オペレーティングシステム

第13回

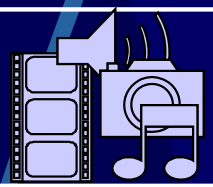
ファイルシステム(2)

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
  - プロセス間で共有可能
  - 大きさに制限無し

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)によって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

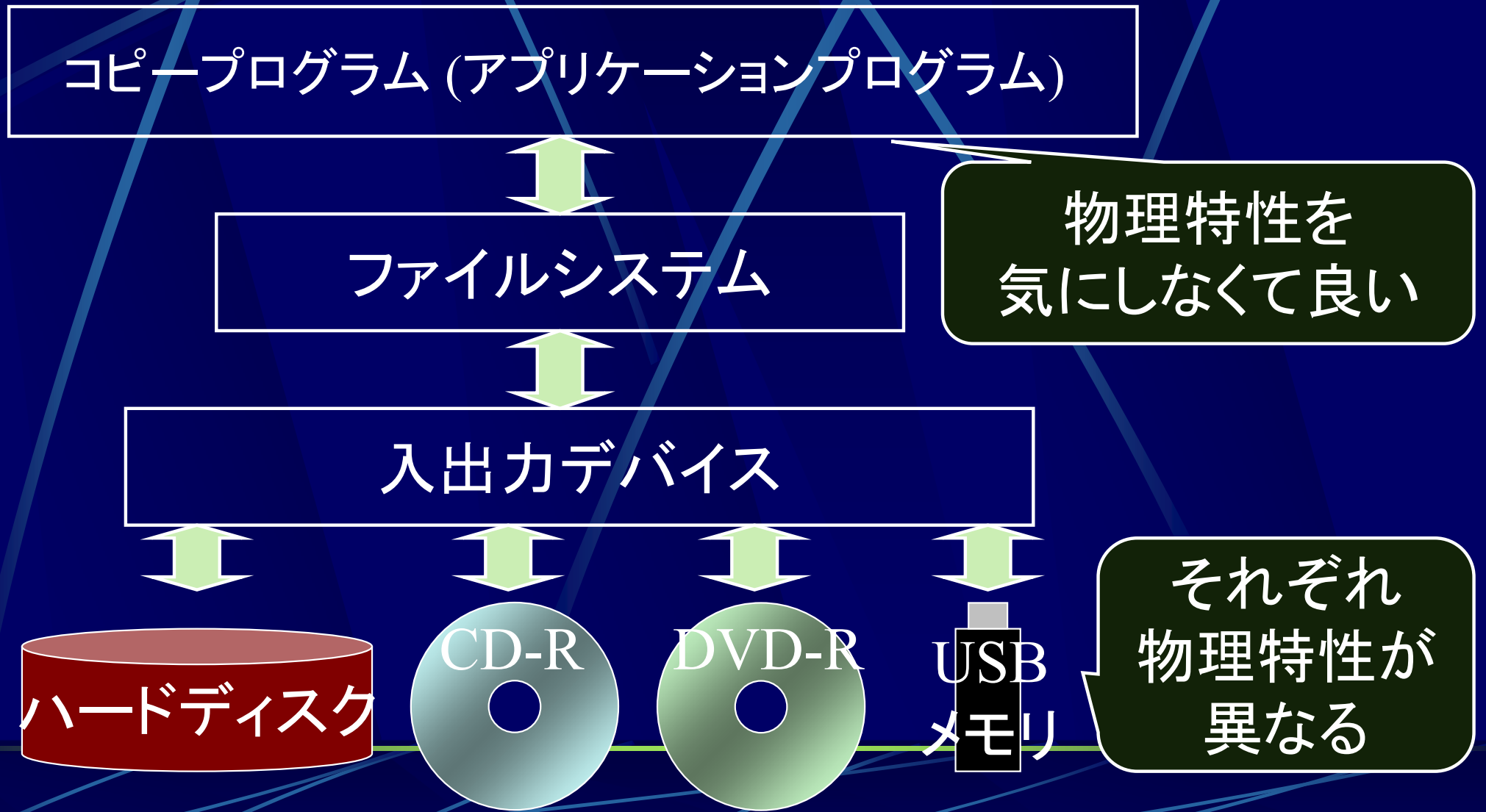
- ハードウェアの性能を引き出す
- データの信頼性を保証する
- ハードウェアを使い易くする
  - ファイルはハードウェアに依存しない

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 indepenence)

ファイルシステムの目的

例：データのコピー



ファイルの型

- 通常ファイル(regular file)
  - プログラムやデータを格納する
  - 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

ファイルの管理

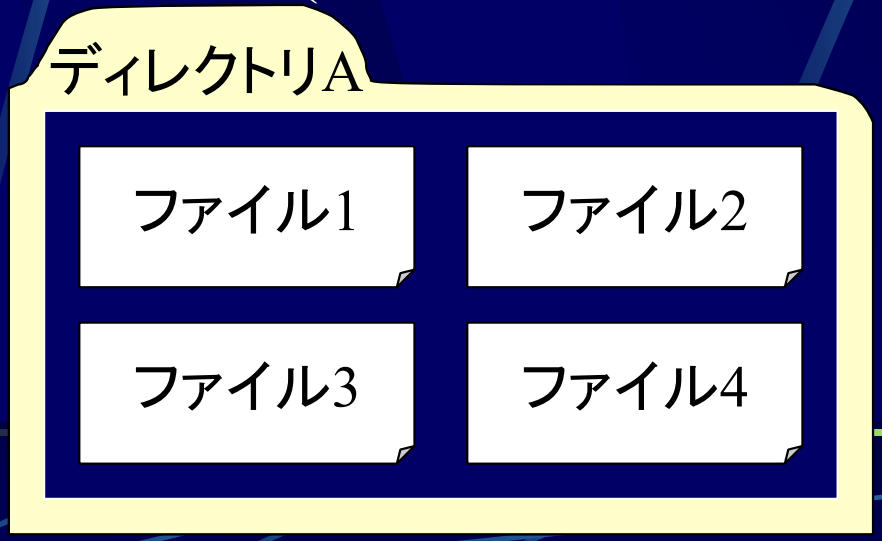
- ファイルの管理
  - ファイルが多い
    - 用途別にファイルを分類したい
  - 複数のユーザが使用する
    - ユーザ別にファイルを分類したい

ファイルを分類・格納する入れ物を用意する

ディレクトリ(directory)

ディレクトリ (directory)

- ディレクトリ (directory)
  - ファイル格納するための入れ物
    (仮想レベル : ユーザにとってのディレクトリ)
  - ファイル情報を管理・保持するためのファイル
    (物理レベル : 計算機にとってのディレクトリ)



ディレクトリA

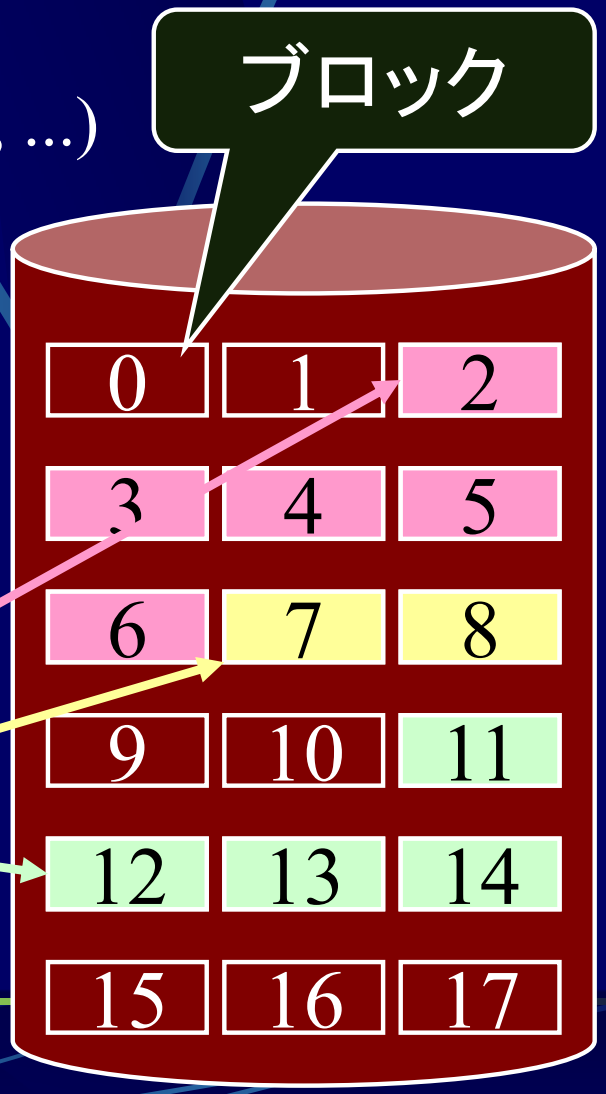
ファイル1 : 500KB : rw-
ファイル2 : 150KB : r--
ファイル3 : 300KB : rwx
ファイル4 : 100KB : r--

ディレクトリ

- ディレクトリ
  - ファイル情報(サイズ, 属性, 所有者, ...)
  - ブロックへのポインタ

ディレクトリ(連続割り付けの場合)

ファイル名	サイズ	属性	ブロック
ファイル1	50KB	rwx	2～6
ファイル2	15KB	r--	7, 8
ファイル3	35KB	rw-	11～14



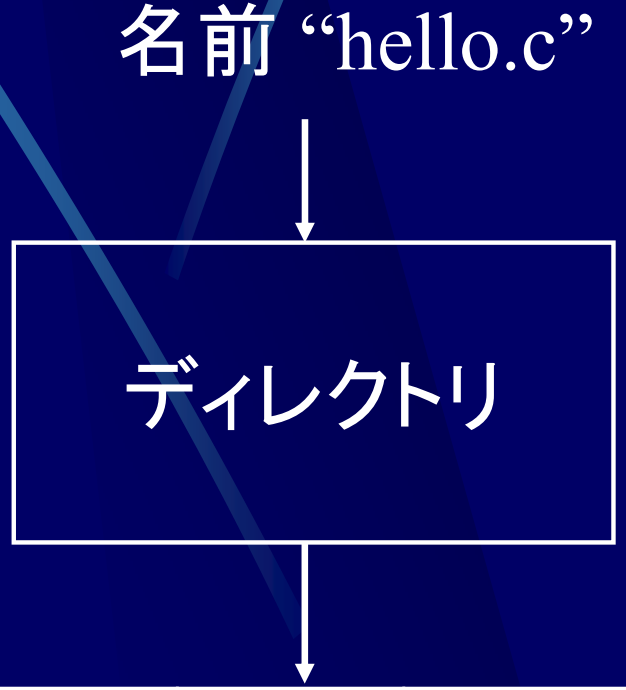
ディレクトリに対する操作

- ディレクトリに対する操作
  - 探索
    - 指定したファイルの情報を得る
  - エントリの追加
    - ファイル追加時にファイル情報を追加する
  - エントリの削除
    - ファイル削除時にファイル情報を削除する
  - 一覧表示
    - ディレクトリが管理するファイル一覧を表示する

ディレクトリの探索

- ディレクトリの探索
  - 線形リスト (lenear list)
  - 2分木 (binary tree)
  - ハッシュテーブル (hash table)

名前をキーとして探索

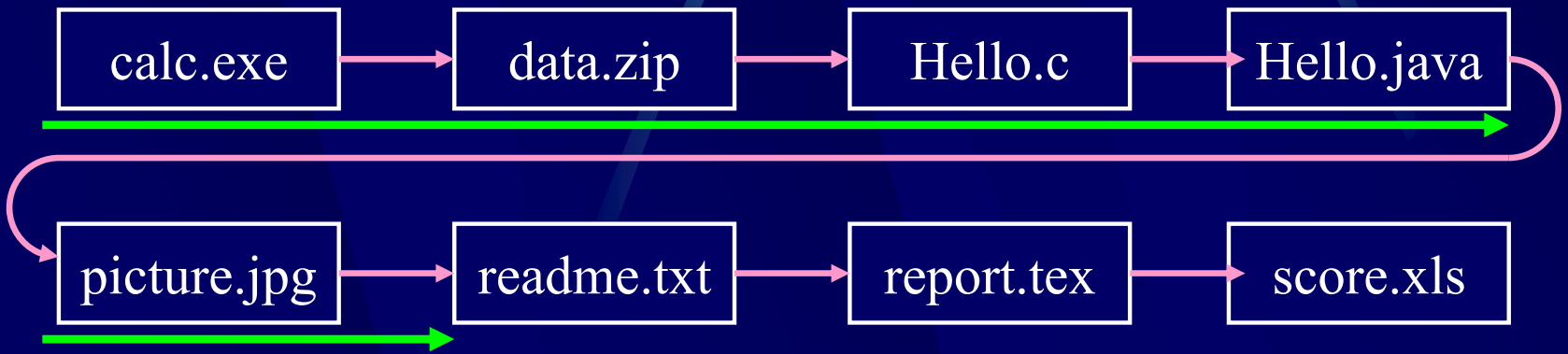


名前	サイズ	属性	ブロック
hello.c	20KB	rw-	15

ディレクトリの探索

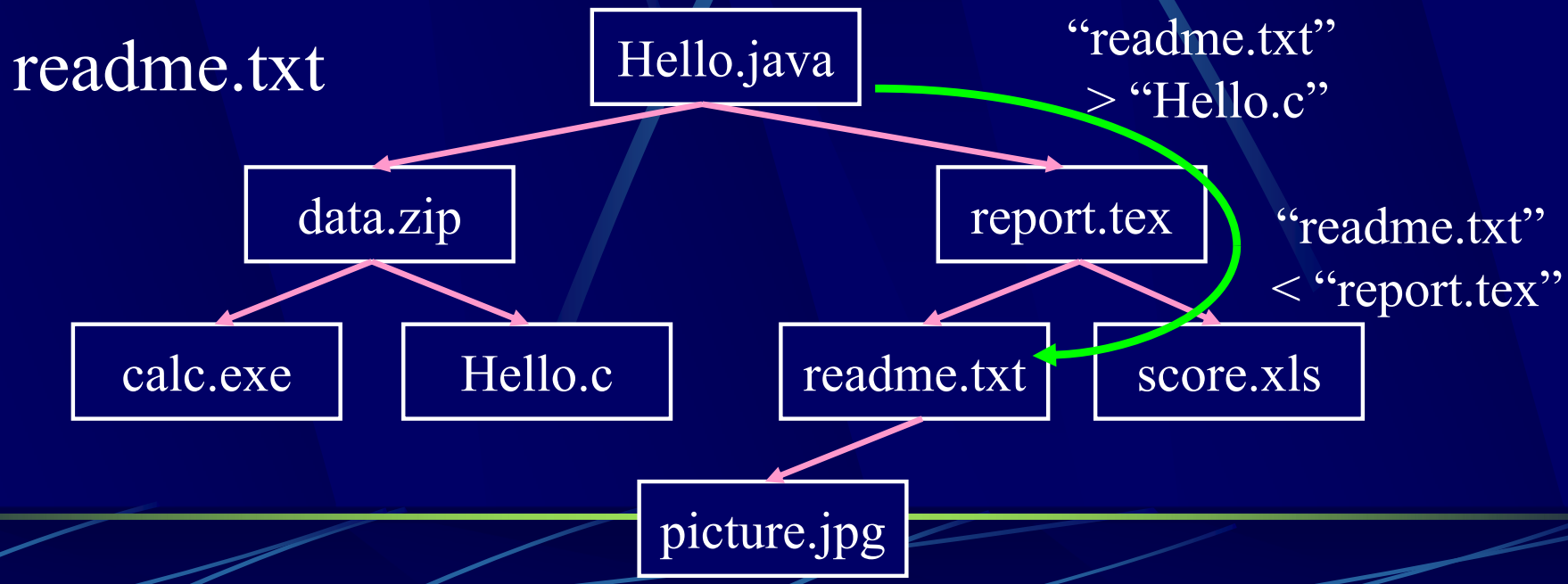
- 線形リスト (linear list)
  - 操作が単純
  - 探索に時間がかかる

readme.txt



ディレクトリの探索

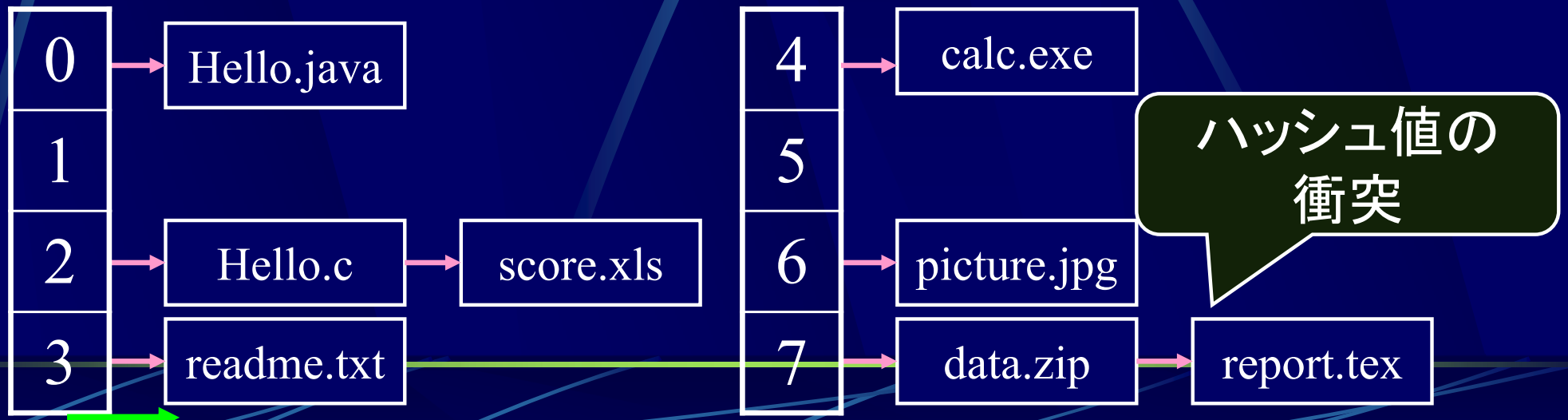
- 2分木 (binary tree)
  - 探索が速い
  - 木構造の管理が必要



ディレクトリの探索

- ハッシュテーブル (hash table)
  - 探索が速い
  - ハッシュ値の衝突に注意が必要

readme.txt　　hash (“readme.txt”) = 3



各探索方法の長所と短所

探索方法	探索時間	長所	短所
 線形探索	遅い $O(n)$	簡単	遅い
ハッシュ探索	速い $O(1)$	最も速い	要ハッシュ関数
2分探索	速い $O(\log n)$	速い	要ソート

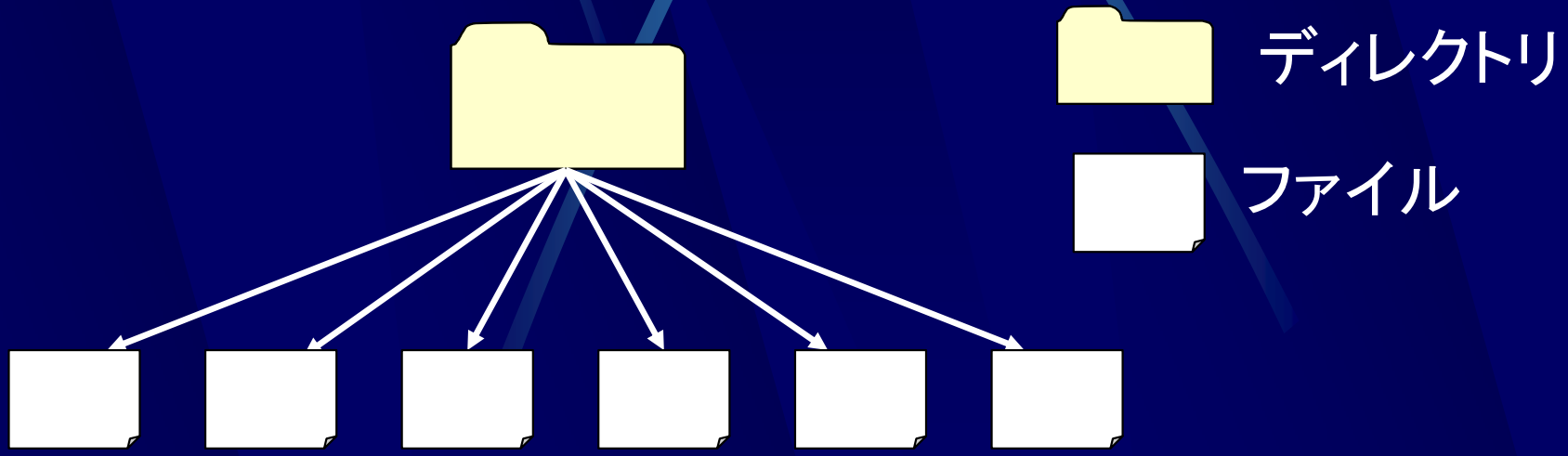
1つのディレクトリに大量のファイルを置かなければ
線形探索でも充分に高速

ディレクトリの階層

- ディレクトリの階層
  - 単階層ディレクトリ (single level directory)
  - 2階層ディレクトリ (two-level directory)
  - 木構造ディレクトリ (tree structured directory)
  - DAG構造ディレクトリ
    (directed acycle graph structured directory)

単階層ディレクトリ (single level directory)

- 単階層ディレクトリ
  - 全てのファイルを1つのディレクトリに置く



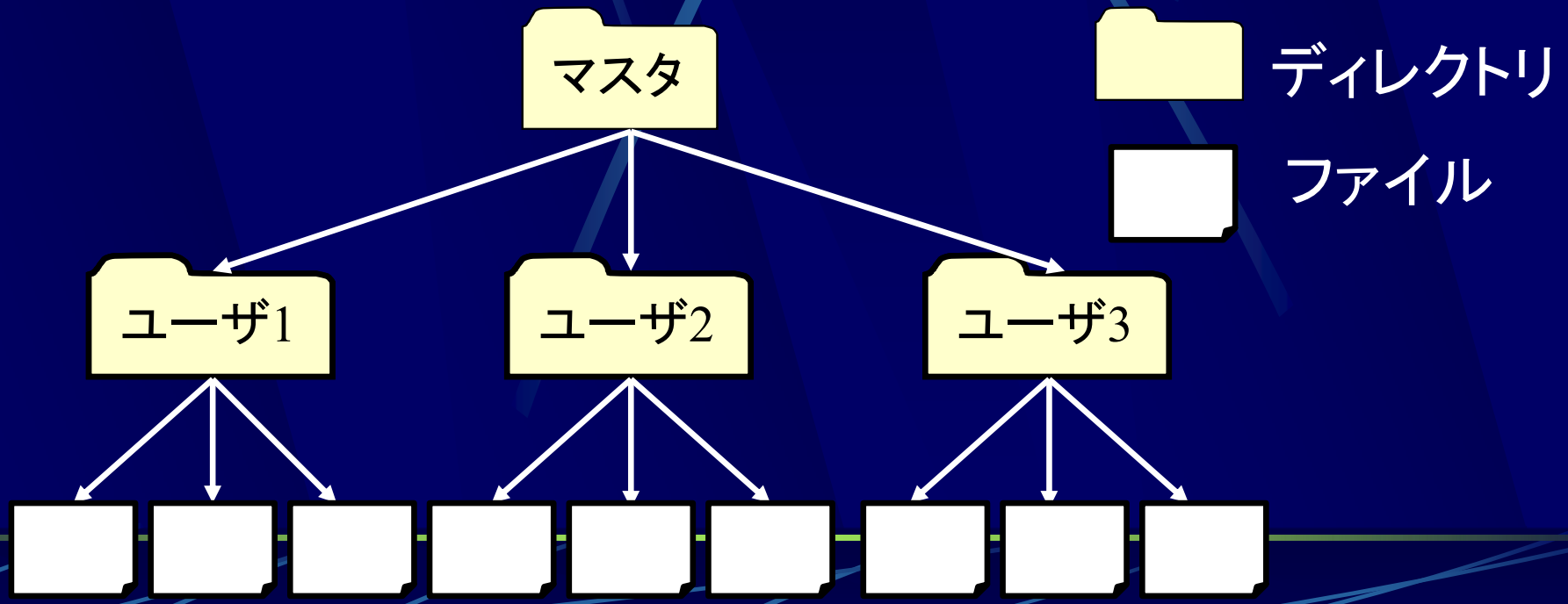
単階層ディレクトリの利点と欠点

- 利点
  - 構造が単純
  - アクセスが高速
- 欠点
  - ファイル数が増えると管理しにくくなる
  - 異なるファイルに同一のファイル名は不可
    - 他のユーザと同じファイル名を使えない
      - ユーザ間でファイル名の調整が必要

複数のユーザがいる場合には不向き

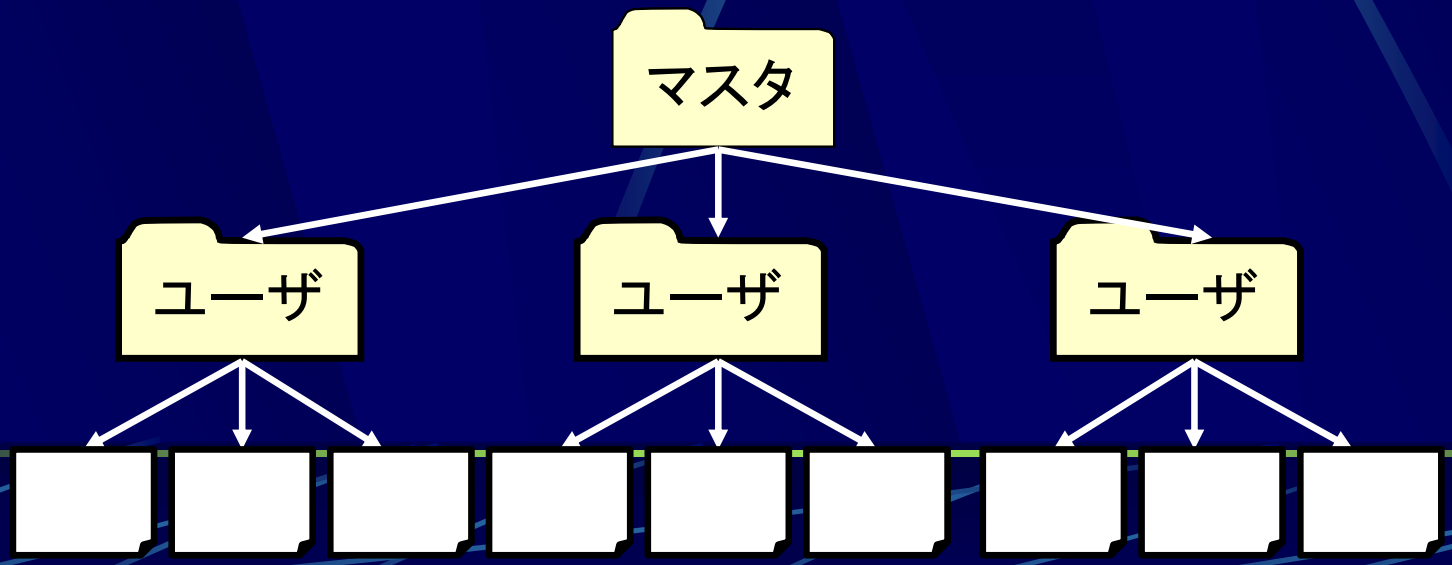
2階層ディレクトリ (two-level directory)

- 2階層ディレクトリ
  - ユーザごとに独立したディレクトリを作成，各ユーザディレクトリの下にファイル置く



2階層ディレクトリ

- ユーザファイルディレクトリ(user file directory)
  - 各ユーザが所有するファイル情報を保持
  - ログイン時は各自のユーザファイルディレクトリに
- マスタファイルディレクトリ(master file directory)
  - ユーザファイルディレクトリ情報を保持

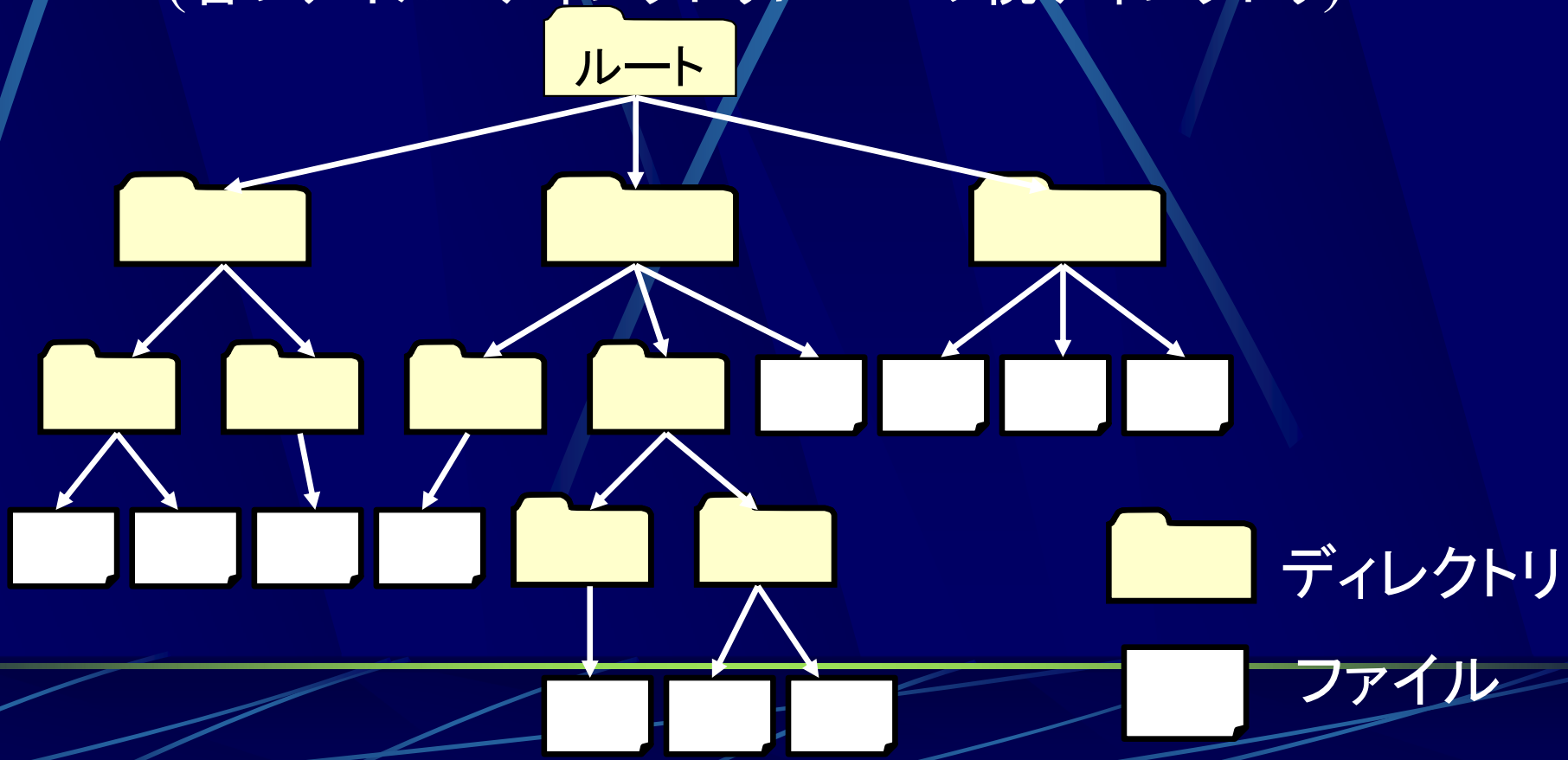


2階層ディレクトリの利点と欠点

- 利点
  - ユーザが違えば同一ファイル名が可能
    - 他のユーザのファイル名を気にする必要が無い
  - ファイルをユーザごとに管理できる
- 欠点
  - 他のユーザとファイルを共有しにくい
  - 1人のユーザで同一ファイル名は不可

木構造ディレクトリ (tree structed directory)

- 木構造ディレクトリ
  - 根付有向木構造, 入れ子構造
    (各ファイル・ディレクトリに1つの親ディレクトリ)

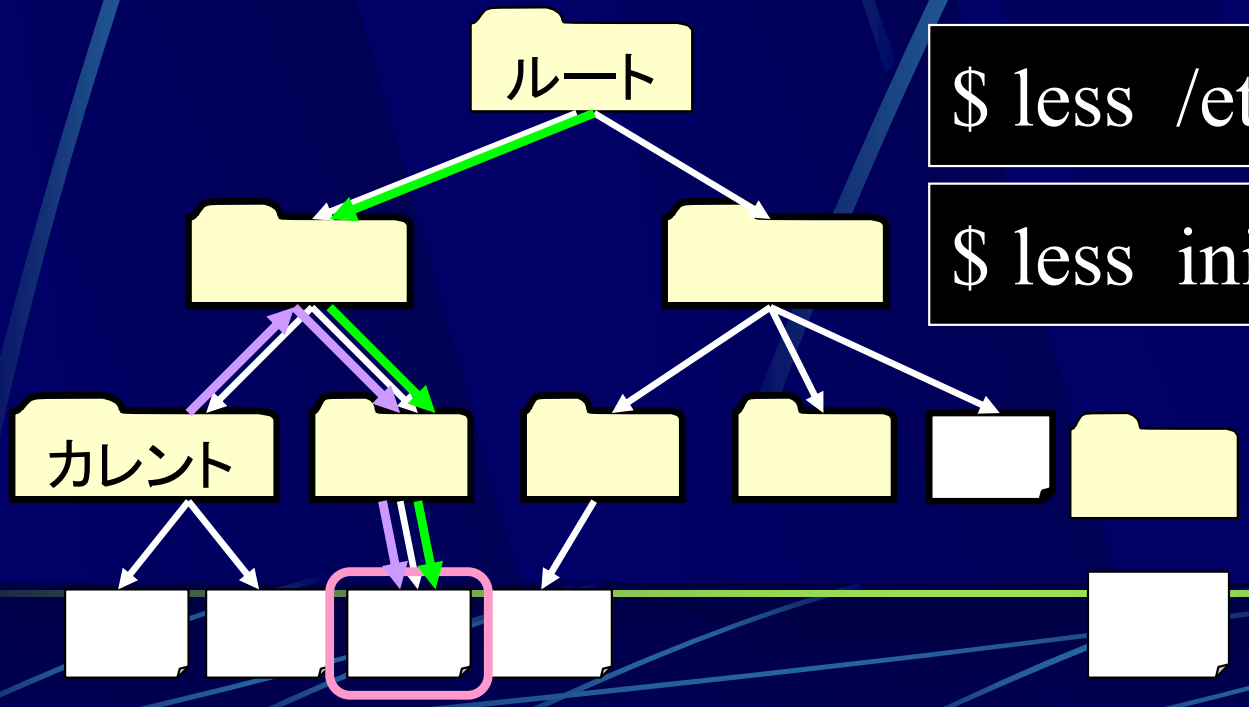


木構造ディレクトリ

- ルートディレクトリ (root directory)
  - 木構造の根に当たるディレクトリ
- カレントディレクトリ (current directory)
  - 現在作業中のディレクトリ
- ユーザディレクトリ (user directory)
  - 各ユーザのログイン時のカレントディレクトリ

ファイルの指定

- 絶対パス名(absolute path name)
  完全パス名(complete path name)
  - ルートディレクトリからの経路を指定
- 相対パス名(relative path name)
  - カレントディレクトリからの経路を指定




```
$ less  /etc/rc.d/init.d/network
```

```
$ less  init.d/network
```

/etc/rc.d にいる場合

ディレクトリ

ファイル

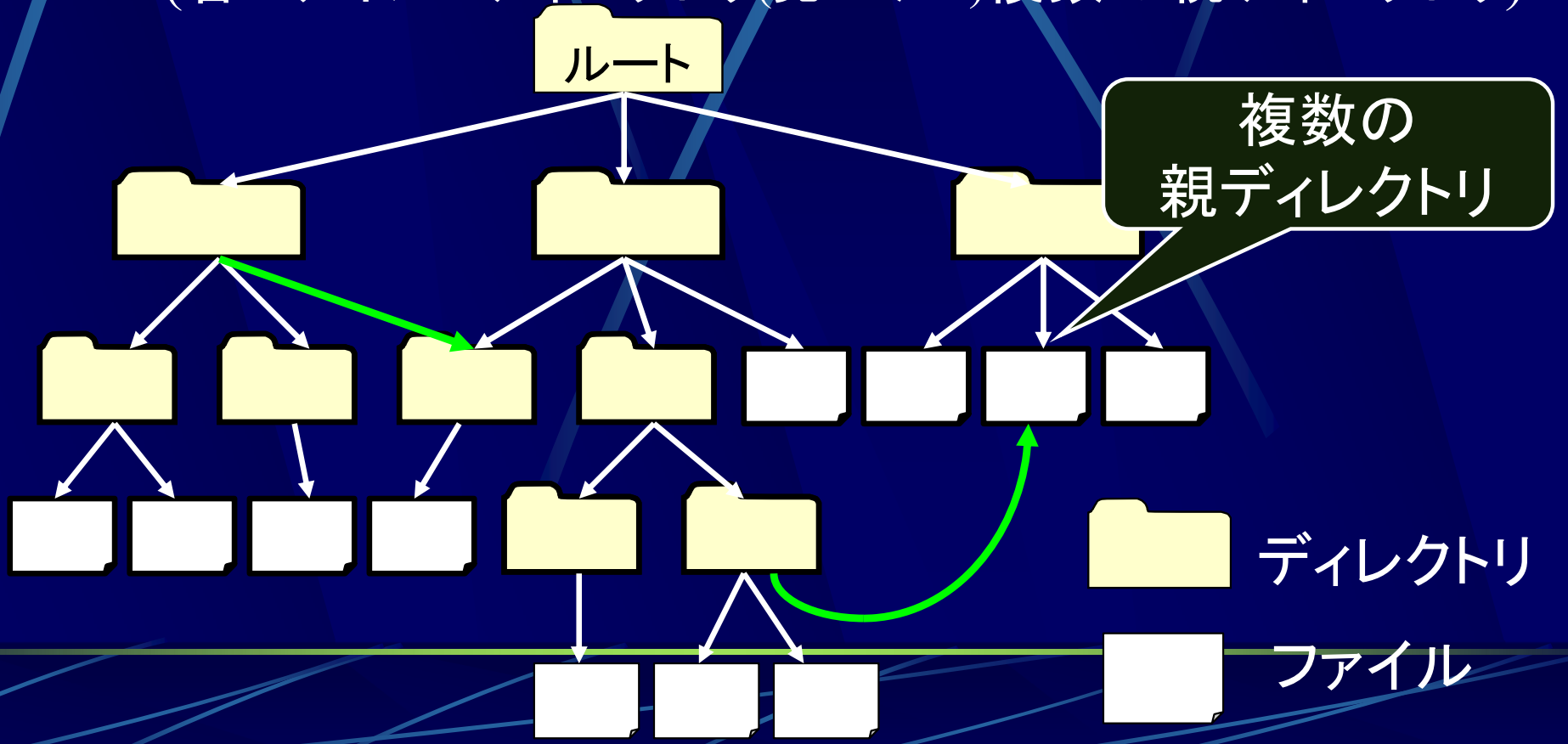
木構造ディレクトリの利点と欠点

- 利点
  - ファイルを用途ごとに分けられる
  - ディレクトリが異なれば同一ファイル名が可能
  - ファイルをユーザごとに管理できる
  - 他のユーザとファイル共有が可能
- 欠点
  - 階層が多いと長いパスの指定必要

```
$ /usr/local/src/javacc-5.0/bin/javacc
```

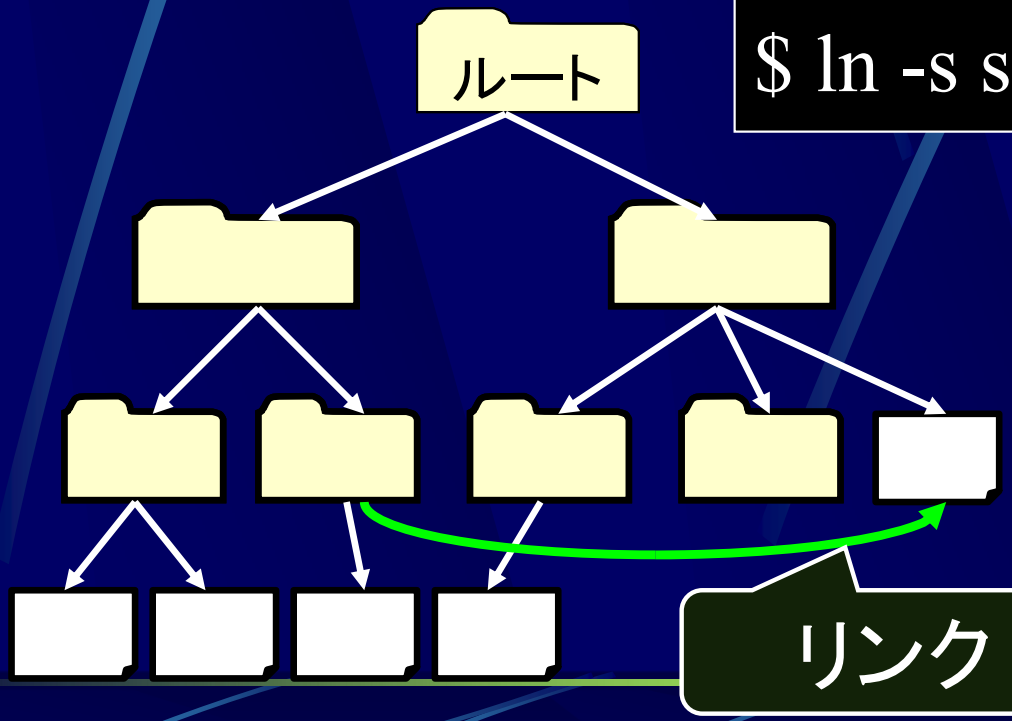
DAG構造ディレクトリ
(directed acycle graph structed directory)

- DAG(directed acycle graph)構造ディレクトリ
  - 根付有向無閉路構造
    (各ファイル・ディレクトリ(見かけ上)複数の親ディレクトリ)



DAG構造ディレクトリ

- DAG構造ディレクトリ
  - 木構造にリンクを張って作成

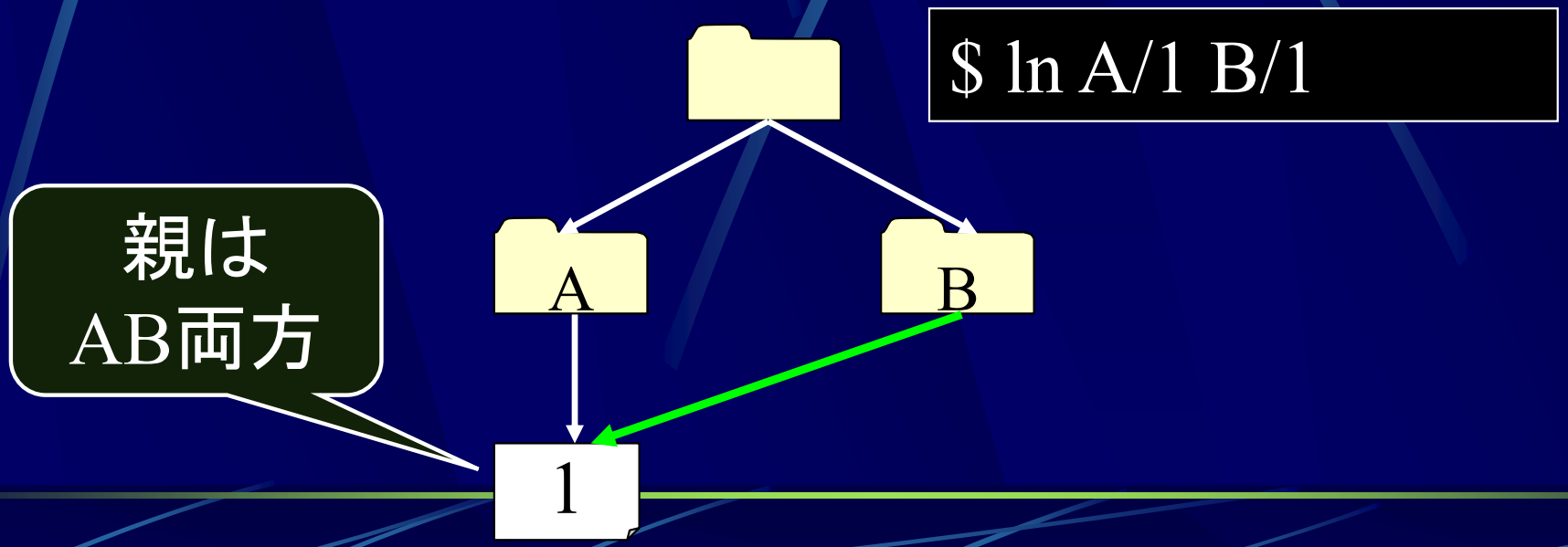



```
$ ln -s src/javacc-5.0/bin/javacc javacc
```

src/javacc-5.0/bin/javacc に
javacc でアクセス可能になる

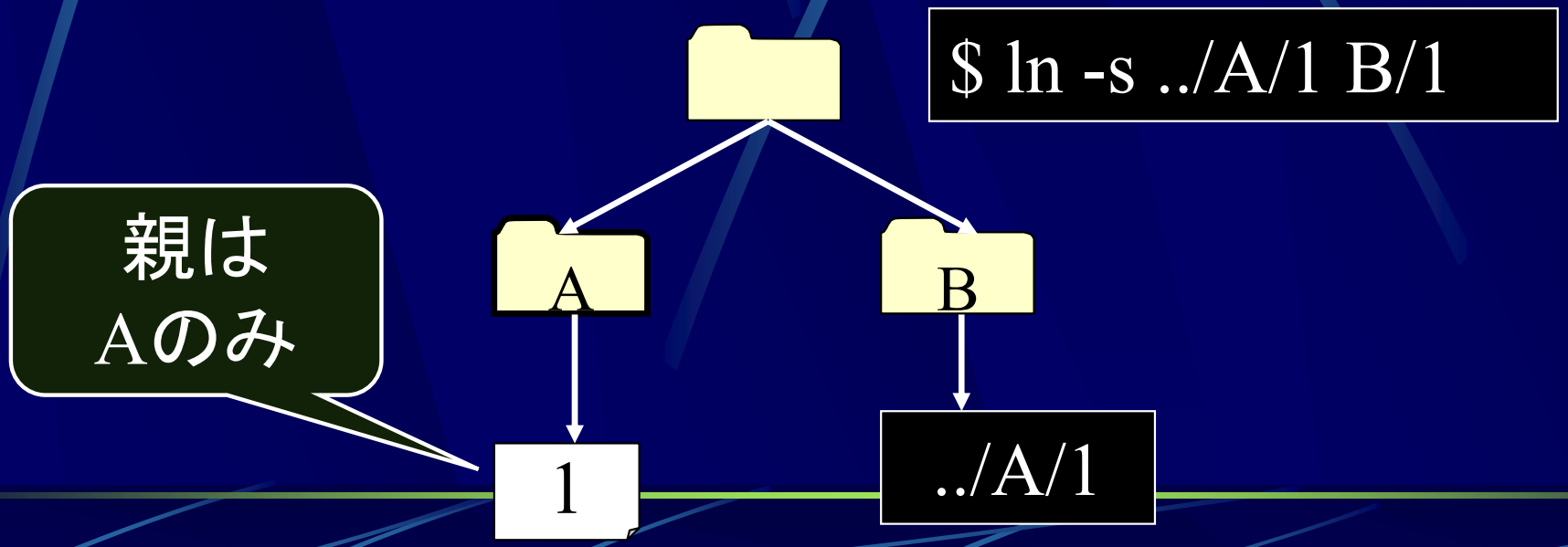
ハードリンク (hard link)

- ハードリンク
  - 対象へのポインタ(通常の親→子と同じ)
  - リンクを張ると子は複数の親を持つ
  - ディレクトリへのハードリンクは不可

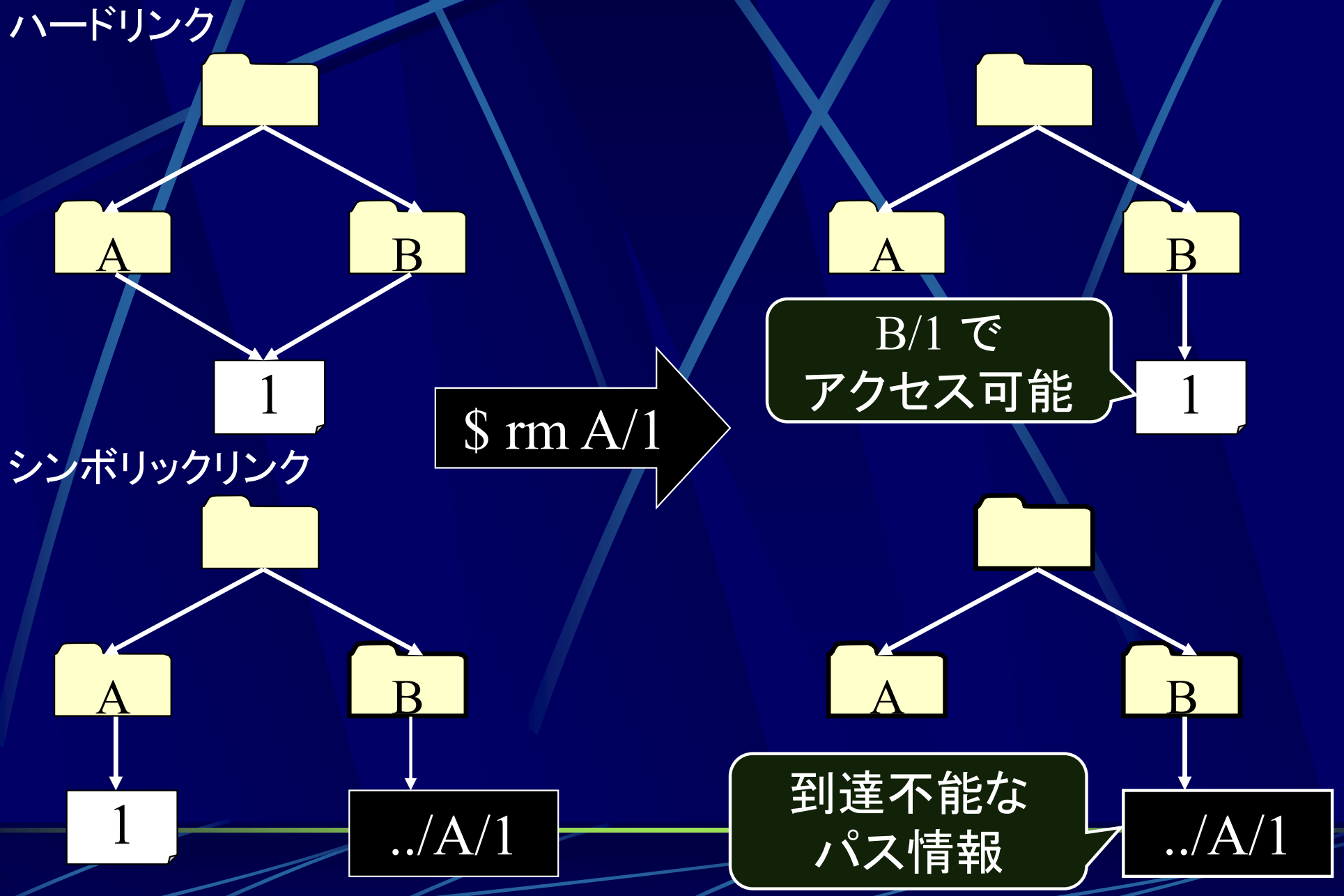


シンボリックリンク (symbolic link)

- シンボリックリンク
  - 対象へのパス情報を持ったファイル
  - リンクを張っても子の親は1つ
  - リンクを逆には辿れない

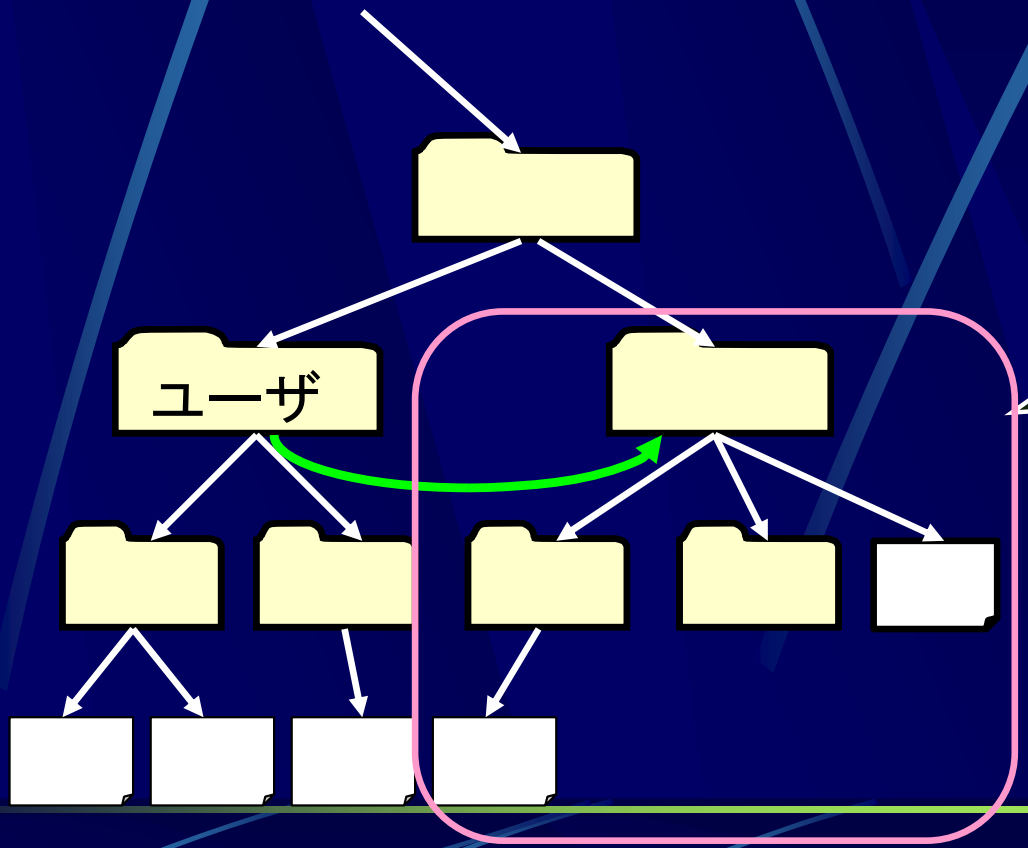


ハードリンクとシンボリックリンク



DAG構造ディレクトリの注意点

- ファイル/ディレクトリの位置が分かりにくい

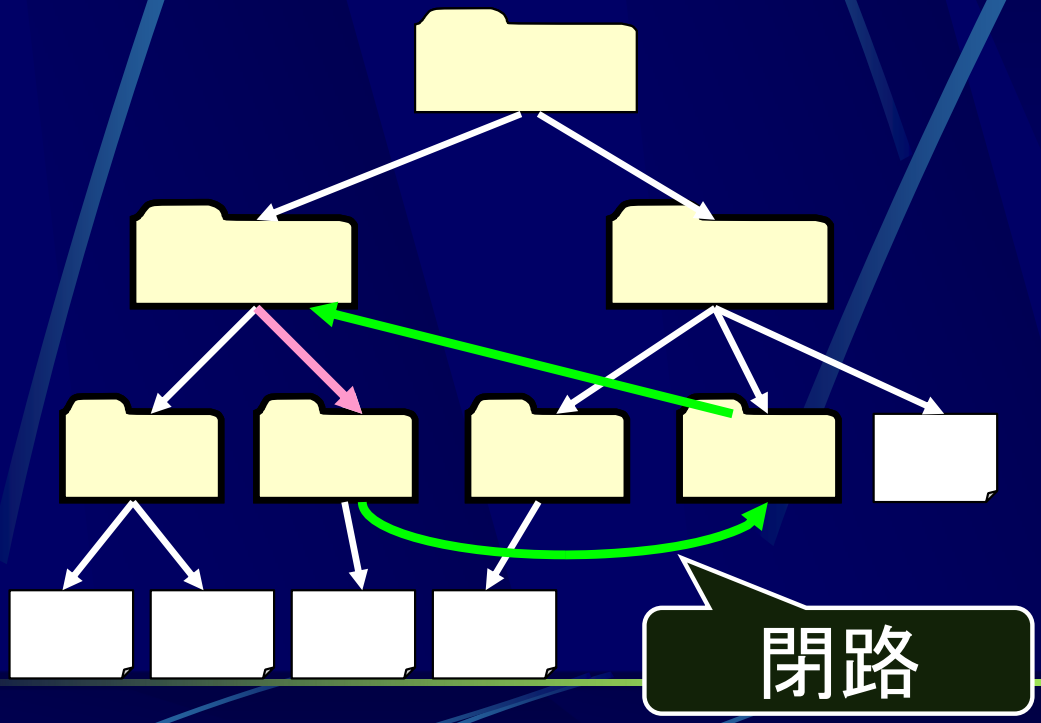


これらは自分の
ユーザディレクトリの
下にあるように見える

自分のファイルと間違って
移動・削除等をする可能性

DAG構造ディレクトリの注意点

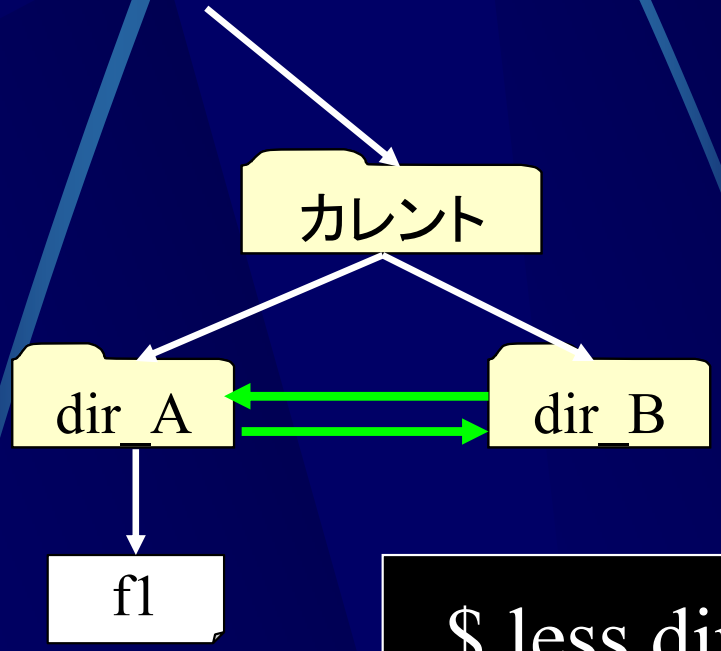
- リンクを張る際に閉路ができてはいけない



閉路を何度も回るパスができてしまう

閉路ができてもエラーにはならないがバグの元

DAG構造ディレクトリの注意点




```
$ ln -s ../dir_A dir_B/dir_A
$ ln -s ../dir_B dir_A/dir_B
```

```
$ less dir_A/dir_B/dir_A/dir_B/dir_A/f1
```

このようなアクセスも可能

ファイルの位置が分かりにくくなる

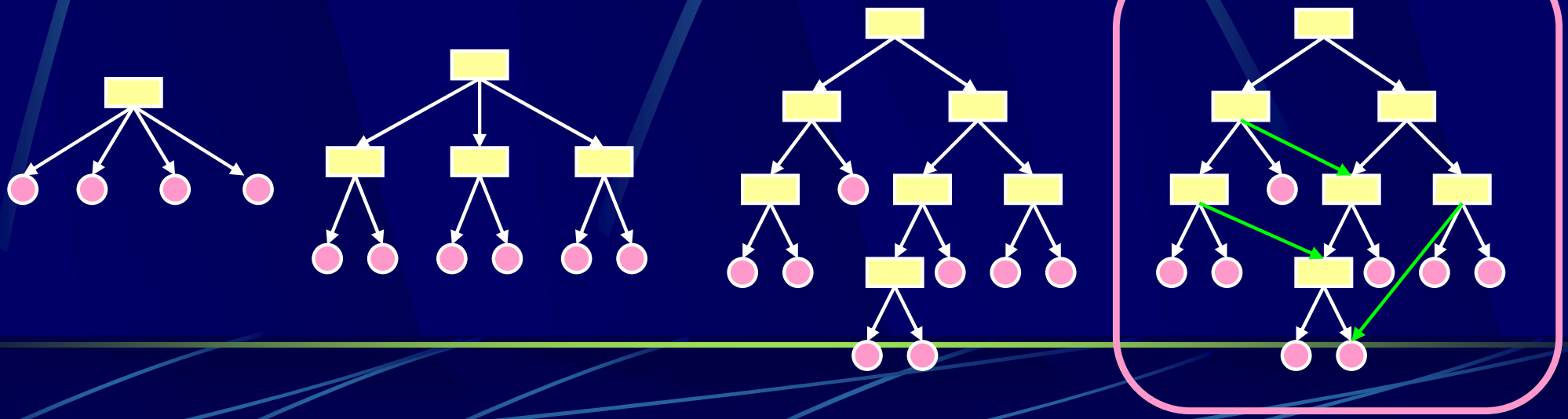
⇒予期せぬバグの原因になる

DAG構造ディレクトリの利点と欠点

- 利点
  - ファイルを用途ごとに分けられる
  - ディレクトリが異なれば同一ファイル名が可能
  - ファイルをユーザごとに管理できる
  - 他のユーザとファイル共有が可能
  - よく使うファイルに短いパスでアクセス可能
- 欠点
  - ファイル・ディレクトリの位置が分かりにくい
  - 閉路ができ易い

ディレクトリの階層

- ディレクトリの階層
  - 単階層ディレクトリ
  - 2階層ディレクトリ
  - 木構造ディレクトリ
  - DAG構造ディレクトリ

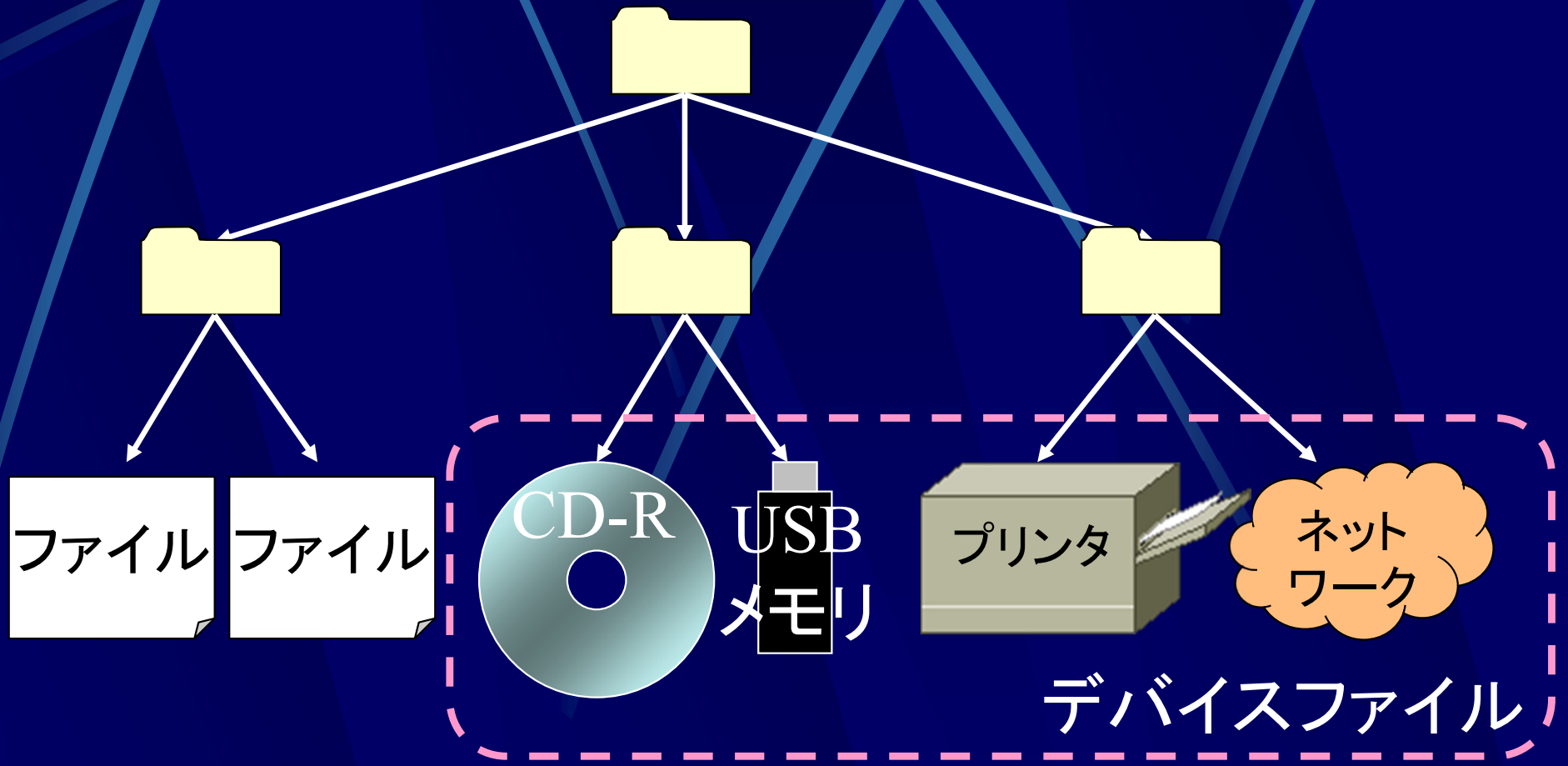


デバイスファイル

- デバイスファイル(device file)
  特殊ファイル(special file)
  - 周辺機器のハードウェアを仮想化したファイル
  - 周辺機器のデバイスドライバのインタフェース
    - 目的:
      周辺機器への入出力を
      ファイルへの読み書きと同様に扱う

- 画面への出力　⇒　ファイル「画面」に書き出す
- キーボードからの入力　⇒　ファイル「キーボード」から読み込む
- プリンタへの出力　⇒　ファイル「プリンタ」に書き出す
- ネットワークからの入力⇒　ファイル「ネットワーク」から読み込む

デバイスファイル



外部デバイスをファイルと同様に扱える

デバイスファイルの種類

- デバイスファイルの種類
  - 文字単位型特殊ファイル(character special file)
    - 端末, プリンタ, ネットワーク等のシリアルデバイス
    - 1文字単位で入出力
  - ブロック単位型特殊ファイル(block special file)
    - ディスク, USBメモリ, SDカード等の記憶装置
    - ブロック単位で入出力

ファイル保護
(file protection)

- ファイル保護(file protection)
  - 物理的な障害, 不当な操作から保護
- アクセス制御(access contorol)
  - ユーザによるファイル操作の種類を制限
- バックアップ(backup)と回復(recovery)
  - 定期的にファイルコピー生成, 破損時に回復

アクセス制御
(access control)

- 主なファイル操作
  - 読み込み
  - 書き出し
  - 実行
  - 追加
  - 削除

これらの操作ごとに (可・不可) を設定する

アクセス制御行列
(access contorol matrix)

ファイル1のアクセス制御行列

ユーザ / 操作	ユーザ1	ユーザ2	ユーザ3	ユーザ4
読み込み	1	1	0	1
書き出し	1	0	0	0
実行	1	1	1	1
追加	1	0	0	0
削除	1	0	0	0

(1：可, 0：不可)

アクセス制御行列

ファイル4	ユーザ1	ユーザ2	ユーザ3	ユーザ4
読み込み	1	1	0	1
書き出し	1	0	0	0
実行	1	1	1	1
追加	1	0	0	0
削除	1	0	0	0

ファイルごとにアクセス制御行列が必要

アクセス制御行列の欠点

- アクセス制御行列の欠点
  - 大きなサイズの行列が必要

    エントリ : (操作数) × (ユーザ数) × (ファイル数)
  - 多くの場合, 疎行列(sparse matrix)になる
    - 書き出し, 削除は通常はファイル所有者のみ可

疎行列 : 大部分が 0 の行列

ユーザクラス(user class)を使用

ユーザクラス(user class)

- ユーザクラス(user class)
  - 所有者(owner)
    - ファイルを作成したユーザ
  - 指定ユーザ(specified user)
    - ファイル所有者が使用を許可したユーザ
  - グループ(group)
    - ファイル所有者と同一グループに属するユーザ
  - 共有(public)
    - 所有者，グループ以外のユーザ

ユーザクラスによるアクセス制御

ファイル1のアクセス制御行列

ユーザ / 操作	所有者	グループ	共有
読み込み	1	1	0
書き出し	1	0	0
実行	1	1	1
追加	1	0	0
削除	1	0	0

エントリを (操作数)×(クラス数)×(ファイル数) に減らせる

ユーザクラスによるアクセス制御

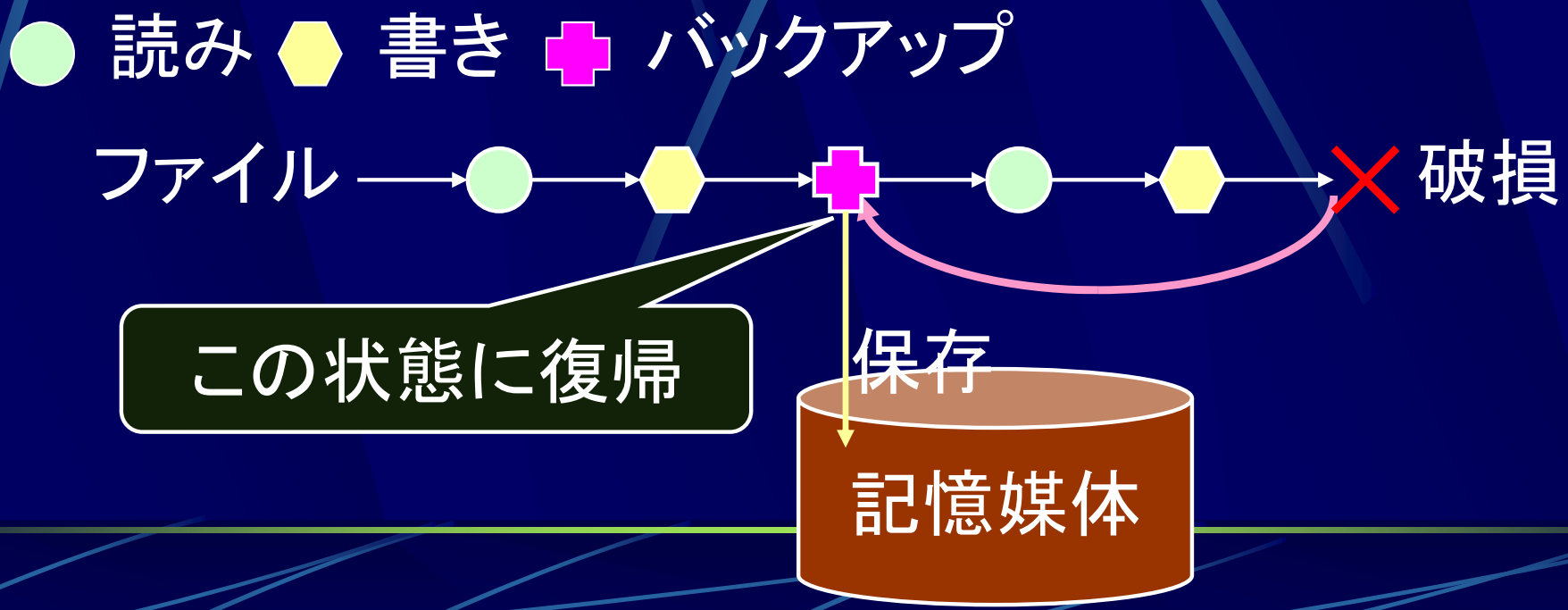
UNIXの場合

```
$ ls -al
total 251
drwxr-xr-x    8 user1 users 1024 Nov 16 19:04 .
drwxr-xr-x    6 root   root   1024 Nov 13 23:57 ..
-rw-r--r--    1 user1 users 1423 Nov 13 23:57 .Xdefaults
-rw-r--r--    1 user1 users   124 Nov 13 23:57 .bashrc
lrwxrwxrwx    1 user1 users    15 Nov 16 23:11 bin -> /usr/local/bin/
drwx------    2 user1 users 1024 Nov 14 00:03 nsmail
drwxr-xr-x    5 user1 users 1024 Nov 16 02:40 public.shtml
-rw-rw-r--                                    est.txt
```

rw-	r--	r--
owner	group	other

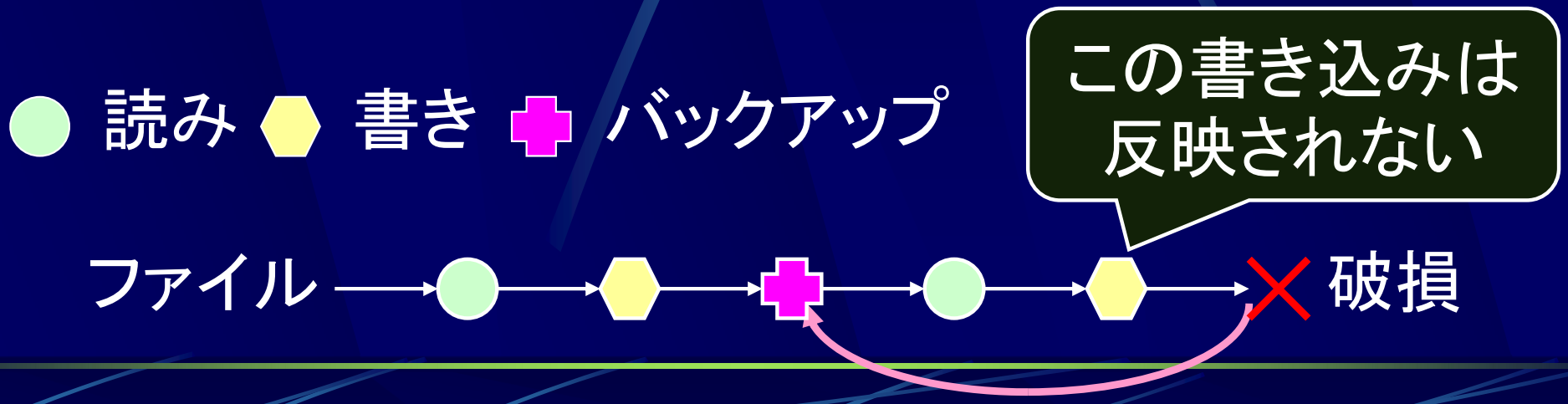
バックアップ(backup)

- バックアップ(buckup)
  - 定期的にファイル全体を他の記憶媒体に保存
    ファイル破損時に保存した状態に復帰



バックアップの欠点

- バックアップの欠点
  - 全てのファイルのコピーには時間が掛かる
  - バックアップ中はシステムを停止する必要
  - 最終バックアップ後の更新は反映されない



分散環境でのバックアップ

- 分散環境では計算機間の整合性が必要

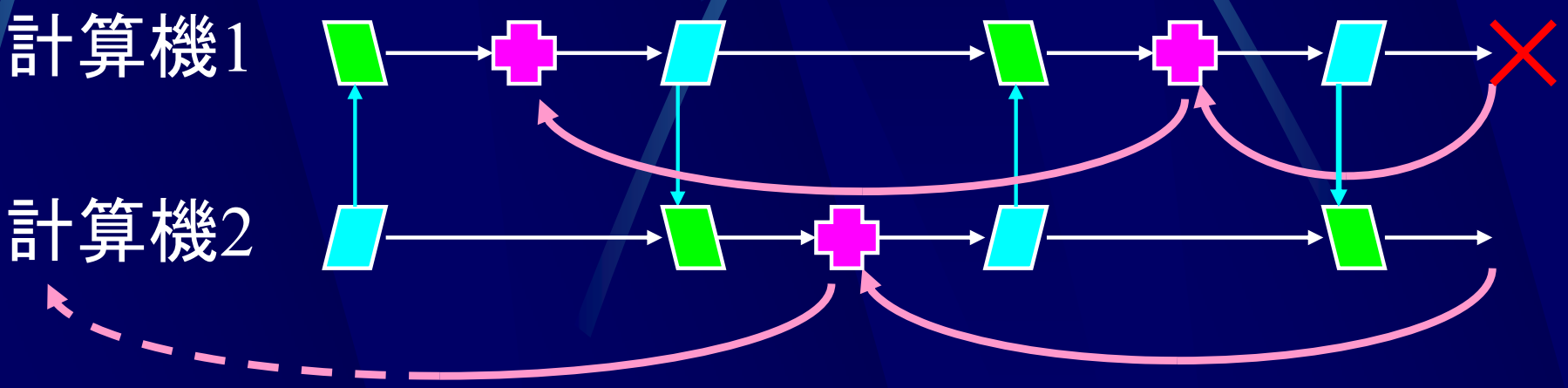
整合性を取るために受信前まで戻す

分散環境でのバックアップ

- 分散環境では計算機間の整合性が必要

読み　書き　バックアップ　送信　受信

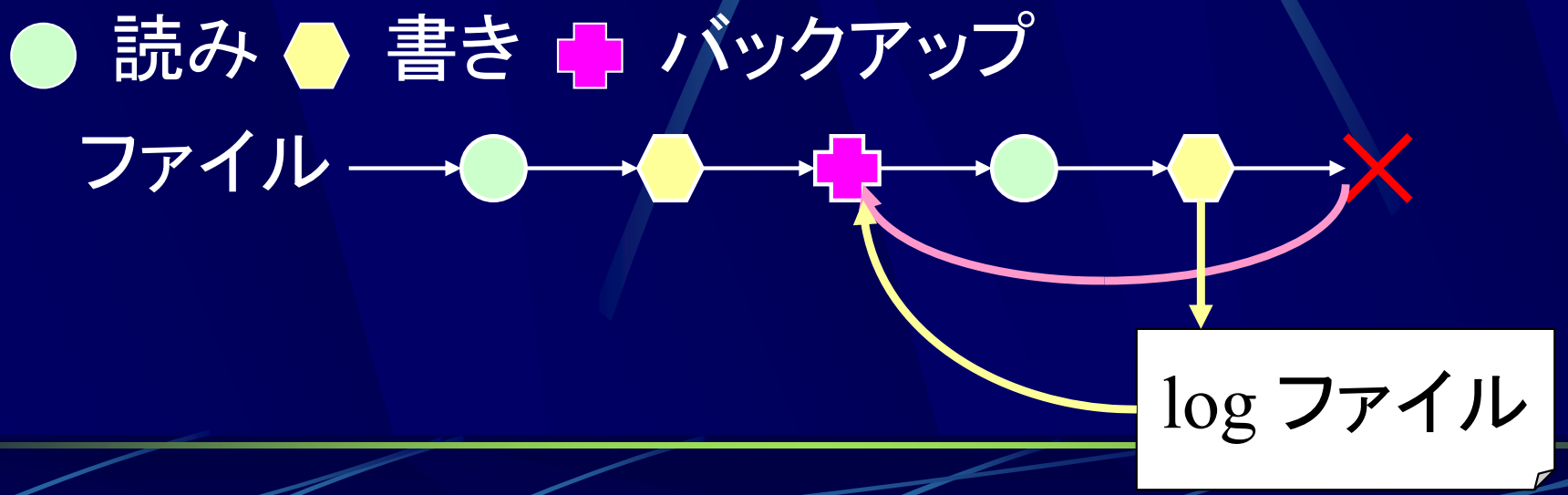
データ



際限なく戻さなくてはならない可能性もある

インクリメンタルダンピング (incremental dumping)

- インクリメンタルダンピング (incremental damping)
  - 変更された部分のみバックアップする
  - 変更された部分は履歴(log)ファイルに保存

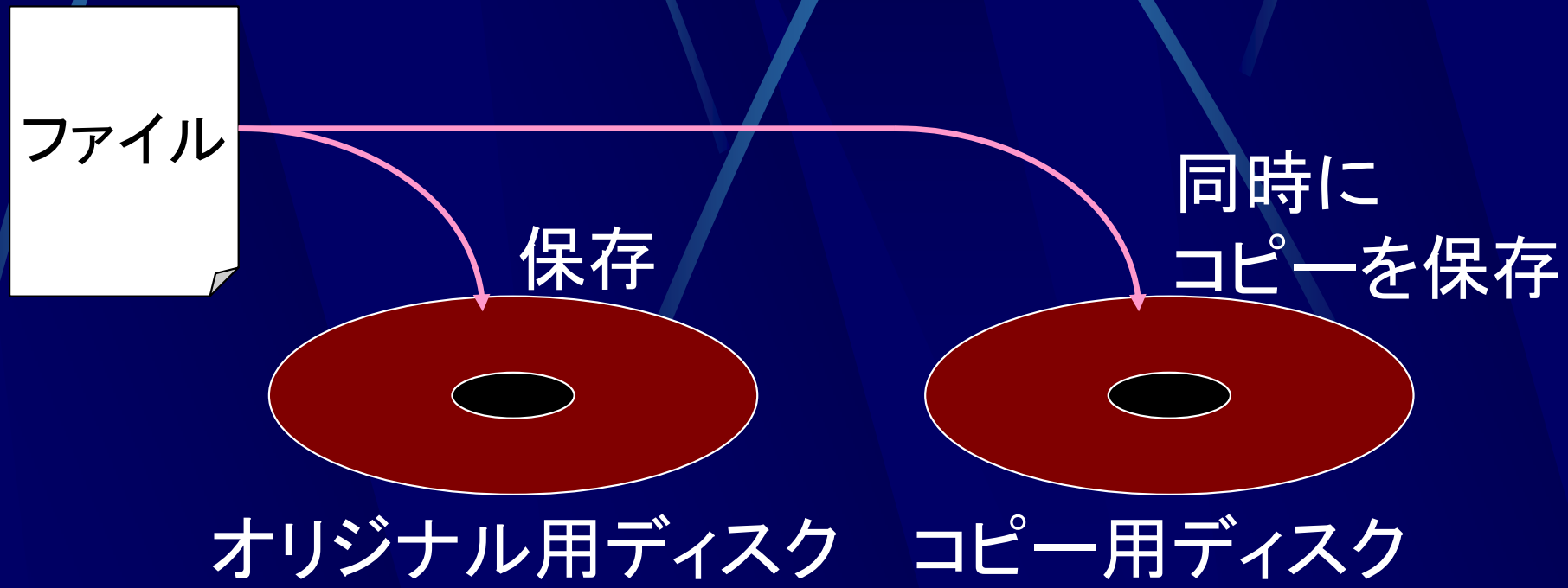


インクリメンタルダンピングの長所と短所

- インクリメンタルダンピングの長所
  - バックアップ後の書き込みも復元可能
- インクリメンタルダンピングの短所
  - バックアップからの時間経過に従い履歴ファイルが大きくなる

ミラーリング(mirroring)

- ミラーリング(mirroring)
  - ファイル格納用のディスクを2重化する



ミラーリングの長所と短所

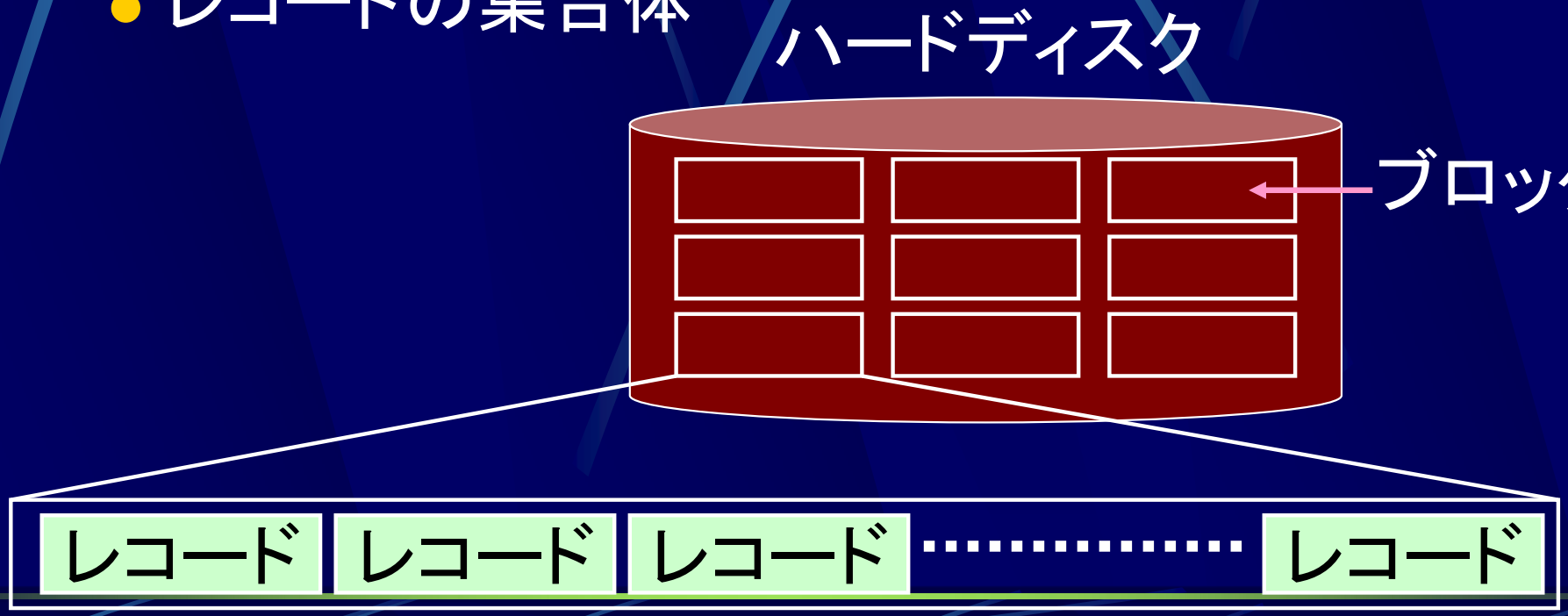
- ミラーリングの長所
  - オリジナル破損時に復元可能
- ミラーリングの短所
  - システム障害発生時にオリジナルとコピーの内容に違いが生じる可能性
  - 記憶領域が2倍必要

ファイルの実装

- ファイルの実装
  - 連続割り付け(contiguous allocation)
    - 連続したブロックに割り付け
  - 非連続割り付け(non-contiguous allocation)
    - リンク割り付け(linked allocation)
      - ブロックのリストとして構成
    - 索引割り付け(index allocation)
      - リンク情報を索引としてメモリ上に置く

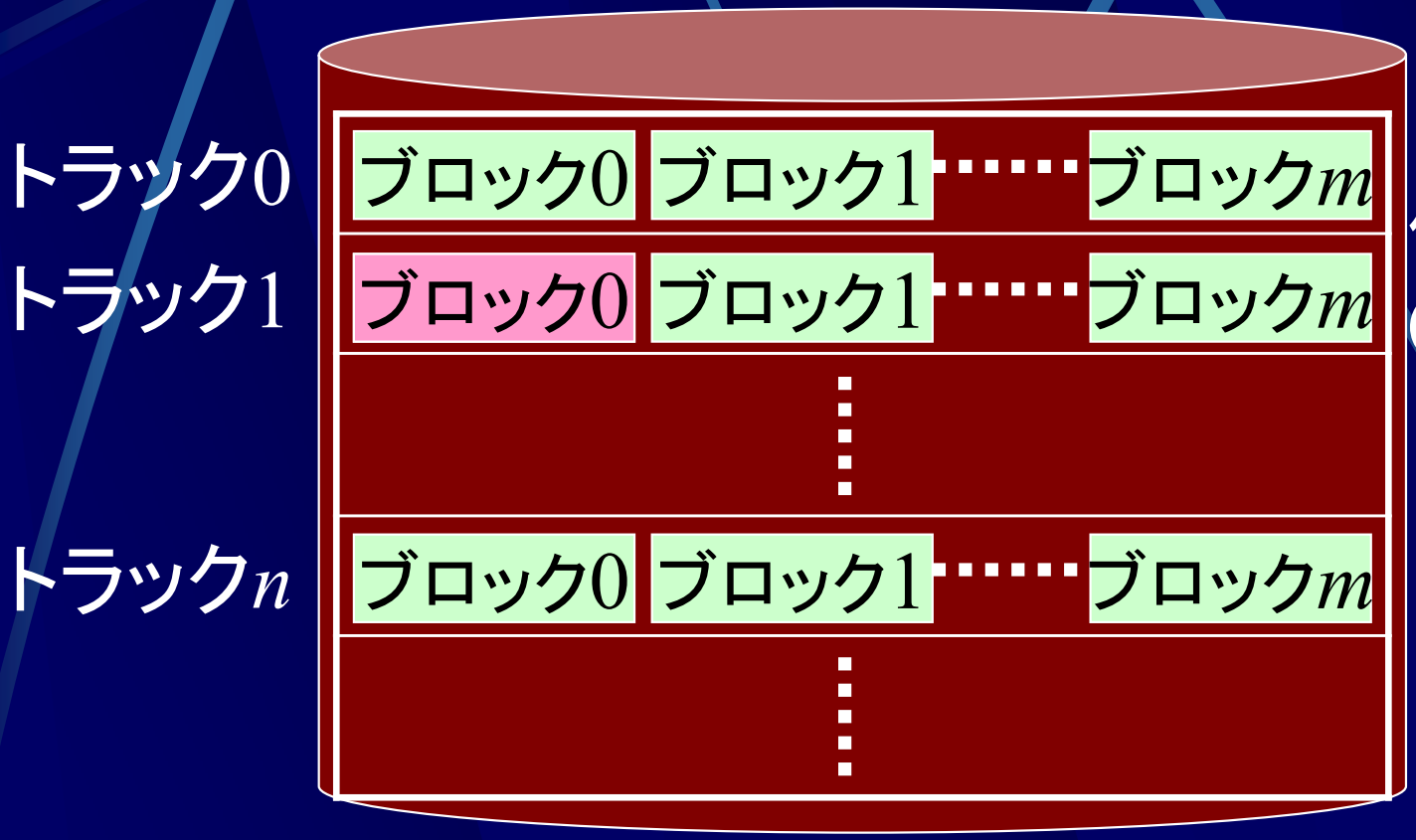
ブロック(block)

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



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



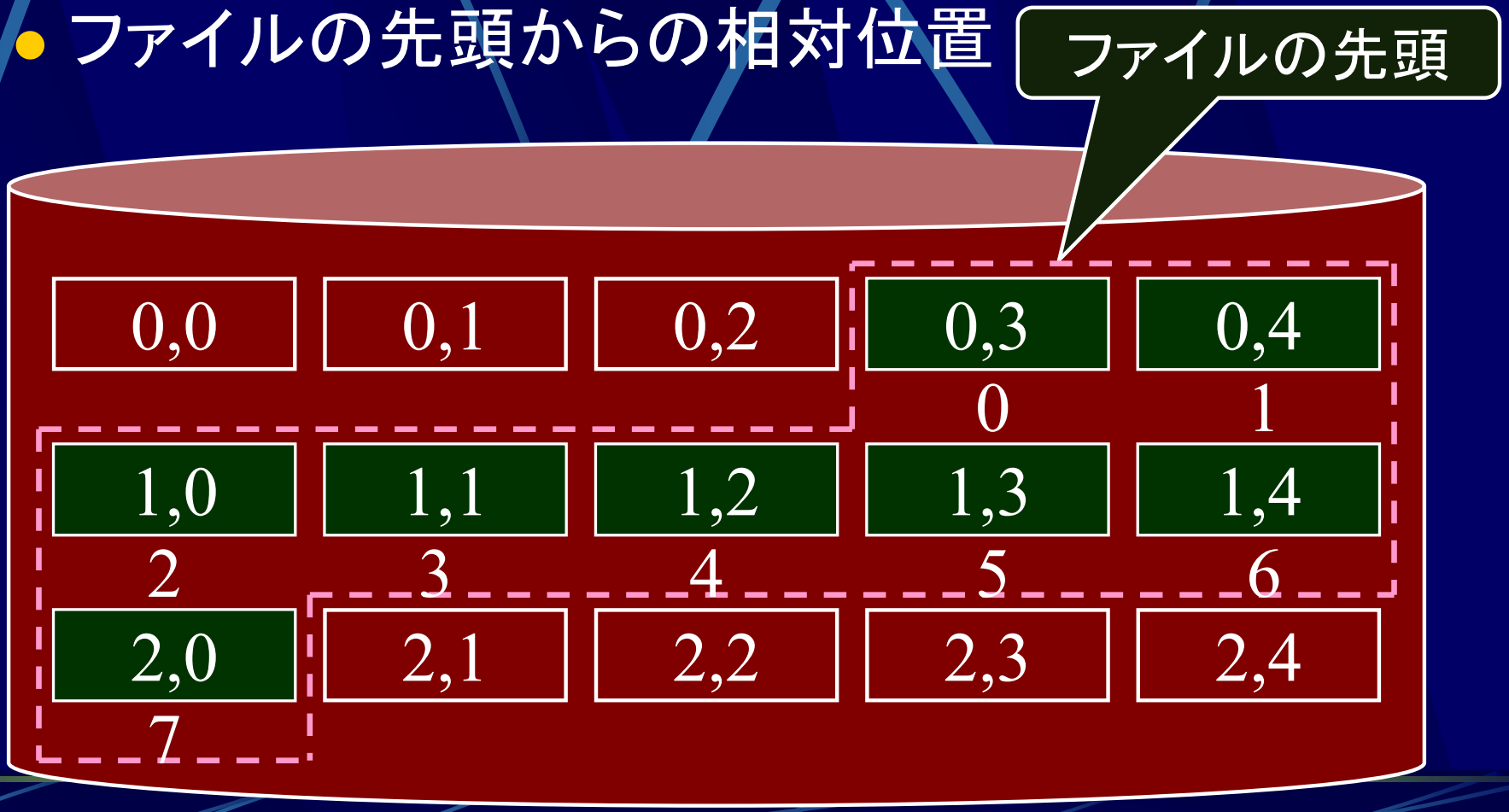
例：ブロック(1, 0)
(トラック, ブロック)

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

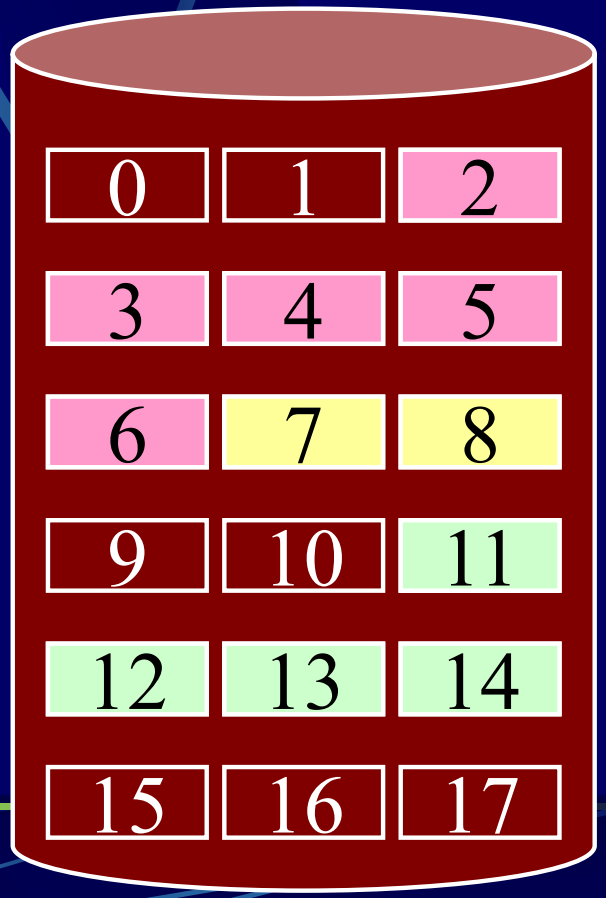


連続割り付け
(contiguous allocation)

- 連続割り付け(contiguous allocation)

ディレクトリ

ファイル名	開始位置	ブロック数
ファイル1	2	5
ファイル2	7	2
ファイル3	11	4



連続割り付けの長所と短所

- 連続割り付けの長所
  - 実装が容易
  - 先頭から連続してのアクセスが短時間で可能
  - 直接アクセスが短時間で可能
- 連続割り付けの短所
  - ファイル生成時にサイズを決める必要がある
    - 領域過小 ⇒ ファイル成長時に再構成が必要
    - 領域過大 ⇒ 未使用領域が増える
  - 断片化が生じる

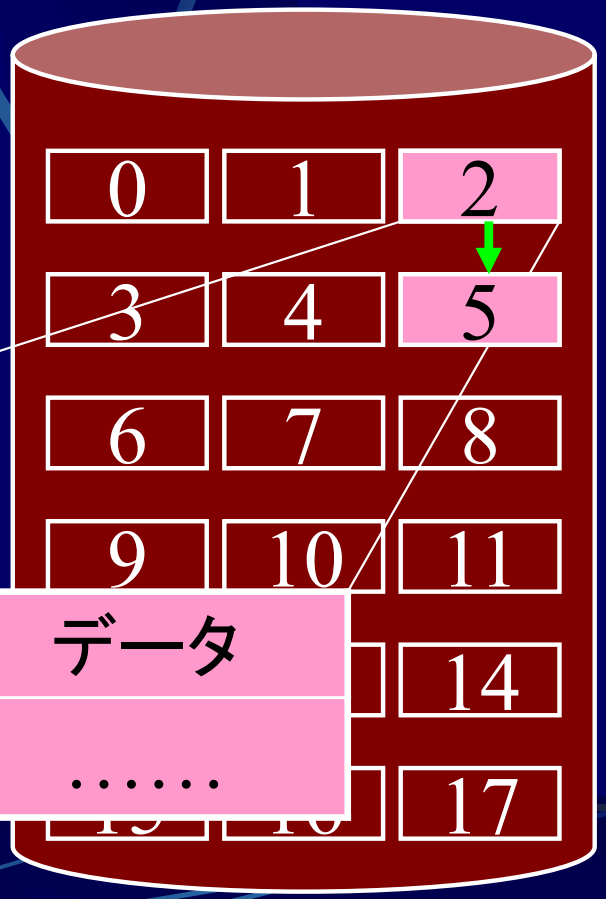
リンク割り付け
(linked allocation)

- リンク割り付け(linked allocation)

ディレクトリ

ファイル名	開始位置	終了位置
ファイル1	2	4
ファイル2	7	10
ファイル3		

ブロック	リンク先	データ
2	5	……

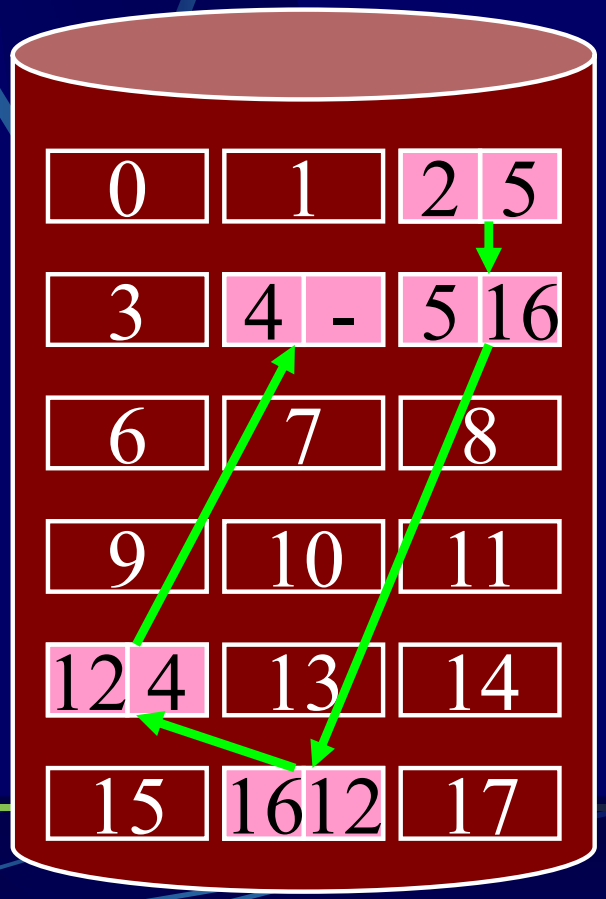


リンク割り付け
(linked allocation)

- リンク割り付け(linked allocation)

ディレクトリ

ファイル名	開始位置	終了位置
ファイル1	2	4
ファイル2	7	10
ファイル3	11	9

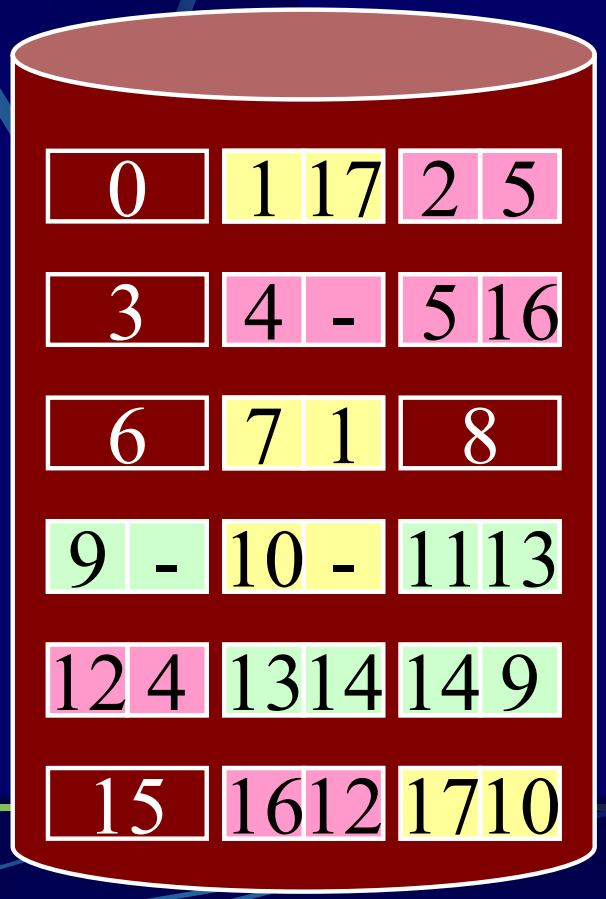


リンク割り付け
(linked allocation)

- リンク割り付け(linked allocation)

ディレクトリ

ファイル名	開始位置	終了位置
ファイル1	2	4
ファイル2	7	10
ファイル3	11	9



リンク割り付けの長所と短所

- リンク割り付けの長所
  - 生成時のファイルサイズ予想が不要
    - 自由に大きさ変更可能
  - 断片化が発生しない
- リンク割り付けの短所
  - 直接アクセスが非効率
    - ポインタを辿る必要
  - 信頼性が低い
    - リンク情報が破損すると読み出せない

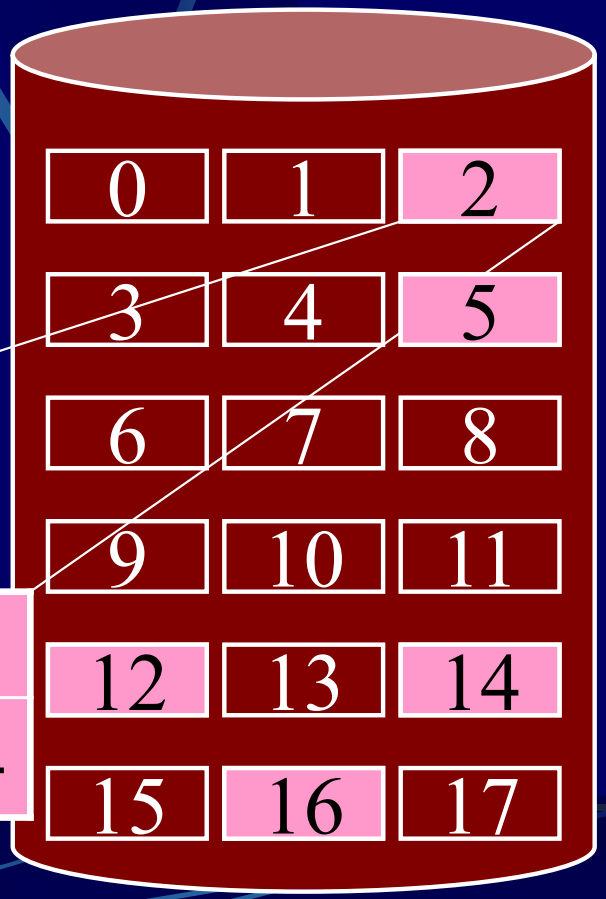
索引割り付け
(index allocation)

- 索引割り付け(index allocation)

ディレクトリ

ファイル名	索引位置
ファイル1	2
ファイル2	7
ファイル3	

ブロック	索引
2	5, 16, 12, 14



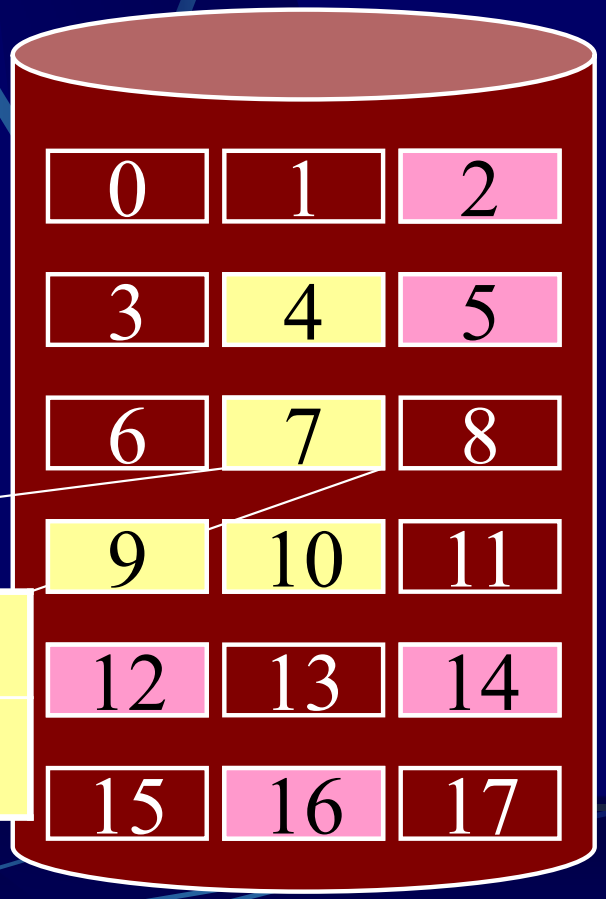
索引割り付け
(index allocation)

- 索引割り付け(index allocation)

ディレクトリ

ファイル名	索引位置
ファイル1	2
ファイル2	7
ファイル3	

ブロック	索引
7	4, 9, 10



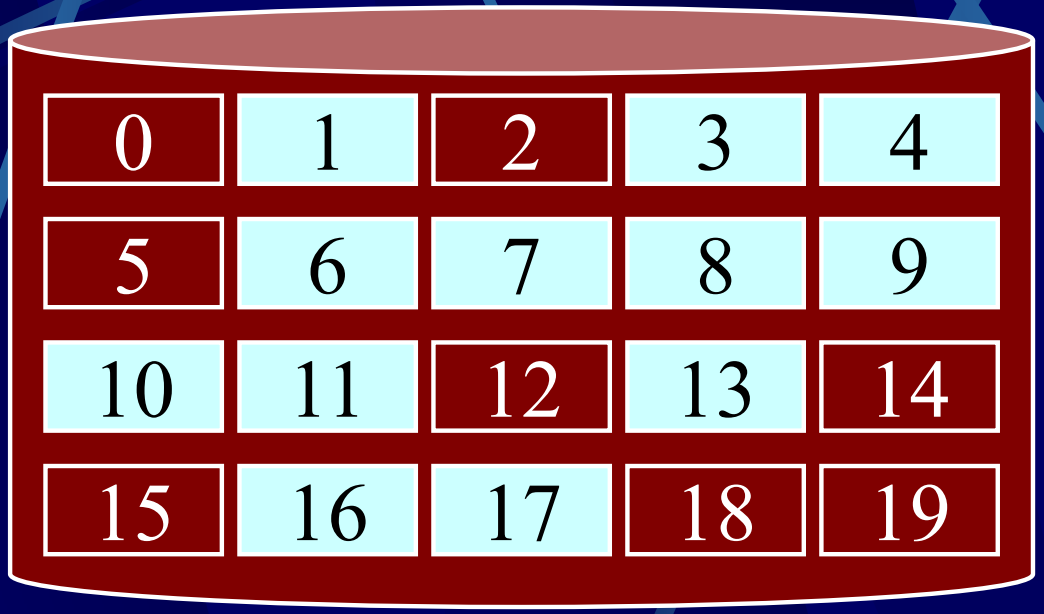
索引割り付けの長所と短所

- 索引割り付けの長所
  - 生成時のファイルサイズ予想が不要
  - 断片化が発生しない
  - 直接アクセスが短時間で可能
- 索引割り付けの短所
  - 索引のための領域が必要

空き領域の管理

- 空き領域の管理
  - ビットマップ方式(bit-map)
    - ブロック毎に空き/使用中 を管理
  - 連結リスト方式
    - ブロック内にポインタを入れて空きブロックを連結
  - 空き領域索引方式
    - 空きブロックの番号をブロックに格納

ビットマップ方式



空き

使用中

0	1	2	3	4	5	6	7	8	9
1	0	1	0	0	1	0	0	0	0
10	11	12	13	14	15	16	17	18	19
0	0	1	0	1	1	0	0	1	1

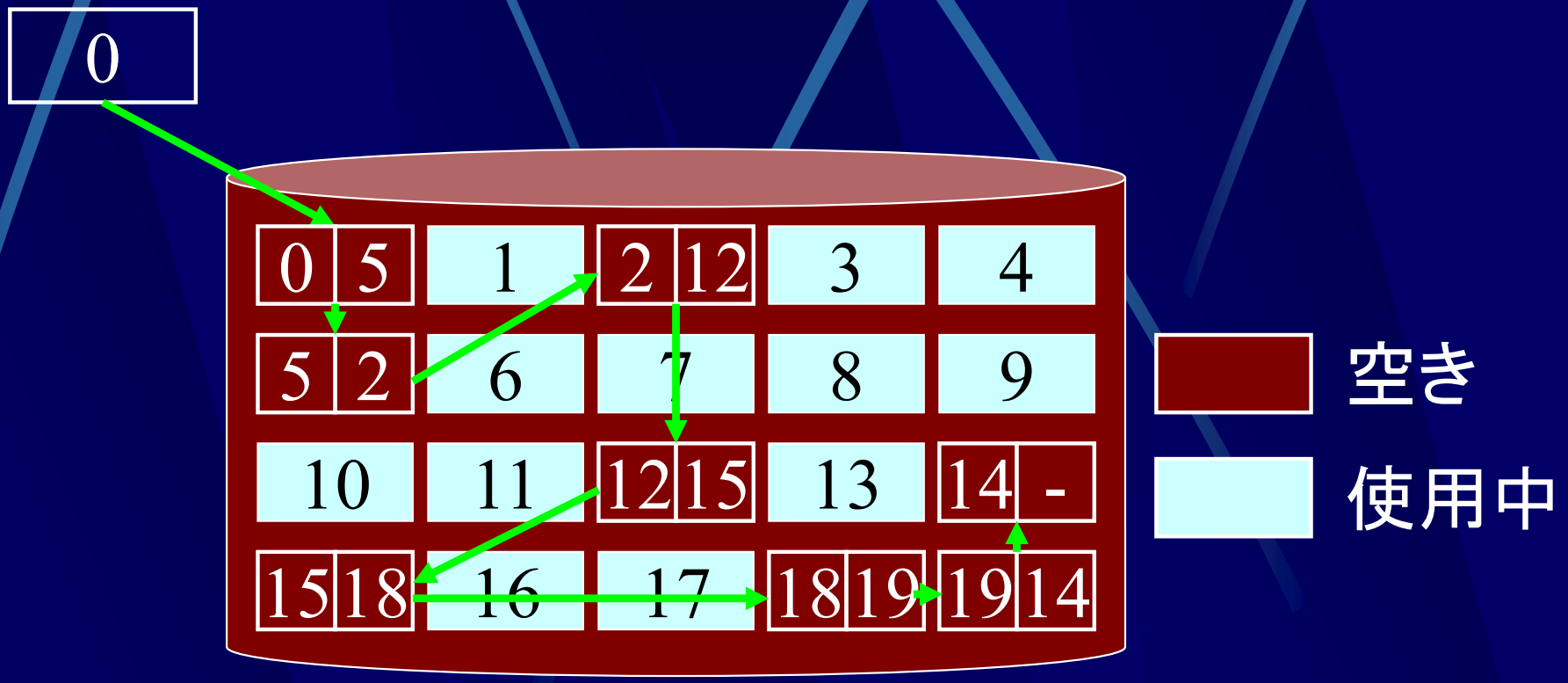
ビットマップ方式の長所と短所

- ビットマップ方式の長所
  - 高速で検索可能
  - 連続した空き領域を探し易い
- ビットマップ方式の短所
  - メモリ上にビットマップ表を置く必要がある
    - 二次記憶の大容量化により表が巨大化

    ⇒ 全ての表をメモリ上に置けないと非効率

連結リスト方式

先頭の空きブロックへのポインタ



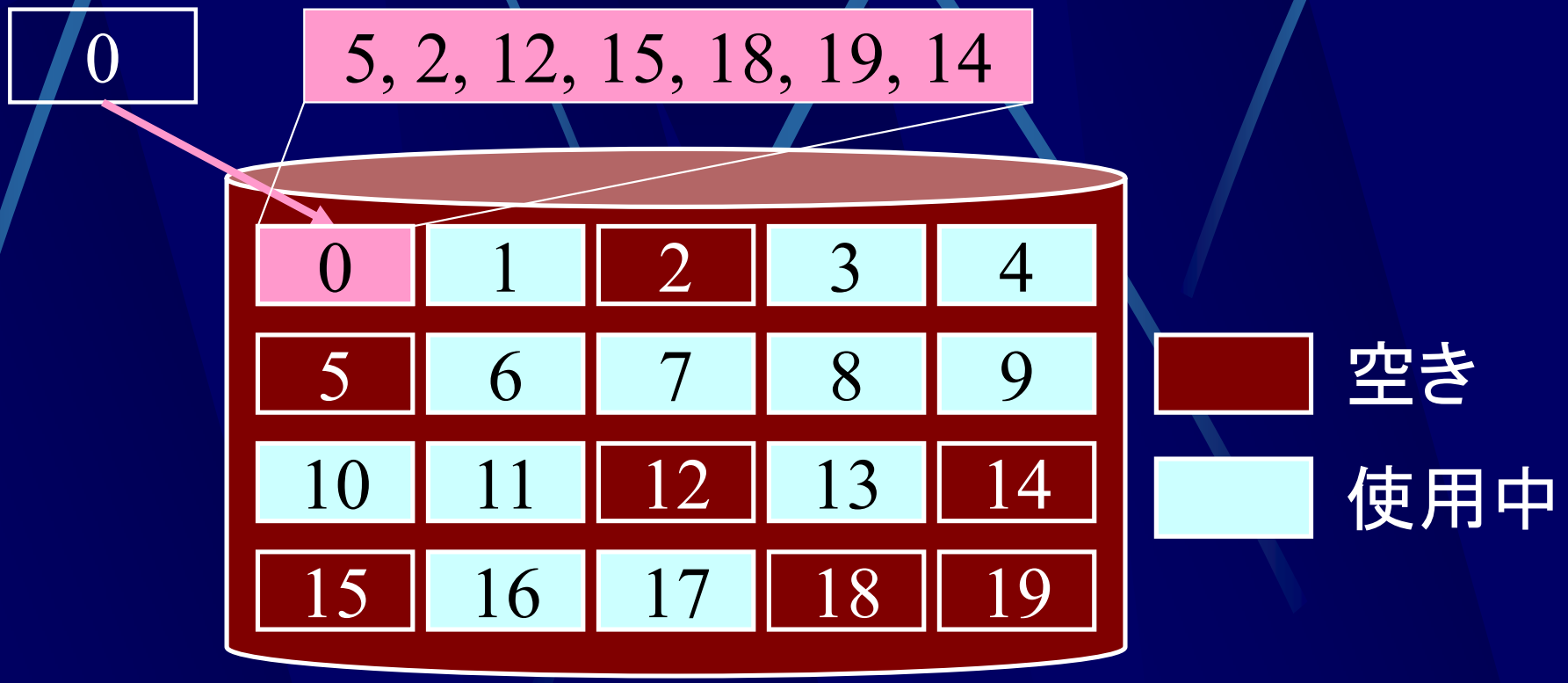
空きブロック中に次の空きブロックへのポインタを入れる

連結リスト方式の長所と短所

- 連結リスト方式の長所
  - メモリには先頭ブロックのみ記憶すればよい
- 連結リスト方式の短所
  - 連続した空き領域を探しにくい
    - ポインタの張替えが必要
  - 信頼性が低い
    - リンク情報が破損すると読み出せない

空き領域索引方式

索引ブロックへのポインタ



ブロックの1つに空きブロックの番号を格納する

空き領域索引の長所と短所

- 空き領域索引方式の長所
  - メモリ領域が充分に無いときに有用
- 空き領域索引方式の短所
  - ビットマップ方式よりも時間がかかる
  - 連続した空き領域を探しにくい

まとめ

- 通常ファイル(regular file)
  - プログラムやデータを格納する
- 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
  - 文字型特殊ファイル(character special file)
    - 1文字単位で入出力するデバイス
  - ブロック型特殊ファイル(block special file)
    - ブロック単位で入出力するデバイス

まとめ

- ディレクトリ
  - 論理的：ファイルを入れる容器
  - 物理的：ファイル管理用のファイル
- ディレクトリの構造
  - 単階層ディレクトリ
  - 2階層ディレクトリ
  - 木構造ディレクトリ
  - DAG構造ディレクトリ

まとめ

- ファイルの保護
  - アクセス制御：ユーザクラスで管理
  - バックアップと回復
- ファイルの実装
  - 連続割り付け
  - リンク割り付け
  - 索引割り付け

まとめ

- 空き領域の管理
  - ビットマップ方式
  - 連結リスト方式
  - 空き領域索引方式

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