ファイルシステムについてです。教科書は、第6章のファイルの管理です。

オペレーティングシステム
（2024年 第８回）

ファイルシステムについて

ページングについての例題

ページングによる仮想記憶において、ページ置換えアルゴリズムにLRU法を採用する。実メモリの容量を3ページとし、プログラムが仮想ページ番号で

0→2→3→1→0→2→4→0→2→3→1→4

の順にアクセスされるとするとき、以下を示せ
（初期状態では実メモリにどのページも存在しないものとする）

① 最初の仮想ページ4のアクセス時点でページアウトされる仮想ページ番号

② ページフォールトの回数

③ 最終時点でのページテーブルの内容（実ページ番号は適宜想定してよい）

- 仮想ページのアクセス系列に対して、実ページがどのように置き換わるかを以下のように図示する

アクセス系列		0	2	3	1	0	2	4	0	2	3	1	4
実ページ番号	x	0	0	0	1	1	1	4	4	4	3	3	3
	y		2	2	2	0	0	0	0	0	0	1	1
	z			3	3	3	2	2	2	2	2	2	4

- 上図より、最初の４のアクセスでページアウトされるのは、ページ１
- □の枠でページフォールトが発生している（また、ページが参照される）
  初期状態では実メモリにどのページも存在しないものとするので、
  最初からページフォールトが発生する（10回発生）
  □の枠ではそのページが参照されるので、LRU法では最近に参照されたことになる
- ページテーブルの形は仮想ページ番号をインデクスとする配列で、
  要素が実ページ番号（0～4までを示す）

	管理情報	実ページ番号
0	非存在	
1	存在	y
2	非存在	
3	存在	x
4	存在	z

解答としては管理情報の部分は無くてもよい

FIFOの場合

- 問題とは直接関連しないが、LRUでなくFIFOでページ置き換えを行う場合は以下のようになり、

アクセス系列		0	2	3	1	0	2	4	0	2	3	1	4
実ページ番号	x	0	0	0	1	1	1	4	4	4	4	4	4
	y		2	2	2	0	0	0	0	0	3	3	3
	z			3	3	3	2	2	2	2	2	1	1

ページフォールトは9回になる

□の枠で存在するページへのアクセスがあっても、置き換え対象の順位は変わらないことに注意する

- ここで、実メモリの容量が4ページであるとすると、以下のようになり、

アクセス系列		0	2	3	1	0	2	4	0	2	3	1	4
実ページ番号	x	0	0	0	0	0	0	4	4	4	4	1	1
	y		2	2	2	2	2	2	0	0	0	0	4
	z			3	3	3	3	3	3	2	2	2	2
	ω				1	1	1	1	1	1	3	3	3

ページフォールトは10回となって増加する

これ（実ページ数が増えてもページフォールトが増加すること）が、第7回の補足資料にある「Beladyの異常」である。LRUではこの現象は発生しない

ファイルとファイルシステム（ファイル管理）

データのやりとりを行うための統一的な仕組み

- ファイル
  - ハードディスクやCD-ROMなどの（二次）記憶装置に記録されたデータのまとまり
    - 外部装置（二次記憶装置など）や他のプログラムへの動的なデータの入出力のためのもの
      - 格納して、後で読み出す
    - プログラムやデータの転送、保存、交換
      - 大量の情報を格納できる
      - 使用したプロセスが終了しても消失しない
      - 複数のプロセスが同時にアクセスできる
- ファイルに格納される情報
  - ソースプログラム、オブジェクトプログラム、データ、文書、…
  - 文字、図形、画像、音声、…

ファイルシステムによってデータのやり取りを統一的に行います。

ファイルは二次記憶装置に記録されたデータのまとまりであり、それを入れている容器と見ることもできます。そこに格納して後で読み出したり、また格納したりします。入れられる中身は数値の列であり、それがプログラムやデータであったりし、解釈によって文字、図形、画像といったものになります。
ファイルでは、それらを大量に格納でき、メモリとは違ってプロセスが終了しても消えることなく、他の人からも利用できることが特徴です。

- ファイルシステム … OSがファイルを扱う部分
  - 二次記憶装置に記録されているデータを管理する方式
    （管理を行うソフトウェアや記憶媒体に設けられた管理領域や管理情報のこともファイルシステムと呼ぶことがある）
    - 記憶装置にファイルやフォルダ（ディレクトリ）を作成したり、移動や削除を行う方法やデータを記録する方式、管理領域の場所や利用方法などを定める
  - プログラマにファイルを論理的なデータの集合として提供する
    （ファイルが格納されている物理的な媒体を意識させないように）
- ファイルシステムの目的
  - 二次記憶装置からアプリケーションプログラムを装置独立にする
  - 二次記憶装置上の記憶領域の割付けをアプリケーションプログラムで行わなくても済むようにする
  - 記憶機能を効率よく利用できるようにする
  - 入出力速度の性能を有効活用する
  - 若干の破損や故障などに対して可能な限りデータの内容を保障する

OSが登場する前は、入出力装置を制御する処理をアプリケーション側で記述していましたが、装置が変わるとアプリケーションプログラムを変更しなければならないとか、アクセス方法が装置ごとにばらばらでプログラミングが煩雑でした。また、ディスク装置のどこをどう使ってどのような形式でデータを格納するかといったことはアプリケーションプログラムしかわからず、複数のアプリケーションプログラムでデータを再利用することが難しかったりします。

このような背景から、OSでファイルシステムを定めて二次記憶装置などを管理し、アプリケーションプログラムで利用できるようにしています。

ファイルシステムの目的は以下のとおりです。

- ファイルシステムによって …
  - 電源を切ってもプログラムやデータが残っており、利用できる（大容量、永続性）
  - 装置独立な形式のAPI により、アプリケーションプログラムでの二次記憶装置の利用が容易になる
  - 共通の形式のAPI により、アプリケーションプログラム間で共通のデータ利用の基盤を確立できる
  - データを保護できる
  - 他のコンピュータとデータを交換できる
- 広義のファイル

  現代のOSではファイルの意味を広くとらえていて、「ファイルはコンピュータに対する入出力のすべてである」と考えている
  - 情報の入出力のすべてをファイルと見た方が、統一して操作可能となる

これが実現できると、以下のような利点を得ることができるようになります。

多くのファイルシステムでは、装置の特性に依存しないファイルを提供し、その記憶域を管理する、名前をつけてファイルを指定できるようにする、APIを提供してアプリケーションプログラムから統一的にデータを参照できるようにする、可能な限りデータの内容を保証する、という機能を有しています。

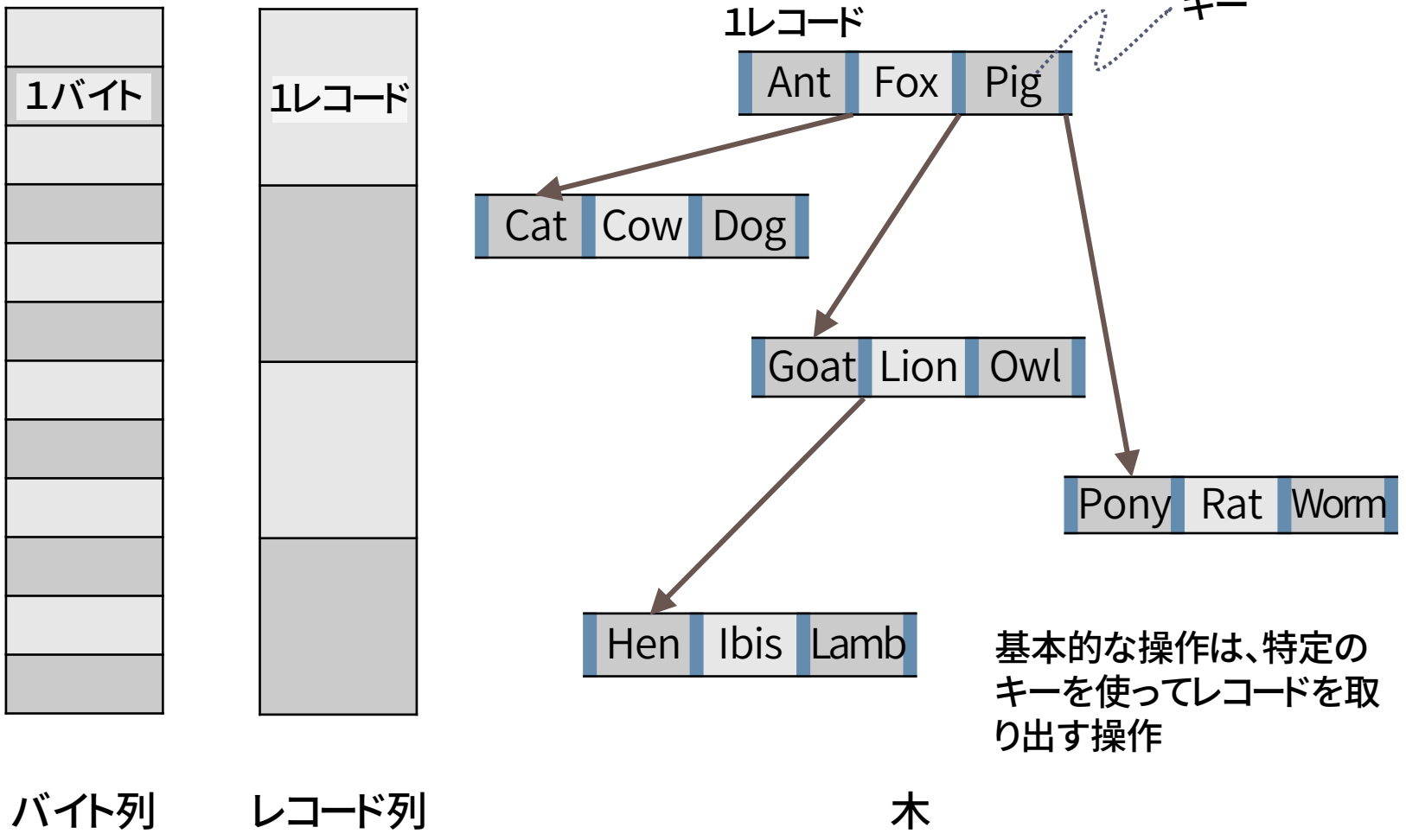
なお、現代のOSでは、ファイルの意味を広くとらえていて、「ファイルはコンピュータに対する入出力のすべてである」と考えます。

ファイル構成

- ファイルの構造
  各アプリケーションが作成・使用するデータであるためアプリケーションが認識すればよいが、オペレーティングシステムがいくつかの固定した形式のファイルを支援する
  - 特定の構造をもたない ー 単なるバイトの列として扱う ー もの
    UNIX（Linux）や Windows（MS-DOS）
    - ファイルは単なるバイトの列であり、特定の区切りはない
    - プログラムからの入出力はバイト単位に行われる
      （物理的には一定の大きさのブロック単位で記憶される）
  - 構造を持つもの
    - レコード（1件ごとの情報の単位）とブロック（入出力の単位）による構成のもの
    - 木構造

ファイルをアプリケーションプログラムにどのように見せるかについては、OSによって異なります。特に、ファイルの構造やファイルのアクセス方法、アクセスする単位で特徴がみられます。

かつての汎用大型計算機のOSではアプリケーションプログラムの目的に応じて複数のアクセス方法やファイル構造などを提供していましたが、UNIXやWindowsなど近年の多くのOSでは単純でシンプルなものを採用しています。
これは、構造を持たず、単なるバイト列として扱うということです。



「モダンオペレーティングシステム」より

これらを図で比較すると、左図のようになり、木構造をもつものは二次記憶内の位置情報をキーとして保持してメモリで木構造を作るのと同じように構造を作ることになります。

木構造では探索や挿入などの操作がうまくできるという特徴があります。

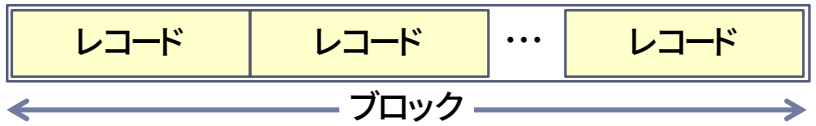
レコードとブロック（論理構造と物理構造）

- UNIXではファイルの論理構造はバイト列、物理構造もバイトがブロック
- メインフレームOSではファイルの論理構造はレコードの集まり
  - レコード（論理レコード）
    ユーザ（アプリケーション）が単位として扱うデータの集合
  - ブロック（物理レコード）
    - 二次記憶装置上の記録の単位 … 一度に読み書きする単位
    - レコードの集合

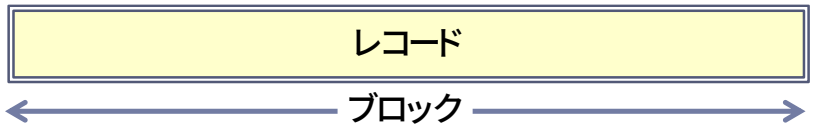
レコードとブロックについては、前回、I/Oシステムでも出てきました。

ブロックは、たとえば100レコードで1ブロックなど、論理的にレコードの集合を表す場合と、HDDなどの二次記憶装置で一度に読み書きされる単位（p.24で「セクタ」という言葉が出てきますが、それなどです）を表すこともあります。

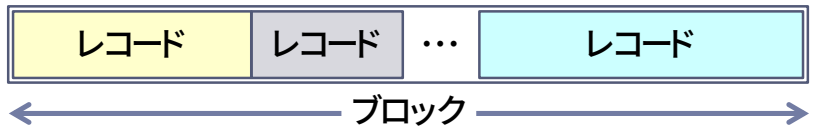
- ブロックレコード … 複数のレコードで構成されたブロック



- 非ブロックレコード … 1つのレコードで構成されたブロック



- 固定長レコード … ブロックサイズはレコード長の整数倍
- 可変長レコード … レコード長はブロック内で可変



ファイルに対するAPI

- システムコールを用いて行う
  - open　：オペレーティングシステムに対してパス名を用いて操作したいファイルを指示
  - read　：プログラム中でファイルから読む
  - write　：プログラム中でファイルへ書く
  - close　：利用し終わったことをオペレーティングシステムに通知する
  - create：新たにファイルを作る
- 標準入出力
  多くのアプリケーションでは1入力・1出力
  実行時に用いるファイルを事前にopenしておき、プログラムでは特に何も指定せず標準的に read、write を行える仕掛けを用意しているOSが多い（たとえば、printf を使う場合、open しなくてよい）

ファイルに対するAPIは、以下のようなものが代表的です。

openでファイル操作を行うと宣言してOSに必要な準備をさせた後、readやwriteによってOS経由でファイルの読み書きを行います。closeについては、プログラムで明示的に行うべきですが、忘れたり異常終了でcloseに到達しないこともあるので、プロセス終了時にopenしたままのファイルがあると、OSが自動的にcloseするようになっています。

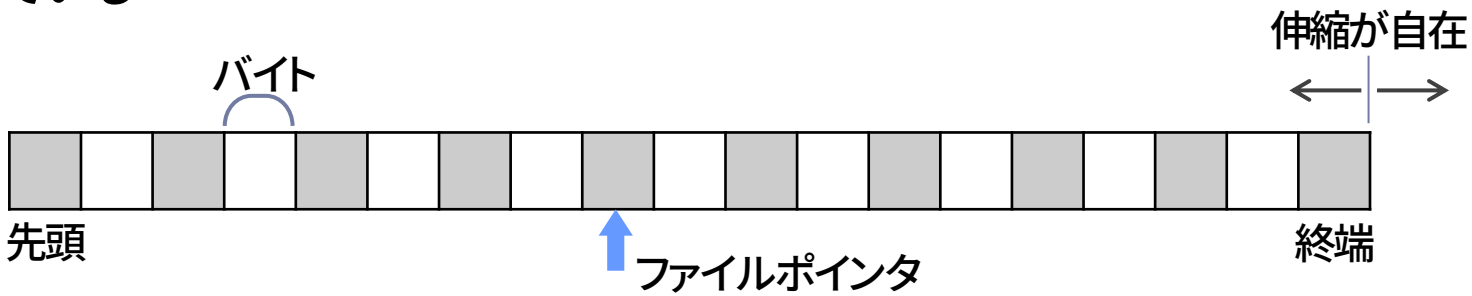
多くのアプリケーションプログラムはファイルから読み出してディスプレイに出力するという1入力・1出力ですので、標準入出力としてユーザがopen、closeしなくてもOSが事前にopenしておき、read、writeを行う仕掛けが用意されています。

ファイルの操作について、詳細を別途提供する「参考書_第10.1.2項.pdf」ー 参考書:「オペレーティングシステムの概念」からの抜粋（ただし、「システム呼出し」を「システムコール」と変更しています）ー で確認してください。

アクセス法

- 逐次アクセス
  - 最後にアクセスした位置から読み出しや書き込みが行われる
  - 最後にアクセスした位置はファイルポインタという内部機構によって保持される
    - 読み出し、書き込み：ファイルポインタが進む
    - 巻き戻し：ファイルポインタをファイルの先頭に移動
- ランダムアクセス（直接アクセス）
  - アクセスするごとにアクセスすべき情報の位置を指定

Linux（UNIX）やMS-DOSなどでは逐次アクセスのみをサポートし、バイト列での読み書きと任意のバイト位置へのファイルポインタの移動（シーク）を許している



アクセス方法には、逐次アクセスと直接アクセスがあります。

逐次アクセスでは、最後にアクセスした位置がファイルポインタで保持されます。読み出しではファイルポインタの指す位置から情報を読み出し、ファイルポインタを自動的にその終わりまで進めます。書込みでは、ファイルの最後に情報を書き足し、ファイルポインタを書き込んだ情報の終わりに進めます。

UNIXやMS-DOSでは逐次アクセスだけをサポートしています。シークがあれば、読み出しや書き込みの前にシークを実行して直接アクセスと同等の機能が実現できます。

ユーザから見たファイルシステム

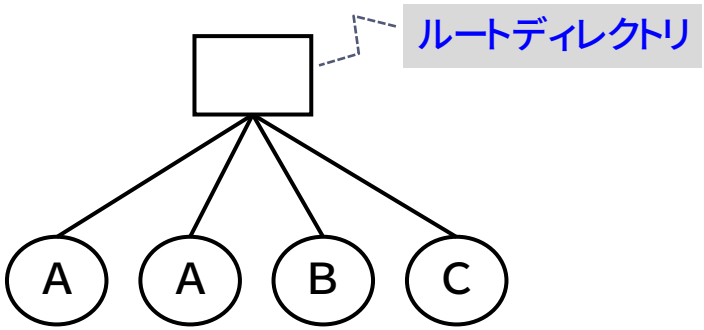
- ディレクトリ
  ファイルを整理・管理するため、複数のファイルをまとめたグループを形成する特別なファイル（Windowsではフォルダと呼ぶ）
  - 1階層ディレクトリシステム
    - すべてのファイルを格納するただ1つのディレクトリ（ルートディレクトリ）からなる
  - 2階層ディレクトリシステム
    - 各ユーザに固有のディレクトリを与える。ユーザ毎に名前が衝突しない
  - 多階層ディレクトリシステム
    - 汎用の階層構造：ディレクトリの中にディレクトリを持てる → ツリー構造
  - UNIX系OSでは最上位を根（ルートディレクトリ）とする多階層の構成
    他のパーティションは設定したマウントポイントを基準点としてツリーを構成
  - Windows、Mac OSではディスクまたはパーティションごとに多階層のツリーを構成
    （この場合、各ツリーの最上位をルートディレクトリと呼ぶことがある）

ユーザから見たファイルシステムは、名前や複数のファイルがどのようにまとめられ、その中の1つをどのようにして選択するかです。

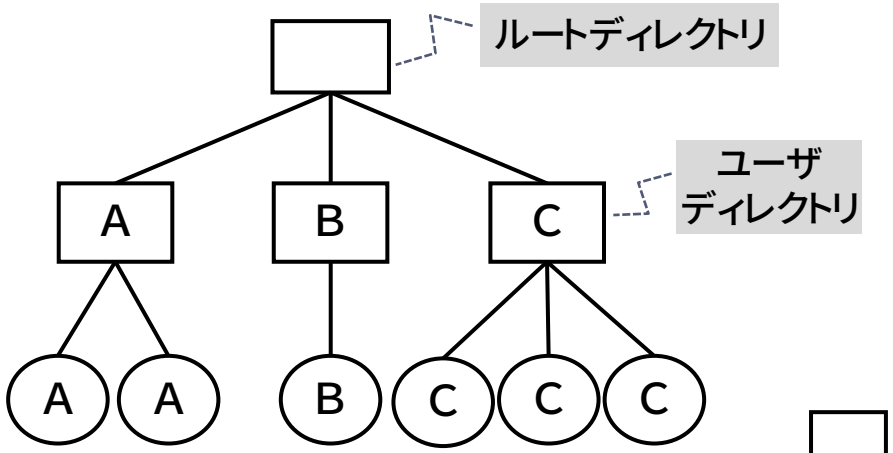
複数のファイルを管理するために、ディレクトリ（フォルダ）という概念があります。これは複数のファイルをまとめたグループを作る特別なファイルです。ディレクトリもファイルなので、ディレクトリの中にさらにディレクトリが入ることができます。
そこで、ディレクトリが1つだけのもの、もう1段階増えて2階層のディレクトリがあるもの、3階層以上任意の階層ディレクトリがあるもの、が考えられます。

UNIXやWindowsなど多くのOSでは多階層のディレクトリシステムであり、UNIXでは、最上位がルートディレクトリとなる1つの木構造を形成します。

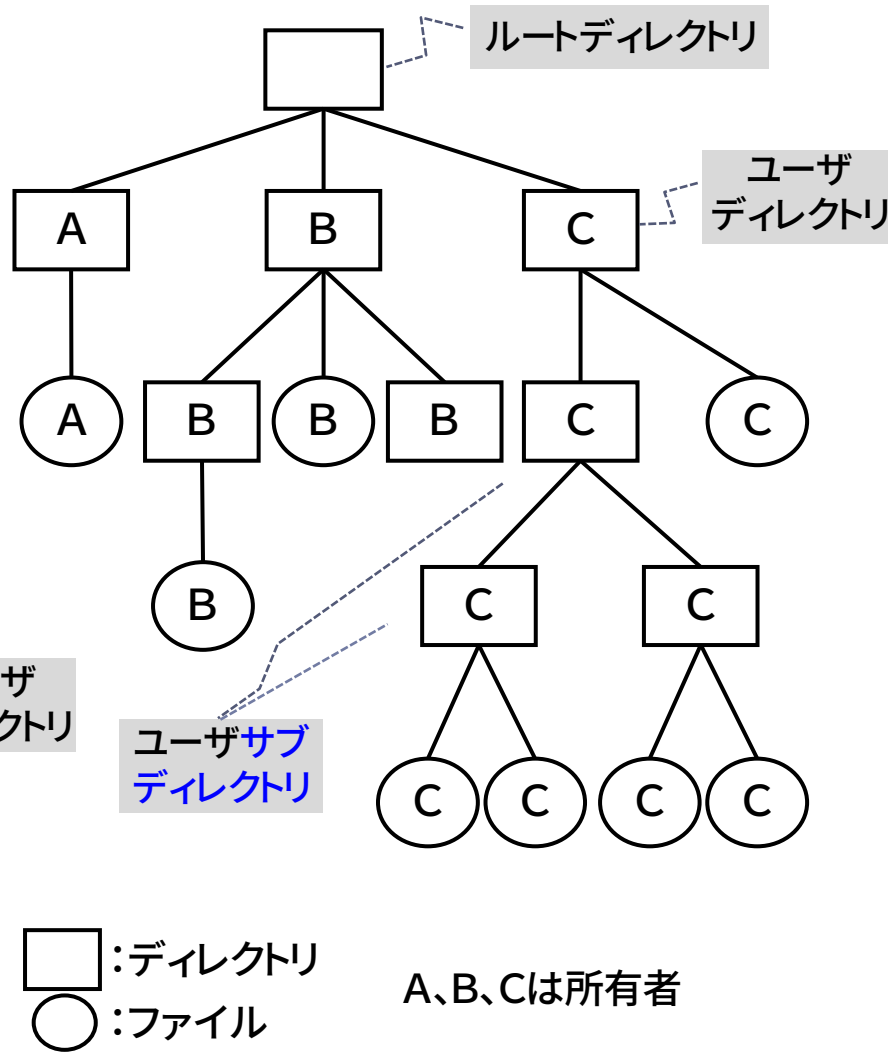
1階層ディレクトリシステム



2階層ディレクトリシステム



多階層ディレクトリシステム



これは、階層のディレクトリ構造を図示したものです。

- カレントディレクトリ（作業ディレクトリ）
  現在開いているディレクトリ、基点となる現在位置
- パス名
  - ファイル名やディレクトリ名はそれが含まれるディレクトリの中で一意であることが必要
    - ディレクトリが異なれば、同名のファイルやディレクトリがあっても構わない

  ファイルやディレクトリを指定するにはそれが含まれるディレクトリも特定することが必要 ・・・ 経路（パス）
- ファイル名やディレクトリ名の指定方法
  - ルートディレクトリからの経路を並べたもの … 絶対パス（フルパス）
  - あるディレクトリからの相対的な経路 … 相対パス
  - ディレクトリ名やファイル名は区切り文字によって区切られる
    - UNIX系OSやMac OSでは「/」、Windowsでは「＼（¥）」が用いられる

ディレクトリシステムで作業しているとき、現在自分がいるディレクトリをカレントディレクトリと呼びます。

ファイルには名前が付けられますが、そのファイルが存在するディレクトリの中では同じ名前があってはなりません。

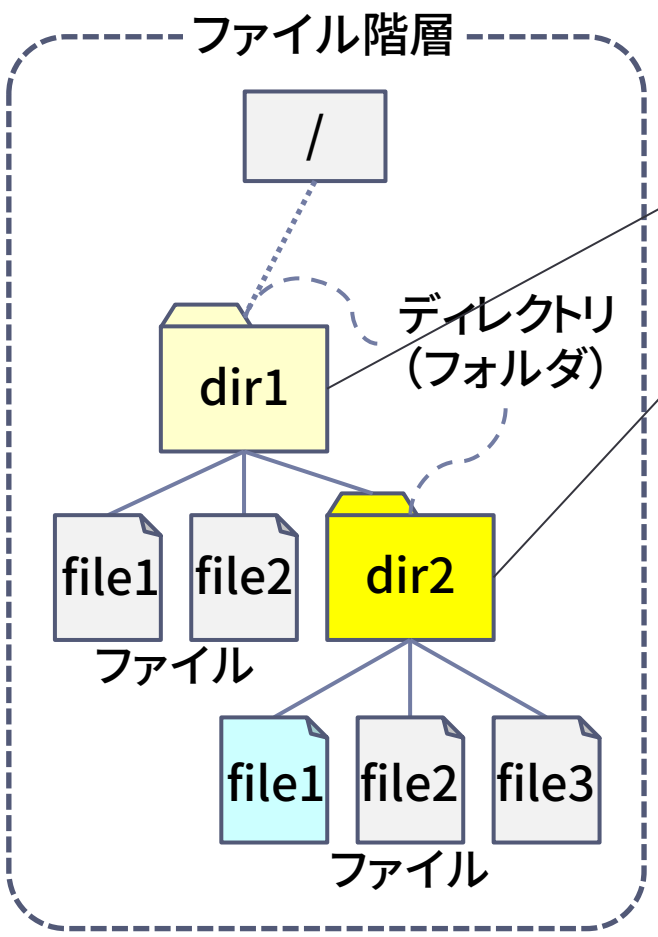
別のディレクトリのものとでは名前が同じでも構わず、一意に指定するにはディレクトリも含めて表現することになります。

ディレクトリ→ディレクトリ→・・・→ファイルと目的のファイルに至る経路（パス）で指定することになり、これをパス名と呼びます。

ルートディレクトリからのパスを順に指定したものを絶対パス、あるディレクトリ（たとえばカレントディレクトリ）からのパスを相対パスと呼びます。

ディレクトリとディレクトリまたはファイルの区切りは、「/」や「¥」の文字が用いられます。
（半角バックスラッシュ（＼）は日本語の環境では¥で表示されます）

ディレクトリとパス



親ディレクトリ「..」

カレントディレクトリ「.」

相対パス指定の場合、カレントディレクトリが起点となる

ここで「file1」は file1 のこと

絶対パス指定の場合、ルートディレクトリが起点となる

file1 は「/･･･/dir1/dir2/file1」

図示するとこのようになります。

図では絶対パスで /･･･/dir1/dir2/file1 と書いていますが、この ･･･ の部分は、ルートから順にたどったもの、たとえば /A/B などのようになります。

相対パスでは、 dir2 にいる場合に「file1」とすると、青色の「file1」で、「../file1」(親:dir1に上がって、そこからfile1とたどる)で灰色のfile1が指定されます。

ファイルの見え方

- ファイルの名前付け
  ファイル名 … 名前で特定する。自由に設定可能
  - 命名規則はオペレーティングシステムにより異なる
  - 文字列で表現、多くのファイルシステムでは255文字まで
    - 文字数、使用不可能文字、大文字小文字の区別など
  - ピリオドで区切られた名前が多用される
    例：「hello.c」、「sum.java」等
    Linux（UNIX系OS）ではファイル名の終端でファイルの種類を表す慣習があり、
    一般にsuffixと呼ぶ。必ずしもピリオドで区切るとは限らない
  - Windowsなどでは、ファイル名の末尾にピリオドで区切ってファイルの種別を表す文字列を記述する（.exe や .bmp など）
    … ファイル拡張子
- ファイルの管理情報
  - 名前、型/種類、大きさ、位置、保護のための情報、アクセスの情報
    …

ディレクトリも含めた個々のファイルの名前のつけかたはおおむね自由な文字列ですが、長さの制限や多少使えない文字があります。
また、英字の大文字と小文字を区別するかどうかがOSによって違ったりします。

名前の文字列からは、名称と種類がわかりますが、それ以外にもファイルの情報は様々なものがあります。

ファイルシステムの例

- Windowsのファイルシステム
  - FAT32　：クラスタと呼ぶ領域を1次元配列で管理
    外付ディスクなど利用、Windows以外でも利用できる
  - NTFS　：Windows NTやXPなどのジャーナリングファイルシステム
  - exFAT　：フラッシュメディア向けのファイルシステム
- UNIX（Linux）のファイルシステム
  - ext3、ext4、XFS
- Mac OSのファイルシステム
  - HFS+、HFSX
    ファイルの内容を格納する領域に加え、ファイルについての情報を格納する領域がある
- CD-ROMなどのファイルシステム
  - ISO 9660ファイルシステム
  - UDF（Universal Disk Format）
    特に書き込み/書き換え可能な光ディスク（Blu-rayやDVD-RAMなど）向け

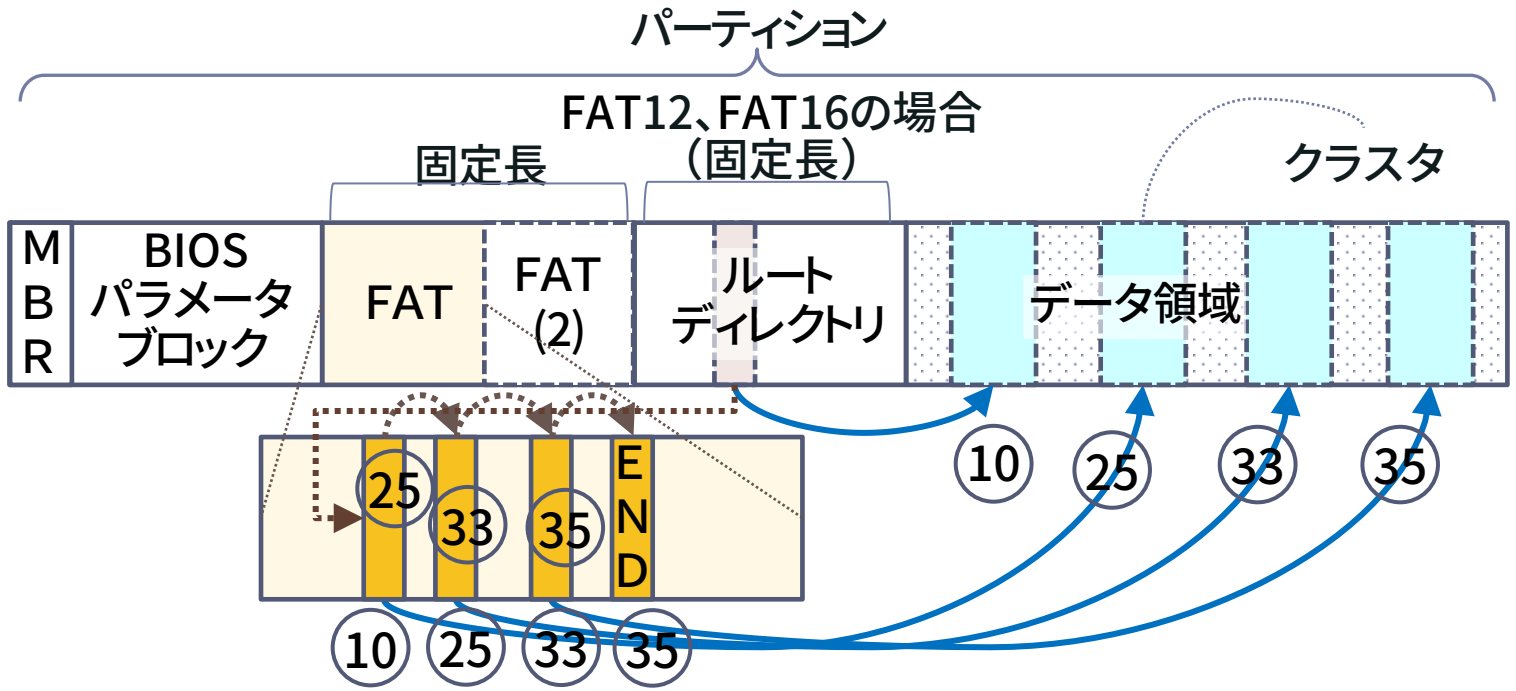
代表的なファイルシステムには以下のようなものがあります。ご存知の名前もあると思います。

古くからあるものですが、簡単なものをそれぞれ説明して行きます。

FAT

ファイルアロケーションテーブル（File Allocation Table）

- MS-DOSのファイルシステムにおけるディスク内のファイルの位置情報などを記録するための領域
- これが転じて、FATを用いるファイルシステムもFATと呼ぶ
  - FAT12、FAT16
  - FAT32 … ルートディレクトリもFATで管理し、データ領域内に置く（大きさ可変）



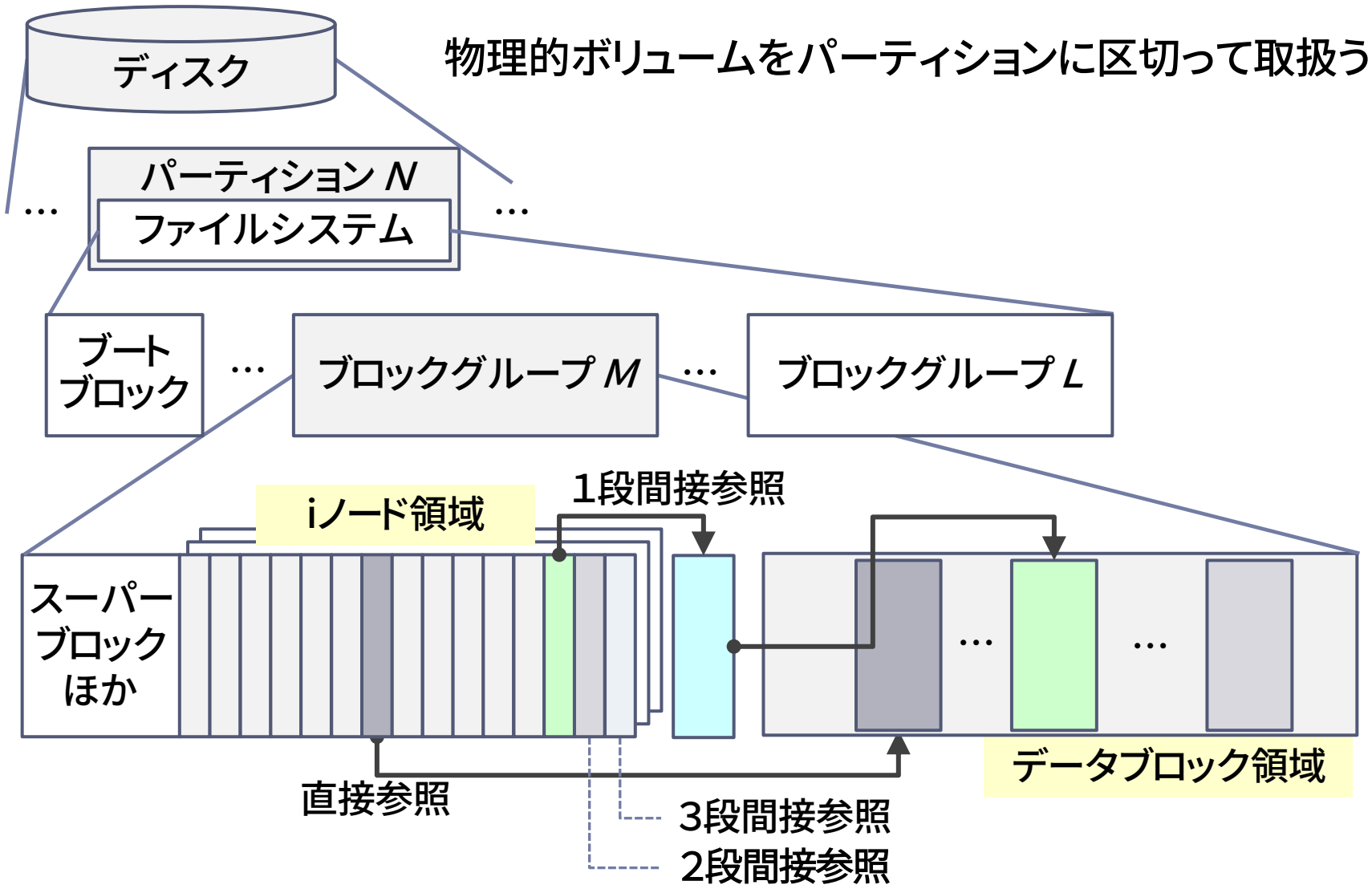
FATはMS-DOSの時代からあるもので、現在でもデータ交換などで用いられることがあります。

ディスクをブロック（クラスタと呼んでいます）単位で扱い、ファイルはクラスタの集合になります。順にクラスタ番号でリストのようにつながれており、そのつなぎを管理しているものがFATです。FATもディスクの中の特別な領域にあります。
FATのエントリが次のクラスタアドレスを指すようになってチェーンして行き、最後は特別な値で終了を示します。

ブロックの集合をこのように管理する方法は、メモリ管理でも出てきました。

（クラスタのサイズは、512B～32KBで、FAT12、16、32の数字は、クラスタ番号を管理するデータのビット数を表します）

Linuxのファイルシステムの構造



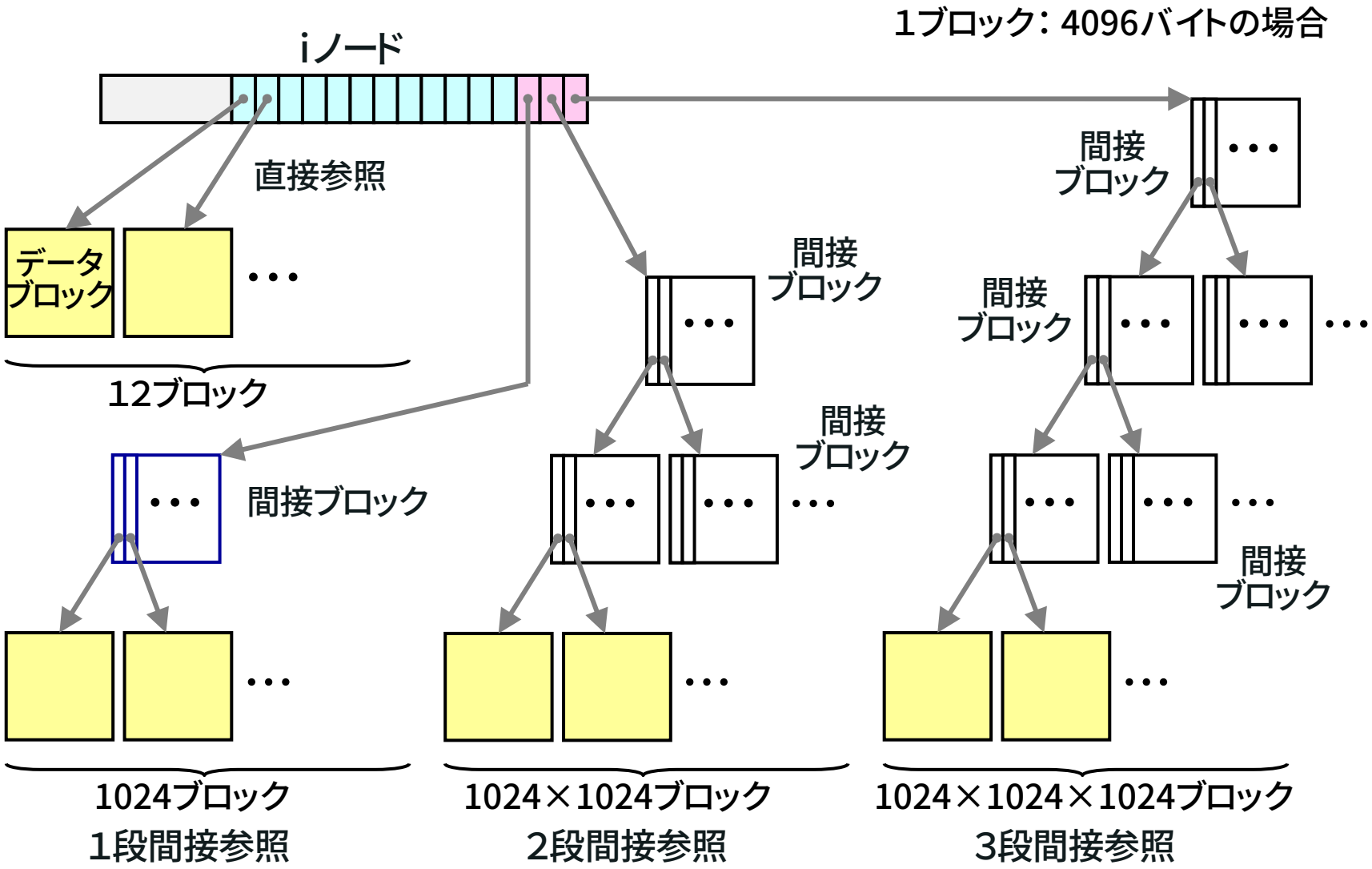
Linuxではそれとは違った構造をとっており、基本的なものはこの図になります。

1つの物理的なディスクはパーティションに分けられ、さらにパーティションがブロックグループに分けられます。

ブロックグループがファイルに対応するもので、1つのファイルが「iノード」と呼ぶ構造で管理されます。
iノードの中の情報で直接にブロックをポイントするものがあり、さらに3段階までの間接参照ができるようになっています。

Linuxファイルの構造

（ext2、ext3）

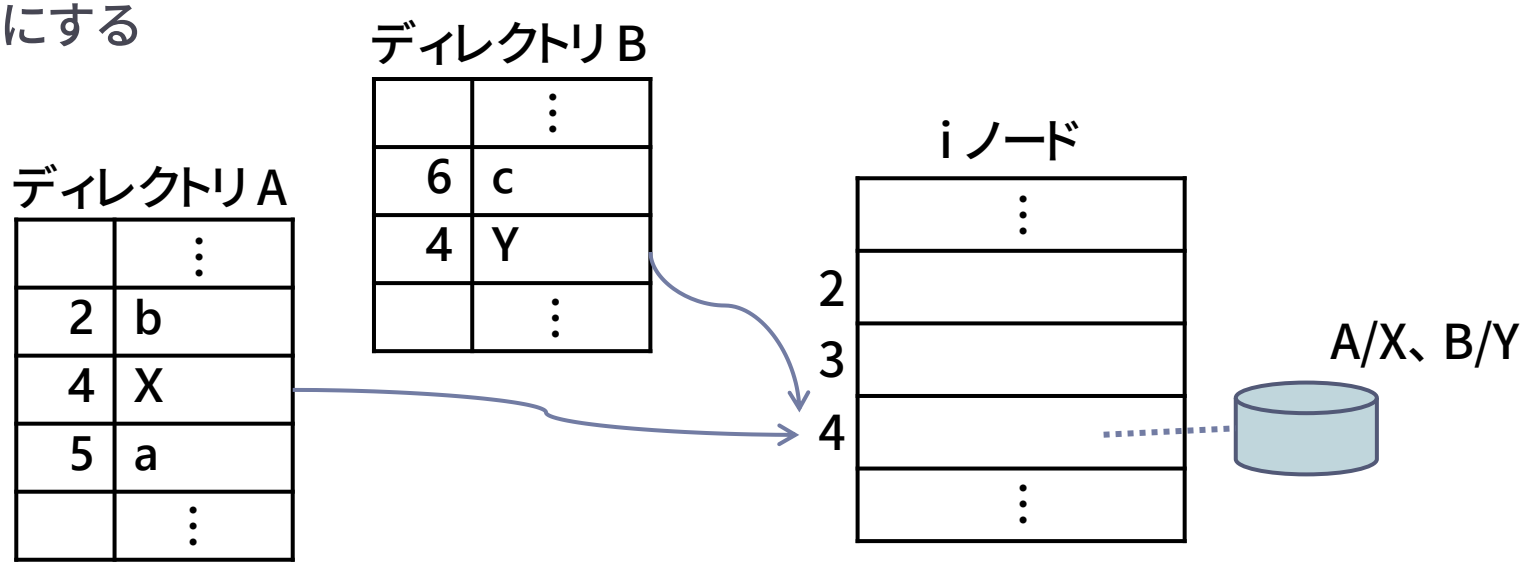


ext2やext3という名前で呼ばれるものの構造です。直接参照できるものは12ブロック分で、それよりサイズの大きいものは間接ブロックを使用します。

ブロックサイズは、4KBが一般的であり、間接ブロックも4KBとなるので、1個の間接ブロックで1024個のブロックが指せます。2段間接では1024個させるものが1024個、3段間接では1024×1024×1024となります。

リンク

- 複数のパス名で同一のファイルやディレクトリにアクセスする仕組み
  - シンボリックリンク（ソフトリンク）
    ファイルのデータとしてパス名を格納しておき、パスを検索する際、シンボリックリンクなら、格納されたパス名を用いて検索するもの
  - ハードリンク
    異なるディレクトリのエントリから同じiノード（のような機構）を指すようにする



ファイルシステムにおいて、リンクとは、別のパス名によって同じファイルにアクセスする仕組みです。シンボリックリンクとハードリンクがあります。

シンボリックリンクは、同じファイルに別名を付けるというものです。Windowsでは「ショートカット」と呼ばれます。

ハードリンクは、iノードのようにファイル名を格納していない管理構造があるもので実現でき、FATのようなファイルシステムでは実現できません。

補足：ジャーナリング

- ファイルシステムに対する書き換え処理のコマンドを逐一記録する機能
  コンピュータが急停止するなどの障害時に有効となる
- 実データの書き込みよりもメタデータ（管理用の情報）の書き込みを先行的に行い、書き込み要求に従ってそのメタデータを「ジャーナル」と呼ばれる領域に逐次記録し、更新する
- メタデータと実データ間の矛盾が発生しても、矛盾が発生する前のメタデータに強制的に戻す事により、実データを含むファイル自体の消失を避ける事ができる
  （ハードディスク全体を詳細にスキャンするより遥かに処理時間を短くする事が可能）

保護に関して、ジャーナリングという機能があります。

ディスクに書き込みを行っているときに障害で急停止した場合など、保持するデータの構造に矛盾を生じて、正しくアクセスできなくなることがあります。このような事態をさけるため、ファイルの管理情報（メタデータ）に対する変更を記録しておこうとするものです。

実際の書き込み作業に先立って、このような管理情報をジャーナルと呼ぶ領域に記録します。

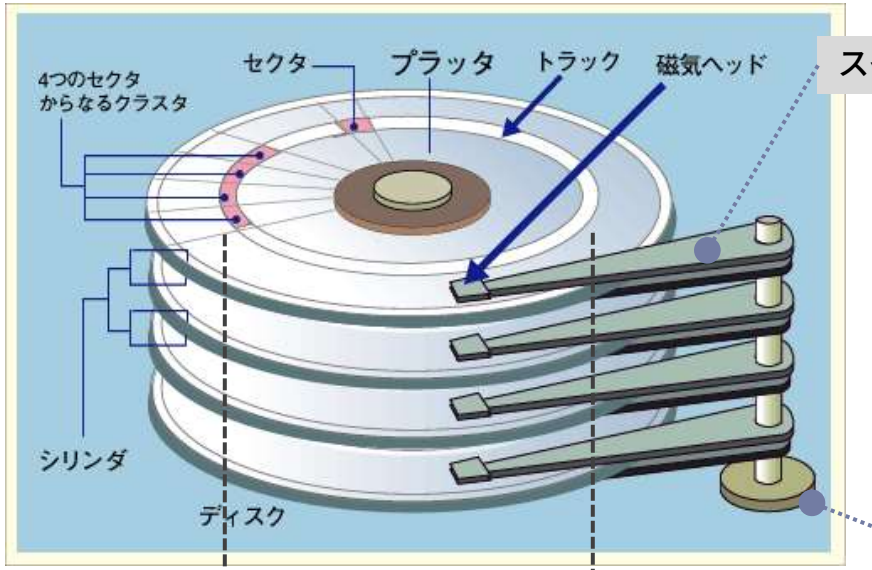
ジャーナルがちゃんとしていれば、実データ書き込みで障害が起こっても矛盾が発生する前の状態に戻して、やり直すことができます。

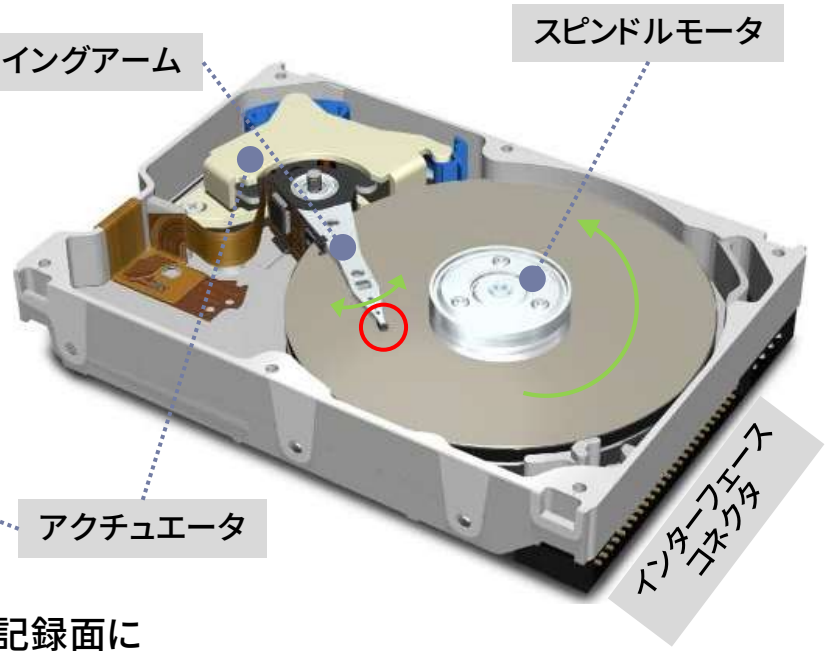
新規ファイル書込みでの例

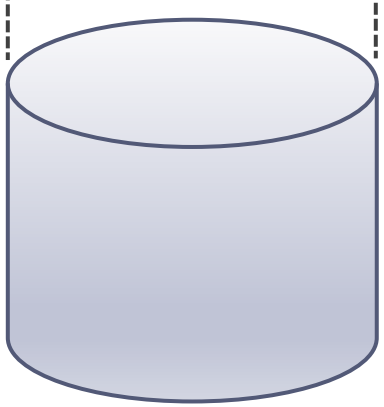
- メタデータ：操作ログ（書き換え処理のコマンド）
  1. 「ジャーナル領域」に操作ログを書く
     - iノード x用のデータ領域を確保（たとえばiノードで管理するものとして）
     - iノード xを確保
     …
  2. 操作ログのI/Oが完了したうえで、操作ログの終わりを示す「コミットログ」を書き、そのI/Oの完了を待つ
- ジャーナルへの記録中にシステムが停止した場合、コミットログが存在しないので、そのジャーナルは破棄される。このとき、ファイルシステムの本体部分はまったく変更されていないので、整合性は保たれる
- コミットログが記録されれば、ファイルシステム本体にログのとおりに変更を加える
- この途中でシステムが停止しても、ログからすべてのI/O操作をやり直せば整合性が保たれる

ジャーナリングの動作例です。

磁気ディスク装置の構造







複数のプラッタ記録面にまたがる同一位置のトラックを、仮想的に積み重ねて構成した円筒形

プラッタが1枚（記録面は2つ）でもシリンダは構成される

シーク：目的のシリンダへの移動
サーチ：目的のセクタへの移動

磁気ディスクのアクセスでは、シークとサーチに要する時間が大部分を占める

同じシリンダ上のトラックなら連続して高速にアクセス可能

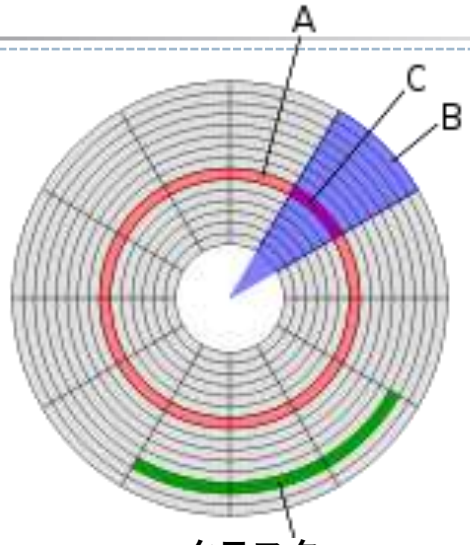
ハードディスクはプラッタと呼ばれる磁性体の円盤が何枚か積み重なった構造で、回転するプラッタの同心円をなす部分に磁気ヘッドが位置づけされ、情報を読み取ります。

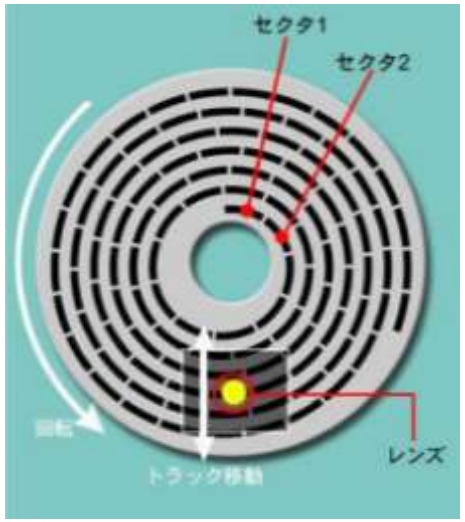
複数枚のプラッタの同じ同心円上にそれぞれヘッドがあって、そこの情報は連続して読み出せるので、シリンダという概念で表します。

シリンダがあり、同心円であるトラック、トラック上の部分セクタという階層構造でとらえられます。

トラックとセクタ

- トラック
  同心円上の部分 … A
  （光ディスクでは内側から外側に向かって1本のトラックが渦巻き状に存在）
- セクタ（ブロック）
  トラックの一部分 … B、C
  トラックをさらに等分した領域
  データを読み書きする単位
  - 磁気ディスクの場合、1セクタ=512バイト
    光ディスクの場合、1セクタ=2048バイトが典型的
- セクタへのアクセス
  最初のセクタから何番目かを示す数値を用いる
  （LBA：Logical Block Address）
  以前は、シリンダ/ヘッド/セクタの3つのパラメータを用いていた（CHS方式）





光ディスクの場合

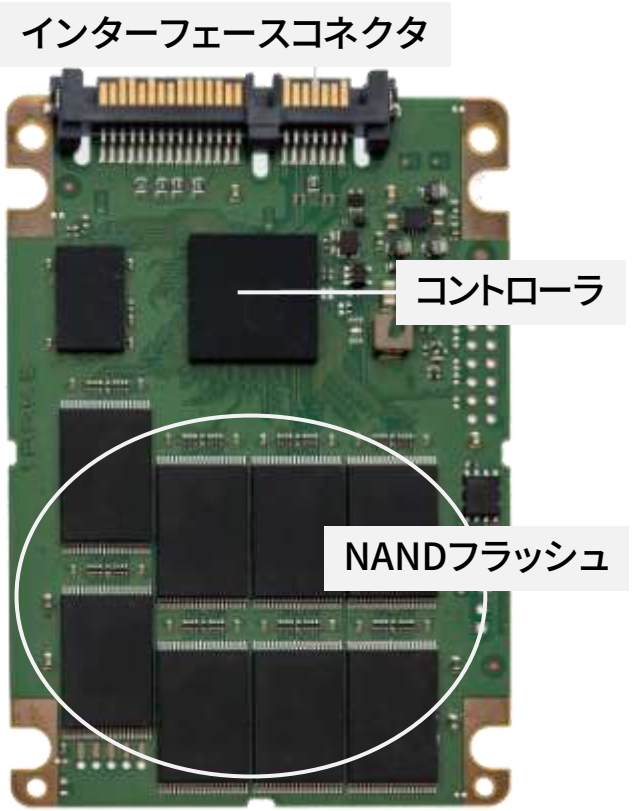
ハードディスクでは、同心円=トラックがいくつもある形式ですが、光ディスクでは1本の渦巻きになっています。

セクタがアクセスの単位になります。最初のセクタから順に何番目であるかの番号が振られています。

SSD（Solid State Drive）

- 記憶媒体にNANDフラッシュメモリを用いる … 電源を切っても情報が保持される
  - トランジスタ（MOSFET）に電荷を蓄積する領域を作り、そこに情報を保持する
  - 読み書きは「ページ」単位（数KB ）、消去は複数ページの「ブロック」単位（数MB）で行う
- HDDとの比較

	SSD	HDD
書き換え耐性	書き込み・消去のたびに素子が劣化	無視できる
年月などによる劣化	考慮が必要	無視できる
容量	○	◎
速度	◎特にランダムアクセスに優れる	○高速
コスト	△容量あたりの単価が比較的高	○
省電力	駆動部分がないため省エネ	消費電力・発熱大



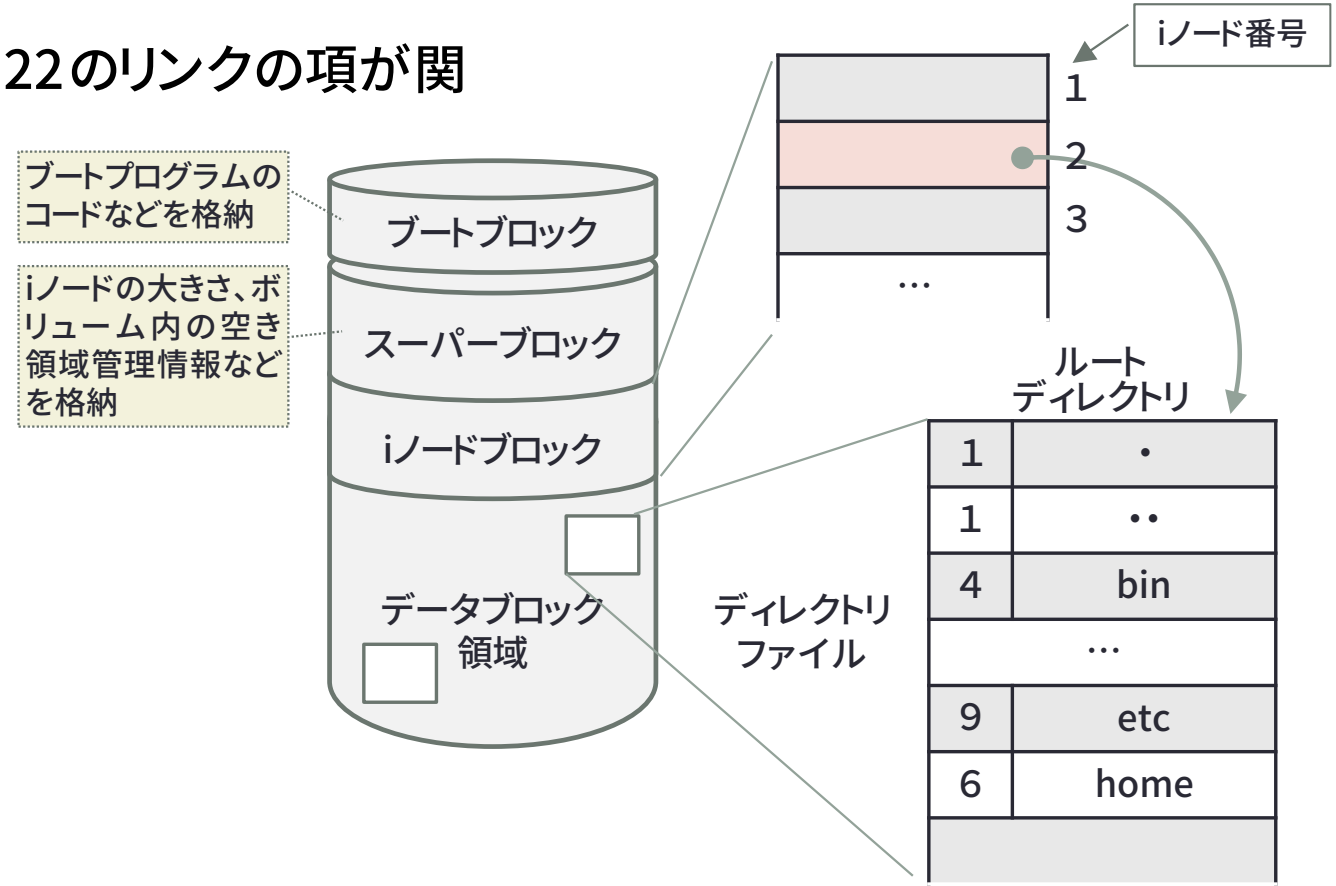
- SSDの弱点
  - 上書きできないため、使い込むと遅くなる
  - 記憶素子に寿命がある

• トリム（TRIM） … 削除済みファイルの残骸のブロックを削除する
• ウェアレベリング … すべてのブロックに均等に書き込むようにする
で対処する

教科書との対応

- メモリマップトファイルについては、第5回の補足 p.4で触れています
- 教科書 図6.8では、間接ブロックは1KB（4Bのエントリが256個）としています
- ディレクトリの構造（教科書 p.82）については p.22のリンクの項が関連します
  UNIXの場合、2番目の i ノードがルートディレクトリを指していて、ディレクトリは i ノード番号とファイル名で構成されます
- ディスク装置については、今回で触れています
- メインフレームOSに関する項目は割愛しました

教科書での説明と記載箇所が違うところがありますので、読むときに注意してください。



第8回の課題

（1）p.19のファイルの構造で、クラスタ番号が 16ビットで表現され、クラスタサイズが32KBの場合、1つのファイルの最大サイズはどうなると考えられるか

（2）p.21のファイルの構造では、1つのファイルの最大サイズはどうなると考えられるか

今回の課題です。クラスウェブのレポートで提出してください。

期限は、6/8の午前中とします。

事後学習・事前学習

- 今回の講義資料に基づいて内容を振り返り、教科書などの該当箇所を読む
- 教科書第8章（8.1）に目を通しておく

今回の講義内容の振り返りと次回の準備をお願いします。