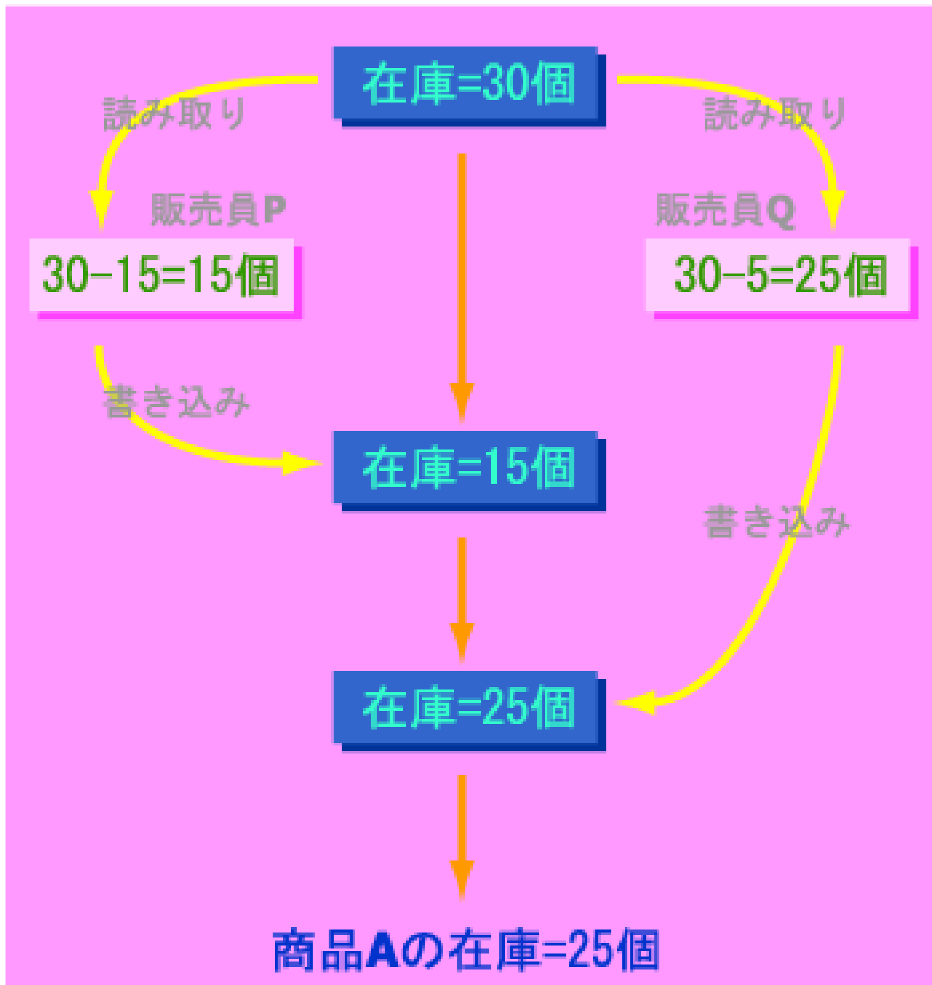
システムソフトウェア講義の概要

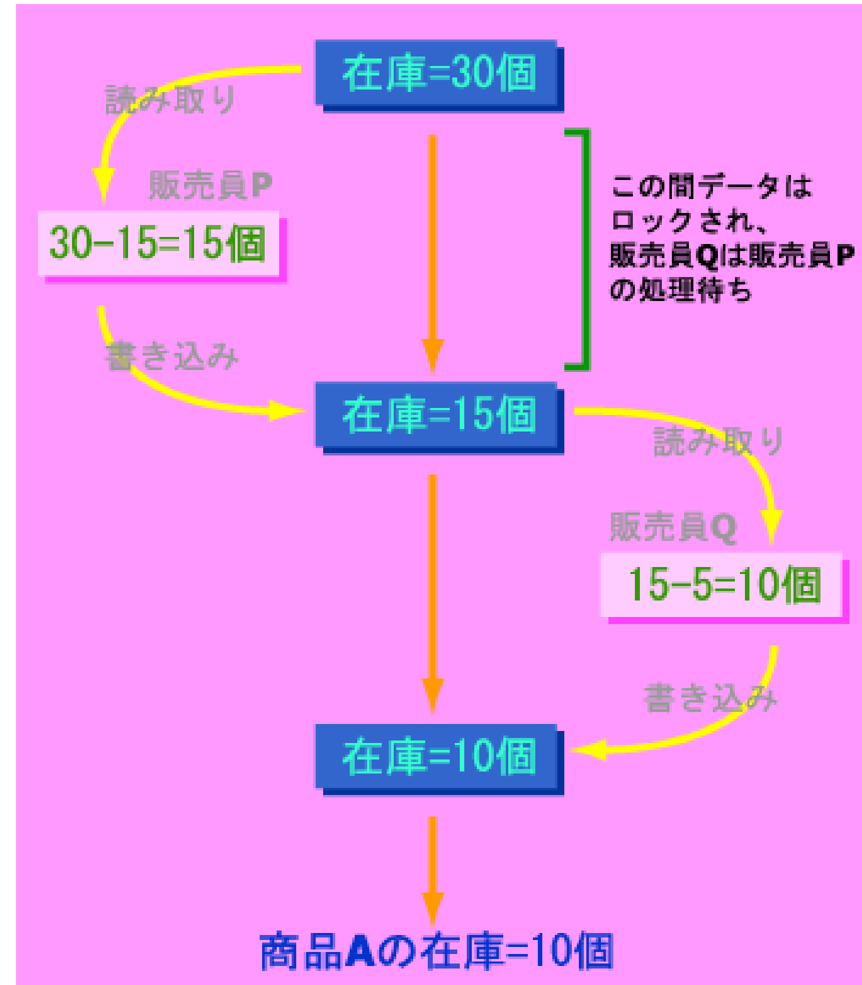
1. 計算機システムの復習：中央演算処理装置(CPU)，プログラムの実行，主記憶装置，補助記憶装置
2. 時分割処理：プロセス，スレッド，スケジューリング
3. スレッド間の排他制御：フラグ，セマフォ，モニタ，デッドロック
4. デバイス管理，HDDへのアクセス制御
5. 記憶管理：メモリ割り当て，ページング，セグメンテーション
6. 仮想記憶とファイルシステム
7. 演習問題
8. プログラミングシステムの概要，文法とそのクラス，字句解析と正規文法
9. 正規表現からの非決定性オートマトンの生成、決定性オートマトンへの変換
10. 字句解析用オートマトン生成ソフトウエアの実際
11. 構文解析と導出，文脈自由文法の構文解析法：LL構文解析
12. 文脈自由文法の構文解析法：LR構文解析
13. コンパイラ-コンパイラと構文解析の実際
14. 演習問題
15. 講義の総括と試験

並行処理における同期・通信(復習)

相互排除（排他制御）

- なぜ排他制御が必要か？





複数のスレッド(プロセス)間で同じ資源を取り合う

- これが，銀行預金の引き落としで起きたらどうだろうか？9万9千円＋千円＝10万円を引き下ろしても，残額が9万9千円残っていたら，銀行は倒産する．
- 前頁の例では，販売員がスレッド，在庫の量が資源．
- 上の例では，引き落としをする人がスレッド，口座残高が資源
- 資源にアクセスするときに，排他制御が必要．

相互排除の方法

- 割り込み禁止：スレッドの切り替わりは割り込みによって起こるので，割り込みを禁止する．
- フラグ：現在アクセス中というフラグを立てる．
- セマフォ：整数値を用いた一般化されたフラグ．排他制御以外に，有限個のリソースを複数スレッドに分配する際にも使う．
- モニタ：排他制御専用．セマフォと論理的に等価．

割り込み禁止

- スレッドの切り替わりは，ディスパッチャが行なっており，これは割り込みによって起動するので，割り込みを禁止すれば，問題も発生しない．
- しかし，この方法では排他制御を必要としないスレッドも停止させてしまうため，スレッドの待ち状態が発生する．このため，この方法は一般的には用いられない．

フラグ

- フラグ : リソースに対して現在アクセス中である，というフラグを立てる.
  - 不可分命令 : これを行うには，1命令でフラグが立たなければならない. （フラグ変更途中に他のスレッドがフラグを操作しないようにするため. ）
    - TAS命令 : Test and Set命令
    - CAS命令 : Compare and Swap命令

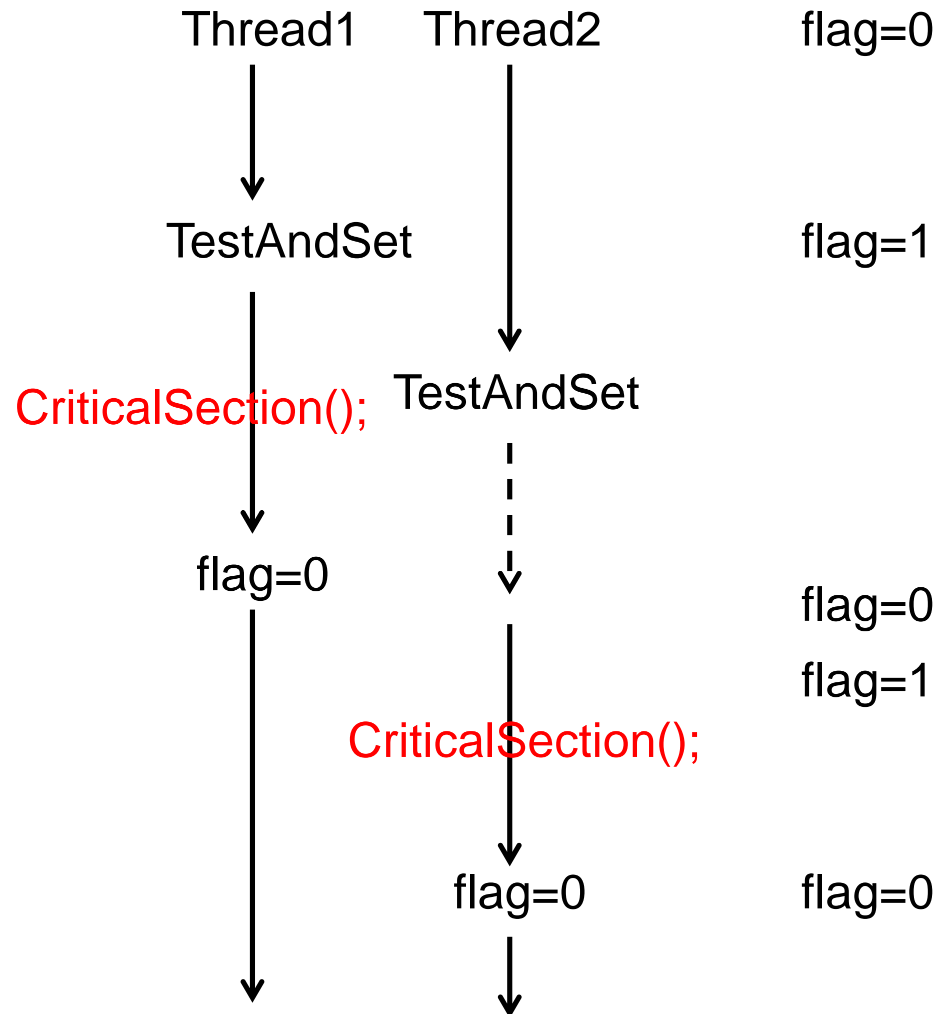
TAS命令：Test and Set命令

```
int TestAndSet(int* Vaddr){
    if (*Vaddr) return 0;
    else { *Vaddr=1;
           return 1;}
}

int flag=0;

void ThreadExecution(){
    while(!TestAndSet(&flag));
    CriticalSection();
    flag=0;
}
```

- TestAndSetは，不可分命令として実装されていなければならない．
- flagの値が先に1にセットされていれば，TestAndSet(&flag)は0になり，whileで空回りする．flagが0であれば，戻り値が1になり，whileループから抜ける．

TAS命令：Test and Set命令

```
int TestAndSet(int* Vaddr){
    if (*Vaddr) return 0;
    else { *Vaddr=1;
           return 1;}
}

int flag=0;

void ThreadExecution(){
    while(!TestAndSet(&flag));
    CriticalSection();
    flag=0;
}
```



CAS命令：Compare and Swap命令

```
int CompareAndSwap(int* Vaddr, int Expect, int New){
    if (*Vaddr != Expect) return 0;
    else { *Vaddr=New;
            return 1;}
}

int flag=0;

void ThreadExecution(){
    while(!CompareAndSwap(&flag,0,1));
    CriticalSection();
    flag=0;
}
```

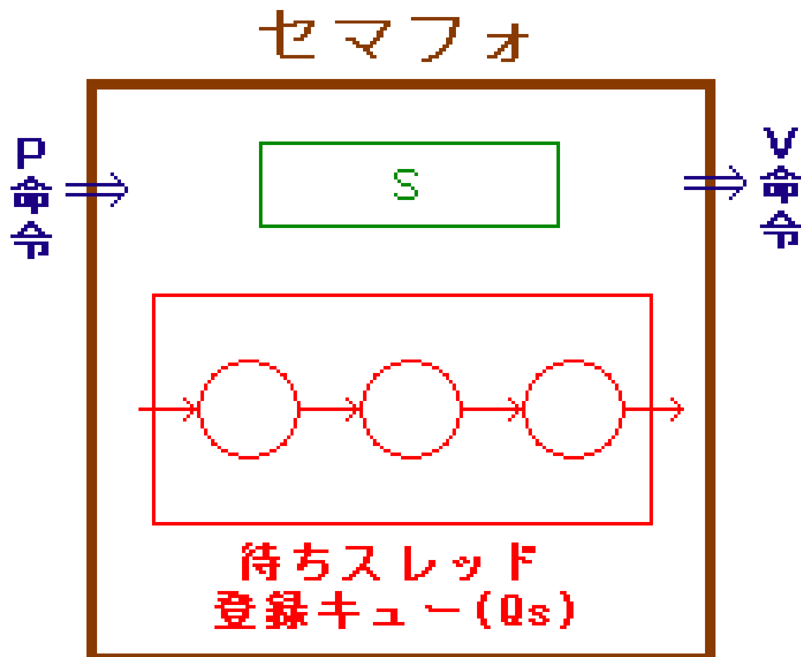
- CompareAndSwapも，不可分命令として実装されていなければならない．

ビジーウエイト

- 何度もTAS あるいは, CAS命令を実行することで, flagの値が変化するのを監視すること. スピンロック, あるいは, ビジーウエイトとも言う.
- ビジーウエイトは, CPUを無駄に消費するため, 資源の無駄遣いになる.

セマフォ

- セマフォ(Semaphore)は整数値sと事象発生待ちのスレッドを貯えるキューQsからなる.



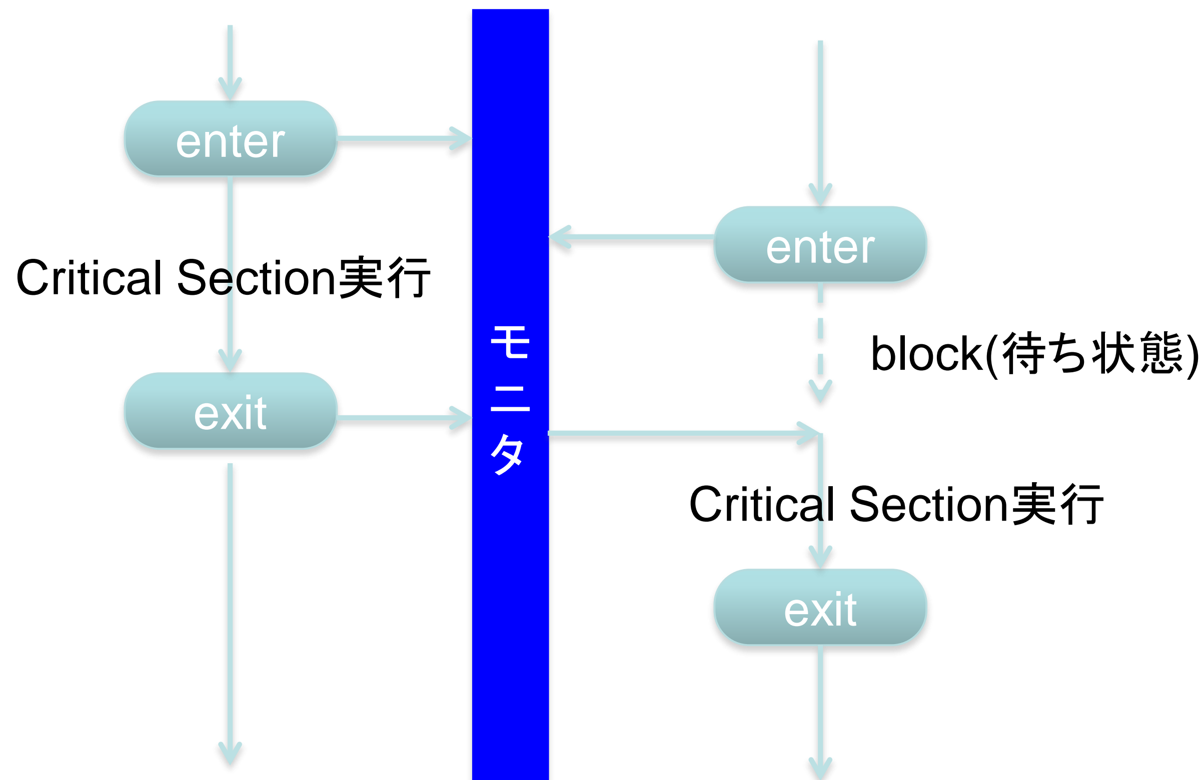

```
int s=1;   //Binary Semaphore
P(){
  if(s>0) s--;
  else{自スレッドをQsに入れる}
 }

V(){
  if(Qsが空でない){
  Qsからスレッドを取り出し実
  行可能にする
  }else s++;
}
```

セマフォ

- スピンロックではなく，サスペンドロック
- 空回りをしないので，CPU資源を無駄遣いしない．
- 起動が多少遅れる可能性がある．
- sの初期値を１以外にすれば，複数のリソースを複数のスレッドで分け合う場合にも使える．

モニタ

モニタに対して，enterし，enterできた場合に
CriticalSectionを実行．その後に，exitする．

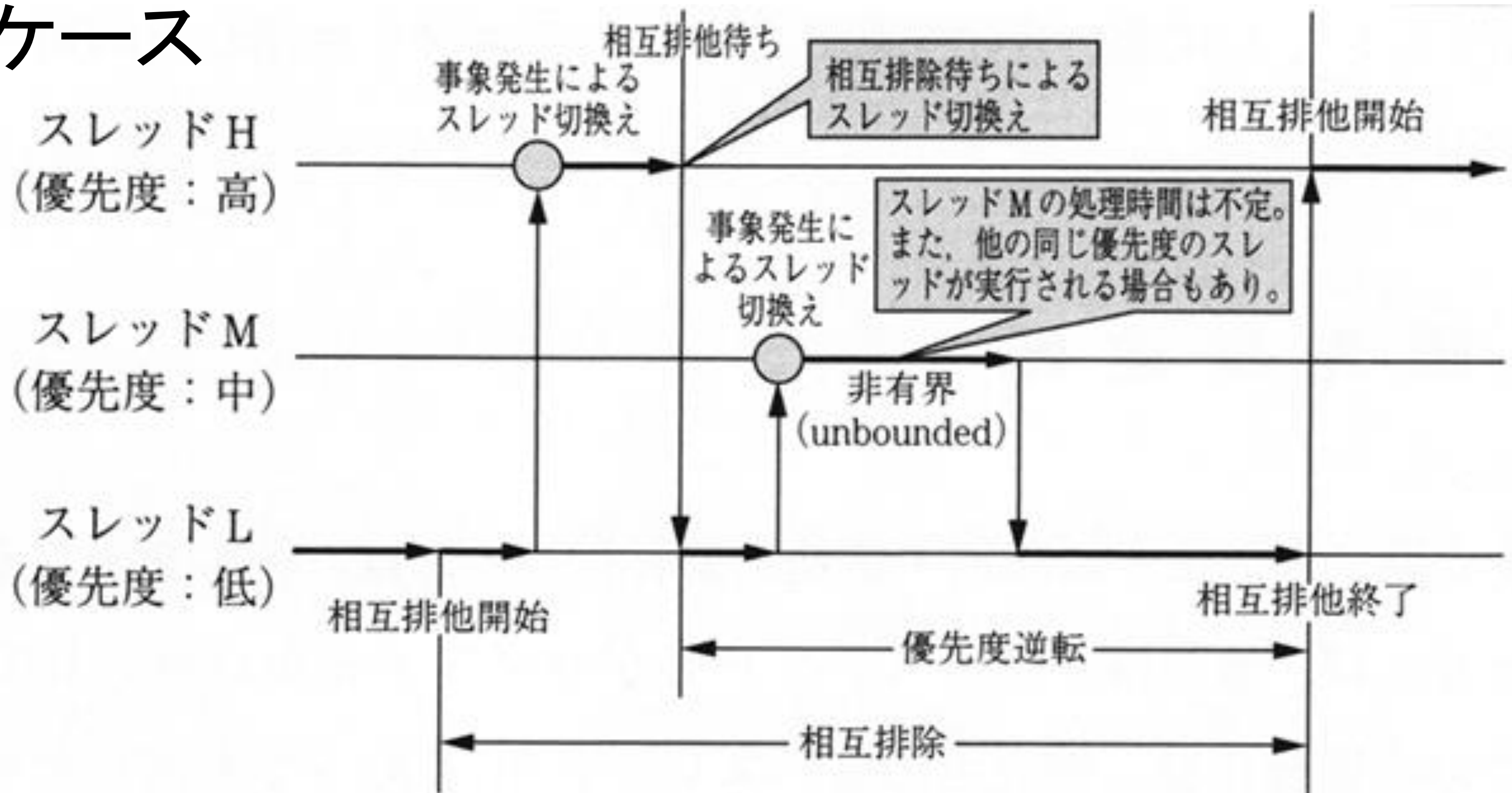


優先度逆転

- 実行の優先度が高いスレッドでも，優先度が低いスレッドがセマフォをロックするなどして，資源を占有している場合は，待たなければならない．
- このように優先度の高いスレッドが優先度の低いスレッドによって実行をブロックされる現象を「優先度逆転」と呼ぶ．
- 通常は問題にならないが，リアルタイムシステムでは問題になる．

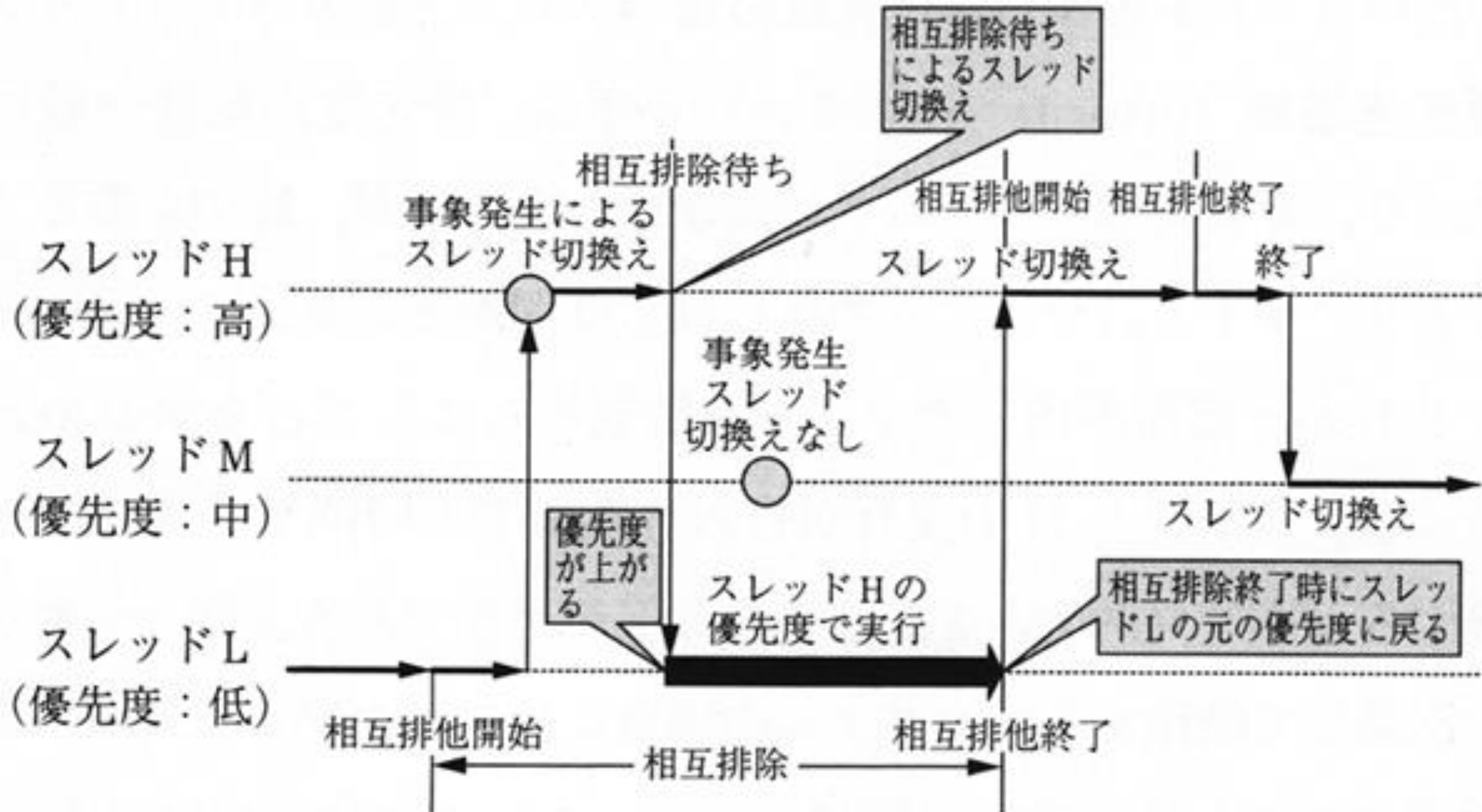
優先度逆転によって生じる問題

- 優先度逆転により，相互排除期間が伸びるケース



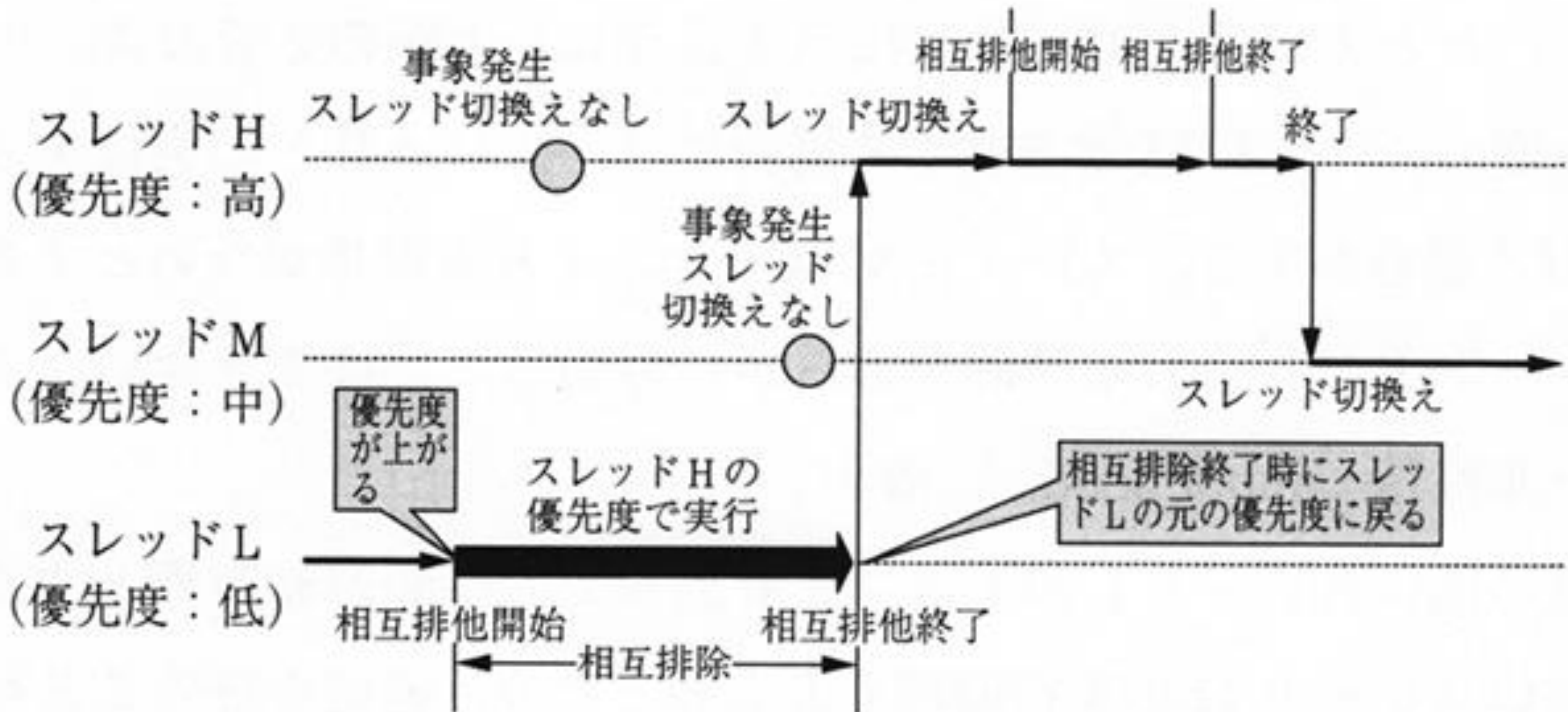
(a) 優先度逆転

優先度継承アルゴリズム



（b） 優先度継承アルゴリズム

優先度上限アルゴリズム



(c) 優先度上限アルゴリズム

デッドロック

プロセスPがその実行過程でOSに資源Rの割り当てを要求するとき，他のプロセスとの関係で，その割当が絶対に行われないという状況をデッドロックと呼ぶ.

デッドロックは，資源割付グラフ上のループとして検出することが出来る.

資源割付グラフ

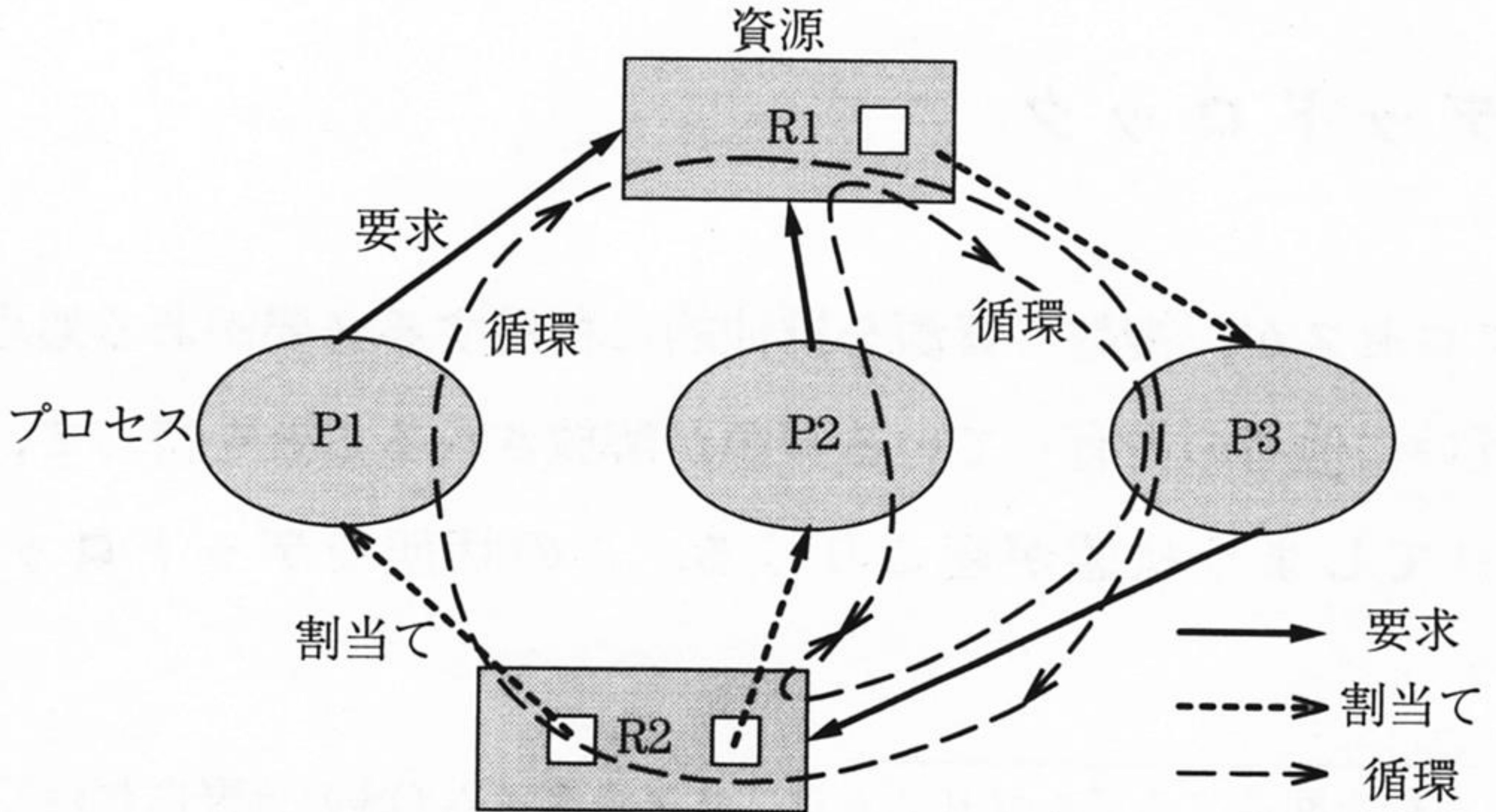


図 3.11 資源割付グラフ

問題

```
int TestAndSet(int* Vaddr){
    int OldValue;
    OldValue=*Vaddr;
    *Vaddr =1;
    return OldValue;
}

int flag=0;

void ThreadExecution(){
    while(!TestAndSet(&flag));
    CriticalSection();
    flag=0;
}
```

TestAndSetを左のように書きなおしたが，間違いがある．それはどこか？

← ！を取る．

xの値が変化するのは右？左？

```
#include <stdio.h>
void func(int x)
{ x=1;
}

int main()
{ int x=0;
  func(x);
  printf("x=%d¥n",x);
  return 0;
}
```

```
#include <stdio.h>
void func(int *x)
{ *x=1;
}

int main()
{ int x=0;
  func(&x);
  printf("x=%d¥n",x);
  return 0;
}
```

デバイス管理

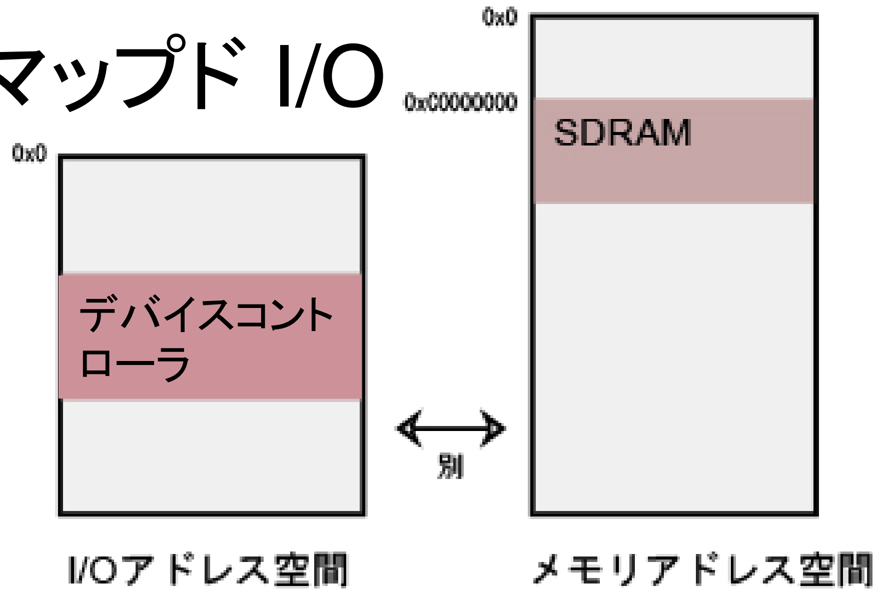
コンピュータ

デバイス
コントローラ

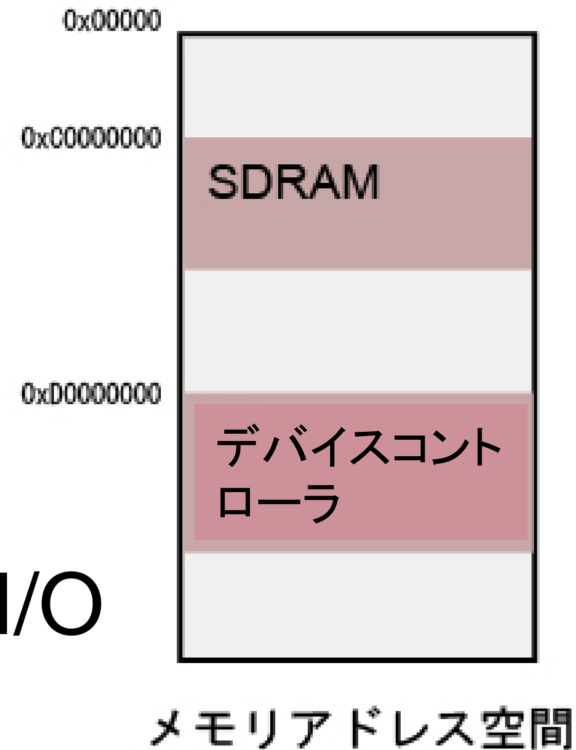
デバイス

入出力方式

- ポートマップド I/O

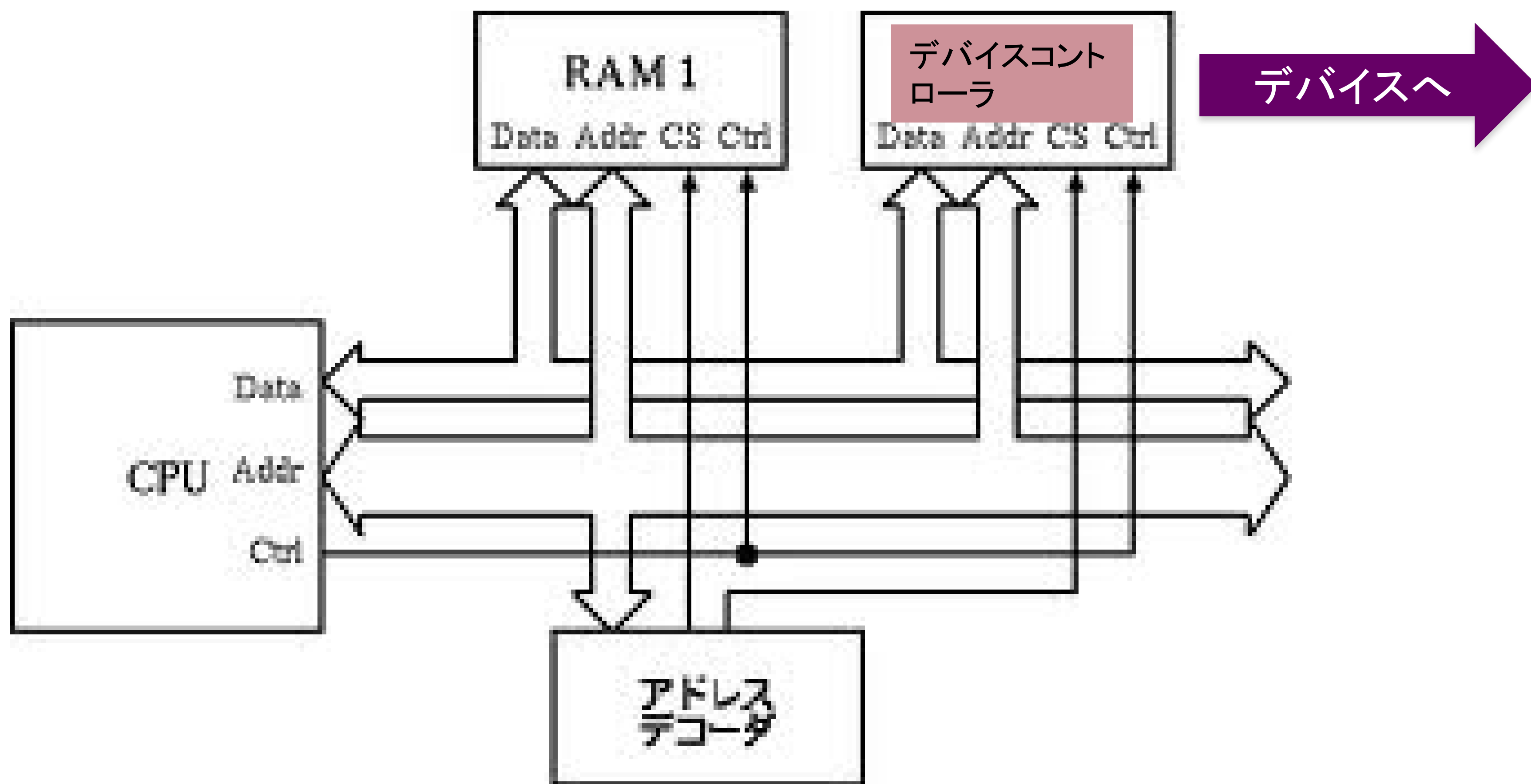


- メモリマップドI/O



メモリマップド I/O

- アドレス

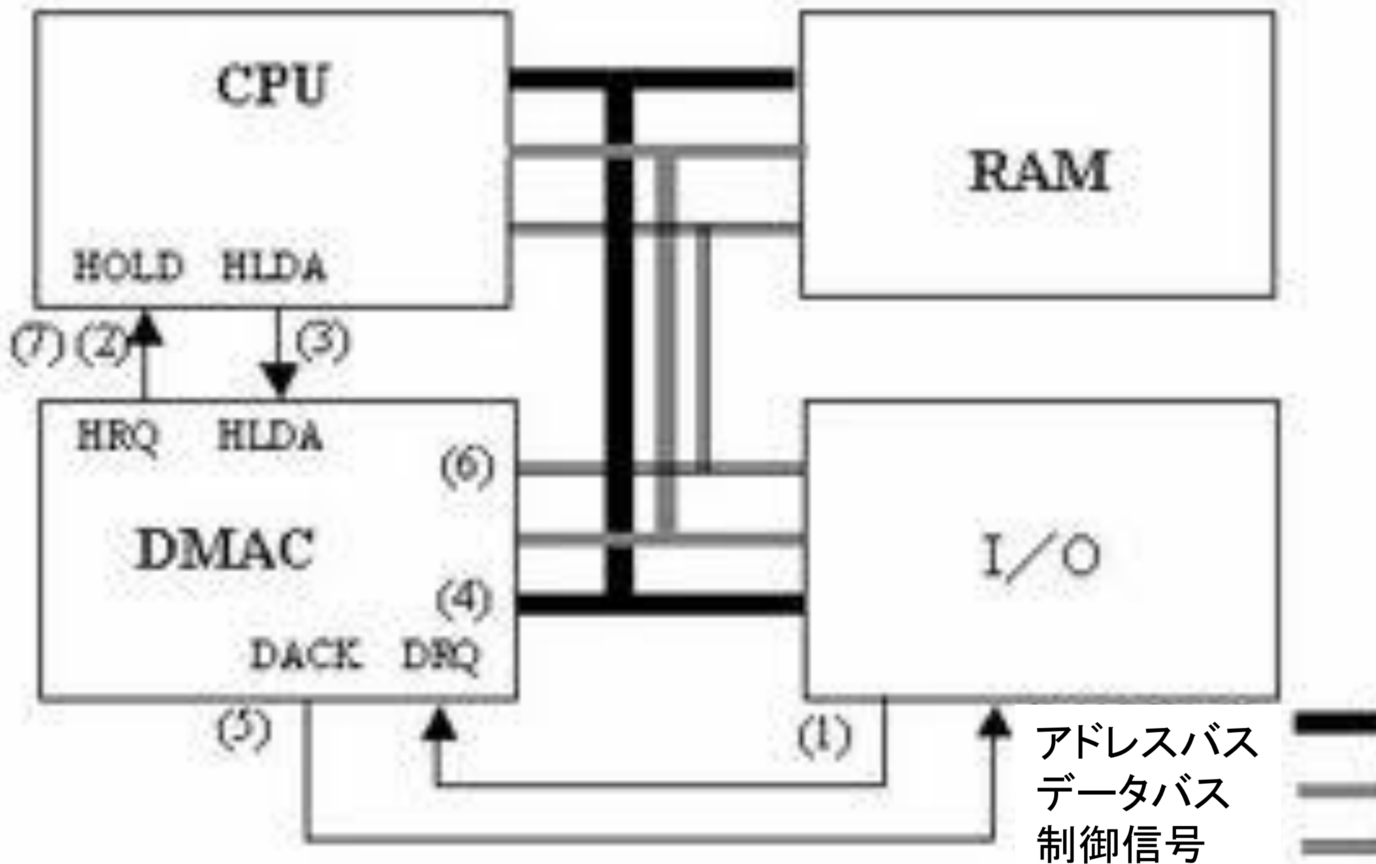


デバイス↔メモリ間のデータ転送

- CPUが介在してデバイスとメモリの間のデータのやり取りをする．（遅い）
- デバイスコントローラとメモリ間で直接データを交換する方式（速い）【DMA：ダイレクトメモリアクセス】DMAコントローラが，デバイスコントローラ側にある場合，バスマスタ転送とも呼ぶ．

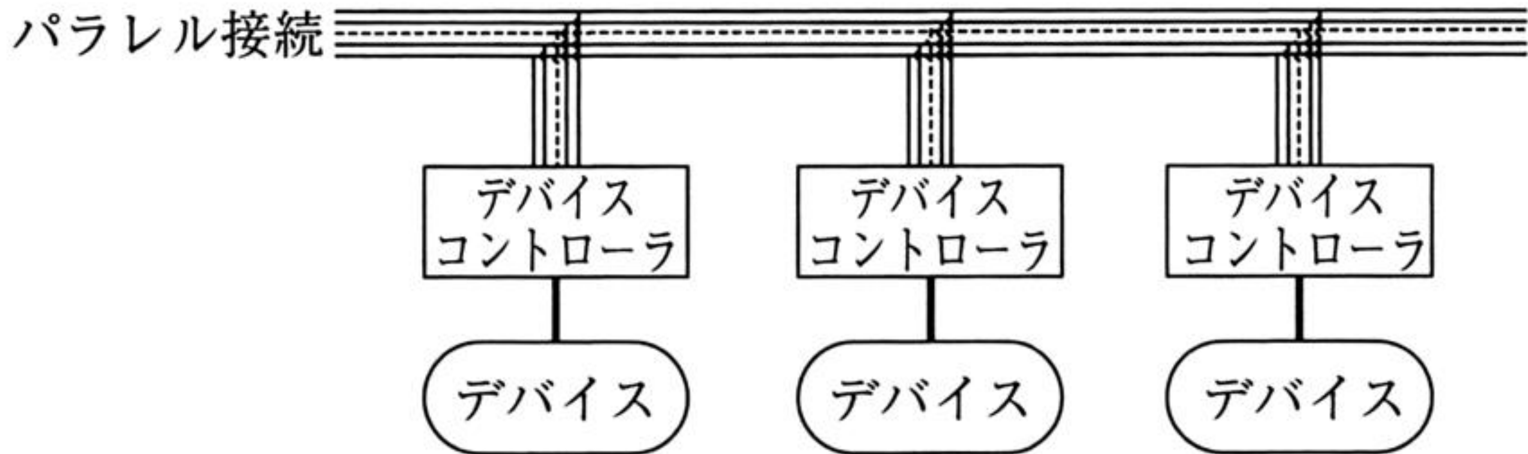
DMA転送

- DMAコントローラ(DMAC)がデータを送る



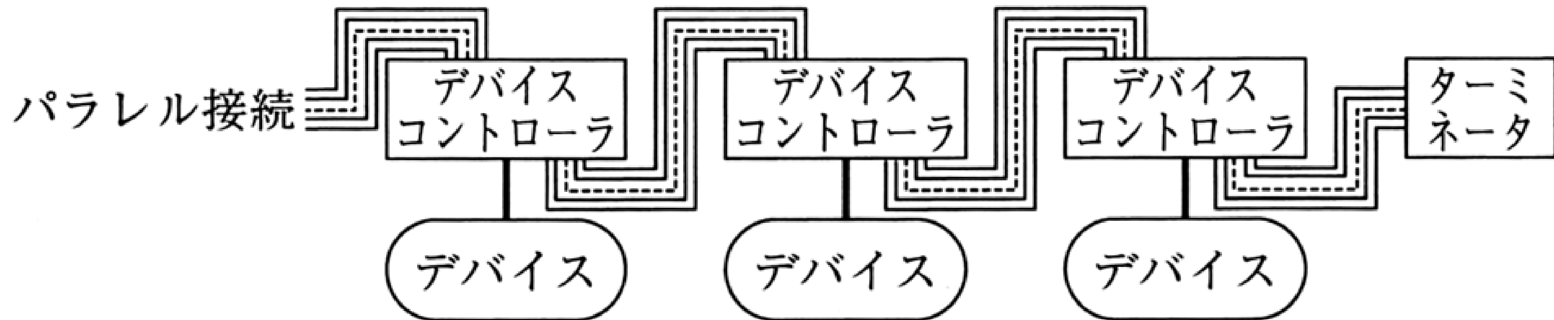
バス

- CPUと，主記憶や周辺機器を接続するための汎用データ通信路.
  - アドレスバスで，デバイスとアクセスするデータを指定し，データバスを介してデータのやり取りをする.

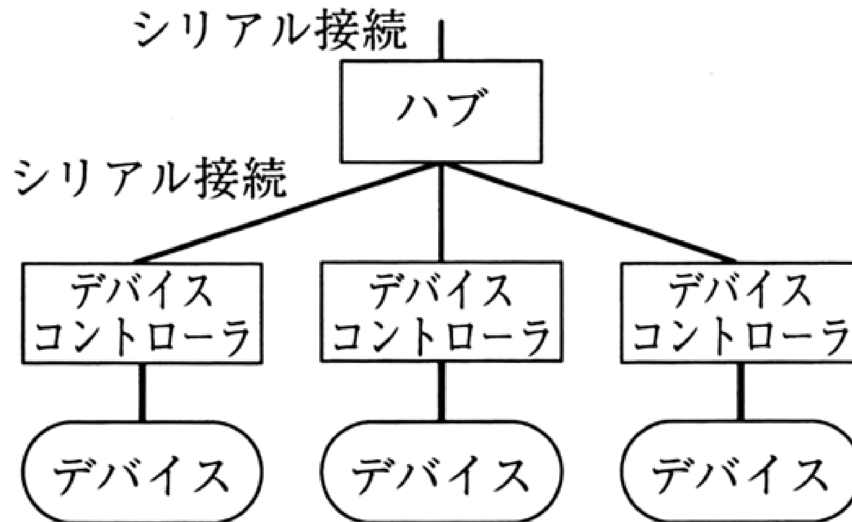


（a） 分岐方式

その他のバス



（b） デイジーチェーン方式



（c） ハブ方式

コンピュータ全体のバス

- 目的に応じて，異なるバスが用いられる.

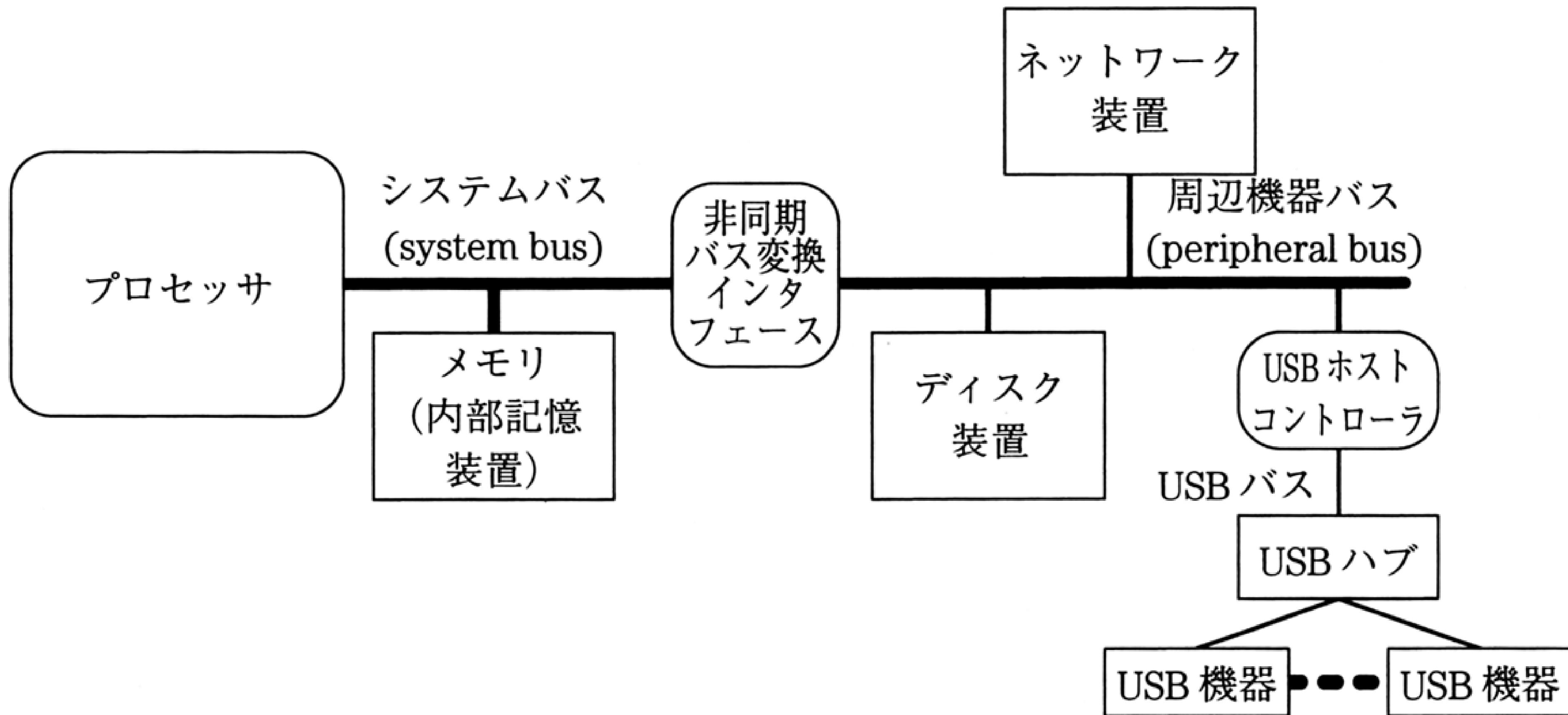


図 4.3 階層化されたバスの例

デバイスをコントロールする

- デバイスドライバ:デバイスを抽象化して，統一的なインタフェースで様々なデバイスコントローラを扱うためのOSの機能.
- 例: open, close, read, write, fseek 等

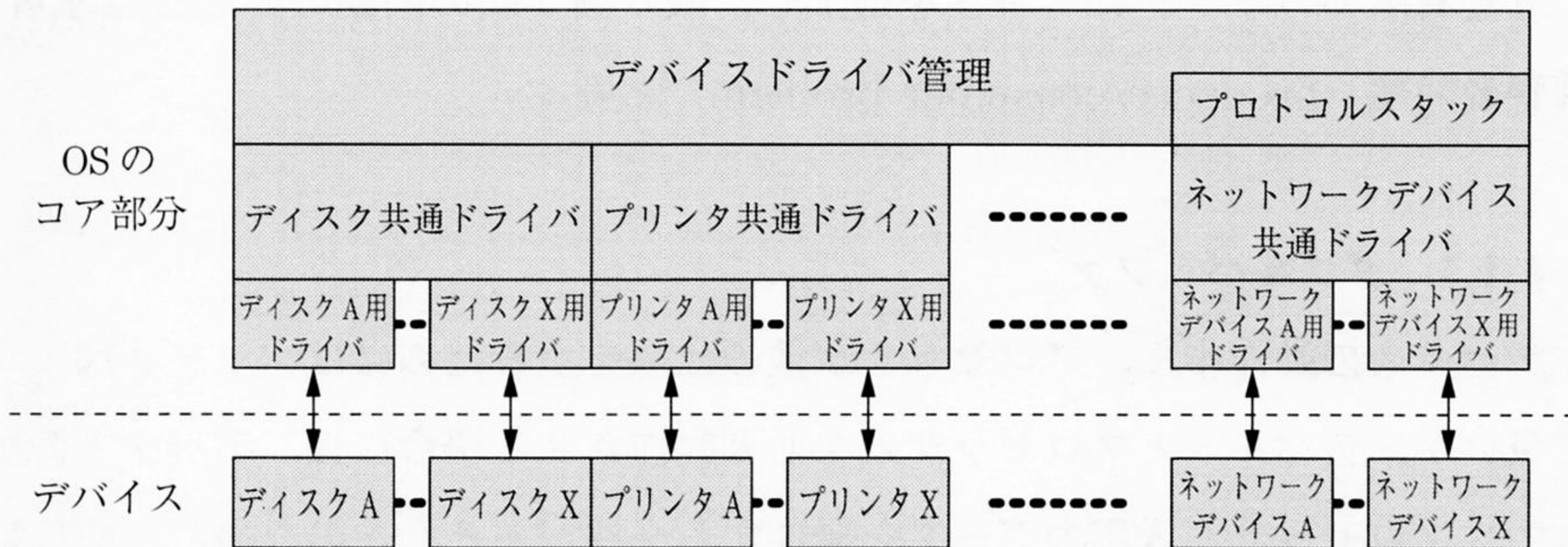


図 4.4　デバイスドライバの階層構造

デバイスの分類

- ブロック型デバイス

　まとまった大きさのデータ単位で，入出力を行うデバイス．（DMAがよく用いられる）

HDD, SSD, 磁気テープ，DVD/CD等

- キャラクタ型デバイス

　1バイトずつ，入出力を行うデバイス．（DMAは用いられない．）

キーボード，マウス，

- パケット型デバイス

構造化されたデータを交換：USBなど

関数のポインタ

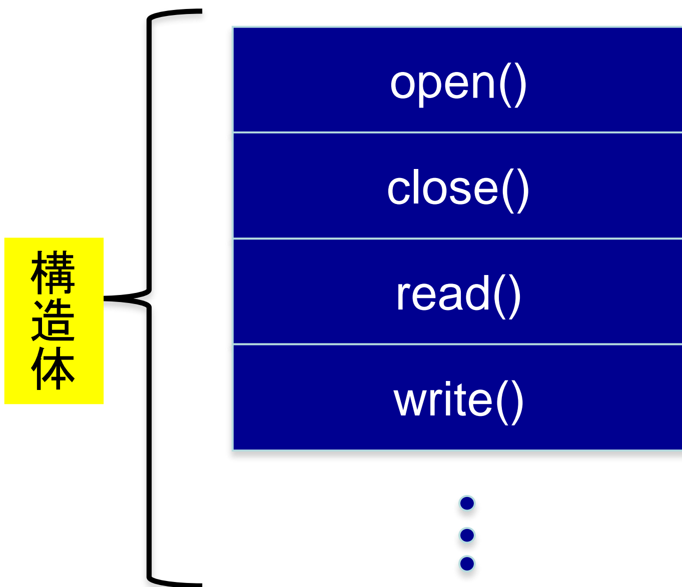
```
#include <stdio.h>
void func1(int *x)
{ *x=1;
}
void func2(int *x)
{ *x=2;
}
```

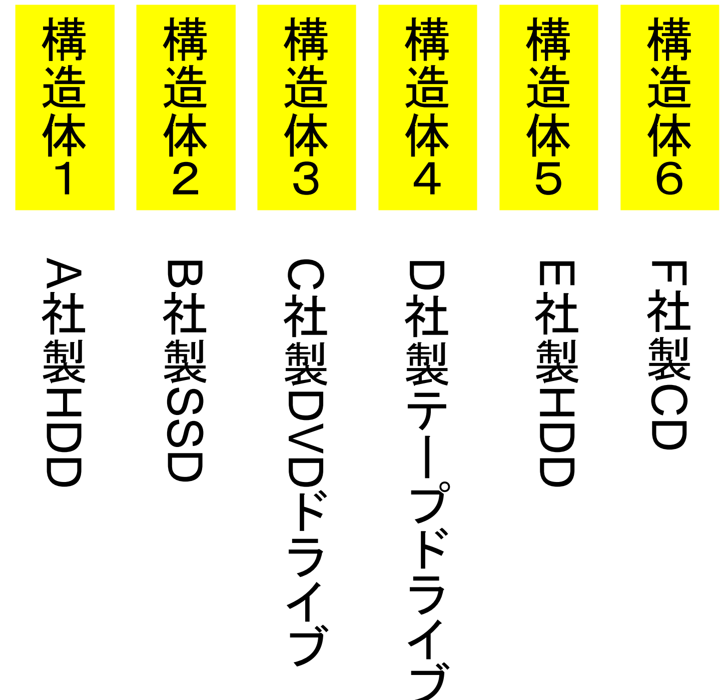
関数や手続きは，ポインタ変数に代入することが出来る！ポインタ変数に代入された関数を呼び出すこともできる.
これを利用すれば，open(), close(), read(), write(),などの関数をデバイスごとに用意しておいて，切り替えて使うことができる.

```
int main()
{ int x=0, i;
  void (*func[2])(int *);
  func[0]=func1;
  func[1]=func2;
  for (i=0; i< 2; i++){
     func[i](&x);
     printf("x=%d¥n",x);
  }
  return 0;
}
```

関数のポインタの構造体の配列

- 1種類のデバイスについては，下記の構造体で表現可能．
- 多数のデバイスについては，構造体の配列で表現可能．





制御する対象毎にアクセスの方法を用意しておくのがデバイスドライバ

- キャラクタ型デバイスやブロック型デバイスでは，下記の関数群が異なる．

```
twada$ ls –l /dev
crw-------     1 twada  staff     0,   0 10 19 17:54 console
crw-rw-rw-  1 root   wheel     11,   1 10 19 17:54 cu.Bluetooth-Modem
crw-rw-rw-  1 root   wheel     11,   3 10 19 17:54 cu.Bluetooth-PDA-Sync
brw-r-----     1 root   operator 14,   0 10 19 17:54 disk0
brw-r-----     1 root   operator 14,   1 10 19 17:54 disk0s1
brw-r-----     1 root   operator 14,   2 10 19 17:54 disk0s2
…
```

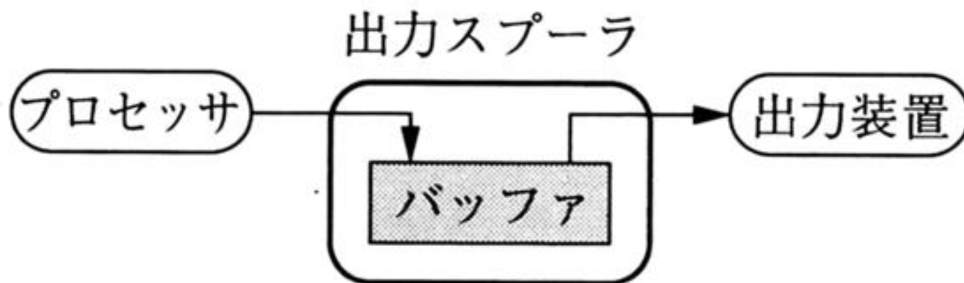
構造体

open()
close()
read()
write()
⋮

先頭のcまたはbはキャラクタ型デバイスかブロック型デバイスかを示している．パーミッション，所有者，グループ以降の番号がデバイスの種類を表すメジャーナンバー，それが何個目のデバイスかを表すマイナーナンバーになっている．

バッファリング

- CPUとデバイスの速度差を埋めるためのメモリ上のキュー(FIFO)をバッファと呼ぶ.
- コンピュータにとっての出力用のバッファは，プリンタのバッファのようなものがある.



（a） 出力スプーリング

- 入力用のバッファは入力用であり，HDDやカメラなど，大量のデータをやり取りする際に重要.
- 入力用バッファ

データが入ってきている時に読み取りをすると，読み取りに失敗することがある.

  – ダブルバッファリング
  – バッファプール
  – リングバッファ

ダブルバッファリング

- プロセッサ(CPU)が，一方のバッファを読み取っている最中に，もう一方のバッファに(DMA)でデータが入り，これを交互に繰り返す．

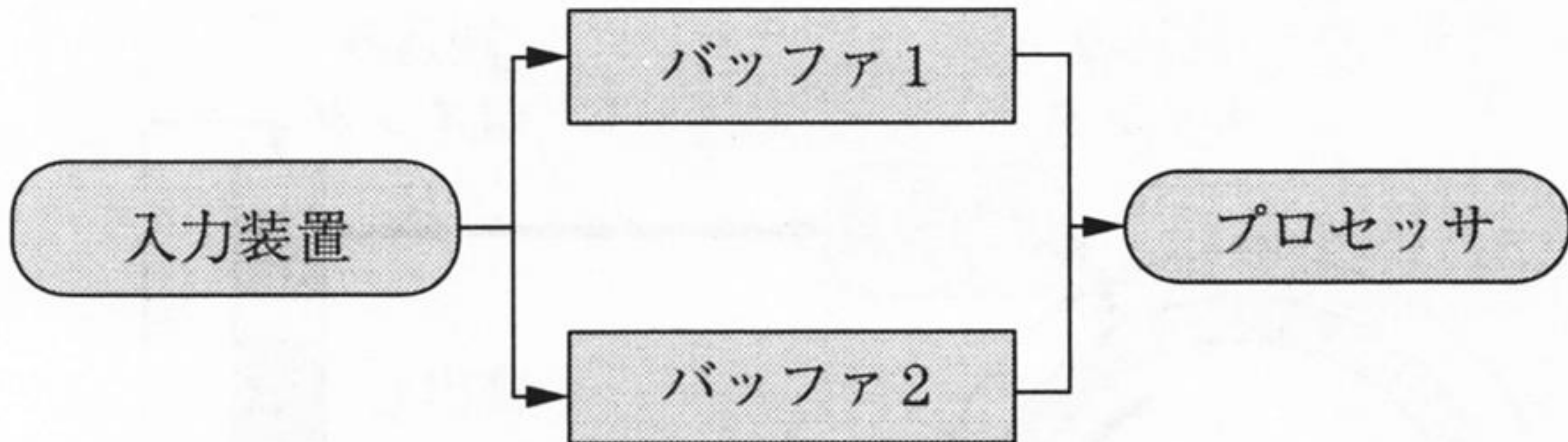
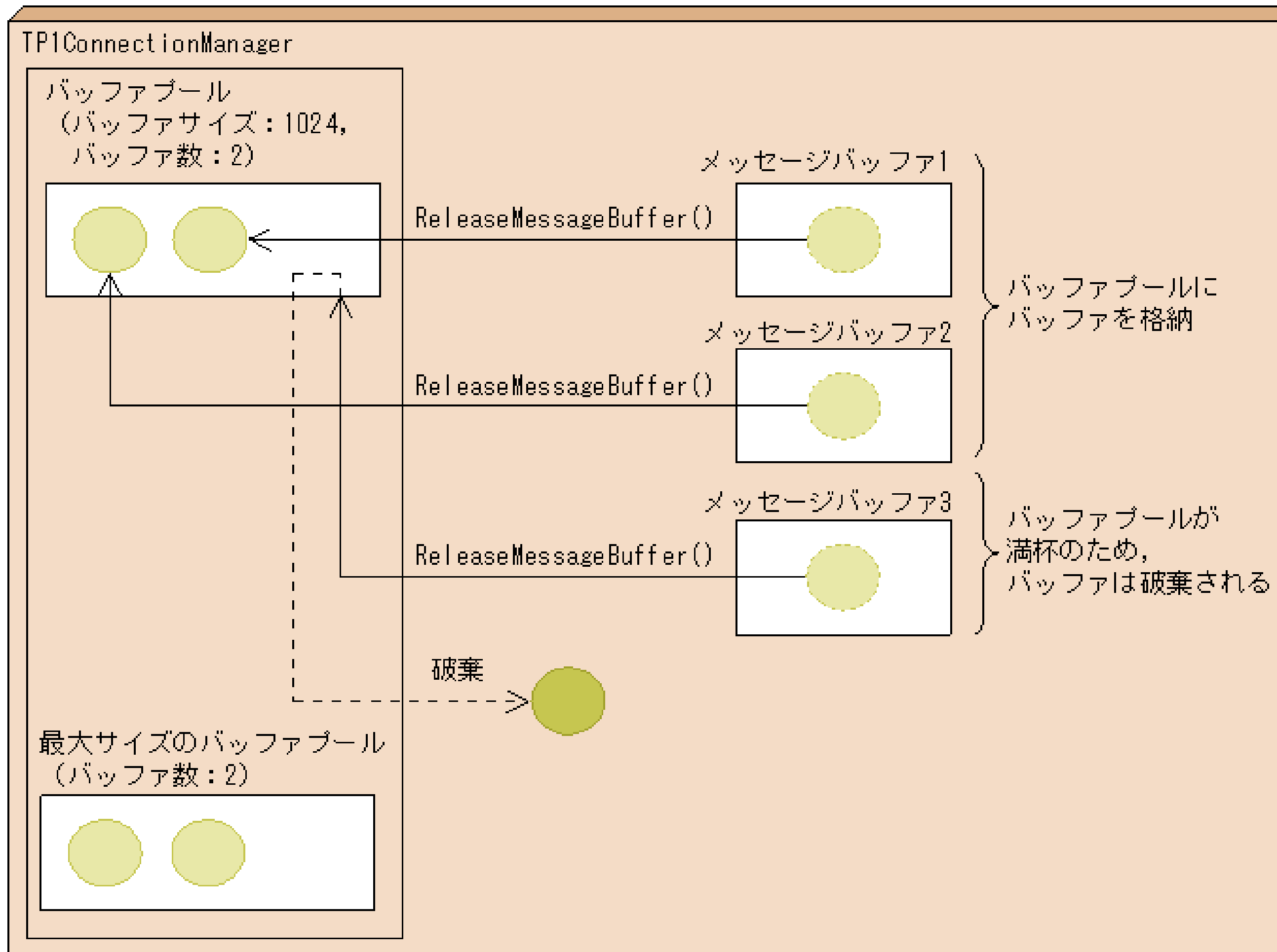


図 4.5　ダブルバッファ

バッファプール



リングバッファ

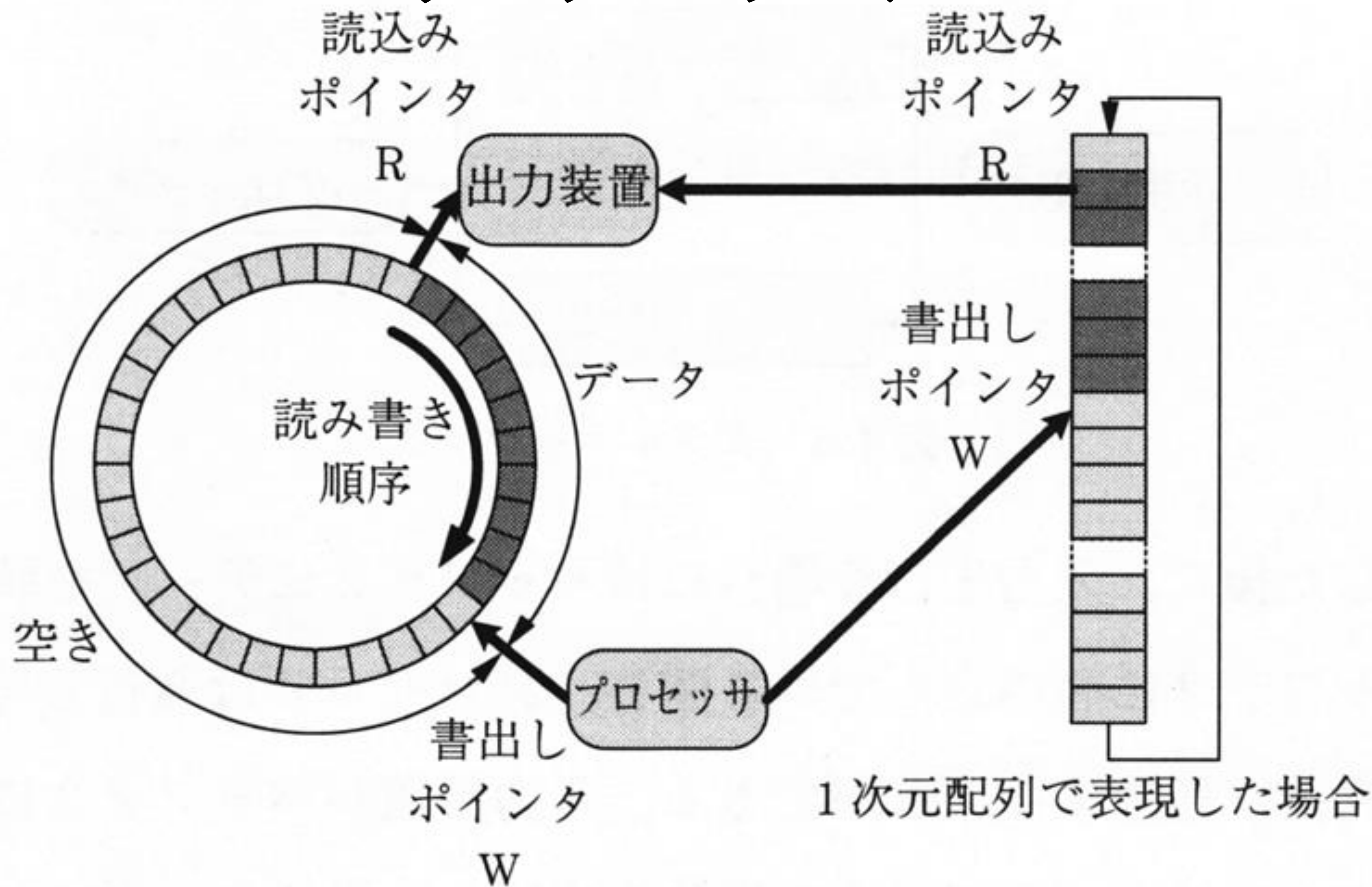
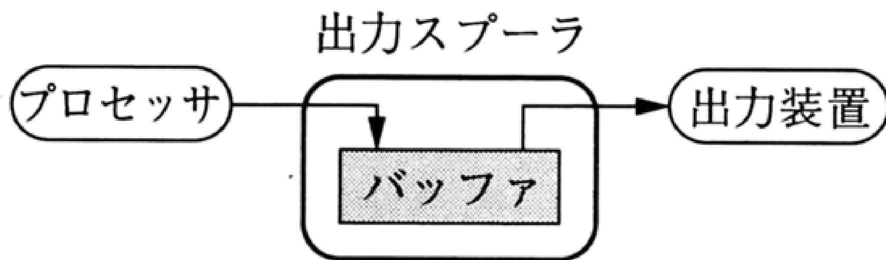


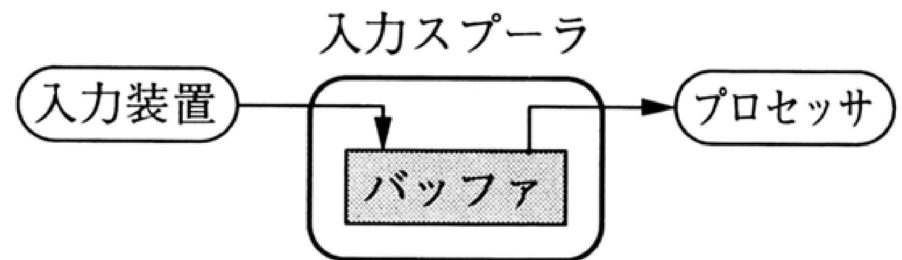
図 4.6　リングバッファ

バッファリングとスプーリング

- バッファリングではCPUとデバイスがおおまかにでも
同期を取る必要がある.
- スプーリングはデータを一旦蓄積するサーバ(スプー
ラ)がバッファリングを行う



（a） 出力スプーリング



（b） 入力スプーリング

Hard Disk Drive

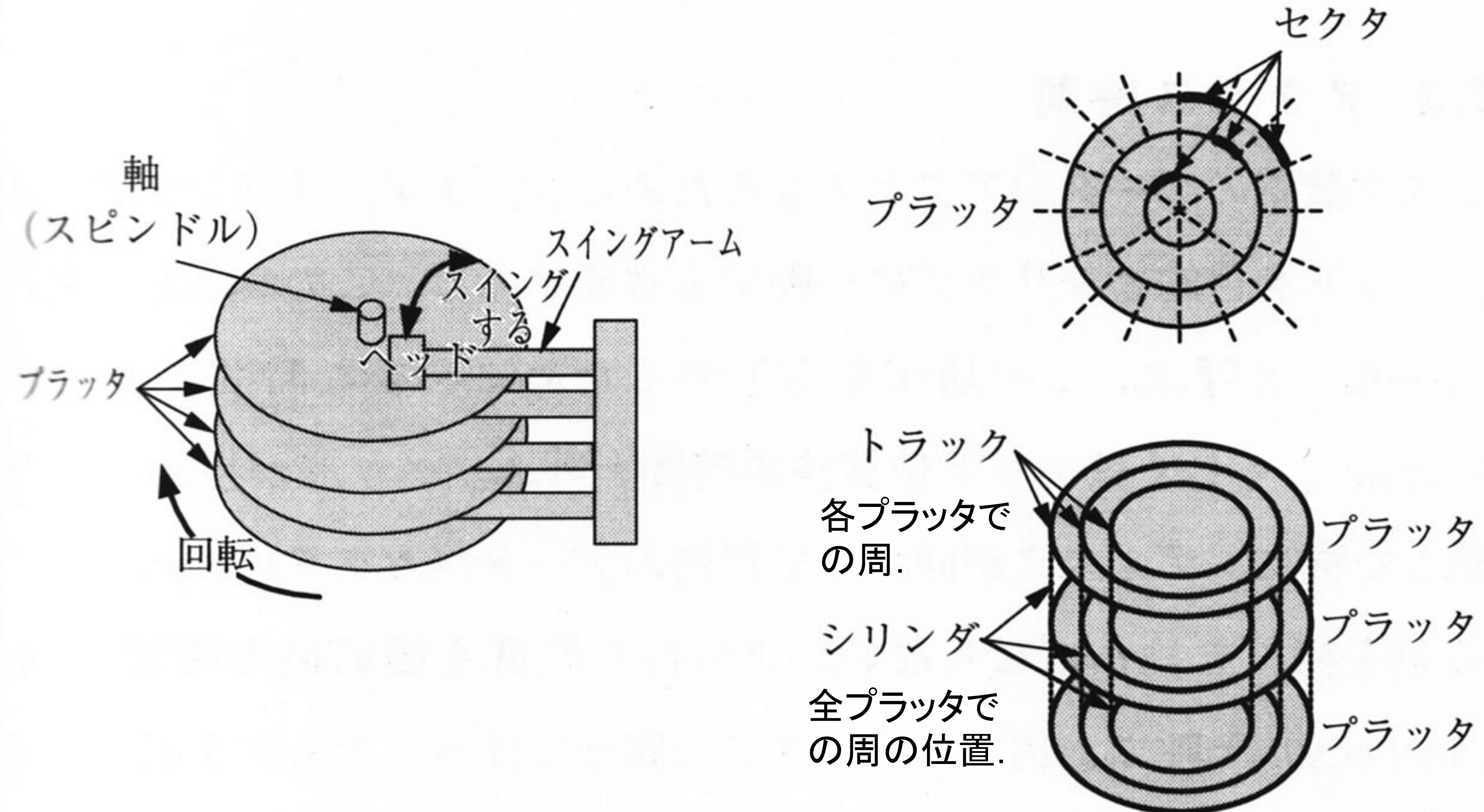


図 4.8　移動ヘッド式ハードディスク装置

動作中のHDD

HDDの速度指標

- シーク: ヘッドを目的のシリンダに移動する
- 回転待ち: 目的のセクタがヘッドの位置に来るまで
- 転送: データの読み取り.

- シーク時間＋回転待ち時間＋転送時間

  数ms　　　数ms　　　　　　数十μs/KB

HDDの速度指標（計算例）

7200rpm（rotation perminute：回転/分）のHDDは60秒/7200=8.33ms

- 回転待ち時間　8.33/2=4.17ms

回転待ち時間は回転数で決まる.

トラック1周のデータ量が1024KB，転送時間を測る際のデータ量が，8KBであった場合，

- 転送時間　8/1024 * 8.33ms = 0.0651ms= 65.1μ秒

　転送時間は回転数とトラックあたりの記録密度で決まる.

ディスクアクセススケジューリング

多数のプロセス，スレッドが動作している場合，ディスクに対するアクセス要求にどう応えるかで，コンピュータの性能が変化する．

- FCFS:先着順
- SSTF:最短シーク時間
- SCAN
- C-SCAN
- LOOK
- C-LOOK

先着順:FCFS

- 先着順に，ディスクへのアクセス要求に応える.
  - 利点：公平である.
  - 欠点：シーク時間が長くなる可能性がある.

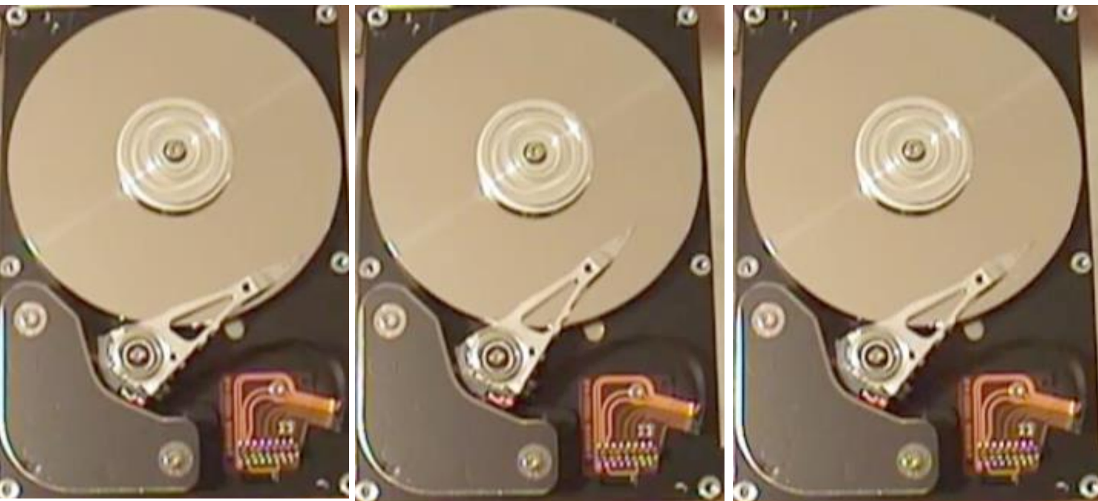
この動作を交互に繰り返すとシーク時間が伸びる

t

最短シーク時間：SSTF

- ヘッドの移動量が小さい順に，ディスクへのアクセス要求に応える．
  - 利点：シーク時間が短い．
  - 欠点：離れたシリンダへのアクセスが待たされる可能性がある．（飢餓状態）

外周へのアクセス要求が続くと

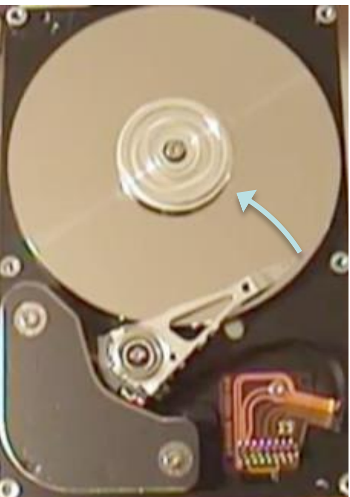


内周への要求が待たされる

t

SCAN

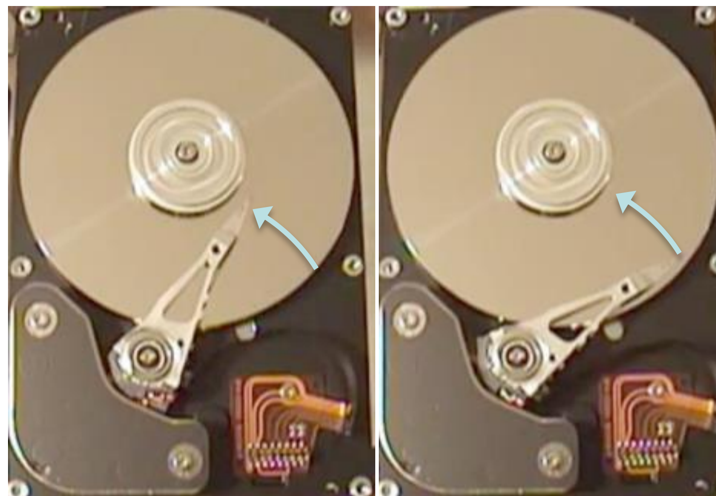
- 外周から内周，内周から外周，という時間とともに変化する位置に近い順序で，ディスクへのアクセス要求に応える.
  - 利点：飢餓状態が発生しない.
  - 欠点：端に行った直後，折り返してもアクセス要求はない.



t

Circular SCAN: C-SCAN

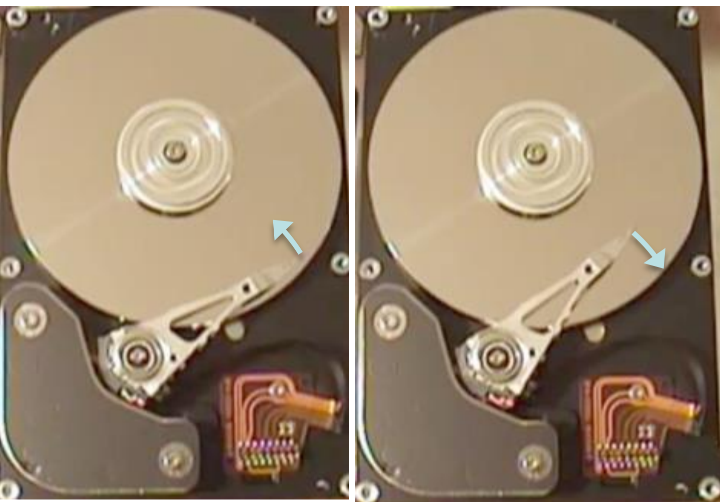
- 外周から内周まで行った時に，折り返さずに，外周から内周という順序で繰り返すSCAN.
  - 利点：折り返しがないため，平均的なアクセスが速い
  - 欠点：アクセス要求がない位置までSCANする.



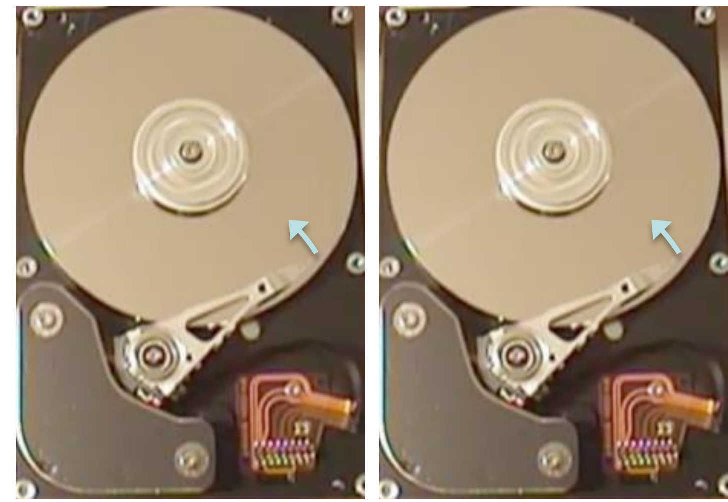
t

LOOK と C-LOOK

- アクセス要求がない位置まで見ないSCANとC−SCAN



t



t

各アルゴリズムでのアクセス順

- 下記ディスクアクセス待ち行列において，数字はトラック番号．最初のヘッドの位置はトラック50に居る．LOOK, C-LOOKでは最初小さい番号に向かって移動する．

先頭 … 末尾

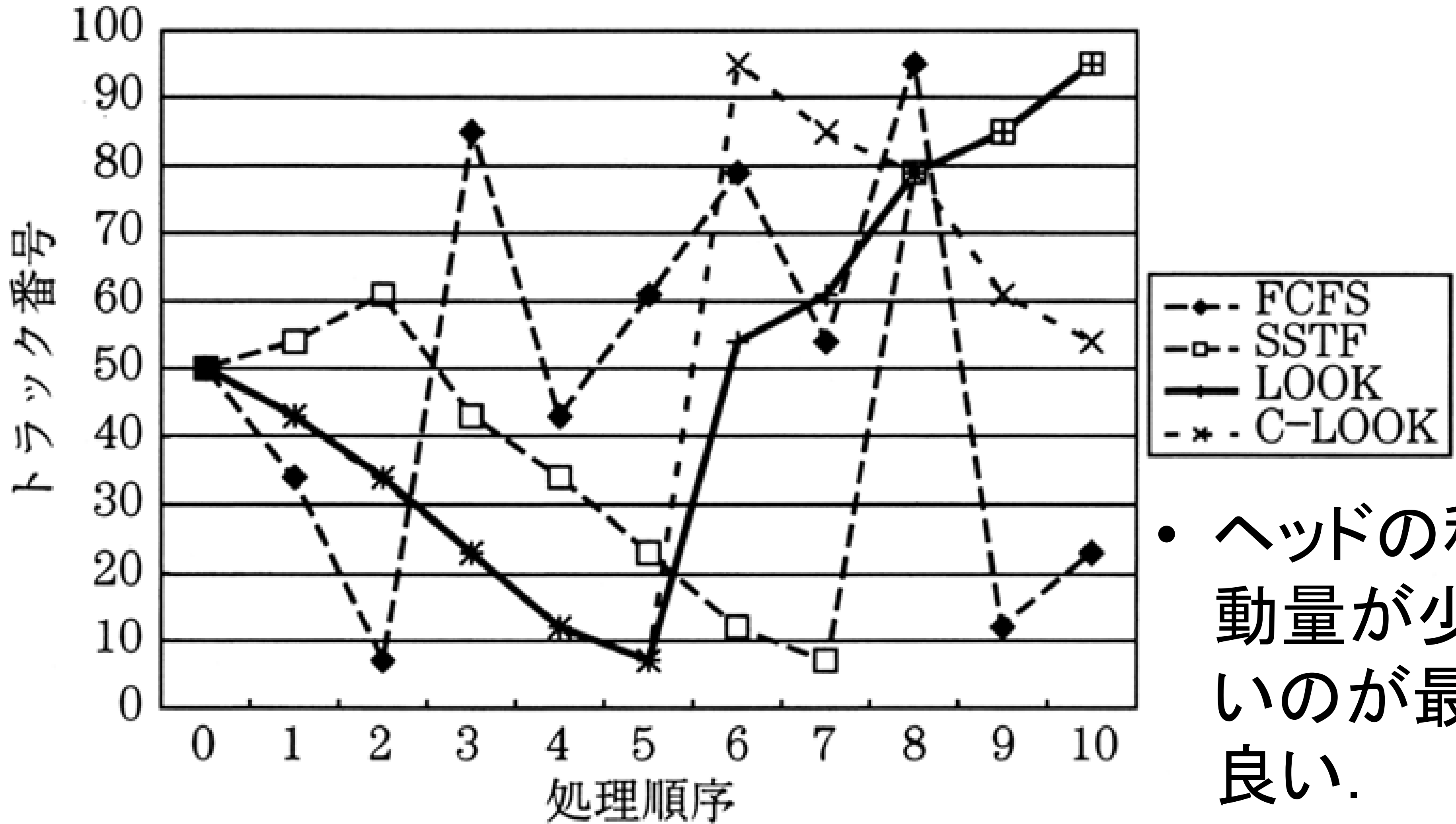
待ち行列	34	7	85	43	61	79	54	95	12	23

（a） ディスクアクセス待ち行列

処理順序	0	1	2	3	4	5	6	7	8	9	10
FCFS	50	34	7	85	43	61	79	54	95	12	23
SSTF	50	54	61	43	34	23	12	7	79	85	95
LOOK	50	43	34	23	12	7	54	61	79	85	95
C-LOOK	50	43	34	23	12	7	95	85	79	61	54

（b） スケジューリング結果

処理順序グラフ



- ヘッドの移
動量が少
いのが最
良い.

（c） 処理順序グラフ

問題

- 下記のHDD1, HDD2の転送速度はどちらがどれぐらい速いか？
  - HDD1: 1536KB/TRACK,　7200rpm
  - HDD2: 2048KB/TRACK,　5400rpm
- 下記のディスクアクセス待ち行列において，ヘッドの初期位置を50，LOOK, C-LOOKでは最初トラック番号が小さくなる方向に移動するとして，FCFS,SSTF,LOOK,C-LOOKの処理順序グラフを描きなさい．

	先頭									末尾
待ち行列	34	12	70	44	55	91	76	84	8	21