受講にあたっての注意事項

本講義の全部あるいは一部の録画・録音および複製ならびに再配布を、厳に禁じます。

大阪大学 基礎工学部
SCHOOL OF ENGINEERING SCIENCE

オペレーティングシステム

3章 メモリ管理

3.3 ファイル管理ーファイルシステムー

3.3.5 ファイル割付 3.3.6 ファイル保護

大阪大学大学院情報科学研究科
村田正幸
murata@ist.osaka-u.ac.jp
http://www.anarg.jp/

3.3 ファイル管理ーファイルシステムー
3.3.5 ファイル割り付け

[1] ボリューム

定義3.15(ボリューム)
ファイルを格納するファイル装置において、独立にアドレス指定できる、すなわち、独立したアドレス空間を構成できる、単位メモリ領域をボリュームという

- (A) 逐次アクセスファイル装置
  - 磁気テープメディア1巻、リールともいう
- (B) 直接アクセスファイル装置
  - ハードディスクドライブ装置やフラッシュメモリの1区域（パーティション）
  - CD-ROMやDVDのメディアの1枚
- ボリューム
  - ファイル装置におけるメモリ領域区分
  - 独立して論理的なアドレス指定が可能な、すなわち、独立して論理的なアドレス空間を構成できる、物理的なファイル装置
  - OSだけではなく、ユーザやユーザプログラムからも「仮想化されたファイル装置」に見える
  - OSがファイル装置の管理機能として、論理的なボリュームと物理的なファイル装置との対応付けを実現している
  - ファイル装置の仮想化による独立したメモリ領域としてのボリュームを実現している
  - 物理的なハードウェア機構（ファイル装置）の隠ぺいの実現例

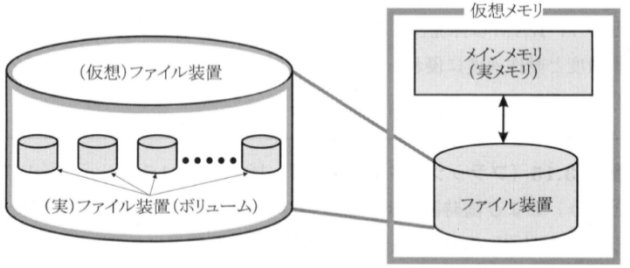


図 3.31 仮想ファイル装置と仮想メモリ

3.3 ファイル管理ーファイルシステムー
3.3.5 ファイル割り付け

[1] ボリューム（続き）

- OS機能の観点から
  - ファイルシステムは以下を実現する
    - (1) ファイル構造
    - (2) ファイルアクセス方式
    - (3) ディレクトリ
  - ボリュームは、ファイルシステムに対して
    - 物理的なファイル装置を、複数個の独立した論理的なファイル装置に見せかけて提供する
- ファイル装置の管理→3.3.7項
  - 仮想化した論理的ファイル装置であるボリュームと実際の物理的ファイル装置との対応付け

3.3 ファイル管理ーファイルシステムー
3.3.5 ファイル割り付け

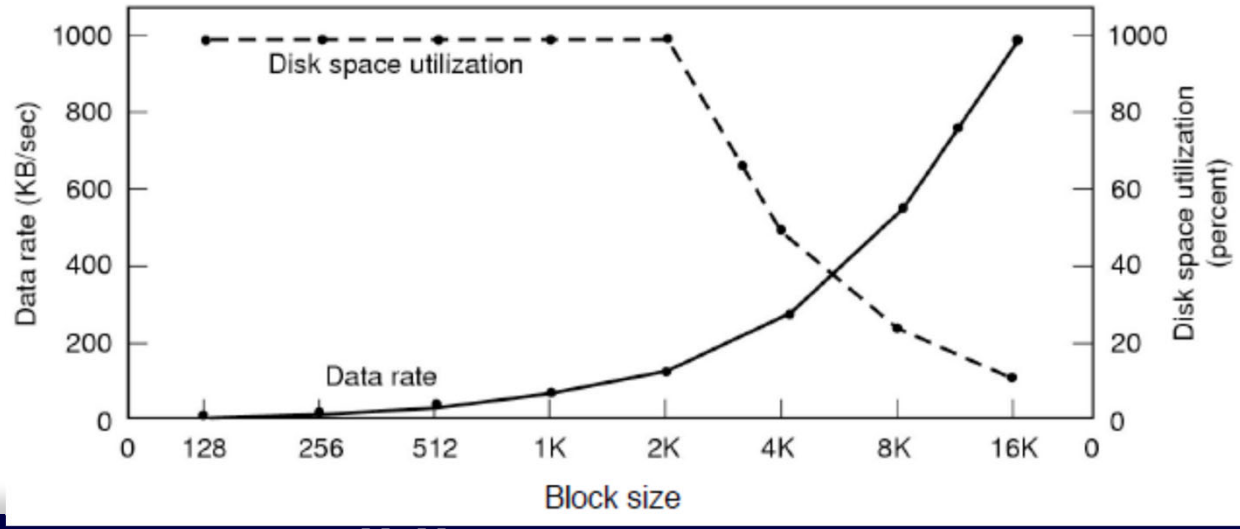
[2] ファイル割り付け

定義3.16（ファイル割り付け）
ファイル装置のどこにファイルを格納するか、すなわち、ファイルとファイル装置（実際にはボリューム）との対応付けをファイル割り付けという

- ファイル装置上での領域管理機能
  - ファイルシステムがユーザプログラムやユーザに提供するファイル割り付け機能
  - ディレクトリが「ファイルとボリューム上での格納場所との対応付け表」、すなわち、「ファイル割り付け表」を保持
- ファイル割り付け
  - ディレクトリによって管理・制御する
  - 固定長のブロック単位
  - 固定長領域割り付け：大規模領域の管理を一定時間で高速に行うため
- ブロックサイズ
  - 大きくしすぎると内部フラグメンテーションが発生する
  - 小さくしすぎると、転送回数が増えて、結果としてアクセス時間が長くなる
  - ファイルシステムにおけるファイル割り付け機能の設計では、ブロックサイズについてのトレードオフを適切に調停することが要点となる

ブロックサイズの決定

- ブロックサイズ小⇒大
  - データのアクセス効率は向上：1度のアクセスでより大きなデータにアクセス可能
  - ディスクの利用効率は低下する：内部フラグメンテーションの発生
- ブロックサイズの決定
  - ファイルのサイズやディスクの利用状況から決定される
  - UNIX: 1KB （1984年の中央値：1KB、2000年：1.68KB）
  - Windows: 4KB～32KB



演習問題3.4

- ファイルが5個ある。サイズは、0.5KB, 1KB, 4KB, 5KB, 120KBとする
- ブロックサイズが2KBと16KBの場合で、それぞれ格納に必要なディスク領域はいくらか？

3.3 ファイル管理ーファイルシステムー
3.3.5 ファイル割り付け

[2] ファイル割り付け（続き）

- ファイル割り付けは、ボリューム上での領域管理
  - (A) 連続ファイル割り付け
    - ファイルを構成するブロックを、順に、ボリュームの連続する、すなわち、アドレス順の領域に割り付ける
    - 論理的に設定する割り付け単位を、物理的に連続する領域に割り付ける「連続領域割り付け」方式
  - (B) 不連続ファイル割り付け
    - ファイルを構成するブロックを、ばらばらに、ボリュームの不連続、すなわち、任意アドレスの領域に割り付ける
    - 論理的に設定する割り付け単位を、物理的に不連続な領域に割り付ける「不連続領域割り付け」方式

3.3 ファイル管理ーファイルシステムー
3.3.5 ファイル割り付け

[3] 連続ファイル割り付け

- 特徴
  - ファイルブロックをそのまま順に、ボリュームの連続する物理ブロックに割り付ける
  - 割り付け先のボリュームの領域は、先頭ブロック番号とブロック総数によって識別し管理できる
  - ディレクトリは、(1)ファイル名、(2)先頭ブロック番号、(3)ブロック総数の3項目を保持すればよい
  - 連続ファイル割り付けは、逐次アクセスファイルとの親和性が高く、逐次アクセスファイル装置のボリュームに適用される

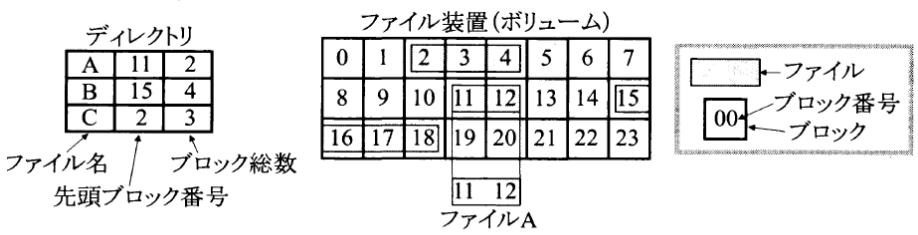
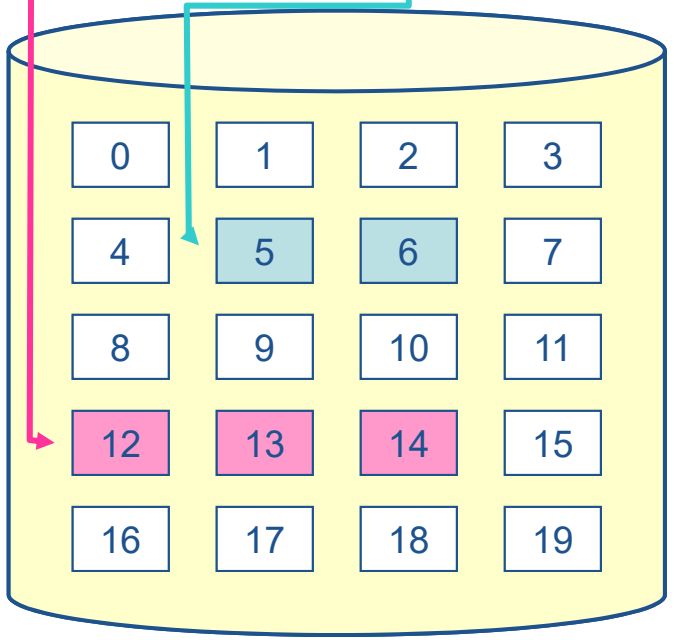


図 3.27 連続ファイル割り付け（例）

ディレクトリ

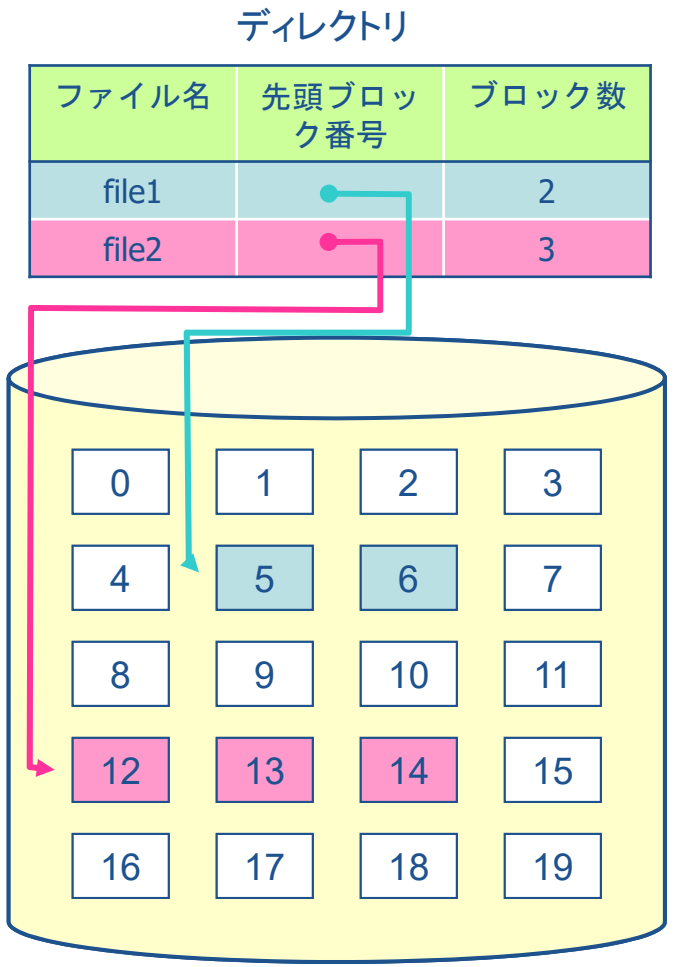
ファイル名	先頭ブロック番号	ブロック数
file1		2
file2		3



3.3 ファイル管理ーファイルシステムー
3.3.5 ファイル割り付け

[3] 連続ファイル割り付け（続き）

- 長所
  - 物理的に連続して、すなわち、物理アドレス順で格納するので、高速アクセスが可能
  - ディレクトリに、先頭ブロック番号とブロック総数のみを保持しているので、ディレクトリの作成や操作が容易
- 短所
  - ユーザプログラムやユーザは、ファイル生成時に、ファイルサイズ（ブロック総数）を指定することが必須となる
  - 外部フラグメンテーションが発生し易い。空き領域の詰め直し（デフラグ）が必要



3.3 ファイル管理ーファイルシステムー
3.3.5 ファイル割り付け

[4] 不連続ファイル割り付け

- 特徴
  - ファイルブロックをばらばらにして、ボリュームの任意すなわち不連続の物理ブロックに割り付ける
  - 直接アクセスファイルとの親和性が高い
  - 直接アクセスファイル装置のボリュームに適用する
- 代表例

  (B-1) リンクファイル割り付け
  - 各物理ブロックに「次の、すなわちリンクするブロック番号（ポインタ）」を付加する
  - ファイル装置に割り付けるファイルは、ファイルブロックのリストとなる

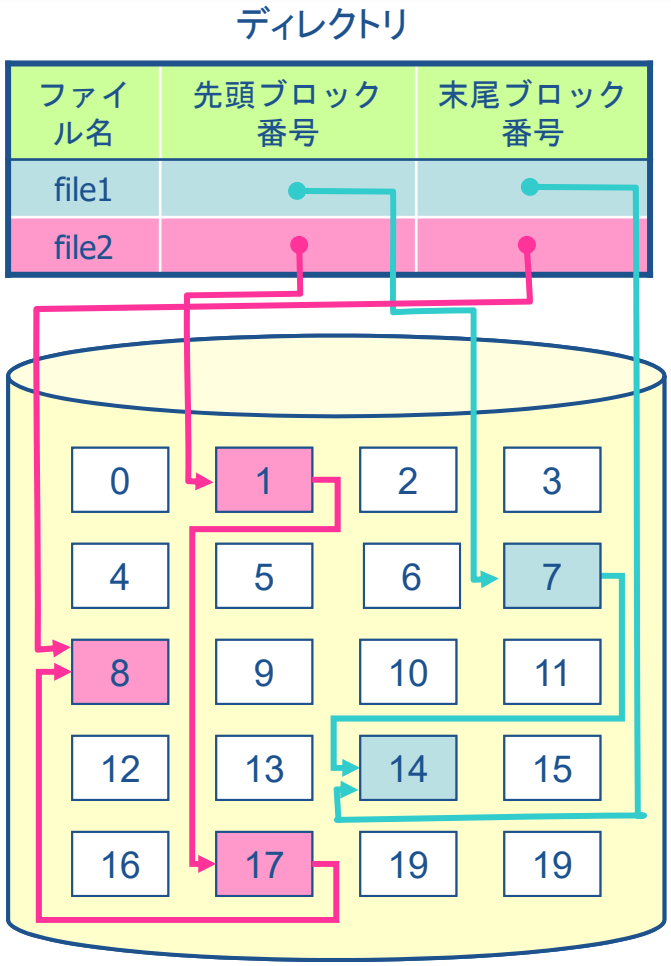
  (B-2) インデックスファイル割り付け
  - すべてのポインタを、インデックスとして、特定のブロックで保持する
  - インデックスブロックを参照することによって、ファイルへのポインタを得る

3.3 ファイル管理ーファイルシステムー
3.3.5 ファイル割り付け

[5] リンクファイル割り付け(B-1)

- 特徴
  - ボリュームに割り付けるファイルは「ファイルブロックのリスト」
  - 各ブロックが「次のすなわちリンクするブロック番号(ポインタ) を保持
  - ディレクトリは、(1) ファイル名、(2) 先頭ブロック番号、(3) 末尾ブロック番号を保持
- 長所
  - 外部フラグメンテーションは発生しない
  - ユーザは、ファイル生成時に、ファイルサイズを指定する必要がない
- 短所
  - 各物理ブロックにポインタのための領域が余分に必要
  - ポインタの破壊(リンク切れ)はファイルの喪失などの致命的な障害を招く
  - リスト途中の特定ブロックへのアクセスは、リストたどりが必須となり、遅い
- 例
  - FAT (File Allocation Table)

図 3.28 リンクファイル割り付け (例)



3.3 ファイル管理ーファイルシステムー
3.3.5 ファイル割り付け

[6] インデックスファイル割り付け(B-2)

- 特徴
  - すべてのポインタをインデックスブロック(ボリュームの目次と見なせるので、VTOC (Volume Table Of Contents)ともいう)として保持
  - ディレクトリは、(1)ファイル名、(2)インデックスブロック番号の2項目を保持
- リンクファイル割り付けと比較した長所
  - インデックスブロックによって特定ブロックへ直接アクセスできるので、リンクファイル割り付けの短所を解消できる
- 短所
  - インデックスブロックのための領域が余分に必要
- 例
  - UFS (Unix File System)

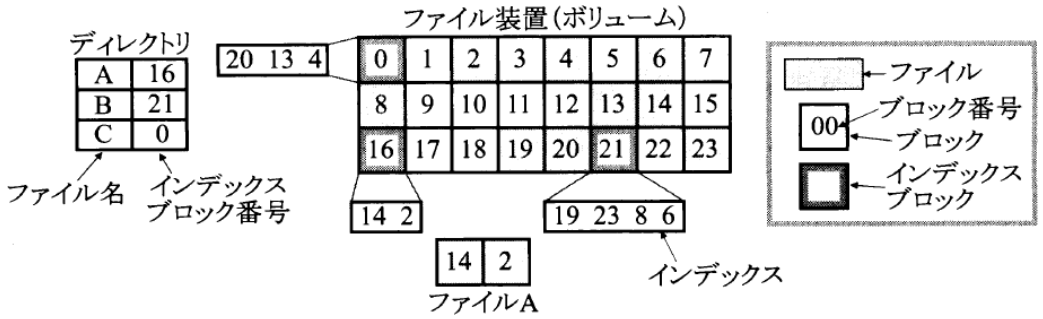
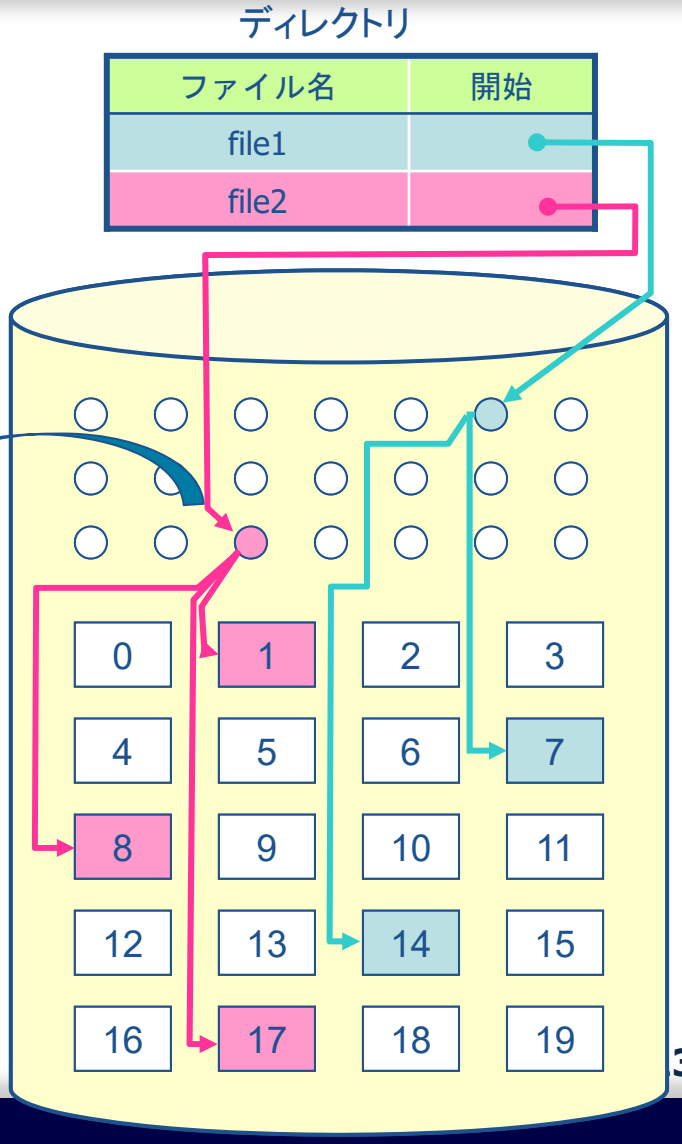
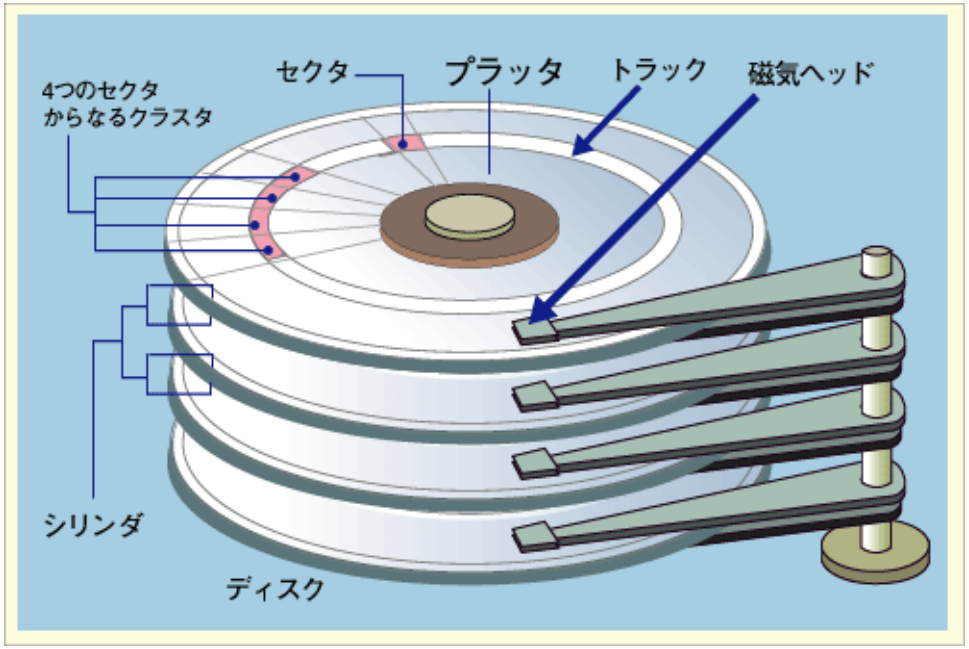


図 3.29 インデックスファイル割り付け(例)



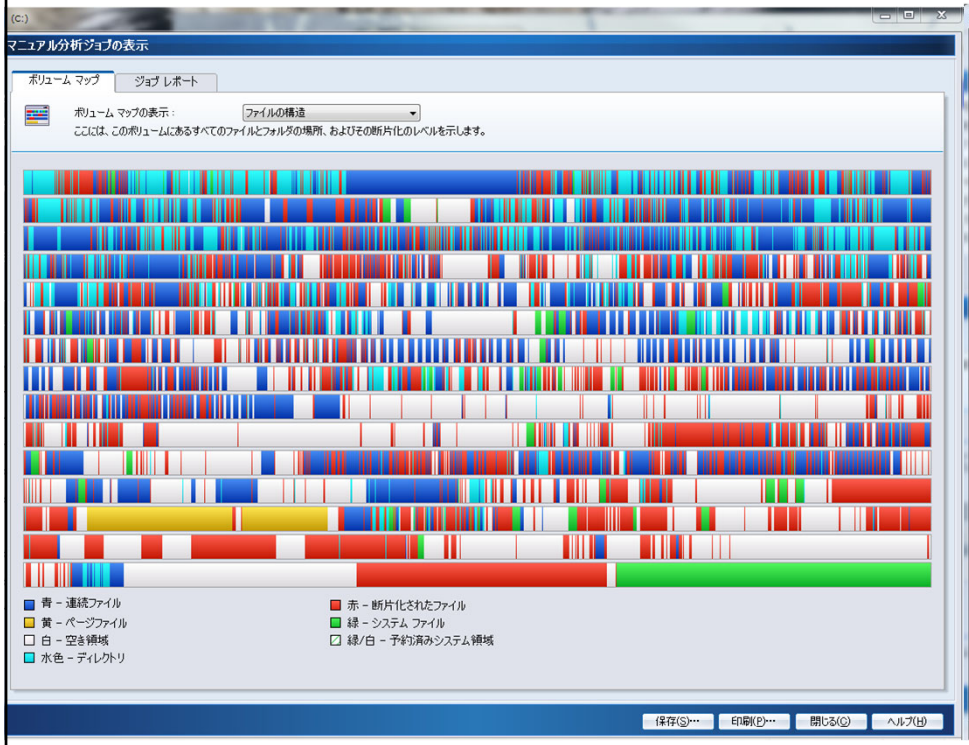
不連続ファイル割付とハードディスクの関係

- 不連続ファイル割付は、直接ファイル割付との親和性が高く、ファイルブロックをばらばらにして、ボリュームの任意すなわち不連続の物理ブロックに割り付けることが可能
- 実際には、
  - 異なるトラヒックに、同じファイルに属するブロックが分かれて格納されていると、磁気ヘッドをスイングさせて、格納されているトラヒックを探す必要がある。
  - さらに、異なるシリンダに格納されていると、用いる磁気ヘッドを切り替える必要がある
- これらのことから、不連続ファイル割付であっても、ファイルブロックをばらばらに格納せず、連続して同じトラックに格納するほうがほんとうはよい。

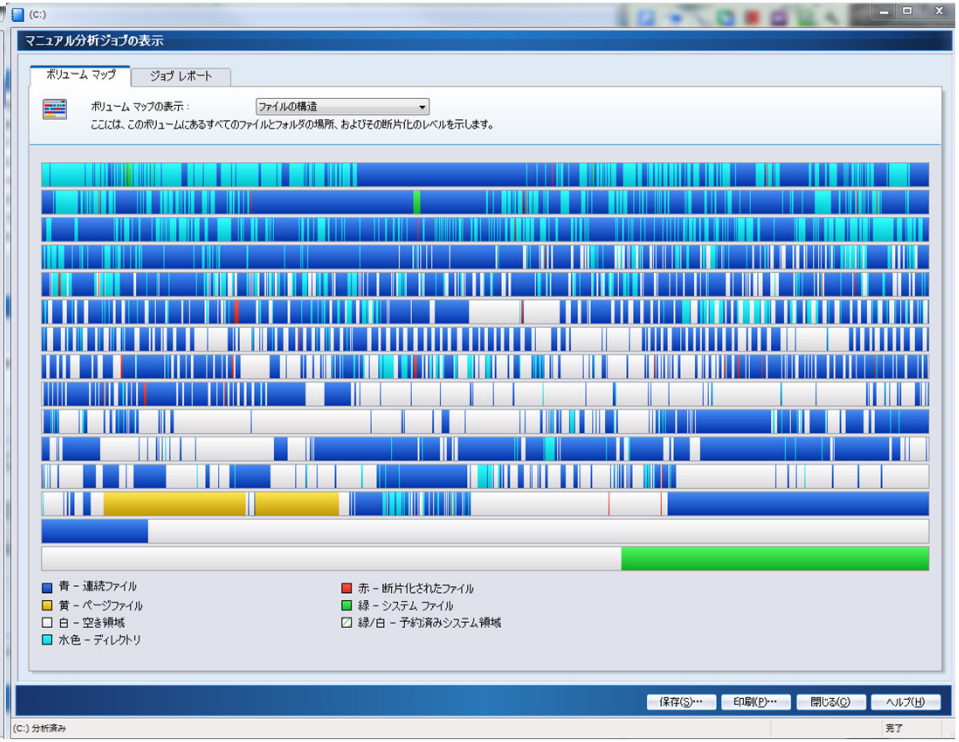


ディスクのデフラグはなんのため？

デフラグ前



デフラグ後



3.3 ファイル管理ーファイルシステムー
3.3.5 ファイル割り付け

[7] ファイルシステムの実例ー不連続ファイル割り付けー

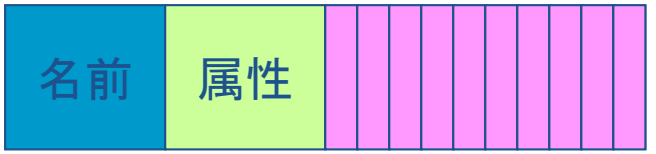
- 現代のOSのほとんどが不連続ファイル割り付けを採用している
  - 不連続ファイル割り付けの短所である「余分な領域の必要性」や「リストアクセスの遅さ」に対しては、現代のコンピュータシステムの高性能ハードウェア機構によって防げる
  - 現代の主要なファイル装置は不連続ファイル割り付けとの親和性が高い直接アクセスファイル装置
- 例
  - (1) FAT (File Allocation Table)
    - リンクファイル割り付けにおけるブロックを構成する「ファイル内容」と「次ブロックへのポインタ」とを分離
    - 次ブロックへのポインタの方を「FAT」と呼ぶリストとして一括保持
    - ポインタが32ビットであるFAT32では、およそ4ギガ個のブロックを対象に管理できる。Windowsで採用されている
  - (2) UFS (UNIX File System)
    - ディレクトリの項目としてインデックスブロックそのものを取り込んだ、iノードという高機能ディレクトリによって管理
    - UNIXが採用している
  - (3) NTFS (NT File System)
    - インデックスファイル割り付けに部分的に連続領域割り付けを採用
    - 高セキュリテイおよび高信頼性機能を付加
      - ハードディスクの書き込み内容の記録を残す
      - アクセス権限をファイルやフォルダごとに設定できる
    - インデックスそのものも木構造ディレクトリである
    - 180億ギガ個のブロックを対象に管理できる
    - Windowsが採用している

ディレクトリの実装

- ディレクトリ
  - ファイル名からファイルが格納されているブロックを知る方法
- ファイルの属性（ファイルの所有者、日時、ファイルサイズなど）をどこにおくか？
  - ディレクトリに直接おく
  - ファイルの占めるブロック番号もおく
  - ファイル側におく（UNIXのi-ノード）
- ディレクトリエントリの保持の方法
  - リスト
    - リストを辿る時間が大きくなる
  - ハッシュ法

先頭のブロック番号

FAT



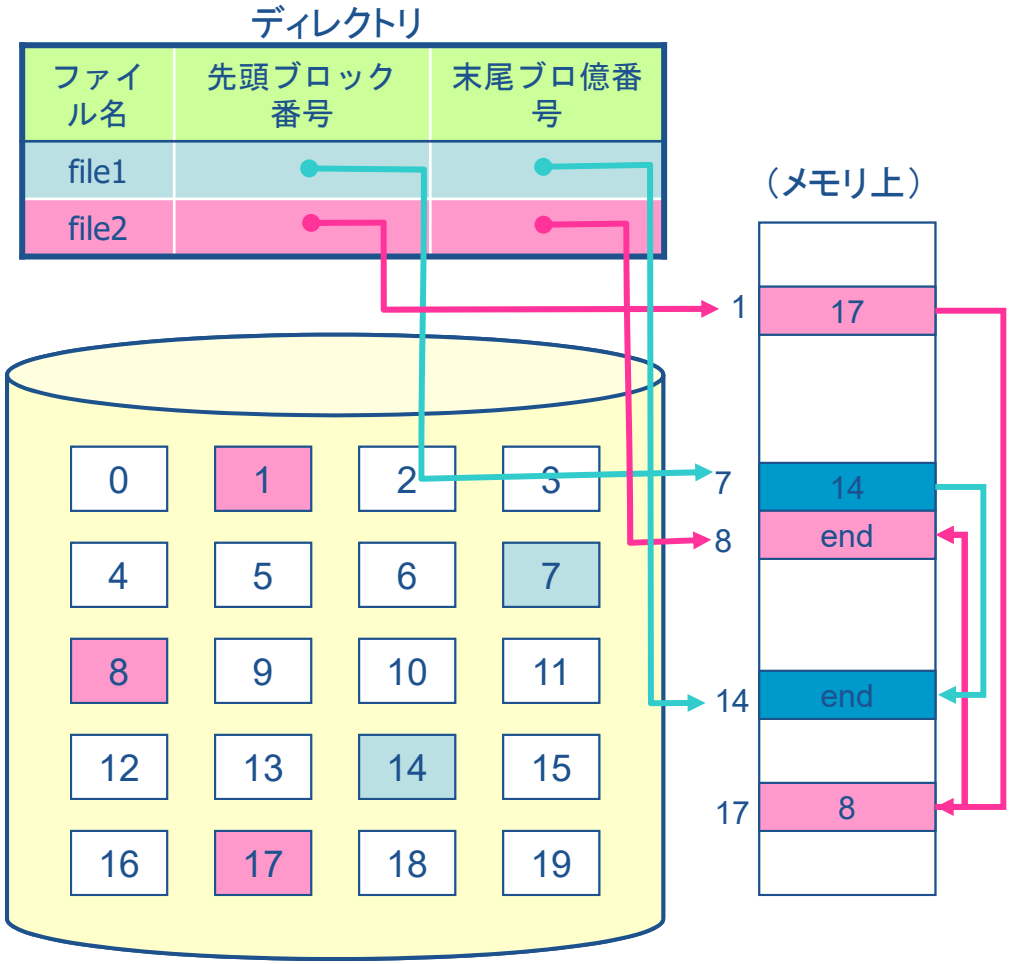
ブロック番号

UFS

i-node番号

FATの例

- FATでは、ポインタのリンク情報を別に索引テーブルとしてメモリ上においている
- 利点
  - 直接アクセスする場合に、ブロックアドレスを知るためにディスクにアクセスする必要はなくなり、高速化される
- 欠点
  - 索引テーブル全体をメモリにおく必要がある
    - 大容量ディスクではブロック数が多くなり、索引テーブルも大きくなる
    - 索引テーブルを小さくしようとすれば、ブロックサイズが大きくなる。ファイル転送の効率はよくなるが、内部フラグメンテーションの度合いが大きくなる



第14回（5月28日2時限）ミニレポート課題

32bit FATの場合、すなわち、1ブロック32ビットの場合、ディスクブロックアドレスは 2^{32}-1までとなる

1. 64GBディスクを仮定した場合、ブロックサイズ2KB、16KBの時のFATサイズはいくらになるか
2. ブロックサイズを小さくした場合の利点、および、問題点をそれぞれ挙げよ。

第13回、第14回のミニレポート課題の解答は6月4日にCLEに掲載します。

締切：6月2日（水）
CLEで提出

FATのディレクトリ構造

- Windowsでは、各エントリは以下のような構成になっている。

8	3	1	1	1	2	2	2	2	2	2	2	4 [B]
ファイル名	拡張子	属性	フラグ	作成 10^{-2} 秒	作成時間	作成日付	参照日付	ブロック番号の上位	変更時刻	変更日付	1番目のブロック番号	大きさ

（a）ディレクトリ構成

ロングファイル名

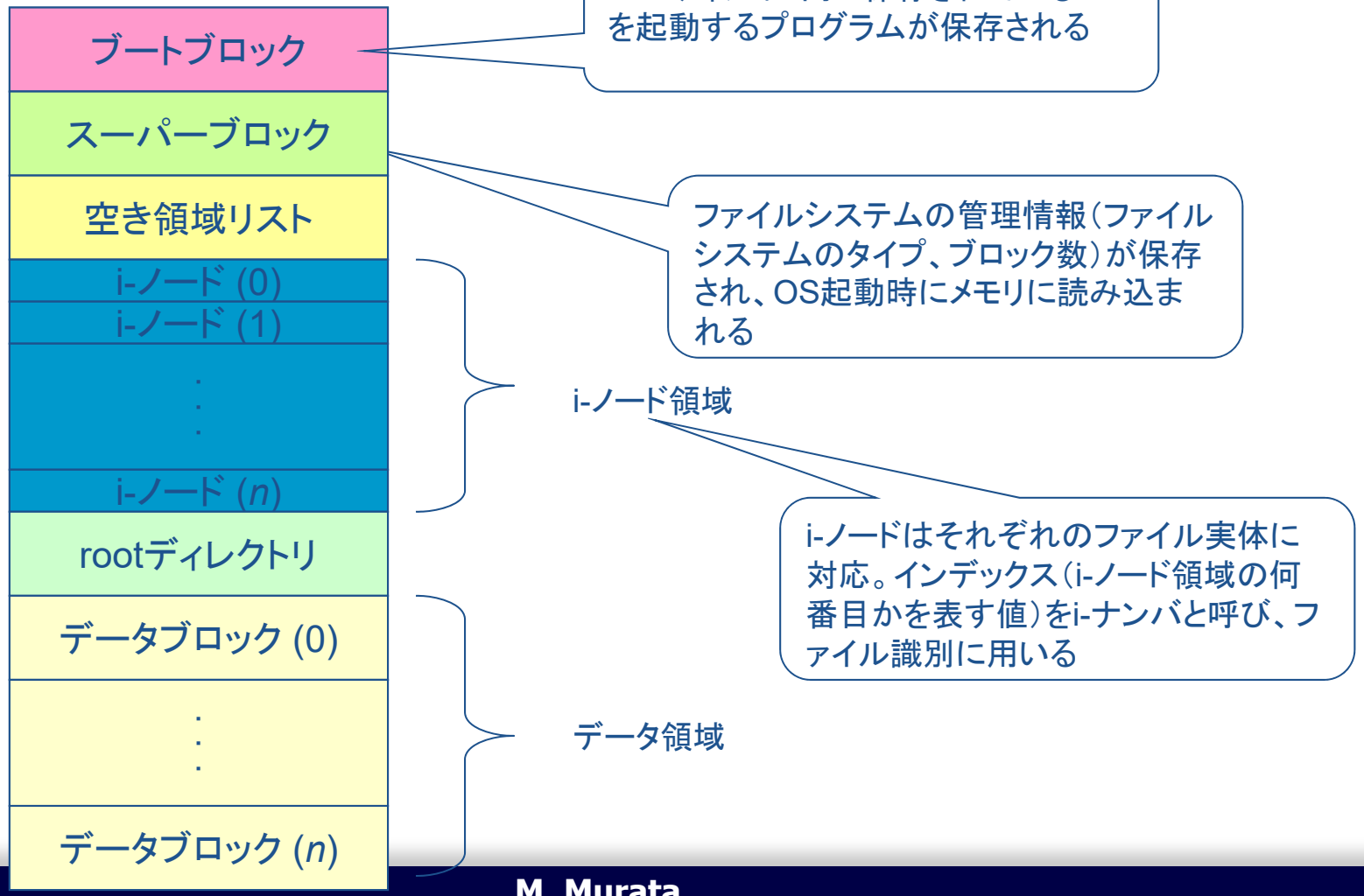
1	10	1	1	1	12	2	4 [B]
番号	ファイル名	0x0f		チェックサム	ファイル名		ファイル名

（b）Windows 98 で導入されたファイル名を保存するエントリー部

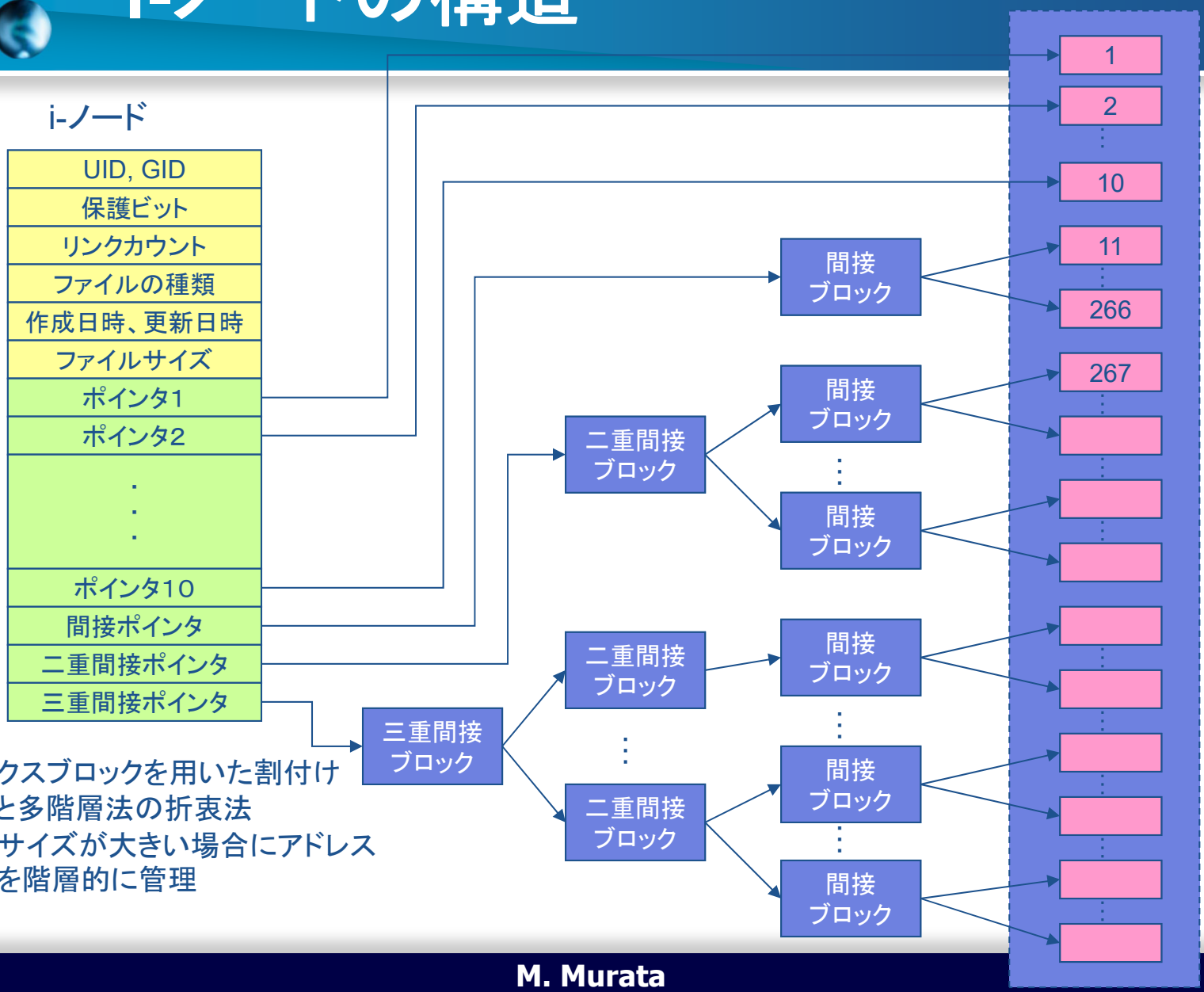
- また、信頼性向上のためにFATは2重化されている。

UNIXの実装

- ファイルシステムの構造



i-ノードの構造



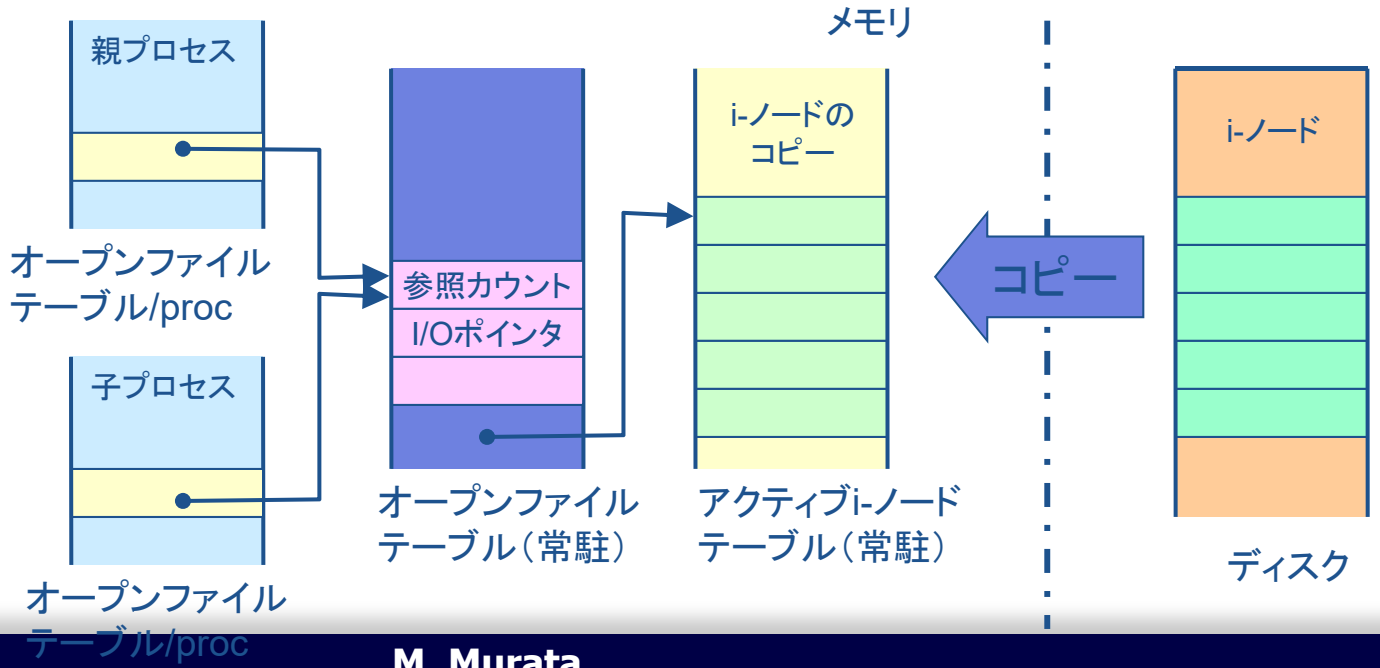
インデックスブロックを用いた割付け
リスト法と多階層法の折衷法
ファイルサイズが大きい場合にアドレスポインタを階層的に管理

演習問題3.5

- UNIXファイルシステムにおいて以下を仮定する
  - データブロックサイズ：1KB
  - ディスクアドレスは32bit
- 1KBのファイル格納に必要なデータブロックはいくらか？
- 100KBのファイル格納に必要なデータブロックはいくらか？

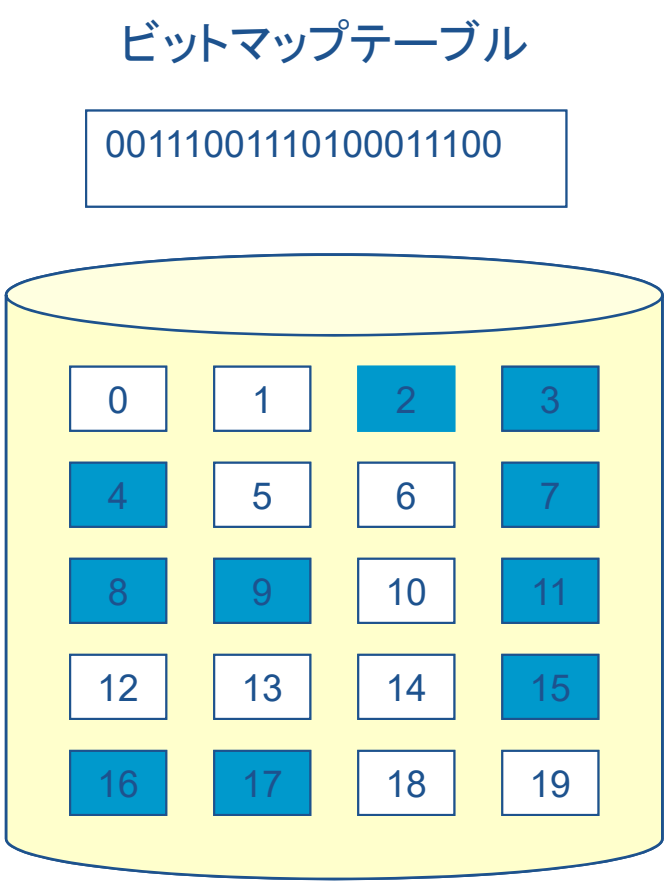
ディレクトリの構造

- ファイル名とi-ナンバの対応付けのみ
- 16バイト固定長のディレクトリエントリ
  - ファイル名 14B、i-ノード番号 2B
    - ファイルシステムの制限になる
- 4.4BSD FFS (Fast File System)
  - ディレクトリにファイル名の長さフィールドを持たせ、可変長 (256B)
  - i-ノード番号 4B



空き領域の管理

- 空きディスクブロックをどのように管理するか
  - 高速に探したい
  - 連続領域のほうがよい
- ビットマップ
  - ブロック個数分のビットを用意し、ブロックの状態を0(使用中)、1(空き)であらわす
  - 利点
    - ビットマップ全体をメモリにおければ効率的
    - 連続した空き領域を探すことも容易
  - 欠点
    - ディスクの大容量化に伴って、ビットマップのメモリ量も増える。メモリに置けない場合は非効率的
    - ディスクが満杯近くになるとビットマップテーブルを広範囲にわたって走査しなければならない



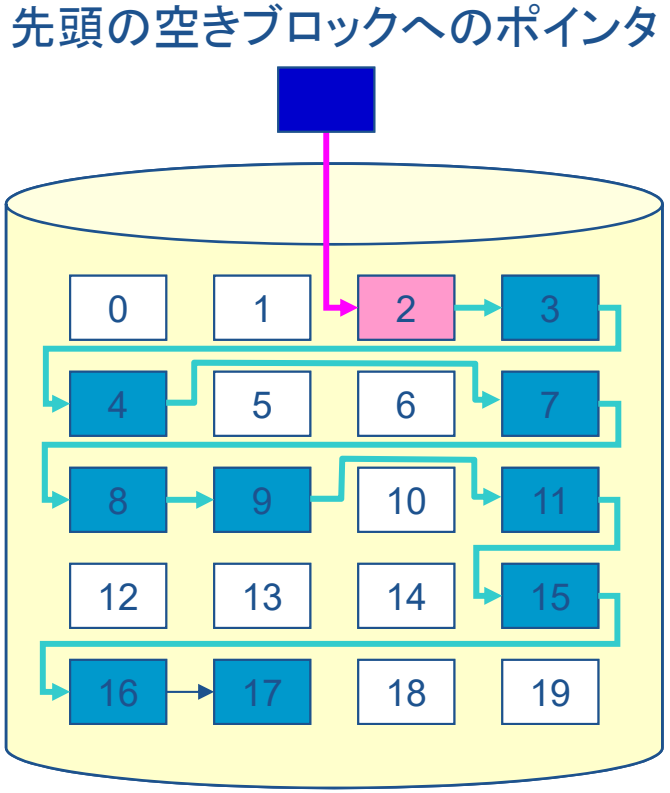
演習問題3.6

- 32GBディスク、8KBブロックの場合、ビットマップテーブルの大きさはいくらになるか

連結リスト

- ブロック内にポインタを付与し、空きブロックを連結する
- 利点
  - 先頭の空きブロックだけを指し示すだけでよいので、メモリ容量の節約になる
  - 要求時には先頭の空きブロックを返せばよいので、効率はよい
  - ファイルが削除されたときは、そのブロックをリストの先頭に加えればよい
- 欠点
  - 信頼性が低い
  - 連続した空き領域かどうかわからない⇒不連続割り当てになる
    - 連続させようとする場合にはポインタの張替えが必要

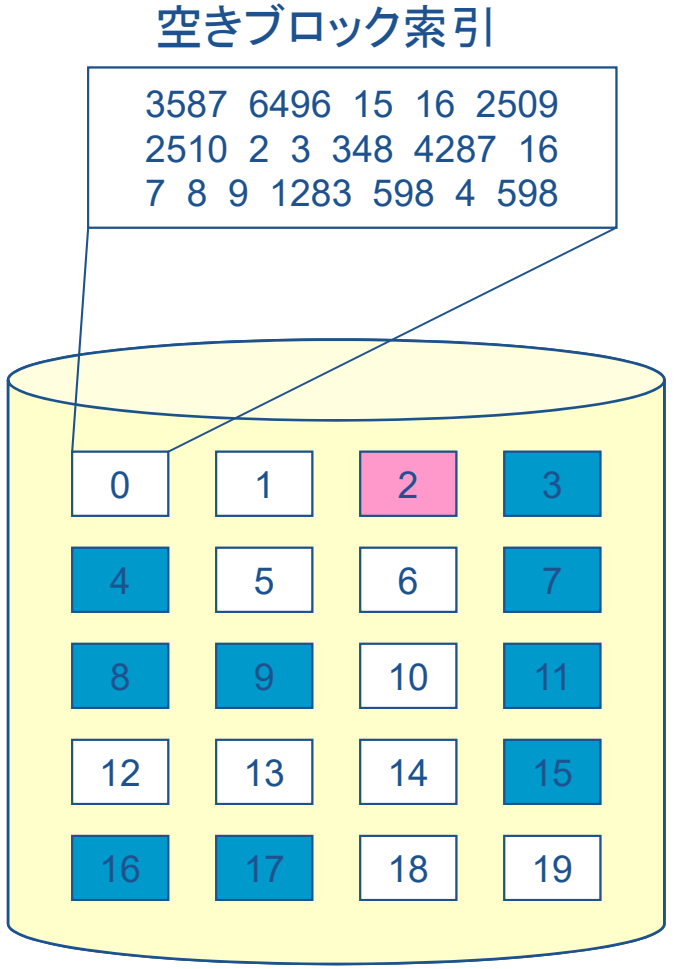
    ⇒ディスクアクセスの増大



なぜ欠点か？

空き領域索引

- 空きブロックの番号を複数ブロックに格納し、ブロックのリストで管理する
- 利点
  - 空き領域管理用メモリが限られているときに有効
- 欠点
  - 隣接した空きブロックの連結は困難（解放されたブロックの番号は最後に追加されるため）
  - 記憶領域のオーバーヘッドがビットマップより大きい（ディスクがほとんど満杯でない限り）



3.3 ファイル管理ーファイルシステムー
3.3.6 ファイル保護

[1] OSによるファイル保護

- 物理的（ハードウェア）障害や、ソフトウェアによる不正操作や不正アクセスからファイルを保護するOS機能
- メモリ保護と、情報への不正アクセスを防御するという基本的な点では同じ
- 異なる点
  - 保護対象が、メモリ保護ではメインメモリで保持しているプロセスであるのに対して、ファイル保護では、ファイル装置に格納してあるファイル
  - メインメモリがプロセス（情報）を一時的に保持するメモリ装置であるのに対して、ファイル装置はファイル（情報）を半永久に格納しておくメモリ装置
  - メモリ保護ではアクセス権の管理だけで十分であるが、ファイル保護では、アクセス権の管理の外にバックアップ機能も必須となる

3.3 ファイル管理ーファイルシステムー
3.3.6 ファイル保護

[1] OSによるファイル保護（続き）

(A) アクセス制御

- ユーザによるファイルアクセスごとに、ファイル情報をもとにして種々のアクセス権限をチェックする
- アクセス権による保護では、例えば、
  - (a) 共用ファイルについては、OSによる「書き込み」を許すが、ユーザには「読み出し」だけを許可する
  - (b) 非共用ファイルについては、所有者（ユーザ）自身のアクセスは「可（自由）」と、他ユーザのアクセスは「不可」とする

(B) バックアップ

- 主要なファイル装置上にあるファイルを、定期的に別の補助ファイル装置（たとえば、超大容量ハードディスクドライブ装置やDVDなど）へ退避する
- 逆操作はリカバリ
  - リカバリ操作は「ファイルのファイル装置への再割り付け」機能も併せもつ

3.3 ファイル管理ーファイルシステムー
3.3.6 ファイル保護

[2] OSによるファイルアクセス制御（例）

(a) アクセス制御リスト
- 各ファイルごとに許可する操作（アクセス権）を単純なリストにして保持
- 長所
  - 仕組みが単純なので、実現が簡単である
- 短所
  - リストは、疎でありかつサイズは大きいので、冗長である

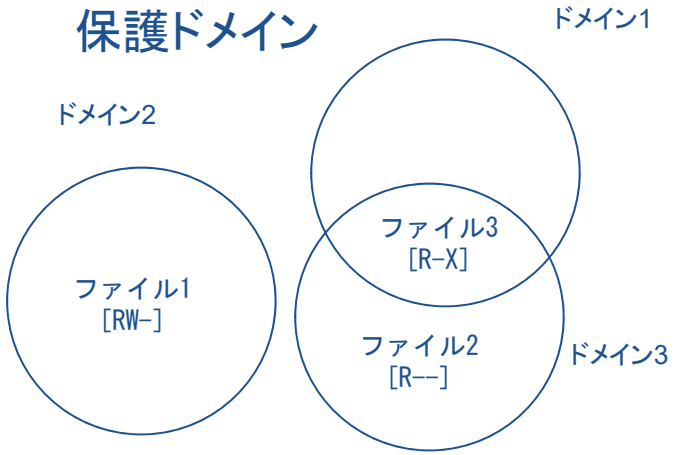
(b) ユーザクラス
- ユーザをクラスに分け、各クラス単位で許可する操作をあらかじめ決めておく
- 長所
  - コンパクトなサイズかつ簡単な仕組み
  - 複数ユーザによるアクセス権の共有などの多彩なアクセス制御が可能となる
- 短所
  - ユーザクラスの設定次第で仕組みが複雑になり、オーバヘッドが顕在化してくる

- アクセス権の対象操作の例
  (1) 読み出し
  (2) 書き込み
  (3) 更新
  (4) 実行
  (5) 生成や削除
- ユーザクラスの例
  (1) 所有者
  (2) グループ
  (3) 使用者
  (4) 共有

ファイルの保護：UNIXの例

- コンピュータ上の資源
  - 名前と可能な操作が決められている
  - 一般プロセスとカーネルが同様にシステム管理情報にアクセス（読み取り、消去、書き換え）するのはよくない
  - ユーザが他のユーザのメールを読むのはよくない
  - プロセスごとに可能な操作を制限したい
- 「保護ドメイン」という概念
  - 保護ドメインは[オブジェクト、権限]の組の集合
  - オブジェクトがファイルの場合
    - 権限は[読み出し、書き込み、実行]の3種類
- 「保護マトリックス」という概念
  - 保護ドメインとオブジェクトに対する権限を表にまとめたもの
  - 列方向のリスト⇒アクセス制御リスト
  - 行方向のリスト⇒ケーパビリティ

保護ドメイン



保護マトリックス

	ファイル1	ファイル2	ファイル3
ドメイン1			Read Execute
ドメイン2	Read Write		
ドメイン3		Read	Read Execute

UNIXにおける保護ドメインと保護マトリックス

- ドメイン：ユーザ識別子(UID)とグループ識別子(GID)、それ以外
- グループはシステム管理者が定義する
- 各オブジェクトがアクセス制御リスト：ACL (Access Control List)を持つ：列方向
- 権限は3種類 (r: 読み出し可能、w: 書き込み可能、x: 実行可能)
- 全部で9ビットでファイルの権限が表される
- 最近は、プロセスは複数のGIDを持ち、可変長のリストであらわせるようになっている

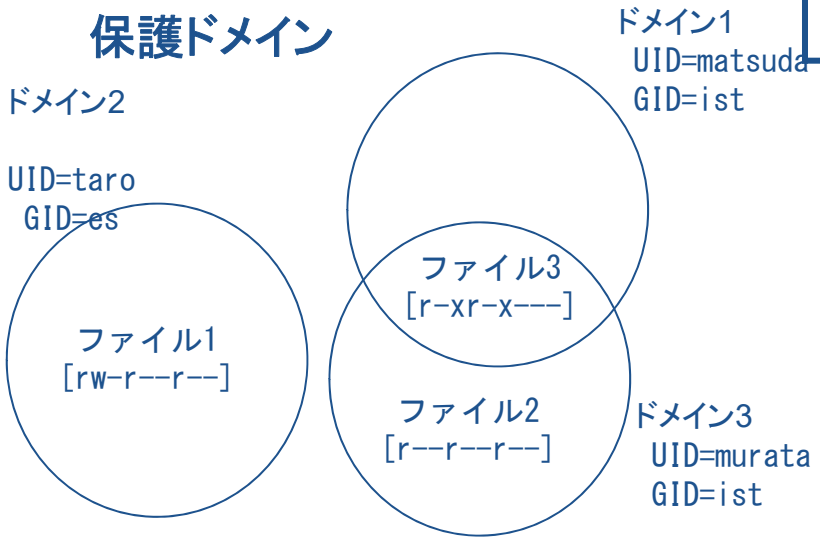
UNIXにおける保護ドメインと保護マトリックス

```
/path/file1   taro      es    -rw-r--r--
/path/file2   murata    ist   -r--------
/path/file3   matsuda   ist   -r-xr-x---
```

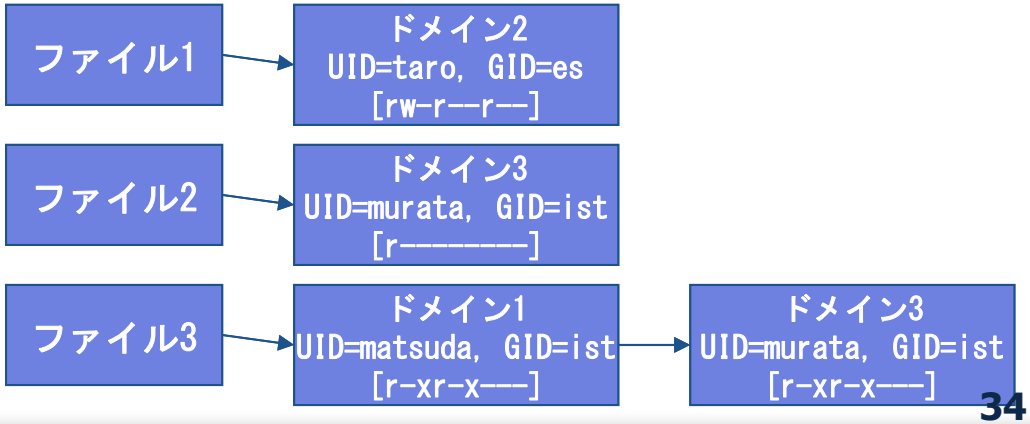
保護マトリックス

	ファイル1	ファイル2	ファイル3
ドメイン1			[r-xr-x---]
ドメイン2	[rw-r--r--]		
ドメイン3		[r--------]	[r-xr-x---]

保護ドメイン



アクセス制御リスト

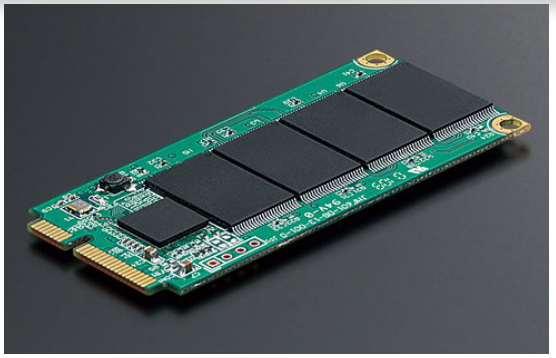


参考3.15(フラッシュメモリ)

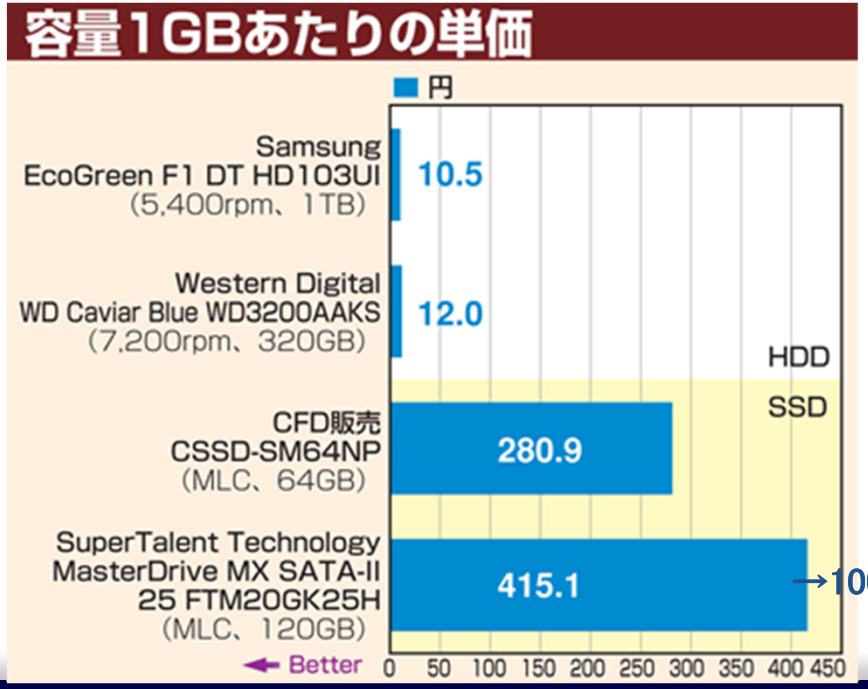
- 一括消去(フラッシュ)および随時読み出し書き込みができるICメモリ
- DRAMとの比較
  - DRAMと同様に任意のアドレスに随時読み出し書き込みができるランダムアクセス性をもつ
  - 書き込みは、ブロック単位で一括消去(フラッシュ)してから行わねばならないので、読み出しやRAMの動作速度に比べると遅い
  - DRAMと比較して、集積度や動作速度では劣るが、不揮発性、すなわち、電源を切ってもメモリ内容を保持できる
- ハードディスクドライブ装置との比較
  - 集積度では劣るが、動作速度と軽さ(可搬性も含む)で優れている
- フラッシュメモリは、メモリ階層では、DRAM(メインメモリ)とハードディスクドライブ装置(主要なファイル装置)の中間に位置する
- ただし、大容量のフラッシュメモリは、ハードディスクドライブ装置を代替して、主要なファイル装置となっている(SSD)

SSD (Solid State Drive)

- 不揮発性の半導体記憶素子であるフラッシュメモリを用いる
- メリット
  - ハードディスクのようにモータやヘッドなどの可動部品がない
    - 衝撃に強い
    - 消費電力が少ない
    - 静穏
    - アクセスタイムが短い
    - サイズがコンパクト
- デメリット
  - 記憶容量が小さい
  - 容量単価が高い
  - 書き換え回数の制限がある
    - ハードディスクに比べると寿命が短く、書き込み回数が増えると誤動作する；書き換え回数の限界10万回
    - ただし、日常的な使用であれば1日に10時間使っても10年から50年は持つ



バッファロー製SSD「SHD-ES9M64G」



3.3 ファイル管理ーファイルシステムー
3.3.7 ファイル装置の管理ーボリュームとの対応付けー

[4] 【まとめ]ファイル装置の管理

- 仮想ファイル装置の実現においては
  - ハードウェア装置の仮想化
    - 異種複数台のファイル装置の隠ぺい、および、それらの管理方式の統一
  - OS(ファイルシステム)の単一化
    - 異種複数台のファイル装置に対する、ディレクトリの単一化や併合、すなわち、単一ディレクトリ管理

  という2つのOS機能による支援が必須
- 特に、OSの単一化においては、形式的な、すなわち、幾何学的形状や構造上でのディレクトリの単一化や併合だけではなく、そのディレクトリを用いて行うディレクトリ管理機能の単一化という「意味的なすなわち論理的な単一化」が必要である
  - 単一ディレクトリ管理によって「ファイル単位の仮想化」および「ファイル単位での管理」が実現できる.
- 現代の代表的なファイルシステムのほとんどが木構造ディレクトリを採用している
  - 木構造は「部分木の集まり」であるので、ある部分木の任意のノードを別の部分木のルートとすることによって、簡単に部分木どうしを併合できる.
  - 複数の木構造ディレクトリを併合して、単一の木構造ディレクトリを構成することによって、ディレクトリの管理が比較的簡単になる
  - →現代のOSのほとんどが、仮想ファイル装置を含めて、木構造ディレクトリを採用している理由