ISP RAS

MAJORCA

Multi-Architecture JOP and ROP Chain Assembler

Alexey Nurmukhametov
Alexey Vishnyakov
Vlada Logunova
Shamil Kurmangaleev

December 3, 2021

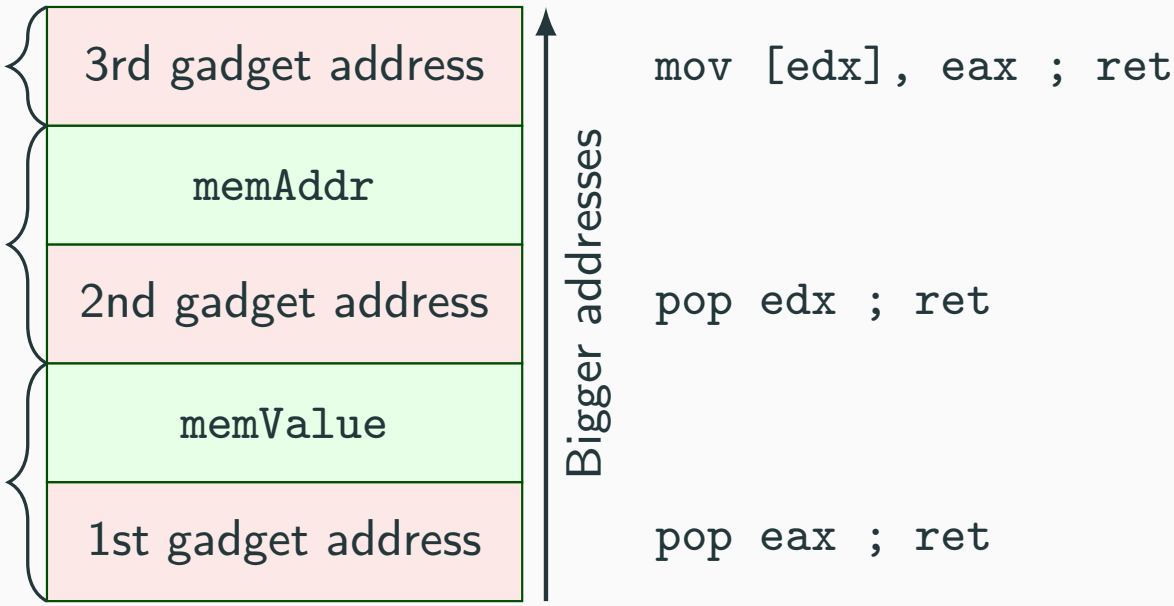
ISP RAS

Motivation

- Errors and vulnerabilities inevitably exist in all programs under development.
- Modern exploits rely on code-reuse techniques as the last stage of exploitation.
- It is crucial to understand how powerful code-reuse attacks can be.
- Mitigation techniques against them have to be thoroughly tested.
- Advanced tools for code-reuse attack generation provide a quality estimation for defensive mechanisms.

Return-Oriented Programming

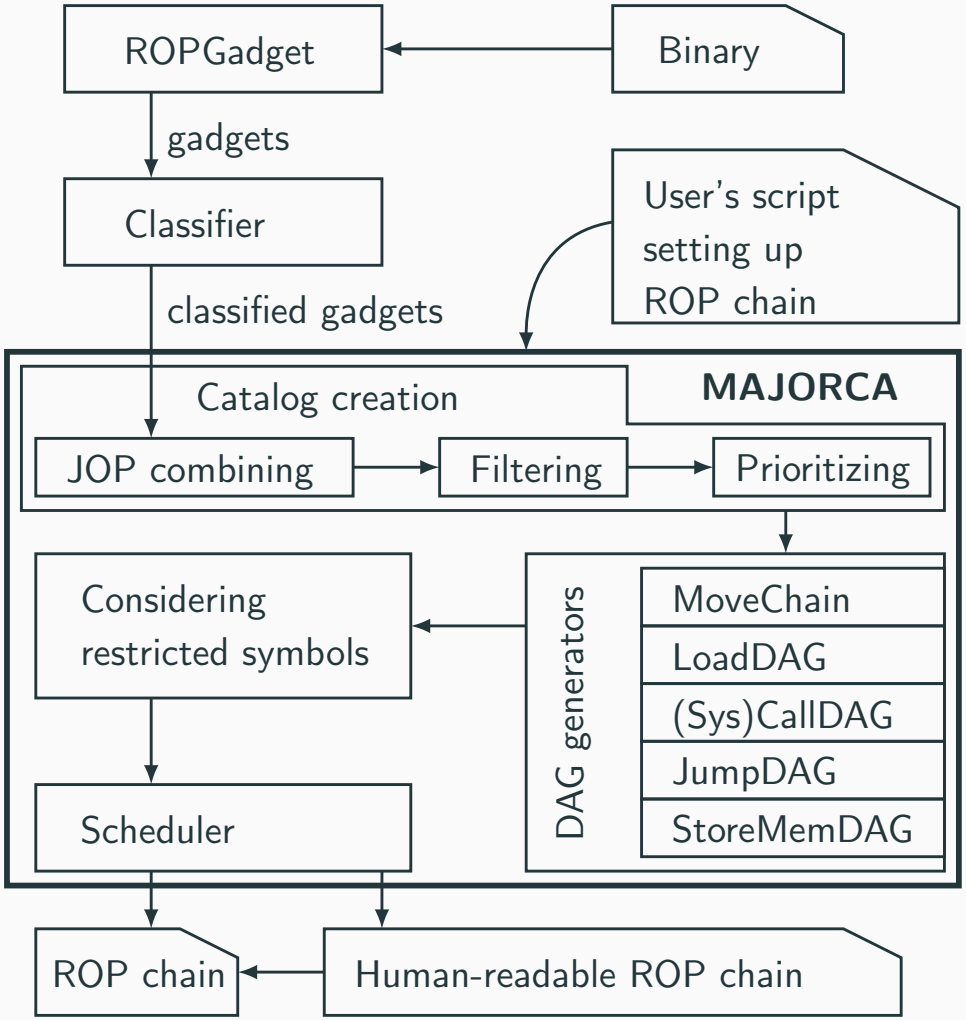
- **Return-oriented programming (ROP)** – an exploitation technique that bypasses DEP (data execution prevention).
- **Gadget** – an instruction sequence that performs some calculations and redirects control flow to another gadget, e.g. via `ret` instruction at the end of the sequence.
- Gadgets have to be located in non-randomized memory regions.



MAJORCA Contributions

- The method to automatically generate both ROP and JOP payloads in an architecture agnostic manner.
- The algorithm that considers restricted symbols in gadget addresses and data.
- MAJORCA tool that generates both ROP and JOP chains for x86 and MIPS considering restricted symbols thoroughly.
- `rop-benchmark` that compares MAJORCA with open-source ROP compilers.
- We define ROP chaining metric to estimate the probability of successful ROP chaining for different OS with the portfolio of ROP compilers.

MAJORCA Design



Gadget Classification

- ROPgadget finds gadgets.
- Gadget types (boolean postconditions) define a new ISA.
  - `MoveRegG: OutReg` $\leftarrow$ `InReg`
  - `LoadConstG: OutReg` $\leftarrow$ `[SP + Offset]`
- Gadget classification finds out:
  - Gadget types and their parameters.
  - A list of clobbered registers (which values are not preserved after gadget execution).
  - Gadget frame info (frame size, an offset of next gadget address).

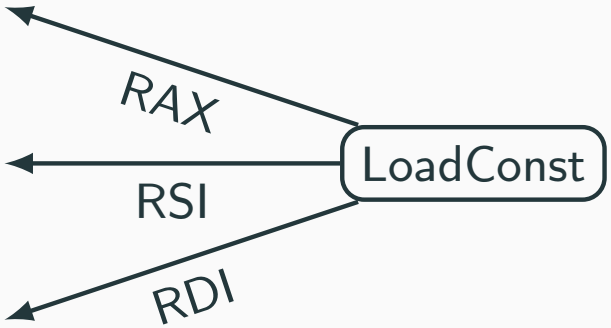
Architecture Specific Description

- Chain generation algorithms are architecture agnostic.
- Architecture description contains:
  - calling convention (which registers are arguments),
  - numbers for system calls,
  - registers that are used in chain.

Gadget Loading Values to Registers

- Classification finds out only gadgets that load value to single register.
- The key difference from Q[1] – MAJORCA combines gadgets loading many values to many registers.

```
LoadConst: POP RAX ; POP RDI ; POP RSI ; RET
```



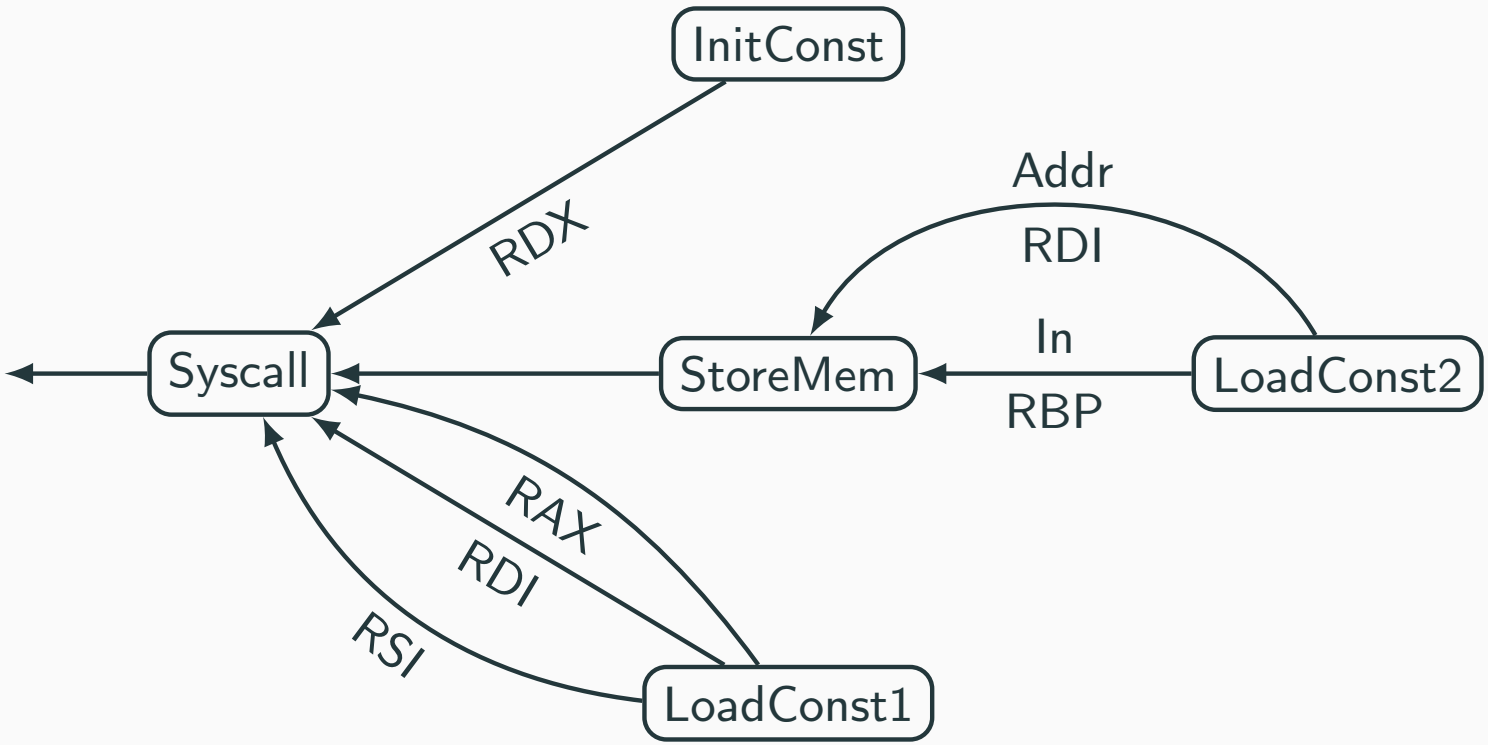
[1] E. J. Schwartz et al. Q: exploit hardening made easy

Preprocessing of Gadgets

- JOP gadgets are combined with ROP gadgets.
  - Resulting into an ordinary ROP gadget.
  - `pop rax ; pop rcx ; ret ; pop rdx ; jmp rcx` — is same as `LoadConst: rax ← [SP], rdx ← [SP + 24]` with frame size `FrameSize = 32` and the next gadget address by offset 8 (`NextAddr = [SP + 8]`), where `pop rdx ; jmp rcx` gadget address has to be placed by offset 16.
- Chain generation is a brute-force task.
- Gadgets are filtered, i.e., duplicates are removed and only the best ones are used to construct the ROP chain.
  - It is a task of searching for extremums in a partially ordered set.
- Gadgets are prioritized by quality.
- Special indexes are built to speed up requests for fetching gadgets of particular type and parameters.

Directed Acyclic Graph (DAG) of Gadgets

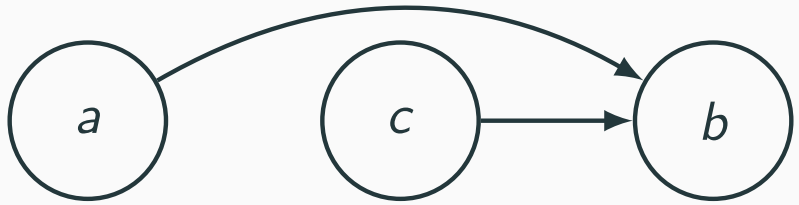
```
LoadConst1: POP RAX ; POP RDI ; POP RSI ; RET
LoadConst2: POP RDI ; POP RBP ; RET
StoreMem  : MOV [RDI], RBP ; RET
InitConst : XOR RDX, RDX ; RET
```



Gadgets DAG Scheduling

The schedule for DAG has to comply with the following:

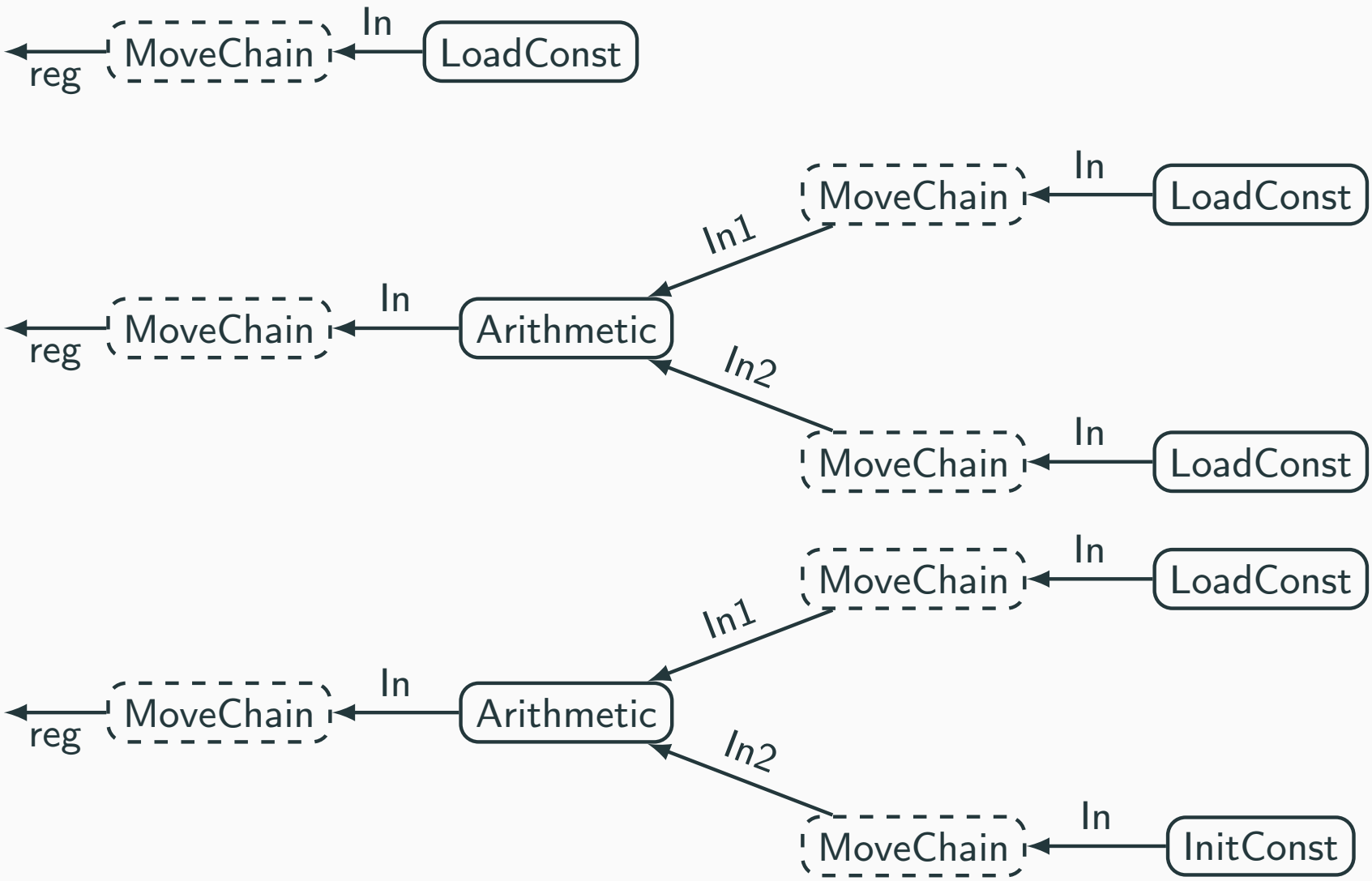
- It has to be a topological sort of DAG.
- If gadget *b* uses the output register of gadget *a*, then this register should not be clobbered by any gadgets in the schedule between *a* and *b*.



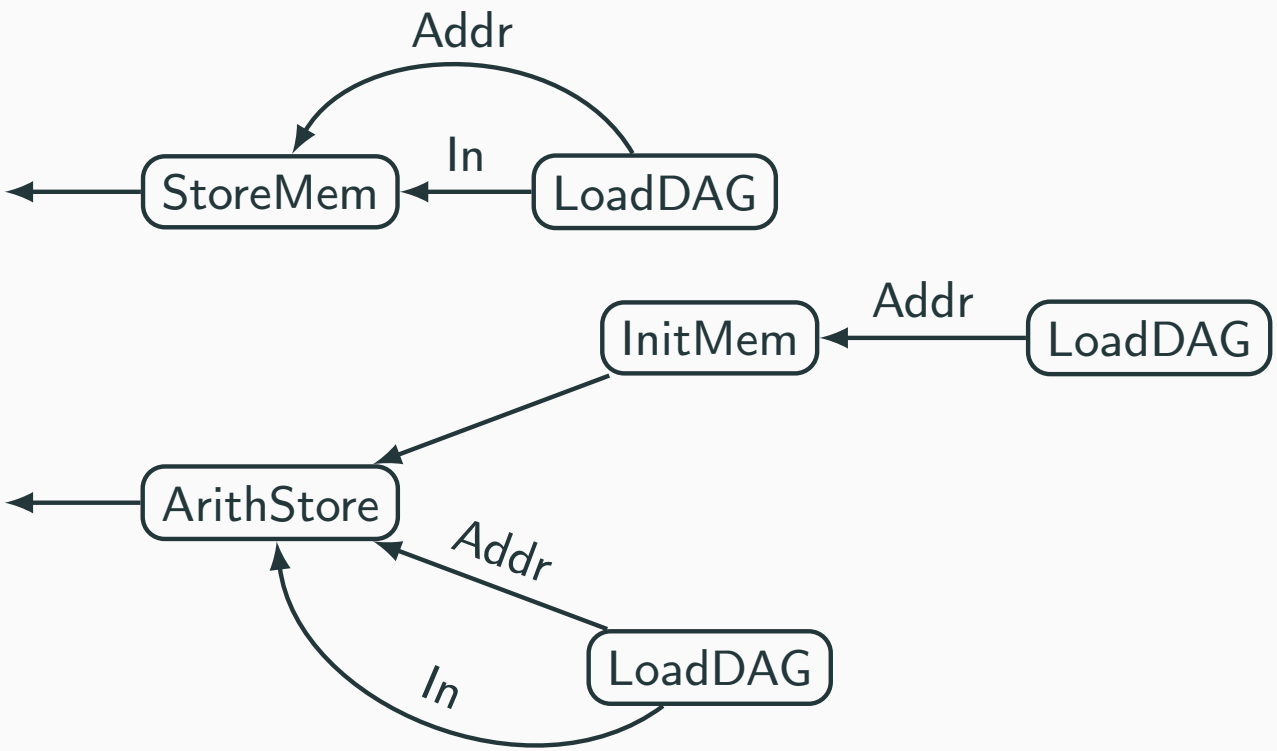
Gadgets DAG Selection

- The task of generating move chains (moving value from one register to another) is the task of finding a feasible path between graph vertices:
  - vertices – registers,
  - egdes – gadgets.
- It is possible to load value to register *a* by loading it into register *b* and then moving it to *a* by corresponding move chain.
- Value can be loaded into register as a result of arithmetic operation.
- Memory can be initialized by combination of following instructions:
  - `mov dword[eax], 0 ; ret ; add [eax], ebx ; ret`

Loading Value from Stack to *reg*

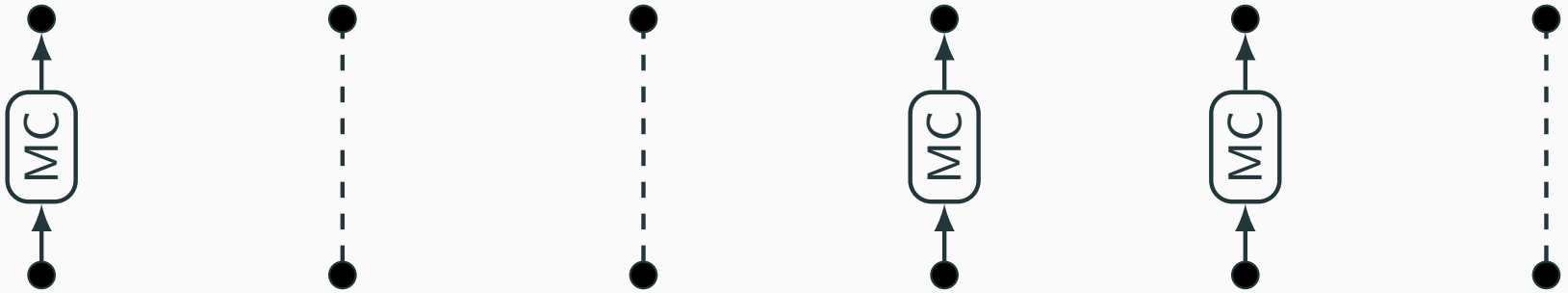


Storing Value to Memory

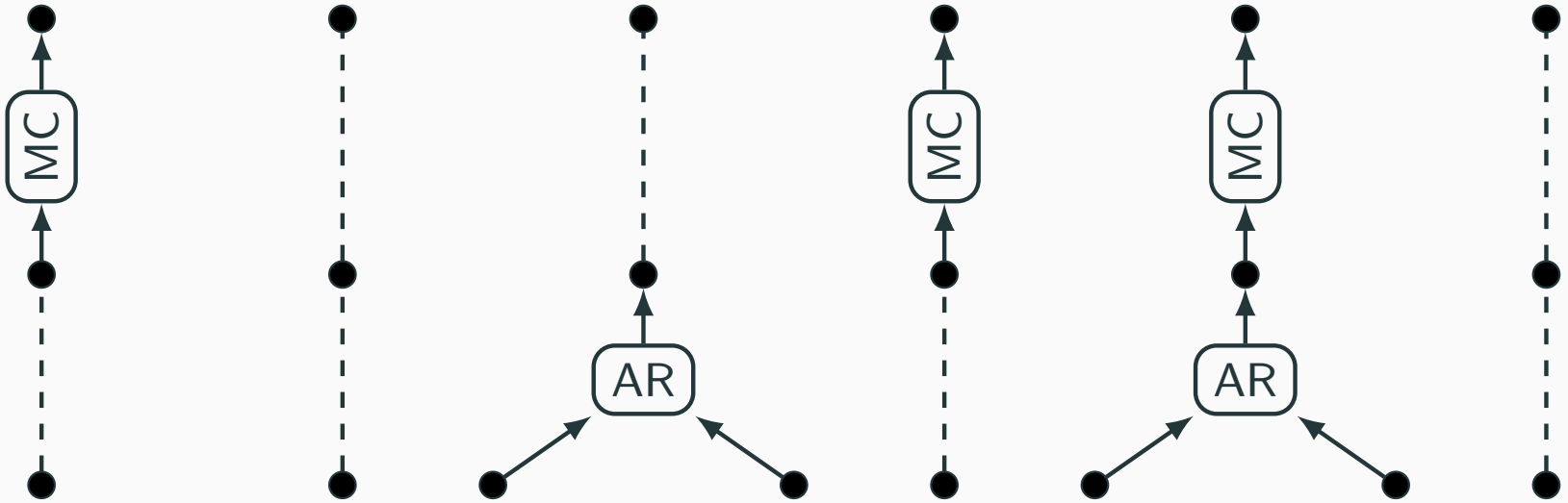


Registers Initialization

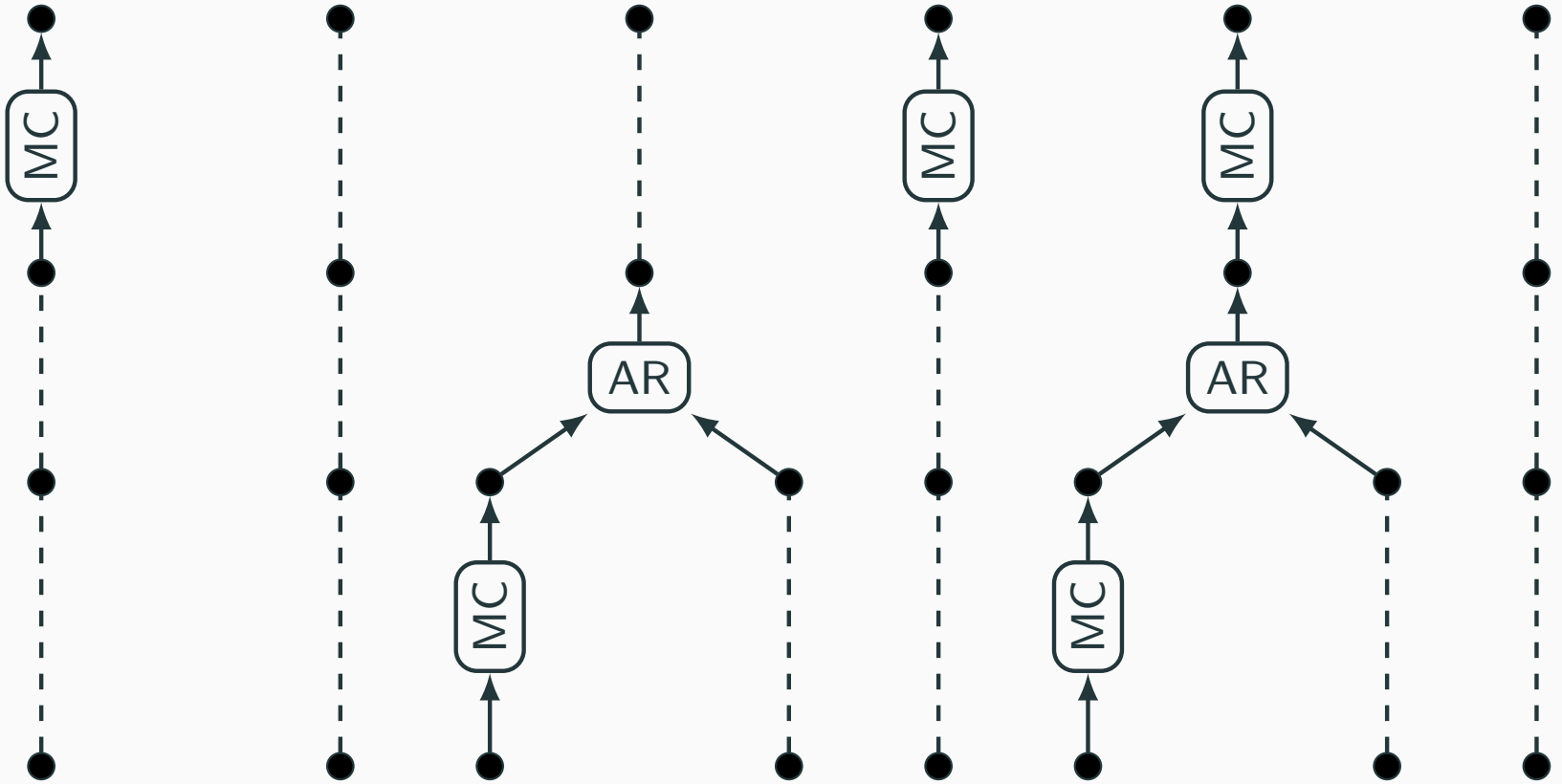
Registers Initialization



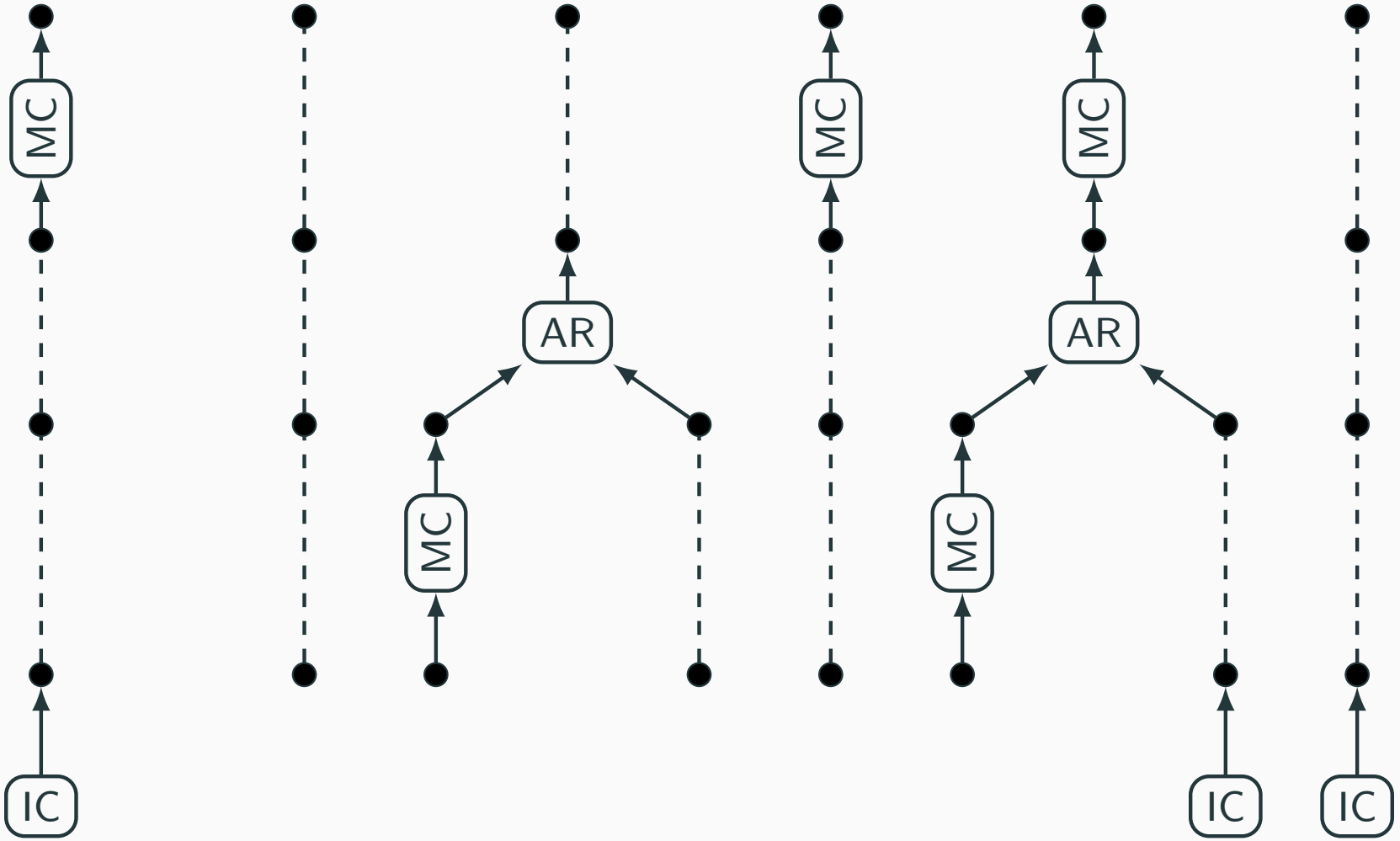
Registers Initialization



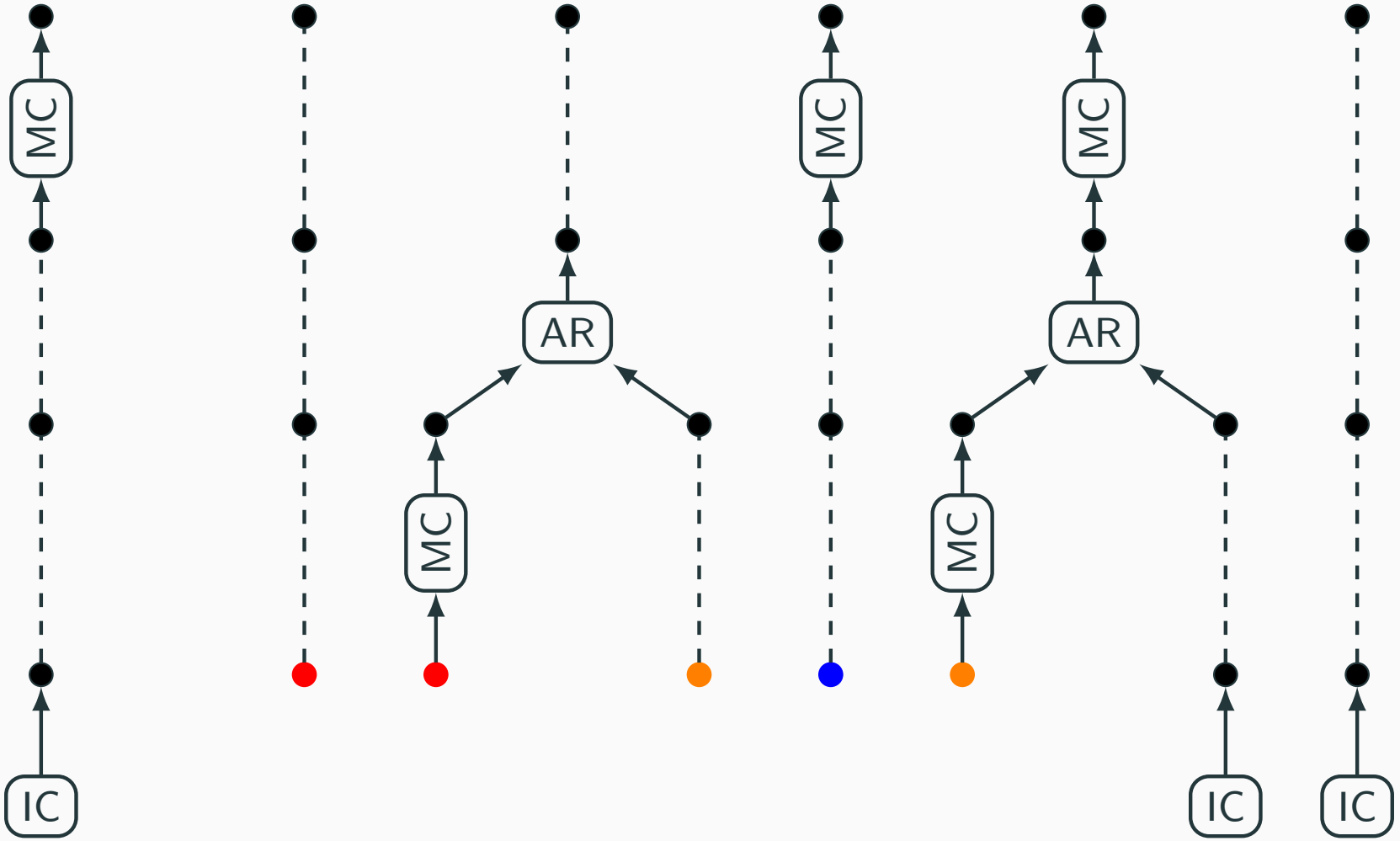
Registers Initialization



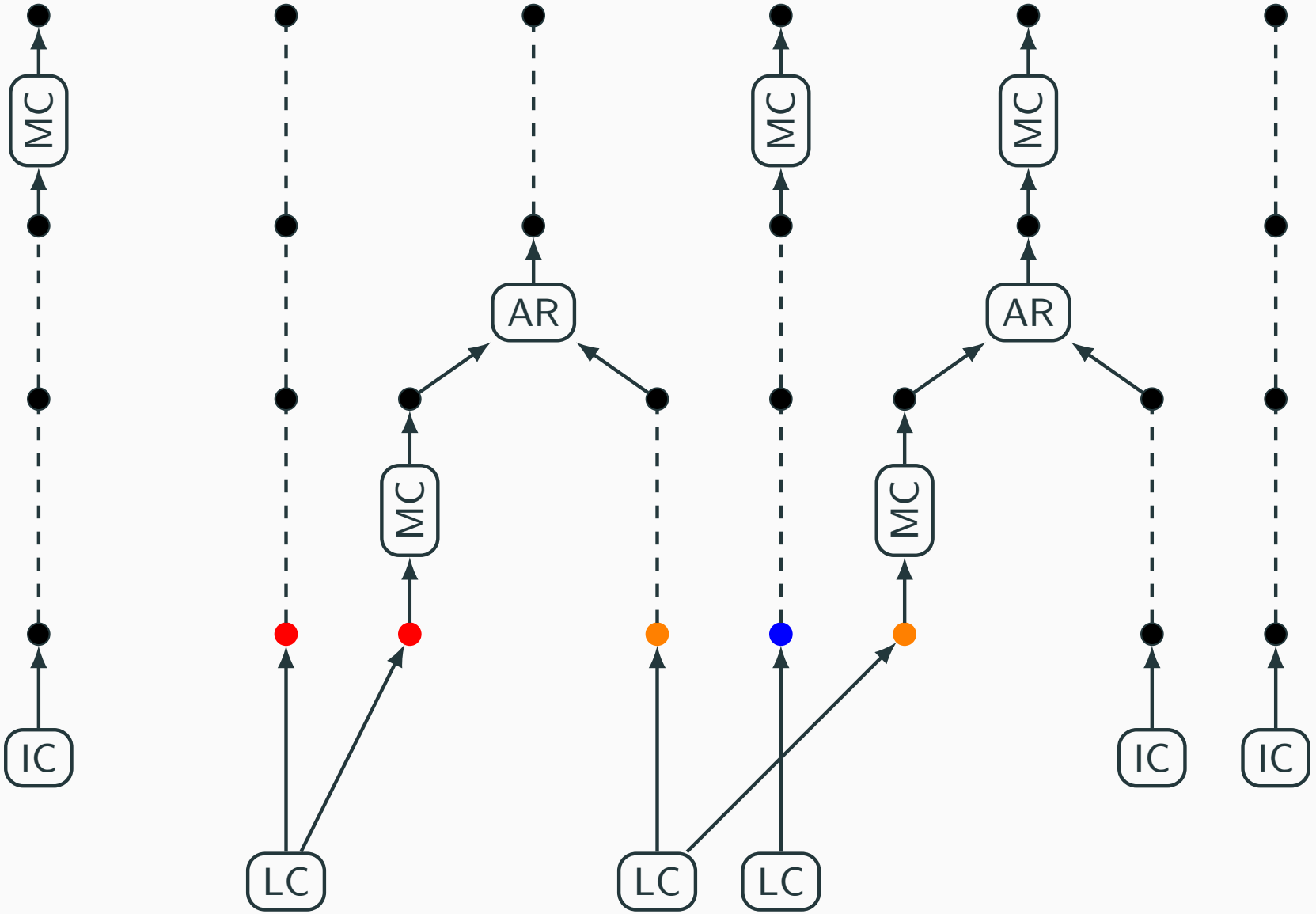
Registers Initialization



Registers Initialization



Registers Initialization



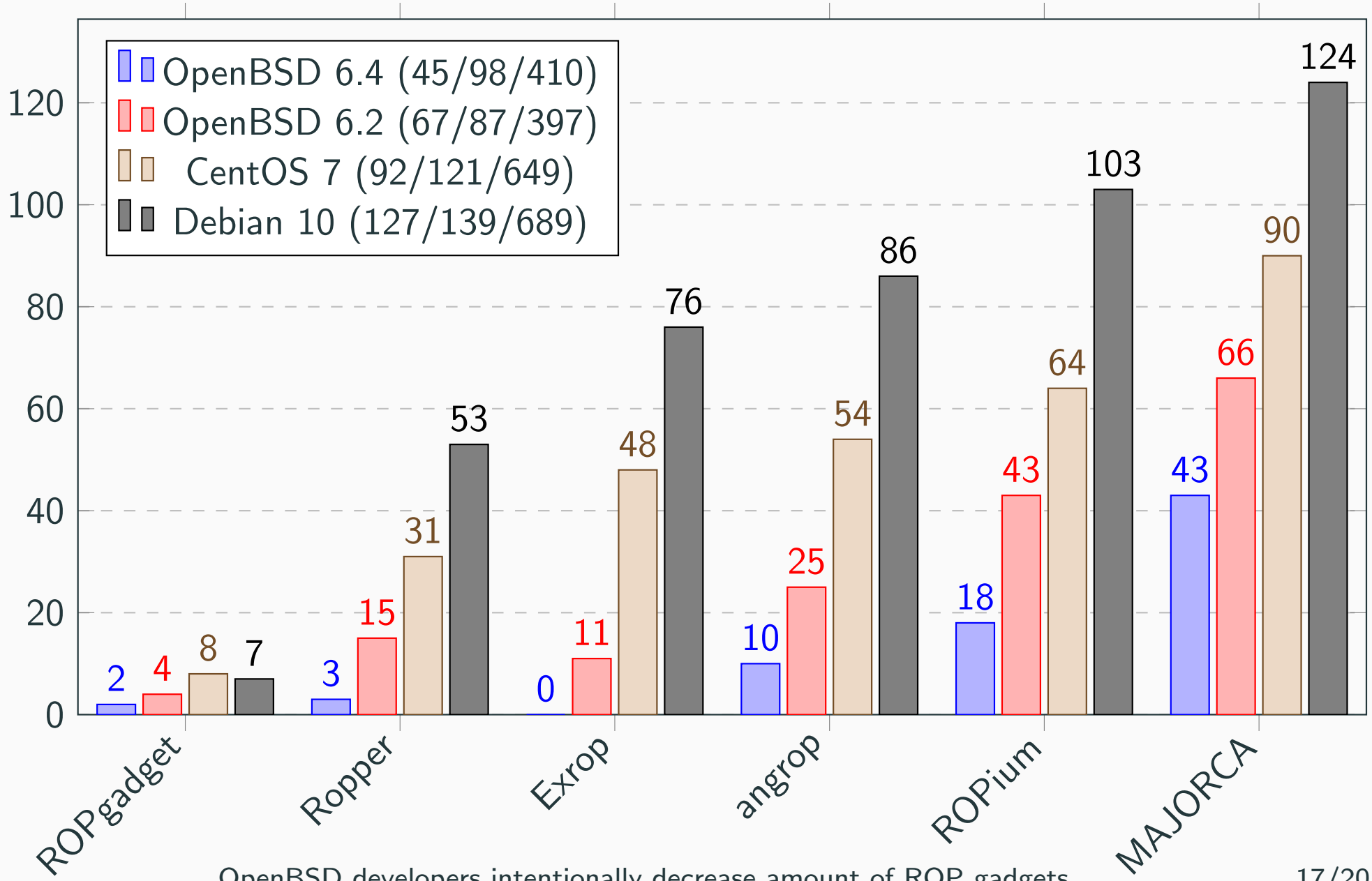
Restricted Symbols Reasoning

- DAG of gadgets must not load (from stack) values containing restricted symbols.
- We use the dynamic programming approach to calculate operands of arithmetic operations (lv и rv), which do not contain restricted symbols
  $lv + rv = value$
  - Two states: presence or absence of carry flag.

(Sys)Call DAG

- Support integers, strings, and arrays of them.
- StoreMem DAGs are created during arguments processing if necessary.
- DAG generators for arguments consider calling convention.
- DAG is extended with vertices that perform (sys)calls.

ROP Benchmark Results (1 hour limit)



OpenBSD developers intentionally decrease amount of ROP gadgets.

MAJORCA Evaluation with Restricted Symbols

Test suite	OpenBSD 6.4			OpenBSD 6.2			Debian 10			CentOS 7		
Number of files	410			397			689			649		
Has syscall gadget	98			87			139			121		
At least one OK	45			67			127			92		
ROP chaining metric	0.46			0.77			0.91			0.76		
Restricted symbols	OK	F	TL	OK	F	TL	OK	F	TL	OK	F	TL
None	43	1	1	66	0	1	124	1	0	90	1	0
slash – `2f`	41	0	3	48	3	13	85	4	34	68	0	22

Other tools generate no workable payloads with restricted symbols.

MAJORCA MIPS Evaluation

- Other tools do not support MIPS.
- 529 tests from 32-bit Malta Linux.
- All tests contain `syscall`.
- MAJORCA generates 112 successful ROP chains (OK).
- Unworkable ROP chains were not generated (F is 0).
- One timeout (TL is 1).

ZSNES 1.51 Linux x86 32-bit, Restricted Symbols: /, \, \0

```
from struct import pack
fill = b'A'  # fill character
chain  = pack('<I', 0x806719a)  # POP EAX ; POP EDI ; RET
chain += pack('<I', 0x91969dd1)
chain += pack('<I', 0x834a860)
chain += pack('<I', 0x807e192)  # NEG EAX ; POP EBX ; RET
chain += 4 * fill
chain += pack('<I', 0x808dbd5)  # MOV DWORD PTR [EDI], EAX ; RET
chain += pack('<I', 0x806719a)  # POP EAX ; POP EDI ; RET
chain += pack('<I', 0xff978cd1)
chain += pack('<I', 0x834a864)
chain += pack('<I', 0x807e192)  # NEG EAX ; POP EBX ; RET
chain += 4 * fill
chain += pack('<I', 0x808dbd5)  # MOV DWORD PTR [EDI], EAX ; RET
chain += pack('<I', 0x807945e)  # POP EBX ; POP EAX ; RET
chain += pack('<I', 0x834a860)
chain += pack('<I', 0xf7c6d8f3)
chain += pack('<I', 0x805318e)  # ADD EAX, 08392718h ; RET
chain += pack('<I', 0x80c6be5)  # XOR ECX, ECX ; RET
chain += pack('<I', 0x81449a0)  # XOR EDX, EDX ; RET
chain += pack('<I', 0x8066984)  # INT 80h  # execve('/bin/sh', 0, 0)
```

Questions?