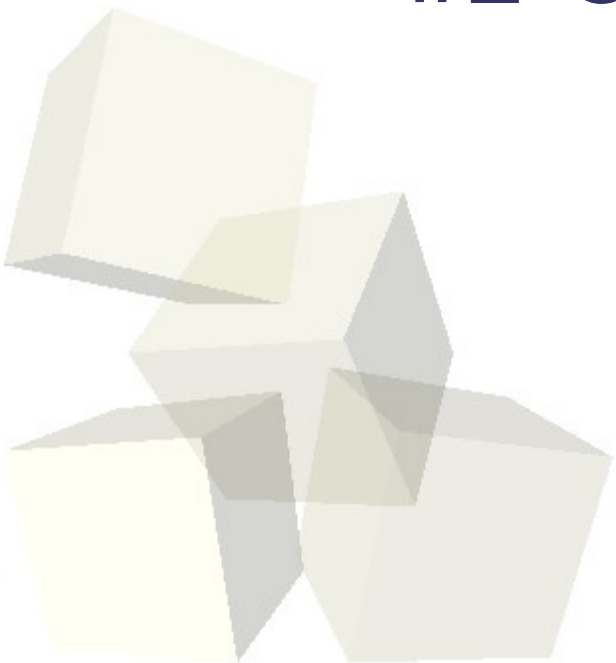
オペレーティングシステム

#2 CPUの仮想化:プロセス



おさらい：OSの目的と恩恵

- 1. リソース抽象化によるアクセス容易性
  - ハードウェアを抽象化
    - → キーボード、テンキー ⇒「入力装置」
  - プログラマはハードウェアに対するアクセスが容易に
- 2. 資源（リソース）管理による確認容易性
  - ハードウェア資源を無限にあるように見せかけ
  - プログラマは、資源の使用可否を確認するのが楽
  - 実際にバッティングしたときはOSが調停
- 3. スケジューリングによる実行効率向上
  - 仕事に応じたスケジューリング
  - スケジューリングの賢さが全体の効率に大きく影響

おさらい：仕事の単位

- プロセス
  - システムが処理する仕事の単位
  - このプロセス単位でリソースは割り当てられる
    （場合が多い）

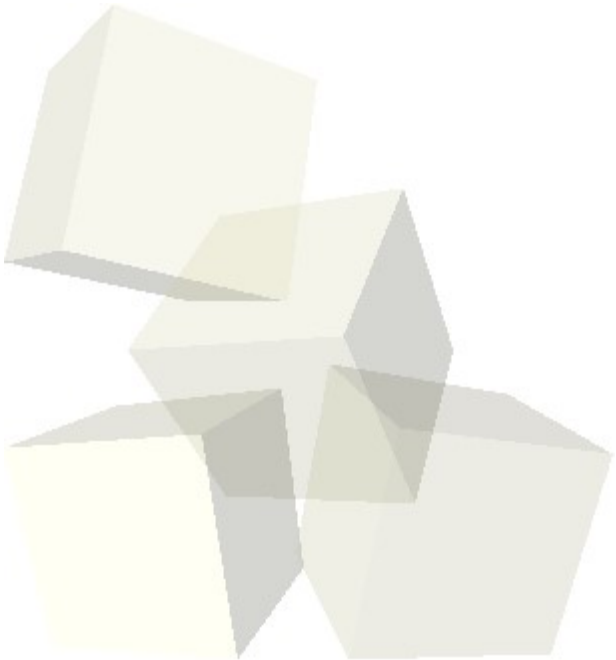
- ジョブ
  - ユーザがシステムに対して依頼する仕事の単位

おさらい：プログラムの処理形態

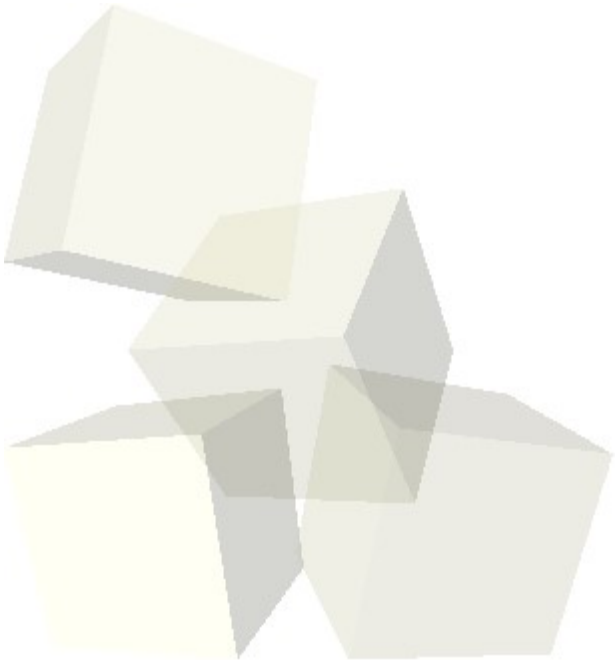
- バッチ処理
  - 必要なリソースや処理に必要なデータを前もって決定
  - 実行してほしいジョブを一括依頼
  - スケジューリングは単純
  - リソースを占有するため、他プロセスは長い待ち
- 対話（インタラクティブ）処理
  - そのつどプログラムに対して入力
  - TSS（タイムシェアリングシステム）などで、細切れのCPU時間を複数プロセスに順に割当
  - 各プロセスの待ち時間は短い
  - スケジューリングは複雑

今日の内容

- CPUの仮想化
  - 特に「プロセス」についてより詳しく
  - プロセスとスレッド
  - 割り込み



2.1
プロセスとは



プロセスとは

- くどいですが…

- プロセス
  - リソースの割当対象となる（仕事の）単位
  - OSに対してリソースを要求
  - OSからリソースの割当を受ける

プロセス(プログラム)とプロセッサ

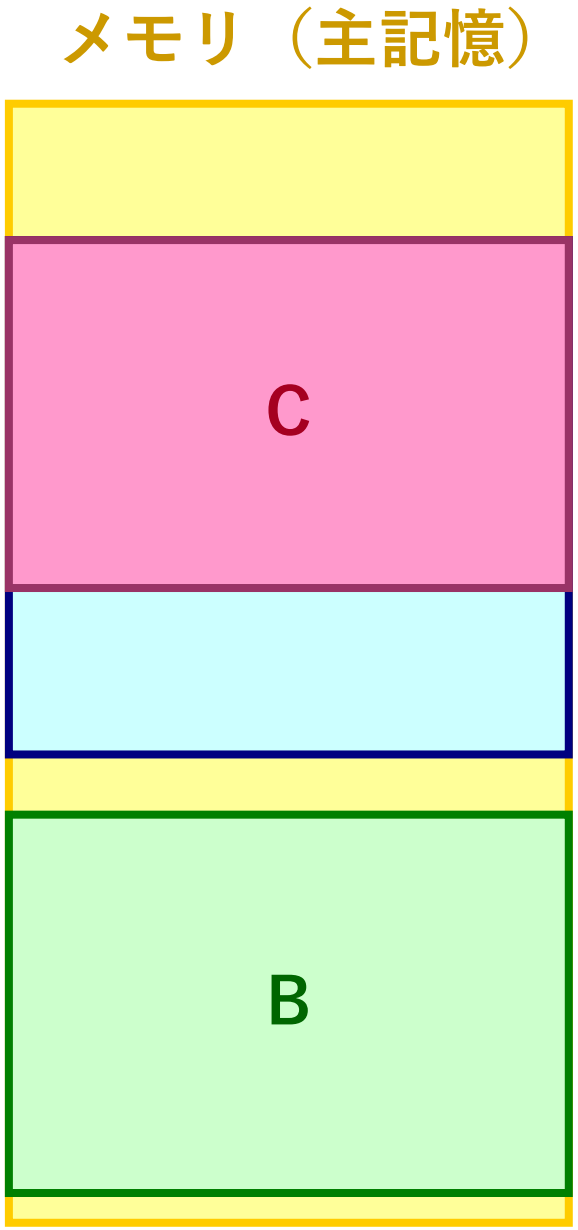
- ユニプロセッサ・ユニプログラミング
  - ひとつのCPUに対してひとつのプロセス
  - バッチ処理

- ユニプロセッサ・マルチプログラミング
  - ひとつのCPUに対して複数のプロセス
  - TSS

- マルチプロセッサ・マルチプログラミング
  - 複数のCPUに対して複数のプロセス
  - 並列・分散処理

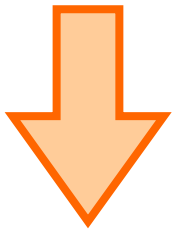
プロセスの切り替え

- 複数プロセスを切替えながら実行
  - プロセスA
  - プロセスB
    - → 記憶領域の圧迫
  - プロセスC
    - → 記憶領域の不足
    - → 置き換えコスト
  - プロセスA
    - → また不足
    - → また置き換え



プロセスとスレッド

- 複数プロセスの同時実行はコストが高い
  - メモリ使用量が増加
  - 切り替えコストも大きい
- 複数CPUを備えた計算機の一般化
  - デュアルプロセッサ，デュアルコア
  - 同時実行できるプロセス数よりCPUが多いと
    CPUが遊んでいてもったいない



- スレッド
  - プロセスをさらに小さい単位に分割
  - CPUリソースをスレッドごとに割当

スレッド

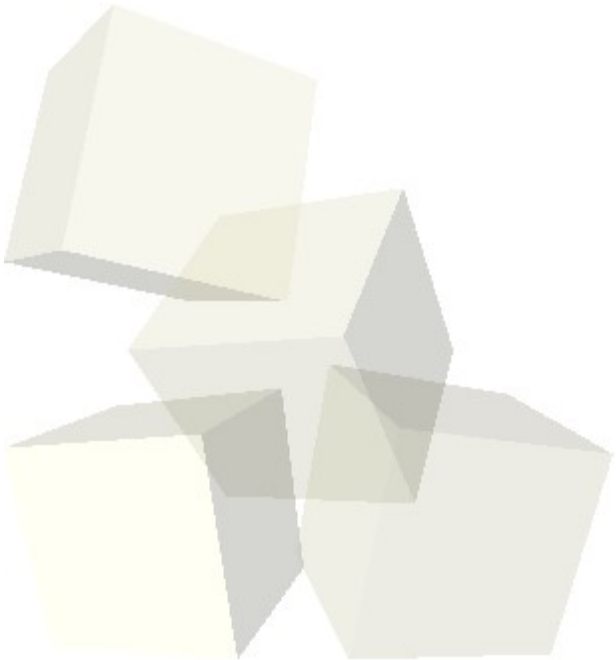
- 例）Microsoft Office
  - プロセス
    - → Microsoft Word
    - → Microsoft Excel
    - → 各プログラムはプロセスとして処理
  - スレッド
    - → たとえばWordの場合
    - → 印刷
    - → 編集
    - → など、同じ「Word」というプログラムの中で、
      同時（並行）動作できる単位がある！

スレッド

- リソース割当
  - プロセス単位
    - → メモリ，入出力デバイス，etc...
  - スレッド単位
    - → CPU

- スレッド（2．5参照）
  - TSSによる切り替えオーバヘッドが軽い
    - → 同一プロセスから生成されてるからメモリ領域が同じ
    - → メモリ使用量は1プロセス分ですむ
  - 別名：Light Weight Process（軽いプロセス）

2.2
割込み



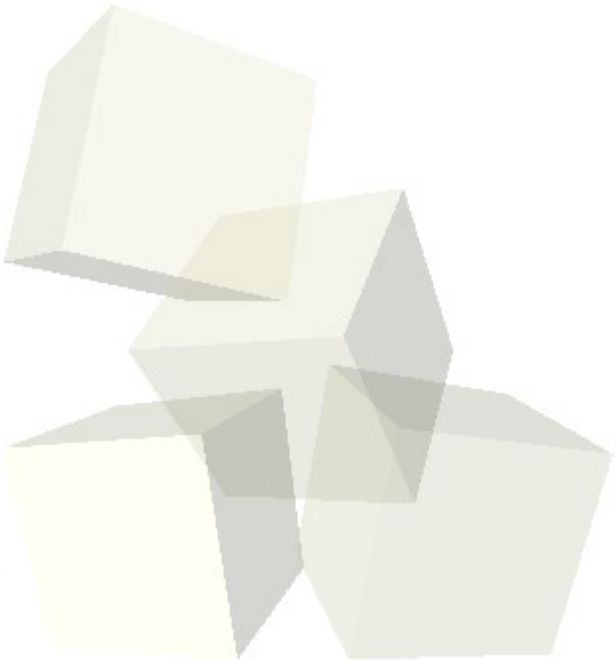
プロセスの切り替え

- CPUの仮想化
  - OSがプロセス・スレッドに対して
    CPUの実行権を微小時間与える
- 割込み
  - 通常のCPU演算動作とは異なる事象のこと
    - → キーボード入力を受け取った
    - → 自動車がどこかに衝突した
    - → サーバからデータが送られてきた
  - 割込み発生時にプロセスの切り替えが起こる
  - TSSでは、プロセス切り替えのために
    インターバルタイマーが定期的に割込みを発生

割込み

割込み処理

- 割込みは、即座に処理すべき場合が多い
- 高速かつ軽量に割込みを処理する実行方式



割込みの種類

- 内部割込み
  - 実行中のプログラムを発生原因とする
  - 例）プログラム自体が他の処理を要求
    プログラム自体の異常

- 外部割込み
  - その他の要因で発生する
  - 例）他の優先的処理からの要求
    順番待ちしていた他の処理への移行
    ハードウェア異常
    特殊な処理

割込みの種類

- 内部割込み
  - スーパバイザコール割込み
  - プログラムチェック(例外)割込み
- 外部割込み
  - 入出力割込み
  - タイマ割込み
  - マシンチェック割込み
  - リスタート割込み

割込みの種類

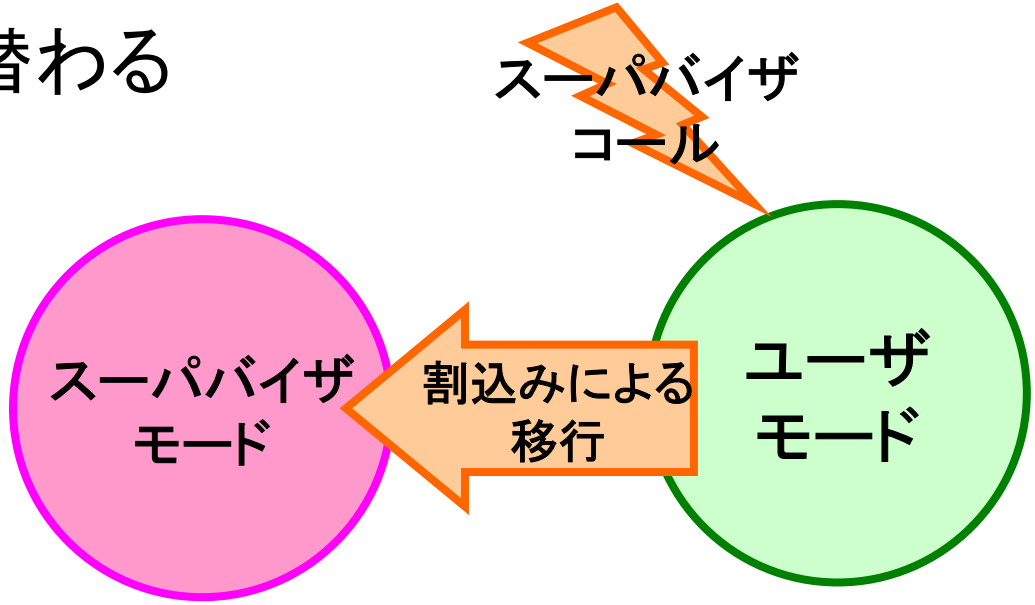
- 内部割込み
  - スーパバイザコール割込み
  - プログラムチェック(例外)割込み
- 外部割込み
  - 入出力割込み
  - タイマ割込み
  - マシンチェック割込み
  - リスタート割込み

内部割込み:
スーパバイザコール割込み

- ユーザモード
  - アプリケーションには許されていない処理がある
    - → プロセスの切り替え
    - → 入出力デバイスへのアクセス
    - → etc..
- スーパバイザモード
  - そこでアプリケーションは、OSに対して処理を依頼
  - OSの権限で，処理を実行してもらう
- スーパバイザコール
  - アプリケーションがOSに処理を依頼すること

内部割込み:
スーパバイザコール割込み

- スーパバイザコール
  - このとき割込みが発生
  - CPUの実行モードが切り替わる



- CPUの実行モード
  - スーパバイザモード
    - → OSを実行するモード
    - → CPU内の全てのリソースを利用可能
  - ユーザモード
    - → アプリケーションを実行するモード
    - → 利用できるリソースに制限あり

割込みの種類

- 内部割込み
  - スーパバイザコール割込み
  - プログラムチェック(例外)割込み
- 外部割込み
  - 入出力割込み
  - タイマ割込み
  - マシンチェック割込み
  - リスタート割込み

プログラムチェック（例外）割込み

- 実行中のプログラムで異常が発生したとき
  - ゼロによる除算

```
division by zero
```

  - 演算時のオーバフロー

```
integer overflow
```

  - 不正なメモリアドレスへのアクセス

```
segmentation violation
```

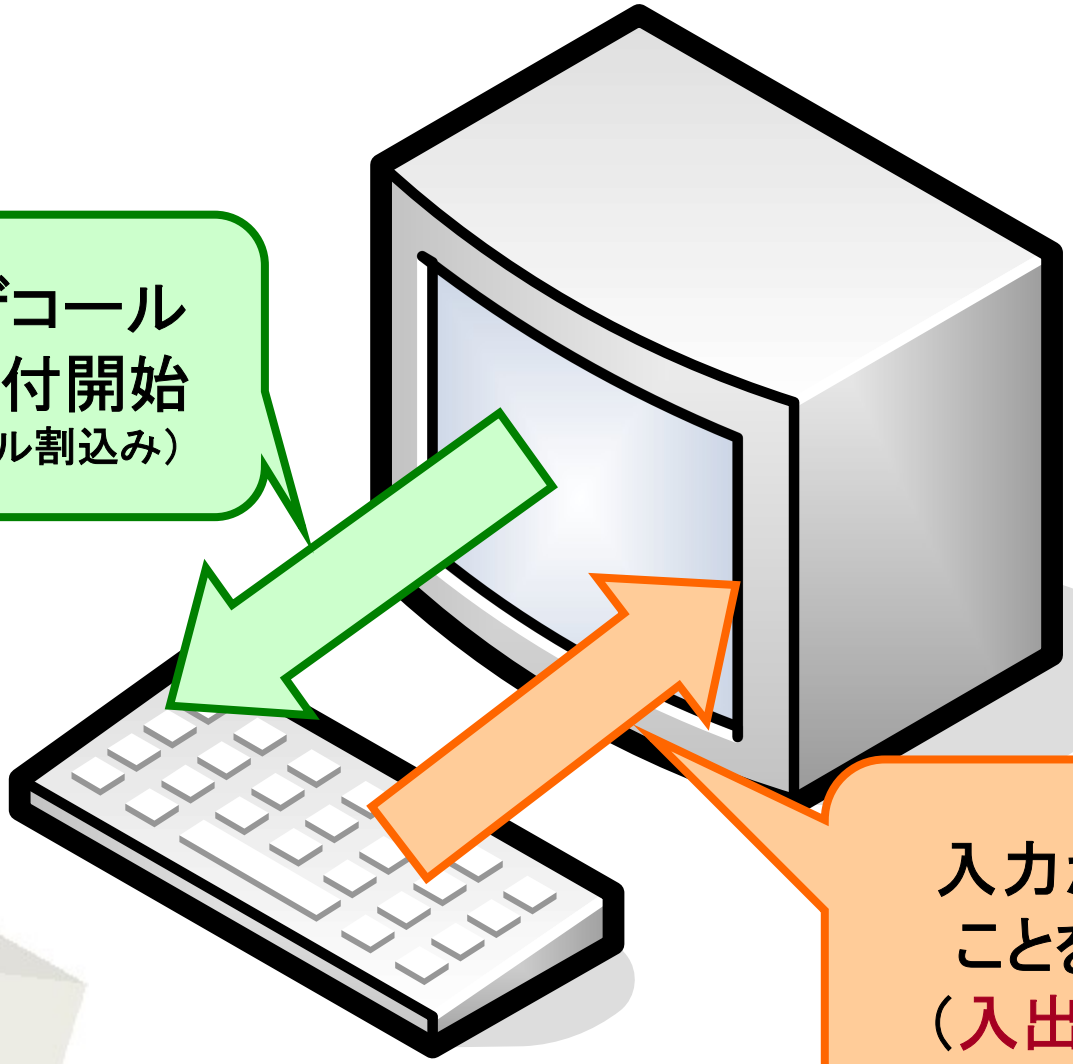
- この割込みを検知するしくみがないと…
  - 上記の異常が発生したときに
    システム全体が停止してしまうかも…

割込みの種類

- 内部割込み
  - スーパバイザコール割込み
  - プログラムチェック(例外)割込み
- 外部割込み
  - 入出力割込み
  - タイマ割込み
  - マシンチェック割込み
  - リスタート割込み

外部割込み:
入出力割込み

- 入出力装置から発生する割込み

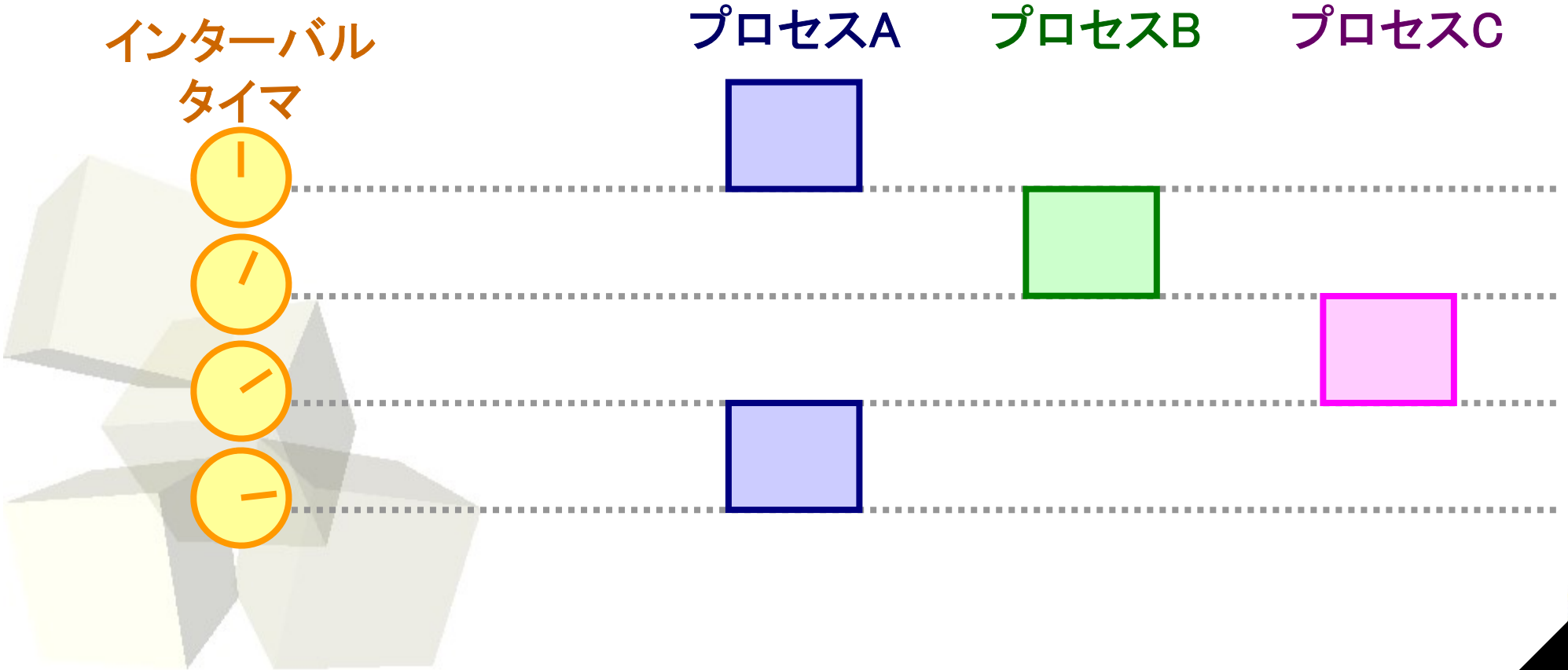


割込みの種類

- 内部割込み
  - スーパバイザコール割込み
  - プログラムチェック(例外)割込み
- 外部割込み
  - 入出力割込み
  - タイマ割込み
  - マシンチェック割込み
  - リスタート割込み

外部割込み:
タイマ割込み

- インターバルタイマによる割込み
  - TSSでは，定期的な切り替えが必要
  - インターバルタイマが定期的に割込みを発生させることで、これを実現

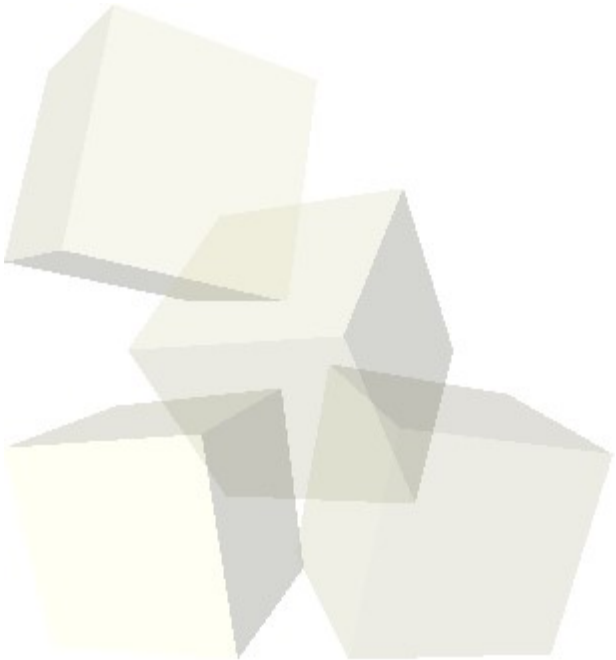


割込みの種類

- 内部割込み
  - スーパバイザコール割込み
  - プログラムチェック(例外)割込み
- 外部割込み
  - 入出力割込み
  - タイマ割込み
  - マシンチェック割込み
  - リスタート割込み

外部割込み:
マシンチェック割込み

- ハードウェアによって通知される異常時に発生する割り込み
  - 冷却装置の異常
    - → 内部の温度が上がりすぎているのを検出
  - 電源装置の異常
  - etc...



外部割込み:
リスタート割込み

- システムをリセットするときに発生する割込み

割込みの種類:まとめ

- 内部割込み
  - スーパバイザコール割込み
  - プログラムチェック(例外)割込み
- 外部割込み
  - 入出力割込み
  - タイマ割込み
  - マシンチェック割込み
  - リスタート割込み

2.3
割込みによる
プロセスの中断と再開

割込み発生時の処理

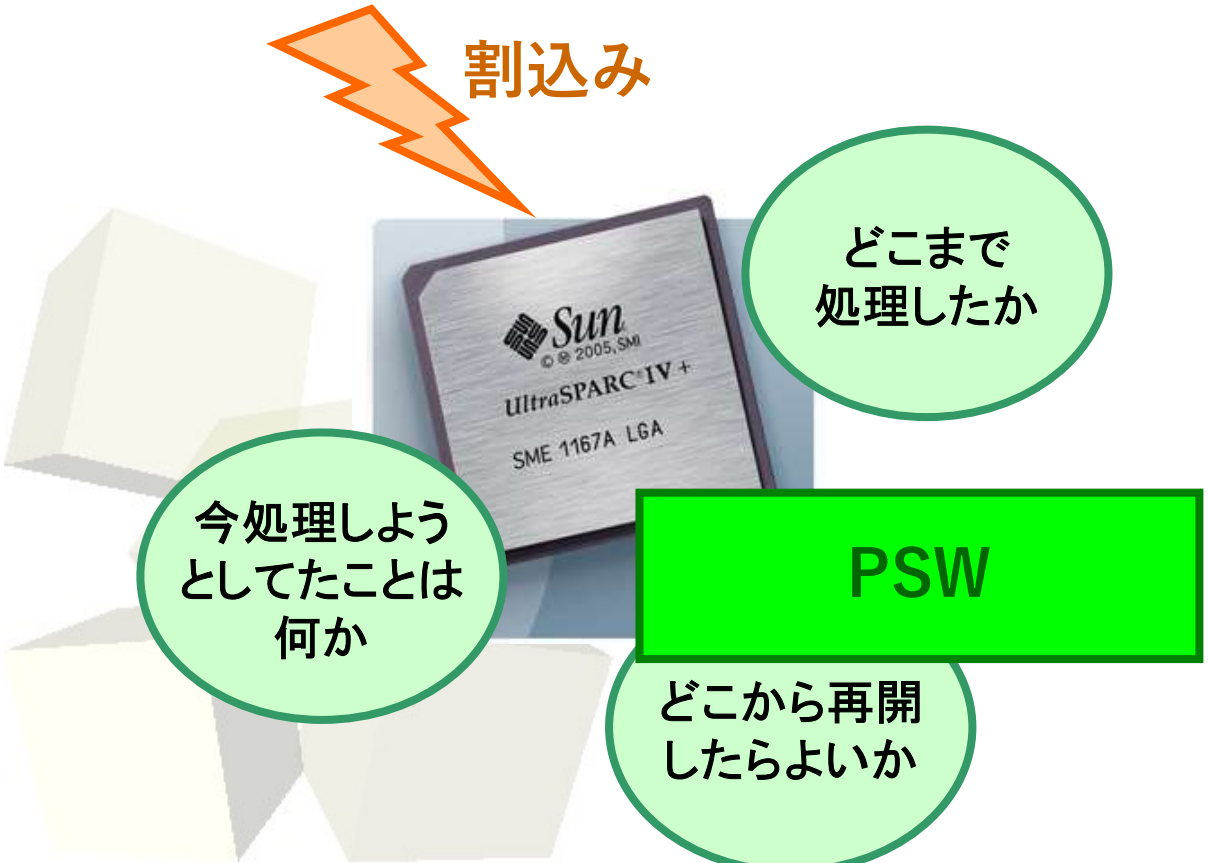
- 実行中のプロセスを中断
- 割込み処理ルーチン(ハンドラ)に移行
- 割込み処理が終わったら、プロセスを再開

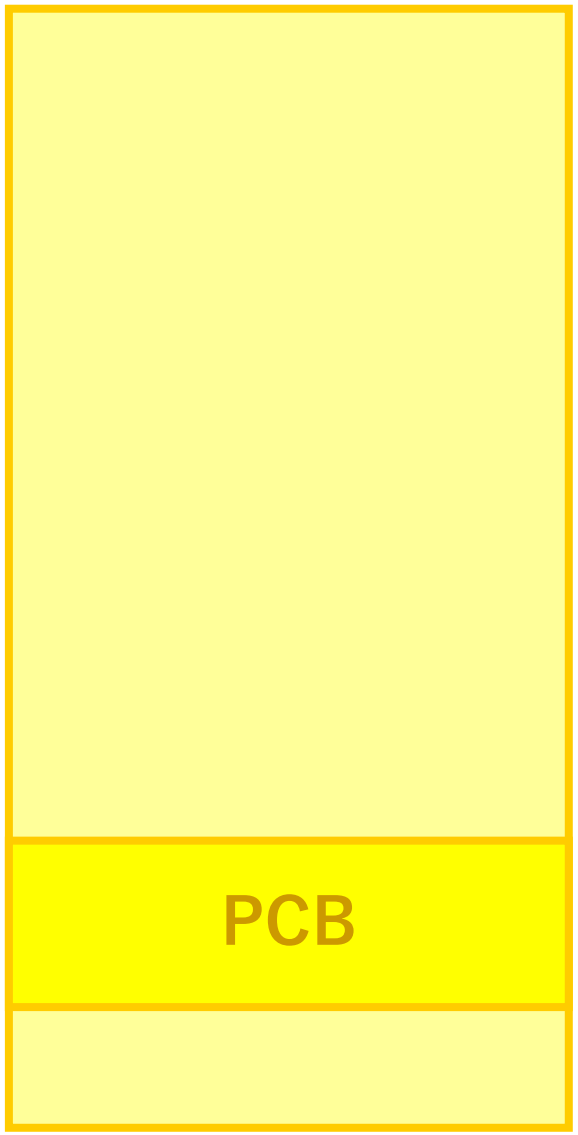
プロセスの中断

- PSW（Program Status Word）
  - プロセスは、後で再開しないといけない
  - 再開するためには、今の途中状態を覚えておかないといけない
  - 状態：
    - → プログラムカウンタの値
    - → スタックレジスタの値
    - → 汎用レジスタの値
    - → 割込みマスクの値
    - → etc...

プロセスの中断

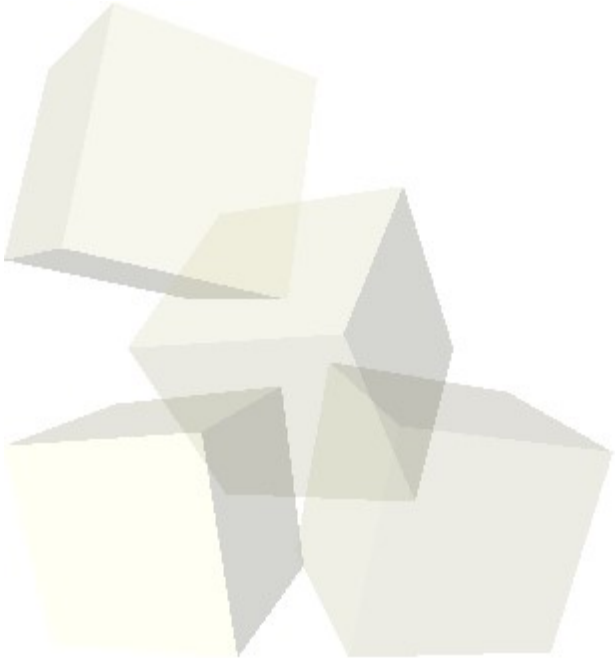
- PCB (Process Control Block)
  - メモリ上の、PSWを退避するための領域





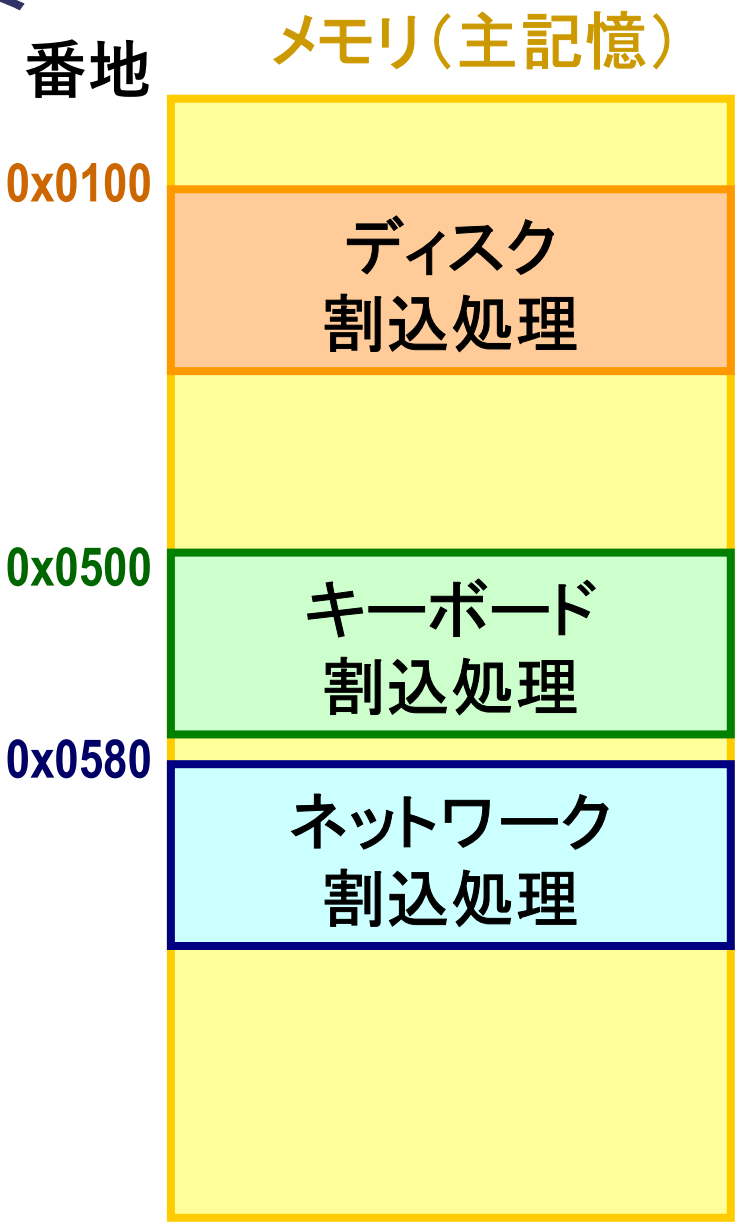
割込み処理ルーチンの仕事

- 割込みの種類を判別する必要
  - 種類に応じて処理を実行
- 割込みベクタ
  - 割込みの種類に対応する数字(ID)



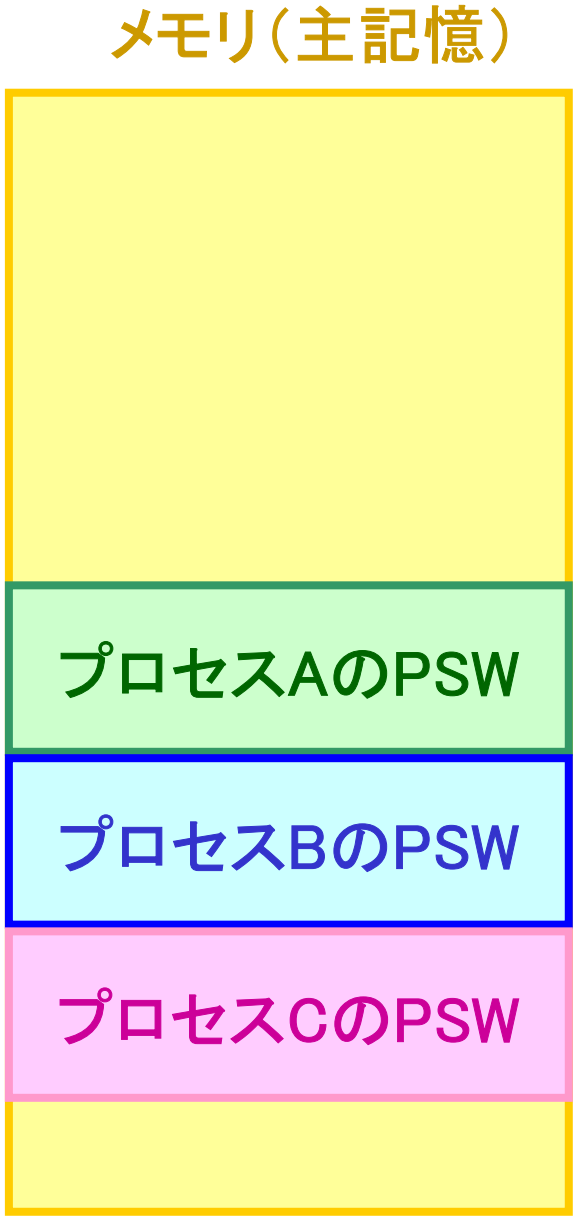
割り込みベクタ
テーブル

0	0x0580
1	0x0100
2	0x0500
:	:



プロセスの再開

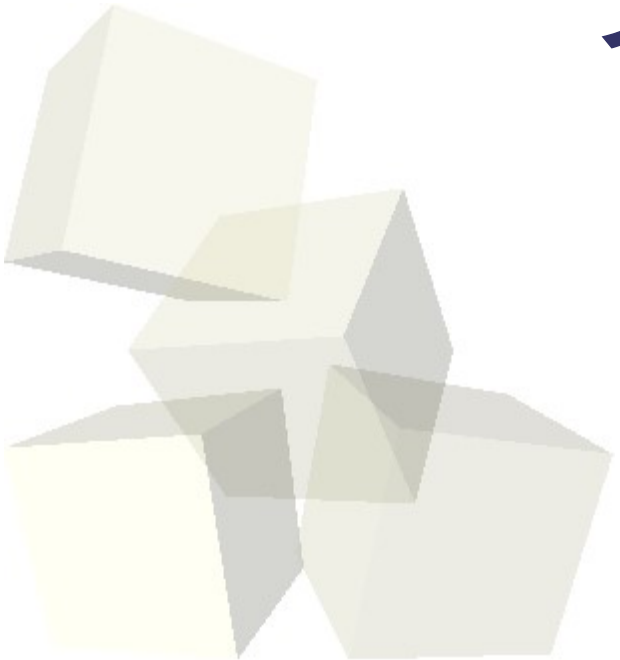
- 実行可能なプロセスから
  プロセスを選択
  - 割込によって中断された
    プロセスが常に再開される
    わけではない
- 選択されたプロセスの
  PSWからCPU状態を復元
  して再開



プロセスの中断と再開:まとめ

- 中断
  - CPU状態をPSWという形で、
    メモリ内のPCBへ保存
- 割込ルーチン(ハンドラ)
  - 割込ベクタ(割込の種類を示す値)を放送
  - その値を割込ベクタテーブルでひいて、
    割込に対応するルーチンの主記憶アドレスを取得
  - ルーチン実行
- 再開
  - 実行可能プロセスからスケジューラが1つ選択
  - 対応するPSWからCPU状態を復元

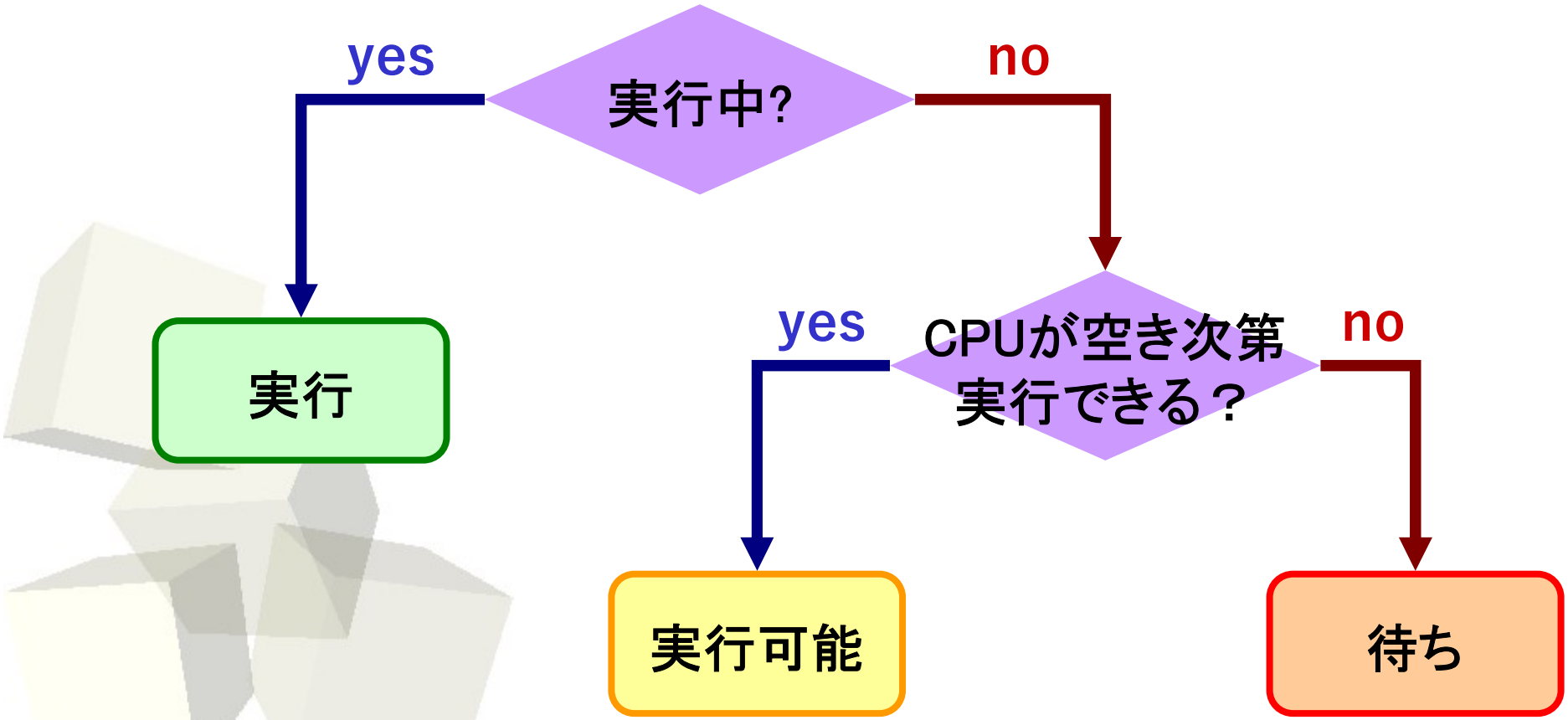
2.4
プロセスの三状態



プロセスの状態

- 「実行可能なプロセス」とは?

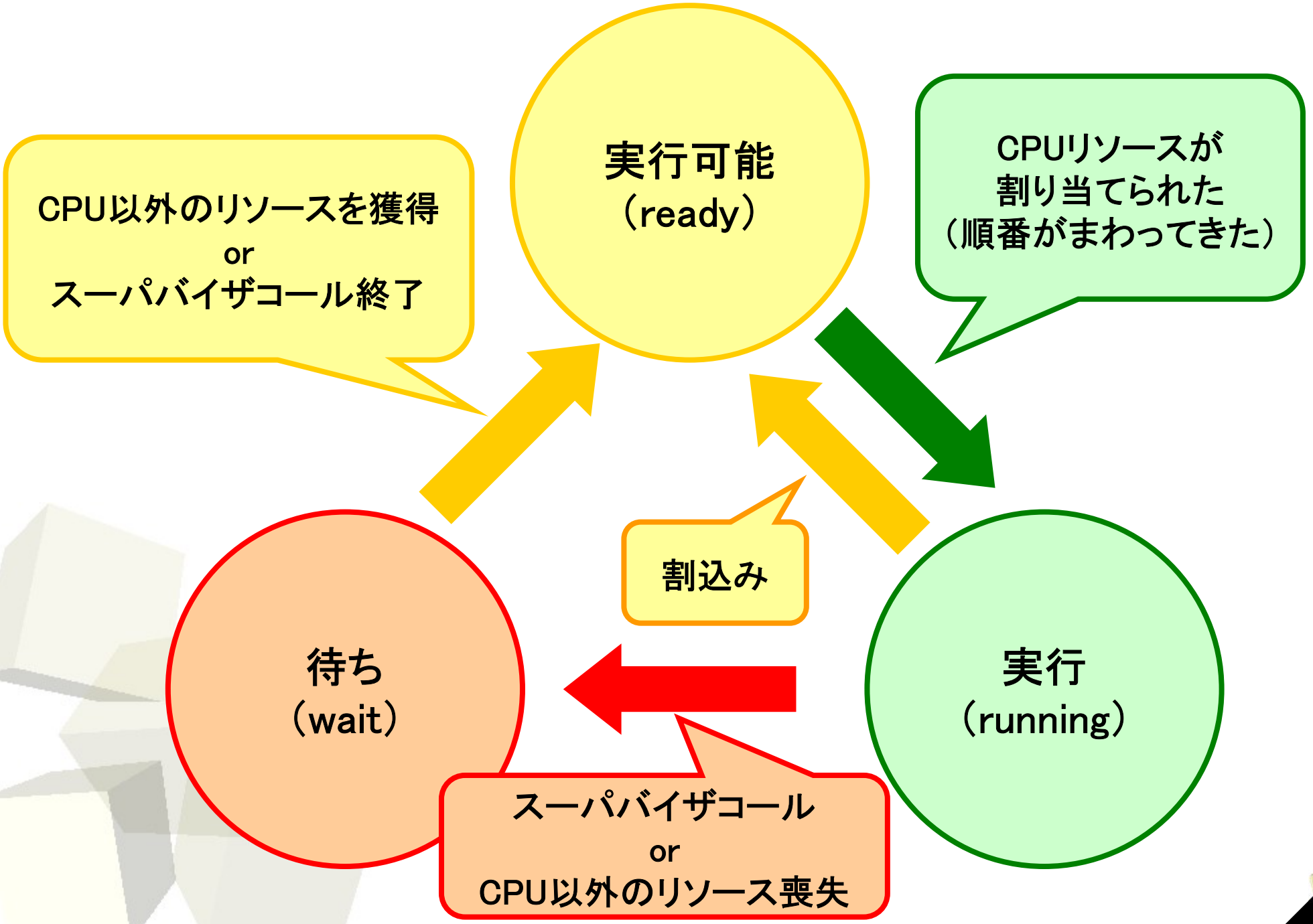
- プロセスの状態



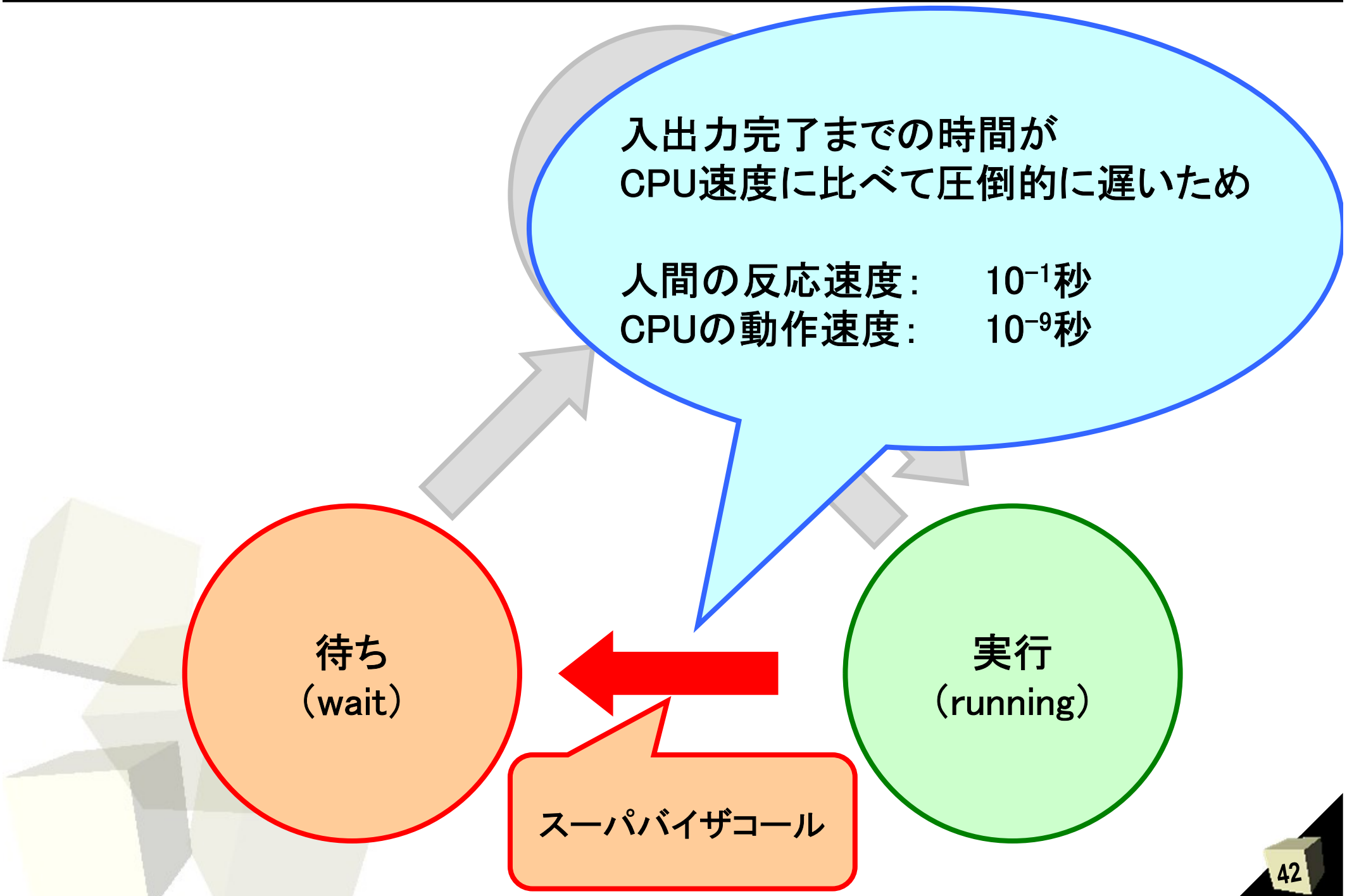
プロセスの状態

- 実行状態（running）
  - プロセスを実行している状態
  - リソースは，そのプロセスのために確保されている
- 実行可能状態（ready）
  - 実行できるが、CPUリソースが確保できていない状態
  - CPUリソースを確保した時点で実行開始される
- 待ち状態（wait）
  - CPU以外のリソースも確保できていない状態
  - 入力待ちなどもこれに含まれる

状態遷移


実行可能
（ready）
CPUリソースが
割り当てられた
（順番がまわってきた）
実行
（running）
割込み
スーパバイザコール
or
CPU以外のリソース喪失
待ち
（wait）
CPU以外のリソースを獲得
or
スーパバイザコール終了

状態遷移


入出力完了までの時間が
CPU速度に比べて圧倒的に遅いため
人間の反応速度：　10^{-1}秒
CPUの動作速度：　10^{-9}秒
待ち
（wait）
実行
（running）
スーパバイザコール

スレッド

CPU
コア

CPU
コア

プロセス

レジスタ群

レジスタ群

主記憶

プログラム領域

静的変数領域

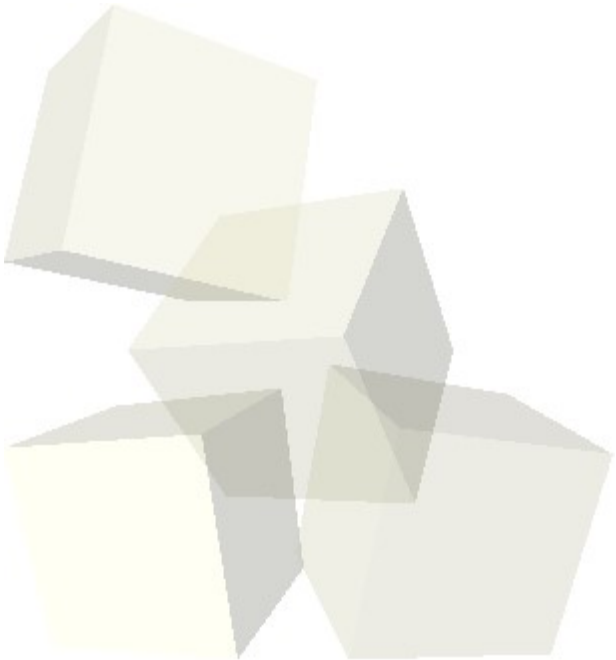
動的変数領域

PSW領域

スタック領域

スタック領域

コラム
CPUの仮想化

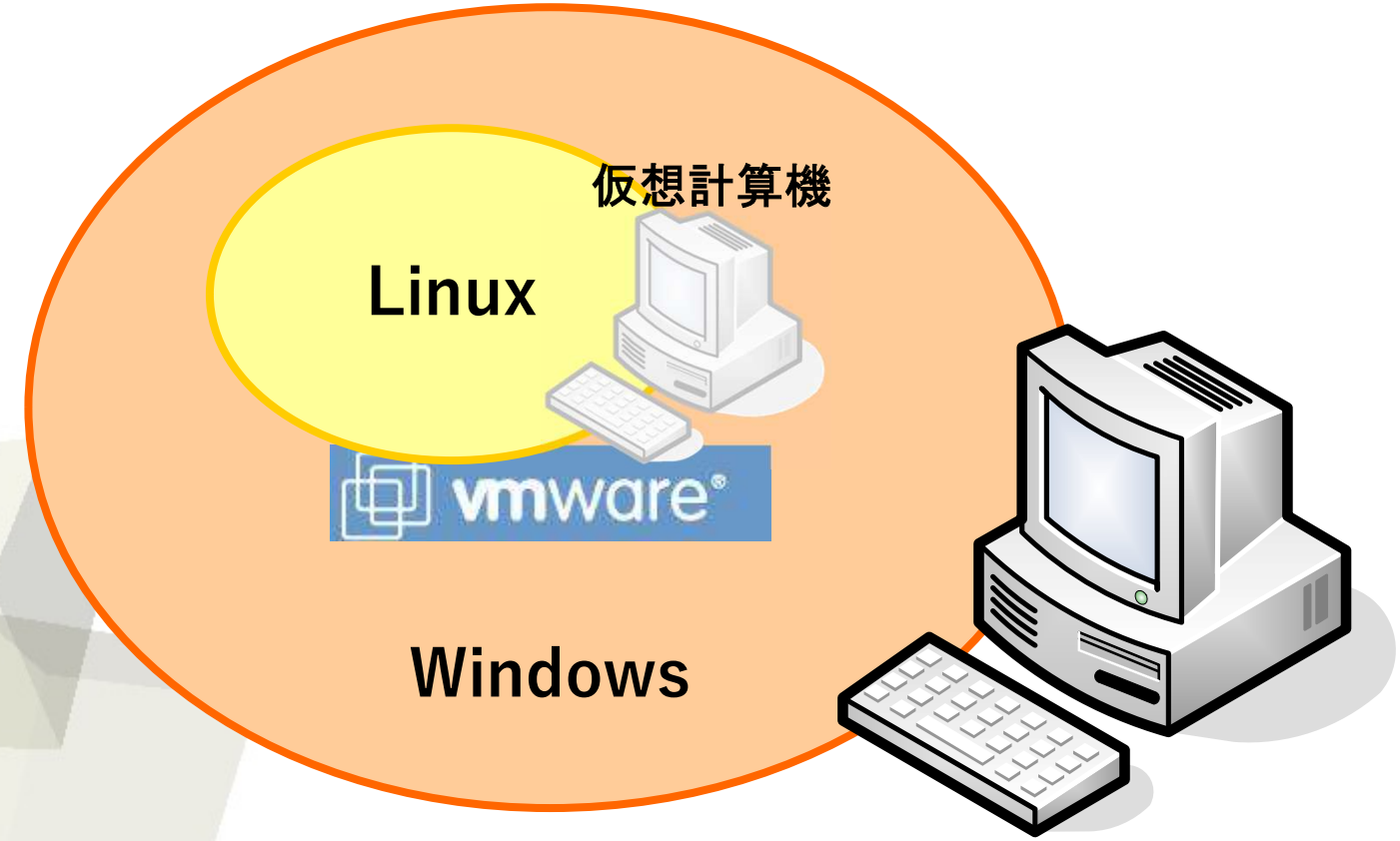


エミュレーション

- 最近は...
  - ハードウェアが非常に高速化
  - 他のハードウェア資源全体(システム)を仮想化することも可能になってきた
- エミュレーション
  - ハードウェア環境をソフトウェアで仮想化
  - 計算機上で他の計算機環境を仮想的に提供

ハードウェアエミュレーション

- 計算機の構造そのものを仮想化
  - VMware
    - → IBM/PC環境のOS（Solaris, Linux, Windows）上に仮想的なIBM/PC環境を構築



さらなる勉強へのキーワード

- 仮想計算機（バーチャルマシン）とコンテナ
  - 仮想マシンは、仮想的なハードウエア全てをソフトウエアで構築する
    - → 主記憶リソースなどを圧迫する
  - OSを論理的に分割することにより1つのOS内に複数のパーティションを作成し、そのパーティション間は完全に分離
    - → コンテナ
    - → https://employment.en-japan.com/engineerhub/entry/2019/02/05/103000
    - → 現状Googleなどでは、コンテナ環境が主流
      - ggrks (^0^)=> kubernetes, Docker