Software Obfuscation by Mixing Instructions and Data

米澤研究室
M1　佐藤秀明

概要

- 命令とデータの区別を難しくする難読化手法の研究

難読化とは

- 難読化 (obfuscation)
  - プログラムを読みにくいものに変換すること
    - 元のセマンティクスはそのまま保存

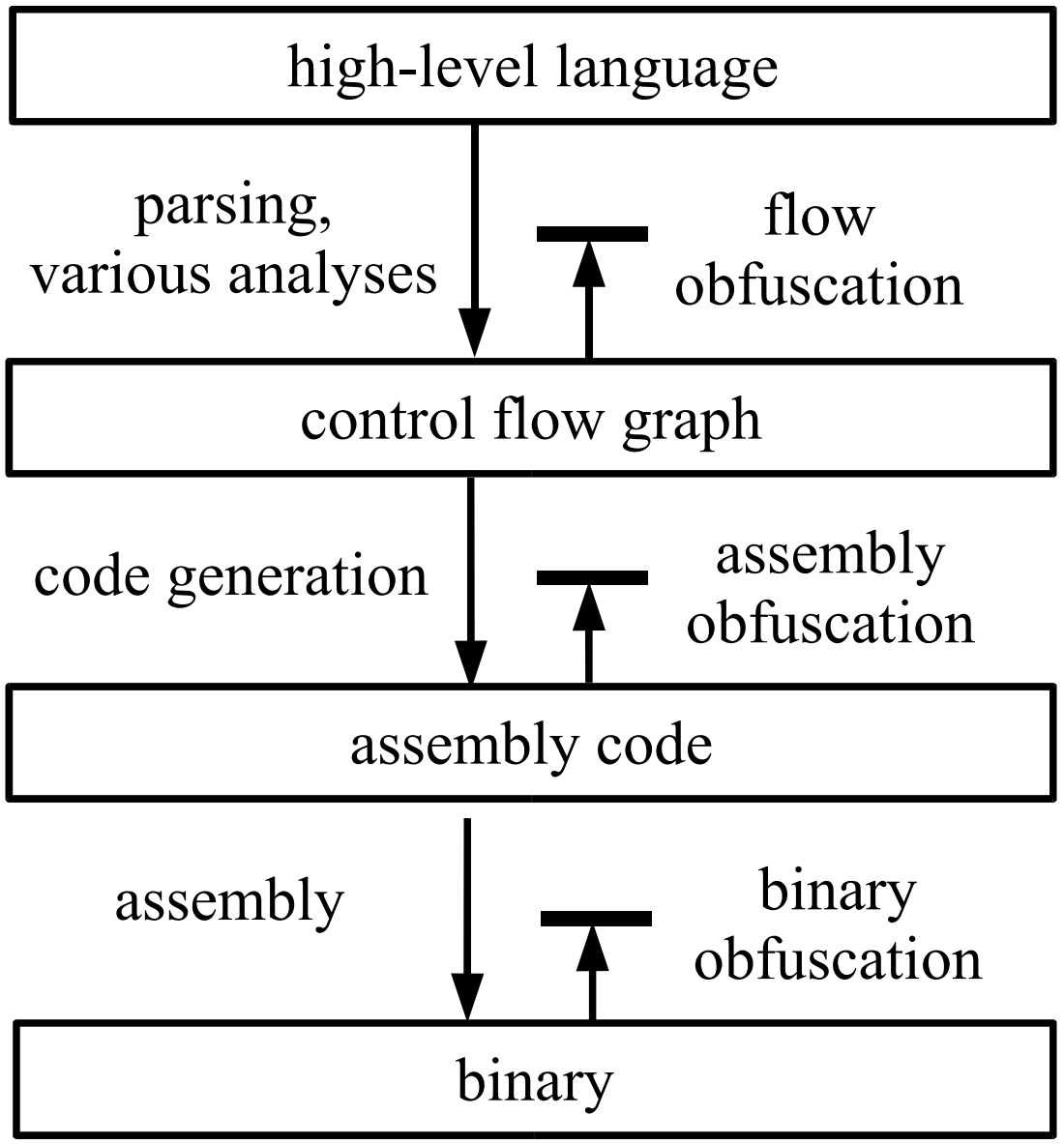
プログラムを難読化する理由

- リバースエンジニアリングの防止
  - 知的財産を保護する
  - 脆弱性を攻撃者が発見しにくくする
  - プログラムの望まれない改変を抑止する

リバースエンジニアリングとの対応 (1)

- Binary obfuscation
  - 逆アセンブルを難しくする技術
- Assembly obfuscation
  - アセンブリから制御フローの復元を難しくする技術
- Flow obfuscation
  - 制御フローから高級言語の復元を難しくする技術

リバースエンジニアリングとの対応 (2)



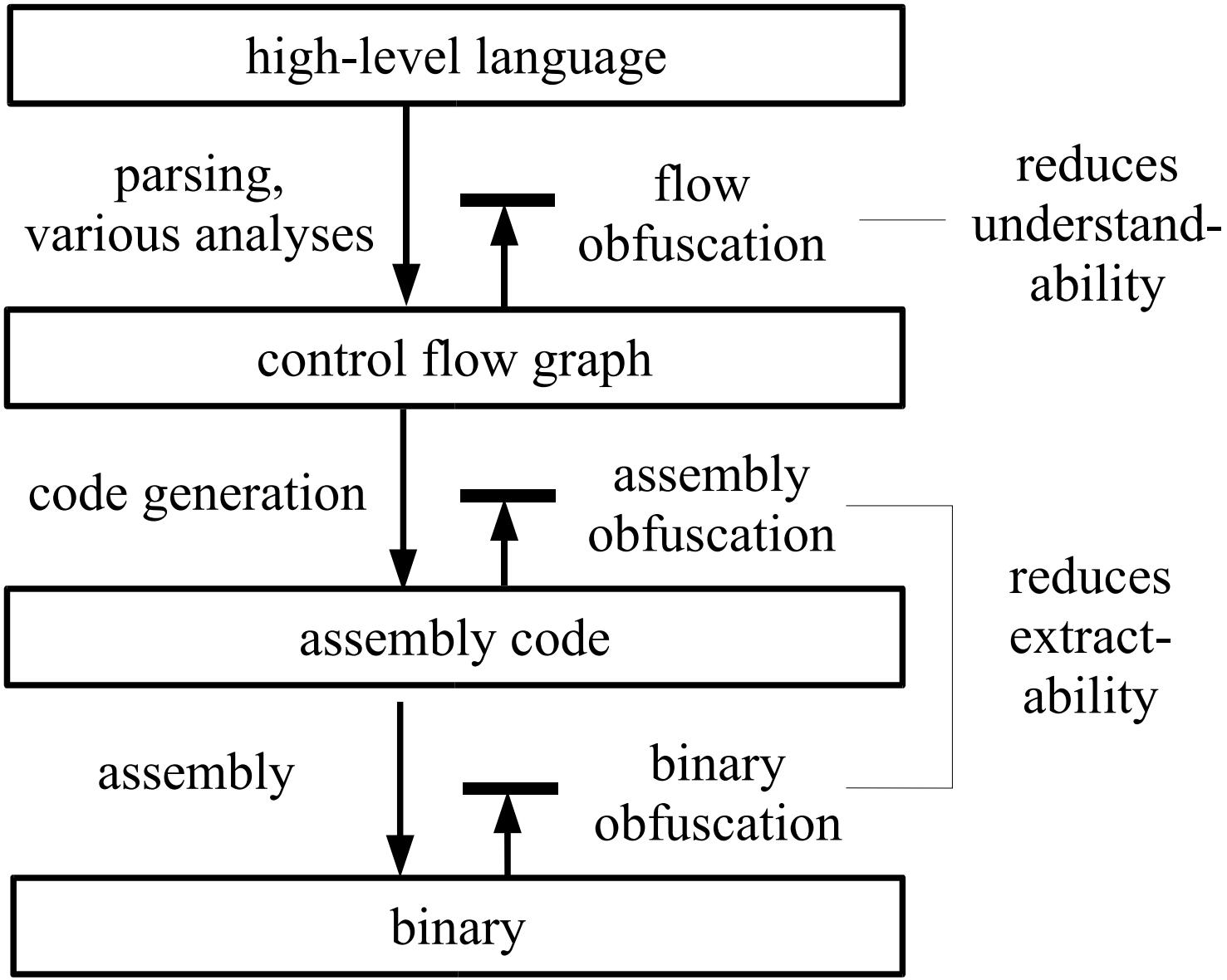
「読みにくさ」とは何か

- Extractability が低い
  - アルゴリズムの正しい抽出が困難
- Understandability が低い
  - 可読性の高い高級言語の復元が困難
    - アルゴリズム自体は既知

「読みにくさ」の定義による分類(1)

- Extractability を低下させる技術
  - Binary obfuscation 、assembly obfuscation
  - 既存研究は少数
- Understandability を低下させる技術
  - Flow obfuscation
  - 既存研究は多数

「読みにくさ」の定義による分類 (2)



我々の方針

- Extractability を低下させる技術に注目
  - 既存技術の数がまだ不足
  - Extractability は厳密な評価が可能
    - Understandability はソースを読む人間の能力によって変化

Binary obfuscation の既存研究

- x86 上でのバイナリ難読化 [Linn et al., 2003]
  - 命令間に junk byte を挿入
    - 誤った命令を逆アセンブラに抽出させる
  - jump 命令を特殊な関数への呼び出しに置換
    - 制御の追跡を困難にする
- 問題
  - 固定長命令の環境では junk byte 法が使えない
  - 彼らの難読化手法の弱点が指摘されている [Kruegel et al., 2004]

Assembly obfuscation の既存研究

- 動的コード生成を用いたバイナリ難読化 [神崎ら、2004]
  - 実行前と実行後はダミー命令で本来の命令を偽装
  - 実行時のみ本来の命令が出現

我々の目的

- Extractability を低下させる新手法の提案
  - 様々なアーキテクチャに適用可能
    - cf. x86 依存 [Linn et al., 2003]
  - 小さなオーバーヘッドで高い難読化効果

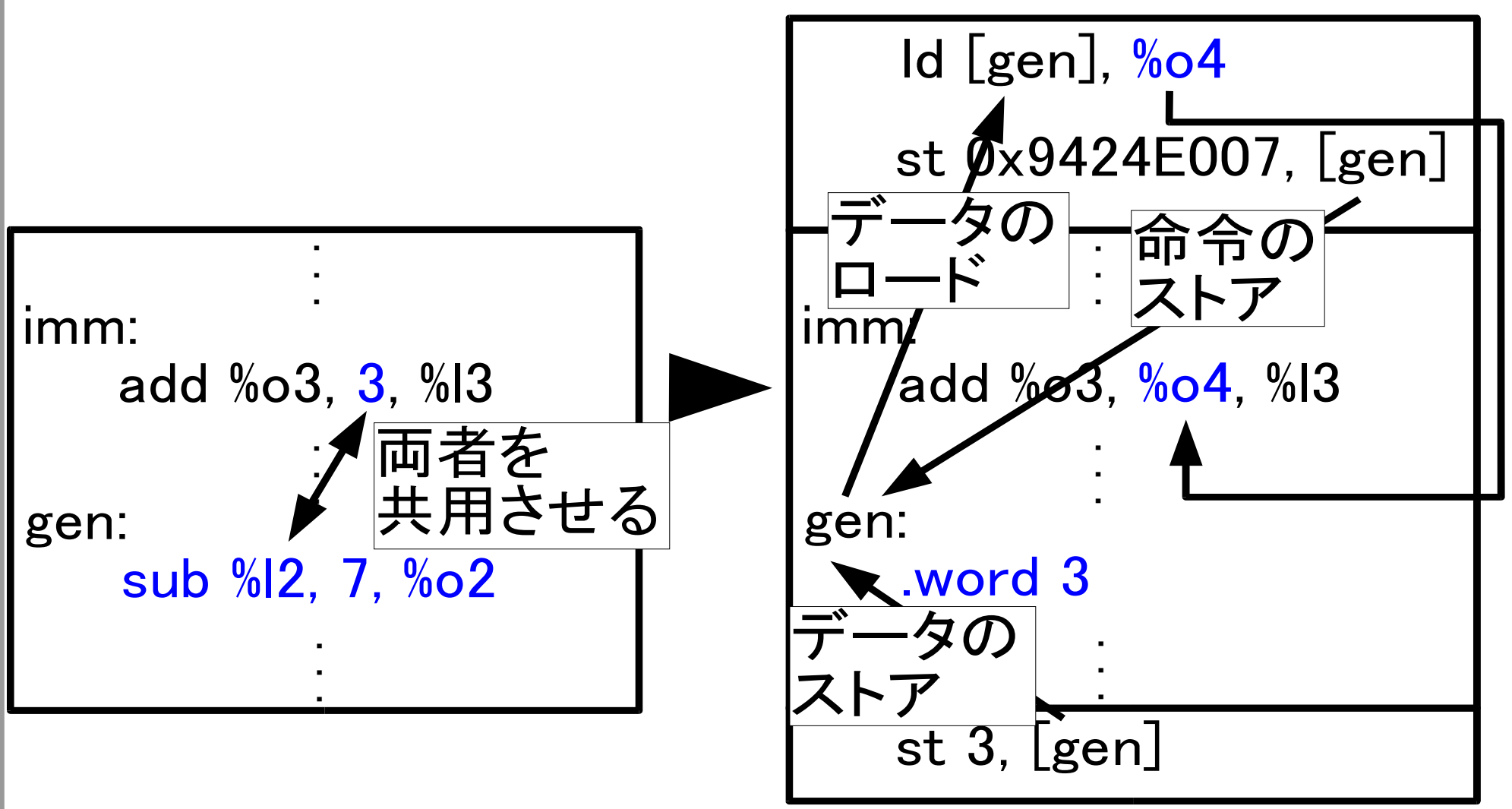
我々のアプローチ

- 命令とデータの区別を難しくするしくみを導入
  - 本来の命令を実際に使用されるデータで隠蔽
    - cf. 本来の命令をダミー命令で偽装 [神崎ら、 2004]
  - 動的生成された命令をデータだと誤解させる

コード難読化の例

難読化前のコード

難読化後のコード



我々の手法の利点

- 静的なリバースエンジニアリングでは検知不可能
  - cf. 動的リバースエンジニアリング…コスト大

実装

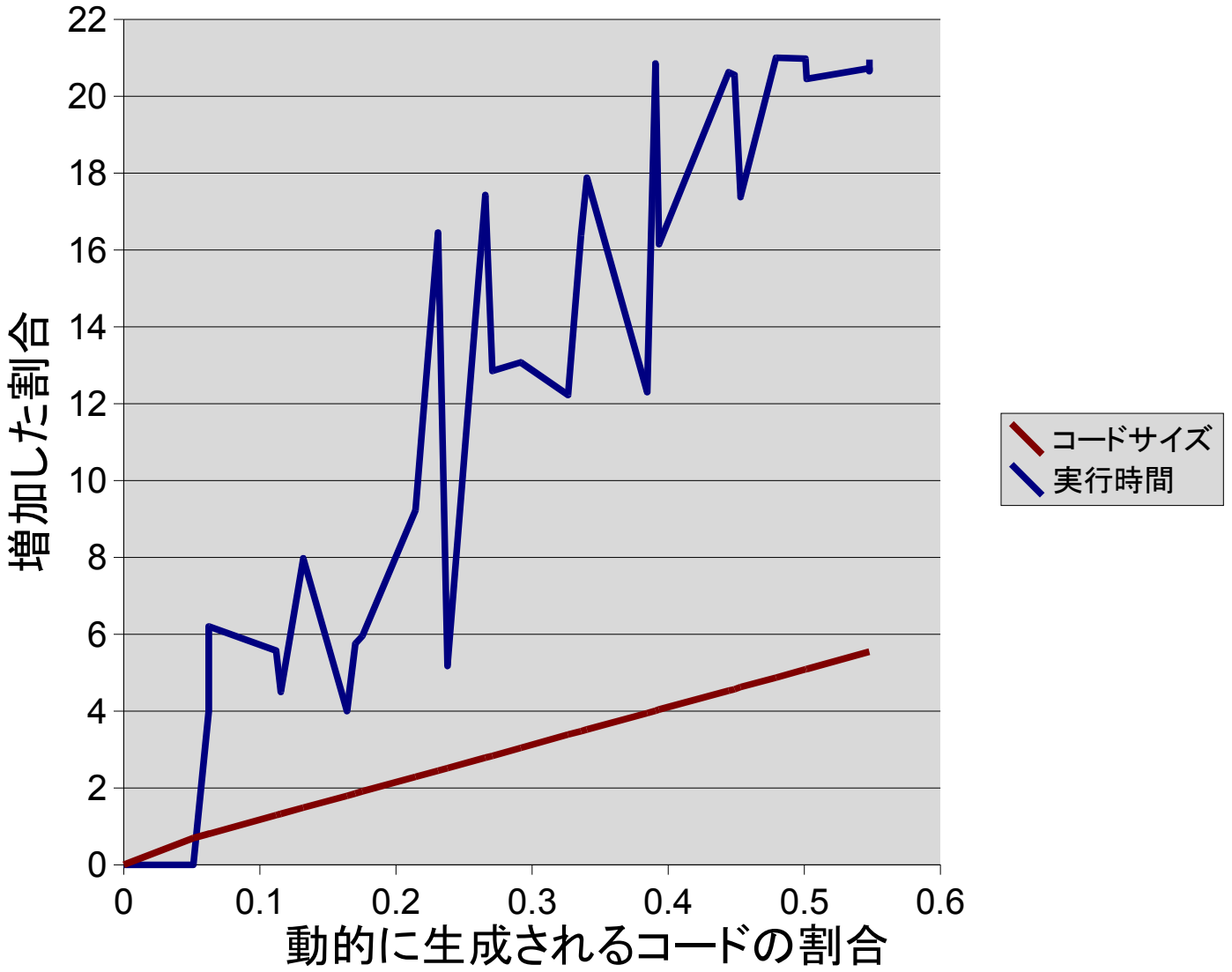
- 環境 :SPARC
- アセンブリ to アセンブリの変換器として実装

評価 (1)

- 逆コンパイラは逆アセンブルの途中で異常終了
  - 逆コンパイラ :boomerang
  - 動的コード生成に対処できない

評価 (2)

- コードサイズと実行時間の測定 (プログラムは LU 分解)



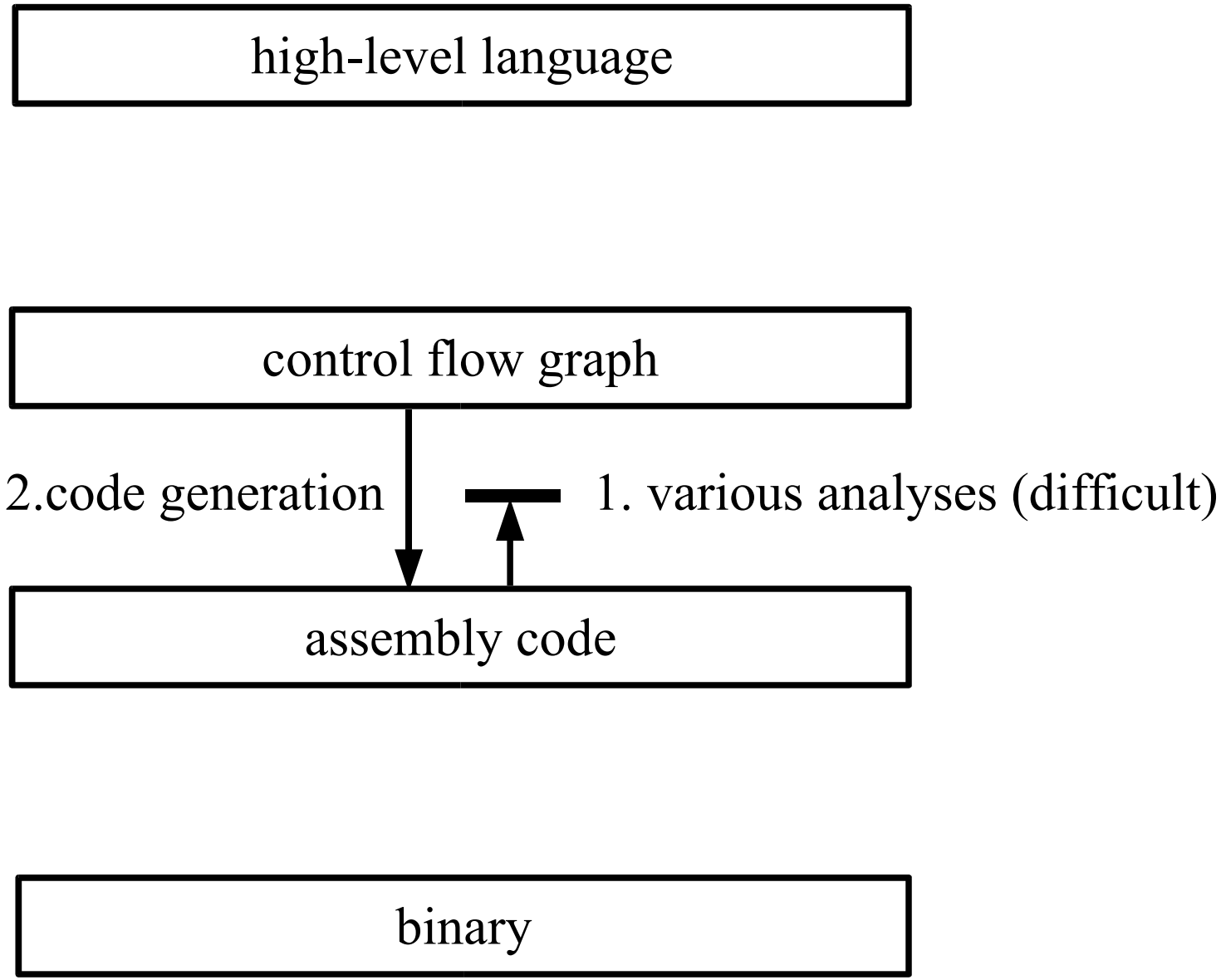
考察

- 同じ動的生成率に対する実行時間の分散が大
  - 繰り返し動的に生成される命令が実行時間に影響大
- 実行時間と難読化レベルとのトレードオフ
  - 難読化の割合に対し実行時間は飛躍的に増加

実装上の問題(1)

- 実際にはアセンブリ to 制御フロー to アセンブリ
  - 生存レジスタの解析などを行うため
  - リバースエンジニアリングの苦労を身をもって知る
    - 失われた情報を復元するのは手間がかかる
    - 難読化されてなくても普通に難しかった

実装上の問題 (2)


high-level language
control flow graph
2.code generation
1. various analyses (difficult)
assembly code
binary

実装上の問題 (3)

- 対策：対象とするアセンブリの種類を制限
  - gcc が C のソースから生成したアセンブリのみ
- 結局はコンパイラの癖に一つ一つ対応するしかない
  - Ad hoc な対処法の寄せ集め
  - 実装が非常に汚なくなる…挫折

しおり

- ここからは構想

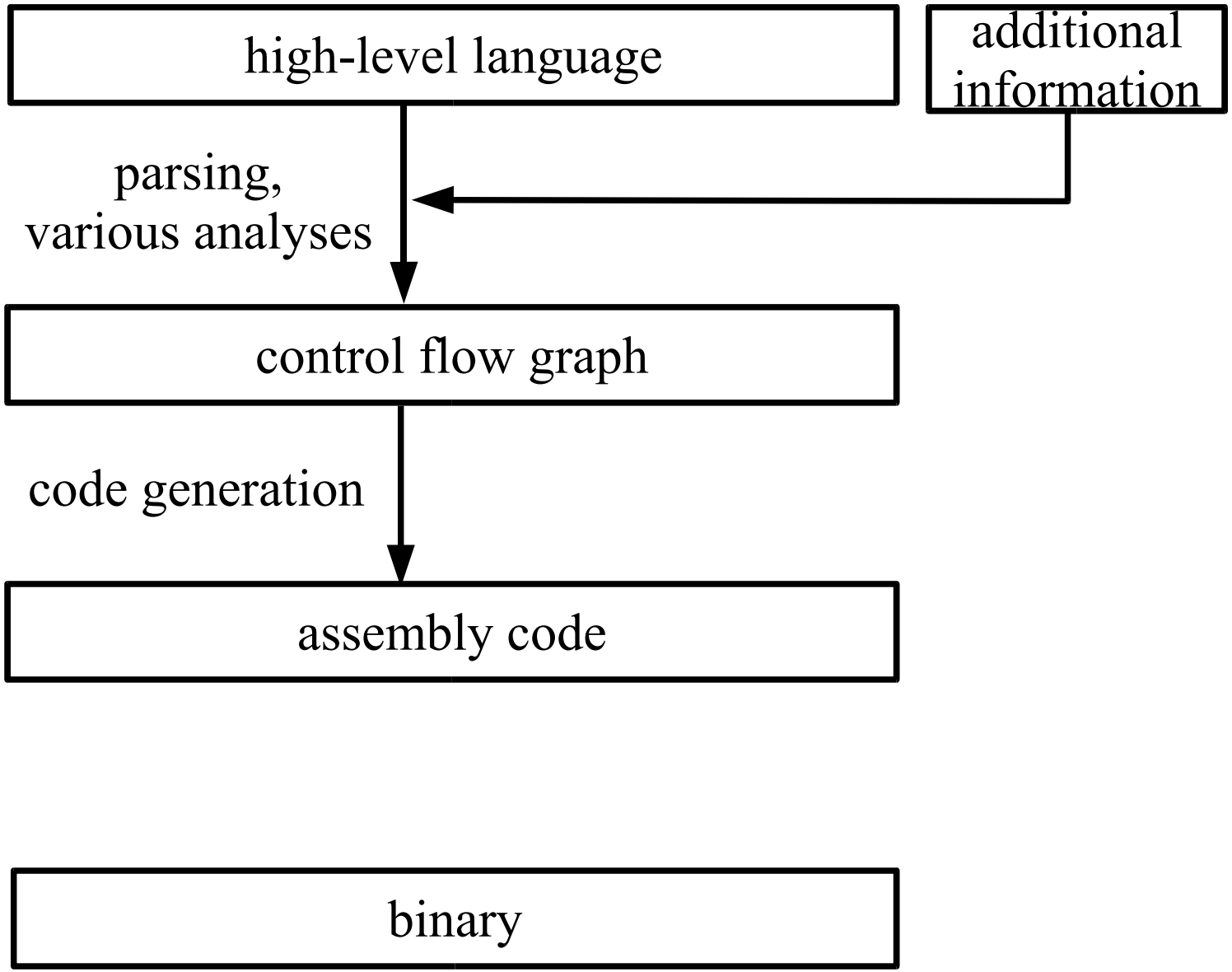
コンパイラとの融合 (消極的な理由)

- ソースコード to アセンブリにすればよい
  - 無理にリバースエンジニアリングする必要はない
  - コンパイラ演習で吸収したノウハウを活かす

コンパイラとの融合 (積極的な理由)

- 難読化したい箇所をプログラマが指定
  - 直接ソース中に annotation
  - アスペクトのように別途指定
- プログラマからの情報を元に不要な難読化を抑止
  - 実行時間オーバーヘッドの減少が期待できる

コンパイラとの融合 (図)


high-level language
additional information
parsing,
various analyses
control flow graph
code generation
assembly code
binary

今後の予定

- 難読化装置をコンパイラに組み込む
  - 元ソースコードの情報を有効活用
  - 難読化する箇所の指定に応える
- 手法の改良 (ちょっと先の話)
  - より aggressive なメモリの使い方
    - 命令やデータの展開先をランダム化