Wydział Elektryczny
Katedra Elektrotechniki, Energoelektroniki i Elektroenergetyki

Materiały do wykładu z przedmiotu:

Informatyka

Kod: EDS1B1007

WYKŁAD NR 7

Opracował: dr inż. Jarosław Forenc

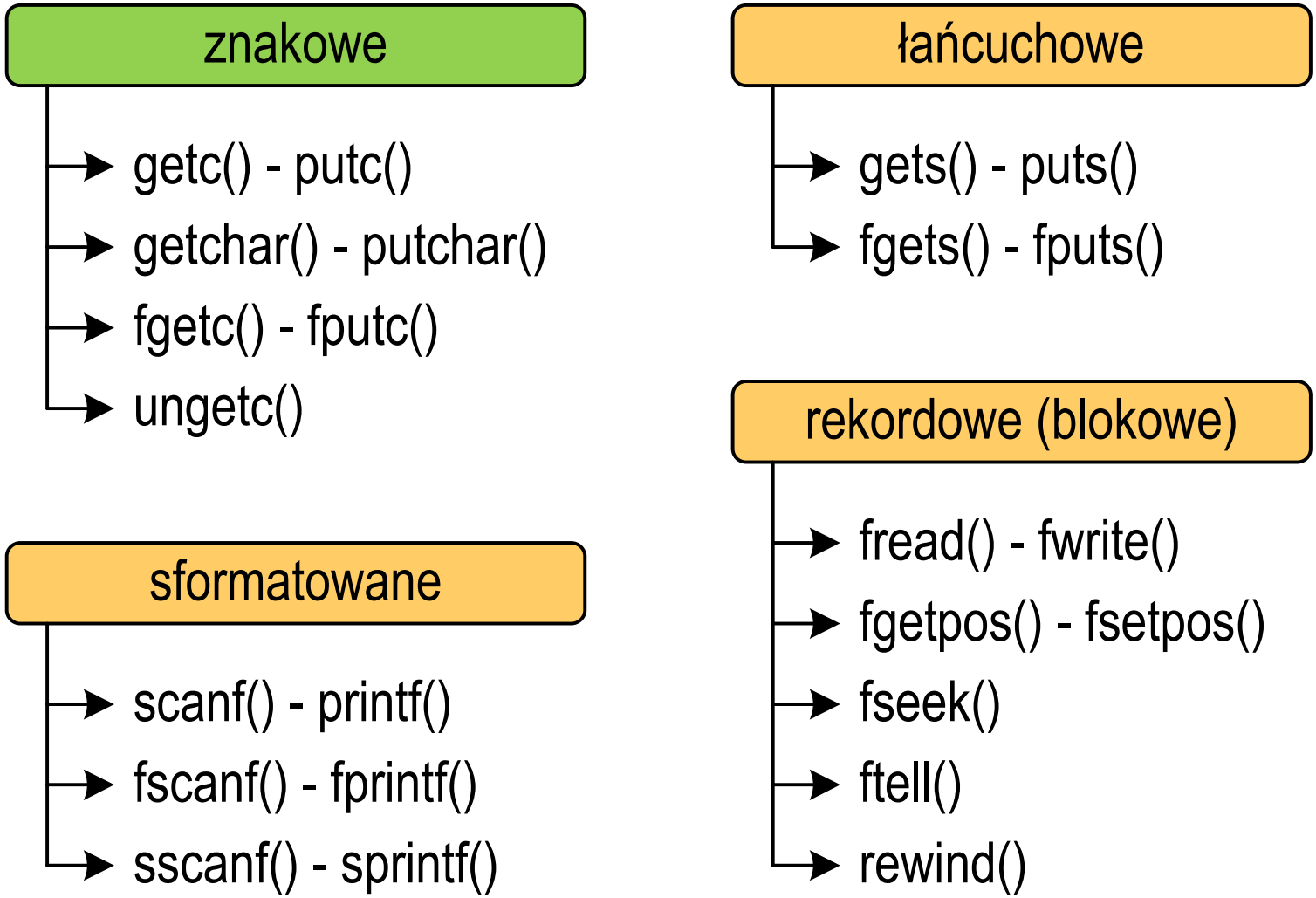
Białystok 2022

Materiały zostały opracowane w ramach projektu „PB2020 - Zintegrowany Program Rozwoju Politechniki Białostockiej” realizowanego w ramach Działania 3.5 Programu Operacyjnego Wiedza, Edukacja, Rozwój 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Społecznego.

Plan wykładu nr 7

- Operacje na plikach
  - operacje znakowe i łańcuchowe
  - operacje sformatowane i rekordowe (blokowe)
- System operacyjny
  - definicje systemu operacyjnego
- Zarządzanie procesami
  - definicja procesu, dwu- i pięciostanowy model procesu
- Zarządzanie dyskowymi operacjami we-wy
  - struktura dysku twardego (MBR, GPT)
  - systemy plików (FAT, NTFS, ext2)
- Zarządzanie pamięcią operacyjną
  - partycjonowanie, stronicowanie, segmentacja, pamięć wirtualna

Znakowe operacje wejścia-wyjścia



Znakowe operacje wejścia-wyjścia

GETC stdio.h

```
int getc(FILE *fp);
```

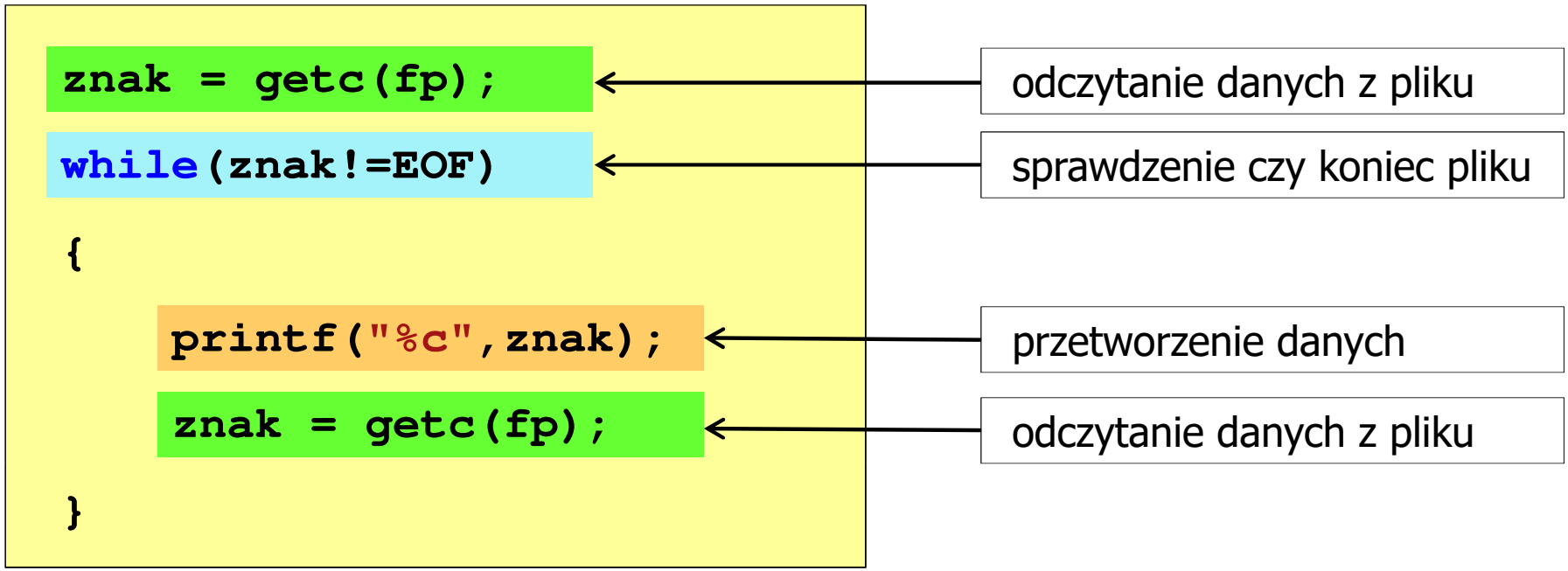
- Pobiera jeden znak z aktualnej pozycji otwartego strumienia fp i uaktualnia pozycję
- Zmienna fp powinna wskazywać strukturę FILE reprezentującą strumień skojarzony z otwartym plikiem lub jeden ze standardowo otwartych strumieni (np. stdin)
- Jeśli wykonanie zakończyło się poprawnie, to funkcja zwraca wartość całkowitą kodu wczytanego znaku (typ int)
- Jeśli wystąpił błąd lub przeczytany został znacznik końca pliku, to funkcja zwraca wartość EOF
- funkcja fgetc() działa tak samo jak getc()

Przykład: wyświetlenie pliku tekstowego

```
#include <stdio.h>

int main(void)
{
    FILE  *fp;
    int   znak;

    fp = fopen("test.txt","r");

    znak = getc(fp);
    while(znak!=EOF)
    {
        printf("%c",znak);
        znak = getc(fp);
    }

    fclose(fp);
    return 0;
}
```

Schemat przetwarzania pliku

- Typowy schemat odczytywania danych z pliku



Przykład: wyświetlenie pliku tekstowego

- Odczytanie i wyświetlenie zawartości pliku tekstowego

```
znak = getc(fp);
while(znak!=EOF)
{
    printf("%c",znak);
    znak = getc(fp);
}
```

można zapisać w krótszej postaci:

```
while((znak=getc(fp))!=EOF)
    printf("%c",znak);
```

Znakowe operacje wejścia-wyjścia

PUTC stdio.h

```
int putc(int znak, FILE *fp);
```

- Wpisuje znak do otwartego strumienia reprezentowanego przez argument fp
- Zmienna fp powinna wskazywać strukturę FILE reprezentującą strumień skojarzony z otwartym plikiem lub jeden ze standardowo otwartych strumieni (np. stdout)
- Jeśli wykonanie zakończyło się poprawnie, to funkcja zwraca wypisany znak
- Jeśli wystąpił błąd, to funkcja zwraca wartość EOF
- funkcja fputc() działa tak samo jak putc()

Przykład: zapisanie alfabetu do pliku tekstowego

```
#include <stdio.h>

int main(void)
{
    FILE *fp = fopen("alfabet.txt","w");

    for (int i='A'; i<='Z'; i++)
        putc(i,fp);

    fclose(fp);

    return 0;
}
```

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

- Stosując strumień stdout można wyświetlić alfabet na ekranie

```
for (int i='A'; i<='Z'; i++)
    putc(i,stdout);
```

Znakowe operacje wejścia-wyjścia

GETCHAR stdio.h

```
int getchar(void);
```

- Pobiera znak ze strumienia stdin (klawiatura)
- Jeśli wykonanie zakończyło się poprawnie, to funkcja zwraca przeczytany znak (typ int)
- Jeśli wystąpił błąd albo został przeczytany znacznik końca pliku, to funkcja zwraca wartość EOF

```
int znak;

znak = getchar();
printf("%c",znak);
```

Znakowe operacje wejścia-wyjścia

PUTCHAR stdio.h

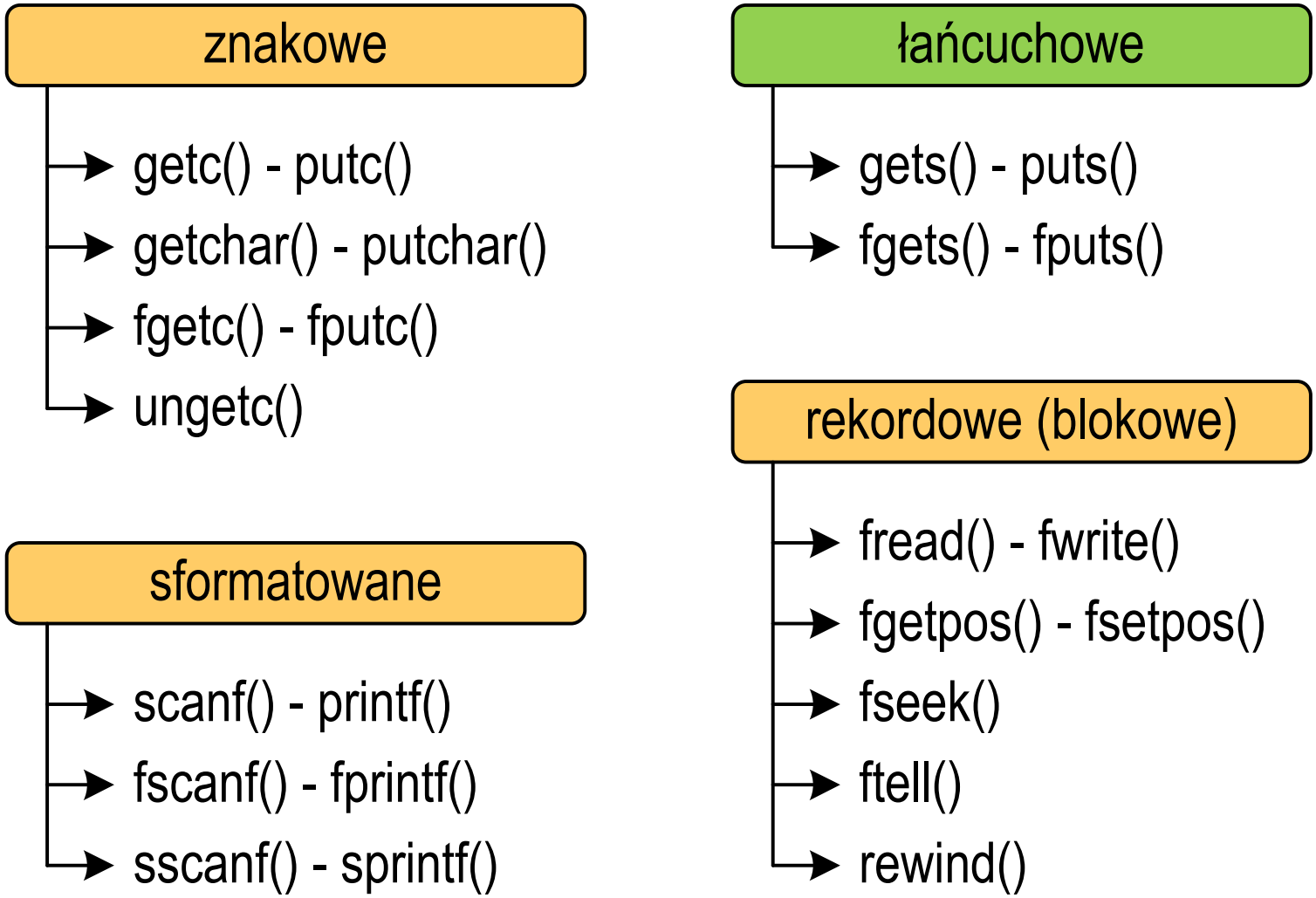
```
int putchar(int znak);
```

- Wpisuje znak do strumienia stdout (standardowo ekran)
- Jeśli wykonanie zakończyło się poprawnie, to funkcja zwraca wypisany znak
- Jeśli wystąpił błąd, to funkcja zwraca wartość EOF

```
for (int i='a'; i<='z'; i++)
    putchar(i);
```

```
abcdefghijklmnopqrstuvwxyz
```

Łańcuchowe operacje wejścia-wyjścia



Łańcuchowe operacje wejścia-wyjścia

GETS stdio.h

```
char* gets(char *buf);
```

- Pobiera do bufora pamięci wskazywanego przez argument buf linię znaków ze strumienia stdin (standardowo klawiatura)
- Wczytywanie jest kończone po napotkaniu znacznika nowej linii '\n', który zastępowany jest znakiem końca łańcucha '\0'
- Funkcja gets() umożliwia wczytanie łańcucha znaków zawierającego spacje i tabulatory
- Jeśli wykonanie zakończyło się poprawnie, to funkcja zwraca wskazanie do łańcucha buf
- Jeśli wystąpił błąd lub podczas wczytywania został napotkany znacznik końca pliku, to funkcja zwraca wartość EOF

Łańcuchowe operacje wejścia-wyjścia

PUTS stdio.h

```
int puts(const char *buf);
```

- Wpisuje łańcuch buf do strumienia stdout (standardowo ekran), zastępując znak '\0' znakiem '\n'
- Jeśli wykonanie zakończyło się poprawnie, to funkcja zwraca ostatni wypisany znak
- Jeśli wystąpił błąd, to funkcja zwraca wartość EOF

```
char tablica[80];

gets(tablica);
puts(tablica);
```

Łańcuchowe operacje wejścia-wyjścia

FGETS stdio.h

```
char* fgets(char *buf, int max, FILE *fp);
```

- Pobiera znaki z otwartego strumienia reprezentowanego przez fp i zapisuje je do bufora pamięci wskazanego przez buf
- Pobieranie znaków jest przerywane po napotkaniu znacznika końca linii '\n' lub odczytaniu max-1 znaków
- Po ostatnim przeczytanym znaku wstawia do bufora buf znak '\0'
- Jeśli wykonanie zakończyło się poprawnie, to funkcja zwraca wskazanie do łańcucha buf
- Jeśli wystąpił błąd lub napotkano znacznik końca pliku, to funkcja zwraca wartość NULL

Łańcuchowe operacje wejścia-wyjścia

FPUTS stdio.h

```
int fputs(const char *buf, FILE *fp);
```

- Wpisuje łańcuch buf do strumienia fp, nie dołącza znaku końca wiersza '\n'
- Jeśli wykonanie zakończyło się poprawnie, to funkcja zwraca ostatni wypisany znak
- Jeśli wystąpił błąd, to funkcja zwraca wartość EOF

Przykład: wyświetlenie pliku tekstowego

```
#include <stdio.h>

int main(void)
{
    FILE  *fp;
    char  buf[15];

    fp = fopen("test.txt","r");

    while (fgets(buf,15,fp)!=NULL)
        fputs(buf,stdout);

    fclose(fp);

    return 0;
}
```

Przykład: wyświetlenie pliku tekstowego

- Zawartość pliku test.txt

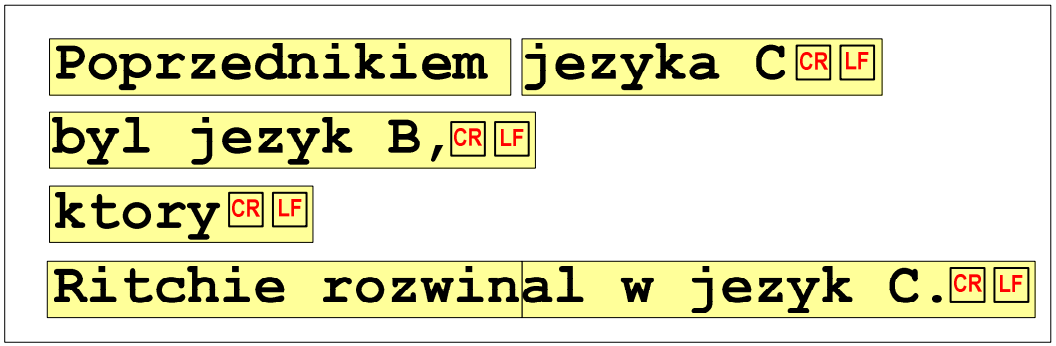
```
Poprzednikiem jezyka C<CR><LF>
byl jezyk B,<CR><LF>
ktory<CR><LF>
Ritchie rozwinal w jezyk C.<CR><LF>
```

- Kolejne wywołania funkcji fgets(buf,15,fp);

```
Poprzednikiem | jezyka C<CR><LF>
byl jezyk B,<CR><LF>
ktory<CR><LF>
Ritchie rozwin | al w jezyk C.<CR><LF>
```

Przykład: wyświetlenie pliku tekstowego

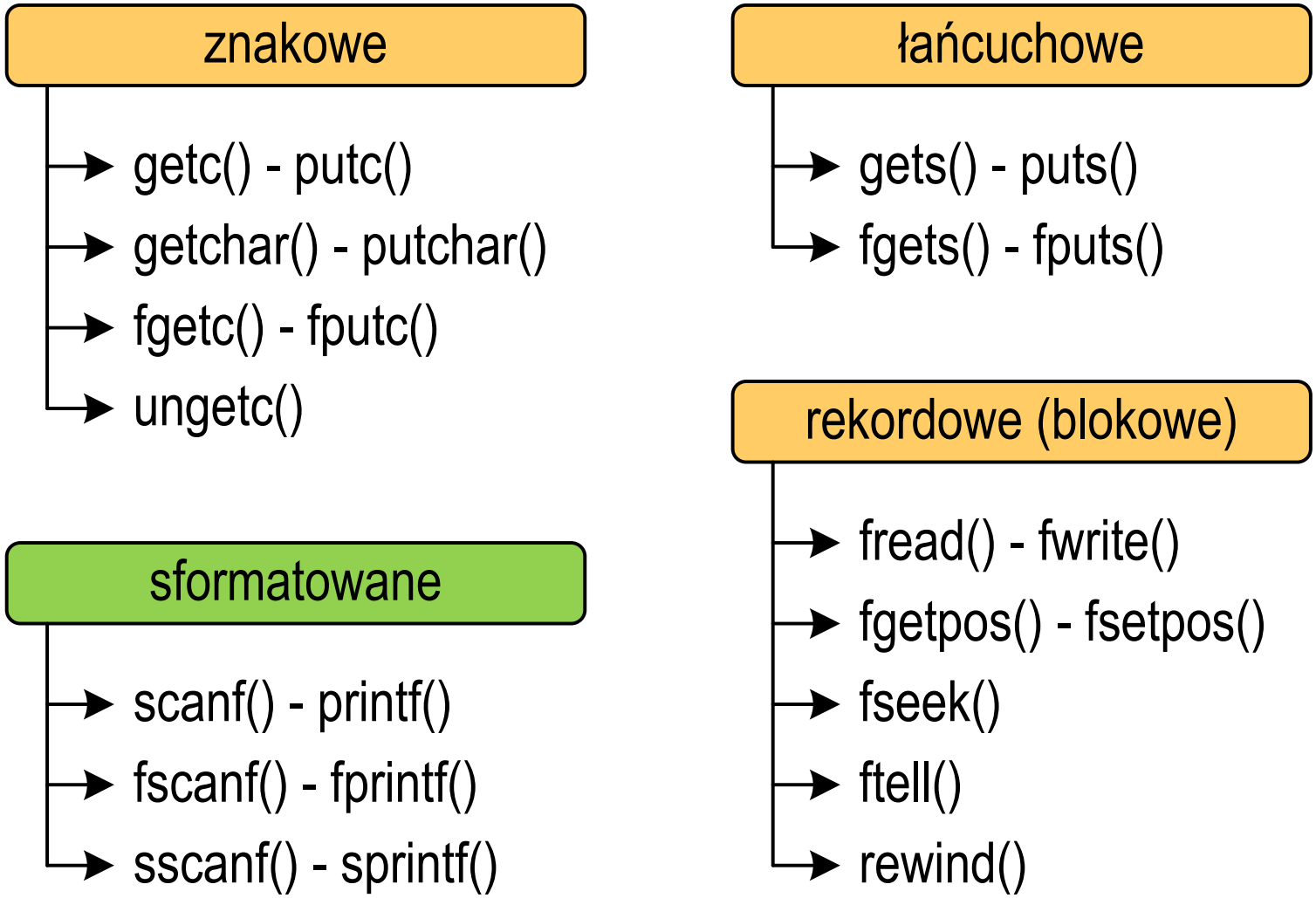
- Kolejne wywołania funkcji fgets(buf,15,fp); i zawartość tablicy buf



P	o	p	r	z	e	d	n	i	k	i	e	m	\0
j	e	z	y	k	a		C	LF	\0				
b	y	l		j	e	z	y	k		B	,	LF	\0
k	t	o	r	y	LF	\0							
R	i	t	c	h	i	e		r	o	z	w	i	n	\0
a	l		w		j	e	z	y	k		C	.	LF	\0

LF = \n

Sformatowane operacje wejścia-wyjścia



Sformatowane operacje wejścia-wyjścia

SCANF stdio.h

```
int scanf(const char *format,...);
```

- Czyta dane ze strumienia stdin (klawiatura)

FSCANF stdio.h

```
int fscanf(FILE *fp, const char *format,...);
```

- Czyta dane z otwartego strumienia (pliku) fp

SSCANF stdio.h

```
int sscanf(char *buf, const char *format,...);
```

- Czyta dane z bufora pamięci wskazywanego przez buf

Sformatowane operacje wejścia-wyjścia

PRINTF stdio.h

```
int printf(const char *format,...);
```

- Wyprowadza dane do strumienia stdout (ekran)

FPRINTF stdio.h

```
int fprintf(FILE *fp, const char *format,...);
```

- Wyprowadza dane do otwartego strumienia (pliku) fp

SPRINTF stdio.h

```
int sprintf(char *buf, const char *format,...);
```

- Wyprowadza dane do bufora pamięci wskazywanego przez buf

Przykład: zapisanie liczb do pliku tekstowego

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
   FILE  *fp; float x; int i;

   srand((unsigned int)time(NULL));
   fp = fopen("liczby.txt","w");
   for (i=0; i<10; i++)
   {
      x = (float)rand()/RAND_MAX*100;
      fprintf(fp,"%f\n",x);
   }
   fclose(fp);

   return 0;
}
```

```
3.830073
70.848717
99.322487
19.812616
7.132175
49.134800
10.238960
18.668173
8.914456
69.258705
```

Przykład: zapisanie danych do pliku tekstowego

```
#include <stdio.h>

int main(void)
{
    FILE  *fp;
    int   wiek = 21;
    float wzrost = 1.78f;
    char  imie[10] = "Jan", nazw[10] = "Kowalski";

    fp = fopen("dane.txt","w");
    fprintf(fp,"Imie:      %s\n",imie);
    fprintf(fp,"Nazwisko:  %s\n",nazw);
    fprintf(fp,"Wiek:      %d [lat]\n",wiek);
    fprintf(fp,"Wzrost:    %.2f [m]\n",wzrost);
    fclose(fp);

    return 0;
}
```

```
Imie:      Jan
Nazwisko:  Kowalski
Wiek:      21 [lat]
Wzrost:    1.78 [m]
```

Obsługa błędów wejścia-wyjścia

FEOF stdio.h

```
int feof(FILE *fp);
```

- Sprawdza, czy podczas ostatniej operacji wejścia dotyczącej strumienia fp został osiągnięty koniec pliku
- Zwraca wartość różną od zera, jeśli podczas ostatniej operacji wejścia został wykryty koniec pliku, w przeciwnym razie zwraca wartość 0 (zero)

Przykład: odczytanie danych z pliku tekstowego

- Odczytanie danych różnych typów z pliku tekstowego

```
Nowak Grzegorz 15-12-2000
Kowalski Wojciech 03-05-1997
Jankowska Anna 23-05-1995
Mazur Krzysztof 14-01-1990
Krawczyk Monika 03-11-1995
Piotrowska Maja 12-06-1998
Dudek Piotr 31-12-1996
Pawlak Julia 01-01-1997
```

```
Grzegorz      Nowak          wiek: 22
Wojciech      Kowalski       wiek: 25
Anna          Jankowska      wiek: 27
Krzysztof     Mazur          wiek: 32
Monika        Krawczyk       wiek: 27
Maja          Piotrowska     wiek: 24
Piotr         Dudek          wiek: 26
Julia         Pawlak         wiek: 25
```

Przykład: odczytanie danych z pliku tekstowego

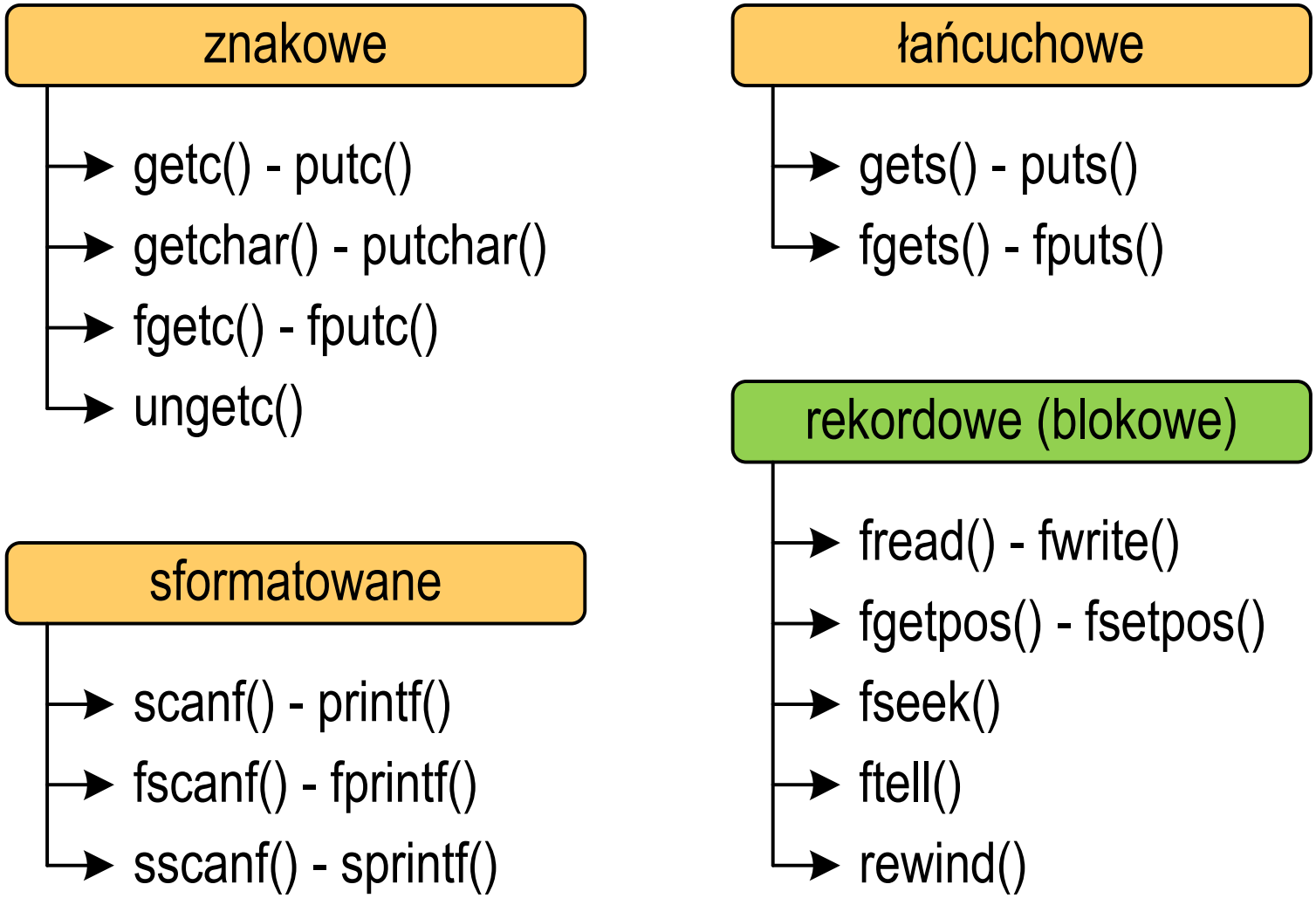
```
#include <stdio.h>

int main()
{
   FILE *fp;
   char naz[20], im[20];
   int d, m, r;

   fp = fopen("osoby.txt","r");
   fscanf(fp,"%s %s %d-%d-%d",naz,im,&d,&m,&r);
   while(!feof(fp))
   {
      printf("%-12s %-12s wiek: %d\n",im,naz,2022-r);
      fscanf(fp,"%s %s %d-%d-%d",naz,im,&d,&m,&r);
   }
   fclose(fp);

   return 0;
}
```

Przykład: odczytanie danych z pliku tekstowego

```
Grzegorz    Nowak       wiek: 22
Wojciech    Kowalski    wiek: 25
Anna        Jankowska   wiek: 27
Krzysztof   Mazur       wiek: 32
Monika      Krawczyk    wiek: 27
Maja        Piotrowska  wiek: 24
Piotr       Dudek       wiek: 26
Julia       Pawlak      wiek: 25
```

```c
#include <stdio.h>

int main()
{
   FILE *fp;
   char naz[20], im[20];
   int d, m, r;

   fp = fopen("osoby.txt","r");
   fscanf(fp,"%s %s %d-%d-%d",naz,im,&d,&m,&r);
   while(!feof(fp))
   {
      printf("%-12s %-12s wiek: %d\n",im,naz,2022-r);
      fscanf(fp,"%s %s %d-%d-%d",naz,im,&d,&m,&r);
   }
   fclose(fp);

   return 0;
}
```

Rekordowe (blokowe) operacje wejścia-wyjścia



Rekordowe (blokowe) operacje wejścia-wyjścia

FWRITE stdio.h

```
size_t fwrite(const void *p, size_t s, size_t n,
              FILE *fp);
```

- Zapisuje n elementów o rozmiarze s bajtów każdy, do pliku wskazywanego przez fp, biorąc dane z obszaru pamięci wskazywanego przez p
- Zwraca liczbę zapisanych elementów - jeśli jest ona różna od n, to wystąpił błąd zapisu (brak miejsca na dysku lub dysk zabezpieczony przed zapisem)

Przykład: zapisanie danych do pliku binarnego

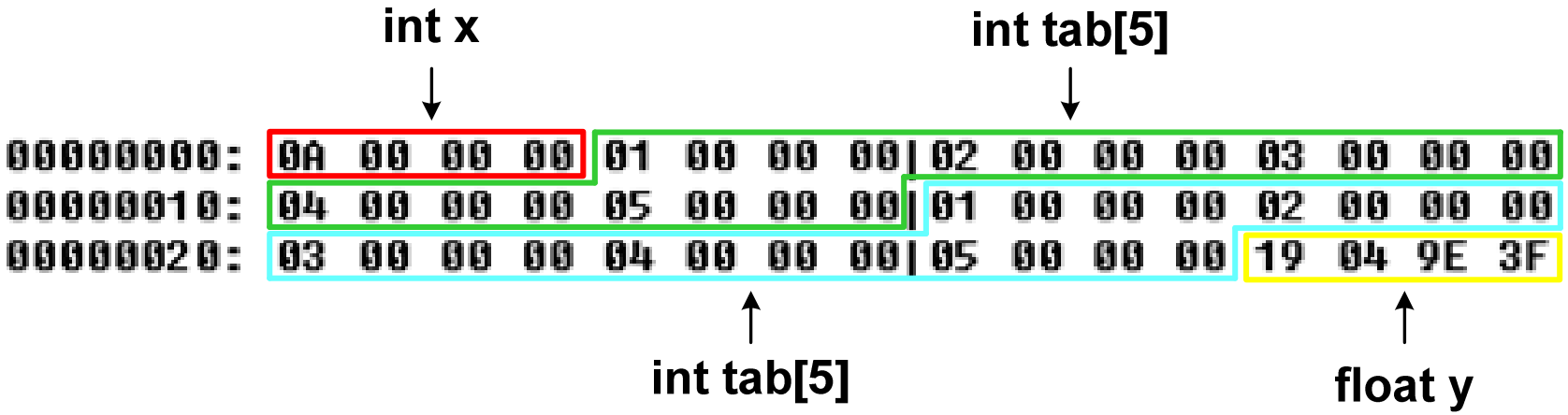
```
#include <stdio.h>

int main(void)
{
    FILE *fp;
    int   x = 10, tab[5] = {1,2,3,4,5};
    float y = 1.2345f;

    fp = fopen("dane.dat","wb");
    fwrite(&x,sizeof(int),1,fp);
    fwrite(tab,sizeof(int),5,fp);
    fwrite(tab,sizeof(tab),1,fp);
    fwrite(&y,sizeof(float),1,fp);
    fclose(fp);

    return 0;
}
```

Przykład: zapisanie danych do pliku binarnego

- Czterokrotne wywołanie funkcji fwrite()

```
fwrite(&x,sizeof(int),1,fp);          // int x = 10;
fwrite(tab,sizeof(int),5,fp);         // int tab[5] = {1,2,3,4,5};
fwrite(tab,sizeof(tab),1,fp);         // int tab[5] = {1,2,3,4,5};
fwrite(&y,sizeof(float),1,fp);        // float y = 1.2345;
```

spowoduje zapisanie do pliku 48 bajtów:



Rekordowe (blokowe) operacje wejścia-wyjścia

FREAD stdio.h

```
size_t fread(void *p, size_t s, size_t n,
             FILE *fp);
```

- Pobiera n elementów o rozmiarze s bajtów każdy, z pliku wskazywanego przez fp i umieszcza odczytane dane w obszarze pamięci wskazywanym przez p
- Zwraca liczbę odczytanych elementów - w przypadku gdy liczba ta jest różna od n, to wystąpił błąd końca strumienia (w pliku było mniej elementów niż podana wartość argumentu n)

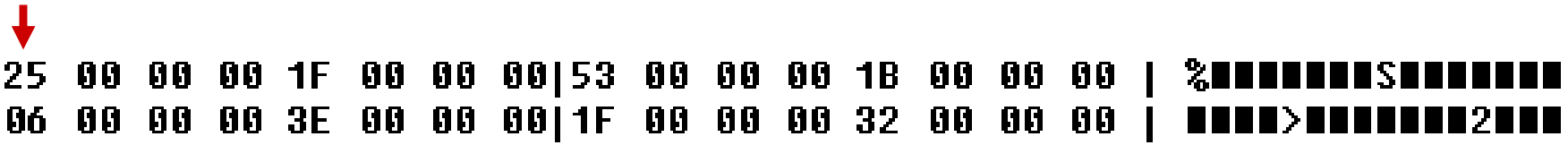
Przykład: odczytanie liczb z pliku binarnego

```c
#include <stdio.h>

int main(void)
{
    FILE *fp;
    int x, ile = 0;

    fp = fopen("liczby.dat","rb");
    fread(&x,sizeof(int),1,fp);
    while (!feof(fp))
    {
        ile++; printf("%d\n",x);
        fread(&x,sizeof(int),1,fp);
    }
    fclose(fp);
    printf("Odczytano: %d liczb\n",ile);
    return 0;
}
```

```
37
31
83
27
6
62
31
50
Odczytano: 8 liczb
```

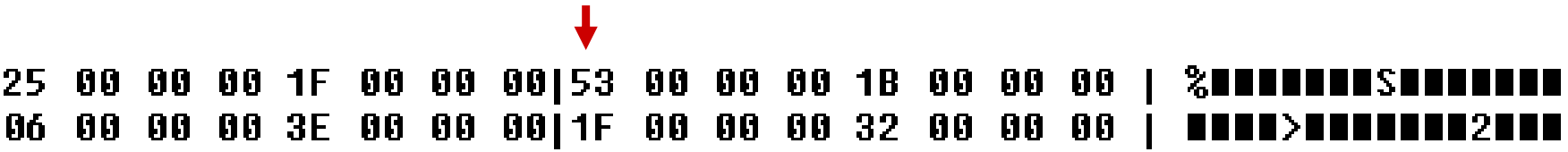
Przykład: odczytanie liczb z pliku binarnego

- Po otwarciu pliku wskaźnik pozycji pliku pokazuje na jego początek



- Po odczytaniu jednej liczby: fread(&x,sizeof(int),1,plik);
  wskaźnik jest automatycznie przesuwany o sizeof(int) bajtów

- Po odczytaniu kolejnej liczby: fread(&x,sizeof(int),1,plik);
  wskaźnik jest ponownie przesuwany o sizeof(int) bajtów



- Plik binarny zawiera liczby: 37 31 83 27 6 62 31 50

Rekordowe (blokowe) operacje wejścia-wyjścia

REWIND stdio.h

```
void rewind(FILE *fp);
```

- Ustawia wskaźnik pozycji w pliku wskazywanym przez fp na początek pliku

FTELL stdio.h

```
long int ftell(FILE *fp);
```

- Zwraca bieżące położeniu w pliku wskazywanym przez fp (liczbę bajtów od początku pliku)

Rekordowe (blokowe) operacje wejścia-wyjścia

FSEEK stdio.h

```
int fseek(FILE *fp, long int offset, int mode);
```

- Pozwala przejść bezpośrednio do dowolnego bajtu w pliku wskazywanym przez fp
- offset określa wielkość przejścia w bajtach, zaś mode - punkt początkowy, względem którego określane jest przejście (SEEK_SET - początek pliku, SEEK_CUR - bieżąca pozycja, SEEK_END - koniec pliku)
- Gdy wywołanie jest poprawne, to funkcja zwraca wartość 0 gdy wystąpił błąd (np. próba przekroczenia granic pliku), to funkcja zwraca wartość -1

Przykład: odczytanie liczby o podanym numerze

```
#include <stdio.h>

int main(void)
{
    FILE  *fp;
    int   x, nr;

    fp = fopen("dane.dat","rb");
    printf("Nr: "); scanf("%d",&nr);
    while (fseek(fp,(nr-1)*sizeof(int),SEEK_SET)==0)
    {
        fread(&x,sizeof(int),1,fp);
        printf("Liczba: %d\n",x);
        printf("Nr: "); scanf("%d",&nr);
    }
    printf("Koniec!\n");
    fclose(fp);
    return 0;
}
```

```
7 3 3 0 3 9 6 4 1 8
6 0 4 5 4 9 4 5 4 5
9 9 8 0 0 5 3 5 1 0
```

```
Nr: 6
Liczba: 9
Nr: 14
Liczba: 5
Nr: 29
Liczba: 1
Nr: -1
Koniec!
```

Rekordowe (blokowe) operacje wejścia-wyjścia

FGETPOS stdio.h

```
int fgetpos(FILE *fp, fpos_t *pos);
```

- Zapamiętuję pod zminną pos bieżące położenie w pliku wskazywanym przez fp; zwraca 0, gdy wywołania jest poprawne i wartość niezerową, gdy wystąpił błąd

FSETPOS stdio.h

```
int fsetpos(FILE *fp, const fpos_t *pos);
```

- Przechodzi do położenia pos w pliku wskazywanym przez fp; zwraca 0, gdy wywołania jest poprawne i wartość niezerową, gdy wystąpił błąd

System operacyjny - definicja

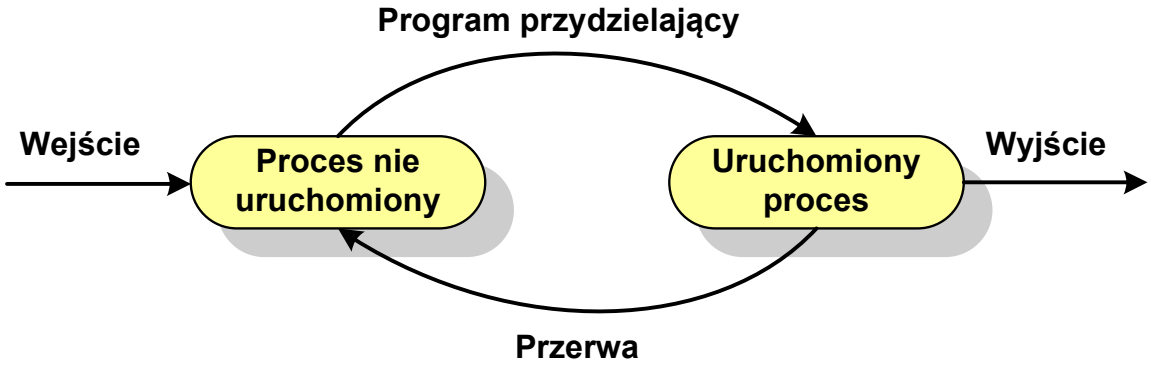
- System operacyjny - jest to program sterujący wykonywaniem aplikacji i działający jako interfejs pomiędzy aplikacjami (użytkownikiem) a sprzętem komputerowym
- System operacyjny - administrator zasobów - zarządza i przydziela zasoby systemu komputerowego oraz steruje wykonaniem programu
- zasób systemu - każdy element systemu, który może być przydzielony innej części systemu lub oprogramowaniu aplikacyjnemu
- do zasobów systemu zalicza się:
  - czas procesora
  - pamięć operacyjną
  - urządzenia zewnętrzne

Zarządzanie procesami

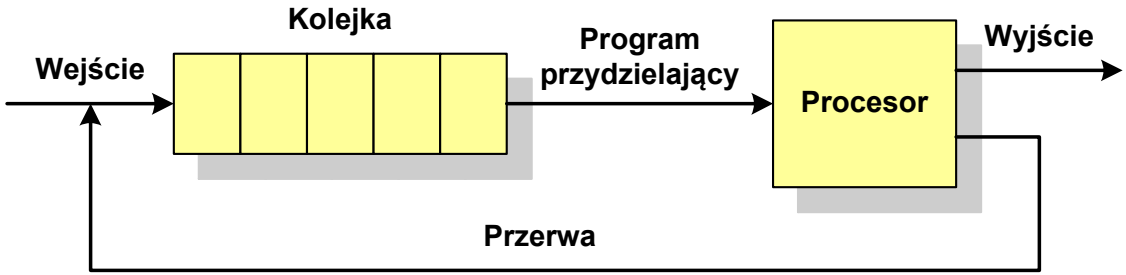
- Głównym zadaniem systemu operacyjnego jest zarządzanie procesami
- Definicja procesu:
  - proces - program w trakcie wykonania
  - proces - ciąg wykonań instrukcji wyznaczanych kolejnymi wartościami licznika rozkazów wynikających z wykonywanej procedury (programu)
  - proces - jednostka, którą można przypisać procesorowi i wykonać
- Proces składa się z kilku elementów:
  - kod programu
  - dane potrzebne programowi (zmienne, przestrzeń robocza, bufory)
  - kontekst wykonywanego programu (stan procesu) - dane wewnętrzne, dzięki którym system operacyjny może nadzorować proces i nim sterować

Dwustanowy model procesu

- najprostszy model polega na tym, że w dowolnej chwili proces jest wykonywany przez procesor (uruchomiony) lub nie (nie uruchomiony)



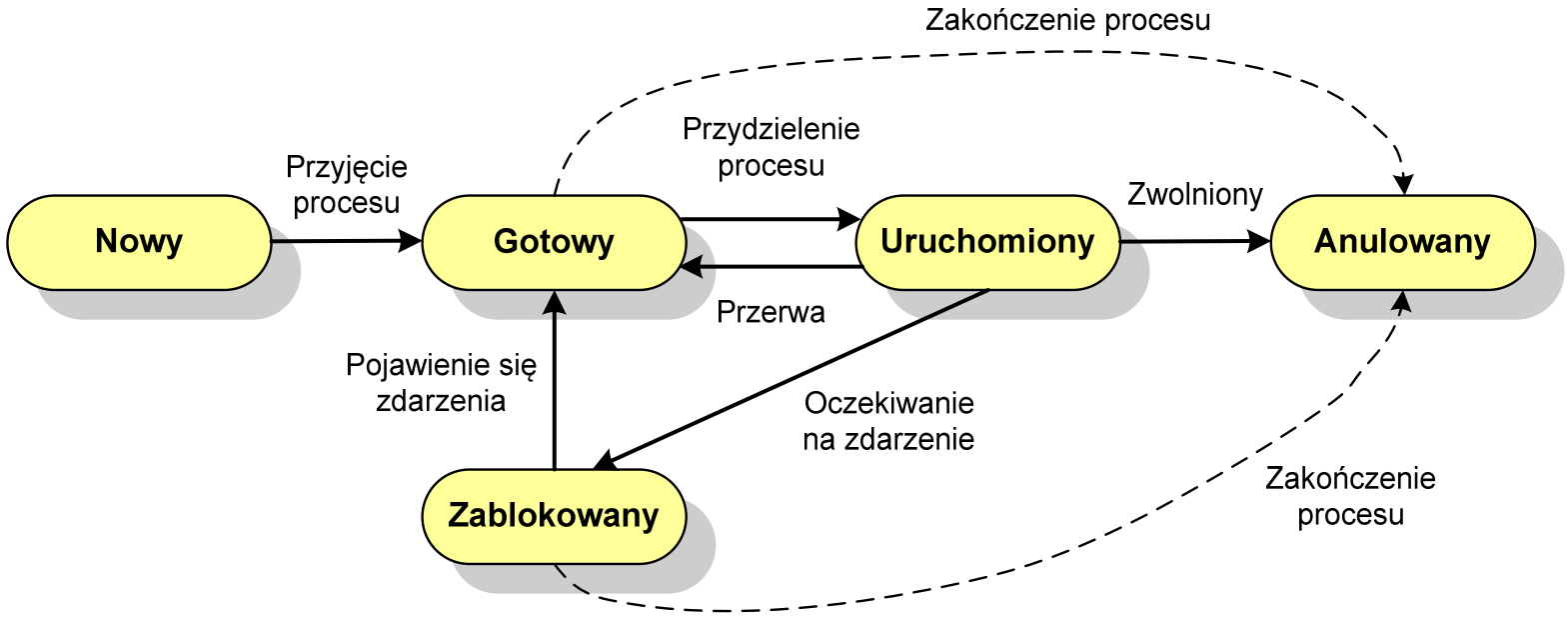
- procesy, które nie są uruchomione czekają w kolejce na wykonanie



- wadą tego modelu jest sytuacja, gdy kolejny proces pobierany do wykonania z kolejki jest zablokowany, gdyż oczekuje na zakończenie operacji we-wy

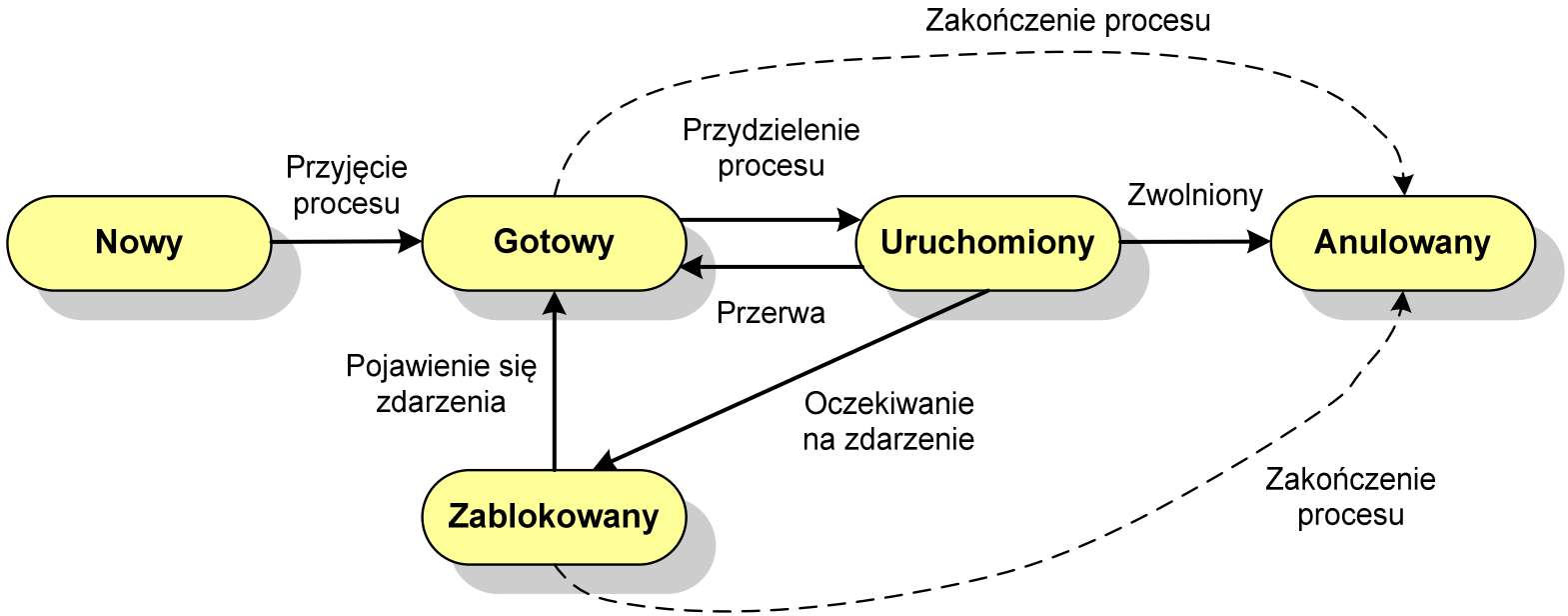
Pięciostanowy model procesu

- rozwiązaniem powyższego problemu jest podział procesów nieuruchomionych na gotowe do wykonania i zablokowane



- pięciostanowy model procesu wymaga zastosowania minimum dwóch kolejek: dla procesów gotowych do wykonania i zablokowanych

Pięciostanowy model procesu



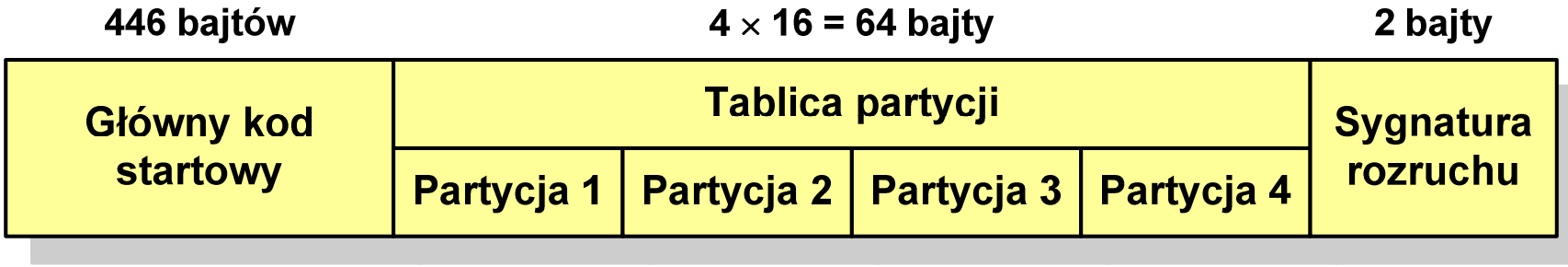
- uruchomiony - proces aktualnie wykonywany
- gotowy - proces gotowy do wykonania przy najbliższej możliwej okazji
- zablokowany - proces oczekujący na zakończenie operacji we-wy
- nowy - proces, który właśnie został utworzony (ma utworzony blok kontrolny procesu, nie został jeszcze załadowany do pamięci), ale nie został jeszcze przyjęty do grupy procesów oczekujących na wykonanie
- anulowany - proces, który został wstrzymany lub anulowany z jakiegoś powodu

Zarządzanie dyskowymi operacjami we-wy

- Struktura dysku twardego
  - MBR (BIOS)
  - GPT (UEFI)
- Systemy plików
  - FAT (FAT12, FAT16, FAT32, exFAT)
  - NTFS
  - ext2

Struktura dysku twardego - MBR

- MBR (Master Boot Record) - główny rekord ładujący (1983, PC DOS 2.0)
- struktura danych opisująca podział dysku na partycje
- pierwszy sektor logiczny dysku (CHS → 0,0,1), zajmuje 512 bajtów



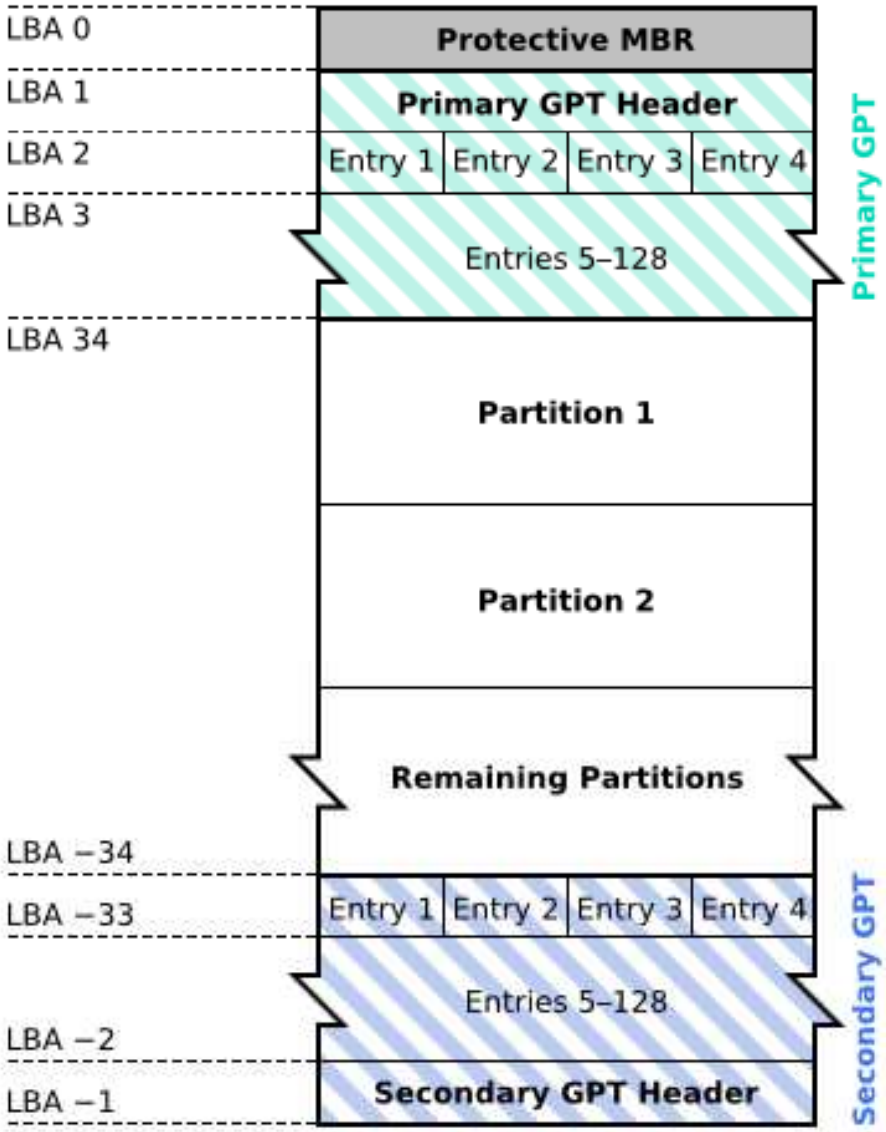
- główny kod startowy (Master Boot Code, bootloader) - program odszukujący i ładujący do pamięci zawartość pierwszego sektora aktywnej partycji
- tablica partycji - cztery 16-bajtowe rekordy opisujące partycje na dysku
- sygnatura rozruchu (boot signature) - znacznik końca MBR (0x55AA)
- maksymalny rozmiar partycji to 2 TB ($2^{32} \times 512$ bajtów)

Struktura dysku twardego - GPT

- GPT (GUID Partition Table) - standard zapisu informacji o partycjach na dysku twardym
- GUID (Globally Unique Identifier) - 128-bitowa liczba stosowana do identyfikowania informacji w systemach komputerowych
- GPT to część standardu UEFI (Unified Extensible Firmware Interface), który zastąpił BIOS w komputerach PC (interfejs graficzny, obsługa myszki)
- opracowanie: IBM/Microsoft, 2010 rok
- maksymalny rozmiar dysku to 9,4 ZB (2^{64} sektorów $\times$ 512 bajtów)
- możliwość utworzenia do 128 partycji podstawowych

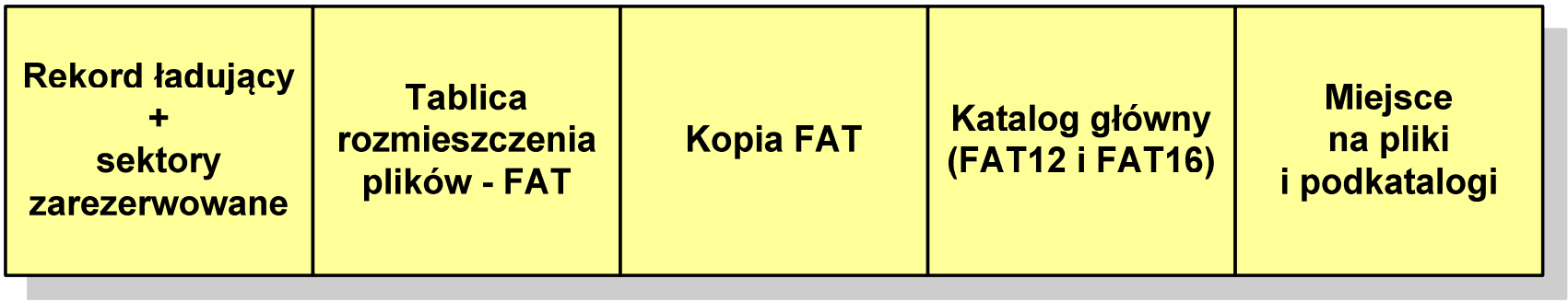
Struktura dysku twardego - GPT (struktura)

- Protective MBR - pozostawiony dla bezpieczeństwa
- GPT Header (512 bajtów):
  - liczba pozycji w tablicy partycji
  - rozmiar pozycji w tablicy partycji
  - położenie zapasowej kopii GPT
  - unikatowy identyfikator dysku
  - sumy kontrolne
- Entry x (128 bajtów):
  - typ partycji
  - unikatowy identyfikator
  - początkowy i końcowy numer LBA
  - atrybuty
  - nazwa



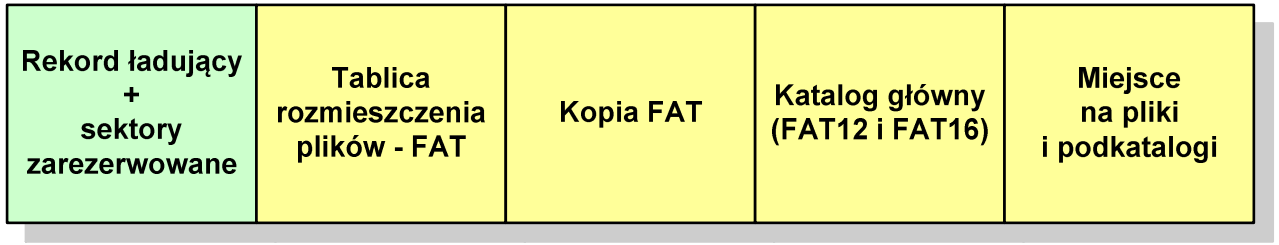
System plików FAT (File Allocation Table)

- opracowany na przełomie lat 70. i 80. dla systemu MS-DOS
- występuje w czterech wersjach: FAT12, FAT16, FAT32 i exFAT (FAT64)
- numer występujący po słowie FAT oznacza liczbę bitów przeznaczonych do kodowania (numeracji) jednostek alokacji pliku (JAP), tzw. klastrów (ang. cluster) w tablicy alokacji plików
  - 12 bitów w systemie FAT12
  - 16 bitów w systemie FAT16
  - 32 bity w systemie FAT32 (praktycznie 28)
  - 64 bity w systemie exFAT (FAT64)
- ogólna struktura dysku logicznego / dyskietki w systemie FAT:



FAT12

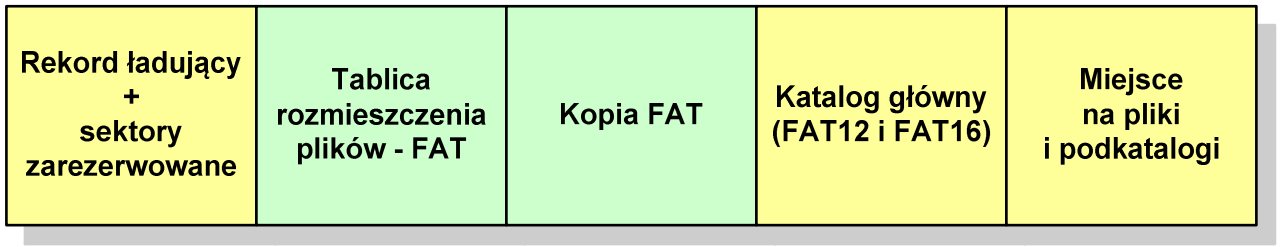
- system plików FAT12 przeznaczony jest dla nośników o małej pojemności
- rekord ładujący zajmuje pierwszy sektor dyskietki lub dysku logicznego



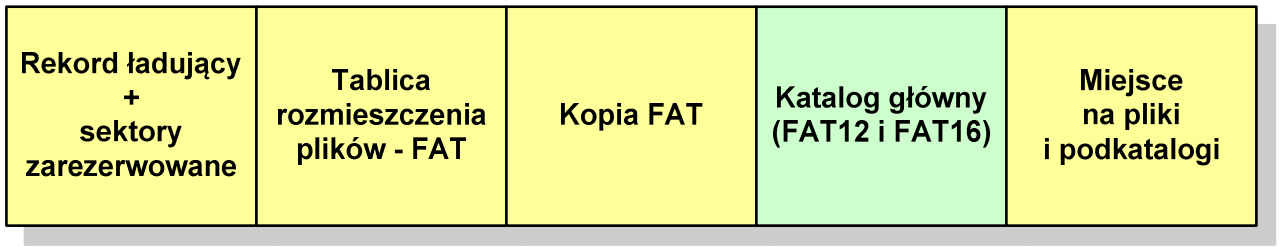
- rekord ładujący zawiera następujące dane:
  - instrukcja skoku do początku programu ładującego (3 bajty)
  - nazwa wersji systemu operacyjnego (8 bajtów)
  - struktura BPB (ang. BIOS Parametr Block) - blok parametrów BIOS (25 bajtów)
  - rozszerzony BPB (ang. Extended BPB, 26 bajtów)
  - wykonywalny kod startowy uruchamiający system operacyjny (448 bajtów)
  - znacznik końca sektora - 55AAH (2 bajty)

FAT12

- tablica rozmieszczenia plików FAT tworzy swego rodzaju „mapę” plików zapisanych na dysku
- za tablicą FAT znajduje się jej kopia, która nie jest wykorzystywana



- za kopią tablicy FAT znajduje się katalog główny zajmujący określoną dla danego typu dysku liczbę sektorów

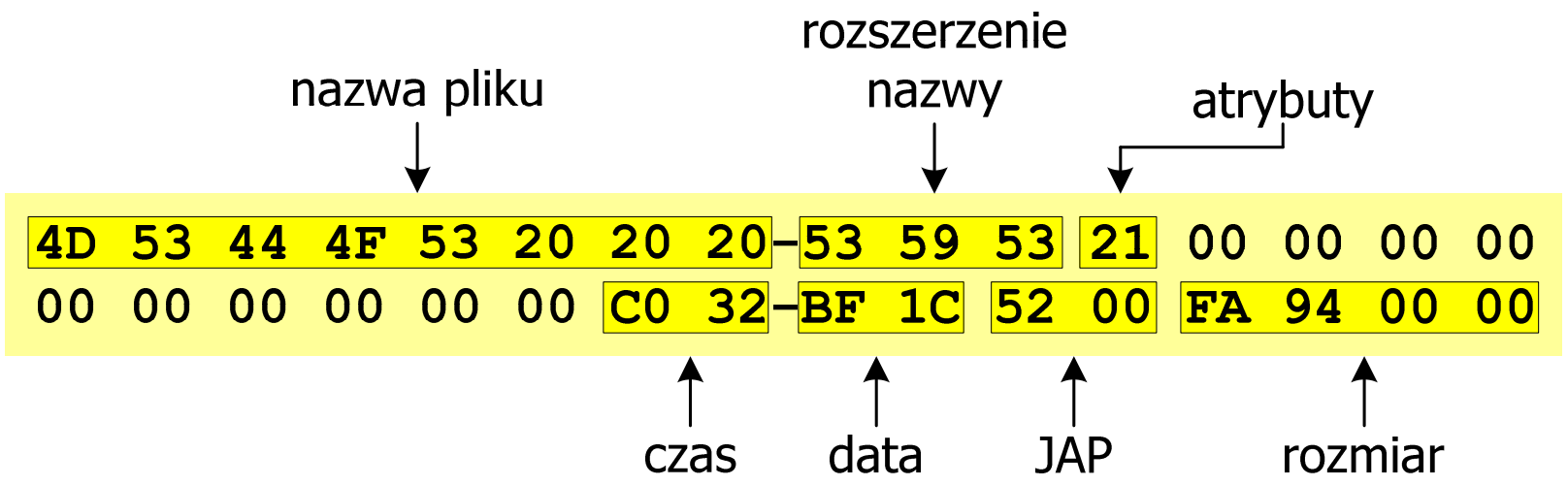


- katalog główny zawiera 32-bajtowe pola mogące opisywać pliki, podkatalogi lub etykietę dysku

FAT12

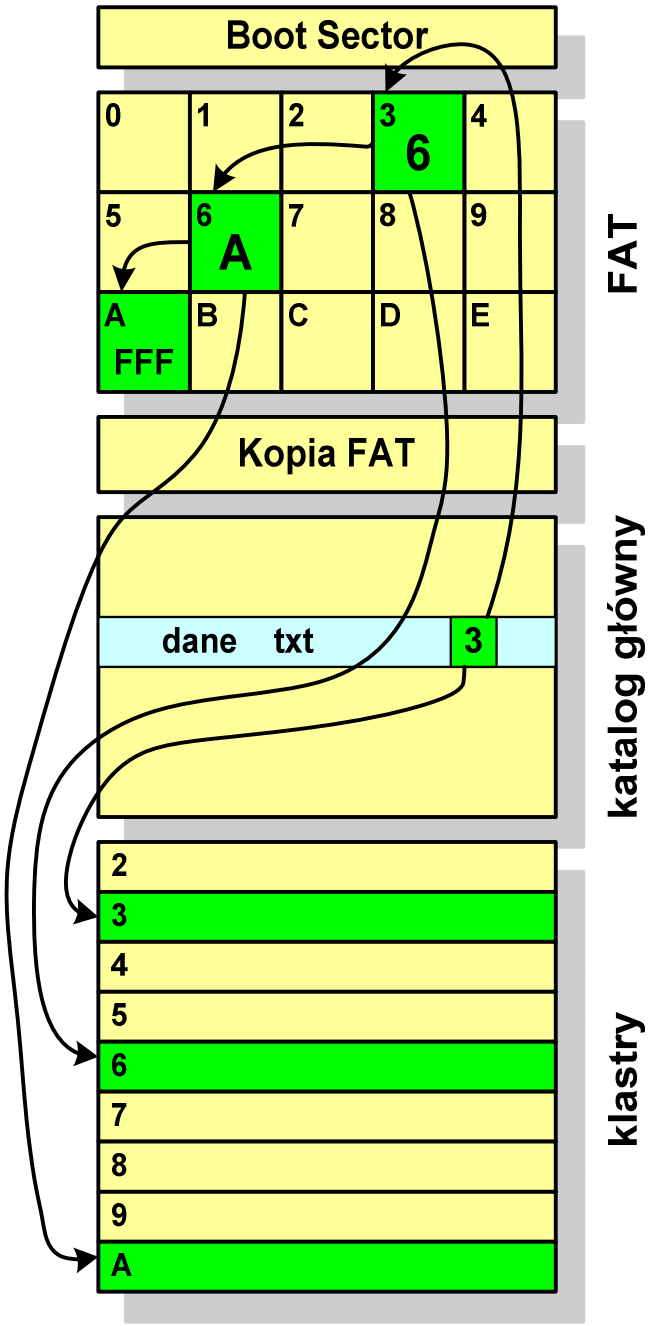
- przykładowa zawartość katalogu głównego:

```
0000   49 4F 20 20 20 20 20 20-53 59 53 21 00 00 00 00    IO      SYS!....
0010   00 00 00 00 00 00 C0 32-BF 1C 02 00 46 9F 00 00    .......2....F...
0020   4D 53 44 4F 53 20 20 20-53 59 53 21 00 00 00 00    MSDOS   SYS!....
0030   00 00 00 00 00 00 C0 32-BF 1C 52 00 FA 94 00 00    .......2..R.....
0040   43 4F 4D 4D 41 4E 44 20-43 4F 4D 20 00 00 00 00    COMMAND COM ....
0050   00 00 00 00 00 00 C0 32-BF 1C 9D 00 75 D5 00 00    .......2....u...
0060   41 54 54 52 49 42 20 20-45 58 45 20 00 00 00 00    ATTRIB  EXE ....
0070   00 00 00 00 00 00 C0 32-BF 1C 08 01 C8 2B 00 00    .......2.....+..
```



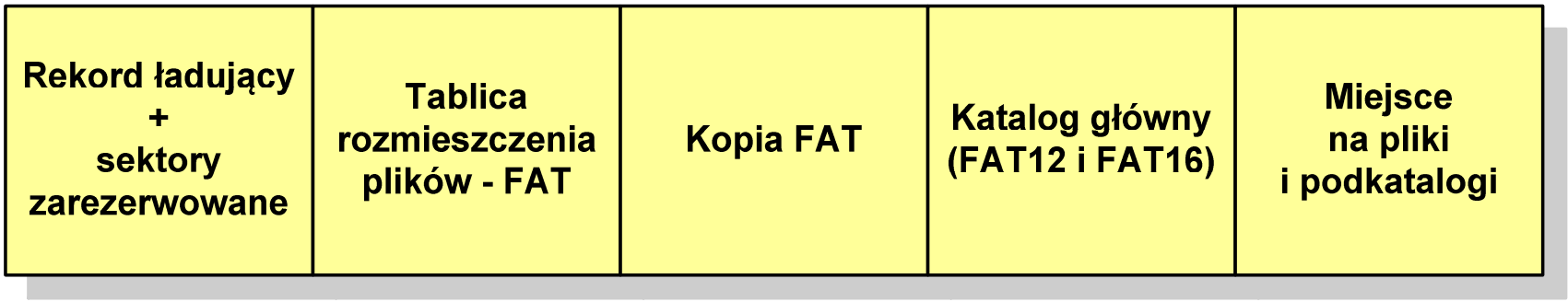
FAT12 - położenie pliku na dysku

- w katalogu, w 32-bajtowym polu każdego pliku wpisany jest początkowy numer JAP
- numer ten określa logiczny numer sektora, w którym znajduje się początek pliku
- ten sam numer JAP jest jednocześnie indeksem do miejsca w tablicy FAT, w którym wpisany jest numer kolejnej JAP
- numer wpisany we wskazanym miejscu tablicy rozmieszczenia plików wskazuje pierwszy sektor następnej części pliku i równocześnie położenie w tablicy FAT numeru następnej JAP
- w ten sposób tworzy się łańcuch, określający położenie całego pliku
- jeśli numer JAP składa się z samych FFF, to oznacza to koniec pliku



FAT16

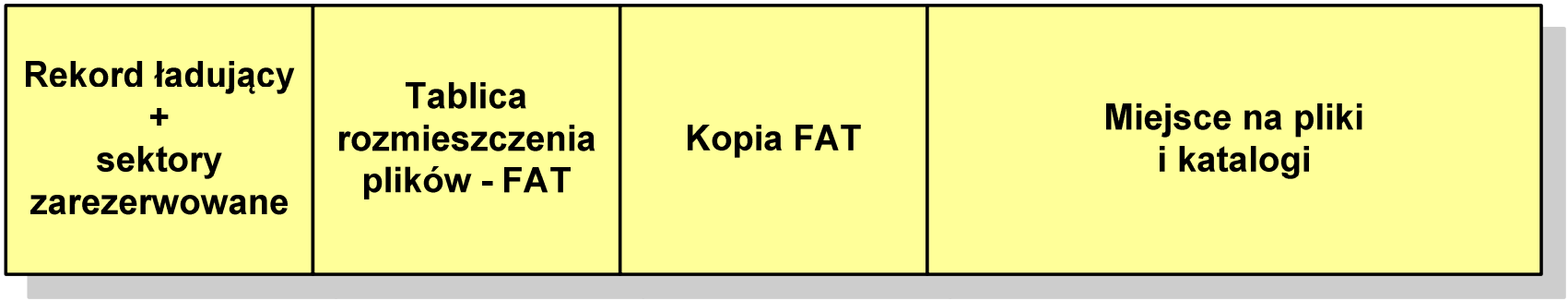
- po raz pierwszy pojawił się w systemie MS-DOS 3.3
- ogólna struktura dyskietki / dysku logicznego w systemie FAT16 jest taka sama jak w przypadku FAT12



- maksymalna liczba JAP ograniczona jest do 2^{16} czyli 65536
- maksymalny rozmiar dysku logicznego:
  - DOS, Windows 95 - ok. 2 GB (gdyż maksymalny rozmiar JAP to 2^{15} bajtów)
  - Windows 2000 - ok. 4 GB (gdyż maksymalny rozmiar JAP to 2^{16} bajtów)

FAT32

- po raz pierwszy wprowadzony w systemie Windows 95 OSR2
- ogólna struktura systemu FAT32 jest taka sama jak w FAT12/FAT16 - nie ma tylko miejsca przeznaczonego na katalog główny
- w systemie FAT32 katalog główny może znajdować się w dowolnym miejscu na dysku i może zawierać maksymalnie 65 532 pliki i katalogi



- do adresowania JAP stosuje się, obcięty o 4 najstarsze bity, adres 32-bitowy i dlatego dysk z FAT32 może zawierać maksymalnie 2^{28} JAP
- w systemie FAT32 można formatować tylko dyski, nie można natomiast zainstalować go na dyskietkach

FAT32 - długie nazwy plików

- wprowadzone w systemie Windows 95
- informacje o nazwie pliku zapamiętywane są jako:
  - długa nazwa (13 znaków w każdej 32-bajtowej strukturze, Unicode)
  - skrócona nazwa (32-bajtowa struktura, stary format 8+3)

długa nazwa pliku

```
0000  43 20 00 64 00 6F 00 6D-00 6F 00 0F 00 CF 77 00   C .d.o.m.o....w.
0010  61 00 2E 00 74 00 78 00-74 00 00 00 00 00 FF FF   a...t.x.t.......

0020  02 63 00 79 00 6A 00 6E-00 65 00 0F 00 CF 20 00   .c.y.j.n.e.... .
0030  2D 00 20 00 70 00 72 00-61 00 00 00 63 00 61 00   -. .p.r.a...c.a.

0040  01 53 00 79 00 73 00 74-00 65 00 0F 00 CF 6D 00   .S.y.s.t.e....m.
0050  79 00 20 00 4F 00 70 00-65 00 00 00 72 00 61 00   y. .O.p.e...r.a.

0060  53 59 53 54 45 4D 7E 31-54 58 54 20 00 4B 03 80   SYSTEM~1TXT .K..
0070  67 32 67 32 00 00 08 80-67 32 02 00 06 00 00 00   g2g2....g2......
```

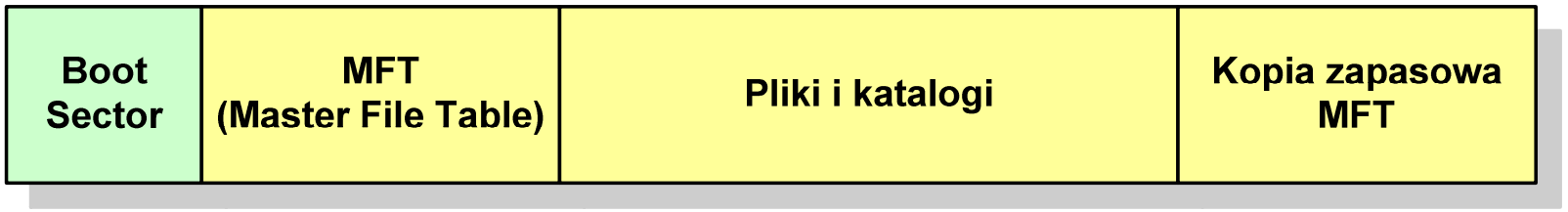
skrócona nazwa pliku

exFAT (FAT64)

- po raz pierwszy pojawił się w listopadzie 2006 roku w Windows Embedded CE 6.0 i Windows Vista SP1
- obsługiwany także przez Windows 7/8/10, Windows Server 2003/2008, Windows XP SP2/SP3, Linux
- stworzony przez Microsoft na potrzeby pamięci Flash
- podstawowe cechy:
  - maksymalna wielkość pliku to 2^{64} = 16 EB
  - maksymalna wielkość klastra - do 32 MB
  - nieograniczona liczba plików w pojedynczym katalogu
  - prawa dostępu do plików i katalogów

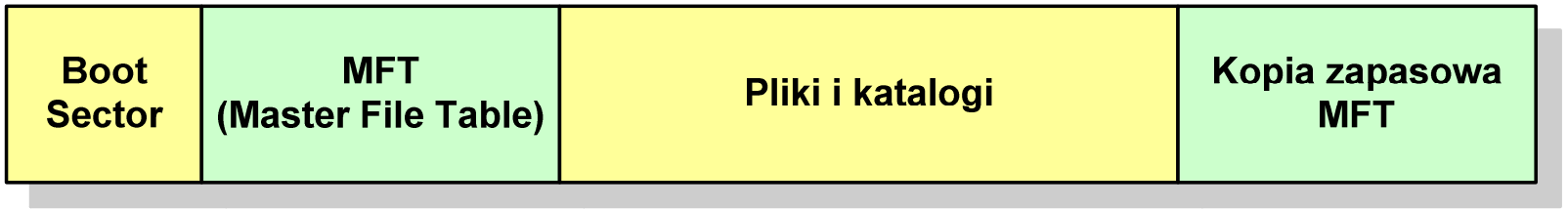
NTFS (New Technology File System)

- wersja 1.0 (połowa 1993 r.) - Windows NT 3.1
- wersja 3.1 (NTFS 5.1) - Windows XP/Server 2003/Vista/7/8/10
- struktura wolumenu (dysku) NTFS:



- Boot Sector rozpoczyna się od zerowego sektora partycji, może zajmować 16 kolejnych sektorów, zawiera podobne dane jak w systemie FAT

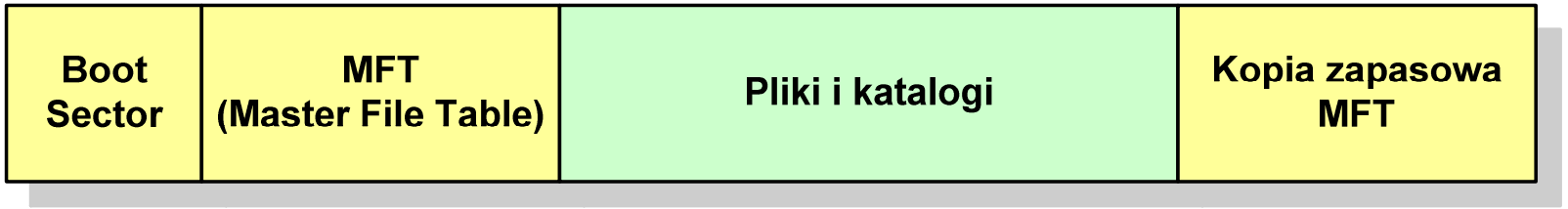
NTFS



- MFT (Master File Table) - specjalny plik, niewidoczny dla użytkownika, zawiera wszystkie dane niezbędne do odczytania pliku z dysku, składa się z rekordów o stałej długości (1 kB - 4 kB)
- pierwsze 16 (NTFS 4) lub 26 (NTFS 5) rekordów jest zarezerwowane dla tzw. metaplików, np.
  - rekord nr: 0 plik: $Mft (główna tablica plików)
  - rekord nr: 1 plik: $MftMirr (główna tablica plików 2)
  - rekord nr: 5 plik: $ (indeks katalogu głównego)
- pozostała część pliku MFT przeznaczona jest na rekordy wszystkich plików i katalogów umieszczonych na dysku

NTFS

- struktura wolumenu (dysku) NTFS:



- plik w NTFS to zbiór atrybutów
- wszystkie atrybuty mają dwie części składowe: nagłówek i blok danych
- nagłówek opisuje atrybut, np. liczbę bajtów zajmowanych przez atrybut, rozmiar bloku danych, położenie bloku danych, znacznik czasu
- bloku danych zawiera informacje zgodne z przeznaczeniem atrybutu

NTFS - Pliki

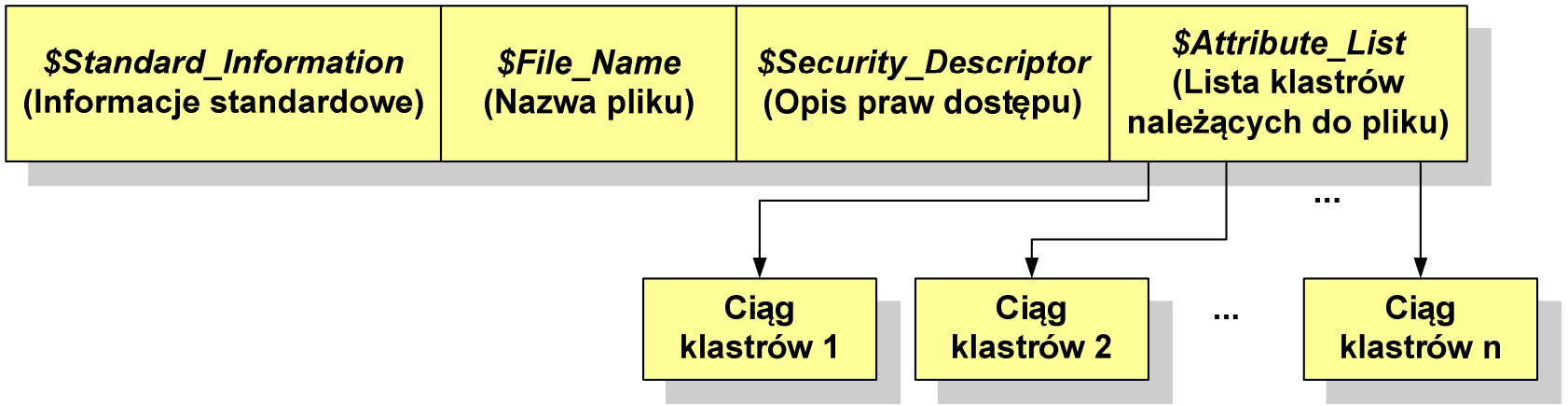
- pliki w systemie NTFS są reprezentowane w MFT przez rekord zawierający atrybuty:
  - $Standard_Information
  - $File_Name
  - $Security_Descriptor
  - $Data

$Standard_Information (Informacje standardowe)	*$File_Name* (Nazwa pliku)	*$Security_Descriptor* (Opis praw dostępu)	*$Data* (Dane)

- w przypadku małych plików wszystkie jego atrybuty zapisywane są bezpośrednio w MFT (atrybuty rezydentne)

NTFS - Pliki

- jeśli atrybuty pliku są duże (najczęściej dotyczy to atrybutu $Data), to w rekordzie w MFT umieszczany jest tylko nagłówek atrybutu oraz wskaźnik do jego bloku danych, a sam blok danych przenoszony jest na dysk poza MFT (atrybuty nierezydentne)
- blok danych atrybutu nierezydentnego zapisywany jest w przyległych klastrach
- jeśli nie jest to możliwe, to dane zapisywane są w kilku ciągach jednostek alokacji i wtedy każdemu ciągowi odpowiada wskaźnik w rekordzie MFT



NTFS - Katalogi

- katalogi reprezentowane są przez rekordy zawierające trzy takie same atrybuty jak pliki:
  - $Standard_Information
  - $File_Name
  - $Security_Descriptor

$Standard_Information (Informacje standardowe)	*$File_Name* (Nazwa pliku)	*$Security_Descriptor* (Opis praw dostępu)	*$Index_Root*	*$Index_Allocation*	*$Bitmap*

- zamiast atrybutu $Data umieszczone są trzy atrybuty przeznaczone do tworzenia list, sortowania oraz lokalizowania plików i podkatalogów
  - $Index_Root
  - $Index_Allocation
  - $Bitmap

ext2

- pierwszy system plików w Linuxie: Minix (14-znakowe nazwy plików i maksymalny rozmiar wynoszący 64 MB)
- system Minix zastąpiono nowym systemem nazwanym rozszerzonym systemem plików - ext (ang. extended file system), a ten, w styczniu 1993 r., systemem ext2 (ang. second extended file system)
- w systemie ext2 podstawowym elementem podziału dysku jest blok
- wielkość bloku jest stała w ramach całego systemu plików, określana na etapie jego tworzenia i może wynosić 1024, 2048 lub 4096 bajtów
- w celu zwiększenia bezpieczeństwa i optymalizacji zapisu na dysku posługujemy się nie pojedynczymi blokami, a grupami bloków

ext2

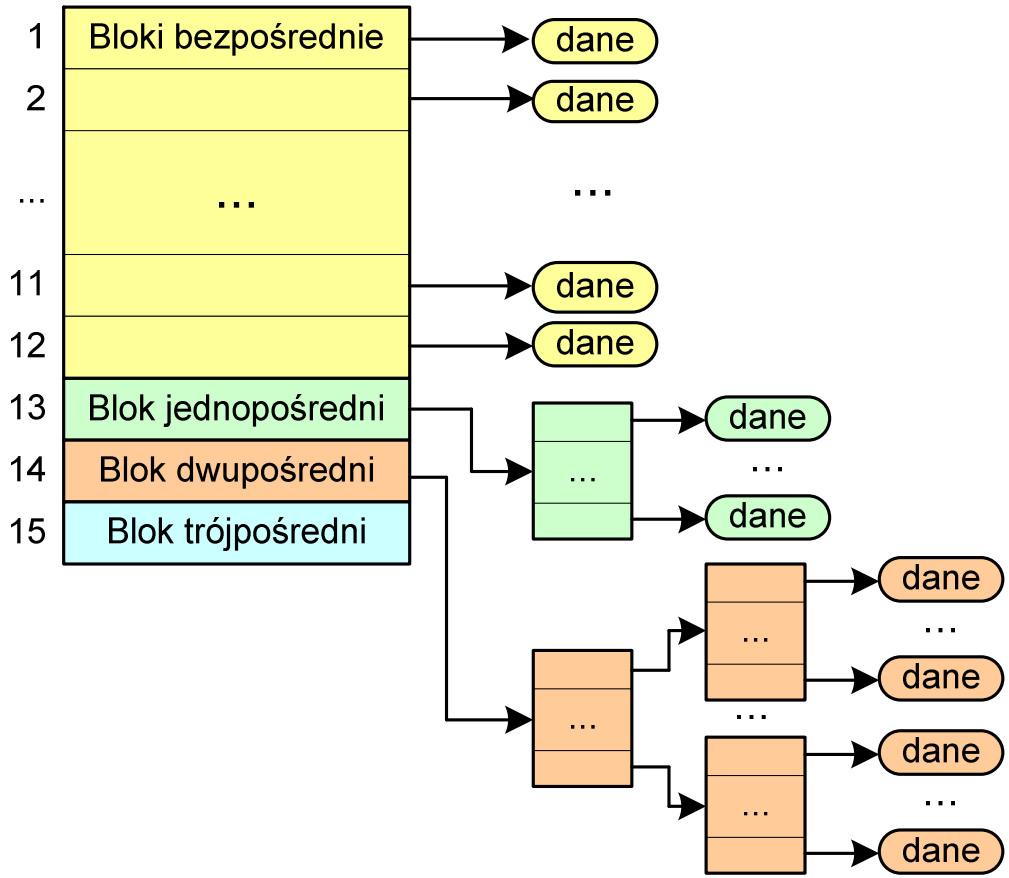
- każda grupa bloków zawiera ten sam blok identyfikacyjny oraz kopie bloków z deskryptorami wszystkich grup
- blok identyfikacyjny zawiera informacje na temat systemu plików (np. rodzaj systemu plików, rozmiar bloku)
- deskryptor grupy opisuje grupę bloków (np. położenie bloków z mapami bitowymi, liczba wolnych bloków, liczba katalogów w grupie)
- blok z mapą bitową zajętości bloków danej grupy - tablica bitów, zajmuje jeden blok (np. dla bloku o rozmiarze 1 kB opisuje 8096 bloków danych)
- blok z mapa bitową zajętości i-węzłów danej grupy - tablica bitów, każdy bit zawiera informację czy dany i-węzeł jest wolny czy zajęty

ext2 - i-węzeł

- pliki na dysku reprezentowane są przez i-węzły (ang. i-node)
- każdemu plikowi odpowiada dokładnie jeden i-węzeł, który jest strukturą zawierającą m.in. następujące pola:
  - numer i-węzła w dyskowej tablicy i-węzłów
  - typ pliku: zwykły, katalog, łącze nazwane, specjalny, znakowy
  - prawa dostępu do pliku: dla wszystkich, grupy, użytkownika
  - liczba dowiązań do pliku
  - identyfikator właściciela pliku
  - identyfikator grupy właściciela pliku
  - rozmiar pliku w bajtach (max. 4 GB)
  - czas utworzenia pliku
  - czas ostatniego dostępu do pliku
  - czas ostatniej modyfikacji pliku
  - liczba bloków dyskowych zajmowanych przez plik

ext2 - i-węzeł

- położenie pliku na dysku określają w i-węźle pola:
  - 12 adresów bloków zawierających dane (w systemie Unix jest ich 10) - bloki bezpośrednie
  - 1 adres bloku zawierającego adresy bloków zawierających dane - blok jednopośredni (ang. single indirect block)
  - 1 adres bloku zawierającego adresy bloków jednopośrednich - blok dwupośredni (ang. double indirect block)
  - 1 adres bloku zawierającego adresy bloków dwupośrednich - blok trójpośredni (ang. triple indirect block)



ext2

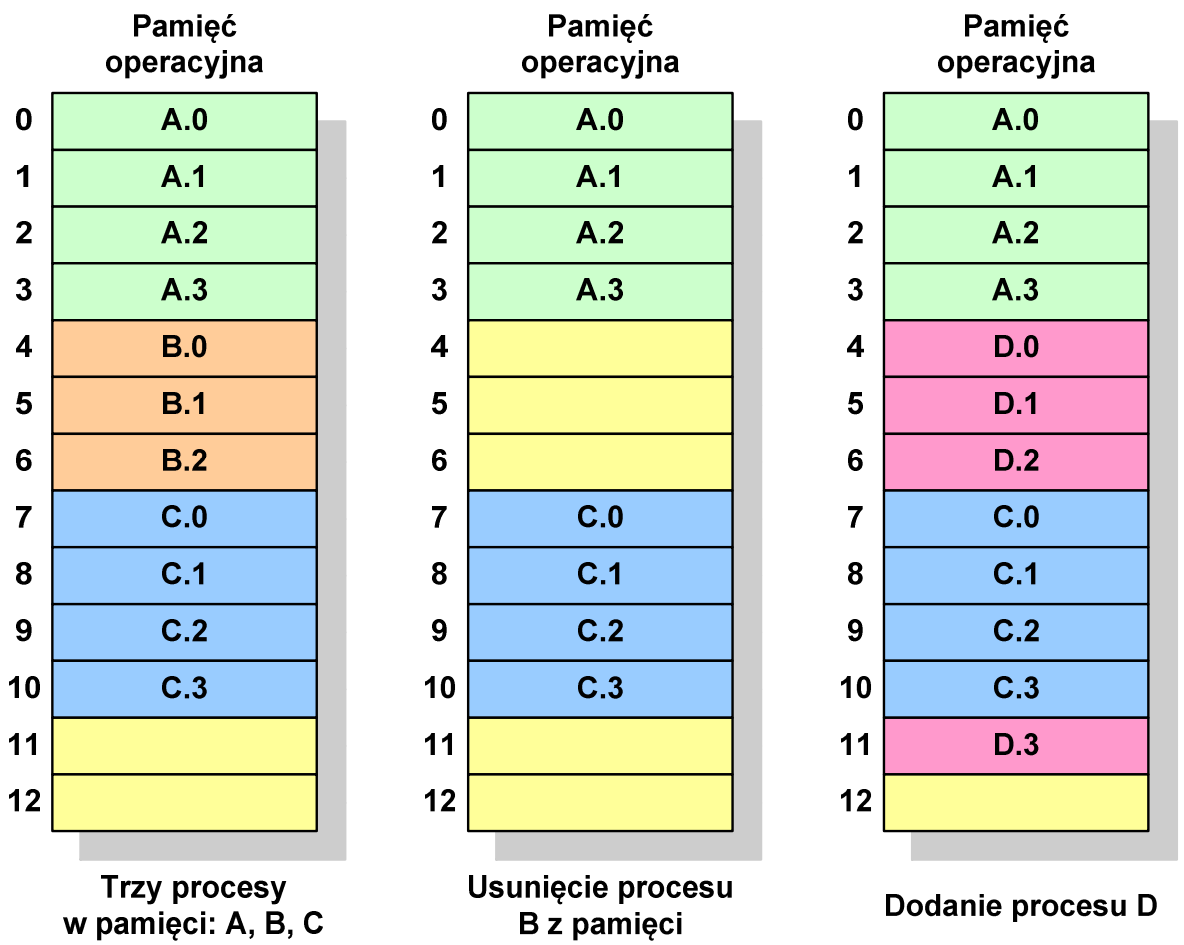
- nazwy plików przechowywane są w katalogach, które w systemie Linux są plikami, ale o specjalnej strukturze
- katalogi składają się z ciągu tzw. pozycji katalogowych o nieustalonej z góry długości
- każda pozycja opisuje dowiązanie do jednego pliku i zawiera:
  - numer i-węzła (4 bajty)
  - rozmiar pozycji katalogowej (2 bajty)
  - długość nazwy (2 bajty)
  - nazwa (od 1 do 255 znaków)

Zarządzanie pamięcią

- zarządzanie pamięcią polega na wydajnym przenoszeniu programów i danych do i z pamięci operacyjnej
- w nowoczesnych wieloprogramowych systemach operacyjnych zarządzanie pamięcią opiera się na pamięci wirtualnej
- pamięć wirtualna bazuje na wykorzystaniu segmentacji i stronicowania
- z historycznego punktu widzenia w systemach komputerowych stosowane były/są następujące metody zarządzania pamięcią:
  - proste stronicowanie, prosta segmentacja
  - stronicowanie pamięci wirtualnej, segmentacja pamięci wirtualnej
  - stronicowanie i segmentacja pamięci wirtualnej

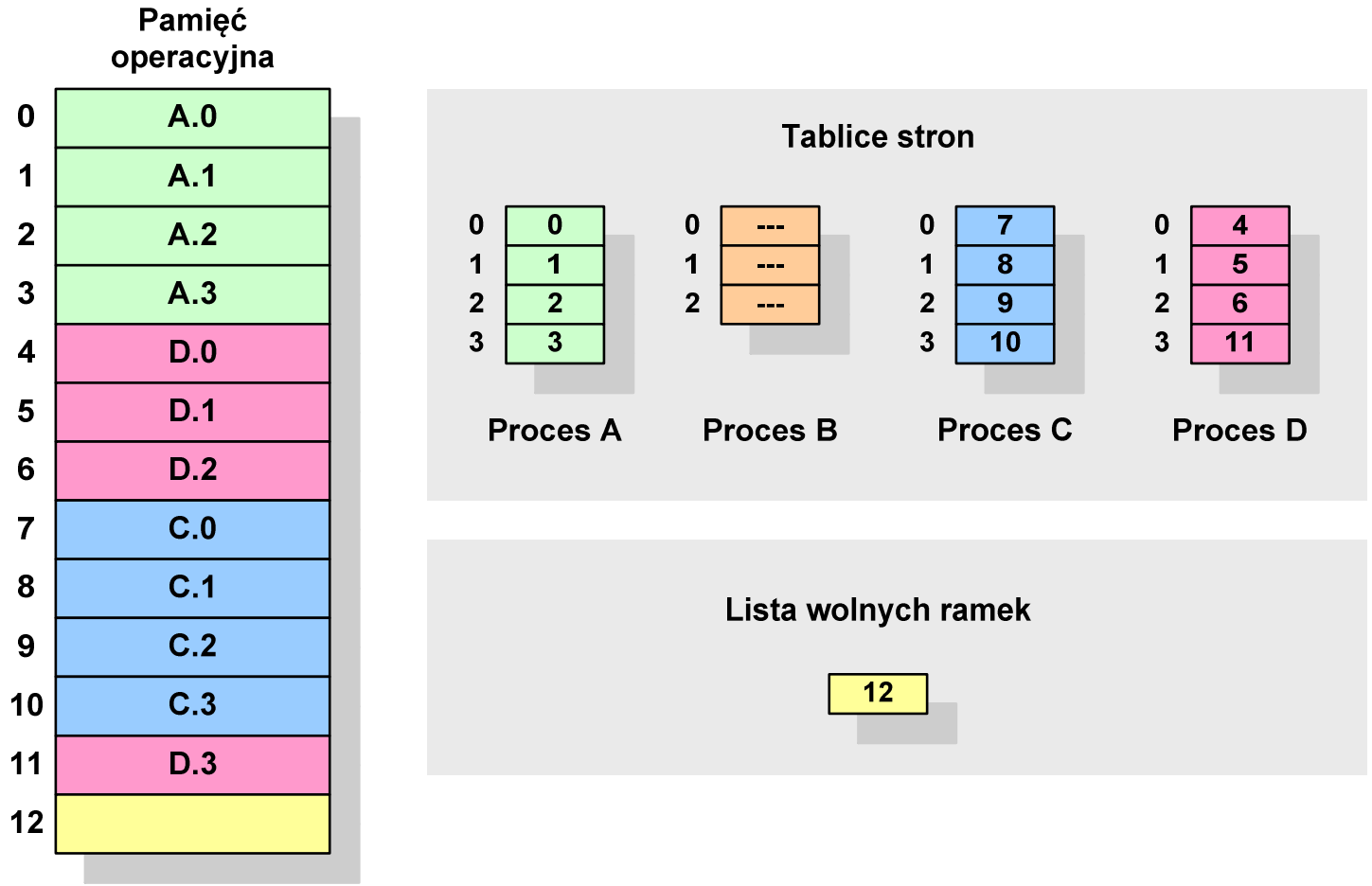
Proste stronicowanie

- pamięć operacyjna podzielona jest na jednakowe bloki o stałym niewielkim rozmiarze nazywane ramkami lub ramkami stron (page frames)
- do tych ramek wstawiane są fragmenty procesu zwane stronami (pages)
- aby proces mógł zostać uruchomiony wszystkie jego strony muszą znajdować się w pamięci operacyjnej



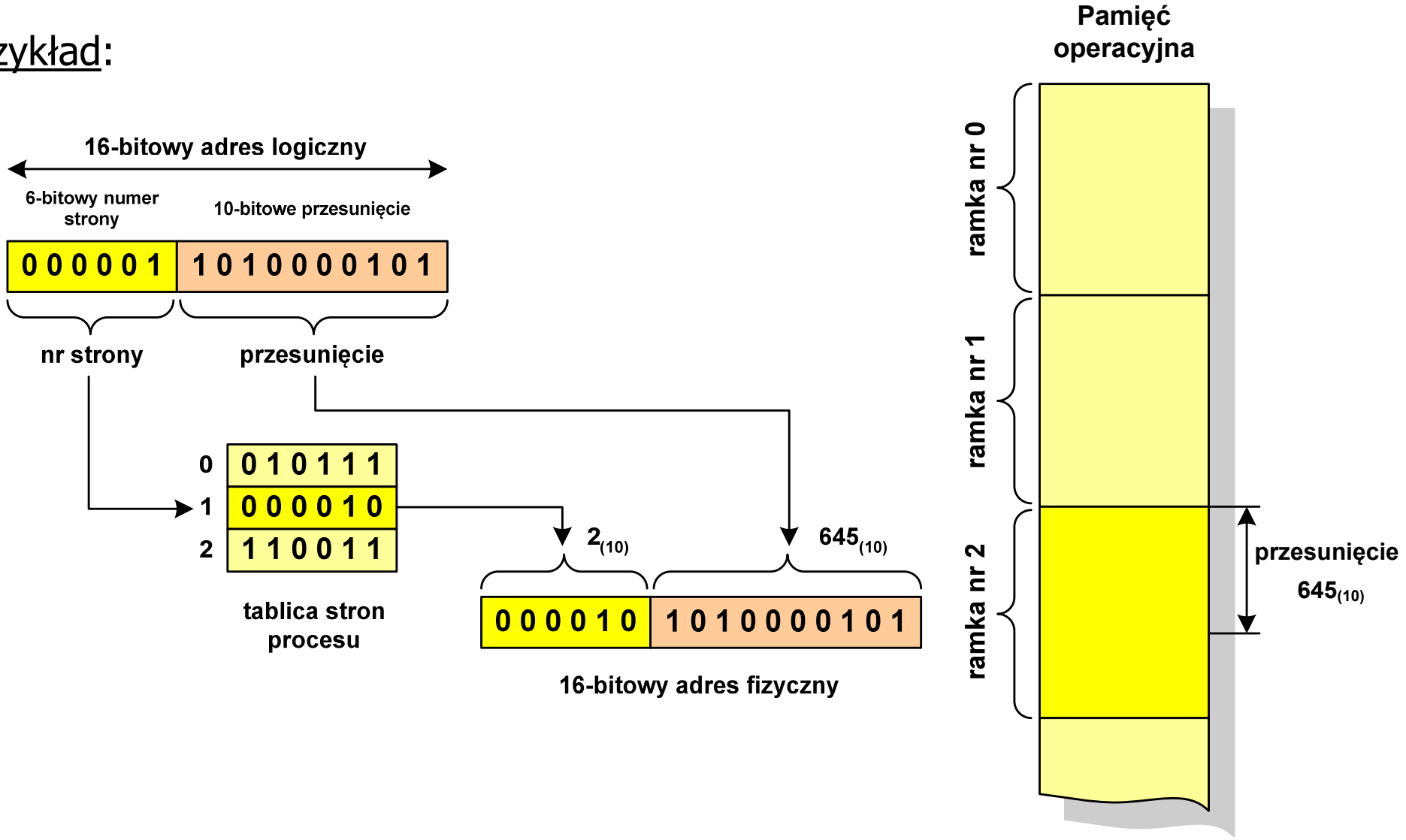
Proste stronicowanie

- dla każdego procesu przechowywana jest tablica strony (page table) zawierająca lokalizację ramki dla każdej strony procesu



Proste stronicowanie

Przykład:

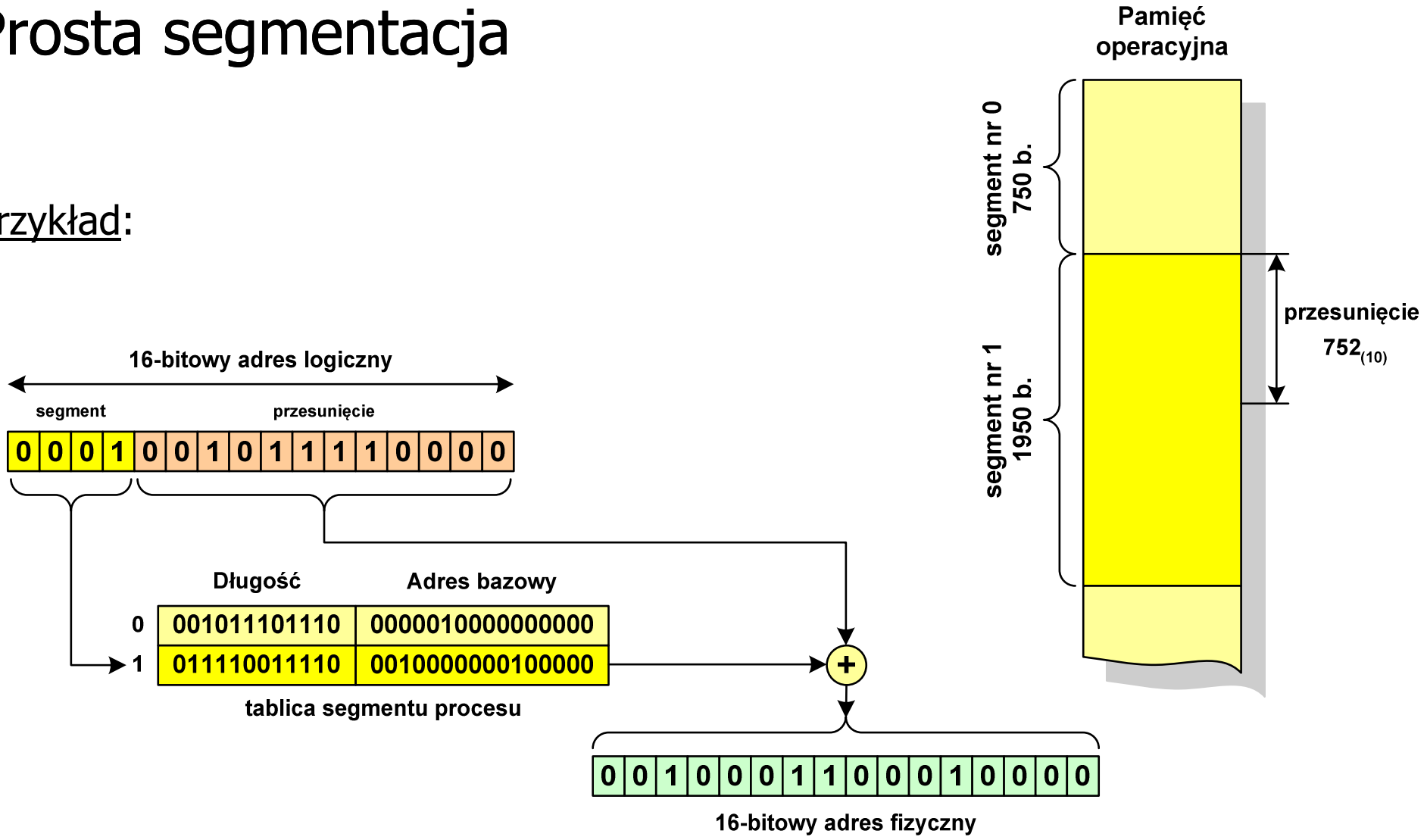


Prosta segmentacja

- polega na podzieleniu programu i skojarzonych z nim danych na odpowiednią liczbę segmentów o różnej długości
- ładowanie procesu do pamięci polega na wczytaniu wszystkich jego segmentów do partycji dynamicznych (nie muszą być ciągłe)
- segmentacja jest widoczna dla programisty i ma na celu wygodniejszą organizację programów i danych
- adres logiczny wykorzystujący segmentację składa się z dwóch części:
  - numeru segmentu
  - przesunięcia
- dla każdego procesu określana jest tablica segmentu procesu zawierająca:
  - długość danego segmentu
  - adres początkowy danego segmentu w pamięci operacyjnej

Prosta segmentacja

Przykład:



Pamięć wirtualna

- pamięć wirtualna umożliwia przechowywanie stron/segmentów wykonywanego procesu w pamięci dodatkowej (na dysku twardym)

Co się dzieje, gdy procesor chce odczytać stronę z pamięci dodatkowej?

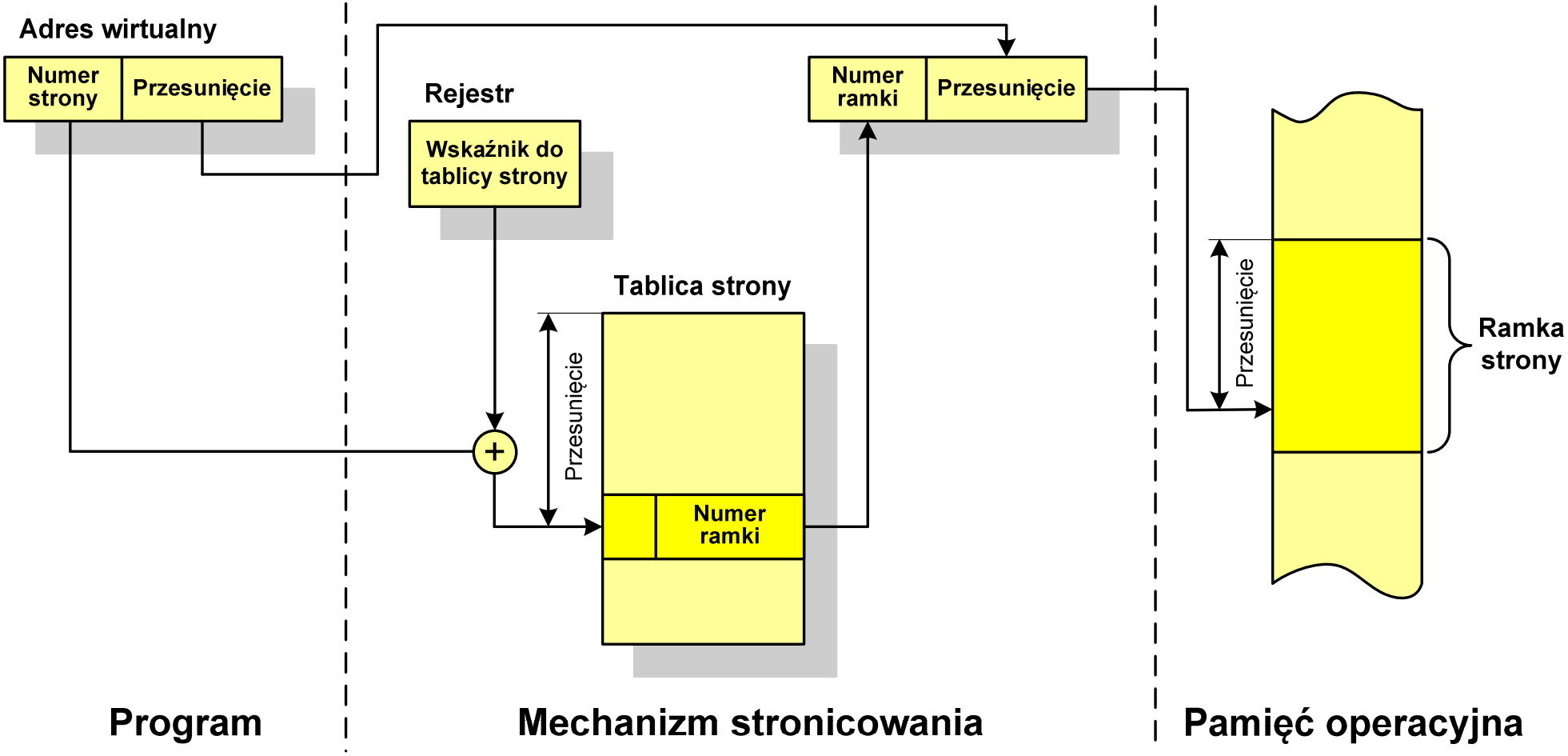
- generowanie przerwania sygnalizującego błąd w dostępie do pamięci
- zmiana stan procesu na zablokowany
- wstawienie do pamięci operacyjnej fragment procesu zawierający adres logiczny, który był przyczyną błędu
- zmiana stanu procesu na uruchomiony

Dzięki zastosowaniu pamięci wirtualnej:

- w pamięci operacyjnej może być przechowywanych więcej procesów
- proces może być większy od całej pamięci operacyjnej

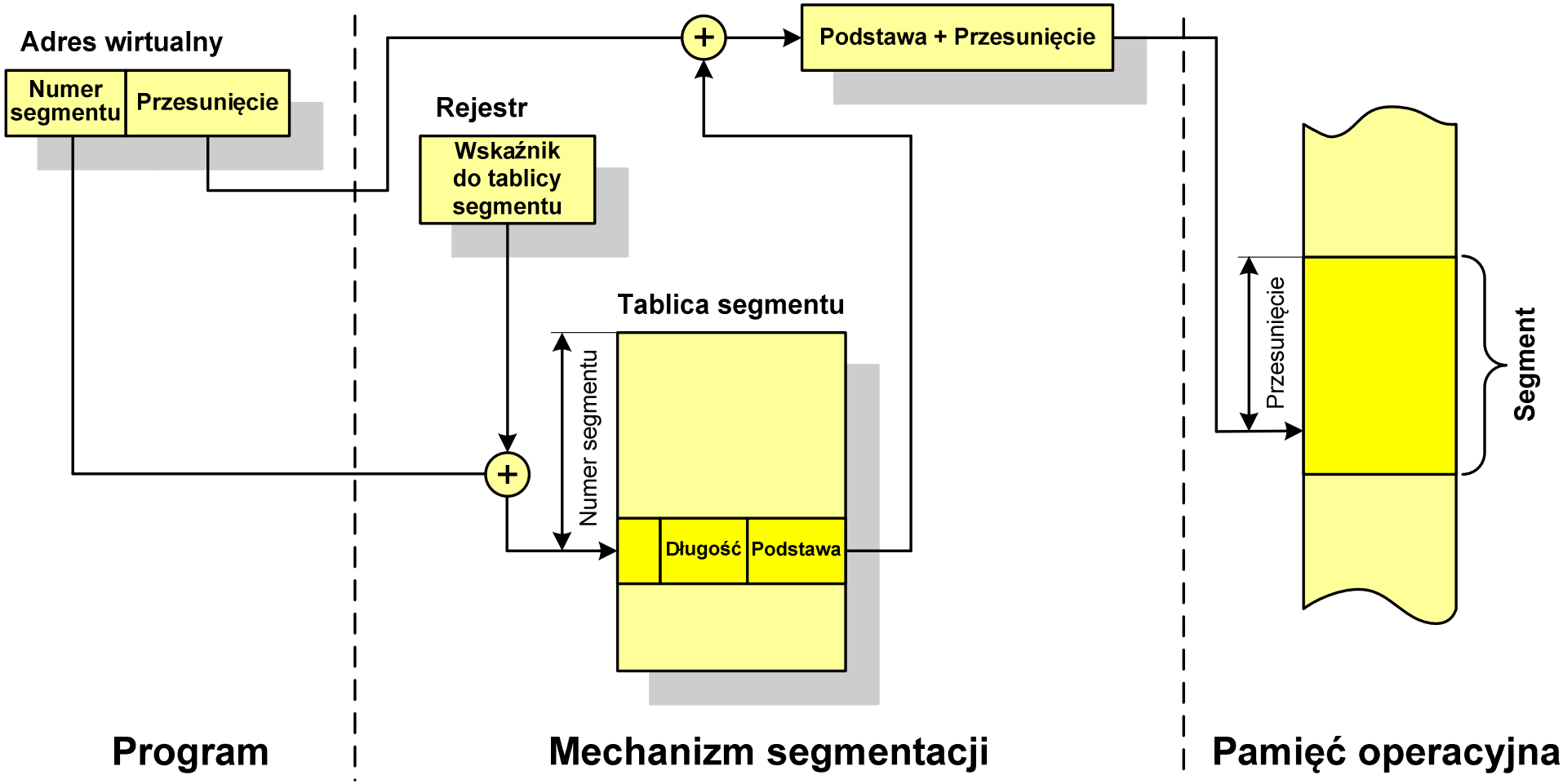
Stronicowanie pamięci wirtualnej

- odczytanie strony wymaga translacji adresu wirtualnego na fizyczny



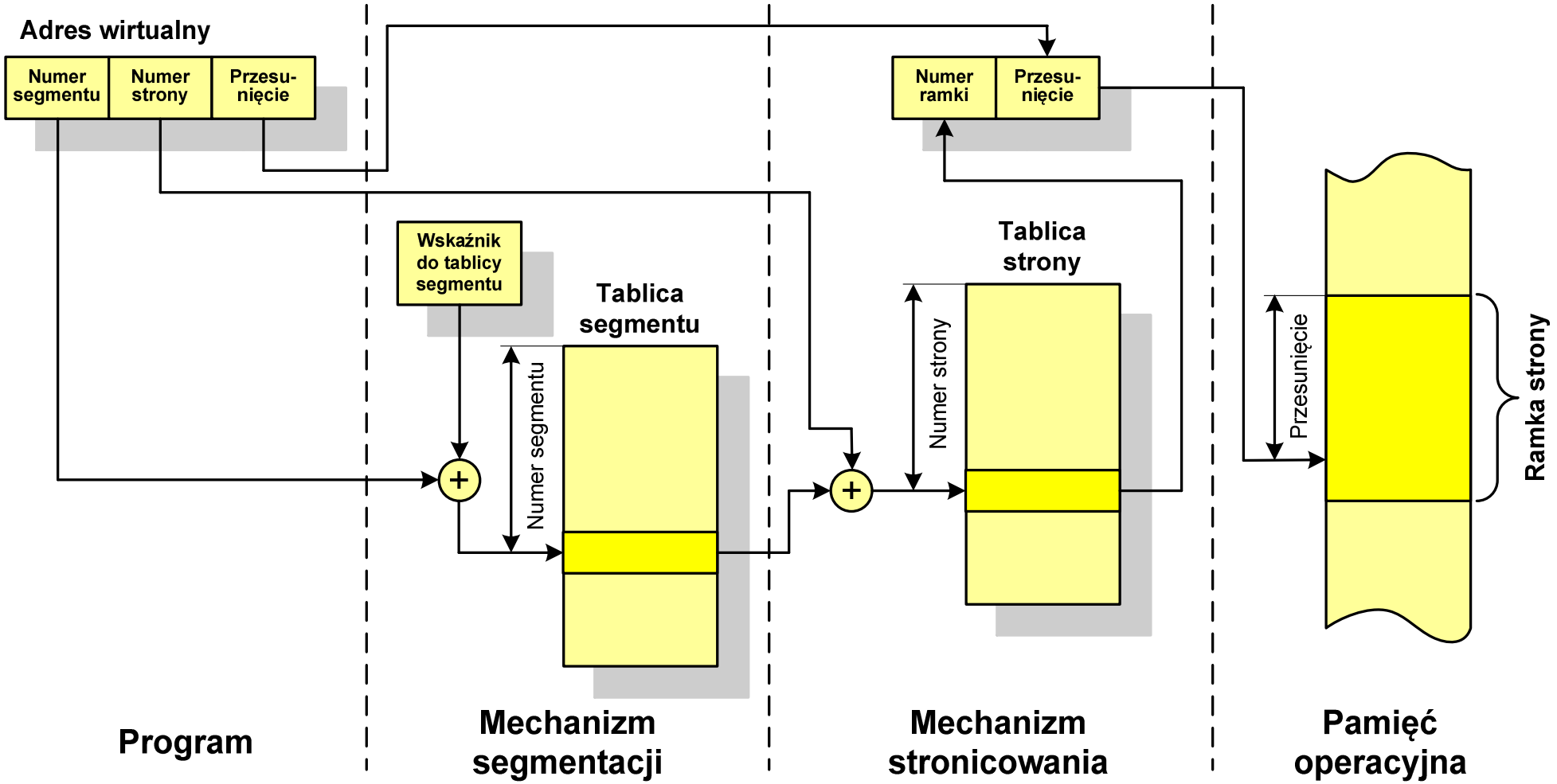
Segmentacja pamięci wirtualnej

- mechanizm odczytania słowa z pamięci obejmuje translację adresu wirtualnego na fizyczny za pomocą tablicy segmentu



Stronicowanie i segmentacja pamięci wirtualnej

- tłumaczenie adresu wirtualnego na adres fizyczny:



Koniec wykładu nr 7

Dziękuję za uwagę!