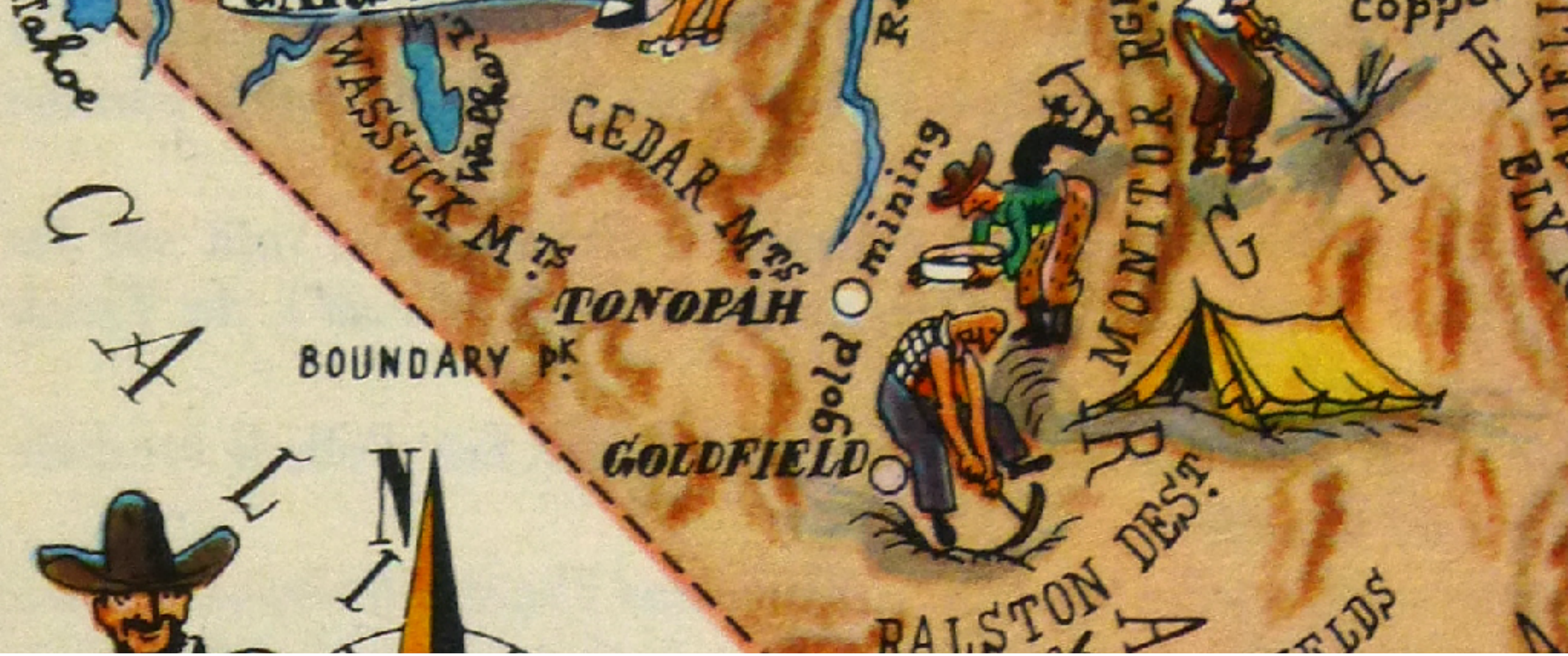


THE ENVIRONMENT MODEL

Theory, not implementation

MAINTAINING YOUR DIGITAL ENVIRONMENT

- So, I never talk about this stuff, but today I will.
  - Keep your discs as lightly filled as possible
    - Clear out that music collection, move it to external storage
  - Use standard formats, and as plain a format as possible
    - .txt is better than any word document format, for example
- This is something I personally do work towards on a frequent basis, but could do even better at

THE ENVIRONMENT MODEL

- We studied the environment model very briefly before the second exam, as we implemented a simple environment for our interpreter
- We'll be back to our interpreter very soon, I promise.
- Today we're going to look at the environment model, however

EXAM Q&A

“

I’m positive my answer was correct, how could you possibly have graded it wrong?

-More than one of you…

UNIT TESTS

What command is
typically used to execute
unit tests in Rust?

cargo run

What is the difference between #[cfg(test)] and #[test]

UNIT TEST SIDE QUESTION(S)

Does every language have a unit testing framework?

What is the difference between unit testing and debugging?

What are the differences between a set and a list?

Lists are ORDERED

Sets are collections of UNIQUE elements

*Sets **may** be ordered*

*Lists **may** have duplicates*

MEMBER METHODS

How does Rust implement member methods on an object? Consider only the primary implementation.

This exam question does not ask how one might write a member method, but how Rust itself implements those methods.

FOLLOWUP ON DISPATCH METHODS

How does the secondary implementation of Rust member methods on objects differ from the primary implementation?

Tricksy question: this question doesn't imply that a secondary method actually exists, but because it's theoretically possible to use message passing between objects, that could be an interpretation that counterindicates table-based static dispatch.

BACK TO THE ENVIRONMENT

Did we ever really leave?

WHAT THE ENVIRONMENT IS NOT

- The environment is *not* the full suite of programs and settings within an operating system
- While this is called an environment, it is not what we refer to when writing programs as the environment model
- The environment model is not a historical artifact

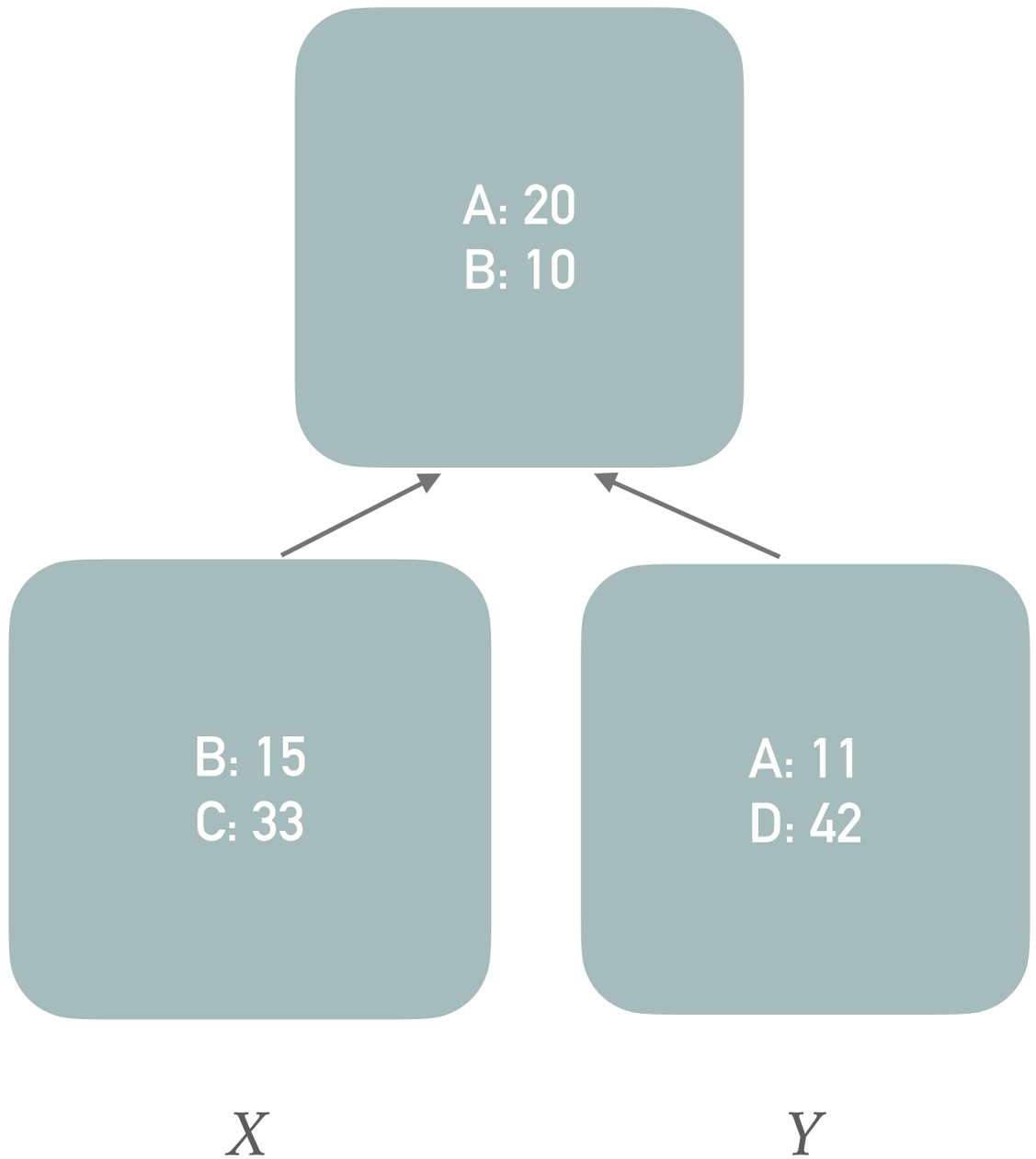
VARIABLES

- When we encountered *assignment* in last week's discussion, we said that it radically changed things and that the substitution model could no longer be used
- The place in which values may be stored is an *environment*
- Please note the singular here; we will actually use a set of environments, known as *frames*.

FRAMES

- Each frame is a table (which may be empty) of *bindings*
- Each frame also has a reference to its *enclosing environment*
  - The exception to this is the *global environment*
- Any variable which has a particular *key* in the environment is considered *bound* within that environment; any variable which does not have a key is considered *unbound* in that environment.

A SAMPLE SET OF ENVIRONMENTS



RULES OF EVALUATION

- These remain unchanged!
  - Evaluate the subexpressions of the combination
  - Apply the value of the operator sub-expression to the values of the operand subexpressions
- The only thing that changes is what "apply" means

PROCEDURES

- Procedures are pairs of some code and a pointer to an environment
- This should sound REALLY familiar to you!

ENVIRONMENT IN RUST

```
// arrange
let environment = Environment { key: String::from("x"),
                                value: Expression::Number(5.0) };
let values = vec![Expression::Number(2.0),
                  Expression::Variable(String::from("x"))];

// act
let sum = evaluate(&Expression::Add(values),
                   &environment);

// assert
assert_eq!(7.0, sum);
```

RULES OF PROCEDURE APPLICATION

- A procedure is applied to a set of arguments by constructing a frame, binding the formal parameters of the procedure to the arguments of the call, and then evaluating the body of the procedure in the context of the new environment constructed.
- A procedure is created by evaluating a lambda expression relative to a given environment

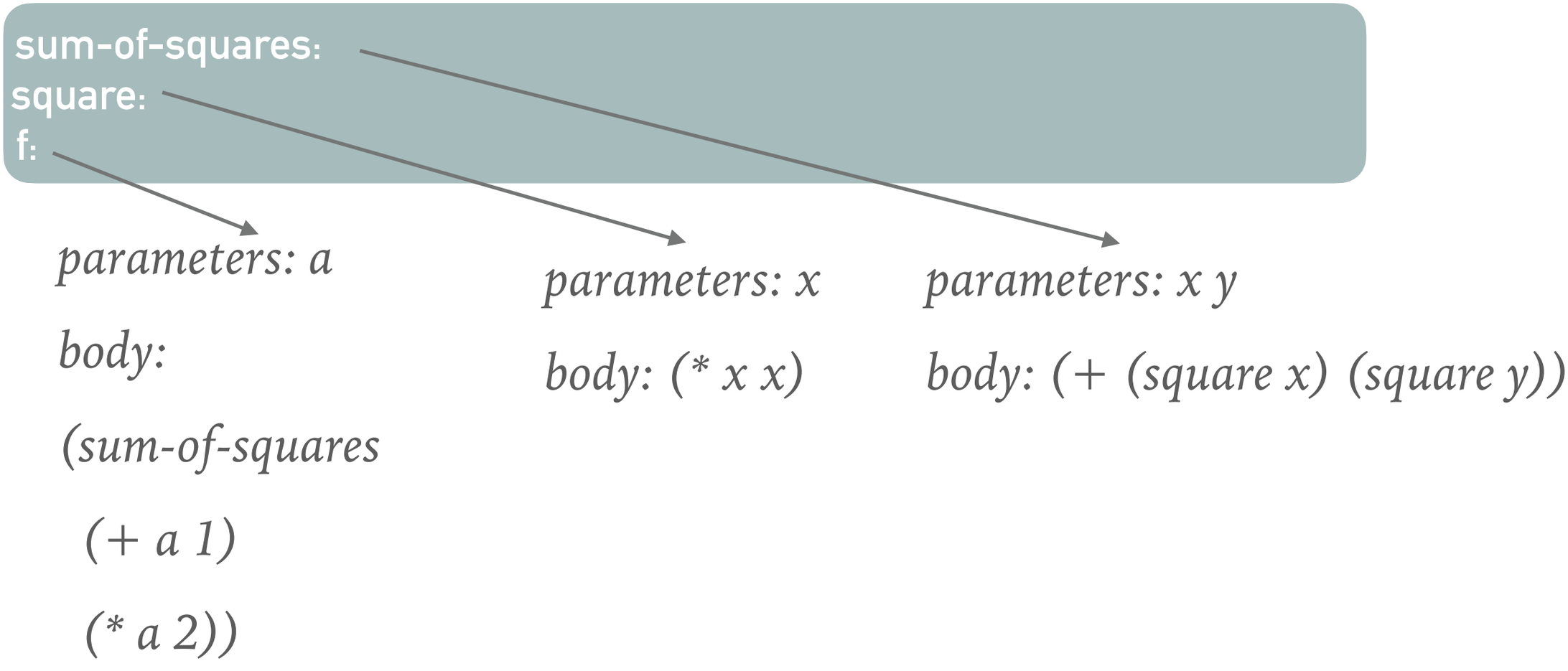
THIS IS MUCH MORE CORRECT!

- You may recall that I did not even consider crafting our interpreter using the substitution model
- Now we're seeing that in process!
- The environment model is much more suitable to our tasks, and can be applied with recursion too!

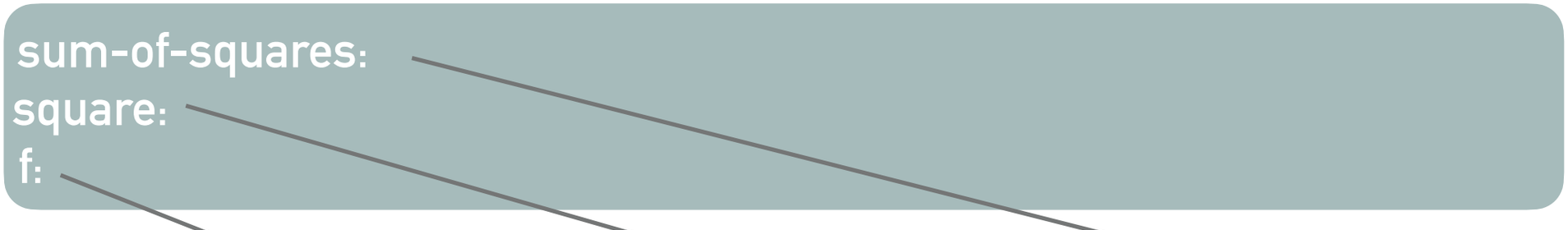
THE SUBSTITUTION MODEL

```
(define (square x)
  (* x x))
(define (sum-of-squares x y)
  (+ (square x) (square y)))
(define (f a)
  (sum-of-squares (+ a 1) (* a 2)))
```

SUBSTITUTION MODEL



SUBSTITUTION MODEL



parameters: a

body:

(sum-of-squares

(+ a 1)

(a 2))*

parameters: x

body: (x x)*

parameters: x y

body: (+ (square x) (square y))

Every call to square creates a new environment containing a binding for x!

LOCAL STATE

- The concept of frames allows us to go further into a repository of local state
- We can more easily consider an object with its own local state using the environment model
- For homework 11, I encouraged a simplistic approach; this may have meant using global state to you. That's what I expected, at least.

REMOVING GLOBALS

- The concept of local state to remove or replace global state is part and parcel of contemporary programming
- Generally, modern programs should avoid global state
- We prefer local *scoping* of variables, broadly speaking

EXAMPLE IN C

```
#include <stdio.h>
#include <stdio.h>

int a = 25;

int main(int argc, char **argv) {

        int a = 10;
        printf("%d", a);
        return 0;

}
```

Where are frames created?