コンパイラ

第1回 コンパイラの概要

http://www.info.kindai.ac.jp/compiler
E館3階E-331 内線5459
takasi-i@info.kindai.ac.jp

本科目の内容

- コンパイラ(compiler)とは何か
- コンパイラの構成
- コンパイラの作成方法
  - 字句解析
  - 構文解析
  - 制約検査
  - コード生成
  - 最適化

情報システムプロジェクトＩと連携

成績について

評価基準	
各週課題	30%
定期試験	70%

- 無届欠席禁止
  - やむを得ず欠席した場合は翌週までに連絡すること
  - 無届欠席が複数回ある場合は試験の点数に関わりなく不受となる

オンライン授業では GoogleClassroom から出席カードが提出されれば出席とします

2021年度

学年	コース	受講者数	合格	不可	不受	合格率
3	システム	72	67	2	3	97%
4	メディア	1	1	0	0	100%

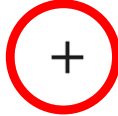
2022年度

学年	コース	受講者数	合格	不可	不受	合格率
3	システム	87	83	2	2	97%
3	メディア	1	1	0	0	100%
4	メディア	2	1	1	0	50%

※全出席し、全レポートを〆切までに提出して
不可になった受講生はいない

classroom.google.com

Google Classroom



ToDo　チェックが必要な課題　カレンダー

2023-基礎線形代数学...
情報学科情報学部1年生

2023-コンパイラ
理工学部情報学科情報システムコース...

2023-基礎微分積分学...
情報学部情報学科1年生

https://classroom.google.com/h

2023-情報システムプ...
理工学部情報学科情報システムコース...

プログラミング基礎1
2023年度

2023-社会情報学実習...

classroom.google.com

Google Classroom

クラスに参加

クラスを作成

ToDo　チェックが必要な課題　カレンダー

2023-基礎線形代数学...
情報学科情報学部1年生

2023-コンパイラ
理工学部情報学科情報システムコース...

2023-基礎微分積分学...
情報学部情報学科1年生

2023-情報システムプ...
理工学部情報学科情報システムコース...

プログラミング基礎1
2023年度

2023-社会情報学実習...

https://classroom.google.com/h

classroom.google.com

クラスに参加　　参加

現在、次のメールアドレスでログインしています

@info.kindai.ac.jp　　アカウントを切り替える

クラスコード

教師にクラスコードを聞いてこちらに入力してください。

クラスコード

クラスコードを使用してログインするには

- 承認済みアカウントを使用します

コンパイラ　zokgxoo

シスプロ1　cr7xwqo

classroom.google.com

Google Classroom

ToDo チェックが必要な課題 カレンダー

2023-基礎線形代数学...
情報学科情報学部1年生

2023-コンパイラ
理工学部情報学科情報システムコース...

2023-基礎微分積分学...
情報学部情報学科1年生

2023-情報システムプ...
理工学部情報学科情報システムコース...

プログラミング基礎1
2023年度

2023-社会情報学実習...

classroom.google.com

2023-コンパイラ
理工学部情報学科情報システムコース3年生

ストリーム 授業 メンバー 採点

2023-コンパイラ
理工学部情報学科情報システムコース3年生

カスタマイズ

Meet
リンクを生成

クラスコード
zokgxoo

期限間近
提出期限の近い課題はありません

クラスへの連絡事項を入力

石水隆 さんが新しい資料を投稿しました: Slack について
昨日

石水隆 さんが新しい資料を投稿しました: 第4回 講義資料
3月24日

石水隆 さんが新しい資料を投稿しました: 第3回 講義資料
3月24日

info.kindai.ac.jp

◆コンパイラ◆

このページは2023

https://www.info.kindai.ac.jp/compiler/

連絡

- 出席について

  単位取得には原則として全ての授業に出席する必要があります。 やむを得ず欠席する場合はその翌週までに必ず欠席届を出してください。 欠席届無しの欠席が複数回ある場合は履修の意思無しと見做して不受扱いにします。

  オンライン授業では、当日 GoogleClassrooom から出席カードが提出がされていれば出席扱いにします。

- 課題について

  単位取得には原則として全ての課題テストを受講する必要があります。 正当な理由のある場合を除いて原則として締切を過ぎた受講は認めません。

講義資料

- 第1回 コンパイラの概要 (4/12) パワーポイント PDF ノート用PDF (4/3 update)
- 第2回 形式言語と形式文法 (4/19) パワーポイント PDF ノート用PDF (4/3 update)
- 第3回 字句解析(1) (4/26) パワーポイント PDF ノート用PDF (4/3 update)
- 第4回 字句解析(2) (5/10) パワーポイント PDF ノート用PDF (4/3 update)

課題テスト

classroom.google.com

2023-コンパイラ
理工学部情報学科情報システムコース3年生

ストリーム　授業　メンバー　採点

＋ 作成

Google カレンダー　クラスのドライブ フォルダ

すべてのトピック
第4回：字句解析(2)
第3回：字句解析(1)
第2回：形式言語と...
第1回：コンパイラ...
Slack について

第4回：字句解析(2)

第4回 講義資料　投稿日: 3月24日

第4回 課題　各週課題　*下書き*

第3回：字句解析(1)

第3回 講義資料　投稿日: 3月24日

第3回 課題　各週課題　*投稿予定: 4月20日 8:00*

第2回：形式言語と形式文法

classroom.google.com

2023-コンパイラ
理工学部情報学科情報システムコース3年生

ストリーム　授業　メンバー　採点

第2回：形式言語と形式文法

第2回 講義資料　投稿日: 3月23日

第2回 課題　各週課題　*投稿予定: 4月13日 8:00*

第1回：コンパイラの概要

第1回 講義資料　投稿日: 3月23日

第1回 課題　各週課題　期限: 4月19日

Slack について

Slack について　投稿日: 昨日

classroom.google.com

2023-コンパイラ
理工学部情報学科情報システムコース3年生

ストリーム　授業　メンバー　採点

第2回 課題　各週課題　*投稿予定: 4月13日 8:00*

第1回：コンパイラの概要

第1回 講義資料　投稿日: 3月23日

Compiler01.pptx : パワーポイントファイル
Compiler01.pdf : pdf ファイル
Compiler01note.pdf : ノート用 pdf ファイル

Compiler01.pptx
PowerPoint

Compiler01.pdf
PDF

Compiler01note.pdf
PDF

資料を表示

第1回 課題　各週課題　期限: 4月19日

導入

Javaプログラムの実行

Hello.java

```
public class Hello {
    public static void main (String args[]) {
        System.out.print(“Hello! World!¥n”);
    }
}
```

```
$ javac Hello.java
$ java Hello
Hello! World!
```

Cプログラムの実行

Hello.c

```
#include <stdio.h>
int main () {
    printf (“Hello! World!¥n”);
}
```

```
$ gcc -o Hello Hello.c
$ Hello
Hello! World!
```

←これは？→

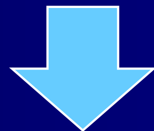
←実行→

実行の前にコンパイル(compile)を行う

機械語(machine language)

- 1,0 の並び
- 計算機で実行可能
- レジスタ, ビット操作が必要
- ハードウェアに依存

```
0001 0000 0101
0010 0000 1010
0000 1100 1110
0100 1111 0011
0101 0000 0001
```



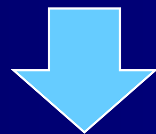
・プログラムの作成が困難
・プログラムの理解が困難
・プログラムのデバグが困難

人間が機械語を直接操作するのは効率が悪い

アセンブリ言語 (assembly language)

- 機械語命令を簡略名で記述
  - レジスタ, ビット操作が必要
  - ハードウェア依存
- 番地・レジスタ等に名前
- 実行は機械語変換が必要

```
A     DC  5
B     DC  10
START LD  GR0, A
      ADD GR0, B
      ST  GR0, A
```

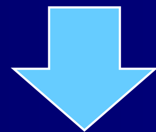


• 機械語よりはプログラムの
作成・理解・デバグが容易

しかしまだ人間がアセンブリ言語を
直接操作するのは効率が悪い

高水準言語 (high level language)

- 命令が基本的に英語
- ハードウェアに依存しない
- 変数名、メソッド名等を付けられる
- メソッド、関数等を定義できる
  - C, Java 等



・人間にとって理解し易い

しかし計算機はそのままでは
高水準言語を理解できない

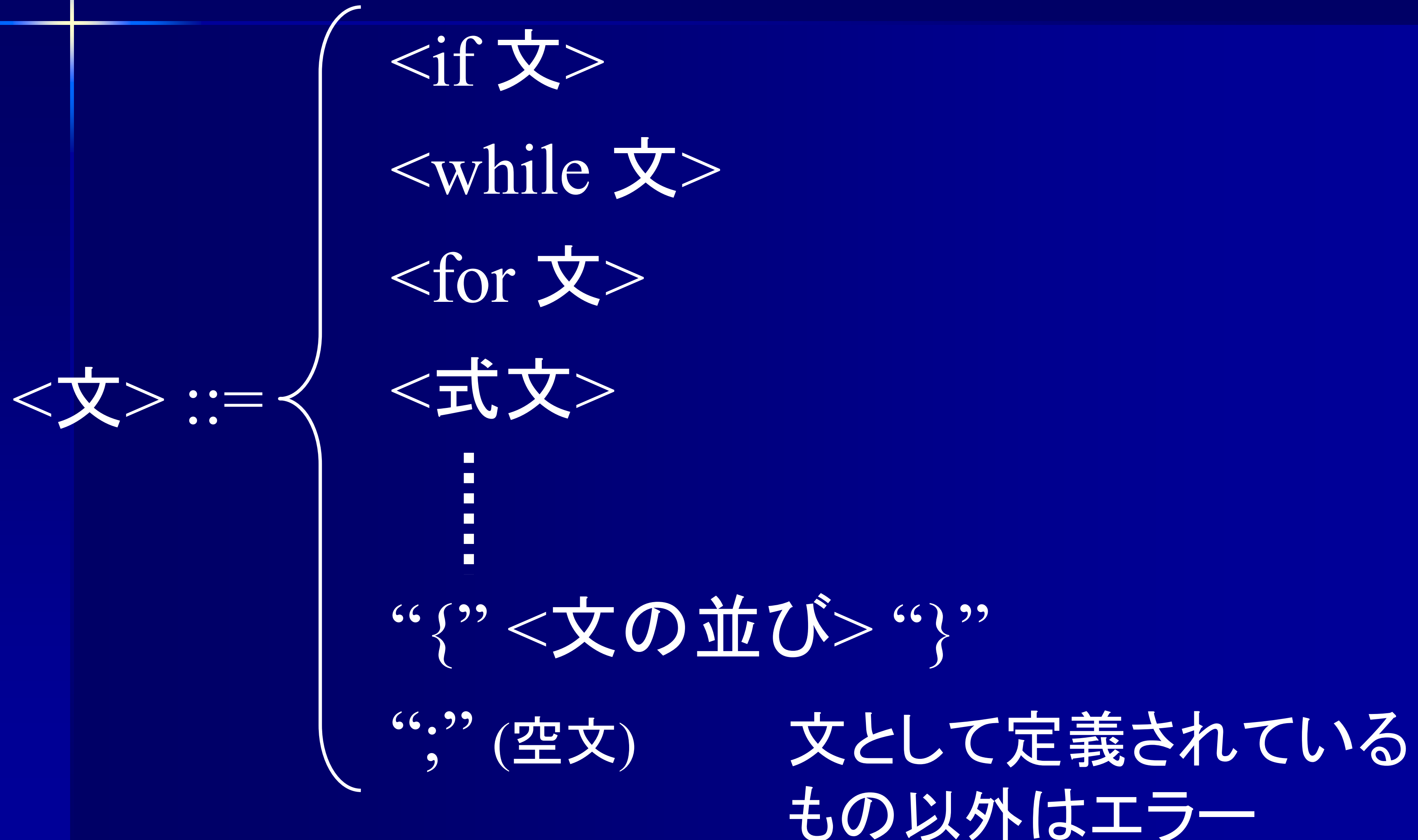
```
public class Sample {
  public static void main
        (String  args[])  {
      int n;
      int a[n] = new int[8];
      for (int i=0; i<n; ++i) {
        a[i] = i*2;
      }
      int x, y, z;
      if (x == 1) {
          System.out.
              print (y) :
      } else {
```

プログラミング言語の翻訳

- プログラミング言語は文法が明確
  ⇒計算機で“翻訳”可能

⇔自然言語は文法に曖昧性
  ⇒計算機での“翻訳”は難しい

プログラミング言語の文法



プログラミング言語の文法

<if文> ::= “if” “(” <式> “)” <文>

または

“if” “(” <式> “)” <文> “else” <文>

<式> ::= <項> “+” <項>

<項> ::= <因子> “*” <因子>

<因子> ::= { <整数> | <変数> | “(” <式> “)” }

- <整数>
- <変数>
- “(” <式> “)”

全て厳密に
定義されている

コンパイラ (compiler)

- コンパイラ
  - 原始プログラム(source program)を目的プログラム(object program)に変換(翻訳)するプログラム

原始プログラム (source program)

- 原始プログラム(source program)
  - 高水準言語(high level language)で記述
  - 人間がエディタで作成
  - そのままでは実行不可
  - C, Java 等

```
public class Sample {
   public static void main (String  args[])  {
      int n;
      int a[n] = new int[8];
      for (int i=0; i<n; ++i) {
         a[i] = i*2;
      }
      int x, y, z;
      if (x == 1) {
           System.out.print (y) :
      } else {
```

目的プログラム (object program)

- 目的プログラム(object program)
  - 低水準言語(low level language)で記述
    (高水準言語を出力するコンパイラもある)
  - 高水準言語からコンパイラが変換
  - 実行可能なプログラムもある
  - 機械語, アセンブリ言語

```
0  PUSHI 0
1  POP     5
2  PUSH 5
3  PUSH 1
4  COMP
5  BGE   20
6  JUMP 11
7  PUSHI 5
8  PUSH  5
9  INC
```

原始プログラムと目的プログラム

原始プログラム → コンパイラ → 目的プログラム

Hello.java → javac → Hello.class

```
public class Hello {
    public static void main (String args[]) {
        System.out.print(“Hello! World!¥n”);
    }
}
```

人間が読み書き可能

```
ﾊ・ｺｾ????2?
??        ???
?? ? ??<init>?()V?Code?
LineNumberTable?main?([Ljava/lang/String;)V?
SourceFile?
Hello.java? ? ????
Hello! World! ????Hello?java/lang/Object
?java/lang/System?out?Ljava/io/PrintStream;
?java/io/PrintStream? println
?(Ljava/lang/String;)V?!????????? ??
?         ???????*ｷ?ｱ????
????????  ?
???       ???%?????  ｲ?ｶ?ｱ????
???
????????
????
```

人間には理解不能

実行形式プログラム (executable program)

- 実行形式プログラム(executable program)
  - 実行可能なプログラム
  - 機械語で記述
  - 高水準言語からコンパイラが変換

```
$ gcc -o Hello Hello.c
$ Hello
Hello! World!
```

実行形式ファイル名を入力すれば実行可能

(注意)ファイル名入力で実行できるもの全てが実行形式プログラムではない

ライブラリ(library)

- 多くのプログラムに共通して使われる機能
  - 入出力関数, 数学関数(三角, 指数対数等)等

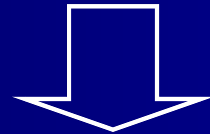
プログラム1	入出力関数
プログラム2	入出力関数
プログラム3	入出力関数

個別に
作るのは無駄

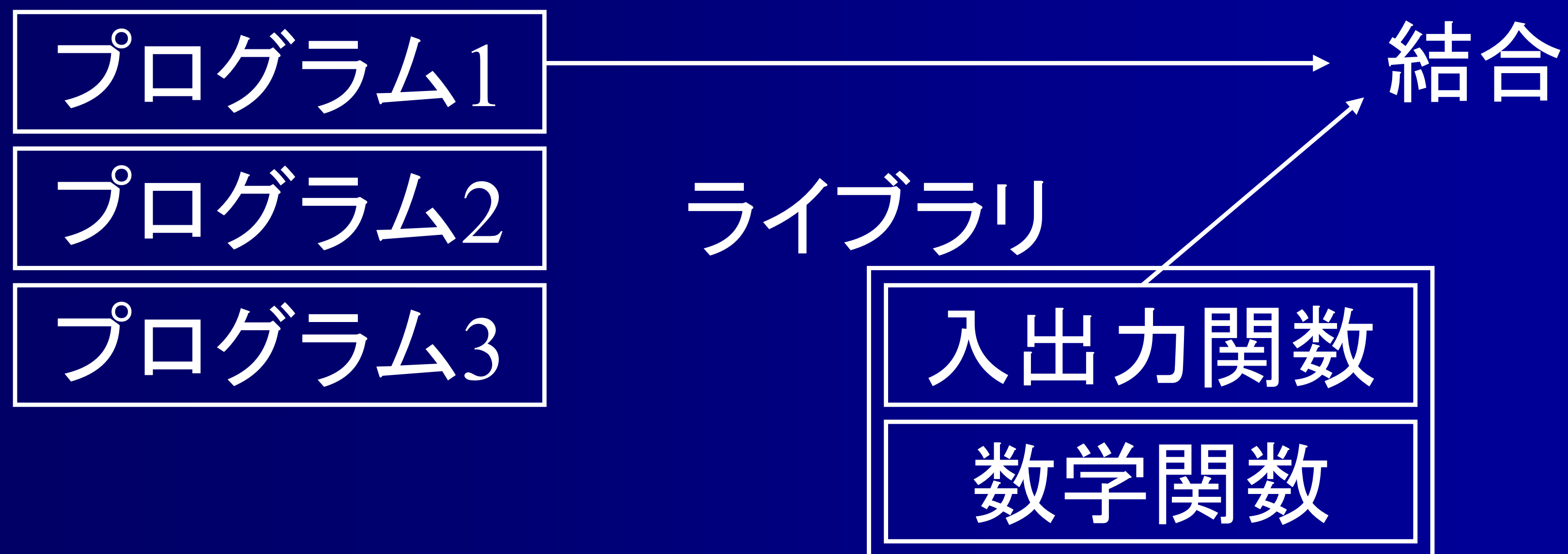
⇒予め作成しておけばいい

ライブラリ(library)

- 多くのプログラムに共通して使われる機能
  ＝プログラムごとに作成するのは無駄

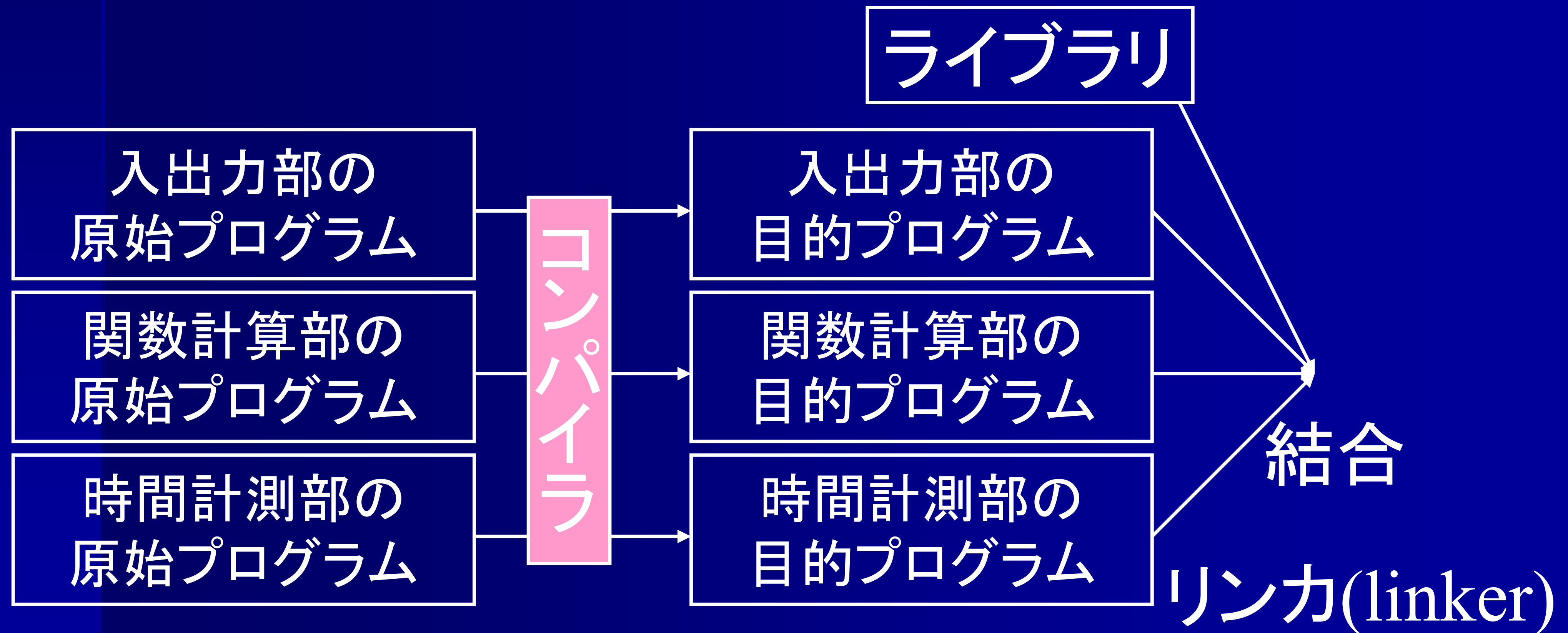


ライブラリ(library)を用いる



分割コンパイル (separate compile)

- 分割コンパイル(separate compile)
  – 原始プログラムをクラス、メソッドごとに分割
  – 各クラスごとにコンパイルする



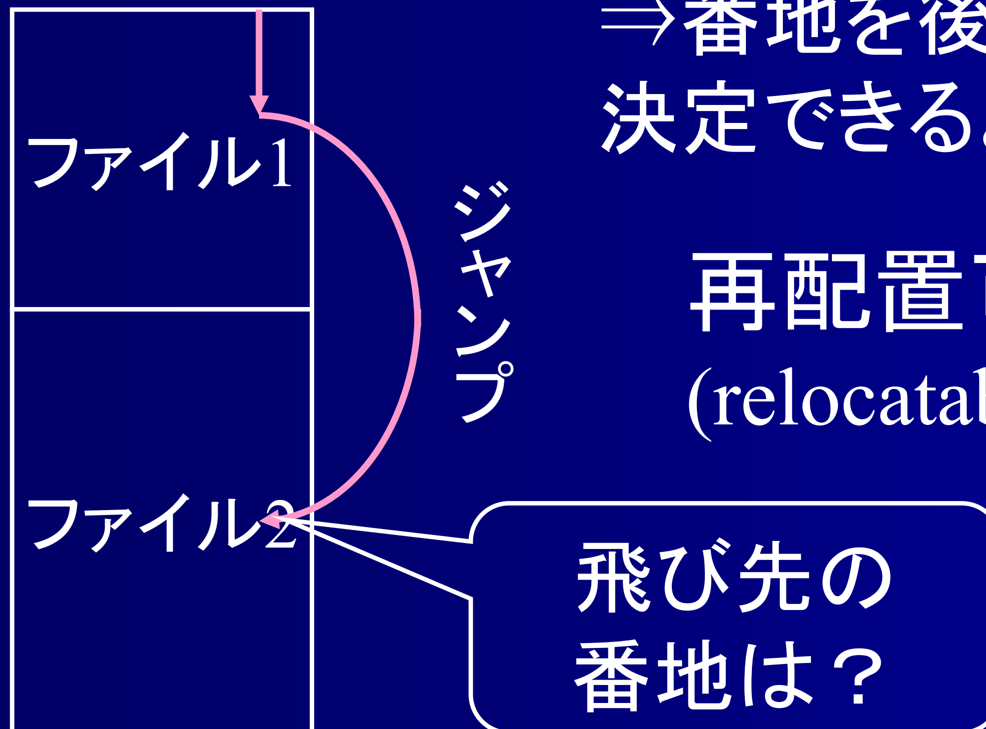
分割コンパイルの問題点

複数のファイルを別々にコンパイル

⇒他のファイルのサイズ、番地が分からない

⇒番地を後から
決定できるようにする

再配置可能プログラム
(relocatable program)



再配置可能プログラム (relocatable program)

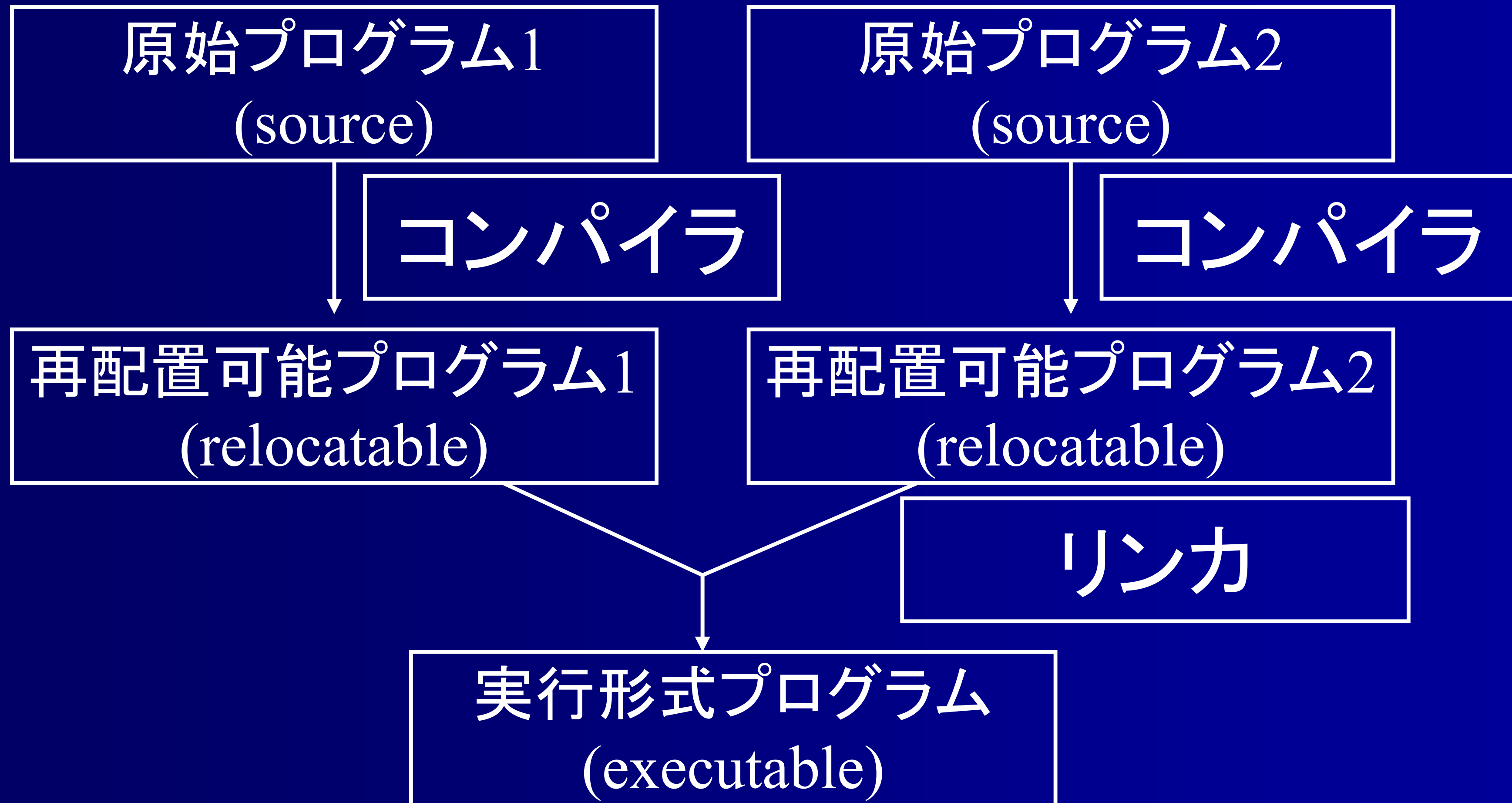
- 再配置可能プログラム
  - プログラム先頭を0番地として相対的に記述
  - 他のプログラムと結合時に番地を再計算

```
0 LOAD  1000
1 LOAD  L1:
2 ADD
3 BEQ       10
4 INPUT
5 STORE 1002
      :
```

先頭を0番地とした番地

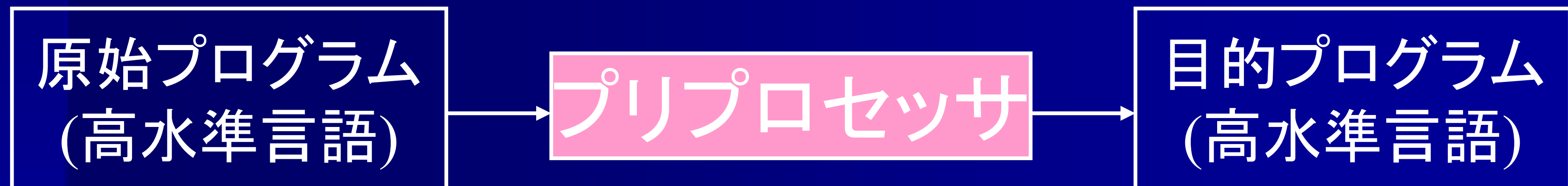
他のプログラムの番地には仮のラベル

分割コンパイル

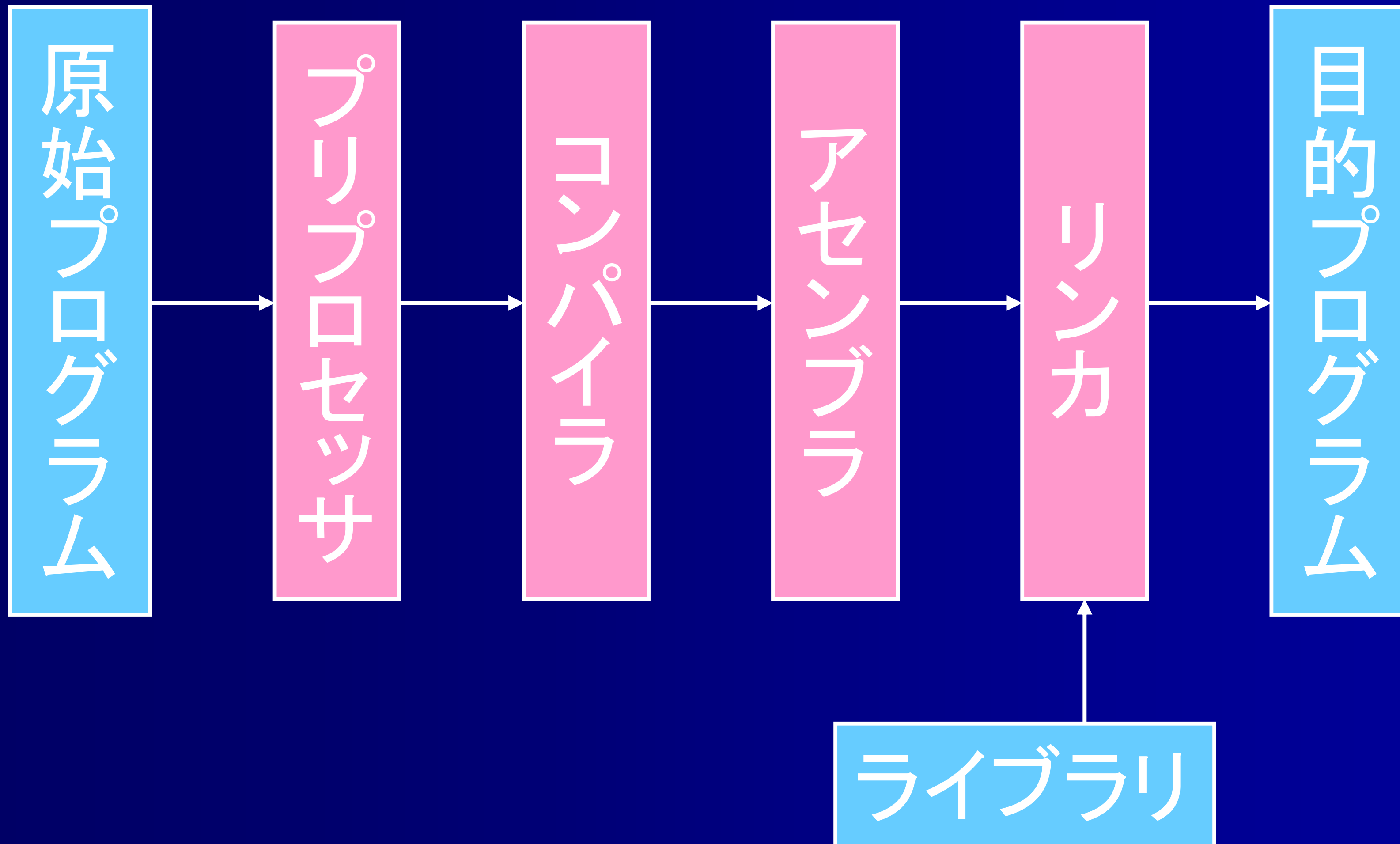


プリプロセッサ(preprocessor)

- プリプロセッサ
  - 目的プログラムが高水準言語のコンパイラ
  - コンパイラの前処理として行う

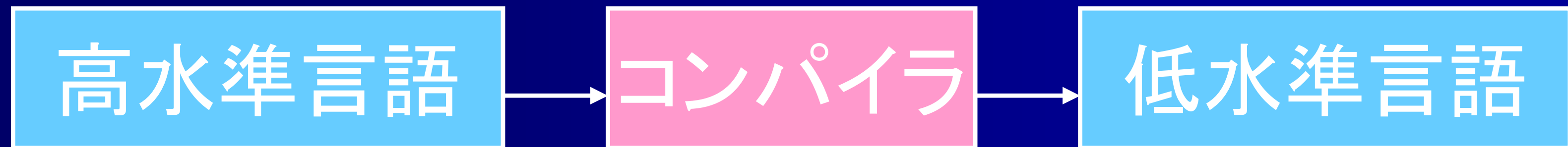


コンパイルシステム例



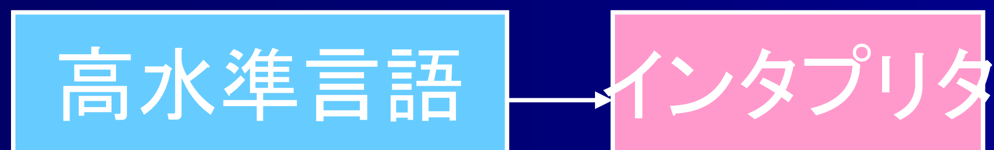
インタプリタ(interpreter)

- コンパイラ



実行

- インタプリタ(interpreter)



実行

高水準言語を解釈して処理

BASIC, perl, ruby 等

コンパイラとインタプリタ

- コンパイラ
  - 一旦コンパイルすれば高速で実行可能
    (インタプリタの数十～数百倍)

  ⇒繰り返し実行するときに有効
- インタプリタ
  - コンパイルすることなく実行可能

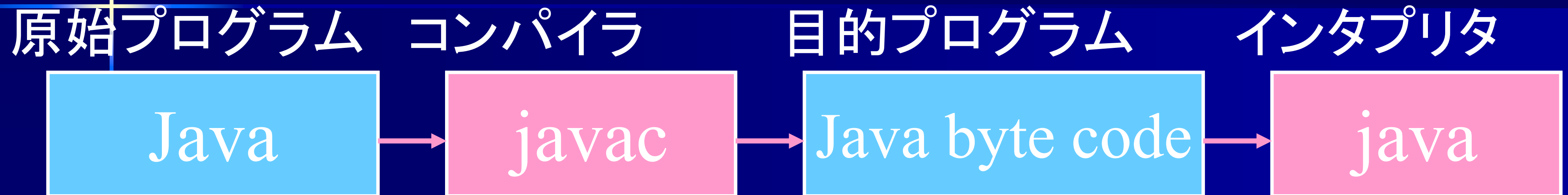
  ⇒1回だけ実行するときに有効

  ⇒作成→実行を繰り返すときに有効

コンパイラとインタプリタ

	コンパイラ	インタプリタ
処理	低水準言語に変換	そのまま実行
プログラム作成+実行	コンパイルが必要	そのまま実行可能
実行速度	速	遅
処理系の多機種への移植	難	易
作成し易さ	難	易

Javaの場合



Java byte code は
中間コード(intermidiate code)
実行形式ではない

```
$ javac Hello.java
$ java Hello
Hello! World!
```

インタプリタ “java” を使用

コンパイラ＋インタプリタ

コンパイラの記述言語

- コンパイラ
  - 原始プログラム(source program)を目的プログラム(object program)に変換(翻訳)するプログラム

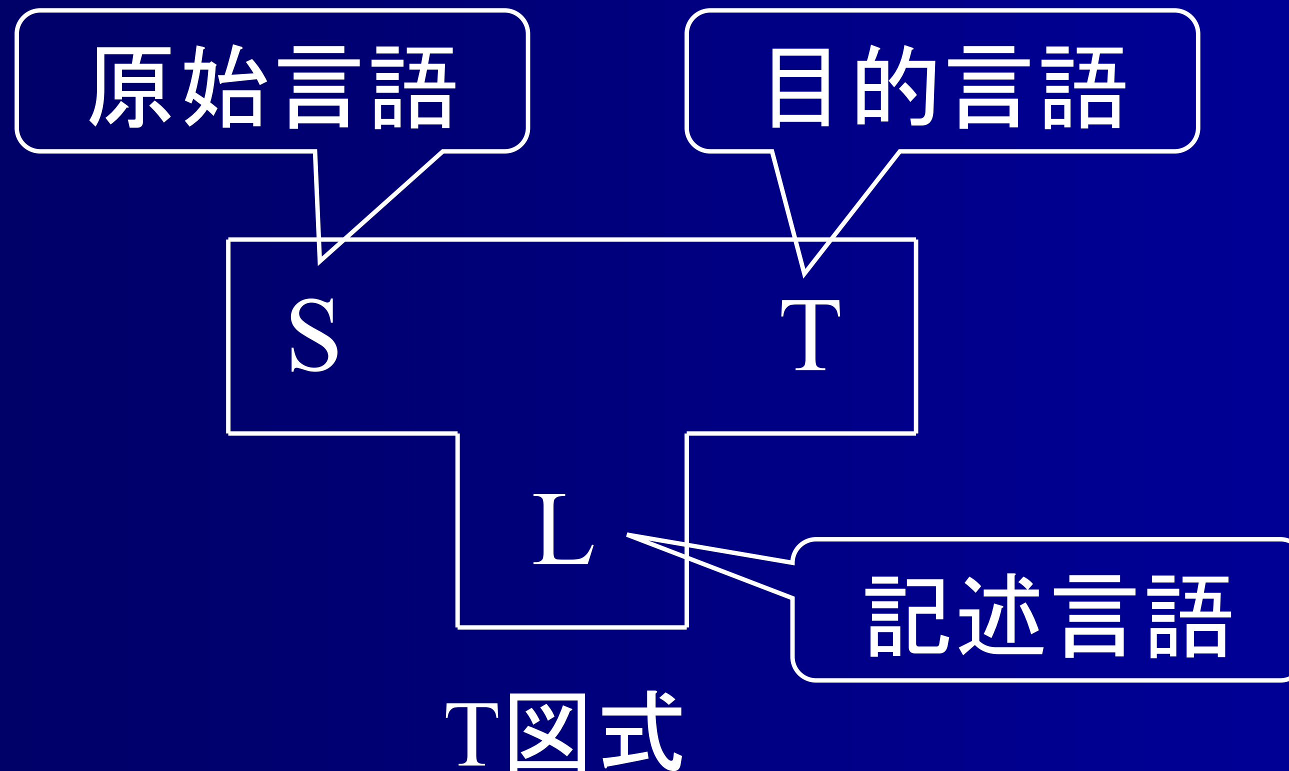
コンパイラもプログラム

その言語は？

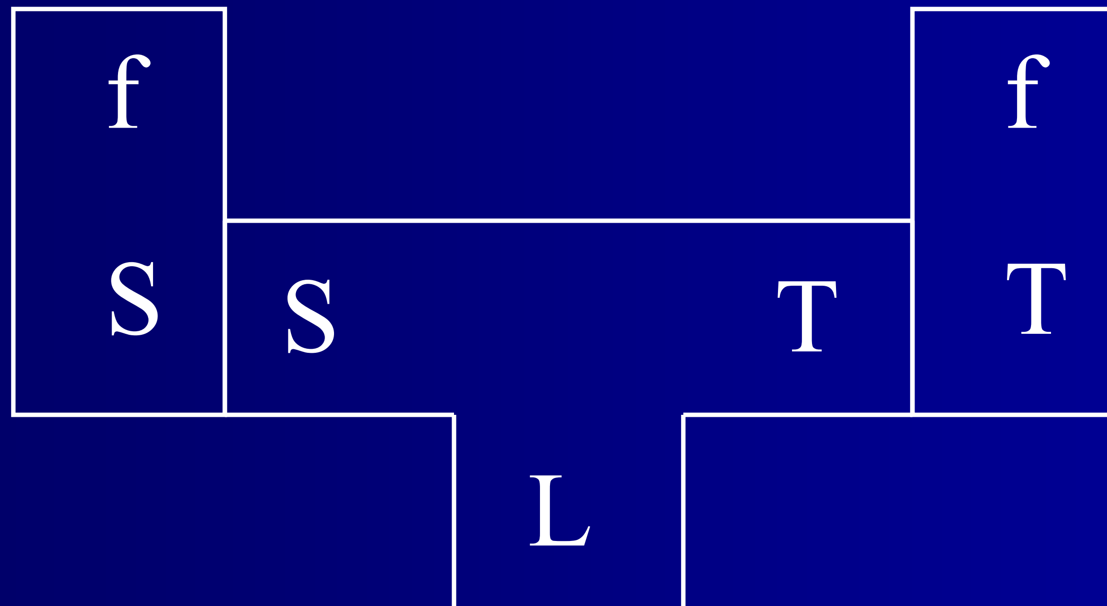
高水準言語？ 低水準言語？

T図式

- 原始言語Sを目的言語Tに変換する言語Lで記述されたコンパイラ



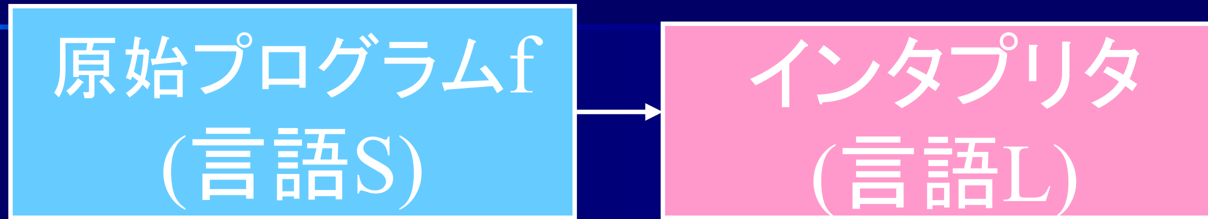
T図式

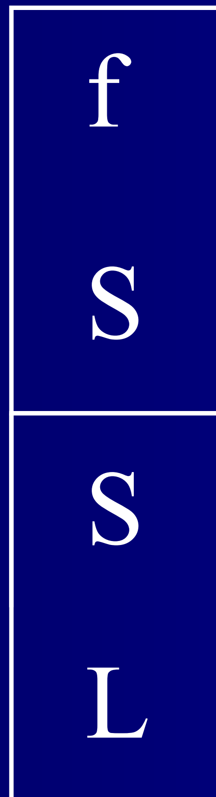


T図式

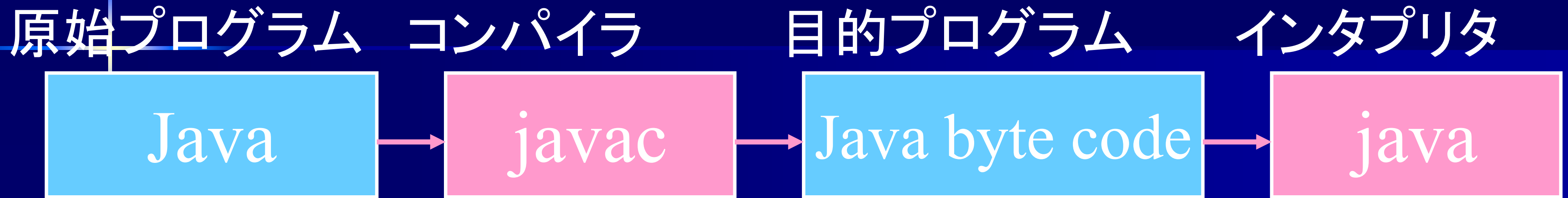
例：Java を JBC(Java byte code) に変換する
機械語 M で記述された javac コンパイラ

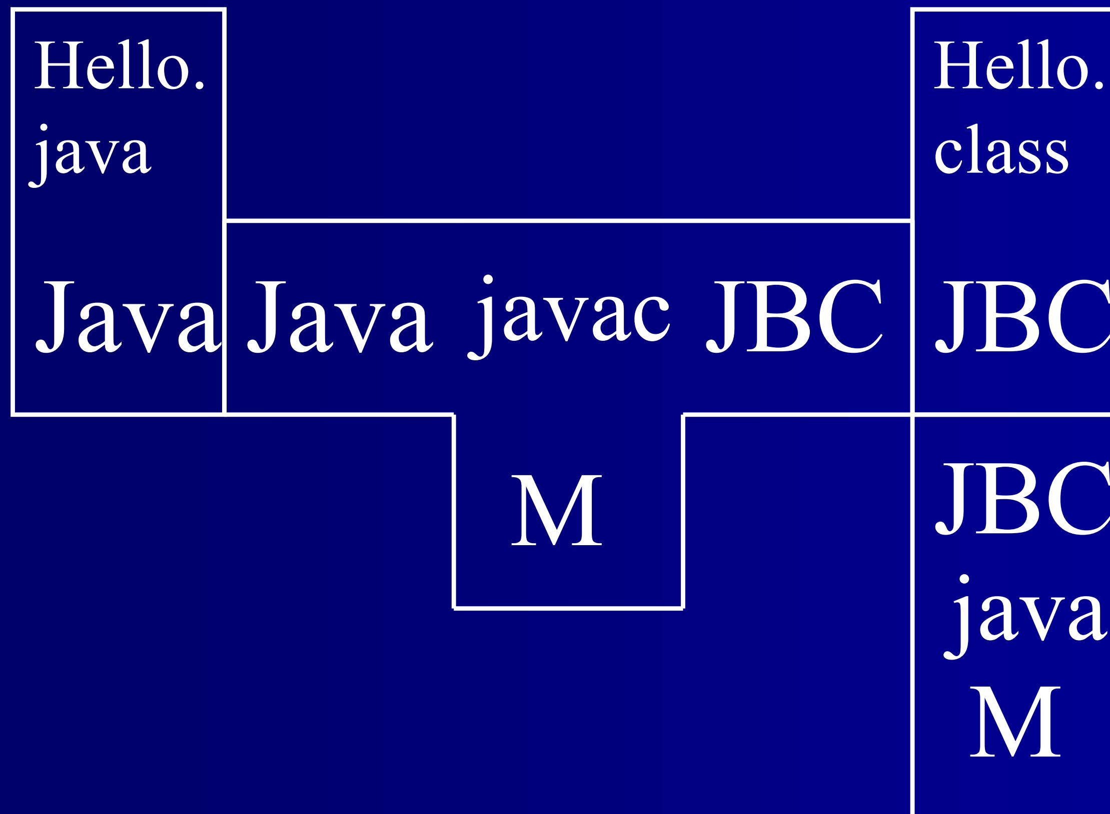
T図式(インタプリタの場合)



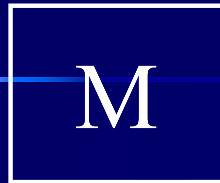


T図式(Javaの場合)





コンパイラの作成

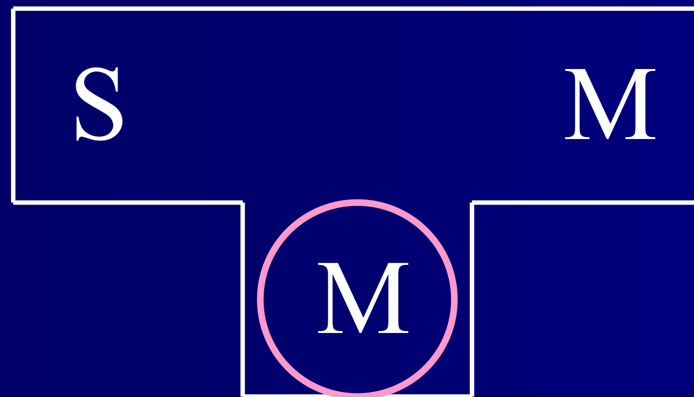


機械語 M のみ実行可能

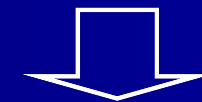
計算機 M

計算機 M 上で動く高水準言語 S のコンパイラが欲しい

必要なコンパイラ



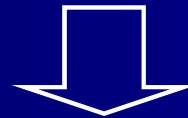
しかし機械語 M で
プログラムは難しい



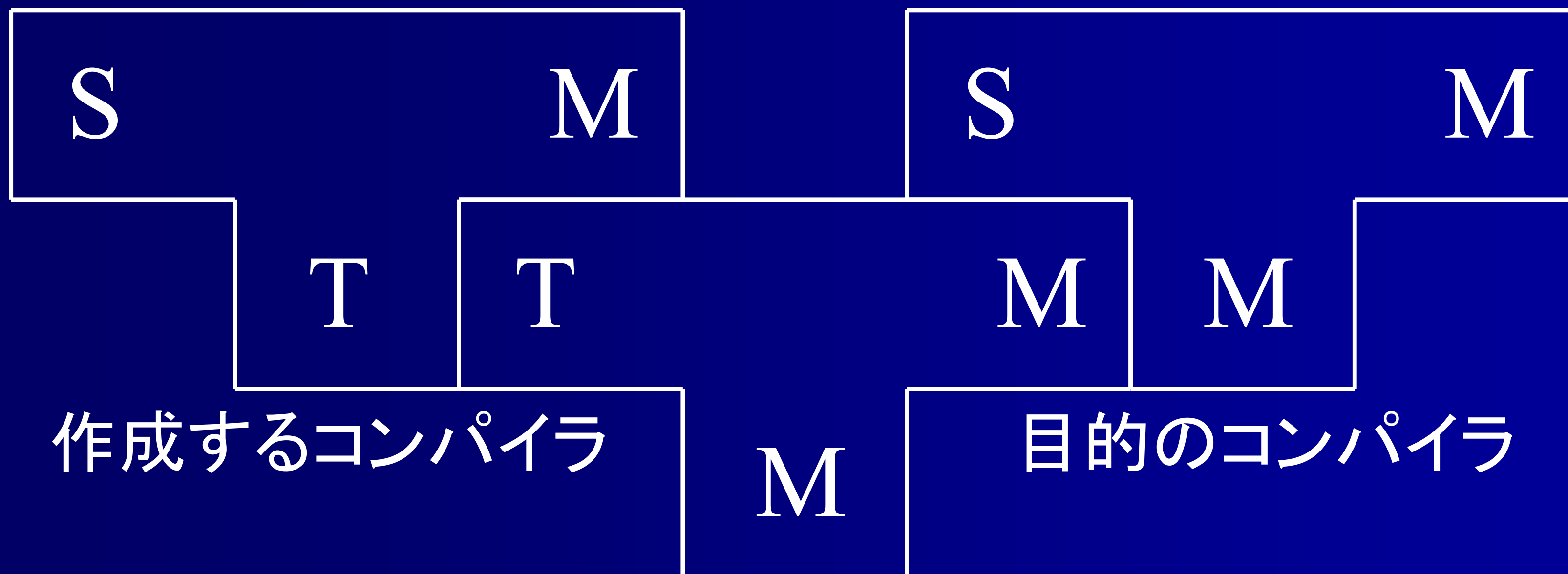
既存の高水準言語
コンパイラを利用

コンパイラの作成

計算機 M 上で動く高水準言語 S のコンパイラが欲しい



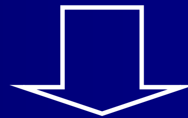
計算機 M 上で動く高水準言語 T のコンパイラを利用



コンパイラの作成は
高水準言語で行える

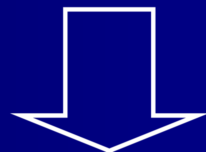
コンパイラの作成

計算機 M 上で動く高水準言語 S のコンパイラが欲しい



計算機 M 上で動く高水準言語 T のコンパイラを利用

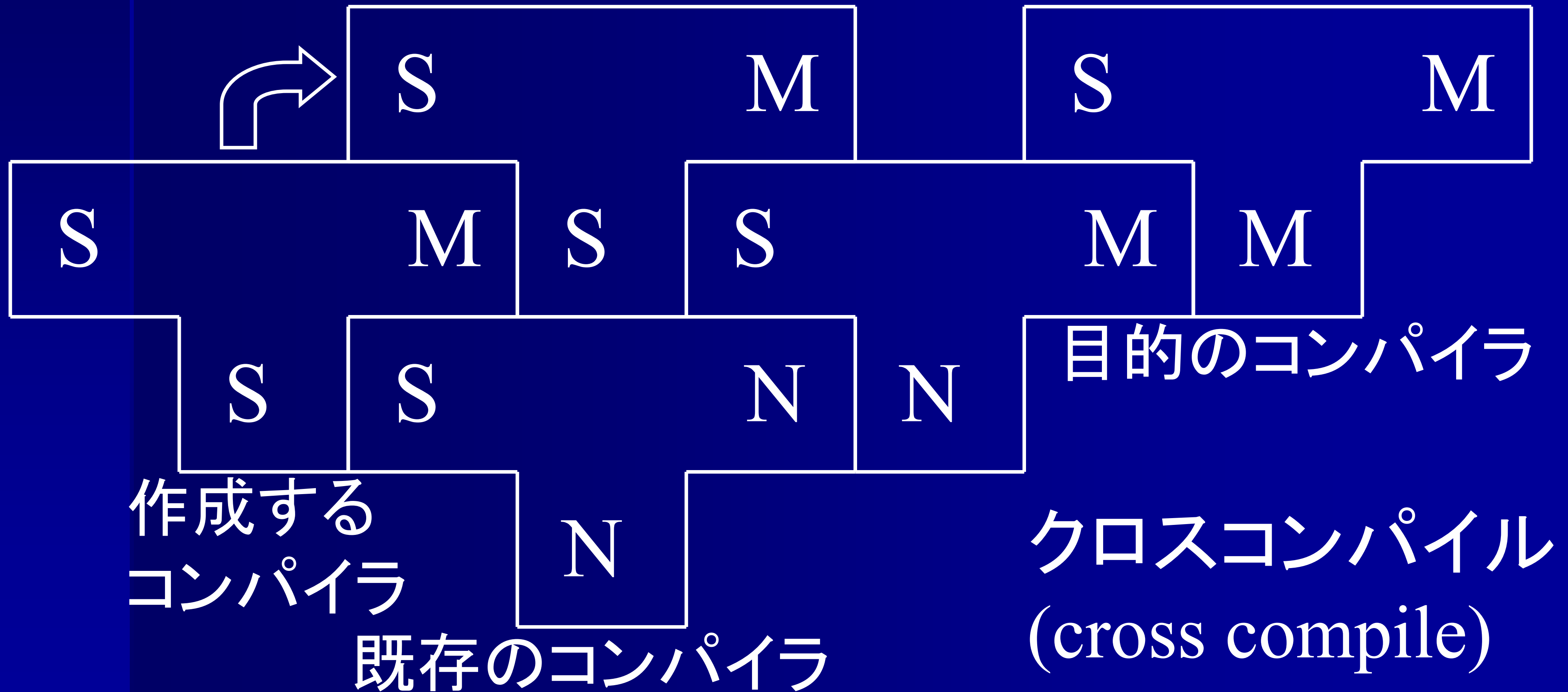
- ✓ ではTのコンパイラはどうやって作る？
- ✓ M上で動く既存の高水準言語コンパイラが無い場合は？



別の計算機 N 上で動くコンパイラを利用

コンパイラの作成

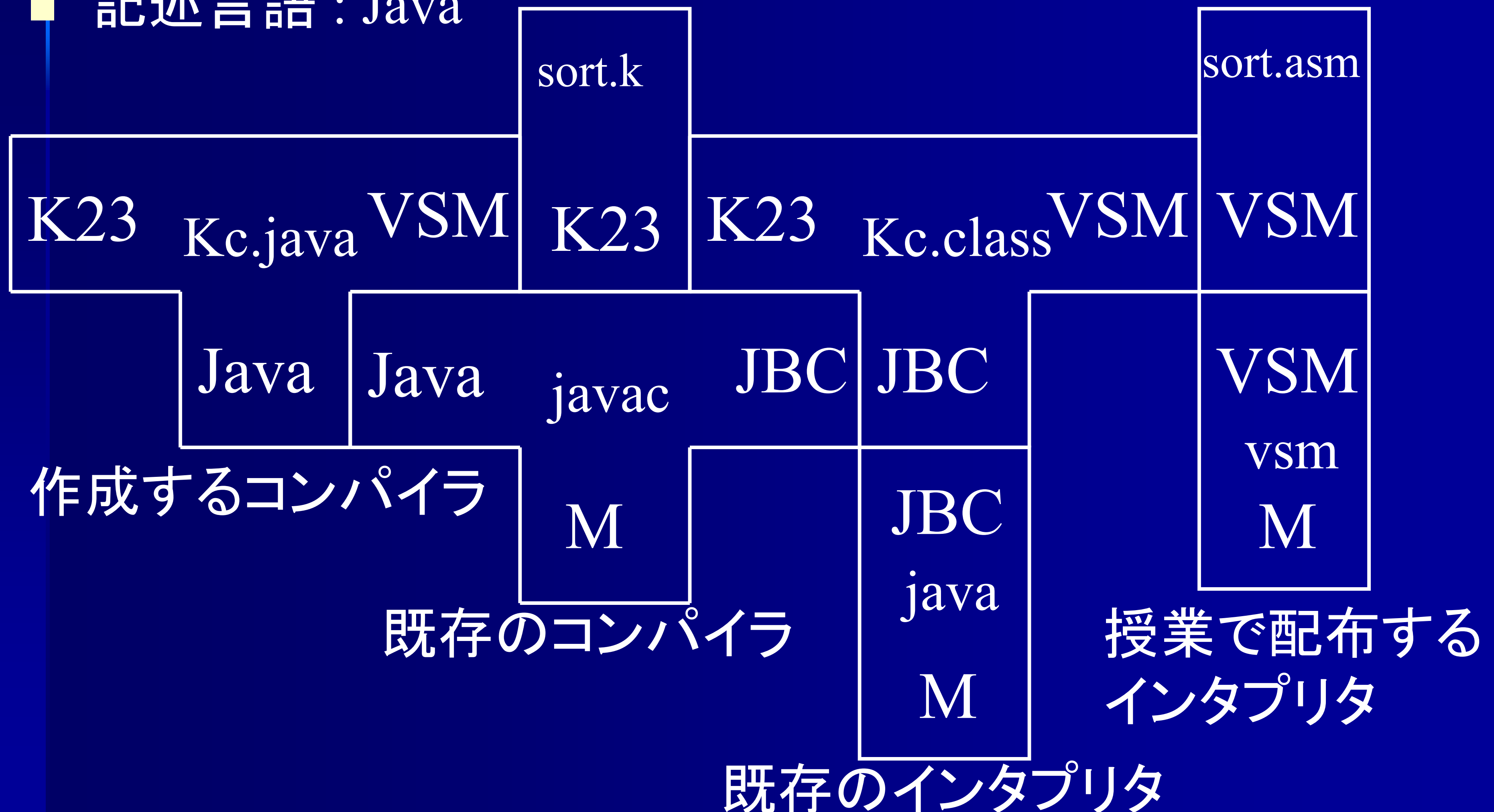
M 新しい計算機 M　　N 既存の計算機 N



クロスコンパイル
(cross compile)

情報システムプロジェクトIの場合

- 原始言語 : K23言語(C風言語)
- 目的言語 : VSM(Virtual Stack Machine)アセンブラ言語
- 記述言語 : Java



コンパイラの構造

- 字句解析系
- 構文解析系
- 制約検査系
- 中間コード生成系
- 最適化系
- 目的コード生成系

字句解析系 (lexical analyzer, scanner)

- 字句解析系
  - 空白、コメントを読み飛ばす
  - 単語(token)に区切る

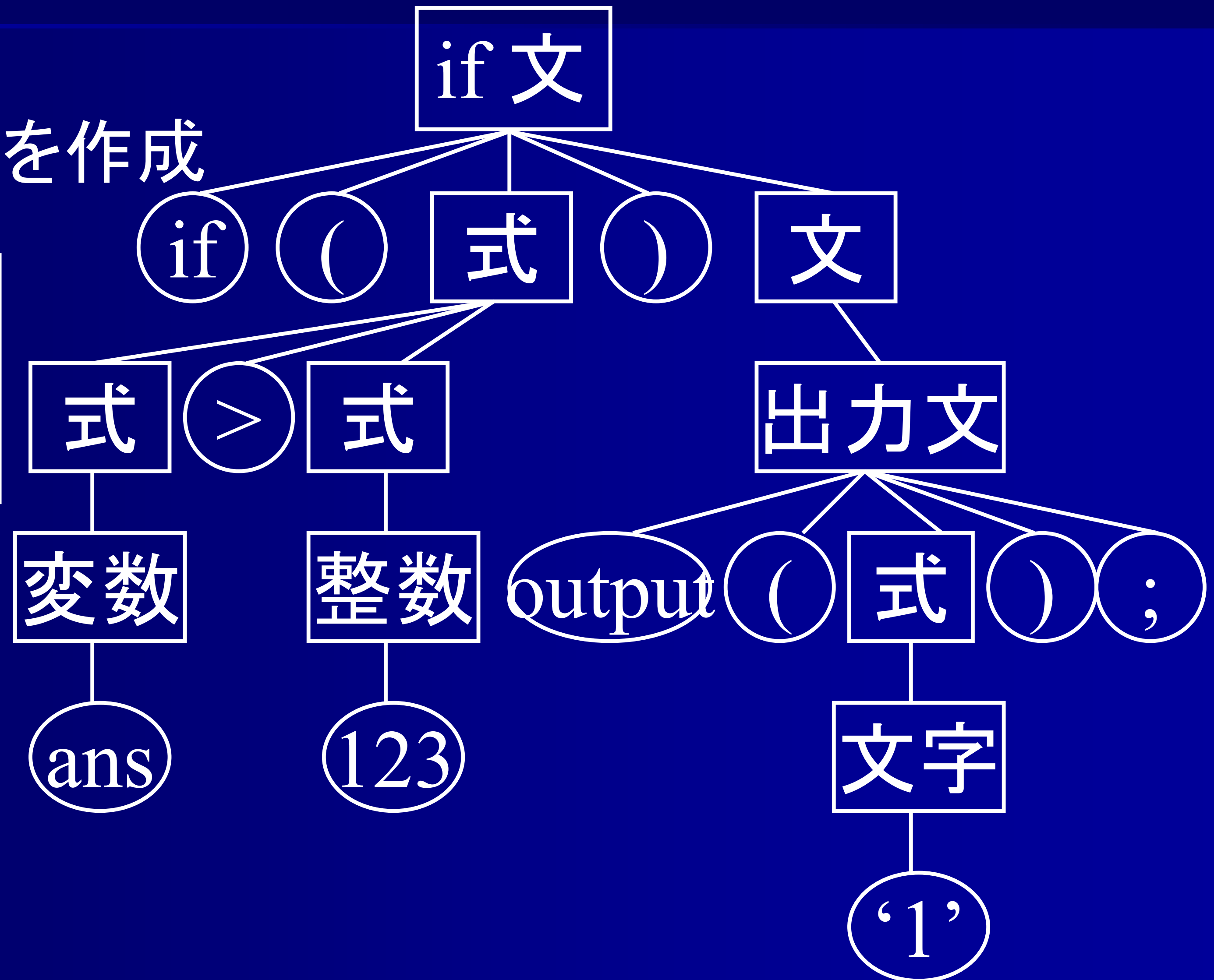
```
if (ans >= 123 ) /* ansの値で分岐 */ (改行)
    (空白)output ('1') ;
```

予約語 "if"
左括弧 "("
変数 "ans"
不等号 ">="
整数 "123"
右括弧 ")"
予約語 "output"
:

構文解析系 (syntax analizer, parser)

- 構文解析系
  - 構文解析木を作成

```
if (ans > 123 )
  output ('1') ;
```



制約検査系 (constraint checker)

- 制約検査系
  - 変数の未定義・二重定義・型の不一致などを検査

変数 x は未定義

変数 i は配列ではない

代入の左辺が変数ではない

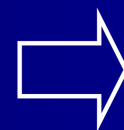
```
int i, j;
x = 0;
i[10] = 5;
0 = 10;
```

中間コード生成系 (semantics analyzer, intermediate code generator)

- 意味解析系
  - 単純な命令の列(中間コード)を生成する
- 中間コード(intermediate code)
  - ハードウェアには依存しない
  - 3番地コード(three address code)が多用される

```
A := B op C
```

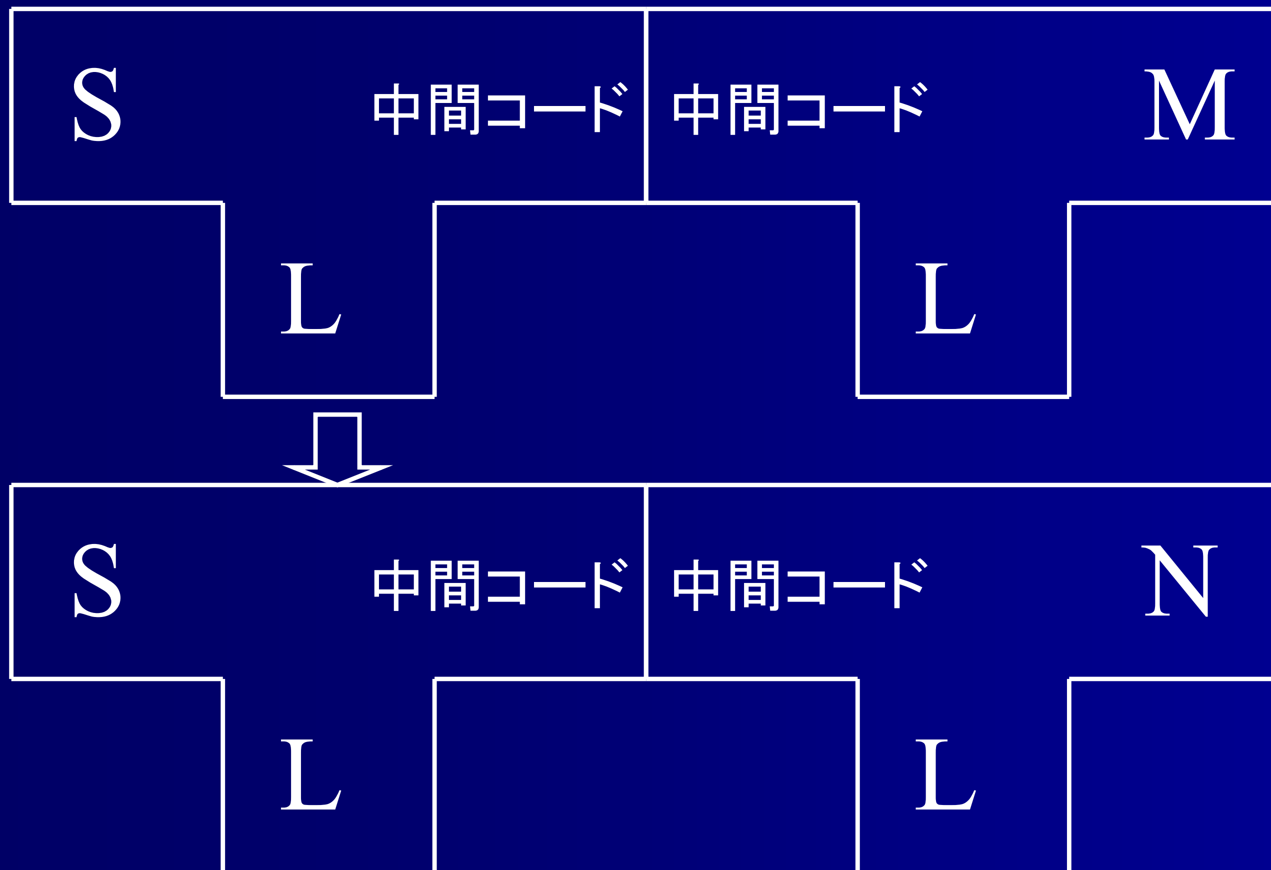
```
if (a>0) b:=2*a+b;
```



```
  if (a≦0) goto L:
  t := 2 * a
  b := t + b
L:
```

中間コードを用いる利点

中間コードはハードに依存しない
⇒異なるハードで共通で使用可能



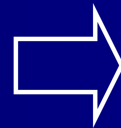
計算機M用
コンパイラ

計算機N用
コンパイラ

最適化系 (optimizer)

- 最適化系
  - 中間コードを改良
    - 実行速度を速く
    - メモリ使用領域を小さく

```
  if (a≦0) goto L:
  t := 2 * a
  b := t + b
L:
```



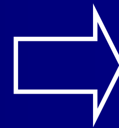
```
  if (a≦0) goto L:
  t := a + a
  b := t + b
L:
```

掛け算より
足し算の方が
速い

目的コード生成系 (object code generator)

- 目的コード生成系
  - 変数の記憶位置決定
  - レジスタの割付

```
   if (a≦0) goto L:
   t := a + a
   b := t + b
L:
```



```
   LD   GR1, a
   LEA  GR1, 0, GR1
   JMI  L:
   JZE  L:
   ADD  GR1, a
   ADD  GR1, b
   ST   GR1, b
L:
```

表管理 (table manegement, bookkeeping)

- 表管理
  - 原始プログラム中の変数,関数等の名前,型情報等を記憶

```
int i, j;
char ch;
int a[10];
```

変数名	型	サイズ	番地
i	int	1	0
j	int	1	1
ch	char	1	2
a	int[]	10	3〜12

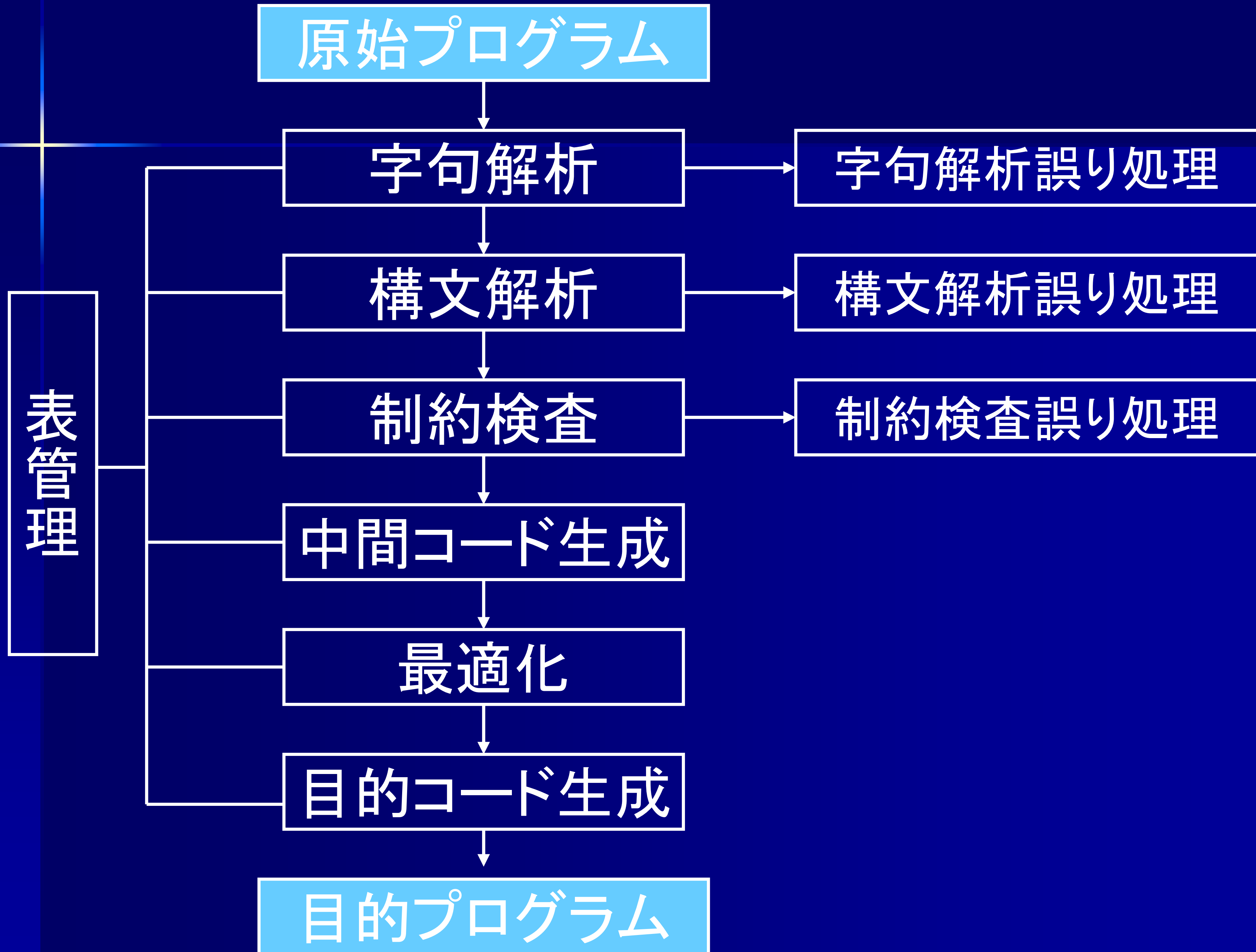
誤り処理(error handling)

- 誤り処理
  - 原始プログラムが言語の制約を満たしていない場合にエラーメッセージを出す

```
int あ, い, う;
if () output (1);
5 = a;
```

1行目で字句解析エラー:
　変数名に日本語は使えません
2行目で構文解析エラー:
　if文の条件式がありません
3行目で制約検査エラー:
　代入の左辺が変数ではありません

コンパイラの構成



コンパイラの構成
(情報システムプロジェクトIの場合)

原始プログラム　K23言語

- 字句解析 → 字句解析誤り処理
- 構文解析 → 構文解析誤り処理
- 制約検査 → 制約検査誤り処理
- ~~中間コード生成~~
- ~~最適化~~
- 目的コード生成

目的プログラム　VSMアセンブラ言語

表管理

処理の流れ
(情報システムプロジェクトIの場合)

output (ab);

↓

字句解析系　　マイクロ構文の文法に従い解析

↓

output（　変数名　）；

↓

構文解析系　　マクロ構文の文法に従い解析

↓

<output_statement> ::= “output” “(” <exp> “)” “;”

↓

コード生成系　　VSMアセンブラの文法に従い生成

↓

1. PUSH ab　　2. OUTPUT

スタックマシン (stack machine)

- スタックマシン
  - Instruction Iseg[] : アセンブラプログラムを格納
  - int Dseg[] : 実行中の変数値を格納
  - int Stack[] : スタック(作業場所)
  - int Program Counter : 現在の Iseg の実行位置
  - int Stack Top : 現在のスタックの操作位置

スタックマシン (stack machine)

Program Counter: 3 (→ Iseg 3)

Iseg

0	PUSHI 0
1	PUSHI 3
2	ASSGN
3	PUSHI 7
4	ASSGN
5	ADD
6	OUTPUT
7	HALT

Dseg

0	3
1	0
2	0
3	0
4	0
5	0
6	0
7	0

Stack

0	3
1	7
2	-
3	-
4	-
5	-
6	-
7	-

Stack Top: 1 (→ Stack 1)

Iseg と Program Counter

- VSM の動作
  1. Iseg の PC 番地の命令を実行
  2. PC := PC+1 or ジャンプ命令で指定した先

Program Counter

4

Iseg

0	PUSHI 0
1	PUSHI 3
2	ASSGN
3	PUSHI 7
4	ASSGN
5	ADD

Dseg

- 実行中の変数値を格納

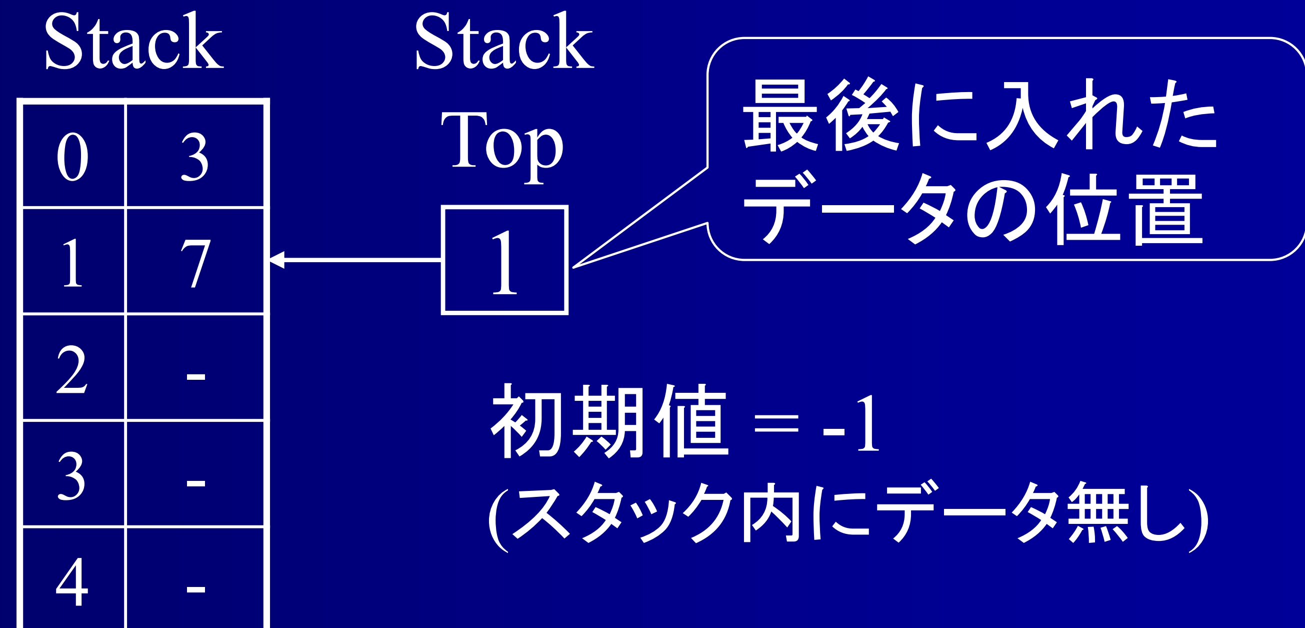
```
int i, j, x=2, y=3;
char c = 'a';
int a[5];
```

Dseg

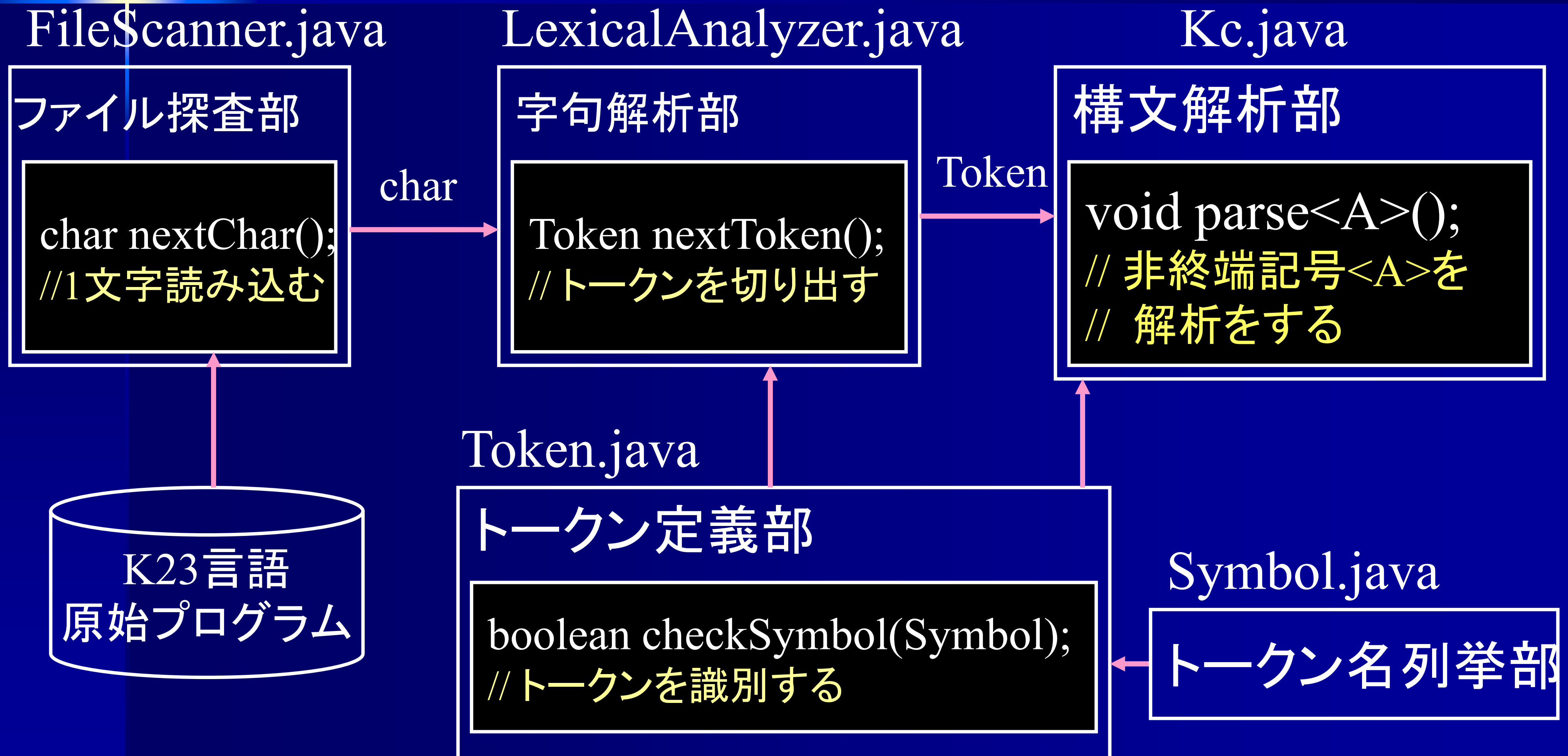
0	0	i
1	0	j
2	2	x
3	3	y
4	'a'	c
5	0	a[0]
6	0	a[1]
7	0	a[2]
8	0	a[3]
9	0	a[4]

Stack

- Stack
  - 作業場所，処理中のデータの一時置き場
  - Last In First Out



プログラムの構造 (字句解析系・構文解析系)



プログラムの構造(コード生成系)

Kc.java

構文解析部

```
void parse<A>();
// 非終端記号<A>を
//  解析をする
```

VarTable.java

変数表格納部

```
boolean registerNewVariable();
// 変数を加える
boolean exist();
// 変数の存在判定
boolean checkType(Type);
// 型識別
```

PseudoIseg.java

命令表格納部

```
int appendCode();
// 命令を加える
void replaceCode();
// 命令を変更する
void dump2file();
// 命令を出力する
```

Instruction.java

命令部

Operetor.java

命令名列挙部

Var.java

変数部

Type.java

型名列挙部

VSMアセンブラ
目的プログラム

宿題

- 「言語理論とオートマトン」の復習をする
  - 有限オートマトン
  - 正則表現
  - 正則文法
  - BNF記法, EBNF記法

課題テスト

- 毎週 GoogleClassroom上で課題テストを行う
  - 授業後～翌週の授業開始まで
- GoogleClassroomで

  コンパイラ

  ⇒授業

  ⇒その回の課題

  と辿る

classroom.google.com

2023-コンパイラ
理工学部情報学科情報システムコース3年生

ストリーム　授業　メンバー　採点

＋ 作成

Google カレンダー　クラスのドライブ フォルダ

- すべてのトピック
- 第4回 : 字句解析(2)
- 第3回 : 字句解析(1)
- 第2回 : 形式言語と...
- 第1回 : コンパイラ...
- Slack について

第4回 : 字句解析(2)

第4回 講義資料		投稿日: 3月24日
第4回 課題	各週課題	*下書き*

第3回 : 字句解析(1)

第3回 講義資料		投稿日: 3月24日
第3回 課題	各週課題	*投稿予定: 4月20日 8:00*

第2回 : 形式言語と形式文法

classroom.google.com

2023-コンパイラ
理工学部情報学科情報システムコース3年生

ストリーム 授業 メンバー 採点

第2回：形式言語と形式文法

第2回 講義資料		投稿日: 3月23日
第2回 課題	各週課題	投稿予定: 4月13日 8:00

第1回：コンパイラの概要

第1回 講義資料		投稿日: 3月23日
第1回 課題	各週課題	期限: 4月19日

Slack について

Slack について	投稿日: 昨日

classroom.google.com

2023-コンパイラ
理工学部情報学科情報システムコース3年生

ストリーム 授業 メンバー 採点

第1回：コンパイラの概要

第1回 講義資料 投稿日: 3月23日

第1回 課題 各週課題 期限: 4月19日

投稿日: 3月23日

Google Form から回答してください。

0 提出済み

0 割り当て済み

第1回：コンパイラの概要
Google フォーム

課題を表示

Slack について

Slack について 投稿日: 昨日

docs.google.com

「2023-コンパイラ 理工学部情報学科情報システムコース３年生」の授業

第１回：コンパイラの概要

第1回：コンパイラの概要

@info.kindai.ac.jp アカウントを切り替える

このフォームを送信すると、メールアドレスが記録されます

*必須

あなたの氏名を入力してください． *

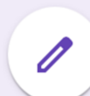
回答を入力

あなたの学籍番号を入力してください．(例：2110370999) 省略形は使用しないでください． *

回答を入力

コンパイラについて正しいものを選べ　20 ポイント

○ 低水準言語で書かれたプログラムを高水準言語で書かれたプログラムに翻訳する



質問

石水隆
E館　3F E-331

オフィスアワー
金曜3,4時限

takasi-i@info.kindai.ac.jp