# Combine Static Analysis and Dynamic Testing to analyze Binary Code

April, 22th 2013

Nguyễn Minh Hải

# Table of Contents

# Binary Code

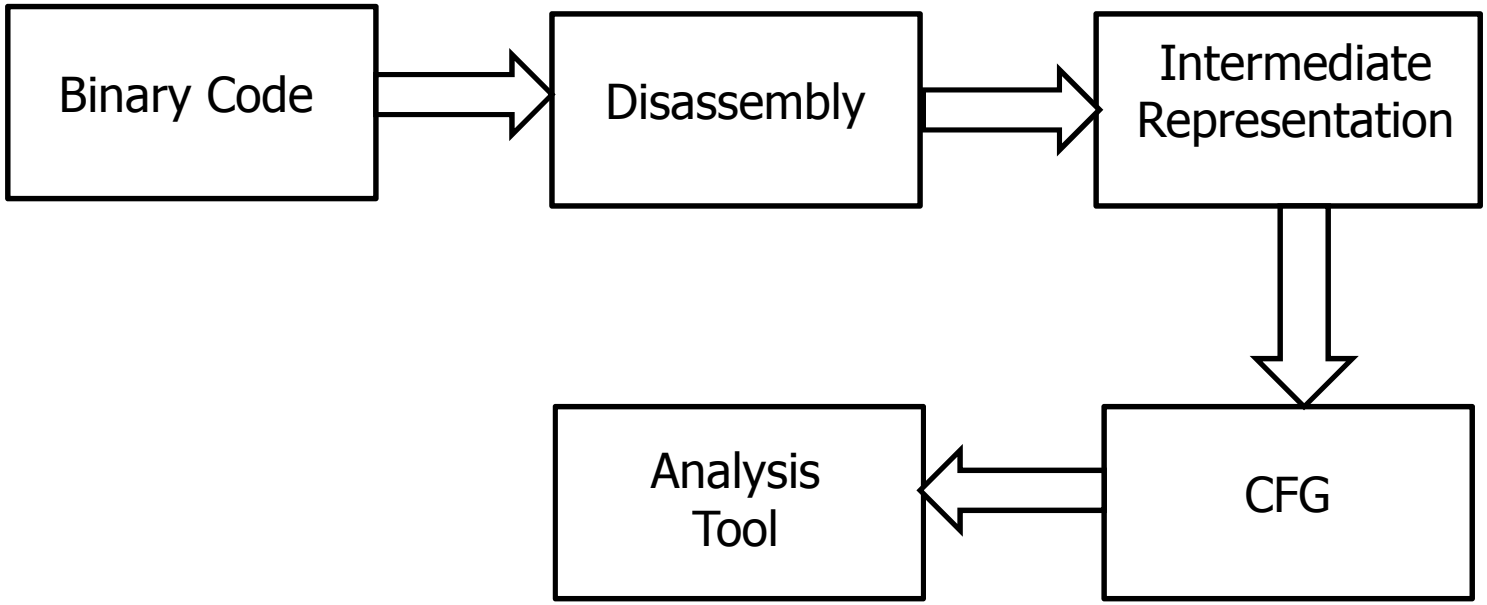
- Binary code or Source code ?
- Benefits:
  - Source code is unavailable
  - Binaries are reliable
- Challenges:
  - Complexity
  - Lack of higher-level semantics
  - Obfuscation of malicious code

# Frameworks of analyzing binaries

- BINCOA
- IDA Pro
- Jakstab

……

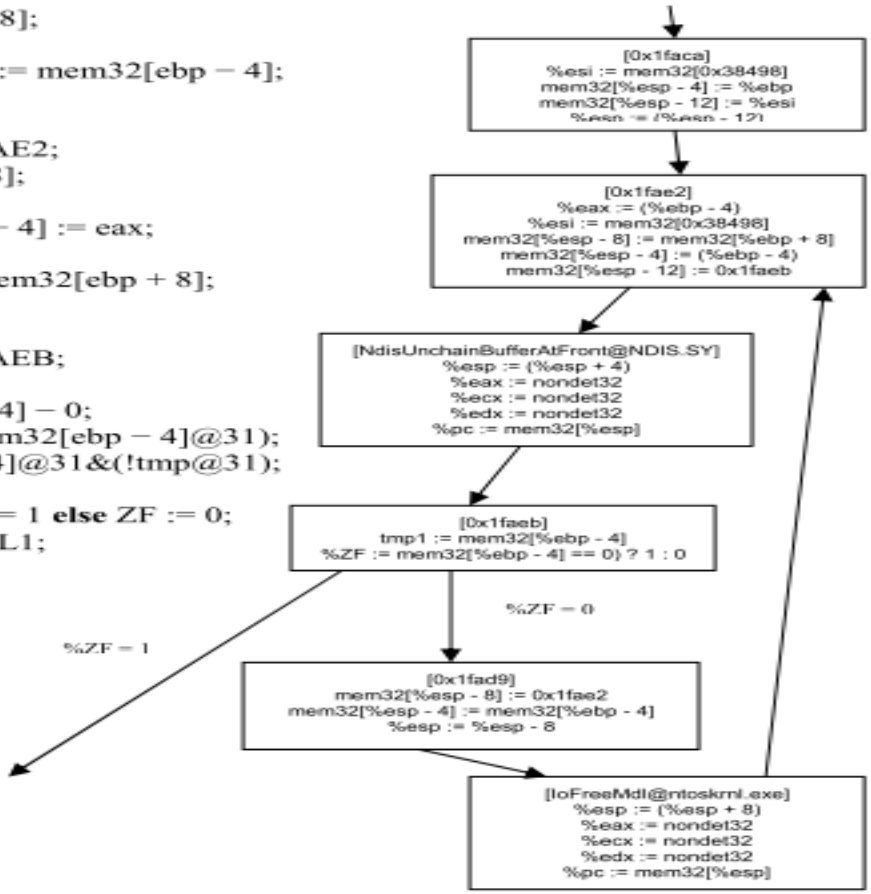
# Architecture



# Example

| Disassembly | Intermediate Representation |
|---|---|
| mov esi, [0x38498] | esi := mem32[0x38498]; |
| jmp 0x1fae2 | **goto** L2; |
| push [ebp - 4] | **L1**: mem32[esp − 4] := mem32[ebp − 4]; |
| | esp := esp − 4; |
| call [0x38588] | esp := esp − 4; |
| | mem32[esp] := 0x1FAE2; |
| | **goto** mem32[0x38588]; |
| lea eax, [ebp - 4] | **L2**: eax := ebp − 4; |
| push eax | mem32[esp − 4] := eax; |
| | esp := esp − 4; |
| push [ebp + 8] | mem32[esp − 4] := mem32[ebp + 8]; |
| | esp := esp − 4; |
| call esi | esp := esp − 4; |
| | mem32[esp] := 0x1FAEB; |
| | **goto** esi; |
| cmp [ebp - 4], 0 | tmp := mem32[ebp − 4] − 0; |
| | CF := tmp@31&(!mem32[ebp − 4]@31); |
| | OF := mem32[ebp − 4]@31&(!tmp@31); |
| | NF := tmp@31; |
| | **if** (tmp = 0) **then** ZF := 1 **else** ZF := 0; |
| jne 0x1fad9 | **if** (ZF = 0) **then goto** L1; |

Control Flow Graph



# Shortcommings

- IDA Pro
  - Closed source software
  - Recursive traversal disassembler decodes binaries by Depth First Search algorithm.
  - Problem of indirect jump
- Jakstab
  - Abstract interpretation with data flow analysis
  - Abstract interpretation can produce imprecision

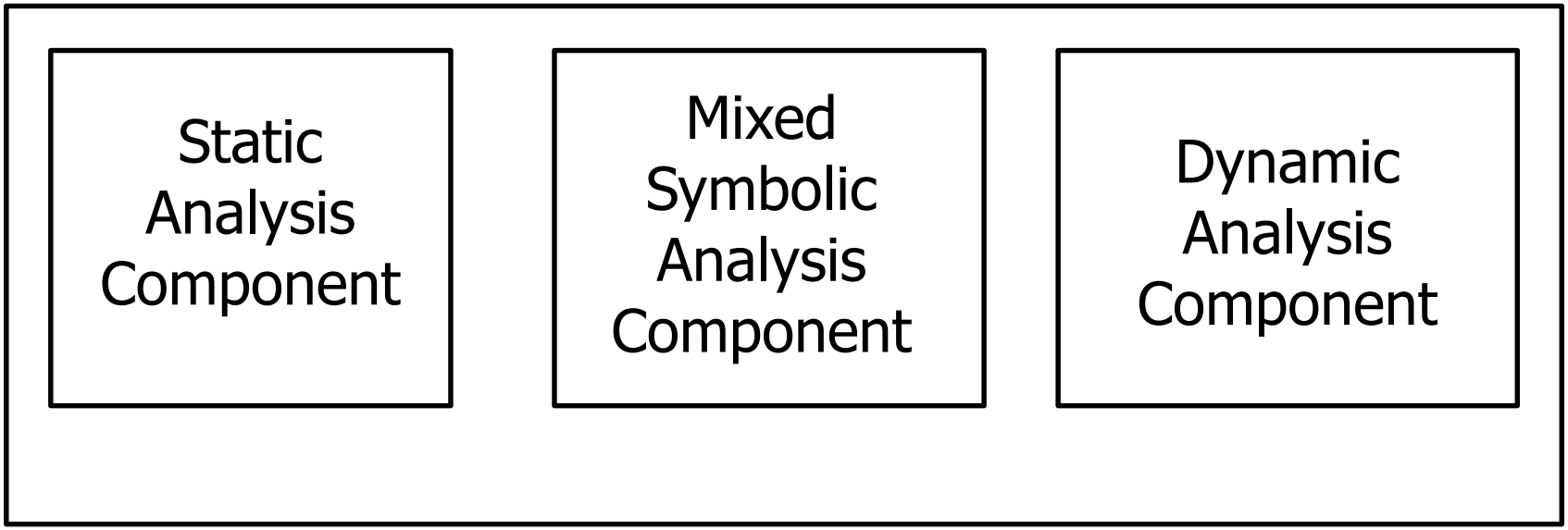
# Table of Contents

- Motivation
  - Binary Code: benefits and challenges
  - Binary Analysis Frameworks
- Goal
  - Architecture
  - Example
  - Project of integration of symbolic execution and linear invariant to Jakstab
- Summary

# Goal

- Create the framework of combining static analysis and dynamic testing to analyze binary files.
- Apply this framework to detect malicious code.

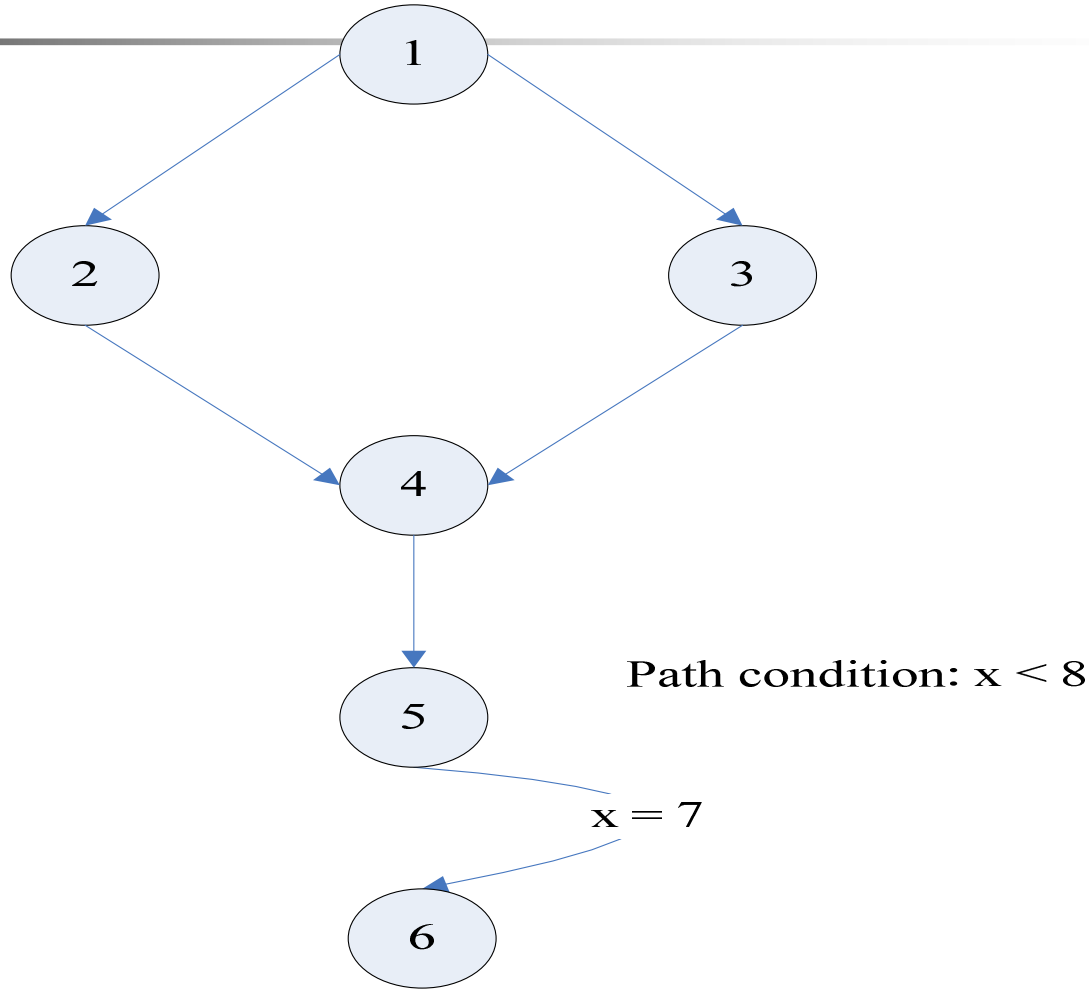
# Architecture



# Architecture (tt)

- Static analysis component (SAC)
  - Translate assembly to intermediate representation (IR)
  - Provide the core functions of static analysis, such as data flow analysis, abstract interpretation …
- Dynamic analysis component (DAC)
  - Provide the core functions of implementing the binary code
  - Allow to monitor the behavior of executable
- Mixed symbolic analysis component (MSAC)
  - Combine symbolic execution and the functions of SAC and DAC
  - Provide the functions of SMT Solvers to resolve the path condition

# Example



# Project

- Integrate symbolic execution and linear invariant to Jakstab
  - When the instructions can be disassembled, apply symbolic execution to find the environment and path condition.
  - When unresolved address found, instead of using abstraction interpretation, stop and use calculated condition to generate a test-case.
    - Dynamic executing the code with given test-case.
    - Find the branch destination.
    - Resume.

# DEMO

*<loop.exe>*

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

*<loop.asm>*

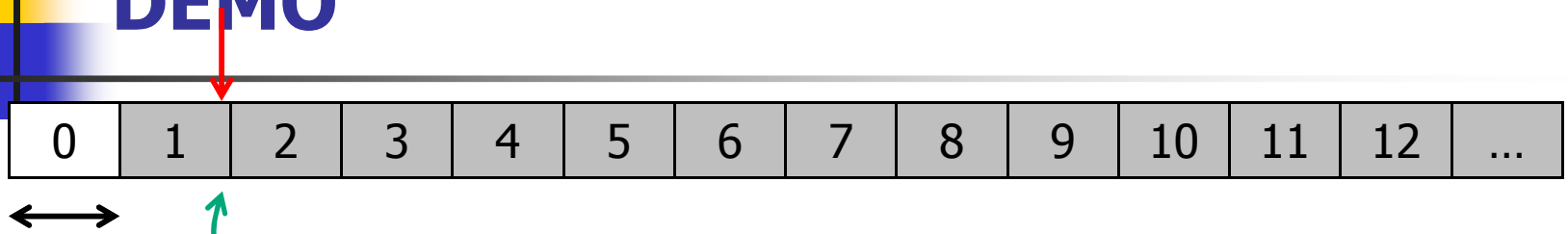
```
start:
    mov ecx, 1000
    mov eax, 0
loop1:
    inc eax
    loop loop1
    ret;
```

**[masm32 assembler]**

# DEMO

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | ... |

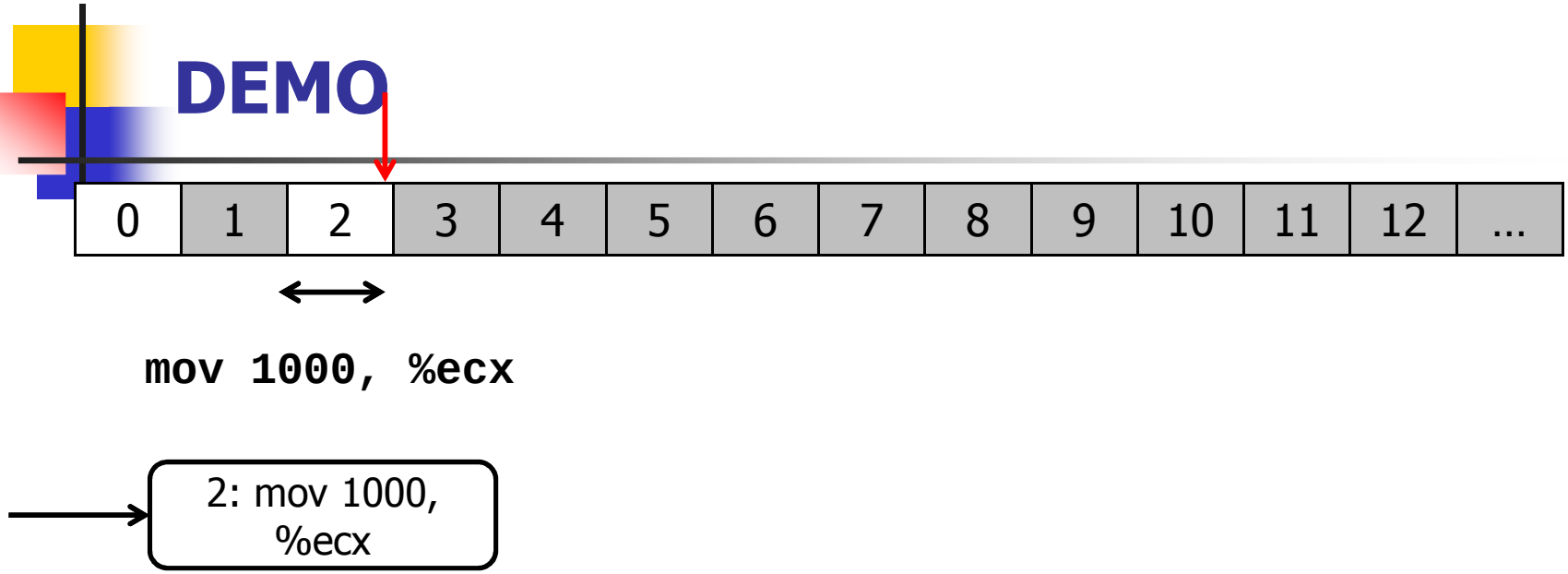
# DEMO



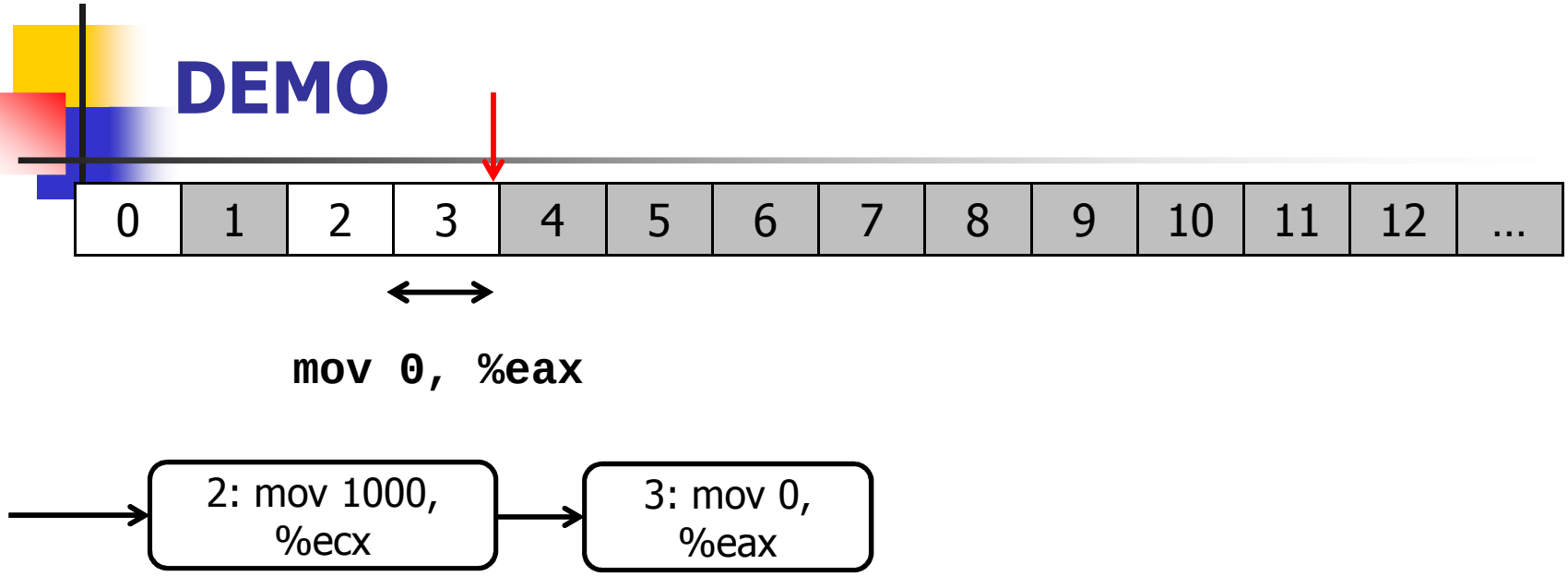

PE Header

- Address of entry point
- Program information

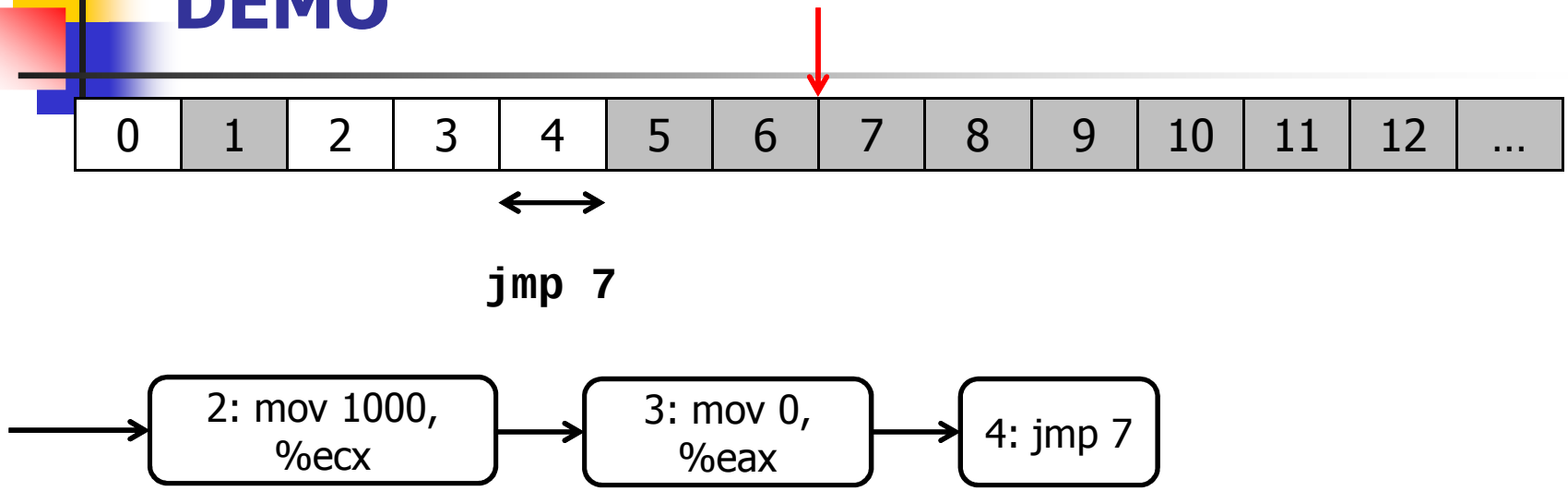
# DEMO



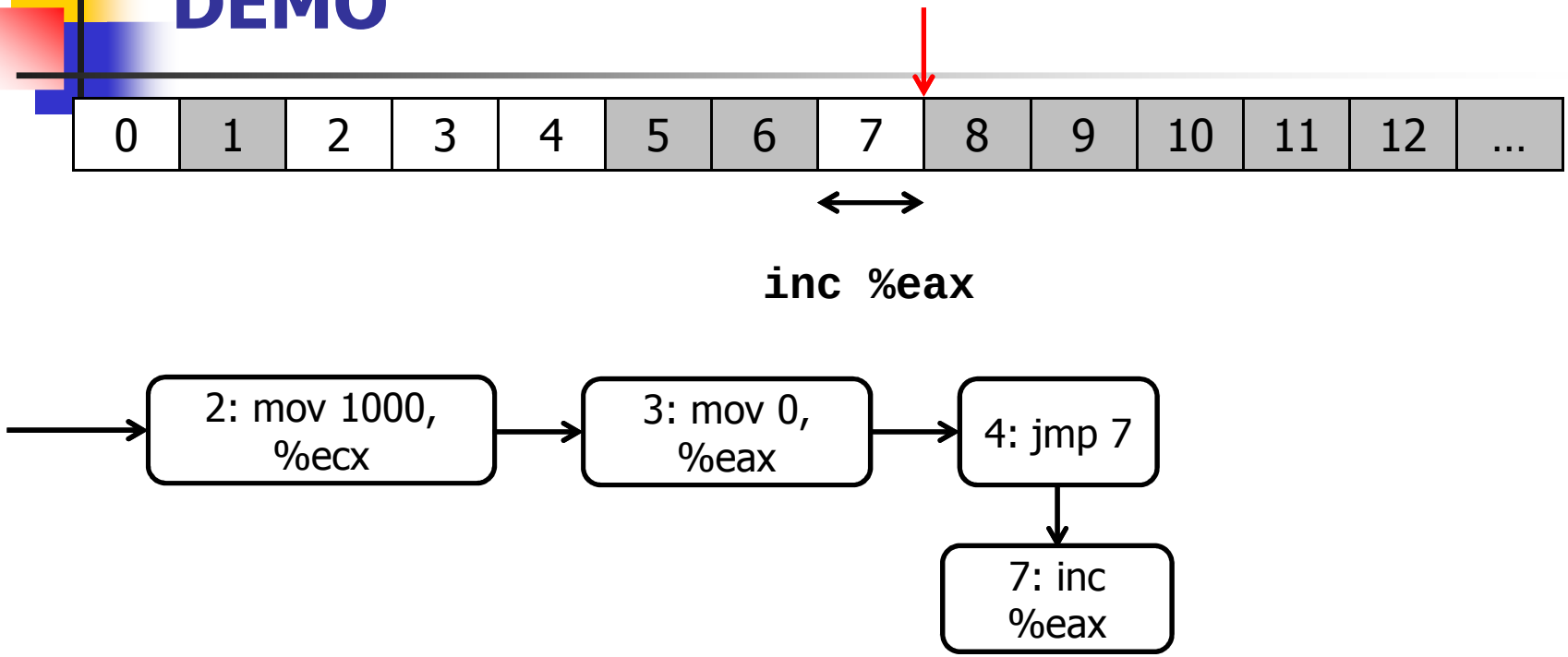
# DEMO



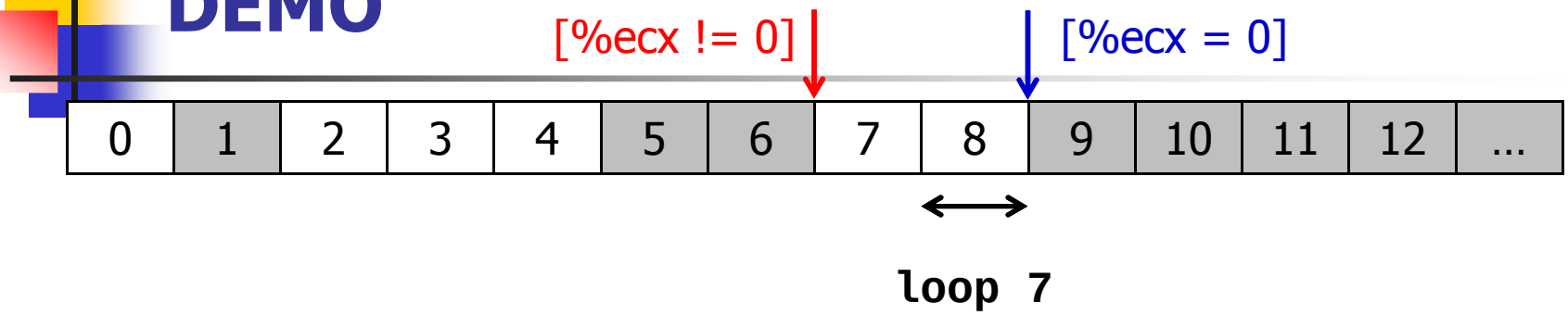
# DEMO

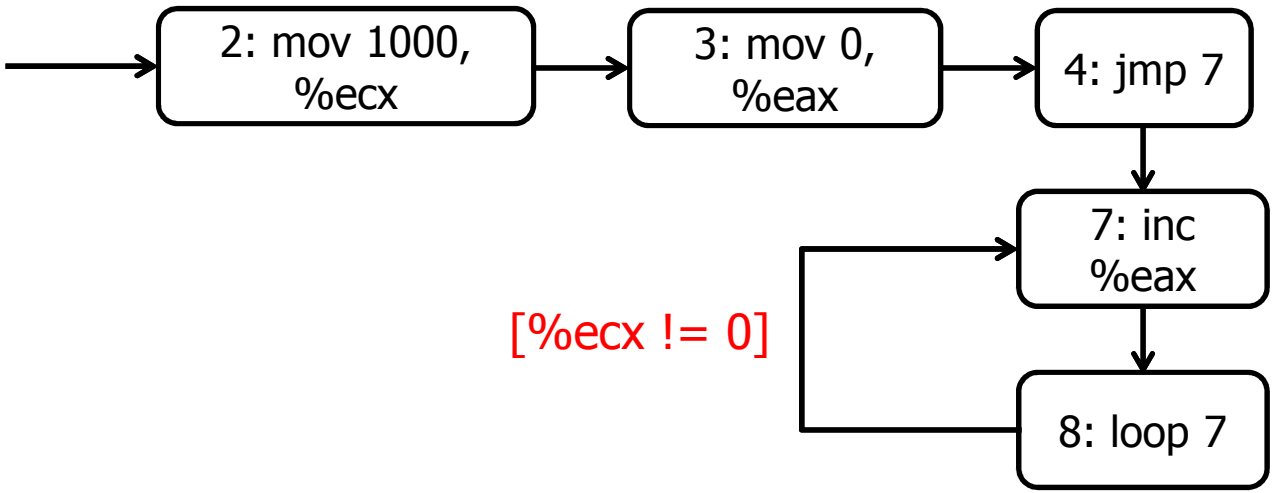


# DEMO

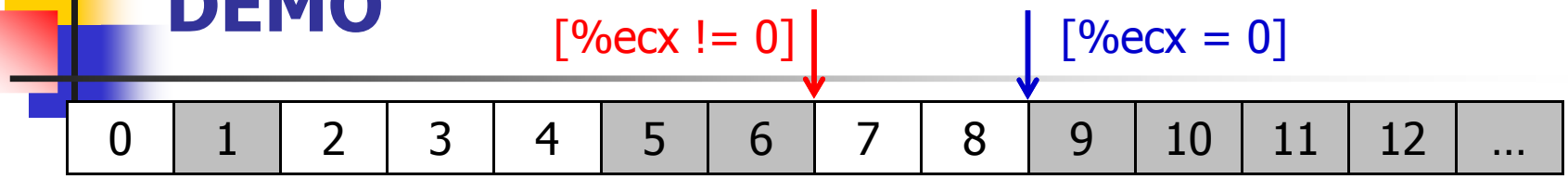

0
1
2
3
4
5
6
7
8
9
10
11
12
...
inc %eax
2: mov 1000, %ecx
3: mov 0, %eax
4: jmp 7
7: inc %eax

# DEMO

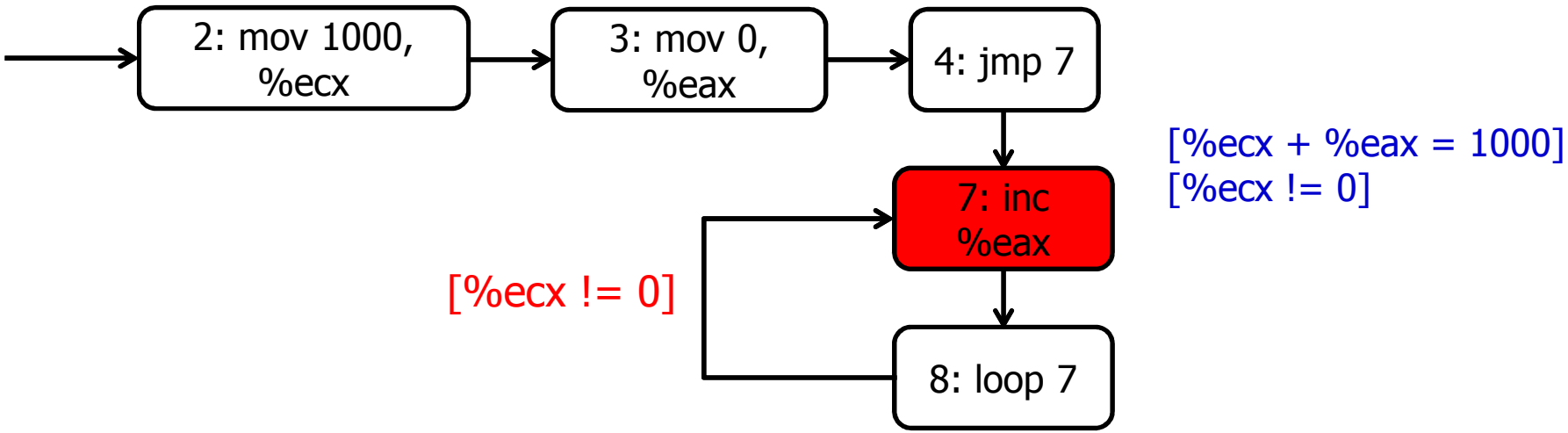

[%ecx != 0]
[%ecx = 0]
0
1
2
3
4
5
6
7
8
9
10
11
12
...
loop 7


2: mov 1000, %ecx
3: mov 0, %eax
4: jmp 7
7: inc %eax
8: loop 7
[%ecx != 0]

# DEMO

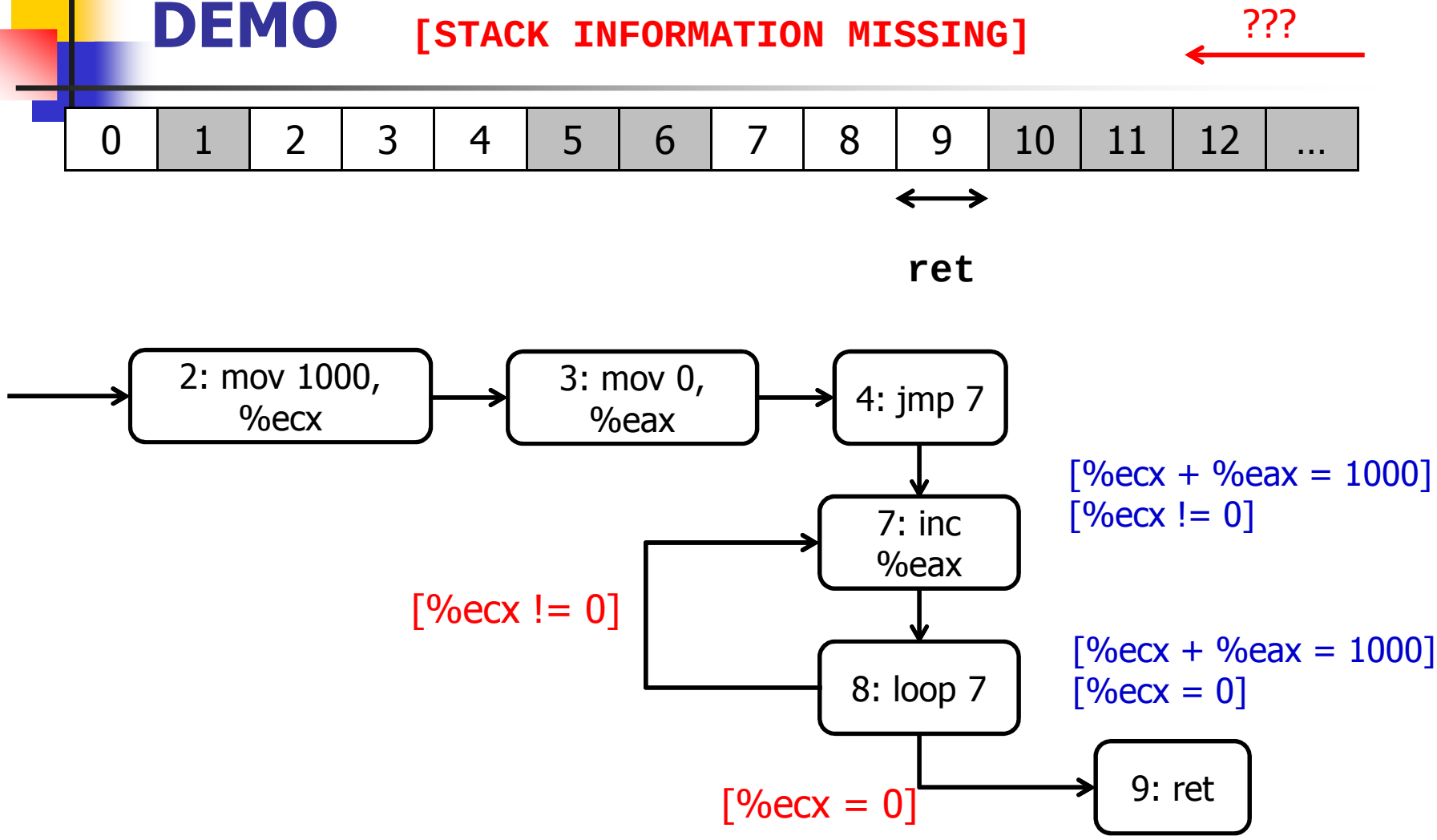



[ Invariant: **%ecx + %eax = 1000** ]



# DEMO

**[STACK INFORMATION MISSING]**



# Summary

- To analyze binary code is imperative and challenging
- Our work is to create a framework of combining dynamic testing and static analysis
  - Static analysis component
  - Dynamic analysis component
  - Mixed symbolic analysis component

# Reference


- [1] Colon, M., Sankaranarayanan, S., and Sipma, H. Linear invariant generation using non-linear constraint solving. In *Computer Aided Verification (July 2003)*
- [2] Johannes Kinder, Helmut Veith, Florian Zuleger. An Abstract Interpretation–Based Framework for Control Flow Reconstruction from Binaries. *In Proceedings of the 10th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2009)*, vol. 5403, Lecture Notes in Computer Science, Springer, January 2009, pp. 214–228
- [3] A. Bockmayr and V. Weispfenning. *Solving numerical constraints"* In A. Robinson and A. Voronkov, editors, Handbook of Automated Reasoning, volume I, chapter 12, pages 751–842. Elsevier Science, 2001
- [4] K. Fukuda and A. Prodon. Double description method revisited. In *Combinatorics and Computer Science*, volume 1120 of LNCS, pages 91–111. Springer-Verlag, 1996
- [5] Johannes Kinder, Helmut Veith. Jakstab: A Static Analysis Platform for Binaries. In *Proceedings of the 20th International Conference on Computer Aided Verification (CAV 2008)*, vol. 5123, Lecture Notes in Computer Science, Springer, July 2008, pp. 423–427
- [6] G. Balakrishnan, T. Reps, D. Melski, T. Teitelbaum:WYSINWYX:What You See Is Not What You eXecute. In *VSTTE,* Zurich, Switzerland (2005)

# Q&A