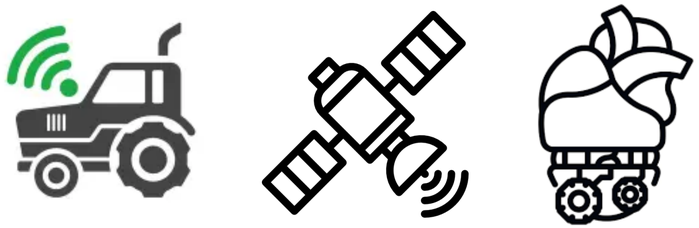
Modal Crash Types for Intermittent Computing

Myra Dotzel
PhD student, CSD, CMU

Joint work with Farzaneh Derakhshan, Milijana Surbatovich, and Limin Jia

Intermittent Computing

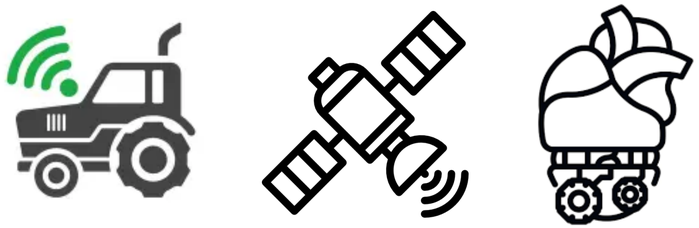
- tiny devices that compute in inaccessible environments



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

Intermittent Computing

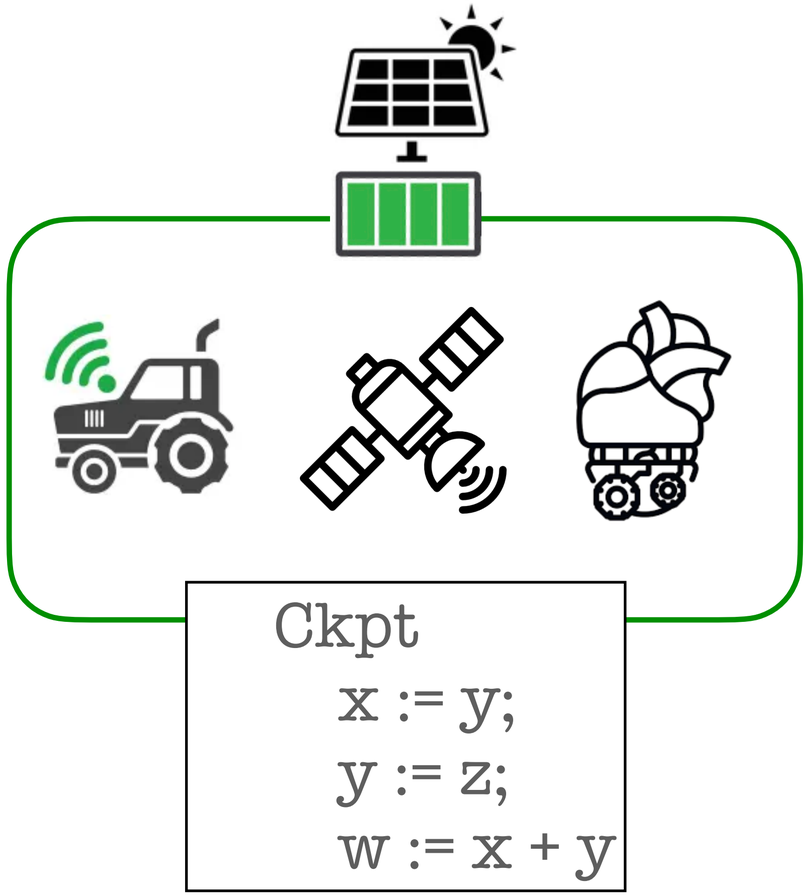
- tiny devices that compute in inaccessible environments
- cannot rely on continuous energy sources



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

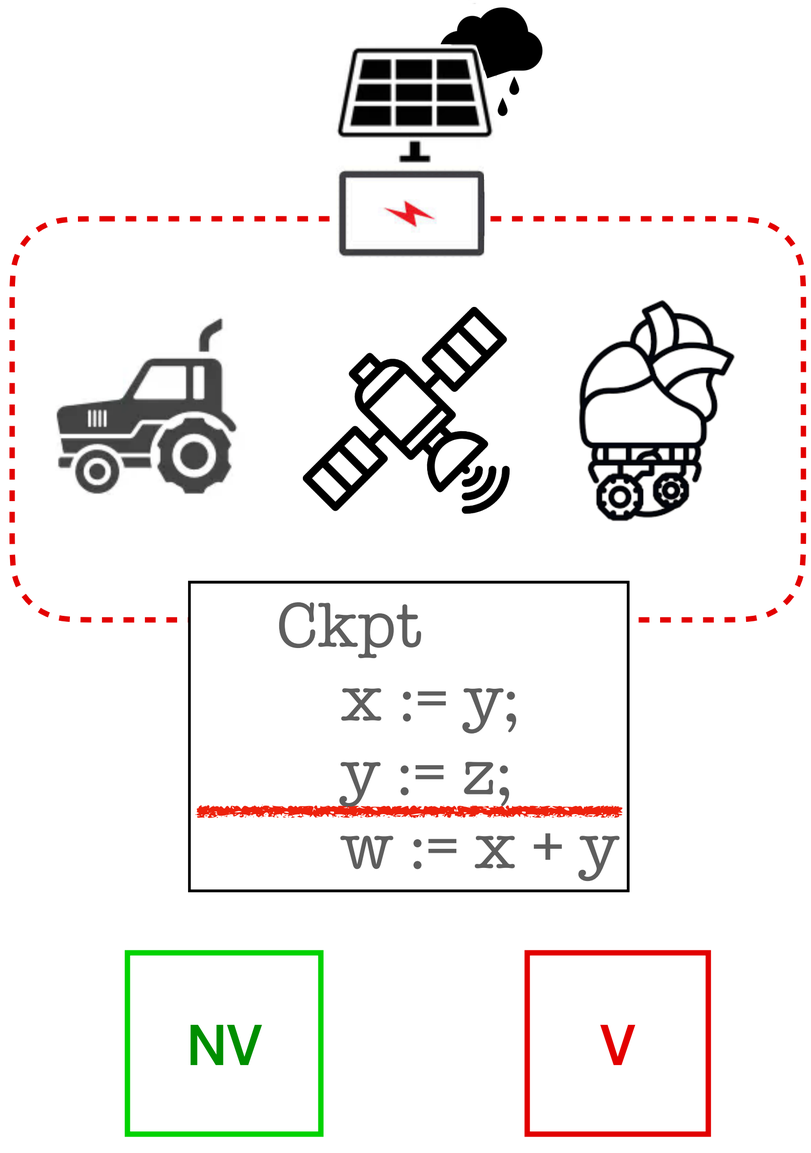
Intermittent Computing

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

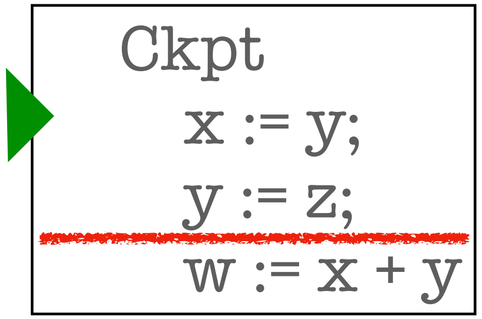


Intermittent Computing

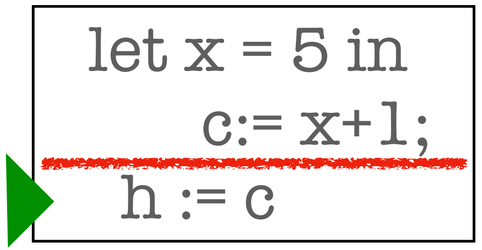
- tiny devices that compute in inaccessible environments
- cannot rely on continuous energy sources
- charge, compute, and crash frequently
- rely on a combination of NV and V
- NV survives power failures, V does not
- runtime system support



Checkpointing depends on execution mode



Atomic region



JIT region

Checkpointing depends on execution mode

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

Atomic region

```
let x = 5 in
    c:= x+1;
  h := c
```

JIT region

Atomic regions checkpoint ahead of time.

Checkpointing depends on execution mode

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

Atomic region

```
let x = 5 in
    c:= x+1;
h := c
```

JIT region

Atomic regions checkpoint ahead of time.

JIT regions checkpoint just in time.

Checkpointing depends on execution mode

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

Atomic r

```
let x = 3 in
    c:= x+1;
    h := c
```

JIT region

ime.

Question:

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

JIT regions checkpoint just in time.

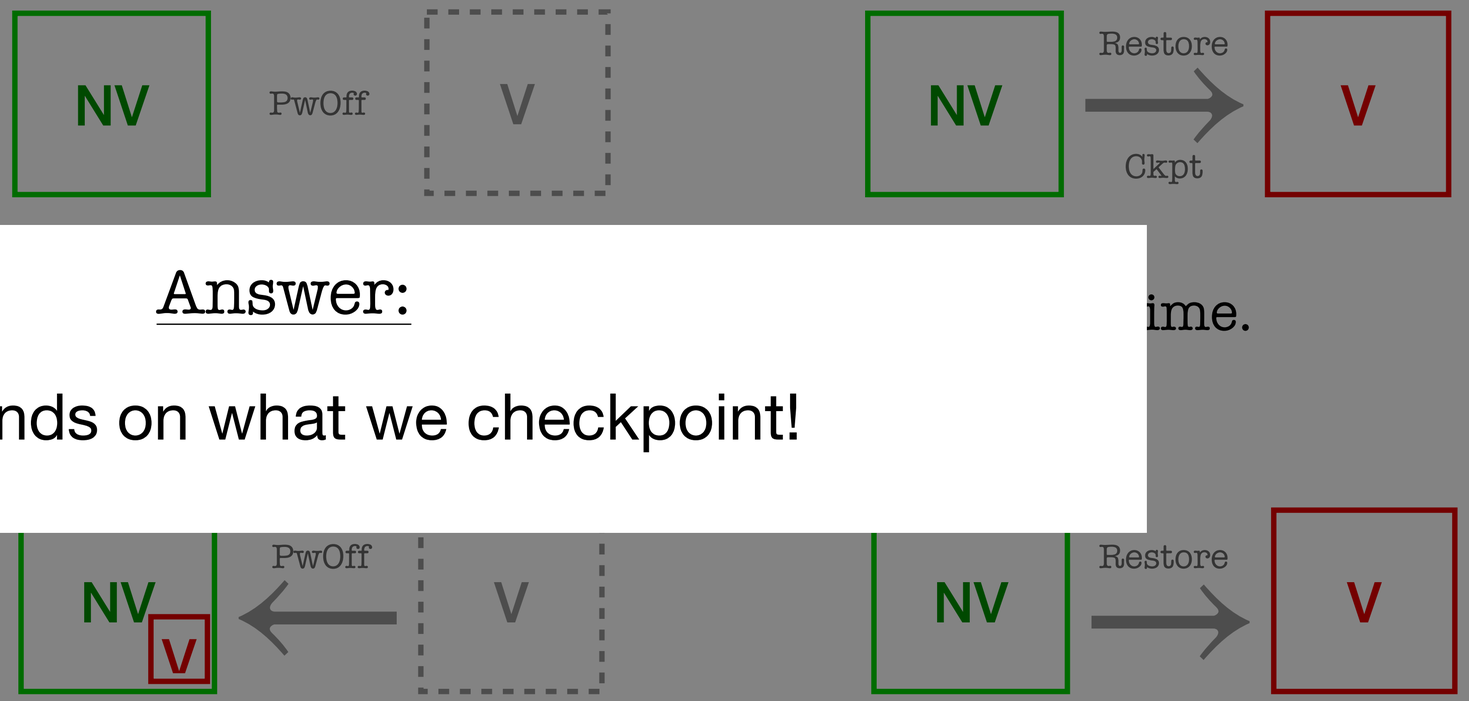
Checkpointing depends on execution mode

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

Atomic r

```
let x = 3 in
    c:= x+1;
h := c
```

JIT region



ime.

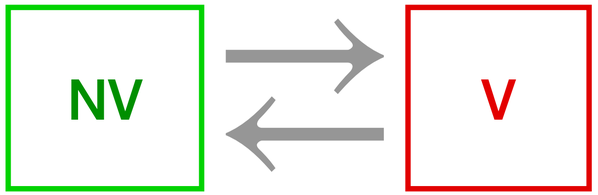
Answer:

It depends on what we checkpoint!

JIT regions checkpoint just in time.

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

Ckpt →

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**

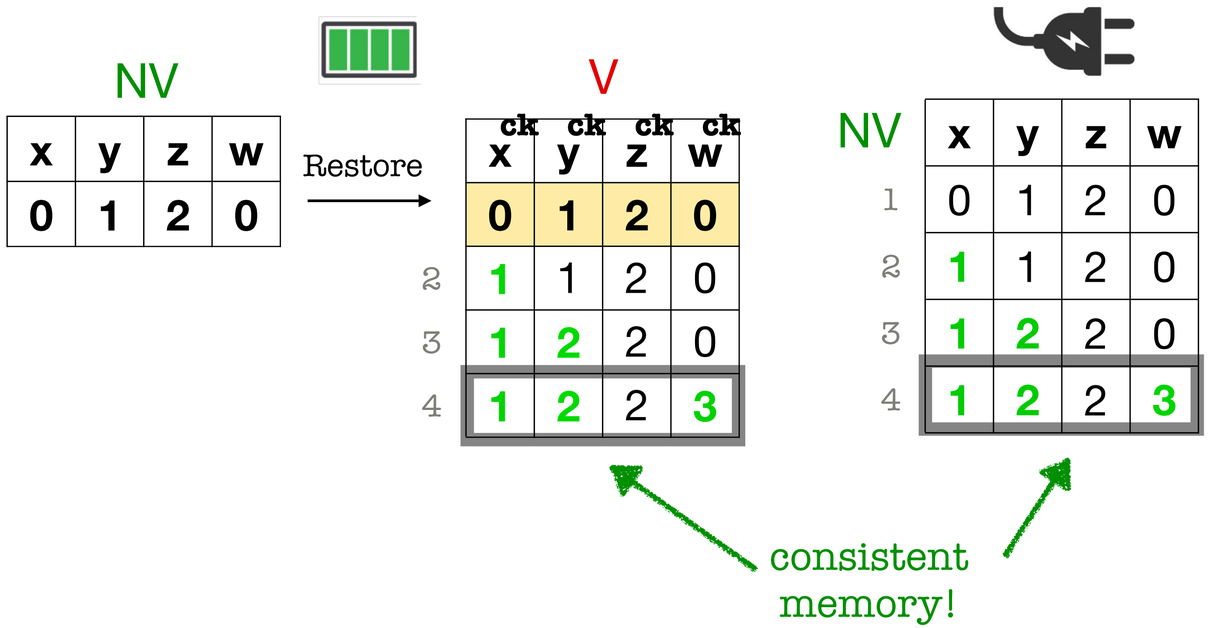
nothing

↬ consistency bugs!

everything

Checkpoint everything!

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



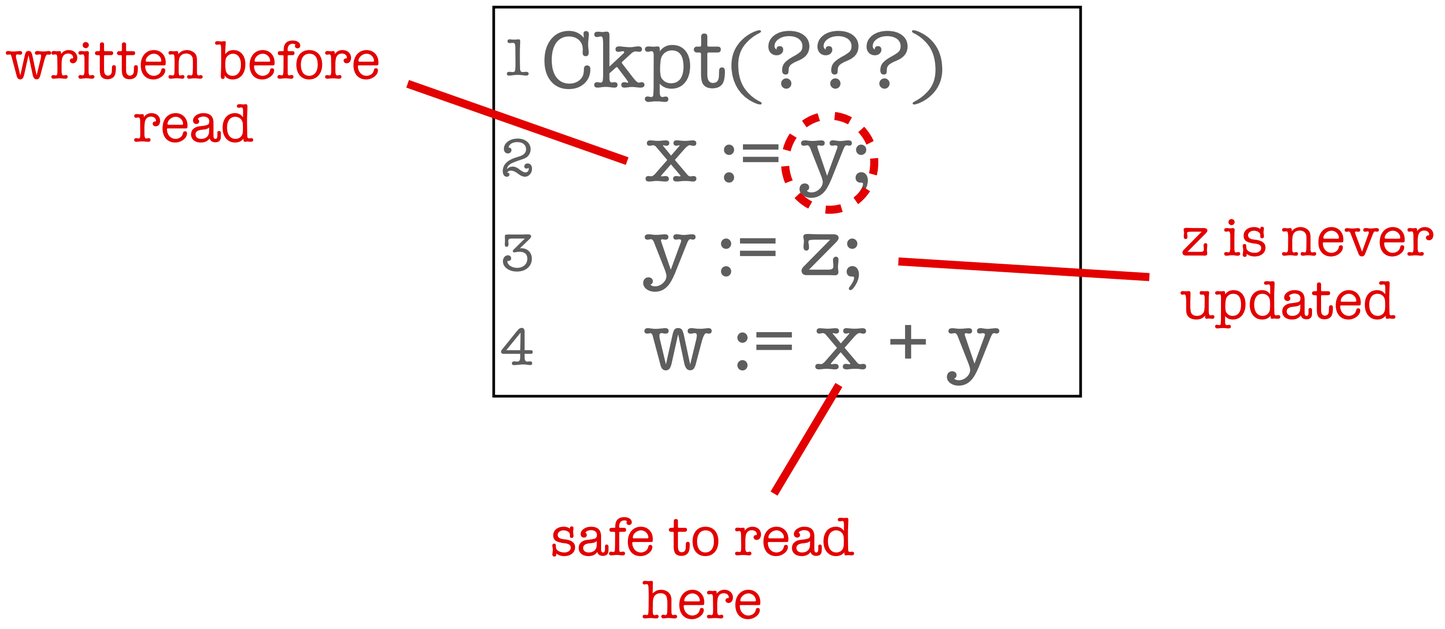
nothing

↳ consistency bugs!

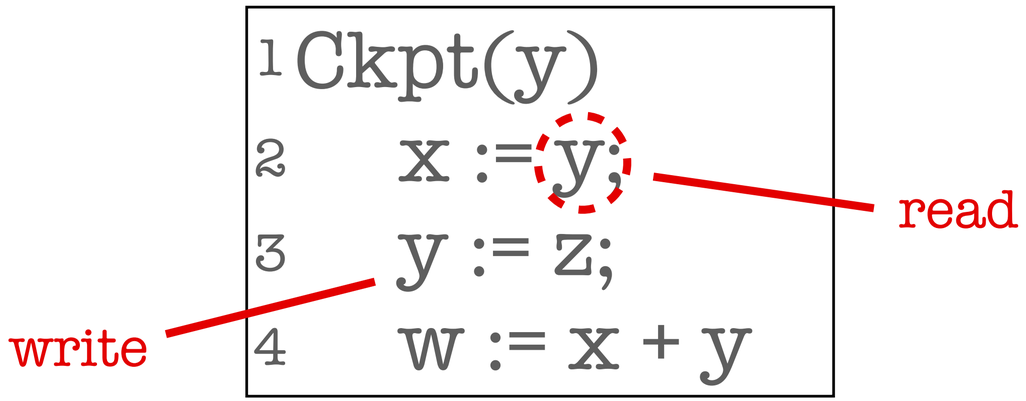
everything

↳ inefficient!

Do we really need to checkpoint everything?



No, 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

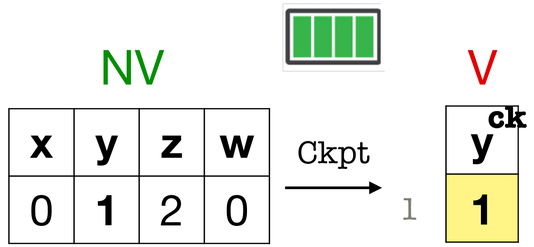
↬ inefficient!

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

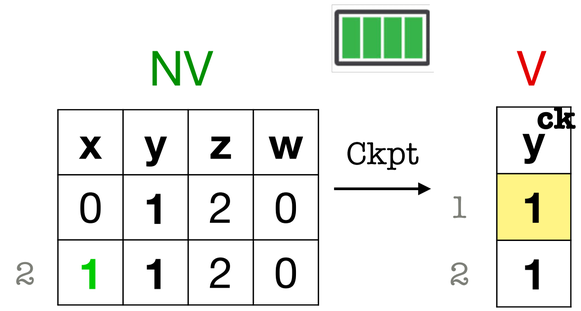
↬ inefficient!

write-after-read variables

✓ real systems

Checkpoint write-after-read variables

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



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

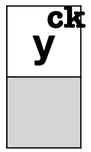
Checkpoint write-after-read variables

```
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

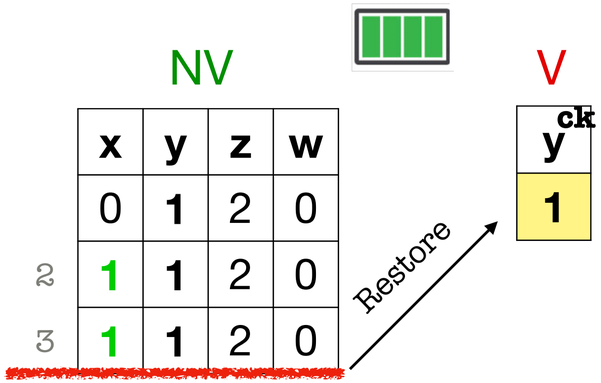
↬ inefficient!

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

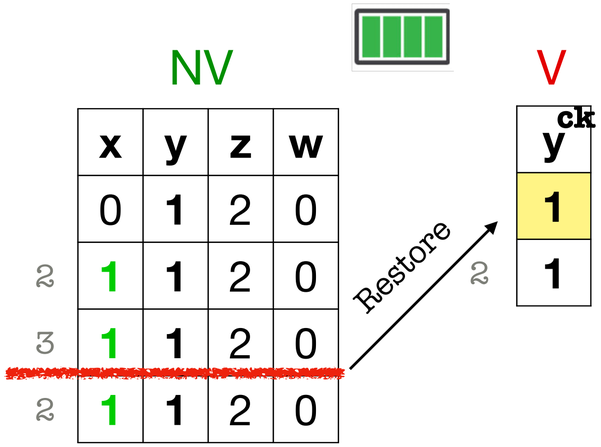
↬ inefficient!

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

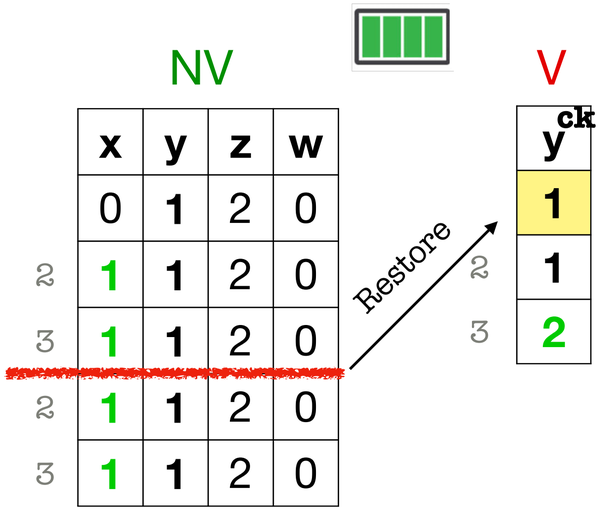
↬ inefficient!

write-after-read variables

✓ real systems

Checkpoint write-after-read variables

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



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

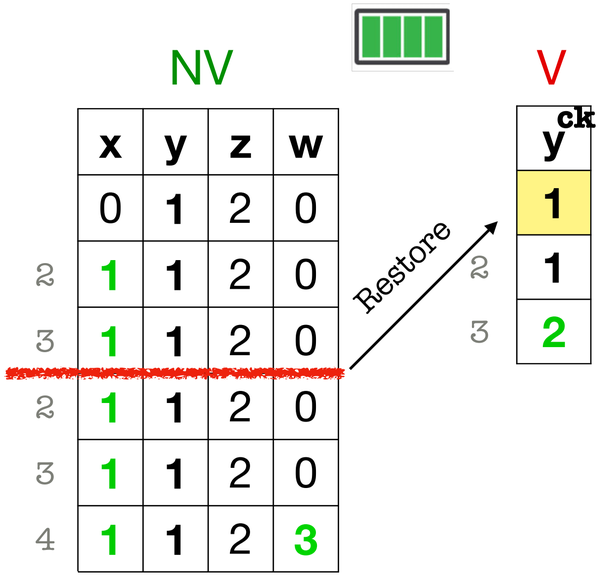
↝ inefficient!

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

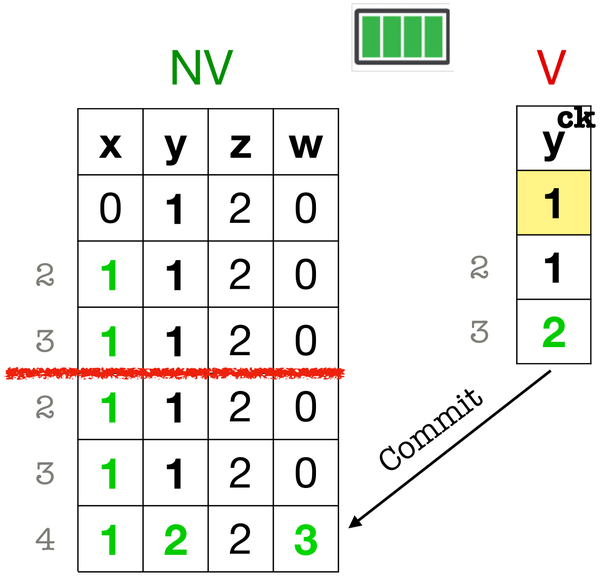
↬ inefficient!

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

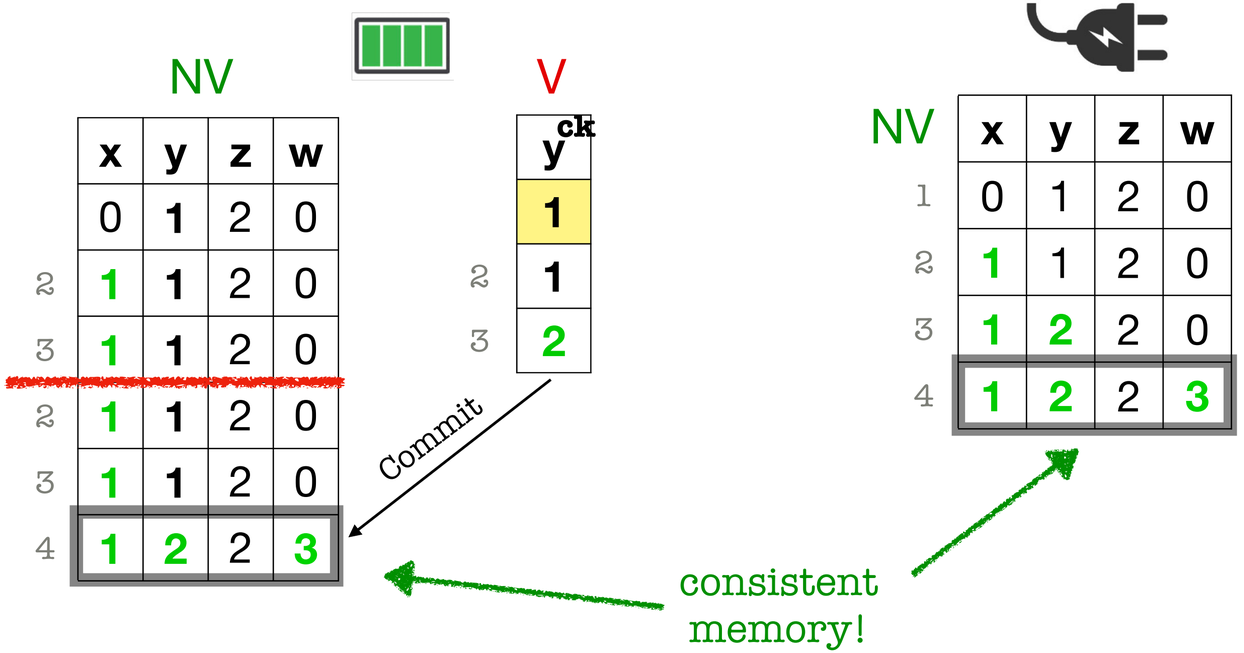
↬ inefficient!

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

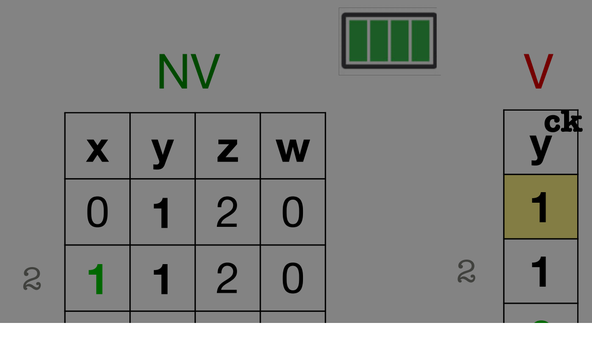
↬ inefficient!

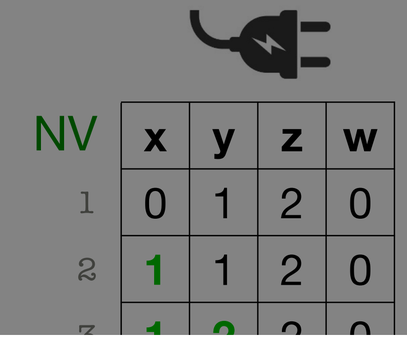
write-after-read variables

✓ real systems

Checkpoint write-after-read variables

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





nothing

↯ consistency bugs!

…thing

inefficient!

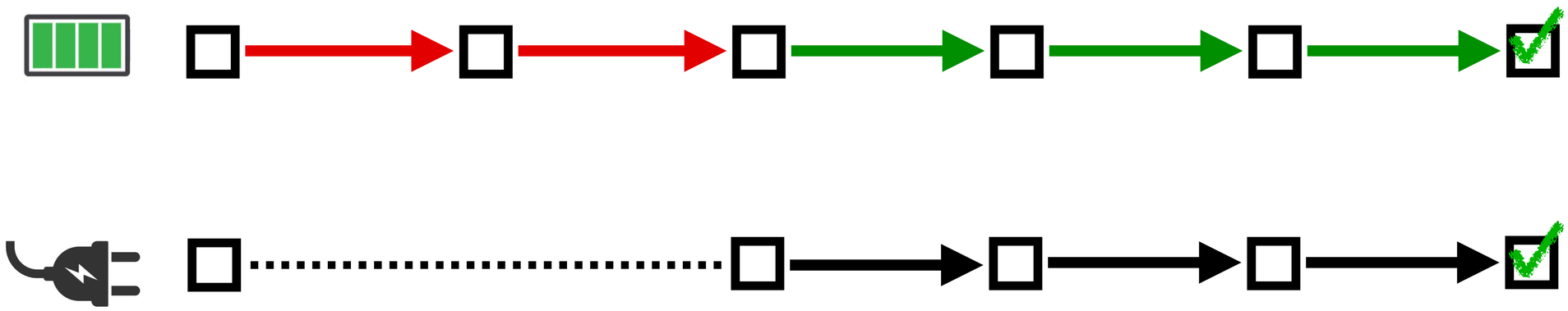
…-after-read
…ables

✓ real systems

More precisely:

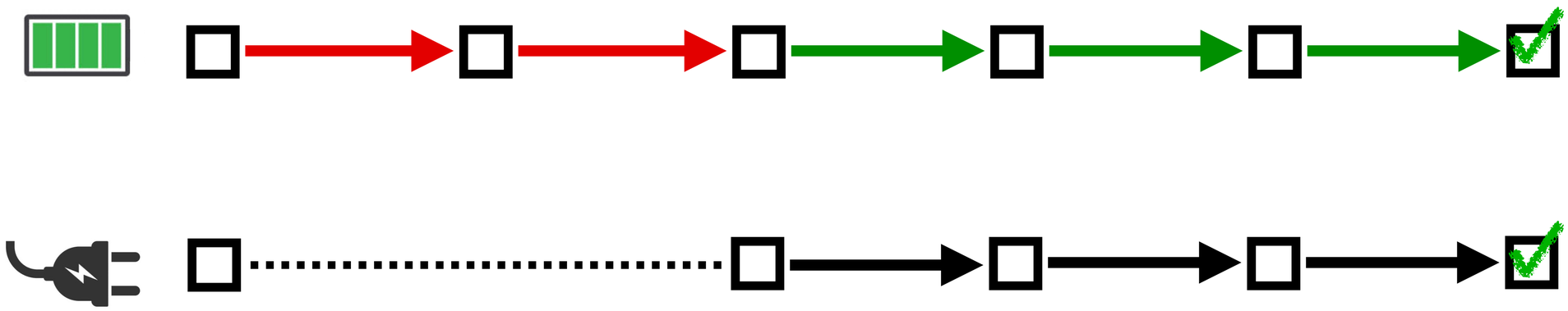
How can we **prove** that programs execute correctly intermittently when checkpointing only WAR variables?

Ensuring correctness



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

Ensuring correctness



- first formal framework for reasoning about intermittent execution [Surbatovich et al. ’20]
- type system focused on correct variable accesses [Surbatovich et al. ’23]

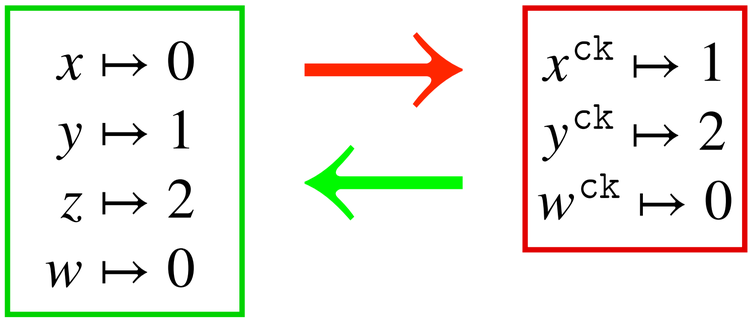
First logical interpretation of intermittent computing

power off, restore, commit.

First logical interpretation of intermittent computing

power off, restore, commit.

One approach: checkpoint everything not read-only [Derakhshan et al. '23]

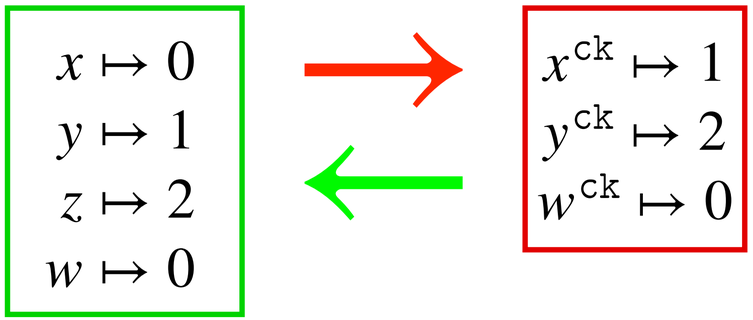



```
Ckpt(x,y,w)
        x := y;
        y := z;
        w := x + y
```

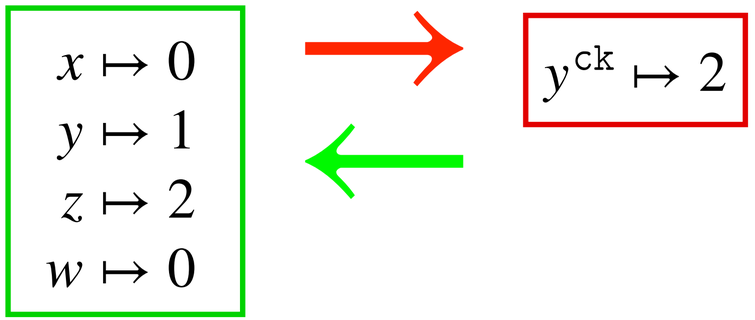
First logical interpretation of intermittent computing

power off, restore, commit.

<u>One approach:</u> checkpoint everything not read-only [Derakhshan et al. '23]



<u>More efficient:</u> checkpoint only write-after-read variables




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

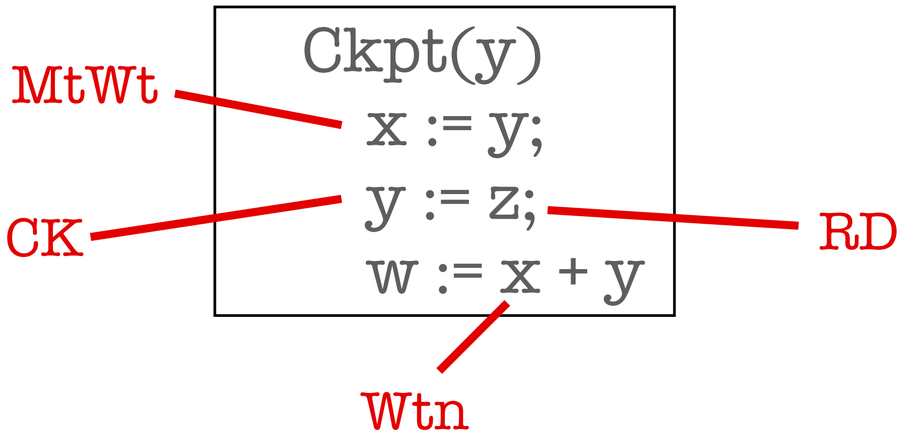
Types allow us to automatically check that programs can execute **correctly intermittently**.

Outline

- Intermittent computing
- Overview
- **Type system**
- Logical relation
- JIT mode
- Key theorems
- Conclusion

Basic types and access qualifiers

basic types	$T := \texttt{int} \mid \texttt{bool} \mid \texttt{unit}$
access qualifiers	$q := \texttt{CK} \mid \texttt{RD} \mid \texttt{MtWt} \mid \texttt{Wtn}$
unstable types	$\tau^u := T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid v_t$
stable types	$\tau^s := \uparrow \tau^u \mid$  $\rightsquigarrow \tau^s$



CK — checkpointed

RD — read only

MtWt — must-first-write

Wtn — written

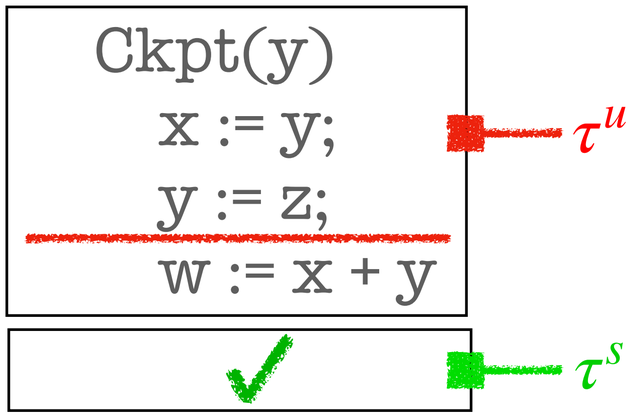
Stable and unstable types

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

access qualifiers $q := \texttt{CK} \mid \texttt{RD} \mid \texttt{MtWt} \mid \texttt{Wtn}$

unstable types $\tau^u := T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid v_t$

stable types $\tau^s := \uparrow \tau^u \mid$  $\rightsquigarrow \tau^s$

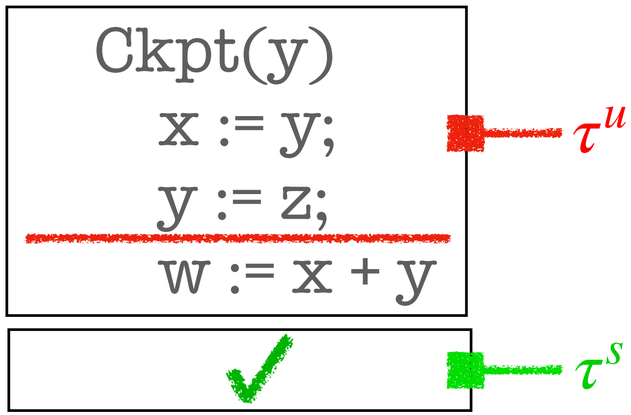


τ^u code inside of checkpoints

τ^s programs after commit

Adjoint Logic*

basic types	$T := \texttt{int} \mid \texttt{bool} \mid \texttt{unit}$
access qualifiers	$q := \texttt{CK} \mid \texttt{RD} \mid \texttt{MtWt} \mid \texttt{Wtn}$
unstable types	$\tau^u := T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid v_t$
stable types	$\tau^s := \uparrow \tau^u \mid$  $\rightsquigarrow \tau^s$

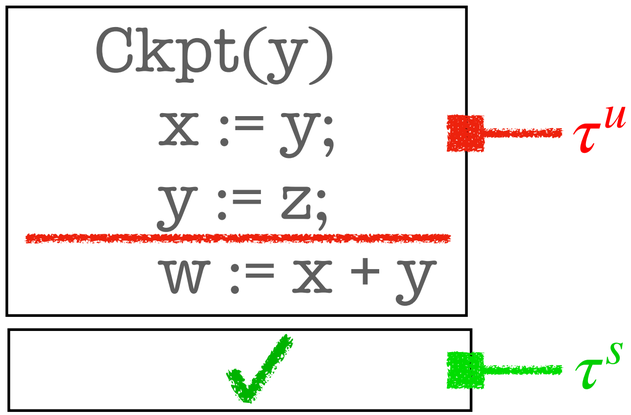


Independence principle: universal truth does not depend on ephemeral truth.

* [Pruiksma et al. '21, Pfenning et al. '01, Benton et al. '97]

Adjoint Logic*

basic types	$T := \texttt{int} \mid \texttt{bool} \mid \texttt{unit}$
access qualifiers	$q := \texttt{CK} \mid \texttt{RD} \mid \texttt{MtWt} \mid \texttt{Wtn}$
unstable types	$\tau^u := T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid \nu_t$
stable types	$\tau^s := \uparrow \tau^u \mid$  $\rightsquigarrow \tau^s$



Independence principle: universal truth does not depend on ephemeral truth.

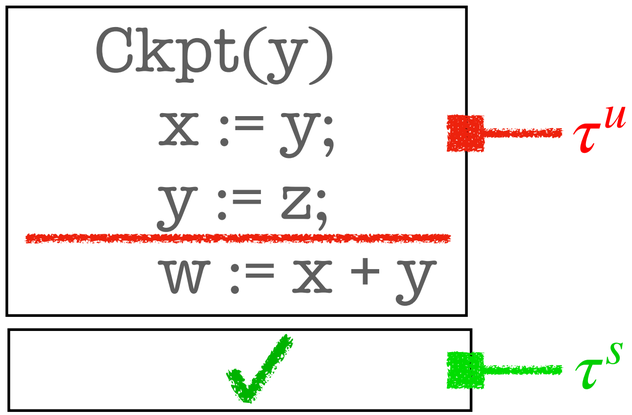
"It is *always* cold"

"It is cold *today*"

* [Pruiksma et al. '21, Pfenning et al. '01, Benton et al. '97]

Adjoint Logic*

basic types	$T := \texttt{int} \mid \texttt{bool} \mid \texttt{unit}$
access qualifiers	$q := \texttt{CK} \mid \texttt{RD} \mid \texttt{MtWt} \mid \texttt{Wtn}$
unstable types	$\tau^u := T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid v_t$
stable types	$\tau^s := \uparrow \tau^u \mid \blacksquare \rightsquigarrow \tau^s$



Independence principle: universal truth does not depend on ephemeral truth.

"It is *always* cold"

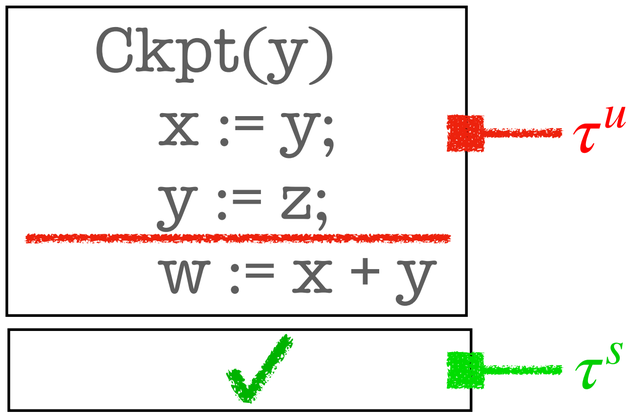
↑ ↓

"It is cold *today*"

* [Pruiksma et al. '21, Pfenning et al. '01, Benton et al. '97]

Adjoint Logic*

basic types	$T := \texttt{int} \mid \texttt{bool} \mid \texttt{unit}$
access qualifiers	$q := \texttt{CK} \mid \texttt{RD} \mid \texttt{MtWt} \mid \texttt{Wtn}$
unstable types	$\tau^u := T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid v_t$
stable types	$\tau^s := \uparrow \tau^u \mid$  $\rightsquigarrow \tau^s$



Independence principle: universal truth does not depend on ephemeral truth.

"It is *always* cold"

↑ ↓

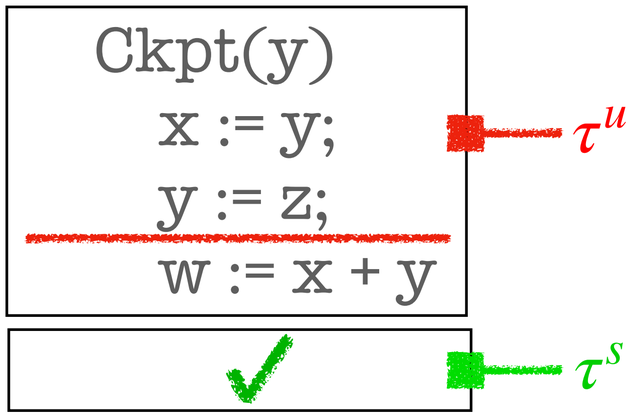
"It is cold *today*"

Here: stable values *(s)* do not depend on unstable values *(u)*.

* [Pruiksma et al. '21, Pfenning et al. '01, Benton et al. '97]

Adjoint Logic*

basic types	$T := \texttt{int} \mid \texttt{bool} \mid \texttt{unit}$
access qualifiers	$q := \texttt{CK} \mid \texttt{RD} \mid \texttt{MtWt} \mid \texttt{Wtn}$
unstable types	$\tau^u := T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid v_t$
stable types	$\tau^s := \uparrow \tau^u \mid$  $\rightsquigarrow \tau^s$



Independence principle: universal truth does not depend on ephemeral truth.

"It is *always* cold"

↑ ↓

"It is cold *today*"

Here: stable values *(s)* do not depend on unstable values *(u)*.

τ^s

Restore ↑ ↓ PwOff

τ^u

* [Pruiksma et al. '21, Pfenning et al. '01, Benton et al. '97]

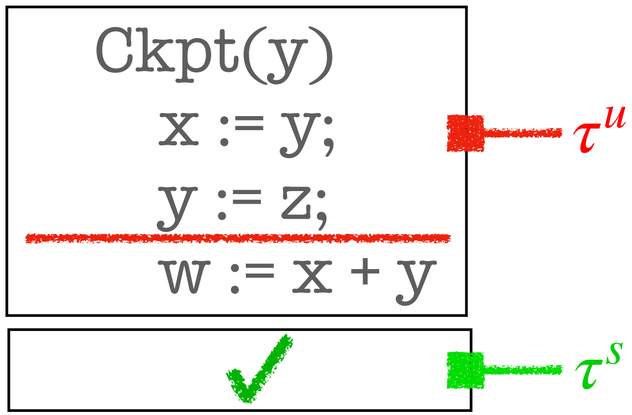
Crash Type

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

access qualifiers $q := \texttt{CK} \mid \texttt{RD} \mid \texttt{MtWt} \mid \texttt{Wtn}$

unstable types $\tau^u := T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid v_t$

stable types $\tau^s := \uparrow \tau^u \mid$  $\rightsquigarrow \tau^s$



$$C = \frac{\quad}{\text{Successful execution}} \vee \frac{\quad}{\text{Failed execution}}$$

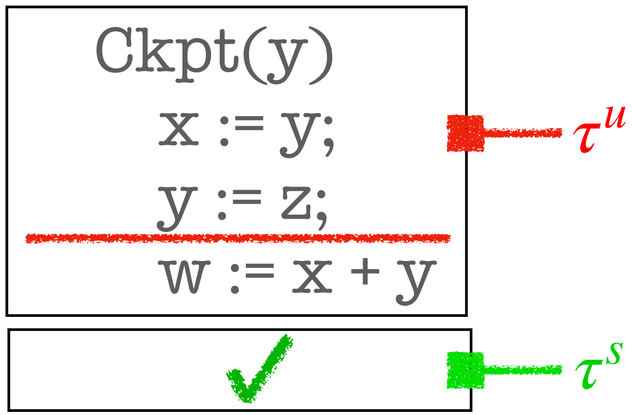
Crash Type

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

access qualifiers $q := \texttt{CK} \mid \texttt{RD} \mid \texttt{MtWt} \mid \texttt{Wtn}$

unstable types $\tau^u := T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid v_t$

stable types $\tau^s := \uparrow \tau^u \mid$  $\rightsquigarrow \tau^s$



Commit

$C = \downarrow$ Successful execution $\vee$ Failed execution

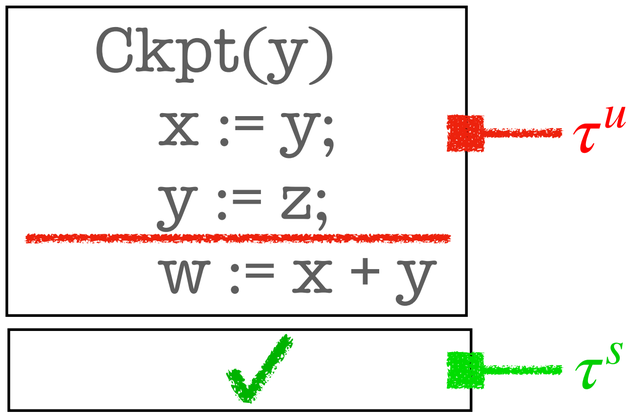
Crash Type

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

access qualifiers $q := \texttt{CK} \mid \texttt{RD} \mid \texttt{MtWt} \mid \texttt{Wtn}$

unstable types $\tau^u := T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid v_t$

stable types $\tau^s := \uparrow \tau^u \mid$  $\rightsquigarrow \tau^s$



Success!

$C = \downarrow \uparrow \texttt{unit} \vee$ ~~$\downarrow \uparrow \texttt{unit}$~~

Successful execution

Failed execution

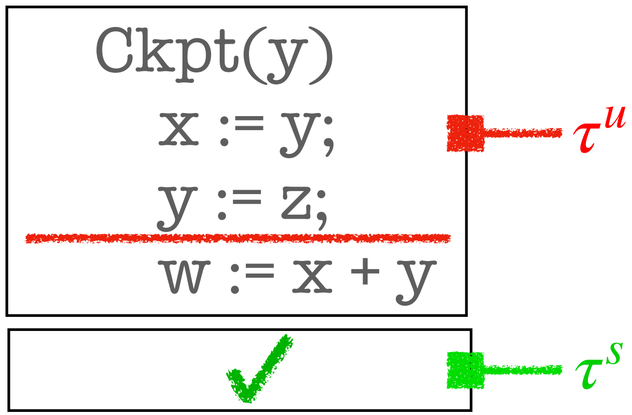
Crash Type

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

access qualifiers $q := \texttt{CK} \mid \texttt{RD} \mid \texttt{MtWt} \mid \texttt{Wtn}$

unstable types $\tau^u := T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid v_t$

stable types $\tau^s := \uparrow \tau^u \mid$  $\rightsquigarrow \tau^s$



PwOff

$$C = \downarrow \uparrow \texttt{unit} \vee \downarrow$$

Successful execution

Failed execution

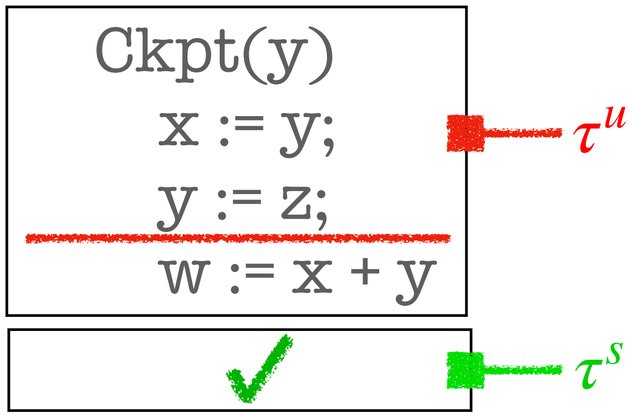
Crash Type

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

access qualifiers $q := \texttt{CK} \mid \texttt{RD} \mid \texttt{MtWt} \mid \texttt{Wtn}$

unstable types $\tau^u := T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid v_t$

stable types $\tau^s := \uparrow \tau^u \mid$  $\rightsquigarrow \tau^s$



Harvest

$C = \downarrow \uparrow \texttt{unit} \vee \downarrow ($  $\rightsquigarrow \_\_\_)$

Successful execution

Failed execution

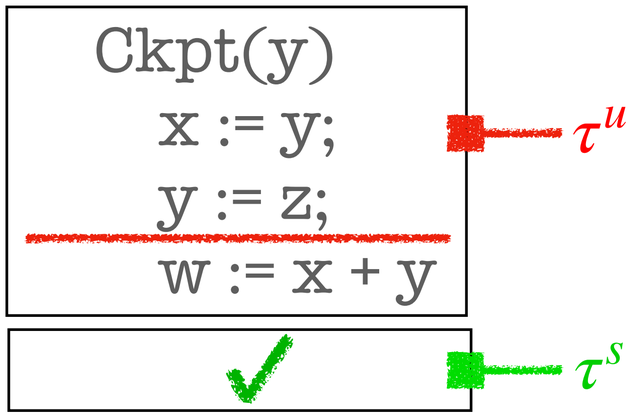
Crash Type

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

access qualifiers $q := \texttt{CK} \mid \texttt{RD} \mid \texttt{MtWt} \mid \texttt{Wtn}$

unstable types $\tau^u := T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid v_t$

stable types $\tau^s := \uparrow \tau^u \mid$  $\rightsquigarrow \tau^s$



Restore

$C = \downarrow \uparrow \texttt{unit} \vee \downarrow ($  $\rightsquigarrow \uparrow \_)$

Successful execution

Failed execution

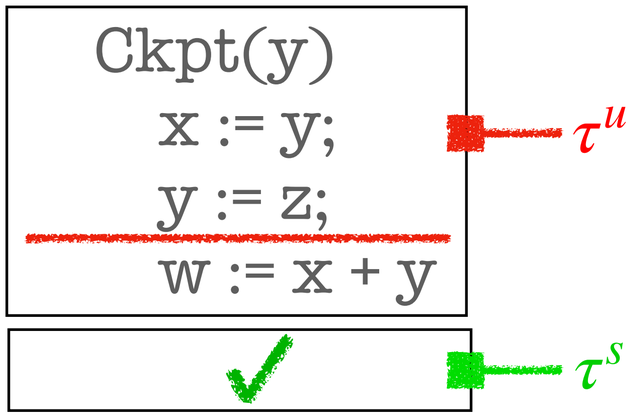
Crash Type

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

access qualifiers $q := \texttt{CK} \mid \texttt{RD} \mid \texttt{MtWt} \mid \texttt{Wtn}$

unstable types $\tau^u := T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid v_t$

stable types $\tau^s := \uparrow \tau^u \mid$  $\rightsquigarrow \tau^s$



Re-execute

$\texttt{C} = \downarrow \uparrow \texttt{unit} \vee \downarrow ($  $\rightsquigarrow \uparrow \texttt{C})$

Successful execution

Failed execution

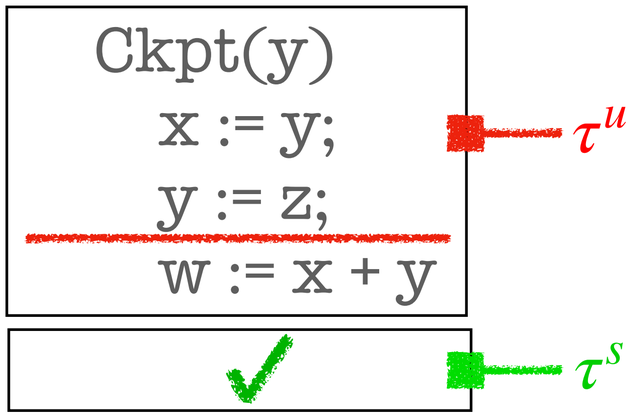
Typing judgments

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

access qualifiers $q := \texttt{CK} \mid \texttt{RD} \mid \texttt{MtWt} \mid \texttt{Wtn}$

unstable types $\tau^u := T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid v_t$

stable types $\tau^s := \uparrow \tau^u \mid$ 🔋 $\rightsquigarrow \tau^s$



Assumptions $\vdash$ To show

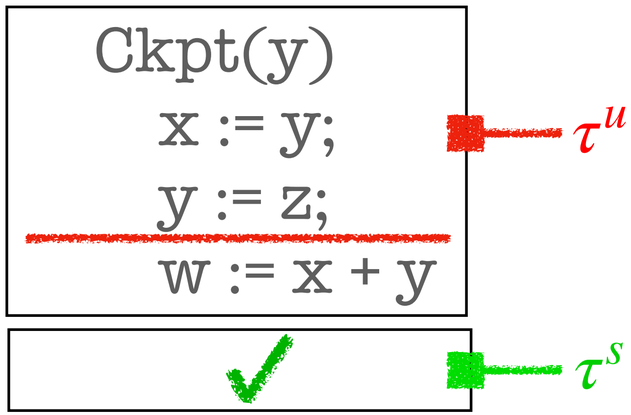
Typing judgments

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

access qualifiers $q := \texttt{CK} \mid \texttt{RD} \mid \texttt{MtWt} \mid \texttt{Wtn}$

unstable types $\tau^u := T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid v_t$

stable types $\tau^s := \uparrow \tau^u \mid \rightsquigarrow \tau^s$



Unstable typing judgment

$$\frac{}{} \vdash \frac{}{}$$

Stable typing judgment

$$\frac{}{} \vdash \frac{}{}$$

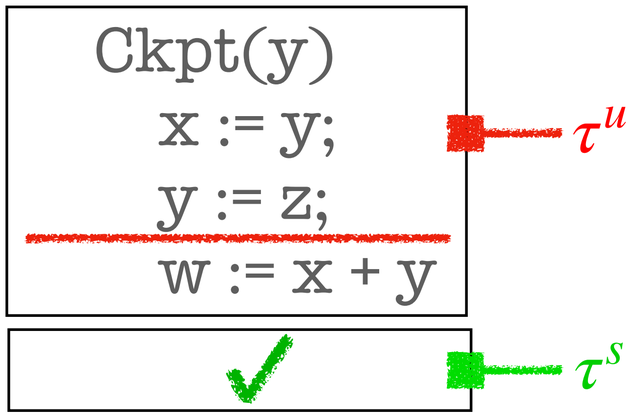
Typing judgments

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

access qualifiers $q := \texttt{CK} \mid \texttt{RD} \mid \texttt{MtWt} \mid \texttt{Wtn}$

unstable types $\tau^u := T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid v_t$

stable types $\tau^s := \uparrow \tau^u \mid$  $\rightsquigarrow \tau^s$



Unstable typing judgment

$$\frac{}{\vdash \texttt{x := y} : \texttt{C}}$$

Command is well-typed

Stable typing judgment

$$\frac{\vdash \quad}{\vdash \quad}$$

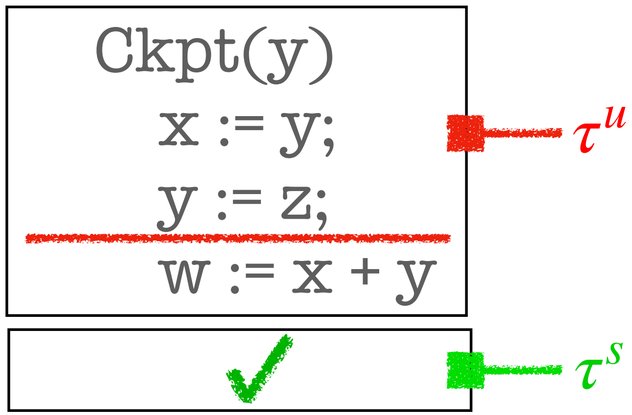
Typing judgments

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

access qualifiers $q := \texttt{CK} \mid \texttt{RD} \mid \texttt{MtWt} \mid \texttt{Wtn}$

unstable types $\tau^u := T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid v_t$

stable types $\tau^s := \uparrow \tau^u \mid$  $\rightsquigarrow \tau^s$



Unstable typing judgment

 $\mid$ _____ $\vdash$ x := y : C

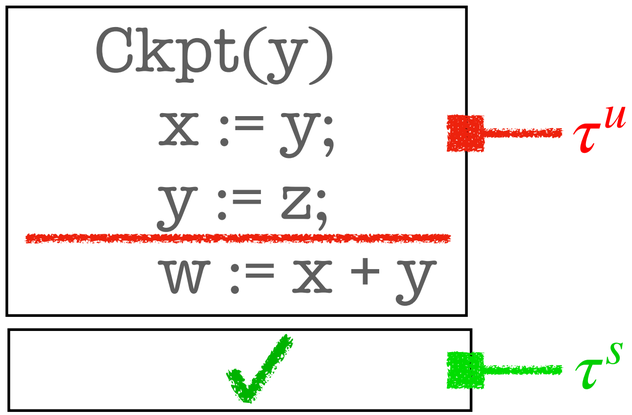
Energy buffer

Stable typing judgment

_____ $\vdash$ _____

Typing judgments

basic types	$T := \texttt{int} \mid \texttt{bool} \mid \texttt{unit}$
access qualifiers	$q := \texttt{CK} \mid \texttt{RD} \mid \texttt{MtWt} \mid \texttt{Wtn}$
unstable types	$\tau^u := T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid v_t$
stable types	$\tau^s := \uparrow \tau^u \mid \rightsquigarrow \tau^s$



Ω : NV

Σ : V

Unstable typing judgment

$$x : \tau^s \quad y : \tau^u$$

$$\text{Energy buffer} \mid \Omega \mid \Sigma \vdash \texttt{x := y : C}$$

NV+V typing contexts

Stable typing judgment

$$\vdash$$

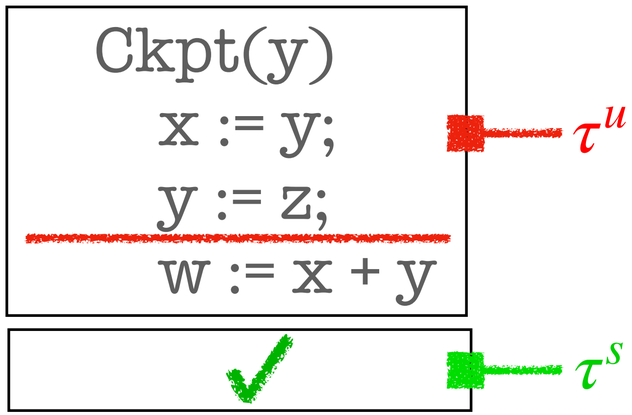
Typing judgments

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

access qualifiers $q := \texttt{CK} \mid \texttt{RD} \mid \texttt{MtWt} \mid \texttt{Wtn}$

unstable types $\tau^u := T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid v_t$

stable types $\tau^s := \uparrow \tau^u \mid \rightsquigarrow \tau^s$



Unstable typing judgment

$$x : \tau^s \quad y : \tau^u$$

$$\text{Energy buffer} \mid \Omega \mid \Sigma \vdash \texttt{x := y} : C \dashv \Omega'$$

Energy buffer — NV+V typing contexts — Post-context

Stable typing judgment

$$\vdash$$

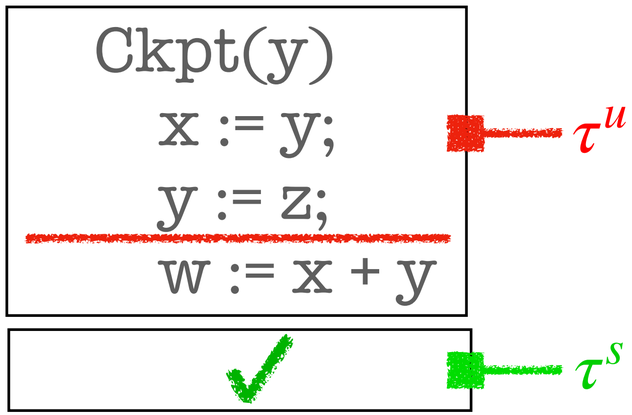
Typing judgments

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

access qualifiers $q := \texttt{CK} \mid \texttt{RD} \mid \texttt{MtWt} \mid \texttt{Wtn}$

unstable types $\tau^u := T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid v_t$

stable types $\tau^s := \uparrow \tau^u \mid \rightsquigarrow \tau^s$



Unstable typing judgment

$$x : \tau^s \quad y : \tau^u$$

$$\rule{1.5em}{1em} \mid \Omega \mid \Sigma \vdash \texttt{x := y} : C \dashv \Omega'$$

Energy buffer

NV+V typing contexts

Post-context

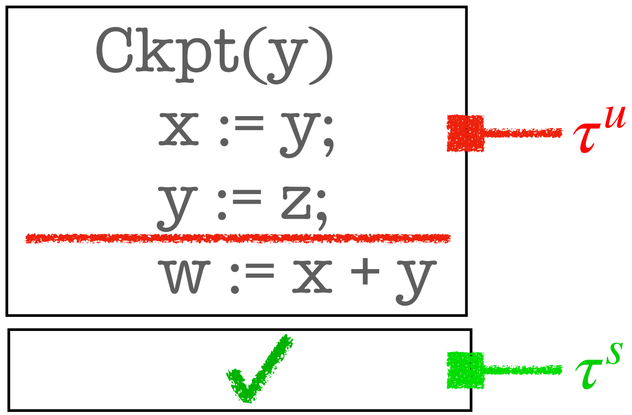
Stable typing judgment

$$x : \tau^s$$

$$\rule{1.5em}{1em} \mid \Omega \vdash \texttt{Ckpt}\ \{\texttt{x := y; y := z; w:= x+y;}\} : \uparrow C$$

Typing a program bottom-up

basic types	$T := \texttt{int} \mid \texttt{bool} \mid \texttt{unit}$
access qualifiers	$q := \texttt{CK} \mid \texttt{RD} \mid \texttt{MtWt} \mid \texttt{Wtn}$
unstable types	$\tau^u := T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid v_t$
stable types	$\tau^s := \uparrow \tau^u \mid \blacksquare\!\!\blacksquare \rightsquigarrow \tau^s$



$\blacksquare\!\!\blacksquare \mid \Omega \vdash$ `Ckpt` `x := y; y := z; w:= x+y;` $: \uparrow C$

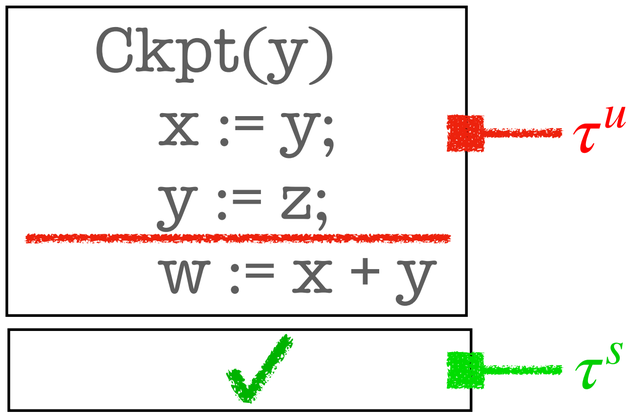
Typing a command

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

access qualifiers $q := \texttt{CK} \mid \texttt{RD} \mid \texttt{MtWt} \mid \texttt{Wtn}$

unstable types $\tau^u := T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid v_t$

stable types $\tau^s := \uparrow \tau^u \mid$  $\rightsquigarrow \tau^s$



$$\frac{\text{  } \mid \Omega \mid \Sigma \vdash \texttt{x := y; y := z; w := x + y} : \texttt{C} \dashv \Omega'''}{\text{  } \mid \Omega \vdash \texttt{Ckpt}\ \boxed{\begin{array}{l}\texttt{x := y;}\\ \texttt{y := z;}\\ \texttt{w:= x+y;}\end{array}} : \uparrow \texttt{C}}$$

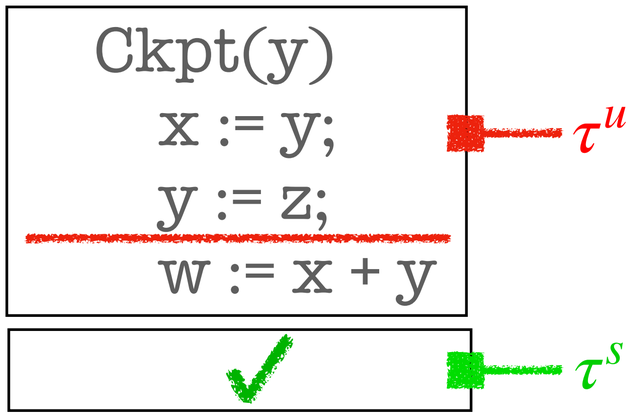
Typing a smaller command

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

access qualifiers $q := \texttt{CK} \mid \texttt{RD} \mid \texttt{MtWt} \mid \texttt{Wtn}$

unstable types $\tau^u := T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid v_t$

stable types $\tau^s := \uparrow \tau^u \mid \blacksquare\!\!\blacksquare\!\!\blacksquare\!\!\blacksquare \rightsquigarrow \tau^s$



$$\frac{\blacksquare\!\!\blacksquare\!\!\blacksquare\!\!\blacksquare \mid \Omega \mid \Sigma \vdash \texttt{x := y;}\ \texttt{y := z} : \texttt{C} \dashv \Omega'' \quad \cdots}{\blacksquare\!\!\blacksquare\!\!\blacksquare\!\!\blacksquare \mid \Omega \mid \Sigma \vdash \texttt{x := y;}\ \texttt{y := z;}\ \texttt{w := x + y} : \texttt{C} \dashv \Omega'''}$$

$$\blacksquare\!\!\blacksquare\!\!\blacksquare\!\!\blacksquare \mid \Omega \vdash \texttt{Ckpt}\ \boxed{\begin{array}{l}\texttt{x := y;}\\ \texttt{y := z;}\\ \texttt{w:= x+y;}\end{array}} : \uparrow \texttt{C}$$

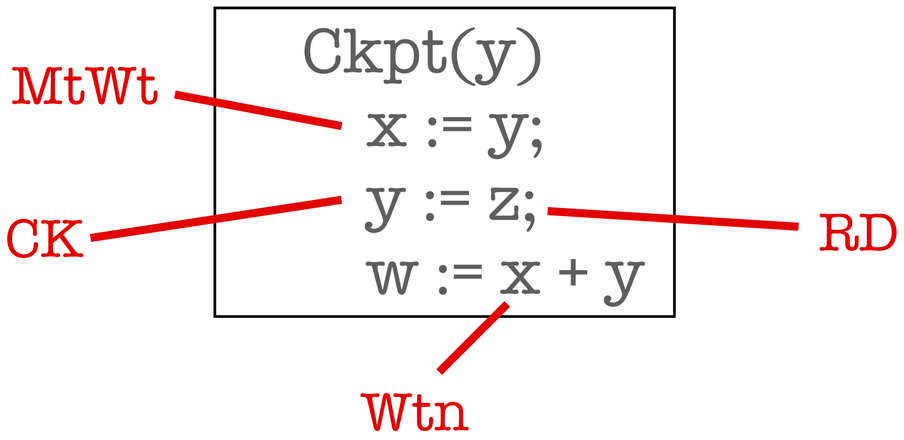
Type checking x:=y; y:=z

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

access qualifiers $q := \texttt{CK} \mid \texttt{RD} \mid \texttt{MtWt} \mid \texttt{Wtn}$

unstable types $\tau^u := T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid v_t$

stable types $\tau^s := \uparrow \tau^u \mid \blacksquare \rightsquigarrow \tau^s$



$x : \texttt{MtWt}, y : \texttt{CK}$ $\qquad$ $x : \texttt{Wtn}, y : \texttt{CK}$

$$\blacksquare \mid \Omega \mid \Sigma \vdash \texttt{x := y} : C \dashv \Omega'$$

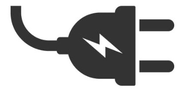
$$\blacksquare \mid \Omega' \mid \Sigma \vdash \texttt{y := z} : \tau^u \dashv \Omega''$$

$$\blacksquare \mid \Omega \mid \Sigma \vdash \texttt{x := y; y := z} : \tau^u \dashv \Omega''$$

$x : \texttt{Wtn}, y : \texttt{CK}$

Example

NV	x	y	z	w
	MtWt	CK	RD	MtWt
	0	**1**	2	0



NV	x	y	z	w
	MtWt	CK	RD	MtWt
1	0	1	2	0

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

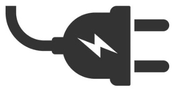
Example

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

NV	x	y	z	w
	MtWt	CK	RD	MtWt
	0	**1**	2	0

Ckpt

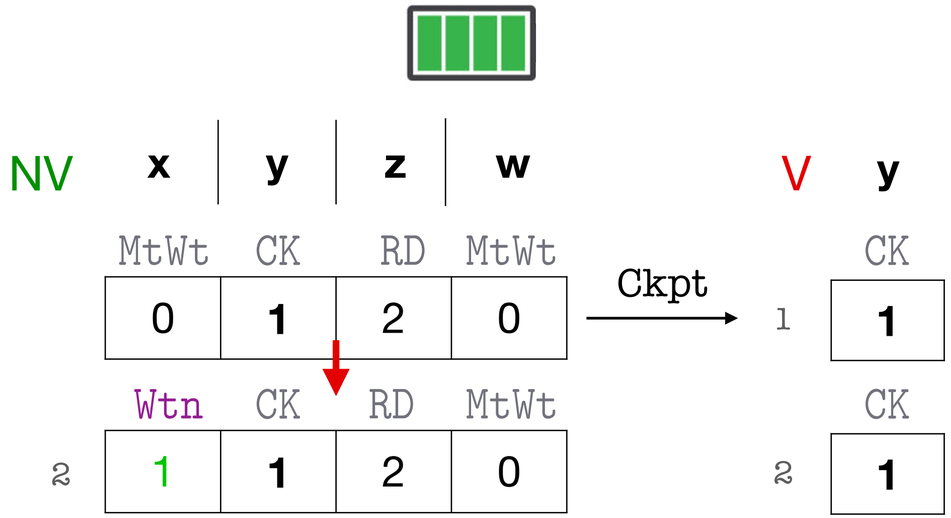
V	y
	CK
1	**1**

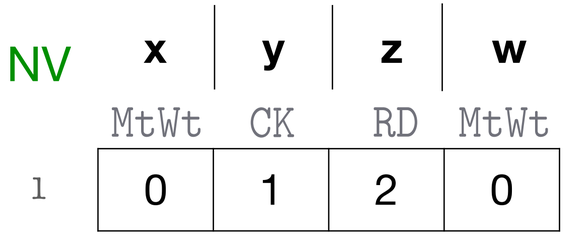


NV	x	y	z	w
	MtWt	CK	RD	MtWt
1	0	1	2	0

Example

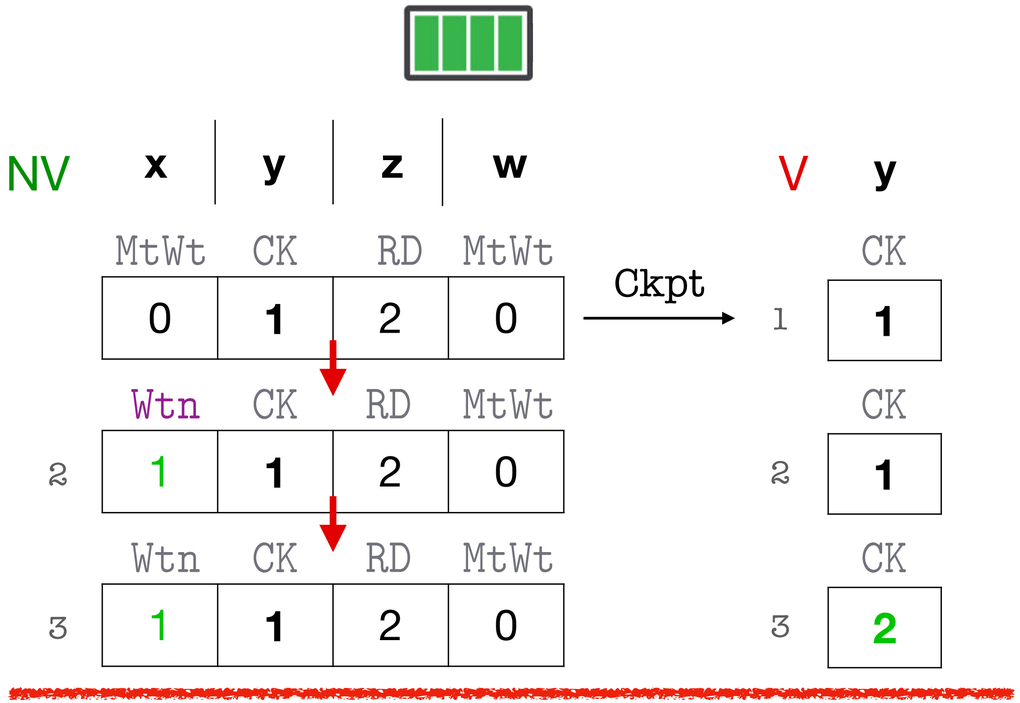
```
1 Ckpt(y)
2     x := y;
3     y := z;
4     w := x + y
```

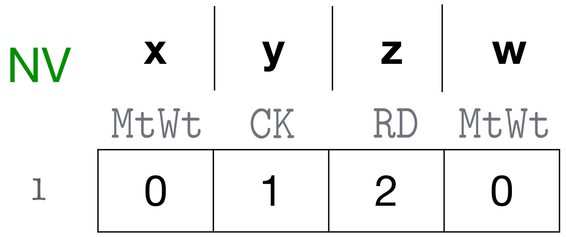




Example

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





Example

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

NV

x	y	z	w
MtWt	CK	RD	MtWt
0	1	2	0
Wtn	CK	RD	MtWt
1	1	2	0
Wtn	CK	RD	MtWt
1	1	2	0

2

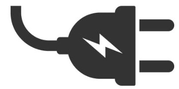
3

V

y
CK

PwOff

1



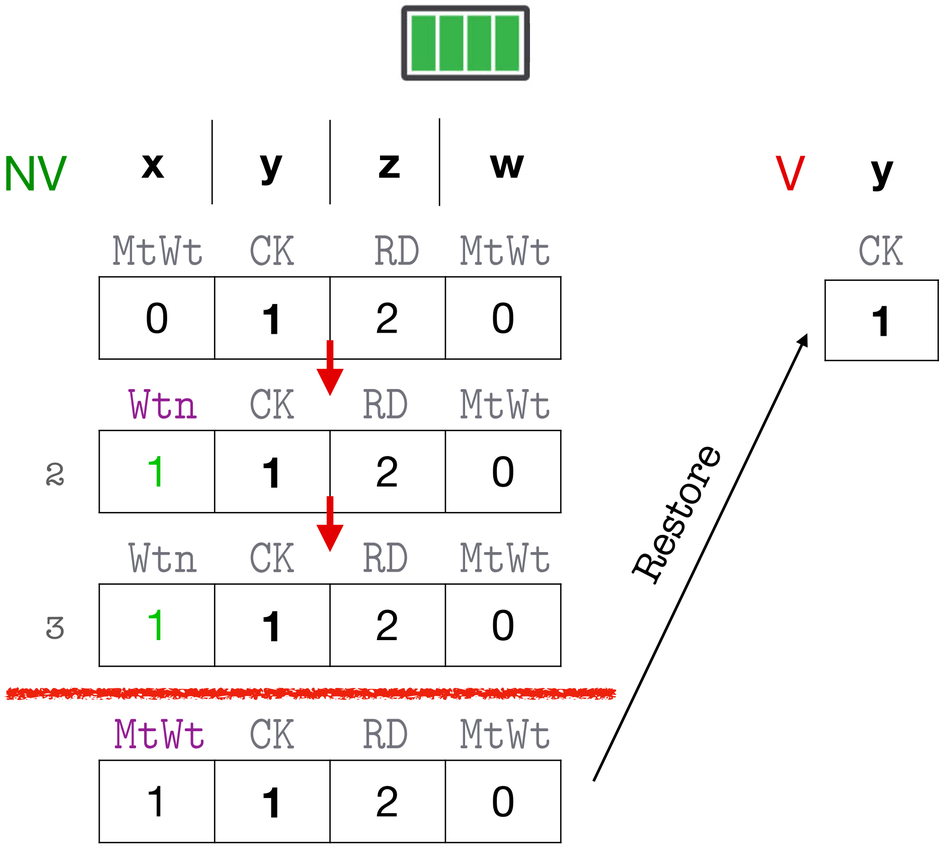
NV

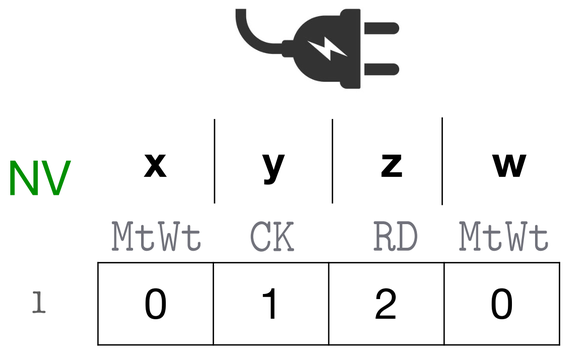
x	y	z	w
MtWt	CK	RD	MtWt
0	1	2	0

1

Example

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





Example

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

NV

x	y	z	w
MtWt	CK	RD	MtWt
0	1	2	0

	Wtn	CK	RD	MtWt
2	1	1	2	0

	Wtn	CK	RD	MtWt
3	1	1	2	0

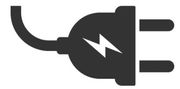
	MtWt	CK	RD	MtWt
	1	1	2	0

	Wtn	CK	RD	MtWt
2	1	1	2	0

V

y
CK
1

	CK
2	1



NV

x	y	z	w
MtWt	CK	RD	MtWt
0	1	2	0

Example

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

NV

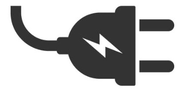
	x	y	z	w
	MtWt	CK	RD	MtWt
	0	**1**	2	0
	Wtn	CK	RD	MtWt
2	1	**1**	2	0
	Wtn	CK	RD	MtWt
3	1	**1**	2	0

V

	y
	CK
	1

	MtWt	CK	RD	MtWt
	1	**1**	2	0
	Wtn	CK	RD	MtWt
2	1	**1**	2	0
	Wtn	CK	RD	MtWt
3	1	**1**	2	0

	CK
2	**1**
	CK
3	**2**



NV

	x	y	z	w
	MtWt	CK	RD	MtWt
1	0	1	2	0

Example

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

NV

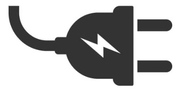
	x	y	z	w
	MtWt	CK	RD	MtWt
	0	1	2	0
	Wtn	CK	RD	MtWt
2	1	1	2	0
	Wtn	CK	RD	MtWt
3	1	1	2	0

V

	y
	CK
	1

	MtWt	CK	RD	MtWt
	1	1	2	0
	Wtn	CK	RD	MtWt
2	1	1	2	0
	Wtn	CK	RD	MtWt
3	1	1	2	0
	Wtn	CK	RD	Wtn
4	1	1	2	3

	CK	
2	1	
	CK	
3	2	
	CK	
4	2	



NV

	x	y	z	w
	MtWt	CK	RD	MtWt
1	0	1	2	0

Example

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

NV

	x	y	z	w
	MtWt	CK	RD	MtWt
	0	1	2	0
	Wtn	CK	RD	MtWt
2	1	1	2	0
	Wtn	CK	RD	MtWt
3	1	1	2	0
	MtWt	CK	RD	MtWt
	1	1	2	0
	Wtn	CK	RD	MtWt
2	1	1	2	0
	Wtn	CK	RD	MtWt
3	1	1	2	0
	Wtn	CK	RD	Wtn
4	1	2	2	3

V

	y
	CK
	1
	CK
2	1
	CK
3	2
	CK
4	2

Commit

NV

	x	y	z	w
	MtWt	CK	RD	MtWt
1	0	1	2	0

Example

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

NV

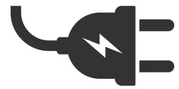
	x	y	z	w
	MtWt	CK	RD	MtWt
	0	1	2	0
	Wtn	CK	RD	MtWt
2	1	1	2	0
	Wtn	CK	RD	MtWt
3	1	1	2	0

	MtWt	CK	RD	MtWt
	1	1	2	0
	Wtn	CK	RD	MtWt
2	1	1	2	0
	Wtn	CK	RD	MtWt
3	1	1	2	0
	Wtn	CK	RD	Wtn
4	1	2	2	3

V

	y
	CK
	1
	CK
2	1
	CK
3	2

=



NV

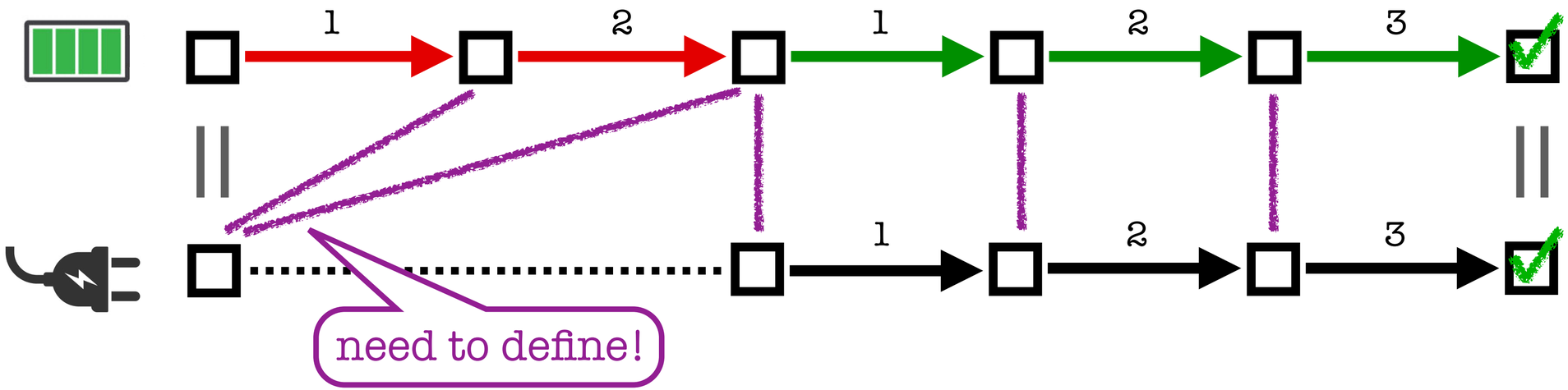
	x	y	z	w
	MtWt	CK	RD	MtWt
1	0	1	2	0
	Wtn	CK	RD	MtWt
2	1	1	2	0
	Wtn	CK	RD	MtWt
3	1	2	2	0
	Wtn	CK	RD	Wtn
4	1	2	2	3

Outline

- Intermittent computing
- Overview
- Type system
- **Logical relation**
- JIT mode
- Key theorems
- Conclusion

Proving Correctness

Every intermittent execution of a **well-typed** program can be simulated by its continuous execution.



Binary Logical Relation

Every intermittent execution of a **well-typed** program can be simulated by its continuous execution.

---

crashes 👀

$$(\ \text{  }\ \mid \text{NV} \mid \text{V} \mid c_1,\ \text{  }\ \mid \text{NV} \mid \text{V} \mid c_2) \in \_\ [\![\text{C}_{Unit}]\!]^m$$

Intermittent execution — Continuous execution — Crash type

Term Relation

---

where c_1 is...

$\mathtt{x} := \mathtt{y}$

$c_1; c_2$

$\mathtt{if}\ v\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2$

$\mathtt{let}\ x = e\ \mathtt{in}\ c$

Value Relation

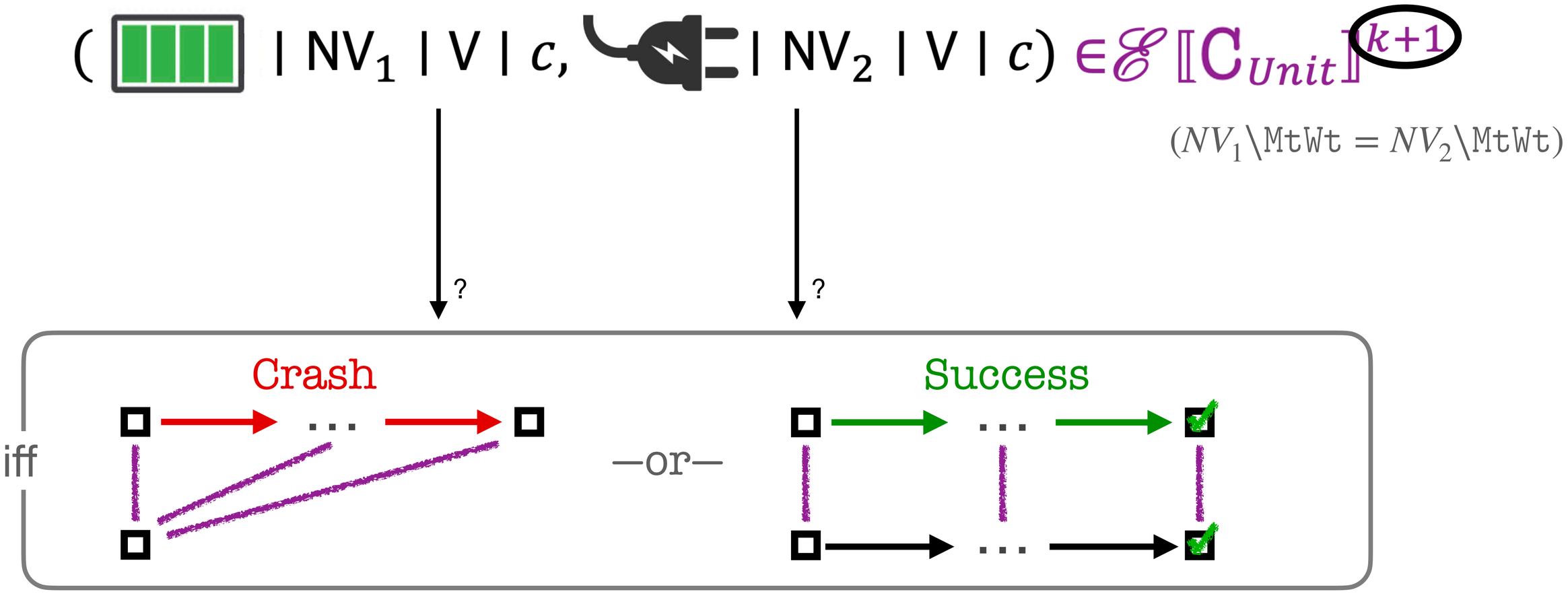
---

where v_1 is...

n	false	↓ (↑ c)	↑ c
true	skip	(↑ c)	

No remaining power failures

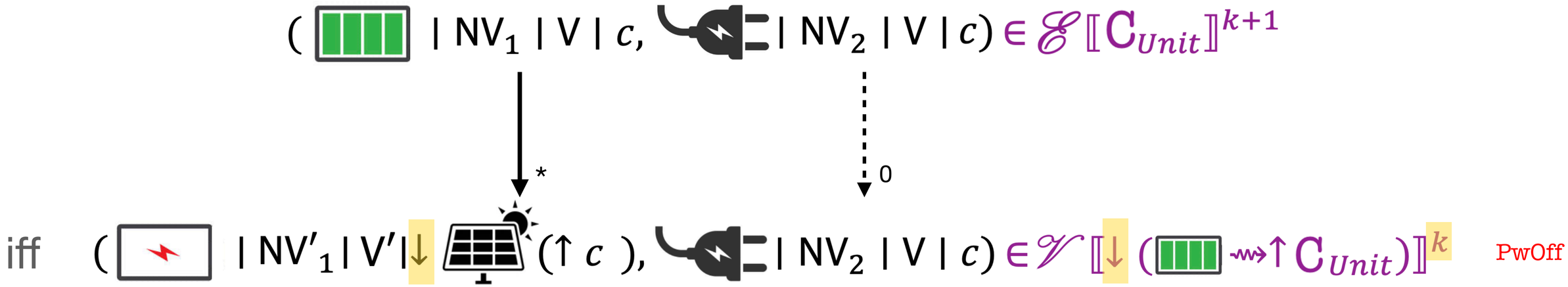
A power failure is possible!



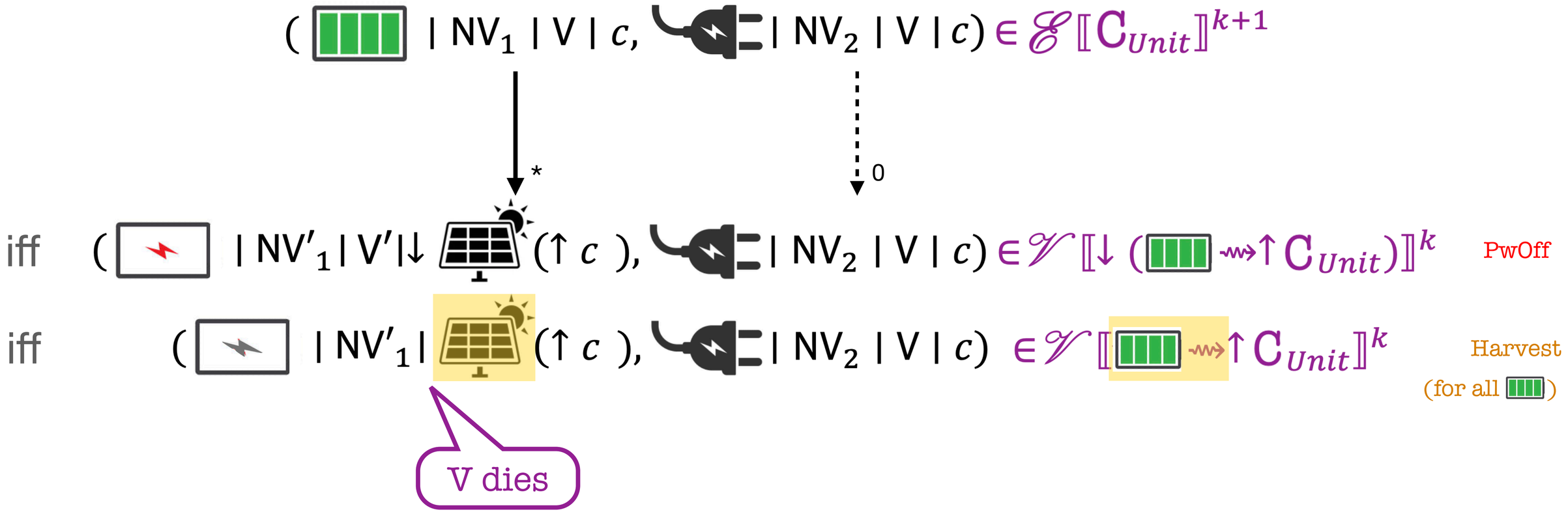
If there is a power failure…

$(\ \text{  }\ \mid \mathrm{NV}_1 \mid \mathrm{V} \mid c,\ \text{  }\ \mid \mathrm{NV}_2 \mid \mathrm{V} \mid c) \in \mathscr{E}[\![\mathbf{C}_{Unit}]\!]^{k+1}$

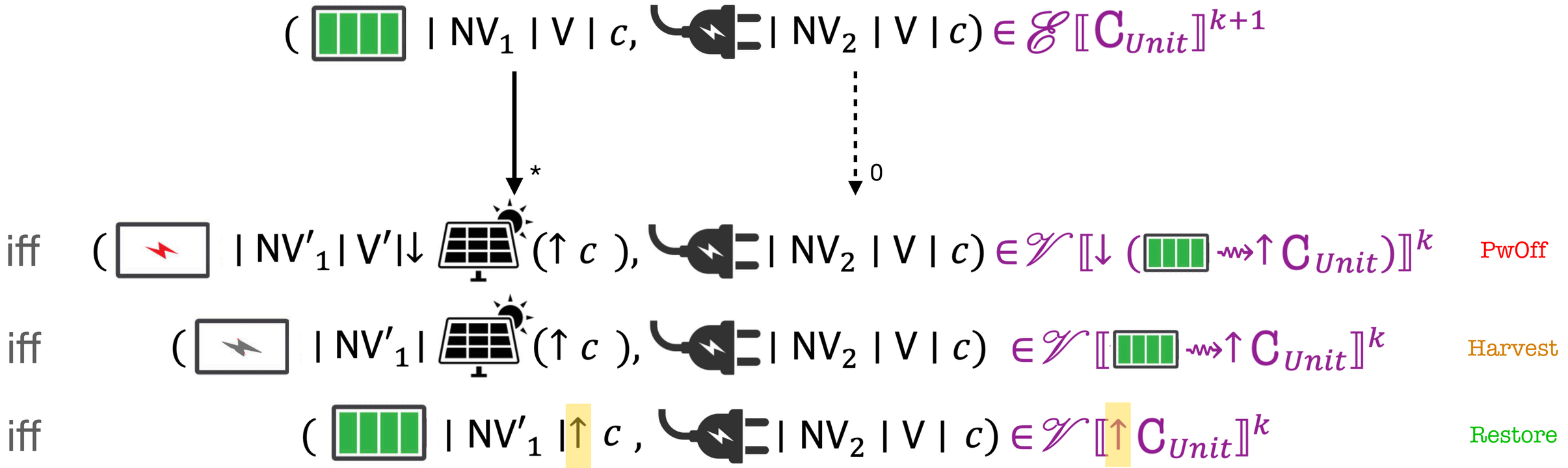
Device powers off



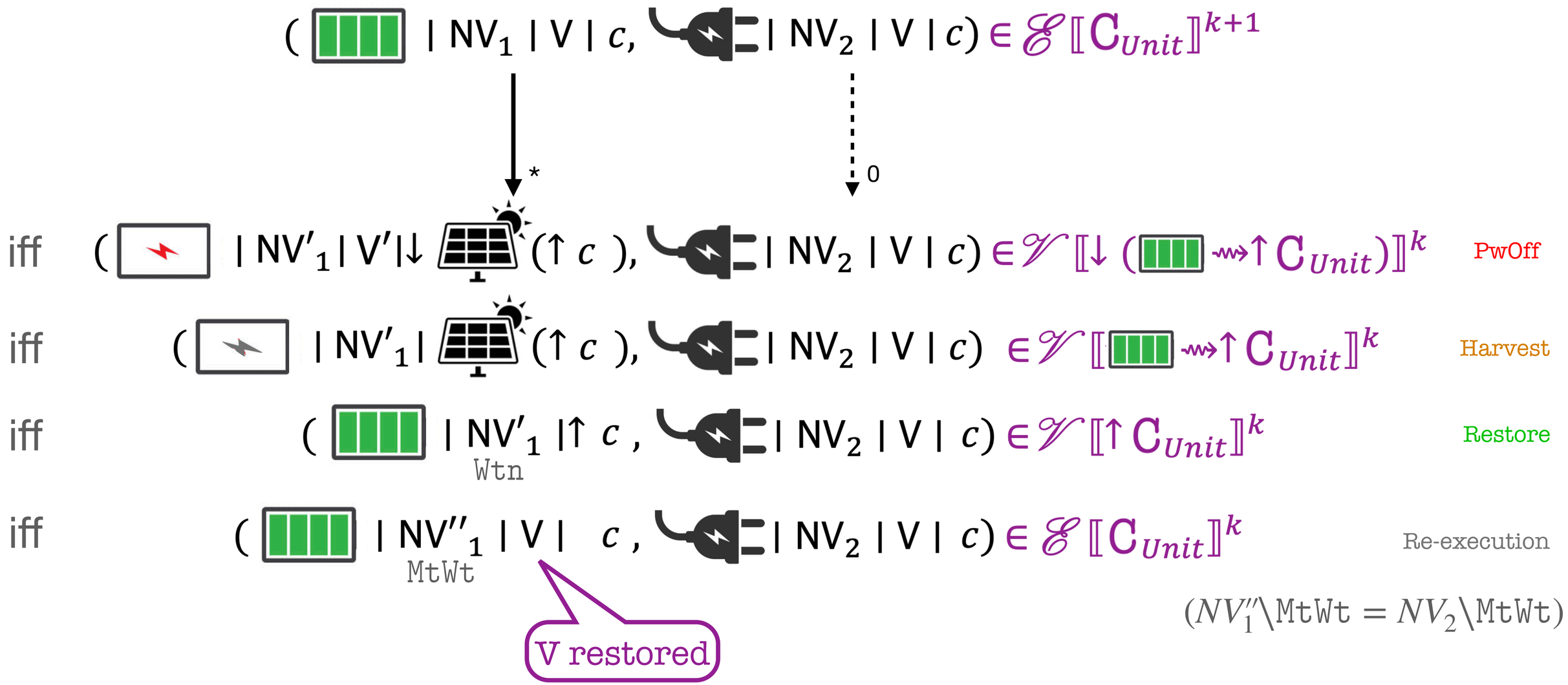
Harvest energy from the environment



Execution state is restored



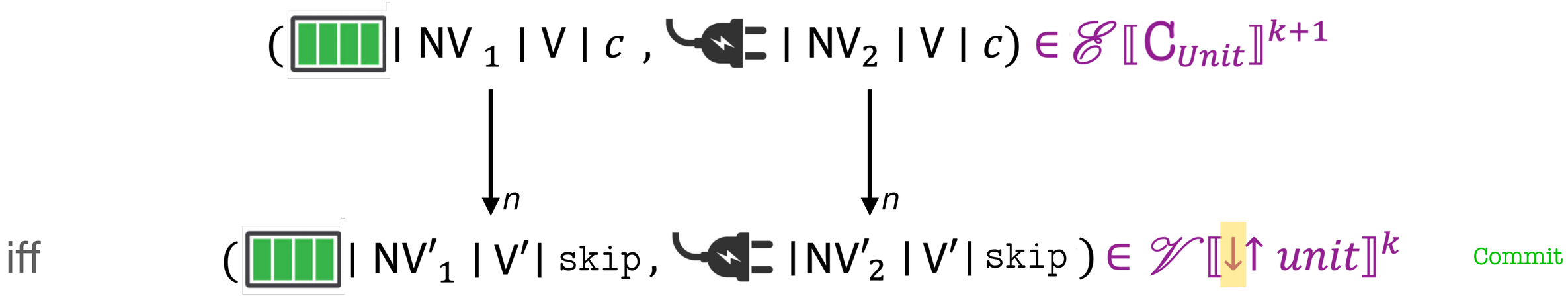
Prepare for re-execution



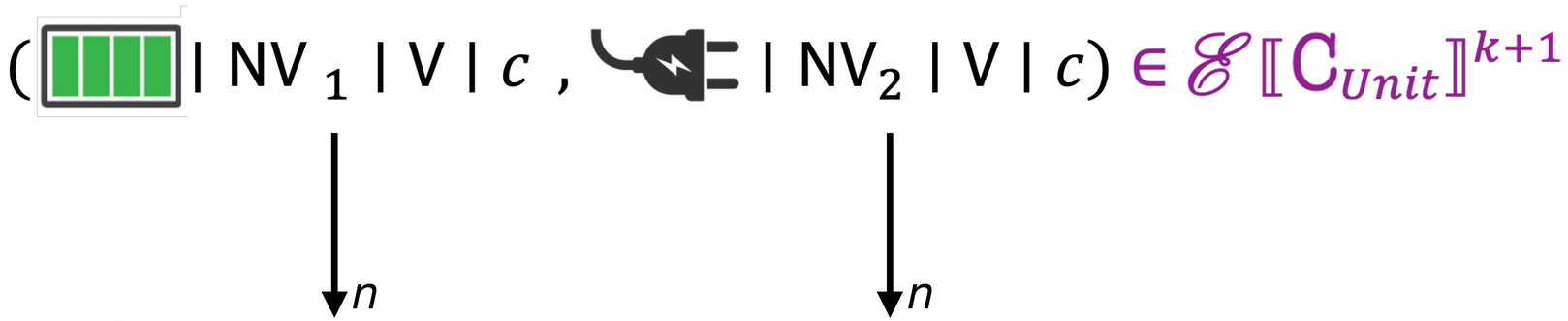
If there is no power failure…

$$(\text{  } \mid \mathrm{NV}_1 \mid \mathrm{V} \mid c\ ,\ \text{  } \mid \mathrm{NV}_2 \mid \mathrm{V} \mid c) \in \mathscr{E}[\![\mathrm{C}_{Unit}]\!]^{k+1}$$

Stable values are committed



We are done!

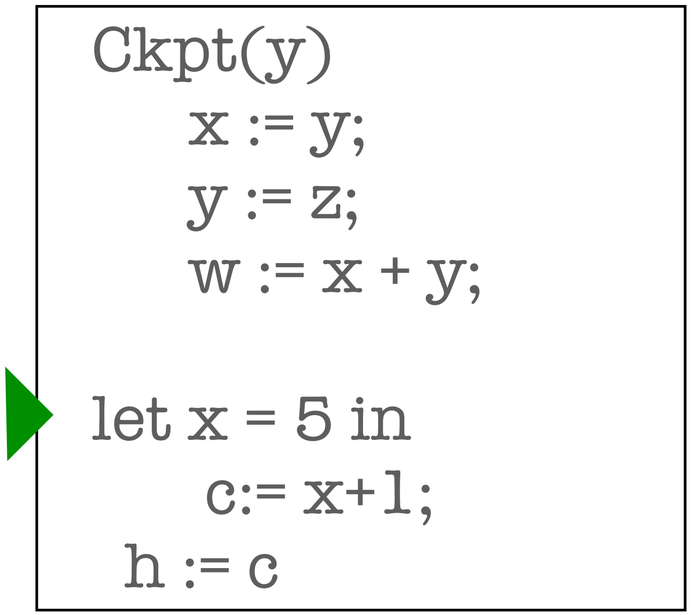

(| NV 1 | V | c , | NV2 | V | c) ∈ ℰ⟦C_Unit⟧^{k+1}
n
n
iff (| NV′1 | V′ | skip, | NV′2 | V′ | skip) ∈ 𝒱⟦↓↑ unit⟧^k
Commit
iff (| NV″1 | V′ | skip, | NV″2 | V′ | skip) ∈ 𝒱⟦↑ unit⟧^k
Success!
(NV″1 = NV″2)

Outline

- Intermittent computing
- Overview
- Type system
- Logical relation
- **JIT mode**
- Key theorems
- Conclusion

Real programs don’t always use Ckpt

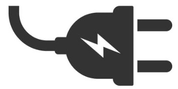
- intermittent systems default to JIT mode
- checkpoints everything “just-in-time”
- no re-execution
- hybrid JIT+Ckpt programs



Example

```
5 let x = 5 in
6     c:= x+1;
7 h := c
```

NV	c	h
	CK	CK
	1	0



NV	c	h
	CK	CK
	1	0

Example

```
5  let x = 5 in
6      c:= x+1;
7  h := c
```

NV

	c	h
	CK	CK
	1	0
	CK	CK
5	1	0

V

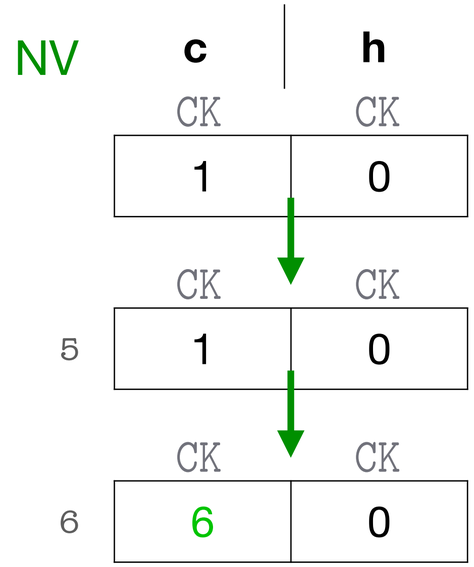
	x
	CK
5	5

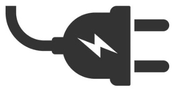
NV

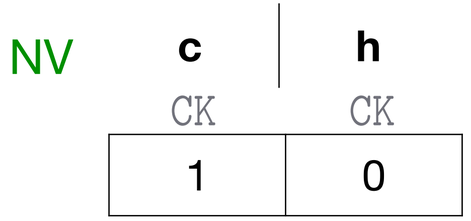
c	h
CK	CK
1	0

Example

```
5  let x = 5 in
6      c:= x+1;
7  h := c
```

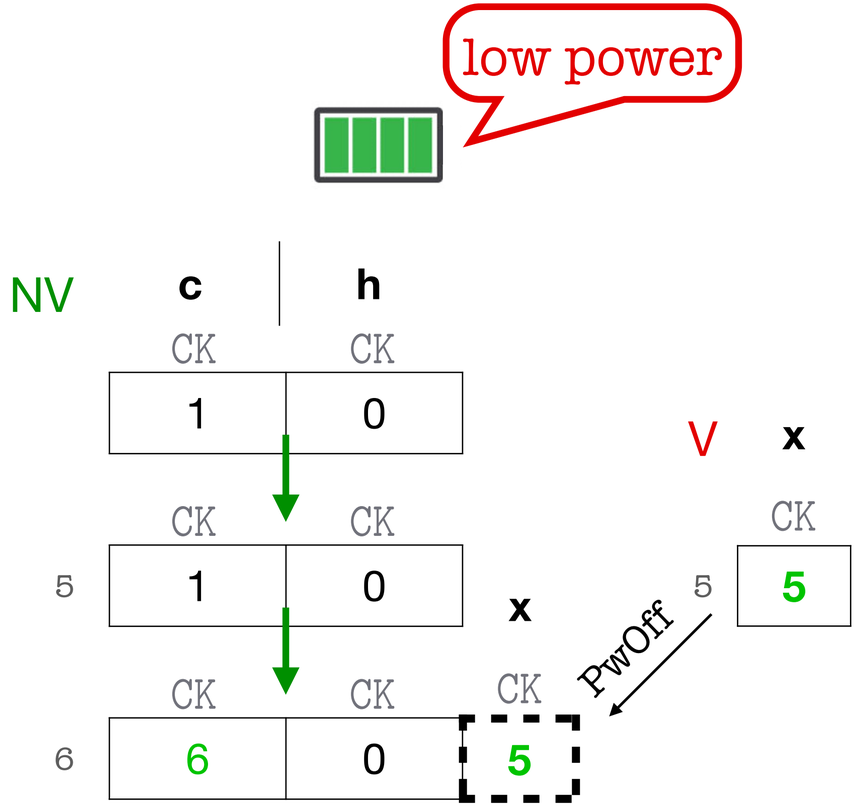


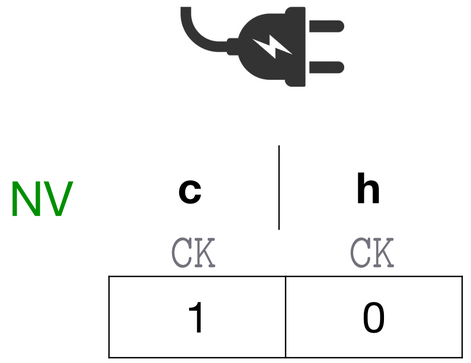




Example

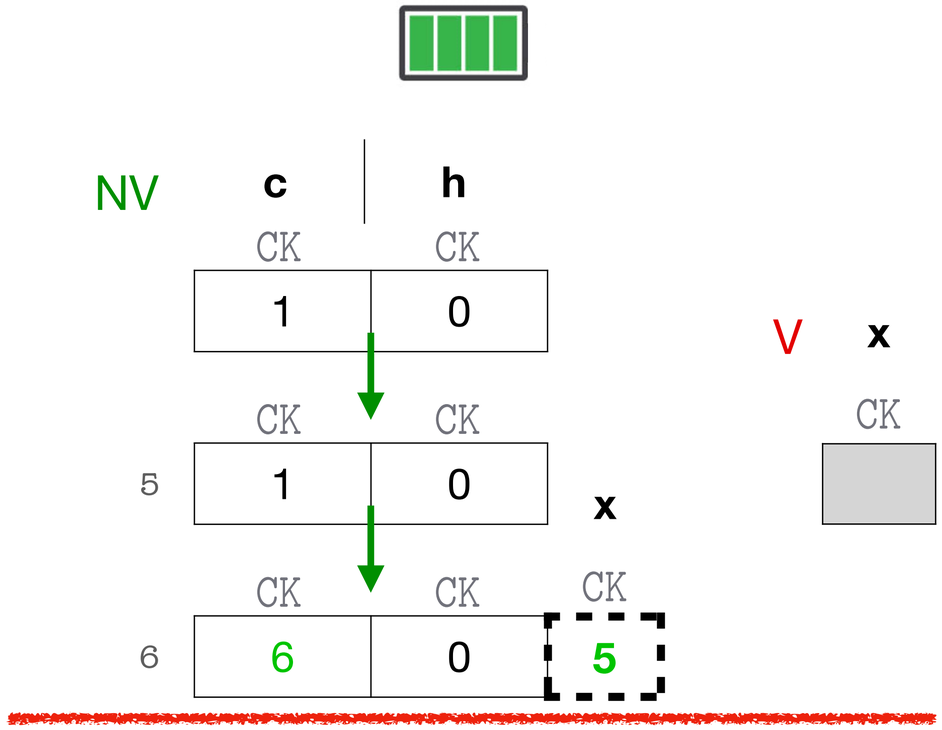
```
5  let x = 5 in
6      c:= x+1;
7  h := c
```

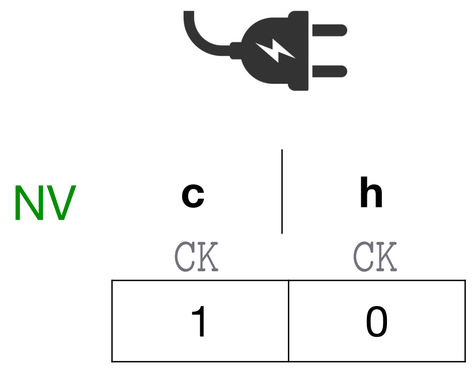




Example

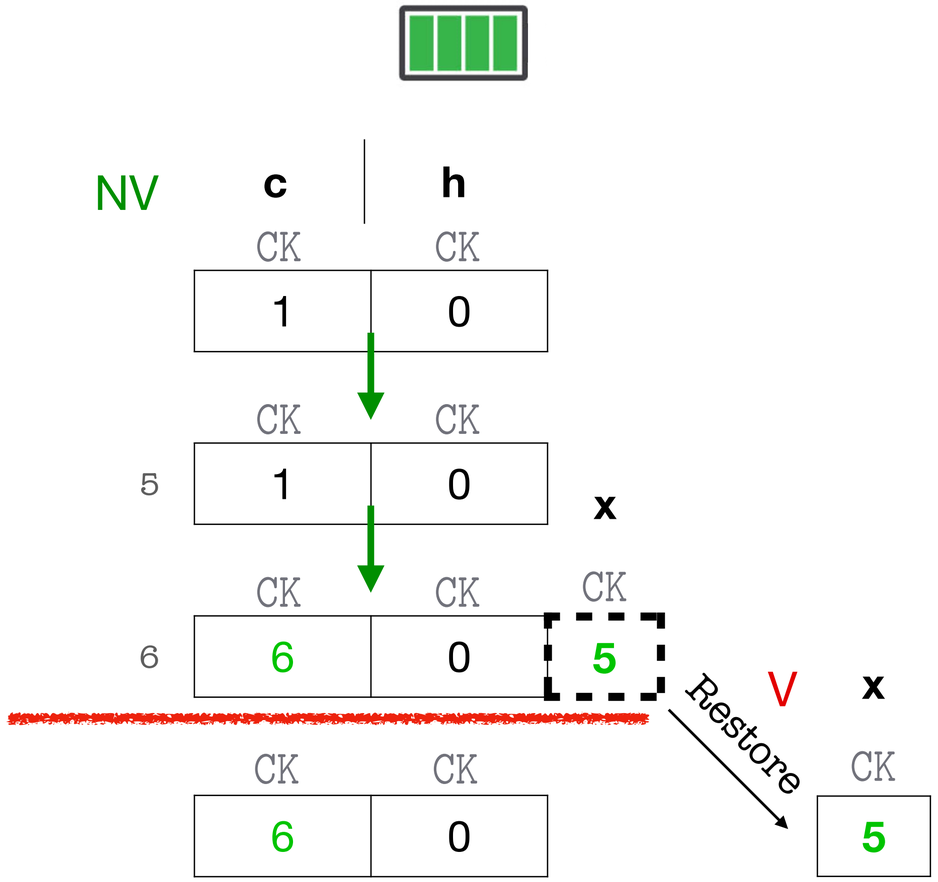
```
5  let x = 5 in
6      c:= x+1;
7  h := c
```

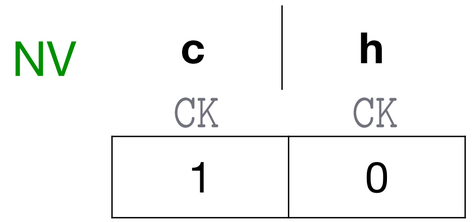




Example

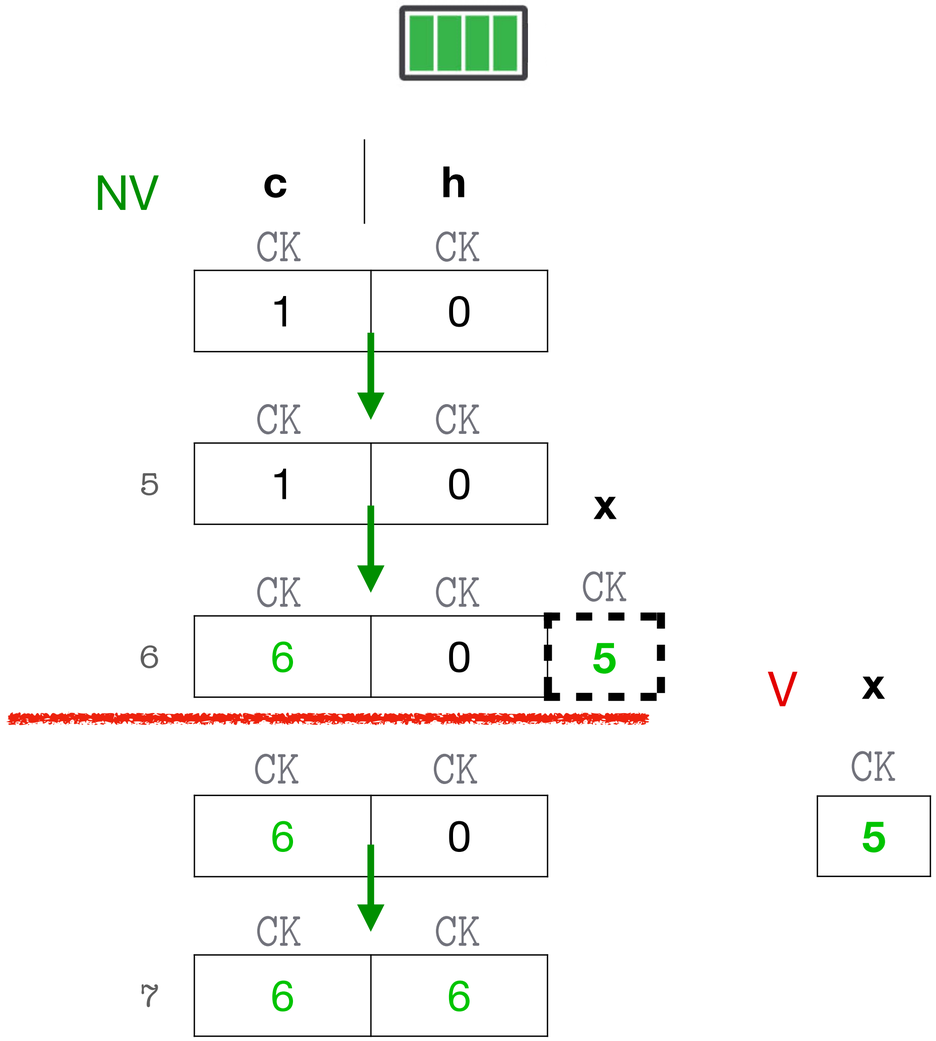
```
5  let x = 5 in
6      c:= x+1;
7  h := c
```

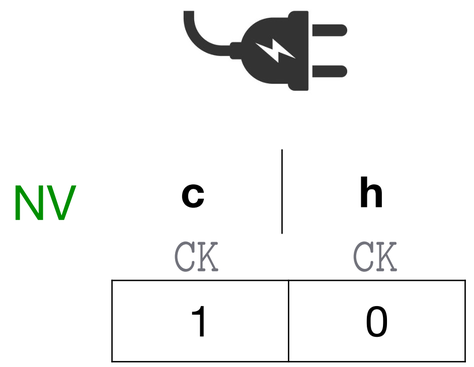




Example

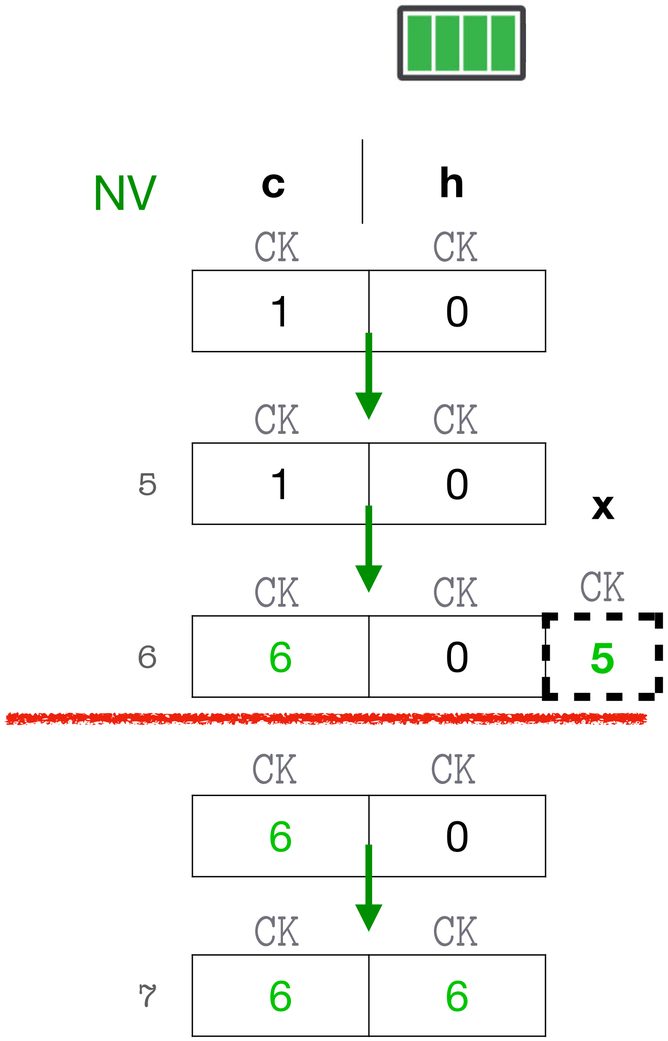
```
5  let x = 5 in
6      c:= x+1;
7  h := c
```

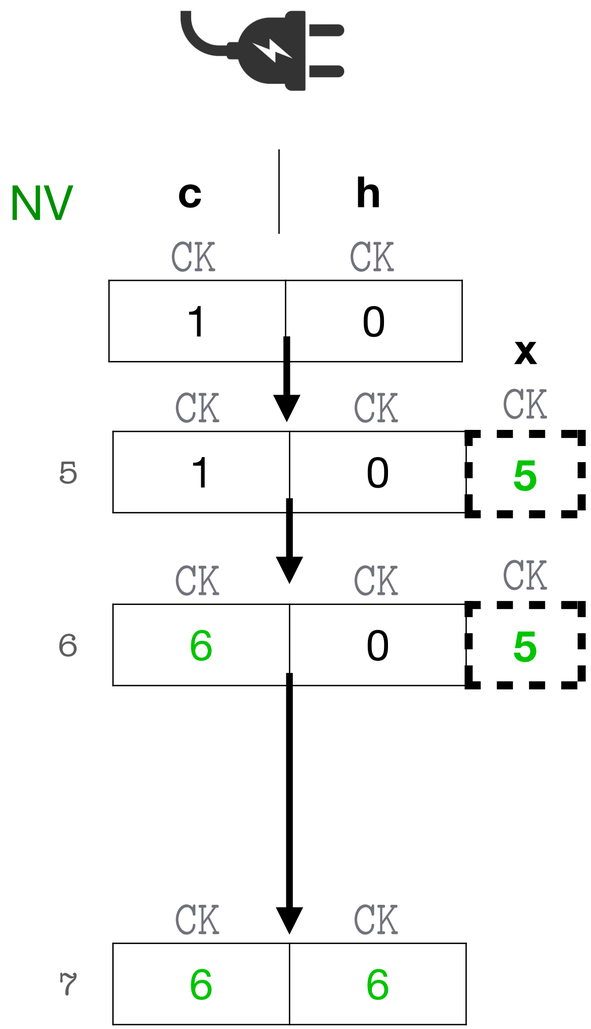




Example

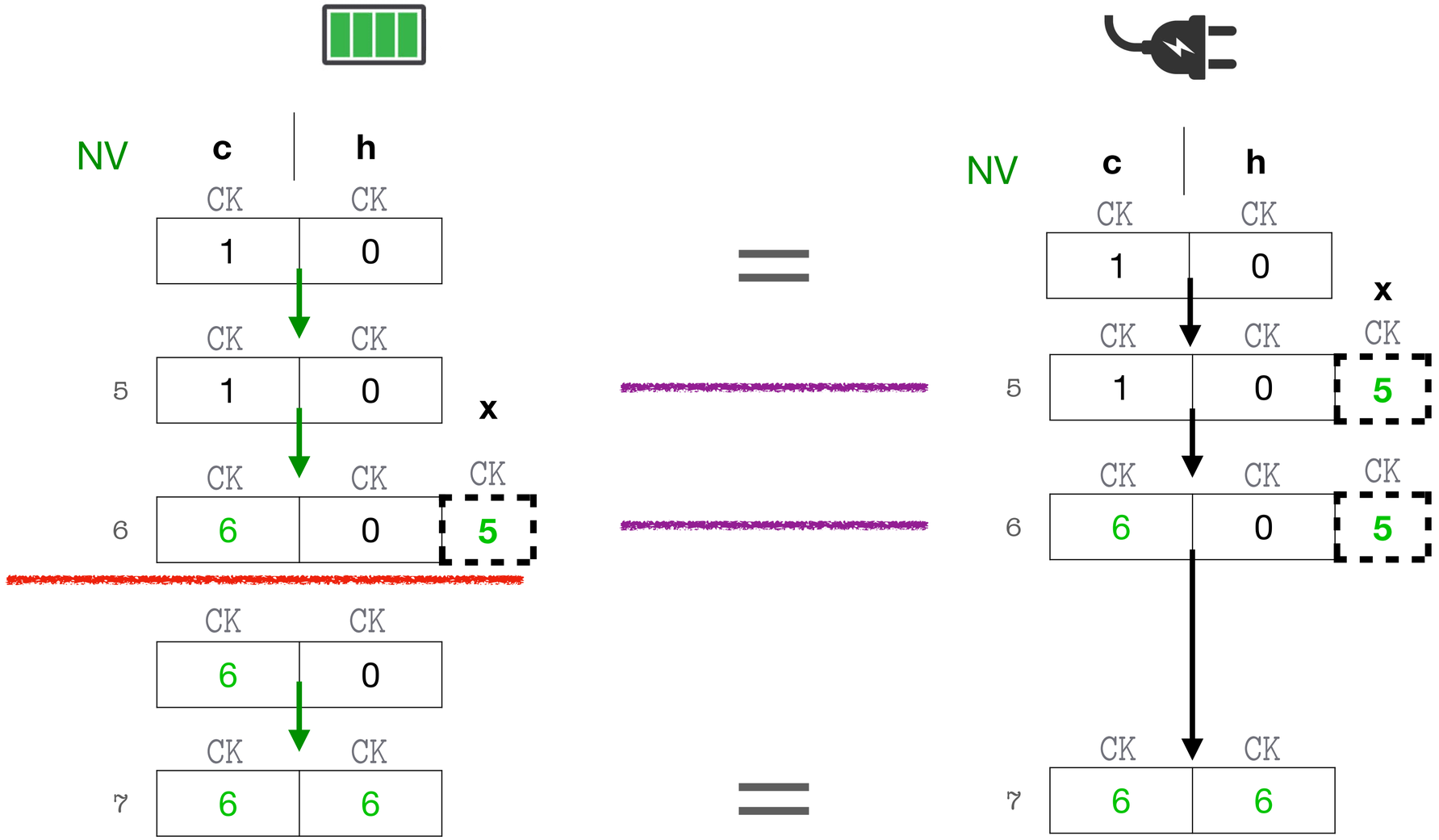
```
5  let x = 5 in
6      c:= x+1;
7  h := c
```





Example

```
5  let x = 5 in
6      c:= x+1;
7  h := c
```

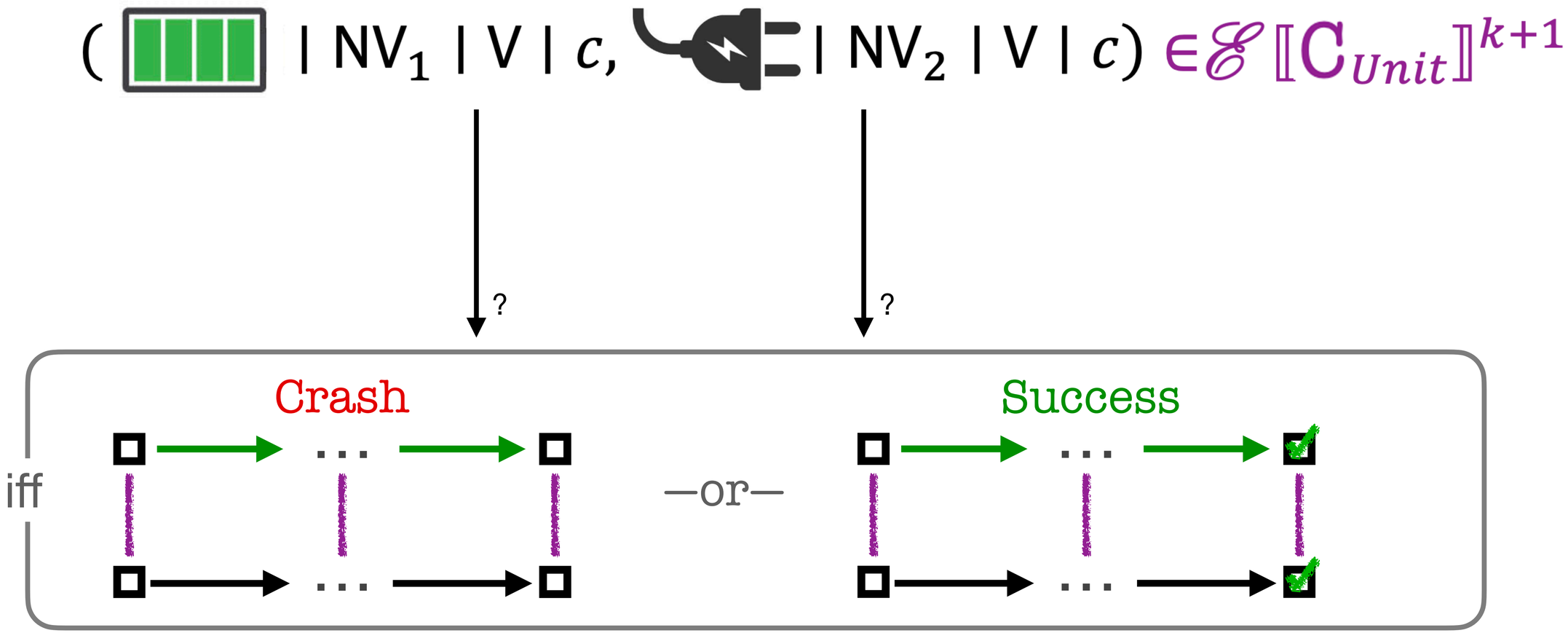


Logical Relation for JIT

$$(\ \text{🔋}\ \mid \mathsf{NV}_1 \mid \mathsf{V} \mid c,\ \text{🔌}\ \mid \mathsf{NV}_2 \mid \mathsf{V} \mid c) \in \mathscr{E}[\![\mathbf{C}_{Unit}]\!]^{k+1}$$

Same great structure,
different PwOff and Restore!

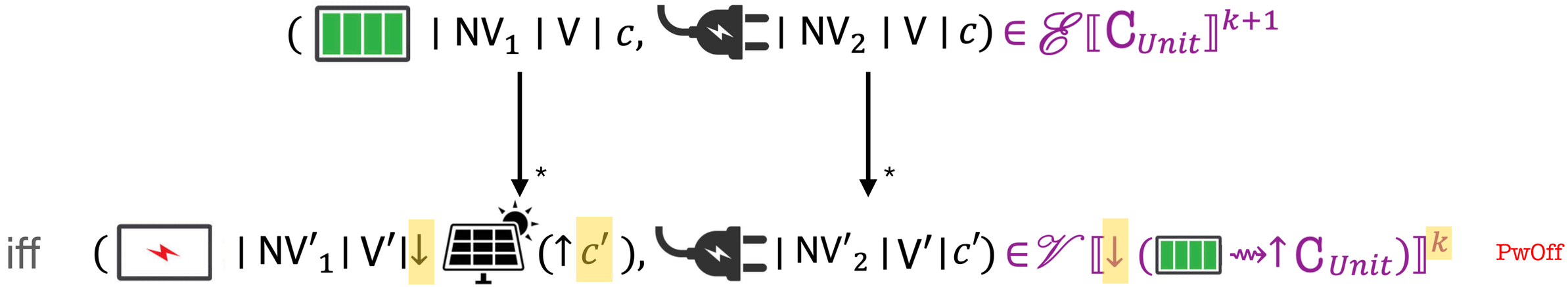
Logical Relation for JIT



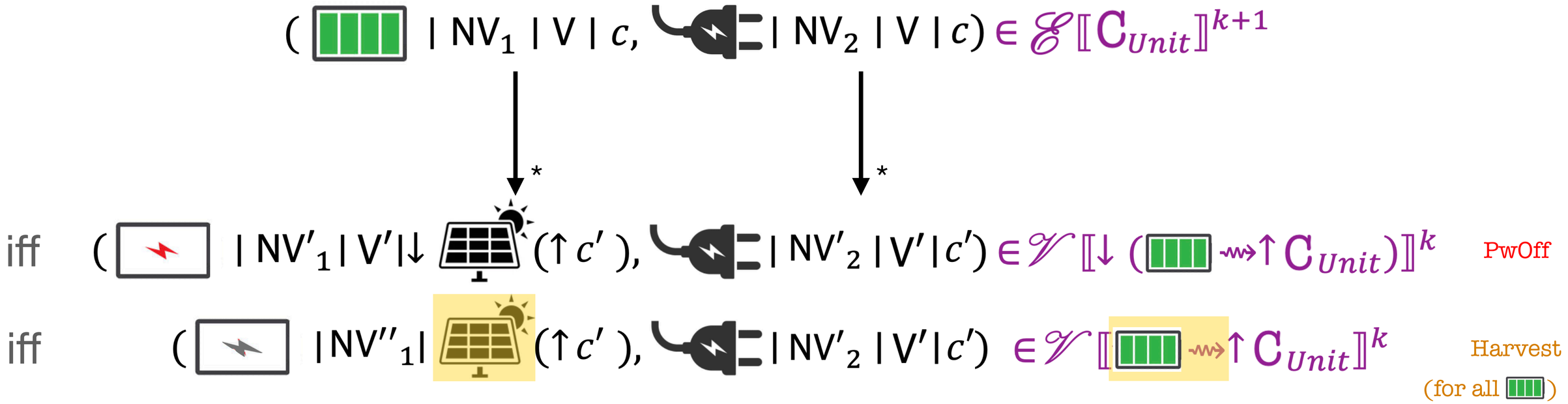
If there is a power failure…

$$(\ \text{  }\ \mid \mathrm{NV}_1 \mid \mathrm{V} \mid c,\ \text{  }\ \mid \mathrm{NV}_2 \mid \mathrm{V} \mid c) \in \mathscr{E}[\![\mathbf{C}_{Unit}]\!]^{k+1}$$

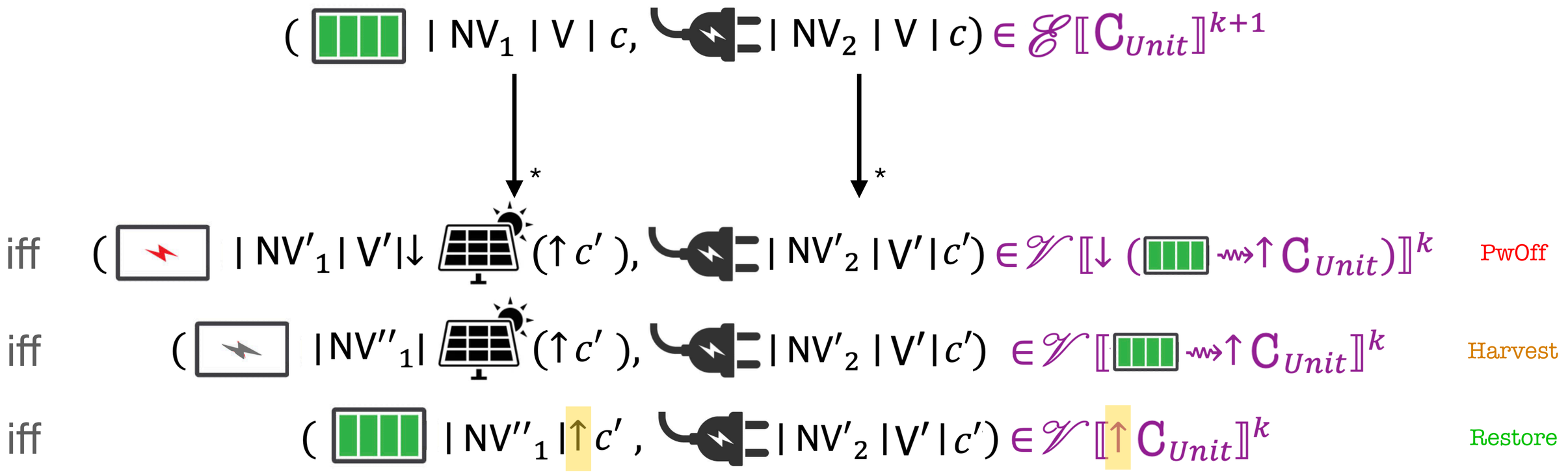
Device powers off



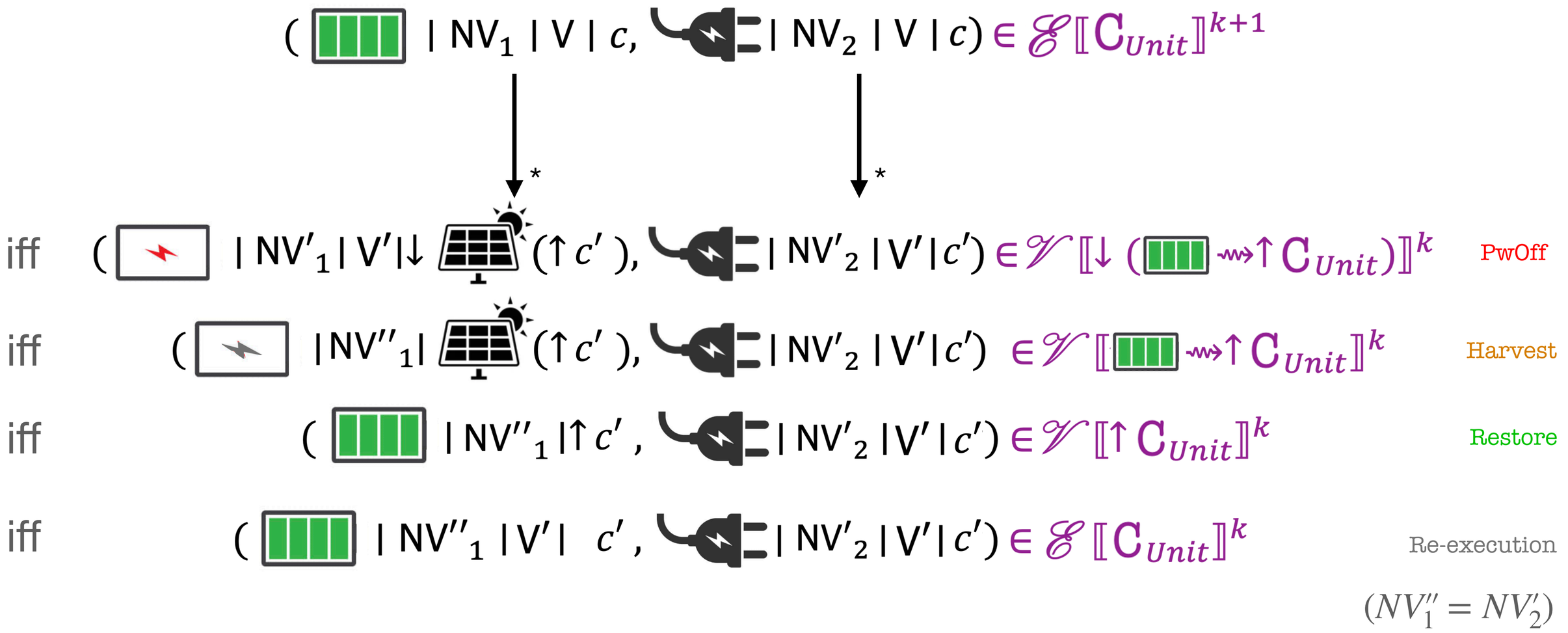
Harvest energy from the environment



Execution state is restored



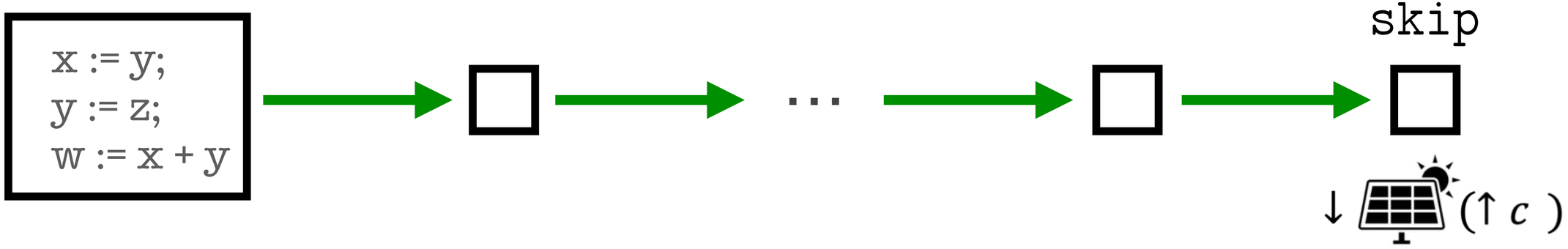
Prepare for re-execution



Outline

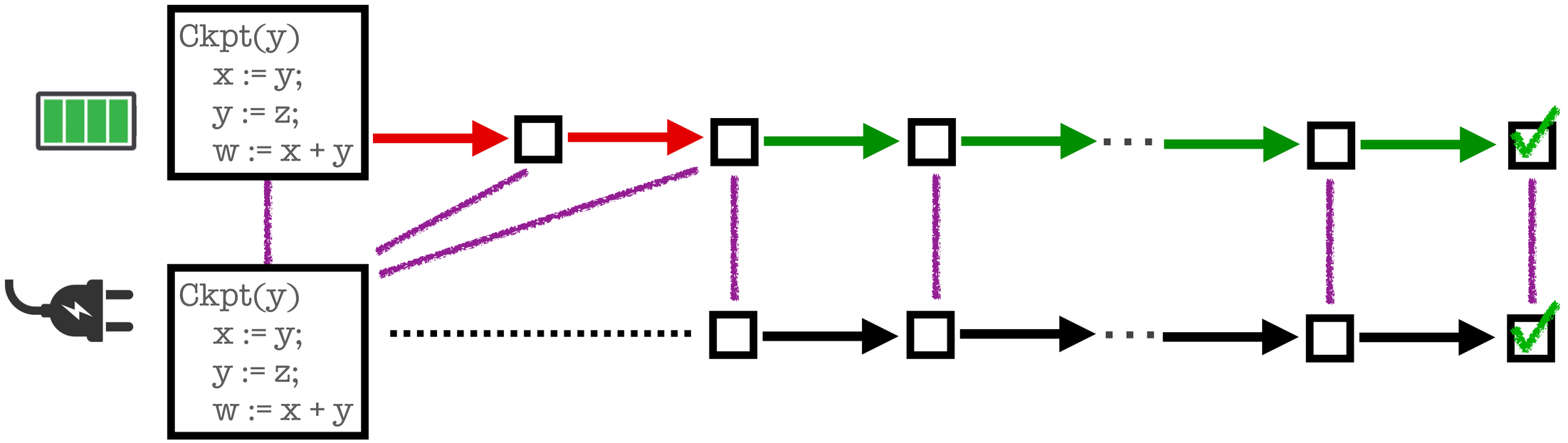
- Intermittent computing
- Overview
- Type system
- Logical relation
- JIT mode
- **Key theorems**
- Conclusion

Progress and Preservation



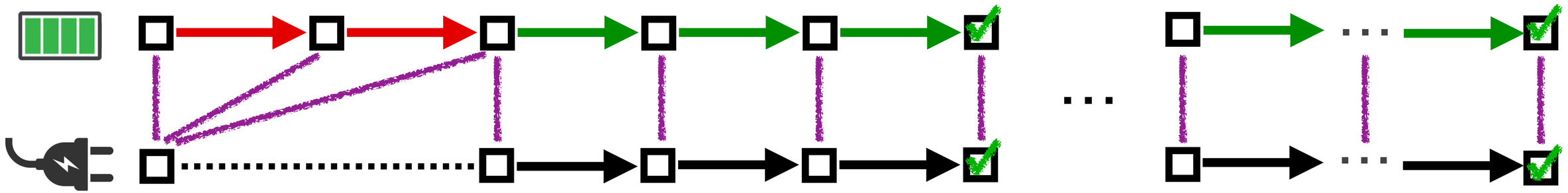
Well-typed code doesn’t get stuck and stays well-typed

Fundamental Theorem of Logical Relation



Well-typed Atomic and JIT regions are self-related

Adequacy



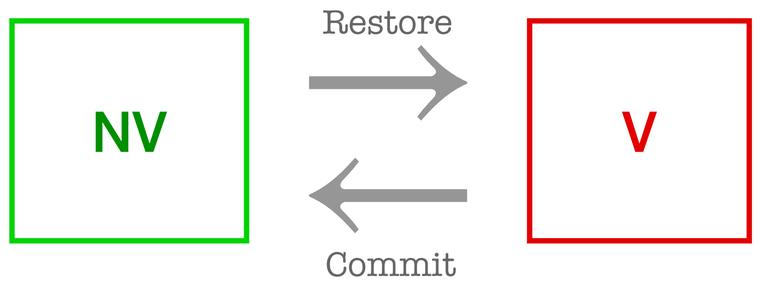
Every intermittent execution of a well-typed program can be simulated by its continuous execution

Outline

- Intermittent computing
- Overview
- Type system
- Logical relation
- JIT mode
- Key theorems
- **Conclusion**

In this work, we developed…

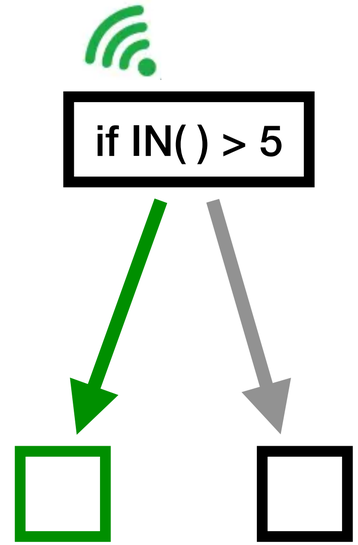
- One of the first type systems for intermittent computing
- First logical interpretation of key operations of intermittent execution
- A calculus for Crash types with type soundness
- A novel logical relation to characterize correctness



$$\mathtt{C} = \downarrow\uparrow \mathtt{unit} \vee \downarrow(\mathit{nat} \rightsquigarrow \uparrow \mathtt{C})$$

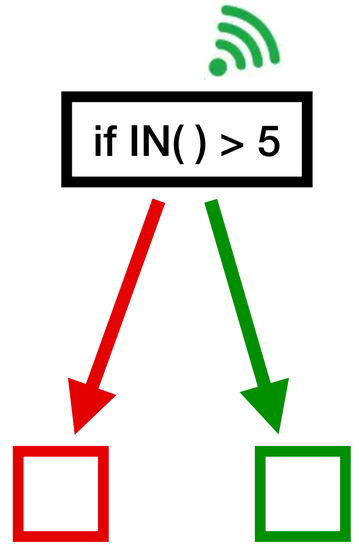
Future work

- Repeated I/O bugs



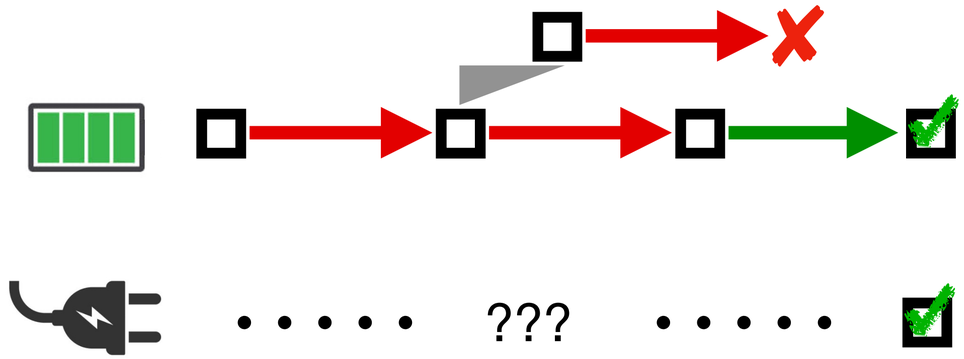
Future work

- Repeated I/O bugs



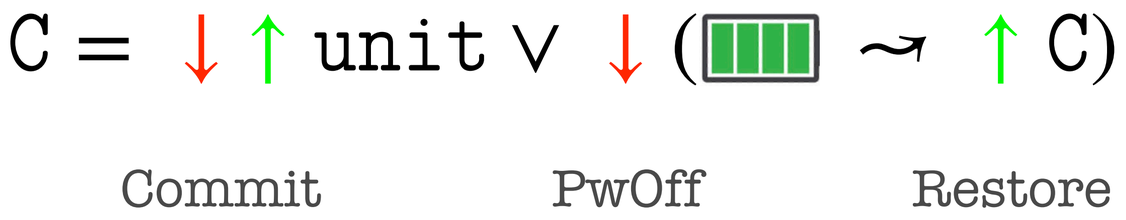
Future work

- Repeated I/O bugs
- Shared memory concurrency



Future work

- Repeated I/O bugs
- Shared memory concurrency
- Other systems crash too!



Modal Crash Types for Intermittent Computing

Myra Dotzel
PhD student, CSD, CMU

Joint work with Farzaneh Derakhshan, Milijana Surbatovich, and Limin Jia