

Formal Reasoning about Capability Machines

Lau Skorstengaard

PhD Defence
November 14, 2019



Programming

High-level

Low-level

Programming

High-level



Low-level



Programming

High-level

```
main:  
  x := 5  
  ...
```



Low-level



Programming

High-level

```
main:  
  x := 5  
  ...
```



Low-level

```
1  
10  
001  
1010  
11101
```



Programming

High-level

```
main:  
  x := 5  
  ...
```



Low-level

```
0  
01  
101  
0010  
110010  
001001  
110010  
001010  
111001
```

```
1  
10  
001  
1010  
11101
```



Programming

High-level

```
main:  
  x := 5  
  ...
```



Low-level

```
0  
01  
101  
0010  
110010  
001001  
110010  
001010  
111001
```

1
10
001
1010
11101



Programming

High-level

```
main:  
  x := 5  
  ...
```



Low-level

```
0  
01  
101  
0010  
110010  
001001  
110010  
001010  
111001
```

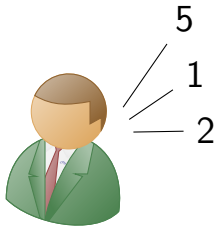
1
10
00
10
11101



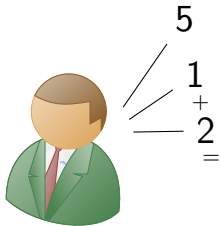
What is a computer?



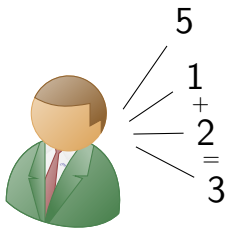
What is a computer?



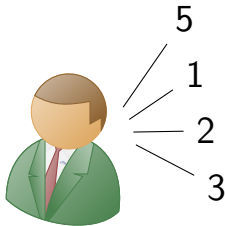
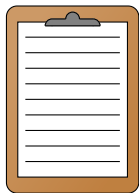
What is a computer?



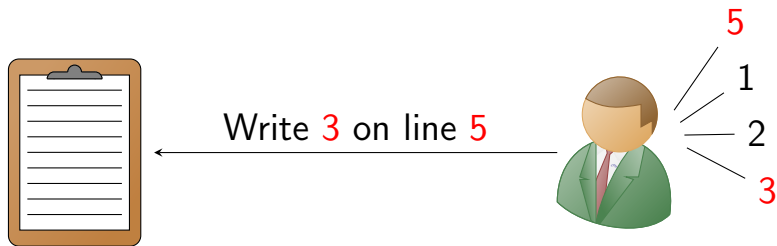
What is a computer?



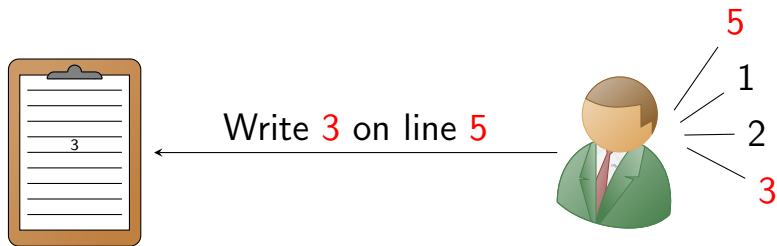
What is a computer?



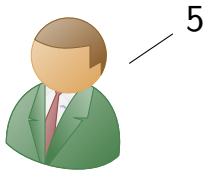
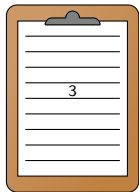
What is a computer?



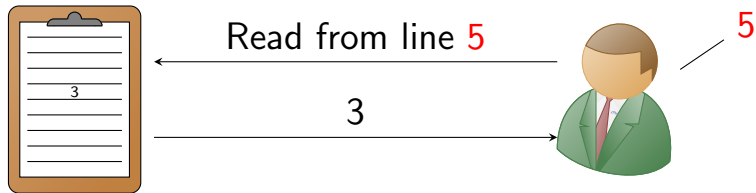
What is a computer?



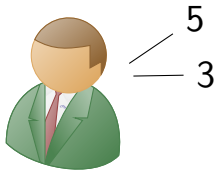
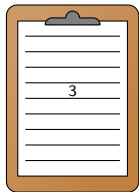
What is a computer?



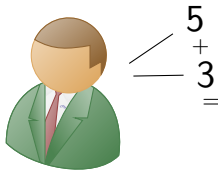
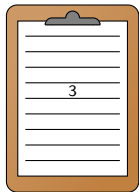
What is a computer?



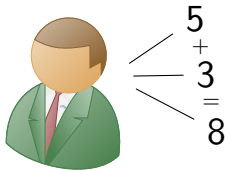
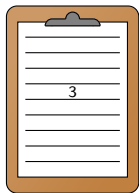
What is a computer?



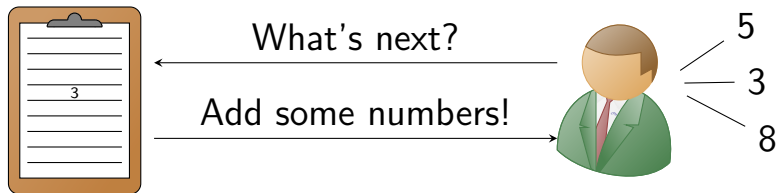
What is a computer?



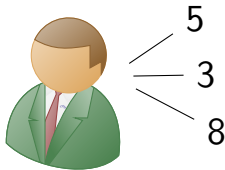
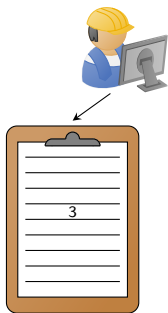
What is a computer?



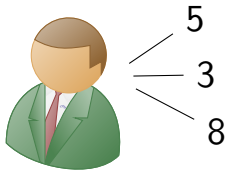
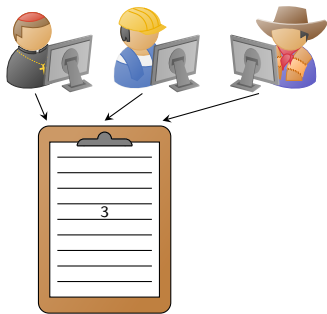
What is a computer?



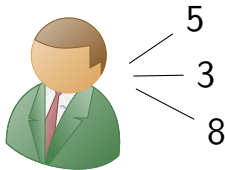
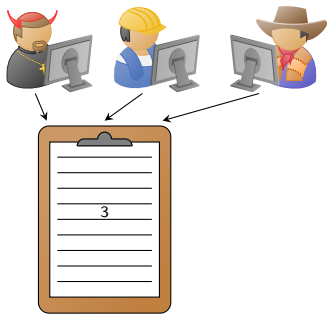
What is a computer?



What is a computer?



What is a computer?



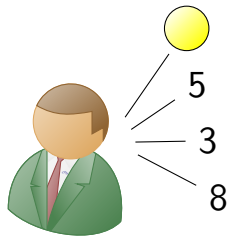
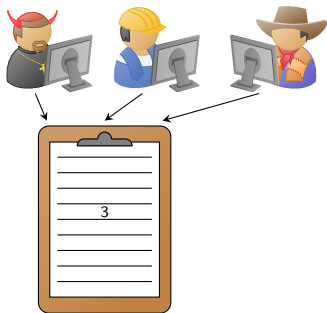
What is a capability?

What is a capability?

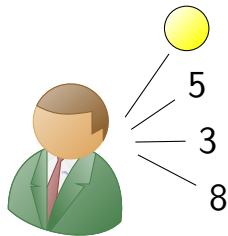
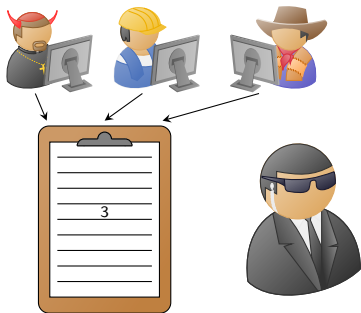
Generally, a capability is

*an unforgeable
token of authority*

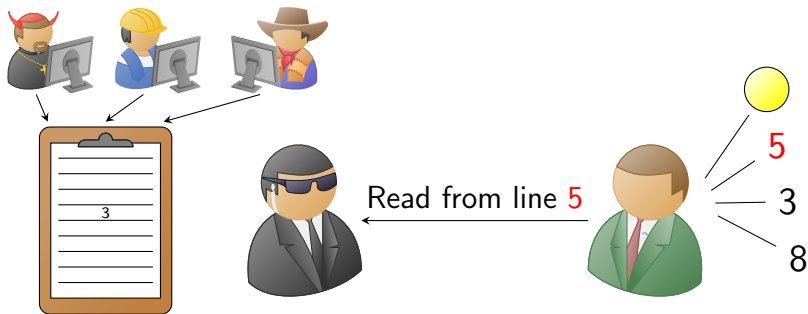
What is a capability machine?



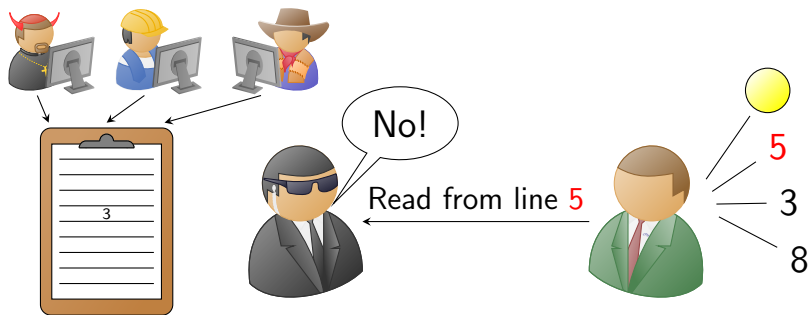
What is a capability machine?



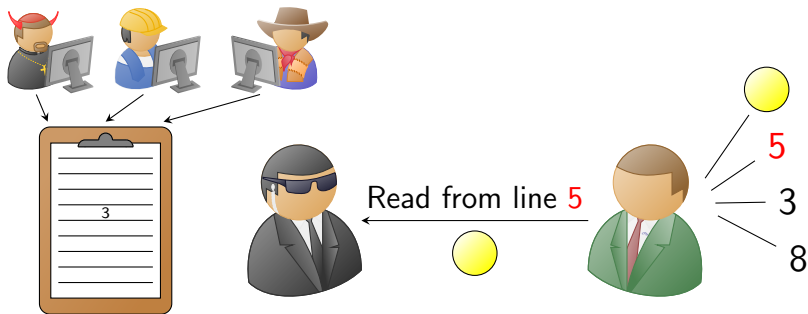
What is a capability machine?



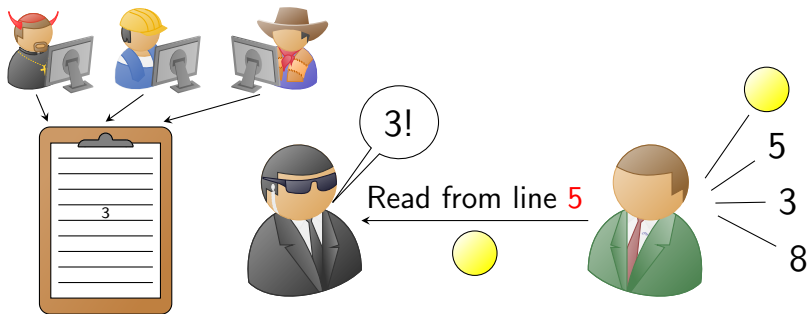
What is a capability machine?



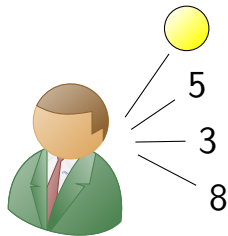
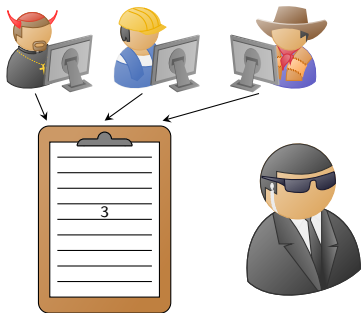
What is a capability machine?



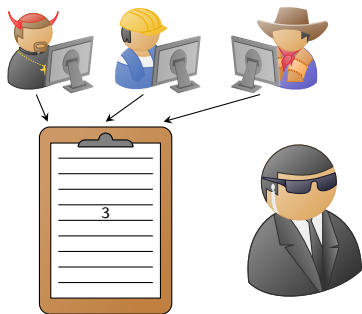
What is a capability machine?




What is a capability machine?



What is a capability machine?

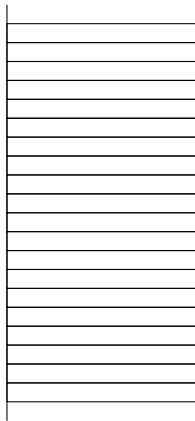


Registers

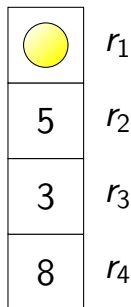
	r_1
5	r_2
3	r_3
8	r_4

What is a capability machine?

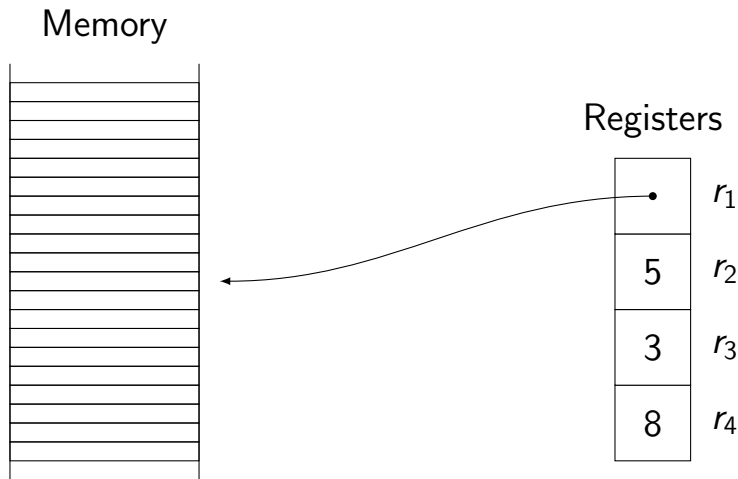
Memory



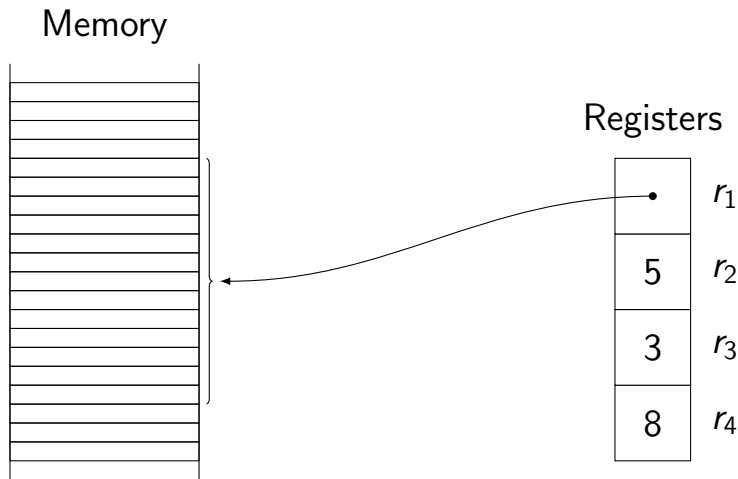
Registers



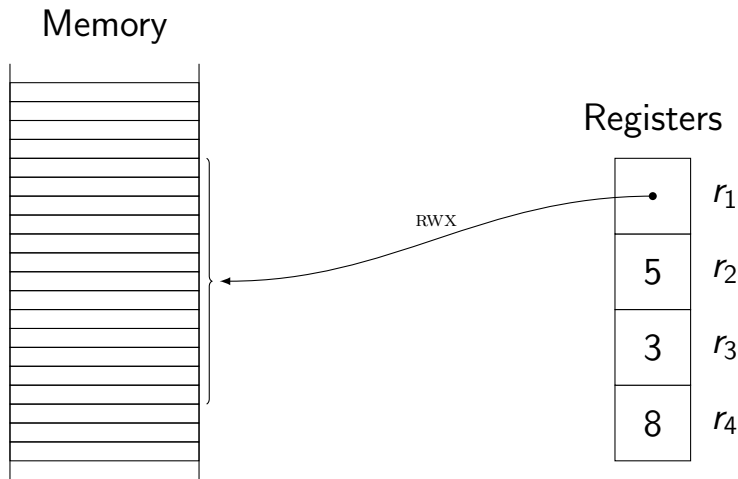
What is a capability machine?



What is a capability machine?



What is a capability machine?



Programming

High-level



Low-level



Program execution

```
void a()  
{  
    int x = 5;  
    b();  
    ...  
    b();  
    return;  
}
```

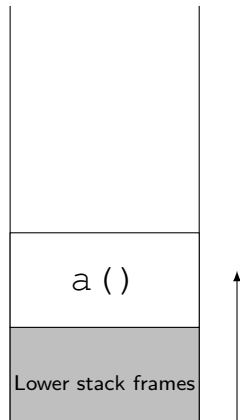
```
void b()  
{  
    ...  
    return;  
}
```


Program execution

```
void a()  
{  
    int x = 5;  
    b();  
    ...  
    b();  
    return;  
}
```

```
void b()  
{  
    ...  
    return;  
}
```

Call stack

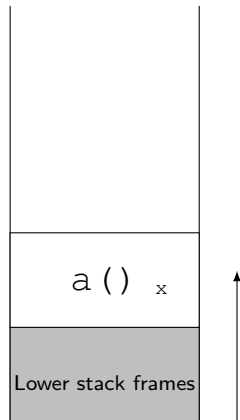


Program execution

```
void a()  
{  
→ int x = 5;  
  b();  
  ...  
  b();  
  return;  
}
```

```
void b()  
{  
  ...  
  return;  
}
```

Call stack

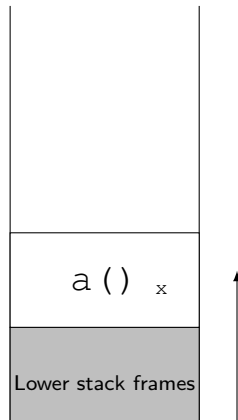


Program execution

```
void a()  
{  
    int x = 5;  
    → b();  
    ...  
    b();  
    return;  
}
```

```
void b()  
{  
    ...  
    return;  
}
```

Call stack

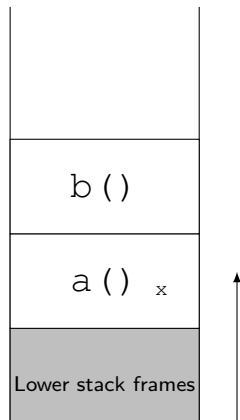


Program execution

```
void a ()  
{  
    int x = 5;  
    b();  
    ...  
    b();  
    return;  
}
```

```
→ void b ()  
{  
    ...  
    return;  
}
```

Call stack

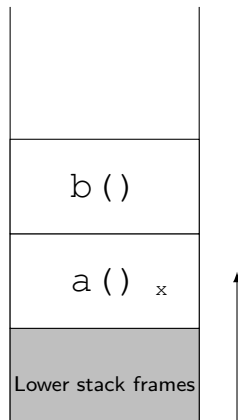


Program execution

```
void a()  
{  
    int x = 5;  
    b();  
    ...  
    b();  
    return;  
}
```

```
void b()  
{  
    ...  
→ return;  
}
```

Call stack

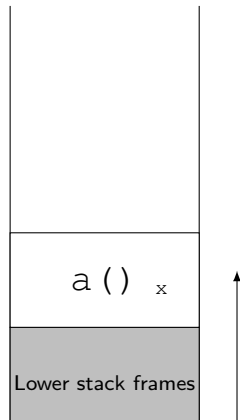


Program execution

```
void a()  
{  
    int x = 5;  
    b();  
→   ...  
    b();  
    return;  
}
```

```
void b()  
{  
    ...  
    return;  
}
```

Call stack

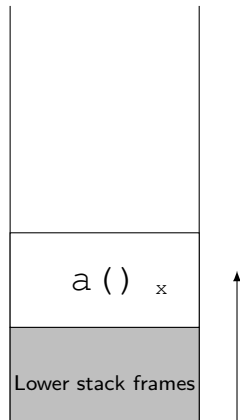


Program execution

```
void a()  
{  
    int x = 5;  
    b();  
    ...  
→ b();  
    return;  
}
```

```
void b()  
{  
    ...  
    return;  
}
```

Call stack

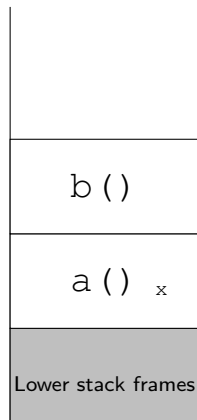


Program execution

```
void a ()  
{  
    int x = 5;  
    b();  
    ...  
    b();  
    return;  
}
```

```
→ void b ()  
{  
    ...  
    return;  
}
```

Call stack

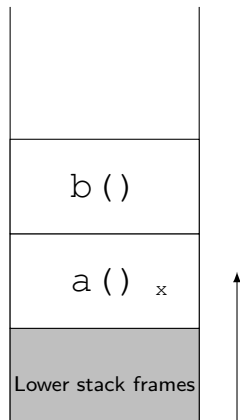


Program execution

```
void a()  
{  
    int x = 5;  
    b();  
    ...  
    b();  
    return;  
}
```

```
void b()  
{  
    ...  
→ return;  
}
```

Call stack

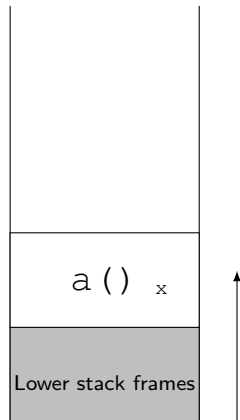


Program execution

```
void a ()  
{  
    int x = 5;  
    b();  
    ...  
    b();  
→ return;  
}
```

```
void b ()  
{  
    ...  
    return;  
}
```

Call stack

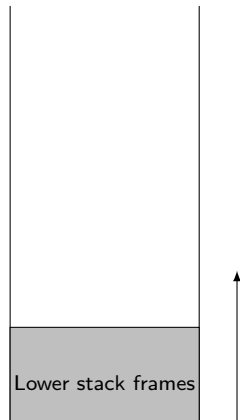


Program execution

```
void a()  
{  
    int x = 5;  
    b();  
    ...  
    b();  
    return;  
}
```

```
void b()  
{  
    ...  
    return;  
}
```

Call stack



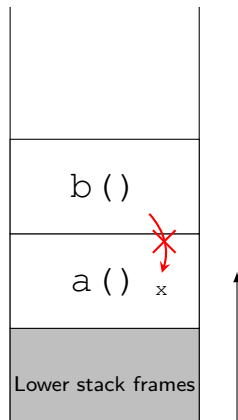
Local-state encapsulation

```
void a()  
{  
    int x = 5;  
    b();  
    ...  
    b();  
    return;  
}
```

```
void b()  
{  
    ...  
    return;  
}
```

} Function b
cannot access
variable x

Call stack

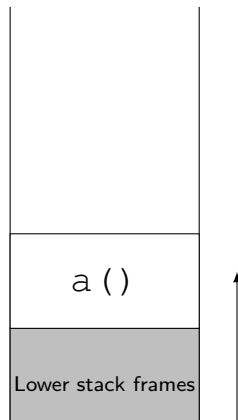


Not well-bracketed control-flow

```
void a()  
{  
    int x = 5;  
    b();  
    ...  
→ b();  
    return;  
}
```

```
void b()  
{  
    ...  
    return;  
}
```

Call stack

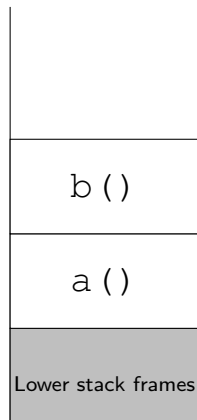


Not well-bracketed control-flow

```
void a ()  
{  
    int x = 5;  
    b();  
    ...  
    b();  
    return;  
}
```

```
→ void b ()  
{  
    ...  
    return;  
}
```

Call stack

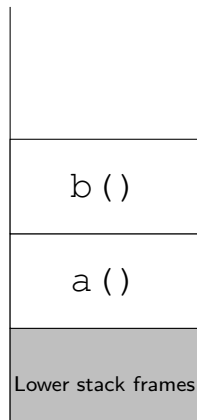


Not well-bracketed control-flow

```
void a ()  
{  
    int x = 5;  
    b();  
    ...  
    b();  
    return;  
}
```

```
void b ()  
{  
    ...  
→ return;  
}
```

Call stack

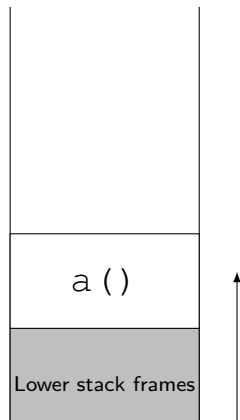


Not well-bracketed control-flow

```
void a ()  
{  
    int x = 5;  
    b();  
→   ...  
    b();  
    return;  
}
```

```
void b ()  
{  
    ...  
    return;  
}
```

Call stack




Not well-bracketed control-flow

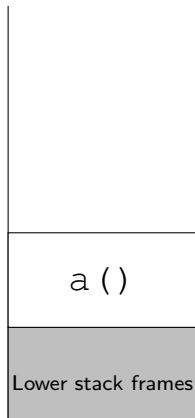
```
void a ()  
{  
    int x = 5;  
    b();  
→   ...  
    b();  
    return;  
}
```

```
void b ()  
{  
    ...  
    return;  
}
```

Should have
returned here!



Call stack



Programming

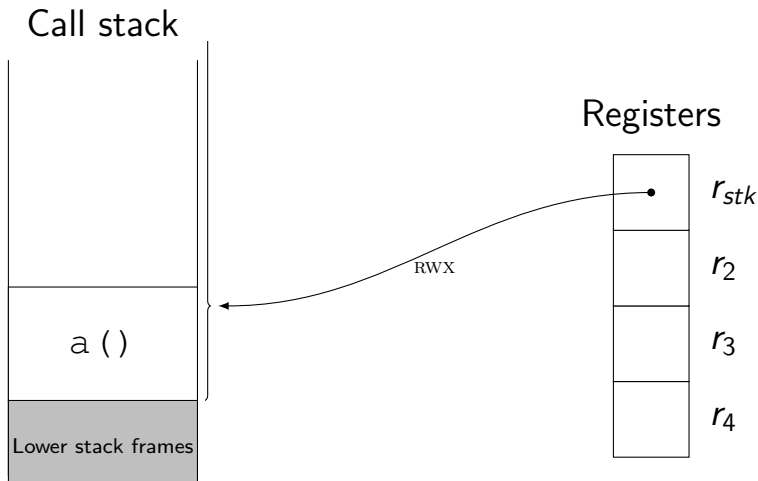
High-level



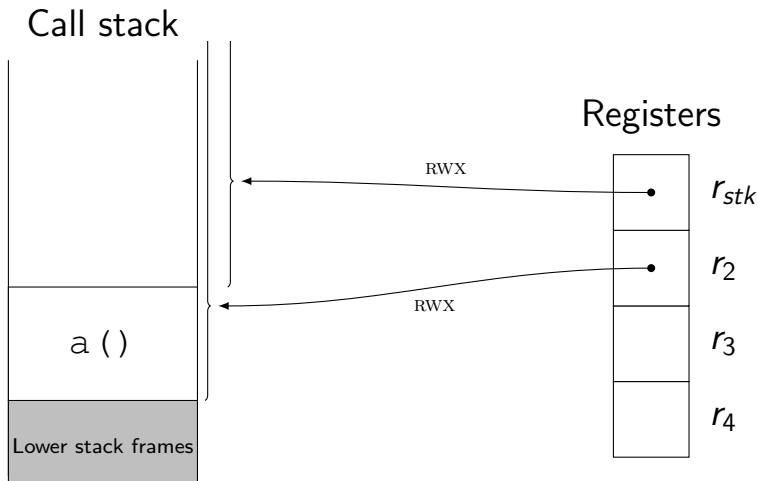
Low-level



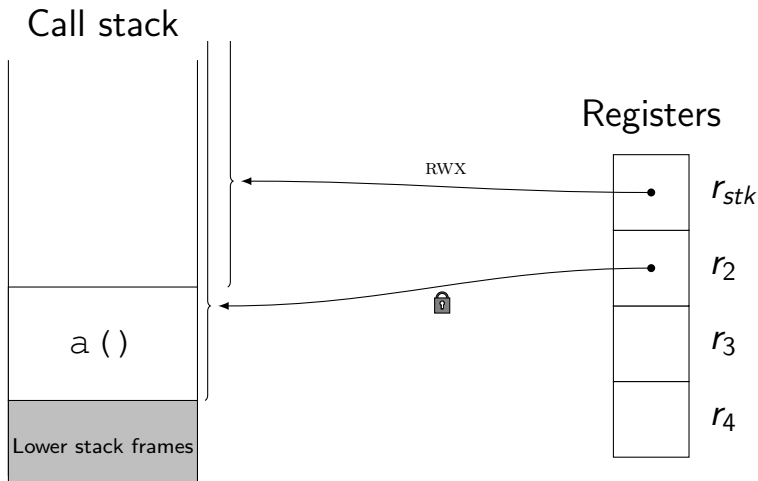
A naïve capability calling convention



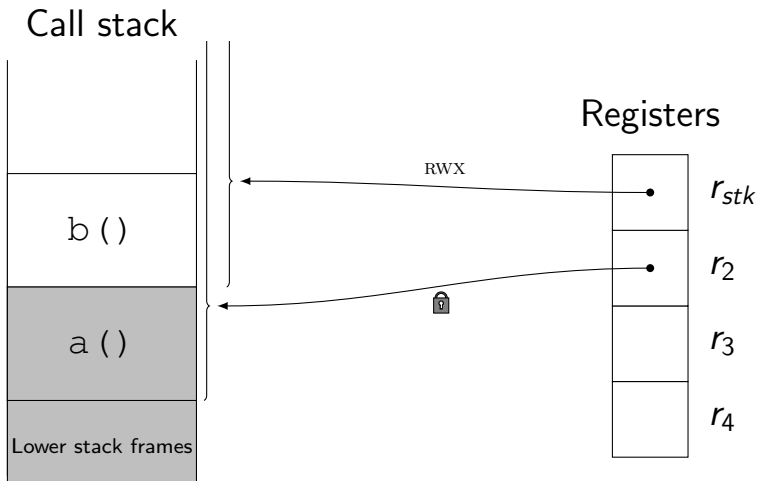
A naïve capability calling convention



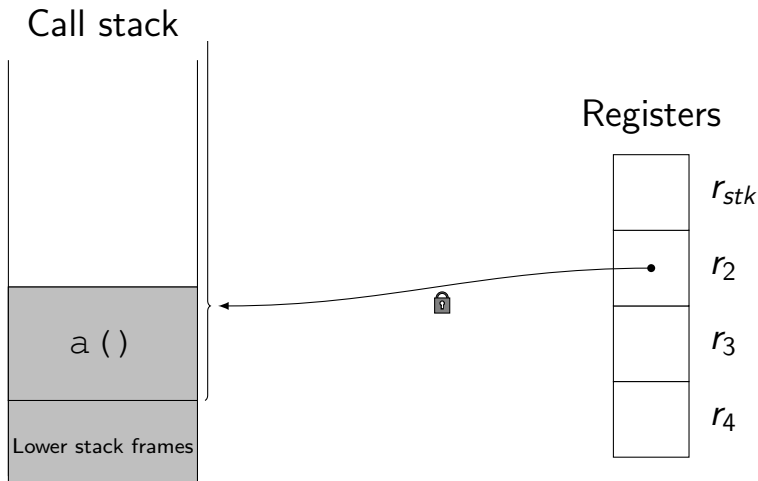
A naïve capability calling convention



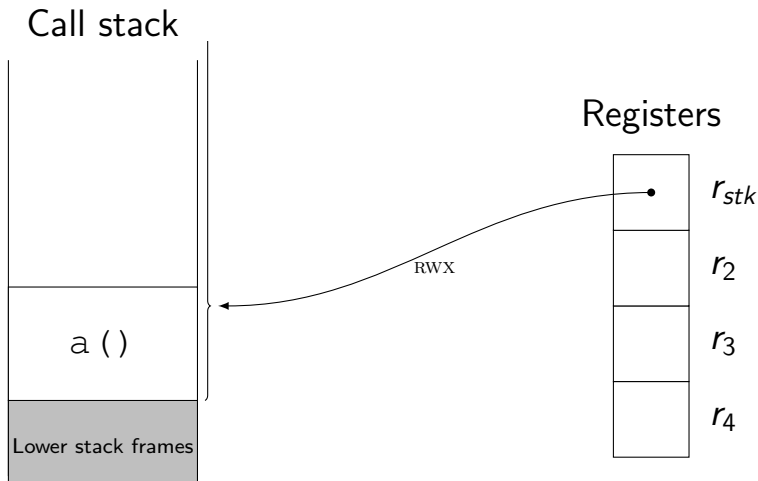
A naïve capability calling convention



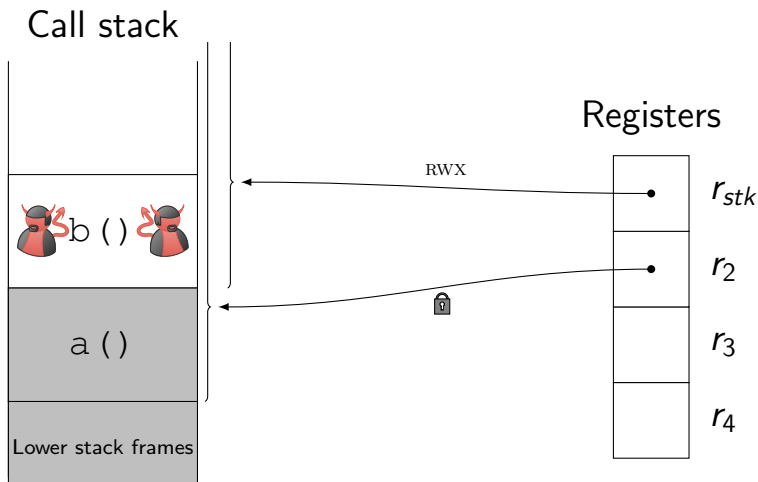
A naïve capability calling convention



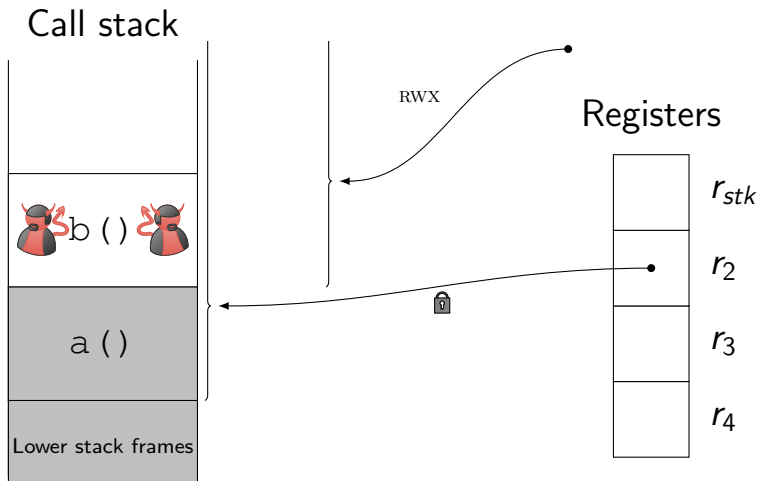
A naïve capability calling convention



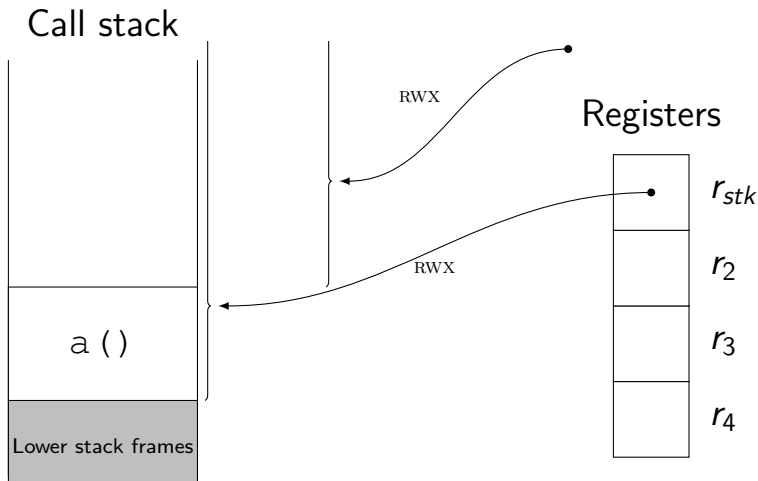
Breaking local-state encapsulation



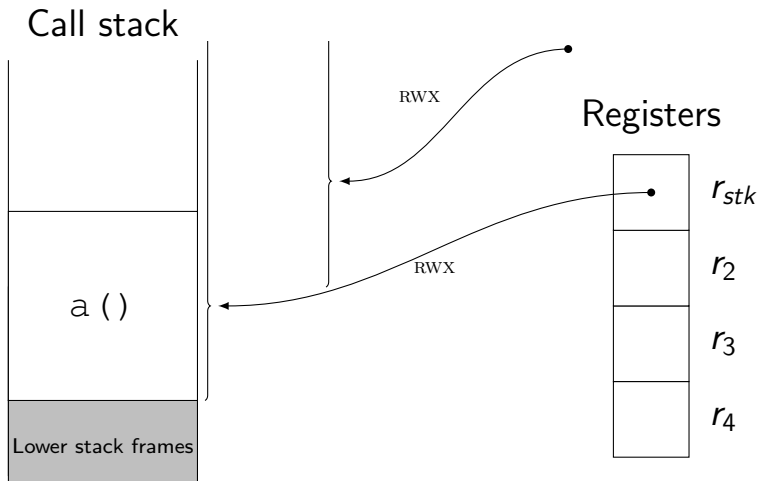
Breaking local-state encapsulation



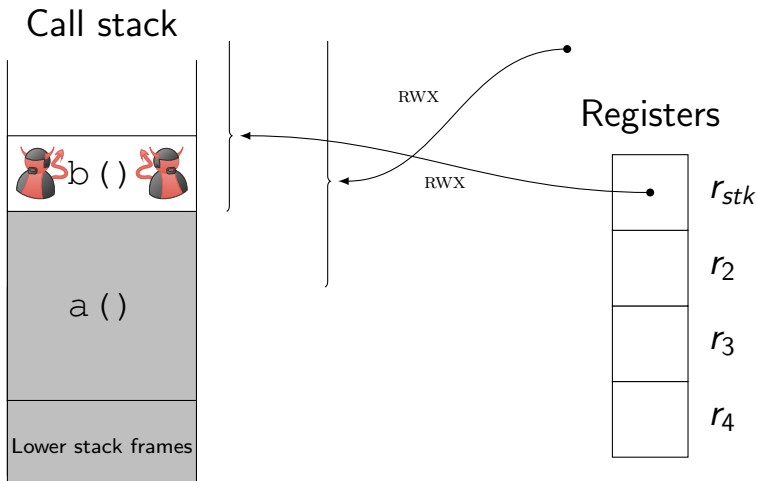
Breaking local-state encapsulation



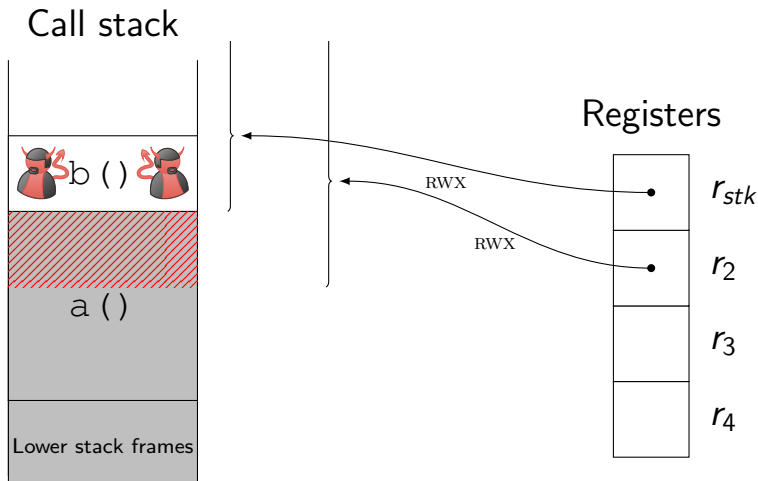
Breaking local-state encapsulation



Breaking local-state encapsulation

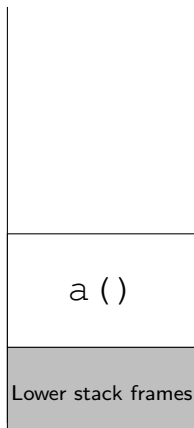


Breaking local-state encapsulation

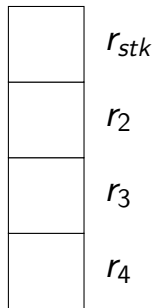


Breaking well-bracketed control flow

Call stack

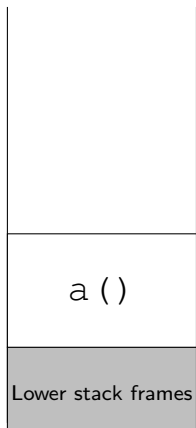


Registers

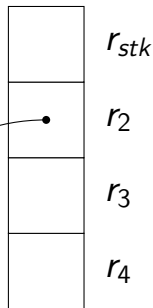


Breaking well-bracketed control flow

Call stack



Registers

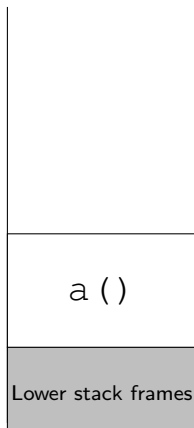


RX

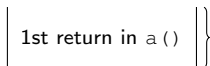
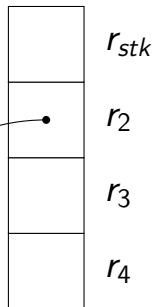
1st return in a ()

Breaking well-bracketed control flow

Call stack

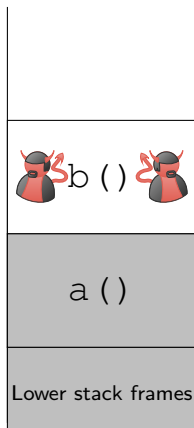


Registers

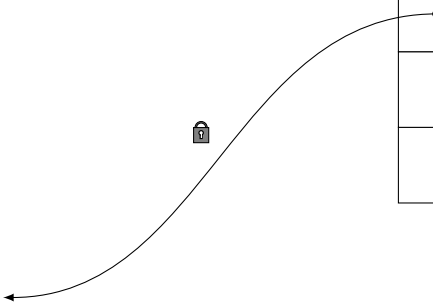
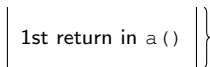
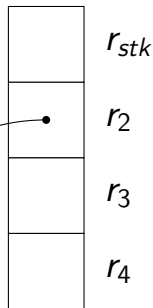


Breaking well-bracketed control flow

Call stack

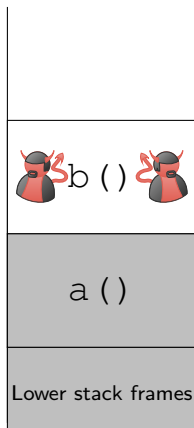


Registers

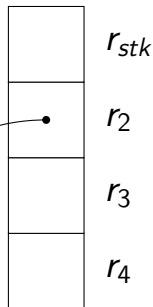


Breaking well-bracketed control flow

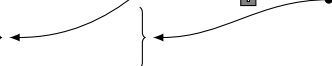
Call stack



Registers

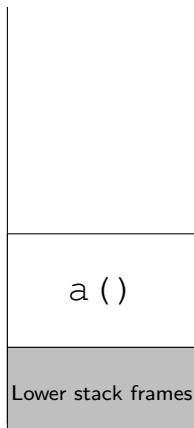


1st return in `a ()`



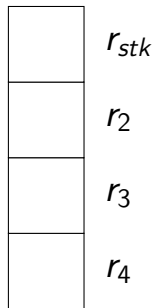
Breaking well-bracketed control flow

Call stack



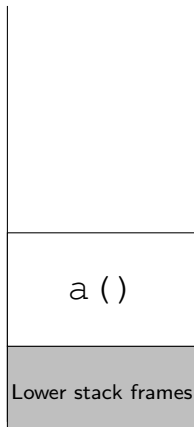
1st return in a ()

Registers



Breaking well-bracketed control flow

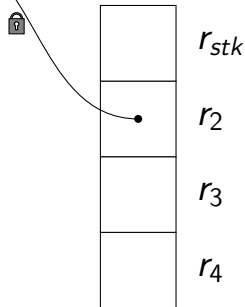
Call stack



1st return in `a ()`

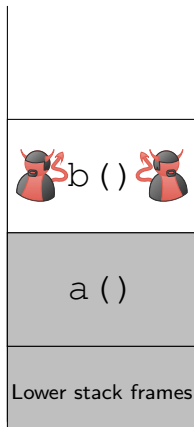
2nd return in `a ()`

Registers



Breaking well-bracketed control flow

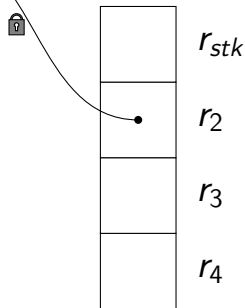
Call stack



1st return in `a()`

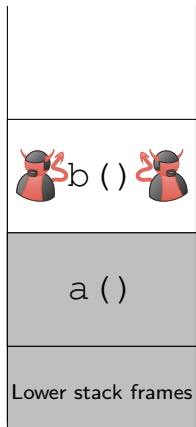
2nd return in `a()`

Registers



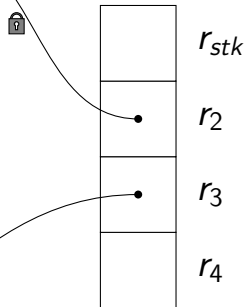
Breaking well-bracketed control flow

Call stack



2nd return in `a()`

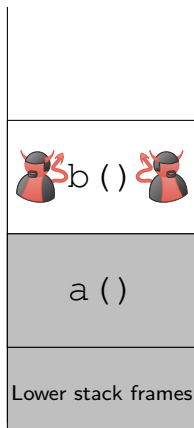
Registers



1st return in `a()`

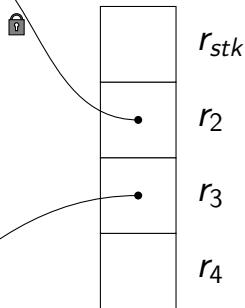
Breaking well-bracketed control flow

Call stack



2nd return in `a()`

Registers



1st return in `a()`

The thesis

The thesis

Well-bracketed control flow and *local-state encapsulation* can be provably enforced on capability machines.

Publication overview

1. *Reasoning About a Machine with Local Capabilities - Provably Safe Stack and Return Pointer Management*
2. *STKTOKENS: Enforcing Well-bracketed Control Flow and Stack Encapsulation Using Linear Capabilities*

Publication overview

1. *Reasoning About a Machine with Local Capabilities - Provably Safe Stack and Return Pointer Management*
 - ▶ Calling convention
2. *STKTOKENS: Enforcing Well-bracketed Control Flow and Stack Encapsulation Using Linear Capabilities*
 - ▶ Calling convention

Publication overview

1. *Reasoning About a Machine with **Local Capabilities** - Provably Safe Stack and Return Pointer Management*
 - ▶ Calling convention
2. *STKTOKENS: Enforcing Well-bracketed Control Flow and Stack Encapsulation Using Linear Capabilities*
 - ▶ Calling convention

Publication overview

1. *Reasoning About a Machine with Local Capabilities - Provably Safe Stack and Return Pointer Management*
 - ▶ Calling convention
2. *STKTOKENS: Enforcing Well-bracketed Control Flow and Stack Encapsulation Using Linear Capabilities*
 - ▶ Calling convention

Publication overview

1. *Reasoning About a Machine with Local Capabilities - Provably Safe Stack and Return Pointer Management*
 - ▶ Calling convention
 - ▶ Correctness proofs of examples
2. *STKTOKENS: Enforcing Well-bracketed Control Flow and Stack Encapsulation Using Linear Capabilities*
 - ▶ Calling convention

Publication overview

1. *Reasoning About a Machine with Local Capabilities - Provably Safe Stack and Return Pointer Management*
 - ▶ Calling convention
 - ▶ Correctness proofs of examples
2. *STKTOKENS: Enforcing Well-bracketed Control Flow and Stack Encapsulation Using Linear Capabilities*
 - ▶ Calling convention
 - ▶ Correctness proof for all programs

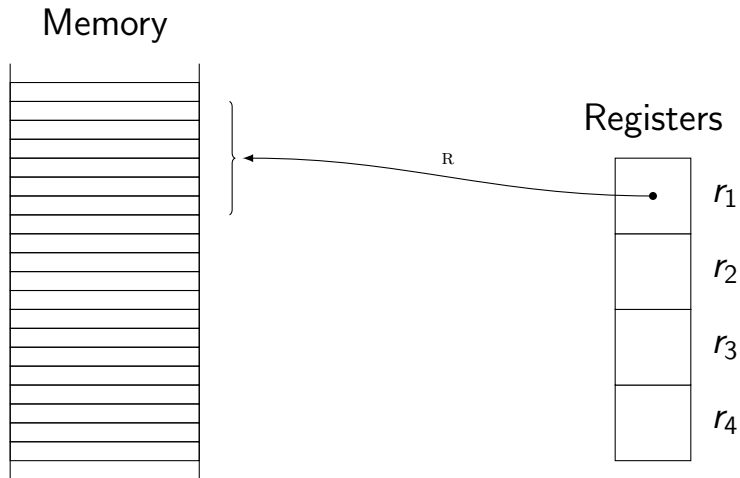
Local capabilities

Register-only capabilities

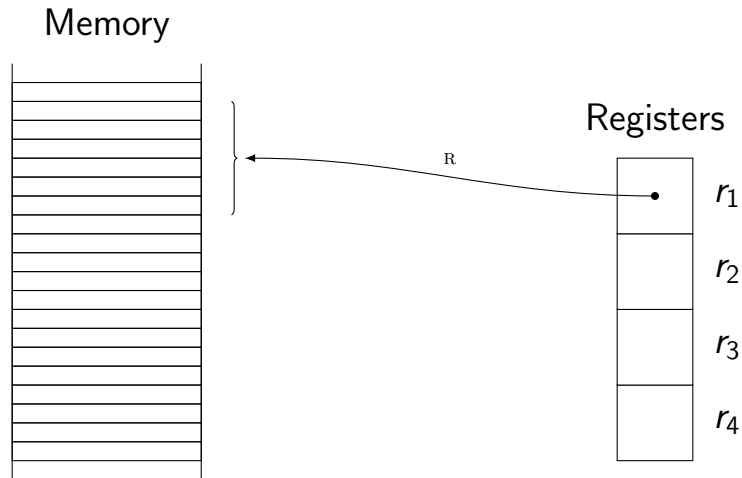
Local capabilities

Register-only capabilities, essentially

Local capabilities by example

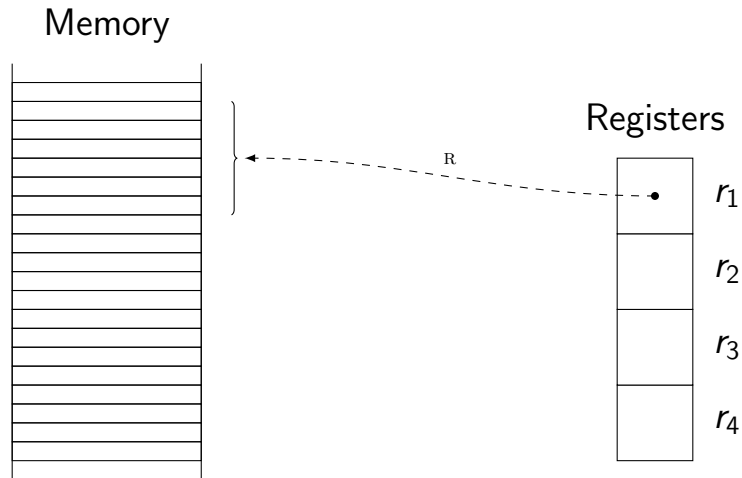


Local capabilities by example



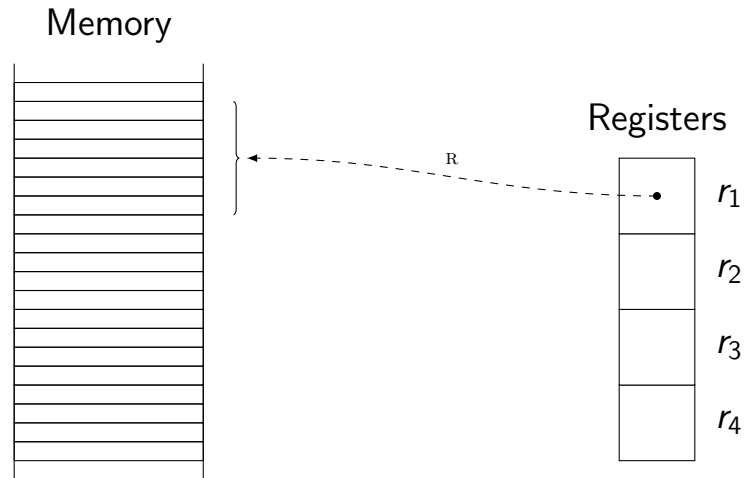
Capabilities can be turned into local capabilities

Local capabilities by example



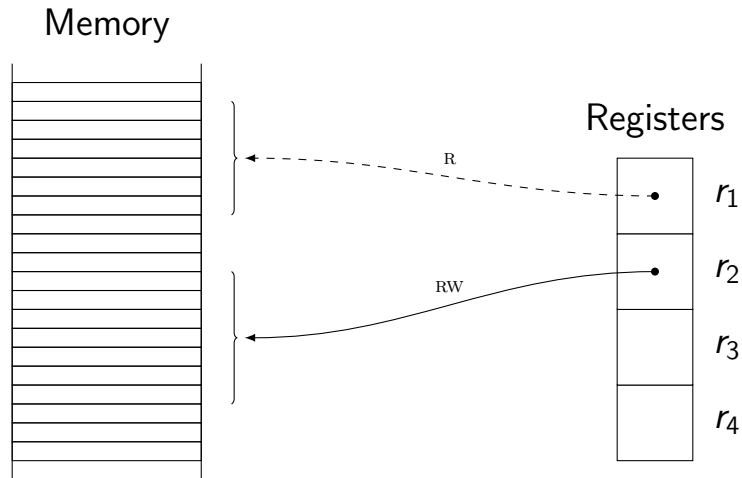
Capabilities can be turned into local capabilities

Local capabilities by example



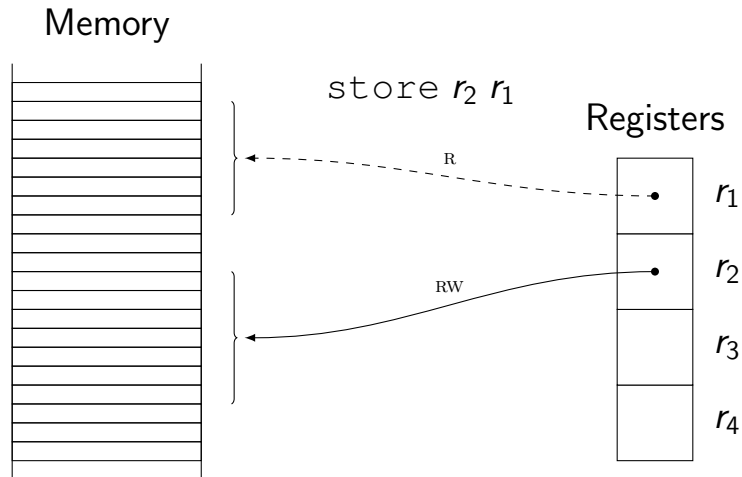
Cannot be stored to memory

Local capabilities by example

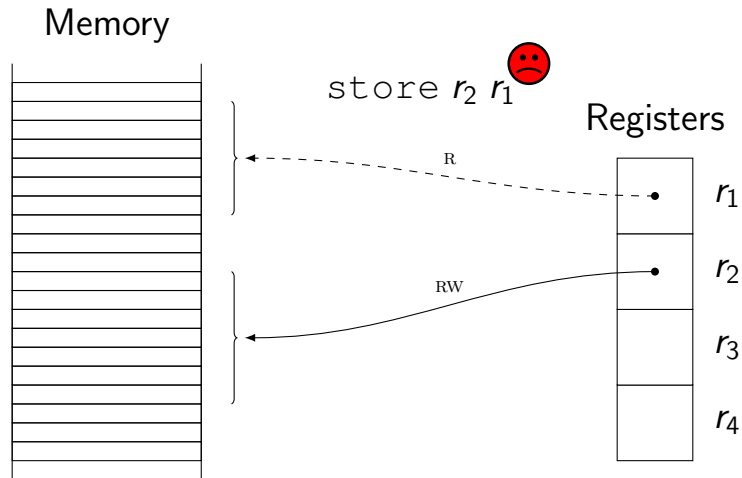


Cannot be stored to memory

Local capabilities by example

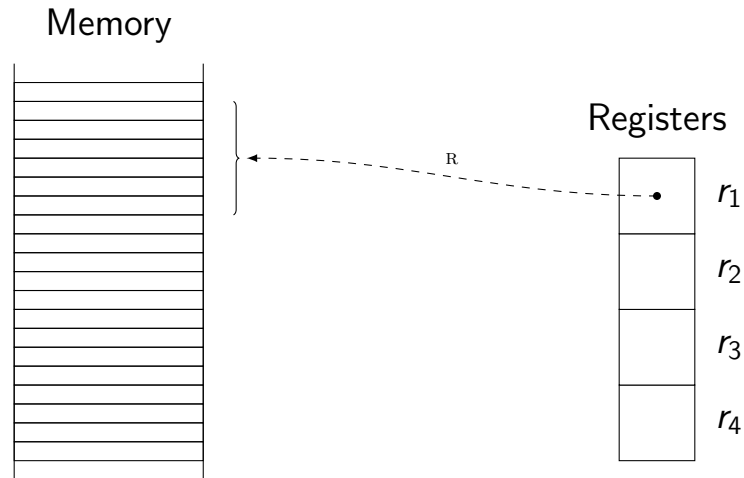


Local capabilities by example



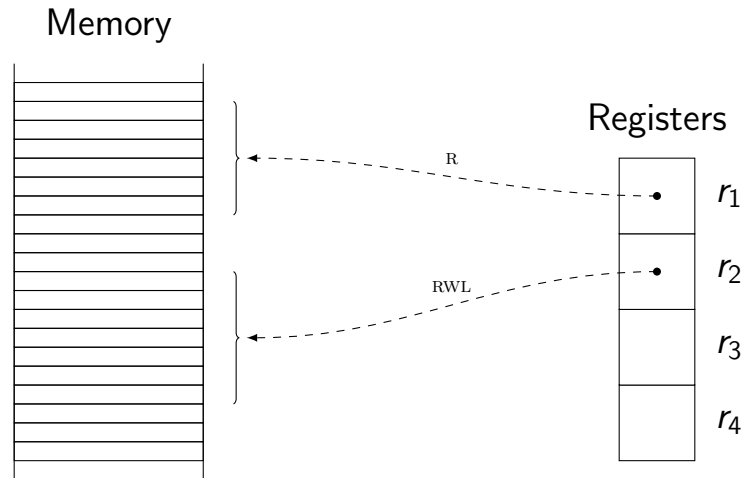
Cannot be stored to memory

Local capabilities by example



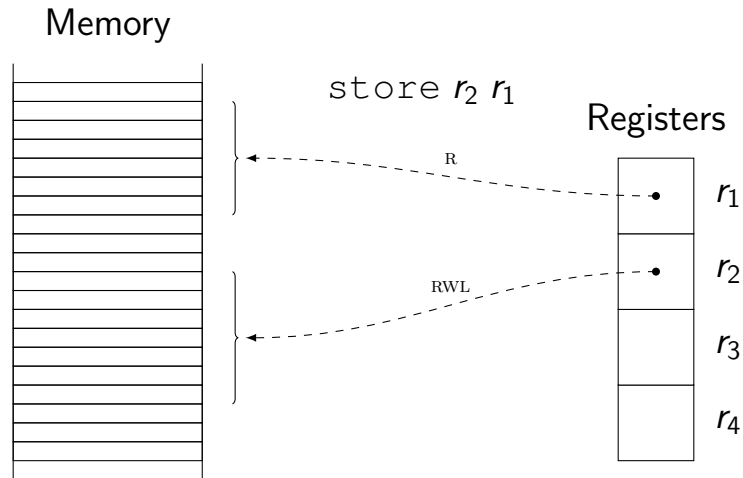
New write-local permission

Local capabilities by example



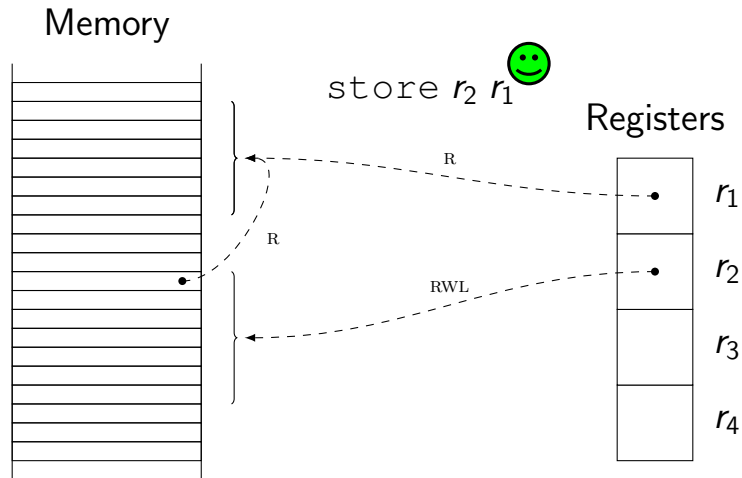
New write-local permission

Local capabilities by example



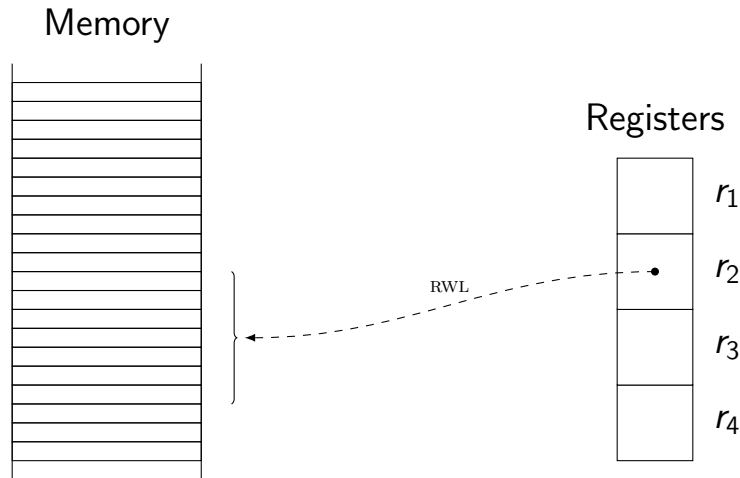
New write-local permission

Local capabilities by example



New write-local permission

Local capabilities by example



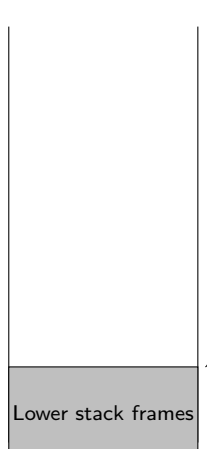
Write-local capability must be local

The calling convention

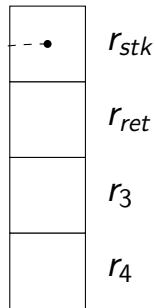
- ▶ Write-local capability for the stack
- ▶ Stack allocated restoration record
- ▶ Return capabilities are “locked” stack capabilities for the restoration record
- ▶ Revoke old stack capabilities and return capabilities by clearing unused stack and registers
- ▶ ... and a few other things

The calling convention by example

Call stack in memory



Registers

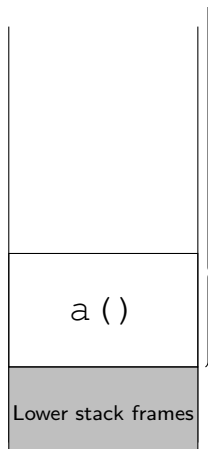


RWLX

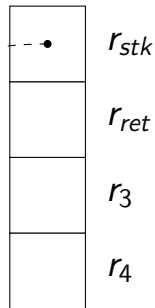


The calling convention by example

Call stack in memory



Registers

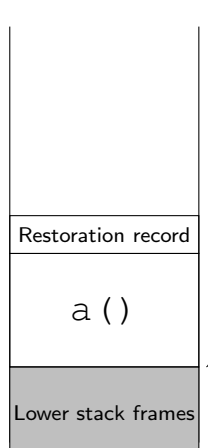


RWLX

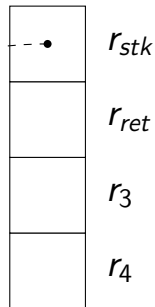


The calling convention by example

Call stack in memory



Registers

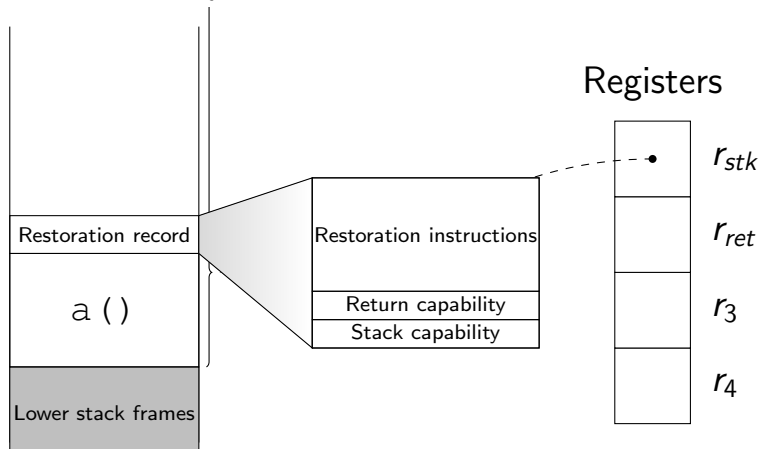


RWLX



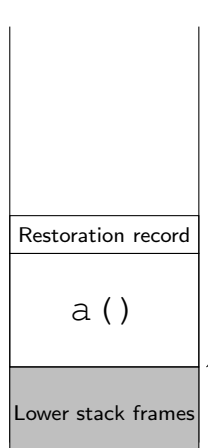
The calling convention by example

Call stack in memory

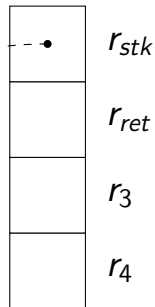


The calling convention by example

Call stack in memory



Registers

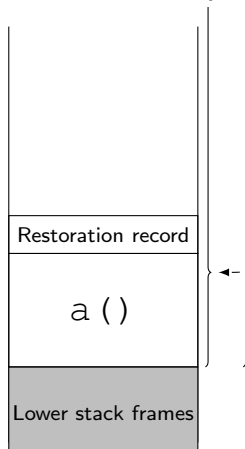


RWLX

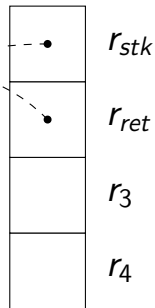


The calling convention by example

Call stack in memory

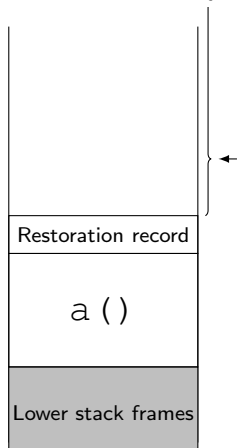


Registers

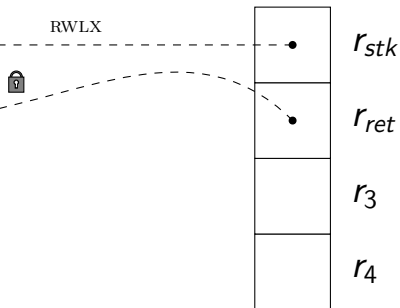


The calling convention by example

Call stack in memory

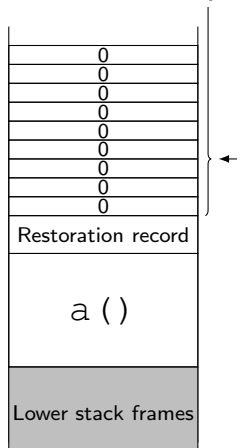


Registers

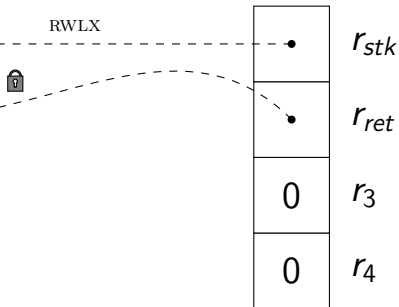


The calling convention by example

Call stack in memory

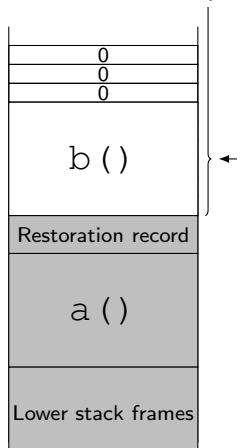


Registers

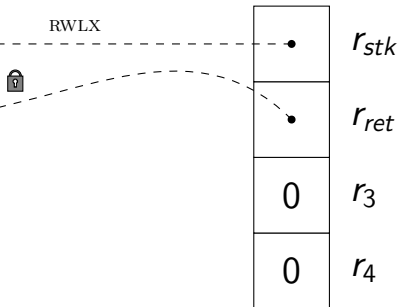


The calling convention by example

Call stack in memory

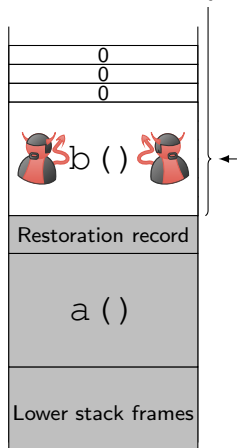


Registers

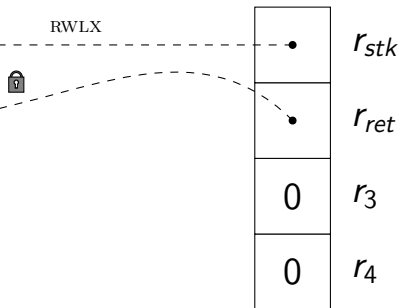


The calling convention by example

Call stack in memory

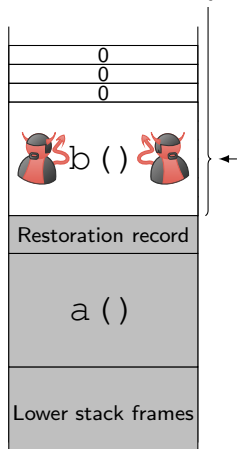


Registers

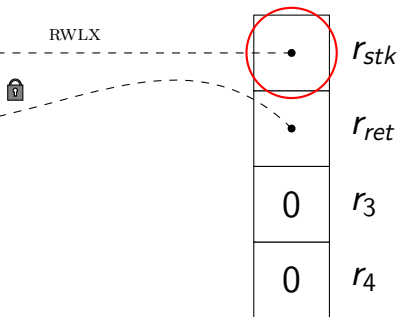


The calling convention by example

Call stack in memory

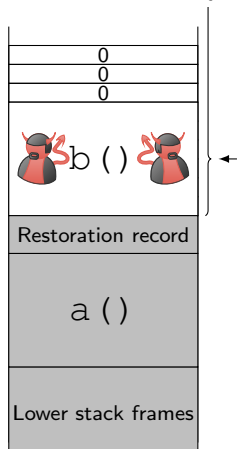


Registers

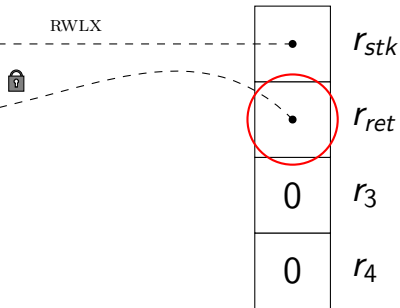


The calling convention by example

Call stack in memory



Registers



Formal result

Lemma (Example correctness)

No matter what program context the program

```
fun _ =>  
  let x = ref 0 in  
    fun callback =>  
      x := 0;  
      callback();  
      x := 1;  
      callback();  
      assert (x == 1)
```

is used in, the assertion never fails.

Summary of the calling convention

It's great!

Summary of the calling convention

It's great!

...except for all the stack clearing.

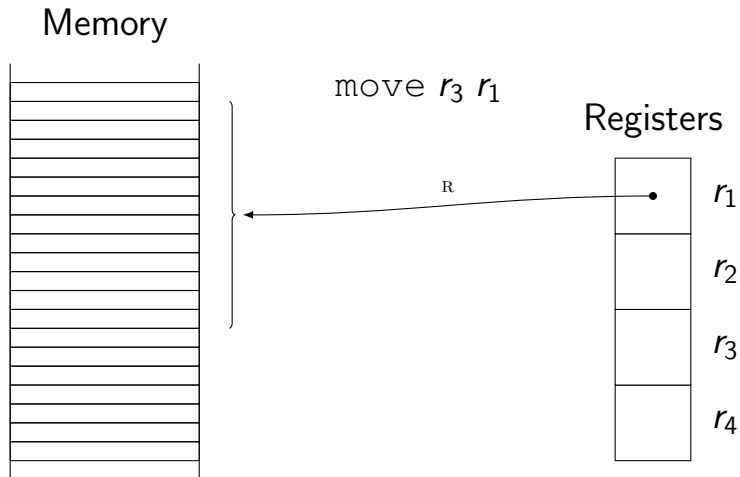
Publication overview

1. *Reasoning About a Machine with Local Capabilities - Provably Safe Stack and Return Pointer Management*
 - ▶ Calling convention
 - ▶ Correctness proofs of examples
2. *STKTOKENS: Enforcing Well-bracketed Control Flow and Stack Encapsulation Using Linear Capabilities*
 - ▶ Calling convention
 - ▶ Correctness proof for all programs

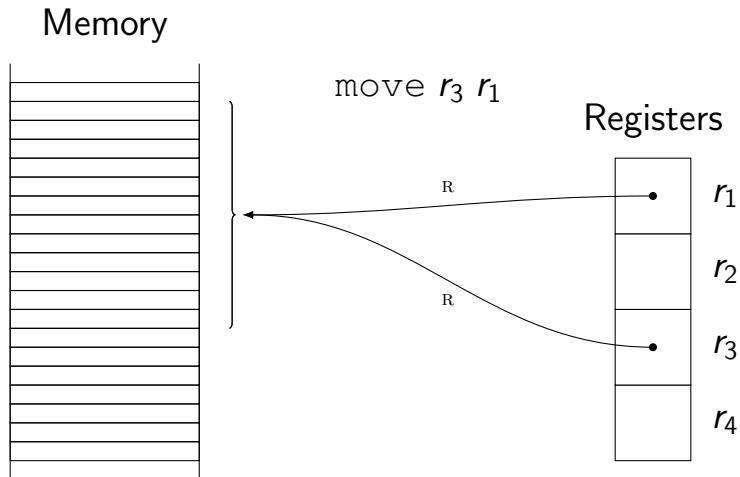
Linear capabilities

Capabilities that cannot be duplicated

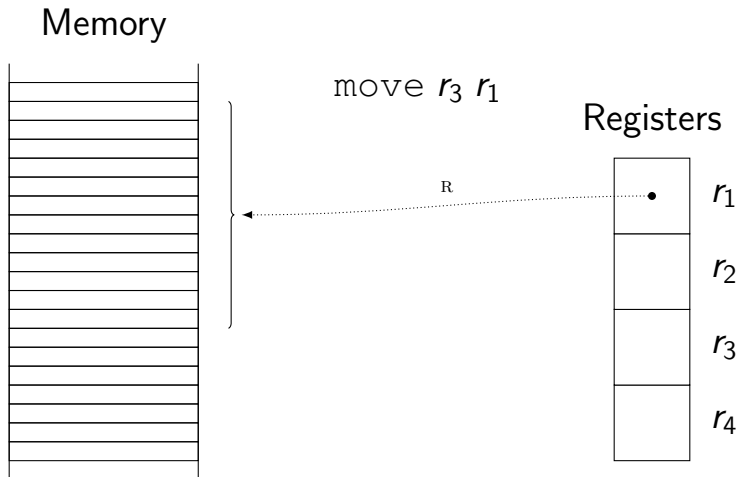
Linear capabilities by example



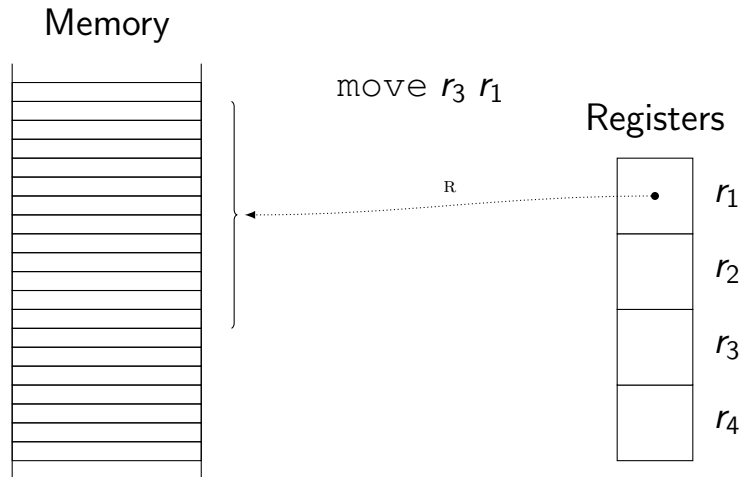
Linear capabilities by example



Linear capabilities by example

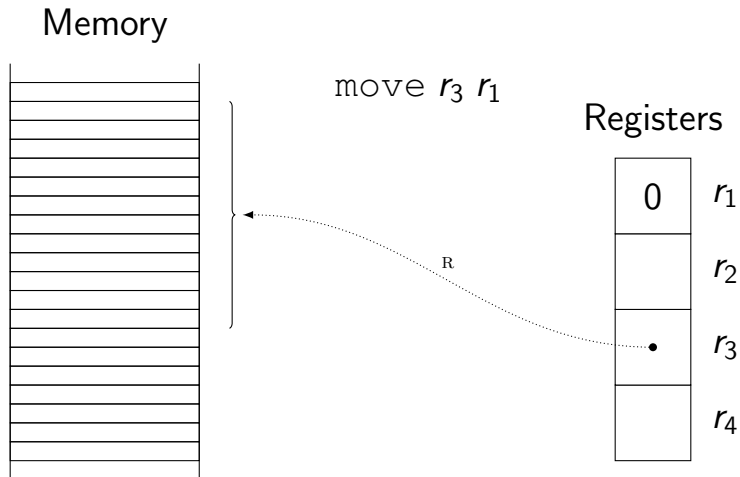


Linear capabilities by example



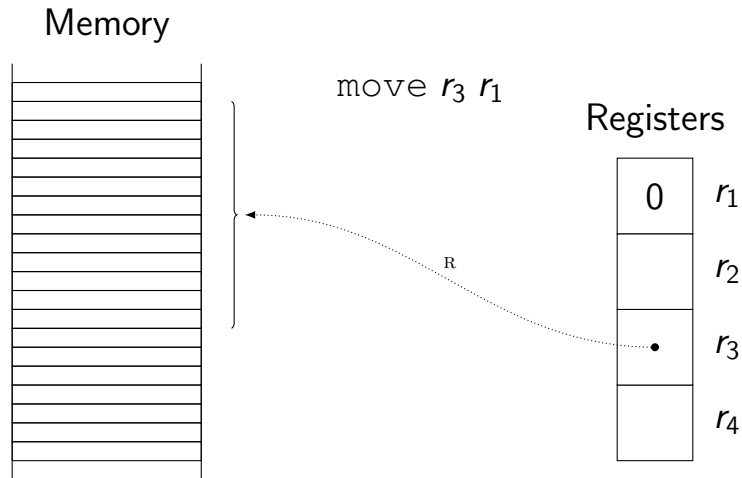
Moving a linear capability clears the source

Linear capabilities by example



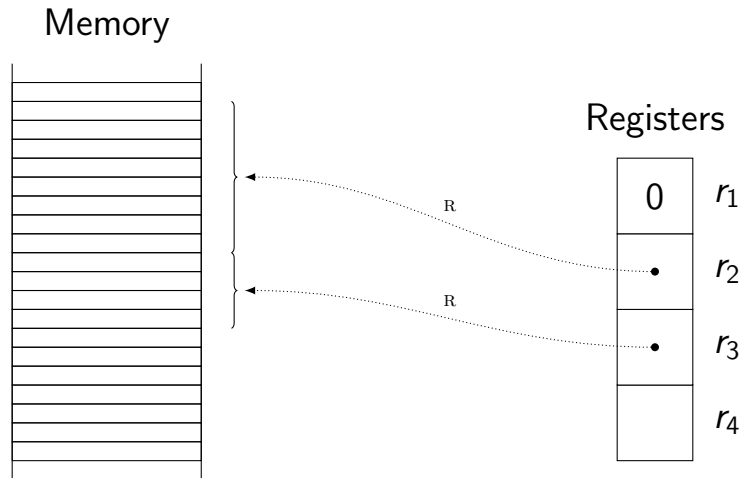
Moving a linear capability clears the source

Linear capabilities by example



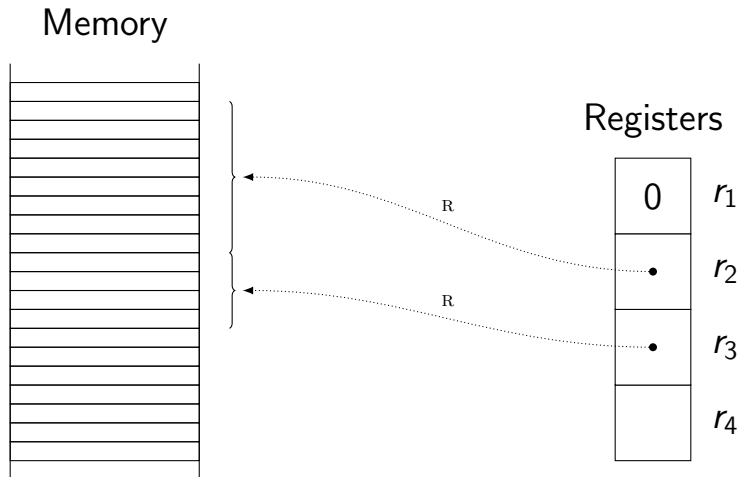
Linear capabilities can be split

Linear capabilities by example



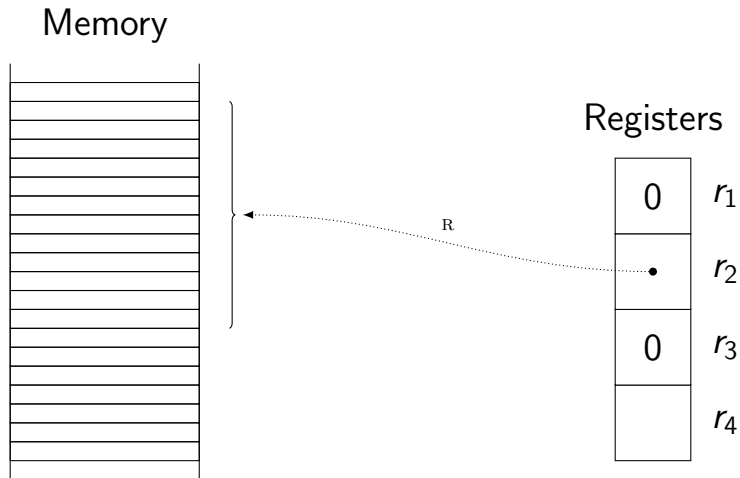
Linear capabilities can be split

Linear capabilities by example



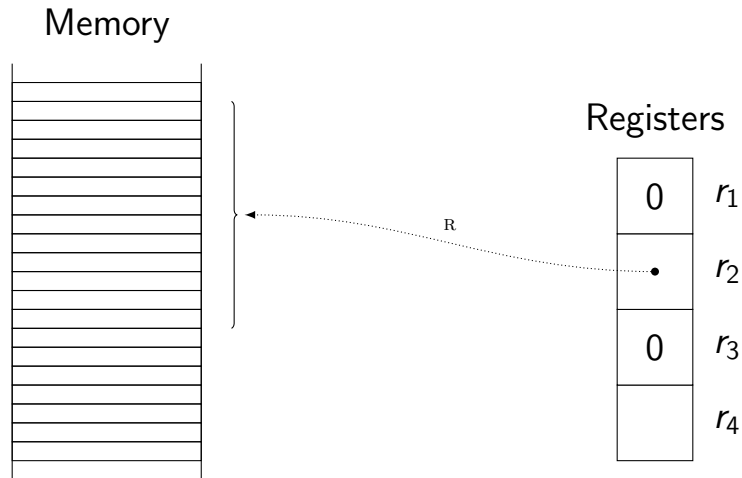
Linear capabilities can be split and spliced

Linear capabilities by example



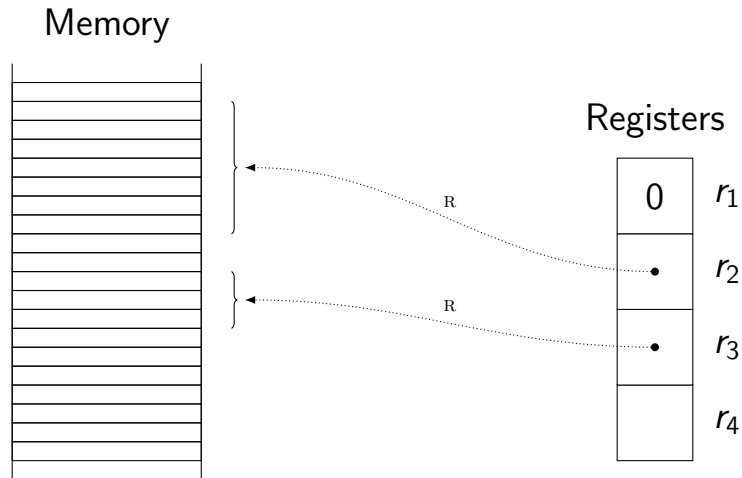
Linear capabilities can be split and spliced

Linear capabilities by example



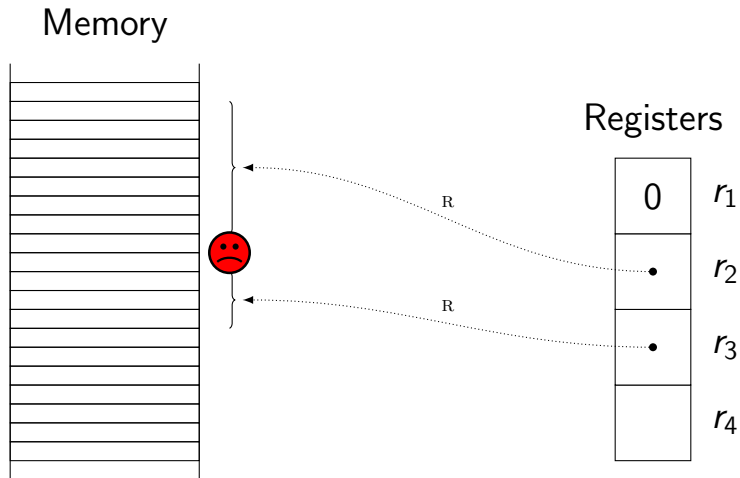
Splice fails for non-adjacent capabilities

Linear capabilities by example



Splice fails for non-adjacent capabilities

Linear capabilities by example



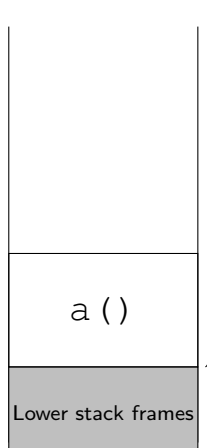
Splice fails for non-adjacent capabilities

STK TOKENS

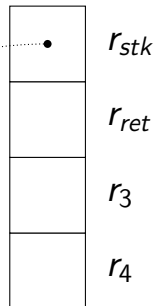
- ▶ Use a *unique* linear capability for the stack
- ▶ Require that callees return their stack capability
- ▶ ... and a few other things

STKTOKENS by example

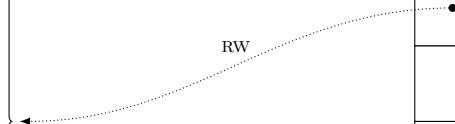
Call stack in memory



Registers

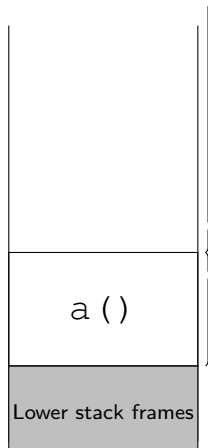


RW

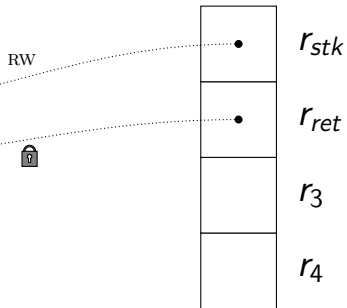


STKTOKENS by example

Call stack in memory

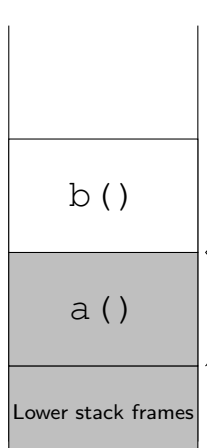


Registers

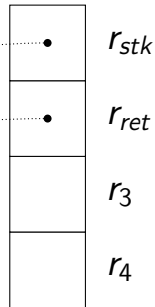


STKTOKENS by example

Call stack in memory



Registers

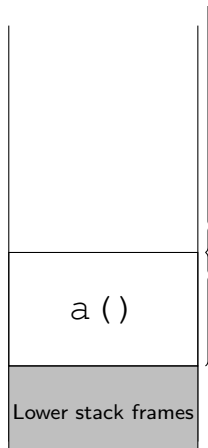


RW

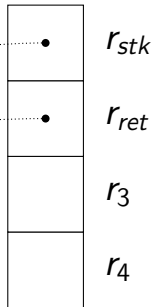


STKTOKENS by example

Call stack in memory



Registers

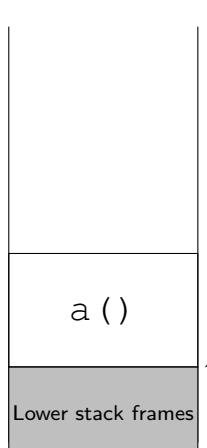


RW

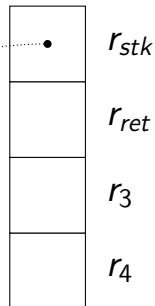
RW

STKTOKENS by example

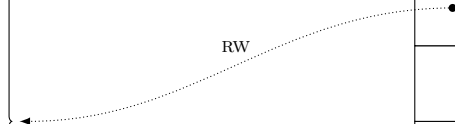
Call stack in memory



Registers

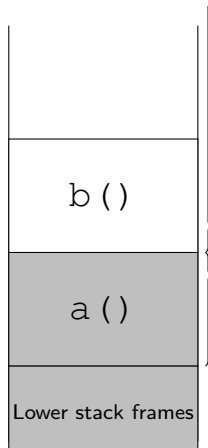


RW

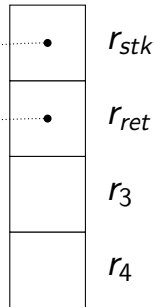


STKTOKENS by example

Call stack in memory



Registers

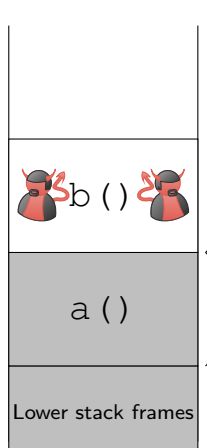


RW

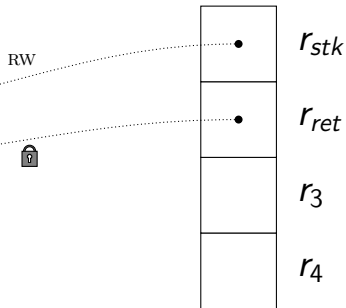


STKTOKENS by example

Call stack in memory

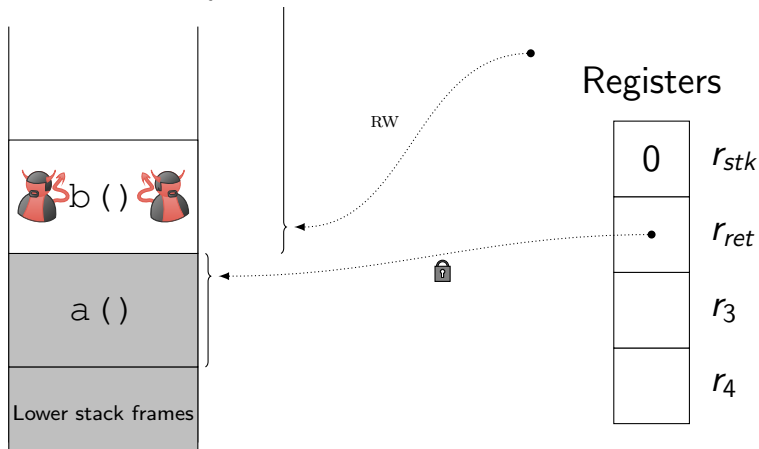


Registers



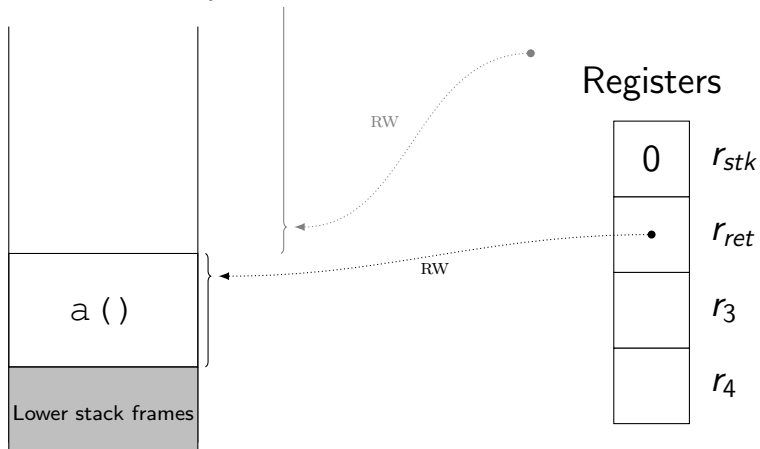
STKTOKENS by example

Call stack in memory



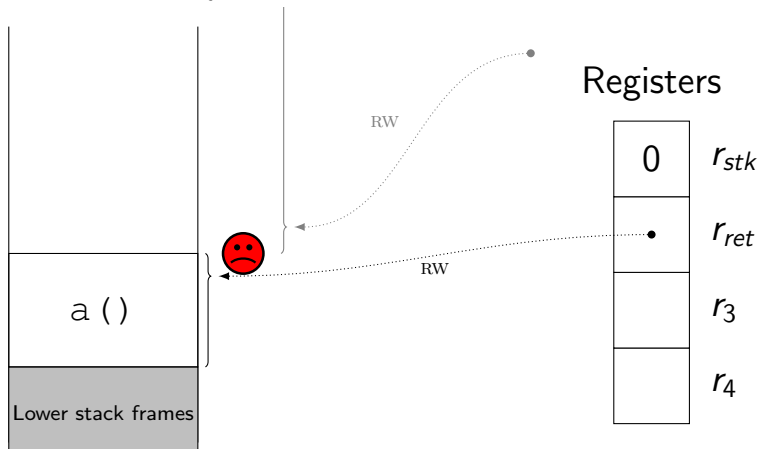
STKTOKENS by example

Call stack in memory



STKTOKENS by example

Call stack in memory



The formal result

Theorem

STKTOKENS *enforces local-state encapsulation and well-bracketed control flow for all programs.*

Proof.

By fully-abstract overlay semantics.



Overlay semantics

Syntax

move	rtmpl	42	load	rtmpl	rtmpl
store	rstk	rtmpl	cca	rtmpl	-21
cca	rstk	-1	cseal	rretc	rtmpl
geta	rtmpl	rstk	move	rretc	pc
cca	rretc	5	xjmp	r1	r2
move	rtmpl	pc	cseal	rretc	rtmpl
cca	rtmpl	-20	move	rtmpl	0

Semantics

Capability machine



Overlay semantics

Syntax

move	rtmpl	42	load	rtmpl	rtmpl
store	rstk	rtmpl	cca	rtmpl	-21
cca	rstk	-1	cseal	rretc	rtmpl
geta	rtmpl	rstk	move	rretc	pc
cca	rretc	5	xjmp	r1	r2
move	rtmpl	pc	cseal	rretc	rtmpl
cca	rtmpl	-20	move	rtmpl	0

Semantics



Capability machine
with a stack!

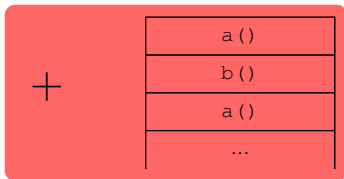
Overlay semantics

Syntax

```
move  rtmp1 42      load  rtmp1 rtmp1
store rstk rtmp1    cca   rtmp1 -21
cca   rstk -1       cseal rretc rtmp1
geta  rtmp1 rstk    move  rretc pc
cca   rretc 5       xjmp  r1 r2
move  rtmp1 pc      cseal rretc rtmp1
cca   rtmp1 -20     move  rtmp1 0
```

Capability machine
with a stack!

Semantics



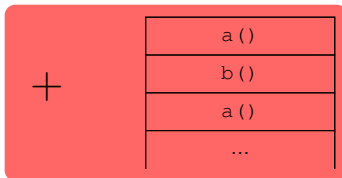
Overlay semantics

Syntax

```
move rtmp1 42      load rtmp1 rtmp1
store rstk rtmp1   cca rtmp1 -21
cca rstk 1          cseal rretc rtmp1
geta rtmp1 rstk     move rretc pc
cca rretc 5         x return
move rtmp1 pc       cseal rretc rtmp1
cca rtmp1 -20       move rtmp1 0
```

Capability machine
with a stack!

Semantics



Fully-abstract overlay semantics

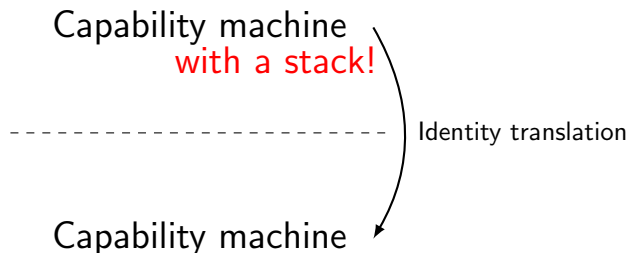
Capability machine

Fully-abstract overlay semantics

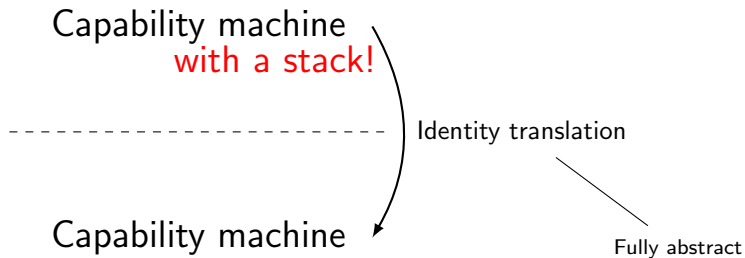
Capability machine
with a stack!

Capability machine

Fully-abstract overlay semantics



Fully-abstract overlay semantics



Summary of STKTOKENS

It's great!

Summary of STK TOKENS

It's great!

...except for the linear capabilities

Conclusions from PhD dissertation

- ▶ *Well-bracketed control flow* and *local-state encapsulation* can be enforced on capability machines.
- ▶ We can prove the enforcement correct using advanced reasoning techniques.

Looking forward...

- ▶ This dissertation is a step on the way towards real secure compilers that target capability machines.

Looking forward...

- ▶ This dissertation is a step on the way towards real secure compilers that target capability machines. We still need to:
 - ▶ Enforce other high-level abstractions.

Looking forward...

- ▶ This dissertation is a step on the way towards real secure compilers that target capability machines. We still need to:
 - ▶ Enforce other high-level abstractions.
 - ▶ Mechanise all proofs.

Looking forward...

- ▶ This dissertation is a step on the way towards real secure compilers that target capability machines. We still need to:
 - ▶ Enforce other high-level abstractions.
 - ▶ Mechanise all proofs.
 - ▶ Program logic?
 - ▶ Proof automation?

Looking forward...

- ▶ This dissertation is a step on the way towards real secure compilers that target capability machines. We still need to:
 - ▶ Enforce other high-level abstractions.
 - ▶ Mechanise all proofs.
 - ▶ Program logic?
 - ▶ Proof automation?
 - ▶ Proof of concept secure compiler.

Thank you!

Bonus slides!

The mistake

For a register file with the contents

- ▶ $((E, \text{GLOBAL}), b, e, a)$ in register r_1
- ▶ $\text{encode}((E, \text{LOCAL}))$ in register r_2

executing `restrict r_1 r_2` would result in a register file with

- ▶ $((E, \text{LOCAL}), b, e, a)$ in register r_1

seemingly fine

The mistake

For a register file with the contents

- ▶ $((E, \text{GLOBAL}), b, e, a)$ in register r_1
- ▶ $\text{encode}((E, \text{LOCAL}))$ in register r_2

executing `restrict r_1 r_2` would result in a register file with

- ▶ $((E, \text{LOCAL}), b, e, a)$ in register r_1

seemingly fine, but the logical relation does not account for this.

Fixing the mistake

Either

- ▶ Prohibit `restrict` from restricting an enter-capability from global to local; or
- ▶ Fix the logical relation: Insert $\forall g' \sqsubseteq g$ where appropriate, e.g. in enter-condition:

$$\begin{aligned} \text{enterCondition}(g)(W) = \\ \{ (n, (base, end, a)) \mid \forall n' < n. \\ \forall W' \sqsupseteq W. \\ (n', ((RX, g), base, end, a)) \in \mathcal{E}(W') \} \end{aligned}$$

where $g = \text{LOCAL} \Rightarrow \sqsubseteq = \sqsubseteq^{pub}$

and $g = \text{GLOBAL} \Rightarrow \sqsubseteq = \sqsubseteq^{priv}$

Flavours of linear capabilities

- ▶ unique linear capability
 - ▶ No alias
 - ▶ STKTOKENS-paper
- ▶ linear
 - ▶ May have aliases
 - ▶ CHERI

