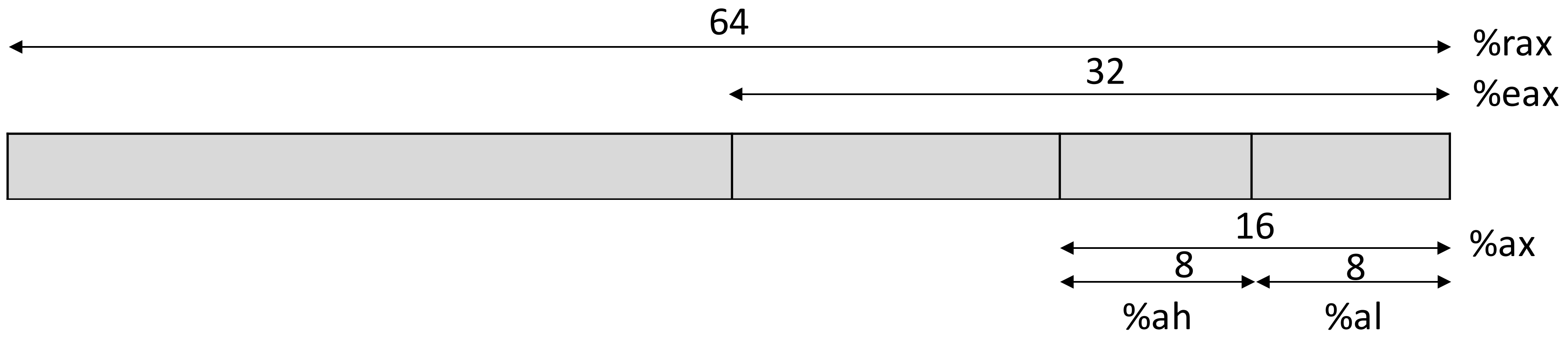
Defining x86-64 Semantics in K

Sandeep Dasgupta

University of Illinois Urbana Champaign

May 23, 2018

Some minor details on nomenclature



addq %src, %dest

movq -8 (%rax, %rbx, 2), %rcx ≡ %rcx ← *(-8 + %rax+%rbx*2)

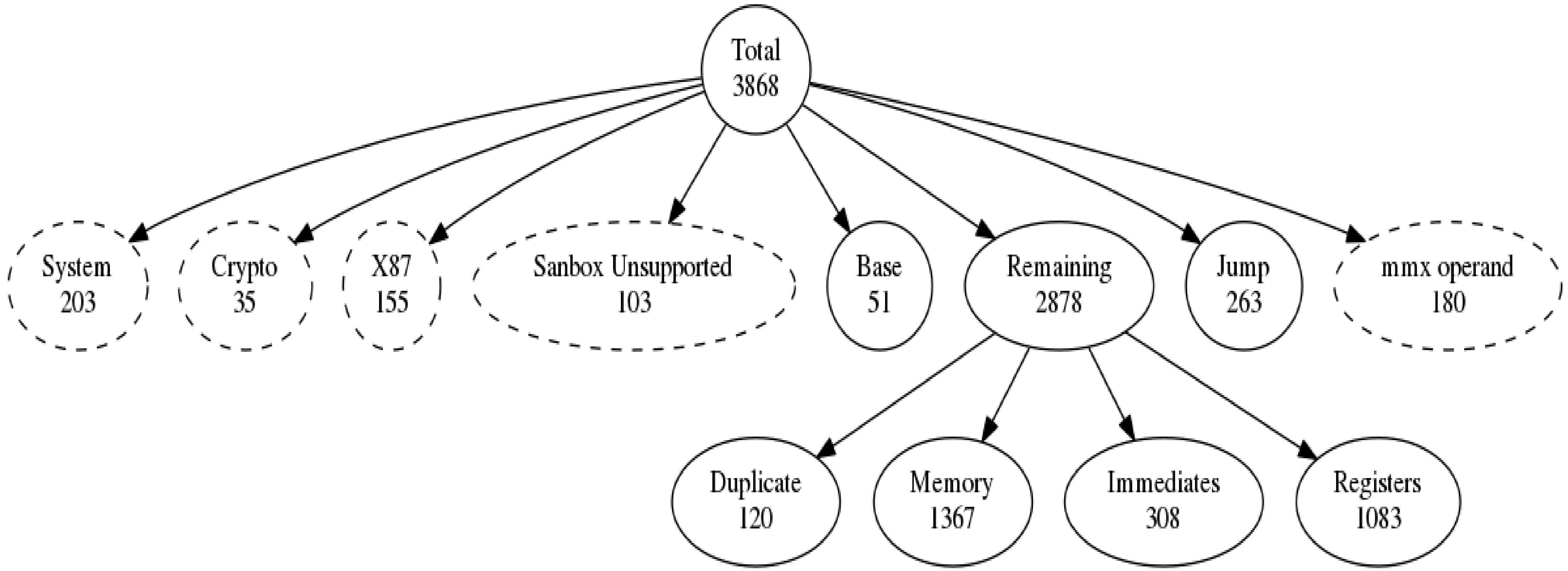
Generic instruction
addq_r64_r64

Concrete instance of above
addq %rax, %rbx

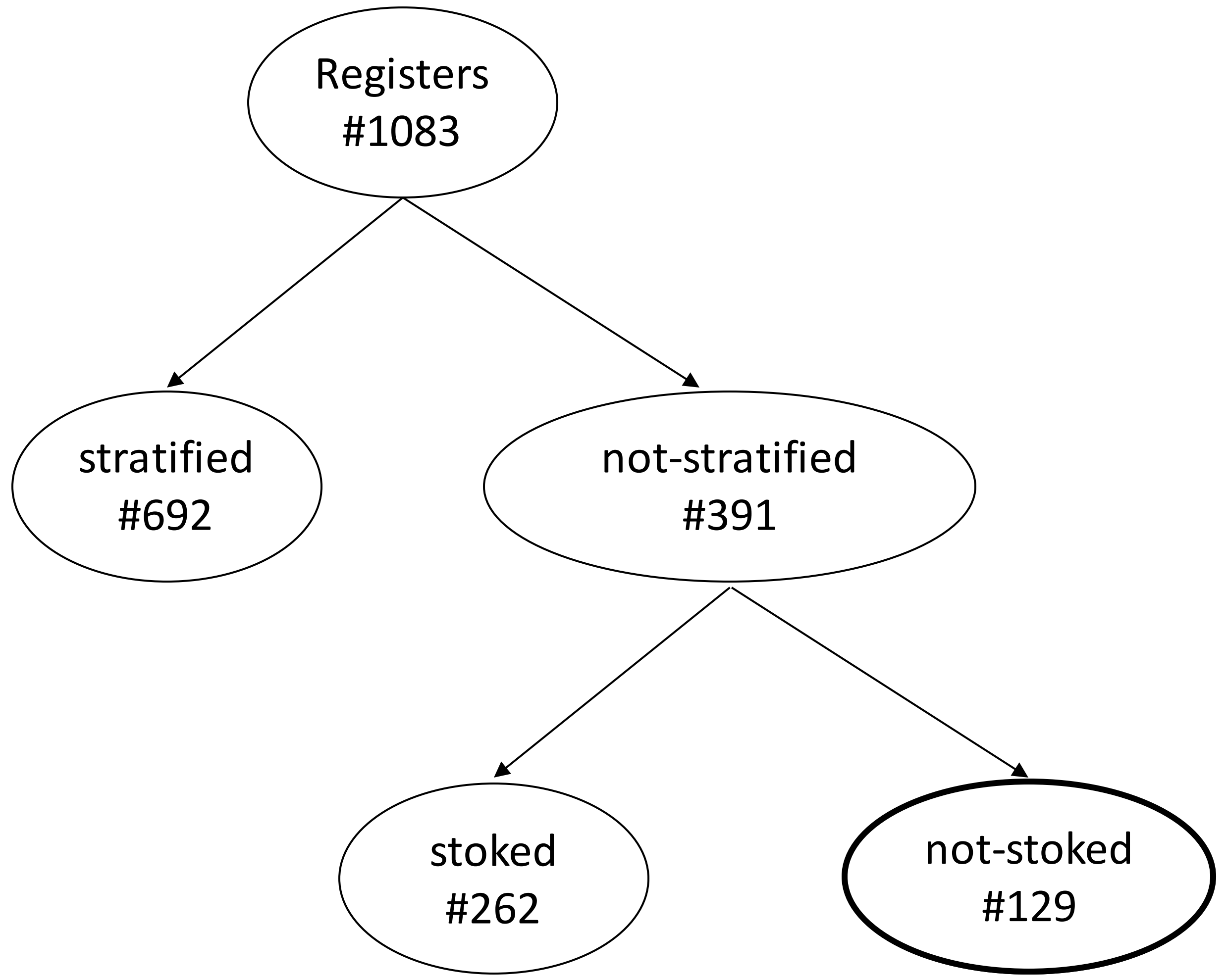
Last presentation: Recap

- Porting stratified register instructions to K rule. 👍
- Supporting the unstratified register instructions.
- Generalizing to Immediate & Memory.

Instruction Status



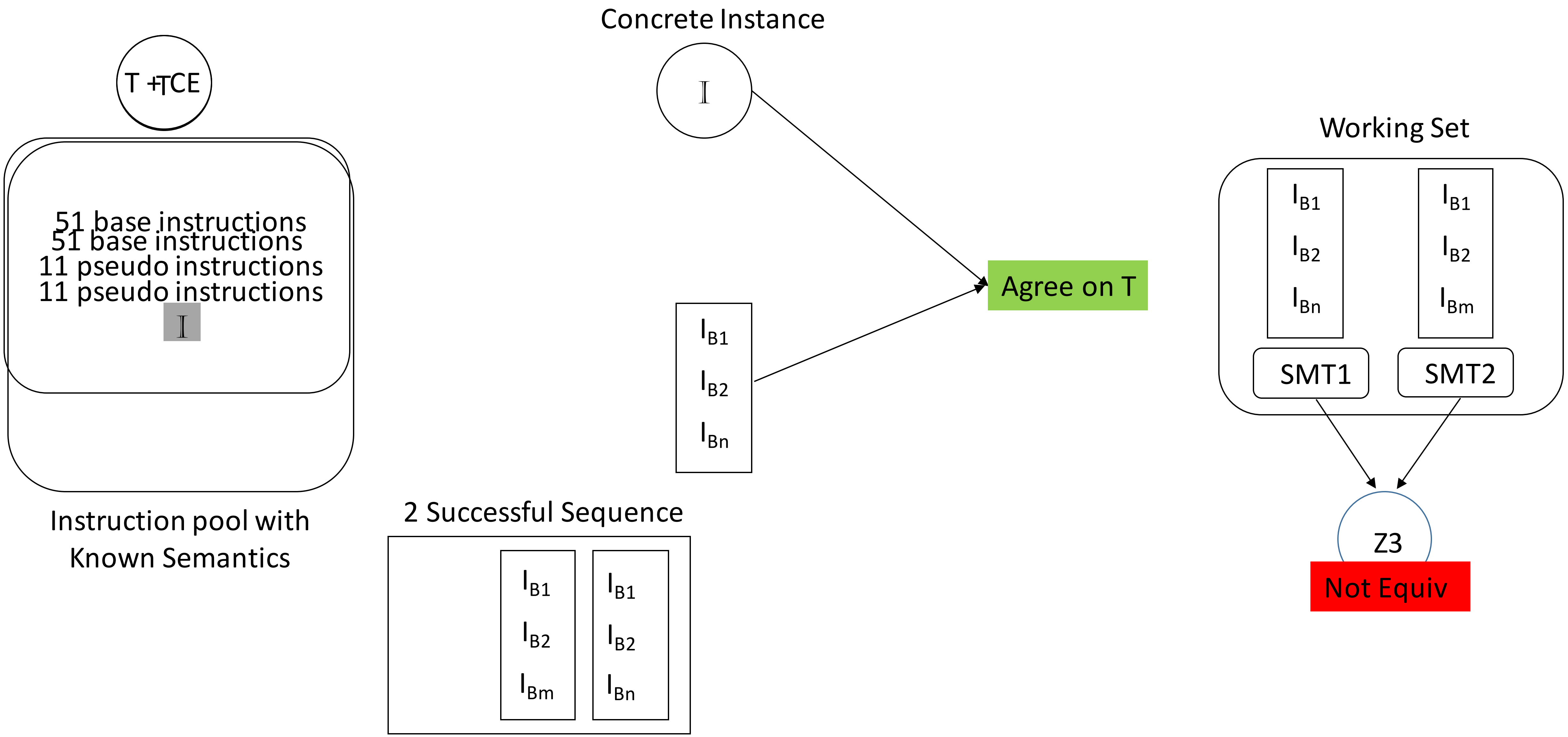
Register Support Status



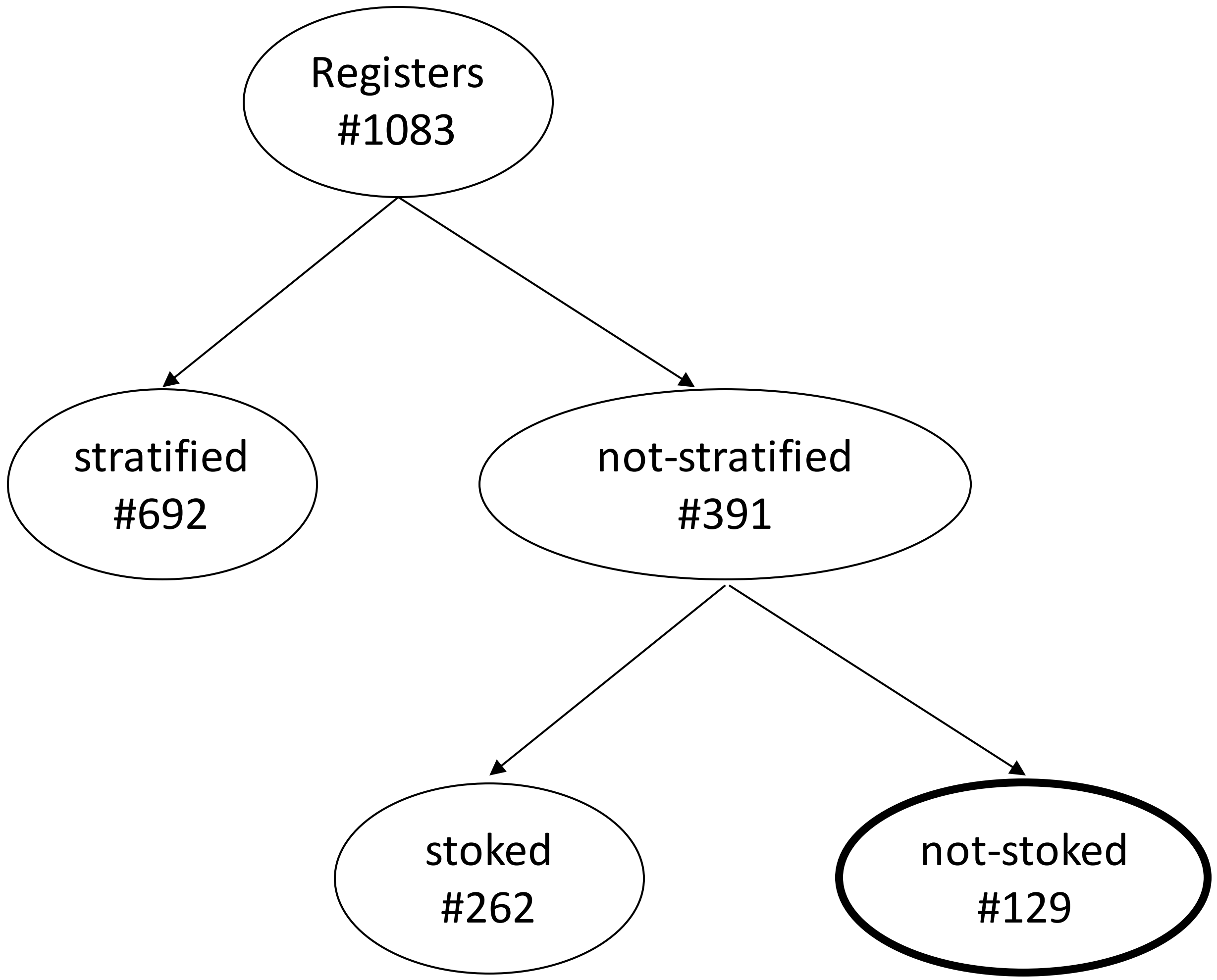
* stoked: Provided by stoke project. Manually written.

* not- stoked: Not provided by stoke project

Recap: Stratified Synthesis



Register Support Status: not-stoked



Extend Stratification

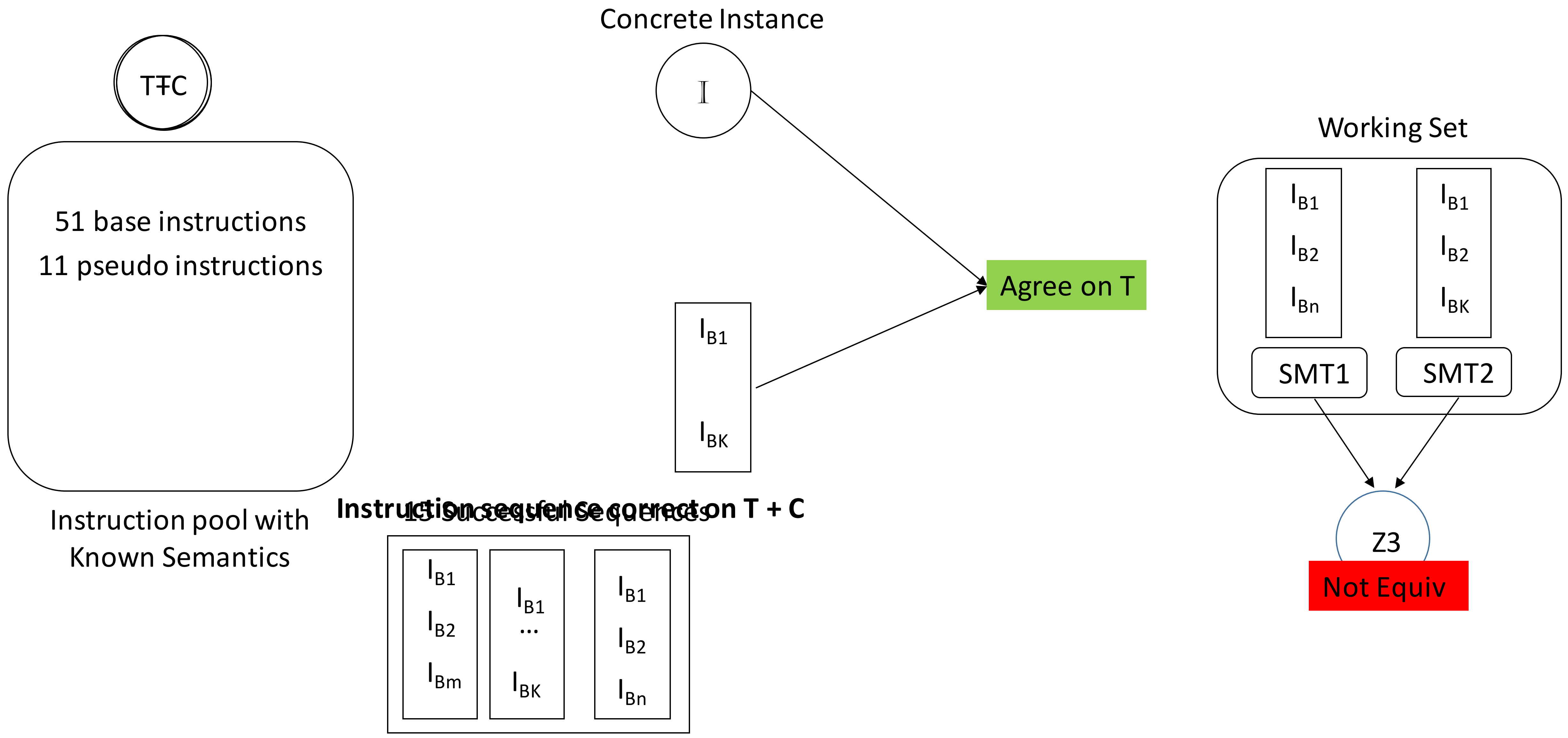
- Issue:
  - Search process is manually assisted.
  - Need insight about instruction semantics.
  - Example

mm

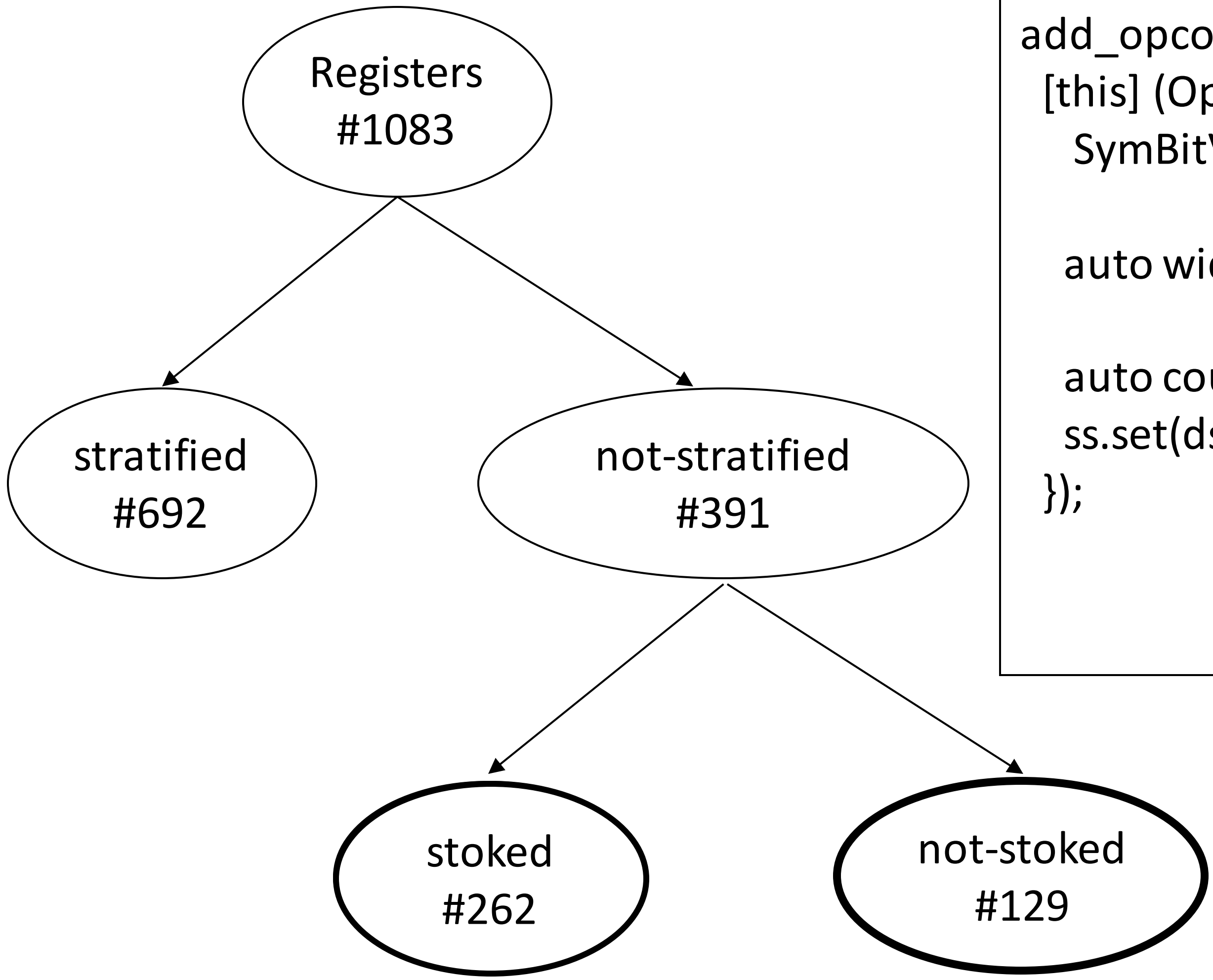
- vmovups_xmm_xmm
- vmaxps_xmm_xmm_xmm

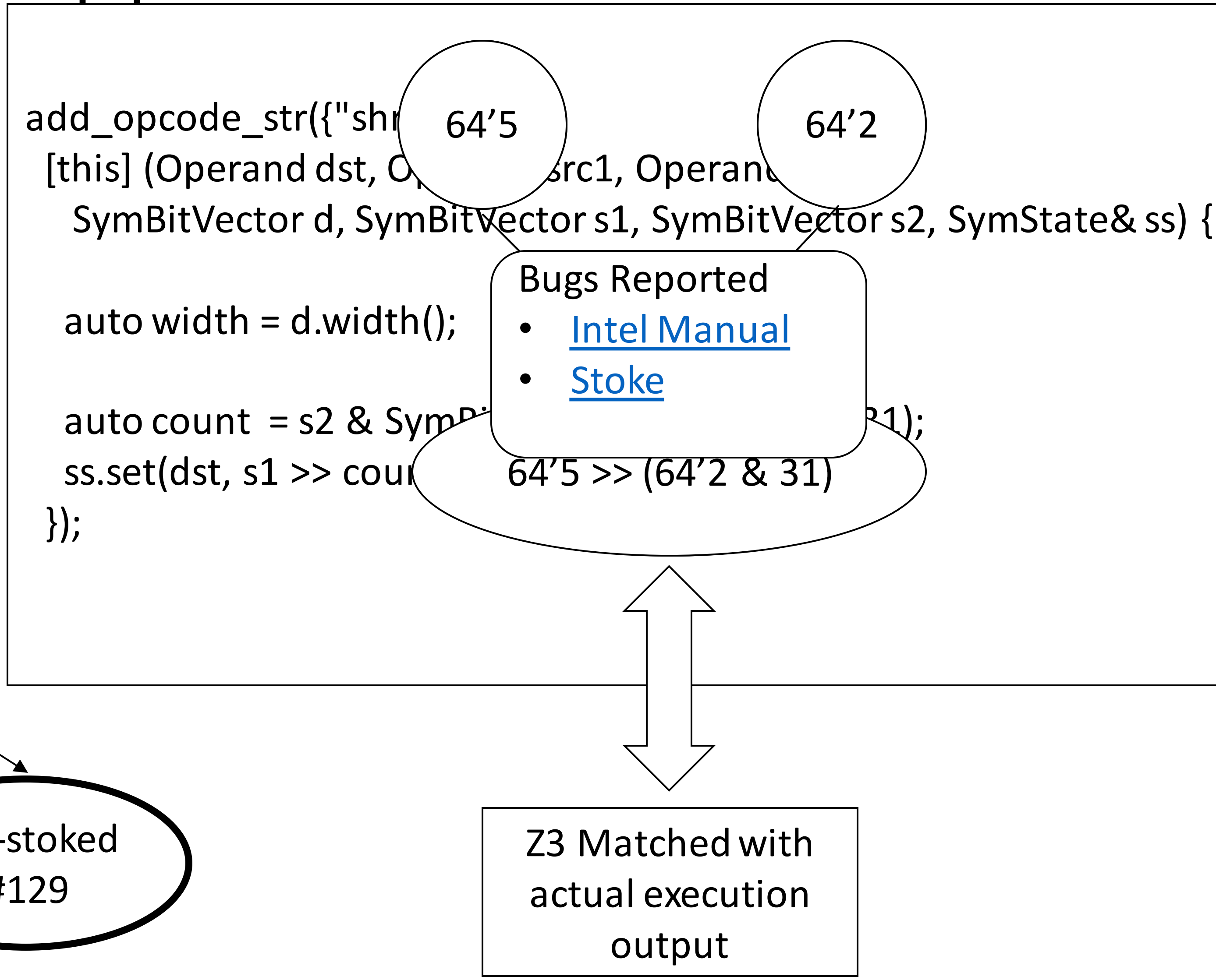
- Example2, Example3

Extending Stratification: Strategy


Concrete Instance
I
T∓C
51 base instructions
11 pseudo instructions
Instruction pool with Known Semantics
Agree on T
I_{B1}
I_{BK}
Instruction sequence correct on T + C
15 Successful Sequences
I_{B1} I_{B2} I_{Bm}
I_{B1} … I_{BK}
I_{B1} I_{B2} I_{Bn}
Working Set
I_{B1} I_{B2} I_{Bn}
I_{B1} I_{B2} I_{BK}
SMT1
SMT2
Z3
Not Equiv

Register Support Status





```
add_opcode_str({"shr
 [this] (Operand dst, O          src1, Operan
   SymBitVector d, SymBitVector s1, SymBitVector s2, SymState& ss) {

  auto width = d.width();

  auto count  = s2 & SymB                                  1);
  ss.set(dst, s1 >> cou
});
```

Bugs Reported

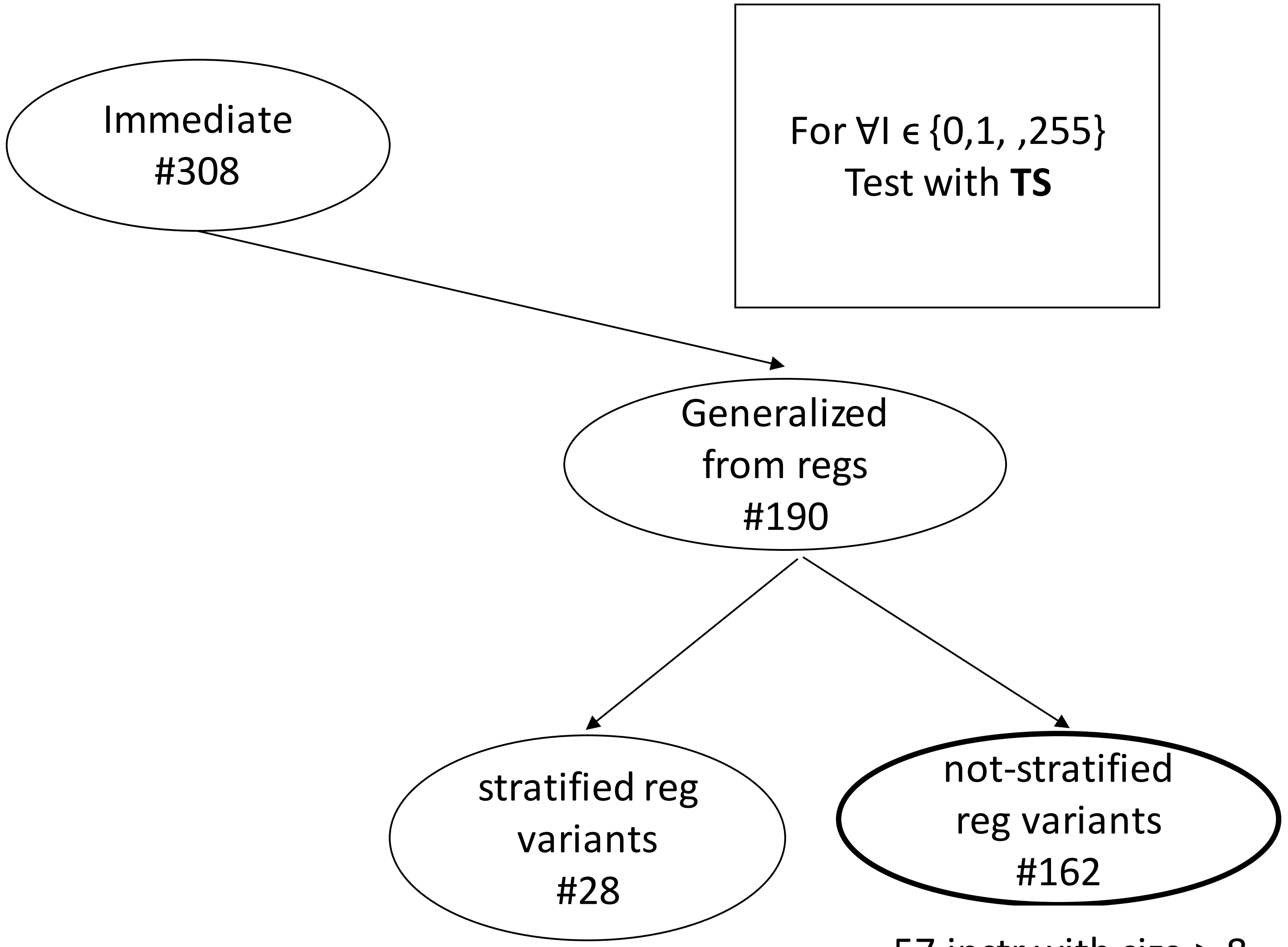
- [Intel Manual]
- [Stoke]

Z3 Matched with actual execution output

Immediate Support Status

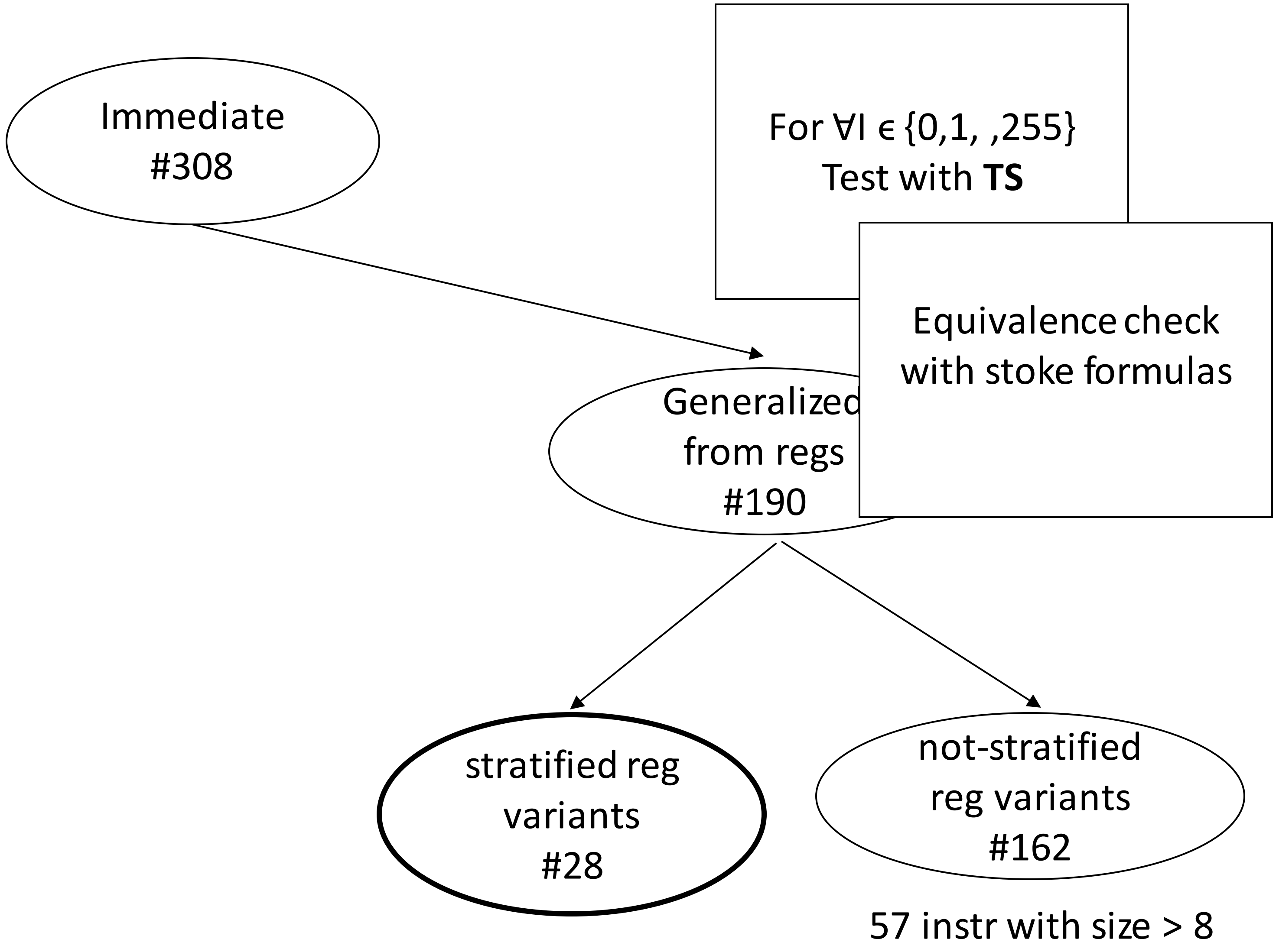
Immediate
#308

Immediate Support Status

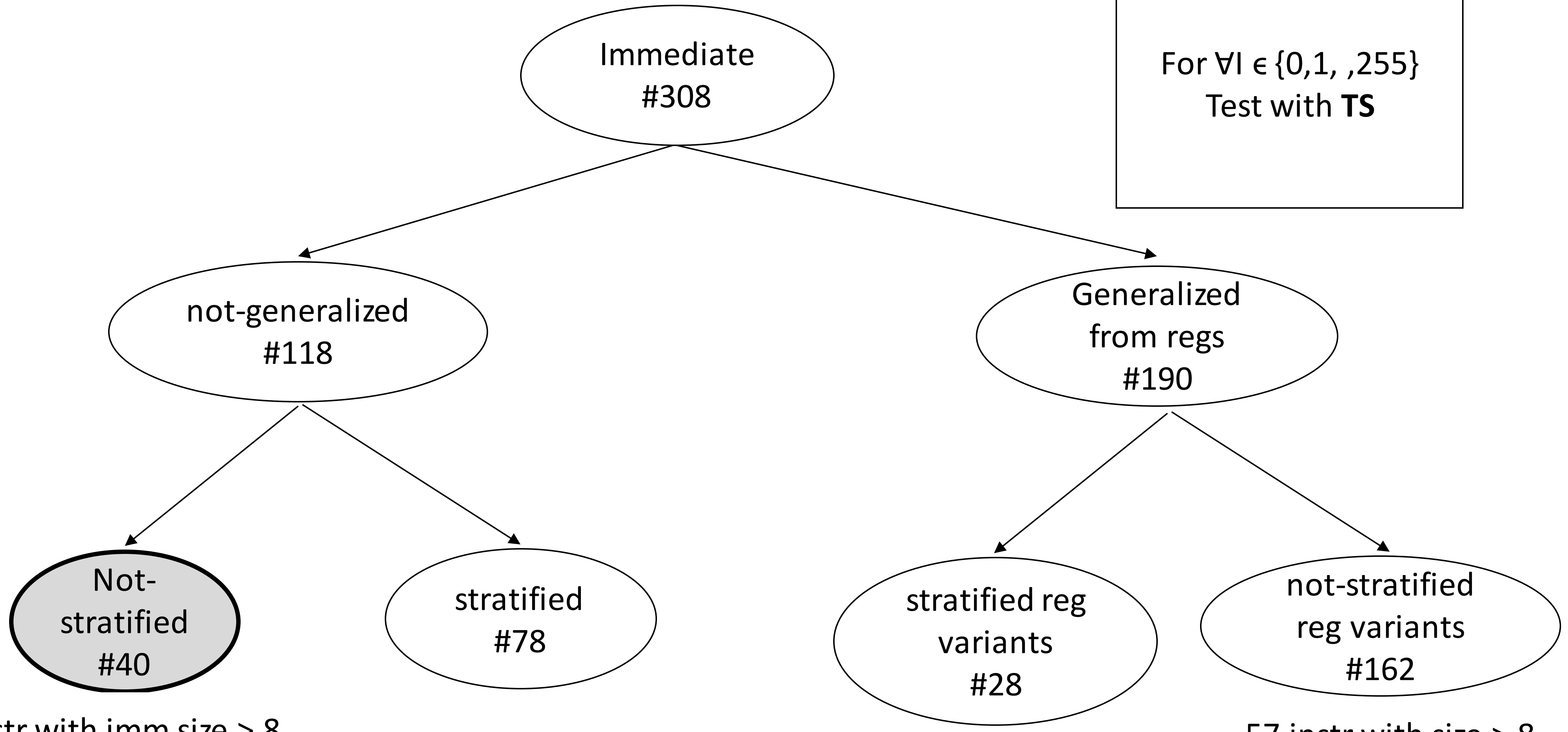


57 instr with size > 8

Immediate Support Status



Immediate Support Status

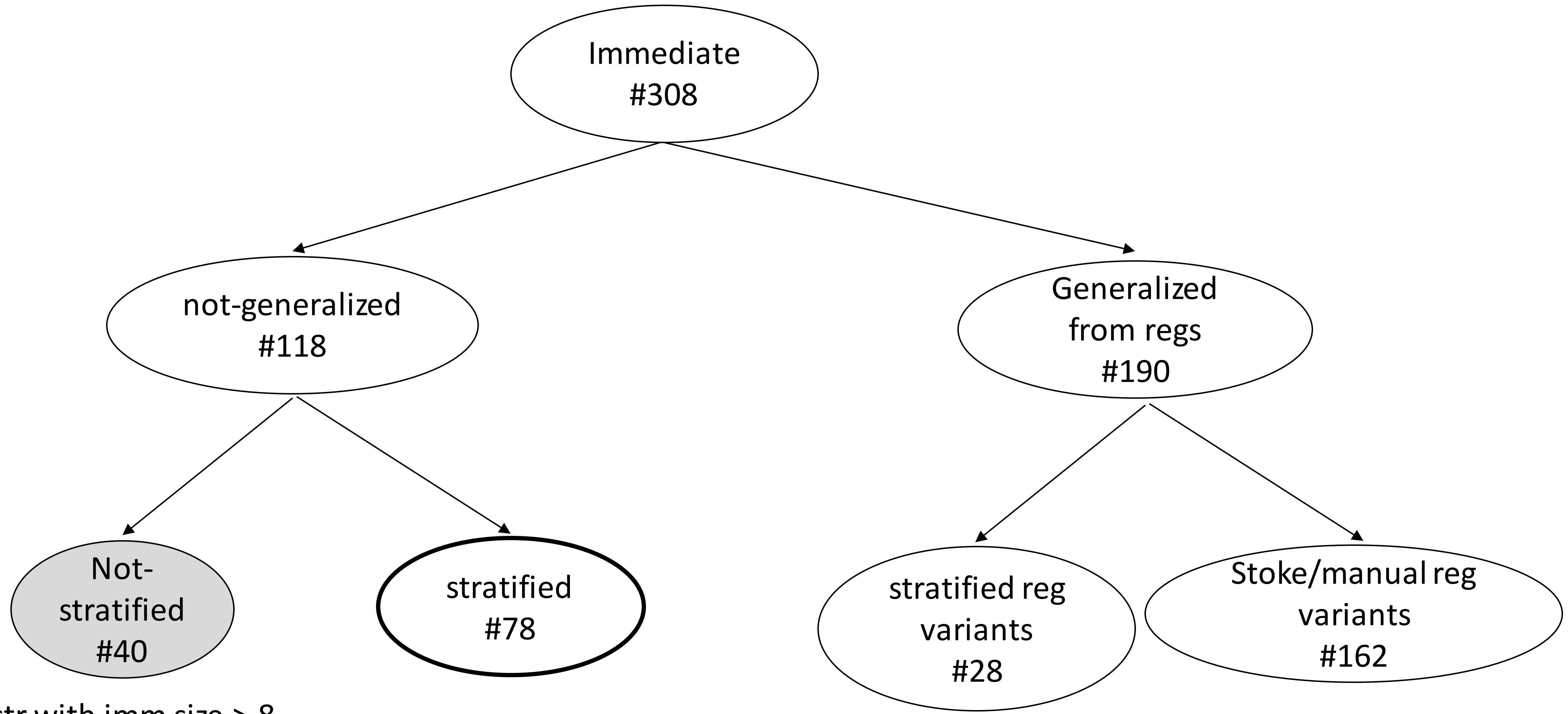

Immediate #308
not-generalized #118
Generalized from regs #190
Not-stratified #40
stratified #78
stratified reg variants #28
not-stratified reg variants #162

For ∀I ϵ {0,1, ,255}
Test with **TS**

5 instr with imm size > 8

57 instr with size > 8

Immediate Support Status



5 instr with imm size > 8

5 instr with size > 57

Immediate Support Status

Non-generalized & stratified #78

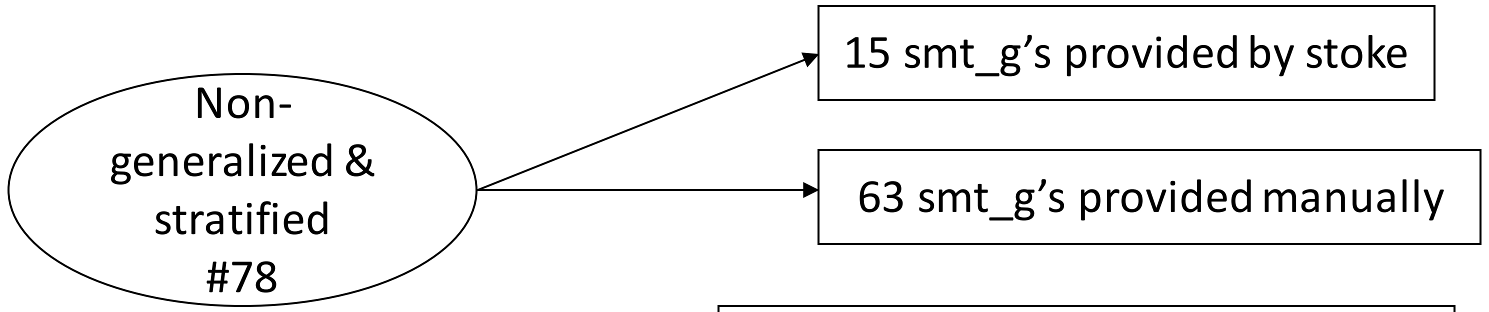
vpshuflw_xmm_xmm_imm8

vpshuflw_xmm_xmm_0 $\rightarrow$ smt_0
vpshuflw_xmm_xmm_1 $\rightarrow$ smt_1
…
vpshuflw_xmm_xmm_255 $\rightarrow$ smt_255

∃ **smt_g**:
!(smt_0 == smt_g(0)) is **unsat**
!(smt_1 == smt_g(1)) is **unsat**
…
!(smt_255 == smt_g(255)) is **unsat**

**** smt_g(I) is the SMT formula obtained by concretizing the symbolic inputs to I**

Immediate Support Status


Non-generalized & stratified #78
15 smt_g's provided by stoke
63 smt_g's provided manually

vpshuflw_xmm_xmm_imm8

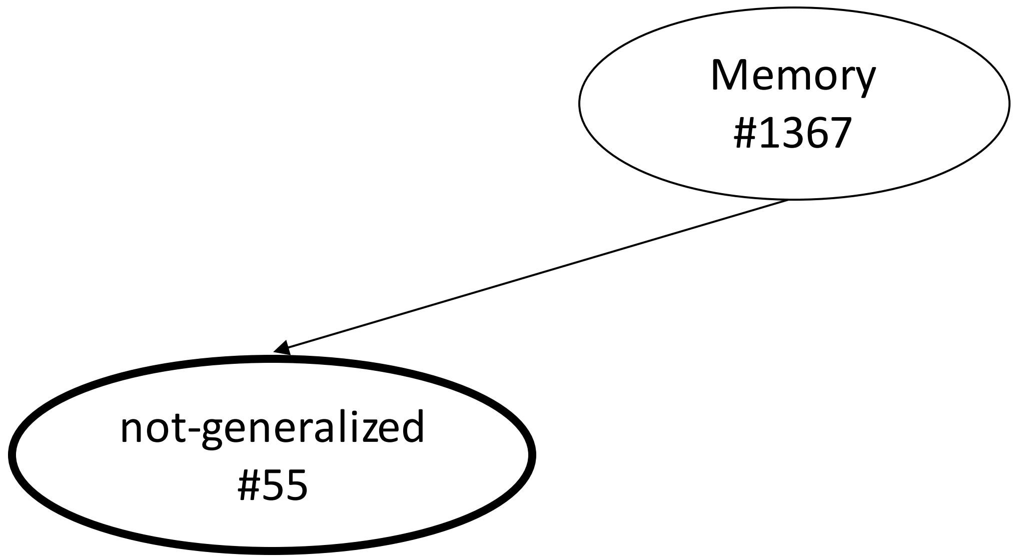
vpshuflw_xmm_xmm_0 → smt_0
vpshuflw_xmm_xmm_3 → smt_1
…
vpshuflw_xmm_xmm_254 → smt_255

∃ **smt_g**:
!(smt_0 == smt_g(0)) is **unsat**
!(smt_3 == smt_g(3)) is **unsat**
!(smt_254 == smt_g(254)) is **unsat**

and

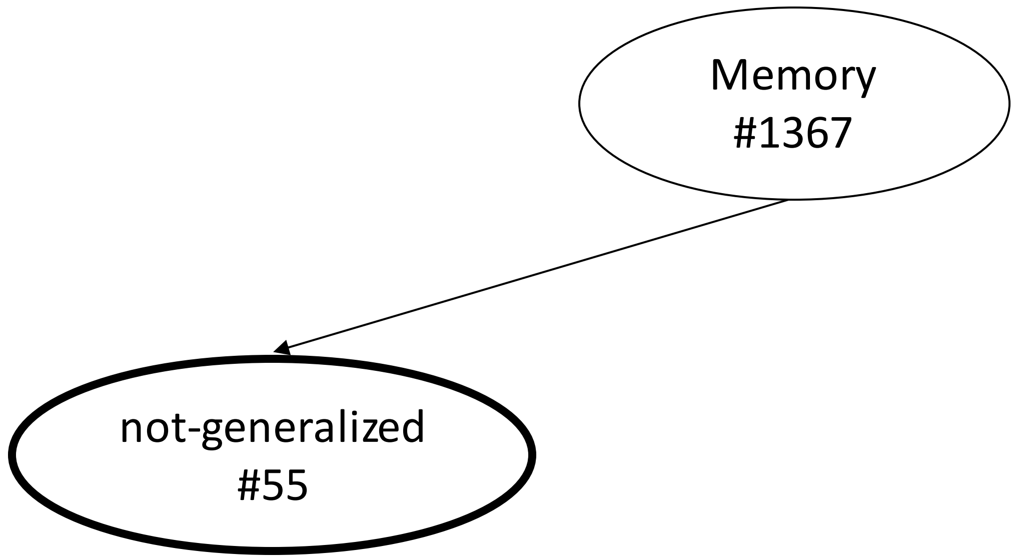
TEST(smt_g(1) == TS)
TEST(smt_g(2) == TS)
…
TEST(smt_g(255) == TS)

Memory Support Status

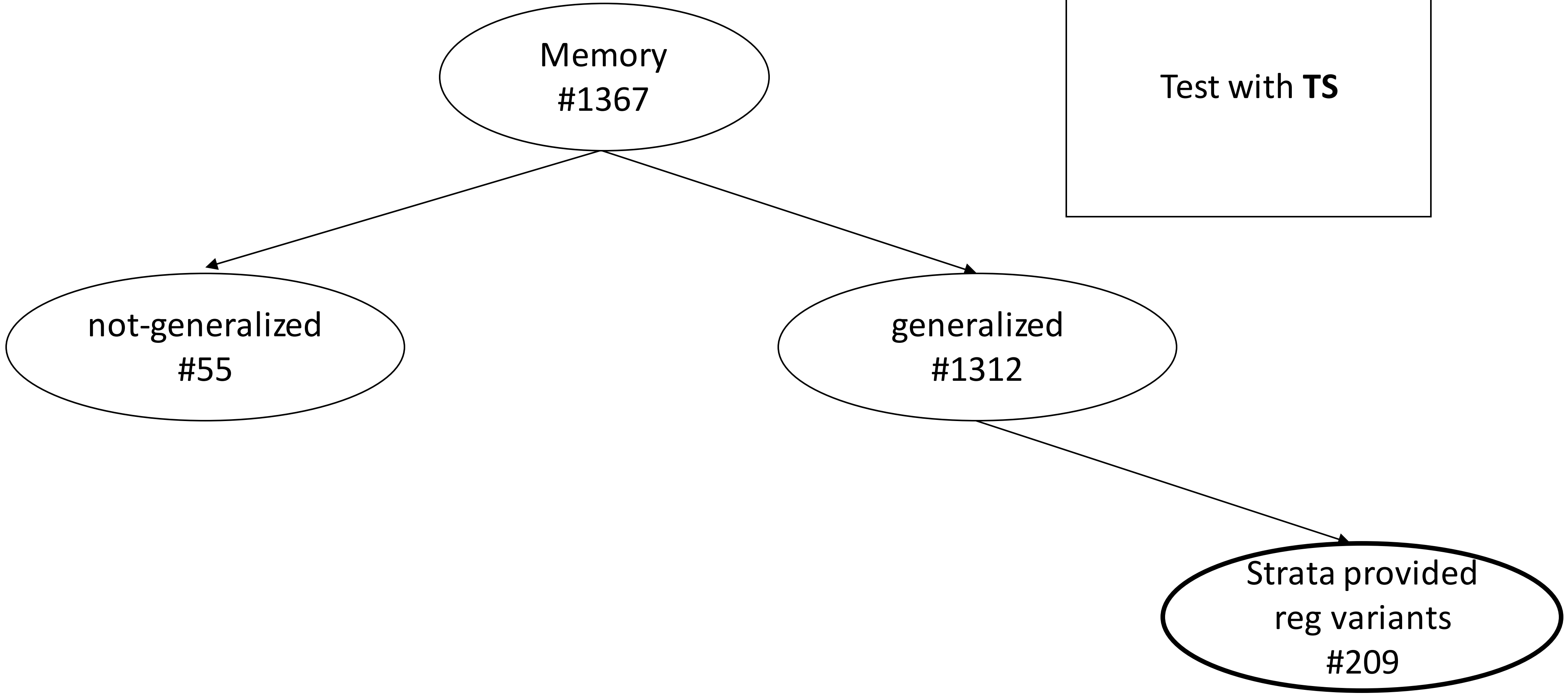


Test with **TS**

Memory Support Status

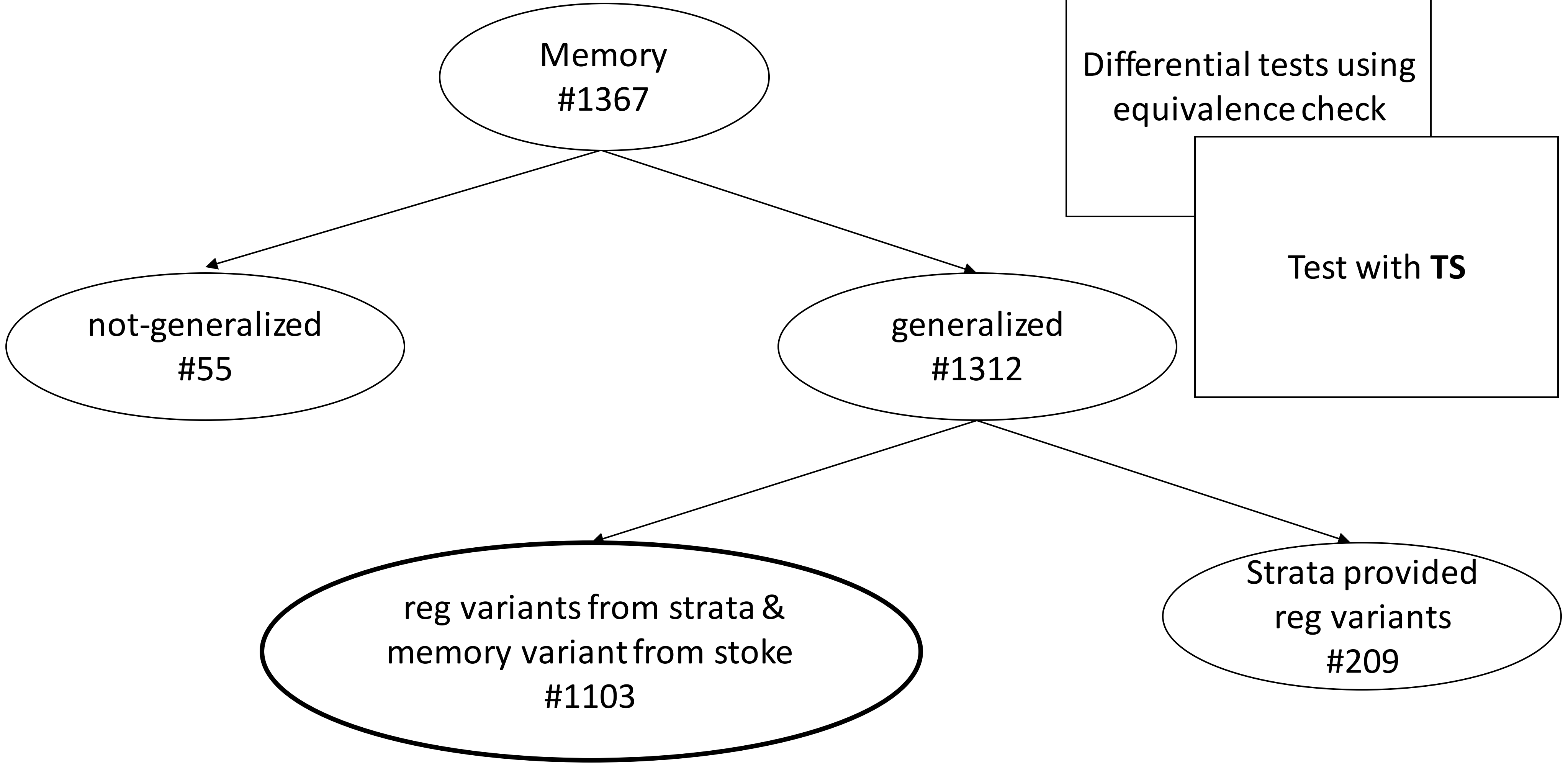


Memory Support Status



Test with **TS**

Memory Support Status



Generalization from to Memory not always correct

movsd %ymm2, %ymm1

Strata Register Variant
%ymm1 : $\%ymm1[255:128] \circ (\%ymm1[127:64] \circ \%ymm2[63:0])$

movsd (%ymm2), %ymm1

Expected Memory Variant from generalization
%ymm1 : $\%ymm1[255:128] \circ (\%ymm1[127:64] \circ \text{memory\_read\_val})$

movsd (%ymm2), %ymm1

Correct Memory Variant (from Stoke)
%ymm1 : $\%ymm1[255:128] \circ (0x0_{64} \circ \text{memory\_read\_val})$

Problem with testing

vaddpd %xmm3, %xmm2, %xmm1

%ymm1 : $0x0_{64} \circ \circ$ ($0x0_{64} \circ$ (add_double(%ymm2[127:64], %ymm3[127:64]) $\circ$ add_double(%ymm2[63:0], %ymm3[63:0])))

Porting Strata formula to K rule

movsd %xmm2, %xmm1

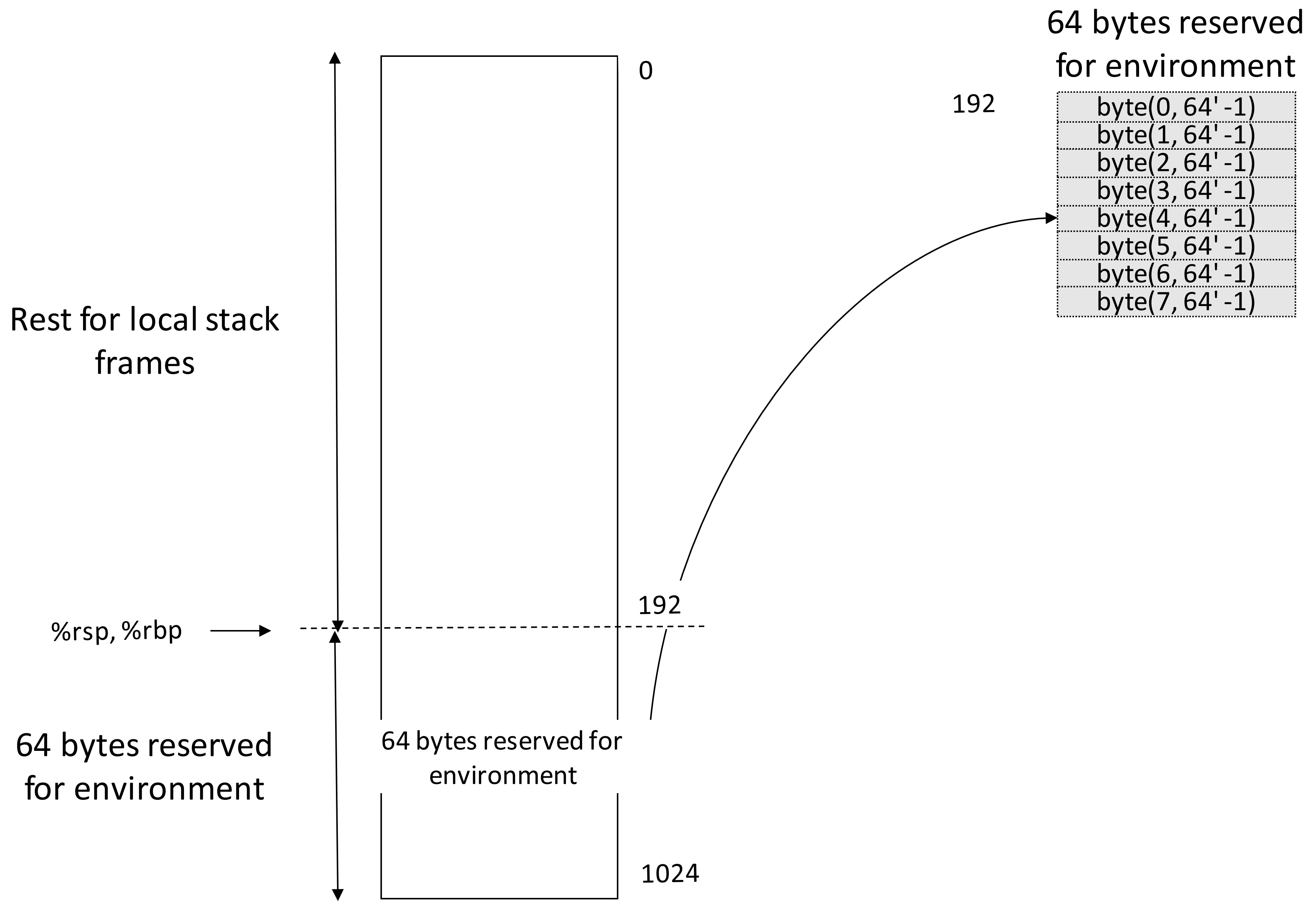
Strata K Formula: movsd %xmm2, %xmm1

```
%ymm1  : concatenateMInt( extractMInt( %ymm1, 0, 128), concatenateMInt( extractMInt( %ymm1,
                          128, 192), extractMInt( %ymm2, 192, 256)))
```

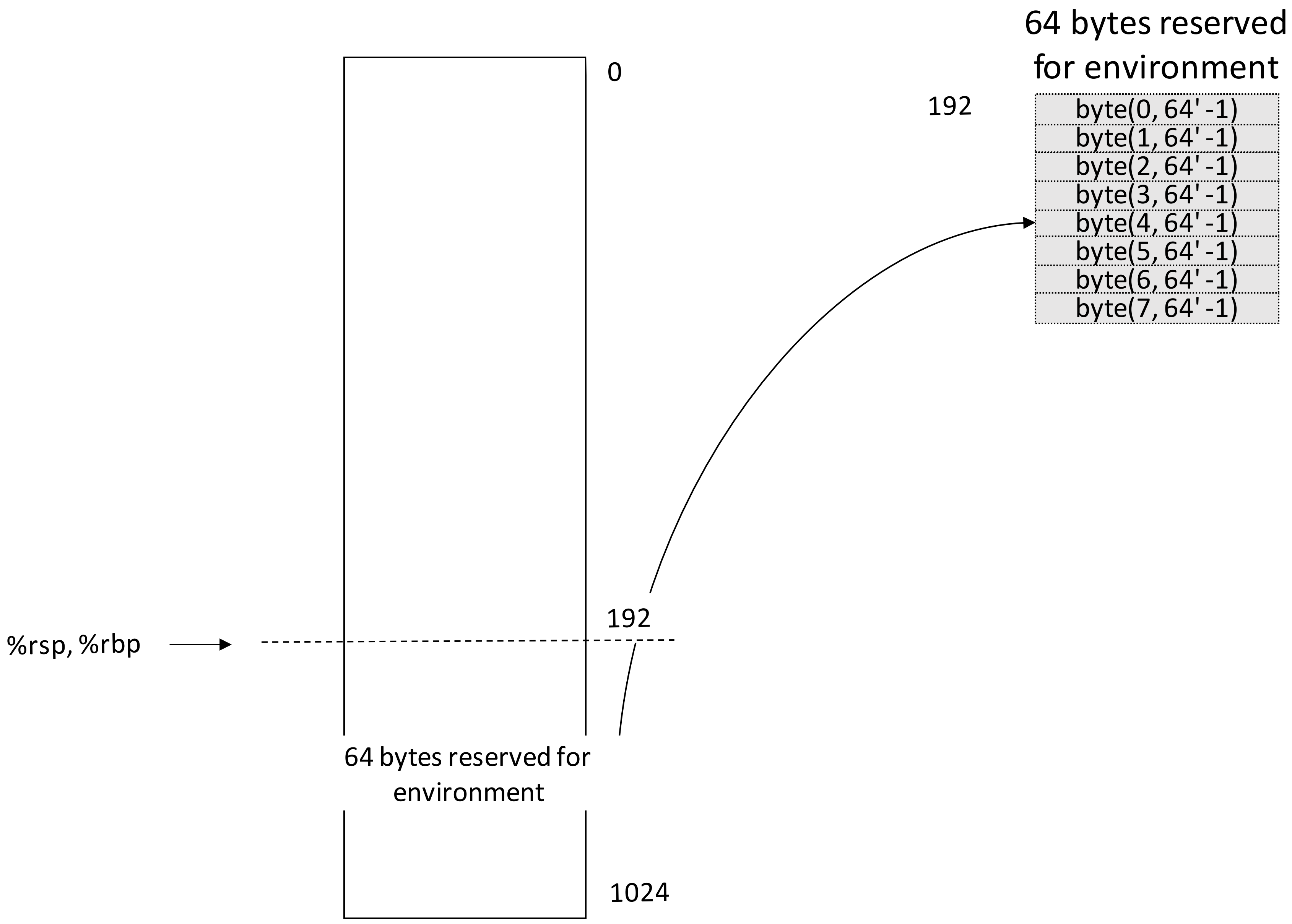
Generalized K Rule: movsd X2, X1

```
rule
  <k> execinstr (movsd R1:Xmm, R2:Xmm, .Operands) => . ...</k>
    <regstate>
      RSMap:Map => updateMap(RSMap,
         convToRegKeys(R2) |-> concatenateMInt( extractMInt( getParentValue(R2, RSMap), 0, 192),
extractMInt( getParentValue(R1, RSMap), 192, 256)))
    </regstate>
```

Memory Semantics



Memory Semantics

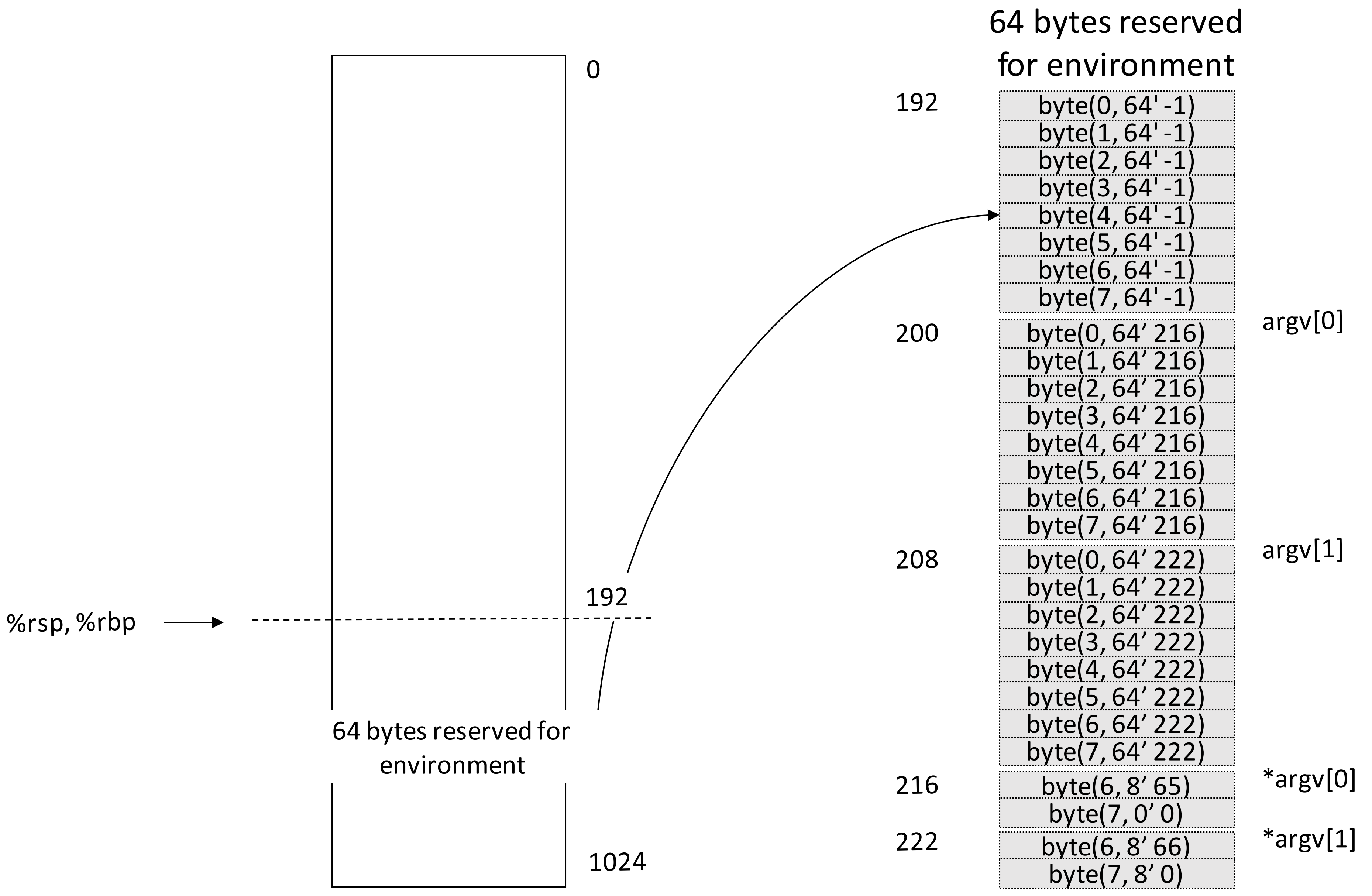


```
$ krun prog.s “A" “B"
pushq %rbp
movq %rsp, %rbp
movq 0x01020304, -8(%rbp)
movb  (%rsi), %rax
movb  (%rax), %bl
addw -6(%rbp), %bx
movq %rbp, %rsp
popq %rbp
```

Memory Semantics

%rdi: 2
%rsi: 200

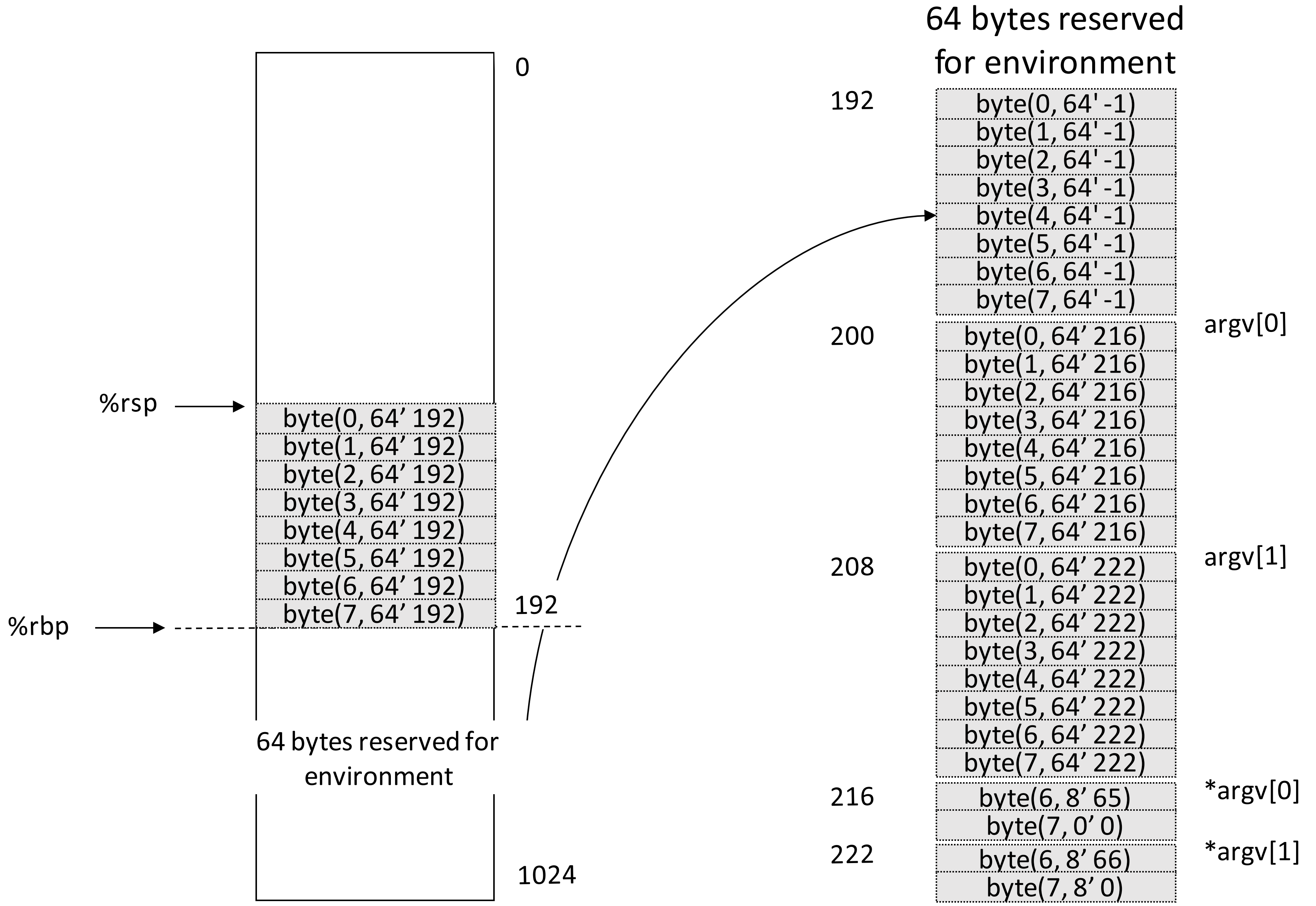
```
$ krun prog.s "A" "B"
pushq %rbp
movq %rsp, %rbp
movq 0x01020304, -8(%rbp)
movb  (%rsi), %rax
movb  (%rax), %bl
addw -6(%rbp), %bx
movq %rbp, %rsp
popq %rbp
```



Memory Semantics

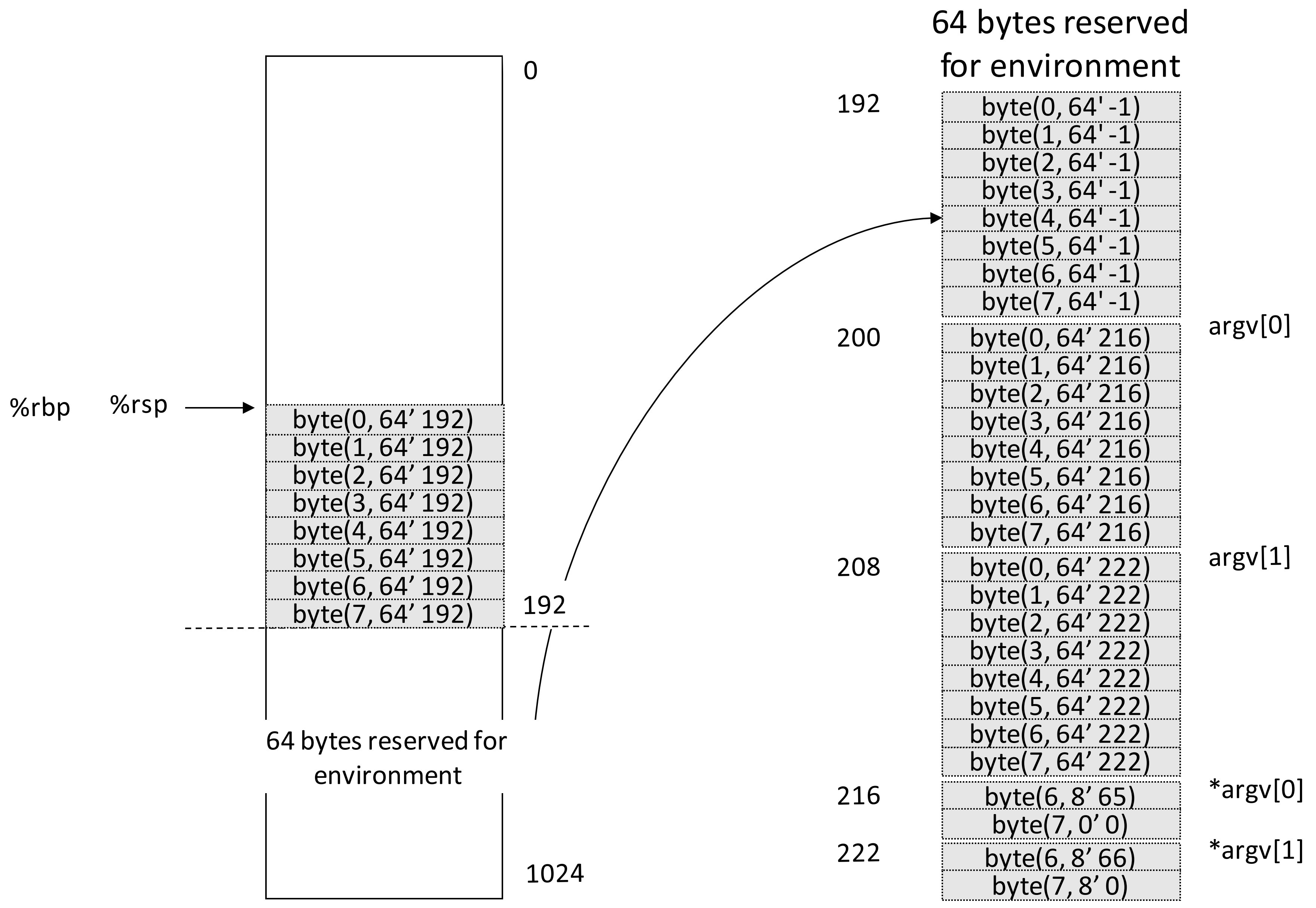
%rdi: 2
%rsi: 200

```
$ krun prog.s “A" “B"
pushq %rbp
movq %rsp, %rbp
movq 0x01020304, -8(%rbp)
movb  (%rsi), %rax
movb  (%rax), %bl
addw -6(%rbp), %bx
movq %rbp, %rsp
popq %rbp
```



Memory Semantics

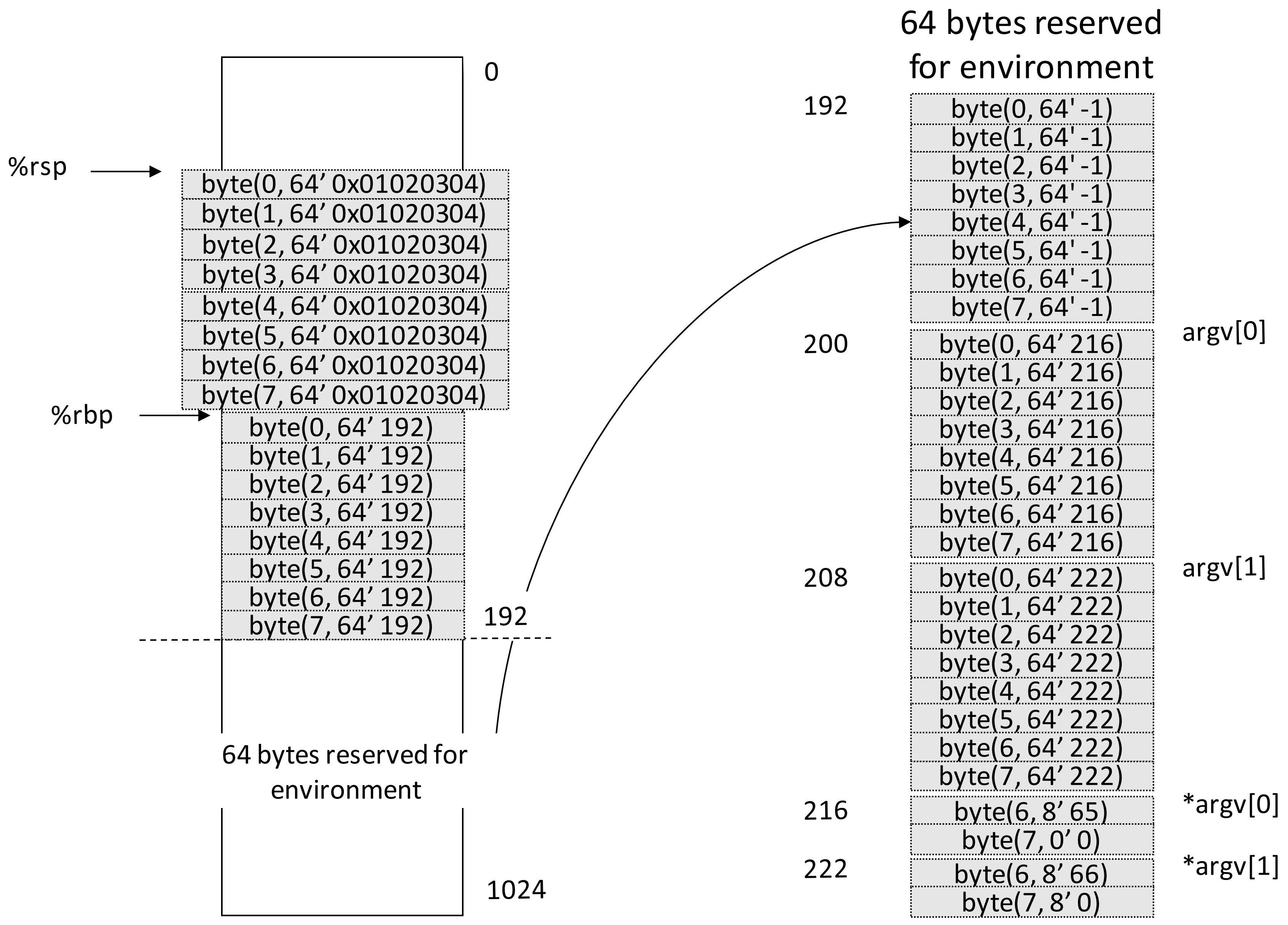
%rdi: 2
%rsi: 200



```
$ krun prog.s “A" “B"
pushq %rbp
movq %rsp, %rbp
movq 0x01020304, -8(%rbp)
movb  (%rsi), %rax
movb  (%rax), %bl
addw -6(%rbp), %bx
movq %rbp, %rsp
popq %rbp
```

Memory Semantics

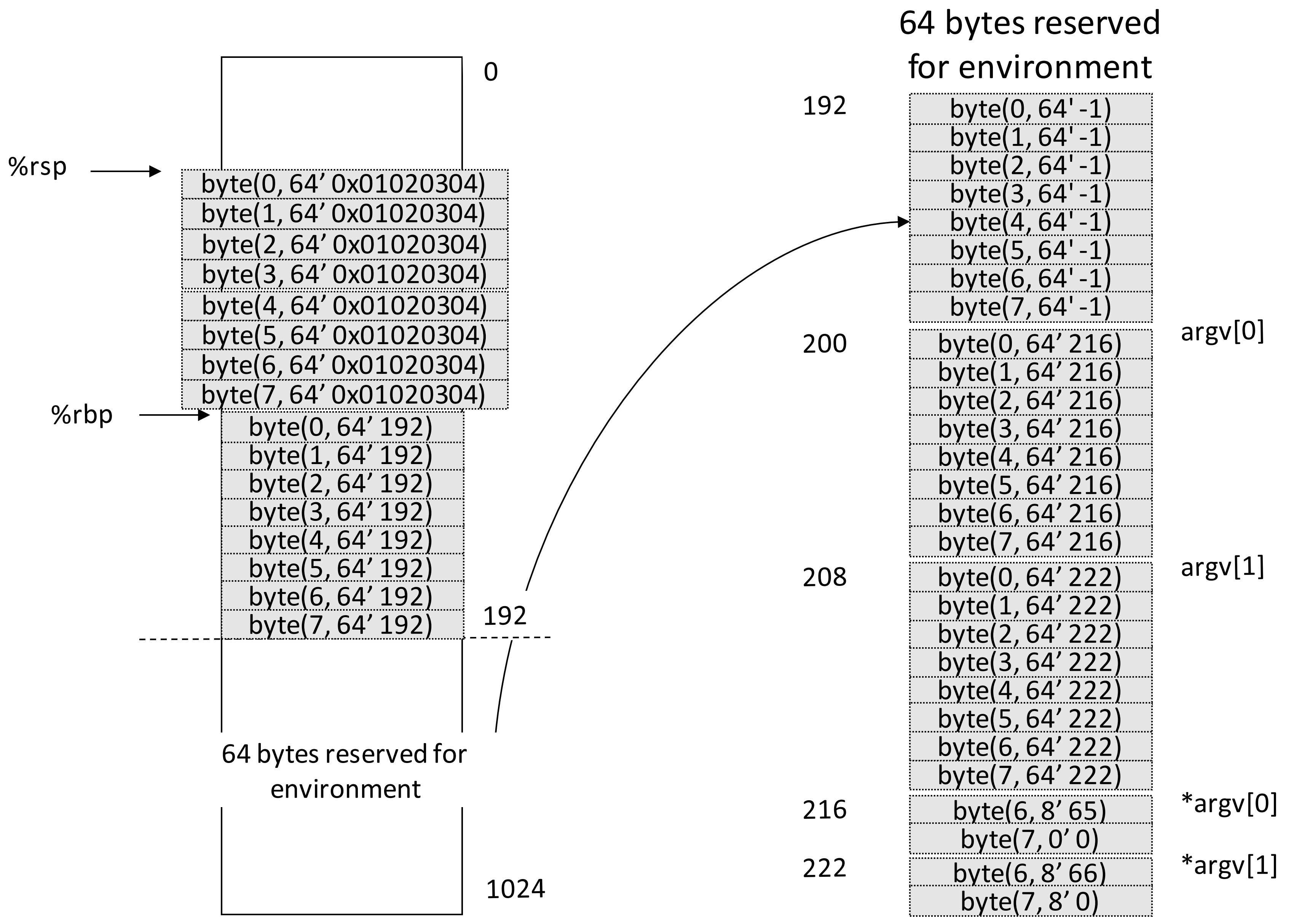
%rdi: 2
%rsi: 200



```
$ krun prog.s “A" “B"
pushq %rbp
movq %rsp, %rbp
movq 0x01020304, -8(%rbp)
movb  (%rsi), %rax
movb  (%rax), %bl
addw -6(%rbp), %bx
movq %rbp, %rsp
popq %rbp
```

Memory Semantics

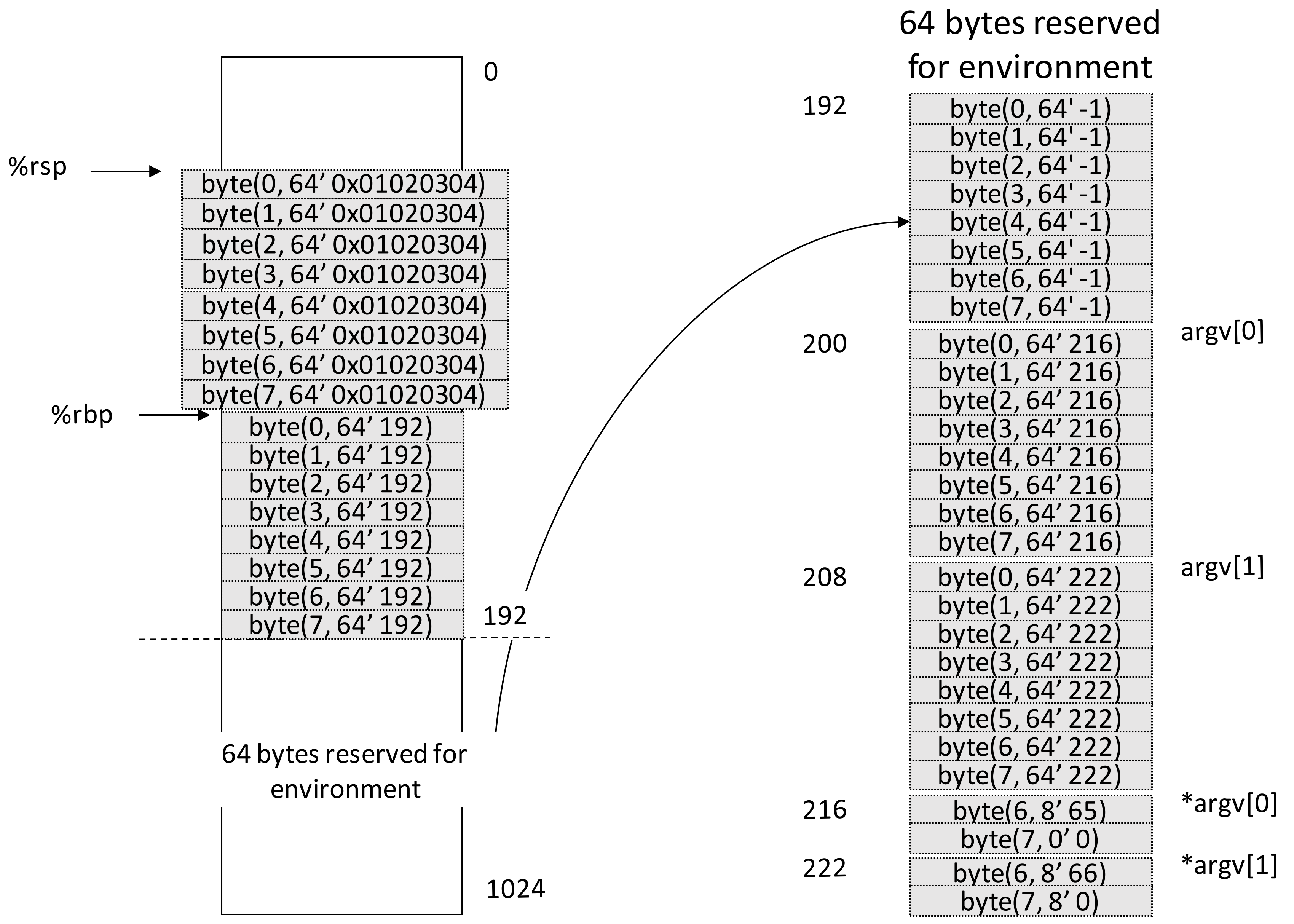
%rdi: 2
%rsi: 200
%rax: 216



```
$ krun prog.s “A" “B"
pushq %rbp
movq %rsp, %rbp
movq 0x01020304, -8(%rbp)
movb  (%rsi), %rax
movb  (%rax), %bl
addw -6(%rbp), %bx
movq %rbp, %rsp
popq %rbp
```

Memory Semantics

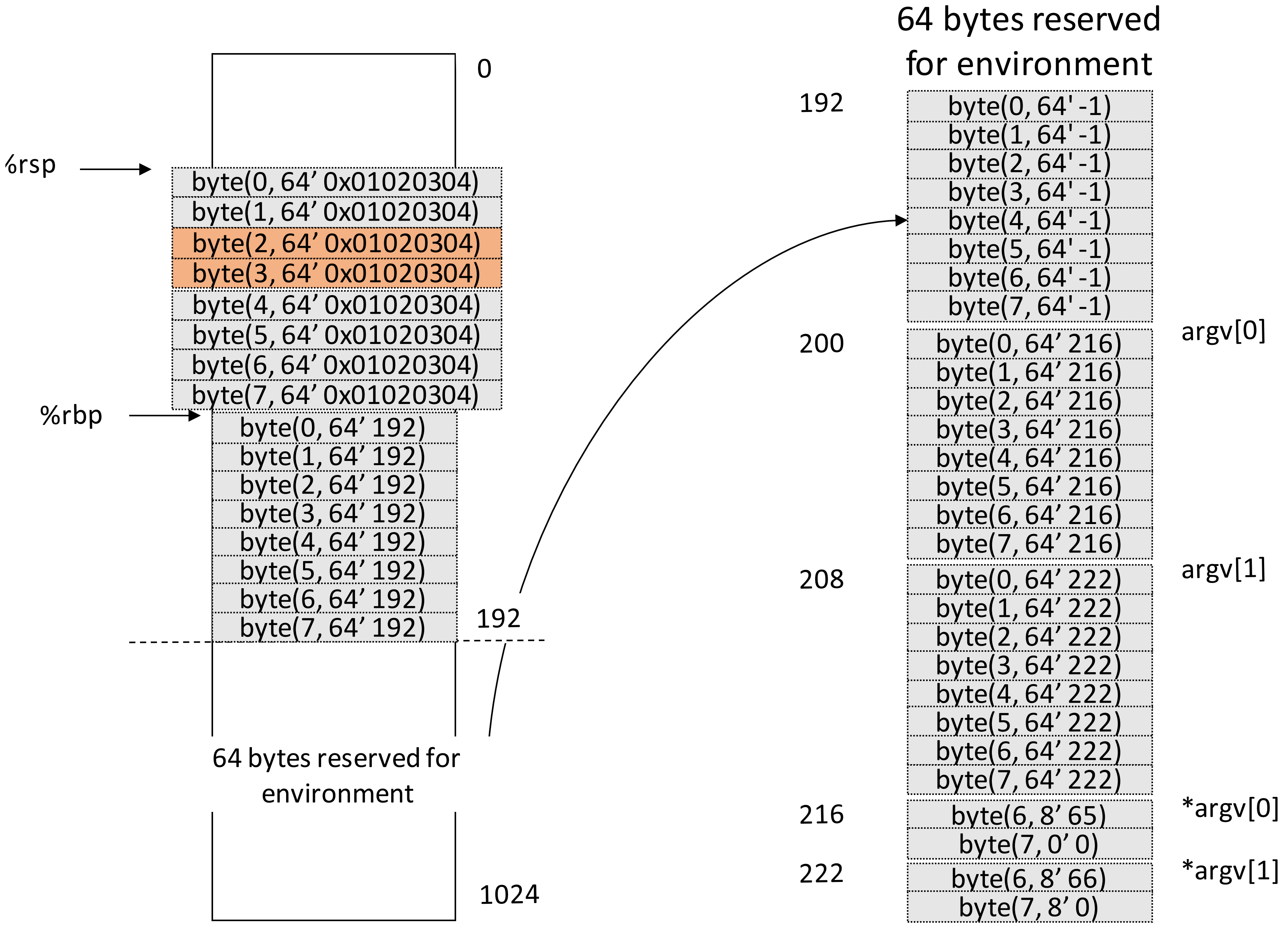
%rdi: 2
%rsi: 200
%rax: 216
%rbx: 65



```
$ krun prog.s “A" “B"
pushq %rbp
movq %rsp, %rbp
movq 0x01020304, -8(%rbp)
movb  (%rsi), %rax
movb  (%rax), %bl
addw -6(%rbp), %bx
movq %rbp, %rsp
popq %rbp
```

Memory Semantics

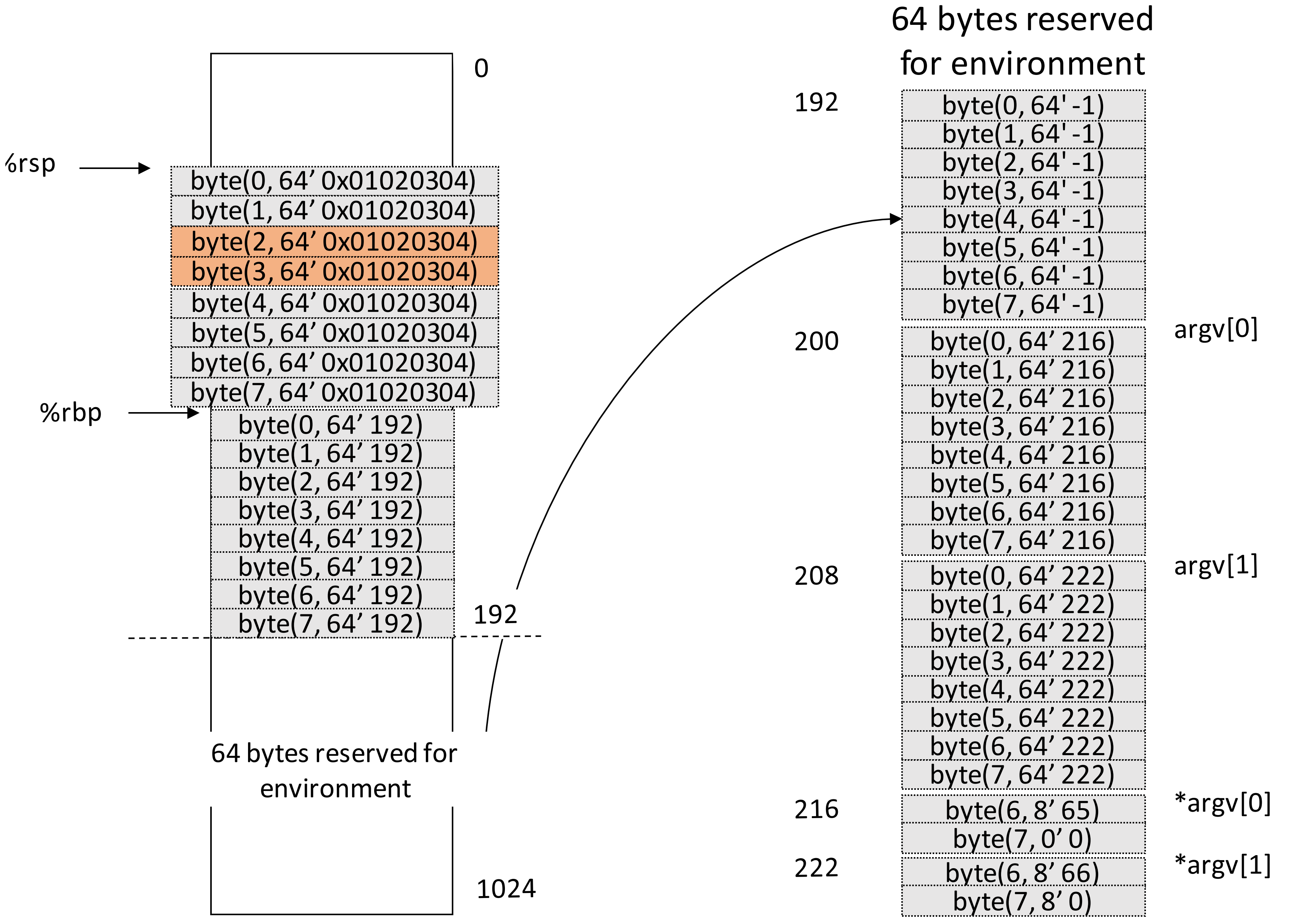
%rdi: 2
%rsi: 200
%rax: 200
%rbx: 216
%rbx: 65 +
  0x01020304[31:24] ∘
            0x01020304[23:16]



```
$ krun prog.s “A" “B"
pushq %rbp
movq %rsp, %rbp
movq 0x01020304, -8(%rbp)
movb  (%rsi), %rax
movb  (%rax), %bl
addw -6(%rbp), %bx
movq %rbp, %rsp
popq %rbp
```

Memory Semantics

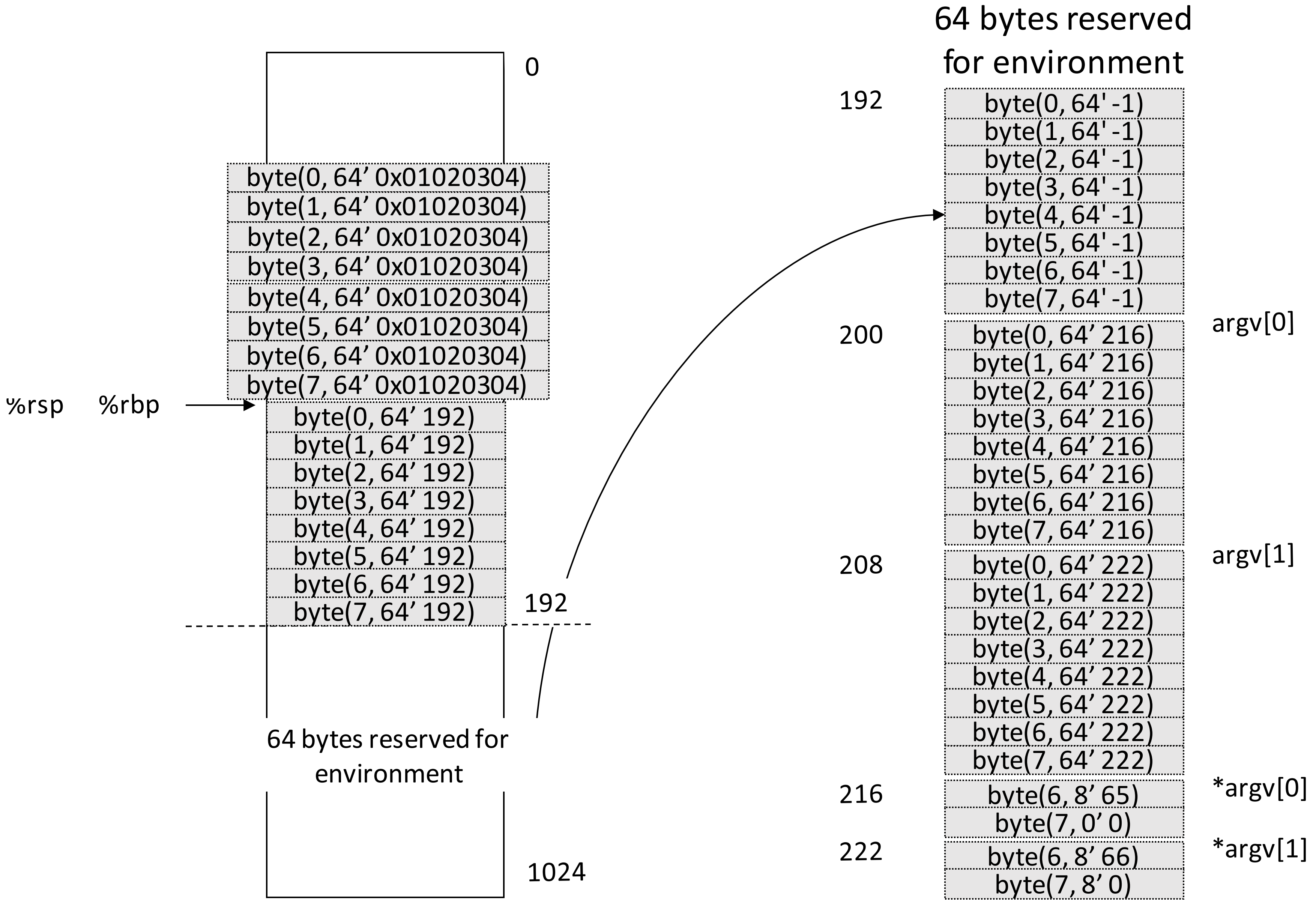
%rdi: 2
%rsi: 200
%rax: 200
%rbx: 216
%rbx: 65 + 0x0102



```
$ krun prog.s “A" “B"
pushq %rbp
movq %rsp, %rbp
movq 0x01020304, -8(%rbp)
movb  (%rsi), %rax
movb  (%rax), %bl
addw -6(%rbp), %bx
movq %rbp, %rsp
popq %rbp
```

Memory Semantics

%rdi: 2
%rsi: 200
%rax: 200
%rbx: 216
%rbx: 65 + 0x0102

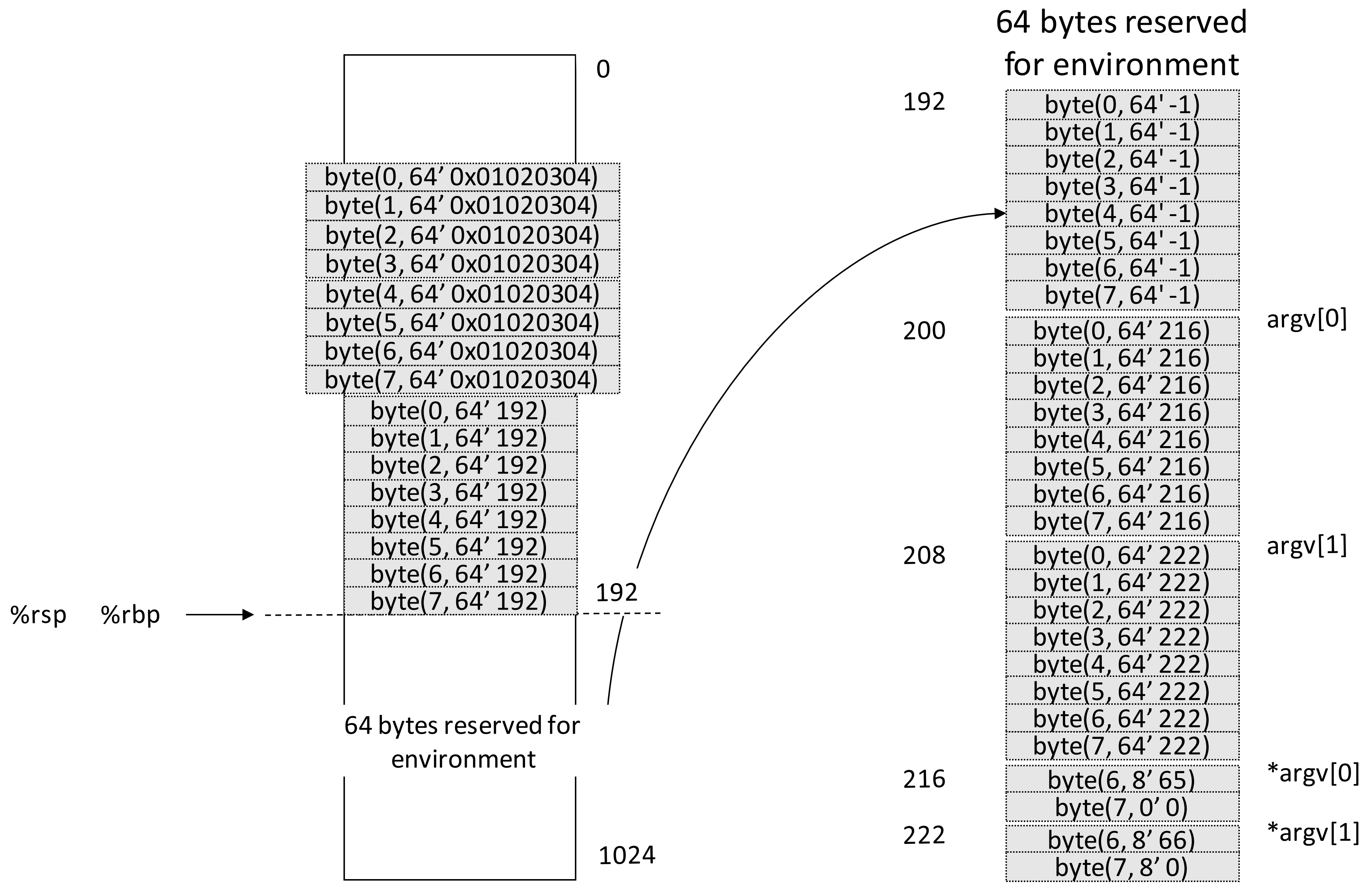


```
$ krun prog.s “A" “B"
pushq %rbp
movq %rsp, %rbp
movq 0x01020304, -8(%rbp)
movb  (%rsi), %rax
movb  (%rax), %bl
addw -6(%rbp), %bx
movq %rbp, %rsp
popq %rbp
```

Memory Semantics

%rdi: 2
%rsi: 200
%rax: 200
%rbx: 216
%rbx: 65 + 0x0102

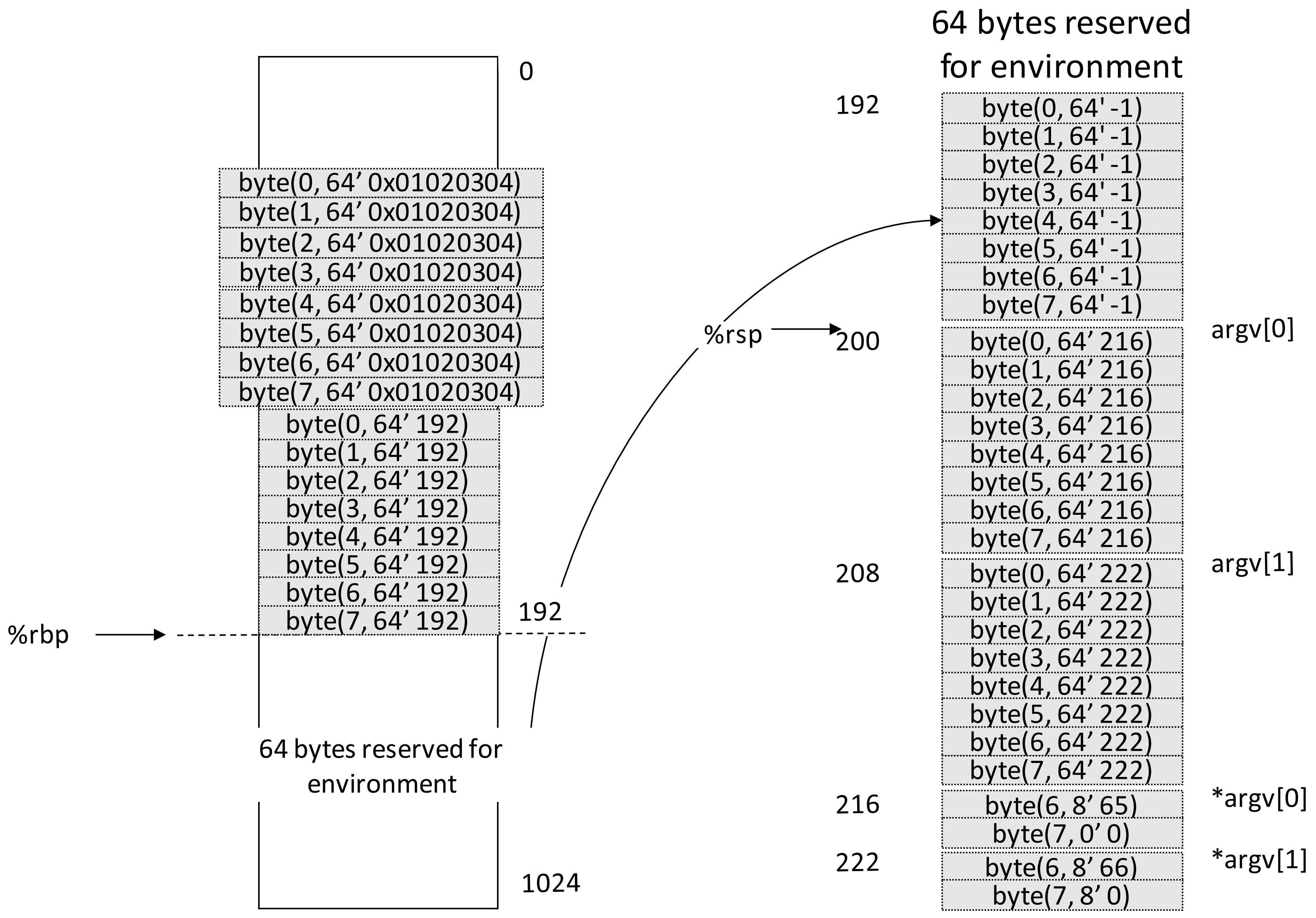
```
$ krun prog.s “A" “B"
pushq %rbp
movq %rsp, %rbp
movq 0x01020304, -8(%rbp)
movb  (%rsi), %rax
movb  (%rax), %bl
addw -6(%rbp), %bx
movq %rbp, %rsp
popq %rbp
ret
```



Memory Semantics

%rdi: 2
%rsi: 200
%rax: 200
%rbx: 216
%rbx: 65 + 0x0102

```
$ krun prog.s “A" “B"
pushq %rbp
movq %rsp, %rbp
movq 0x01020304, -8(%rbp)
movb  (%rsi), %rax
movb  (%rax), %bl
addw -6(%rbp), %bx
movq %rbp, %rsp
popq %rbp
ret
```



Memory & Control flow Semantics: Demo

Going forward

- Testing
  - Testing on Practical programs Vs Coverage
    - Testing individual instructions: More coverage
    - Testing a test suite: More practical
- Applications
  - Translation validation of instruction semantics used by McSema.