大容量問題と オーバーレイ

大容量問題

- 物理メモリの容量より大きいプログラムを実行させたい、という問題
- 昔　⇒　<オーバーレイ技法>
  - 結局、不便・使いにくい
- 仮想記憶の出現
  - 大容量問題をうまく解決

大容量問題

- 物理メモリの容量より大きいプログラムを実行させたい、という問題
- 昔　⇒　<オーバーレイ技法>
  - 結局、不便・使いにくい
- 仮想記憶の出現
  - 大容量問題をうまく解決

こちらが主テーマ

大容量メモリへの欲求

- メモリスペースの容量はいつも足りなかった

大容量メモリへの欲求

- メモリスペースの容量はいつも足りなかった
  - ステップ数が多い(長い)プログラムを走らせたい
    ステップ数多い ⇒ 命令数多い ⇒ メモリ不足

大容量メモリへの欲求

- メモリスペースの容量はいつも足りなかった
  - ステップ数が多い(長い)プログラムを走らせたい
    ステップ数多い ⇒ 命令数多い ⇒ メモリ不足
  - データの多いプログラムを走らせたい
    データが多い ⇒ メモリ不足

大容量メモリへの欲求

- メモリスペースの容量はいつも足りなかった
  - ステップ数が多い(長い)プログラムを走らせたい
    ステップ数多い ⇒ 命令数多い ⇒ メモリ不足
  - データの多いプログラムを走らせたい
    データが多い ⇒ メモリ不足
- メモリは高価で、簡単に増やせなかった
  （今の感覚からは想像できないほど小さかった）

大容量メモリへの欲求

- メモリスペースの容量はいつも足りなかった
  - ステップ数が多い(長い)プログラムを走らせたい
    ステップ数多い ⇒ 命令数多い ⇒ メモリ不足
  - データの多いプログラムを走らせたい
    データが多い ⇒ メモリ不足
- メモリは高価で、簡単に増やせなかった
  （今の感覚からは想像できないほど小さかった）
  - 何とかしたい
    ⇒ ディスクと入替えながら使う工夫

ディスクと入替えながら使う

- 前提
  - CPUはメモリ上の命令しか実行できない
  - CPUはメモリ上のデータしか直接に演算できない

ディスクと入替えながら使う

- 前提
  - CPUはメモリ上の命令しか実行できない
  - CPUはメモリ上のデータしか直接に演算できない
- 今実行していない部分はメモリから追出せる
  - ディスクにおいて置けばよい

東邦大学

ディスクと入替えながら使う

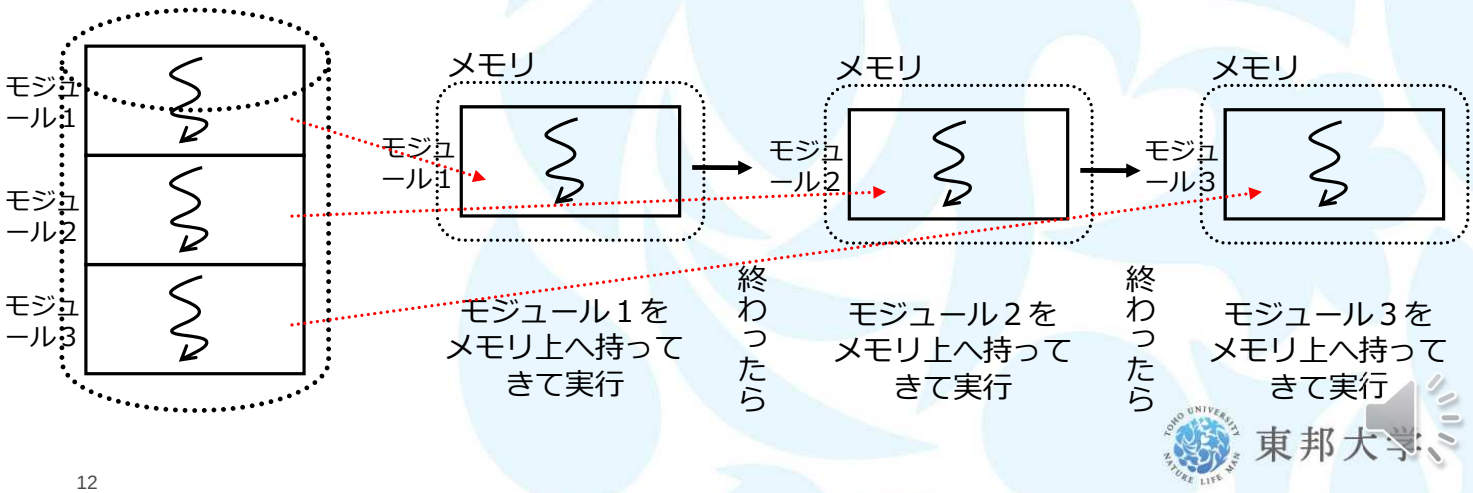
- 前提
  - CPUはメモリ上の命令しか実行できない
  - CPUはメモリ上のデータしか直接に演算できない
- 今実行していない部分はメモリから追出せる
  - ディスクにおいて置けばよい
  - ディスクからメモリへ移すのに時間がかかるので
    今＋少し後の命令・データもメモリに置きたい

ディスクと入替えながら使う

- 前提
  - CPUはメモリ上の命令しか実行できない
  - CPUはメモリ上のデータしか直接に演算できない
- 今実行していない部分はメモリから追出せる
  - ディスクにおいて置けばよい
  - ディスクからメモリへ移すのに時間がかかるので
    今＋少し後の命令・データもメモリに置きたい
  - プログラムの構造から考えて
    しばらく使わない命令・データをディスクへ

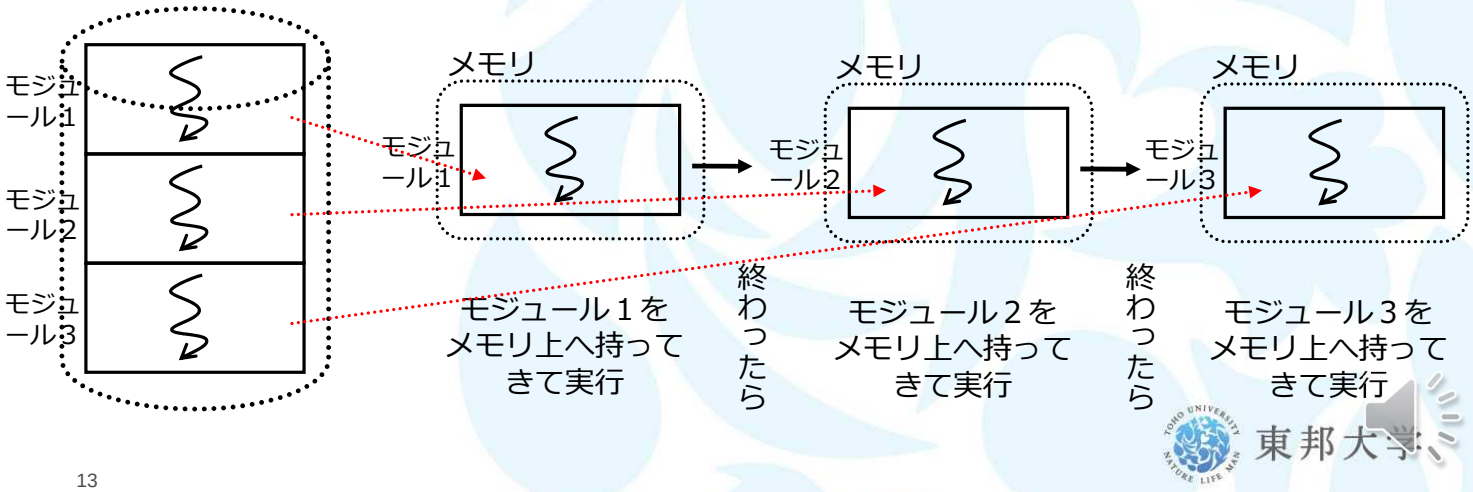
オーバーレイ

- オーバーレイ（Overlay）は
  - プログラムをモジュールに分割する
  - モジュール単位でディスクからメモリにもってくる



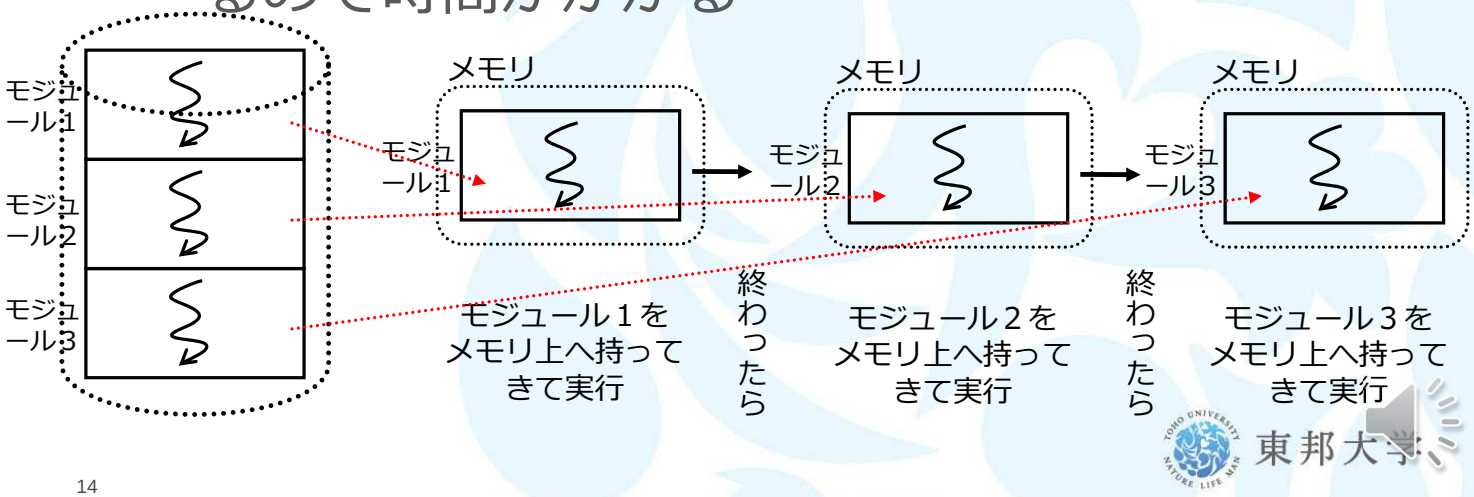
オーバーレイ

- 分割の仕方はプログラマが決める
  - 区切りのよい所（行ったり来たりしない）で分割する



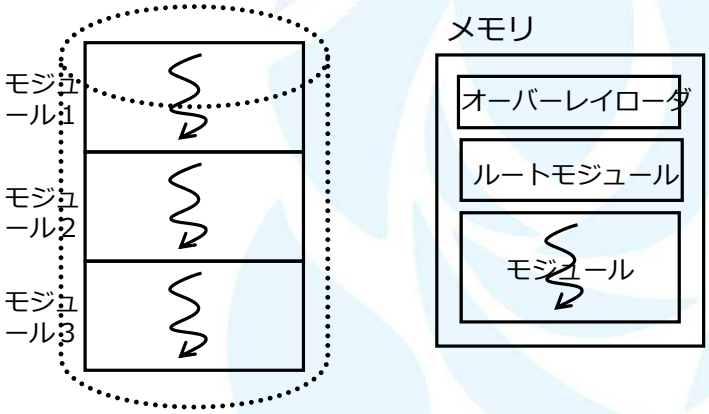
オーバーレイ

- 分割の仕方はプログラマが決める
  - 区切りのよい所（行ったり来たりしない）で分割する
    行ったり来たりするとそのたびにメモリを入替えるので時間がかかる



オーバーレイ

- プログラマ(OSのユーザ)が実行を管理する
  - ルートモジュール　全体の流れを制御する
  - オーバーレイローダ　モジュールをディスクからロードする（持ってくる）



オーバーレイ

- 分割の仕方を決めるのはプログラマ
  - 行ったり来たりがなるべく無いように
  - データの受け渡しがなるべく少ないように

オーバーレイ

- 分割の仕方を決めるのはプログラマ
  - 行ったり来たりがなるべく無いように
  - データの受け渡しがなるべく少ないように
- モジュール間でデータの受け渡しが必要なら
  そのやり方を考えるのもプログラマ

オーバーレイ

- 分割の仕方を決めるのはプログラマ
  - 行ったり来たりがなるべく無いように
  - データの受け渡しがなるべく少ないように
- モジュール間でデータの受け渡しが必要なら
  そのやり方を考えるのもプログラマ

↓

- プログラマの負担が大きい

オーバーレイ

- 分割の仕方を決めるのはプログラマ
  - 行ったり来たりがなるべく無いように
  - データの受け渡しがなるべく少ないように
- モジュール間でデータの受け渡しが必要なら
  そのやり方を考えるのもプログラマ

↓

- プログラマの負担が大きい
- 別の機械へ移ってメモリ容量が変わると
  分割の仕方を再設計する必要　⇒　やっかい

オーバーレイ

- というわけで、オーバーレイは
  かなりやっかいな方法だった

オーバーレイ

- というわけで、オーバーレイは
  かなりやっかいな方法だった

↓

- 次の項目で学ぶ「デマンドページング」は
  これをきれいに（？）解決した

オーバーレイ

- というわけで、オーバーレイは
  かなりやっかいな方法だった

↓

- 次の項目で学ぶ「デマンドページング」は
  これをきれいに（？）解決した
  - システムの持つ物理メモリの大きさに制約されず
    大きなプログラムが実行できる

  （プログラミングするとき、メモリの大きさの
  制約を考えたことなんてないでしょう？

ここまでのまとめ

- オーバーレイはメモリに入りきらない大きな
  プログラムを実行する手段
- プログラムをモジュールに分割して、
  １つだけをだけをメモリに置いて実行する
  - なるべく行き来しないように分割
- 分割作業も実行中の入替もプログラマの責任
  - プログラマにとって負担が大
- 「デマンドページング」により解消

オーバーレイの仕組が理解できましたか？

次へ