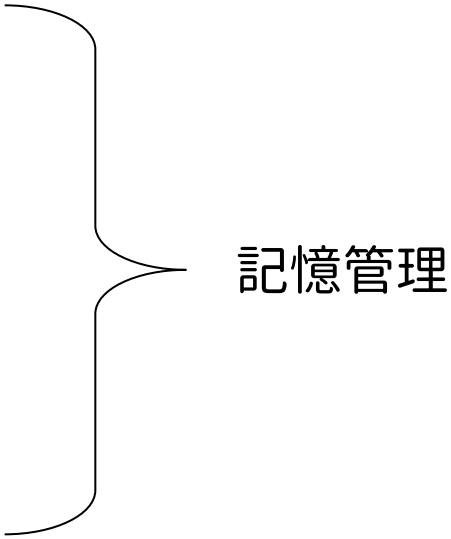
Operating Systems

ファイルシステム

2020-12

OSの代表的機能(復習)

- プロセス管理
  - 実行中のプログラムの管理
  - コンピュータの実行状態の制御
- メモリ管理
  - 主記憶の管理
  - プログラム実行のためのメモリの管理
- ファイル管理
  - 補助記憶(ストレージ)の管理
  - HDDやSSDに保存されたデータの管理



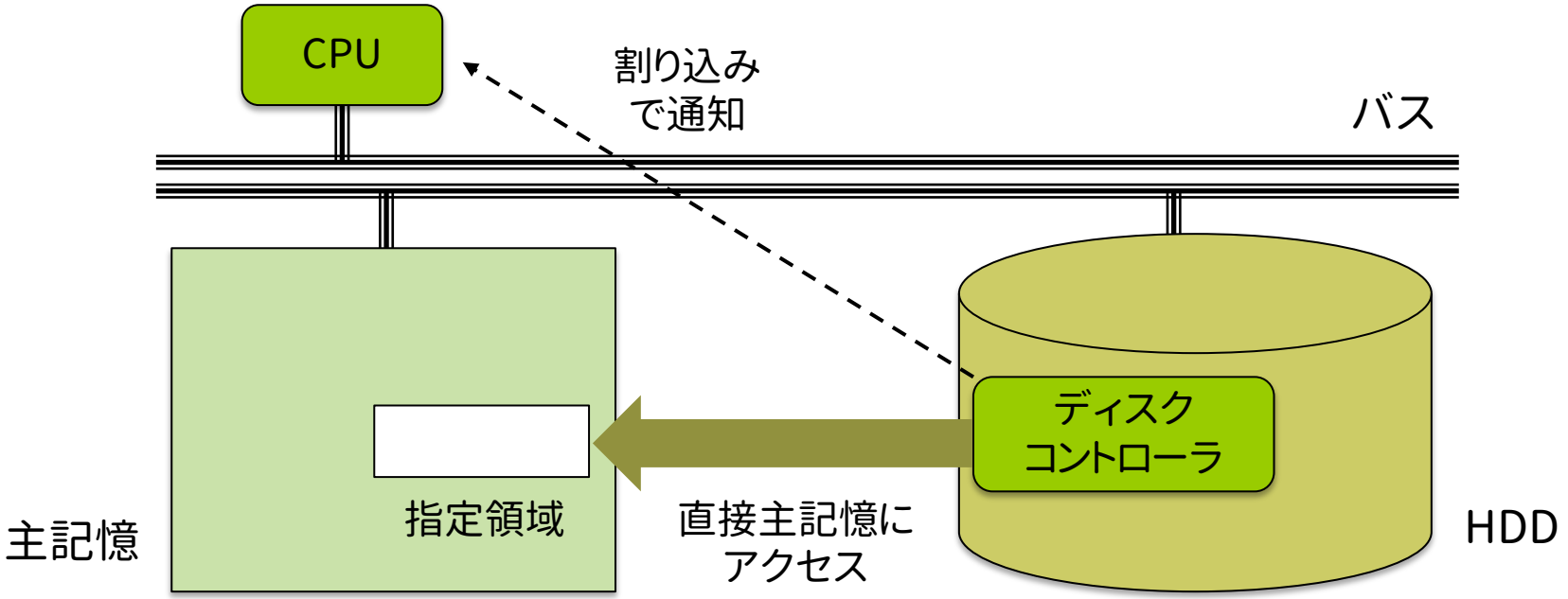
- その他
  - 通信・ネットワーク,セキュリティ,ユーザ・課金管理など…

入出力デバイス制御

- 入出力（I/O = Input / Output）
  - 主記憶（メモリ）とデバイス（外部装置）と間でのデータのやりとり
- CPUによる入出力の方法
  - CPUがレジスタにデータを読み込み，レジスタからデータを書き出す
    主記憶（メモリ） ⇔ CPU（レジスタ） ⇔ デバイス
  - ポートIO：専用のポート命令でデバイス番号を指定して入出力する
  - メモリマップトIO：メモリの特定アドレスをデバイスへの入出力に使う
- CPUによる入出力の問題点
  - 入出力（データ転送処理）の最中に，CPUは他の処理ができない
  - 高速大容量のデータ転送では，CPUの負荷が大きくなる
  - ⇒ システム全体の性能低下の原因になる

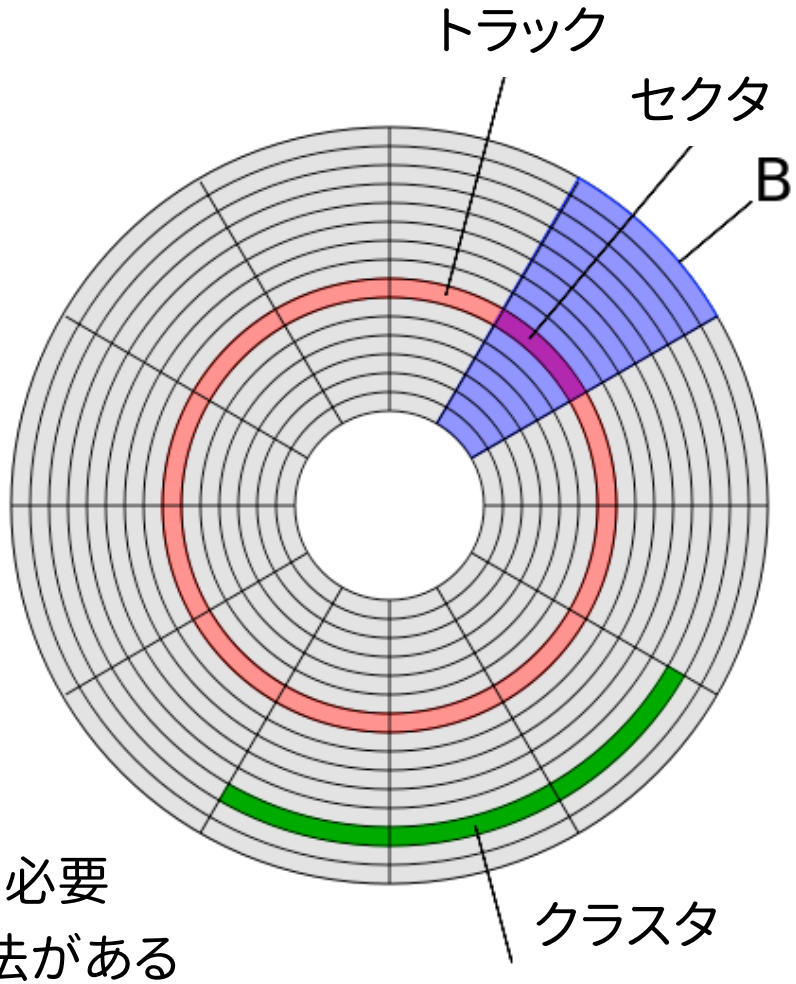
DMA（直接メモリアクセス）

- DMA（direct memory access）
  - デバイス側に搭載されたプロセッサ（デバイスコントローラ）が，直接本体の主記憶にアクセスしてデータを読み書きできるしくみ
  - 転送の完了は，“割り込み”で本体CPUに通知される
  - この制御のために，デバイスドライバは物理アドレスを操作できる



ディスク装置の構造

- 磁気ディスクの構造
  - トラック：円周（同心円）の1周分
  - セクタ：トラック内の小さい扇型（ディスクの最小記憶単位）
  - クラスタ：一定数の連続セクタ（OSが管理する最小単位）
- ファイルシステムの実現
  - ファイルの実体＝クラスタの集合
  - ファイルはディスク上で隣接して保存されているとは限らない
  - クラスタを順々につなげるしくみが必要
  - クラスタの割り当てには様々な方法がある



図はWikipediaから引用

パーティションとボリューム

- パーティションとは
  - 大容量記憶装置の領域を区切り，1個～数個の区画に分けたもの
  - 各区画は別々に「フォーマット」されてファイルシステムが構築される
  - パーティションごとに異なるOSをインストールすることも可能
- ボリューム
  - フォーマットされた区画は，OSから「ボリューム」として認識される
  - Windowsでは，「C:」等のドライブレターで指定してアクセスできる
  - UNIX等では複数のボリュームがある場合，「マウント」して結合する
- パーティションとボリュームの確認
  - Windows：田アイコンを右クリック → [ディスクの管理]
  - macOS：[ユーティリティ] → [ディスクユーティリティ]

ファイルシステム

- ファイルとは何か
  - 名前がつけられて，補助記憶（ストレージ）に保存されるデータ単位
  - 通常は“静的”＝保存した状態のまま，消滅したり変化したりしない
  - 例） プログラム，文書，ソフトウェアやOSの設定情報，その他…
- ファイルシステム
  - 補助記憶装置のボリュームに，ファイルを格納する方式・構造
  - 名前から情報を検索する一種のデータベースともいえる
- ファイル管理
  - ファイルシステムの構造（階層型など）を維持・管理するOSの機能
  - ファイルの作成・保存・読み込み，ファイルの検索，ファイルの保護
  - その他…

ファイルの形式

- ファイルの内部構造
  - 大型汎用機のOS：表構造を基本としたいくつかのファイル編成法が決められており，データの種類やアクセス方法によって使い分ける
  - Windows，UNIX等：OSはファイルの内部構造に関与せず，バイト列（バイトストリーム）として扱う ⇒ 内部構造は各ソフトに任されている
- ファイルのアクセス方法
  - 順次（sequential）アクセス：先頭のデータから順々にアクセスする
  - 直接（direct/random）アクセス：途中位置を指定してアクセスできる
  - 記憶媒体（メディア）によって，サポートされるアクセス方法が異なる
- ファイルの種類
  - ファイルには，通常その種類を表すデータ項目が付加される
  - ファイル名の一部に含まれる場合は，拡張子や接尾辞と呼ばれる

ファイル操作API

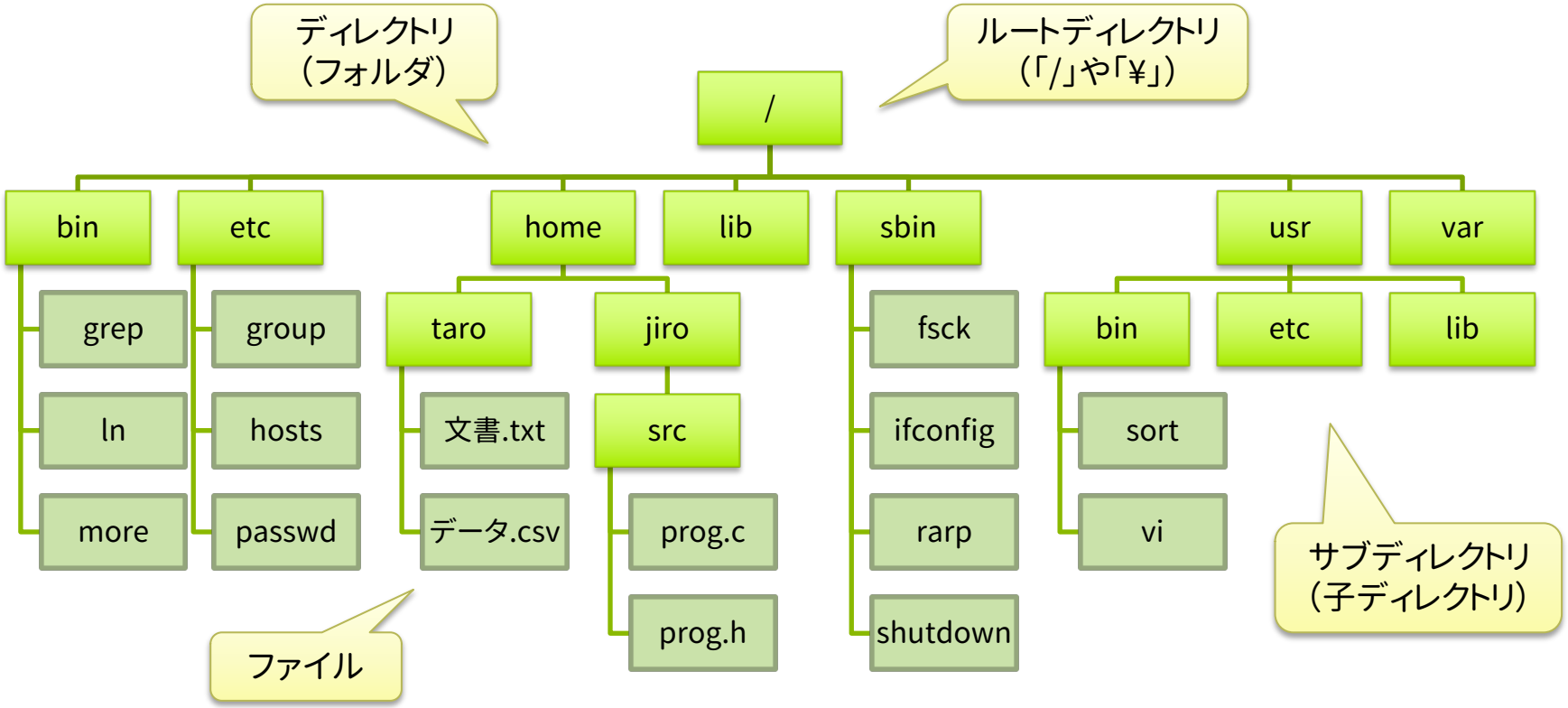
- ストリーム型ファイル入出力
  - UNIXやWindowsで標準的な方法
  - ファイルの中身は，バイト列として読み書きできる
  - open　　名前を指定してファイルを開く（必ず必要な操作）
  - read　　開いたファイルからデータを順に読み込む
  - write　　開いたファイルにデータを順に書き込む
  - seek　　ファイル内の自由な位置に移動する（ランダムアクセス）
  - close　　ファイルを閉じる
- ファイルのロック
  - 複数のプロセスによる同時アクセスを防ぐ排他アクセス機能を提供
  - Windows: ファイルを開くと自動的に全体がロックされる
  - UNIX系: 通常はロックされず，標準のロック機能も強制力が弱い

メモリマップトファイル

- メモリマップトファイル
  - ディスク上のファイルの内容を主記憶に“貼り付ける”アクセス方法
  - メモリを読み書きすると，ファイルの内容の対応部分が書き換わる
  - 仮想記憶の技術を応用して，ディスク領域を主記憶に貼り付ける
- メモリマップトファイルの用途
  - プロセス起動時のOSによるプログラムのロード
  - 複数のプロセスで共有するメモリの実現（ファイルとして共有）
  - ファイルに対する読み書き処理の高速化
- メモリマップトファイルのAPI
  - UNIX, macOS等：mmap(), munmap(), msync(), など
  - Windows：CreateFileMapping(), MapViewOfFile(), など
  - その他，Java，C#，Pythonなどの言語でサポートされている

階層型ファイルシステム

- UNIXやWindows
  - ディレクトリやフォルダの階層（木構造）でファイルを管理する



ディレクトリ(フォルダ)

- ディレクトリ操作
  - ディレクトリのファイル一覧　　Windows: dir　UNIX: ls
  - ディレクトリの作成/削除　　mkdir / rmdir
  - カレントディレクトリの変更　　cd (chdir)
- カレント(ワーキング)ディレクトリ
  - 実行中のプロセスは,「現在作業中のディレクトリ」という情報を持つ
  - デフォルトでは,各種ファイル操作はカレントディレクトリが対象になる
  - プロセスは,自由にカレントディレクトリを移動することができる(cd)
- 特別なディレクトリ記法
  - .　そのディレクトリ自身を表す
  - ..　親ディレクトリ(ひとつ上のディレクトリ)を表す

ディレクトリのしくみ

- ディレクトリファイル
  - 「フォルダ」(または「ディレクトリ」)も一種のファイルとして実現されるのがその中身には,ファイルの一覧表(名前とファイル番号)が含まれる

ディレクトリファイル

ファイル名	番号
aaa.txt	3410
bbb	1034
ccc.c	4566
ddd/	10035
…	

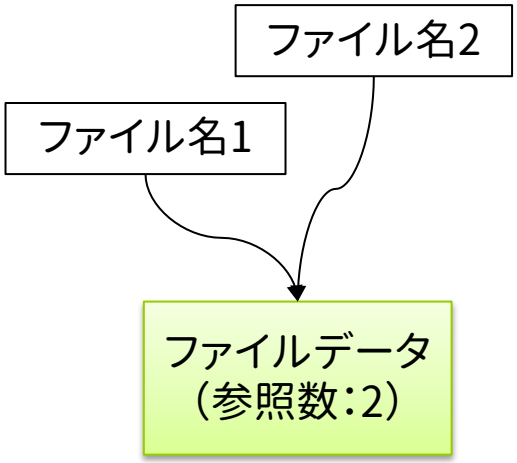
- ボリュームのマウント
  - ディレクトリの内容として,他のボリュームを「接ぎ木」すること
  - 複数のファイルシステムを結合して,1つの階層構造を構成する
  - UNIX系では「mount」,Windowsでは「mountvol」コマンドを利用
- 特殊ファイルとディレクトリ
  - UNIXでは /dev にデバイス等を操作できる特殊ファイルが配置
  - デバイスファイルによってディスク装置全体などへの操作も可能

パス(PATH)

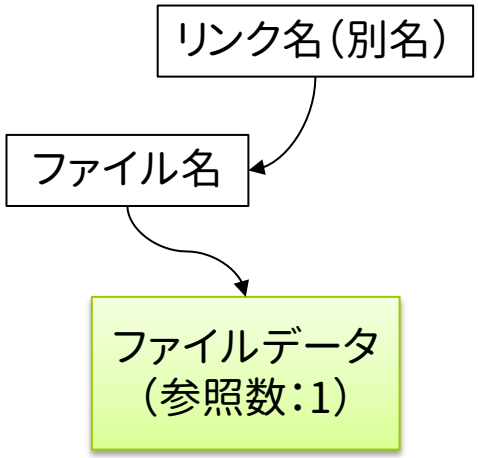
- パス
  - 階層型ファイルシステムの中で，ファイルの位置を表す記法
- 絶対パス
  - ルート(root)ディレクトリからたどった位置で示す
  - 例) C:¥Windows¥system32¥drivers¥etc¥hosts
  - 区切り記号は，UNIXでは「/」，Windowsでは半角「＼」(¥)
- 相対パス
  - あるディレクトリからの相対位置を示す
  - 1階層上のディレクトリ(親ディレクトリ)を「..」で示すことができる
  - 例) cd ../../kadai
    2つ上のフォルダの下にあるkadaiというフォルダに移動する

リンク

- ハードリンク
  - 複数のファイルまたはディレクトリ名から共通のデータ(実体)に直接つながるリンク
  - データ側が参照数を持ち,0になったら削除
  - UNIX: lnコマンドで作成(スーパーユーザ)
  - Windows: mklink /h コマンドで作成



- ソフトリンク
  - リンク先のパス名を文字列で保持するリンク
  - アクセスされるとリンク先を自動的にたどる
  - UNIX: シンボリックリンク(ln -sで作成)
  - Windows: ショートカット(GUI / 拡張子.lnk),シンボリックリンク(mklink),ジャンクション
  - macOS: エイリアス(GUI),シンボリックリンク

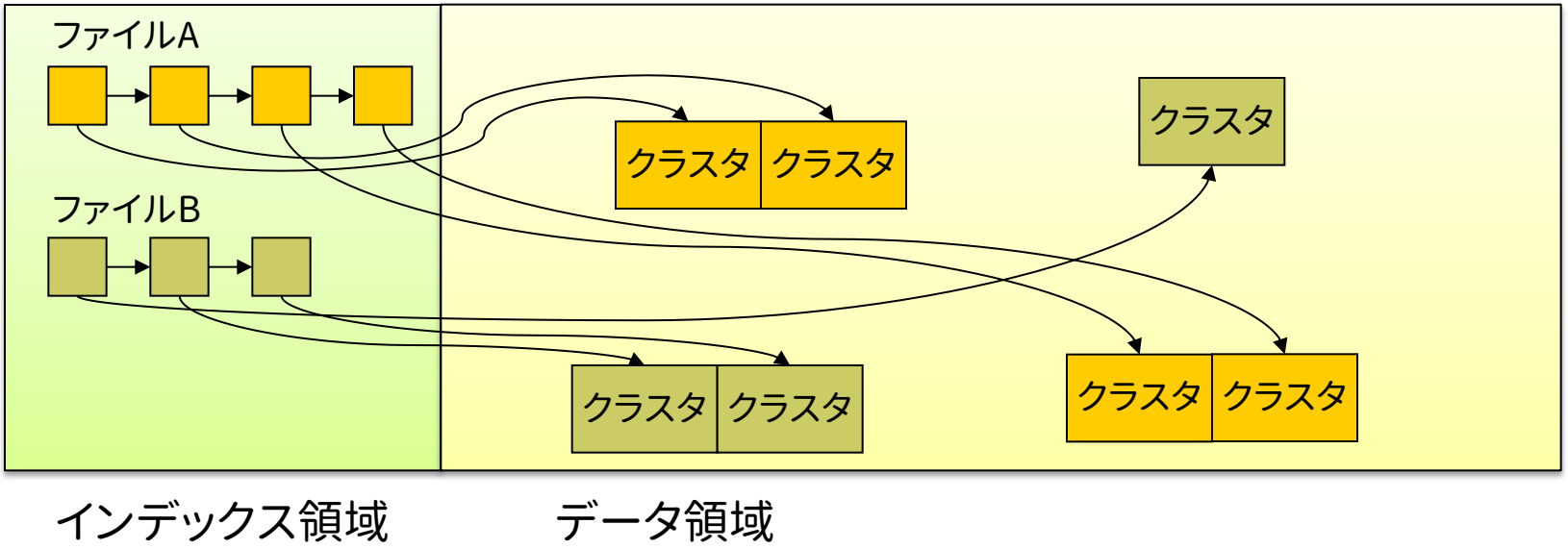


ファイルの所有権と属性

- ファイルの所有権
  - マルチユーザシステムでは，ファイルは所有者（owner）情報を持つ
  - 所有者（と管理者）だけに，ファイルの属性を変更する権限がある
  - さらに，所有グループも設定できるのが一般的である（UNIX系など）
- ファイルの属性
  - 作成日，変更日，内容の読み・書きの許可，ディレクトリー覧許可，隠しファイル，実行許可（プログラムとして実行できるか），など
- アクセス制御
  - UNIXでは，各ファイルは，所有者（u），所有グループ（g），その他（o）に対して，読み（r），書き（w），実行（x）の許可を設定できる
  - Windowsでは，アクセス制御リスト（ACL: Access Control List）によって，個別のユーザに対して細かなアクセス許可を登録できる

ファイルシステムの構造

- ファイルシステムの実現方式
  - どんなタイプのファイルを考えるか？
  - クラスタをどうつなげてファイルを保存するか？
  - 階層型（ディレクトリ・フォルダ）をどう実現するか？
  - ドライブ内部をインデックス領域とデータ領域に分けるのが一般的



FATファイルシステム

- FAT（File Allocation Table）
  - MS-DOS（および昔のWindows）の方式
  - 各ディレクトリの領域にクラスタのリンクを示す“表”（FAT）を格納
  - FATの各セルに対応するクラスタの次のクラスタの番号を保持する
  - 単純で高速だが事故に弱い（ルートディレクトリのFATが壊れると…）

FAT

ここから ↓

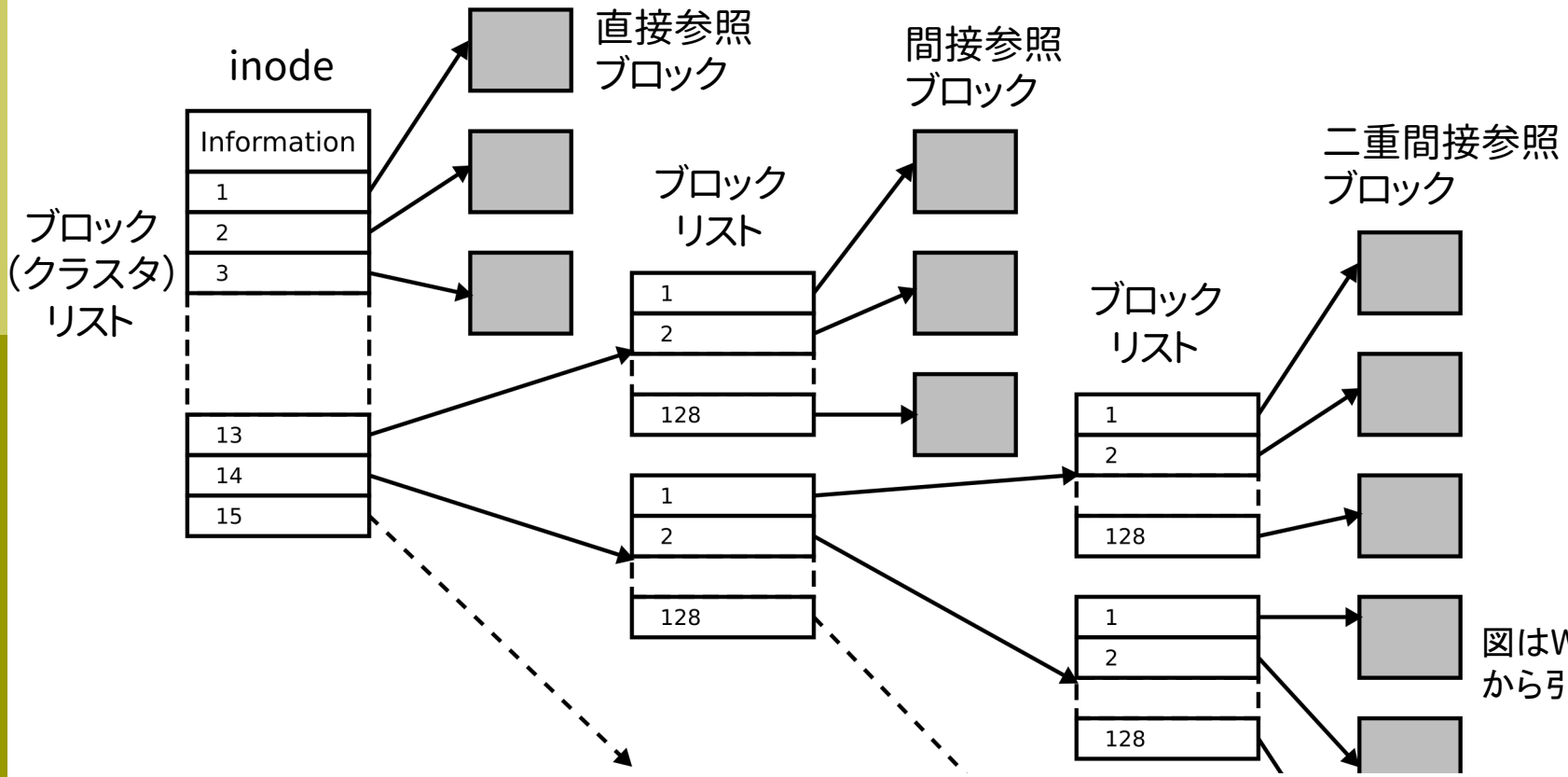
クラスタ番号	0	1	2	3	4	5	6	7	8	9	10
リンク	X	X	03	04	07	06	FF	05	00	01	F7

データ領域

クラスタ番号	0	1	2	3	4	5	6	7	8	9	10
内容	X	X	File A #1	File A #2	File A #3	File A #5	File A #6	File A #4	空き	予約	不良

UNIX iノード方式

- inode (UNIX/Linux系)
  - 各ファイルの情報は，インデックス領域の「inode」に記録される
  - 各inodeからは，クラスタのつながりを表す多分木が構成される



図はWikipediaから引用

Windows NTFS

- NTFS
  - Windows（NT以降）のファイルシステム
  - インデックス領域として，MFT（Master File Table）が使われる

小さいファイルのMFTレコード

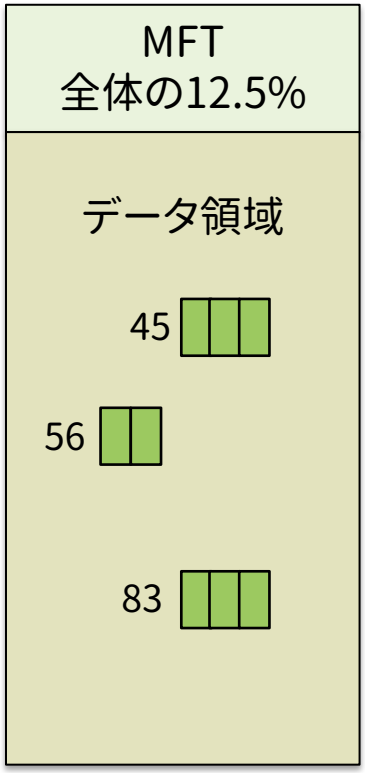
標準情報	ファイル名	データ（ファイル本体）

大きいファイルのMFTレコード

標準情報	ファイル名	全長	位置	長さ	位置	長さ	…

例	8	45	3	56	2	83	3	未使用

通常1Kバイトの固定長



高速性と信頼性

- ディスクキャッシュ
  - 補助記憶装置から読み込んだデータを，主記憶等にしばらく保持する
  - 同じ領域を頻繁に読み書きする場合に，劇的に速度が改善する
- ジャーナリングファイルシステム
  - ファイルの変更履歴を保持していくファイルシステム
  - 何らかの事故が起きたときに，（可能な限り）前の状態に復旧できる
- ディスクアレイ（RAID等）
  - 複数のドライブを統合し，仮想的に1台のドライブとして操作できる
- 暗号化ファイルシステム
  - ドライブ全体またはファイルを暗号化し，アクセス時に随時復号化する
  - 盗難・紛失等への対策，ユーザ別のセキュリティの強化

RAID

- Redundant Arrays of Inexpensive Disks
  - 複数のドライブを仮想的に1台のドライブとして結合する
  - データに加えて，誤り訂正符号を分散して記録する（冗長性）
  - 信頼性の向上や無停止運転を実現できる
- RAID1
  - 単に2台のドライブを常に同じ内容に保つ（ミラーリング）
- RAID5
  - 3台以上のドライブにデータとその誤り訂正符号を分散して記録する
  - 1台が故障しても復旧できる
- RAID6
  - 4台以上のドライブにデータとその誤り訂正符号を分散して記録する
  - 2台が故障しても復旧できる

ファイルシステムの拡張

- 仮想ファイルシステム
  - あるOSにおいて，ファイルシステムが備えるべき共通のAPI群
  - 直接接続された補助記憶装置以外でも，「仮想ファイルシステム」の機能を提供（継承）すれば，ファイルシステムとして操作可能になる
- ネットワークファイルシステム
  - サーバのドライブをクライアントから透過的にアクセスできる機能
  - Windows SMB/CIFS（Samba），Unix NFS
  - 近年，NAS（Network Attached Storage）装置として販売
- 仮想ドライブ
  - 記憶装置の内容を複製したファイルを，外部のドライブのように扱う
  - 例） CD/DVDのISOイメージファイル，macOSの.dmgファイル

その他のトピック

- ファイルの断片化
  - ファイルを構成するクラスタがドライブの中で分散していき，アクセス（シーク，読み書き）時間が低下する（フラグメンテーション）
  - 再配置（デフラグメンテーション）することで，速度が回復する
- バックアップ
  - フルバックアップ：全データをバックアップ（磁気テープ等を使用）
  - 差分バックアップ：フルバックアップからの差分を毎回バックアップ
  - 増分バックアップ：前回バックアップからの変更分のみバックアップ
  - 例）macOS Time Machine, Windows ファイル履歴
- 検索機能/通知機能
  - Windows Search, macOS Spotlight
  - OSは，ファイルの変更がソフトウェアに通知される仕組みを提供

演習課題

- 課題 12a ファイルの内部構造の表示
  - この課題の狙いは，メモリマップトファイルのプログラミングを体験するとともに，各種ファイルの内部形式を見てみることである。
  - ファイル（テキストファイル以外）の先頭には，種類を表す数バイトの「マジックナンバー」が格納されていることが多い。
- 手順
  - 次ページのプログラムは，メモリマップトファイルを用いて，ファイルの内容の指定位置1バイトを16進数で表示するJavaプログラムである。
  - このプログラムを，ファイルの先頭128バイトを1行に16バイトずつの「16進ダンプ形式」で表示するプログラムに修正せよ。
  - さらに，実行ファイル（.exe），画像ファイル（.png等），音楽ファイル（.mp3等），ZIPファイルなどから，2つ以上の種類（拡張子）を選び，
  - 同じ拡張子の複数のファイルの先頭の16進ダンプを出力して比較し，拡張子の種類ごとに共通点を考察せよ。

Javaでメモリマップトファイル

```
CF FA ED FE 07 00 00 01 03 00 00 80 02 00 00 00
0F 00 00 00 00 05 00 00 85 00 20 00 00 00 00 00
19 00 00 00 48 00 00 00 5F 5F 50 41 47 45 5A 45
52 4F 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 19 00 00 00 D8 01 00 00
5F 5F 54 45 58 54 00 00 00 00 00 00 00 00 00 00
```

こうする

```
import java.io.RandomAccessFile;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("File path: ");  String fname = sc.nextLine();
        System.out.print("Position : ");  int position = sc.nextInt();

        try (RandomAccessFile file = new RandomAccessFile(fname, "r")) {
            FileChannel channel = file.getChannel();
            MappedByteBuffer in
                = channel.map(FileChannel.MapMode.READ_ONLY, 0, channel.size());
            byte b = in.get(position);
            System.out.printf("%02X", b);
            System.out.println();
        } catch (Exception e) { System.out.println(e.getMessage()); }
    }
}
```

ファイル内容の16進数表示

```
#include <stdio.h>

int main(void)
{
    char fname[256];
    FILE *fp;
    int ch;

    printf("File name (path) ? ");
    scanf("%s", fname);
    printf("%s¥n", fname);

    fp = fopen(fname, "rb");
    ch = getc(fp);
    printf("%02X¥n", ch);

    return 0;
}
```

```
CF FA ED FE 07 00 00 01 03 00 00 80 02 00 00 00
0F 00 00 00 00 05 00 00 85 00 20 00 00 00 00 00
19 00 00 00 48 00 00 00 5F 5F 50 41 47 45 5A 45
52 4F 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 19 00 00 00 D8 01 00 00
5F 5F 54 45 58 54 00 00 00 00 00 00 00 00 00 00
```

ファイルの内容の先頭128バイトが
このように表示されるように修正する
（このような表示を「16進ダンプ」という）

Visual Studioの場合は、scanfの行を
scanf_s("%s", fname, 256);に変更