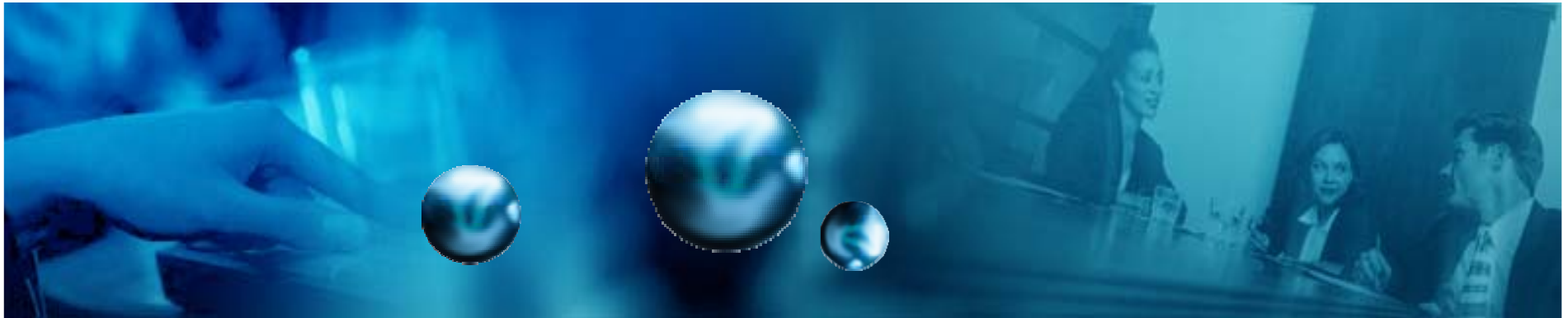


オペレーティングシステム

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



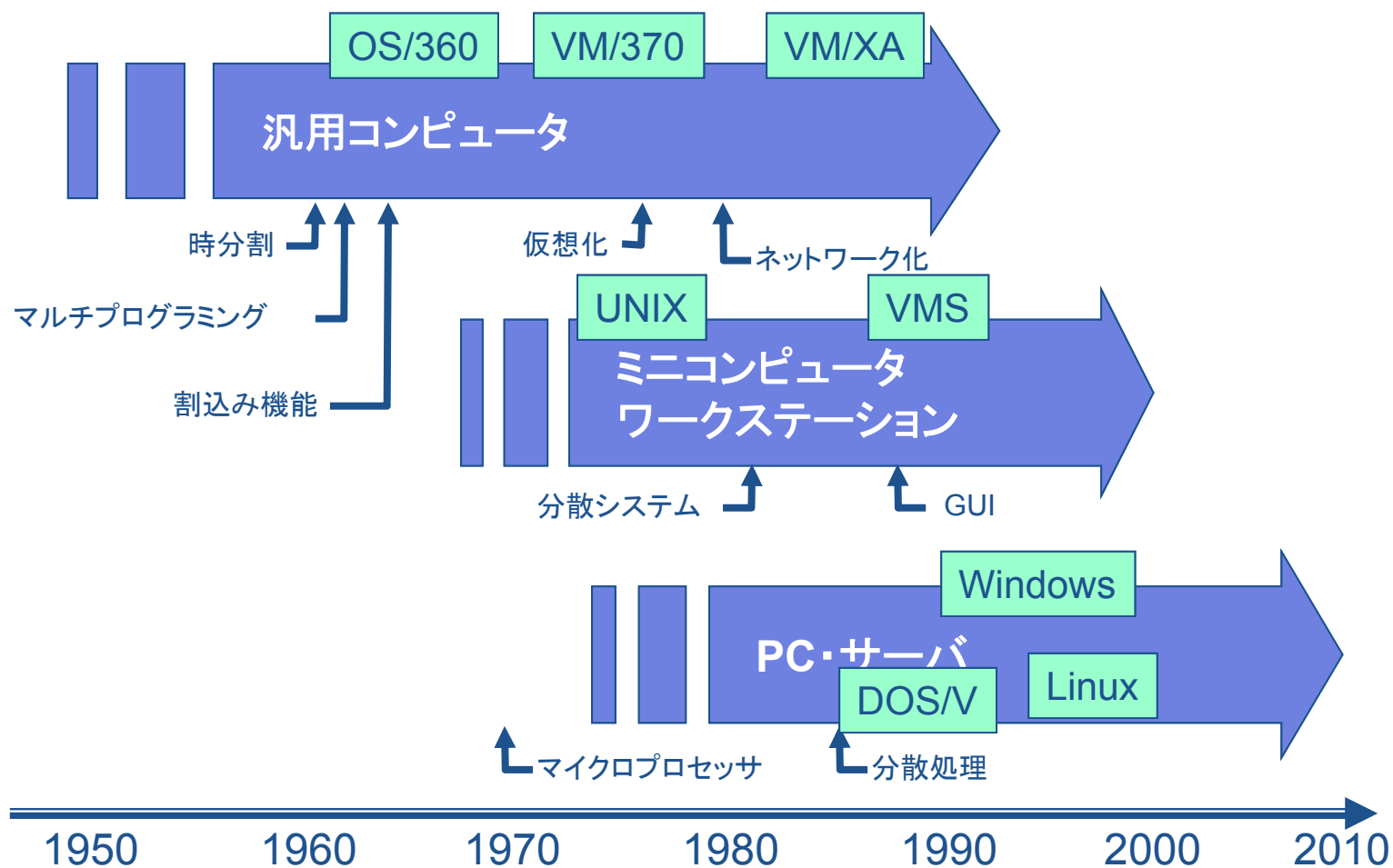
大阪大学大学院情報科学研究科
村田正幸

murata@ist.osaka-u.ac.jp

<http://www.anarg.jp/>



OSの歴史





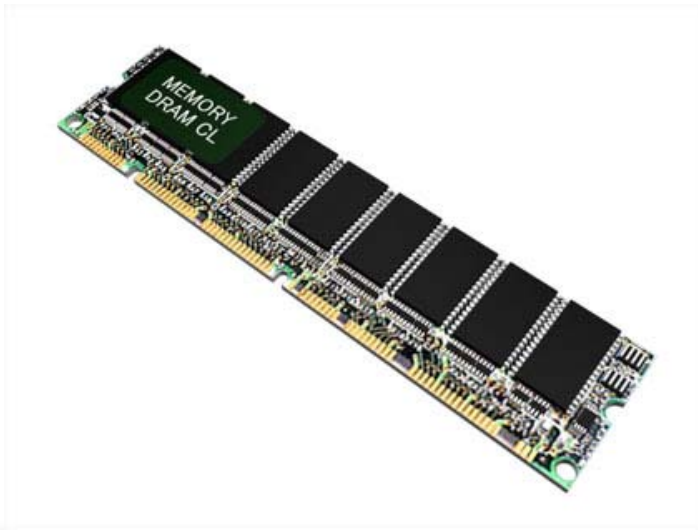
「なぜこんなものが必要なの」と思ったときは思い出そう

- 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、単純な取り扱いが容易になる
 - 「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、性能は上がっていない」



3. メモリ管理ーメモリ空間の管理ー

- メモリ
 - ー コンピュータシステムの主要なハードウェア装置
 - ー メインメモリとファイル装置(ディスク)
 - ー 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



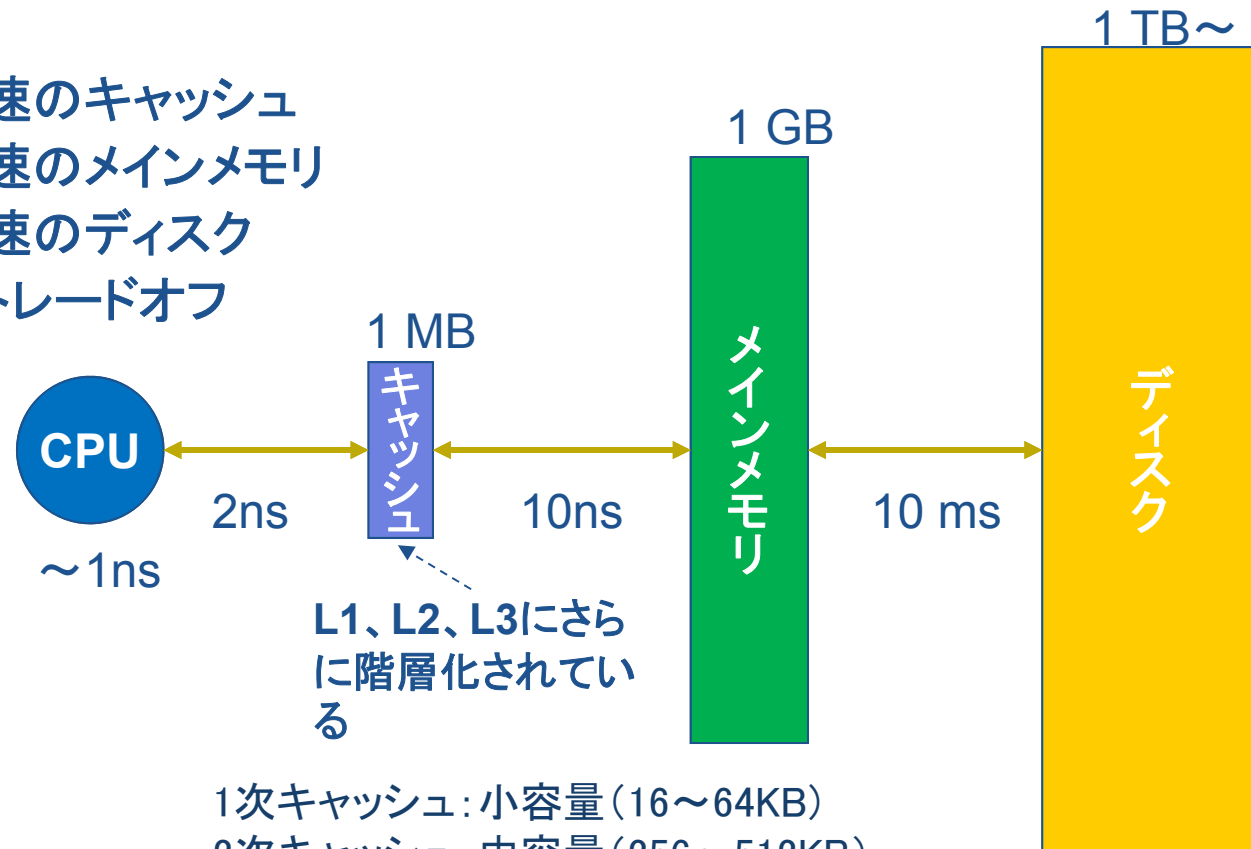
- メモリ管理の重要性
 - プロセスをメモリのどの部分にどのように割り当てるか



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



- なぜこれでよいのか？

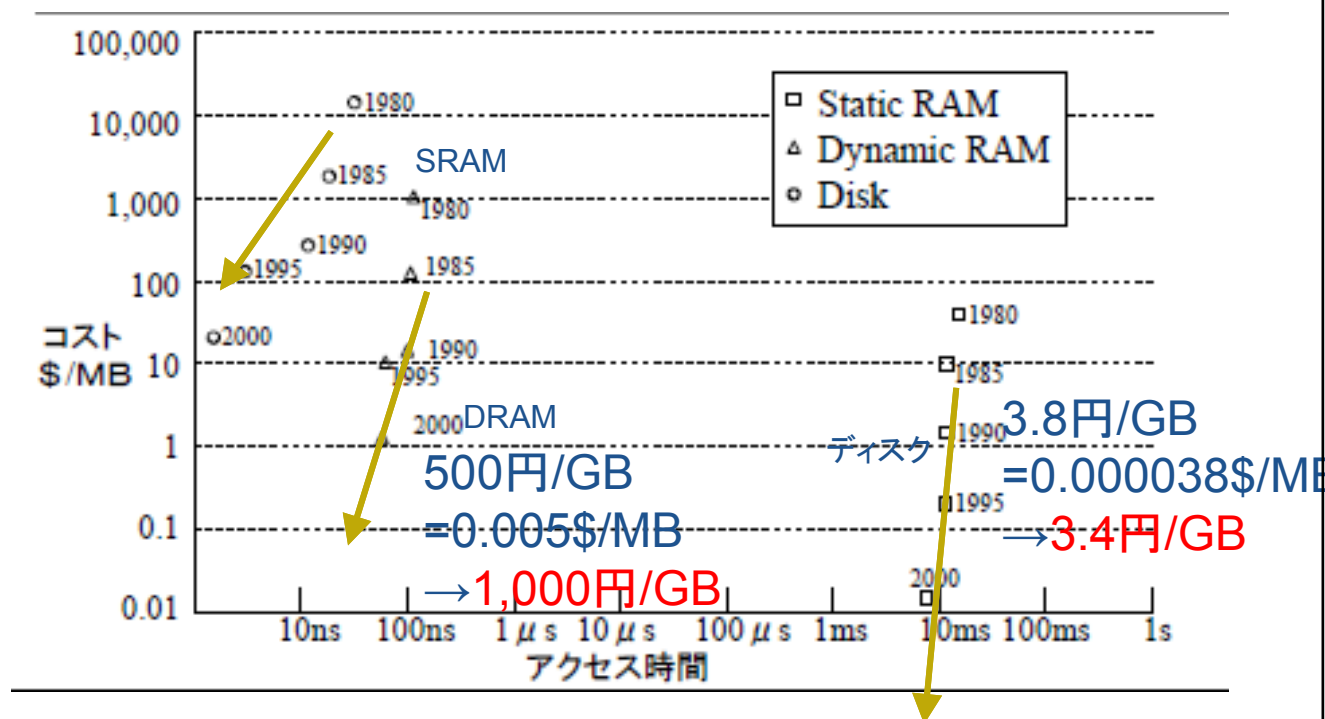
→参照局所性

1次キャッシュ: 小容量 (16~64KB)
2次キャッシュ: 中容量 (256~512KB)
3次キャッシュ: 大容量 (1~8MB)



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

- とりあえず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

- ー 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
- ー それに対して空間的管理を行う「メモリ領域の管理」
- ー 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。

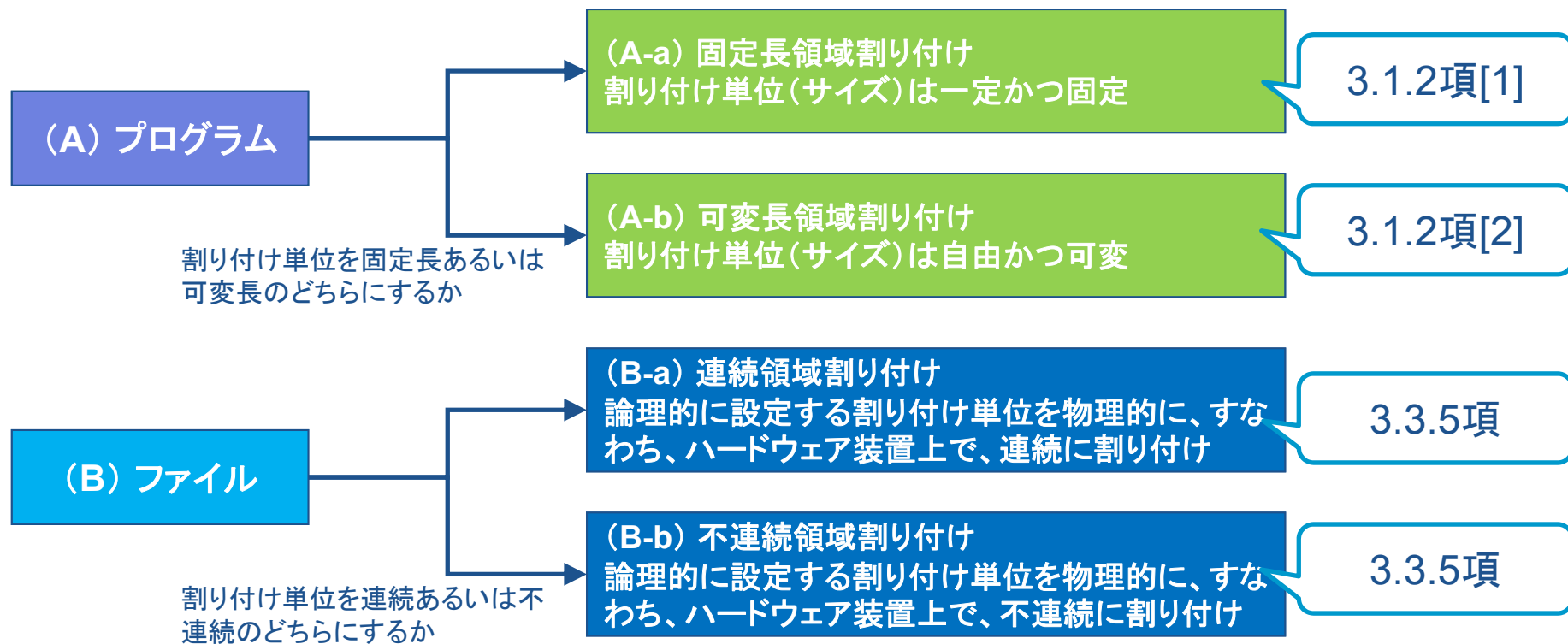
未使用のメモリ領域を収集して割り付け可能状態にする



3.1 領域の管理—メモリの空間的管理—

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—メモリの空間的管理—

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

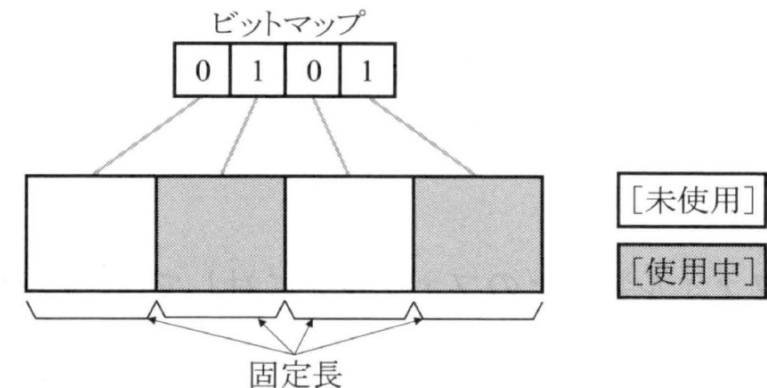


図 3.1 固定長領域割り付け —ビットマップ法—



(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しやすい

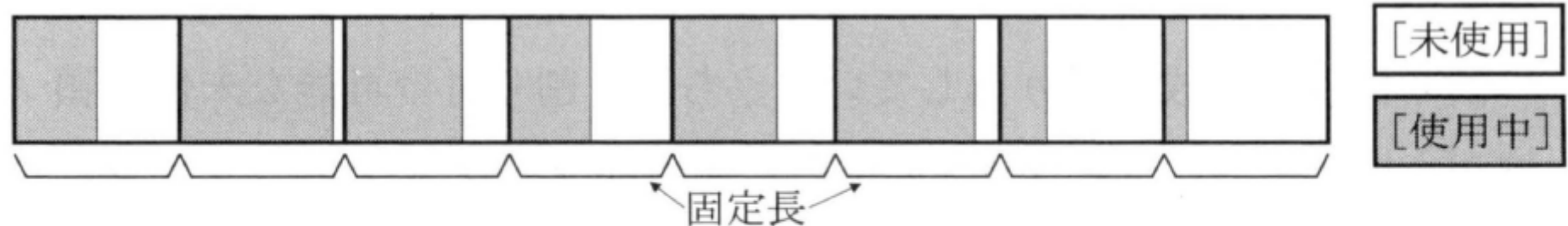


図 3.2 内部フラグメンテーション (参考 3.3)



演習問題3.2

- 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
 - ビットマップテーブルの大きさはどれだけ必要か？
 - 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？
 - 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？



3.1 領域の管理—メモリの空間的管理—

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

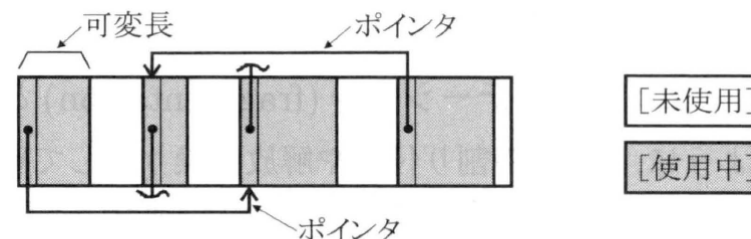


図 3.3 可変長領域割り付け —リスト法—



(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しやすい
- デフラグと呼ぶ、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

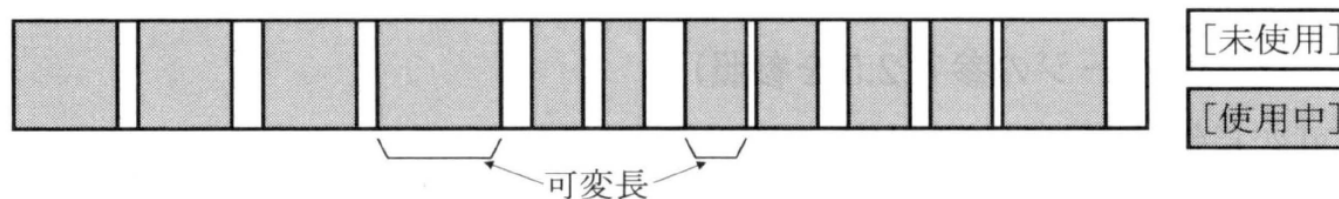
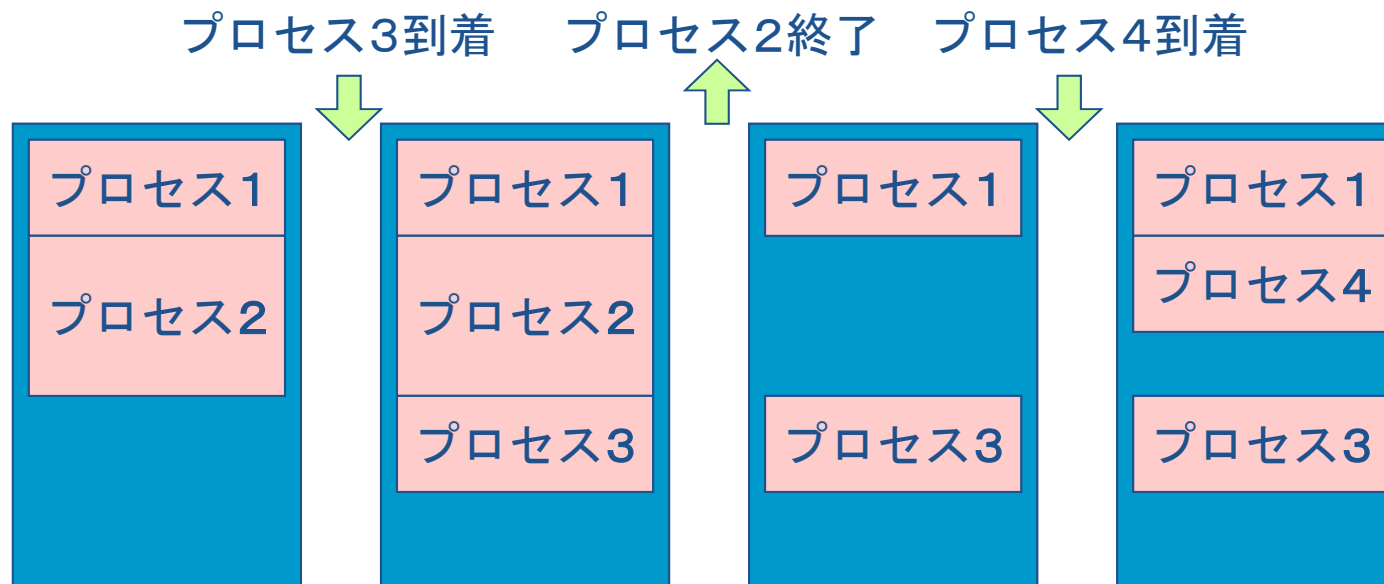


図 3.4 外部フラグメンテーション (参考3.4)



可変長領域割り付け

- 区画サイズを動的に変更
- 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

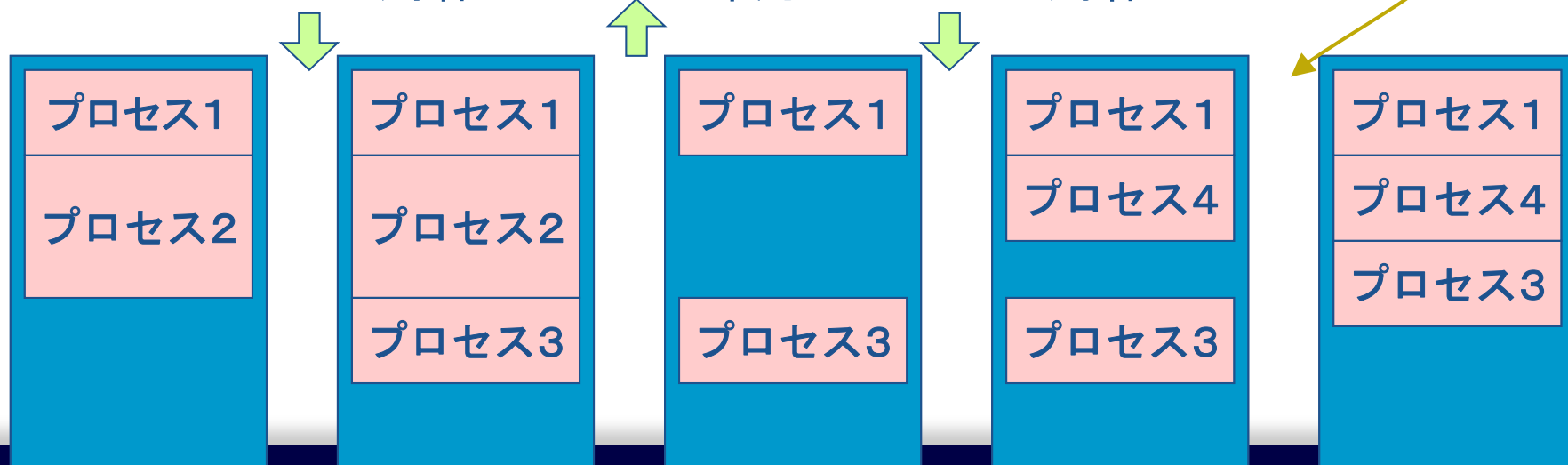




メモリ利用率向上策

- デフラグ(ガベージコレクション)
 - 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
 - 内部断片化が避けられる
 - (外部断片化は起こる)
- 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のプログラム移動に25ミリ秒

プロセス3到着 プロセス2終了 プロセス4到着





参考：ディスクのデフラグ

