4. オペレーティングシステム

http://cobayasi.com/koza/ap/4_os.pdf

4.1 OSの基本機能

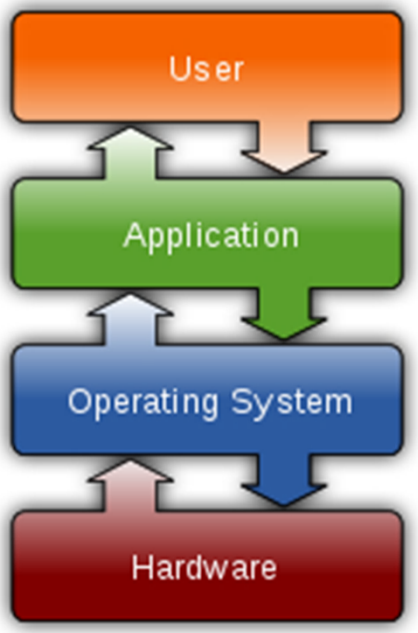
4.2 記憶管理と同期・排他制御

オペレーティングシステムの役割り

オペレーティングシステム(operating system : OS)は、コンピュータを、容易にかつ効率よく使用するための一連の制御プログラム

【容易に】
ハードウェアの様々な機能を操作するために、使いやすいインターフェースを提供すること。また、ハードウェアの様々な機能が簡単に維持できること。

【効率よく】
ハードウェアの様々な機能が、複数のアプリケーションによって共有され、その優れた能力が無駄なく発揮できること。



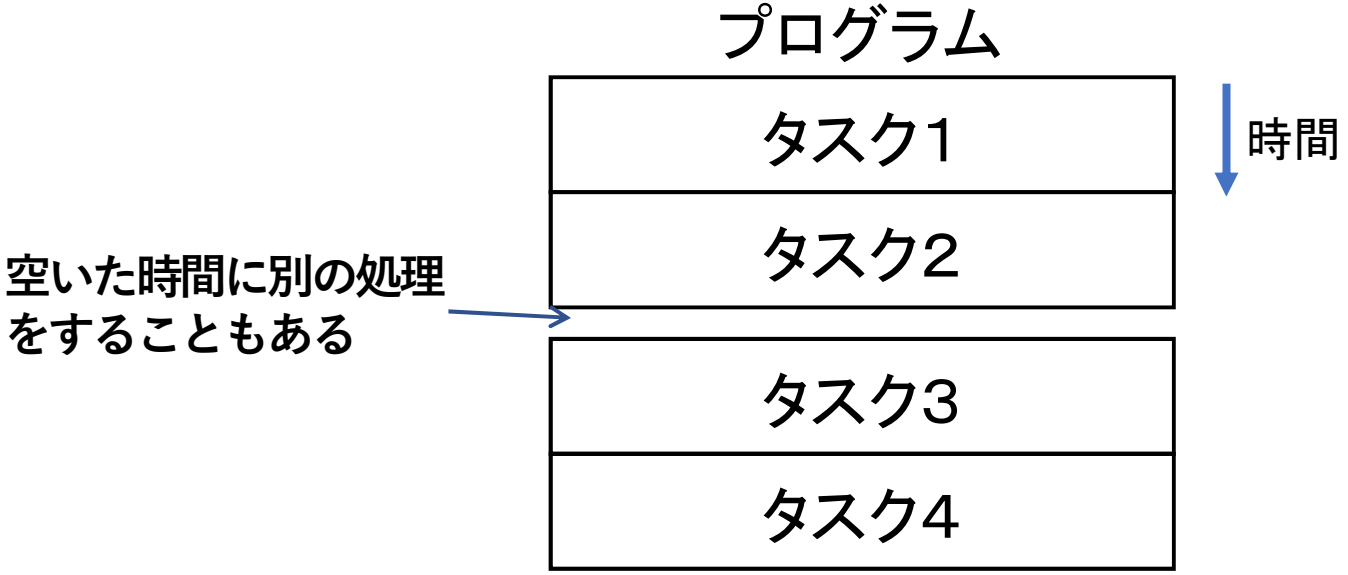
4.1 OSの基本機能

4.1.1 タスク管理(プロセス管理)
4.1.2 データ管理と入出力管理
4.1.3 マルチプログラミング

4.1.1 タスク管理(プロセス管理)

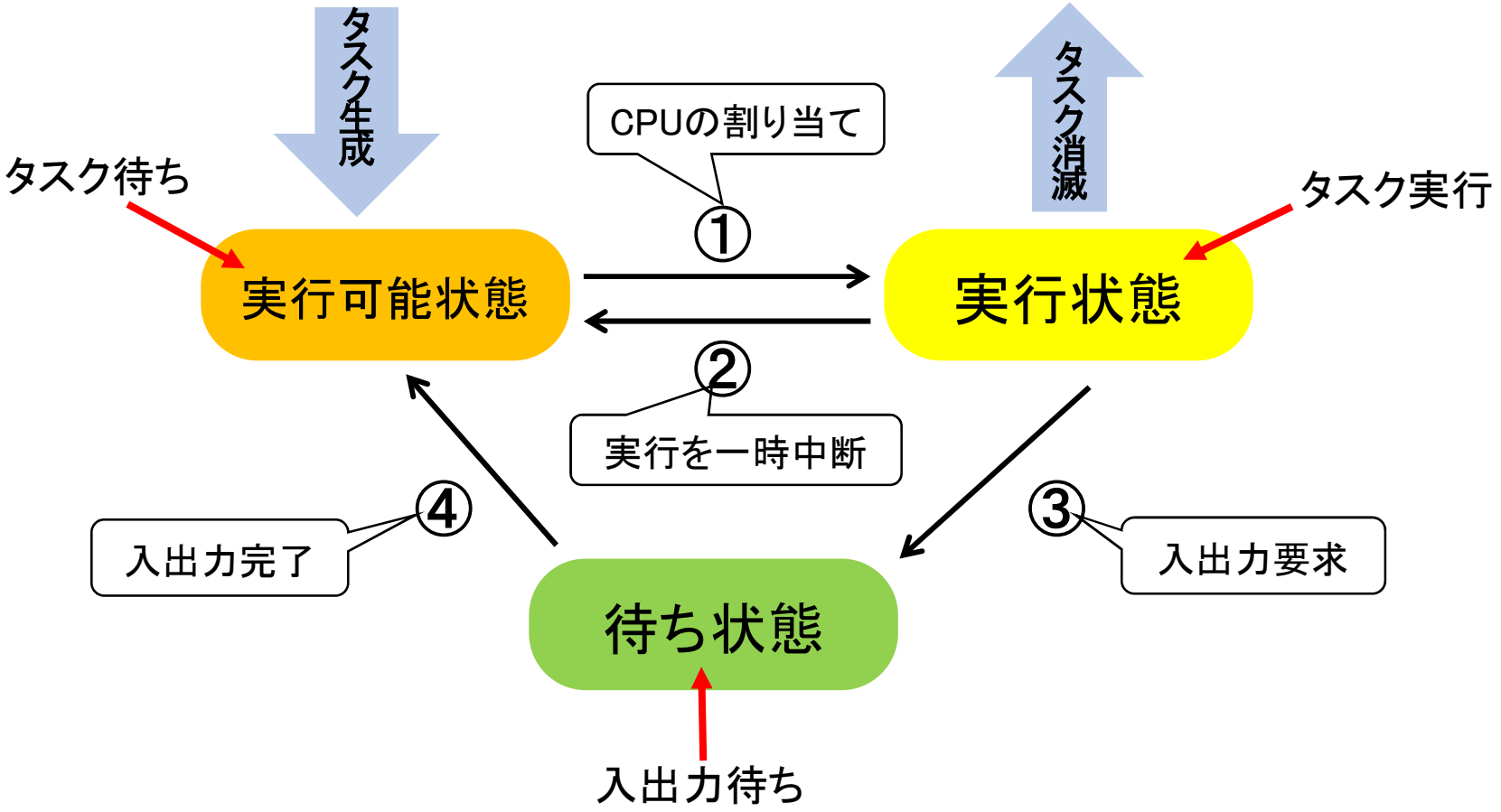
コンピュータの構成要素であるメモリやファイルなどの資源(コンピュータリソース)を使って、CPUが処理する最小実行単位のことを、タスク(プロセス)と呼ぶ。

1つのプログラムを動かすと、いくつかのタスクに分けて効率よく処理するために、分けられたタスク毎に、コンピュータリソースが適切に割り当てられる。



● タスクの状態遷移

生成したタスクを効率よく実行させるために、OSはCPUの使用権を適切に割り当てる。この割り当てを行うために、タスクを3つの状態に分けて管理する。



- 実行可能状態(Ready)
  いつでも実行が可能な、CPUの使用権が回ってくるのを待っている状態
- 実行状態(Run)
  CPUの使用権が与えられて、実行中の状態
- 待ち状態(Wait)
  入出力処理が発生したので、その処理が終了するのを待っている状態

① CPUの実行権が回ってきたので、実行状態に移る
② 優先度の高いタスクが発生したので、実行可能状態に移る
③ 入出力処理が発生したので、待ち状態に移る
④ 入出力処理が終わったので、実行可能状態に移る

● スケジューリングと優先順位

OSが実行するタスクを選び（ディスパッチ）、その順番を決めることを、スケジューリングと呼ぶ

- スケジューリング方式

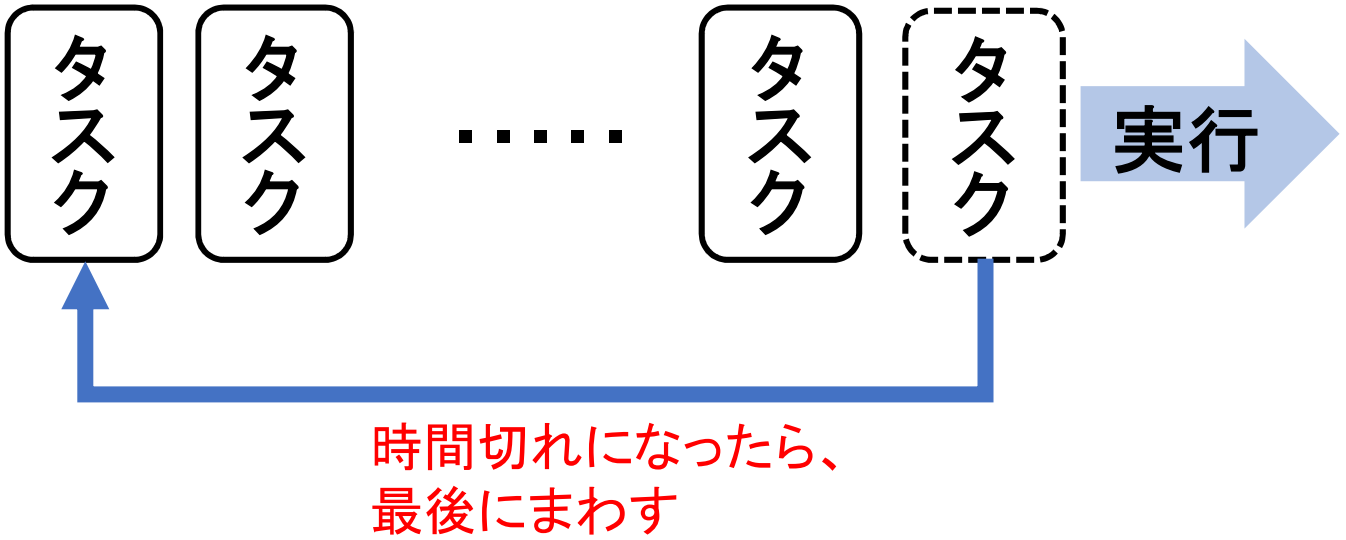
イベントドリブン	マウスによる入力など、環境の変化をタスクの状態遷移のきっかけ（トリガ）として、スケジューリングする
タイムスライス	タイマー割込みで一定時間（約数十ミリ秒）ごとにスケジューリングし、タスクを切り替える

- 優先順位方式

静的優先順位	予め決められた優先度を設定し、その優先度の高いものから順に実行する
動的優先順位	タスクにCPUの使用権の割り当てをするまでの待ち時間に応じて、優先度を徐々に上げて行く

● ラウンドロビン方式(Round Robin:RR)

タイムスライス方式で、CPUの使用権を一定時間(タイムウォンタム)毎に切り替えて、タスクを実行する。実行状態になったタスクは、CPUの使用権が規定時間内で終了できなかった場合は、次のタスクに使用権が与えられ、このタスクは後にまわされる。



4.1.2 データ管理と入出力管理

- データ管理

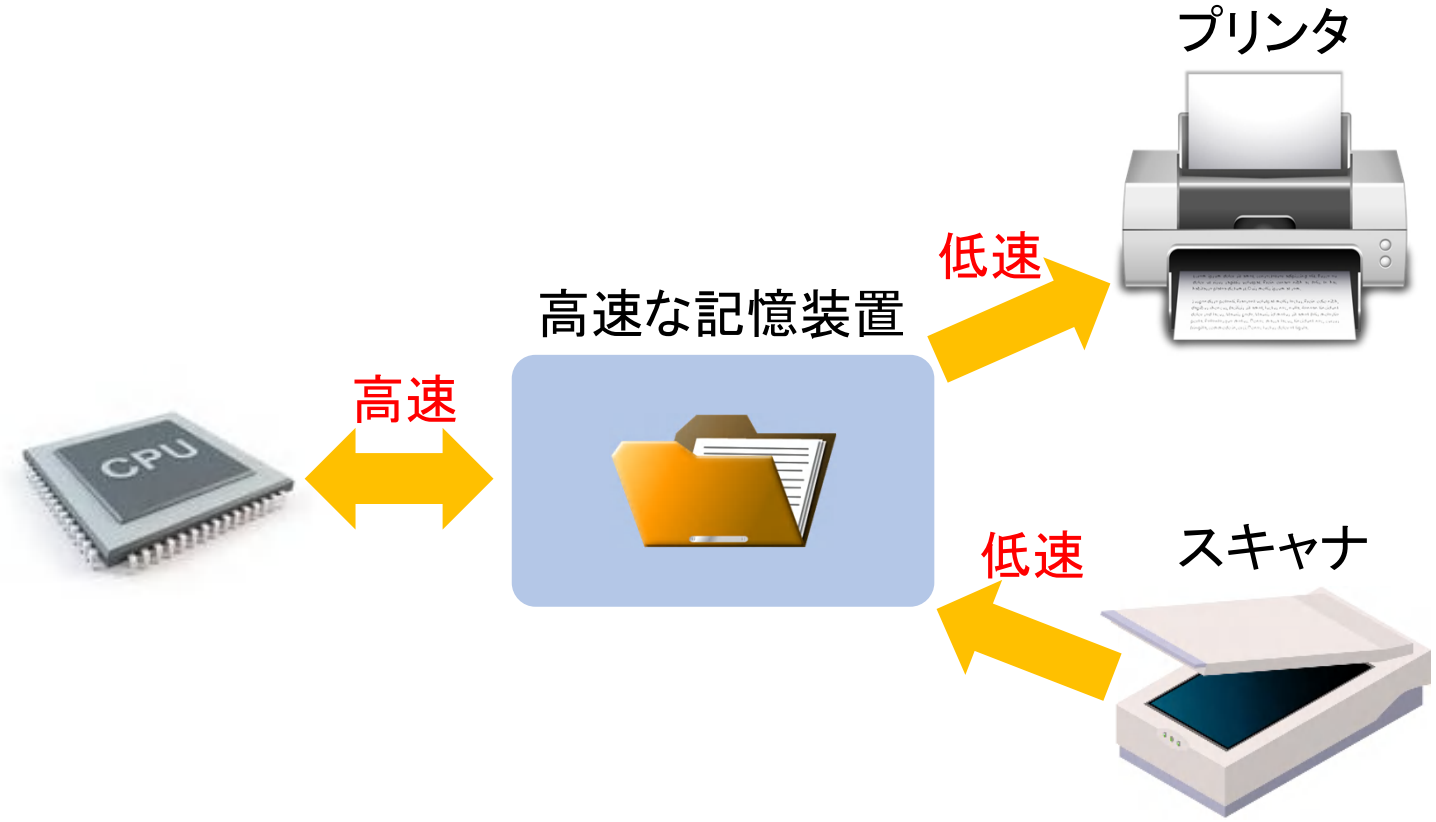
  コンピュータシステムでは、プログラムやデータをすべてファイルと呼ぶ統一した単位にして扱うことで、処理に使用するプログラムとは、別のリソース（共有資源）としている

- 入出力管理

  データの読込みや出力データの書出し、入出力装置の制御を行うデータ管理の一部を、入出力管理と呼ぶ

● スプーリング

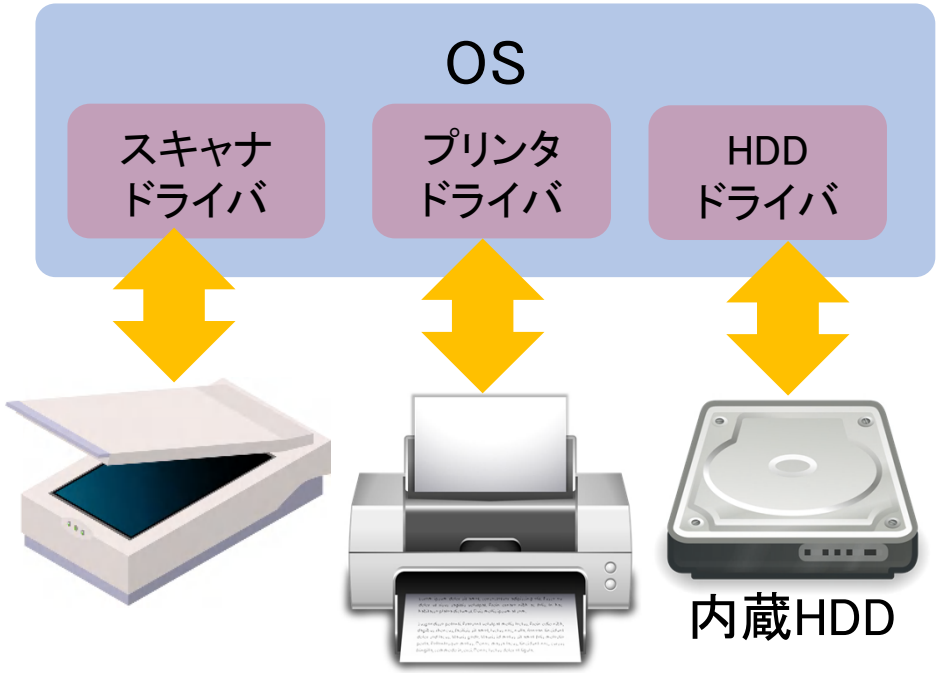
CPUと入出力装置の処理速度には、大きな差がある。そこで、入出力データを一時的に高速な記憶装置へ蓄え、CPUが入出力装置の待たなくて済むようにする。



● デバイスドライバ

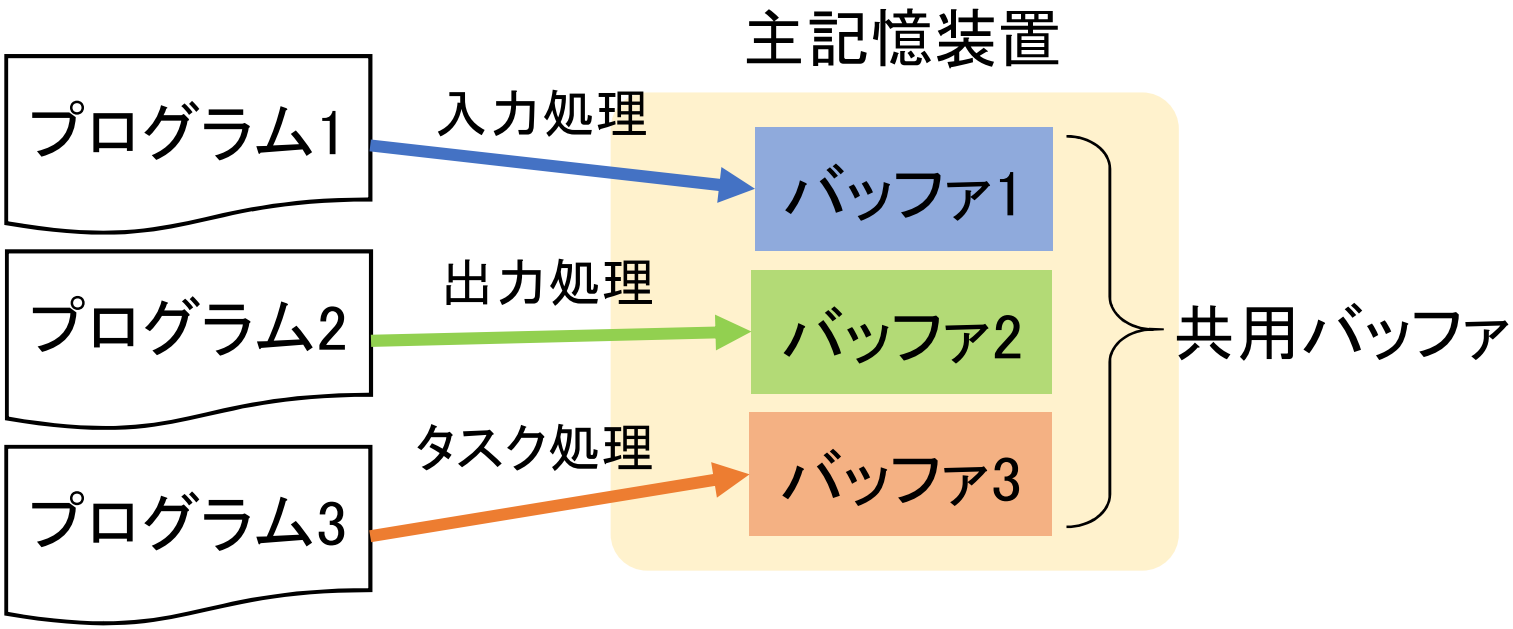
コンピュータ内部に取り付けた機器や、外部に接続した機器を制御・操作するためのソフトウェア。

OSがこうした機器を制御するための橋渡しを行なうもので、利用者が直接操作することは稀で、OSに組み込まれてその機能の一部として振舞うようにできているものが多い。



● バッファプール

主記憶装置内に用意した複数の共用バッファ(バッファプール)を、複数のプログラムで共用することで、複数のプログラムによる入出力処理とタスク処理を並行動作させ、処理速度を上げることができる

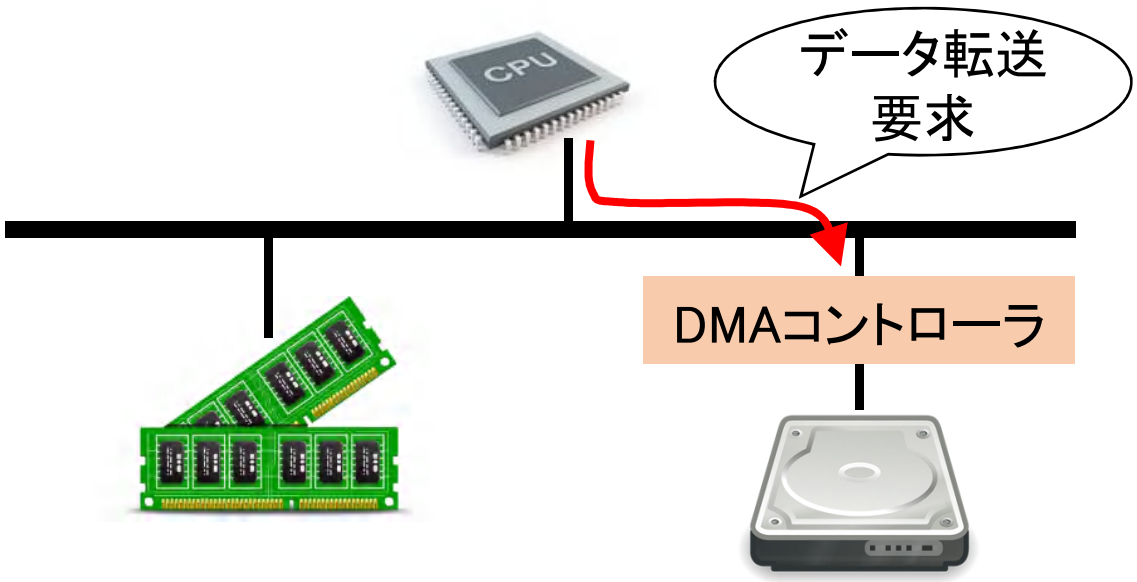


● 直接メモリアクセス機構

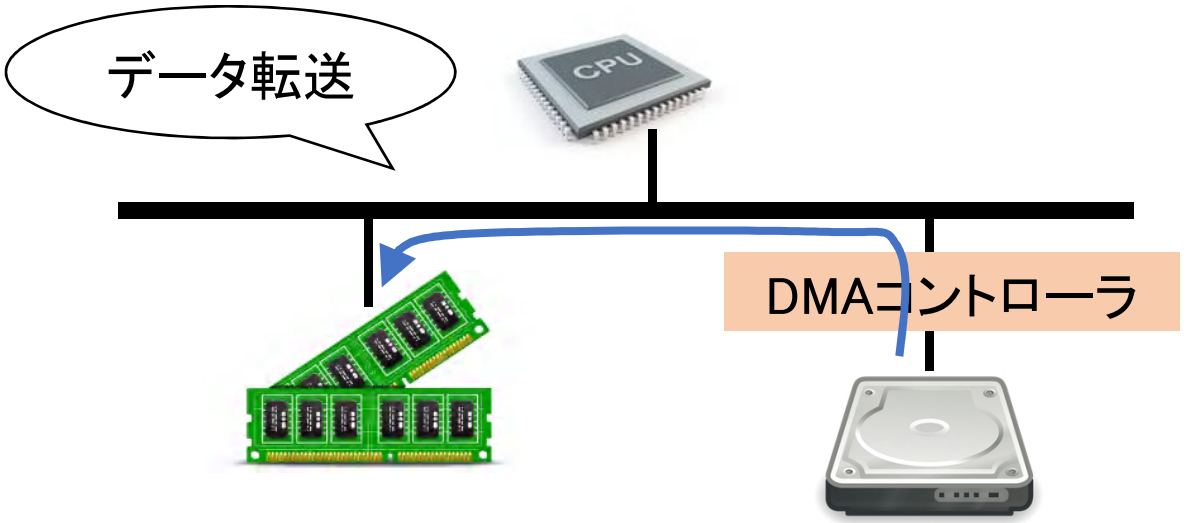
（Direct Memory Access : DMA）

入出力装置とメモリ間のデータ転送を、CPUの介入なしに行う。
DMAでは、専用の制御回路(DMAコントローラ)が転送制御を行う。

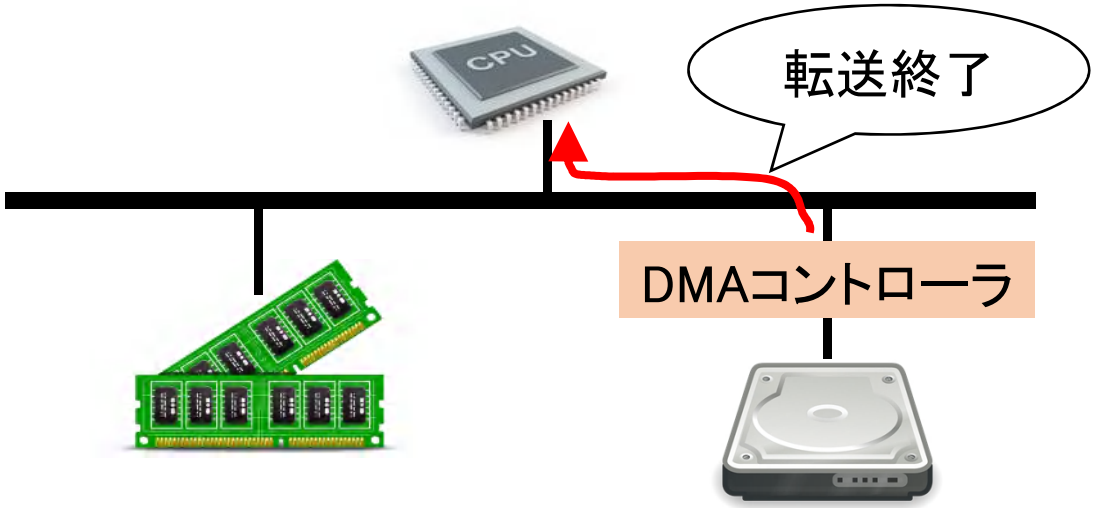
① CPUはDMAコントローラに対して制御信号を送る



② DMAコントローラが転送制御を行う



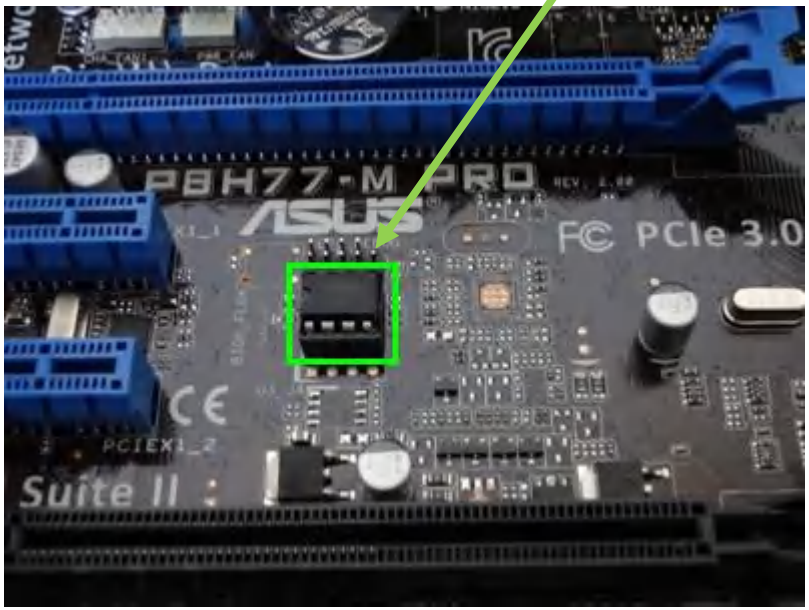
③ DMAコントローラはCPUに対して転送終了を通知する



● BIOS(Basic Input Output System)

起動時のOSの読み込みや、接続された装置・機器の基本的な入出力制御などを行うソフトウェア。

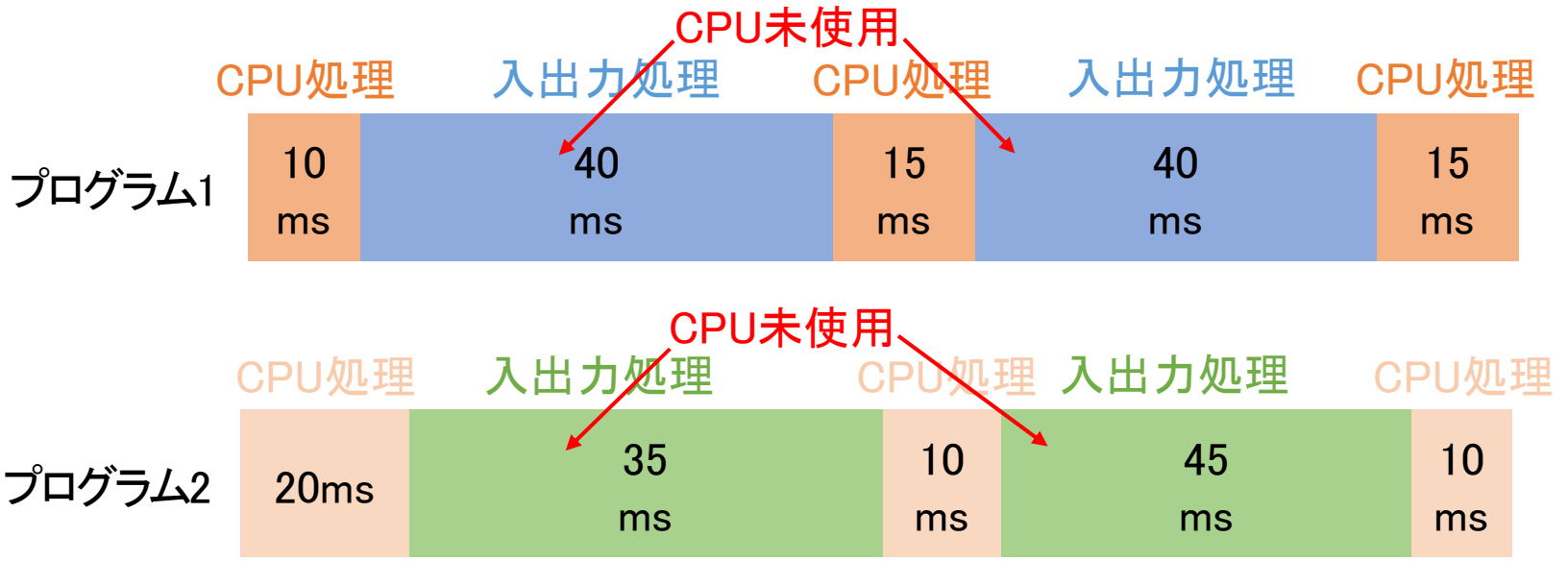
BIOSは、ROMやフラッシュメモリなどの不揮発メモリに記録し、電源投入後に最初に実行する。コンピュータ内の各装置を初期化して利用可能な状態にし、ストレージ(外部記憶装置)からOSを起動するためのブートローダと呼ばれるプログラムを読み込んで実行する。



コンピュータのマザーボード

4.1.3 マルチプログラミング

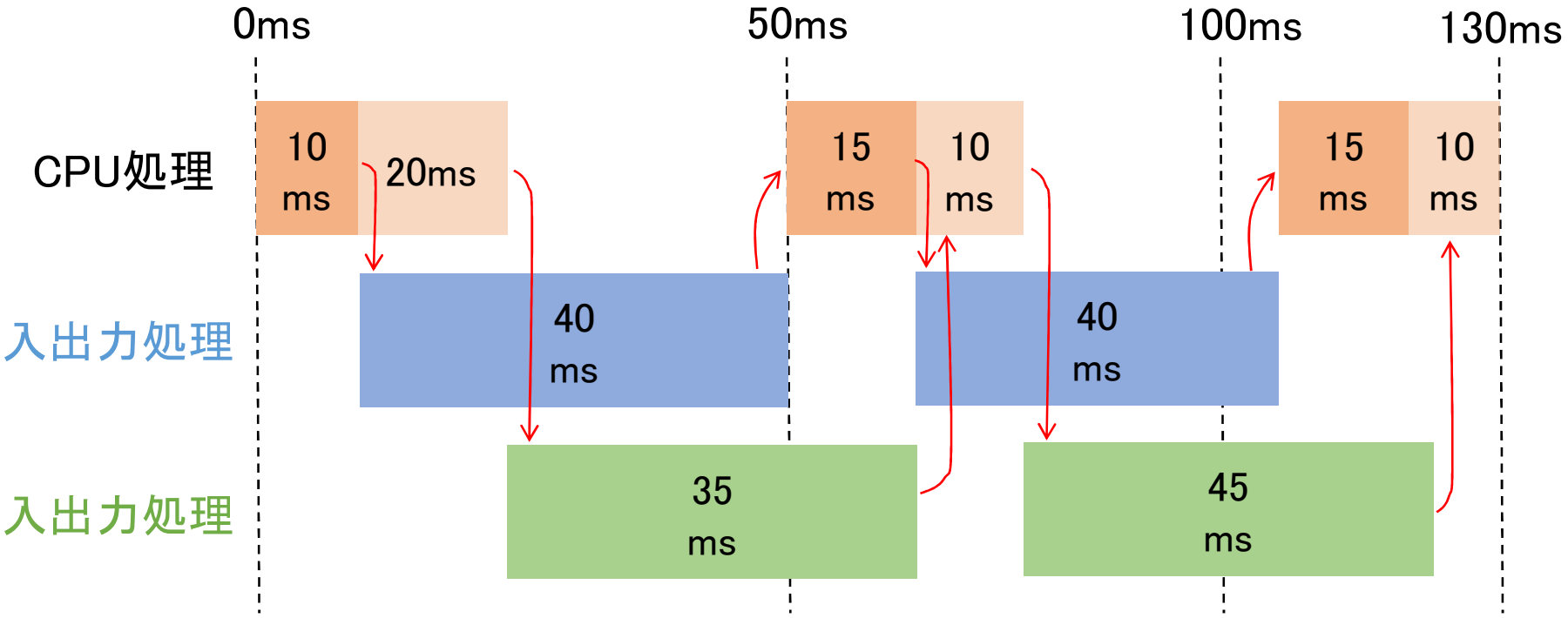
1つのCPUで、複数のプログラムを見かけ上同時に実行する。CPUの未使用時間を減らし、利用効率を向上させる。



普通（シングルプログラミング）でやると、両プログラム(120ms+120ms)が終わるのは240ms

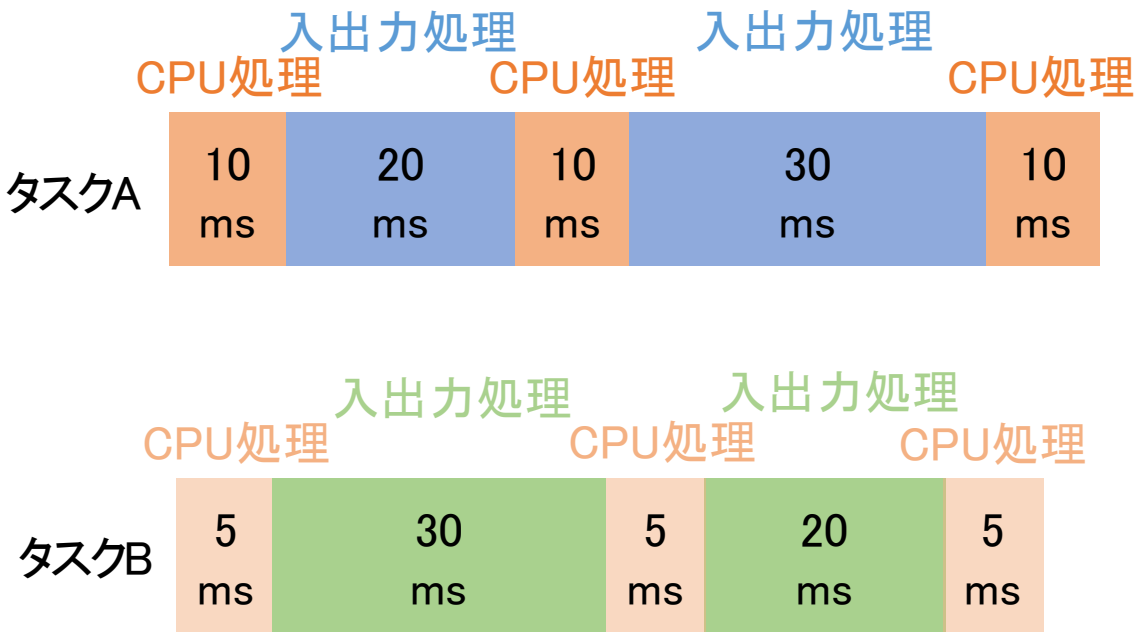
マルチプログラミングの実行条件

- プログラム1は、プログラム2よりも優先度が高い
- プログラム1の入出力処理とプログラム2の入出力処理は、競合しない



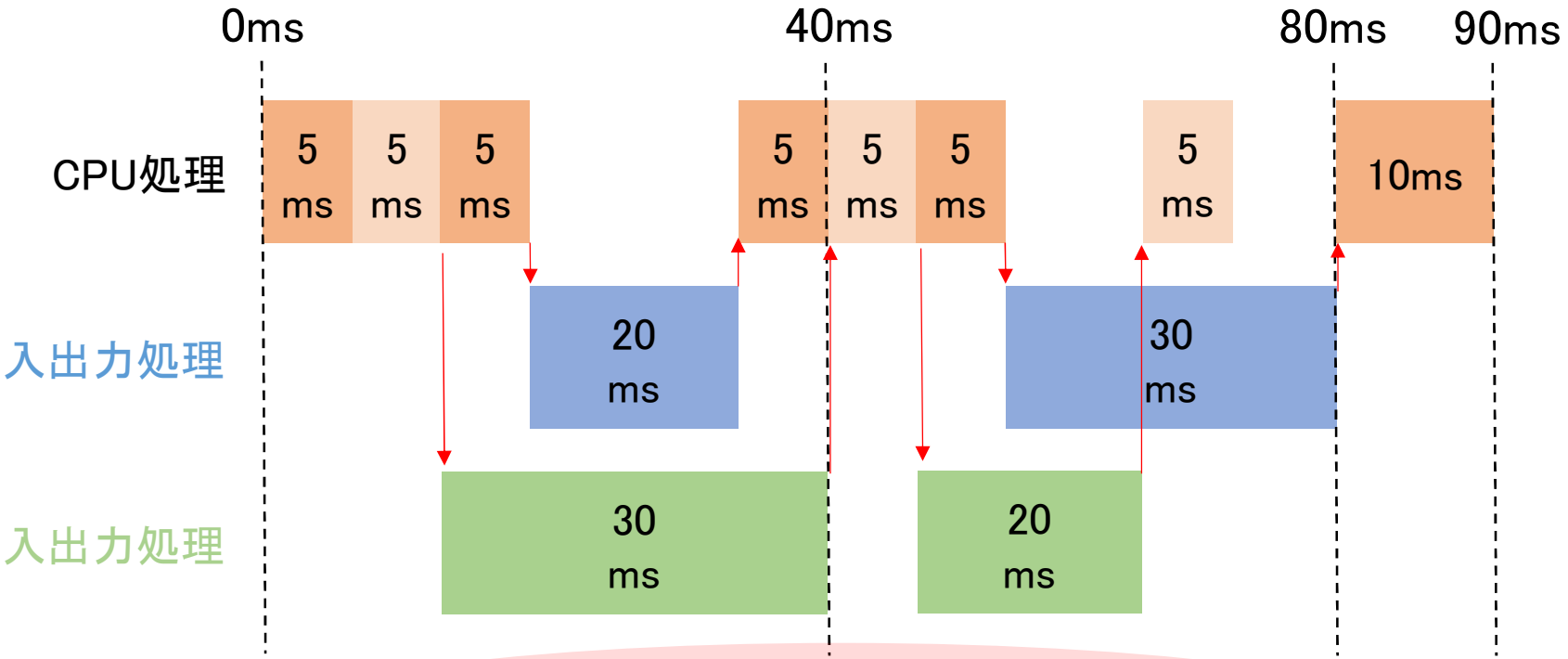
マルチプログラミングでやると、両プログラムが終わるのは130ms

● タイムチャートの作成例(1)



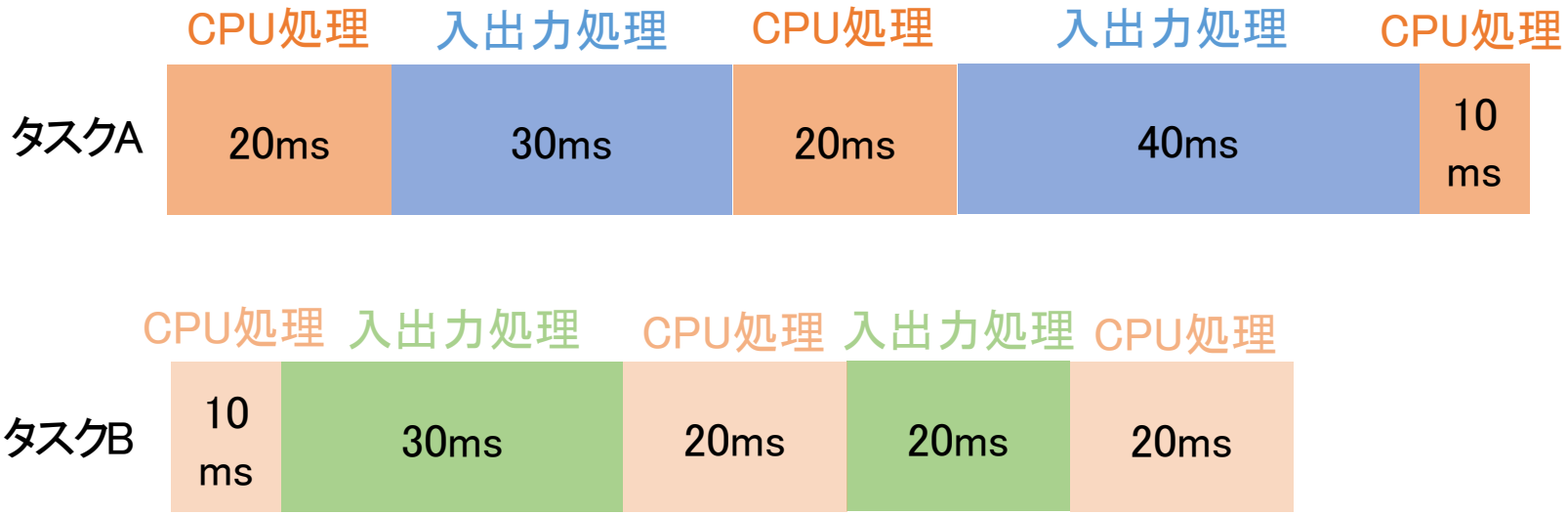
マルチプログラミングの実行条件

- 最初はタスクAにCPUが割り当てられる
- CPU割り当て時間は、5msごとのラウンドロビン方式でスケジュールする
- 2つのタスクがアクセスする入出力処理は、異なる装置で行い、同時処理ができる



マルチプログラミングでやると、両プログラムが終わるのは90ms

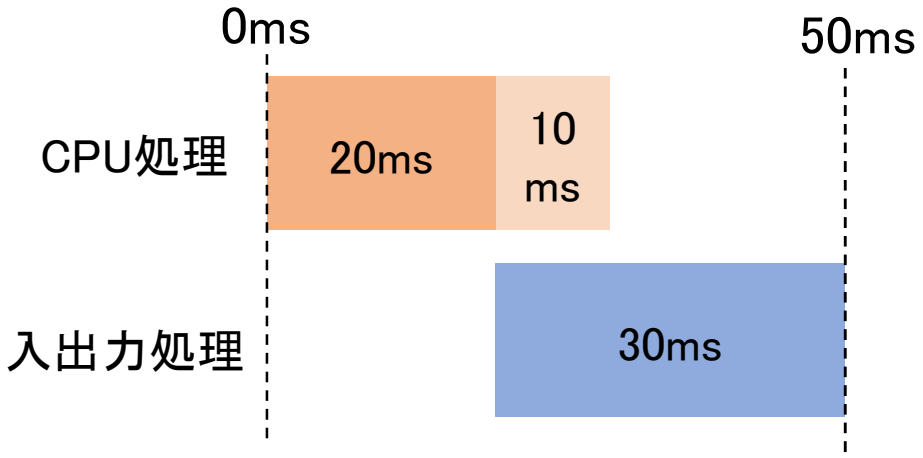
● タイムチャートの作成例(2)



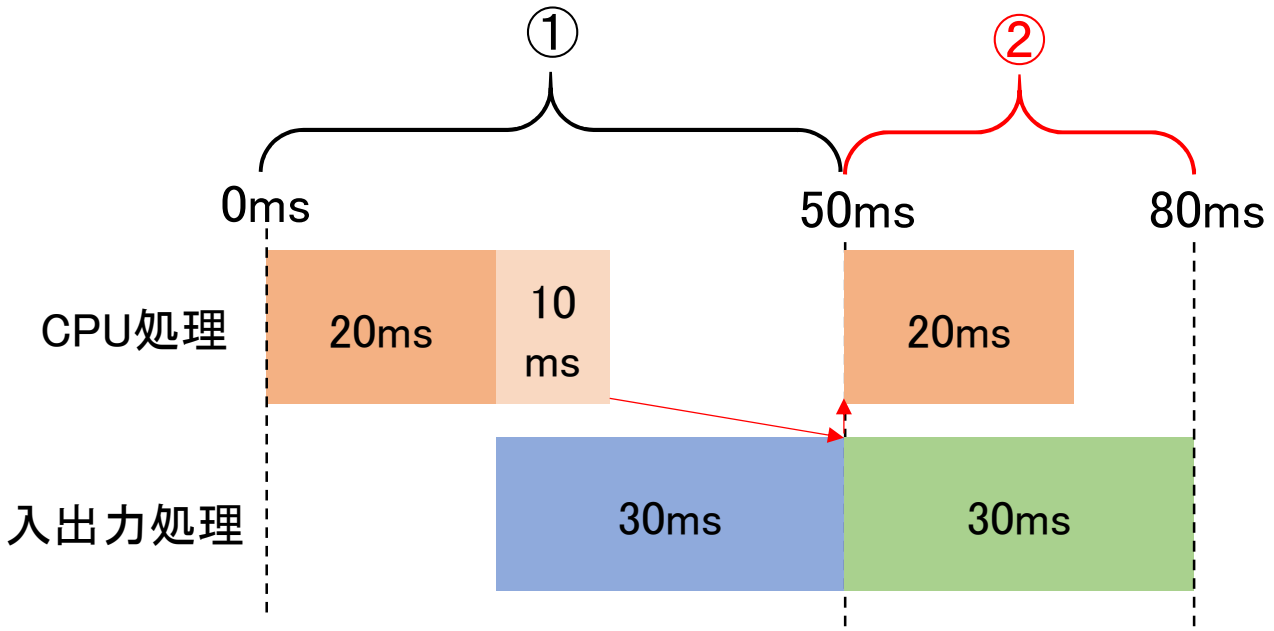
マルチプログラミングの実行条件

- タスクの実行優先順位は、タスクAの方がタスクBより高い
- タスクA及びタスクBは、同じ入出力装置を使う
- CPU処理を実行中のタスクは、入出力処理が終了するまでは、実行を中断しない
- 入出力装置は、入出力処理が終了するまでは、実行を中断しない

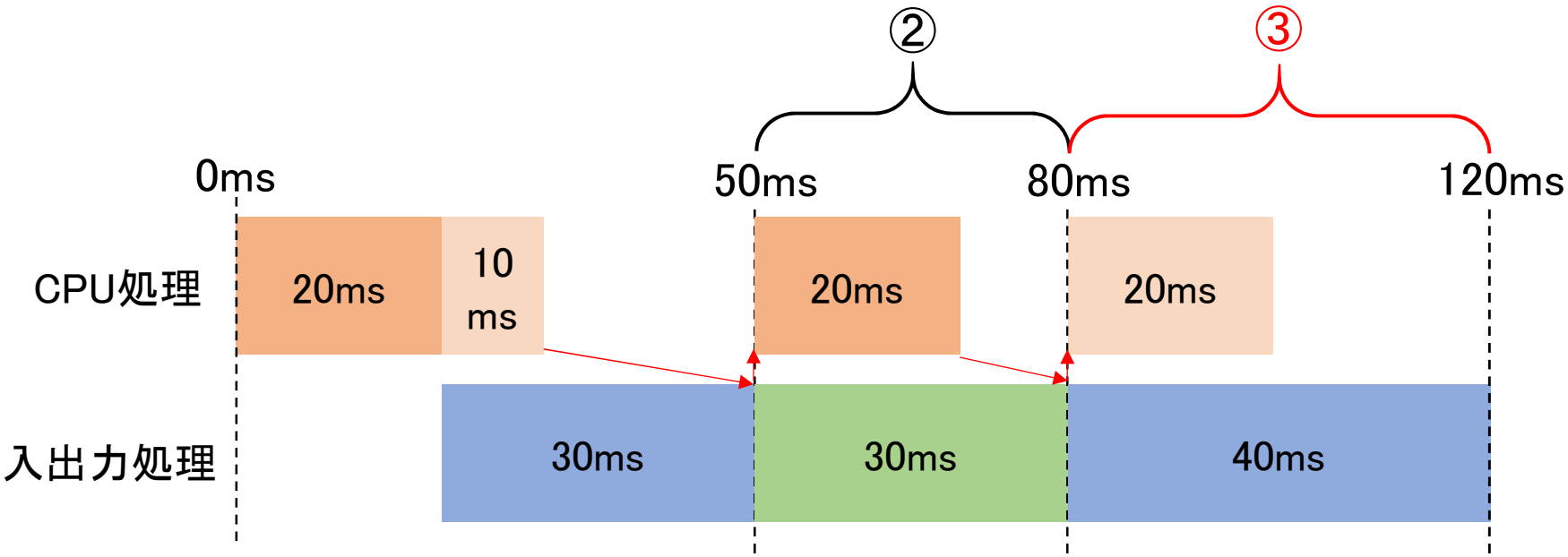
① タスクAがCPUを20ms使用した後、タスクBがCPUを10msすると同時にタスクAは、入出力処理を30ms使用する



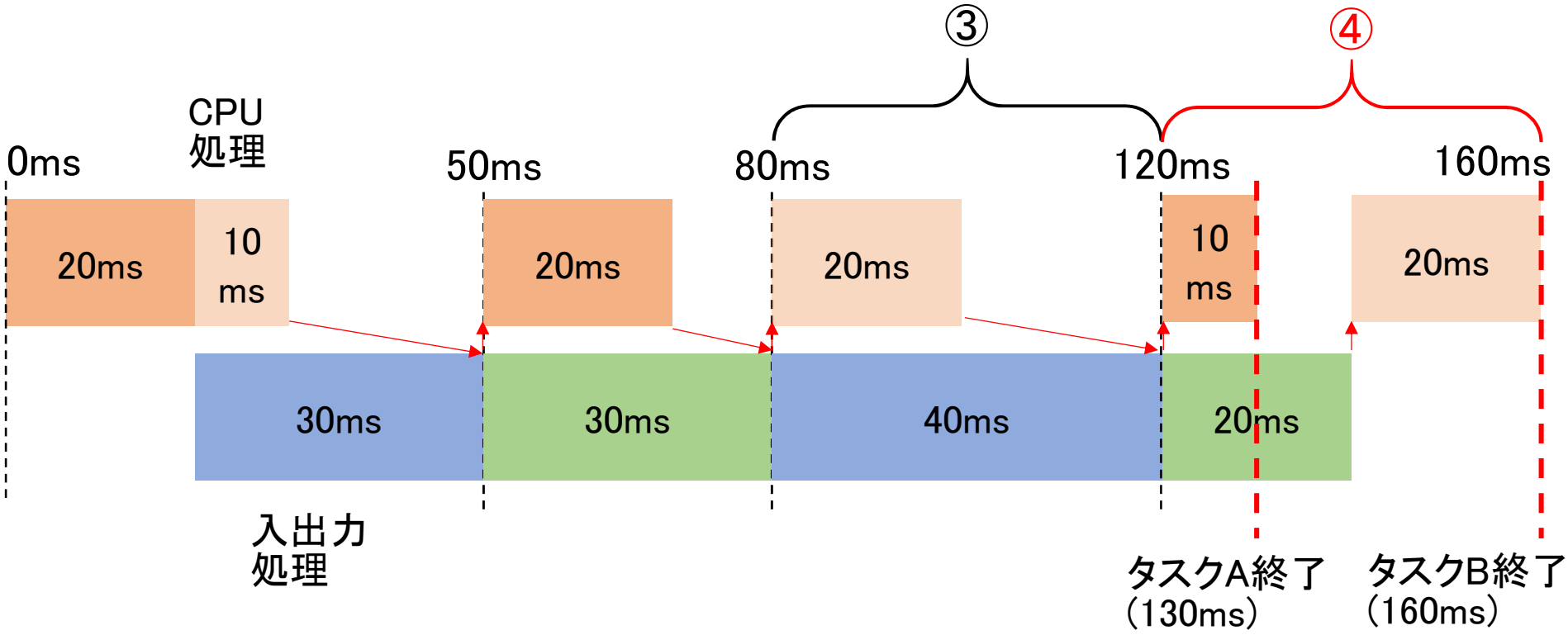
② タスクBが入出力処理を実行しようとするが、タスクAが入出力処理を実行中なので、待たされる。
一方、タスクAは入出力処理の実行後、CPU処理を20ms実行すると同時に、タスクBが入出力処理を30ms実行する



③ タスクAが入出力処理を実行しようとするが、タスクBが入出力処理を実行中なので、待たされる。
一方、タスクBは入出力処理の実行後、CPU処理を20ms実行すると同時に、タスクAが入出力処理を40ms実行する



④ タスクBが入出力処理を実行しようとするが、タスクAが入出力処理を実行中なので、待たされる。
一方、タスクAは入出力処理の実行後、CPU処理を10ms実行後に終了。同時にタスクBが入出力処理を20ms実行後に、CPU処理を20ms実行して終了。



CPUの使用率

$$\text{CPUの使用率} = \frac{CPU\text{の全使用時間}}{\text{タスクの実行時間}}$$

タスクA
CPUの使用時間

タスクB
CPUの使用時間

$$= \frac{50ms + 50ms}{160ms}$$

$$= 0.625 = 62.5\%$$

■ 過去問題1

OSのスケジューリング方式に関する記述のうち、適切なものはどれか？

ア：処理時間順方式では、既に消費したCPU時間の長いジョブに高い優先度を与える

イ：到着順方式では、ラウンドロビン方式に比べて特に処理時間の短いジョブの応答時間が短くなる

ウ：優先度順方式では、一部のジョブの応答時間が極端に長くなることがある

エ：ラウンドロビン方式では、ジョブに割り当てるCPU時間(タイムクウォンタム)を短くするほど、到着順方式に近づく

ア：処理時間順方式は、処理時間の短いジョブに対して高い優先度を与え、最初に実行する方式

イ：ラウンドロビン方式では、実効可能待ち行列に並んだジョブに対して順番に一定時間のCPU使用権を与える。このため処理時間の短いジョブは、先に実行可能待ち行列に並んでいたジョブよりも早く終了する可能性がある。

　一方、到着順方式では、実効可能待ち行列の最後でほかのジョブの終了を待ってから実行する。

　つまり到着順方式では、ラウンドロビン方式に比べて特に処理時間の短いジョブの応答時間が長くなる

エ：ラウンドロビン方式は、実効可能状態になった順番(到着順)で、ジョブに一定時間のCPU使用権を割り当てる。つまりタイムクウォンタムを長くするほど到着順方式に近づいていき、タイムクウォンタムを無限に長くすれば到着順方式と全く同じになる。

■ 過去問題2

CPUと磁気ディスク装置で構成されるシステムで、下表に示すジョブA,Bを実行する。この二つのジョブが実行を終了するまでのCPUの使用率と磁気ディスク装置の使用率との組合せのうち、適切なものはどれか？

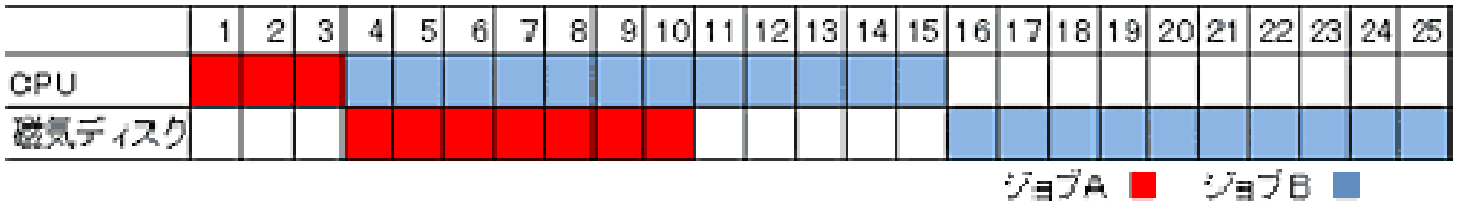
ここで、ジョブA、Bはシステムの動作開始時点ではいずれも実行可能状態にあり、A,Bの順で実行される。CPU及び磁気ディスク装置は、ともに一つの要求だけを発生順に処理する。ジョブA,Bとも、CPUの処理を終了した後に磁気ディスク装置の処理を実行する。

単位 秒

ジョブ	CPU の処理時間	磁気ディスク装置の処理時間
A	3	7
B	12	10

	CPU の使用率	磁気ディスク装置の使用率
ア	0.47	0.53
イ	0.60	0.68
ウ	0.79	0.89
エ	0.88	1.00

問題文の条件を考慮して、CPUと磁気ディスク装置の使用状況を図にすると以下のようになる



2つのジョブが終了するまでの時間は25秒で、そのうち赤色及び青色が付いている部分が使用している時間

赤色=3秒
青色=12秒

CPUの使用率は、**15(秒)/25(秒)=0.60=60%**

磁気ディスクディスクの使用率は、**17(秒)/25(秒)=0.68=68%**

赤色=7秒
青色=10秒

4.2 記憶管理と同期・排他制御

4.2.1 主記憶管理
4.2.2 仮想記憶システム
4.2.3 同期・排他制御

4.2.1 主記憶管理

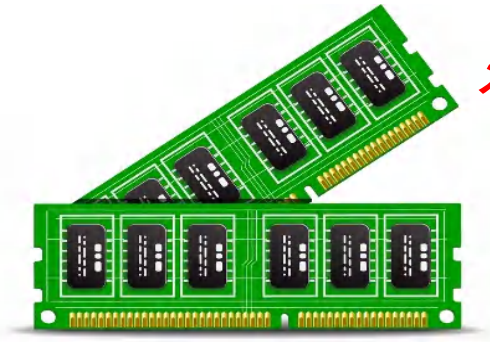
役割
- 主記憶を効果的に使用する
- 容量の制約をカバーする

補助記憶装置を記憶空間として使用

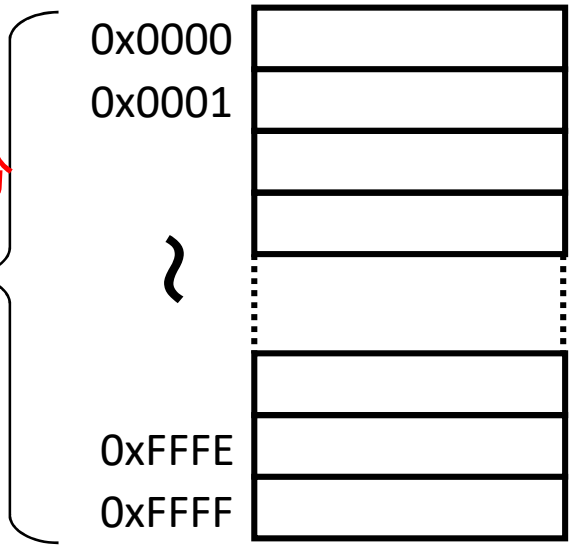
● アドレス空間
アドレスを指定することによって、アクセスできるメモリの範囲

- 物理アドレス
  主記憶上に割り当てられた実際のアドレス
- 論理アドレス
  プログラムの中で使用するアドレス（通常はプログラムの先頭がゼロ番地）

物理アドレス

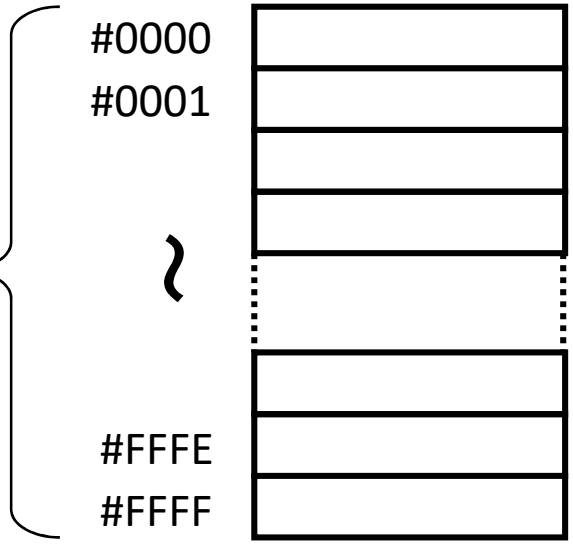


メモリの大きさ分



論理アドレス

プログラムの大きさ



● 主記憶の管理方式

主記憶を効率よく使用するための管理方式

• オーバーレイ方式

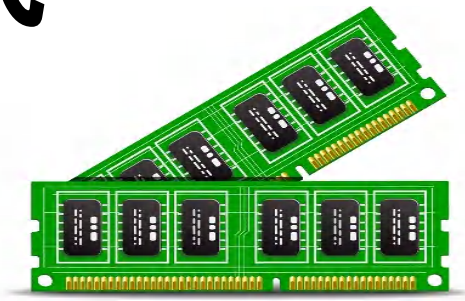
主記憶の容量が、プログラムの容量よりも小さい場合に使用する方式。

プログラムをセグメントと呼ぶ単位に分割し、必要な分だけ主記憶に読込んで実行する。

オーバレイ方式

給与プログラム

セグメント1 （プログラムの制御）
セグメント2 （社員の給与計算）
セグメント3 （稼働時間の集計）
セグメント4 （給与明細の印刷）



給与計算

読込み

給与印刷

セグメント1 （プログラムの制御）
セグメント2 （社員の給与計算）
セグメント3 （稼働時間の集計）

セグメント1 （プログラムの制御）
セグメント4 （給与明細の印刷）

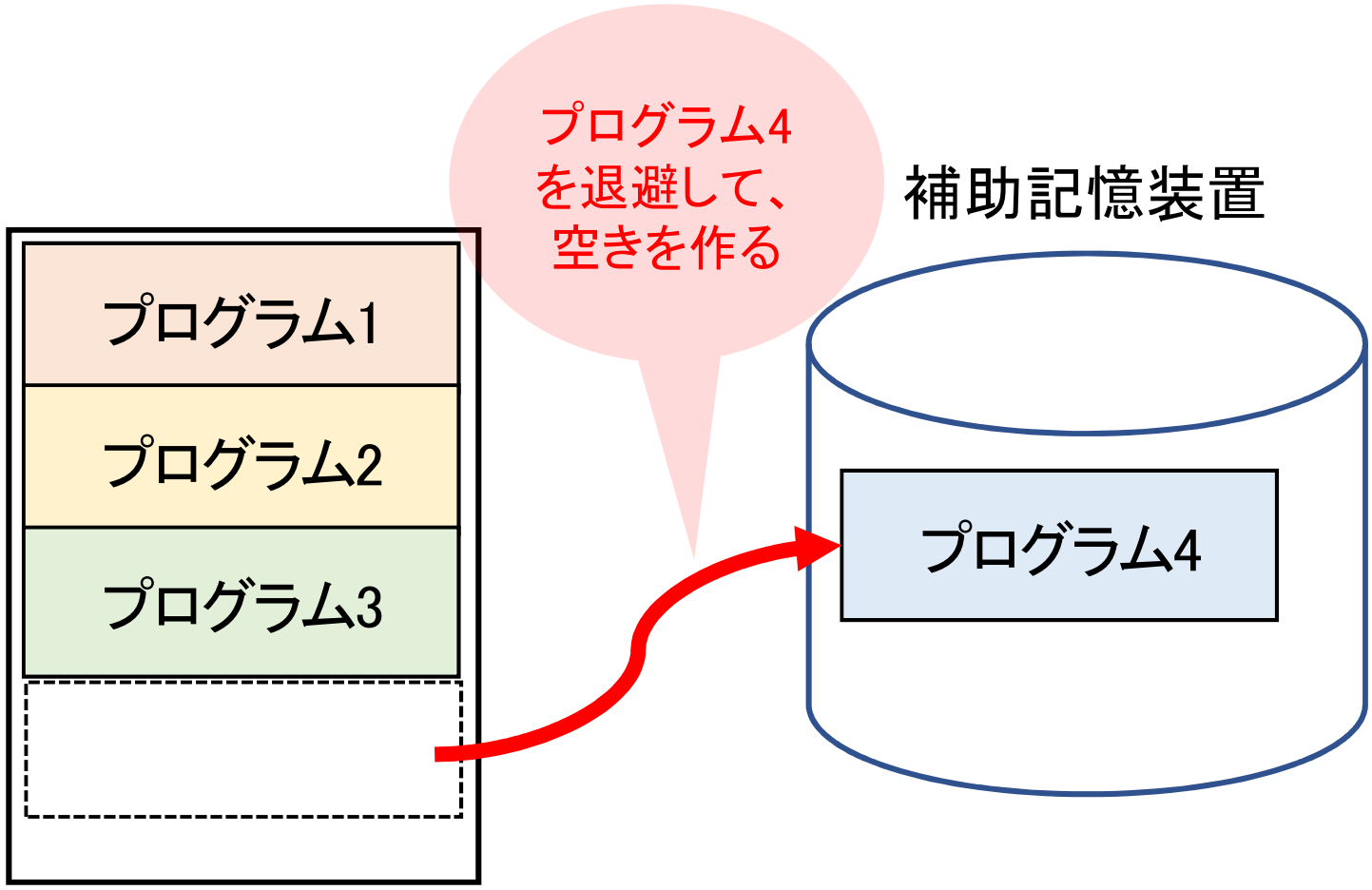
- **スワッピング方式**

　マルチプログラミング環境で、優先度が高いプログラムによって割込みなどが発生した場合、現在実行中のプログラムを中断して、優先度が高いプログラムを主記憶に読込む。

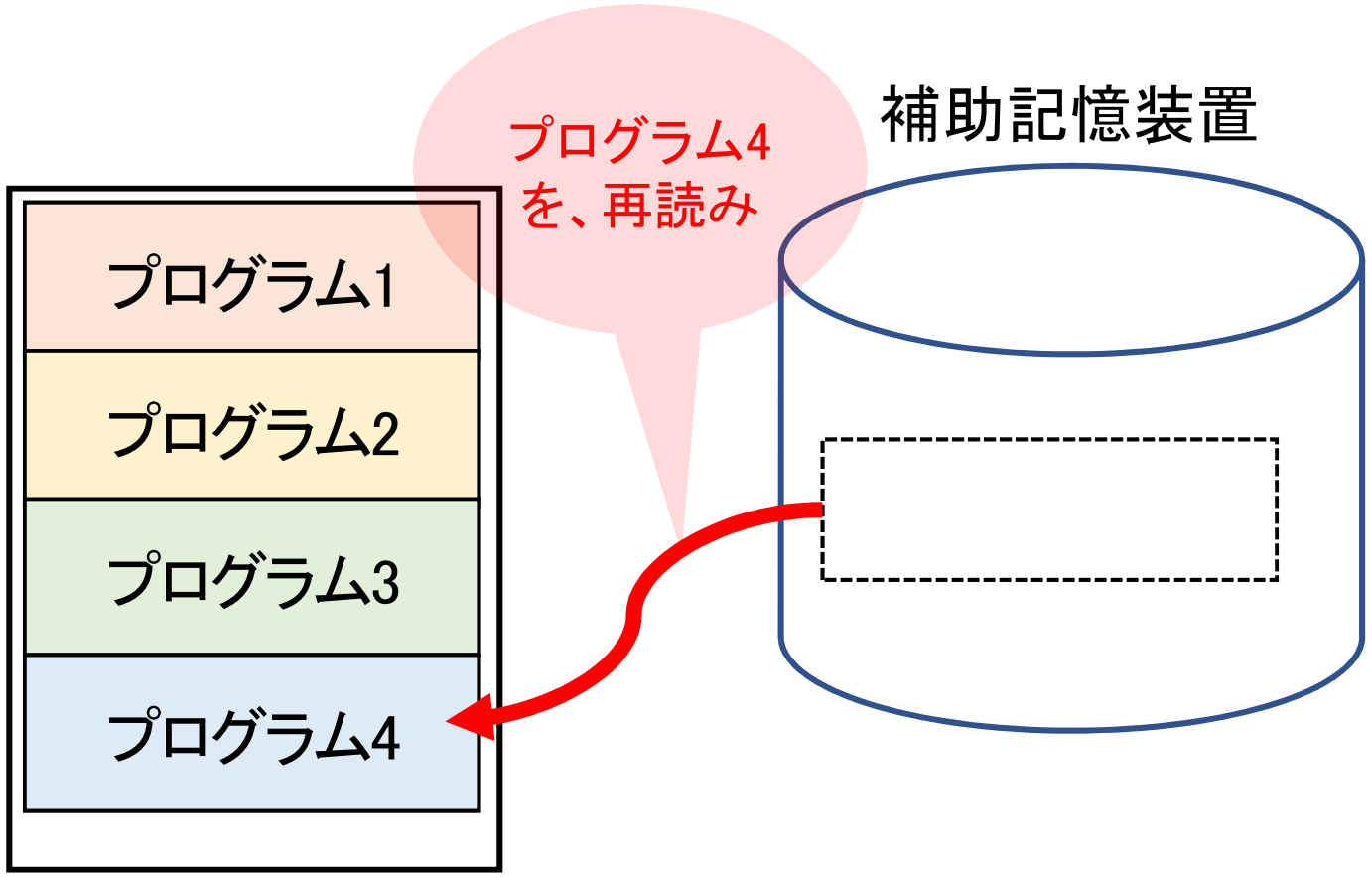
　こうしたことが頻繁に起こると、中断したプログラムが主記憶内に蓄積し、肝心の実行すべきプログラムが実行できないことがある。

　このとき、中断したプログラムを補助記憶装置(HDDなど)に退避（スワップアウト）し、再びCPUの使用権が与えられたときには、退避したプログラムを補助記憶装置から主記憶へ戻す（スワップイン）

スワッピング方式（スワップアウト）

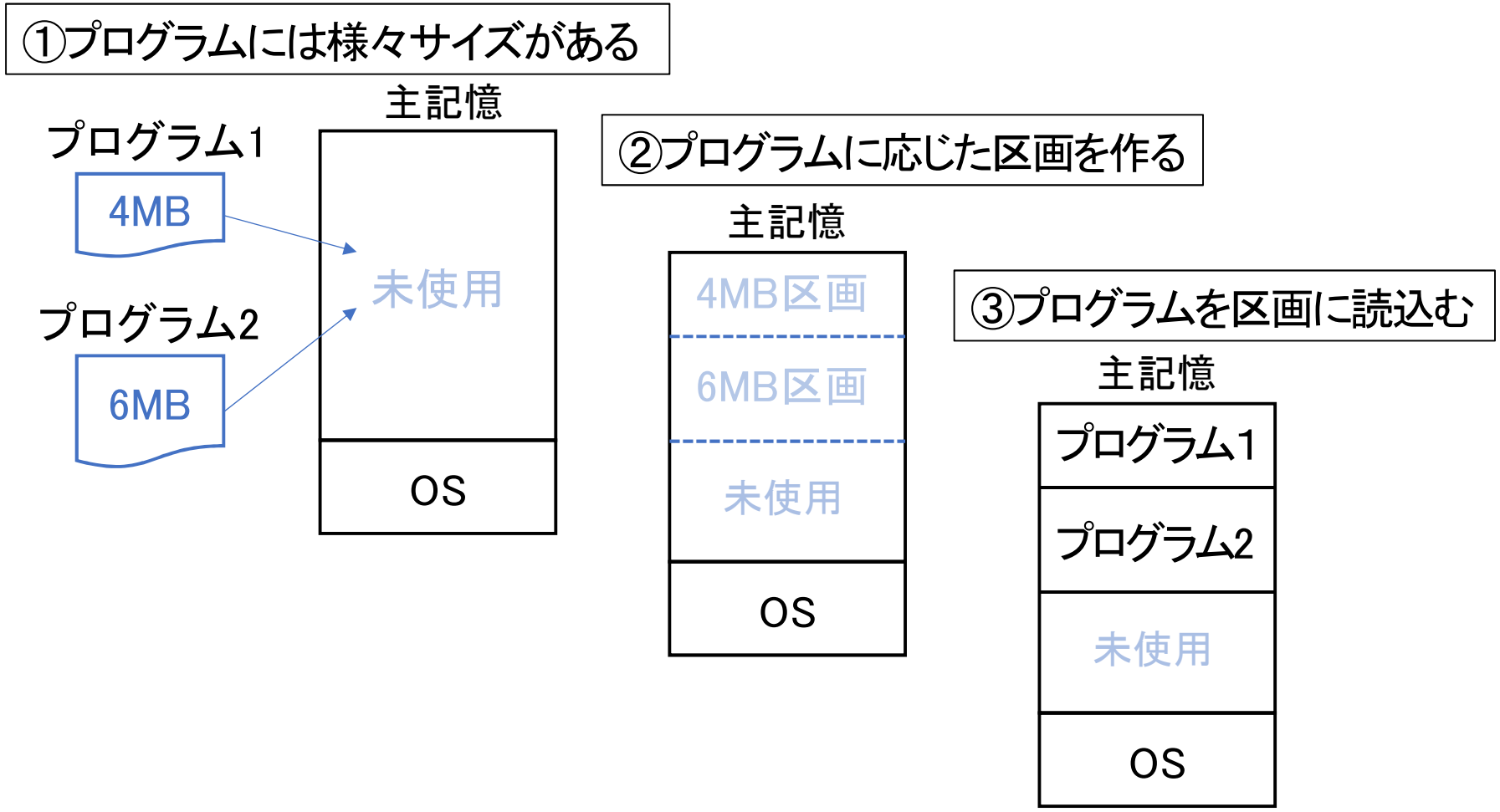


スワッピング方式（スワップイン）

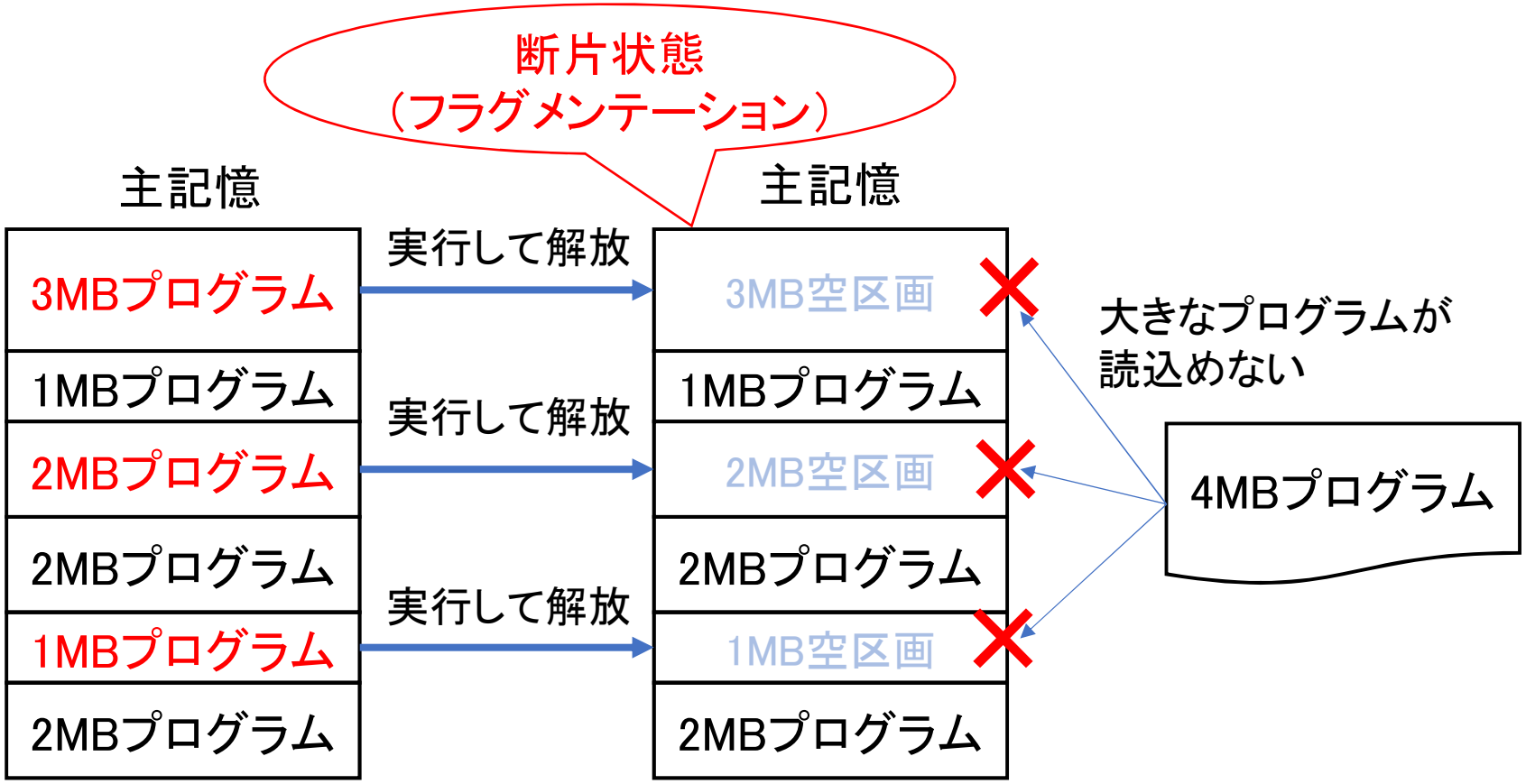


- **可変区画方式**

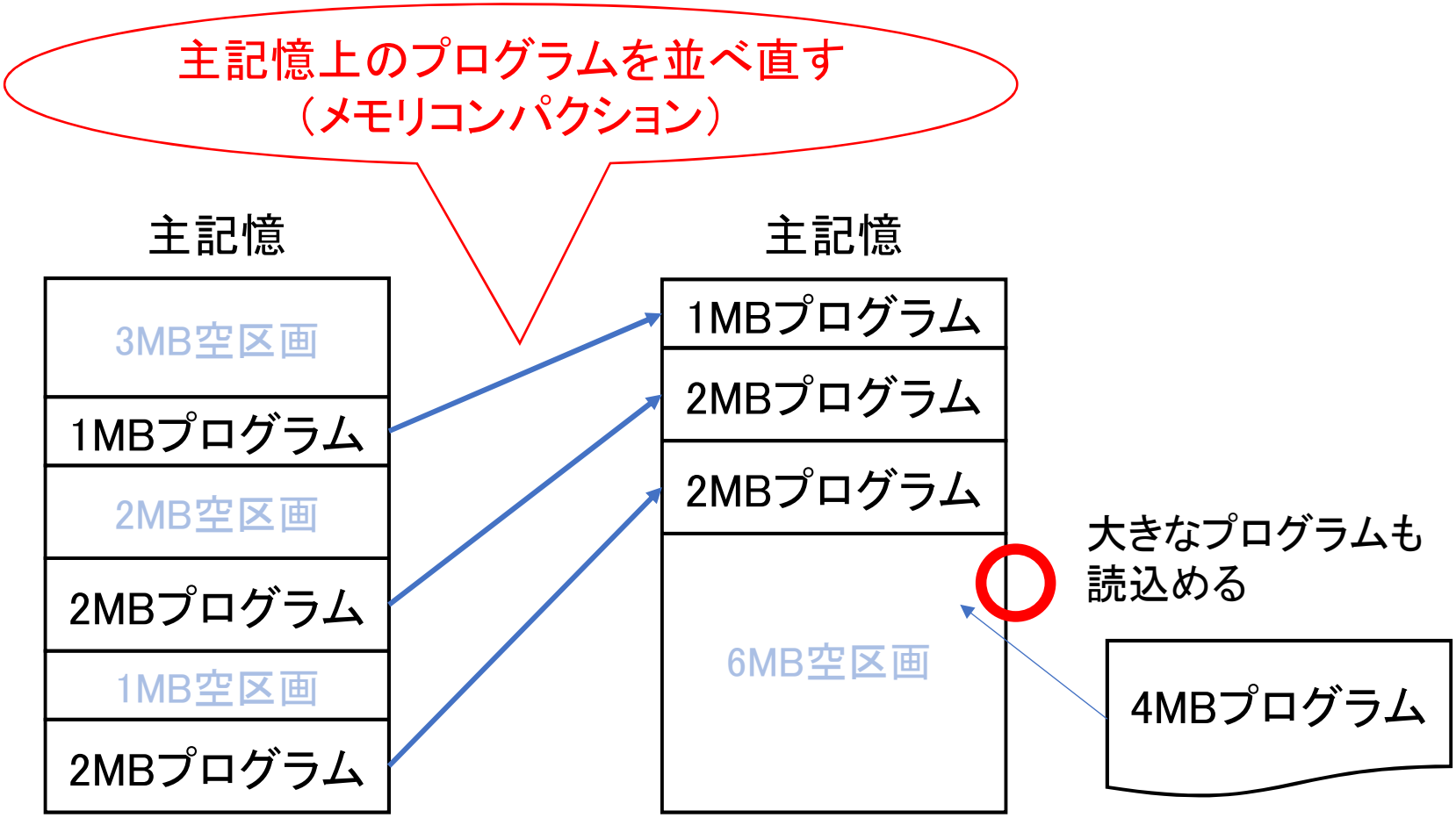
  プログラムを主記憶に読込むとき、プログラムの大きさに応じて、主記憶の未使用領域を区切る



可変区画方式は、区画に無駄がないので、主記憶の利用効率がよい。しかし、大きさが異なるプログラムを実行していくと、主記憶が断片化する（フラグメンテーション）

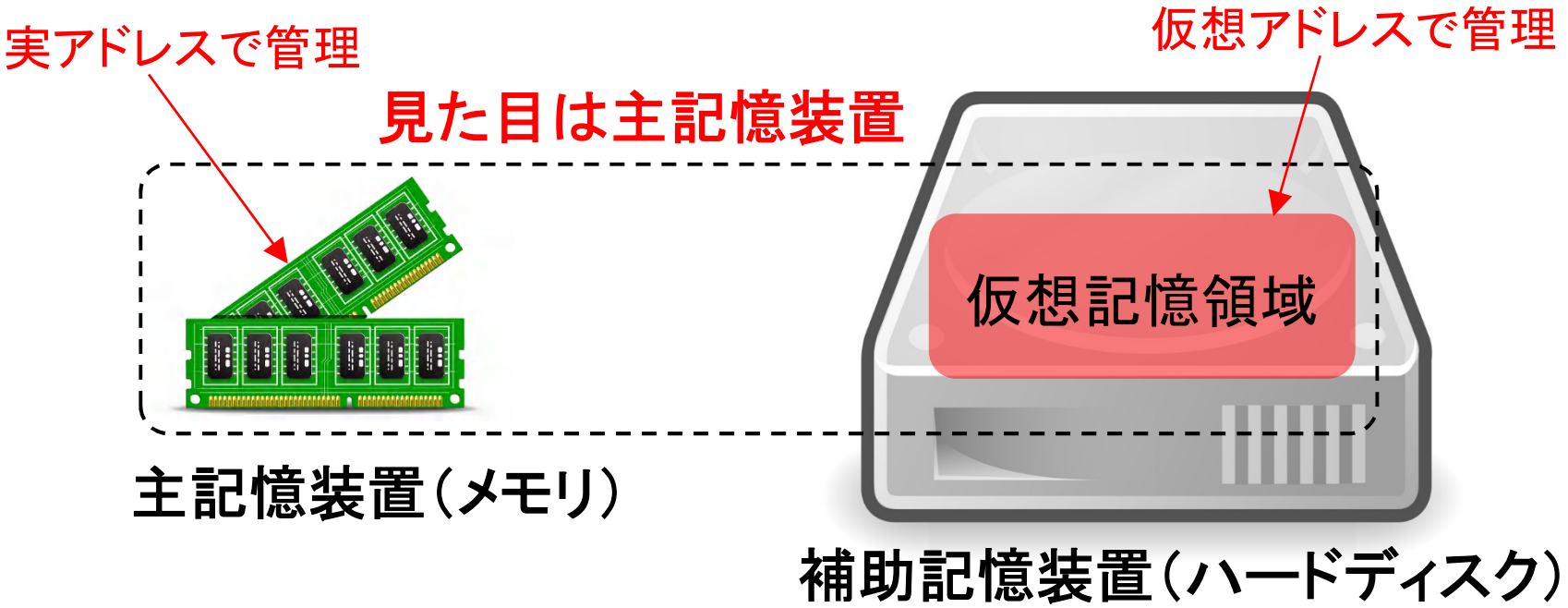


フラグメンテーションを解決するためには、読込まれているプログラムを再配置して、断片状態の空区画を、連続した区画にする（メモリコンパクション）



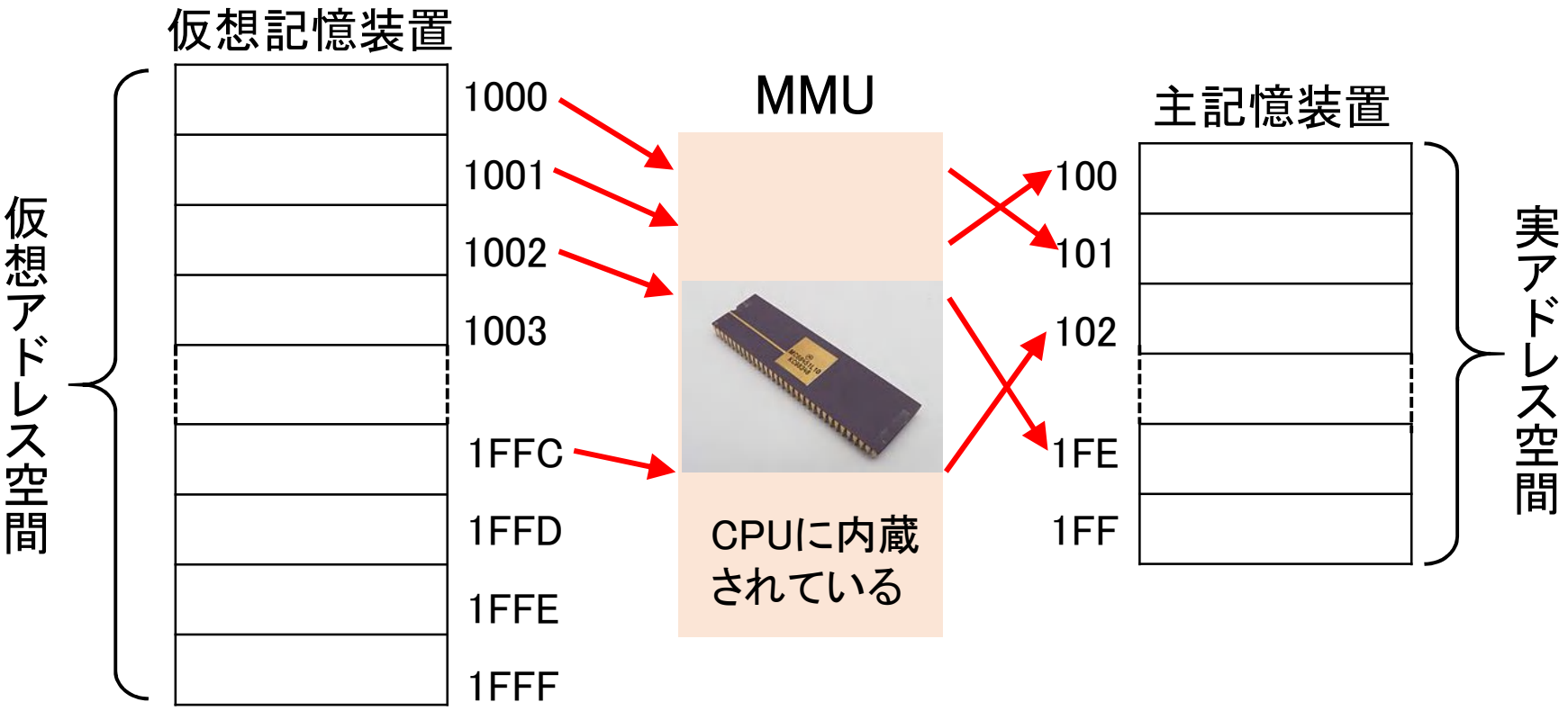
4.2.2 仮想記憶システム

主記憶の容量以上の記憶（メモリ）空間を実現するシステム。主記憶は実アドレス（空間）、仮想記憶は仮想アドレス（空間）で、記憶内容を管理する。



● 仮想記憶の実現方式

プログラムの実行する部分だけを、実記憶（主記憶）に保存する。
このため、仮想記憶アドレスから実アドレスに変換する必要がある。
このアドレス変換（動的アドレス変換と呼ぶ）は、メモリ変換ユニット
(MMU : Memory Management Unit)と呼ぶ、ハードウェアが担当する。

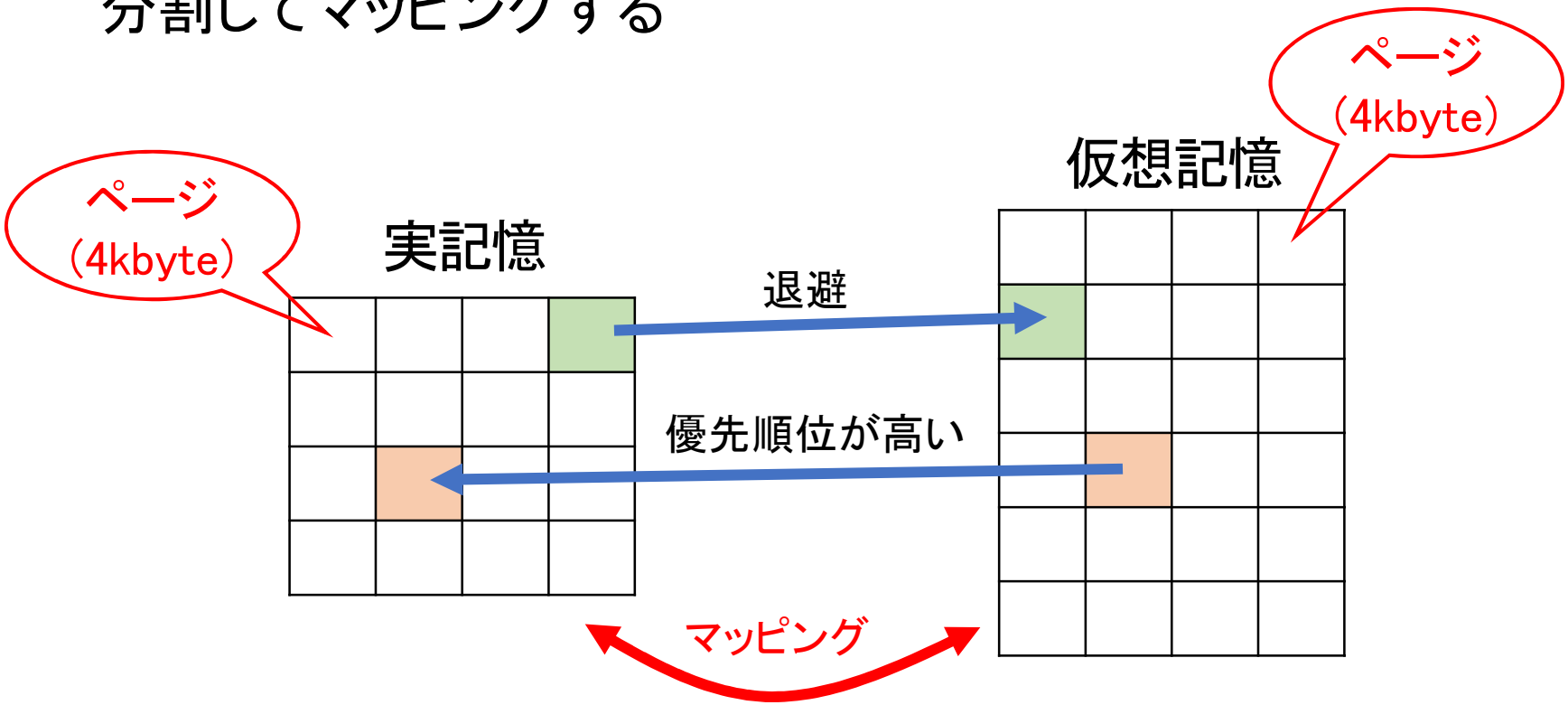


● マッピング方式

実記憶と仮想記憶の間で、プログラムやデータを対応付けること

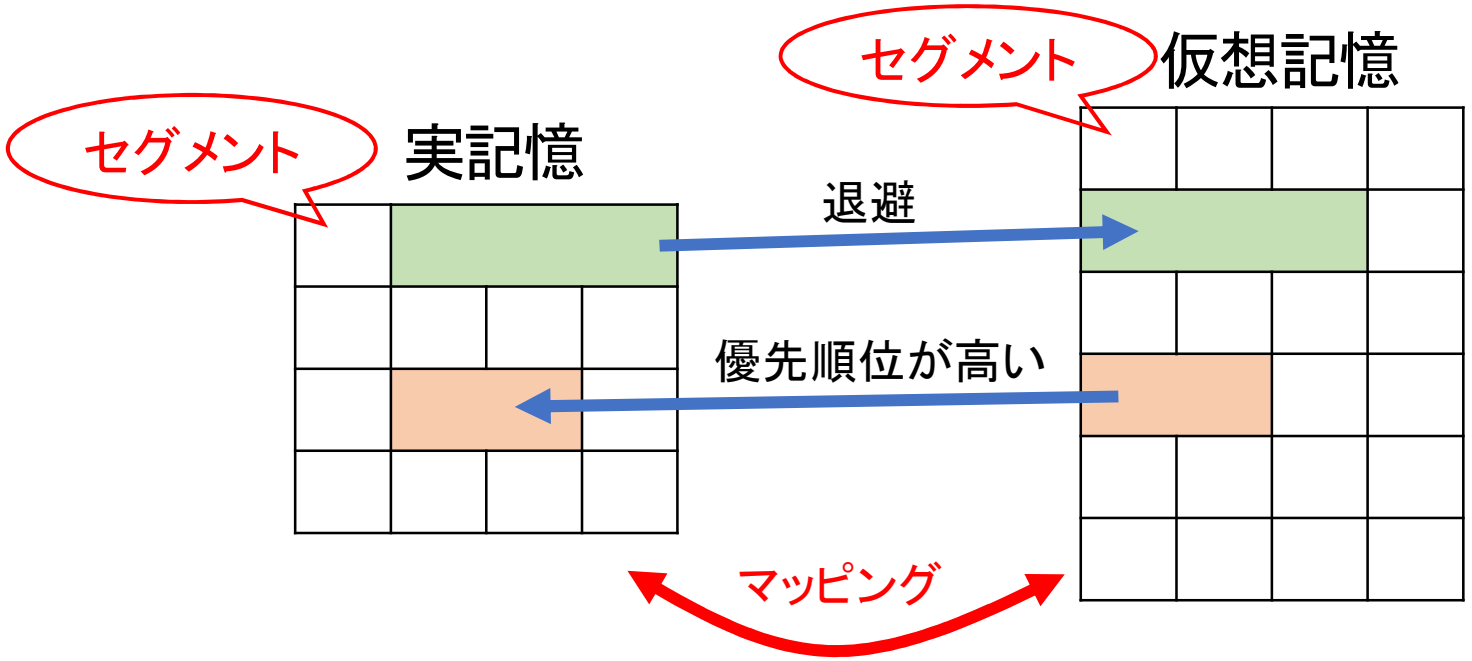
1ページ
（512byte～4kbyte）

- ページング方式
  プログラムやデータを、固定長のページと呼ぶ単位に分割してマッピングする



タスクで使うメモリの連続領域

- セグメント方式
  プログラムやデータを、可変長のセグメントと呼ぶ単位に分割してマッピングする

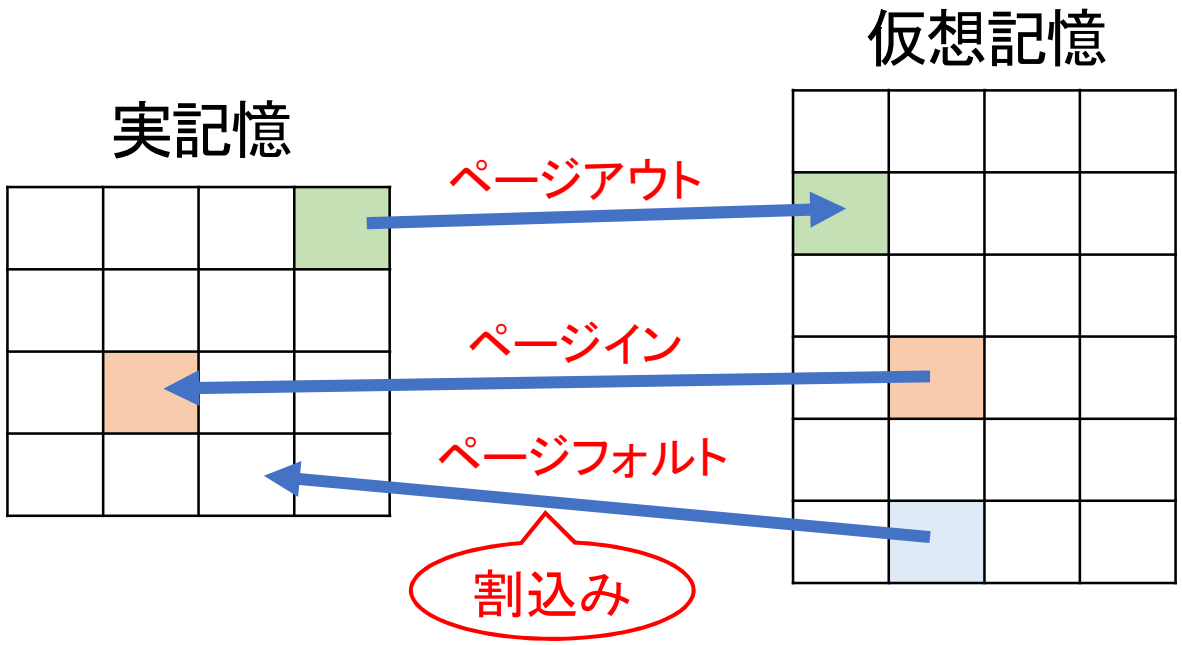


※ページング方式とセグメント方式を組み合わせた方式を、
セグメンテーションページング方式と呼ぶ

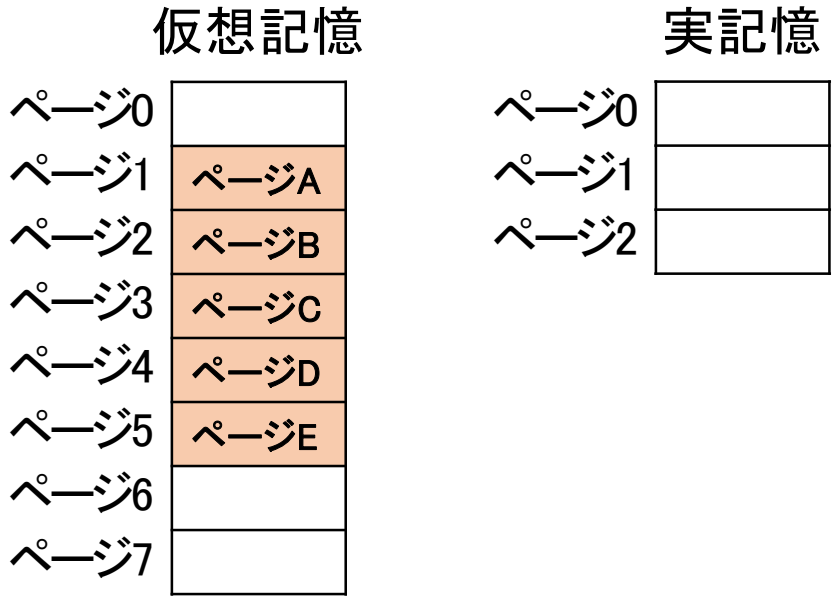
● ページングアルゴリズム

ページング方式において、実記憶で不要になったページを決定する方法。

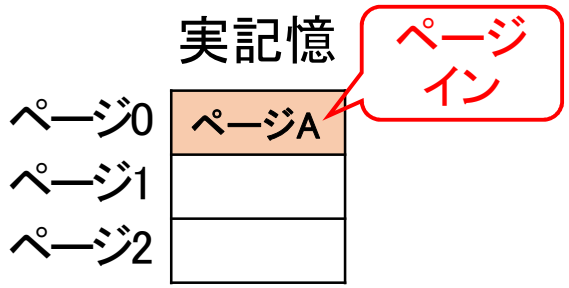
- プログラム実行中に必要なページが実記憶にない状態 ⇒ “ページフォルト“と呼ぶ割込みが発生
- 不要なページを仮想記憶に追い出す ⇒ ページアウト
- 必要なページを仮想記憶から主記憶に読込む ⇒ ページイン



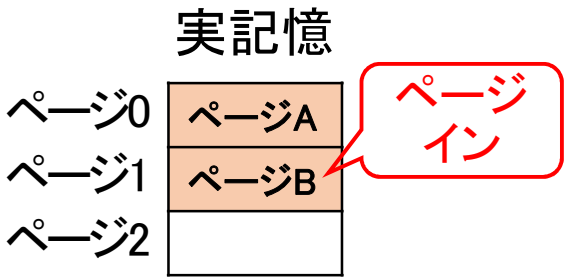
次に示すような実記憶と仮想記憶において、
仮想記憶のページ番号を 1→ 2 → 3 → 1 → 5
の順番に参照したとする



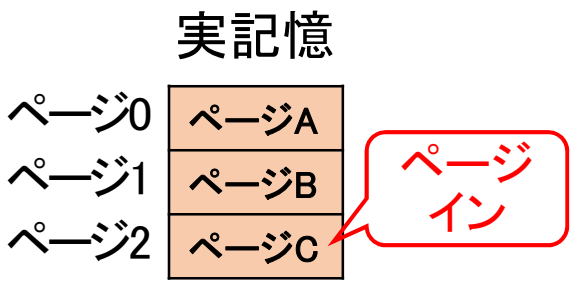
①ページ1を参照する



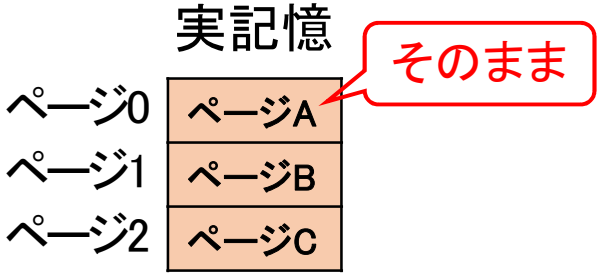
②ページ2を参照する



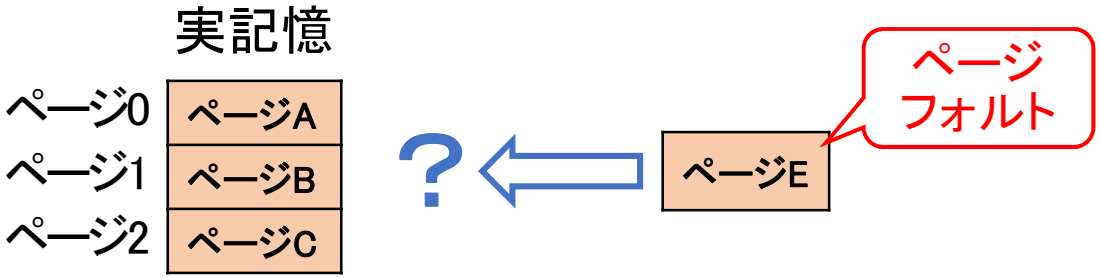
③ページ3を参照する



④ページ1を参照する

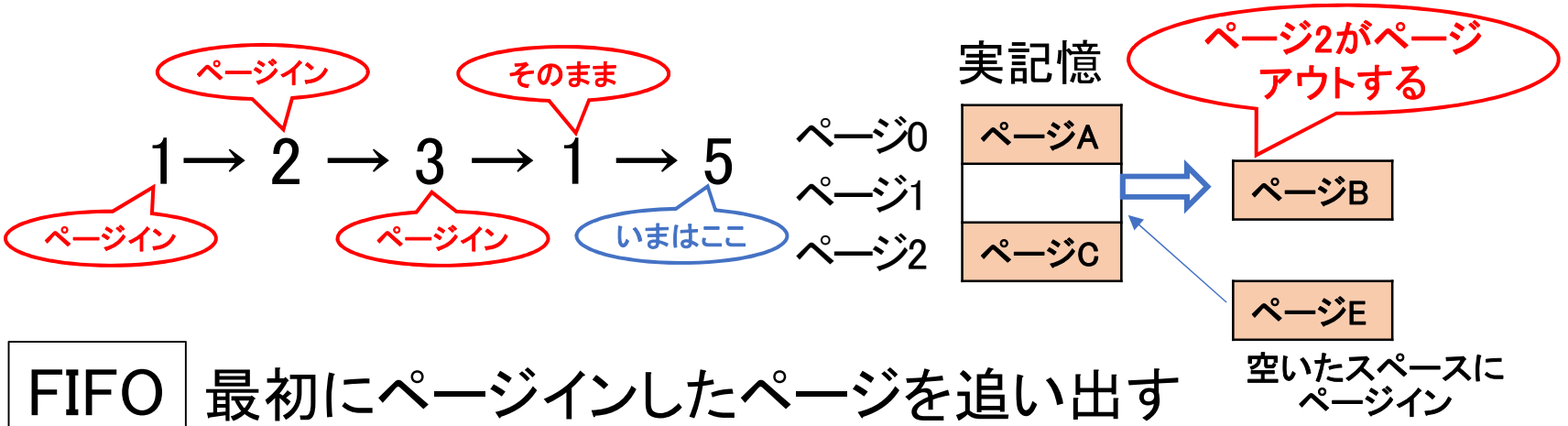


⑤ページ5を参照する

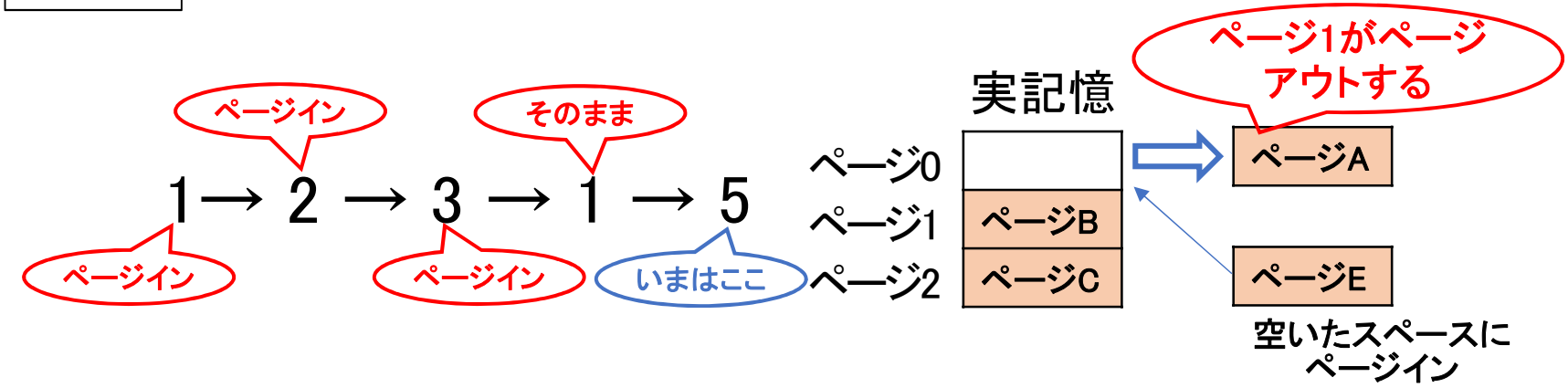


ページングアルゴリズムには、LRU(Least Recently Used)やFIFO(First In First Out)などがある

LRU もっとも長い間、参照していないページを追い出す



FIFO 最初にページインしたページを追い出す

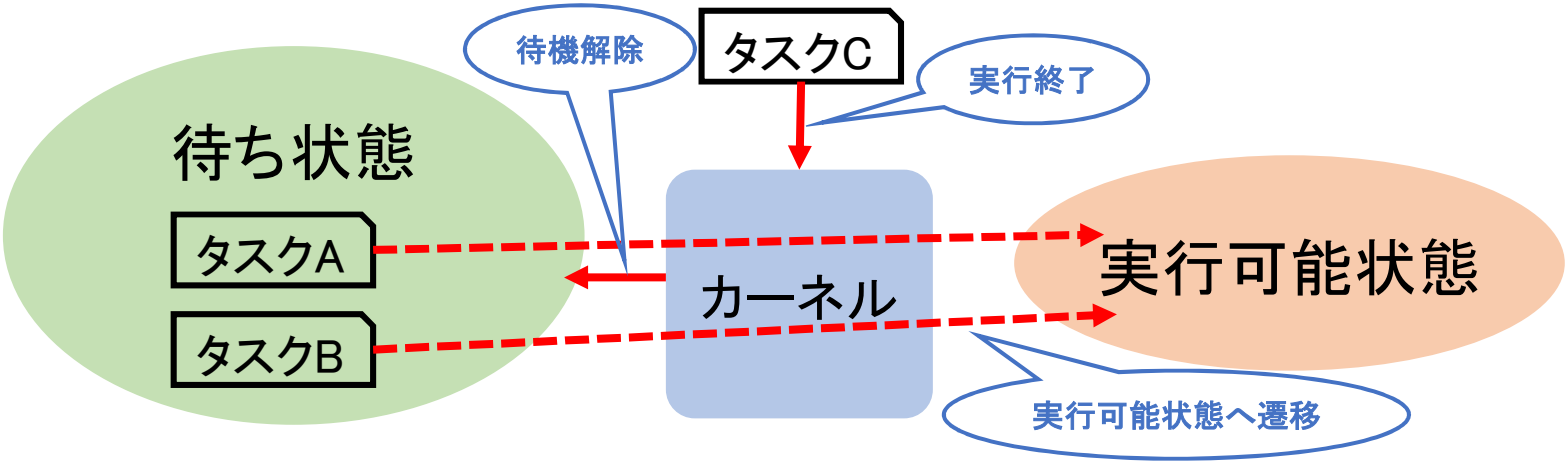


4.2.3 同期・排他制御

タスクが複数同時に実行する環境では、お互いの干渉を避けるための排他制御や協調動作するための同期制御が必要となる

- 同期制御

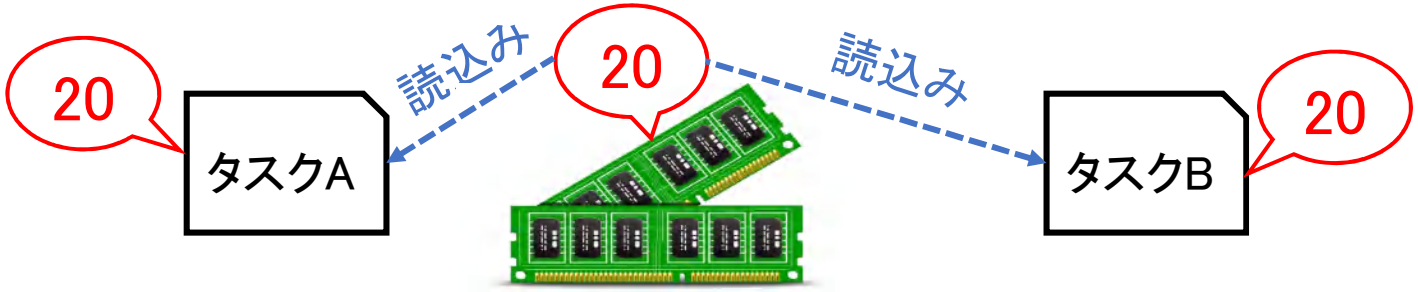
タスク同士がお互いに依存関係を持ち、一方の処理の終了を待って、他方が実行を再開するような協調動作をするための実行タイミングを図る仕組み



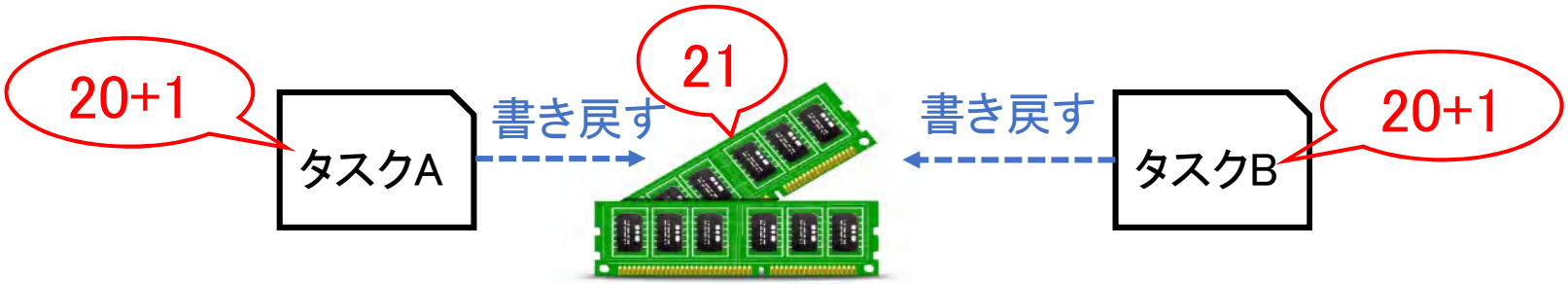
● 排他制御

メモリなどの共有資源を、同時に複数のタスクから内容の更新処理（こうした処理をクリティカルセッションと呼ぶ）をしようとすると、実行状態の複数のタスクは、CPUの使用権を奪われて、実行可能状態（実行中断）になる。こうした状態にならないように、複数のタスクを排他制御する。

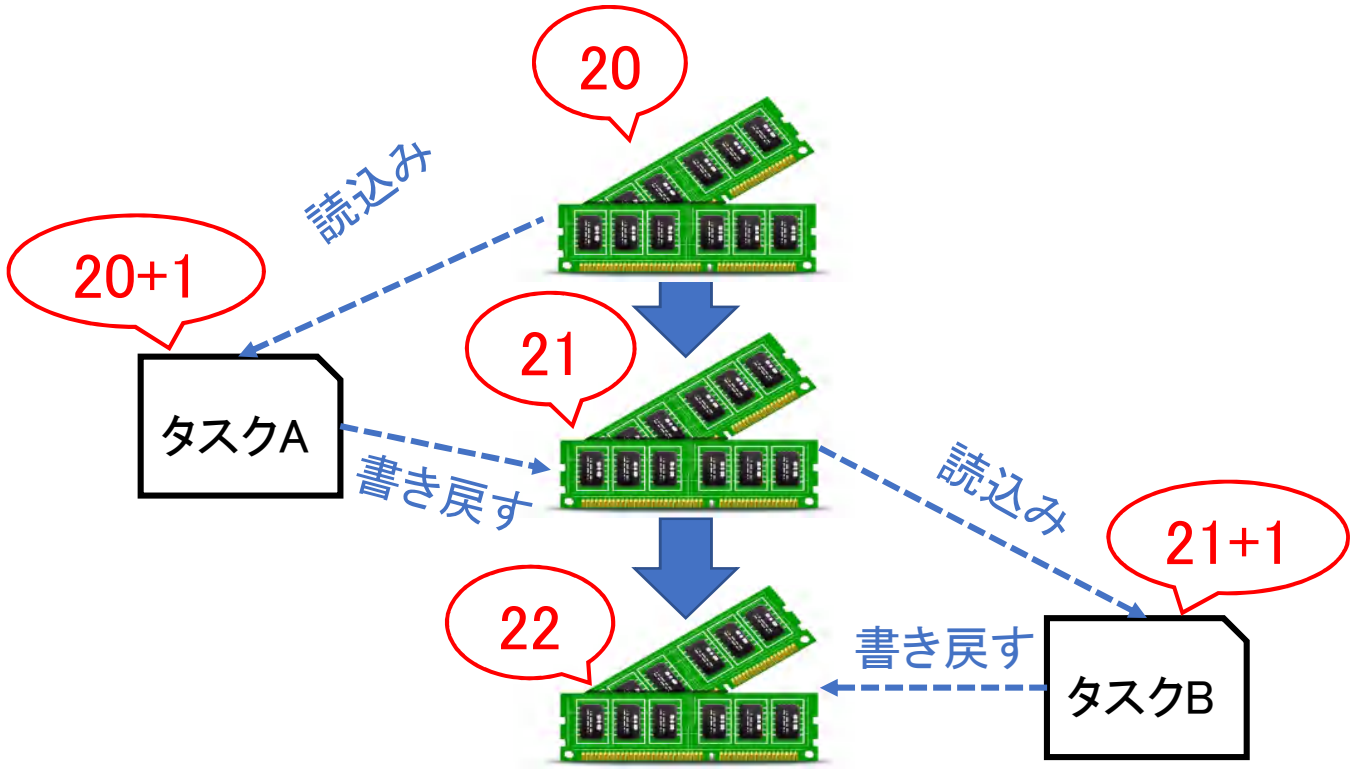
①タスクAとタスクBの両方からアクセスできる領域に数値20が入っている



②両タスクでは、読取った数値に1を加算して書き戻す処理をする



③本来であればメモリには、最後に数値22が入っているべき

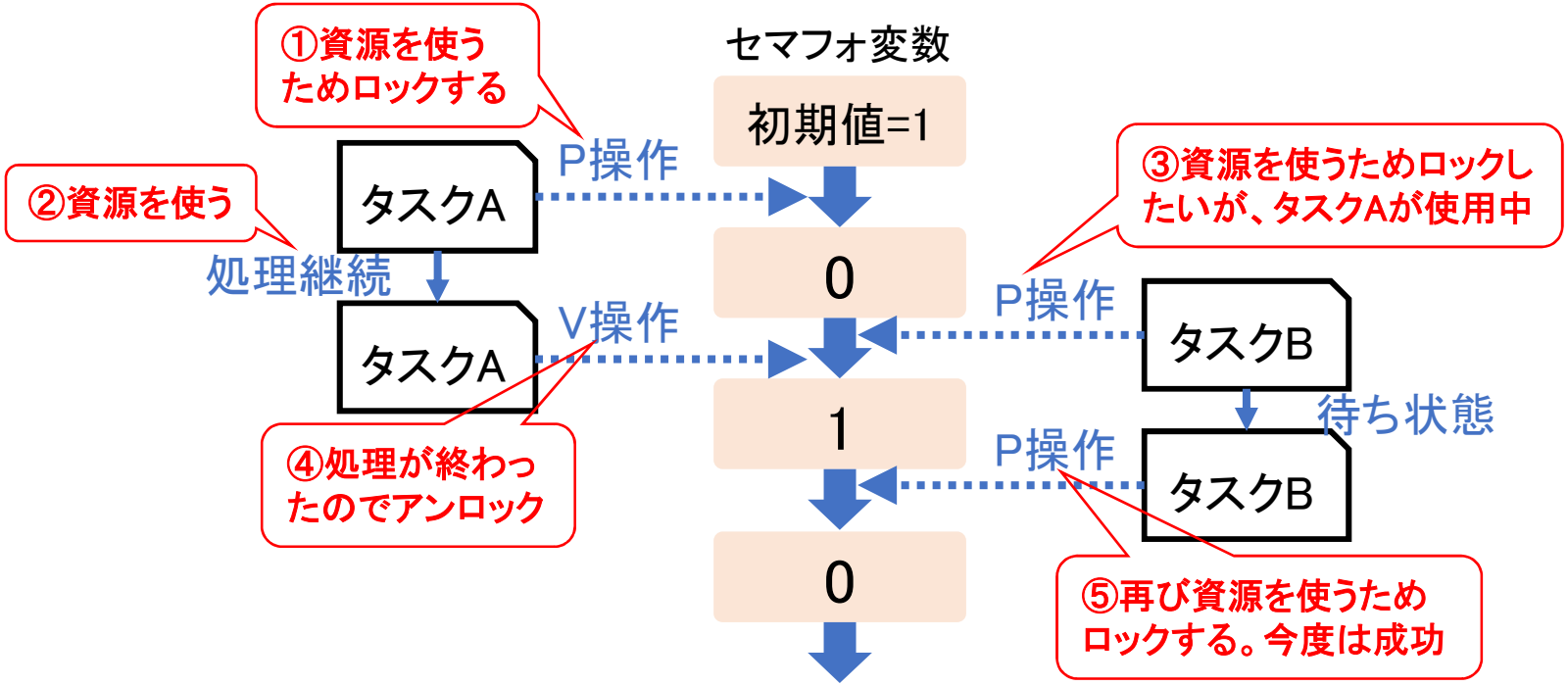


②では片方の処理がない状態が起こる

- セマフォ

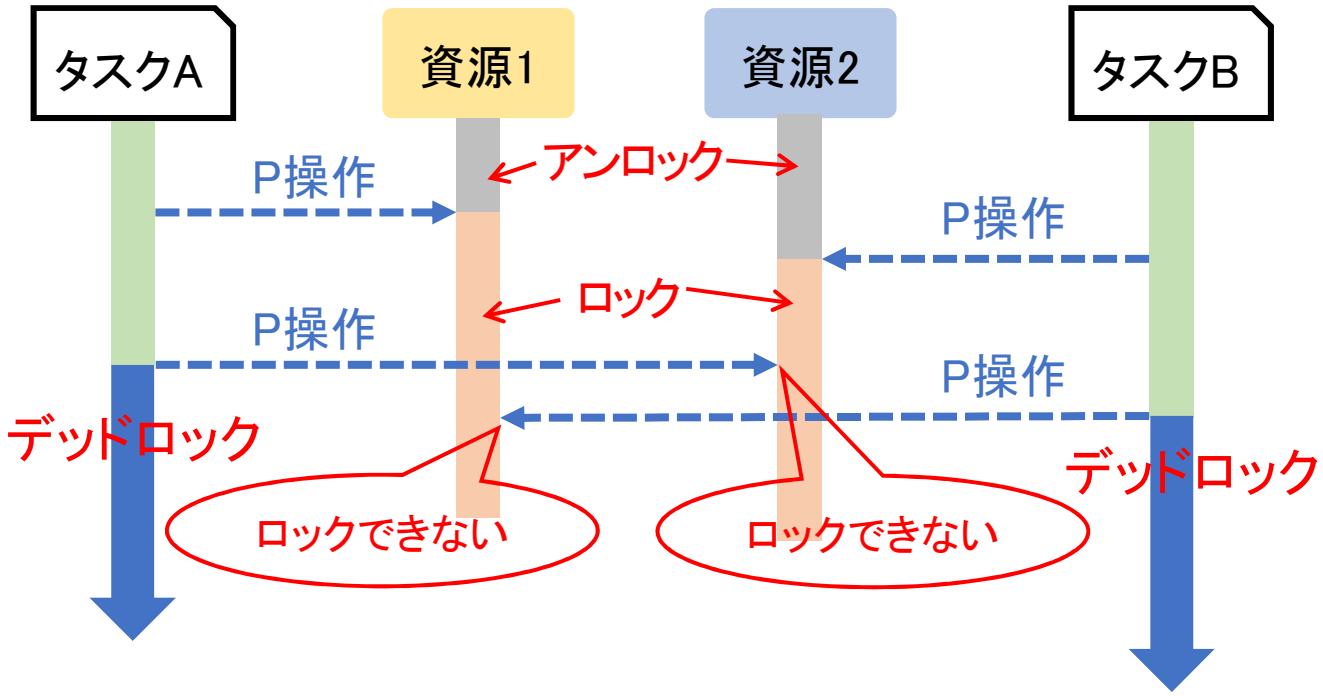
排他制御の方法の1つで、コンピュータ上の共有資源について、利用可能な資源の数を指し示している。プログラムが資源を占有するときには、セマフォの値から1を減じ、処理が終わって解放するときには、1を加える。セマフォが0のときは空いている資源がないため、正の値になるまで待機する。

P操作：セマフォ変数値を1減算し、資源をロック
V操作：セマフォ変数値を1加算し、資源をアンロック



● デッドロック

複数のタスクが互いに相手の占有している資源の解放を待ってしまい、処理が停止してしまう



デッドロックの発生を避けるためには、資源をロックする順番を同じにする

- タスクA：資源1 → 資源2　タスクB：資源1 → 資源2
- タスクA：資源2 → 資源1　タスクB：資源2 → 資源1

■ 過去問題3

仮想記憶方式におけるプログラムやデータの格納方法に関する記述のうち、適切なものはどれか？

ア：一つのプログラムや一連のデータは，主記憶装置及び補助記憶装置で必ず連続した領域に格納される

イ：頻繁に参照されるプログラムやデータが主記憶装置に格納されているので，仮想記憶を用いない場合に比べて主記憶の平均アクセス時間が短くなる

ウ：プログラムやデータを補助記憶装置に格納し，必要に応じて主記憶に読み込むので，主記憶の見かけの容量を拡大できる

エ：ページアウトされたプログラムやデータがシステムの停止後も補助記憶装置に保持されるので，再起動後に主記憶の内容が復元される

ア：仮想記憶方式では、ページと呼ばれる固定長のブロックと呼ぶ単位でアドレス変換を行うため、連続した領域に格納する必要はない

イ：キャッシュメモリにおけるプログラムやデータの格納方法

エ：仮想記憶の内容は、システムの停止時に初期化される

■過去問題4

二つのタスクが共有する二つの資源を排他的に使用するとき、デッドロックが発生する可能性がある。このデッドロックの発生を防ぐ方法はどれか？

ア：一方のタスクの優先度を高くする

イ：資源獲得の順序を両方のタスクで同じにする

ウ：資源獲得の順序を両方のタスクで逆にする

エ：両方のタスクの優先度を同じにする

デッドロックは、共有資源を使用する2つ以上のプロセスが、互いに相手プロセスが必要とする資源を排他的に使用していて、互いのプロセスが相手が使用している資源の解放を待っている状態。

デッドロックが発生すると双方のプロセスが永遠に待ち状態になるため、プロセスの続行ができなる。

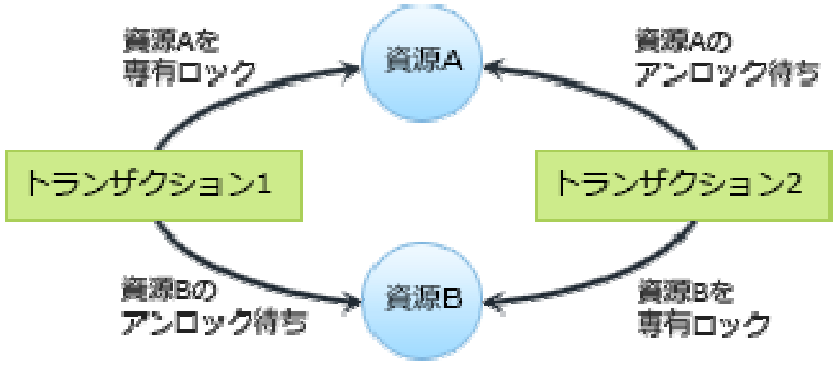


図　デッドロック

ア：優先度を高くしても、資源の占有は解放されない

ウ：デッドロックは、資源獲得順序が異なるときに発生しない

エ：資源の占有は、優先度の高低により影響しない