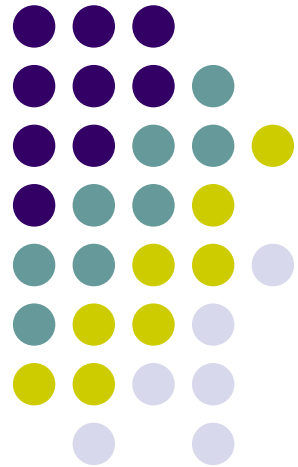
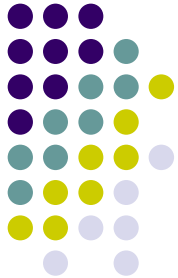
情報処理概論

工学部 物質科学工学科
応用化学コース
機能物質化学クラス



第3回

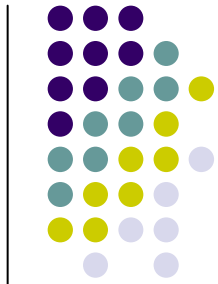
2005年　4月28日



○　計算機に関する基礎知識

- Fortranプログラムの基本構造
- 文字や数値を画面に表示する
- コンパイル時のエラーへの対処

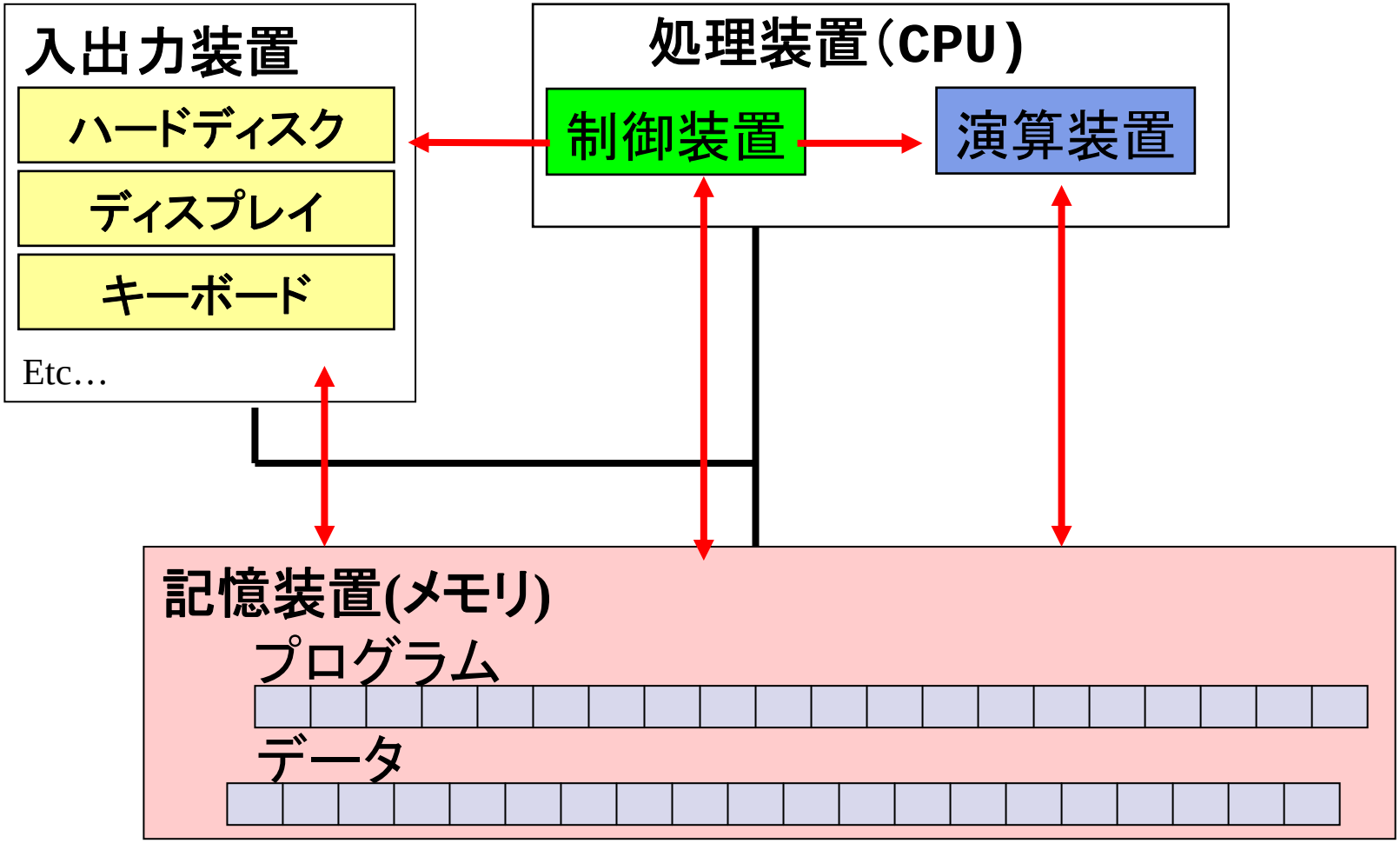
ハードウェアとソフトウェア



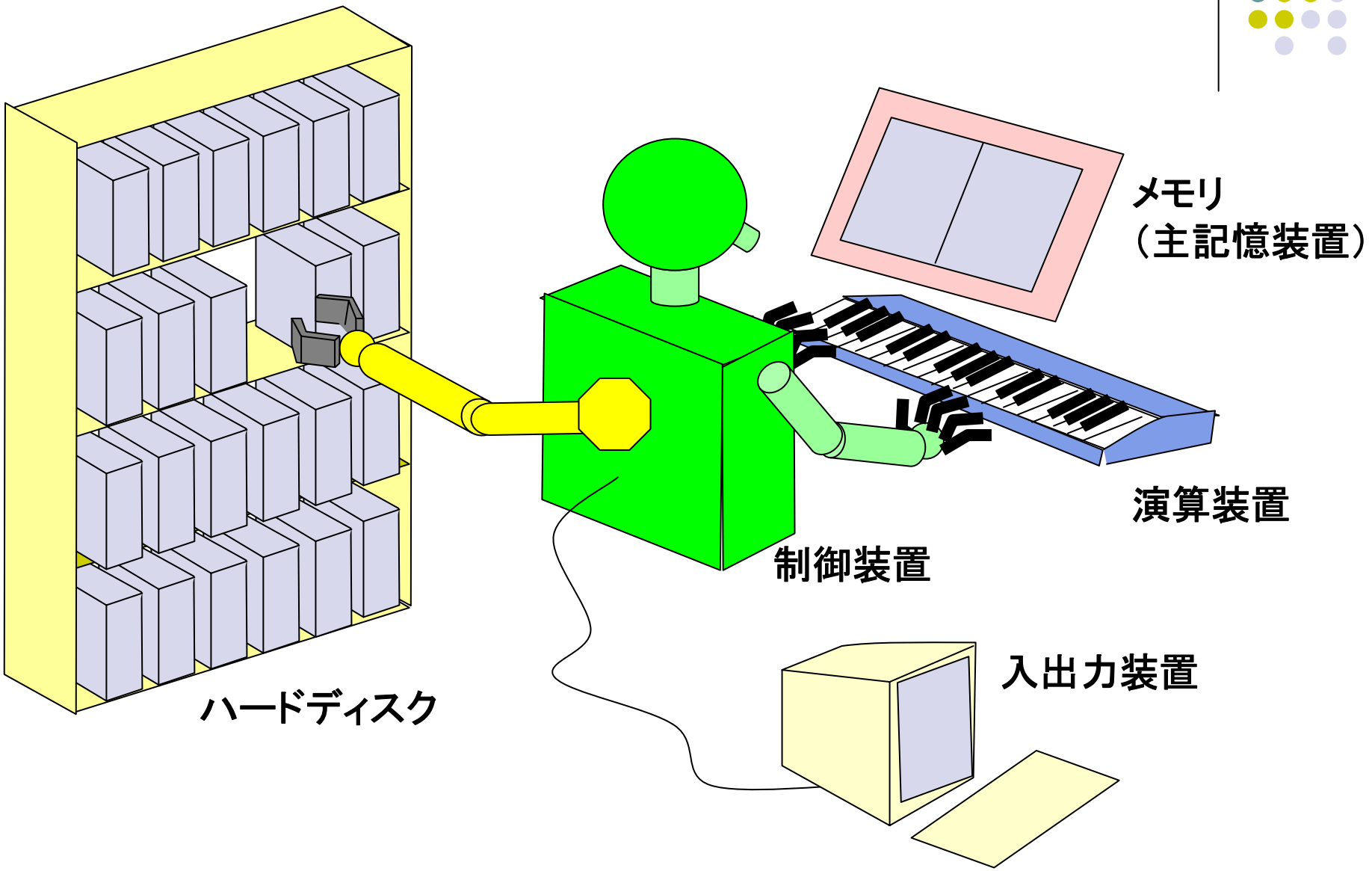
- ハードウェア
  - 計算，記憶等を行う機械
- ソフトウェア
  - ハードウェアに対する命令
  - データ

ソフトウェア（命令）がないとハードウェアは動かない

計算機の構造


入出力装置
ハードディスク
ディスプレイ
キーボード
Etc…
処理装置(CPU)
制御装置
演算装置
記憶装置(メモリ)
プログラム
データ

イメージとしては…



計算機内のデータ：　全て2進数

- ２進数
  ＝0と1だけで数値を表記する方法

2進数	10進数
0	0
1	1
10	2
11	3
100	4

- 何故　２進数？
  - 計算機の動作に利用される
    電気，磁気，光による
    保存・伝達に適しているから
    - 例えば，電圧が高い＝１，低い＝０
  - 計算機の回路を単純にできるから

- ２進数だけで計算や文字操作や画像編集ができる？

2進数の計算

- 和

  0+0=0　　0+1=1

  1+0=1　　1+1=10

- 積

  0×0=0　　0×1=0

  1×0=0　　1×1=1

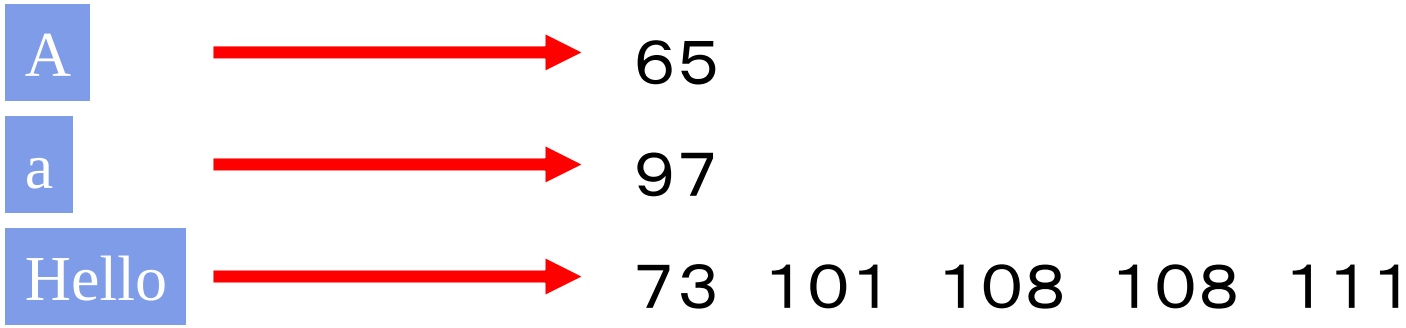
計算回路の実装が簡単

計算例:

```
  101      111        010        111
 +100     +010       ×110       ×110
-----    ----       ----       ----
 1001     1001       000        000
                     010        111
                    010        111
                   ------     -----
                    01100     101010
```

数字以外のデータも数値化

- 文字：　それぞれの文字に対応する番号を使用



- 画像：　点毎に赤・緑・青の明るさを数値化
- 音声：　一定間隔の周波数毎に強さを数値化

結局，すべてのデータが2進数で扱われる

プログラムも数値化

- 例：　変数 a と b の和を計算
  - Fortran：
    a + b
  - 機械語：
    - aの格納場所から値を取り出し，計算スペース1へ
    - bの格納場所から値を取り出し，計算スペース2へ
    - 計算スペース1の値に計算スペース2の値を加算

機械語と命令コード

- 機械語の作成には
  ハードウェアの命令コード表を利用

命令コード	動作
1	加算
2	積算
3	値の取り出し
4	値の格納

- aの格納場所(番地100)から値を取り出し，
  計算スペース1へ

3	100	1

- bの格納場所(番地200)から値を取り出し，
  計算スペース2へ

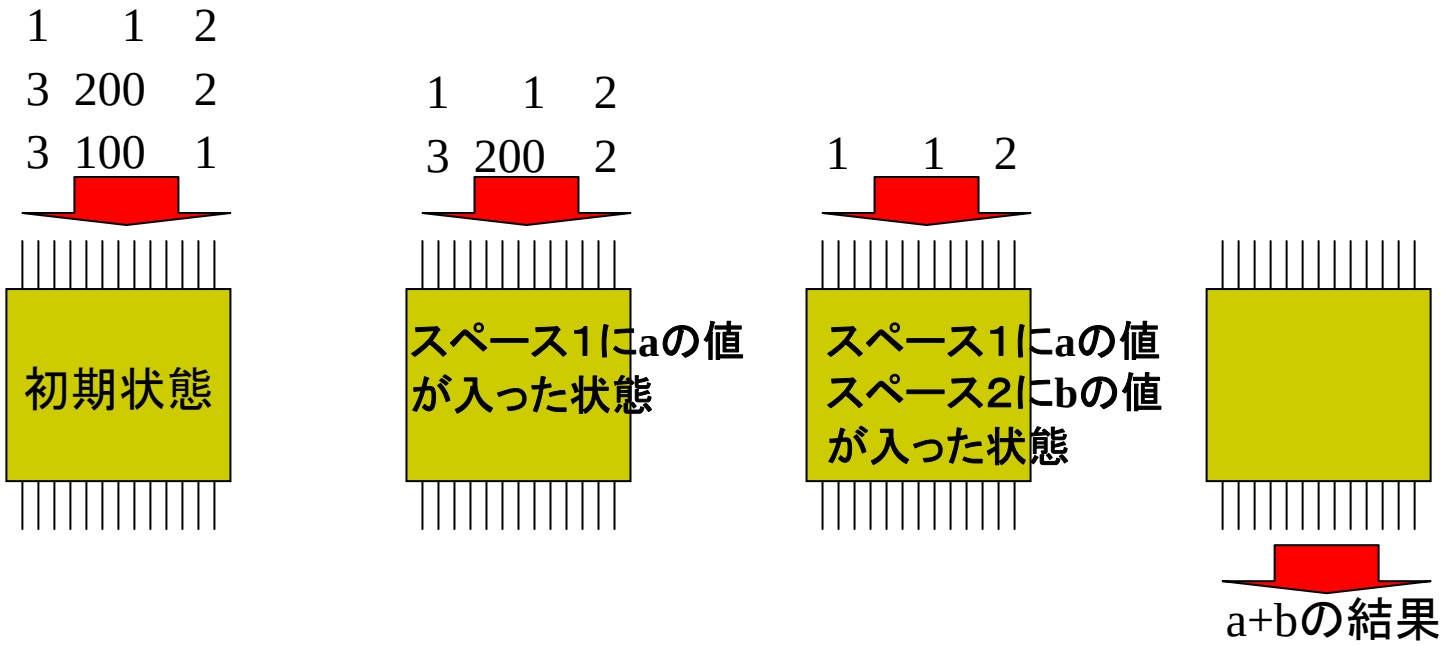
3	100	2

- 計算スペース1の値に
  計算スペース2の値を加算

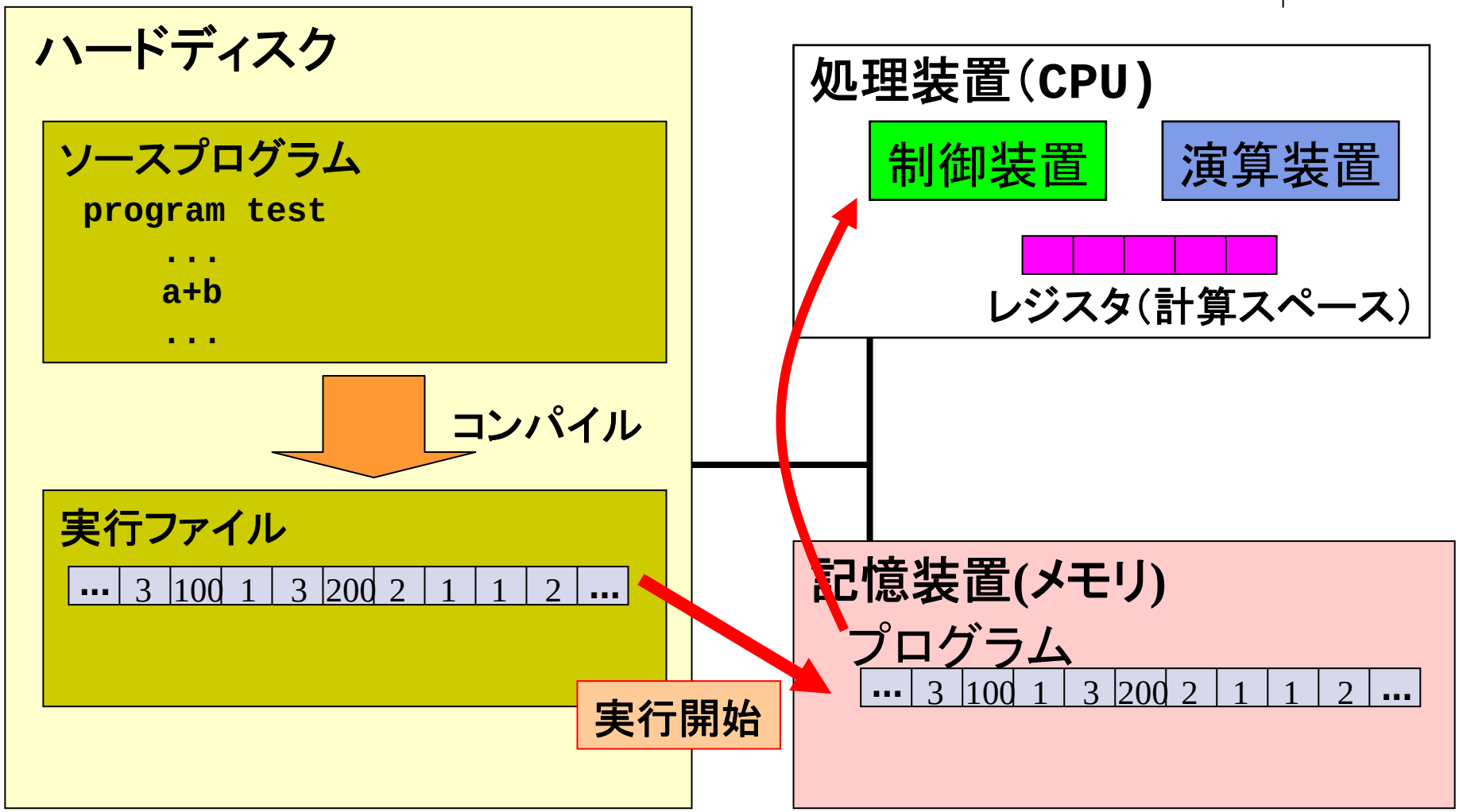
1	1	2

2進数の処理

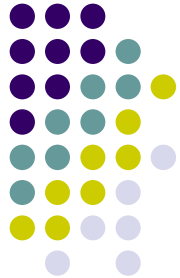
- 電圧の高低で1と0を伝達
- 入力信号(=プログラム，データ)に応じて
  プロセッサ内部の電圧状態を変更したり，
  信号を出力したりして計算を進める．



プログラムのコンパイルから実行まで



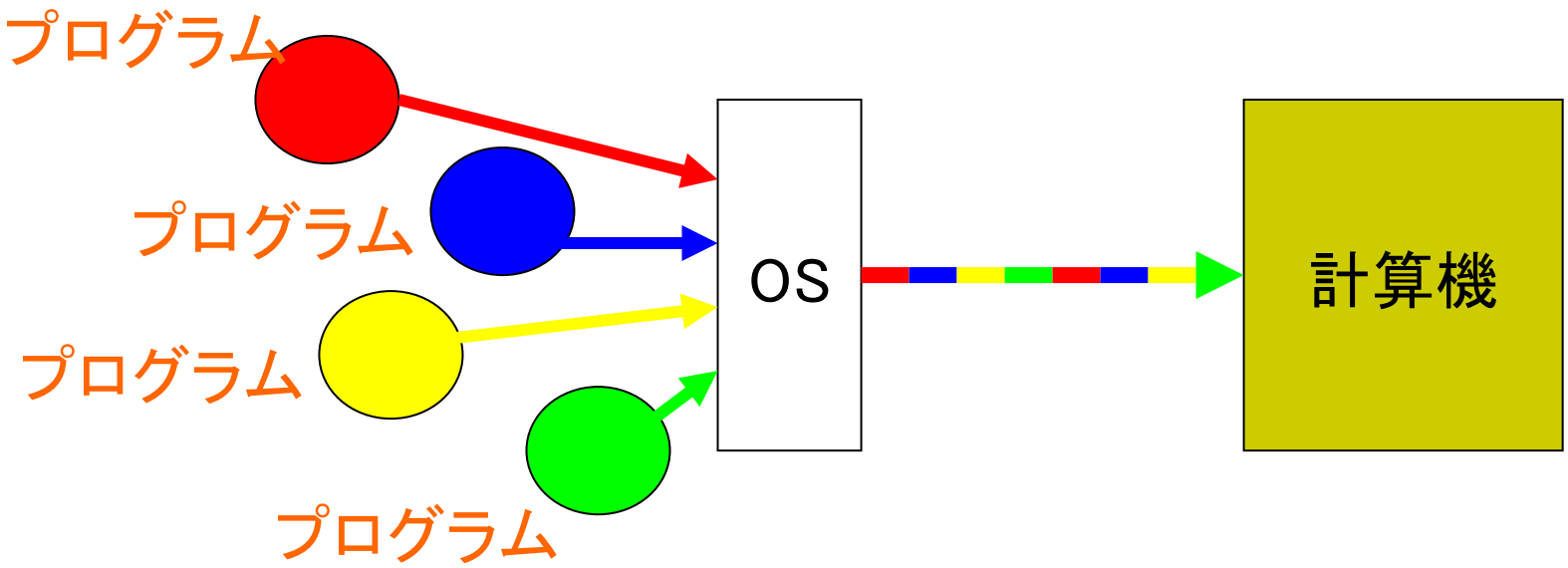
CPUの性能



- 「動作クロック周波数」と「CPUの名前」でなんとなく分かる
  - 動作クロック周波数
    - 計算機内部の状態変化の周期
      → 大きいほど速い
    - 単位：MHz（メガヘルツ） 1，000，000Hz
      GHz（ギガヘルツ） 1，000MHz
  - CPUの名前
    Pentium4 > Athlon, Pentium III > Celeron, Duron
    - 命令の実行方法や計算スペースの大きさ等に違い
- 計算機全体の性能には，他の装置の要素（メモリの容量や種類等）も大きく影響

OS(オペレーティングシステム)

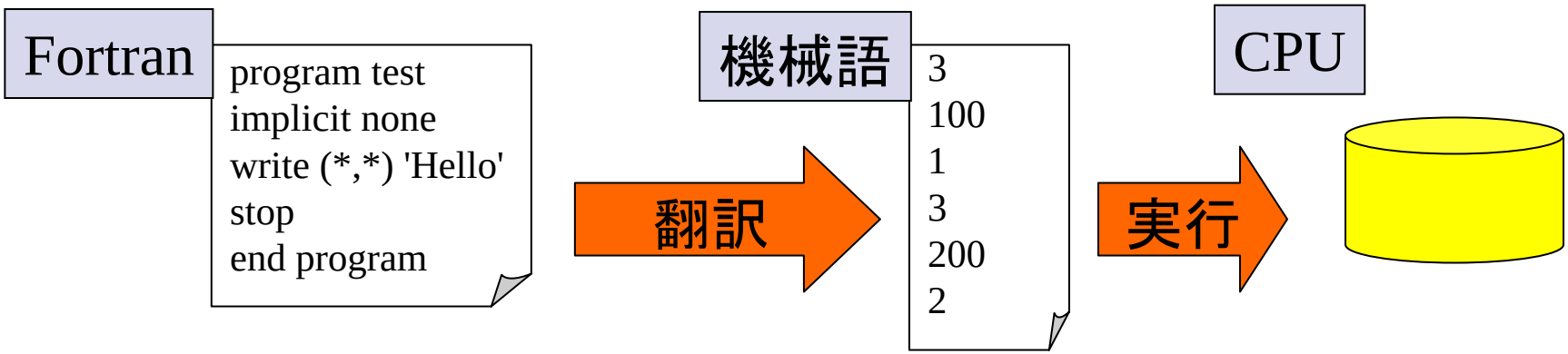
- 全てのプログラムが効率よく実行できるよう資源の割り当てを調整する
  - 複数のプログラムを「見かけ上」同時に実行
    - 実は　1/1，000秒程度の間隔でプログラムを切り替え



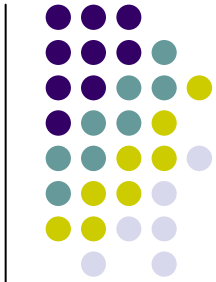
プログラミング言語とコンパイラ

- プログラミング言語は人間にとって読み書きが容易なプログラムの記述方法
  → 計算機は直接実行できない

- そこで，プログラミング言語を機械語に翻訳するソフトウェア
  ＝ コンパイラを利用.

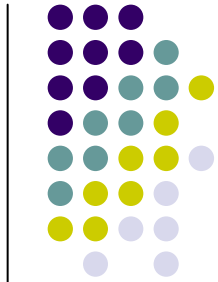


Fortran言語



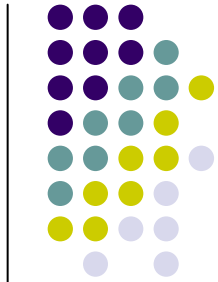
- 最も古いプログラミング言語の一つ
  - 1957年に最初のコンパイラ開発
- 特徴:
  - 覚えることが比較的少ない
  - 過去のプログラムが豊富に蓄積
  - 特に数値計算に適した機能が充実
- Fortranの歴史
  - Fortran II(1958)
    → Fortran IV(1962) 後の Fortran66
    → Fortran 77(1977)
    → Fortran90(1991)
    → Fortran95(1998) → Fortran2000

Fortran 77 と Fortran90



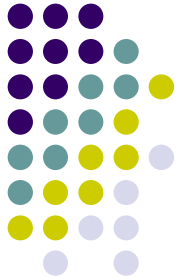
- 文法に大きな変更
  - 主に書き易さと実行効率の向上が目的
  - しかし研究室等には Fortran 77 で書かれたプログラムがまだたくさん残っている
    （現在利用可能なほとんどのコンパイラは
    両方サポート）
- 本講義では Fortran90 を扱う
  - 今後の主流

計算機のプログラムとは



- 仕事の手順書
  - 計算機で処理できるように記述したもの
- 記述方法に関する規則＝文法
- この講義では，
  - 問題に応じて計算機用の解決手順（アルゴリズム）を考え
  - 文法に従ってプログラミング言語で記述

  するための知識を習得する．



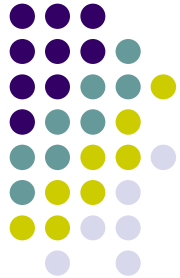
- 計算機に関する基礎知識

○ Fortranプログラムの基本構造

- 文字や数値を画面に表示する

- コンパイル時のエラーへの対処

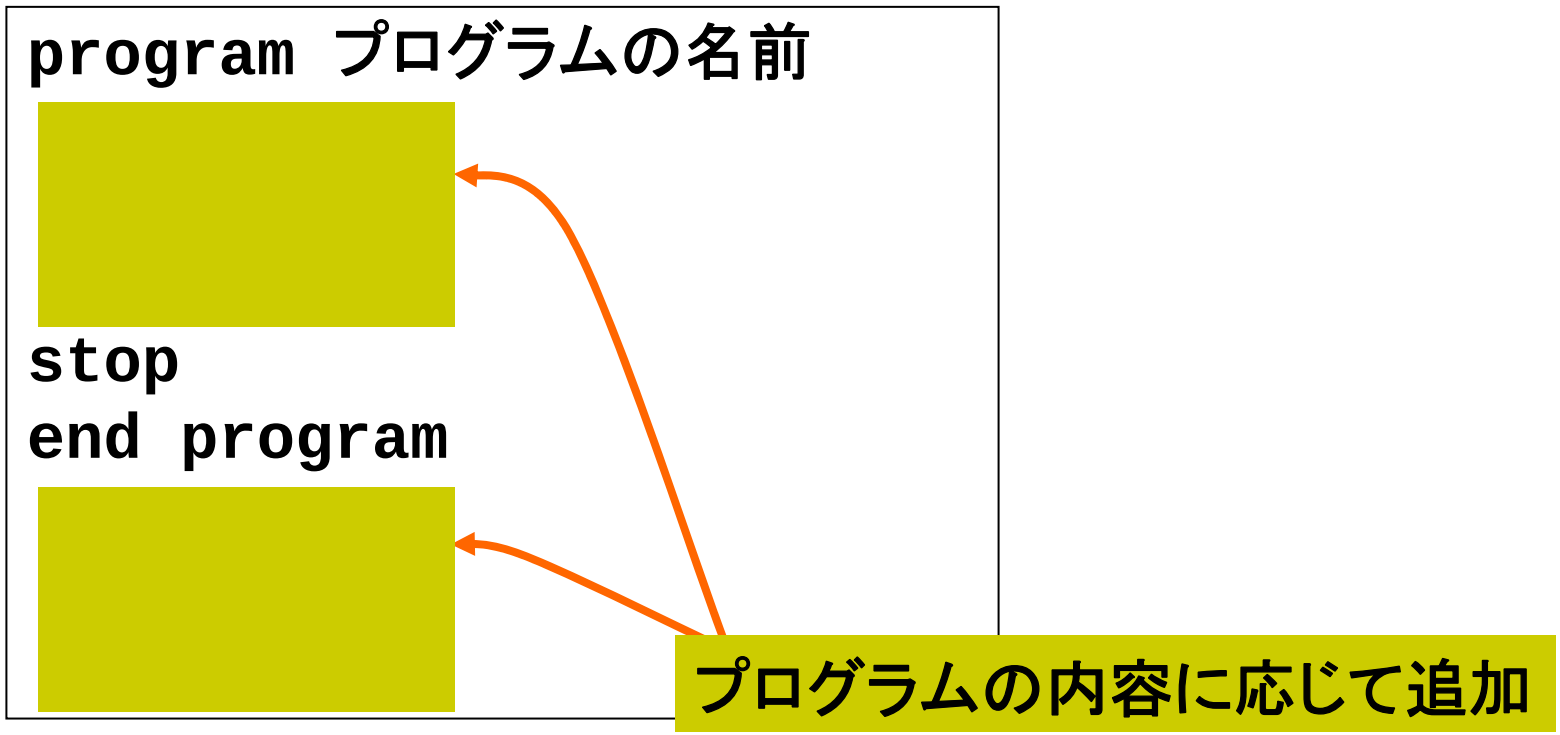
Fortran90のプログラム例



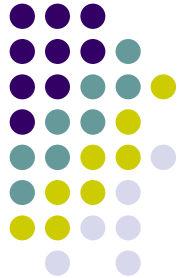
```
program hello
  write(*, *) 'Hello, Fortran.'
stop
end program
```

Fortran90プログラムの基本的な形

- どんなプログラムにも必ず書く3行



program と end program



- 主プログラムの開始と終了を指定
  - 主プログラム：　プログラムの主体

  ⇔ 副プログラム：　プログラムの中で呼び出される
  関数やサブルーチン（ずっと後で説明）
  - どんなプログラムでも必ず主プログラムの最初から
  実行が開始される

- プログラムの名前に関する決まり：
  - 使用できる文字は英数字と _ だけ
  - 空白を入れない
  - 先頭は英字
  - 長さは　３１文字以内

stop 文

- 実行を停止

- end program 直前以外でも
  必要に応じて記述可能

プログラムの書き方

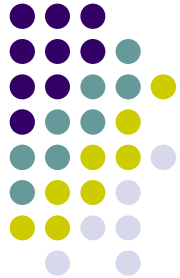
- 大文字と小文字:
  - Fortranでは大文字と小文字は区別しない
    - ただし，全角と半角は区別するので注意.
      - 例） ． と . は別の記号.
      - 全角文字を使うのは画面に出力する文字くらい

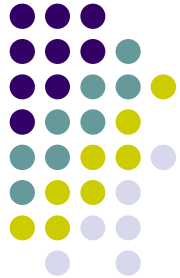
```
write(*, *)  'こんにちは'
```

- 空白:
  - 1個でも複数個でも働きは同じ
    - 見やすさを考えて調整
    - 命令や変数の名前を分けないよう注意
      （pro　gram，writ　e　等）.
- 改行
  - 1行でも複数行でも働きは同じ
    - 見やすさを考えて調整
      (通常，プログラムの意味的なまとまり毎に空行)

空白、空行の例

```
program kuku
implicit none
integer :: i,j

  do i=1,9
    write(*, '(i4,":"$)') i
    do j=1,9
      write(*, '(1x, i4$)') i * j
    end do
    write(*, *)
  end do

stop
end program
```





- 計算機に関する基礎知識
- Fortranプログラムの基本構造
- ○ 文字や数値を画面に表示する
- コンパイル時のエラーへの対処

write文

- 文字列や数値の表示

  `write(*,*) 表示内容`

- 文字列: ' または " で囲む
  - ' や " を表示したいとき:

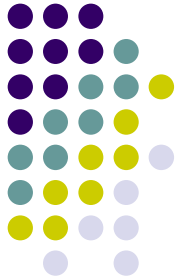
```
write(*, *) "I'm a student."
write(*, *) 'She said, "Hello." '
```

    - もしくは, 直前に ¥ を書いても良い.

```
write(*, *) 'I¥'m a student.'
```

- 数種類の内容を表示したい場合は , で区切る.

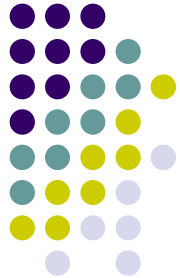
```
write(*, *) 'Total = ', 1000, ' yen.'
```



- 計算機に関する基礎知識
- Fortranプログラムの基本構造
- 文字や数値を画面に表示する

○　コンパイル時のエラーへの対処

コンパイル時のメッセージの意味



- 正常終了　→　何も表示されない

```
% f90 hello2.f90
%
```

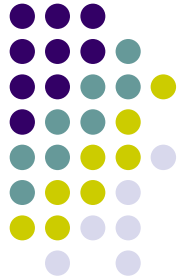
- 異常終了　→　以下のようなメッセージ表示

```
% f90 hello2.f90
Fortran diagnostic messages: program name(test)
  jwd1003i-s  "hello2.f90", line 2: 文字定数が途中で終わっています.
```

```
% f90 hello3.f90
Fortran diagnostic messages: program name(main)
  jwd1302i-s  "hello3.f90", line 1: この文は,FORTRANの文とはみなせません.
```

```
% f90 hello4.f90
Fortran diagnostic messages: program name(main)
  jwd1333i-s  "hello4.f90", line 1: 名前であるべきところが名前でありません.
```

デバッグ：プログラムの間違い修正



- エラーメッセージから間違いの場所や内容を推定

```
% f90 hello2.f90
Fortran diagnostic messages: program name(test)
  jwd1003i-s  "hello2.f90", line 2: 文字定数が途中で終わっています.
```

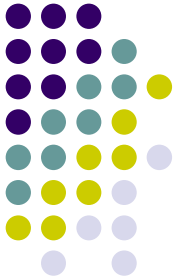
2行目

' が無い

```
program hello
  write(*, *) 'Hello, Fortran.
stop
end program
```

デバッグ

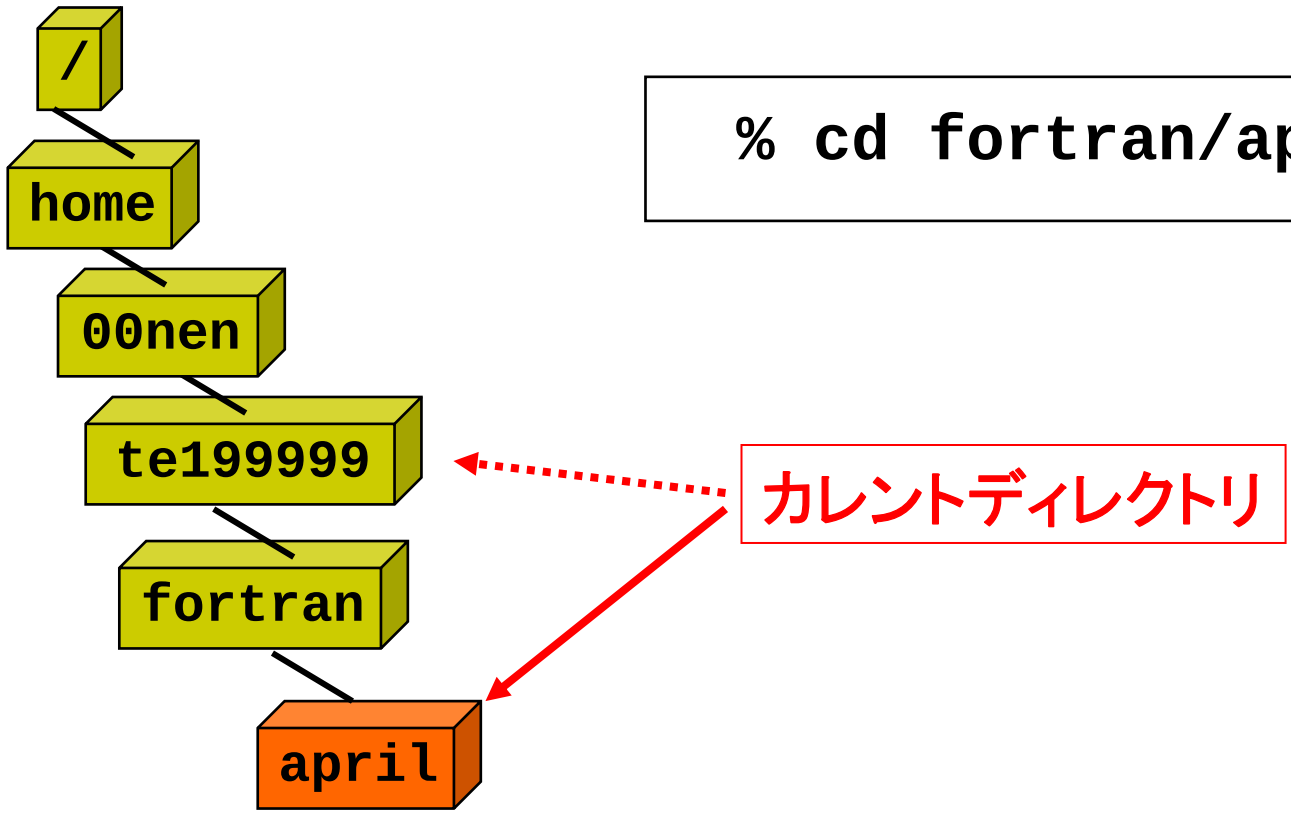
- エラーの内容
  - 文字定数が途中で終わっています
    → ' や ” の対応関係を調べる
  - 区切り記号が正しくありません
    → （ ） 等の対応関係を調べる
  - この文はFortranの文とはみなせません
    → 綴り違いを調べる
  - 名前であるべきところが名前でありません
    → プログラム名や変数名の先頭文字が
    数字になっていないか調べる



演習

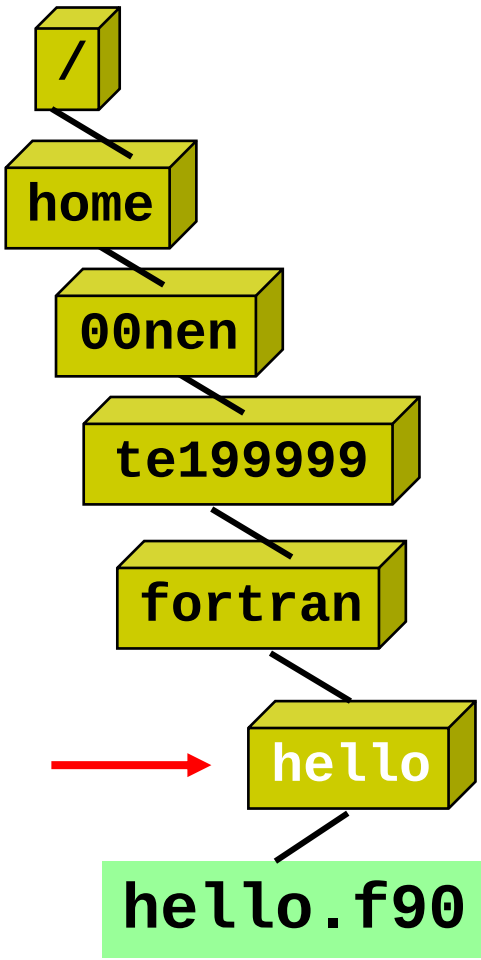
- 作業ディレクトリへの移動
- プログラムの入力
- コンパイル（デバッグ）
- 実行
- 応用：
  1. 表示文字列の変更
  2. 複数行の表示

作業対象ディレクトリへの移動　cd




```
% cd fortran/april
```

ファイルの編集　　emacs

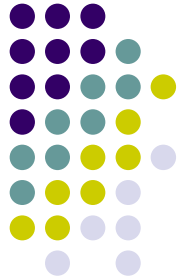



```
% ls
./      ../
% emacs hello.f90

プログラム編集

% ls
./      ../     hello.f90
%
```

内容

```
program hello
  write(*, *) 'Hello, Fortran.'
stop
end program
```

プログラムのコンパイル/実行



コンパイル

```
% f90 hello.f90 -o hello
```

間違いを修正したら
再コンパイル.

エラーメッセージが表示された？

YES

再編集

```
% emacs hello.f90
```

NO

実行

```
% ./hello
```

結果がおかしい、もしくはエラーメッセージが表示された？

YES

NO

Congratulations !!

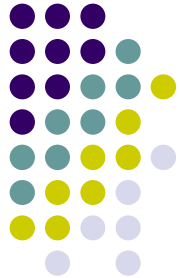
応用1： 表示文字列の変更

```
program hello
  write(*, *) 'All you need is love.'
stop
end program
```

文字列は全角（漢字やひらがな）でも可.

但し,他の部分では一切不可なので
半角／全角の切り替え忘れに注意.

応用2: 複数行の表示

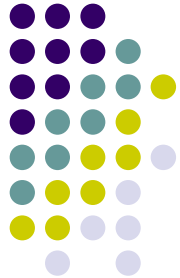


```
program hello
  write(*, *) 'Yesterday,'
  write(*, *) 'All my trouble'
  write(*, *) 'Seemed so far away.'
stop
end program
```

Emacsのコマンド（追加）

キー操作	意味
C-k	カーソルから右を切り取り
C-y	直前に切り取った部分を 貼り付け
C-a	行の先頭に移動
C-e	行の末尾に移動

切り取りと貼り付けを利用した行のコピー
注意:カーソルは行の先頭に置く

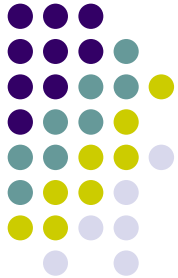


```
program hello
  write(*, *) 'All you need is love.'
  write(*, *) 'All you need is love.'
stop
end program
```

C-k　C-k　C-y　C-y

```
write(*, *) 'All you need is love.'
改行
```

連続行のコピー



- 連続行の切り取り:
  C-k を連続して押すと連続行を切り取り可

- C-k 以外の操作を行うと連続行の切り取り終了
  - C-y を押すとその時点で切り取られた連続行を全て貼り付け
  - C-k 以外の操作を行った後再び　C-k を押すとそれまでの切り取り分はクリアされる

次回

- 計算
  - 四則演算
  - 数学関数
  - 計算結果の表示