Engineering Code Obfuscation

EUROCRYPT 2016

Christian Collberg

Department of Computer Science
University of Arizona

http://collberg.cs.arizona.edu

collberg@gmail.com

Supported by NSF grants 1525820 and 1318955 and
by the private foundation that shall not be named

Man-At-The-End Applications

Tools and Counter Tools

Obfuscation vs. Deobfuscation

Deploying Obfuscation

Evaluation

Discussion

Man-At-The-End Applications

Tools and Counter Tools

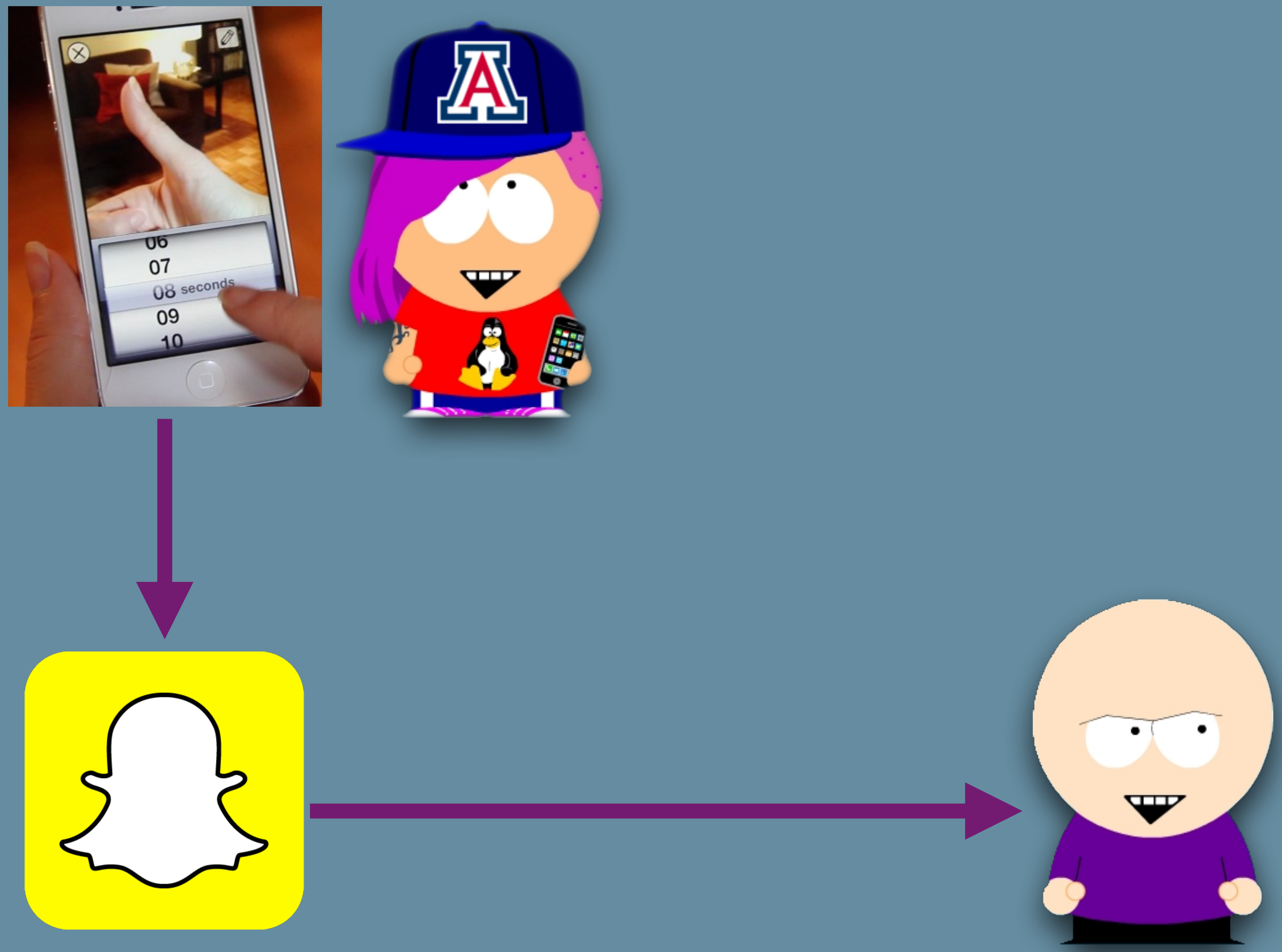
Obfuscation vs. Deobfuscation

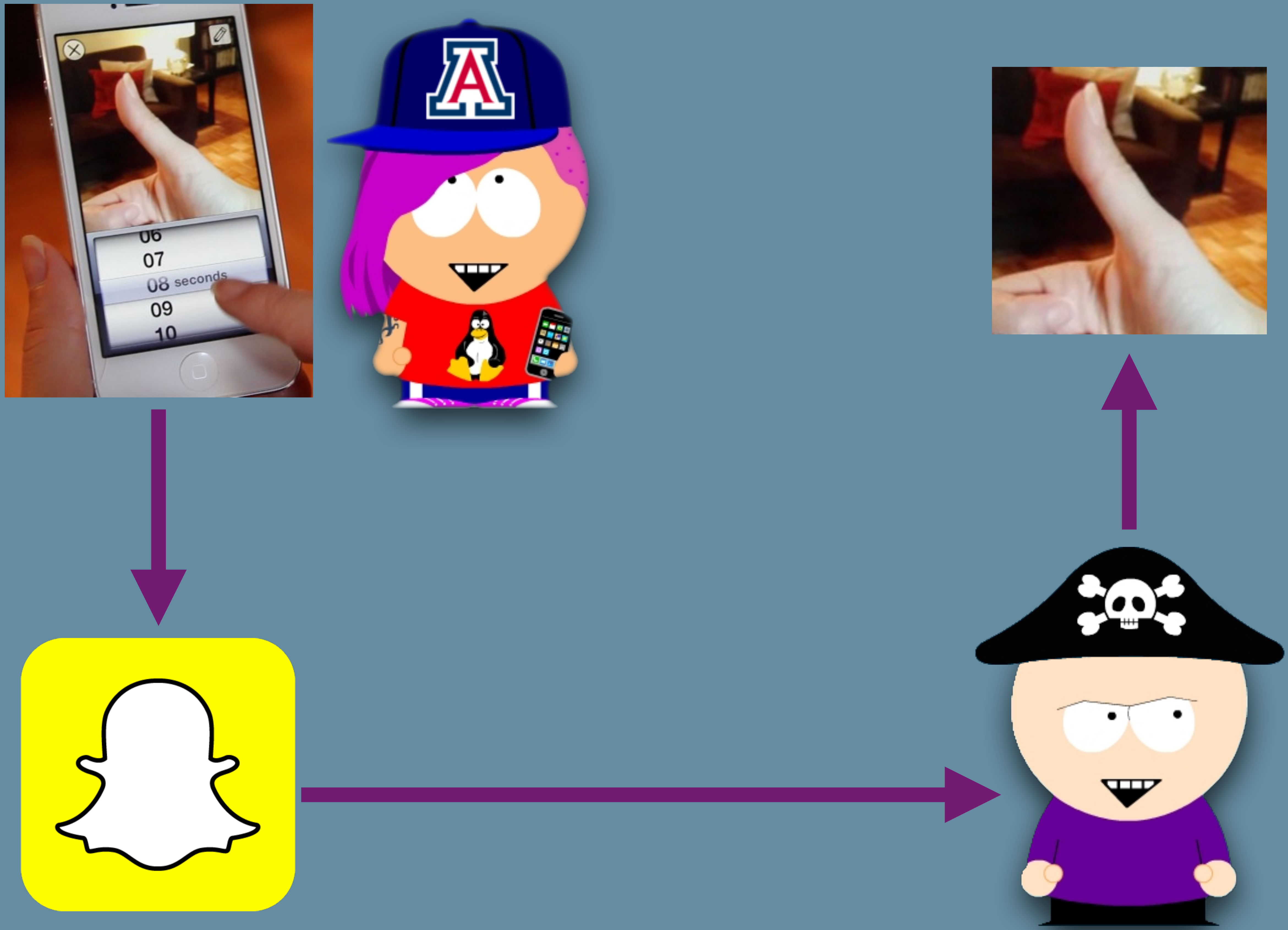
Deploying Obfuscation

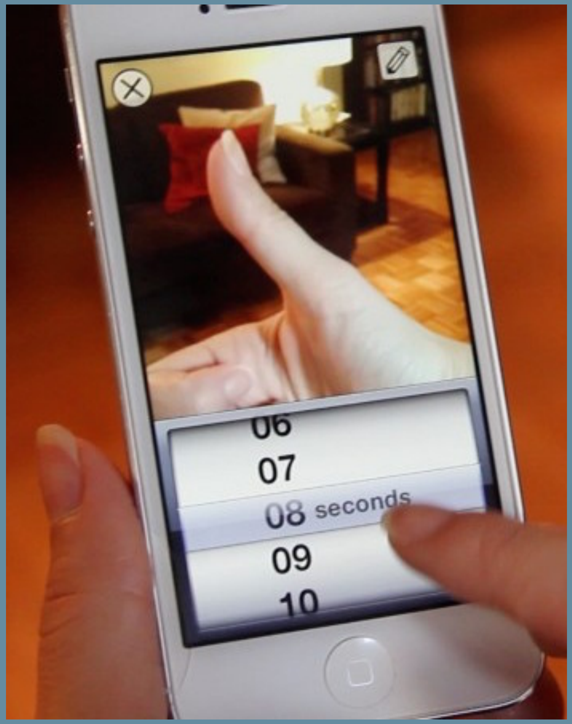
Evaluation

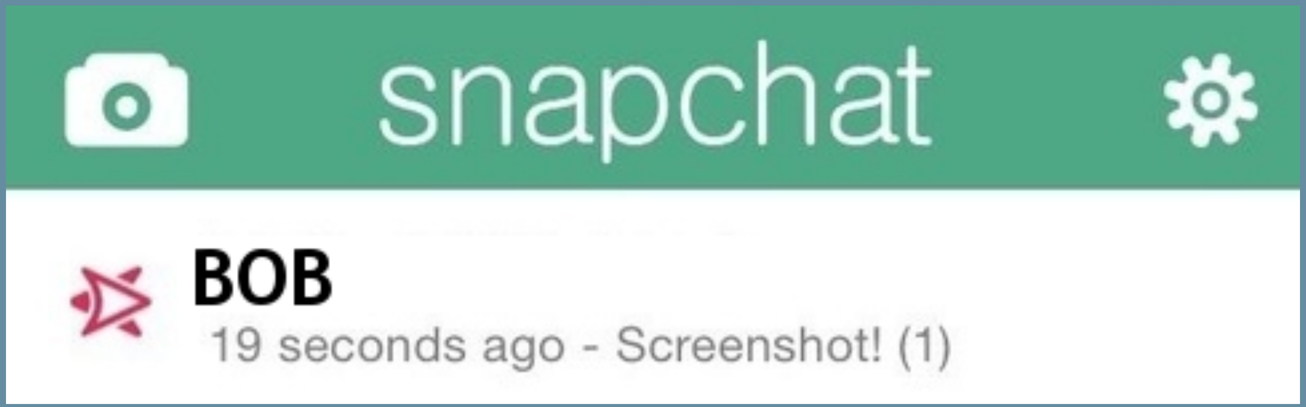
Discussion

Man-at-the-End Scenarios


06
07
08 seconds
09
10


06
07
08 seconds
09
10

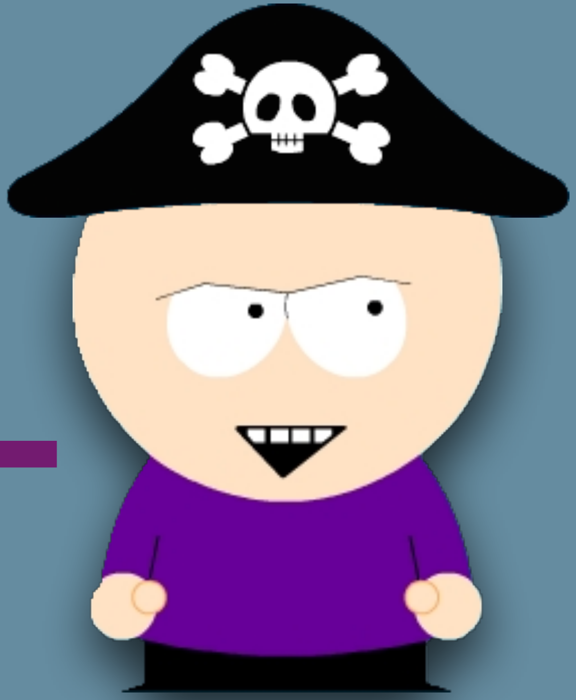


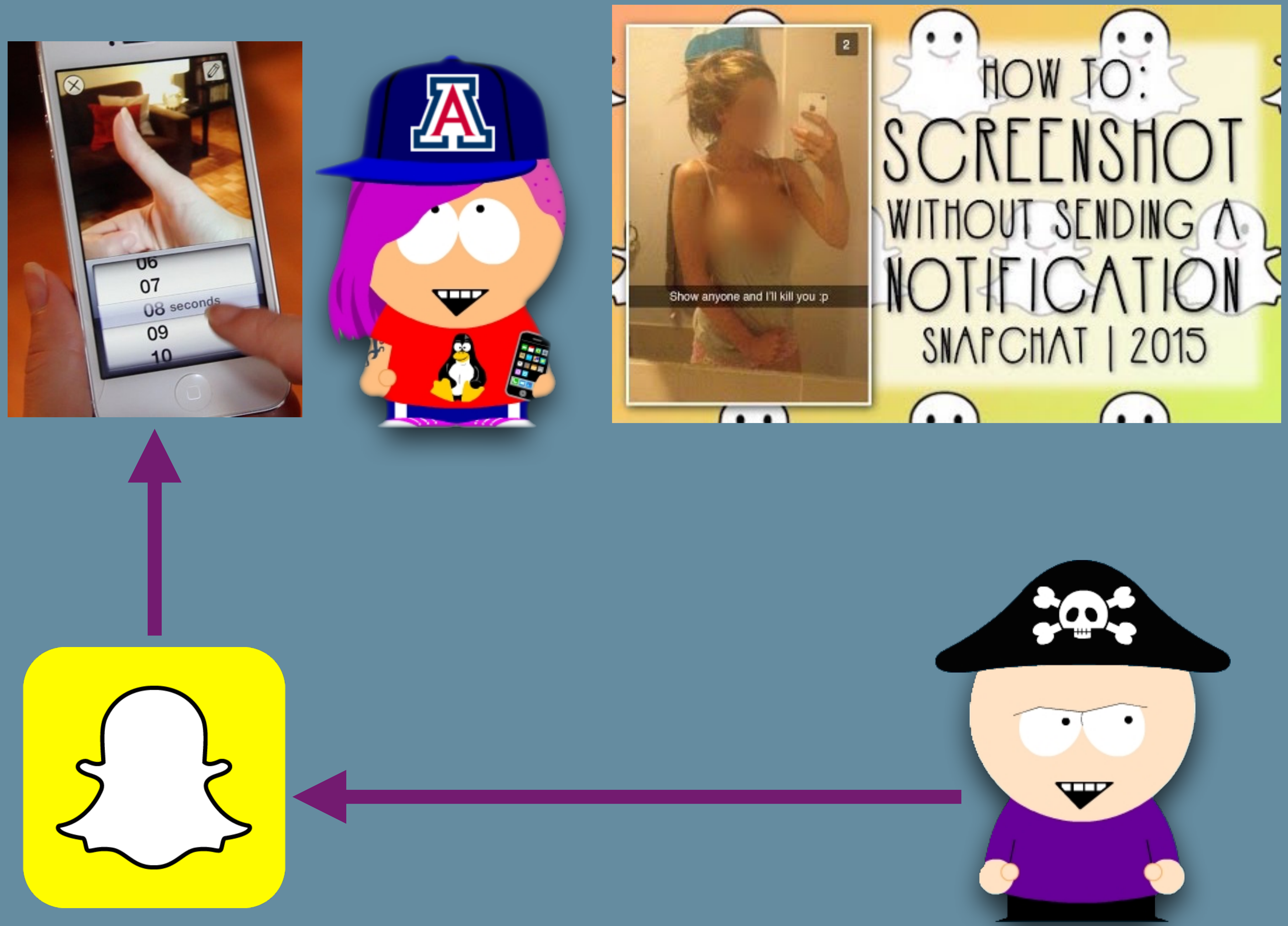


snapchat

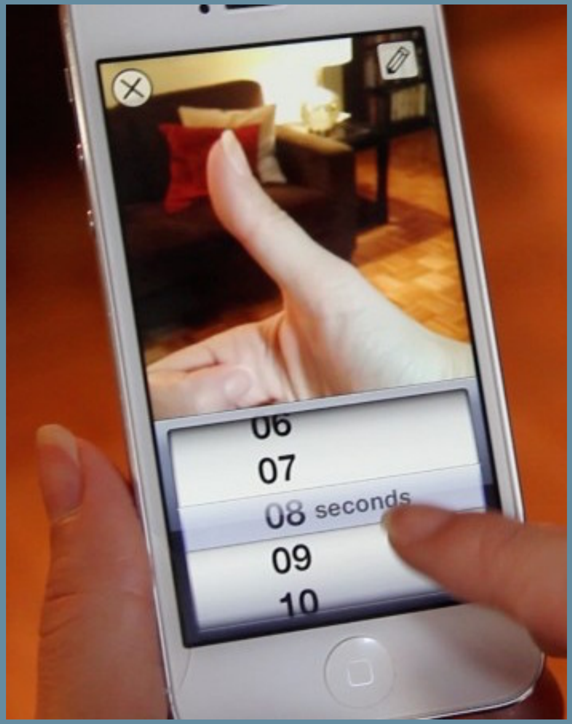
BOB

19 seconds ago - Screenshot! (1)

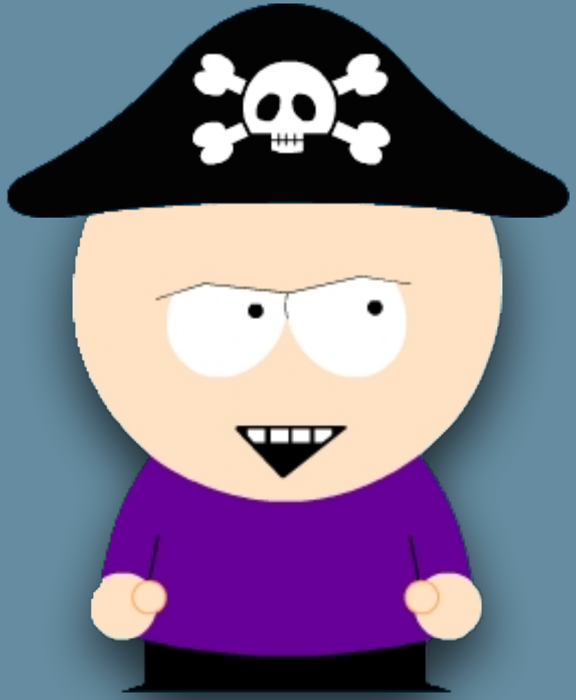



06
07
08 seconds
09
10
2
Show anyone and I'll kill you :p
HOW TO:
SCREENSHOT
WITHOUT SENDING A
NOTIFICATION
SNAPCHAT | 2015

```
snapchat() {
   after (8 seconds)
      remove_picture();
   if (screenshot())
      notify_sender();
   if (app_is_tampered()
      ||
      env_is_suspicious()
      ||
      bob_is_curious())
         punish_bob();
}
```



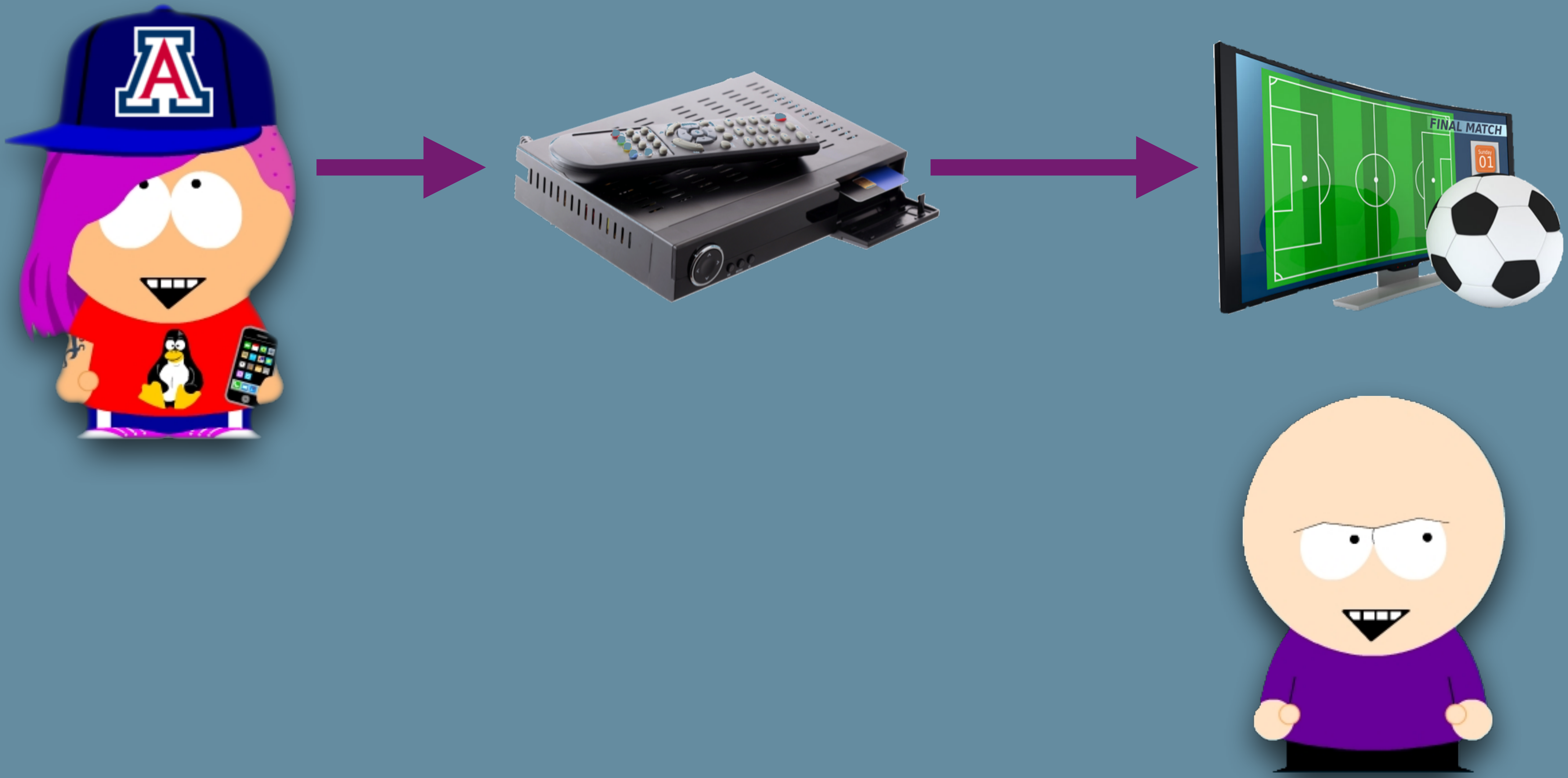
MATE

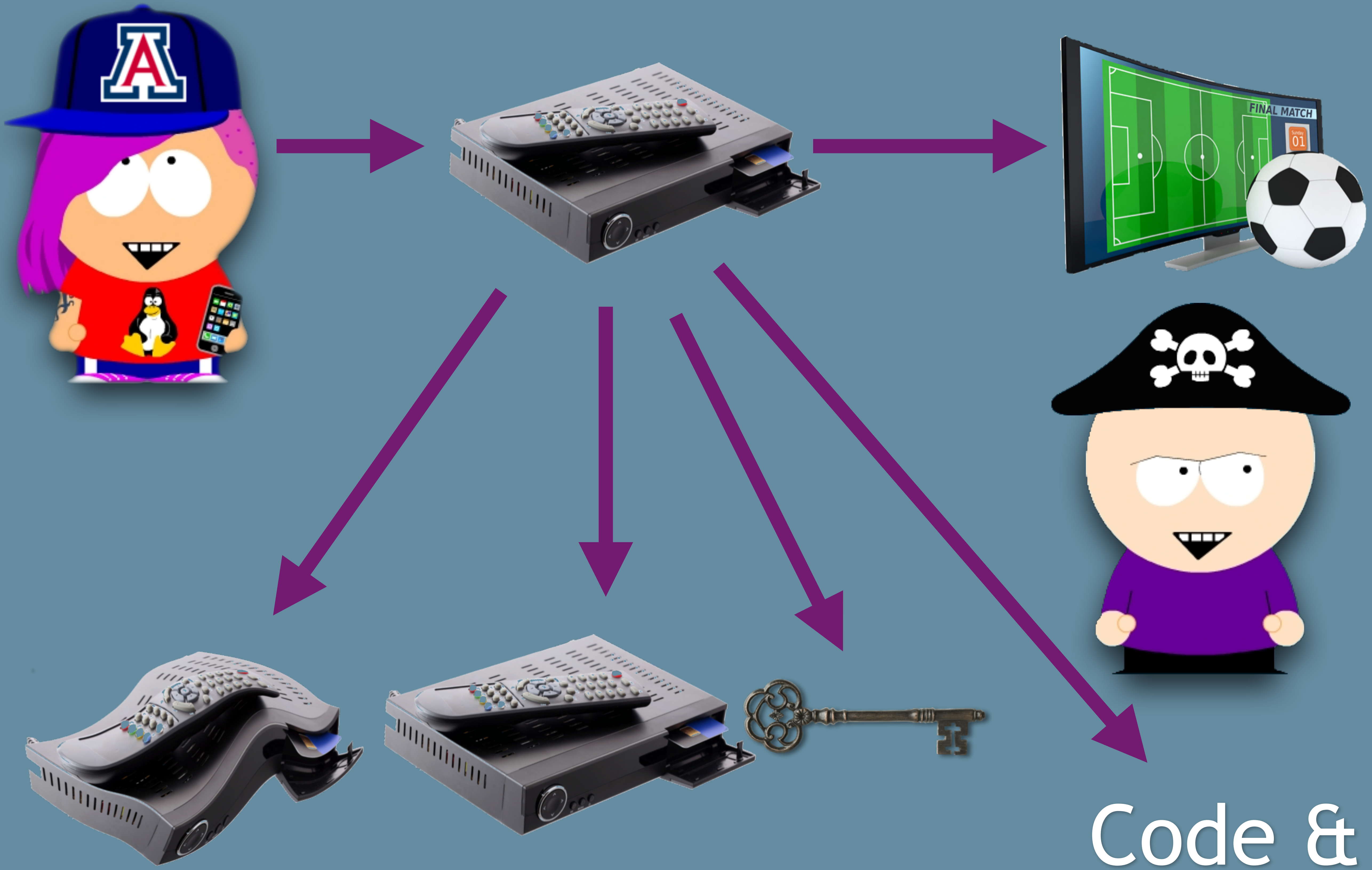


Security and Privacy Scientist

Man-At-The-End

MATE attacks occur in any setting where an adversary has physical access to a device and compromises it by inspecting, reverse engineering, or tampering with its hardware or software.


FINAL MATCH
Sunday
01



Tamper

Clone

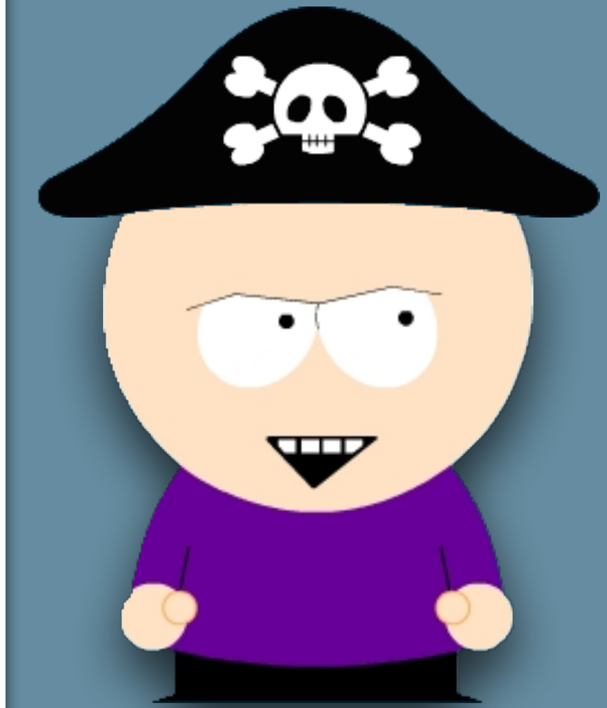
Keys

Code & Content

```
set_top_box() {
   if (bob_paid(“ESPN”))
      allow_access();

   if (hw_is_tampered()
      ||
      sw_is_tampered()
      ||
      bob_is_curious()
      ||…)
          punish_bob();

}
```



00268
Meter quality checked
Old Meter Readings

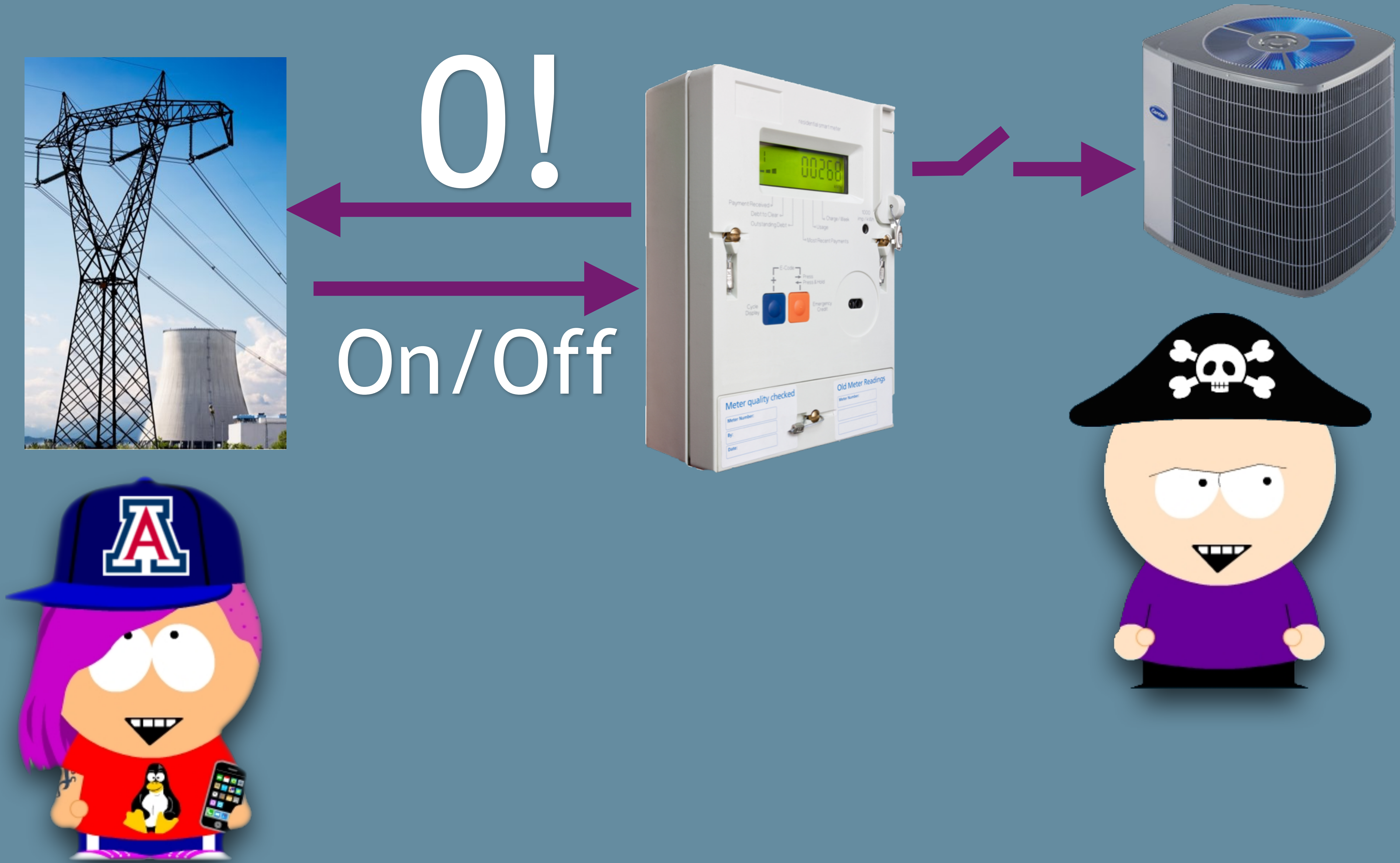
kWh

Cleemput, Mustafa, Preneel, *High Assurance Smart Metering*

00268
Meter quality checked
Old Meter Readings

0!

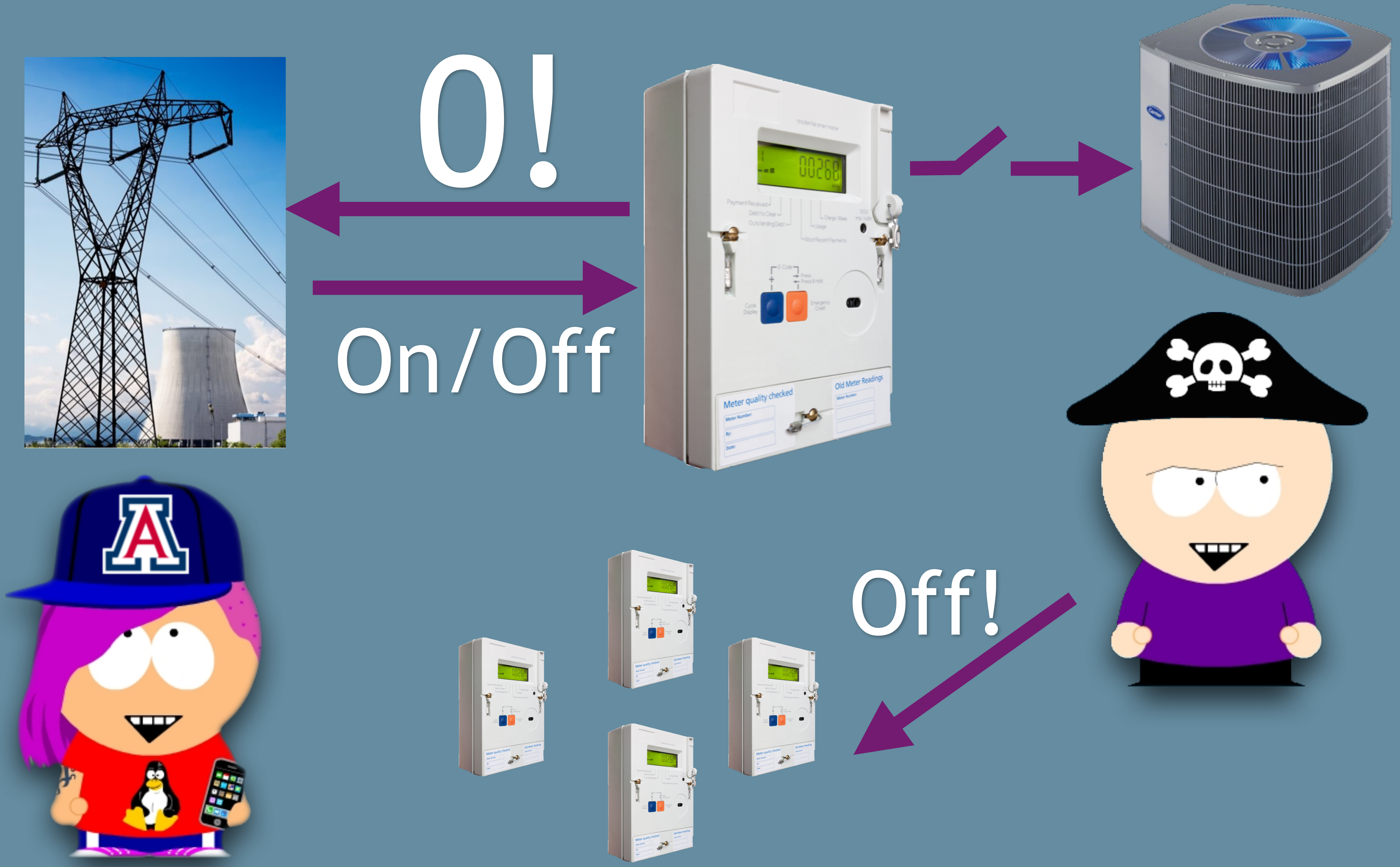
Cleemput, Mustafa, Preneel, *High Assurance Smart Metering*



0!

On/Off

Cleemput, Mustafa, Preneel, *High Assurance Smart Metering*



0!

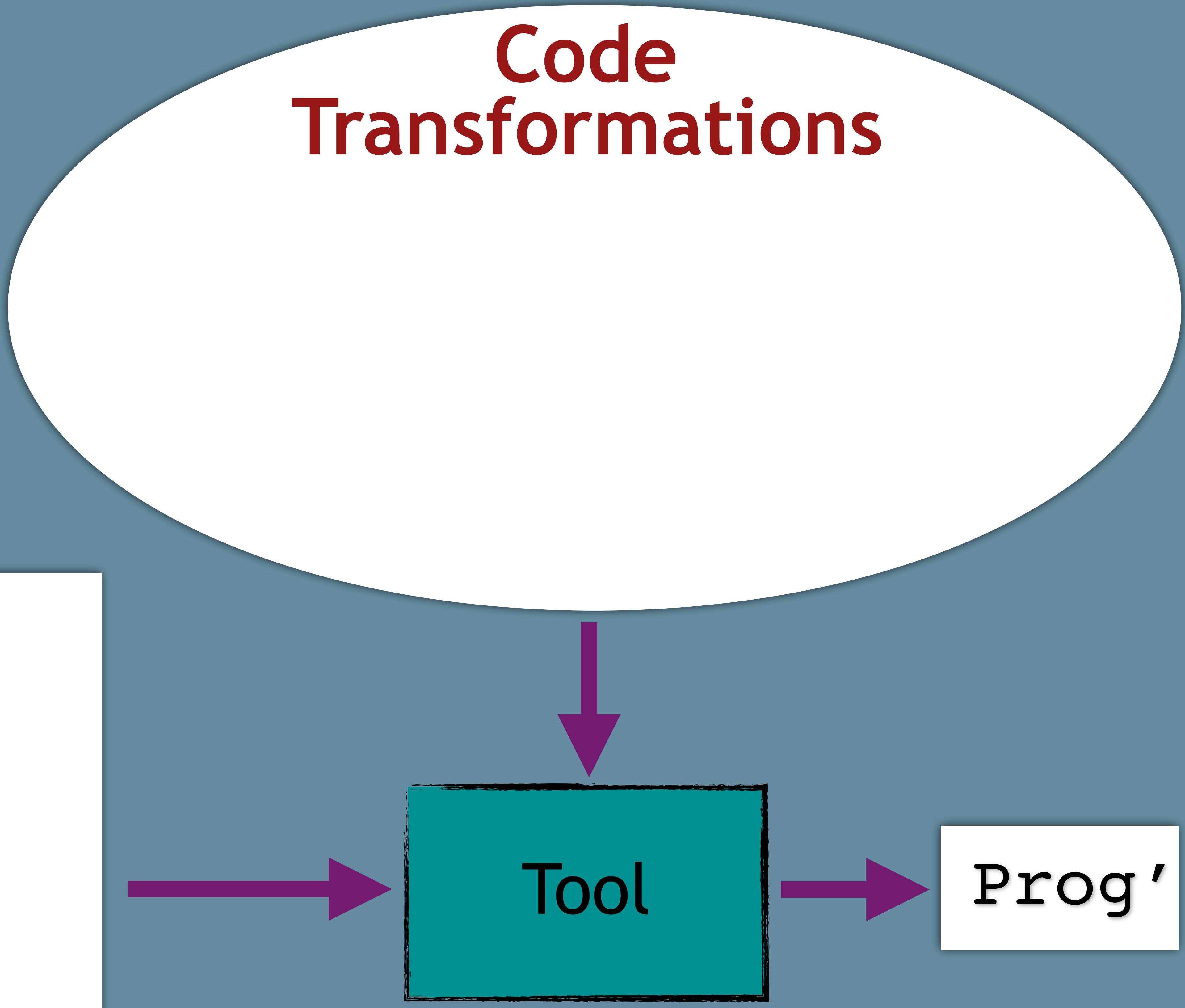
On/Off

Off!

Cleemput, Mustafa, Preneel, *High Assurance Smart Metering*

Tools
vs.
Counter Tools

Code Transformations



Tool

Prog’

```
Prog() {







}
```

Code Transformations

```
Prog() {
```

Assets

- Source
- Algorithms
- Keys
- Media

```
}
```

Tool

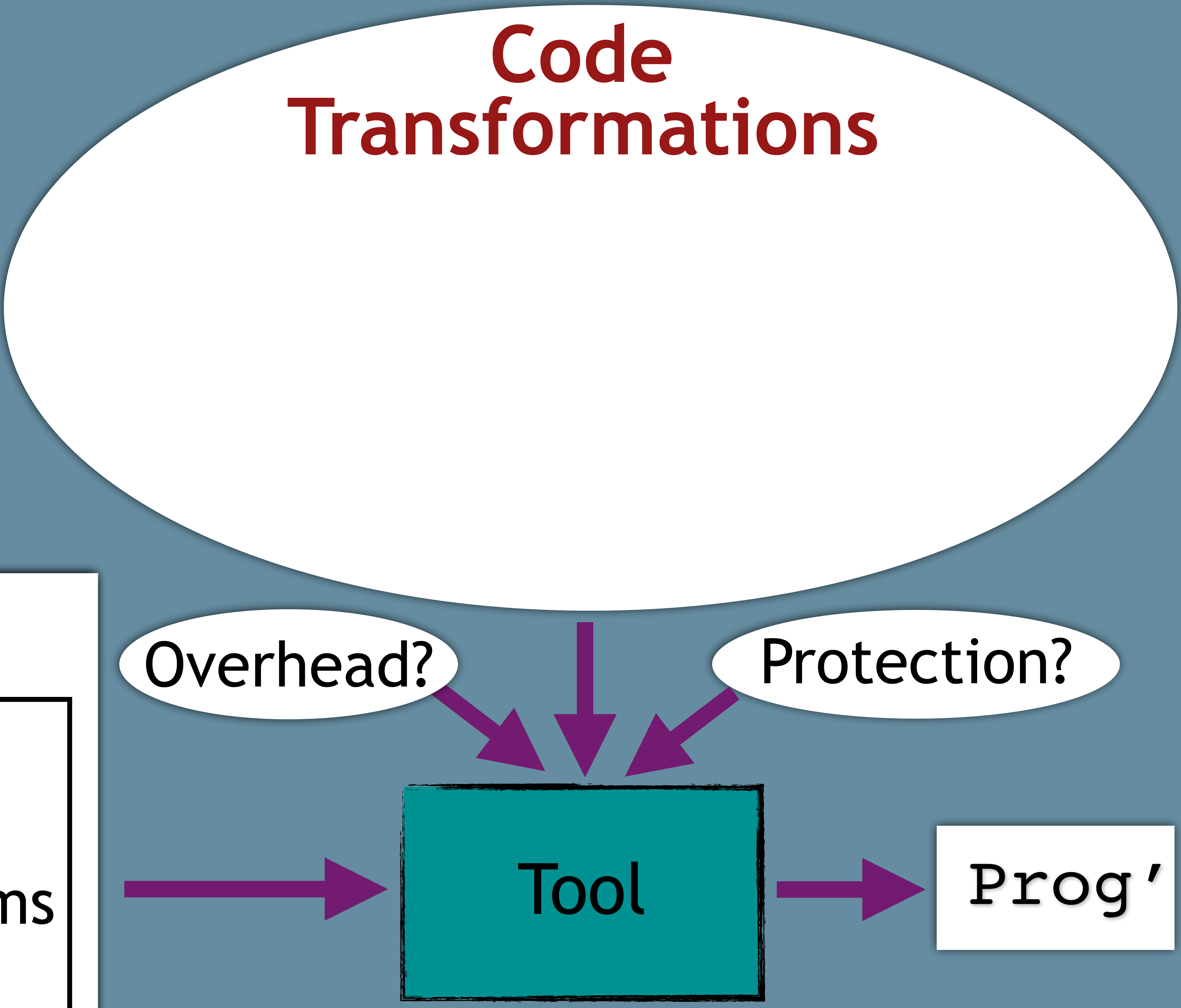
```
Prog’
```

```
Prog() {
```

Assets

- Source
- Algorithms
- Keys
- Media

```
}
```



Code Transformations

Obfuscation

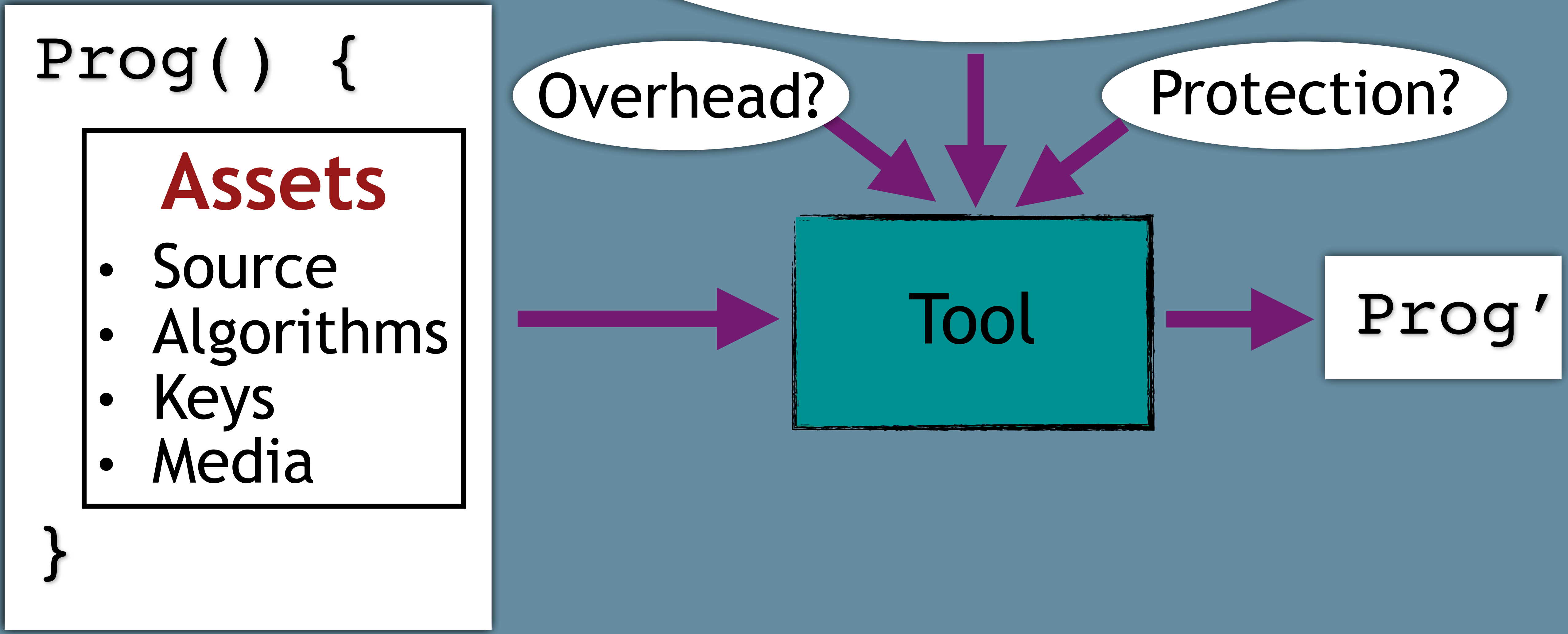
Tamperproofing

Remote Attestation

Whitebox Cryptography

Environment Checking

Watermarking



```
Prog() {
```

Assets

- Source
- Algorithms
- Keys
- Media

```
}
```

Overhead?

Protection?

Tool

```
Prog'
```

Code Transformations

Obfuscation

Tamperproofing

Remote Attestation

Whitebox Cryptography

Environment Checking

Watermarking

```
Prog() {
```

Assets

- Source
- Algorithms
- Keys
- Media

```
}
```

Aspire

NAGRA KUDELSKI GROUP

ARXAN

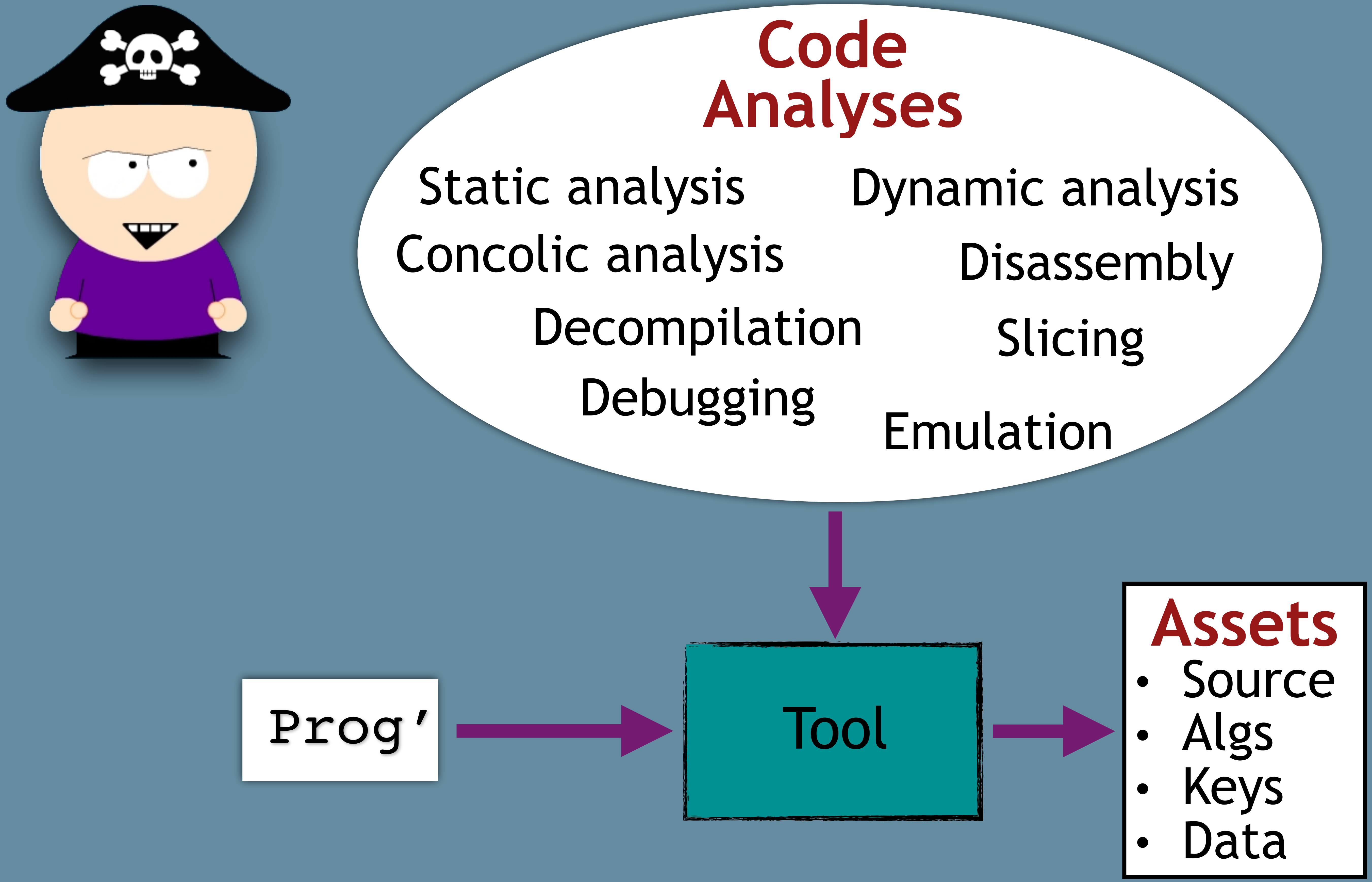
code Virtualizer

irdeto

Tigress

VMProtect software

Obfuscator-LLVM



Code Analyses

Static analysis

Dynamic analysis

Concolic analysis

Disassembly

Decompilation

Slicing

Debugging

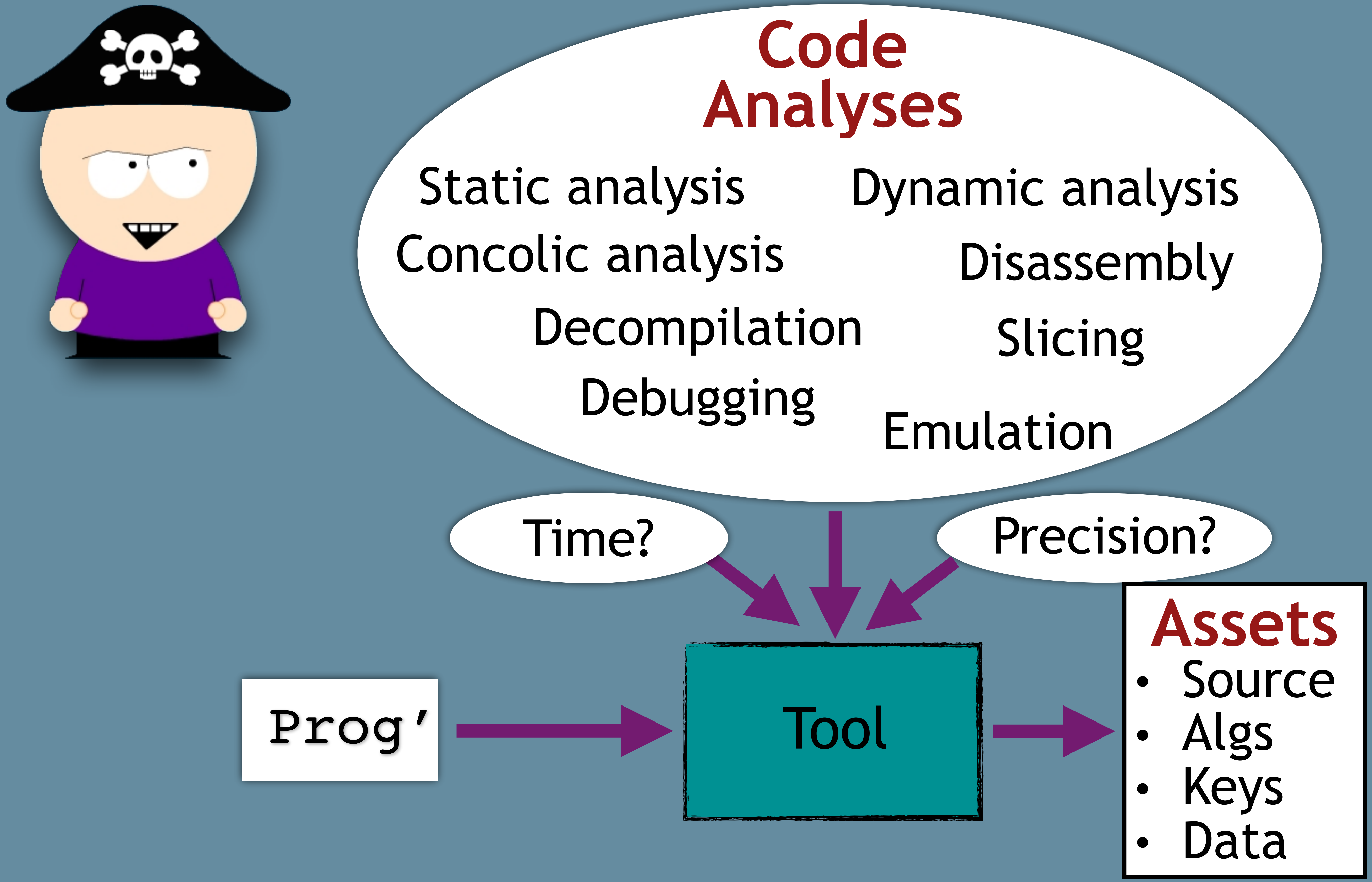
Emulation

```
Prog'
```

Tool

Assets

- Source
- Algs
- Keys
- Data



Code Analyses

Static analysis

Dynamic analysis

Concolic analysis

Disassembly

Decompilation

Slicing

Debugging

Emulation

Time?

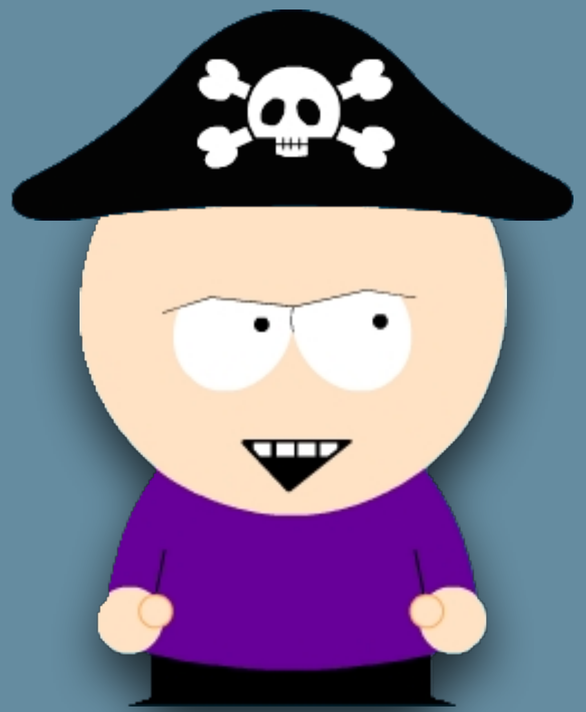
Precision?

Prog’

Tool

Assets

- Source
- Algs
- Keys
- Data



Code Analyses

Static analysis

Dynamic analysis

Concolic analysis

Disassembly

Decompilation

Slicing

Debugging

Emulation

KLEE

TRITON
Dynamic Binary Analysis

Prog’

Hex-Rays
state-of-the-art code analysis

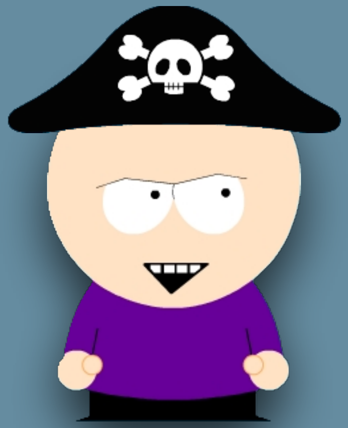
angr

S^2E

What Matters?

Performance

Time-to-Crack



S²E

angr

TRITON
Dynamic Binary Analysis

KLEE

Hex-Rays
state-of-the-art code analysis

Stealth

Performance Matters?

Metric	Program	Slowdown
absolute time	application	<1s
relative	application	1.5x
relative	security kernel	100x-1000x

Liem, Gu, Johnson: A compiler-based infrastructure for software-protection, PLAS'08

Performance Matters?

Metric	Program	Slowdown
absolute time	application	<1s
relative	application	1.5x
relative	security kernel	100x-1000x

Code virtualizer	ExeCryptor	VMProtect	Themida
100x	700x	500x	1200x

Liem, Gu, Johnson: A compiler-based infrastructure for software-protection, PLAS'08

Indistinguishability Obf.

Program	Generate	Run
2-bit multiplier	1027 years	10^8 years
16-bit point function	7 hours, 25G	4 hours (later, 20 minutes)

Bernstein et al., Bad Directions in Cryptographic Hash Functions, IS&P'15

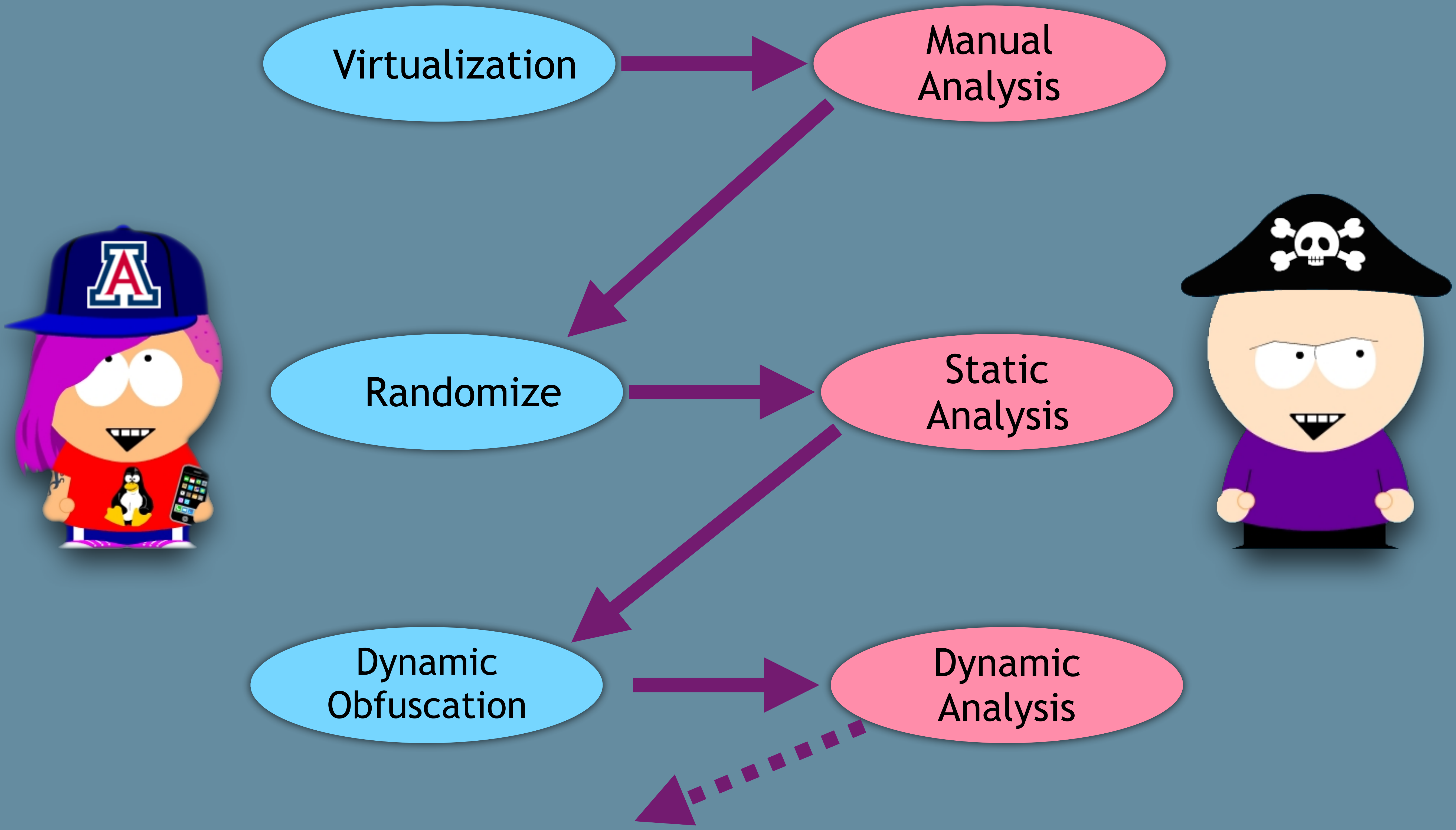
Apon, et al., Impl. Cryptographic Program Obfuscation, CRYPTO'14

Banescu, et al, Benchmarking Indistinguishability Obf. - A candidate impl.

Time-to-Crack Matters

Program	Adversary	Time
hw+sw		many years
well protected	highly skilled, motivated	4-6 weeks
≈VMProtect	experienced reverse engineer	≈12 months
mass market malware		minutes-hours

Obfuscation
vs.
Deobfuscation



Virtualization

Manual Analysis

Randomize

Static Analysis

Dynamic Obfuscation

Dynamic Analysis

P_0

Tigress

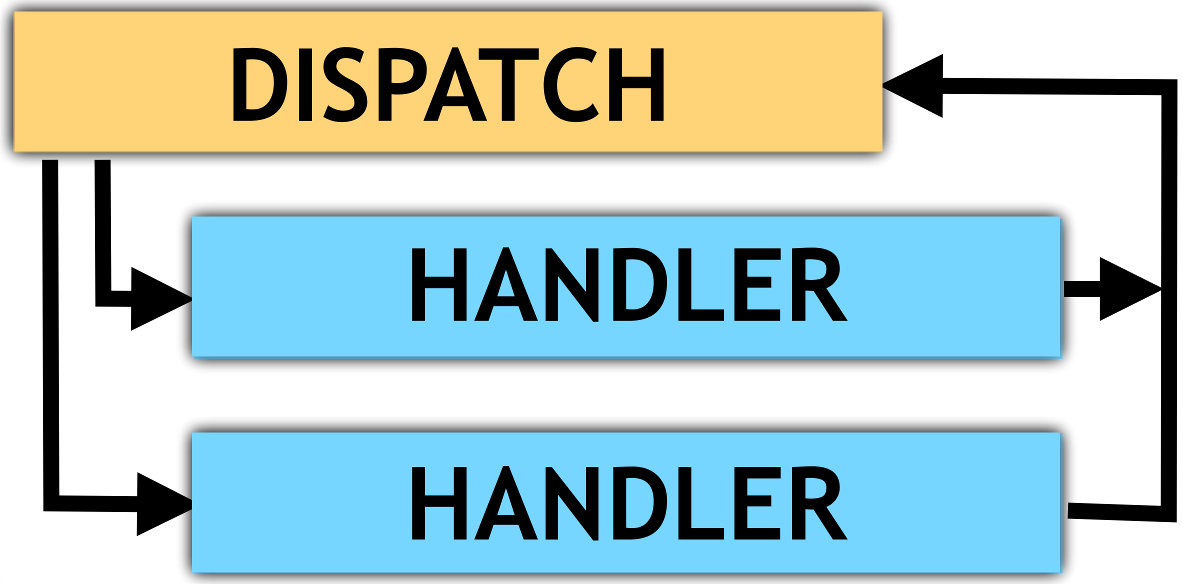
Virtual Instruction Set

Opcode	Mnemonic	Semantics
0	add	push(pop()+pop())
1	store L	Mem[L]=pop()
2	breq L	if pop()=pop() goto L

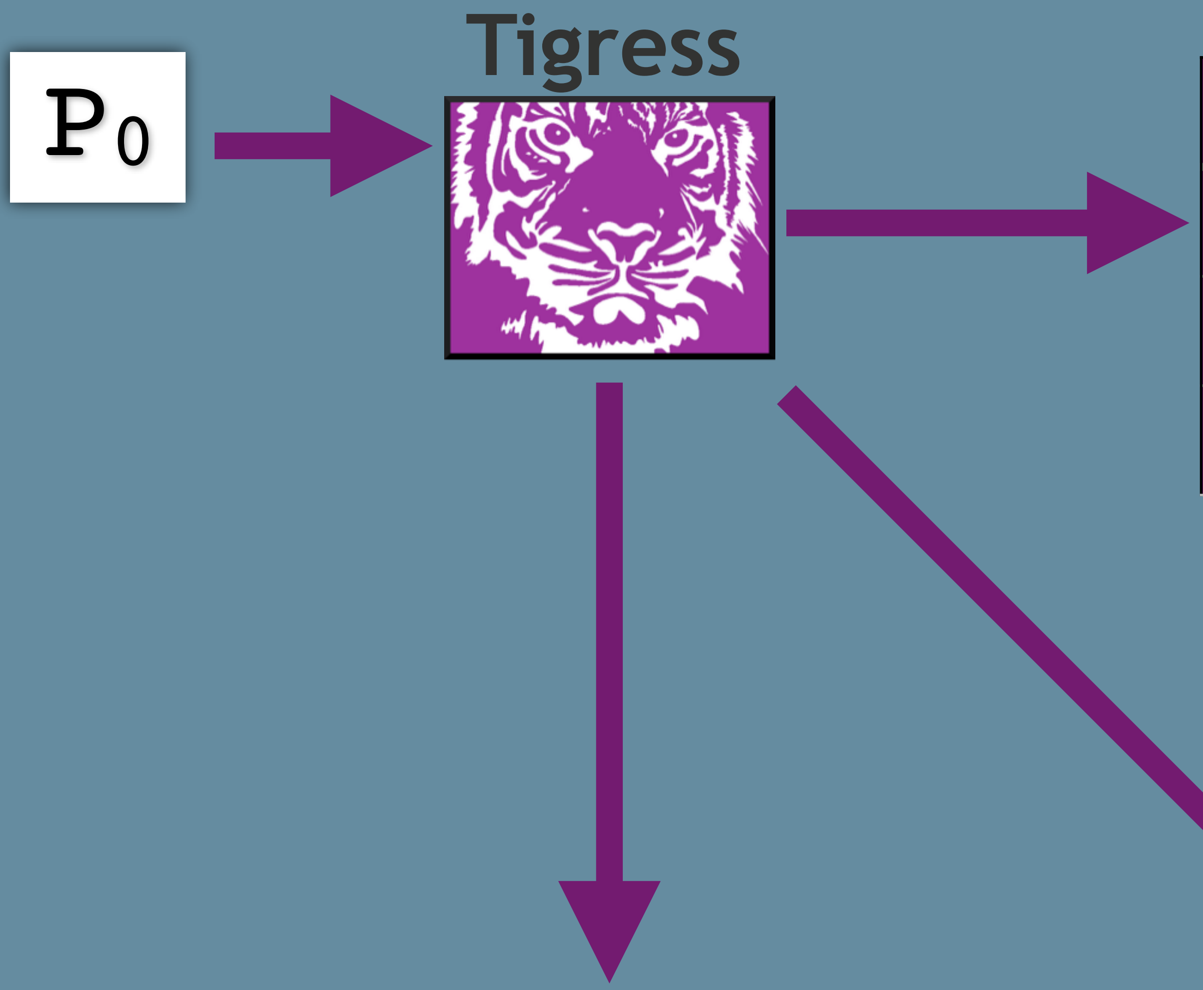
Virtual Instruction Set

Opcode	Mnemonic	Semantics
0	add	push(pop()+pop())
1	store L	Mem[L]=pop()
2	breq L	if pop()=pop() goto L

```
void P1(){
    VPC = 0;
    STACK = [];
```



```
}
```



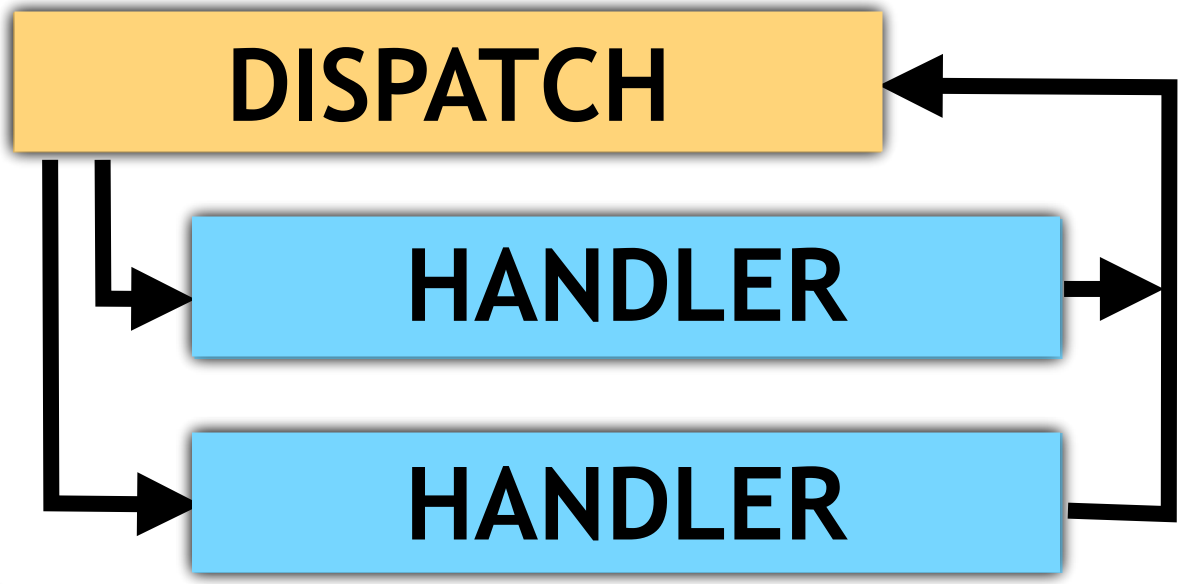
Virtual Instruction Set

Opcode	Mnemonic	Semantics
0	add	push(pop()+pop())
1	store L	Mem[L]=pop()
2	breq L	if pop()=pop() goto L

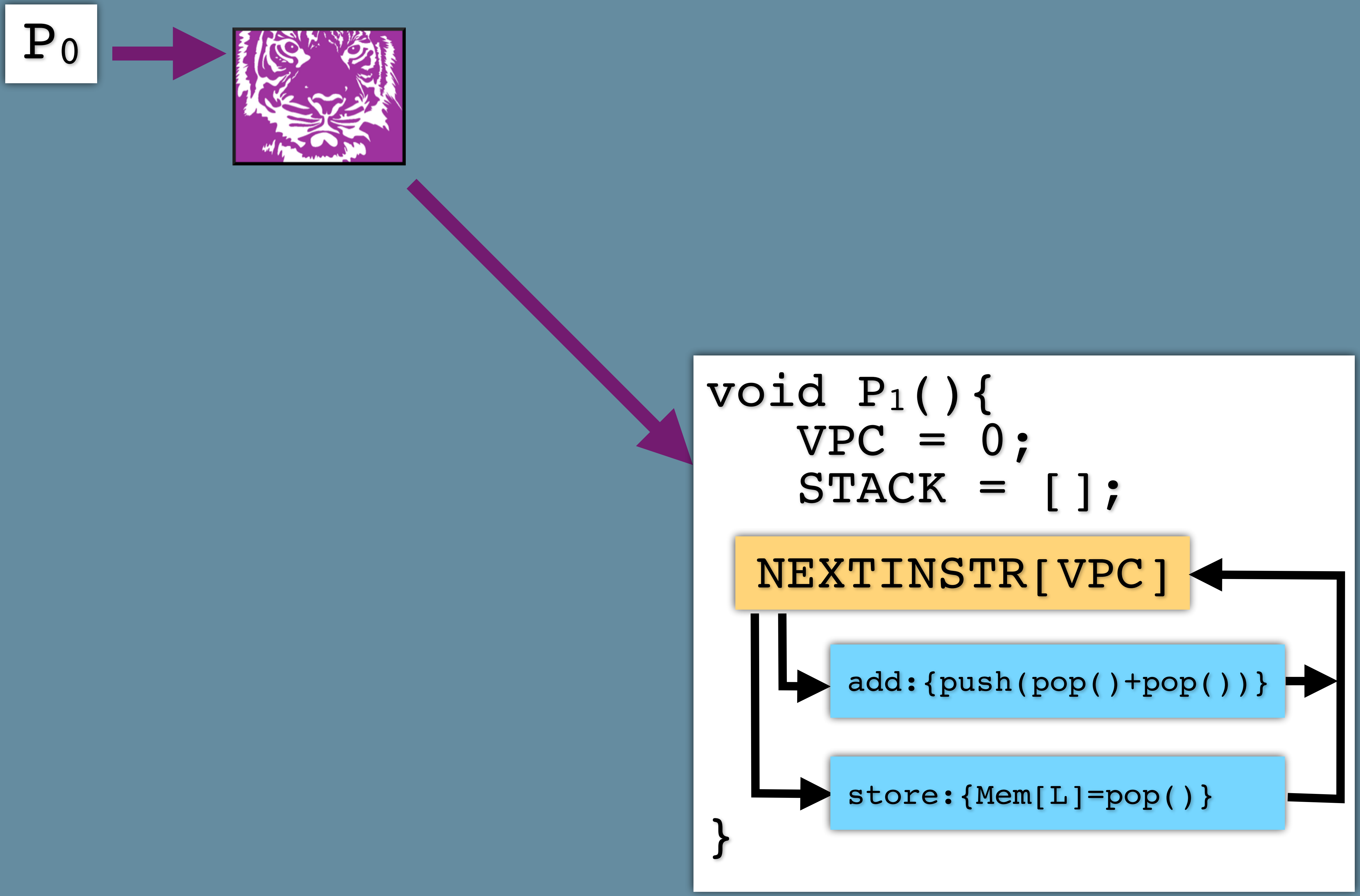
Virtual Program Array

breq L1	add	store L2	push

```
void P1(){
    VPC = 0;
    STACK = [];
```

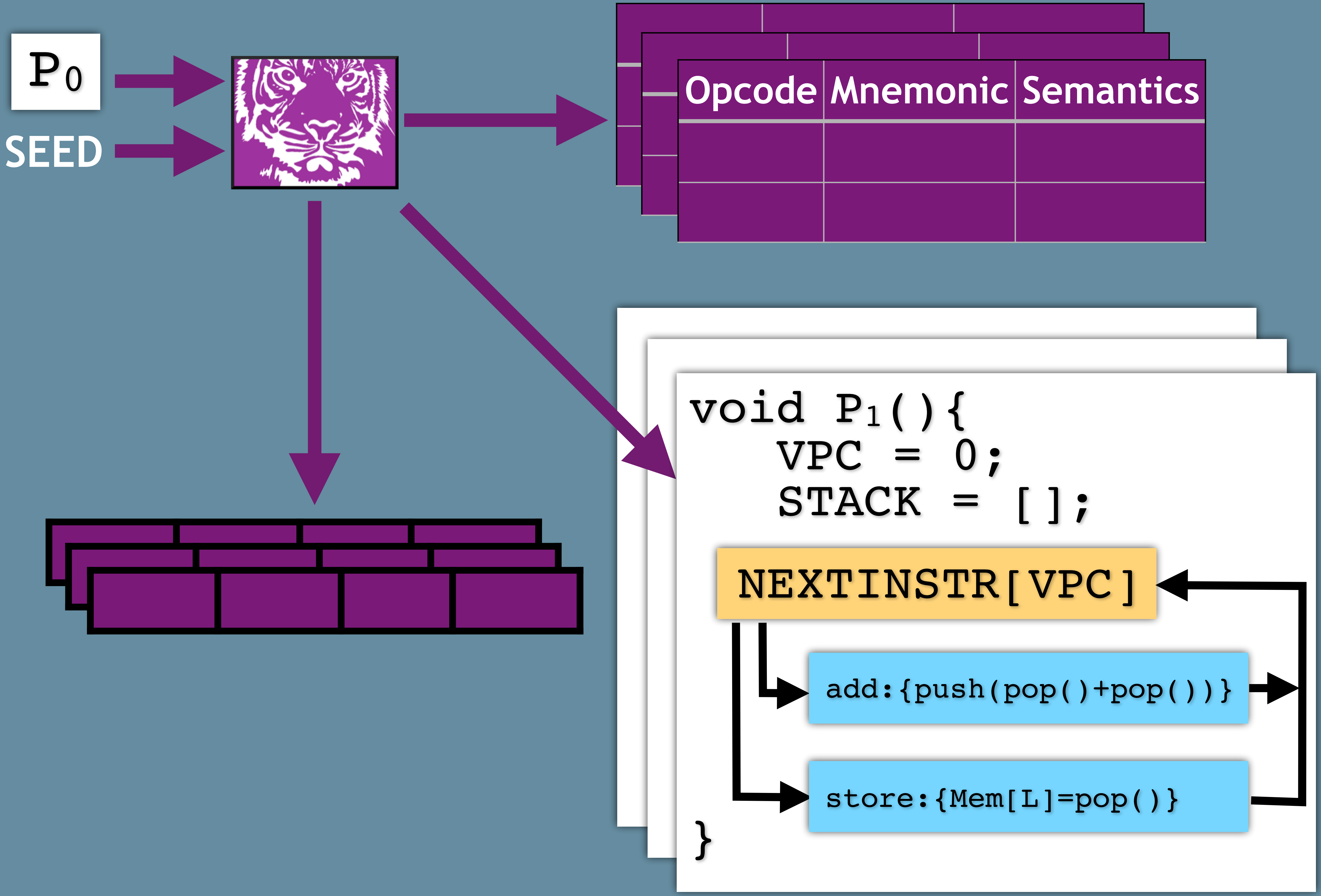


```
}
```



P_0

```
void P1(){
    VPC = 0;
    STACK = [];
  NEXTINSTR[VPC]
      add:{push(pop()+pop())}
      store:{Mem[L]=pop()}
}
```

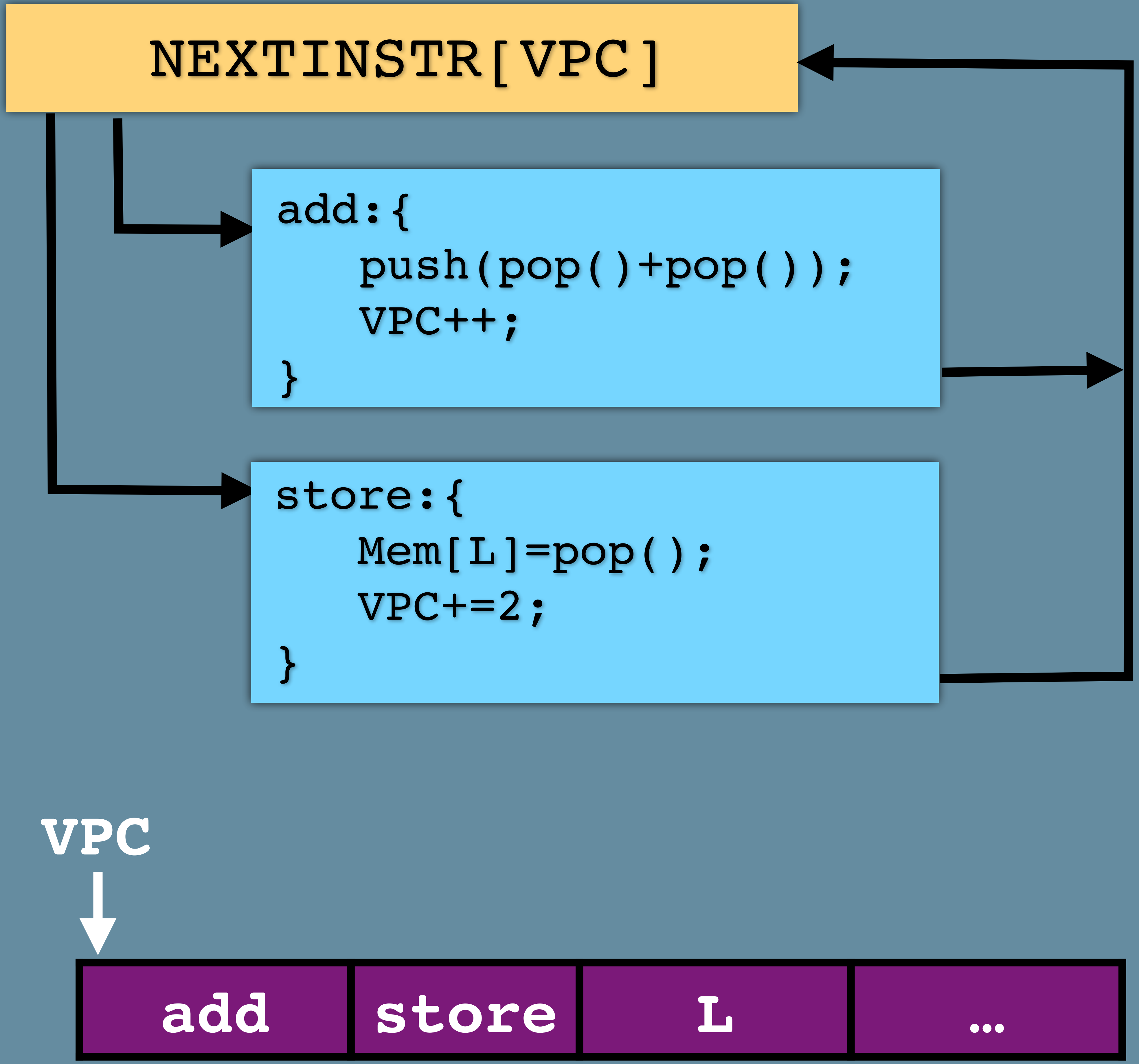


P_0

SEED

Opcode	Mnemonic	Semantics

```
void P1(){
    VPC = 0;
    STACK = [];
  NEXTINSTR[VPC]
      add:{push(pop()+pop())}
      store:{Mem[L]=pop()}
}
```



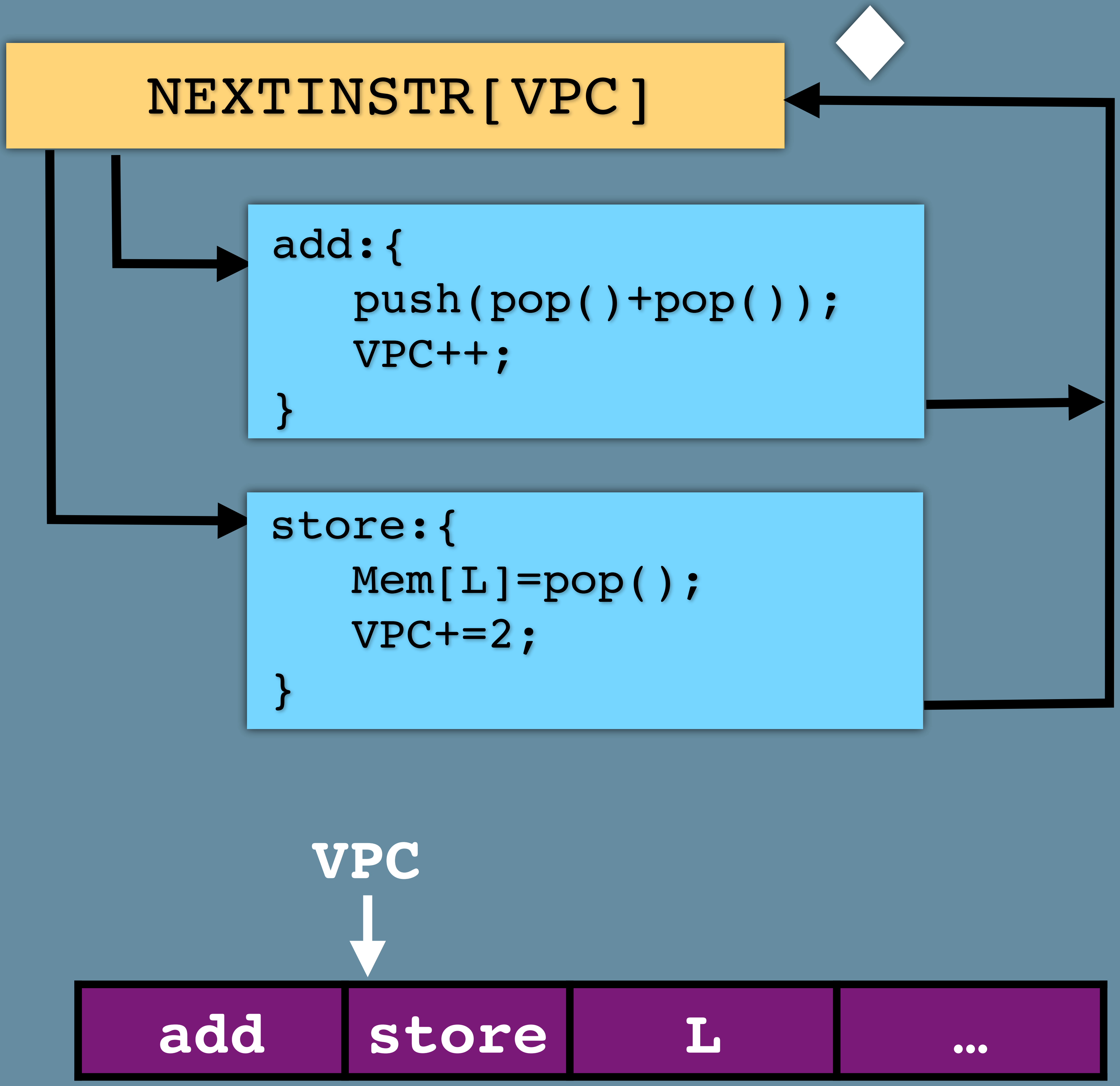
NEXTINSTR[VPC]

```
add:{
    push(pop()+pop());
    VPC++;
}
```

```
store:{
    Mem[L]=pop();
    VPC+=2;
}
```

VPC

add	store	L	…



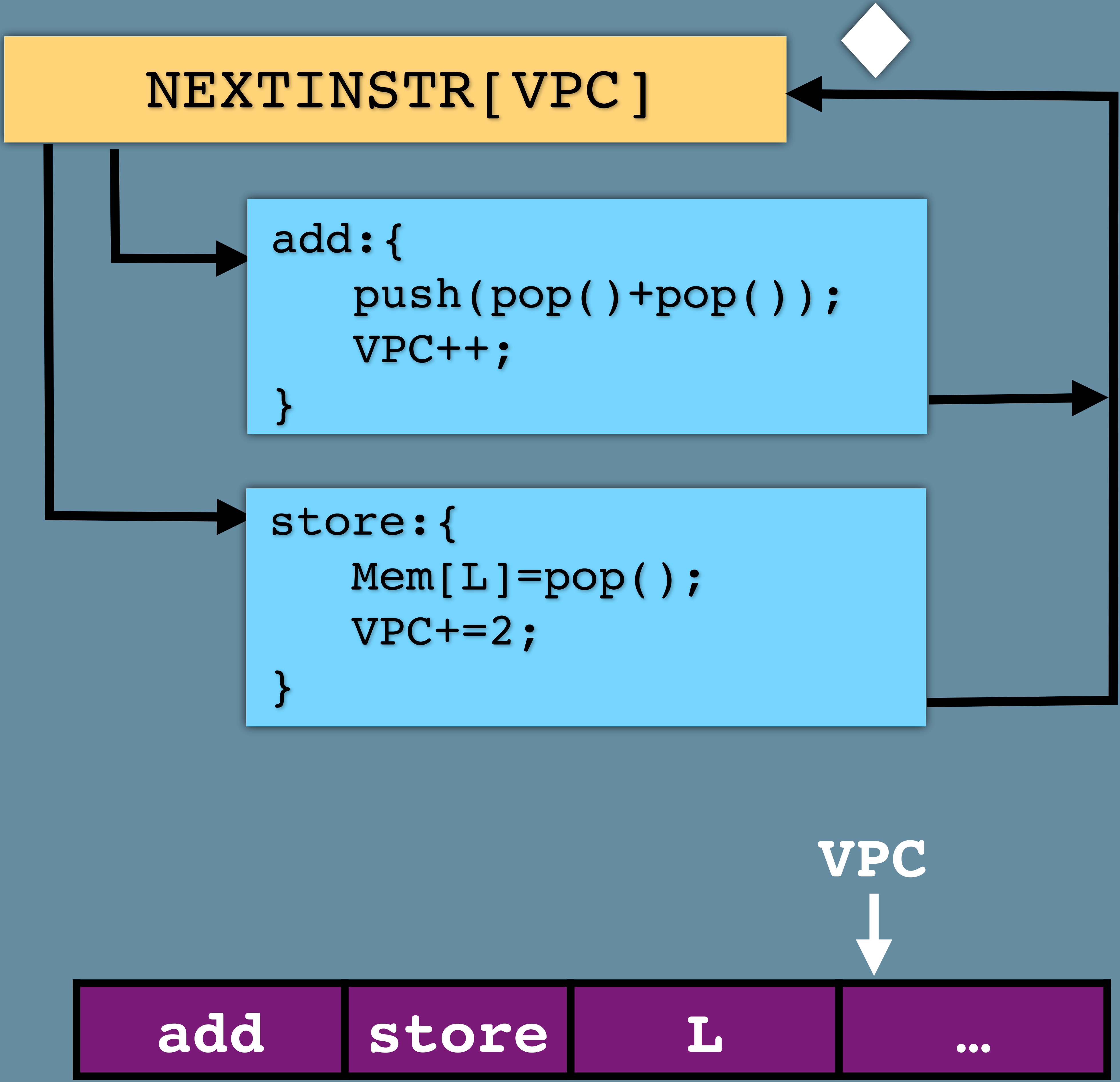
NEXTINSTR[VPC]

```
add:{
    push(pop()+pop());
    VPC++;
}
```

```
store:{
    Mem[L]=pop();
    VPC+=2;
}
```

VPC

add	store	L	…

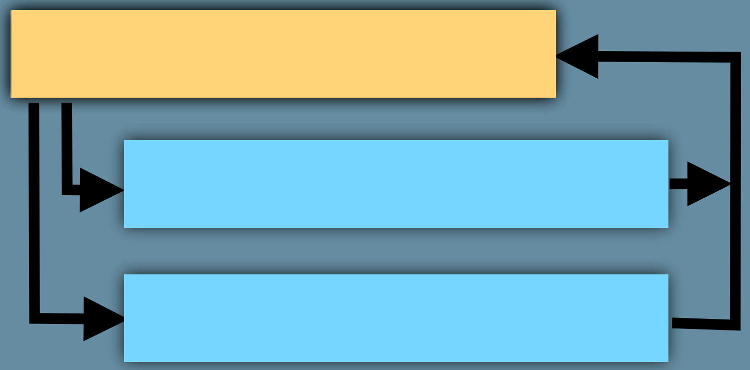


NEXTINSTR[VPC]

```
add:{
    push(pop()+pop());
    VPC++;
}
```

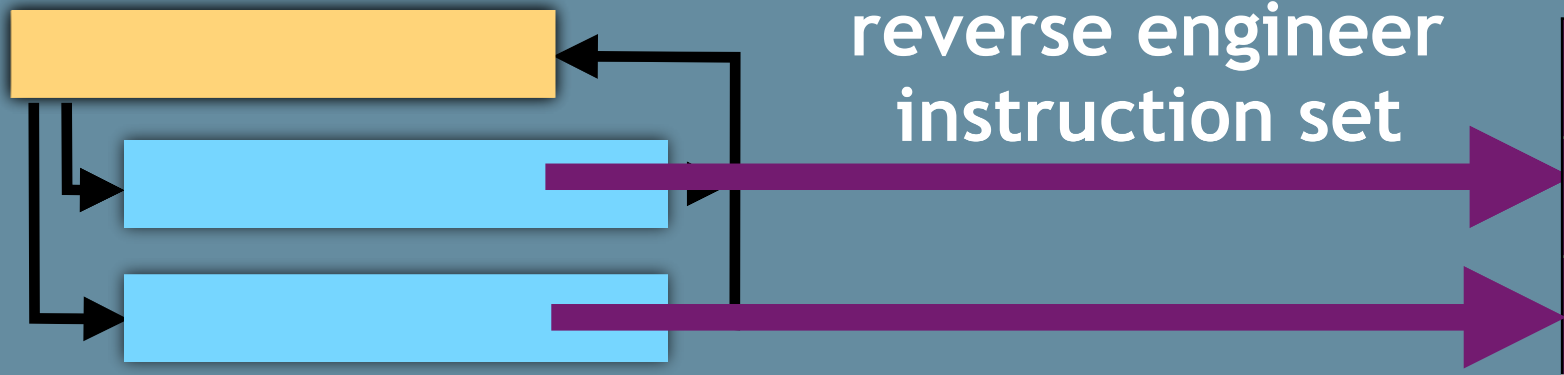
```
store:{
    Mem[L]=pop();
    VPC+=2;
}
```

Manual Analysis



Rolles, Unpacking virtualization obfuscators, WOOT'09

Manual Analysis



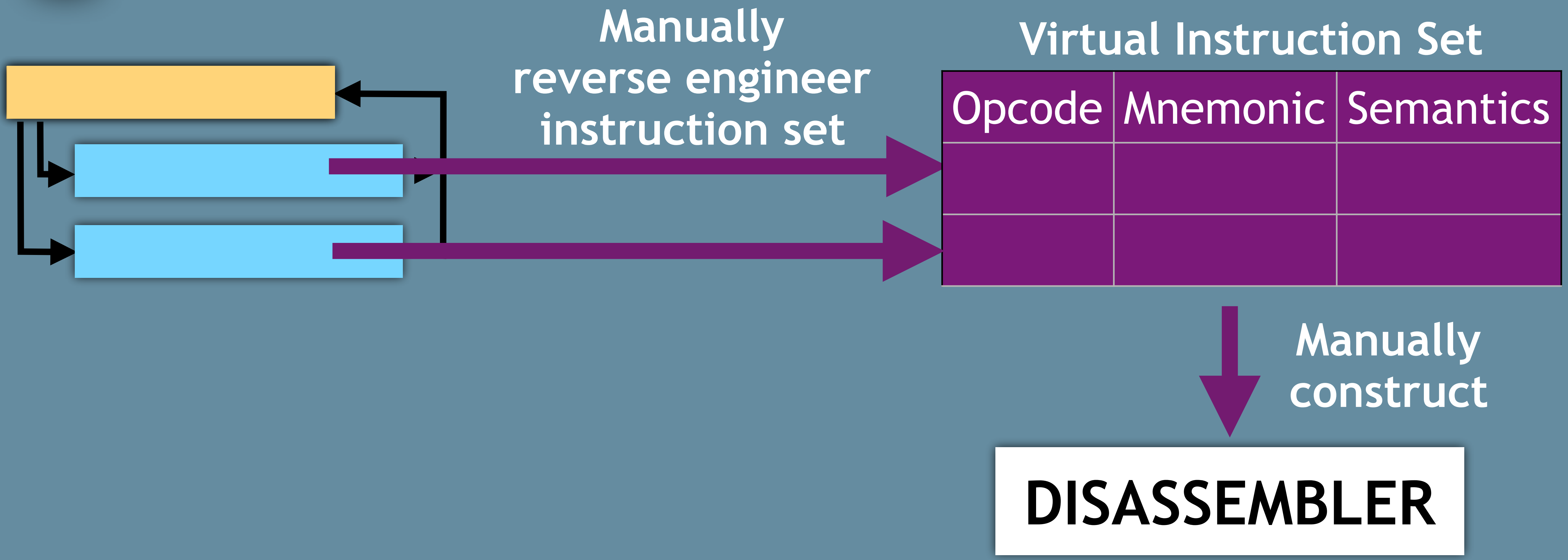
Manually
reverse engineer
instruction set

Virtual Instruction Set

Opcode	Mnemonic	Semantics

Rolles, Unpacking virtualization obfuscators, WOOT'09

Manual Analysis



Manually reverse engineer instruction set

Virtual Instruction Set

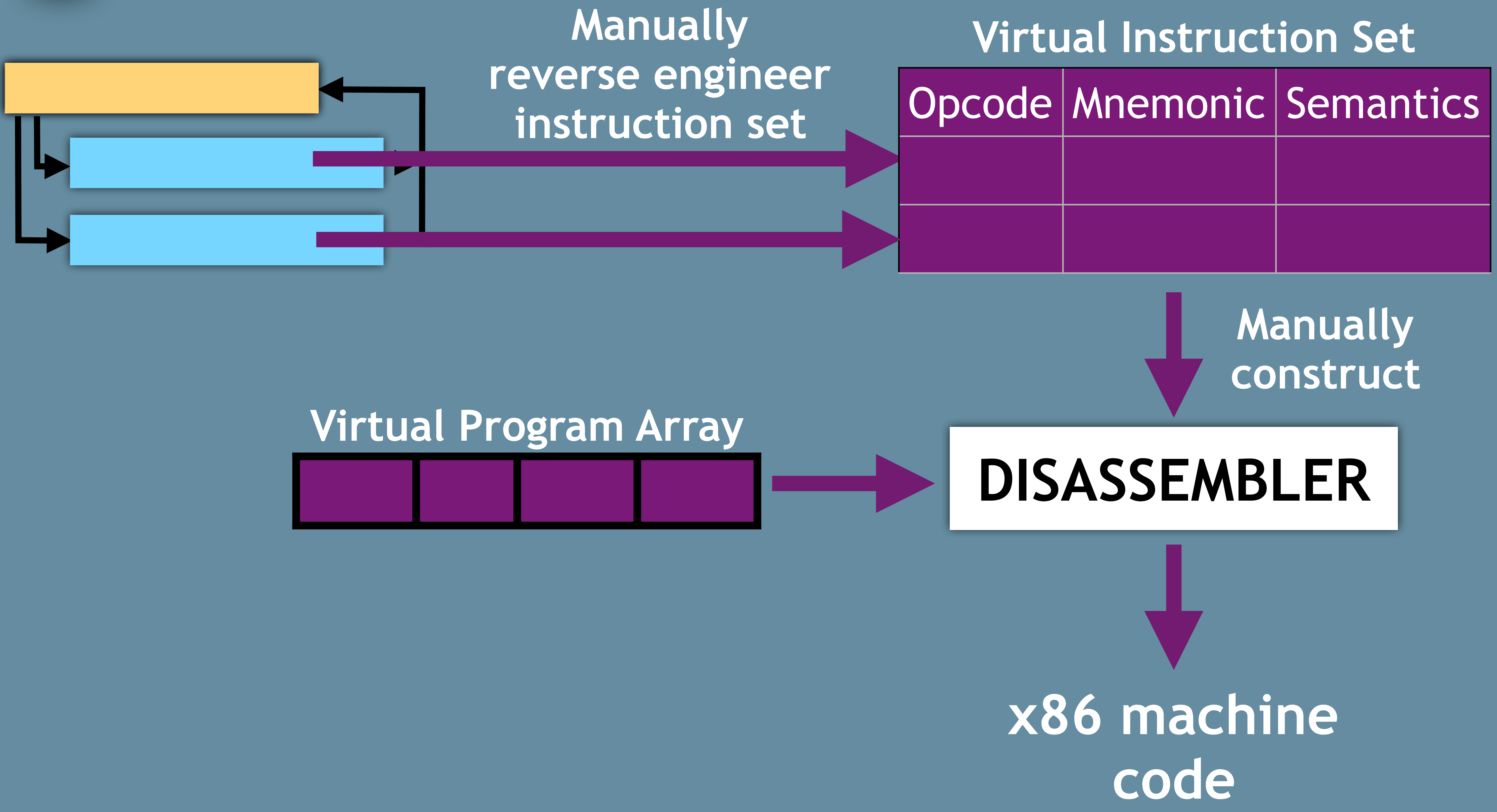
Opcode	Mnemonic	Semantics

Manually construct

DISASSEMBLER

Rolles, Unpacking virtualization obfuscators, WOOT'09

Manual Analysis



Manually reverse engineer instruction set

Virtual Instruction Set

Opcode	Mnemonic	Semantics

Manually construct

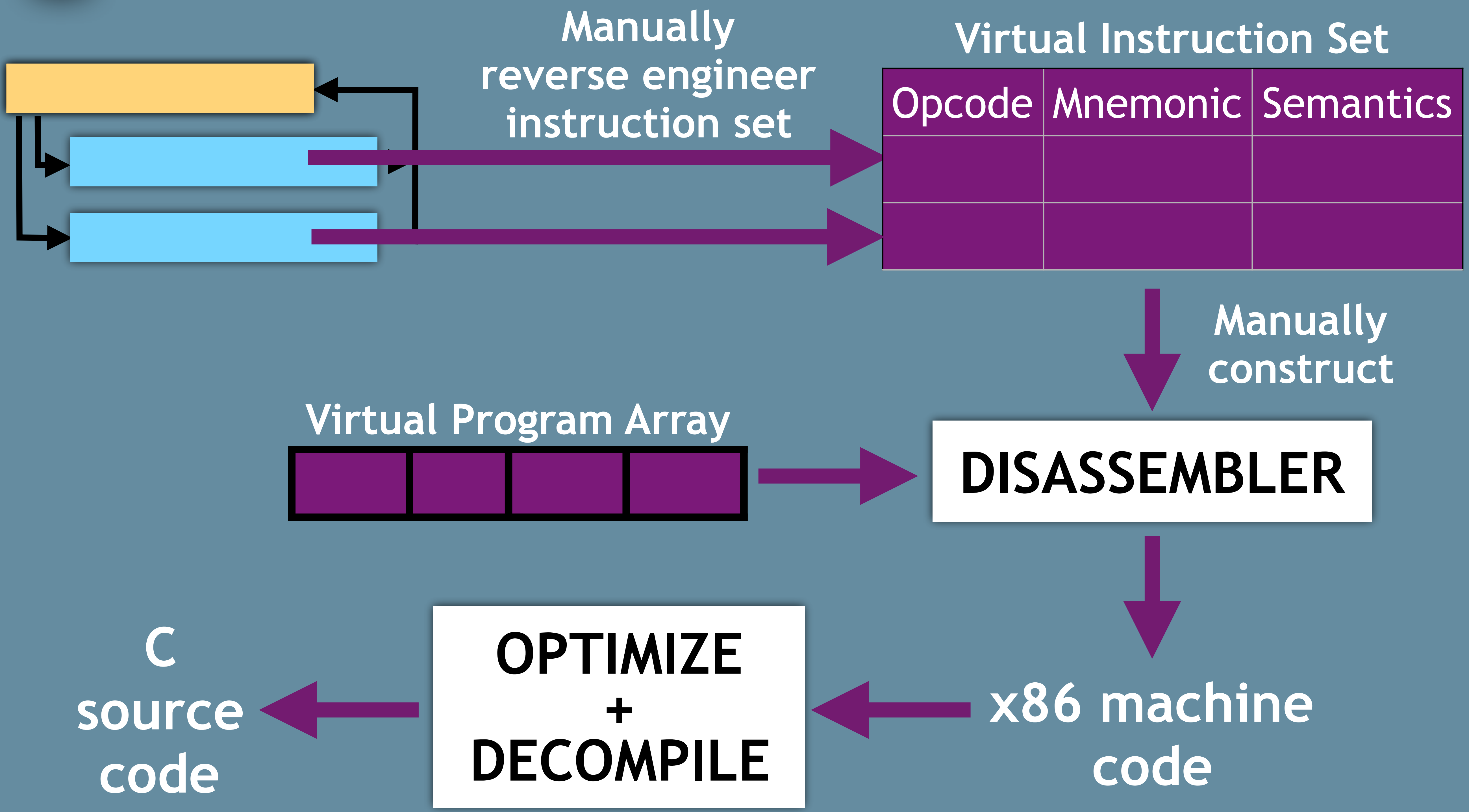
Virtual Program Array

DISASSEMBLER

x86 machine code

Rolles, Unpacking virtualization obfuscators, WOOT'09

Manual Analysis



Manually reverse engineer instruction set

Virtual Instruction Set

Opcode	Mnemonic	Semantics

Manually construct

Virtual Program Array

DISASSEMBLER

x86 machine code

OPTIMIZE + DECOMPILE

C source code

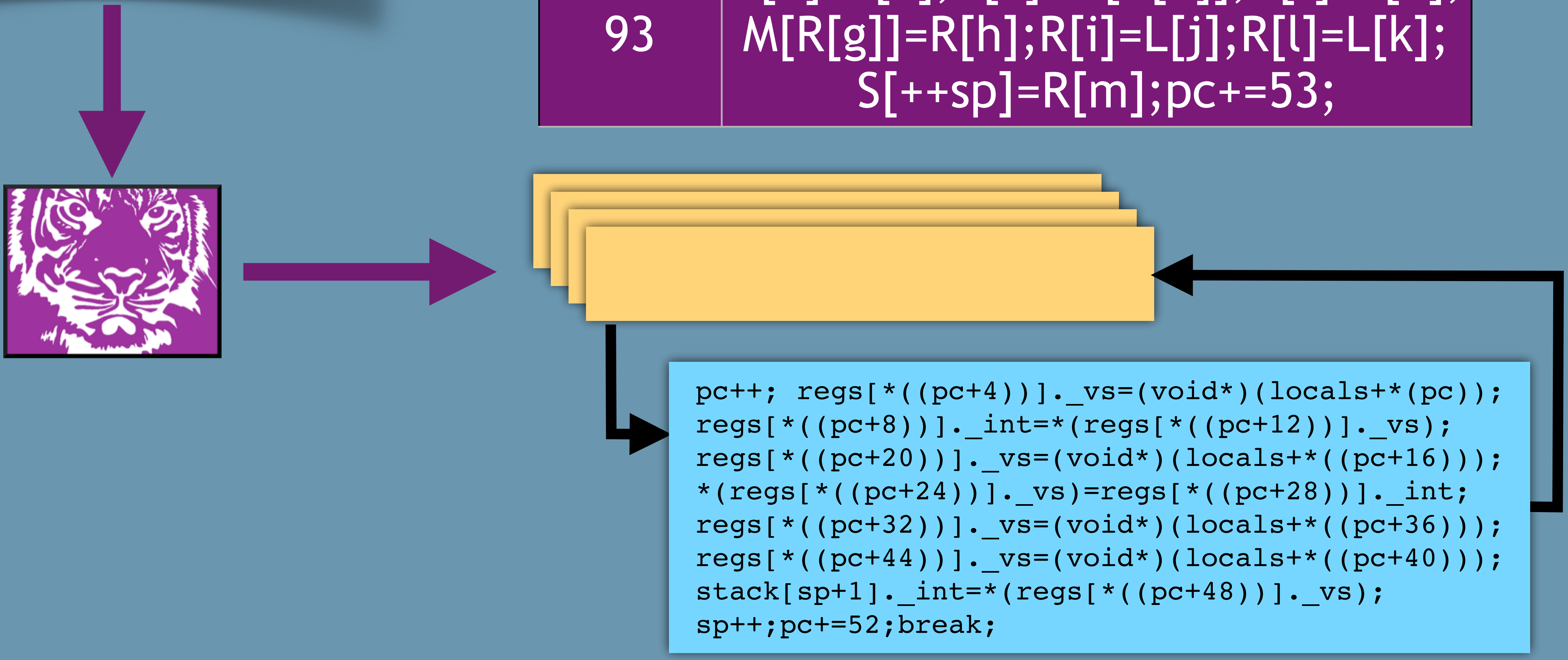
Rolles, Unpacking virtualization obfuscators, WOOT'09

Randomize

- Superoperators
- Randomize operands
- Randomize opcodes
- Random dispatch

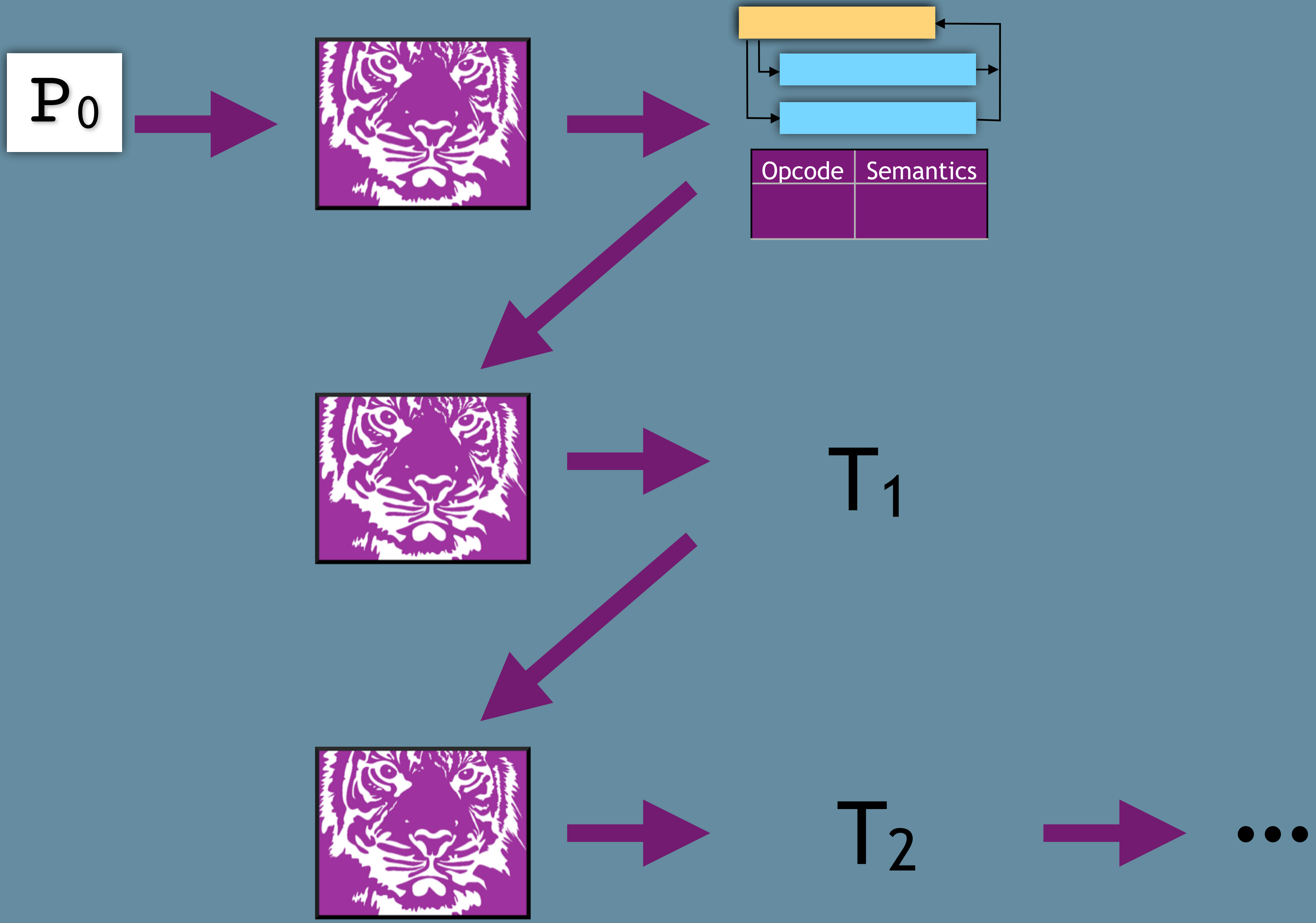
Opcode	Semantics

93	R[b]=L[a];R[c]=M[R[d]];R[f]=L[e];
	M[R[g]]=R[h];R[i]=L[j];R[l]=L[k];
	S[++sp]=R[m];pc+=53;

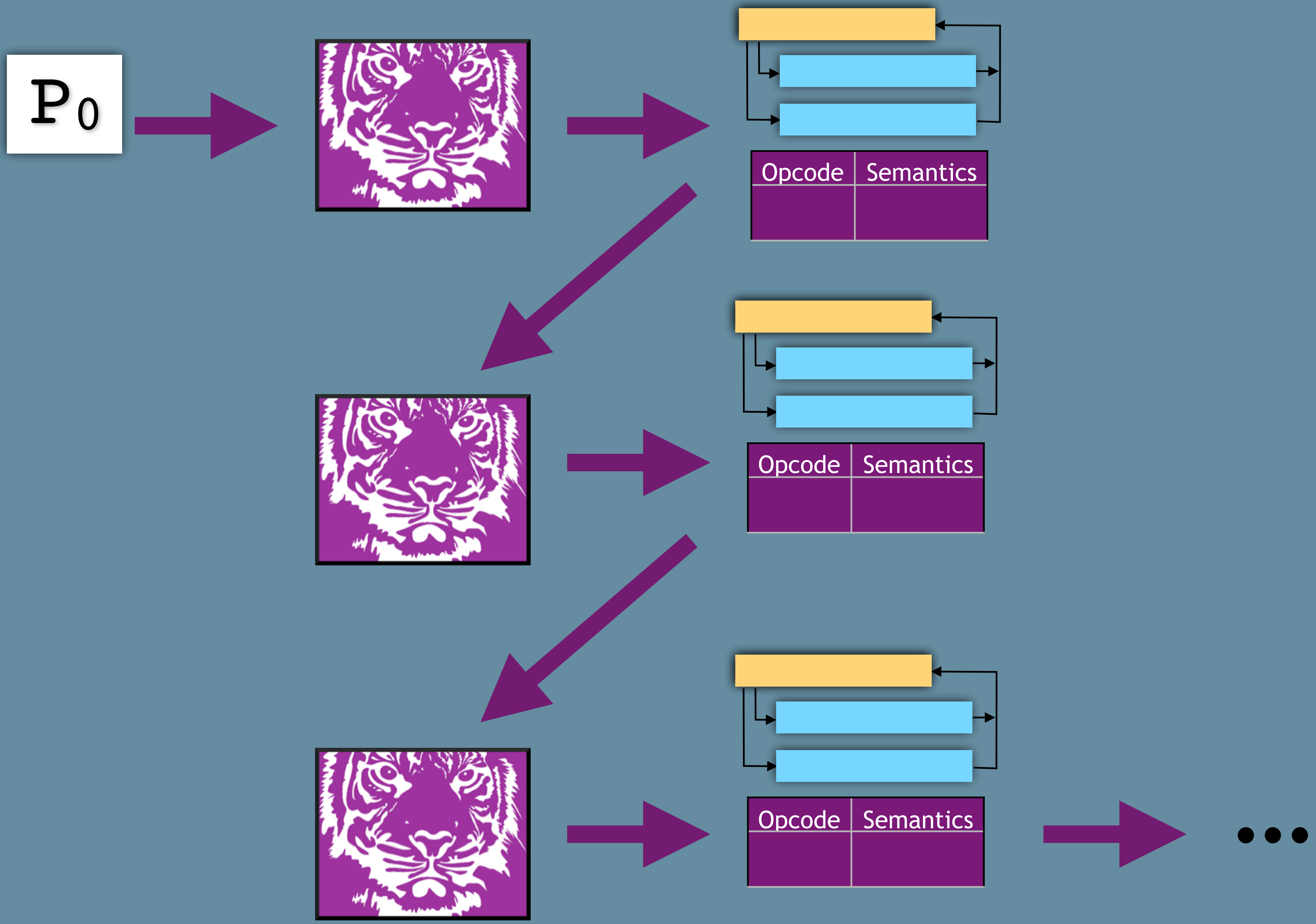


```
pc++; regs[*((pc+4))]._vs=(void*)(locals+*(pc));
regs[*((pc+8))]._int=*(regs[*((pc+12))]._vs);
regs[*((pc+20))]._vs=(void*)(locals+*((pc+16)));
*(regs[*((pc+24))]._vs)=regs[*((pc+28))]._int;
regs[*((pc+32))]._vs=(void*)(locals+*((pc+36)));
regs[*((pc+44))]._vs=(void*)(locals+*((pc+40)));
stack[sp+1]._int=*(regs[*((pc+48))]._vs);
sp++;pc+=52;break;
```

Composition

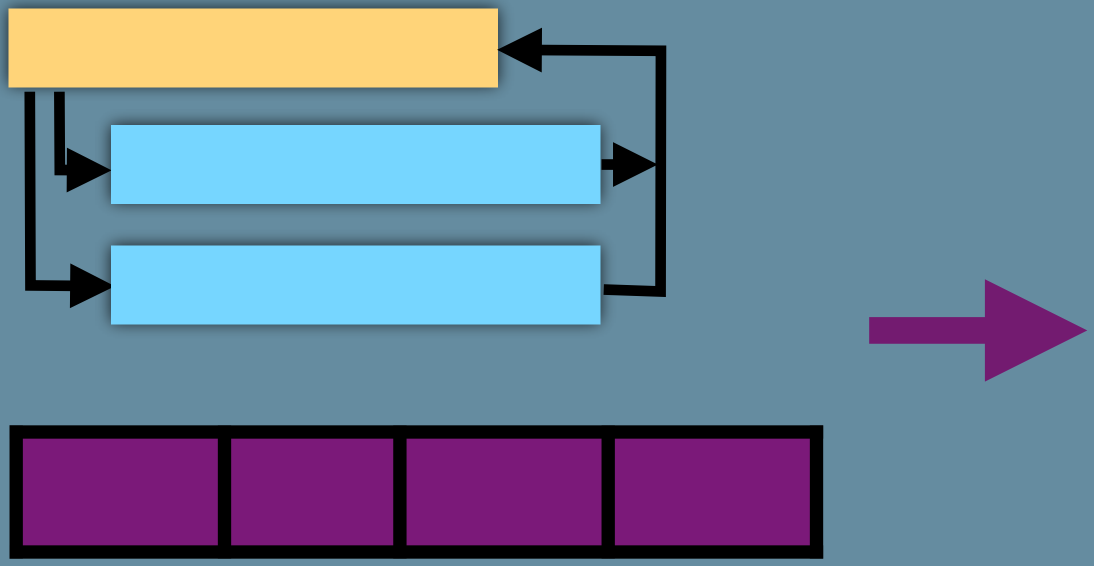


Composition



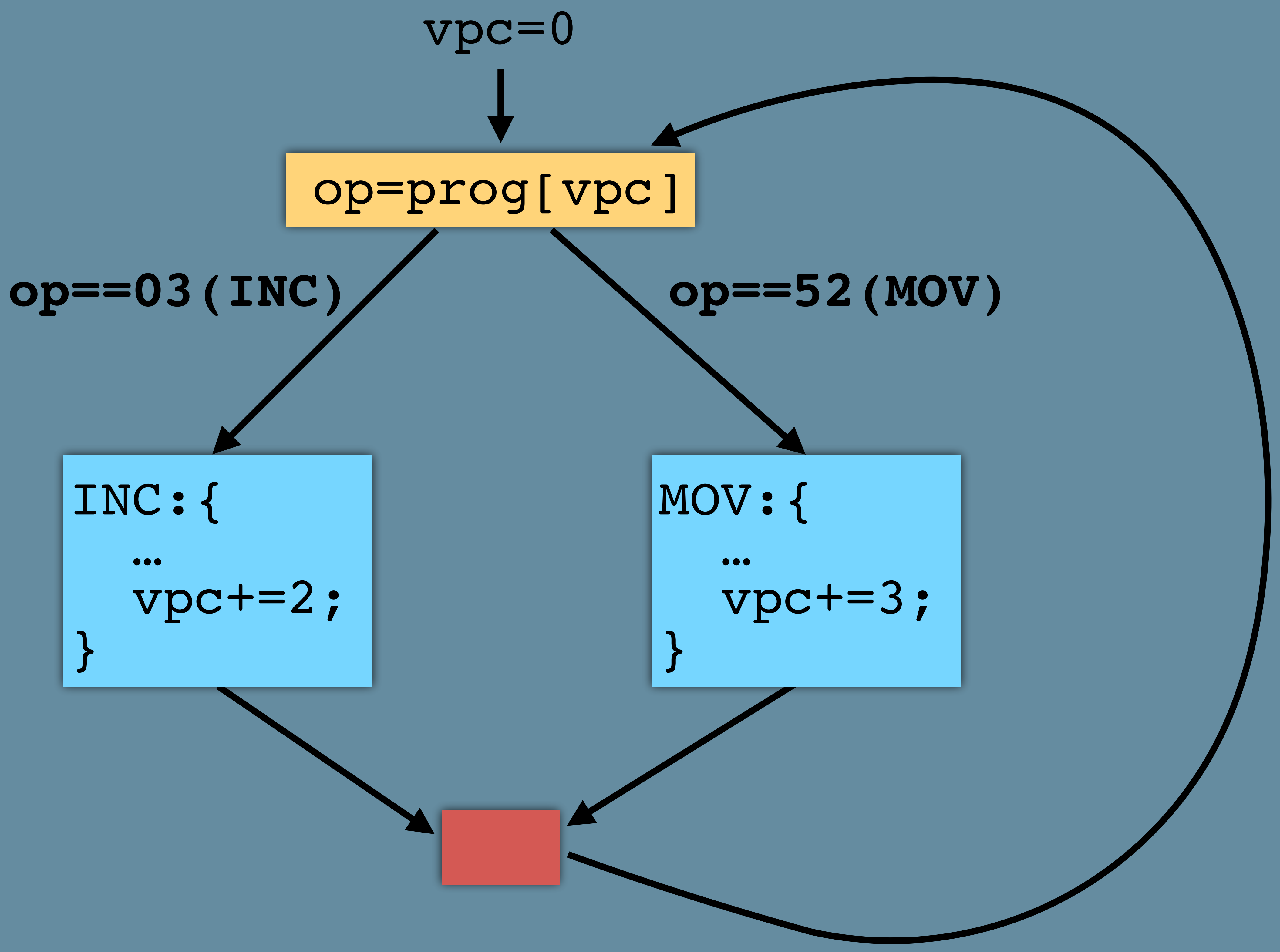
Static Analysis

- Automatically reason about the program without executing it
- A sound analysis computes a valid over-approximation of the program semantics



Property that holds for all possible executions of the program

Kinder, Towards Static Analysis of Virtualization-Obfuscated Binaries, WCRE'12



vpc=0

```
op=prog[vpc]
```

op==03(INC)

op==52(MOV)

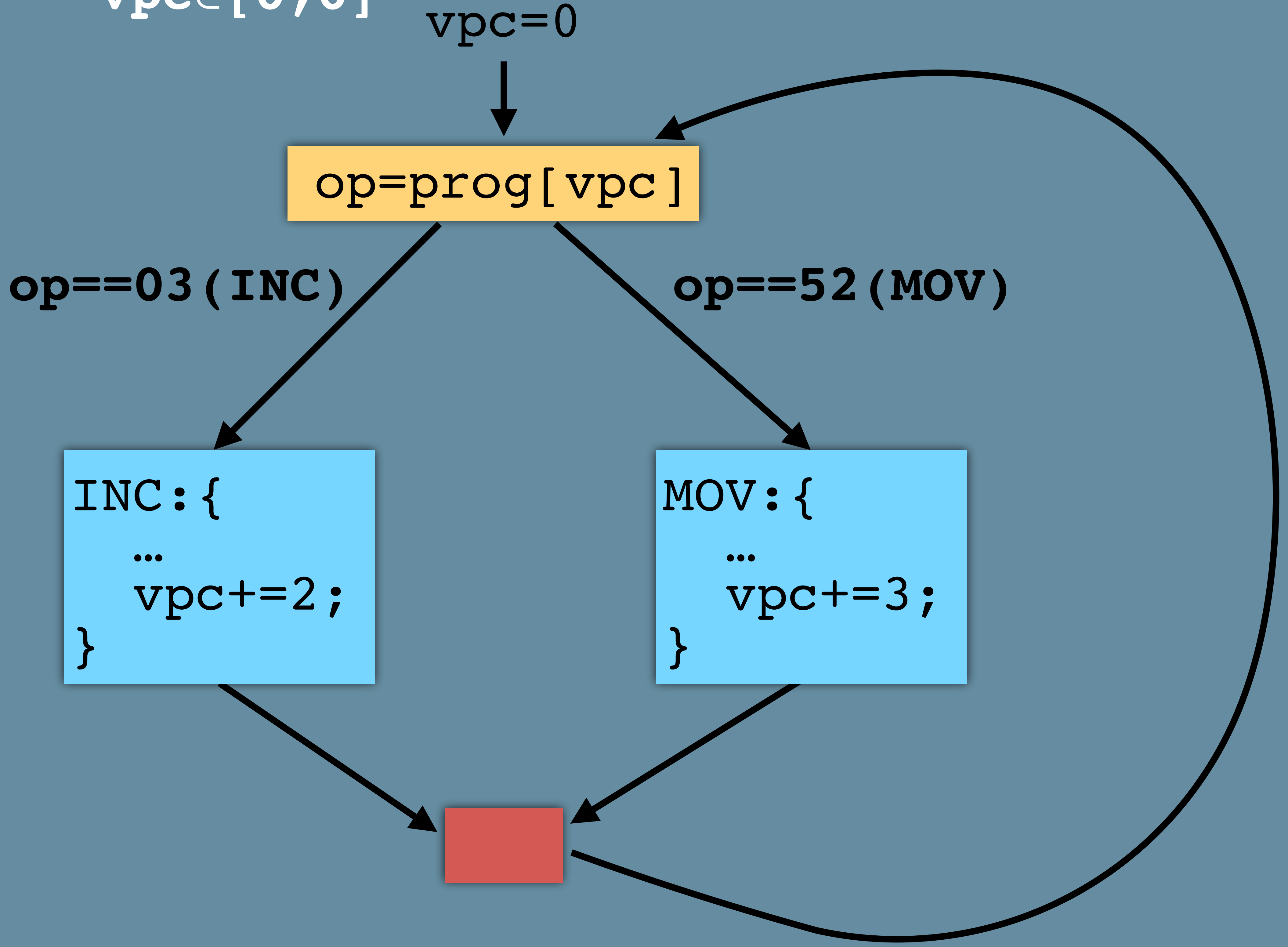
```
INC:{
   …
   vpc+=2;
}
```

```
MOV:{
   …
   vpc+=3;
}
```

0	1	2	3	4	5	6	
52	01	02	03	01	03	01	…
MOV	x	y	INC	x	INC	x	

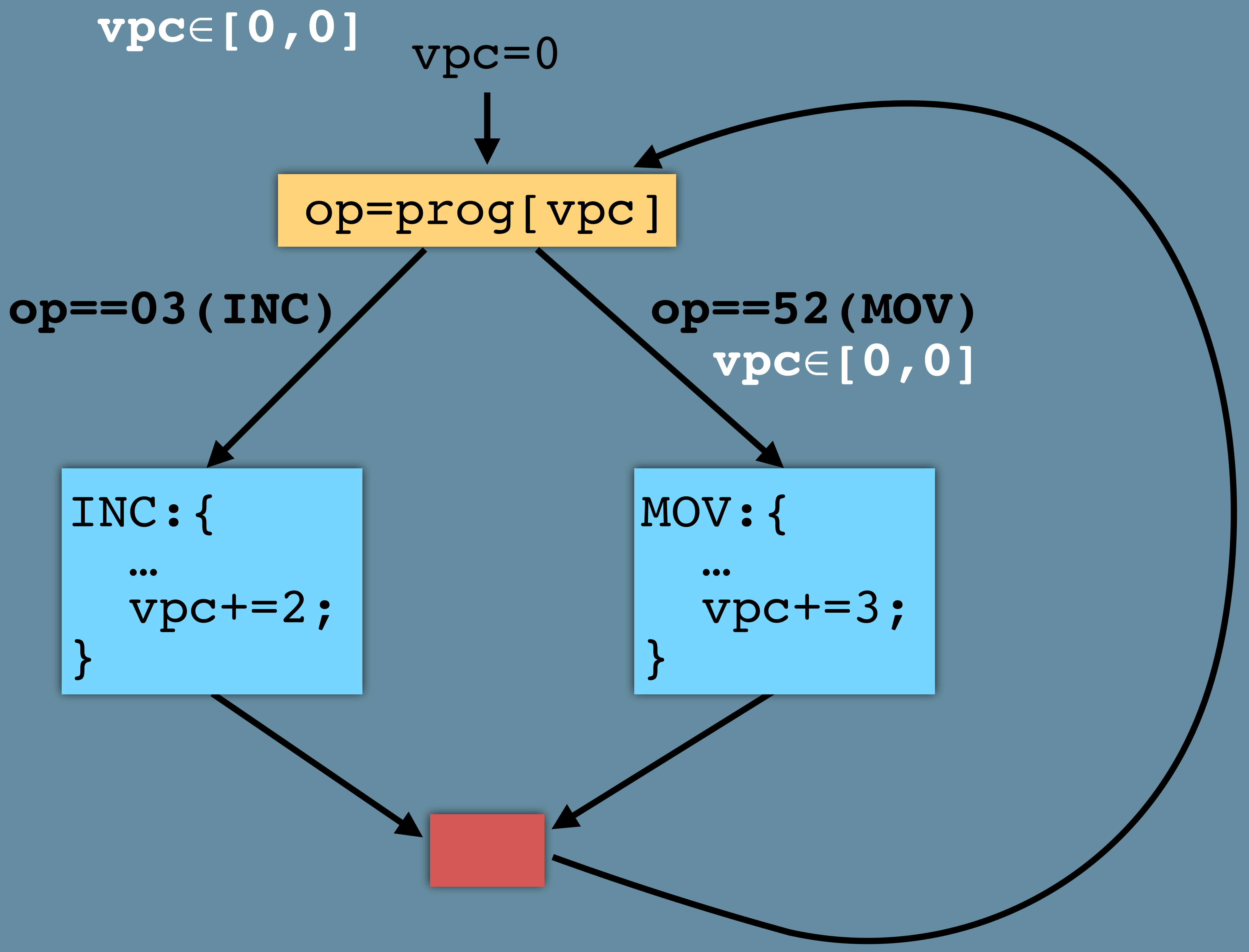
Abstract domain:
interval of VPC indices

vpc∈[0,0]



0	1	2	3	4	5	6	
52	01	02	03	01	03	01	…
MOV	**x**	**y**	**INC**	**x**	**INC**	**x**	

Abstract domain:
interval of VPC indices



vpc∈[0,0]

vpc=0

```
op=prog[vpc]
```

op==03(INC)

op==52(MOV)

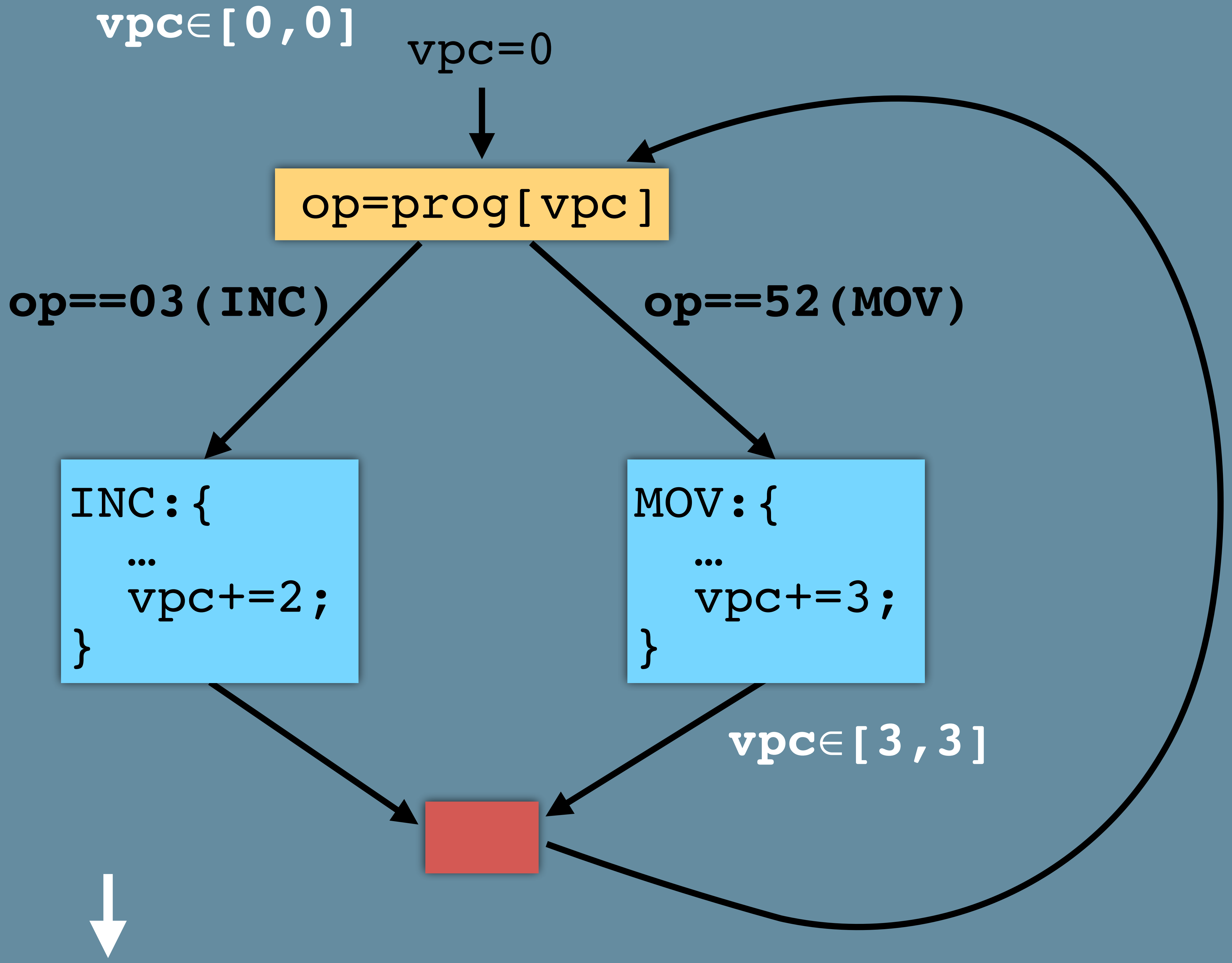
vpc∈[0,0]

```
INC:{
   …
   vpc+=2;
}
```

```
MOV:{
   …
   vpc+=3;
}
```

↓							
0	1	2	3	4	5	6	
52	01	02	03	01	03	01	…
MOV	**x**	**y**	**INC**	**x**	**INC**	**x**	

Abstract domain:
interval of VPC indices



vpc∈[0,0]

vpc=0

```
op=prog[vpc]
```

op==03(INC)

op==52(MOV)

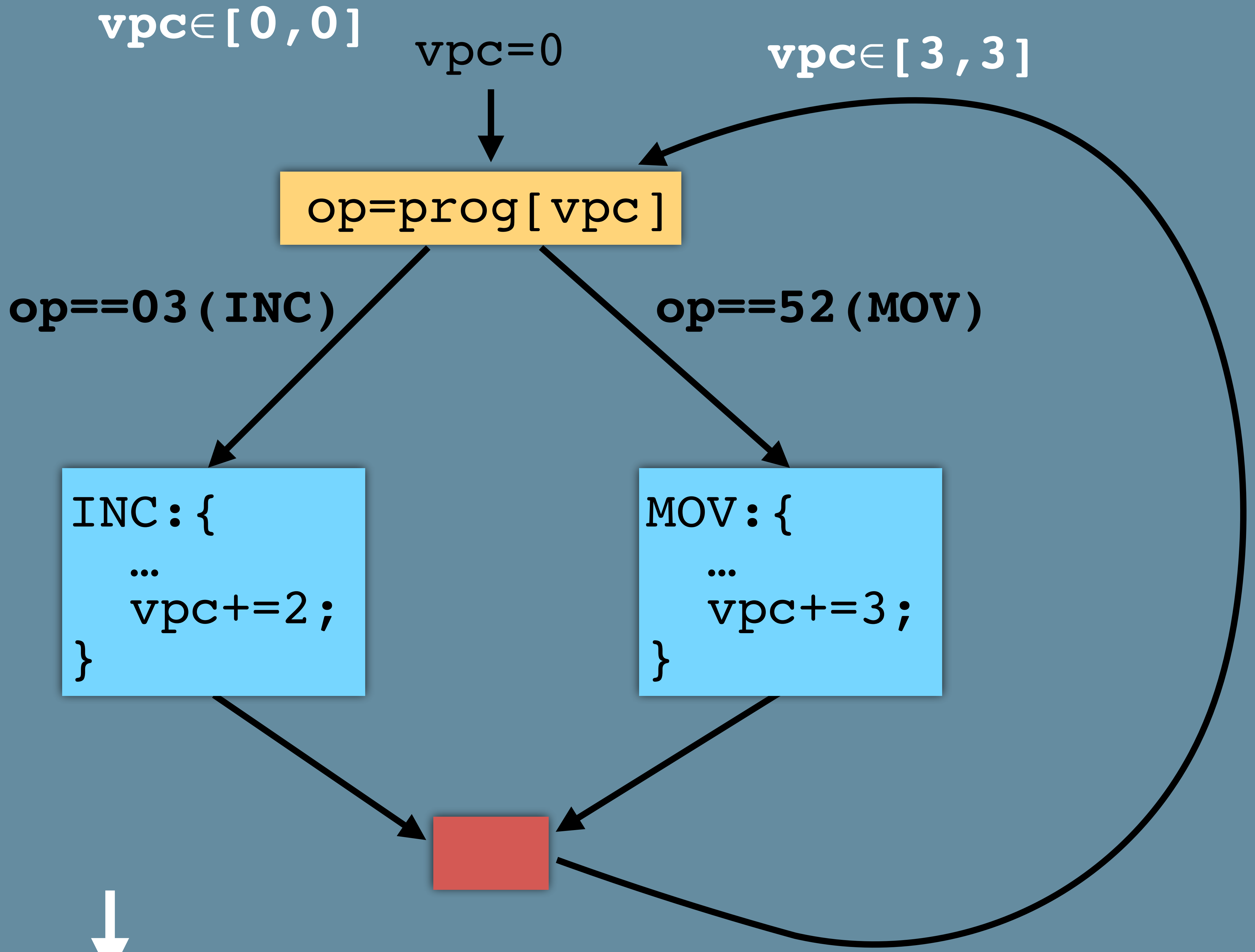
```
INC:{
   …
   vpc+=2;
}
```

```
MOV:{
   …
   vpc+=3;
}
```

vpc∈[3,3]

0	1	2	3	4	5	6	
52	01	02	03	01	03	01	…
MOV	x	y	INC	x	INC	x	

Abstract domain:
interval of VPC indices



vpc∈[0,0]

vpc=0

vpc∈[3,3]

```
op=prog[vpc]
```

op==03(INC)

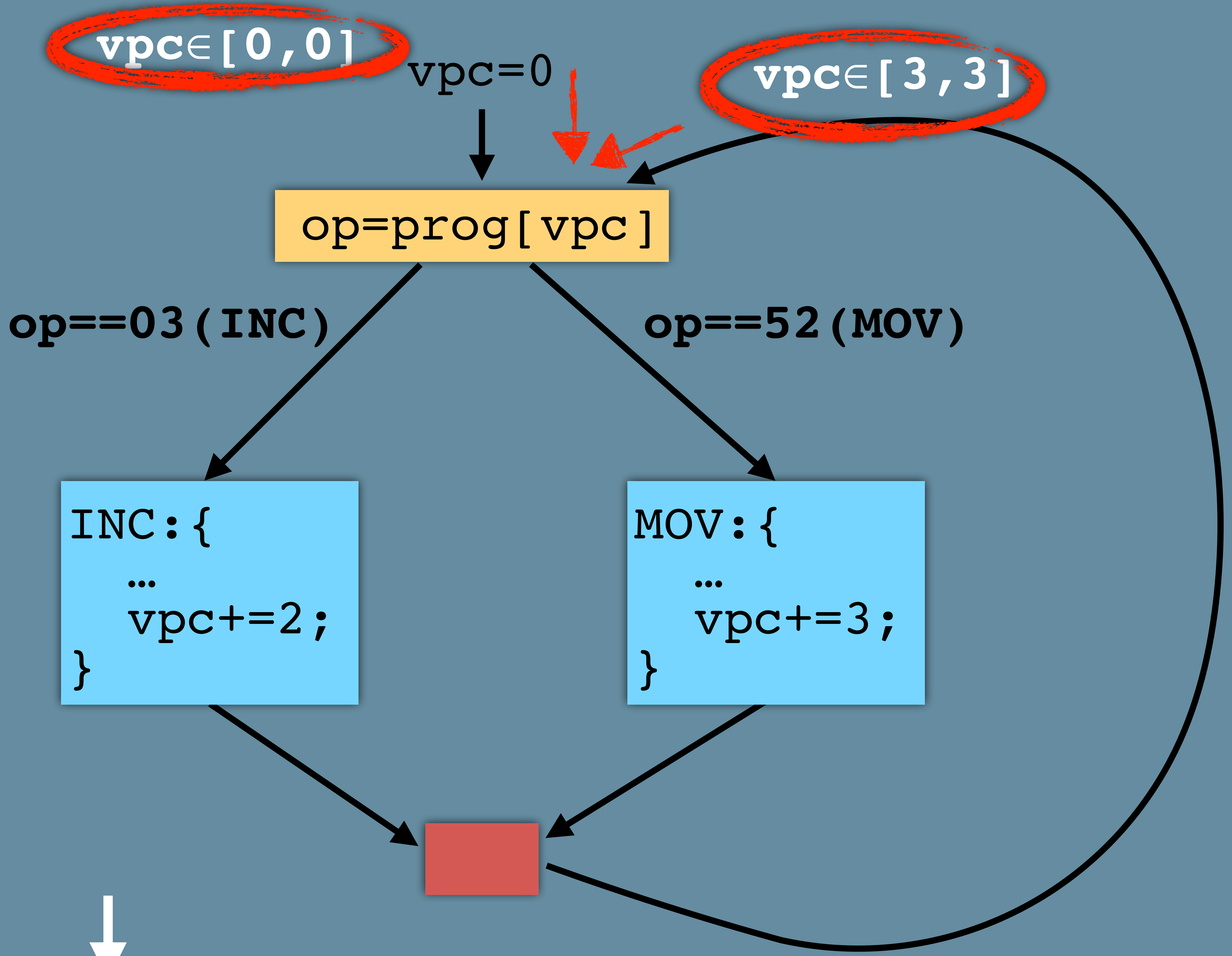
op==52(MOV)

```
INC:{
   …
   vpc+=2;
}
```

```
MOV:{
   …
   vpc+=3;
}
```

0	1	2	3	4	5	6	
52	01	02	03	01	03	01	…
MOV	**x**	**y**	**INC**	**x**	**INC**	**x**	

Abstract domain:
interval of VPC indices



vpc∈[0,0]

vpc=0

vpc∈[3,3]

```
op=prog[vpc]
```

op==03(INC)

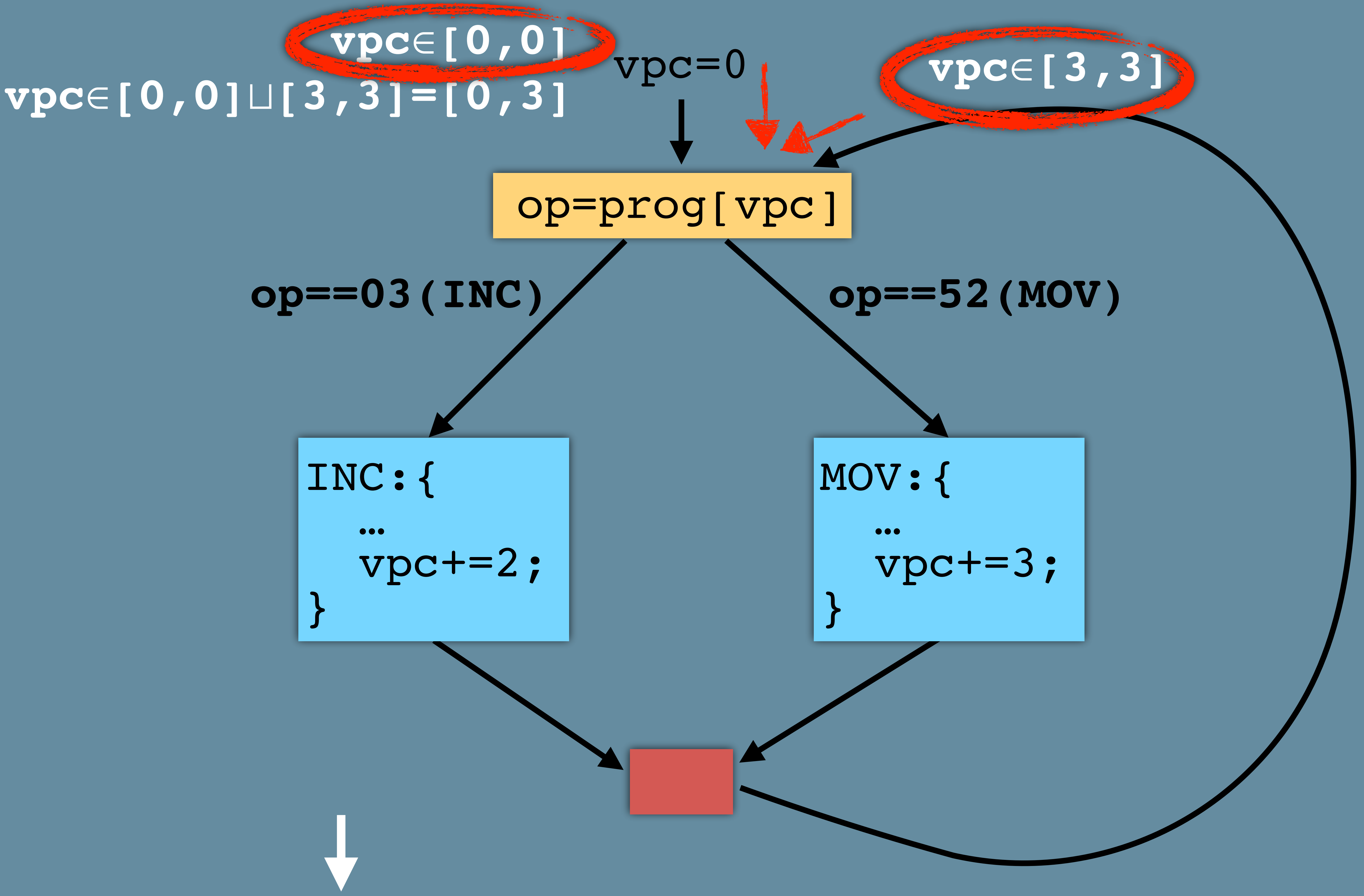
op==52(MOV)

```
INC:{
   …
   vpc+=2;
}
```

```
MOV:{
   …
   vpc+=3;
}
```

0	1	2	3	4	5	6	
52	01	02	03	01	03	01	…
MOV	x	y	INC	x	INC	x	

Abstract domain: interval of VPC indices



vpc∈[0,0]

vpc∈[0,0]⊔[3,3]=[0,3]

vpc=0

vpc∈[3,3]

```
op=prog[vpc]
```

op==03(INC)

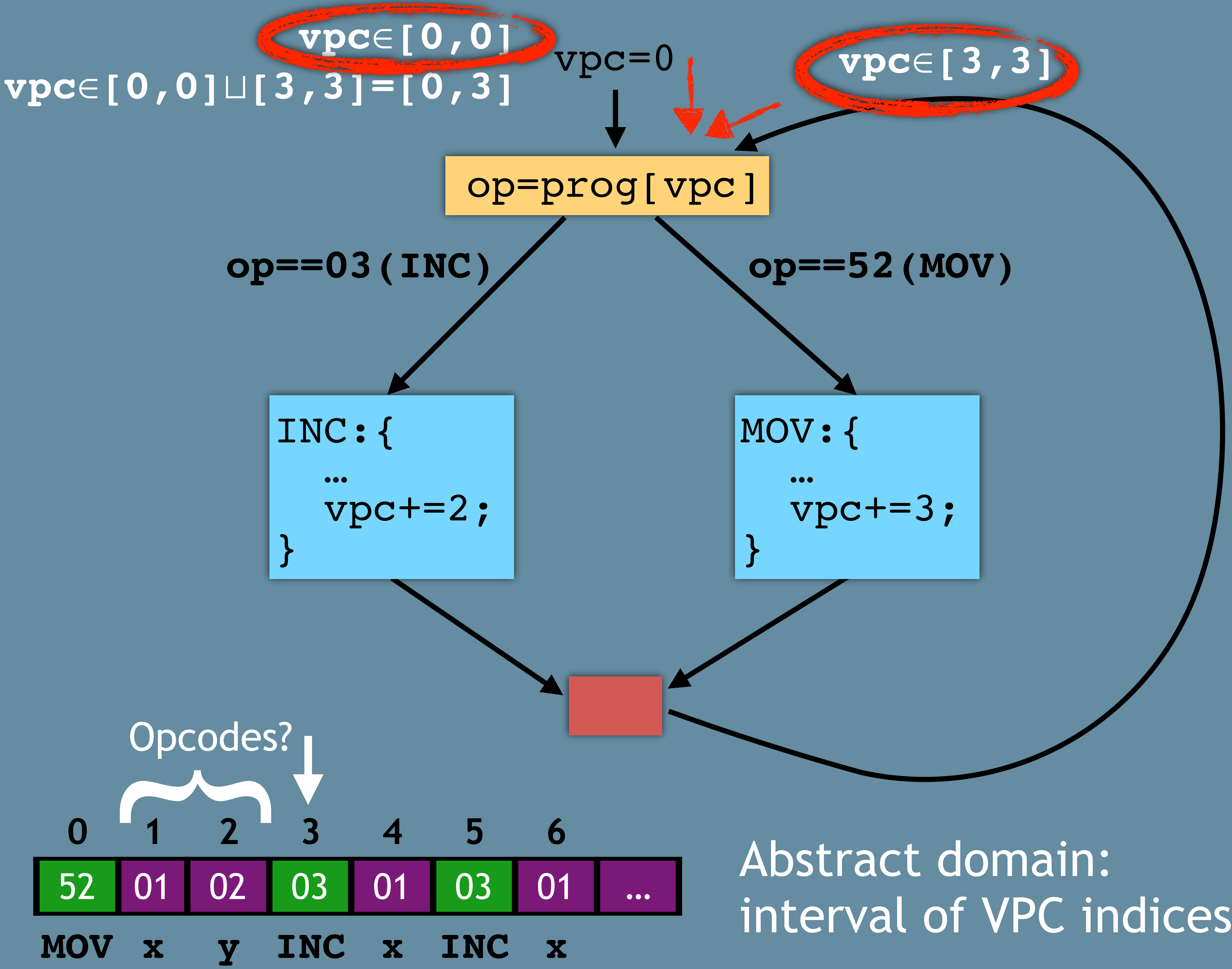
op==52(MOV)

```
INC:{
   …
   vpc+=2;
}
```

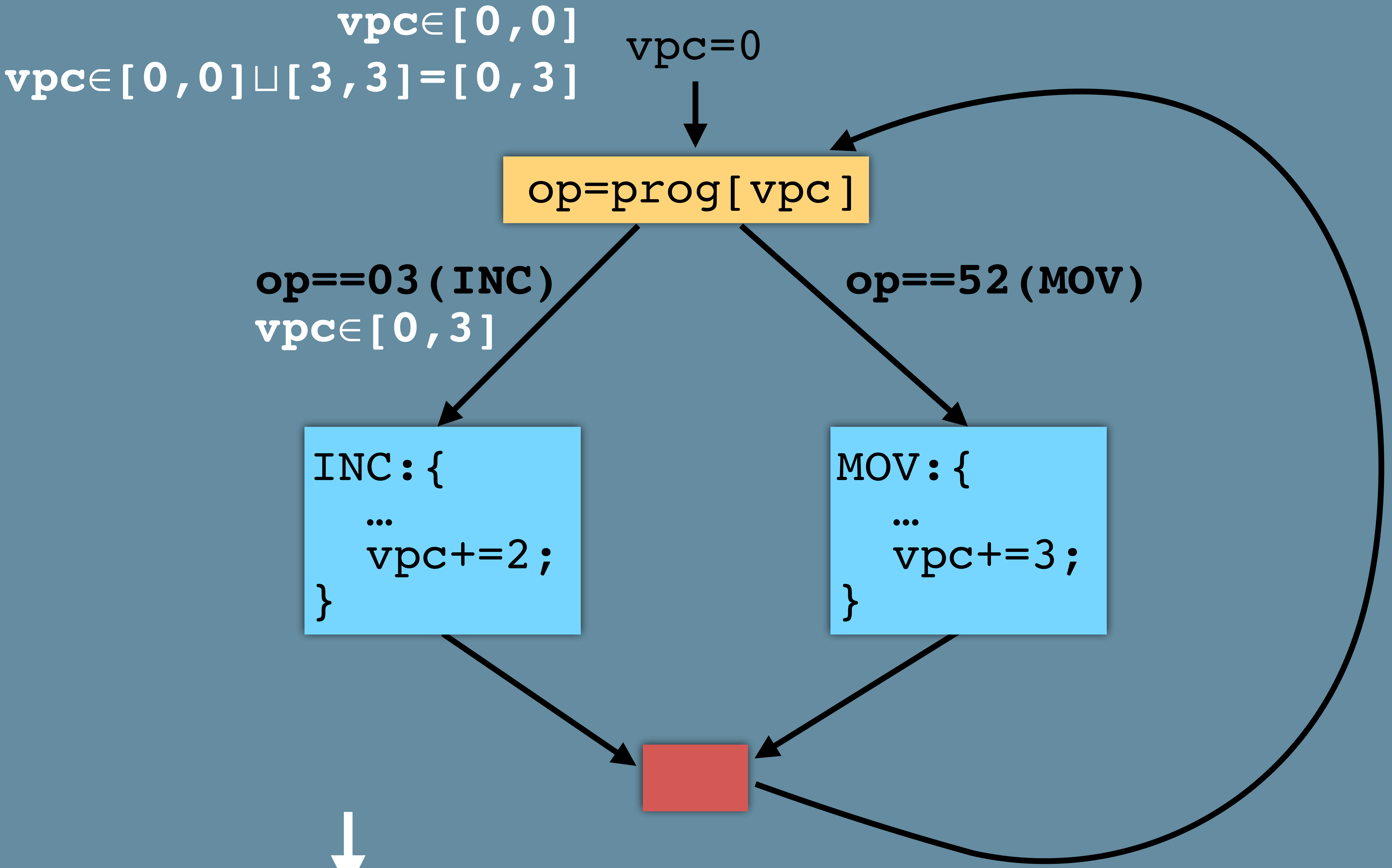
```
MOV:{
   …
   vpc+=3;
}
```

0	1	2	3	4	5	6	
52	01	02	03	01	03	01	…
MOV	x	y	INC	x	INC	x	

Abstract domain:
interval of VPC indices



Abstract domain:
interval of VPC indices



vpc∈[0,0]

vpc∈[0,0]⊔[3,3]=[0,3]

vpc=0

```
op=prog[vpc]
```

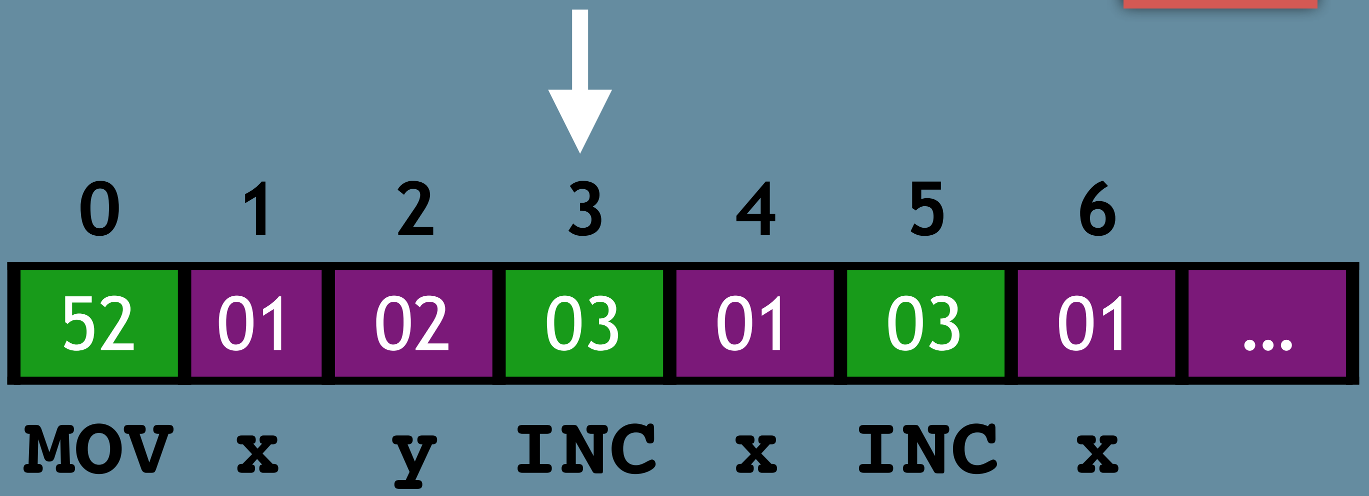
op==03(INC)

vpc∈[0,3]

op==52(MOV)

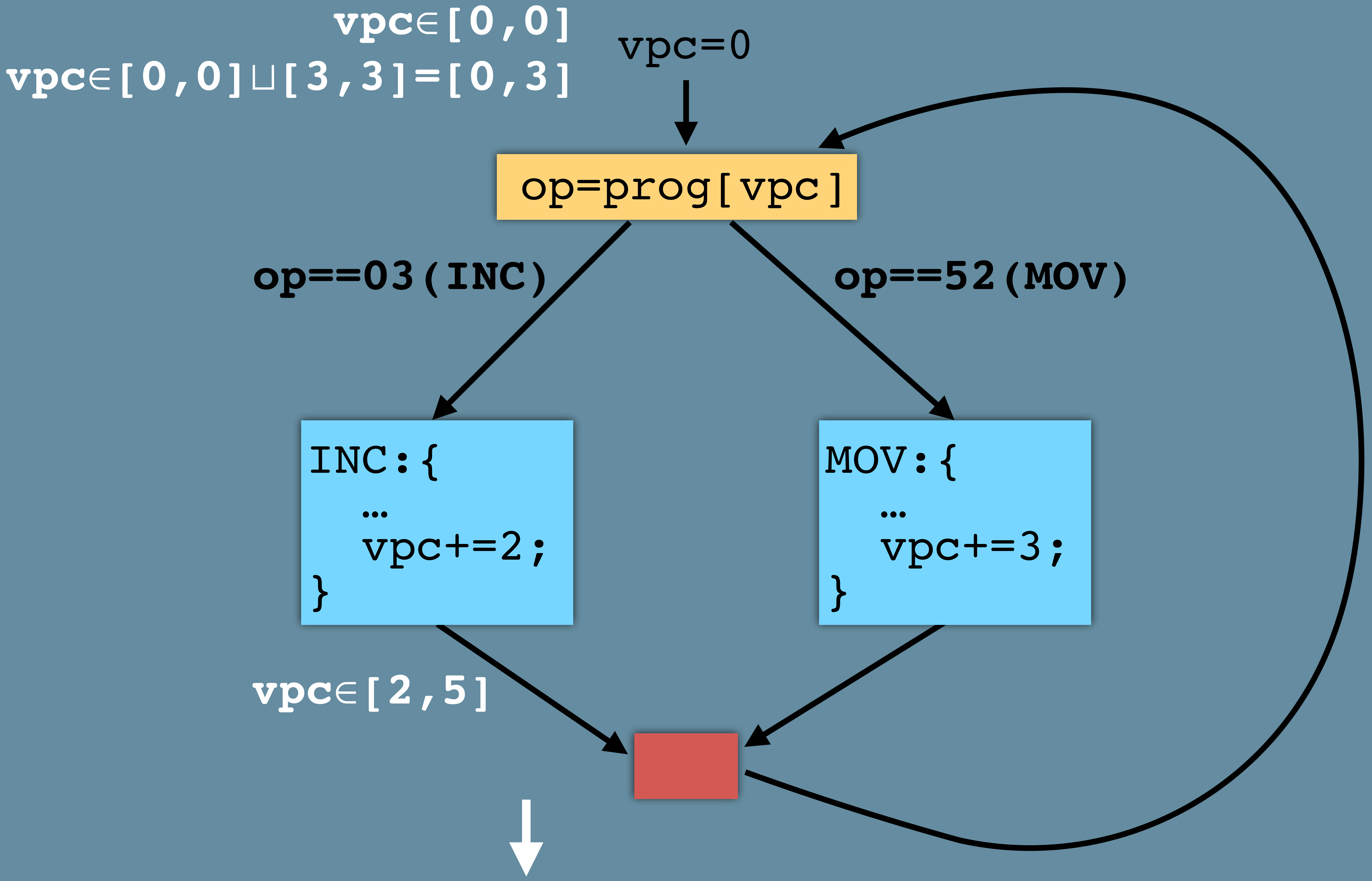
```
INC:{
   …
   vpc+=2;
}
```

```
MOV:{
   …
   vpc+=3;
}
```



0	1	2	3	4	5	6	
52	01	02	03	01	03	01	…
MOV	**x**	**y**	**INC**	**x**	**INC**	**x**	

Abstract domain:
interval of VPC indices



vpc∈[0,0]

vpc∈[0,0]⊔[3,3]=[0,3]

vpc=0

```
op=prog[vpc]
```

op==03(INC)

op==52(MOV)

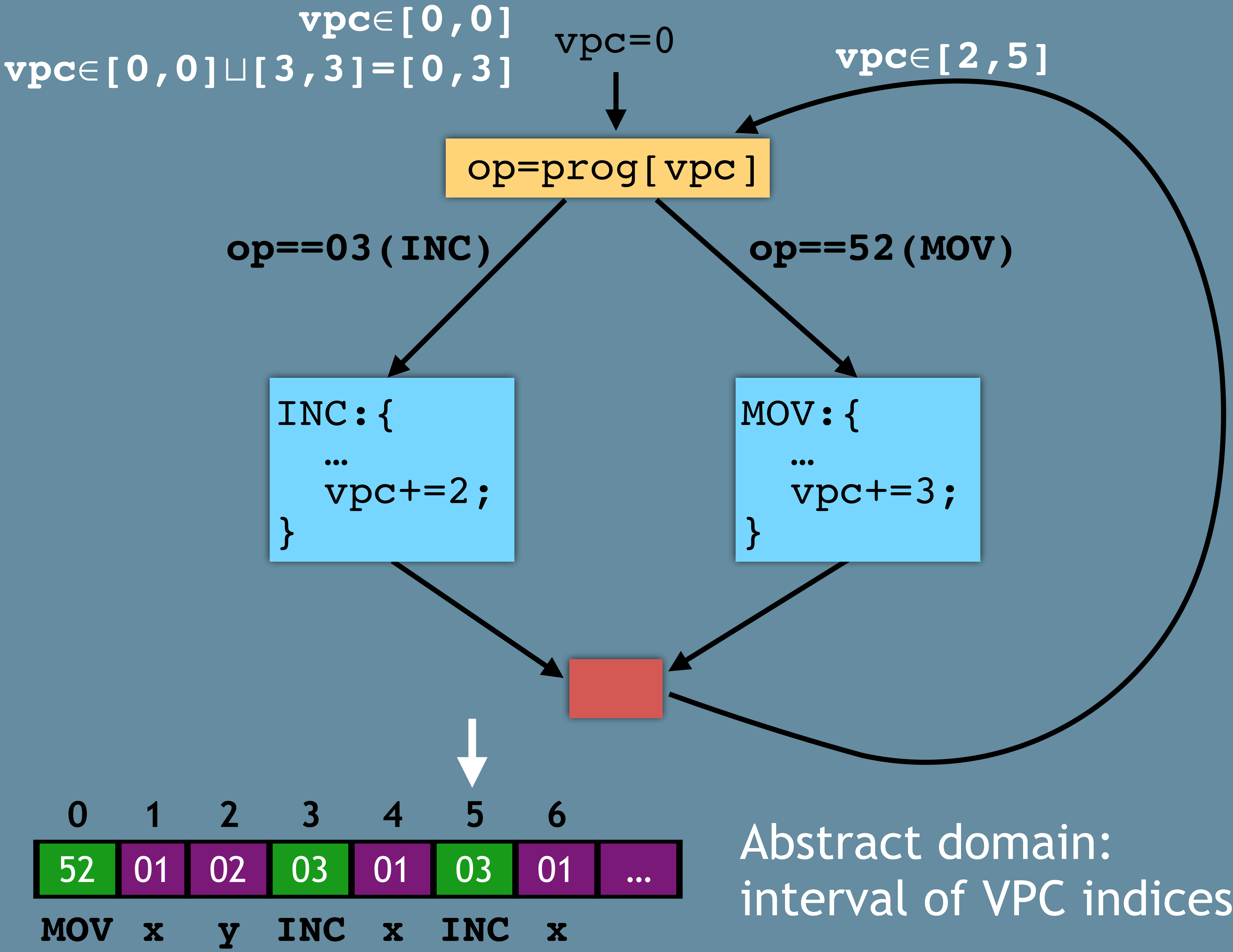
```
INC:{
   …
   vpc+=2;
}
```

```
MOV:{
   …
   vpc+=3;
}
```

vpc∈[2,5]

0	1	2	3	4	5	6	
52	01	02	03	01	03	01	…
MOV	x	y	INC	x	INC	x	

Abstract domain:
interval of VPC indices



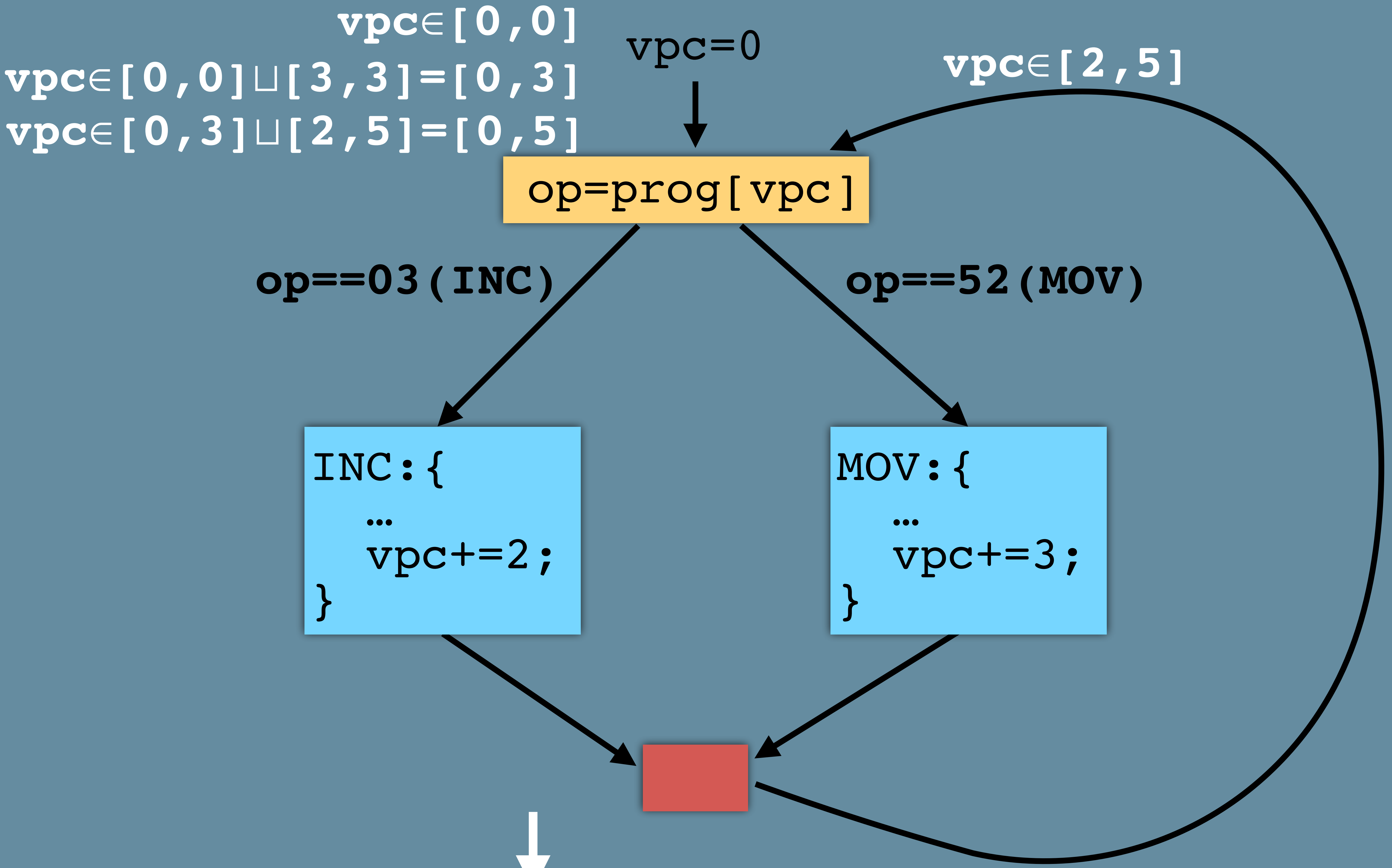
```
op=prog[vpc]
```

```
INC:{
   …
   vpc+=2;
}
```

```
MOV:{
   …
   vpc+=3;
}
```

0	1	2	3	4	5	6	
52	01	02	03	01	03	01	…
MOV	x	y	INC	x	INC	x	

Abstract domain:
interval of VPC indices



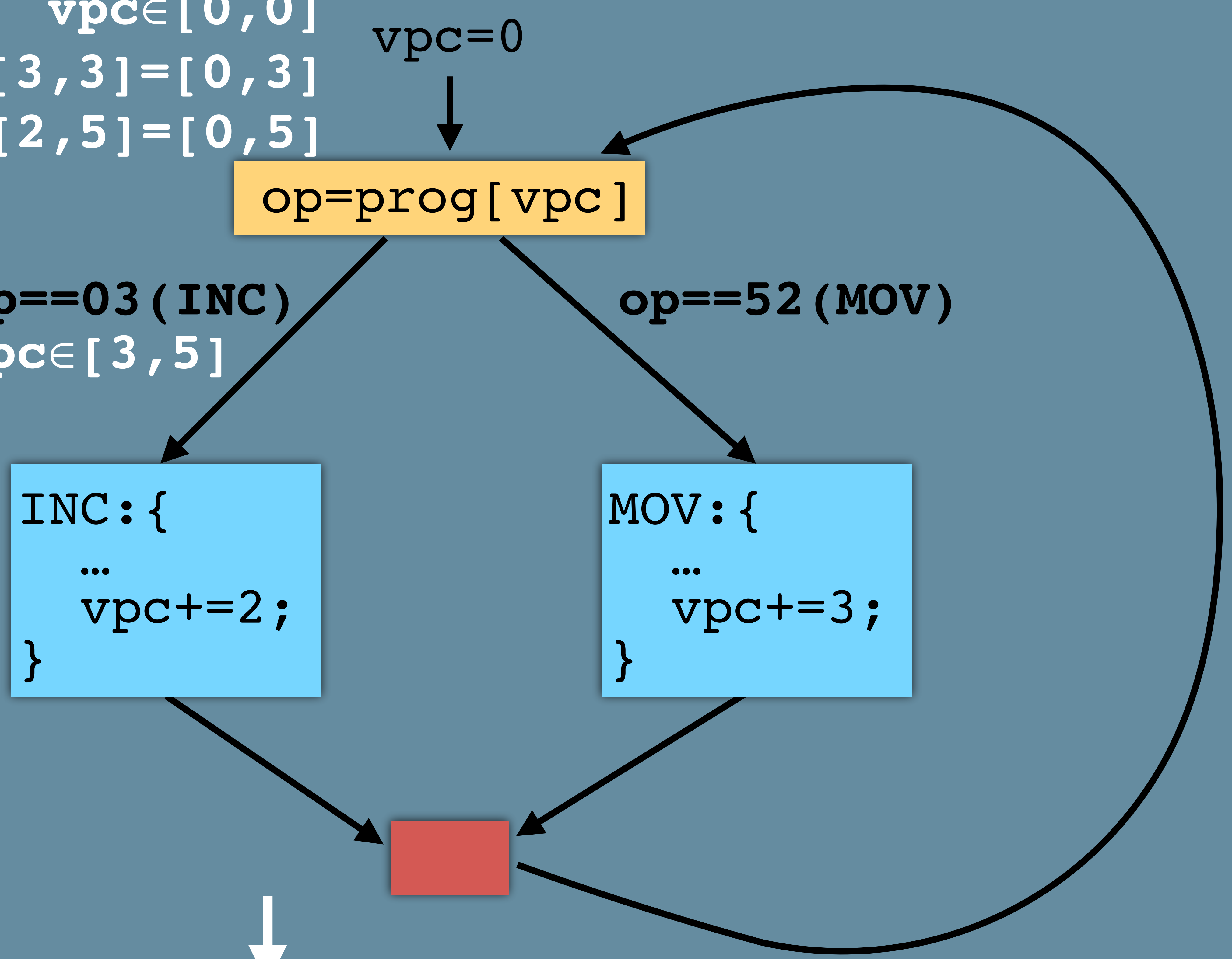
0	1	2	3	4	5	6	
52	01	02	03	01	03	01	…
MOV	x	y	INC	x	INC	x	

Abstract domain:
interval of VPC indices

vpc∈[0,0]

vpc∈[0,0]⊔[3,3]=[0,3]

vpc∈[0,3]⊔[2,5]=[0,5]



```
INC:{
   …
   vpc+=2;
}
```

```
MOV:{
   …
   vpc+=3;
}
```

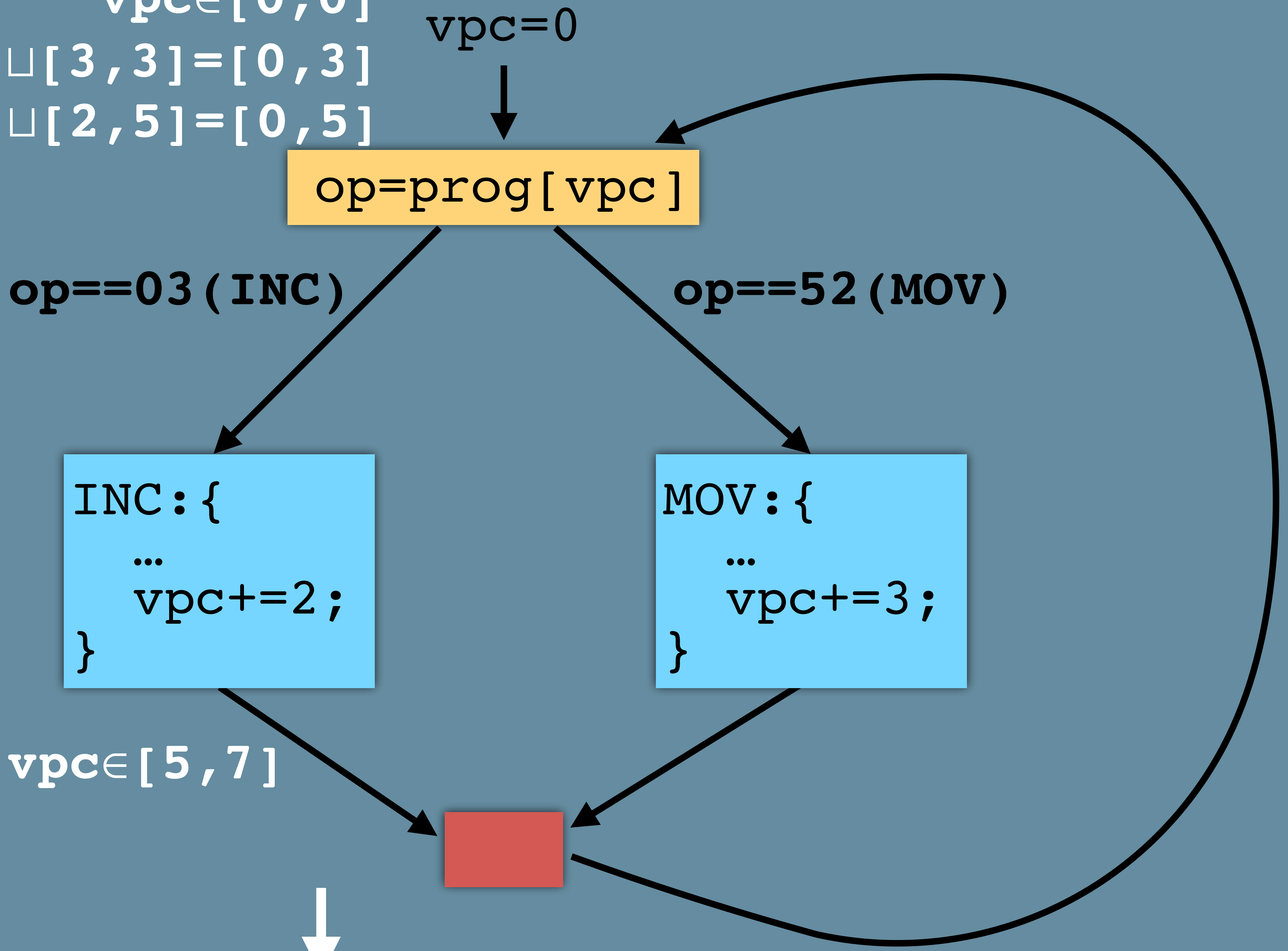
0	1	2	3	4	5	6	
52	01	02	03	01	03	01	…
MOV	**x**	**y**	**INC**	**x**	**INC**	**x**	

Abstract domain:
interval of VPC indices

vpc∈[0,0]

vpc∈[0,0]⊔[3,3]=[0,3]

vpc∈[0,3]⊔[2,5]=[0,5]



vpc=0

```
op=prog[vpc]
```

op==03(INC)

op==52(MOV)

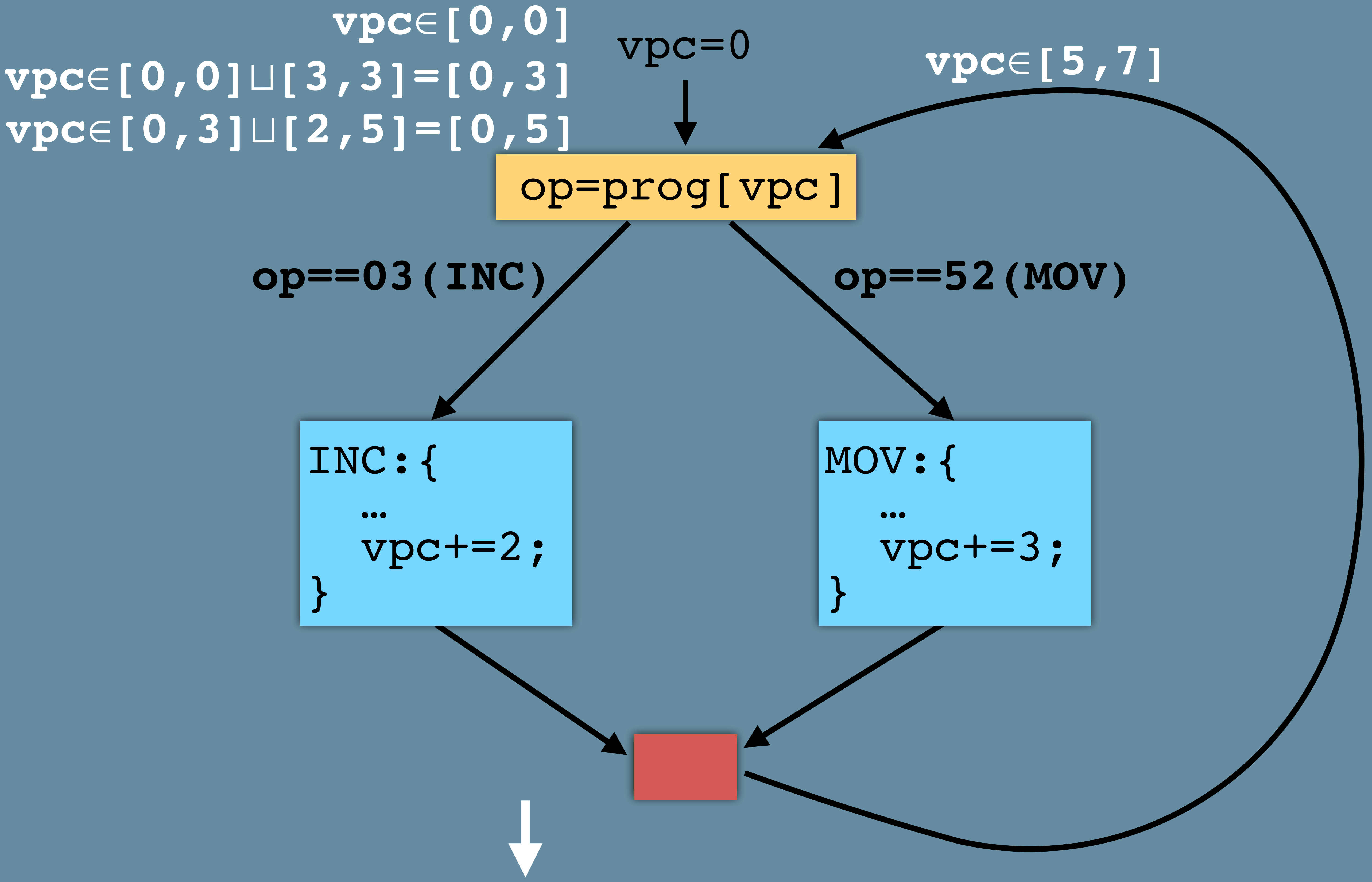
```
INC:{
   …
   vpc+=2;
}
```

```
MOV:{
   …
   vpc+=3;
}
```

vpc∈[5,7]

0	1	2	3	4	5	6	
52	01	02	03	01	03	01	…
MOV	**x**	**y**	**INC**	**x**	**INC**	**x**	

Abstract domain:
interval of VPC indices



vpc∈[0,0]

vpc∈[0,0]⊔[3,3]=[0,3]

vpc∈[0,3]⊔[2,5]=[0,5]

vpc=0

vpc∈[5,7]

```
op=prog[vpc]
```

op==03(INC)

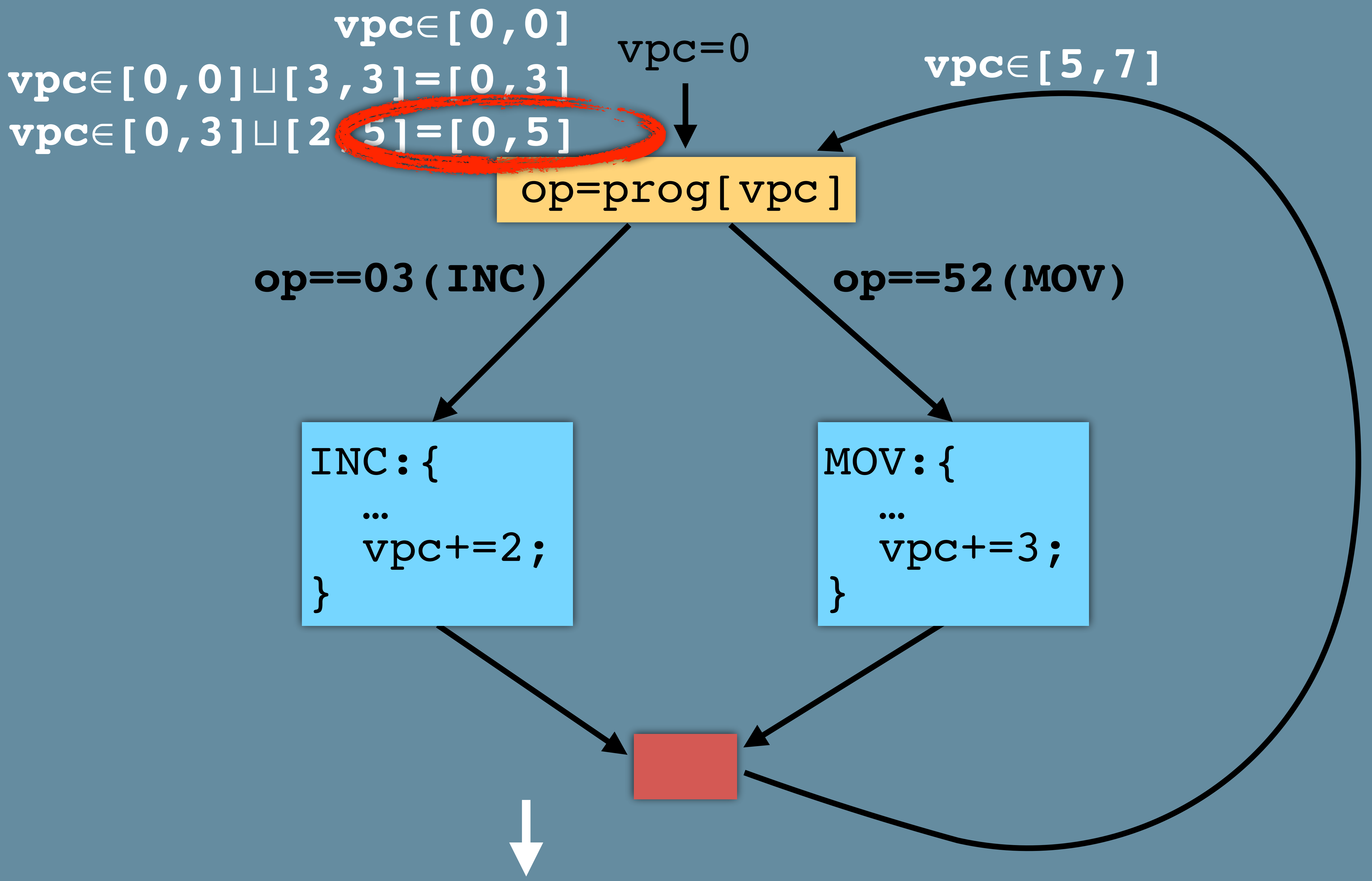
op==52(MOV)

```
INC:{
   …
   vpc+=2;
}
```

```
MOV:{
   …
   vpc+=3;
}
```

0	1	2	3	4	5	6	
52	01	02	03	01	03	01	…
MOV	x	y	INC	x	INC	x	

Abstract domain:
interval of VPC indices



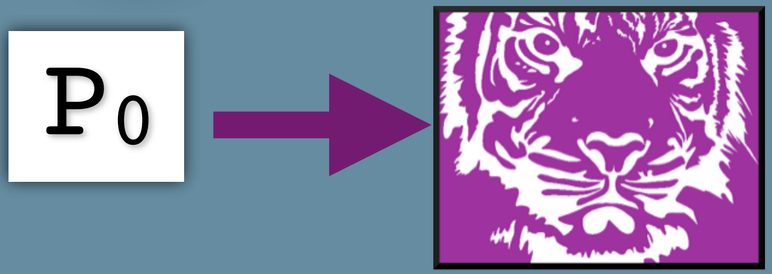
```
INC:{
   …
   vpc+=2;
}
```

```
MOV:{
   …
   vpc+=3;
}
```

0	1	2	3	4	5	6	
52	01	02	03	01	03	01	…
MOV	x	y	INC	x	INC	x	

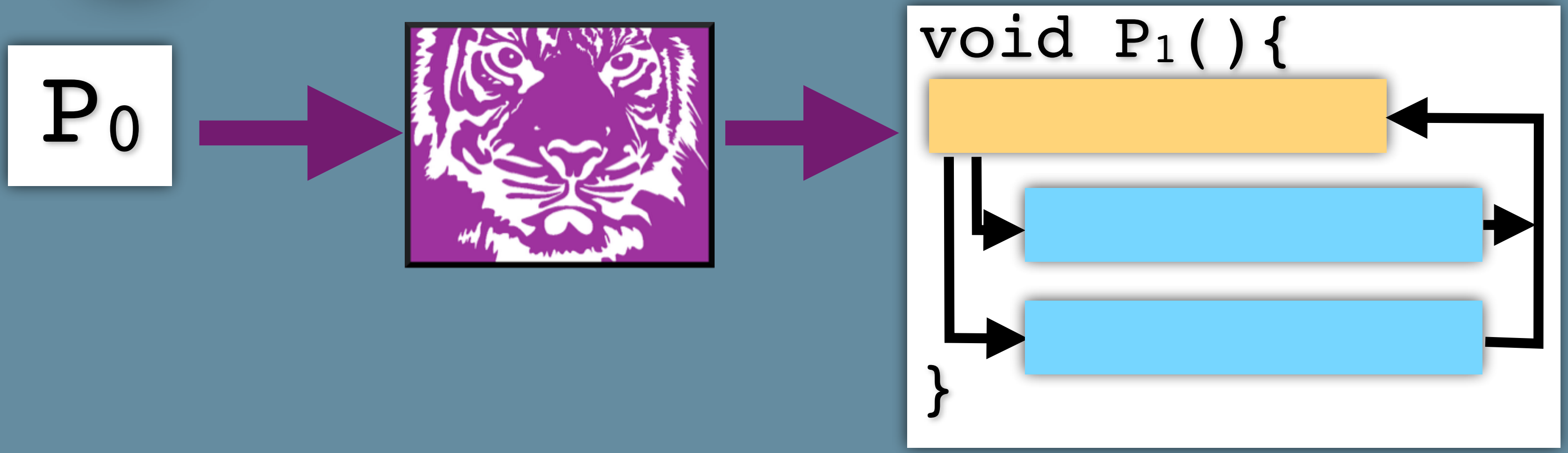
Abstract domain:
interval of VPC indices

Virtualize+JIT



P_0

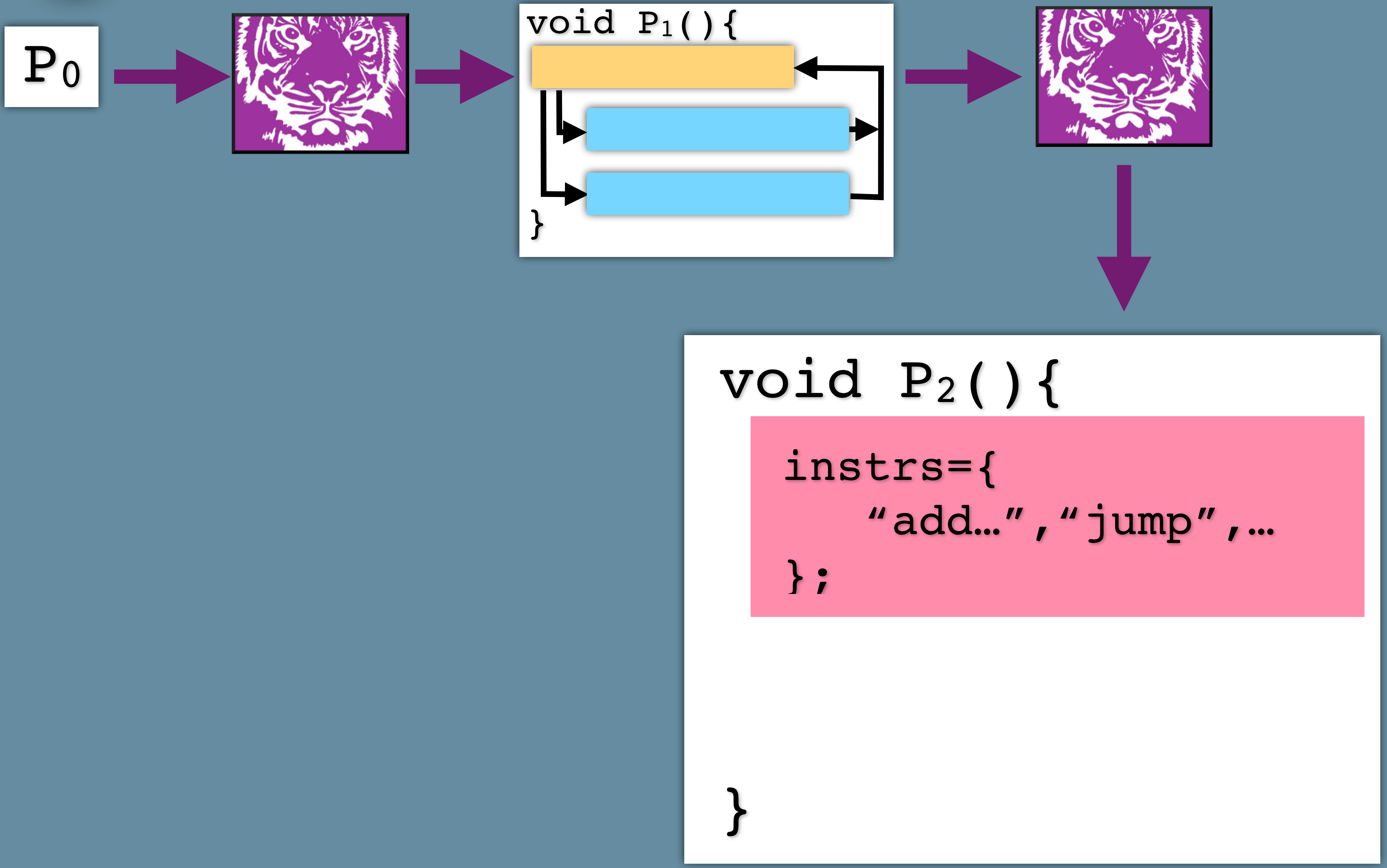
Virtualize+JIT



P_0

```
void P1(){
}
```

Virtualize+JIT



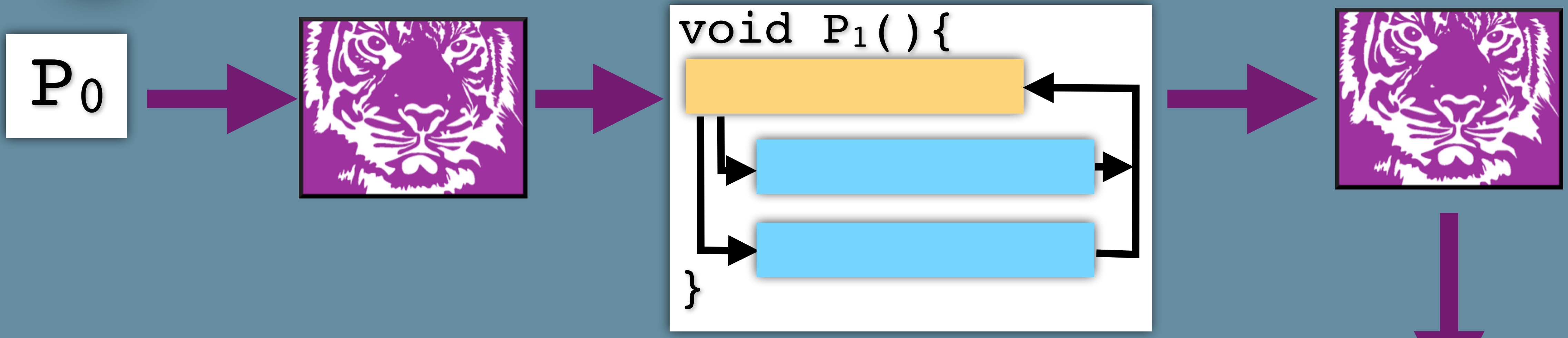
```
void P1(){
}
```

```
void P2(){
  instrs={
     "add…","jump",…
  };



}
```

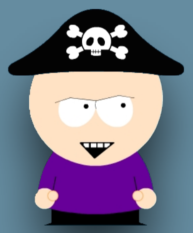
Virtualize+JIT



```
void P2(){
  instrs={
     "add…","jump",…
  };
  code=compile(instrs);
  goto *code;
}
```

x86

```
add    %cl,(%rax,%rax,1)
imul   %ecx,%ebx
ja     0x4242
```



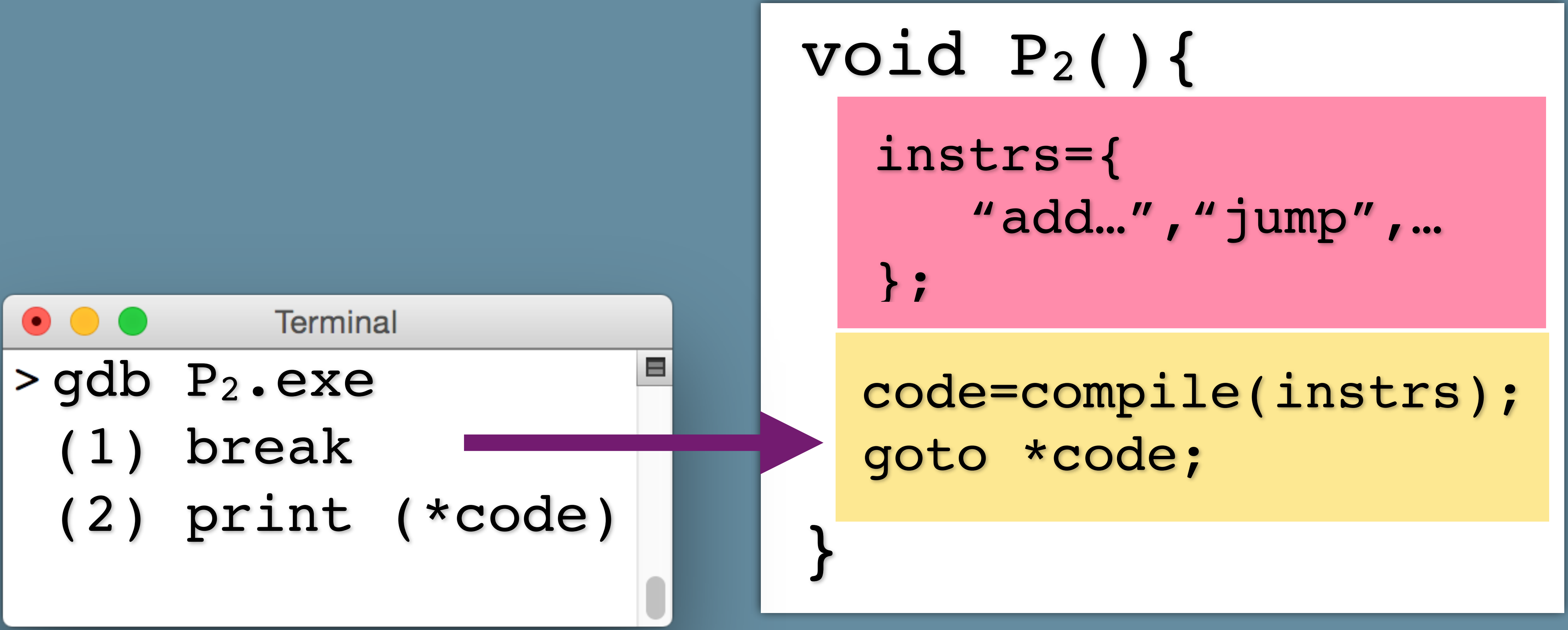
Unpack+Print

1. Find the point where the code exists in cleartext
2. Print it
3. Statically analyze the cleartext code

```
void P2(){
  instrs={
     "add…","jump",…
  };
  code=compile(instrs);
  goto *code;
}
```

Unpack+Print

1. Find the point where the code exists in cleartext
2. Print it
3. Statically analyze the cleartext code

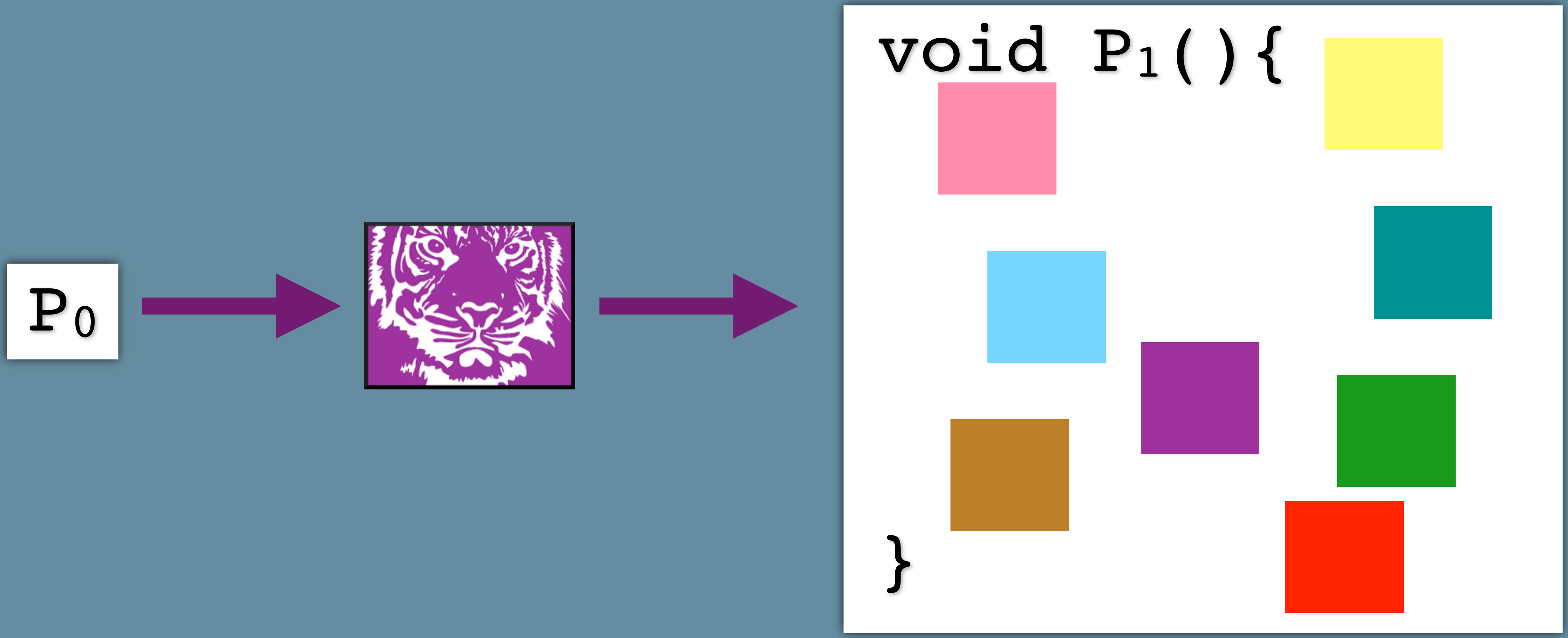


```
>gdb P2.exe
 (1) break
 (2) print (*code)
```

```
void P2(){
  instrs={
      "add…","jump",…
  };
  code=compile(instrs);
  goto *code;
}
```

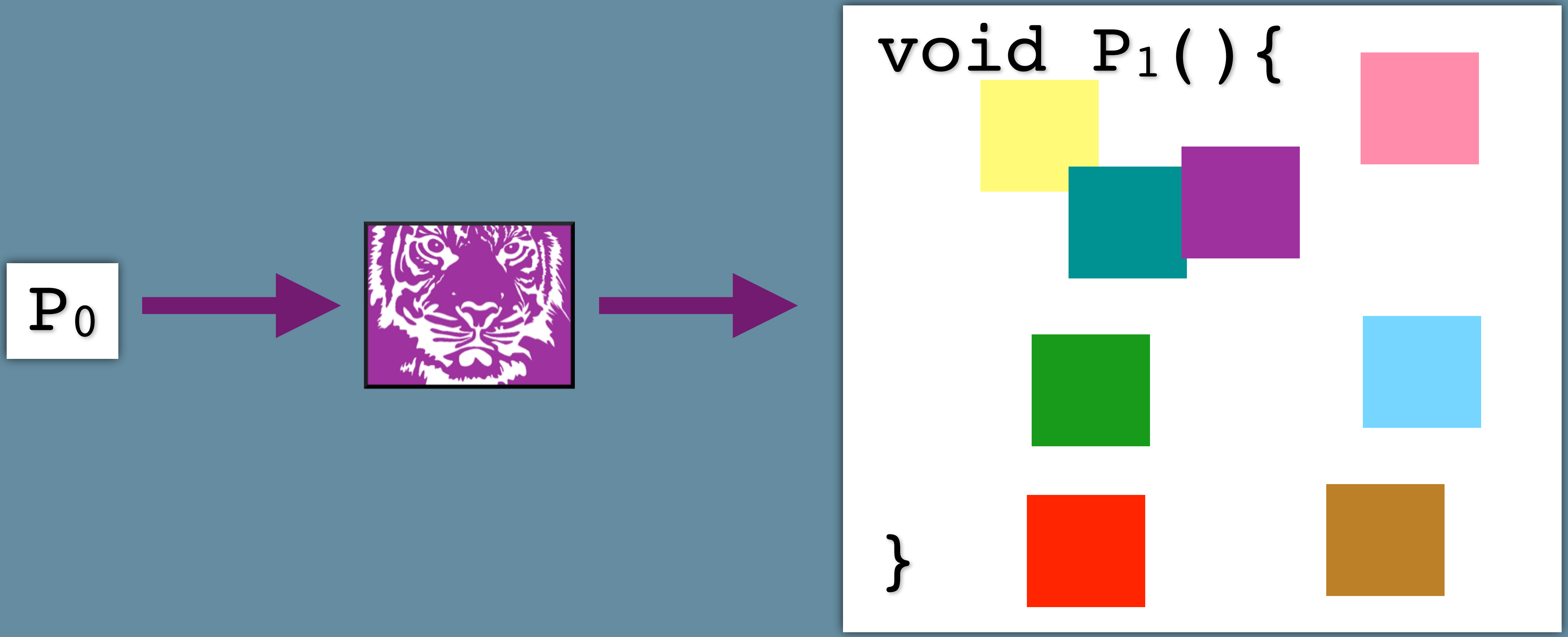
Dynamic Obfuscation

- Keep the code in constant flux at runtime
- At no point should the entire code exist in cleartext



Dynamic Obfuscation

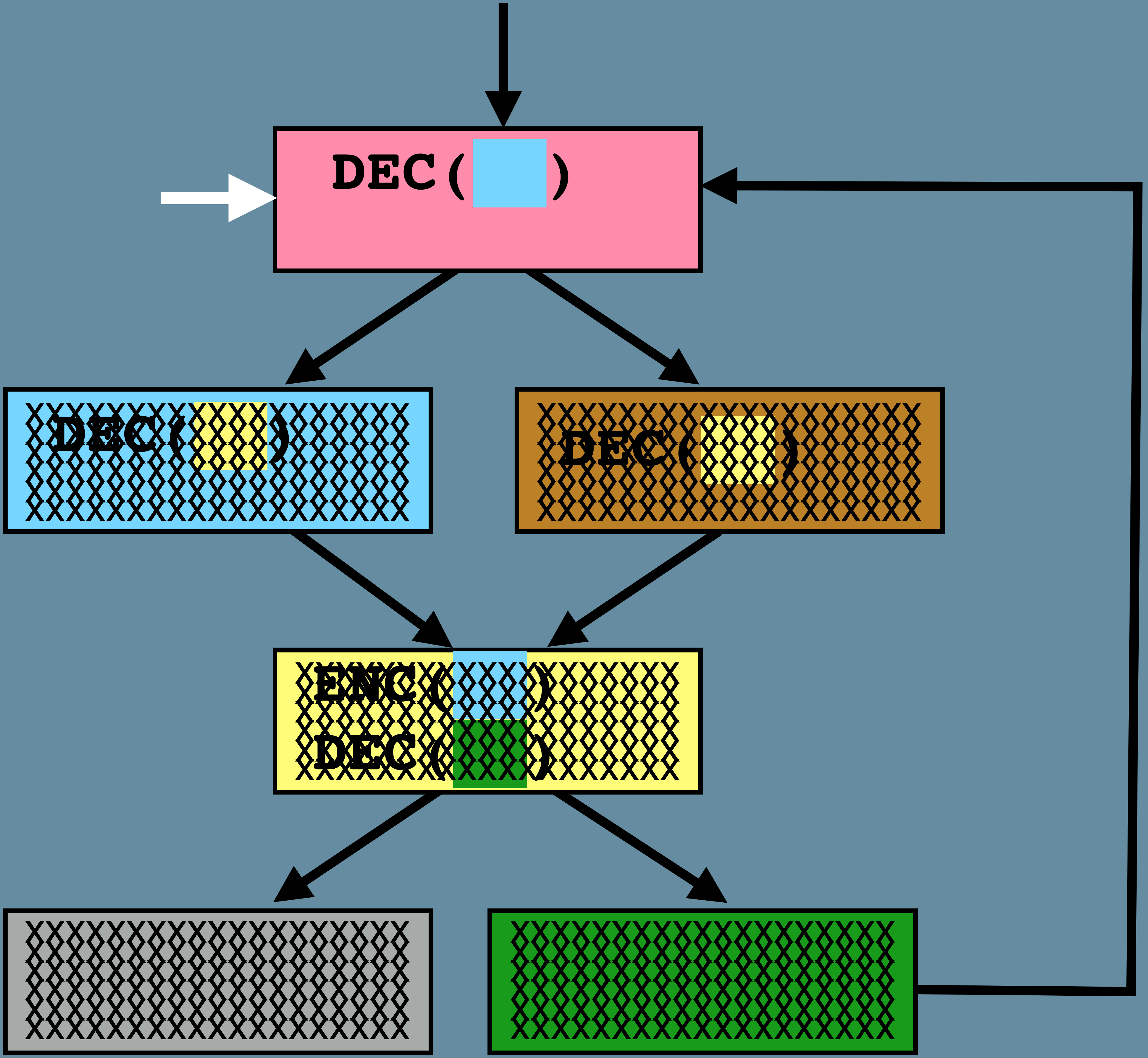
- Keep the code in constant flux at runtime
- At no point should the entire code exist in cleartext

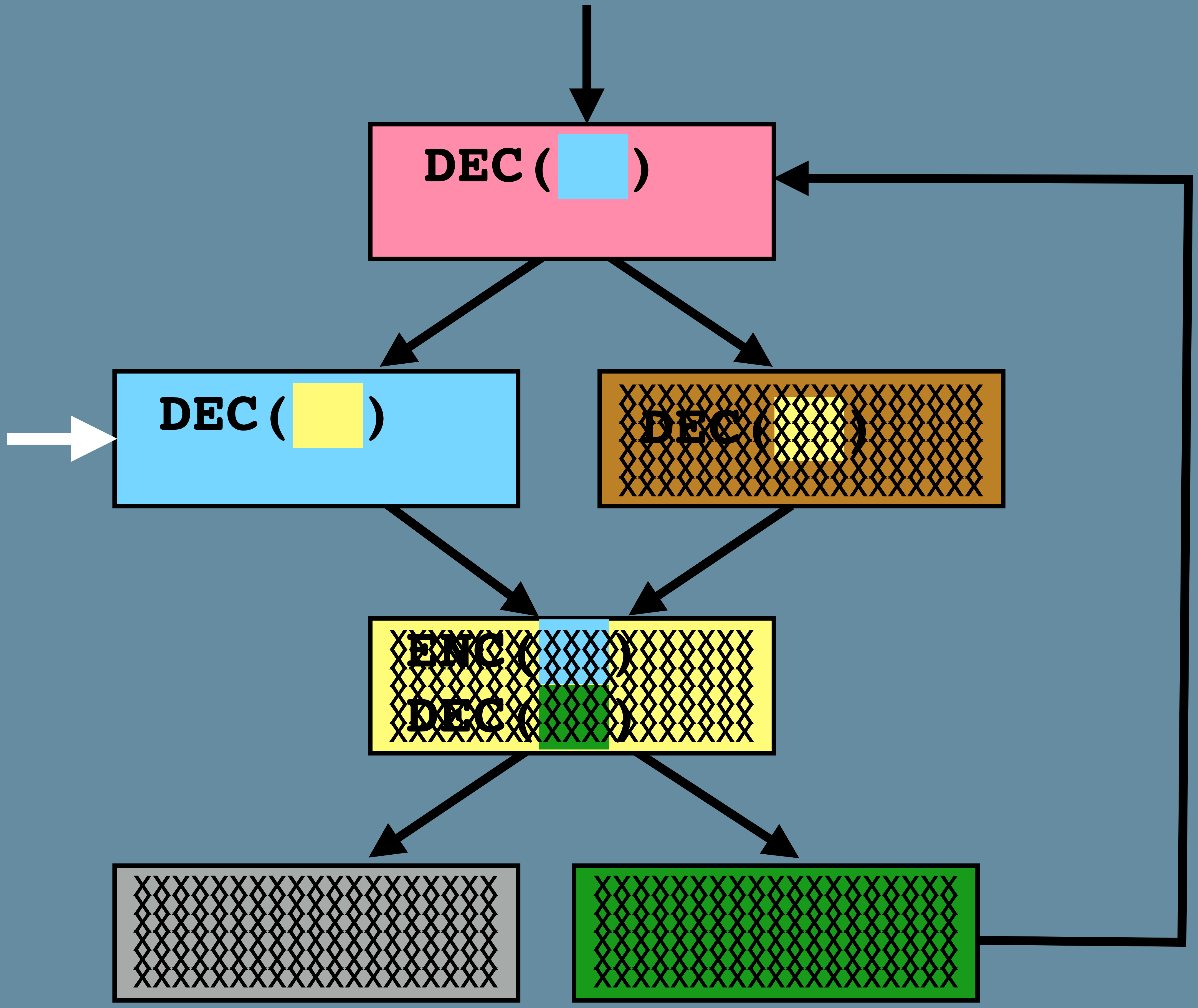


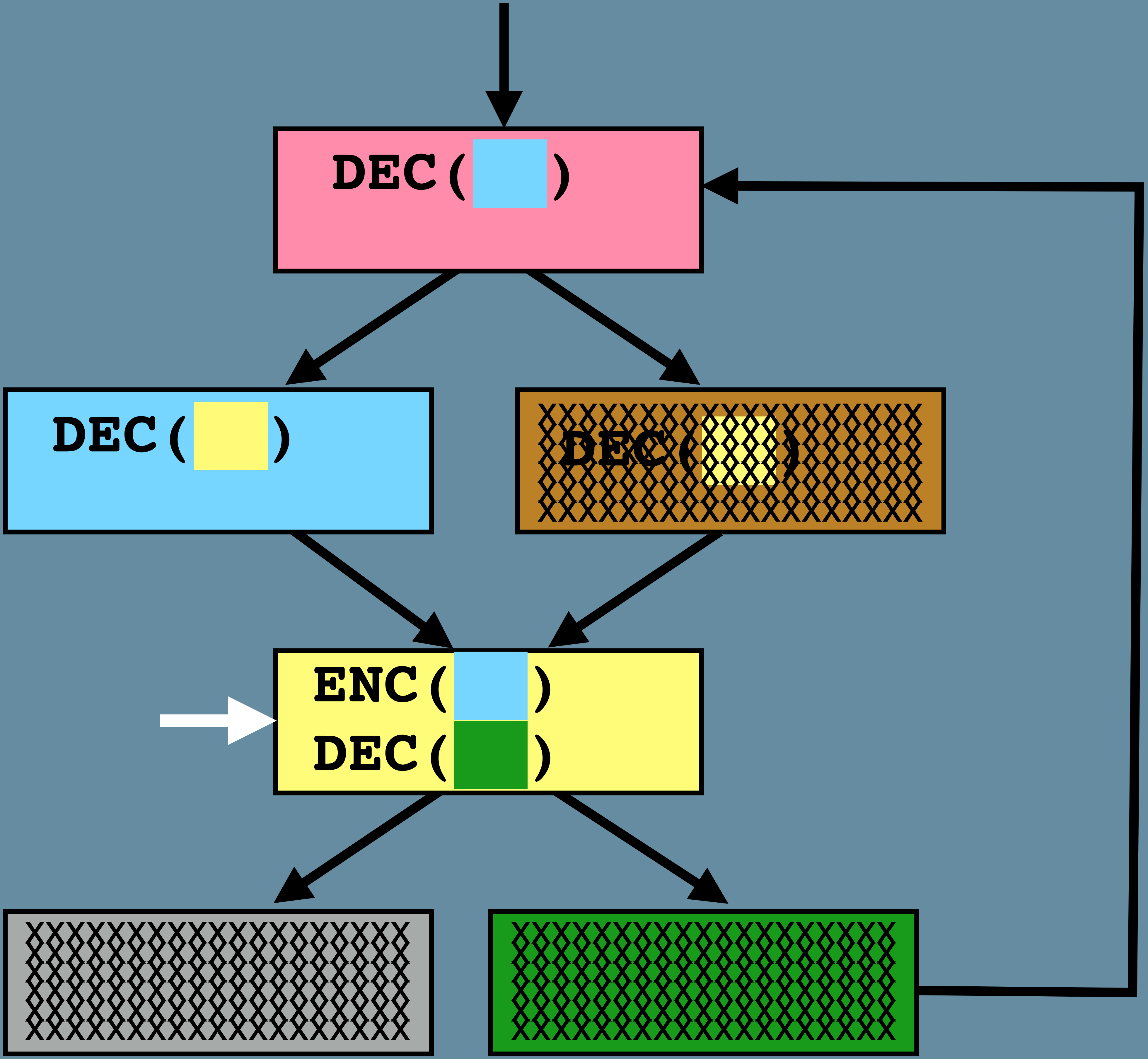
P_0

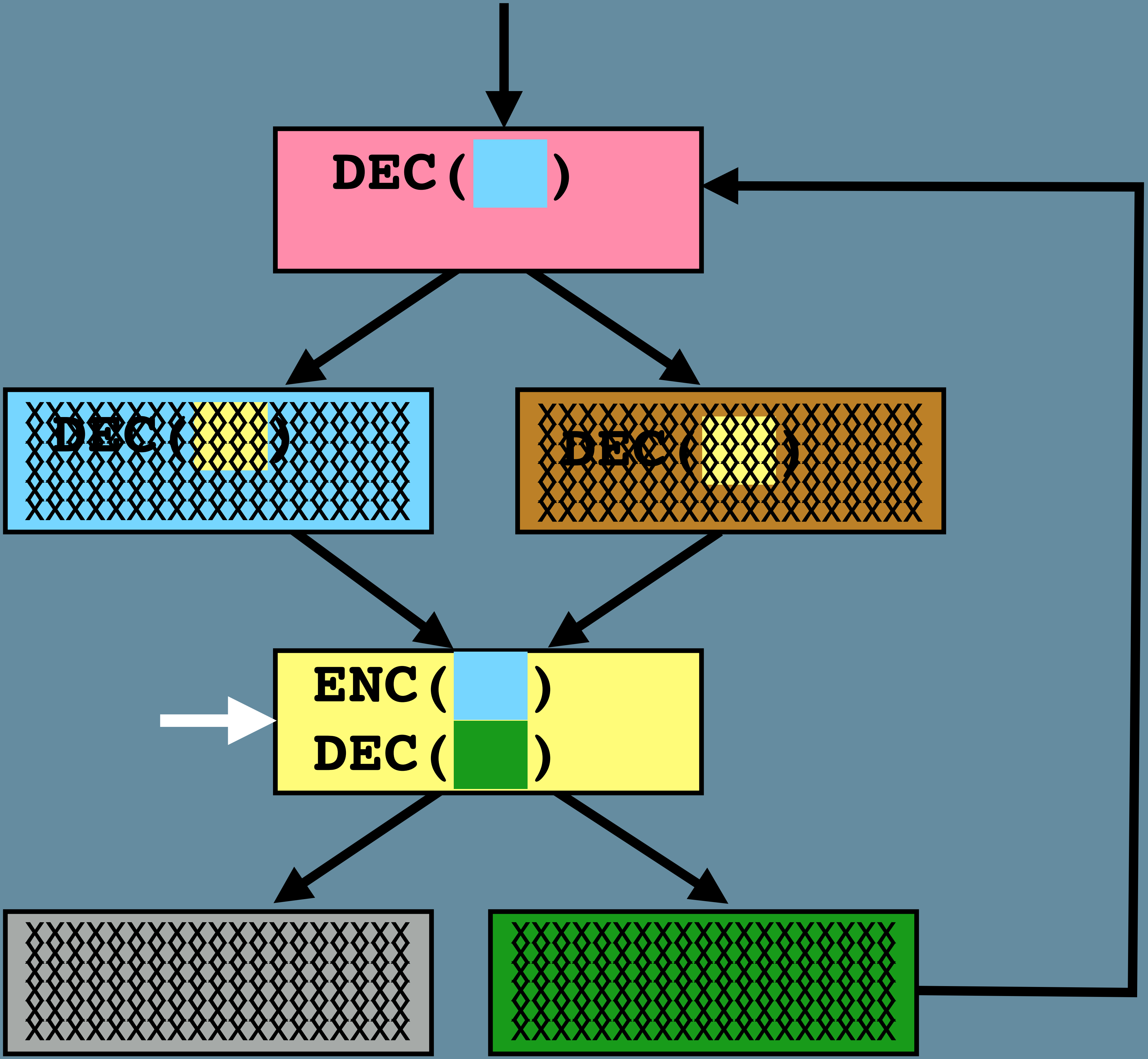
void P_1(){

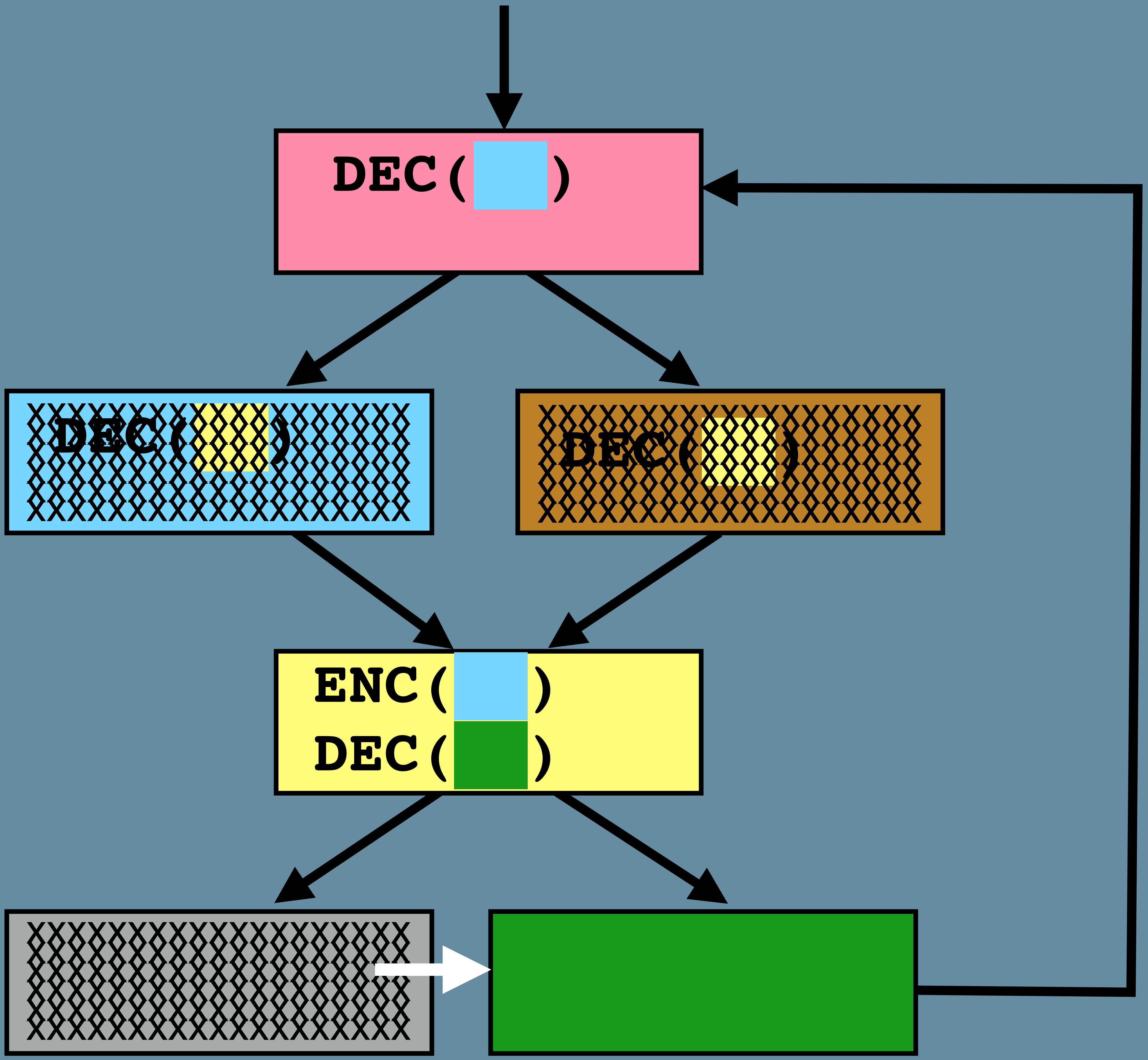
}


DEC( )
DEC( )
DEC( )
ENC( )
DEC( )


DEC ( )
DEC ( )
DEC ( )
ENC ( )
DEC ( )


DEC ( )
DEC ( )
DEC ( )
ENC ( )
DEC ( )


DEC ( )
DEC ( )
DEC ( )
ENC ( )
DEC ( )

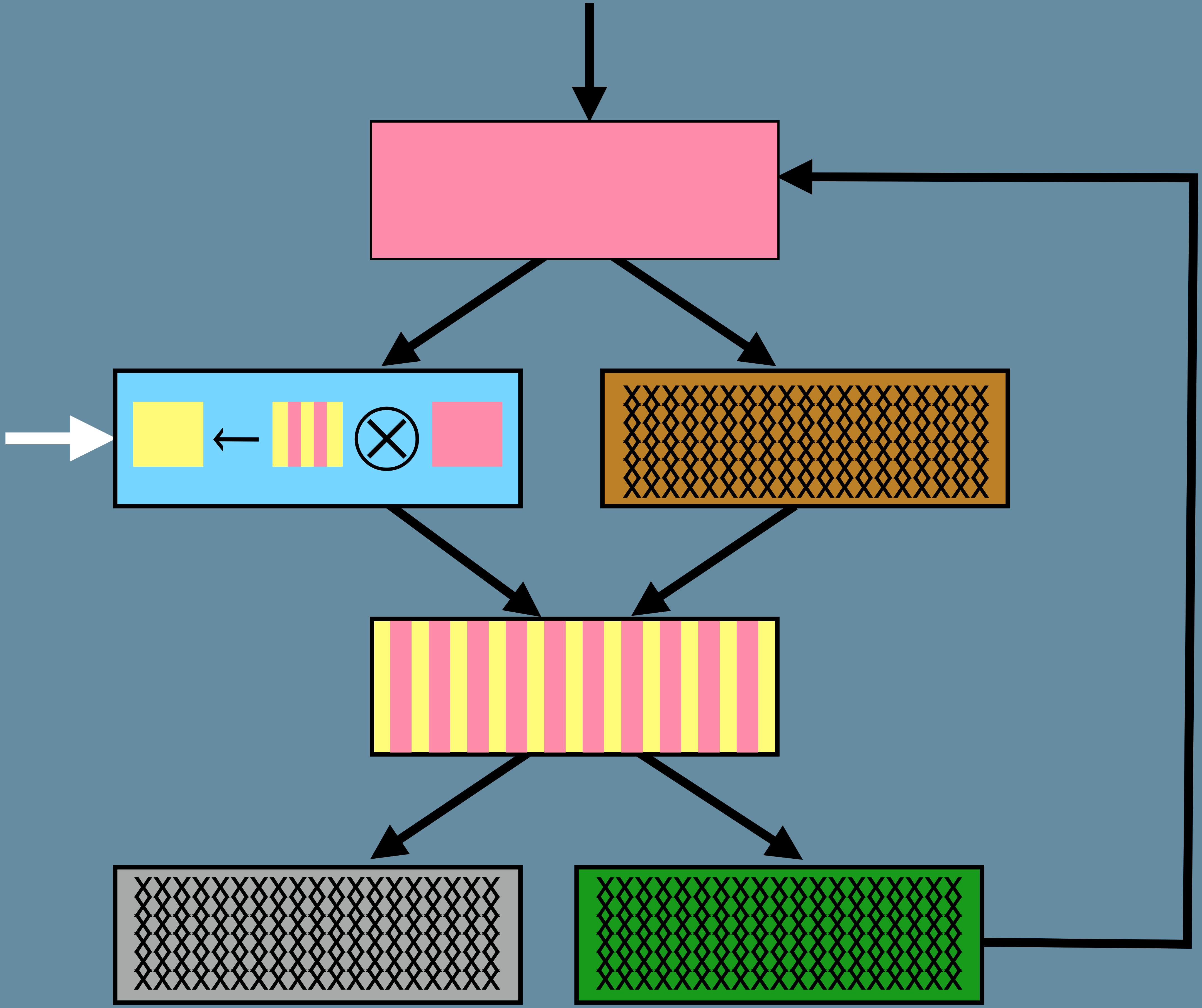


DEC (🟦)

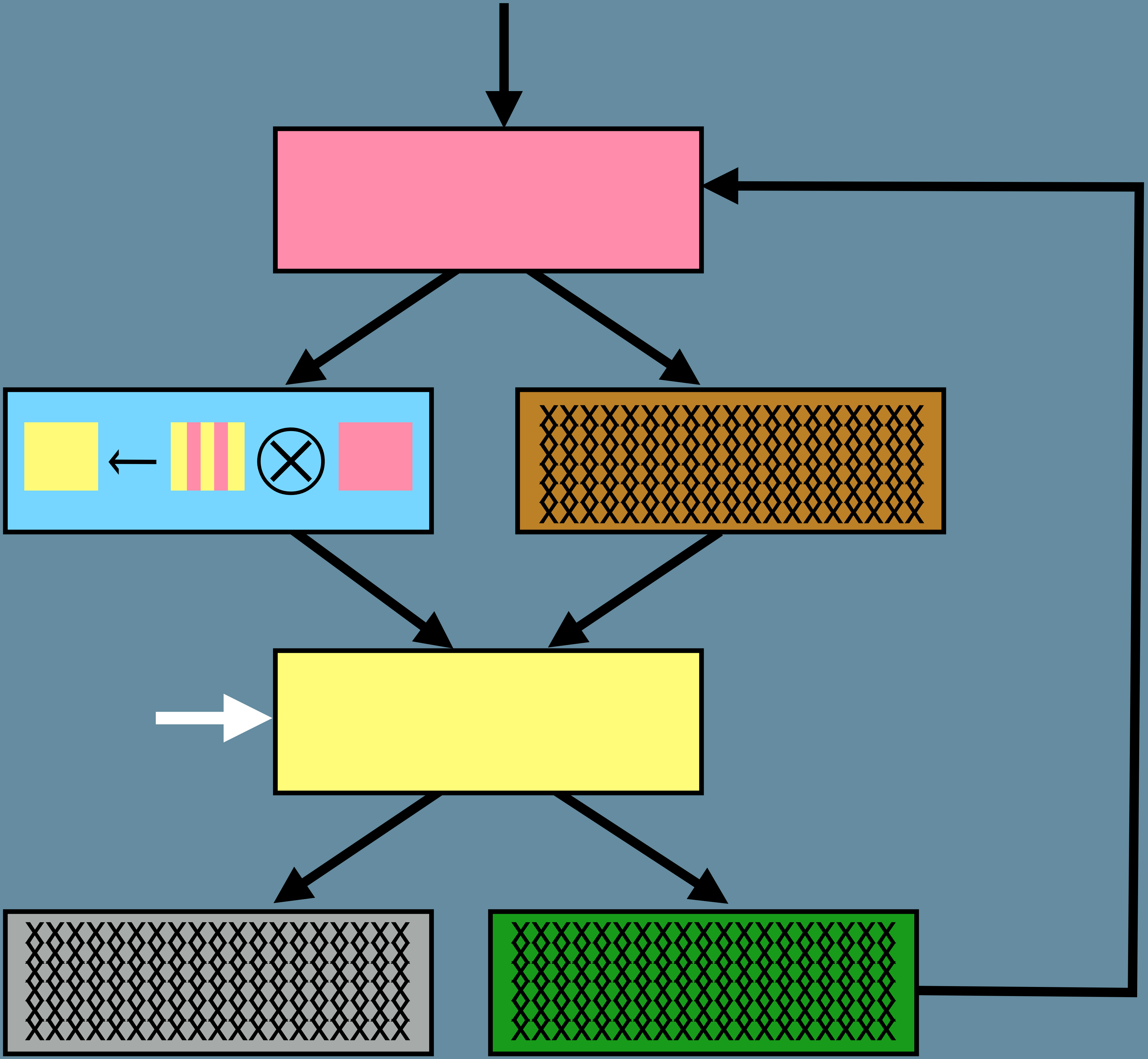
DEC (🟨)

DEC (🟨)

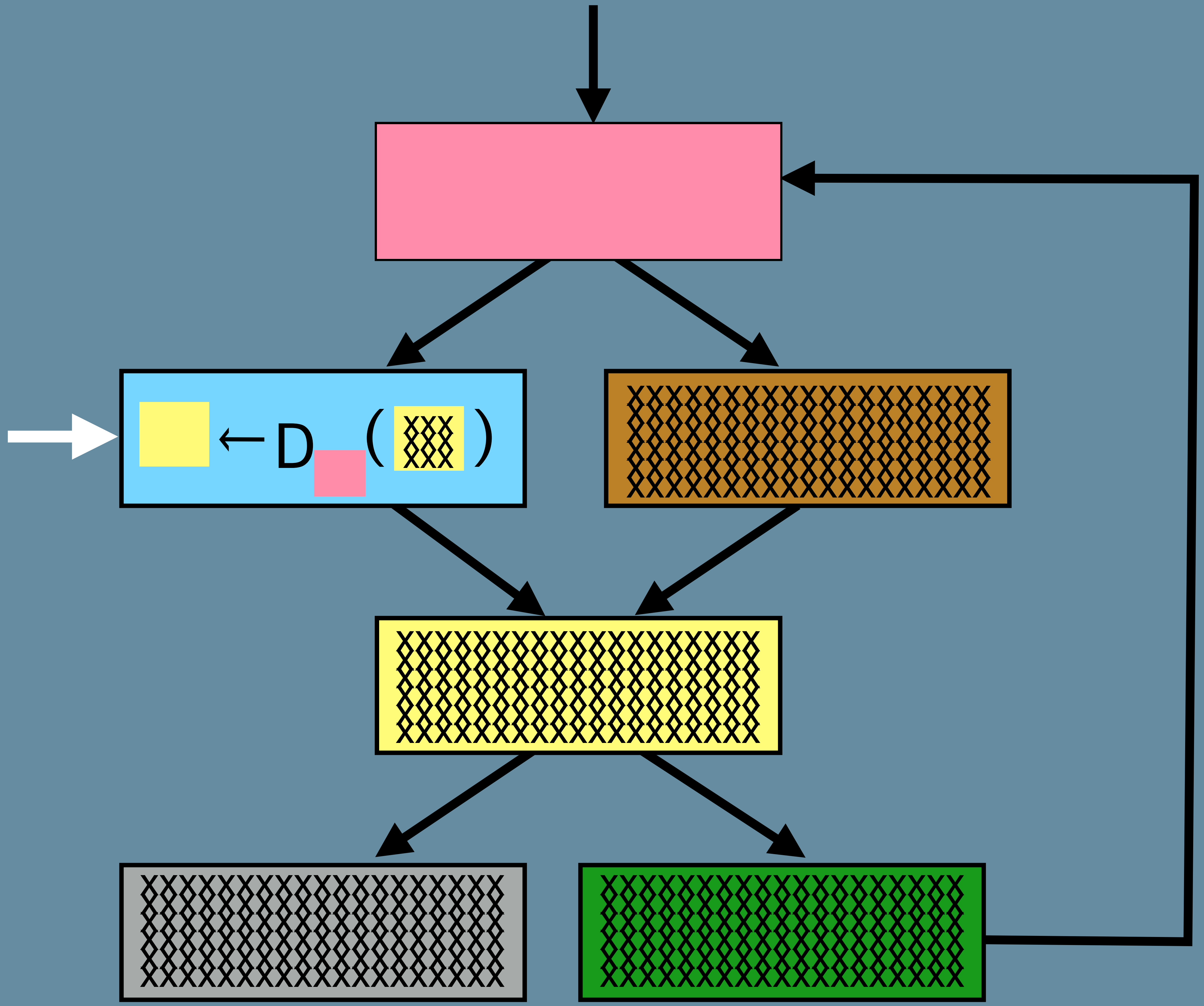
ENC (🟦)
DEC (🟩)



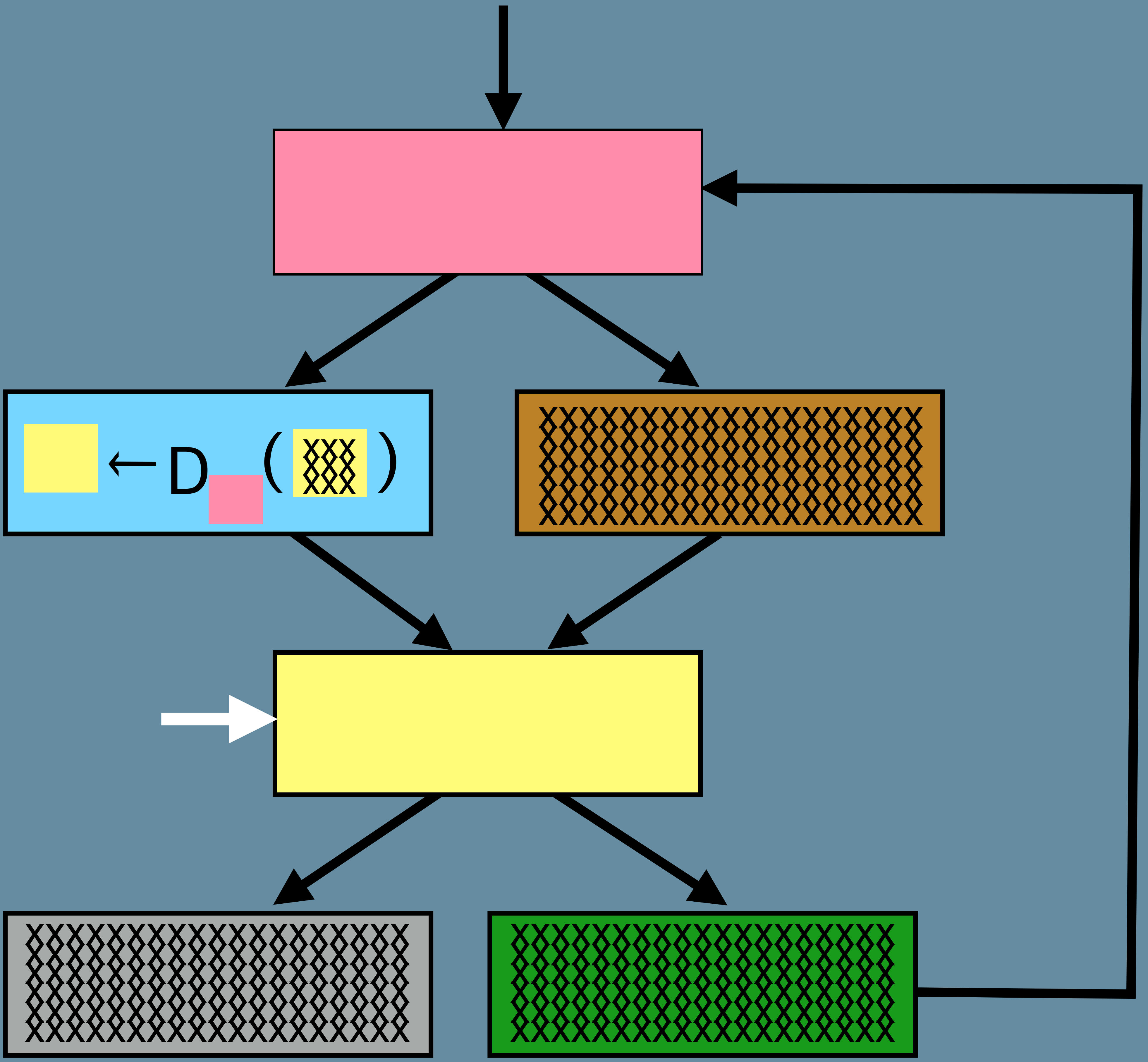
Aucsmith, Tamper Resistant Software: An Implementation, IH'96



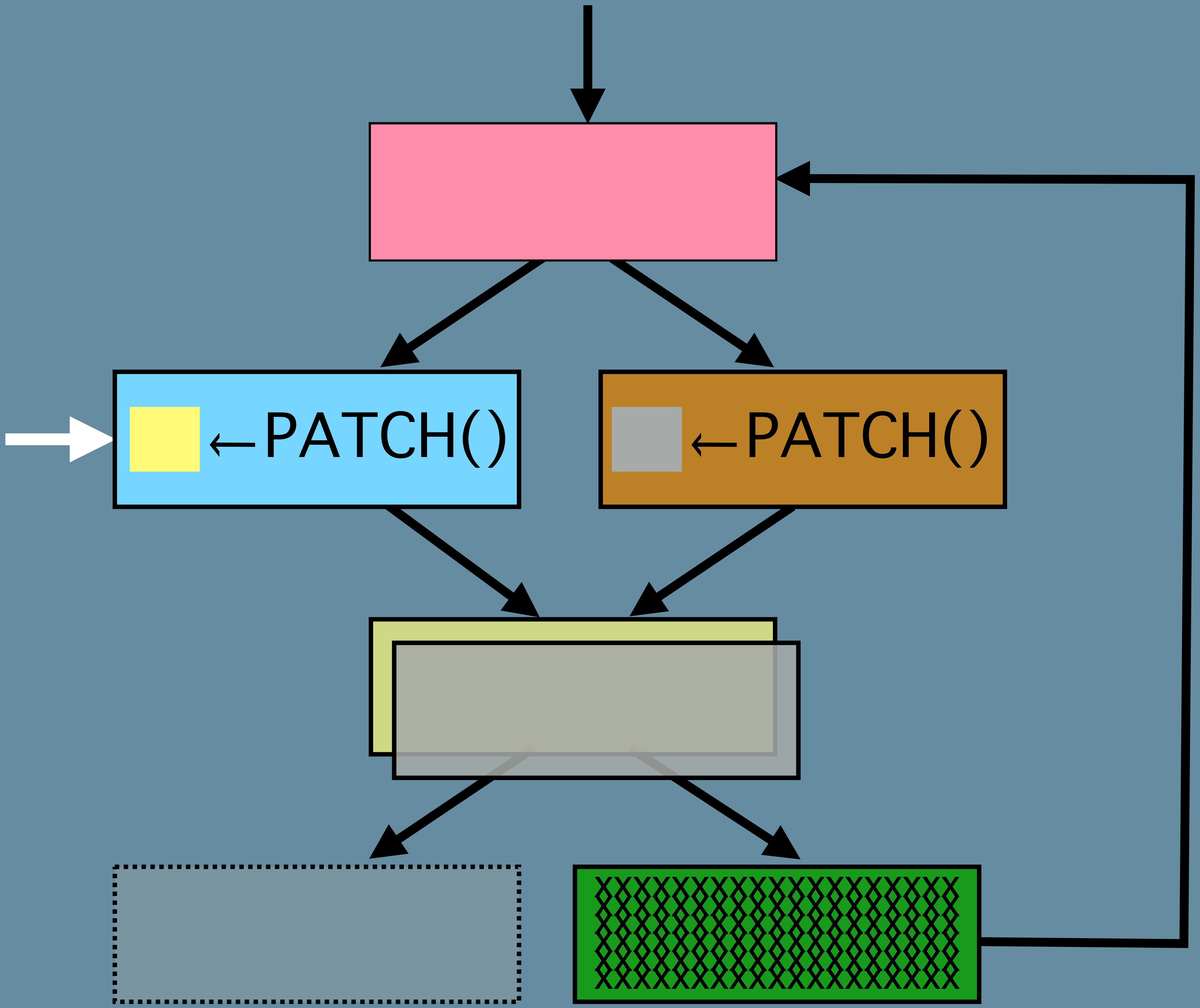
Aucsmith, Tamper Resistant Software: An Implementation, IH’96



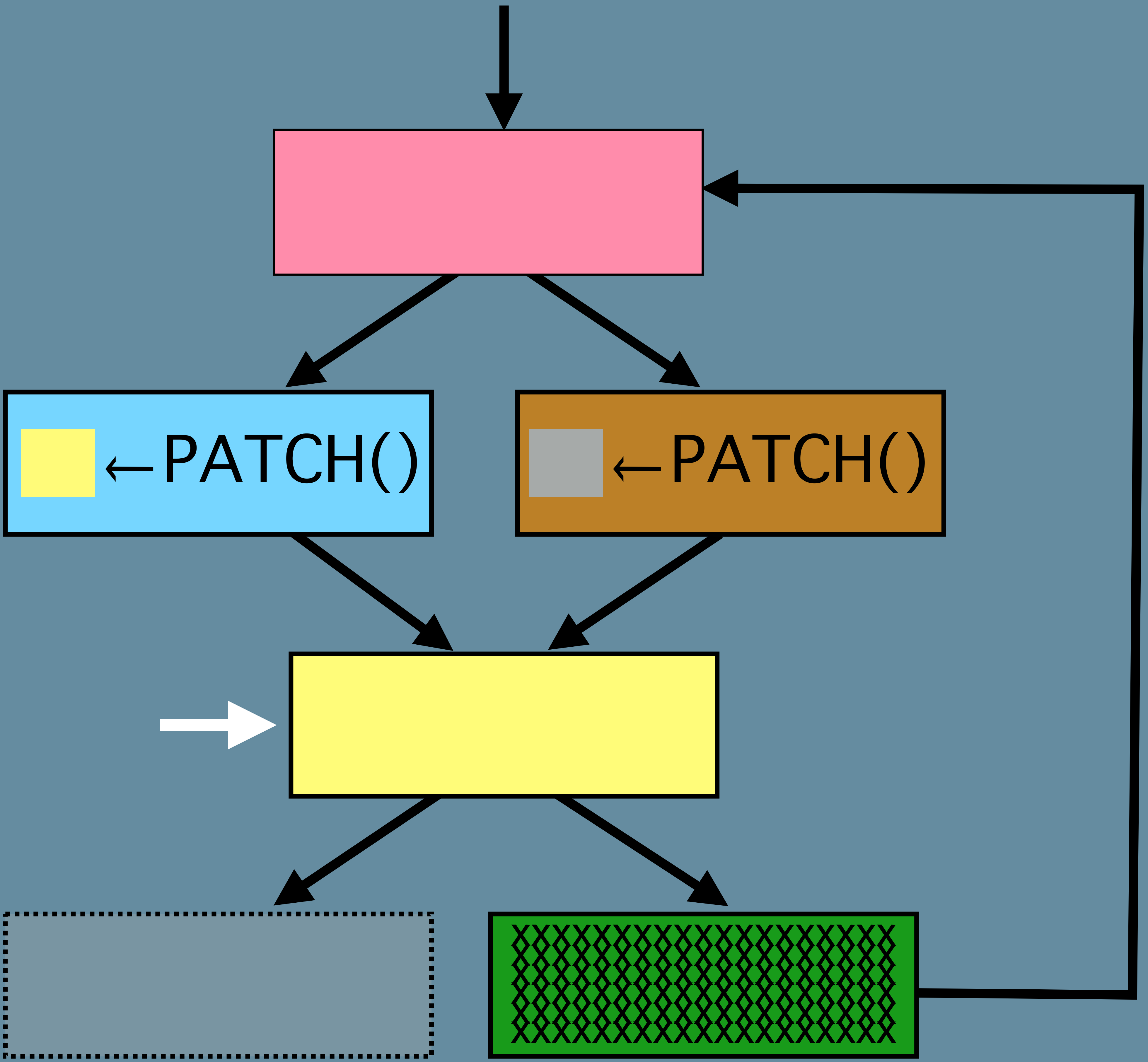
Cappaert, Preneel, et al. Towards Tamper Resistant Code Encryption P&E, ISPEC'08



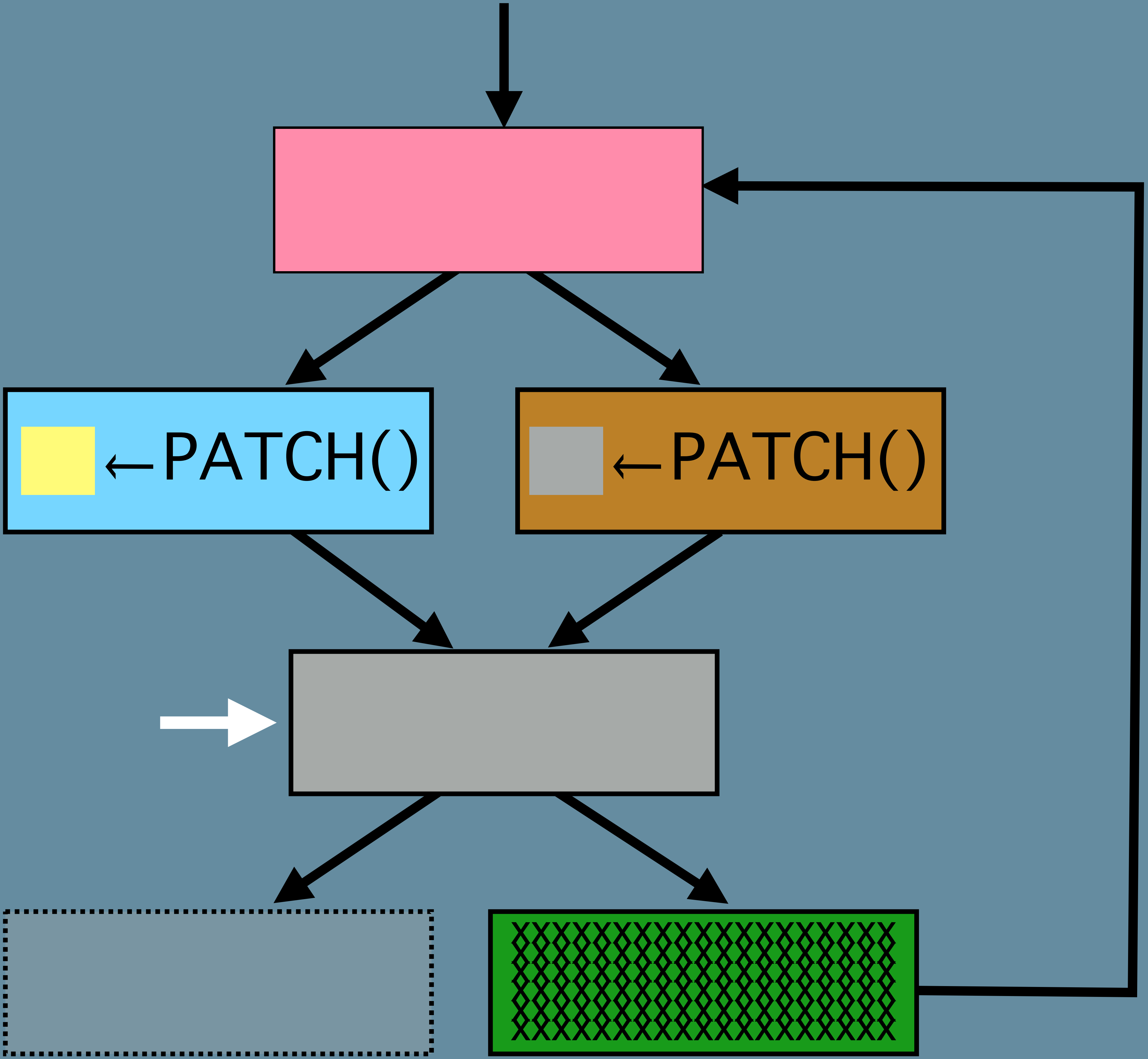
Cappaert, Preneel, et al. Towards Tamper Resistant Code Encryption P&E, ISPEC'08



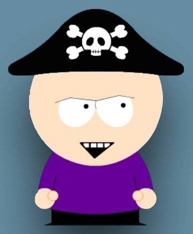
Madou, et al., Software protection through dynamic code mutation, WISA'05



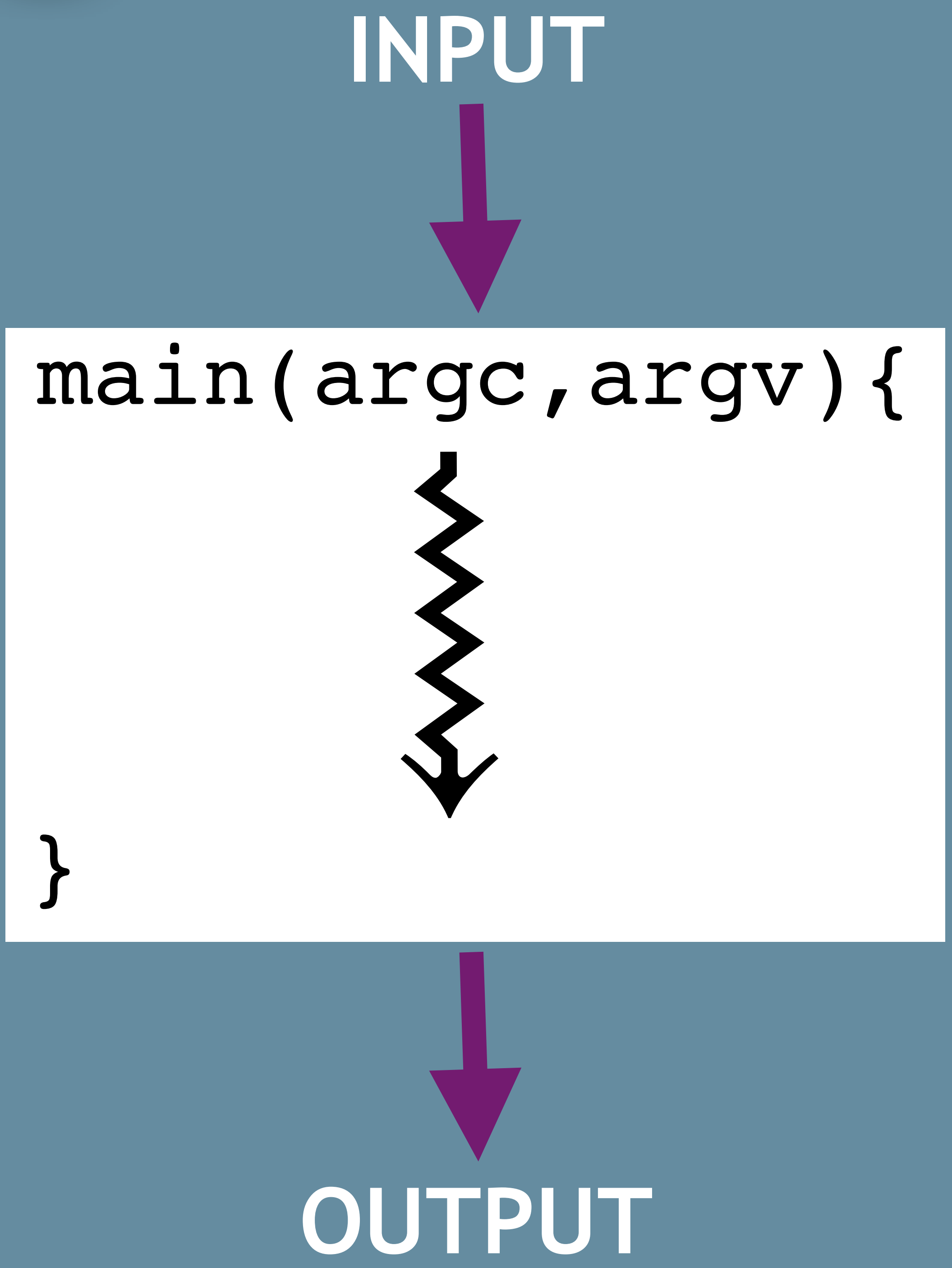
Madou, et al., Software protection through dynamic code mutation, WISA’05



Madou, et al., Software protection through dynamic code mutation, WISA'05



Dynamic Analysis



INPUT

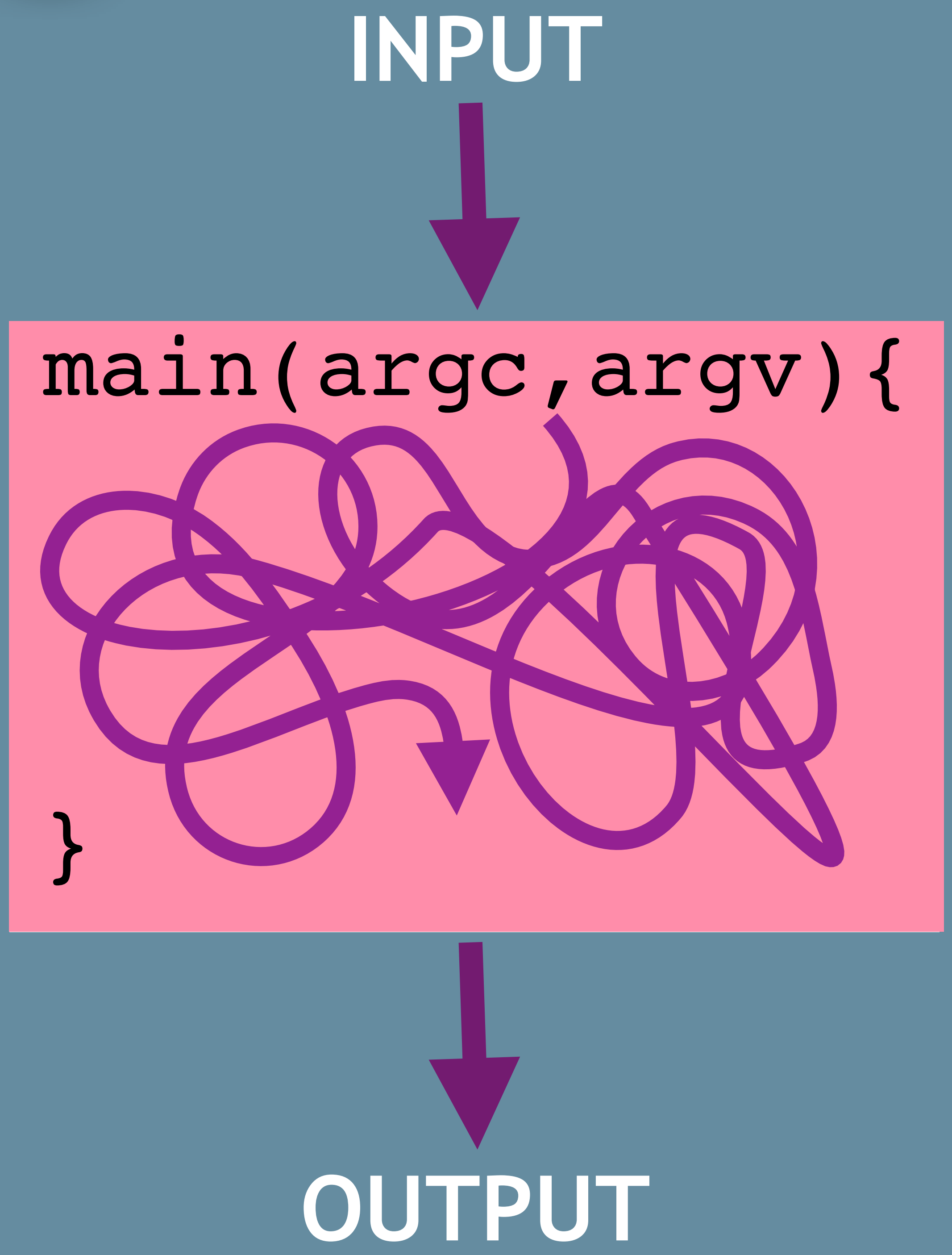
```
main(argc,argv){

}
```

OUTPUT

Yadegari, et al., A Generic Approach to Deobfuscation. IEEE S&P'15

Dynamic Analysis



INPUT

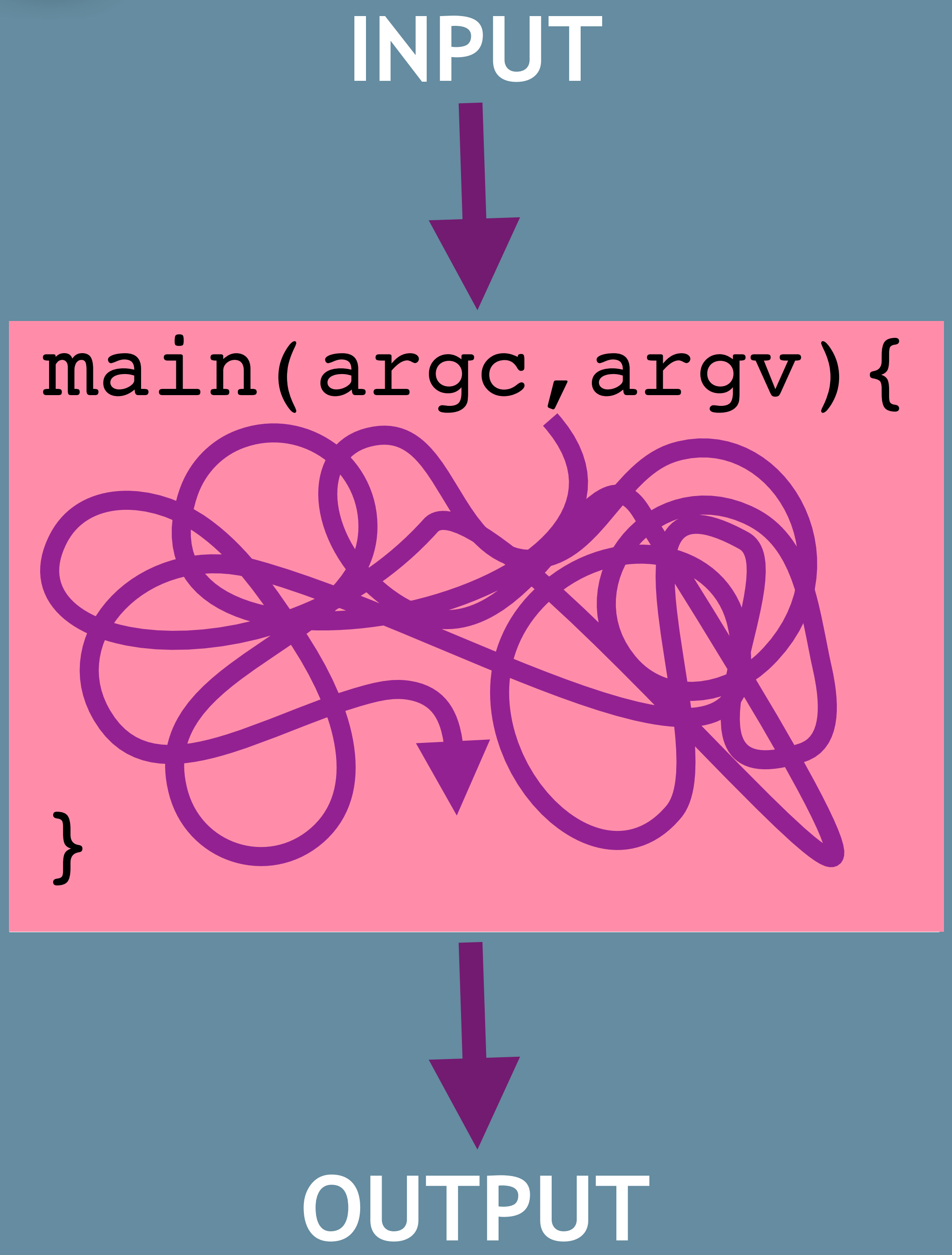
```
main(argc,argv){

}
```

OUTPUT

Yadegari, et al., A Generic Approach to Deobfuscation. IEEE S&P'15

Dynamic Analysis



INPUT

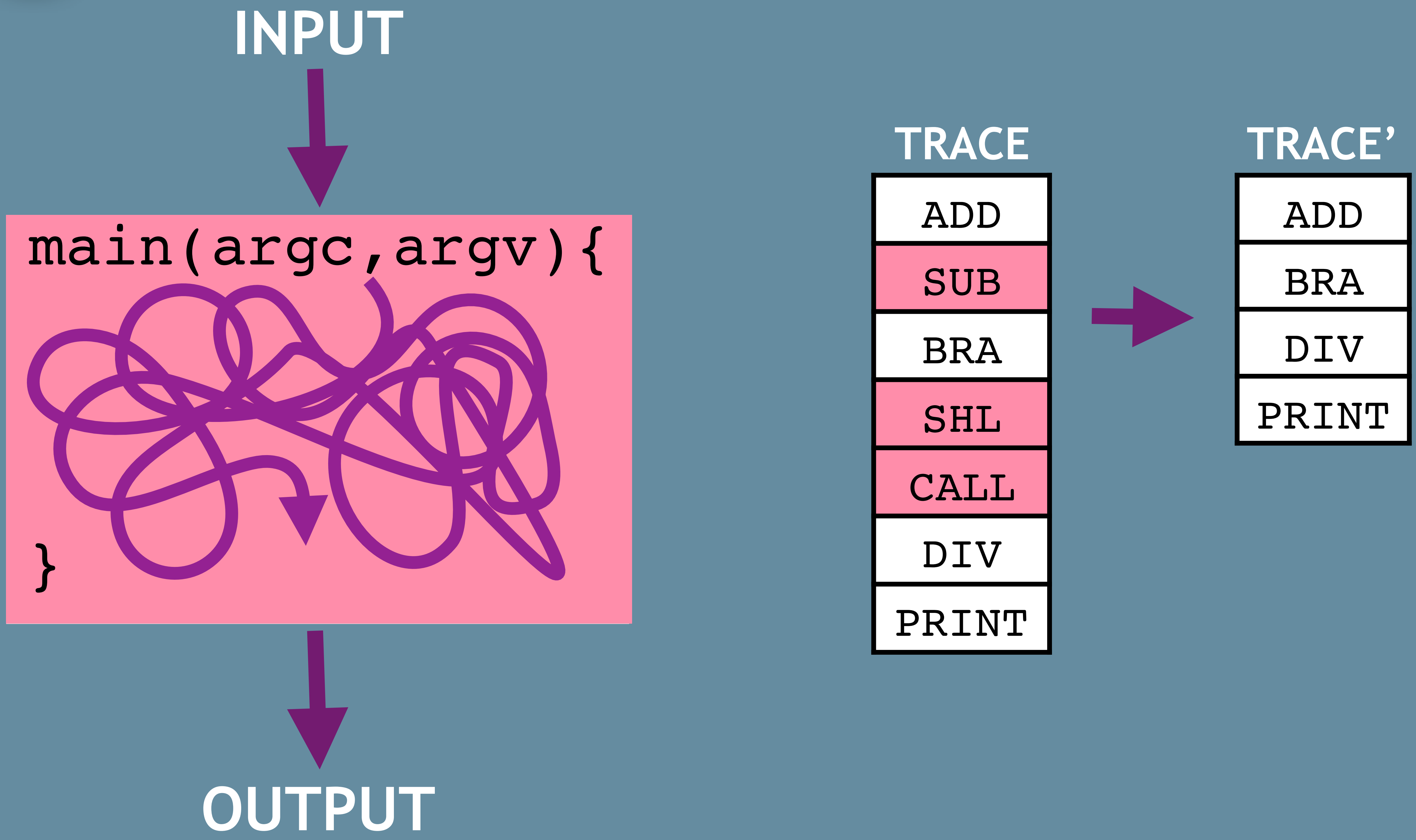
```
main(argc,argv){
}
```

OUTPUT

TRACE
ADD
SUB
BRA
SHL
CALL
DIV
PRINT

Yadegari, et al., A Generic Approach to Deobfuscation. IEEE S&P'15

Dynamic Analysis



INPUT

```
main(argc,argv){

}
```

OUTPUT

TRACE

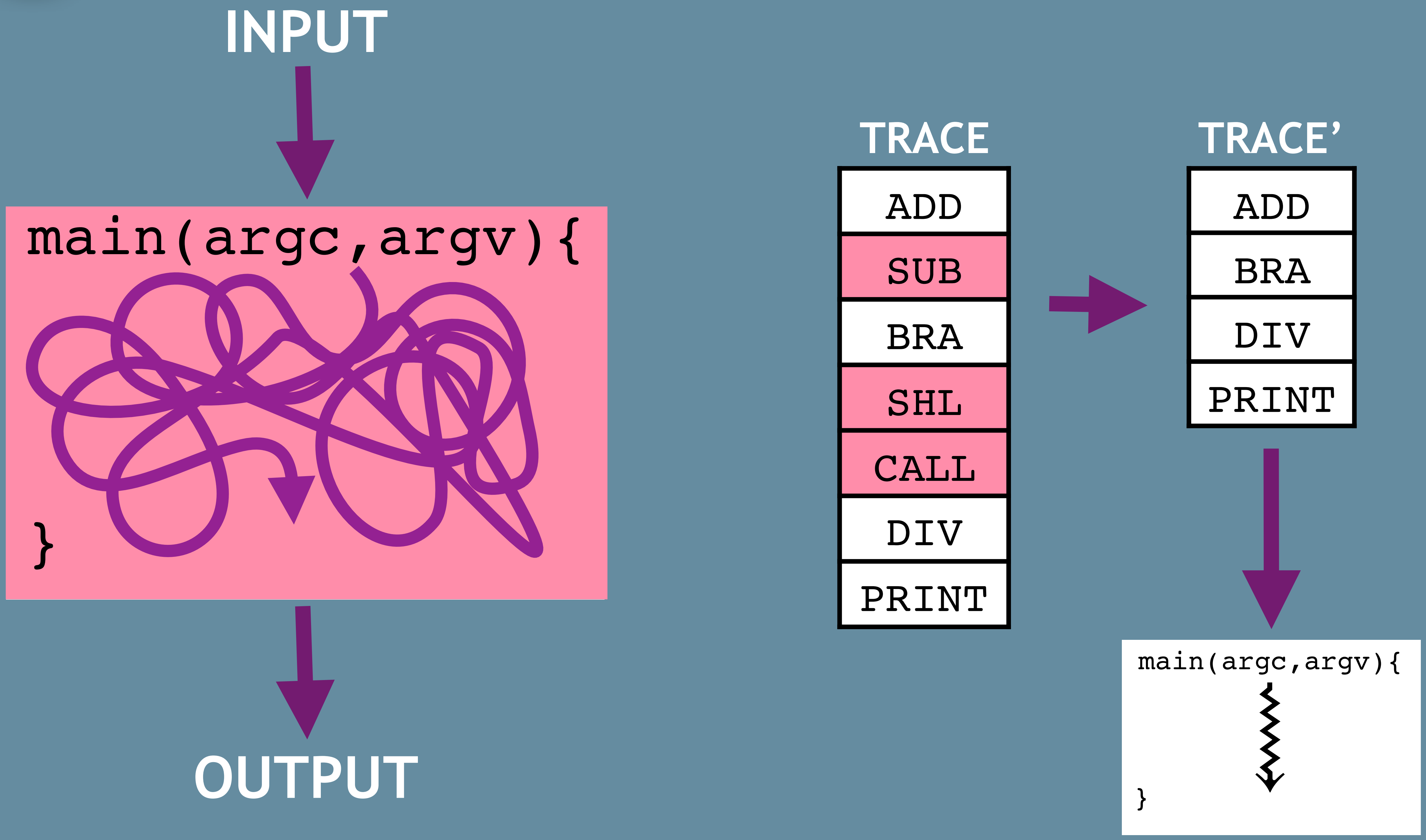
- ADD
- SUB
- BRA
- SHL
- CALL
- DIV
- PRINT

TRACE’

- ADD
- BRA
- DIV
- PRINT

Yadegari, et al., A Generic Approach to Deobfuscation. IEEE S&P’15

Dynamic Analysis



INPUT

```
main(argc,argv){
}
```

OUTPUT

TRACE

- ADD
- SUB
- BRA
- SHL
- CALL
- DIV
- PRINT

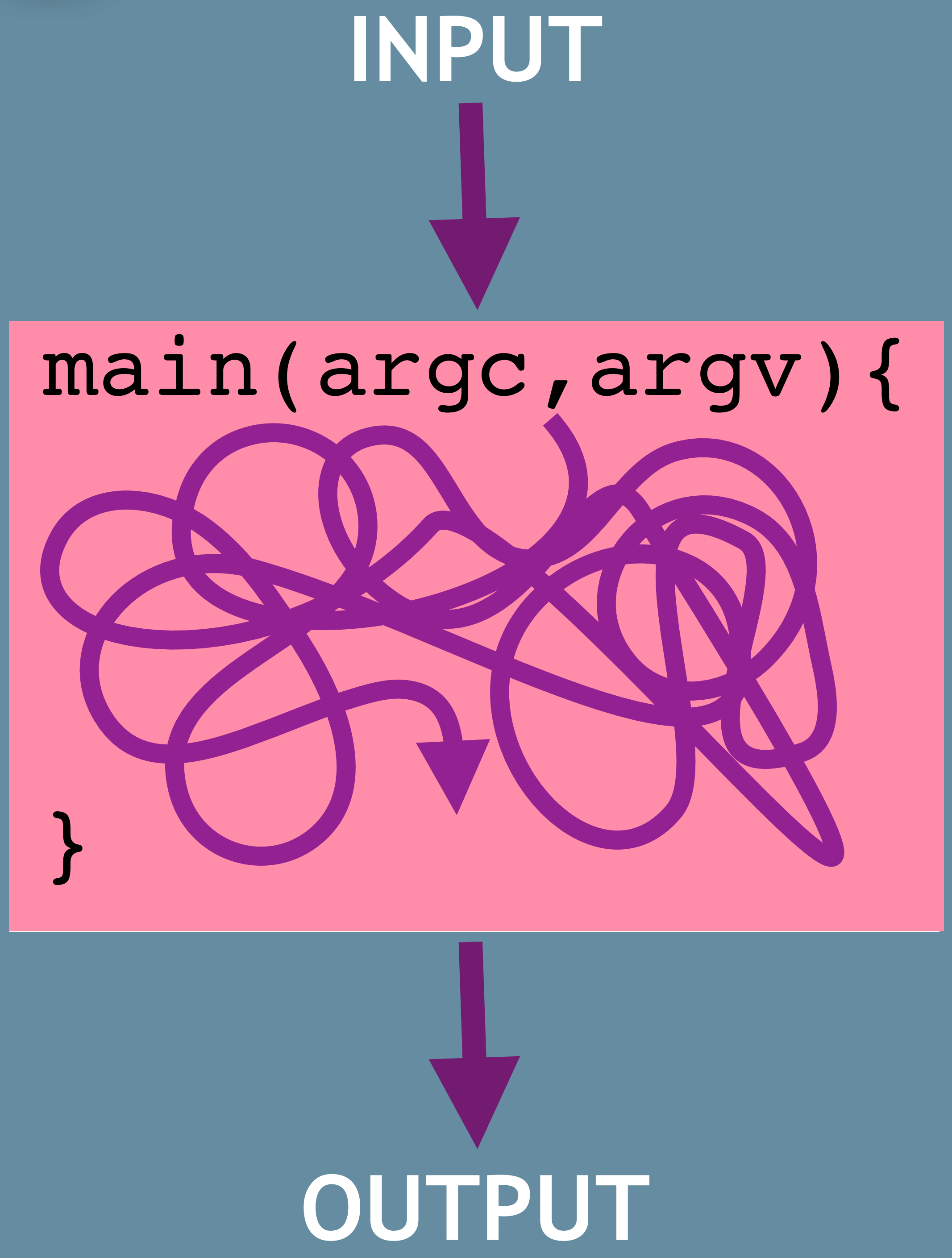
TRACE’

- ADD
- BRA
- DIV
- PRINT

```
main(argc,argv){
}
```

Yadegari, et al., A Generic Approach to Deobfuscation. IEEE S&P’15

Dynamic Analysis



INPUT

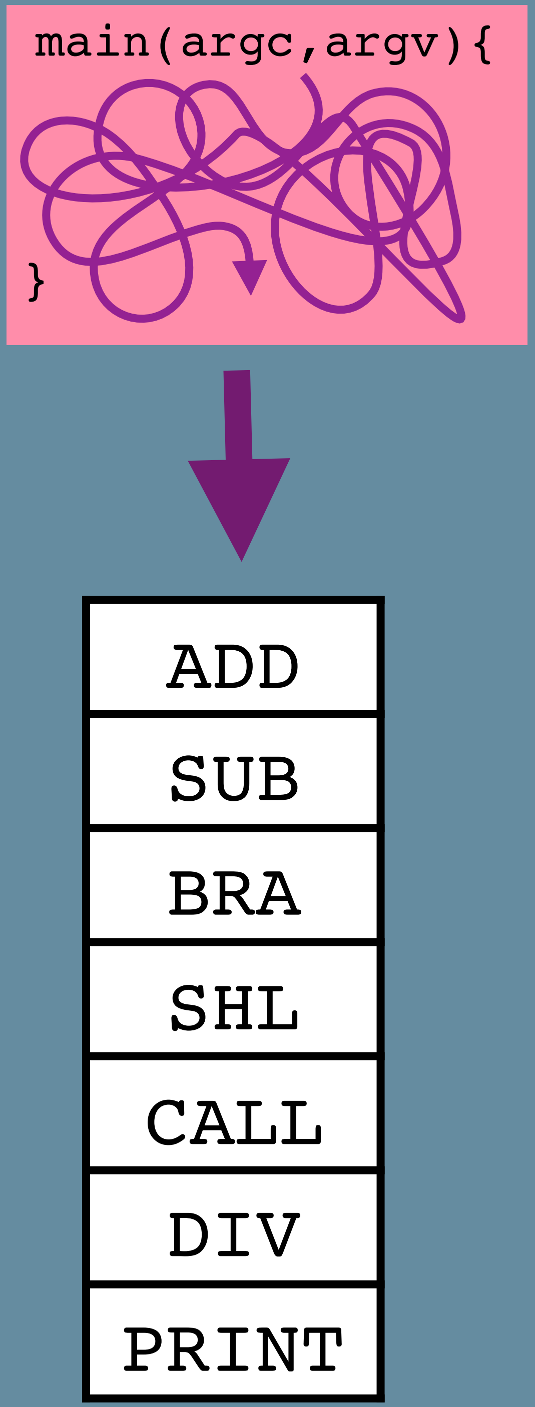
```
main(argc,argv){

}
```

OUTPUT

- Huge traces
- Make traces even larger
- Trace may not cover all paths
- Prevent traces from being collected

Yadegari, et al., A Generic Approach to Deobfuscation. IEEE S&P'15

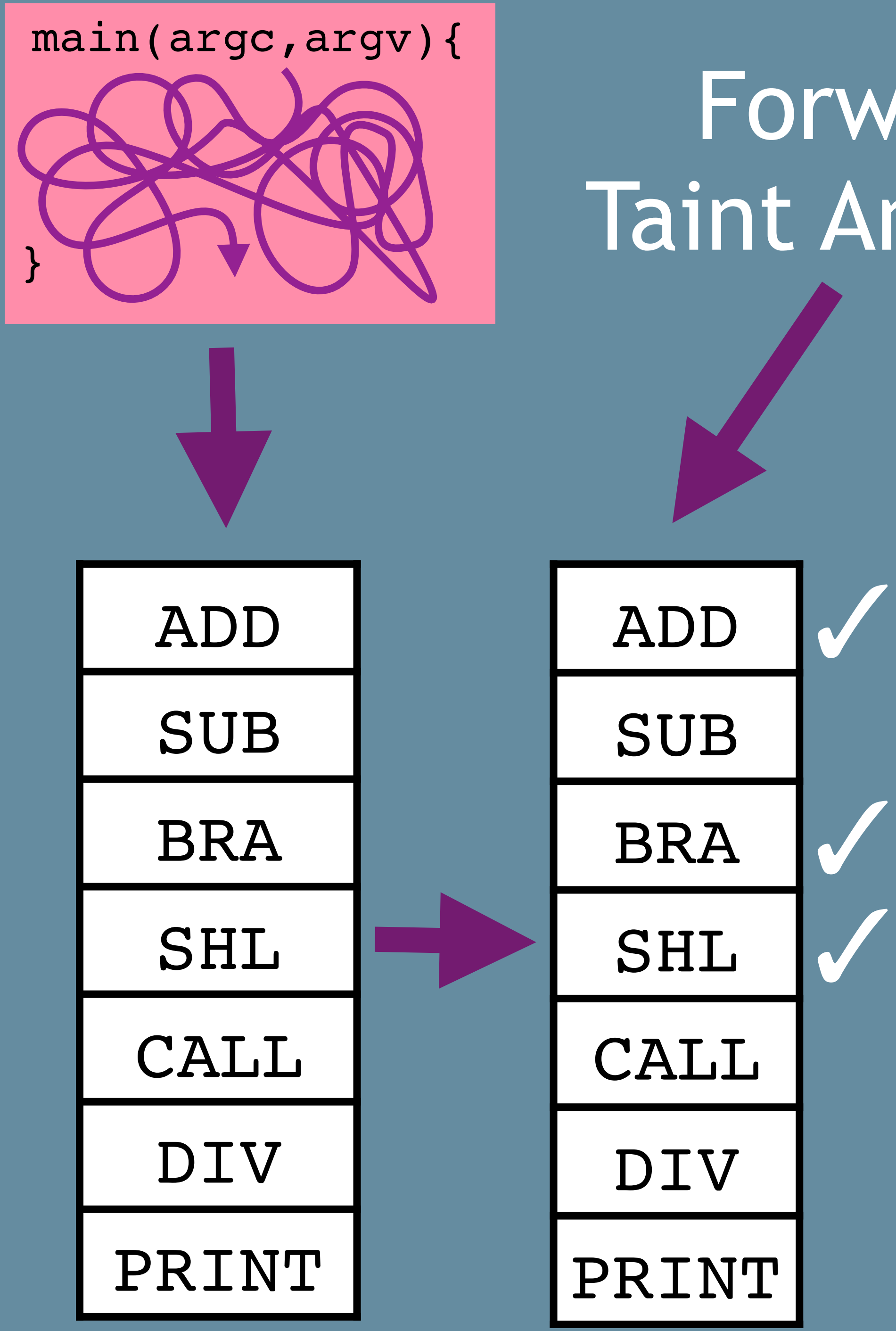


```
main(argc,argv){
}
```

ADD
SUB
BRA
SHL
CALL
DIV
PRINT

Yadegari, et al., A Generic Approach to Deobfuscation. IEEE S&P'15

Forward Taint Analysis



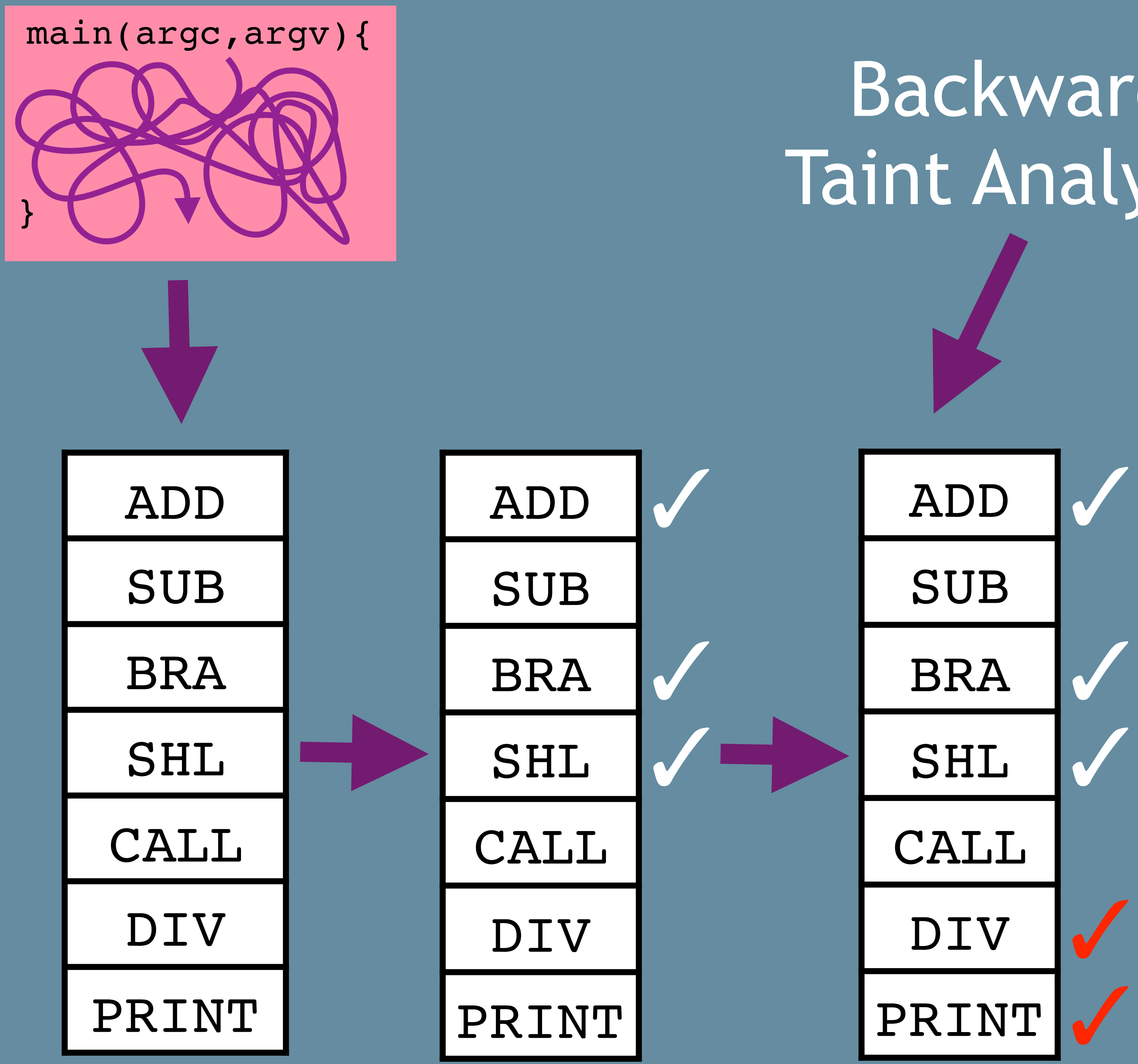
```
main(argc,argv){
}
```

- ADD
- SUB
- BRA
- SHL
- CALL
- DIV
- PRINT

- ADD ✓
- SUB
- BRA ✓
- SHL ✓
- CALL
- DIV
- PRINT

Yadegari, et al., A Generic Approach to Deobfuscation. IEEE S&P'15

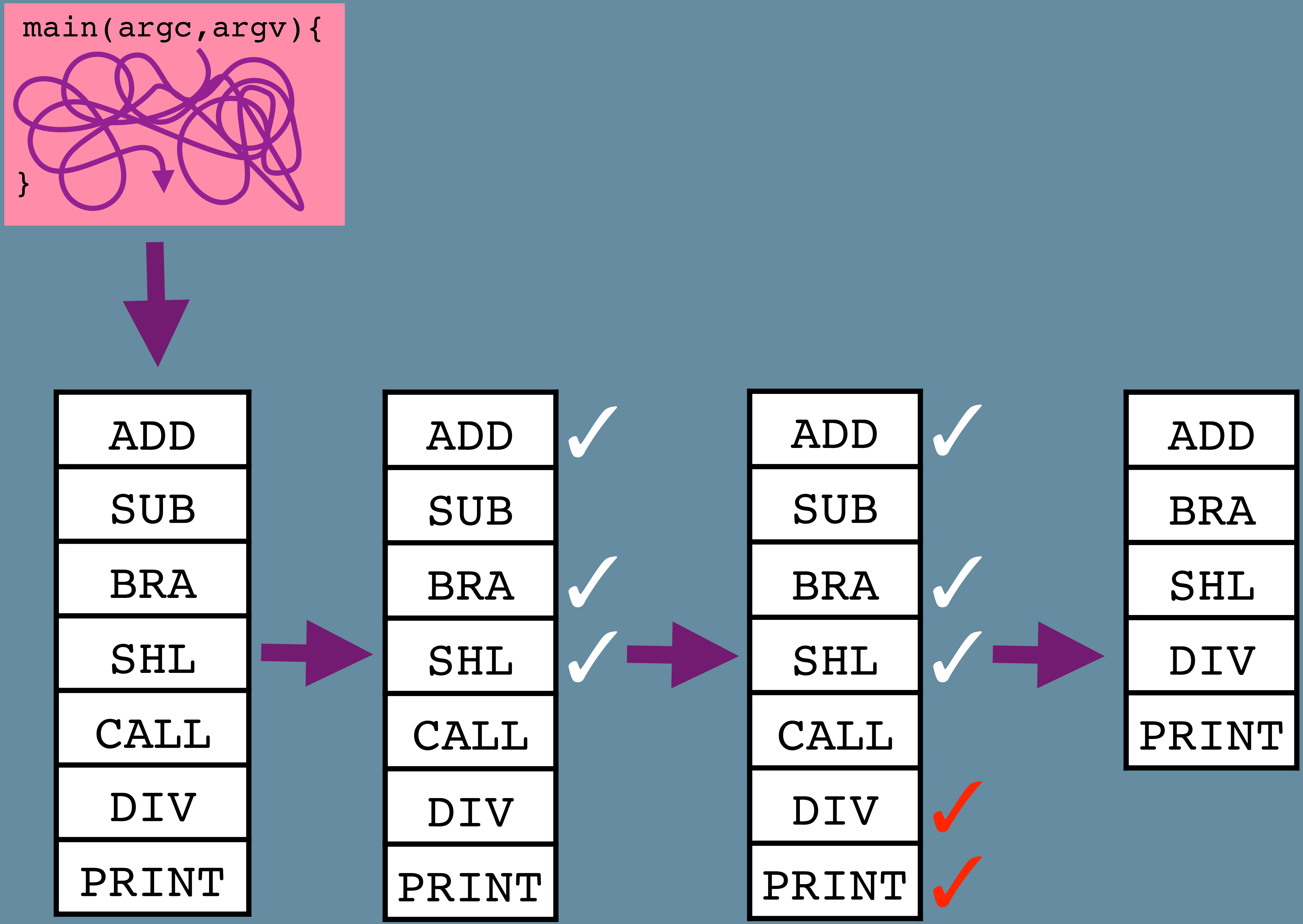
Backward Taint Analysis



```
main(argc,argv){
}
```

ADD	ADD	✓	ADD	✓
SUB	SUB		SUB	
BRA	BRA	✓	BRA	✓
SHL	SHL	✓	SHL	✓
CALL	CALL		CALL	
DIV	DIV		DIV	✓
PRINT	PRINT		PRINT	✓

Yadegari, et al., A Generic Approach to Deobfuscation. IEEE S&P'15



```
main(argc,argv){
}
```

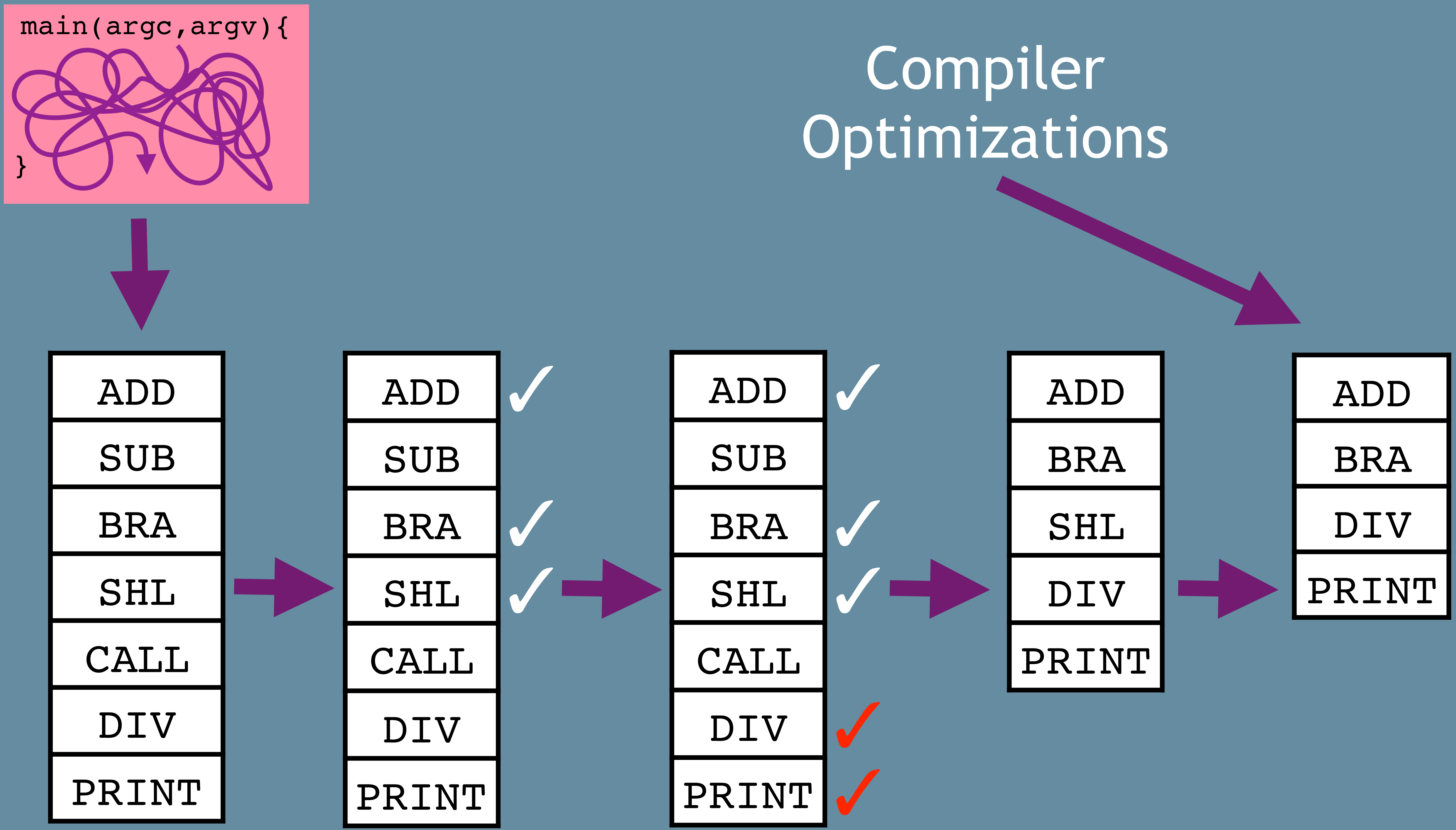
ADD
SUB
BRA
SHL
CALL
DIV
PRINT

ADD ✓
SUB
BRA ✓
SHL ✓
CALL
DIV
PRINT

ADD ✓
SUB
BRA ✓
SHL ✓
CALL
DIV ✓
PRINT ✓

ADD
BRA
SHL
DIV
PRINT

Yadegari, et al., A Generic Approach to Deobfuscation. IEEE S&P'15

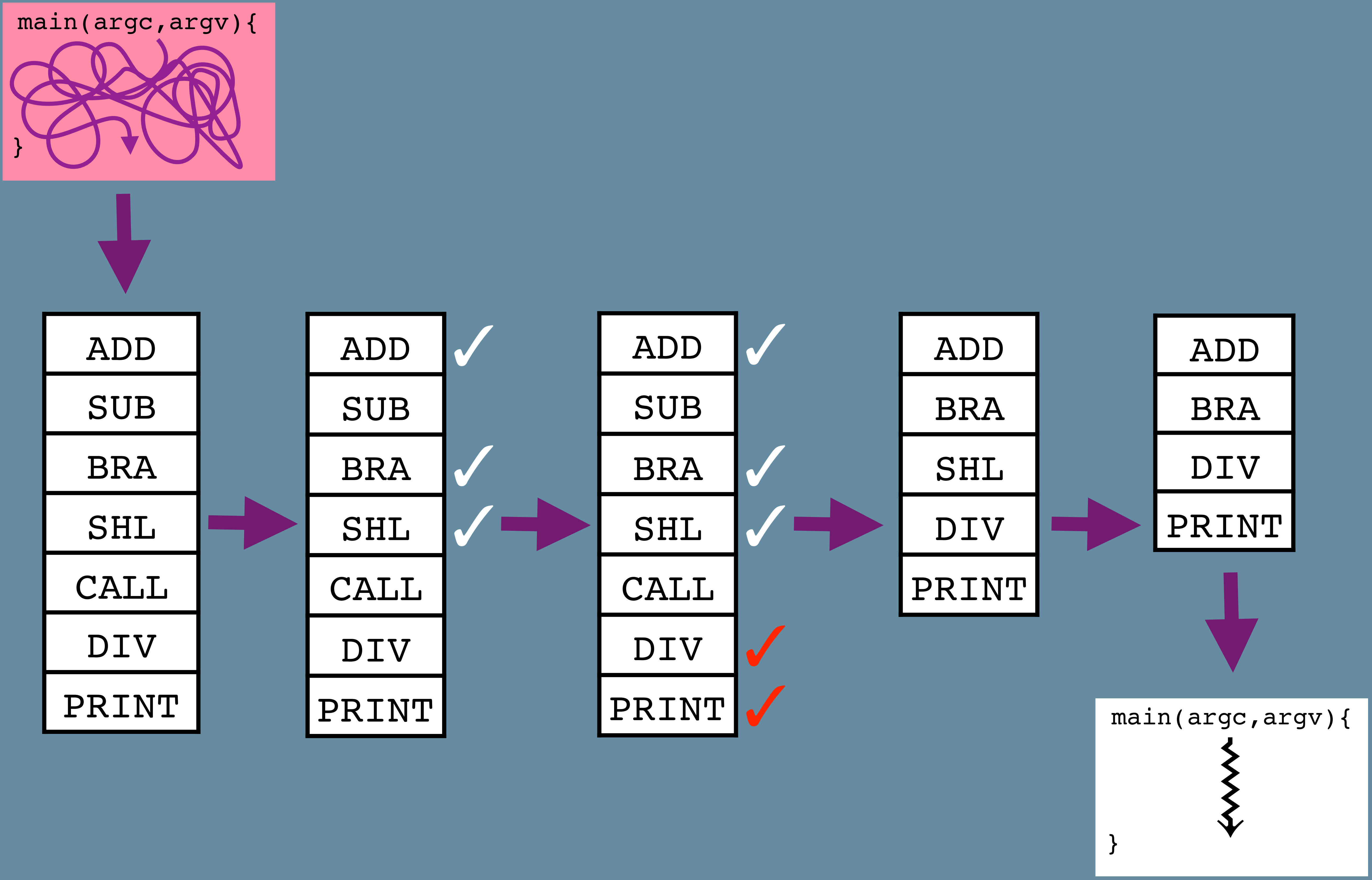


```
main(argc,argv){
}
```

Compiler Optimizations

ADD	ADD ✓	ADD ✓	ADD	ADD
SUB	SUB	SUB	BRA	BRA
BRA	BRA ✓	BRA ✓	SHL	DIV
SHL	SHL ✓	SHL ✓	DIV	PRINT
CALL	CALL	CALL	PRINT	
DIV	DIV	DIV ✓		
PRINT	PRINT	PRINT ✓		

Yadegari, et al., A Generic Approach to Deobfuscation. IEEE S&P’15



main(argc,argv){
}

ADD
SUB
BRA
SHL
CALL
DIV
PRINT

ADD ✓
SUB
BRA ✓
SHL ✓
CALL
DIV
PRINT

ADD ✓
SUB
BRA ✓
SHL ✓
CALL
DIV ✓
PRINT ✓

ADD
BRA
SHL
DIV
PRINT

ADD
BRA
DIV
PRINT

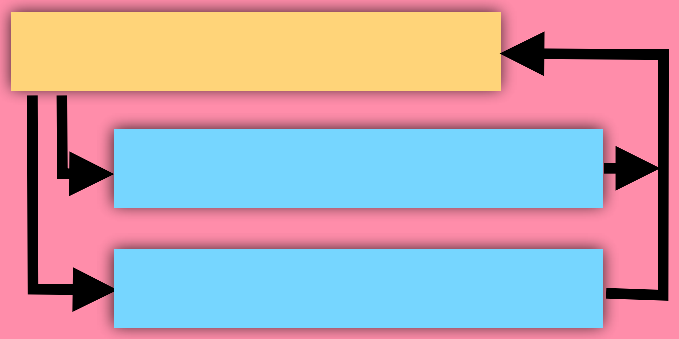
main(argc,argv){
}

Yadegari, et al., A Generic Approach to Deobfuscation. IEEE S&P'15

```
void main(argc,argv){
    VPC = 0;
    STACK = [];
```

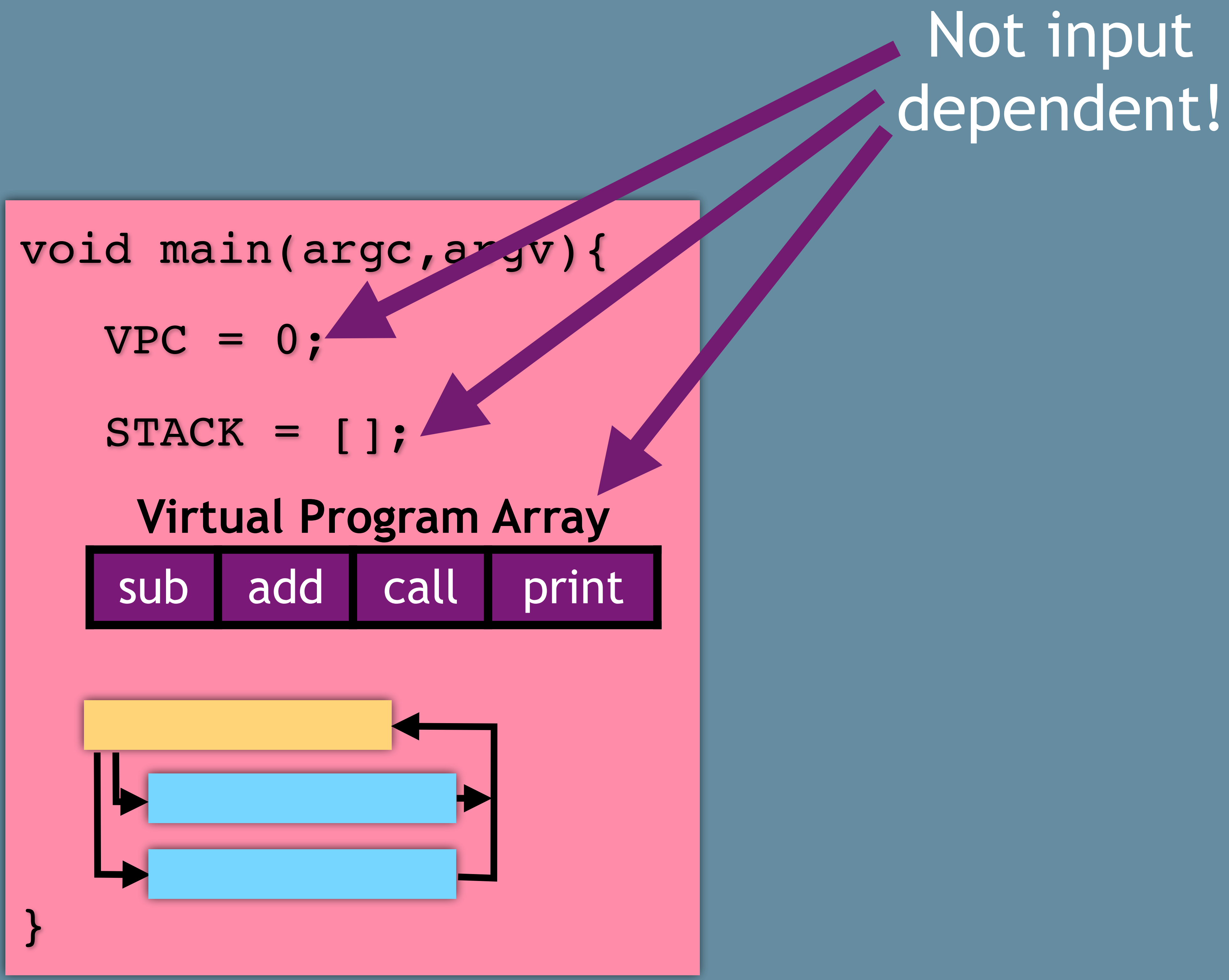
Virtual Program Array

sub	add	call	print



```
}
```

Yadegari, et al., A Generic Approach to Deobfuscation. IEEE S&P’15



Not input dependent!

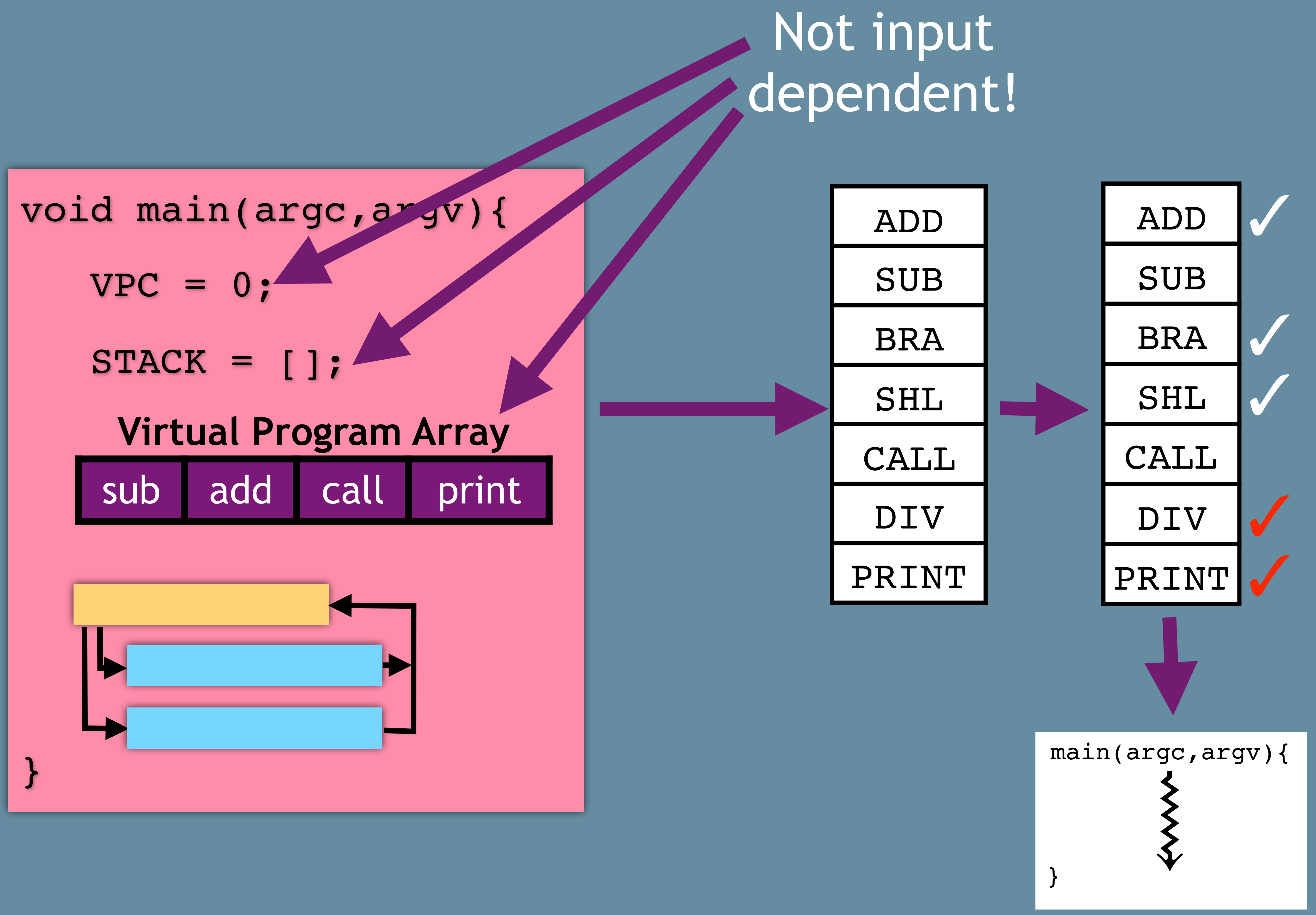
```
void main(argc,argv){
    VPC = 0;
    STACK = [];
```

Virtual Program Array

sub	add	call	print

```
}
```

Yadegari, et al., A Generic Approach to Deobfuscation. IEEE S&P'15



Not input dependent!

```
void main(argc,argv){
    VPC = 0;
    STACK = [];
```

Virtual Program Array

sub	add	call	print

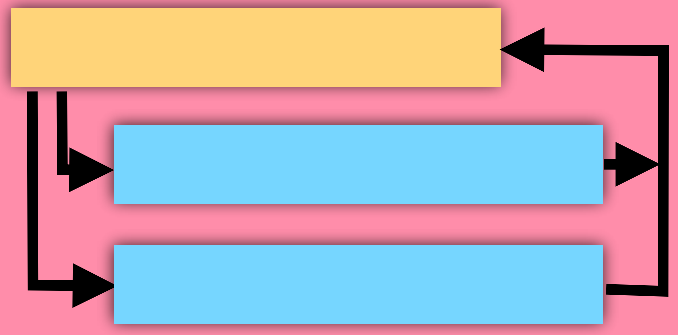
```
}
```

```
main(argc,argv){
}
```

Yadegari, et al., A Generic Approach to Deobfuscation. IEEE S&P’15

Anti-Taint Analysis

```
void main(argc,argv){
    VPC =
    STACK =
    | sub | add | call | print | =
```



```
}
```

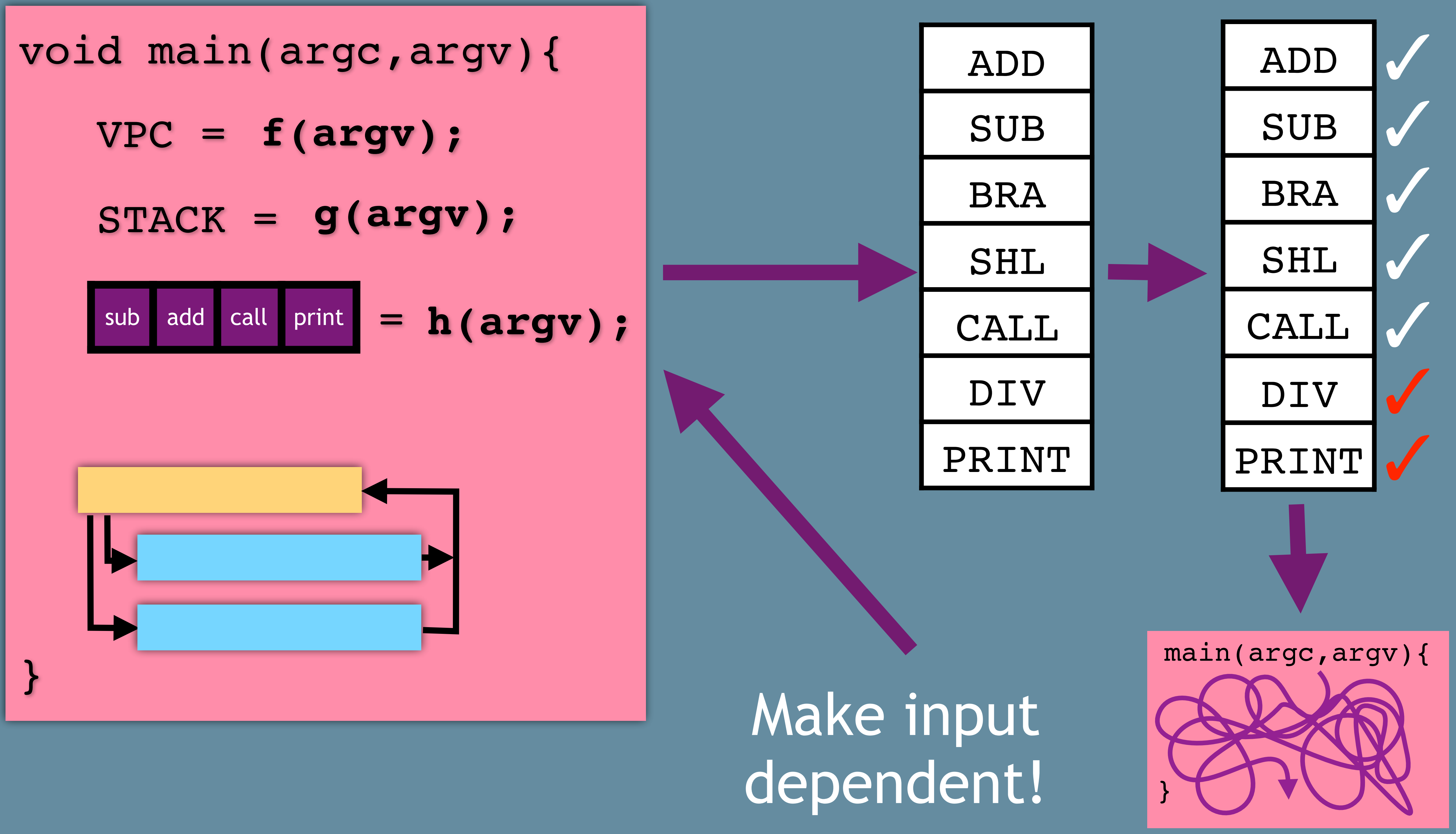
Anti-Taint Analysis

```
void main(argc,argv){
    VPC = f(argv);
    STACK = g(argv);
    sub add call print = h(argv);
```

```
}
```

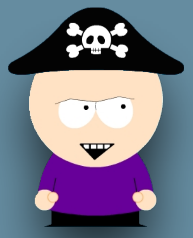
Make input dependent!

Anti-Taint Analysis



```
void main(argc,argv){
    VPC = f(argv);
    STACK = g(argv);
    sub add call print = h(argv);
}
```

Make input dependent!



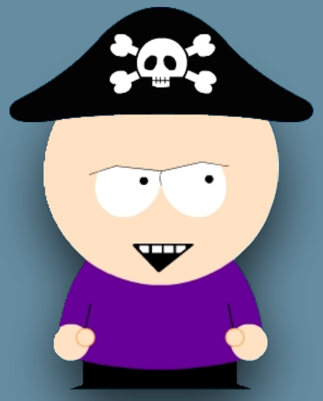
Analysis Performance

Analysis	Program	Virtualization	Analysis Performance
Static Analysis	Fibonacci	Tigress	40s, 71MB
Bit-level taint analysis	Huffman coding	VMProtect	449s, trace size 32M instructions.
Concolic analysis	14 line program	VMProtect	14,160s

Yadegari, Automatic Deobfuscation and Reverse Engineering of Obfuscated Code

Kinder, Towards Static Analysis of Virtualization-Obfuscated Binaries, WCRE'12

Time-Limited Protection



Hohl, Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts

Time-Limited Protection

Hohl, Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts

Time-Limited Protection

Obfuscation provides *time-limited protection*: an adversary will require greater-than-zero length of time to extract an asset from an obfuscated program.

Hohl, Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts

Time-Limited Protection

Obfuscation provides *time-limited protection*: an adversary will require greater-than-zero length of time to extract an asset from an obfuscated program.

How can we get useful levels of protection from individual transformations that only provide time-limited protection?

Hohl, Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts

Deploying Obfuscation

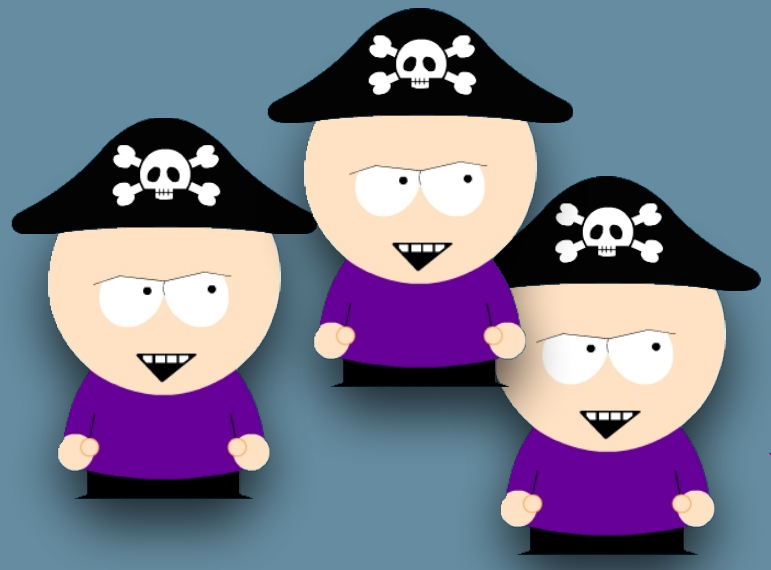
Deploying Obfuscation

?

Deploying Obfuscation

Monitor adversarial communities

Deploying Obfuscation

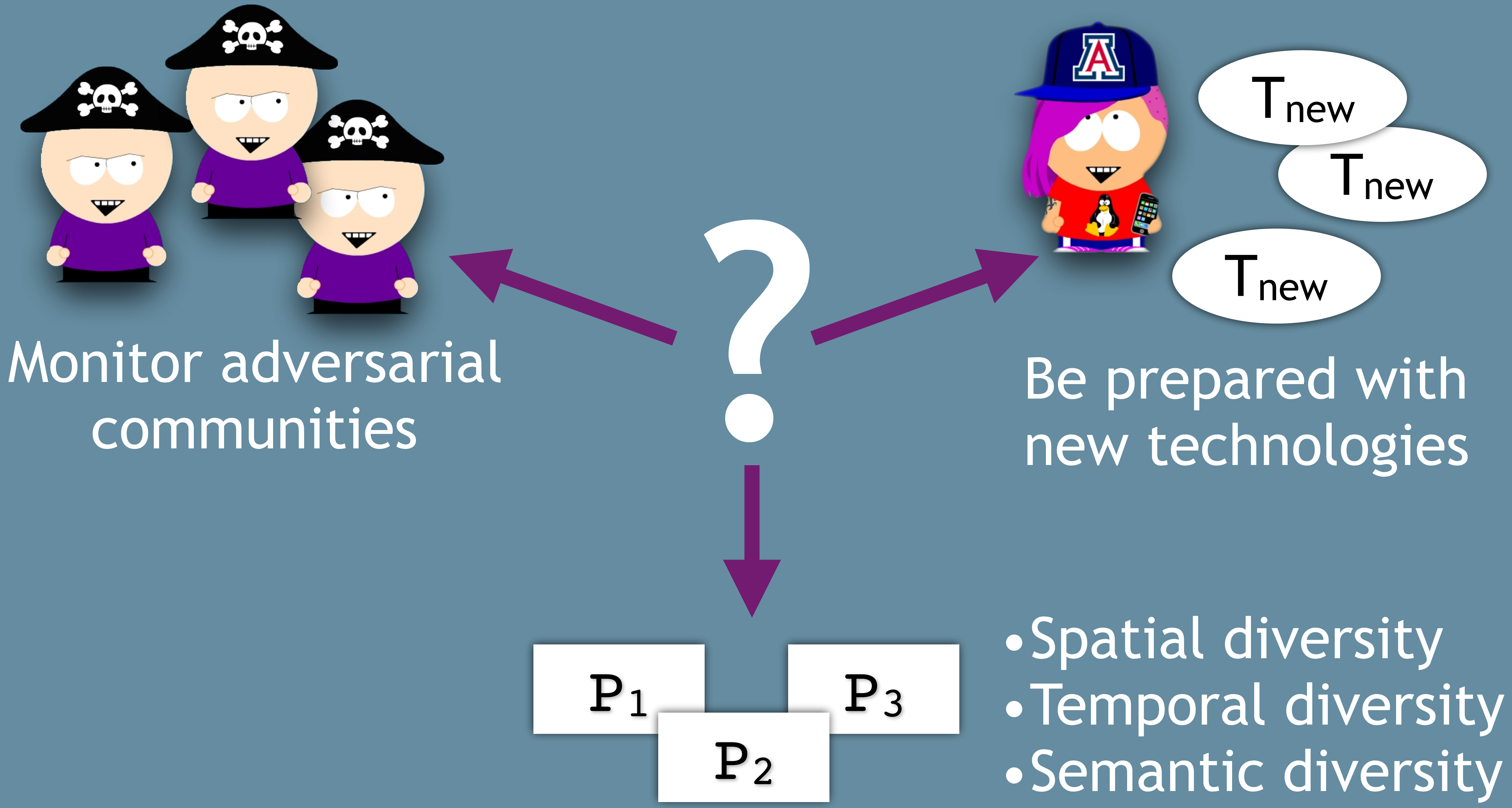


Monitor adversarial communities

?

Be prepared with new technologies

Deploying Obfuscation



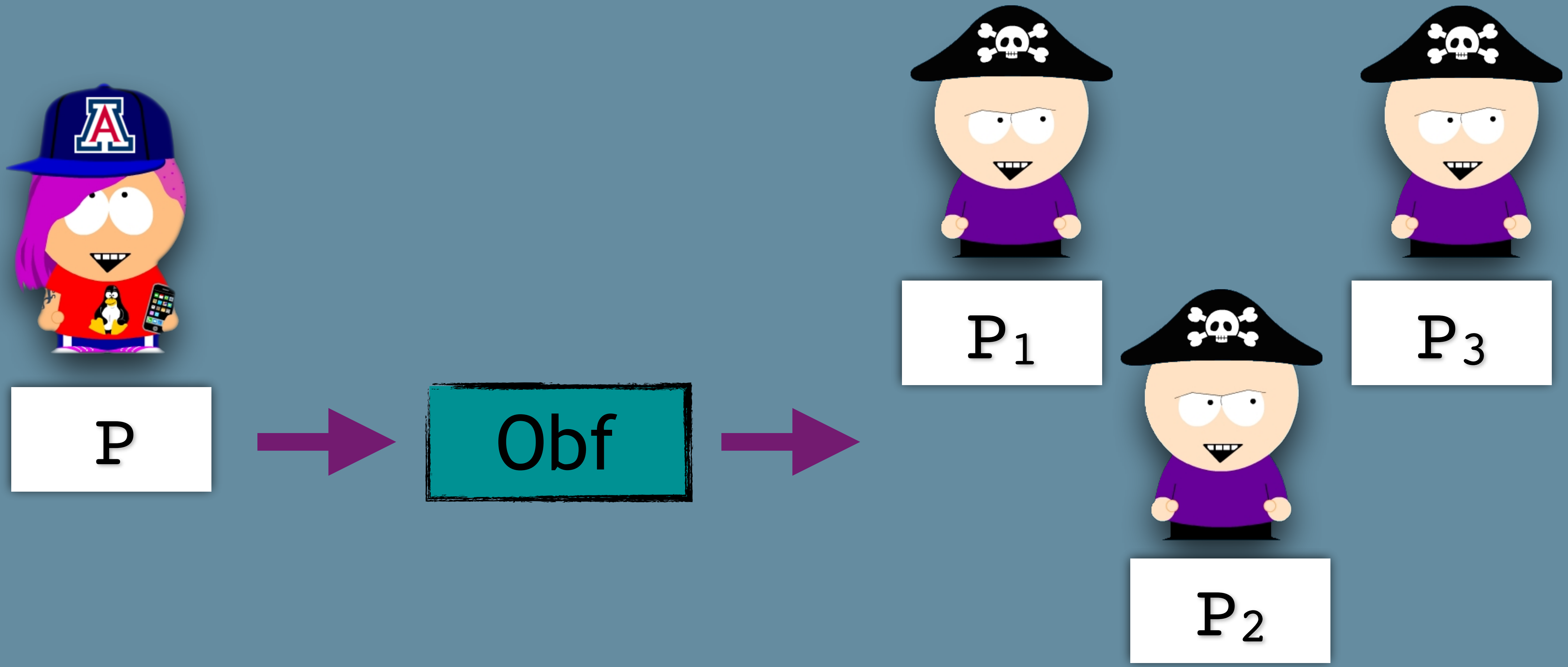
Monitor adversarial communities

Be prepared with new technologies

- Spatial diversity
- Temporal diversity
- Semantic diversity

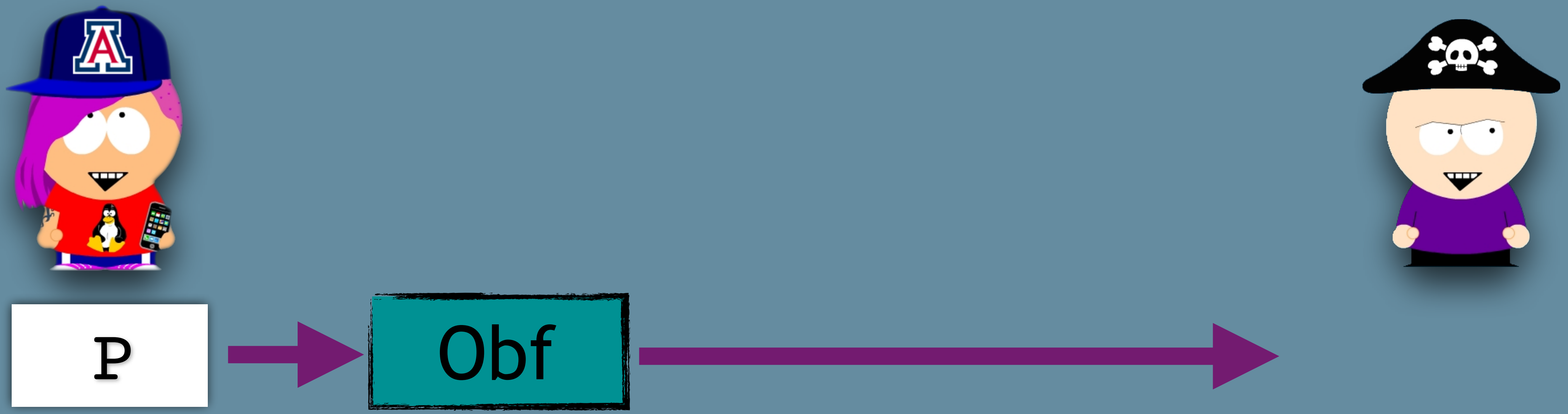
Give adversaries a diversity of targets

Spatial Diversity



- Prevent collusion by giving each adversary a differently obfuscated program

Temporal Diversity



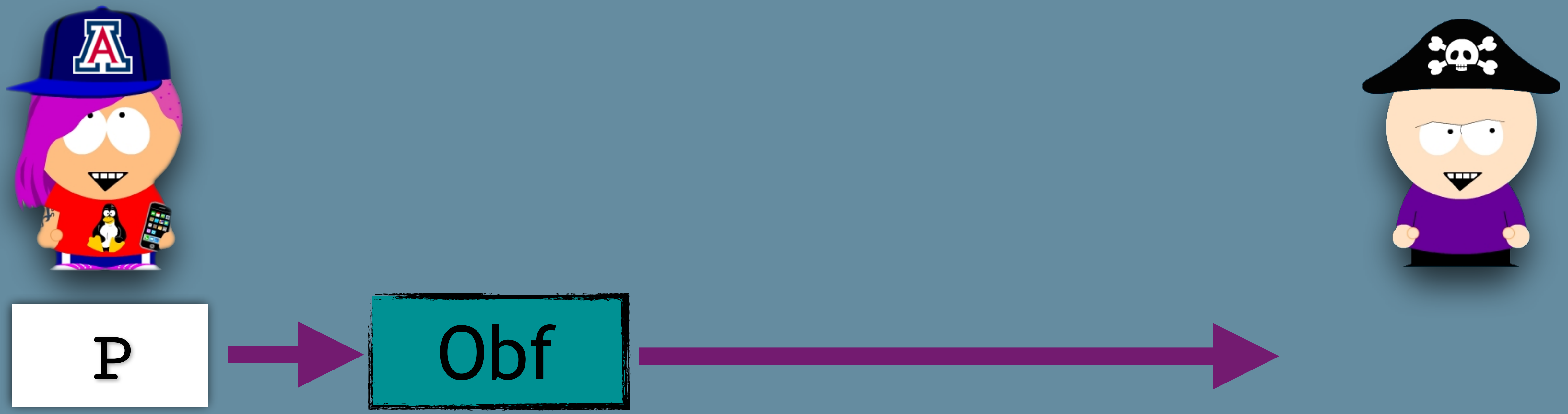
P

Obf

- Adversary sees a sequence of code variants over time
- Overwhelm his analytical abilities
- Small time window to execute an attack
- Known as “*Planned Obsolescence*”

London, Ending the Depression Through Planned Obsolescence, 1932

Temporal Diversity



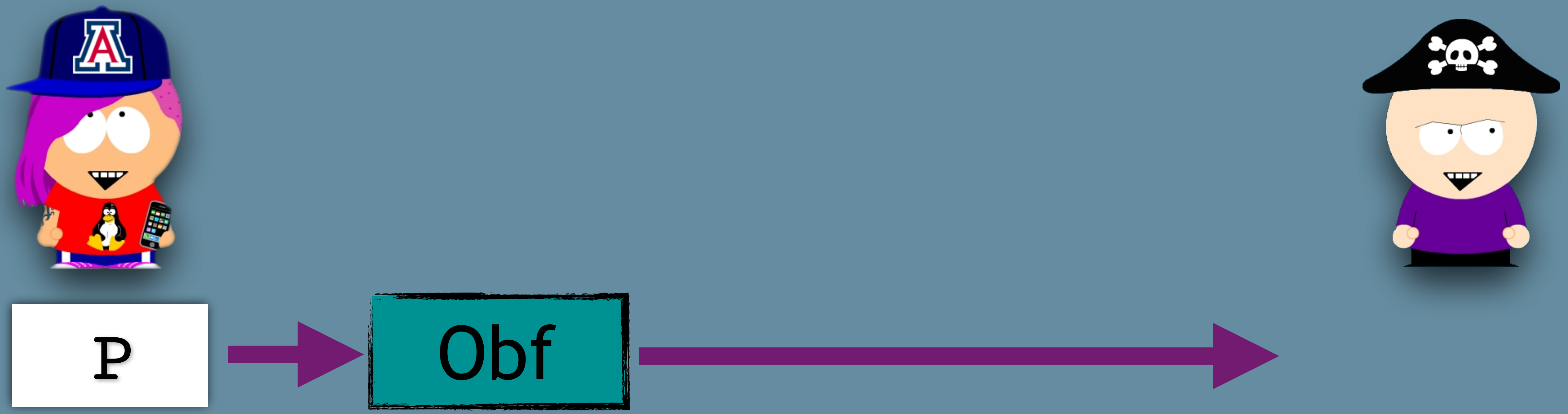
P

Obf

- Adversary sees a sequence of code variants over time
- Overwhelm his analytical abilities
- Small time window to execute an attack
- Known as “*Planned Obsolescence*”

London, Ending the Depression Through Planned Obsolescence, 1932

Semantic Diversity



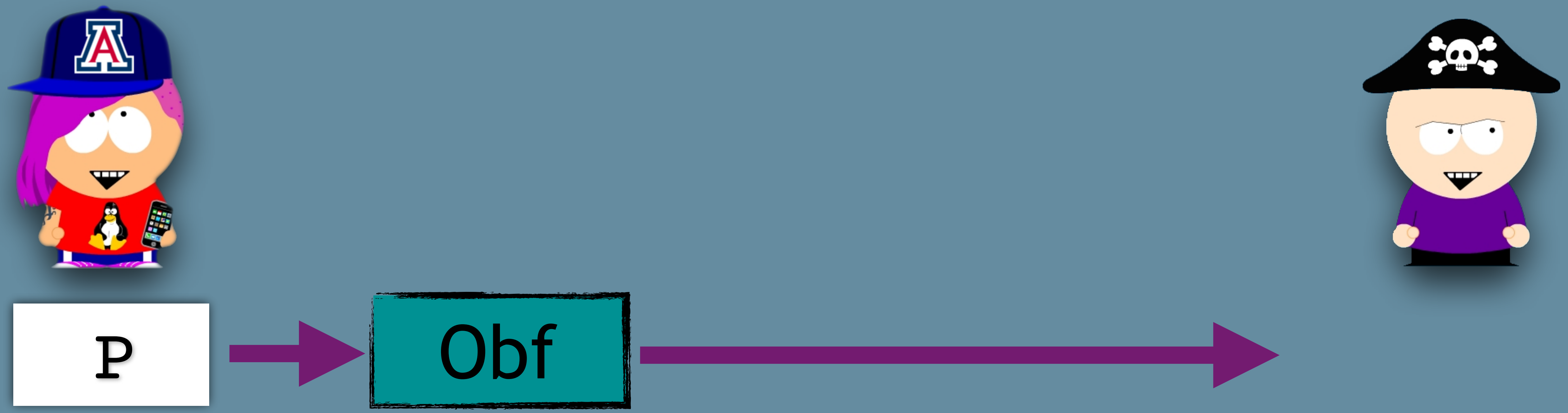
P

Obf

- Code variants are semantically incompatible
- Previously cracked code variants have no value
- Known as "*Software Aging*"

Jakobsson, et al., Discouraging Software Piracy Using Software Aging, S&P in DRM, 2002

Semantic Diversity



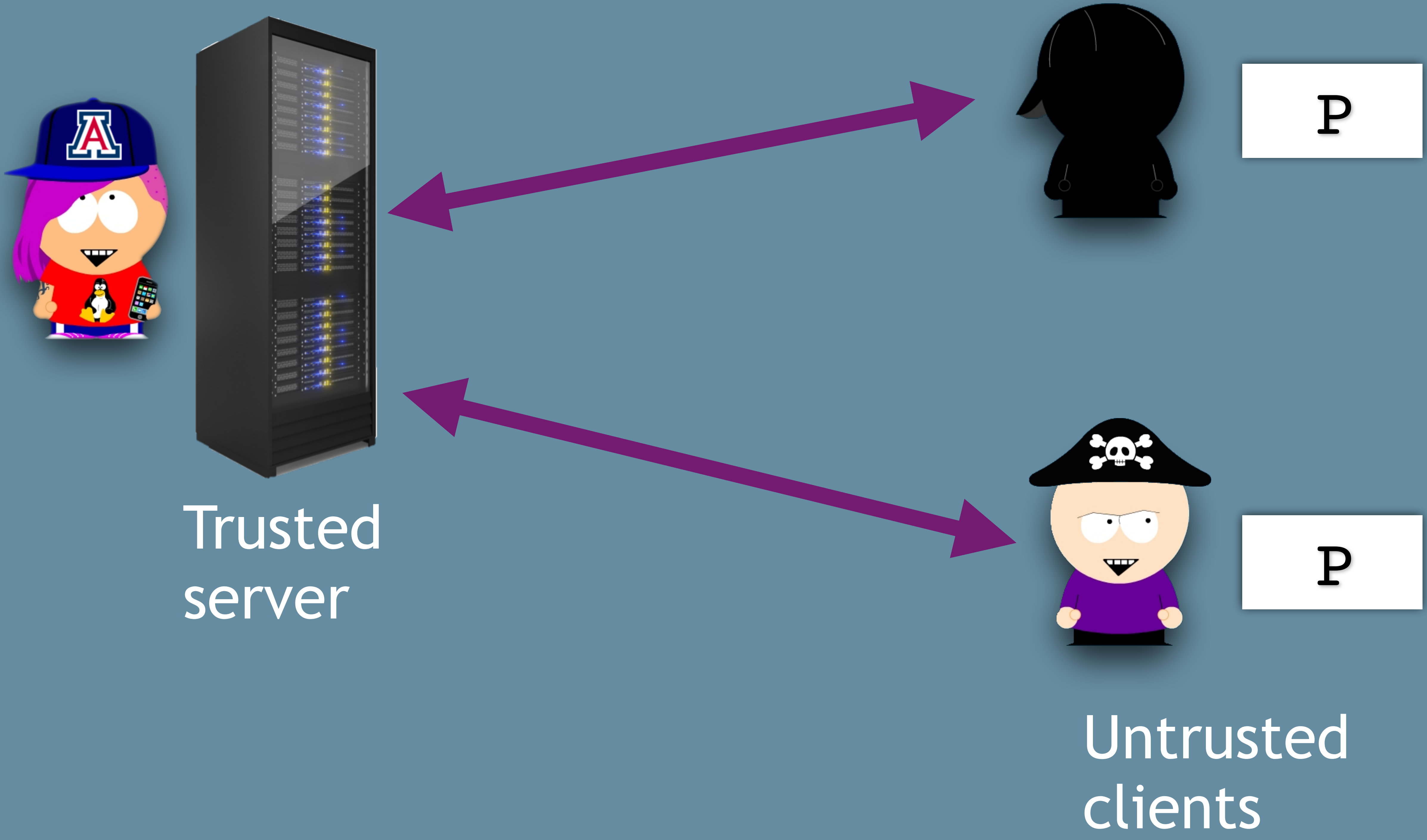
P

Obf

- Code variants are semantically incompatible
- Previously cracked code variants have no value
- Known as "*Software Aging*"

Jakobsson, et al., Discouraging Software Piracy Using Software Aging, S&P in DRM, 2002

Updatable Security



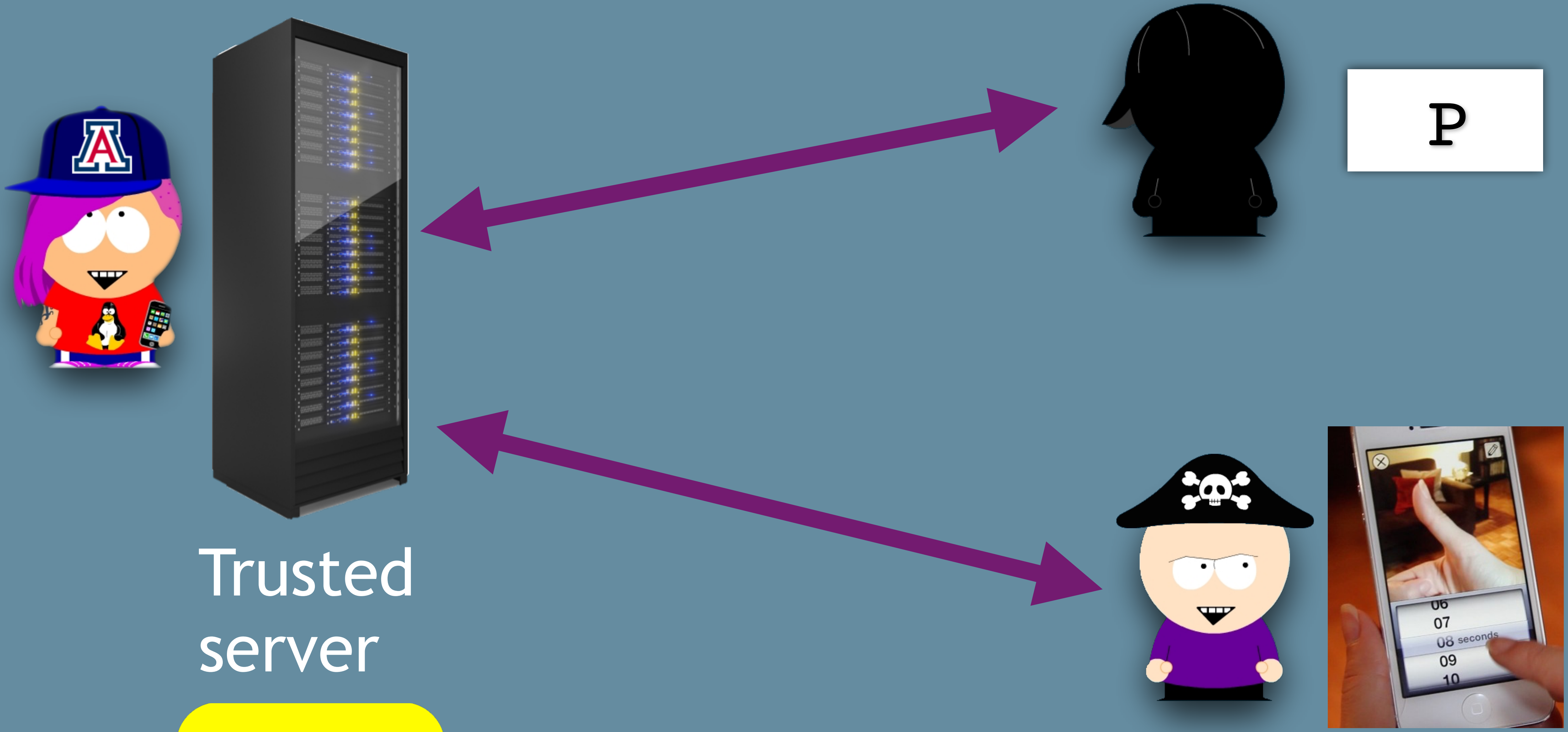
P

P

Trusted server

Untrusted clients

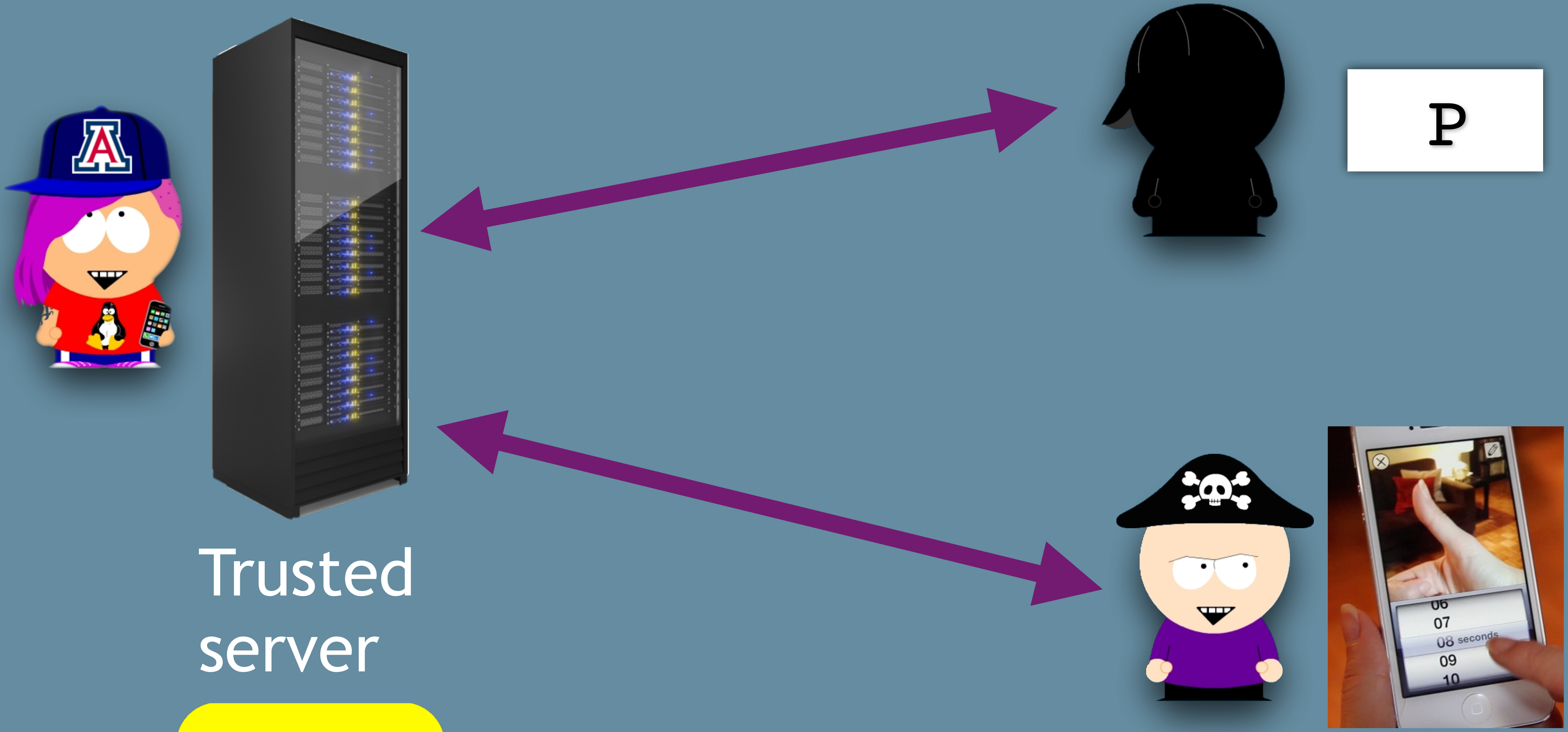
Updatable Security



Trusted server

Untrusted clients

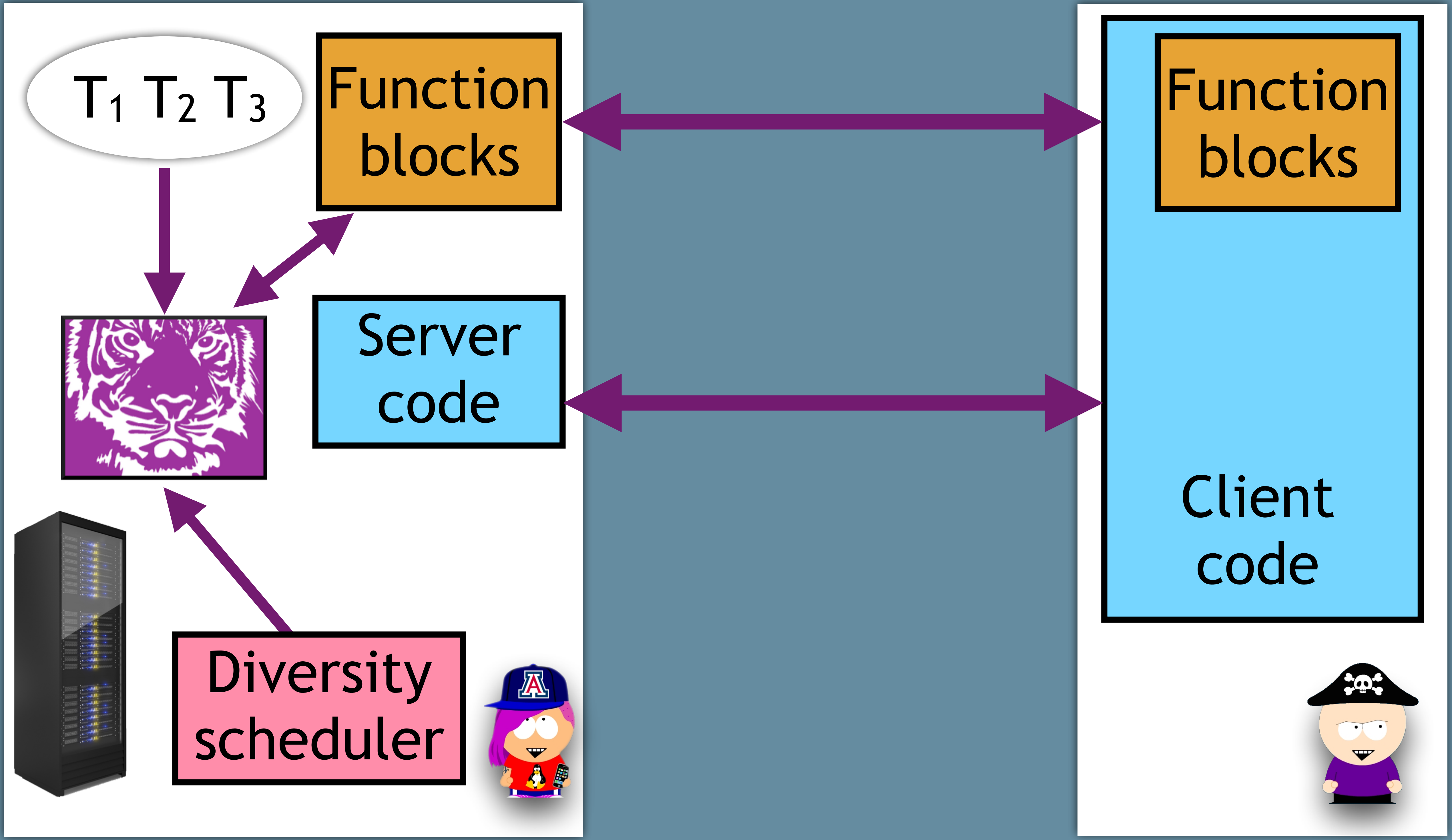
Updatable Security



Trusted server

Untrusted clients

Continuous Replacement



T_1 T_2 T_3

Function blocks

Server code

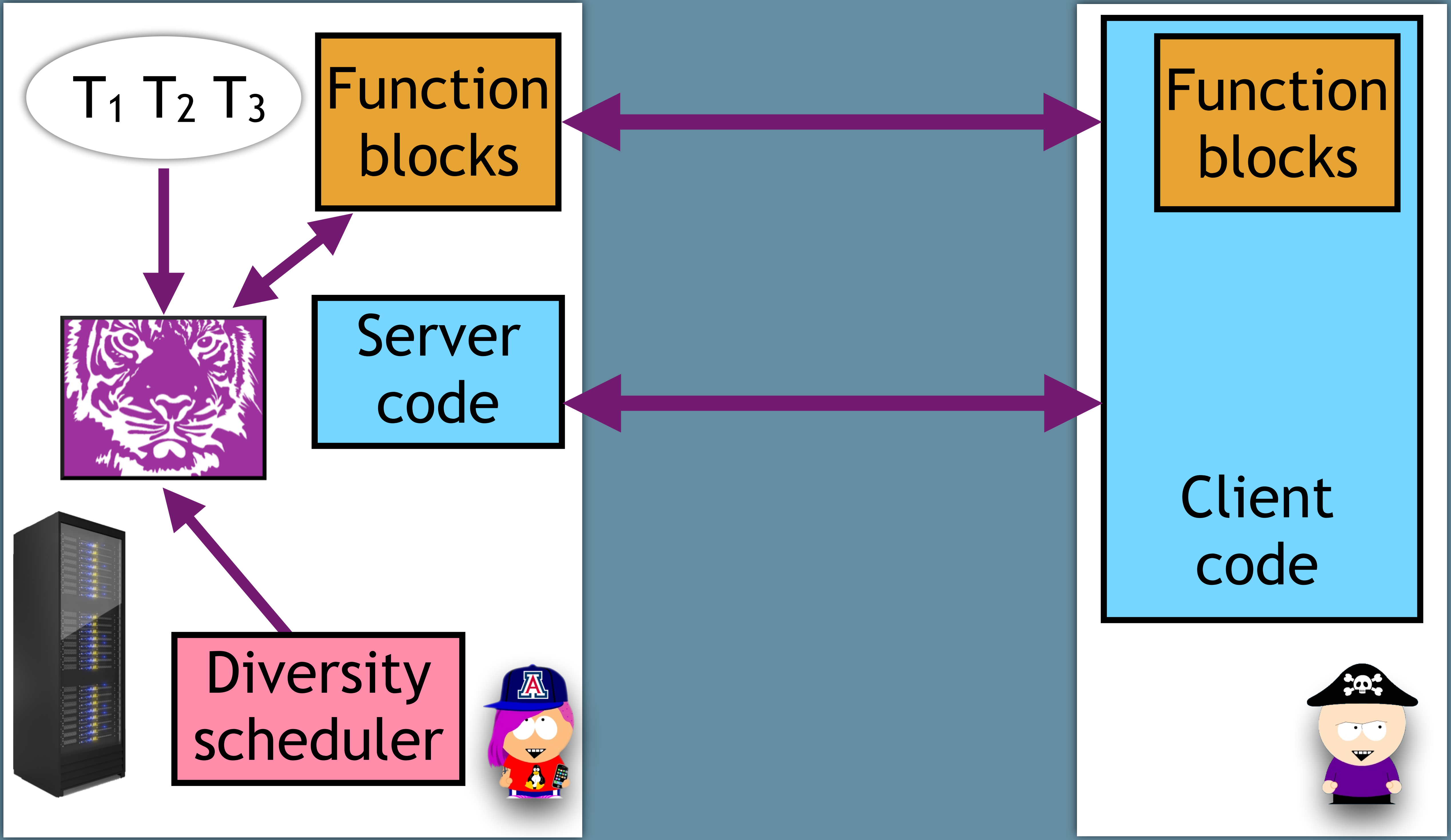
Diversity scheduler

Function blocks

Client code

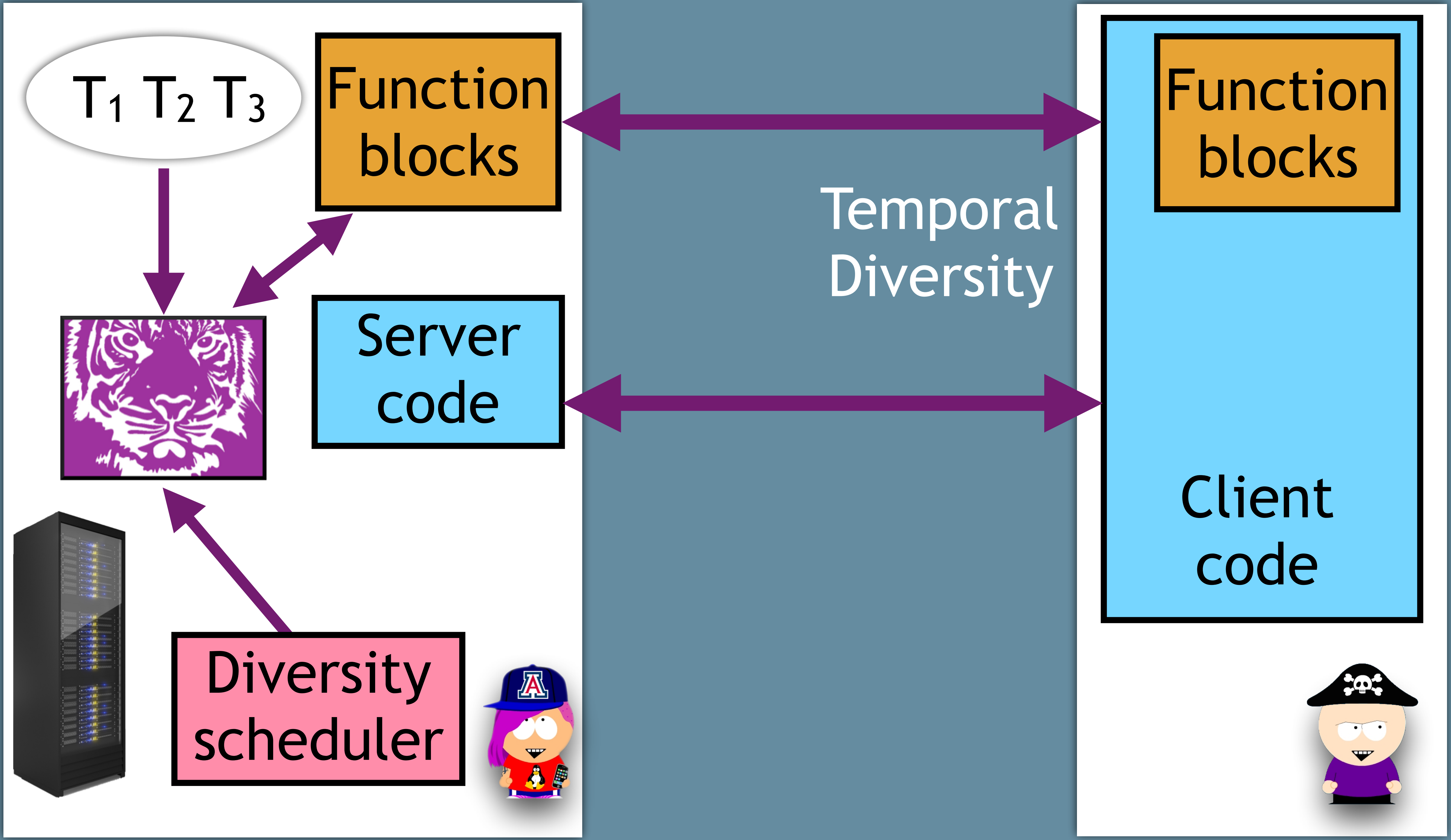
Collberg, et al., Distr. app. tamper detection via continuous softw. updates, ACSAC'12

Continuous Replacement



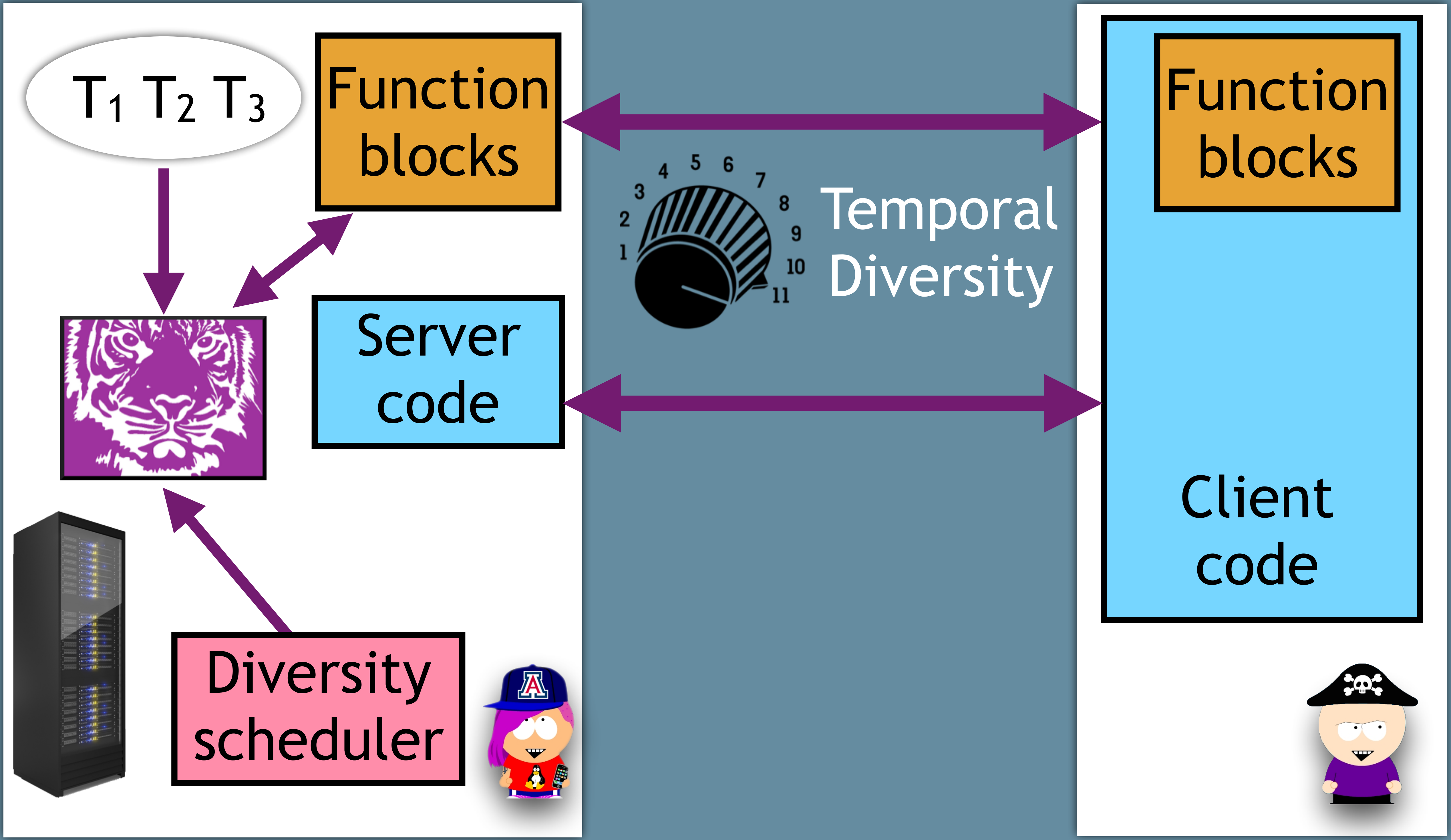
Collberg, et al., Distr. app. tamper detection via continuous softw. updates, ACSAC’12

Continuous Replacement



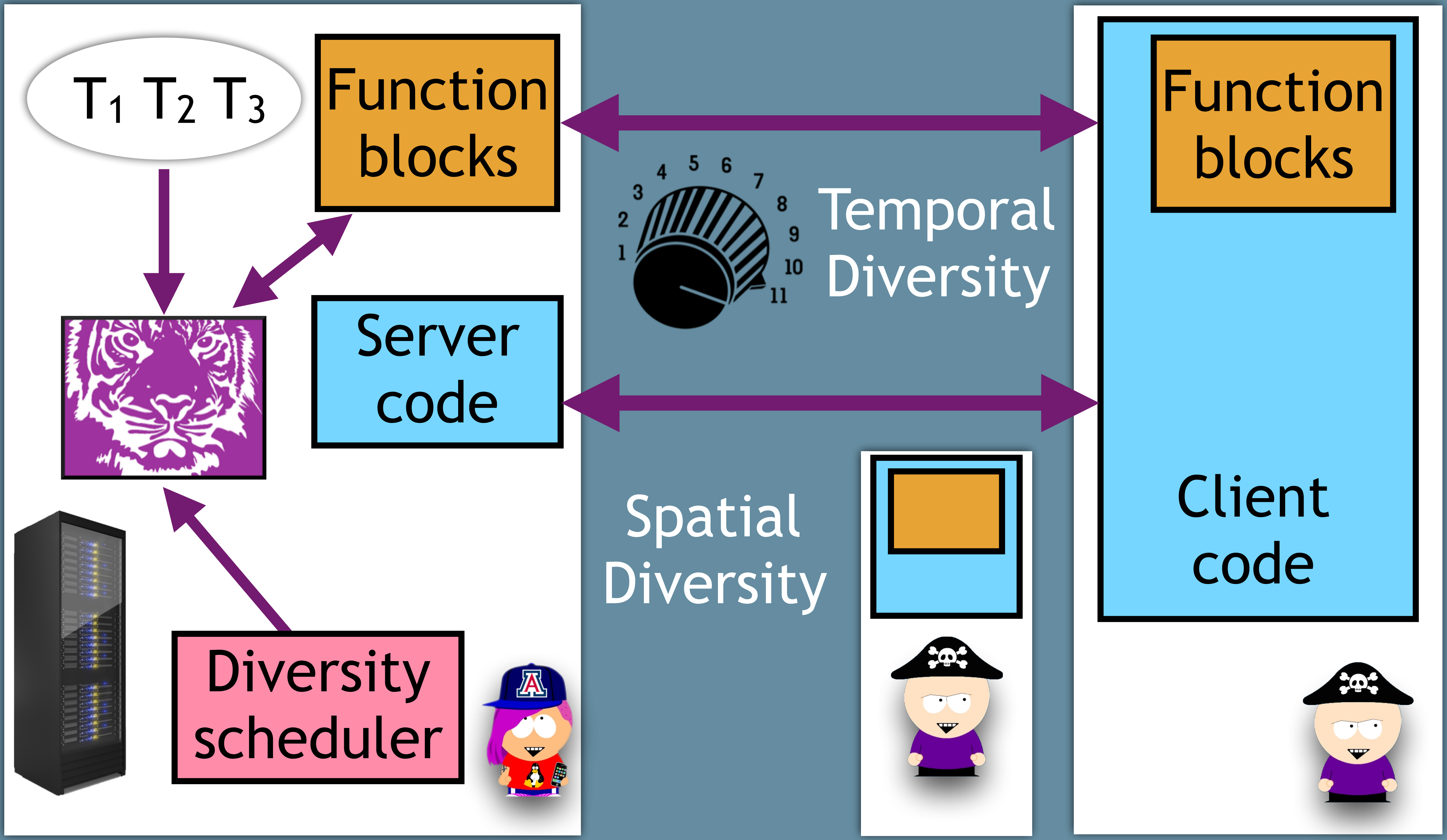
Collberg, et al., Distr. app. tamper detection via continuous softw. updates, ACSAC'12

Continuous Replacement



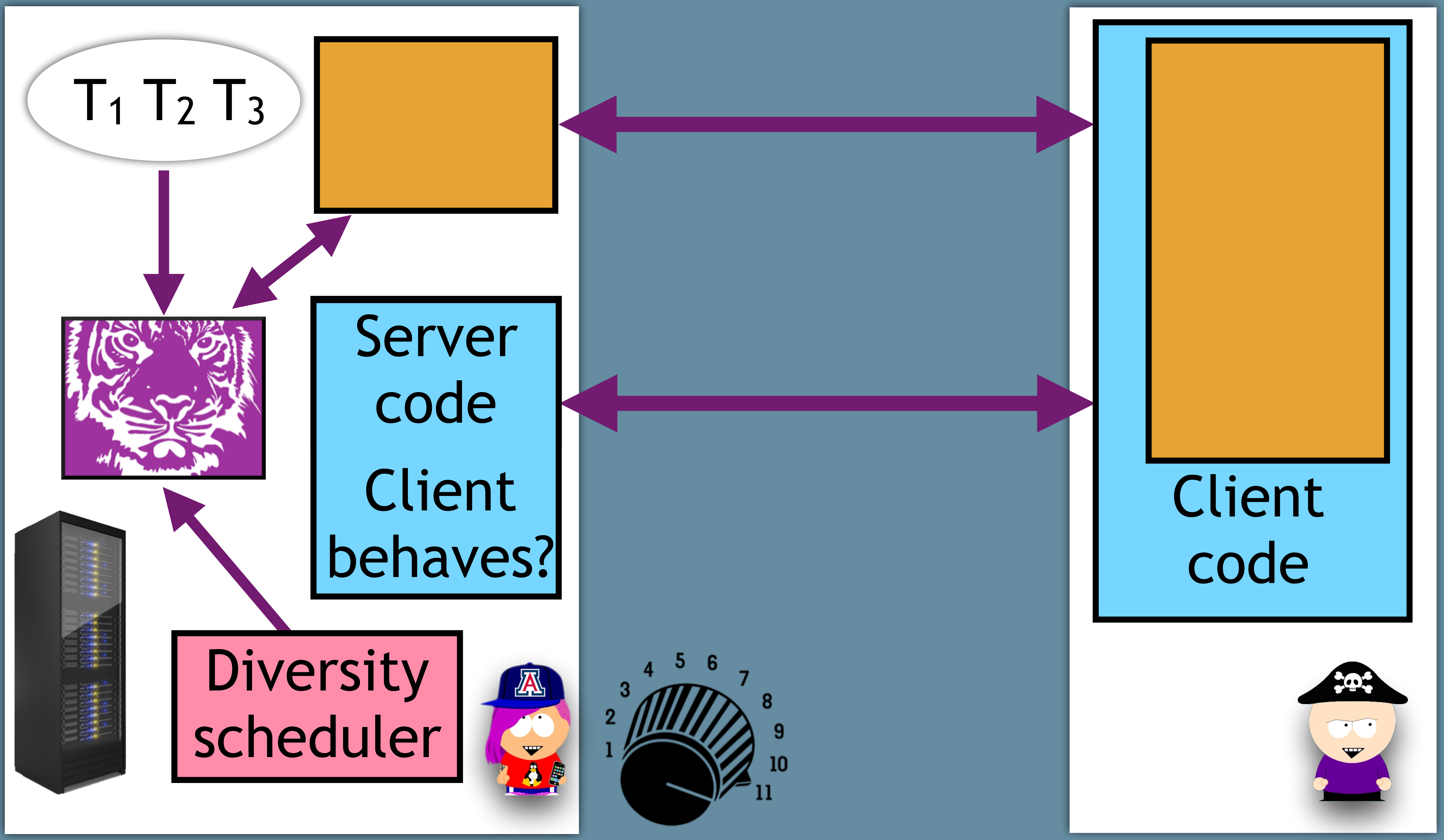
Collberg, et al., Distr. app. tamper detection via continuous softw. updates, ACSAC'12

Continuous Replacement



Collberg, et al., Distr. app. tamper detection via continuous softw. updates, ACSAC'12

Continuous Replacement



T_1 T_2 T_3

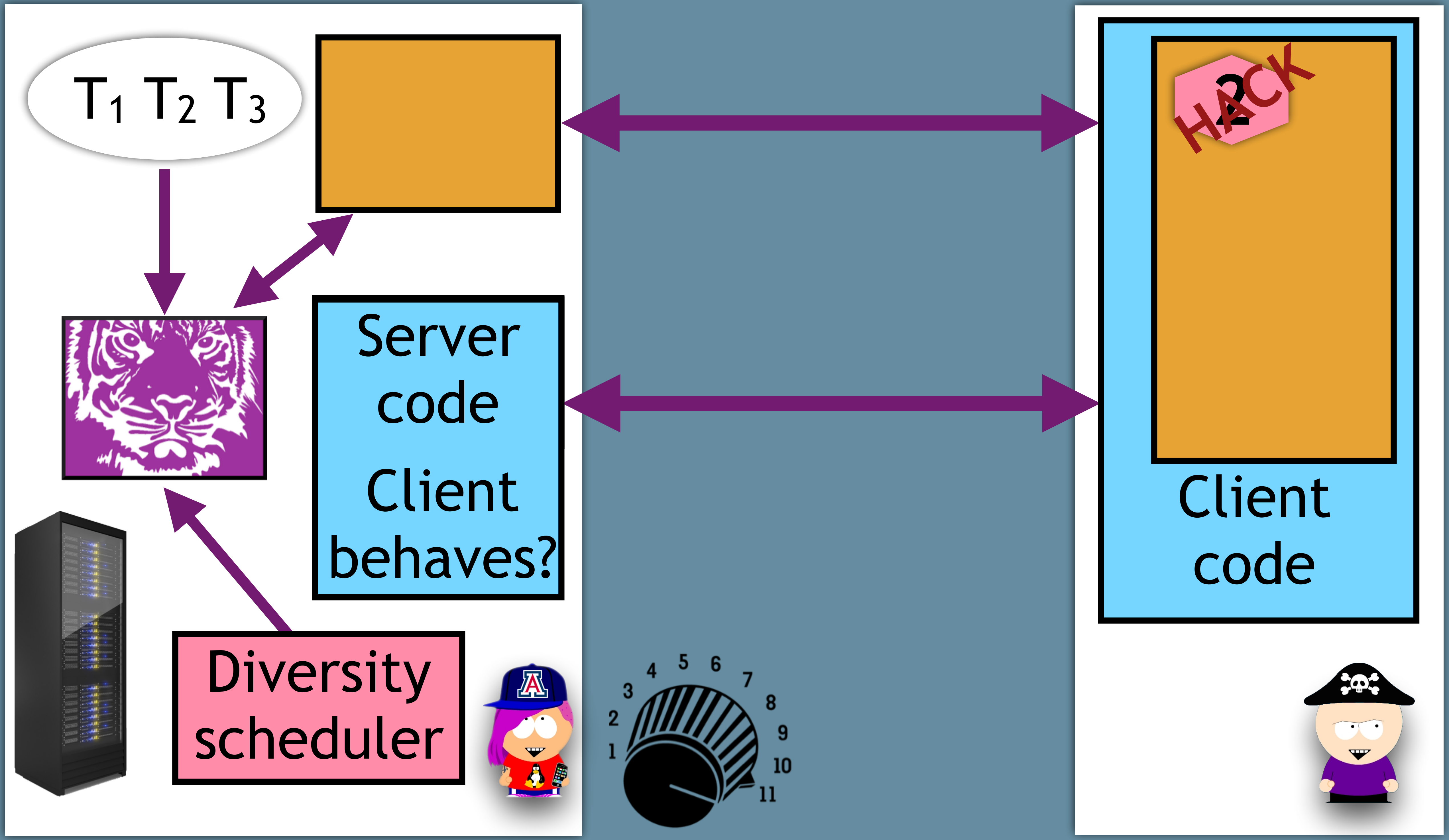
Server code

Client behaves?

Diversity scheduler

Client code

Continuous Replacement



T_1 T_2 T_3

Server code

Client behaves?

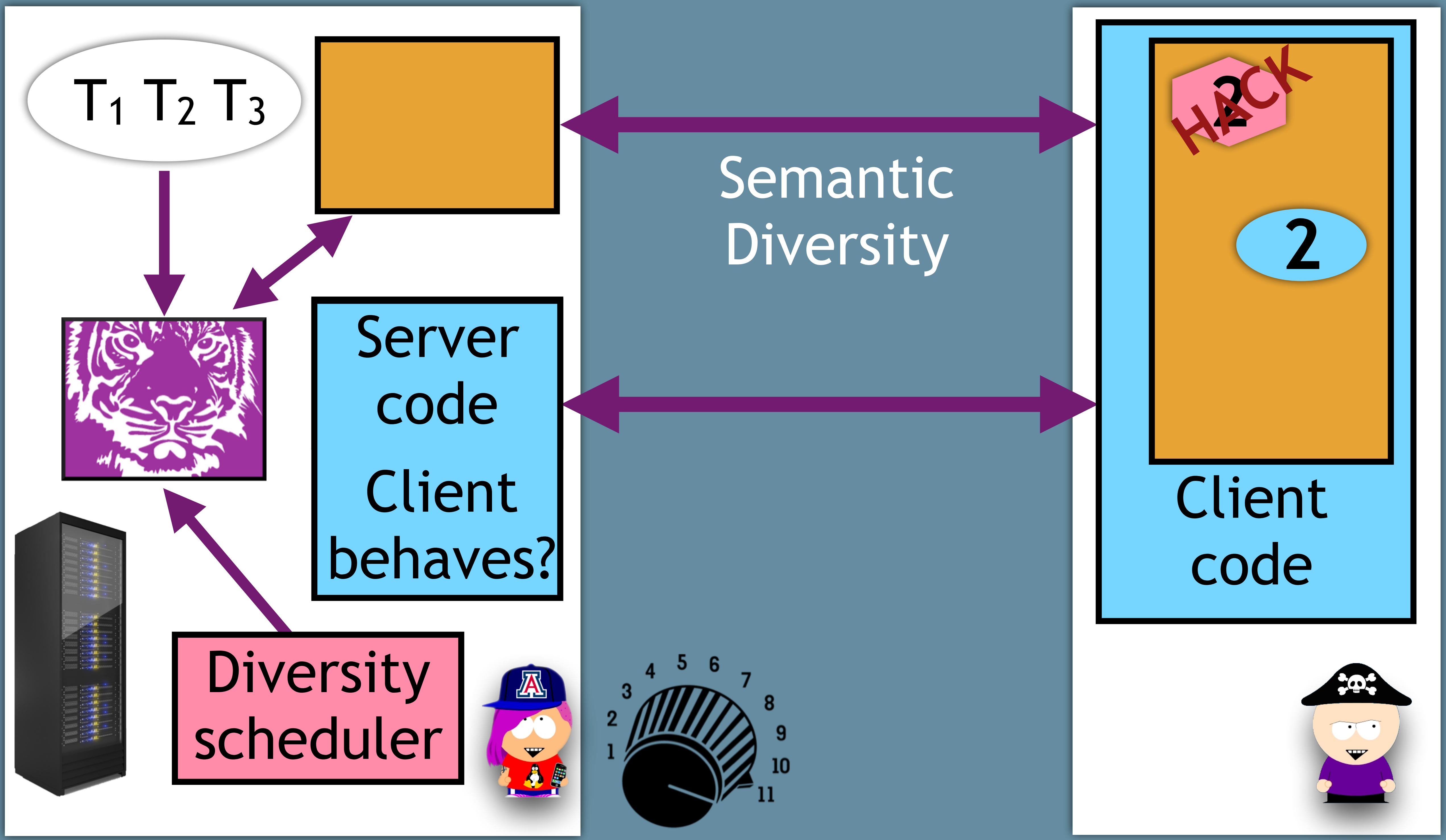
Diversity scheduler

2

HACK

Client code

Continuous Replacement



T_1 T_2 T_3

Server code

Client behaves?

Diversity scheduler

Semantic Diversity

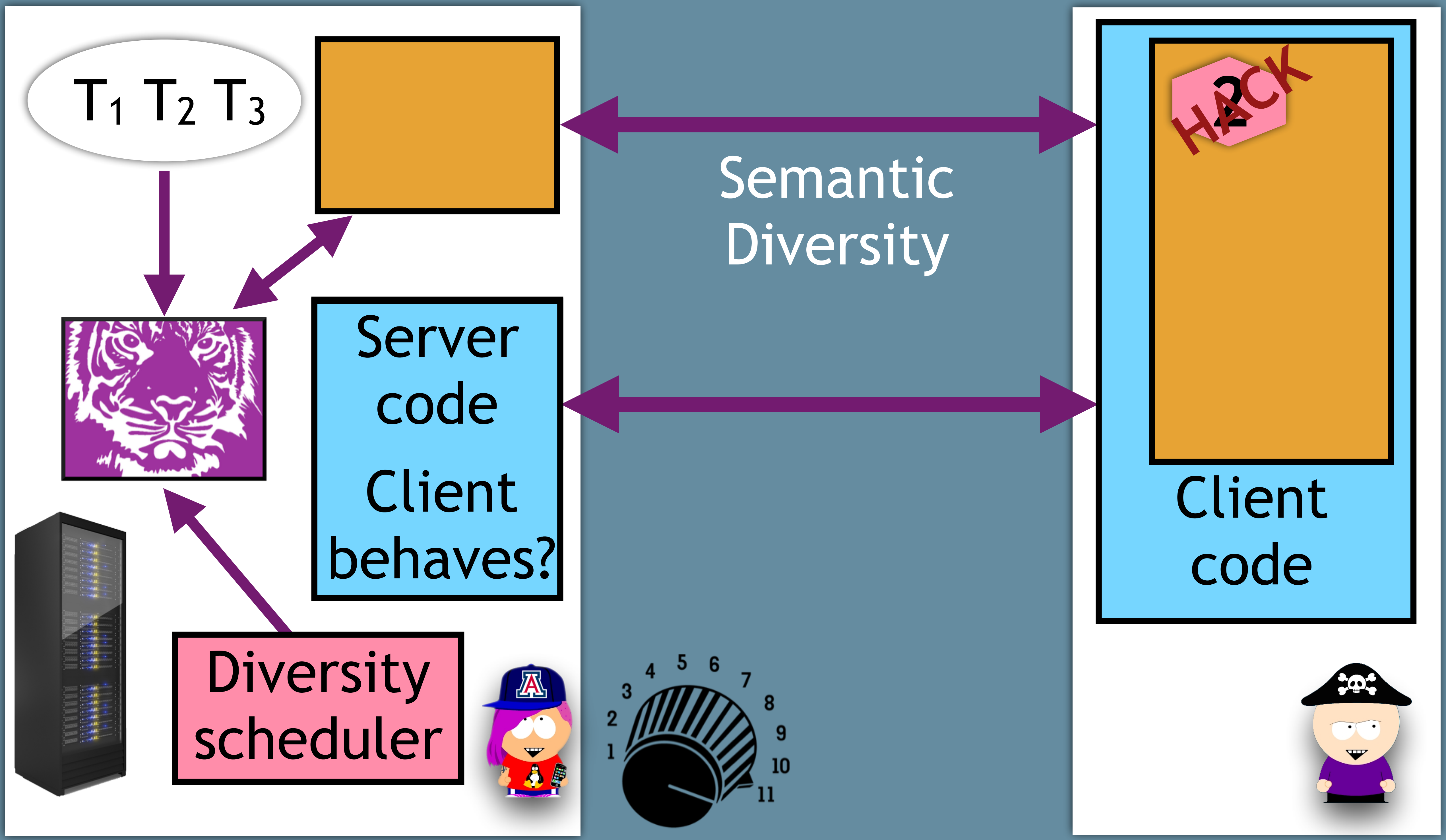
2

HACK

2

Client code

Continuous Replacement



T_1 T_2 T_3

Server code

Client behaves?

Diversity scheduler

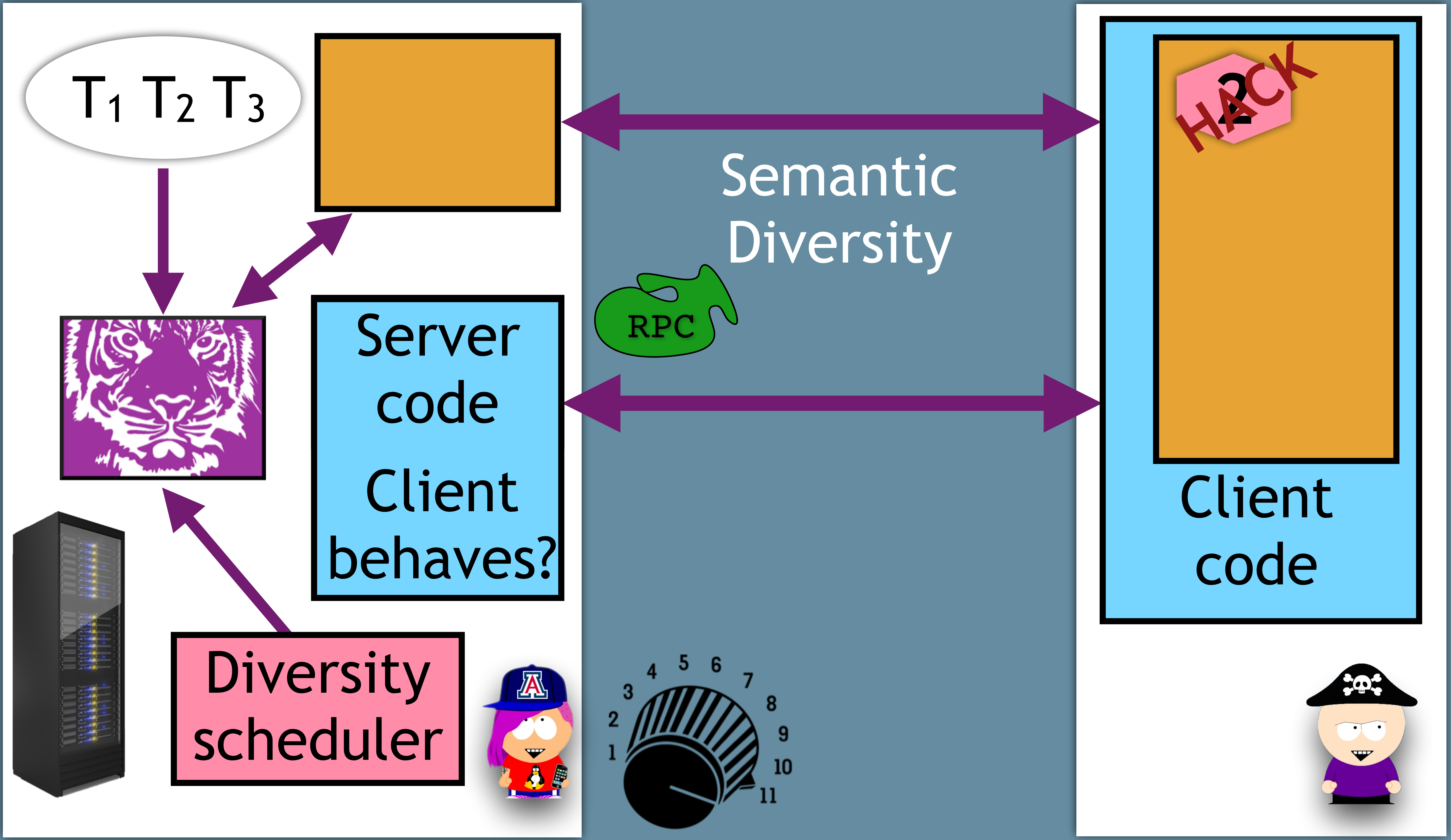
Semantic Diversity

2

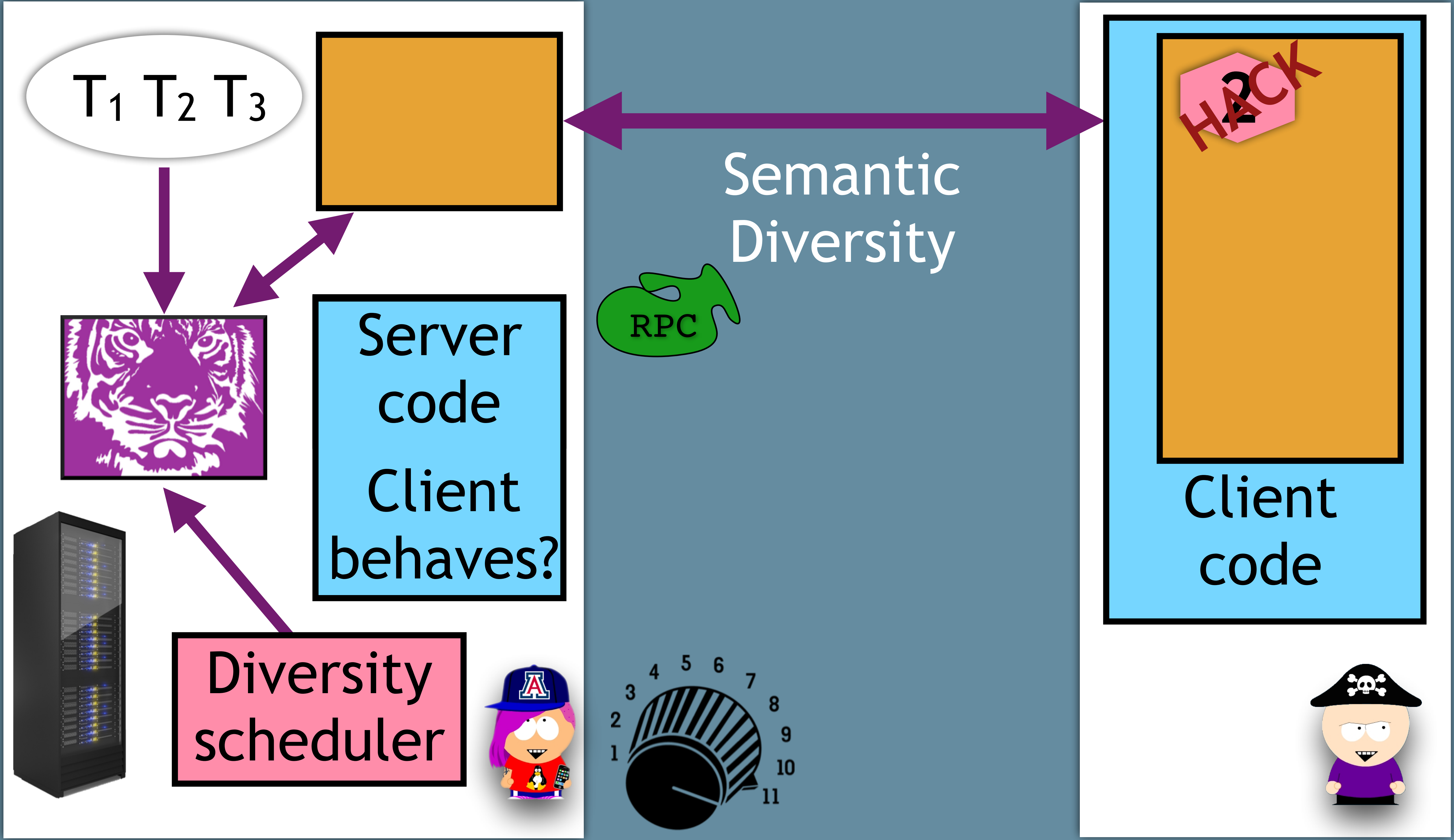
HACK

Client code

Continuous Replacement



Continuous Replacement



T_1 T_2 T_3

Server code

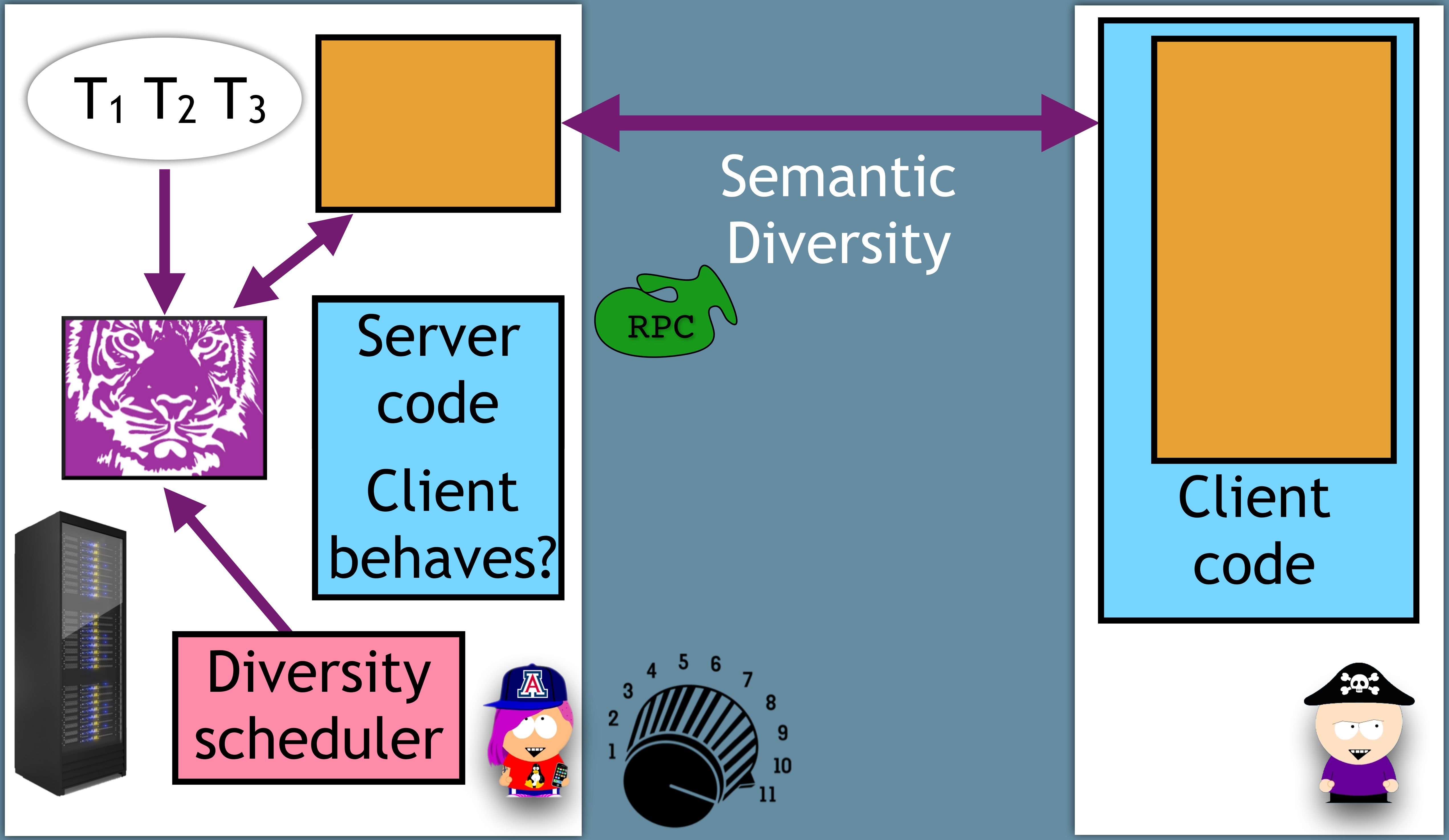
Client behaves?

Diversity scheduler

Semantic Diversity

Client code

Continuous Replacement



T_1 T_2 T_3

Semantic Diversity

Server code

Client behaves?

Diversity scheduler

Client code

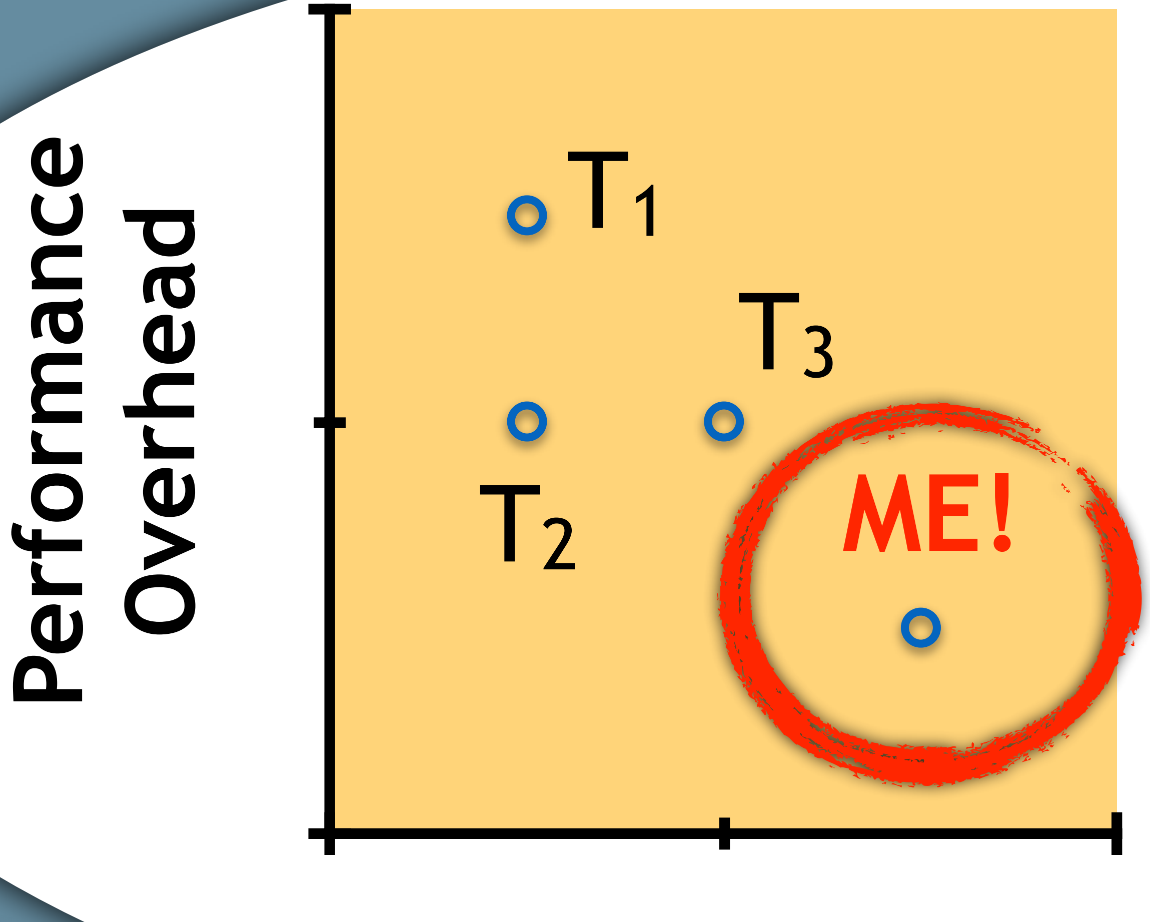
Our Story So Far…

1. Scenarios where obfuscation can be useful
2. Obfuscating transformations that give time-limited protection
3. Updatable security for longer-term protection

But, how do we know we’re doing anything good?

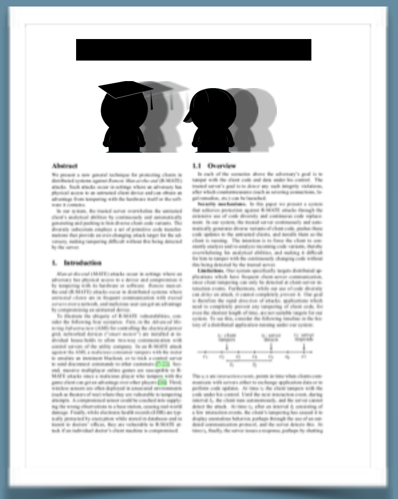
Evaluation

Evaluation in Academia

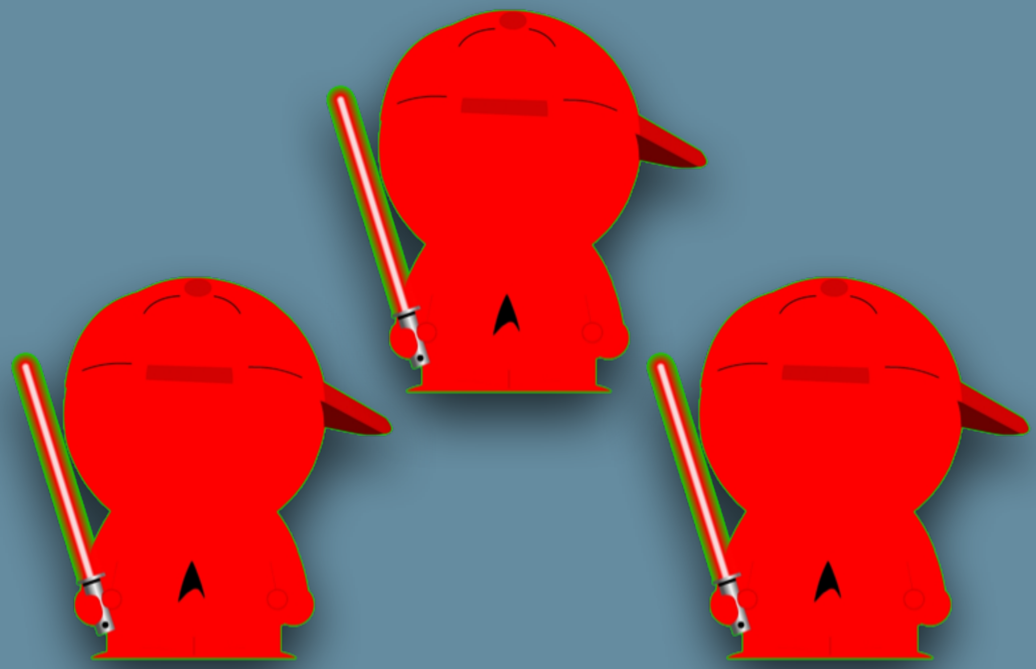


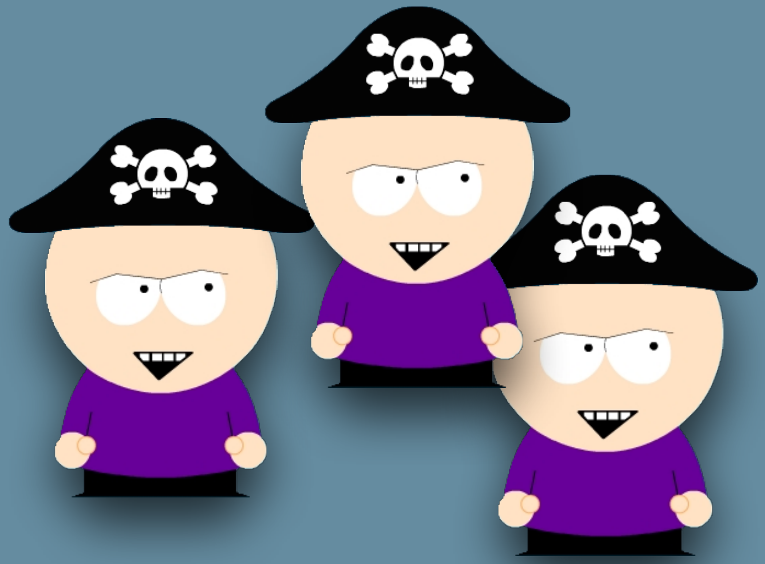
My transformation gets better security and performance than previous ones!

If we can’t do this, how can we make progress?



Evaluation in Industry



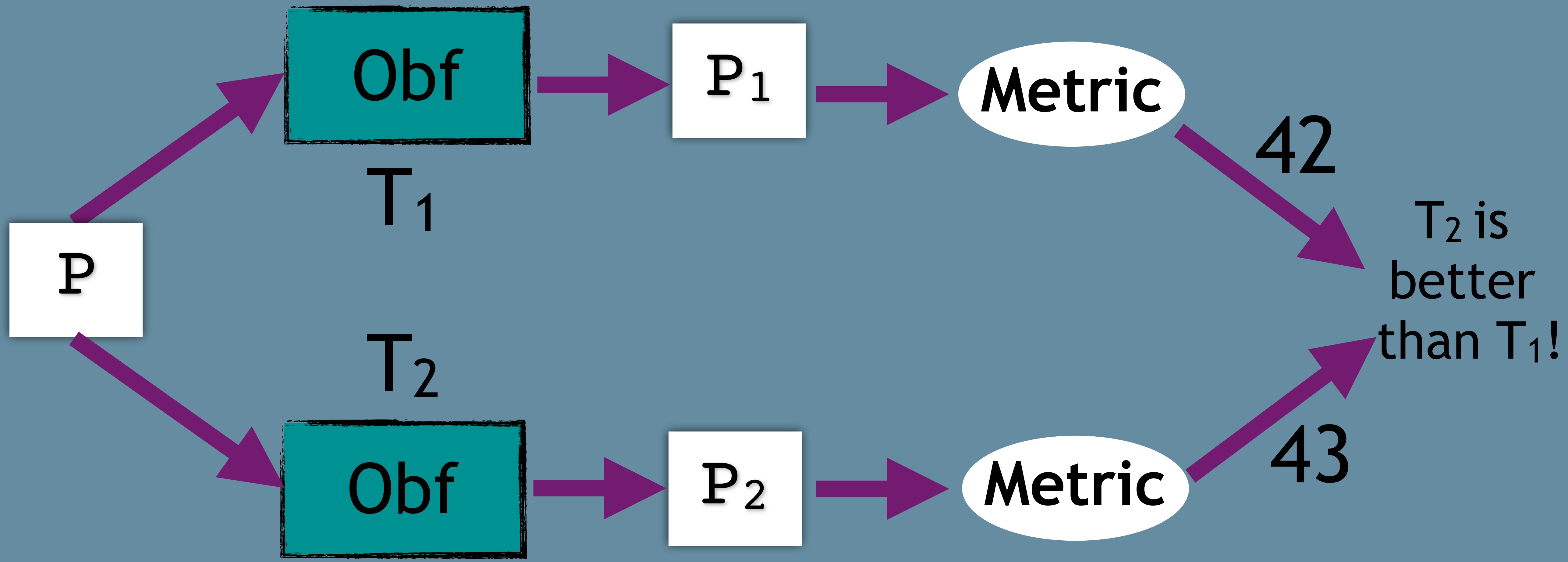


Transformation	Status
T_1	Broken in ‘09
T_2	Soon to be broken
T_3	Works for now

Professional red teams evaluate new transformations

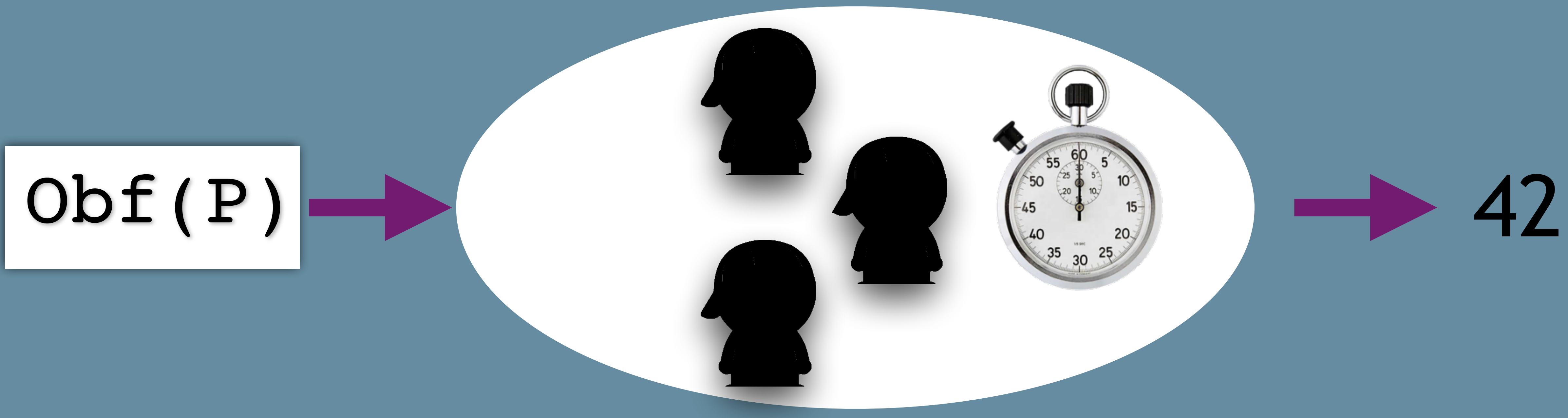
Experience from monitoring real world adversaries

Programmatic Evaluation



- Invent “stand-ins” for red team evaluation
- Which metrics should we use?

Metric 1: Students



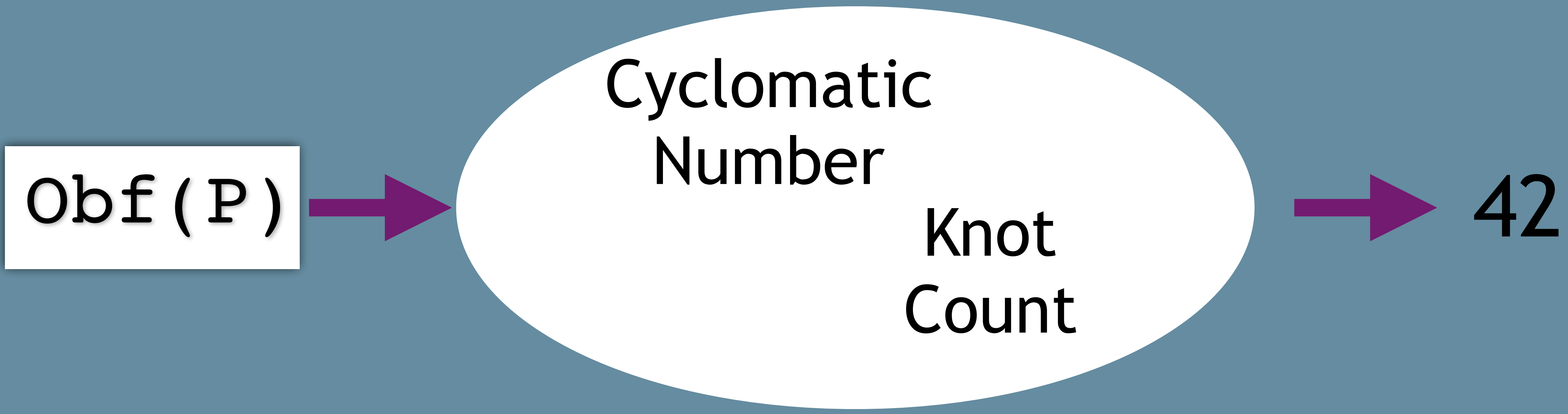
`Obf(P)`

42

- Measure the time it takes for students to solve a task on the obfuscated code
- **Issues**: Inexperience, doesn’t scale, students get better over time

Ceccato et al., The effectiveness of source code obfuscation: …, ICPC’09

Metric 2: SW Metrics



`Obf(P)`

Cyclomatic Number

Knot Count

42

- Combine a few Software Complexity Metrics
- **Issues**: SCMs were not designed to measure code badness; 100s of SCMs - which ones should we use?

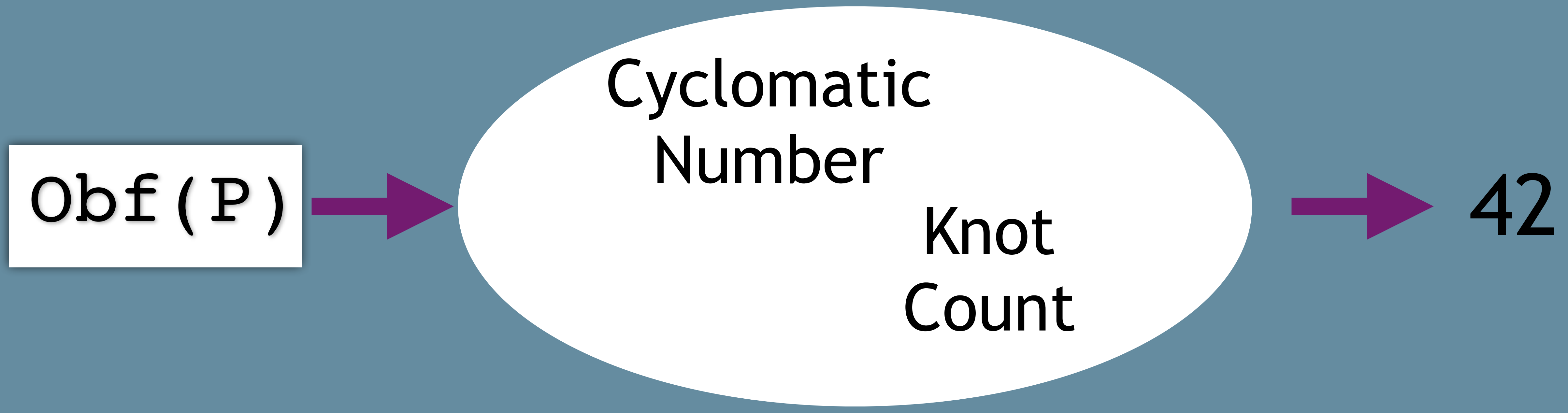
Anckaert, et al., Program Obfuscation: A Quantitative Approach

Metric 2: SW Metrics

Complexity Metric	Definition
Knot Count	Number of crossings of control flow arrows in a graph
Cyclomatic number	Number of decision points: #edges−#nodes+2*(#connected components)

Anckaert, et al., Program Obfuscation: A Quantitative Approach

Metric 2: SW Metrics



`Obf(P)`

Cyclomatic Number

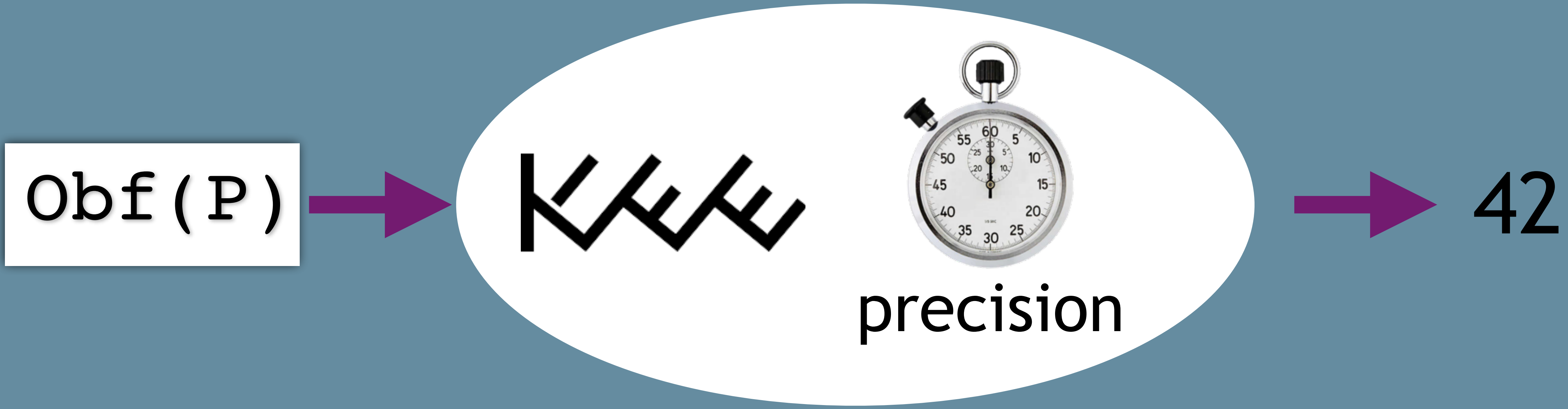
Knot Count

42

- Combine a few Software Complexity Metrics
- **Issues**: SCMs were not designed to measure code badness; 100s of SCMs - which ones should we use?

Anckaert, et al., Program Obfuscation: A Quantitative Approach

Metric 3: Analysis Tools

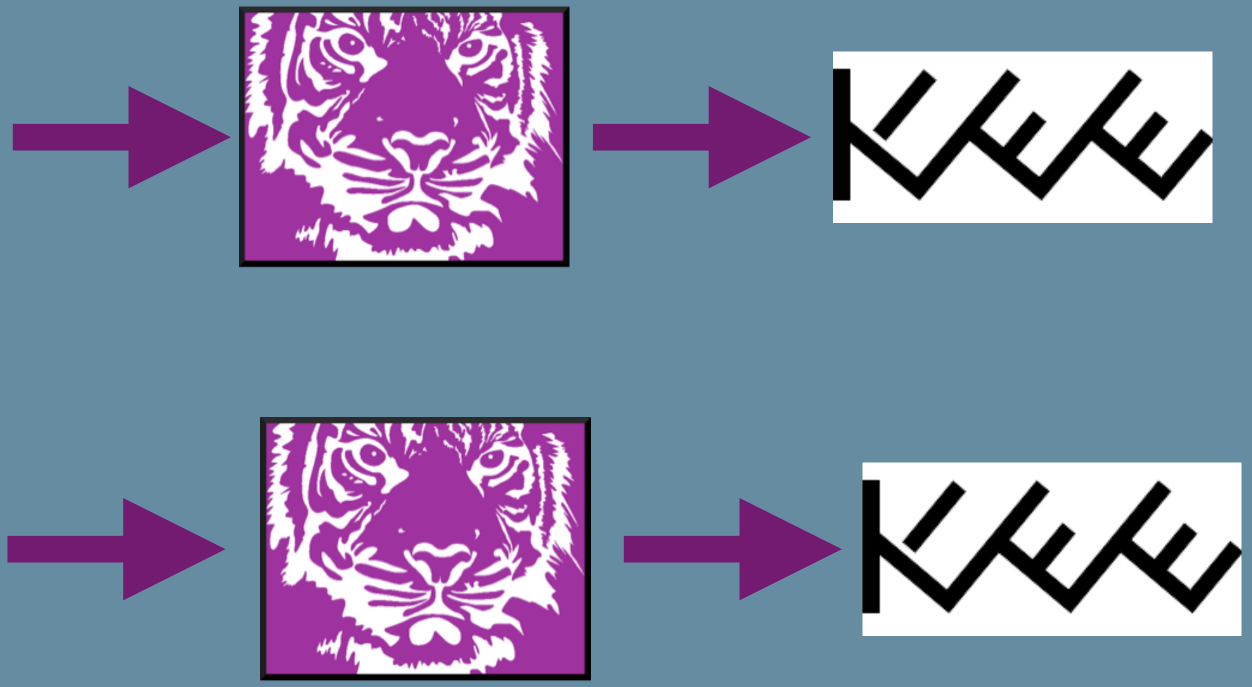


- Measure the runtime & precision of code analysis tool

Banescu, et al., A Framework for Measuring Soft. Obf. Res. Against Automated Attacks

```
int main(int argc,
             char* argv[]) {
   if (argv[1][0] == 97 &&
         argv[1][1] == 98 &&
         argv[1][2] == 99 &&
         argv[1][3] == 100 &&
         argv[1][4] == 101) {
      printf("win\n");
   } else {
      printf("lose\n");
   }
}
```

Virtualize



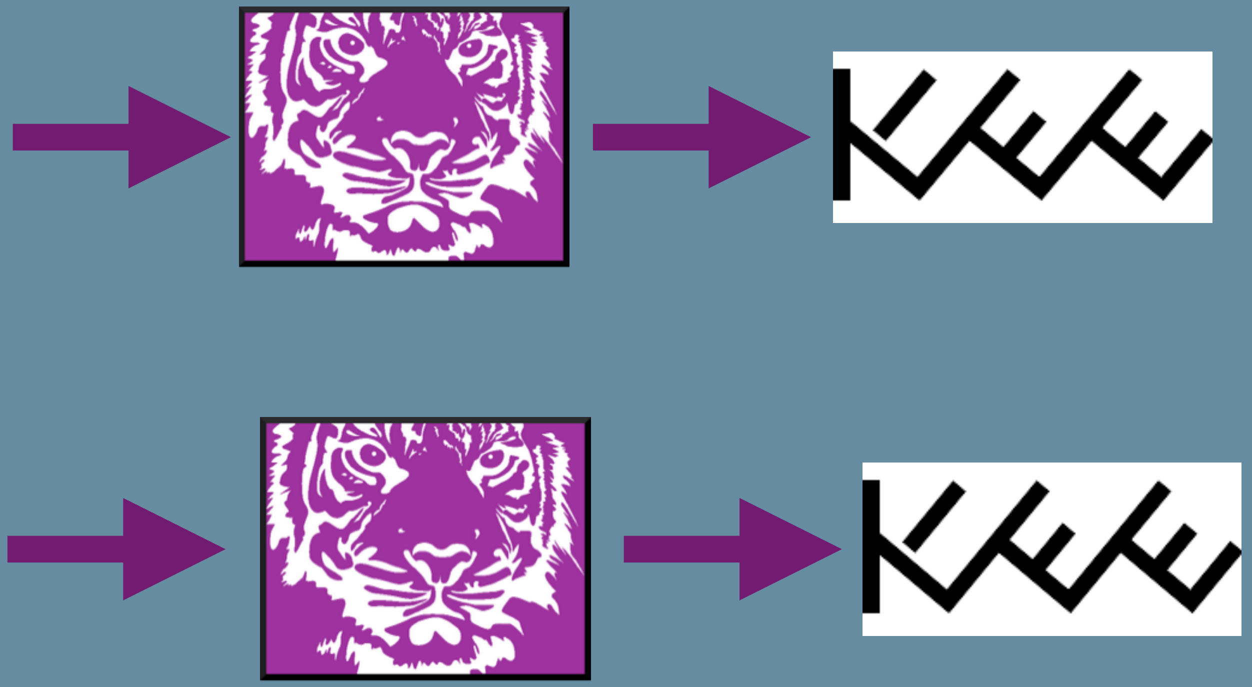
0.5sec

Virtualize +
Encode Program Array +
Make Input Dependent

- Failure due to bugs, lack of performance tuning, or your transformation is good, …

```
int main(int argc,
             char* argv[]) {
   if (argv[1][0] == 97 &&
         argv[1][1] == 98 &&
         argv[1][2] == 99 &&
         argv[1][3] == 100 &&
         argv[1][4] == 101) {
      printf("win\n");
   } else {
      printf("lose\n");
   }
}
```

Virtualize



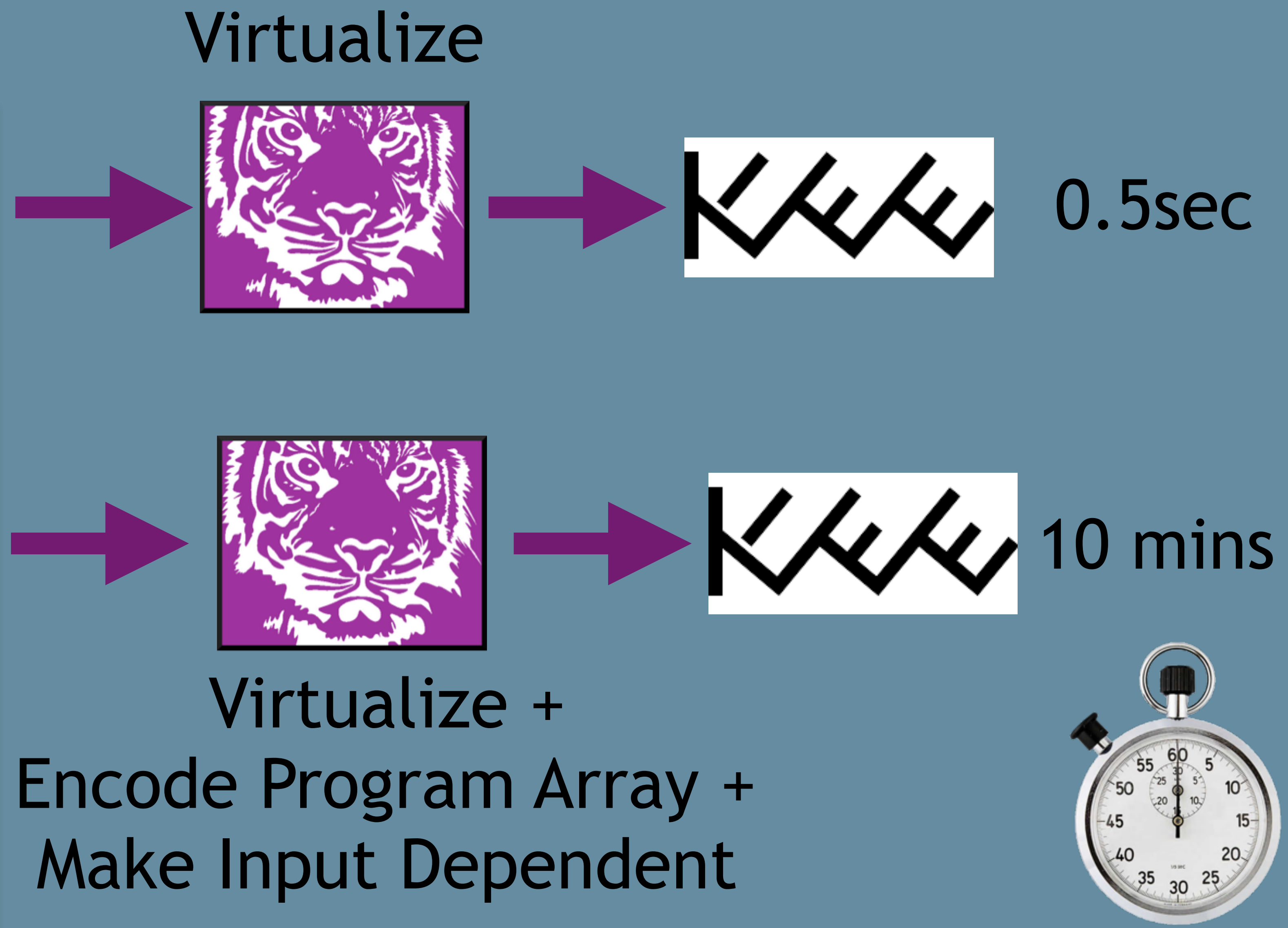
0.5sec

0.5s

Virtualize +
Encode Program Array +
Make Input Dependent

- Failure due to bugs, lack of performance tuning, or your transformation is good, …

```
int main(int argc,
             char* argv[]) {
   if (argv[1][0] == 97 &&
         argv[1][1] == 98 &&
         argv[1][2] == 99 &&
         argv[1][3] == 100 &&
         argv[1][4] == 101) {
      printf("win\n");
   } else {
      printf("lose\n");
   }
}
```



Virtualize

0.5sec

10 mins

Virtualize +
Encode Program Array +
Make Input Dependent

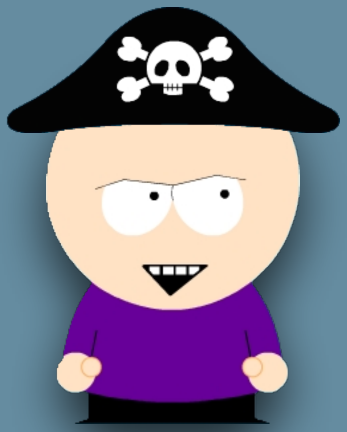
- Failure due to bugs, lack of performance tuning, or your transformation is good, …

Missing: Validation

1. Build model from the behavior of real hackers:

Adversarial Model

- X is hard
- Y is easy

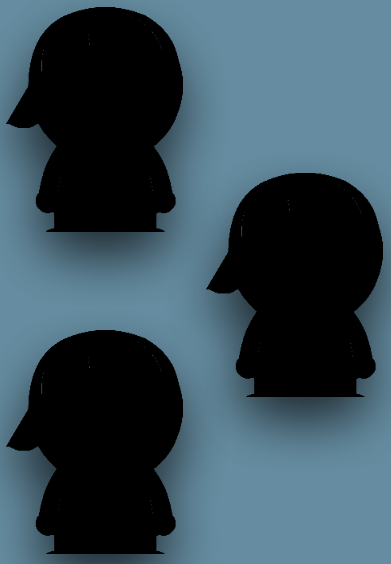


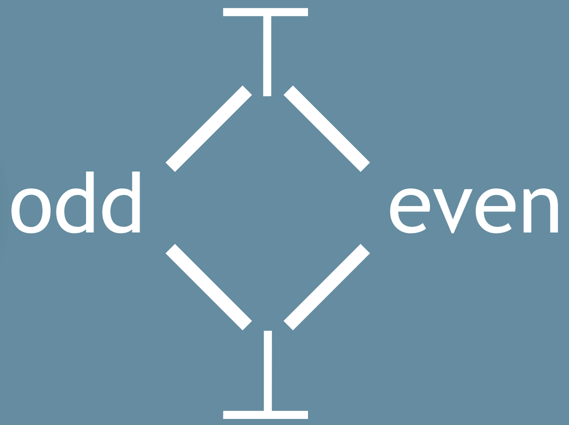
2. Correlate with potential metrics:

SCM_1

SCM_2

SCM_3



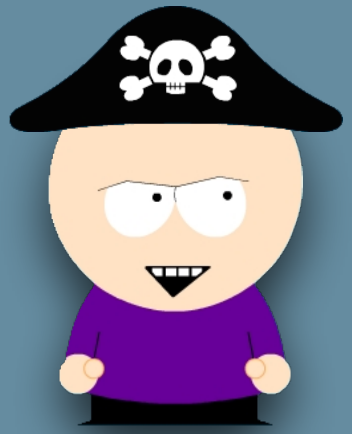


S²E angr

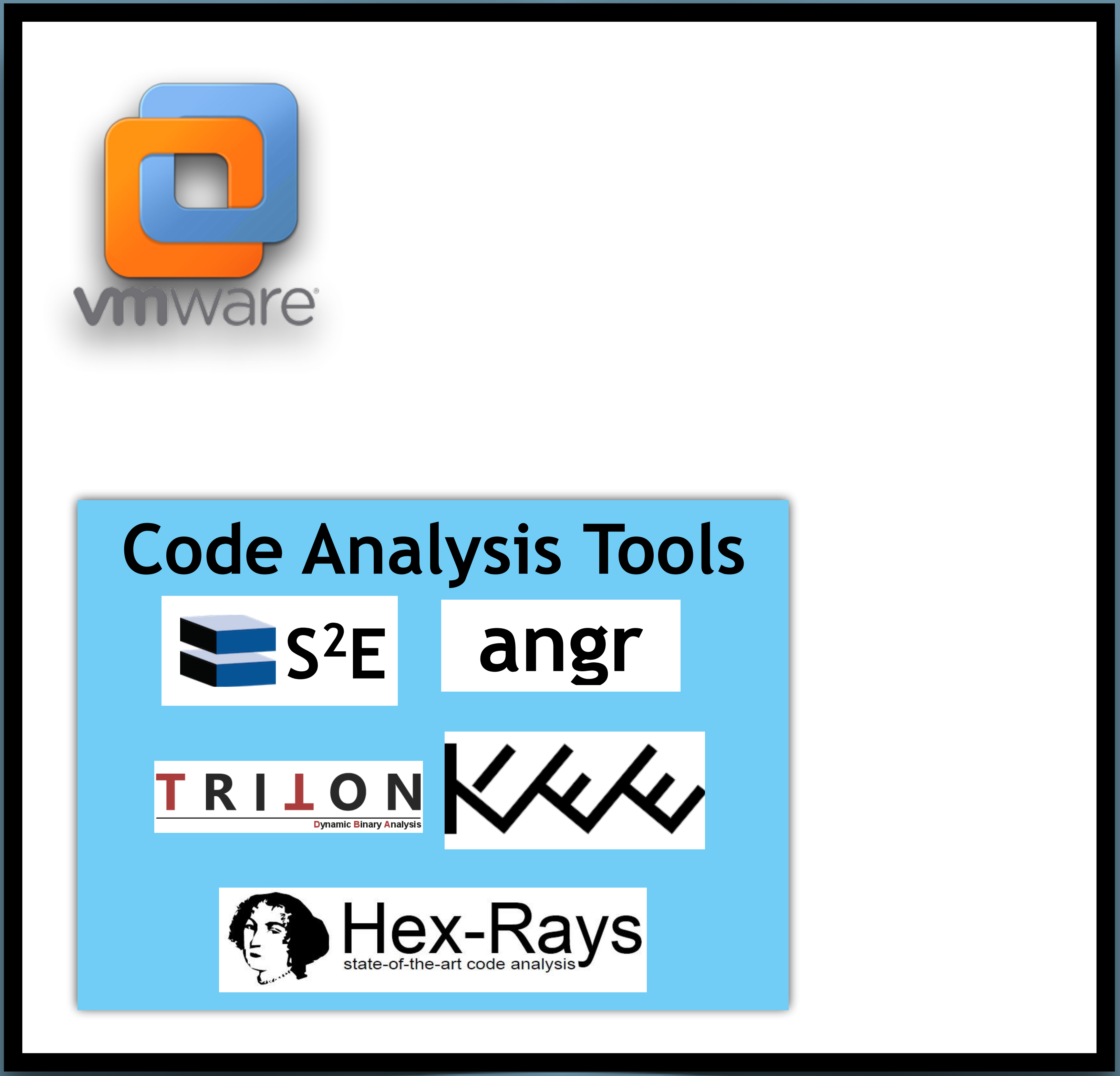
TRITON Dynamic Binary Analysis KLEE

Hex-Rays state-of-the-art code analysis

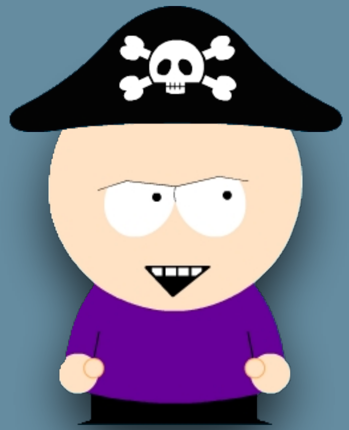
Adversarial Model Building



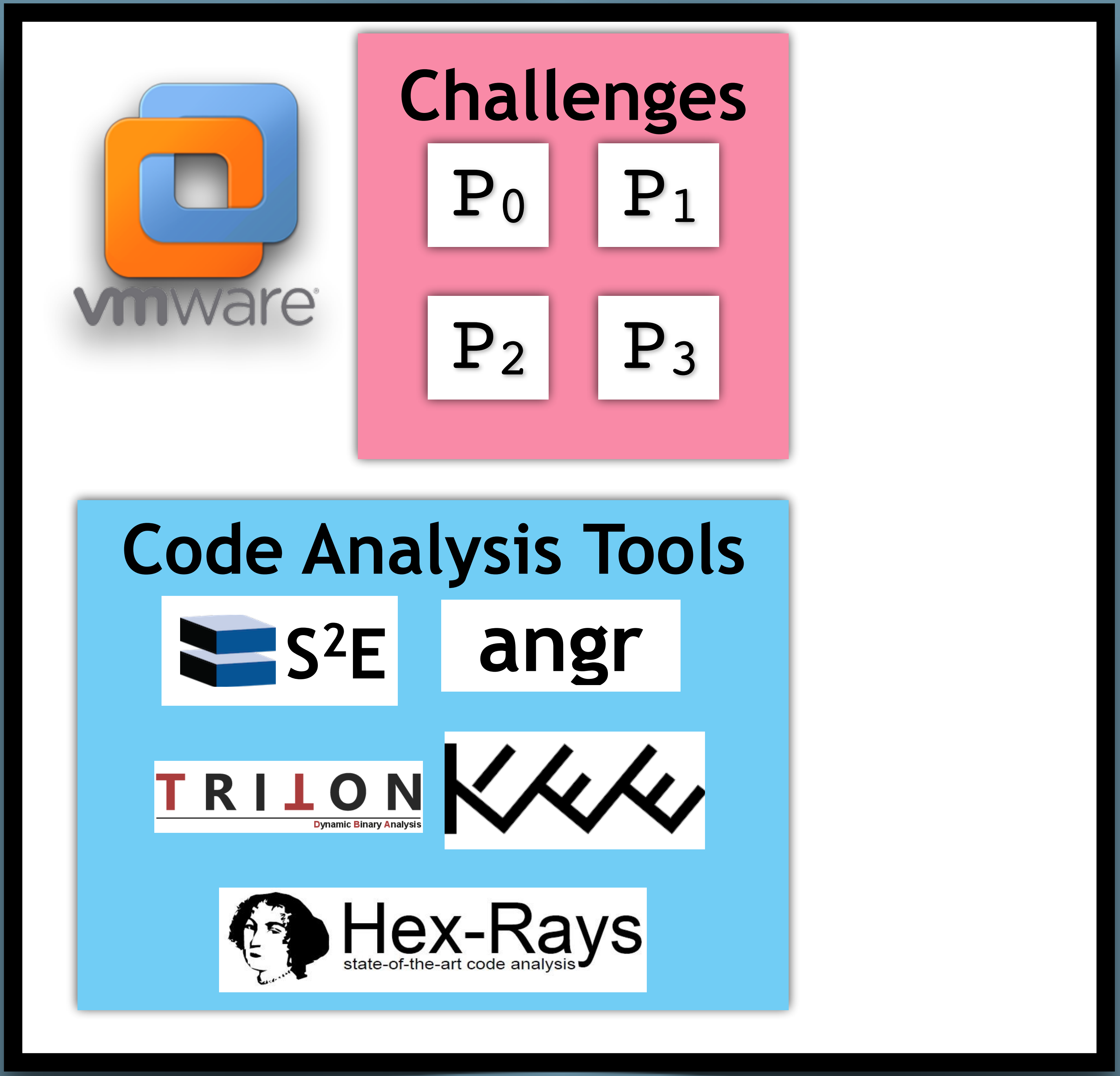
Adversarial Model Building



Code Analysis Tools



Adversarial Model Building

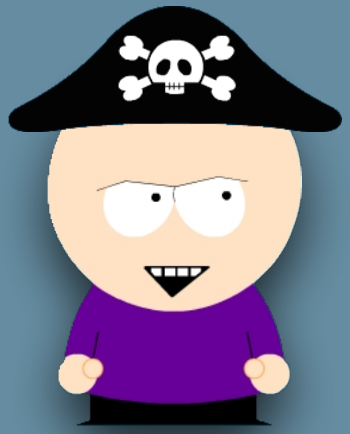


Challenges

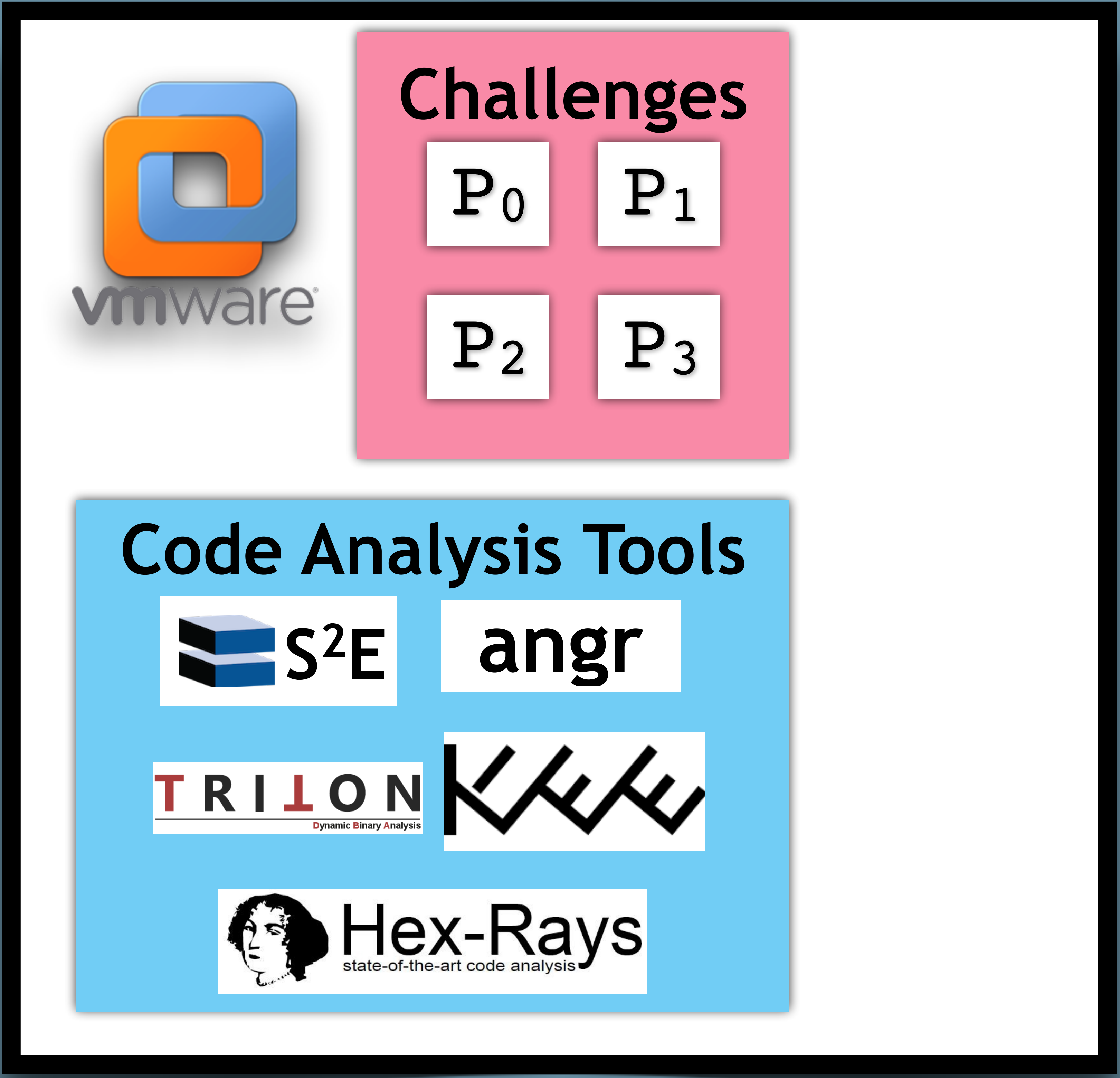
P_0 P_1

P_2 P_3

Code Analysis Tools



Adversarial Model Building



vmware

Challenges

P_0 P_1

P_2 P_3

Code Analysis Tools

S^2E

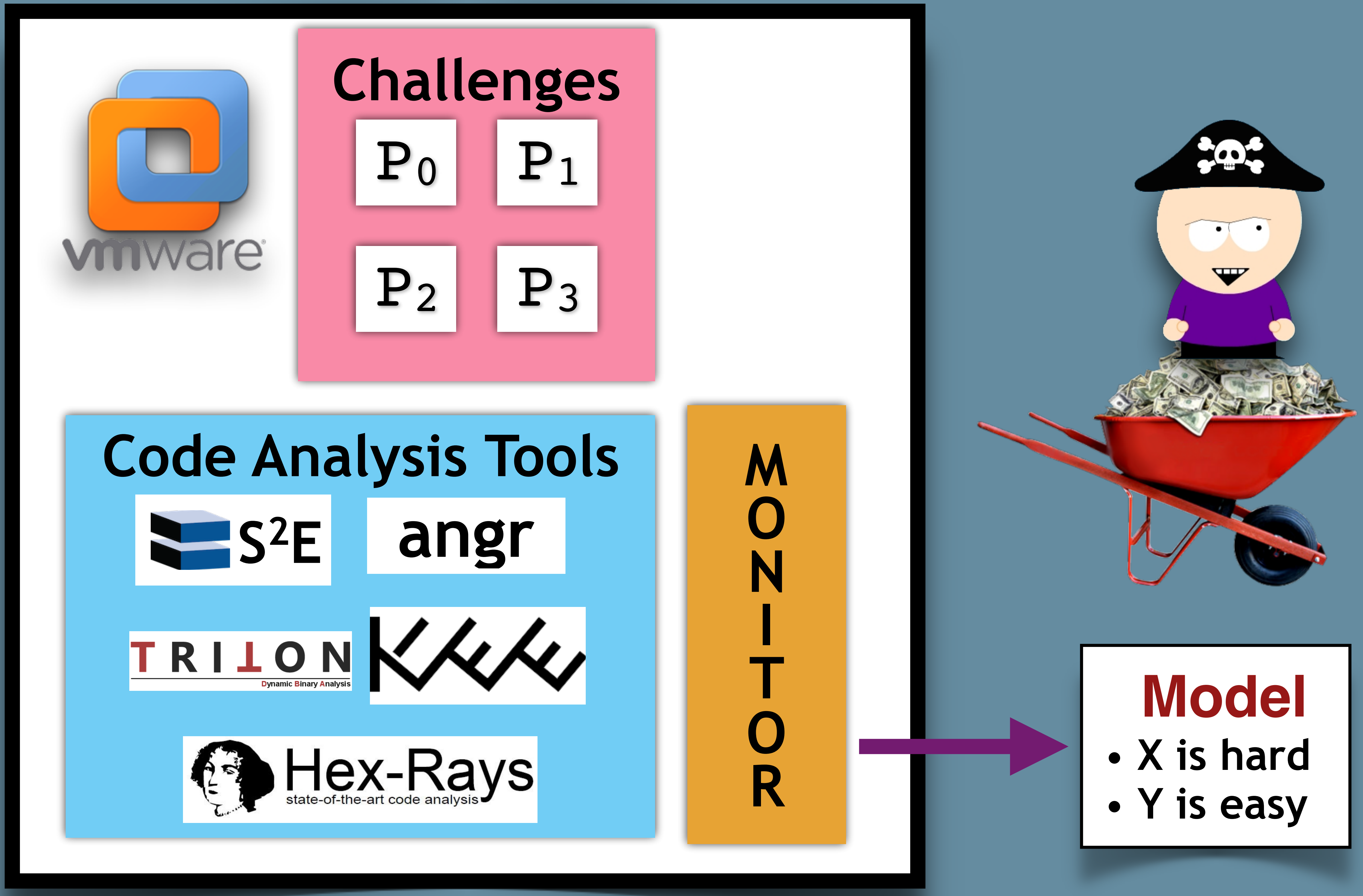
angr

TRITON
Dynamic Binary Analysis

KLEE

Hex-Rays
state-of-the-art code analysis

Adversarial Model Building



vmware

Challenges

P_0 P_1

P_2 P_3

Code Analysis Tools

S²E

angr

TRITON
Dynamic Binary Analysis

KLEE

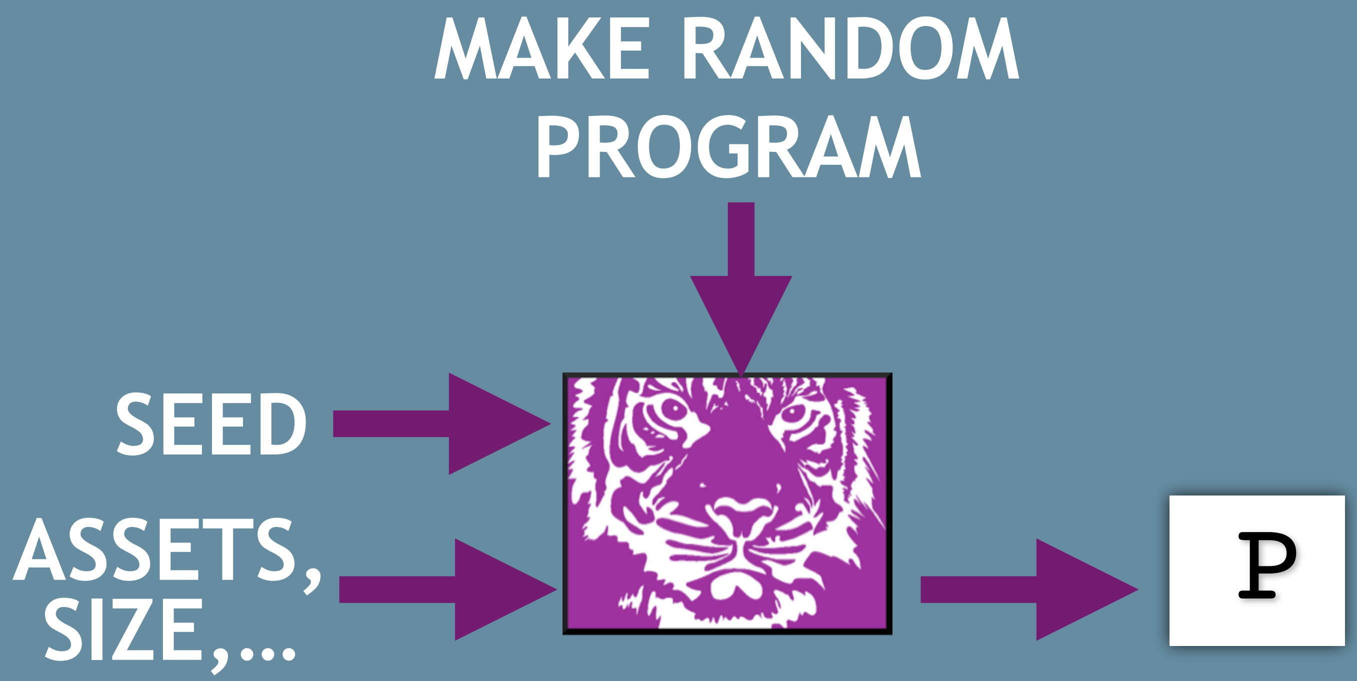
Hex-Rays
state-of-the-art code analysis

MONITOR

Model

- X is hard
- Y is easy

Generating Challenges



MAKE RANDOM PROGRAM

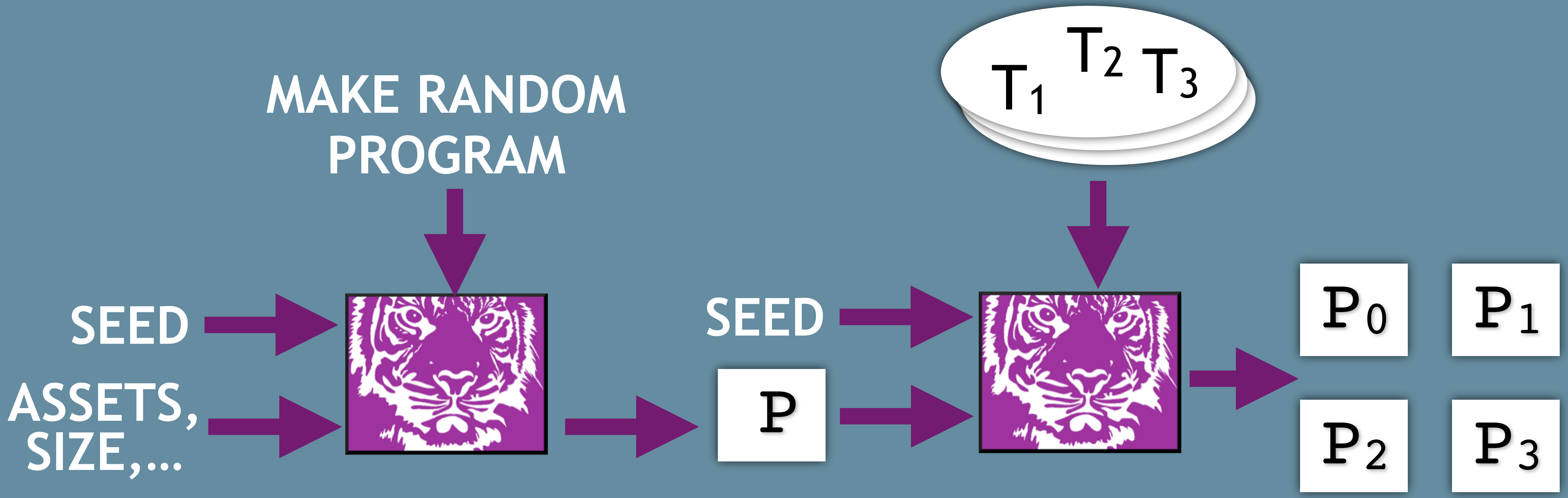
SEED

ASSETS, SIZE,...

P

- Automatically generate many challenges
- Varying levels of complexity

Generating Challenges



- Automatically generate many challenges
- Varying levels of complexity

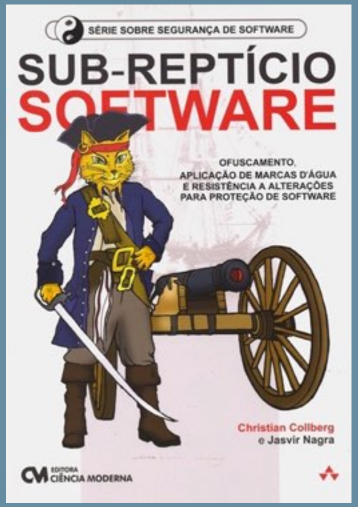
Challenges So Far…

- Easiest challenge broken by Google engineer in 8 hours.

http://tigress.cs.arizona.edu/challenges.html

Cash and/or book prizes!

SOFTWARE SECURITY SERIES
SURREPTITIOUS SOFTWARE
Obfuscation, Watermarking and Tamperproofing for Software Protection
Christian Collberg | Jasvir Nagra

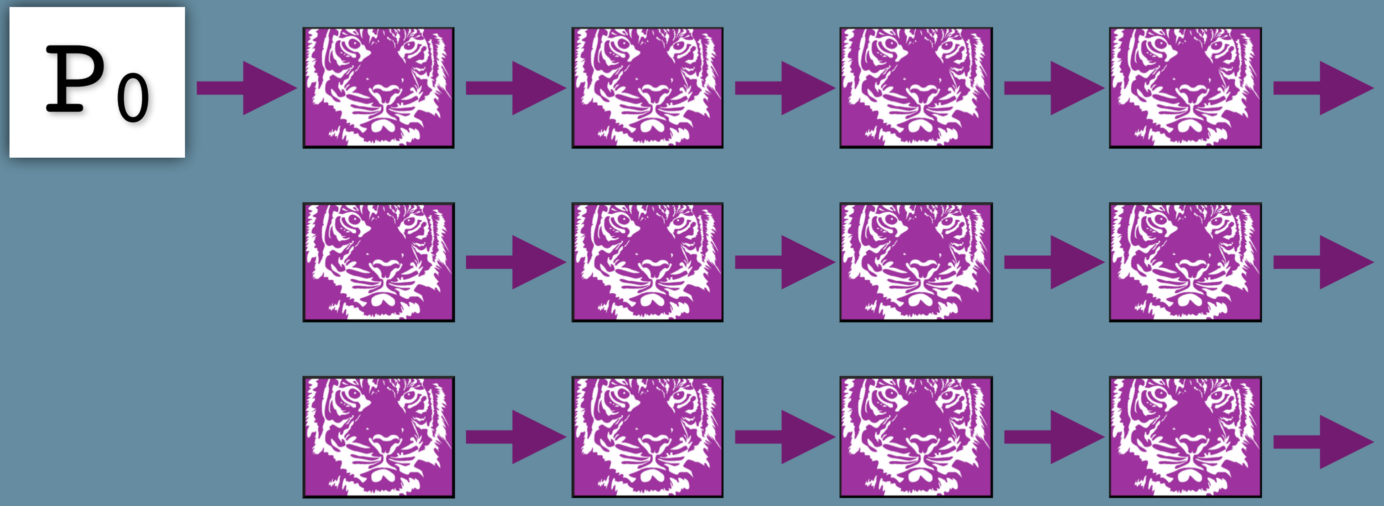

SÉRIE SOBRE SEGURANÇA DE SOFTWARE
SUB-REPTÍCIO SOFTWARE
OFUSCAMENTO, APLICAÇÃO DE MARCAS D'ÁGUA E RESISTÊNCIA A ALTERAÇÕES PARA PROTEÇÃO DE SOFTWARE
Christian Collberg e Jasvir Nagra
EDITORA CIÊNCIA MODERNA

Discussion

Meeting security criteria without meeting performance criteria is not a solution in a MATE scenario.

Meeting security criteria without meeting performance criteria is not a solution in a MATE scenario.

- Arbitrary levels of protection, at arbitrary levels of slowdown, is easy:

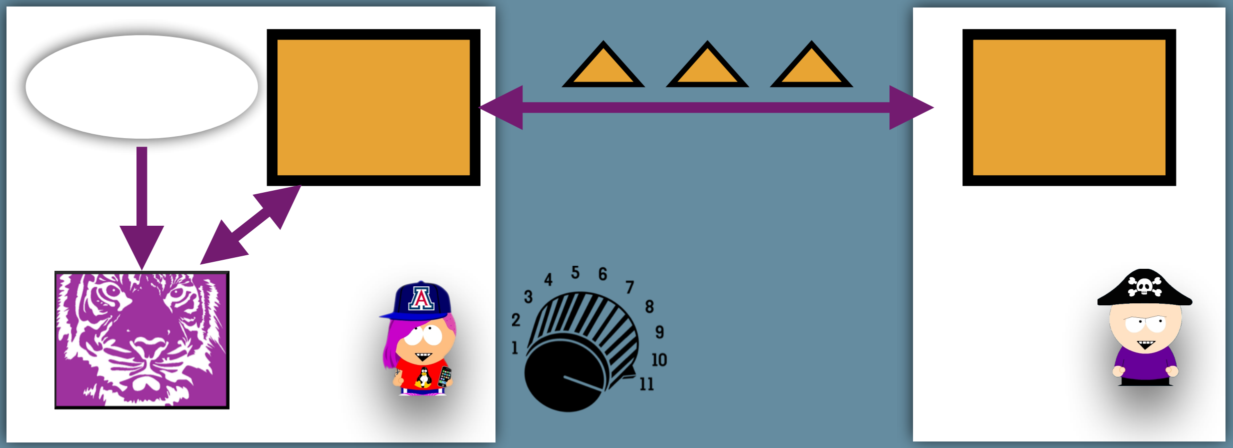


Meeting precision criteria without meeting performance criteria is not a solution for anti-MATE analyses.

- Real programs are large, and analyses need to scale.
- Saying that an obfuscation falls against a particular analysis is meaningless without knowing the performance cost.

Obfuscating transformations are primitives that provide time-limited protection. Updatable security can extend the protection they provide.

- All language-based obfuscations will break.
- Updatable security can increase the cost to the attacker.



To make progress in this field, the community must settle on rigorous evaluation procedures.

- Evaluation is a mess — we need to fix this.
- Help, anyone?
- Learn from public challenges.

MATE Predictions?

	Performance	Security	Scenarios
Hardware based			
Language based			
Crypto based			
Updatable Security			

- Which techniques will prevail?
- Will they coexist, but in different scenarios?
- Will we see combinations of techniques?

Questions?

collberg@gmail.com

Slides: tigress.cs.arizona.edu/eurocrypt.pdf