Programming Research Laboratory

An axiomatic basis for bidirectional programming

Josh H-S Ko[1] and **Zhenjiang Hu**[1,2]

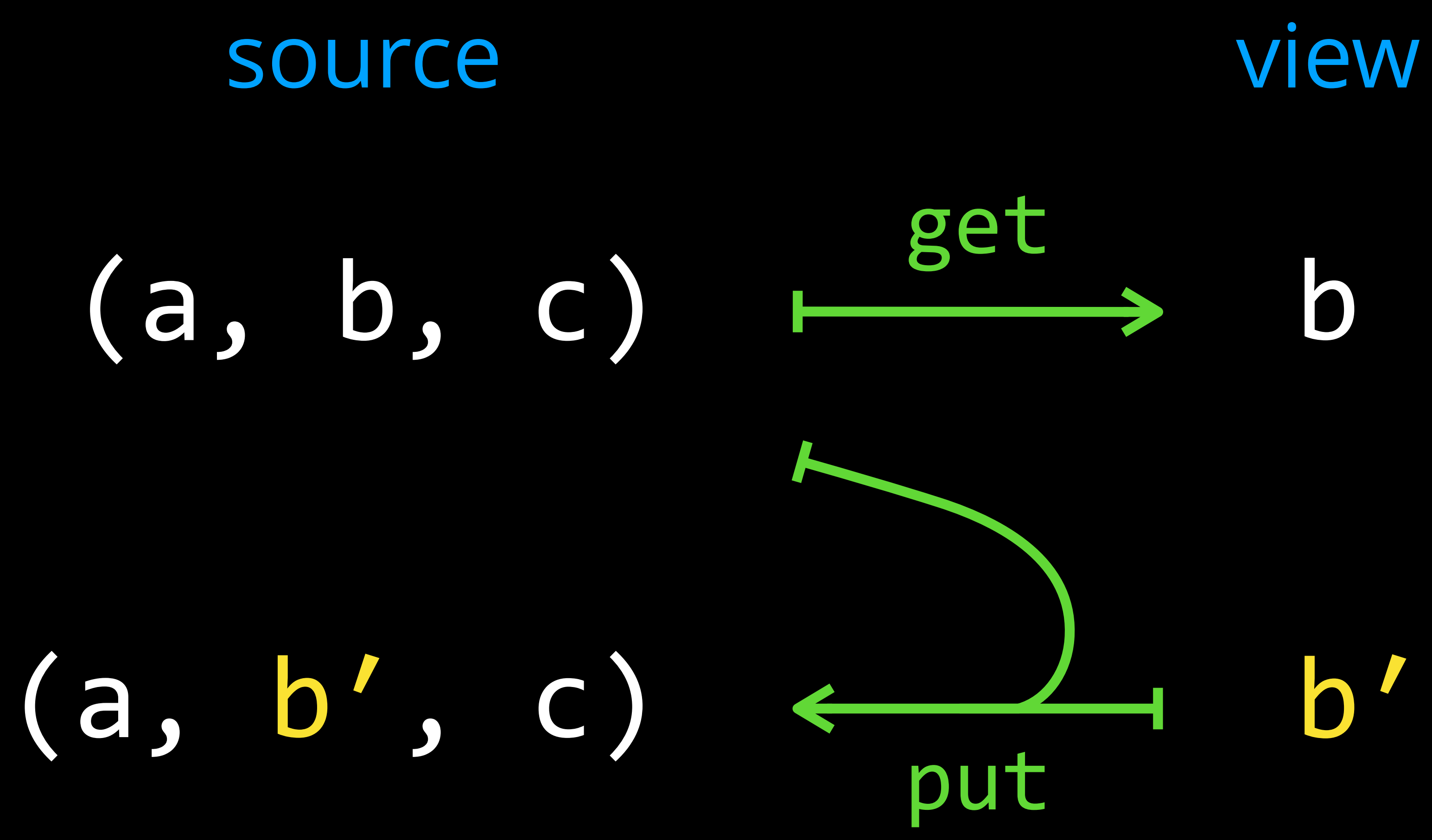
[1] National Institute of Informatics (NII), Japan
[2] SOKENDAI (The Graduate University for Advanced Studies), Japan

Symposium on Principles of Programming Languages (POPL)
11 January 2018, Los Angeles, CA, US

Lenses

(asymmetric & state-based)

Extraction & update

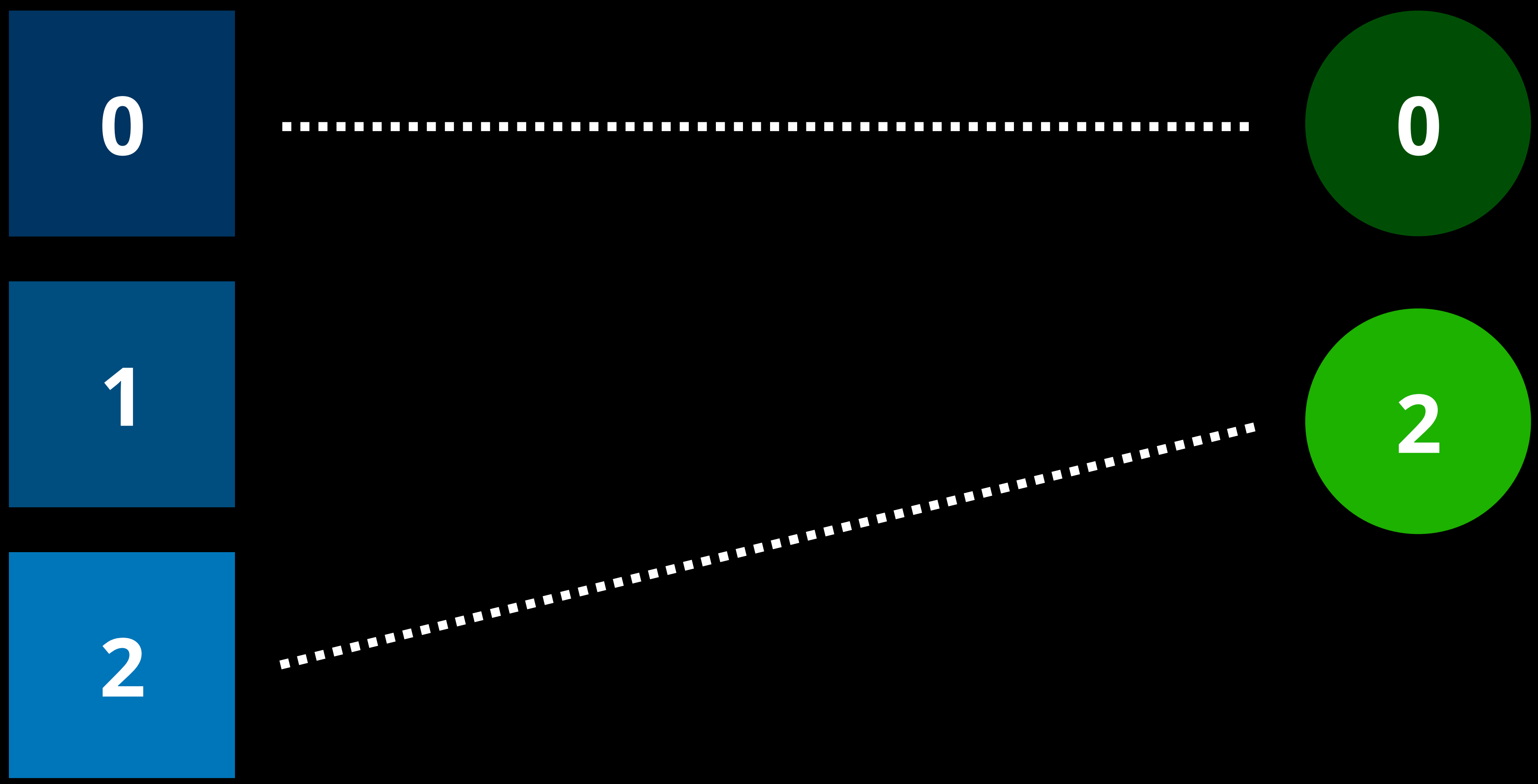


Well-behaved lenses

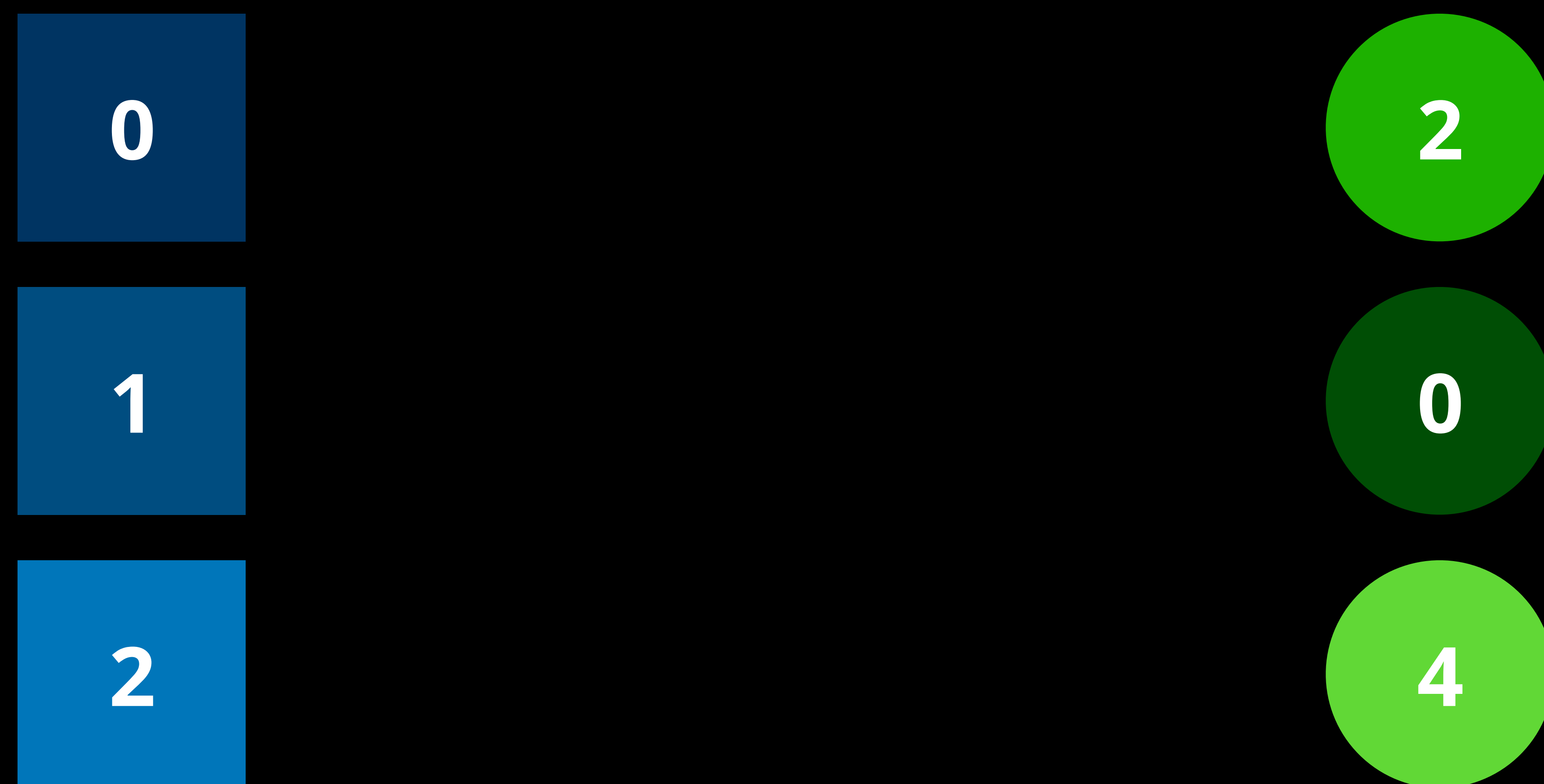
```
get : S → V
put : S → V → S

get (put s v) = v
put s (get s) = s
```

Get

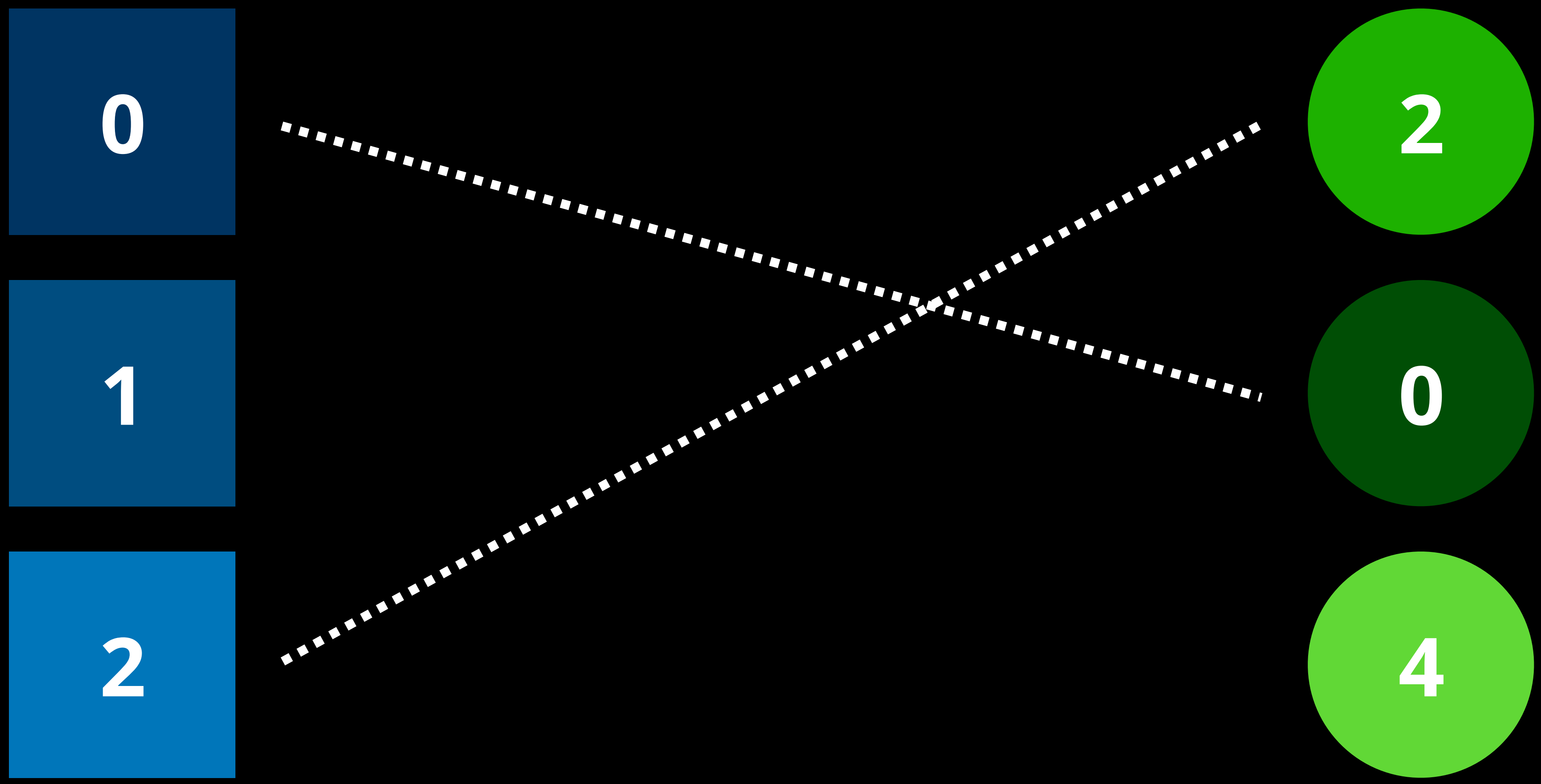
`map f ∘ filter p`



Put

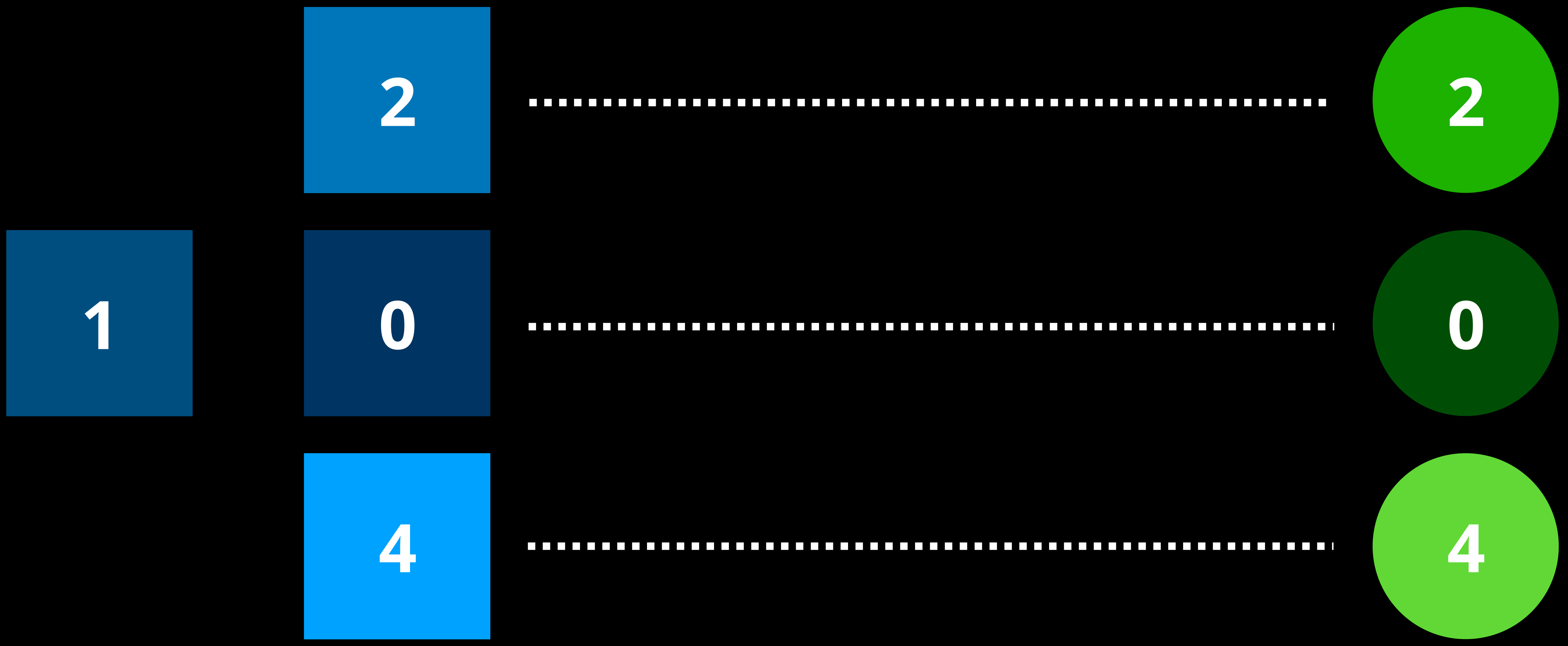


Put

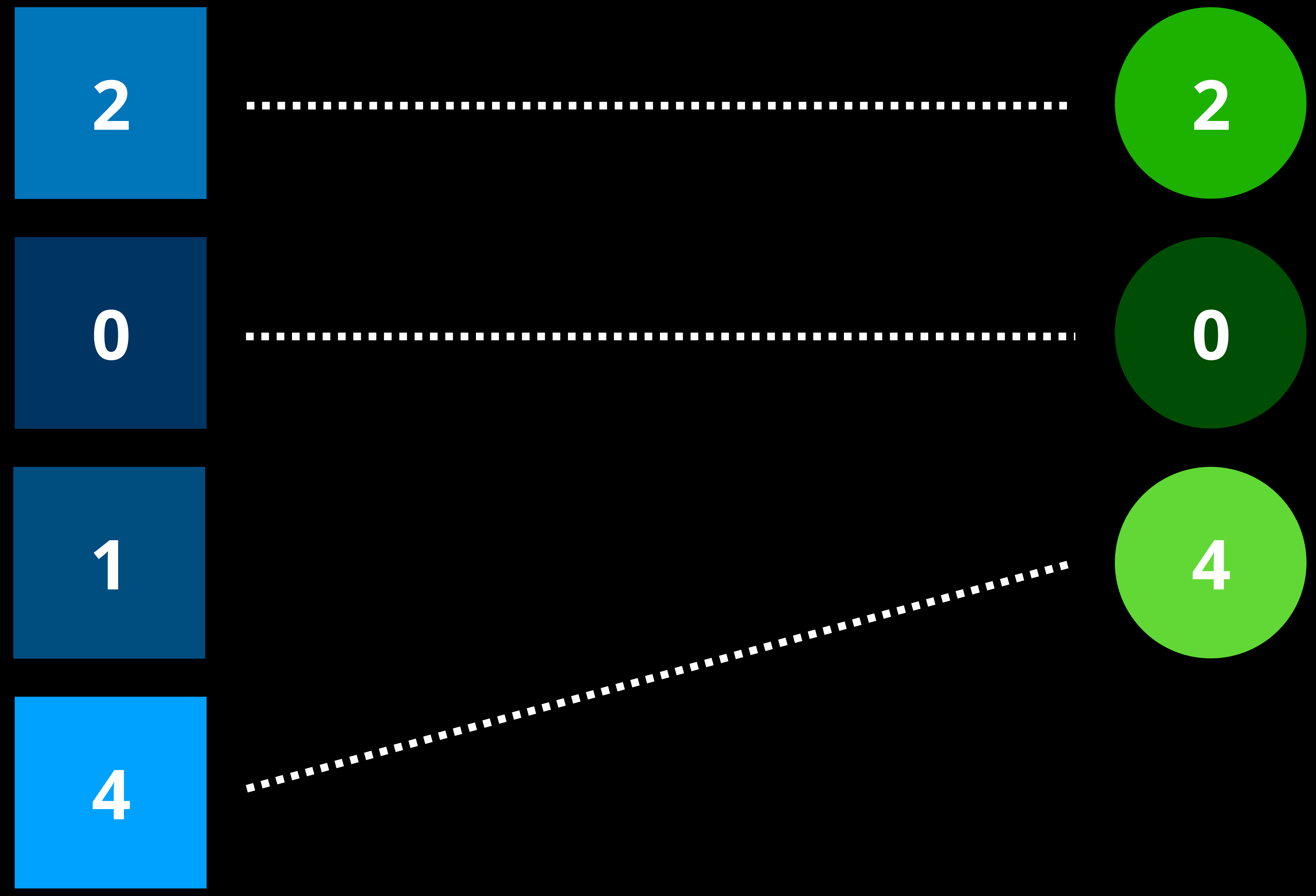


Put

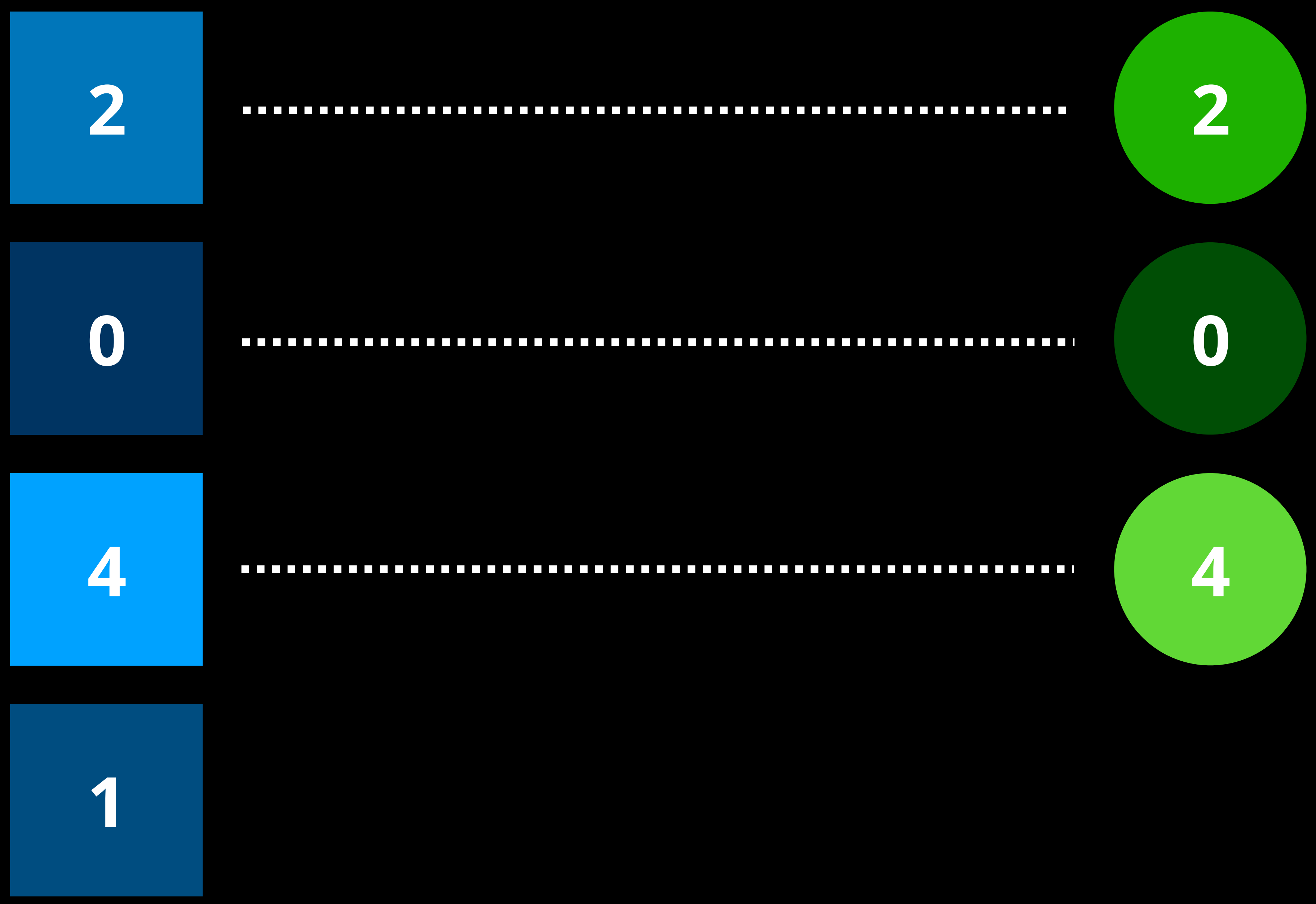
Put



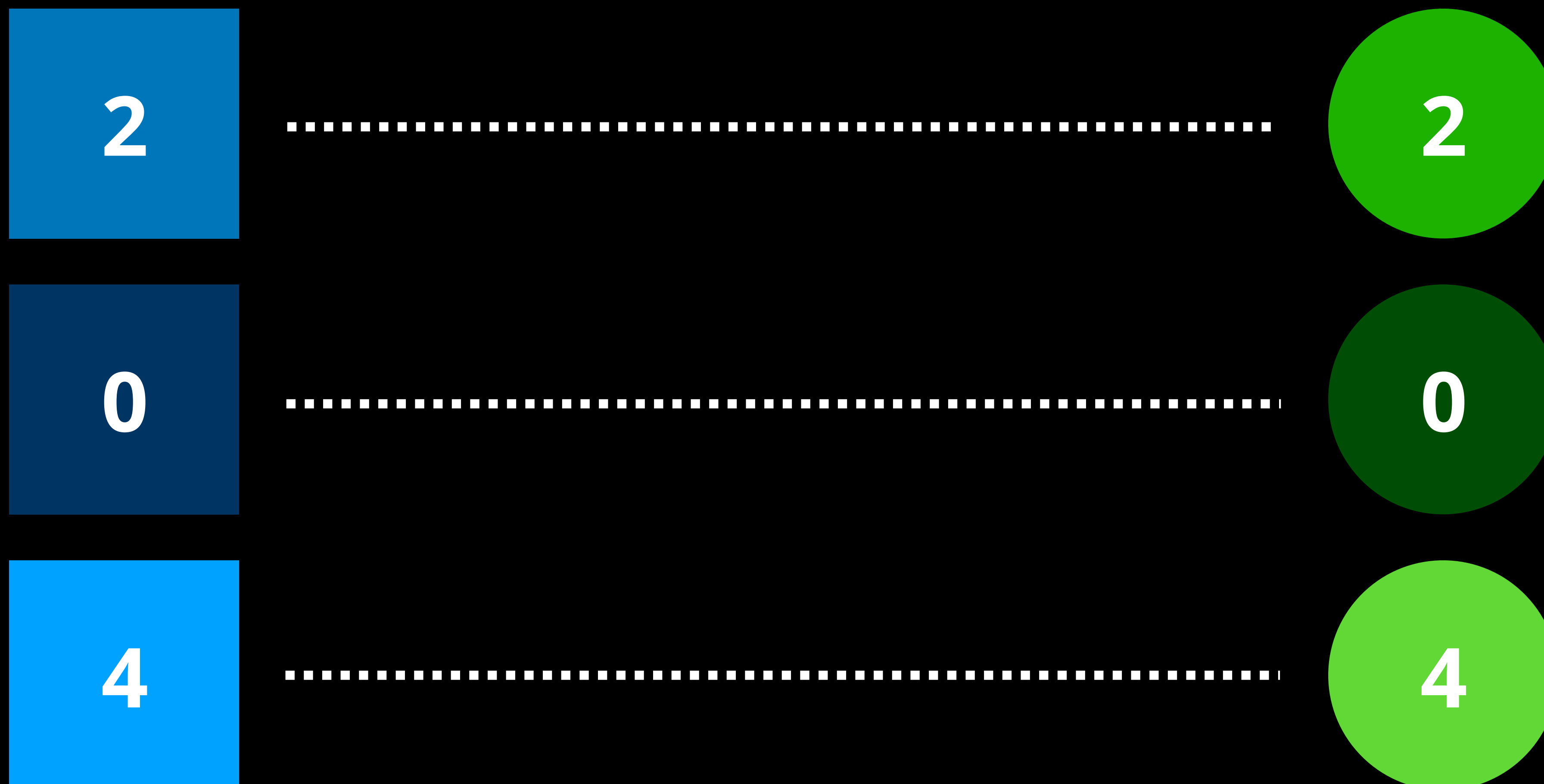
Put



Put



Put



Bidirectional programming languages

one program for two directions

Bidirectional programming

write one program to control two directions

“Get-based” approach

```
  map f <alignment strategy>
∘ filter p <management of ignored elements>
```

“Put-based” approach

```
align p match b create conceal =
  case
    normal [] [] exit []
      rearrV [] -> ()
        skip const ()
    normal (s::_) (v::_) | p s && match s v exit (s::_) | p s
      rearrS (s::ss) -> (s, ss)
        rearrV (v::vs) -> (v, vs)
          b * align p match b create conceal
    adaptive (s::_) [] | p s
      \ss -> let (prefix, remaining) = span p ss
             in catMaybes (map conceal prefix) ++ remaining
```

If put is well-behaved with both get and get′,

then get = get′.

bidirectional programming ➡ unidirectional programming

You still need to figure out the get behaviour!

Yes, but we can do it just from the put direction…

Get is really
a part of put.

Synchronisation

Maintaining a consistency relation

```
get : S → V
```

"executable" consistency relation

```
put : S → V → S
```

consistency restorer

```
get (put s v) = v
```

correctness

```
put s (get s) = s
```

hippocraticness

“Get-based” approach

First: write a consistency relation

```
  map f <alignment strategy>
∘ filter p <management of ignored elements>
```

Second: annotate the consistency relation with restoration behaviour

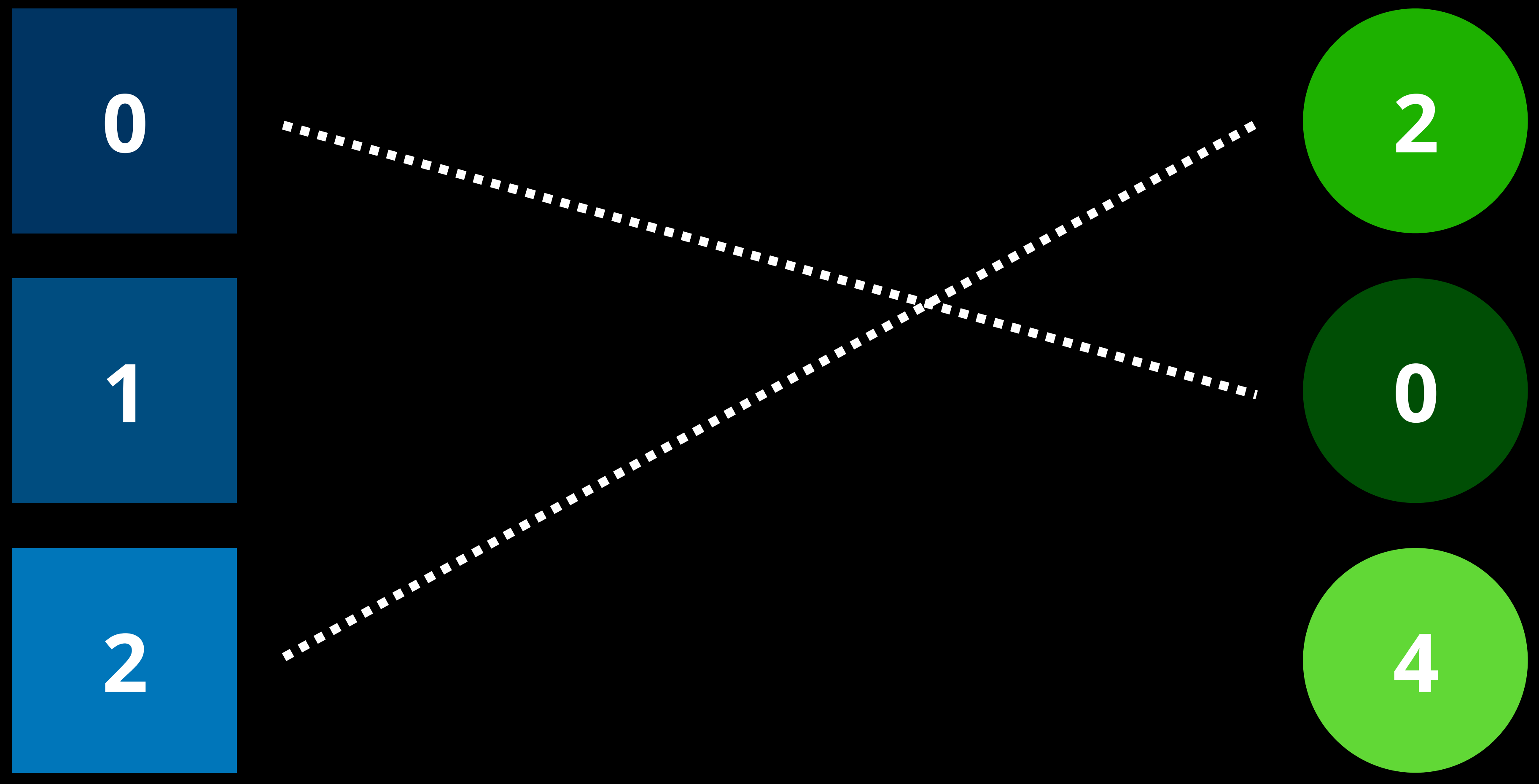
“Put-based” approach

```
align p match b create conceal =
  case
    normal [] [] exit []
      rearrV [] -> ()
        skip const ()
    normal (s::_) (v::_) | p s && match s v exit (s::_) | p s
      rearrS (s::ss) -> (s, ss)
        rearrV (v::vs) -> (v, vs)
          b * align p match b create conceal
    adaptive (s::_) [] | p s
```

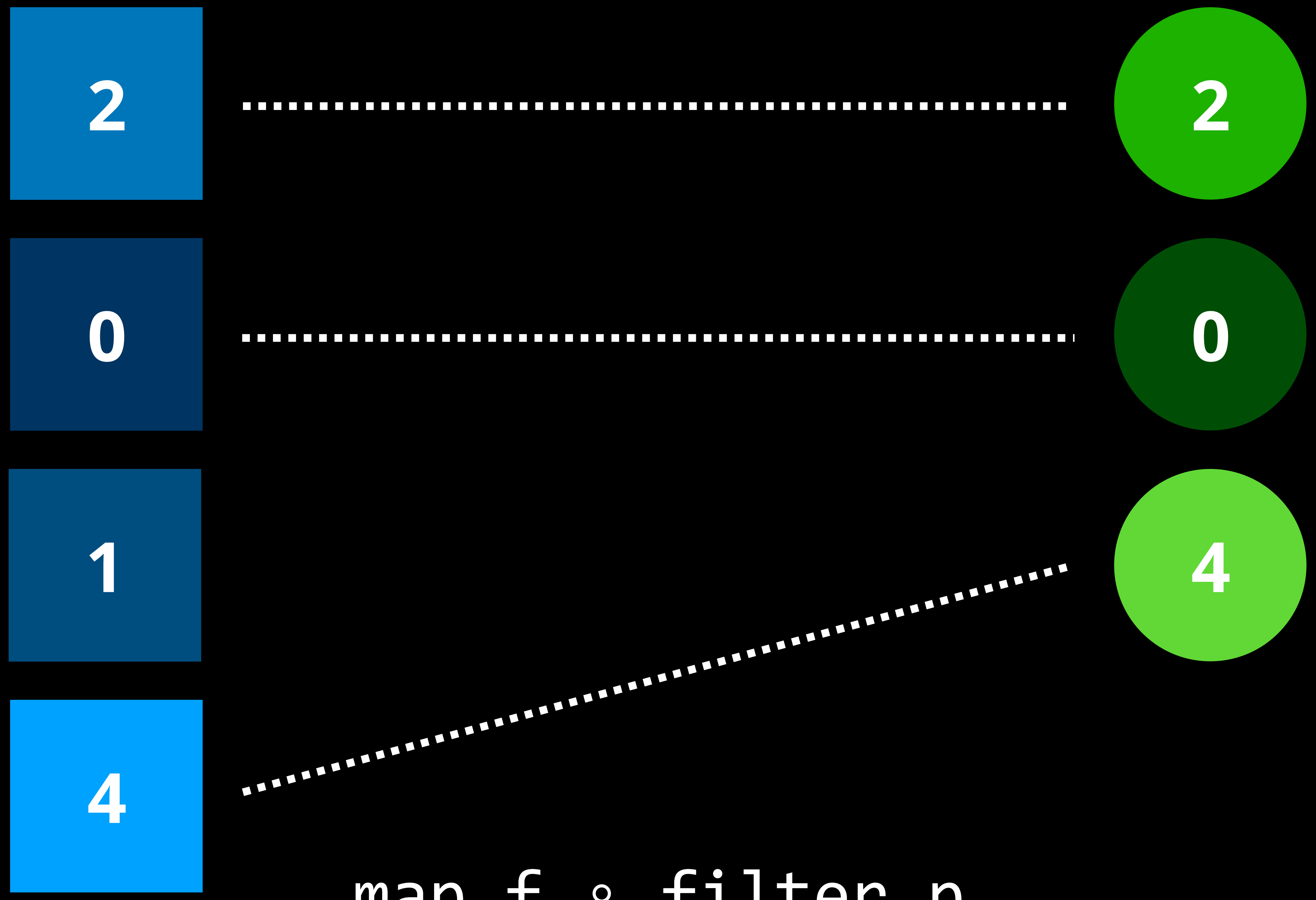
First: write a program to restore a consistency relation in mind

Second: the consistency relation becomes executable for free

Put



Put



map f ∘ filter p

To program a consistency restorer, the programmer must have a consistency relation in mind.

a put-based language makes ↑ executable

Formalise!

in terms of a program logic

BiGUL

Bidirectional Generic Update Language

lens combinators

```
rearrV v -> (v, ())
  replace * skip const ()
```

atomic lenses

Hoare-style logic

$$\frac{}{\{\ s\ v \mid \mathrm{True}\ \}\ \ \mathbf{replace}\ \ \{\ s'\ s\ v \mid s' = v\ \}}$$

$$\frac{}{\{\emptyset\}\ \mathbf{fail}\ \{\emptyset\}} \qquad \frac{}{\{\_\ \_\}\ \mathbf{replace}\ \{s'\ \_\ v \mid s' = v\}} \qquad \frac{}{\{s\ v \mid f\ s = v\}\ \mathbf{skip}\ f\ \{s'\ s\ \_ \mid s' = s\}}$$

$$\frac{\{L\}\ l\ \{L'\} \quad \{R\}\ r\ \{R'\}}{\{L * R\}\ l * r\ \{L' * R'\}} \qquad \frac{T \subseteq R \quad \{R\}\ b\ \{R'\} \quad R' \cap \langle\, \_\ s\ v \mid T\ s\ v \,\rangle \subseteq T'}{\{T\}\ b\ \{T'\}}$$

$$\frac{\{\, s\ wpat \mid R\ s\ \overline{wpat}\,\}\ \ b\ \ \{\, s'\ s\ wpat \mid R'\ s'\ s\ \overline{wpat}\,\}}{\{\, s\ vpat \mid R\ s\ \overline{vpat}\,\}\ \ \mathbf{rearrV}\ vpat \rightarrow wpat \looparrowright b\ \ \{\, s'\ s\ vpat \mid R'\ s'\ s\ \overline{vpat}\,\}}$$

$$\frac{\{\, tpat\ v \mid R\ \overline{tpat}\ v\,\}\ \ b\ \ \{\, tpat'\ tpat\ v \mid R'\ \overline{tpat'}\ \overline{tpat}\ v\,\}}{\{\, spat\ v \mid R\ \overline{spat}\ v\,\}\ \ \mathbf{rearrS}\ spat \rightarrow tpat \looparrowright b\ \ \{\, spat'\ spat\ v \mid R'\ \overline{spat'}\ \overline{spat}\ v\,\}}$$

$\forall(\mathbf{normal}\ M\ \mathbf{exit}\ E \looparrowright b) \in bs.$

$\{\, R \cap \widehat{M}\,\}\ \ b\ \ \{\, R' \cap \langle\, s'\ \_\ v \mid \widehat{M}\ s'\ v \wedge \widehat{E}\ s' \,\rangle \,\}$

$\forall(\mathbf{adaptive}\ M \looparrowright f) \in bs.$

Reasoning

```
{ _ _ }
rearrV v → (v, ())
  { _ (_, ()) }
    { _ _ }
    replace
    { w′ _ v | w′ = v }
  * { _ () }
    { _ () | const () s = () }
    skip const ()
    { h′ h () | h′ = h }
  { (w′, h′) (_, h) (v, ()) | w′ = v ∧ h′ = h }
{ (w′, h′) (_, h) v | w′ = v ∧ h′ = h }
```

Main theorem

```
If    { s v | R s v }  b  { s' _ v | C s' v }
then  b.get ∩ R  ⊆  C
```

A part of

get behaviour can be found in put triples.

Main theorem

```
If    { s v | R s v }  b  { s' _ v | C s' v }
then  b.get ∩ R  ⊆  C

      { s v | False }  b  { s' _ v | C s' v }
      { s v | True  }  b  { s' _ v | C s' v }
```

Domain of get

Range of put

Range triples

```
{{ s v | R s v }} b {{ s' | P' s' }}
```


$$\frac{}{\{\!\{\, \emptyset \,\}\!\}\ \ \mathbf{fail}\ \ \{\!\{\, \emptyset \,\}\!\}} \qquad \frac{}{\{\!\{\, s\ v \mid s = v \,\}\!\}\ \ \mathbf{replace}\ \ \{\!\{\, \_ \,\}\!\}} \qquad \frac{}{\{\!\{\, s\ v \mid f\ s = v \,\}\!\}\ \ \mathbf{skip}\ f\ \ \{\!\{\, \_ \,\}\!\}}$$

$$\frac{\{\!\{\, L \,\}\!\}\ \ l\ \ \{\!\{\, P' \,\}\!\} \qquad \{\!\{\, R \,\}\!\}\ \ r\ \ \{\!\{\, Q' \,\}\!\}}{\{\!\{\, L * R \,\}\!\}\ \ l * r\ \ \{\!\{\, P' * Q' \,\}\!\}} \qquad \frac{R \cap \langle\, s\ \_ \mid Q'\ s \,\rangle \subseteq T \qquad \{\!\{\, R \,\}\!\}\ \ b\ \ \{\!\{\, P' \,\}\!\} \qquad Q' \subseteq P'}{\{\!\{\, T \,\}\!\}\ \ b\ \ \{\!\{\, Q' \,\}\!\}}$$

$$\frac{\{\!\{\, s\ wpat \mid R\ s\ \overline{wpat} \,\}\!\}\ \ b\ \ \{\!\{\, P' \,\}\!\}}{\{\!\{\, s\ vpat \mid R\ s\ \overline{vpat} \,\}\!\}\ \ \mathbf{rearrV}\ vpat \rightarrow wpat \hookrightarrow b\ \ \{\!\{\, P' \,\}\!\}}$$

$$\frac{\{\!\{\, tpat\ v \mid R\ \overline{tpat}\ v \,\}\!\}\ \ b\ \ \{\!\{\, tpat \mid P'\ \overline{tpat} \,\}\!\}}{\{\!\{\, spat\ v \mid R\ \overline{spat}\ v \,\}\!\}\ \ \mathbf{rearrS}\ spat \rightarrow tpat \hookrightarrow b\ \ \{\!\{\, spat \mid P'\ \overline{spat} \,\}\!\}}$$

$$\frac{\forall n = (\mathbf{normal}\ M\ \mathbf{exit}\ E \hookrightarrow b) \in bs.\quad \{\!\{\, R \cap \widehat{M} \,\}\!\}\ \ b\ \ \{\!\{\, P'_n \,\}\!\}}{\{\!\{\, R \,\}\!\}\ \ \mathbf{case} \hookrightarrow bs\ \ \{\!\{\, \mathcal{P}' \,\}\!\}}$$

where

$$\mathcal{P}' = \bigcup [\, P'_n \cap \widehat{E} \mid n = (\mathbf{normal}\ M\ \mathbf{exit}\ E \hookrightarrow b) \in bs \,]$$

Main theorem MK II

```
If     { s v | R s v }  b  { s’ _ v | C s’ v }
and   {{ s v | R s v }} b {{ s’ | P’ s’ }}

then  b.get is defined on P’
and   b.get|P’  ⊆  C
```

Get behaviour can be found in put and range triples.

Also in the paper

- An introduction to BiGUL in terms of the axiomatic semantics
- Recursion rules and key-based alignment
- Everything formalised in Agda

41

An Axiomatic Basis for Bidirectional Programming

HSIANG-SHANG KO, National Institute of Informatics, Japan
ZHENJIANG HU*, National Institute of Informatics, Japan

Get is really
a part of put.

↓

Get behaviour can be found in put and range triples.

bidirectional programming ➔ unidirectional programming

```
get : S → V
put : S → V → S

get (put s v) = v
put s (get s) = s
```

unidirectional programming

↑?

one program for two directions

An axiomatic basis for bidirectional programming

Josh H-S Ko and Zhenjiang Hu (NII & SOKENDAI, Japan)

{ R } b { R′ }

{{ R }} b {{ P′ }}