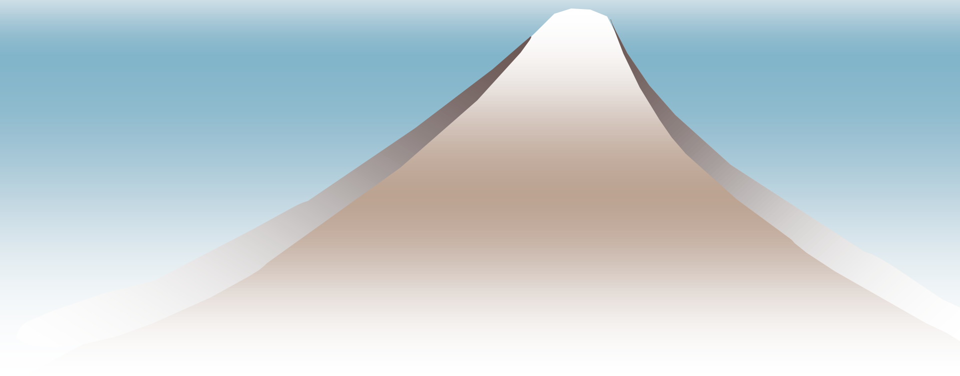
メモリマップドファイル

オペレーティングシステム



今日の流れ (12/10)

- ディスクの話の残り
  - ディスクを高速に使う工夫
- メモリとディスクの簡単なまとめ
- メモリマップト・ファイル (mmap)

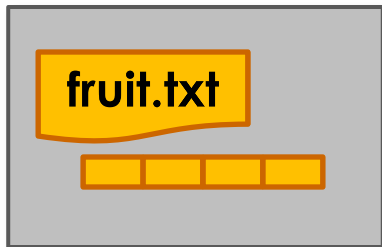
ディスクについて（前回の続き）

- ディスク (ハードディスク, DVDなど)
  - 電源を切っても消えない
  - 物理的にはシリンダ・ブロックに分かれている
  - OSによって抽象化され，ファイル単位でデータを管理できる
- アクセスはメモリに比べて遅い
  - →高速化する工夫

連続した領域への割り当て

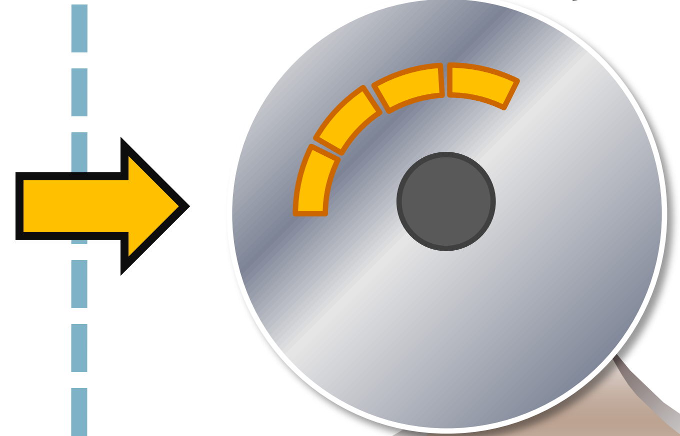
- 一度に読み出すのに都合の良いブロック(例: 同じシリンダ(円周)内の全ブロック)にファイルの連続した領域を割り当てる
  - cf. いわゆる「デフラグツール」
- 先読みの効果を大きくする

OS上では
一つのファイル



ディスク上では
断片化している

"デフラグ"によって
連続領域に割り当て

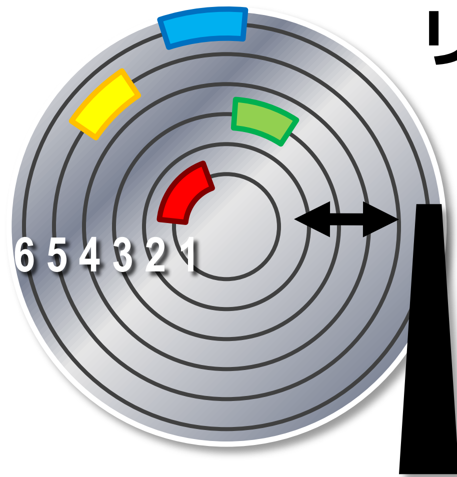


http://funini.com/kei/mmap/

ディスクスケジューリング

- アクセスすべきブロックを並び替えて，少ないヘッドの動きで一度に読む
  - 1,5,3,6というリクエストが来ても，1,3,5,6と並べ替えて読み，ヘッドの動きを少なくする

元々のリクエスト: 赤(1) 黄(5) 緑(3) 青(6)
リクエスト処理順: 赤(1) 緑(3) 黄(5) 青(6)



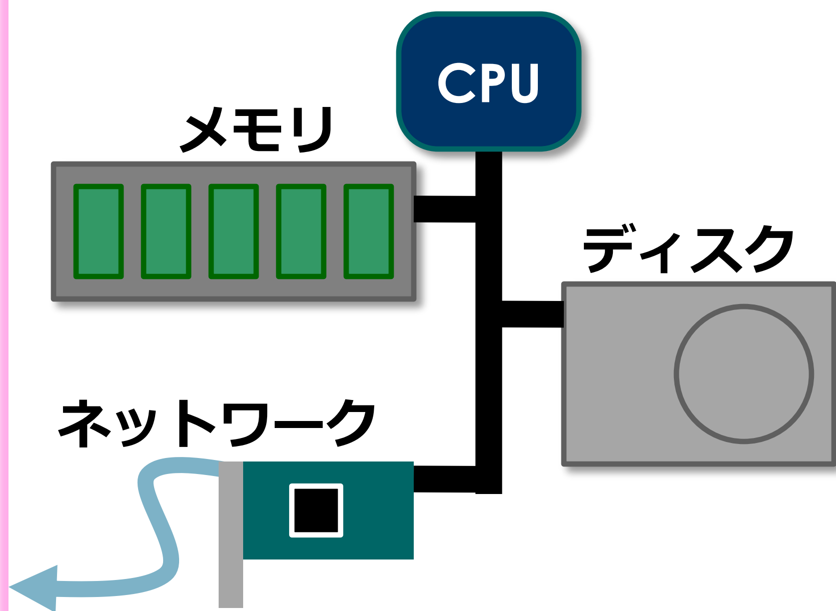
読み取りヘッド

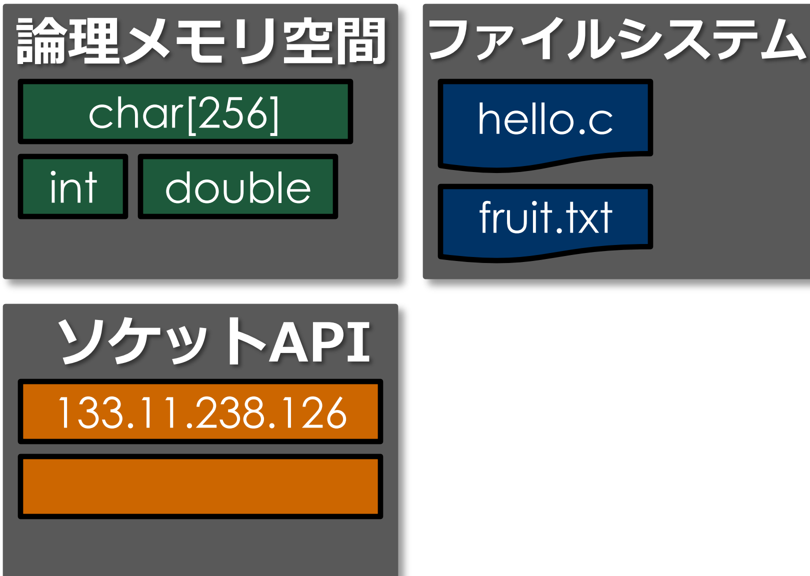
Agenda

- ディスクの話の残り
- メモリとディスクの簡単なまとめ
  - 仮想メモリ
  - ディスクキャッシュ
- メモリマップド・ファイル (mmap)

OSによるデバイスの抽象化

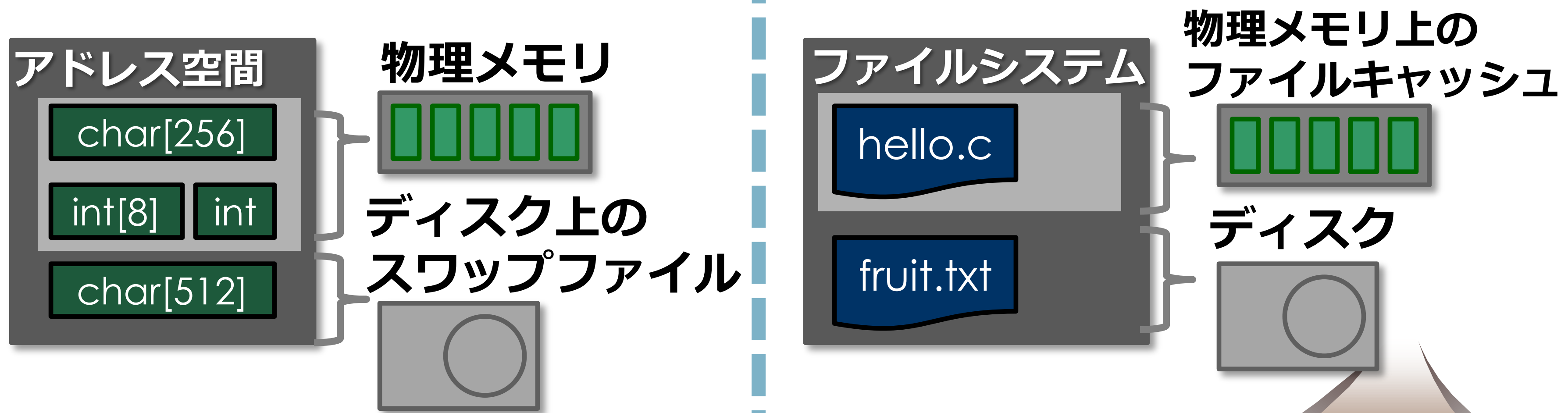
デバイス	CPU	メモリ	ディスク	ネットワーク
OSの見せ方	プロセス スレッド	論理メモリ空間 変数	ファイルシステム ファイル	TCP/IP ソケット





メモリとディスク

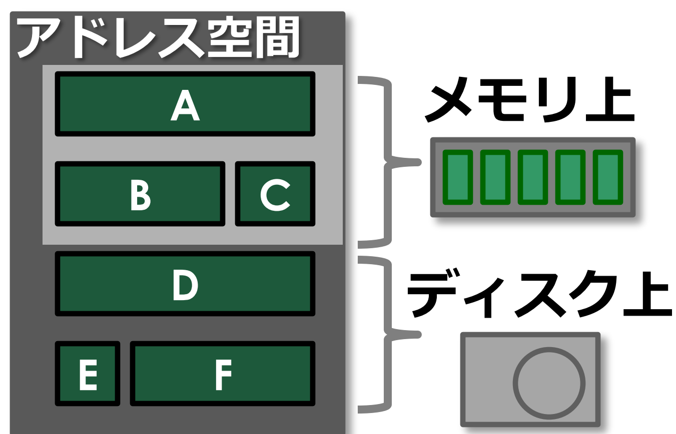
- OSは柔軟にメモリとディスクを組み合わせる
  - 物理メモリ: 速い・高価・揮発性
    →頻繁にアクセスするデータに適する
  - ディスク: 遅い・安価・不揮発性
    →広大な空間を必要とするデータに適する
- 仮想メモリ:「メモリに見えて実はディスク」
- File Cache:「ディスクに見えて実はメモリ」

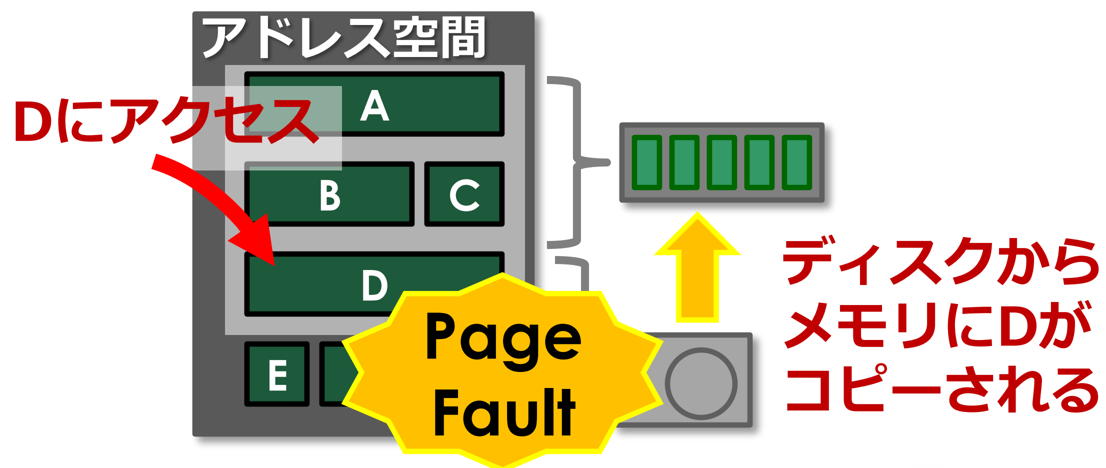


http://funini.com/kei/mmap/

仮想メモリ:ディスクを用いてメモリを拡張

- 物理メモリより大きなアドレス空間を提供
  - 頻繁にアクセスされるページは物理メモリ上
  - 物理メモリ上に無い番地にアクセスすると,
    ページフォルト(Page Fault)が発生して
    ディスクからメモリにデータが読み込まれる

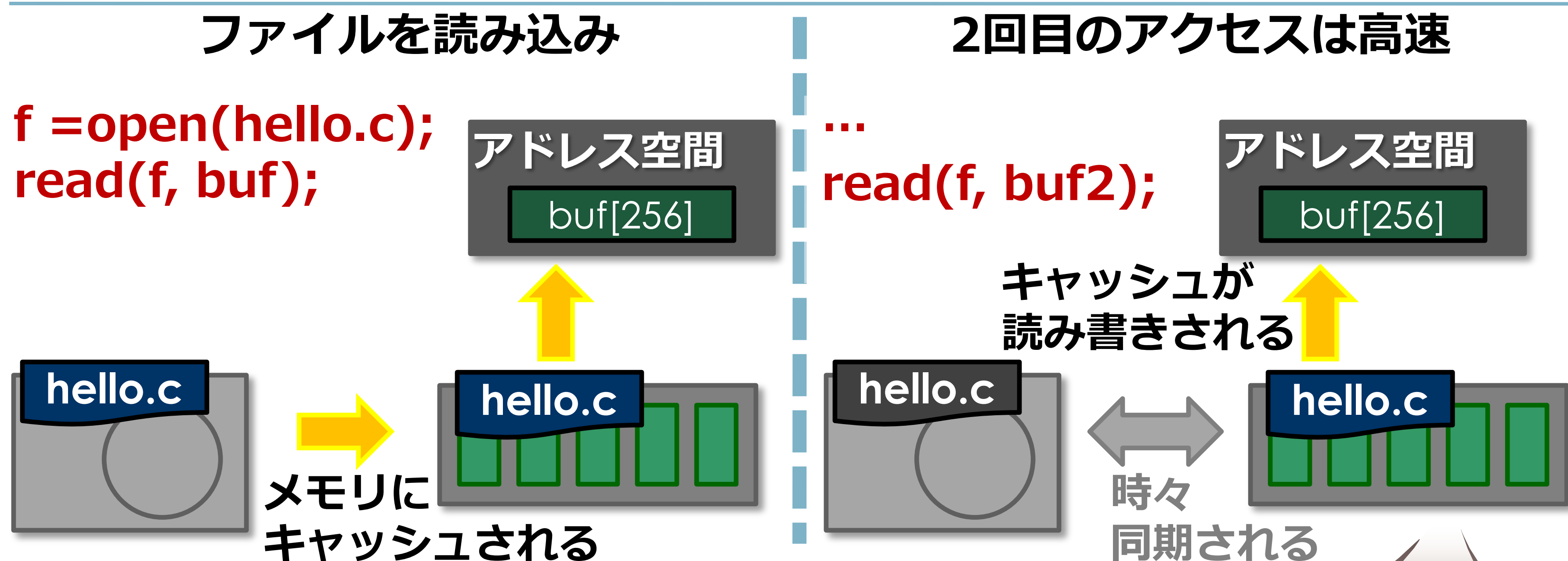




http://funini.com/kei/mmap/

File Cache:メモリを用いてディスクを高速化

- ファイルの一部をメモリ上にキャッシュ
  - アクセスしたファイルをメモリ上にキャッシュ
  - 2回目からはキャッシュに対しアクセス
- 2回目はメモリコピーと同じアクセス速度になる



Agenda

- ディスクの話の残り
- メモリとディスクの簡単なまとめ
- メモリマップド・ファイル(Mmap)
  - 使い方
    - ファイルをメモリみたいにアクセス
    - 共有マッピングでプロセス間でデータの共有
    - メモリ確保 (mallocの実体?)
  - 仕組み
  - プライベートマッピングの最適化
  - mmapの利用価値

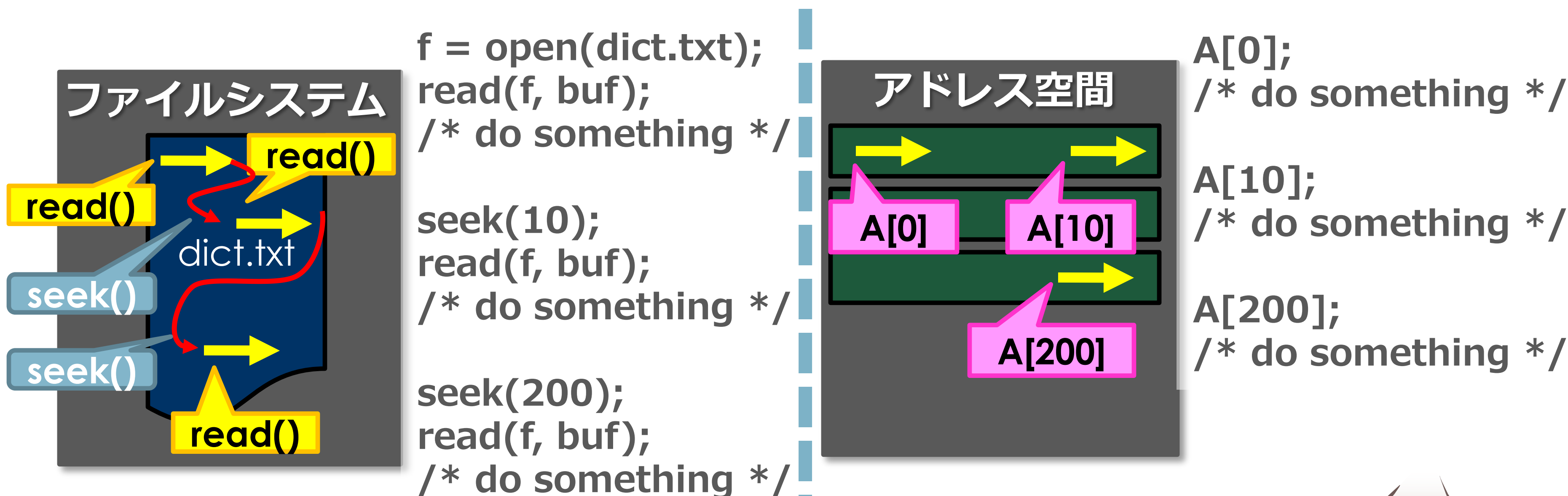
ファイルAPI

- ファイルAPIはstream(流れ)志向
  - read()は前の読み出し位置を覚えている
- メモリはランダムアクセス志向
  - いつでも配列の任意の場所を読み書きできる

	開く	読み込み	書き込み
メモリ	malloc(128) int A[10];	i A[3]	i = 10 A[10] = 128
ファイル	open()	read() seek()	write() seek()
ネットワーク	socket, connect	recv()	send()

ファイルをランダムアクセスしたい場合

- 例:大きな辞書ファイルを引く
- seek(), read()を繰り返してもいいが面倒
- ファイルをメモリのように扱えると便利
  →**メモリマップドファイル (mmap)**

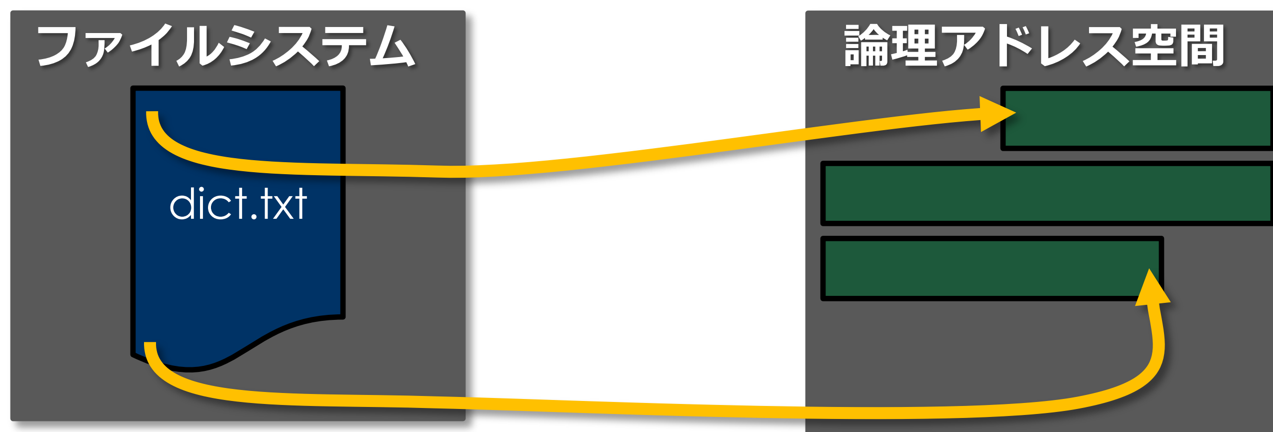


http://funini.com/kei/mmap/

メモリマップドファイル

- 基本:ファイルを明示的なread/writeではなく「あたかもメモリの様に」読み書きするAPI

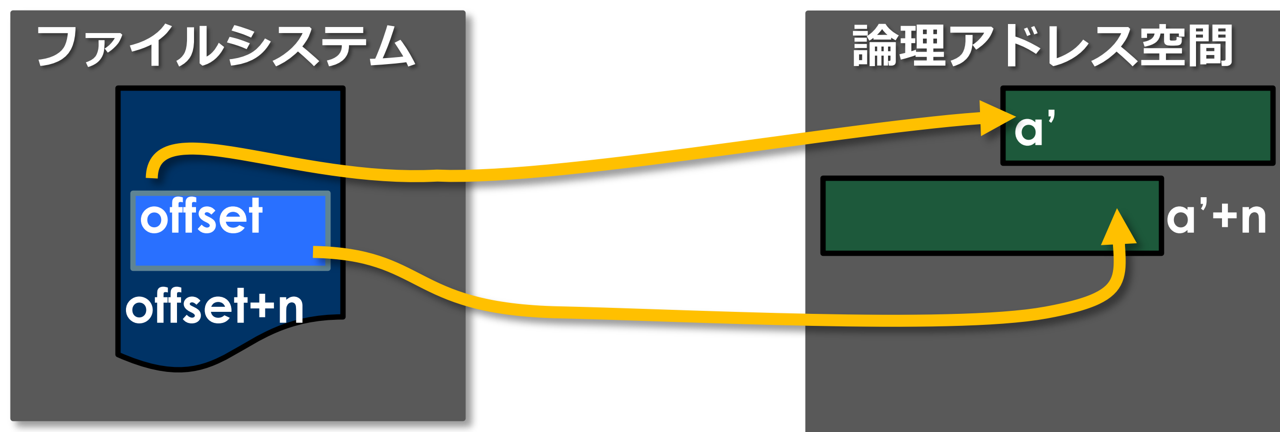
```
fd=open("dict.txt"...);
A=mmap(.., fd, ..);
/* do something */
s = A[100];
```



http://funini.com/kei/mmap/

メモリマップドファイル: Unix API

- fd = open(file, access);
  a′ = **mmap(a, n**, prot, share, **fd**, **offset);**
  - 意味: “fileのoffsetバイトから始まるnバイトを, アドレス[a′, a′ + n)でaccess可能にする”
    - a ≠ 0 ⇒ a′ = a (空いていれば)
    - a = 0 ⇒ a′はOSが選ぶ



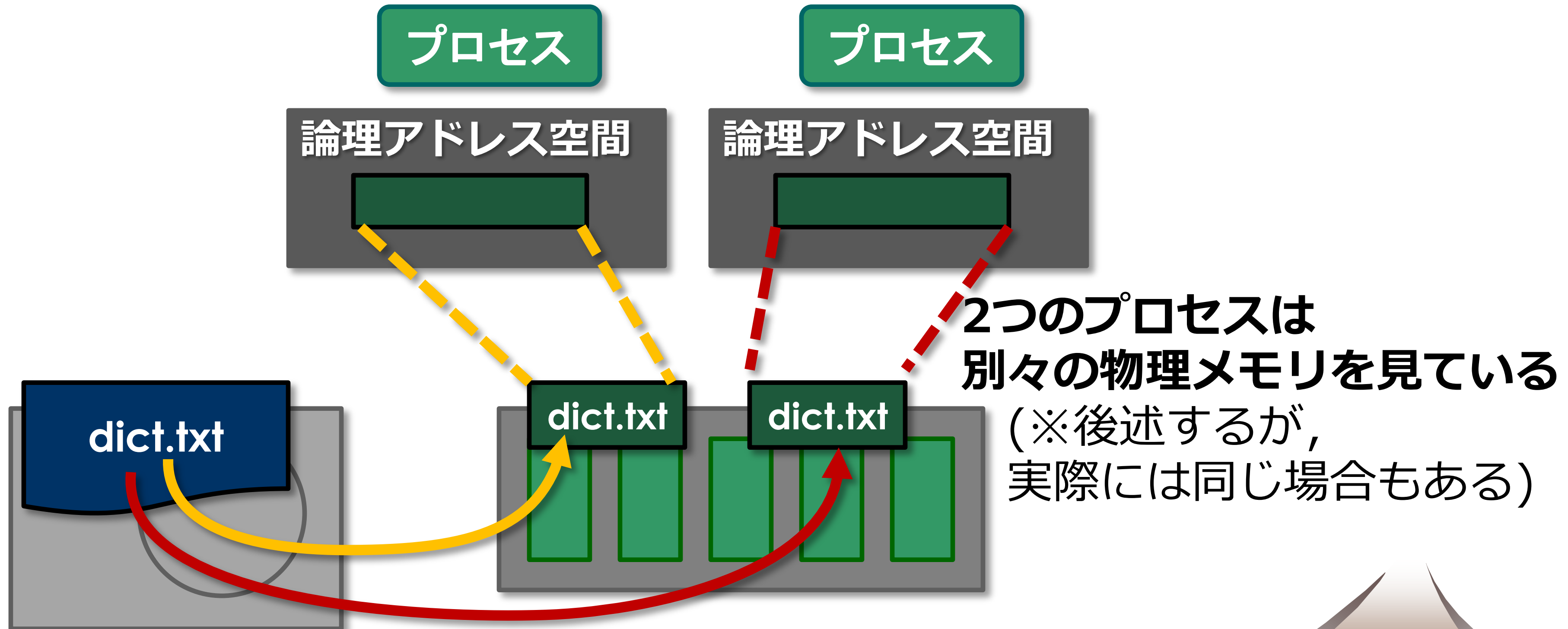
プライベート/共有マッピング

mmap(a, n, prot, **share**, fd, offset**);**

- パラメータ*share*
  - 複数のプロセスが同じファイルをmmapした場合の挙動を指定
  - ***share* = MAP_PRIVATE**
    - プロセスごとに別のコピーを見る
    - 書き込み結果はファイルに反映されず，プロセス間でも共有されない
  - ***share* = MAP_SHARED**
    - 複数のプロセスが同じデータを見る
    - 書き込み結果はプロセス間で共有され，ファイルにも反映される

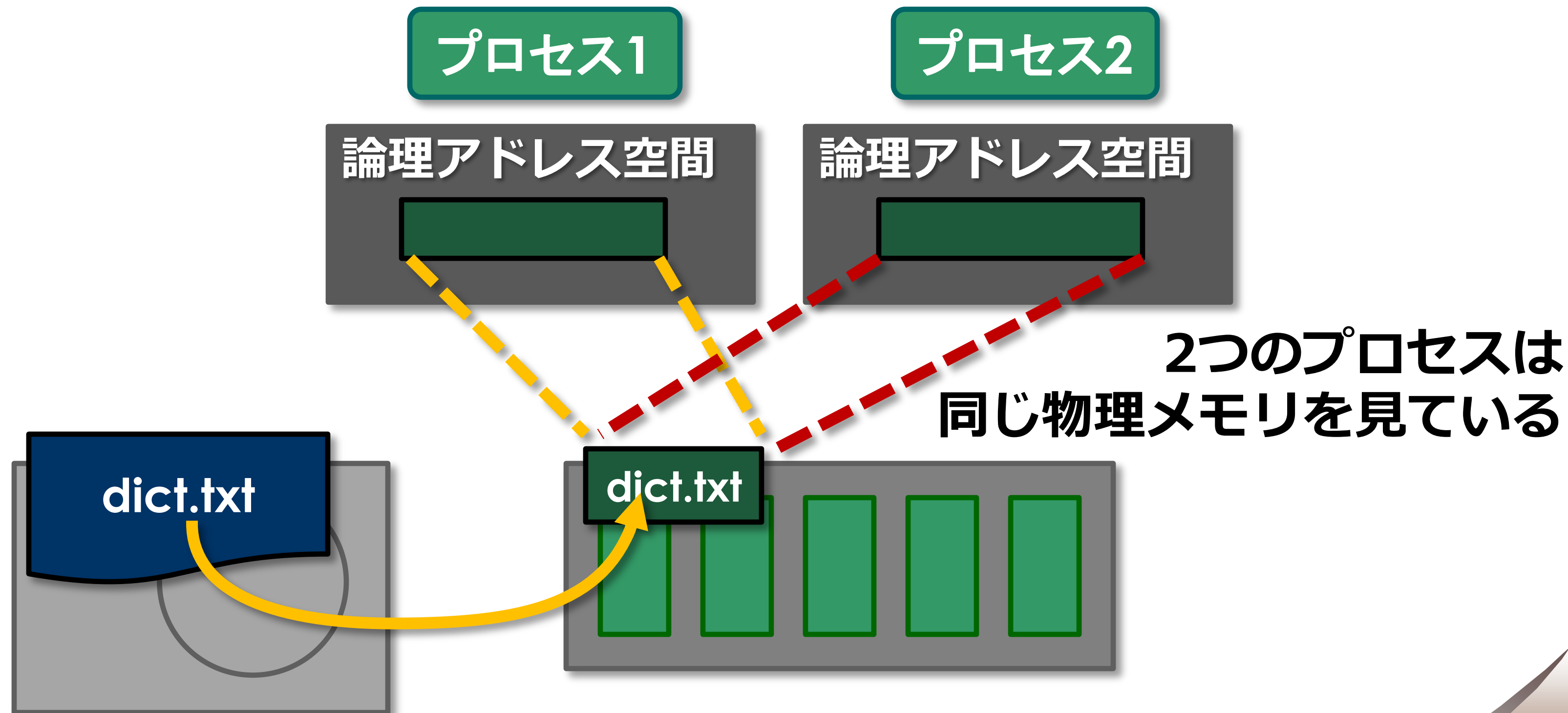
プライベートマッピング

- 同じファイルをマップした際，複数のプロセスが独立した領域を持つ



共有マッピング

- 同じファイルをマップした際，複数のプロセスが共通の物理メモリを参照できる
  - 書き込んだデータが共有される



メモリマップドファイル: Windows API

- h = CreateFile($file$, $access$, ⋯);
  m = CreateFileMapping(h, ⋯);
  a' = MapViewOfFileEx(m, $prot$, $offset1$, $offset2$, n, a);
- $prot$ = FILE_MAP_COPYでMAP_PRIVATEと似た効果を持つ

mmap()によるメモリの割り当て

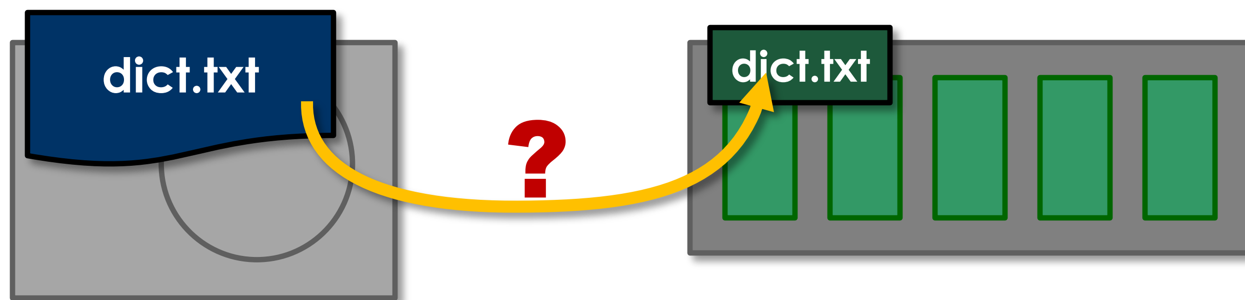
- brk (Unix)やVirtualAlloc (Win32)に代わるメモリ割り当て手段になっている
  - **Unix:** 特別なファイル/dev/zeroをMAP_PRIVATEでmmapすると，特定のファイルに結びついていないメモリ領域を得る
  - **Win32:** INVALID_HANDLE_VALUEをCreateFileMappingに渡すと同様の効果
  - malloc()の中で使われている

Agenda

- ディスクの話の残り
- メモリとディスクの簡単なまとめ
- メモリマップド・ファイル(Mmap)
  - 使い方
  - 仕組み
  - プライベートマッピングの最適化
  - mmapの利用価値

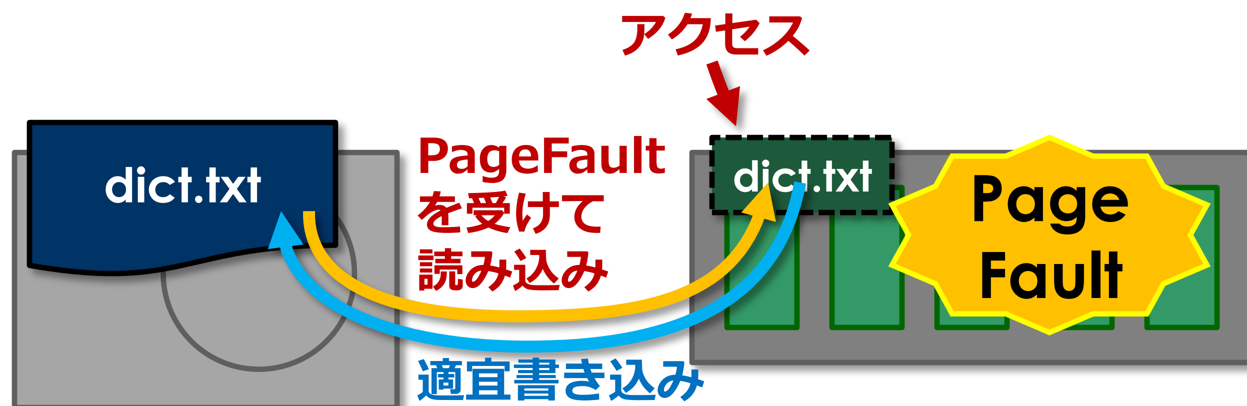
メモリマップドファイルの仕組み(1)

- mmap/MapViewOfFile etc.の実行時にファイルの中身をすべて読むわけではない
- mmapシステムコール内の動作:
  アドレス空間記述表へ，新たにmmapされた領域を記録する(だけ)
  - まだ物理メモリは割り当てない



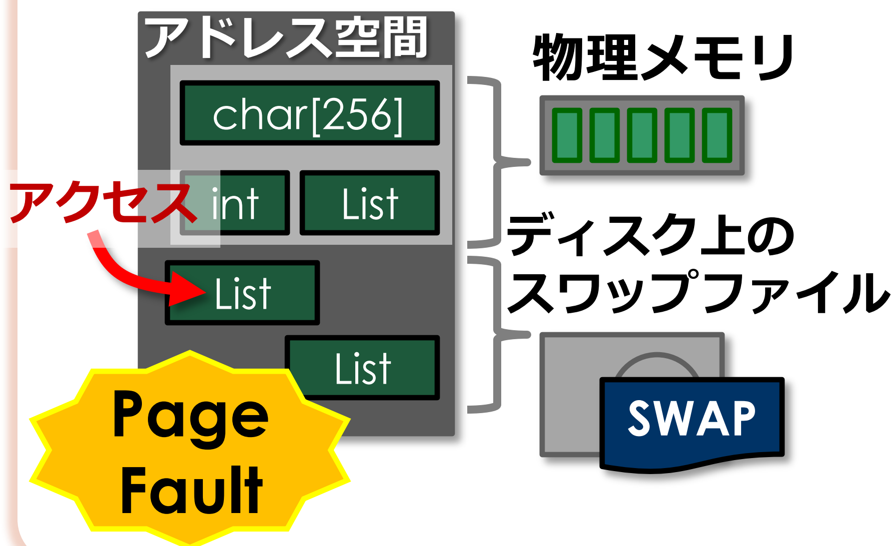
メモリマップドファイルの仕組み(2)

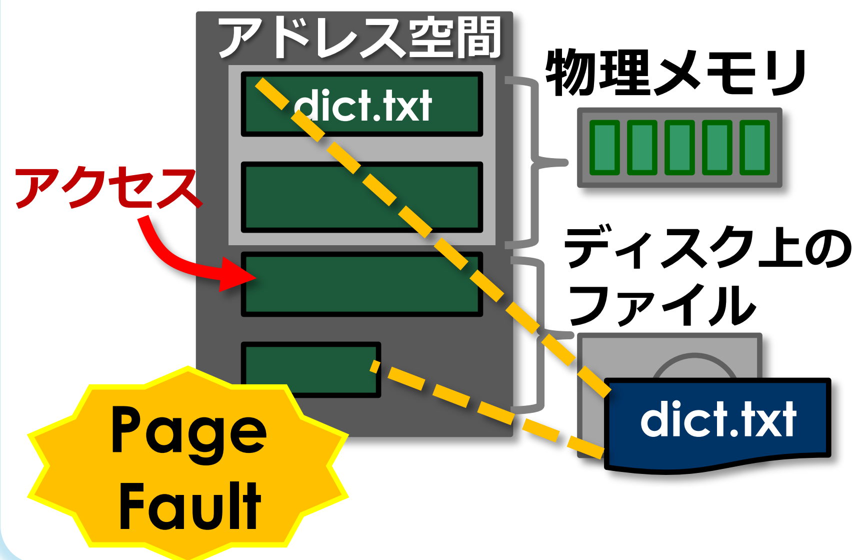
- mmapされたページが初めてアクセスされた際に，ページフォルトが発生
  ⇒ OSがファイルから内容を読み込む
- ページへの書き込み
  ⇒ 適当なタイミングで元のファイルに反映



メモリマップドファイルの仕組み(3)

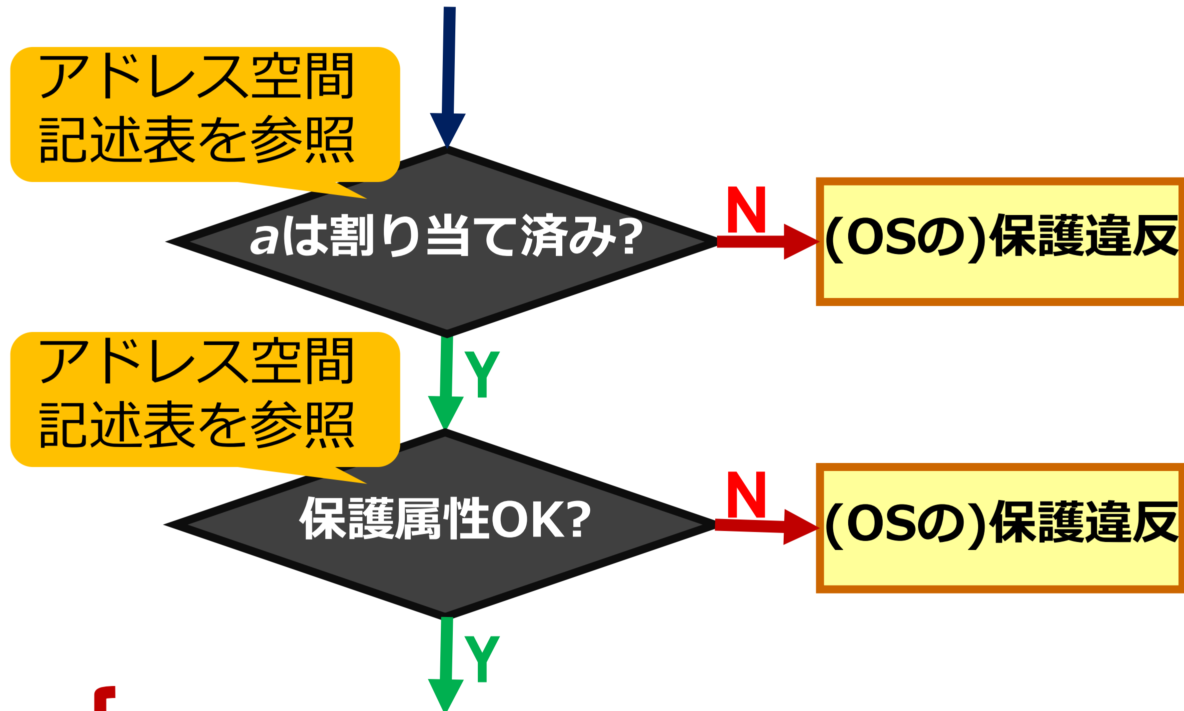
- OSにとっては，メモリ管理(仮想記憶)機構の自然な延長
  - メモリの退避場所としてスワップ領域の代わりに通常のファイルを使うだけ





http://funini.com/kei/mmap/

ページフォルト処理（復習）



次のスライドでは
ここを詳しく説明

物理ページ割り当て処理とその拡張

http://funini.com/kei/mmap/

デモ: mmapとreadの性能挙動観察

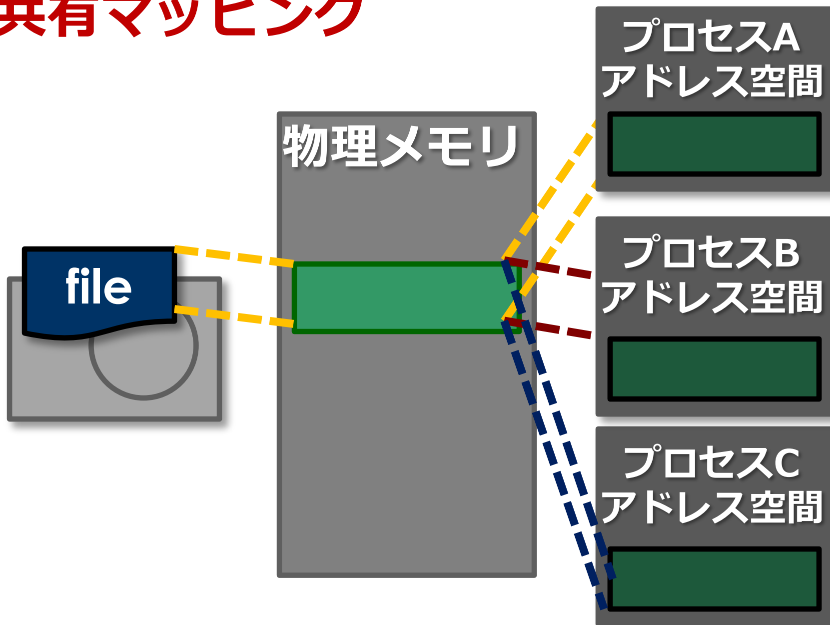
- 大きなファイルの全内容を次の二通りの方法でアクセス
  - mallocとreadでファイル全体に相当する内容を読み込んで、アクセス
  - mmapして配列のようにアクセス
  - 両方の手法で2回ずつ読み込み

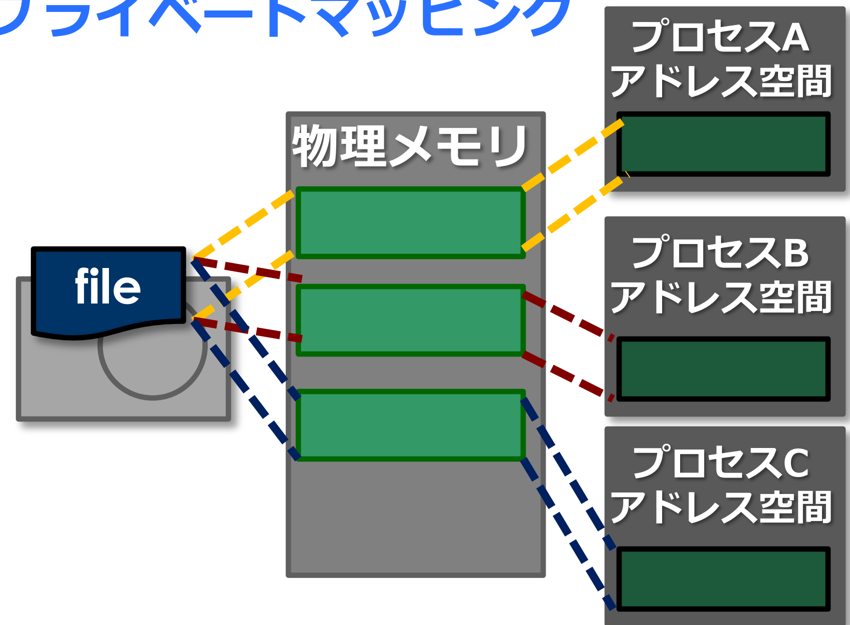
Agenda

- ディスクの話の残り
- メモリとディスクの簡単なまとめ
- メモリマップド・ファイル(Mmap)
  - 概要
  - 仕組み
  - プライベートマッピングの最適化
    - 読み出し専用マッピング
    - Copy-on-writeマッピング
  - mmapの利用価値

プライベート/共有マッピングの違い

- 共有: 全てのマッピングで物理メモリを共有
- プライベート: マッピングの数だけ物理メモリを消費
  - プライベートマッピングは(そのままだと)共有マッピングに比べて物理メモリの利用効率が悪い





http://funini.com/kei/mmap/

OSのプライベートマッピング最適化

- 考え方: 可能な限り物理メモリを共有する
  - **読み出し専用マッピング**
    - 明示的に「読み出し専用」としてマッピング
  - **"Copy-on-write"マッピング**
    - 書き込みが起こったらはじめてコピーする

読み出し専用マッピング

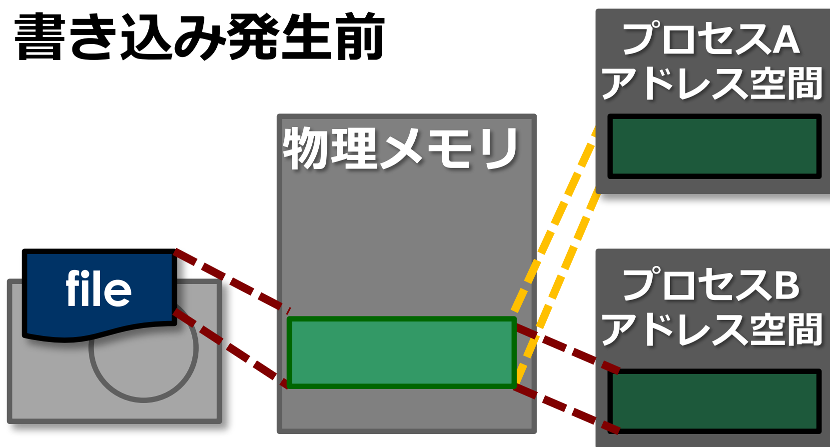
- 利用者が読み出し専用であることを指定する
  - 書き込みが起こらないので，プライベートマッピング間で常に物理メモリを共有できる
- 典型的使用場面
  - プログラム開始時にプログラムテキストを読み出すために使われている

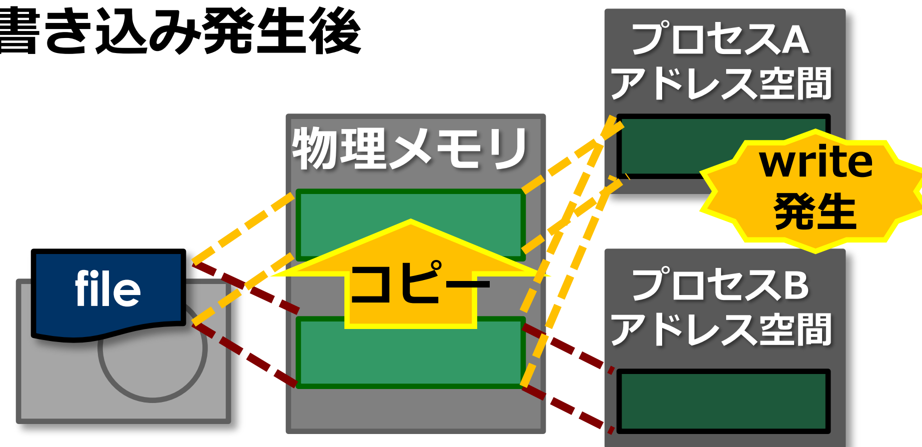
Copy-on-write

- コピーを作らないといけない場面で，実際に書き込みが起こるまでコピーをしない
  - mmap()でのプライベートマッピング
  - fork()でのメモリコピー
  - PHPやPythonでの値渡しの変数

Copy-on-writeマッピング

- 書き込み可でマップされた領域も，実際に書き込まれるまで物理メモリを共有しておく
  - 保護属性を「書き込み不可」にしておく (ページテーブル，TLB上で)
- 最初に書き込みが起きた時にCPU保護例外が発生
  - ここでOSが新しい物理ページを割り当て，コピーを作る



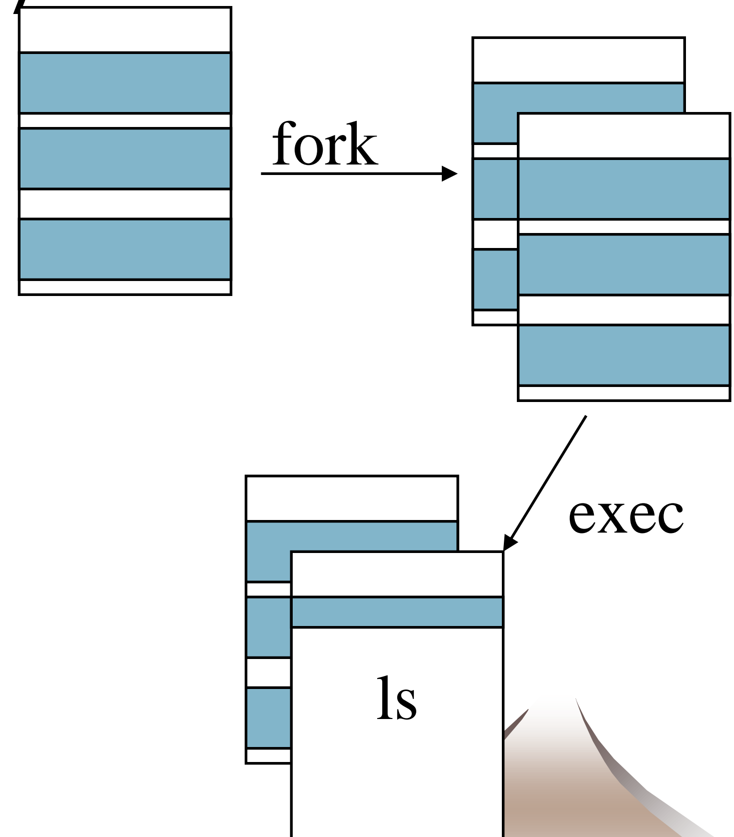


http://funini.com/kei/mmap/

応用: Copy-on-writeによる高速fork(1)

- fork：アドレス空間のコピー
- pid = fork();

```
pid = fork();
if (pid == 0) { /* child */
  …;
  execve(“/bin/ls”, …);
} else { /* parent */
  …;
}
```



http://funini.com/kei/mmap/

応用: Copy-on-writeによる高速fork(2)

- 子プロセス生成＝ページテーブル+アドレス空間記述表のコピー(≠物理メモリのコピー)
  - 生成直後は物理メモリを親子で共有
  - ただし「書き込み不可」に設定しておく
- 書き込まれたページのみ，書き込まれた時点でコピーを生成していく
- 子プロセスがやがてexecveを実行すると，子プロセスのマッピングは除去される

Agenda

- ディスクの話の残り
- メモリとディスクの簡単なまとめ
- メモリマップド・ファイル(Mmap)
  - 使い方
  - 仕組み
  - プライベートマッピングの最適化
  - mmapの利用価値

メモリマップドファイルの利用価値 (1)

- 大きなファイルの一部だけをランダムアクセスする場合
  - 実はプログラムコード(特にライブラリ)はメモリマップドファイルを利用して共有されている
    - printfやmallocが含まれるlibc.soなど
  - straceで観察してみよう

strace: Linuxでプロセスが呼んだシステムコールを表示

$ strace <コマンド名>

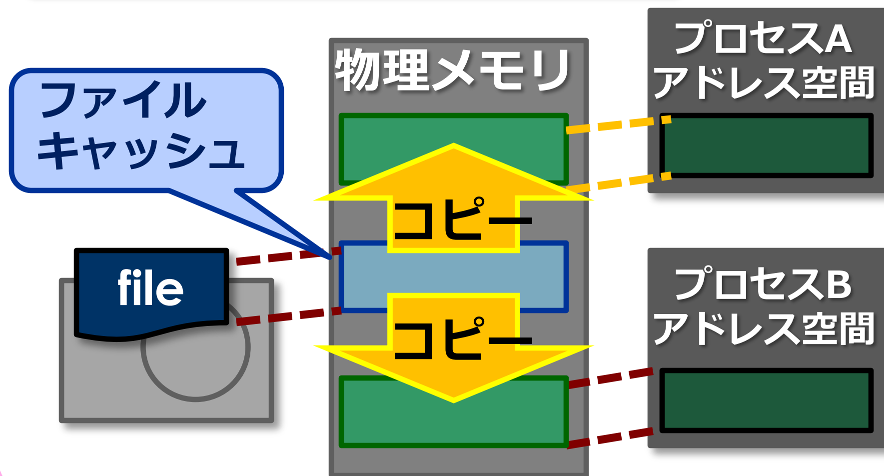
```
execve("./a.out", ["./a.out"], [/* 27 vars */]) = 0
…
open("/lib/libc.so.6", O_RDONLY)  = 3
…
mmap(NULL, 20466, PROT_READ,
MAP_PRIVATE, 3, 0) = 0x2af2c3b88000
```

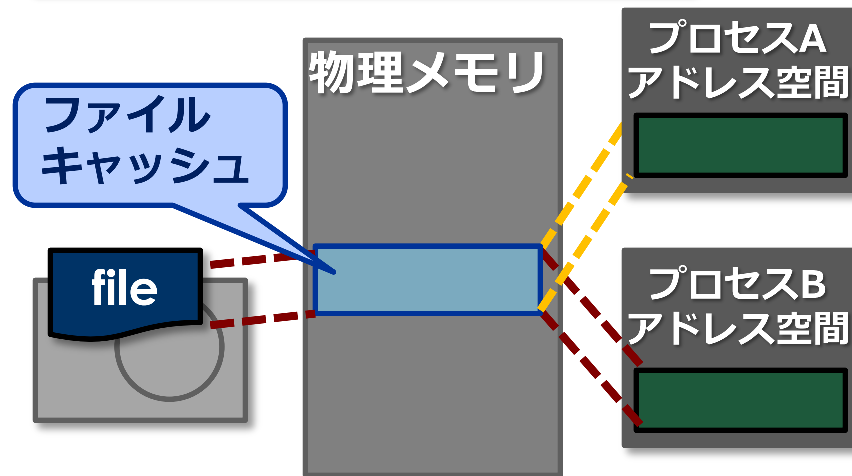
メモリマップドファイルの利用価値 (2)

- 多数のプロセスが大きなファイルをアクセスする場合
  - 共有マッピング：常に物理ページが共有される
  - プライベートマッピング：書き込まれるまで物理ページが共有される
- malloc()したメモリにread()でデータを読み込む場合よりもメモリの節約になる
  - さらに，メモリコピーが発生しない分高速

read vs. mmap

- 二つのプロセスA, Bが同じファイルをread()する場合と，mmap()する場合を比較





http://funini.com/kei/mmap/