

# **Inside Adaption**

**The Demanded Computation Graph (DCG)**

<https://docs.rs/adapton/0.3.2/adapton/macros/index.html#demand-driven-change-propagation>

```
#[test]
fn avoid_divide_by_zero () {
    use adapton::macros::*;
    use adapton::engine::*;
    manage::init_dcg();

    // Construct two mutable inputs, `num` and `den`, a
    // computation `div` that divides the numerator in `num` by
    // the denominator in `den`, and a thunk `check` that first
    // checks whether the denominator is zero (returning None if
    // so) and if non-zero, returns the value of the division.

    let num  = cell!(42);
    let den  = cell!(2);

    // In Rust, cloning is explicit:
    let den2 = den.clone(); // here, we clone the _global reference_ to the cell.
    let den3 = den.clone(); // here, we clone the _global reference_ to the cell, again.

    let div  = thunk! [ get!(num) / get!(den) ];
    let check = thunk! [ if get!(den2) == 0 { None } else { Some(get!(div)) } ];

    assert_eq!(get!(check), Some(21));
    set(&den3, 0); assert_eq!(get!(check), None);
    set(&den3, 2); assert_eq!(get!(check), Some(21));
}
```

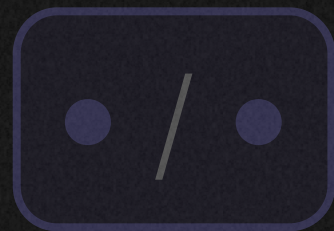
**num**

42

**den**

2

*Allocate  
input cells*



**div**

**check**

if • == 0  
then None  
else Some(•)



```
let num = cell(42)
let den = cell(2)
let div = thunk [
  get(num) / get(den)
]
let check = thunk [
  if get(den) == 0
  then None
  else Some(get(div))
]

get(check)
set(den, 0); get(check)
set(den, 2); get(check)
```



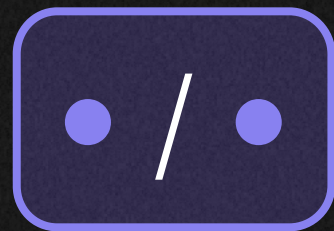
num

42

den

2

*Allocates  
thunks ...*



div

check

if • == 0  
then None  
else Some(•)



```
let num = cell(42)
let den = cell(2)
let div = thunk [
  get(num) / get(den)
]
let check = thunk [
  if get(den) == 0
  then None
  else Some(get(div))
]
```

```
get(check)
set(den, 0); get(check)
set(den, 2); get(check)
```

num

42

den

2



check

```
if • == 0
then None
else Some(•)
```

```
let num = cell(42)
let den = cell(2)
let div = thunk [
  get(num) / get(den)
]
let check = thunk [
  if get(den) == 0
  then None
  else Some(get(div))
]
```



***Demand output,  
mutate inputs***



num

42

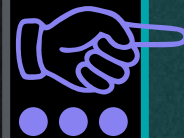
den

2



check

if • == 0  
then None  
else Some(•)



From-scratch  
evaluation

root

```
let num = cell(42)
let den = cell(2)
let div = thunk [
  get(num) / get(den)
]
let check = thunk [
  if get(den) == 0
  then None
  else Some(get(div))
]

get(check)
set(den, 0); get(check)
set(den, 2); get(check)
```

num

42

den

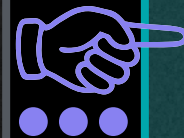
2

• / •

div

check

if • == 0  
then None  
else Some(•)



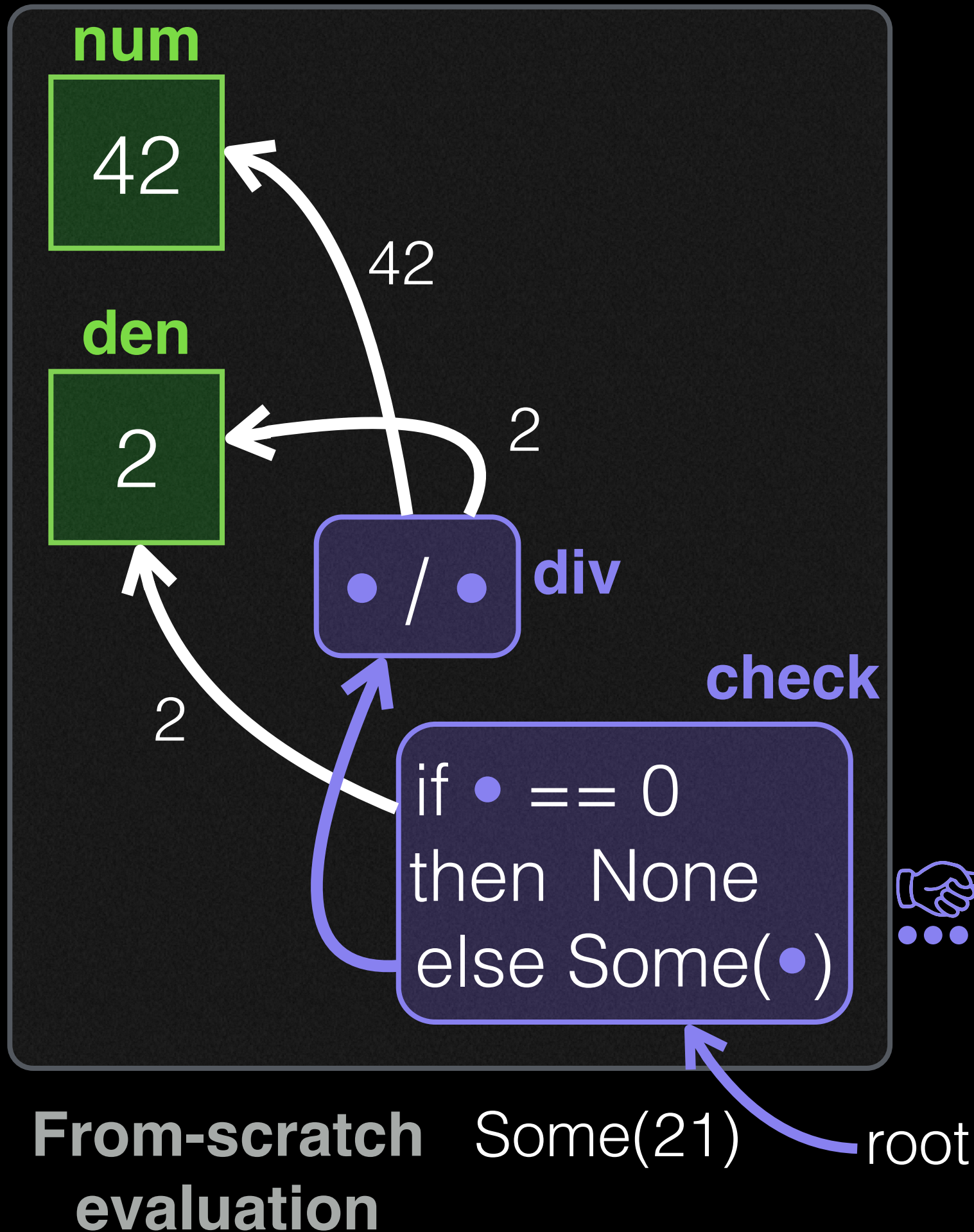
```
let num = cell(42)
let den = cell(2)
let div = thunk [
  get(num) / get(den)
]
let check = thunk [
  if get(den) == 0
  then None
  else Some(get(div))
]

get(check)
set(den, 0); get(check)
set(den, 2); get(check)
```

From-scratch  
evaluation

root

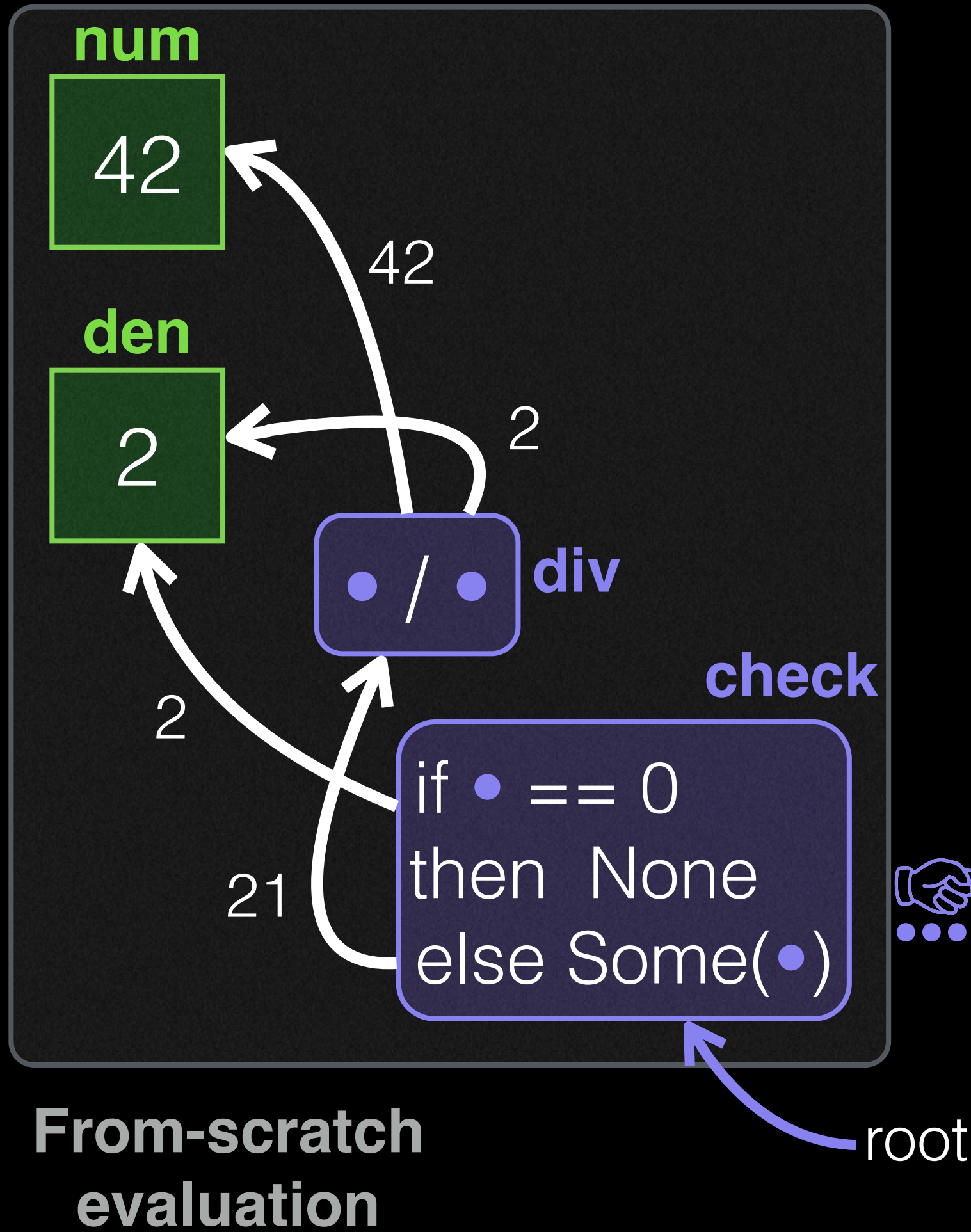




```
let num = cell(42)
let den = cell(2)
let div = thunk [
  get(num) / get(den)
]
let check = thunk [
  if get(den) == 0
  then None
  else Some(get(div))
]

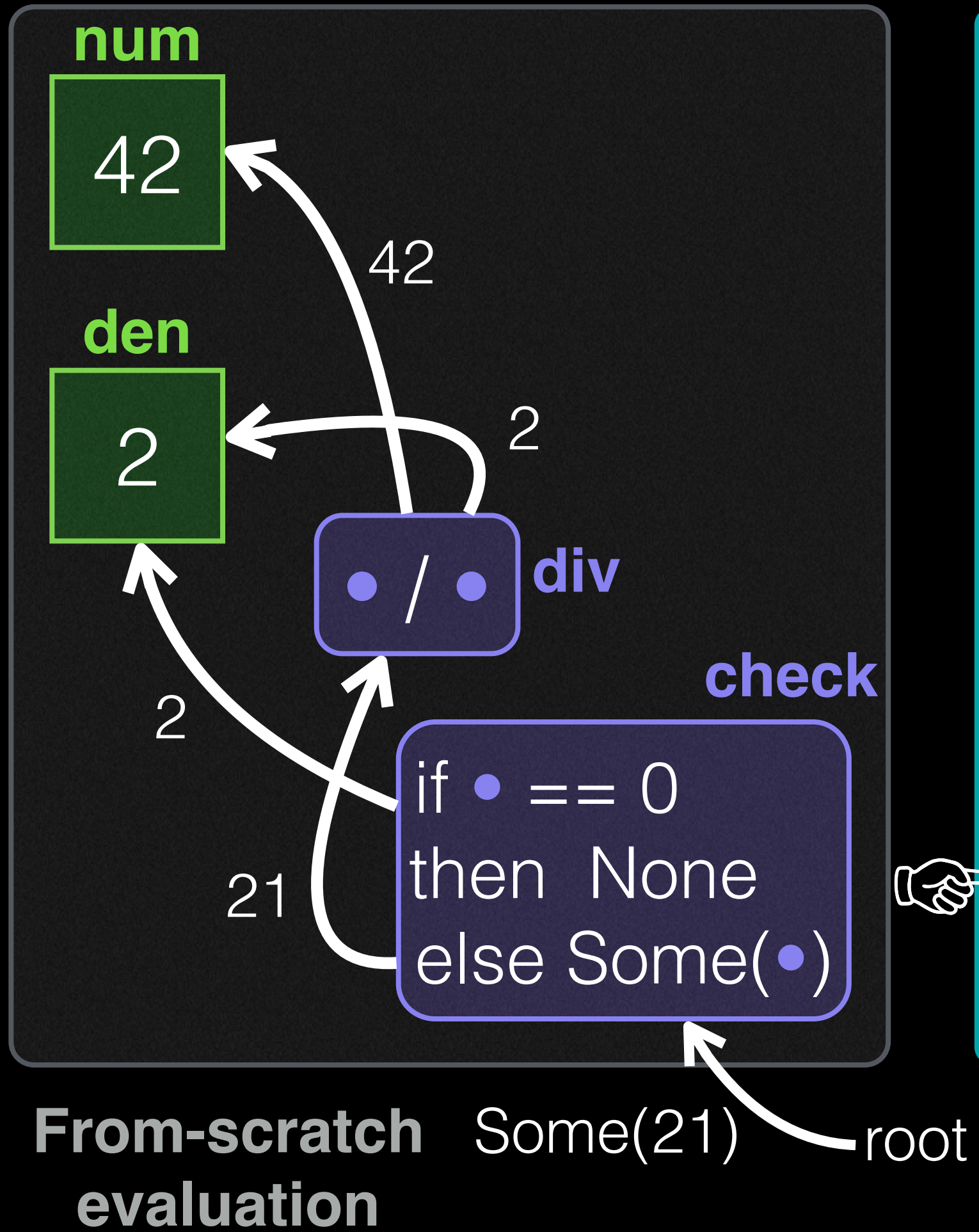
get(check)
set(den, 0); get(check)
set(den, 2); get(check)
```





```
let num = cell(42)
let den = cell(2)
let div = thunk [
  get(num) / get(den)
]
let check = thunk [
  if get(den) == 0
  then None
  else Some(get(div))
]

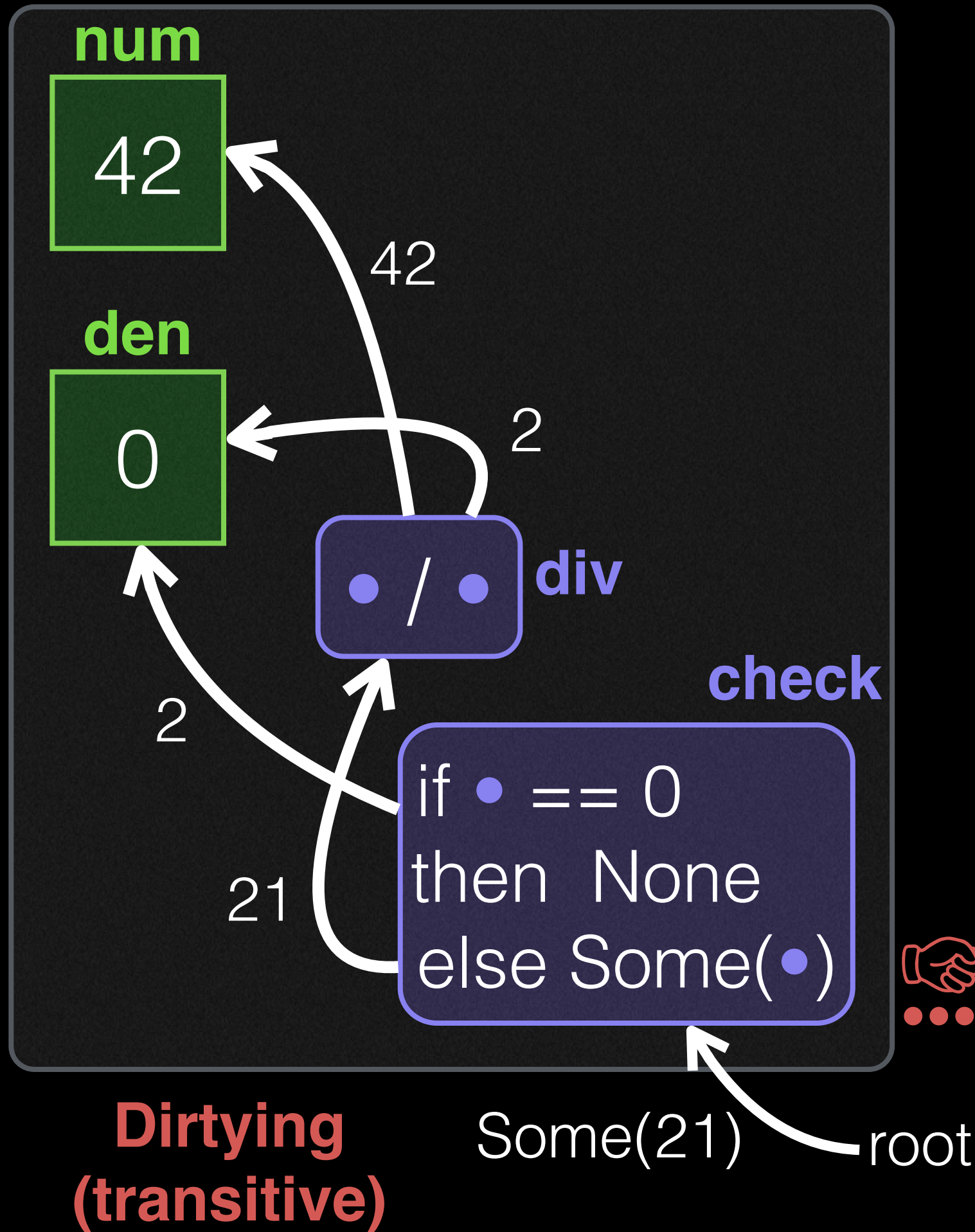
get(check)
set(den, 0); get(check)
set(den, 2); get(check)
```



```
let num = cell(42)
let den = cell(2)
let div = thunk [
  get(num) / get(den)
]
let check = thunk [
  if get(den) == 0
  then None
  else Some(get(div))
]

get(check)
set(den, 0); get(check)
set(den, 2); get(check)
```





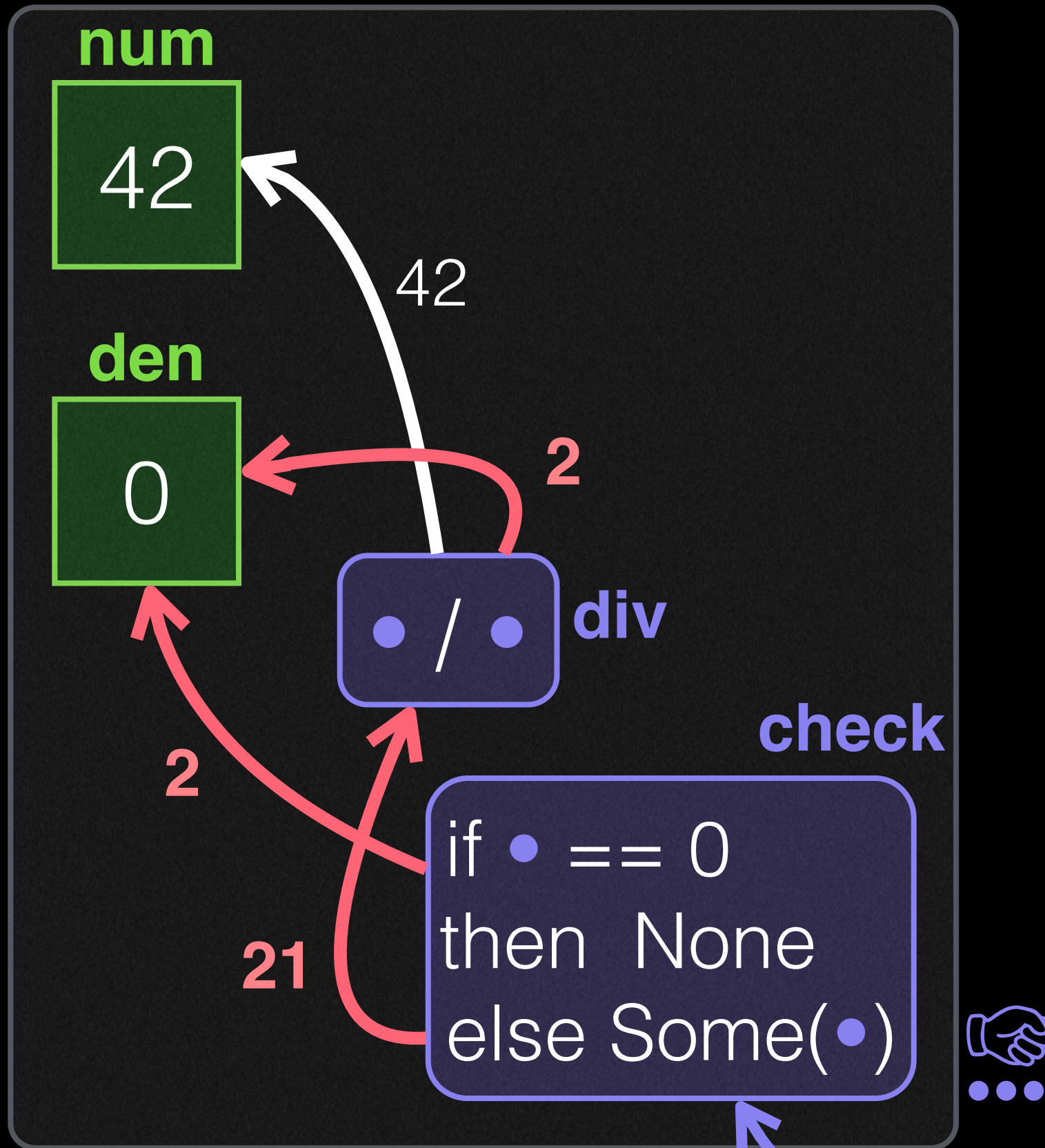
```

let num = cell(42)
let den = cell(2)
let div = thunk [
    get(num) / get(den)
]
let check = thunk [
    if get(den) == 0
    then None
    else Some(get(div))
]

get(check)
set(den, 0); get(check)
set(den, 2); get(check)
  
```





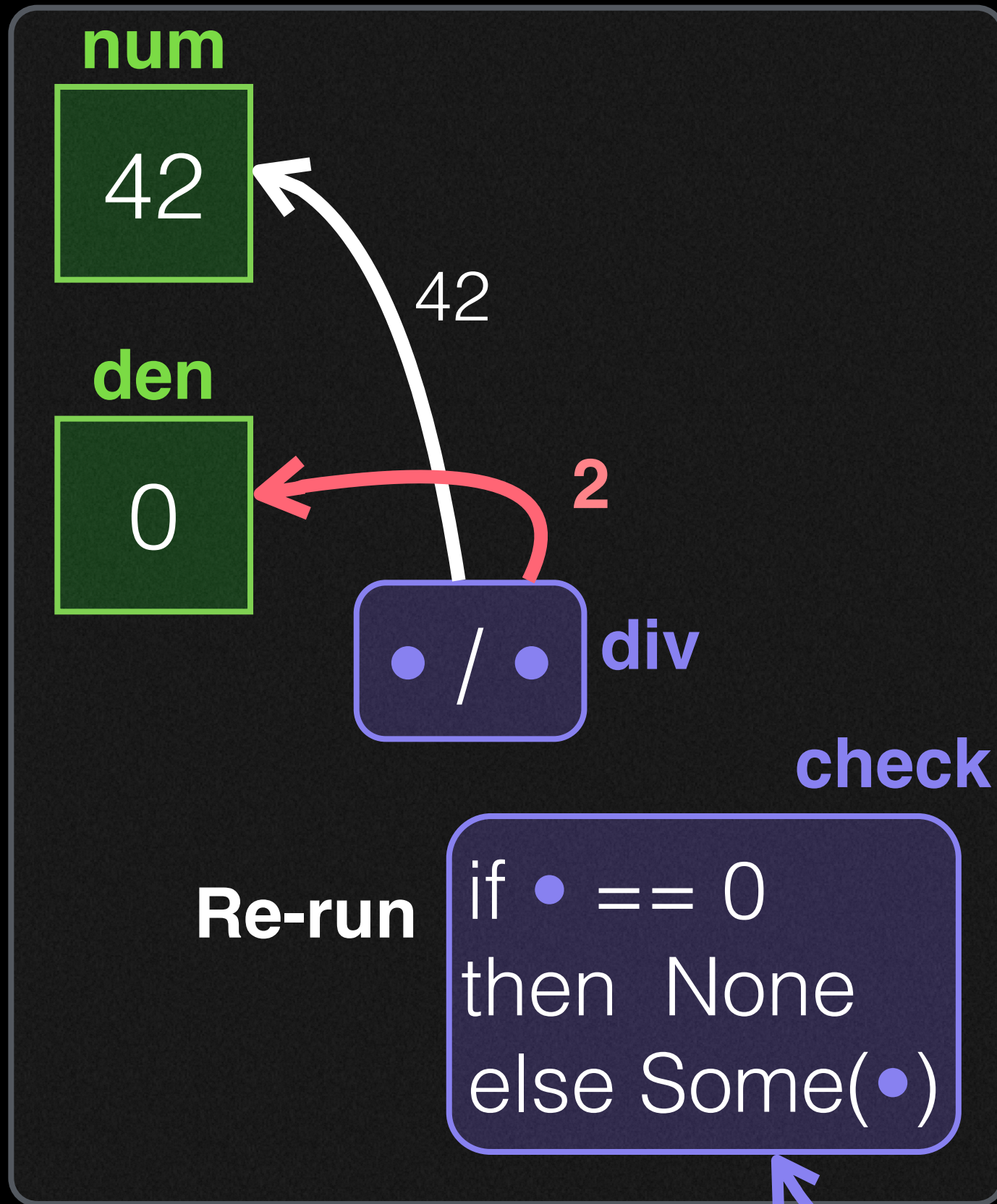


```

let num = cell(42)
let den = cell(2)
let div = thunk [
    get(num) / get(den)
]
let check = thunk [
    if get(den) == 0
    then None
    else Some(get(div))
]

get(check)
set(den, 0); get(check)
set(den, 2); get(check)
  
```

**Change propagation  
(transitive cleaning)**

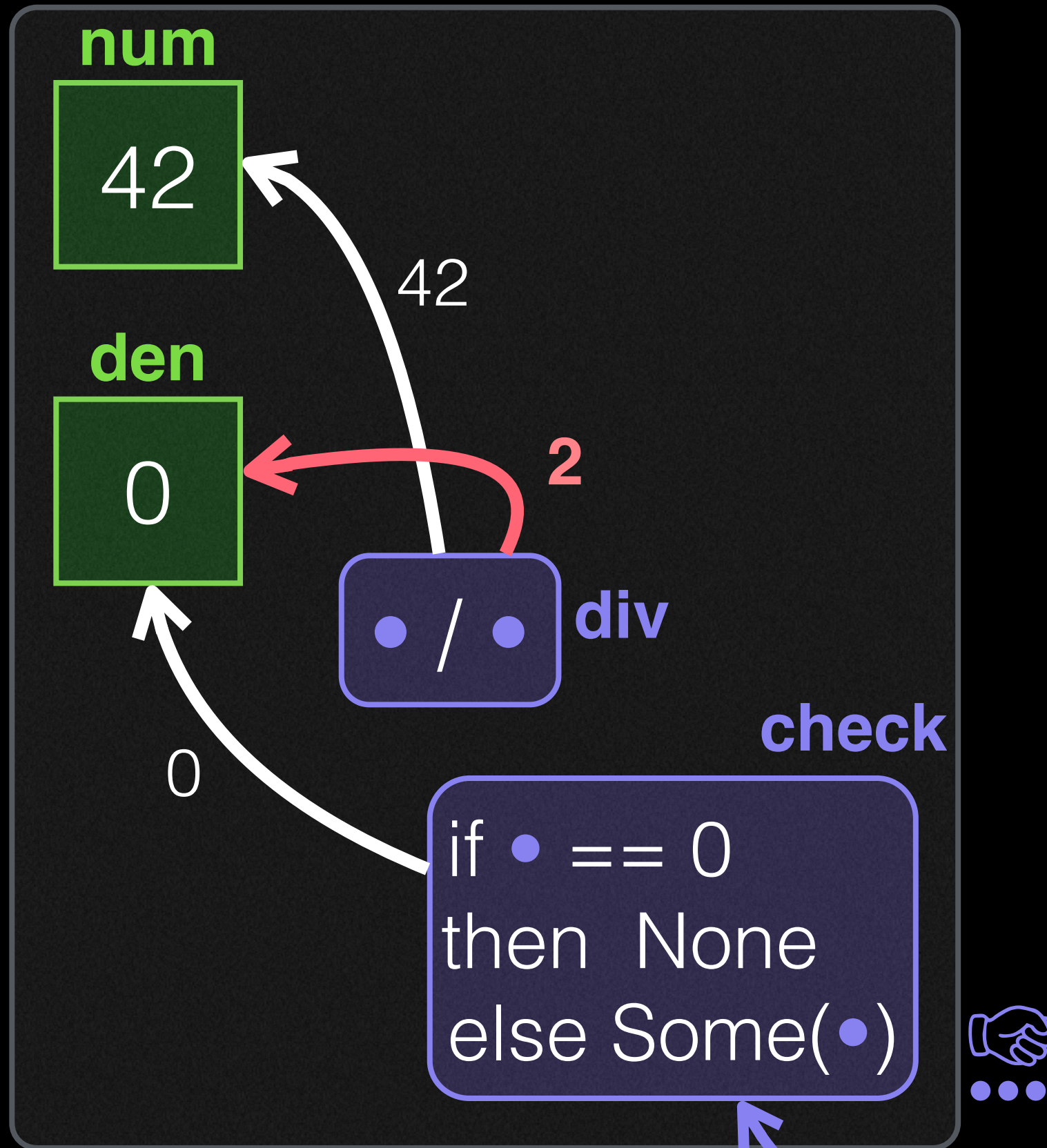


```
let num = cell(42)
let den = cell(2)
let div = thunk [
  get(num) / get(den)
]
let check = thunk [
  if get(den) == 0
  then None
  else Some(get(div))
]

get(check)
set(den, 0); get(check)
set(den, 2); get(check)
```

Change propagation  
(transitive cleaning)



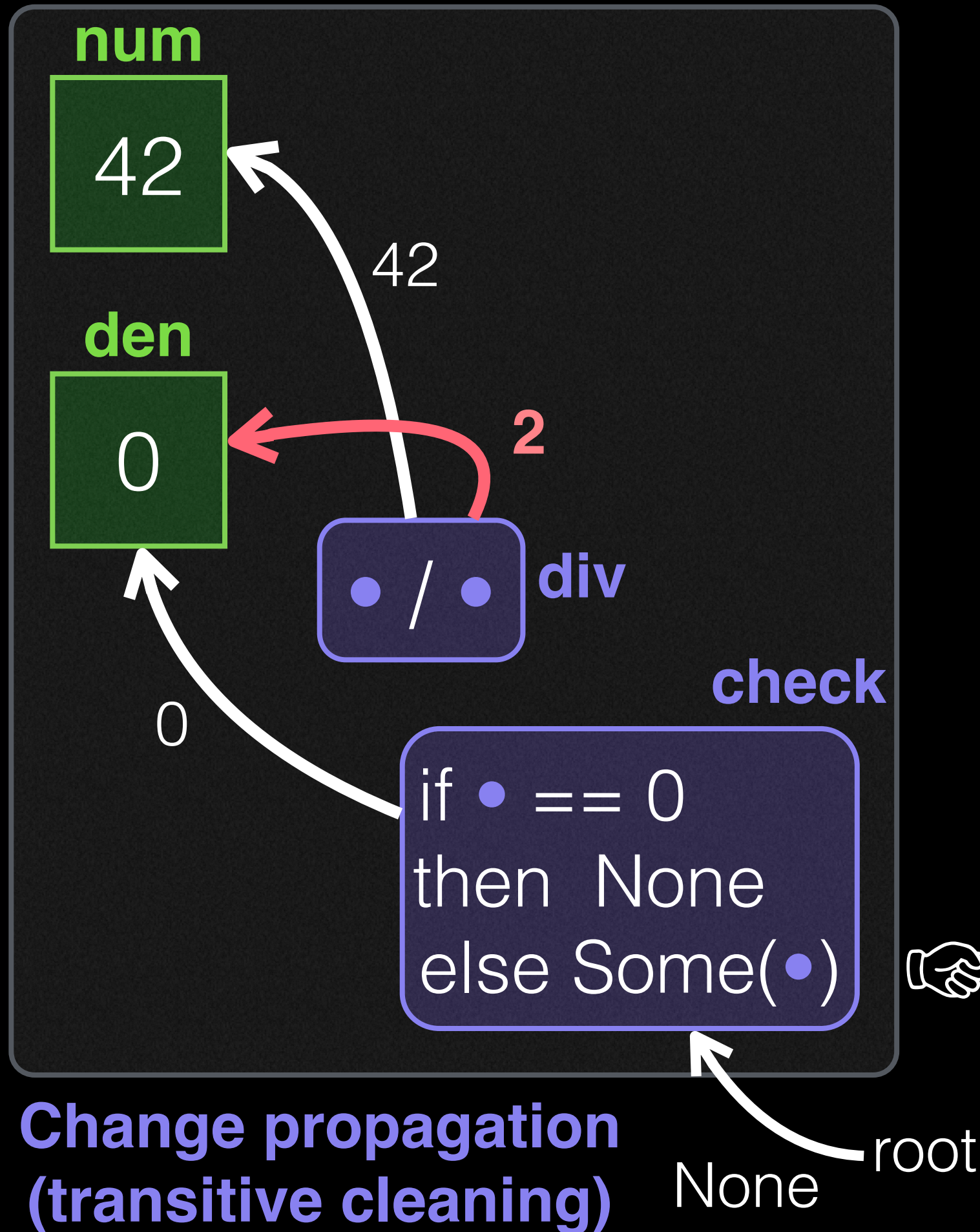


```

let num = cell(42)
let den = cell(2)
let div = thunk [
    get(num) / get(den)
]
let check = thunk [
    if get(den) == 0
    then None
    else Some(get(div))
]

get(check)
set(den, 0); get(check)
set(den, 2); get(check)
  
```

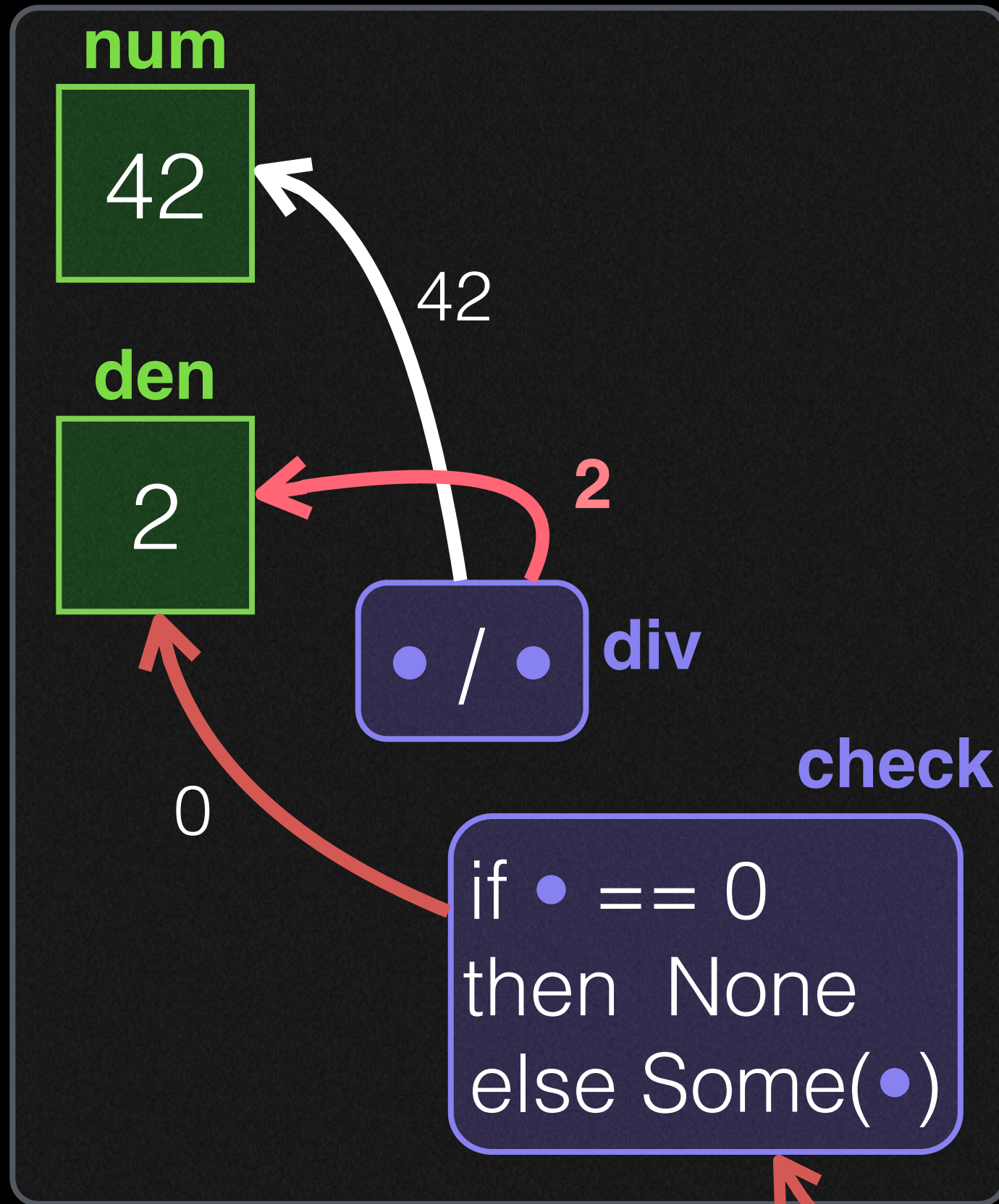
**Change propagation  
(transitive cleaning)**



```
let num = cell(42)
let den = cell(2)
let div = thunk [
  get(num) / get(den)
]
let check = thunk [
  if get(den) == 0
  then None
  else Some(get(div))
]

get(check)
set(den, 0); get(check)
set(den, 2); get(check)
```

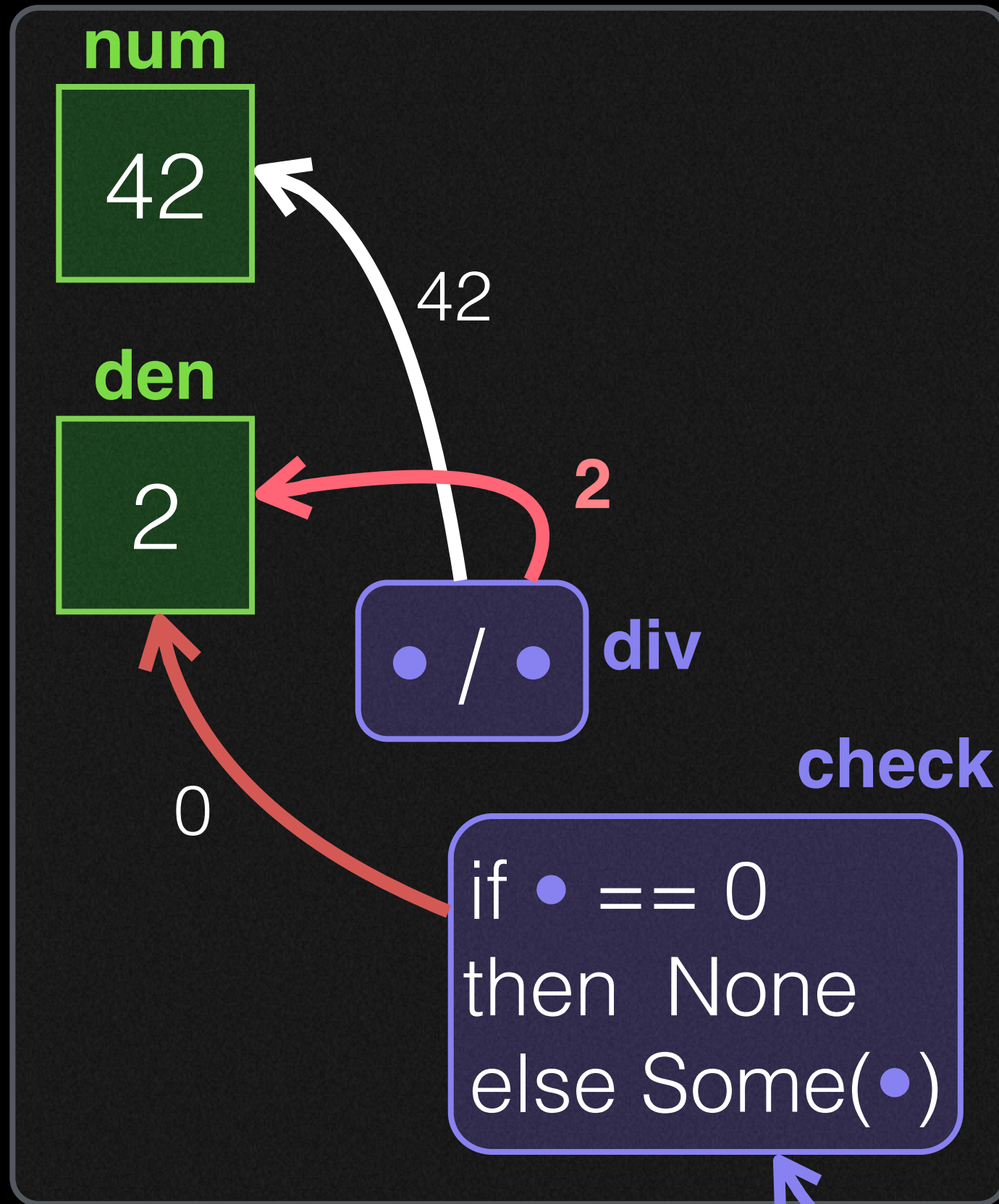




```
let num = cell(42)
let den = cell(2)
let div = thunk [
  get(num) / get(den)
]
let check = thunk [
  if get(den) == 0
  then None
  else Some(get(div))
]

get(check)
set(den, 0); get(check)
set(den, 2); get(check)
```

**Dirtying  
(transitive)**

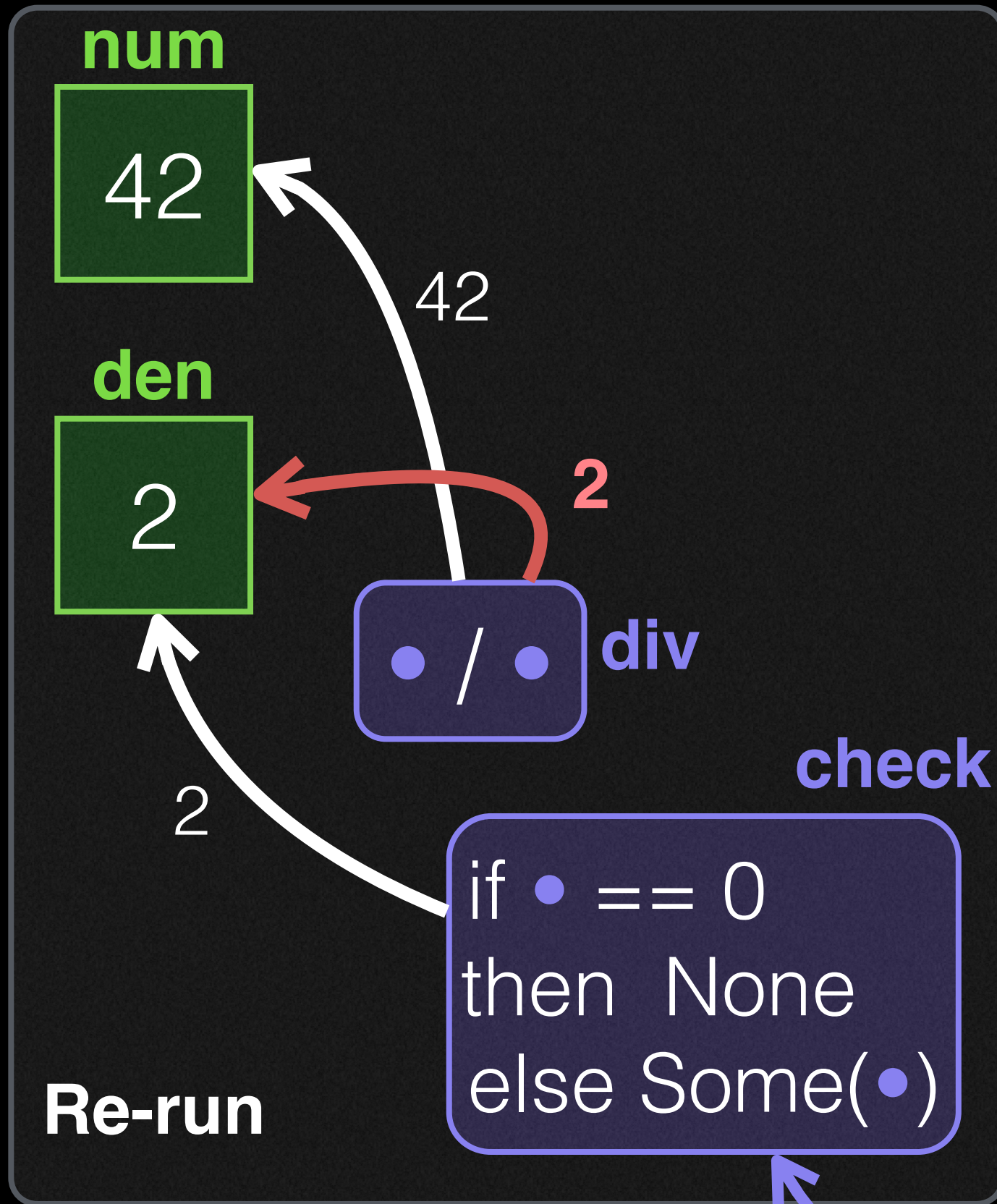


```
let num = cell(42)
let den = cell(2)
let div = thunk [
  get(num) / get(den)
]
let check = thunk [
  if get(den) == 0
  then None
  else Some(get(div))
]

get(check)
set(den, 0); get(check)
set(den, 2); get(check)
```

**Change propagation  
(transitive cleaning)**

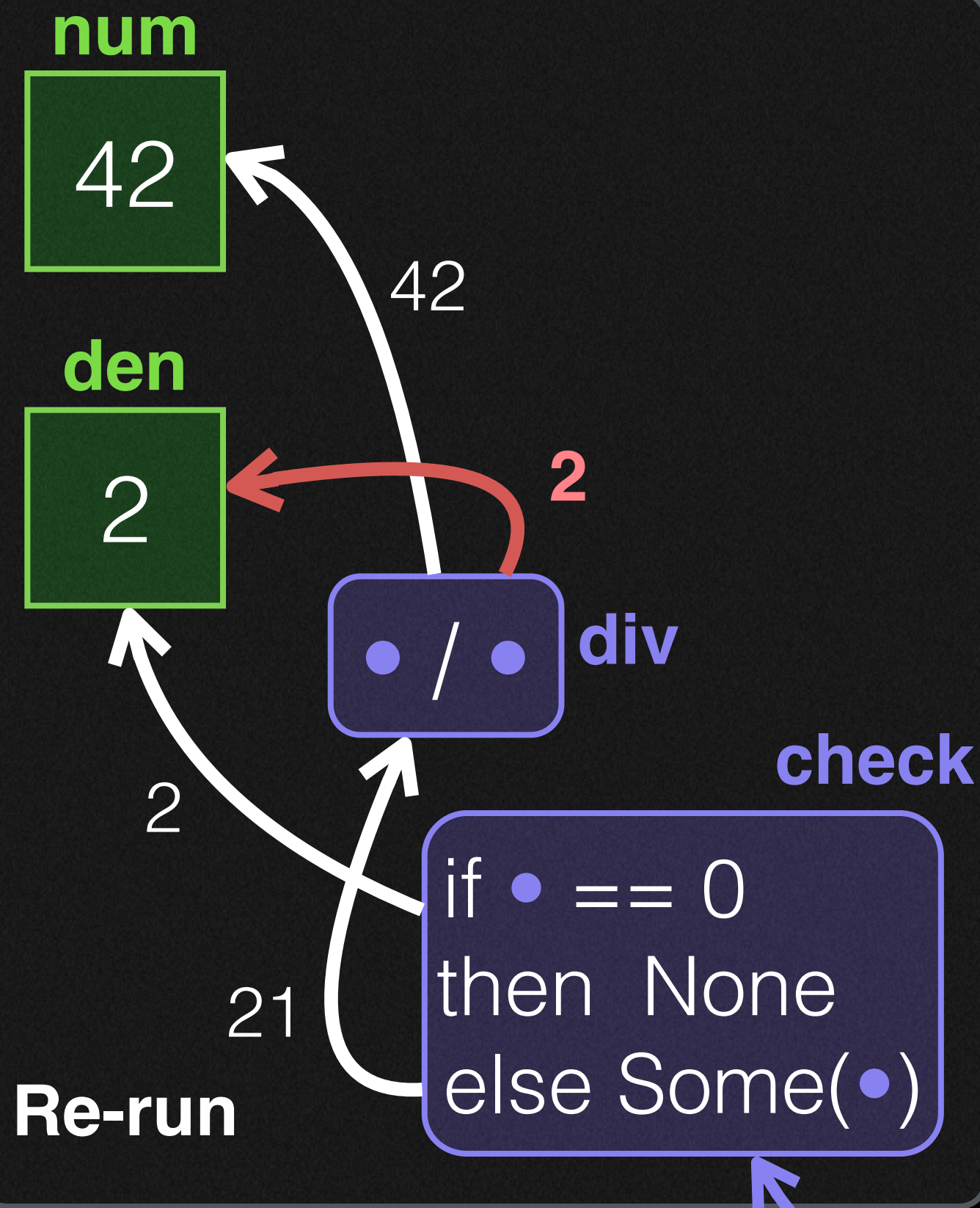




```
let num = cell(42)
let den = cell(2)
let div = thunk [
  get(num) / get(den)
]
let check = thunk [
  if get(den) == 0
  then None
  else Some(get(div))
]

get(check)
set(den, 0); get(check)
set(den, 2); get(check)
```

Change propagation  
(transitive cleaning)

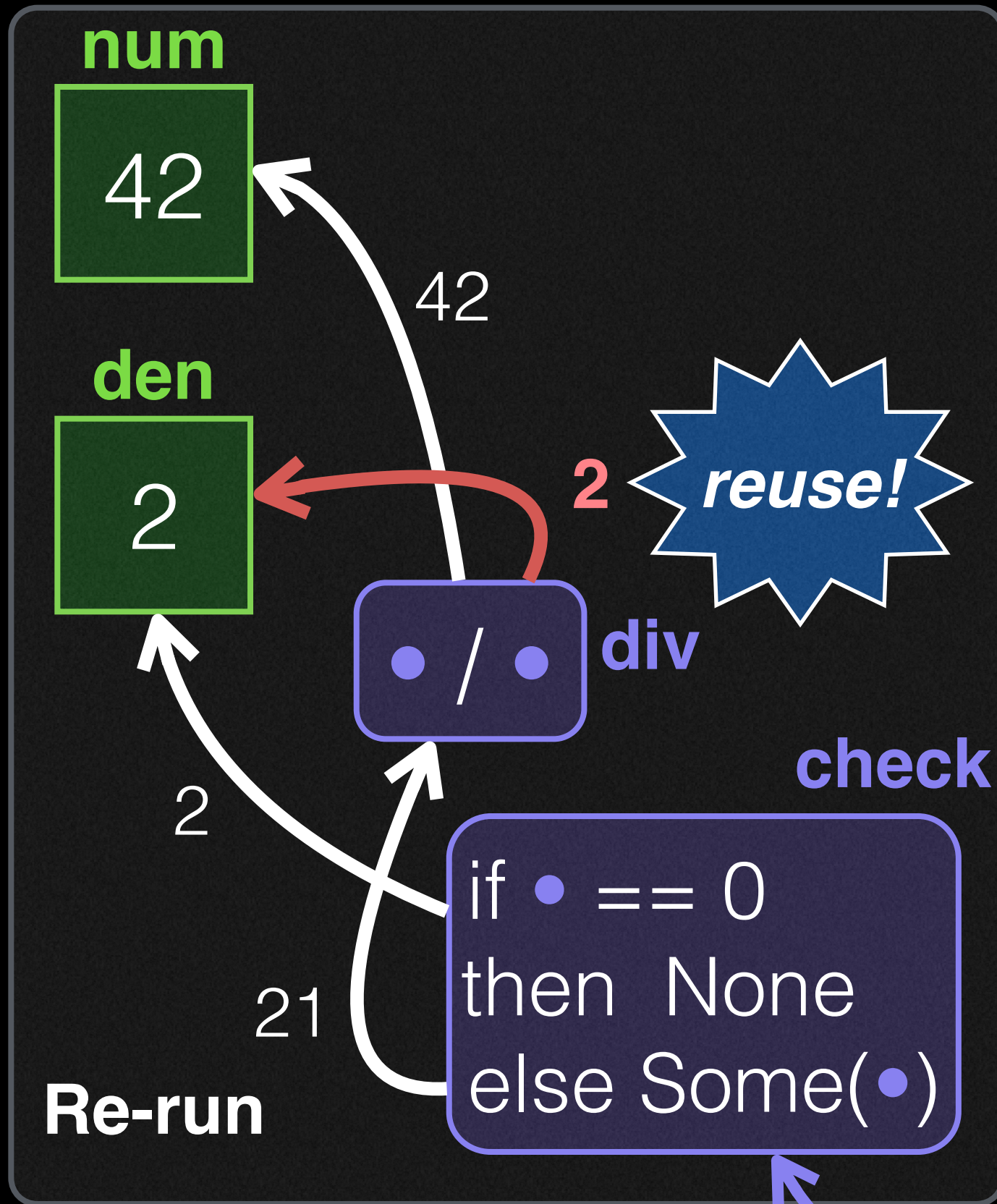


```
let num = cell(42)
let den = cell(2)
let div = thunk [
  get(num) / get(den)
]
let check = thunk [
  if get(den) == 0
  then None
  else Some(get(div))
]

get(check)
set(den, 0); get(check)
set(den, 2); get(check)
```

**Change propagation  
(transitive cleaning)**



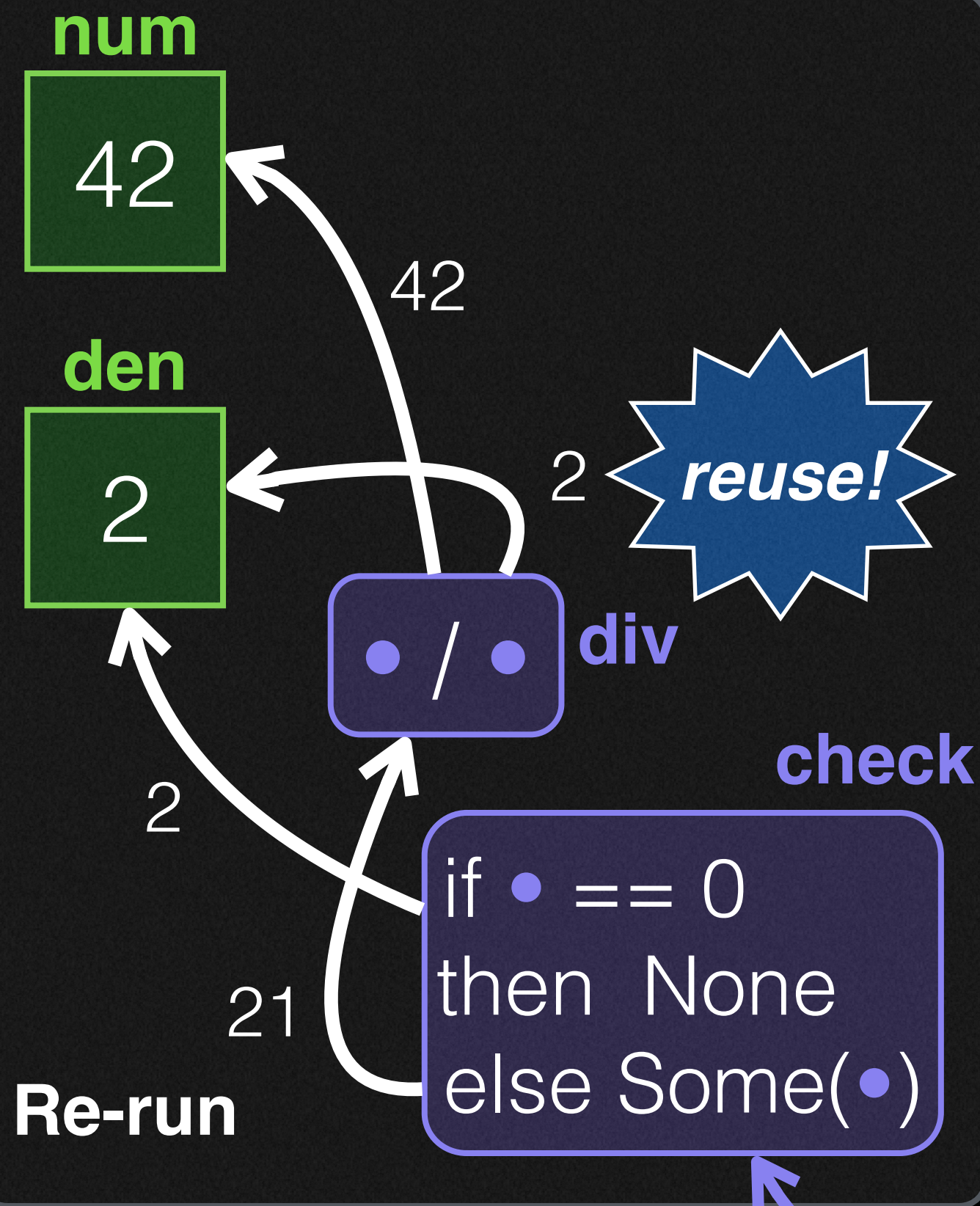


```
let num = cell(42)
let den = cell(2)
let div = thunk [
  get(num) / get(den)
]
let check = thunk [
  if get(den) == 0
  then None
  else Some(get(div))
]

get(check)
set(den, 0); get(check)
set(den, 2); get(check)
```

**Change propagation  
(transitive cleaning)**



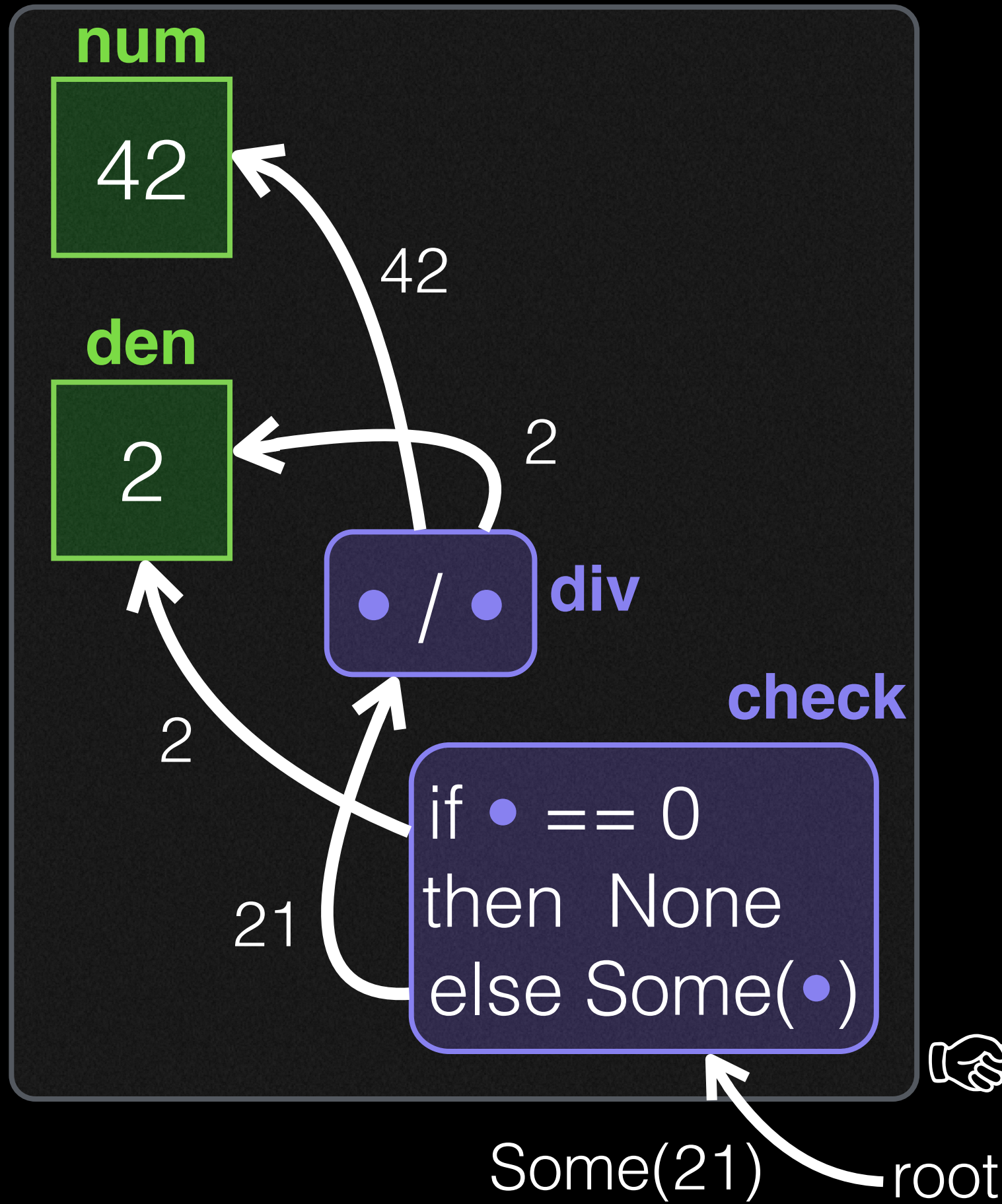


```
let num = cell(42)
let den = cell(2)
let div = thunk [
  get(num) / get(den)
]
let check = thunk [
  if get(den) == 0
  then None
  else Some(get(div))
]

get(check)
set(den, 0); get(check)
set(den, 2); get(check)
```

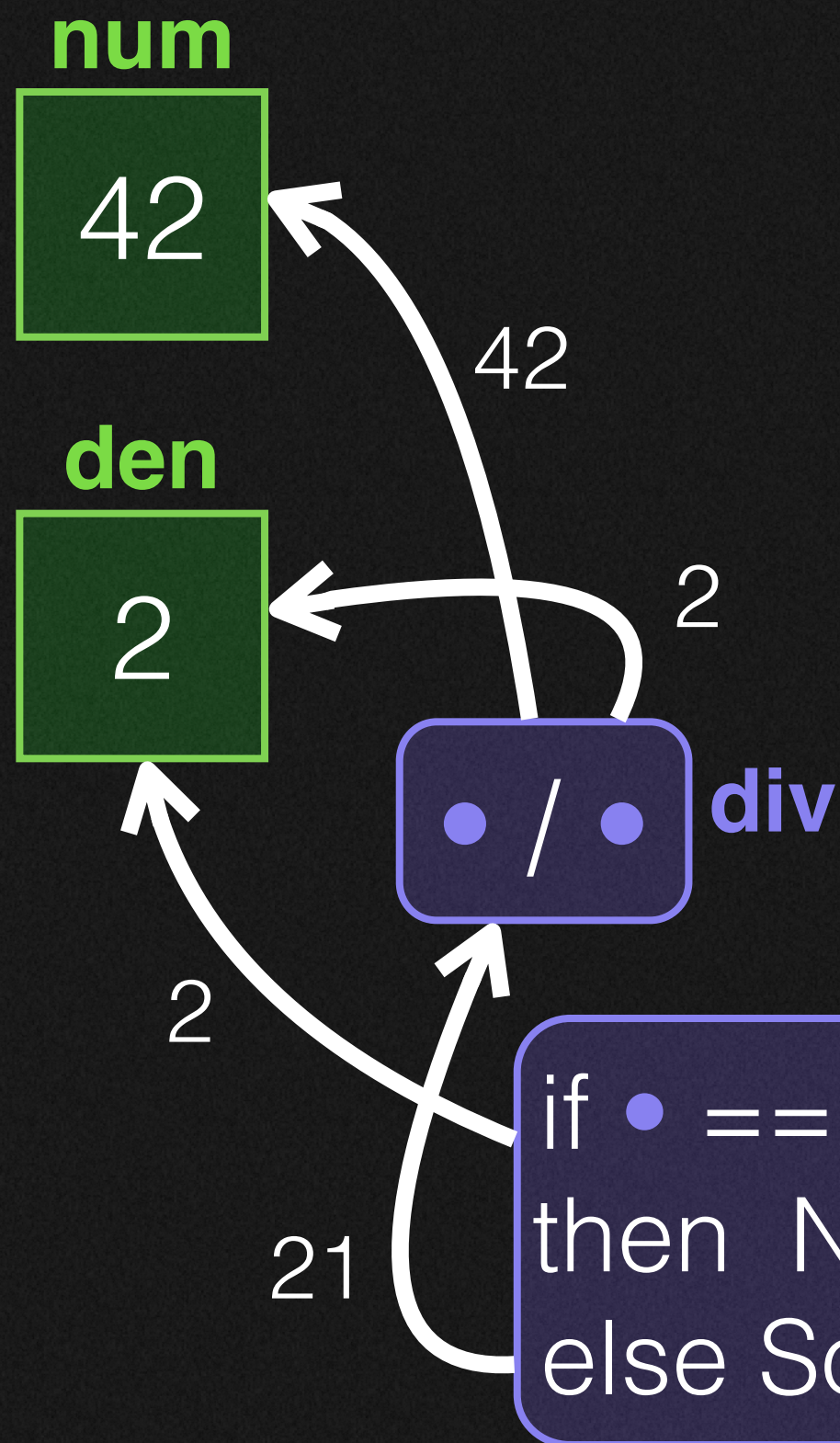
Change propagation  
(transitive cleaning)





```
let num = cell(42)
let den = cell(2)
let div = thunk [
  get(num) / get(den)
]
let check = thunk [
  if get(den) == 0
  then None
  else Some(get(div))
]

get(check)
set(den, 0); get(check)
set(den, 2); get(check)
```



## Summary:

- **dirtying** is eager
- **cleaning** is **demand-driven**  
*c.f.* self-adjusting computation
- **cleaning** is **consistent / sound**  
("from-scratch consistent", FSC)

*c.f.* "height-based" change prop  
<https://gist.github.com/khooyip/98abc0e64dc296deaa48>

Some(21)  $\xrightarrow{\text{root}}$