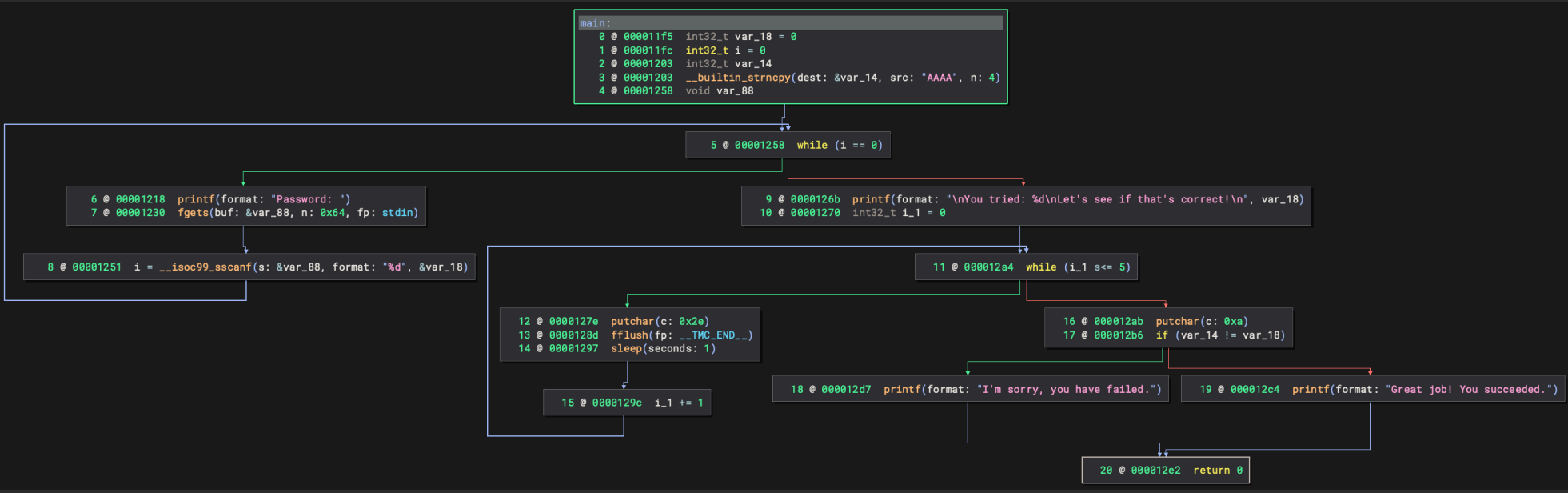
"A Slice of" Modern Program Analysis

Graph is Better than Linear

(IN MOST CASES)

An Aside
Goals & Motivation
Prior Work
Getting Started
Staying Oriented
Cutting Deeper
Staying Organized
Dominance
Finding Flows
ILs
Variable Slices
Intraprocedural
Interprocedural
Conclusion

(JK JUST MY THESIS)



GRAPH SHILLING

RECOGNIZABLE PATTERNS

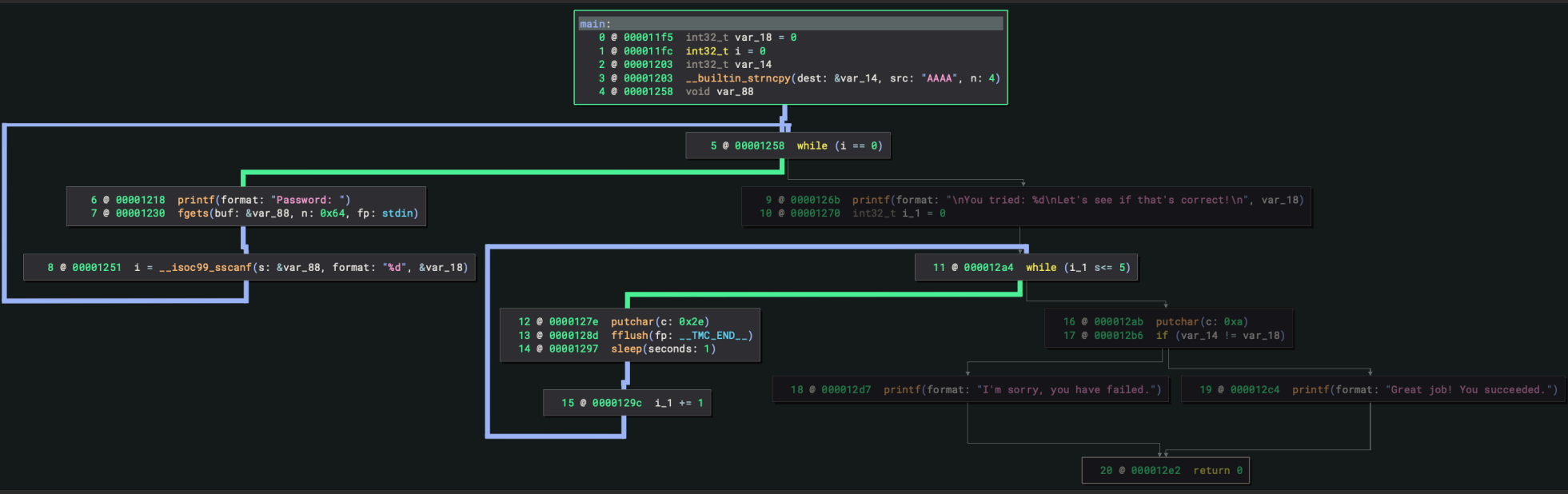
Two

Loops

```
main:
  0 @ 000011f5  int32_t var_18 = 0
  1 @ 000011fc  int32_t i = 0
  2 @ 00001203  int32_t var_14
  3 @ 00001203  __builtin_strncpy(dest: &var_14, src: "AAAA", n: 4)
  4 @ 00001258  void var_88

  5 @ 00001258  while (i == 0)

  6 @ 00001218  printf(format: "Password: ")
  7 @ 00001230  fgets(buf: &var_88, n: 0x64, fp: stdin)

  8 @ 00001251  i = __isoc99_sscanf(s: &var_88, format: "%d", &var_18)

  9 @ 0000126b  printf(format: "\nYou tried: %d\nLet's see if that's correct!\n", var_18)
 10 @ 00001270  int32_t i_1 = 0

 11 @ 000012a4  while (i_1 s<= 5)

 12 @ 0000127e  putchar(c: 0x2e)
 13 @ 0000128d  fflush(fp: __TMC_END__)
 14 @ 00001297  sleep(seconds: 1)

 15 @ 0000129c  i_1 += 1

 16 @ 000012ab  putchar(c: 0xa)
 17 @ 000012b6  if (var_14 != var_18)

 18 @ 000012d7  printf(format: "I'm sorry, you have failed.")

 19 @ 000012c4  printf(format: "Great job! You succeeded.")

 20 @ 000012e2  return 0
```

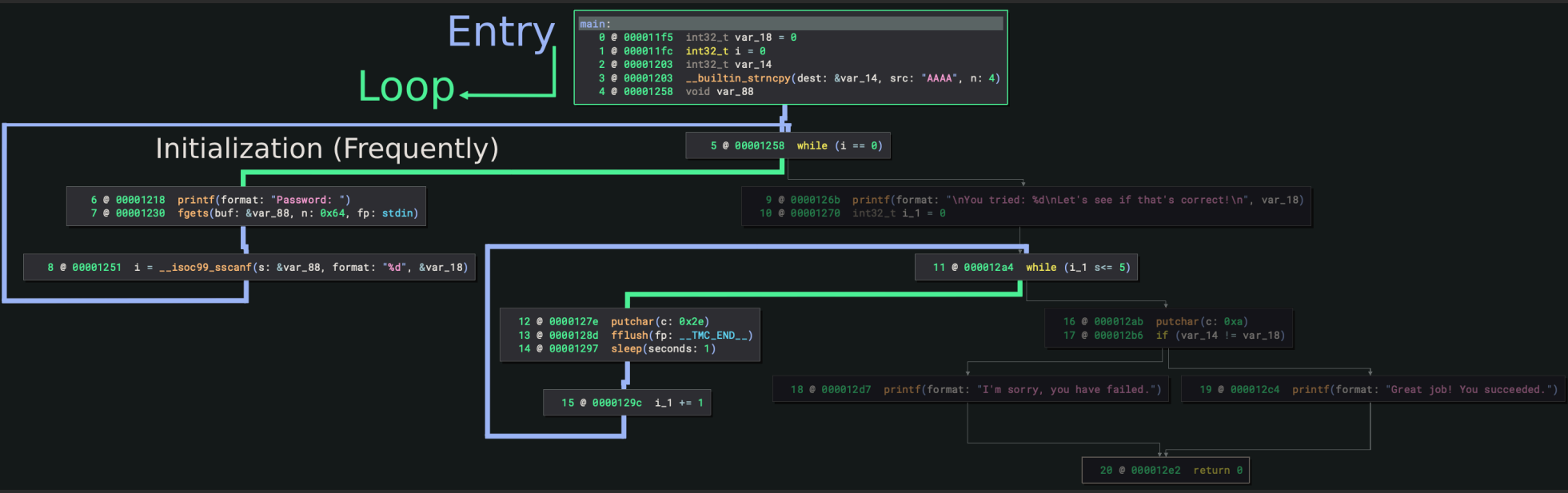
GRAPH SHILLING

RECOGNIZABLE PATTERNS



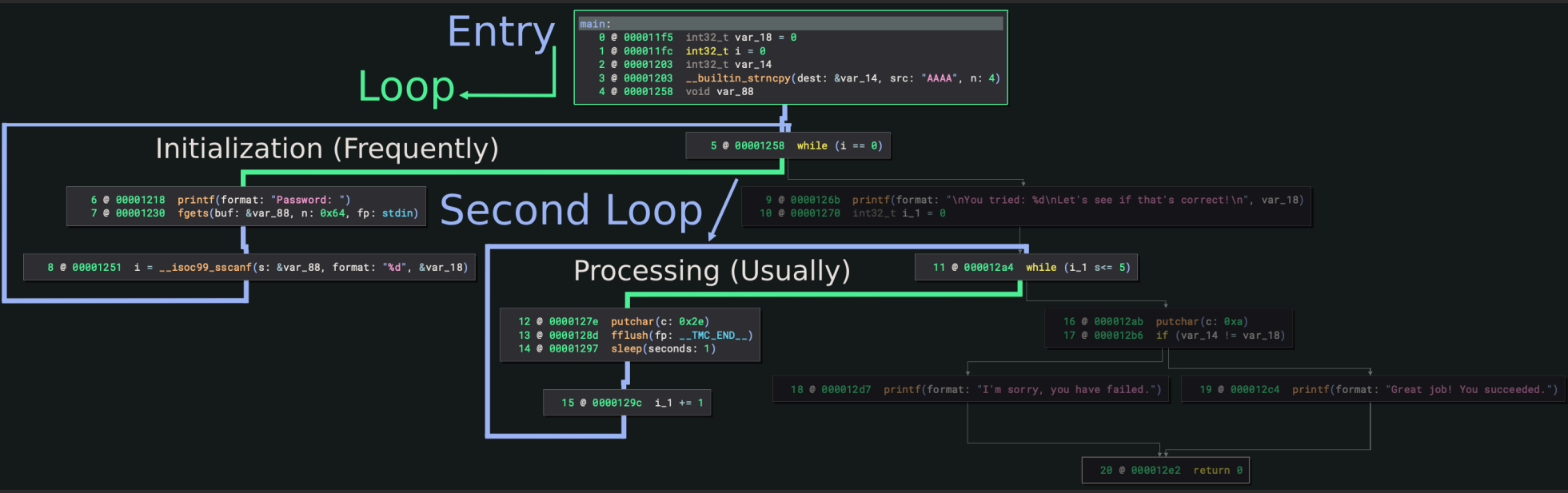
GRAPH SHILLING

RECOGNIZABLE PATTERNS



GRAPH SHILLING

RECOGNIZABLE PATTERNS

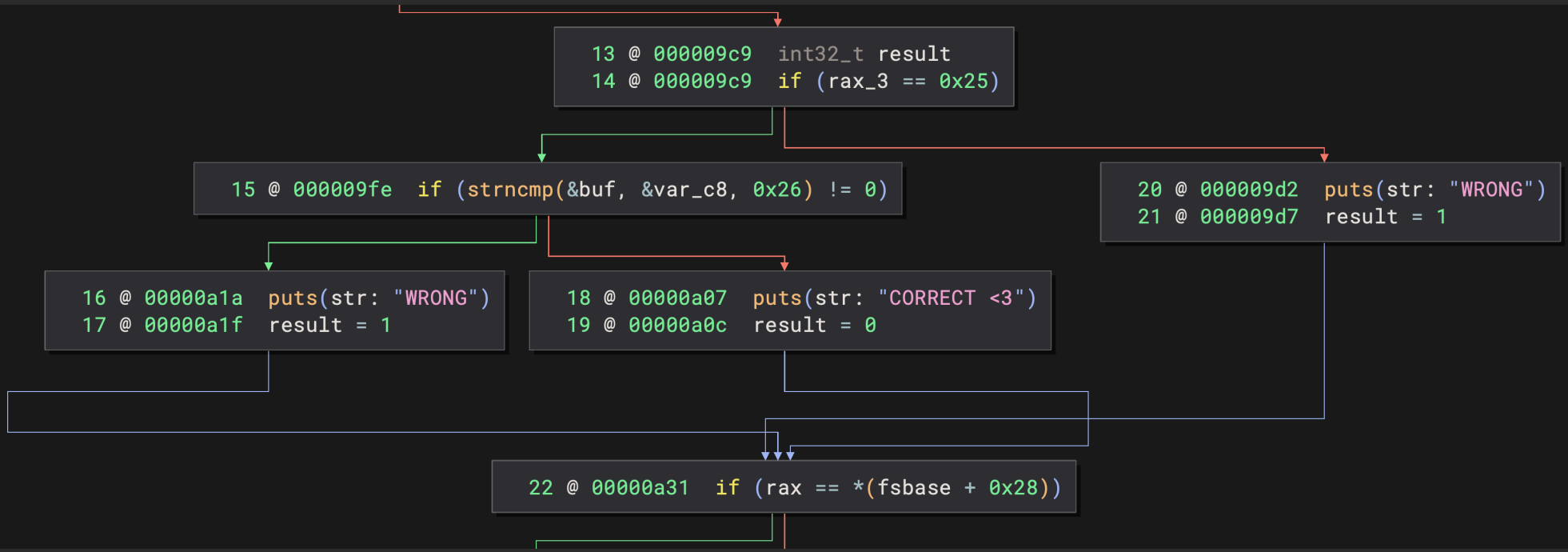


GRAPH SHILLING

RECOGNIZABLE PATTERNS

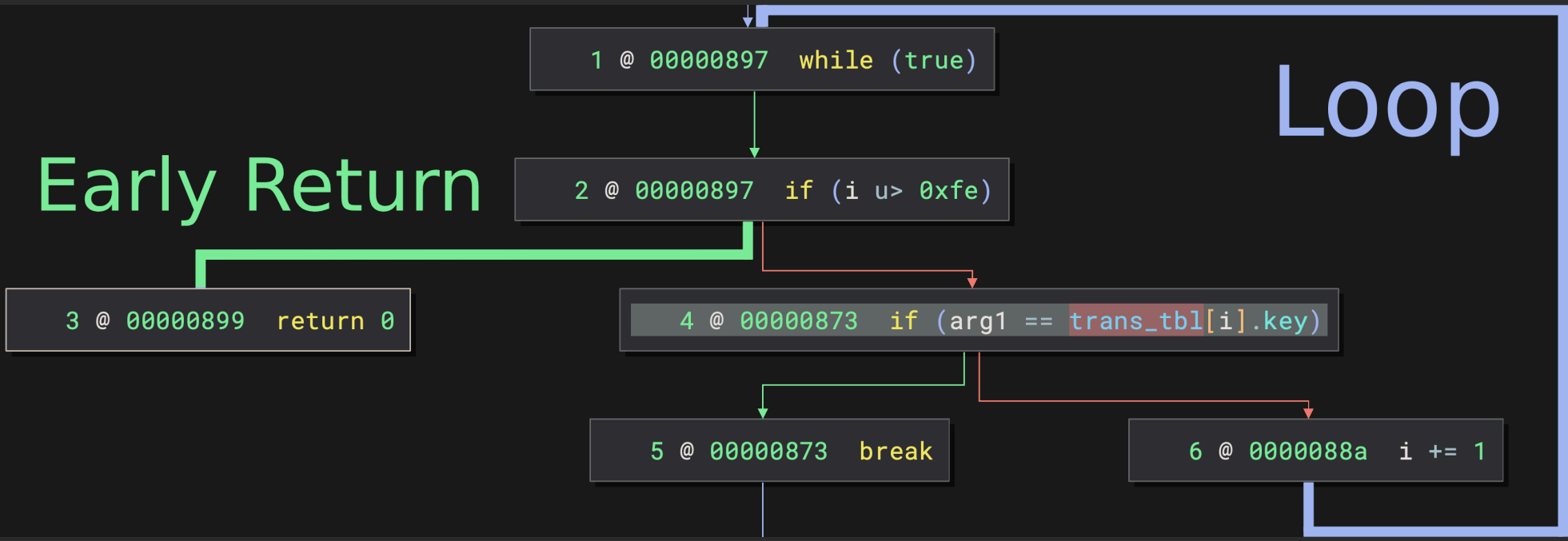
GRAPH SHILLING

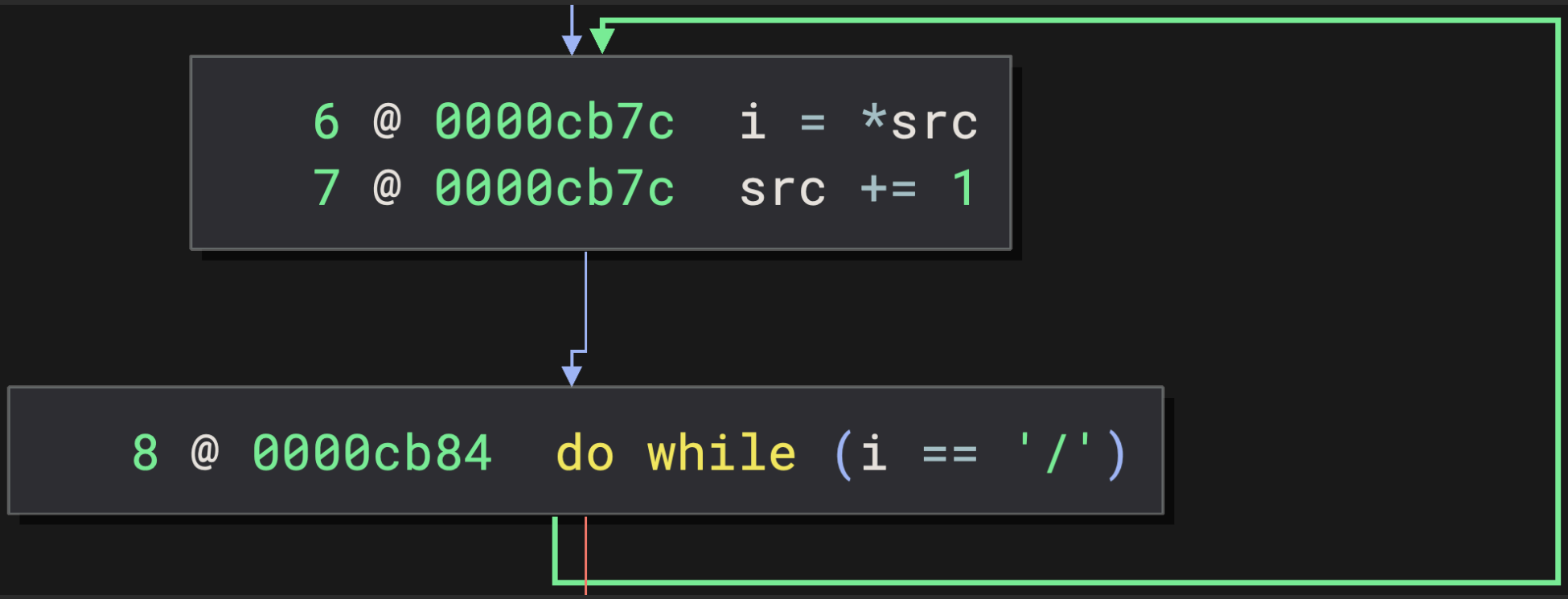
RECOGNIZABLE PATTERNS

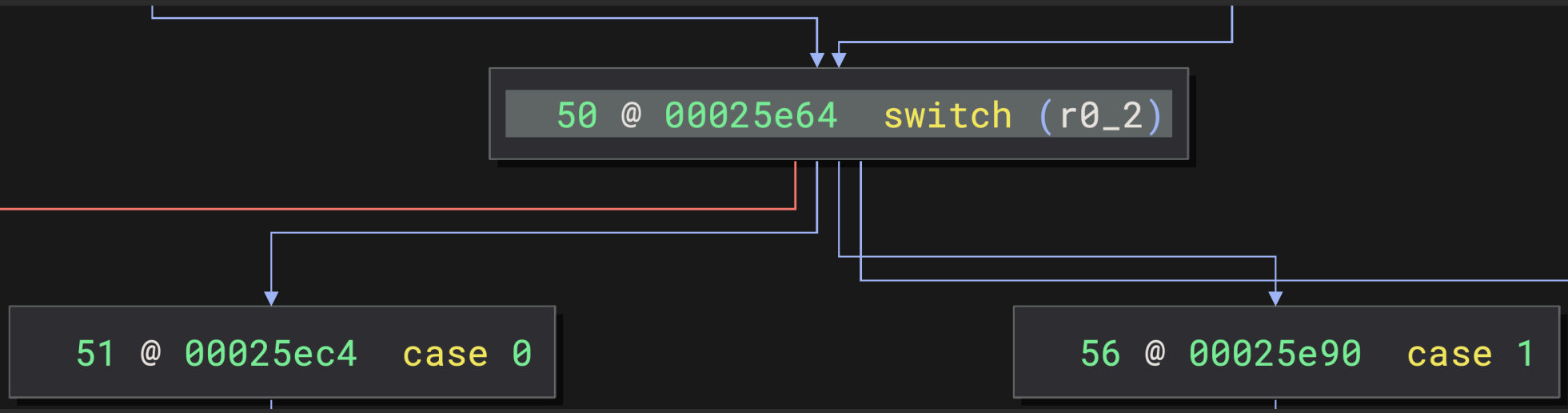


RECOGNIZABLE PATTERNS

IF-ELSE CHAINS

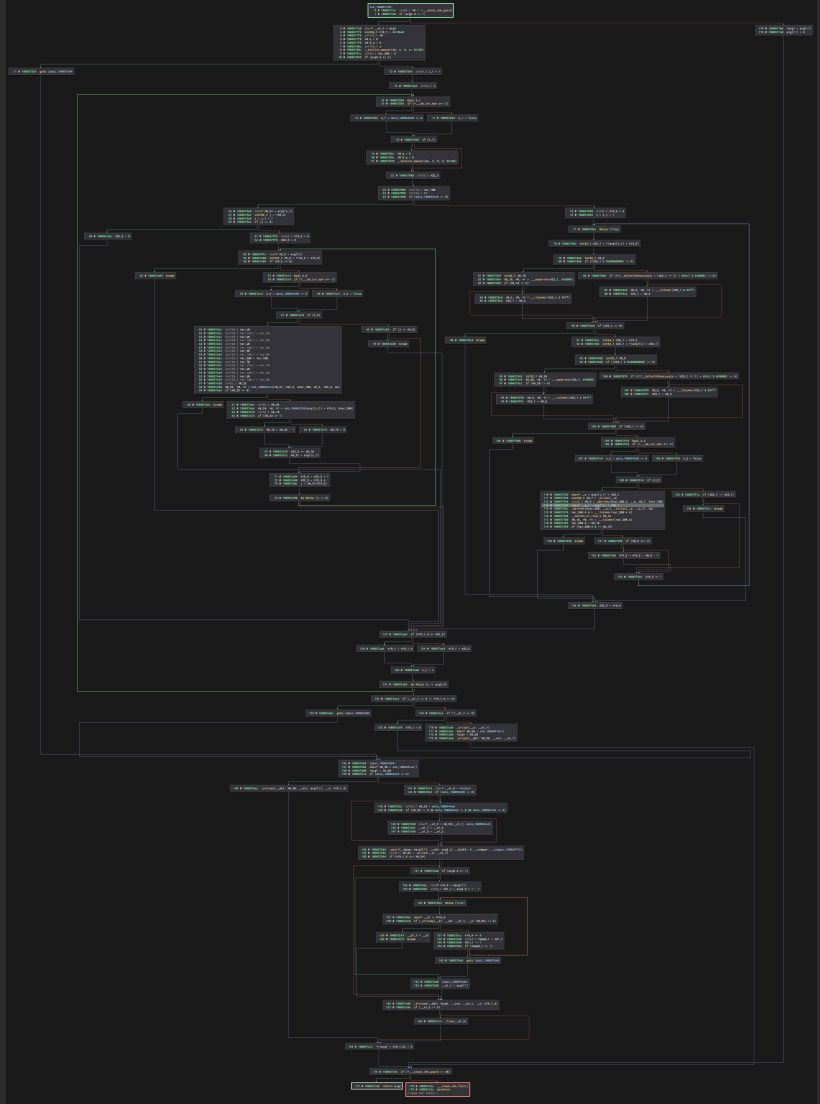

Loop
Early Return
1 @ 00000897 while (true)
2 @ 00000897 if (i u> 0xfe)
3 @ 00000899 return 0
4 @ 00000873 if (arg1 == trans_tbl[i].key)
5 @ 00000873 break
6 @ 0000088a i += 1


6 @ 0000cb7c i = *src
7 @ 0000cb7c src += 1
8 @ 0000cb84 do while (i == '/')



RECOGNIZABLE PATTERNS

DISPATCH



CFG PATTERNS

USE MORE GRAPHS

An Aside
Goals & Motivation
Prior Work
Getting Started
Staying Oriented
Cutting Deeper
Staying Organized
Dominance
Finding Flows
ILs
Variable Slices
Intraprocedural
Interprocedural
Conclusion

Tantō 2.0

INTRODUCING…

open-source
slicing framework
& plugin for Binary Ninja

Users:

- Understand Faster
- More Immersive, Stay Oriented
- Take Better Notes

Developers:

- Powerful Visualizations
- Abundant Simplifications
- Reduce Discovery and Verification Timelines

Researchers:

- Framework for Experimentation
- Simple Visualizations for Complex Concepts

- 5 years teaching Program Analysis
- Inspired by 'DARPA Hard' problems
  - (ARMADA, AMP & DIGIHEALS)
- With Vector 35 since 2018
  - (Makers of Binary Ninja)
- Interaction was novel in 1991
- New visualizations in major tools?
- Want to solve problems
- January 11th, 2022: Tantō 1.0

KYLE MARTIN (@ELYKDEER)

ABOUT ME / BACKGROUND FOR THIS TALK

Ghidra’s Backwards Slices

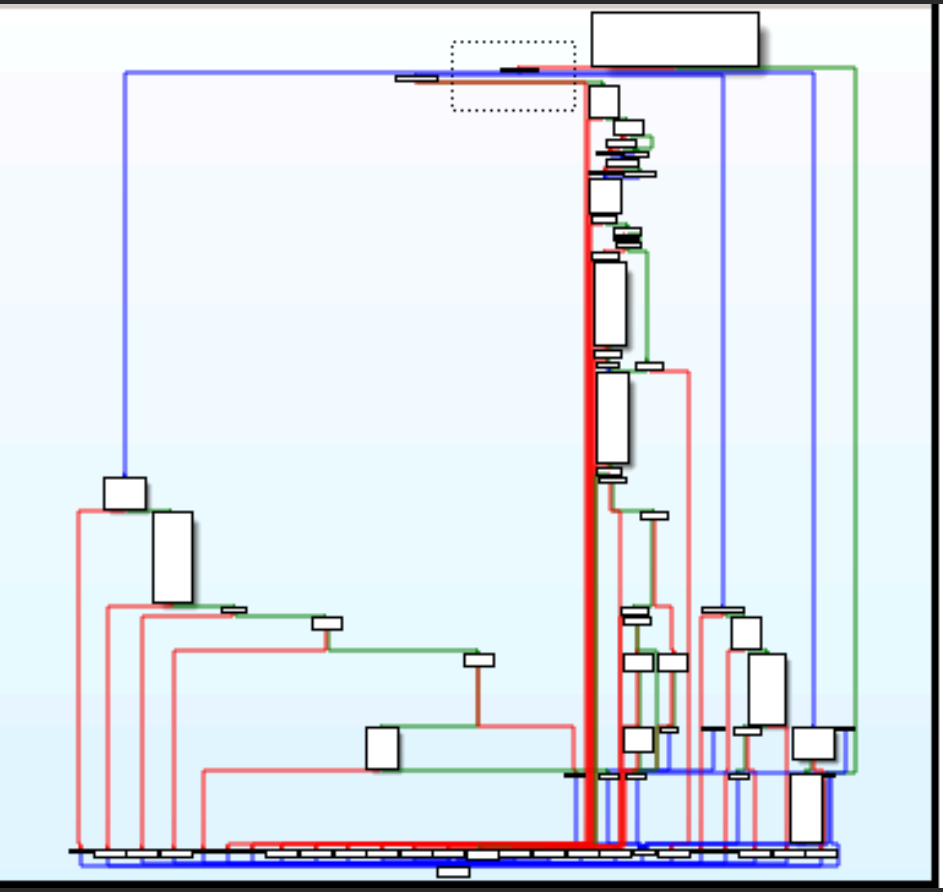
```
28    while (side_count < 2) {
29      index = 0;
30      while (index < 3) {
31        surface_area = surface_area + surfaces[index];
32        fprintf(stderr,"Intermediate Value: %lu\n" ,surface_area);
33        index = index + 1;
34      }
```

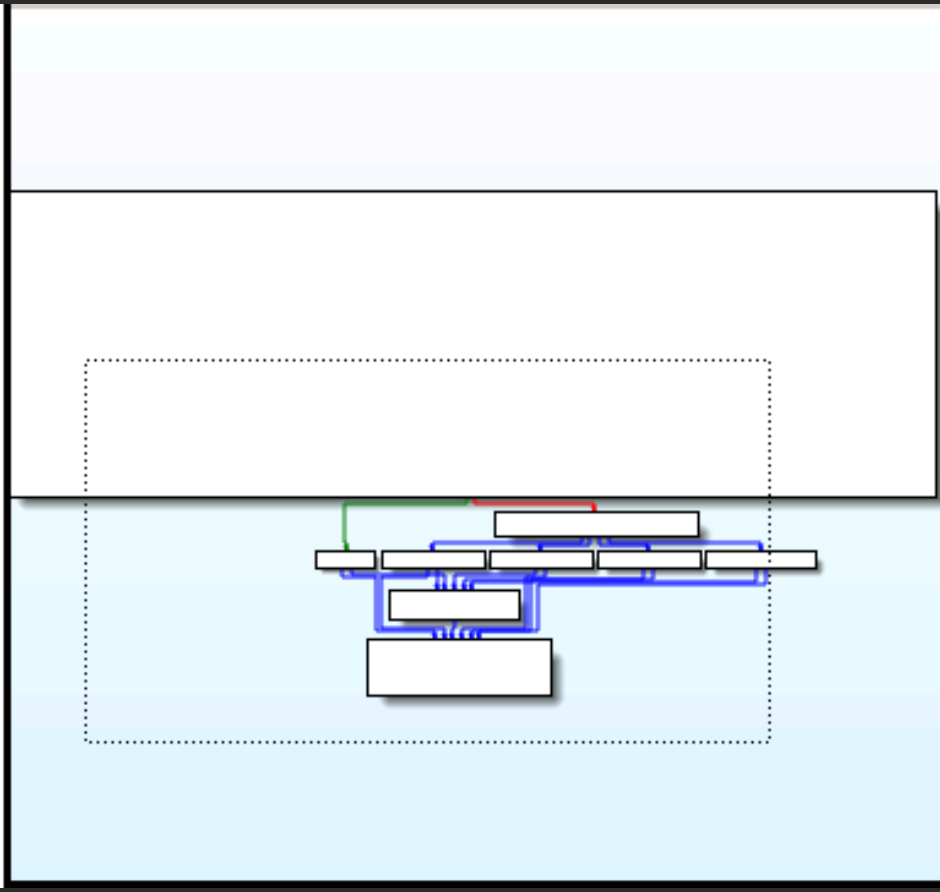
```
28    while (side_count < 2) {
29      index = 0;
30      while (index < 3) {
31        surface_area = surface_area + surfaces[index];
32        fprintf(stderr,"Intermediate Value: %lu\n" ,surface_area);
33        index = index + 1;
34      }
```

Ghidra’s Forward Slices

An Aside
Goals & Motivation
Prior Work
Getting Started
Staying Oriented
Cutting Deeper
Staying Organized
Dominance
Finding Flows
ILs
Variable Slices
Intraprocedural
Interprocedural
Conclusion

IMAGE CREDIT TRIPWIRE.COM





SIMPLIFYGRAPH BY MANDIANT FOR IDA

IMAGE CREDIT SIMPLIFYGRAPH

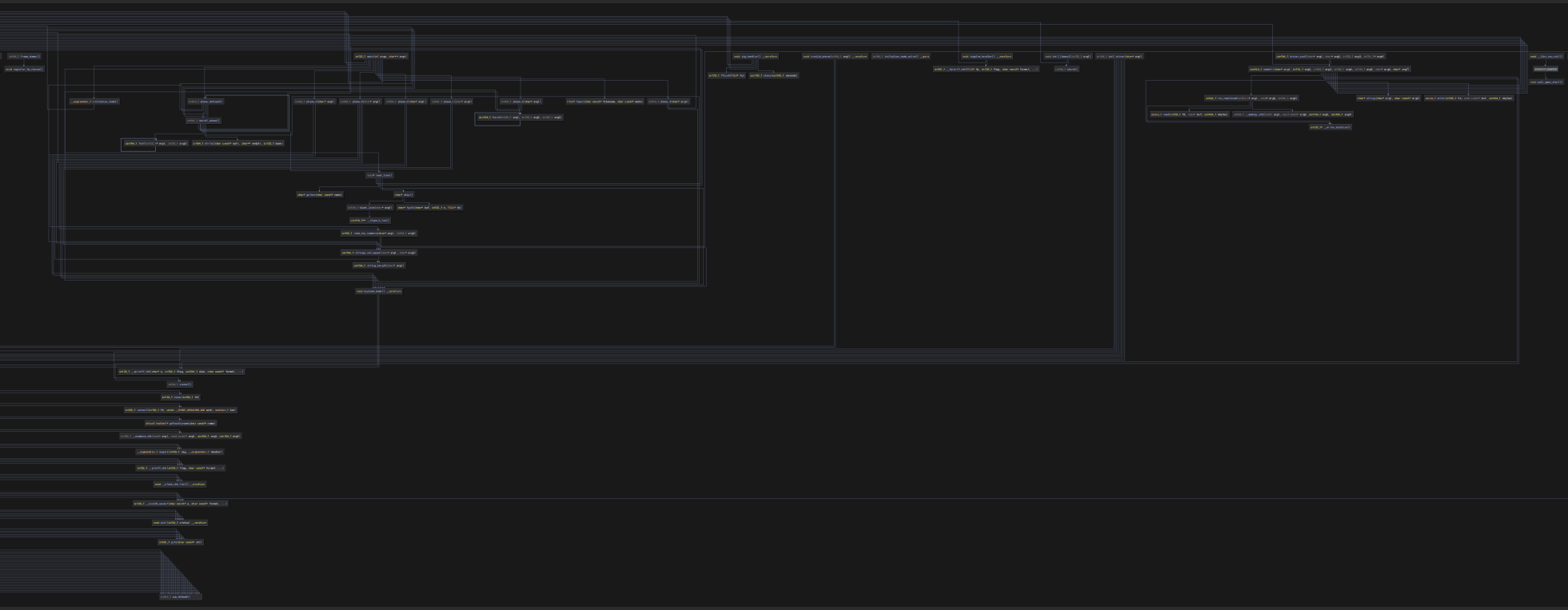
Tons of creative slice types

- Variable Based
- Simplification
- Informative
- Interactive
- Call Graphs
- Control Flow
- Dataflow
- Intra- vs Inter- Procedural

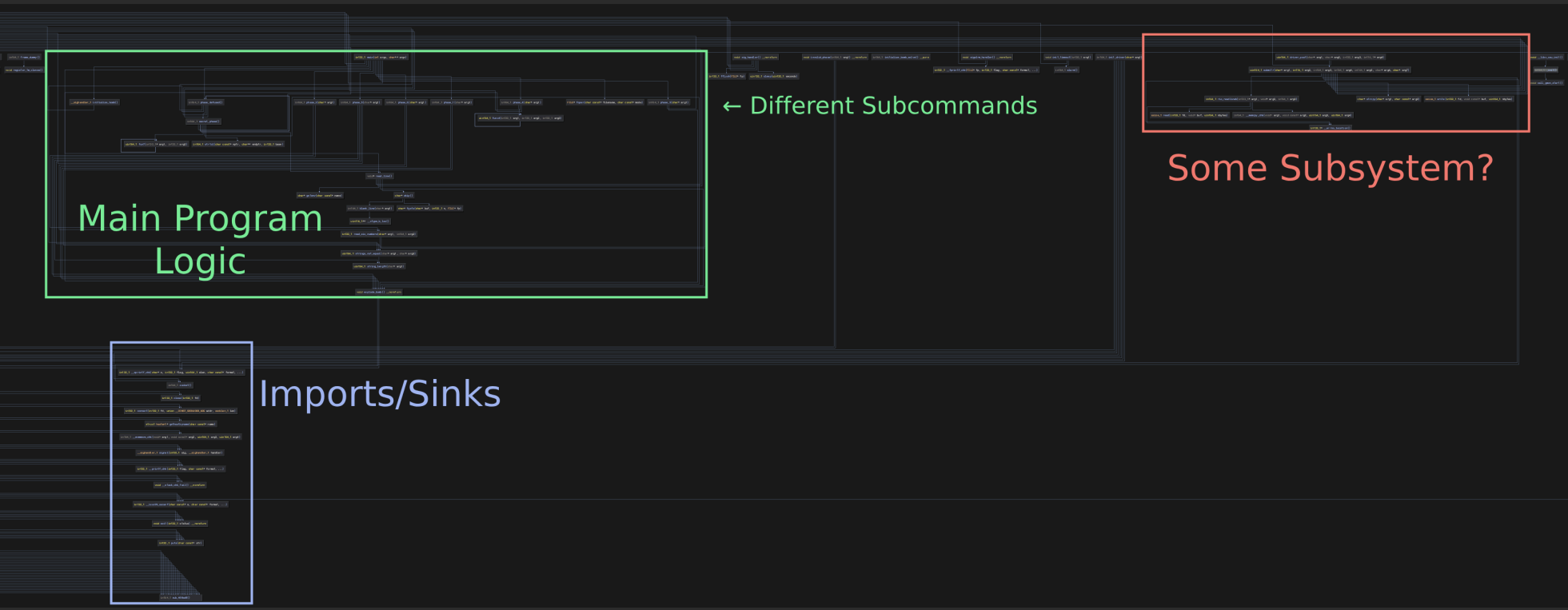
✨ 🌈 Anything you can imagine! 🌈 ✨

An Aside
Goals & Motivation
Prior Work
Getting Started
Staying Oriented
Cutting Deeper
Staying Organized
Dominance
Finding Flows
ILs
Variable Slices
Intraprocedural
Interprocedural
Conclusion

PROBLEM 1

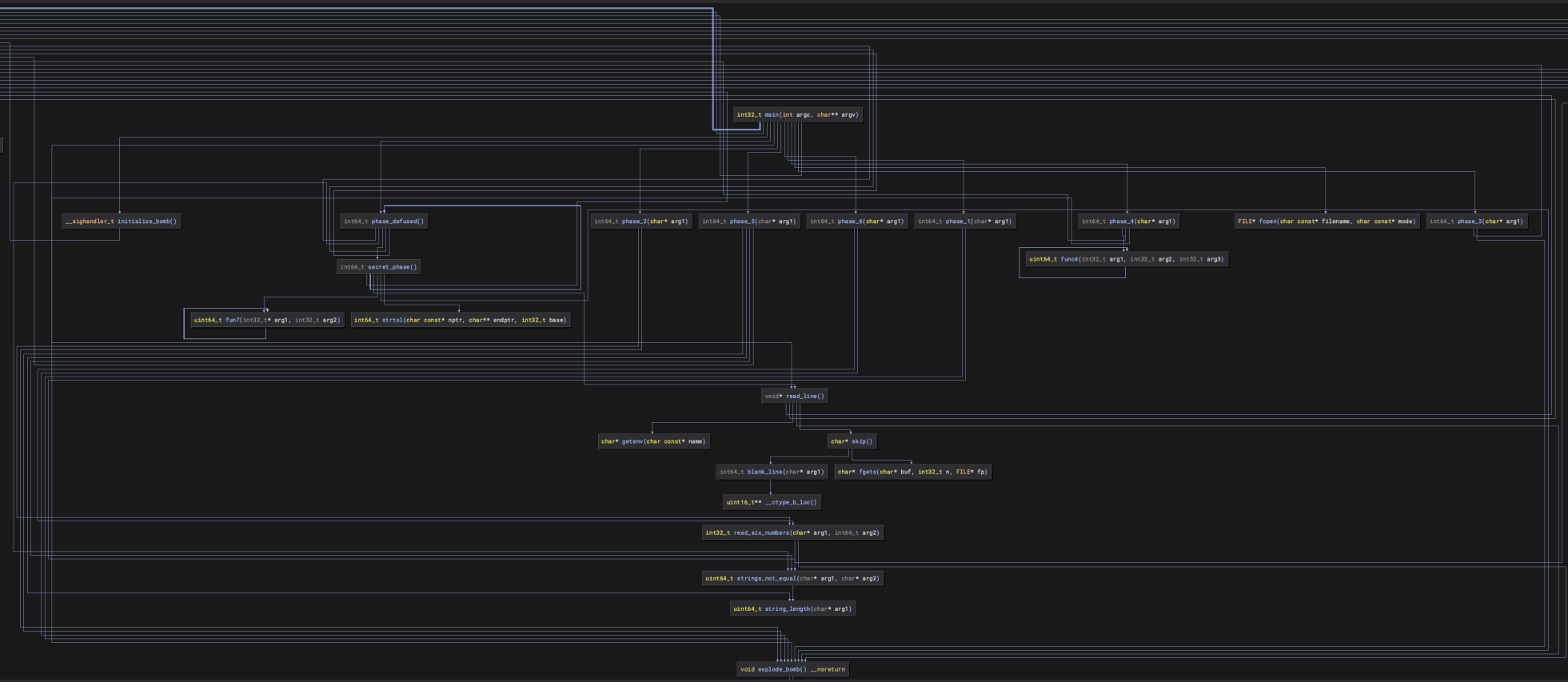


FULL CALL GRAPH



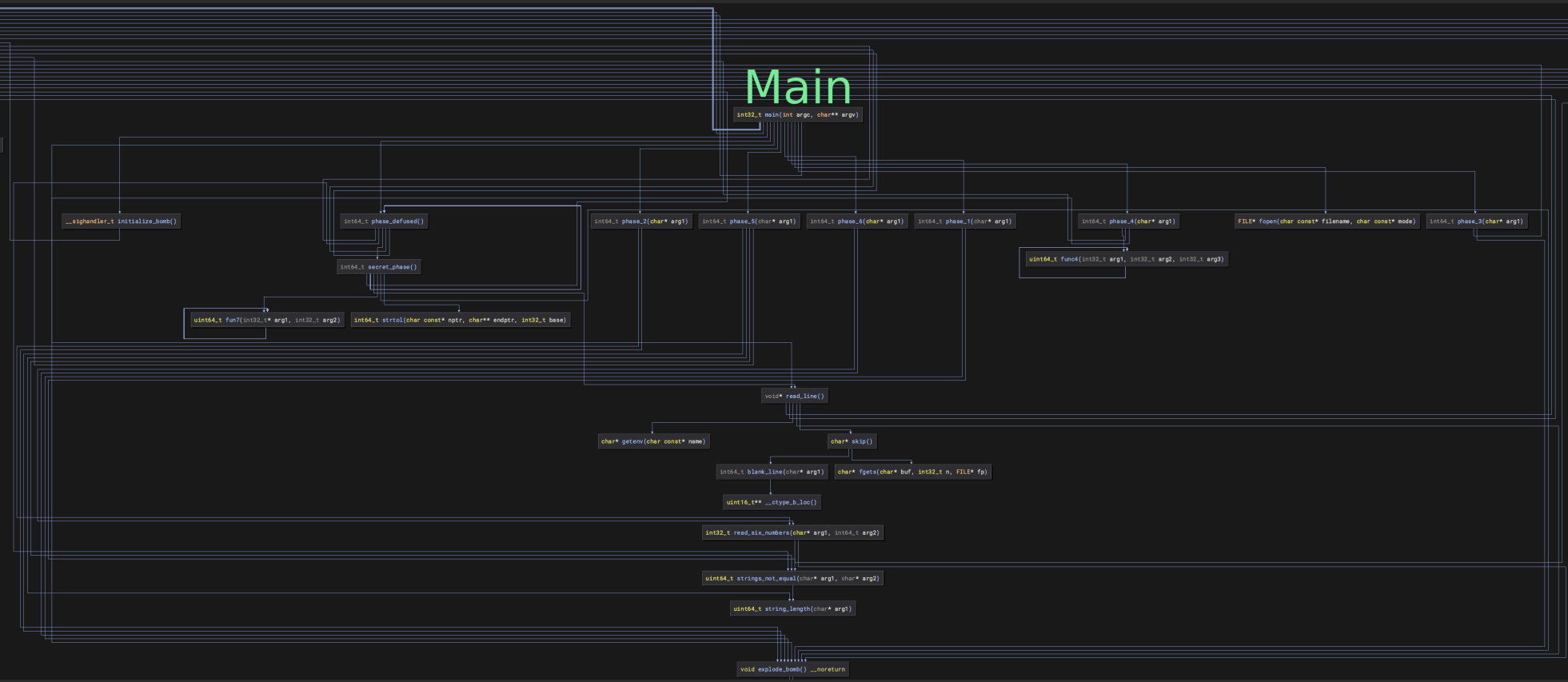
FULL CALL GRAPH

PATTERNS



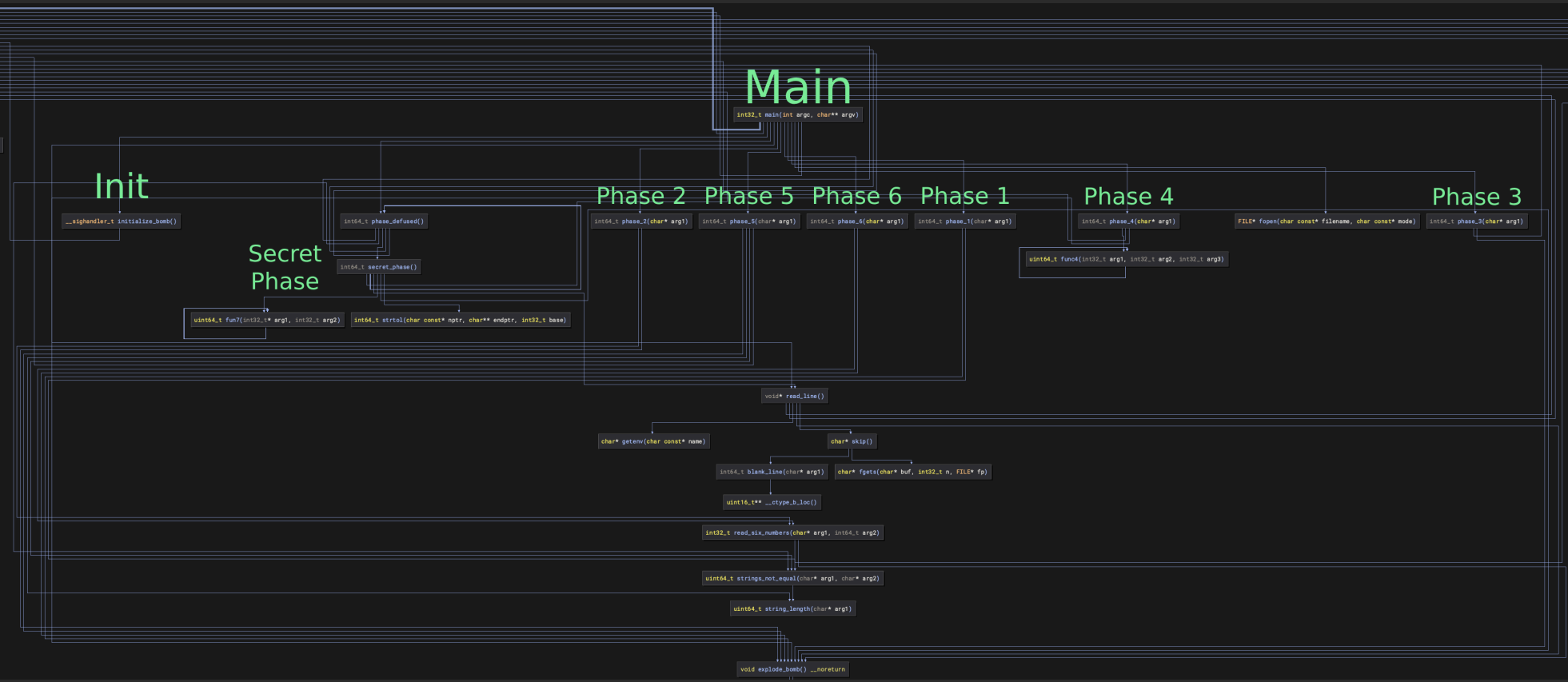
FULL CALL GRAPH

COMMAND DISPATCH



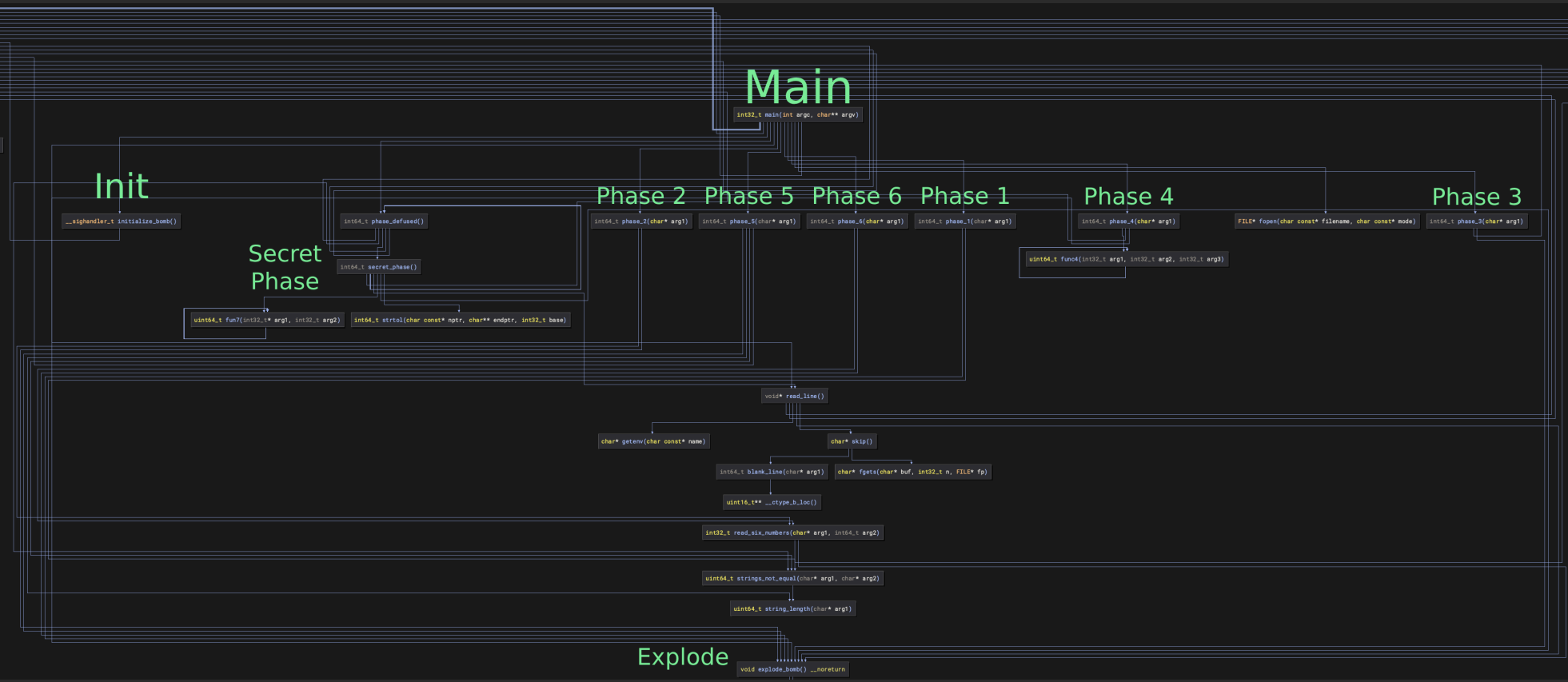
FULL CALL GRAPH

REVERSING MAIN



FULL CALL GRAPH

RECOGNIZING PROGRAM STRUCTURE



FULL CALL GRAPH

UNDERSTANDING PROGRAM BEHAVIOR

FULL CALL GRAPH

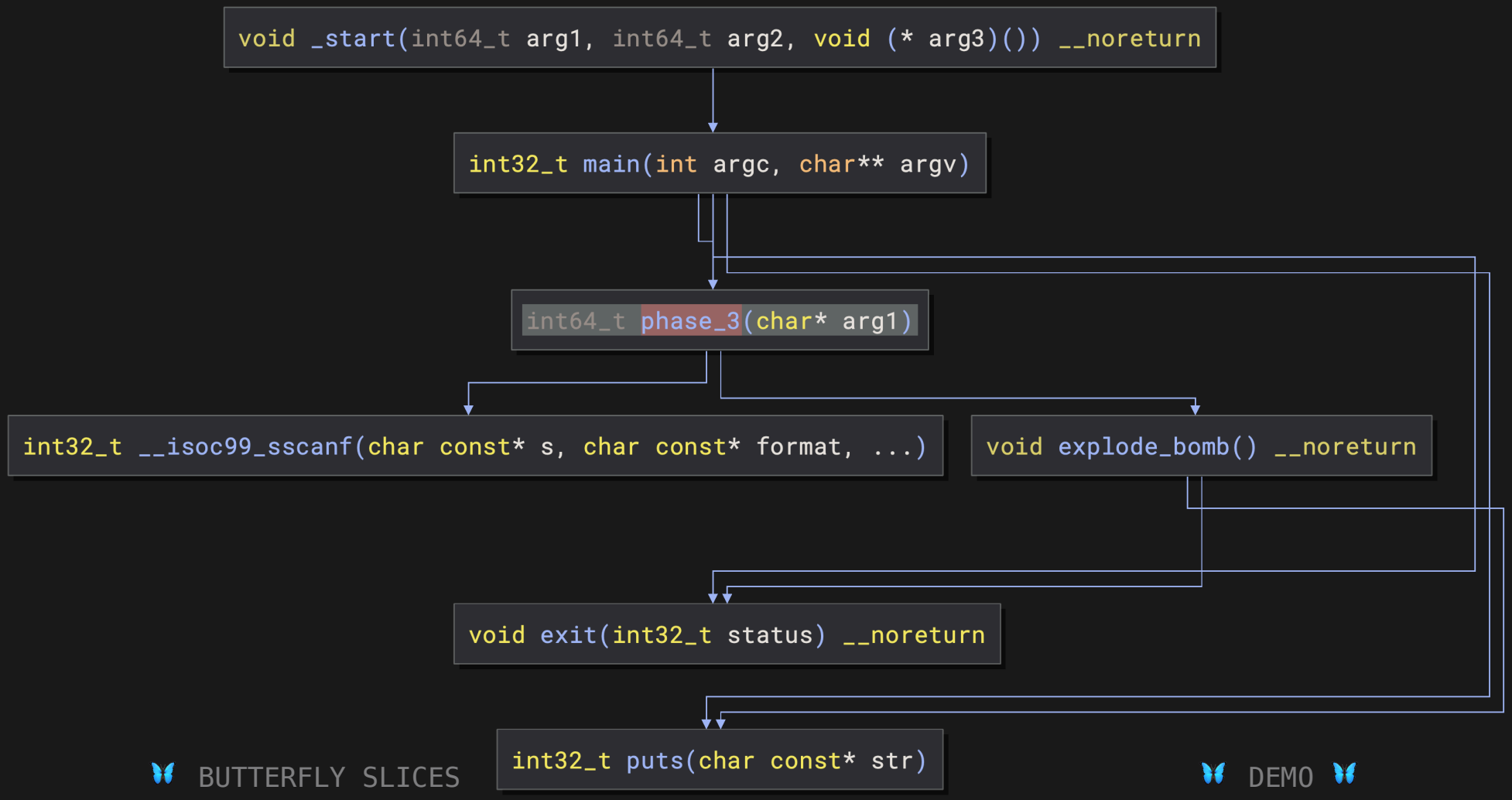
DEMO TIME

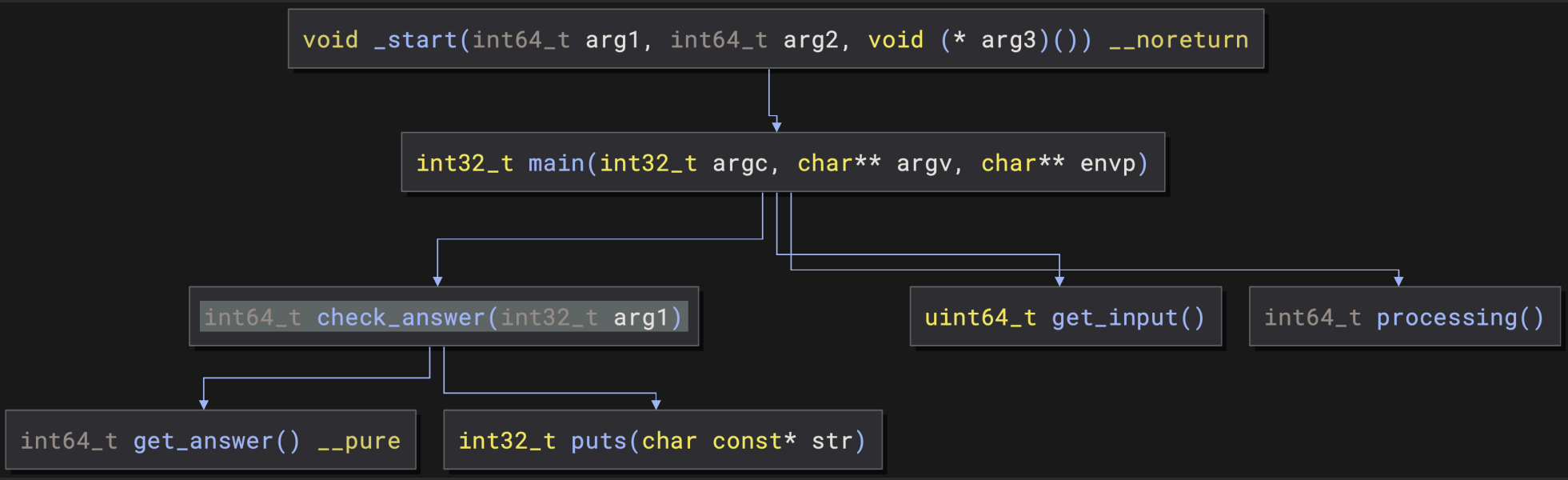
```python
import tanto
from binaryninja import FlowGraph, FlowGraphNode, DisassemblyTextLine, BranchType

class FullCallGraph(tanto.slices.Slice):
  def __init__(self, parent: 'tanto.tanto_view.TantoView'):
    self.bv = parent.bv
    self.navigation_style = tanto.slices.NavigationStyle.FUNCTION_START

  def get_flowgraph(self) -> FlowGraph:
    flowgraph = FlowGraph()
    nodes = {}

    # Add Nodes & Lines
    for function in self.bv.functions:
      new_node = FlowGraphNode(flowgraph)
      new_node.lines = [DisassemblyTextLine(function.type_tokens, function.start)]
      flowgraph.append(new_node)
      nodes[f"{function.symbol.full_name}@{function.start}"] = new_node

    # Stitch Nodes Together
    for function in self.bv.functions:
      node = nodes[f"{function.symbol.full_name}@{function.start}"]
      for edge in set(function.callees):
        node.add_outgoing_edge(BranchType.UnconditionalBranch, nodes[f"{edge.symbol.full_name}@{edge.start}"])
    return flowgraph

tanto.tanto_view.TantoView.register_slice_type("Full Call Graph", FullCallGraph)
```

Butterfly & Scatter Slices

PROBLEM 2

An Aside
Goals & Motivation
Prior Work
Getting Started
Staying Oriented
Cutting Deeper
Staying Organized
Dominance
Finding Flows
ILs
Variable Slices
Intraprocedural
Interprocedural
Conclusion


void _start(int64_t arg1, int64_t arg2, void (* arg3)()) __noreturn
int32_t main(int argc, char** argv)
int64_t phase_3(char* arg1)
int32_t __isoc99_sscanf(char const* s, char const* format, ...)
void explode_bomb() __noreturn
void exit(int32_t status) __noreturn
int32_t puts(char const* str)


void _start(int64_t arg1, int64_t arg2, void (* arg3)()) __noreturn
int32_t main(int32_t argc, char** argv, char** envp)
int64_t check_answer(int32_t arg1)
uint64_t get_input()
int64_t processing()
int64_t get_answer() __pure
int32_t puts(char const* str)

An Aside
Goals & Motivation
Prior Work
Getting Started
Staying Oriented
Cutting Deeper
Staying Organized
Dominance
Finding Flows
ILs
Variable Slices
Intraprocedural
Interprocedural
Conclusion

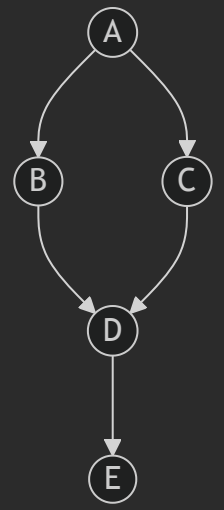
SOURCE-TO-SINK SLICES

An Aside
Goals & Motivation
Prior Work
Getting Started
Staying Oriented
Cutting Deeper
Staying Organized
Dominance
Finding Flows
ILs
Variable Slices
Intraprocedural
Interprocedural
Conclusion

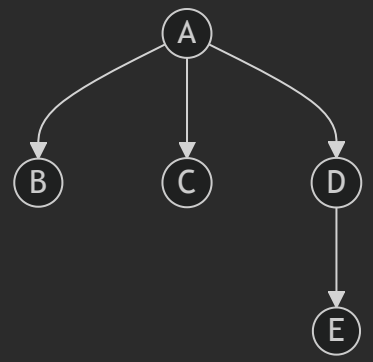
MOVING BEYOND CALL RELATIONSHIPS

An Aside
Goals & Motivation
Prior Work
Getting Started
Staying Oriented
Cutting Deeper
Staying Organized
Dominance
Finding Flows
ILs
Variable Slices
Intraprocedural
Interprocedural
Conclusion

`A` dominates `B`



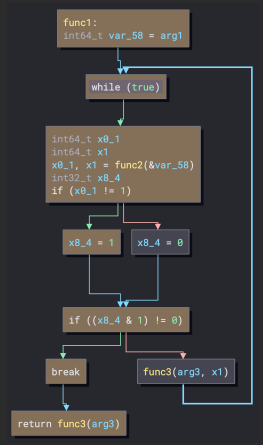
`D` dominates `E`
`A` dominates `D`
`A` dominates `B`
`A` dominates `C`
`B` and `C` dominate nothing


A
B
C
D
E

Dominator Tree Children

Dominance Frontier

Post Dominators



BASIC BLOCK SLICES

An Aside
Goals & Motivation
Prior Work
Getting Started
Staying Oriented
Cutting Deeper
Staying Organized
Dominance
Finding Flows
ILs
Variable Slices
Intraprocedural
Interprocedural
Conclusion

BNIL

An Aside
Goals & Motivation
Prior Work
Getting Started
Staying Oriented
Cutting Deeper
Staying Organized
Dominance
Finding Flows
ILs
Variable Slices
Intraprocedural
Interprocedural
Conclusion

```
mov  rax, qword [rbp-0×450]
inc  rax
mov  qword [rbp-0×450], rax
mov  r12, qword [r13+rax*8]
test r12, r12
jne  0×358e
```

↓

```
rax = [rbp - 0×450].q
rax = rax + 1
[rbp - 0×450].q = rax
r12 = [r13 + (rax << 3)].q
if (r12 ≠ 0) then 20 else 103
```

↓

```
int64_t rax_26 = var_458
int64_t rax_27 = rax_26 + 1
int64_t var_458 = rax_27
int64_t r12 = [r13 + (rax_27 << 3)].q
if (r12 ≠ 0) then 9 else 78
```

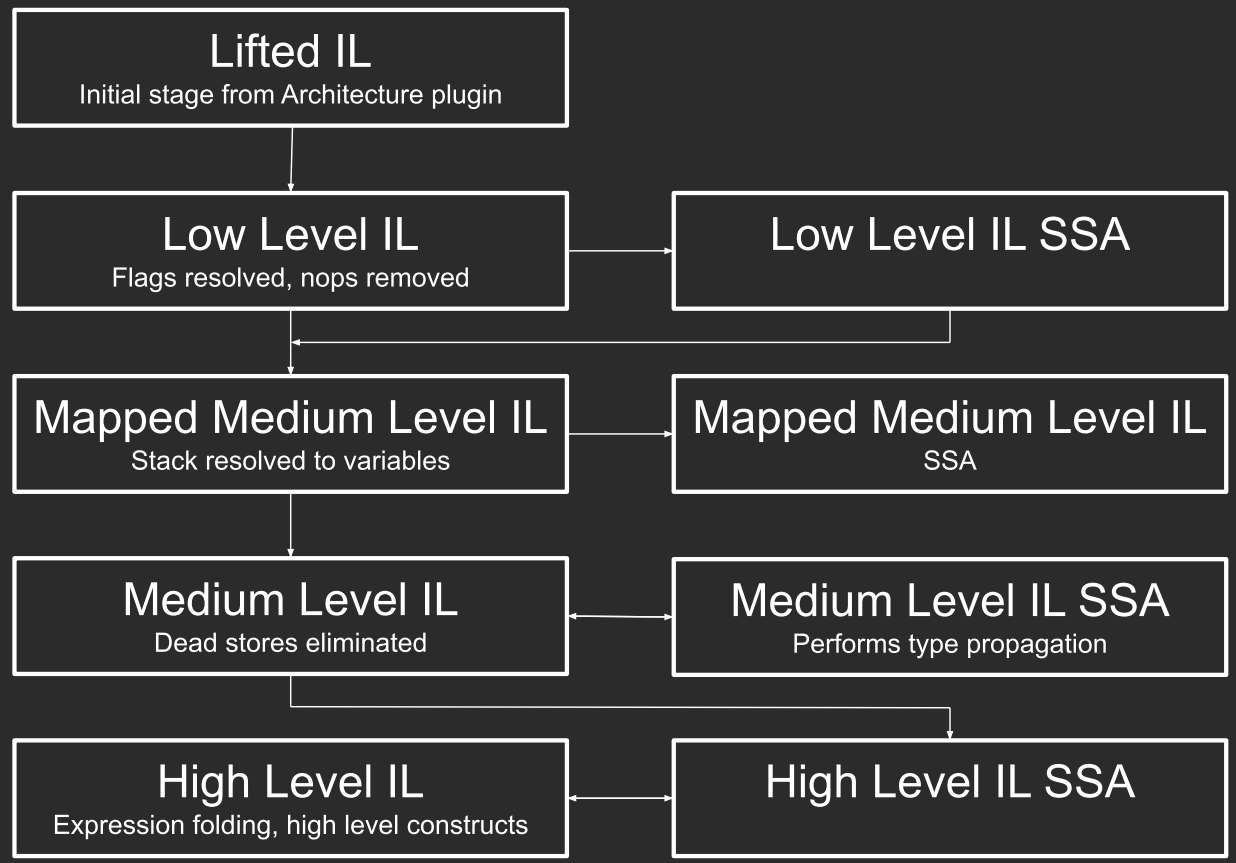
↓

```
int64_t rax_27 = var_458 + 1
var_458 = rax_27
r12 = r13[rax_27]
do while (r12 ≠ 0)
```

BNIL

- Binary Ninja ILs (BNIL)
- Layered
- Tree-based
- Interactive
- Deferred flags
- Extremely machine-readable (API)
- Extremely human-readable
- SSA, non-SSA, AST, mappings, oh my!

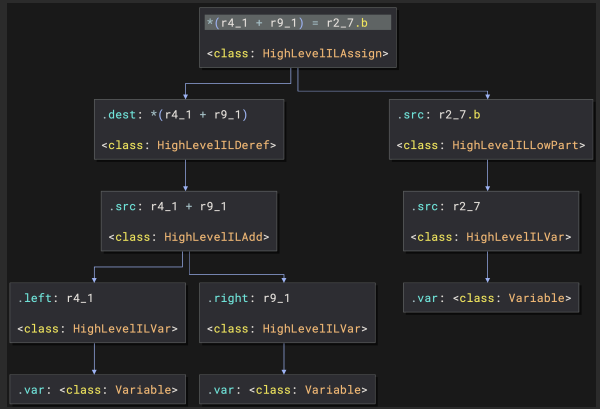
FEATURES



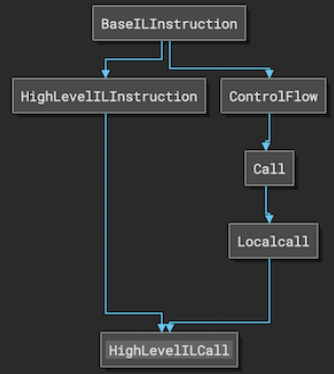
BNIL

LAYERED ILS

```python
for func in bv.functions:
  for basic_block in func.hlil:
    for inst in basic_block:
      print(f"{str(inst.operation)}: {inst}")
```



Instruction Slice

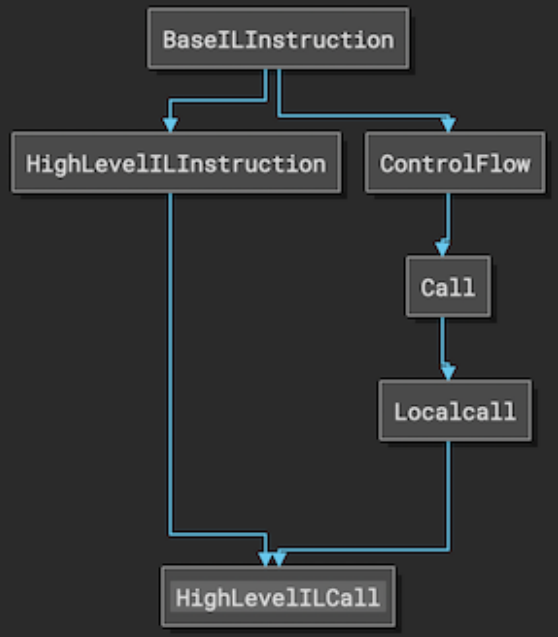


Hierarchy Slice

```python
def traverser(inst) -> int: # return whatever data you want!
  if isinstance(inst, Constant):
    return inst.constant # Yield this constant then keep recursing

# Finds all constants used in the program
for inst in bv.hlil_instructions:
  # Don't forget to actually yield the generator!!
  for const in inst.traverse(traverser):
    print(f"Found constant {const}")
```

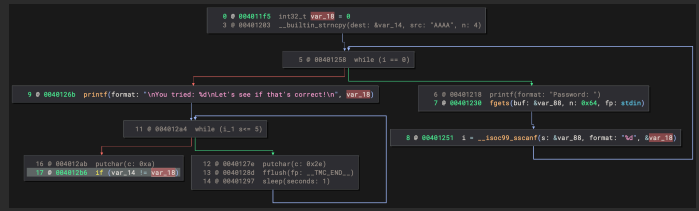
Tree Data Structure == Recursive Visitors

```python
from binaryninja.commmonil import Call, Localcall

for inst in current_hlil.instructions:
  if inst.operation == HighLevelILOperation.HLIL_TAILCALL:
    print(f"{inst} is a tail call")

  if isinstance(inst, Call):
    print(f"{inst} is a Call of some sort")
  if isinstance(inst, Localcall):
    print(f"{inst} is not a syscall!")
```



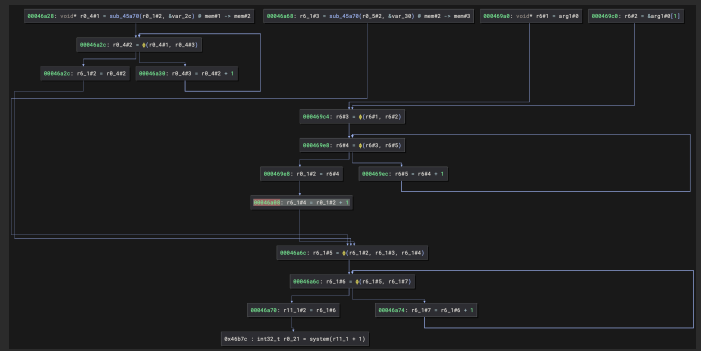
THERE ARE TREES IN THE TREES

`.OPERATION` VS `CLASS`



An Aside
Goals & Motivation
Prior Work
Getting Started
Staying Oriented
Cutting Deeper
Staying Organized
Dominance
Finding Flows
ILs
Variable Slices
Intraprocedural
Interprocedural
Conclusion

SLICE TYPES FOR THE PEOPLE!



An Aside

Goals & Motivation

Prior Work

Getting Started

Staying Oriented

Cutting Deeper

Staying Organized

Dominance

Finding Flows

ILs

Variable Slices

Intraprocedural

Interprocedural

Conclusion

An Aside
Goals & Motivation
Prior Work
Getting Started
Staying Oriented
Cutting Deeper
Staying Organized
Dominance
Finding Flows
ILs
Variable Slices
Intraprocedural
Interprocedural
Conclusion

What’s left to be solved?

What problems did we solve?

An Aside
Goals & Motivation
Prior Work
Getting Started
Staying Oriented
Cutting Deeper
Staying Organized
Dominance
Finding Flows
ILs
Variable Slices
Intraprocedural
Interprocedural
Conclusion

Slice on Edges

Configurable Options (Slice Depth)

Save slices to Database

Include Indirect Flows in CFGs

Track Out-Paramters

Automate More?

Integrate Simplifications?

Better Visualizations? Graphs in Graphs?

A dozen more slice ideas, at least!

Reduced Discovery Time

Read Less Code – or None

Identified POIs and Sub-Programs

Automated Source-to-Sink Graphs

Improved Binary Exploration/Navigation

Control Flow and Dataflow Slicing

Understanding Dominance

Working with the IL

…so much more!

Thank you for listening to

"*A Slice of*" Modern Program Analysis!

Experiement with Tantō 2.0

github.com/Vector35/Tanto

^slides in repo^

@ELYK / @ELYKDEER