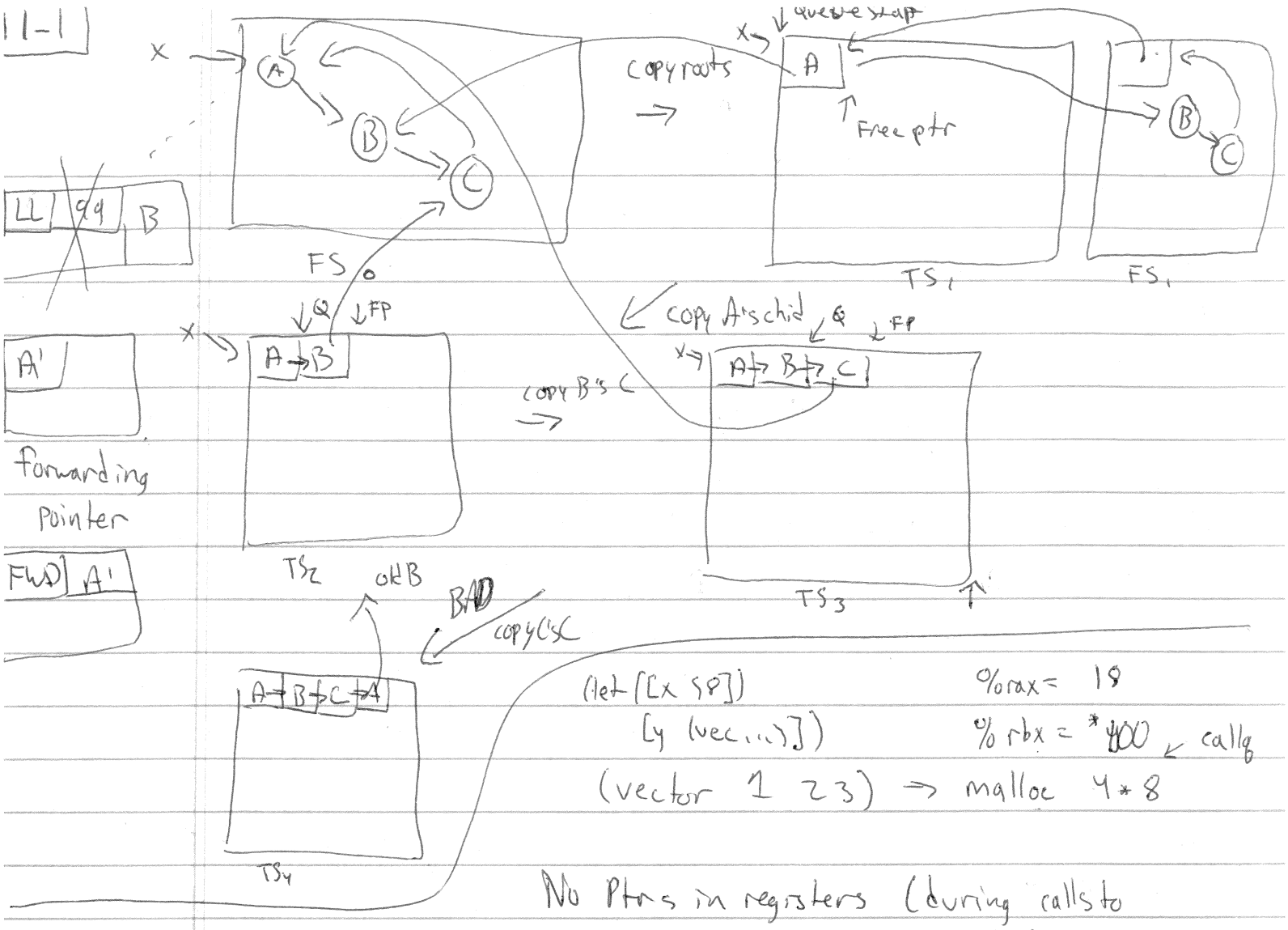
11-1



(let [x sp])
[y (vec ...)])

%rax = 19
%rbx = *400 ← callq
(vector 1 23) → malloc 4*8

No ptrs in registers (during calls to malloc)

Two stacks (1 for values + return
1 for pointers)

rsp
rbp

```
(8 (pushq R)
   (set! *rsp  *R)
   (set! rsp  (-rsp 8)))
(8 (ret)
   (set! pc *rsp)
   (set! rsp (+rsp 8)))
```

Code:

```
int64_t* root_stack_ptr;
void init( heap-size, root-stk-size)
int64_t* collect ( int64_t*
                   root-stack-top)
int64_t* free_ptr, *fromspace_end;
```

2mil: callq _read_int → R.2mil

(callq _read_int(%rip))

movq _read_int , %rax 0(%rax)%0

movq _root_stack_ptr , %rax

11-2 uniquify → typec → flatten → select → assign → patch → print

$R \Rightarrow R^T$ has typeinfor

expose-alloc : $R^T \Rightarrow R^*$
- to remove (vector e, ..)
- replace with (alloc i)

R has vector
R^* ✗ vector
$\Rightarrow$ { alloc, collect, global }

(vector e_0 ... e_n)
$\Rightarrow$ (let ([x_0 e_0]) ... (let ([x_n e_n]) ↙ compute in expose
(let ([_ (if (< (+ (global free-ptr) bytes) (global from-space-end))
(void)
(collect bytes))])
(let ([v (allocate n type)])
(let ([_ (vector-set! v 0 x_0)])
... (let ([_ (vector-set! v n x_n)])
v
)...)))...)

count → we compute

Flatten C Expr += alloc, void, global, vector-ref ~~vector-set!~~
C Stmt += collect, ↖ vector-set!

Old: set of variables
New: Map from Var to Type