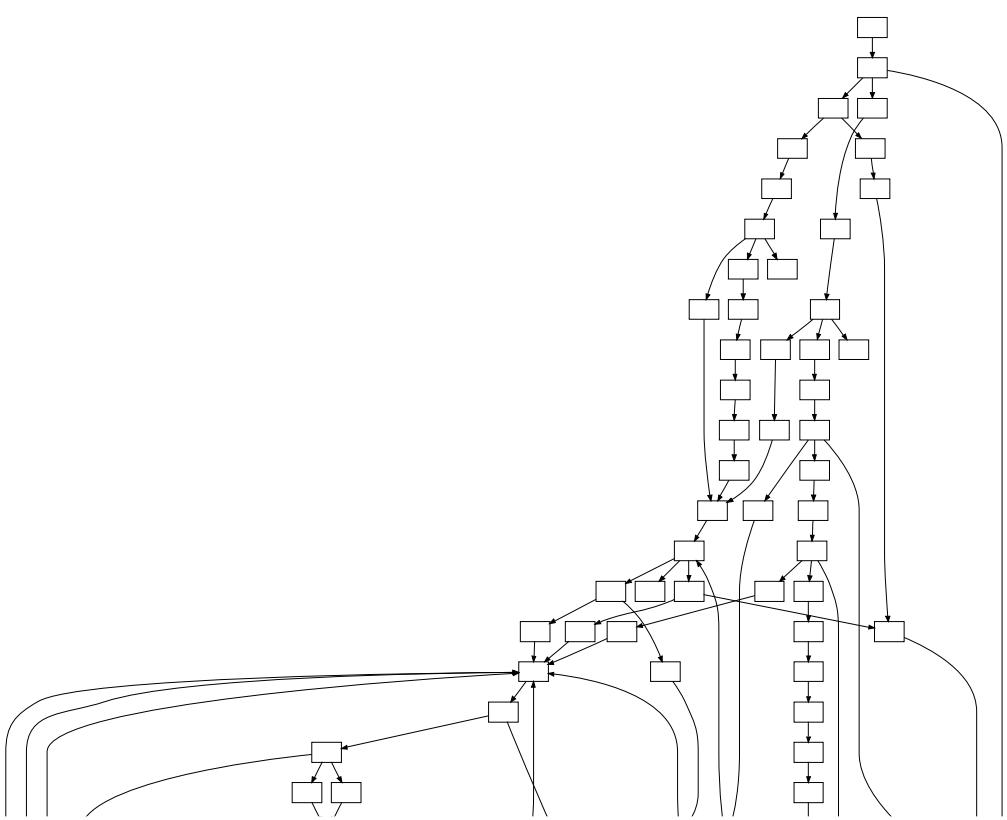


```
29  L_004b:  stloc.s     V_4
30  L_004d:  ldloc.3
31  L_004e:  callvirt    System.Void System.Diagnos
32  L_0053:  ldstr       "Difference: "
33  L_0058:  ldloc.3
34  L_0059:  callvirt    System.Int64 System.Diagn                    illiseconds()
35  L_005e:  stloc.s     V_10
36  L_0060:  ldloca.s    V_10
37  L_0062:  call        System.String System.Int64
38  L_0067:  call        System.String System.String                  String)
39  L_006c:  call        System.Void System.Console::WriteLine(System.String)
40  L_0071:  ldloc.0
41  L_0072:  ldloc.3
42  L_0073:  callvirt    System.Int64 System.Diagnostics.Stopwatch::get_ElapsedMilliseconds()
43  L_0078:  stloc.s     V_11
44  L_007a:  ldloca.s    V_11
45  L_007c:  call        System.String System.Int64::ToString()
46  L_0081:  callvirt    System.Void System.IO.TextWriter::WriteLine(System.String)
47  L_0086:  ldloc.0
48  L_0087:  callvirt    System.Void System.IO.TextWriter::Close()
49  L_008c:  ldstr       "BeGood.BeAwesome.BeAFluxFinger!"
50  L_0091:  stloc.s     V_5
51  L_0093:  call        System.Text.Encoding System.Text.Encoding::get_ASCII()
52  L_0098:  ldloc.s     V_5
53  L_009a:  callvirt    System.Byte[] System.Text.Encoding::GetBytes(System.String)
54  L_009f:  stloc.s     V_6
55  L_00a1:  ldloc.2
56  L_00a2:  call        System.Byte[] System.IO.File::ReadAllBytes(System.String)
```

RUB

RUHR-UNIVERSITÄT BOCHUM

Probfuscation: An Obfuscation Approach using Probabilistic Control Flows

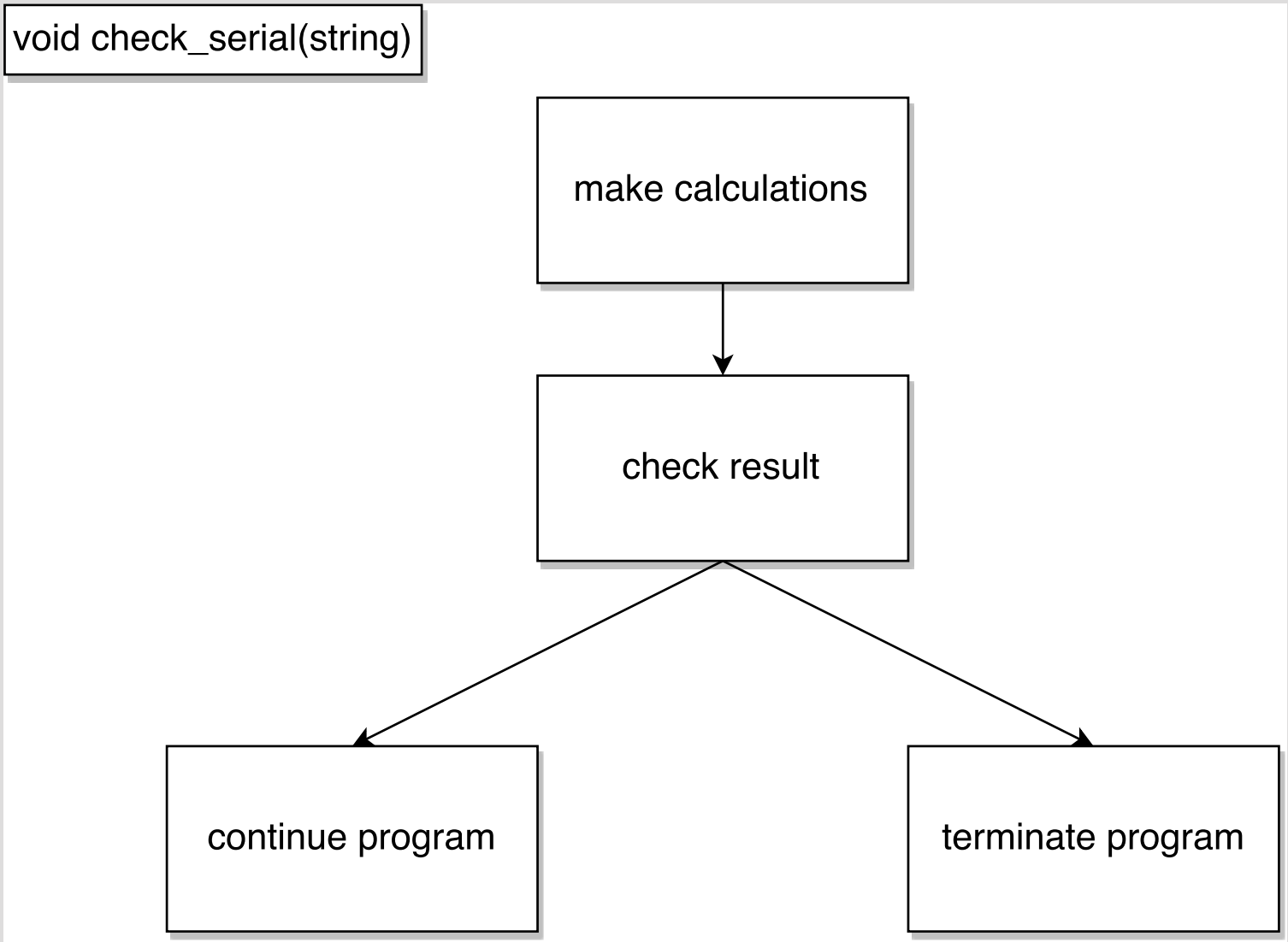
SIG SIDAR DIMVA 2016

July 7, 2016

Andre Pawlowski, Moritz Contag, Thorsten Holz

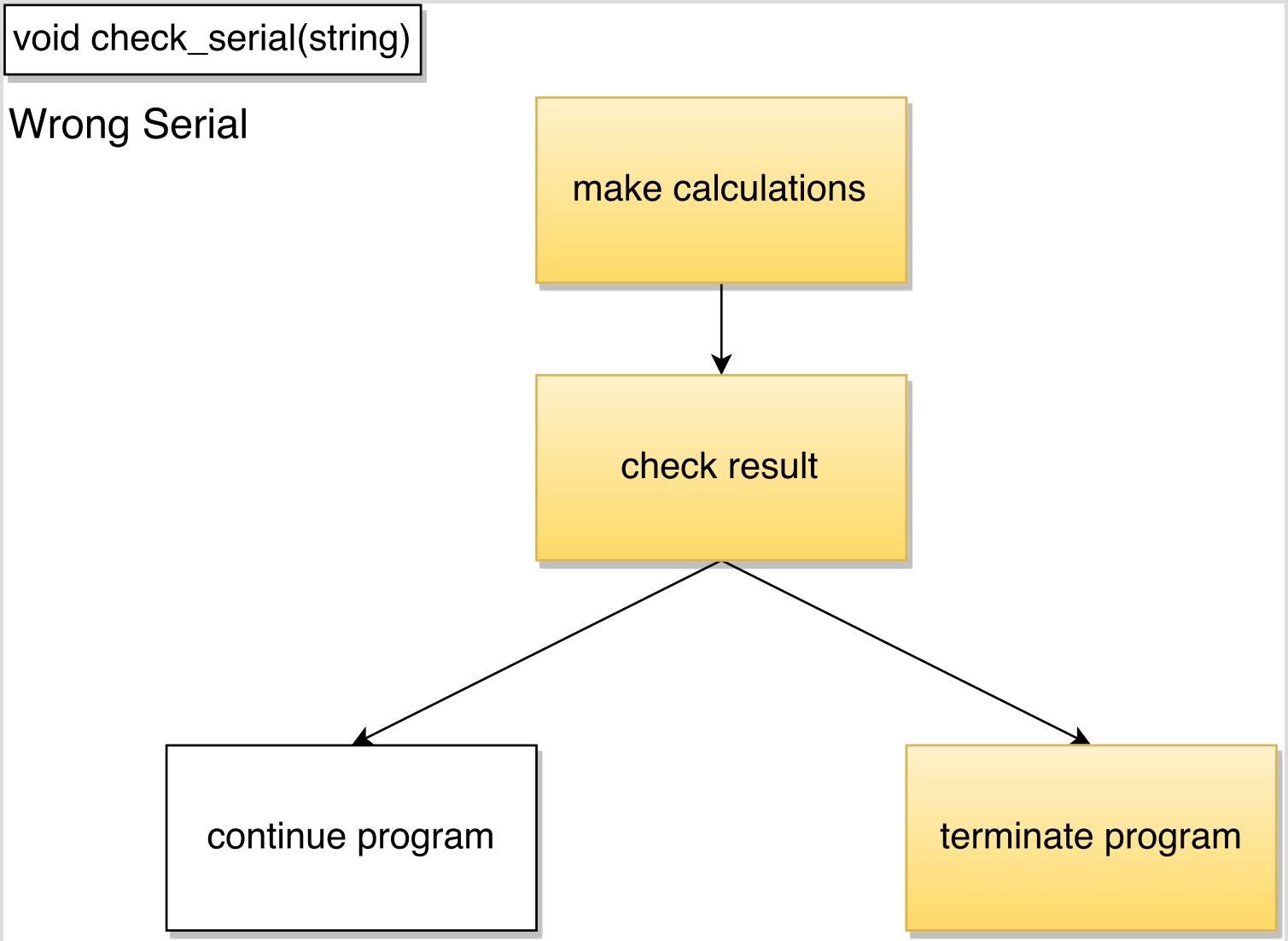
Introduction

Motivation



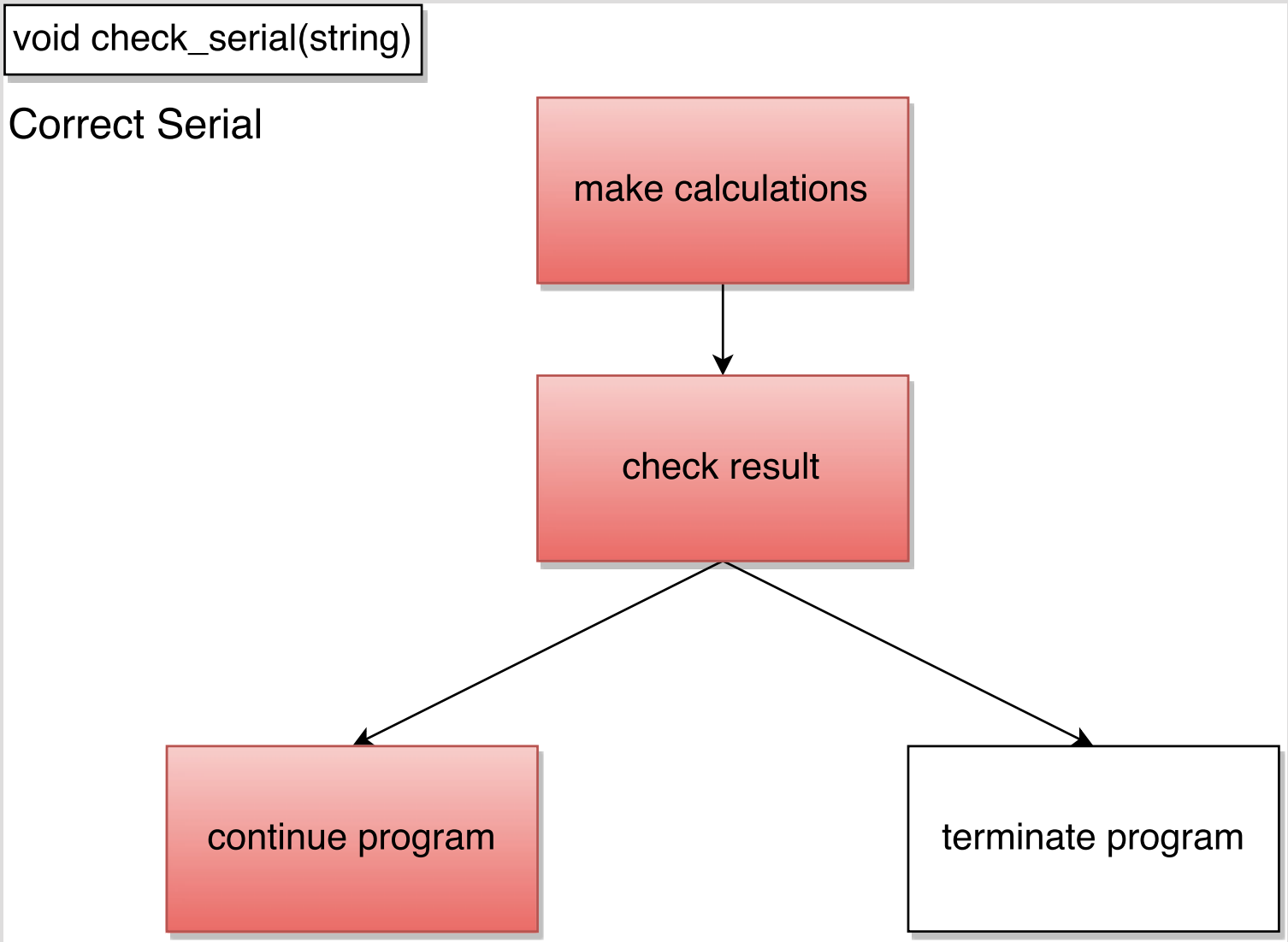
Introduction

Motivation



Introduction

Motivation



Introduction

Software Obfuscation

- Technique to make analysis of software harder
- Transforms program into hardly understandable code
- Use cases:
  - Protect intellectual property
  - Make analysis of Malware harder

Introduction

Software Obfuscation

- Technique to make analysis of software harder
- Transforms program into hardly understandable code
- Use cases:
  - Protect intellectual property
  - Make analysis of Malware harder

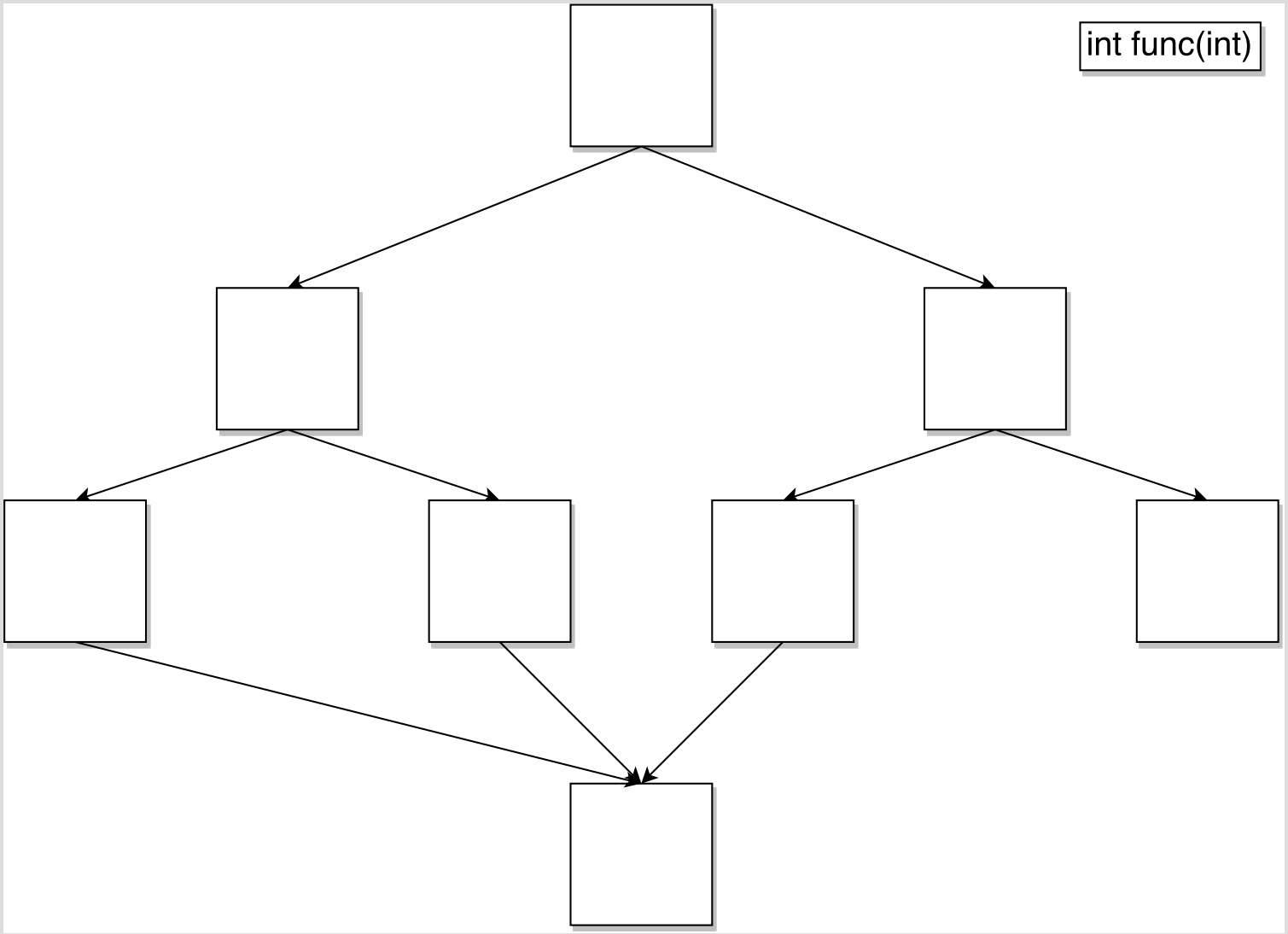
Introduction

Software Obfuscation

- Technique to make analysis of software harder
- Transforms program into hardly understandable code
- Use cases:
  - Protect intellectual property
  - Make analysis of Malware harder

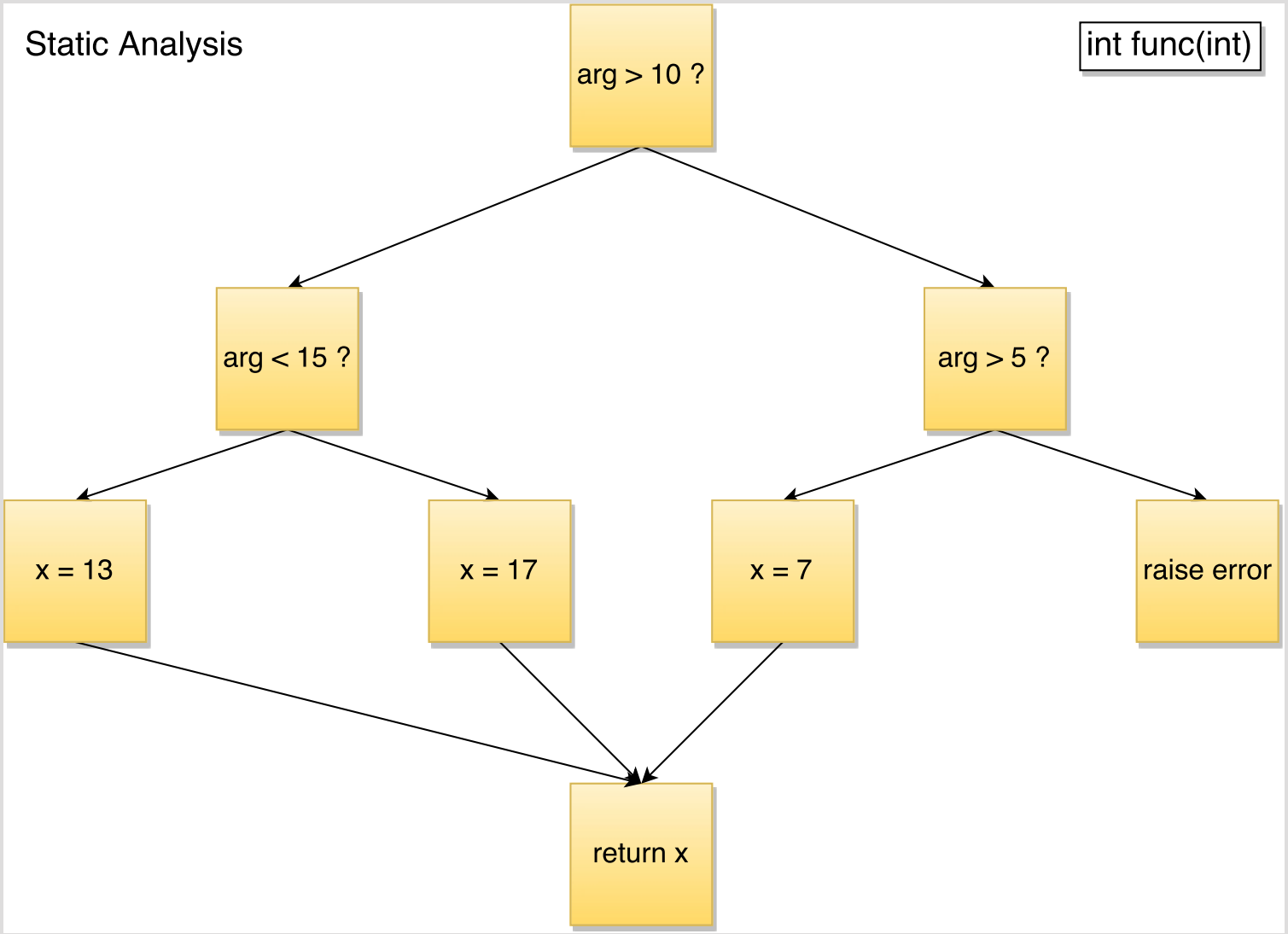
Introduction

Static vs. Dynamic Analysis



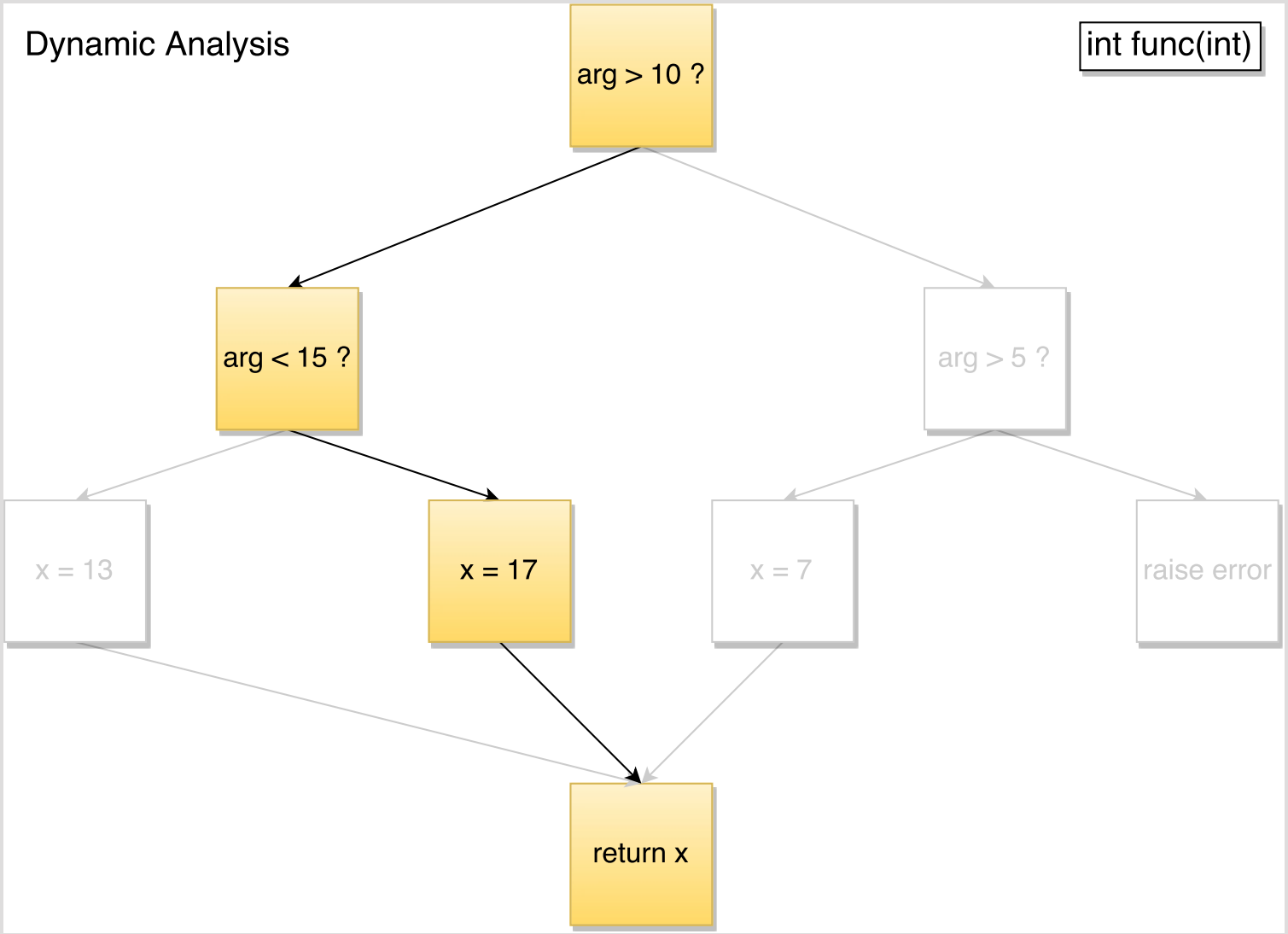
Introduction

Static vs. Dynamic Analysis



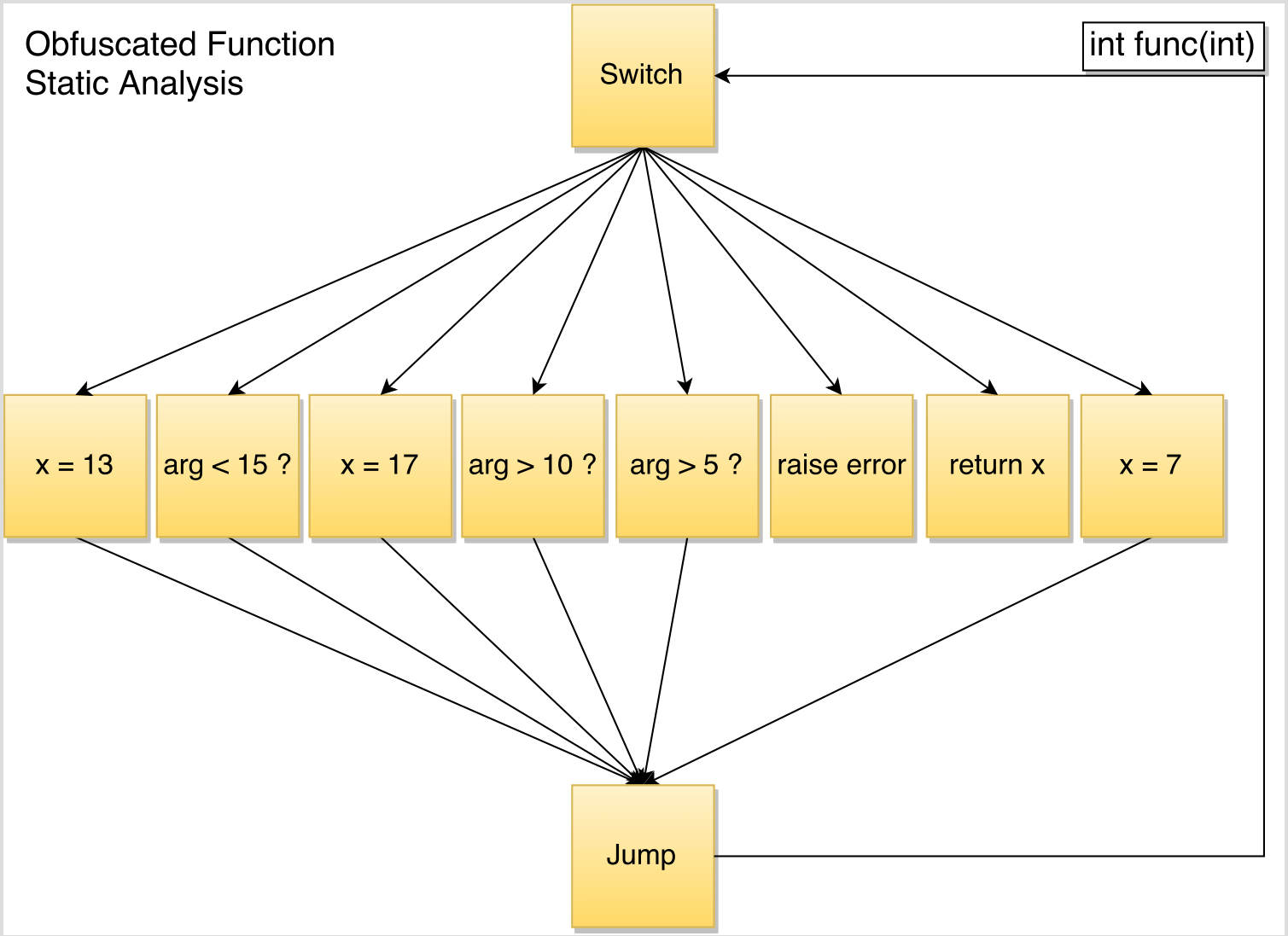
Introduction

Static vs. Dynamic Analysis



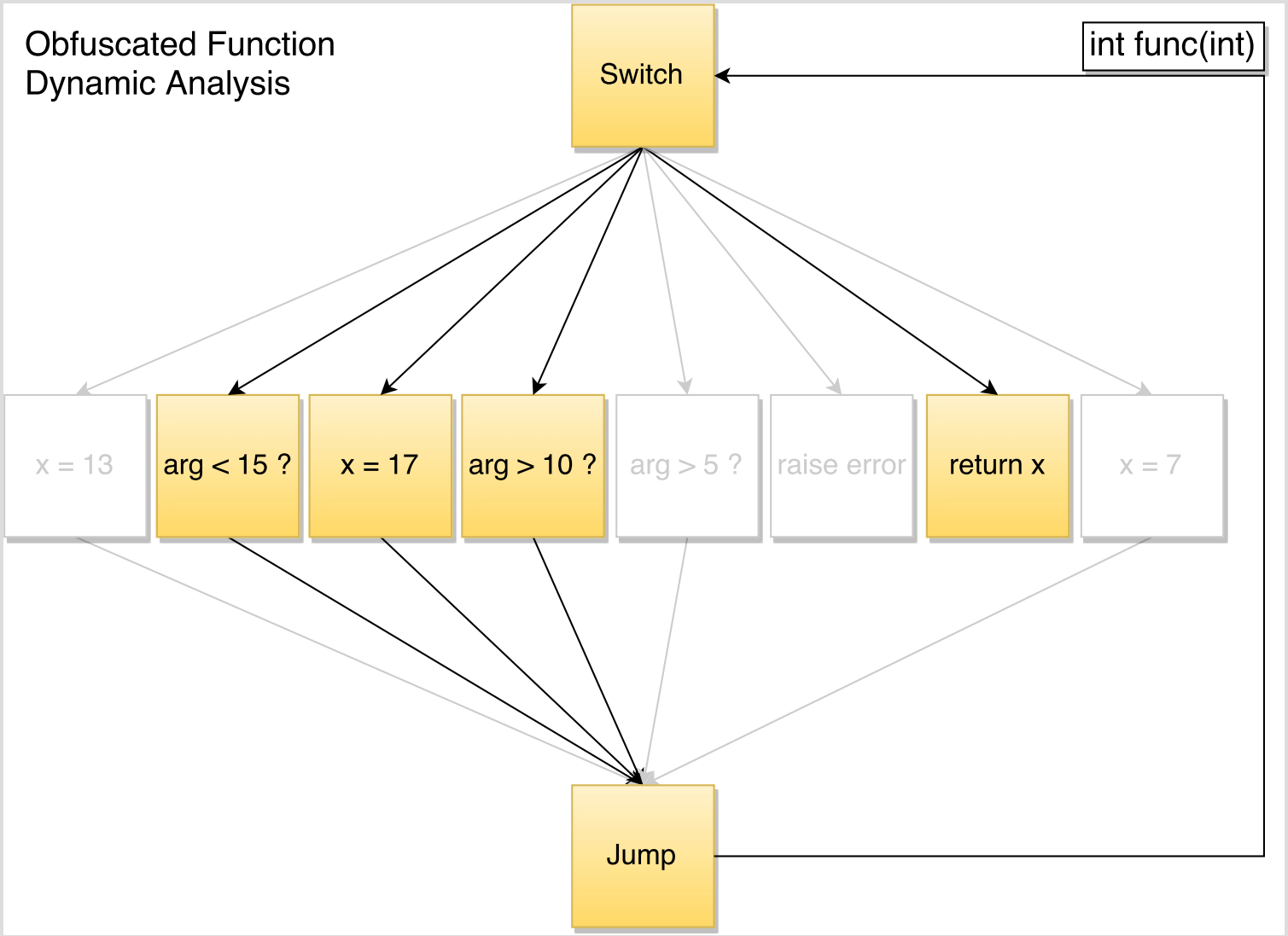
Introduction

Static vs. Dynamic Analysis



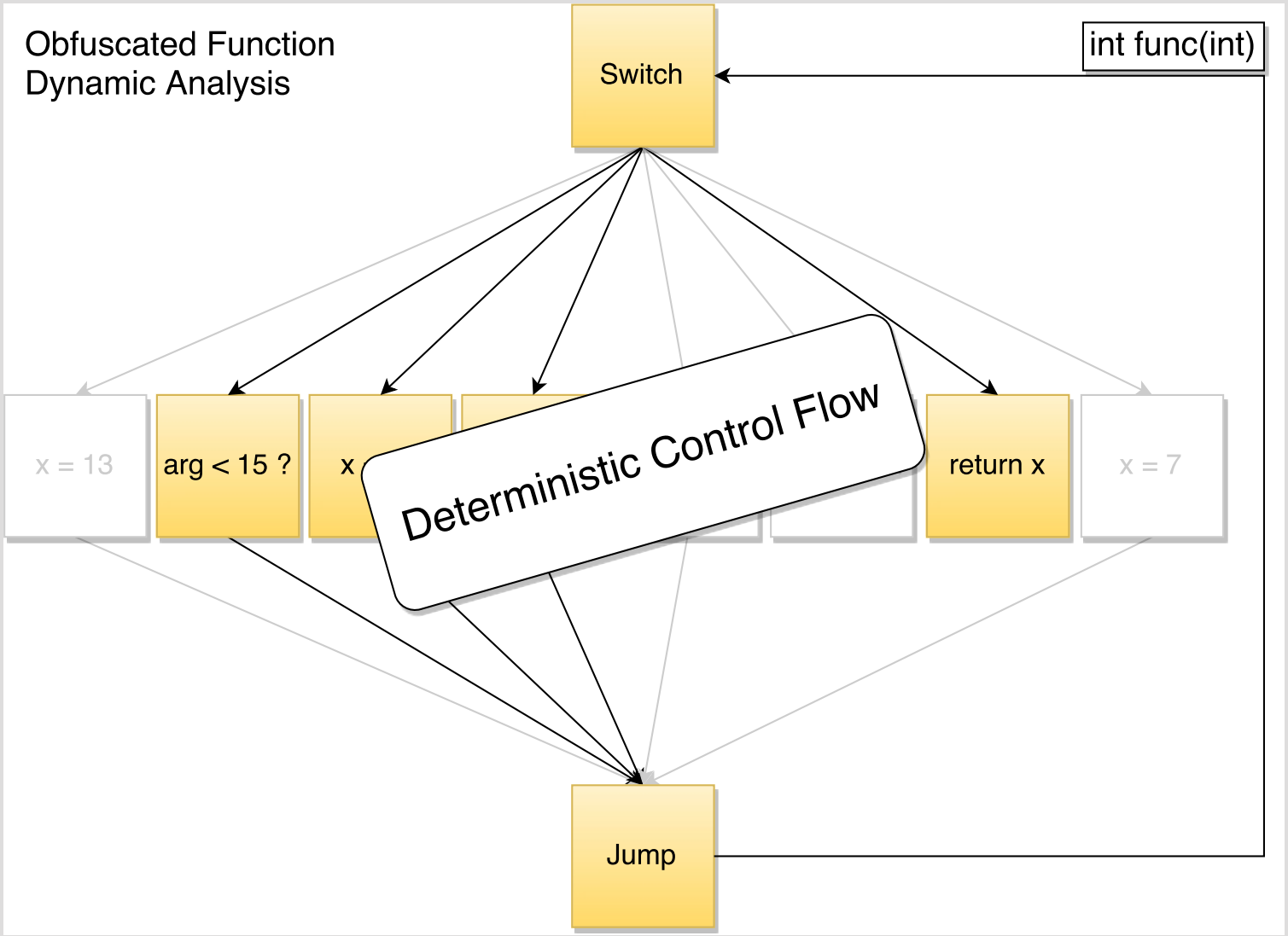
Introduction

Static vs. Dynamic Analysis



Introduction

Static vs. Dynamic Analysis



Introduction

Motivation

- State-of-the-art deobfuscation techniques use dynamic analysis

Introduction

Motivation

- State-of-the-art deobfuscation techniques use dynamic analysis
  - Use multiple traces to tackle code coverage problem

Introduction

Motivation

- State-of-the-art deobfuscation techniques use dynamic analysis
  - Use multiple traces to tackle code coverage problem
- Existing obfuscation techniques do not change determinism of control flow

Introduction

Motivation

- State-of-the-art deobfuscation techniques use dynamic analysis
  - Use multiple traces to tackle code coverage problem
- Existing obfuscation techniques do not change determinism of control flow

$\Rightarrow$ Idea: obfuscation focusing on dynamic analysis

Overview

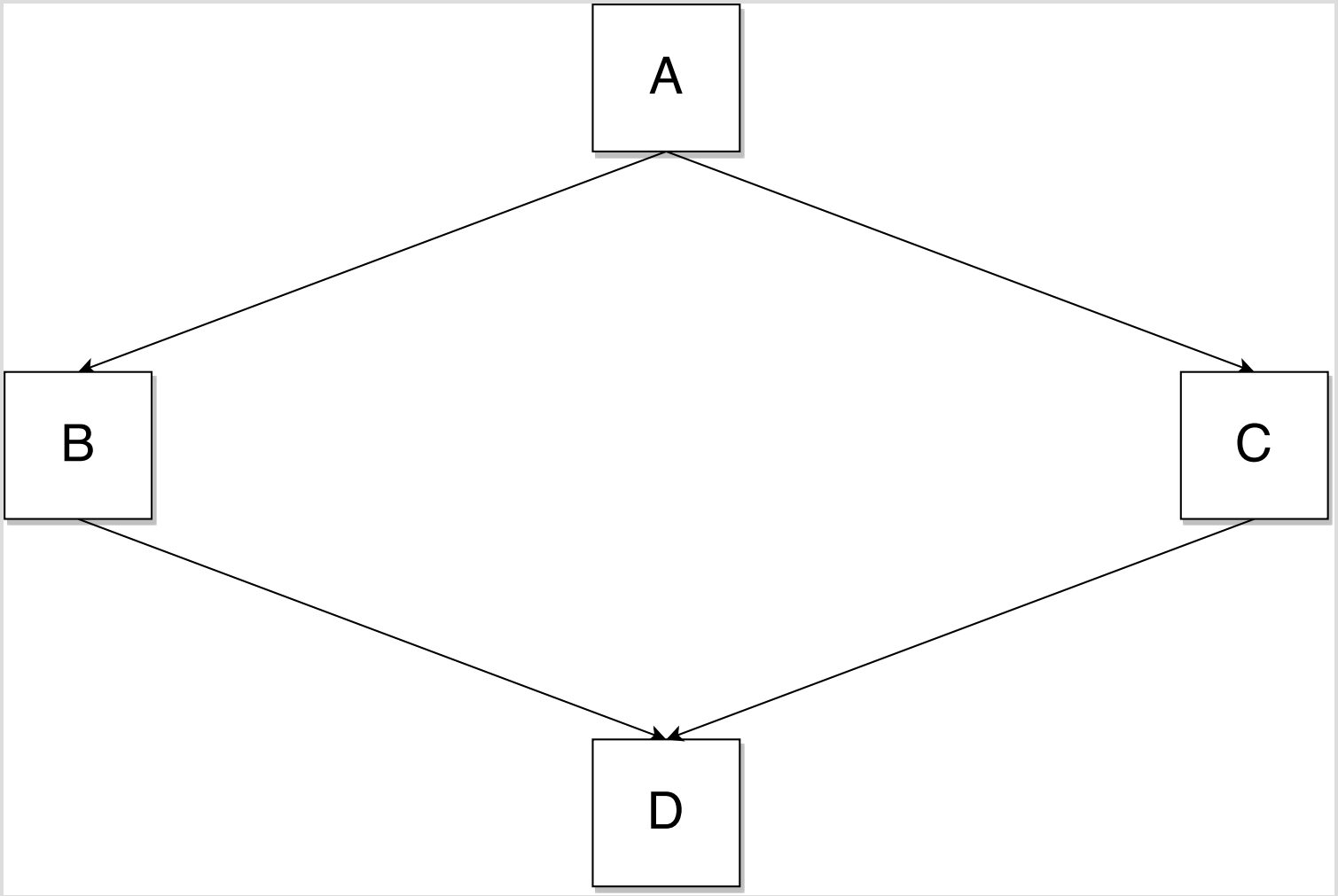
Approach

Basic Concept

- Adding a probabilistic control flow
  $\Rightarrow$ Same input values have different control flows

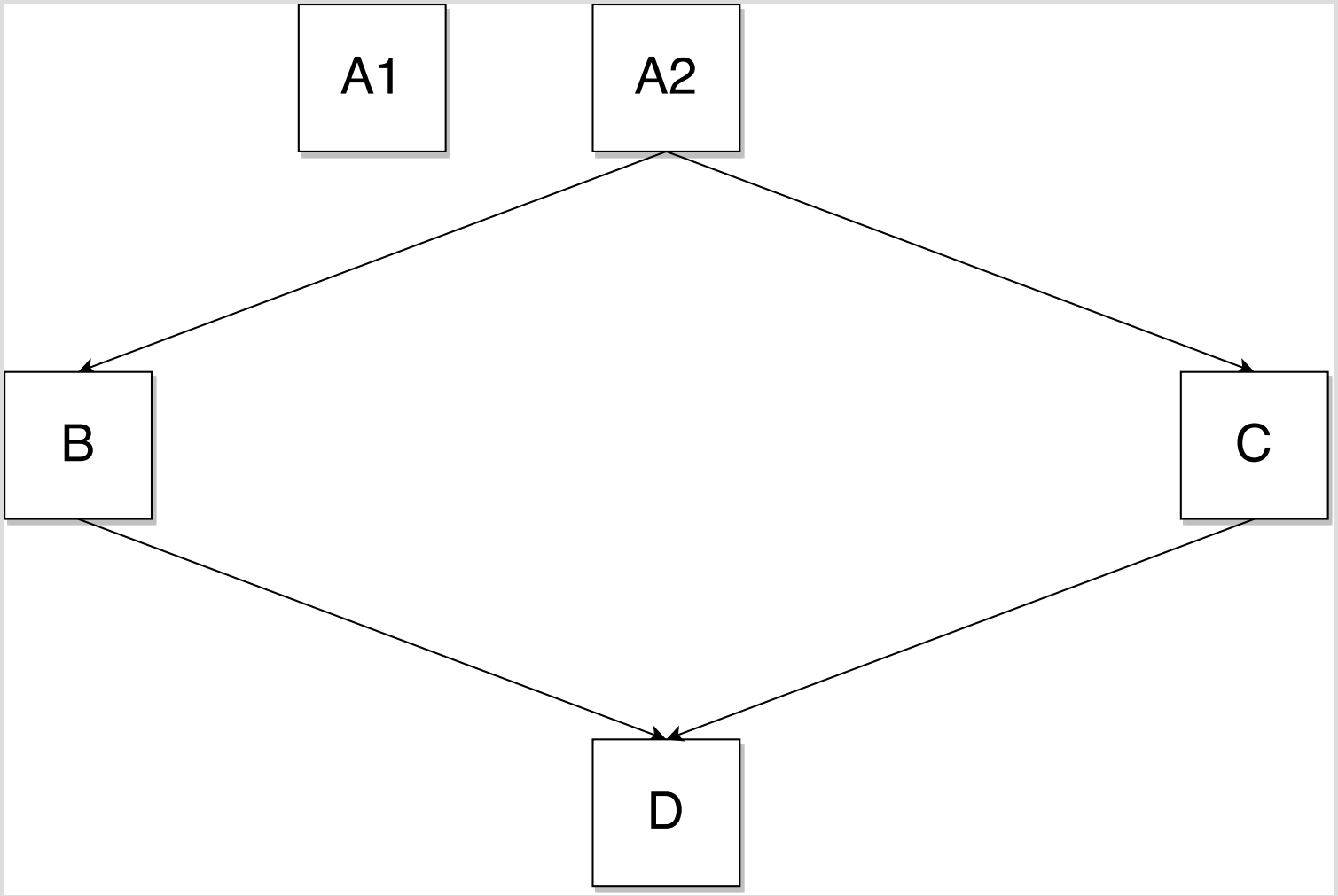
Approach

Probabilistic Control Flow



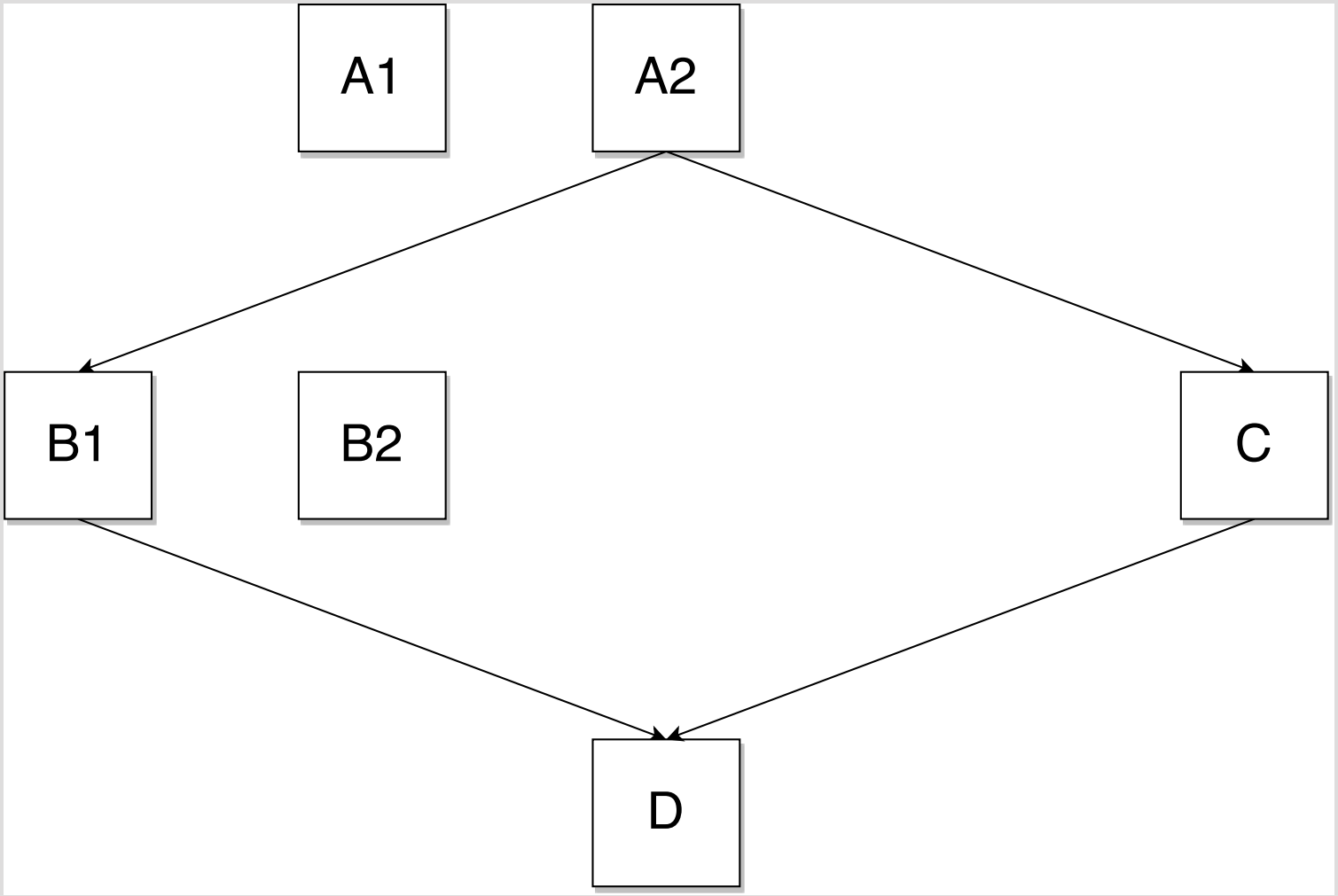
Approach

Probabilistic Control Flow



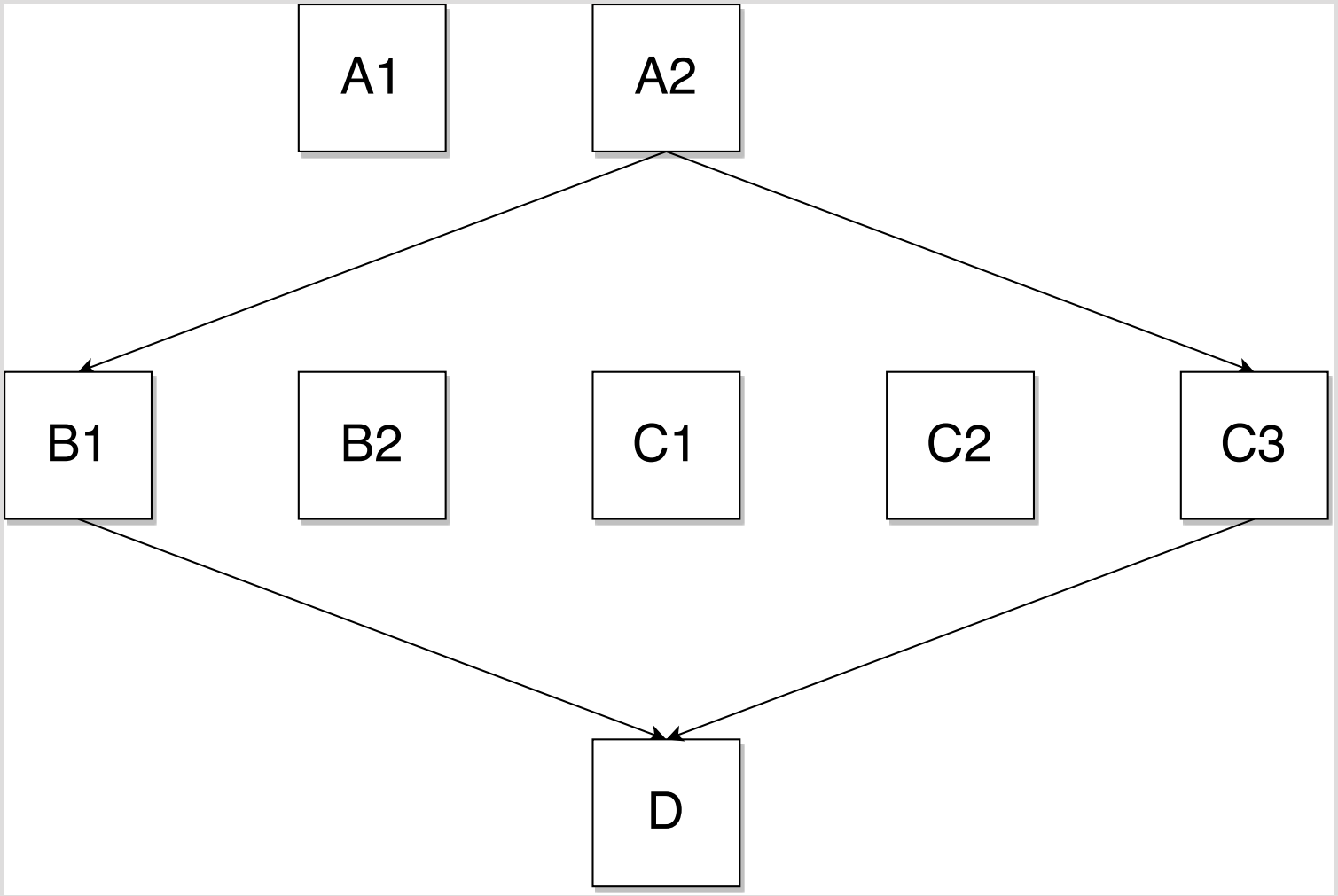
Approach

Probabilistic Control Flow



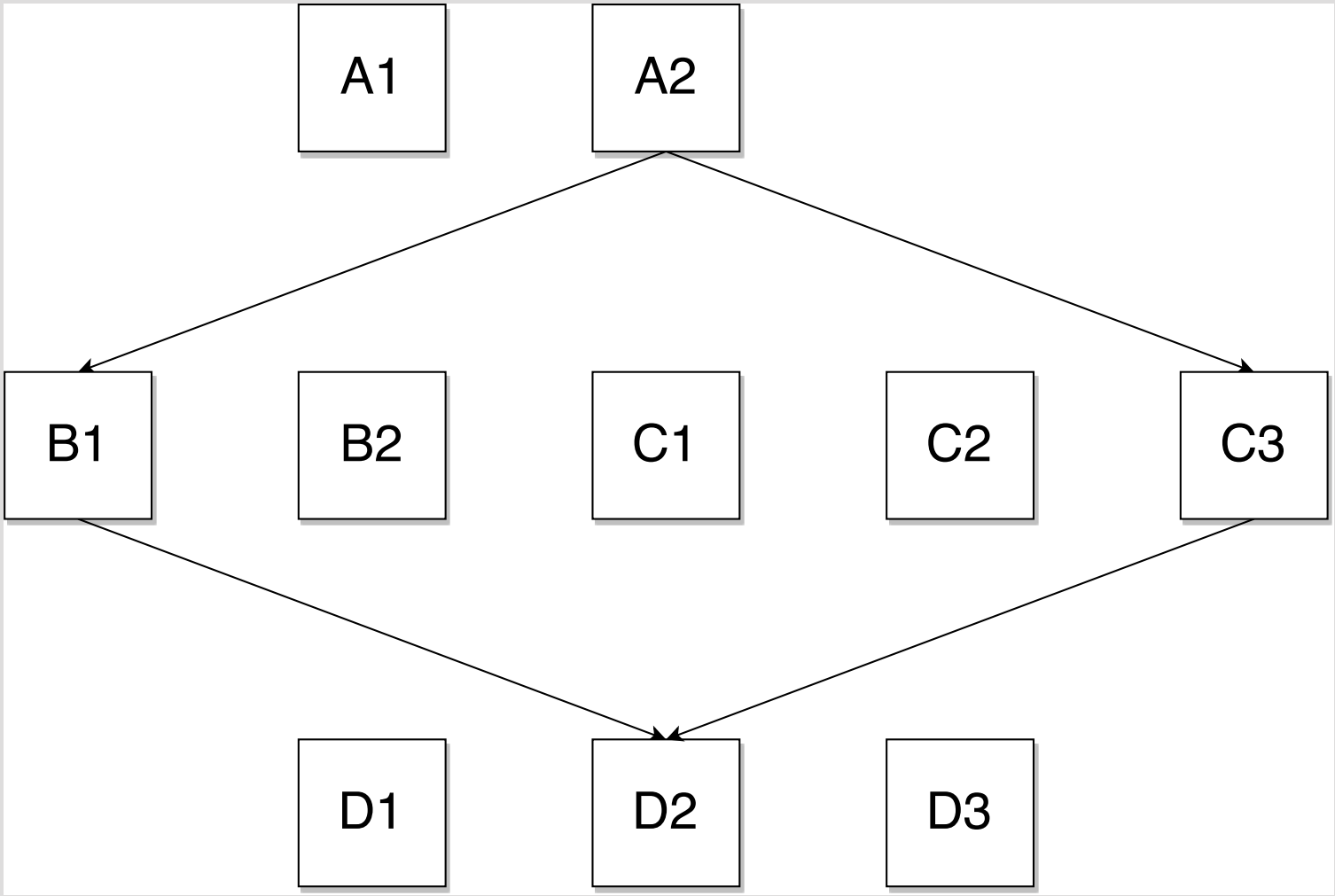
Approach

Probabilistic Control Flow



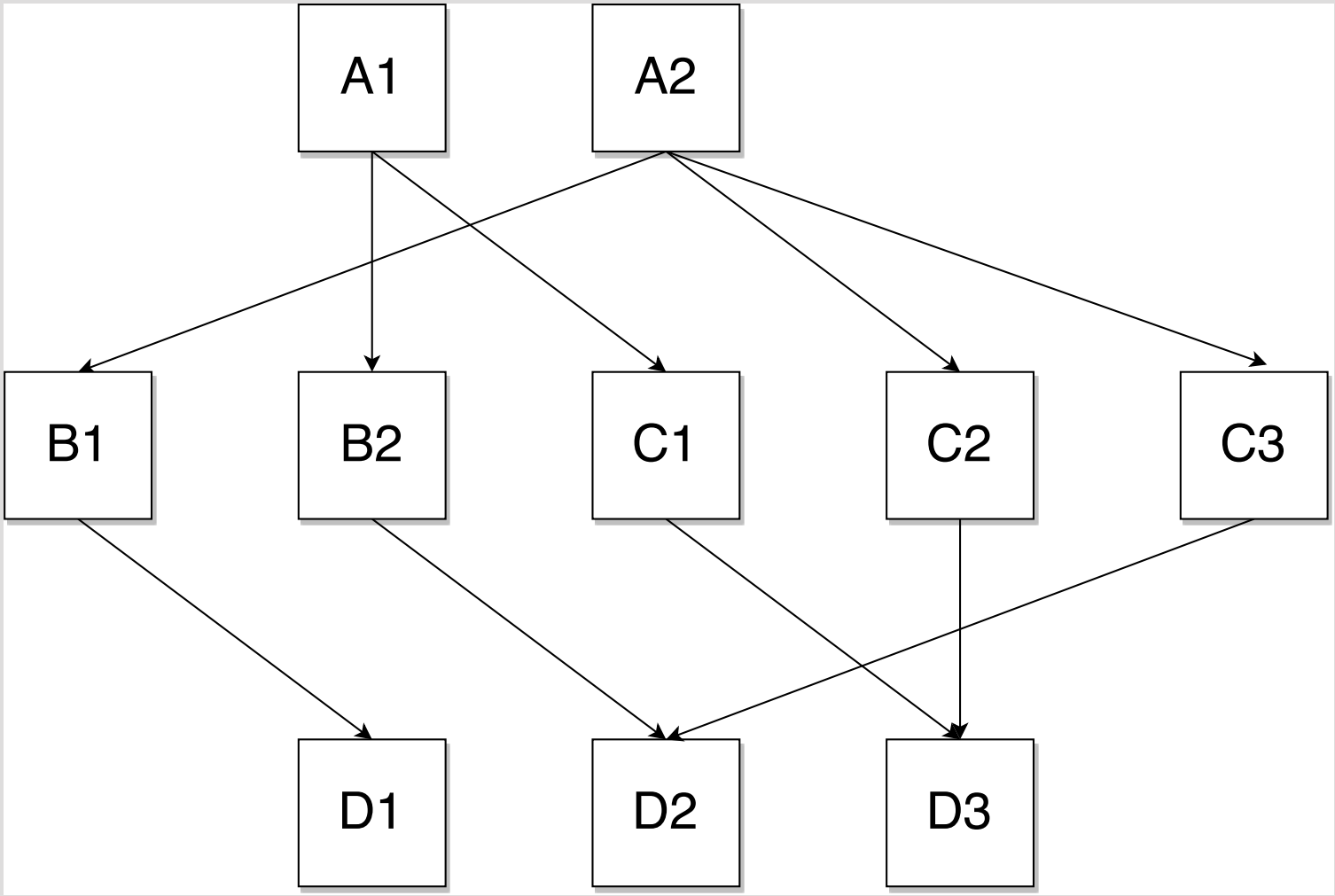
Approach

Probabilistic Control Flow



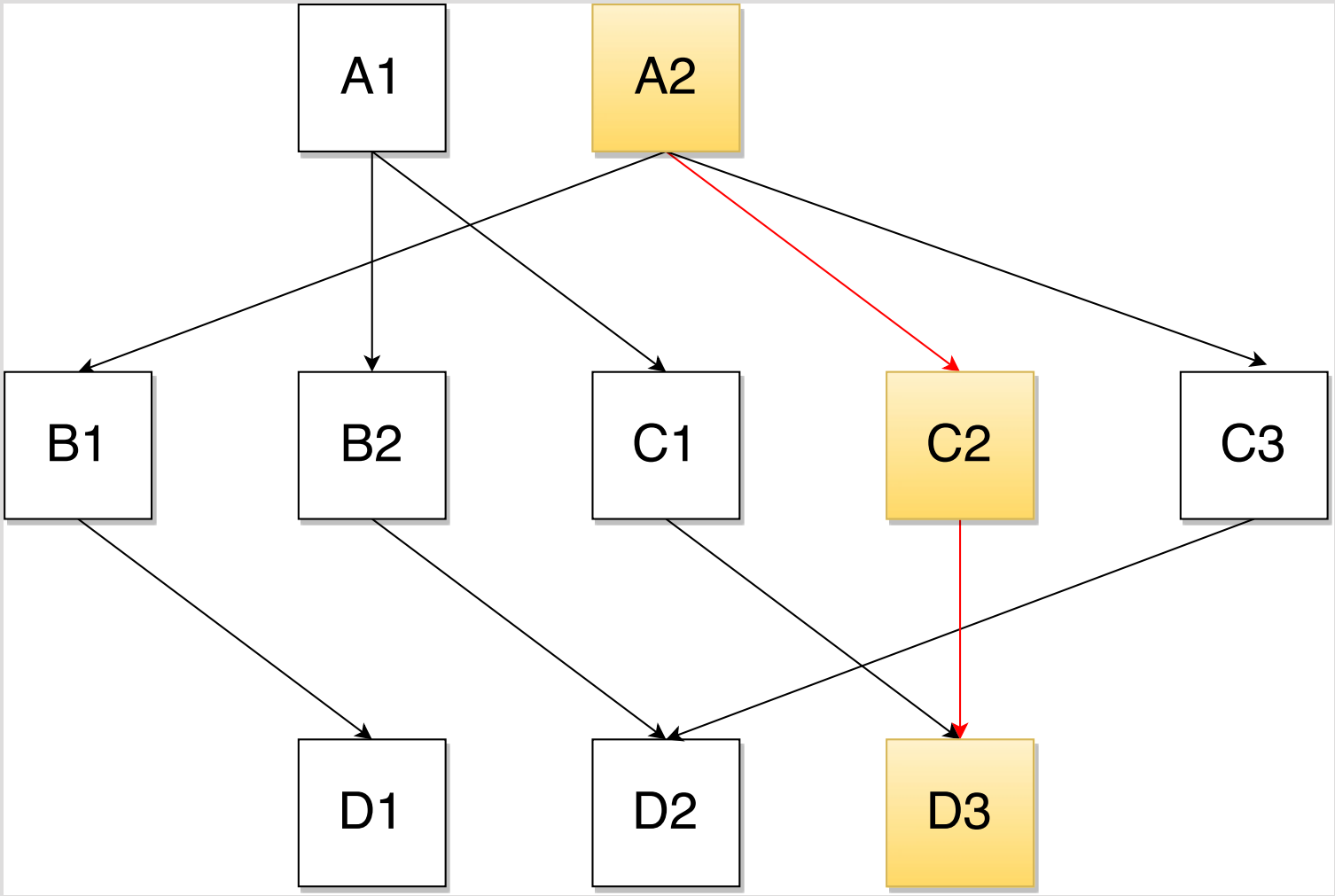
Approach

Probabilistic Control Flow



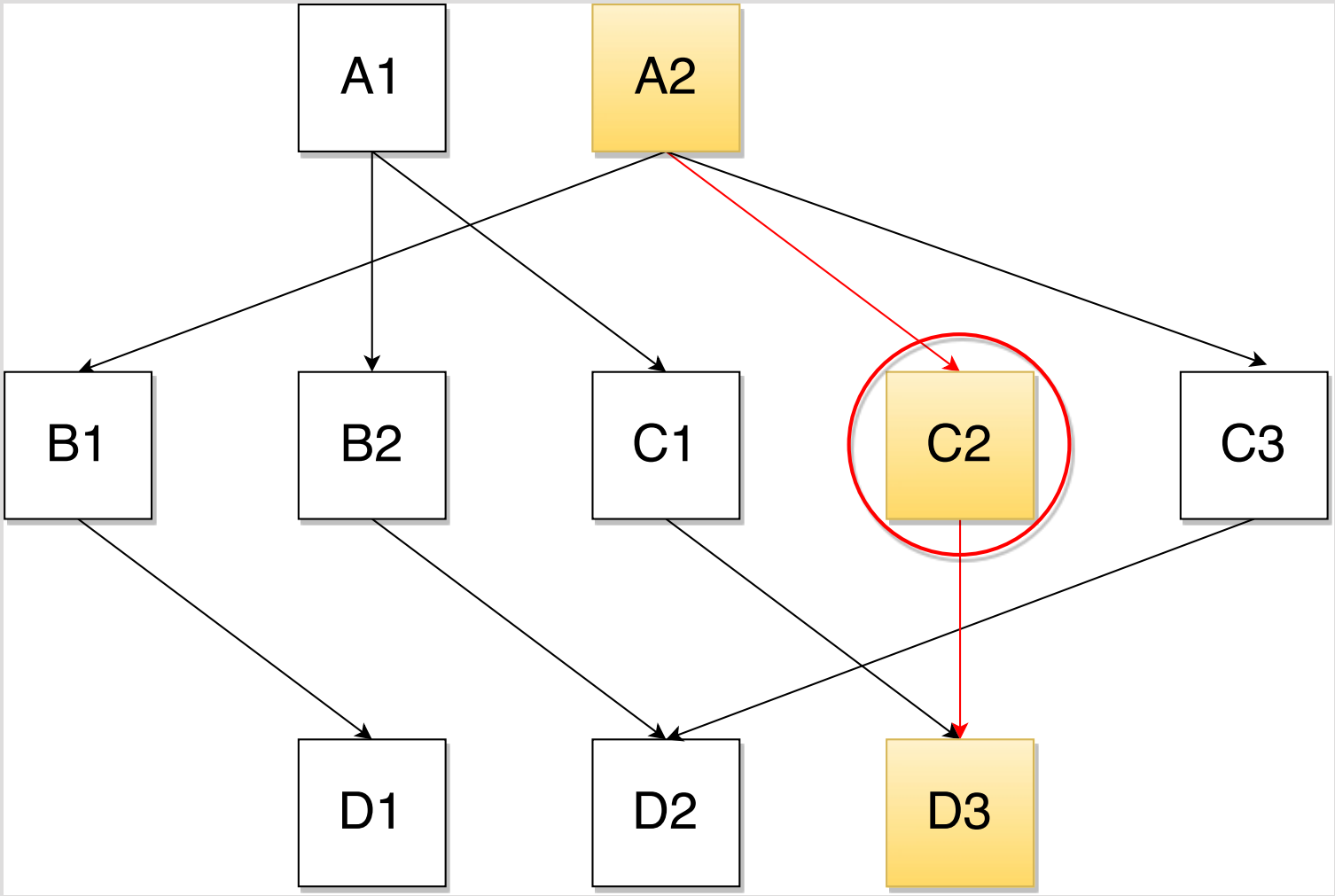
Approach

Probabilistic Control Flow



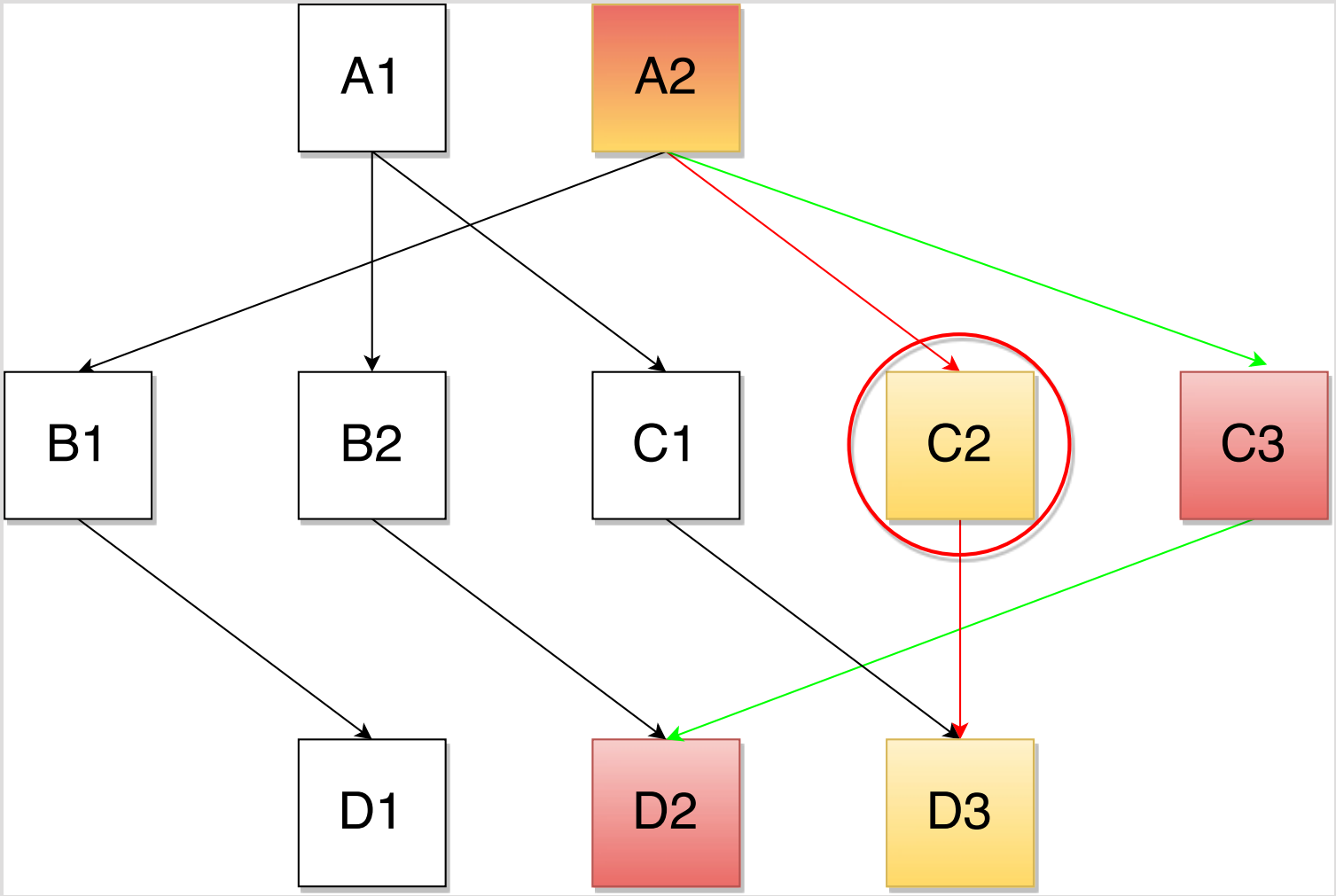
Approach

Probabilistic Control Flow



Approach

Probabilistic Control Flow



Approach

Probabilistic Control Flow

- Adding additional code that controls the control flow with the help of random values
- Adding an artificial graph structure into program
- Using pointer to graph structure to navigate control flow (aliasing)

$\Rightarrow$ Details are in the paper

Approach

Probabilistic Control Flow

- Adding additional code that controls the control flow with the help of random values
- Adding an artificial graph structure into program
- Using pointer to graph structure to navigate control flow (aliasing)

$\Rightarrow$ Details are in the paper

Approach

Probabilistic Control Flow

- Adding additional code that controls the control flow with the help of random values
- Adding an artificial graph structure into program
- Using pointer to graph structure to navigate control flow (aliasing)

$\Rightarrow$ Details are in the paper

Approach

Probabilistic Control Flow

- Adding additional code that controls the control flow with the help of random values
- Adding an artificial graph structure into program
- Using pointer to graph structure to navigate control flow (aliasing)

$\Rightarrow$ Details are in the paper

Overview

Evaluation

Metrics

- Aspects proposed by Collberg et al.
  1. Cost
  2. Resilience
  3. Potency
  4. Stealth

Evaluation

Cost

Cost

Overhead of time and space

Evaluation

Cost

Program	Size (kB)		Avg. Computation (ms)		Memory (kB)	
	Orig.	Obfu.	Orig.	Obfu.	Orig.	Obfu.
SHA-256	12.3	7,666.7	785	5658	1,480	28,852
MD5	13.8	7,068.2	302	491	1,480	28,880
RC4	9.2	7,058.4	1209	1842	1,488	28,828

Evaluation

Cost

Program	Size (kB)		Avg. Computation (ms)		Memory (kB)	
	Orig.	Obfu.	Orig.	Obfu.	Orig.	Obfu.
SHA-256	12.3	7,666.7	785	5658	1,480	28,852
MD5	13.8	7,068.2	302	491	1,480	28,880
RC4	9.2	7,058.4	1209	1842	1,488	28,828

Evaluation

Cost

Program	Size (kB)		Avg. Computation (ms)		Memory (kB)	
	Orig.	Obfu.	Orig.	Obfu.	Orig.	Obfu.
SHA-256	12.3	7,666.7	785	5658	1,480	28,852
MD5	13.8	7,068.2	302	491	1,480	28,880
RC4	9.2	7,058.4	1209	1842	1,488	28,828

Evaluation

Cost

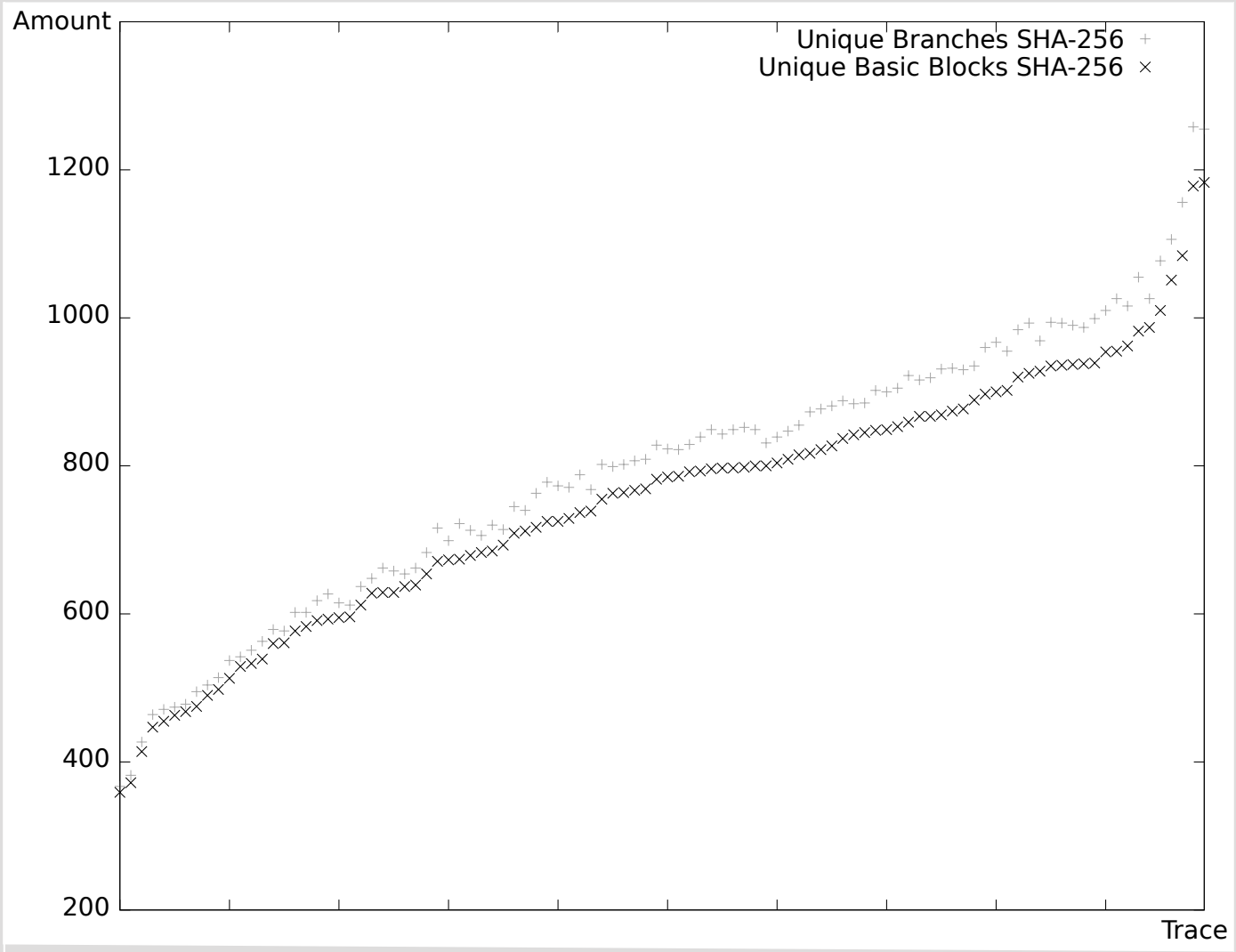
Program	Size (kB)		Avg. Computation (ms)		Memory (kB)	
	Orig.	Obfu.	Orig.	Obfu.	Orig.	Obfu.
SHA-256	12.3	7,666.7	785	5658	1,480	28,852
MD5	13.8	7,068.2	302	491	1,480	28,880
RC4	9.2	7,058.4	1209	1842	1,488	28,828

Resilience

Resistance against deobfuscation attempts

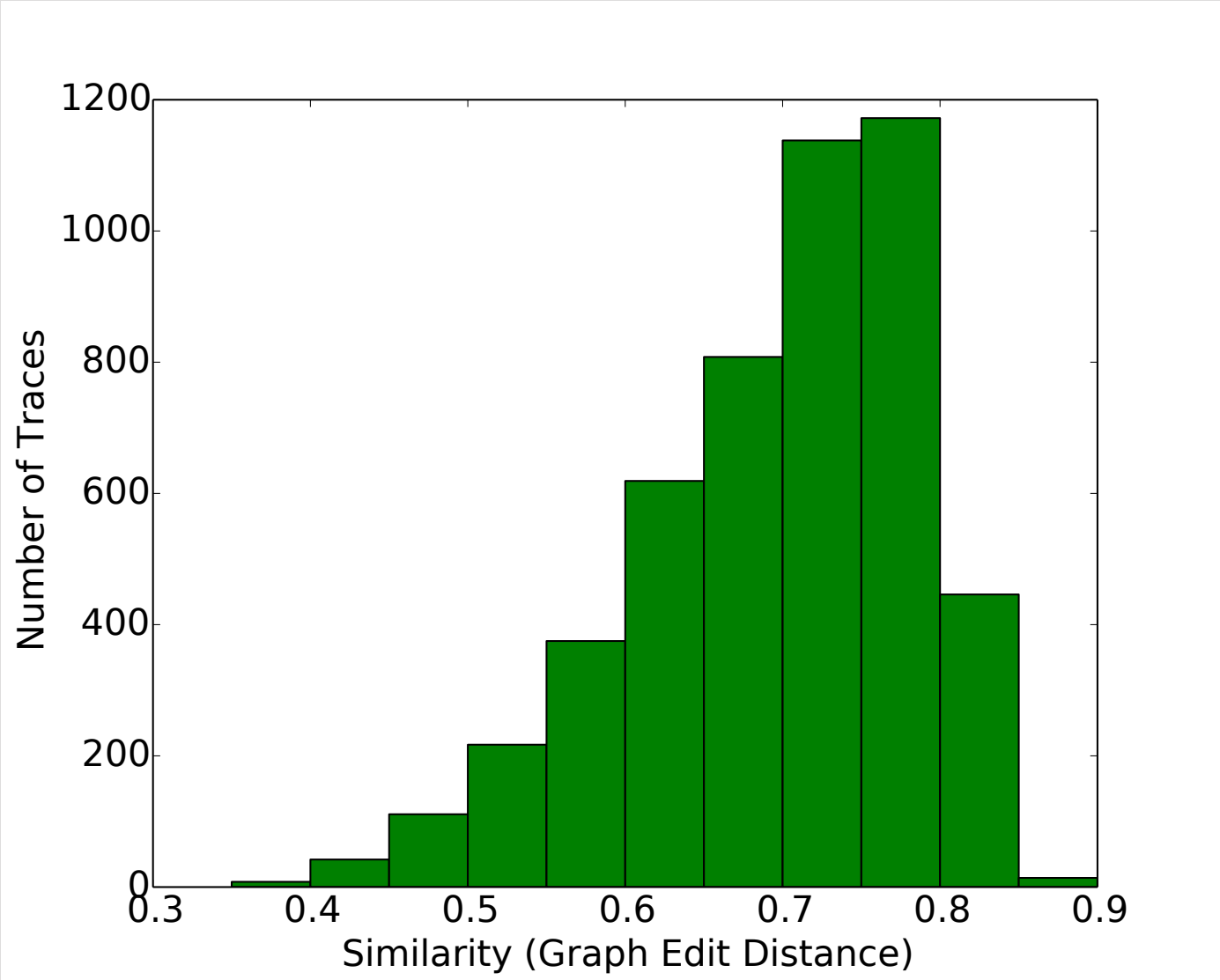
Evaluation

Resilience



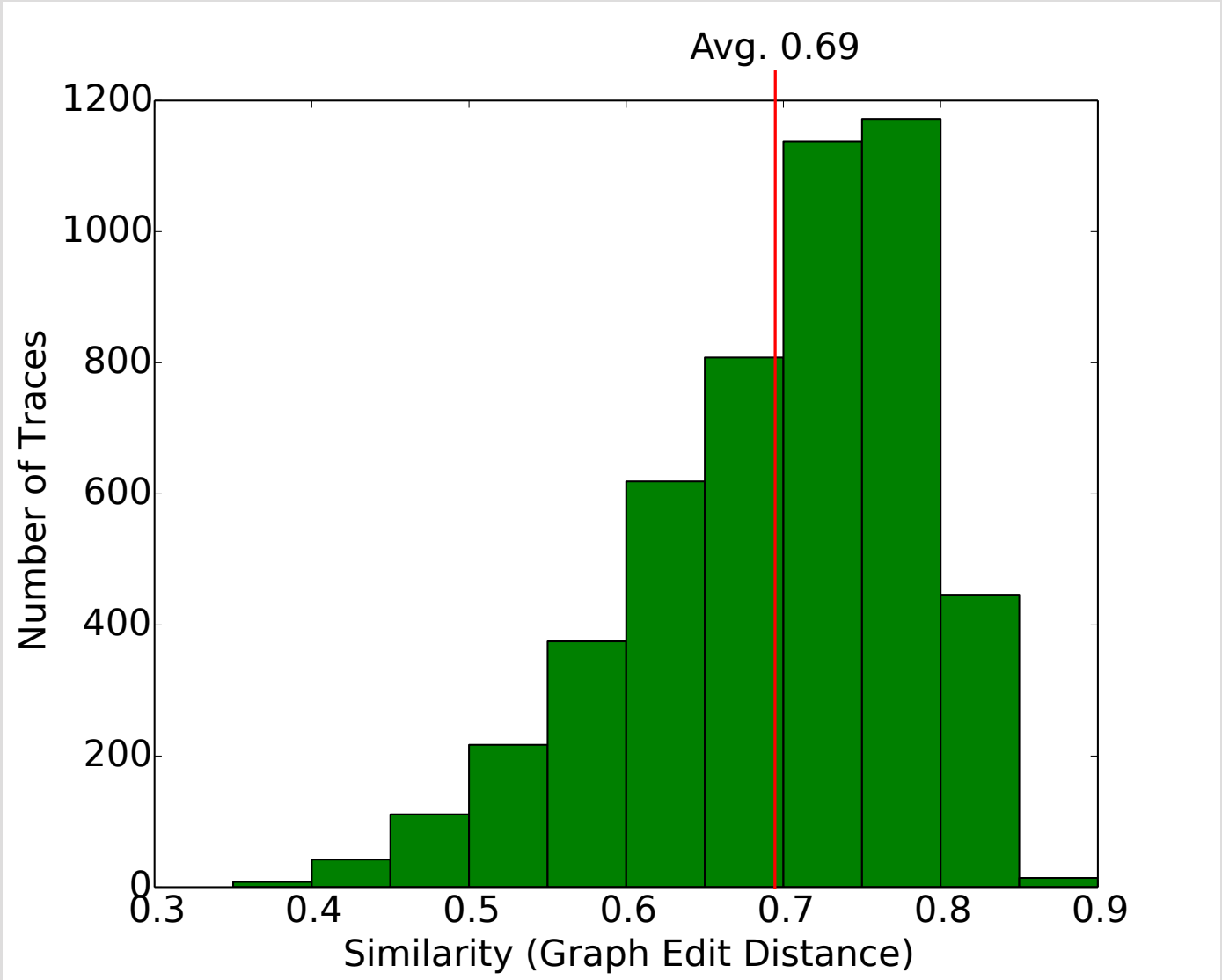
Evaluation

Resilience



Evaluation

Resilience



Potency

Complexity after obfuscation process

Evaluation

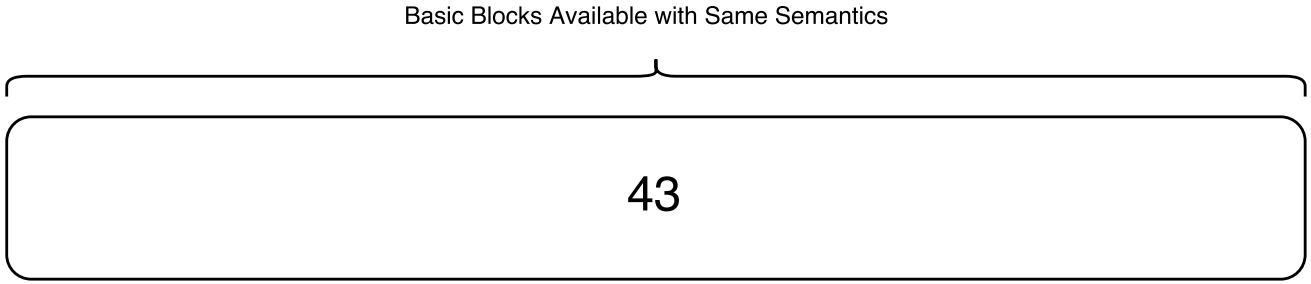
Potency

ID	0	1	...	14	15	16	Total
|avail|	9	43	...	43	23	33	469
|exec|	1	20	...	130	2	128	572
|unique|	1	10	...	24	2	20	108
`Util`	100	50	...	55.8	100	60.6	71

Evaluation

Potency

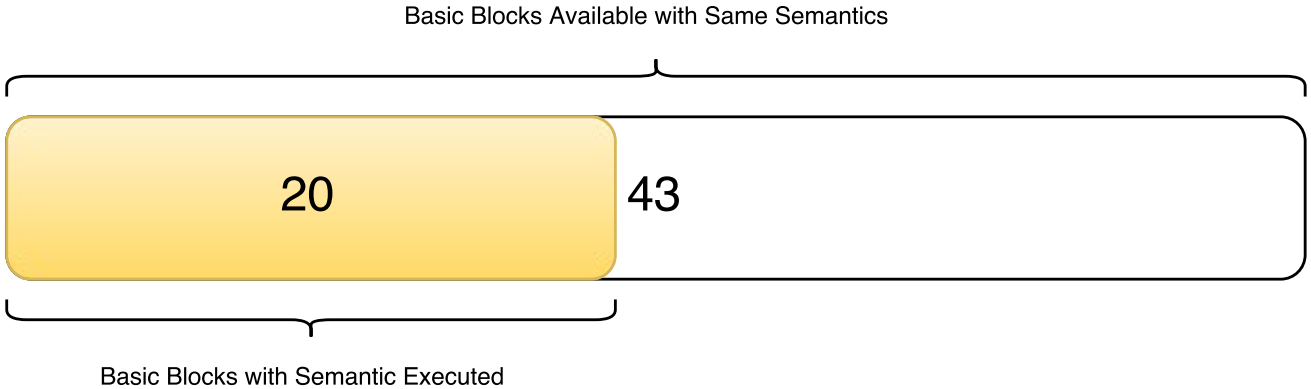
ID	0	1	...	14	15	16	Total
|avail|	9	43	...	43	23	33	469
|exec|	1	20	...	130	2	128	572
|unique|	1	10	...	24	2	20	108
`Util`	100	50	...	55.8	100	60.6	71



Evaluation

Potency

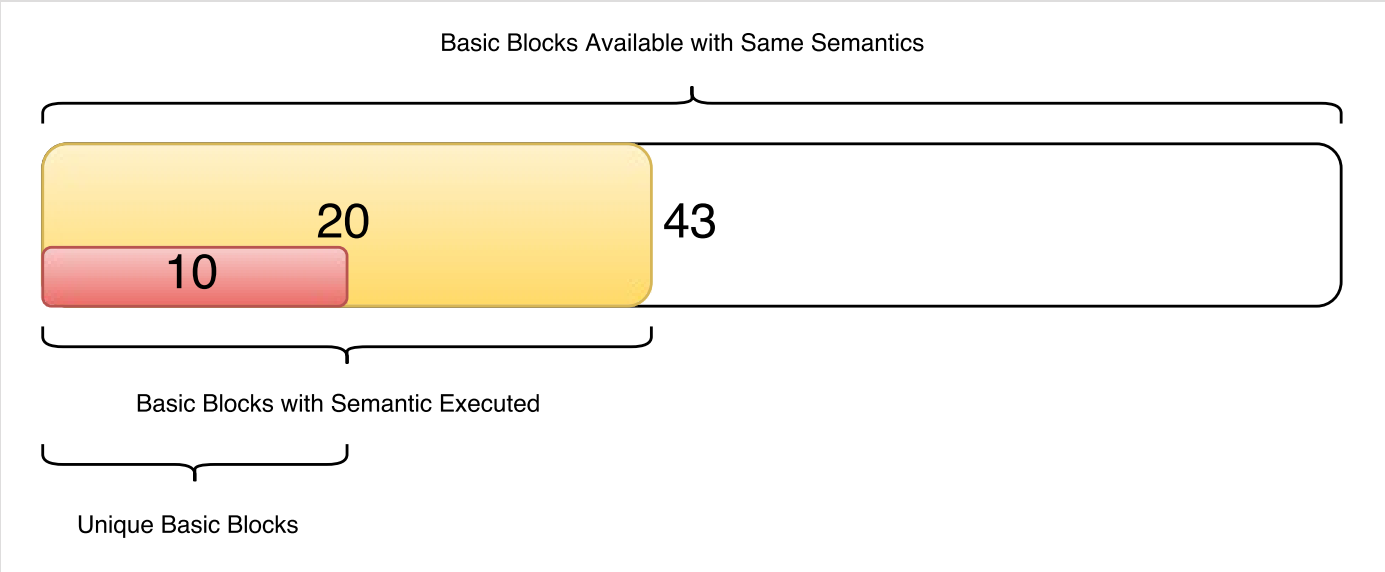
ID	0	1	...	14	15	16	Total
|avail|	9	43	...	43	23	33	469
|exec|	1	20	...	130	2	128	572
|unique|	1	10	...	24	2	20	108
Util	100	50	...	55.8	100	60.6	71



Evaluation

Potency

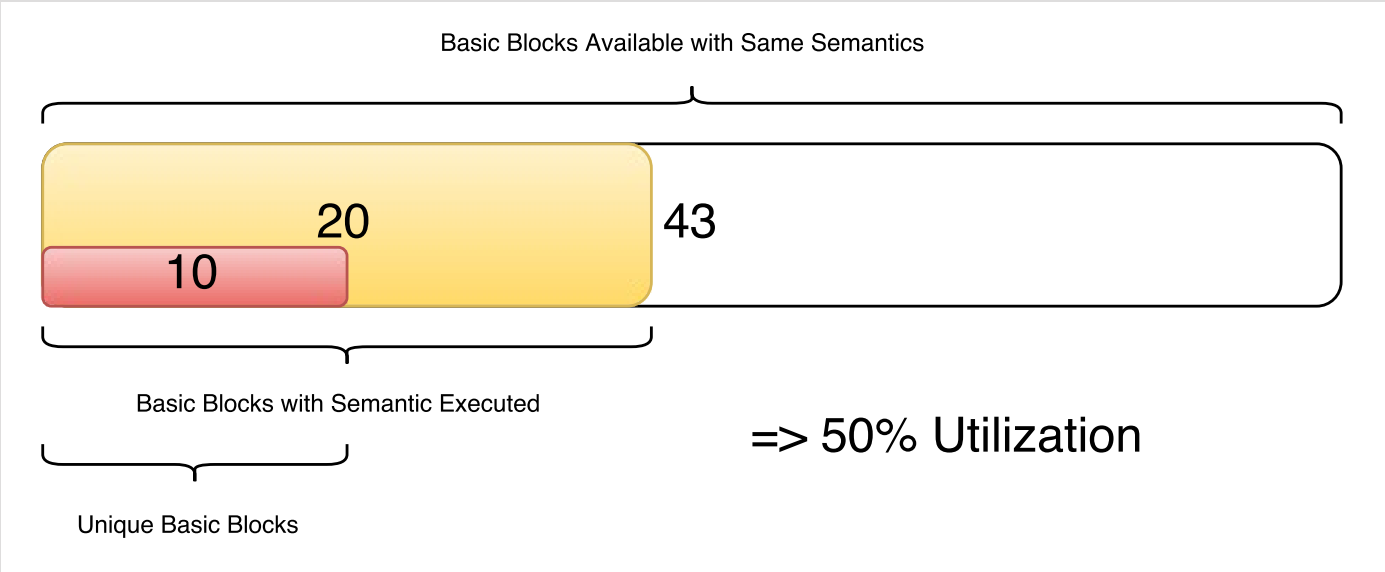
ID	0	1	...	14	15	16	Total
|avail|	9	43	...	43	23	33	469
|exec|	1	20	...	130	2	128	572
|unique|	1	10	...	24	2	20	108
Util	100	50	...	55.8	100	60.6	71



Evaluation

Potency

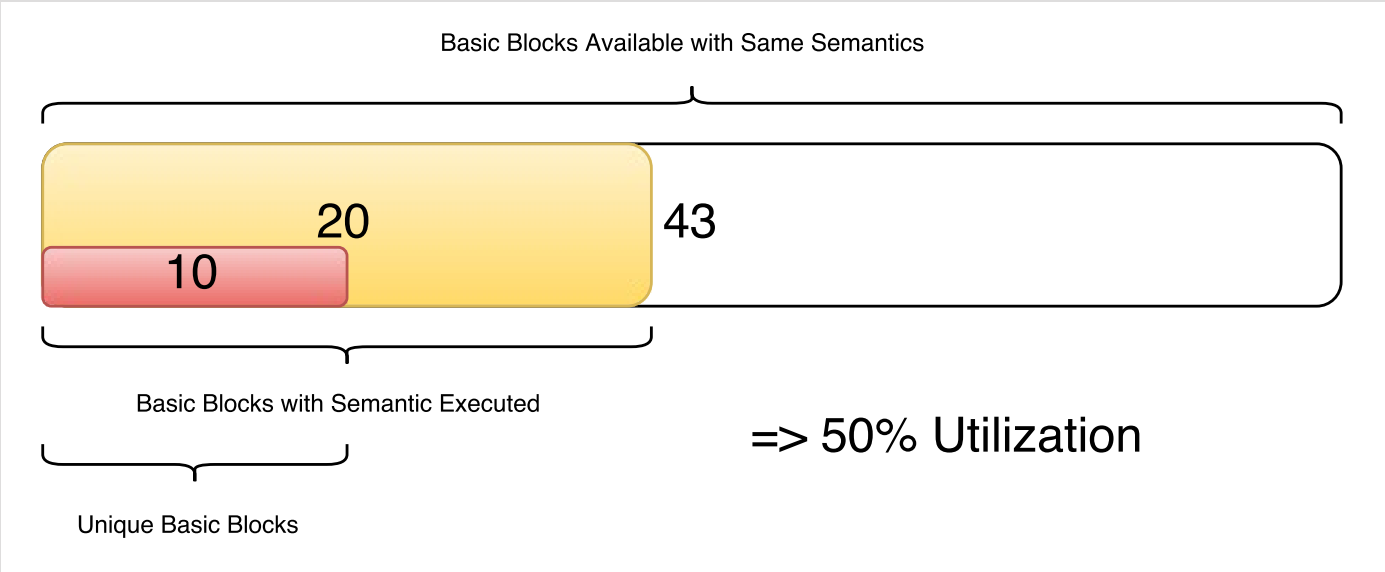
ID	0	1	...	14	15	16	Total
|avail|	9	43	...	43	23	33	469
|exec|	1	20	...	130	2	128	572
|unique|	1	10	...	24	2	20	108
Util	100	50	...	55.8	100	60.6	71



Evaluation

Potency

ID	0	1	...	14	15	16	Total
|avail|	9	43	...	43	23	33	469
|exec|	1	20	...	130	2	128	572
|unique|	1	10	...	24	2	20	108
Util	100	50	...	55.8	100	60.6	71



Evaluation

Stealth

Stealth

Obfuscation blends into original program

Evaluation

Stealth

- Not an objective of our approach
- Evaluated dynamic behavior of obfuscated program for completeness
- Control flow changes with each execution
  $\Rightarrow$ Easy to detect by an adversary

Overview

Conclusion

Discussion & Limitations

- Analysis working on a single execution trace
  - Still gets harder if a loop is involved
- Random numbers for probabilistic control flow
  - Values used to seed random number generator
  - Control flow deterministic when seed fixed

Conclusion

Discussion & Limitations

- Analysis working on a single execution trace
  - Still gets harder if a loop is involved
- Random numbers for probabilistic control flow
  - Values used to seed random number generator
  - Control flow deterministic when seed fixed

Conclusion

Results

- Proof-of-concept has significant performance and memory penalty
  - One has to weigh up constraints on time and space
- Obfuscated methods do not exhibit the same execution trace
  - Thwarting dynamic analysis
- Implementation of prototype obfuscator for .NET applications
  - https://github.com/RUB-SysSec/Probfuscator

Conclusion

Results

- Proof-of-concept has significant performance and memory penalty
  - One has to weigh up constraints on time and space
- Obfuscated methods do not exhibit the same execution trace
  - Thwarting dynamic analysis
- Implementation of prototype obfuscator for .NET applications
  - https://github.com/RUB-SysSec/Probfuscator

Conclusion

Results

- Proof-of-concept has significant performance and memory penalty
  - One has to weigh up constraints on time and space
- Obfuscated methods do not exhibit the same execution trace
  - Thwarting dynamic analysis
- Implementation of prototype obfuscator for .NET applications
  - https://github.com/RUB-SysSec/Probfuscator

Conclusion

Thank you for your attention.
Any Questions?

Conclusion

Results

- Proof-of-concept has significant performance and memory penalty
  - One has to weigh up constraints on time and space
- Obfuscated methods do not exhibit the same execution trace
  - Thwarting dynamic analysis
- Implementation of prototype obfuscator for .NET applications
  - https://github.com/RUB-SysSec/Probfuscator