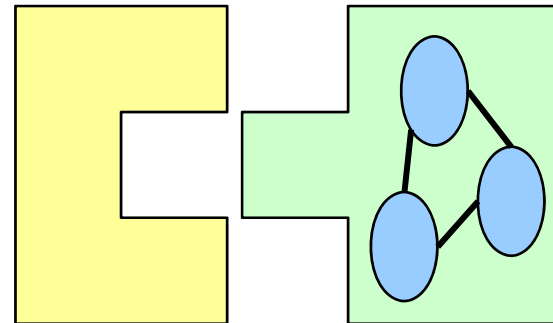
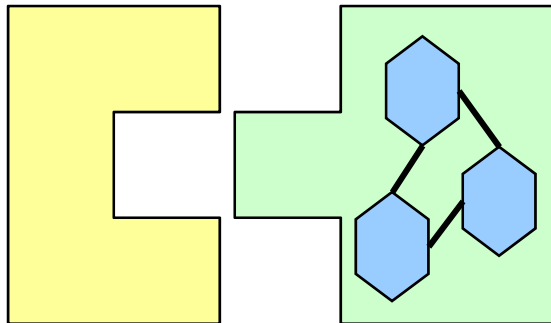


本日の話題

- 仮想化
- ブートストラップ
- 「複雑さ」の問題

仮想化 (1/2)

- Virtual とは？
 - 「仮の」という意味と「実質的な」という意味がある
 - 「本当はない」より「実質的にはある」の方が重要
- 実体を隠して、別の姿を見せる技術
 - 部品の置換・交換が楽になる

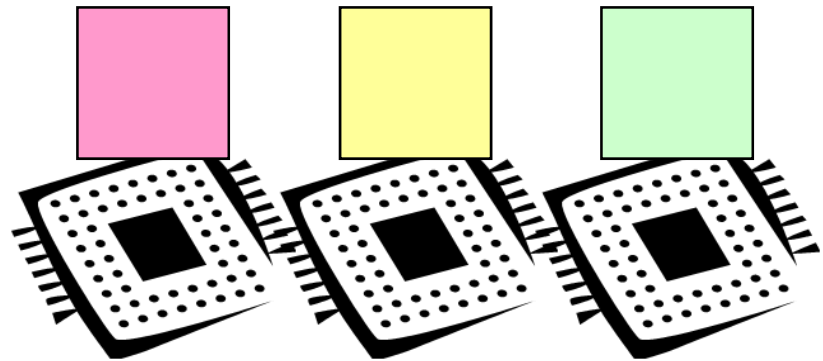
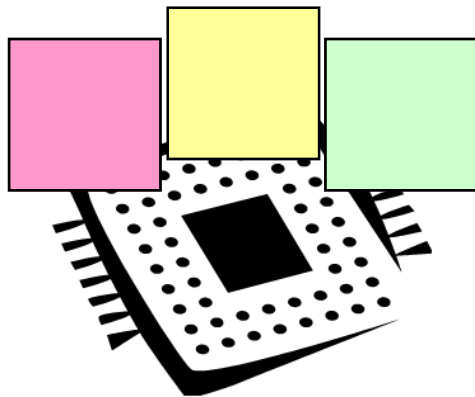


仮想化 (2/2)

- コンピュータシステムの有効利用のために、仮想化のアイデアを用いることが多い
 - CPU の仮想化
 - メモリの仮想化
 - ストレージの仮想化
 - マシン全体の可能化

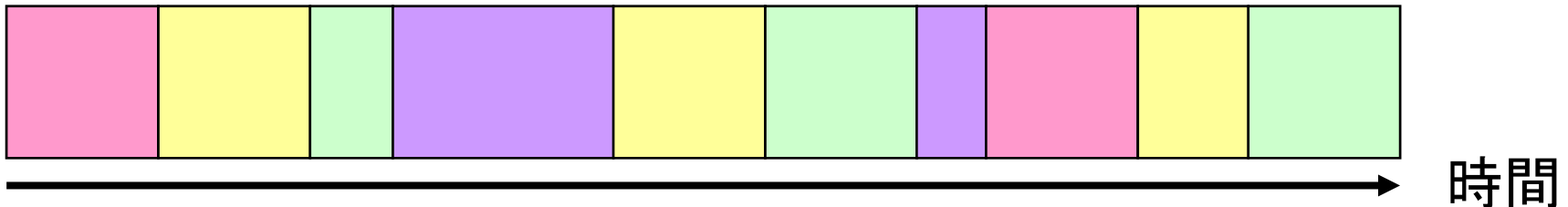
CPU の仮想化 (1/3)

- 複数のアプリケーションやユーザで1個のCPUを共有し, しかも, それぞれが占有している幻想を抱かせる
 - i.e. 多重化(multiplexing)
- 元々はコンピュータが高価だった時代に有効利用するために使われた技術



CPU の仮想化 (2/3)

- 時分割(time slicing)を行う
 - 1個のCPUでは、各瞬間に一つのアプリケーション(または OS)しか動かない
 - (注)最近の CPU の中には複数のアプリケーションを(本当に)同時処理できるものもある
 - 短時間で次々に切り替え(context switch)を行う
 - 通常 0.1秒以下



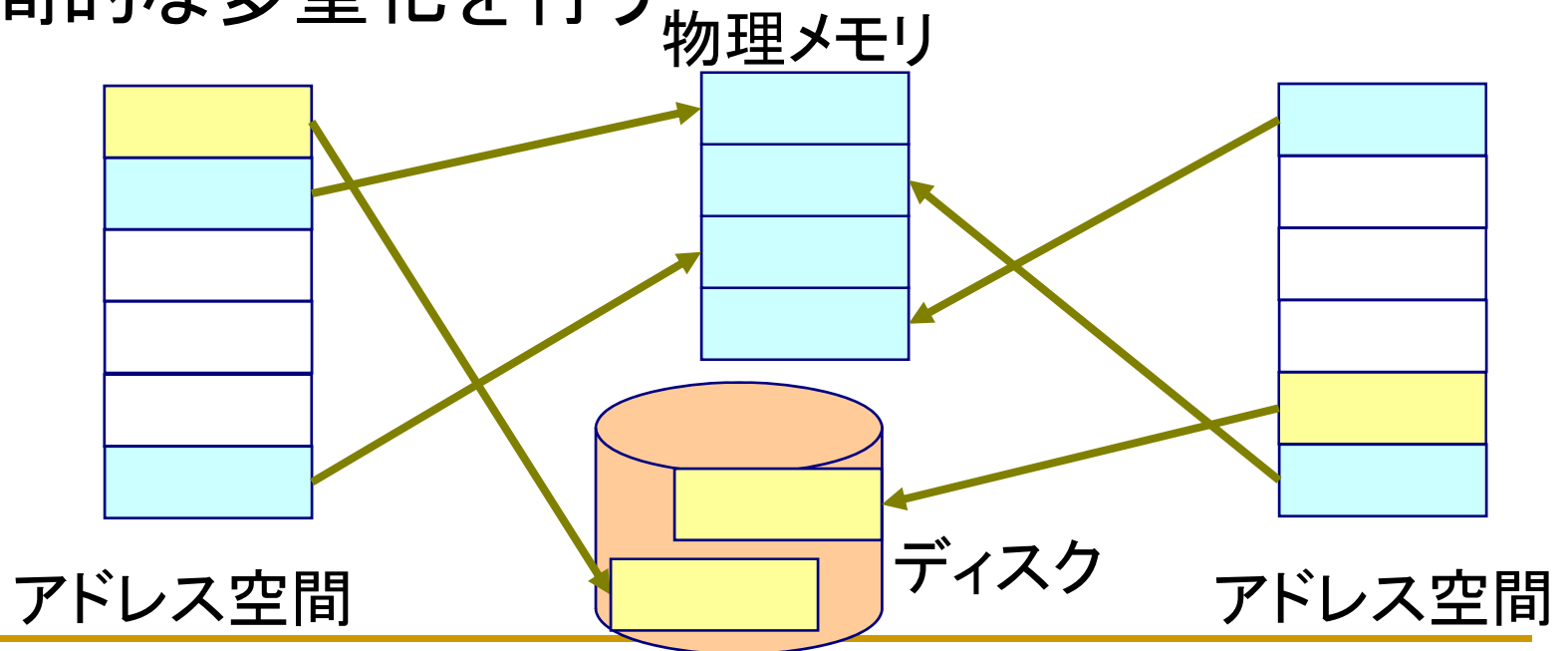
CPU の仮想化 (3/3)

■ メカニズムの概略

- コンピュータ内蔵のタイマー(timer)が, 適当な間隔でタイマー割り込みを発生させる
 - タイマーは, CPUの処理とは独立に, カウントダウン処理を行う
- 割り込みが発生すると, CPU によるプログラムの処理(機械語の実行)が一旦中断され, OS に制御が移る
- OS は次に動かすアプリケーションを決定し, 切り替えに必要な処理を行う

メモリの仮想化 (1/2)

- 各アプリケーションが独立したアドレス空間を占有している幻想を抱かせる
 - 全空間に物理メモリが割り当てられる保証はない
 - メモリが足りなければディスクに頼ることもある
- 空間的な多重化を行う



メモリの仮想化 (2/2)

■ メカニズムの概要

- 物理メモリが割り当てられていない論理アドレスにプログラムがアクセスしようとする時、ページフォールト (page fault) 割り込みが発生する
- 割り込みが発生すると、CPU によるプログラムの処理が一旦中断され、OS に制御が移る
- OS は、物理メモリを確保し、必要ならディスクから読み込みを行い、論理アドレスと物理アドレスの対応表を更新し、制御を元のプログラムに戻す

■ アドレス対応表の管理には謎な点がある

- メモリアクセスに使う対応表もメモリ上に格納される
- 興味があれば、TLB (Translation Lookaside Buffer) というキーワードを調べると良いだろう

ストレージの仮想化 (1/3)

- 一般アプリケーション向けの仮想化
 - (物理的に存在する) ディスク等の存在を隠す
 - (物理的には存在しない) ファイルが存在するような幻想を抱かせる
- OS向けの仮想化
 - (実際には存在しない) 仮想ディスクが存在するような幻想を抱かせる
 - 一般家庭等ではあまり使われない機能

ストレージの仮想化 (2/3)

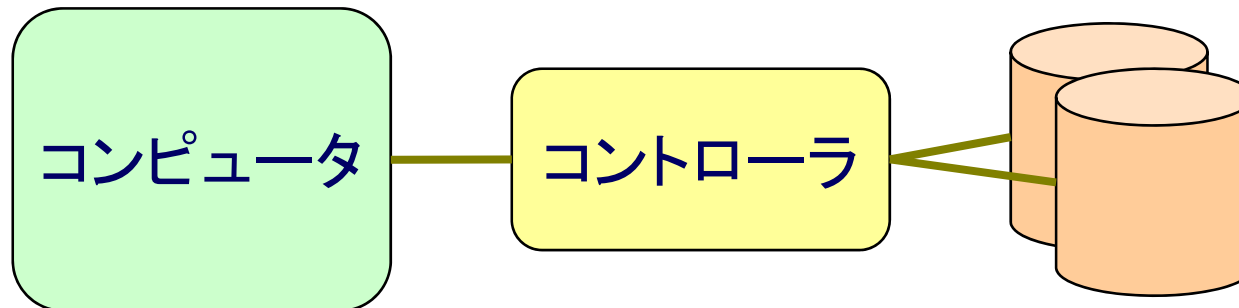
■ 仮想的なディスクの例

□ ミラーリング

- 同じ容量の2台のディスクに同じデータを書き込む
- 1台が故障してもデータの消失を免れる

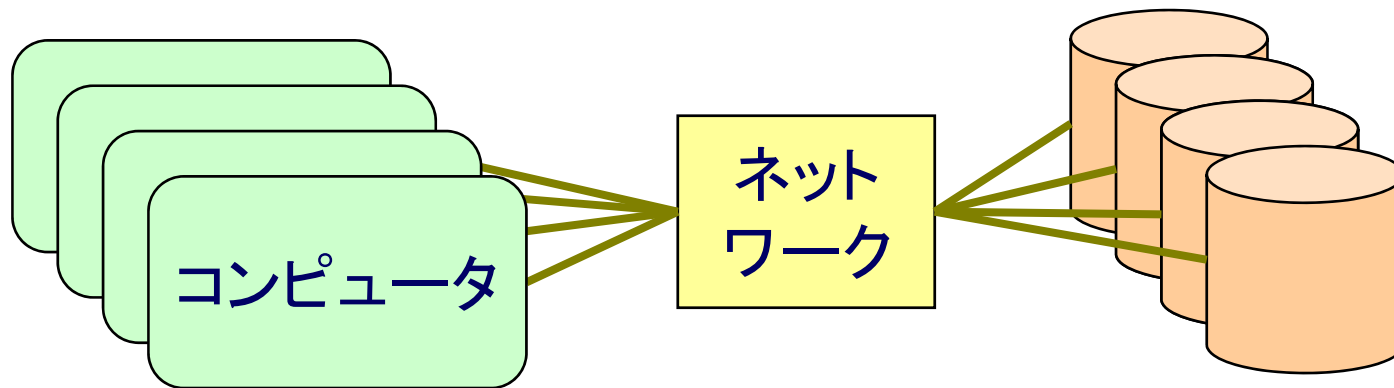
□ 論理ボリューム

- 複数のディスクを使い、大きな一塊の記憶領域(ボリューム)があるように見せかける
- 後からディスクを追加することもできる



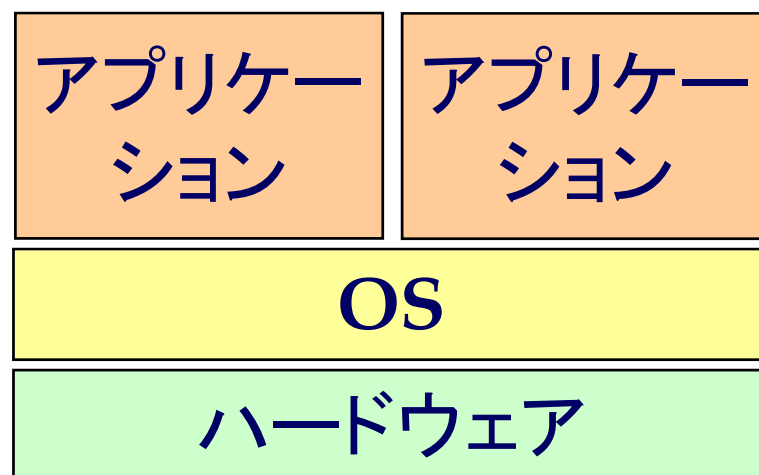
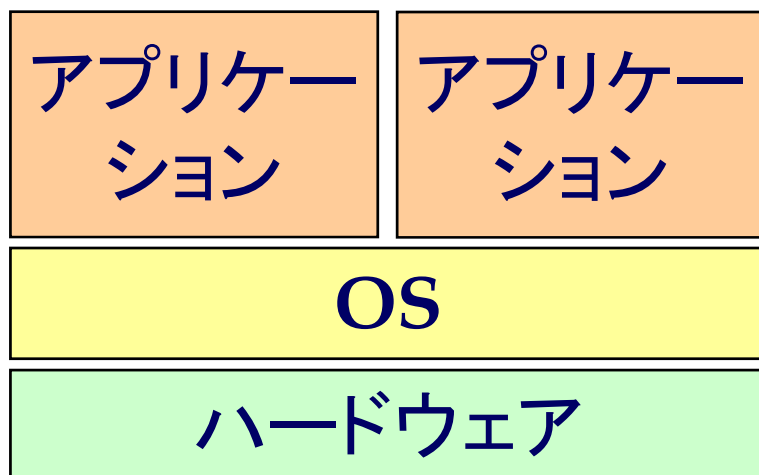
ストレージの仮想化 (3/3)

- ストレージプールと仮想ボリューム
 - ディスクをストレージ用ネットワークにたくさんつなげる
 - 必要なだけ使って仮想ボリュームを作る
 - ディスクやコンピュータの物理的台数に依存しない形でシステムを組める



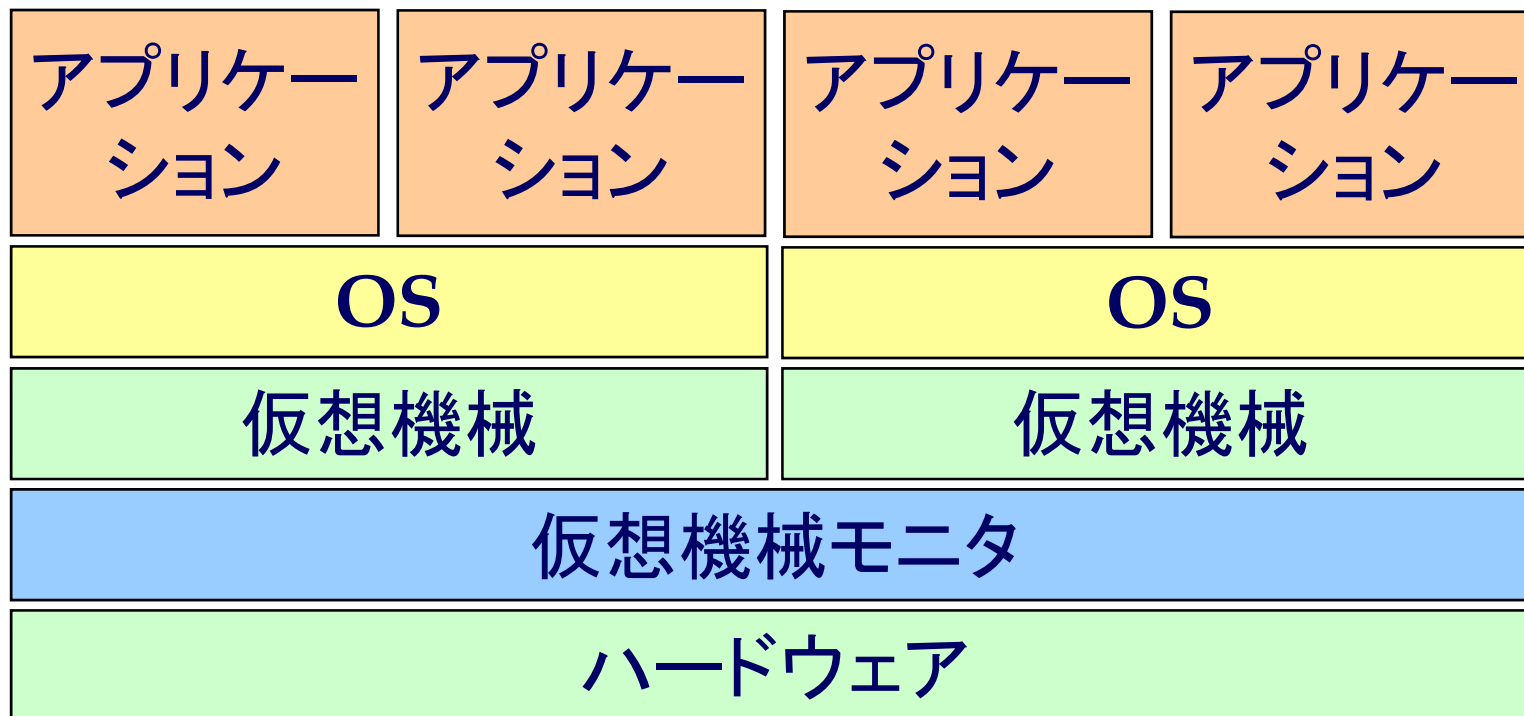
マシン全体の仮想化 (1/2)

- 通常は、各コンピュータでOSが一つ動き、OSの機能を使って各種アプリケーションが動く
- 一台のコンピュータで複数のOSを動かしたいこともある
 - e.g. 特定OS専用アプリケーションの利用
 - e.g. 動作チェック



マシン全体の仮想化 (2/2)

- ハードウェアの機能を仮想的に実現
 - 機械語や割り込みの挙動が OS から見て同じなら良い



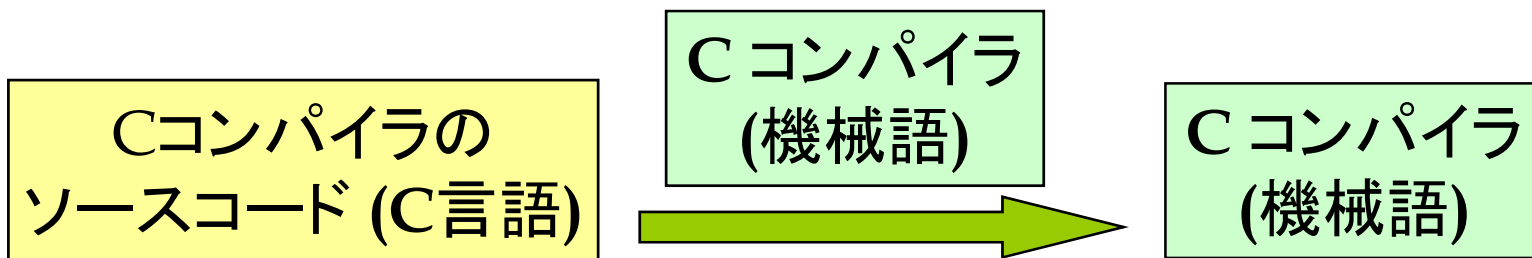
復習：部品の必要性

- ーから作るのは無理
- cf. カレーの作り方
 - レトルトパックを買ってきて作る
 - カレールー, 肉, 野菜を買ってきて作る
 - 香辛料や野菜の栽培, 牛や豚の牧畜からはじめる
 - 畑や牧草地, 鍋の材料となる銅の鉱山, 燃料となる天然ガスのガス田などの開拓からはじめる
 - 玉ねぎや牛の進化の過程を再現するところからはじめる

では、一番最初はどうするのか？

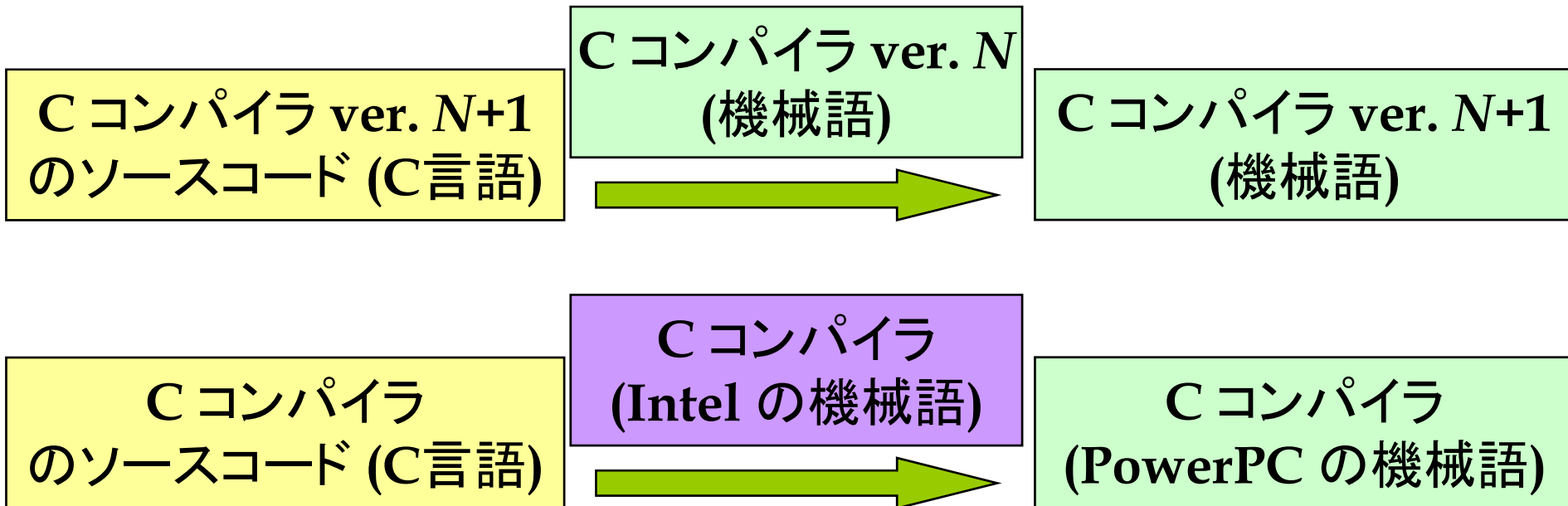
ブートストラップ (1/5)

- コンパイラの作り方には謎な点がある
 - C 言語で書いたプログラムを実行するためには、まずコンパイラ(e.g. gcc) を使って機械語に翻訳する
 - コンパイラもまたソフトウェアであり、たいていは C 言語で書いてある
 - したがって、コンパイラを実行する前に、コンパイラ自身をコンパイルする必要がある



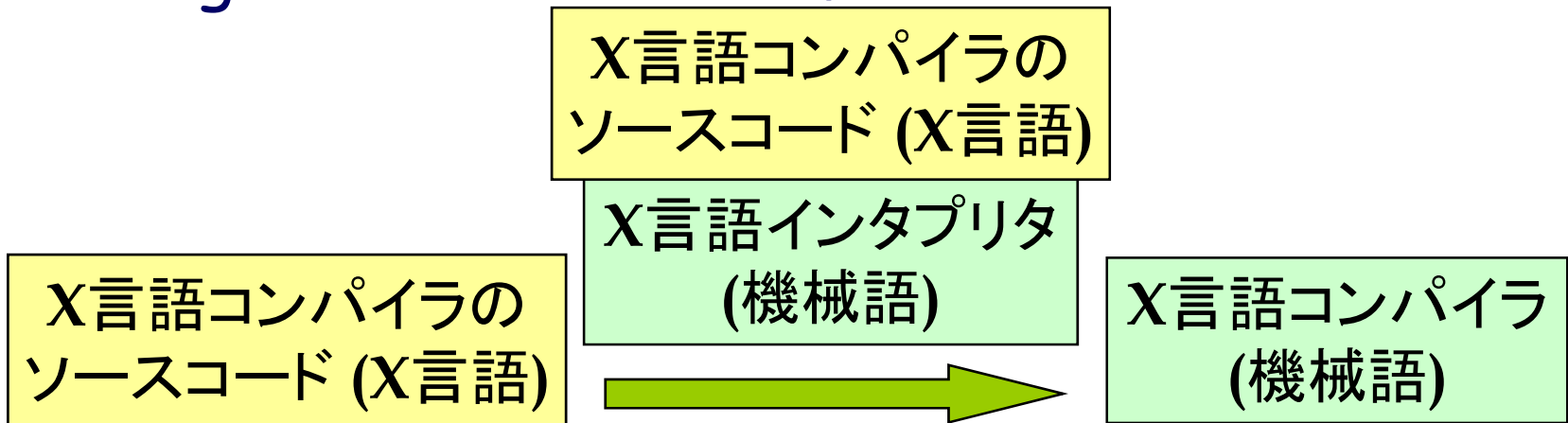
ブートストラップ (2/5)

- コンパイラが一度できてしまえば何とかなる
 - e.g. 古いバージョンで新しいバージョンをコンパイルする
 - e.g. 別の種類のコンピュータ上でコンパイルする



ブートストラップ (3/5)

- 結局, 最初のバージョンが問題
 - e.g. コンパイラ ver. 1 はあらかじめ機械語で書く
 - e.g. 機能限定版のコンパイラ ver. 0 を機械語で書き, コンパイラ ver. 1 は ver. 0 がサポートしている機能だけで実現する
 - e.g. インタプリタを機械語で書く



ブートストラップ (4/5)

- 循環的依存関係はいたるところで発生している
 - e.g. C 言語のコンパイラを書くために, C 言語で書かれたエディタを使う
 - e.g. C 言語のコンパイラを実行するために, C 言語で書かれた OS の機能を使う
 - e.g. OS の本体は, 起動時にファイルから読まれるが, ファイルから読むのは OS の機能
- 最初に最低限必要なものだけ別の方法で用意するのが原則

ブートストラップ (5/5)

- 循環的依存関係があり、一見不可能に思える立ち上げ作業のことを、ブートストラッピング (bootstrapping) と呼ぶことがある
 - 自分の靴紐(bootstrap)を引っ張り、自分自身を持ち上げるような作業
- 転じて、OS の起動のことをブート(boot)と呼ぶようになった

計算のドグマ

- 計算(computation)は、機械的で単純な操作の繰り返し
 - i.e. 機械(computer)で実現可能
 - i.e. 数学的に定義可能
- 多くの現象(自然現象, 社会現象, 仮想現象など)は計算でシミュレート可能
 - i.e. 単純な操作の繰り返しで複雑な現象に迫れる
 - cf. 「知能を計算で実現できるか？」等の哲学的議論はいろいろある

「複雑さ」の問題 (1/3)

- 計算の量的な複雑さ
 - コンピュータの仕事が多すぎる
 - 昨年11月現在の世界最速スーパーコンピュータの演算性能は 478TFlops
 - i.e. 巨大な行列計算を行う場合に, 1.5×10^{22} 回/年程度の浮動小数点数演算が可能
 - cf. アボガドロ定数は約 6.02×10^{23}
 - cf. 地球の質量は約 6.0×10^{24} kg

「複雑さ」の問題 (2/3)

- 記述の複雑さ
 - 人間の仕事が多すぎる
 - 表は, Debian Gnu/Linux と Windows の肥大化の歴史

年	Ver.	百万行
2000	2.2	55-59
2002	3.0	104
2005	3.1	215
2007	4.0	283

年	Ver.	百万行
1993	NT 3.1	6
1994	NT 3.5	10
1996	NT 4.0	16
2000	2000	29
2002	XP	40
2005	Vista β2	50

http://en.wikipedia.org/wiki/Source_lines_of_code より

「複雑さ」の問題 (3/3)

■ 分析の複雑さ

- 計算の対象（自然現象，社会現象，仮想現象など）が複雑でよくわからない
- コンピュータ自体が複雑でよくわからない
- 図では省略したが，ユーザも複雑

