Balancing the Books

Lisa Lippincott

C++now, April 2025

If your subprogram follows certain rules, it gets a gold star!

If your subprogram follows certain rules, it gets a gold star!

If your subprogram follows certain rules, it gets a gold star!

Const correctness in C++

RAII resource management in C++

Borrow/freeze correctness in Rust

If your subprogram follows certain rules, it gets a gold star!

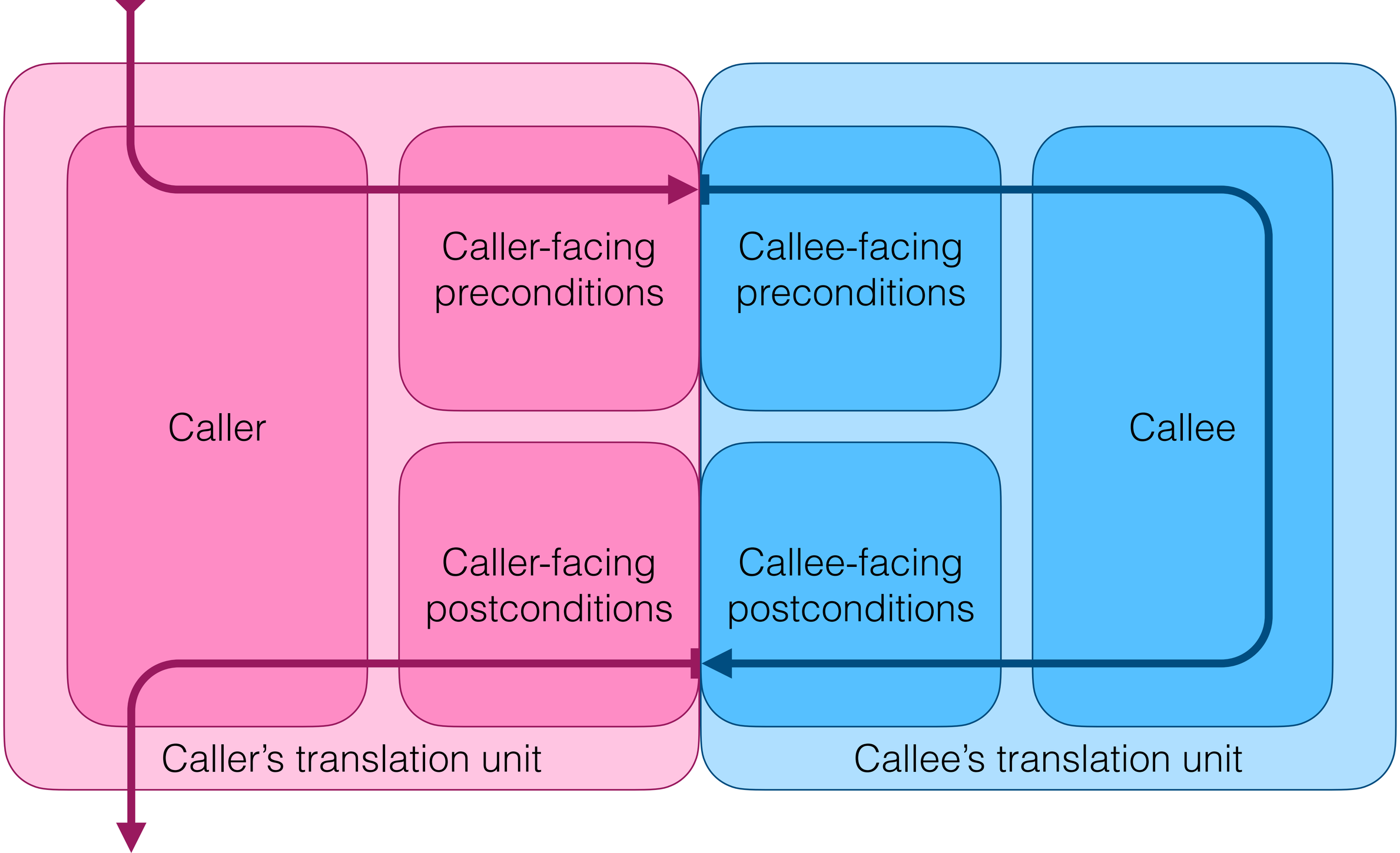
If your subprogram follows certain rules, it gets a gold star!

Rust-style borrow/freeze correctness copied into C++

The contracts facility in C++26 does not provide Rust-style borrow/freeze correctness.

But could it in a future version of C++?

- Users choose between efficient trust and redundant verification.
- The choice is made outside the source code.
- The choice is granular, at translation unit scale or smaller.
- The choices made for different translation units are independent.
- At boundaries, either side can insist on verification.


Caller
Caller-facing preconditions
Caller-facing postconditions
Callee-facing preconditions
Callee-facing postconditions
Callee
Caller's translation unit
Callee's translation unit

- Assertions are verified by inspecting runtime data.
- The verification is hidden from the code being verified.

For contract-like verification that goes beyond the values of objects, we need runtime data that is:

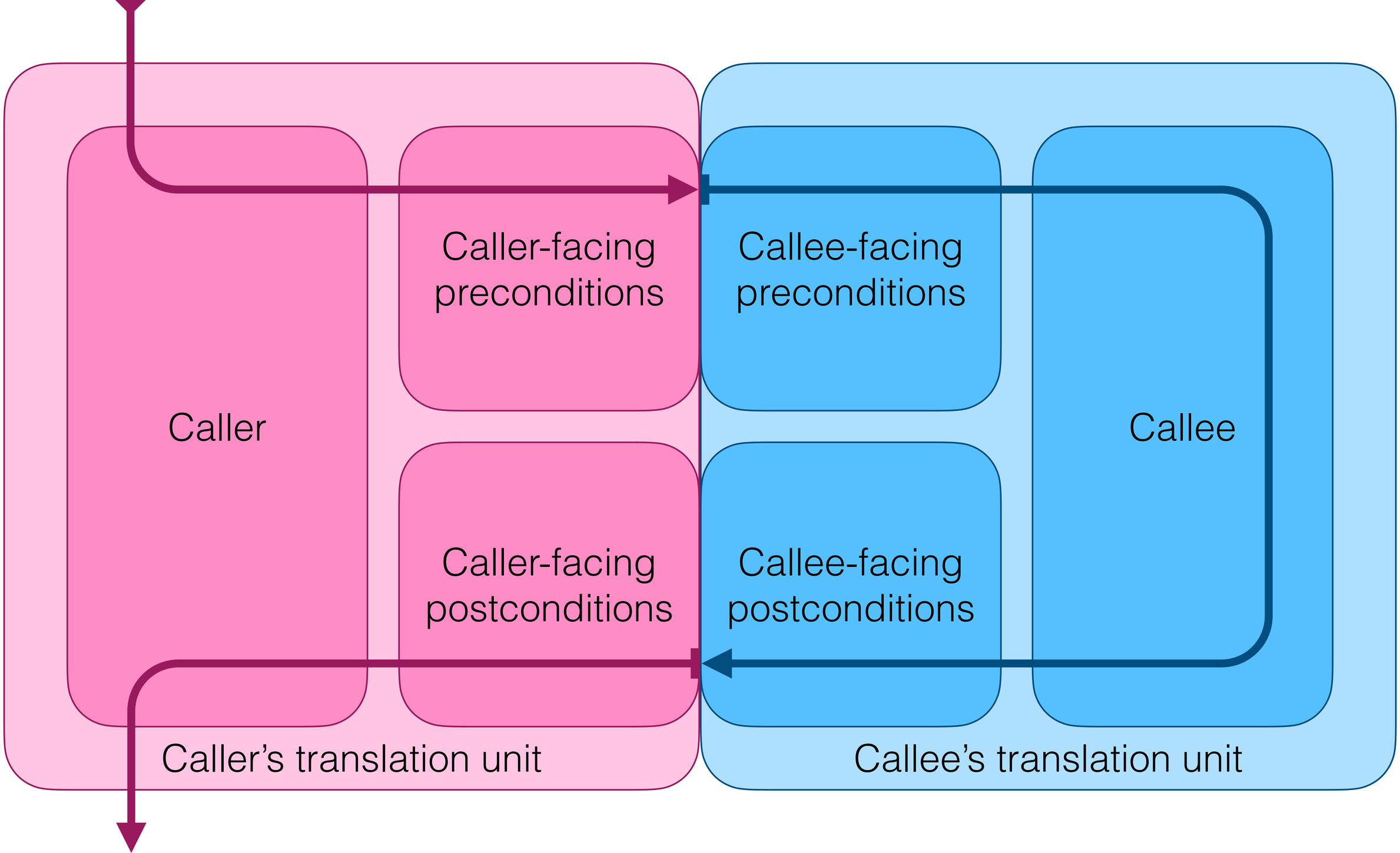
— hidden from the code in our translation unit

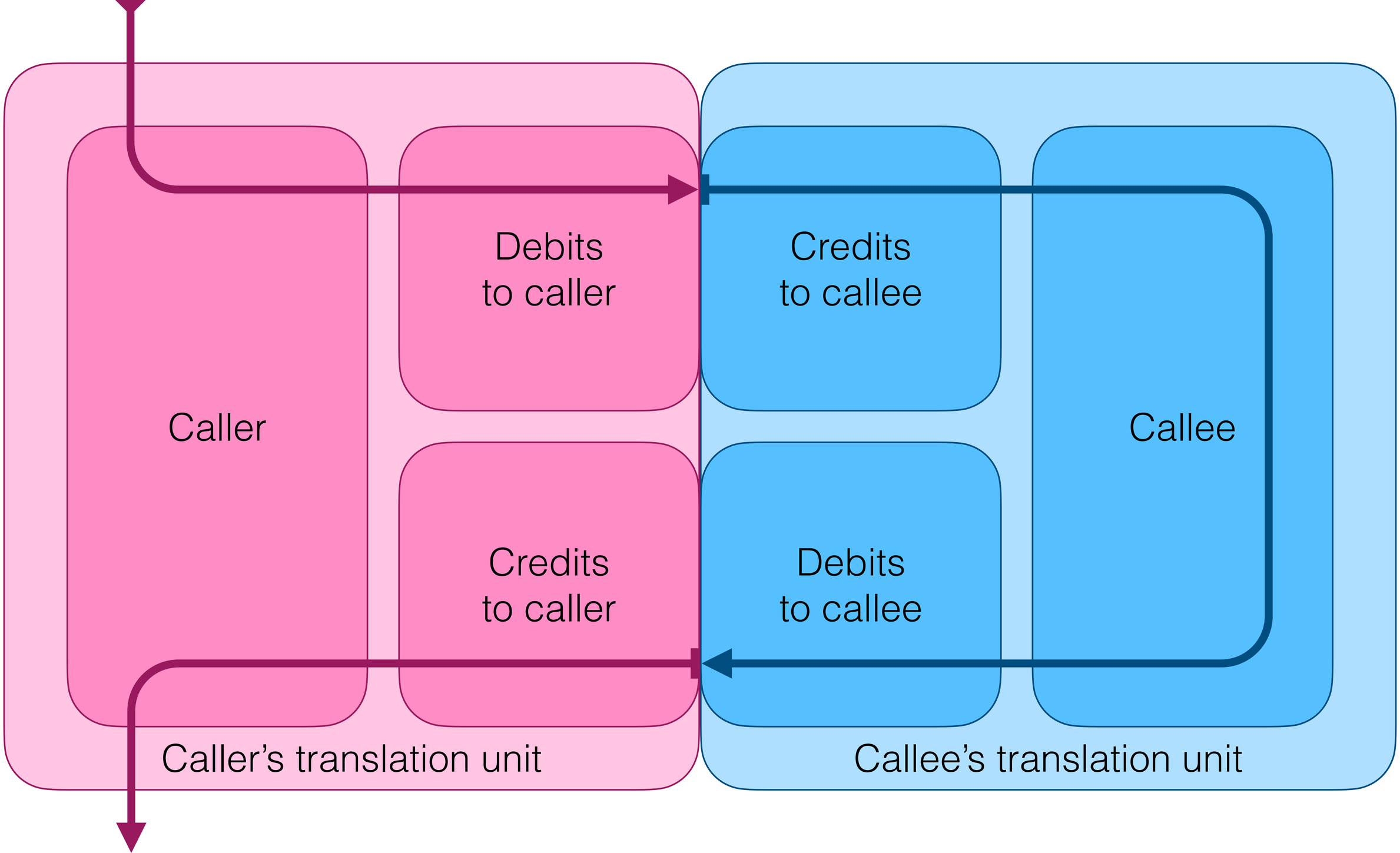
— not reliant on other translation units

— but can still affect the outcome of verification.

Double-entry bookkeeping

Inventor unknown. Earliest known use 1299.


Caller
Caller-facing preconditions
Caller-facing postconditions
Callee-facing preconditions
Callee-facing postconditions
Callee
Caller's translation unit
Callee's translation unit


Caller
Debits
to caller
Credits
to caller
Credits
to callee
Debits
to callee
Callee
Caller's translation unit
Callee's translation unit

I'd like a variable with the value five, please.

OK, I made a variable. It's value is five.

Can I have some assurance that the value will continue to be five in the future?

I promise it won't change. I give you this token as a symbol of my vow.

What if I grow tired of five, and want a different value?

Return the token to me. I'll change things, and then I'll give you a new token.

Wesley Newcomb Hohfeld (1879-1918)

Some Fundamental Legal Conceptions
as Applied in Judicial Reasoning

Yale Law Journal, 1913

Duty

Right-duty relationship

Right

Hohfeldian right

- Prevents change
- Not duplicable
- Transferrable

Hohfeldian right

- Prevents change
- Not duplicable
- Transferrable
- Remitted intact to request change

Tell me the value of my variable, please.

Did I say it had a stable value?

Oh yes, you gave me a token as symbol.

Show me, please.

But if I give you back the token you gave me, you could change the value.

I see. Bit of a conundrum, that.

Tell me the value of my variable, please. 🙂

🤖 Did I say it had a stable value?

Oh yes, you gave me a token as symbol. 🙂

🤖 Show me, please.

But if I give you back the token you gave me, you could change the value. 🙂

🤖 I see. Bit of a conundrum, that.

I'll make a new token for you out of a little piece of the one you gave me. 🙂 

🤖 Did I say it had a stable value?

Oh yes, you gave me a token as symbol. 🙂

🤖 Show me, please.

But if I give you back the token you gave me, you could change the value. 🙂

🤖 I see. Bit of a conundrum, that.

I’ll make a new token for you out of a little piece of the one you gave me. 🙂 

🟡 🤖 Excellent. Here’s your value, and here’s your token back.

Oh yes, you gave me a token as symbol.

Show me, please.

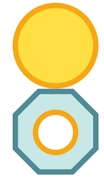
But if I give you back the token you gave me, you could change the value.

I see. Bit of a conundrum, that.

I’ll make a new token for you out of a little piece of the one you gave me.

Excellent. Here’s your value, and here’s your token back.

Thanks! I'll fill in the hole now.



Wesley Newcomb Hohfeld (1879-1918)

Some Fundamental Legal Conceptions as Applied in Judicial Reasoning

Yale Law Journal, 1913

Immunity — **Immunity-disability relationship** — **Disability**

Hohfeldian immunity

- Prevents change
- Not duplicable
- Transferrable only to issuer

Hohfeldian immunity

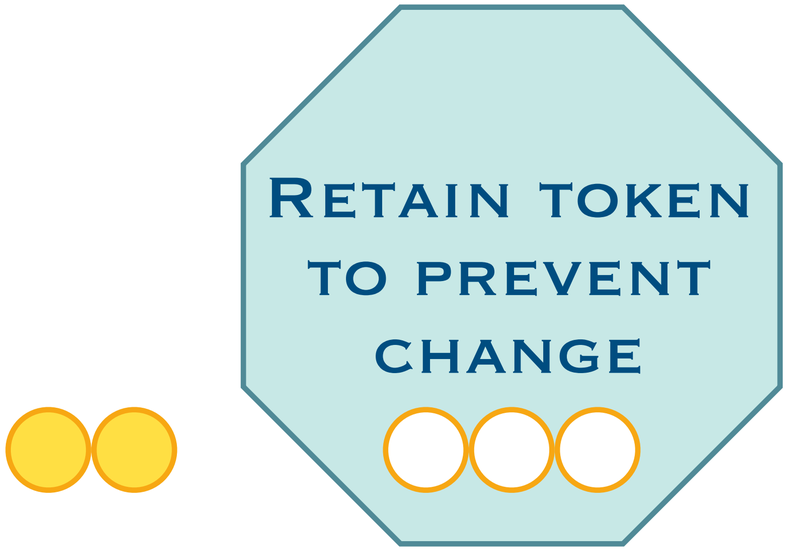
- Prevents change
- Not duplicable
- Transferrable only to issuer
- Not valid for change requests

Hohfeldian right

- Prevents change
- Not duplicable
- Transferrable
- Remitted intact to request change

Hohfeldian immunity

- Prevents change
- Not duplicable
- Transferrable only to issuer
- Not valid for change requests



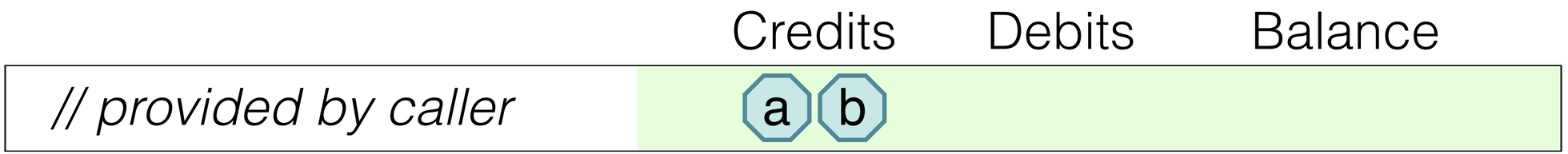
Hohfeldian right

- Prevents change
- Not duplicable
- Transferrable
- Remitted intact to request change
- May issue immunities by taking on disabilities

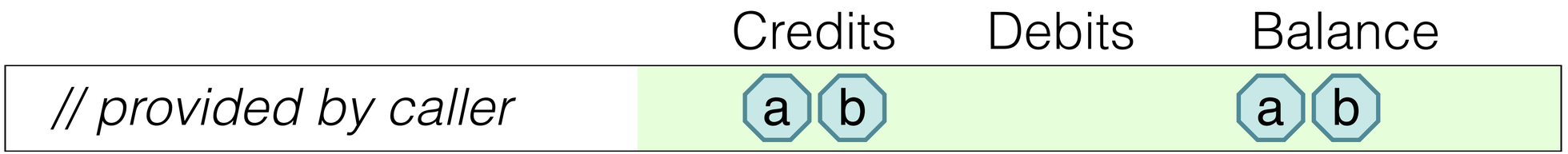
Hohfeldian immunity

- Prevents change
- Not duplicable
- Transferrable only to issuer
- Not valid for change requests
- May issue further immunities by taking on disabilities

```
void swap( int& a, int& b )
   {
    int c;
    c = a;
    a = b;
    b = c;
   }
```




```
int c;
c = a;
a = b;
b = c;
```



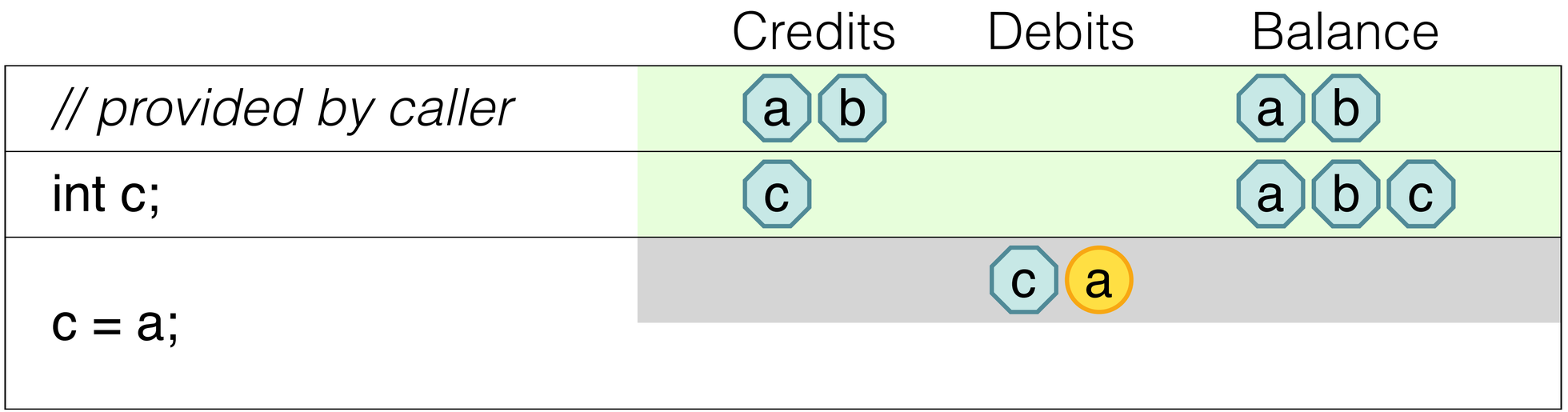

```
int c;
c = a;
a = b;
b = c;
```

	Credits	Debits	Balance
// provided by caller	a b		a b
int c;	c		

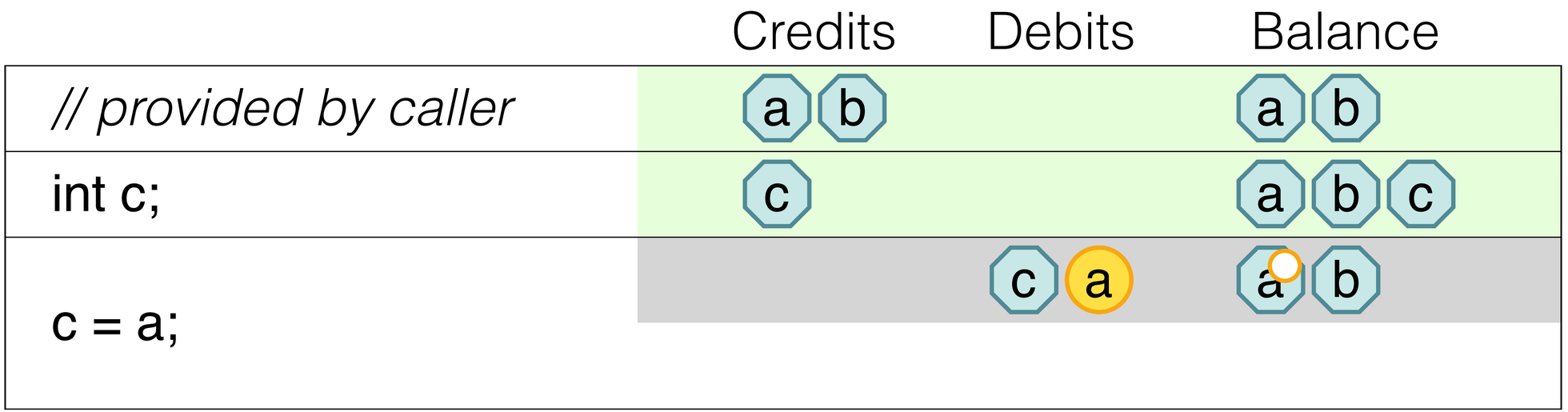
c = a;
a = b;
b = c;

	Credits	Debits	Balance
// provided by caller	a b		a b
int c;	c		a b c

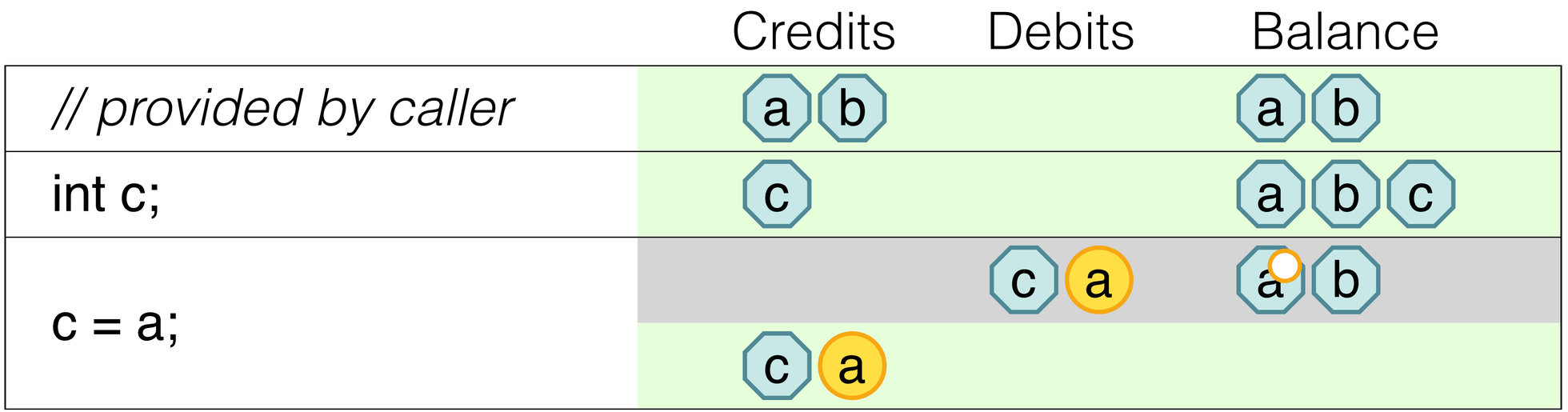
c = a;
a = b;
b = c;


Credits
Debits
Balance
// provided by caller
a
b
a
b
int c;
c
a
b
c
c = a;
c
a

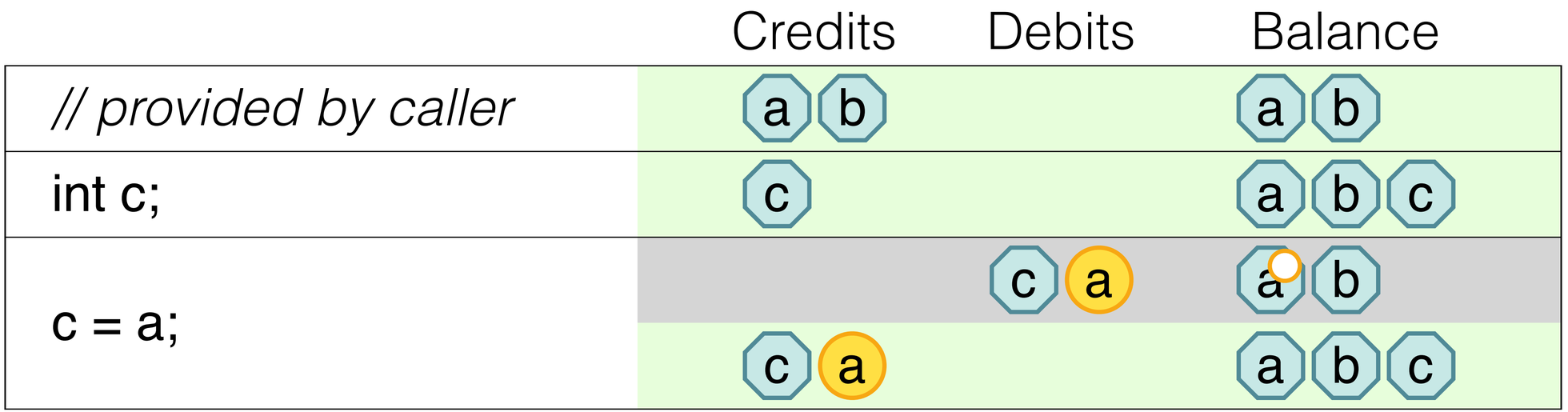
a = b;
b = c;


Credits
Debits
Balance
// provided by caller
a
b
a
b
int c;
c
a
b
c
c = a;
c
a
a
b

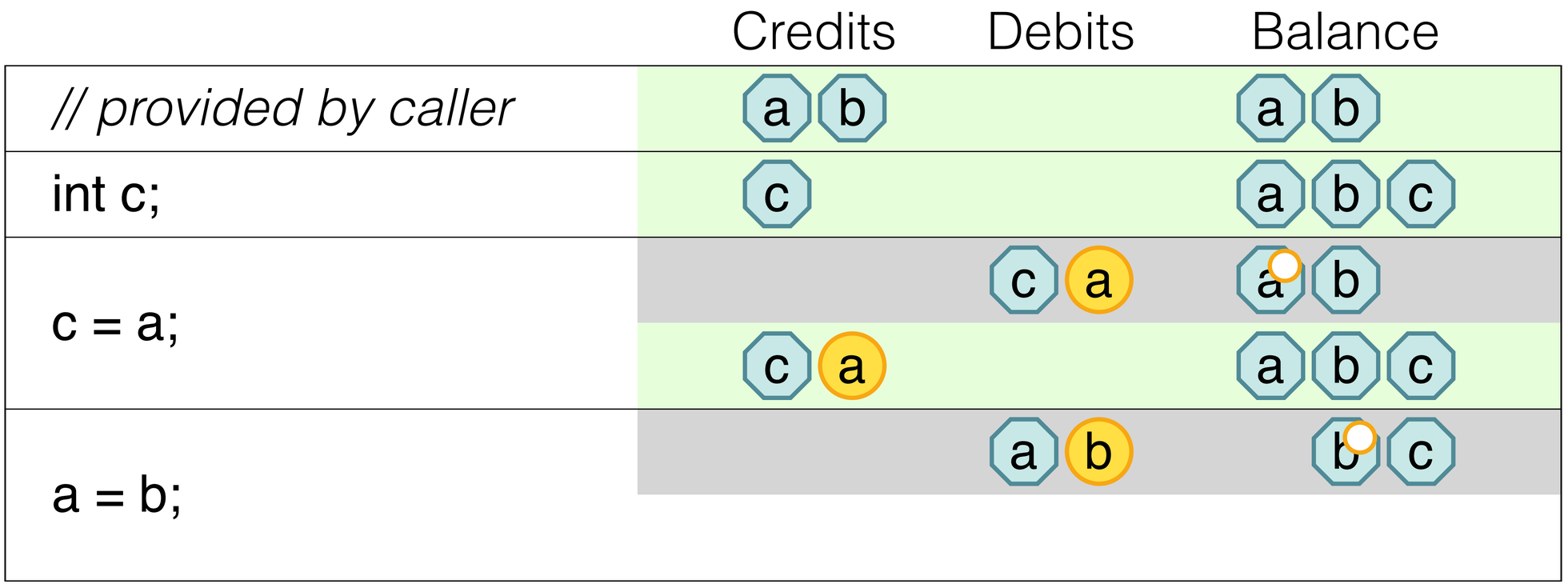
a = b;
b = c;


Credits
Debits
Balance
// provided by caller
a
b
a
b
int c;
c
a
b
c
c = a;
c
a
a
b
c
a

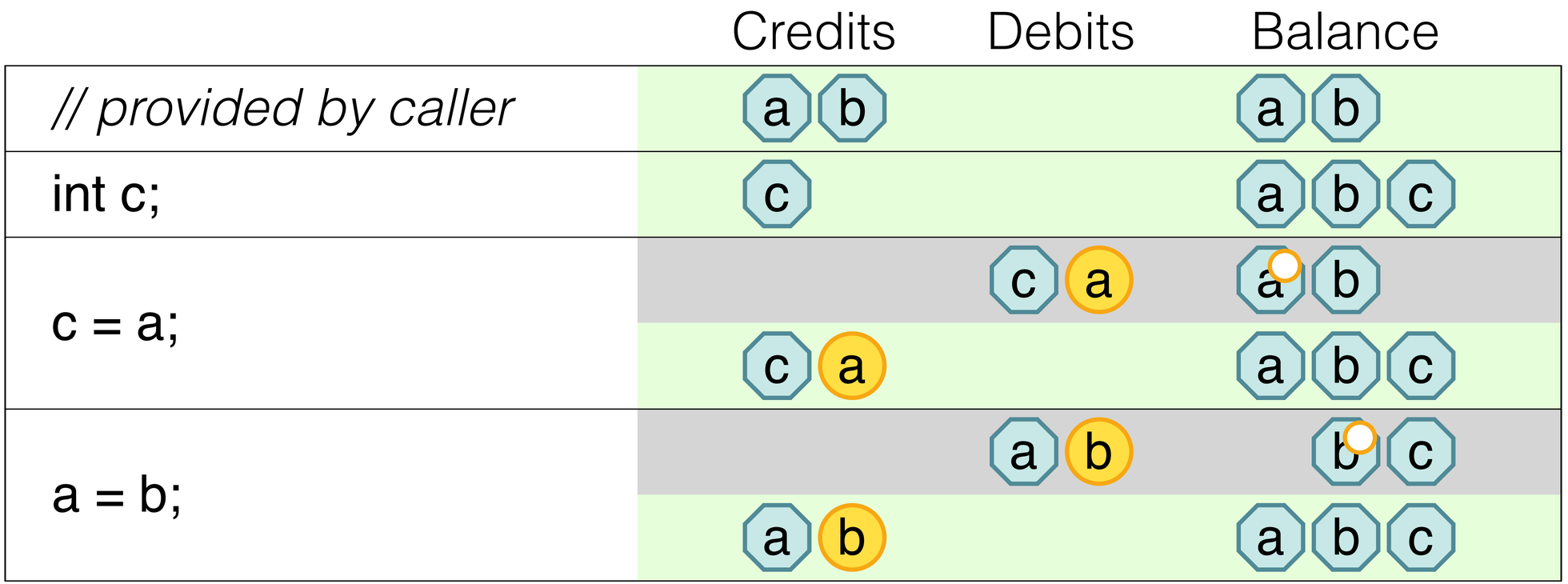
a = b;
b = c;

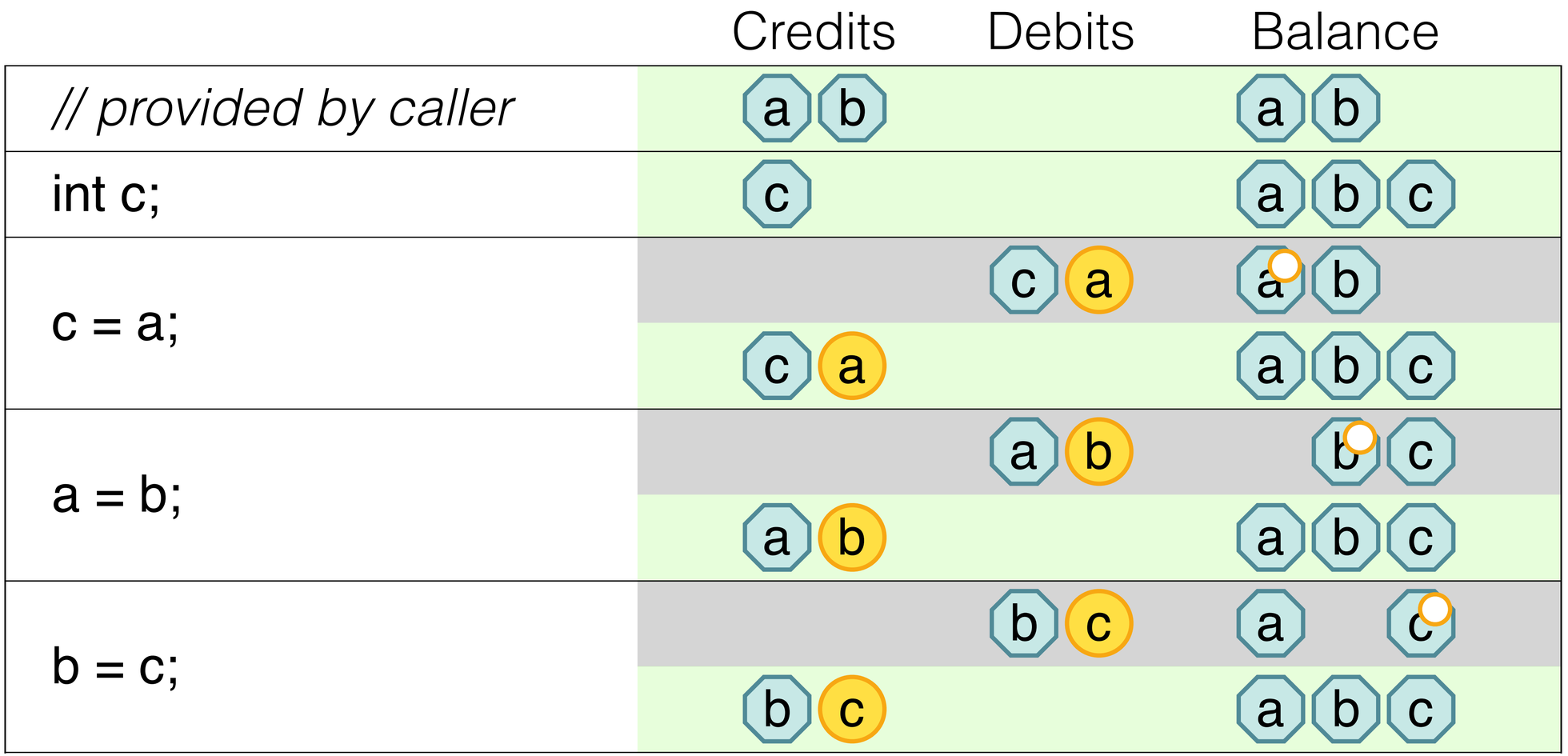

Credits
Debits
Balance
// provided by caller
a b
a b
int c;
c
a b c
c = a;
c a
a b
c a
a b c

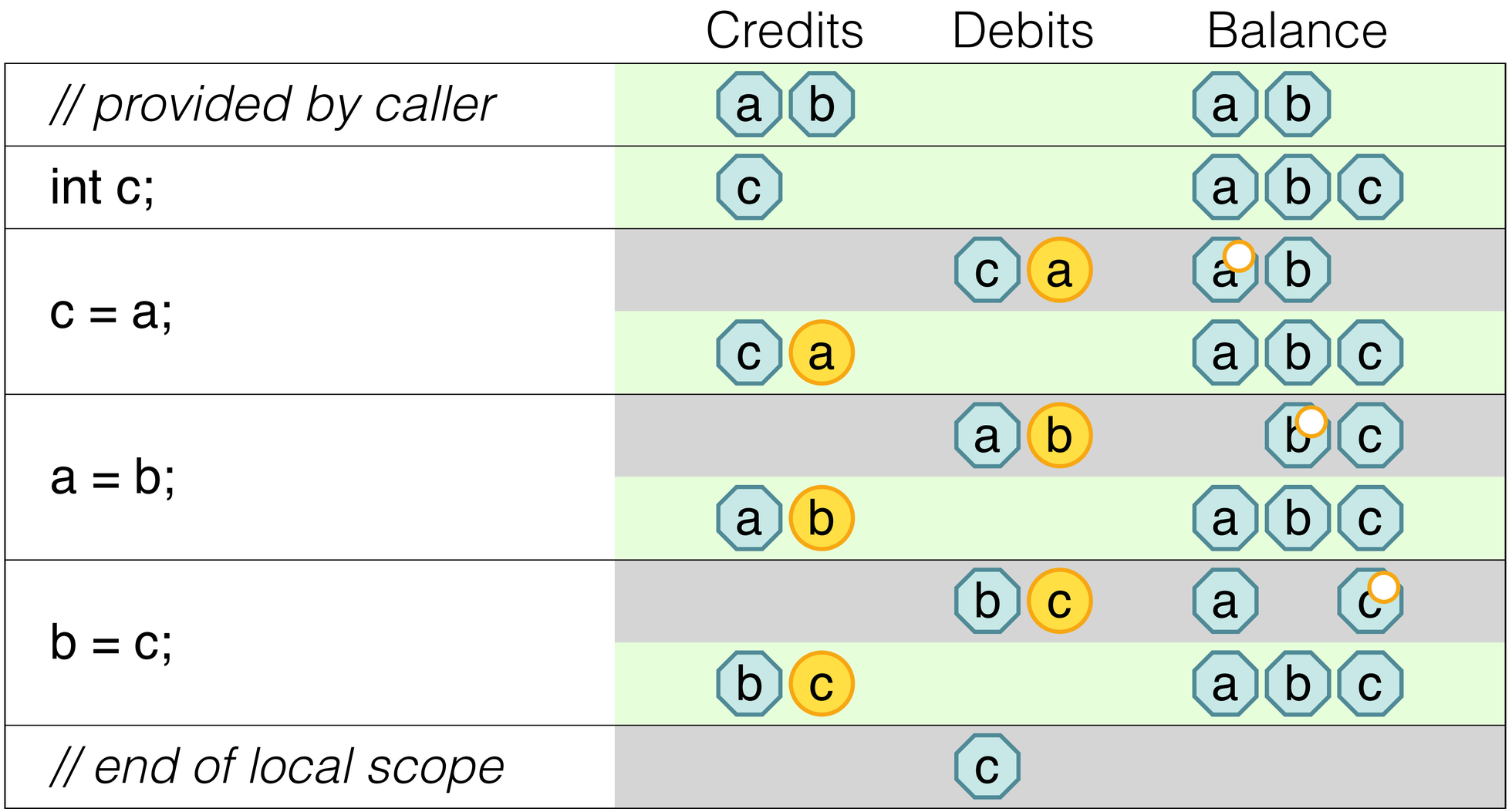
a = b;
b = c;

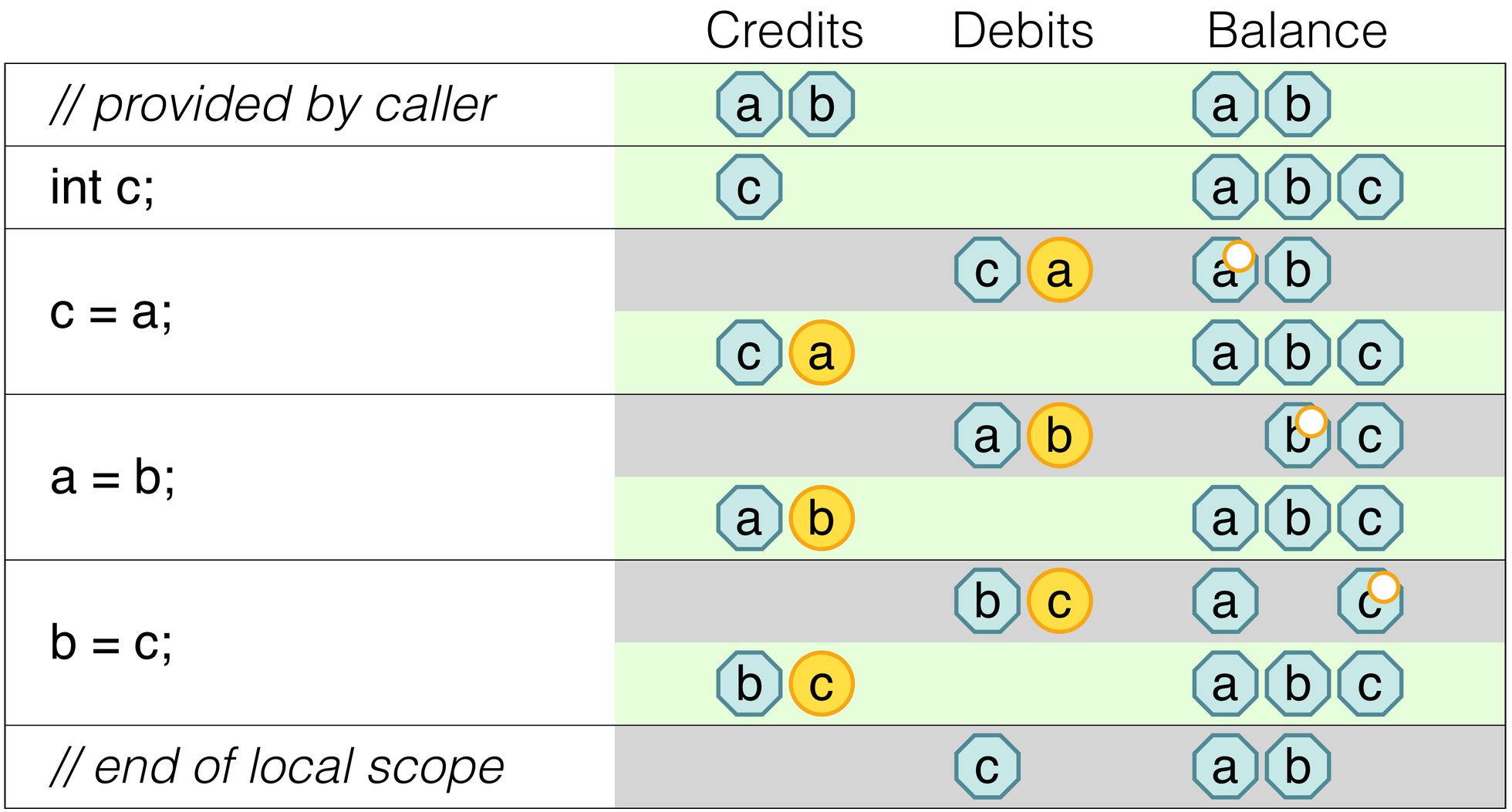

Credits
Debits
Balance
// provided by caller
a b
a b
int c;
c
a b c
c = a;
c a
a b
c a
a b c
a = b;
a b
b c

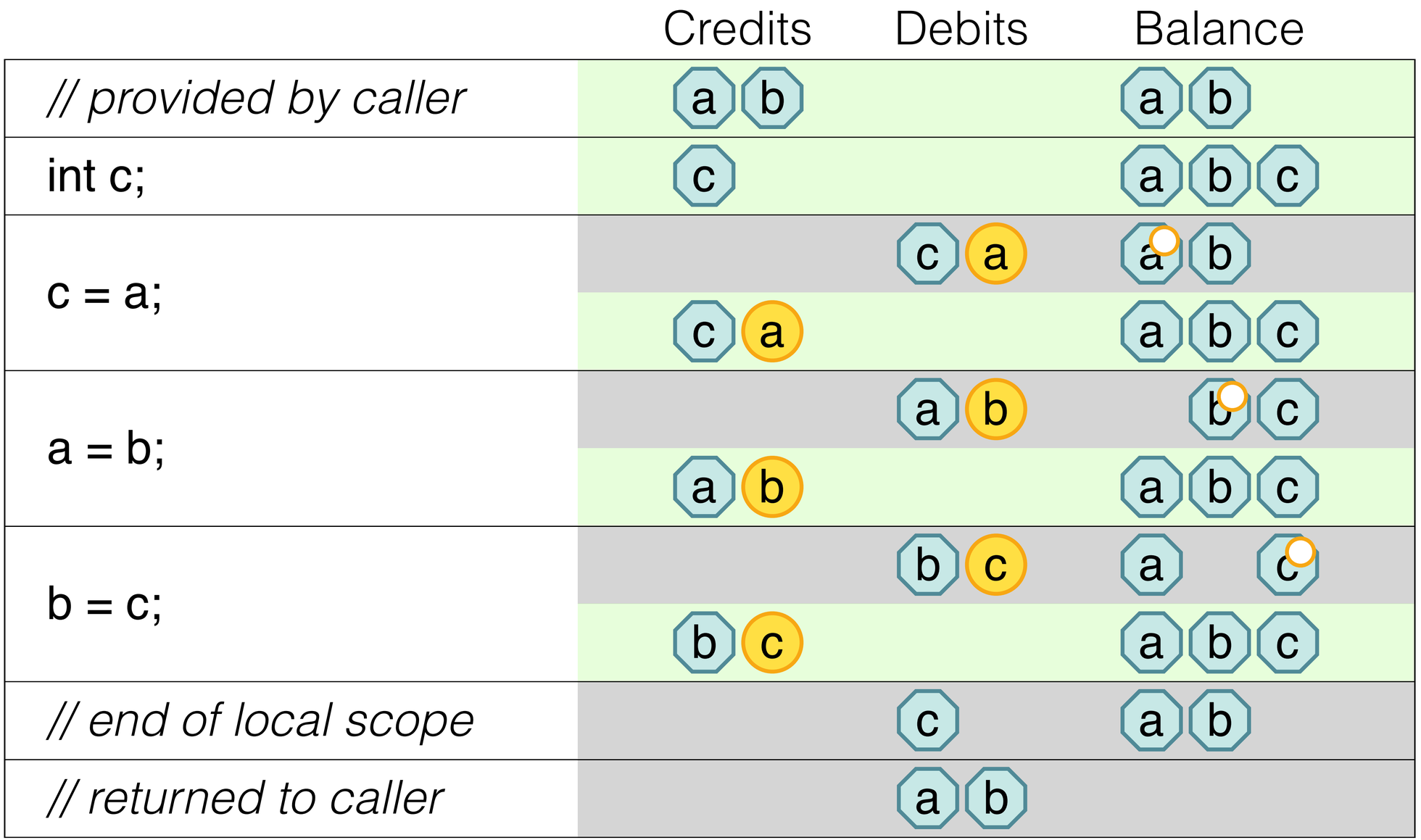
b = c;

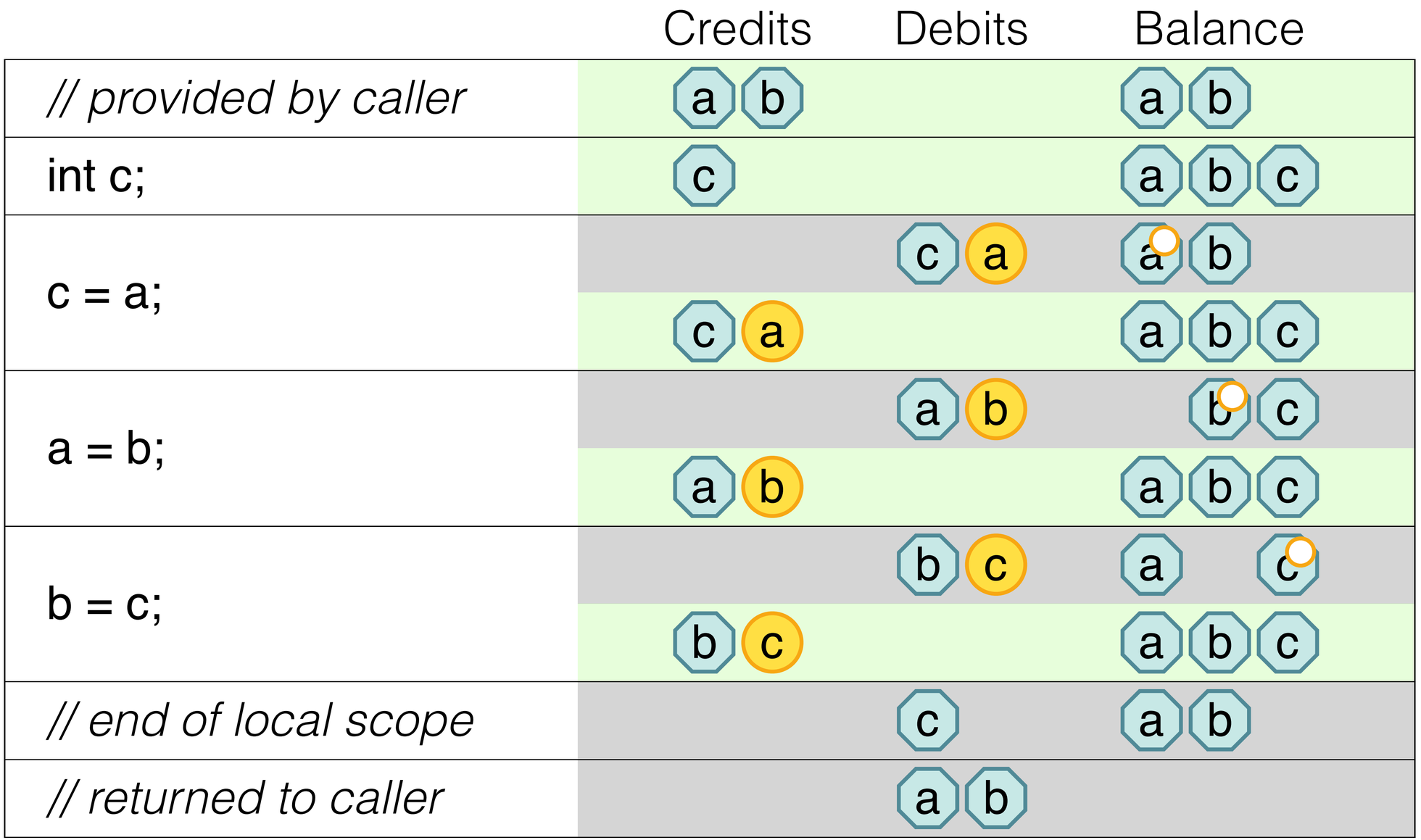

Credits
Debits
Balance
// provided by caller
a b
a b
int c;
c
a b c
c = a;
c a
a b
c a
a b c
a = b;
a b
b c
a b
a b c
b = c;

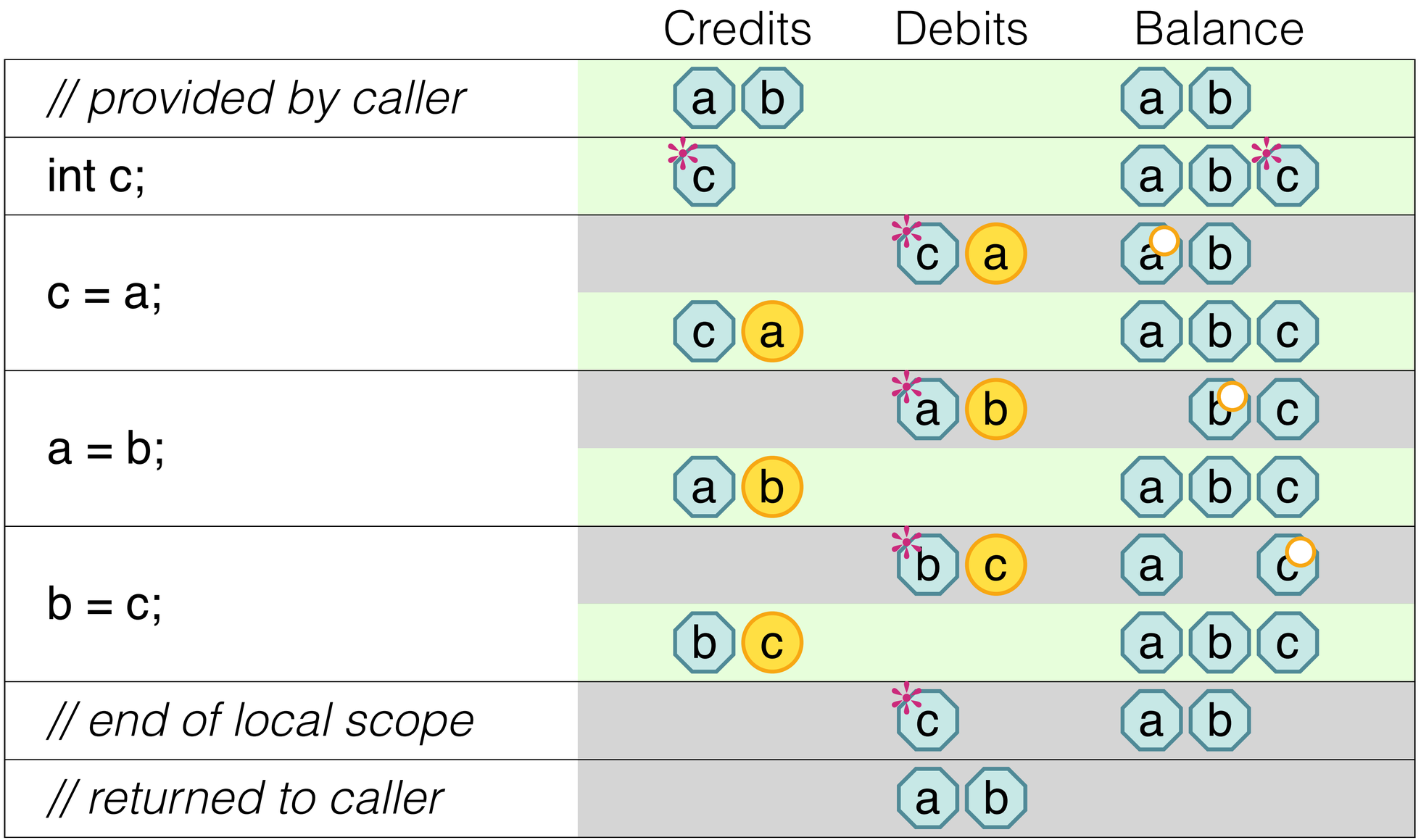

Credits
Debits
Balance
// provided by caller
a b
a b
int c;
c
a b c
c = a;
c a
a b
c a
a b c
a = b;
a b
b c
a b
a b c
b = c;
b c
a c
b c
a b c

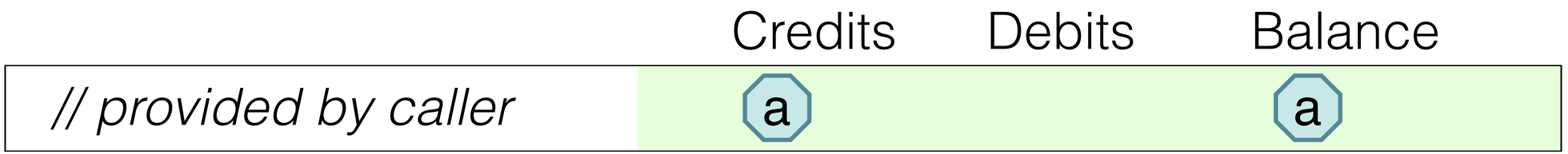

Credits
Debits
Balance
// provided by caller
a b
a b
int c;
c
a b c
c = a;
c a
a b
c a
a b c
a = b;
a b
b c
a b
a b c
b = c;
b c
b c
a c
a b c
// end of local scope
c


Credits
Debits
Balance
// provided by caller
a b
a b
int c;
c
a b c
c = a;
c a
a b
c a
a b c
a = b;
a b
b c
a b
a b c
b = c;
b c
b c
a c
a b c
// end of local scope
c
a b

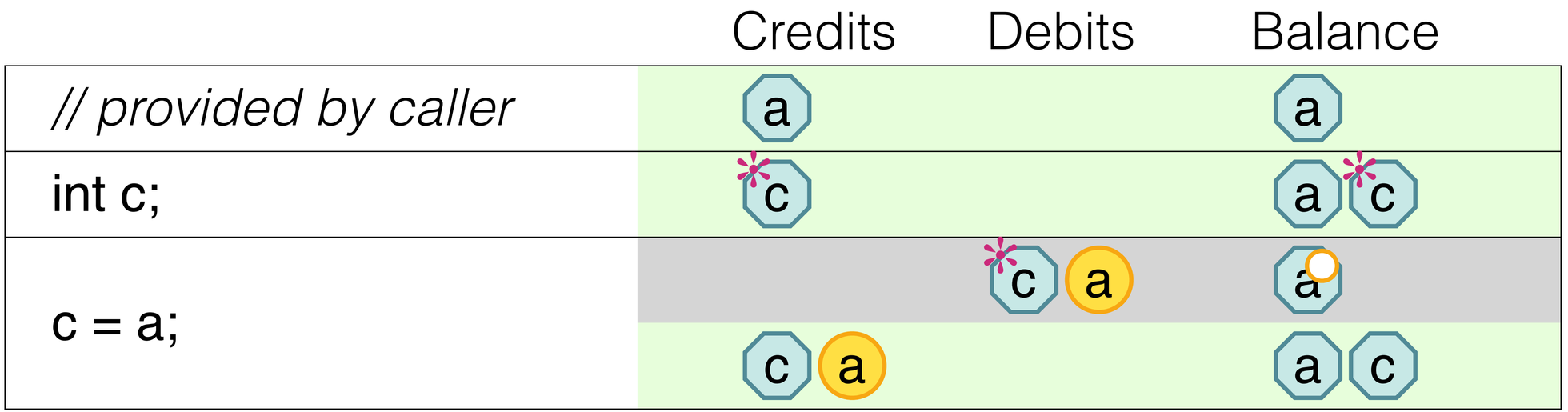

Credits
Debits
Balance
// provided by caller
a b
a b
int c;
c
a b c
c = a;
c a
a b
c a
a b c
a = b;
a b
b c
a b
a b c
b = c;
b c
b c
a c
a b c
// end of local scope
c
a b
// returned to caller
a b


Credits
Debits
Balance
// provided by caller
a b
a b
int c;
c
a b c
c = a;
c a
a b
c a
a b c
a = b;
a b
b c
a b
a b c
b = c;
b c
b c
a c
a b c
// end of local scope
c
a b
// returned to caller
a b

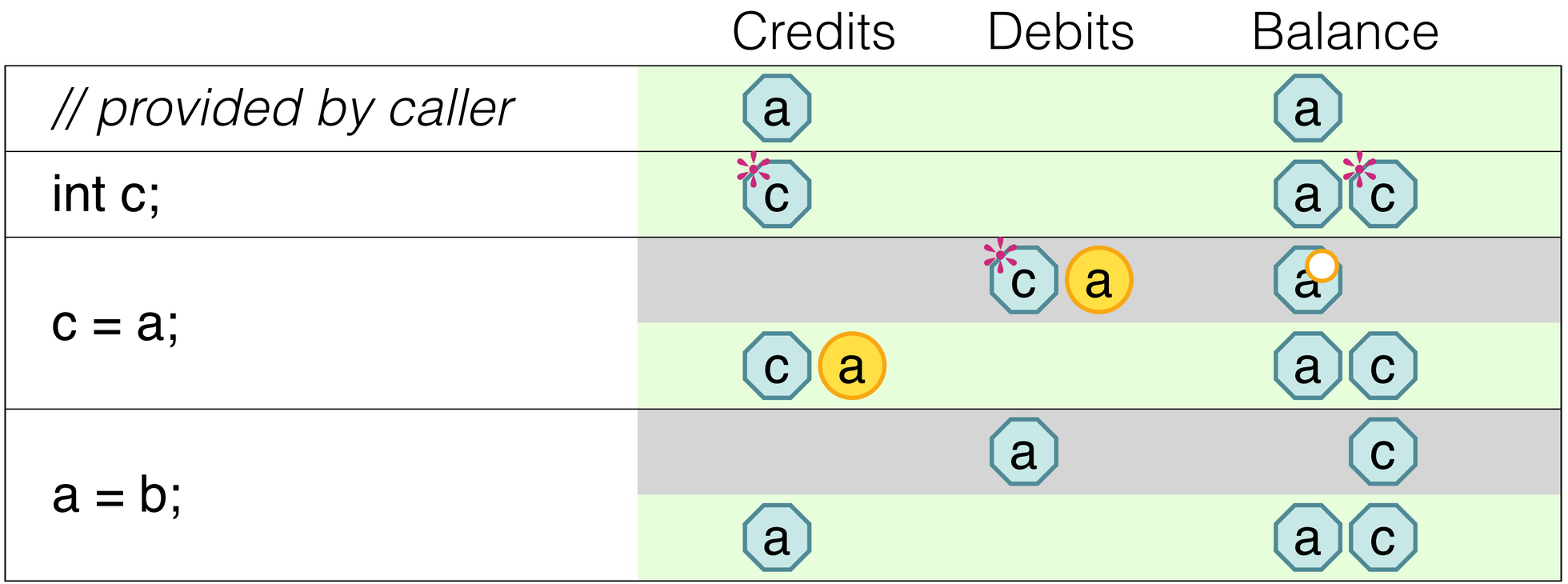

Credits
Debits
Balance
// provided by caller
int c;
c = a;
a = b;
b = c;
// end of local scope
// returned to caller

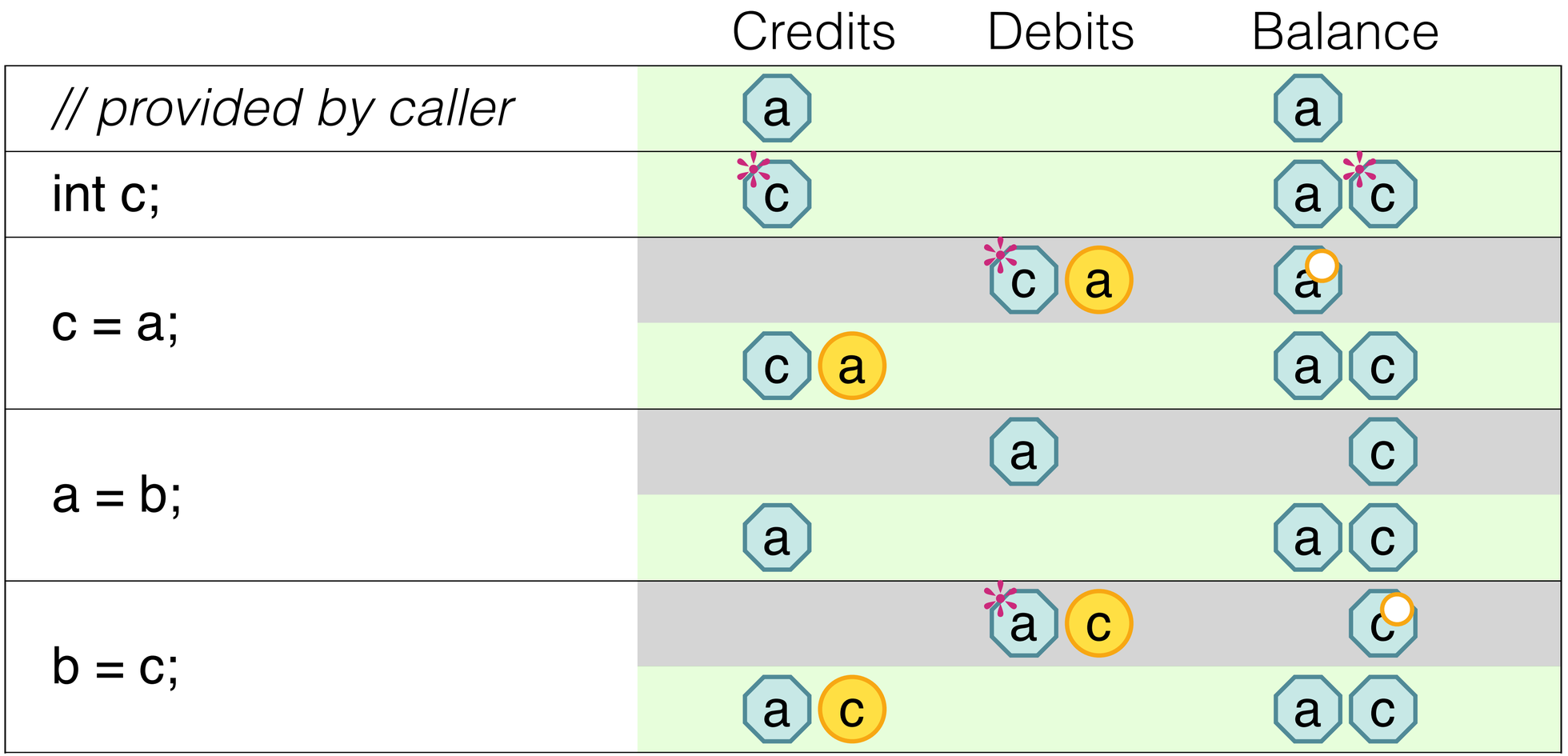


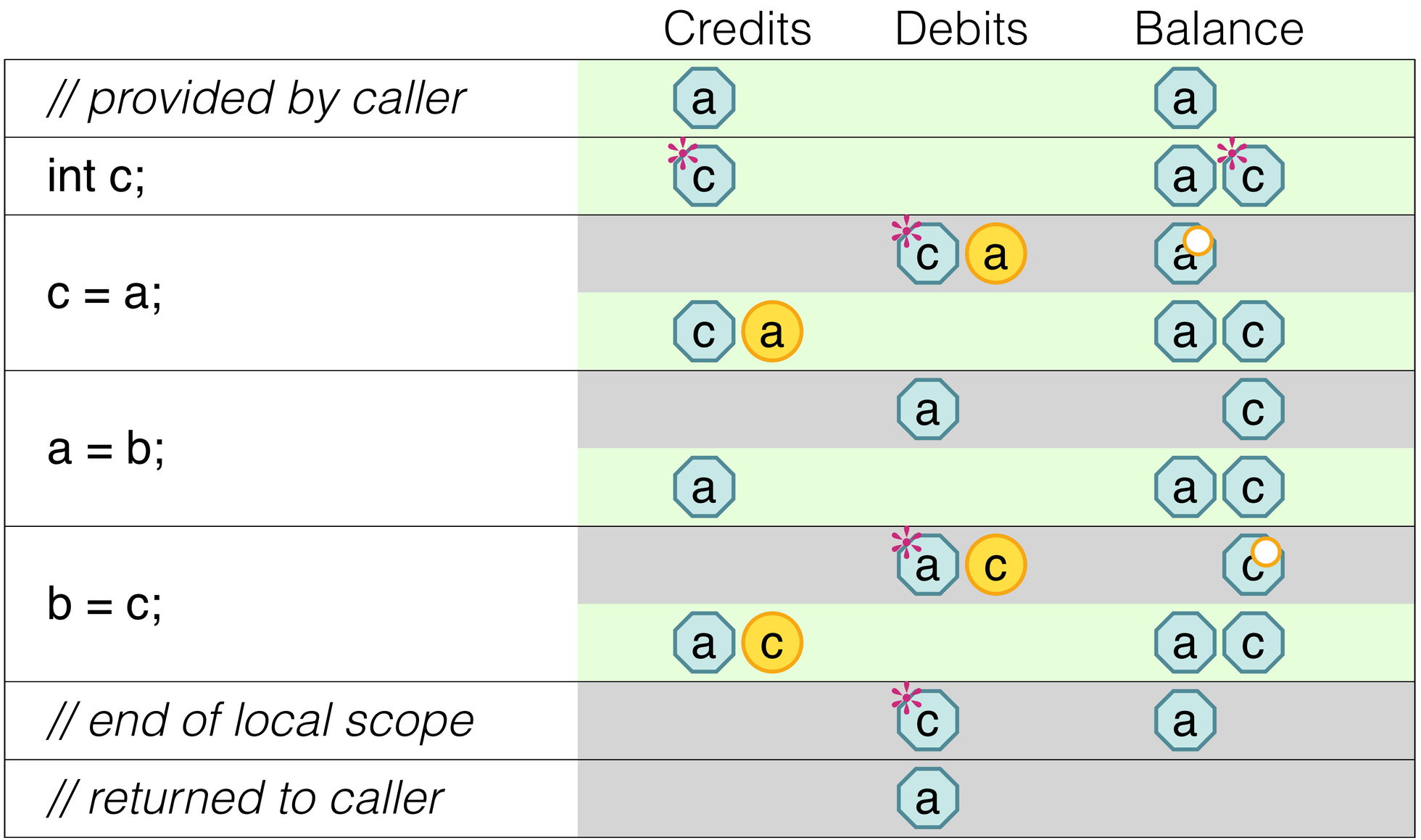

```
int c;
c = a;
a = b;
b = c;
```


Credits
Debits
Balance
// provided by caller
a
a
int c;
c
a
c
c = a;
c
a
a
c
a
a
c

a = b;
b = c;


Credits
Debits
Balance
// provided by caller
int c;
c = a;
a = b;
b = c;


Credits
Debits
Balance
// provided by caller
int c;
c = a;
a = b;
b = c;


Credits
Debits
Balance
// provided by caller
int c;
c = a;
a = b;
b = c;
// end of local scope
// returned to caller

```
void swap( int& a, int& b )
interface
 {
  if ( &a != &b )
   {
    transfer a, b;
    implementation;
    transfer a, b;
   }
  else
   {
    transfer a;
    transfer_reference b = a;
    implementation;
    transfer a;
   }
 }
```

Two object rights transferred

Two object rights returned

One object right transferred

One object right returned

```
void swap( int& a, int& b )
interface
 {
  if ( &a != &b )
   {
    transfer a, b;
    implementation;
    transfer a, b;
   }
  else
   {
    transfer a;
    transfer_reference b = a;
    implementation;
    transfer a;
   }
 }
```

```
int& operator=( int& a, const int& b )
interface
 {
  if ( &a != &b )
   {
    transfer a(maybe_uninitialized), const b;
    implementation;
    transfer a, const b;
   }
  else
   {
    transfer a;
    transfer_reference b = a;
    implementation;
    transfer a;
   }
  transfer_reference result = a;
 }
```

- value of **a**
- value of **b**
- value of **c**

initialization: **initialized** or **maybe_uninitialized**

value of **a**

value of **b**

value of **c**

initialization:

initialized or
maybe_uninitialized

lifetime of **a**

lifetime of **b**

lifetime of **c**

mutability:

not_declared_const or
maybe_declared_const

value of **a**

value of **b**

value of **c**

initialization: **initialized** or **maybe_uninitialized**

lifetime of **a**

lifetime of **b**

lifetime of **c**

mutability: **not_declared_const** or **maybe_declared_const**

storage for **a**

storage for **b**

storage for **c**

storage boundary: [**begin**, **end**)

value of **a**

value of **b**

value of **c**

initialization: **initialized** or **maybe_uninitialized**

referent: ledger column + generation

lifetime of **a**

lifetime of **b**

lifetime of **c**

mutability: **not_declared_const** or **maybe_declared_const**

storage for **a**

storage for **b**

storage for **c**

storage boundary: [**begin**, **end**)

```
void iter_swap( int *a, int *b )
 {
  swap( *a, *b );
 }
```

```
void reverse( int *a, int *b )
 {
  auto p = a, q = b;
  while ( p != q )
   {
    --q;
    if ( p == q )
      break;
    iter_swap( p, q );
    ++p;
   }
 }
```

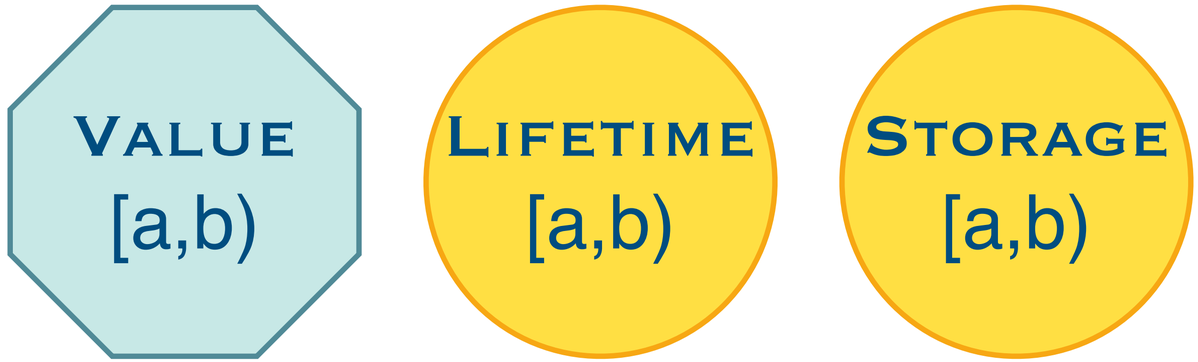
```
void iter_swap( int *a, int *b )
interface
 {
  if ( a != b )
   {
    int& sa = *a, sb = *b;
    transfer sa, sb, a = &sa, b = &sb;
    implementation;
    transfer sa, sb, a, b;
   }
  else
   {
    int& sa = *a;
    transfer sa, a = &sa, b = &sa;
    implementation;
    transfer sa, a, b;
   }
 }
```

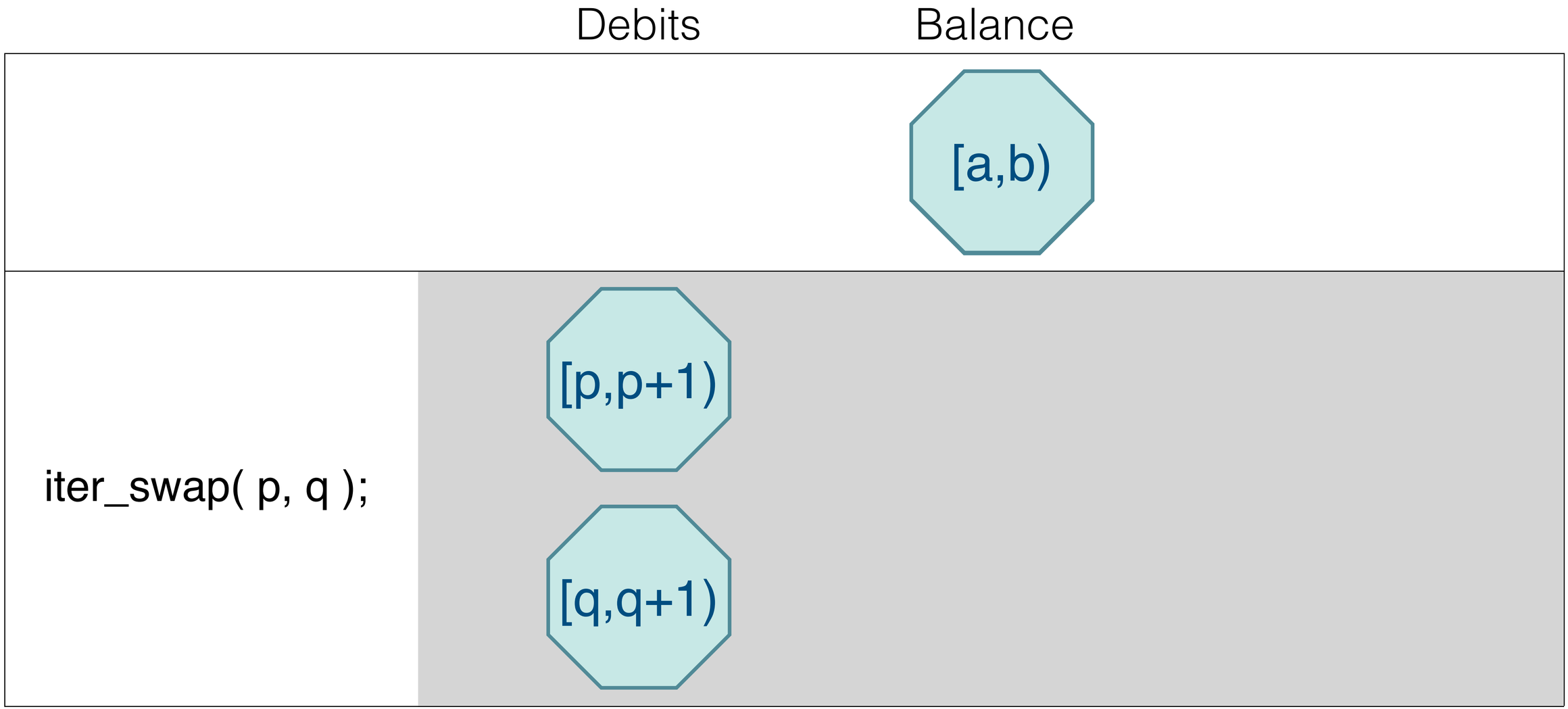
The pointers **a** and **b** point into the same ledger columns as **&sa** and **&sb**, respectively.

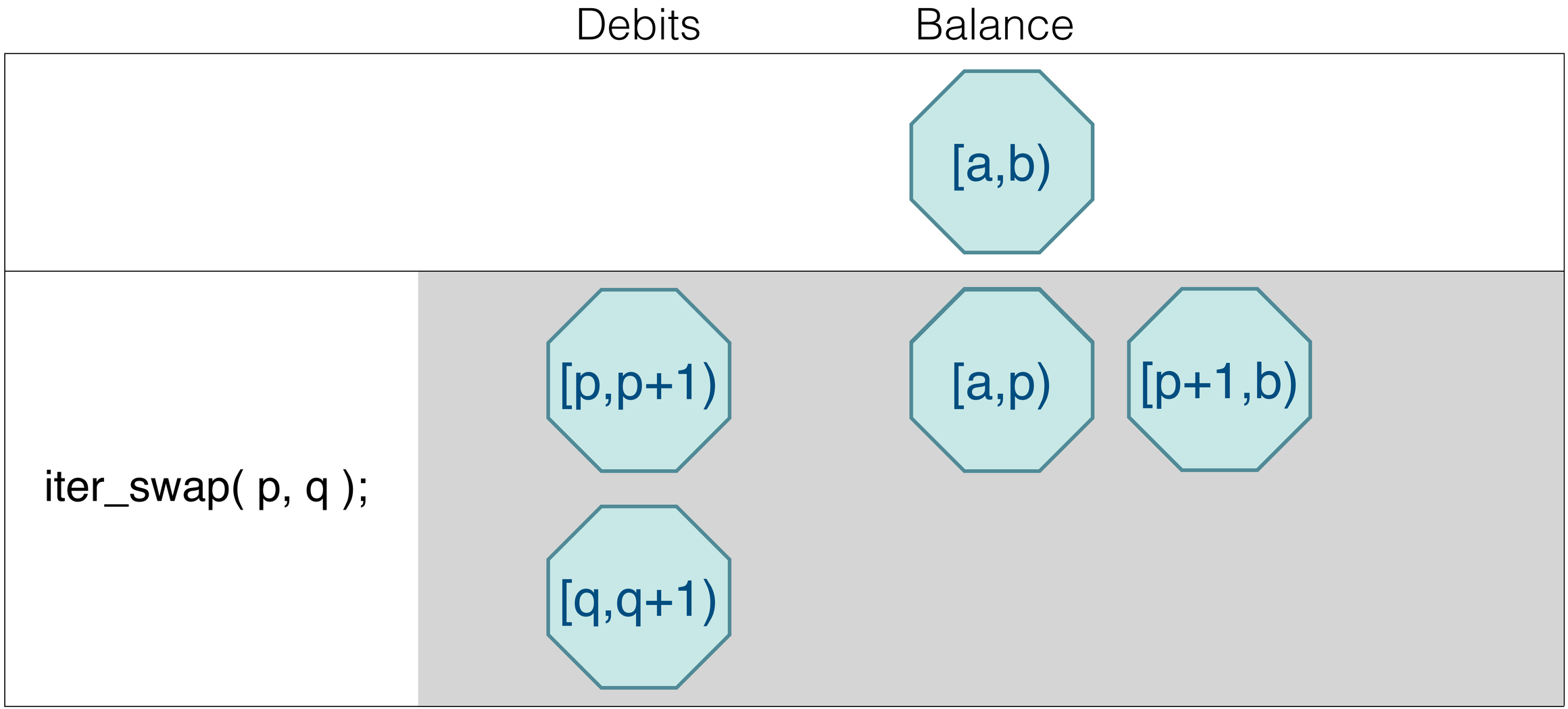
The pointers **a** and **b** both point to the same ledger column as **&sa**.

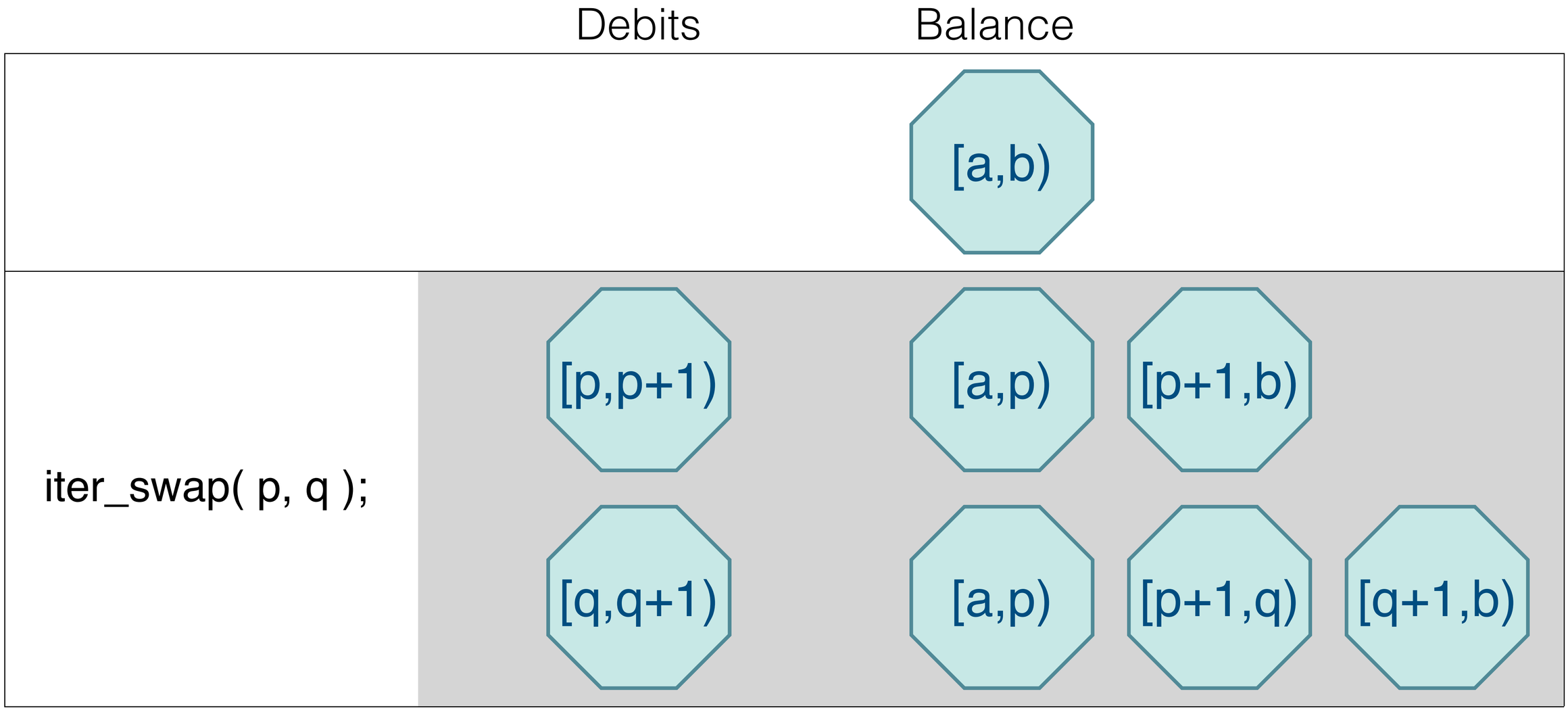
```
void reverse( int *a, int *b )
interface
 {
  const span_ref s( a, b );

  transfer s,
             a = s.begin(),
             b = s.end();

  implementation;

  transfer s, a, b;
 }
```

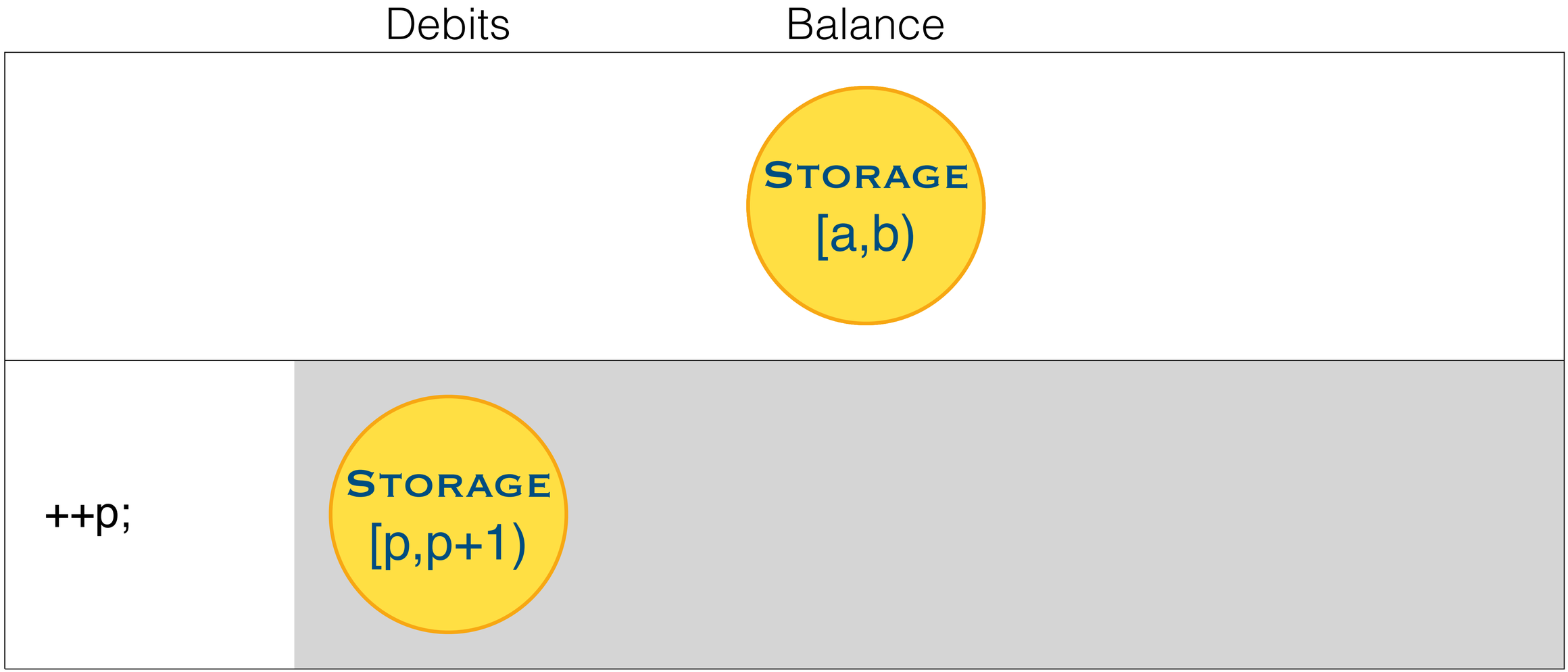
The referents of bulk reference types like **span_ref** are transferred as bulk tokens.

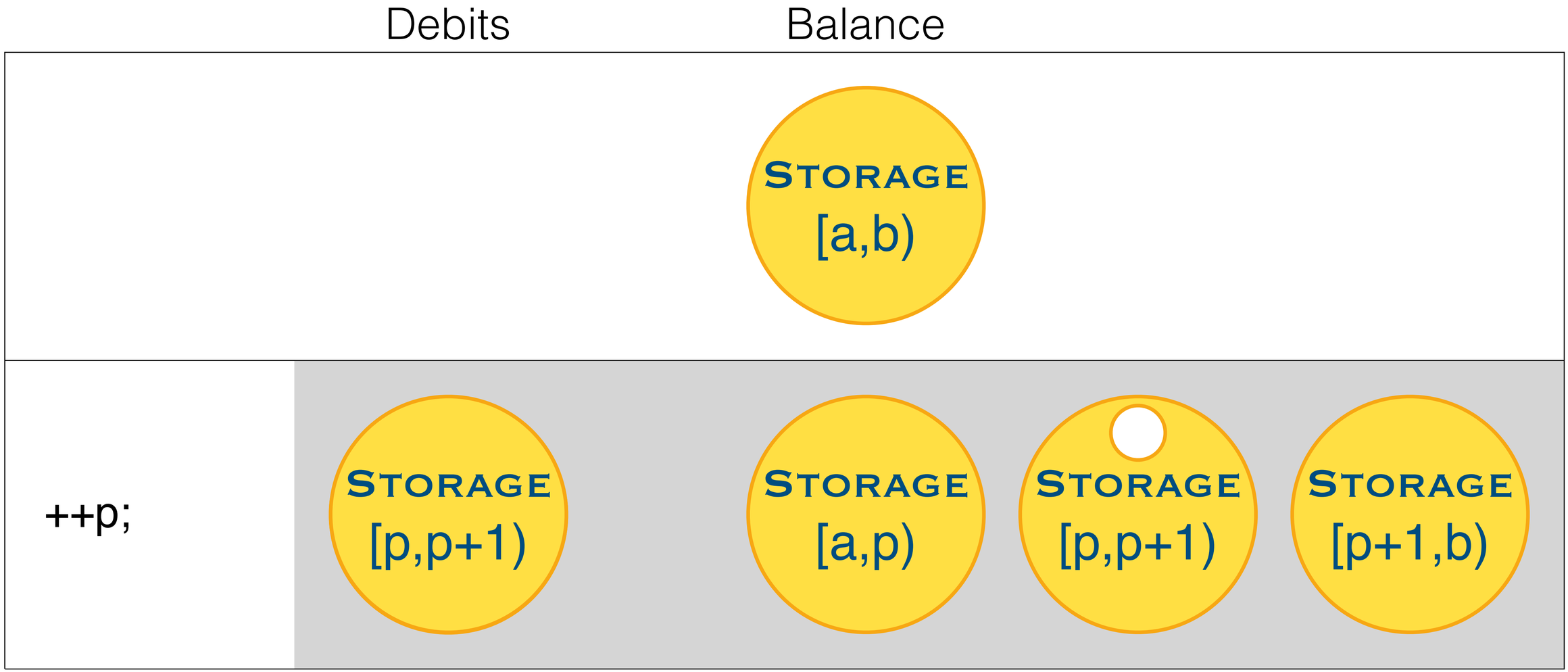


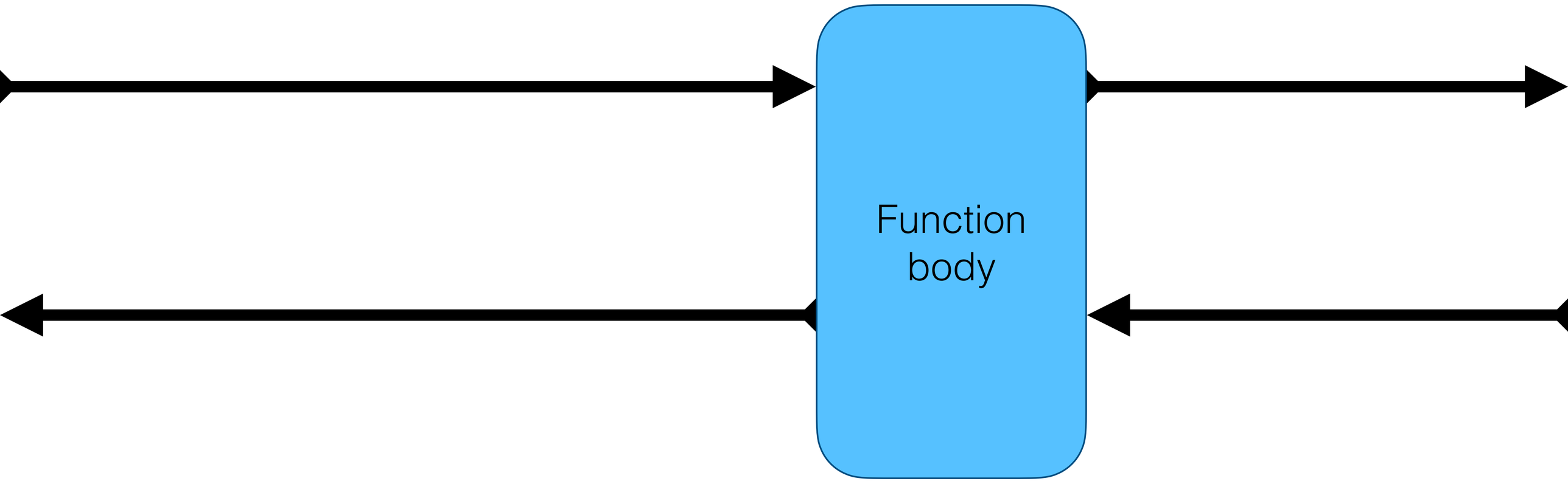

Debits
Balance
[a,b)
iter_swap(p, q);
[p,p+1)
[q,q+1)

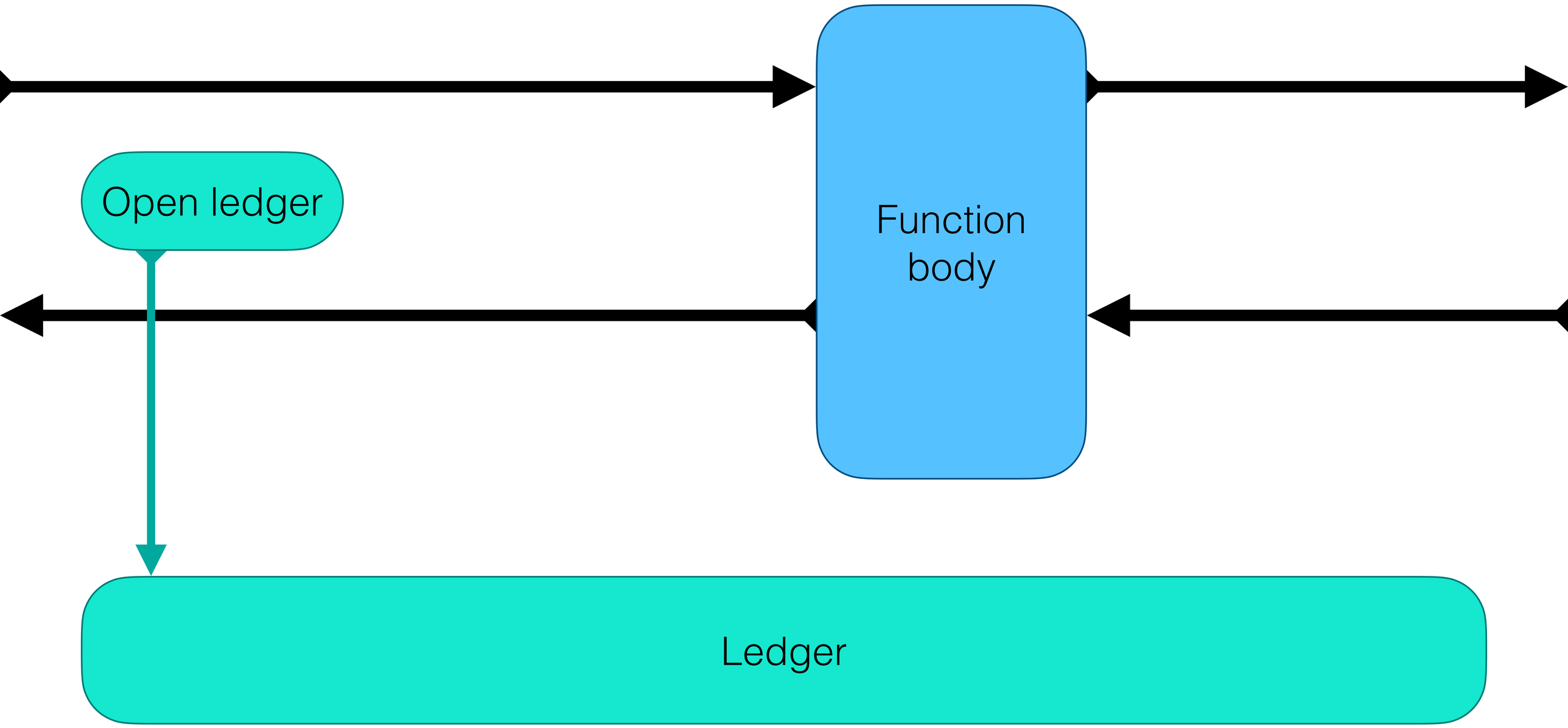

Debits
Balance
[a,b)
iter_swap(p, q);
[p,p+1)
[q,q+1)
[a,p)
[p+1,b)

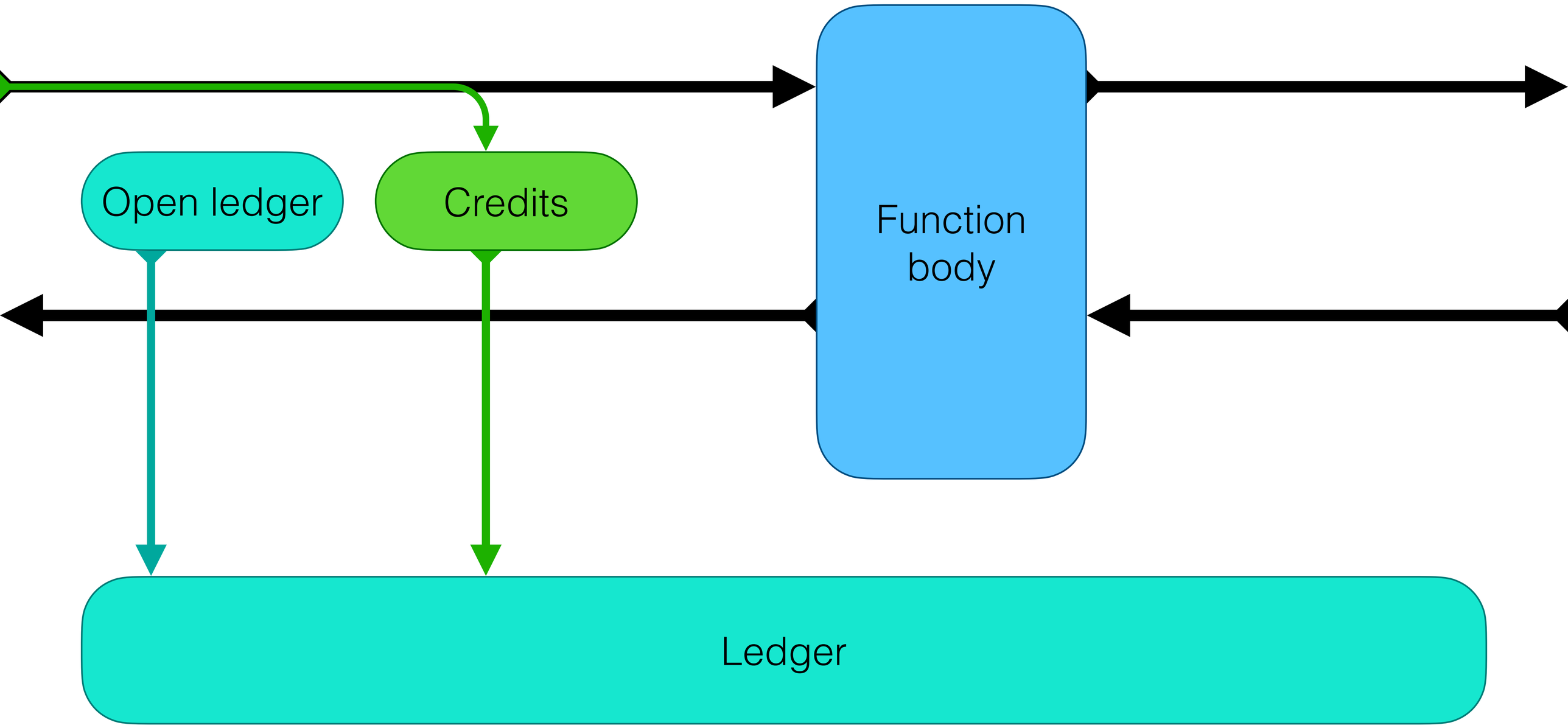

Debits
Balance
[a,b)
iter_swap(p, q);
[p,p+1)
[a,p)
[p+1,b)
[q,q+1)
[a,p)
[p+1,q)
[q+1,b)

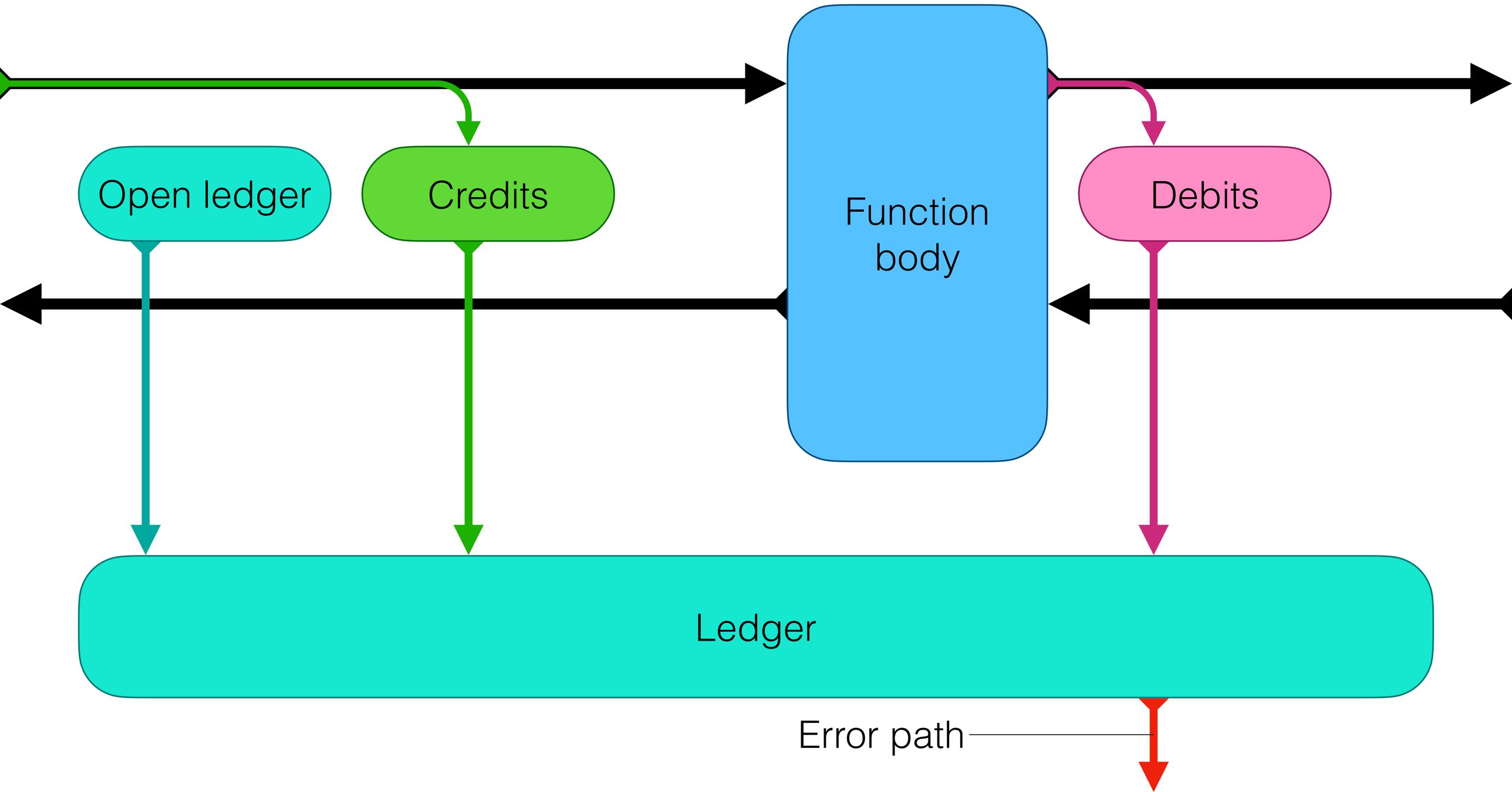

Debits
Balance
Storage
[a,b)
++p;
Storage
[p,p+1)

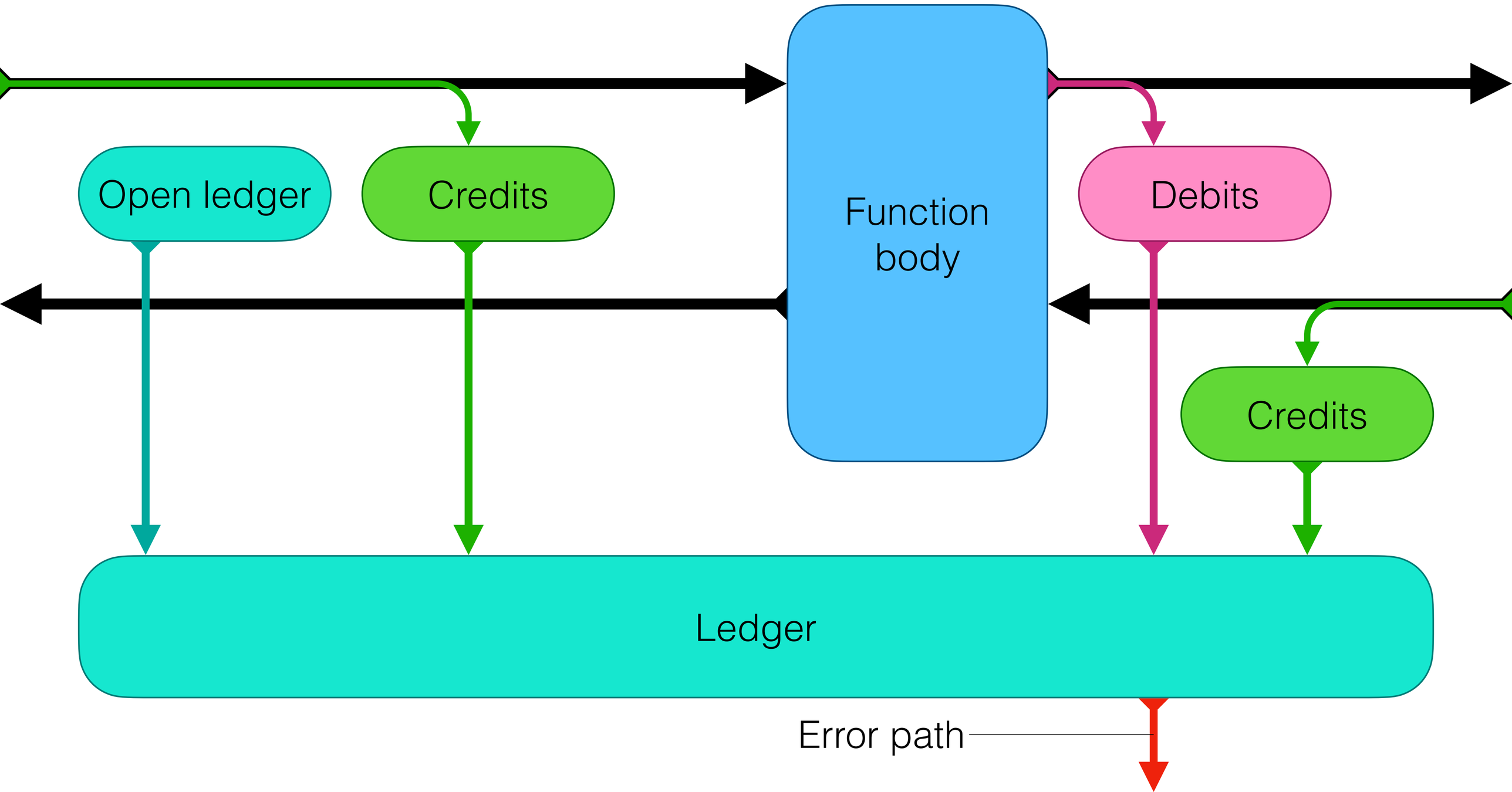

Debits
Balance
Storage
[a,b)
++p;
Storage
[p,p+1)
Storage
[a,p)
Storage
[p,p+1)
Storage
[p+1,b)

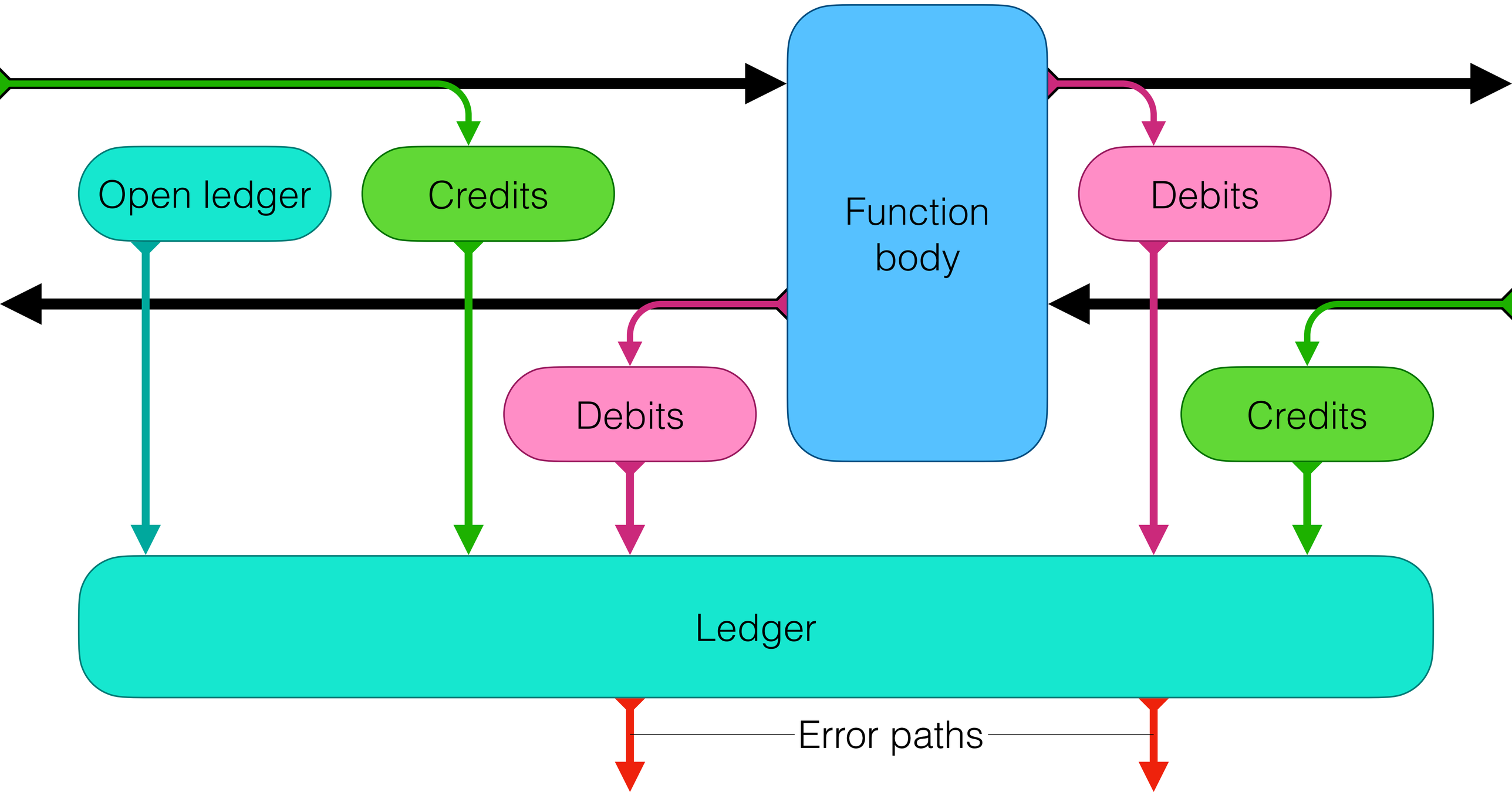

Function
body

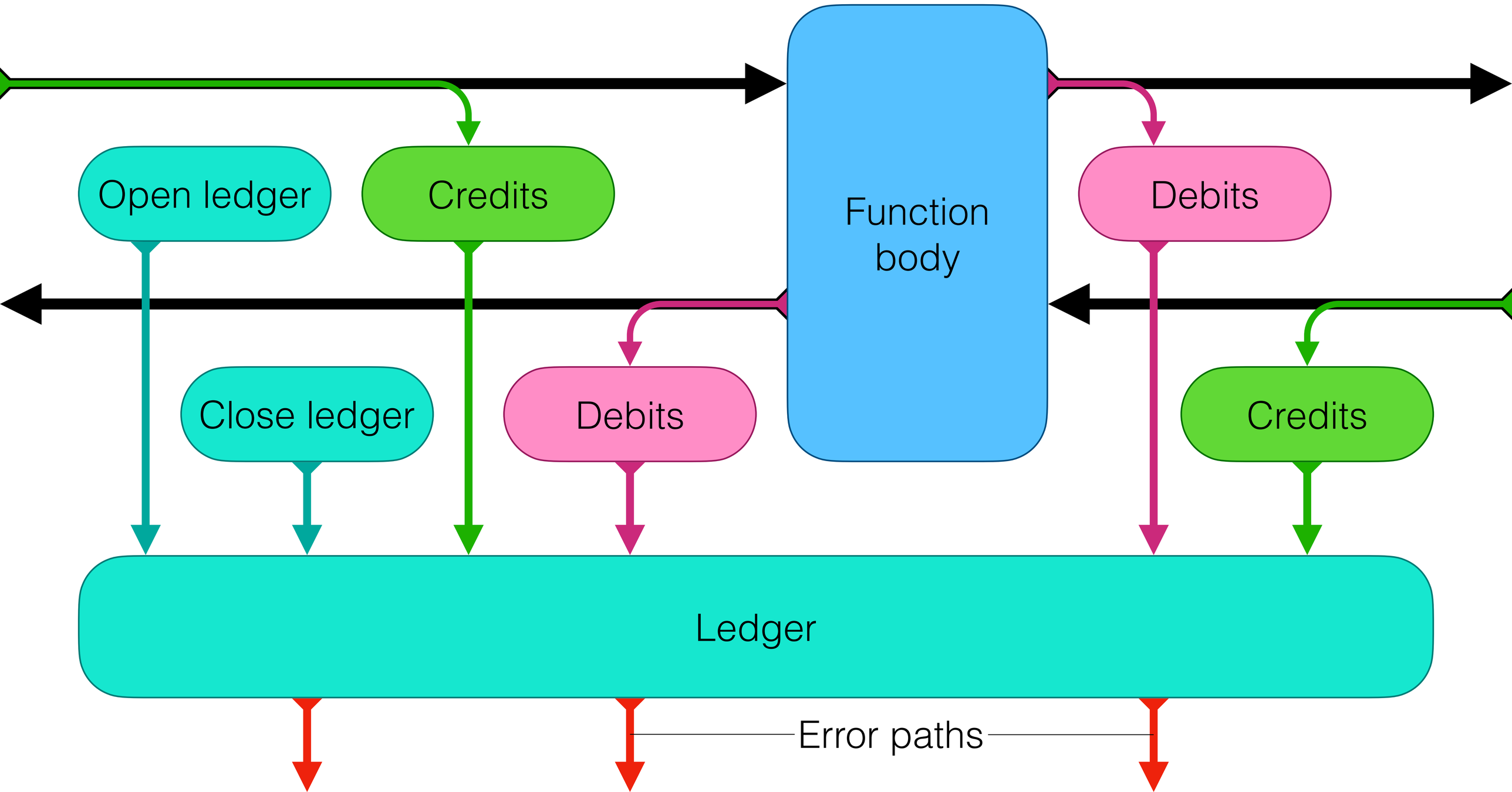

Open ledger
Function body
Ledger

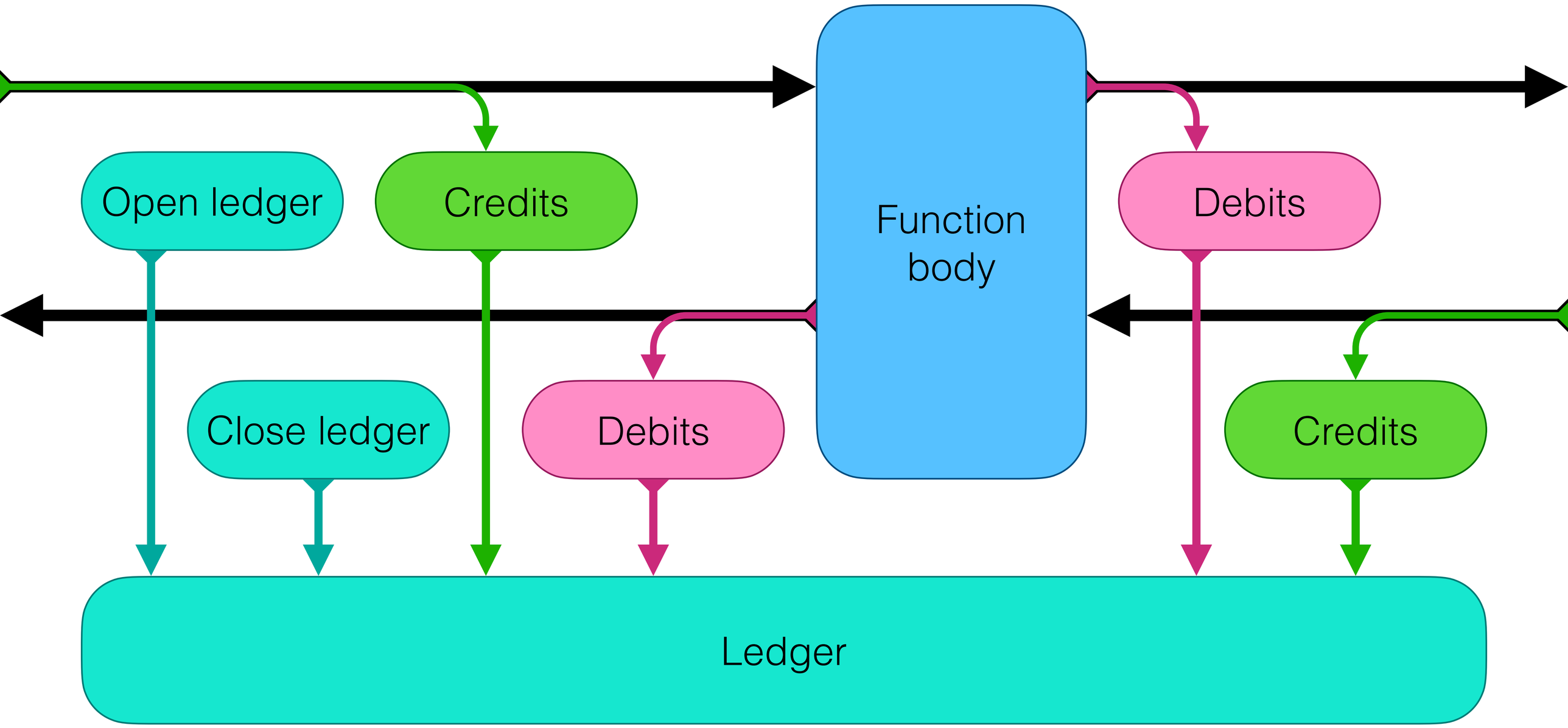

Open ledger
Credits
Function body
Ledger

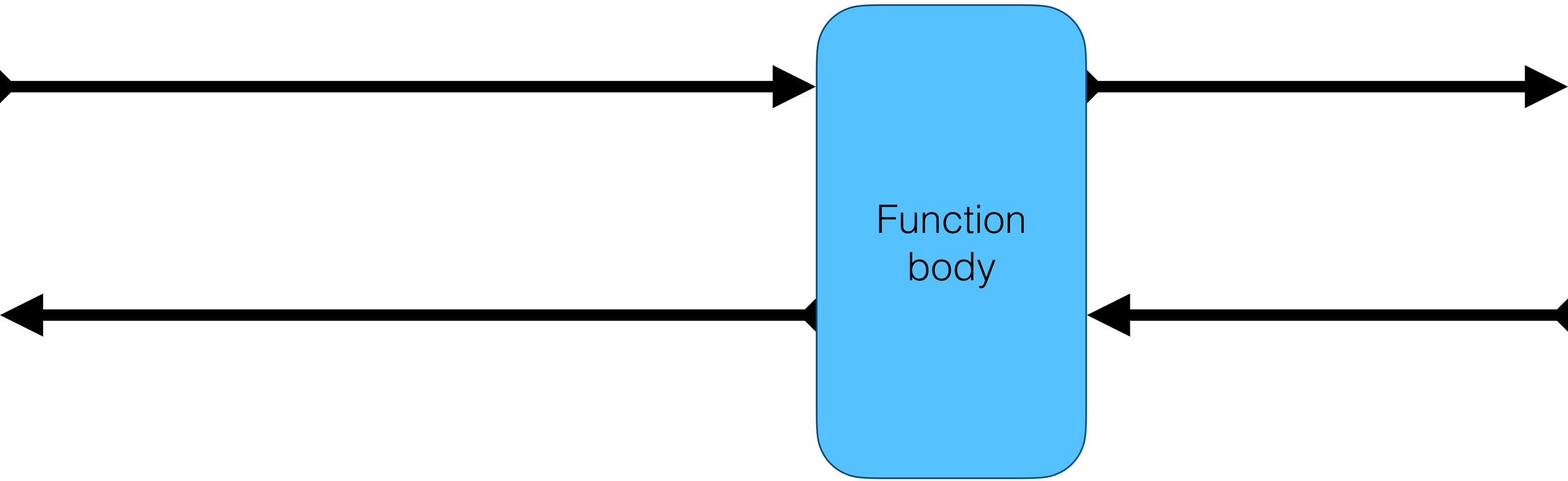

Open ledger
Credits
Function body
Debits
Ledger
Error path


Open ledger
Credits
Function body
Debits
Credits
Ledger
Error path

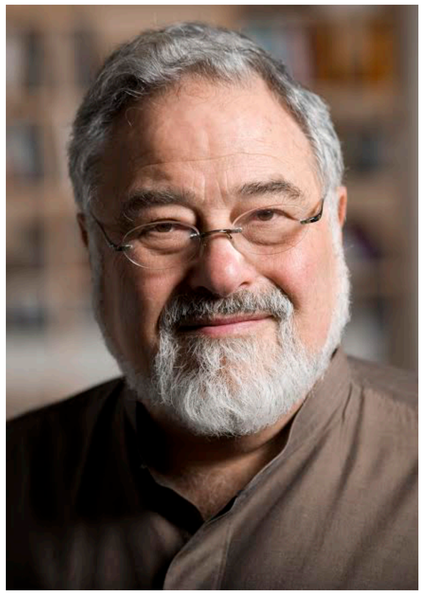

Open ledger
Credits
Function body
Debits
Debits
Credits
Ledger
Error paths


Open ledger
Credits
Function body
Debits
Close ledger
Debits
Credits
Ledger
Error paths


Open ledger
Credits
Function body
Debits
Close ledger
Debits
Credits
Ledger


Function
body

If your optimizer removes every ledger error path,
your function gets a gold star!



George Lakoff (1941-)

Metaphors We Live By
(with Mark Johnson)
University of Chicago Press, 1980

Where Mathematics Comes From
(with Rafael E. Núñez)
Basic Books, 2000

The world is rich with solutions.
A good metaphor lets us apply them.

Thank you for listening.

Questions?