LECTURE 3

THURSDAY SEPTEMBER 12

- In-Lab Demo yesterday (source code, recording)
- 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, WithdrawAmountTooLargeExceptio
    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
```

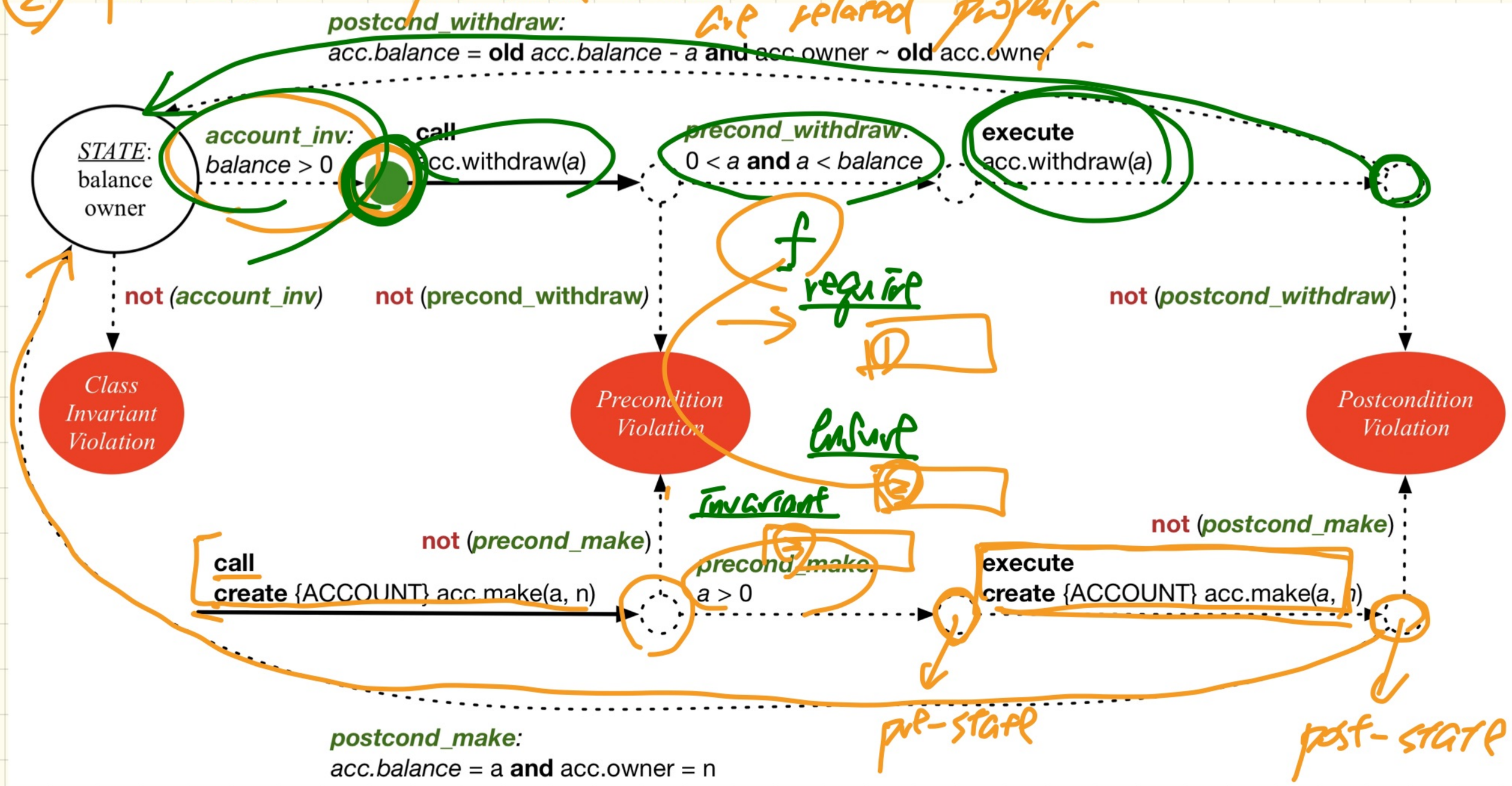
```
class ACCOUNT
create
	make
feature -- Attributes
	owner : STRING
	balance : INTEGER
feature -- Constructors
	make(nn: STRING; nb: INTEGER)
		require -- precondition
			positive_balance: nb > 0
		end
feature -- Commands
	withdraw(amount: INTEGER)
		require -- precondition
			non_negative_amount: amount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

tag

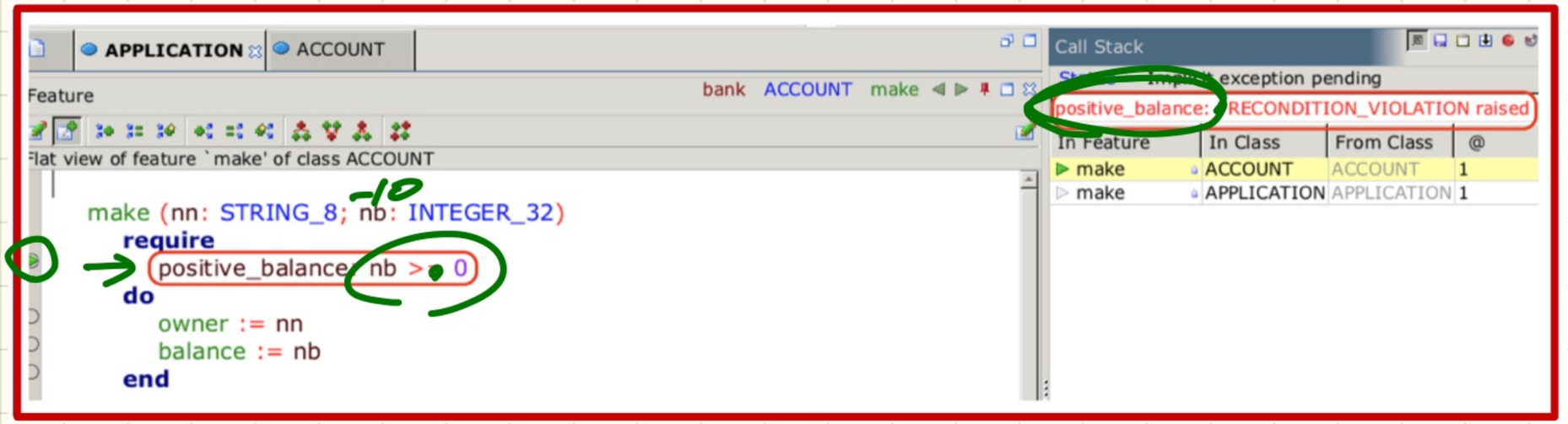
Contract View

Runtime Monitoring of Contracts

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



Precondition Violation: positive_balance



Supplier

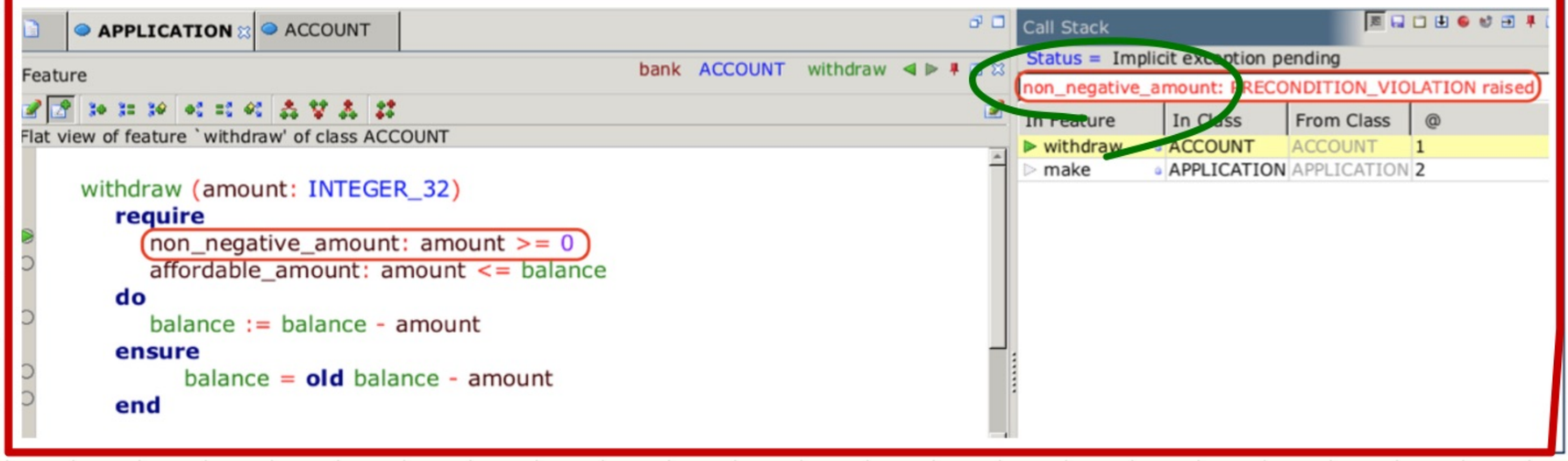
```
class ACCOUNT
create
        make
feature -- Attributes
        owner : STRING
        balance : INTEGER
feature -- Constructors
        make(nn: STRING; nb: INTEGER)
                require -- precondition
                        positive_balance: nb > 0
                end
feature -- Commands
        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
```

Client

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

Precondition Violation:

non_negative_amount



Supplier

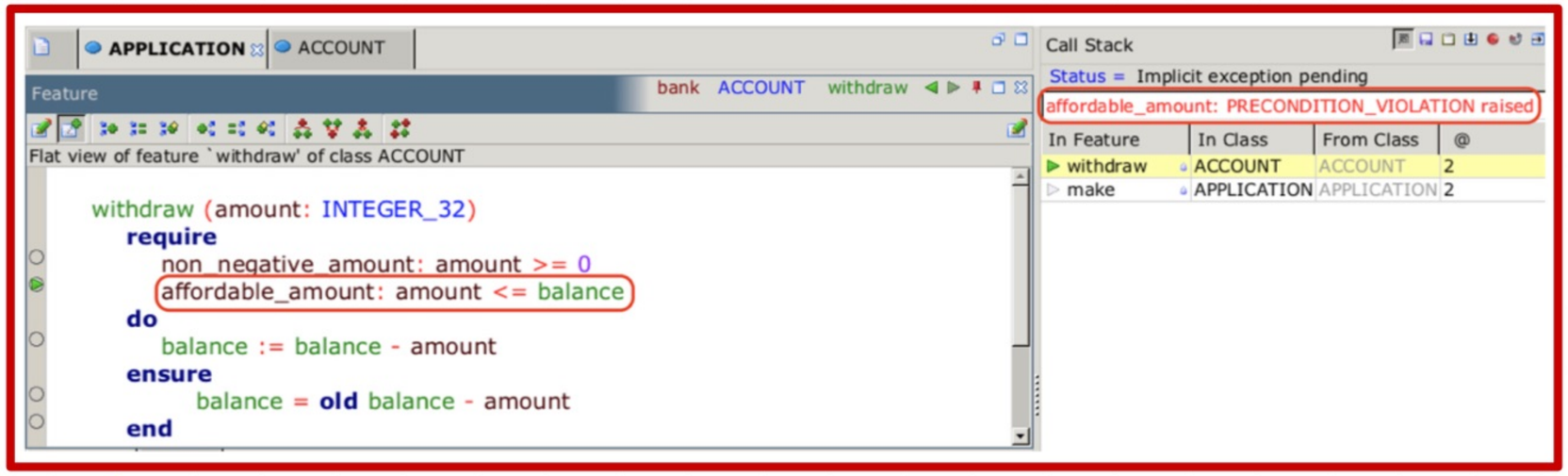
```
class ACCOUNT
create
	make
feature -- Attributes
	owner : STRING
	balance : INTEGER
feature -- Constructors
	make(nn: STRING; nb: INTEGER)
		require -- precondition
			positive_balance: nb > 0
		end
feature -- Commands
	withdraw(amount: INTEGER)
		require -- precondition
			non_negative_amount: amount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

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
```

Precondition Violation:

affordable_amount



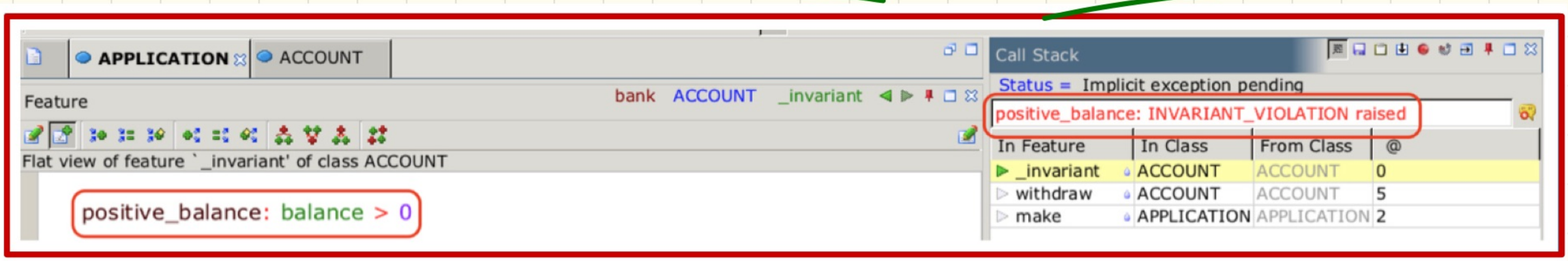
Supplier

```
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
```

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 Invariant Violation: positive_balance



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
```

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
```

Postcondition Violation: balance_deducted

APPLICATION | ACCOUNT

Feature — bank ACCOUNT withdraw

Flat view of feature `withdraw' of class ACCOUNT

```
        affordable_amount: amount <= balance
    do
        balance := balance + amount
    ensure
        balance_deducted: balance = old balance - amount
    end
```

Call Stack

Status = Implicit exception pending

balance_deducted: POSTCONDITION_VIOLATION raised

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

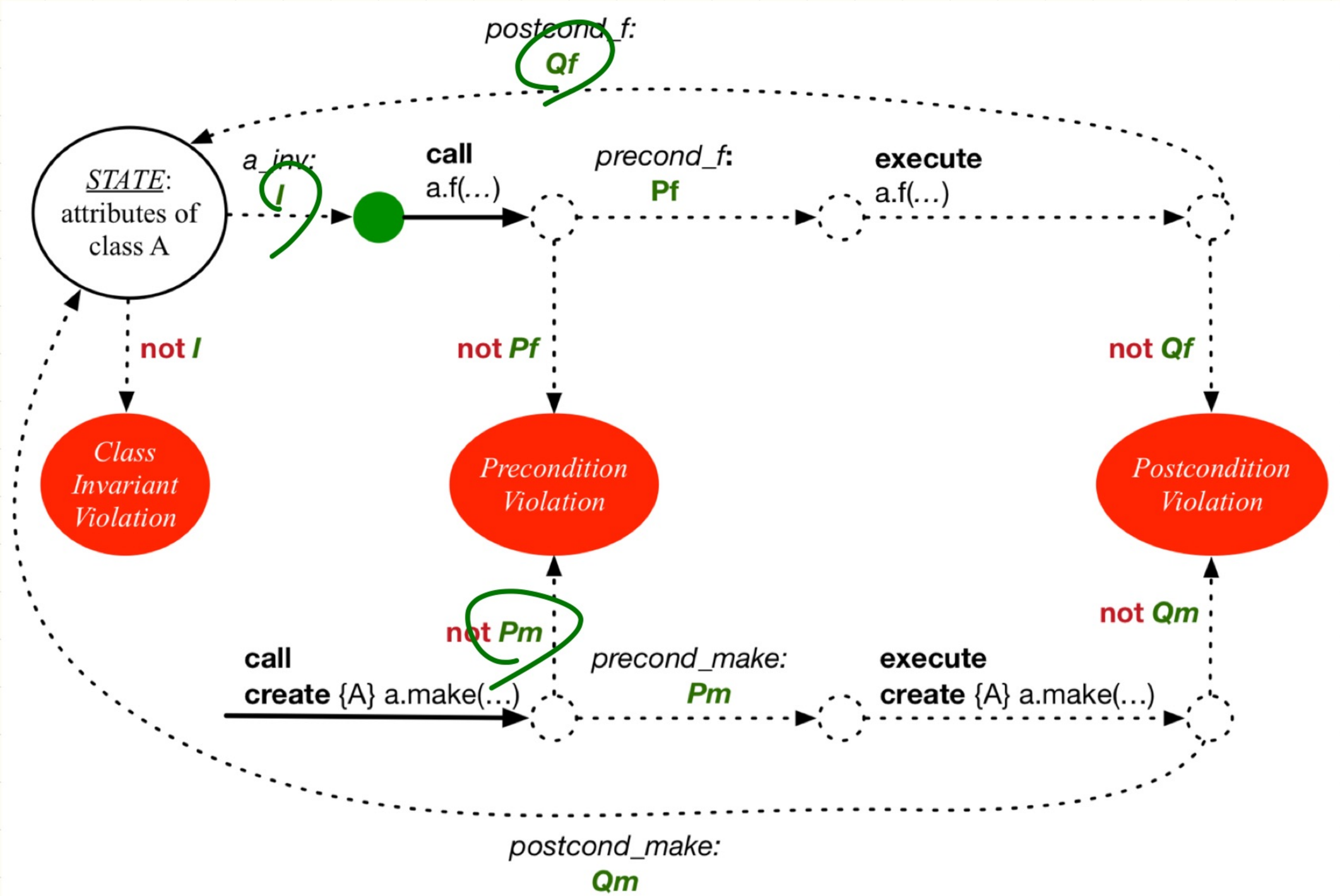
Supplier

```
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
```

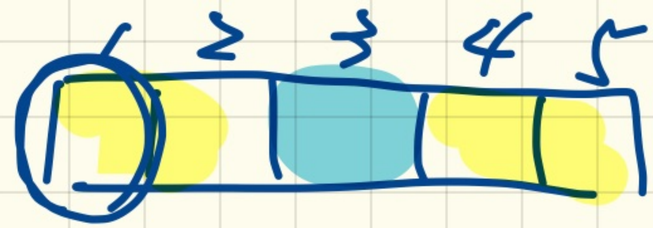
Client

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

Runtime Monitoring of Contracts



Precondition & Postcondition Exercise



3

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] := "junk"] → wrong imp?

⇒ a[i] ~ ns

F ? ? = T and

F ⇒ ? = T or tuples

?

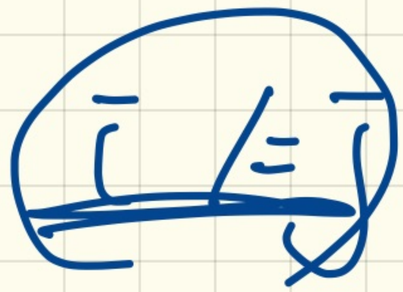
tuples

∀ j | 1 ≤ j ≤ a.count • j /= i and old a[j] ~ a[j]

j	T
1	1 /= 3
2	2 /= 3
3	3 /= 3
4	
5	

T

F

$\Rightarrow \forall i \mid 1 \le i \le j \le a.count \cdot$  $\Rightarrow$ implies

$a[i] \sim$ old $a[i]$

across $1 \mid .. \mid a.count$ is $i..j$

all

$i \neq i..j$ implies

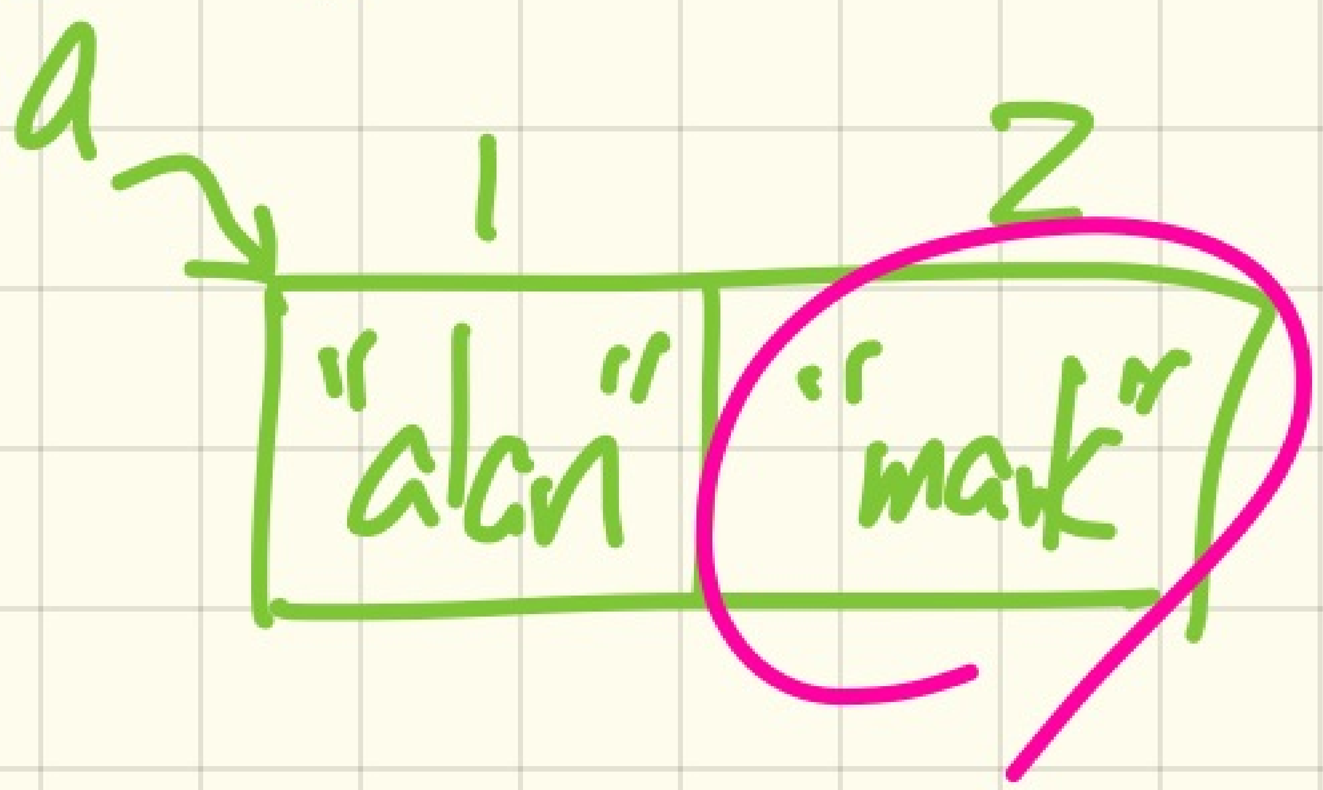
$a[i] \sim$ old $a[i]$

practice: turn this into a single across

$$\forall j \mid 1 \le j \le a.count \bullet \left((i = j \Rightarrow a[j] \sim ns) \land (i /= j \Rightarrow a[j] \sim old\ a[j]) \right)$$

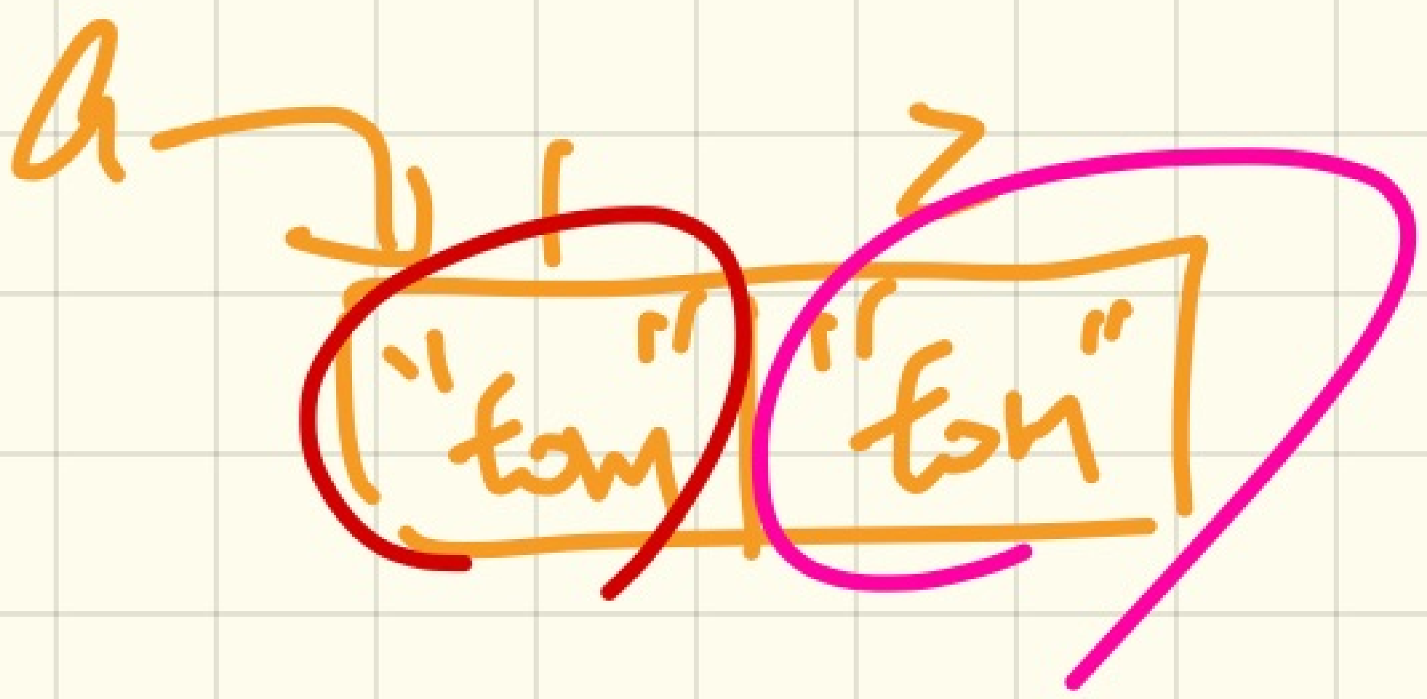
i	j	
	1	✓
	2	F

(old)

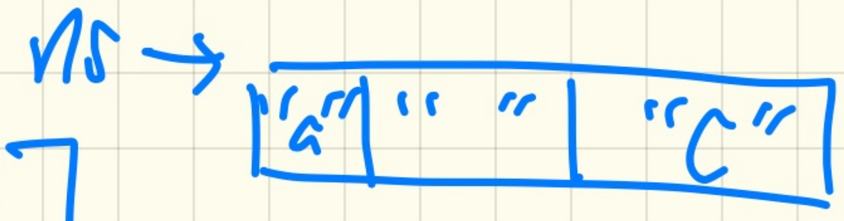


change_at (a, i, "tom")

wrong imp: $a[i] := ns$
$a[i+1] := ns$



ns : ARRAY [STRING]



Are all names in 'ns' non-empty?

V1

across 1 |..| ns.count is i

all

not ns[i].is_empty

end

V2

across ns is n

all

not n.is_empty

end

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

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

f

ensure

t1: x > y

y > z

f

require

ensure

f

ensure

t1: x>y and

y>z

t1: p1

t2: p2

tn: pn

t: p1 ∧ p2 ∧ ... ∧ pn

f (i : _ _ _)

local

j : INT

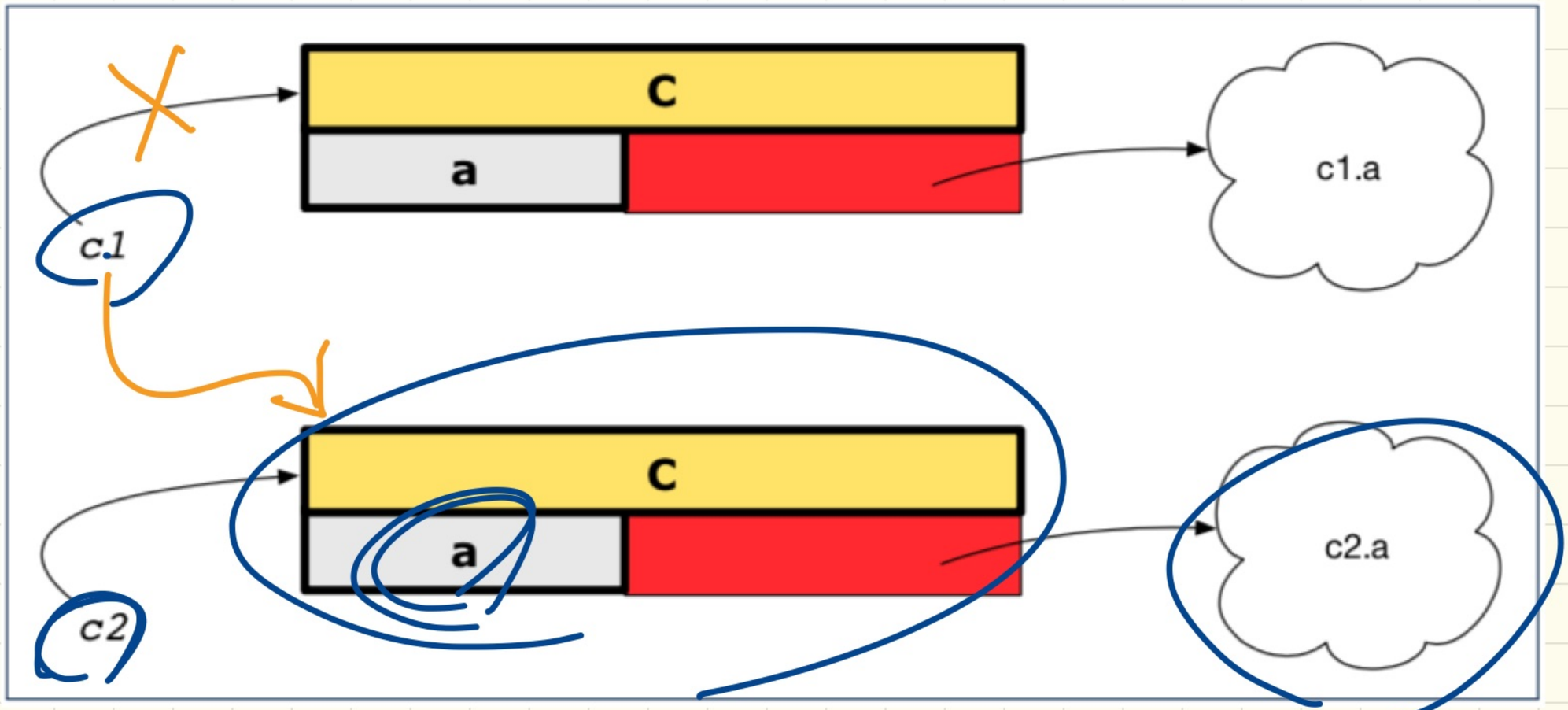
:

:

ENDIF

across ... is i

Reference Copy : $c_1 := c_2$



$c1 = c2$ T

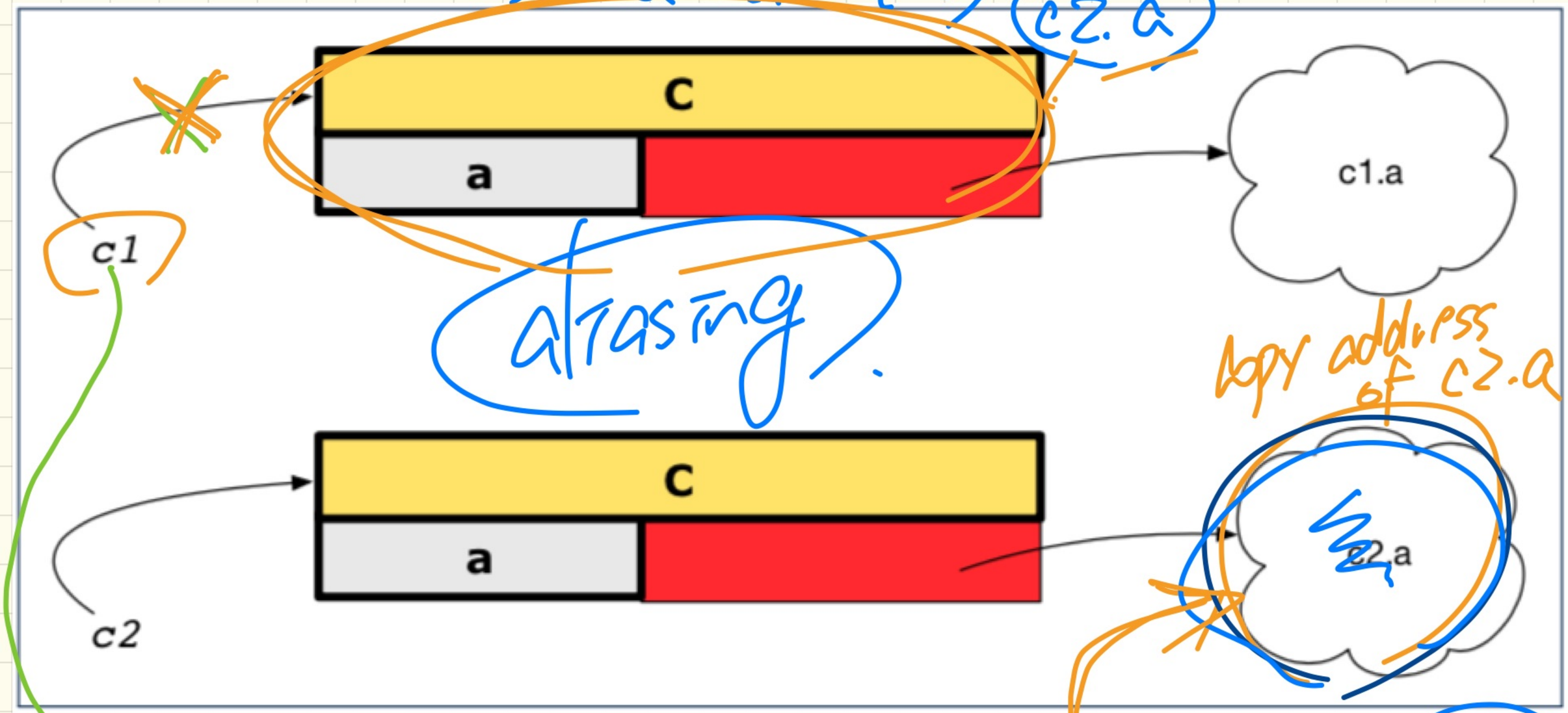
$c1.a = c2.a$ T

Shallow Copy : c1 := c2.twin

↳ c1.a.set_val(...)

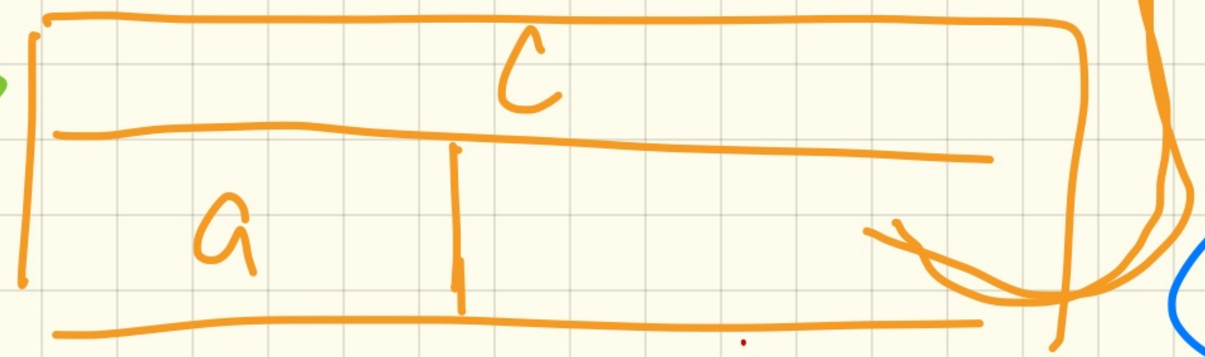
first-level copy

c2.a



aliasing

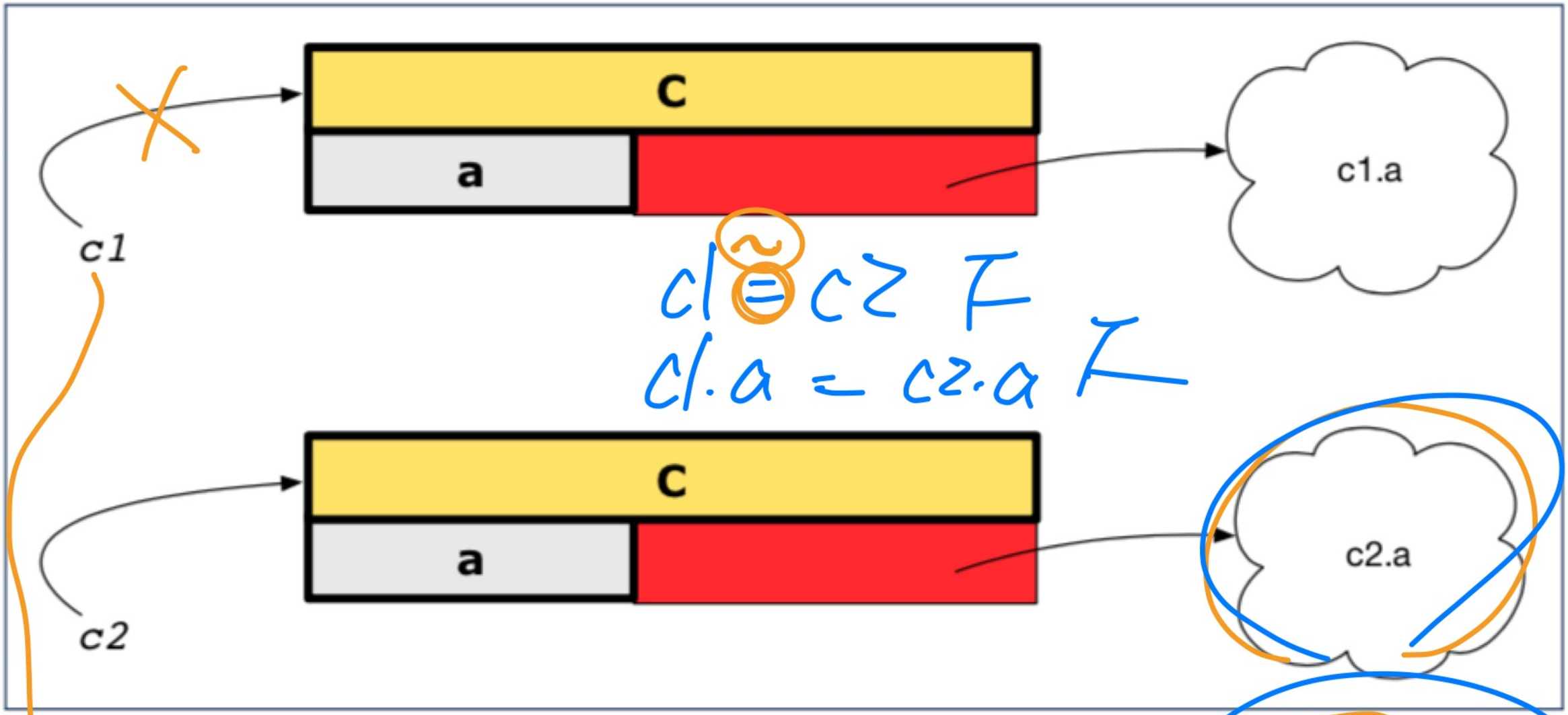
copy address of c2.a

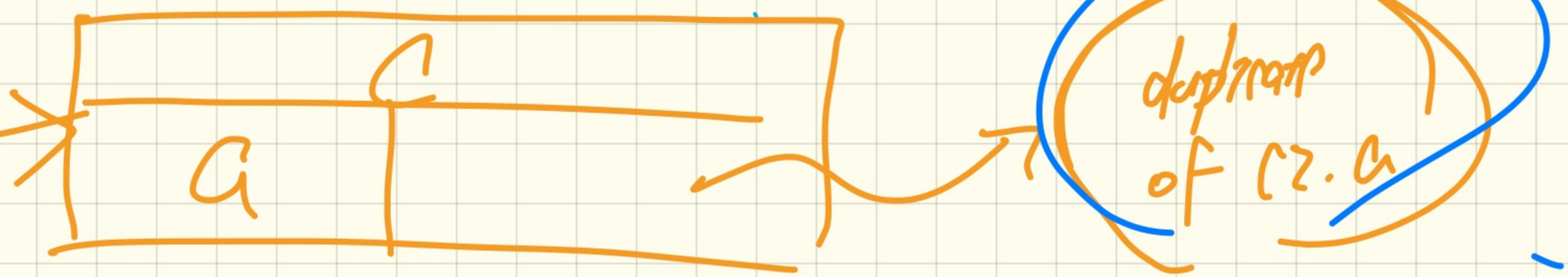


c1 = c2 F

c1.a = c2.a T

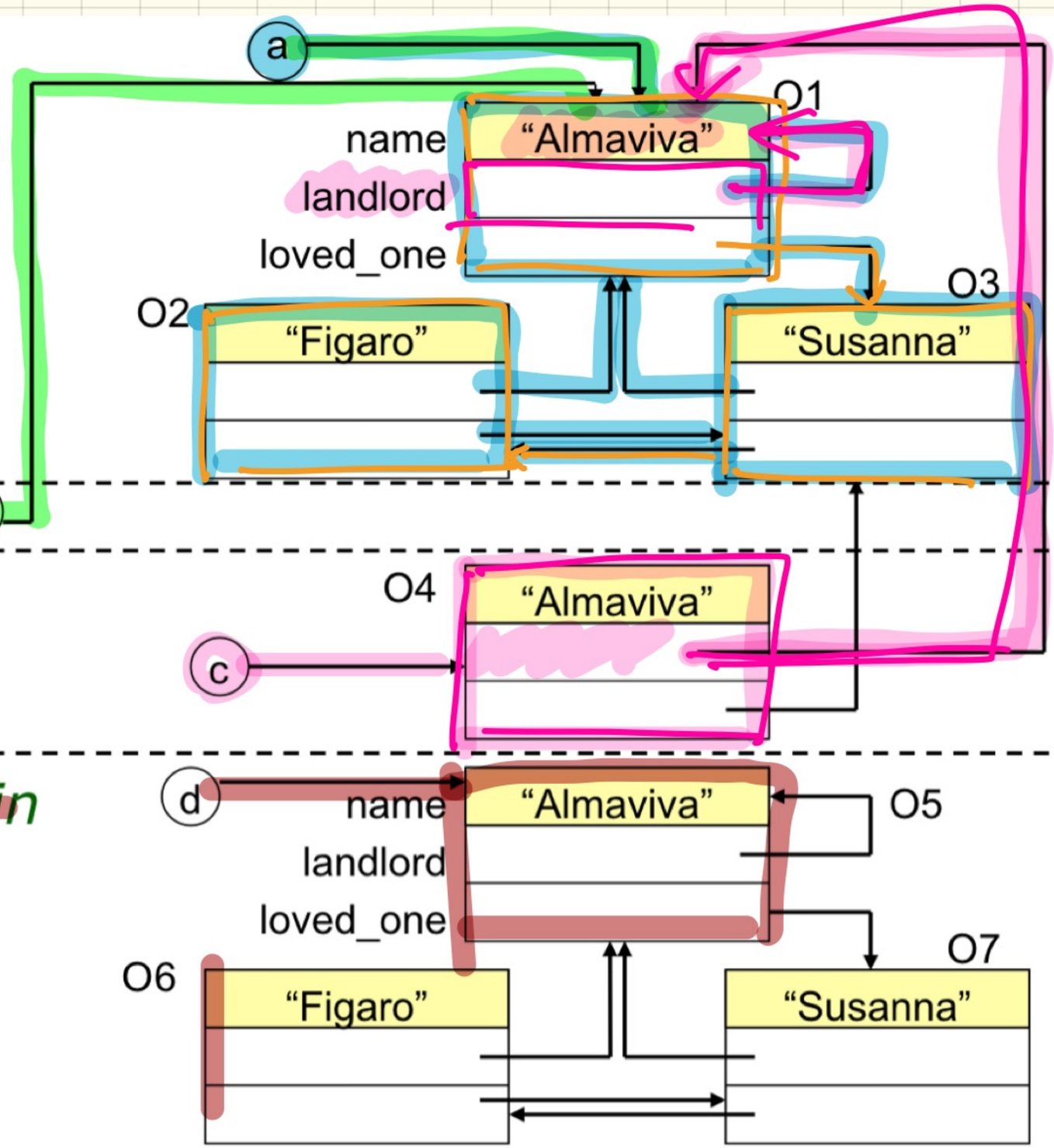
Deep Copy : c1 := c2.deep_twin





Ref. vs. Shallow vs. Deep Copies

- Initial situation:



- Result of:

b := a

c := a.twin

d := a.deep_twin