

LECTURE 3

THURSDAY SEPTEMBER 12

- In-Lab Demo yesterday
(source code, reading)

- TA office hours
11 am ~ 2pm FRIDAYS
LAS 2056

- slides on Bow diagram soon.

Design by Contract in Java

CLIENT

```
public static void main(String[] args) {
    System.out.println("Create an account for Jim with balance 100:");
    try {
        AccountV2 jim = new AccountV2("Jim", 100);
        System.out.println(jim);
        System.out.println("Withdraw 100 from Jim's account:");
        jim.withdraw(100);
        System.out.println(jim);
    }
    catch (BalanceNegativeException bne) {
        System.out.println("Illegal negative account balance.");
    }
    catch (WithdrawAmountNegativeException wane) {
        System.out.println("Illegal negative withdraw amount.");
    }
    catch (WithdrawAmountTooLargeException wane) {
        System.out.println("Illegal too large withdraw amount.");
    }
}
```

SUPPLIER

```
public class AccountV5 {
    public void withdraw(int amount) throws
        WithdrawAmountNegativeException, WithdrawAmountTooLargeException {
        int oldBalance = this.balance;
        if (amount < 0) { /* negated precondition */
            throw new WithdrawAmountNegativeException(); }
        else if (balance < amount) { /* negated precondition */
            throw new WithdrawAmountTooLargeException(); }
        else { this.balance = this.balance - amount; }
        assert this.getBalance() > 0 : "Invariant: positive balance";
        assert this.getBalance() == oldBalance - amount :
            "Postcondition: balance deducted"; }
}
```

Design by Contract in Eiffel

Implementation View

```
class ACCOUNT
create
  make
feature -- Attributes
  owner : STRING
  balance : INTEGER
feature -- Constructors
  make(nn: STRING; nb: INTEGER)
    require -- precondition
      positive_balance: nb > 0
    do
      owner := nn
      balance := nb
    end
feature -- Commands
  withdraw(amount: INTEGER)
    require -- precondition
      non_negative_amount: amount > 0
      affordable_amount: amount <= balance -- problematic
    do
      balance := balance - amount
    ensure -- postcondition
      balance_deducted: balance = old balance - amount
    end
invariant -- class invariant
  positive_balance: balance > 0
end
```

```
class ACCOUNT
create
  make
feature -- Attributes
  owner : STRING
  balance : INTEGER
feature -- Constructors
  make(nn: STRING; nb: INTEGER)
    require -- precondition
      positive_balance: nb > 0
    end
feature -- Commands
  withdraw(amount: INTEGER)
    require -- precondition
      non_negative_amount: amount ≥ 0
      affordable_amount: amount <= balance -- problematic, why?
    ensure -- postcondition
      balance_deducted: balance = old balance - amount
    end
invariant -- class invariant
  positive_balance: balance > 0
end
```

Contract View

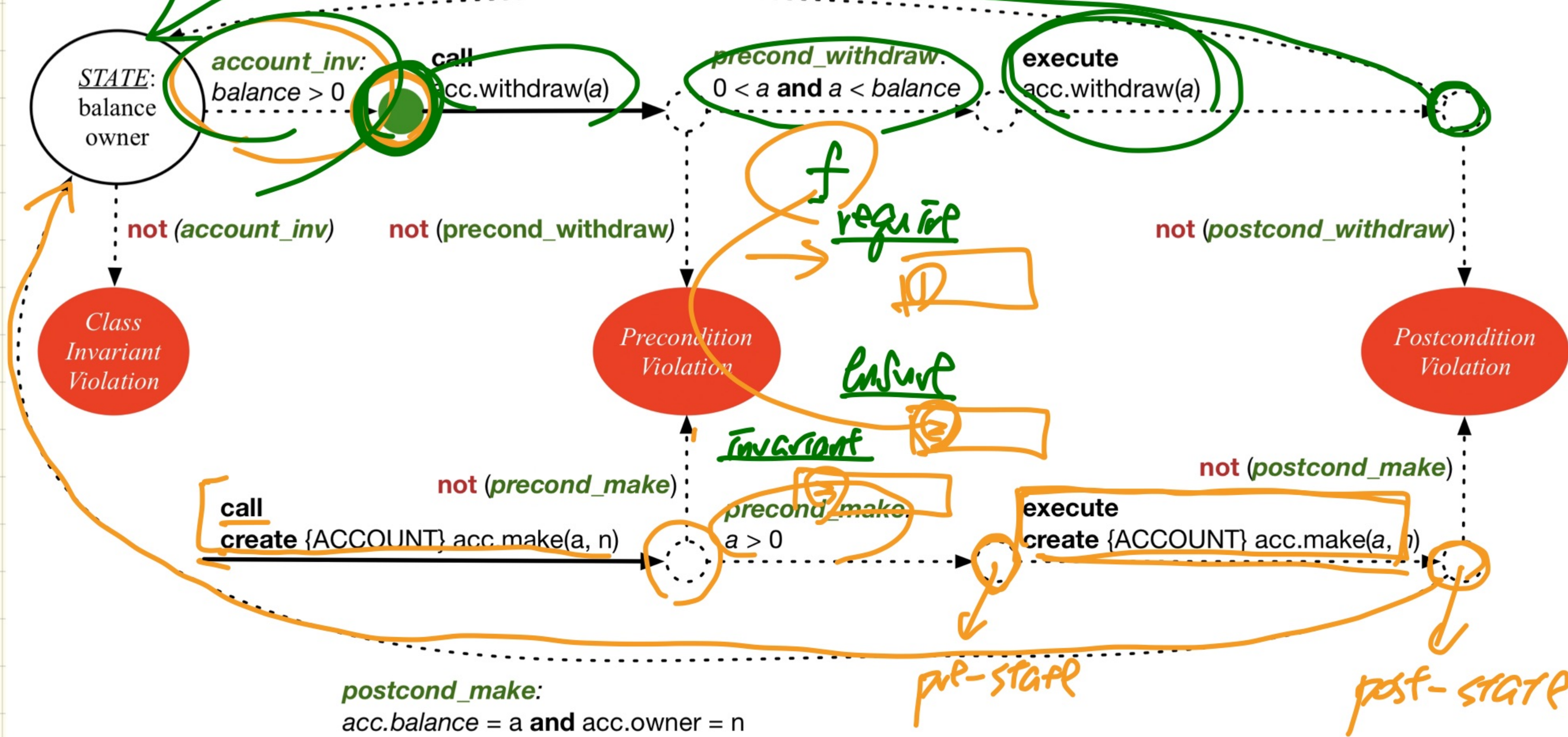
Runtime Monitoring of Contracts

- ① valid inputs (invariant not violated)
- ③ valid objects
- ② pre-state and post-state values are related properly

```

acc: ACCOUNT
create acc.make(a, n)
acc.withdraw(a)
    
```

postcond withdraw:
 $acc.balance = \text{old } acc.balance - a$ and $acc.owner \sim \text{old } acc.owner$



Precondition Violation: **positive_balance**

Call Stack

positive_balance: RECONDITION_VIOLATION raised

In Feature	In Class	From Class	@
make	ACCOUNT	ACCOUNT	1
make	APPLICATION	APPLICATION	1

```

make (nn: STRING_8; nb: INTEGER_32)
  require
  positive_balance nb > 0
  do
    owner := nn
    balance := nb
  end
  
```

Client

```

class BANK_APP
inherit
  ARGUMENTS
create
  make
feature -- Initialization
  make
  -- Run application.
local
  alan: ACCOUNT
do
  -- A precondition violation with tag end
  create {ACCOUNT} alan.make ("Alan", -10)
end
end
  
```

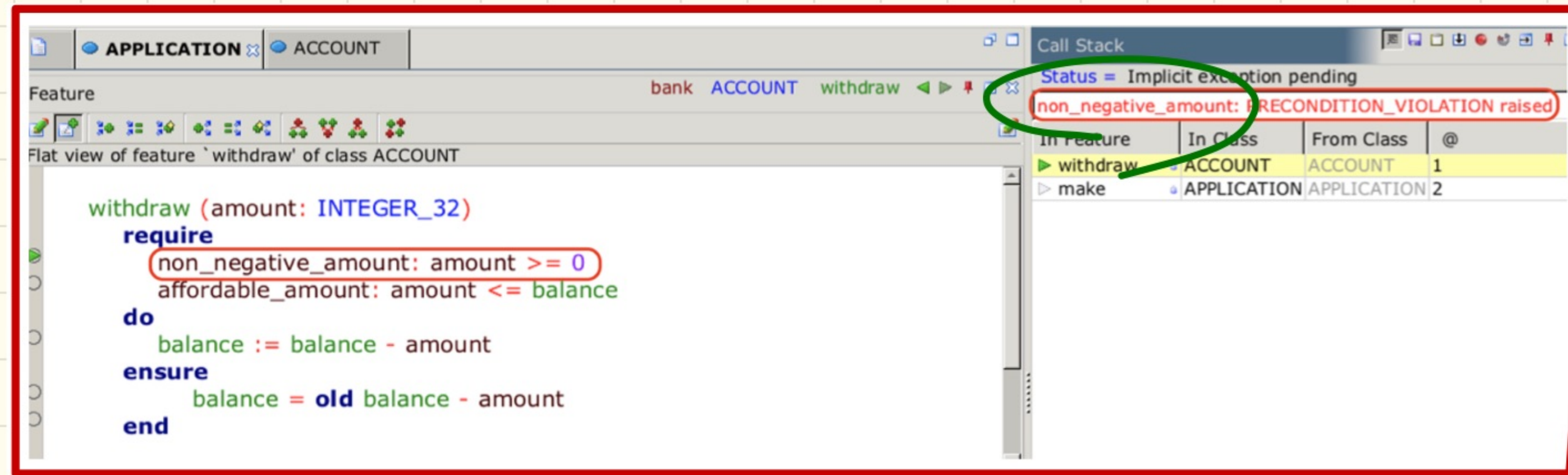
Supplier

```

class ACCOUNT
create
  make
feature -- Attributes
  owner : STRING
  balance : INTEGER
feature -- Constructors
  make (nn: STRING; nb: INTEGER)
  require -- precondition
    positive_balance: nb > 0
  end
feature -- Commands
  withdraw (amount: INTEGER)
  require -- precondition
    non_negative_amount: amount >= 0
    affordable_amount: amount <= balance -- problema
  ensure -- postcondition
    balance_deducted: balance = old balance - amount
  end
invariant -- class invariant
  positive_balance: balance > 0
  
```

(F)

Precondition Violation: non_negative_amount



```
APPLICATION ACCOUNT  
Feature bank ACCOUNT withdraw  
Flat view of feature `withdraw` of class ACCOUNT  
withdraw (amount: INTEGER_32)  
  require  
    non_negative_amount: amount >= 0  
    affordable_amount: amount <= balance  
  do  
    balance := balance - amount  
  ensure  
    balance = old balance - amount  
end
```

Call Stack
Status = Implicit exception pending
non_negative_amount: PRECONDITION_VIOLATION raised
In Feature In Class From Class @
withdraw ACCOUNT ACCOUNT 1
make APPLICATION APPLICATION 2

Client

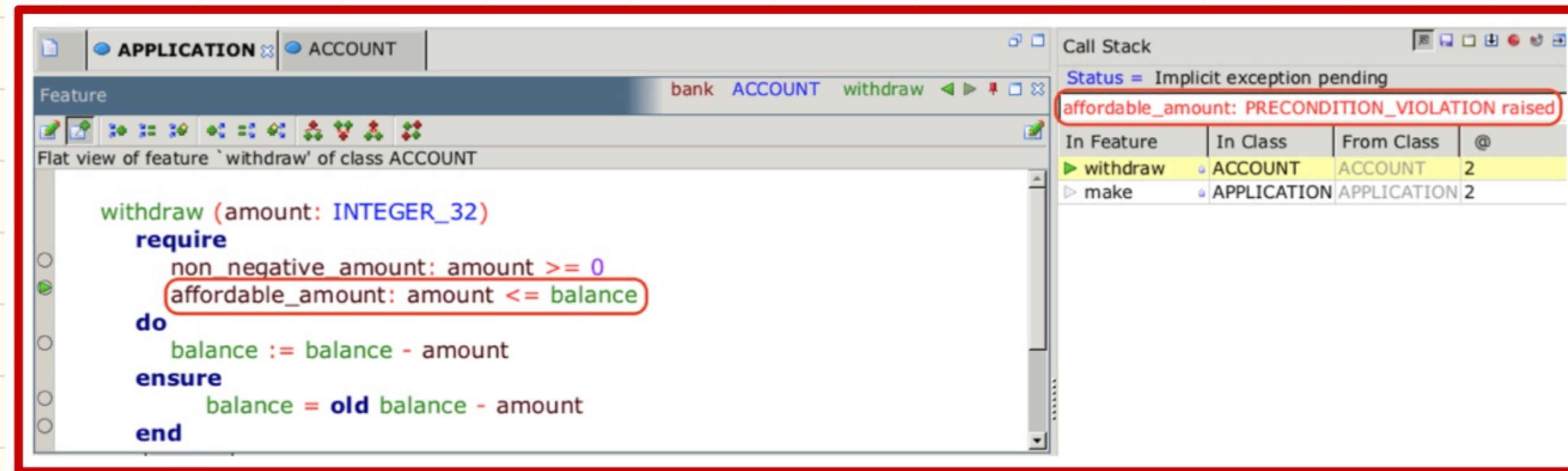
```
class BANK_APP  
inherit  
  ARGUMENTS  
create  
  make  
feature -- Initialization  
  make  
  -- Run application.  
local  
  mark: ACCOUNT  
do  
  create {ACCOUNT} mark.make ("Mark", 100)  
  -- A precondition violation with tag "nc"  
  mark.withdraw(-1000000)  
end  
end
```

Supplier

```
class ACCOUNT  
create  
  make  
feature -- Attributes  
  owner : STRING  
  balance : INTEGER  
feature -- Constructors  
  make(nn: STRING; nb: INTEGER)  
    require -- precondition  
      positive_balance: nb > 0  
    end  
feature -- Commands  
  withdraw(amount: INTEGER)  
    require -- precondition  
      non_negative_amount: amount >= 0  
      affordable_amount: amount <= balance -- problema  
    ensure -- postcondition  
      balance_deducted: balance = old balance - amount  
    end  
invariant -- class invariant  
  positive_balance: balance > 0  
end
```

Precondition Violation:

affordable_amount



Supplier

Client

```
class BANK_APP
inherit
  ARGUMENTS
create
  make
feature -- Initialization
  make
    -- Run application.
local
  tom: ACCOUNT
do
  create {ACCOUNT} tom.make ("Tom", 100)
  -- A precondition violation with tag "
  tom.withdraw(150)
end
end
```

```
class ACCOUNT
create
  make
feature -- Attributes
  owner : STRING
  balance : INTEGER
feature -- Constructors
  make(nn: STRING; nb: INTEGER)
    require -- precondition
      positive_balance: nb > 0
    end
feature -- Commands
  withdraw(amount: INTEGER)
    require -- precondition
      non_negative_amount: amount >= 0
      affordable_amount: amount <= balance -- problema
    ensure -- postcondition
      balance_deducted: balance = old balance - amount
    end
invariant -- class invariant
  positive_balance: balance > 0
end
```


Class Invariant Violation: **positive_balance**

Call Stack

Status = Implicit exception pending

positive_balance: INVARIANT_VIOLATION raised

In Feature	In Class	From Class	@
▶ _invariant	ACCOUNT	ACCOUNT	0
▷ withdraw	ACCOUNT	ACCOUNT	5
▷ make	APPLICATION	APPLICATION	2

positive_balance: balance > 0

Client

```
class BANK_APP
inherit
  ARGUMENTS
create
  make
feature -- Initialization
  make
    -- Run application.
  local
    jim: ACCOUNT
  do
    create {ACCOUNT} tom.make ("Jim", 100)
    jim.withdraw(100)
    -- A class invariant violation with tag "positive_balance"
  end
end
```

Supplier

```
class ACCOUNT
create
  make
feature -- Attributes
  owner : STRING
  balance : INTEGER
feature -- Constructors
  make(nn: STRING; nb: INTEGER)
    require -- precondition
      positive_balance: nb > 0
    end
feature -- Commands
  withdraw(amount: INTEGER)
    require -- precondition
      non_negative_amount: amount ≥ 0
      affordable_amount: amount ≤ balance -- problema
    ensure -- postcondition
      balance_deducted: balance = old balance - amount
    end
invariant -- class invariant
  positive_balance: balance > 0
end
```

Postcondition Violation: balance_deducted

The screenshot shows a development environment with two panes. The left pane displays the source code for the `ACCOUNT` class, specifically the `withdraw` feature. The code includes a pre-condition `affordable_amount: amount <= balance`, a `do` block with `balance := balance + amount`, and an `ensure` block with the post-condition `balance_deducted: balance = old balance - amount`. The `ensure` block is highlighted with a red circle. The right pane shows the Call Stack, with the status `Implicit exception pending` and a message `balance_deducted: POSTCONDITION_VIOLATION raised` highlighted in red. Below the message is a table:

In Feature	In Class	From Class	@
▶ withdraw	ACCOUNT	ACCOUNT	4
▶ make	APPLICATION	APPLICATION	2

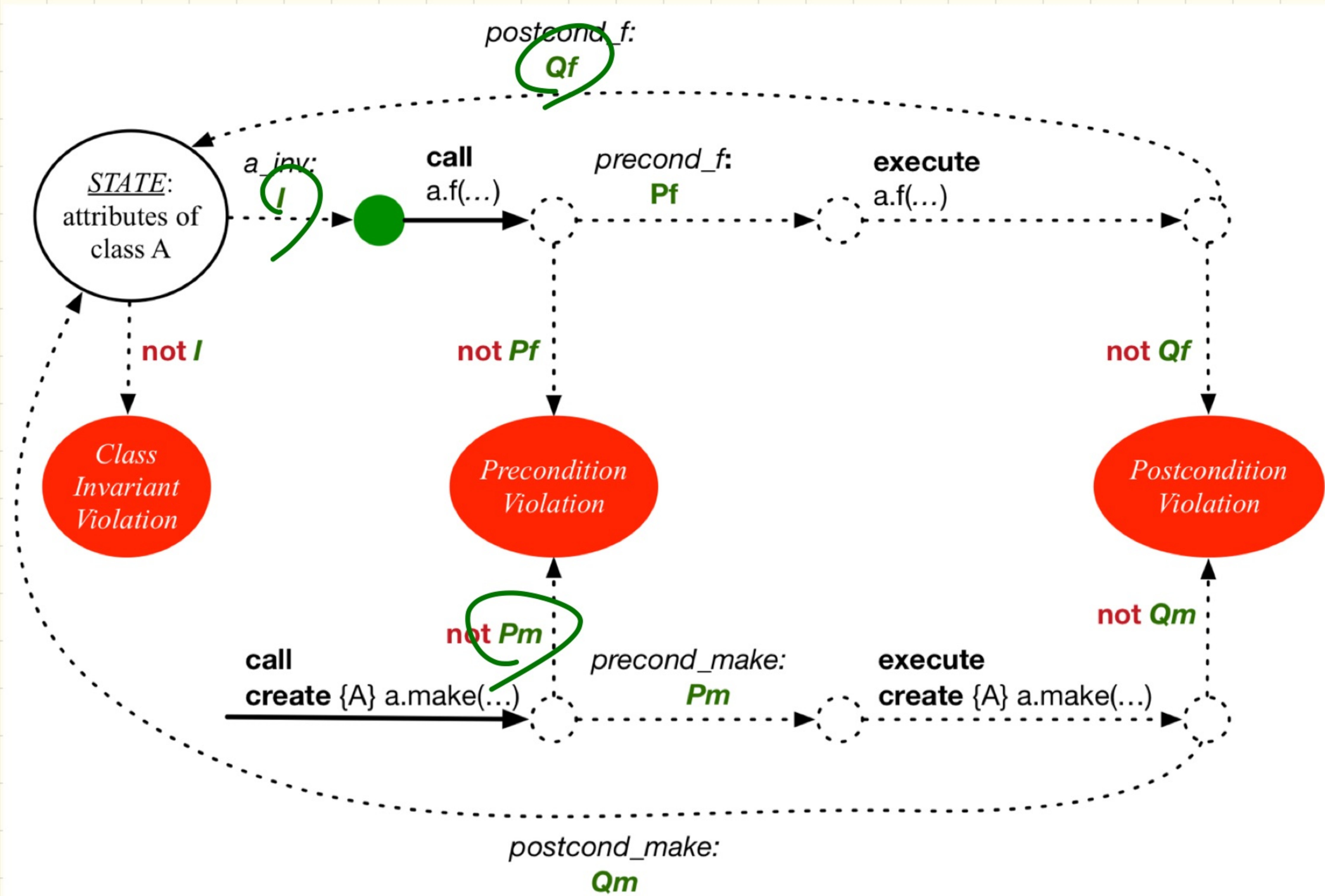
Client

```
class BANK_APP
inherit ARGUMENTS
create make
feature -- Initialization
make
  -- Run application.
local
  jeremy: ACCOUNT
do
  -- Faulty implementation of withdraw in ACCOUNT
  -- balance := balance + amount
  create {ACCOUNT} jeremy.make ("Jeremy", 100)
  jeremy.withdraw(150)
  -- A postcondition violation with tag "balance_deducted"
end
end
```

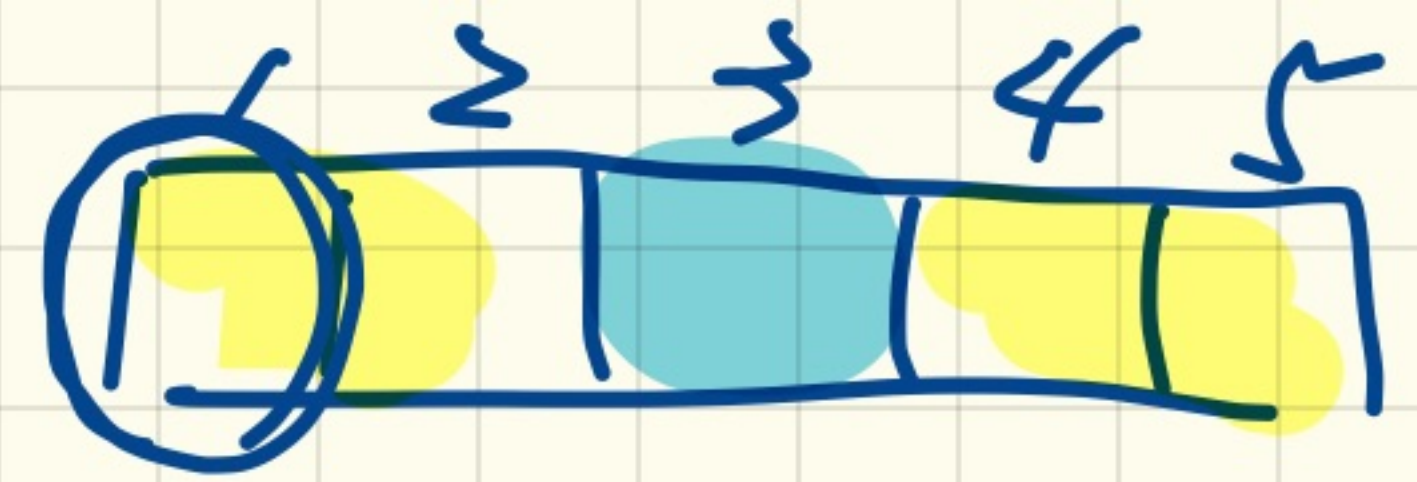
Supplier

```
class ACCOUNT
create
  make
feature -- Attributes
  owner : STRING
  balance : INTEGER
feature -- Constructors
  make(nn: STRING; nb: INTEGER)
    require -- precondition
      positive_balance: nb > 0
    end
feature -- Commands
  withdraw(amount: INTEGER)
    require -- precondition
      non_negative_amount: amount >= 0
      affordable_amount: amount <= balance -- problema
    ensure -- postcondition
      balance_deducted: balance = old balance - amount
    end
invariant -- class invariant
  positive_balance: balance > 0
end
```

Runtime Monitoring of Contracts



Precondition & Postcondition Exercise



→ `change_at (a: ARRAY[STRING]; i: INTEGER; ns; STRING)`
 - - Change index `i` in array `a` to string `ns`

require

??

ensure

??

do → $a[i] := ns$
 $a[i-1] := \text{"junk"}$ } → wrong temp?

→ $a[i] \sim ns$ $F \Rightarrow ? = T$ and $F \Rightarrow ? = T$ or implies

$\forall j \mid 1 \leq j \leq a.Count$ • $j \neq i \Rightarrow \text{old } a[j] \sim a[j]$ implies

j	T	
1	1 = 3	
2	2 = 3	
3	3 = 3	(F)
4		
5		

$\Rightarrow \forall i \mid 1 \leq j \leq \text{a.Count}$ • $\overline{i=j}$ implies

$a[\overline{j}] \sim \underline{\text{old}}$

aCross $\mid 1 \dots \mid \text{a.Count}$ is $1-j$

all

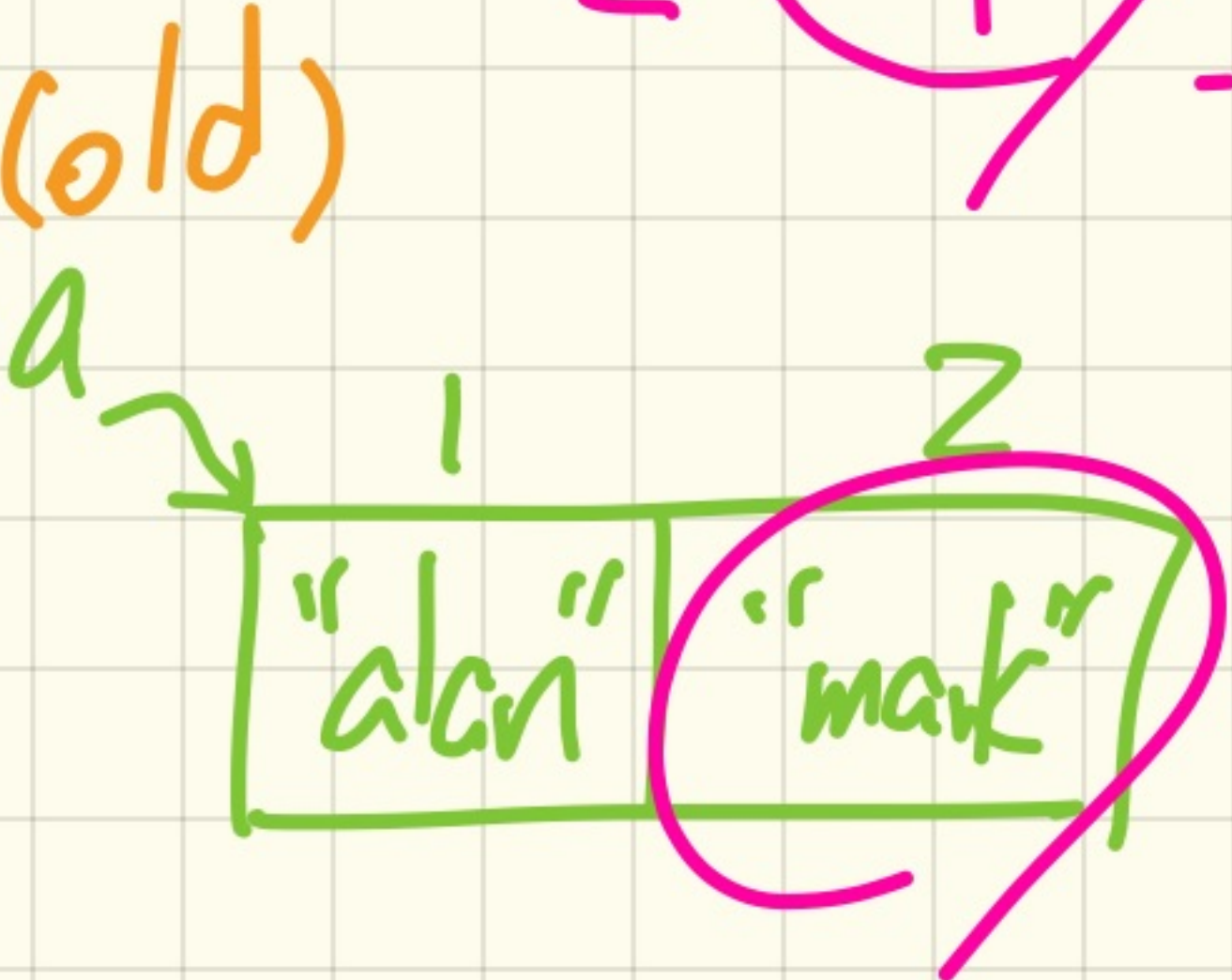
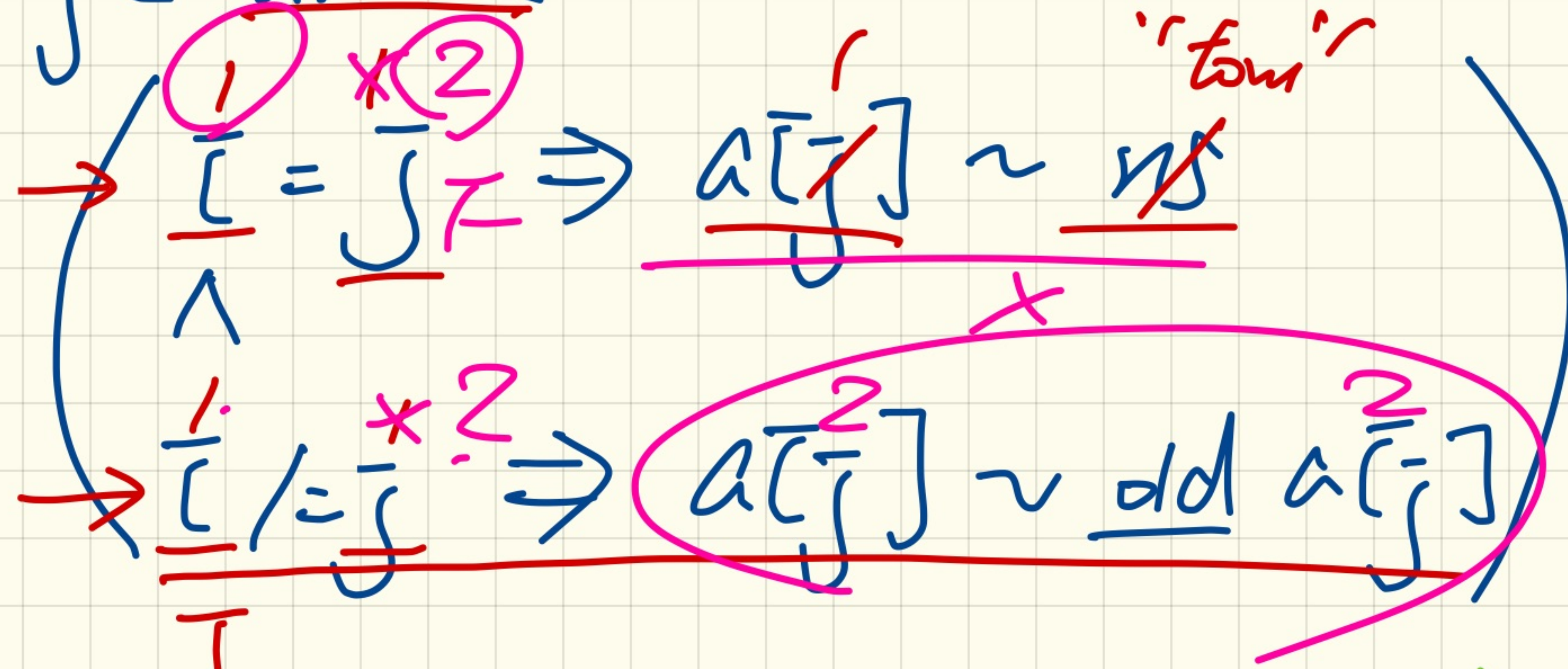
$\overline{i=j}$ implies

$a[\overline{j}] \sim \underline{\text{old}}$ $a[j]$

$a[\overline{j}]$

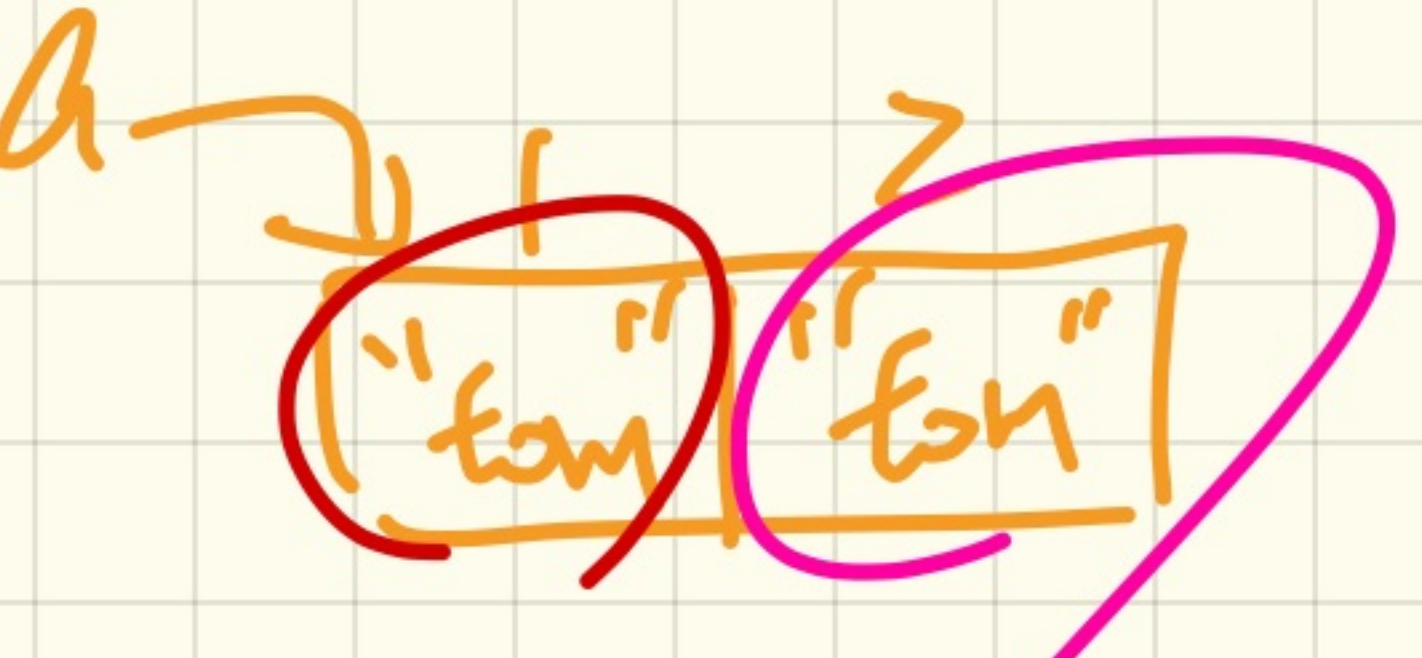
practice: turn this into a single across

$$\forall j \mid 1 \leq j \leq a.\text{count}$$

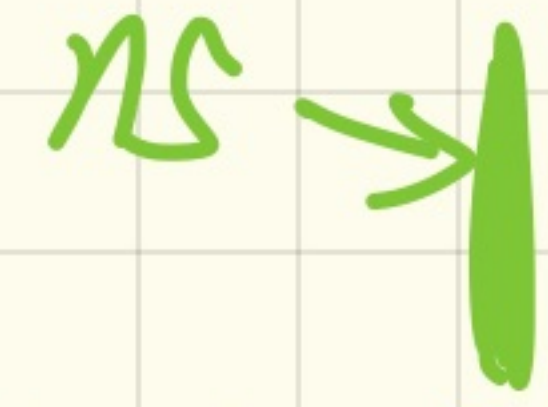
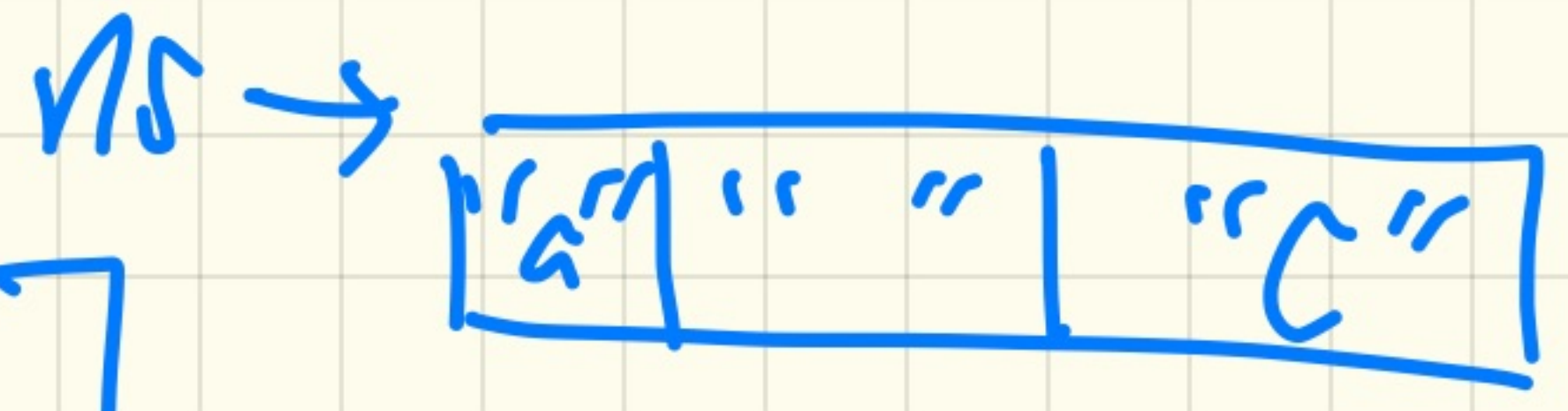


change_at (a, i, "tom")

wrong map: $a[\bar{i}] := \text{NS}$
 $a[\bar{i}+1] := \text{NS}$



NS: ARRAY [STRING]



Are all names in 'NS' non-empty?

V1.

across | 1.. | NS count is

all not NS [c]. is - empty

end

V2

across

NS is 1

all

not 1. is - empty

end

$$\forall x \mid \text{False} \cdot \underline{P(x)} \equiv \text{T}$$

$$\exists x \mid \underline{\text{False}} \cdot P(x) \equiv \text{F}$$

f

require

ensure

(t1: p1

t2: p2

⋮

tn: pn

]

t: p1 ∧ p2 ∧
... ∧ pn

f

ensure

t1: x > y

y > z

f

ensure

t1: x > y and

y > z

$f(\bar{i}: \dots)$

local

$j: \text{INT}$

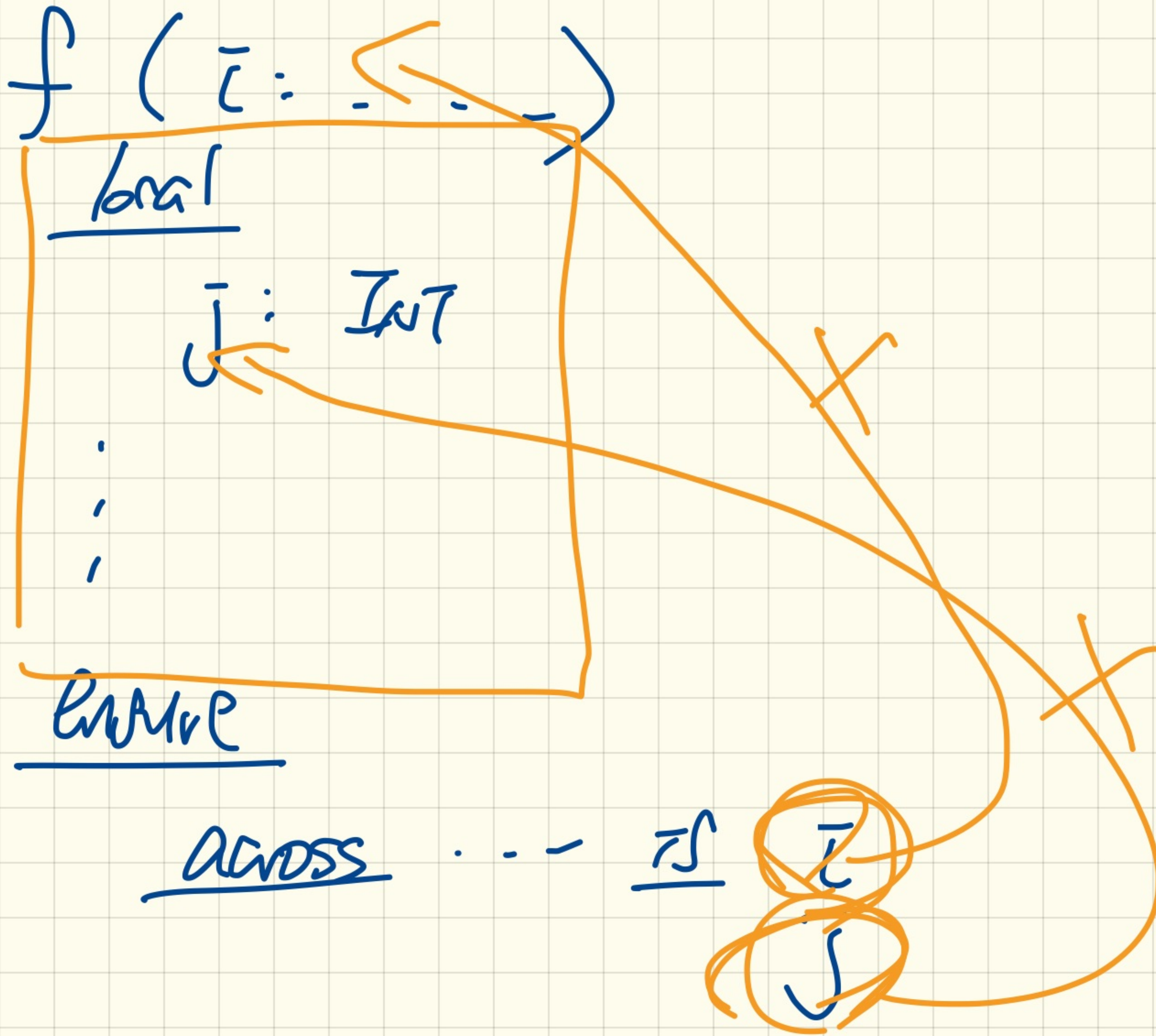
⋮

example

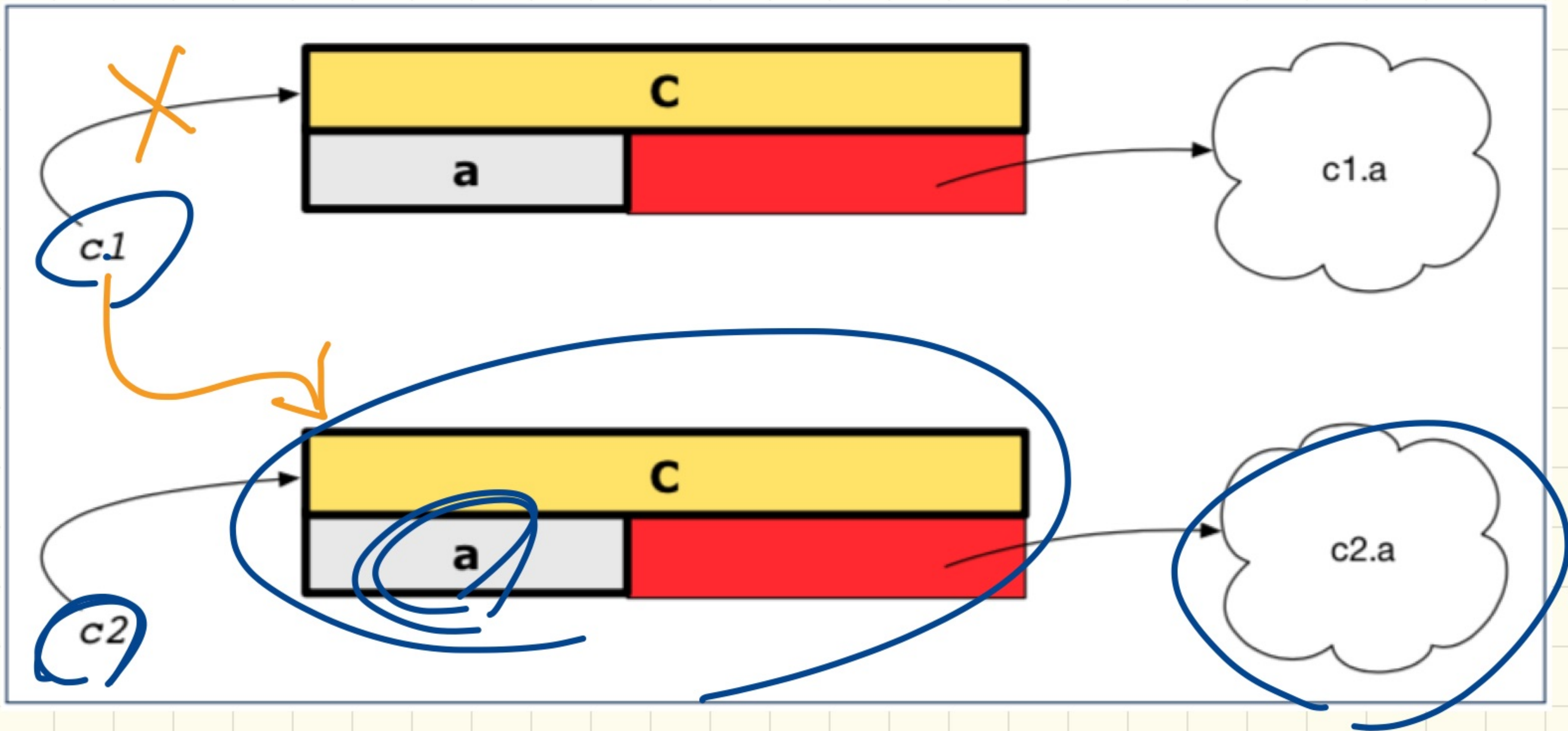
across ...

\rightarrow

\bar{i}
 j



Reference Copy : $C_1 := C_2$

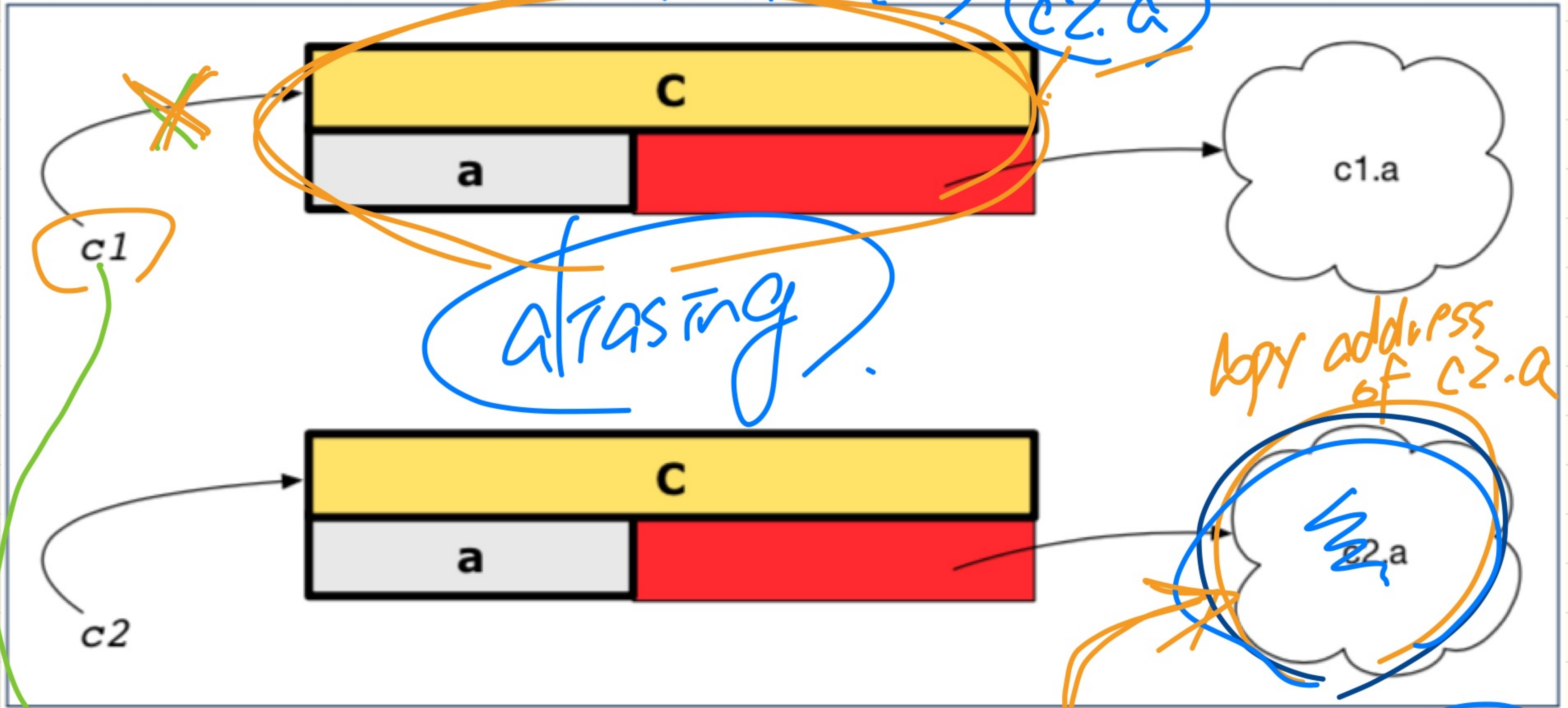


$$c_1 = c_2 \quad \top$$

$$c_1.a = c_2.a \quad \top$$

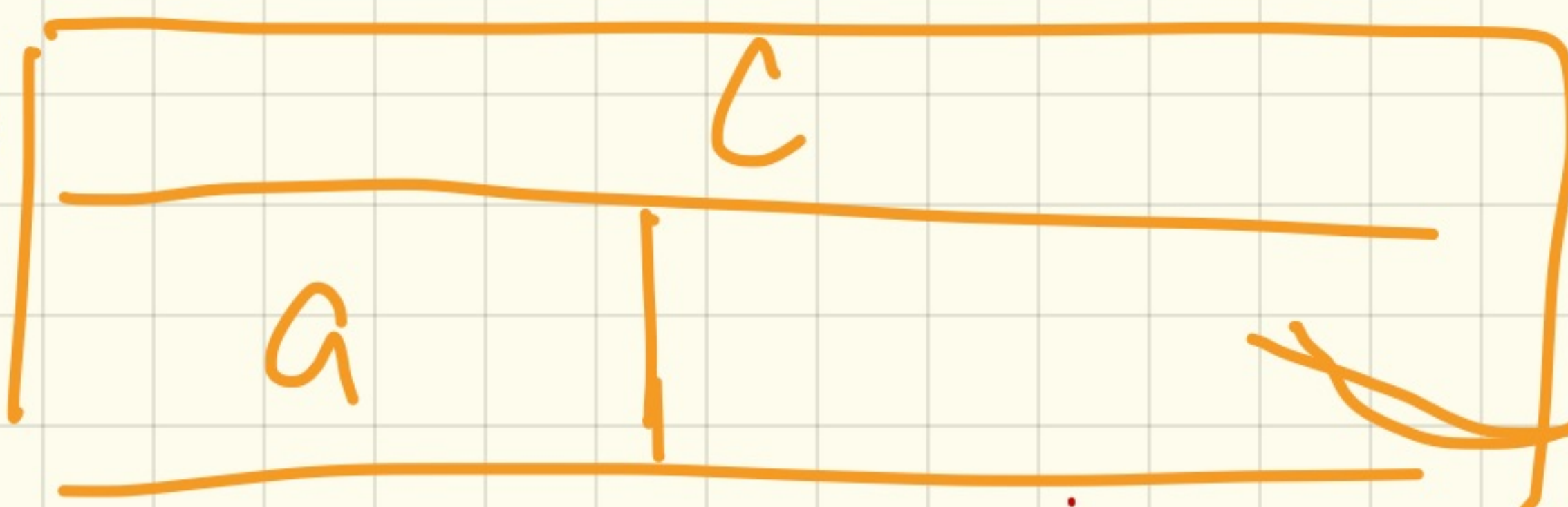
Shallow Copy :

$c1 := c2$ twin
first-level copy
c1.a. ref_val(...)



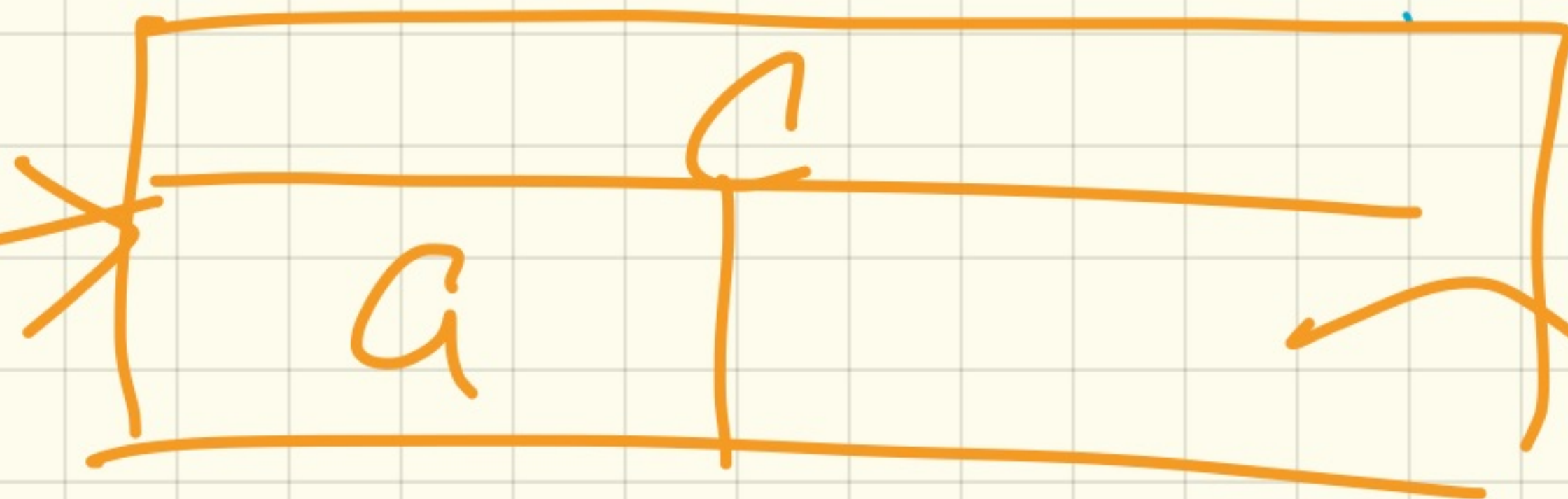
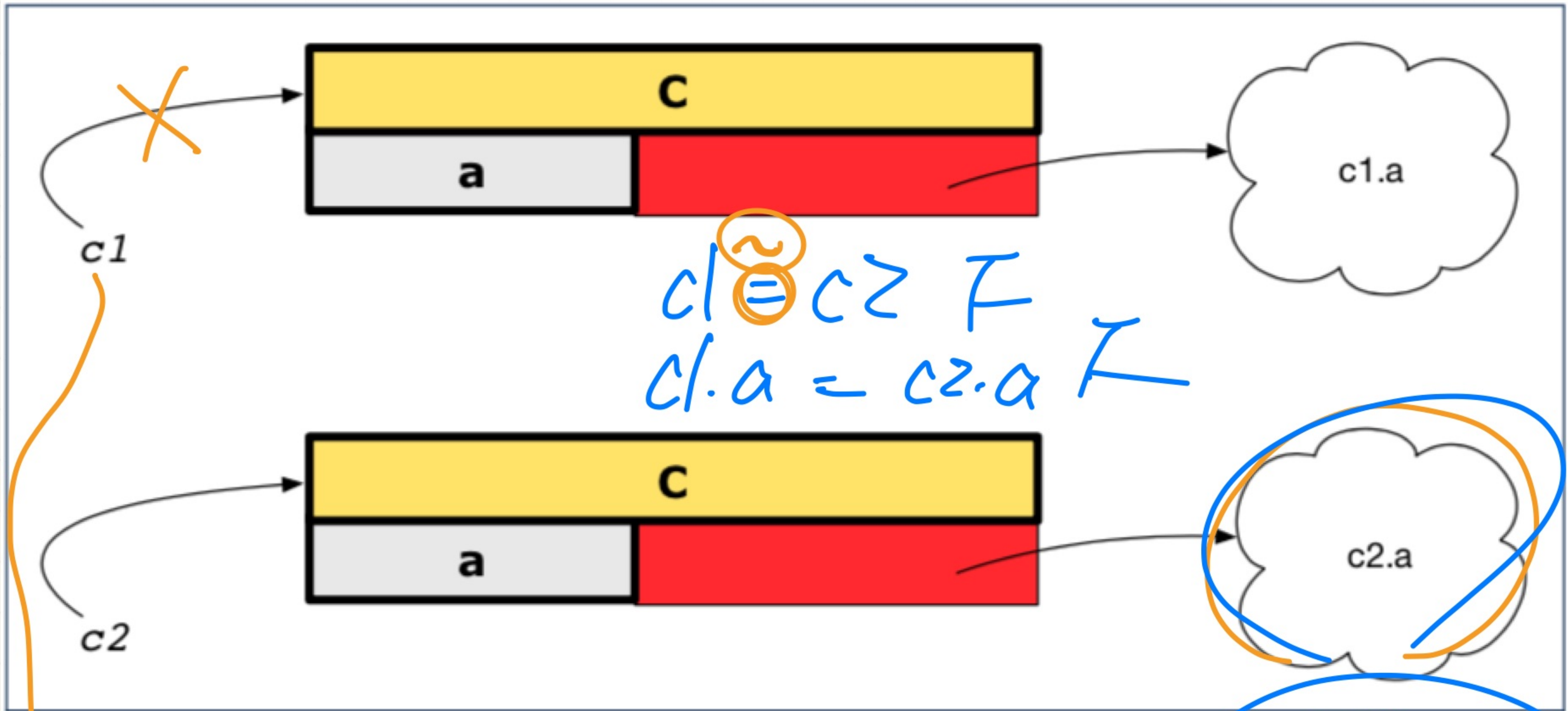
aliasing

copy address of c2.a



$c1 = c2$ F
 $c1.a = c2.a$ T

Deep Copy : $c1 := c2$. deep copy



deep copy of $c2.a$

Ref. vs. Shallow vs. Deep Copies

Initial situation:

Result of:

$b := a$

$c := a.twin$

$d := a.deep_twin$

