Formalizing SPARCv8 Instruction Set Architecture in Coq

Jiawei Wang[1] Ming Fu[1]
Lei Qiao[2] Xinyu Feng[1]

University of Science and Technology of China[1]
Beijing Institute of Control Engineering[2]

October 24, 2017

Motivations

Motivations

Computer System

widely used in many safety critical areas

Motivations

Computer System

↓

Operating System

widely used in many safety critical areas

implemented in C + assembly language

Motivations

Computer System

↓

Operating System

widely used in many safety critical areas

implemented in C + assembly language

verified

not verified

Motivations

Computer System — widely used in many safety critical areas

↓

Operating System — implemented in C + assembly language

↓ verified

SpaceOS — C + SPARCv8 assembly language

↘ not verified

Motivations

Computer System — widely used in many safety critical areas

↓

Operating System — implemented in C + assembly language

↓

SpaceOS

verified

not verified

C + SPARCv8 assembly language

To verify SpaceOS, it is inevitable to formally verify the assembly code

Verification

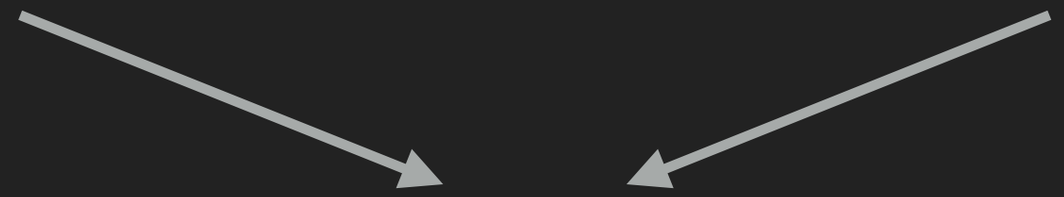
Verification

Specification

(mathematical description)

Program

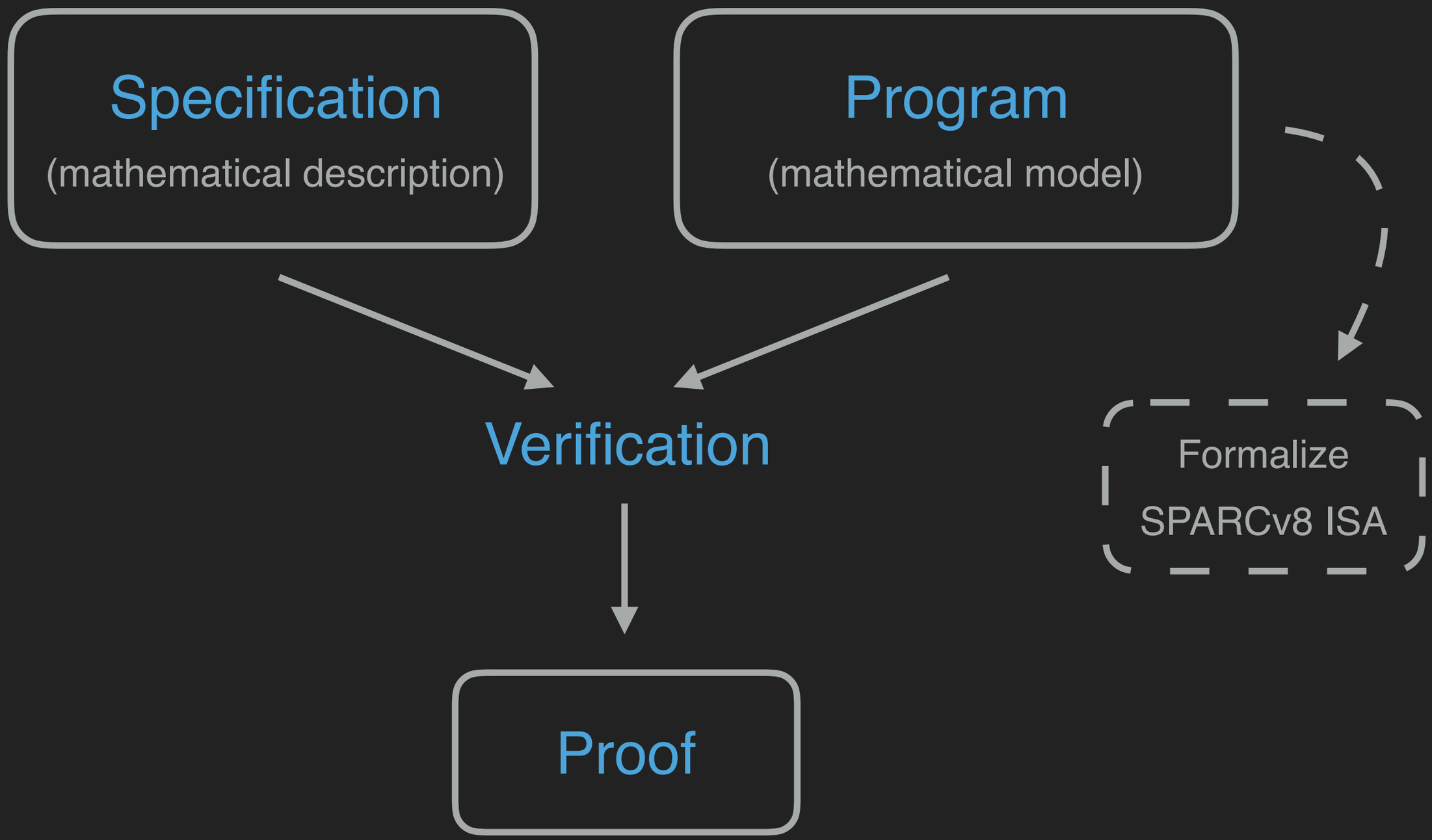
(mathematical model)



Verification

Proof

Verification



Formalize a Language?

Formalize a Language?

- Syntax

  C ::= while e do C I

  if e then C1 else C2 I …

Formalize a Language?

- Syntax

  C ::= while e do C I

  if e then C1 else C2 I …

- Machine State

  state ::= (M, …)

Formalize a Language?

- Syntax

  C ::= while e do C I

  if e then C1 else C2 I …

- Machine State

  state ::= (M, …)

- Operational Semantics

  (C, state) → (C’, state')

Contributions

- Modeling SPARCv8 ISA
  - Syntax
  - Machine State
  - Operational Semantics

Contributions

- Modeling SPARCv8 ISA
  - Syntax
  - Machine State
  - Operational Semantics
- Properties of the Model
  - Determinacy Property
  - Isolation Property

Contributions

- Modeling SPARCv8 ISA
  - Syntax
  - Machine State
  - Operational Semantics
- Properties of the Model
  - Determinacy Property
  - Isolation Property
- Verifying a Window Overflow Trap Handler

Contributions

- Modeling SPARCv8 ISA
  - Syntax
  - Machine State
  - Operational Semantics
- Properties of the Model
  - Determinacy Property
  - Isolation Property
- Verifying a Window Overflow Trap Handler
- In Coq (11,000 lines of code)

Challenges


CWP
(current window)
WIM
CWP–1
CWP+1
w7 ins
w7 locals
w1 outs
w0 locals
w0 outs
w0 ins
w1 locals
w7 outs
w6 ins
w1 ins
w2 outs
w6 locals
RESTORE,
RETT
SAVE,
trap
w2 locals
w6 outs
w2 ins
w3 outs
w4 ins
w5 locals
w4 outs
w4 locals

Challenges

SPARCv8 has the following unique mechanisms:

- Delayed-transfer instruction
- Register windows

Example: Add Three Variables

```
caller:
  mov    1,  %o0
  mov    2,  %o1
  call       sum3
  mov    3,  %o2
  …
```

```
sum3:
  save    %sp, -64, %sp
  add     %i0, %i1, %l7
  add     %l7, %i2, %l7
  ret
  restore %l7,   0, %o0
```

Example: Add Three Variables

```
caller:
   mov     1,   %o0
   mov     2,   %o1
   call       sum3
   mov     3,   %o2
   …
```

place the arguments 1 and 2

call sum3

place the argument 3

```
sum3:
   save     %sp, -64, %sp
   add      %i0, %i1, %l7
   add      %l7, %i2, %l7
   ret
   restore %l7,    0, %o0
```

Example: Add Three Variables

```
caller:
  mov    1,  %o0
  mov    2,  %o1
  call       sum3
  mov    3,  %o2
  …
```

```
sum3:
  save    %sp, -64, %sp
  add     %i0, %i1, %l7
  add     %l7, %i2, %l7
  ret
  restore %l7,   0, %o0
```

Example: Add Three Variables

```
caller:
PC   mov    1,  %o0
nPC  mov    2,  %o1
     call       sum3
     mov    3,  %o2
     …
```

```
sum3:
     save    %sp, -64, %sp
     add     %i0, %i1, %l7
     add     %l7, %i2, %l7
     ret
     restore %l7,   0, %o0
```

Example: Add Three Variables

```
caller:
    mov    1,  %o0
PC  mov    2,  %o1
nPC call       sum3
    mov    3,  %o2
    …
```

```
sum3:
    save    %sp, -64, %sp
    add     %i0, %i1, %l7
    add     %l7, %i2, %l7
    ret
    restore %l7,   0, %o0
```

Example: Add Three Variables

```
caller:
    mov    1,  %o0
PC  mov    2,  %o1
nPC call       sum3
    mov    3,  %o2
    …

sum3:
    save   %sp, -64, %sp
    add    %i0, %i1, %l7
    add    %l7, %i2, %l7
    ret
    restore %l7,   0, %o0
```

Example: Add Three Variables

```
caller:
    mov    1,  %o0
    mov    2,  %o1
PC  call       sum3
nPC mov    3,  %o2
    …

sum3:
    save   %sp, -64, %sp
    add    %i0, %i1, %l7
    add    %l7, %i2, %l7
    ret
    restore %l7,   0, %o0
```

Example: Add Three Variables

```
caller:
    mov    1,  %o0
    mov    2,  %o1
PC  call      sum3
nPC mov    3,  %o2
    …

                        sum3:
                          save    %sp, -64, %sp
                          add     %i0, %i1, %l7
                          add     %l7, %i2, %l7
                          ret
                          restore %l7,   0, %o0
```

Example: Add Three Variables

```
caller:
    mov     1,  %o0
    mov     2,  %o1
    call        sum3
PC  mov     3,  %o2
    …

                    sum3:
                nPC save    %sp, -64, %sp
                    add     %i0, %i1, %l7
                    add     %l7, %i2, %l7
                    ret
                    restore %l7,   0, %o0
```

Example: Add Three Variables

```
caller:
    mov     1,  %o0
    mov     2,  %o1
    call        sum3
PC  mov     3,  %o2
    …

                    sum3:
nPC                 save    %sp, -64, %sp
                    add     %i0, %i1, %l7
                    add     %l7, %i2, %l7
                    ret
                    restore %l7,   0, %o0
```

Example: Add Three Variables

```
caller:
  mov    1,  %o0
  mov    2,  %o1
  call       sum3
  mov    3,  %o2
  …
```

```
sum3:
PC  save    %sp, -64, %sp
nPC add     %i0, %i1, %l7
    add     %l7, %i2, %l7
    ret
    restore %l7,   0, %o0
```

Example: Add Three Variables

```
caller:
  mov    1,  %o0
  mov    2,  %o1
  call       sum3
  mov    3,  %o2
  …

sum3:
PC  save   %sp, -64, %sp
nPC add    %i0, %i1, %l7
    add    %l7, %i2, %l7
    ret
    restore %l7,   0, %o0
```

Example: Add Three Variables

```
caller:
  mov    1,  %o0
  mov    2,  %o1
  call      sum3
  mov    3,  %o2
  …

sum3:
  save    %sp, -64, %sp
  add     %i0, %i1, %l7
  add     %l7, %i2, %l7
  ret
  restore %l7,   0, %o0
```

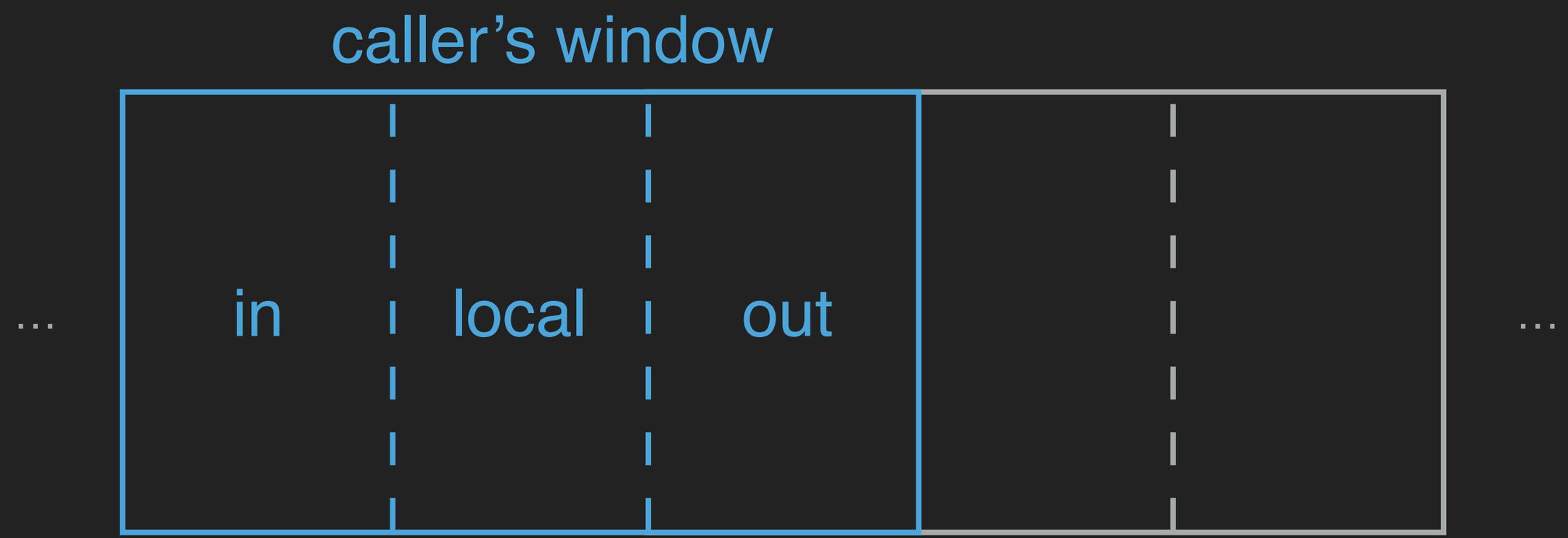
PC

nPC

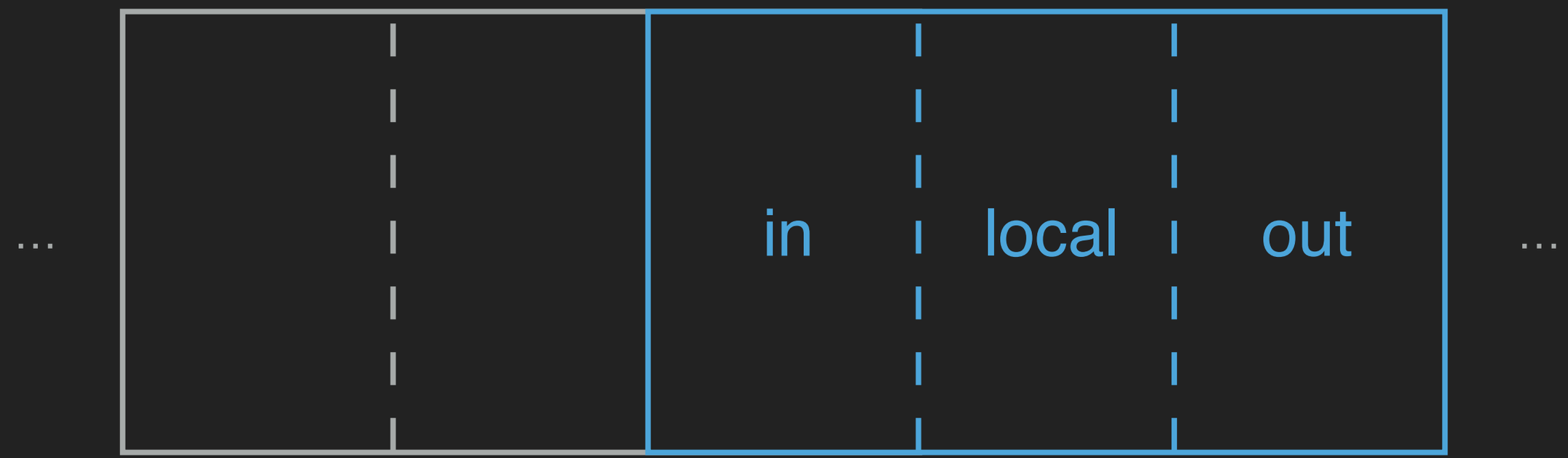
Delayed-Transfer:

call is placed in front of **mov 3**,

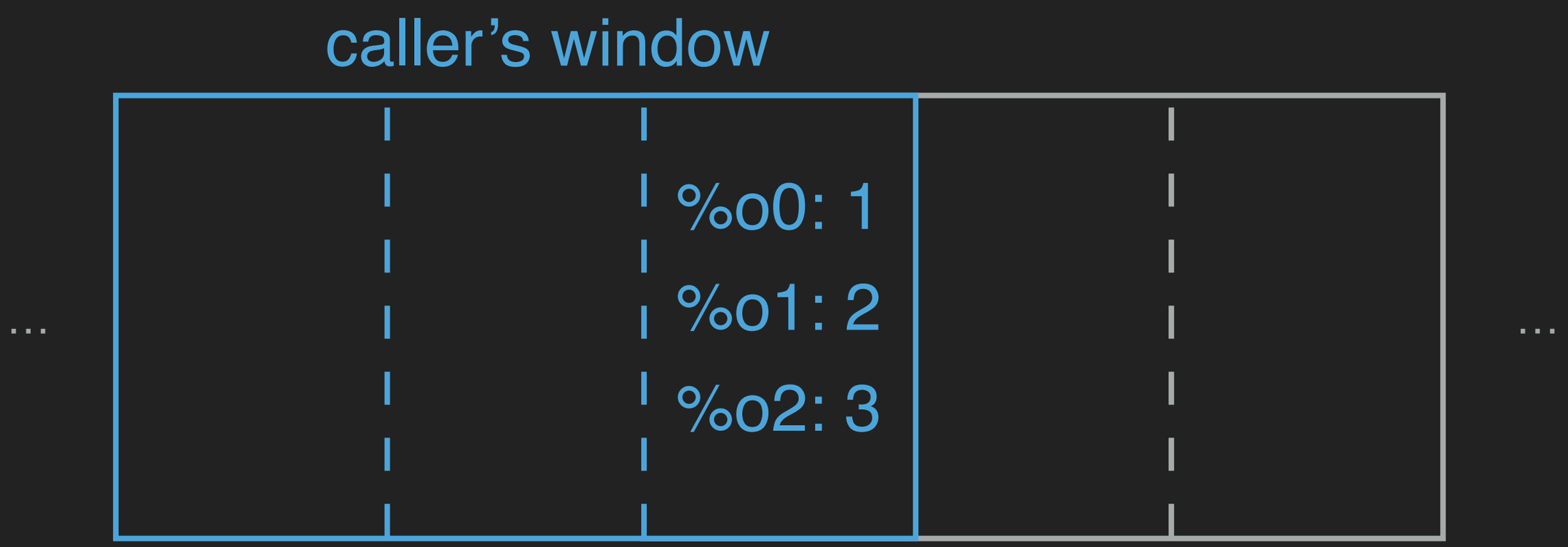
but it's executed after **mov 3**.



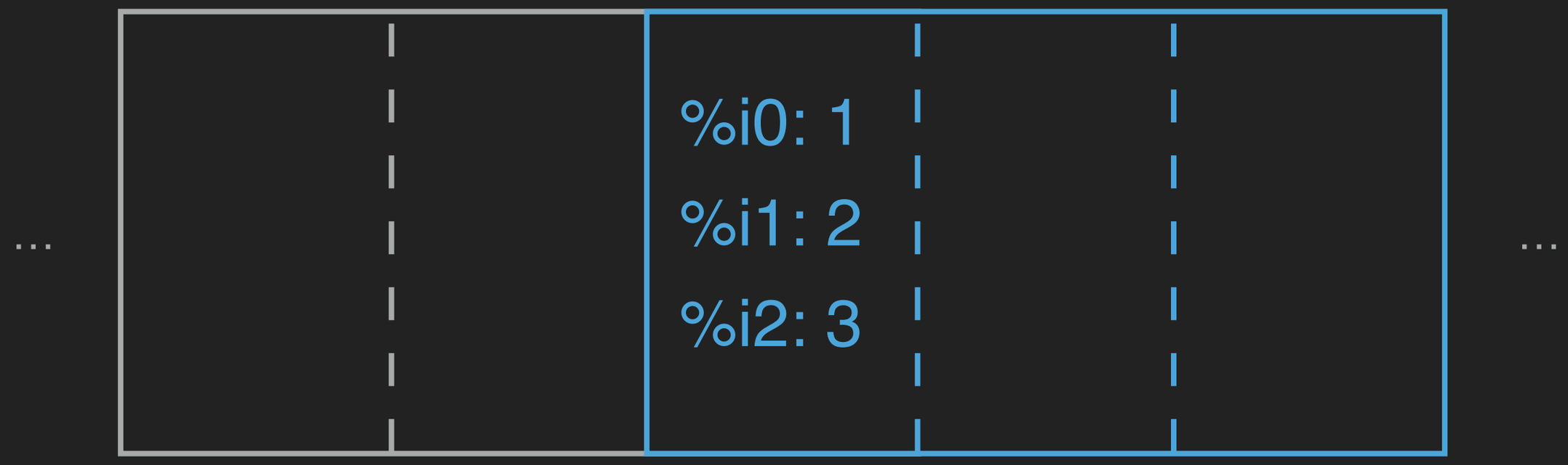

```
caller:                    sum3:
  mov    1,  %o0     PC      save    %sp, -64, %sp
  mov    2,  %o1     nPC     add     %i0, %i1, %l7
  call      sum3             add     %l7, %i2, %l7
  mov    3,  %o2             ret
  …                          restore %l7,   0, %o0
```



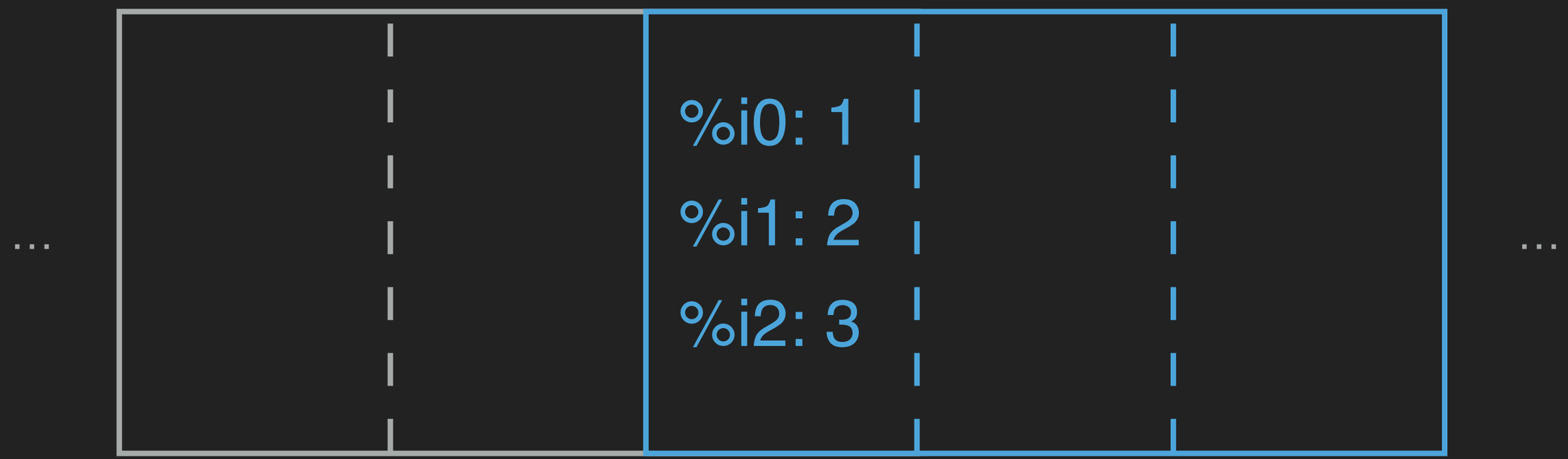

```
caller:                    sum3:
  mov    1,  %o0     PC    save    %sp, -64, %sp
  mov    2,  %o1     nPC   add     %i0, %i1, %l7
  call     sum3            add     %l7, %i2, %l7
  mov    3,  %o2           ret
  …                        restore %l7,   0, %o0
```



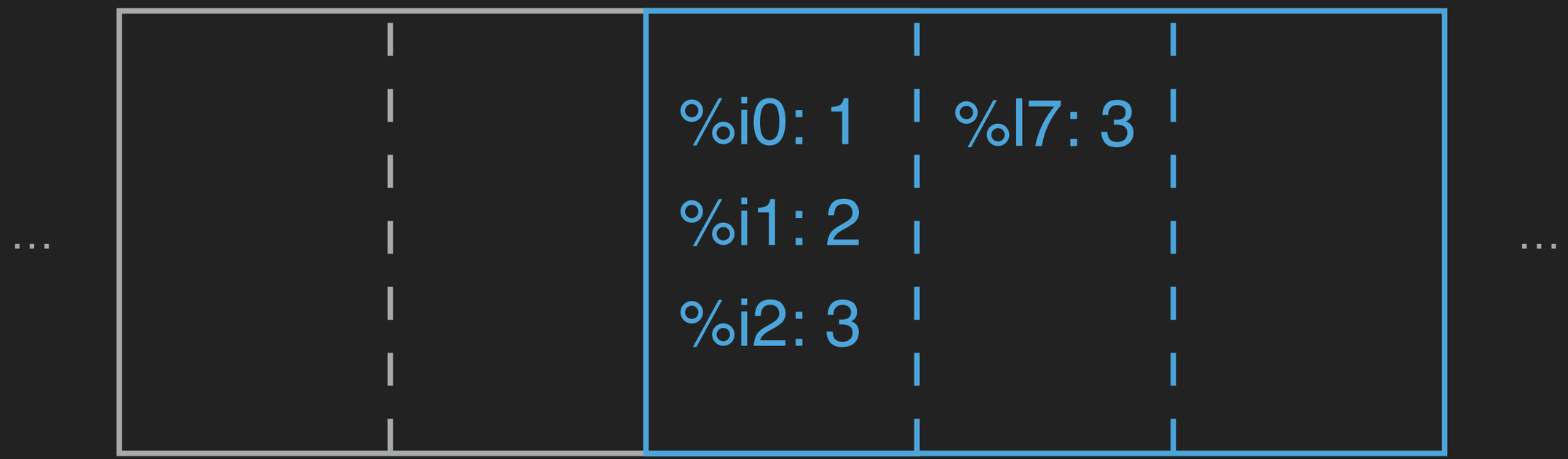

```
caller:                    sum3:
  mov    1,  %o0     PC      save    %sp, -64, %sp
  mov    2,  %o1     nPC     add     %i0, %i1, %l7
  call      sum3             add     %l7, %i2, %l7
  mov    3,  %o2             ret
  …                          restore %l7,   0, %o0
```



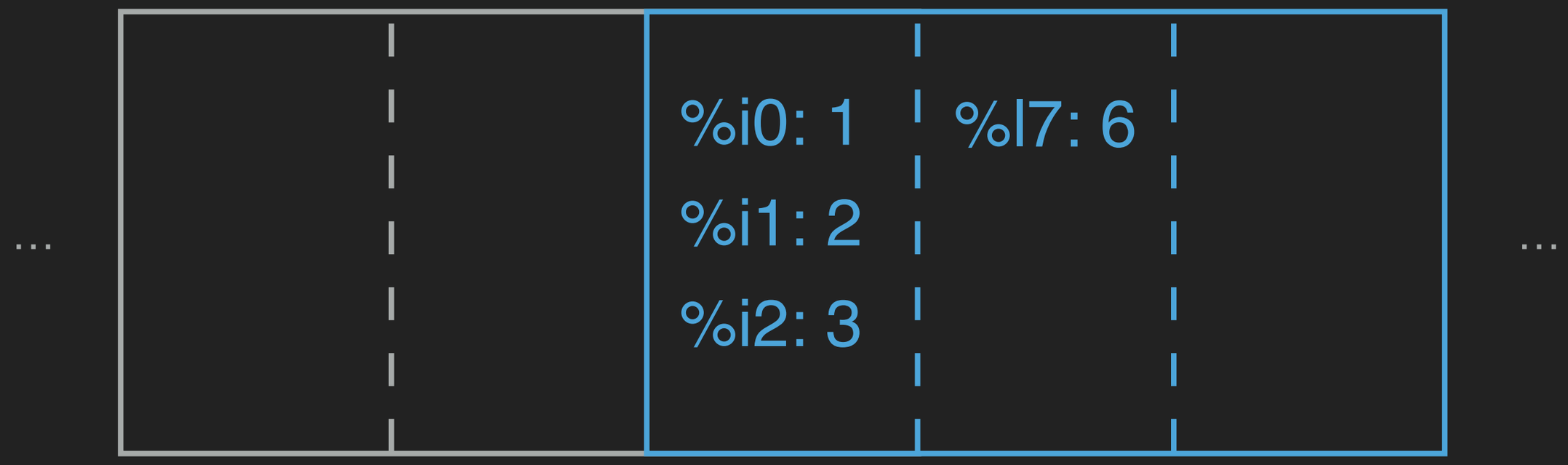

```
caller:                    sum3:
  mov    1,  %o0     PC      save    %sp, -64, %sp
  mov    2,  %o1     nPC     add     %i0, %i1, %l7
  call      sum3             add     %l7, %i2, %l7
  mov    3,  %o2             ret
  …                          restore %l7,   0, %o0
```



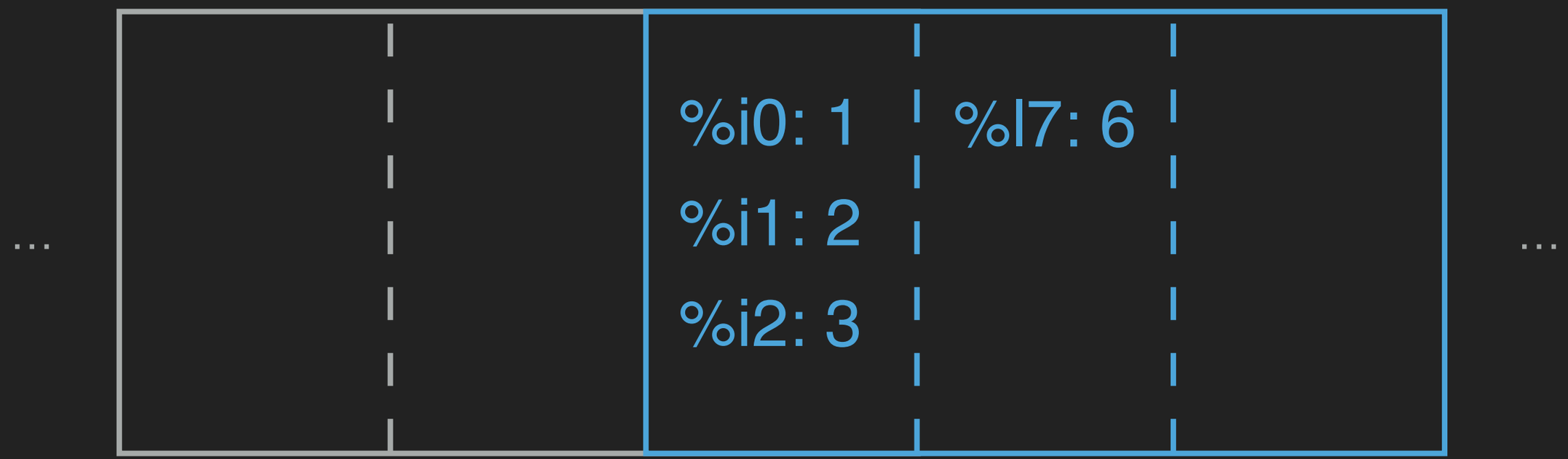

```
caller:                    sum3:
  mov    1,  %o0               save    %sp, -64, %sp
  mov    2,  %o1      PC       add     %i0, %i1, %l7
  call       sum3     nPC      add     %l7, %i2, %l7
  mov    3,  %o2               ret
  …                            restore %l7,   0, %o0
```




```
caller:                    sum3:
  mov    1,  %o0               save    %sp, -64, %sp
  mov    2,  %o1               add     %i0, %i1, %l7
  call      sum3      PC       add     %l7, %i2, %l7
  mov    3,  %o2      nPC      ret
  …                             restore %l7,   0, %o0
```

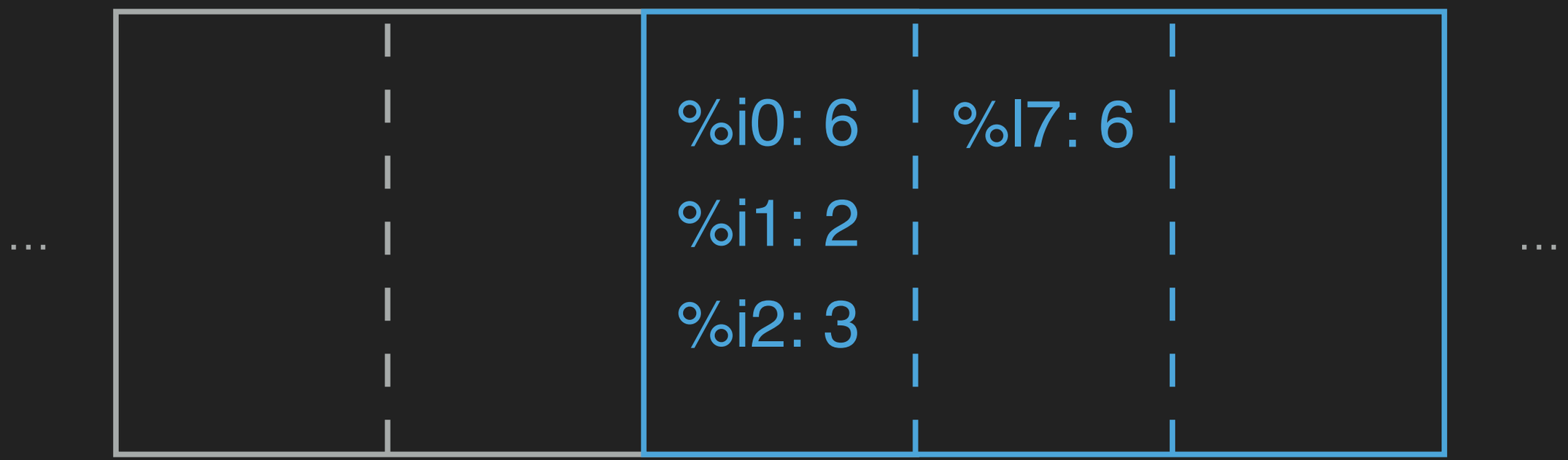



```
caller:                    sum3:
  mov    1,  %o0               save    %sp, -64, %sp
  mov    2,  %o1               add     %i0, %i1, %l7
  call      sum3               add     %l7, %i2, %l7
  mov    3,  %o2      PC       ret
  …                   nPC      restore %l7,   0, %o0
```



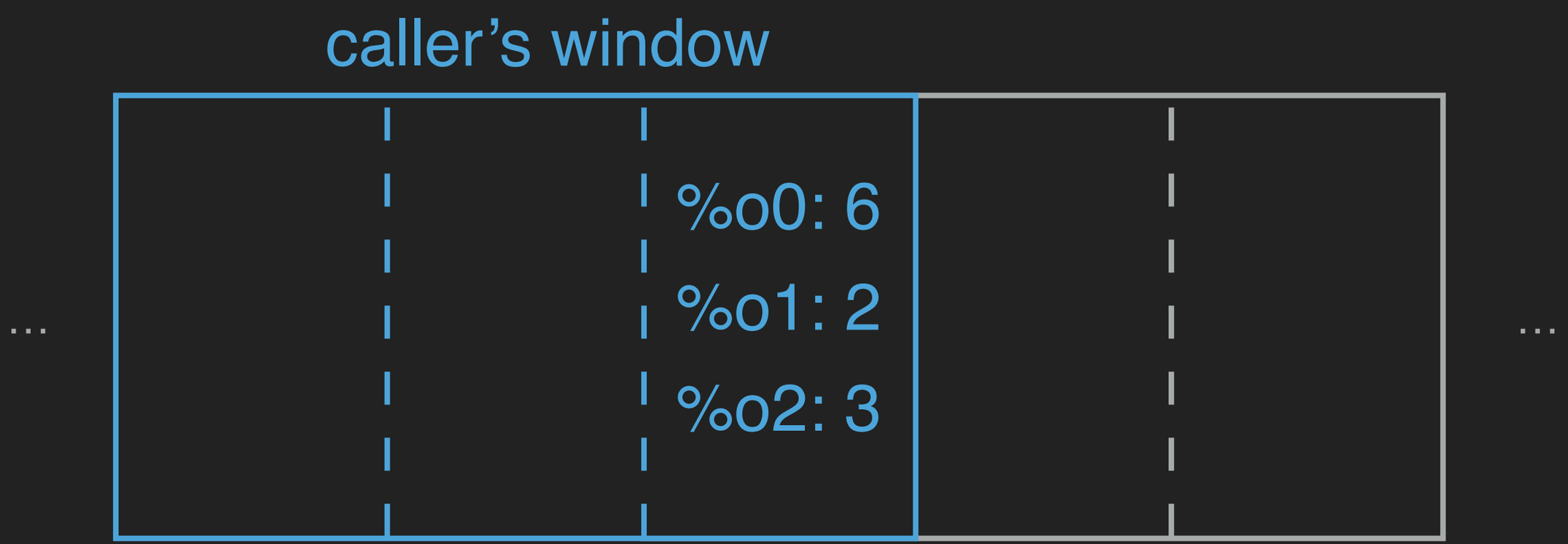

```
caller:
    mov    1,  %o0
    mov    2,  %o1
    call      sum3
    mov    3,  %o2
nPC …
```

```
sum3:
    save    %sp, -64, %sp
    add     %i0, %i1, %l7
    add     %l7, %i2, %l7
    ret
PC  restore %l7,   0, %o0
```




```
caller:
    mov    1,  %o0
    mov    2,  %o1
    call      sum3
    mov    3,  %o2
nPC …
```

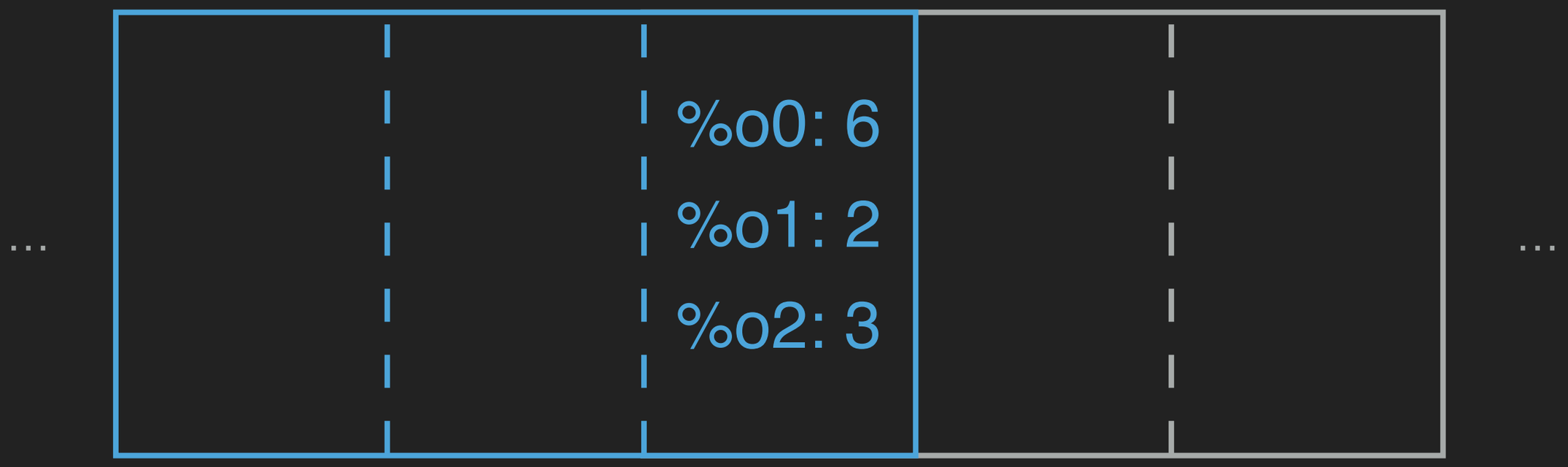
```
sum3:
    save    %sp, -64, %sp
    add     %i0, %i1, %l7
    add     %l7, %i2, %l7
    ret
PC  restore %l7,   0, %o0
```




```
caller:
   mov    1,  %o0
   mov    2,  %o1
   call      sum3
   mov    3,  %o2
nPC …
```

```
sum3:
   save    %sp, -64, %sp
   add     %i0, %i1, %l7
   add     %l7, %i2, %l7
   ret
PC restore %l7,   0, %o0
```

caller's window




```
caller:                    sum3:
  mov    1,  %o0               save    %sp, -64, %sp
  mov    2,  %o1               add     %i0, %i1, %l7
  call      sum3               add     %l7, %i2, %l7
  mov    3,  %o2               ret
PC …                           restore %l7,   0, %o0
```

Challenges

SPARCv8 has the following unique mechanisms:

- Delayed-transfer instruction
- Register windows

Challenges

SPARCv8 has the following unique mechanisms:

- Delayed-transfer instruction
- Register windows
- Annulled delay instruction

Annulled Instruction

```
foo:
    beq bar
    xxx
    yyy
    yyy
    …
```

```
bar:
    xxx
    xxx
    xxx
    …
```

Annulled Instruction

```
foo:
    beq bar
    xxx
    yyy
    yyy
    …
```

```
bar:
    xxx
    xxx
    xxx
    …
```

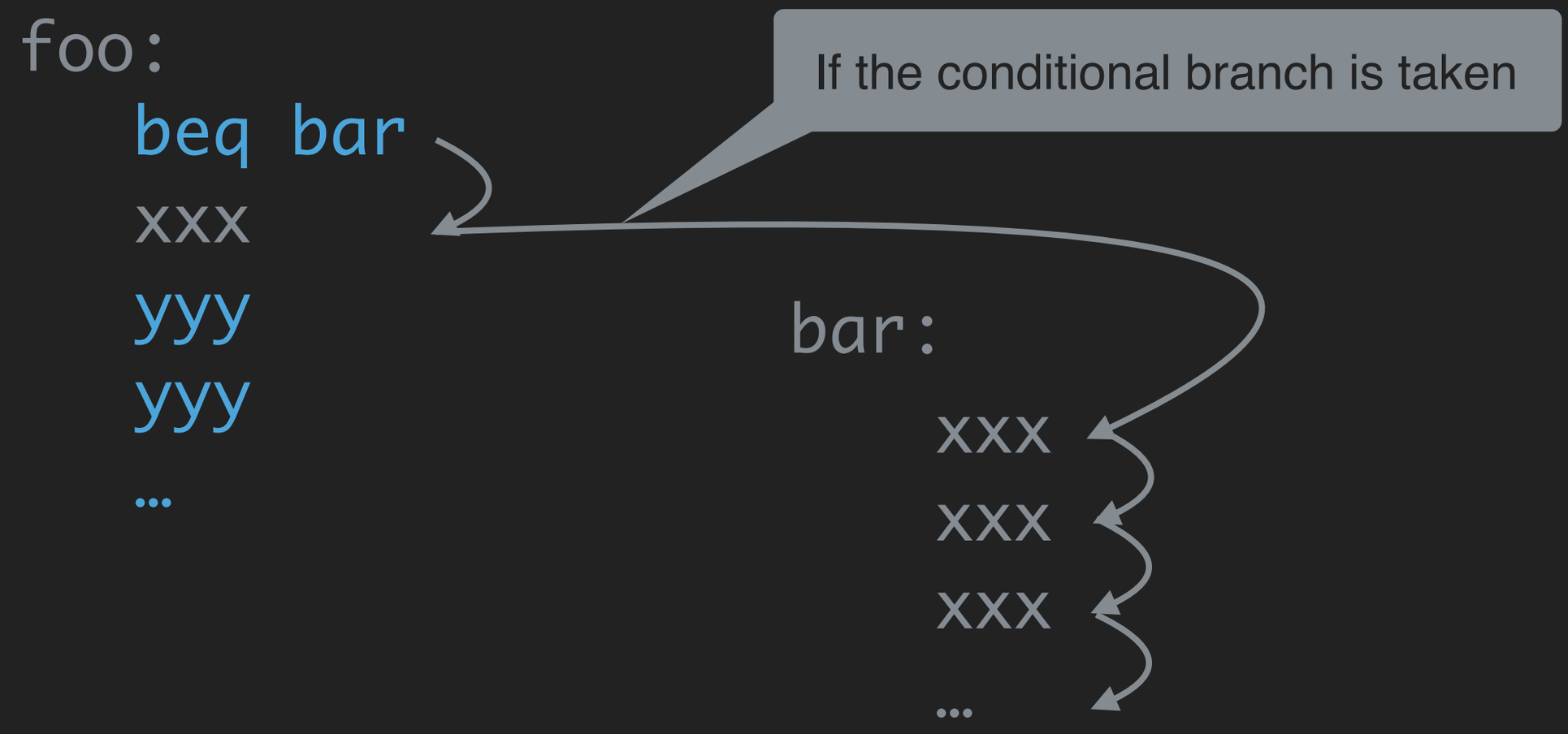
Annulled Instruction

```
foo:
    beq bar
    xxx
    yyy
    yyy
    …
```

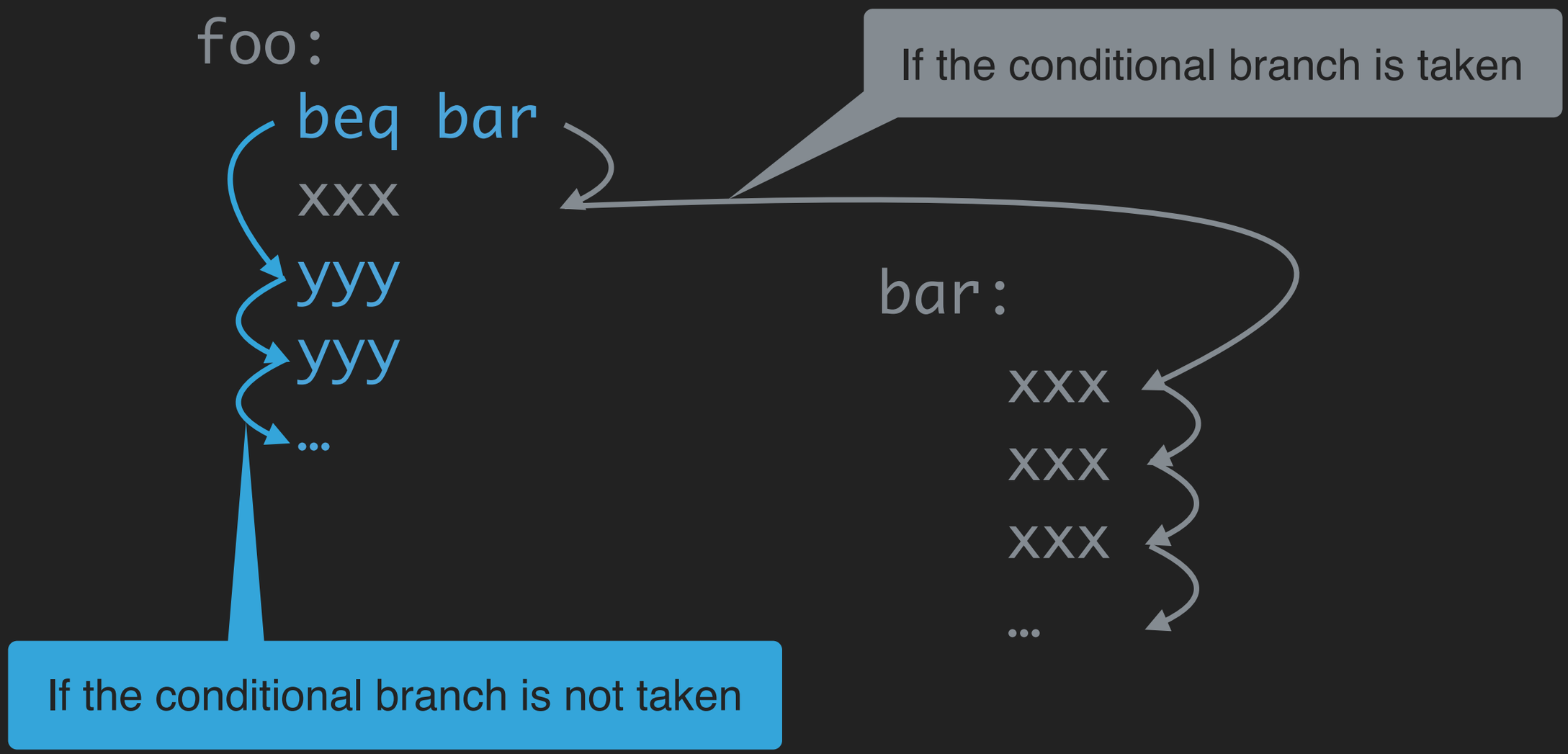
If the conditional branch is taken

```
bar:
    xxx
    xxx
    xxx
    …
```

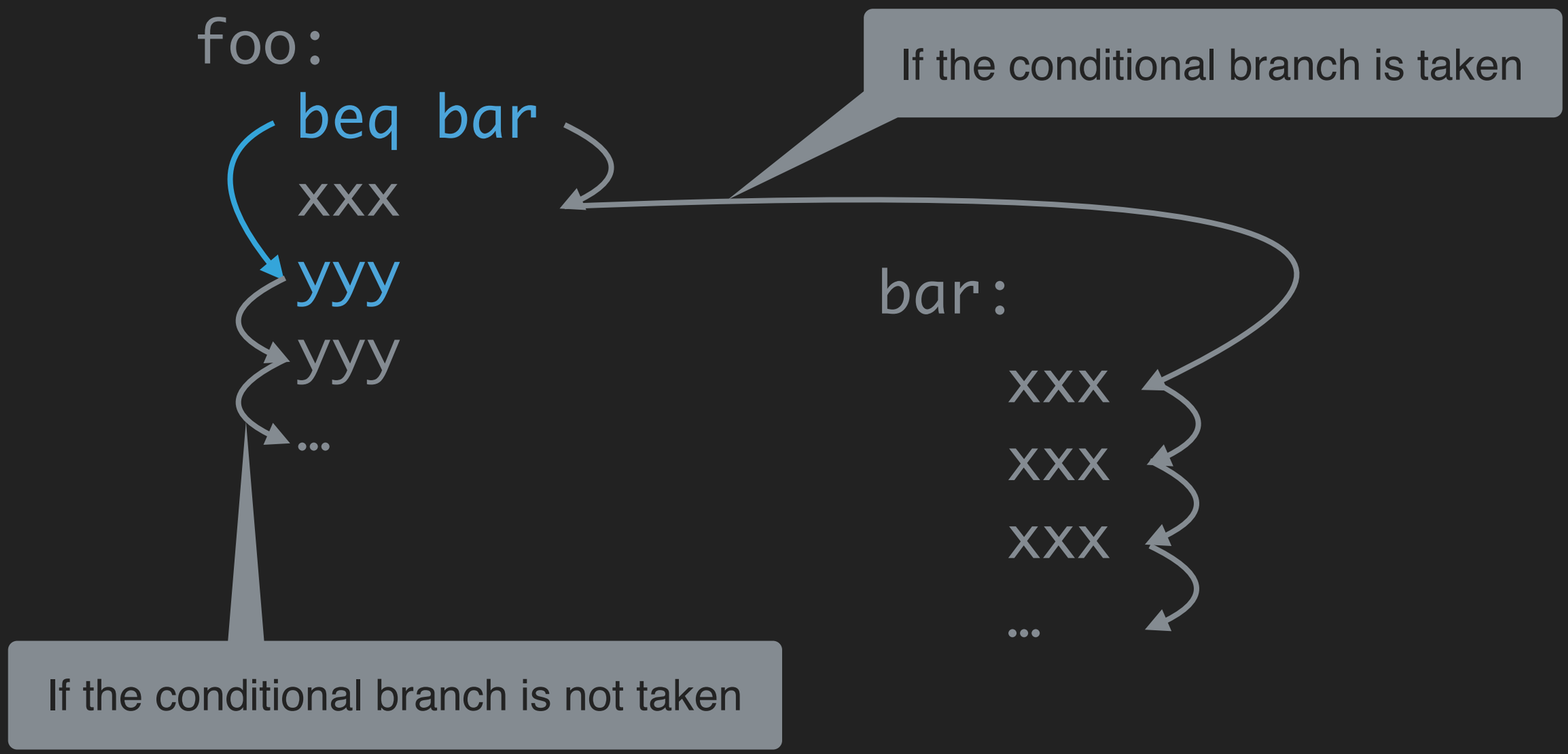
Annulled Instruction



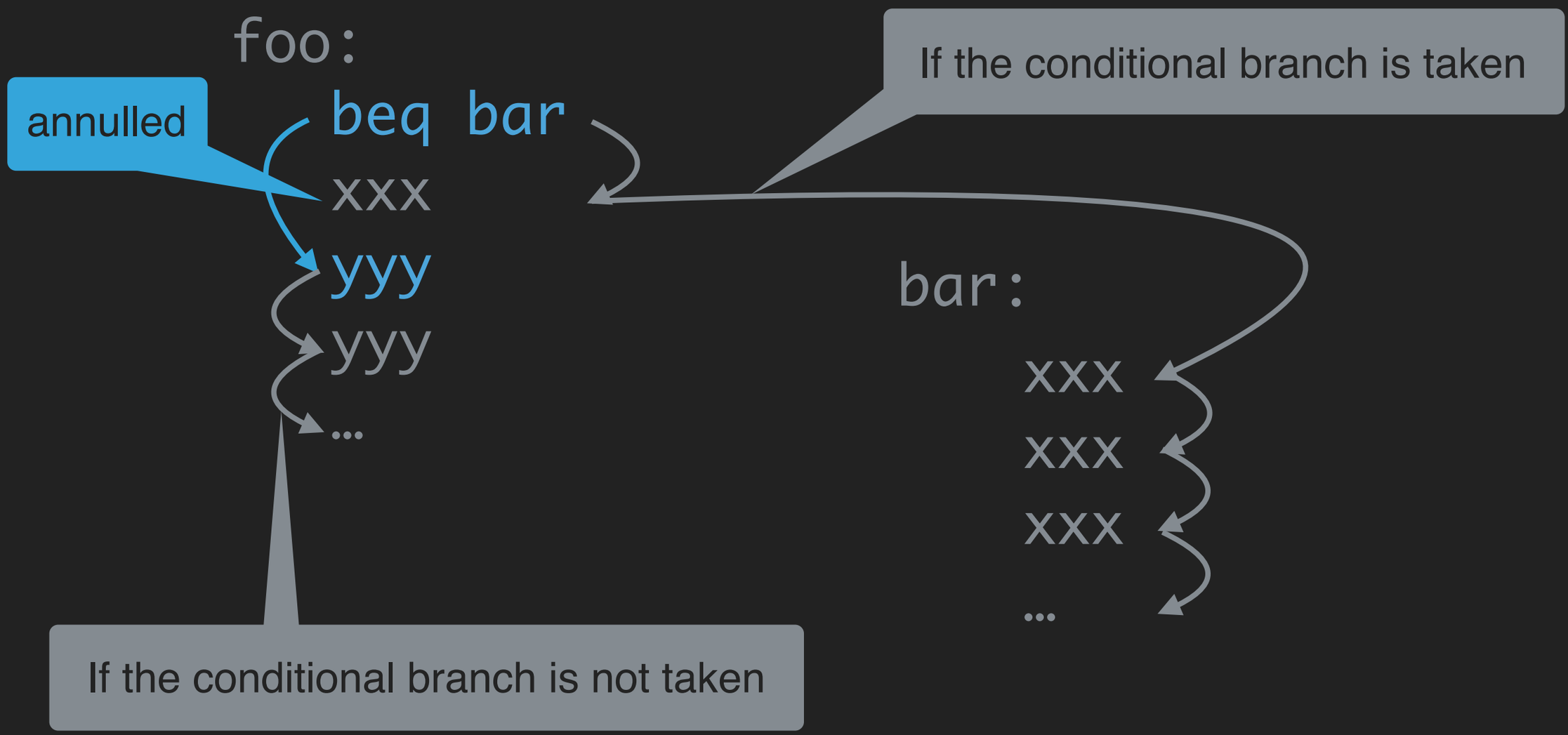
Annulled Instruction



Annulled Instruction



Annulled Instruction



Challenges

SPARCv8 has the following unique mechanisms:

- Delayed-transfer instruction
- Register windows
- Annulled delay instruction

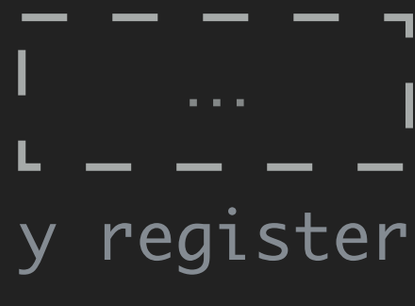
Challenges

SPARCv8 has the following unique mechanisms:

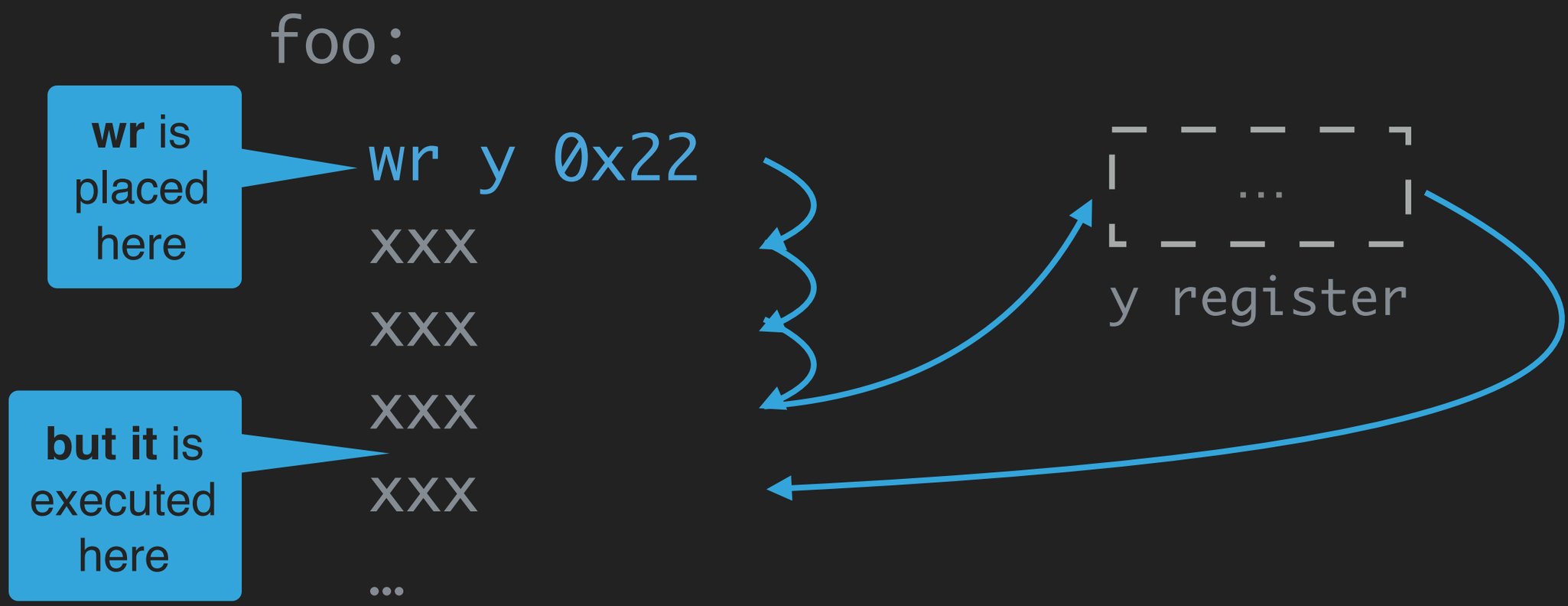
- [ ] Delayed-transfer instruction
- [ ] Register windows
- [ ] Annulled delay instruction
- [x] Delayed-write mechanism

Delayed-Write

```
foo:
    wr y 0x22
    xxx
    xxx
    xxx
    xxx
    …
```



Delayed-Write



Challenges

SPARCv8 has the following unique mechanisms:

- ☐ Delayed-transfer instruction
- ☐ Register windows
- ☐ Annulled delay instruction
- ☐ Delayed-write mechanism

Challenges

SPARCv8 has the following unique mechanisms:

- ☐ Delayed-transfer instruction
- ☐ Register windows
- ☐ Annulled delay instruction
- ☐ Delayed-write mechanism
- ☐ User mode and supervisor mode
- ☐ A variety of traps

Trap and Mode

```
foo:
 ld r0 0x123
 …
 ticc 5
 …
 save
 …
```

Trap and Mode

may cause an error

```
foo:
ld r0 0x123
…
ticc 5
…
save
…
```

Trap and Mode

```
foo:
ld r0 0x123
…
ticc 5
…
save
…
```

may cause an error

system call

Trap and Mode

```
foo:
ld r0 0x123
…
ticc 5
…
save
…
```

may cause an error

system call

may cause an exception

Trap and Mode

```
foo:
ld r0 0x123
…
ticc 5
…
save
…
```

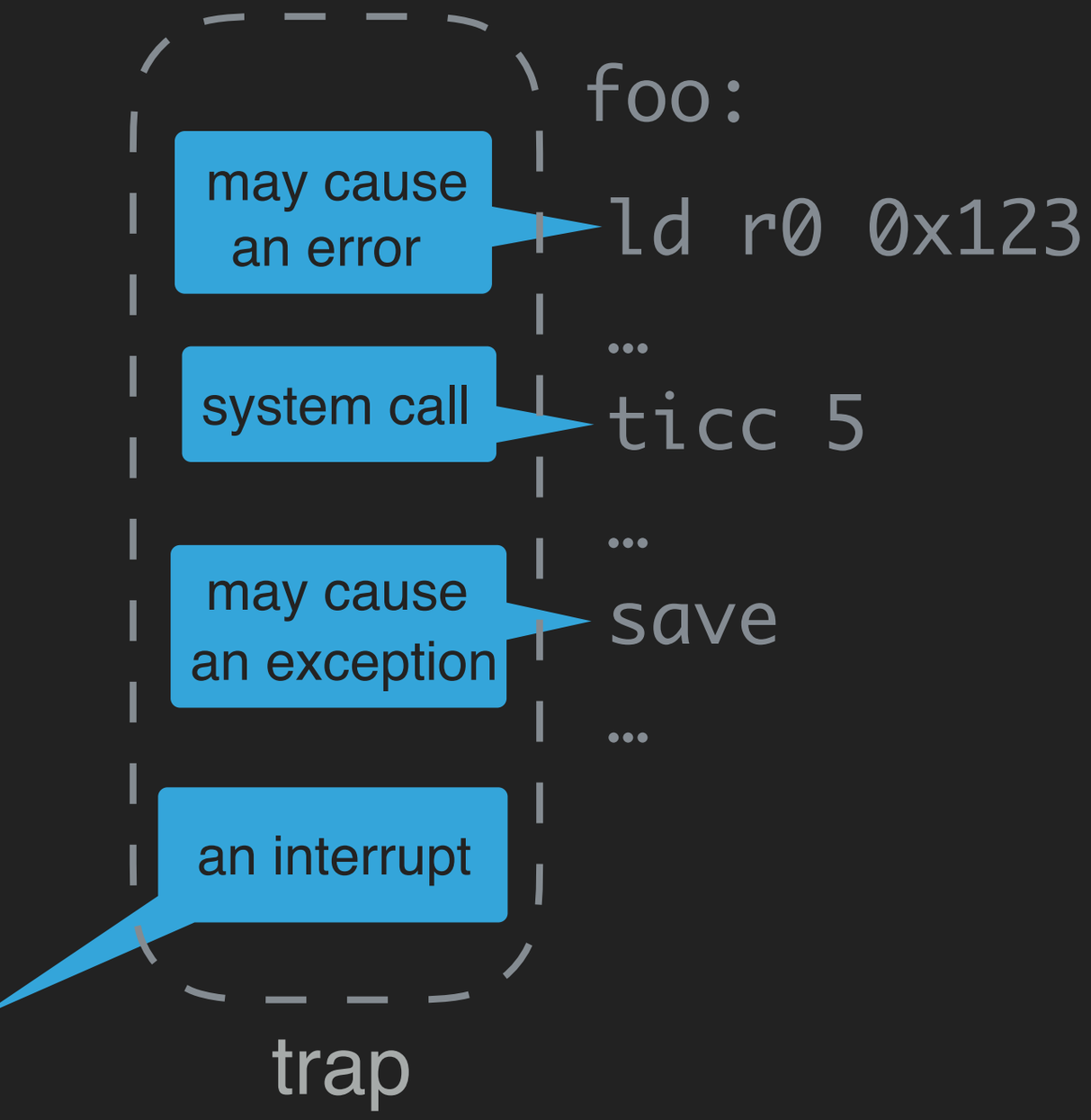
may cause an error

system call

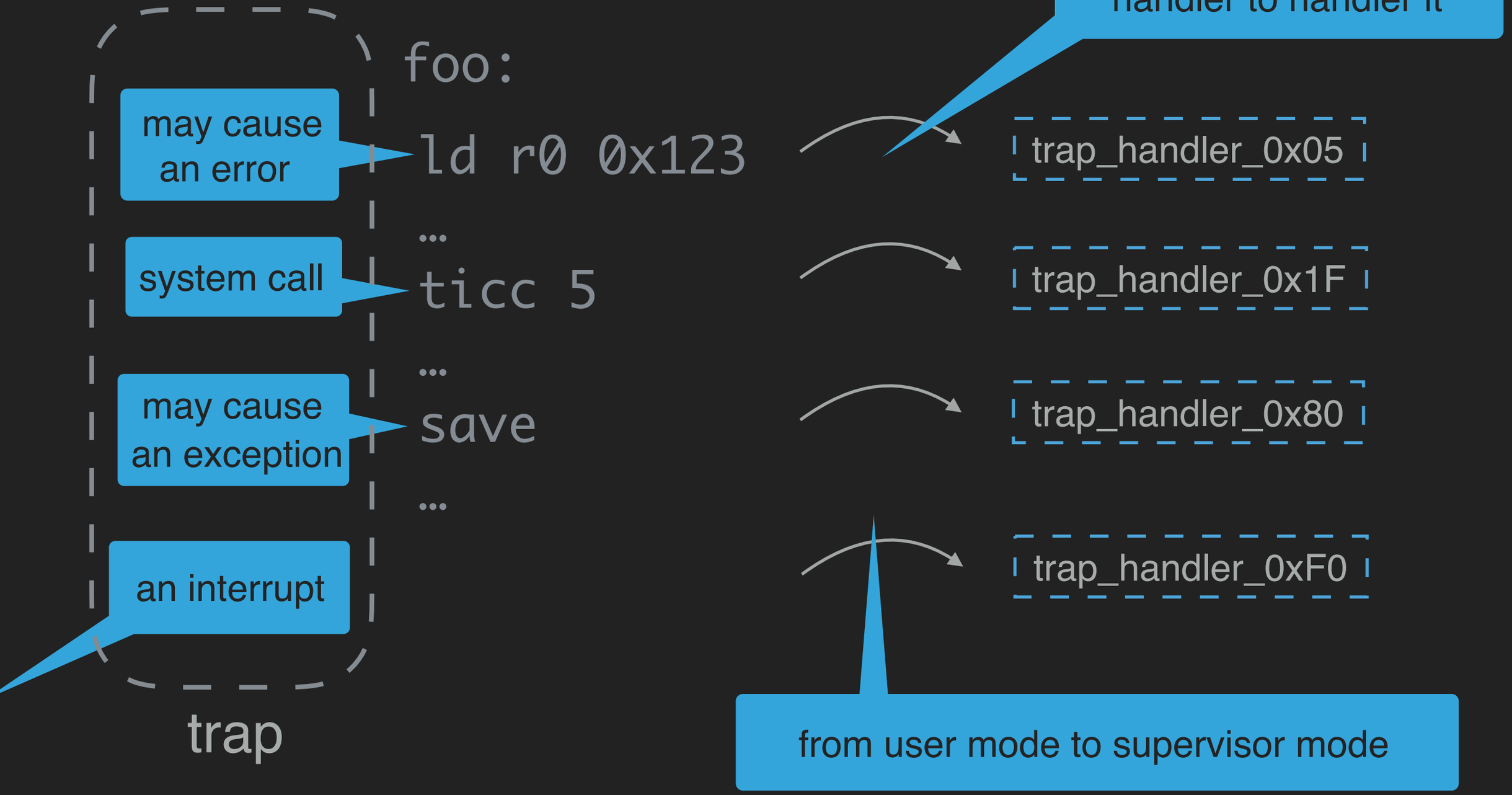
may cause an exception

an interrupt

Trap and Mode



Trap and Mode



Challenges

SPARCv8 has the following unique mechanisms:

- Delayed-transfer instruction
- Register windows
- Annulled delay instruction
- Delayed-write mechanism
- User mode and supervisor mode
- A variety of traps

Outline

Outline

- Modeling SPARCv8 ISA
  - Syntax
  - Operational Semantics
  - Machine State

Syntax

SPARC instructions

(SparcIns) i ::= bicca η β | save rs α rd | ticc η γ |
restore rs α rd | rett β | wr rd α ς | …

(OpExp) α ::= r | w — operand expressions

(AddrExp) β ::= α | r + α — address expressions

(TrapExp) γ ::= α | r + α — trap expressions

(Word) w ∈ Int32 — 32-bit integers

(GenReg) r ::= r0 | ... | r31 — general registers

(TestCond) η ::= al | nv | ne | eq | … — test conditions

(Symbol) ς ::= psr | wim | tbr | y | asr — symbol registers

(AsReg) asr ::= asr0 | ... | asr31 — ancillary state registers

Syntax in Coq

```
Inductive SparcIns: Type :=
 | bicca: TestCond -> AddrExp -> SparcIns
 | jmpl: AddrExp -> GenReg -> SparcIns
 | ld: AddrExp -> GenReg -> SparcIns
 | st: GenReg -> AddrExp -> SparcIns
 | save: GenReg -> OpExp -> GenReg -> SparcIns
 | restore: GenReg -> OpExp -> GenReg -> SparcIns
 | ticc: TestCond -> TrapExp -> SparcIns
 | rett: AddrExp -> SparcIns
 | rd: Symbol -> GenReg -> SparcIns
 | wr: GenReg -> OpExp -> Symbol -> SparcIns
 | …
```

Syntax in Coq

```
Inductive Symbol: Type :=
 | psr: Symbol
 | wim: Symbol
 | tbr: Symbol
 | y: Symbol
 | Sasr: AsReg -> Symbol.

Inductive GenReg: Type :=
 | r0: GenReg
 | …

Inductive AsReg: Type :=
 | asr0: AsReg
 | …
```

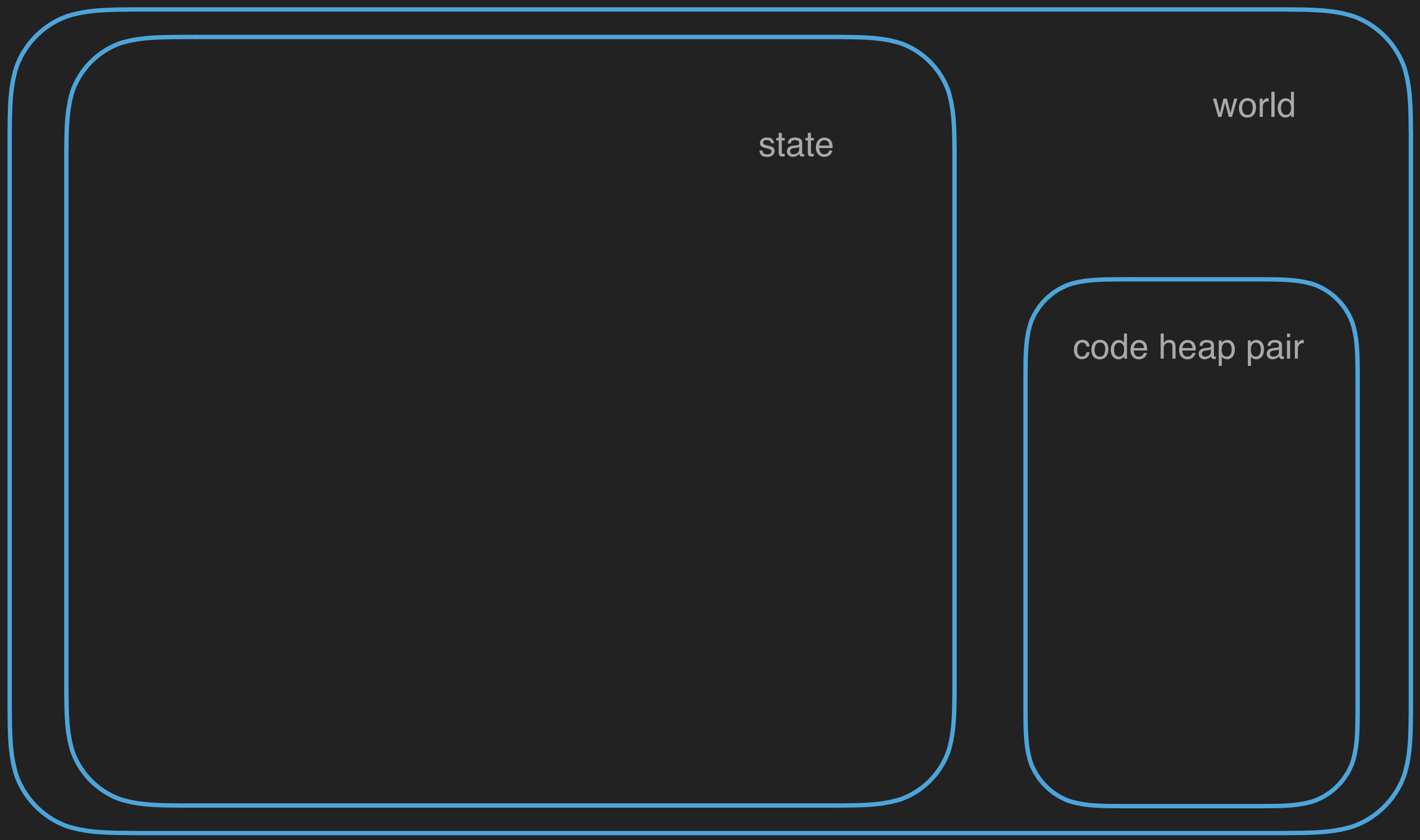
```
Inductive AddrExp: Type :=
 | Ao: OpExp-> AddrExp
 | Aro: GenReg -> OpExp -> AddrExp.

Inductive TrapExp: Type :=
 | Tr: GenReg -> TrapExp
 | Trr: GenReg -> GenReg -> TrapExp
 | Trw: GenReg -> Word -> TrapExp
 | Tw: Word -> TrapExp.

Inductive OpExp: Type :=
 | Or: GenReg -> OpExp
 | Ow: Word -> OpExp.
…
```

Machine State

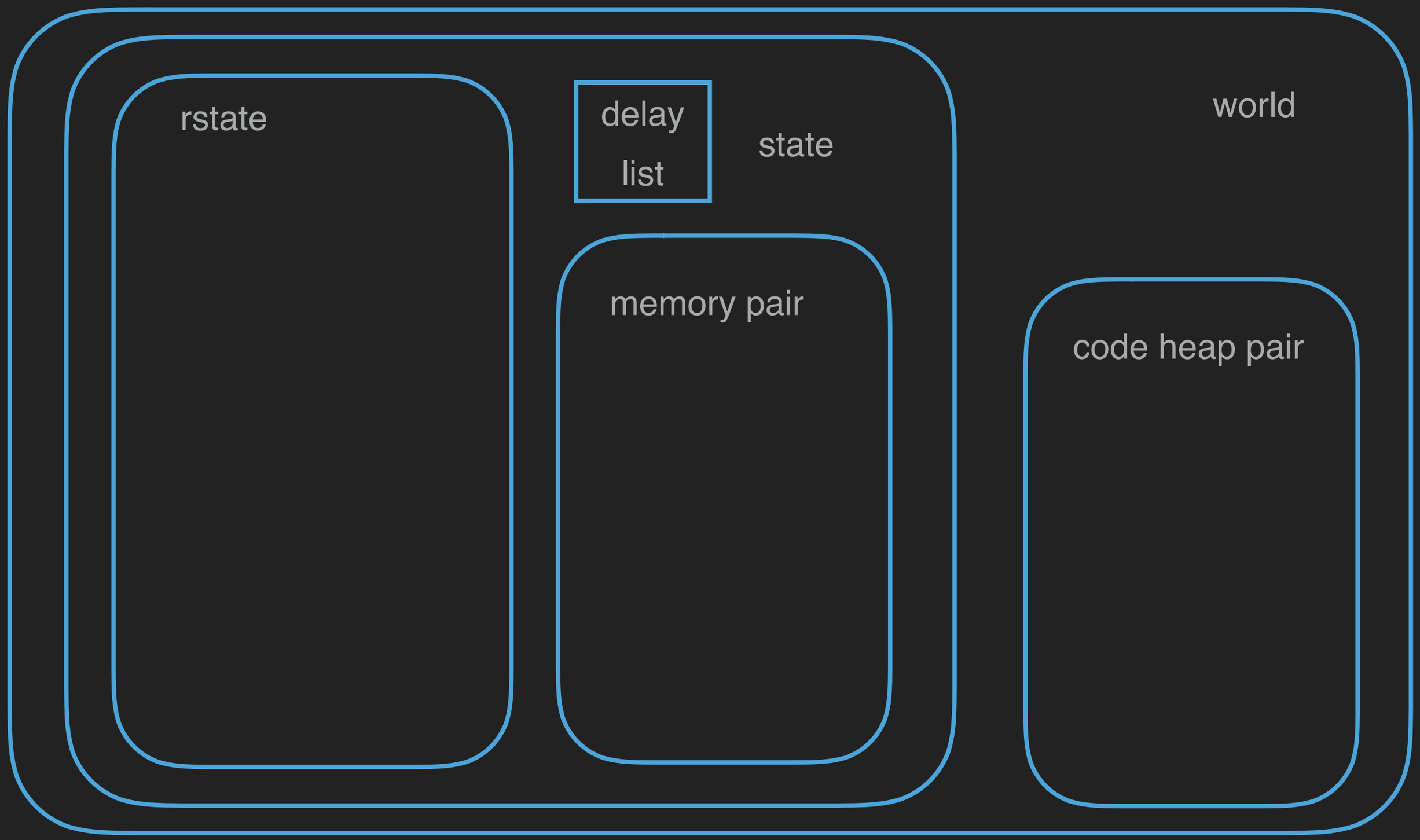
Machine State

world

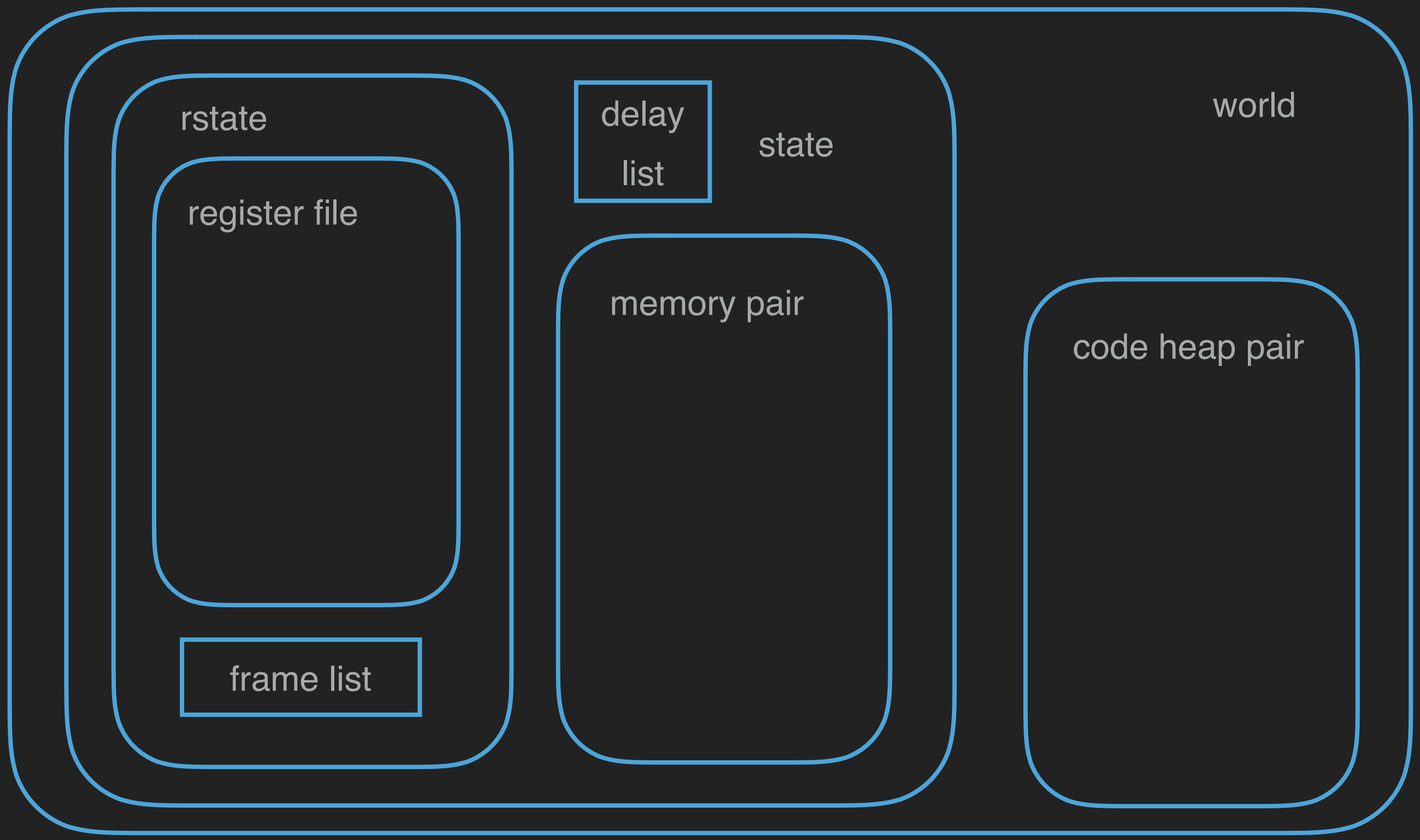
Machine State



Machine State



Machine State



Machine State

world

state

rstate

register file

symbols:

wim, y, psr…

registers:

r0, r1, …

frame list

delay

list

memory pair

code heap pair

Machine State

world

state

rstate

register file

symbols:

wim, y, psr…

registers:

r0, r1, …

frame list

delay

list

memory pair

user

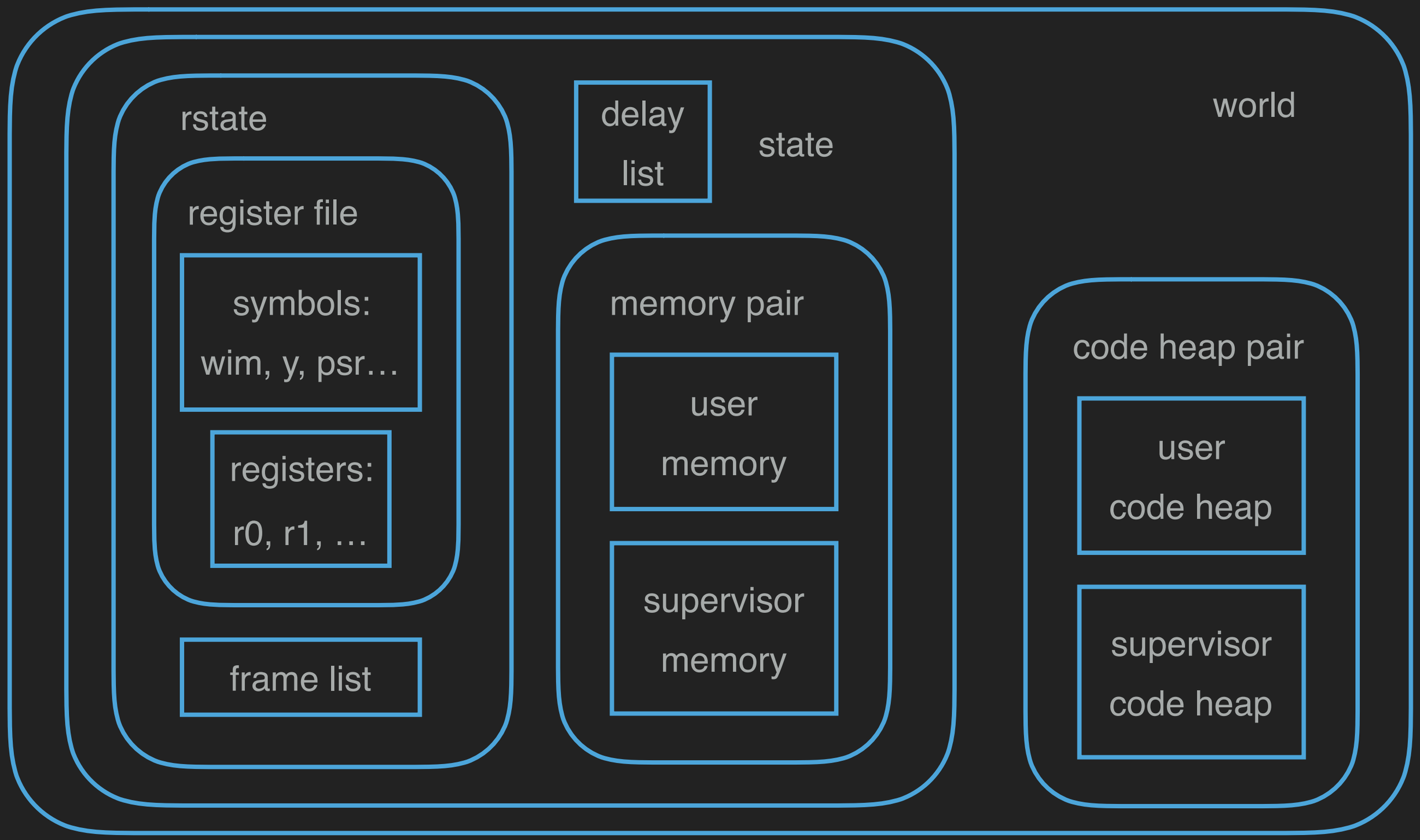
memory

supervisor

memory

code heap pair

Machine State



Machine State

(World) W ::= (Δ, S)

(CodePair) Δ ::= (Cu, Cs)

(CodeHeap) C ∈ Label ⇀ SparcIns

(Label) l ∈ Word

(State) S ::= (Φ, Q, D)

(MemPair) Φ ::= (Mu, Ms)

(Memory) M ∈ Address ⇀ Word

(Address) a ∈ Word

(RState) Q ::= (R, F)

(RegFile) R ∈ RegName → Word

(RegName) q ::= r | ς | pc | npc | κ | τ

(FrameList) F ::= nil | f :: F

(Frame) f ::= [w0, …, w7]

Machine State in Coq

```
Definition RegFile := RegMap.t Word.
Definition Frame: Type := list Word.
Definition FrameList: Type := list Frame.
Definition DelayCycle := nat.
Definition DelayItem: Type := DelayCycle * Symbol * Word.
Definition DelayList: Type := list DelayItem.
Definition RState: Type := RegFile * FrameList.
Definition Memory := WordMap.t (option Word).
Definition CodeHeap := WordMap.t (option SparcIns).
Definition CodePair: Type := CodeHeap * CodeHeap.
Definition MemPair: Type := Memory * Memory.
Definition State: Type := MemPair * RState * DelayList.
Definition World: Type := CodePair * State.
```

Operational Semantics

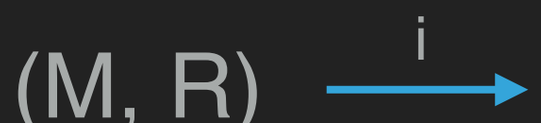
Operational Semantics

Operational Semantics

$(M, R) \xrightarrow{i} (M', R')$

Simple Instructions

Operational Semantics



similar to x86

Simple Instructions

Operational Semantics

similar to x86

$(M, R) \xrightarrow{i} (M', R')$ Simple Instructions

$(M, Q, D) \xrightarrow{i}_{\circ} (M', Q', D')$ Window Registers and Delayed Write

Operational Semantics

similar to x86

$(M, R) \xrightarrow{i} (M', R')$ Simple Instructions

$(M, Q, D) \xrightarrow{i}_{\circ} (M', Q', D')$ Window Registers and Delayed Write

$C \vdash (M, Q, D) \bullet\!\!\rightarrow (M', Q', D')$ Executing Delay and Handling Annulling Flag

Operational Semantics

similar to x86

$(M, R) \xrightarrow{i} (M', R')$ Simple Instructions

$(M, Q, D) \xrightarrow{i}_{\circ} (M', Q', D')$ Window Registers and Delayed Write

$C \vdash (M, Q, D) \xrightarrow{\bullet} (M', Q', D')$ Executing Delay and Handling Annulling Flag

$\Delta \vdash S \xRightarrow{e} S'$ Interrupts, Traps and Mode Switch

Operational Semantics

Operational Semantics

save the value of pc to register r_d

$$\frac{[[\beta]]_R = w \quad \text{word\_aligned}(w) \quad \text{save\_pc}(r_d, R) = R'}{(M, R) \xrightarrow{\text{jmpl } \beta\ r_d} (M, \text{djmp}(w, R'))} \text{JMPL}$$

```
…
call sum3
…
```

Operational Semantics

save the value of pc to register r_d

$$\frac{[[\beta]]_R = w \quad \text{word\_aligned}(w) \quad \text{save\_pc}(r_d, R) = R'}{(M, R) \xrightarrow{\text{jmpl } \beta\ r_d} (M, \text{djmp}(w, R'))}\ \text{JMPL}$$

```
…
call sum3
…
```

$$\frac{\text{dec\_win}(R,F) = (R',F') \quad [[\alpha]]_R = a \quad R'' = R'\{r_d \text{ —> } [[r_s]]_R + a\}}{(M, (R, F), D) \xrightarrow{\text{save } r_s\ \alpha\ r_d} (M', (\text{next}(R''), F'), D)}\ \text{SAVE}$$

Operational Semantics

save the value of pc to register r_d

$$\frac{[[\beta]]_R = w \quad \text{word\_aligned}(w) \quad \text{save\_pc}(r_d, R) = R'}{(M, R) \xrightarrow{\text{jmpl } \beta\ r_d} (M, \text{djmp}(w, R'))}\ \text{JMPL}$$

```
…
call sum3
…
```

$$\frac{\text{dec\_win}(R,F) = (R',F') \quad [[\alpha]]_R = a \quad R'' = R'\{r_d \text{ —> } [[r_s]]_R + a\}}{(M, (R, F), D) \xrightarrow{\text{save } r_s\ \alpha\ r_d} (M', (\text{next}(R''), F'), D)}\ \text{SAVE}$$

```
…
save  %sp, -64, %sp
…
```

Operational Semantics

save the value of pc to register r_d

$$\frac{[[\beta]]_R = w \quad \text{word\_aligned}(w) \quad \text{save\_pc}(r_d, R) = R'}{(M, R) \xrightarrow{\text{jmpl } \beta\ r_d} (M, \text{djmp}(w, R'))}\ \text{JMPL}$$

```
…
call sum3
…
```

$$\frac{\text{dec\_win}(R,F) = (R',F') \quad [[\alpha]]_R = a \quad R'' = R'\{r_d \text{ —> } [[r_s]]_R + a\}}{(M, (R, F), D) \xrightarrow{\text{save } r_s\ \alpha\ r_d} (M', (\text{next}(R''), F'), D)}\ \text{SAVE}$$

```
…
save  %sp, -64, %sp
…
```

Operational Semantics

save the value of pc to register r_d

$$\frac{[[\beta]]_R = w \quad \text{word\_aligned}(w) \quad \text{save\_pc}(r_d, R) = R'}{(M, R) \xrightarrow{\text{jmpl } \beta\ r_d} (M, \text{djmp}(w, R'))}\ \text{JMPL}$$

```
…
call sum3
…
```

$$\frac{\text{dec\_win}(R,F) = (R',F') \quad [[\alpha]]_R = a \quad R'' = R'\{r_d \text{ —> } [[r_s]]_R + a\}}{(M, (R, F), D) \xrightarrow{\text{save } r_s\ \alpha\ r_d} (M', (\text{next}(R''), F'), D)}\ \text{SAVE}$$

Others can be found in our paper or technical report.

```
…
save  %sp, -64, %sp
…
```

Operational Semantics in Coq

```
Inductive ArrowB:
  Memory * RState * DelayList ->
  SparcIns ->
  Memory * RState * DelayList ->
  Prop :=
| MR:
    forall i M R M' R' F D,
    ArrowA (M,R) i (M',R') ->
    ArrowB (M,(R,F),D) i (M',(R',F),D)
| Save:
    forall ri o rj i a M R R' R'' F F' D,
    i = save ri o rj ->
    dec_win (R,F) = Some (R',F') ->
    eval_OpExp o R = Some a ->
    R'' = R'#rj <- ((R#ri) +i a) ->
    ArrowB (M,(R,F),D) i (M,(next R'',F'),D)
| Restore:
    forall ri o rj i a M R R' R'' F F' D,
    i = restore ri o rj ->
    inc_win (R,F) = Some (R',F') ->
    eval_OpExp o R = Some a ->
    R'' = R'#rj <- ((R#ri) +i a) ->
    ArrowB (M,(R,F),D) i (M,(next R'',F'),D)
| Rett:
    forall addr i w M R R' F F' D,
    i = rett addr ->
    trap_disabled_R R ->
    sup_mode_R R ->
    eval_AddrExp addr R = Some w ->
    word_aligned_R w ->
    rett_f (R,F) = Some (R',F') ->
    ArrowB (M,(R,F),D) i (M,(djmp w R',F'),D)
```

Outline

- Modeling SPARCv8 ISA
  - Syntax
  - Operational Semantics
  - Machine State

Outline

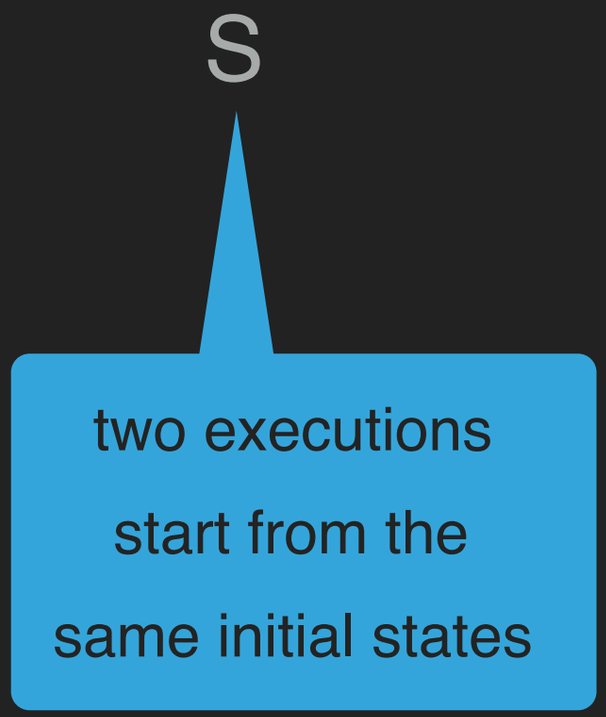
- Modeling SPARCv8 ISA
  - Syntax
  - Operational Semantics
  - Machine State
- Properties of the Model
  - Determinacy Property
  - Isolation Property

Properties

Properties

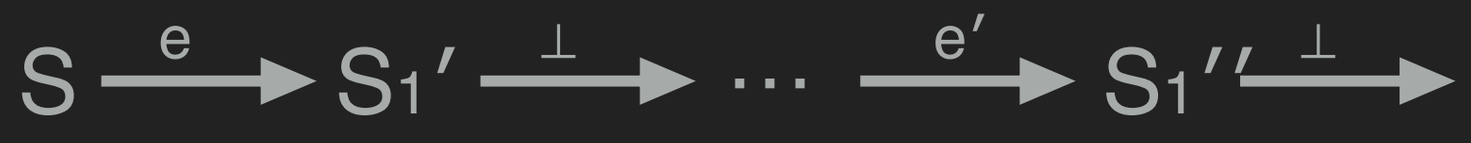
Theorem 1 (Determinacy)

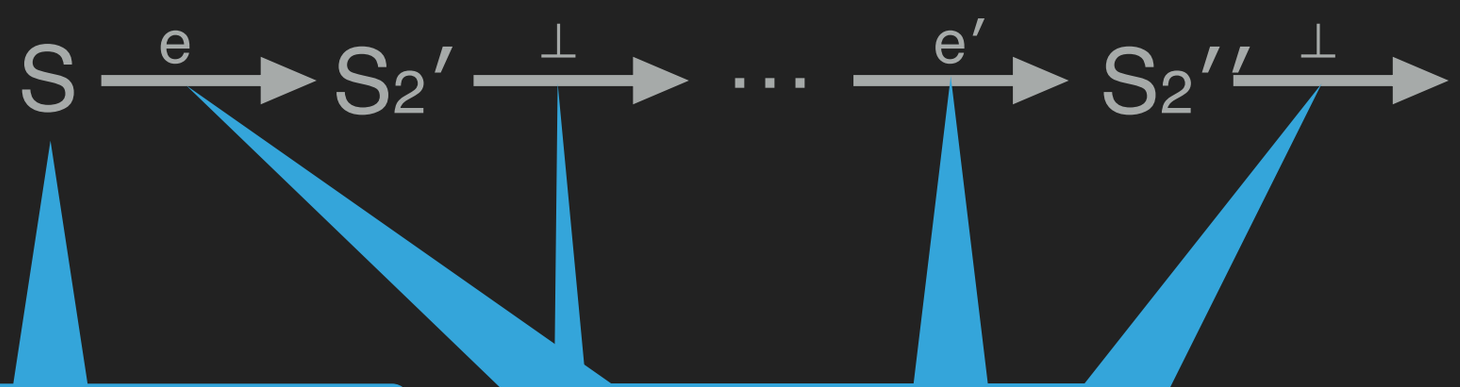
S



Properties

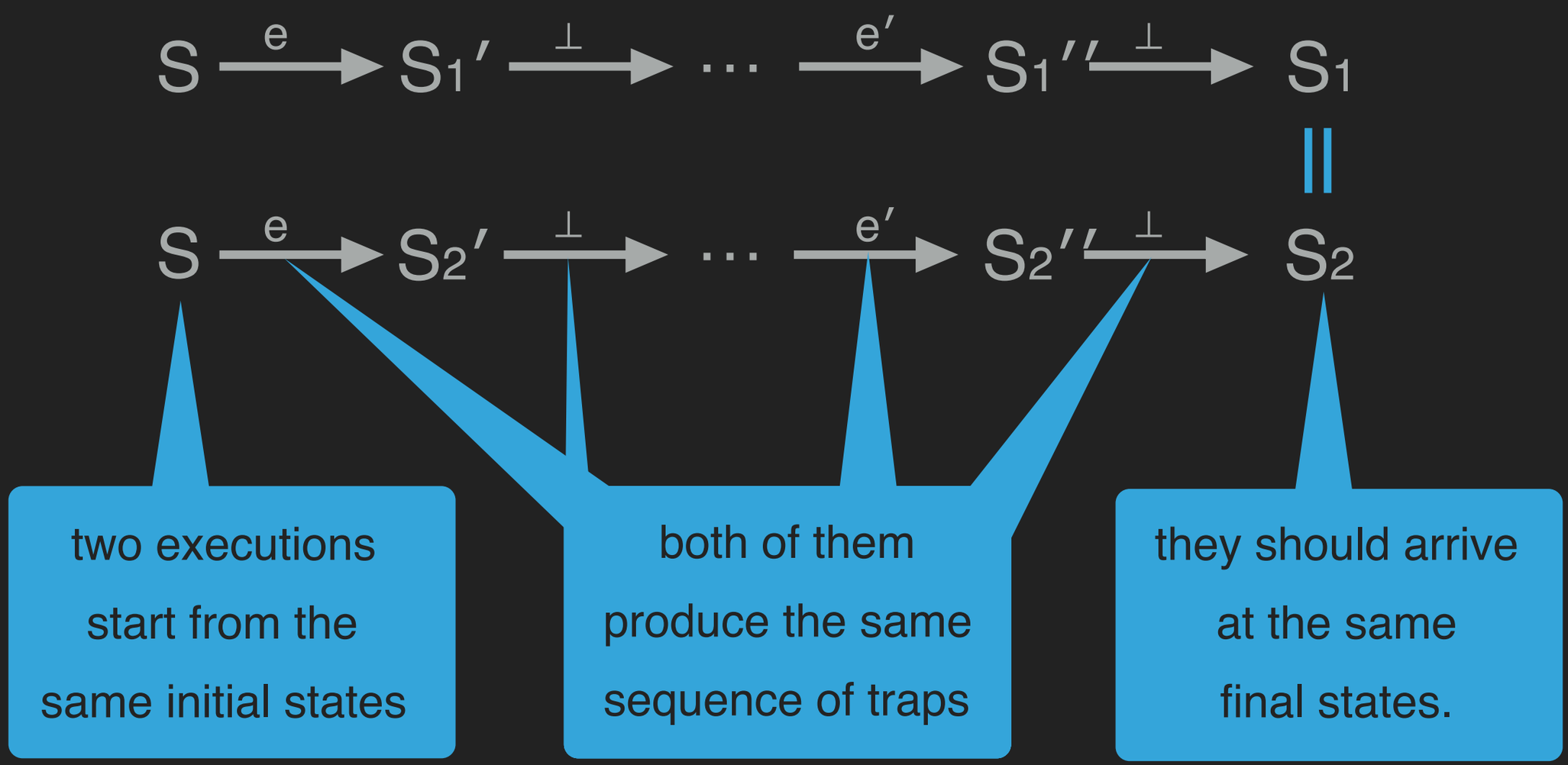
Theorem 1 (Determinacy)





Properties

Theorem 1 (Determinacy)



Properties

Theorem 1 (Determinacy)

$$S \xrightarrow{e} S_1' \xrightarrow{\perp} \cdots \xrightarrow{e'} S_1'' \xrightarrow{\perp} S_1 \quad \| \quad S \xrightarrow{e} S_2' \xrightarrow{\perp} \cdots \xrightarrow{e'} S_2'' \xrightarrow{\perp} S_2$$

If $\Delta \vdash S \xrightarrow{E}{}^{*} S_1$, $\Delta \vdash S \xrightarrow{E}{}^{*} S_2$, then $S_1 = S_2$.

where $\Delta \vdash S \xrightarrow{E}{}^{*} S'$ is defined as:

$\exists n, \Delta \vdash S \xrightarrow{E}{}^{n} S'$.

Properties

Properties

Theorem 2 (In User Mode)

If $\Delta \vdash S \longrightarrow^{n} S'$, then usr_mode(S′).

where $\Delta \vdash S \longrightarrow^{n} S'$ …

Properties

Theorem 2 (In User Mode)

If $\Delta \vdash S \longrightarrow^{n} S'$, then usr_mode(S′).

where $\Delta \vdash S \longrightarrow^{n} S'$ …

the conditions we need to maintain to keep the system in the user mode

Properties

shows the sufficiency of that definition

Theorem 2 (In User Mode)

If $\Delta \vdash S \longrightarrow^{n} S'$, then usr_mode(S′).

where $\Delta \vdash S \longrightarrow^{n} S'$ …

the conditions we need to maintain to keep the system in the user mode

Properties

Properties

Theorem 3 (Write Isolation)

If $\Delta \vdash S \xrightarrow{\bullet}{}^{n} S'$, then sup_part_eq(S′,S′).

Properties

Theorem 3 (Write Isolation)

If $\Delta \vdash S \xrightarrow{}{}^{n} S'$, then sup_part_eq(S′,S′).

The system does not modify
the resource that belongs
to the supervisor mode

Properties

Properties

Theorem 4 (Read Isolation)

If usr_code_eq(Δ_1, Δ_2), usr_state_eq(S_1, S_2),
$\Delta_1 \vdash S_1 \xrightarrow{\bullet}{}^{n} S_1'$, $\Delta_2 \vdash S_2 \xrightarrow{\bullet}{}^{n} S_2'$, then
usr_state_eq(S_1', S_2').

Properties

Theorem 4 (Read Isolation)

If usr_code_eq(Δ_1, Δ_2), usr_state_eq(S_1, S_2),
$\Delta_1 \vdash S_1 \xrightarrow{\bullet}{}^{n} S_1'$, $\Delta_2 \vdash S_2 \xrightarrow{\bullet}{}^{n} S_2'$, then
usr_state_eq(S_1', S_2').

The system is only affected by part of the state

Properties in Coq

```
Theorem Determinacy:
  forall n E CP S S1 S2,
  ArrowT CP S E n S1 ->
  ArrowT CP S E n S2 ->
  S1 = S2.

Theorem InUserMode:
  forall CP S n S',
    ArrowWR CP S n S' ->
    usr_mode_S S’.
```

```
Theorem WriteIsolation:
  forall CP S n S',
    ArrowWR CP S n S' ->
    sup_mem_eq S S'.

Theorem ReadIsolation:
  forall n CP1 CP2 S1 S1' S2 S2',
    usr_code_eq CP1 CP2 ∧
    low_eq S1 S2 ->
    ArrowWR CP1 S1 n S1' ∧
    ArrowWR CP2 S2 n S2' ->
    low_eq S1' S2'.
```

Outline

- Modeling SPARCv8 ISA
  - Syntax
  - Operational Semantics
  - Machine State
- Properties of the Model
  - Determinacy Property
  - Isolation Property

Outline

- Modeling SPARCv8 ISA
  - Syntax
  - Operational Semantics
  - Machine State
- Properties of the Model
  - Determinacy Property
  - Isolation Property
- Verifying a Window Overflow Trap Handler

Verifying a Window Overflow Trap Handler

Verifying a Window Overflow Trap Handler

```
window overflow:
   mov  %wim,  %l3
   mov   %g1,  %l7
   srl   %l3,   1,  %g1
   sll   %l3,  N-1, %l4
   or    %l4,  %g1, %g1
   save
   mov   %g1, %wim
   nop
   nop
   nop
   st    %l0, [%sp+0]
   ...
   st    %i7, [%sp+60]
   restore
   mov   %l7, %g1
   jmp   %l1
   rett  %l2
```

Verifying a Window Overflow Trap Handler

```
window overflow:
   mov  %wim,  %l3
   mov   %g1,  %l7
   srl   %l3,   1,  %g1
   sll   %l3,  N-1, %l4
   or    %l4,  %g1, %g1
   save
   mov   %g1, %wim
   nop
   nop
   nop
   st    %l0, [%sp+0]
   ...
   st    %i7, [%sp+60]
   restore
   mov   %l7, %g1
   jmp   %l1
   rett  %l2
```

- overflow_pre_cond (W) ==

  next window is not available ∧ …

- overflow_post_cond (W) ==

  next window is available ∧ …

Verifying a Window Overflow Trap Handler

```
window overflow:
   mov  %wim,  %l3
   mov   %g1,  %l7
   srl   %l3,   1,  %g1
   sll   %l3,  N-1, %l4
   or    %l4,  %g1, %g1
   save
   mov   %g1, %wim
   nop
   nop
   nop
   st    %l0, [%sp+0]
   ...
   st    %i7, [%sp+60]
   restore
   mov   %l7, %g1
   jmp   %l1
   rett  %l2
```

- overflow_pre_cond (W) ==

  next window is not available $\wedge$ …

- overflow_post_cond (W) ==

  next window is available $\wedge$ …

Verifying a Window Overflow Trap Handler

```
window overflow:
   mov  %wim,  %l3
   mov   %g1,  %l7
   srl   %l3,   1,  %g1
   sll   %l3,  N-1, %l4
   or    %l4,  %g1, %g1
   save
   mov   %g1, %wim
   nop
   nop
   nop
   st    %l0, [%sp+0]
   ...
   st    %i7, [%sp+60]
   restore
   mov   %l7, %g1
   jmp   %l1
   rett  %l2
```

- overflow_pre_cond (W) ==

  next window is not available ∧ …

- overflow_post_cond (W) ==

  next window is available ∧ …

- Theorem 5 (Correctness of the Window Overflow Trap Handler)

  If overflow_pre_cond(Δ, S), then forall S’ and E, if $\Delta \vdash S \bullet\!\!\rightarrow^{30} S'$, then overflow_post_cond(Δ, S′) and no_trap_event(E).

Verifying a Window Overflow Trap Handler

```
window overflow:
   mov  %wim,  %l3
   mov   %g1,  %l7
   srl   %l3,   1,  %g1
   sll   %l3,  N-1, %l4
   or    %l4,  %g1, %g1
   save
   mov   %g1, %wim
   nop
   nop
   nop
   st    %l0, [%sp+0]
   ...
   st    %i7, [%sp+60]
   restore
   mov   %l7, %g1
   jmp   %l1
   rett  %l2
```

- overflow_pre_cond (W) ==

  next window is not available ∧ …

- overflow_post_cond (W) ==

  next window is available ∧ …

- Theorem 5 (Correctness of the Window Overflow Trap Handler)

  If overflow_pre_cond(Δ, S), then forall S' and E, if $\Delta \vdash S \bullet\!\!\rightarrow^{30} S'$, then overflow_post_cond(Δ, S′) and no_trap_event(E).

Details can be found in our paper or technical report.

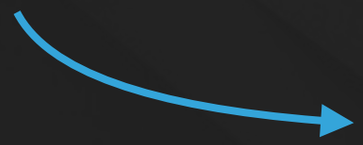
Summary

Summary

SPARCv8 ISA

Summary

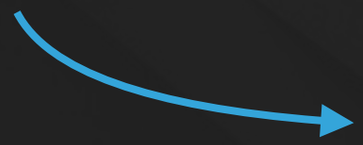
SPARCv8 ISA



Determinacy &
Isolation Properties

Summary

SPARCv8 ISA

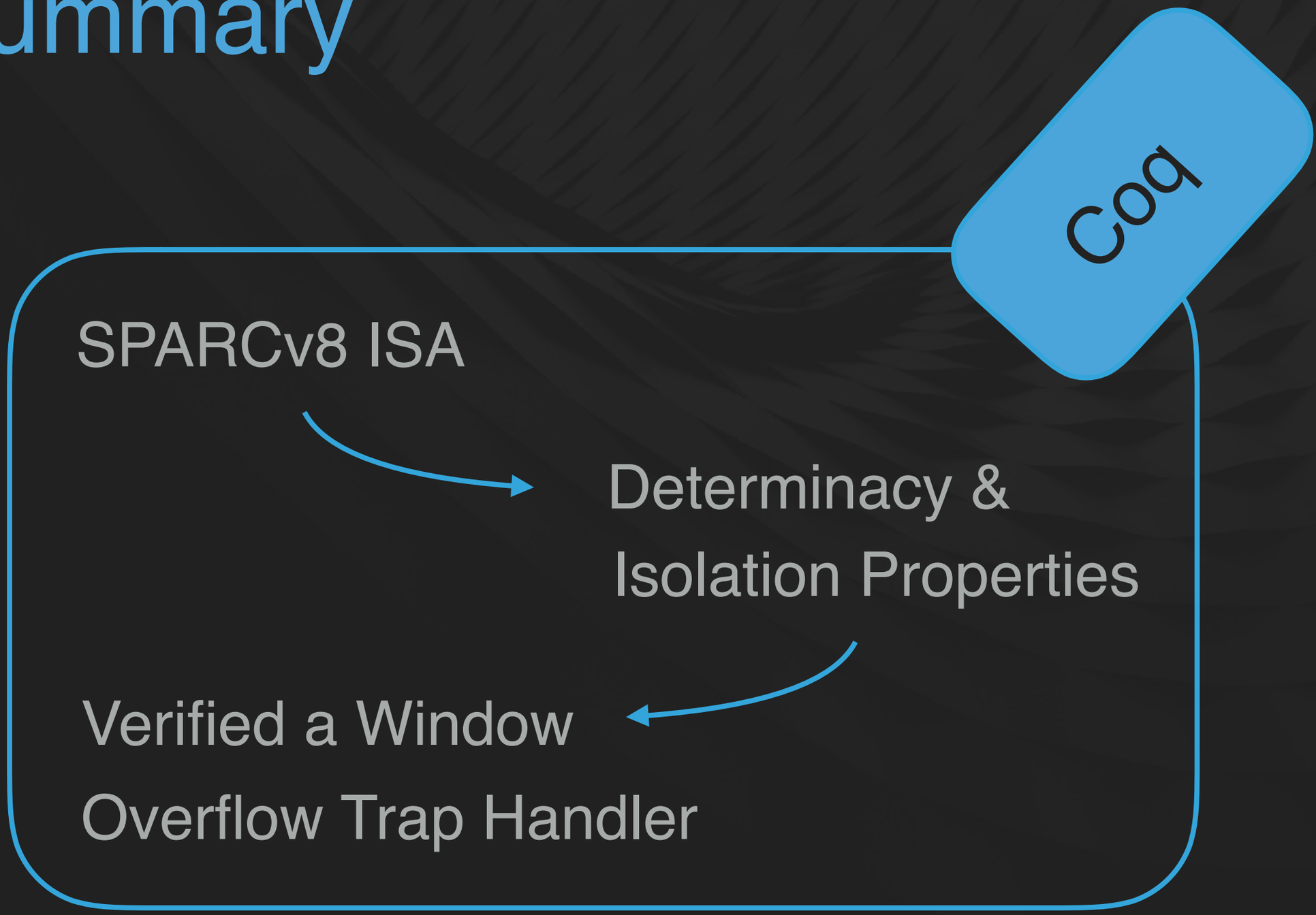


Determinacy &
Isolation Properties

Verified a Window

Overflow Trap Handler

Summary


Coq
SPARCv8 ISA
Determinacy &
Isolation Properties
Verified a Window
Overflow Trap Handler

Thank you !