

# Programowanie, dodatek

Rafał J. Wysocki

Instytut Fizyki Teoretycznej, Wydział Fizyki UW

28 maja 2012

## Wpływ sprzętu na szybkość obliczeń

Rzecz jasna szybkość działania programu komputerowego zależy od sprzętu, który go wykonuje.

## Wpływ sprzętu na szybkość obliczeń

Rzecz jasna szybkość działania programu komputerowego zależy od sprzętu, który go wykonuje.

Na przykład czas wykonywania programu (po skompilowaniu) jest w ogólności krótszy dla procesorów o wyższej częstotliwości zegara.

## Wpływ sprzętu na szybkość obliczeń

Rzecz jasna szybkość działania programu komputerowego zależy od sprzętu, który go wykonuje.

Na przykład czas wykonywania programu (po skompilowaniu) jest w ogólności krótszy dla procesorów o wyższej częstotliwości zegara.

Jednakże częstotliwość zegara procesora jest jednym z **wielu** parametrów wpływających na wydajność komputera (a przez to na szybkość obliczeń).

## Wpływ sprzętu na szybkość obliczeń

Rzecz jasna szybkość działania programu komputerowego zależy od sprzętu, który go wykonuje.

Na przykład czas wykonywania programu (po skompilowaniu) jest w ogólności krótszy dla procesorów o wyższej częstotliwości zegara.

Jednakże częstotliwość zegara procesora jest jednym z **wielu** parametrów wpływających na wydajność komputera (a przez to na szybkość obliczeń).

Co więcej, szczególne własności sprzętu mogą powodować, że algorytmy o (teoretycznie) jednakowej złożoności obliczeniowej będą wykonywane z różną szybkością.

## Przykład – mnożenie macierzy

Rozważmy dwie macierze kwadratowe  $\hat{A}$  i  $\hat{B}$  o wymiarze  $n$  oraz macierze  $\hat{C} = A \cdot B$  i  $\hat{D} = A \cdot B^T$ . Mamy

$$C_{ij} = \sum_{k=1}^n A_{ik} B_{kj}, \quad D_{ij} = \sum_{k=1}^n A_{ik} B_{jk}$$

## Przykład – mnożenie macierzy

Rozważmy dwie macierze kwadratowe  $\hat{A}$  i  $\hat{B}$  o wymiarze  $n$  oraz macierze  $\hat{C} = A \cdot B$  i  $\hat{D} = A \cdot B^T$ . Mamy

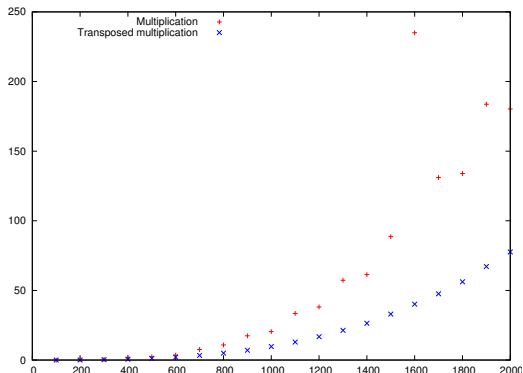
$$C_{ij} = \sum_{k=1}^n A_{ik} B_{kj}, \quad D_{ij} = \sum_{k=1}^n A_{ik} B_{jk}$$

Okazuje się, że obliczanie elementów macierzowych  $\hat{C}$  w pętli po lewej stronie zajmuje zwykle znacznie więcej czasu, niż obliczanie elementów macierzowych  $\hat{D}$  w pętli po prawej stronie:

```
for (i = 1; i <= n; i++)
  for (j = 1; j <= n; j++) {
    c[i][j] = 0;
    for (k = 1; k <= n; k++)
      c[i][j] += a[i][k] * b[k][j];
  }
```

```
for (i = 1; i <= n; i++)
  for (j = 1; j <= n; j++) {
    d[i][j] = 0;
    for (k = 1; k <= n; k++)
      d[i][j] += a[i][k] * b[j][k];
  }
```

# Wydajność kodu przy mnożeniu macierzy



**Rysunek:** Czas obliczeń w sekundach (oś pionowa) dla mnożenia macierzy (kolor czerwony) i mnożenia macierzy z transpozycją (kolor niebieski) w zależności od wymiaru macierzy (oś pozioma).



## Przykład – mnożenie macierzy (wnioski)

Obliczenia zajmują znacznie mniej czasu w przypadku, gdy w najbardziej wewnętrznej pętli indeks  $k$  przebiega elementy macierzowe w **jednym wierszu**  $\hat{B}$  (a nie w jednej kolumnie).

## Przykład – mnożenie macierzy (wnioski)

Obliczenia zajmują znacznie mniej czasu w przypadku, gdy w najbardziej wewnętrznej pętli indeks  $k$  przebiega elementy macierzowe w **jednym wierszu**  $\hat{B}$  (a nie w jednej kolumnie).

Ponadto czas trwania obliczeń dla elementów macierzowych  $\hat{D}$  jest znacznie bardziej przewidywalny, niż czas trwania obliczeń dla elementów macierzowych  $\hat{C}$ .

## Przykład – mnożenie macierzy (wnioski)

Obliczenia zajmują znacznie mniej czasu w przypadku, gdy w najbardziej wewnętrznej pętli indeks  $k$  przebiega elementy macierzowe w **jednym wierszu**  $\hat{B}$  (a nie w jednej kolumnie).

Ponadto czas trwania obliczeń dla elementów macierzowych  $\hat{D}$  jest znacznie bardziej przewidywalny, niż czas trwania obliczeń dla elementów macierzowych  $\hat{C}$ .

Zatem w celu obliczenia elementów macierzowych  $\hat{C}$  korzystne może być transponowanie macierzy  $\hat{B}$  przed przeprowadzeniem obliczeń i zastosowanie mnożenia z transpozycją zamiast „naiwnego” algorytmu.

## Przykład – mnożenie macierzy (wnioski)

Obliczenia zajmują znacznie mniej czasu w przypadku, gdy w najbardziej wewnętrznej pętli indeks  $k$  przebiega elementy macierzowe w **jednym wierszu**  $\hat{B}$  (a nie w jednej kolumnie).

Ponadto czas trwania obliczeń dla elementów macierzowych  $\hat{D}$  jest znacznie bardziej przewidywalny, niż czas trwania obliczeń dla elementów macierzowych  $\hat{C}$ .

Zatem w celu obliczenia elementów macierzowych  $\hat{C}$  korzystne może być transponowanie macierzy  $\hat{B}$  przed przeprowadzeniem obliczeń i zastosowanie mnożenia z transpozycją zamiast „naiwnego” algorytmu.

Aby wyjaśnić te obserwacje, trzeba wziąć pod uwagę konstrukcję współczesnych komputerów oraz ich sposób działania.

# Pamięć RAM

RAM (*ang. Random Access Memory*)

Pamięć o dostępie swobodnym pozwalająca na modyfikowanie zawartości dowolnej komórki pamięci (o pojemności 1 bitu) w dowolnym momencie.

# Pamięć RAM

RAM (*ang. Random Access Memory*)

Pamięć o dostępie swobodnym pozwalająca na modyfikowanie zawartości dowolnej komórki pamięci (o pojemności 1 bitu) w dowolnym momencie.

Jest to pamięć ulotna (*ang. volatile*)

Wymaga ciągłego zasilania do zachowania zawartości.

# Pamięć RAM

## RAM (*ang. Random Access Memory*)

Pamięć o dostępie swobodnym pozwalająca na modyfikowanie zawartości dowolnej komórki pamięci (o pojemności 1 bitu) w dowolnym momencie.

## Jest to pamięć ulotna (*ang. volatile*)

Wymaga ciągłego zasilania do zachowania zawartości.

## Statyczna pamięć RAM (*ang. static RAM*)

Jest zbudowana z tak zwanych **przerzutników** (*ang. flip-flop*), które z kolei składają się z **tranzystorów** i nie wymaga **odświeżania** (*ang. refresh*).

# Pamięć RAM

## RAM (*ang. Random Access Memory*)

Pamięć o dostępie swobodnym pozwalająca na modyfikowanie zawartości dowolnej komórki pamięci (o pojemności 1 bitu) w dowolnym momencie.

## Jest to pamięć ulotna (*ang. volatile*)

Wymaga ciągłego zasilania do zachowania zawartości.

## Statyczna pamięć RAM (*ang. static RAM*)

Jest zbudowana z tak zwanych **przerzutników** (*ang. flip-flop*), które z kolei składają się z **tranzystorów** i nie wymaga **odświeżania** (*ang. refresh*).

Statyczna pamięć RAM charakteryzuje się krótkim **czasem dostępu** (*ang. access time*), czyli czasem potrzebnym do przeprowadzenia operacji zapisu bądź odczytu danych.



# Dynamiczna pamięć RAM

## DRAM (*ang. Dynamic RAM*)

Jest zbudowana z tranzystorów i **kondensatorów** (1 komórka składa się z 1 kondensatora i 1 tranzystora). Kondensatory służą do przechowywania bitów danych, w związku z czym taka pamięć **wymaga** odświeżania (kondensatory z czasem tracą ładunek).

# Dynamiczna pamięć RAM

## DRAM (*ang. Dynamic RAM*)

Jest zbudowana z tranzystorów i **kondensatorów** (1 komórka składa się z 1 kondensatora i 1 tranzystora). Kondensatory służą do przechowywania bitów danych, w związku z czym taka pamięć **wymaga** odświeżania (kondensatory z czasem tracą ładunek).

Odświeżanie polega na okresowym **odczytywaniu danych** zapisanych w pamięci.

# Dynamiczna pamięć RAM

## DRAM (*ang. Dynamic RAM*)

Jest zbudowana z tranzystorów i **kondensatorów** (1 komórka składa się z 1 kondensatora i 1 tranzystora). Kondensatory służą do przechowywania bitów danych, w związku z czym taka pamięć **wymaga** odświeżania (kondensatory z czasem tracą ładunek).

Odświeżanie polega na okresowym **odczytywaniu danych** zapisanych w pamięci.

## Jak to działa?

W celu odczytania zawartości komórki pamięci trzeba zmierzyć ładunek na kondensatorze, więc trzeba pozwolić mu rozładować się poprzez układ pomiarowy (wzmacniający sygnał). Później (jeśli trzeba) kondensator jest ponownie ładowany na zasadzie **sprzężenia zwrotnego** (*ang. feedback*).

# Zalety dynamicznych pamięci RAM

## Zalety DRAM w stosunku do pamięci statycznych

- 1 Znacznie mniejsze rozmiary komórek.
- 2 Znacznie mniejsza liczba tranzystorów na komórkę (typowo 1:6).
- 3 Bardziej regularna struktura komórki (możliwe ciaśniejsze upakowanie).
- 4 Brak konieczności doprowadzania napięcia zasilającego do każdej komórki z osobna.

# Zalety dynamicznych pamięci RAM

## Zalety DRAM w stosunku do pamięci statycznych

- 1 Znacznie mniejsze rozmiary komórek.
- 2 Znacznie mniejsza liczba tranzystorów na komórkę (typowo 1:6).
- 3 Bardziej regularna struktura komórki (możliwe ciaśniejsze upakowanie).
- 4 Brak konieczności doprowadzania napięcia zasilającego do każdej komórki z osobna.

Wymienione cechy pamięci DRAM powodują, że ten rodzaj pamięci jest **dramatycznie** tańszy w produkcji, zwłaszcza dla dużych pojemności (im większa pojemność, tym pamięci DRAM są tańsze).

# Zalety dynamicznych pamięci RAM

## Zalety DRAM w stosunku do pamięci statycznych

- 1 Znacznie mniejsze rozmiary komórek.
- 2 Znacznie mniejsza liczba tranzystorów na komórkę (typowo 1:6).
- 3 Bardziej regularna struktura komórki (możliwe ciaśniejsze upakowanie).
- 4 Brak konieczności doprowadzania napięcia zasilającego do każdej komórki z osobna.

Wymienione cechy pamięci DRAM powodują, że ten rodzaj pamięci jest **dramatycznie** tańszy w produkcji, zwłaszcza dla dużych pojemności (im większa pojemność, tym pamięci DRAM są tańsze).

Dlatego w komputerach używa się głównie pamięci DRAM.

# Wady dynamicznych pamięci RAM

## Niestety technologia DRAM ma poważne wady

- 1 Znaczna upływność kondensatorów dla dużych pojemności pamięci (trzeba często odświeżać, typowo co 64 *ms*).
- 2 Konieczność wzmacniania sygnału przy odczycie (prąd pochodzący z rozładowania kondensatora w komórce pamięci jest bardzo słaby).
- 3 Konieczność ponownego ładowania kondensatora po odczycie (zwiększa zużycie energii i zabiera czas).
- 4 Ładowanie i rozładowywanie kondensatora nie jest natychmiastowe (przy odczycie dostatecznie wyraźny sygnał pojawia się z opóźnieniem).

# Wady dynamicznych pamięci RAM

## Niestety technologia DRAM ma poważne wady

- 1 Znaczna upływność kondensatorów dla dużych pojemności pamięci (trzeba często odświeżać, typowo co  $64\text{ ms}$ ).
- 2 Konieczność wzmacniania sygnału przy odczycie (prąd pochodzący z rozładowania kondensatora w komórce pamięci jest bardzo słaby).
- 3 Konieczność ponownego ładowania kondensatora po odczycie (zwiększa zużycie energii i zabiera czas).
- 4 Ładowanie i rozładowywanie kondensatora nie jest natychmiastowe (przy odczycie dostatecznie wyraźny sygnał pojawia się z opóźnieniem).

Powodują one, że operowanie pamięciami DRAM jest skomplikowane oraz istnieje **fizyczne** ograniczenie czasu dostępu dla nich, którego **nie da się** wyeliminować.



# Organizacja pamięci DRAM

Komórki pamięci DRAM są rozmieszczane w prostokątnych **matrycach** (*ang. array*). Inny sposób uporządkowania ich byłby zbyt kosztowny.

# Organizacja pamięci DRAM

Komórki pamięci DRAM są rozmieszczane w prostokątnych **matrycach** (*ang. array*). Inny sposób uporządkowania ich byłby zbyt kosztowny.

Przy odwołaniach do pamięci adres (fizyczny) lokacji jest dzielony na dwie części, z których jedna,  $\overline{RAS}$  (*ang. Row Address Selection*), wybiera **wiersz** (*ang. row*) matrycy, a druga,  $\overline{CAS}$  (*ang. Column Access Selection*), określa **kolumnę** (*ang. column*), z której mają być pobrane dane.

# Organizacja pamięci DRAM

Komórki pamięci DRAM są rozmieszczane w prostokątnych **matrycach** (*ang. array*). Inny sposób uporządkowania ich byłby zbyt kosztowny.

Przy odwołaniach do pamięci adres (fizyczny) lokacji jest dzielony na dwie części, z których jedna,  $\overline{RAS}$  (*ang. Row Address Selection*), wybiera **wiersz** (*ang. row*) matrycy, a druga,  $\overline{CAS}$  (*ang. Column Access Selection*), określa **kolumnę** (*ang. column*), z której mają być pobrane dane.

$\overline{RAS}$  jest podawany na demultiplekser, który ma tyle wyjść, ile jest wierszy w matrycy. Generuje on sygnał „aktywujący” wszystkie komórki pamięci w danym wierszu.

## Odczyt z pamięci DRAM

Wszystkie naładowane kondensatory komórek pamięci w wierszu matrycy „aktywowanym” przez  $\overline{RAS}$  rozładowują się i pochodzące z nich sygnały (po wzmacnieniu) są podawane na multiplekser razem z  $\overline{CAS}$ .

## Odczyt z pamięci DRAM

Wszystkie naładowane kondensatory komórek pamięci w wierszu matrycy „aktywowanym” przez  $\overline{RAS}$  rozładowują się i pochodzące z nich sygnały (po wzmacnieniu) są podawane na multiplekser razem z  $\overline{CAS}$ .

Multiplekser „wypuszcza” sygnał z jednej kolumny jako odczytany bit danych. Kombinacja bitów  $\overline{CAS}$  decyduje o tym, z której kolumny pochodzi ten sygnał.

## Odczyt z pamięci DRAM

Wszystkie naładowane kondensatory komórek pamięci w wierszu matrycy „aktywowanym” przez  $\overline{RAS}$  rozładowują się i pochodzące z nich sygnały (po wzmacnieniu) są podawane na multiplekser razem z  $\overline{CAS}$ .

Multiplekser „wypuszcza” sygnał z jednej kolumny jako odczytany bit danych. Kombinacja bitów  $\overline{CAS}$  decyduje o tym, z której kolumny pochodzi ten sygnał.

Aby uzyskać liczbę bitów odpowiadającą szerokości magistrali pamięci (typowo 64), należy równolegle przeprowadzać operacje odczytu na odpowiedniej liczbie matryc DRAM.

## Zapis do pamięci DRAM

Wszystkie naładowane kondensatory komórek pamięci w wierszu matrycy „aktywowanym” przez  $\overline{RAS}$  rozładowują się.

## Zapis do pamięci DRAM

Wszystkie naładowane kondensatory komórek pamięci w wierszu matrycy „aktywowanym” przez  $\overline{RAS}$  rozładowują się.

$\overline{CAS}$  oraz bit danych do zapisania są podawane na demultiplekser, który (jeśli bit do zapisania odpowiada niezerowemu sygnałowi) generuje sygnał na linii odpowiadającej wybranej kolumnie. W komórce pamięci w tej kolumnie zapisywana jest wartość reprezentowana przez wejściowy bit.



## Zapis do pamięci DRAM

Wszystkie naładowane kondensatory komórek pamięci w wierszu matrycy „aktywowanym” przez  $\overline{RAS}$  rozładowują się.

$\overline{CAS}$  oraz bit danych do zapisania są podawane na demultiplekser, który (jeśli bit do zapisania odpowiada niezerowemu sygnałowi) generuje sygnał na linii odpowiadającej wybranej kolumnie. W komórce pamięci w tej kolumnie zapisywana jest wartość reprezentowana przez wejściowy bit.

W pozostałych komórkach w wierszu wybranym przez  $\overline{RAS}$  zapisywana jest ich poprzednia zawartość.

## Przesyłanie $\overline{RAS}$ i $\overline{CAS}$

Każda dodatkowa linia adresowa w magistrali pamięci zwiększa koszt produkcji komputera.

## Przesyłanie $\overline{RAS}$ i $\overline{CAS}$

Każda dodatkowa linia adresowa w magistrali pamięci zwiększa koszt produkcji komputera.

W związku z tym, dla ograniczenia liczby potrzebnych linii adresowych do niezbędnego minimum, słowa  $\overline{RAS}$  i  $\overline{CAS}$  są przesyłane oddzielnie **tymi samymi** liniami adresowymi.

## Przesyłanie $\overline{RAS}$ i $\overline{CAS}$

Każda dodatkowa linia adresowa w magistrali pamięci zwiększa koszt produkcji komputera.

W związku z tym, dla ograniczenia liczby potrzebnych linii adresowych do niezbędnego minimum, słowa  $\overline{RAS}$  i  $\overline{CAS}$  są przesyłane oddzielnie **tymi samymi** liniami adresowymi.

Odstęp czasu od wysłania  $\overline{RAS}$  do wysłania  $\overline{CAS}$  (*ang.  $\overline{RAS}$ -to- $\overline{CAS}$  delay*) musi być ściśle dostosowany do możliwości matrycy pamięci w modułach DRAM.

## Przesyłanie $\overline{RAS}$ i $\overline{CAS}$

Każda dodatkowa linia adresowa w magistrali pamięci zwiększa koszt produkcji komputera.

W związku z tym, dla ograniczenia liczby potrzebnych linii adresowych do niezbędnego minimum, słowa  $\overline{RAS}$  i  $\overline{CAS}$  są przesyłane oddzielnie **tymi samymi** liniami adresowymi.

Odstęp czasu od wysłania  $\overline{RAS}$  do wysłania  $\overline{CAS}$  (ang.  *$\overline{RAS}$ -to- $\overline{CAS}$  delay*) musi być ściśle dostosowany do możliwości matrycy pamięci w modułach DRAM.

Zapisanie (lub odczytanie) większej liczby bajtów wymaga przeprowadzenia ciągu takich operacji.

# Protokół dostępu do pamięci DRAM

## Synchroniczne pamięci DRAM (SDRAM)

Wszystkie operacje z udziałem magistrali pamięci są przeprowadzane zgodnie z dodatkowym sygnałem **zegarowym** (*ang. clock*).

# Protokół dostępu do pamięci DRAM

## Synchroniczne pamięci DRAM (SDRAM)

Wszystkie operacje z udziałem magistrali pamięci są przeprowadzane zgodnie z dodatkowym sygnałem zegarowym (*ang. clock*).

W najprostszej wersji protokołu wszystkie operacje zaczynają się przy narastającym zboczu (*ang. rising edge*) sygnału zegarowego.

# Protokół dostępu do pamięci DRAM

## Synchroniczne pamięci DRAM (SDRAM)

Wszystkie operacje z udziałem magistrali pamięci są przeprowadzane zgodnie z dodatkowym sygnałem zegarowym (*ang. clock*).

W najprostszej wersji protokołu wszystkie operacje zaczynają się przy narastającym zboczcu (*ang. rising edge*) sygnału zegarowego.

## Procedura odczytu

- 1 Numer wiersza na liniach adresowych, aktywny sygnał  $\overline{RAS}$ .
- 2 Odstęp czasu  $t_{RCD}$ .
- 3 Numer kolumny na liniach adresowych, aktywny sygnał  $\overline{CAS}$ .
- 4 Odstęp czasu  $t_{CL}$  (*ang.  $\overline{CAS}$  Latency*).
- 5 Dane na liniach danych.



## Protokół dostępu do pamięci DRAM (c. d.)

Po przesłaniu danych potrzebny jest czas na przygotowanie następnego numeru wiersza do zaadresowania,  $t_{RP}$  (*ang. Row Precharge*).

## Protokół dostępu do pamięci DRAM (c. d.)

Po przesłaniu danych potrzebny jest czas na przygotowanie następnego numeru wiersza do zaadresowania,  $t_{RP}$  (ang. *Row Precharge*).

Dodatkowo, po aktywowaniu sygnału  $\overline{RAS}$  musi upłynąć pewien czas, po którym może on być ponownie aktywowany,  $t_{RAS}$  (ang.  *$\overline{RAS}$  active to precharge delay*).

## Protokół dostępu do pamięci DRAM (c. d.)

Po przesłaniu danych potrzebny jest czas na przygotowanie następnego numeru wiersza do zaadresowania,  $t_{RP}$  (ang. *Row Precharge*).

Dodatkowo, po aktywowaniu sygnału  $\overline{RAS}$  musi upłynąć pewien czas, po którym może on być ponownie aktywowany,  $t_{RAS}$  (ang.  *$\overline{RAS}$  active to precharge delay*).

W niektórych przypadkach  $t_{RAS}$  może być dłuższy od sumy  $t_{RCD}$ ,  $t_{CL}$  oraz czasu przesyłania adresów i danych.

## Protokół dostępu do pamięci DRAM (c. d.)

Po przesłaniu danych potrzebny jest czas na przygotowanie następnego numeru wiersza do zaadresowania,  $t_{RP}$  (*ang. Row Precharge*).

Dodatkowo, po aktywowaniu sygnału  $\overline{RAS}$  musi upłynąć pewien czas, po którym może on być ponownie aktywowany,  $t_{RAS}$  (*ang.  $\overline{RAS}$  active to precharge delay*).

W niektórych przypadkach  $t_{RAS}$  może być dłuższy od sumy  $t_{RCD}$ ,  $t_{CL}$  oraz czasu przesyłania adresów i danych.

Dlatego, jeżeli w następnej operacji dane mają być pobierane z tego samego wiersza, jego numer jest „zatrzaskiwany” (*ang. latch*) przez kontroler pamięci i opóźnienia  $t_{RAS}$  oraz  $t_{RP}$  można pominąć.

## Protokół dostępu do pamięci DRAM (c. d.)

Sygnal  $\overline{CAS}$  może być wielokrotnie aktywowany przy „zatrzaśniętym” numerze wiersza, jednak nie można tego robić zbyt często.

## Protokół dostępu do pamięci DRAM (c. d.)

Sygnal  $\overline{CAS}$  może być wielokrotnie aktywowany przy „zatrzaśniętym” numerze wiersza, jednak nie można tego robić zbyt często.

Decyduje o tym parametr zwany **szybkością przetwarzania poleceń** (ang. *Command Rate*) modułów pamięci.

## Protokół dostępu do pamięci DRAM (c. d.)

Sygnał  $\overline{\text{CAS}}$  może być wielokrotnie aktywowany przy „zatrzaśniętym” numerze wiersza, jednak nie można tego robić zbyt często.

Decyduje o tym parametr zwany **szybkością przetwarzania poleceń** (ang. *Command Rate*) modułów pamięci.

Dodatkowo, jeśli kolejne dostępy dotyczą danych z tego samego wiersza i wielu kolejnych kolumn, mogą one być przesłane w ciągu **jednej operacji**, w tak zwanym trybie **nagłym** (ang. *burst*).

## Protokół dostępu do pamięci DRAM (c. d.)

Sygnał  $\overline{CAS}$  może być wielokrotnie aktywowany przy „zatrzaśniętym” numerze wiersza, jednak nie można tego robić zbyt często.

Decyduje o tym parametr zwany **szybkością przetwarzania poleceń** (*ang. Command Rate*) modułów pamięci.

Dodatkowo, jeśli kolejne dostępy dotyczą danych z tego samego wiersza i wielu kolejnych kolumn, mogą one być przesłane w ciągu **jednej operacji**, w tak zwanym trybie **nagłym** (*ang. burst*).

Dlatego z punktu widzenia wydajności najlepiej jest programować dostępy do pamięci tak, aby dotyczyły one jak najczęściej danych przechowywanych **w tych samych wierszach** matryc DRAM.



# Wpływ odświeżania pamięci DRAM

Odświeżanie pamięci DRAM jest przeprowadzane wiersz po wierszu.

## Wpływ odświeżania pamięci DRAM

Odświeżanie pamięci DRAM jest przeprowadzane wiersz po wierszu.

W trakcie odświeżania jednego wiersza matryca pamięci nie jest dostępna dla użytecznego odczytu i zapisu danych.

## Wpływ odświeżania pamięci DRAM

Odświeżanie pamięci DRAM jest przeprowadzane wiersz po wierszu.

W trakcie odświeżania jednego wiersza matryca pamięci nie jest dostępna dla użytecznego odczytu i zapisu danych.

Zgodnie z obowiązującymi standardami każda komórka DRAM powinna być odświeżana co 64 *ms*. Oznacza to, że w praktyce polecenie odświeżenia wiersza jest wysyłane do matrycy DRAM co około 7,8  $\mu$ s.

## Wpływ odświeżania pamięci DRAM

Odświeżanie pamięci DRAM jest przeprowadzane wiersz po wierszu.

W trakcie odświeżania jednego wiersza matryca pamięci nie jest dostępna dla użytecznego odczytu i zapisu danych.

Zgodnie z obowiązującymi standardami każda komórka DRAM powinna być odświeżana co 64 *ms*. Oznacza to, że w praktyce polecenie odświeżenia wiersza jest wysyłane do matrycy DRAM co około 7,8  $\mu s$ .

Czas trwania jednej operacji odświeżania wiersza zależy od modułów DRAM i (jeśli jest zbyt długi) może powodować znaczne opóźnienia w dostępie do danych.

# Różnice szybkości między procesorami i pamięciami DRAM

W latach 1990-2010 miał miejsce bardzo szybki rozwój technologii półprzewodnikowych, pozwalający na wielokrotne zmniejszenie rozmiarów i skrócenie czasu przełączania  **tranzystorów** .

# Różnice szybkości między procesorami i pamięciami DRAM

W latach 1990-2010 miał miejsce bardzo szybki rozwój technologii półprzewodnikowych, pozwalający na wielokrotne zmniejszenie rozmiarów i skrócenie czasu przełączania  **tranzystorów** .

Niestety nie jest możliwe podobne przyspieszenie matryc pamięci DRAM.

## Różnice szybkości między procesorami i pamięciami DRAM

W latach 1990-2010 miał miejsce bardzo szybki rozwój technologii półprzewodnikowych, pozwalający na wielokrotne zmniejszenie rozmiarów i skrócenie czasu przełączania **tranzystorów**.

Niestety nie jest możliwe podobne przpieszenie matryc pamięci DRAM.

Stworzono technologie pozwalające korzystać z wielu matryc DRAM równolegle, ale pojawiła się kolejna przeszkoda w postaci trudności w konstruowaniu magistral pamięci (im większa jest częstotliwość zegara magistrali, tym trudniejsze jest poprowadzenie linii w taki sposób, aby magistrala nie była zbyt czuła na zakłócenia).

## Różnice szybkości między procesorami i pamięciami DRAM

W latach 1990-2010 miał miejsce bardzo szybki rozwój technologii półprzewodnikowych, pozwalający na wielokrotne zmniejszenie rozmiarów i skrócenie czasu przełączania **tranzystorów**.

Niestety nie jest możliwe podobne przpieszenie matryc pamięci DRAM.

Stworzono technologie pozwalające korzystać z wielu matryc DRAM równolegle, ale pojawiła się kolejna przeszkoda w postaci trudności w konstruowaniu magistral pamięci (im większa jest częstotliwość zegara magistrali, tym trudniejsze jest poprowadzenie linii w taki sposób, aby magistrala nie była zbyt czuła na zakłócenia).

Nawet ostatnia generacja pamięci (DDR3 SDRAM) nie pozwala na zredukowanie różnicy szybkości działania między procesorami i DRAM.



## Problemy z DRAM i schowki procesorów

Odwołania do pamięci DRAM z programu wykonywanego przez współczesny procesor mogą powodować opóźnienia wykonania tego programu rzędu 100-250 (i więcej) cykli zegara.

## Problemy z DRAM i schowki procesorów

Odwołania do pamięci DRAM z programu wykonywanego przez współczesny procesor mogą powodować opóźnienia wykonania tego programu rzędu 100-250 (i więcej) cykli zegara.

W przypadku zastąpienia pamięci DRAM pamięciami statycznymi, opóźnienia te można by było znacznie zredukować (do ok. 90%).

## Problemy z DRAM i schowki procesorów

Odwołania do pamięci DRAM z programu wykonywanego przez współczesny procesor mogą powodować opóźnienia wykonania tego programu rzędu 100-250 (i więcej) cykli zegara.

W przypadku zastąpienia pamięci DRAM pamięciami statycznymi, opóźnienia te można by było znacznie zredukować (do ok. 90%).

Niestety z ekonomicznego punktu widzenia jest to nierealne.

## Problemy z DRAM i schowki procesorów

Odwołania do pamięci DRAM z programu wykonywanego przez współczesny procesor mogą powodować opóźnienia wykonania tego programu rzędu 100-250 (i więcej) cykli zegara.

W przypadku zastąpienia pamięci DRAM pamięciami statycznymi, opóźnienia te można by było znacznie zredukować (do ok. 90%).

Niestety z ekonomicznego punktu widzenia jest to nierealne.

W związku z tym pojawiła się koncepcja wprowadzenia tak zwanych **schowków** (*ang. cache*), czyli bloków statycznej pamięci RAM służących do tymczasowego przechowywania najczęściej używanych danych.

# Strategia wykorzystania schowka

## Pamięć główna (*ang. main memory*)

Pamięć (zwykle zbudowana z modułów DRAM), w której przechowywana jest większość danych (i rozkazów) wykorzystywanych przez procesory w trakcie wykonywania programów.

# Strategia wykorzystania schowka

## Pamięć główna (*ang. main memory*)

Pamięć (zwykle zbudowana z modułów DRAM), w której przechowywana jest większość danych (i rozkazów) wykorzystywanych przez procesory w trakcie wykonywania programów.

Dane odczytywane przez procesor z pamięci głównej najpierw trafiają do schowka. Dane, które już znajdują się w schowku, **nie muszą** być pobierane z pamięci głównej.

# Strategia wykorzystania schowka

## Pamięć główna (*ang. main memory*)

Pamięć (zwykle zbudowana z modułów DRAM), w której przechowywana jest większość danych (i rozkazów) wykorzystywanych przez procesory w trakcie wykonywania programów.

Dane odczytane przez procesor z pamięci głównej najpierw trafiają do schowka. Dane, które już znajdują się w schowku, **nie muszą** być pobierane z pamięci głównej.

## Strategia zapisu zwrotnego (*ang. write back*)

W przypadku zapisu, dane są początkowo zapisywane w schowku. Są one kopiowane do pamięci głównej w czasie, gdy w schowku trzeba je zastąpić innymi danymi (tzn. z opóźnieniem).

## Zasada lokalności

Ponieważ częstotliwość zegara dla komórek pamięci, z których zbudowany jest schowek, może być taka, jak dla procesora, powinno być jasne, że zastosowanie schowka znacznie zwiększa wydajność przy wykonywaniu **niewielkich** programów.



## Zasada lokalności

Ponieważ częstotliwość zegara dla komórek pamięci, z których zbudowany jest schowek, może być taka, jak dla procesora, powinno być jasne, że zastosowanie schowka znacznie zwiększa wydajność przy wykonywaniu **niewielkich** programów.

Okazuje się, że dzięki zastosowaniu schowków można zwiększyć szybkość wykonywania **wszystkich** programów, ponieważ większość programów wykazuje tendencję do „koncentrowania się” na stosunkowo małych blokach danych.

## Zasada lokalności

Ponieważ częstotliwość zegara dla komórek pamięci, z których zbudowany jest schowek, może być taka, jak dla procesora, powinno być jasne, że zastosowanie schowka znacznie zwiększa wydajność przy wykonywaniu **niewielkich** programów.

Okazuje się, że dzięki zastosowaniu schowków można zwiększyć szybkość wykonywania **wszystkich** programów, ponieważ większość programów wykazuje tendencję do „koncentrowania się” na stosunkowo małych blokach danych.

### Zasada lokalności (*ang. locality principle*)

Większość odwołań do pamięci w programach ma lokalny charakter (tzn. średnia różnica adresów przy **kolejnych** odwołaniach do pamięci jest stosunkowo niewielka).

# Hierarchie schowków

Początkowo procesory dysponowały jednym schowkiem, jednak okazało się, że korzystne jest przechowywanie rozkazów i danych w różnych schowkach.

## Hierarchie schowków

Początkowo procesory dysponowały jednym schowkiem, jednak okazało się, że korzystne jest przechowywanie rozkazów i danych w różnych schowkach.

Z przyczyn technicznych czas potrzebny na pobranie danych ze schowka lub zapis danych do schowka jest tym większy, im większe są rozmiary schowka.

## Hierarchie schowków

Początkowo procesory dysponowały jednym schowkiem, jednak okazało się, że korzystne jest przechowywanie rozkazów i danych w różnych schowkach.

Z przyczyn technicznych czas potrzebny na pobranie danych ze schowka lub zapis danych do schowka jest tym większy, im większe są rozmiary schowka.

Dlatego zaczęto stosować niewielkie schowki **I poziomu** (*ang. level 1 cache*) oddzielne dla rozkazów i danych oraz znacznie większe schowki **II poziomu** (*ang. level 2 cache*).

## Hierarchie schowków

Początkowo procesory dysponowały jednym schowkiem, jednak okazało się, że korzystne jest przechowywanie rozkazów i danych w różnych schowkach.

Z przyczyn technicznych czas potrzebny na pobranie danych ze schowka lub zapis danych do schowka jest tym większy, im większe są rozmiary schowka.

Dlatego zaczęto stosować niewielkie schowki **I poziomu** (*ang. level 1 cache*) oddzielne dla rozkazów i danych oraz znacznie większe schowki **II poziomu** (*ang. level 2 cache*).

Przy okazji wprowadzenia na rynek procesorów **wielordzeniowych** (*ang. multicore*) stwierdzono, że korzystne jest dodanie **III poziomu** schowków (wspólnych dla wielu rdzeni).

## Wiersze schowka

Przy kopiowaniu danych z pamięci głównej do schowka (lub ze schowka do pamięci głównej przy zapisie zwrotnym) poszczególne bajty danych **nie są** kopiowane oddzielnie (byłoby to bardzo niewydajne).

## Wiersze schowka

Przy kopiowaniu danych z pamięci głównej do schowka (lub ze schowka do pamięci głównej przy zapisie zwrotnym) poszczególne bajty danych **nie są** kopiowane oddzielnie (byłoby to bardzo niewydajne).

Pamięć główna jest podzielona na rozłączne bloki o jednakowych rozmiarach (zwykle 32 B lub 64 B), z których każdy jest kopiowany **w całości**.



## Wiersze schowka

Przy kopiowaniu danych z pamięci głównej do schowka (lub ze schowka do pamięci głównej przy zapisie zwrotnym) poszczególne bajty danych **nie są** kopiowane oddzielnie (byłoby to bardzo niewydajne).

Pamięć główna jest podzielona na rozłączne bloki o jednakowych rozmiarach (zwykle 32 B lub 64 B), z których każdy jest kopiowany **w całości**.

Odwołanie (w programie) do **jednej** z lokacji w obrębie bloku powoduje skopiowanie **całego bloku**.

## Wiersze schowka

Przy kopiowaniu danych z pamięci głównej do schowka (lub ze schowka do pamięci głównej przy zapisie zwrotnym) poszczególne bajty danych **nie są** kopiowane oddzielnie (byłoby to bardzo niewydajne).

Pamięć główna jest podzielona na rozłączne bloki o jednakowych rozmiarach (zwykle 32 B lub 64 B), z których każdy jest kopiowany **w całości**.

Odwołanie (w programie) do **jednej** z lokacji w obrębie bloku powoduje skopiowanie **całego bloku**.

Schowek jest podzielony na bloki o rozmiarach identycznych z rozmiarami bloków w pamięci głównej, zwane **wierszami schowka** (*ang. cache line*), w których można zapisywać dane pochodzące z bloków w pamięci głównej.

# Pobieranie z wyprzedzeniem

## Obserwacja I

Programy często odwołują się do sekwencji kolejnych lokacji w pamięci (tzn. np. po owołaniu do lokacji o adresie  $n$  następuje odwołanie do lokacji  $n + 1$  itd.).

# Pobieranie z wyprzedzeniem

## Obserwacja I

Programy często odwołują się do sekwencji kolejnych lokacji w pamięci (tzn. np. po owołaniu do lokacji o adresie  $n$  następuje odwołanie do lokacji  $n + 1$  itd.).

## Obserwacja II

Protokół dostępu do pamięci DRAM sprzyja przeprowadzaniu operacji na ciągach lokacji o kolejnych adresach.

# Pobieranie z wyprzedzeniem

## Obserwacja I

Programy często odwołują się do sekwencji kolejnych lokacji w pamięci (tzn. np. po owołaniu do lokacji o adresie  $n$  następuje odwołanie do lokacji  $n + 1$  itd.).

## Obserwacja II

Protokół dostępu do pamięci DRAM sprzyja przeprowadzaniu operacji na ciągach lokacji o kolejnych adresach.

Konstruktorzy procesorów usiłują wykorzystać te obserwacje poprzez stosowanie spekulatywnego **pobierania z wyprzedzeniem** (*ang. prefetch*) danych z bloków w pamięci głównej, do których prawdopodobnie nastąpi odwołanie w dalszej części programu.

# Organizacja schowka

Schowek można wyobrazić sobie jako tablicę złożoną z  $2^k$  wierszy o rozmiarach  $2^w$  B każdy. Łączna pojemność schowka wynosi wtedy  $2^{k+w}$  B.

## Organizacja schowka

Schowek można wyobrazić sobie jako tablicę złożoną z  $2^k$  wierszy o rozmiarach  $2^w$  B każdy. Łączna pojemność schowka wynosi wtedy  $2^{k+w}$  B.

Jeżeli pamięć główna ma pojemność  $2^M = 2^m \times 2^w$  B, to na jeden wiersz schowka przypada  $2^{m-k}$  bloków w pamięci głównej ( $m > k$ ).

## Organizacja schowka

Schowek można wyobrazić sobie jako tablicę złożoną z  $2^k$  wierszy o rozmiarach  $2^w$  B każdy. Łączna pojemność schowka wynosi wtedy  $2^{k+w}$  B.

Jeżeli pamięć główna ma pojemność  $2^M = 2^m \times 2^w$  B, to na jeden wiersz schowka przypada  $2^{m-k}$  bloków w pamięci głównej ( $m > k$ ).

Okazuje się, że najlepsze efekty dają **zbiorowo-asocjatywne** (*ang. set-associative*) sposoby przyporządkowywania bloków w pamięci głównej do wierszy schowka.



## Organizacja schowka

Schówek można wyobrazić sobie jako tablicę złożoną z  $2^k$  wierszy o rozmiarach  $2^w$  B każdy. Łączna pojemność schowka wynosi wtedy  $2^{k+w}$  B.

Jeżeli pamięć główna ma pojemność  $2^M = 2^m \times 2^w$  B, to na jeden wiersz schowka przypada  $2^{m-k}$  bloków w pamięci głównej ( $m > k$ ).

Okazuje się, że najlepsze efekty dają **zbiorowo-asocjatywne** (*ang. set-associative*) sposoby przyporządkowywania bloków w pamięci głównej do wierszy schowka.

Schówek jest dzielony na  $2^s$  części, zwanych **zbiorami** (*ang. set*), po  $2^r$  wierszy w każdym. Wtedy  $k = s + r$ .

## Organizacja schowka (c. d.)

Ponumerujmy bloki w pamięci głównej od 0 do  $(2^m - 1)$  i wprowadźmy przyporządkowanie, w którym bloki o numerach  $n + j \times 2^s$ , gdzie  $n < 2^s$ , odpowiadają **temu samemu zbiorowi** wierszy w schowku.

## Organizacja schowka (c. d.)

Ponumerujmy bloki w pamięci głównej od 0 do  $(2^m - 1)$  i wprowadźmy przyporządkowanie, w którym bloki o numerach  $n + j \times 2^s$ , gdzie  $n < 2^s$ , odpowiadają **temu samemu zbiorowi** wierszy w schowku.

Przyjmijmy, że dla każdego z tych bloków w pamięci głównej dane będą mogły być przechowywane **tylko** w jednym z wierszy schowka należących do zbioru, który odpowiada temu blokowi.

## Organizacja schowka (c. d.)

Ponumerujmy bloki w pamięci głównej od 0 do  $(2^m - 1)$  i wprowadźmy przyporządkowanie, w którym bloki o numerach  $n + j \times 2^s$ , gdzie  $n < 2^s$ , odpowiadają **temu samemu zbiorowi** wierszy w schowku.

Przyjmijmy, że dla każdego z tych bloków w pamięci głównej dane będą mogły być przechowywane **tylko** w jednym z wierszy schowka należących do zbioru, który odpowiada temu blokowi.

Wtedy dla każdego bloku w pamięci głównej dane z tego bloku mogą być przechowywane w schowku na  $2^r$  sposobów oraz istnieje  $2^{m-s-r} - 1$  bloków, które „konkurują” z nim o „dostęp” do schowka.

## Organizacja schowka (c. d.)

Ponumerujmy bloki w pamięci głównej od 0 do  $(2^m - 1)$  i wprowadźmy przyporządkowanie, w którym bloki o numerach  $n + j \times 2^s$ , gdzie  $n < 2^s$ , odpowiadają **temu samemu zbiorowi** wierszy w schowku.

Przyjmijmy, że dla każdego z tych bloków w pamięci głównej dane będą mogły być przechowywane **tylko** w jednym z wierszy schowka należących do zbioru, który odpowiada temu blokowi.

Wtedy dla każdego bloku w pamięci głównej dane z tego bloku mogą być przechowywane w schowku na  $2^r$  sposobów oraz istnieje  $2^{m-s-r} - 1$  bloków, które „konkurują” z nim o „dostęp” do schowka.

Przy tym dane z  $2^r$  bloków w każdej grupie  $2^{m-s}$  bloków w pamięci głównej mogą **jednocześnie** być w schowku.

## Organizacja schowka (c. d.)

Wtedy mówi się, że schowek ma  $2^r$ -torową organizację zbiorowo-asocjatywną (*ang.  $2^r$ -way set-associative*).

## Organizacja schowka (c. d.)

Wtedy mówi się, że schowek ma  $2^r$ -torową organizację zbiorowo-asocjatywną (*ang.*  $2^r$ -way set-associative).

Okazuje się, że dla  $r = 1$  wydajność rośnie blisko dwukrotnie w stosunku do  $r = 0$ . Jednak dla  $r > 3$  wydajność jest praktycznie taka, jak dla  $r = 3$ .

## Organizacja schowka (c. d.)

Wtedy mówi się, że schowek ma  $2^r$ -torową organizację zbiorowo-asocjatywną (*ang.*  $2^r$ -way set-associative).

Okazuje się, że dla  $r = 1$  wydajność rośnie blisko dwukrotnie w stosunku do  $r = 0$ . Jednak dla  $r > 3$  wydajność jest praktycznie taka, jak dla  $r = 3$ .

Ogólnie stopień skomplikowania elektroniki realizującej schowek rośnie wraz z  $r$ , natomiast czas dostępu do schowka rośnie wraz z  $s$ .



## Organizacja schowka (c. d.)

Wtedy mówi się, że schowek ma  $2^r$ -torową organizację zbiorowo-asocjatywną (*ang.*  $2^r$ -way set-associative).

Okazuje się, że dla  $r = 1$  wydajność rośnie blisko dwukrotnie w stosunku do  $r = 0$ . Jednak dla  $r > 3$  wydajność jest praktycznie taka, jak dla  $r = 3$ .

Ogólnie stopień skomplikowania elektroniki realizującej schowek rośnie wraz z  $r$ , natomiast czas dostępu do schowka rośnie wraz z  $s$ .

Dlatego schowki I poziomu zwykle mają 2-torową organizację zbiorowo-asocjatywną, natomiast dla schowków II i III poziomu  $r$  może być w granicach od 2 do 4.

## Adresy fizyczne i wirtualne

Adresy używane przez programy na ogół **nie są** adresami, których używa kontroler pamięci do dentyfikacji bitów w matrycach DRAM.

## Adresy fizyczne i wirtualne

Adresy używane przez programy na ogół **nie są** adresami, których używa kontroler pamięci do dentyfikacji bitów w matrycach DRAM.

### Adresy fizyczne (*ang. physical*)

Adresy używane do identyfikacji lokacji pamięci przy fizycznym przesyłaniu danych (np. z CPU do modułów DRAM).

## Adresy fizyczne i wirtualne

Adresy używane przez programy na ogół **nie są** adresami, których używa kontroler pamięci do dentyfikacji bitów w matrycach DRAM.

### Adresy fizyczne (*ang. physical*)

Adresy używane do identyfikacji lokacji pamięci przy fizycznym przesyłaniu danych (np. z CPU do modułów DRAM).

### Adresy wirtualne (*ang. virtual*)

Adresy używane przez oprogramowanie przy odwołaniach do pamięci.

## Adresy fizyczne i wirtualne

Adresy używane przez programy na ogół **nie są** adresami, których używa kontroler pamięci do dentyfikacji bitów w matrycach DRAM.

### Adresy fizyczne (*ang. physical*)

Adresy używane do identyfikacji lokacji pamięci przy fizycznym przesyłaniu danych (np. z CPU do modułów DRAM).

### Adresy wirtualne (*ang. virtual*)

Adresy używane przez oprogramowanie przy odwołaniach do pamięci.

Jednemu adresowi fizycznemu może odpowiadać wiele adresów wirtualnych.

## Bloki stronicowe

Przyporządkowanie między adresami wirtualnymi i fizycznymi jest realizowane poprzez użycie **stron pamięci** (*ang. memory page*) oraz **tabel translacji** (*ang. translation table*).

## Bloki stronicowe

Przyporządkowanie między adresami wirtualnymi i fizycznymi jest realizowane poprzez użycie **stron pamięci** (*ang. memory page*) oraz **tabel translacji** (*ang. translation table*).

**Fizyczna przestrzeń adresowa** (*ang. physical address space*)

Zbiór wszystkich możliwych fizycznych adresów, których może używać dany procesor (lub ogólnie rodzina procesorów). Jej rozmiar jest maksymalnym możliwym rozmiarem pamięci RAM dla tego procesora.

## Bloki stronicowe

Przyporządkowanie między adresami wirtualnymi i fizycznymi jest realizowane poprzez użycie **stron pamięci** (*ang. memory page*) oraz **tabel translacji** (*ang. translation table*).

### Fizyczna przestrzeń adresowa (*ang. physical address space*)

Zbiór wszystkich możliwych fizycznych adresów, których może używać dany procesor (lub ogólnie rodzina procesorów). Jej rozmiar jest maksymalnym możliwym rozmiarem pamięci RAM dla tego procesora.

Fizyczna przestrzeń adresowa jest dzielona na **bloki stronicowe** (*ang. page frame*) o rozmiarach  $2^b$  B każdy ( $b \geq 12$ ). Jeżeli  $f$  jest numerem bloku stronicowego, to fizyczne adresy lokacji w obrębie tego bloku mają postać  $f + o$ , gdzie  $o < 2^b$  nazywa się **ofsetem** (*ang. offset*).



# Strony pamięci

## Wirtualna przestrzeń adresowa (*ang. virtual address space*)

Zbiór wszystkich możliwych wirtualnych adresów, których może używać oprogramowanie wykonywane przez dany procesor (lub ogólnie rodzinę procesorów).

# Strony pamięci

## Wirtualna przestrzeń adresowa (*ang. virtual address space*)

Zbiór wszystkich możliwych wirtualnych adresów, których może używać oprogramowanie wykonywane przez dany procesor (lub ogólnie rodzinę procesorów).

Rozmiary wirtualnej przestrzeni adresowej na ogół są różne od rozmiarów fizycznej przestrzeni adresowej (i różne od rozmiarów pamięci RAM).

# Strony pamięci

## Wirtualna przestrzeń adresowa (*ang. virtual address space*)

Zbiór wszystkich możliwych wirtualnych adresów, których może używać oprogramowanie wykonywane przez dany procesor (lub ogólnie rodzinę procesorów).

Rozmiary wirtualnej przestrzeni adresowej na ogół są różne od rozmiarów fizycznej przestrzeni adresowej (i różne od rozmiarów pamięci RAM).

Potrzebna jest metoda wyznaczania adresu fizycznego dla danego adresu wirtualnego.

# Strony pamięci

## Wirtualna przestrzeń adresowa (*ang. virtual address space*)

Zbiór wszystkich możliwych wirtualnych adresów, których może używać oprogramowanie wykonywane przez dany procesor (lub ogólnie rodzinę procesorów).

Rozmiary wirtualnej przestrzeni adresowej na ogół są różne od rozmiarów fizycznej przestrzeni adresowej (i różne od rozmiarów pamięci RAM).

Potrzebna jest metoda wyznaczania adresu fizycznego dla danego adresu wirtualnego.

W tym celu bloki adresów w wirtualnej przestrzeni adresowej, zwane **stronami pamięci** (*ang. memory page*), mapuje się na odpowiednie bloki stronicowe.

## Translacja adresów stron pamięci

Część adresu wirtualnego „ponad” ofsetem dzielona jest na ciągi bitów (zwykle o jednakowej długości). Nazwijmy je  $k_1, \dots, k_n$  tak, aby  $k_1$  był ciągiem najmniej znaczących, a  $k_n$  – ciągiem najbardziej znaczących bitów.

## Translacja adresów stron pamięci

Część adresu wirtualnego „ponad” ofsetem dzielona jest na ciągi bitów (zwykle o jednakowej długości). Nazwijmy je  $k_1, \dots, k_n$  tak, aby  $k_1$  był ciągiem najmniej znaczących, a  $k_n$  – ciągiem najbardziej znaczących bitów.

- 1  $k_n$  jest indeksem w tabeli, której adres fizyczny znajduje się w jednym z rejestrów (*ang. register*) procesora i wyznacza komórkę zawierającą adres fizyczny kolejnej tabeli.
- 2  $k_{n-1}$  jest indeksem w tabeli, której adres fizyczny został wyznaczony w poprzednim kroku. Określa on komórkę zawierającą adres fizyczny kolejnej tabeli.
- 3 ...
- 4  $k_1$  jest indeksem w tabeli, której adres fizyczny został wyznaczony w poprzednim kroku. Określa on komórkę zawierającą adres fizyczny poszukiwanego bloku stronicowego.

## Procesy i tabele translacji

Część adresu wirtualnego odpowiadającą ofsetowi dodaje się do fizycznego adresu bloku stronicowego odczytanego z tabel translacji adresów stron pamięci. W ten sposób otrzymuje się adres fizyczny lokacji, której dotyczy odwołanie do pamięci w programie.

## Procesy i tabele translacji

Część adresu wirtualnego odpowiadającą ofsetowi dodaje się do fizycznego adresu bloku stronicowego odczytanego z tabel translacji adresów stron pamięci. W ten sposób otrzymuje się adres fizyczny lokacji, której dotyczy odwołanie do pamięci w programie.

Każdy proces w ogólności dysponuje **własną** strukturą tabel translacji adresów stron pamięci. Dlatego można powiedzieć, że każdy proces dysponuje własną przestrzenią adresową i „widzi” pamięć komputera w szczególny sposób.



## Procesy i tabele translacji

Część adresu wirtualnego odpowiadającą ofsetowi dodaje się do fizycznego adresu bloku stronicowego odczytanego z tabel translacji adresów stron pamięci. W ten sposób otrzymuje się adres fizyczny lokacji, której dotyczy odwołanie do pamięci w programie.

Każdy proces w ogólności dysponuje **własną** strukturą tabel translacji adresów stron pamięci. Dlatego można powiedzieć, że każdy proces dysponuje własną przestrzenią adresową i „widzi” pamięć komputera w szczególny sposób.

Podczas przełączania zadań procesor jest „przełączany” z tabel translacji stron pamięci jednego procesu na tabele translacji stron pamięci drugiego procesu.

# TLB

Mechanizm pamięci wirtualnej bez buforowania byłby **bardzo** kosztowny (wymaga  $n$  odwołań do pamięci w celu uzyskania 1 adresu fizycznego).

# TLB

Mechanizm pamięci wirtualnej bez buforowania byłby **bardzo** kosztowny (wymaga  $n$  odwołań do pamięci w celu uzyskania 1 adresu fizycznego).

Dlatego wyniki pewnej liczby ostatnich operacji poszukiwania fizycznego adresu bloku stronicowego dla danego adresu wirtualnego są przechowywane w specjalnym schowku.

# TLB

Mechanizm pamięci wirtualnej bez buforowania byłby **bardzo** kosztowny (wymaga  $n$  odwołań do pamięci w celu uzyskania 1 adresu fizycznego).

Dlatego wyniki pewnej liczby ostatnich operacji poszukiwania fizycznego adresu bloku stronicowego dla danego adresu wirtualnego są przechowywane w specjalnym schowku.

TLB (*ang. Translation Lookaside Buffer*)

Schówek, w którym procesor przechowuje fizyczne adresy dla pewnej liczby adresów wirtualnych, ostatnio używanych przez oprogramowanie przy odwołaniach do pamięci.

## Wpływ TLB na wydajność

TLB ma strukturę **w pełni asocjatywną** (*ang. fully associative*), co oznacza, że pełny adres wirtualny strony pamięci jest używany do wyznaczenia wiersza przechowującego odpowiadający mu adres fizyczny.

## Wpływ TLB na wydajność

TLB ma strukturę **w pełni asocjatywną** (*ang. fully associative*), co oznacza, że pełny adres wirtualny strony pamięci jest używany do wyznaczenia wiersza przechowującego odpowiadający mu adres fizyczny.

Dlatego rozmiary TLB są zazwyczaj **bardzo** ograniczone.

## Wpływ TLB na wydajność

TLB ma strukturę **w pełni asocjatywną** (*ang. fully associative*), co oznacza, że pełny adres wirtualny strony pamięci jest używany do wyznaczenia wiersza przechowującego odpowiadający mu adres fizyczny.

Dlatego rozmiary TLB są zazwyczaj **bardzo** ograniczone.

Jednocześnie odwołania z użyciem adresów wirtualnych, dla których nie ma adresów fizycznych w TLB są **bardzo** kosztowne.

## Wpływ TLB na wydajność

TLB ma strukturę **w pełni asocjatywną** (*ang. fully associative*), co oznacza, że pełny adres wirtualny strony pamięci jest używany do wyznaczenia wiersza przechowującego odpowiadający mu adres fizyczny.

Dlatego rozmiary TLB są zazwyczaj **bardzo** ograniczone.

Jednocześnie odwołania z użyciem adresów wirtualnych, dla których nie ma adresów fizycznych w TLB są **bardzo** kosztowne.

Dlatego program, który powoduje częste zmiany w TLB (odwołuje się ciągle do innych obszarów przestrzeni adresowej), będzie wykonywany bardzo wolno w porównaniu z programami, które tego nie robią.



# Wpływ architektury klasy NUMA na wydajność

## NUMA (*ang. Non-Uniform Memory Access*)

Sposób konstrukcji komputera, przy którym pamięć RAM jest podzielona na tak zwane **węzły** (*ang. node*) w taki sposób, że czas dostępu do pamięci w różnych węzłach z tego samego CPU jest w ogólności różny.

# Wpływ architektury klasy NUMA na wydajność

## NUMA (*ang. Non-Uniform Memory Access*)

Sposób konstrukcji komputera, przy którym pamięć RAM jest podzielona na tak zwane **węzły** (*ang. node*) w taki sposób, że czas dostępu do pamięci w różnych węzłach z tego samego CPU jest w ogólności różny.

Zwykle dla każdego procesora (lub rdzenia) istnieje jeden węzeł, do którego ma on „najszybszy” dostęp oraz wiele węzłów, czas dostępu do których jest dla niego znacznie dłuższy.

# Wpływ architektury klasy NUMA na wydajność

## NUMA (*ang. Non-Uniform Memory Access*)

Sposób konstrukcji komputera, przy którym pamięć RAM jest podzielona na tak zwane **węzły** (*ang. node*) w taki sposób, że czas dostępu do pamięci w różnych węzłach z tego samego CPU jest w ogólności różny.

Zwykle dla każdego procesora (lub rdzenia) istnieje jeden węzeł, do którego ma on „najszybszy” dostęp oraz wiele węzłów, czas dostępu do których jest dla niego znacznie dłuższy.

Wtedy, jeśli program wykonywany przez procesor odwołuje się do pamięci zlokalizowanej w „bardziej odległych” węzłach pamięci, to w ogólności będzie on wykonywany dużo wolniej od programu, który odwołuje się do pamięci zlokalizowanej w węzle „najbliższym” danemu procesorowi.

# Literatura



U. Drepper, *What every programmer should know about memory*  
(<http://people.redhat.com/drepper/cpumemory.pdf>).