Program Analysis

Lecture 01: *Introduction*
Winter term 2011/2012

Prof. Thorsten Holz

Outline

- Overview of class structure
- Basic overview of class topics
  - Goals of the lecture
  - Literature recommendations
- Brief and high-level introduction to program analysis

Open Positions

- We are looking for some "*SHKs*"
  - Different research projects (smartphone security, botnet detection, binary analysis, ...)
  - Admins
- You need to have some experience in programming
- Start is as soon as possible
- Please send a mail with short CV + motivation to thorsten.holz@rub.de

Websites

- Website: http://emma.rub.de/teaching/courses/514/
- moodle https://moodle.rub.de
  - eTutors will prepare e-learning elements
  - Discussion forum (both admin and questions)
  - Slides will be published after the lectures
  - Interactive questions to test your knowledge
  - Please register soon!

Structure

- Lectures (V2)
  - Thursdays between 9:00 st. and 10:30
- Exercises (Ü1)
  - Every other week between10:30 st. and 12:00
  - Starts on October 27 (*two weeks*)
    - Overview of OllyDbg and IDA Pro
    - First exercise will be handed out

Exercises

- Exercises will introduce tools and techniques for the methods introduced in the lectures, for example:
  - Reverse engineering of binaries
  - Taint analysis to understand programs
  - Binary instrumentation
- Six to seven exercises during the semester
  - Start on October 27 with tool overview

Exercises

- We will have exercises to solve at home
- You can get bonus points for final exam
  - Up to 10 bonus points possible (= *max. +10%*)
  - Round off to nearest number
  - Bonus points count fully in exam
  - No lower limit

Exercises

- We have ordered several licenses for IDA Pro
- Will be available in SysSec lab
  - We are still moving...
  - Should be available by the middle of November
  - Access will be provided during specific times
- We will monitor the machines

Exam

- Written examination at the end of the semester
  - You can get 100% in exam (+ bonus from exercises)
- How many students want to have a certificate?
  - Depending on number of attendees we might do oral examination
  - *Oral examination if there are less than 15 students to take the exam at the end of the semester*

Sources

- Lecture *Software Reverse Engineering* at University of Mannheim (Spring term 2010)
  - Ralf Hund, Carsten Willems, and Prof. Felix Freiling
  - First part of this lecture is based on their slides
  - They kindly allowed me to use their material
- Research papers and other sources will be provided in Moodle
- Provide feedback! (thorsten.holz@rub.de)

Lecture
Program Analysis

Analysis of Programs

- We often want to understand what a given binary program does, for example
  - Program from an unknown source
  - Compatibility
  - Source code is not available (anymore)
  - Unknown malware binaries
  - ...
- *Reverse Engineering* as technique to help us

Goals I

- Introduce different techniques to analyze programs (in particular binary executables)
  - Static analysis
    - Dynamic analysis
- Present the concepts and also tools
- More in-depth overview in “Master-Praktikum” that will presumably be offered in the summer term

Goals II

- Focus of this class
  - Algorithms and techniques to analyze and understand programs
  - Introduction to *software reverse engineering*
  - Analyze a given binary executable on your own
- Basic knowledge for thesis in this area
  - Honeypots, botnets, malware analysis, security of smartphones, ...

Non-Goals

- What is not the goal of this class
  - Complete overview of all aspects of reverse engineering
  - Specific system details might not be covered in too much detail
  - ...

Class will provide you with an overview of techniques and tools - learning by doing!

Topics

- Static and dynamic analysis
- Analysis of control and data flows
- Symbolic execution
- Binary instrumentation
- Taint analysis
- Virtual machine introspection (VMI)
- ...

Guest Lectures

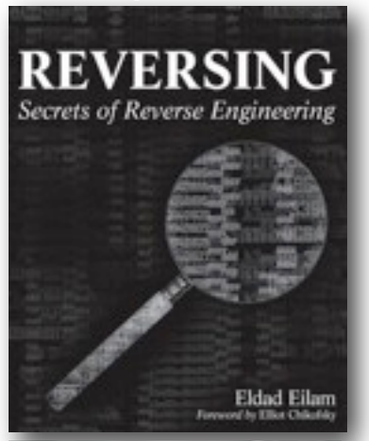
- We plan to have several guest lectures
- Typically in weeks where no exercise takes place
  - Sometimes also instead of lecture or exercise
- More info in the near future
  - Announcement via Moodle

Reading Material

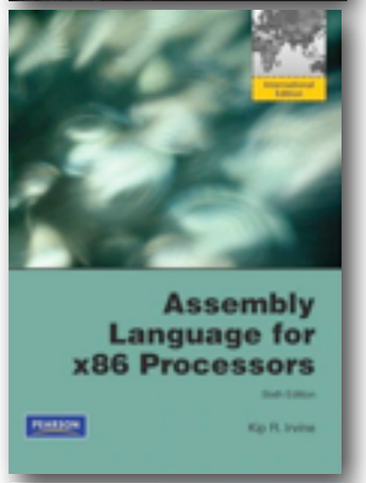
- We will provide an overview of relevant papers in Moodle
  - Typically one or two papers per lecture (*if any*)
  - Useful if you want to get additional information
  - Non-required reading, but we **recommend** to take a look at them

Books

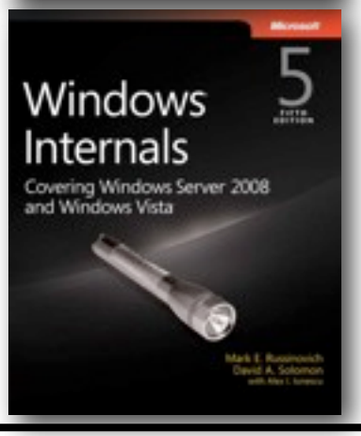
- Eilam: *Reversing: Secrets of Reverse Engineering*, John Wiley & Sons, 2005



- Irvine: *Assembly Language for Intel-based Computers*, Pearson Education, 2010

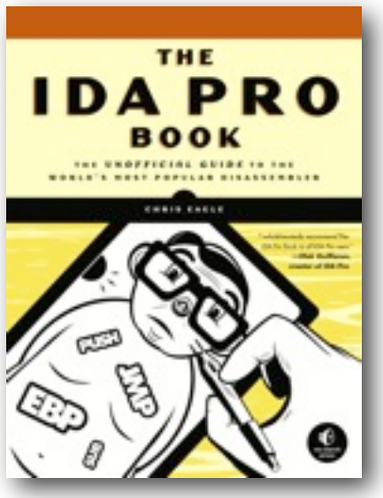


- Russinovich and Solomon: *Windows Internals*, Microsoft Press, 2009

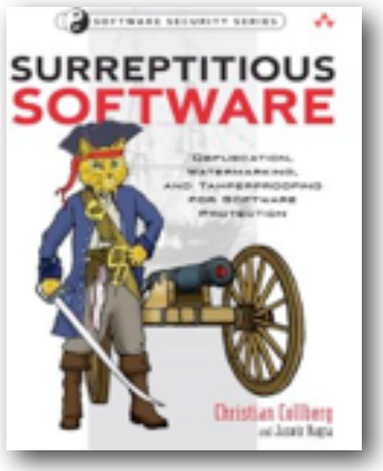


Books

- Eagle: *The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler*, No Starch Press, 2008



- Collberg and Nagra: *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection*, Addison-Wesley, 2009



Introduction

Reverse Engineering

- Our goal: understand different aspects of the compilation step and "reverse it"
  - Reconstruct the control and data flow
  - Understand access to variables
  - Identify higher-level structures (e.g., loops)
  - ...
- Different tools and techniques will be introduced in the next lectures

Definition

- Definition of *Reverse Engineering* by Chikofsky and Cross:

> *Process of analyzing a subject system to:*
> *- identify the system's components and their interrelationships and*
> *- create representations of the system in another form or at a higher level of abstraction*

- Origin is the analysis of hardware designs

Compiler


```
main () {
  // sum numbers from 1 to 42
  int i;
  int j = 0; // initialize counter
  for (i = 1; i <= 42; i++)
      j += 1;
}
```

Compiler

sample.exe

Analysis of Programs

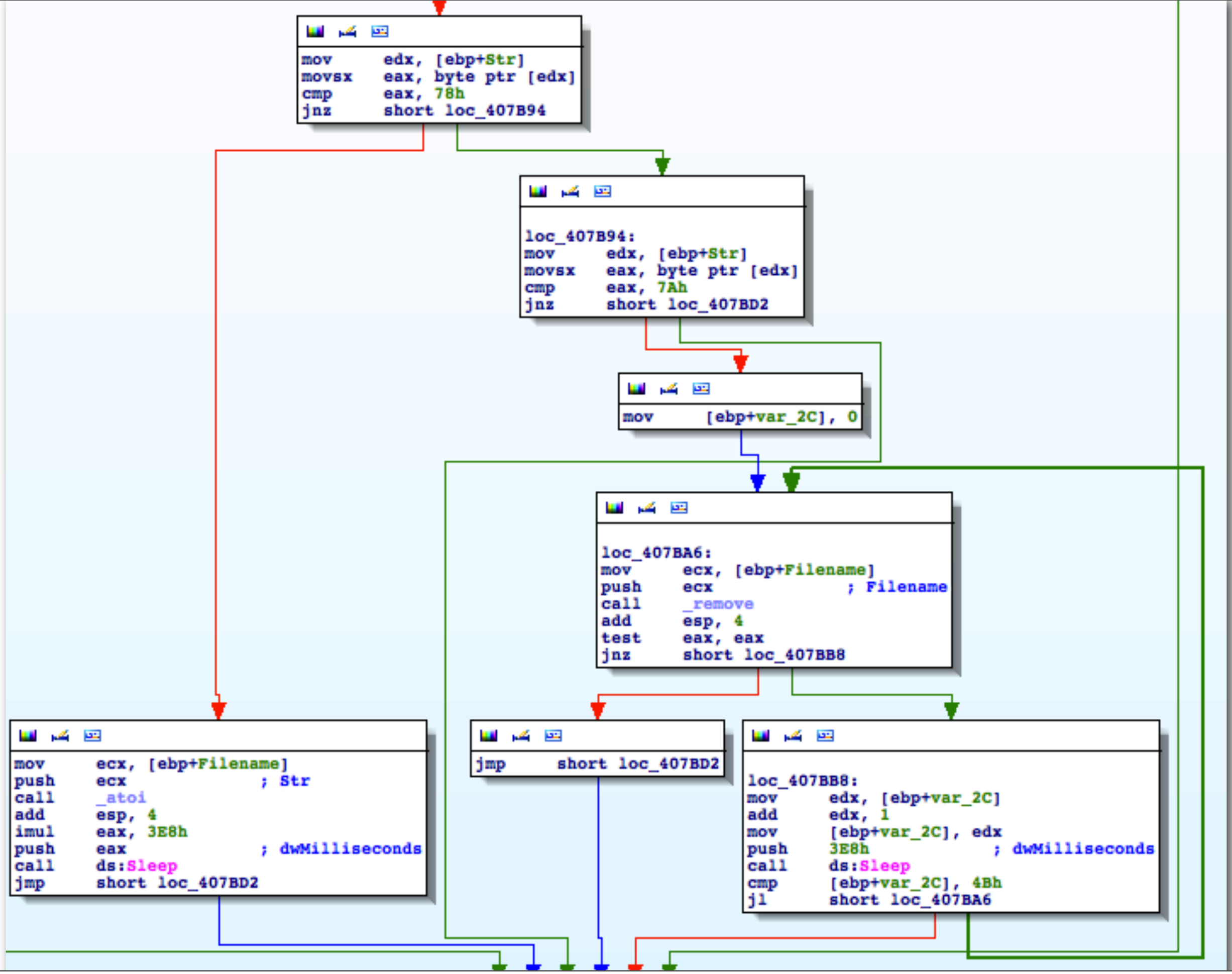
foo.exe

Analysis of Programs

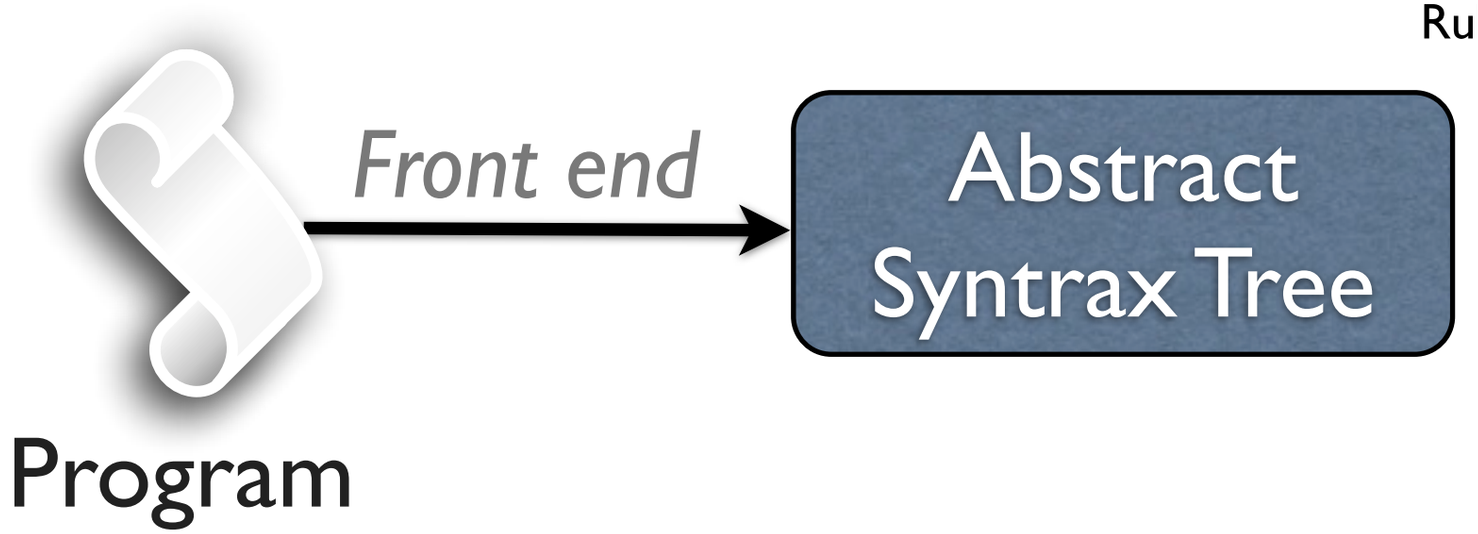
foo.exe

```
0000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00   MZ..............
0010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00   ........@.......
0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0030  00 00 00 00 00 00 00 00 00 00 00 00 D0 00 00 00   ................
0040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68   ........!..L.!Th
0050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F   is program canno
0060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20   t be run in DOS
0070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00   mode....$.......
0080  E5 DD 04 08 A1 BC 6A 5B A1 BC 6A 5B A1 BC 6A 5B   ......j[..j[..j[
0090  2F B4 35 5B A3 BC 6A 5B A1 BC 6B 5B 3A BC 6A 5B   /.5[..j[..k[:.j[
00A0  22 B4 37 5B B0 BC 6A 5B F5 9F 5A 5B AB BC 6A 5B   ".7[..j[..Z[..j[
00B0  66 BA 6C 5B A0 BC 6A 5B 52 69 63 68 A1 BC 6A 5B   f.l[..j[Rich..j[
00C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00D0  50 45 00 00 4C 01 05 00 1F 5A A0 49 00 00 00 00   PE..L....Z.I....
00E0  00 00 00 00 E0 00 0F 01 0B 01 06 00 00 5E 00 00   .............^..
```

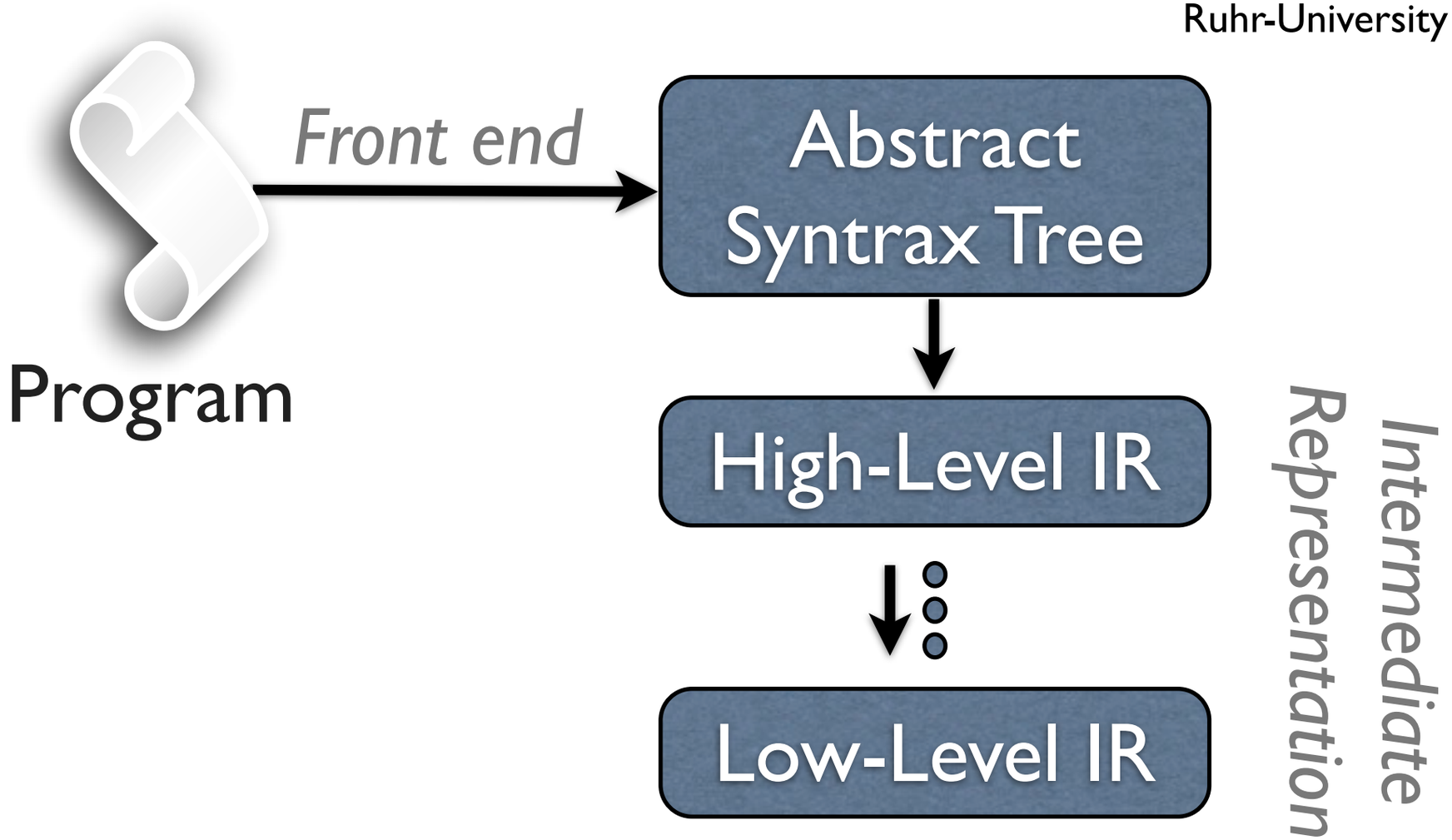
```
.text:00407B74                  cmp     eax, 78h
.text:00407B77                  jnz     short loc_407B94
.text:00407B79                  mov     ecx, [ebp+Filename]
.text:00407B7C                  push    ecx                     ; Str
.text:00407B7D                  call    _atoi
.text:00407B82                  add     esp, 4
.text:00407B85                  imul    eax, 3E8h
.text:00407B8B                  push    eax                     ; dwMilliseconds
.text:00407B8C                  call    ds:Sleep
.text:00407B92                  jmp     short loc_407BD2
.text:00407B94 ; ---------------------------------------------------------------------------
.text:00407B94
.text:00407B94 loc_407B94:                                      ; CODE XREF: WinMain(x,x,x,x)+87|j
.text:00407B94                  mov     edx, [ebp+Str]
.text:00407B97                  movsx   eax, byte ptr [edx]
.text:00407B9A                  cmp     eax, 7Ah
.text:00407B9D                  jnz     short loc_407BD2
.text:00407B9F                  mov     [ebp+var_2C], 0
.text:00407BA6
.text:00407BA6 loc_407BA6:                                      ; CODE XREF: WinMain(x,x,x,x)+E0↓j
.text:00407BA6                  mov     ecx, [ebp+Filename]
.text:00407BA9                  push    ecx                     ; Filename
.text:00407BAA                  call    _remove
.text:00407BAF                  add     esp, 4
.text:00407BB2                  test    eax, eax
.text:00407BB4                  jnz     short loc_407BB8
.text:00407BB6                  jmp     short loc_407BD2
.text:00407BB8 ; ---------------------------------------------------------------------------
.text:00407BB8
.text:00407BB8 loc_407BB8:                                      ; CODE XREF: WinMain(x,x,x,x)+C4|j
.text:00407BB8                  mov     edx, [ebp+var_2C]
.text:00407BBB                  add     edx, 1
.text:00407BBE                  mov     [ebp+var_2C], edx
.text:00407BC1                  push    3E8h                    ; dwMilliseconds
.text:00407BC6                  call    ds:Sleep
.text:00407BCC                  cmp     [ebp+var_2C], 4Bh
.text:00407BD0                  jl      short loc_407BA6
.text:00407BD2
.text:00407BD2 loc_407BD2:                                      ; CODE XREF: WinMain(x,x,x,x)+62|j
.text:00407BD2                                                  ; WinMain(x,x,x,x)+7C|j ...
.text:00407BD2                  call    sub_407980
.text:00407BD7                  cmp     eax, 0FFFFFFFFh
.text:00407BDA                  jnz     short loc_407C51
.text:00407BDC                  mov     eax, lpExistingFileName
.text:00407BE1                  push    eax                     ; lpString
```


mov edx, [ebp+Str]
movsx eax, byte ptr [edx]
cmp eax, 78h
jnz short loc_407B94
loc_407B94:
mov edx, [ebp+Str]
movsx eax, byte ptr [edx]
cmp eax, 7Ah
jnz short loc_407BD2
mov [ebp+var_2C], 0
loc_407BA6:
mov ecx, [ebp+Filename]
push ecx ; Filename
call _remove
add esp, 4
test eax, eax
jnz short loc_407BB8
mov ecx, [ebp+Filename]
push ecx ; Str
call _atoi
add esp, 4
imul eax, 3E8h
push eax ; dwMilliseconds
call ds:Sleep
jmp short loc_407BD2
jmp short loc_407BD2
loc_407BB8:
mov edx, [ebp+var_2C]
add edx, 1
mov [ebp+var_2C], edx
push 3E8h ; dwMilliseconds
call ds:Sleep
cmp [ebp+var_2C], 4Bh
jl short loc_407BA6

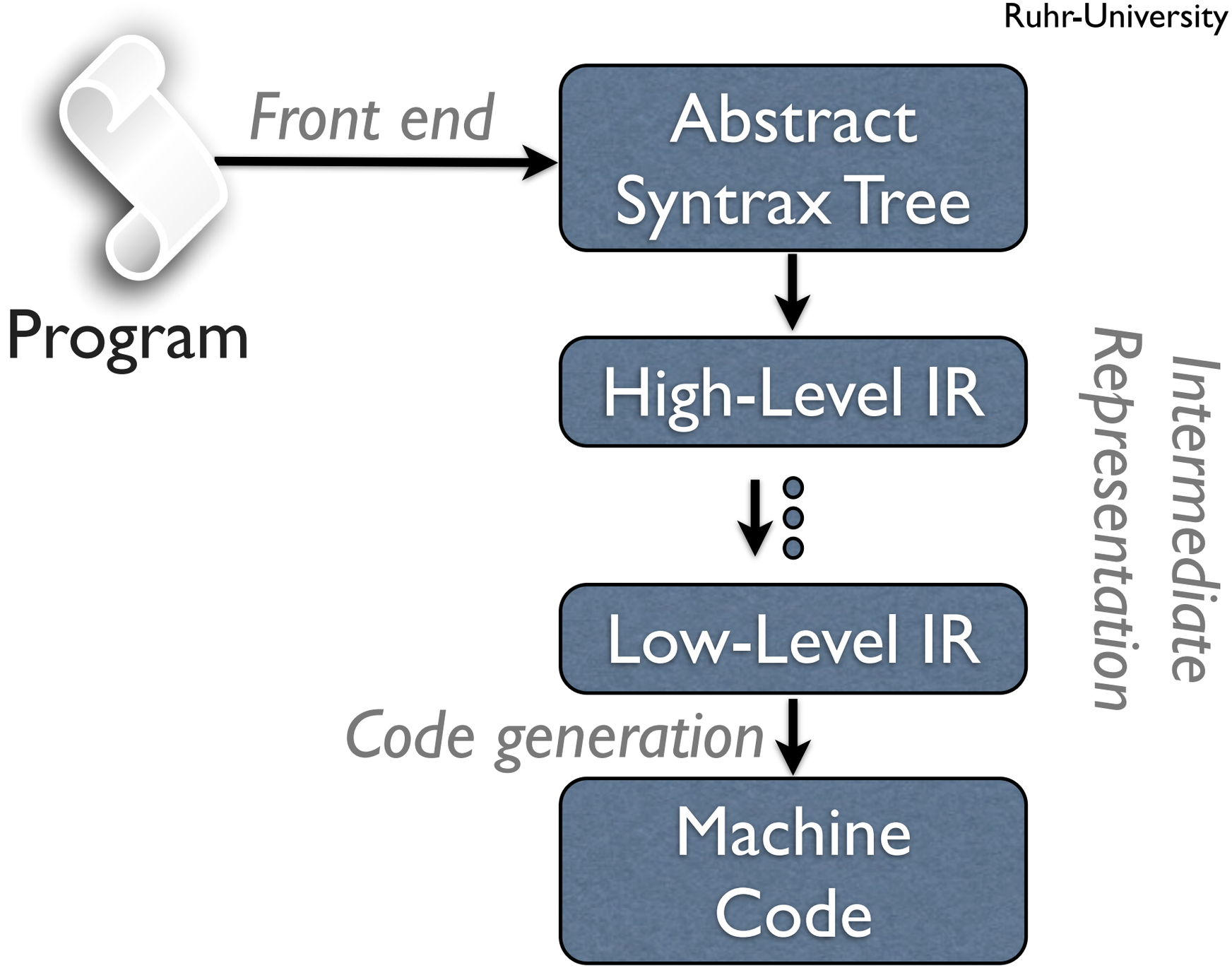
Compiler

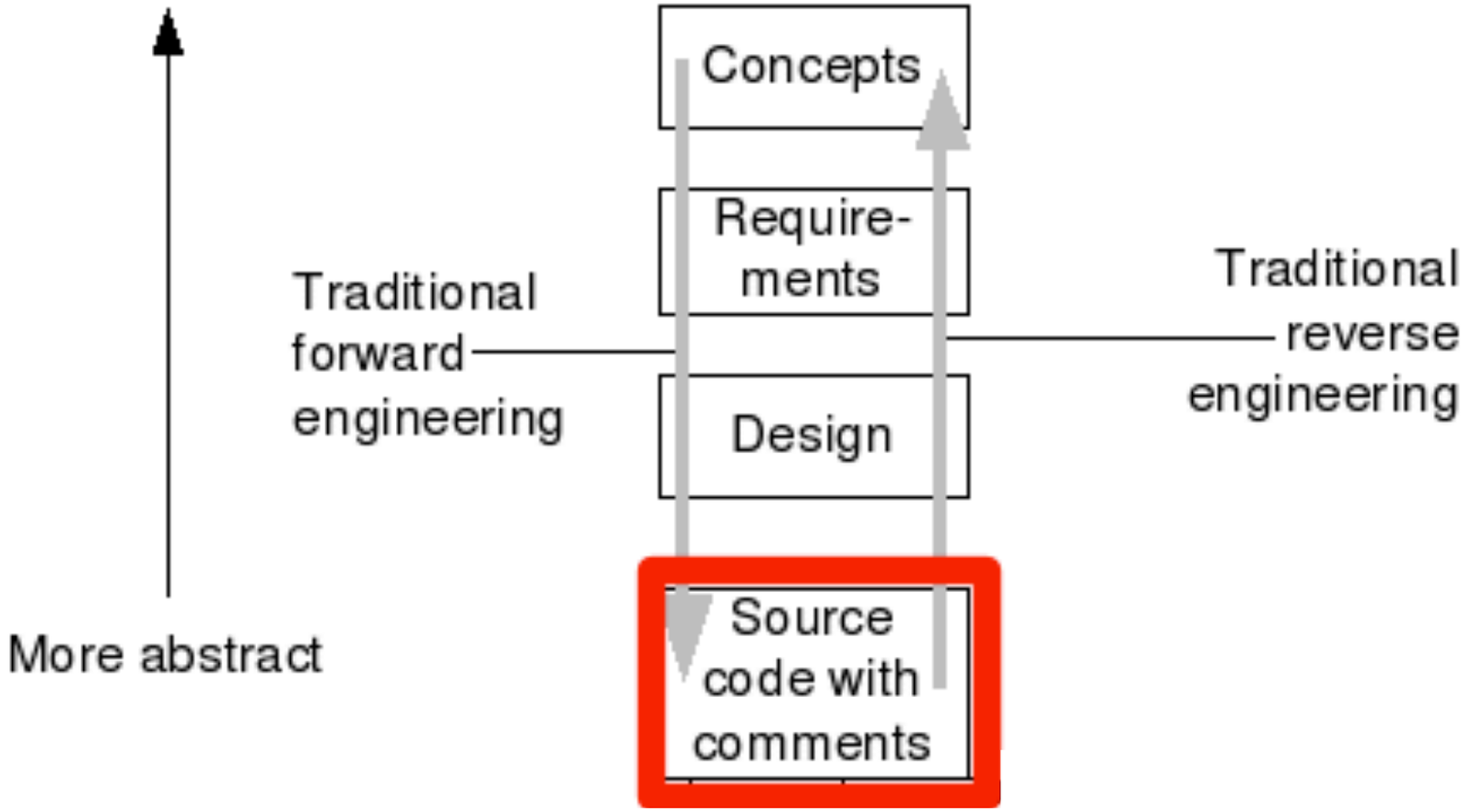


Compiler

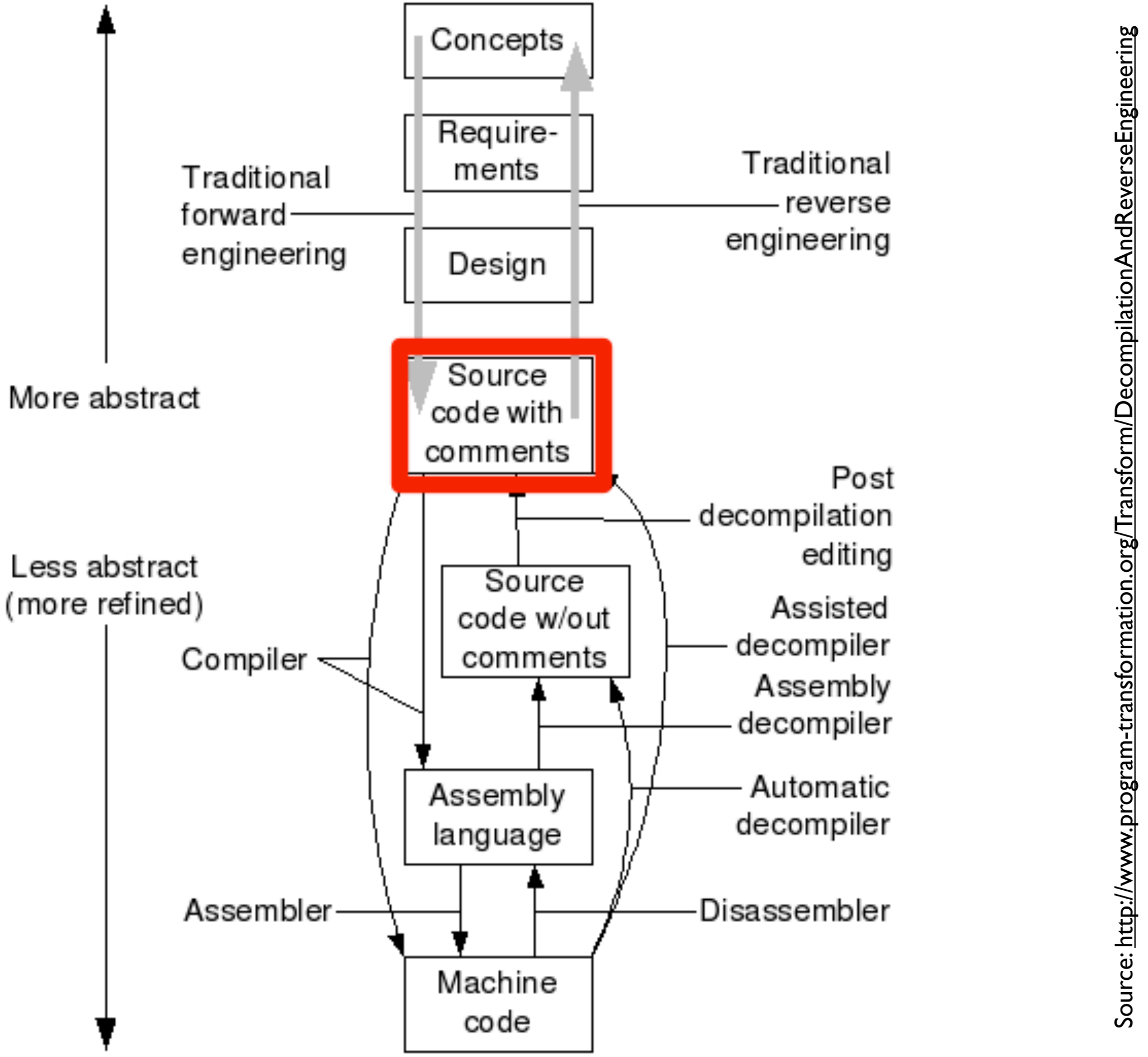

Front end
Abstract Syntrax Tree
Program
High-Level IR
Low-Level IR
Intermediate Representation

Compiler

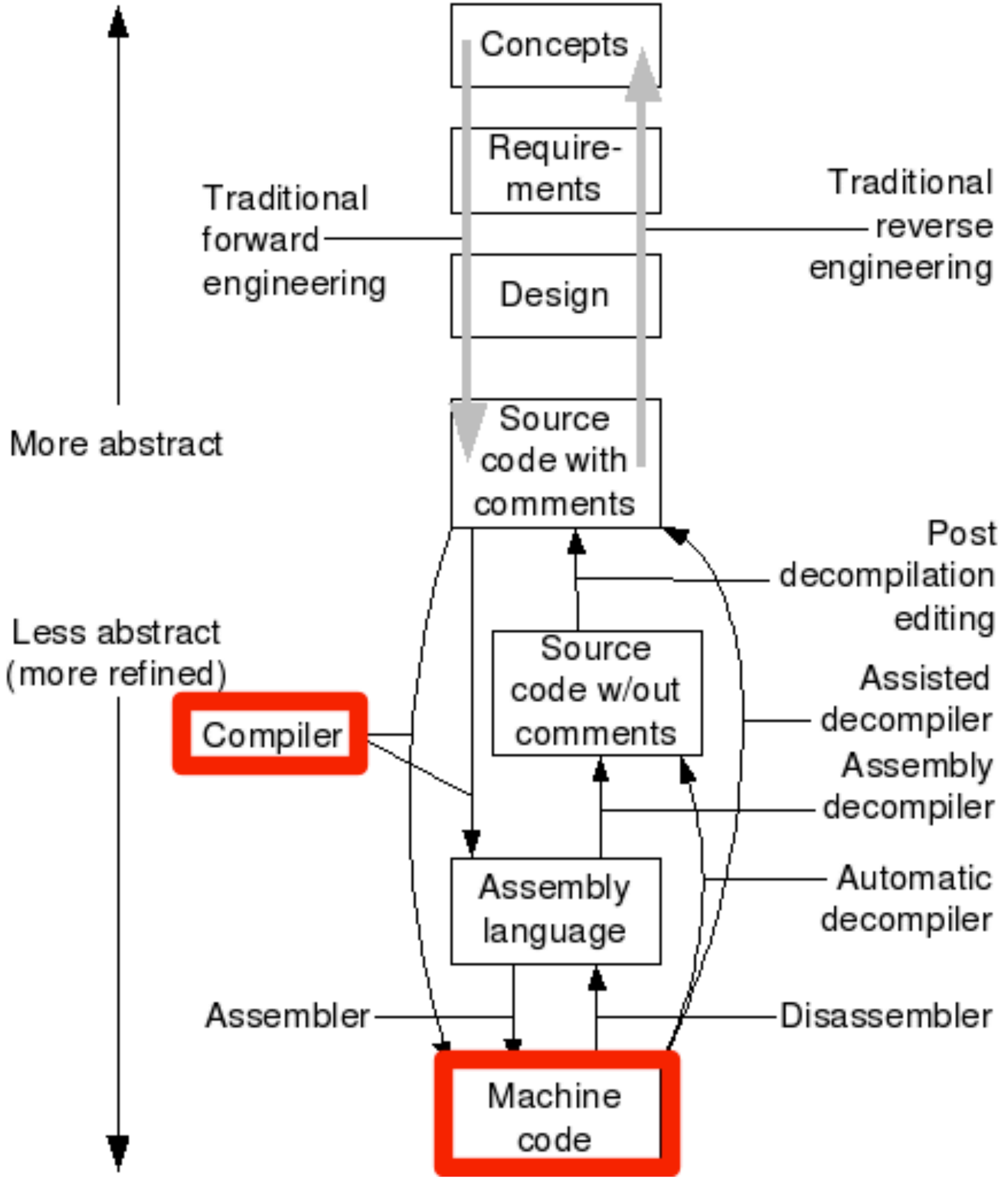

Front end
Abstract Syntrax Tree
Program
High-Level IR
Intermediate Representation
Low-Level IR
Code generation
Machine Code



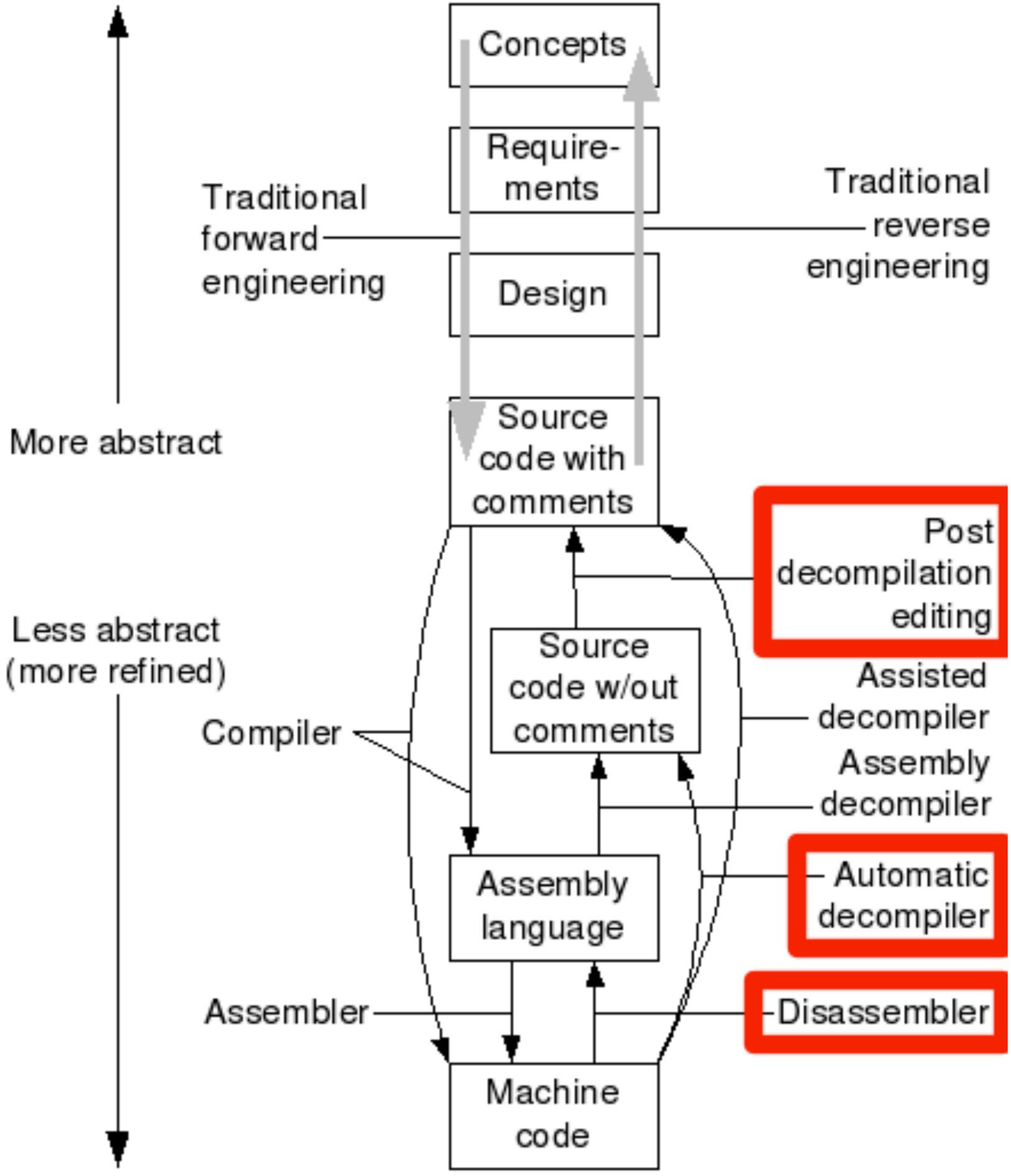
Source: http://www.program-transformation.org/Transform/DecompilationAndReverseEngineering



Source: http://www.program-transformation.org/Transform/DecompilationAndReverseEngineering



Source: http://www.program-transformation.org/Transform/DecompilationAndReverseEngineering



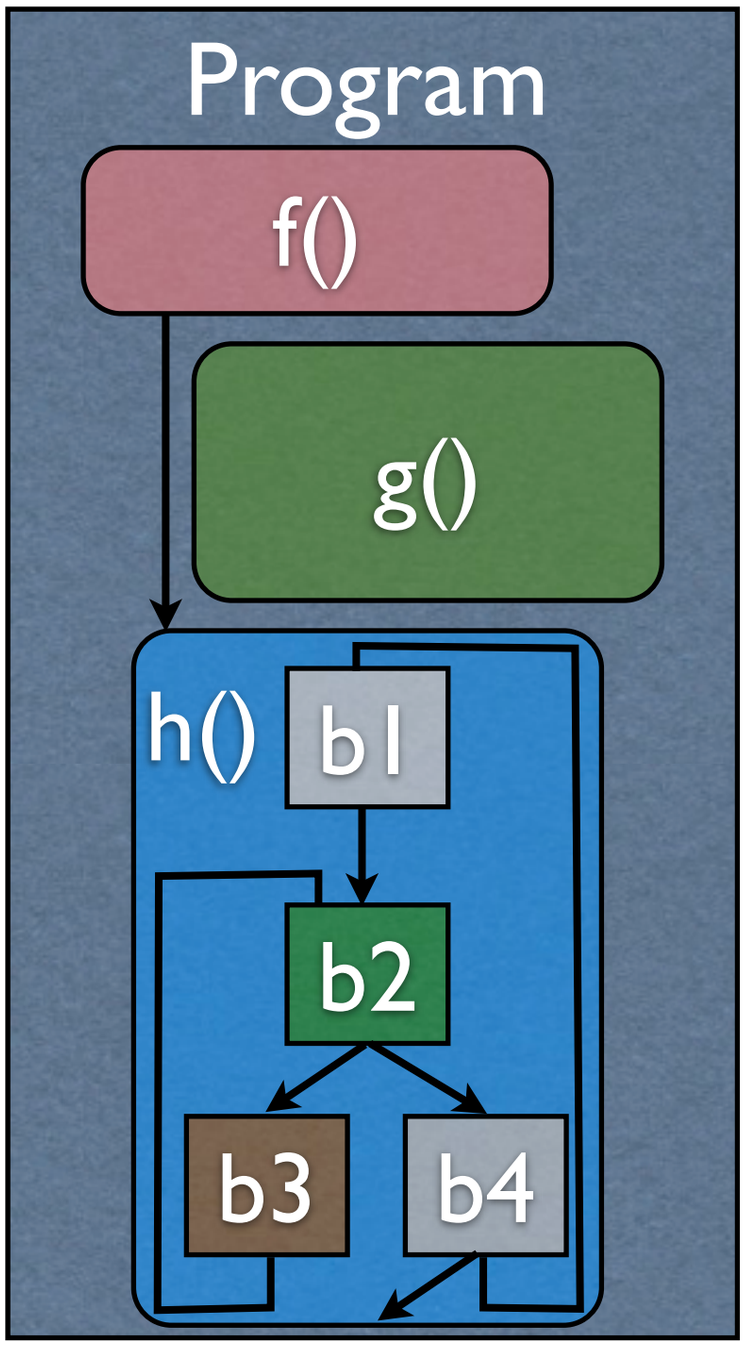
Source: http://www.program-transformation.org/Transform/DecompilationAndReverseEngineering

Basic Terms and Pitfalls

Basic Terms

- *Basic block*: maximal sequence of consecutive instructions with two properties:
  - flow of control only enters at the beginning
  - flow of control can only leave at the end (no halting or branching except at end of block)
- *Flow graph* G(V, E):
  - Vertices are basic blocks
  - Edge $b_i \rightarrow b_j$ iff b_i can follow b_j immediately in execution

Basic Terms



- Program consists of several *procedures*
  - f() calls h()
- Produce consists of several *basic blocks*
  - "Atomic" units of a program

Generated Code

- When translating a program, compilers often add their own piece of code, for example
  - Start-up code to include libraries / initialize runtime
  - Code to save processor registers before jumping to subroutine
- There is no such code in high-level representation
- Also characteristic traces left by compiler
  - Specific translation of code constructs

Idioms

- A *idiom* is some kind of programming trick / typical piece of code used by programmers
  - shift left (1 bit) is equal to multiplication by 2
  - Add two dwords
    - First add lower words
    - Then add higher words, including carry-over
- You need to know idioms to be able to understand and recognize them

Obfuscation

- Code is intentionally more complex
  - Hide semantic from human analysts
  - Often used to complicate reverse engineering
- Commonly found in *malicious software*, e.g., malware using some kind of packer
  - Actual malware binary is packed payload
  - Small stub at beginning of sample to decode payload
  - Complete binary only available during runtime

Self-Modifying Code

- A program can modify its own code during runtime
  - Arbitrary changes possible
  - Can depend on user input
  - Makes analysis harder
- Commonly found in malware samples
  - Annoy human analysts
  - Bypass automated detection systems

Limitations

- Reverse engineering is a hard problem
  - Can not be fully automated
    - Tools can help us to understand the code/data, but a human analyst needs to interpret the results
    - Related to *Halting Problem* in theoretical computer science

Halting Problem

- We want: program *H*
  - *Input*: arbitrary program *P*, arbitrary input *I*
  - *Output*:
    - `True` if P(I) eventually terminates
    - `False` if P(I) never terminates
- Theorem: There is no such program *H* (proof by Turing)
  - Halting problem is *undecidable* over Turing machines

Summary

- We introduce several techniques from the area of *program analysis* in this lecture
  - Both static and dynamic techniques
  - Present basic principles and techniques
- Enables us to analyze and understand a given piece of code (e.g., binary code or some other input)
- Learn more about practical aspects in exercises
  - *Learning by doing!*

Questions?

Contact:
Prof. Thorsten Holz
thorsten.holz@rub.de
@thorstenholz on Twitter

More information:
http://emma.rub.de
(soon: http://syssec.rub.de)
https://moodle.rub.de

Sources

- Lecture *Software Reverse Engineering* at University of Mannheim, spring term 2010 (Ralf Hund, Carsten Willems and Felix Freiling)
- Chikofsky and Cross: *Reverse Engineering and Design Recovery: A Taxonomy*, IEEE Software 7(1):13-17, 1990
- Van Emmerik - http://www.program-transformation.org/