LECTURE 5

THURSDAY SEPTEMBER 19

Caching Values for old Expressions in Postconditions

ensure (in context of BANK)

① old accounts[i].id

② (old accounts[i]).id

③ (old accounts[i].twin).id

④ (old accounts)[i].id

⑤ (old accounts.twin)[i].id

⑥ (old Current).accounts[i].id

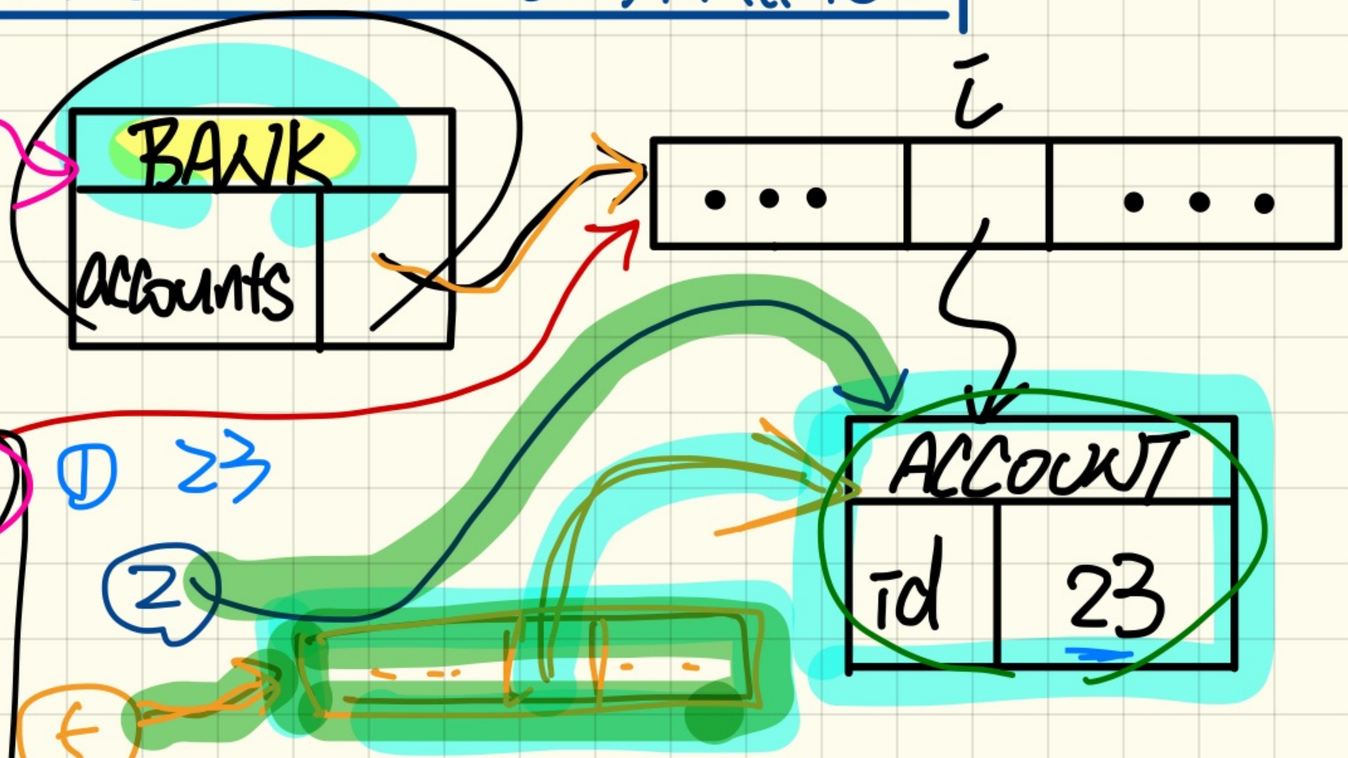
⑦ (old Current.twin).accounts[i].id

INT

shallow copy of account

class BANK
...
ensure
...
end

How to Cache at Runtime?



⑦ → BANK, accs → shallow copy of array

① := accounts[i].id

② := accounts[i]

⑤ := accounts.twin

⑦ := Current.twin

⑥ := Current

```
class BANK
create make
feature
  accounts: ARRAY[ACCOUNT]
  make do create accounts.make_empty end
  account_of (n: STRING): ACCOUNT
    require -- the input name exists
      existing: across accounts is acc some acc.owner ~ n end
        -- not (across accounts is acc all acc.owner /~ n end)
    do ... ensure Result.owner ~ n end
  add (n: STRING)
    require -- the input name does not exist
      non_existing: across accounts is acc all acc.owner /~ n end
        -- not (across accounts is acc some acc.owner ~ n end)
    local new_account: ACCOUNT
    do
      create new_account.make (n)
      accounts.force (new_account, accounts.upper + 1)
    end
end
```

not(across accounts is acc all acc.owner /~ n end)

```
class
  ACCOUNT

inherit
  ANY
    redefine is_equal end

create
  make

feature -- Attributes
  owner: STRING
  balance: INTEGER

feature -- Commands
  make (n: STRING)
    do
      owner := n
      balance := 0
    end

  deposit(a: INTEGER)
    do
      balance := balance + a
    ensure
      balance = old balance + a
    end

  is_equal(other: ACCOUNT): BOOLEAN
    do
      Result :=
            owner ~ other.owner
        and balance = other.balance
    end
end
```

$$\forall x : R(x) \bullet P(x)$$

$$\equiv \neg(\exists x : R(x) \bullet \neg P(x))$$

f

across

require

tag: -ToDo

g(...)

g(...)

f

ensure

tag: Todo

h(...)

h(----)

do

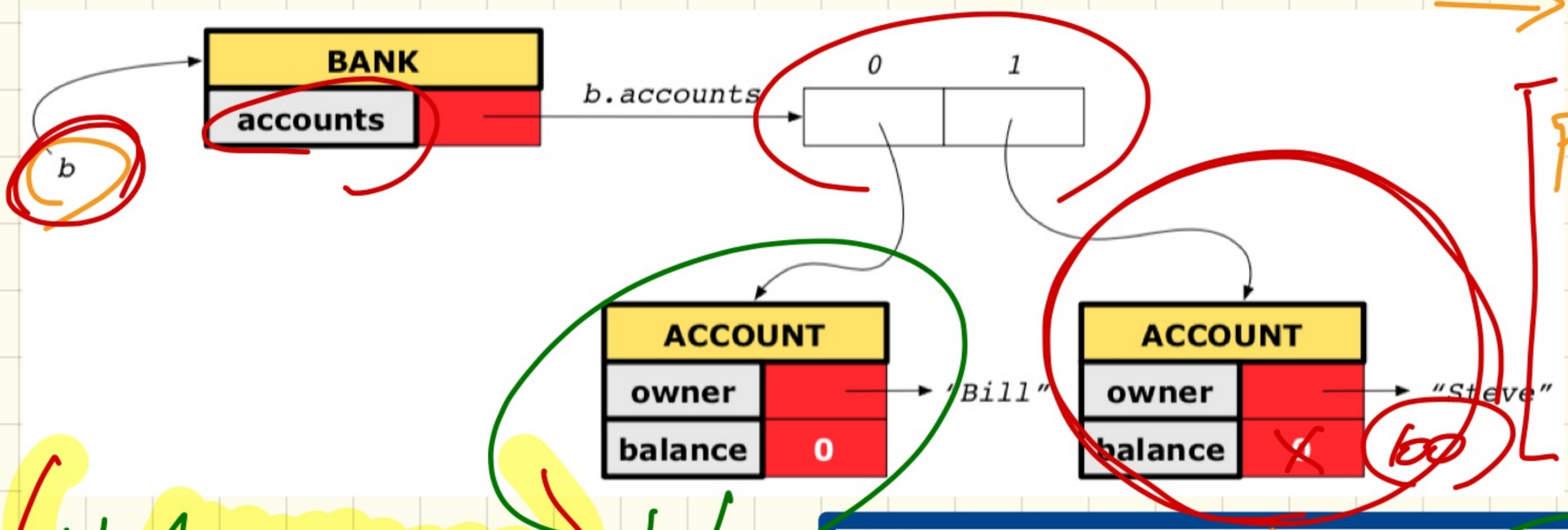
end

Unit Test for All 5 Versions

```
class TEST_BANK
  test_bank_deposit_correct_imp_incomplete_contract: BOOLEAN
    local
      b: BANK
    do
      comment("t1: correct imp and incomplete contract")
      create b.make
      b.add ("Bill")
      b.add ("Steve")

      -- deposit 100 dollars to Steve's account
      b.deposit_on_v1 ("Steve", 100)
      Result :=
            b.account_of("Bill").balance = 0
        and b.account_of("Steve").balance = 100
      check Result end
  end
end
```

Version 1: Incomplete Contracts, Correct Implementation



b.deposit("Steve", 100)

pre-stmt: 2

old_count := accounts.count

old_balance := 0

Current.a_o(n).b

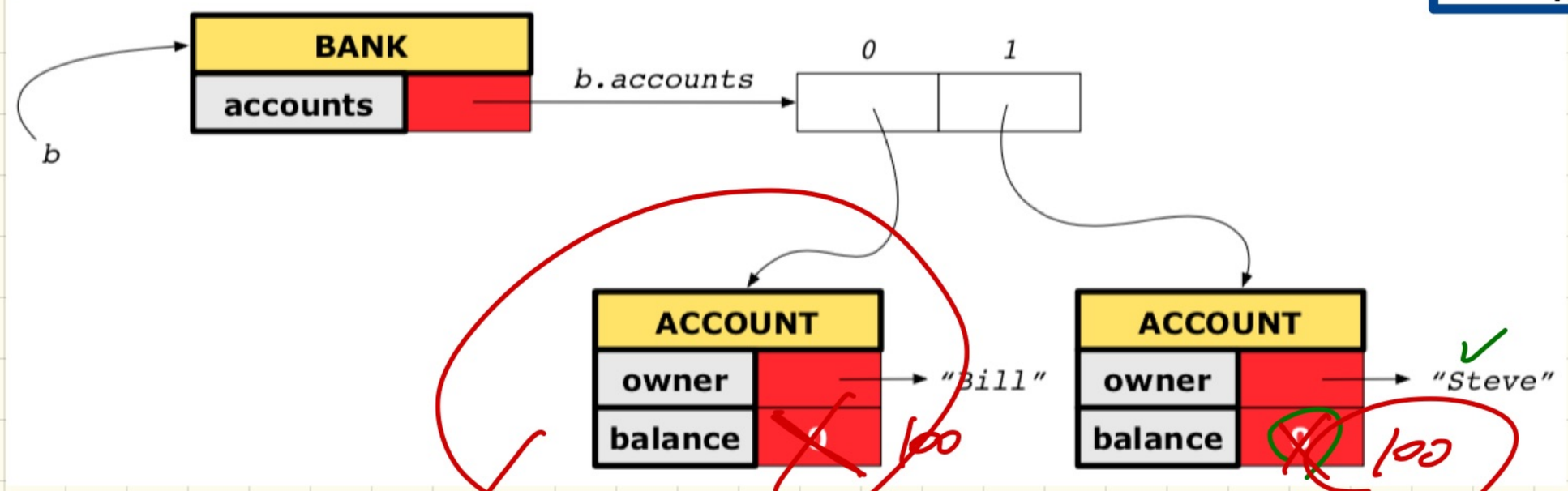
old Current.a_o(n).balance

Incomplete :(It talks nothing about all other accounts.

```
class BANK
  deposit_on_v1 (n: STRING; a: INTEGER)
    require across accounts is acc some acc.owner ~ n end
    local i: INTEGER
    do
      from i := accounts.lower
      until i > accounts.upper
      loop
        if accounts[i].owner ~ n then accounts[i].deposit(a) end
        i := i + 1
      end
    ensure
      num_of_accounts_unchanged:
        accounts.count = old accounts.count
      balance_of_n_increased:
        Current.account_of(n).balance =
          old Current.account_of(n).balance + a
    end
end
```

Version 2: Incomplete Contracts, Wrong Implementation

b.deposit("Steve", 100)

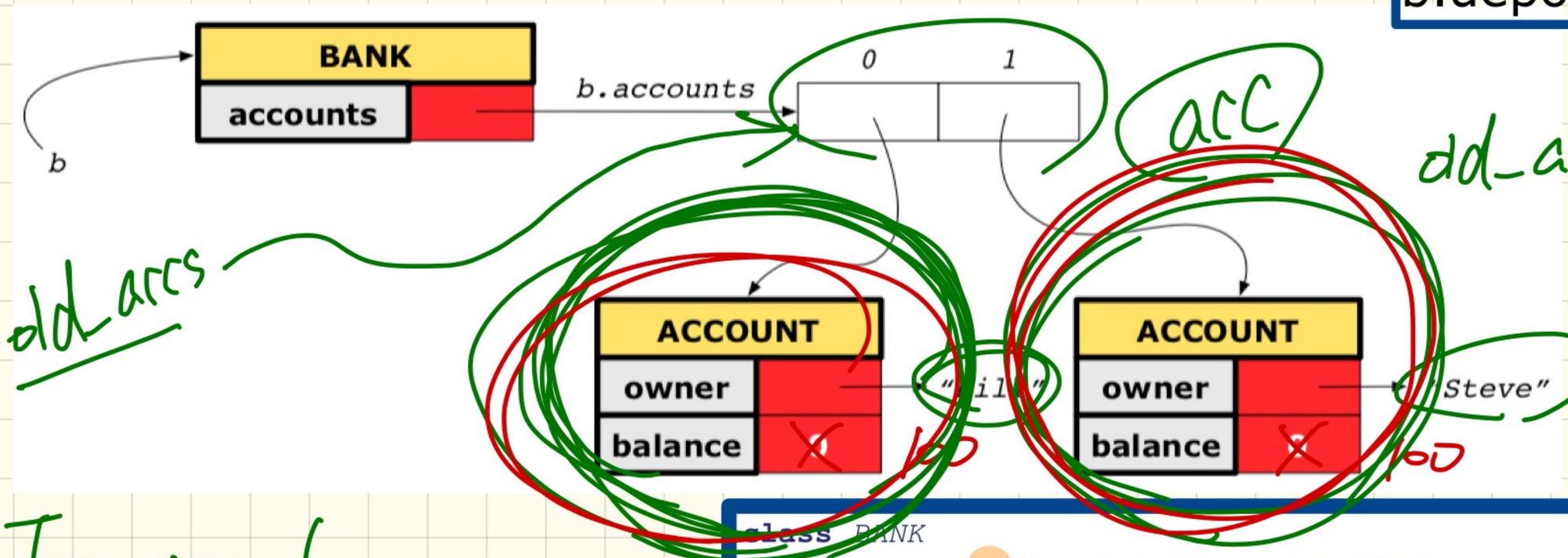



```
class BANK
  deposit_on_v2 (n: STRING; a: INTEGER)
    require across accounts is acc some acc.owner ~ n end
    local i: INTEGER
    do ...
      -- imp. of version 1, followed by a deposit into 1st account
      accounts[accounts.lower].deposit(a)
    ensure
      num_of_accounts_unchanged:
        accounts.count = old accounts.count
      balance_of_n_increased:
        Current.account_of(n).balance =
          old Current.account_of(n).balance + a
    end
end
```

how this object (ref of accounts) should be changed (not) addressed in postcond.

Version 3: Complete Contracts (Ref. Copy), Correct Implementation

b.deposit("Steve", 100)



Iteration 1

"Bill" /~ "Steve" implies

↳ true simply because we considered the objects equal to themself.

Iteration 2

T F

"Steve" /~ "Steve" implies

```
class BANK
  deposit_on_v3 (n: STRING; a: INTEGER)
    require across accounts is acc some acc.owner ~ n end
  local i: INTEGER
  do ...
    -- imp. of version 1, followed by a deposit into 1st account
    accounts[accounts.lower].deposit(a)
  ensure
    num_of_accounts_unchanged: accounts.count = old accounts.count
    balance_of_n_increased:
      Current.account_of(n).balance =
        old Current.account_of(n).balance + a
    others_unchanged:
      across old accounts is acc
      all
        acc.owner /~ n implies acc ~ Current.account_of(acc.owner)
      end
  end
end
```

2 iterations

"Bill" Iteration

Use of across in Postcondition

Version 1

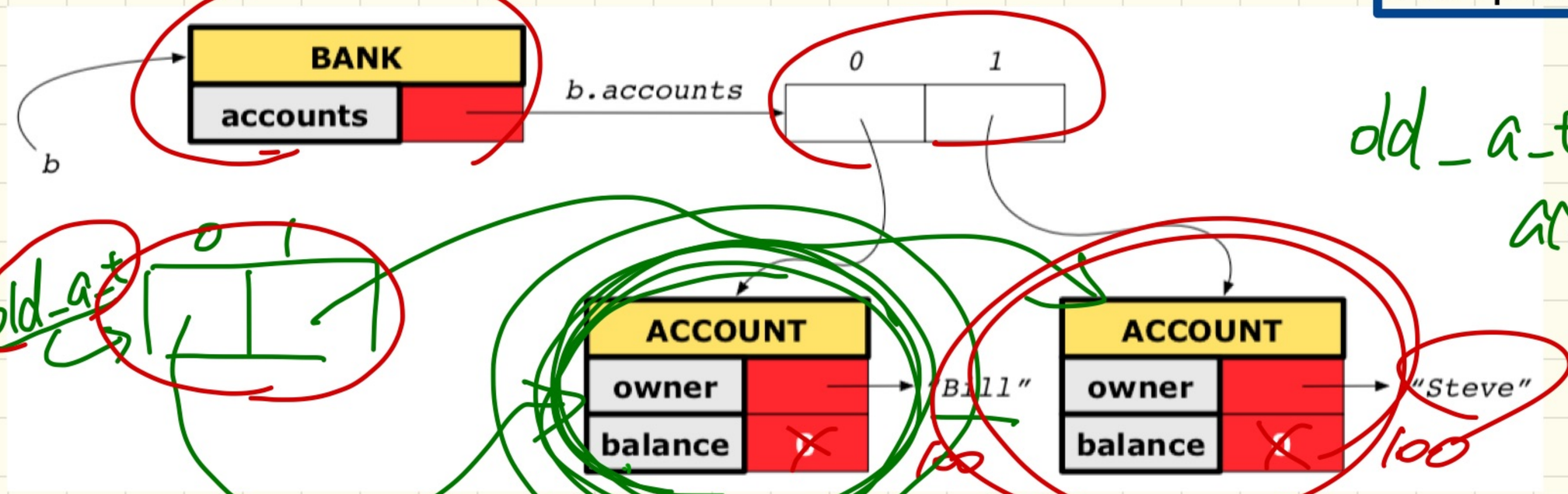
```
across old accounts is acc
all
   acc.owner /~ n
   implies
   acc ~ Current.account_of (acc.owner)
end
```

Version 2

```
across (old accounts.lower |..| old accounts.upper) is i
all
   (old accounts)[i].owner /~ n
   implies
   (old accounts)[i] ~ Current.account_of ( (old accounts)[i].owner )
end
```

Version 4: Complete Contracts (Shallow Copy), Correct Implementation

b.deposit("Steve", 100)



1st iteration

"Bill" /~ "Steve" ⇒

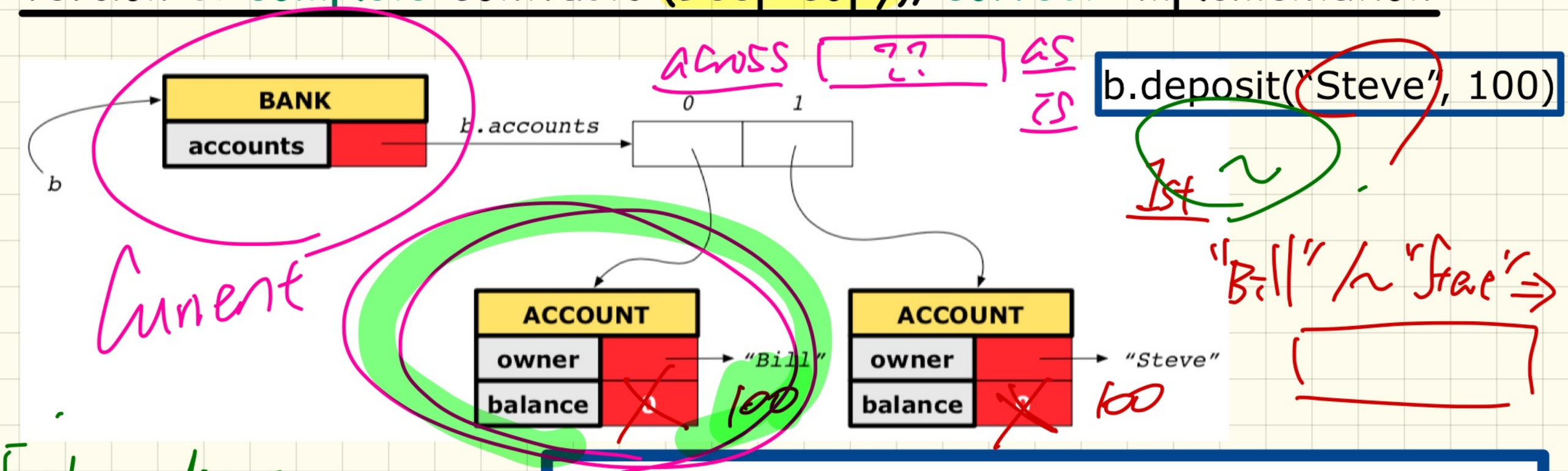
2nd iteration ✓

"Steve" /~ "Steve" ⇒

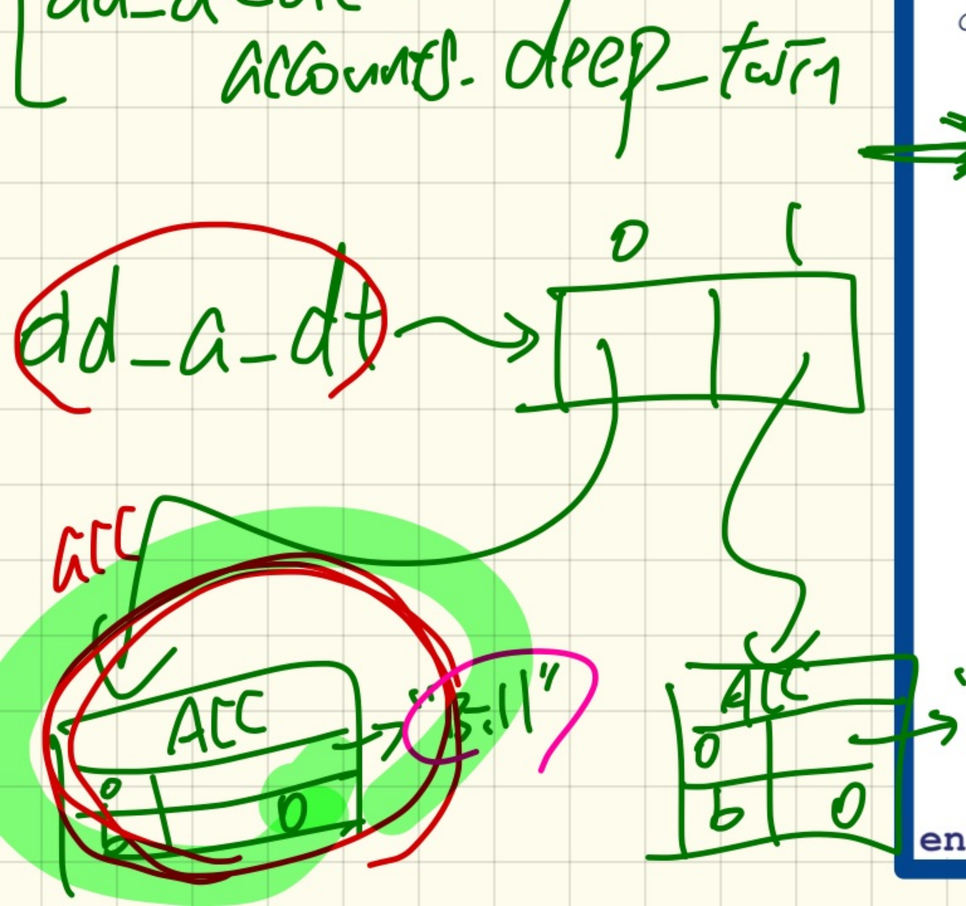
```
class BANK
  deposit_on_v4 (n: STRING; a: INTEGER)
    require across accounts is acc some acc.owner ~ n end
    local i: INTEGER
    do ...
      -- imp. of version 1, followed by a deposit into 1st account
      accounts[accounts.lower].deposit(a)
    ensure
      num_of_accounts_unchanged: accounts.count = old accounts.count
      balance_of_n_increased:
        Current.account_of(n).balance =
          old Current.account_of(n).balance + a
      others_unchanged:
        across old accounts.twin is acc
        all
          acc.owner /~ n implies acc ~ Current.account_of(acc.owner)
        end
    end
end
```

same object

Version 5: Complete Contracts (Deep Copy), Correct Implementation




```
class BANK
  deposit_on_v5 (n: STRING; a: INTEGER)
    require across accounts is acc some acc.owner ~ n end
    local i: INTEGER
  do ...
    -- imp. of version 1, followed by a deposit into 1st account
    accounts[accounts.lower].deposit(a)
  ensure
    num_of_accounts_unchanged: accounts.count = old accounts.count
    balance_of_n_increased:
      Current.account_of(n).balance =
        old Current.account_of(n).balance + a
    others_unchanged:
      across old accounts.deep_twin is acc
      all
        acc.owner /~ n implies acc ~ Current.account_of(acc.owner)
      end
  end
end
```



Complete Postcondition: Exercise

Consider the query *account_of (n: STRING)* of *BANK*.

How do we specify (part of) its postcondition to assert that the state of the bank remains unchanged:

- `accounts = old accounts`
- `accounts = old accounts.twin`
- `accounts = old accounts.deep_twin`
- `accounts ~ old accounts`
- `accounts ~ old accounts.twin`
- `accounts ~ old accounts.deep_twin`

Writing Postcondition: Exercise 1.1

*is_positive (*i: **INTEGER**): **BOOLEAN**

ensure

tag: x > 0

TS_p (3)

TS_p (-4) ← Result correctly ref to false but a postcond_ violation

Writing Postcondition: Exercise 1.2

```
is_positive (x: INTEGER): BOOLEAN
    ensure
        if x > 0 then
            Result = True
        end
```

→ Syntax.

else

not Result

end

$x > 0$ implies Result

$\neg(x > 0)$ implies not Result