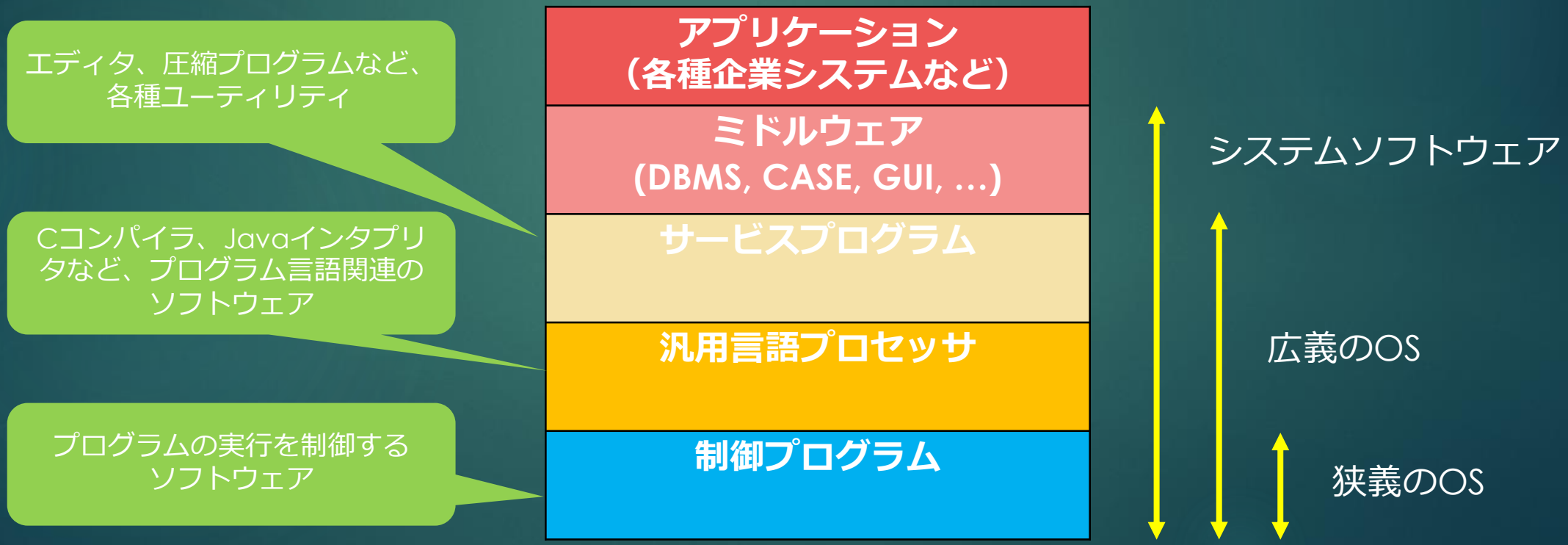
応用情報技術者試験

第4章　ソフトウェア

1. OSの全体像
OSを構成するソフトウェア

- OSはコンピュータの基本機能を提供するためのもので、次のソフトウェアから構成される。



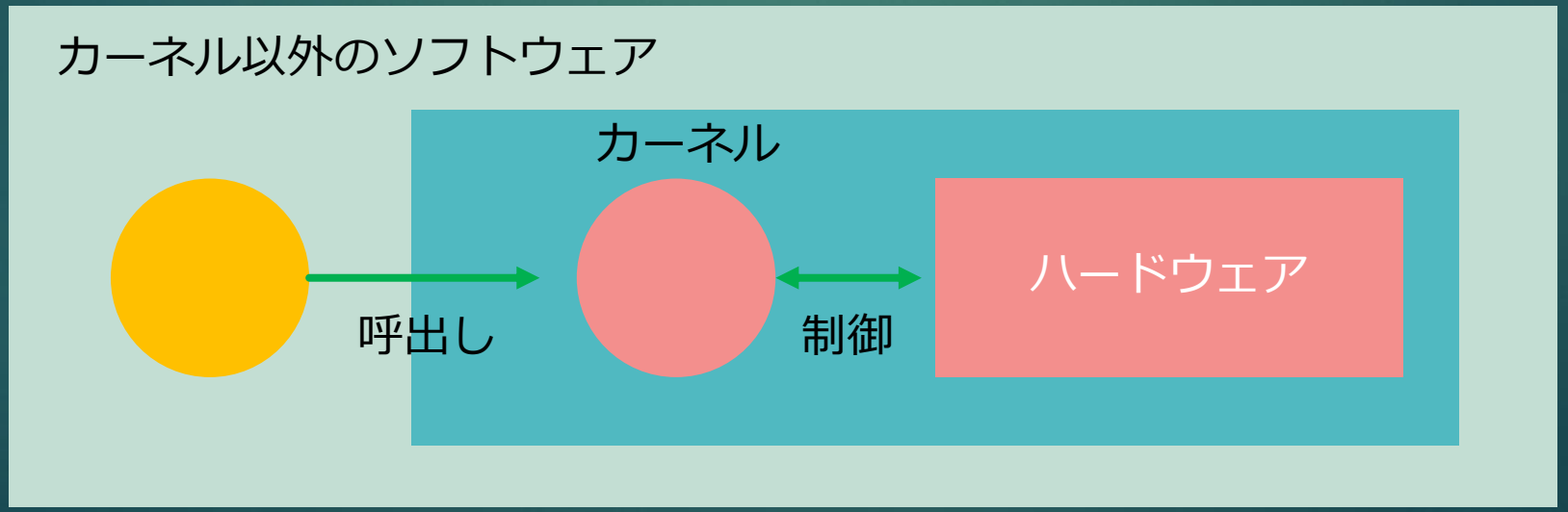
OSを構成するソフトウェア

- ソフトウェアの階層において**制御プログラム**を狭義のOSと呼ぶ。これは**プロセス管理**、**記憶管理**、**ファイル管理**など、プログラムの実行に欠かせない管理・制御プログラムが含まれている。

- **ミドルウェア**とはOSとアプリケーションの中間的なソフトウェアで、多くの利用分野に共通する基本的機能を実現する。

カーネルの分類

- OSは**カーネル(Kernel)**とそれ以外の部分に分けることができる。カーネルはOSの中核部分であり、プロセッサなど**コンピュータのリソースを管理**する。またユーザやアプリケーションからの要求に応じて、ハードウェアレベルでの処理を実行する。カーネルの役割はアプリケーションが動作するための基本機能の提供である。



カーネルの分類

- カーネルにどの程度の管理機能をもたせるかにより、カーネルは次の二つに分類することができる。

モノリシックカーネル	プロセス管理、記憶管理、ファイルシステム、デバイス管理など様々な機能をカーネルに持たせる方式。**効率面で優れるが、複雑で保守が困難**
マイクロカーネル	プロセス管理など**最小限の機能**のみカーネルに持たせる方式

- モノリシックカーネルはOSの全サービスをカーネルに取り込んだ構造である

OSの起動

- コンピュータの電源を投入するとOSが起動する。
- まずOSをロードするためのプログラム（**ブートローダ**）が実行され、OSがコンピュータにロードされる。OSを構成するプログラムは、ロードされたものから順次実行され、コンピュータを利用する準備が整えられる。
- 電源が投入されてからシステムが起動するまでに必要な一連の手順を**ブート(boot)**または**ブートストラップ**と呼ぶ。

○OSの起動

- ブートに関わる用語

ブートローダ	システムをロードし起動するためのプログラム。OSの起動に当たっては、OS用のブートローダを実行する
ネットワークブート	サーバに格納された起動システムやOSをネットワーク経由で読み込んで起動すること
マルチブート	1台のコンピュータを複数のOSをインストールしておき、起動時に選択できること
フラッシュブートローダ	通常はROMに格納するブートローダをフラッシュメモリに格納し、更新可能としたもの

OSの例

- MVSはIBM社のメインフレーム用に開発されたOSで、主な特徴は次の通りである。

・**マルチユーザ**、**マルチプロセス**

・マルチプロセッサをサポート

・**VSAM**を用いたデータ管理に対応

マルチユーザとは同時に複数のユーザが利用できること

マルチプロセスとは同時に複数のプロセスを起動・実行できること

VSAMとは媒体や編成を抽象化したデータ管理のこと

OSの例

- **UNIX**は、**マルチユーザ、マルチプロセスのOS**で主にワークステーションなどで用いられる

・**ディレクトリ**やプロセスをファイルと同様に扱える。

・OS自体がC言語で記述されている。

ディレクトリとはファイルを管理するための登録簿。ファイル名とファイルの実体を対応付けている。

UNIXに限らず、OSでは様々なサービスを提供するプロセスがバックグラウンドで動作しているが、UNIXではそのようなバックグラウンドプロセス群を**“デーモン”**と呼ぶ

OSの例

- 組込システムでは、制限時間内に応答を返さなければならないことも少なくない。そのような場合に**リアルタイムOS**が用いられる。

・仮想記憶やファイル管理などの一般機能は必須ではない

・**イベントドリブン**な制御を行う

・処理に必要なプロセス（タスク）は、起動時にあらかじめ生成しておくことが多い。

イベントドリブンとは何らかの契機（トリガ）に反応して処理を実行することである。

必要になった時点でタスク生成を行う動的なタスク生成は、タスク生成の時間がリアルタイム性を損ねてしまうことがあるのでリアルタイムOSは「起動時」に「静的」にタスクを生成する。

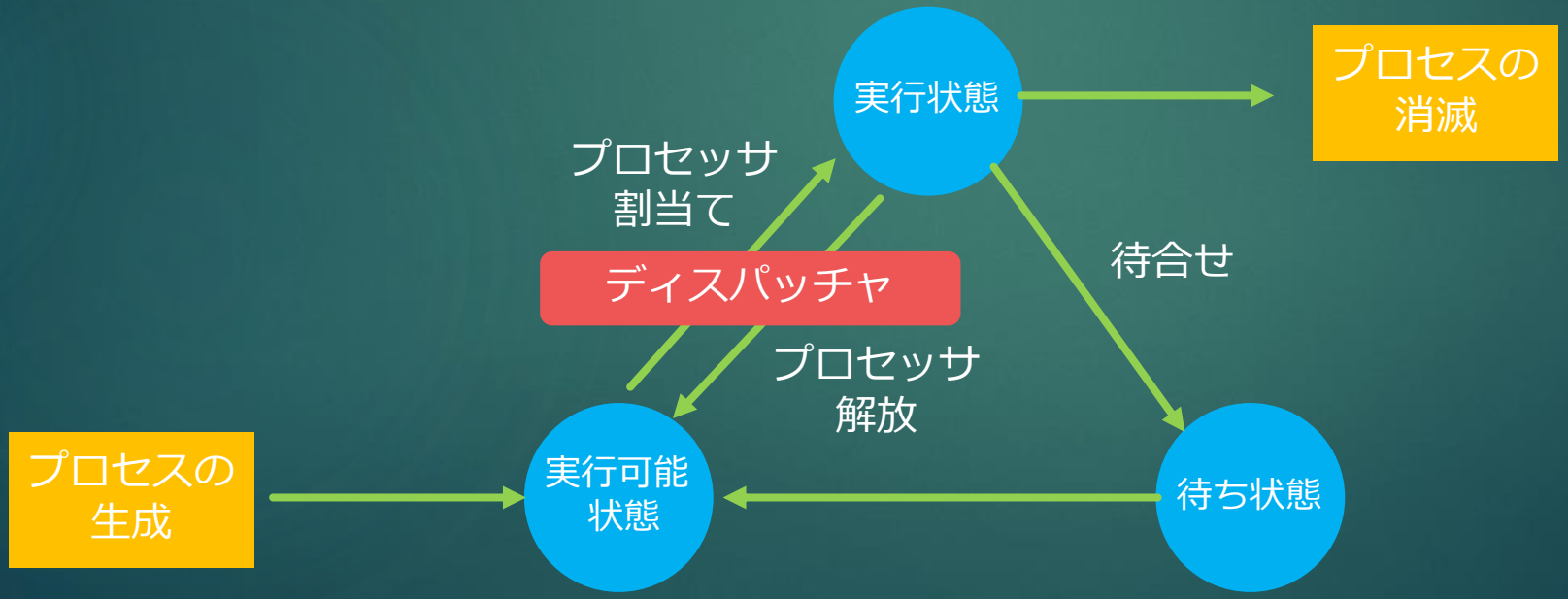
2. プロセスの状態遷移
プロセスの状態遷移

- **プロセス**はプログラムの実行実体で、プログラムの実行時に作成される。
- 例えば、同じプログラムを5人の利用者が同時に実行したとき、5個のプロセスが作成され、それぞれが実行されることになる。OSは、プロセッサやメモリ、入出力装置といった**各種リソース(資源)を個々のプロセスに割り当て、その実行を制御する**。

プロセスの状態遷移

- プロセスの3状態

実行状態	実行に必要な資源を割り当てられ、実行している状態
実行可能状態	資源の割当てがあれば、すぐに実行を開始できる状態
待ち状態	入出力完了など、何らかの事象発生を待っている状態



ディスパッチャ

- **ディスパッチング（ディスパッチャ）**

実行可能状態のプロセスに資源を割当て、**実行権を渡して実行状態に移すこと**

- **ディスパッチャ**

ディスパッチングを行う機構

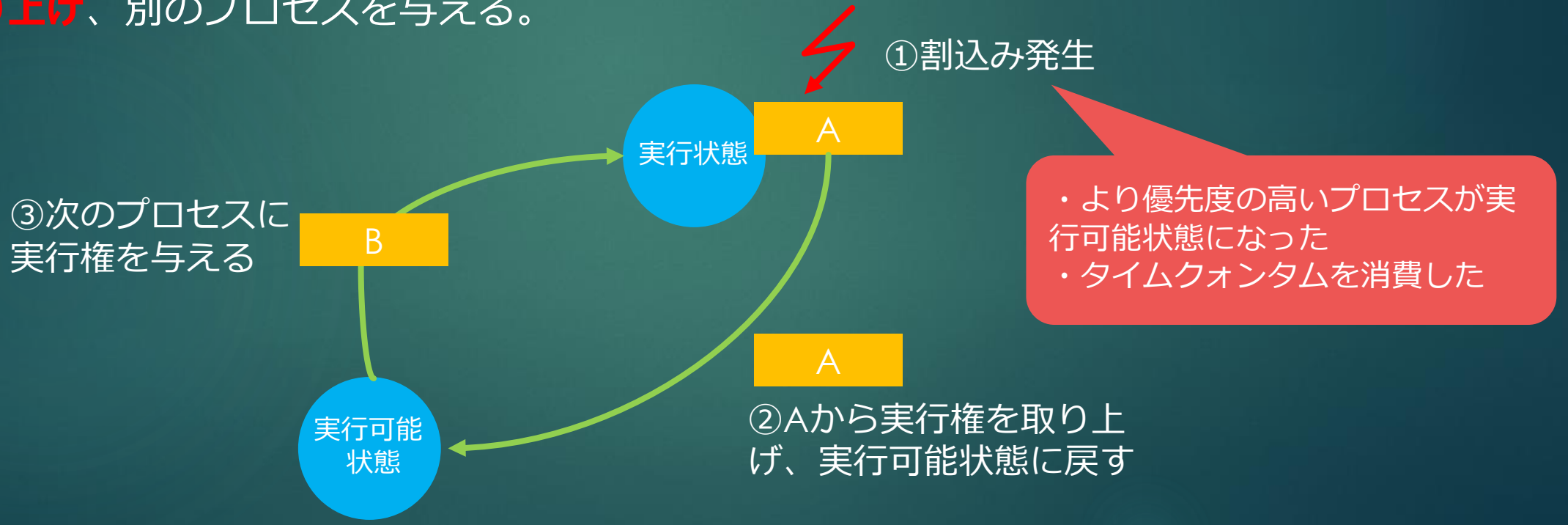
コンテキスト切替え

- **コンテキスト切替え**

あるプロセスから別のプロセスに、プロセッサの割当てを変更すること

- **プリエンプティブ**

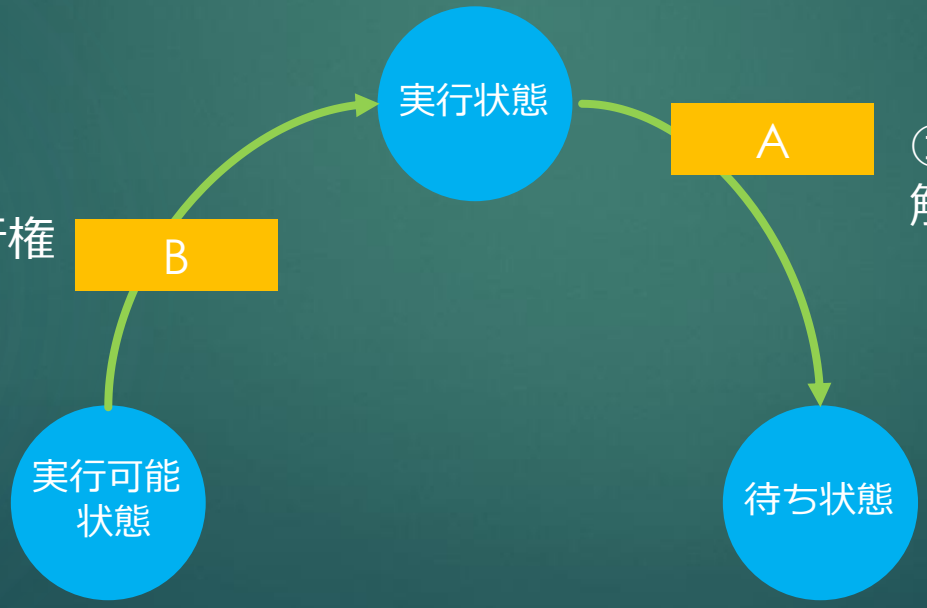
割り込みを発生させ、実行中のプロセスから**強制的にプロセッサの使用権を取り上げ**、別のプロセスを与える。



コンテキスト切替え

- **ノンプリエンプティブ**

強制的な切替えを行わない。プロセスが次自主的に解放したプロセッサの使用権を、別のプロセスに与える。



コンテキスト切替え

- 複数のプロセスを同時並行的に実行する**マルチプログラミング（マルチタスク）では、プリエンプティブ方式が望ましいといえる**。強制的な切替えのないノンプリエンプティブ方式では、プロセッサ使用時間の長いプロセスがCPUを保持し続け、ほかのプロセスに順番が回らないといったことがあり得るからである。

スケジュールリング

- 実行可能状態プロセスの待ち行列をどのように形成し、どのようにディスパッチするかを**スケジューリング**と呼ぶ
- スケジューリング方式

到着順方式	各プロセスを到着順で待ち行列に並べ、先頭から順にプロセッサを割り当てる。
SJF(Shortest Job First)方式	より短い時間で終了するプロセスを優先的に処理する。応答時間の短縮には理想的であるが、完全な実装は困難
ラウンドロビン方式	各プロセスに一定のCPU使用時間(タイムクォンタム）与え、これを使い切った場合は待ち行列の最後尾に回す
優先度順方式	各プロセスにあらかじめ「優先度」を設定して起き、優先度ごとに待ち行列を設定する。
多重待ち行列方式	優先度順方式と、ラウンドロビン方式を組み合わせた方式。各優先度ごとに待ち行列を用意し、それぞれはラウンドロビン方式で制御する。
フィードバック待ち行列方式	優先度順方式と、ラウンドロビン方式を組み合わせた方式。タイムクォンタムを使い切ったプロセスの優先度を動的に下げてゆくことで、疑似的なSJFを実現する

スケジューリング

- **ラウンドロビン方式**は、タイムクォンタムを長くすれば**到着順方式**に近づき、タイムクォンタムを短くすれば**SJF**に近づくことになる。

- **優先度順方式**では、**優先度の低いプロセスが長時間処理されない**という状況が起こりえることになる。これを防ぐため、一定時間のあいだ待ち状態であったプロセスの優先度を引き上げる方式（**動的優先順方式**）もある。

3. プロセスの排他 / 同期制御
プロセスの排他制御

- マルチプログラミング環境では複数のプロセスが同時並行的に実行されるため、資源の競合を解決する排他制御が必要となる。

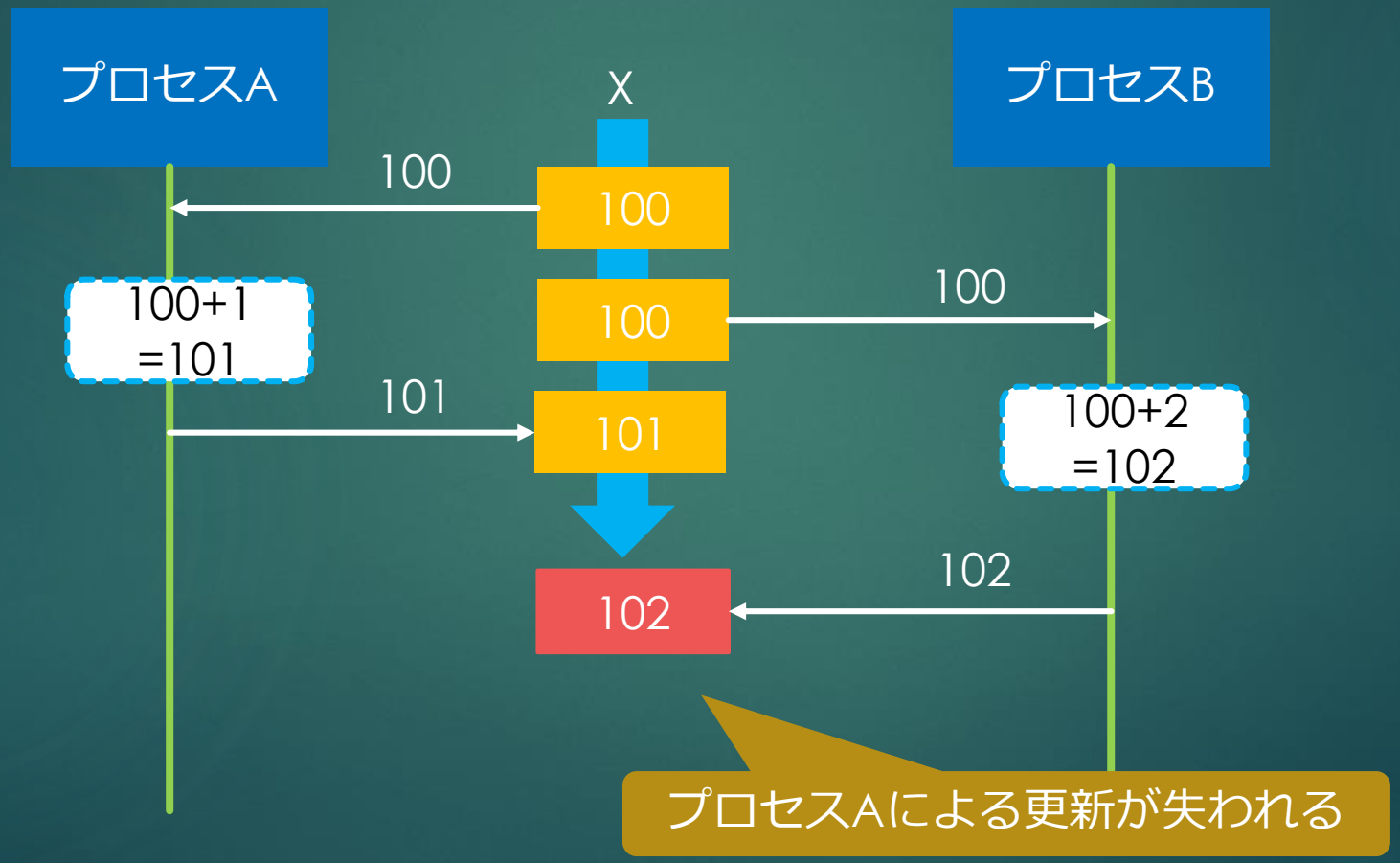
- **排他制御**

あるプロセスが資源を利用している間、ほかのプロセスによるアクセスを許さないような仕組み

プロセスの排他制御

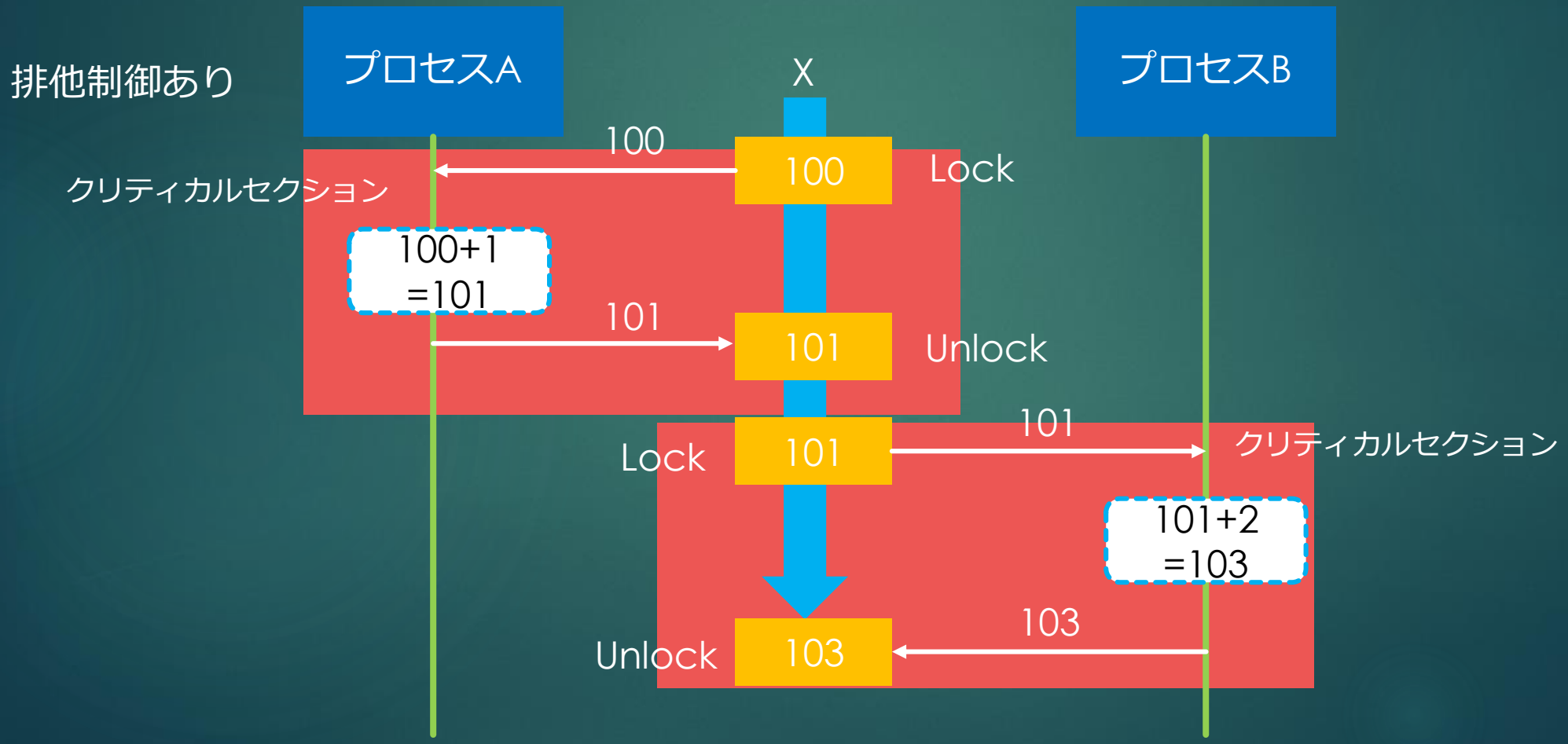
＜例＞資源Xの値を参照し、これに1を加えるプロセスAと2を加えるプロセスB

排他制御なし



プロセスの排他制御

＜例＞資源Xの値を参照し、これに1を加えるプロセスAと2を加えるプロセスB



クリティカルセクション
他のプロセスのアクセスを許すと、正しい処理が行えなくなる部分

プロセスの排他制御

- 排他制御をおこなわない場合実行の順序によって更新の結果が失われる恐れがある。これを防ぐために参照から更新までを排他制御をする。具体的にXの参照前にXに**ロック**をかけて占有し、更新後に占有状態を解除（**アンロック**）する。これにより資源Xは「一方の参照・更新→他方の参照・更新」の順でアクセスされ、矛盾なく実行される。

セマフォシステム

- **セマフォシステム**

Dijkstra(ダイクストラ）によって考案された同期・排他制御のメカニズム

セマフォは、資源の残量を示すための変数と、資源解放を待っているプロセスの待ち行列とで構成される。プロセスはクリティカルセクションに入る前に**P操作**を行い、クリティカルセクションから出る際に**V操作**を行う。

P操作	**セマフォ変数から1減じる** 結果が負になれば、実行を中断して待ち行列に入る
V操作	**セマフォ変数を1増加する** 増加前の値が負であれば、待ち行列の先頭プロセスを待ち行列から外し、実行可能状態にする

セマフォシステム（参考）

- **バイナリ(2進)セマフォ**

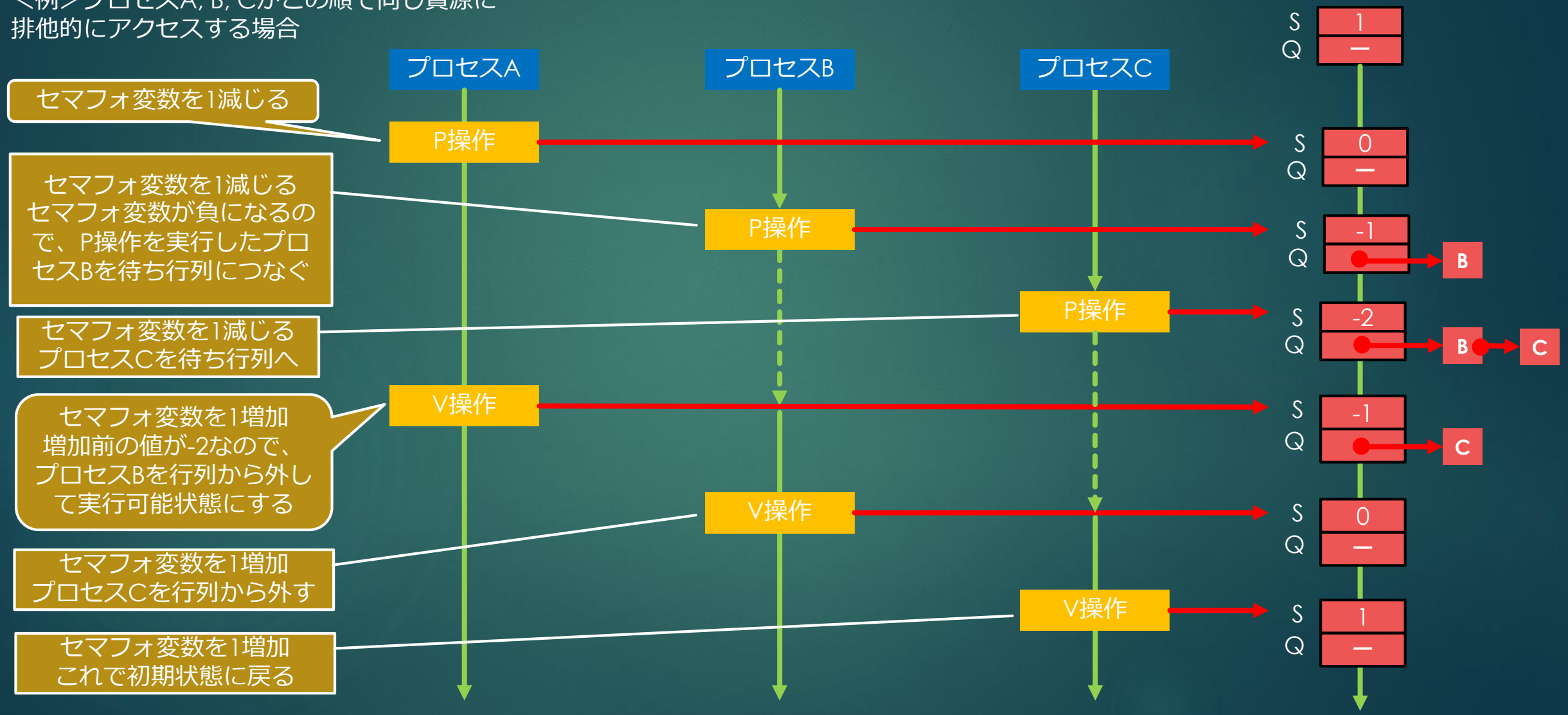
セマフォ変数として0/1の値しかとらないもの

- **ジェネラル(汎用、計数）セマフォ**

セマフォ変数で0/1以外の値をとるもの

セマフォシステム

<例>プロセスA, B, Cがこの順で同じ資源に
排他的にアクセスする場合



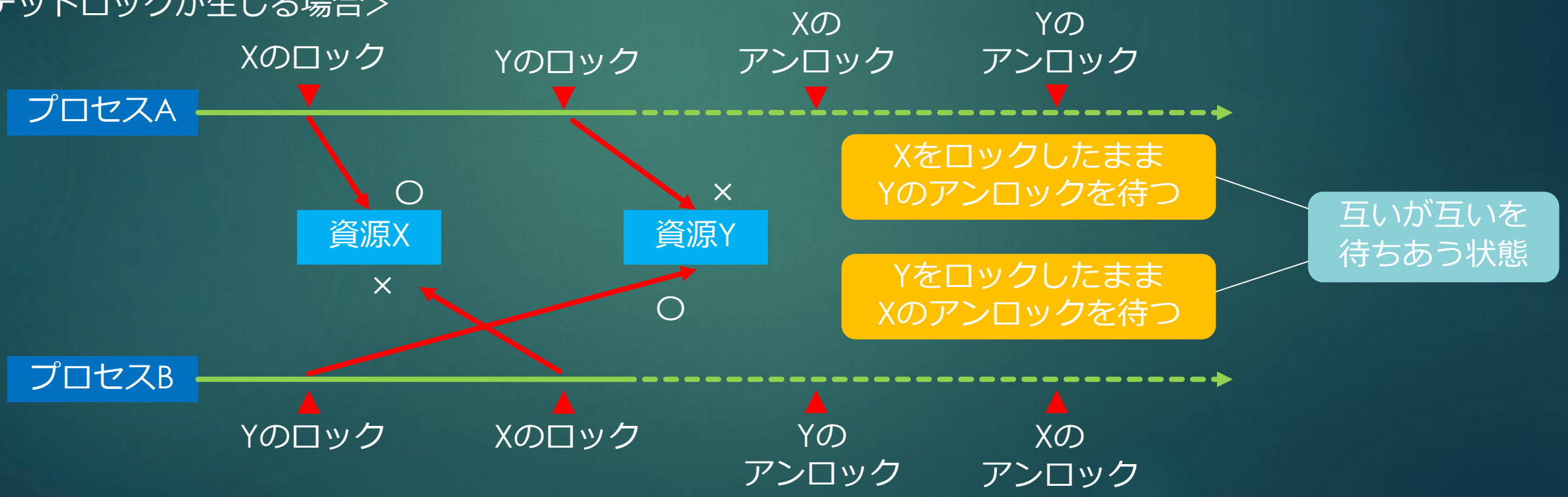
デッドロック

- **デッドロック**

複数のプロセスが互いに資源解放を待ちあってしまい、先に進めなくなる膠着状態

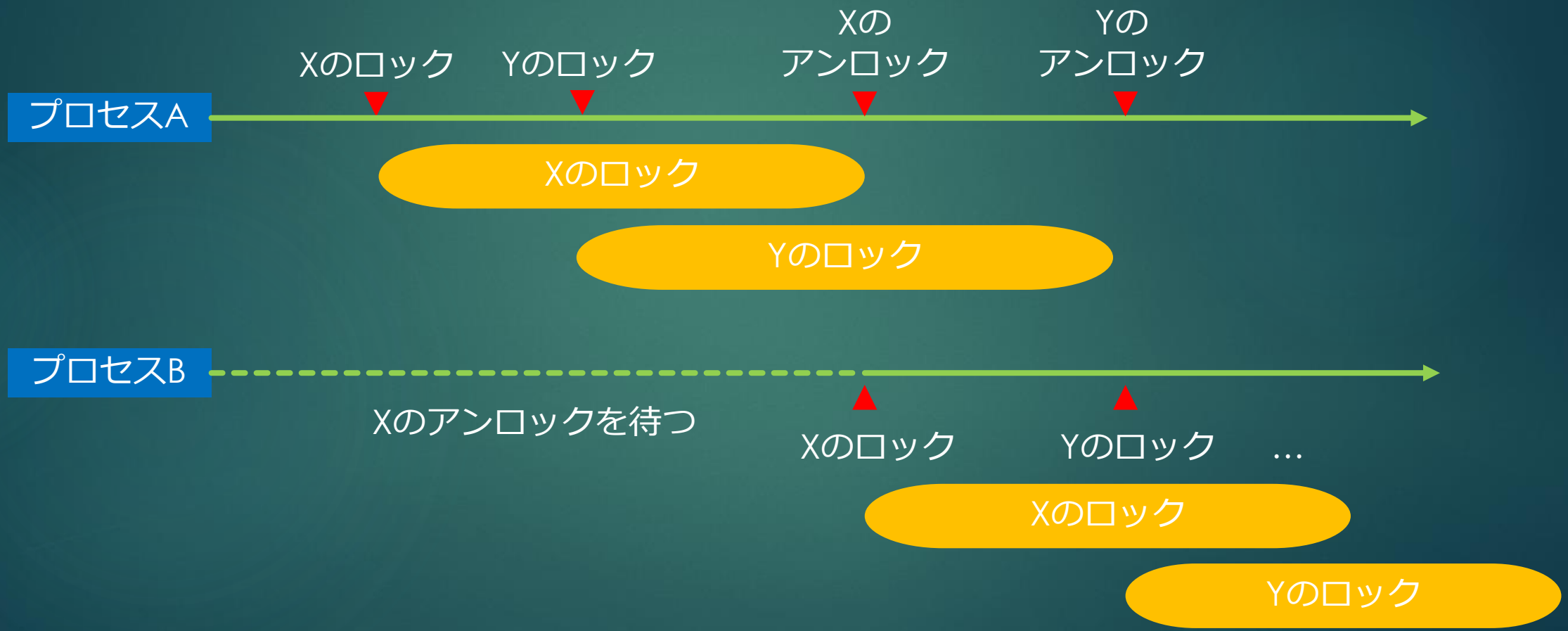
プロセスが無秩序に資源のロック/アンロックを行うとデッドロックが生じる恐れがある

<デッドロックが生じる場合>



デッドロック

<デッドロックが生じない場合>



デッドロック

- 先の例ではプロセスAが「X→Y」、プロセスBが「Y→X」という順序で資源のロックを試みたことで、デッドロックが発生した。これを防ぐためには、**資源のロック順序を一意にそろえる**ようにする必要がある。これによってデッドロックが発生しなくなる。

- デッドロックを検知した場合には、原因となっているプロセスを強制終了させて回復する（**検知と回復**）。

プロセスの同期制御

- マルチプログラミング環境では、あるプロセスの結果を別のプロセスが利用する、といったように、複数のプロセスが協調しながら処理を行うことがある。そのために、プロセス間の同期制御が必要となる。

- **同期制御**

各プロセス間で相互に連絡を取り、実行タイミングを合わせるための制御

イベントフラグ方式

- **イベントフラグ方式**は、各種の状態情報を表すビット列（**イベントフラグ**）を設ける方式である。たとえば「プロセス1の処理Aの後でプロセス2の処理Bを実行する」という場合、プロセスA、Bの間で同期が必要である。このため、プロセスA、B間で同期用フラグをイベントフラグから選び、これをOFFにしたうえで、

・プロセス1は、処理Aが終わった時点でフラグをONにする

・プロセス2は処理Bの直前でフラグを調べ、OFFであればONになるのを待ってから実行する

とプログラミングする。

4. 割込み制御
割込み制御

- **割込み制御**の基本は、割込みが発生した場合に、

・割込みの種類に応じた正しい**割込みルーチン**を実行する

・割込みルーチン終了後、元のプログラムを再開することである

割込み制御

- 割込みが発生した場合の手順

[1] 割込みを受けたプロセスのPSWの内容を退避する

[2] 割込みルーチンのPSWを取り出し、PSWレジスタに設定する

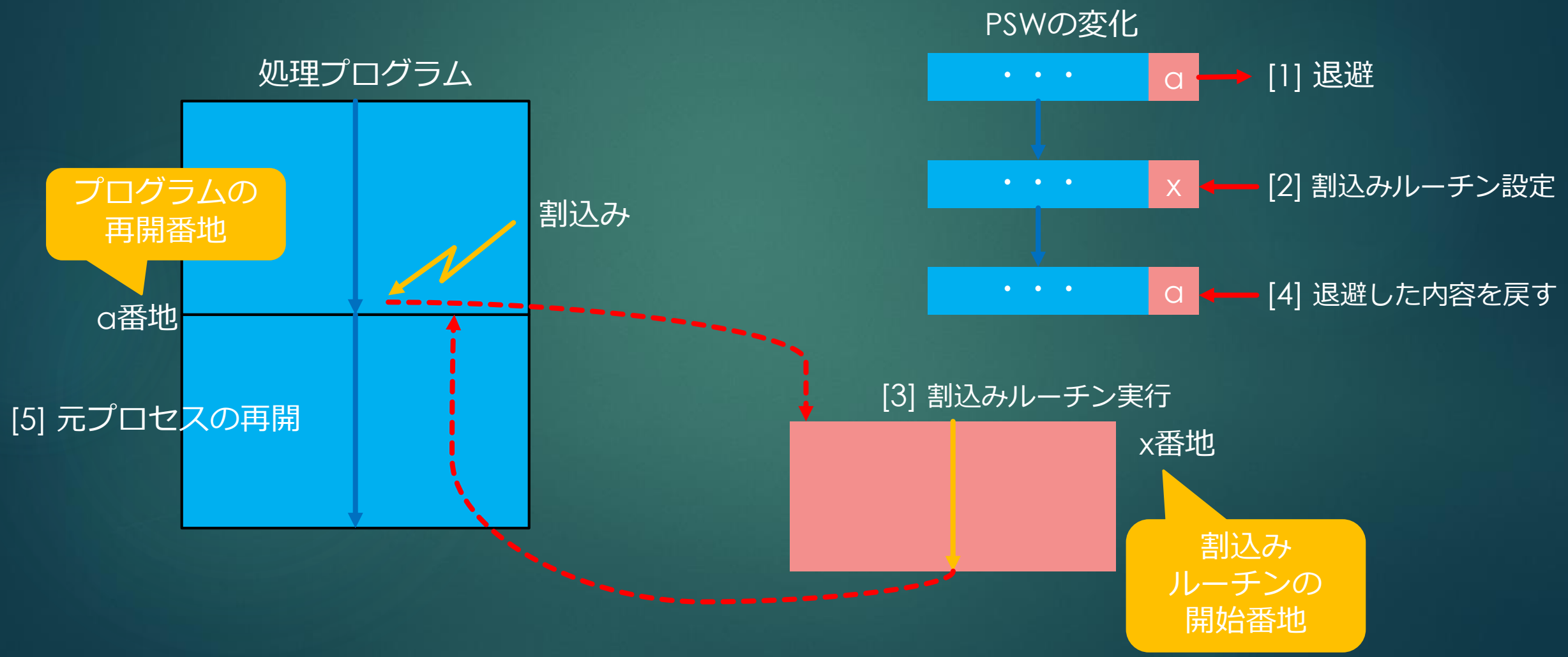
[3] 新しいPSWレジスタの内容に従って、割込みルーチンを実行する

[4] 割込みルーチンの終了後、退避していたPSWの内容をPSWレジスタに戻す

[5] 復旧したPSWレジスタの内容に従って、元プロセスの実行を再開する

PSW (Program Status Word)
プログラムの状態を記憶している領域。この内容を入れ替えることで、実行するプログラムを切り替えることができる。

割込み制御


処理プログラム
プログラムの
再開番地
a番地
割込み
[5] 元プロセスの再開
PSWの変化
・・・
a
[1] 退避
・・・
x
[2] 割込みルーチン設定
・・・
a
[4] 退避した内容を戻す
[3] 割込みルーチン実行
x番地
割込み
ルーチンの
開始番地

割込みの分類

- 割込みはその発生要因によって、外部割込みと内部割込みの二つに大別することができる。
- **外部割込み**

プロセッサ以外のハードウェア（入出力装置やメモリのECC、ハードウェアタイマなど）によって生じる割込み

- 外部割込みの分類

名称	発生要因
入出力割込み	入出力動作の完了、入出力装置の状態変化
タイマ割込み	タイマ（計時機構）に設定された所定の時間が経過
外部信号割込み	このソールからの入力、ほかの処理装置からの連絡など
異常割込み	電源異常、処理装置 / 主記憶装置の障害など。機械チェック割込みともいう

割込みの分類

- **内部割込み**

プロセッサ内部の要因によって生じる割込み

- 内部割込みの分類

名称	発生要因
演算例外	オーバーフロー / アンダーフローの発生、0による除算
不正な命令コードの実行	存在しない命令や形式が一致しない命令を実行
モード違反	特権モードの命令をユーザモードで実行
ページフォールト	仮想記憶において存在しないページを指定
割出し	SVC命令の実行、トラップ処理

割込み優先度と割込みマスク

- 割込みには優先度がある。ある割込み処理中に、より高い優先度の割込みが生じた場合に「割込み処理がさらに割り込まれる」といった多重割込みが発生することになる。

割込み優先度と割込みマスク

- **割込みは必要に応じて禁止することもできる**。割込みは禁止するには、**割込みマスク**を用いて対応する割込みフラグを0にする。なお、禁止できない割込み（**ノンマスカブル割込み**）もある。

5. 記憶管理
実記憶管理

- 限られた主記憶領域を多くのプログラムで利用するために、「無駄なく」かつ「コンパクト」使用することが求められる。これを目的に、様々な管理・制御がおこなわれることになる。

- 新たなプログラムを実行するとき、これをロードする空き領域が主記憶に存在しないことがある。このとき、**主記憶上のいずれかのプログラムを外すことで空き領域を確保**し、そこにプログラムをロードして実行する。主記憶から外されたプログラムは、適切なタイミングで主記憶に戻され、処理が再開する。

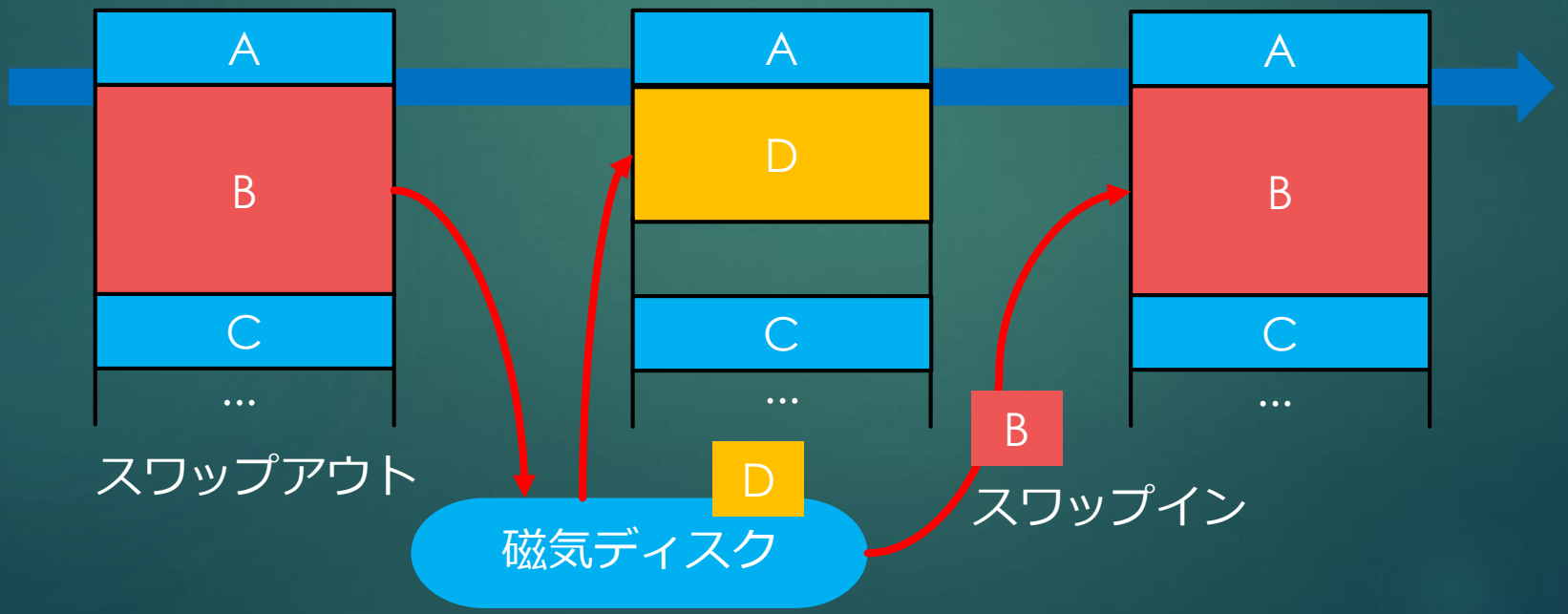
スワッピング

- **スワップアウト**

空き領域確保のため、プログラムを主記憶から外すこと

- **スワップイン**

スワップアウトしたプログラムを主記憶に戻すこと



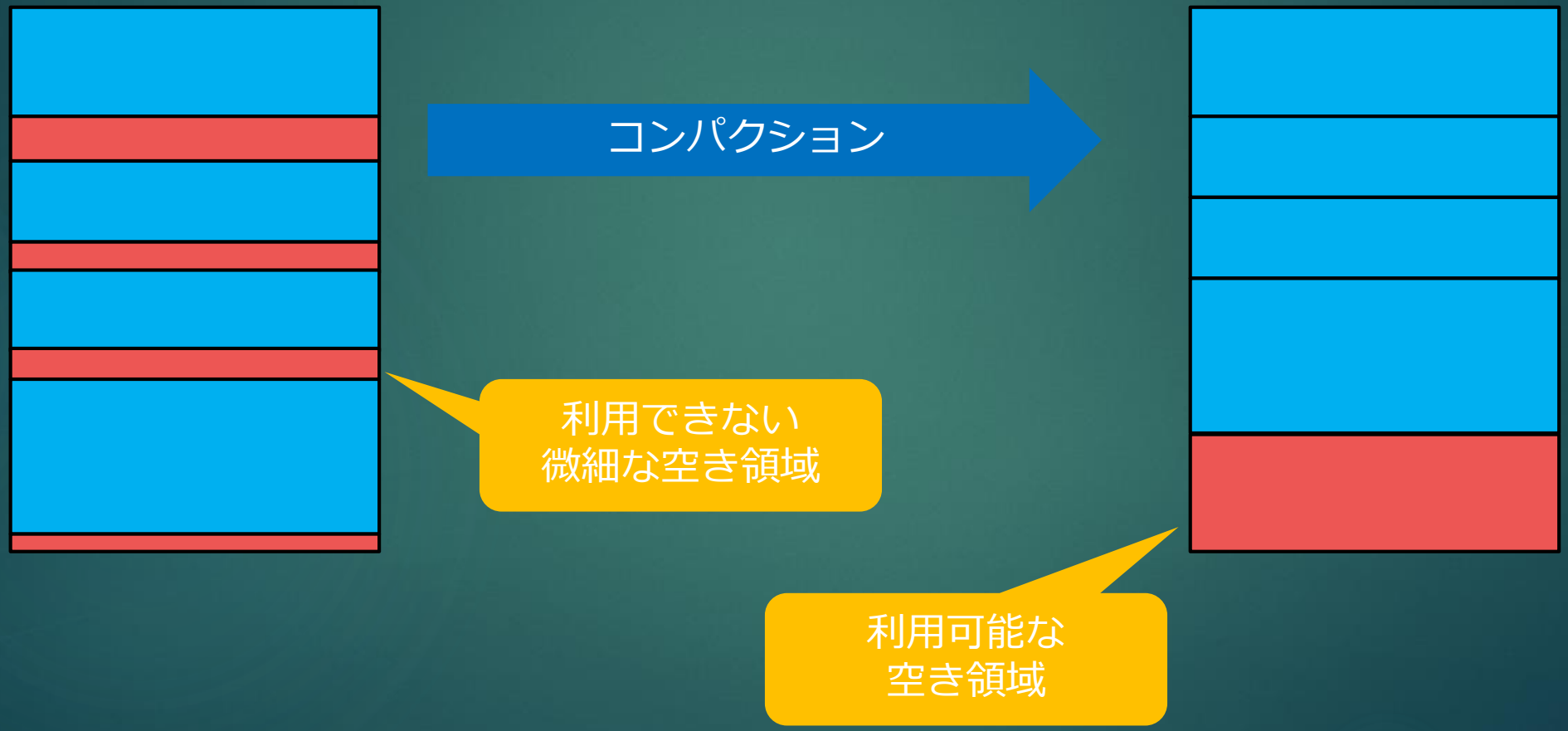
ガーベジコレクション / コンパクション

- OSの記憶管理は、主記憶上の空き領域を「空き領域リスト」で管理している。プログラムが解放した領域は、記憶管理がこのリストに戻す。これを**ガーベジコレクション**とよぶ。
- ガーベジコレクションだけであると、空き領域の管理は十分とはいえない。それはプログラムの実行や収量が繰り返されると「空きリスト上には十分な空き領域が管理されているにも関わらず、一つひとつは微細なため利用できない」という現象が生じる恐れがあるため。これを**主記憶の断片化（フラグメンテーション）**とよぶ。
- **フラグメンテーションが生じると主記憶の利用効率は低下する。**そこで記憶管理は、適切なタイミングでプログラムを再配置し、微細な空き領域を一つの利用可能な領域にまとめる。この処理を**コンパクション**と呼ぶ。

ガーベジ
コレクション

使わなくなったメモ (ガーベジ=ゴミ) を収集すること

フラグメンテーションとコンパクション



オーバーレイ

- **オーバーレイ**

プログラムをいくつかの**セグメント**に分け**実行に必要なセグメントのみ組み合わせてロード**することで、プログラムが占有する主記憶領域を節約する方法

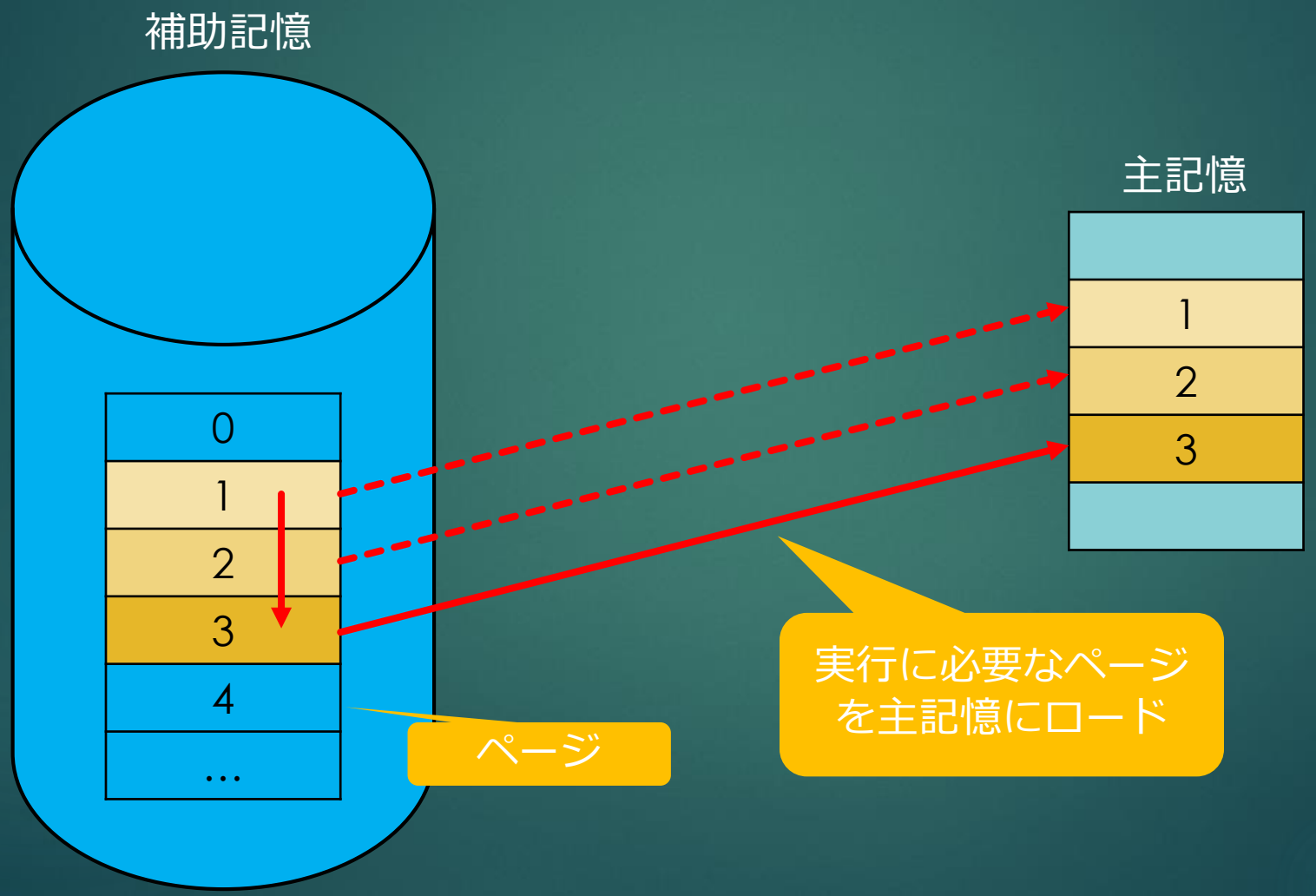
- ただし、セグメントとの分割や組み合わせはアプリケーションがOSに指示するため、アプリケーションの作成時にプログラマに大きな負担をかけることになる。

仮想記憶管理の概要

- 長大なプログラムであっても、実際に実行している部分はほんの一部にすぎない。よって、プログラムのうち「実行に必要な部分」のみを抜き出してロードすれば、主記憶領域を大幅に節約できる。このための仕組みが**仮想記憶**である。その管理機能を**仮想記憶管理**と呼ぶ。

- 仮想記憶は、補助記憶上のプログラムを固定長のページに分割し、実行に必要なページのみ主記憶にロード（ページイン）することで実現する。

仮想記憶管理の概要



仮想アドレス

- 補助記憶上のプログラムは**仮想アドレス**で構築されている。これは（ページ番号、ページ内オフセット）で表されるアドレスで、命令の実行時に実アドレスに変換される。この変換を行うユニットが**MMU**である。

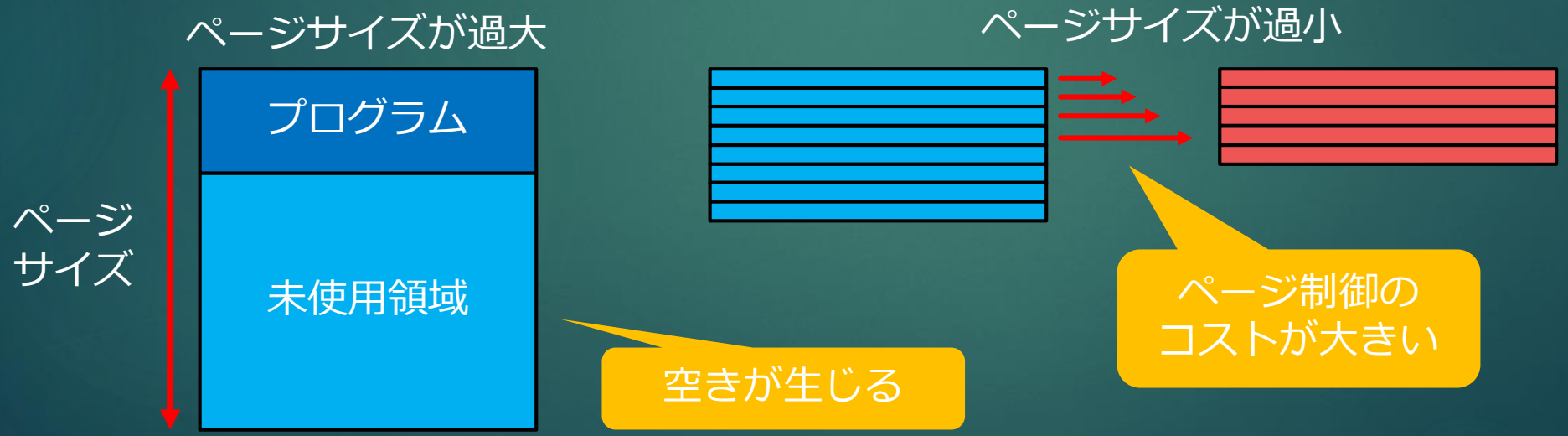
- MMUはCPUが指定した仮想アドレスを物理アドレスに対応させる働きをもつ。

ページサイズと効率

- ページサイズは仮想記憶の効率に大きく影響する。ページサイズがプログラムのサイズを超えれば、剰余部分が未使用領域（内部フラグメンテーション）となり、主記憶の利用効率が低下することになる。反対にページサイズを小さくしすぎると、ページの入れ替え頻度が高くなり、実行効率を低下させることになる。
- ページサイズはあらかじめハードウェアレベルで設定されているため、これをプログラム実行中に変えることはできない。

内部フラグメンテーション

区画内に生じる未使用領域



ページ置換え（ページリプレースメント）

- 仮想記憶を用いてプログラムを実行するとき、実行に必要なページが主記憶に存在しないときは**ページフォルト**が生じる。これを受けて仮想記憶管理は必要なページを**ページイン**する。その際に主記憶に空きページ領域がなければ、主記憶上のいずれかのページを**ページアウト**して空き領域を確保してからページインを行う。このようなページの置き換えを**ページリプレースメント**と呼ぶ。

- **ページフォールト**

実行に必要なページが主記憶に存在しないときに生じる例外

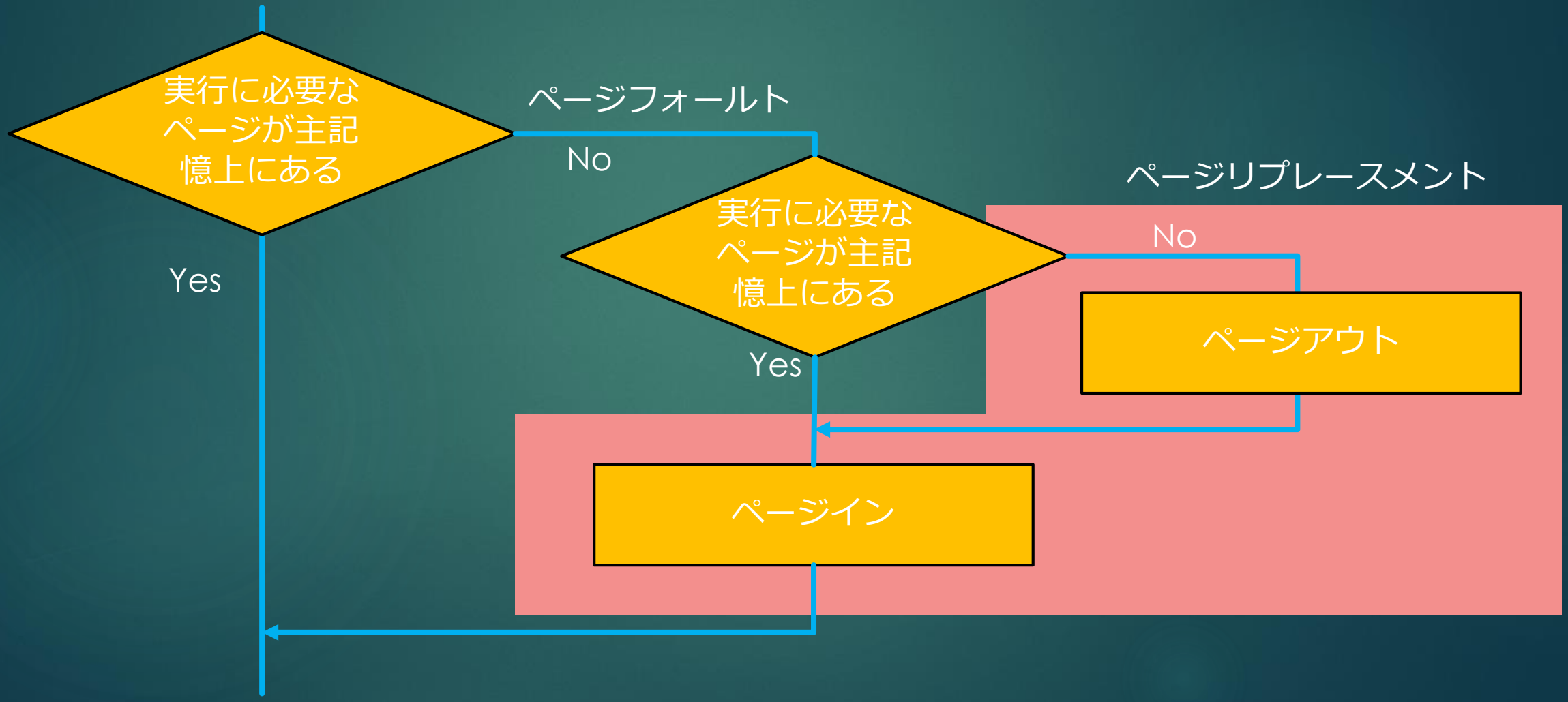
- **ページイン**

実行に必要なページを主記憶にロードすること

- **ページアウト**

主記憶上のページを補助記憶に戻すこと

ページ置換え（ページリプレースメント）



ページインの方式

- ページインの方式には、デマンドページングとプリページングがある。**デマンドページング**は、**ページフォールトを契機として該当ページのページインを行う方式**で、これまで説明に用いた方式はデマンドページングに該当する。

- デマンドページングに対して、あらかじめ必要とされそうなページを予測し、そのページを**ページフォールト発生よりも先にページインしておく方式**が**プリページング**である。予測が正確ならば、ページフォールトの発生を抑えることができる。

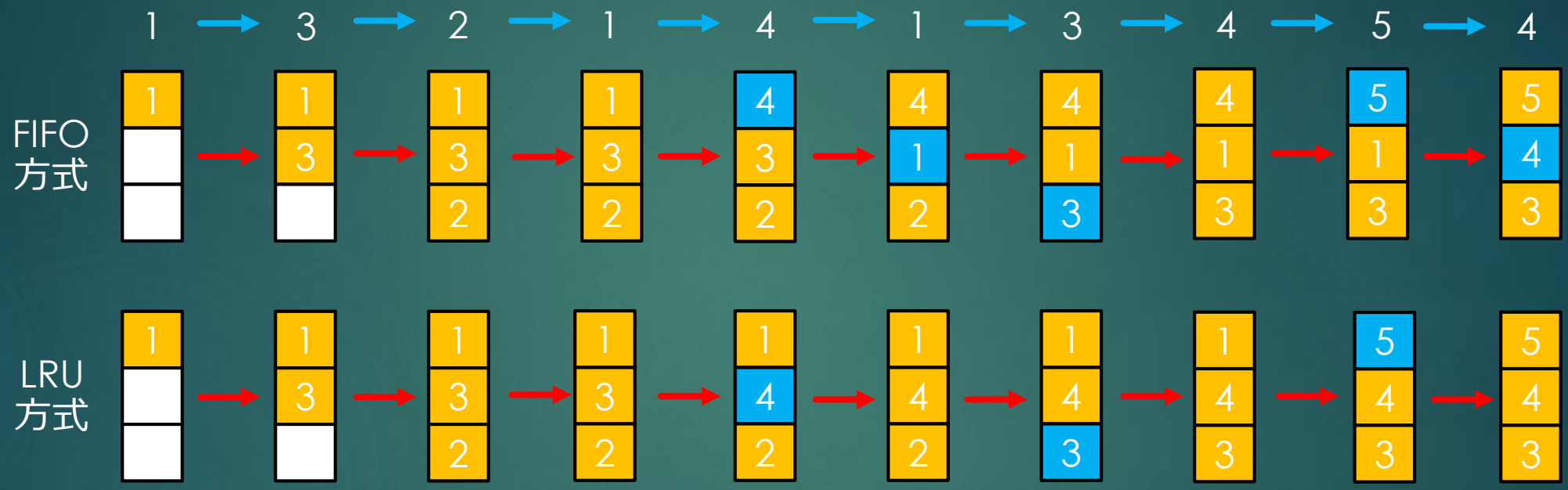
ページ置換えアルゴリズム

- ページリプレースメントが生じるとき、仮想記憶管理は主記憶上のページを選んでページアウトさせる。この時選ぶページは「実行可能に当面不要なページ」であるべきである。実行に必要なページを選んでしまうと、すぐにページフォールトが生じるためである。

- ページリプレースメントにおいて、ページアウトするページを選ぶアルゴリズムを、**ページ置換えアルゴリズム**とよぶ。

名称	ページアウト対象
FIFO(First-In First-Out)	最も長く主記憶に存在する（＝もっとも先に読み込んだ）ページ
LRU(Least Recently Used)	最後に参照されてからの経過時間が最も長いページ
LFU(Least Frequency Used)	最も使用頻度が少ないページ

ページ置換えアルゴリズム



- 一般にLRU方式の方がFIFO方式よりも平均的なページ置換えの回数が少なく、実行効率の面で有利である。

スラッシング

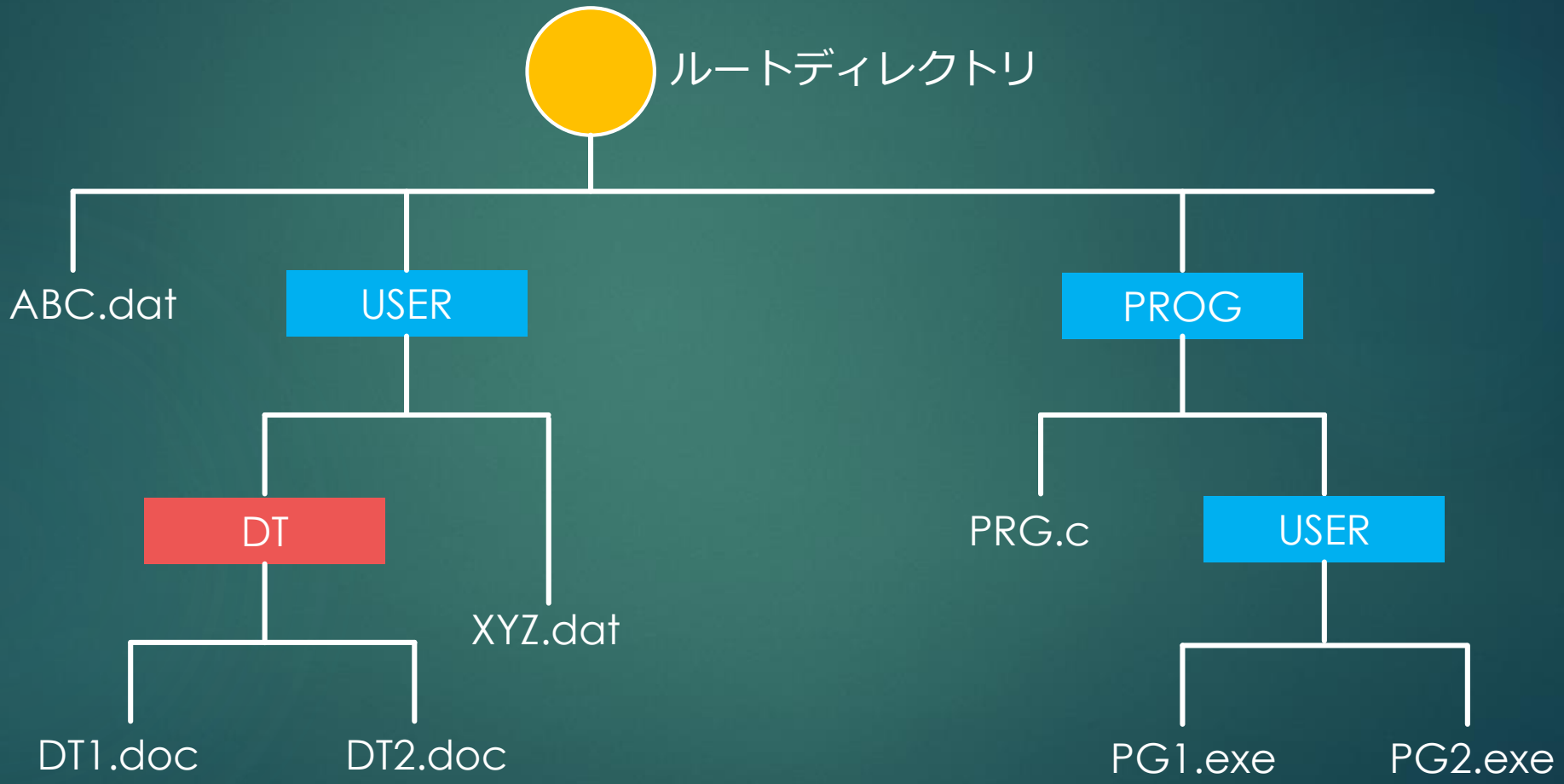
- 同時並行的に実行するプロセスが多くなり、主記憶の利用頻度が高くなってくると、ページ置換えが多発し、そのオーバヘッドによって**システムの処理効率が急激に低下**する。そのような現象を**スラッシング**とよぶ。

6. その他の管理
ファイルシステム

- データやプログラムはファイルという単位で記録・利用される。これらのファイルを管理する仕組みがファイルシステムで、多くのOSではファイルシステムに階層型の**ディレクトリ**で用いている。

- ディレクトリはファイルを管理するためのファイルで、ファイル名とファイルの実体との対応を管理する。ディレクトリはいわば「ファイルの入れ物」であり、その入れ物に、さらにディレクトリを収めることで、階層構造を実現する。

ファイルシステム


ルートディレクトリ
ABC.dat
USER
PROG
DT
XYZ.dat
PRG.c
USER
DT1.doc
DT2.doc
PG1.exe
PG2.exe

ファイルシステム

- 階層構造内のファイル位置を指定することをパス指定と呼ぶ。パス指定には、**ルートディレクトリ**からの位置を指定する**絶対パス**と、**カレントディレクトリ**からの位置を指定する**相対パス**がある。
- パスの指定におけるディレクトリ移動の表記はOSによって異なるが、一般的には、

・ディレクトリの切れ目：＼または／

・ルートディレクトリ：先頭に＼または先頭に／

・カレントディレクトリ：.

・一つ上のディレクトリ：..

カレントディレクトリがDTであるとき、PRG.cを指定するには

絶対パス:/PROG/PRG.c

相対パス:./../../PROG/PRG.c(先頭の./は省略可)

ルートディレクトリ

最上位のディレクトリ

カレントディレクトリ

現在作業中のディレクトリ

入出力管理

- 入出力制御ソフトウェア

名称	特徴
入出力割込みルーチン	入出力割込み時に起動され、割込み内容に対応した処理を行う
デバイスドライバ	各種機器の動作を制御するためのモジュール
入出力API	応用プログラムが、ファイル単位で容易にデータを読み書きできるようインターフェースを提供する

- **デバイスドライバ**の内容は各OSおよび各機器の仕様に依存するため、OSごと、機器ごとに異なった内容のものを用意する必要がある

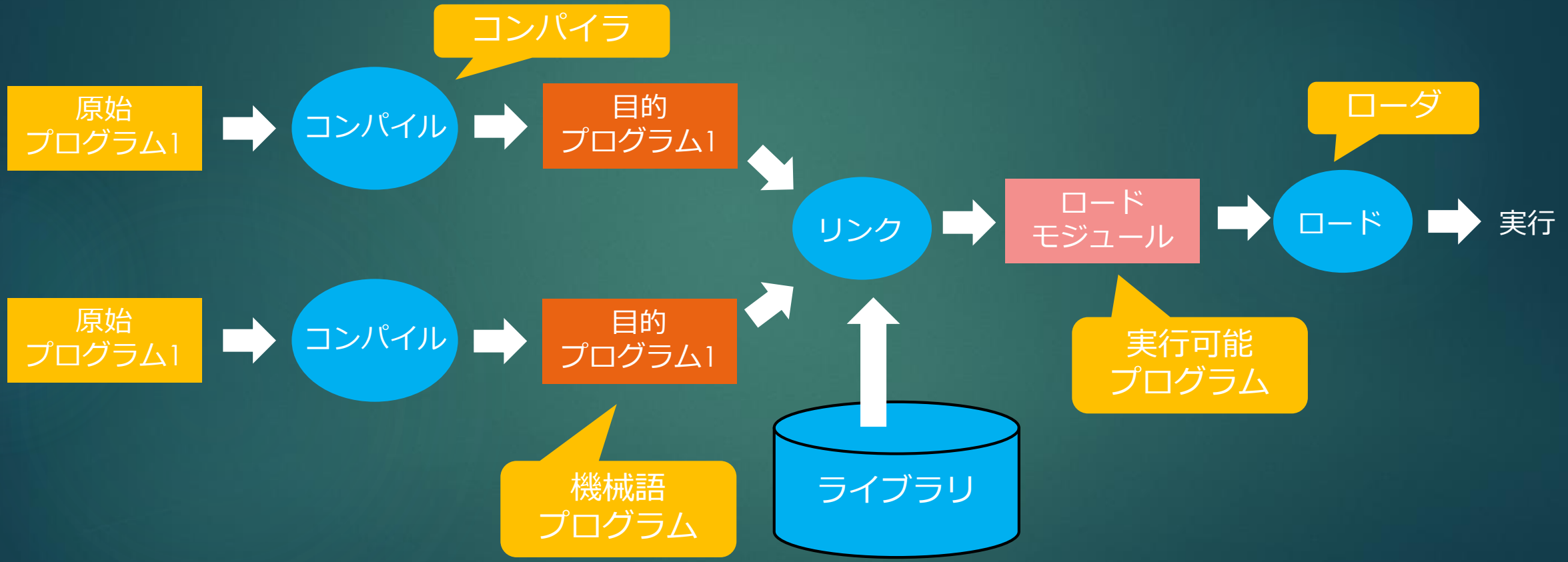
入出力の高速化

- 入出力の動作は一般に低速であり、システム性能のボトルネックとなりがちである。

- 入出力の高速化

スプーリング	磁気ディスクなどの補助記憶装置を「仮想的な入出力装置」とみなして、これに対して入出力を行う
バッファリング	主記憶領域の一部を「一時的なデータの置き場」とし、そこを介して入出力を行う
RAMディスク	主記憶領域の一部を、仮想的な補助記憶装置として扱う。ファイルアクセスが高速化する。

7. プログラムの実行制御
プログラム実行の流れ



プログラム実行の流れ

- モジュールの形態

ソースモジュール（原子プログラム）	アセンブラ言語やC言語、COBOLといった高水準言語で記述されたプログラムコード
オブジェクトモジュール（目的プログラム）	ソースモジュールをコンパイラやアセンブラによって翻訳して得られる、実行プラットフォーム（ハードウェア）が理解できるコード
ロードモジュール	すぐに主記憶にロードし、実行できる形式のモジュール

- プログラム実行制御に用いるソフト

コンパイラ	高水準言語で記述されたソースからオブジェクトモジュールを作成する
アセンブラ	アセンブラ言語で記述されたソースからオブジェクトモジュールを作成する
リンカ	オブジェクトモジュールおよびライブラリを連係（リンク）し、ロードモジュールを作成する
ローダ	ロードモジュールを主記憶にロードし、実行する

静的リンクと動的リンク

- リンクは、これを行うタイミングによって**静的（スタティック）リンク**と**動的（ダイナミック）リンク**に分類することとすることができる

静的リンク	実行に先立ってリンクを済ませておく
動的リンク	実行後に**ライブラリが必要になった時点**でリンクを行う

- 静的リンクは個々のプログラムに**ライブラリ**を結合するため、プログラムのサイズが大きくなってしまう。また、ライブラリがバージョンアップしたとき、プログラムの再リンクが必要である。

- 動的リンクは、**ライブラリを複数のプログラムで共有できる**ため、主記憶の利用効率は向上する。ただし、ライブラリ側のプログラムは複数のプログラムで共有されるので、同時に呼び出されても正しく処理を行うような性質（再入可能性）を持たなければならない

静的リンクと動的リンク

- 動的リンクに用いられるライブラリを**ダイナミックリンクライブラリ(DLL)**または**共有ライブラリ**と呼ぶ

プログラムの再利用性

- ライブラリ側のプログラムは何度も繰り返し使われるため、再利用性を持たなければならない。

- プログラムの性質

逐次再利用可能性	主記憶のロードされたプログラムを再びロードすることなく実行できる性質。ただし、同時に実行できない。
再入可能性（リエントラント）	複数のプロセスから**同時に呼び出されても、正しい結果を返す**ことができる。
リカーブ（再帰）性	**自分自身を呼び出す**ことができる。

プログラムが利用するデータ領域

- プログラムが使用するデータ領域にはスタックとヒープがある。

スタック領域	LIFOで管理するメモリ領域。**プログラム内部で宣言した変数（局所変数）の割当**てや**戻り番地の格納**などに用いる
ヒープ領域	リストで管理されたメモリ領域。割当てと解法の順序に関連のないデータの格納に用いる

- スタック領域はアプリケーションを起動するごとに割り当てられる領域である。プログラムが宣言した局所変数は、それを呼び出したアプリケーションのスタックに確保される。そのため、プログラムが局所変数を使用する限り、その再入可能性は保証されることになる。

8. オープンソフトウェア
オープンソフトウェアとは

- **オープンソフトウェア(Open Source Software : OSS)**は、言葉通りソースコードが開示されたソフトウェアのことであり、次の特徴をもつ。

・自由に再頒布できる
・ソースコードを入手できる
・は生物に同じライセンスを適用できる
・利用者や適用領域を差別（制限）しない

自由な再頒布は、「**有料で再頒布してもライセンスには違反しない**」ということである。実際に様々なOSSをパッケージ化したり、自社のソフトウェアを組み合わせて販売したりすることも少なくはない。

オープンソフトウェア

- **OSI(Open Source Initiative)**

OSSを促進することを目的とした組織で、OSSの定義(**OSD**)を提唱している。

- 主なOSS

OS	Linux、各種BSD系OS
Webサーバ	Apache
メールサーバ	Postfix
DNSサーバ	BIND
プロキシサーバ	Squid
アプリケーションサーバ	Tomcat

主なオープンソースソフトウェアのライセンス

- **GPL(GNU General Public License)**

GNUプロジェクトの一環として定められたライセンス。無保証、直策件表示の保持、**同一ライセンスの適用を条件**に、改変や再頒布が認められている。派生物に同一ライセンスが適用されるということは、GPLライセンスの下では派生物を含むすべてのソフトウェアがOSSであることを表している。

GNUプロジェクト
UNIX互換のソフトウェアを、すべてフリーソフトで実装することを目標としたプロジェクト

- **BSDライセンス**

BSD系のOSで採用されるライセンス。無保証、著作権およびライセンス条文の表示を条件に、改変や再頒布を認めている。**改変後の派生物はソースコードを非公開で再頒布することができる。**

BSD(Berkeley Software Distribution)
カリフォルニア大学バークレイ校が開発・配布したUNIX関連のソフトウェア

主なオープンソースソフトウェアのライセンス

- **Apacheライセンス**

Apacheソフトウェア財団が定めたライセンス。再頒布にあたっては**Apacheライセンス**のコードが使われていることを明示する必要がある。また、**改変した場合にはその告知文を追加**しなければならない。

- **デュアル（マルチ）ライセンス**

一つのソフトウェアを2種類以上のライセンスの下で頒布する方法。たとえば無料のOSSとその商用版（有料ライセンス）を組み合わせるなど、OSSやフリーソフトをビジネスに利用する場合に用いられることが多いライセンス。