COMP4901O Competitive Programming in Cybersecurity II

SOMP1010: angr Management

Crack binaries with righteous indignation!

TrebledJ

May 6, 2022

>FIREBIRD CTF
TEAM

Code of Ethics

- The exercises for the course should be attempted ONLY INSIDE THE SECLUDED LAB ENVIRONMENT documented or provided. Please note that most of the attacks described in the slides would be ILLEGAL if attempted on machines that you do not have explicit permission to test and attack. The university, course lecturer, lab instructors, teaching assistants and the Firebird CTF team assume no responsibility for any actions performed outside the secluded lab.
- Do not intentionally disrupt other students who are working on the challenges or disclose private information you found on the challenge server (e.g. IP address of other students).
- When solving CTF challenges or hunting angr documentation for some obscure class, it is imperative that you remain calm. Do not destroy the challenge server or make stupid decisions out of frustration or angr!

Today's Talk

- Reverse!
- Less focus on angr's API
- More focus on the big picture
- More memes
- Unrelated to the GUI tool

How to Manage Your angr

Introduction

What is this angr?

Highly modularised and customisable framework for...

- Symbolic execution
- Binary analysis (control flow, value set, dependencies)
- Reversing binaries with righteous indignation

Repo: https://github.com/angr/angr
Docs (Book-like): https://docs.angr.io/
API: https://api.angr.io/

See Slide 72 for angr setup tips!

Motivation

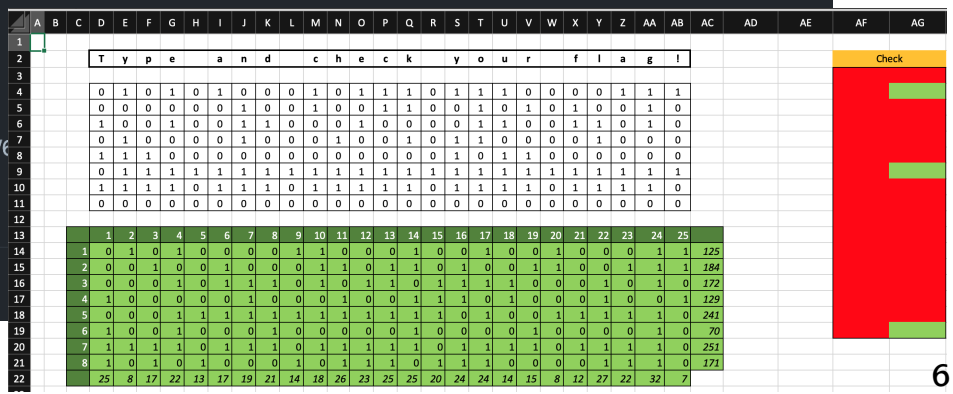
逐步拾回自己 / Step by Step

Summary

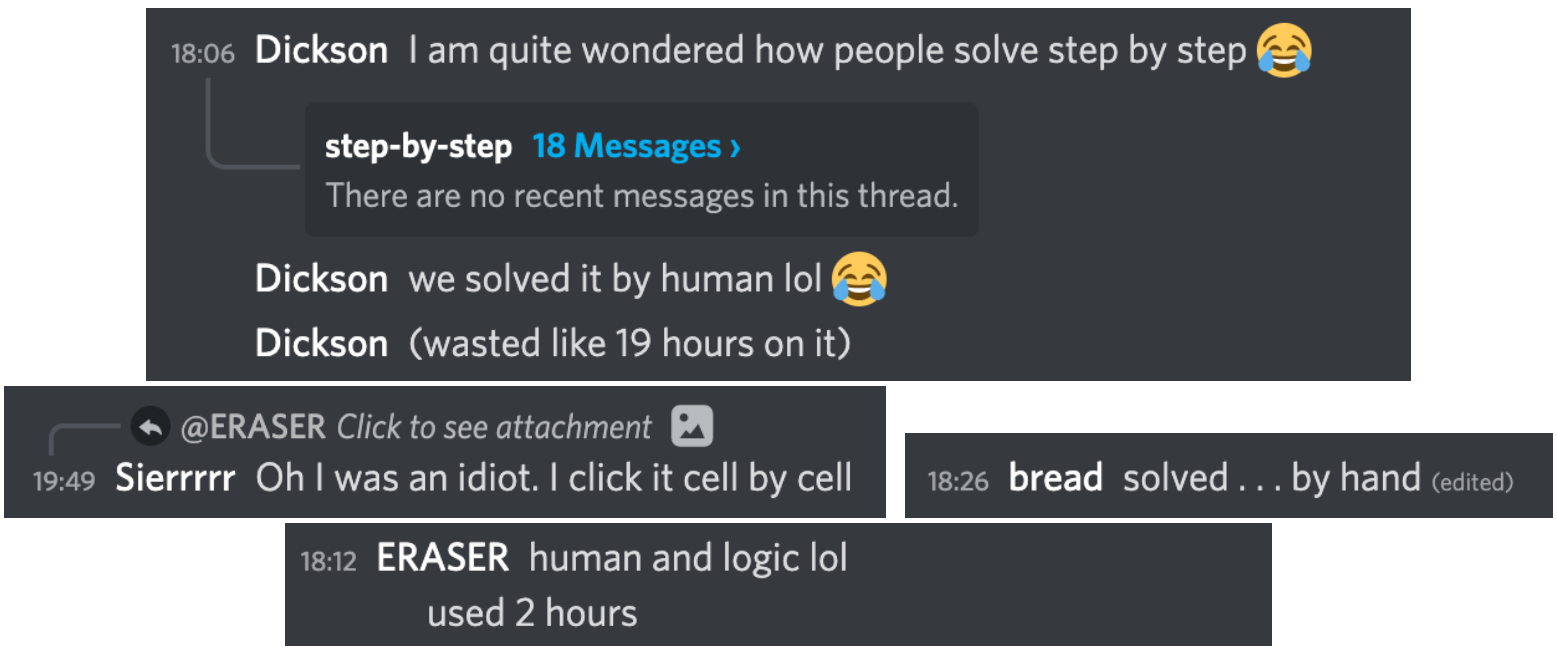
- **Thumbnail:**
- **Song:** https://www.youtube.com/watch?v=-DZd0CF13nI
- **Author:** cire_meat_pop
- **Categories:** Reverse, Misc, ★☆☆☆☆
- **Points:** 100
- **Solves:** 68/234 (Secondary: 19/103, Tertiary: 25/65, Open: 21/6

Description

Retrieve the flag step by step.



Motivation


18:06 Dickson I am quite wondered how people solve step by step 😂
step-by-step 18 Messages ›
There are no recent messages in this thread.
Dickson we solved it by human lol 😂
Dickson (wasted like 19 hours on it)
@ERASER Click to see attachment
19:49 Sierrrrr Oh I was an idiot. I click it cell by cell
18:26 bread solved . . . by hand (edited)
18:12 ERASER human and logic lol
used 2 hours

Back to the Basics

Early Stages of Reverse-Engineering Grief and Trauma

Symbolic Execution – General Idea

Code

```
void hello(uint8_t c) {
    uint8_t win;
    uint32_t a = c;
    a = a*2 + 0xdeadbeef;
    if (a >= 0 && a < 0xbaad)
        win = 1;
    else
        win = 0;
}
```

Concrete

```
c = 0xf0

a = 0x000000f0
a = 0xdeadc0cf



win = 0
```

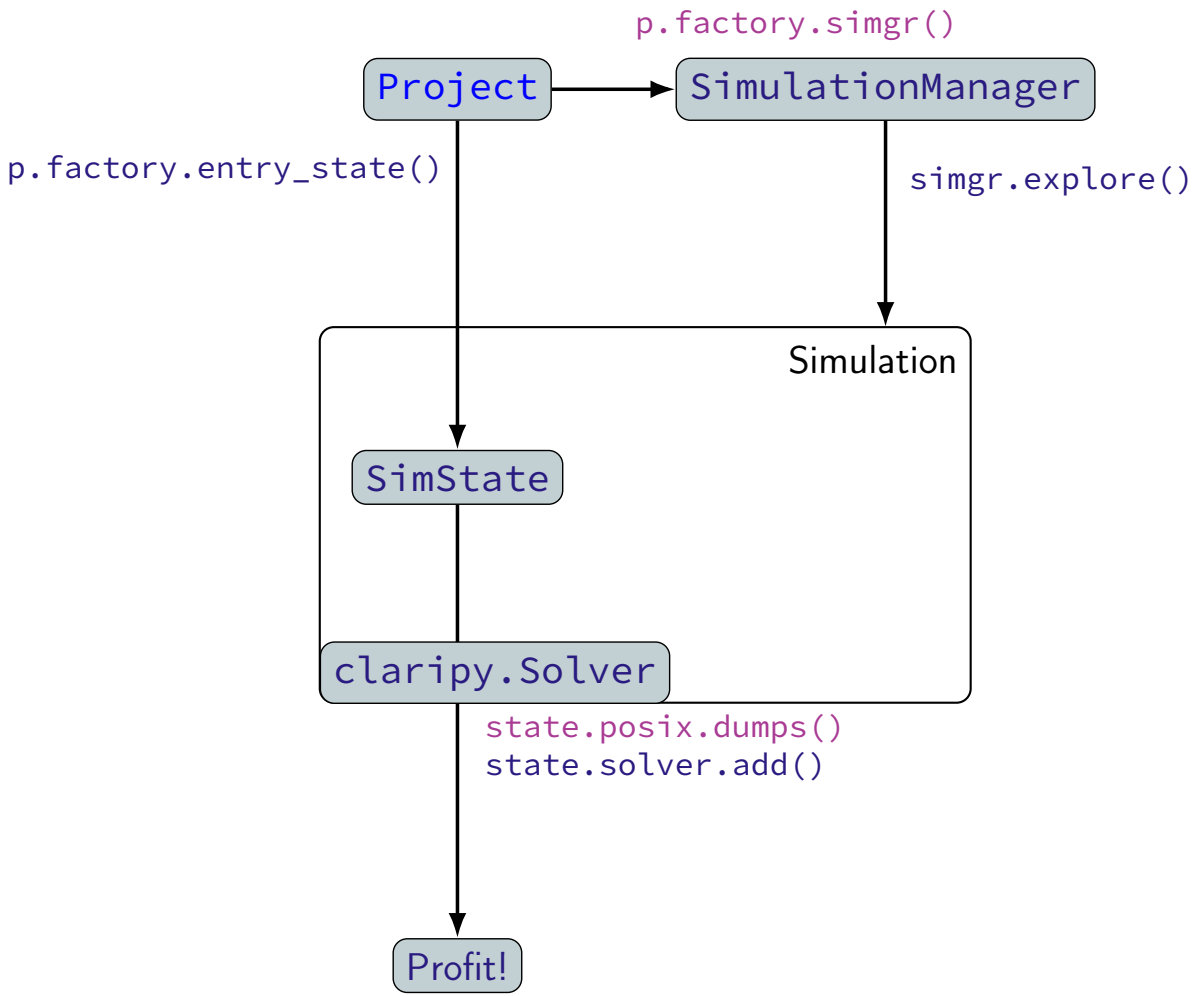
Symbolic

```
c is an 8-bit input var

a_0 = ZeroExt(24, c)
a = a_0 * 2 + 0xdeadbeef
win = If(And(UGE(a, 0),
             ULT(a, 0xbaad)),
         1,
         0)
```

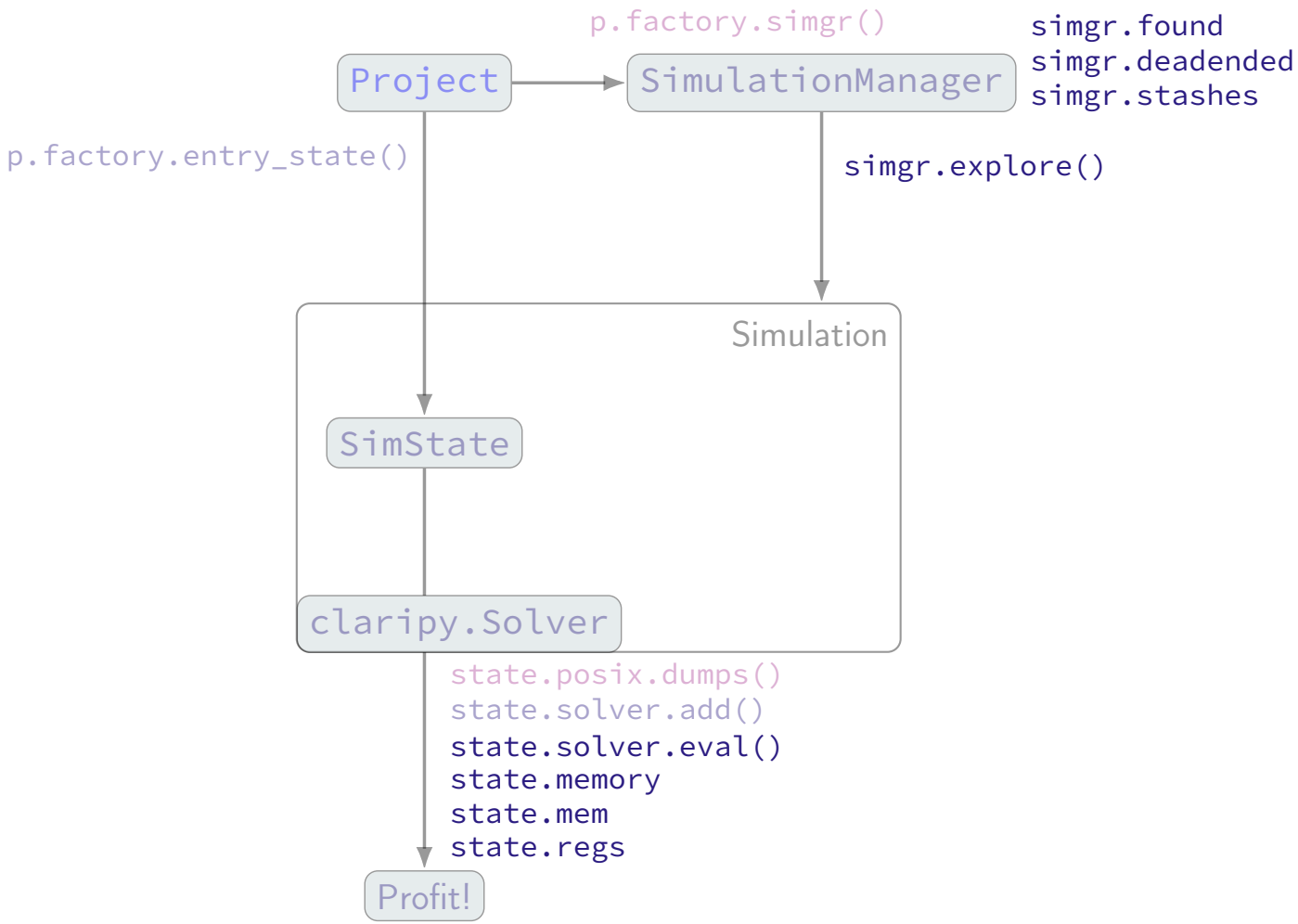
angr Level 1 – Constrain your angr!

```
import angr
from claripy import *

p = angr.Project('./level1')
flag_bvs = [BVS(f'f_{i}', 8) for i in range(32)]
state = p.factory.entry_state(
            stdin=Concat(*flag_bvs, BVV('\n', 8)))

for bv, ch in zip(flag_bvs, b'flag{'):
    state.solver.add(bv == BVV(ch, 8))

for bv in flag_bvs:
    state.solver.add(bv >= BVV(0x20, 8))
    state.solver.add(bv <= BVV(0x7f, 8))

simgr = p.factory.simgr(state)
simgr.explore(find=0x40170f)
print(simgr.found[0].posix.dumps(0))
```

angr Level 1 – Constrain your angr!

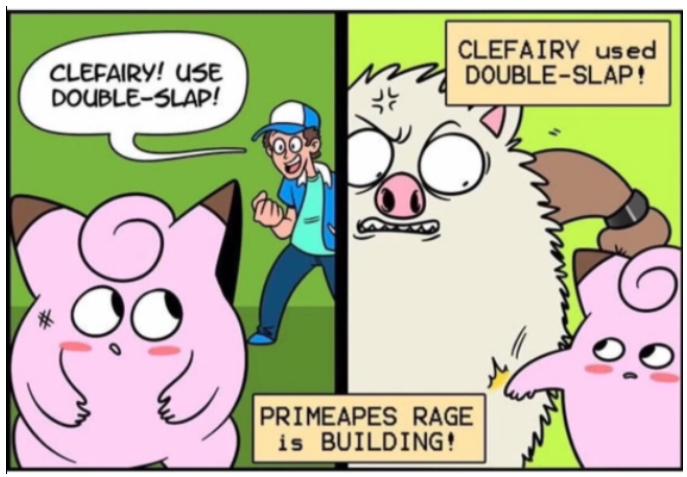


What's next?


p.factory.simgr()
simgr.found
simgr.deadended
simgr.stashes
Project
SimulationManager
p.factory.entry_state()
simgr.explore()
Simulation
SimState
claripy.Solver
state.posix.dumps()
state.solver.add()
state.solver.eval()
state.memory
state.mem
state.regs
Profit!

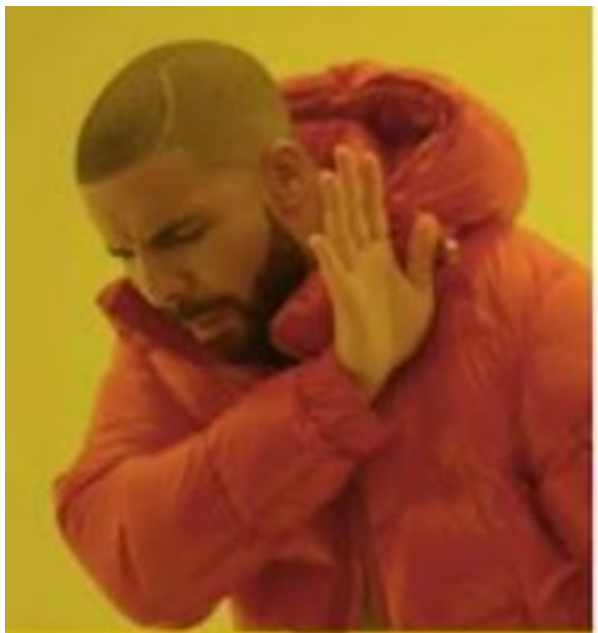
Training Your angr

Wild **PRIMEAPE** used **RAGE**!



Some Terminology

- bitvector (BV): a sequence of bits
- symbol: unknown data (Schrödinger's cat)
- concrete: known data (Schrödinger's cat outside the box)
- concretise: process of [symbol $\rightarrow$ concrete] w.r.t. constraints (opening Schrödinger's box)
- mixin: a design pattern promoting customisations (plugins)




SIMCITY GAMES


imgflip.com


SIMSTATE GAMES

SimState Concretisation

`state.solver.eval(e, cast_to=None)`

Evaluates (concretises) a bitvector and optionally casts to the type specified by `cast_to` (`int` or `bytes`).

`state.posix.dumps(fd)`

Concretises and dumps stdin (0), stdout (1), or stderr (2) as bytes.

More: https://docs.angr.io/core-concepts/solver#more-solving-methods.

A Tale of Two SimState Memory Lanes

DefaultMemory (Mixin Galore)

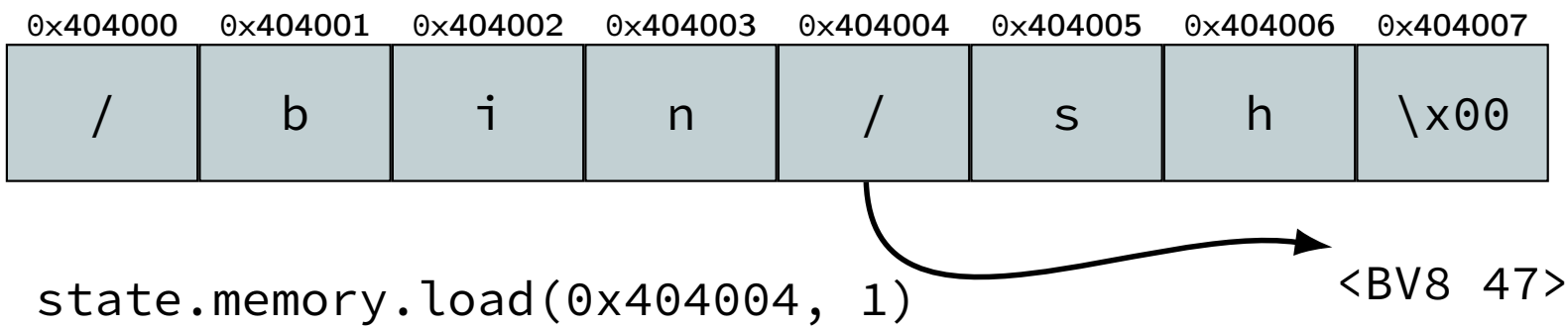
```
state.memory.store(addr: int, data: int | bytes | BV)
state.memory.load(addr: int, size: int) → BV
```

SimMemView

```
state.mem[<addr>].<type>.<resolved | concrete>
```

state.memory

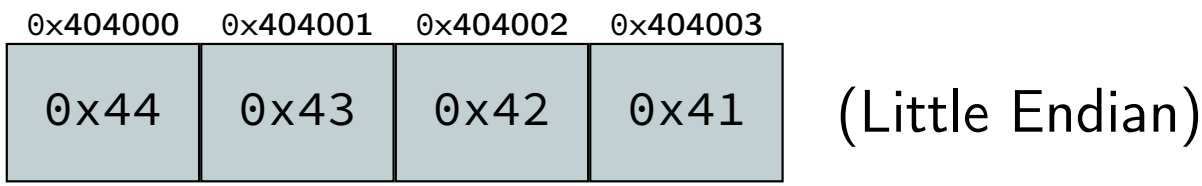
```
state.memory.store(0x404000, b'/bin/sh\x00')

bv = state.memory.load(0x404004, 4)
state.solver.eval(bv, cast_to=bytes)
→ b'/sh\x00'
```



Byte order depends on endianness! Check state.arch!
More: https://docs.angr.io/core-concepts/states#low-level-interface-for-memory.

state.mem

```
state.mem[0x404000].uint32_t = 0x41424344

state.mem[0x404000].uint16_t.resolved
→ <BV16 0x4344>

state.mem[0x404000].uint16_t.concrete
→ 0x4344
```



Byte order depends on endianness! Check `state.arch`!

state.mem

```
>>> state.mem[0x404000]
<<untyped> <unresolvable> at 0x404000>
>>> _.
_.CharT                 _.init_state(           _.resolved              _.types
_.STRONGREF_STATE       _.int                   _.set_state(            _.uint16_t
_.array(                _.int16_t               _.set_strongref_state(  _.uint32_t
_.basic_string          _.int32_t               _.short                 _.uint64_t
_.bool                  _.int64_t               _.signed                _.uint8_t
_.byte                  _.int8_t                _.size_t                _.uintptr_t
_.char                  _.long                  _.ssize                 _.unsigned
_.concrete              _.memo(                 _.ssize_t               _.va_list
_.copy(                 _.merge(                _.state                 _.void
_.deref                 _.ptrdiff_t             _.store(                _.widen(
_.double                _.qword                 _.string                _.word
_.dword                 _.register_default(     _.struct                _.wstring
_.float                 _.resolvable            _.time_t
```

Note: some types are not available for some arches (e.g. no `int64_t` for 32-bit arches).

state.regs

```
state.regs.pc
→ <BV64 0x4011c8>
```

```
>>> state.regs.
Display all 222 possibilities? (y or n)
state.regs.STRONGREF_STATE      state.regs.fc3210           state.regs.r13w             state.regs.tag2
state.regs.ac                   state.regs.flags            state.regs.r14              state.regs.tag3
state.regs.acflag               state.regs.fpreg            state.regs.r14b             state.regs.tag4
state.regs.ah                   state.regs.fpround          state.regs.r14d             state.regs.tag5
state.regs.al                   state.regs.fptag            state.regs.r14w             state.regs.tag6
state.regs.ax                   state.regs.fpu_regs         state.regs.r15              state.regs.tag7
state.regs.bh                   state.regs.fpu_tags         state.regs.r15b             state.regs.widen(
state.regs.bl                   state.regs.fs               state.regs.r15d             state.regs.xmm0
state.regs.bp                   state.regs.fs_const         state.regs.r15w             state.regs.xmm0hq
state.regs.bph                  state.regs.ftop             state.regs.r8               state.regs.xmm0lq
state.regs.bpl                  state.regs.get(             state.regs.r8b              state.regs.xmm1
state.regs.bx                   state.regs.gs               state.regs.r8d              state.regs.xmm10
state.regs.cc_dep1              state.regs.gs_const         state.regs.r8w              state.regs.xmm10hq
state.regs.cc_dep2              state.regs.id               state.regs.r9               state.regs.xmm10lq
state.regs.cc_ndep              state.regs.idflag           state.regs.r9b              state.regs.xmm11
state.regs.cc_op                state.regs.init_state(      state.regs.r9d              state.regs.xmm11hq
state.regs.ch                   state.regs.ip               state.regs.r9w              state.regs.xmm11lq
state.regs.cl                   state.regs.ip_at_syscall    state.regs.rax              state.regs.xmm12
state.regs.cmlen                state.regs.memo(            state.regs.rbp              state.regs.xmm12hq
state.regs.cmstart              state.regs.merge(           state.regs.rbx              state.regs.xmm12lq
state.regs.copy(                state.regs.mm0              state.regs.rcx              state.regs.xmm13
state.regs.cr0                  state.regs.mm1              state.regs.rdi              state.regs.xmm13hq
state.regs.cr2                  state.regs.mm2              state.regs.rdx              state.regs.xmm13lq
state.regs.cr3                  state.regs.mm3              state.regs.register_default(  state.regs.xmm14
```

Not all completions are shown.

`simgr.explore(find, avoid, n, num_find, ...)`

Simulate until constraints are met (or until no more states could be stepped).

- `find: int | [int] | (SimState → bool) = None`
- `avoid: int | [int] | (SimState → bool) = None`
  - An address, a list of addresses, or a predicate which takes a state and returns whether it should be marked found/avoided (`return True`) or continue to be active (`return False`).
- `n: int = None`
  - Number of steps to advance. If `None`, run until other conditions are met.
- `num_find: int = 1`
  - Number of states to look for in the `'found'` stash.

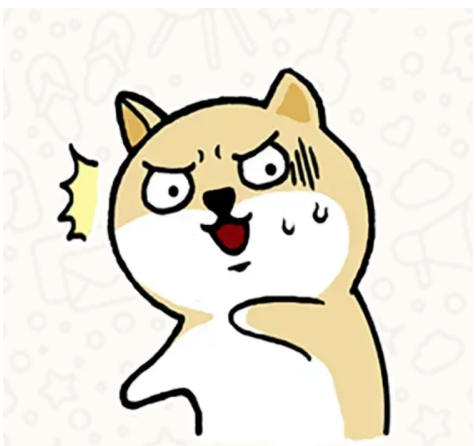
`simgr.stashes[...]`

Where are the simulated states?

- `'active'`: States here are alive and well.
- `'found'`: States here match search criteria and constraints.
- `'deadended'`: States finished executing or can't continue.
  - Possible causes: `exit(n)`, no more sat states, program crashed (invalid instruction, memory access, etc.).
  - "Why did my state deadend?" See Slide 66 to see where to start debugging!
- `'errored'`: States that terminated due to a Python exception (simulation error, claripy error, etc.).

Note: `simgr.stashes['active'] == simgr.active`.
More: https://docs.angr.io/core-concepts/pathgroups#stash-types.

SOMP1010: angr Management – Midterm



Q1 – Address?

```
if (DAT_001041c9 != '\0') {
                    /* Interesting loop... */
  lVar3 = 0;
  do {
    (&DAT_00104180)[lVar3] =
         (&DAT_00104180)[lVar3] ^ (byte)(0x750729cbe569d1dd >> (((byte)lVar3 & 7) << 3));
    lVar3 += 1;
  } while (lVar3 != 0x49);
  DAT_001041c9 = '\0';
}
```

What is the address of the secret?

(A) 0x104180

(B) 0x1041c9

(C) 0x49

(D) 0x750729cbe569d1dd

Q2 – Length?

```
if (DAT_001041c9 != '\0') {
                    /* Interesting loop... */
  lVar3 = 0;
  do {
    (&DAT_00104180)[lVar3] =
         (&DAT_00104180)[lVar3] ^ (byte)(0x750729cbe569d1dd >> (((byte)lVar3 & 7) << 3));
    lVar3 += 1;
  } while (lVar3 != 0x49);
  DAT_001041c9 = '\0';
}
```

What is the length of the secret?

(B) 0x48

Q3 – How to print?

```
simgr = p.factory.simgr()
simgr.explore(find=0x401206)
s = simgr.found[0]
```

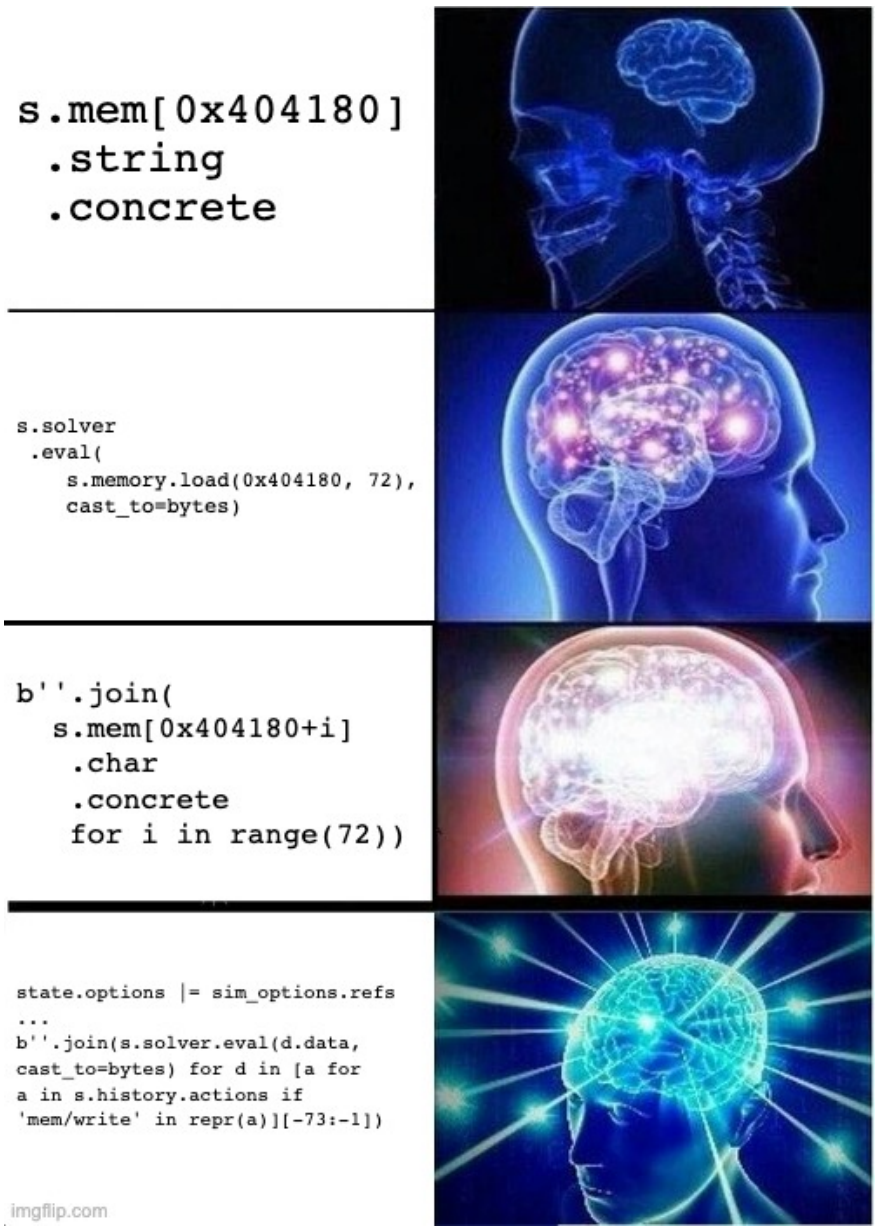
Which of the following correctly concretises and prints out the secret (assuming there are no null bytes in the 72 characters)?

(A) `s.mem[0x404180].string.concrete`

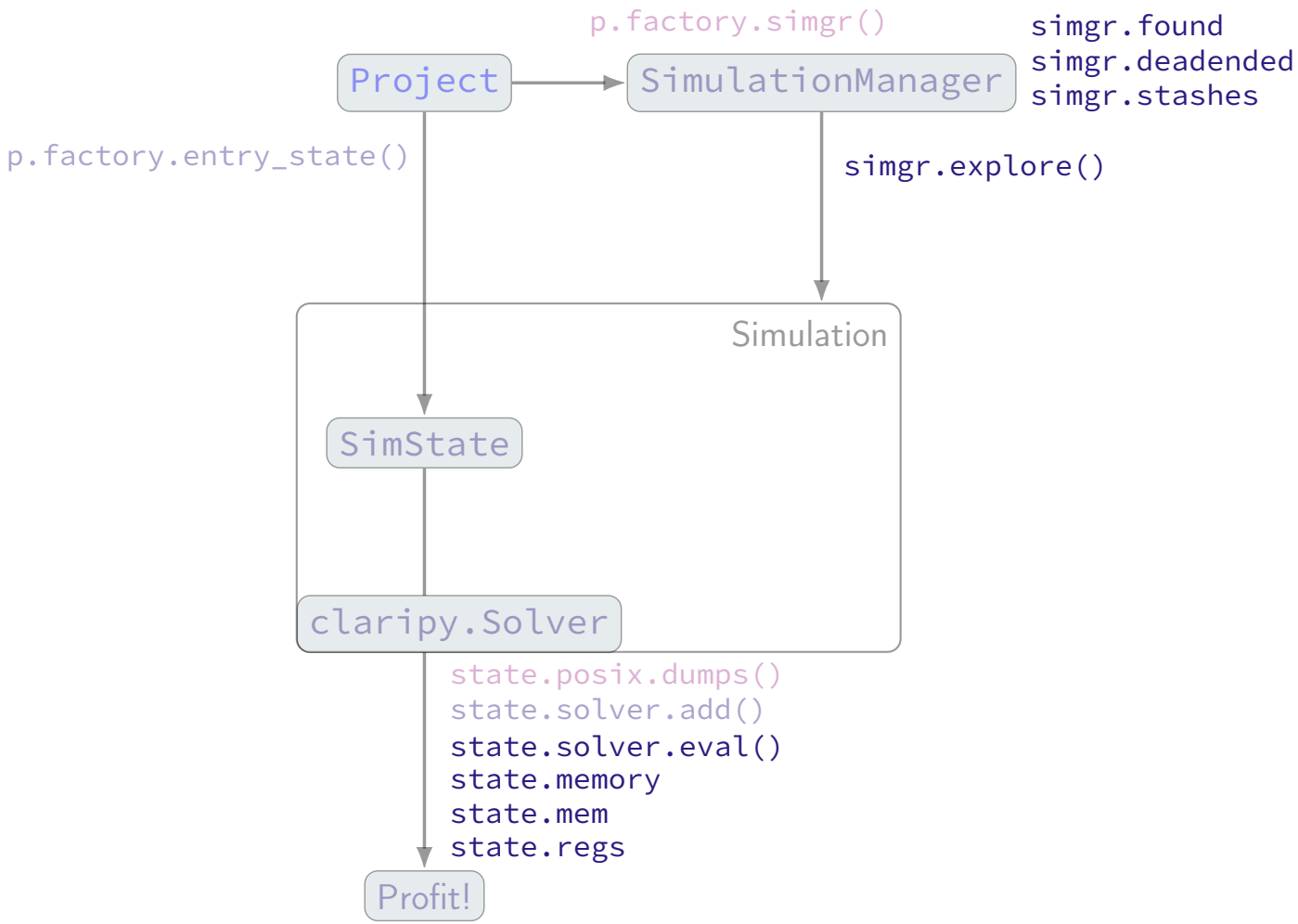
(B) `s.solver.eval(s.memory.load(0x404180, 72), cast_to=bytes)`

(C) `b''.join(s.mem[0x404180+i].char.concrete for i in range(72))`

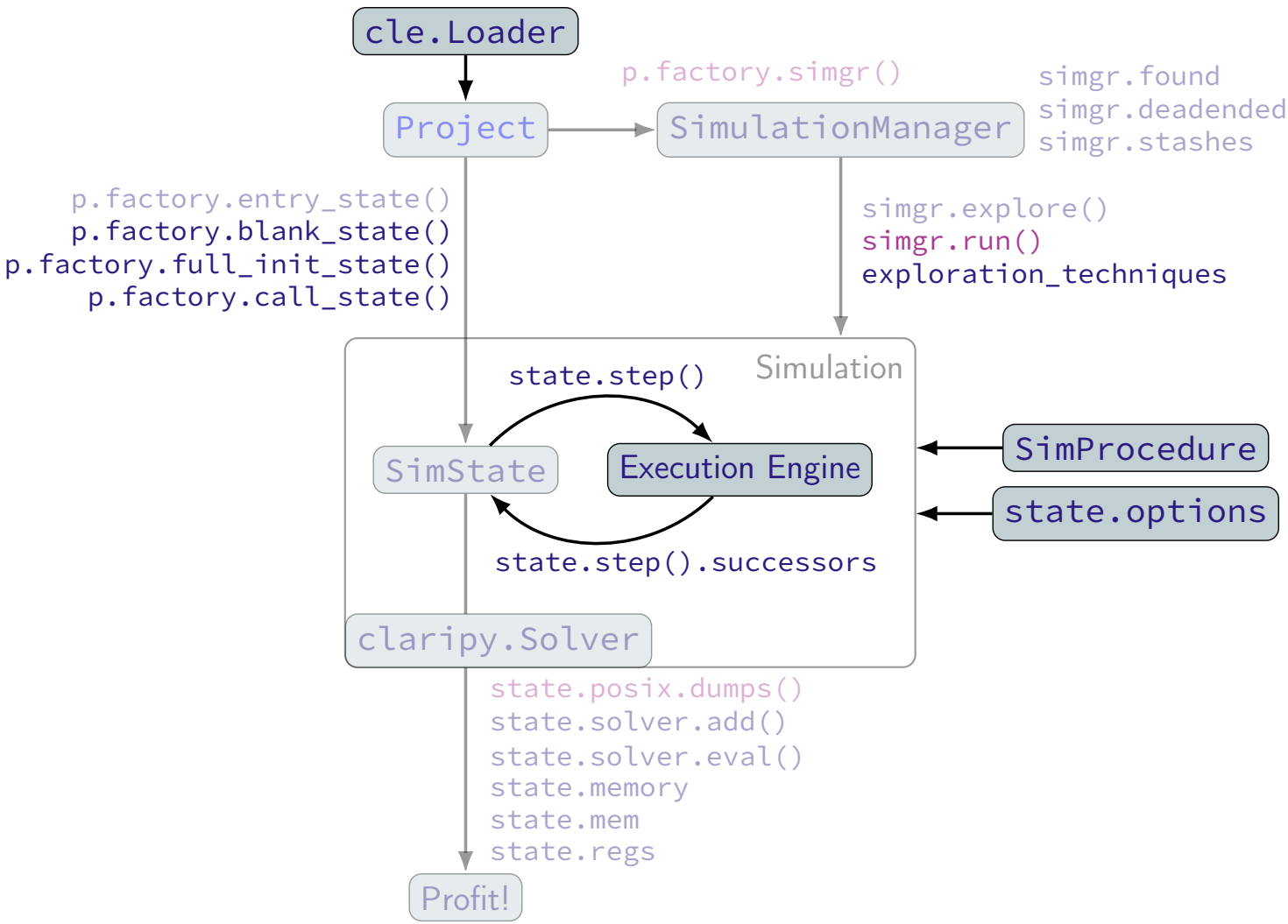
Note: tooling is a PIE binary. By default, ghidra and angr loads PIE binaries at a base address of 0x100000 and 0x400000 respectively; so by default, 0x101234 in ghidra == 0x401234 in angr.


s.mem[0x404180]
.string
.concrete
s.solver
.eval(
s.memory.load(0x404180, 72),
cast_to=bytes)
b''.join(
s.mem[0x404180+i]
.char
.concrete
for i in range(72))
state.options |= sim_options.refs
...
b''.join(s.solver.eval(d.data,
cast_to=bytes) for d in [a for
a in s.history.actions if
'mem/write' in repr(a)][-73:-1])
imgflip.com

Summary


p.factory.simgr()
simgr.found
simgr.deadended
simgr.stashes
Project
SimulationManager
p.factory.entry_state()
simgr.explore()
Simulation
SimState
claripy.Solver
state.posix.dumps()
state.solver.add()
state.solver.eval()
state.memory
state.mem
state.regs
Profit!

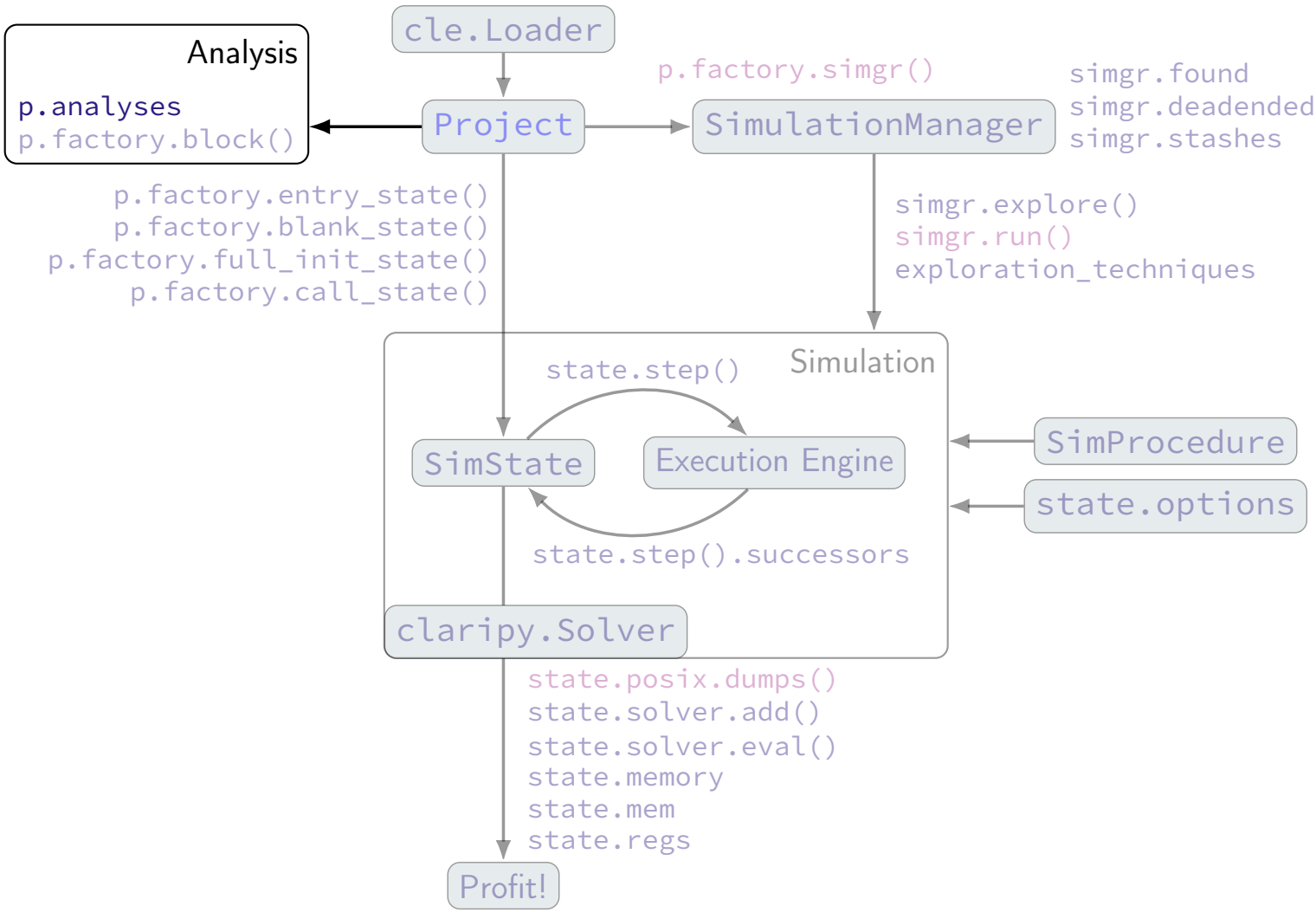
But wait– there's more!



But wait– there's more!

- `cle.Loader`: options to load x86, AVR, MIPS, etc. instructions from binary, Intel hex, etc. into an unified interface.
- State Presets: start simulating from the beginning of an arbitrary function or point in code.
- `exploration_techniques`: different ways to explore paths, keep track of states, etc.
- Hooks and `SimProcedures`: attach your own emulation code to functions/callbacks.
- `state.options`: fine-tune the simulation and solver engines.
- Execution Engine: customise which mixins are used to process instructions.

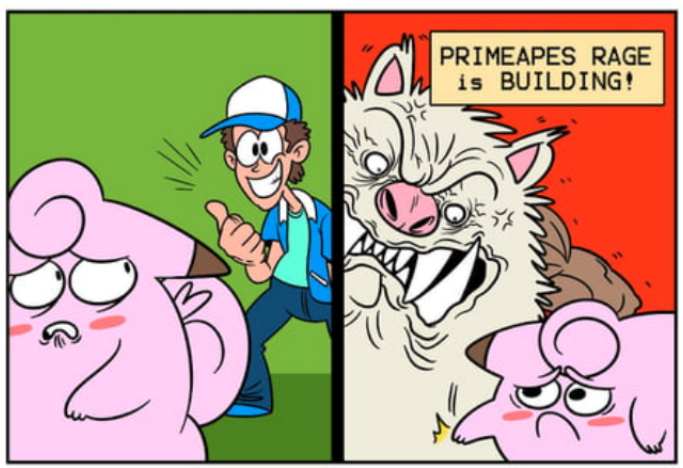
What's next?


cle.Loader
Analysis
p.analyses
p.factory.block()
Project
p.factory.simgr()
SimulationManager
simgr.found
simgr.deadended
simgr.stashes
p.factory.entry_state()
p.factory.blank_state()
p.factory.full_init_state()
p.factory.call_state()
simgr.explore()
simgr.run()
exploration_techniques
Simulation
state.step()
SimState
Execution Engine
SimProcedure
state.options
state.step().successors
claripy.Solver
state.posix.dumps()
state.solver.add()
state.solver.eval()
state.memory
state.mem
state.regs
Profit!

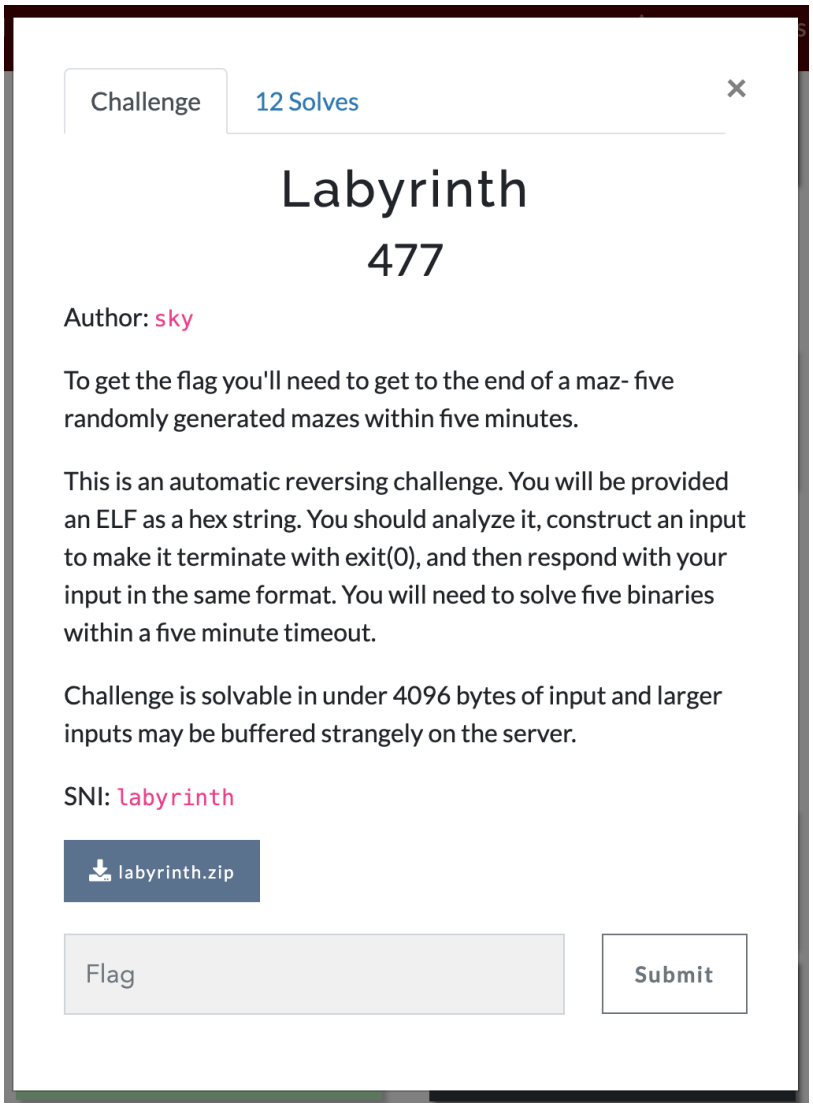
Questions?

Analysis with angr – CFGs

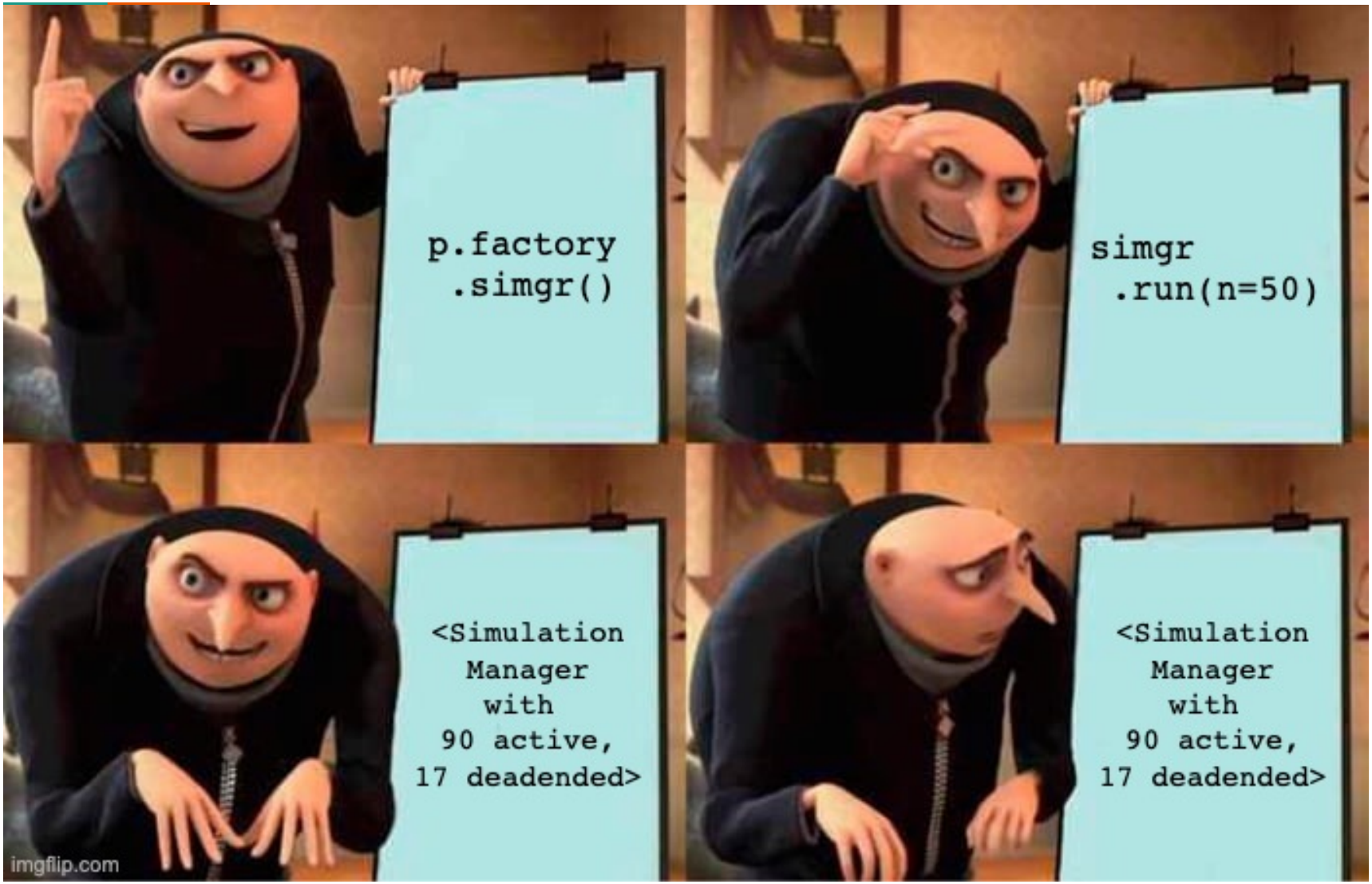
Investigating the Behavioural Economics of Machine Code



An Appetiser



Gru's Plan


p.factory
.simgr()
simgr
.run(n=50)
<Simulation
Manager
with
90 active,
17 deadended>
<Simulation
Manager
with
90 active,
17 deadended>
imgflip.com

What went wrong?

Decompile: main - (labyrinth-elf)

```
1
2  undefined8 main(void)
3
4  {
5    setvbuf(stdout,NULL,2,0);
6    setvbuf(stdin,NULL,2,0);
7    setvbuf(stderr,NULL,2,0);
8    function_133();
9    return 1;
10 }
11
```

What went wrong?

Decompile: function_133 - (labyrinth-elf)

```
1
2  void function_133(void)
3
4  {
5    uint local_c;
6
7    local_c = 0;
8    __isoc99_scanf(&DAT_00119004,&local_c);
9    local_c ^= 0x5e371ffe;
10   if (local_c == 0x178) {
11     function_376();
12     return;
13   }
14                     /* WARNING: Subroutine does not return */
15   exit(1);
16 }
17
```

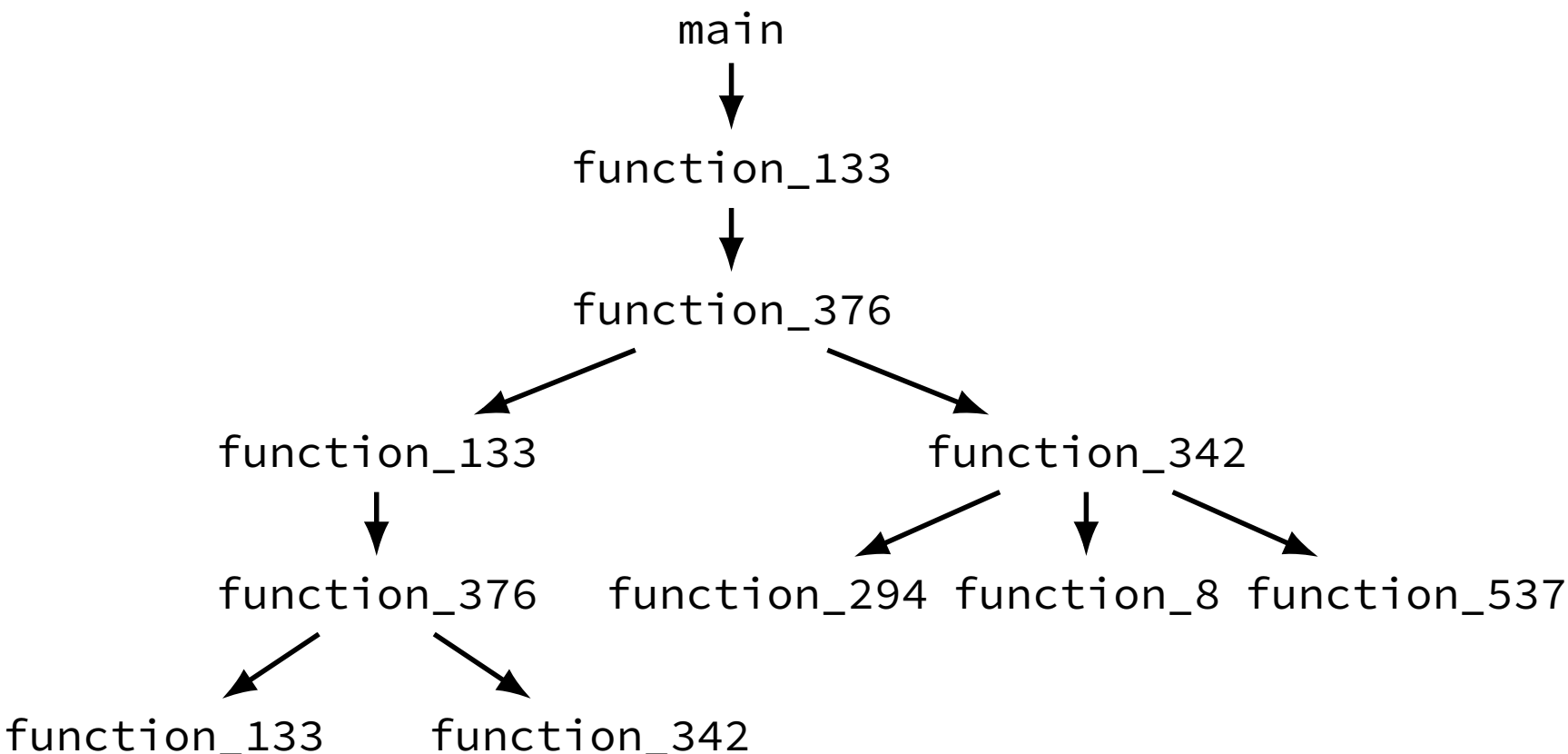
What went wrong?

Decompile: function_376 - (labyrinth-elf)

```
1
2  void function_376(void)
3
4  {
5    uint local_c;
6
7    local_c = 0;
8    __isoc99_scanf(&DAT_00119004,&local_c);
9    local_c ^= 0xef2bbe35;
10   if (local_c == 0x85) {
11     function_133();
12   }
13   else {
14     if (local_c != 0x156) {
15                     /* WARNING: Subroutine does not return */
16       exit(1);
17     }
18     function_342();
19   }
20   return;
21 }
22
```

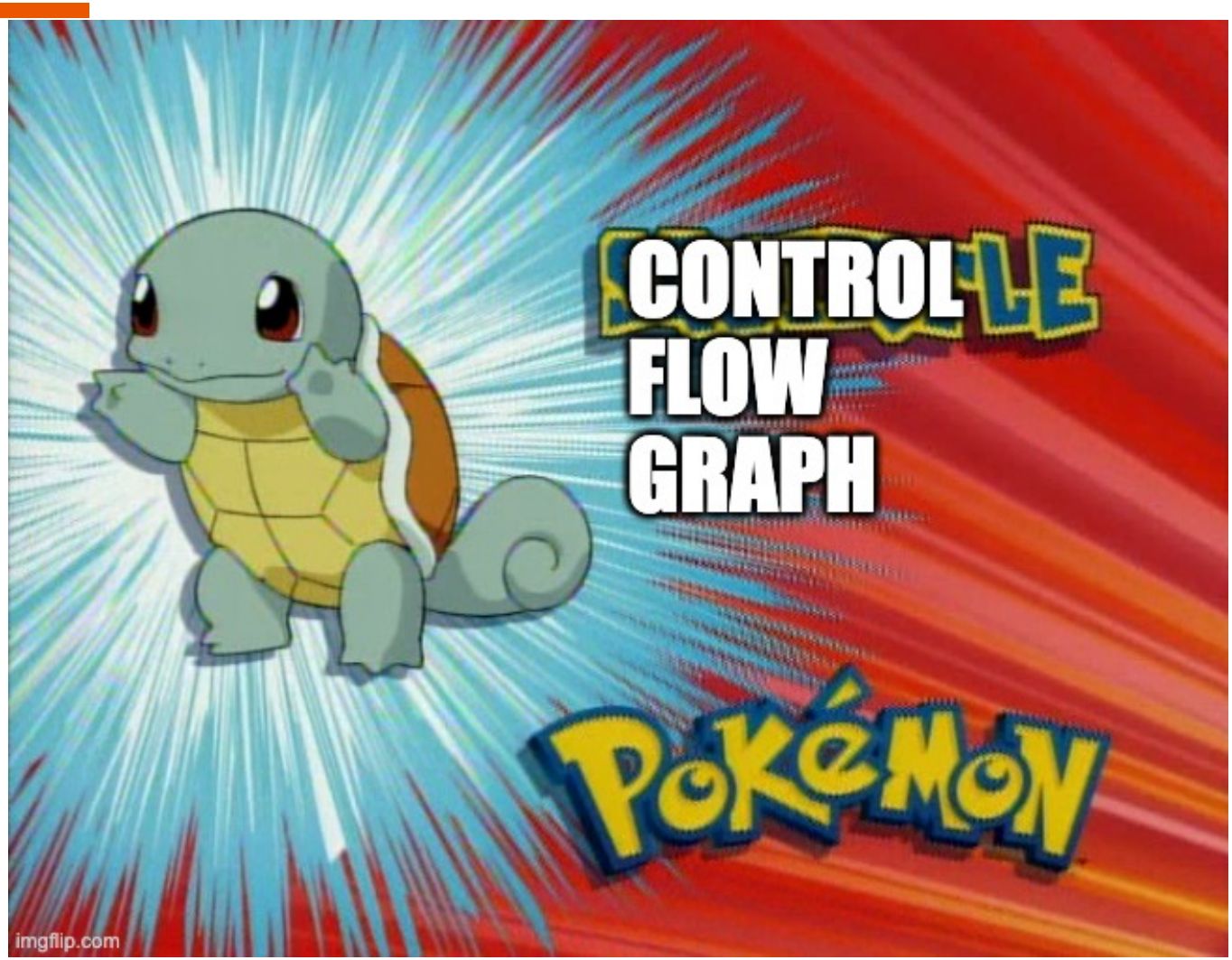
Repeats!

Path Explosion



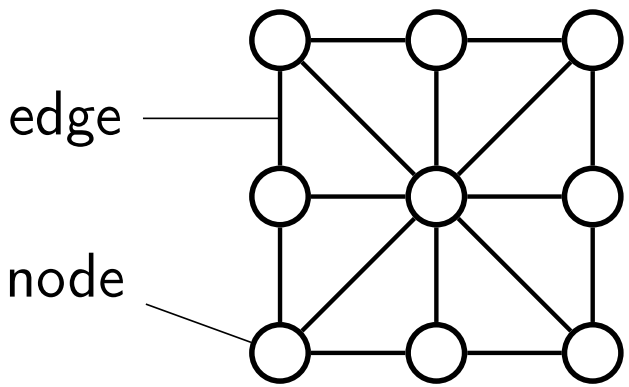
Path Explosion

Control Flow Graphs to the Rescue

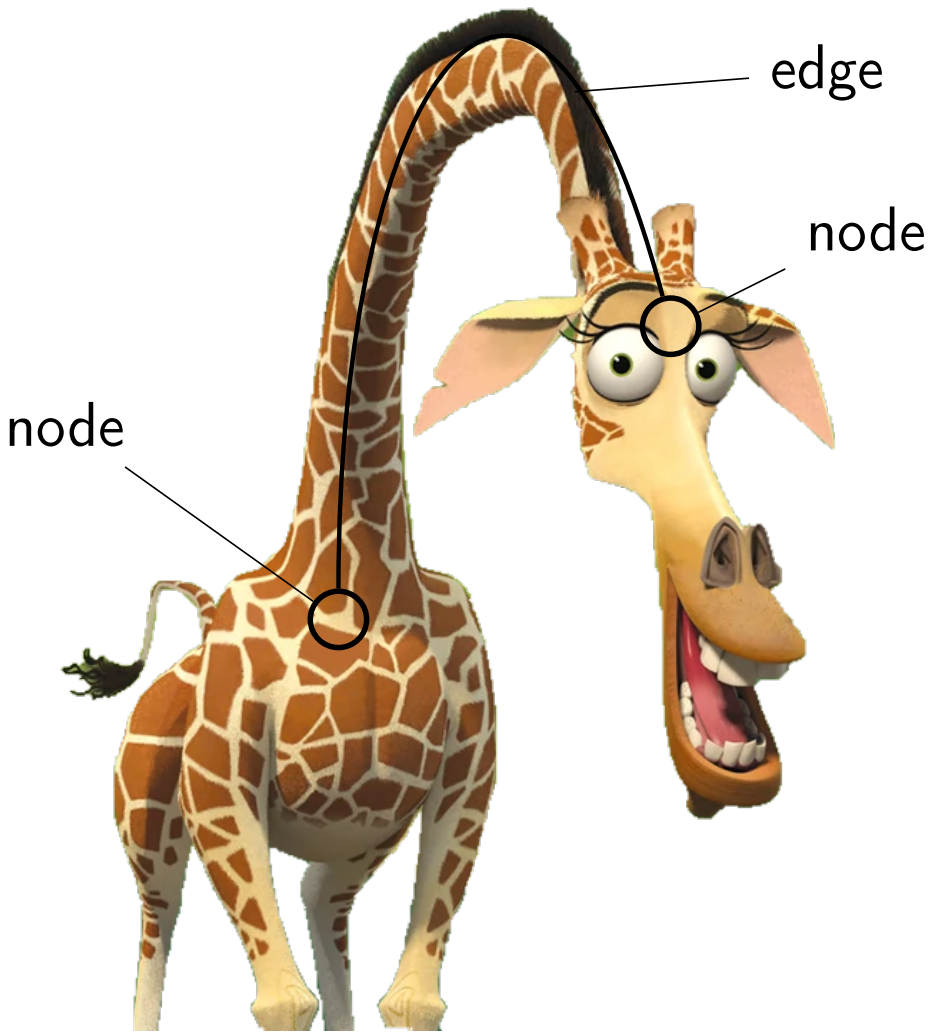


G-raph?

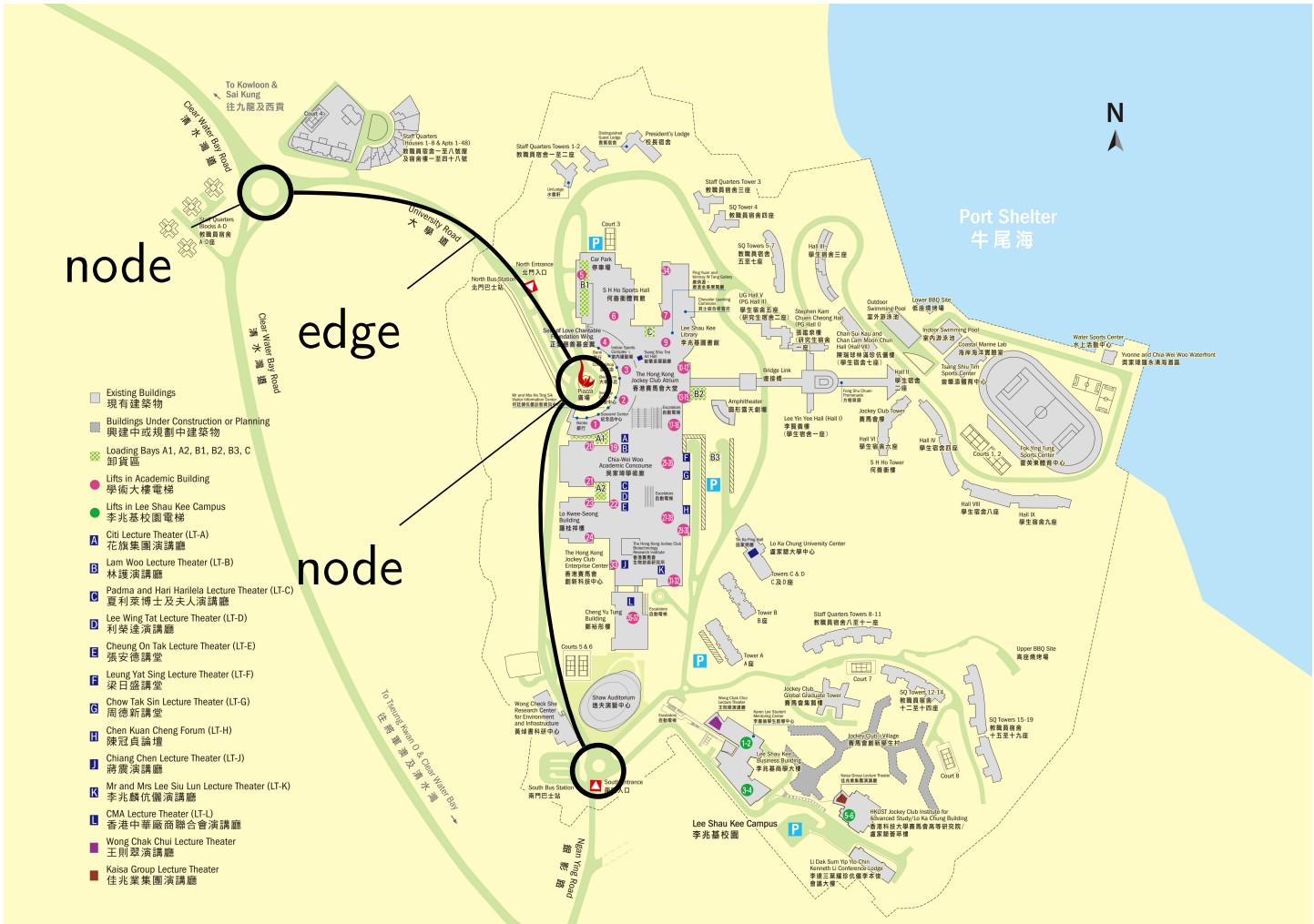
Graphs



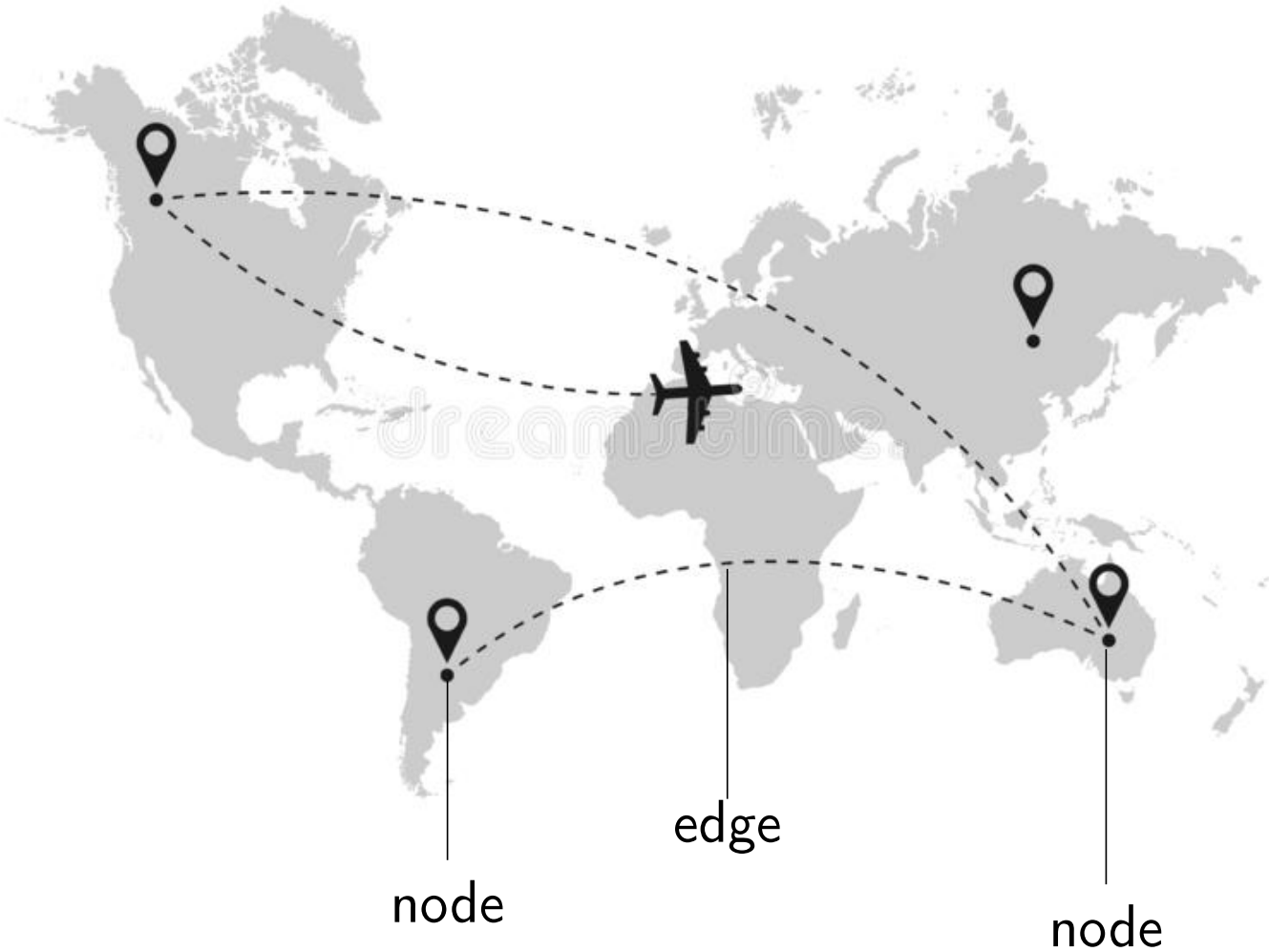
G-raph!



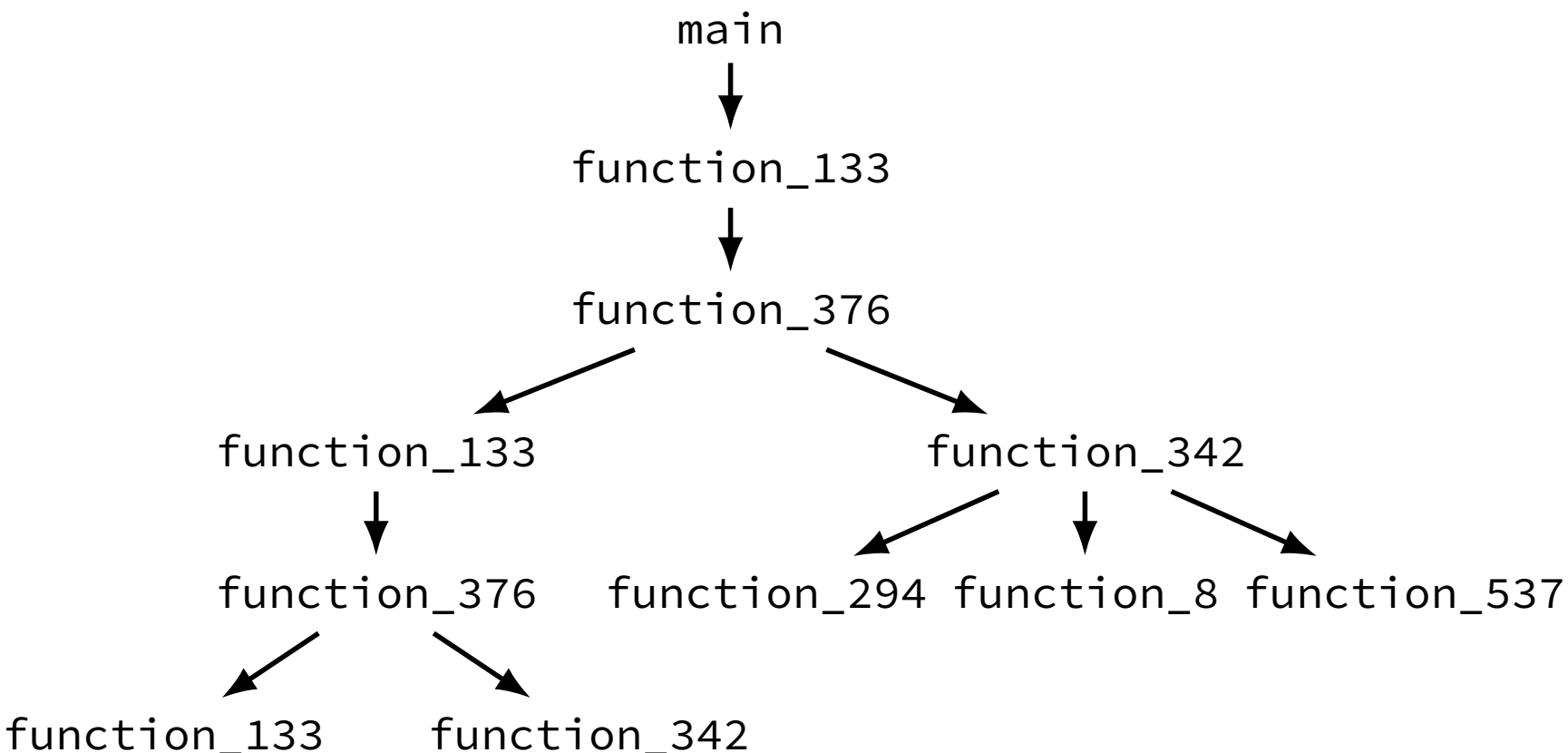
Graphs


node
edge
node
N
Port Shelter
牛尾海
Lee Shau Kee Campus
李兆基校園
University Road
大學道
Clear Water Bay Road
清水灣道
Existing Buildings
現有建築物
Buildings Under Construction or Planning
興建中或規劃中建築物
Loading Bays A1, A2, B1, B2, B3, C
卸貨區
Lifts in Academic Building
學術大樓電梯
Lifts in Lee Shau Kee Campus
李兆基校園電梯
A Citi Lecture Theater (LT-A)
花旗集團演講廳
B Lam Woo Lecture Theater (LT-B)
林護演講廳
C Padma and Hari Harilela Lecture Theater (LT-C)
夏利萊博士及夫人演講廳
D Lee Wing Tat Lecture Theater (LT-D)
利榮達演講廳
E Cheung On Tak Lecture Theater (LT-E)
張安德講堂
F Leung Yat Sing Lecture Theater (LT-F)
梁日盛講堂
G Chow Tak Sin Lecture Theater (LT-G)
周德新講堂
H Chen Kuan Cheng Forum (LT-H)
陳冠貞論壇
J Chiang Chen Lecture Theater (LT-J)
蔣震演講廳
K Mr and Mrs Lee Siu Lun Lecture Theater (LT-K)
李兆麟伉儷演講廳
L CMA Lecture Theater (LT-L)
香港中華廠商聯合會演講廳
Wong Chak Chui Lecture Theater
王則翠演講廳
Kaisa Group Lecture Theater
佳兆業集團演講廳

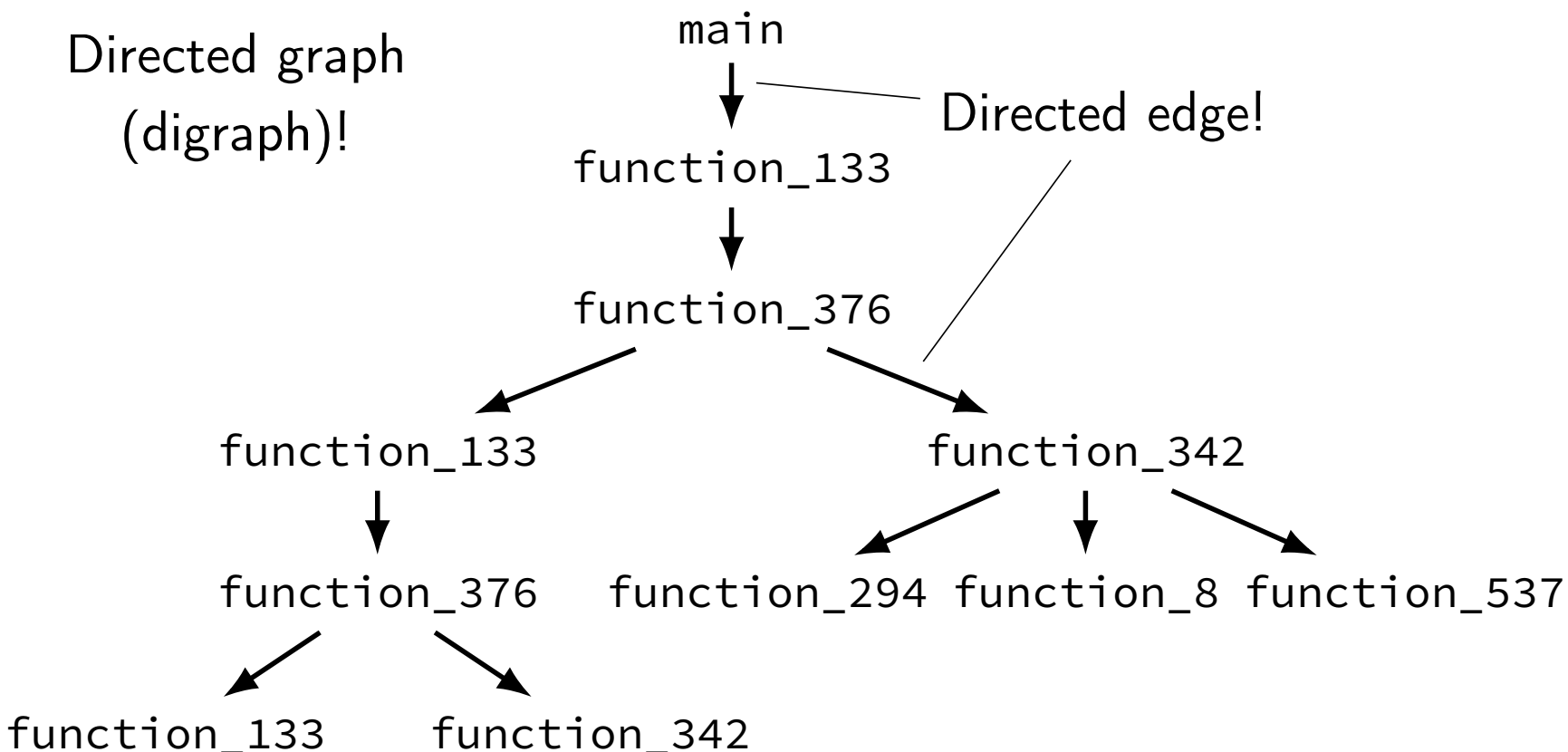
Graphs



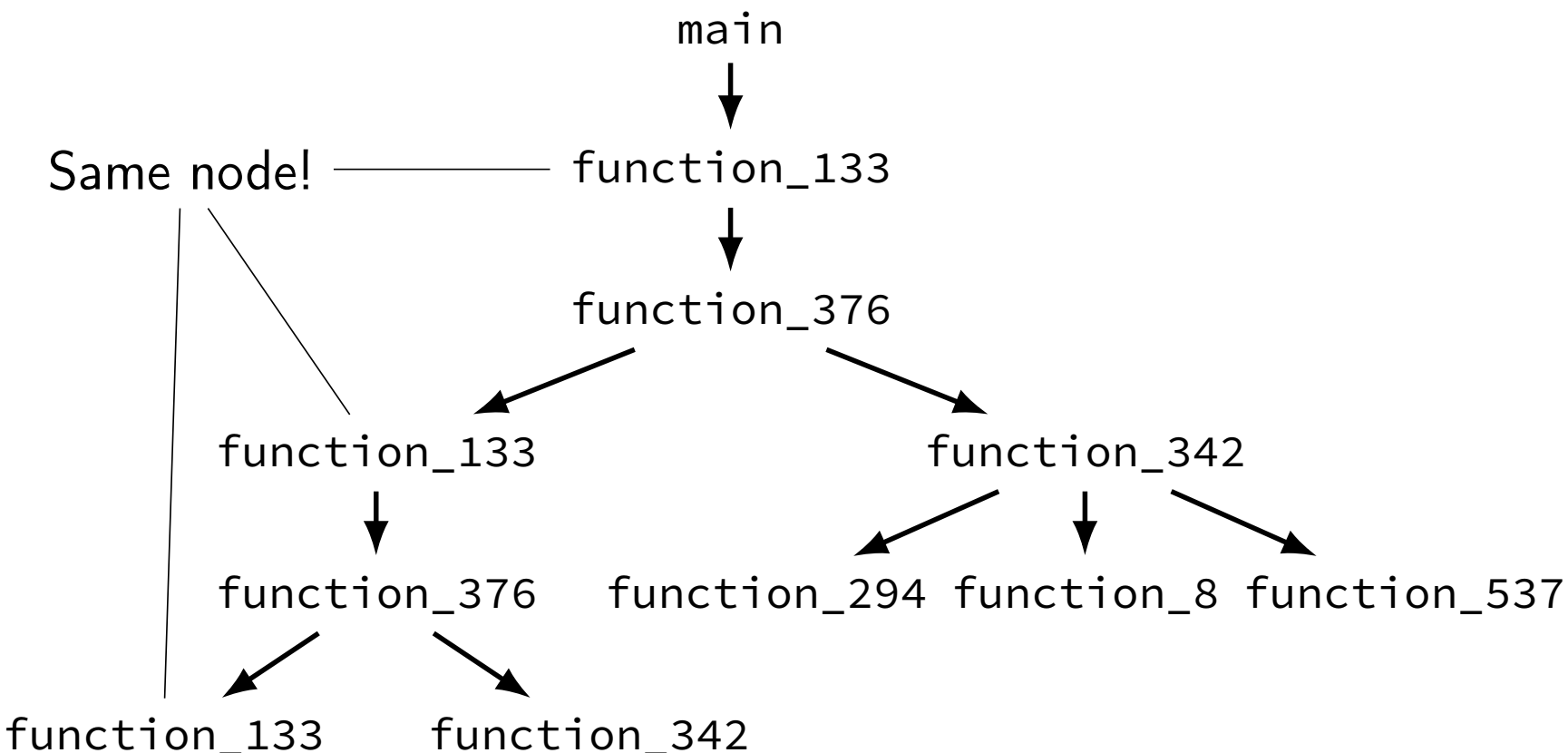
Programs as Graphs



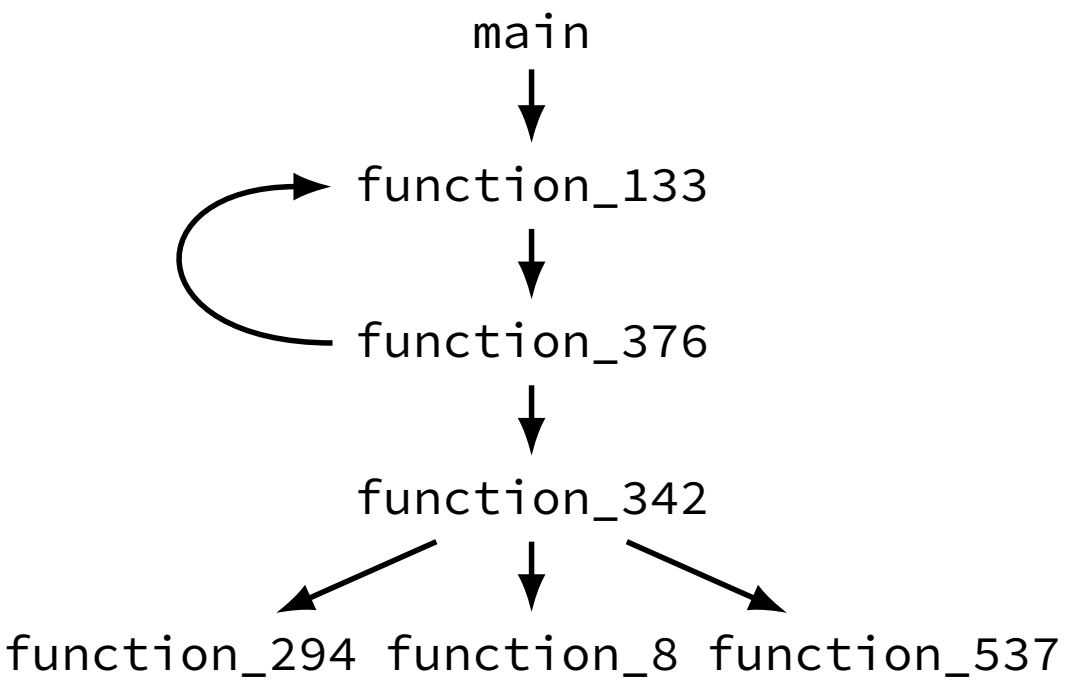
Programs as Graphs



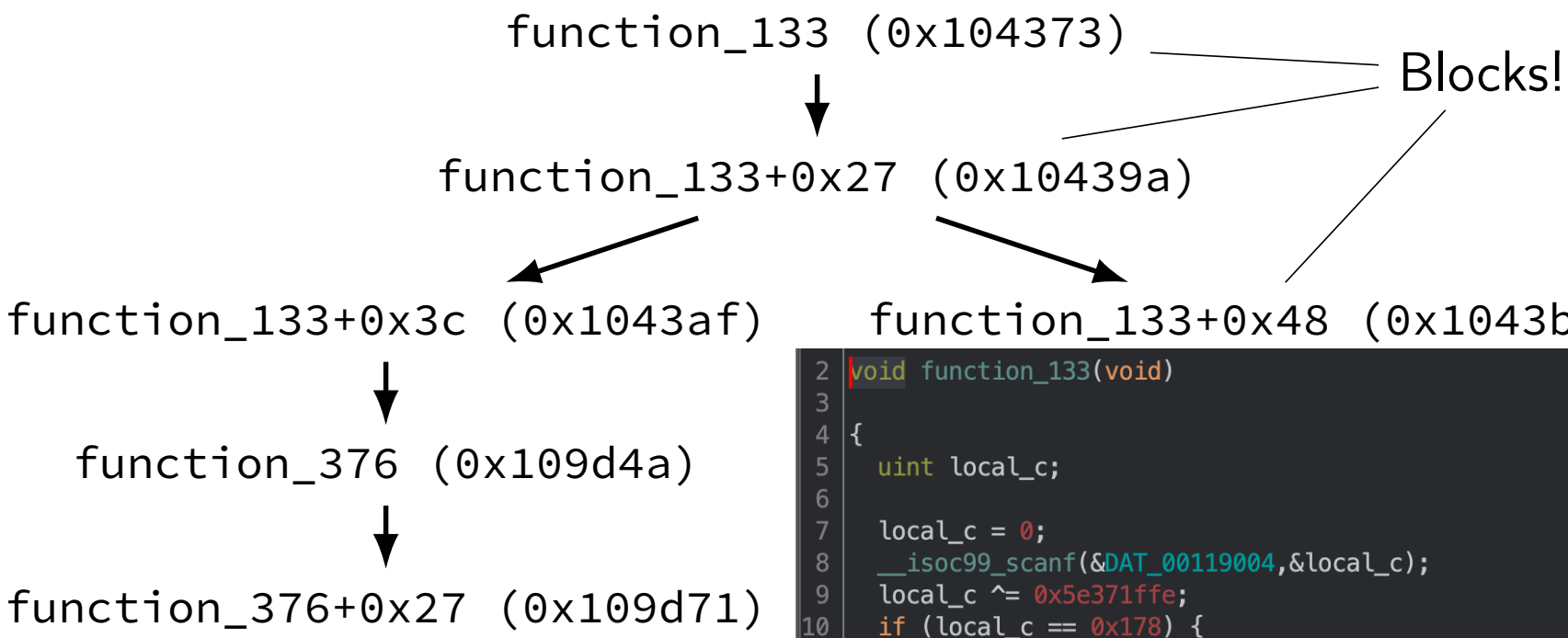
Programs as Graphs



Programs as Graphs

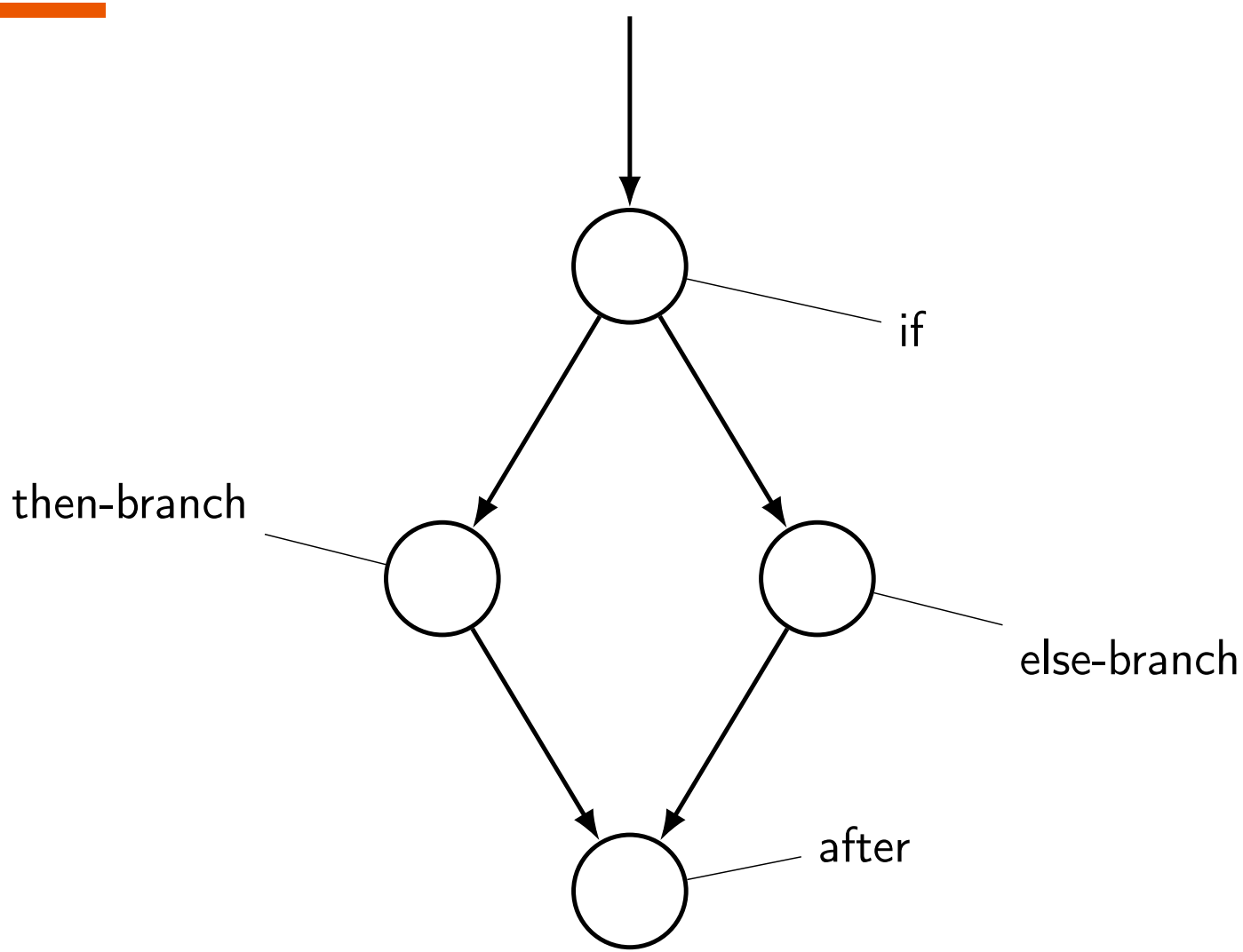


What about branches and loops?

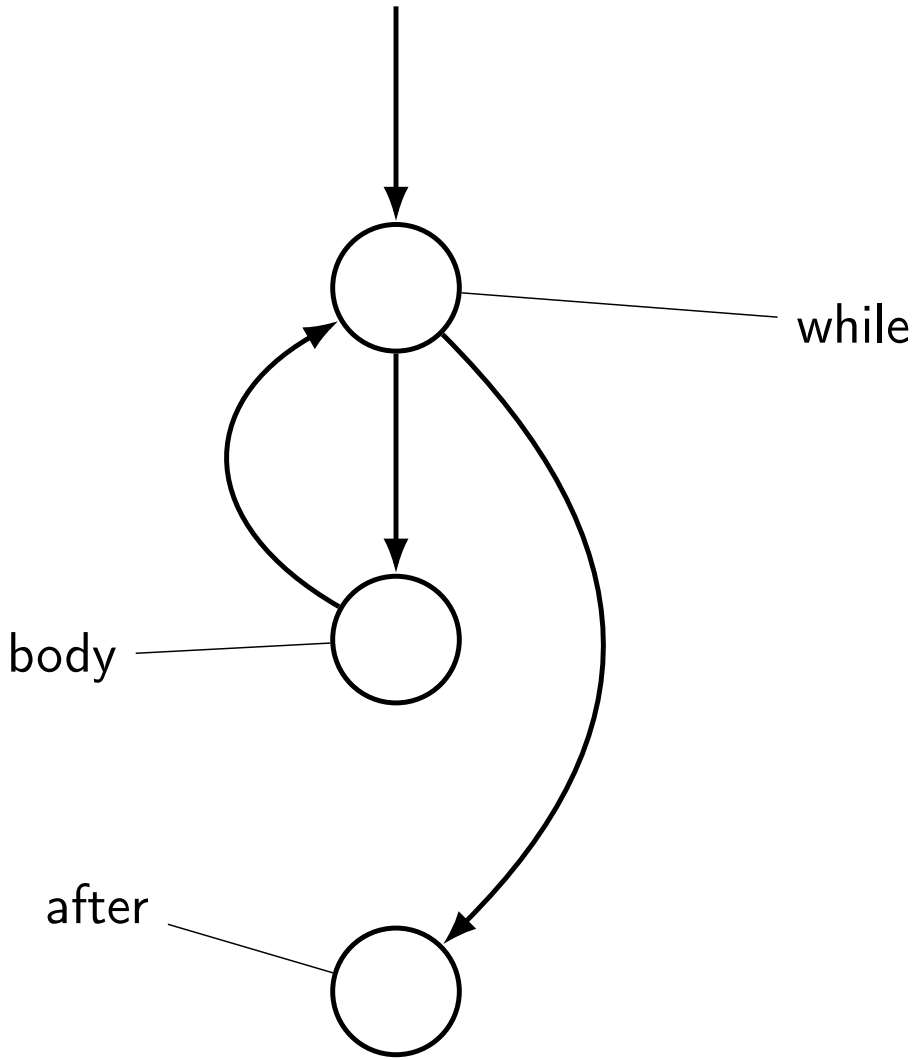



```
2  void function_133(void)
3
4  {
5    uint local_c;
6
7    local_c = 0;
8    __isoc99_scanf(&DAT_00119004,&local_c);
9    local_c ^= 0x5e371ffe;
10   if (local_c == 0x178) {
11     function_376();
12     return;
13   }
14                     /* WARNING: Subroutine does not return
15   exit(1);
16 }
```

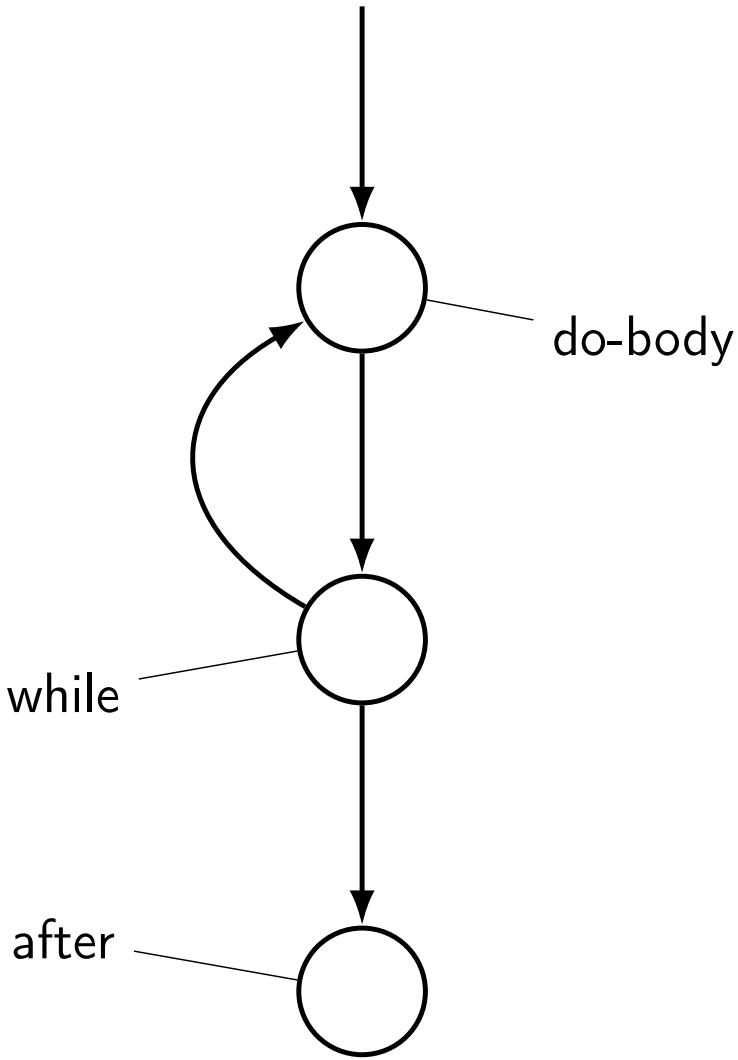
Do you recognise these CFGs?



Do you recognise these CFGs?



Do you recognise these CFGs?



CFGs in angr

`p.analyses`
`.CFGFast`

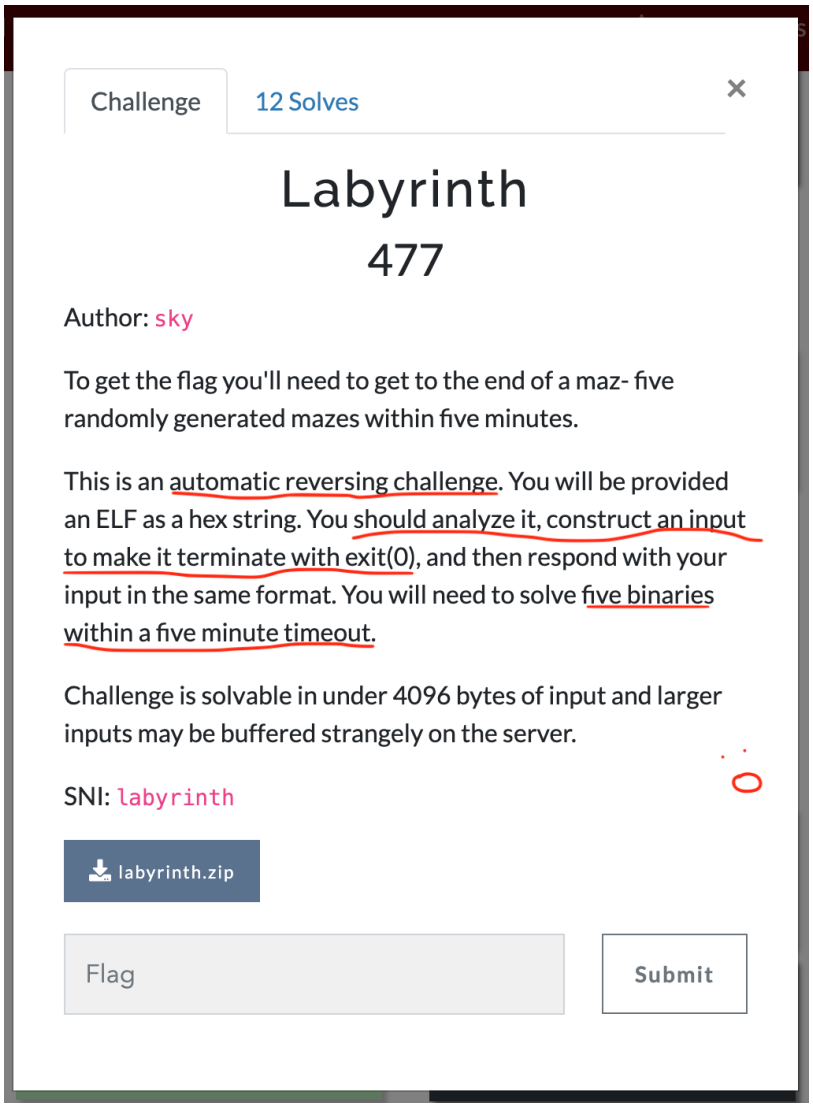
- static
- faster
- can analyse basic jumps

`p.analyses`
`.CFGEmulated`

- dynamic
- slower
- can analyse more obscure/hidden jumps, with context

Class parameters? Check the docs + source code!

Demo – Labyrinth



Summary – Labyrinth

1. Load project
2. Construct CFG (`p.analyses.CFGFast`)
3. Get 'from' and 'to' nodes (`main`, `exit(0)`)
4. Construct shortest path (`networkx.shortest_path`)
5. Follow shortest path
6. Take a dump
7. Profit!

But wait– there's moar!

Fun with CFGs

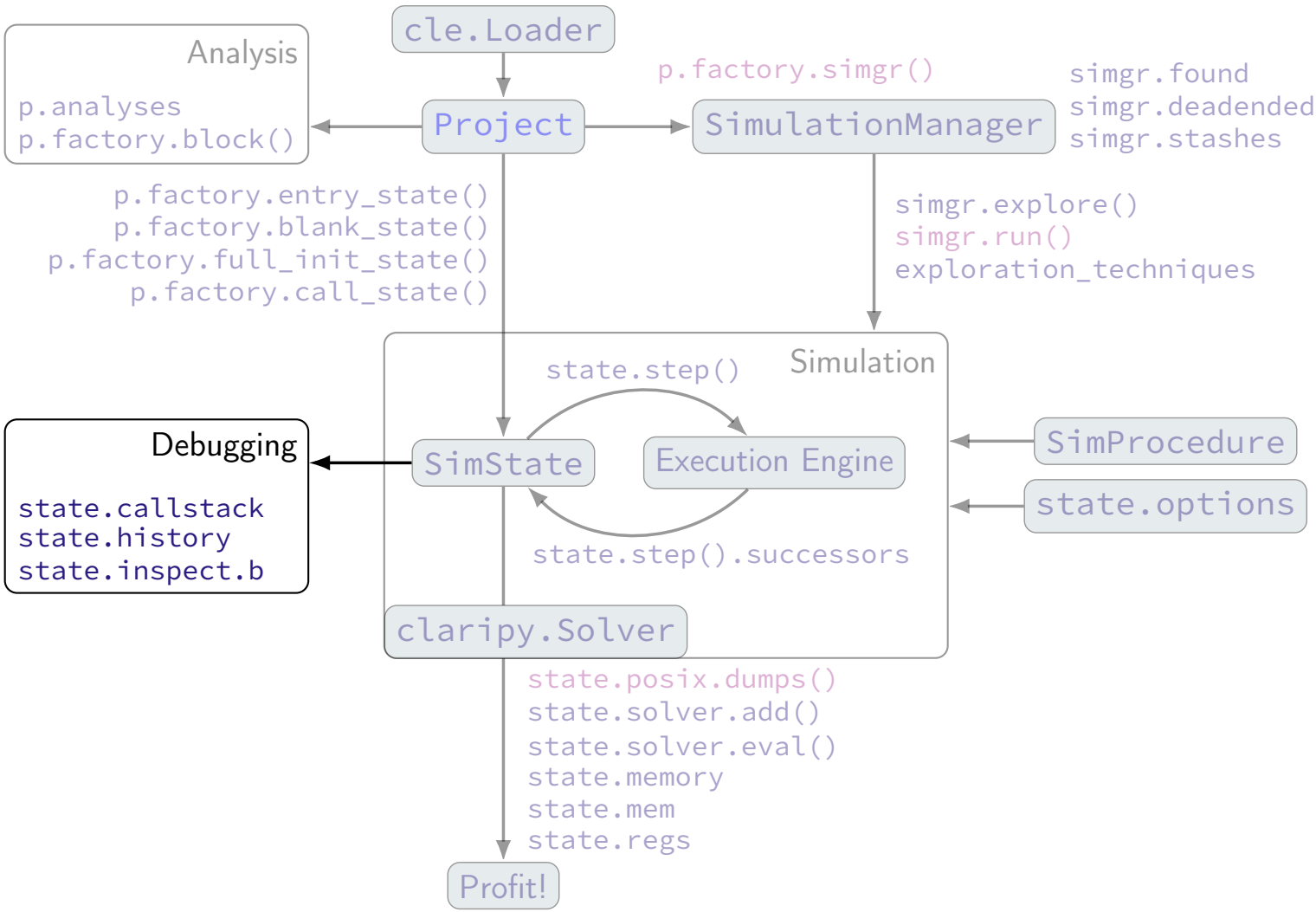
- Reachability
- Compiler Optimisations

Moar Analyses

- Value Flow Graph (VFG), Value Set Analysis (VSA)
  - What values can this variable hold?
  - CFGs are to instructions as VFGs are to data.
- Data Dependency Graph (DDG)
  - What statements does this variable depend on?
- Moar!

Moar: https://docs.angr.io/core-concepts/analyses.

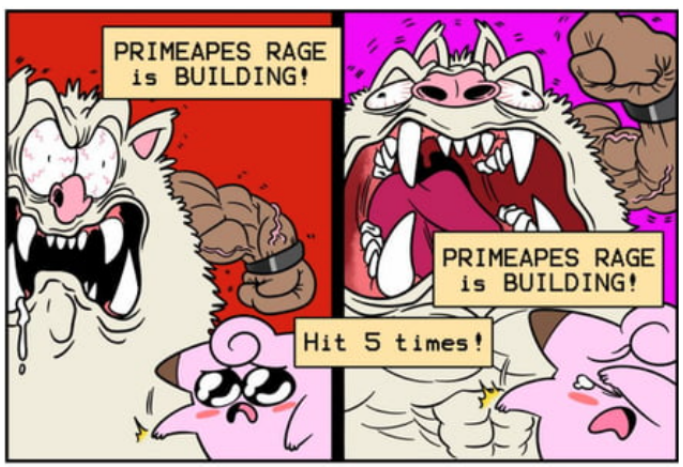
What's next?

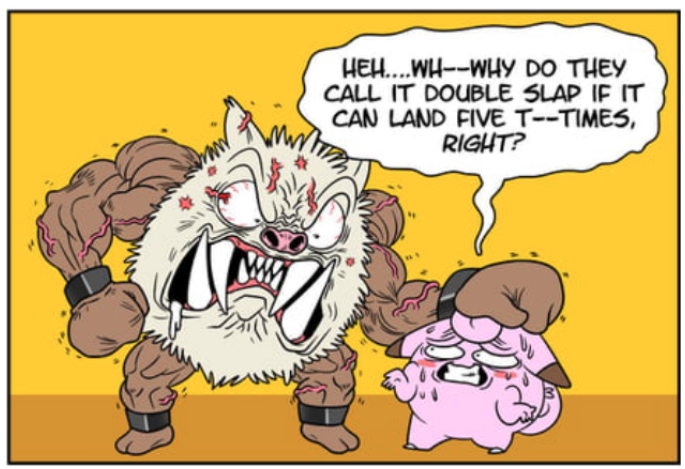

cle.Loader
Analysis
p.analyses
p.factory.block()
Project
p.factory.simgr()
SimulationManager
simgr.found
simgr.deadended
simgr.stashes
p.factory.entry_state()
p.factory.blank_state()
p.factory.full_init_state()
p.factory.call_state()
simgr.explore()
simgr.run()
exploration_techniques
Simulation
state.step()
Debugging
state.callstack
state.history
state.inspect.b
SimState
Execution Engine
SimProcedure
state.options
state.step().successors
claripy.Solver
state.posix.dumps()
state.solver.add()
state.solver.eval()
state.memory
state.mem
state.regs
Profit!

Questions?

Debugging angr Programs

Towards Enhanced Emotional Resilience and Cognizance





logging

```
import logging

logging.getLogger('angr')
       .setLevel(logging.INFO)

logging.getLogger('angr.analyses.cfg')
       .setLevel(logging.DEBUG)
```


```
INFO    | 2022-04-28 13:11:56,279 | angr.analyses.cfg.cfg_base | Loaded 3 indirect jump resolvers (2 timeless, 1 generic).
DEBUG   | 2022-04-28 13:11:56,282 | angr.analyses.cfg.cfg_fast | CFG recovery covers 1 regions:
DEBUG   | 2022-04-28 13:11:56,282 | angr.analyses.cfg.cfg_fast | ... 0x401155 - 0x418be0
INFO    | 2022-04-28 13:11:56,294 | angr.analyses.cfg.cfg_fast | Loaded 0 exception handlings from 0 binaries.
INFO    | 2022-04-28 13:11:56,311 | angr.analyses.cfg.cfg_fast | Found 1745 functions with prologue scanning.
DEBUG   | 2022-04-28 13:12:01,389 | angr.analyses.cfg.cfg_fast | Returning a new recon address: 0x418bda
DEBUG   | 2022-04-28 13:12:01,389 | angr.analyses.cfg.cfg_fast | Searching address 418bda
DEBUG   | 2022-04-28 13:12:01,389 | angr.analyses.cfg.cfg_fast | Searching address 418bdb
DEBUG   | 2022-04-28 13:12:01,389 | angr.analyses.cfg.cfg_fast | Force-scanning to 0x418bda
DEBUG   | 2022-04-28 13:12:01,390 | angr.analyses.cfg.cfg_fast | Force-scan jumping failed
DEBUG   | 2022-04-28 13:12:03,074 | angr.analyses.cfg.cfg_base | Function chunk 0x418bda is probably used as a function ali
INFO    | 2022-04-28 13:12:03,178 | angr.sim_manager | Stepping active of <SimulationManager with 1 active>
INFO    | 2022-04-28 13:12:03,190 | angr.engines.engine | Ticked state: <IRSB from 0x4011b3: 1 sat>
INFO    | 2022-04-28 13:12:03,190 | angr.sim_manager | Stepping active of <SimulationManager with 1 active>
```

Check `logging.root.manager.loggerDict` for existing loggers!

Debugging States

- `state.callstack`: addresses, flow.
- `state.history`: data, constraints, jumps.
- `state.inspect.b`: breakpoints!

Actually these aren't just for debugging!

state.callstack

```
>>> print(state.callstack)
Backtrace:
Frame 0: 0x40a327 => 0x4011c4, sp = 0x7fffffffefe78
Frame 1: 0x411863 => 0x40a2d8, sp = 0x7fffffffefe98
Frame 2: 0x401f84 => 0x411809, sp = 0x7fffffffefeb8
Frame 3: 0x406c14 => 0x401f41, sp = 0x7fffffffefed8
Frame 4: 0x4069b2 => 0x406bbe, sp = 0x7fffffffefef8
Frame 5: 0x402b53 => 0x406936, sp = 0x7fffffffeff18
Frame 6: 0x401d4b => 0x402aed, sp = 0x7fffffffeff38
Frame 7: 0x4104d4 => 0x401d08, sp = 0x7fffffffeff58
Frame 8: 0x40b5aa => 0x410464, sp = 0x7fffffffeff78
Frame 9: 0x413179 => 0x40b4f3, sp = 0x7fffffffeff98
Frame 10: 0x4148df => 0x413123, sp = 0x7fffffffeffb8
Frame 11: 0x402523 => 0x41487d, sp = 0x7fffffffeffd8
Frame 12: 0x4011b3 => 0x4024c1, sp = 0x7fffffffefff8
Frame 13: 0x0 => 0x0, sp = 0xffffffffffffffff
```

Useful Stuff

```
print(state.callstack)     # Display backtrace.
state.callstack.stack_ptr # Stack pointer (int).
state.callstack.func_addr # Address of current function.
state.callstack.ret_addr  # Return address.
```

More: https://docs.angr.io/core-concepts/states#the-callstack-plugin.

`state.history` – Overview

A collection of different iterables...

- `.actions`: tracks data and constraints.
- `.bbl_addrs`: blocks executed to reach this state.
- `.descriptions`: logs as strings.
- `.jump_guards`: branch constraints.
- `.events`: similar to `.actions` but moar.
  - Useful for checking why states deadended.

More: https://docs.angr.io/core-concepts/states#the-history-plugin.

`state.history` – Viewing

Useful Stuff

- `state.history.actions.hardcopy`
  - Turn the actions iterable into a list.
- `print(*state.history.actions, sep='\n')`
  - Prints each action on its own line.

state.history.actions

Tracks data (register/memory read and writes) and constraints.

```
>>> print(*state.history.actions, sep='\n')
<SimActionData __isoc99_scanf() reg/read: rdi  ----->>  <BV64 0x419004>>
<SimActionData __isoc99_scanf() mem/read: <BV64 0x419004>  ----->>  <BV1032 0x25750a00011b033b741b0
000bc81feff201c0000ca81feff3c1c00001f82feff5c1c0000da82feff7c1c00005583feff9c1c0000aa83feffbc1c000
<SimActionData __isoc99_scanf() mem/read: <BV64 0x419004>  ----->>  <BV24 0x25750a>>
<SimActionConstraint __isoc99_scanf() <SAO <Bool packet_0_stdin_3_80[7:0] == scanf_u_2_32 / 0x1 %
<SimActionConstraint __isoc99_scanf() <SAO <Bool packet_0_stdin_3_80[15:8] == scanf_u_2_32 / 0xa %
<SimActionConstraint __isoc99_scanf() <SAO <Bool packet_0_stdin_3_80[23:16] == scanf_u_2_32 / 0x64
<SimActionConstraint __isoc99_scanf() <SAO <Bool packet_0_stdin_3_80[31:24] == scanf_u_2_32 / 0x3e
<SimActionConstraint __isoc99_scanf() <SAO <Bool packet_0_stdin_3_80[39:32] == scanf_u_2_32 / 0x27
<SimActionConstraint __isoc99_scanf() <SAO <Bool packet_0_stdin_3_80[47:40] == scanf_u_2_32 / 0x18
<SimActionConstraint __isoc99_scanf() <SAO <Bool packet_0_stdin_3_80[55:48] == scanf_u_2_32 / 0xf4
<SimActionConstraint __isoc99_scanf() <SAO <Bool packet_0_stdin_3_80[63:56] == scanf_u_2_32 / 0x98
<SimActionConstraint __isoc99_scanf() <SAO <Bool packet_0_stdin_3_80[71:64] == scanf_u_2_32 / 0x5f
<SimActionConstraint __isoc99_scanf() <SAO <Bool packet_0_stdin_3_80[79:72] == scanf_u_2_32 / 0x3b
<SimActionData __isoc99_scanf() reg/read: rsi  ----->>  <BV64 0x7ffffffffeffec>>
<SimActionData __isoc99_scanf() mem/write: <BV64 0x7ffffffffeffec>  <<-----  <BV32 scanf_u_2_32>>
<SimActionConstraint __isoc99_scanf() <SAO <Bool packet_1_stdin_4_8 == 10>>>
```

Useful Stuff

Add this before constructing the simgr!

```
state.options |= sim_options.refs    # Track EVERYTHING!
```

`state.inspect.b(event_type, **kwargs)`

`event_type`

- `mem_read`, `mem_write`: memory is being read/written.
- `reg_read`, `reg_write`: register is being read/written.
- `constraints`: new constraints are being added.
- `call`: a `call` instruction is hit.
- `ret`: a `ret` instruction is hit.
- `symbolic_variable`: a new symbolic variable is being created.
- `syscall`: a syscall is being executed.

More: https://docs.angr.io/core-concepts/simulation#breakpoints.

`state.inspect.b(event_type, **kwargs)`

`when=`

- `BP_BEFORE`: trigger before the breakpoint.
- `BP_AFTER`: trigger after the breakpoint.

`action=`

- `func`: execute your own Python handler.
- `BP_IPDB`: enter an IPython debugger.
- `BP_IPYTHON`: enter an IPython shell.

`**kwargs`

- Different kwargs are available for each `event_type`! Check the link below.

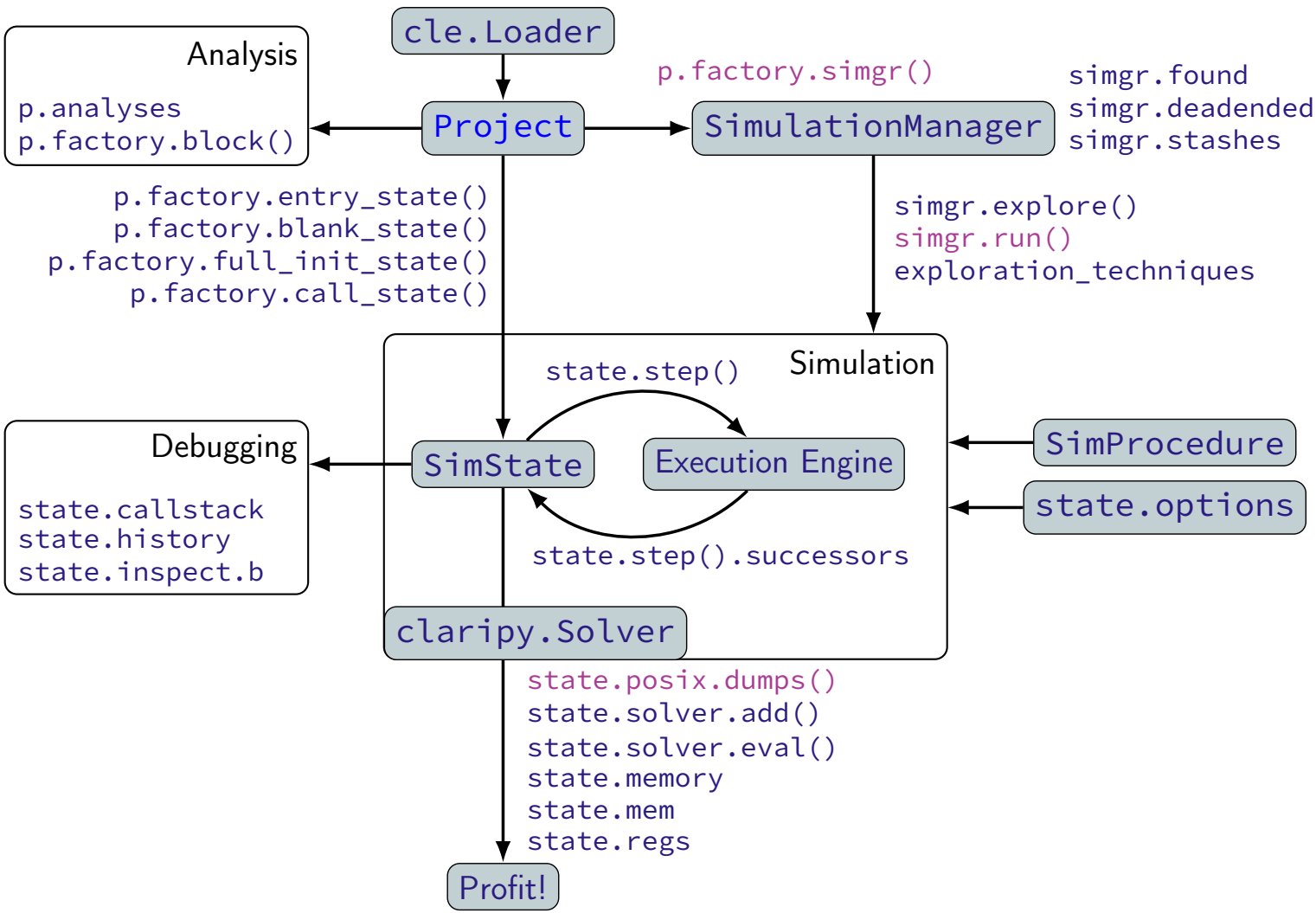
More: https://docs.angr.io/core-concepts/simulation#breakpoints.

6 Appendix

angr Tips

- Breathe and relax.
- Always use angr in a virtual environment (e.g. venv, conda).
- Use a Jupyter notebook/IPython, or run scripts interactively (`python3 -i script.py`) to easily play around with variables and methods.
- `import monkeyhex` to conveniently view numbers as hex.
- Tab completion is your friend.
- The most updated documentation is the code itself.

angr Model (Link Galore)



More Links

- Repo: https://github.com/angr/angr
- Docs (Book-like): https://docs.angr.io/
- API: https://api.angr.io/
- Examples: https://docs.angr.io/examples
- More Examples: https://github.com/angr/angr-doc/blob/master/docs/more-examples.md
- Attack Plan (Strategies): https://github.com/bordig-f/angr-strategies
- CFG (Wikipedia): https://en.wikipedia.org/wiki/Control-flow_graph
- CFG (1970s paper, a potentially interesting read, mathy/theoretical): https://dl.acm.org/doi/10.1145/390013.808479

Questions?