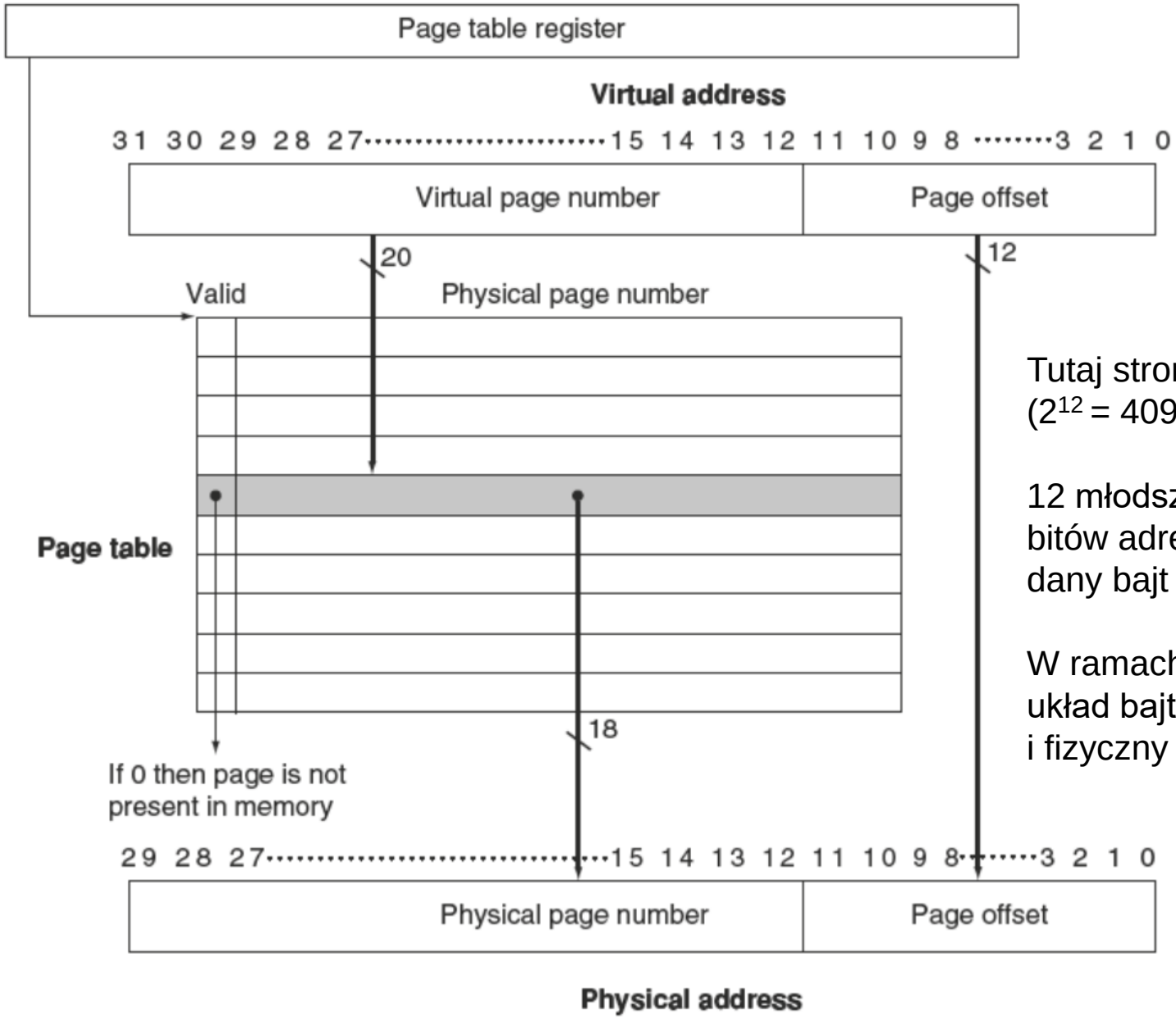
- Mechanizm tłumaczenia adresów wykorzystuje dedykowaną strukturę danych: **tablicę stron** (***page table***).

Na podstawie starszej części adresu wirtualnego wybierany jest odpowiedni element tablicy stron zawierający starszą część adresu fizycznego (oraz inne informacje m. in. czy strona znajduje się w pamięci RAM, czy w pliku wymiany, czy była modyfikowana).

W praktyce tłumaczone są tylko adresy stron (starsze części adresów – początków bloków pamięci, najczęściej – 4, 8, 16 KiB, ale mogą być również powyżej megabajta), a ramach danej strony porządek bajtów (wirtualny-fizyczny) jest taki sam,

(czyli układ fizyczny od wirtualnego różni się tylko kolejnością przechowywania stron w obu przestrzeniach adresowych).

Pamięć wirtualna



Tutaj strona ma 4 kB (2^{12} = 4096).

12 młodszych bitów adresuje dany bajt na stronie.

W ramach strony układ bajtów wirtualny i fizyczny jest taki sam.

Pamięć wirtualna

W tak przyjętym rozwiązaniu pojawia się problem:

-> każdy dostęp do pamięci operacyjnej wymaga tak naprawdę dwóch dostępów:

1. Do tablicy stron – w celu przetłumaczenia adresu.

2. Znając już adres fizyczny – do właściwych danych.

Aby temu zaradzić, podobnie jak w przypadku *cache*, wykorzystuje się lokalność:

przestrzenną:

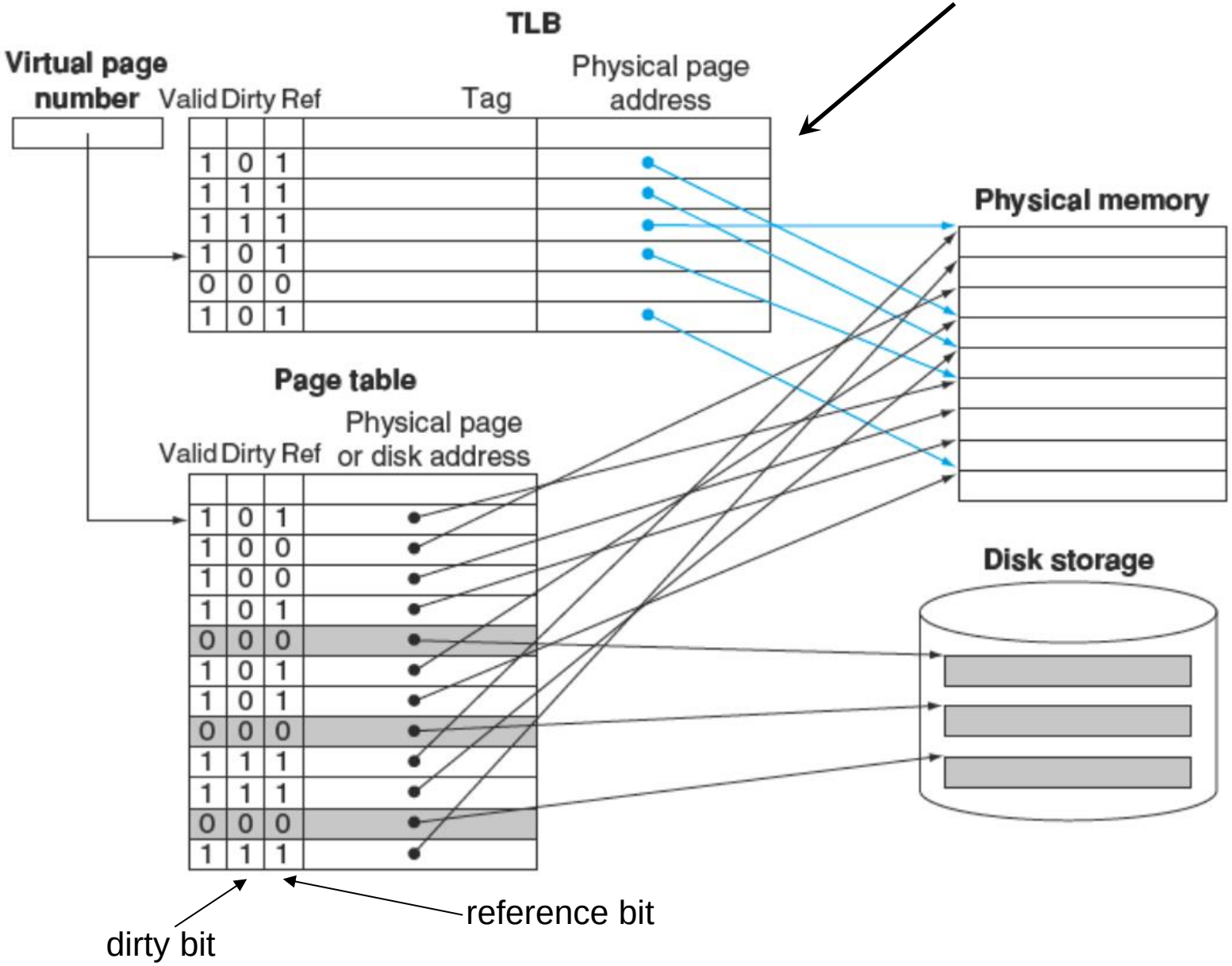
- dane są przenoszone (z pliku wymiany do RAM) stronami – blokami: (wymagane dane/instrukcje + sąsiedztwo),

oraz lokalność czasową:

- procesor posiada dodatkową pamięć *cache* – ***Translation Lookaside Buffer*** (TLB) zawierającą fragmenty tablicy stron, odpowiadające ostatnio tłumaczonym adresom.

Pamięć wirtualna

Translation-Lookaside Buffer
cache tablicy stron (w CPU)



Pamięć wirtualna

- Procesor żąda dostępu do pamięci (argumentem rozkazu jest adres wirtualny).

- W pierwszej kolejności informacja o adresie fizycznym poszukiwana jest w TLB:

- jeżeli jest obecna (trafienie – *TLB hit*) - adres wirtualny jest szybko tłumaczony na fizyczny (ponieważ TLB zawiera tylko ostatnio używaną część tablicy stron to jeśli wpis odpowiadający danej stronie jest w TLB, to taka strona już jest w pamięci RAM i nie nic trzeba ściągać z pliku wymiany – niebieskie linie na schemacie),

- jeżeli wpisu nie ma w TLB (*TLB miss* - chybienie) to trzeba sprawdzić:

  - czy nie ma informacji tylko w TLB, a strona w pamięci jednak jest (tylko *TLB miss*): pobierany jest wtedy brakujący wpis z tablicy stron (w miarę szybko),

  - czy w ogóle strony w pamięci nie ma (*page fault*) – system operacyjny ładuje stronę z pliku wymiany (oraz uaktualnia TLB).

Pamięć wirtualna

W każdym wpisie tablicy stron, jak i w TLB, znajdują się dodatkowe bity-flagi, ich zadanie jest podobne jak w przypadku pamięci *cache*:

Dirty bit – ustawiany kiedy zawartość danej strony została zmodyfikowana, (gdy dane „nie mieszczą się” w pamięci i jedna strona będzie zastąpiona inną, w przypadku zmodyfikowanej zawartości musi zostać wpierw zapisana w pliku wymiany).

Reference bit: informuje czy dana strona była w ostatnim czasie używana (tj. odbywał się do niej dostęp). Podobnie jak w przypadku pamięci cache, strony nieużywane od dłuższego czasu będą pierwszymi kandydatkami do zastąpienia nowymi - są wymieniane na zasadzie LRU.

Dodatkowe pola z strukturze tablicy stron może jeszcze wykorzystywać mechanizm ochrony pamięci: np. poziom uprzywilejowania (czy dana strona jest dostępna tylko dla systemu, czy również dla użytkownika, czy możliwy jest zapis, czy tylko odczyt itp.)

Algorytm zapisu TLB->tablica stron i RAM->plik wymiany **: tylko Write-Back !**

Należy pamiętać, że tak jak w przypadku cache i RAM, tak między TLB a tablicą stron musi być zachowana spójność danych…

Pamięć wirtualna

Podsumowanie:

- Mechanizm pamięci wirtualnej (i ochrony pamięci – o tym później) jest obsługiwany zarówno przez sprzęt (procesor i kontroler pamięci MMU Memory Management Unit) jak i oprogramowanie (system operacyjny).

- MMU i TLB są zintegrowane obecnie w jednej strukturze razem z procesorem.

- Tablica stron jest ulokowana w pamięci RAM.

- Każdy program / proces ma swoją tablicę stron – ponieważ ma własne mapowanie adresów wirtualnych na fizyczne.

- Ze względu na czas dostępu do pliku wymiany, błędy stron są obsługiwane przez funkcję systemu operacyjnego.

Pamięć wirtualna

Porównanie parametrów procesora ARM (urządzeń przenośnych) i typowego x86-64

Characteristic	ARM Cortex-A8	Intel Core i7
Virtual address	32 bits	48 bits
Physical address	32 bits	44 bits
Page size	Variable: 4, 16, 64 KiB, 1, 16 MiB	Variable: 4 KiB, 2/4 MiB
TLB organization	1 TLB for instructions and 1 TLB for data Both TLBs are fully associative, with 32 entries, round robin replacement TLB misses handled in hardware	1 TLB for instructions and 1 TLB for data per core Both L1 TLBs are four-way set associative, LRU replacement L1 I-TLB has 128 entries for small pages, 7 per thread for large pages L1 D-TLB has 64 entries for small pages, 32 for large pages The L2 TLB is four-way set associative, LRU replacement The L2 TLB has 512 entries TLB misses handled in hardware

Źródło: Patterson Hennessy: Computer Organisation and Design…

Pamięć wirtualna

Porównanie parametrów procesora ARM (urządzeń przenośnych) i typowego x86-64

Characteristic	ARM Cortex-A8	Intel Nehalem
L1 cache organization	Split instruction and data caches	Split instruction and data caches
L1 cache size	32 KiB each for instructions/data	32 KiB each for instructions/data per core
L1 cache associativity	4-way (I), 4-way (D) set associative	4-way (I), 8-way (D) set associative
L1 replacement	Random	Approximated LRU
L1 block size	64 bytes	64 bytes
L1 write policy	Write-back, Write-allocate(?)	Write-back, No-write-allocate
L1 hit time (load-use)	1 clock cycle	4 clock cycles, pipelined
L2 cache organization	Unified (instruction and data)	Unified (instruction and data) per core
L2 cache size	128 KiB to 1 MiB	256 KiB (0.25 MiB)
L2 cache associativity	8-way set associative	8-way set associative
L2 replacement	Random(?)	Approximated LRU
L2 block size	64 bytes	64 bytes
L2 write policy	Write-back, Write-allocate (?)	Write-back, Write-allocate
L2 hit time	11 clock cycles	10 clock cycles
L3 cache organization	–	Unified (instruction and data)
L3 cache size	–	8 MiB, shared
L3 cache associativity	–	16-way set associative
L3 replacement	–	Approximated LRU
L3 block size	–	64 bytes
L3 write policy	–	Write-back, Write-allocate
L3 hit time	–	35 clock cycles

Caches In the ARM Cortex-A8 and Intel Core I7 920.