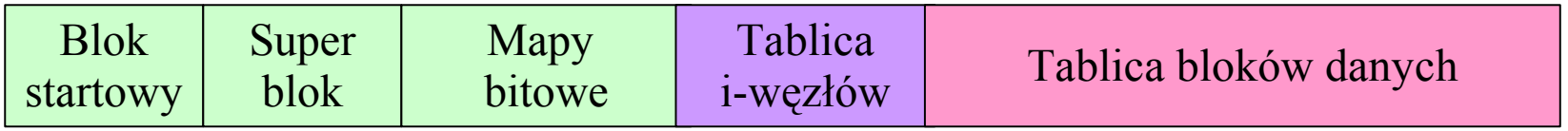
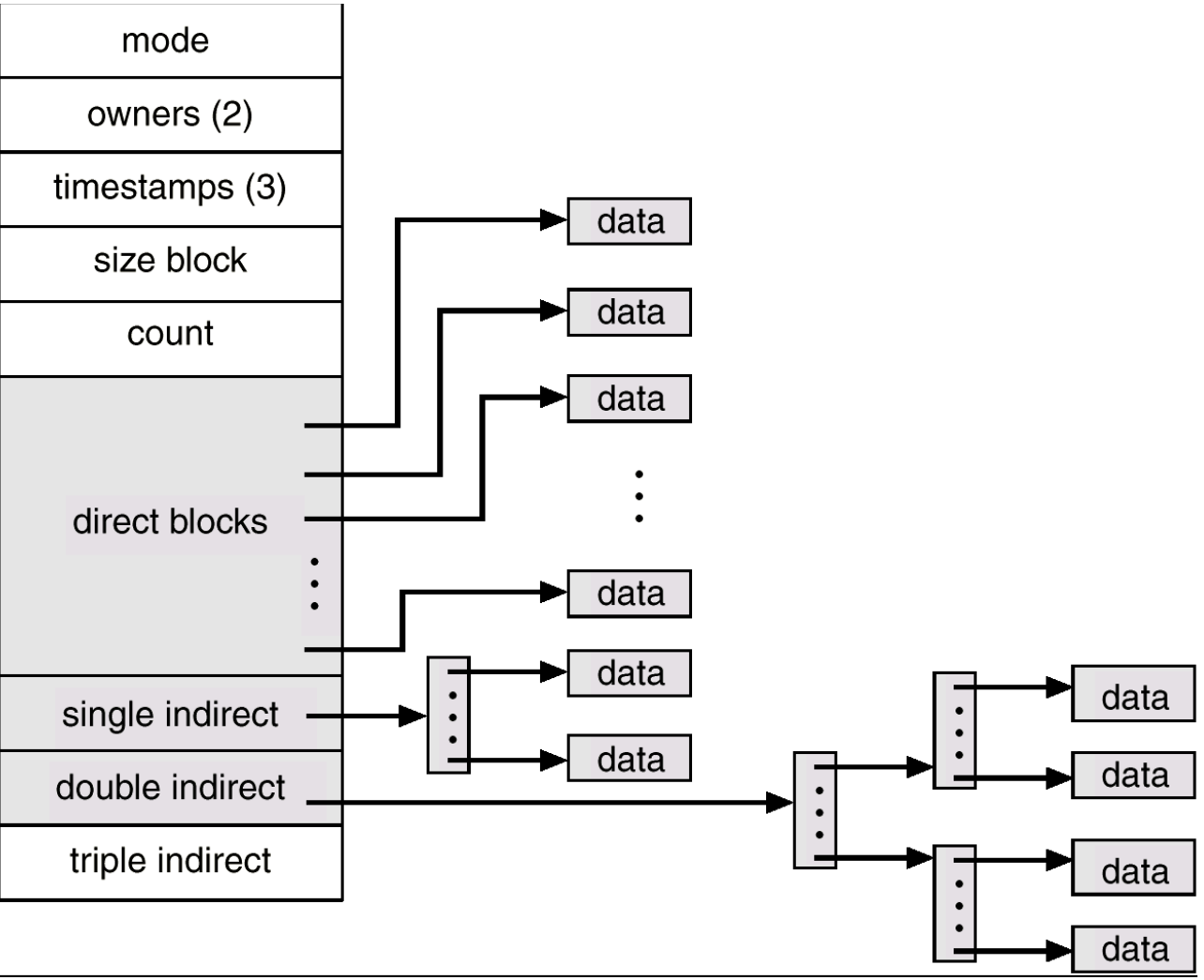
Wykład 8
Systemy plików w Linuksie

Przypomnienie – klasyczny system plików Unixa System V



- Blok startowy – z niego wczytywany jest system.
- Super blok – zawiera informacje o systemie.
- Tablica i – węzłów (ang i-node) – z każdym plikiem na dysku związany jest jego i-węzeł.
  - i-węzeł zawiera wszystkie (prawie – poza nazwą) informacje o pliku.
- Mapy bitowe – zawierają informację, które elementy tablicy i-węzłów oraz tablicy bloków są wolne, a które nie.
  - Jedna dla tablicy i-węzłów, druga dla tablicy bloków danych
  - Mają postać wektora bitów 0/1
- Pytanie: jak znaleźć bloki danych danego pliku.

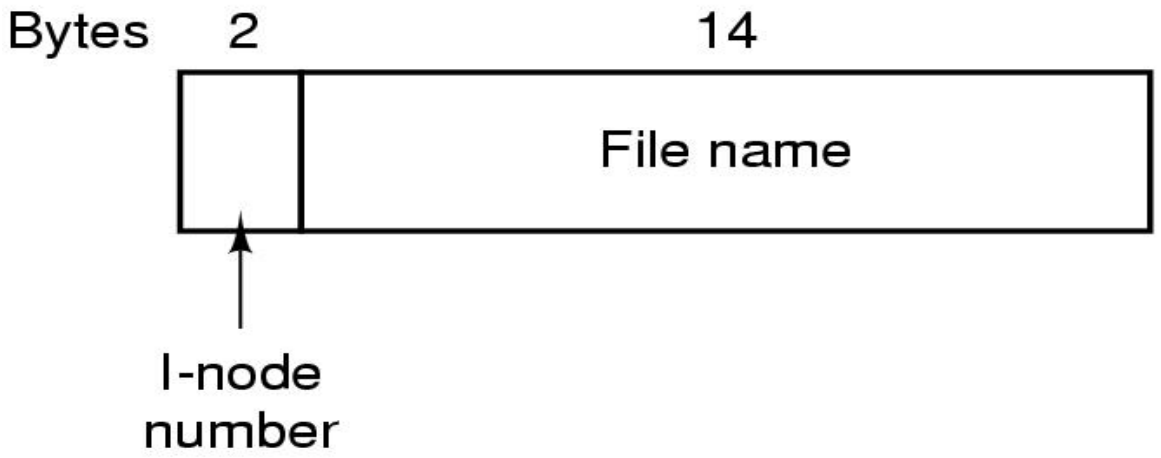
System plików Unixa System V



- Numery pierwszych kilku bloków przechowywane są bezpośrednio w i-węźle (direct blocks). Wystarcza to dla małych plików.

- W przypadku większych plików wykorzystywane są bloki indeksowe. Blok indeksowy jest w całości przeznaczony na numery bloków danych (blok indeksowy pierwszego poziomu) lub innych bloków indeksowych (bloki indeksowe wyższych poziomów)

Katalogi



- Postać (jednej) pozycji katalogu w starszej wersji systemu plików

- Katalog jest plikiem o specjalnej strukturze.
  - Pozycja w katalogu składa się z nazwy i numeru oraz numeru i-węzła.
  - W nowszych systemach plików (255 znakowe nazwy) dochodzi długość.

Warstwa VFS

- Linuks obsługuje wiele systemów plików. Problemem jest w jaki sposób zintegrować je z resztą jądra.
- W tym celu wykorzystuje się warstwę (VFS – virtual file system).
- Warstwa VFS jest interfejsem, który każdy system plików musi udostępnić pozostałym częściom jądra.
- VFS oparta jest na modelu plików Uniksa i podejściu obiektowym.
  - Obiekt super-bloku.
  - Obiekt i-węzła VFS.
  - Obiekt pliku.
- Każdy system plików musi “wyglądać” dla jądra jak system oparty na i-węzłach.
  - W przypadku systemów Unixowych (np minix,ext,xia,ext2) jest to proste.
  - W przypadku innych skomplikowane. Np. W systemie FAT brak i-węzłów a atrybuty pliku przechowywane są w katalogu

Rejestracja systemu plików

- System plików może być implementowany jako moduł. Do rejestracji/odrejestrowania służą funkcje

```
int register_filesystem(struct file_system_type *);
int unregister_filesystem(struct file_system_type *);
```

gdzie struktura file_system_type ma postać:

```
struct file_system_type {
    struct super_block *(*read_super) (struct super_block *, void
   *,int);
    const char *name;
    int requires_dev;
    struct file_system_type * next;
};
```

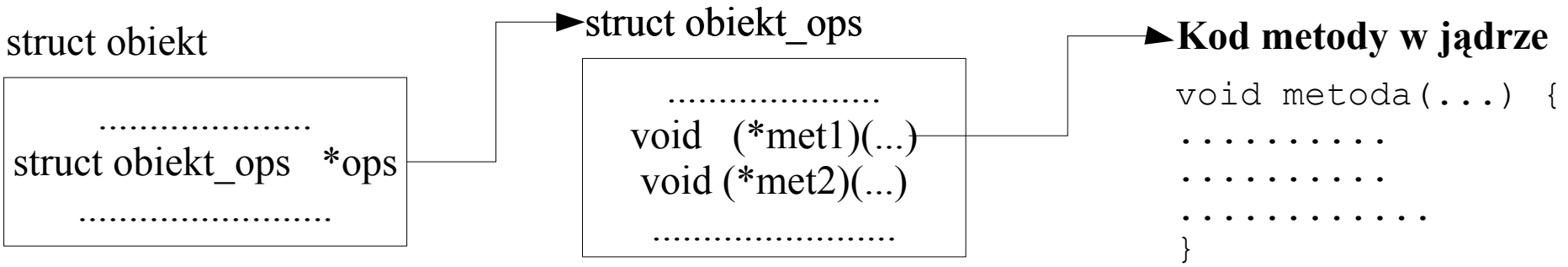
w której dla pojedynczego systemu next==NULL, name jest nazwą systemu plików, requires_dev==1 wtw gdy system wymaga użądzenia blokowego, read_super jest adresem funkcji odczytującej superblok z dysku, postaci:

```
struct super_block *read_super(struct super_block *s,void *data, int
silent)
```

Wypełnia ona egzemplarz struktury super_block wskazanymi przez s danymi z systemu plików i zwraca go jako wynik, lub NULL gdy nastąpił błąd.

  - void * data przechowuje opcje montowania

Implementacja metod VFS



- Jądro jest napisane w C, więc musimy symulować metody wirtualne.
- Funkcję tablicy metod wirtualnych pełni egzemplarz struktury obiekt_ops, na który wskazuje pole ops.
  - Konieczność ustawienia poprawnych wartości pół przy inicjalizacji.
- Wywołanie metody na rzecz obiektu:

```
// Zakładamy deklaracje struct obiekt *pobiekt
pobiekt->ops->met1(pobiekt, ...)
```

- Prawie zawsze pierwszym parametrem metody jest wskaźnik na obiekt (odpowiednik this).

struktura (obiekt) super_block

```
struct super_block {
   kdev_t s_dev;  // numer urządzenia
   unsigned long s_blocksize; // rozmiar bloku dyskowego
   unsigned char s_blocksize_bits; // log o podstawie 2 z rozmiaru bloku
   unsigned char s_lock;     // !=0 gdy obiekt jest zablokowany
   unsigned char s_rd_only; // !=0 gdy system plików tylko do odczytu
   unsigned char s_dirt;    // !=0 gdy obiekt został zmodyfikowany
   struct file_system_type *s_type;
   struct super_operations *s_op; // wskaźnik do tablicy metod obiektu !!!
   struct dquot_operations *dq_op;
   unsigned long s_flags;
   unsigned long s_magic;
   unsigned long s_time;
   struct inode * s_covered; // wskaźnik na punkt zamontowania systemu
   struct inode * s_mounted; // wskaźnik na główny i-węzeł systemu plików
   struct wait_queue * s_wait;
   union { // informacje specyficzne dla danego systemu plików
        struct minix_sb_info minix_sb;
        struct ext_sb_info ext_sb;
        ..............................
        // Każdy system plików ma swoją specyficzną strukturę w tej unii
        void *generic_sbp;
        } u;
};
```

- Struktura przechowuje informacje dotyczące *zamontowanego* systemu plików (np. liczba i-węzłów i bloków danych)

Struktura super_block

- Funkcja read_super powinna wypełnić pola w strukturze, odczytując niezbędne dane z urządzenia s_dev.
  - W szczególności należy ustawić tablicę metod obiektu (pole s_op).
  - Odczytać (lub stworzyć) główny i-węzeł reprezentujący główny katalog systemu plików i adres do niego umieścić w polu s_mounted.
  - Jeżeli pole s_op zostało poprawnie zainicjalizowane to można wykorzystać funkcję *iget*.
- Przy dostępie do struktury super_block należy posługiwać się parą funkcji lock_super/unlock_super o ile w czasie dostępu może nastąpić przełączenie kontekstu.
  - lock_super wywoływana jest przed dostępem , unlock_super po zakończeniu dostępu
  - Realizują semafor binarny (wzajemne wykluczanie) z wykorzystaniem pól s_lock oraz s_wait.
- Pole flags jest logiczną sumą następujących masek bitowych: MS_RDONLY – system tylko do odczytu, MS_NOSUID – nie działają bity SUIG i SGID, MS_NOEXEC – nie można uruchamiać programów, MS_SYNCHRONOUS bezpośredni zapis na dysk.

Metody obiektu superbloku

```
struct super_operations {
 void (*read_inode) (struct inode *);
 int (*notify_change) (struct inode *, struct iattr *);
 void (*write_inode) (struct inode *);
 void (*put_inode) (struct inode *);
 void (*put_super) (struct super_block *);
 void (*write_super) (struct super_block *);
 void (*statfs) (struct super_block *, struct statfs *, int);
 int (*remount_fs) (struct super_block *, int *, char *);
};
```

- Wskaźnik do struktury super_operations jest polem (s_op) w strukturze super_block. Super_operations pełni funkcję analogiczną do tablicy metod wirtualnych w C++.
- read_inode odczytuje i-węzeł o podanym numerze (pole i_ino, urządzenie i_idev).
- notify_change, wywoływana gdy nastąpiła zmiana w strukturze i-węzła. Ważne dla NFS.
- write_inode zapisanie struktury i-węzła
- write_super zapisanie super_bloku na dysku.
- put_super wywoływana podczas odmontowania systemu plików, powinna zwolnić dane związane z super blokiem i nie tylko (np. zwolnić bufory przez brelse i)
- put_inode wywoływane przy zwalnianiu i-węzła. Odpowiedzialna jest za kasowanie plików, jeżeli licznik odwołań do dowiązań do i-węzła osiągnie 0.

Struktura i-node

- Reprezentuje plik albo katalog. Masa pól, z których najważniejsze to:

```
struct inode {
  kdev_t          i_dev;   // urządzenie na którym jest system plików
  unsigned long   i_ino;   //numer i-węzła
  umode_t         i_mode;  // typ pliku oraz prawa dostępu
  nlink_t         i_nlink; // liczba dowiązań do pliku 0 - plik usunąć
  uid_t           i_uid;   // identyfikator właściciela pliku
  gid_t           i_gid;   // identyfikator grupy pliku
  kdev_t          i_rdev;  // urządzenie reprezentowane przez plik specjalny
  off_t           i_size;  // rozmiar pliku w bajtach
  time_t          i_atime; // czas ostatniego dostępu
  time_t          i_mtime; // czas ostatniej modyfikacji
  time_t          i_ctime; // czas utworzenia pliku
  unsigned long   i_blksize; // rozmiar bloku
  unsigned long   i_blocks;  // rozmiar pliku w blokach
  unsigned long   i_nrpages; // liczba stron zawierających plik
  struct inode_operations *i_op; // adresy metod i-węzła
  struct super_block *i_sb;      // adres superbloku
  unsigned short i_count;        // liczba procesów korzystających z i-węzła
  unsigned short i_flags;        // flagi montowania
  unsigned char  i_dirty; // !=0 gdy i-węzeł został zmieniony
  union {
     struct ext2_inode_info ext2_i;
     // **** każdy system plików ma tu swoje dane
     void * generic_ip;
  } u;
```

- Pominięto pola związane z synchronizacją oraz strukturami danych

Przykład wykorzystanie metod obiektowych – algorytm iget

- Chcąc pobrać i-węzeł pliku bądź też katalogu należy wykonać funkcję:

```
struct inode * iget(struct super_block * sb,int nr)
```

- Aby uniknąć wielokrotnych odczytów z urządzenia funkcja ta wykorzystuje pamięć podręczną i-węzłów..
  - Jest ona oparta na funkcjach mieszających i działa podobnie jak podręczna pamięć buforowa. Między innymi wykorzystuje zliczanie odniesień.
  - Kluczem w wyszukiwaniu jest numer i-węzła + numer urządzenia.
- Problem: jak zaimplementować tę funkcję w sytuacji, w której w każdym systemie plików odczyt i-węzła jest robiony inaczej.
- Rozwiązanie tradycyjne (studenci I roku + studenci studiów zaocznych dowolnego roku): w każdej sytuacji, w której potrzebne jest odmienne zachowanie w zależności od systemu plików skonstruuj piękną instrukcję switch.
- Rozwiązanie w Linuksie (państwo): wykorzystaj namiastkę programowania obiektowego.

Algorytm iget - pseudokod

```
struct inode * iget(struct super_block * sb,int nr) {
   struct inode *inode=Przeszukaj tablicę mieszającą();
   if (inode==NULL) {
      inode=Przydziel_nowy_węzeł();
      inode->i_no=nr
      inode->i_dev=sb->dev
      sb->sb_ops->read_inode(inode); // Programowanie obiektowe !!!
      Wstaw_do_tablicy_mieszającej(inode);
   }
   inode->i_count++;
   return inode;
}
```

- Przypomnienie super blok ma pole sb_ops wskazujące na strukturę super_block operations, w której read_inode jest wskaźnikiem na funkcję !!!
- Pominięto: (a) synchronizację – bardzo kłopotliwą (b) szczególne traktowanie punktów montowania (c) pewnie coś o czym zapomniałem – szczegóły w pliku ./fs/inode.c
- Implementując swój system plików korzystasz zawsze z funkcji iget, nigdy nie wywołujesz bezpośrednio read_inode

Algorytm iput

```
void iput(struct inode *inode) {
   if (inode->i_count>1) {
      inode->i_count--;
      return;
   }
   inode->i_sb->s_op->put_inode(inode); // Programowanie obiektowe
   if (inode->i_nlink==0) // plik lub katalog został skasowany
      return;
   if (inode->i_dirt) // i-węzeł został zmodyfikowany
     inode->i_sb->s_op->write_inode(inode); //Programowanie obiektowe
   inode->i_count=0;
   return;
}
```

- i-węzeł pobrany przez iget należy zwolnić przez i-put.
- Pomijamy sporo spraw: (a) zarządzanie wolnymi i-węzłami (b) synchronizację (c) potoki

Metody i-węzła

```
struct inode_operations {
struct file_operations * default_file_ops;
int (*create) (struct inode *,const char *,int,int,struct inode **)
int (*lookup) (struct inode *,const char *,int,struct inode **);
int (*link) (struct inode *,struct inode *,const char *,int);
int (*unlink) (struct inode *,const char *,int);
int (*symlink) (struct inode *,const char *,int,const char *);
int (*mkdir) (struct inode *,const char *,int,int);
int (*rmdir) (struct inode *,const char *,int);
int (*mknod) (struct inode *,const char *,int,int,int);
int (*rename) (struct inode *,const char *,int,struct inode *,const
   char*,int, int);
int (*readlink) (struct inode *,char *,int);
int (*follow_link) (struct inode *,struct inode *,int,int,struct
   inode *);
int (*readpage) (struct inode *, struct page *);
int (*writepage) (struct inode *, struct page *);
int (*bmap) (struct inode *,int);
void (*truncate) (struct inode *);
int (*permission) (struct inode *, int);
int (*smap) (struct inode *,int);
};
```

- default_file_ops tablica metod działających na otwartym pliku

Szybki przegląd metod i-węzła

- create (dir,name,len,mode, res_inode) – utworzenie pliku w katalogu dir i przydzielenie dla niego (za pomocą get_empty_inode) i-węzła. Adres nowego i-węzła zwracany przez res_inode.
- lookup(dir,name,len,res_inode). Przeszukanie katalogu dir w poszukiwaniu pliku o nazwie name. Odnaleziony i-węzeł pliku zwracany jest w res_inode.
- link(oldinode,dir,name,len). Utworzenie nowego sztywnego łącza o nazwie name do pliku oldinode i umieszczenie go w katalogu dir.
- unlink(dir,name,len) Usunięcie pliku nazwie name z katalogu dir.
- symlink(dir,name,len,synname) Utworzenie nowego łączna symbolicznego o nazwie name w katalogu dir wskazującego na ścieżkę synname.
- mkdir(dir,name,len,mode) Utworzenie w katalogu dir podkatalogu o nazwie name i o uprawnieniach mode.
- rmdir (dir,name,len). Usunięcie katalogu name z podkatalogu dir
- mknod(dir,name,len,mode,rdev) Utworzenie pliku specjalnego w katalogu dir o nazwie name, o uprawnieniach mode dla urządzenia rdev

Szybki przegląd metod i-węzła (2)

- rename (odir,oname,olen,ndir,nname,nlen) Przeniesienie pliku o nazwie oname z katalogu odir do katalogu ndir i nadanie mu nazwy nname.
- readlink(inode,buf,size) odczytanie łącza symbolicznego i umieszczenie w buf scieżki, na którą te łącze wskazuje
- follow_link(dir,inode,flag,mode,res_inode). Zwraca w res_inode i-węzeł na który wskazuje łącze symboliczne inode umieszczone w katalogu dir.
- bmap(inode,block). Zamienia numer bloku w pliku na numer bloku n urządzeniu.
  - Mając zaimplementowane bmap możemy skorzystać z uogólnionej funkcji odczytu pliku (generic_file_read).
- truncate(inode). Skraca plik do nowej długości. Pole i_size zawiera nową długość.
- permission(inode,flag) sprawdza prawa dostępu.
- smap(inode,sector) podobnie jak bmap, ale mapuje numer sektora (dla systemów FAT)

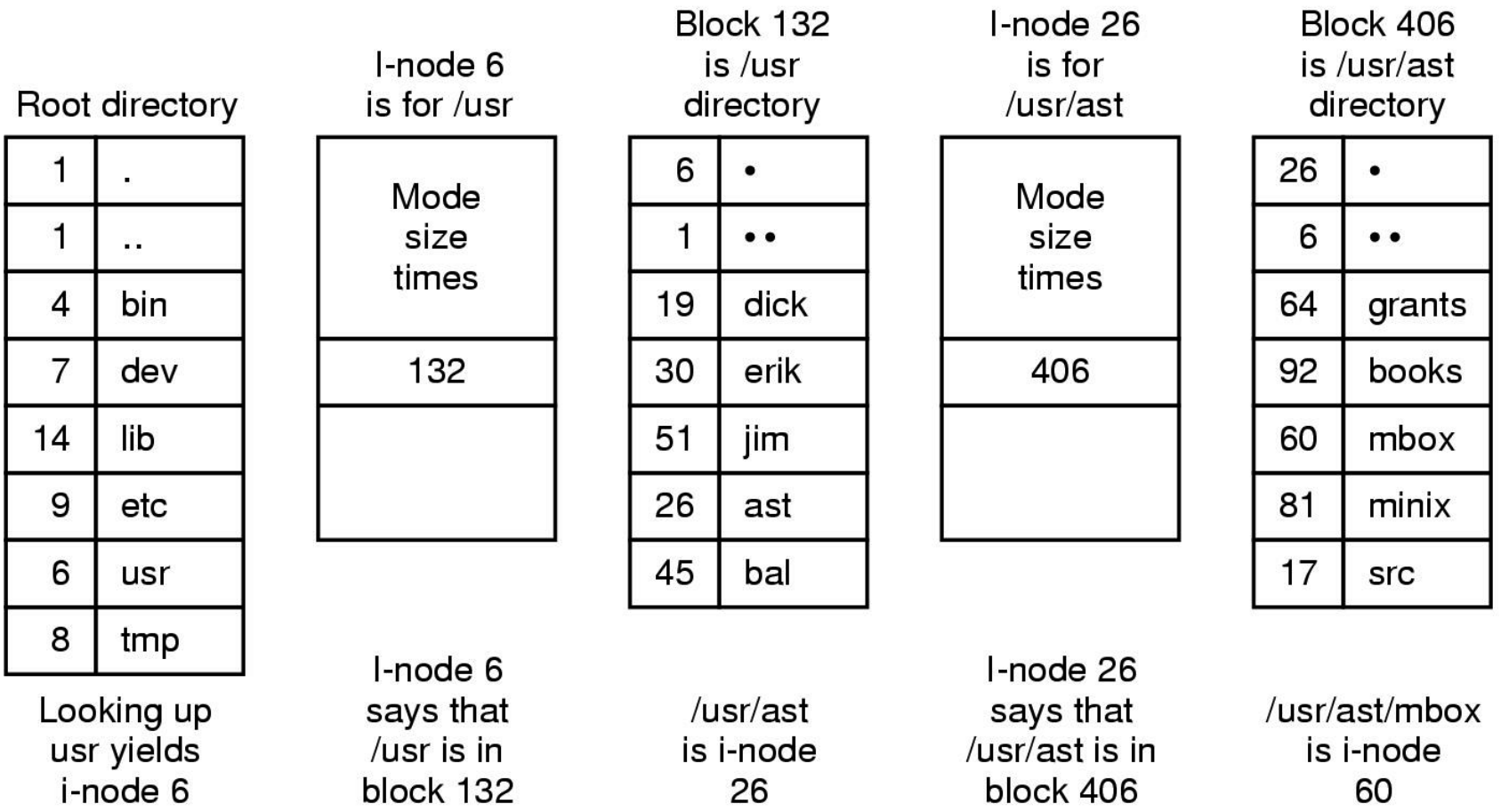
Metody i-węzła: uwagi

- Pierwszy parametr jest zawsze i-węzłem na rzecz którego uruchamiamy metodę:

```
dir->i_ops->unlink(dir,name,len);
inode->i_ops->permission(inode,flags);
```

- Niektóre metody nie mają sensu dla niektórych rodzajów i węzłów.
  - lookup nie ma sensu w przypadku nie-katalogów
  - readlink nie ma sensu w przypadku nie-łączy symbolicznych
- Dlatego też zazwyczaj implementacja systemu plików deklaruje kilka struktur inode_operations: odzielną dla katalogów, oddzielną dla zwykłych plików itd...
- Po wczytaniu i-węzła (metoda read_inode obiektu super bloku) ustawiana jest odpowiednia wartość pola i_ops.
-

Przykład wykorzystanie metod: algrytm namei



- Tłumaczenie nazwy scieżki np. /usr/ast/mbox na numer i-węzła.
- W Linuksie podczas tłumaczenia możemy poruszać się po różnych systemach plików.

Metody i-węzła upraszczają implementację namei

- Można to zrobić wykorzystując następującą koncepcję:

```
struct inode *inode=katalog_startowy();
while (koniec_ścieżki()) {
   char *s=Pobierz_fragment_scieżki();
   // Programowanie obiektowe poniżej
   inode->i_ops->lookup(inode,s,len,&inode);
}
```

- Oczywiście rzeczywista implementacja jest o wiele bardziej skomplikowana
  - Dowiązania symboliczne.
  - Punkty montowania.
  - Plik ./fs/namei.c ma 20 KB.