オペレーティングシステム

第12回

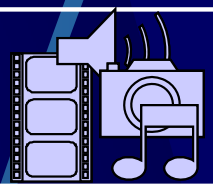
ファイルシステム(1)

http://www.info.kindai.ac.jp/OS

E号館3階E-331 内線5459

takasi-i@info.kindai.ac.jp

ファイルシステム(file system)



アプリケーションプログラム

制御

ハードウェアを論理的に扱いたい

データの共通規格が必要

ファイルシステム(file system)

ハードウェア

機械語, 物理デバイス
マイクロプログラム 等

ファイル(file)

- ファイル(file)
  - データ, プログラムの集合体
  - データ, プログラムを格納するための論理単位
  - 格納された情報は永続性(persistent)がある
  - 任意の時点で作成可能
  - 大きさを拡大・縮小可能
  - 大きさに制限無し
  - プロセス間で共有可能

ファイルシステム(file system)

- ファイルシステム(file system)
  - ファイル操作の統一的な方法を提供
  - DOS (disk operation system)によりディレクトリとファイルを構成するシステム
    - 膨大な量の情報を格納可能
  - プロセスが終了しても残存
    - 複数のプロセスが同時に情報を共有可能

ファイルシステムの目的

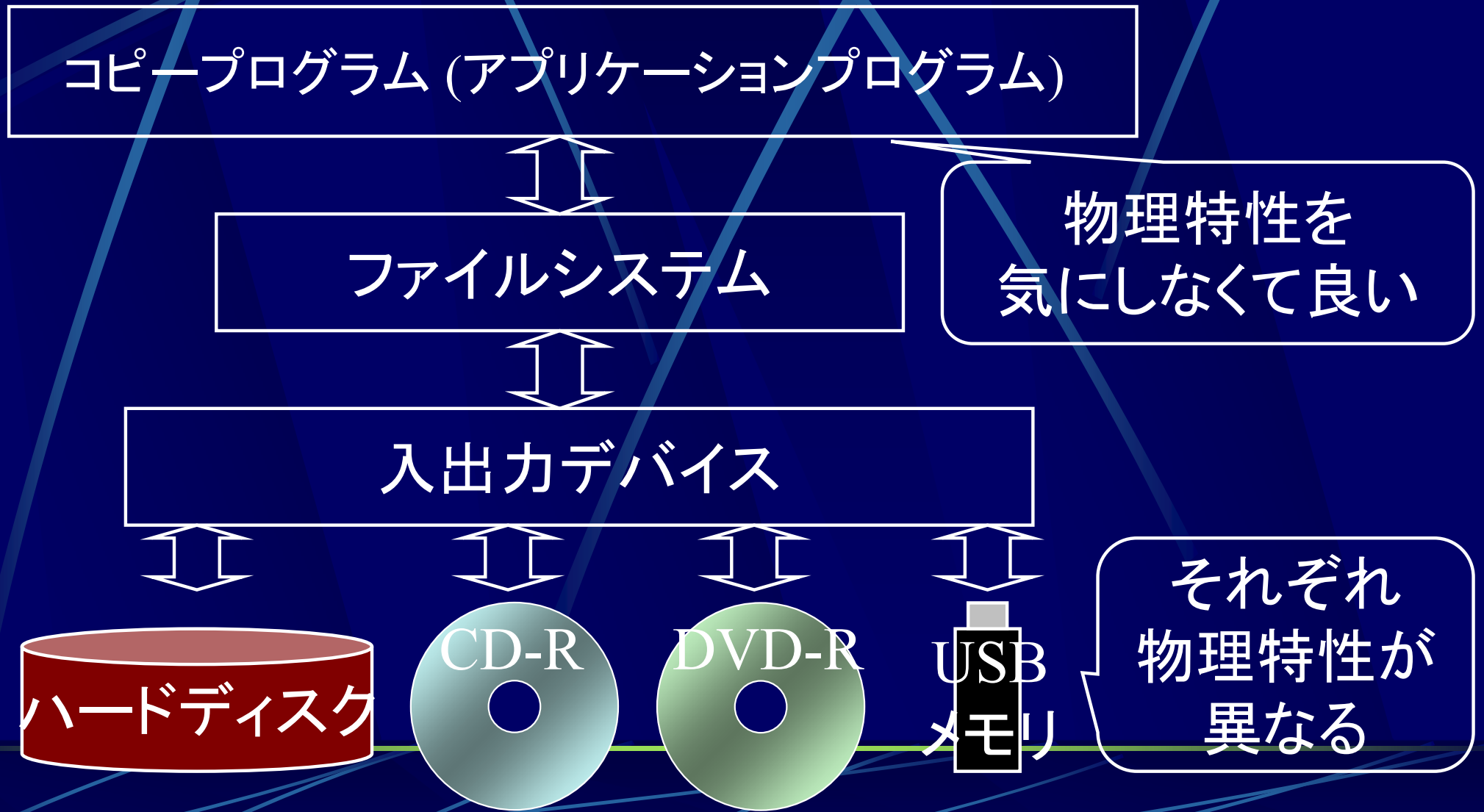
- データの信頼性を保証する
- ハードウェアの性能を引き出す
- ハードウェアを使い易くする
  - ファイルはハードウェアに依存しない

異なるハードウェアに同一の
プログラムを使用可能

装置独立性(device independence)

ファイルシステムの目的

例：データのコピー



ファイルの種類

- ファイルの種類
  - ソースプログラム(source program)
  - オブジェクトプログラム(object program)
  - バイナリプログラム(binary program)
  - バイナリデータ(binary data)
  - テキストデータ(text data)
  - 画像データ(image data)

：　　種類毎に適した構造/アクセス法

ファイル名

- ファイル名
  - 自由に設定可能
  - 同一ファイルを複数の名前で参照
  - ピリオドで区切られた名前が多用される
    - 例 : hello.c, sum.java, report.txt, picture.bmp 等

      拡張子

      ファイルの種類を示すために使用

    複数の拡張子が使われる場合もある
    - 例 : hello.c.gz 等

ファイル名

命名規則はOSにより異なる

OS	文字数制限	使用不可文字	大文字小文字
MS-DOS	8.3形式 (ファイル名8文字, 拡張子3文字まで)	¥/:*?"<>|	区別無し
Windows 95 以降	ロングファイル名 (255文字まで, 空白可)	¥/:*?"<>|	区別無し
UNIX	255文字まで, 空白可	/	区別有り
MAC OS 9 以前	31文字まで	:	区別無し
MAC OS X 以降	255文字まで	/:	区別無し

ファイル構造(file structure)

- ファイル構造(file structure)
  - 論理構造
    - プログラム等で扱うソフトウェアレベルでの構造
  - 物理構造
    - 2次記憶上で扱う物理的なファイルの格納構造

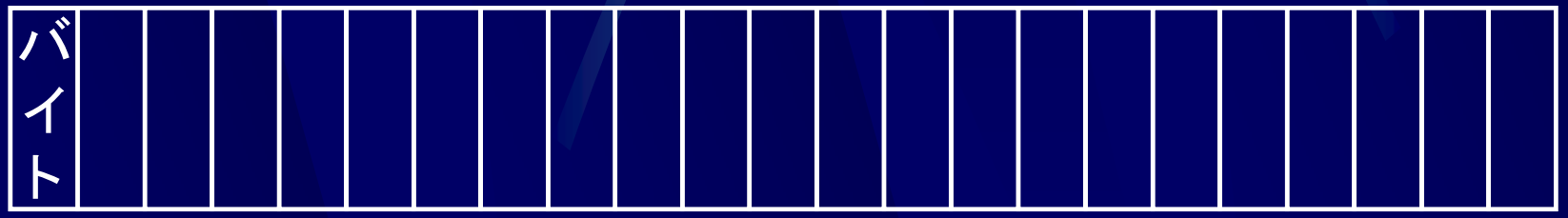
(※)通常は「ファイル構造」とは論理構造のこと

ファイル構造

- ファイル構造
  - バイト列
    - 特に構造無し
    - UNIX, MS-DOS
  - レコード列
    - アプリケーションが規定する情報の単位
  - 属性列
    - 様々な情報を持つ
    - Windows , MAC OS

ファイル構造
バイト列

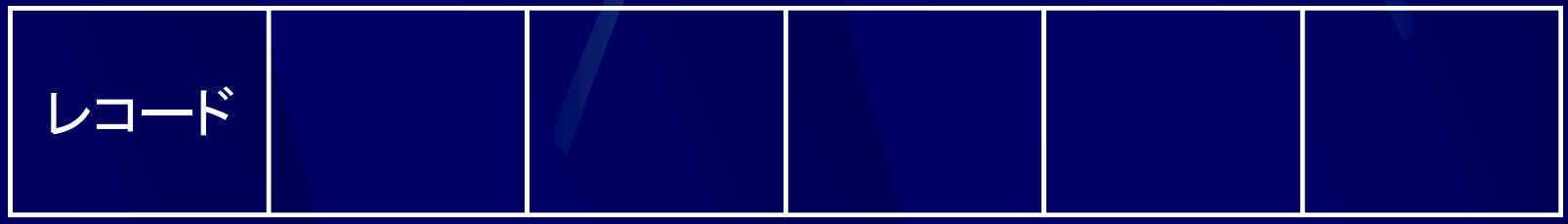
- バイト列
  - ファイルはバイトの並びとして扱う
    (ASCIIファイル, バイナリファイル)
  - 特に何の構造も無し
  - OSは構造について関知せず
    (アプリケーションに任せる)



UNIX : 全てのファイルは単なるバイト列

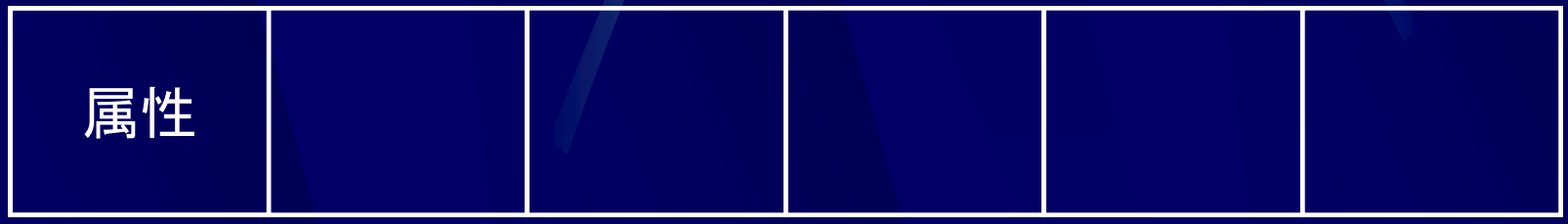
ファイル構造
レコード列

- レコード列
  - ファイルは固定長のレコードの並びとして扱う
  - レコード単位で読み書き可能
    - レコード：アプリケーションが指定する情報の単位



ファイル構造
属性列

- 属性列
  - ファイルはデータだけでなく様々な情報を持つ
    - ファイルの大きさ, 所有者, グループ, アクセス時刻, inode 番号等



Windows, MAC OS : 全てのファイルは属性を持つ

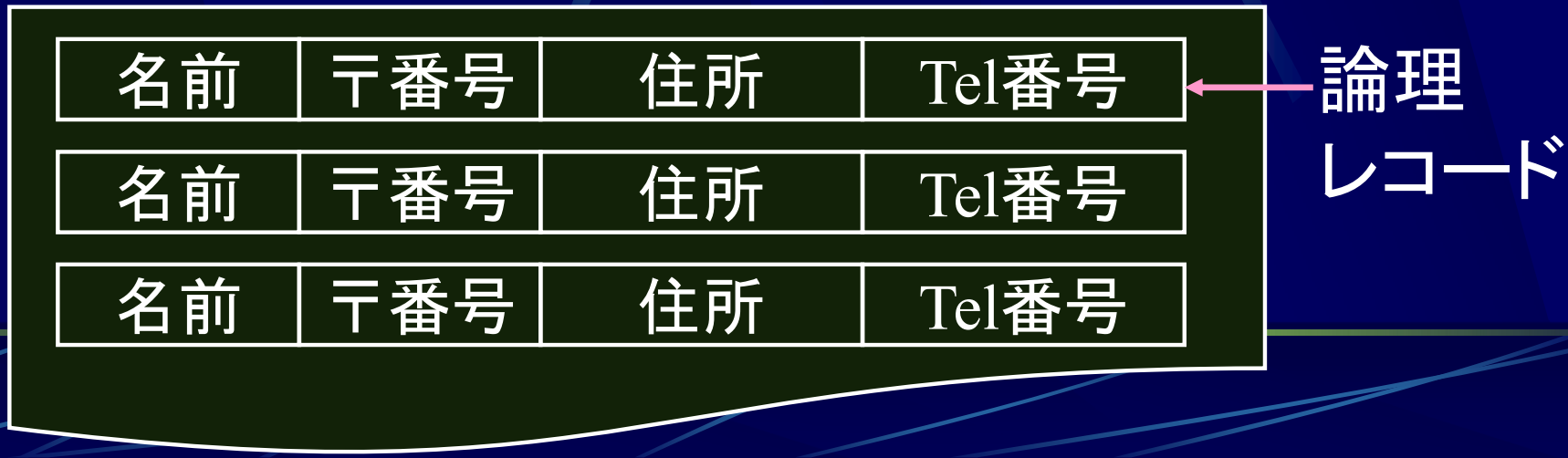
レコード

- 論理レコード
  - ユーザが定めたデータの単位
- 物理レコード，ブロック
  - 物理的に読み書きするデータの単位

論理レコード(logical record)

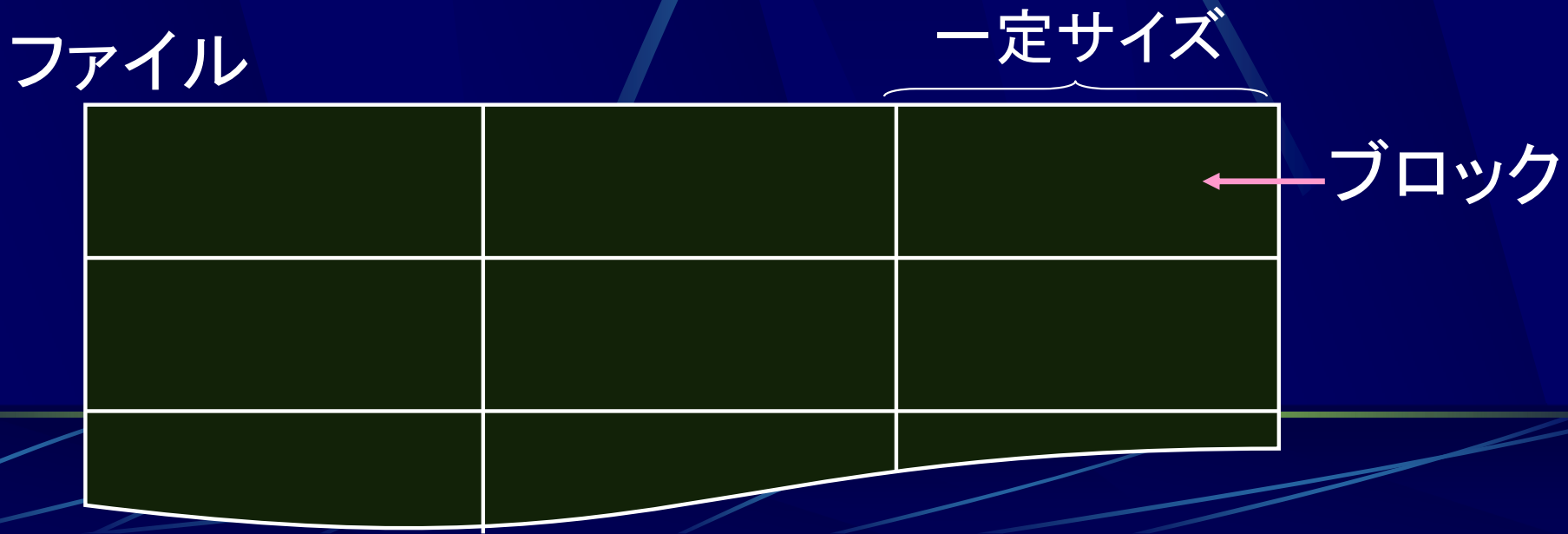
- 論理レコード(logical record)
  - ユーザが1単位として扱うデータの集合
    - 例：住所録：名前, 郵便番号, 住所, 電話番号等
    - 例：学籍簿：名前, 学籍番号, 住所, 電話番号, 各科目の成績, 所属研究室等

ファイル



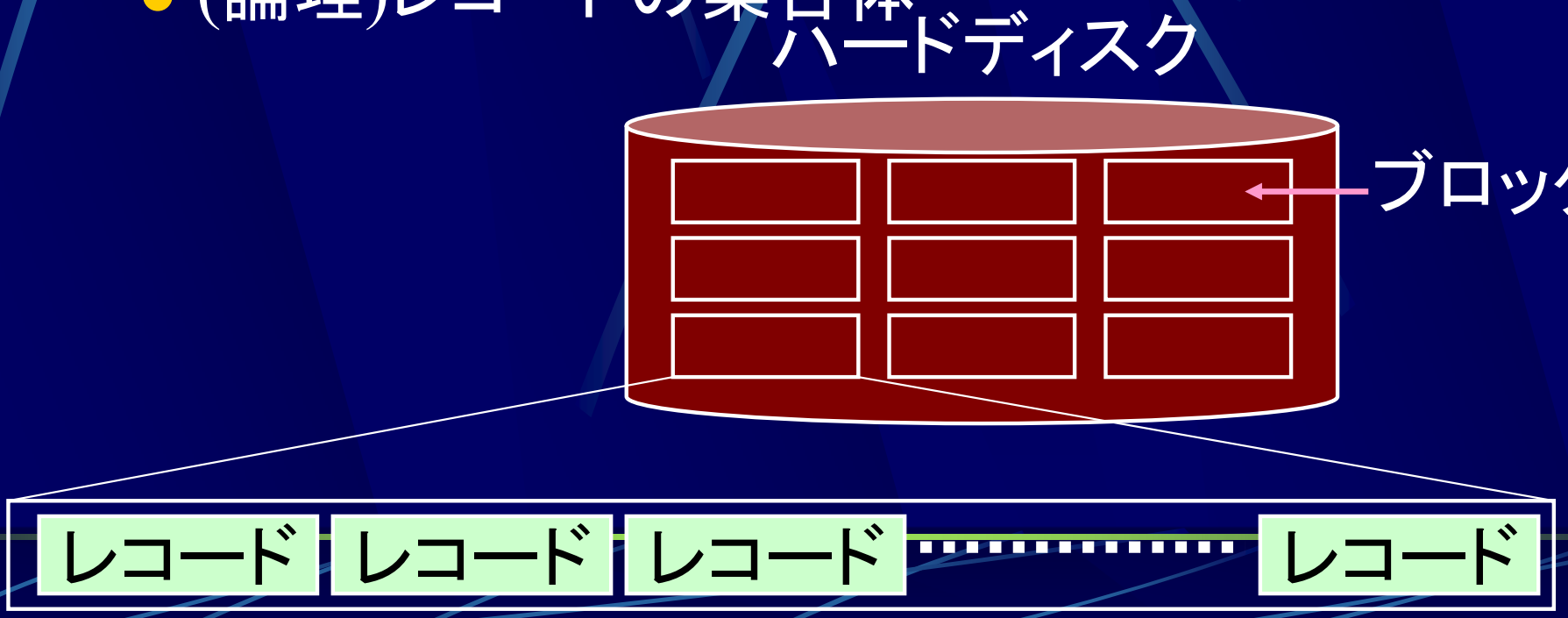
物理レコード(physical record)
ブロック(block)

- 物理レコード(physical record)
- ブロック(block)
  - 記憶装置に読み書きする情報の単位
    - 例：パンチカード：1レコード80文字
    - 例：ラインプリンタ：1レコード132文字



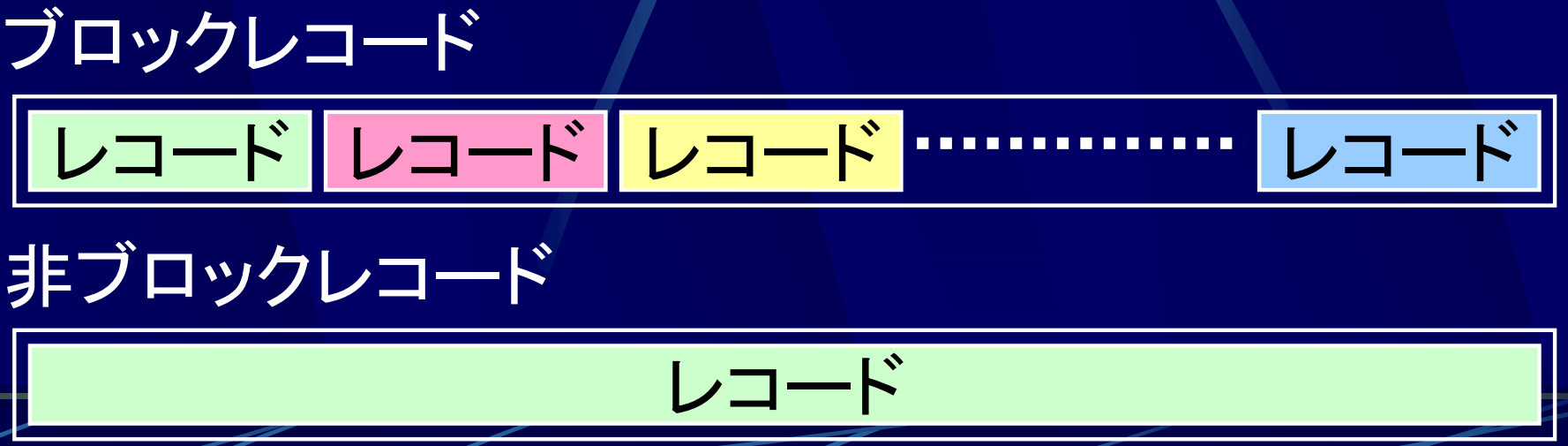
物理レコード,ブロック

- 物理レコード
- ブロック
  - 記憶装置上の記録の単位
  - (論理)レコードの集合体



ブロック/非ブロックレコード
(blocked / unblocked record)

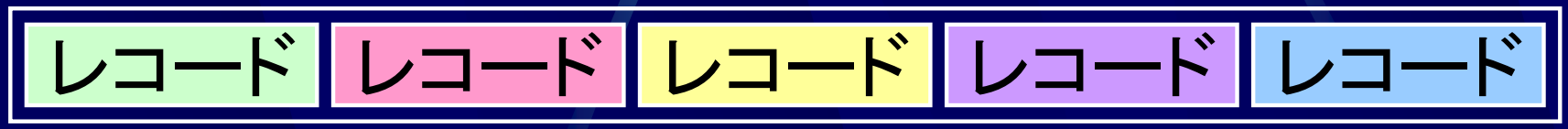
- ブロックレコード(blocked record)
  - 複数の論理レコードで構成された物理ブロック
- 非ブロックレコード(unblocked record)
  - 1つの論理レコードで構成された物理ブロック



固定長/可変長レコード
(fixed / variable length record)

- 固定長レコード(fixed length record)
  - ブロックサイズはレコード長の整数倍
- 可変長レコード(variable length record)
  - レコード長はブロック内で可変

固定長レコード

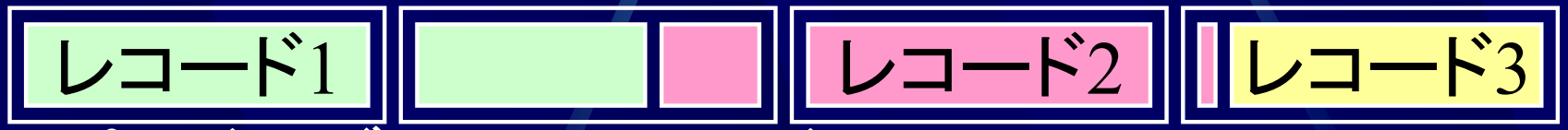


可変長レコード

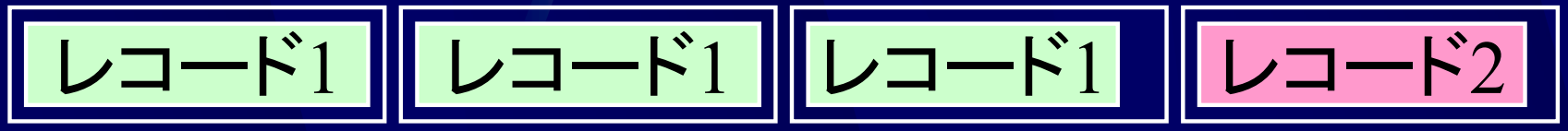
スパンドレコード
(spanned record)

- スパンドレコード(spanned record)
  - ブロック長よりも大きなレコード
    - 複数のブロックにまたがって格納される
    - 可変長レコードの特殊な形式

スパンドブロックレコード



スパンド非ブロックレコード



ブロック1　ブロック2　ブロック3　ブロック4

レコード形式

	ブロックレコード (1ブロック複数レコード)	非ブロックレコード (1ブロック1レコード)
固定長レコード (レコード長固定, n*レコード長＝ブロック長)	固定長 ブロックレコード 	固定長 非ブロックレコード 
可変長レコード (レコード長可変, レコード長≦ブロック長)	可変長 ブロックレコード 	可変長 非ブロックレコード 
スパンドレコード (レコード長可変, レコード長>ブロック長)	スパンド ブロックレコード 	スパンド 非ブロックレコード 

ファイルの型

- 通常ファイル(regular file)
  - ASCIIファイルまたはバイナリファイル
- ディレクトリ(directory)
  - ファイルを管理するためのファイル
- デバイスファイル(device file)
  特殊ファイル(special file)
  - 入出力関連のデバイスを示す

ファイルの型

- ASCIIファイル
  - 表示可能(人間が判読可能)
  - エディタで編集可能
  - 複数行から成る
  - 改行コード(CR, LF等)はシステムにより異なる

```
public class Hello {
    public static void main (String args[]) {
        System.out.print(“Hello! World!¥n”)
    }
}
```

ファイルの型

- バイナリファイル
  - 表示不可能(人間には判読不可能)
  - 何らかの構造を持つ
  - 実行可能なバイナリファイルもある

```
6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00
27 6A 4D 11 63 0B 23 42 63 0B 23 42 63 0B 23 42
A9 28 04 42 62 0B 23 42 99 2F 3E 42 6A 0B 23 42
99 2F 63 42 7A 0B 23 42 99 2F 3F 42 33 0B 23 42
B9 28 3E 42 62 0B 23 42 99 28 3A 42 67 0B 23 42
E0 03 7E 42 6E 0B 23 42 63 0B 22 42 C7 0B 23 42
63 0B 23 42 67 0B 23 42 99 2F 0F 42 0F 0B 23 42
```

ファイル操作

- ファイル操作
  - 作成, 削除
  - 開く, 閉じる
  - 読み出し, 書き込み, 追加
  - 一覧リスト表示

ファイル操作はシステムコールにより行う

ファイル制御ブロック(file control block)で管理

ファイル制御ブロック (file control block)

- ファイル制御ブロック, FCB(file control block)
  ファイル記述子(file descriptor)
  - ファイルの情報を格納

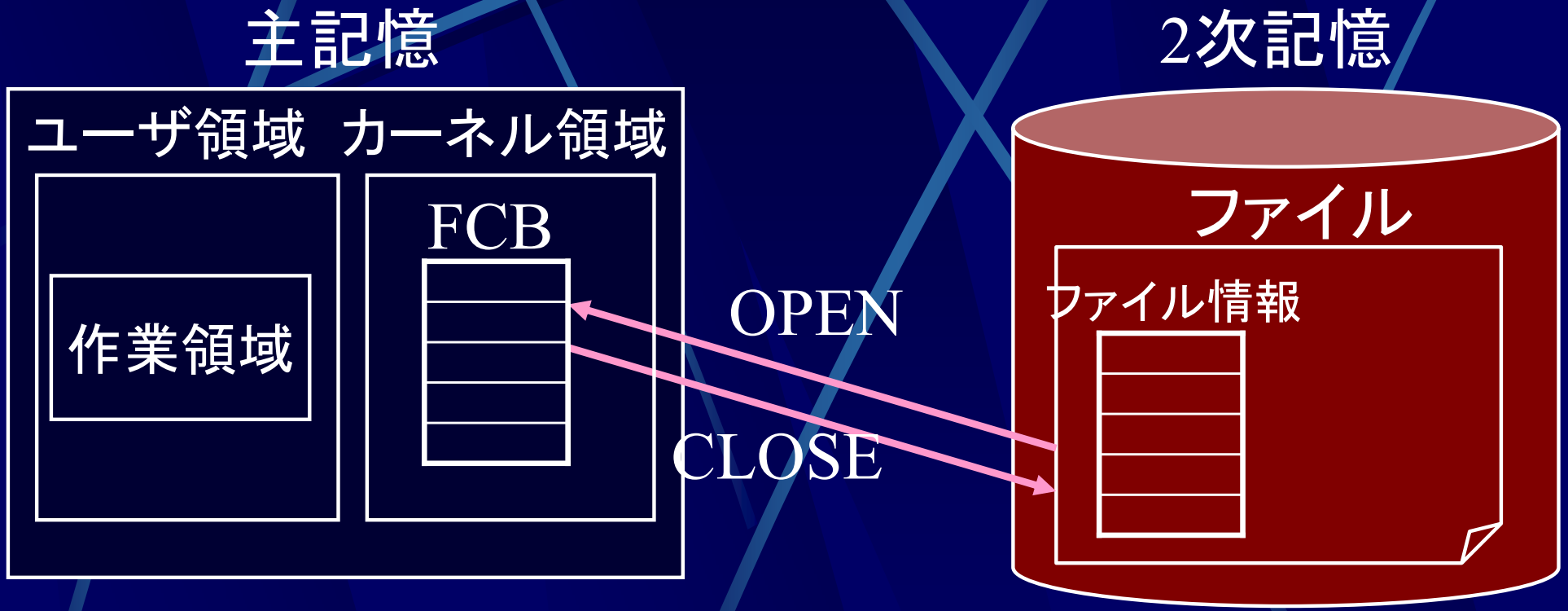
主記憶のカーネル領域に
領域が確保

c.f. プロセス制御ブロック
プロセス記述子
- プロセスの情報を格納

FCB

ファイル名
ファイルの型
サイズ
ファイル構造情報
ファイル保護情報
2次記憶上のアドレス
作成日時
最終変更日時
読み出し/書き込み回数

ファイル制御ブロック



ファイルOPEN時： FCBにファイル情報を読み込み
作業領域を確保

ファイルCLOSE時：2次記憶にFCBを書き出し
作業領域を開放

ファイルそのものは参照時にスワップイン

ファイル操作

システムコール	操作内容
CREATE	空のファイルを作成する
DELETE	ファイルを削除する
OPEN	ファイルを開く
CLOSE	ファイルを閉じる
READ	ファイルからメモリにデータを読み込む
WRITE	メモリからファイルにデータを書き込む
APPEND	メモリからファイルの末尾にデータを書き込む
SEEK	直接アクセスをする
GET ATTRIBUTE	ファイルの属性を読み取る
SET ATTRIBUTE	ファイルの属性を設定する
RENAME	ファイル名を変更する

ファイル操作

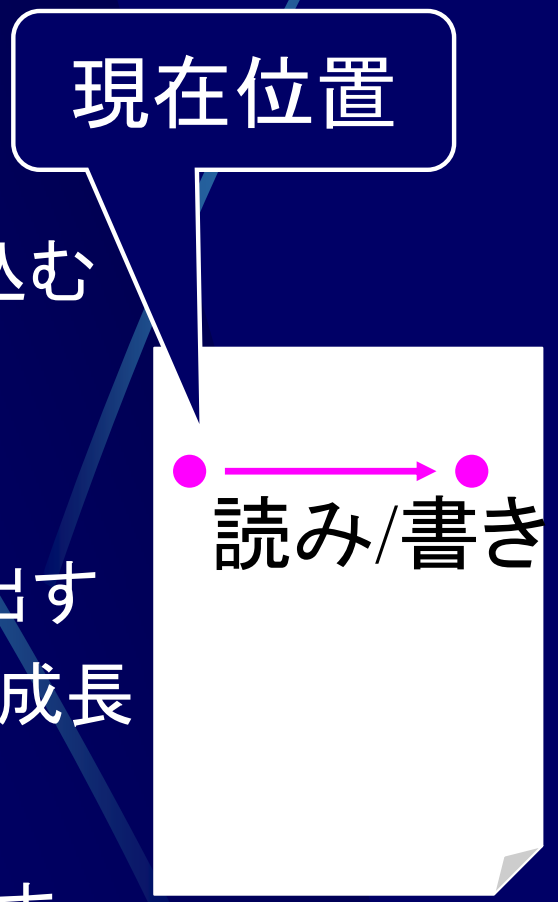
- OPEN
  - ファイルの属性等を二次記憶からFCBに読み込む
  - 作業領域の確保
- CLOSE
  - FCBの情報を二次記憶に書き出す
  - 作業領域の解放

多数のファイルを同時にOPENすると
大量の作業領域が必要になる

必要なファイルのみOPENした方がよい

ファイル操作

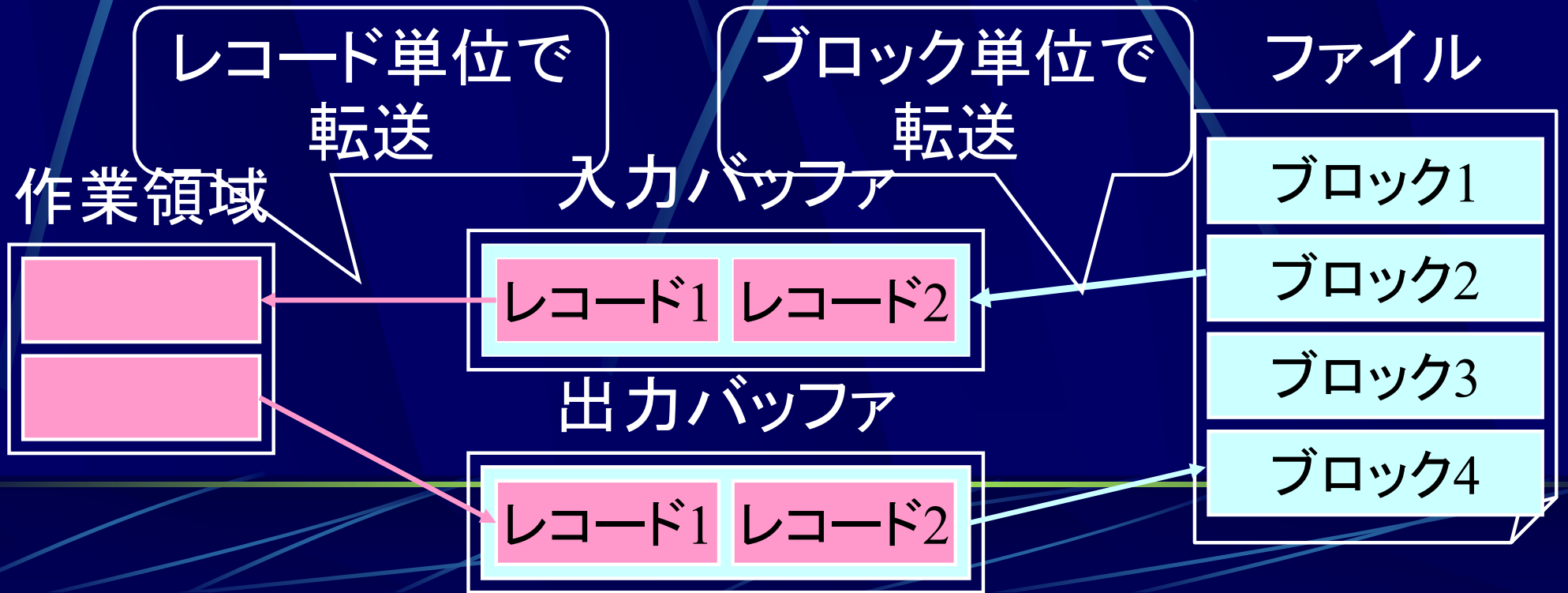
- READ
  - 現在位置から指定されたバイト数読み込む
  - 現在位置を指定されたバイト数進める
- WRITE
  - 現在の位置に指定されたデータを書き出す
  - 現在位置がファイル末ならばファイルは成長
- APPEND
  - ファイル末に指定されたデータを書き出す
  - 現在位置をファイル末に移動
- SEEK
  - 指定された位置に移動する



バッファリング(buffering)

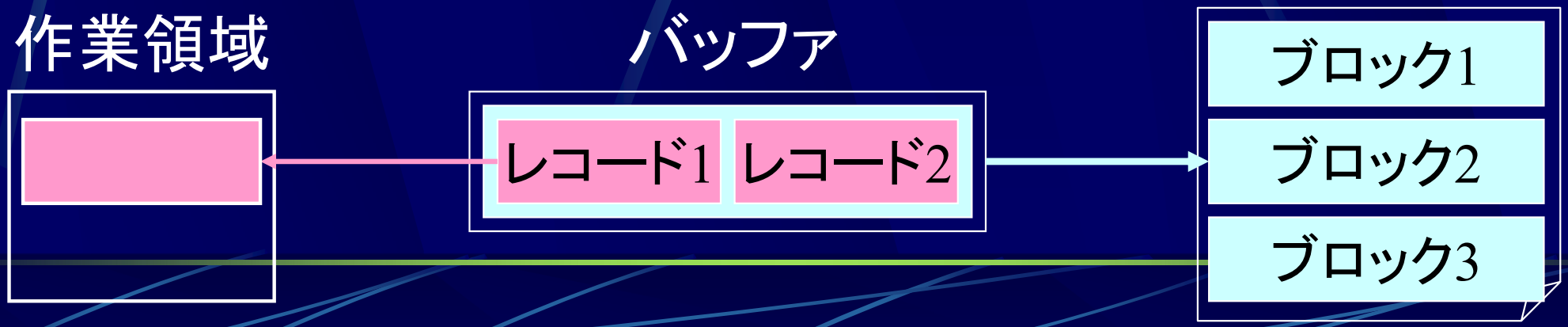
- バッファリング(buffering)
  - 参照時にファイルを一旦バッファに入れる
    - CPU処理と入出力処理のオーバラップ

    ⇒ CPU利用率が向上



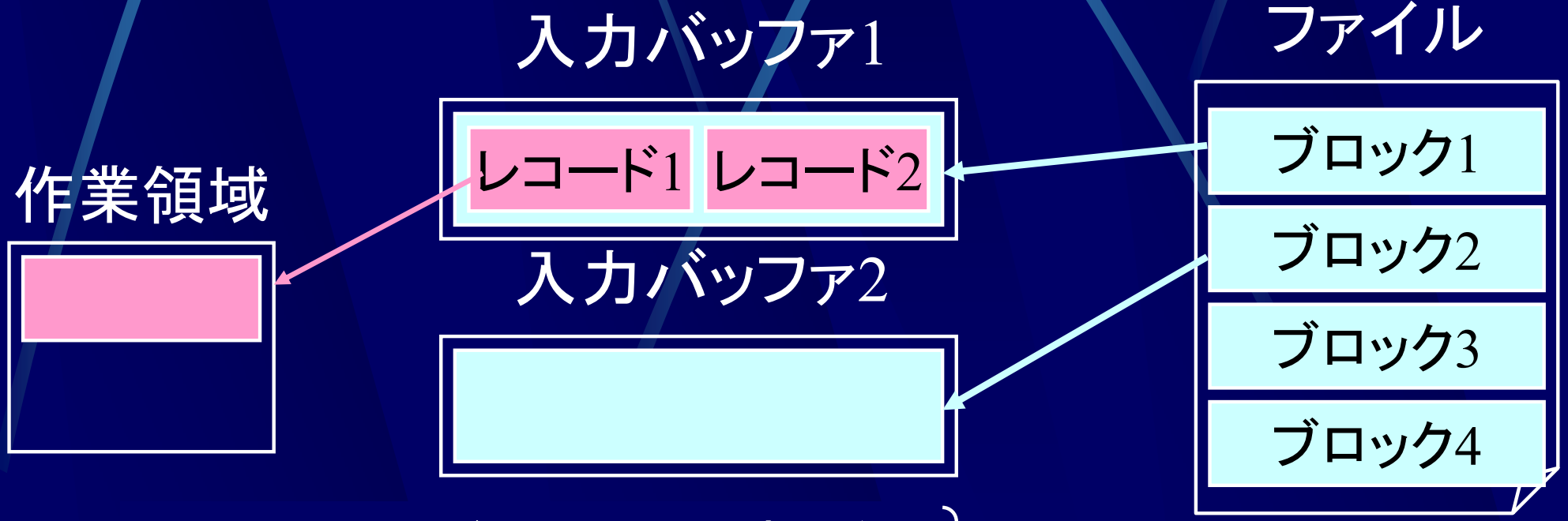
ブロッキング，デブロッキング (blocking, deblocking)

- ブロッキング(blocking)
  - 複数のレコードをブロックにまとめる
  - ファイルへの書き込み時
- デブロッキング(deblocking)
  - ブロックを複数のレコードに分割
  - 作業領域への読み込み時



ダブルバッファリング (double buffering)

- ダブルバッファリング(double buffering)
  - 2つのバッファを用意して交互に使用する



ファイル→バッファ2 の転送
バッファ1→作業領域 の転送
} 同時に可能

記憶へのアクセス方式

- 主記憶
  - アドレス(実アドレス, 仮想アドレス)
- 2次記憶
  - 管理すべき領域が膨大

↓

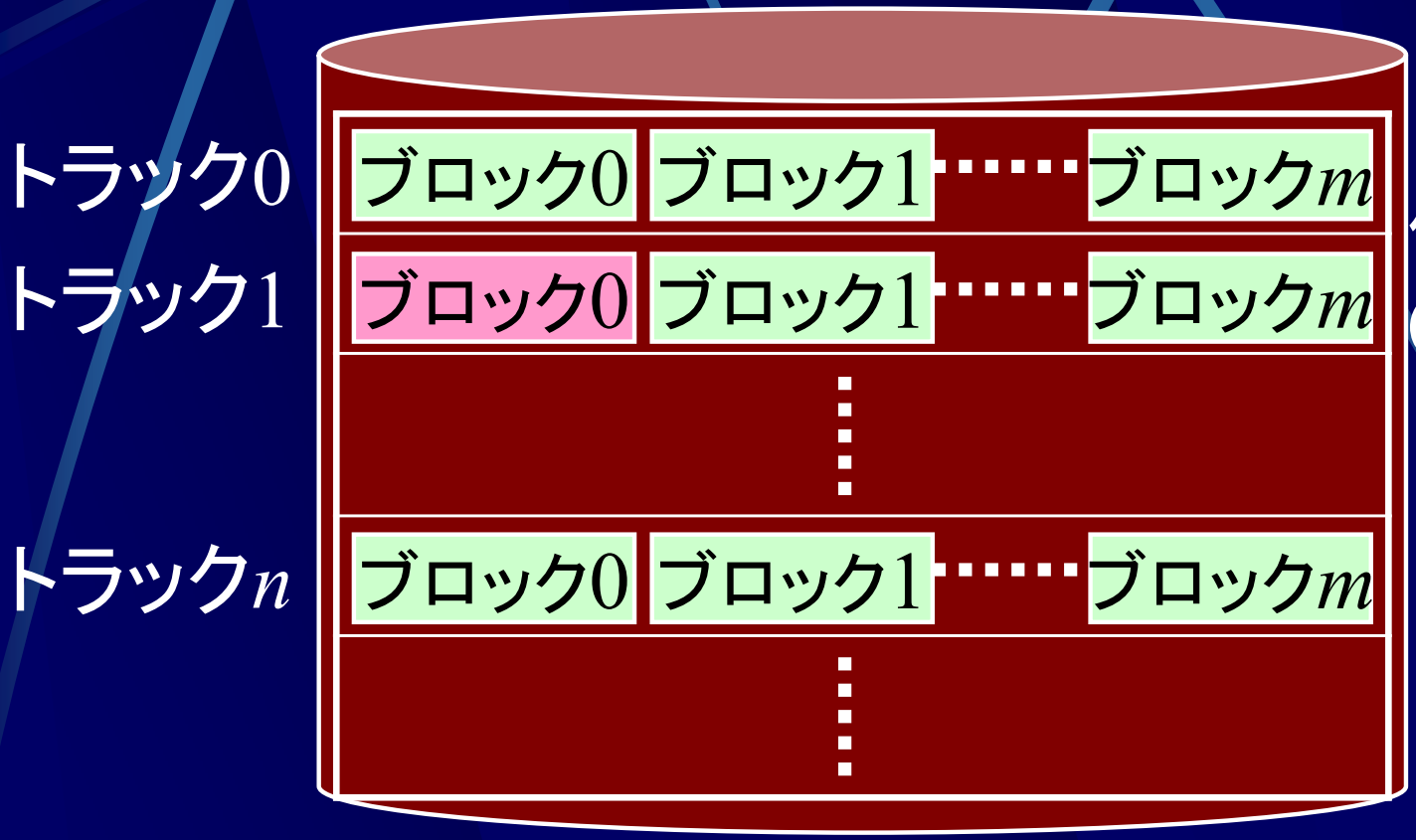
アドレスだけでは管理しにくい

↓

ブロック単位でアクセス

ブロック番号(block address)

ファイルへのアクセス時はブロック番号を指定する



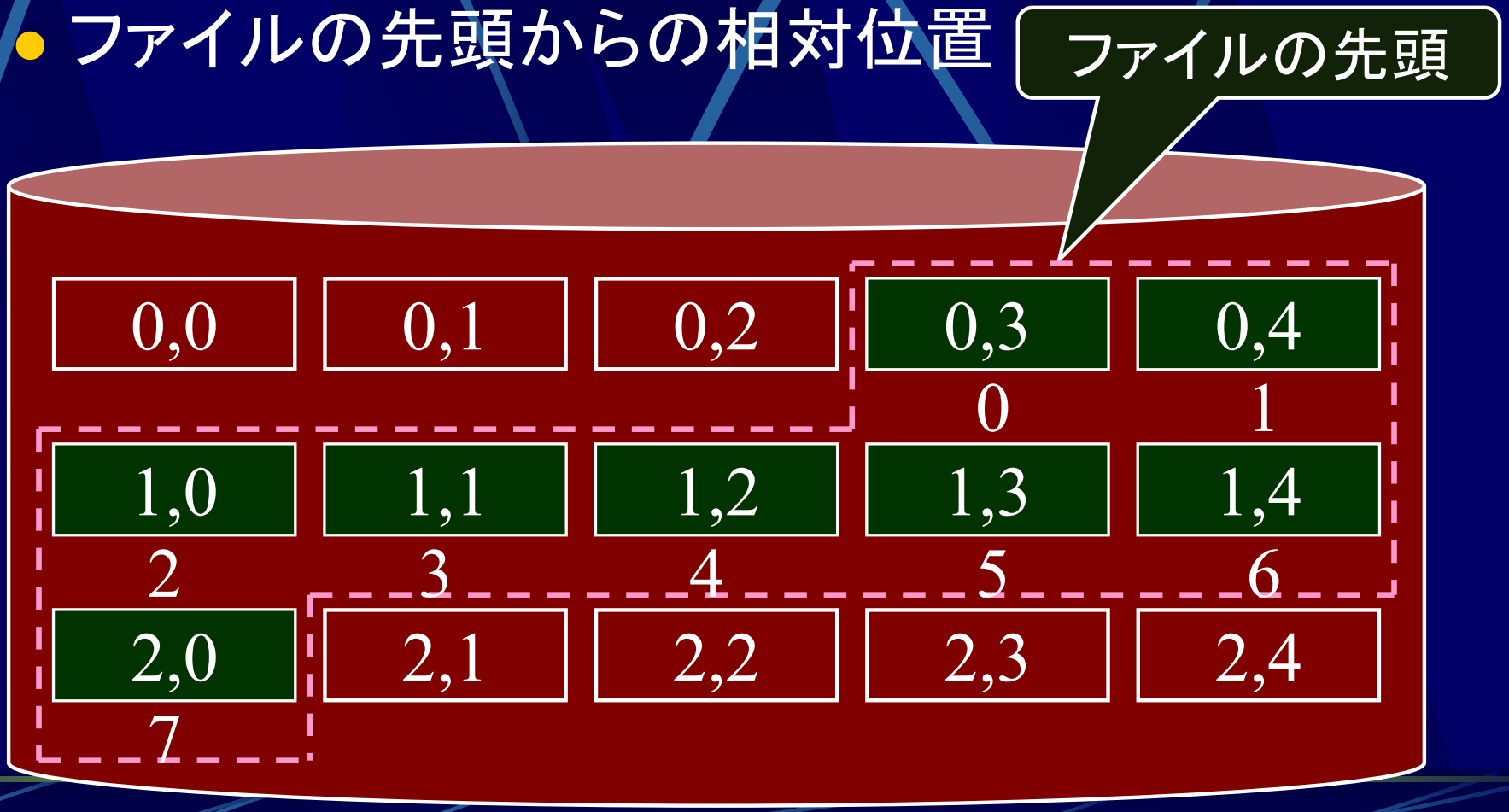
例：ブロック(1, 0)
(トラック, ブロック)

ファイルの先頭からの位置を表す

相対ブロック番号(relative block number)でもアクセス可能

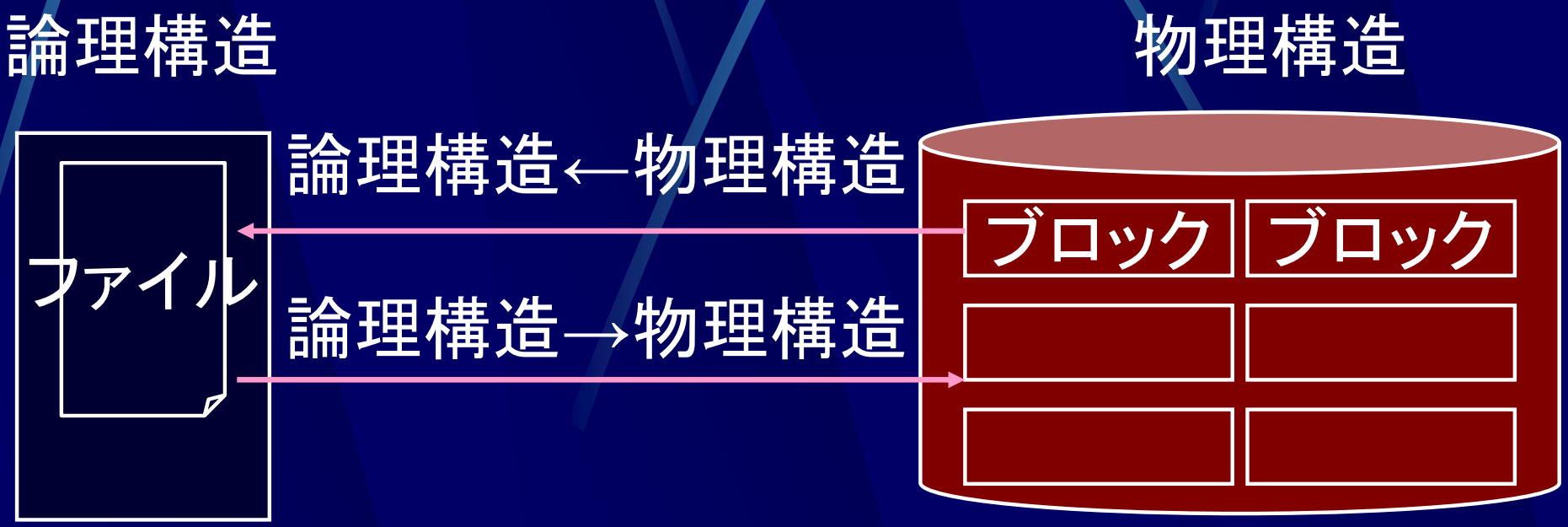
相対ブロック番号

- 相対ブロック番号(relative block number)
  - ファイルの先頭からの相対位置



アクセス法(access method)

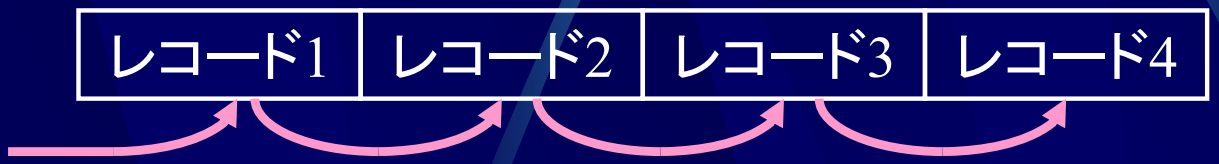
- アクセス法(access method)
  - ファイルの論理構造と物理構造の間の変換



ゆ

アクセス法

- 逐次アクセス(sequential access)
  - ファイルの先頭から順にアクセス
- 直接アクセス(direct access)
  ランダムアクセス(random access)
  - 任意の場所をアクセス可能

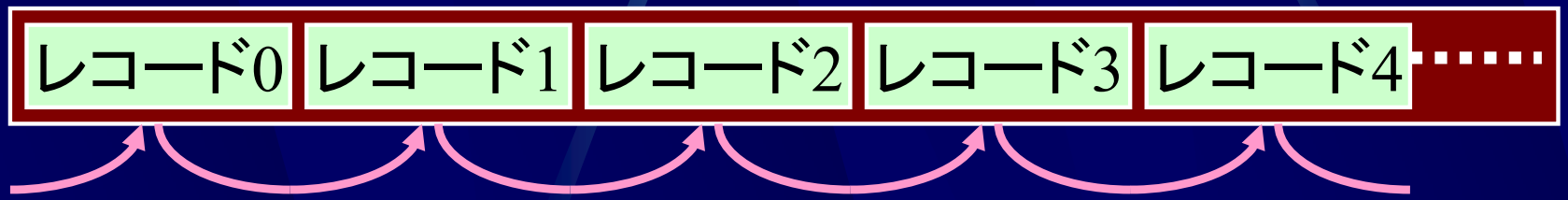


ファイルアクセス

- 逐次アクセスファイル(sequential access file)
  - ファイルの先頭のレコードから順にアクセス
- 直接アクセスファイル(direct access file)
  ランダムアクセスファイル(random access file)
  - 任意のレコードにアクセス可能
- 参照付きファイル(indexed file)
  - 各レコードが参照用のキーを持つ
- 区分編成ファイル(partitioned file)
  - サブファイルから成るファイル

逐次アクセスファイル
(sequential access file)

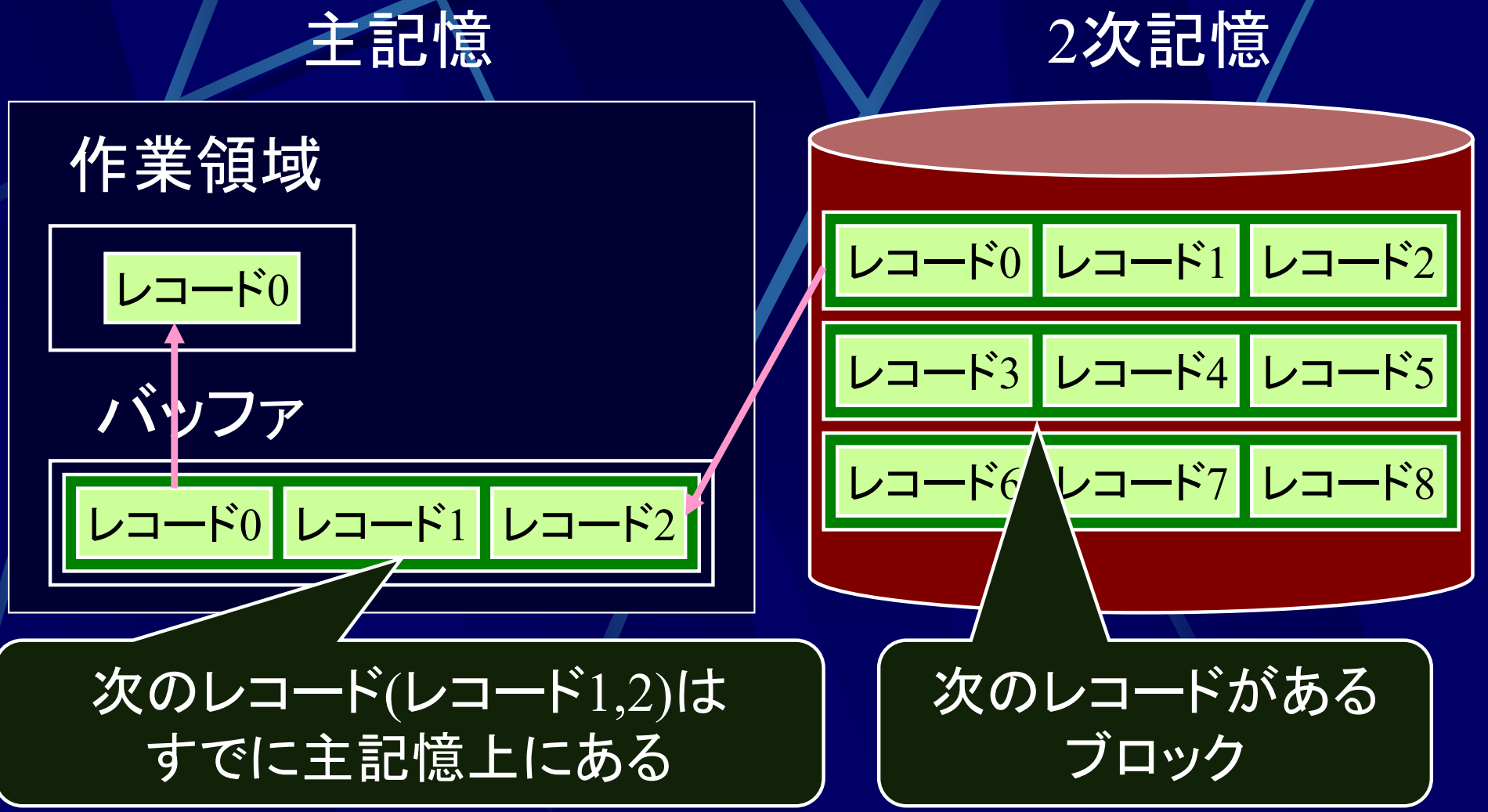
- 逐次アクセスファイル(sequential access file)
  - 読み書きは先頭から順番にしかできない
  - スキップしたり順番を外れたアクセスは不可
  - 巻き戻しはできる場合もある



逐次アクセス(sequential access)

命令	操作内容
READ	現在位置のレコードを読み次に移動
WRITE	現在位置のレコードに書き次に移動
SKIP	現在位置を前後のレコードに移動
REWIND	現在位置を先頭のレコードに移動

逐次アクセス



レコード順なら高速に
アクセス可能

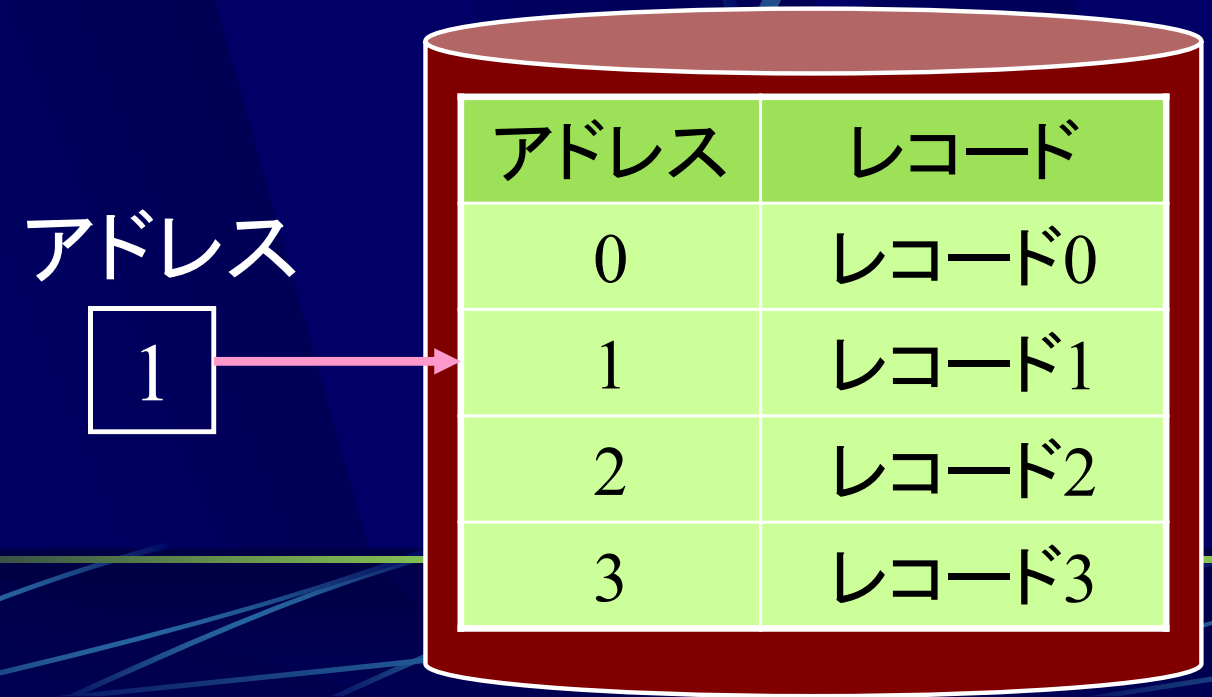
予め次のブロックを読み込める
(プリフェッチ)

逐次アクセスの利点と欠点

- 逐次アクセスの利点
  - 連続したブロックを効率良く読み書き可能
- 逐次アクセスの欠点
  - 途中のブロックの読み書き, 挿入, 削除が困難
  - ファイルサイズが大きいとアクセス時間増大

直接アクセスファイル
(direct access file)

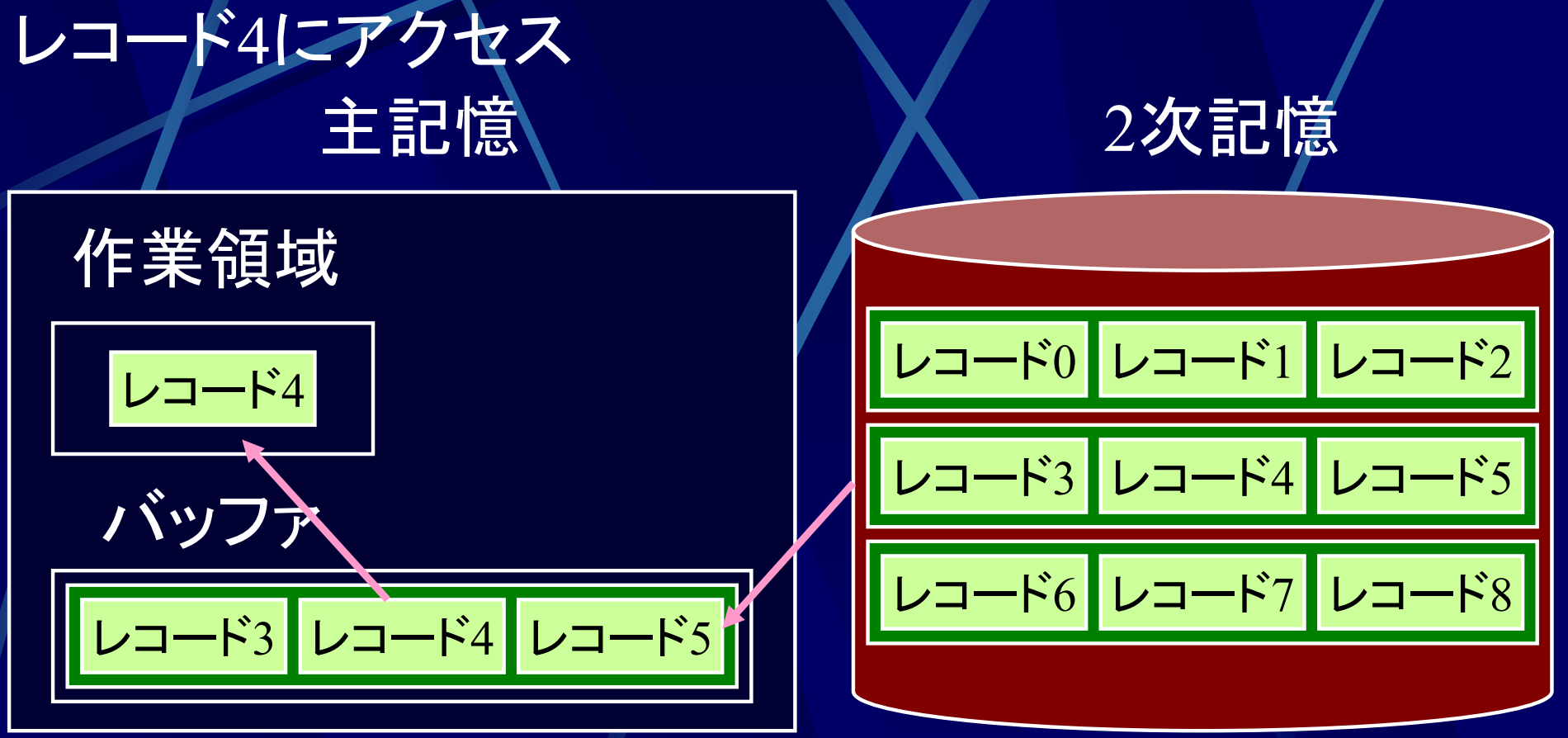
- 直接アクセスファイル(direct access file)
  ランダムアクセスファイル(random access file)
  - 任意の位置を読み書き可能



直接アクセス(direct access)

命令	操作内容
READ	現在位置のレコードを読み次に移動
WRITE	現在位置のレコードに書き次に移動
SEEK *l*	現在位置を *l* 番地のレコード移動
READ *n*	*n* 番地のレコードを読み *n*+1 番地に移動
WRITE *n*	*n* 番地のレコードに書き *n*+1 番地に移動

直接アクセス



次に使用するレコード/ブロックが予測しにくい

プリフェッチが難しい

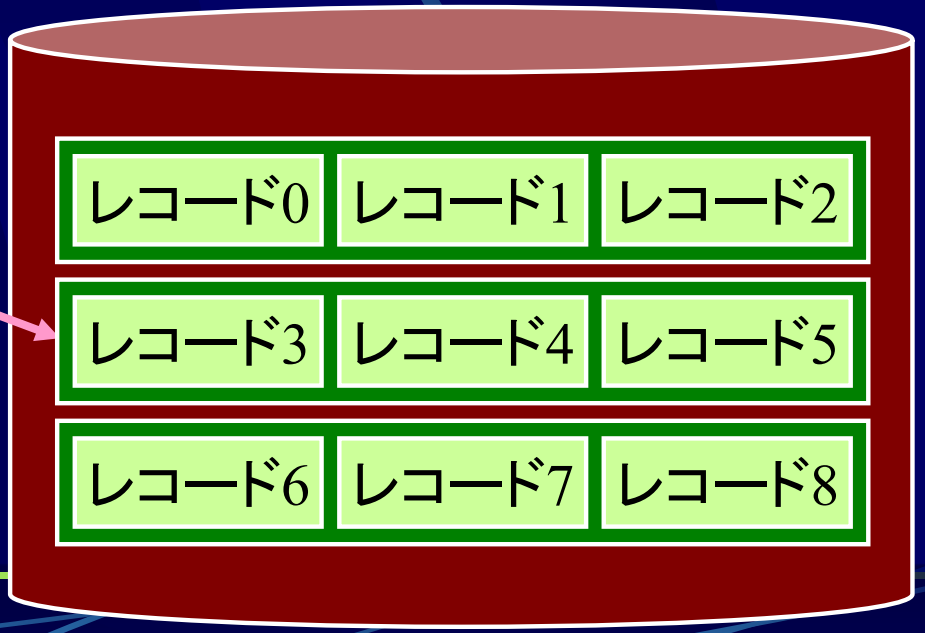
直接アクセスの利点と欠点

- 直接アクセスの利点
  - ファイルの必要な部分だけアクセス可能
  - ファイルサイズが大きくてもアクセス時間は変わらない
- 直接アクセスの欠点
  - アドレスの管理が必要
  - 2次記憶からの読み込み時間が長い

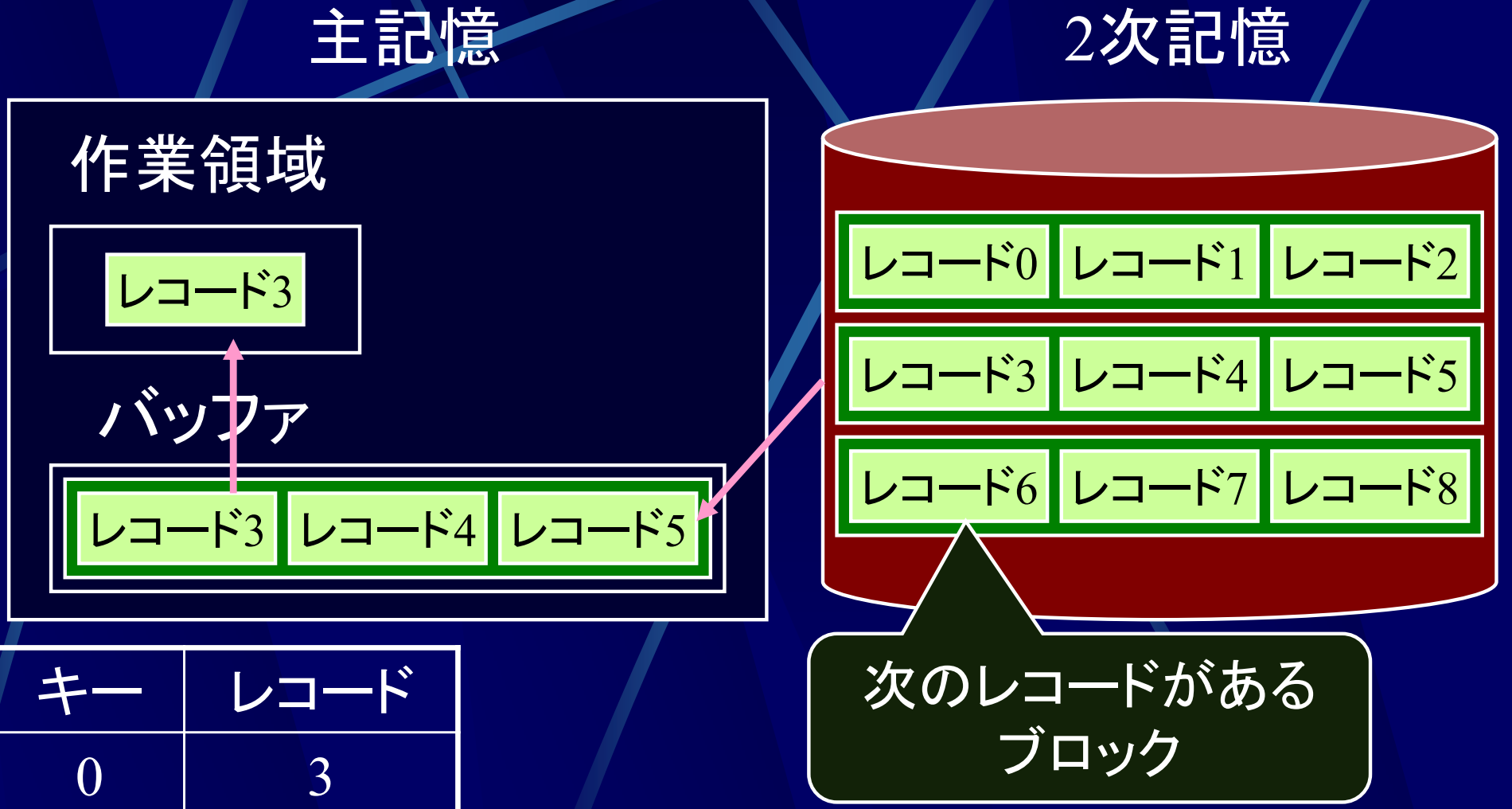
索引付きファイル
(indexed file)

- 索引付きファイル(indexed file)
  - 索引(index)と論理的な順序を表すキー(key)の付けられたファイルに対するアクセス

キー	レコード
0	3
1	6
2	0
3	2



索引付アクセス



キー	レコード
0	3
1	6
2	0
3	2

キーの順でアクセスするなら
次に使うレコードが予測できる

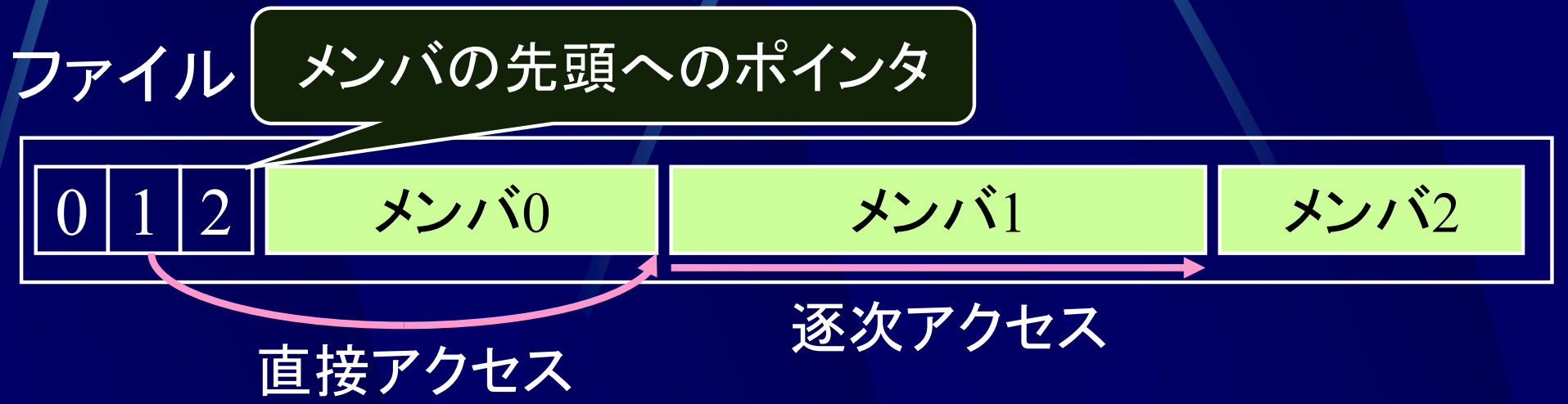
⇒プリフェッチ可能

索引付きファイルの利点と欠点

- 索引付きファイルの利点
  - キーを介して直接アクセス可能
  - キーの順序で逐次アクセス可能
    - 逐次アクセスはプリフェッチにより高速にアクセス可能
- 索引付きファイルの欠点
  - 索引の維持にコストが掛かる
  - 索引の効率の良い検索ができねばならない

区分編成ファイル (partitioned file)

- 区分編成ファイル(partitioned file)
  - サブファイル(メンバ,member)で構成されたファイル
    - 各メンバの先頭へは直接アクセス可能
    - 各メンバの中身へは逐次アクセス



区分編成ファイルの利点と欠点

- 区分編成ファイルの利点
  - それぞれは逐次処理を行うファイルが大量にあるときに効率良く処理できる
- 区分編成ファイルの欠点
  - メンバの削除・挿入が難しい
- 区分編成ファイルの用途
  - プログラムライブラリ, マイクロライブラリ等

まとめ：ファイルシステム

- ファイルシステム
  - ファイル操作の統一的な方法を提供
- ファイル
  - 永続性
  - 大きさは任意
  - プロセス間共有可能
- ファイル制御ブロック (FCB)
  - ファイル情報を保持
  - カーネル領域に作られる

まとめ：ファイル構造

- ファイル構造
  - (論理)レコード
    - 関連した項目をまとめたもの
  - ブロック (物理レコード)
    - 2次記憶上での記憶単位
    - レコードの集合体

ブロック

まとめ:レコード形式

	ブロックレコード (1ブロック複数レコード)	非ブロックレコード (1ブロック1レコード)
固定長レコード (レコード長固定, n*レコード長＝ブロック長)	固定長 ブロックレコード 	固定長 非ブロックレコード 
可変長レコード (レコード長可変, レコード長≦ブロック長)	可変長 ブロックレコード 	可変長 非ブロックレコード 
スパンドレコード (レコード長可変, レコード長>ブロック長)	スパンド ブロックレコード 	スパンド 非ブロックレコード 

まとめ：ファイル転送

- ファイル転送
  - ブロッキング
    - 書き出し時　：　レコード⇒ブロック
  - デブロッキング
    - 読み込み時　：　ブロック⇒レコード
- バッファリング
  - CPU処理と二次記憶からの転送をオーバラップ
- ダブルバッファリング
  - 転送処理同士をオーバラップ

まとめ：ファイルアクセス

- 逐次アクセスファイル(sequential access file)
  - ファイルの先頭のレコードから順にアクセス
- 直接アクセスファイル(direct access file)
  ランダムアクセスファイル(random access file)
  - 任意のレコードにアクセス可能
- 参照付きファイル(indexed file)
  - 各レコードが参照用のキーを持つ
- 区分編成ファイル(partitioned file)
  - メンバ(サブファイル)から成るファイル

期末テスト

- 試験日：1月23日(月)
- 試験時間：60分
- 試験範囲：第1～14回
- 配点：70点満点
- GoogleClassroom 上で行う