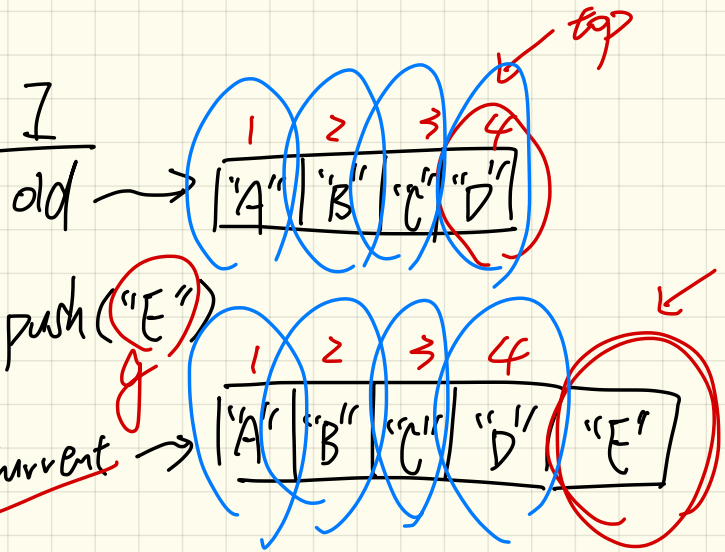
Lecture 16
Tuesday November 5

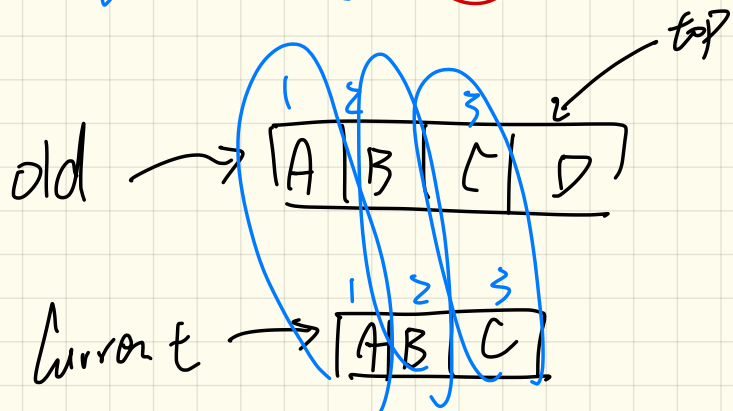
push

Str. 1



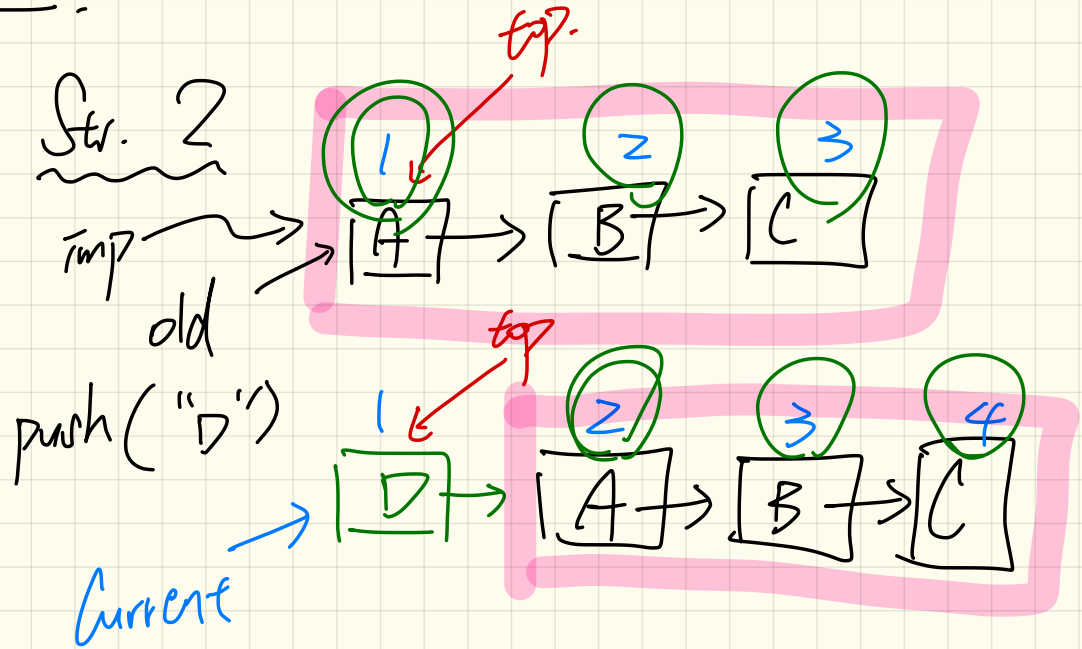
pop

Str 1



push

Str. 2



across 2 1..| count is i

all

end

tmp[i] ~ (old tmp.all)[i-1]

Developing a LIFO Stack

SCP violated:

1. duplicates (contracts)

2. change on implementation may involve multiple changes on contracts

```
class LIFO_STACK[G] create make
feature {NONE} -- Strategy 1: array
  imp: ARRAY[G]
feature -- Initialization
  make do create imp.make_empty ensure imp.count = 0 end
feature -- Commands
  push(g: G)
    do imp.force(g, imp.count + 1)
    ensure
      changed: imp[imp.count] ~ g
      unchanged: across 1 |..| count - 1 as i all
                   imp[i.item] ~ (old imp.deep_twin)[i.item] end
    end
  pop
    do imp.remove_tail(1)
    ensure
      changed: count = old count - 1
      unchanged: across 1 |..| count as i all
                   imp[i.item] ~ (old imp.deep_twin)[i.item] end
    end
```

1. use a linked list, where the 1st element is the top.

imp[1]

```
class LIFO_STACK[G] create make
feature {NONE} -- Strategy 2: linked-list first item as top
  imp: LINKED_LIST[G]
feature -- Initialization
  make do create imp.make ensure imp.count = 0 end
feature -- Commands
  push(g: G)
    do imp.put_front(g)
    ensure
      changed: imp.first ~ g
      unchanged: across 2 |..| count as i all
                   imp[i.item] ~ (old imp.deep_twin)[i.item - 1] end
    end
  pop
    do imp.start ; imp.remove
    ensure
      changed: count = old count - 1
      unchanged: across 1 |..| count as i all
                   imp[i.item] ~ (old imp.deep_twin)[i.item + 1] end
    end
```

```
class LIFO_STACK[G] create make
feature {NONE} -- Strategy 3: linked-list last item as top
  imp: LINKED_LIST[G]
feature -- Initialization
  make do create imp.make ensure imp.count = 0 end
feature -- Commands
  push(g: G)
    do imp.extend(g)
    ensure
      changed: imp.last ~ g
      unchanged: across 1 |..| count - 1 as i all
                   imp[i.item] ~ (old imp.deep_twin)[i.item] end
    end
  pop
    do imp.finish ; imp.remove
    ensure
      changed: count = old count - 1
      unchanged: across 1 |..| count as i all
                   imp[i.item] ~ (old imp.deep_twin)[i.item] end
    end
```

Using MATHMODELS Library

Implementing an Abstraction Function

```
class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation
  imp: LINKED_LIST[G]
feature -- Abstraction function of the stack ADT
  model: SEQ[G]
    do create Result.make_empty
      across imp as cursor loop Result.append(cursor.item) end
    end
```

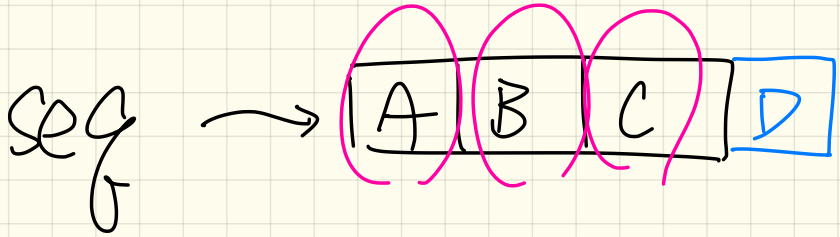
Strategy 3

from MATHMODELS

Writing Contracts using the Abstraction Function

```
class LIFO_STACK[G -> attached ANY] create make
feature -- Abstraction function of the stack ADT
  model: SEQ[G]
feature -- Commands
  push (g: G)
    ensure model ~ (old model.deep_twin).appended(g) end
```

two seperate calls to model query

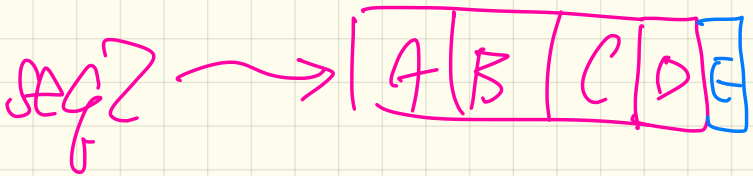


seq.append(D)

command.

seq2 := seq.appended(E)

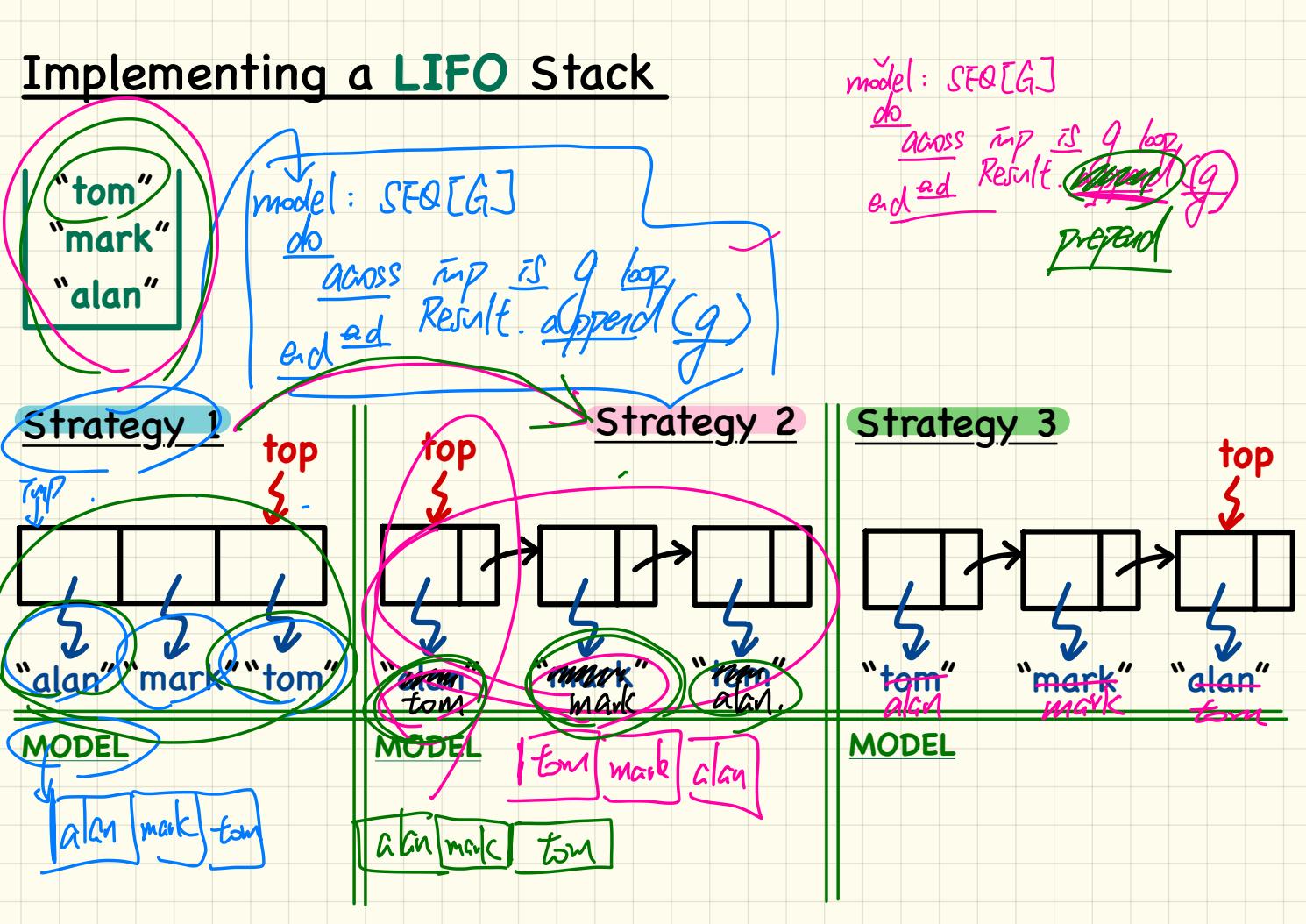
immutable query.



MATH MODELS

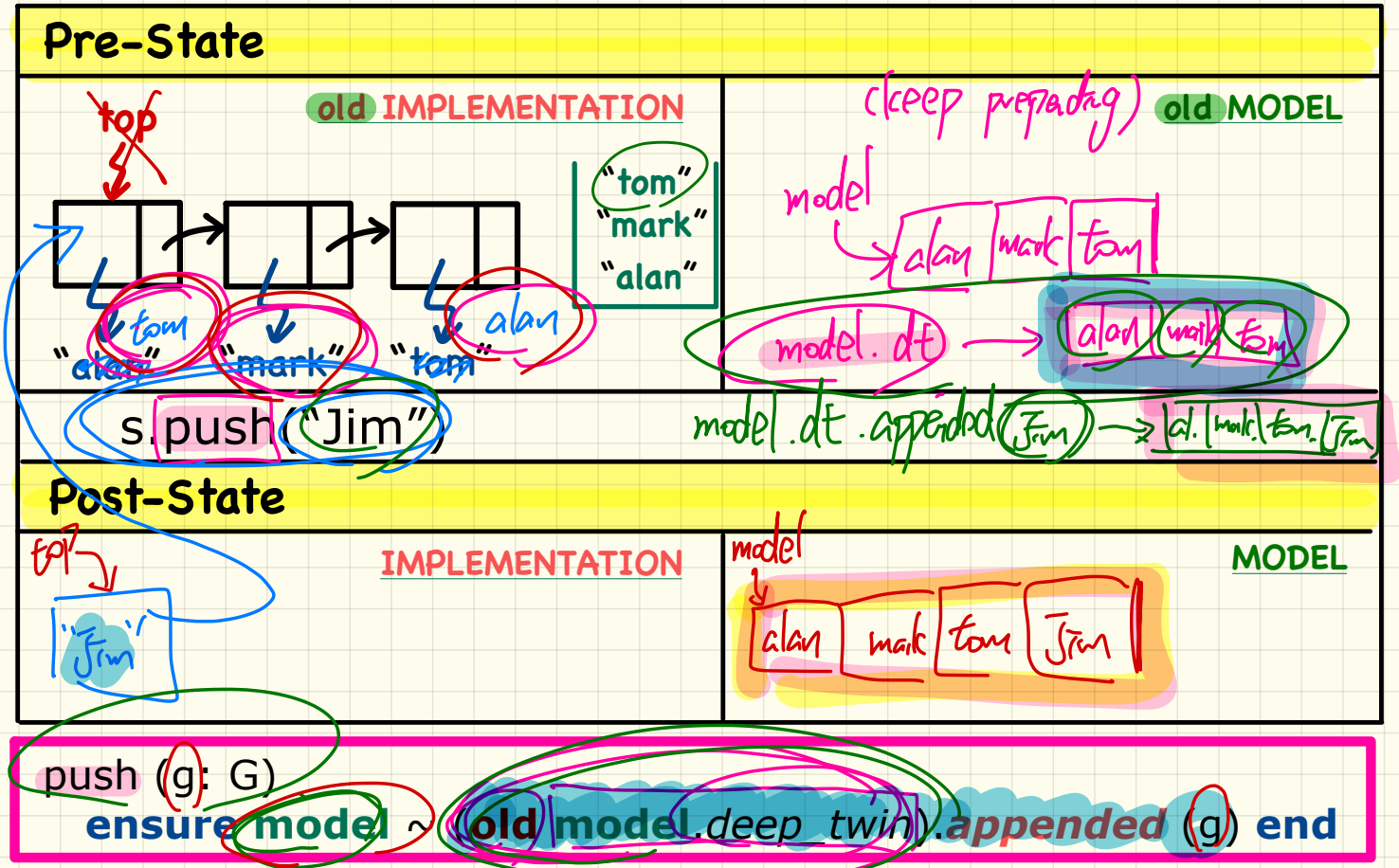
1. Commands
   - ↳ use to implement the "mode" e.g. (append)
2. Immutable queues
   - ↳ use to write contracts (e.g. push)

Implementing a LIFO Stack



Checking MATHMODELS Contracts at Runtime

Strategy 2



class STACK[G]

feature {NONE}
tmp: ~~ARRAY~~[G]
LL [G]
↳ front of list ≈ top

feature
public model: SEQ[G]
do
[keep ~~appending~~ prepending to the end of Result
top: G
ensure
Result ~ model.last

tmp → alan | mark | tom

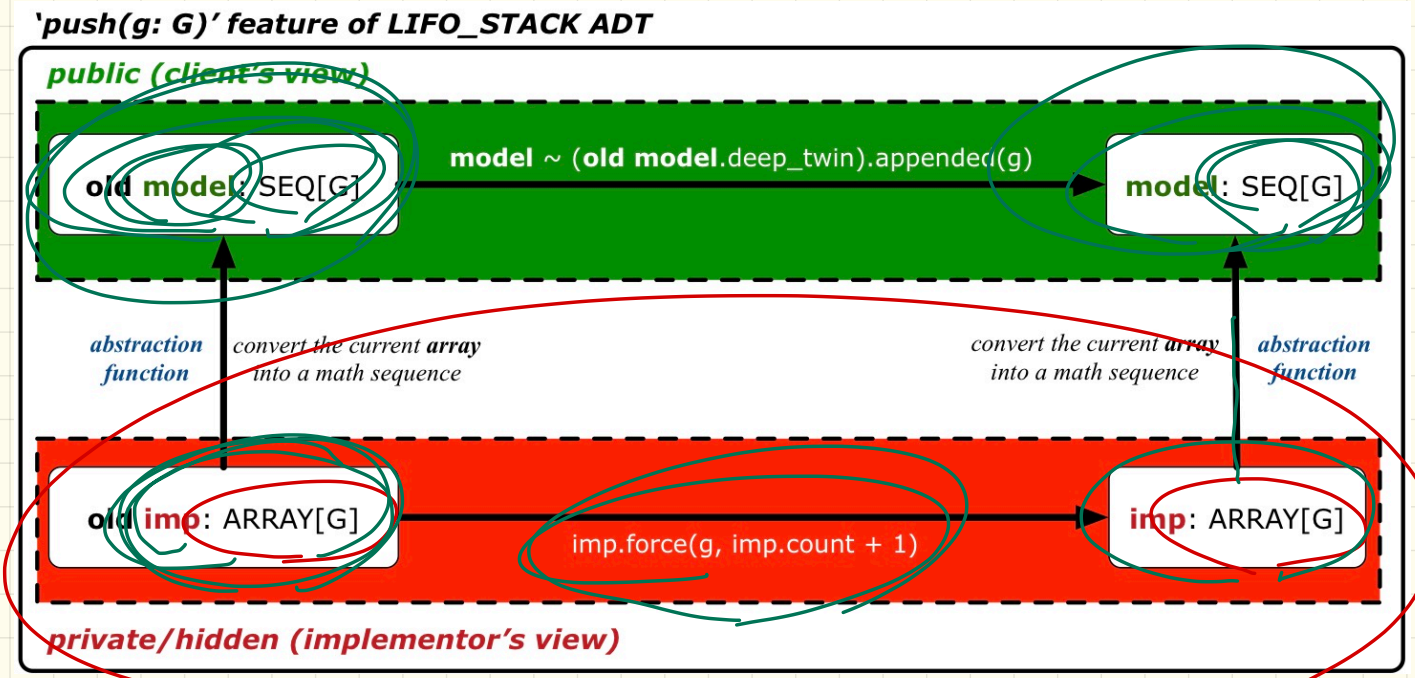
tom
mark
alan

tmp → tom → mark → alan

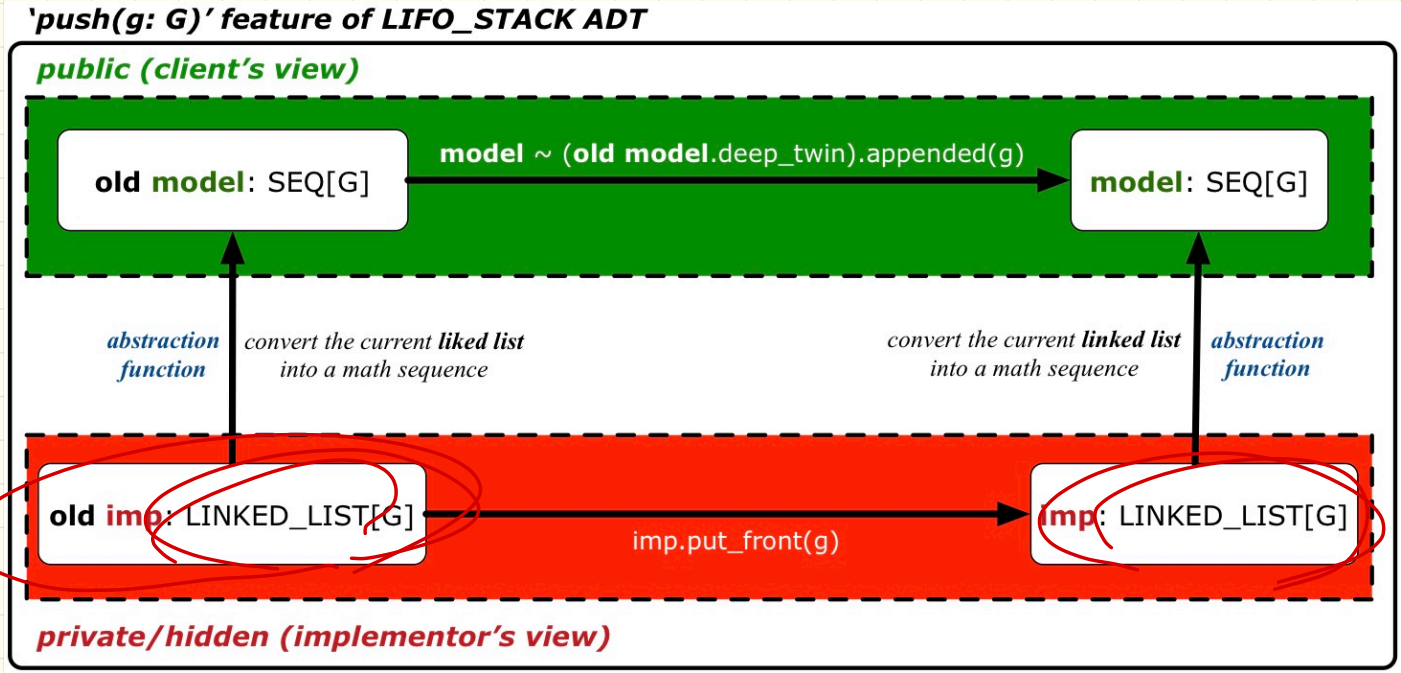
model → alan | mark | tom

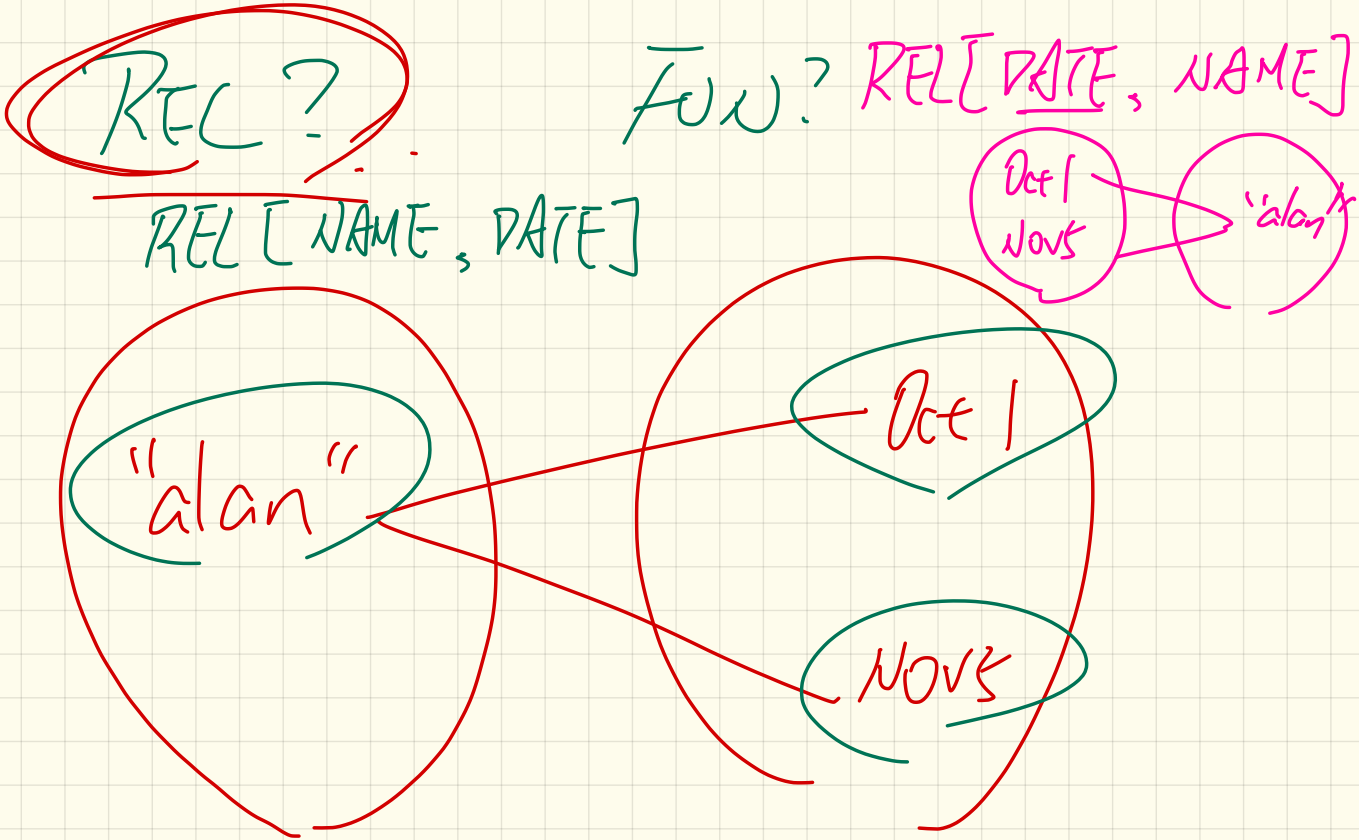
model → alan | mark | tom

Strategy 1: Mathematical Abstraction



Strategy 2: Mathematical Abstraction




REL?
REL[NAME, DATE]
FUN? REL[DATE, NAME]
Oct 1
Nov 5
"alan"
"alan"
Oct 1
Nov 5

```
feature {NONE}
    model: SEQ[G]

feature
    get_model: SEQ[G]
```