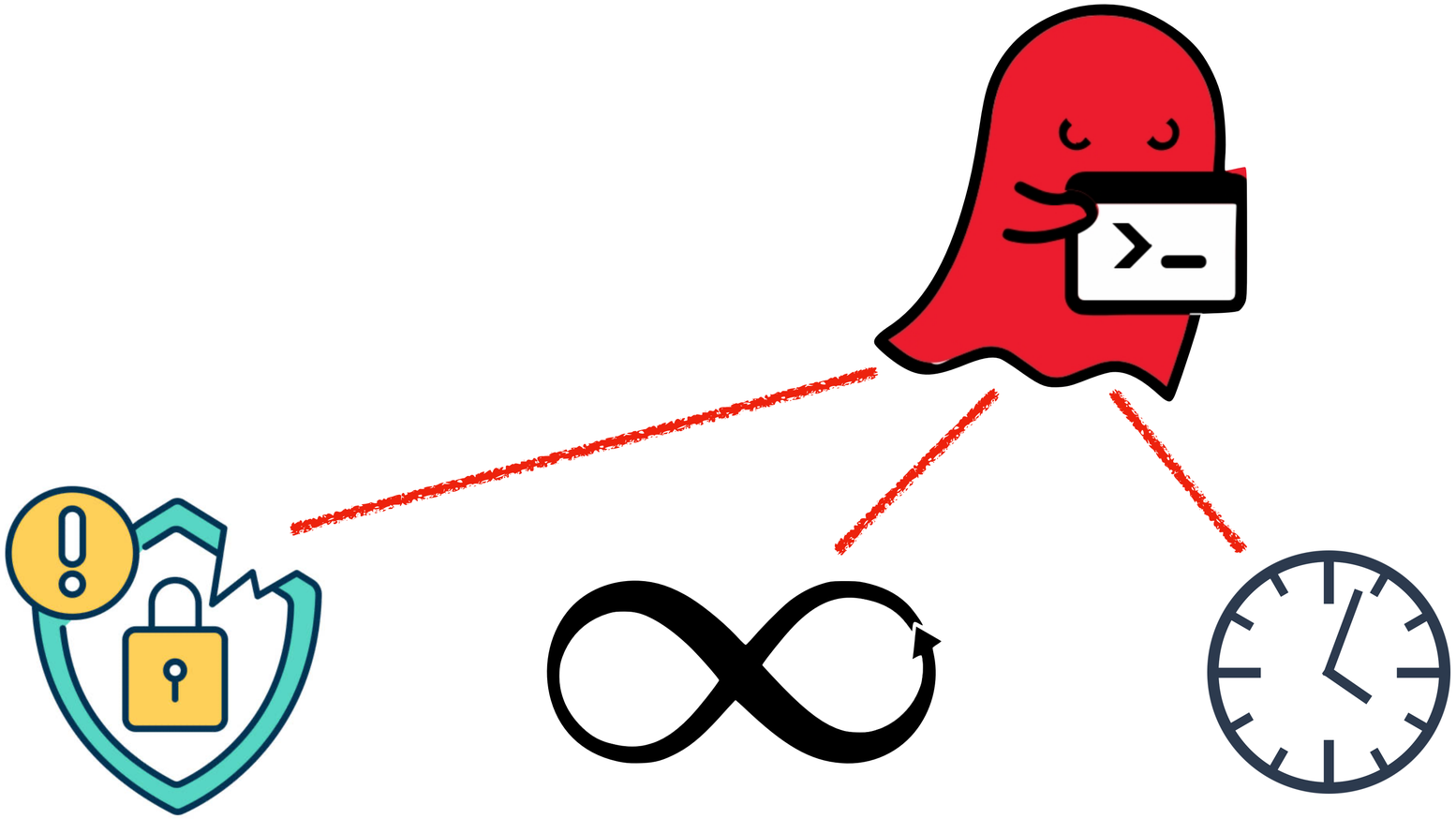
Intermittent Computing

15-316 Foundations of Security and Privacy

Myra Dotzel, 12/3/24

Previously in 15-316…

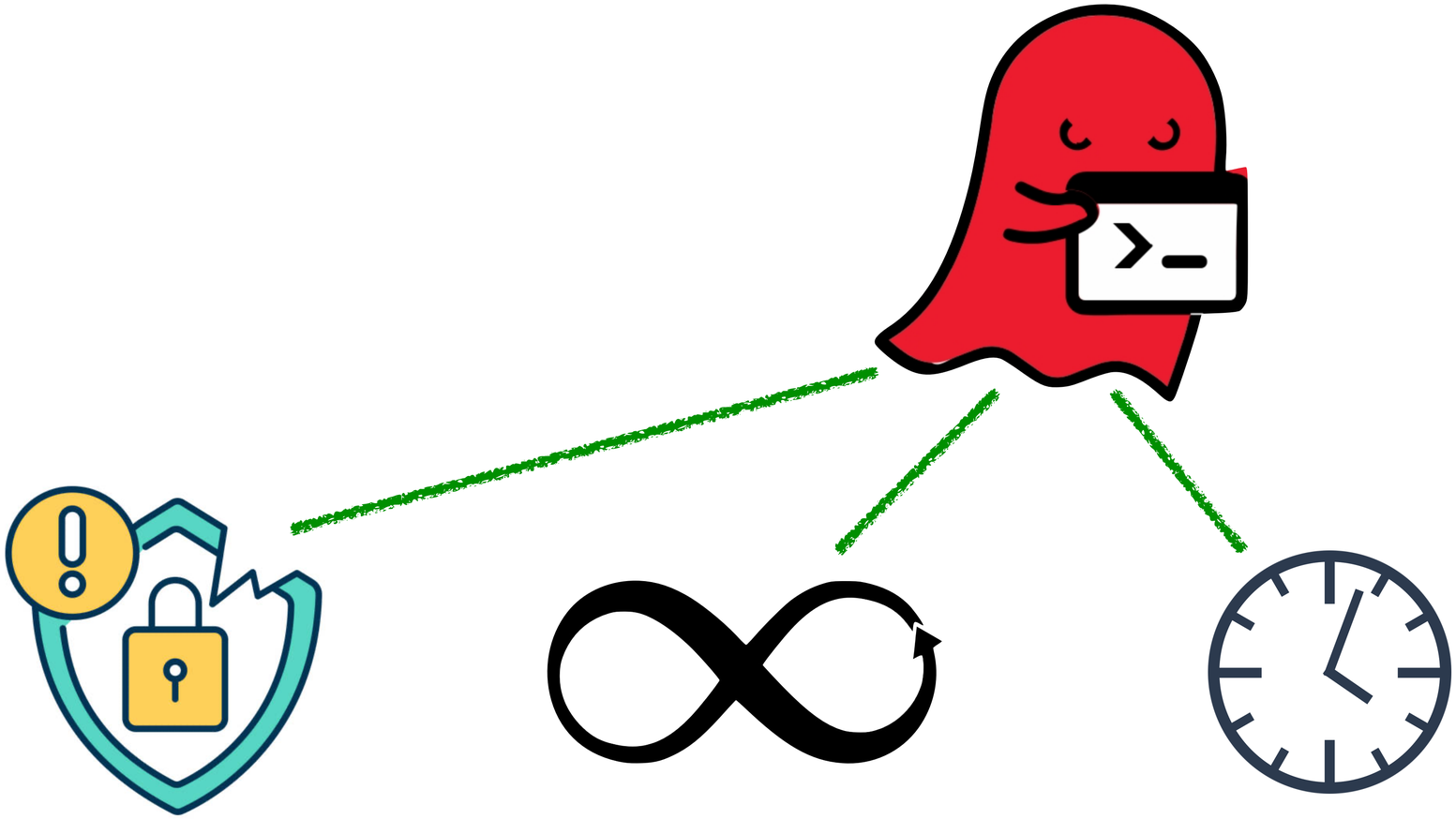


security leaks

termination

timing attacks

Previously in 15-316…

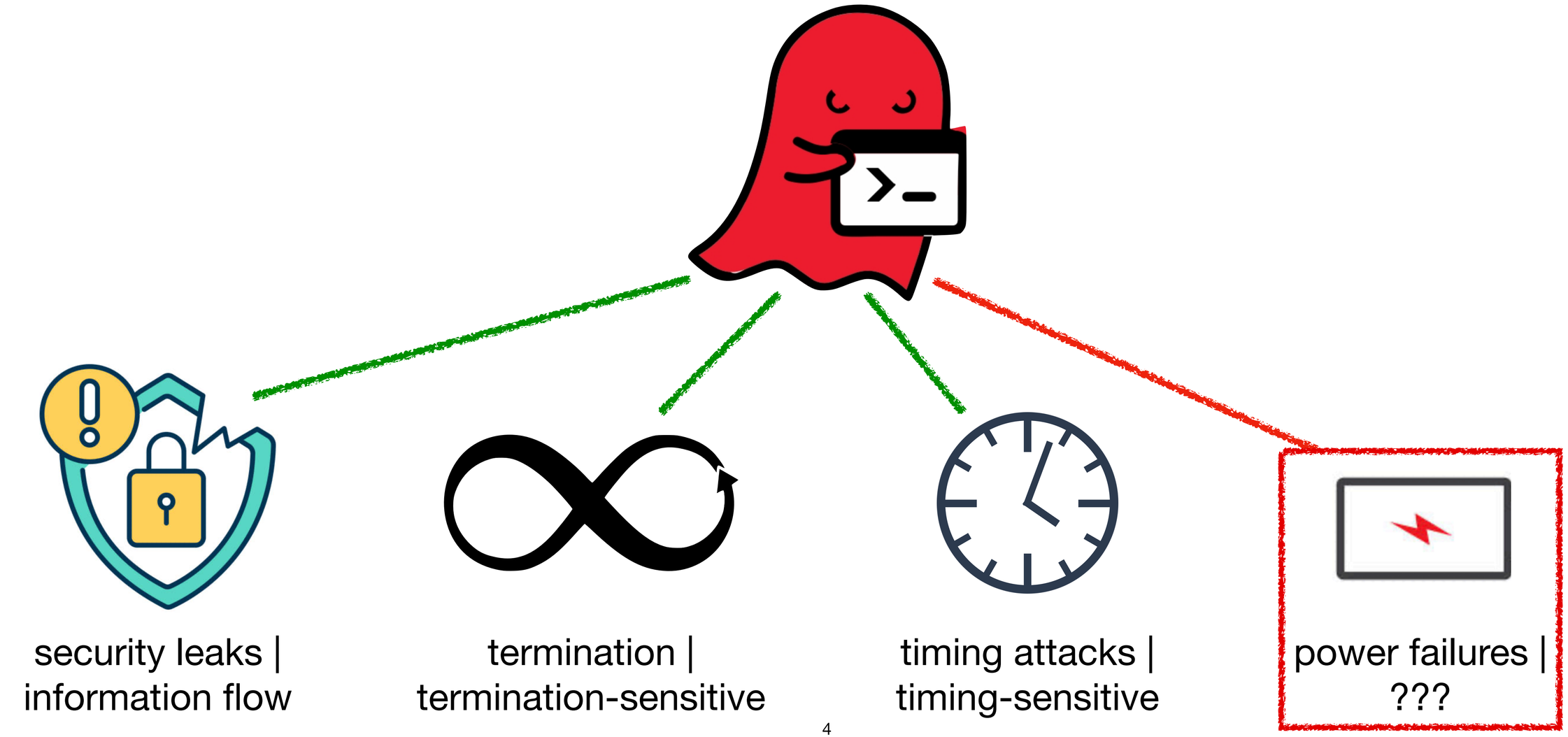


security leaks |
information flow

termination |
termination-sensitive

timing attacks |
timing-sensitive

Today: Power failures as an attack!



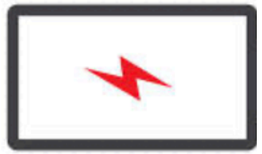
security leaks |
information flow

termination |
termination-sensitive

timing attacks |
timing-sensitive

power failures |
???

Today: Power failures as an attack!



power failures |
intermittent systems

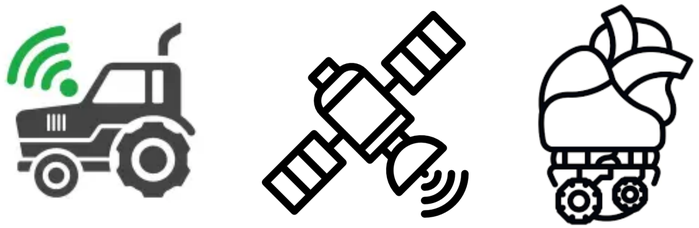
- nondeterministic, frequent, arbitrary
- all code susceptible
- must provision our systems to be robust against them

Today: Intermittent Computing

1. **intermittent computing + correctness**
   [Surbatovich et al. ’20]
2. correctness as a noninterference property
   [Surbatovich et al. ’23]

Intermittent Computing

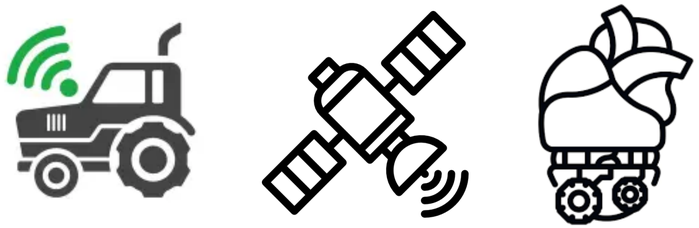
- tiny devices that compute in inaccessible environments



```
Ckpt
    x := y;
    y := z;
    w := x + y
```

Intermittent Computing

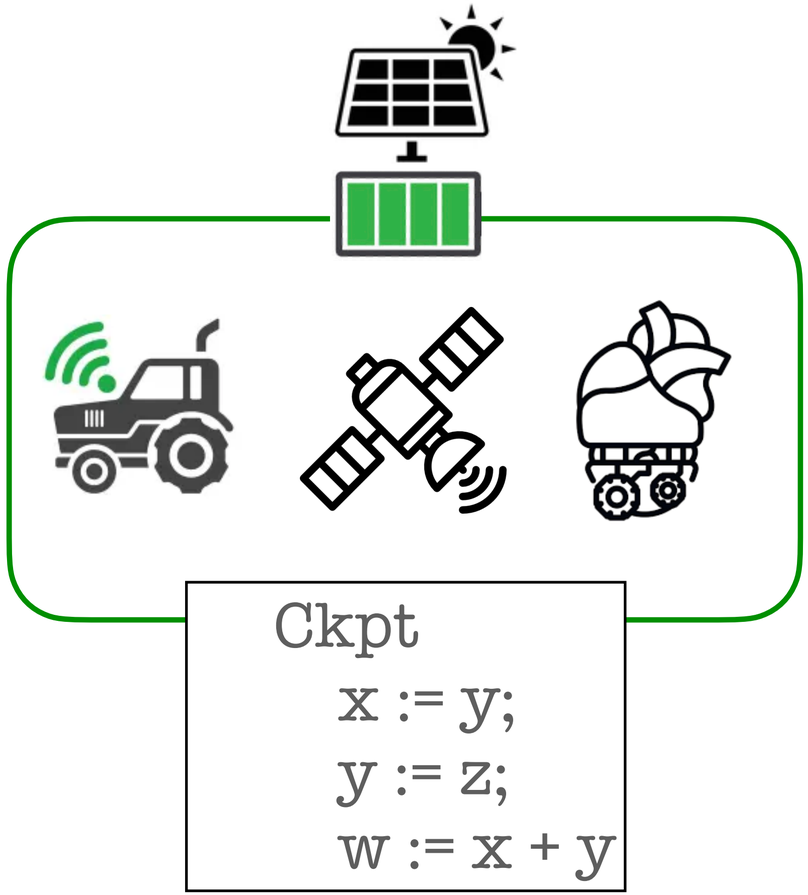
- tiny devices that compute in inaccessible environments
- cannot rely on continuous energy sources



```
Ckpt
    x := y;
    y := z;
    w := x + y
```

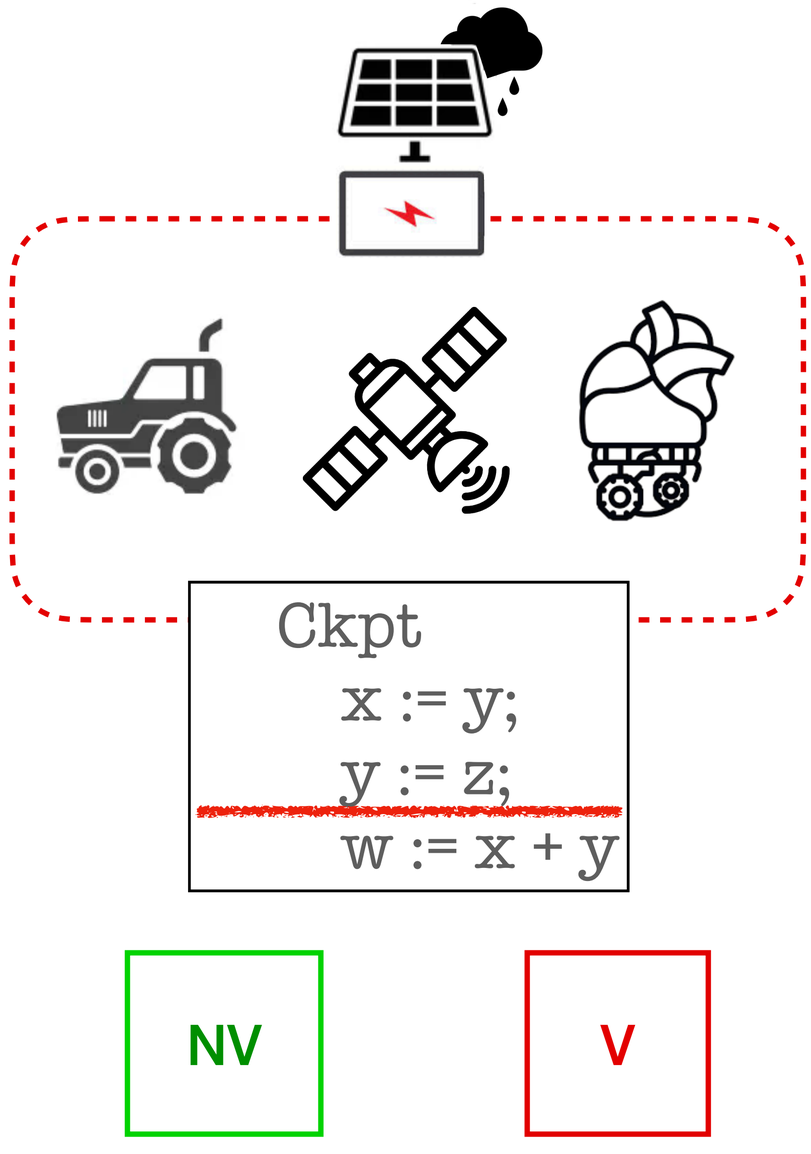
Intermittent Computing

- tiny devices that compute in inaccessible environments
- cannot rely on continuous energy sources
- charge, compute, and crash frequently

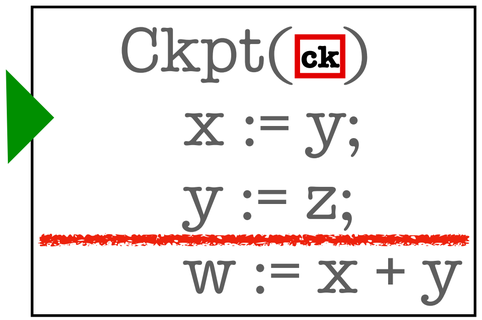


Intermittent Computing

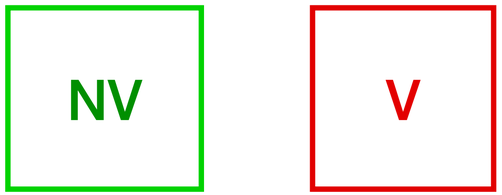
- tiny devices that compute in inaccessible environments
- cannot rely on continuous energy sources
- charge, compute, and crash frequently
- rely on a combination of NV and V
- NV survives power failures, V does not
- runtime system support

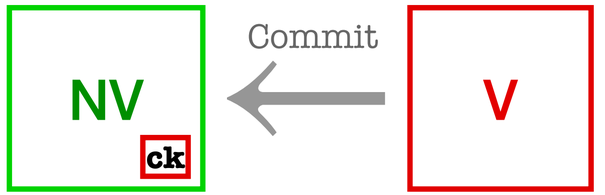


Atomic regions checkpoint ahead of time

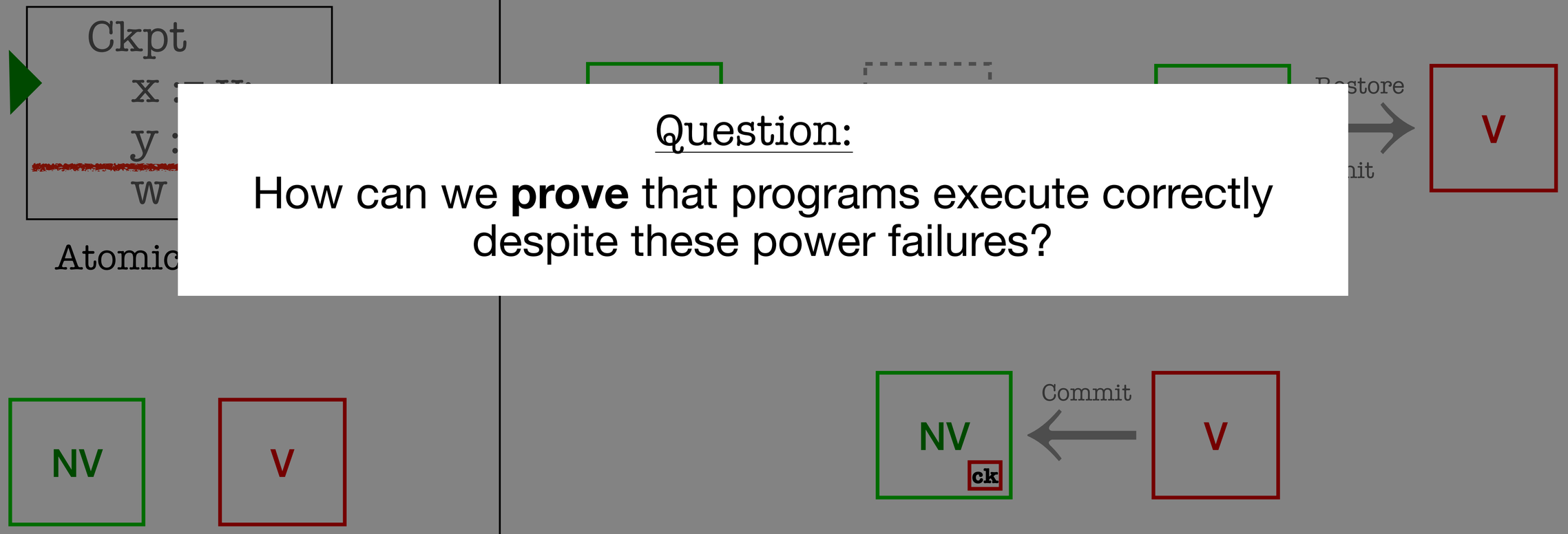


Atomic region





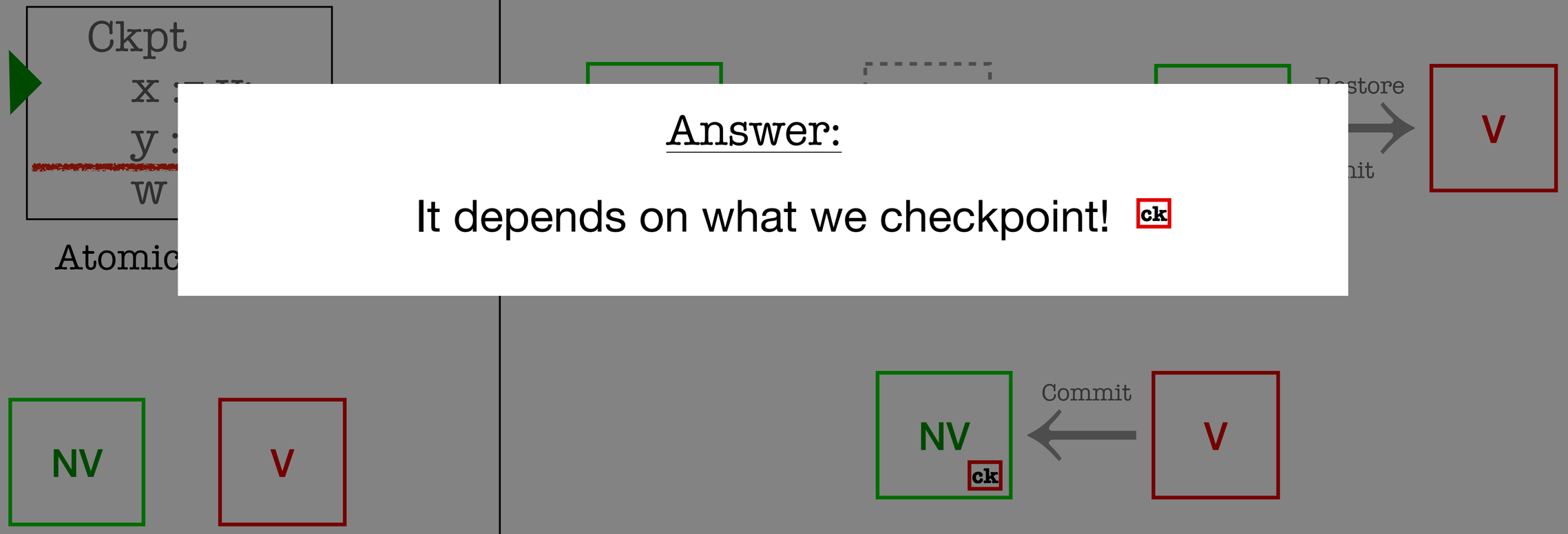
Atomic regions checkpoint ahead of time



Question:

How can we **prove** that programs execute correctly despite these power failures?

Atomic regions checkpoint ahead of time

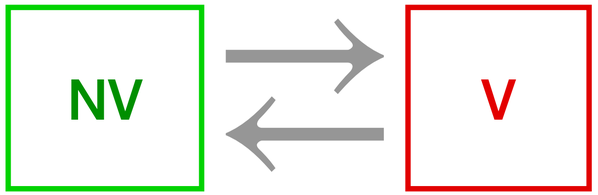


Answer:

It depends on what we checkpoint! ck

What do we need to checkpoint?

```
Ckpt
    x := y;
    y := z;
    w := x + y
```



Checkpoint nothing

```
1 Ckpt( )
2       x := y;
3       y := z;
4       w := x + y
```

NV	x	y	z	w
1	0	1	2	0
2	**1**	1	2	0
3	**1**	**2**	2	0

NV	x	y	z	w
1	0	1	2	0
2	**1**	1	2	0
3	**1**	**2**	2	0
4	**1**	**2**	2	**3**

nothing

Checkpoint nothing

```
1 Ckpt( )
2       x := y;
3       y := z;
4       w := x + y
```

NV	x	y	z	w
1	0	1	2	0
2	1	1	2	0
3	1	2	2	0
Restore	1	2	2	0
2	2	2	2	0
3	2	2	2	0
4	2	2	2	4

NV	x	y	z	w
1	0	1	2	0
2	1	1	2	0
3	1	2	2	0
4	1	2	2	3

inconsistent memory!

nothing

↳ consistency bugs!

Checkpoint everything!

```
1 Ckpt(x,y,w,z)
2       x := y;
3       y := z;
4       w := x + y
```

NV	x	y	z	w
	0	1	2	0

NV	x	y	z	w
1	0	1	2	0
2	**1**	1	2	0
3	**1**	**2**	2	0
4	**1**	**2**	2	**3**

nothing

↬ consistency bugs!

everything

Checkpoint everything!

```
1 Ckpt(x,y,w,z)
2       x := y;
3       y := z;
4       w := x + y
```

NV

x	y	z	w
0	1	2	0

Init

V

	ck x	ck y	ck z	ck w
1	0	1	2	0
2	1	1	2	0
3	1	2	2	0

NV	x	y	z	w
1	0	1	2	0
2	1	1	2	0
3	1	2	2	0
4	1	2	2	3

nothing

↯ consistency bugs!

everything

Checkpoint everything!

```
1 Ckpt(x,y,w,z)
2       x := y;
3       y := z;
4       w := x + y
```

NV

x	y	z	w
0	1	2	0

V

ck x	ck y	ck z	ck w

pwOff

NV	x	y	z	w
1	0	1	2	0
2	**1**	1	2	0
3	**1**	**2**	2	0
4	**1**	**2**	2	**3**

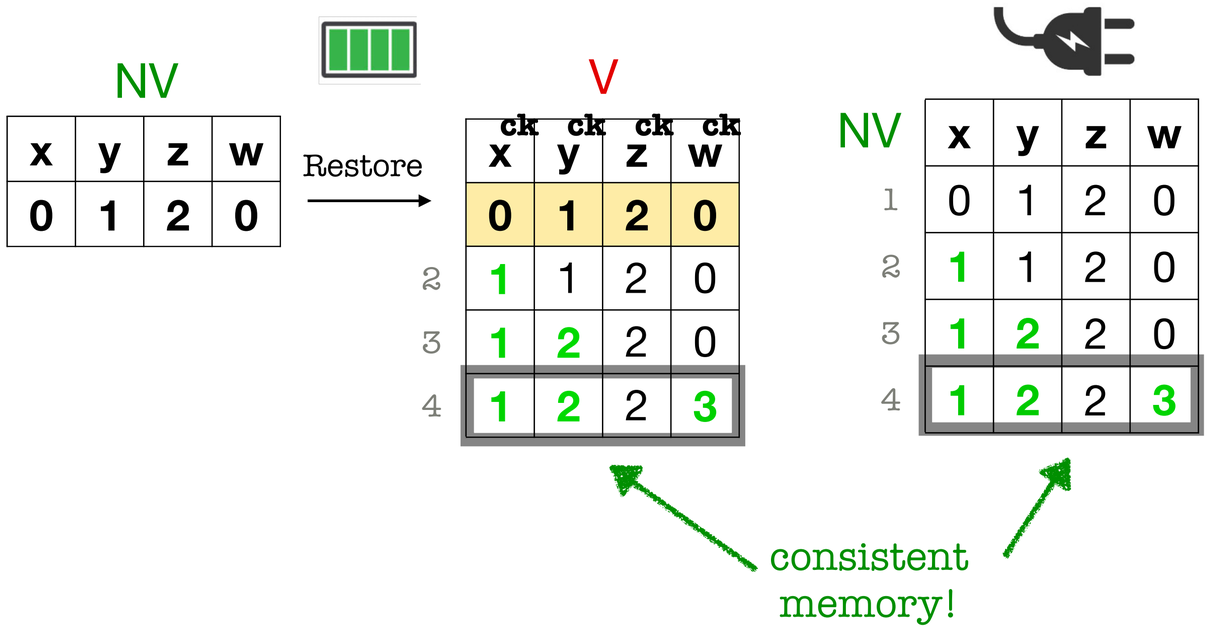
nothing

↬ consistency bugs!

everything

Checkpoint everything!

```
1 Ckpt(x,y,w,z)
2       x := y;
3       y := z;
4       w := x + y
```



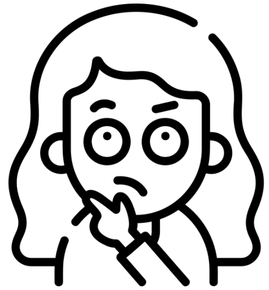
nothing

↬ consistency bugs!

everything

↬ inefficient!

What do we need to checkpoint?



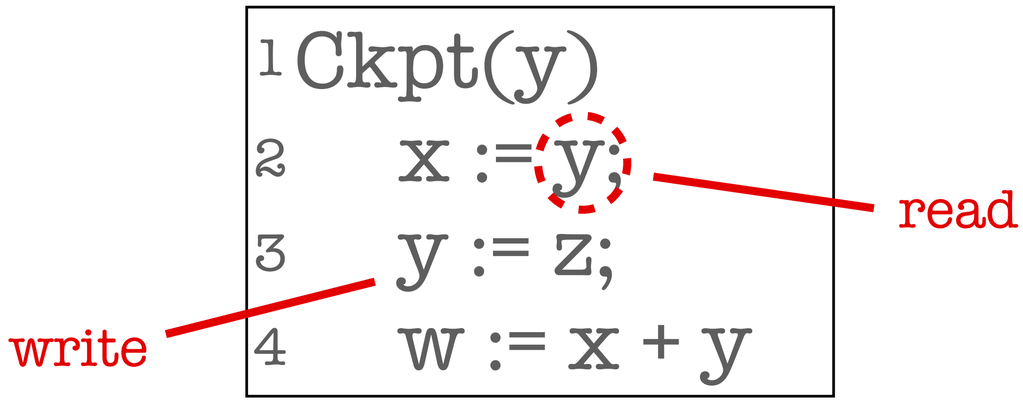
written before read

```
1 Ckpt(???)
2     x := y;
3     y := z;
4     w := x + y
```

z is never updated

safe to read here

Only the write-after-read variables! [Lucia et al. '15]



y is a write-after-read (WAR) variable.

Checkpoint write-after-read variables

```
1 Ckpt(y)
2       x := y;
3       y := z;
4       w := x + y
```

NV

x	y	z	w
0	**1**	2	0

NV	x	y	z	w
1	0	1	2	0
2	**1**	1	2	0
3	**1**	**2**	2	0
4	**1**	**2**	2	**3**

nothing

↬ consistency bugs!

everything

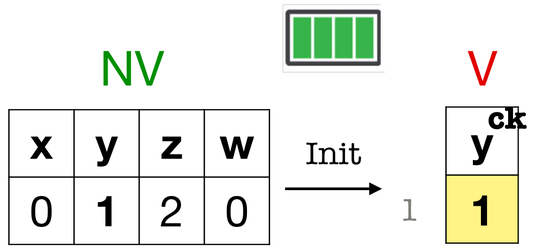
↬ inefficient!

write-after-read variables

✓ real systems

Checkpoint write-after-read variables

```
1 Ckpt(y)
2       x := y;
3       y := z;
4       w := x + y
```



nothing

↬ consistency bugs!

everything

↬ inefficient!

write-after-read variables

✓ real systems

Checkpoint write-after-read variables

```
1 Ckpt(y)
2       x := y;
3       y := z;
4       w := x + y
```

NV

	x	y	z	w
	0	1	2	0
2	1	1	2	0

V

	y (ck)
1	1
2	1

NV

	x	y	z	w
1	0	1	2	0
2	1	1	2	0
3	1	2	2	0
4	1	2	2	3

nothing

↝ consistency bugs!

everything

↝ inefficient!

write-after-read variables

✓ real systems

Checkpoint write-after-read variables

```
1 Ckpt(y)
2       x := y;
3       y := z;
4       w := x + y
```

NV

	x	y	z	w
	0	1	2	0
2	1	1	2	0
3	1	1	2	0

V

	y^{ck}
1	1
2	1
3	2

NV

	x	y	z	w
1	0	1	2	0
2	1	1	2	0
3	1	2	2	0
4	1	2	2	3

nothing

↬ consistency bugs!

everything

↬ inefficient!

write-after-read variables

✓ real systems

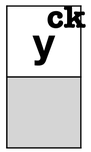
Checkpoint write-after-read variables

```
1 Ckpt(y)
2       x := y;
3       y := z;
4       w := x + y
```

NV

	x	y	z	w
	0	1	2	0
2	1	1	2	0
3	1	1	2	0

V



pwOff

NV	x	y	z	w
1	0	1	2	0
2	1	1	2	0
3	1	2	2	0
4	1	2	2	3

nothing

↝ consistency bugs!

everything

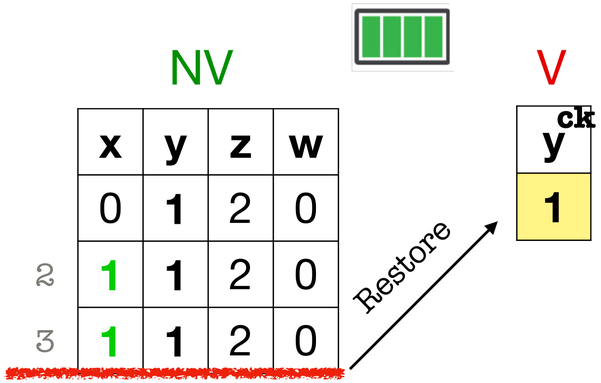
↝ inefficient!

write-after-read variables

✓ real systems

Checkpoint write-after-read variables

```
1 Ckpt(y)
2       x := y;
3       y := z;
4       w := x + y
```



NV	x	y	z	w
1	0	1	2	0
2	1	1	2	0
3	1	2	2	0
4	1	2	2	3

nothing

↬ consistency bugs!

everything

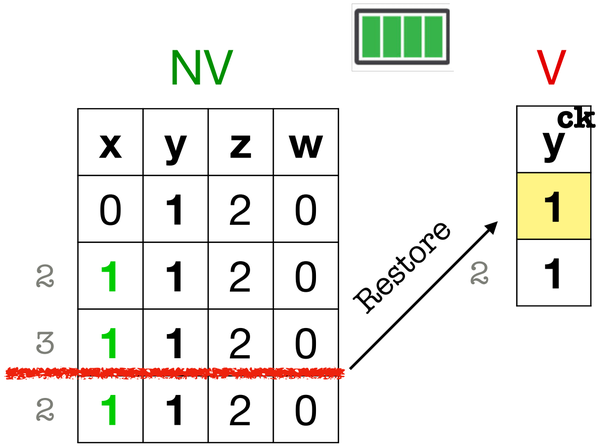
↬ inefficient!

write-after-read variables

✓ real systems

Checkpoint write-after-read variables

```
1 Ckpt(y)
2       x := y;
3       y := z;
4       w := x + y
```



nothing

↬ consistency bugs!

everything

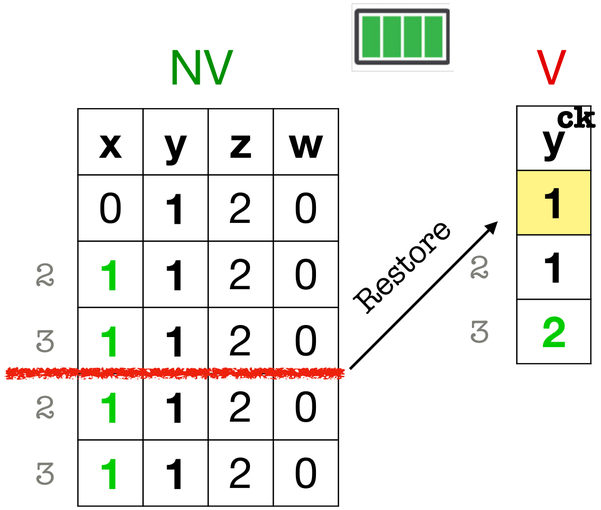
↬ inefficient!

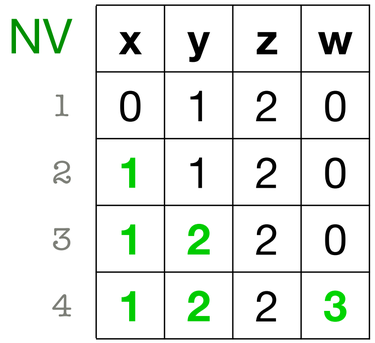
write-after-read variables

✓ real systems

Checkpoint write-after-read variables

```
1 Ckpt(y)
2       x := y;
3       y := z;
4       w := x + y
```





nothing

↬ consistency bugs!

everything

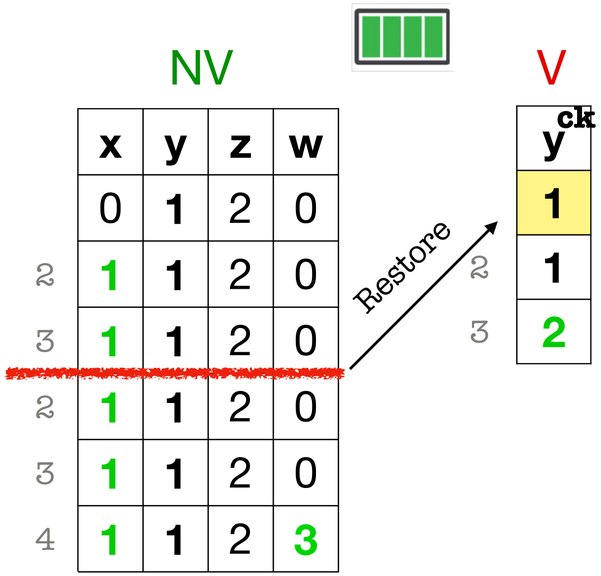
↬ inefficient!

write-after-read variables

✓ real systems

Checkpoint write-after-read variables

```
1 Ckpt(y)
2       x := y;
3       y := z;
4       w := x + y
```



nothing

↯ consistency bugs!

everything

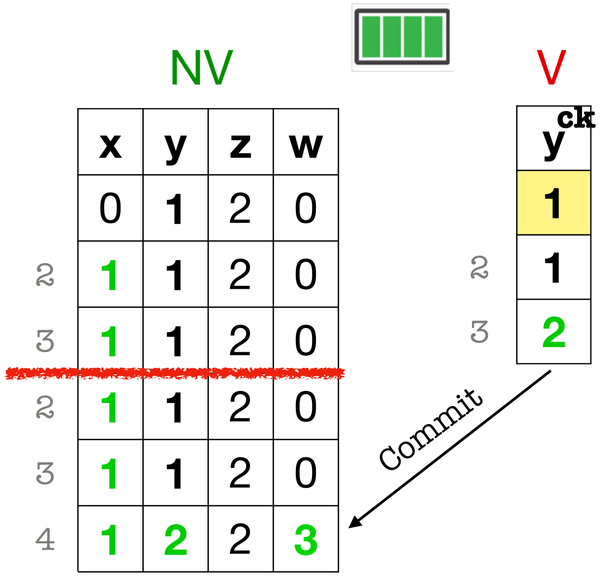
↯ inefficient!

write-after-read variables

✓ real systems

Checkpoint write-after-read variables

```
1 Ckpt(y)
2       x := y;
3       y := z;
4       w := x + y
```



nothing

↬ consistency bugs!

everything

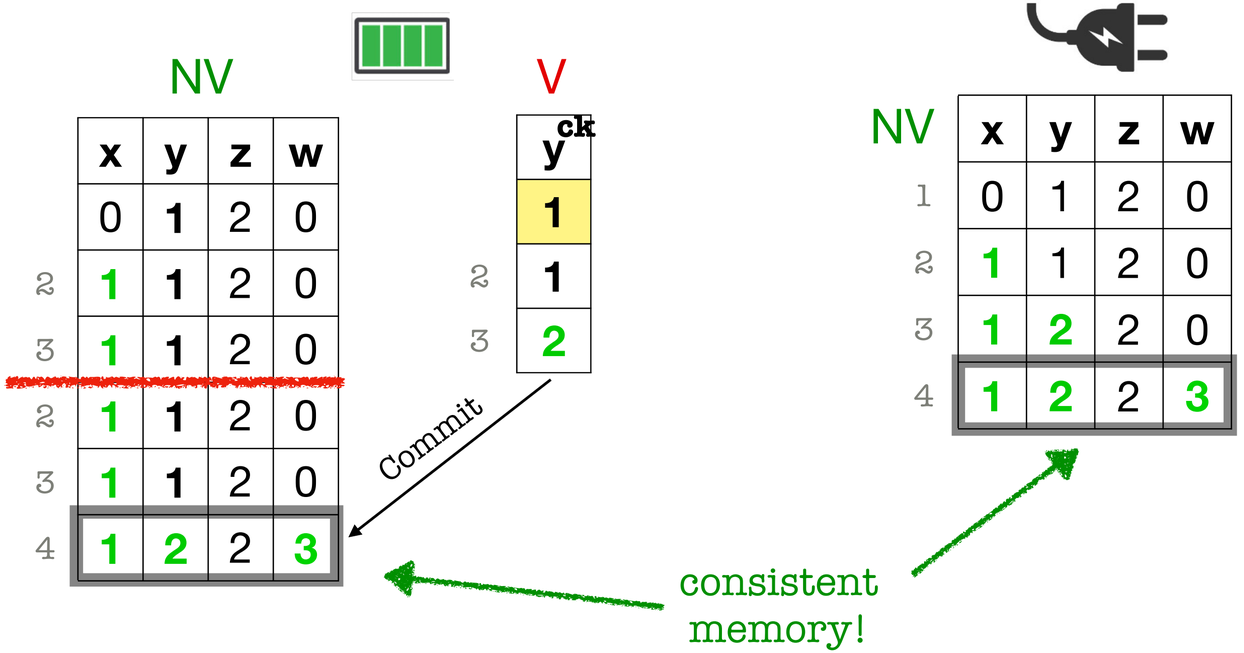
↬ inefficient!

write-after-read variables

✓ real systems

Checkpoint write-after-read variables

```
1 Ckpt(y)
2       x := y;
3       y := z;
4       w := x + y
```



nothing

↬ consistency bugs!

everything

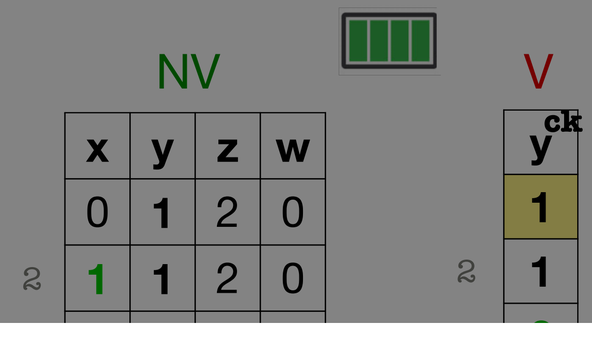
↬ inefficient!

write-after-read variables

✓ real systems

Checkpoint write-after-read variables

```
1 Ckpt(y)
2       x := y;
3       y := z;
4       w :=
```



nothing

consistency bugs!

thing

inefficient!

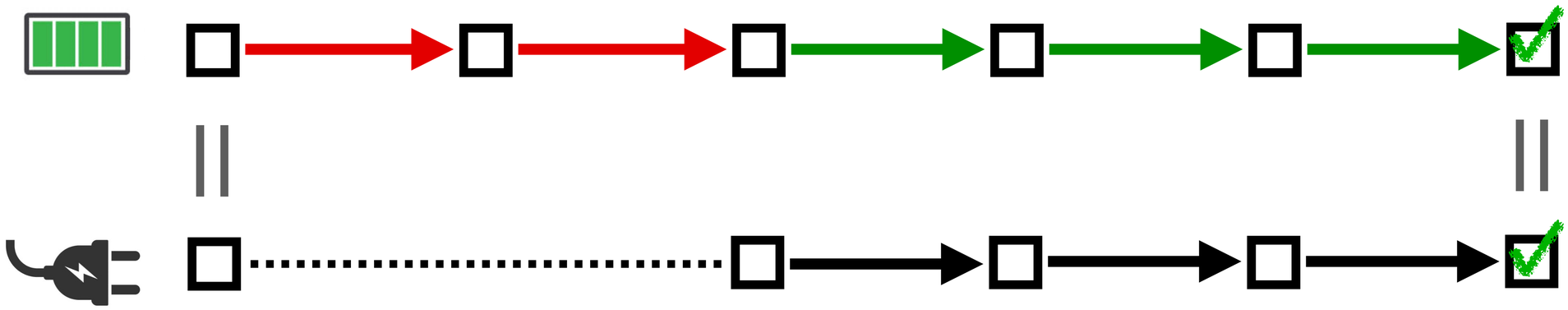
-after-read
ables

real systems

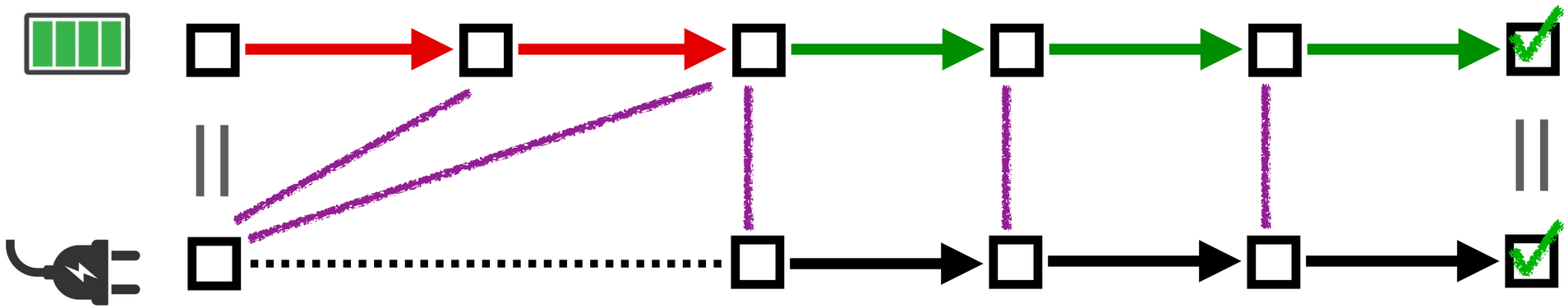
More precisely:

How can we **prove** that programs execute correctly intermittently when checkpointing only WAR variables?

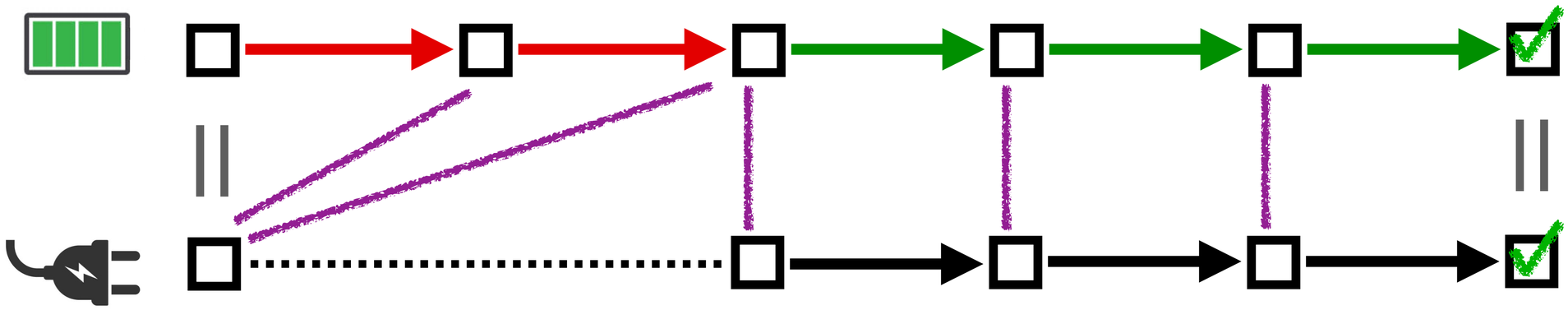
Correctness



Correctness

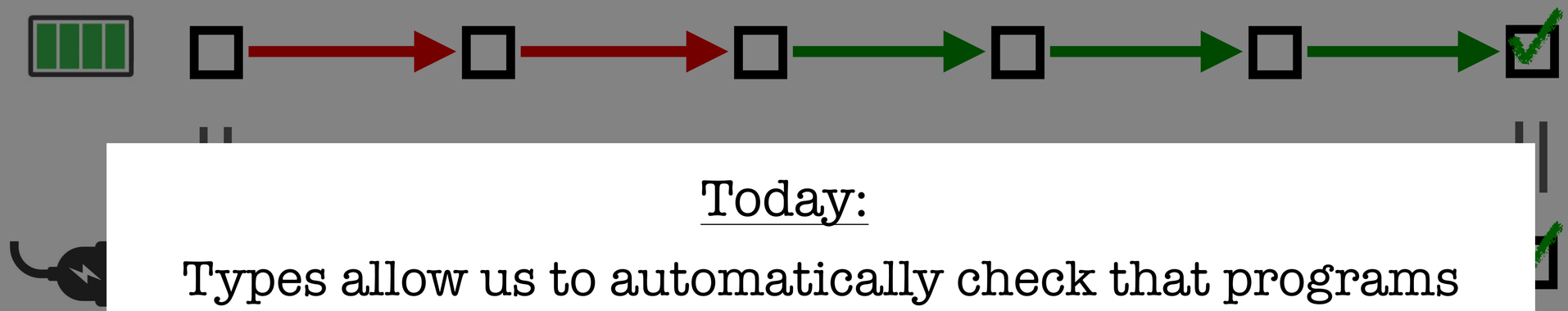


Correctness



first formal framework for reasoning about intermittent execution
[Surbatovich et al. ’20]

Correctness



Today:

Types allow us to automatically check that programs can execute **correctly intermittently**.

first formal framework for reasoning about intermittent execution [Surbatovich et al. '20]

Today: Intermittent Computing

1. intermittent computing + correctness
   [Surbatovich et al. ’20]
2. **correctness as a noninterference property**
   [Surbatovich et al. ’23]

Correctness (as a noninterference property)

A Type System for Safe Intermittent Computing [Surbatovich et al. ’23]

Real atomic regions may count attempts

```
1 Ckpt(y)
2       rb := rb + 1
3       x := y;
4       y := z;
5       w := x + y
```

`rb` **should not** be checkpointed!

Checkpoint WAR variables (except rb)

```
1 Ckpt(y)
2       rb := rb + 1
3       x := y;
4       y := z;
5       w := x + y
```

NV	x	y	z	w	rb
	0	**1**	2	0	0

NV	x	y	z	w	rb
1	0	1	2	0	0
2	0	1	2	0	**1**
3	**1**	1	2	0	**1**
4	**1**	**2**	2	0	**1**
5	**1**	**2**	2	**3**	**1**

nothing

↯ consistency bugs!

everything

↯ inefficient!

write-after-read variables (*)

✓ real systems

Checkpoint WAR variables (except rb)

```
1 Ckpt(y)
2       rb := rb + 1
3       x := y;
4       y := z;
5       w := x + y
```

NV

x	y	z	w	rb
0	**1**	2	0	0

Init → 1

V

y ck
1

NV	x	y	z	w	rb
1	0	1	2	0	0
2	0	1	2	0	**1**
3	**1**	1	2	0	**1**
4	**1**	**2**	2	0	**1**
5	**1**	**2**	2	**3**	**1**

nothing

↯ consistency bugs!

everything

↯ inefficient!

write-after-read variables (*)

✓ real systems

Checkpoint WAR variables (except rb)

```
1 Ckpt(y)
2       rb := rb + 1
3       x := y;
4       y := z;
5       w := x + y
```

NV

	x	y	z	w	rb
2	0	1	2	0	0
	0	1	2	0	1

V

	y ck
1	1
2	1

NV

	x	y	z	w	rb
1	0	1	2	0	0
2	0	1	2	0	1
3	1	1	2	0	1
4	1	2	2	0	1
5	1	2	2	3	1

nothing

↯ consistency bugs!

everything

↯ inefficient!

write-after-read variables (*)

✓ real systems

Checkpoint WAR variables (except rb)

```
1 Ckpt(y)
2       rb := rb + 1
3       x := y;
4       y := z;
5       w := x + y
```

NV

	x	y	z	w	rb
	0	1	2	0	0
2	0	1	2	0	1
3	1	1	2	0	1

V

	y (ck)
1	1
2	1
3	1

NV

	x	y	z	w	rb
1	0	1	2	0	0
2	0	1	2	0	1
3	1	1	2	0	1
4	1	2	2	0	1
5	1	2	2	3	1

nothing

↯ consistency bugs!

everything

↯ inefficient!

write-after-read variables (*)

✓ real systems

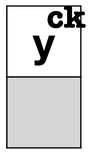
Checkpoint WAR variables (except rb)

```
1 Ckpt(y)
2       rb := rb + 1
3       x := y;
4       y := z;
5       w := x + y
```

NV

	x	y	z	w	rb
	0	1	2	0	0
2	0	1	2	0	1
3	1	1	2	0	1

V



pwOff

NV	x	y	z	w	rb
1	0	1	2	0	0
2	0	1	2	0	1
3	1	1	2	0	1
4	1	2	2	0	1
5	1	2	2	3	1

nothing

↯ consistency bugs!

everything

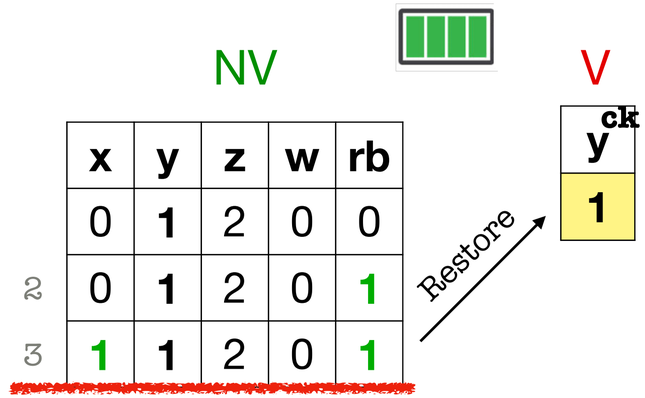
↯ inefficient!

write-after-read variables (*)

✓ real systems

Checkpoint WAR variables (except rb)

```
1 Ckpt(y)
2       rb := rb + 1
3       x := y;
4       y := z;
5       w := x + y
```



nothing

↯ consistency bugs!

everything

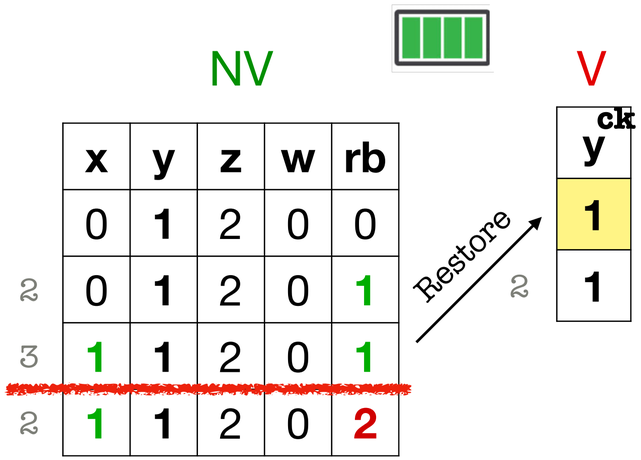
↯ inefficient!

write-after-read variables (*)

✓ real systems

Checkpoint WAR variables (except rb)

```
1 Ckpt(y)
2       rb := rb + 1
3       x := y;
4       y := z;
5       w := x + y
```



NV	x	y	z	w	rb
1	0	1	2	0	0
2	0	1	2	0	1
3	1	1	2	0	1
4	1	2	2	0	1
5	1	2	2	3	1

nothing

↯ consistency bugs!

everything

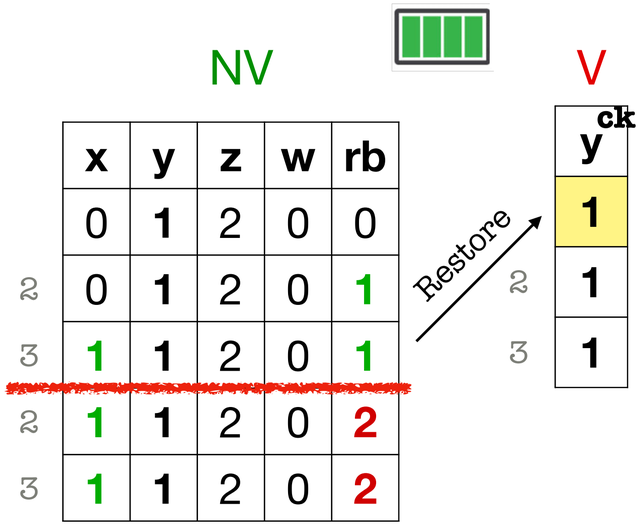
↯ inefficient!

write-after-read variables (*)

✓ real systems

Checkpoint WAR variables (except rb)

```
1 Ckpt(y)
2       rb := rb + 1
3       x := y;
4       y := z;
5       w := x + y
```



nothing

↝ consistency bugs!

everything

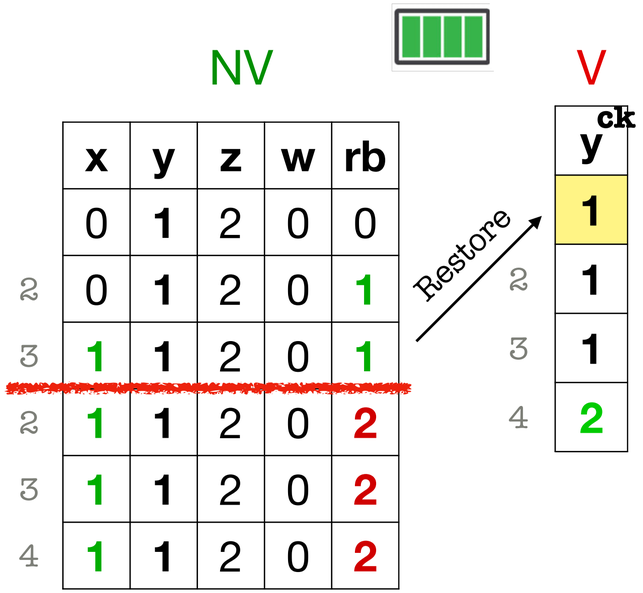
↝ inefficient!

write-after-read variables (*)

✓ real systems

Checkpoint WAR variables (except rb)

```
1 Ckpt(y)
2       rb := rb + 1
3       x := y;
4       y := z;
5       w := x + y
```



nothing

↬ consistency bugs!

everything

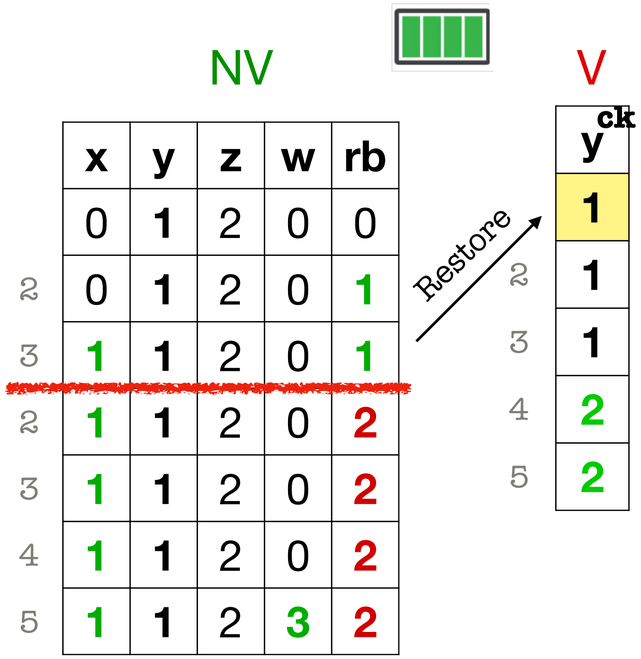
↬ inefficient!

write-after-read variables (*)

✓ real systems

Checkpoint WAR variables (except rb)

```
1 Ckpt(y)
2       rb := rb + 1
3       x := y;
4       y := z;
5       w := x + y
```



	x	y	z	w	rb
	0	1	2	0	0
2	0	1	2	0	1
3	1	1	2	0	1
2	1	1	2	0	2
3	1	1	2	0	2
4	1	1	2	0	2
5	1	1	2	3	2

	y
	1
2	1
3	1
4	2
5	2

NV

	x	y	z	w	rb
1	0	1	2	0	0
2	0	1	2	0	1
3	1	1	2	0	1
4	1	2	2	0	1
5	1	2	2	3	1

nothing

↯ consistency bugs!

everything

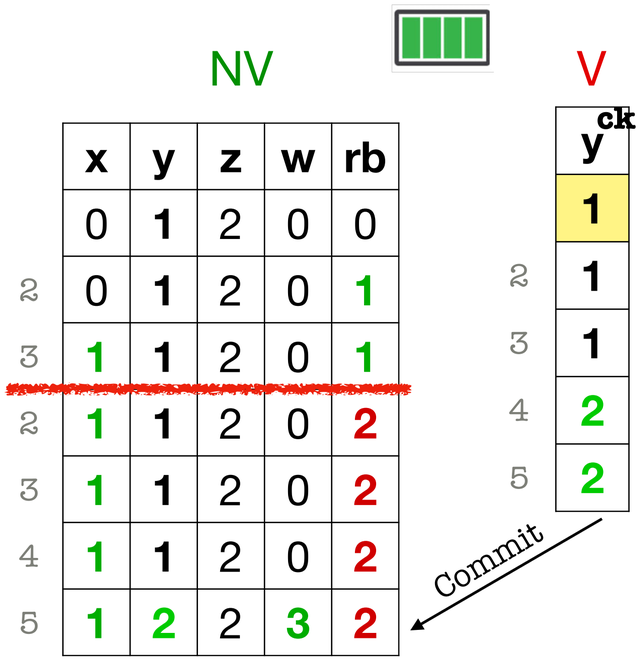
↯ inefficient!

write-after-read variables (*)

✓ real systems

Checkpoint WAR variables (except rb)

```
1 Ckpt(y)
2       rb := rb + 1
3       x := y;
4       y := z;
5       w := x + y
```



	x	y	z	w	rb
	0	1	2	0	0
2	0	1	2	0	1
3	1	1	2	0	1
2	1	1	2	0	2
3	1	1	2	0	2
4	1	1	2	0	2
5	1	2	2	3	2

	y ck
	1
2	1
3	1
4	2
5	2

NV

	x	y	z	w	rb
1	0	1	2	0	0
2	0	1	2	0	1
3	1	1	2	0	1
4	1	2	2	0	1
5	1	2	2	3	1

nothing

↝ consistency bugs!

everything

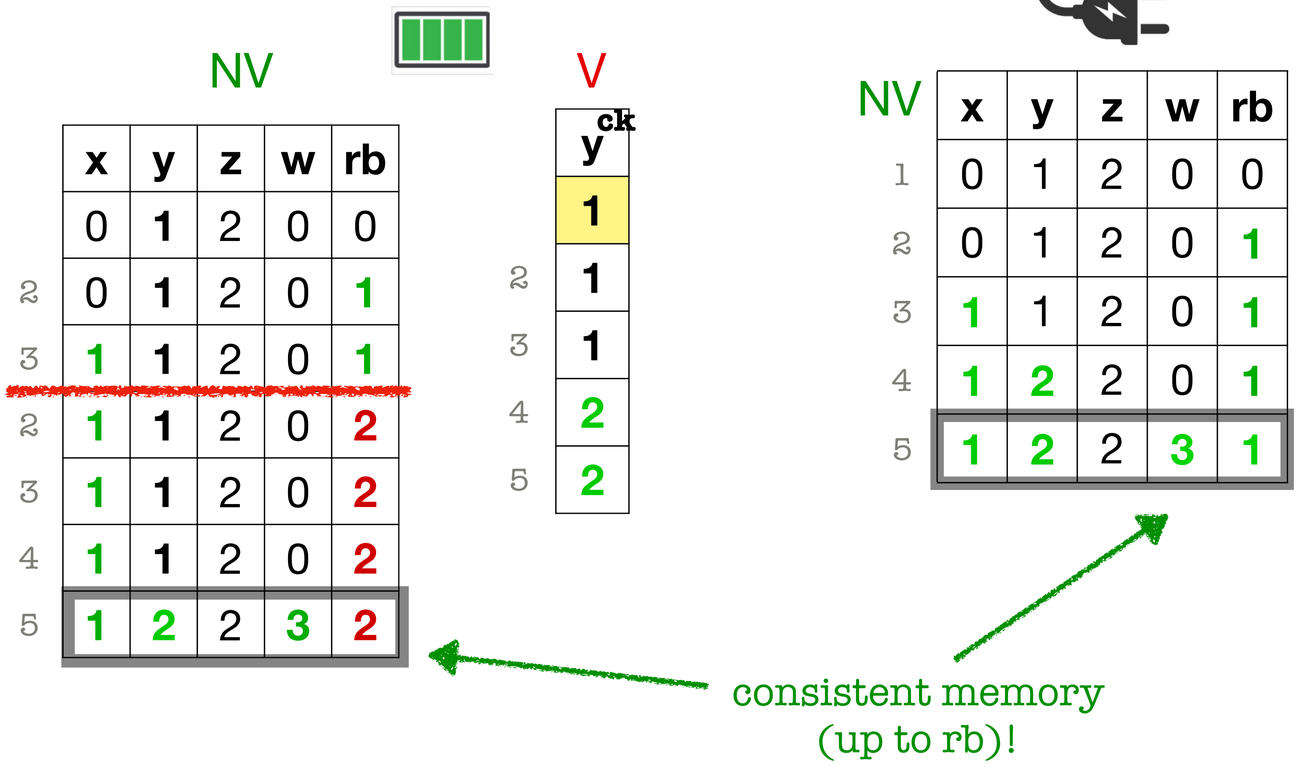
↝ inefficient!

write-after-read variables (*)

✓ real systems

Checkpoint WAR variables (except rb)

```
1 Ckpt(y)
2       rb := rb + 1
3       x := y;
4       y := z;
5       w := x + y
```



nothing

↯ consistency bugs!

everything

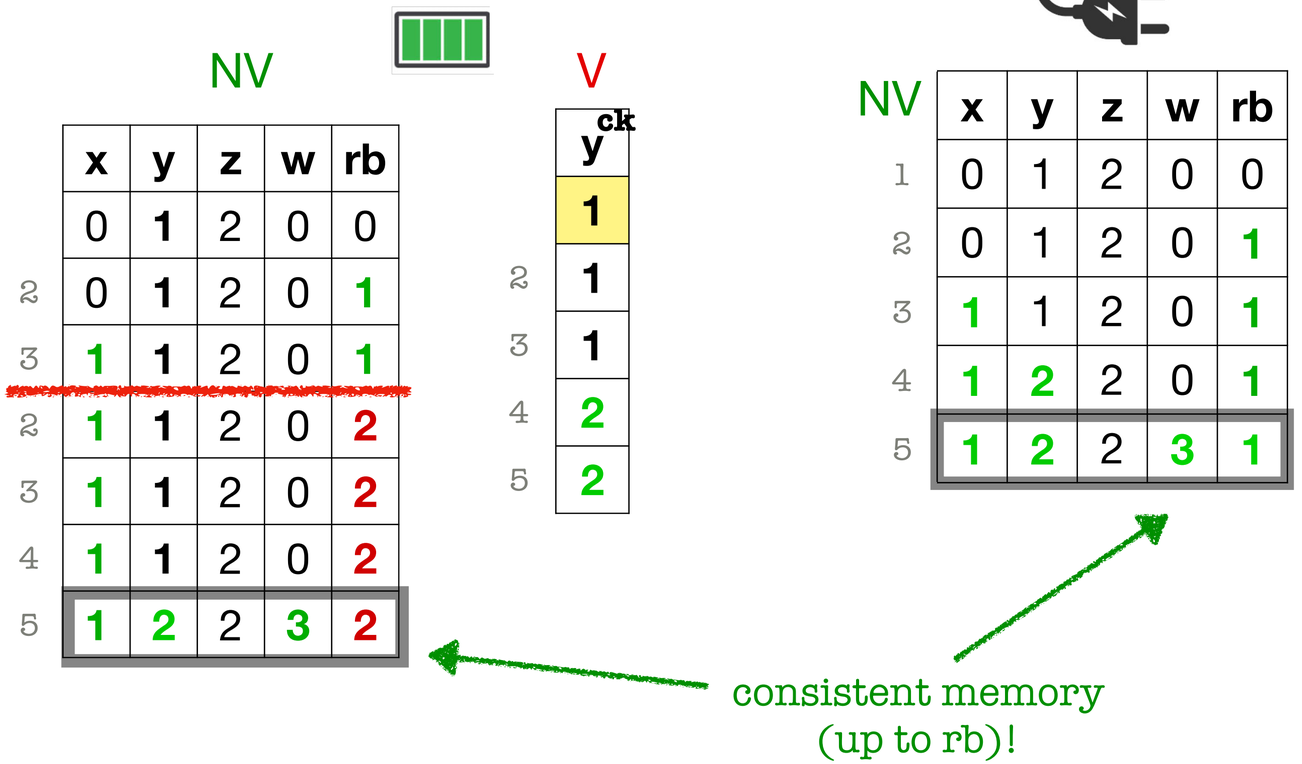
↯ inefficient!

write-after-read variables (*)

✓ real systems

Checkpoint WAR variables (except rb)

```
1 Ckpt(y)
2       rb := rb + 1
3       x := y;
4       y := z;
5       w := x + y
```



nothing

↯ consistency bugs!

everything

↯ inefficient!

write-after-read variables (*)

✓ real systems

rb is nonidempotent (Nid)

x,y,z,w are idempotent (Id)

Syntax

What do these programs look like?

```
Ckpt(y)
    rb := rb + 1
    x := y;
    y := z;
    w := x + y
```

Syntax

What do these programs look like?

values

$$v ::= n \mid \text{true} \mid \text{false}$$

```
Ckpt(y)
    rb := rb + 1
    x := y;
    y := z;
    w := x + y
```

Syntax

What do these programs look like?

values $v ::= n \mid \text{true} \mid \text{false}$

expressions $e ::= v \mid x \mid e_1 \odot e_2 \mid \oslash e$

```
Ckpt(y)
    rb := rb + 1
    x := y;
    y := z;
    w := x + y
```

Syntax

What do these programs look like?

```
Ckpt(y)
    rb := rb + 1
    x := y;
    y := z;
    w := x + y
```

values $v ::= n \mid \text{true} \mid \text{false}$

expressions $e ::= v \mid x \mid e_1 \odot e_2 \mid \oslash e$

programs $prog ::= \text{Ckpt}(aID, ckvars, rds, mtwt, nIds)\ c$

var. sets $ckvars ::= \cdot \mid x, ckvars$ $\quad nIds ::= \cdot \mid x, nIds$

$rds ::= \cdot \mid x, rds$ $\quad mtwt ::= \cdot \mid x, mtwt$

Syntax

What do these programs look like?

```
Ckpt(y)
    rb := rb + 1
    x := y;
    y := z;
    w := x + y
```

values $v ::= n \mid \text{true} \mid \text{false}$

expressions $e ::= v \mid x \mid e_1 \odot e_2 \mid \oslash\, e$

programs $prog ::= \text{Ckpt}(aID, ckvars, rds, mtwt, nIds)\ c$

var. sets $ckvars ::= \cdot \mid x, ckvars$ $\quad nIds ::= \cdot \mid x, nIds$

$rds ::= \cdot \mid x, rds$ $\quad mtwt ::= \cdot \mid x, mtwt$

commands $cmd ::= \text{skip} \mid x := e \mid \text{if } e \text{ then } c_1 \text{ else } c_2 \mid c_1; c_2 \mid \text{let } x = e \text{ in } c$

Basic Types

What kind of data can we store?

basic types

$T := \texttt{int} \mid \texttt{bool}$

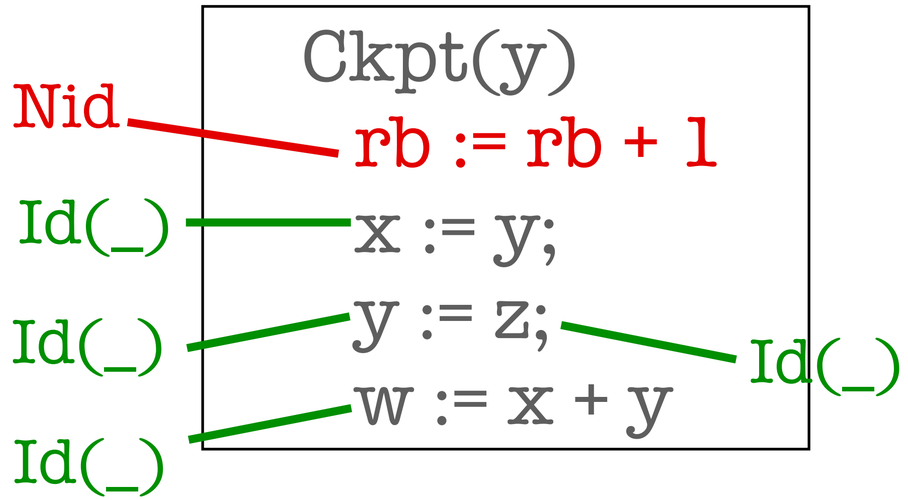
```
Ckpt(y)
  rb := rb + 1
  x := y;
  y := z;
  w := x + y
```

Idempotence qualifiers

Can variables be accessed idempotently?

basic types $T := \texttt{int} \mid \texttt{bool}$

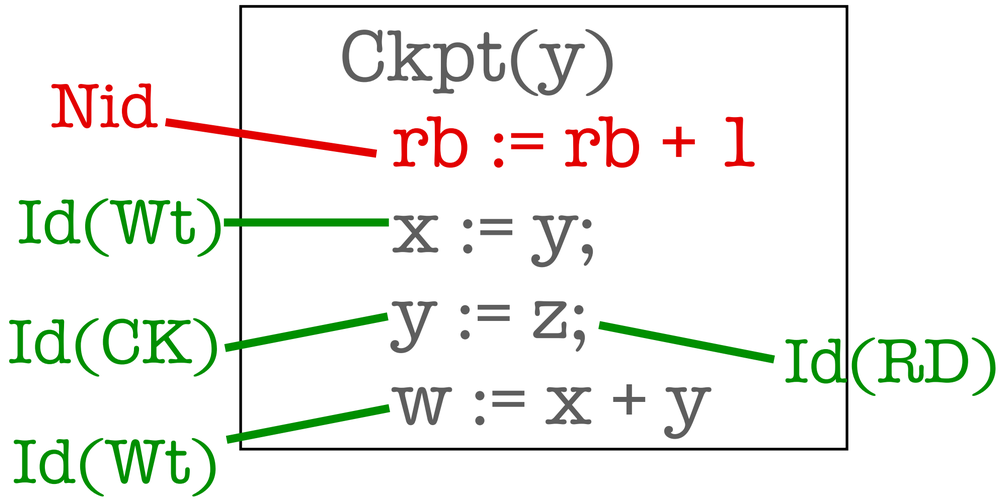
idempotence qualifiers $q := \text{Id}(qAcc) \mid \text{Nid} \mid \text{Id}$



Variable access modes

Why can variables be accessed idempotently?

basic types	$T := \texttt{int} \mid \texttt{bool}$
idempotence qualifiers	$q := \text{Id}(qAcc) \mid \text{Nid} \mid \text{Id}$
access qualifiers	$qAcc := \texttt{CK} \mid \texttt{RD} \mid \texttt{Wt}$



CK — checkpointed

RD — read only

Wt — write

Variable access modes

Why can variables be accessed idempotently?

basic types	$T := \texttt{int} \mid \texttt{bool}$
idempotence qualifiers	$q := \text{Id}(qAcc) \mid \text{Nid} \mid \text{Id}$
access qualifiers	$qAcc := \texttt{CK} \mid \texttt{RD} \mid \texttt{Wt}$

```
Ckpt(y)
  rb := rb + 1
  x := y;
  y := z;
  w := x + y
```

rb : int@Nid z : int@Id(RD)

x : int@Id(Wt) w : int@Id(Wt)

y : int@Id(CK)

Qualifier structure

Nid variables should not influence Id variables.

$\mathtt{Id}(\_) \sqsubset \mathtt{Nid}$

x:Nid, y:Id(_)

x := y ✓

y := x ✗

$\mathtt{CK} < \mathtt{RD}$

$\mathtt{CK} < \mathtt{Wt}$

TBT: Information Flow!

Nid variables should not influence Id variables.

$\mathtt{Id}(\_) \sqsubset \mathtt{Nid}$

x:Nid, y:Id(_) x := y ✓ y := x ✗

$\mathtt{CK} < \mathtt{RD}$

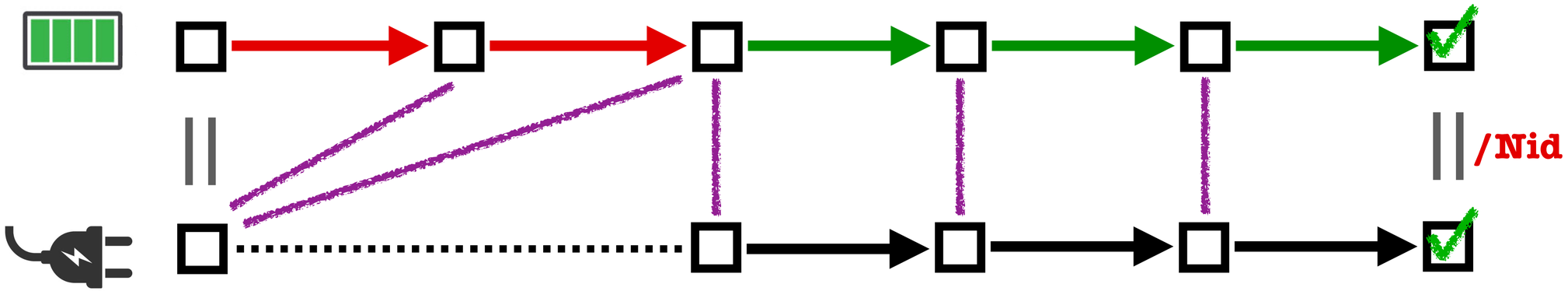
$\mathtt{CK} < \mathtt{Wt}$

H variables should not flow to L variables.

$\mathtt{L} \sqsubset \mathtt{H}$

x:H, y:L x := y ✓ y := x ✗

Correctness in terms of noninterference



correctness

We define that program α satisfies ~~*noninterference*~~ *with respect to policy* Σ iff for all ω_1, ω_2, ν_1, and ν_2,
$\Sigma \vdash \omega_1 \approx_{\not L} \omega_2$, eval $\omega_1\ \alpha = \nu_1$, and eval $\omega_2\ \alpha = \nu_2$ implies $\Sigma \vdash \nu_1 \approx_{\not L} \nu_2$.

Id(_) *Id(_)*

Typing Judgments

values $\Sigma \vdash v : T@q$

expressions $\Sigma \vdash e : T@q$

commands $\Sigma, \mathsf{Wts} \vdash cmd \dashv \mathsf{Wts}'$

programs $\Sigma \vdash prog$ correct

Type Checking Values

$$\boxed{\Sigma \vdash v : T @ q}$$

$$\frac{}{\Sigma \vdash \texttt{true} : \texttt{bool} @ \texttt{Id}}\ \textit{true-C}$$

Type Checking Values

$$\boxed{\Sigma \vdash v : T @ q}$$

$$\frac{}{\Sigma \vdash \mathsf{true} : \texttt{bool} @ \texttt{Id}}\ \textit{true-C}$$

$$\frac{}{\Sigma \vdash \mathsf{false} : \texttt{bool} @ \texttt{Id}}\ \textit{false-C}$$

Type Checking Values

$$\boxed{\Sigma \vdash v : T @ q}$$

$$\frac{}{\Sigma \vdash \mathsf{true} : \texttt{bool} @ \texttt{Id}}\ \textit{true-C}$$

$$\frac{}{\Sigma \vdash \mathsf{false} : \texttt{bool} @ \texttt{Id}}\ \textit{false-C}$$

$$\frac{}{\Sigma \vdash n : \texttt{int} @ \texttt{Id}}\ \textit{int-C}$$

Type Checking Expressions

$$\boxed{\Sigma \vdash e : T@q}$$

$$\frac{\Sigma(x) = T@q}{\Sigma \vdash x : T@q}\ \textit{var-C}$$

Type Checking Expressions

$$\boxed{\Sigma \vdash e : T@q}$$

$$\frac{\Sigma(x) = T@q}{\Sigma \vdash x : T@q}\ \textit{var-C}$$

$$\frac{\Sigma \vdash e : T@q}{\Sigma \vdash \oslash\, e : T@q}\ \textit{unop-C}$$

Type Checking Expressions

$$\boxed{\Sigma \vdash e : T@q}$$

$$\frac{\Sigma(x) = T@q}{\Sigma \vdash x : T@q}\ \textit{var-C}$$

$$\frac{\Sigma \vdash e : T@q}{\Sigma \vdash \oslash\, e : T@q}\ \textit{unop-C}$$

$$\frac{\Sigma \vdash e_1 : T@q_1 \quad \Sigma \vdash e_2 : T@q_2}{\Sigma \vdash e_1 \odot e_2 : T@q_1 \sqcup q_2}\ \textit{binop-C}$$

Type Checking Commands: Assignment

$\Sigma \vdash \alpha$ secure

$$\frac{\Sigma \vdash e : \ell \quad \ell' = \Sigma(pc) \sqcup \ell \quad \ell' \sqsubseteq \Sigma(x)}{\Sigma \vdash x := e \text{ secure}} :=F$$

Σ, Wts $\vdash$ c $\dashv$ Wts'

$$\frac{}{\Sigma, \text{Wts} \vdash x := e \dashv \text{Wts} \cup \{x\}} :=F$$

Type Checking Commands: Assignment

$\Sigma \vdash \alpha$ secure

$$\frac{\Sigma \vdash e : \ell \quad \ell' = \Sigma(pc) \sqcup \ell \quad \ell' \sqsubseteq \Sigma(x)}{\Sigma \vdash x := e \text{ secure}} :=F$$

Σ, Wts $\vdash c \dashv$ Wts'

$$\frac{\Sigma \vdash e : T@q \qquad q' = \Sigma(pc) \sqcup q \qquad q' \sqsubseteq q_x}{\Sigma, \text{Wts} \vdash x := e \dashv \text{Wts} \cup \{x\}} :=F$$

if e is Nid, then can only write to x if it is Nid

Type Checking Commands: Assignment

$\Sigma \vdash \alpha$ secure

$$\frac{\Sigma \vdash e : \ell \quad \ell' = \Sigma(pc) \sqcup \ell \quad \ell' \sqsubseteq \Sigma(x)}{\Sigma \vdash x := e \text{ secure}} :=F$$

Σ, Wts $\vdash$ c $\dashv$ Wts'

$$\Sigma \vdash x : T @ q_x$$

if q_x = Id(q) then q < Wt

$$\frac{\Sigma \vdash e : T @ q \qquad q' = \Sigma(pc) \sqcup q \qquad q' \sqsubseteq q_x}{\Sigma, \text{Wts} \vdash x := e \dashv \text{Wts} \cup \{x\}} :=F$$

check that x is writeable

Type Checking Commands: Sequencing

$\Sigma \vdash \alpha$ secure

$$\frac{\Sigma \vdash \alpha \text{ secure} \quad \Sigma \vdash \beta \text{ secure}}{\Sigma \vdash \alpha \,;\, \beta \text{ secure}} \;;F$$

Σ, Wts $\vdash$ c $\dashv$ Wts'

$$\frac{}{\Sigma, \text{ Wts} \vdash c_1; c_2 \dashv \text{Wts''}} \;;F$$

Type Checking Commands: Sequencing

$\Sigma \vdash \alpha$ secure

$$\frac{\Sigma \vdash \alpha \text{ secure} \quad \Sigma \vdash \beta \text{ secure}}{\Sigma \vdash \alpha \,;\, \beta \text{ secure}} \;;F$$

$\Sigma,$ Wts $\vdash c \dashv$ Wts'

$$\frac{\Sigma, \text{Wts} \vdash c_1 \dashv \text{Wts}'}{\Sigma, \text{Wts} \vdash c_1; c_2 \dashv \text{Wts}''} \;;F$$

check the first program

Type Checking Commands: Sequencing

$\Sigma \vdash \alpha$ secure

$$\frac{\Sigma \vdash \alpha \text{ secure} \quad \Sigma \vdash \beta \text{ secure}}{\Sigma \vdash \alpha \,;\, \beta \text{ secure}} \;;F$$

$\Sigma,$ Wts $\vdash c \dashv$ Wts'

$$\frac{\Sigma, \text{Wts} \vdash c_1 \dashv \text{Wts}' \qquad \Sigma, \text{Wts}' \vdash c_2 \dashv \text{Wts}''}{\Sigma, \text{Wts} \vdash c_1; c_2 \dashv \text{Wts}''} \;;F$$

check the second program

Type Checking Commands: If-then-else

$\Sigma \vdash \alpha$ secure

$$\frac{\Sigma \vdash P : \ell \quad \ell' = \Sigma(pc) \sqcup \ell \quad \Sigma' = \Sigma[pc \mapsto \ell'] \quad \Sigma' \vdash \alpha \text{ secure} \quad \Sigma' \vdash \beta \text{ secure}}{\Sigma \vdash \textbf{if } P \textbf{ then } \alpha \textbf{ else } \beta \text{ secure}} \textbf{if} F$$

Σ, Wts $\vdash$ c $\dashv$ Wts'

$$\frac{}{\Sigma, \text{Wts} \vdash \textbf{if } e \textbf{ then } c_1 \textbf{ else } c_2 \dashv \text{Wts}'} \textbf{if} F$$

Type Checking Commands: If-then-else

$\Sigma \vdash \alpha$ secure

$$\frac{\Sigma \vdash P : \ell \quad \ell' = \Sigma(pc) \sqcup \ell \quad \Sigma' = \Sigma[pc \mapsto \ell'] \quad \Sigma' \vdash \alpha \text{ secure} \quad \Sigma' \vdash \beta \text{ secure}}{\Sigma \vdash \textbf{if } P \textbf{ then } \alpha \textbf{ else } \beta \text{ secure}} \textbf{ if}F$$

Σ, Wts $\vdash$ c $\dashv$ Wts'

check the first branch

$$\frac{\Sigma', \text{Wts} \vdash c_1 \dashv \text{Wts}'}{\Sigma, \text{Wts} \vdash \textbf{if } e \textbf{ then } c_1 \textbf{ else } c_2 \dashv \text{Wts}'} \textbf{ if}F$$

Type Checking Commands: If-then-else

$\Sigma \vdash \alpha$ secure

$$\frac{\Sigma \vdash P : \ell \quad \ell' = \Sigma(pc) \sqcup \ell \quad \Sigma' = \Sigma[pc \mapsto \ell'] \quad \Sigma' \vdash \alpha \text{ secure} \quad \Sigma' \vdash \beta \text{ secure}}{\Sigma \vdash \textbf{if } P \textbf{ then } \alpha \textbf{ else } \beta \text{ secure}} \textbf{if} F$$

Σ, Wts $\vdash c \dashv$ Wts'

check the second branch

$$\frac{\Sigma', \text{Wts} \vdash c_1 \dashv \text{Wts}' \qquad \Sigma', \text{Wts} \vdash c_2 \dashv \text{Wts}'}{\Sigma, \text{Wts} \vdash \textbf{if } e \textbf{ then } c_1 \textbf{ else } c_2 \dashv \text{Wts}'} \textbf{if} F$$

Type Checking Commands: If-then-else

$\Sigma \vdash \alpha$ secure

$$\frac{\Sigma \vdash P : \ell \quad \ell' = \Sigma(pc) \sqcup \ell \quad \Sigma' = \Sigma[pc \mapsto \ell'] \quad \Sigma' \vdash \alpha \text{ secure} \quad \Sigma' \vdash \beta \text{ secure}}{\Sigma \vdash \textbf{if } P \textbf{ then } \alpha \textbf{ else } \beta \text{ secure}} \textbf{ if } F$$

Σ, Wts $\vdash c \dashv$ Wts'

set the pc

$$\frac{\Sigma \vdash e : \texttt{bool}@q \quad q' = \Sigma(pc) \sqcup q \quad \Sigma' = \Sigma[pc \mapsto q'] \quad \Sigma', \text{Wts} \vdash c_1 \dashv \text{Wts}' \quad \Sigma', \text{Wts} \vdash c_2 \dashv \text{Wts}'}{\Sigma, \text{Wts} \vdash \textbf{if } e \textbf{ then } c_1 \textbf{ else } c_2 \dashv \text{Wts}'} \textbf{ if } F$$

Type Checking Programs

$\Sigma \vdash prog$ correct

check the command c

$$\Sigma_{\text{init}}, \cdot \vdash c \dashv \text{Wts}$$

$$\frac{}{\Sigma \vdash \text{Ckpt}(aID, ckvars, rds, mtwt, nIds)\ c \ \text{ correct}} \ \textit{Ckpt-C}$$

Type Checking Programs

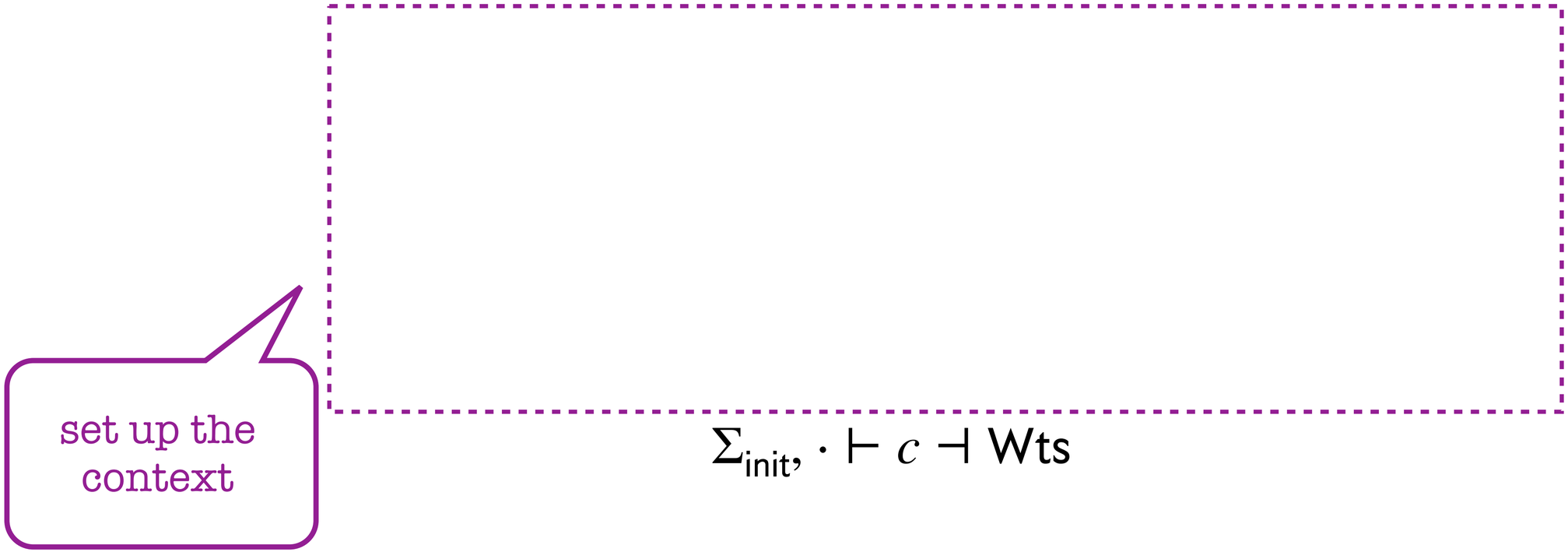
$\Sigma \vdash prog$ correct

but wait, what's this?

$$\frac{\Sigma_{\text{init}}, \cdot \vdash c \dashv \text{Wts}}{\Sigma \vdash \text{Ckpt}(aID, ckvars, rds, mtwt, nIds)\ c \ \text{ correct}}\ \textit{Ckpt-C}$$

Type Checking Programs

$\Sigma \vdash prog$ correct



Type Checking Programs

initialize the pc

correct

$\Sigma_{\text{init}}(pc) = \texttt{Id}$

set up the context

$\Sigma_{\text{init}}, \cdot \vdash c \dashv \text{Wts}$

$$\frac{}{\Sigma \vdash \text{Ckpt}(aID, ckvars, rds, mtwt, nIds)\ c\ \ \text{correct}}\ \textit{Ckpt-C}$$

Type Checking Programs

$\Sigma \vdash p$

set up Nids

$$\Sigma_{\text{init}}(pc) = \texttt{Id}$$

$$\forall x \in nIds\,.\,\Sigma_{\text{init}}(x) = T@\texttt{Nid} \text{ where } \Gamma(x) = T@q'$$

set up the context

$$\Sigma_{\text{init}}, \cdot \vdash c \dashv \text{Wts}$$

$$\Sigma \vdash \text{Ckpt}(aID, ckvars, rds, mtwt, nIds)\ c \quad \text{correct}$$

Ckpt-C

Type Checking Programs

$\Sigma \vdash prog$ correct

set up checkpoints

$$\Sigma_{\text{init}}(pc) = \texttt{Id}$$

$$\forall x \in nIds\,.\,\Sigma_{\text{init}}(x) = T@\texttt{Nid} \text{ where } \Gamma(x) = T@q'$$

$$\forall x \in ckvars\,.\,\Sigma_{\text{init}}(x) = T@\texttt{Id(CK)} \text{ where } \Gamma(x) = T@q'$$

set up the context

$$\Sigma_{\text{init}}, \cdot \vdash c \dashv \text{Wts}$$

$$\frac{}{\Sigma \vdash \text{Ckpt}(aID, ckvars, rds, mtwt, nIds)\ c \ \text{ correct}}\ \textit{Ckpt-C}$$

Type Checking Programs

$\Sigma \vdash prog$ correct

$$\frac{\begin{array}{c}\Sigma_{\text{init}}(pc) = \texttt{Id} \\ \forall x \in nIds\,.\,\Sigma_{\text{init}}(x) = T@\texttt{Nid} \text{ where } \Gamma(x) = T@q' \\ \forall x \in ckvars\,.\,\Sigma_{\text{init}}(x) = T@\texttt{Id(CK)} \text{ where } \Gamma(x) = T@ \\ \forall x \in rds\,.\,\Sigma_{\text{init}}(x) = T@\texttt{Id(RD)} \text{ where } \Gamma(x) = T@q' \\ \Sigma_{\text{init}}, \cdot \vdash c \dashv \text{Wts}\end{array}}{\Sigma \vdash \text{Ckpt}(aID, ckvars, rds, mtwt, nIds)\ c \ \text{ correct}}\ \textit{Ckpt-C}$$

set up read-only vars.

set up the context

Type Checking Programs

$\Sigma \vdash prog$ correct

$$\frac{\begin{array}{c}\Sigma_{\text{init}}(pc) = \texttt{Id} \\ \forall x \in nIds\,.\,\Sigma_{\text{init}}(x) = T@\texttt{Nid} \text{ where } \Gamma(x) = T@q' \\ \forall x \in ckvars\,.\,\Sigma_{\text{init}}(x) = T@\texttt{Id(CK)} \text{ where } \Gamma(x) = T@ \\ \forall x \in rds\,.\,\Sigma_{\text{init}}(x) = T@\texttt{Id(RD)} \text{ where } \Gamma(x) = T@q' \\ \forall x \in mtwt\,.\,\Sigma_{\text{init}}(x) = T@\texttt{Id(Wt)} \text{ where } \Gamma(x) = T@q' \\ \Sigma_{\text{init}}, \cdot \vdash c \dashv \text{Wts}\end{array}}{\Sigma \vdash \text{Ckpt}(aID, ckvars, rds, mtwt, nIds)\ c \ \text{ correct}}\ \textit{Ckpt-C}$$

set up the context

set up the writes

Type Checking Programs

$\Sigma \vdash prog$ correct

$$\Sigma_{\text{init}}(pc) = \texttt{Id}$$

$$\forall x \in nIds\,.\,\Sigma_{\text{init}}(x) = T@\texttt{Nid} \text{ where } \Gamma(x) = T@q'$$

$$\forall x \in ckvars\,.\,\Sigma_{\text{init}}(x) = T@\texttt{Id(CK)} \text{ where } \Gamma(x) = T@q'$$

$$\forall x \in rds\,.\,\Sigma_{\text{init}}(x) = T@\texttt{Id(RD)} \text{ where } \Gamma(x) = T@q'$$

$$\forall x \in mtwt\,.\,\Sigma_{\text{init}}(x) = T@\texttt{Id(Wt)} \text{ where } \Gamma(x) = T@q'$$

check that all writes have been written

$$\Sigma_{\text{init}}, \cdot \vdash c \dashv \text{Wts}$$

$$\forall x \text{ s.t. } \Sigma_{\text{init}}(x) = T@\texttt{Id(Wt)}, x \in \text{Wts}$$

$$\frac{}{\Sigma \vdash \text{Ckpt}(aID, ckvars, rds, mtwt, nIds)\ c \text{ correct}}\ \textit{Ckpt-C}$$

Set up the context

$$\Sigma_0 \vdash$$

```
Ckpt(y)
    rb := rb + 1
    x := y;
    y := z;
    w := x + y
```

$$\Sigma = \{\texttt{rb} : \texttt{Nid}, \texttt{x} : \texttt{Id(Wt)}, \texttt{y} : \texttt{Id(CK)}, \texttt{z} : \texttt{Id(RD)}, \texttt{w} : \texttt{Id(Wt)}\}$$

Check the command

$$\Sigma, \varnothing \vdash \texttt{rb := rb + 1; x := y; y := z; w := x + y}$$

$$\Sigma_0 \vdash \boxed{\begin{array}{l}\texttt{Ckpt(y)} \\ \quad \texttt{rb := rb + 1} \\ \quad \texttt{x := y;} \\ \quad \texttt{y := z;} \\ \quad \texttt{w := x + y}\end{array}}$$

$\Sigma = \{\texttt{rb} : \texttt{Nid}, \texttt{x} : \texttt{Id(Wt)}, \texttt{y} : \texttt{Id(CK)}, \texttt{z} : \texttt{Id(RD)}, \texttt{w} : \texttt{Id(Wt)}\}$

Sequencing

$$\frac{\Sigma, \mathrm{Wts} \vdash \alpha \dashv \mathrm{Wts}' \qquad \Sigma, \mathrm{Wts}' \vdash \beta \dashv \mathrm{Wts}''}{\Sigma, \mathrm{Wts} \vdash \alpha; \beta \dashv \mathrm{Wts}''}$$

$$\frac{\Sigma, \emptyset \vdash \texttt{rb := rb + 1}}{\Sigma, \emptyset \vdash \begin{array}{l}\texttt{rb := rb + 1; x := y;}\\ \texttt{y := z; w := x + y}\end{array}}\ ;F$$

$$\Sigma_0 \vdash \boxed{\begin{array}{l}\texttt{Ckpt(y)}\\ \quad\texttt{rb := rb + 1}\\ \quad\texttt{x := y;}\\ \quad\texttt{y := z;}\\ \quad\texttt{w := x + y}\end{array}}$$

$\Sigma = \{\texttt{rb} : \texttt{Nid}, \texttt{x} : \texttt{Id(Wt)}, \texttt{y} : \texttt{Id(CK)}, \texttt{z} : \texttt{Id(RD)}, \texttt{w} : \texttt{Id(Wt)}\}$

Assignment

$$\frac{\Sigma \vdash x : T@q_x \quad \text{if } q_x = \mathtt{Id(q)} \text{ then } \mathtt{q} < \mathtt{Wt} \quad \Sigma \vdash e : T@q \quad q' = \Sigma(pc) \sqcup q \quad q' \sqsubseteq q_x}{\Sigma, \mathtt{Wts} \vdash x := e \dashv \mathtt{Wts} \cup \{x\}} :=F$$

$$q' = \Sigma(pc) \sqcup \mathtt{Nid}$$

$$q' \sqsubseteq \mathtt{Nid}$$

$$\frac{\Sigma, \emptyset \vdash \mathtt{rb} + 1 : \mathtt{Nid} \quad \Sigma \vdash \mathtt{rb} : \mathtt{Nid}}{\Sigma, \emptyset \vdash \mathtt{rb} := \mathtt{rb} + 1 \dashv \{\mathtt{rb}\}} :=F$$

$$\frac{\Sigma, \emptyset \vdash \mathtt{rb} := \mathtt{rb} + 1 \dashv \{\mathtt{rb}\}}{\Sigma, \emptyset \vdash \mathtt{rb} := \mathtt{rb} + 1;\ \mathtt{x} := \mathtt{y};\ \mathtt{y} := \mathtt{z};\ \mathtt{w} := \mathtt{x} + \mathtt{y}} ;F$$

$$\Sigma_0 \vdash \boxed{\begin{array}{l}\mathtt{Ckpt(y)} \\ \quad \mathtt{rb} := \mathtt{rb} + 1 \\ \quad \mathtt{x} := \mathtt{y}; \\ \quad \mathtt{y} := \mathtt{z}; \\ \quad \mathtt{w} := \mathtt{x} + \mathtt{y}\end{array}}$$

$$\Sigma = \{\mathtt{rb} : \mathtt{Nid}, \mathtt{x} : \mathtt{Id(Wt)}, \mathtt{y} : \mathtt{Id(CK)}, \mathtt{z} : \mathtt{Id(RD)}, \mathtt{w} : \mathtt{Id(Wt)}\}$$

Sequencing

$$\frac{\Sigma, \mathtt{Wts} \vdash \alpha \dashv \mathtt{Wts'} \qquad \Sigma, \mathtt{Wts'} \vdash \beta \dashv \mathtt{Wts''}}{\Sigma, \mathtt{Wts} \vdash \alpha; \beta \dashv \mathtt{Wts''}}$$

$$\Sigma_0 \vdash \boxed{\begin{array}{l}\mathtt{Ckpt(y)}\\ \quad \mathtt{rb := rb + 1}\\ \quad \mathtt{x := y;}\\ \quad \mathtt{y := z;}\\ \quad \mathtt{w := x + y}\end{array}}$$

$$\frac{\dfrac{\dfrac{\ell' = \Sigma(pc) \sqcup \mathtt{Nid} \quad \ell' \sqsubseteq \mathtt{Nid} \quad \Sigma, \varnothing \vdash \mathtt{rb + 1 : Nid}}{\Sigma, \varnothing \vdash \mathtt{rb := rb + 1} \dashv \{\mathtt{rb}\}}\ :=F \qquad \Sigma, \{\mathtt{rb}\} \vdash \begin{array}{l}\mathtt{x := y;}\\ \mathtt{y := z;\ w := x + y}\end{array}}{\Sigma, \varnothing \vdash \begin{array}{l}\mathtt{rb := rb + 1;\ x := y;}\\ \mathtt{y := z;\ w := x + y}\end{array}}\ ;F}{\Sigma_0 \vdash \boxed{\begin{array}{l}\mathtt{Ckpt(y)}\\ \quad \mathtt{rb := rb + 1}\\ \quad \mathtt{x := y;}\\ \quad \mathtt{y := z;}\\ \quad \mathtt{w := x + y}\end{array}}}$$

$$\Sigma = \{\mathtt{rb : Nid}, \mathtt{x : Id(Wt)}, \mathtt{y : Id(CK)}, \mathtt{z : Id(RD)}, \mathtt{w : Id(Wt)}\}$$

Assignment

$\Sigma \vdash x : T@q_x$

if q_x = Id(q) then q < Wt

$$\frac{\Sigma \vdash e : T@q \qquad q' = \Sigma(pc) \sqcup q \qquad q' \sqsubseteq q_x}{\Sigma, \mathtt{Wts} \vdash x := e \dashv \mathtt{Wts} \cup \{x\}} :=F$$

$q' = \Sigma(pc) \sqcup \mathtt{Id}$

$q' \sqsubseteq \mathtt{Id}$

$$\dfrac{\dfrac{\dfrac{\Sigma, \{\mathtt{rb}\} \vdash \mathtt{y} : \mathtt{Id(CK)} \quad \Sigma \vdash \mathtt{x} : \mathtt{Id(Wt)}}{\Sigma, \{\mathtt{rb}\} \vdash \mathtt{x := y} \dashv \{\mathtt{rb}, \mathtt{x}\}} :=F \qquad \Sigma, \{\mathtt{rb}, \mathtt{x}\} \vdash \begin{array}{l}\mathtt{y := z;}\\ \mathtt{w := x + y}\end{array}}{\Sigma, \{\mathtt{rb}\} \vdash \mathtt{x := y;\ y := z;\ w := x + y}} ;F}{\Sigma, \varnothing \vdash \begin{array}{l}\mathtt{rb := rb + 1;\ x := y;}\\ \mathtt{y := z;\ w := x + y}\end{array}} ;F$$

$$\Sigma_0 \vdash \boxed{\begin{array}{l}\mathtt{Ckpt(y)}\\ \quad \mathtt{rb := rb + 1}\\ \quad \mathtt{x := y;}\\ \quad \mathtt{y := z;}\\ \quad \mathtt{w := x + y}\end{array}}$$

$\Sigma = \{\mathtt{rb} : \mathtt{Nid}, \mathtt{x} : \mathtt{Id(Wt)}, \mathtt{y} : \mathtt{Id(CK)}, \mathtt{z} : \mathtt{Id(RD)}, \mathtt{w} : \mathtt{Id(Wt)}\}$

Assignment

$$\dfrac{\dfrac{\dfrac{\Sigma, \{\texttt{rb}, \texttt{x}\} \vdash\ \texttt{y := z; w := x + y}}{\Sigma, \{\texttt{rb}\} \vdash\ \texttt{x := y; y := z;\ w := x + y}}\ ;F}{\Sigma, \emptyset \vdash\ \texttt{rb := rb + 1;\ x := y;}\ \texttt{y := z;\ w := x + y}}\ ;F}{\Sigma_0 \vdash \boxed{\begin{array}{l}\texttt{Ckpt(y)}\\ \quad\texttt{rb := rb + 1}\\ \quad\texttt{x := y;}\\ \quad\texttt{y := z;}\\ \quad\texttt{w := x + y}\end{array}}}$$

$\Sigma = \{\texttt{rb} : \texttt{Nid}, \texttt{x} : \texttt{Id(Wt)}, \texttt{y} : \texttt{Id(CK)}, \texttt{z} : \texttt{Id(RD)}, \texttt{w} : \texttt{Id(Wt)}\}$

Sequencing

$$\frac{\Sigma, \mathrm{Wts} \vdash \alpha \dashv \mathrm{Wts}' \qquad \Sigma, \mathrm{Wts}' \vdash \beta \dashv \mathrm{Wts}''}{\Sigma, \mathrm{Wts} \vdash \alpha;\beta \dashv \mathrm{Wts}''}$$

$$\frac{\dfrac{\dfrac{\dfrac{\cdots}{\Sigma, \{\mathtt{rb}, \mathtt{x}\} \vdash \mathtt{y := z;\ w := x + y} \dashv \{\mathtt{rb}, \mathtt{x}, \mathtt{y}, \mathtt{w}\}}\ ;F}{\Sigma, \{\mathtt{rb}\} \vdash \mathtt{x := y;\ y := z;\ w := x + y}}\ ;F}{\Sigma, \emptyset \vdash \mathtt{rb := rb + 1;\ x := y;\ y := z;\ w := x + y}}\ ;F}{\Sigma_0 \vdash \boxed{\begin{array}{l}\mathtt{Ckpt(y)} \\ \quad \mathtt{rb := rb + 1} \\ \quad \mathtt{x := y;} \\ \quad \mathtt{y := z;} \\ \quad \mathtt{w := x + y}\end{array}}}$$

$\Sigma = \{\mathtt{rb} : \mathtt{Nid}, \mathtt{x} : \mathtt{Id(Wt)}, \mathtt{y} : \mathtt{Id(CK)}, \mathtt{z} : \mathtt{Id(RD)}, \mathtt{w} : \mathtt{Id(Wt)}\}$

Sequencing

$$\frac{\Sigma, \text{Wts} \vdash \alpha \dashv \text{Wts'} \qquad \Sigma, \text{Wts'} \vdash \beta \dashv \text{Wts''}}{\Sigma, \text{Wts} \vdash \alpha; \beta \dashv \text{Wts''}}$$

$$\dfrac{\dfrac{\dfrac{\dots}{\Sigma, \{\texttt{rb}, \texttt{x}\} \vdash \mathrm{y := z;\ w := x + y} \dashv \{\texttt{rb}, \texttt{x}, \texttt{y}, \texttt{w}\}}\ ;F}{\Sigma, \{\texttt{rb}\} \vdash \mathrm{x := y;\ y := z;\ w := x + y} \dashv \{\texttt{rb}, \texttt{x}, \texttt{y}, \texttt{w}\}}\ ;F}{\Sigma, \varnothing \vdash \mathrm{rb := rb + 1;\ x := y;\ y := z;\ w := x + y}}\ ;F$$

$\Sigma_0 \vdash$

```
Ckpt(y)
    rb := rb + 1
    x := y;
    y := z;
    w := x + y
```

$\Sigma = \{\texttt{rb} : \texttt{Nid}, \texttt{x} : \texttt{Id(Wt)}, \texttt{y} : \texttt{Id(CK)}, \texttt{z} : \texttt{Id(RD)}, \texttt{w} : \texttt{Id(Wt)}\}$

Sequencing

$$\frac{\Sigma, \text{Wts} \vdash \alpha \dashv \text{Wts'} \qquad \Sigma, \text{Wts'} \vdash \beta \dashv \text{Wts''}}{\Sigma, \text{Wts} \vdash \alpha; \beta \dashv \text{Wts''}}$$

$$\dfrac{\dfrac{\dfrac{\cdots}{\Sigma, \{\mathtt{rb}, \mathtt{x}\} \vdash \mathrm{y} := \mathrm{z};\ \mathrm{w} := \mathrm{x} + \mathrm{y} \dashv \{\mathtt{rb}, \mathtt{x}, \mathtt{y}, \mathtt{w}\}} ;F}{\Sigma, \{\mathtt{rb}\} \vdash \mathrm{x} := \mathrm{y};\ \mathrm{y} := \mathrm{z};\ \mathrm{w} := \mathrm{x} + \mathrm{y} \dashv \{\mathtt{rb}, \mathtt{x}, \mathtt{y}, \mathtt{w}\}} ;F}{\Sigma, \varnothing \vdash \mathrm{rb} := \mathrm{rb} + 1;\ \mathrm{x} := \mathrm{y};\ \mathrm{y} := \mathrm{z};\ \mathrm{w} := \mathrm{x} + \mathrm{y} \dashv \{\mathtt{rb}, \mathtt{x}, \mathtt{y}, \mathtt{w}\}} ;F$$

$\Sigma_0 \vdash$

```
Ckpt(y)
    rb := rb + 1
    x := y;
    y := z;
    w := x + y
```

$\Sigma = \{\mathtt{rb} : \mathtt{Nid}, \mathtt{x} : \mathtt{Id(Wt)}, \mathtt{y} : \mathtt{Id(CK)}, \mathtt{z} : \mathtt{Id(RD)}, \mathtt{w} : \mathtt{Id(Wt)}\}$

The program is correct

$$\dfrac{\dfrac{\dfrac{\dfrac{\dots}{\Sigma, \{\texttt{rb}, \texttt{x}\} \vdash \texttt{y := z; w := x + y} \dashv \{\texttt{rb}, \texttt{x}, \texttt{y}, \texttt{w}\}}\,;F}{\Sigma, \{\texttt{rb}\} \vdash \texttt{x := y; y := z; w := x + y} \dashv \{\texttt{rb}, \texttt{x}, \texttt{y}, \texttt{w}\}}\,;F}{\Sigma, \varnothing \vdash \texttt{rb := rb + 1; x := y; y := z; w := x + y} \dashv \{\texttt{rb}, \texttt{x}, \texttt{y}, \texttt{w}\}}\,;F$$

$$\Sigma_0 \vdash \boxed{\begin{array}{l}\texttt{Ckpt(y)} \\ \quad \texttt{rb := rb + 1} \\ \quad \texttt{x := y;} \\ \quad \texttt{y := z;} \\ \quad \texttt{w := x + y}\end{array}}$$

correct

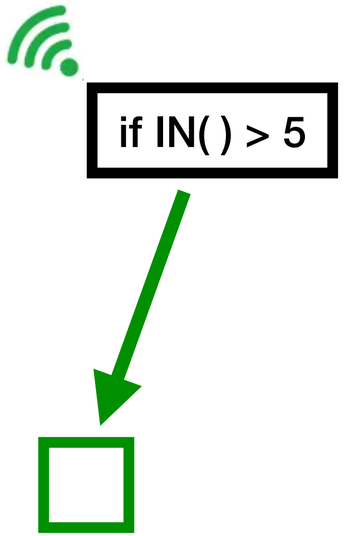
all writes have been written $x, w \in \text{Wts}$

$\Sigma = \{\texttt{rb} : \texttt{Nid}, \texttt{x} : \texttt{Id(Wt)}, \texttt{y} : \texttt{Id(CK)}, \texttt{z} : \texttt{Id(RD)}, \texttt{w} : \texttt{Id(Wt)}\}$

* pause for questions *

But what about sensor inputs?

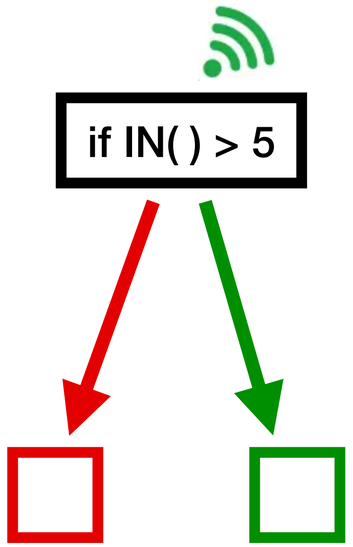
```
Ckpt( )
  let x = In( ) in
    if x > 5 then
      y := 1
    else
      y := 0
```



commands $cmd ::= \ldots \mid$ let $x = \text{In}()$ in c

What about sensor inputs?

```
Ckpt( )
  let x = In( ) in
    if x > 5 then
      y := 1
    else
      y := 0
```



commands $cmd ::= \ldots \mid \text{let } x = \text{In}() \text{ in } c$

Example: I/O Dependent Branching

```
1  Ckpt( )
2    let x = In( ) in
3      if x > 5 then
4        y := 1
5      else
6        y := 0
```

NV	x	y
1	0	0

Example: I/O Dependent Branching

```
1  Ckpt( )
2    let x = In( ) in
3      if x > 5 then
4        y := 1
5      else
6        y := 0
```

NV	x	y
1	0	0
2	**6**	0

Example: I/O Dependent Branching

```
1  Ckpt( )
2    let x = In( ) in
3      if x > 5 then
4        y := 1
5      else
6        y := 0
```

NV	x	y
1	0	0
2	**6**	0
3	6	0

Example: I/O Dependent Branching

```
1  Ckpt( )
2    let x = In( ) in
3      if x > 5 then
4        y := 1
5      else
6        y := 0
```

NV	x	y
1	0	0
2	**6**	0
3	6	0
4	6	**1**

Example: I/O Dependent Branching

```
1  Ckpt( )
2    let x = In( ) in
3      if x > 5 then
4        y := 1
5      else
6        y := 0
```

NV	x	y
1	0	0
2	**6**	0
3	6	0
4	6	**1**

Example: I/O Dependent Branching

```
1  Ckpt( )
2    let x = In( ) in
3      if x > 5 then
4        y := 1
5      else
6        y := 0
```

NV	x	y
1	0	0
2	**6**	0
3	6	0
4	6	**1**
reboot	6	**1**
2	**3**	1

Example: I/O Dependent Branching

```
1  Ckpt( )
2    let x = In( ) in
3      if x > 5 then
4        y := 1
5      else
6        y := 0
```

NV	x	y
1	0	0
2	**6**	0
3	6	0
4	6	**1**
reboot	6	**1**
2	**3**	1
5	3	1

Example: I/O Dependent Branching

```
1  Ckpt( )
2    let x = In( ) in
3      if x > 5 then
4        y := 1
5      else
6        y := 0
```

NV	x	y
1	0	0
2	**6**	0
3	6	0
4	6	**1**
reboot	6	**1**
2	**3**	1
5	3	1
6	**3**	**0**

Example: I/O Dependent Branching

```
1  Ckpt( )
2    let x = In( ) in
3      if x > 5 then
4        y := 1
5      else
6        y := 0
```

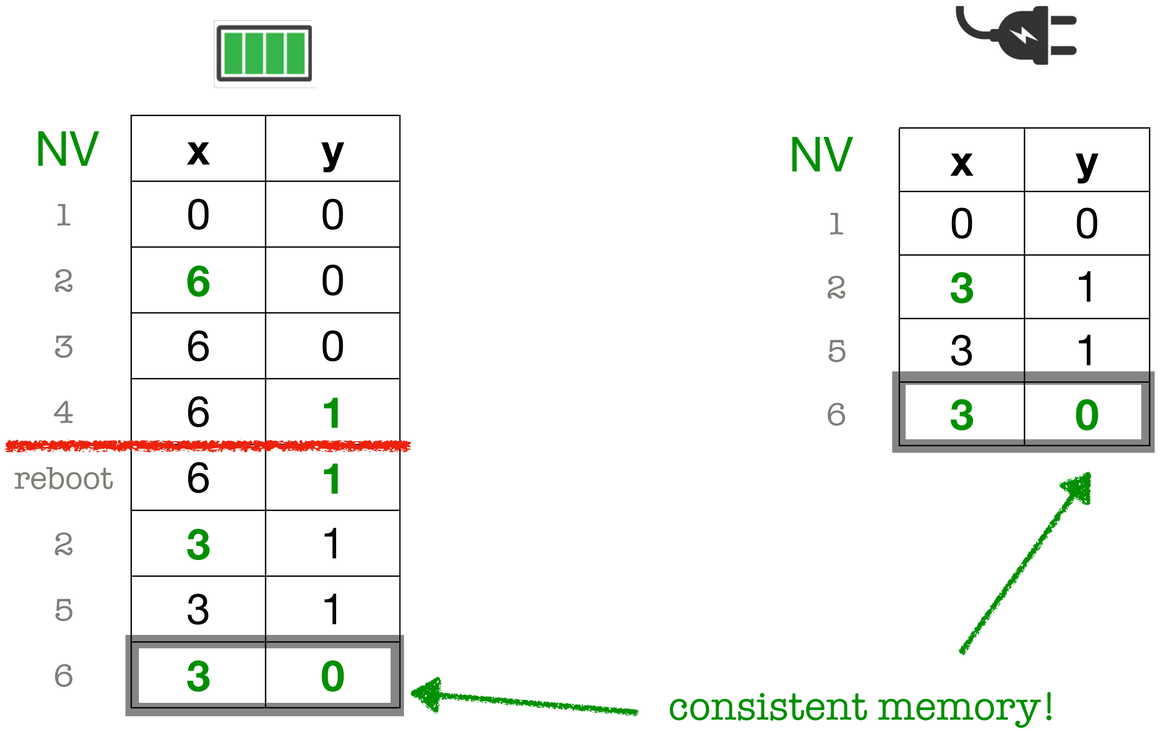
NV	x	y
1	0	0
2	**6**	0
3	6	0
4	6	**1**
reboot	6	**1**
2	**3**	1
5	3	1
6	**3**	**0**

NV	x	y
1	0	0
2	**3**	1
5	3	1
6	**3**	**0**

last successful execution!

Example: I/O Dependent Branching

```
1  Ckpt( )
2    let x = In( ) in
3      if x > 5 then
4        y := 1
5      else
6        y := 0
```



x and y should be allowed to differ across intermediate states!

New taint qualifiers!

basic types	$T := \texttt{int} \mid \texttt{bool}$
type qualifiers	$tq := \langle q, qIO \rangle$
idempotence qualifiers	$q := \mathsf{Id}(qAcc) \mid \mathsf{Nid} \mid \mathsf{Id}$
access qualifiers	$qAcc := \texttt{CK} \mid \texttt{RD} \mid \texttt{Wt}$
taint qualifiers	$qIO := \mathsf{Tnt} \mid \mathsf{Nt}$

Tnt	“tainted”
Nt	“not tainted”

New taint qualifiers!

basic types	$T := \texttt{int} \mid \texttt{bool}$
type qualifiers	$tq := \langle q, qIO \rangle$
idempotence qualifiers	$q := \mathsf{Id}(qAcc) \mid \mathsf{Nid} \mid \mathsf{Id}$
access qualifiers	$qAcc := \texttt{CK} \mid \texttt{RD} \mid \texttt{Wt}$
taint qualifiers	$qIO := \mathsf{Tnt} \mid \mathsf{Nt}$

```
Ckpt( )
  let x = In( ) in
    if x > 5 then
      y := 1
    else
      y := 0
```

x : int@<Id(Wt), Tnt>

y : int@<Id(Wt), Tnt>

New taint qualifiers!

basic types	$T := \texttt{int} \mid \texttt{bool}$
type qualifiers	$tq := \langle q, qIO \rangle$
idempotence qualifiers	$q := \mathsf{Id}(qAcc) \mid \mathsf{Nid} \mid \mathsf{Id}$
access qualifiers	$qAcc := \texttt{CK} \mid \texttt{RD} \mid \texttt{Wt}$
taint qualifiers	$qIO := \mathsf{Tnt} \mid \mathsf{Nt}$

```
Ckpt( )
  let x = In( ) in
    if x > 5 then
      y := 1
    else
      y := 0
```

x : int@<Id(Wt), Tnt>

y : int@<Id(Wt), Tnt>

Qualifier structure

Tnt variables should not influence Nt variables. Nt ⊏ Tnt

x:Tnt, y:Tnt, z:Nt

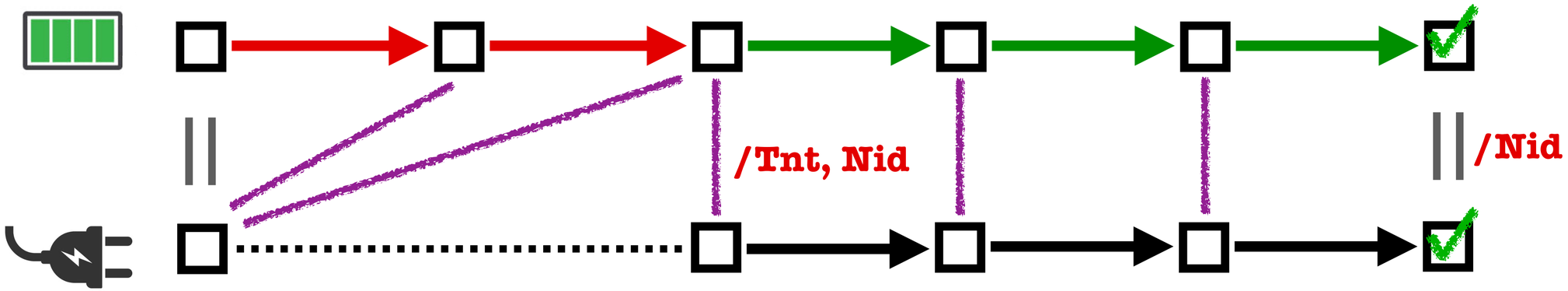
x := y 

z := x 

```
if x > 5 then
  y := 1
else
  y := 0
```

```
if x > 5 then
  z := 1
else
  z := 0
```

Correctness in terms of noninterference



correctness

We define that program α satisfies ~~*noninterference*~~ *with respect to policy* Σ iff for all ω_1, ω_2, ν_1, and ν_2,

$\Sigma \vdash \omega_1 \approx_{\not L} \omega_2$, eval $\omega_1\ \alpha = \nu_1$, and eval $\omega_2\ \alpha = \nu_2$ implies $\Sigma \vdash \nu_1 \approx_{\not L} \nu_2$.

Id(_) *Id(_)*

Typing Judgments

values $\Sigma \vdash v : T@tq$

expressions $\Sigma \vdash e : T@tq$

commands $\Sigma, \mathsf{Wts} \vdash cmd \dashv \mathsf{Wts}'$

programs $\Sigma \vdash prog$ correct

Type Checking Values

$$\boxed{\Sigma \vdash v : T @ tq}$$

$$\frac{}{\Sigma \vdash \mathsf{true} : \mathtt{bool} @ \langle \mathtt{Id}, \mathtt{Nt} \rangle}\ \textit{true-C}$$

$$\frac{}{\Sigma \vdash \mathsf{false} : \mathtt{bool} @ \langle \mathtt{Id}, \mathtt{Nt} \rangle}\ \textit{false-C}$$

$$\frac{}{\Sigma \vdash n : \mathtt{int} @ \langle \mathtt{Id}, \mathtt{Nt} \rangle}\ \textit{int-C}$$

Type Checking Expressions

$$\boxed{\Sigma \vdash e : T @ tq}$$

$$\frac{\Sigma(x) = T @ tq}{\Sigma \vdash x : T @ tq}\ \textit{var-C} \qquad \frac{\Sigma \vdash e : T @ tq}{\Sigma \vdash \oslash\, e : T @ tq}\ \textit{unop-C}$$

$$\frac{\Sigma \vdash e_1 : T @ tq_1 \quad \Sigma \vdash e_2 : T @ tq_2}{\Sigma \vdash e_1 \odot e_2 : T @ tq_1 \sqcup tq_2}\ \textit{binop-C}$$

Type Checking Expressions

$$\boxed{\Sigma \vdash e : T@tq}$$

$$\frac{\Sigma(x) = T@tq}{\Sigma \vdash x : T@tq}\ \textit{var-C} \qquad \frac{\Sigma \vdash e : T@tq}{\Sigma \vdash \oslash\, e : T@tq}\ \textit{unop-C}$$

$$\frac{\Sigma \vdash e_1 : T@tq_1 \quad \Sigma \vdash e_2 : T@tq_2}{\Sigma \vdash e_1 \odot e_2 : T@tq_1 \sqcup tq_2}\ \textit{binop-C}$$

where $tq_1 \sqcup tq_2 = \langle q_1 \sqcup q_2, qIO_1 \sqcup qIO_2 \rangle$

$$tq_1 = \langle q_1, qIO_1 \rangle$$

$$tq_2 = \langle q_2, qIO_2 \rangle$$

Type Checking Commands: Assignment

$\Sigma \vdash \alpha$ secure

$$\frac{\Sigma \vdash e : \ell \quad \ell' = \Sigma(pc) \sqcup \ell \quad \ell' \sqsubseteq \Sigma(x)}{\Sigma \vdash x := e \text{ secure}} :=F$$

Σ, Wts $\vdash$ c $\dashv$ Wts'

$$\Sigma \vdash x : T@tq_x$$

if tq_x = <Id(q), qIO> then q < Wt

$$\frac{\Sigma \vdash e : T@tq \quad tq' = \Sigma(pc) \sqcup tq \quad tq' \sqsubseteq tq_x}{\Sigma, \text{Wts} \vdash x := e \dashv \text{Wts} \cup \{x\}} :=F$$

Type Checking Commands: Assignment

$\Sigma \vdash \alpha$ secure

$$\frac{\Sigma \vdash e : \ell \quad \ell' = \Sigma(pc) \sqcup \ell \quad \ell' \sqsubseteq \Sigma(x)}{\Sigma \vdash x := e \text{ secure}} :=F$$

Σ, Wts $\vdash$ c $\dashv$ Wts'

$$\Sigma \vdash x : T@tq_x$$

if tq_x = <Id(q), qIO> then q < Wt

$$\frac{\Sigma \vdash e : T@tq \quad tq' = \Sigma(pc) \sqcup tq \quad tq' \sqsubseteq tq_x}{\Sigma, \text{Wts} \vdash x := e \dashv \text{Wts} \cup \{x\}} :=F$$

where $tq_1 \sqsubseteq tq_2$ means $q_1 \sqsubseteq q_2$ and $qIO_1 \sqsubseteq qIO_2$

$tq_1 = \langle q_1, qIO_1 \rangle$ $\qquad$ $tq_2 = \langle q_2, qIO_2 \rangle$

Type Checking Commands: Sequencing

$\Sigma \vdash \alpha$ secure

$$\frac{\Sigma \vdash \alpha \text{ secure} \quad \Sigma \vdash \beta \text{ secure}}{\Sigma \vdash \alpha \,;\, \beta \text{ secure}} \;;F$$

$\Sigma,$ Wts $\vdash c \dashv$ Wts'

$$\frac{\Sigma, \text{Wts} \vdash c_1 \dashv \text{Wts}' \qquad \Sigma, \text{Wts}' \vdash c_2 \dashv \text{Wts}''}{\Sigma, \text{Wts} \vdash c_1; c_2 \dashv \text{Wts}''} \;;F$$

same as before

Type Checking Commands: If-then-else

$\Sigma \vdash \alpha$ secure

$$\frac{\Sigma \vdash P : \ell \quad \ell' = \Sigma(pc) \sqcup \ell \quad \Sigma' = \Sigma[pc \mapsto \ell'] \quad \Sigma' \vdash \alpha \text{ secure} \quad \Sigma' \vdash \beta \text{ secure}}{\Sigma \vdash \textbf{if } P \textbf{ then } \alpha \textbf{ else } \beta \text{ secure}} \textbf{ if}F$$

Σ, Wts $\vdash c \dashv$ Wts'

$$\frac{\Sigma \vdash e : \texttt{bool}@tq \quad tq' = \Sigma(pc) \sqcup tq \quad \Sigma' = \Sigma[pc \mapsto tq'] \quad \Sigma', \text{Wts} \vdash c_1 \dashv \text{Wts}' \quad \Sigma', \text{Wts} \vdash c_2 \dashv \text{Wts}'}{\Sigma, \text{Wts} \vdash \textbf{if } e \textbf{ then } c_1 \textbf{ else } c_2 \dashv \text{Wts}'} \textbf{ if}F$$

Updated qualifiers help us rule out implicit flows for sensor inputs!

Type Checking Programs

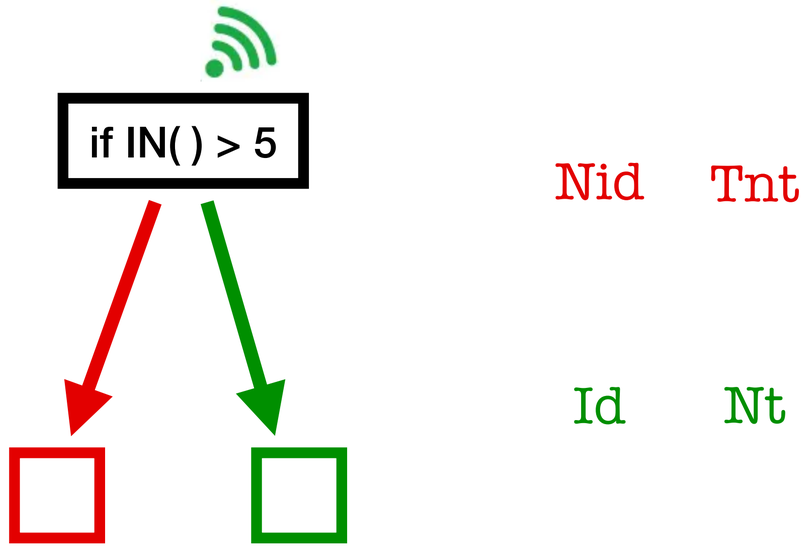
$\Sigma \vdash prog$ correct

$$\frac{\begin{array}{c}\Sigma_{\text{init}}(pc) = \langle \texttt{Id}, \texttt{Nt} \rangle \\ \forall x \in nIds\,.\,\Sigma_{\text{init}}(x) = T@\langle \texttt{Nid}, \texttt{Nt} \rangle \text{ where } \Gamma(x) = T@tq' \\ \forall x \in ckvars\,.\,\Sigma_{\text{init}}(x) = T@\langle \texttt{Id(CK)}, \texttt{Nt} \rangle \text{ where } \Gamma(x) = T@tq' \\ \forall x \in rds\,.\,\Sigma_{\text{init}}(x) = T@\langle \texttt{Id(RD)}, \texttt{Nt} \rangle \text{ where } \Gamma(x) = T@tq' \\ \forall x \in mtwt\,.\,\Sigma_{\text{init}}(x) = T@\langle \texttt{Id(Wt)}, \texttt{Nt} \rangle \text{ where } \Gamma(x) = T@tq' \\ \Sigma_{\text{init}}, \cdot \vdash c \dashv \text{Wts} \\ \forall x \text{ s.t. } \Sigma_{\text{init}}(x) = T@\langle \texttt{Id(Wt)}, \texttt{Nt} \rangle, x \in \text{Wts}\end{array}}{\Sigma \vdash \text{Ckpt}(aID, ckvars, rds, mtwt, nIds)\ c \text{ correct}}\ \textit{Ckpt-C}$$

initialize qualifiers to be untainted

See paper for other interesting features!

- pointers and functions
- branches with different write sets
- just-in-time mode
- type inference
- practical implementations



A Type System for Safe Intermittent Computing
[Surbatovich et al. '23]