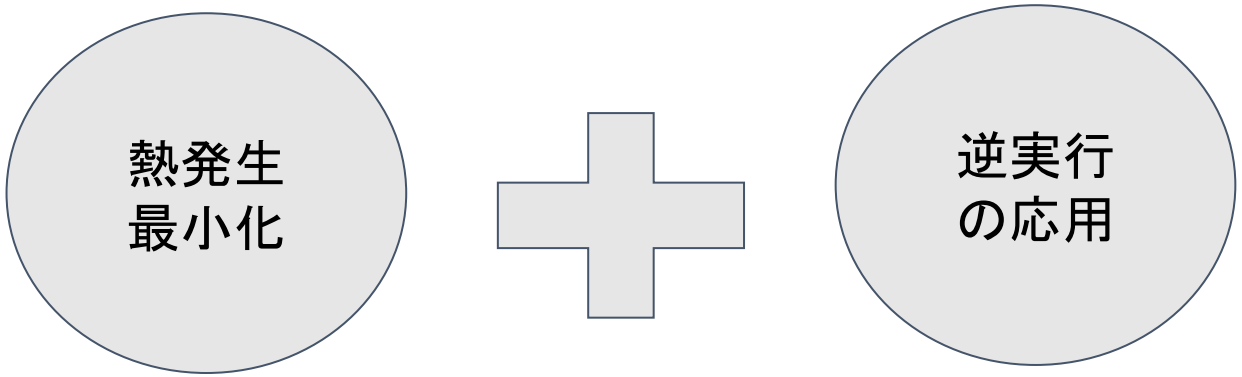
可逆コンパイラの設計および実装

2016SE082　外川 淳平
指導教員　横山 哲郎

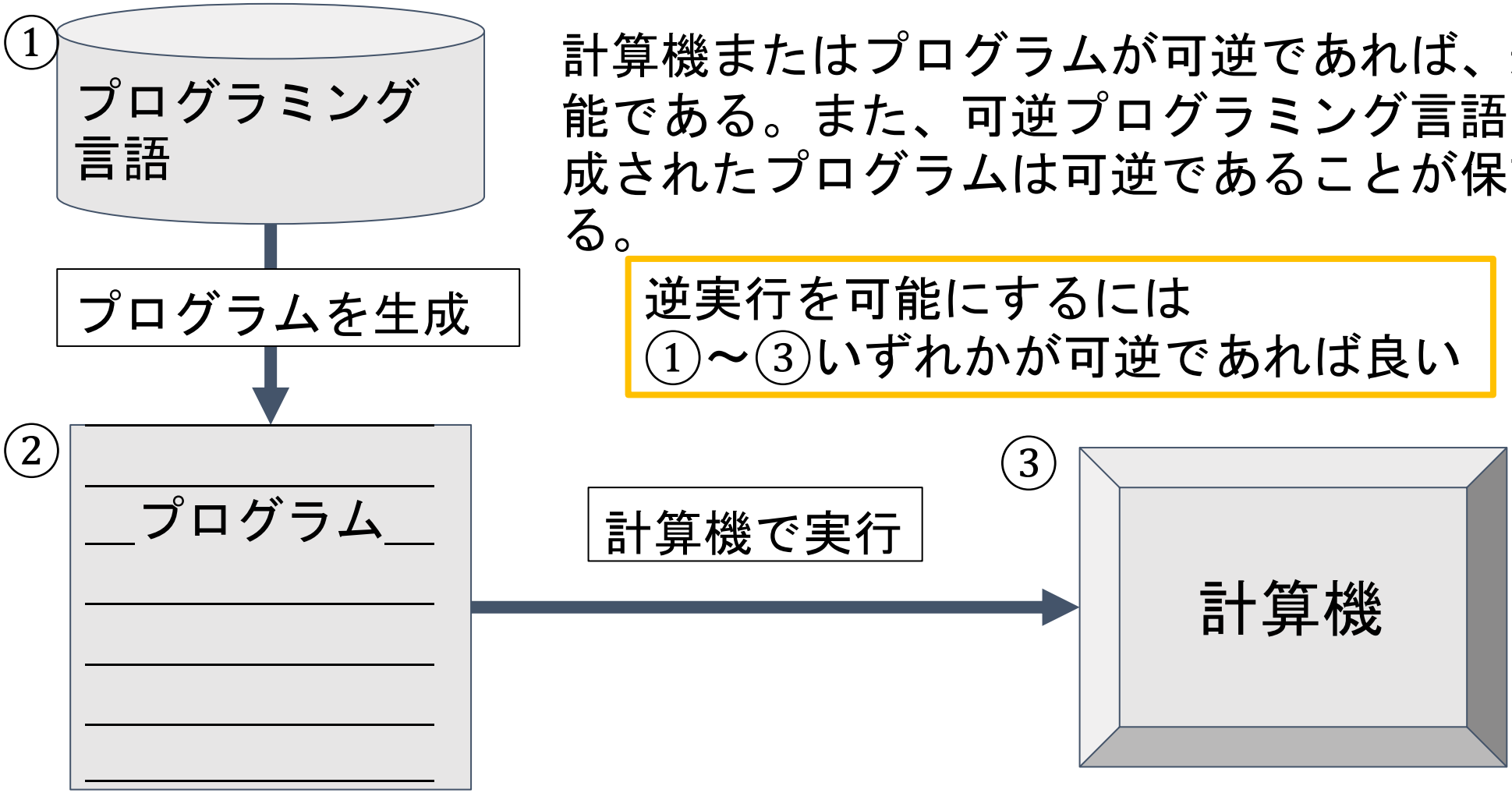
目次

1．背景：可逆プログラミング

- 可逆論理演算
  - 可逆な論理演算では熱発生の下限が存在しない
- 可逆計算機（例：可逆チューリングマシン）
  - チューリング完全
- プログラム逆実行の応用
  - 投機的実行でのロールバックの代用
  - デバッグの効率化



1　背景:プログラム逆実行



計算機またはプログラムが可逆であれば、逆実行は可能である。また、可逆プログラミング言語によって生成されたプログラムは可逆であることが保証されている。

逆実行を可能にするには
①～③いずれかが可逆であれば良い

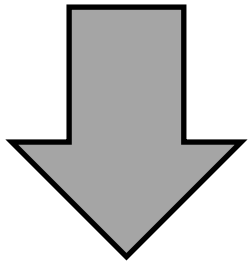
関連研究：可逆コンパイラ

- 可逆コンパイラを利用した投機的実行を含む並列シミュレーション［3］
  - 従来の状態保存方式に対して可逆化されたプログラムを利用することでプロセッサ数に対してキャッシュミスやイベント毎秒での効率向上を提示
  - +=, -=などの基本的なオペレータ、乱数生成や配列操作演算等の逆計算がメモリ効率の向上を提示

- 投機的実行を含む並列離散事象シミュレーションでの使用［2］
  - C++ Standard Libraryから演算子オーバロード、動的メモリ確保、ユーザ定義型、新しい演算子を示し、非可逆なC++を可逆化
  - ROSSシミュレータとの並列実行が可能

２．目的

効率的な可逆プログラムの生成



単一変換をより効率的に行うことにより、
実行性能の向上、メモリ効率の向上、
変換結果のオーバヘッドの削減

アプローチ

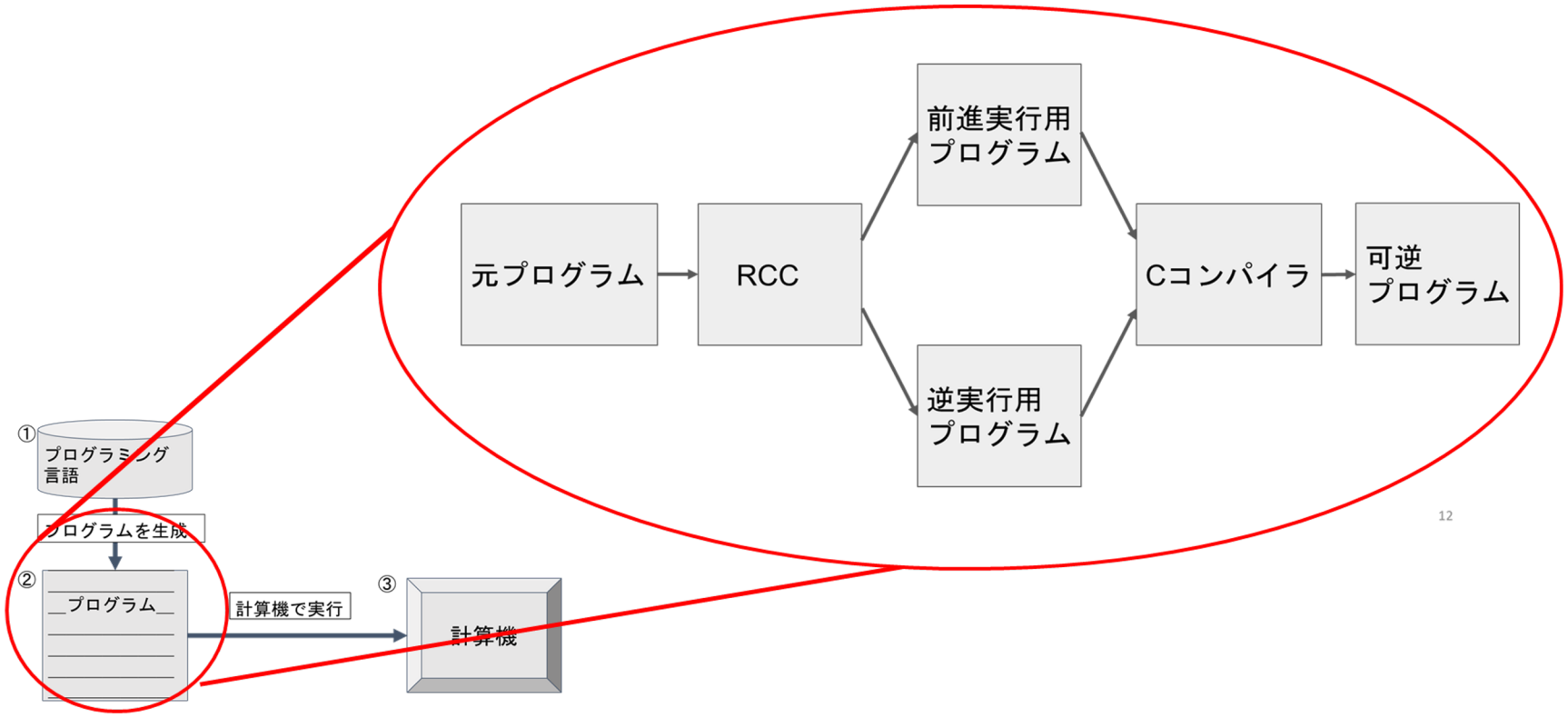
１．RCCの設計と実装(Perumalla らの研究[4])

２．　RCCの設計と実装

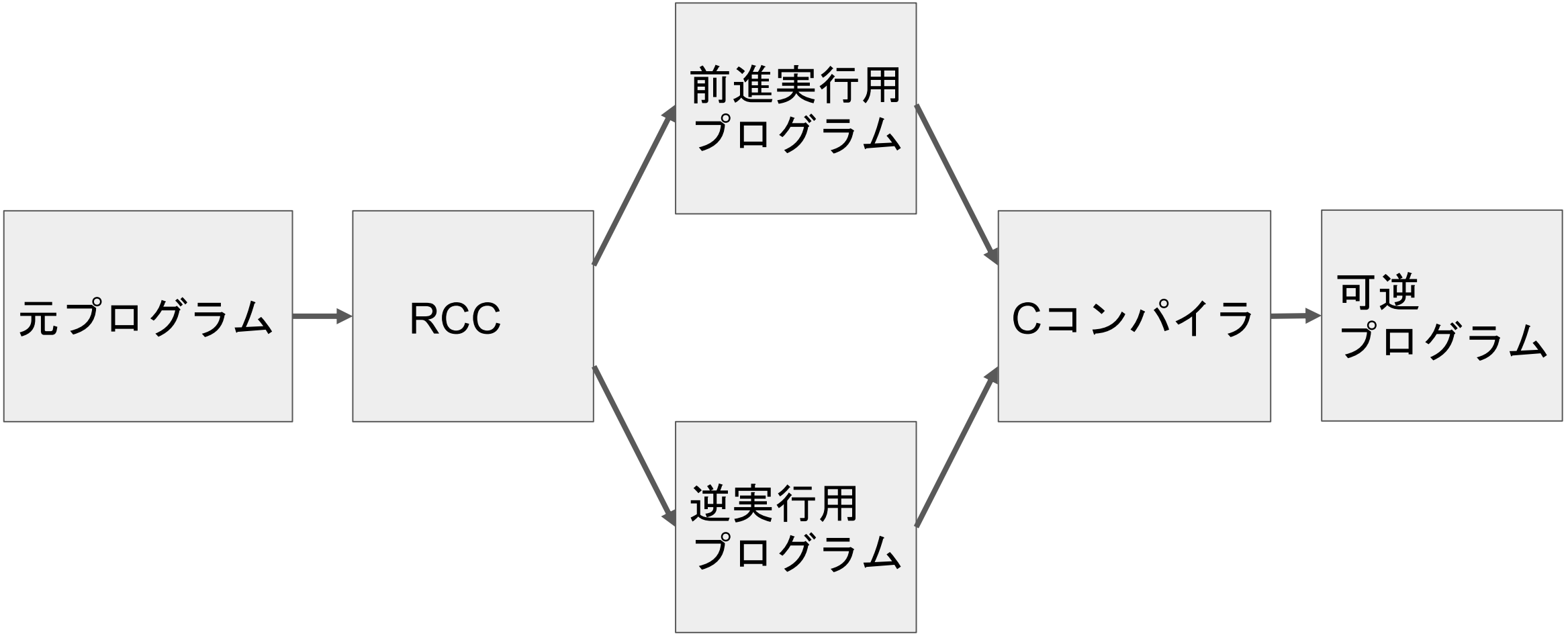
３．　１と２の比較を行う

３．RCCの設計： Reverse C Compiler

- Reverse C Compiler（RCC）とはC言語で書かれた非可逆を含むプログラムを可逆なプログラムへと変換するものである



３．RCCの設計： Reverse C Compiler



３．RCCの設計：繰り返し文

元プログラム	前進実行用 プログラム	逆実行用 プログラム

while(condition)
　　s
	{
	　Int c = 0;
	　while(condition)
	　{
	　 c++;
	　 s
	　}
	　SAVE(c);
	}
		{
		　Int c = 0;
		　RESTORE(c);
		　while(c > 0)
		　{
		　 rs
		　 --c;
		　}
		}

３．RCCの設計：分岐文

元プログラム	前進実行用 プログラム	逆実行用 プログラム

switch(expr)
{
case v1:
s1
break;
case v2:
s2
/*fall-thru*/
case v3:
s1
break;
default:
s4
break;
}
	{
	char c = 0, d = 0;
	switch(expr)
	{
	case v1: c=1; d++;
	s1
	break;
	case v2: c=2; d++;
	s2
	/*fall-thru*/
	case v3: c=3; d++;
	s1
	break;
	default: c=0; d++;
	s4
	break;
	}
	SAVE(c);
	SAVE(d);
	}
		{
		char c , d ;
		RESTORE(d);
		RESTORE(c);
		switch(c)
		{
		case 0:
		rs4
		if(--d <= 0) break;
		case 3:
		rs3
		if(--d <= 0) break;
		case 2:
		rs2
		if(--d <= 0) break;
		case 1:
		rs1
		}
		}

４．RCCの実装

ツールはBNFC、言語はHaskellを用いる。

１．Cコンパイラをダウンロード
→既存のものを使用
２．入力と出力が全く同じコンパイラの作成
→無変換のコンパイラの作成
３．非可逆部分を変更するコンパイラの作成
→Translate.hs ,InvTranslate.hsのファイルを作成

４．RCCの実装:SAVEとRESTOREのマクロ

【Translate.hsにSAVEのマクロを追加】

```
save str = ExprS (SexprTwo (Efunkpar (Evar (Ident "SAVE")) [Evar (Ident str)]))
```

【InvTranslate.hsにRESTOREのマクロの追加】

```
restore str = ExprS (SexprTwo (Efunkpar (Evar (Ident "RESTORE")) [Evar (Ident "c")]))
```

４．RCCの実装:分岐文

【Translate.hs の変更】

```
SselOne exp stm ->
  let stm' = transStm stm in
   CompS (ScompFour [Declarators [Type Tchar] [InitDecl (NoPointer (Name
   (Ident "c"))) (InitExpr (Epreop Logicalneg (Epreop Logicalneg exp)))]] [SelS (SselOne
   (Evar (Ident "c")) stm'),save "c"])
```

【InvTranslate.hs の変更】

```
SselOne exp stm ->
  let rstm = transStm stm in
   CompS (ScompFour [Declarators [Type Tchar][OnlyDecl (NoPointer (Name
   (Ident "c")))]][restore "c",SelS (SselOne (Evar (Ident "c")) rstm)])
```

５．まとめ

- 可逆プログラムの効率化

→RCCでの単一変換の変更

- RCC（Perumalla らの研究［4］）の設計と実装を行う。

→BNFCをつかいHaskellで設計と実装を行う。

５．まとめ：今後の課題

・効率化の方法

→現状が想像程度であるため、具体化が必要。

・RCCの実装

→分岐、繰り返し、代入の代表的な非可逆部分から優先に実装し比較対象の確保

・RCCの効率的な環境の定義

→投機的実行を含む離散事象シミュレーションを想定しているがより詳しく確認する必要があると考えている

６．参考文献

[1] Hänninen, I., Lu, H., Blair, P.E., et al.: Reversible and Adiabatic Computing: Energy-Efficiency Maximized, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Preface, pp.341–356 (2014).

[2] Schordan, M., Oppelstrup, T., Jefferson, D., et al.: Generation of Reversible C++ Code for Optimistic Parallel Discrete Event Simulation, New Generation Computing, Vol.36, No.3, pp.257–280 (2018)

[3] Carothers, D.C., Perumalla, S.K., Fujimoto, M.R.: Efficient Optimistic Parallel Simulations Using Reverse Computation, ACM TOMACS, Vol.9, No.3, pp.224–253 (1999)

[4] Perumalla S.K.,: introduction to Reversible Computing, CRC (2013)

前提条件の定義

・投機的実行を含む並列離散事象シミュレーション

　元プログラムより効率でないとその効率化には意味がない。よってこの前提が必要となる。

・特定条件で正当性がないのを認める。

　丸め処理、オーバーフロー、アンダーフローなどにより、正当性が損なわれる場合がある。

５．まとめ：今後の課題

・効率化の方法

→現状が想像程度であるため、具体化が必要。

・追加情報記憶用の変数の再利用

（現状、以前に使われてないある変数を毎回定義）

・RCC（［４］）は追加情報をその分だけの変数を用意して記憶するという、単純な仕組みが主であるため、変換規則に工夫を持ち込むことは可能と考える。

４．期待される効果

- 単一変換をより効率的に行うことにより、実行性能の向上、メモリ効率の向上、変換結果のオーバヘッドの削減

- 可逆コンパイラをGitHubで公開することで非可逆コードの可逆変換を誰でも簡単に利用可能

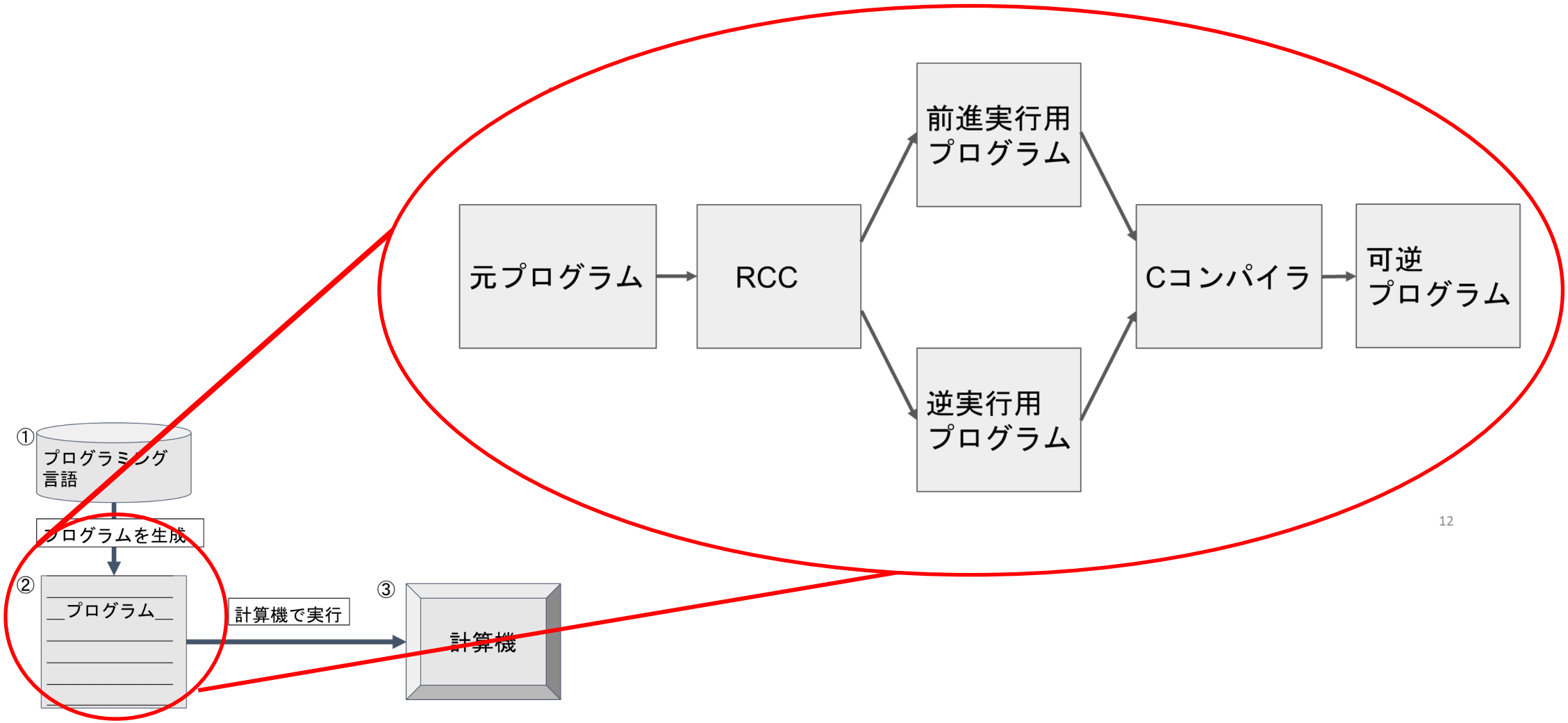
元プログラム	可逆プログラム	逆プログラム

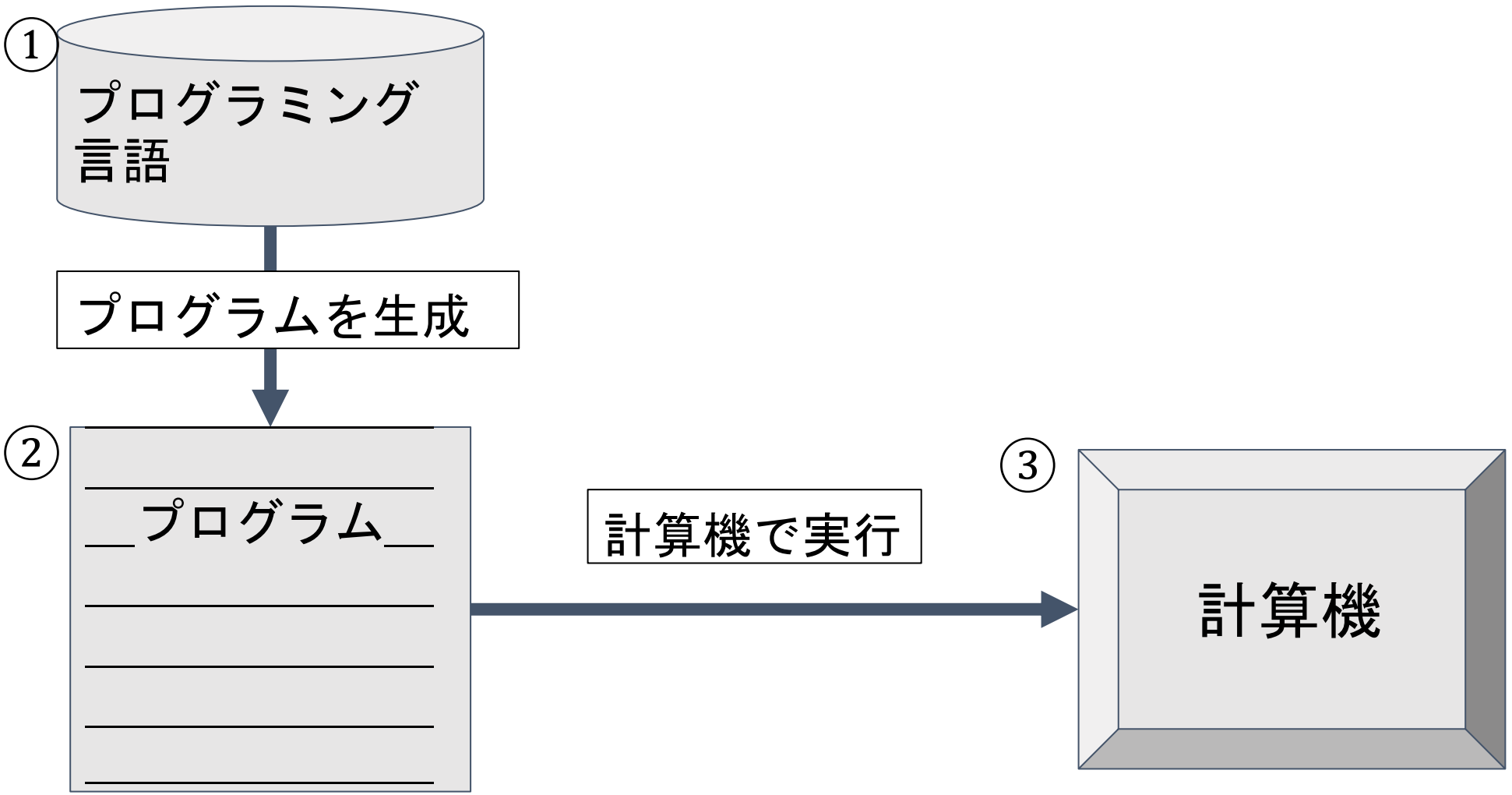
if(test)
  s1
else
  s2
	{
	char c = !!test;
	if(c)
	  s1
	else
	  s2
	SAVE(c);
	}
		{
		char c ;
		RESTORE(c);
		if(c)
		  rs1
		else
		  rs2
		}

３．アプローチ

- テストケースコードを作成し、コンパイラを通して可逆変換ができているか、変換前後のふるまいが同様か確認
- アーキテクチャの提示
- If, jump, while　表
- 構文の変換規則のカバレッジを提示

３．RCCの設計


元プログラム
RCC
前進実行用
プログラム
逆実行用
プログラム
Cコンパイラ
可逆
プログラム
12
①
プログラミング
言語
プログラムを生成
②
__プログラム__
計算機で実行
③
計算機


①
プログラミング
言語
プログラムを生成
②
プログラム
計算機で実行
③
計算機