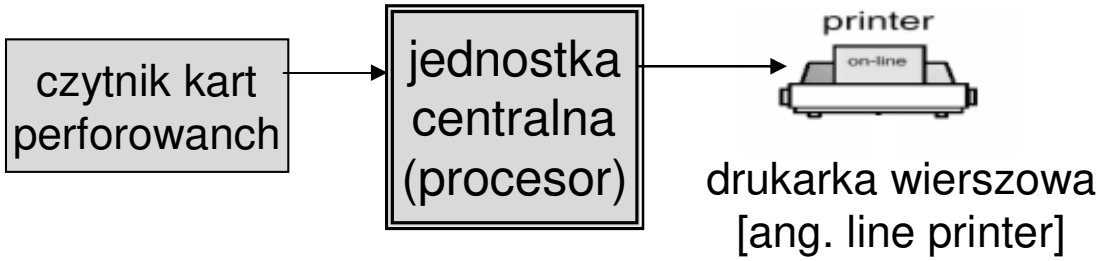
# Budowa "ogólnego" s.o.

- Historia rozwoju s.o.
  - proste systemy wsadowe
  - systemy wsadowe ze SPOOLingiem
  - wieloprogramowe systemy wsadowe
  - systemy z podziałem czasu
  - s.o. dla komputerów osobistych
  - systemy rozproszone/ równoległe

# Historia rozwoju s.o.

## *Proste systemy wsadowe*

- zadanie [ang. task] = program + dane wejściowe
- wsad [ang. batch] = zbiór zadań
- cecha charakterystyczna systemów wsadowych:
  użytkownik NIE współpracuje bezpośrednio z komputerem




drukarka wierszowa
[ang. line printer]

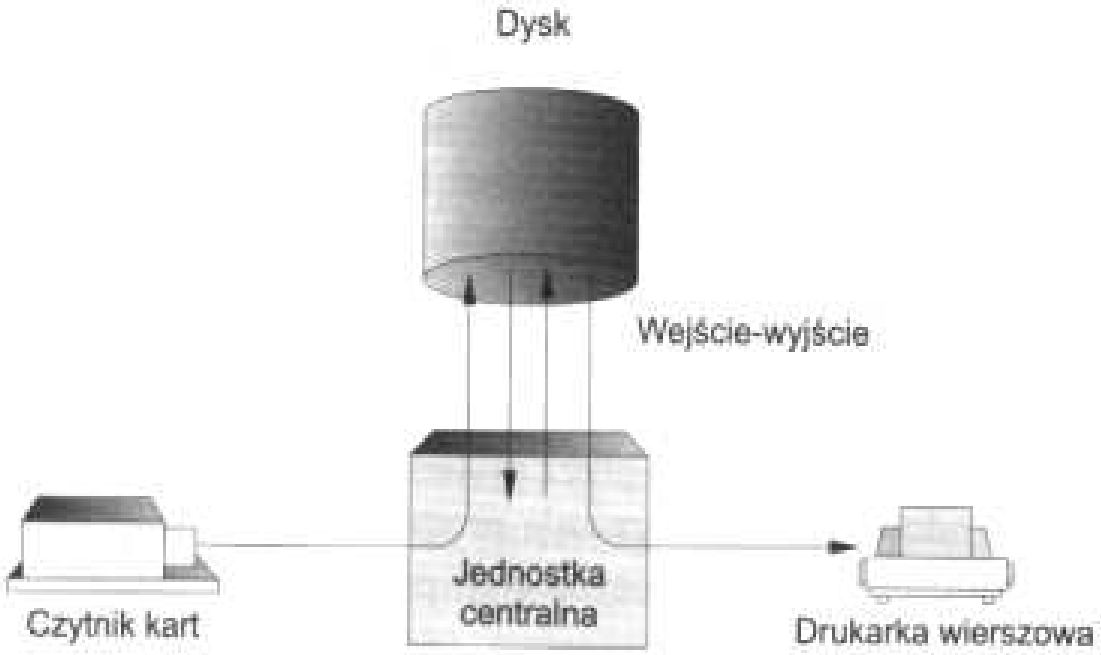
- jak to działa ?
  - jedn. centralna czyta "zadanie 1" z kart perforowanych do pamięci operacyjnej
  - "zadanie 1" zaczyna się wykonywać
  - "zadanie 1" drukuje wyniki działania
  - jedn. centralna czyta "zadanie 2" z kart perforowanych
  - "zadanie 2" zaczyna się wykonywać
  - .....

***kolejne zadanie wprowadzamy dopiero gdy poprzednie wydrukowało wyniki swojego działania !***

- wady systemów wsadowych:
  - w danej chwili pracuje 1 z 3 urządzeń
  - w czasie działania czytnika lub drukarki
    jednostka centralna (=procesor; bardzo drogi) "marnuje" swój czas
- rola s.o. w prostym systemie wsadowym:
  - jedynie wprowadzanie kolejnych zadań

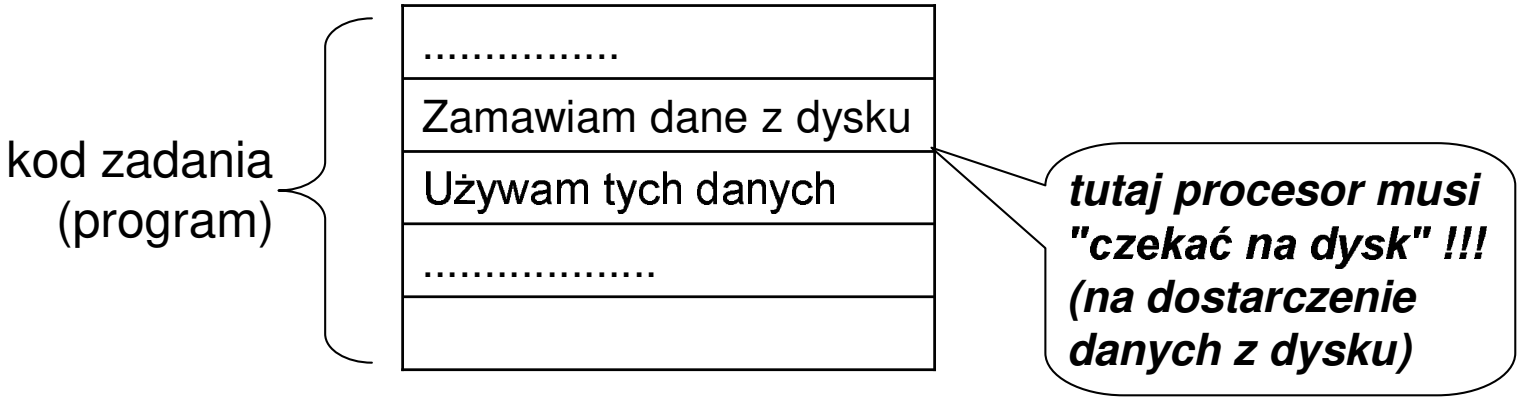
# *Systemy wsadowe ze SPOOL-ingiem*

- aby zwiększyć wykorzystanie procesora wprowadzono SPOOLing
- SPOOL = jednoczesna, bezpośrednia praca urządzeń [ang. Simultaneous Peripherial Operations Online]
- używa się ***bufora*** na dysku magnetycznym który pośredniczy między:
  - czytnikiem kart i jedn. centralną
  - jedn. centralną i drukarką



- jak to działa:
  1. jedn. centr. wykonuje zadanie 1
  2. równocześnie czytnik zaczyna wczytywać na dysk zadania 2, 3, ... "z wyprzedzeniem"
  3. zadanie 1 drukuje wyniki "na dysk"
  4. jedn. centr. pobiera z dysku zadanie 2 i zaczyna je wykonywać
  5. równocześnie drukarka zaczyna drukować wyniki zadania 1 (pobierane z dysku)
  6. .....
- wszystkie trzy urządzenia pracują równocześnie (nie są bezczynne !)
- istnieje dysproporcja szybkości działania urządzeń:
  - dysk magn. jest dużo szybszy od czytnika i drukarki
  - jedn. centralna jest dużo szybsza niż dysk magnetyczny
- SPOOLing neutralizuje (częściowo) różnicę prędkości działania jedn. centralnej i czytnika/drukarki przy pomocy dysku magn. !!!
  - zamiast drukować wyniki na powolnej drukarce jedn. centralna zapisuje je na trochę szybszym dysku ("drukowanie" jest pozornie szybsze)
  - zamiast czytać kolejne zadania z wolnego czytnika, jedn. centralna odczytuje je z szybszego dysku (musiały być tam zapisane "z wyprzedzeniem")

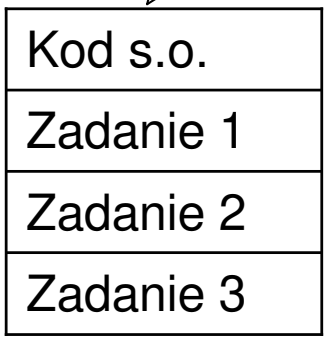
- problem którego SPOOLing ***nie*** rozwiązuje:
  - dysproporcja między szybkością jedn. centr. i dysku magnetycznego



# *Wieloprogramowe systemy wsadowe*

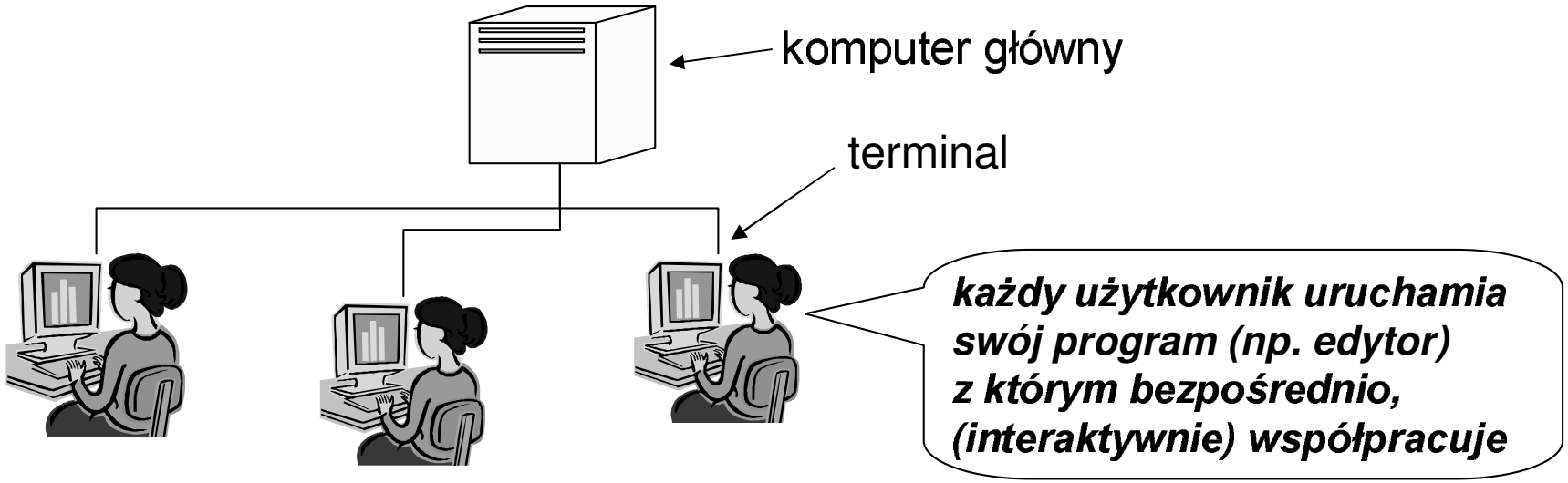
- (w SPOOLingu mieliśmy *pulę zadań na* dysku ...)
- ***w wieloprogramowym sys. wsadowym do pamięci operacyjnej wprowadzamy kilka zadań z puli***
- jak to działa:
  - gdy zadania 1 zamawia dane z dysku i ***czeka*** na ich dostarczenie wtedy procesor ***przełącza się*** na inne zdanie (np. zadanie 2)
- gdy choć jedno zadanie nie czeka na dane z dysku to procesor nie jest bezczynny
- dysk i procesor działają równocześnie
- ***przełączenie procesora na inne zadanie następuje wyłącznie wtedy gdy bieżące zadanie zaczyna czekać na dane z dysku***
- nowe problemy:
  - które zadania z puli zadań załadować do pamięci operacyjnej ?
    - żeby nie powstawały "dziury" w pamięci oper.
    - żeby w pamięci oper. była mieszanina procesów często/ rzadko wykonujących operacje dyskowe (aby procesor i dysk miały zawsze zajęcie)

***tak wygląda zawartość pamięci operacyjnej w wieloprog. systemie wsadowym:***

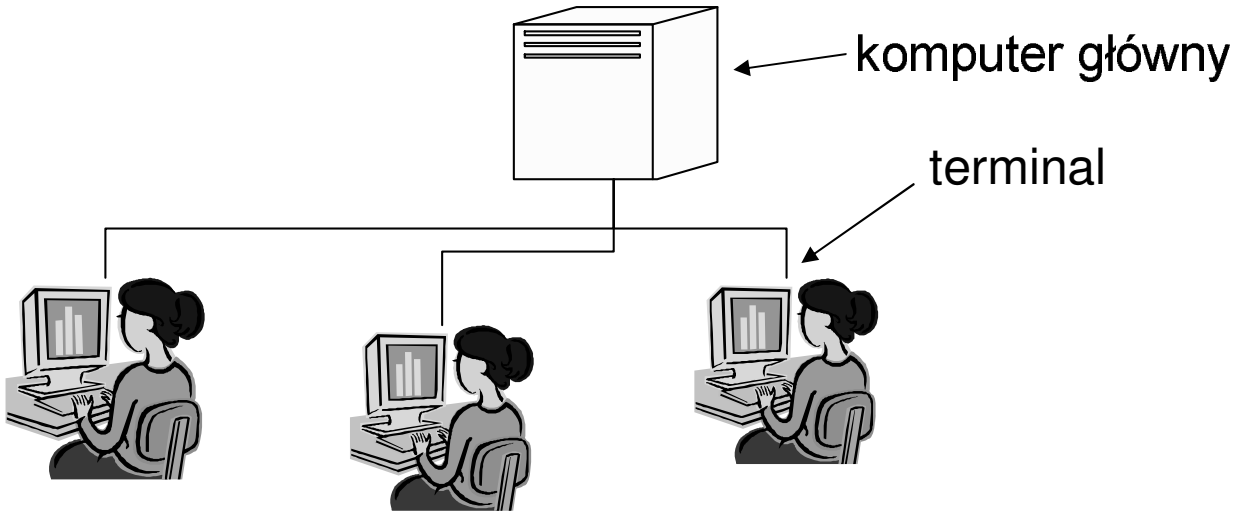


# *Systemy z podziałem czasu*

- ***są to systemy wieloprogramowe w których przełączanie procesora między zadaniami następuje bardzo szybko (jest wymuszane)***
- terminologia:
  - podział czasu = wielozadaniowość
  - wymuszanie przełączenia procesora = wywłaszczanie
- skąd się wzięła potrzeba podziału czasu ?
  Odp: ***wielodostęp ...***



- co by się stało gdyby użyć systemu wieloprogramowego *bez* podziału czasu ?
  Odp: program który nie wykonuje operacji dyskowych zawłaszczyłby procesor dla siebie ...
- podział czasu powoduje że każdemu użytkownikowi zdaje się że ma komputer wyłącznie dla siebie (czyli że nie ma innych użytkow.)
- podział czasu ma sens także w jedno-użytkownikowych s.o. !
  (np. gdy użytkownik uruchamia edytor + program wykonujący obliczenia)



# *Systemy "wsadowe" kontra "z podziałem czasu"*

- *systemu obu typów są przystosowane do wykonywania innych programów ...*
- *systemy wsadowe*
  - *brak interakcji użytkownik/program*
  - *Unix: procesy drugoplanowe*
    *(idea systemów wsadowych przetrwała w postaci procesów drugoplanowych !!!)*
- *podział czasu*
  - *jest interakcja z użytkownikiem*
  - *Unix: procesy pierwszoplanowe*

# S.o. dla komputerów osobistych.

- DOS = Disk Operating System
  - zaprojektowany dla procesorów firmy Intel, 8086, 80286
  - obsługiwał 640 KB pamięci operacyjnej (+ pamięć Extended i Expanded)
  - różne wersje:
    MS DOS (Microsoft),
    PC DOS (IBM),
    DR DOS (Digital Research)
  - przykładowo PC DOS składał się z następujących plików:
    - ibmbio.com
    - ibmdos.com

    } jądro
    - command.com – interpreter poleceń
  - DOS zawierał:
    - obsługę systemu plików FAT (drzewo katalogów, nazwy plików 8+3)
    - zarządzanie pamięcią operacyjną
    - (kiepskie) zarządzanie procesami
      - można uruchomić kilka programów, ale tylko 1 z nich działa w danej chwili
    - możliwość dołączania dodatkowych "driverów" (= programów obsługi urządzeń)
      -plik config.sys, polecenie DEVICE=)

- **MS-Windows 3.1** (1992) – nakładka na DOS ale wielozadaniowość (!),
  Windows 3.11 i Windows for Workgroups.
- Windows NT 3.1 (1993),
  Windows NT 3.5 (1994),
  Windows NT 4 (1996);
  **Windows NT:**
  **NT = New Technology = Nowa Technologia;**
  **samodzielny system operacyjny, a nie nakładka na DOS**
  **nowoczesny s.o. z architekturą opartą o tzw "mikrojądro"**
- Windows 95 (listopad 1995),
  4 wersje Windows 98 (1998)
- Windows 2000
- Windows XP
  - XP Professional
  - XP HomeEdition

produkty firmy Microsoft

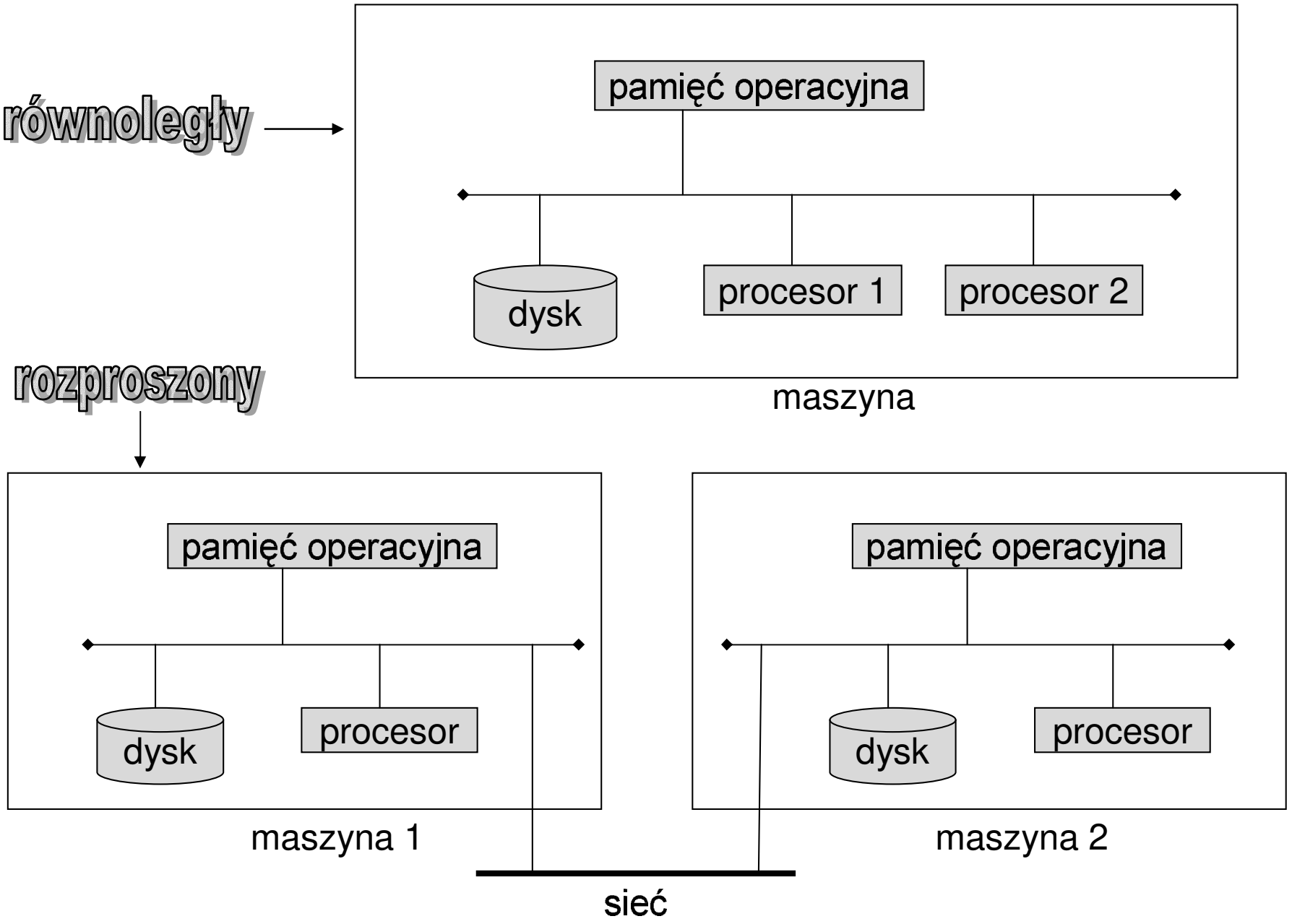
- Linux
  całkowicie darmowy !!!
  różne "dystrybucje" (czyli wersje): RedHat, Mandrake, Debian, Caldera

# Systemy równoległe i rozproszone.

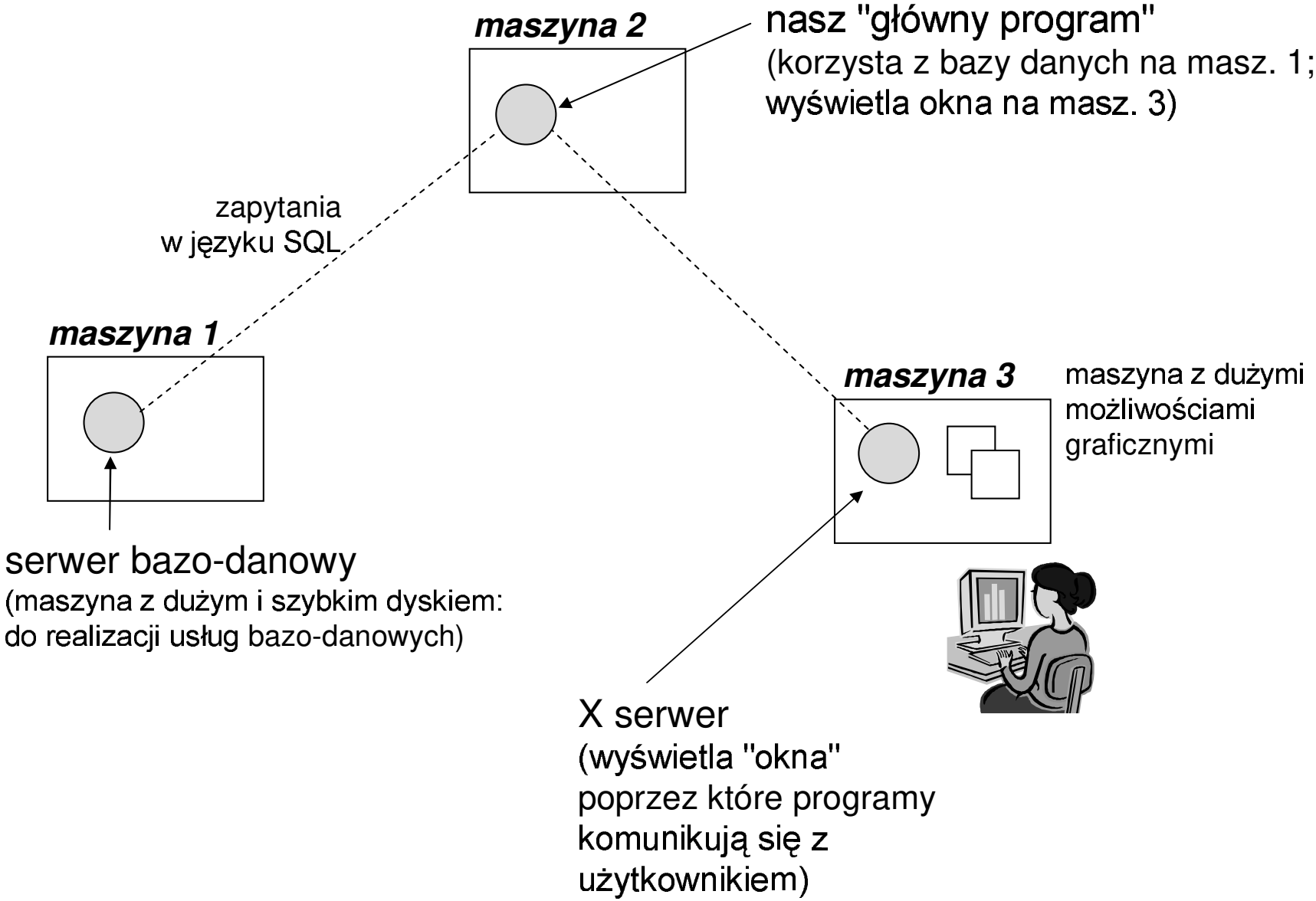
- chodzi o systemy z wieloma procesorami ...
  - przez "system" rozumiemy sprzęt + oprogramowanie (kod s.o.)
  - wyjaśnimy różnicę między systemami równoległymi i rozproszonymi
- systemy równoległe
  - inaczej: systemy ciasno powiązane
  - procesory mają dostęp do wspólnej pamięci operacyjnej, do wspólnych dysków i innych urządzeń
  - ***cel***: zwiększenie szybkości "obliczeń" (niektóre zadania obliczeniowe można podzielić na części które mogą być obliczane równocześnie, przez osobne procesory)
  - ***niestety***, wprowadzenie n -procesorów nie dzieli czasu przez "n" !!!
  - modele wieloprzetwarzania (ang. multiprocessing)
    - symetryczne (SMP)
      na każdym procesorze identyczna kopia s.o.
    - asymetryczne
      procesor główny i podrzędne, które dostają zadania od głównego

- systemy rozproszone
  - inaczej: systemy luźno powiązane
  - procesory komunikują się poprzez sieć (nie mają wspólnej pamięci operacyjnej !)
  - ***cel:***
    - podział zasobów (np. "dzielone" drukarki i katalogi w sieci lokalnej z komputerami wyposażonymi w Win95)
    - przyspieszenie obliczeń (ale komunikacja jest bardziej czasochłonna niż w systemach równoległych)
    - niezawodność (gdy jeden z komputerów [składników systemu rozproszonego] się zepsuje inne mogą przejąć jego funkcje)
    - aplikacje rozproszone
      - składają się z kilku części (procesów) działających na różnych maszynach
      - sens: każda część działa na maszynie która się do tego najlepiej nadaje
      - model klient/serwer

# Architektura systemu równoległego i rozproszonego ...



# Prosty przykład aplikacji rozproszonej ...



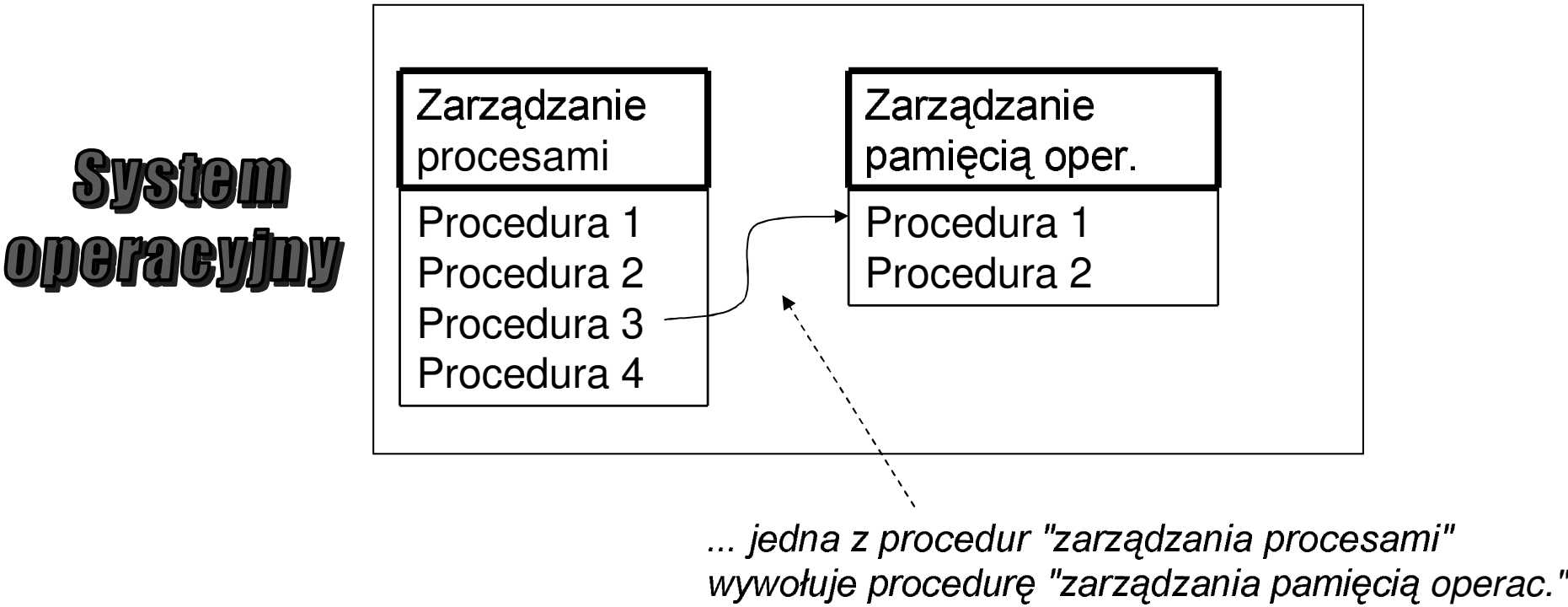
# Budowa "ogólnego" s.o.

- Klasyfikacja architektur s.o.
  - architektura prosta
  - architektura warstwowa
  - architektura z mikrojądrem
  - architektura z maszynami wirtualnymi

# Klasyfikacja architektur s.o.

wstęp:

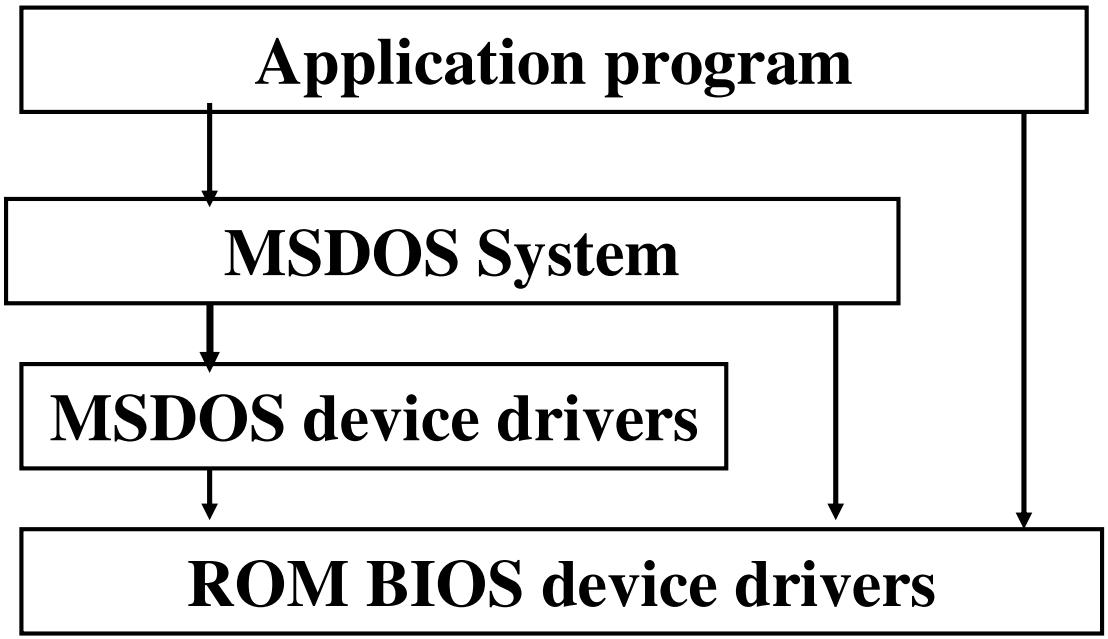
- w s.o. można wyróżnić części, np: "zarządzanie procesami", "zarządzanie pamięcią operacyjną", ...
- każda część zawiera *zbiór procedur*; procedury te wywołują się wzajemnie
- będziemy się zajmować organizacją tych procedur ...



# Architektura prosta

- zwana także "monolityczną"
- każda procedura może wywoływać każdą inną
- przykłady: MS-DOS, wczesne klasyczne wersje Unix-a, **a także Linux !!!**

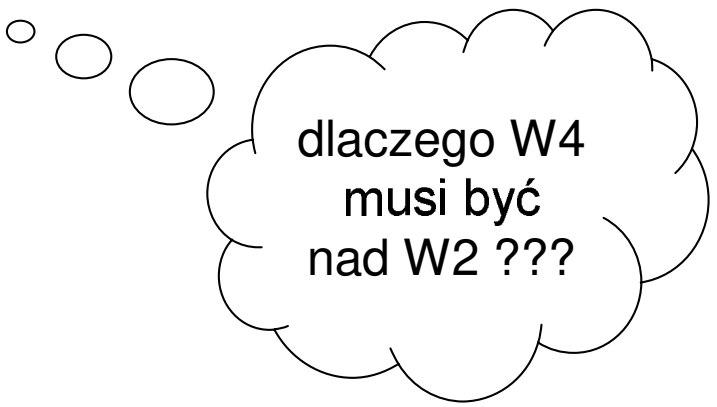
***MS DOS:***



# Architektura warstwowa

- procedury są podzielone na *warstwy* ...
- procedura warstwy M może wywoływać procedury warstw <= M
- implementacja danej warstwy opiera się na procedurach dostarczonych przez niższe warstwy
- warstwa najwyższa = interfejs użytkownika
  warstwa najniższa (nr 0) = sprzęt
- przykład: system THE (Technische Hogeschool w Eindhoven)
  – pierwszy w którym zastosowano podejście warstwowe :

| | |
|---|---|
| W5 | Programy użytkowników |
| W4 | Buforowanie urządzeń we/wy |
| W3 | Program obsługi konsoli operatora |
| W2 | Zarządzanie pamięcią operacyjną |
| W1 | Planowanie przydziału procesora |
| W0 | Sprzęt |



# Architektura warstwowa c.d.

- zalety:
  - łatwość programowania s.o.;
    można zacząć od niższej warstwy, po jej zaprogramowaniu i przetestowaniu przechodzimy do następnej wyższej warstwy ...
- wady:
  - kłopoty z podziałem na warstwy;
    jeśli procedury ze zbiorów Z1 i Z2 wywołują się wzajemnie to nie można ich umieścić w oddzielnych warstwach ...
  - mniejsza wydajność: wywołanie procedury wymaga niekiedy przejścia przez kilka warstw (dlatego współczesne s.o. składają się z niewielkiej liczby warstw) [?]

# Architektura z "mikrojądrem"

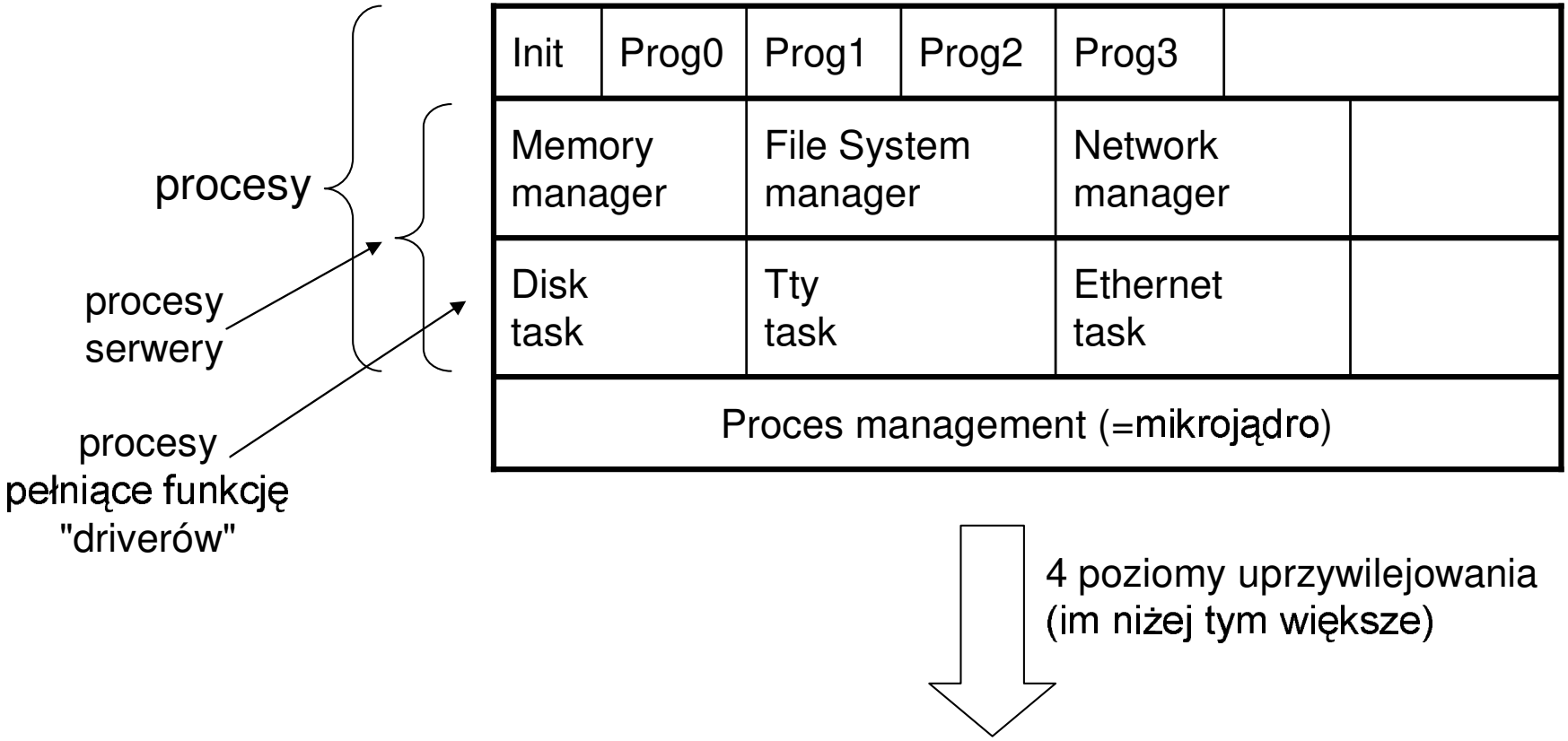
- *mikrojądro* = zredukowane jądro, wykonujące jedynie elementarne funkcje systemu operacyjnego:
  - przełączanie procesora między procesami (realizacja wielozadaniowości)
  - komunikacja między procesami (IPC = Inter Process Communication)
    sposób komunikacji: przesyłanie komunikatów;
    komunikaty mogą być wysyłane przez:
    - proces do procesu
    - mikrojądro do procesu
  - podstawowe zarządzanie pamięcią operacyjną (w tym "ochrona pamięci")
  - podstawowa obsługa przerwań sprzętowych (zgłaszanych przez urządzenia we/wy)
- *procesy/serwery (zwane też "podsystemami")*
  - pozostałe funkcje s.o. wykonują procesy zwane "serwerami" (zasadniczo nie różniące się od procesów użytkowników !!!)
  - przywileje serwerów są znacznie mniejsze niż przywileje mikrojądra (chodzi np. o przywileje w dostępie do sprzętu)

# Architektura z "mikrojądrem" c.d.

- zalety:
  - łatwość programowania s.o. (koncentrujemy się na jednym serwerze)
  - łatwo wymienić serwer na nowy
  - serwery mogą działać na innych maszynach (rozproszony s.o.)
- wady:
  - spowolnienie działania: wywołanie funkcji systemowej przez program pociąga za sobą IPC między serwerami
- przykłady:
  - WinNT 4.0
    - "zmodyfikowana architektura oparta o mikrojądro"
      (serwery pracują w tzw trybie jądra - mają duże przywileje, większe od przywilejów procesów użytkowników)
  - Minix
    - szkoleniowy, unix-o podobny system operacyjny
      z książki Tannenbauma "Operating systems"

# Architektura z "mikrojądrem" c.d.

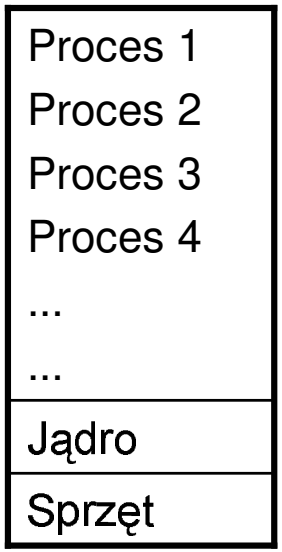
- Rysunek pokazujący architekturę s.o. Minix:



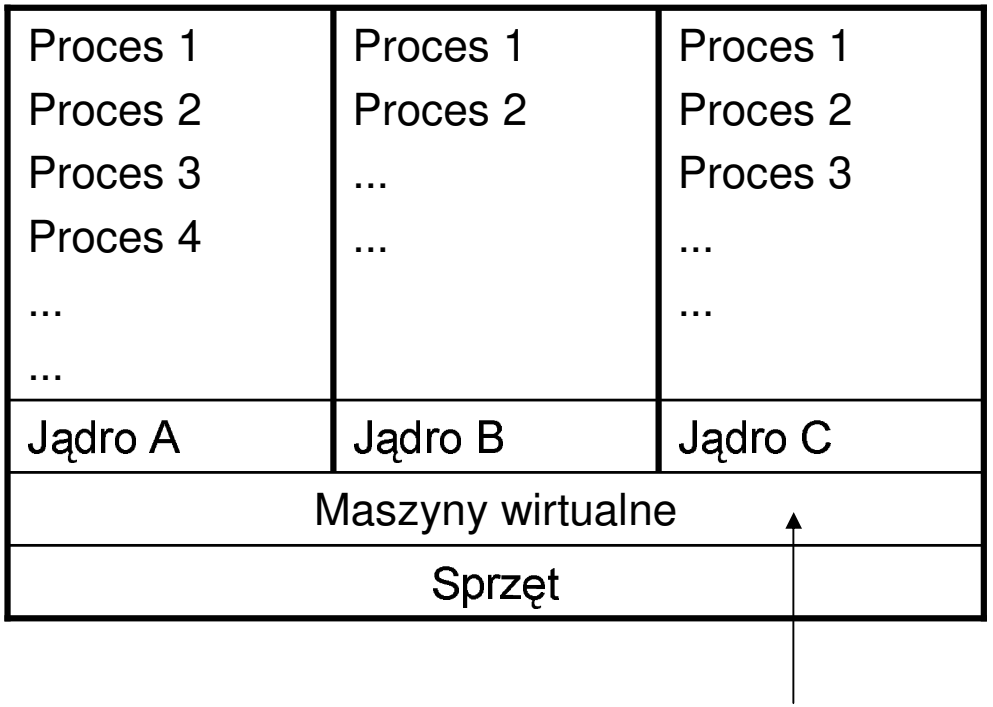
# Architektura z maszynami wirtualnymi"

- na *fizycznej* maszynie tworzy się (symuluje) kilka maszyn *logicznych* (inaczej *wirtualnych*)
- na każdej maszynie wirtualnej może działać inny s.o.

***s.o. bez maszyn wirtualnych***




***s.o. z maszynami wirtualnymi***




część s.o. która tworzy maszyny wirtualne

# Architektura z maszynami wirtualnymi" c.d.

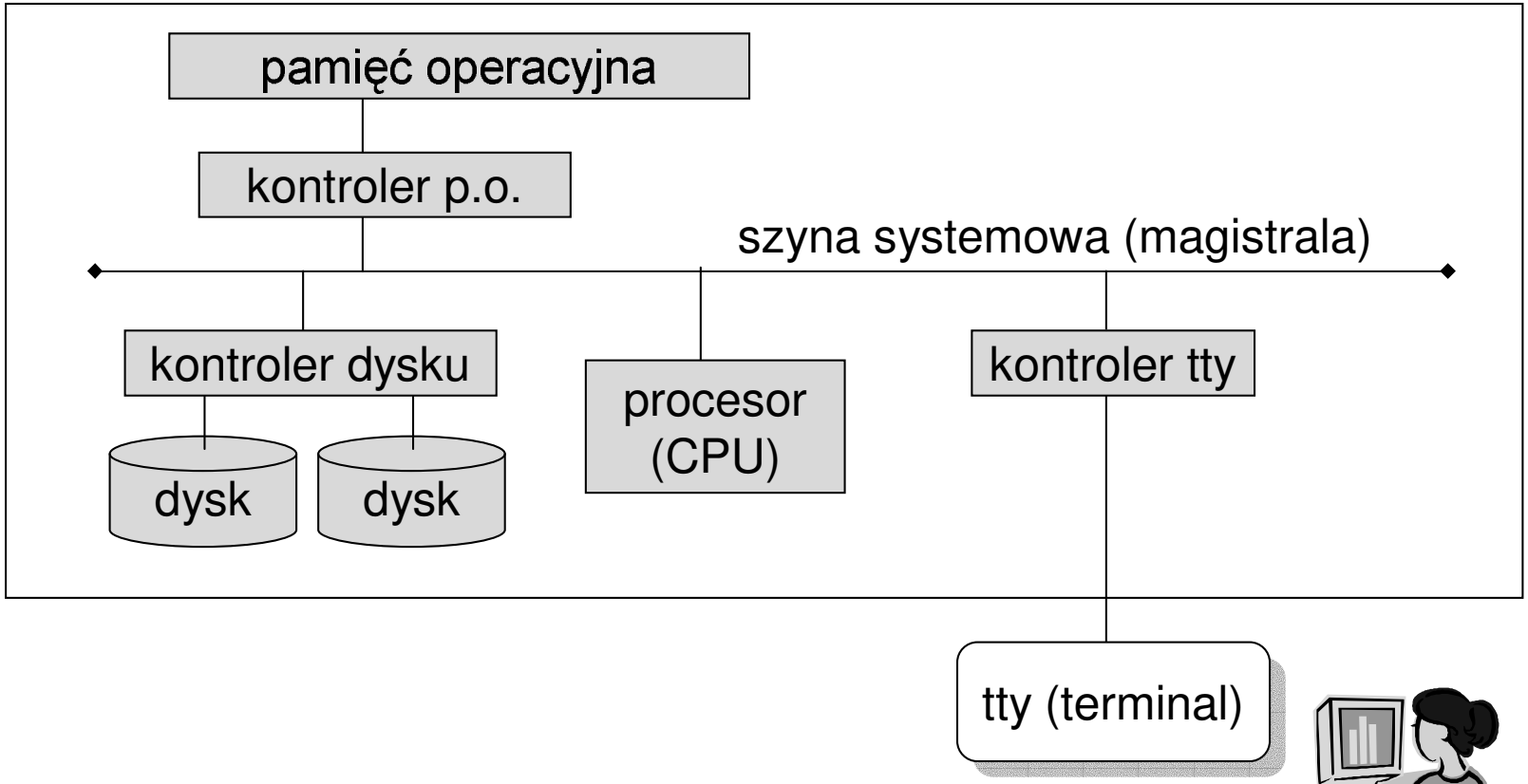
- zalety:
  - nadaje się do eksperymentów z s.o. : na maszynie wirtualnej możemy uruchamiać nowy, testowany/tworzony przez nas s.o. !
  - często w maszynach wirtualnych uruchamia się starszą wersje s.o. dzięki czemu jest możliwe uruchamianie "starych" programów !
- przykłady:
  - WinNT + tzw pudełka DOS-owe
    - w każdym pudełku DOS-owym mamy maszynę wirtualną z procesorem 8086 i załadowanym DOS-em 5.0
    - ***Uwaga***: żeby zrealizować powyższe jest potrzebne wsparcie sprzętowe: tzw "tryb wirtualny V86" procesora 80386; dzięki temu każda "pudełko" ma m.in. własną pamięć video zaczynającą się od adresu B800:0, czy własny wektor przerwań ...

# Budowa "ogólnego" s.o.

- Sprzęt komputerowy [ang. *hardware*]

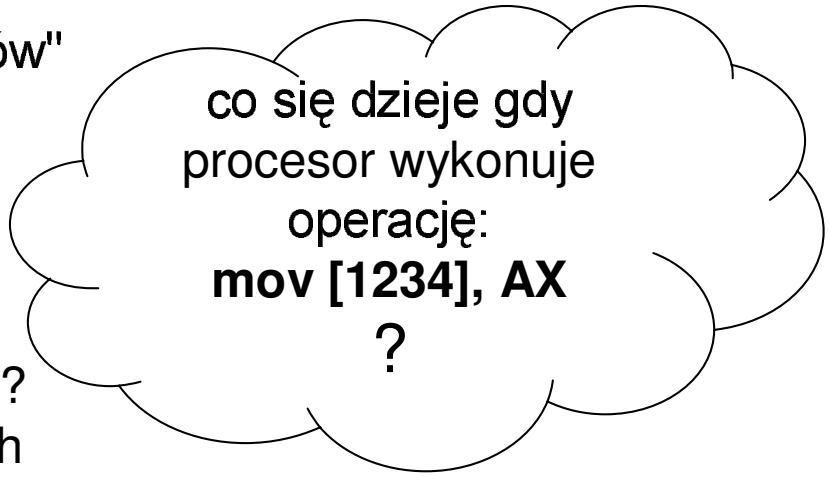
# Sprzęt komputerowy

- współczesny system komputerowy składa się z następujących części:

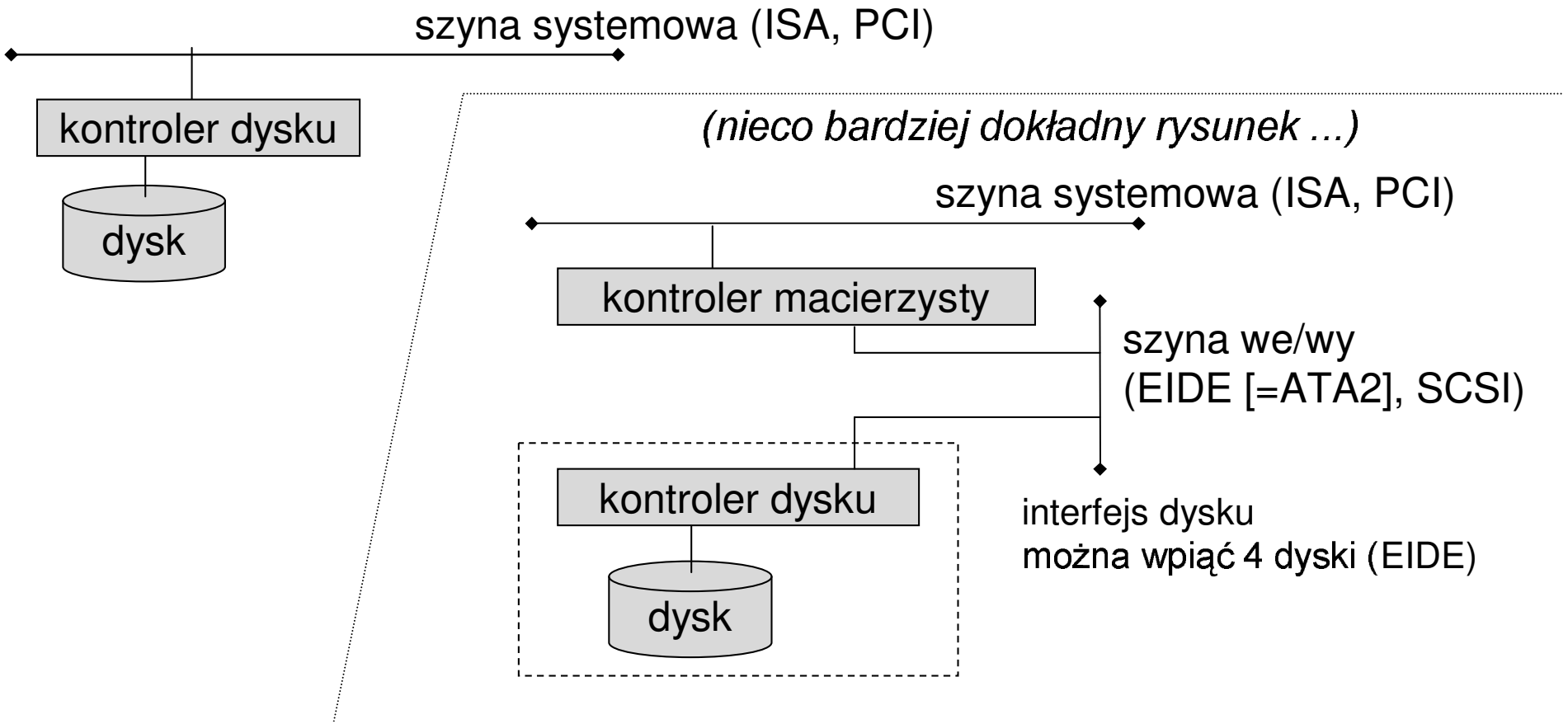


sterownik
- sprzętowy = kontroler (ang. controller)
- programowy (ang. driver); inaczej: program obsługi urządzenia część s.o. (kod)

- z czego składa się *szyna systemowa* ?
  - szyna systemowa to "wiązka przewodów"
  - szyna systemowa składa się z:
    - szyny adresowej
    - szyny danych
    - szyny sterującej
  - co to znaczy że procesor jest x-bitowy ?
    Odp: ma x-przewodów w szynie danych
    procesory firmy Intel:
    - 8086 – 16 bitowy
    - 8088 – 8 bitowy (16 bitową liczbę przesyła w 2 krokach !)
    - 80386 – 32 bitowy
    - 80386SX – 16 bitowy
    - 80486, Pentium – 32 bitowe



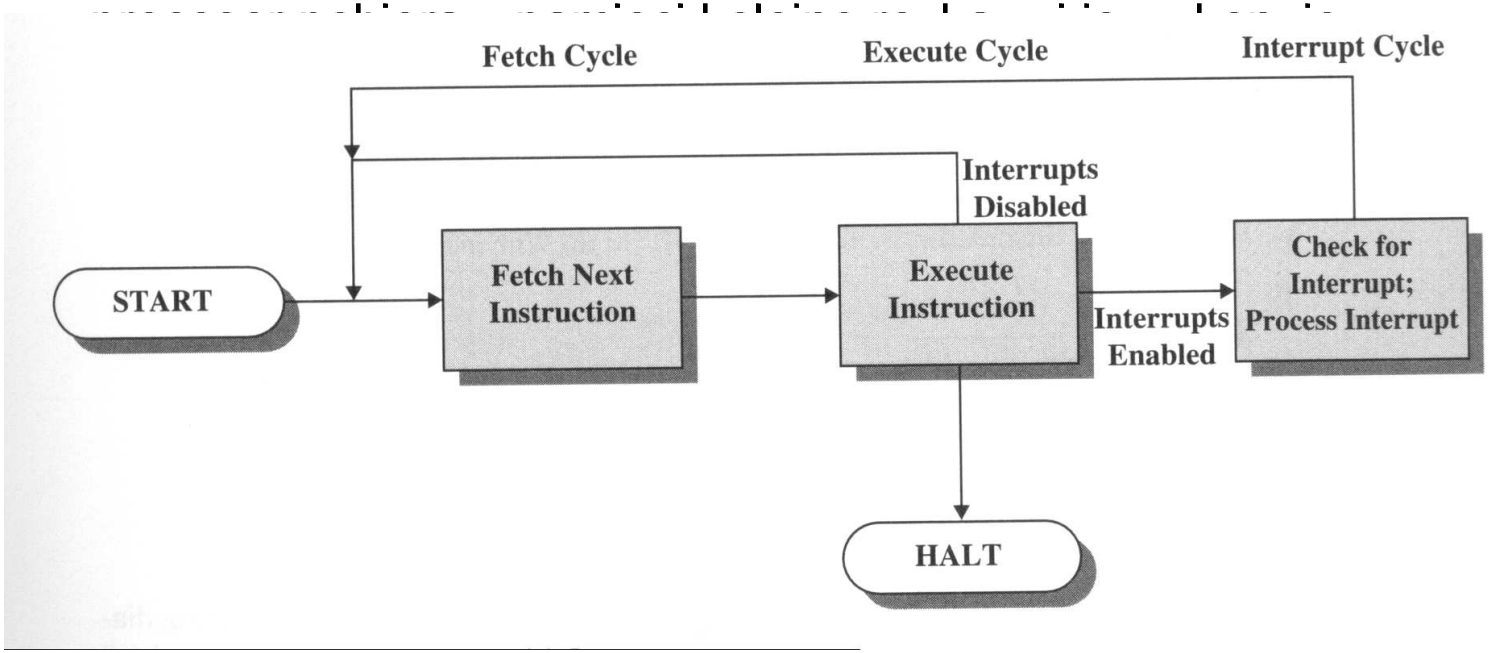
- jaka jest rola kontrolera ?
  - *nieformalnie*: procesor "rozmawia" z dyskiem poprzez szynę systemową używając języka rozkazów; kontroler tłumaczy te rozkazy na sygnały niskiego poziomu i przesyła je do dysku
  - kontroler steruje dyskiem
    - kontroler posiada "lokalną pamięć buforową" oraz "rejestry steru jące"
    - kontroler przemieszcza dane między "lok. pam. buf." a dyskiem oraz sygnalizuje zakończenie tej operacji



# Jak to wszystko działa ?

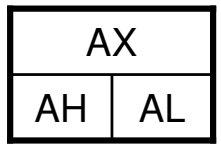
- architektura von Neumann-a:
  - dane oraz kod programu znajdują się w pamięci operacyjnej
  - przykładowy kod (intel 8086):

```
mov AX,[1234]     // AX := [1234]
          // AX, BX – 16-bitowe rejestry procesora
          // [adr] –zawartość komórki pamięci o adresie adr
mov BX,[2000]     // BX := [2000]
add AX,BX         // AX := AX + BX
mov [3000],AX     // [3000] := AX
```

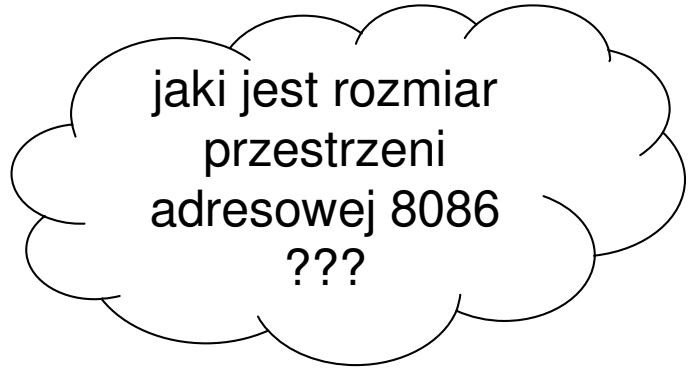


# Krótkie wprowadzenie do procesora Intel 8086

- 16-bitowe rejestry (wewnętrzne) ogólnego przeznaczenia:
  AX, BX, CX, DX
  rejestr AX jest też dostępny jako dwa 8-bitowe rejestry AL i AH (podobnie pozostałe rejestry)




- 16-bitowe rejestry segmentowe:
  CS (segment kodu),
  DS (segment danych),
  SS (segment stosu),
  ES (segment pomocniczy)




rozkaz:

```
mov [adr],AL // [adr]:=AL; adr= 16-bitów
```

oznacza zapis bajtu AL pod adres (DS*16 + adr)
rozkaz:

```
jmp adr // skok bezwzględny do adresu "adr"
```

oznacza skok do adresu (CS*16 + adr)

| Adres logiczny: | adr | rej_seg : adr |
|---|---|---|
| Adres fizyczny: | rej_seg*16 + adr | |

# Procesor Intel 8086 c.d.

- rejestry specjalne:
  SP, BP – związane z tzw stosem; SS:SP = wierzchołek stosu
  SI, DI – indeksy tablic
  F – (ang. flags) zawiera informacje o wyniku działania ostatniej operacji
- inne rozkazy:

```
mov AX,1234 // AX:=1234 - wartość 1234
mov AX,[1234] // AX:=[1234] - komórka o adresie 1234

cmp AL,123 // AL-123
jnz 1234
   // skok pod adres 1234 jeśli AL<>123; "nz" ⇔ "nie zero"

push AX // położenie na stosie wartości AX (zmienia się rej. SP)
pop BX  // zdjęcie liczby ze stosu i umieszczenie w BX

call 1234
   // skok do procedury pod adresem 1234
ret // powrót z procedury
    // (wykorzystuje się stos do przechowywania adr powrotnego)
```

- dwie przestrzenie adresowe:
  - pamięci (mov, jmp, j*, call)
  - we/wy (in, out)

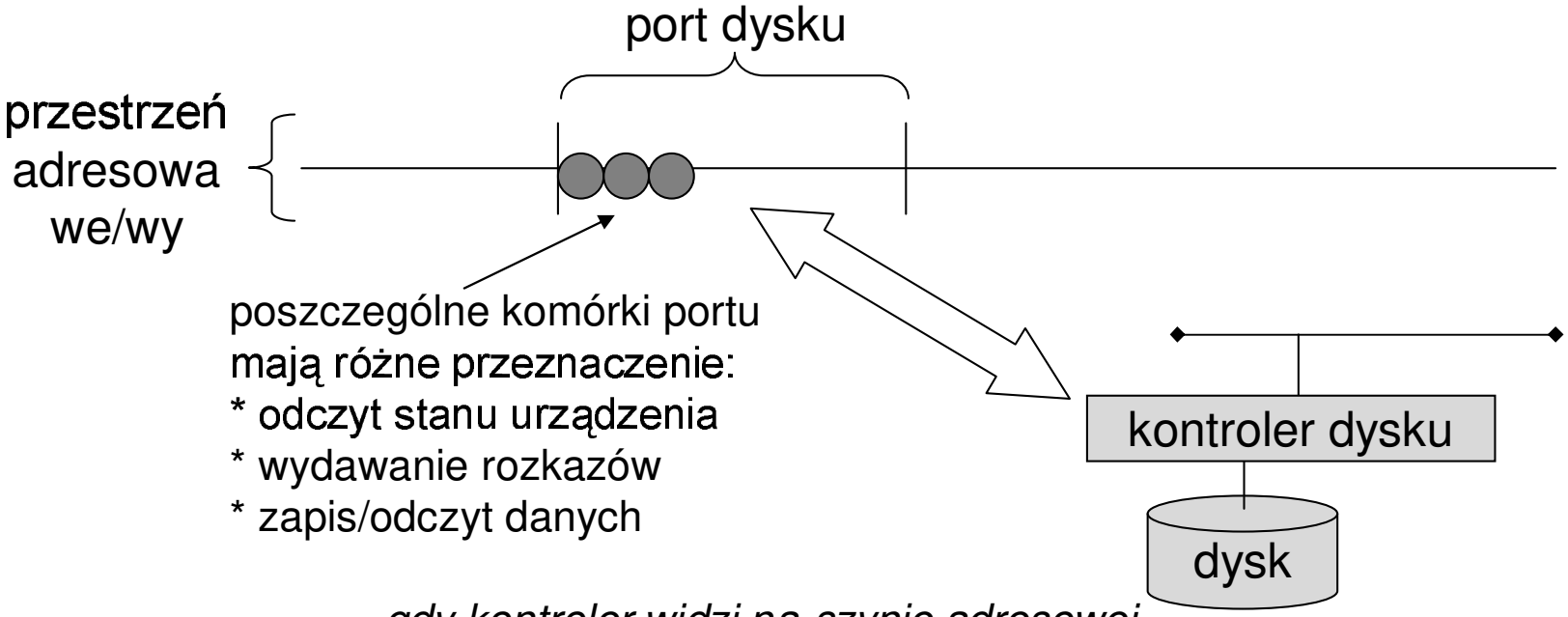
# Jak procesor (8086) rozmawia z urządzeniami we/wy (np. z dyskiem) ?

- przy pomocy rozkazów "in" i "out" oraz przestrzeni adresowej we/wy

```
in AL,123 // odczyt komórki o adresie 123; AL:=[123]
out 123,AL // zapis do komórki o adresie 123; [123]:=AL

mov DX,1234 // 16-bitowe adresy we/wy
in AL,DX // AL:=[DX]
out DX,AL // [DX]:=AL
```

- **port** = zakres prz. adr. we/wy przeznaczony dla jednego urządzenia




*gdy kontroler widzi na szynie adresowej adres należący do jego portu ...*

# Jak procesor (8086) rozmawia z urządzeniami we/wy c.d.

- niektóre urządzenia są dostępne poprzez prz. adr. pamięci operacyjnej (np. pamięć video)
- inna definicja pojęcia **port** : punkt styku między procesorem a urządzeniem we/wy

# Przerwania (ang. interrupts)

- udogodnienia sprzętowe o których do tej pory mówiliśmy nie wystarczą do zbudowania s.o. – jest jeszcze potrzebny *mechanizm przerwań ...*
- przerwanie to "sygnał" wyzwalający procedurę obsługi
- przerwania dzielimy na:
  - *sprzętowe*: generowane przez urządzenia gdy potrzebują one obsługi (np. dysk odczytał blok danych, znajduje się on w lok. pam. buf. kontrolera dysku i teraz musi być skopiowany do pam. operacyjnej)
  - *programowe*:
    - pułapki/ wyjątki (powodowane przez błędy, np. dzielenie przez 0)
    - specjalny rozkaz wymuszający przerwanie:
      int (8086)
      trap
      syscall
      (służy on głównie do uruchamiania funkcji systemowych)
  - przerwania sprzętowe i programowe są obsługiwane przez ten sam mechanizm !!!

# Przerwania c.d.

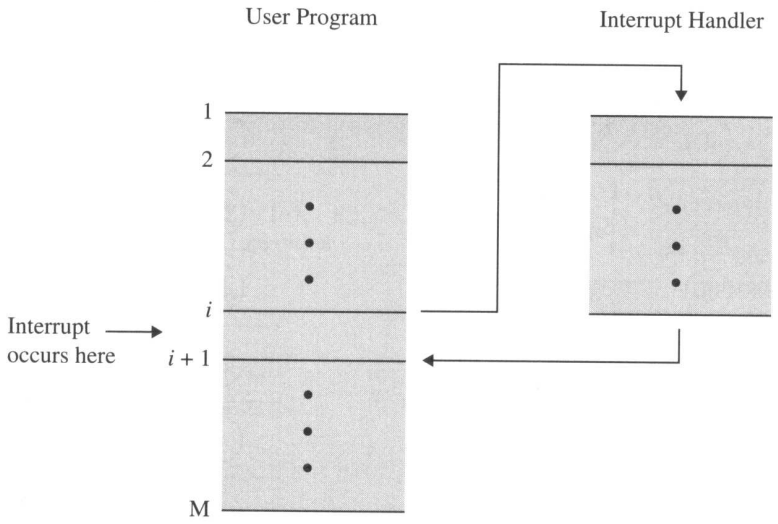



**Figure 1.6** Transfer of Control via Interrupts

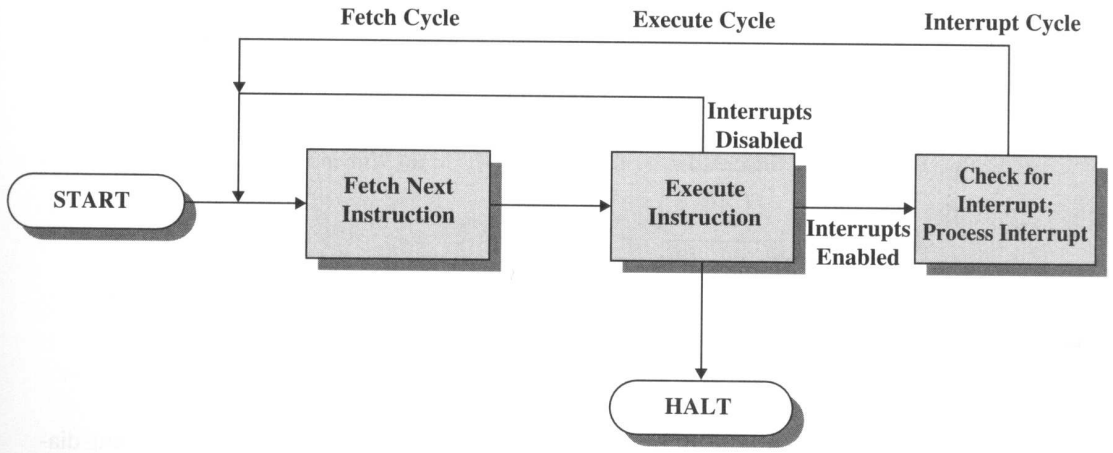



**Figure 1.7** Instruction Cycle with Interrupts

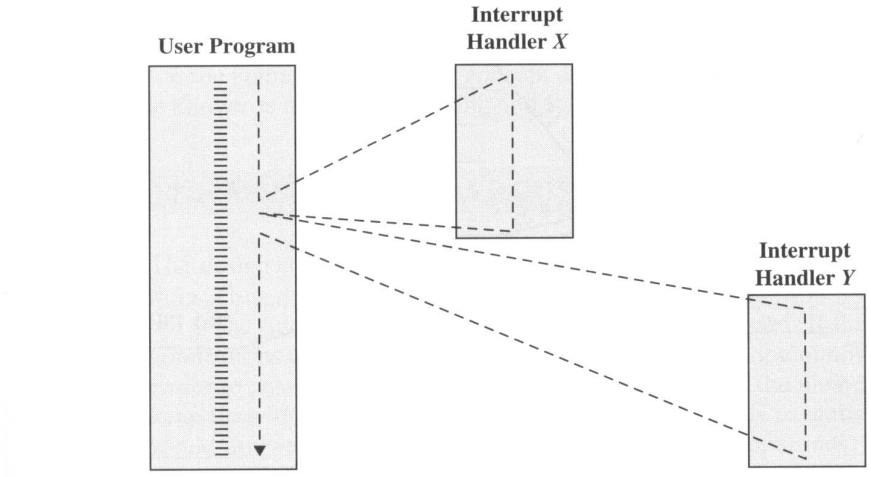
# Przerwania c.d.

- skąd wiadomo które urządzenie zgłosiło przerwanie ?
  - odpytywanie wszystkich urządzeń
    - gdy jest jedna globalna procedura obsługująca przerwania
  - numery przerwań + wektor przerwań
    - urządzenie zgłaszające przerwanie podaje jego numer
    - istnieje wektor przerwań **V** zawierający adresy procedur obsługi przerwań o poszczególnych numerach;
      gdy zostanie zgłoszone przerwanie nr "i" to jest uruchamiana procedura obsługi spod adresu **V[i]**
    - ten mechanizm jest stosowany w procesorze 8086;

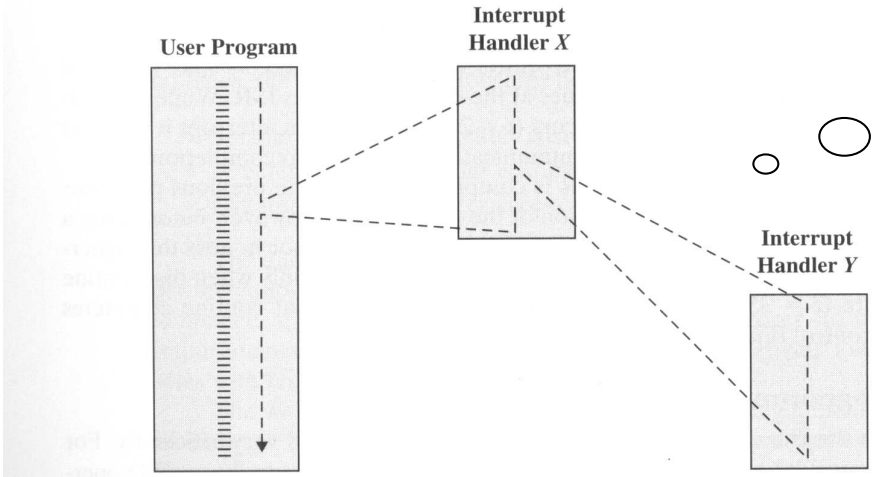
```
int nr_przerwania
```

# Przerwania c.d.

- problemy z wieloma przerwaniami: zagnieżdżanie ...

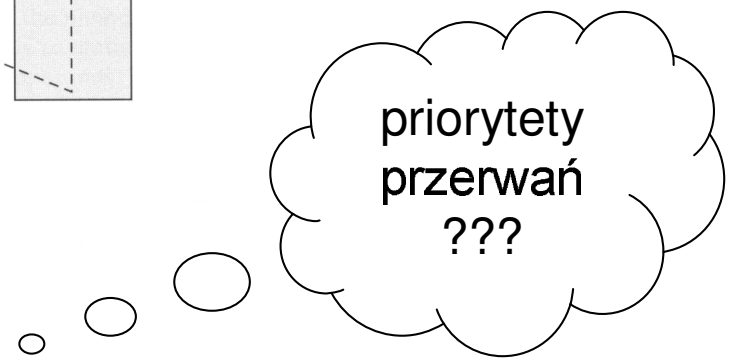



(a) Sequential interrupt processing




(b) Nested interrupt processing

**Figure 1.12** Transfer of Control with Multiple Interrupts



# Dwa sposoby wykonywania operacji we/wy

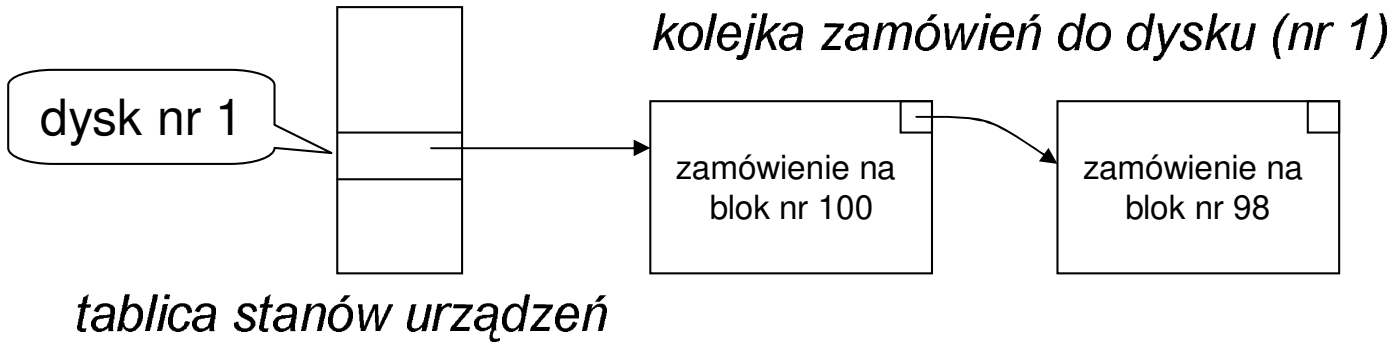
- omówimy to na przykładzie "odczytu bloku danych z dysku" ...
- sposób ***synchroniczny***
  1. zamawiamy odczyt bloku z dysku
  2. wykonujemy pętlę w której sprawdza się czy blok został odczytany (tzw *odpytywanie*)
  3. gdy stwierdzimy że blok został odczytany (jest w "lok. pam. buf." kontrolera) to przepisujemy go do pamięci operacyjnej
  4. blok dyskowy jest używany
- sposób ***asynchroniczny***
  - *zasada*: nie czeka się (jawnie) na zakończenie operacji we/wy lecz dysk informuje o tym przez przerwanie sprzętowe
  1. zamawiamy odczyt bloku z dysku
  2. "zamówienie" zostaje umieszczone w *kolejce zamówień do dysku*
  3. po zrealizowaniu naszego zamówienia dysk zgłasza przerwanie a procedura obsługi przerwania kopiuje blok do pamięci operacyjnej
  4. blok dyskowy może być używany

# Asynchroniczne wykonywanie operacji we/wy

- jaki sens ma "sposób asynchroniczny" skoro zazwyczaj :
  - zamawiamy blok dyskowy
  - używamy go

tutaj musimy poczekać
na dostarczenie bloku !

- to ma sens w wieloprogramowych s.o. !
- po złożeniu zamówienia procesor przełącza się na inny proces (a nasz proces przechodzi do stanu C*zekający*)
- procedura obsługi przerwań dyskowych nie tylko kopiuje blok do pamięci operacyjnej ale także zmienia stan naszego procesu z C*zekający* na *Gotowy* ("gotowy do działania" bo ma już potrzebny blok dyskowy)



# DMA = Direct Memory Access

- przypuśćmy że dysk zgłasza 1 przerwanie sprzętowe na każdy bajt który można odczytać (bloki dyskowe odczytuje się "po jednym bajcie")
- jeśli chcemy odczytać blok dyskowym zawierający 4096 bajtów to wymaga to 4096 przerwań sprzętowych i wywołań procedury obsługi przerwania
- obsługa przerwań jest dość kosztowna / czasochłonna
  (patrz: procedura handler z zadania domowego dla chętnych)
- *rozwiązanie problemu:*
  - kanał DMA czyli bezpośredni dostęp do pamięci operacyjnej
  - kontroler dysku ma bezpośredni dostęp do pamięci operacyjnej
    i może sam (bez pomocy procesora) przepisać lok. pam. buf. pod odpowiedni adres pamięci operacyjnej ...
- jak to działa:
  - składamy zamówienie na blok dyskowy
  - po jakimś czasie kontroler dysku i kanał DMA kopiują blok dyskowy
    do pamięci operacyjnej (pod odpowiedni adres pamięci)
  - zgłaszane jest przerwanie sprzętowe (gdy blok jest już w pamięci operacyjnej !)
- DMA spowalnia "trochę" działanie procesora (wspólna szyna adresowa)
- PIO = Programmed Input Output
  - nazwa trybu pracy w którym procedura obsługi przerwania kopiuje blok do pamięci operacyjnej (innymi słowy: procesor kopiuje dane z dysku do pamięci oper. a nie kanał DMA)