

## practice problem: memoization

Memoization is a clever idea that allows us to save on computation. It works by keeping track of evaluation of a procedure on a specific argument, and simply returns the remembered value if the procedure has already been run on that argument. Draw the environment diagram for the following procedure definition and application.

```
(define (memoize proc)
  (let ((cache '()))
    (lambda (arg)
      (if (assoc arg cache)
          (cadr (assoc arg cache))
          (let ((answer (proc arg)))
            (set! cache (cons (list arg answer) cache))
            answer))))))
(define my-sq (memoize (lambda (x) (* x x))))
(my-sq 5)
```

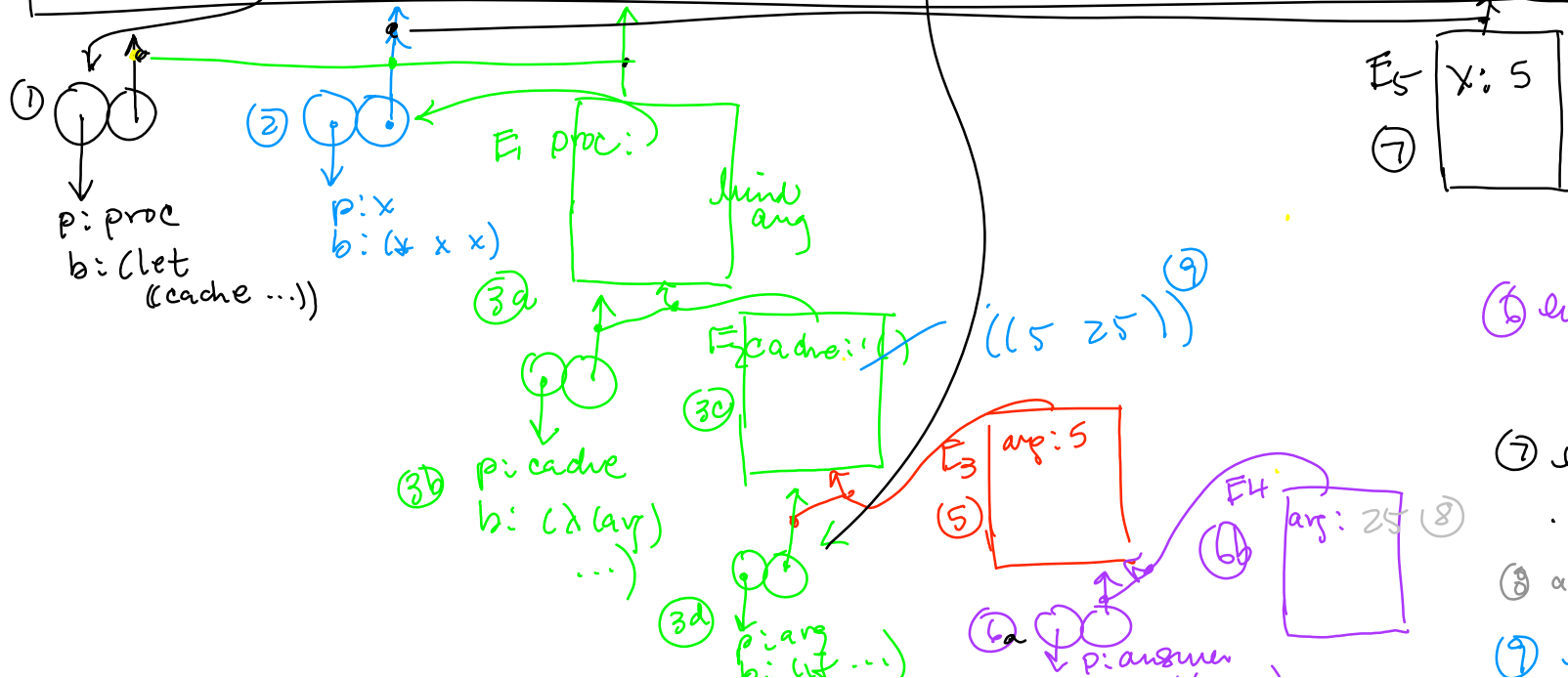
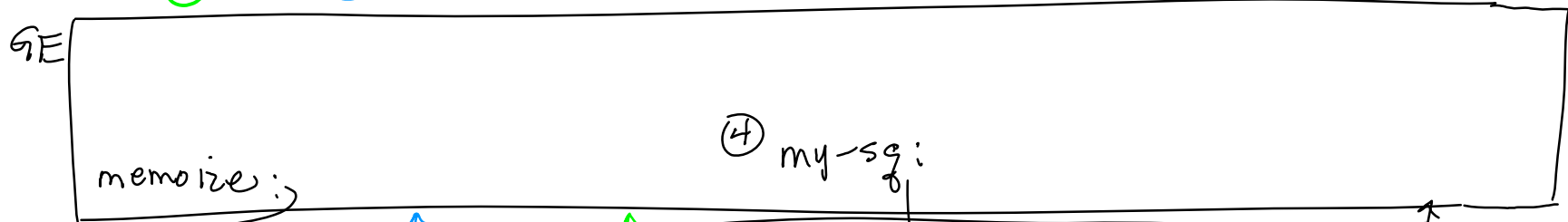
# memoization

Memoization is a clever idea that allows us to save on computation. It works by keeping track of evaluation of a procedure on a specific argument, and simply returns the remembered value if the procedure has already been run on that argument.

```

(1) (define (memoize proc)
      (let ((cache '()))
        (lambda (arg)
          (if (assoc arg cache)
              (cadr (assoc arg cache))
              (let ((answer (proc arg)))
                (set! cache (cons (list arg answer) cache))
                answer))))))
(4) (define my-sq (memoize (lambda (x) (* x x))))
(5) (my-sq 5)
  
```

- (3) apply memoize
- (3a) drop frame, bind arg, tie env
- (3b) eval body memoize; desugar to  $(\lambda (x) (cadr \dots)) '()$  | GE
- (3c) apply  $\lambda$
- (3d) eval body = create proc



- (6) eval body (lambda ...) ..
- (7) eval (pr ... find pro
- (8) apply pra
- (9) eval loc