Verified Peephole Optimization for CompCert

Eric Mullen, Daryl Zuniga, Zach Tatlock, Dan Grossman
University of Washington

Verified Compilers

X Leroy, Formal certification
of a compiler back-end.
POPL 2006

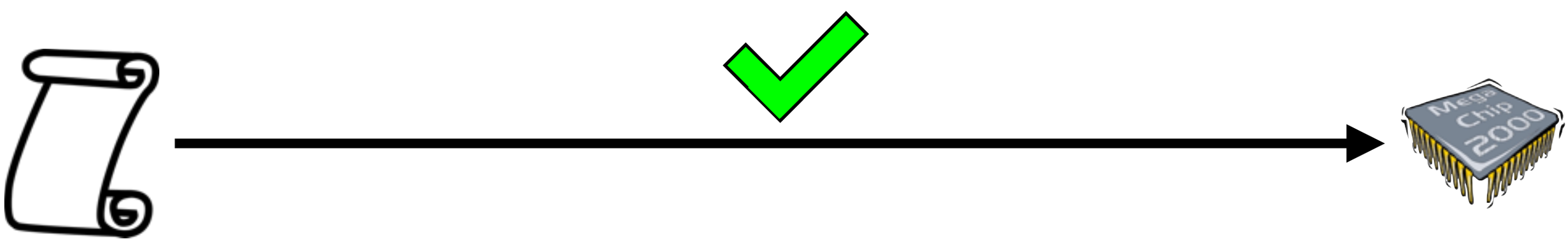
Verified Compilers

CompCert

X Leroy, Formal certification
of a compiler back-end.
POPL 2006

Verified Compilers

CompCert



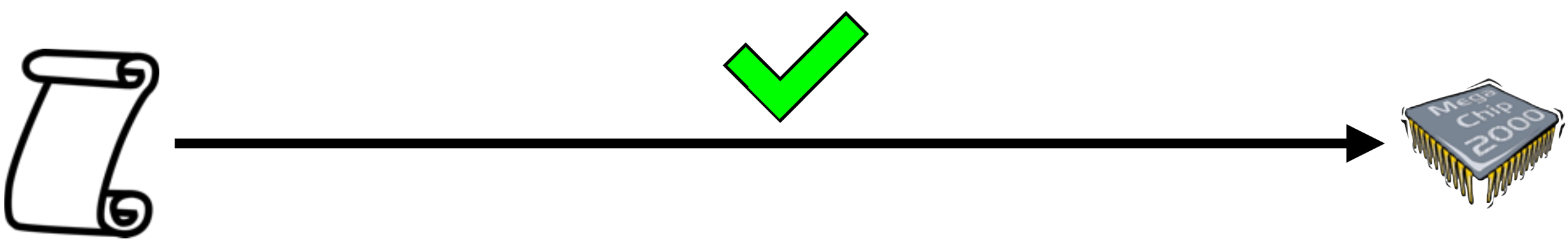
X Yang, Y Chen, E Eide, J Regehr. Finding and Understanding Bugs in C Compilers. PLDI 2011

X Leroy, Formal certification of a compiler back-end. POPL 2006

V Le, M Afshari, Z Su. Compiler Validation via Equivalence Modulo Inputs. PLDI 2014

Verified Compilers

CompCert



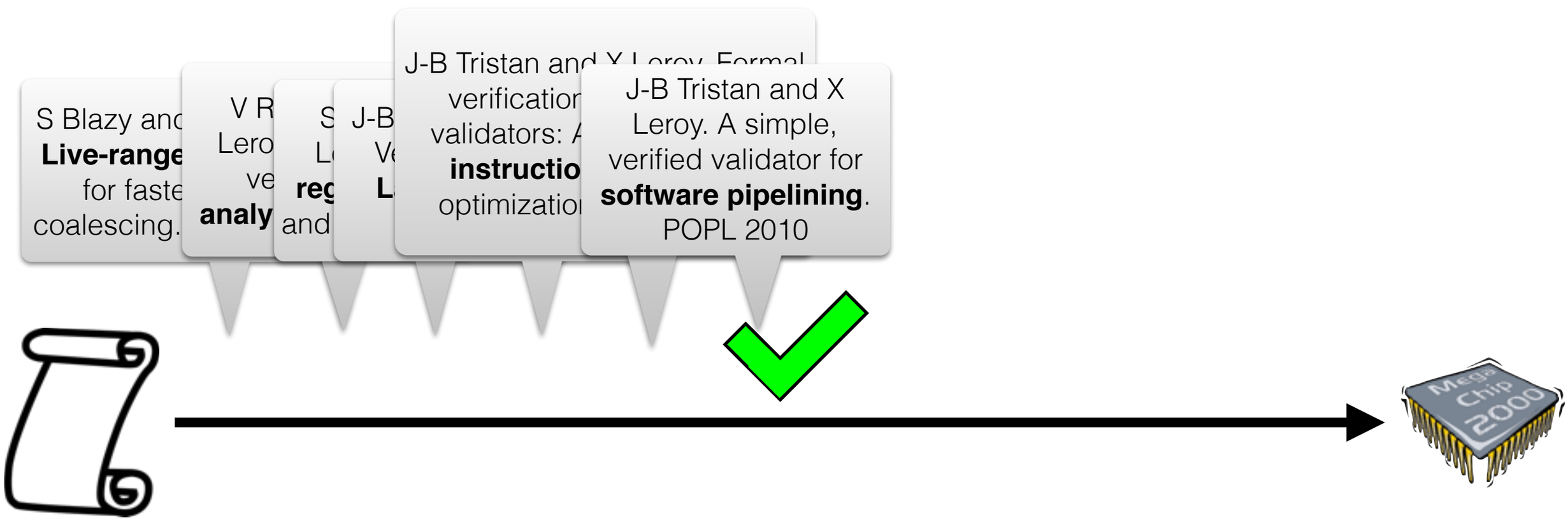
X Yang, Y Chen, E Eide, J Regehr. Finding and Understanding Bugs in C Compilers. PLDI 2011

X Leroy, Formal certification of a compiler back-end. POPL 2006

V Le, M Afshari, Z Su. Compiler Validation via Equivalence Modulo Inputs. PLDI 2014

Verified Compilers

CompCert



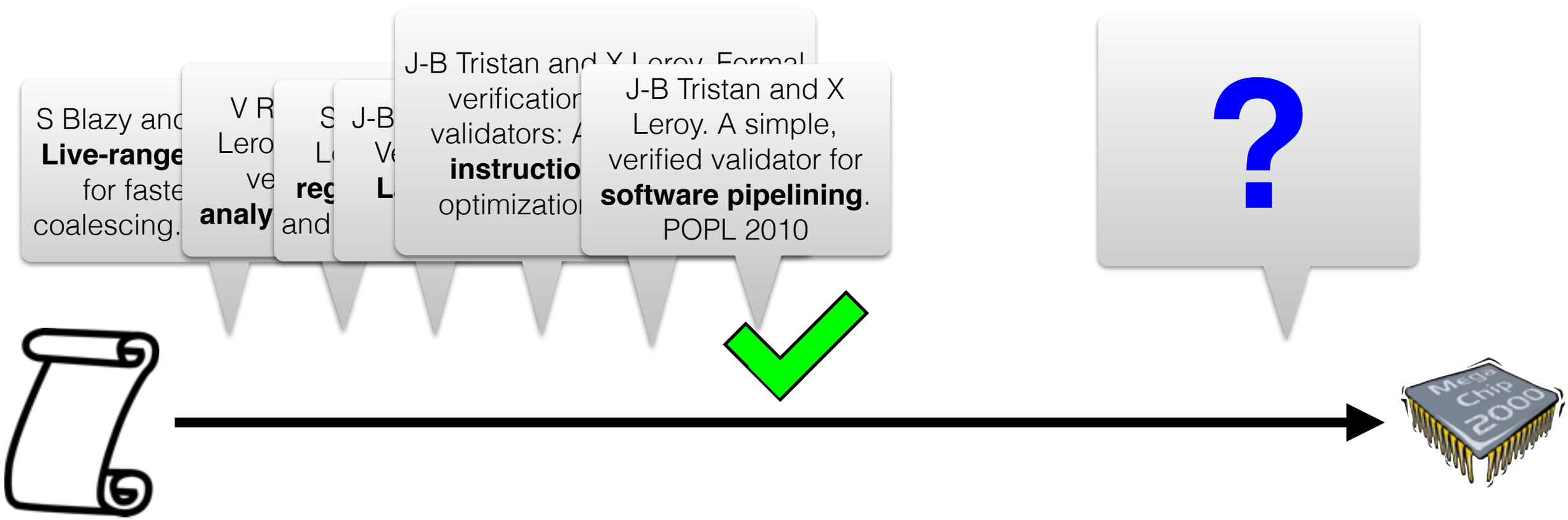
X Yang, Y Chen, E Eide, J Regehr. Finding and Understanding Bugs in C Compilers. PLDI 2011

X Leroy, Formal certification of a compiler back-end. POPL 2006

V Le, M Afshari, Z Su. Compiler Validation via Equivalence Modulo Inputs. PLDI 2014

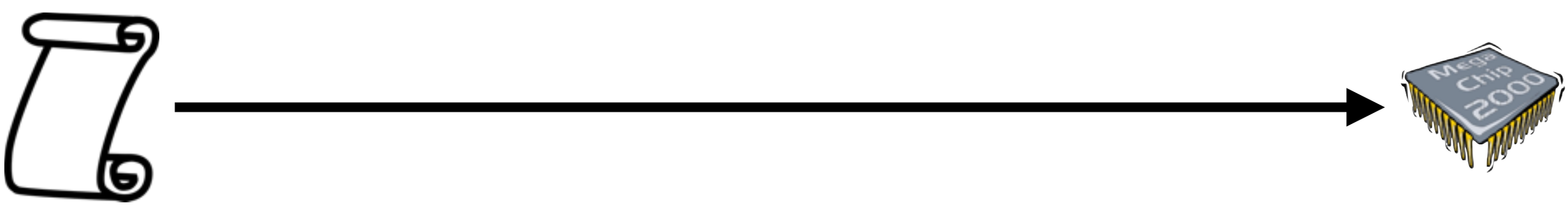
Verified Compilers

CompCert

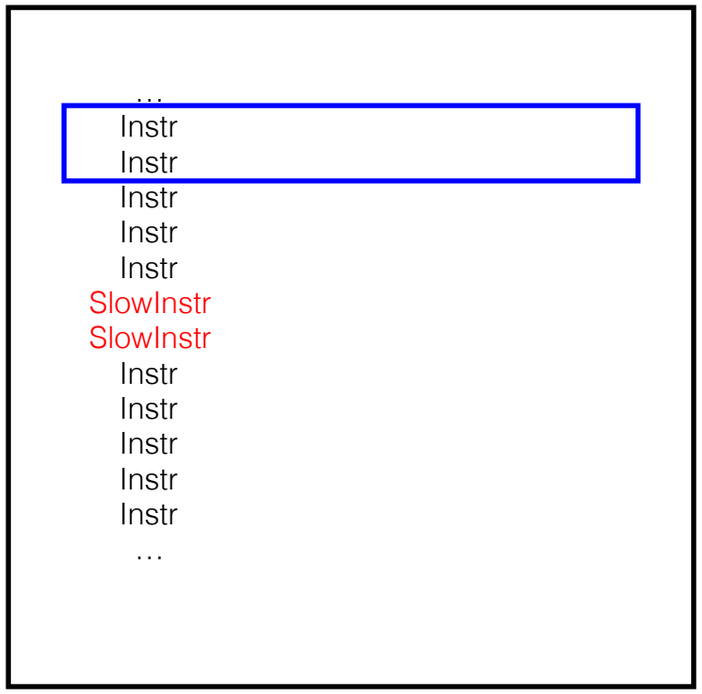


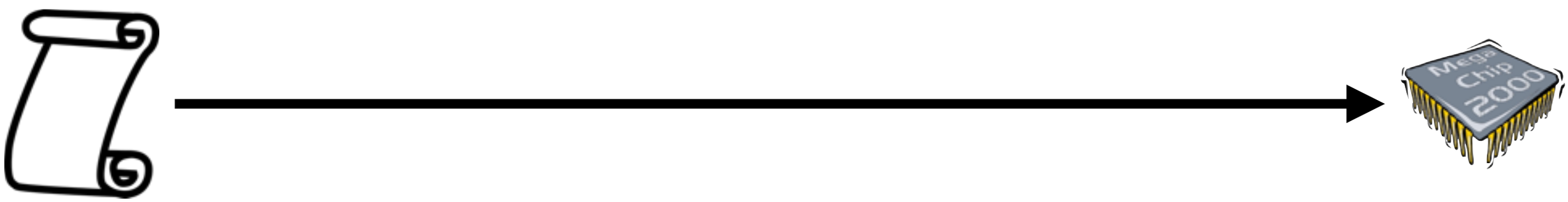
Assembly Transformations

```
...
Instr
Instr
Instr
Instr
Instr
SlowInstr
SlowInstr
Instr
Instr
Instr
Instr
Instr
Instr
...
```

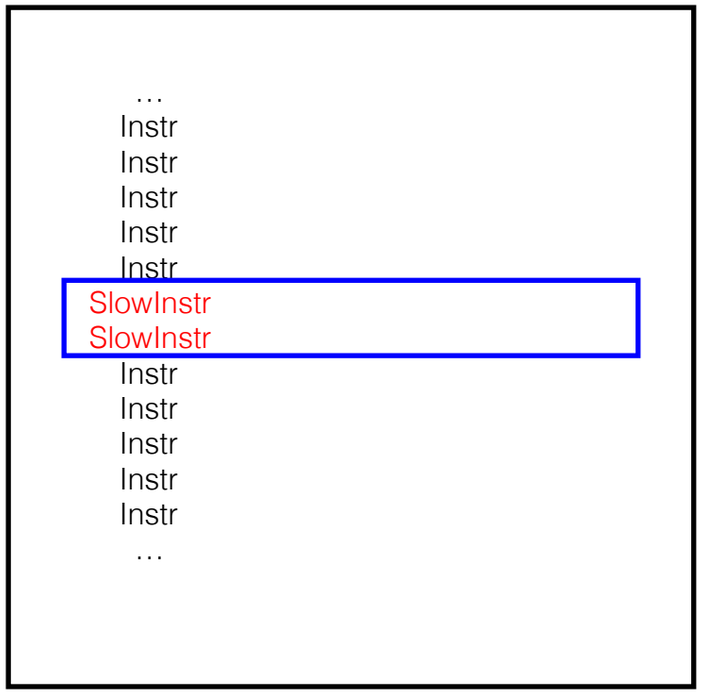


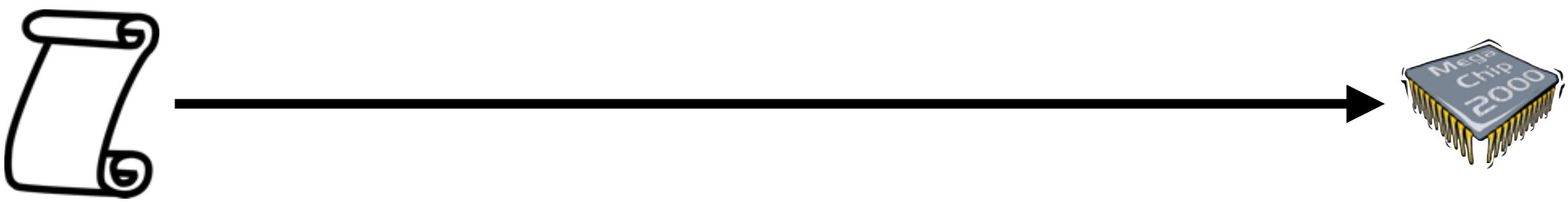
Assembly Transformations



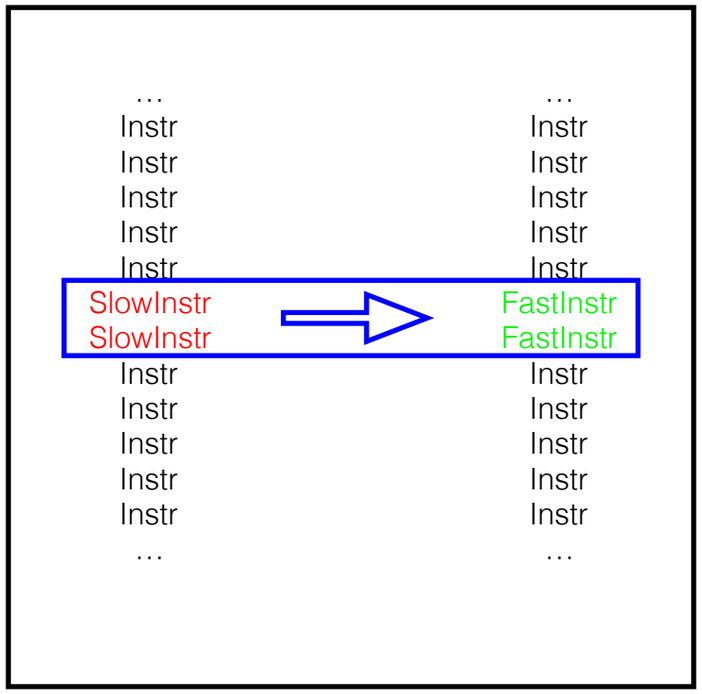


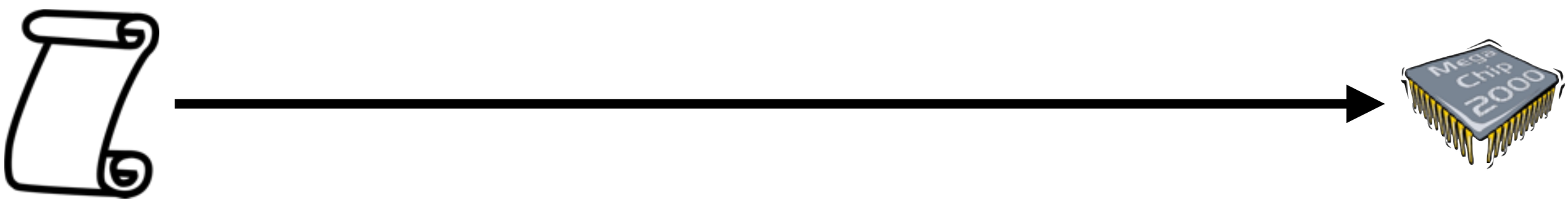
Assembly Transformations



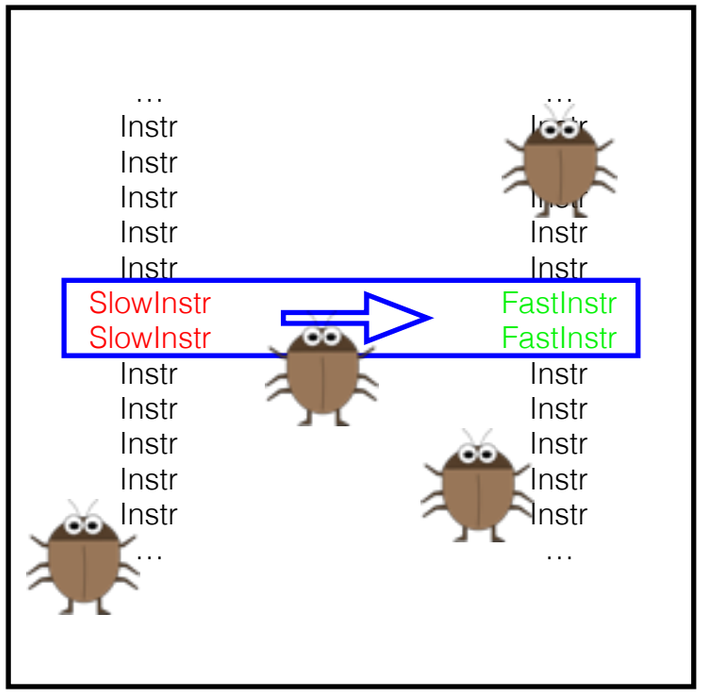


Assembly Transformations

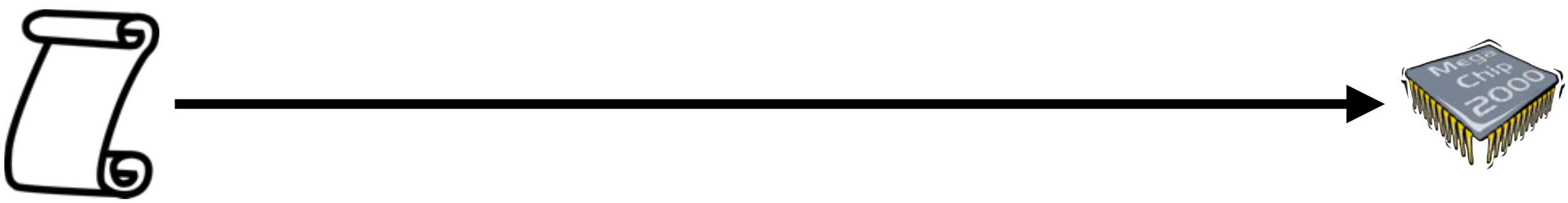




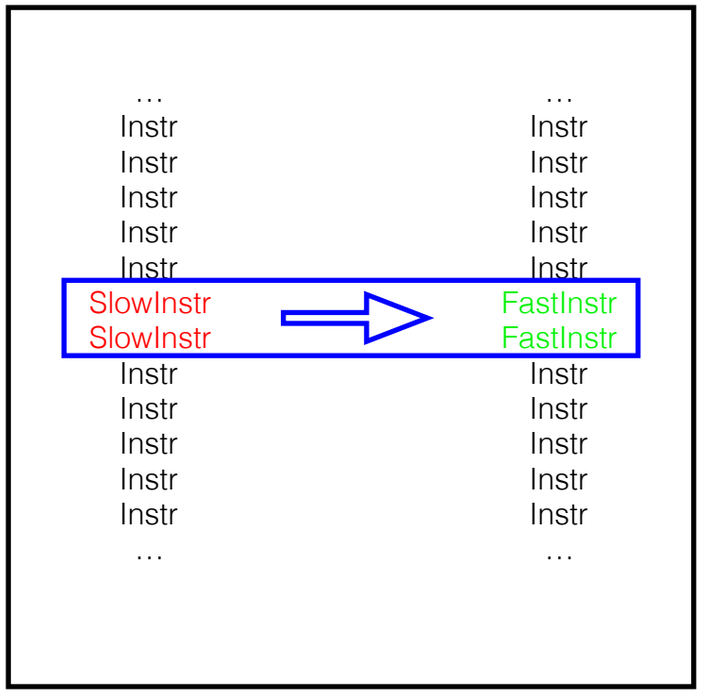
Assembly Transformations

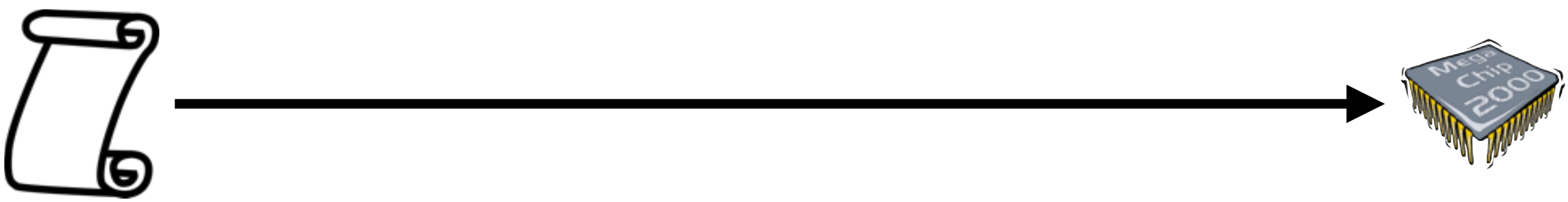


Alive
Lopes et al
PLDI 15

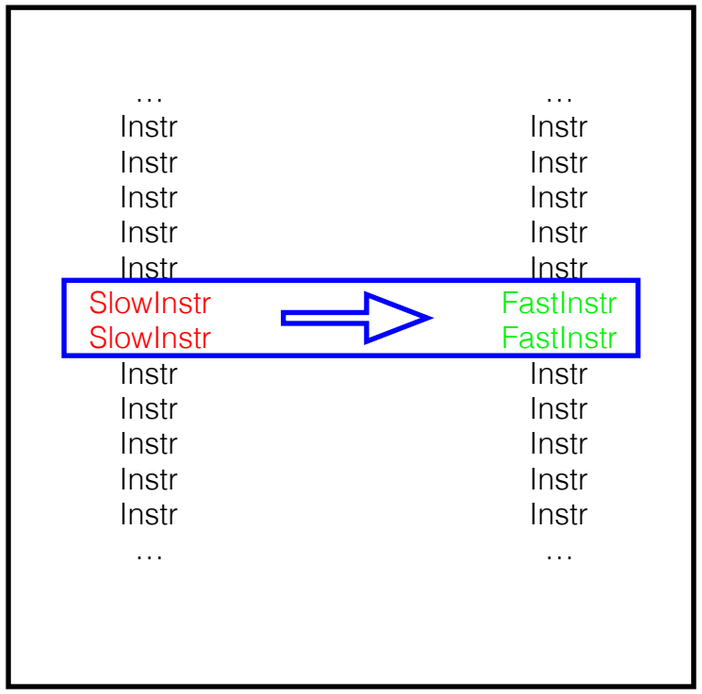


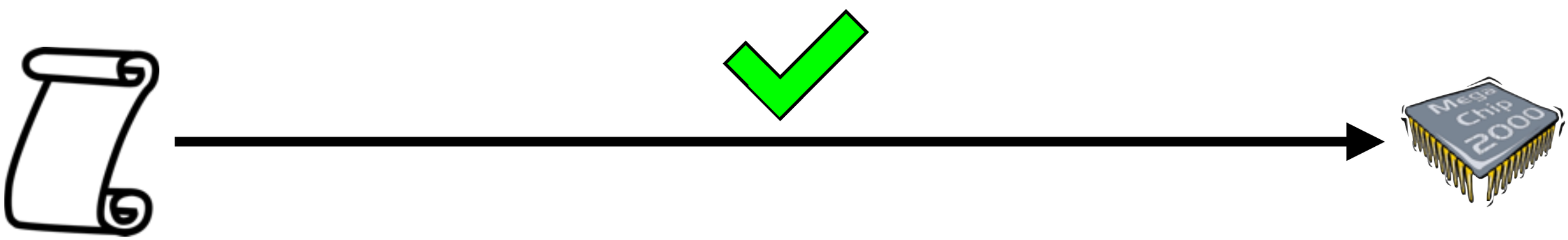
Assembly Transformations



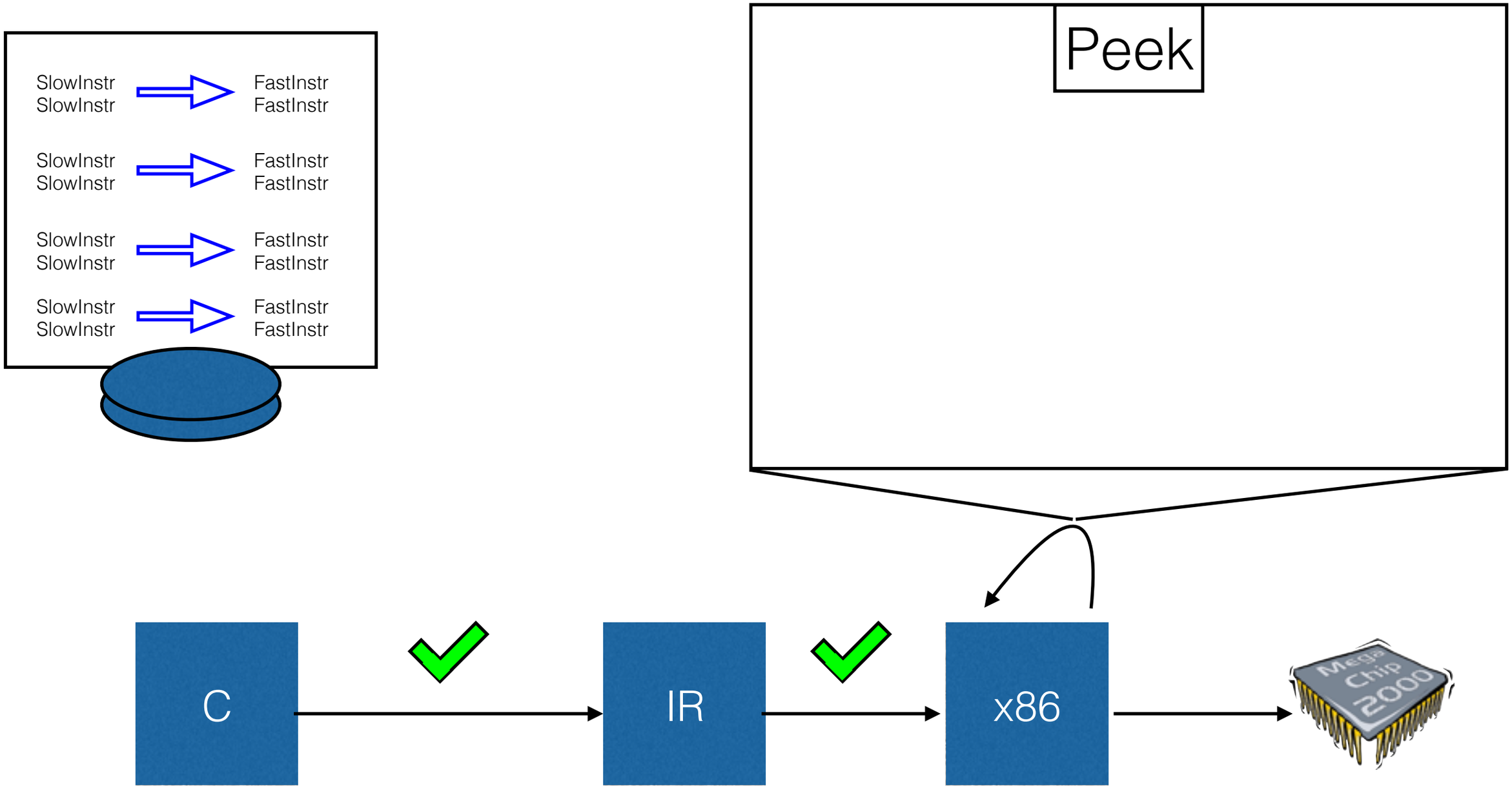


Assembly Transformations

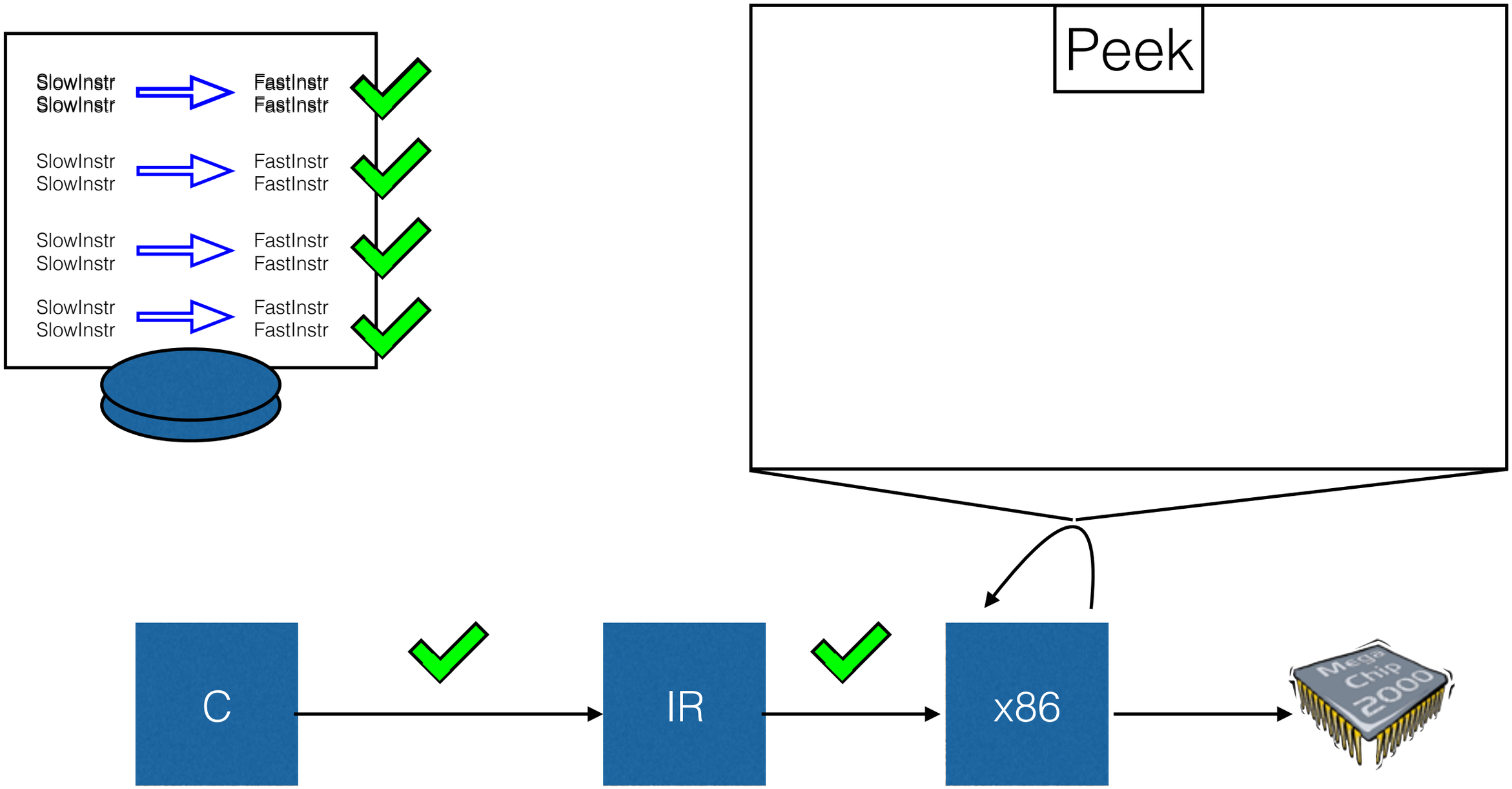




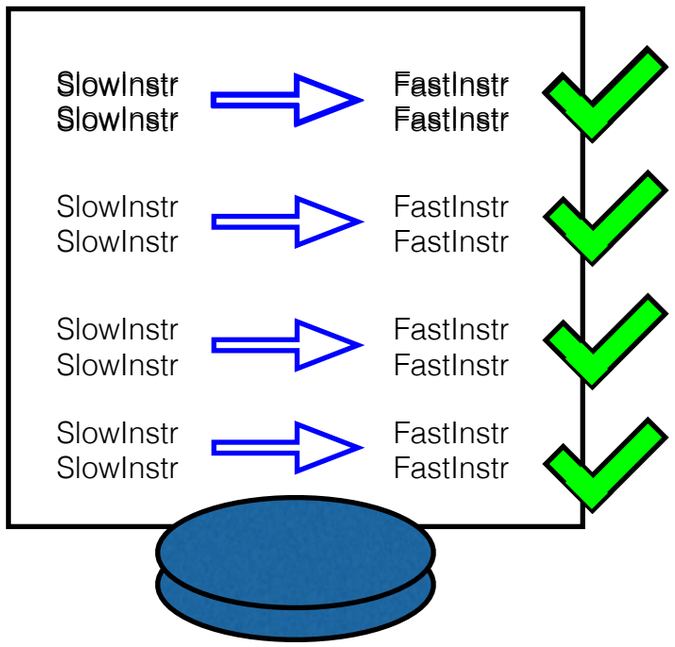
CompCert + Peek

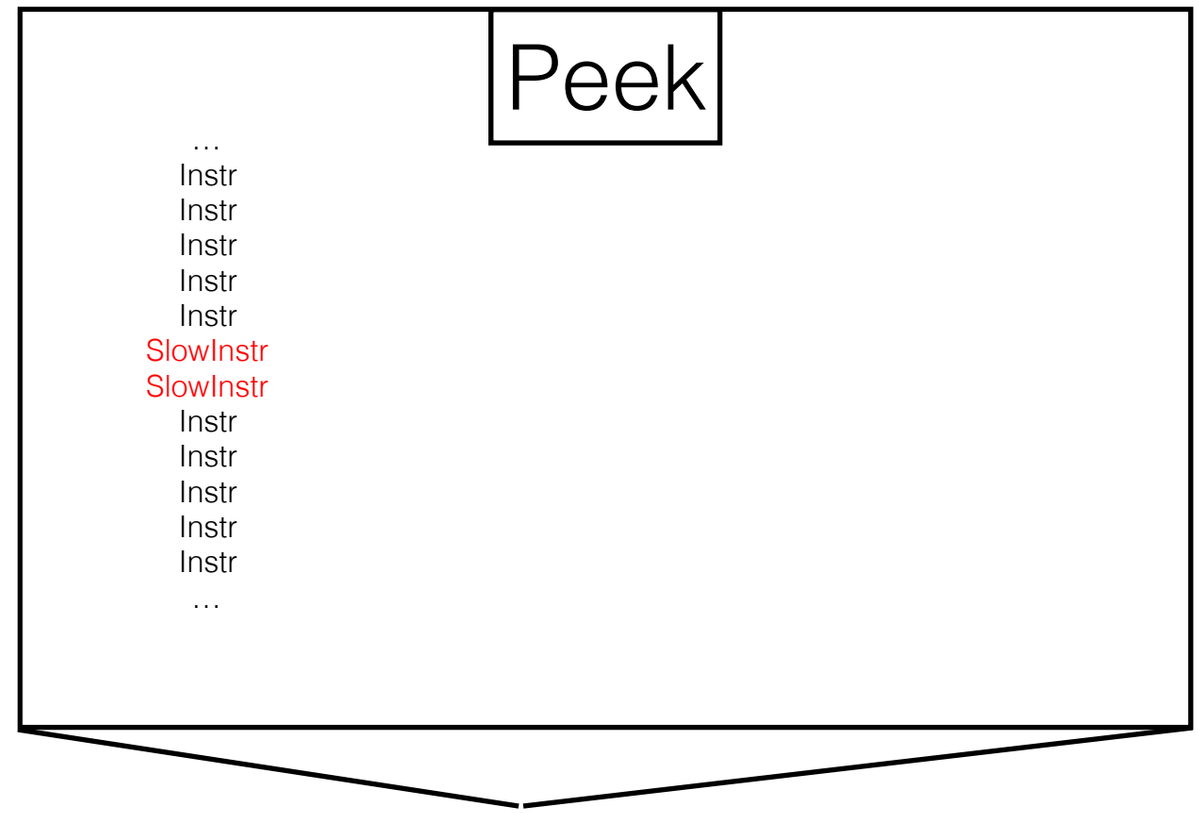


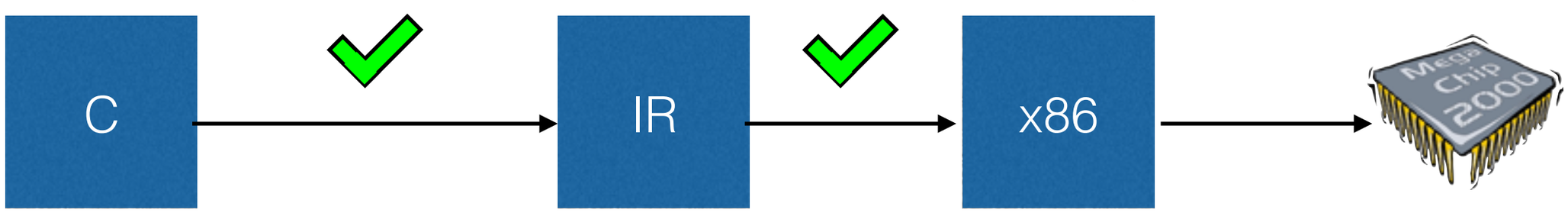
CompCert + Peek



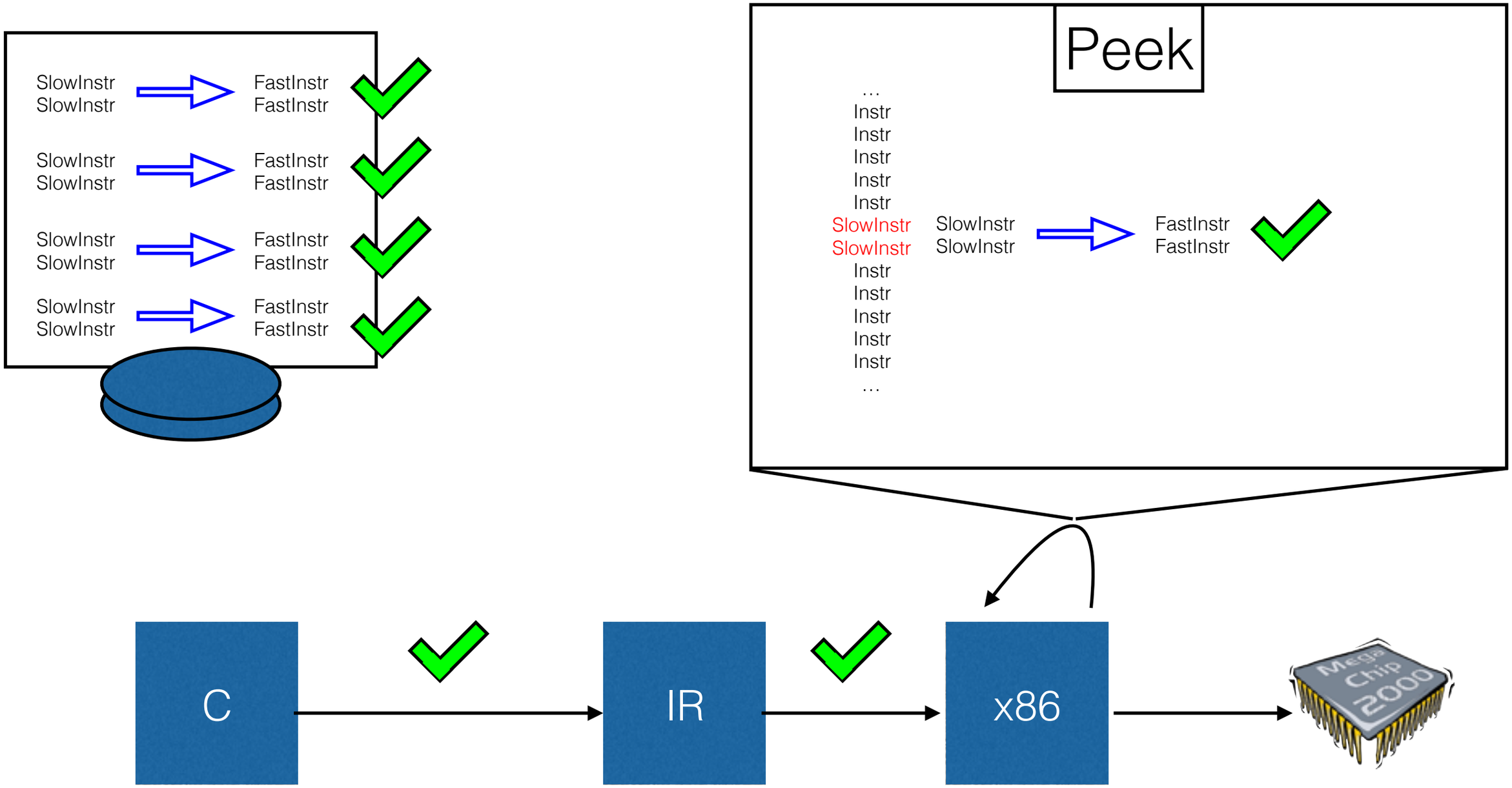
CompCert + Peek



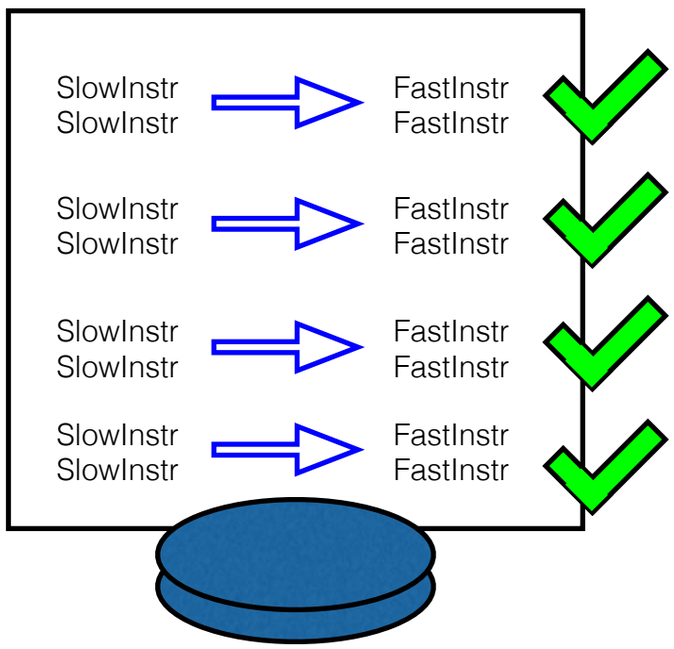


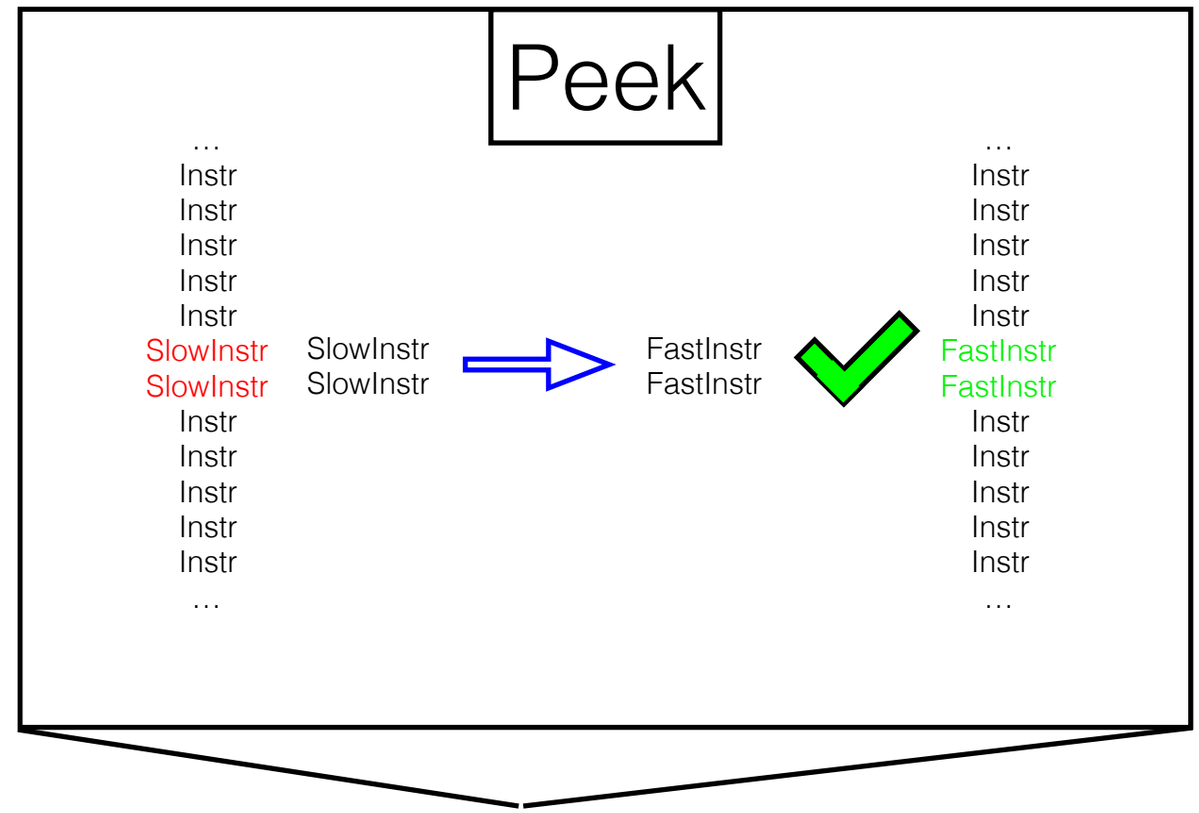


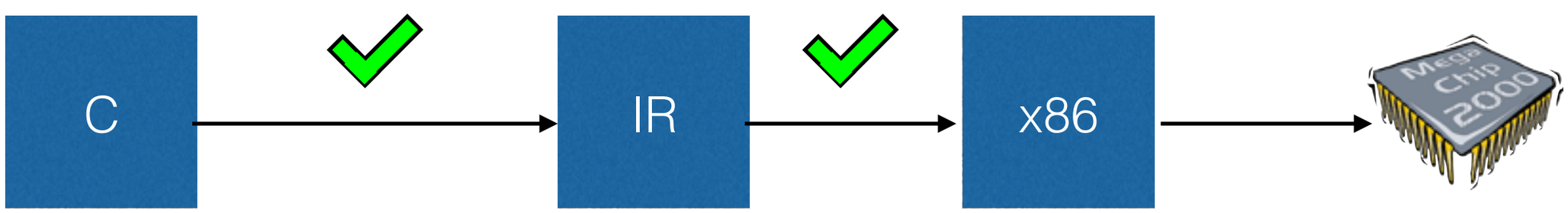
CompCert + Peek



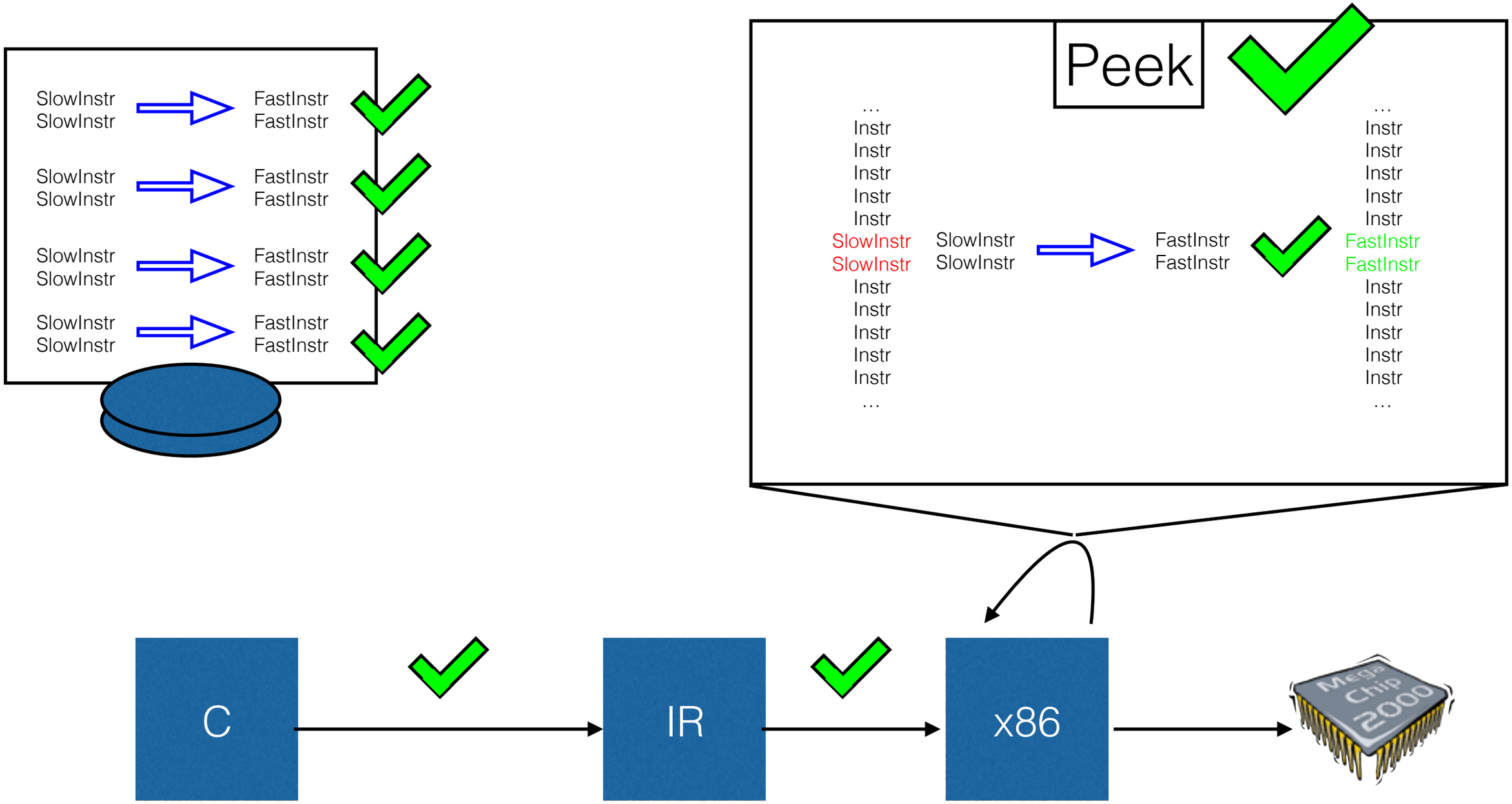
CompCert + Peek



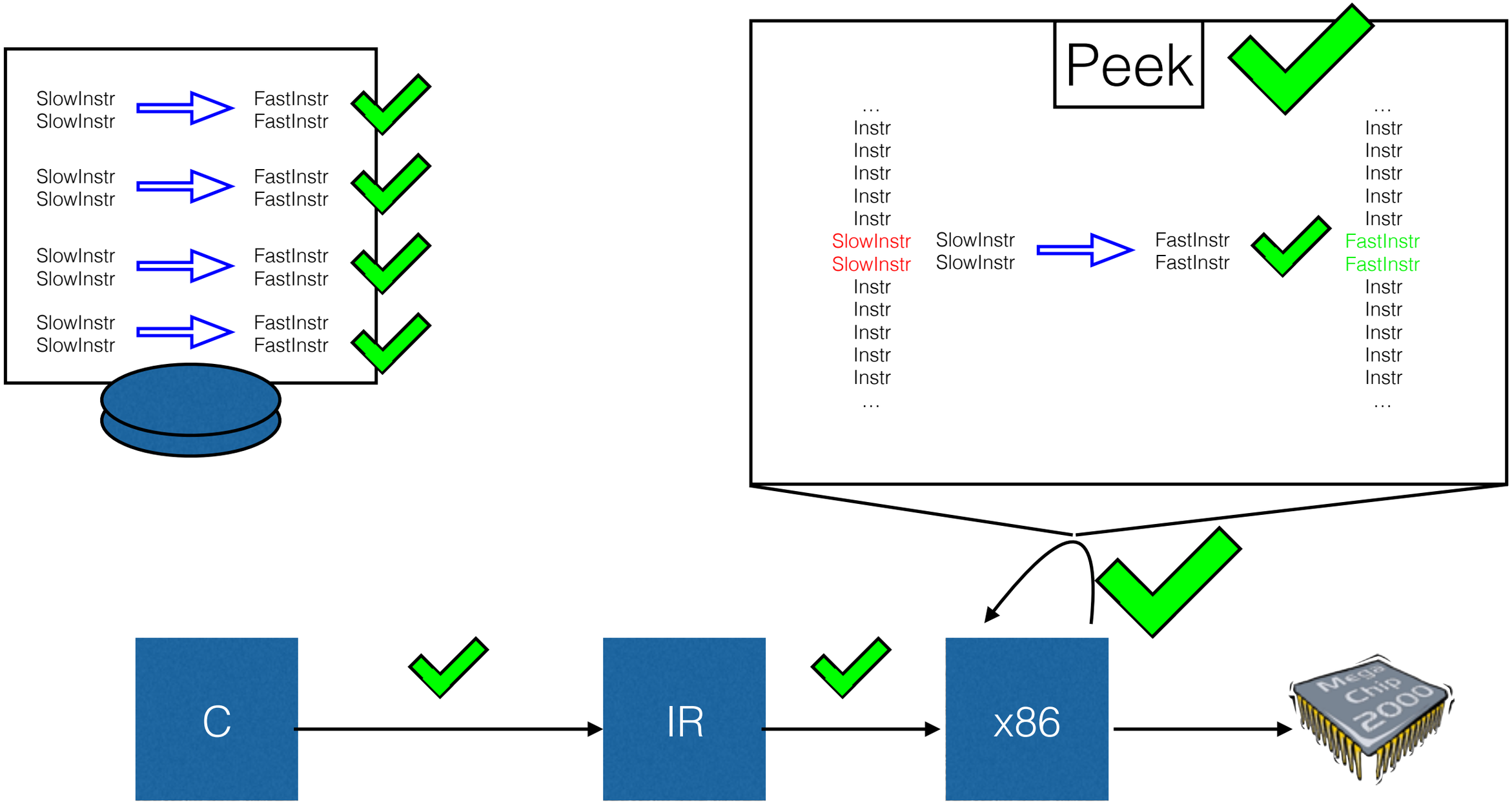




CompCert + Peek

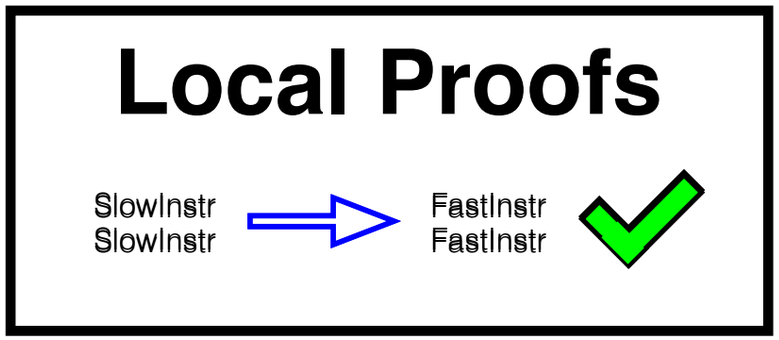


CompCert + Peek



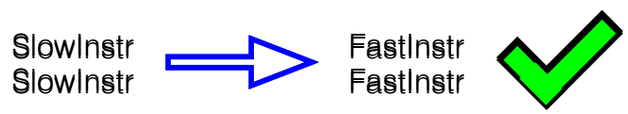
Outline

Outline

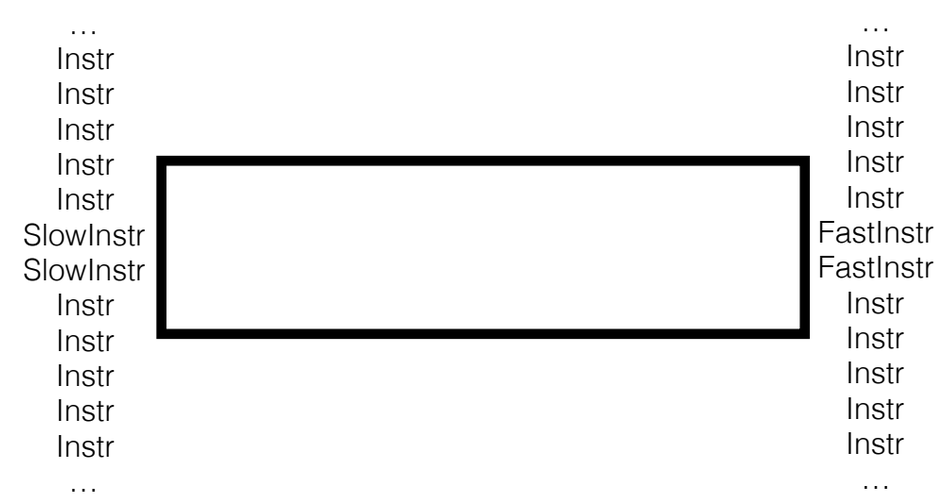


Outline

Local Proofs

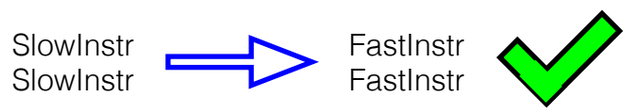


Global Correctness

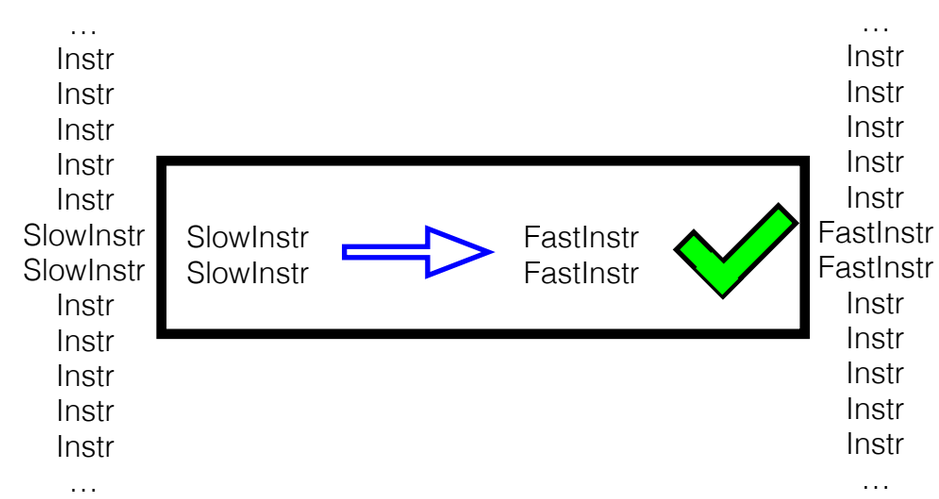


Outline

Local Proofs



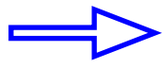
Global Correctness



Outline

Local Proofs

SlowInstr
SlowInstr

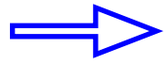


FastInstr
FastInstr

Global Correctness

…
Instr
Instr
Instr
Instr
Instr
SlowInstr
SlowInstr
Instr
Instr
Instr
Instr
Instr
Instr
…

SlowInstr
SlowInstr



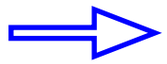
FastInstr
FastInstr

…
Instr
Instr
Instr
Instr
Instr
FastInstr
FastInstr
Instr
Instr
Instr
Instr
Instr
Instr
…

Outline

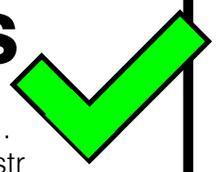
Local Proofs

SlowInstr
SlowInstr



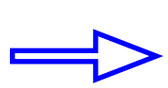
FastInstr
FastInstr

Global Correctness



…
Instr
Instr
Instr
Instr
Instr
SlowInstr
SlowInstr
Instr
Instr
Instr
Instr
Instr
Instr
…

SlowInstr
SlowInstr



FastInstr
FastInstr

…
Instr
Instr
Instr
Instr
Instr
FastInstr
FastInstr
Instr
Instr
Instr
Instr
Instr
Instr
…

? Liveness

Outline

Local Proofs

SlowInstr
SlowInstr
⇒
FastInstr
FastInstr
✓

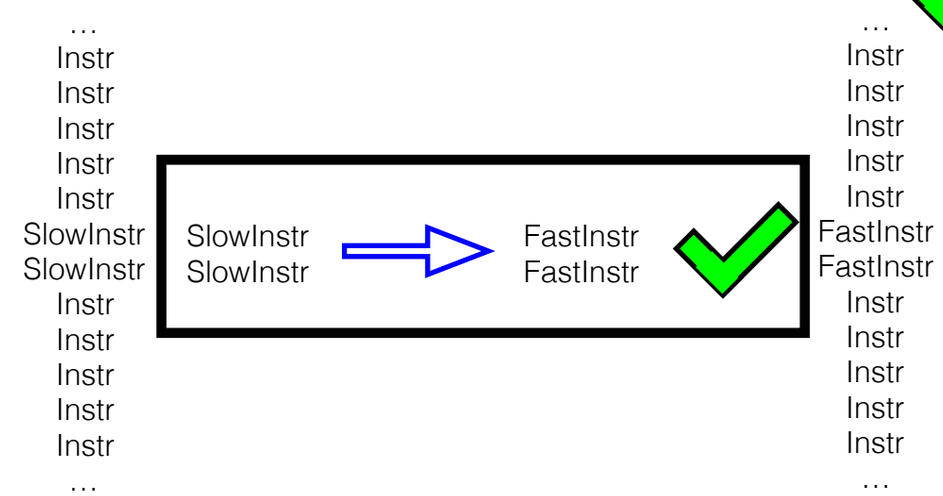
New Semantics

Pointers

⇩

Bit Vectors

Global Correctness



? Liveness

Outline

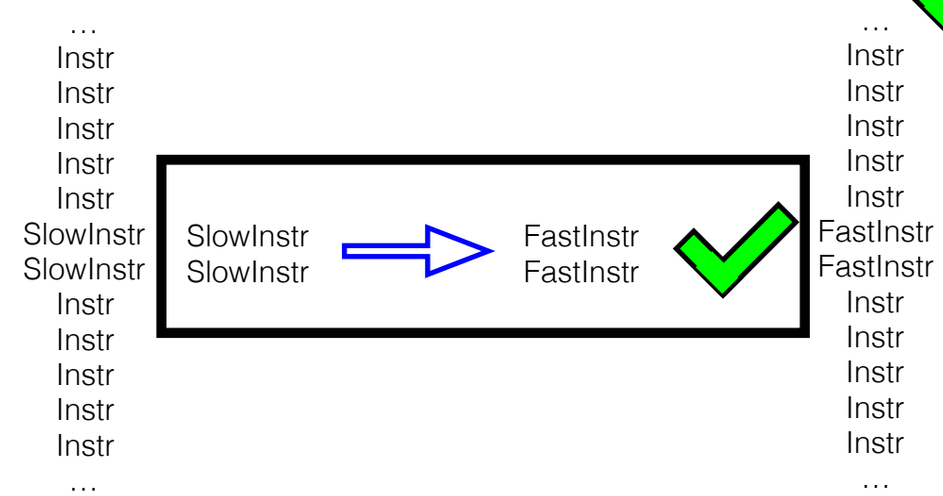
Local Proofs

SlowInstr
SlowInstr
⇒
FastInstr
FastInstr
✓

New Semantics

Pointers
⇩
Bit Vectors

Global Correctness



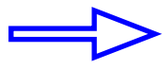
? Liveness

Evaluation

Outline

Local Proofs

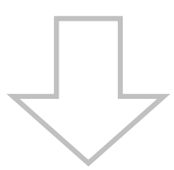
SlowInstr
SlowInstr



FastInstr
FastInstr

New Semantics

Pointers

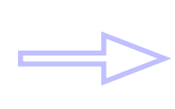


Bit Vectors

Global Correctness

…
Instr
Instr
Instr
Instr
Instr
SlowInstr
SlowInstr
Instr
Instr
Instr
Instr
Instr
Instr
…

SlowInstr
SlowInstr



FastInstr
FastInstr

…
Instr
Instr
Instr
Instr
Instr
FastInstr
FastInstr
Instr
Instr
Instr
Instr
Instr
Instr
…

? Liveness

Evaluation

S. Bansal and A. Aiken, Automatic Generation of Peephole Superoptimizers, ASPLOS 2006

Example Peephole

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```

PC

%**eax**: A_0

%**ecx**: C_0

S. Bansal and A. Aiken, Automatic Generation of Peephole Superoptimizers, ASPLOS 2006

Example Peephole

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```

PC

%**eax**: A_0

%**ecx**: C_0

Example Peephole

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```

%**eax**: A_0	%**eax**: A_0
%**ecx**: C_0	%**ecx**: $C_0 - A_0$

Example Peephole

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```

%**eax**: A_0	%**eax**: A_0
%**ecx**: C_0	%**ecx**: $C_0 - A_0$

Example Peephole

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```

%**eax**: A_0	%**eax**: A_0	%**eax**: $C_0 - A_0$
%**ecx**: C_0	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$

Example Peephole

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```

%**eax**: A_0	%**eax**: A_0	%**eax**: $C_0 - A_0$
%**ecx**: C_0	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$

Example Peephole

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```

%**eax**: A_0	%**eax**: A_0	%**eax**: $C_0 - A_0$	%**eax**: $C_0 - A_0 - 1$
%**ecx**: C_0	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$

Example Peephole

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```

%**eax**: A_0	%**eax**: A_0	%**eax**: $C_0 - A_0$	%**eax**: $C_0 - A_0 - 1$
%**ecx**: C_0	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$

Example Peephole

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```

Recall:
x - y - 1 = x + ~y
for two's complement

%**eax**: A_0	%**eax**: A_0	%**eax**: $C_0 - A_0$	%**eax**: $C_0 - A_0 - 1$
%**ecx**: C_0	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$

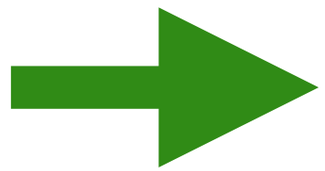
Example Peephole

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```

%**eax**: A_0	%**eax**: A_0	%**eax**: $C_0 - A_0$	%**eax**: $C_0 - A_0 - 1$
%**ecx**: C_0	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$

Example Peephole

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```



```
notl    %eax
addl    %ecx, %eax
```

%**eax**: A_0	%**eax**: A_0	%**eax**: $C_0 - A_0$	%**eax**: $C_0 - A_0 - 1$
%**ecx**: C_0	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$

Example Peephole

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```

PC

```
notl    %eax
addl    %ecx, %eax
```

%**eax**: A_0	%**eax**: A_0	%**eax**: $C_0 - A_0$	%**eax**: $C_0 - A_0 - 1$	%**eax**: A_0
%**ecx**: C_0	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: C_0

Example Peephole

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```

PC

```
notl    %eax
addl    %ecx, %eax
```

%**eax**: A_0	%**eax**: A_0	%**eax**: $C_0 - A_0$	%**eax**: $C_0 - A_0 - 1$	%**eax**: A_0
%**ecx**: C_0	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: C_0

Example Peephole

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```

PC

```
notl    %eax
addl    %ecx, %eax
```

%**eax**: A_0	%**eax**: A_0	%**eax**: $C_0 - A_0$	%**eax**: $C_0 - A_0 - 1$	%**eax**: A_0	%**eax**: $\sim A_0$
%**ecx**: C_0	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: C_0	%**ecx**: C_0

Example Peephole

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```


```
notl    %eax
addl    %ecx, %eax
```

%**eax**: A_0	%**eax**: A_0	%**eax**: $C_0 - A_0$	%**eax**: $C_0 - A_0 - 1$	%**eax**: A_0	%**eax**: $\sim A_0$
%**ecx**: C_0	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: C_0	%**ecx**: C_0

Example Peephole

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```

PC

```
notl    %eax
addl    %ecx, %eax
```

%**eax**: A_0	%**eax**: A_0	%**eax**: $C_0 - A_0$	%**eax**: $C_0 - A_0 - 1$	%**eax**: A_0	%**eax**: $\sim A_0$	%**eax**: $C_0 + \sim A_0$
%**ecx**: C_0	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: C_0	%**ecx**: C_0	%**ecx**: C_0

Example Peephole

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```

PC

```
notl    %eax
addl    %ecx, %eax
```

%**eax**: A_0	%**eax**: A_0	%**eax**: $C_0 - A_0$	%**eax**: $C_0 - A_0 - 1$	%**eax**: A_0	%**eax**: $\sim A_0$	%**eax**: $C_0 + \sim A_0$
%**ecx**: C_0	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: C_0	%**ecx**: C_0	%**ecx**: C_0

Example Peephole

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```


```
notl    %eax
addl    %ecx, %eax
```

%**eax**: A_0	%**eax**: A_0	%**eax**: $C_0 - A_0$	%**eax**: $C_0 - A_0 - 1$	%**eax**: A_0	%**eax**: $\sim A_0$	%**eax**: $C_0 + \sim A_0$
%**ecx**: C_0	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: C_0	%**ecx**: C_0	%**ecx**: C_0

Example Peephole

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```

PC

```
notl    %eax
addl    %ecx, %eax
```

%**eax**: A_0	%**eax**: A_0	%**eax**: $C_0 - A_0$	%**eax**: $C_0 - A_0 - 1$
%**ecx**: C_0	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$

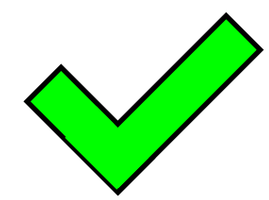
%**eax**: A_0	%**eax**: $\sim A_0$	%**eax**: $C_0 + \sim A_0$
%**ecx**: C_0	%**ecx**: C_0	%**ecx**: C_0

Example Peephole

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```


```
notl    %eax
addl    %ecx, %eax
```

%**eax**: A_0	%**eax**: A_0	%**eax**: $C_0 - A_0$	%**eax**: $C_0 - A_0 - 1$
%**ecx**: C_0	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$

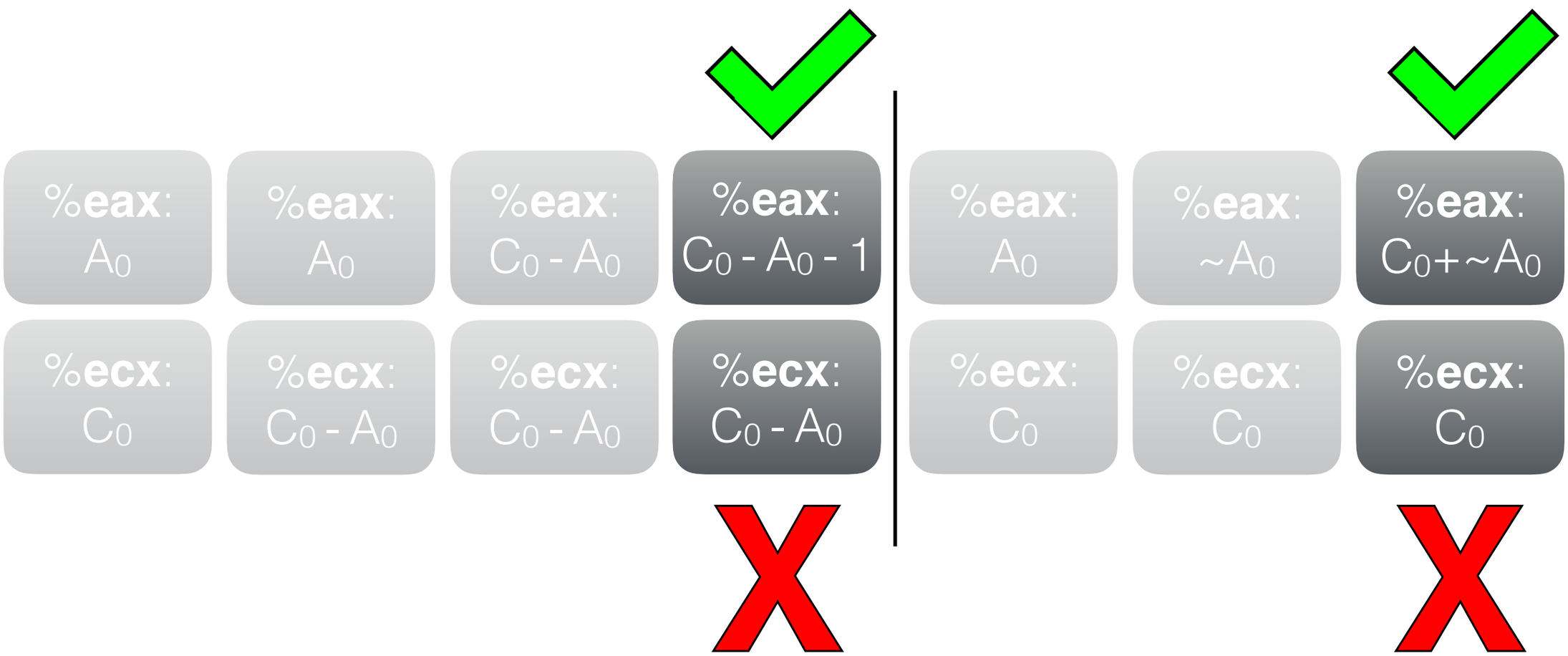


%**eax**: A_0	%**eax**: $\sim A_0$	%**eax**: $C_0 + \sim A_0$
%**ecx**: C_0	%**ecx**: C_0	%**ecx**: C_0

Example Peephole

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```

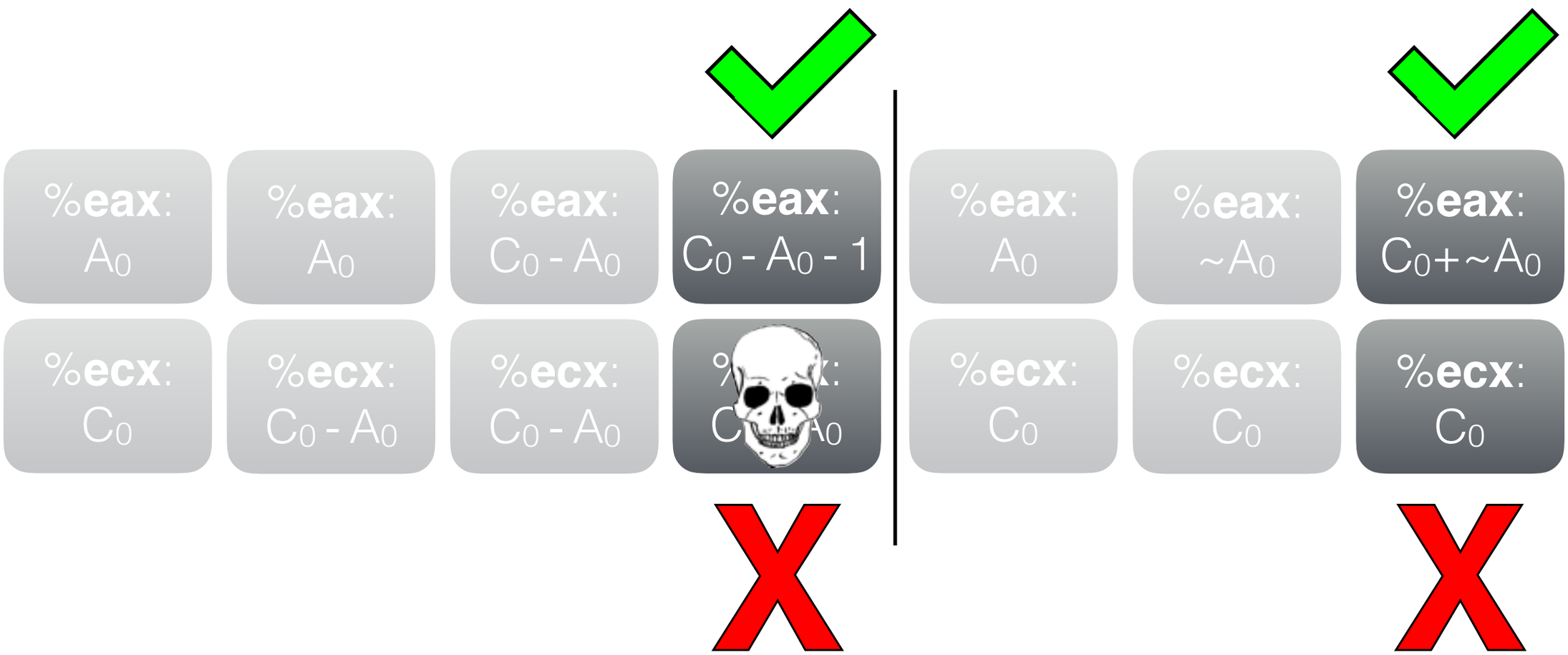

```
notl    %eax
addl    %ecx, %eax
```



Example Peephole

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```


```
notl    %eax
addl    %ecx, %eax
```



Example Peephole

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```

PC

```
notl    %eax
addl    %ecx, %eax
```

%**eax**: A_0	%**eax**: A_0	%**eax**: $C_0 - A_0$	%**eax**: $C_0 - A_0 - 1$
%**ecx**: C_0	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$

%**eax**: A_0	%**eax**: $\sim A_0$	%**eax**: $C_0 + \sim A_0$
%**ecx**: C_0	%**ecx**: C_0	%**ecx**:

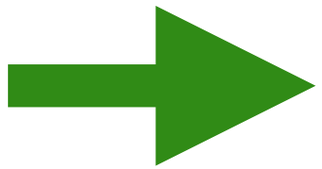
Local Correctness

Execution produces:

- Identical values in *live registers*
- **No guarantees** for dead registers
- Identical memories

Parameterization

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```



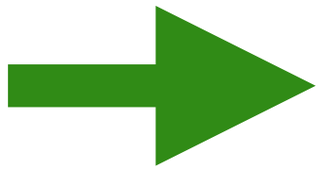
```
notl    %eax
addl    %ecx, %eax
```

Parameterization

```
∀ reg1, reg2,

reg1 ≠ reg2 →

subl    reg1, reg2
movl    reg2, reg1
decl    reg1
```

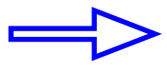


```
notl    reg1
addl    reg2, reg1
```

Outline

Local Proofs

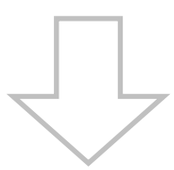
SlowInstr
SlowInstr



FastInstr
FastInstr

New Semantics

Pointers

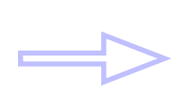


Bit Vectors

Global Correctness

…
Instr
Instr
Instr
Instr
Instr
SlowInstr
SlowInstr
Instr
Instr
Instr
Instr
Instr
Instr
…

SlowInstr
SlowInstr



FastInstr
FastInstr

…
Instr
Instr
Instr
Instr
Instr
FastInstr
FastInstr
Instr
Instr
Instr
Instr
Instr
Instr
…

? Liveness

Evaluation

Outline

Local Proofs

SlowInstr
SlowInstr
⇨
FastInstr
FastInstr
✓

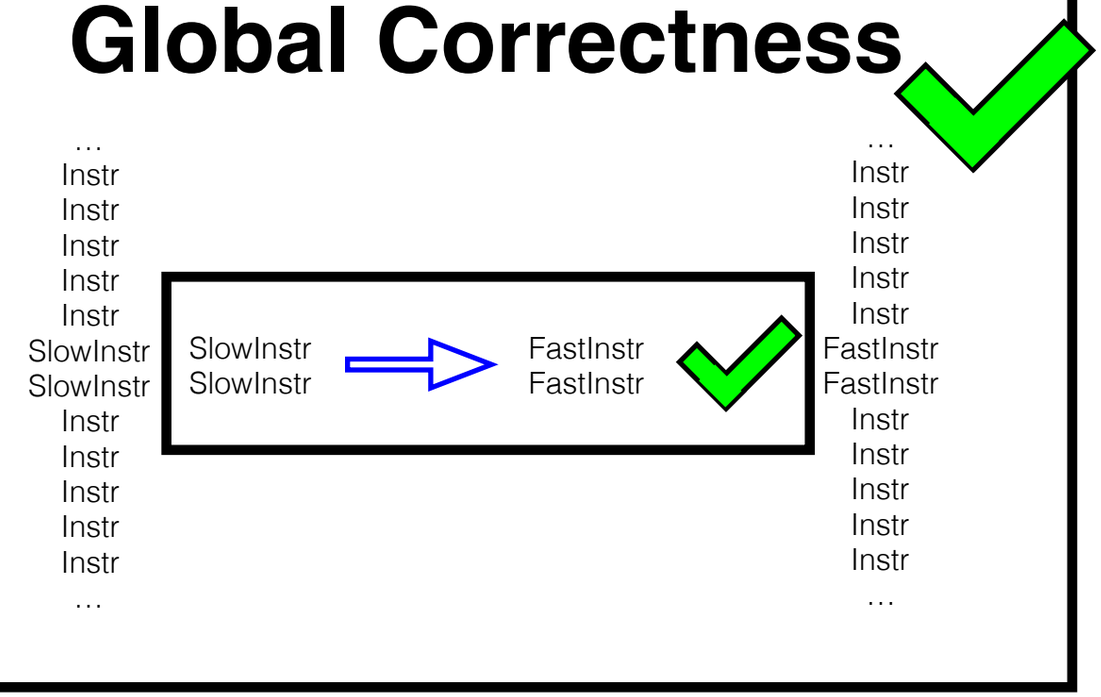
New Semantics

Pointers

⇩

Bit Vectors

Global Correctness



Liveness

Evaluation

Program Equivalence

…
Instr
Instr
Instr
Instr
Instr
SlowInstr
SlowInstr
Instr
Instr
Instr
Instr
Instr
…

…
Instr
Instr
Instr
Instr
Instr
FastInstr
FastInstr
Instr
Instr
Instr
Instr
Instr
…

Program Equivalence

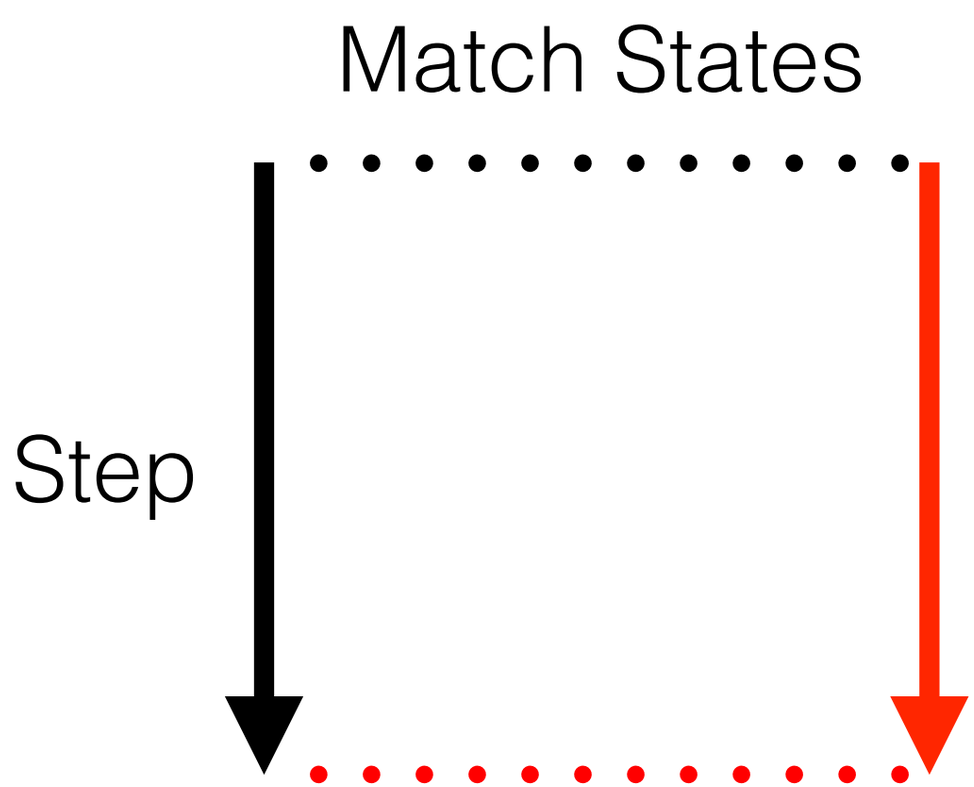
Prog:

```
…
Instr
Instr
Instr
Instr
Instr
SlowInstr
SlowInstr
Instr
Instr
Instr
Instr
Instr
…
```

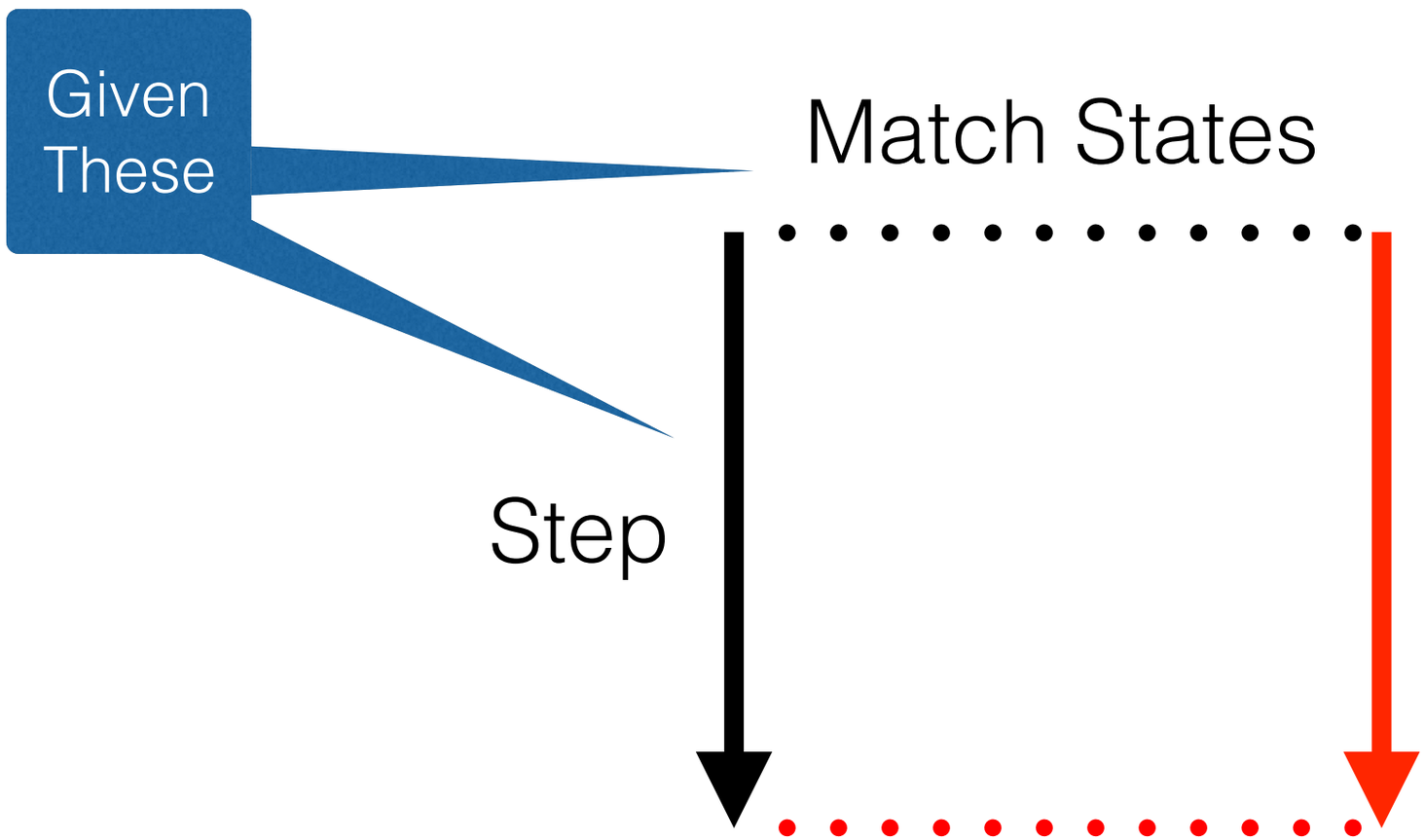
Tprog:

```
…
Instr
Instr
Instr
Instr
Instr
FastInstr
FastInstr
Instr
Instr
Instr
Instr
Instr
…
```

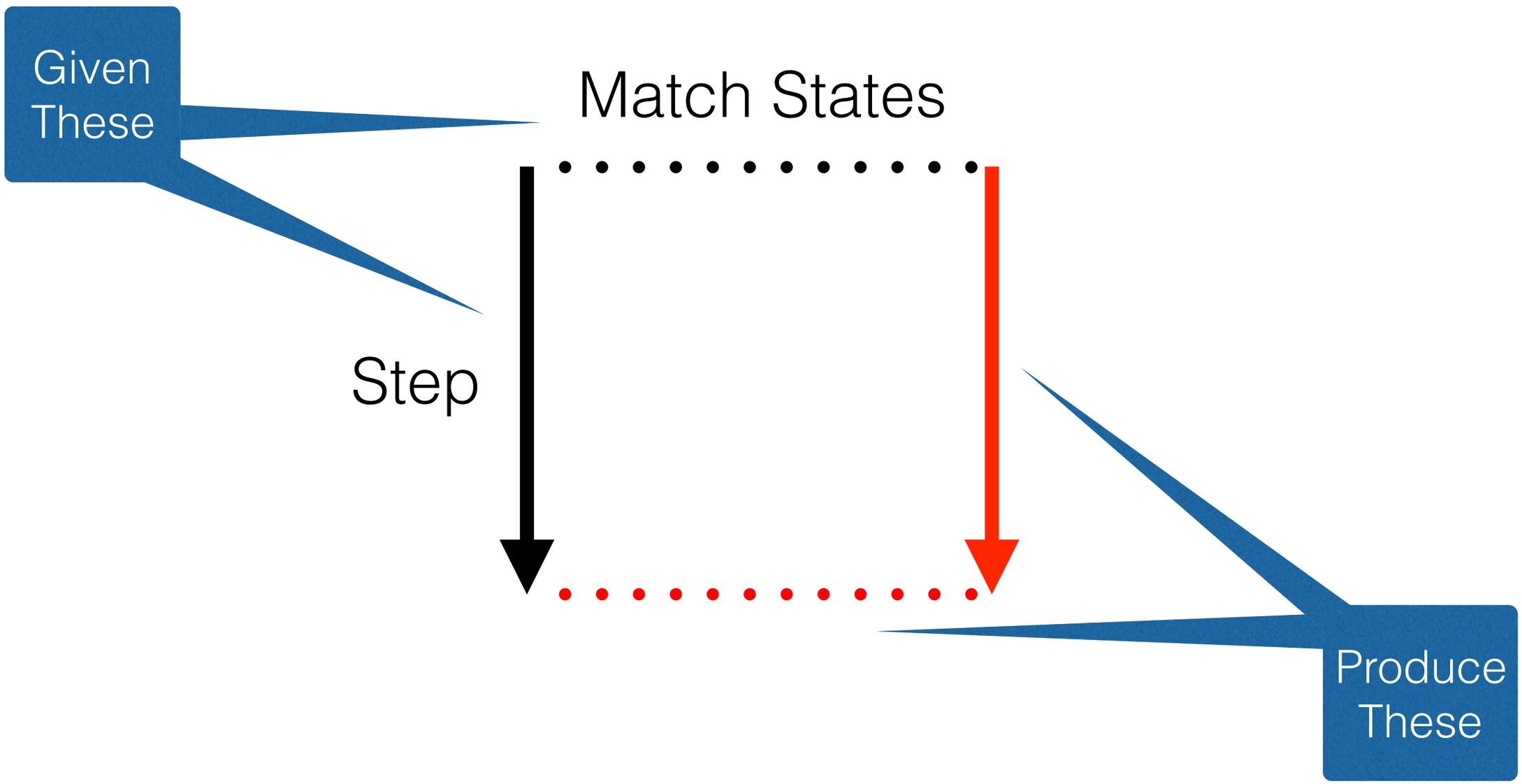
Simulation Relation



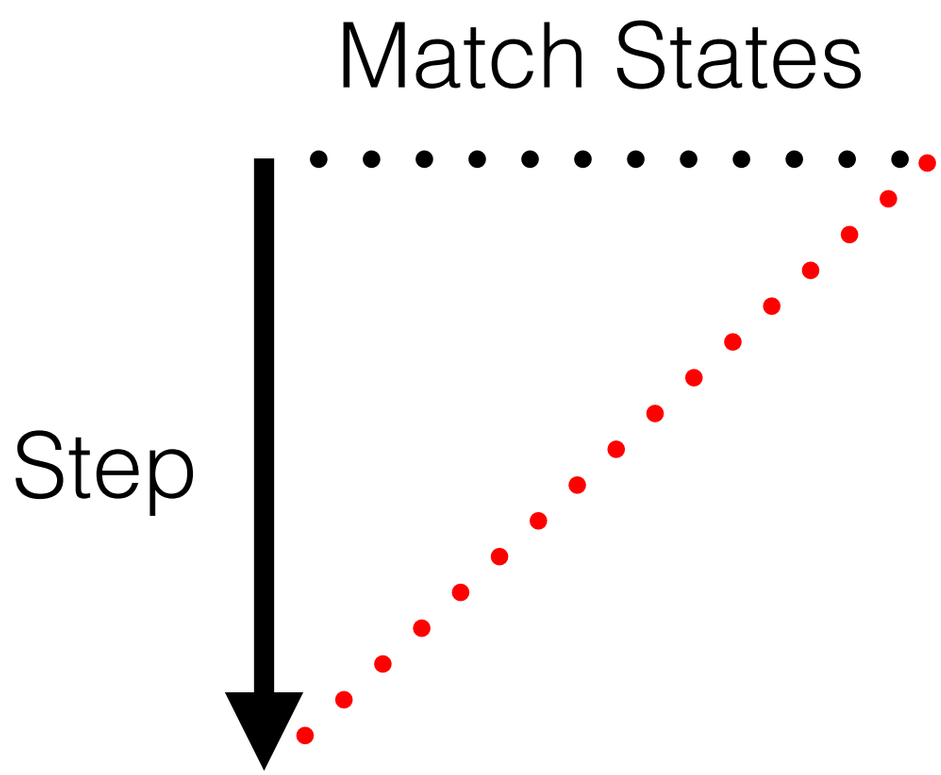
Simulation Relation



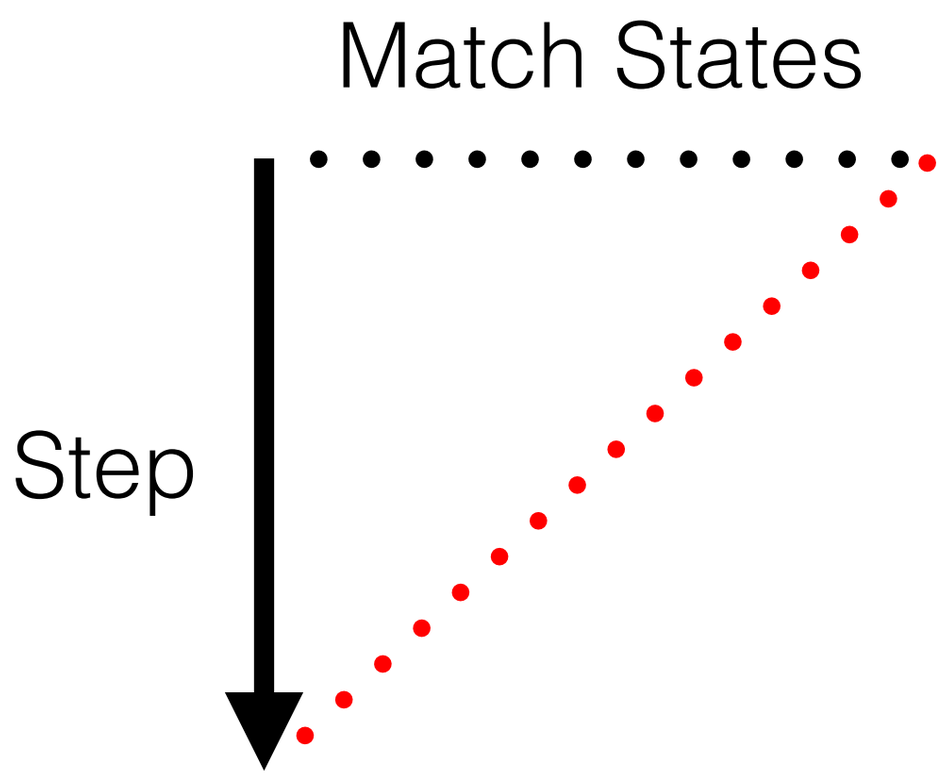
Simulation Relation



Simulation Relation

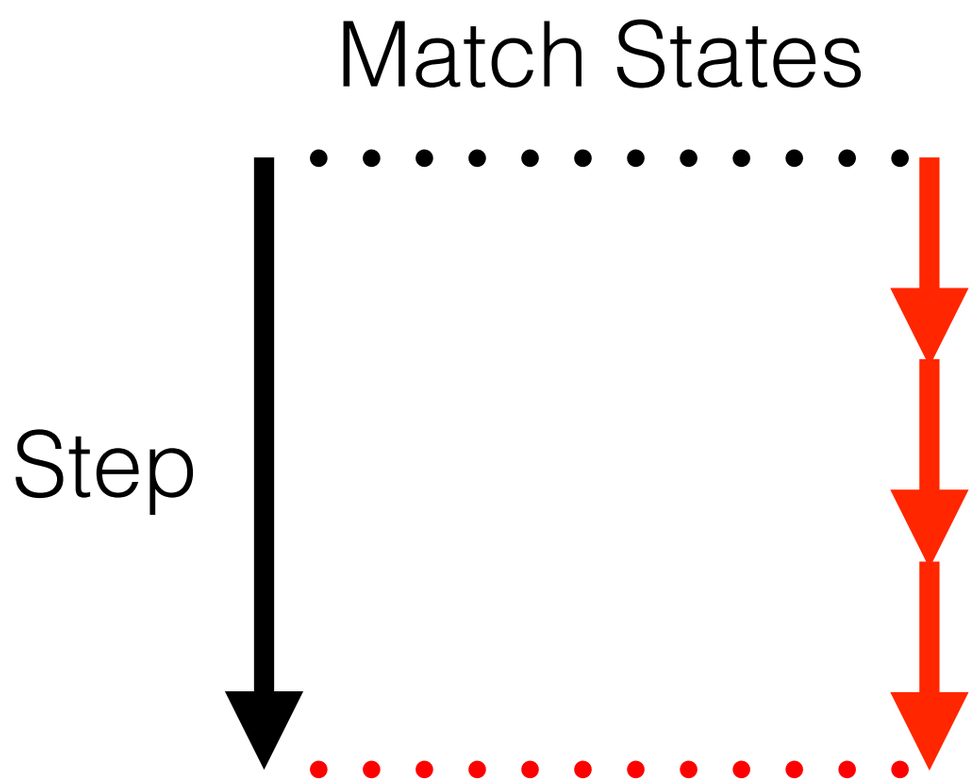


Simulation Relation



*As long as a measure decreases

Simulation Relation



Match States

Original

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```

Transformed

```
notl    %eax
addl    %ecx, %eax
nop
```

Match States

Original

Transformed

`subl %eax, %ecx`	`notl %eax`
`movl %ecx, %eax`	`addl %ecx, %eax`
`decl %eax`	`nop`

Match States

Original	Transformed
`subl %eax, %ecx`	`notl %eax`
`movl %ecx, %eax`	`addl %ecx, %eax`
`decl %eax`	`nop`

Match States

Original	Transformed
`subl %eax, %ecx`	`notl %eax`
`movl %ecx, %eax`	`addl %ecx, %eax`
`decl %eax`	`nop`

σ

σ

Match States

Original	Transformed
`subl %eax, %ecx`	`notl %eax`
`movl %ecx, %eax`	`addl %ecx, %eax`
`decl %eax`	`nop`

Match States

Original	Transformed
`subl %eax, %ecx`	`notl %eax`
`movl %ecx, %eax`	`addl %ecx, %eax`
`decl %eax`	`nop`

σ ..... σ

Local Peephole Correctness

Program Properties

Necessary properties of programs to rewrite:

- Correct Liveness Information
- Single Entry
- Single Exit
- etc. (see paper)

Single Entry

```
decl    %eax
pushl   %esp
leal    0(%ecx,%edx,4), %esi
incl    %edx
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
popl    %edx
decl    %edi
addl    $31, %eax
```

Single Entry


```
decl    %eax
pushl   %esp
leal    0(%ecx,%edx,4), %esi
incl    %edx
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
popl    %edx
decl    %edi
addl    $31, %eax
```

Single Entry

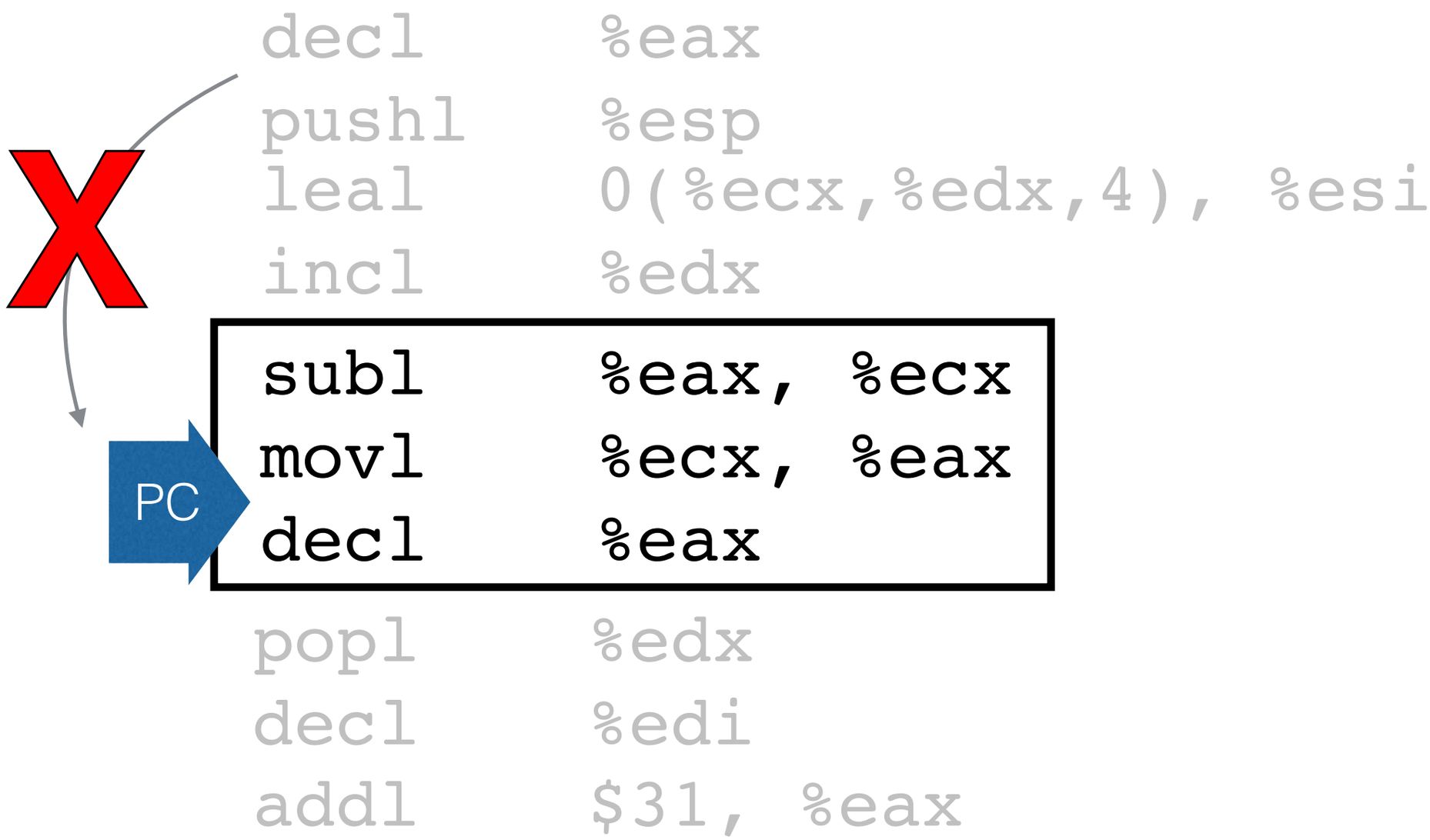
```
decl    %eax
pushl   %esp
leal    0(%ecx,%edx,4), %esi
incl    %edx
```

PC

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```

```
popl    %edx
decl    %edi
addl    $31, %eax
```

Single Entry



Single Exit

```
decl    %eax
pushl   %esp
leal    0(%ecx,%edx,4), %esi
incl    %edx
jmp     .L1
L1:
decl    %eax
popl    %edx
decl    %edi
addl    $31, %eax
```

Single Exit

```
L1:
decl    %eax
pushl   %esp
leal    0(%ecx,%edx,4), %esi
incl    %edx
jmp     .L1
L1:
decl    %eax
popl    %edx
decl    %edi
addl    $31, %eax
```

Single Exit

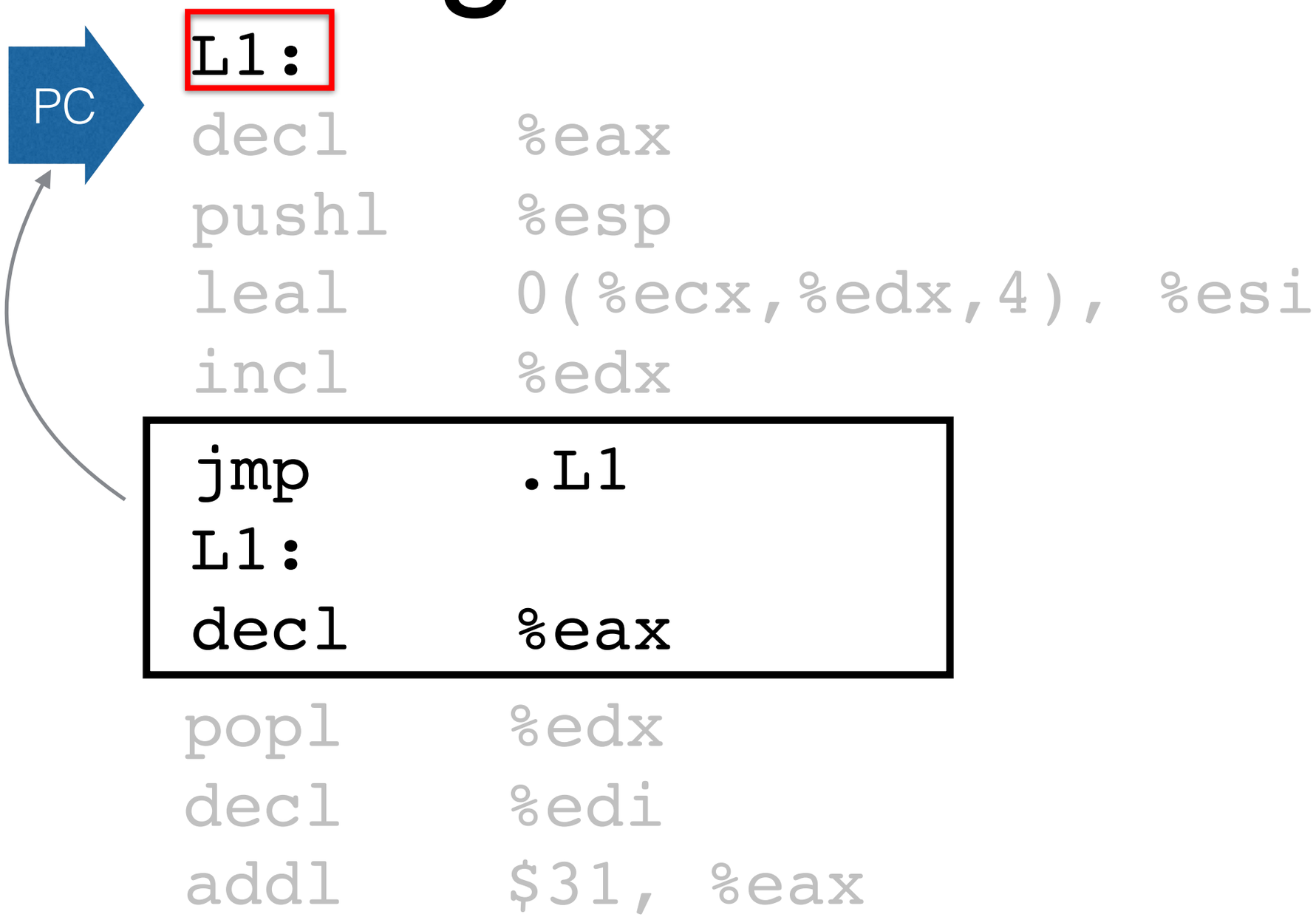
```
L1:
decl    %eax
pushl   %esp
leal    0(%ecx,%edx,4), %esi
incl    %edx
```

PC

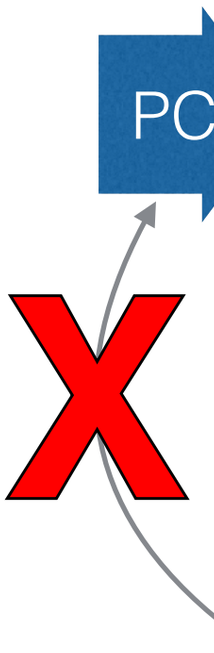
```
jmp     .L1
L1:
decl    %eax
```

```
popl    %edx
decl    %edi
addl    $31, %eax
```

Single Exit


PC
L1:
decl %eax
pushl %esp
leal 0(%ecx,%edx,4), %esi
incl %edx
jmp .L1
L1:
decl %eax
popl %edx
decl %edi
addl $31, %eax

Single Exit




```
L1:
decl    %eax
pushl   %esp
leal    0(%ecx,%edx,4), %esi
incl    %edx
jmp     .L1
L1:
decl    %eax
popl    %edx
decl    %edi
addl    $31, %eax
```

Outline

Local Proofs

SlowInstr
SlowInstr
⇨
FastInstr
FastInstr
✓

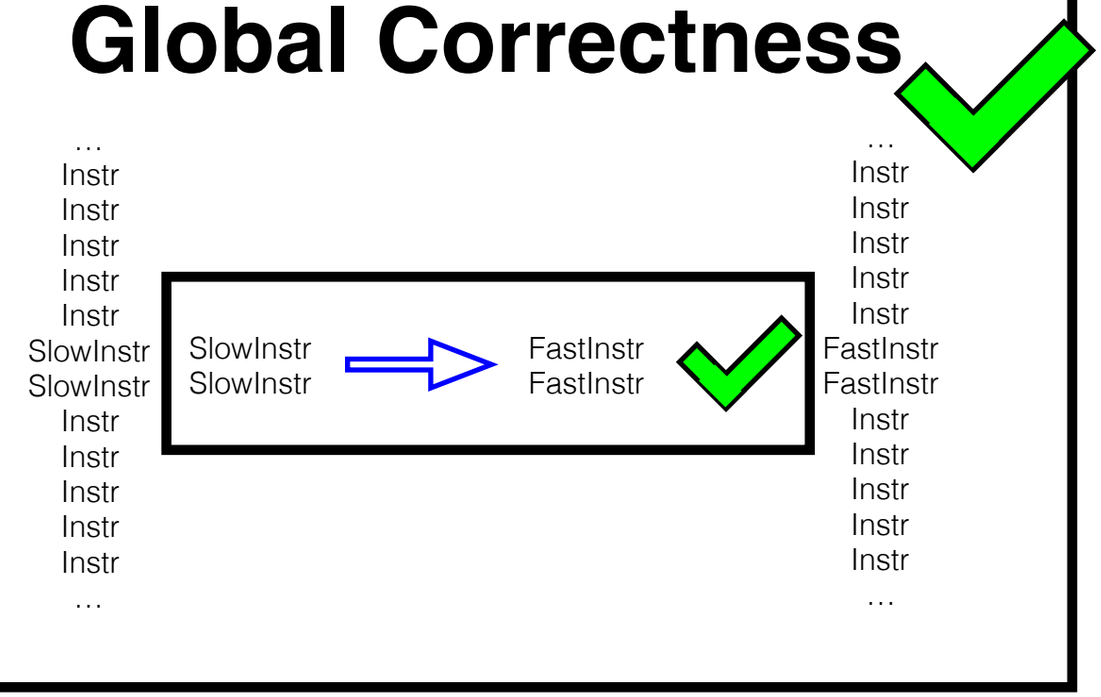
New Semantics

Pointers

⇩

Bit Vectors

Global Correctness



Liveness

Evaluation

Outline

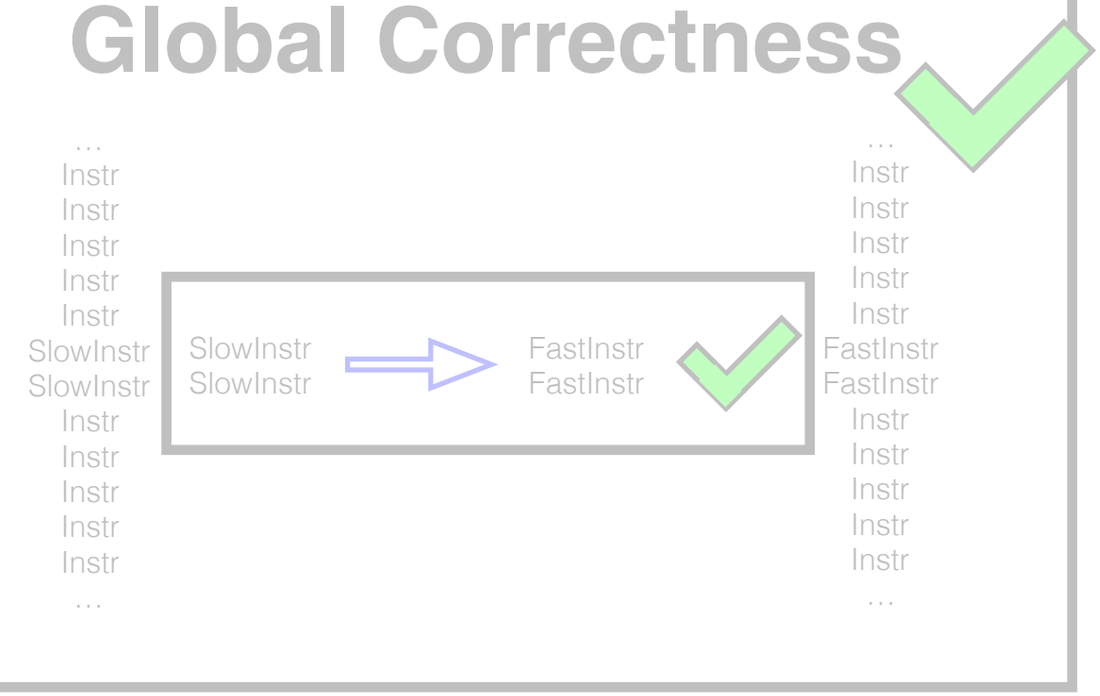
Local Proofs

SlowInstr
SlowInstr
⇨
FastInstr
FastInstr
✓

New Semantics

Pointers
⇩
Bit Vectors

Global Correctness



? Liveness

Evaluation

Example

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```

```
notl    %eax
addl    %ecx, %eax
```

PC

%**eax**: A_0	%**eax**: A_0	%**eax**: $C_0 - A_0$	%**eax**: $C_0 - A_0 - 1$	%**eax**: A_0	%**eax**: $\sim A_0$	%**eax**: $C_0 + \sim A_0$
%**ecx**: C_0	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: C_0	%**ecx**: C_0	%**ecx**: C_0

Example

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```

```
notl    %eax
addl    %ecx, %eax
```

PC

%**eax**: A_0	%**eax**: A_0	%**eax**: $C_0 - A_0$	%**eax**: $C_0 - A_0 - 1$	%**eax**: A_0	%**eax**: $\sim A_0$	%**eax**: $C_0 + \sim A_0$
%**ecx**: C_0	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: C_0	%**ecx**: C_0	%**ecx**:

Example

```
leal 0(%ecx,%eax) %eax
```

Example

```
leal 0(%ecx,%eax) %eax
```

→

```
addl %ecx, %eax
```

Example

```
leal 0(%ecx,%eax) %eax
```

Preserves Flags

```
addl %ecx, %eax
```

Example

```
leal 0(%ecx,%eax) %eax
```

Preserves Flags

Writes Flags

```
addl %ecx, %eax
```

Example

```
leal 0(%ecx,%eax) %eax
```

Preserves Flags

Writes Flags

```
addl %ecx, %eax
```

```
movl    $0, %eax
```

Example

```
leal 0(%ecx,%eax) %eax
```

Preserves Flags

Writes Flags

```
addl %ecx, %eax
```

```
movl    $0, %eax
```

```
xorl    %eax, %eax
```

Example

```
leal 0(%ecx,%eax) %eax
```

Preserves Flags

Writes Flags

```
addl %ecx, %eax
```

```
movl    $0, %eax
```

Preserves Flags

```
xorl    %eax, %eax
```

Example

```
leal 0(%ecx,%eax) %eax
```

Preserves Flags

Writes Flags

```
addl %ecx, %eax
```

```
movl    $0, %eax
```

Preserves Flags

Writes Flags

```
xorl    %eax, %eax
```

Liveness

Will a particular register be read before it's written at this program point?

Maybe Yes

Definitely No

Liveness

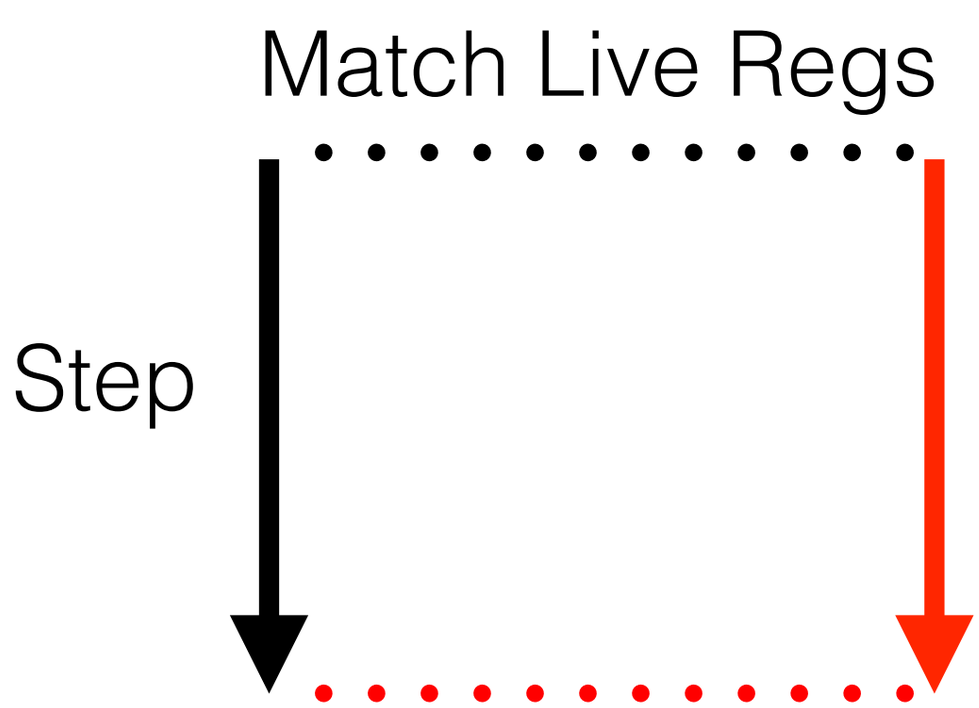
Will a particular register be read before it's written at this program point?

Maybe Yes

Definitely No

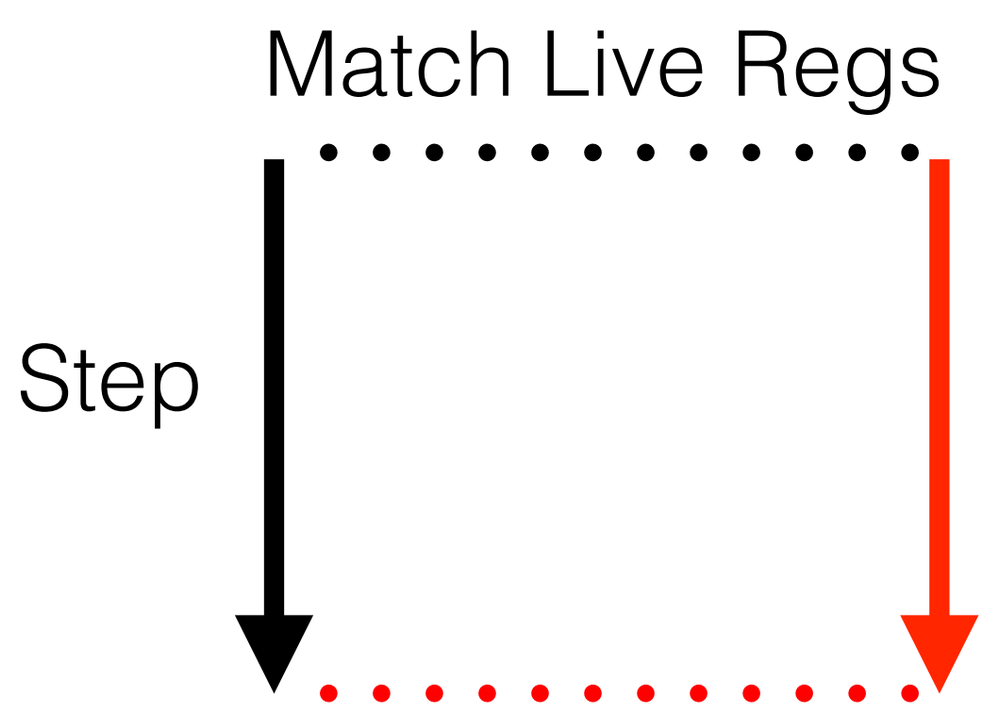
Liveness

- Iterative worklist style algorithm
- Verified against simulation style specification



Liveness

- Iterative worklist style algorithm
- Verified against simulation style specification



* *Verified assuming facts about calling convention*

Outline

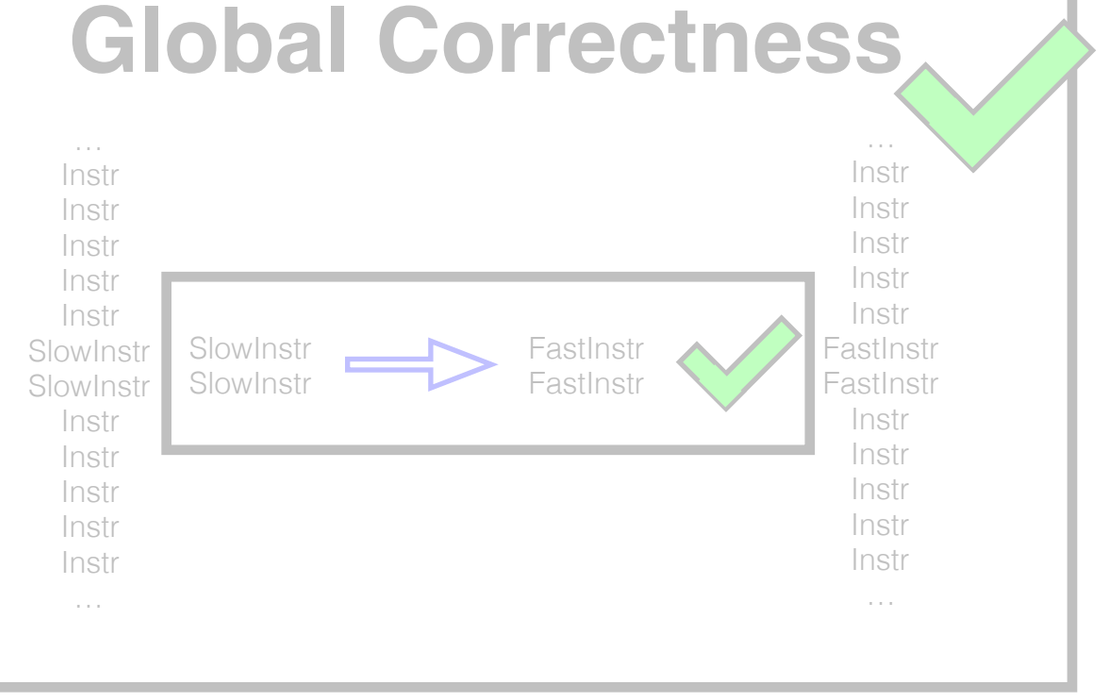
Local Proofs

SlowInstr
SlowInstr
⇨
FastInstr
FastInstr
✓

New Semantics

Pointers
⇩
Bit Vectors

Global Correctness



? Liveness

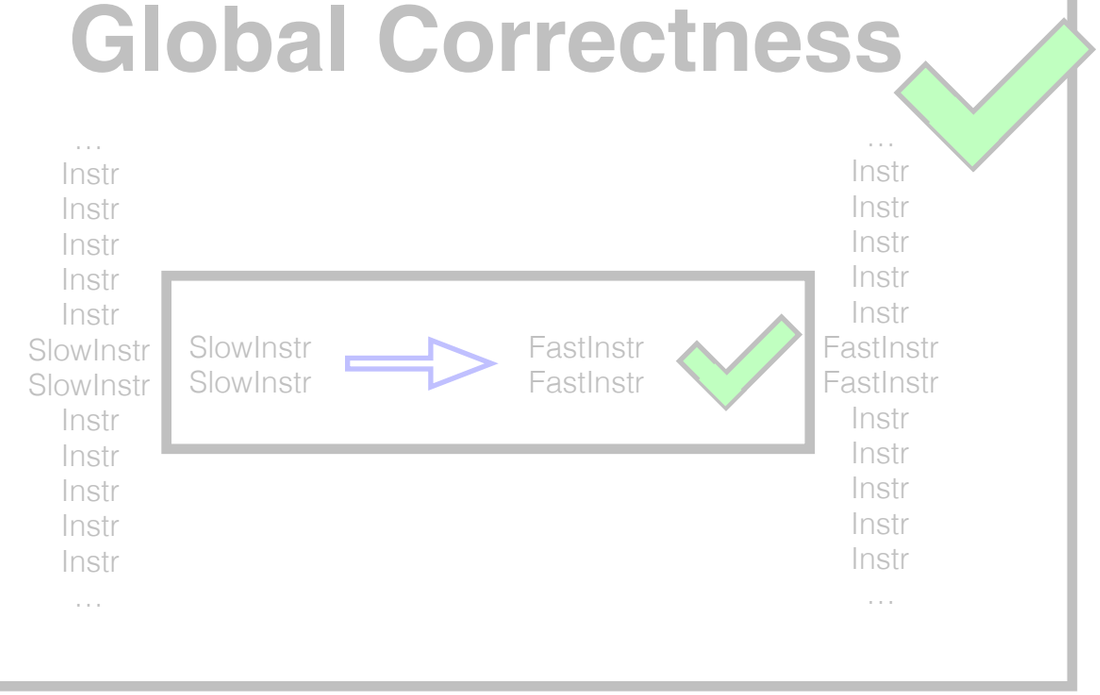
Evaluation

Outline

Local Proofs

SlowInstr
SlowInstr
⇨
FastInstr
FastInstr
✓

Global Correctness



New Semantics

Pointers

⇩

Bit Vectors

Liveness

Evaluation

Example

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```

```
notl    %eax
addl    %ecx, %eax
```

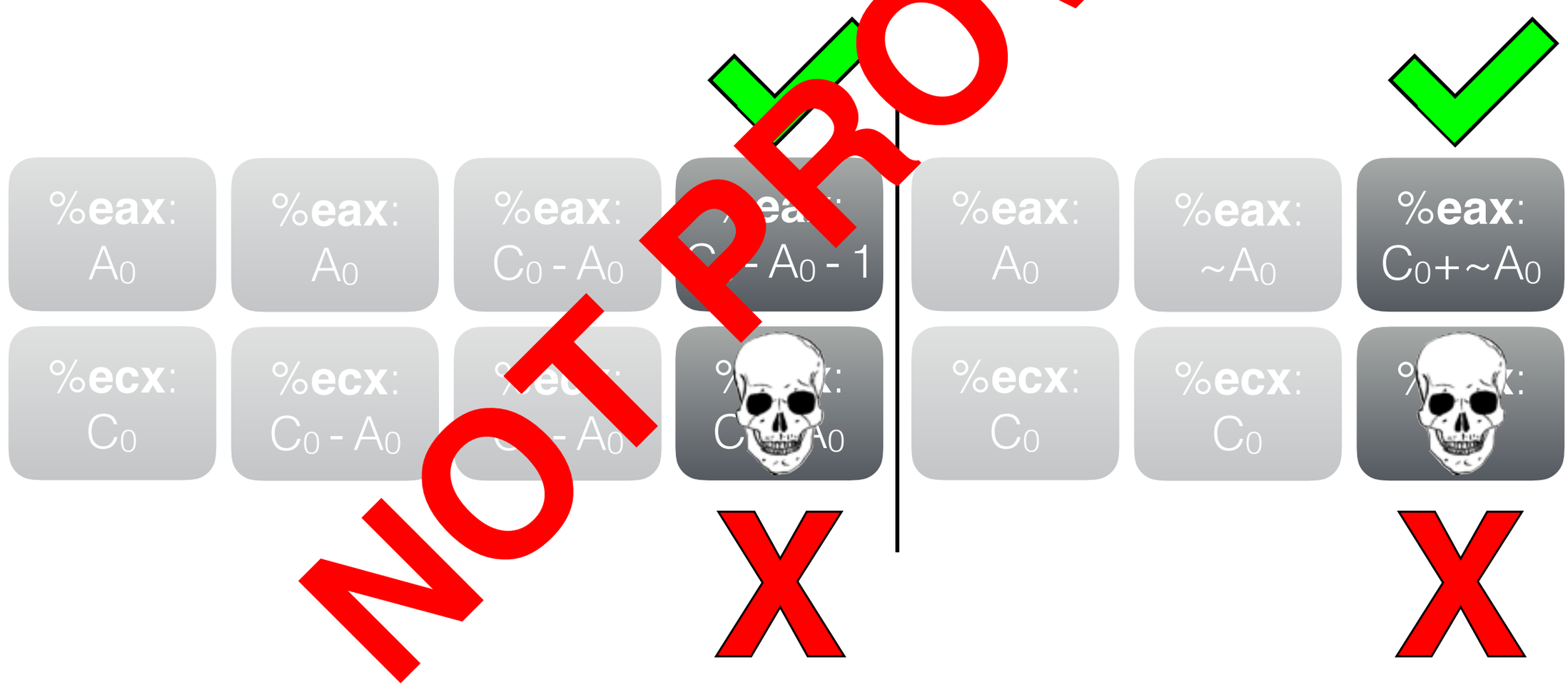
PC

%**eax**: A_0	%**eax**: A_0	%**eax**: $C_0 - A_0$	%**eax**: $C_0 - A_0 - 1$	%**eax**: A_0	%**eax**: $\sim A_0$	%**eax**: $C_0 + \sim A_0$
%**ecx**: C_0	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: $C_0 - A_0$	%**ecx**: C_0	%**ecx**: C_0	%**ecx**:

Example

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```

```
notl    %eax
addl    %ecx, %eax
```



Example

%**eax**:
$C_0 - A_0 - 1$

%**eax**:
$C_0 + \sim A_0$

Exam

Recall:
x - y - 1 = x+ ~y
for two's complement

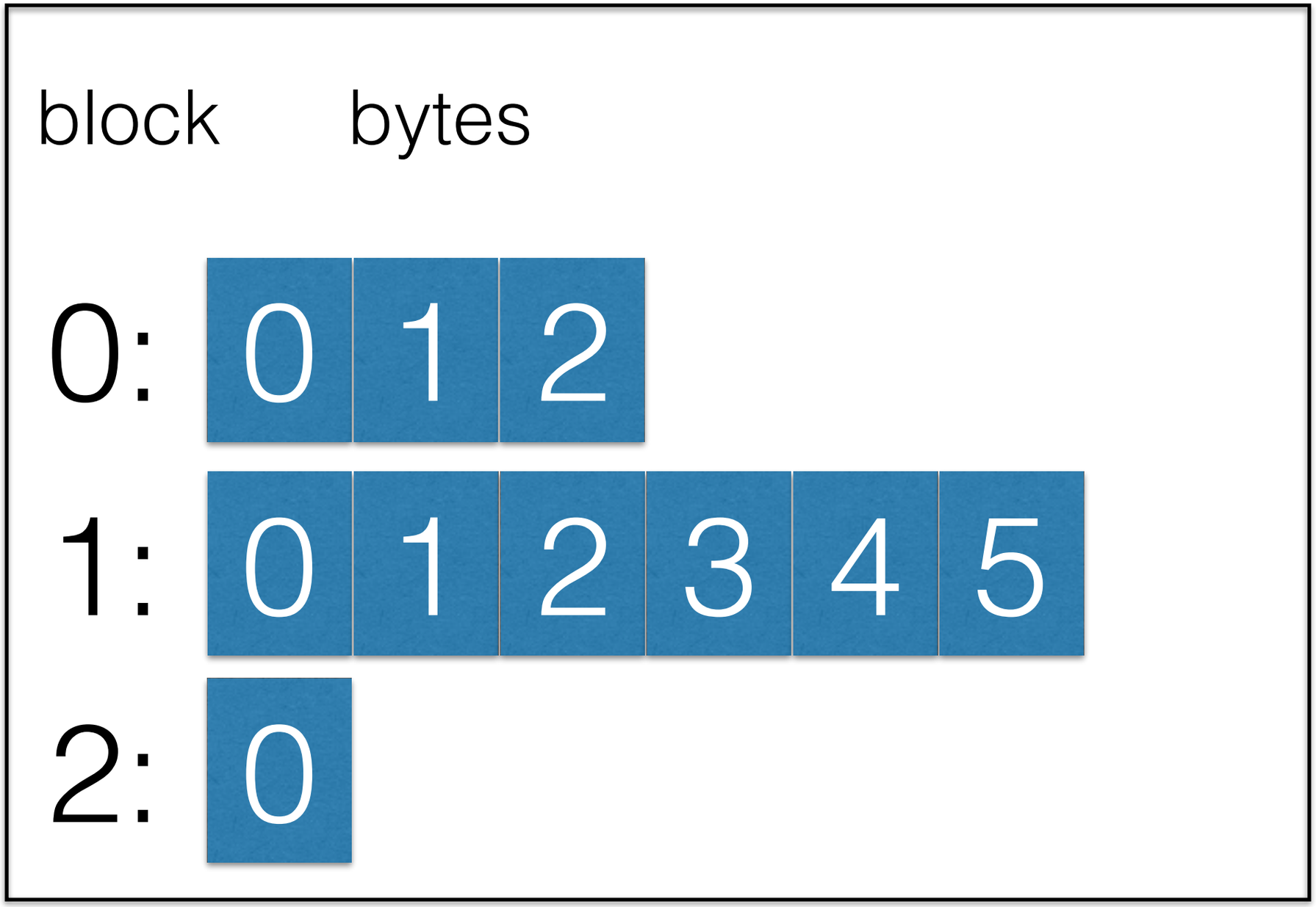
%**eax**:
$C_0 - A_0 - 1$

%**eax**:
$C_0 + \sim A_0$

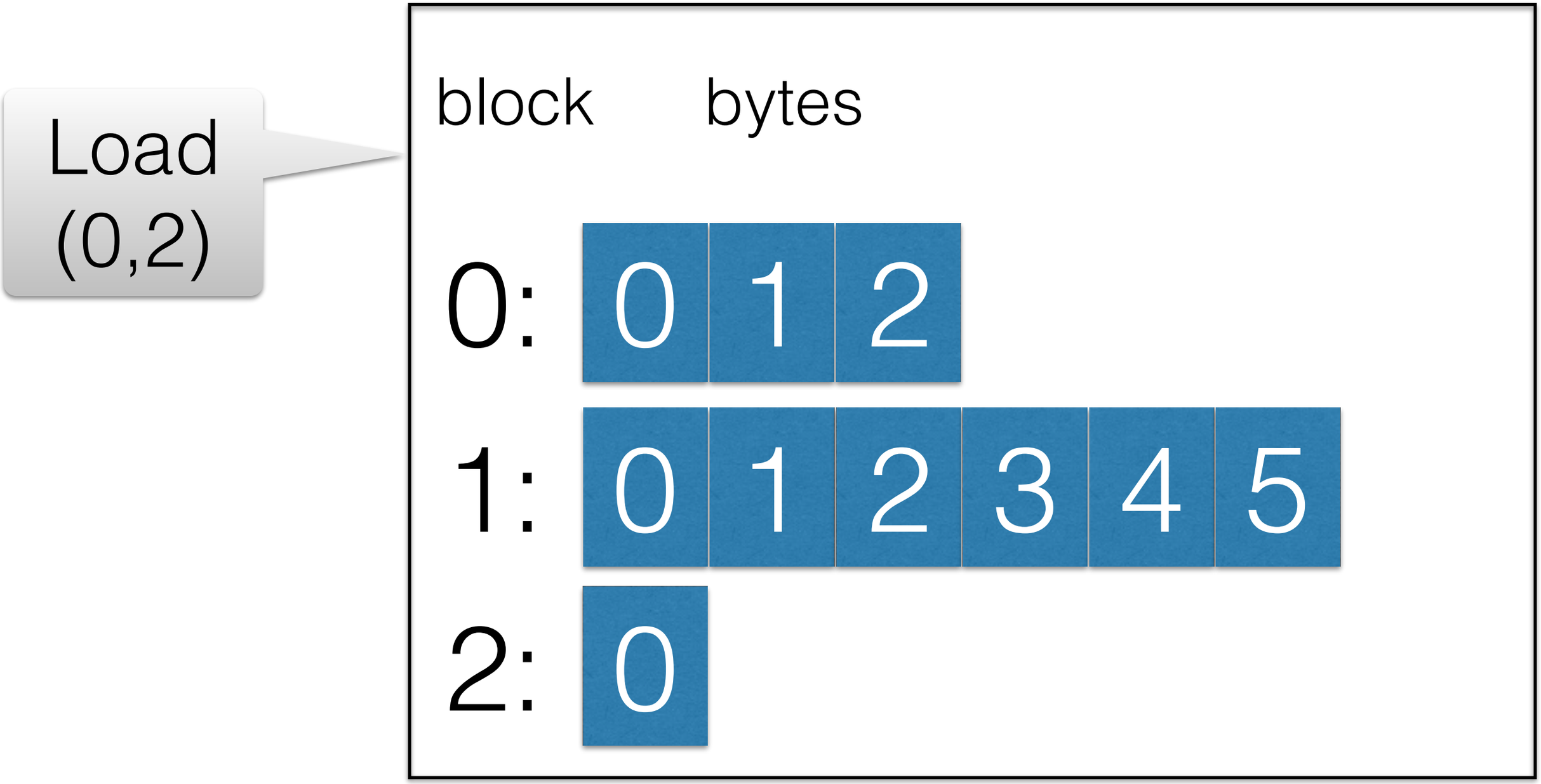
Assembly Semantics

```
Inductive val:Type :=
  | Vundef: val
  | Vint: int -> val
  | Vlong: int64 -> val
  | Vfloat: float -> val
  | Vsingle: float32 -> val
  | Vptr: block -> int -> val.
```

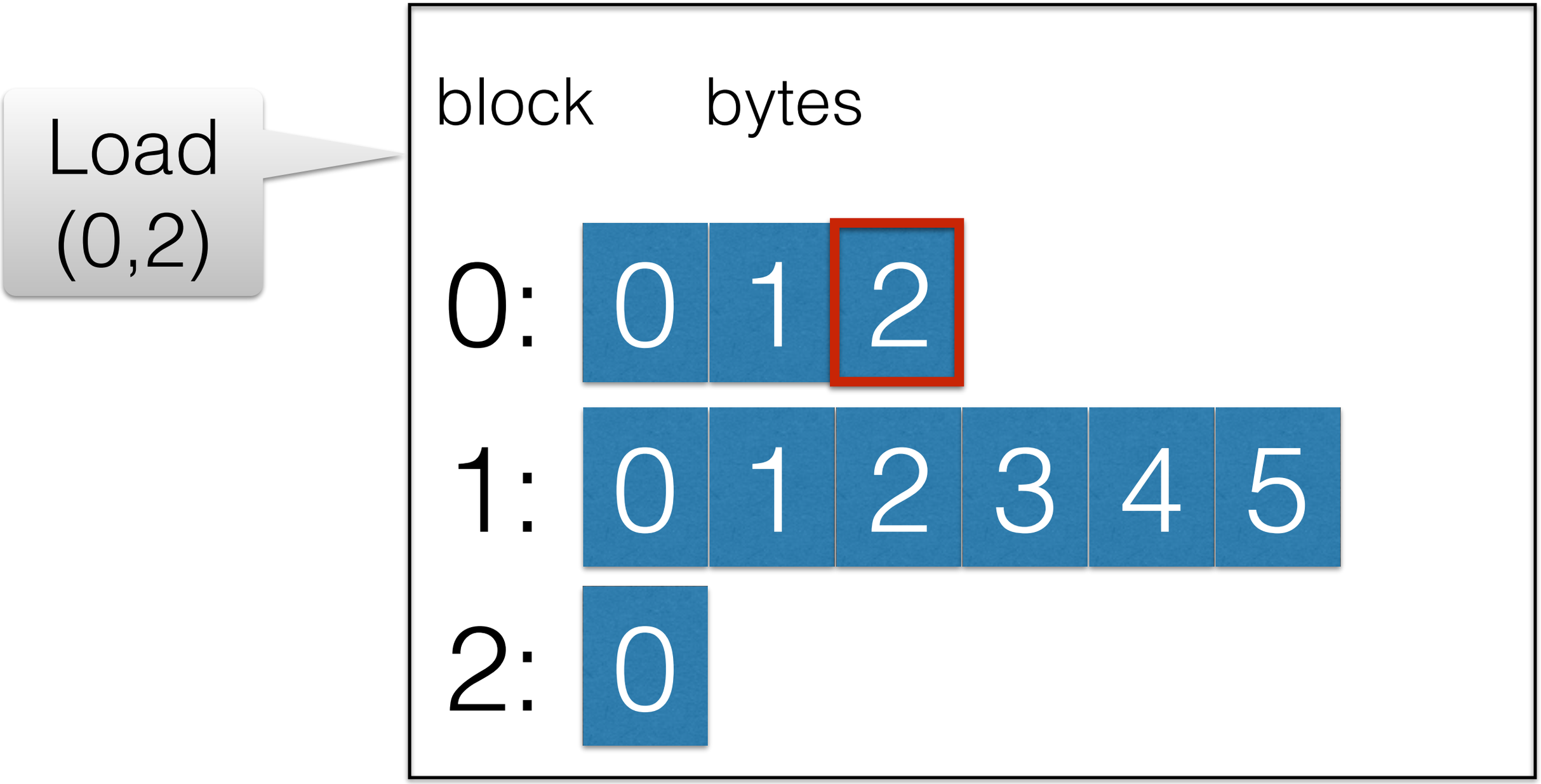
Addressing Memory



Addressing Memory



Addressing Memory



Assembly Semantics

```
Definition sub (v1 v2: val): val :=
  match v1, v2 with
    | Vint n1, Vint n2 =>
       Vint(Int.sub n1 n2)
    | Vptr b1 ofs1, Vint n2 =>
      Vptr b1 (Int.sub ofs1 n2)
  | Vptr b1 ofs1, Vptr b2 ofs2 =>
      if eq_block b1 b2
       then Vint(Int.sub ofs1 ofs2)
       else Vundef
  | _, _ => Vundef
 end.
```

Assembly Semantics

```
Definition sub (v1 v2: val): val :=
  match v1, v2 with
  | Vint n1, Vint n2 =>
      Vint(Int.sub n1 n2)
  | Vptr b1 ofs1, Vint n2 =>
      Vptr b1 (Int.sub ofs1 n2)
  | Vptr b1 ofs1, Vptr b2 ofs2 =>
      if eq_block b1 b2
       then Vint(Int.sub ofs1 ofs2)
       else Vundef
  | _, _ => Vundef
end.
```

Assembly Semantics

```
Definition sub (v1 v2: val): val :=
  match v1, v2 with
   | Vint n1, Vint n2 =>
```

```
| Vptr b1 ofs1, Vint n2 =>
      Vptr b1 (Int.sub ofs1 n2)
```

```
    if eq_block b1 b2
     then Vint(Int.sub ofs1 ofs2)
     else Vundef
  | _, _ => Vundef
end.
```

Assembly Semantics

```
Definition sub (v1 v2: val): val :=
  match v1, v2 with
   | Vint n1, Vint n2 =>
      Vint(Int.sub n1 n2)
   | Vptr b1 ofs1, Vint n2 =>
      Vptr b1 (Int.sub ofs1 n2)
```

```
| Vptr b1 ofs1, Vptr b2 ofs2 =>
      if eq_block b1 b2
       then Vint(Int.sub ofs1 ofs2)
       else Vundef
```

Example

%**eax**:
$C_0 - A_0 - 1$

%**eax**:
$C_0 + \sim A_0$

Example

A_0 = Vptr b ofs1
C_0 = Vptr b ofs2

%**eax**:
$C_0 - A_0 - 1$

%**eax**:
$C_0 + \sim A_0$

Example

A_0 = Vptr b ofs1
C_0 = Vptr b ofs2

%**eax**:
$C_0 - A_0 - 1$

%**eax**:
$C_0 + \sim A_0$

$C_0 - A_0 - 1$ =
Vint (ofs2 - ofs1 - 1)

Example

A_0 = Vptr b ofs1
C_0 = Vptr b ofs2

%**eax**:
C_0 - A_0 - 1

C_0 - A_0 - 1 =
Vint (ofs2 - ofs1 - 1)

%**eax**:
C_0+~A_0

C_0 + ~A_0 =
Vptr b ofs2 + Vundef =
Vundef

New Semantics

Getting Between

```
pinj : md -> block -> ofs -> option int32

psur : md -> int32 -> option (block * ofs)
```

`pinj` has option return type due to pigeonhole

`psur` has option return type since whole address space not always in use

Behaviors getting between

```
pinj : md -> block -> ofs -> option int32
psur : md -> int32 -> option (block * ofs)
```

- `pinj` remembers a mapping
- `pinj` allows for pointer arithmetic (+)
- `pinj` is injective within the same block
- if `pinj` of b,ofs is `NULL`, that b,ofs are invalid for memory access
- `pinj` and `psur` are inverses when the b,ofs is valid for memory access
- `pinj` preserves pointer comparison within the same block

Behaviors getting between

```
pinj : md -> block -> ofs -> option int32
psur : md -> int32 -> option (block * ofs)
```

- `pinj` remembers a mapping
- `pinj` allows for pointer arithmetic (+)
- `pinj` is injective within the same block
- if `pinj` of b,ofs is NULL that b,ofs are invalid for memory access
- `pinj` and `psur` are inverses when the b,ofs is valid for memory access
- `pinj` preserves pointer comparison within the same block

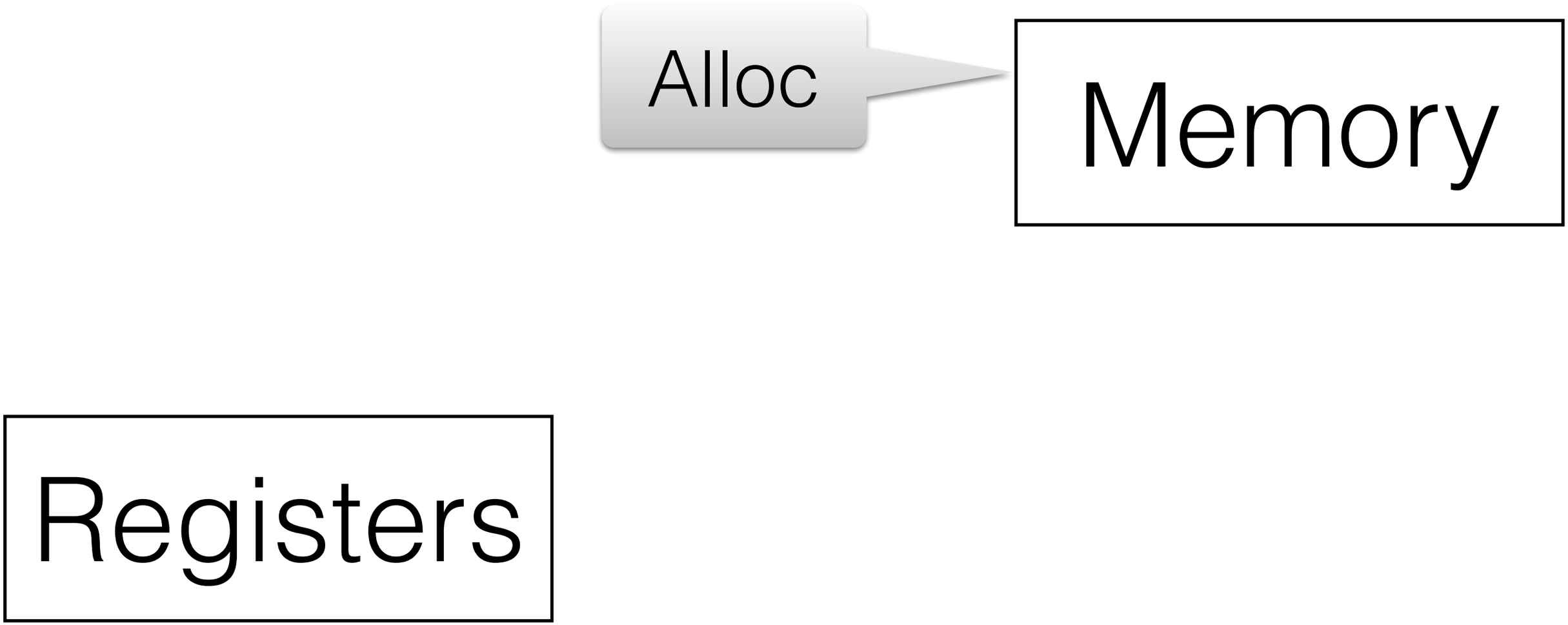
Axiomatize System Memory Allocator

Pointers $\longleftrightarrow$ Integers

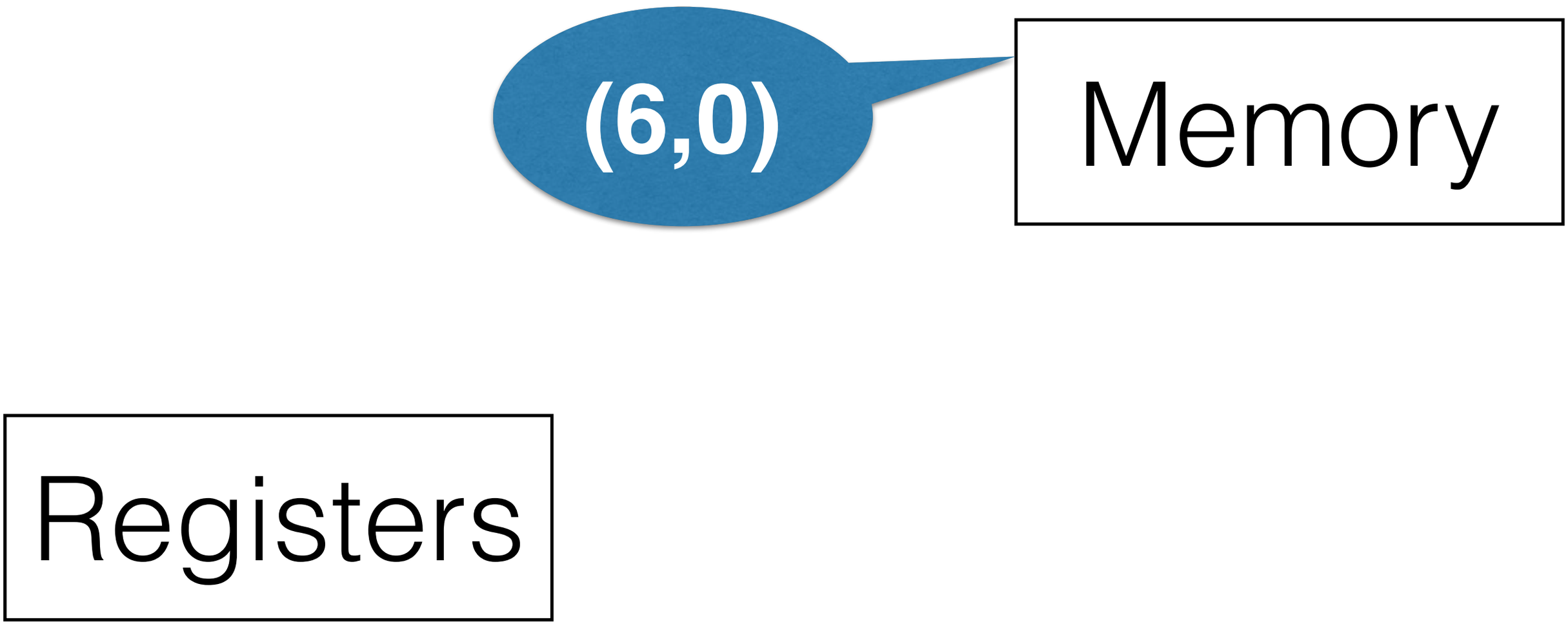
Memory

Registers

Pointers ⟷ Integers



Pointers $\longleftrightarrow$ Integers

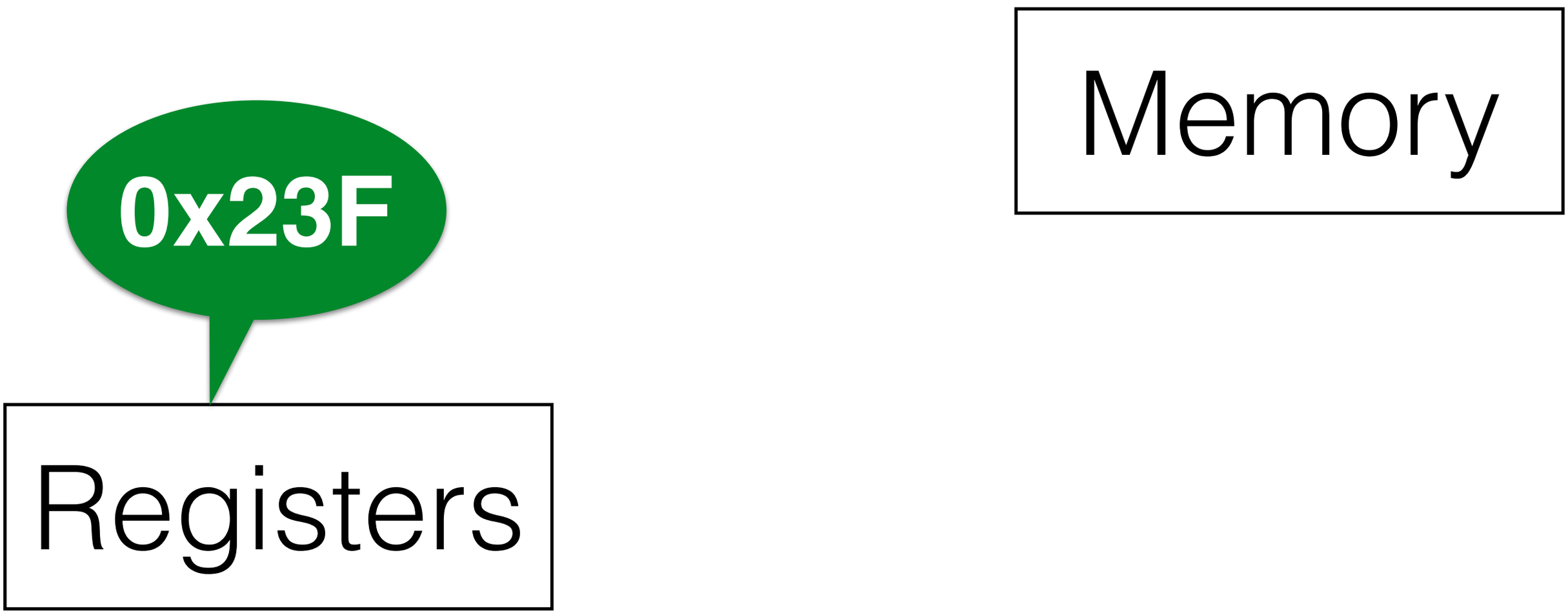


Pointers $\longleftrightarrow$ Integers

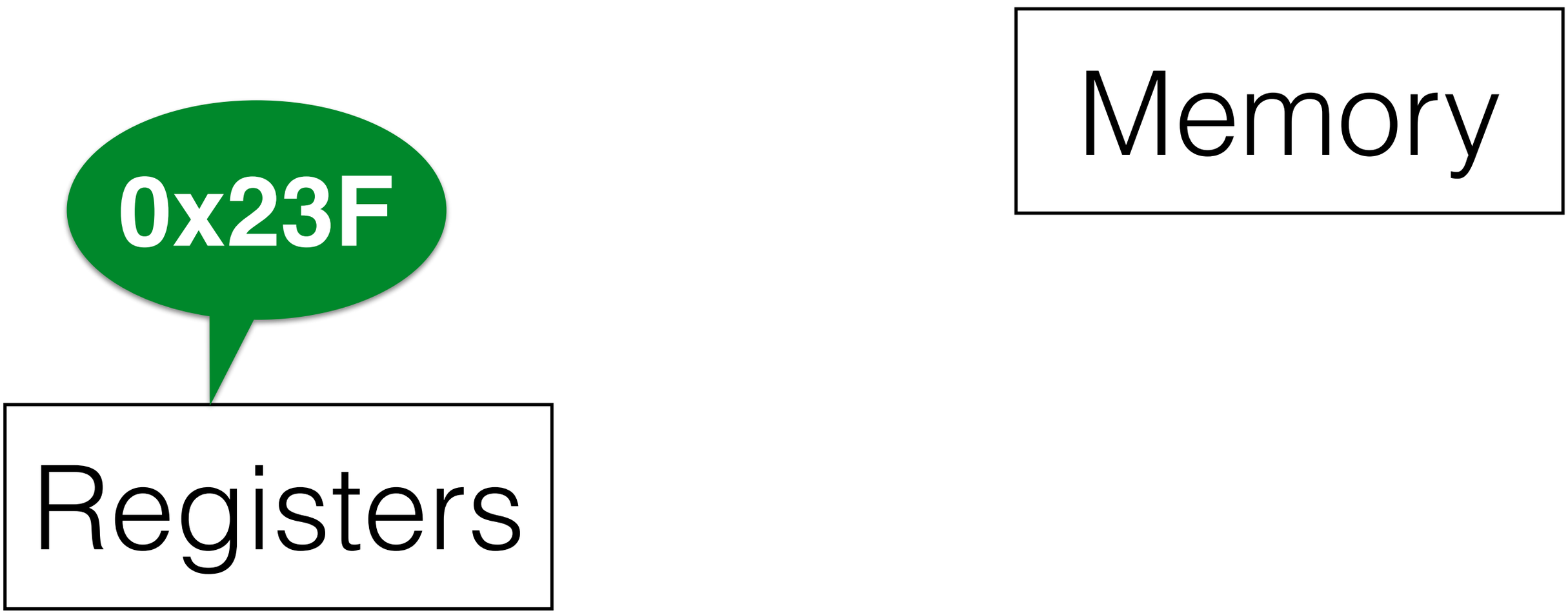
Memory

Registers

Pointers $\longleftrightarrow$ Integers



Pointers $\longleftrightarrow$ Integers

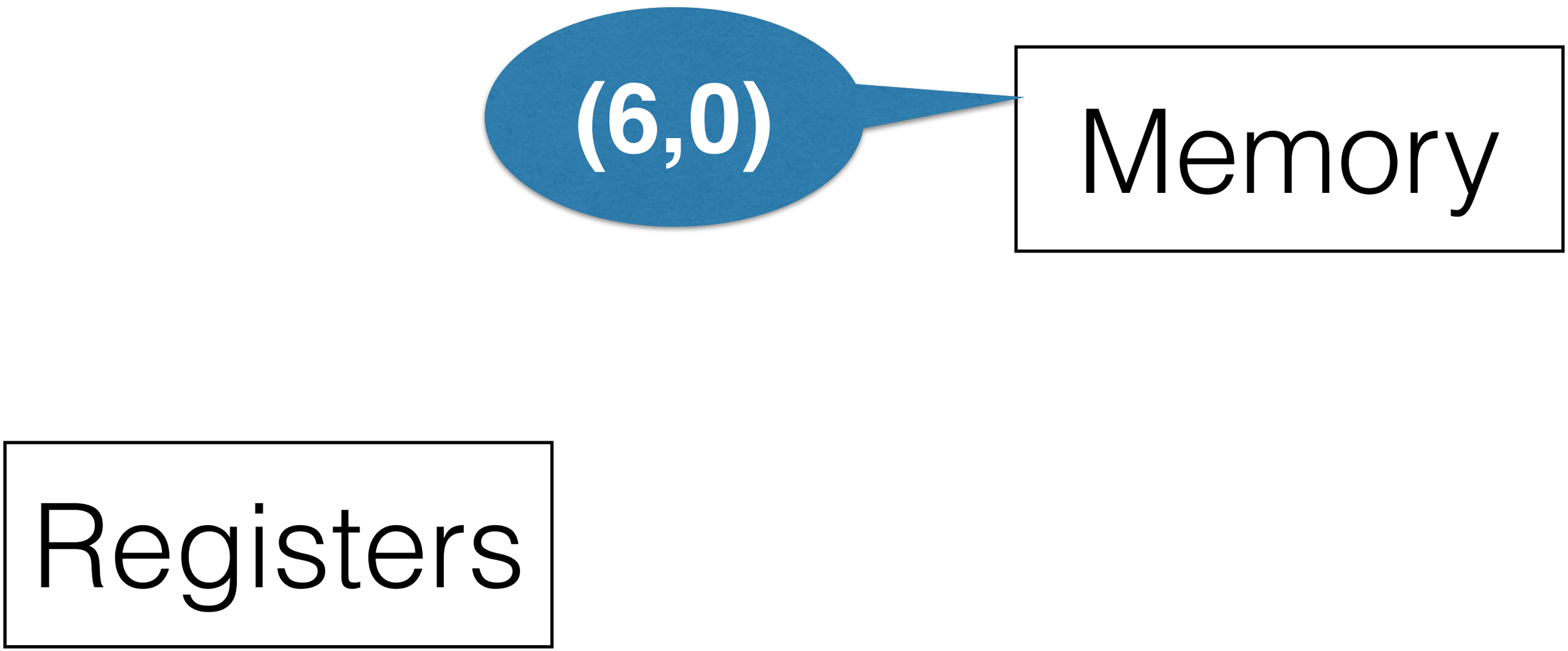


Pointers $\longleftrightarrow$ Integers

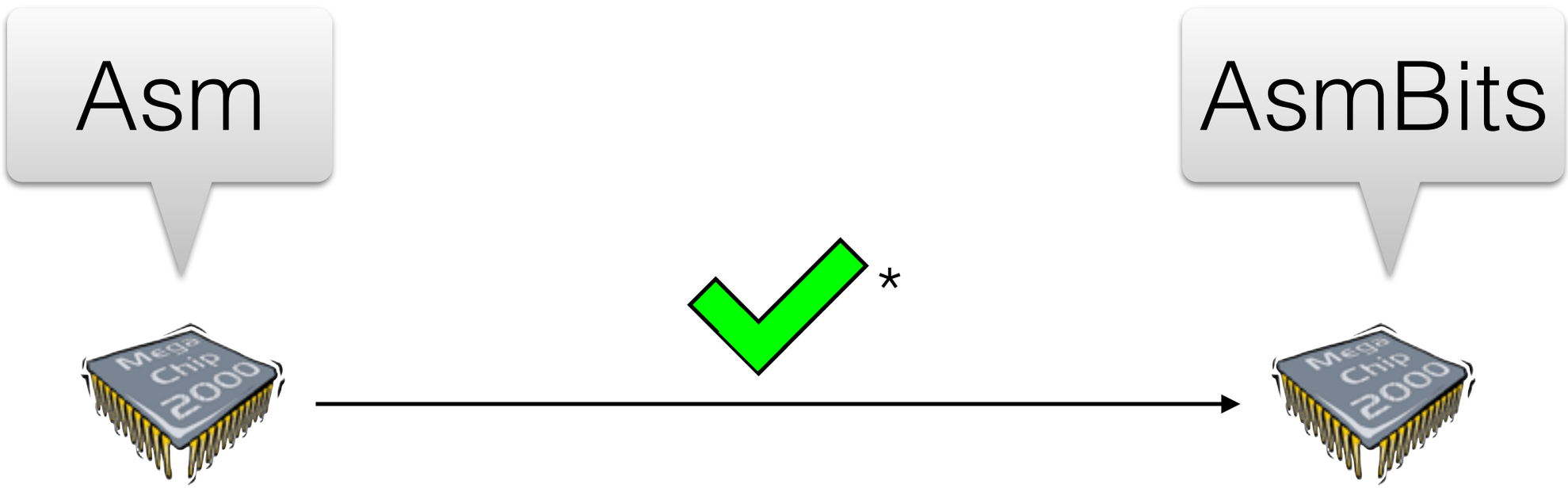
Memory

Registers

Pointers $\longleftrightarrow$ Integers



Proof



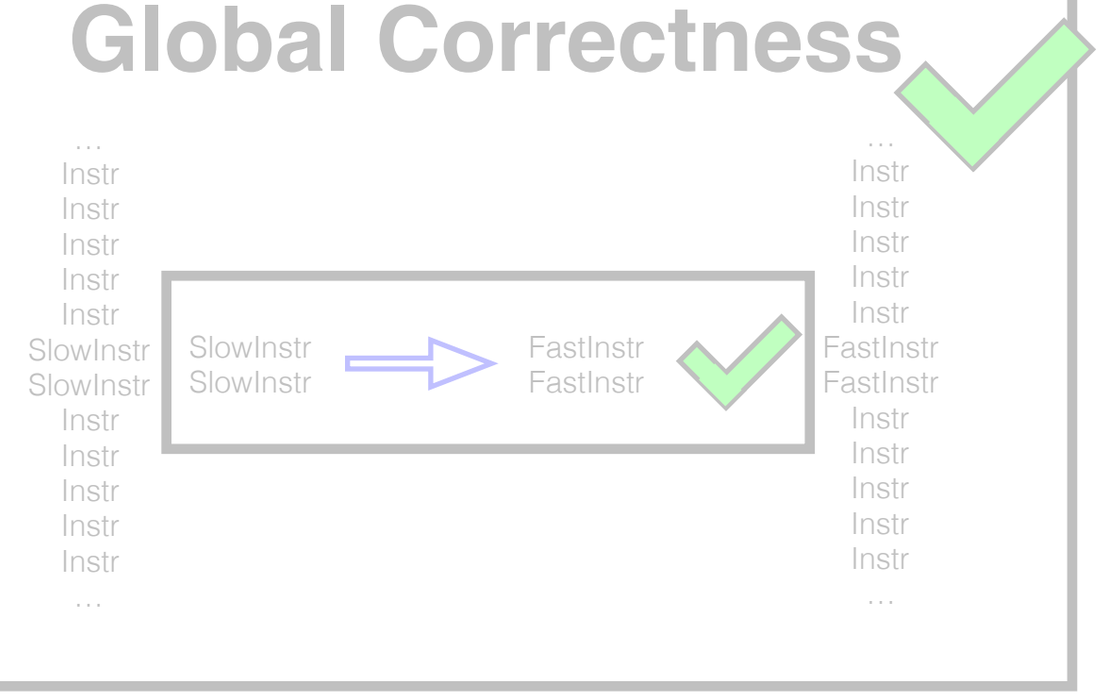
*As long as allocation doesn't fail

Outline

Local Proofs

SlowInstr
SlowInstr
⇨
FastInstr
FastInstr
✓

Global Correctness



New Semantics

Pointers

⇩

Bit Vectors

Liveness

Evaluation

Outline

Local Proofs

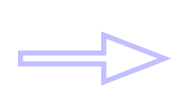
SlowInstr
SlowInstr

FastInstr
FastInstr

Global Correctness

…
Instr
Instr
Instr
Instr
Instr
SlowInstr
SlowInstr
Instr
Instr
Instr
Instr
Instr
Instr
Instr
…

SlowInstr
SlowInstr

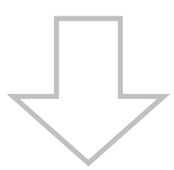


FastInstr
FastInstr

…
Instr
Instr
Instr
Instr
Instr
FastInstr
FastInstr
Instr
Instr
Instr
Instr
Instr
Instr
Instr
…

New Semantics

Pointers



Bit Vectors

Liveness

Evaluation

Expressiveness

Expressiveness

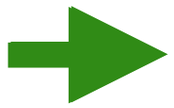
S. Bansal and A. Aiken,
Automatic Generation of
Peephole Superoptimizers,
ASPLOS 2006

Expressiveness

S. Bansal and A. Aiken, Automatic Generation of Peephole Superoptimizers, ASPLOS 2006

- Replace Subtraction with Not

```
subl    %eax, %ecx
movl    %ecx, %eax
decl    %eax
```



```
notl    %eax
addl    %ecx, %eax
```

Expressiveness

S. Bansal and A. Aiken, Automatic Generation of Peephole Superoptimizers, ASPLOS 2006

- Replace Subtraction with Not
- Conditional Jump to Conditional Move

```
test %ecx, %ecx
je .L0
mov %edx, %ebx
.L0
```

→

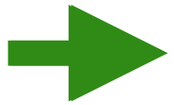
```
test %ecx, %ecx
cmovne %edx %ebx
```

Expressiveness

S. Bansal and A. Aiken, Automatic Generation of Peephole Superoptimizers, ASPLOS 2006

- Replace Subtraction with Not
- Conditional Jump to Conditional Move
- Swap Idiom to Exchange Instruction

```
mov %eax, %ecx
mov %edx, %eax
mov %ecx, %edx
```



```
xchg %eax, %edx
```

Expressiveness

S. Bansal and A. Aiken, Automatic Generation of Peephole Superoptimizers, ASPLOS 2006

- Replace Subtraction with Not
- Conditional Jump to Conditional Move
- Swap Idiom to Exchange Instruction
- Conditional Mask to Conditional Move

```
setg %al
movzbl %al, %eax
dec %eax
and %eax, %esi
```

→

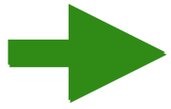
```
mov $0, %eax
cmovg %eax, %esi
```

Expressiveness

S. Bansal and A. Aiken, Automatic Generation of Peephole Superoptimizers, ASPLOS 2006

- Replace Subtraction with Not
- Conditional Jump to Conditional Move
- Swap Idiom to Exchange Instruction
- Conditional Mask to Conditional Move
- Change Constant in Program Text

```
mov $8, %eax
sub %ecx, %eax
dec %eax
```



```
mov $7, %eax
sub %ecx, %eax
```

Expressiveness

S. Bansal and A. Aiken, Automatic Generation of Peephole Superoptimizers, ASPLOS 2006

- Replace Subtraction with Not
- Conditional Jump to Conditional Move
- Swap Idiom to Exchange Instruction
- Conditional Mask to Conditional Move
- Change Constant in Program Text
- Eliminate Redundant Load

```
mov %eax, -20(%ebp)
mov -20(%ebp), %ecx
```

→

```
mov %eax, -20(%ebp)
mov %eax, %ecx
```

Expressiveness

28 peepholes verified

Average of 70 lines of proof per peephole

Code Size

	Specification	Proof	Total
Peek	6000	10000	16000
AsmBits	3300	5500	8800
Peephole Lib	2000	3100	5100
Total	11300	18600	**29900**

SHA-256

```
movl    %eax, 56(%esp)
movl    56(%esp), %edx
andl    $15, %edx
movl    0(%ecx,%edx,4), %eax
```

```
movl    %eax, 56(%esp)
andl    $15, %eax
movl    0(%ecx,%eax,4), %eax
```

A. Appel, Verification of a Cryptographic Primitive, TOPLAS 2015

SHA-256

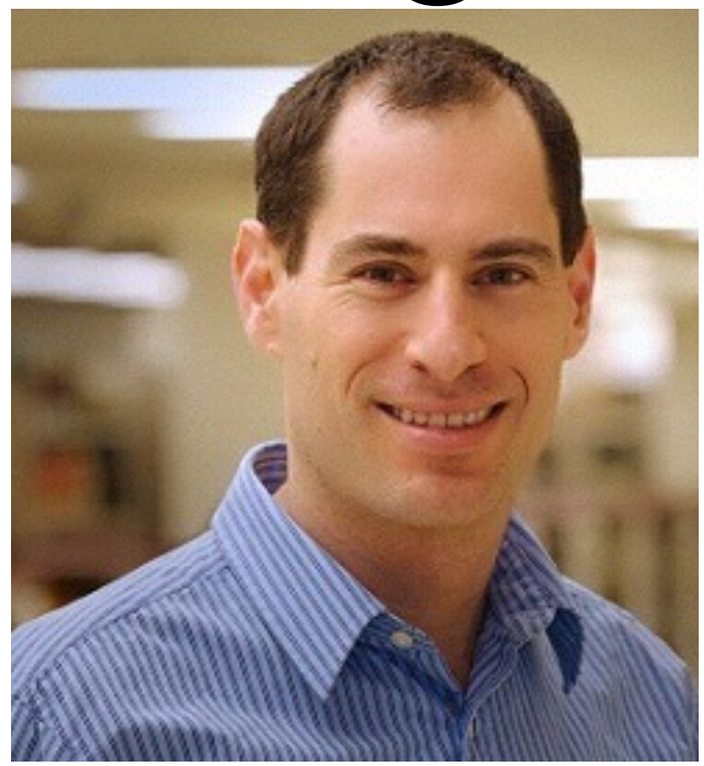
```
movl    %eax, 56(%esp)
movl    56(%esp), %edx
andl    $15, %edx
movl    0(%ecx,%edx,4), %eax
```

4% speedup!

```
movl    %eax, 56(%esp)
andl    $15, %eax
movl    0(%ecx,%eax,4), %eax
```

A. Appel, Verification of a Cryptographic Primitive, TOPLAS 2015

Acknowledgements



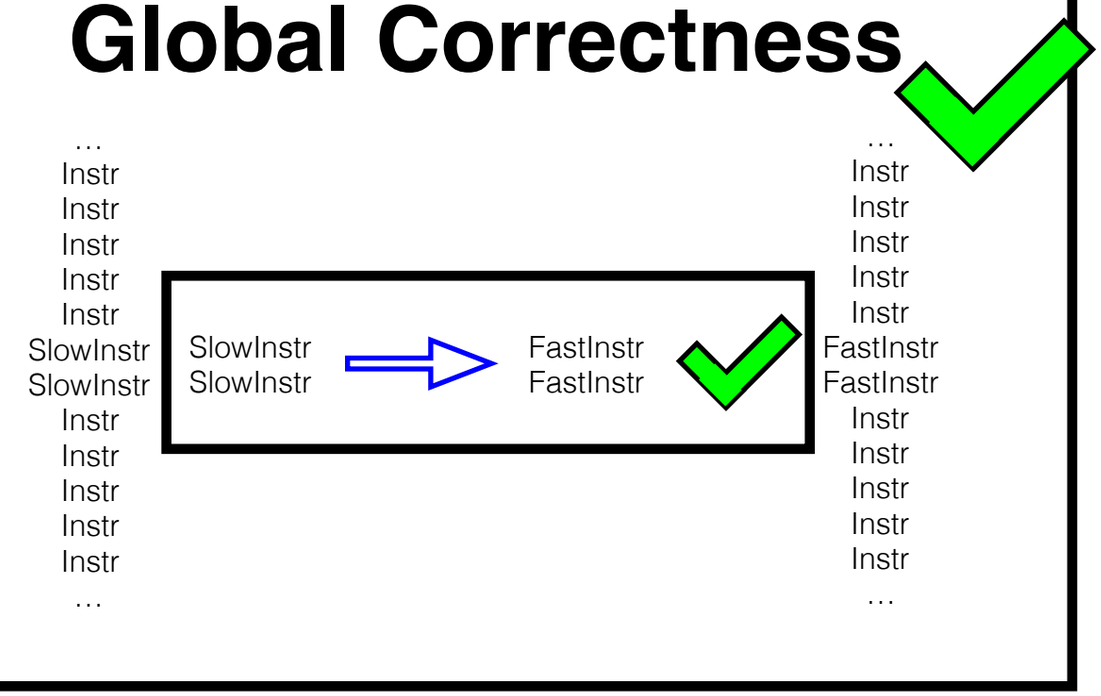
Acknowledgements

Questions?

Local Proofs

SlowInstr
SlowInstr
⇒
FastInstr
FastInstr
✓

Global Correctness



New Semantics

Pointers

⇩

Bit Vectors

?

Liveness

Evaluation

emullen@cs.washington.edu

Backup Slides