受講にあたっての注意事項

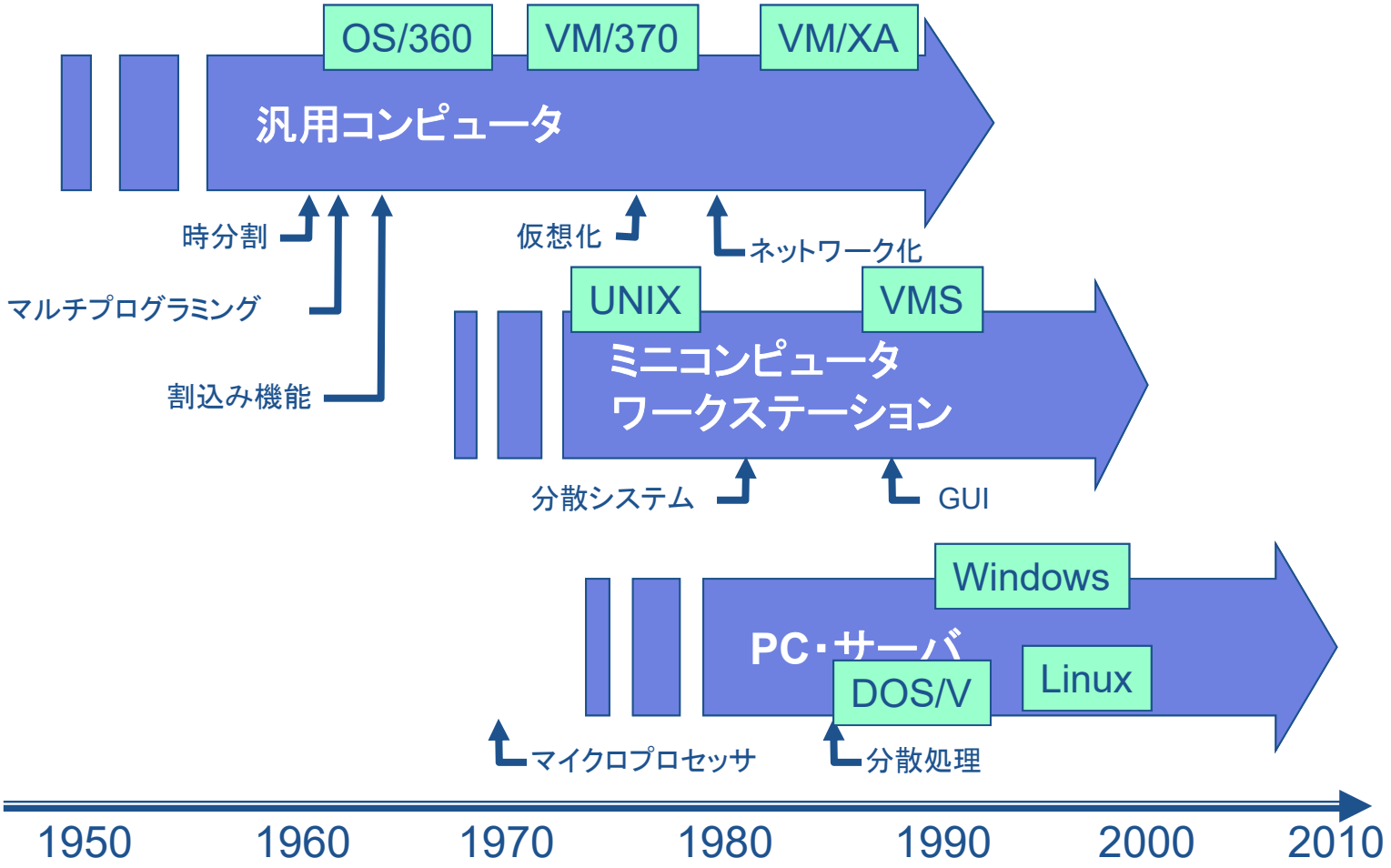
本講義の全部あるいは一部の録画・録音および複製ならびに再配布を、厳に禁じます。

大阪大学 基礎工学部
SCHOOL OF ENGINEERING SCIENCE

オペレーティングシステム
3章 メモリ管理
3.1節 領域の管理ーメモリの空間的管理ー

大阪大学大学院情報科学研究科
村田正幸
murata@ist.osaka-u.ac.jp
http://www.anarg.jp/

OSの歴史

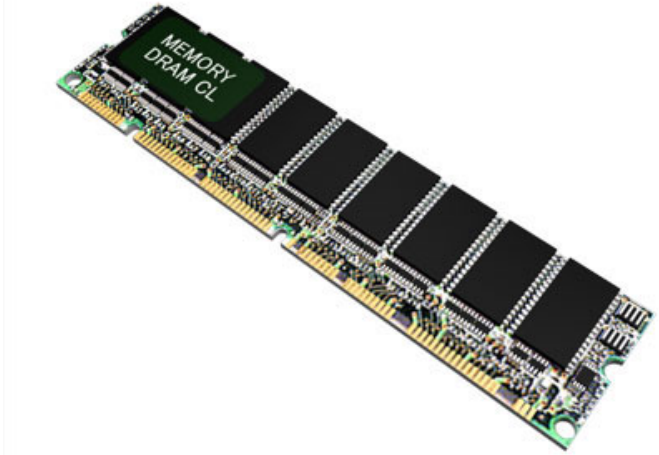

OS/360
VM/370
VM/XA
汎用コンピュータ
時分割
マルチプログラミング
割込み機能
仮想化
ネットワーク化
UNIX
VMS
ミニコンピュータ
ワークステーション
分散システム
GUI
Windows
PC・サーバ
DOS/V
Linux
マイクロプロセッサ
分散処理
1950
1960
1970
1980
1990
2000
2010

「なぜこんなものが必要なの」と思ったときは思い出そう

- オペレーティングシステムの目的
  - 資源の有効利用を図り、ユーザ間の資源の共有を実現する
    - 資源：プロセッサ、メモリ、ディスク、プリンタなどのハードウェア資源と、プログラムやデータなどのソフトウェア資源
  - ハードウェアを正しくかつ効率よく操作するために、ハードウェア資源をファイルなどの概念に「抽象化」し、それらに対する操作を提供する
    - 抽象化によって、単純な取り扱いが可能になる
    - 「隠蔽する」
  - ハードウェアの、ソフトウェアによる機能拡張
    - 仮想化によって、高級なプログラミング環境を実現する
- あるひとつの機能を実現するために、いろいろな方法が考えられる
  - 最適な方法は、その時のH/W技術の制約によって異なってくる
    - 「昔は、ハードが高かったが、今はふんだんに利用できる」
    - 「安くはなっているが、性能は上がってない」

3. メモリ管理ーメモリ空間の管理ー

- メモリ
  - コンピュータシステムの主要なハードウェア装置
  - メインメモリとファイル装置（ディスク）
  - 情報（命令とデータ）を格納する



メモリ階層

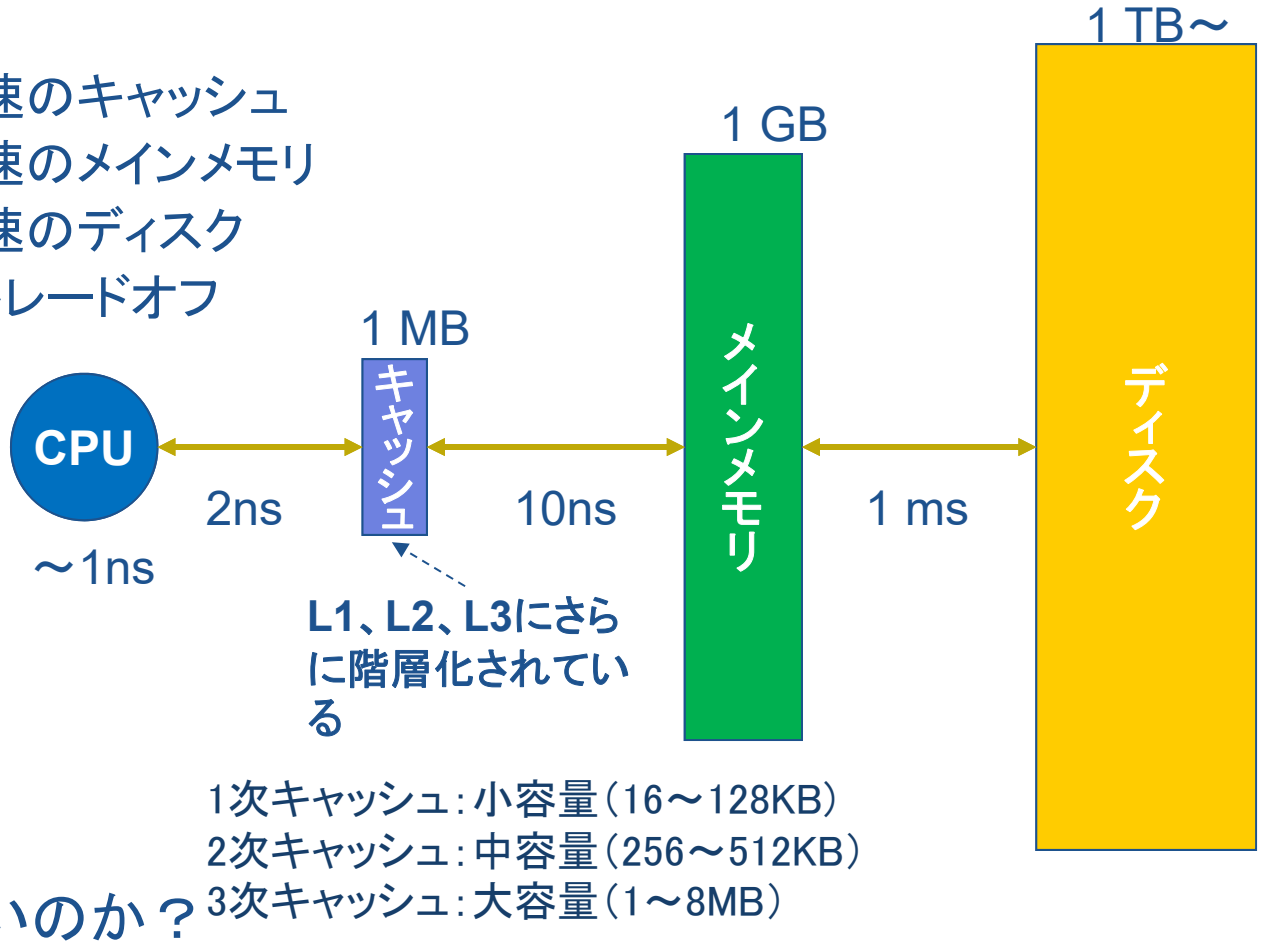
- CPUが直接アクセスするのはもともとメインメモリ（主記憶）
- プログラムやデータは
  - 二次記憶装置（ハードディスク）に保管
  - 実行時に主記憶に転送
- 高速、大容量、不揮発性のメモリがあれば問題はない
  - 実際には性能、コスト、実現性の問題がある
  - 高速にすれば、高価、小容量
  - 大容量にすれば、安価だが低速
- CPU ⇔メインメモリ⇔ディスク

- メモリ管理の重要性
  - プロセスをメモリのどの部分にどのように割り当てるか

メモリ階層の実際

- メモリ階層
  - 少容量、高速のキャッシュ
  - 中容量、中速のメインメモリ
  - 大容量、低速のディスク

  速度と容量のトレードオフ

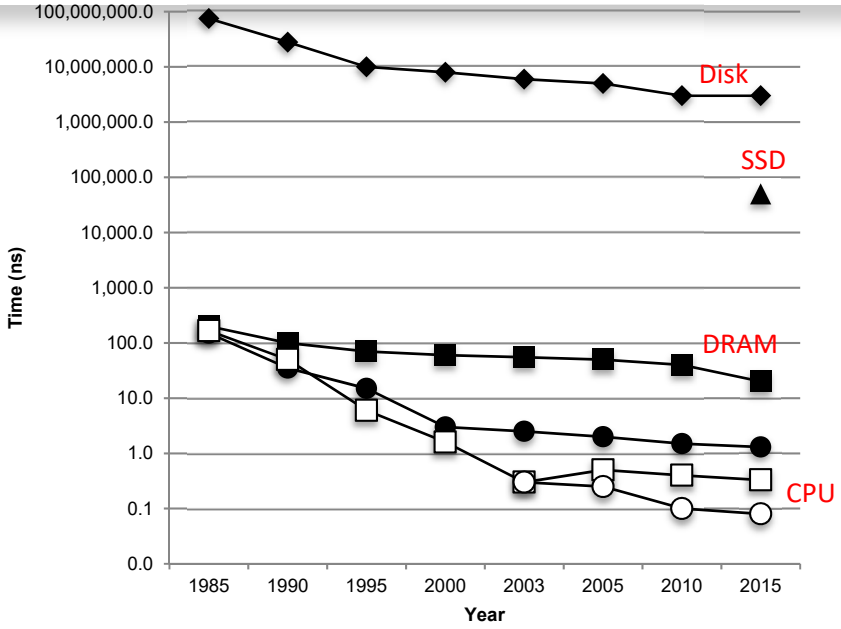


1次キャッシュ：小容量（16～128KB）
2次キャッシュ：中容量（256～512KB）
3次キャッシュ：大容量（1～8MB）

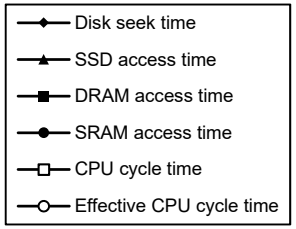
- なぜこれでよいのか？

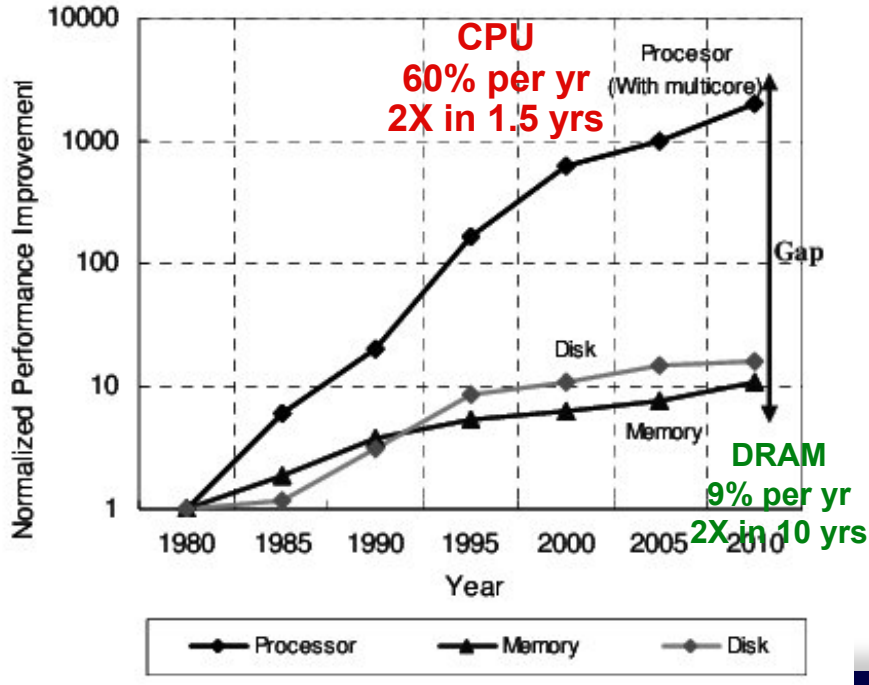
  →参照局所性

ディスクとメモリの速度向上



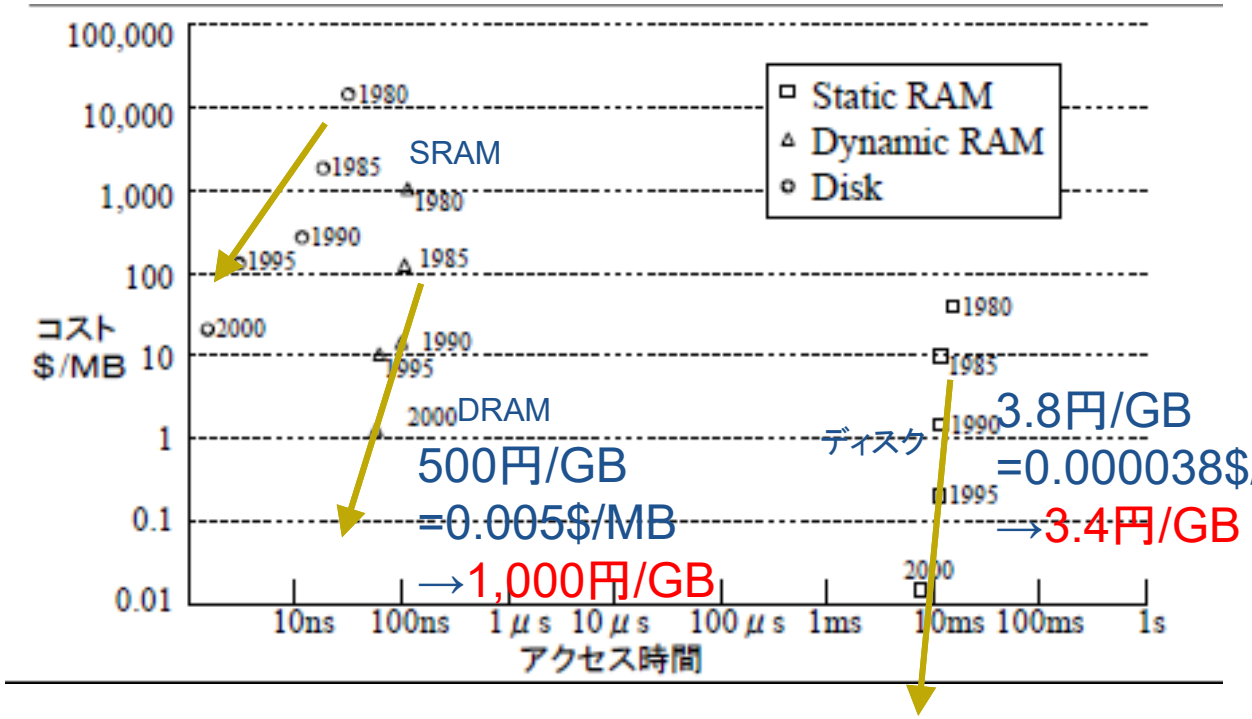
https://cse.sc.edu/~matthews/Courses/513/Lectures.html





ディスクとメモリのコスト推移

- ディスクの発展はメモリの発展に追随している
  - アーキテクチャの変化を促さない



演習問題3.1（復習）

- ディスクアクセスは考えない
- キャッシュのアクセス時間が2ns、キャッシュミス時にはよぶんにメインメモリへのアクセス時間18nsを必要とする。キャッシュのヒット率が90%の時、平均アクセス時間はいくらか？
- アクセス時間が3nsのキャッシュに変更し、そのかわりに容量を増やして性能を確保しようと考えた。ヒット率が何%なら上と同じ平均アクセス時間になるか？

3.1 領域の管理ーメモリの空間的管理ー
3.1.1 メモリ領域の管理

[1] OSによるメモリ管理

- メインメモリとファイル装置を一括してメモリと見なす
- それに対して空間的管理を行う「メモリ領域の管理」
- 具体的には

(A) メインメモリの管理 →3.1節

プロセス割り付け

アクセス権限のチェック

(B) 仮想メモリの管理→3.2節

メインメモリとファイル装置との対応付け

(C) ファイルとファイル装置の管理→3.3節

ファイル割り付け

ファイル保護

3.1 領域の管理ーメモリの空間的管理ー
3.1.1 メモリ領域の管理

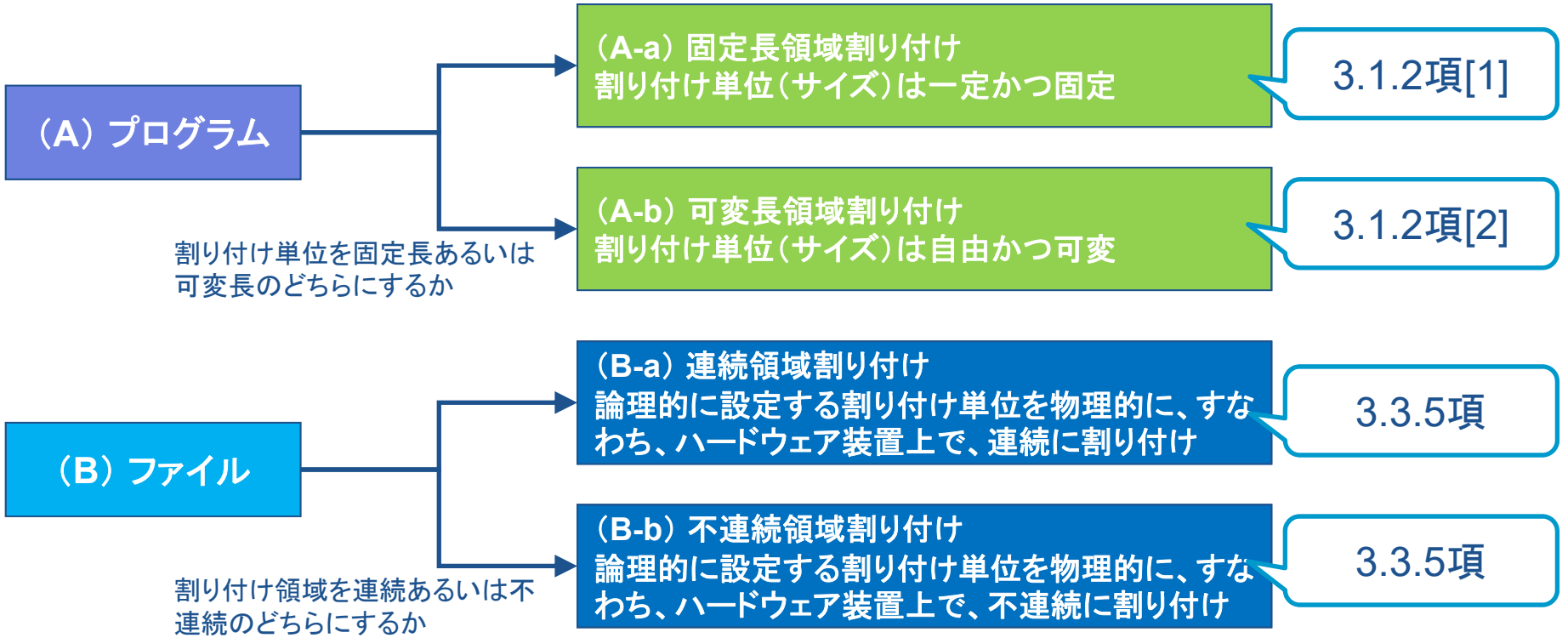
[2] 領域管理の実際

- OSによるメモリ領域管理機能における具体的な領域、および、保持する対象
  - (A) メインメモリ上のプログラムあるいはプロセス
    - 動的(実行時)にしか決定できない要素や属性がほとんど
  - (B) ファイル装置上のファイル
    - 静的(実行前)に決定できる要素や属性がほとんど
- 領域管理の具体的な機能
  - (a) 割り付け(アロケーション)
    - OSという領域管理者が、メモリ装置上でソフトウェア(プロセス、プログラム、ファイル)を保持する
    - そのために、それらのソフトウェアが使用するメモリ領域を確保して、使用可能にする
  - (b) 解放(リリース)
    - 確保し使用しているメモリ領域を未使用(状態)にする
      →OSという領域管理者に返却する
    - あるいはOSという領域管理者による「ごみ集め」の対象に加える。

未使用のメモリ領域を収集して割り付け可能状態にする

3.1 領域の管理ーメモリの空間的管理ー
3.1.1 メモリ領域の管理

- (A)プログラム(プロセス)、(B)ファイルの備える性質が異なる
  →それぞれを対象とするOSの領域管理機能が満たすべき要件も異なる
  →管理対象によって、管理(割り付け)方式を選択する際の判断指標が異なる
  →判断指標は管理(割り付け)方式の分類指標でもある



3.1 領域の管理ーメモリの空間的管理ー
3.1.2 メインメモリ上での領域割り付け

[1] 固定長領域割り付け

- 例:ビットマップ法
  - 固定長(固定サイズ)の領域ごとに、1ビットのフラグ(0:空き、1:割り付け済み)を充てる
  - 長所
    - 管理が簡単
    - 割り付けや解放に要する時間が一定かつ高速
  - 短所
    - 全体が可変長である領域を対象とする場合には、複雑な領域分割や管理が必要となり、最適化が難しい
    - 内部フラグメンテーションが不可避

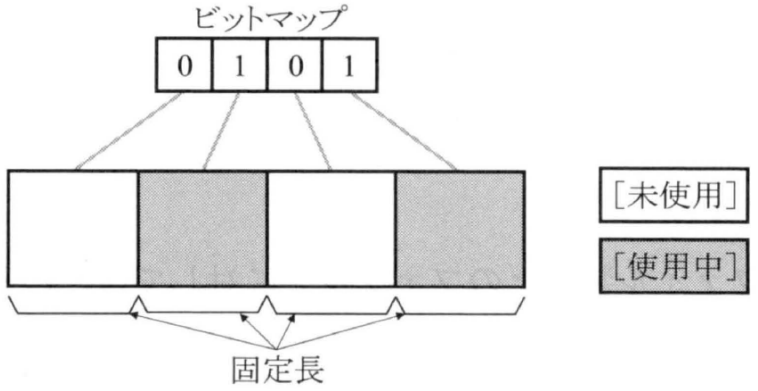


図 3.1 固定長領域割り付け —ビットマップ法—

（参考3.3）内部フラグメンテーション

- フラグメンテーション
  - 領域の断片化や小片化
- 内部フラグメンテーション
  - 領域の割り付けや解放を繰り返しているうちに、各領域の内部に発生する『割り付け不可能な未使用領域の断片や小片』
  - 多数の無駄な領域が発生する
  - 固定長領域割り付けで発生する

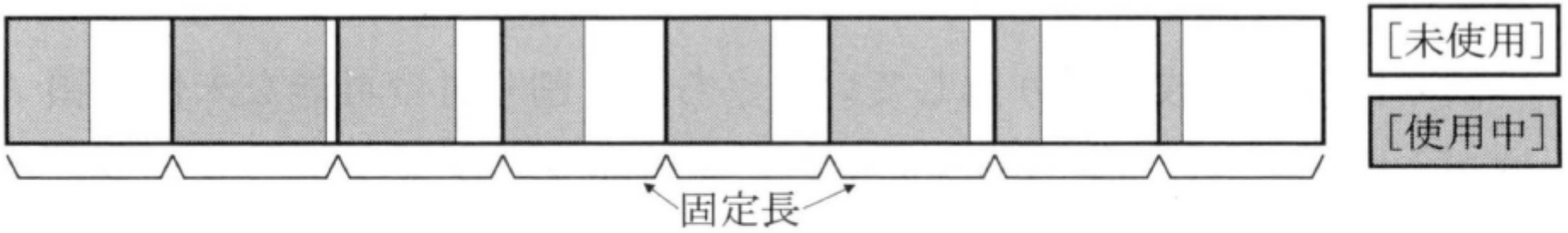


図 3.2　内部フラグメンテーション（参考3.3）

第8回（5月8日2時限）ミニレポート課題

- 1GBのメインメモリがあり、4KBずつ固定長領域をとるとする。メインメモリ上での領域割付としてビットマップ法を用いたとする。
  1. ビットマップテーブルの大きさは何KB必要か？
  2. 1KBのプロセスはいくつ同時に領域を割りつけられるか？　メモリの利用率はいくらになるか？
  3. 100KBのプロセスはいくつ同時に領域を割りつけられるか？　メモリの利用率はいくらになるか？

締切：5月12日（水）
CLEで提出

3.1 領域の管理ーメモリの空間的管理ー
3.1.2 メインメモリ上での領域割り付け

[2] 可変長領域割り付け

- リスト法
  - 可変長の領域をそれに付加したヘッダ（領域サイズおよび連結する次領域へのポインタ)によって管理
- 長所
  - 対象が固定長あるいは可変長のどちらでも、同じ管理方法が適用できる
  - アクセス管理が比較的容易に実現できる
- 短所
  - 割り付け処理も解放処理もどちらも複雑で遅い
  - 外部フラグメンテーションが不可避

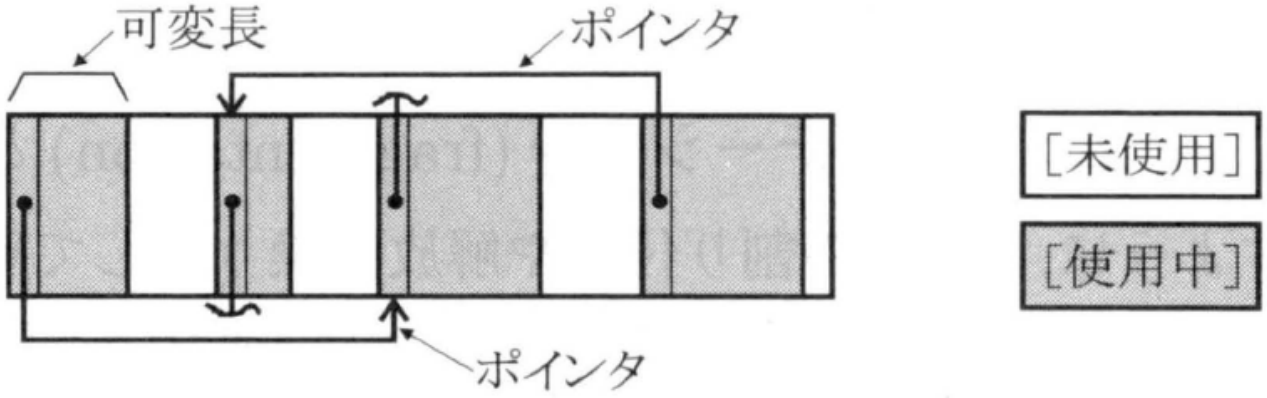


図 3.3　可変長領域割り付け ―リスト法―

（参考3.4）外部フラグメンテーション

- 領域の割り付けや解放を繰り返しているうちに発生する、割り付け可能な未使用領域が小片化あるいは断片化
- 不連続で、効率の悪い割り付けしかできないので、領域管理（割り付けや解放）が遅くなる
- 可変長領域割り付けで発生する
- デフラグ（ゴミ集め）と呼ぶ、小片化や断片化した領域の収集と詰め直し機能によって解消

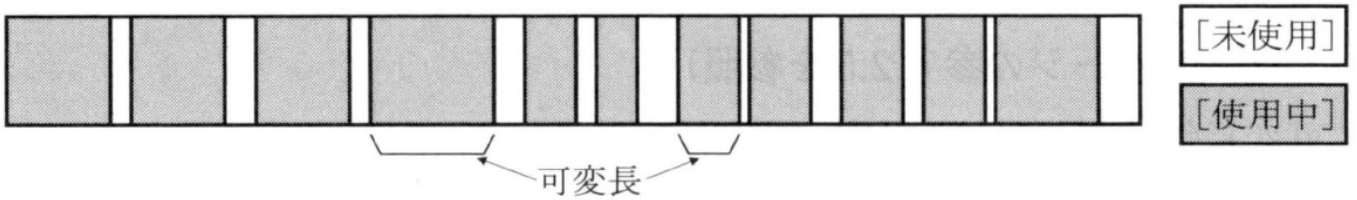
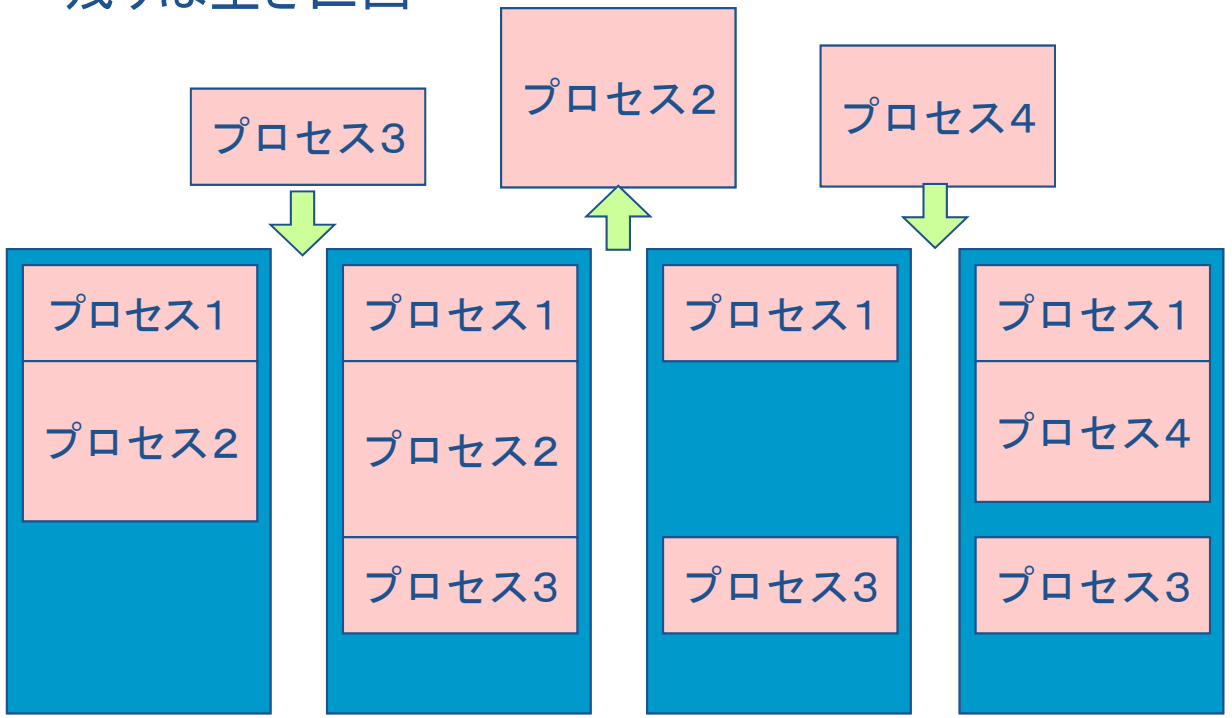


図 3.4 外部フラグメンテーション（参考3.4）

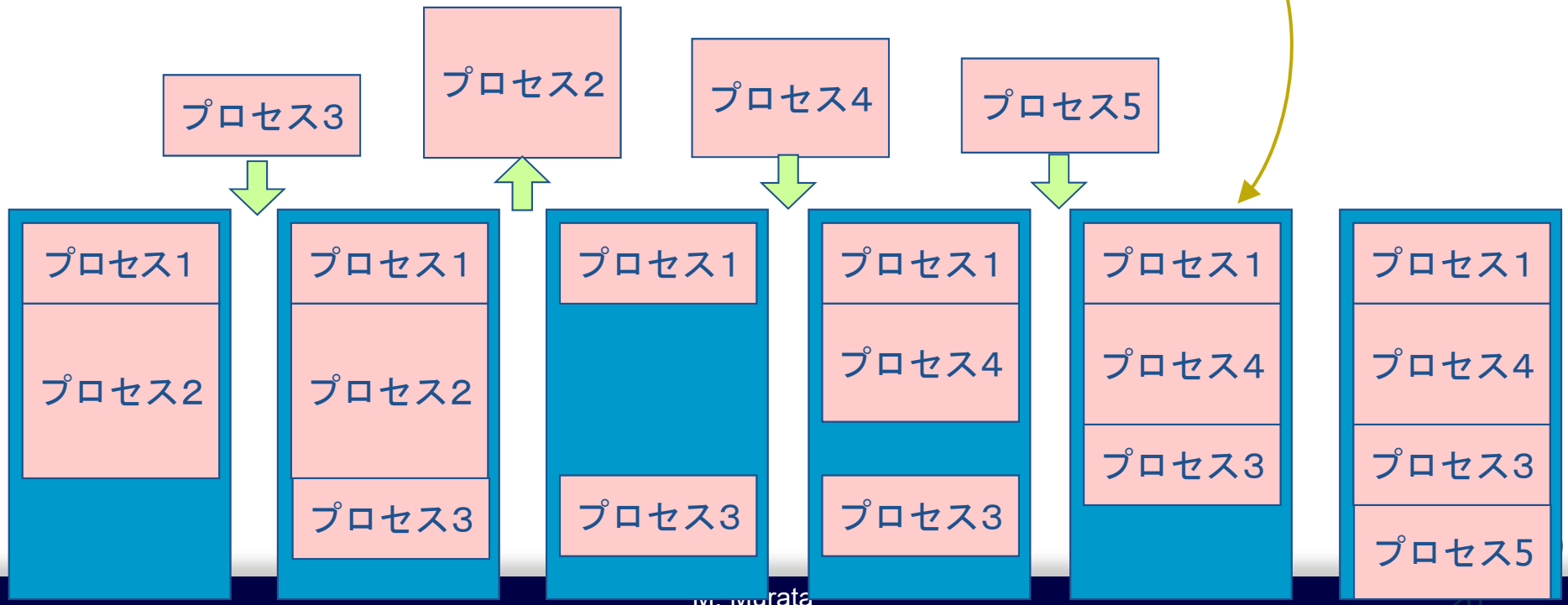
可変長領域割り付け

- 区画サイズを動的に変更して割り付ける
- プロセスが到着すると
  - 格納できる空き区画を探す
  - 空き区画にプロセスを割り当てる
  - 残りは空き区画



メモリ利用率向上策

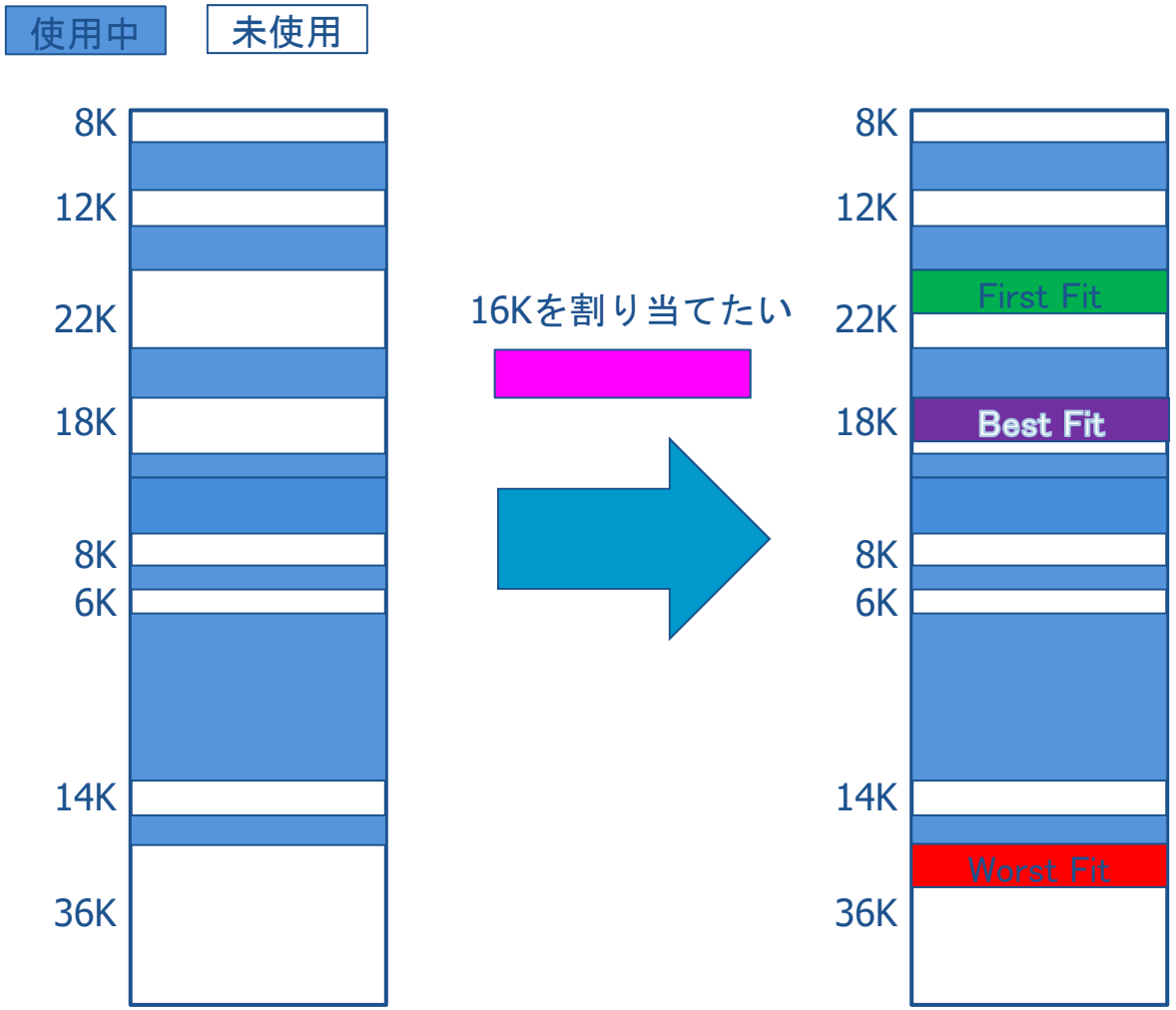
- ガベージコレクション（デフラグ）
  - 使用中の区画を移動して、空き領域を連結
  - 動的再配置前提
- メモリの利用率向上
  - 常にガベージコレクションを実行すれば、全体のメモリ領域に入り切らない場合を除いて、外部フラグメンテーションは避けられる。
- ただし、CPUを浪費するため、実際には使われない
  - 4バイトのコピーに10nsecかかる場合→10MBのプロセス移動に25ミリ秒



可変長領域割り付けの方法

1. ファーストフィット (First Fit) 方式
   - 空き領域探索時に最初にみつかったプロセスからのメモリ要求領域を格納するのに十分な空き領域を割り当てる方式
2. ベストフィット (Best Fit) 方式
   - プロセスからの要求領域を探索する際、領域を割り当てた後の残り領域が一番少ない空き領域を割り当てる方式
3. ワーストフィット (Worst Fit) 方式
   - ベストフィットと逆に、プロセスからの要求領域を探索する際領域を割り当てた後の残り領域が一番大きい空き領域を割り当てる方式

可変長領域割り付けの方法



どれがよいか？

- ファーストフィット方式
  - 空き領域を最後まで探索する必要がないため、探索が高速である。
  - アドレスの下位から順次検索していく検索方式の場合は、アドレスの上位に比較的大きな空き領域が残りやすくなり、大きな領域を要求するプロセスの要求にも答えることが可能となる。
- ベストフィット方式
  - いちばん効率的であるように思える。しかし、空き領域探索で時間的なコストが高くなってしまう場合がある。
  - 割り当て後に残った小さな空き領域が新しいプロセスからのメモリ要求サイズより小さくなりすぎ、使えない領域として残ってしまう。
- ワーストフィット方式
  - 空き領域の大きさが均一化する特性がある。したがって、大きな領域を確保できず、メインメモリの利用効率が悪くなる。