Estructuras de Control Avanzadas:

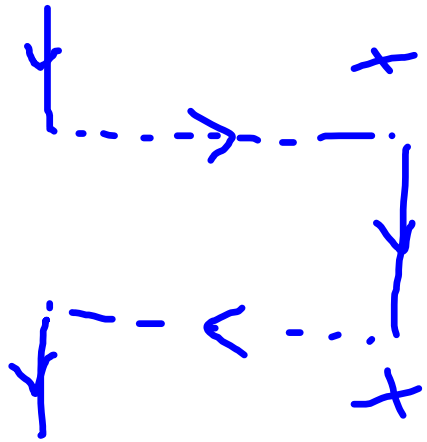
- Excepciones
- Longjmp
- Interrupciones

→ retroceder rápido en la pila

Funciones:

- Llamador / Llamado
- Maestro / Esclavo

↳ no siempre sirve

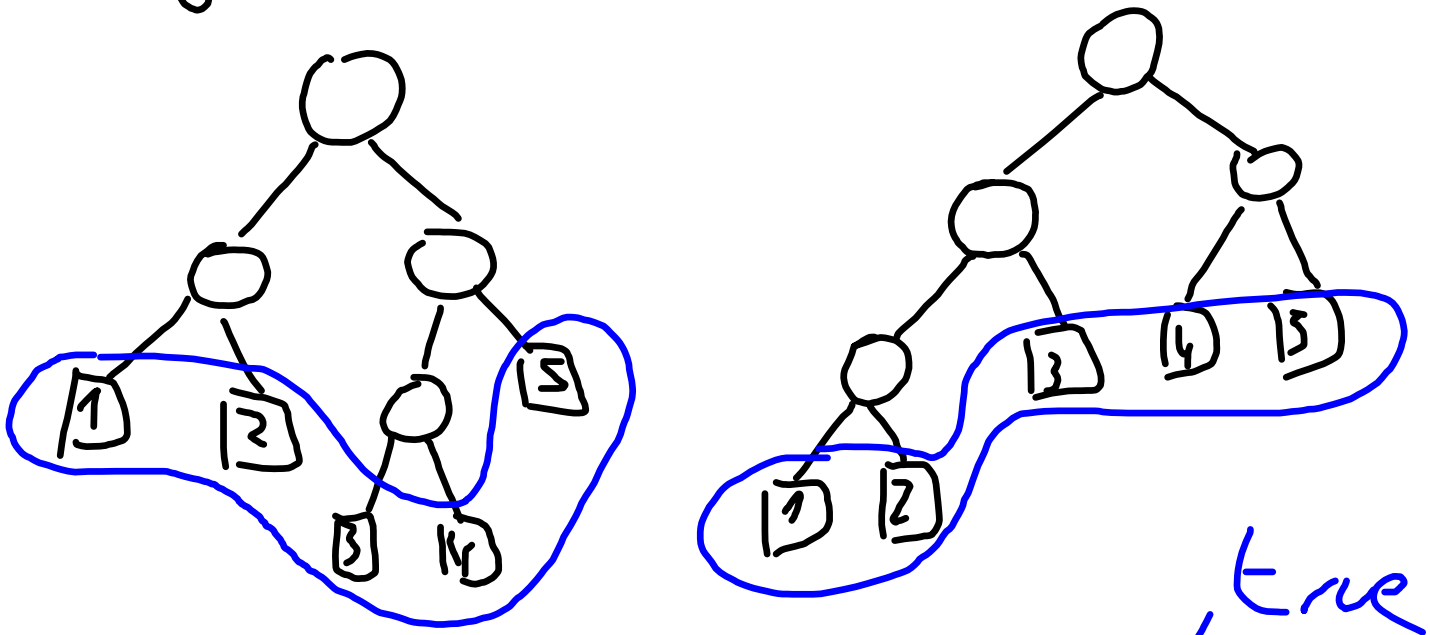


Ej 1:

```
calcular()
{
while(true)
{
  dato = ___
  .
  .
  .
}
```

```
more()
{while(true){
  for(i=0; i<20; i++)
    printf(data);
  wait_for_enter();
}}
```

Ej 2:

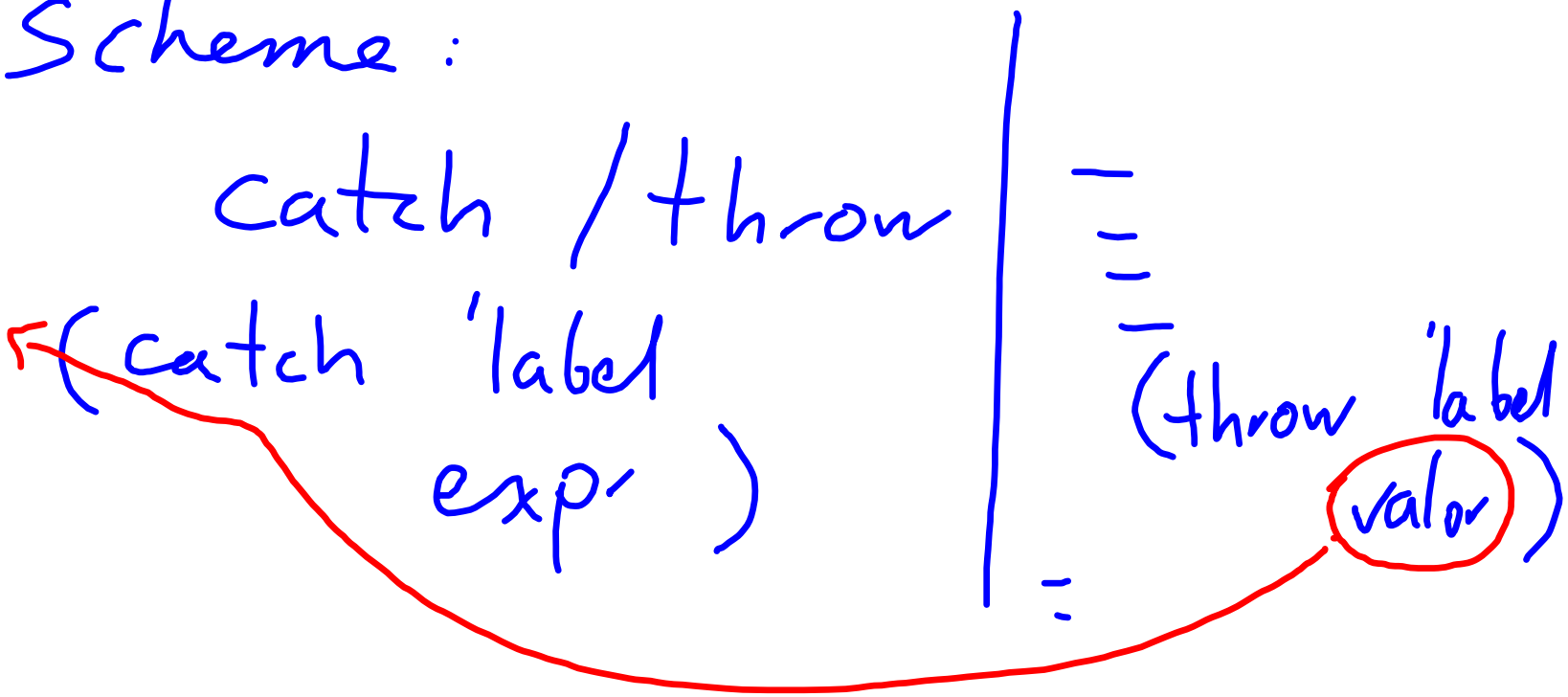


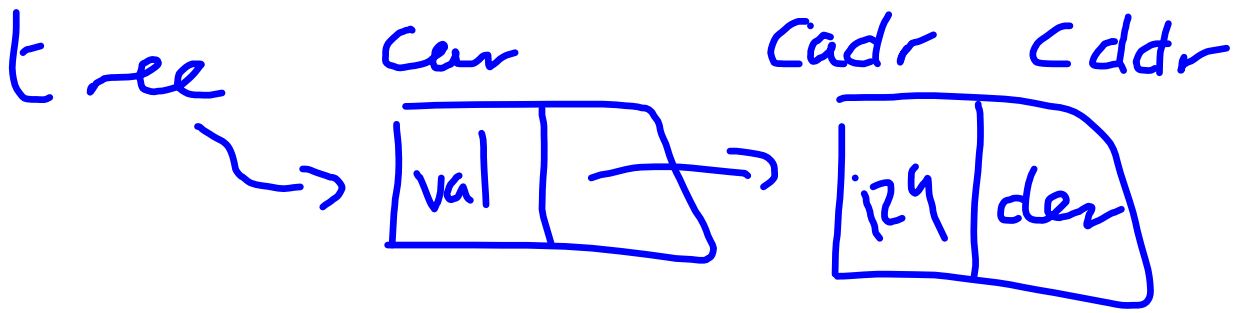
same_fringe (t1, t2) { true / false

Long jmp / setjmp :

Scheme :

catch / throw





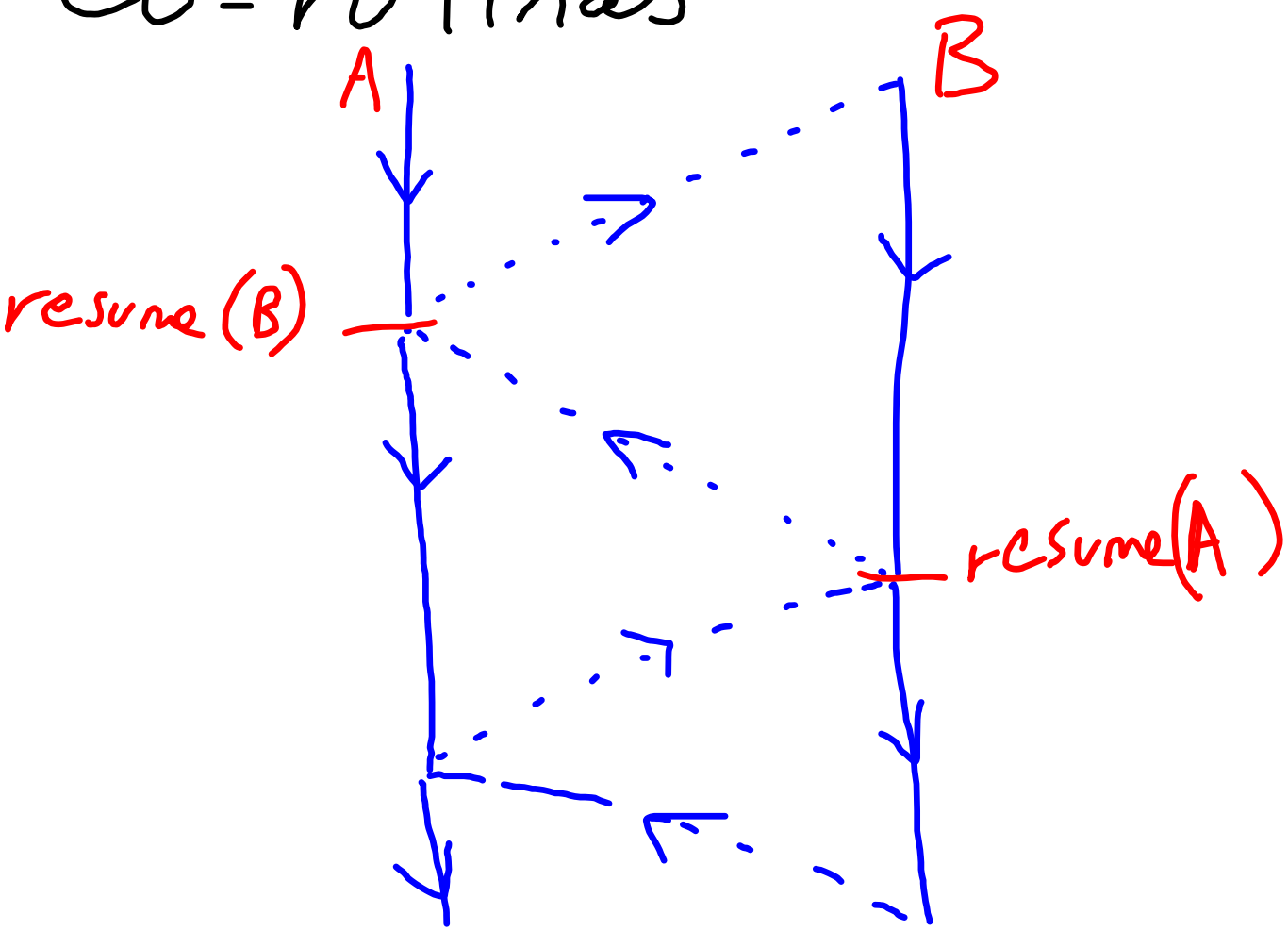
Si encuentro el valor

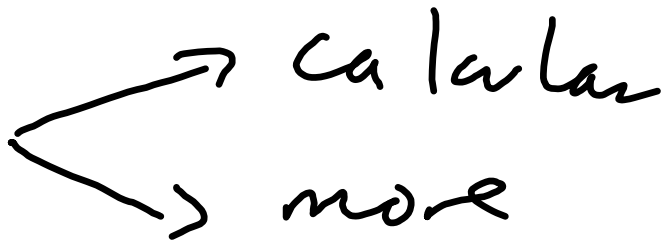
throw #t

en vez de retornar

↳ ¿más eficiente?

Co-rutinas



Ej 1 


```
calculan(more_id)
{
while (true) {
   data = -
   resume(more_id, data)
}
```

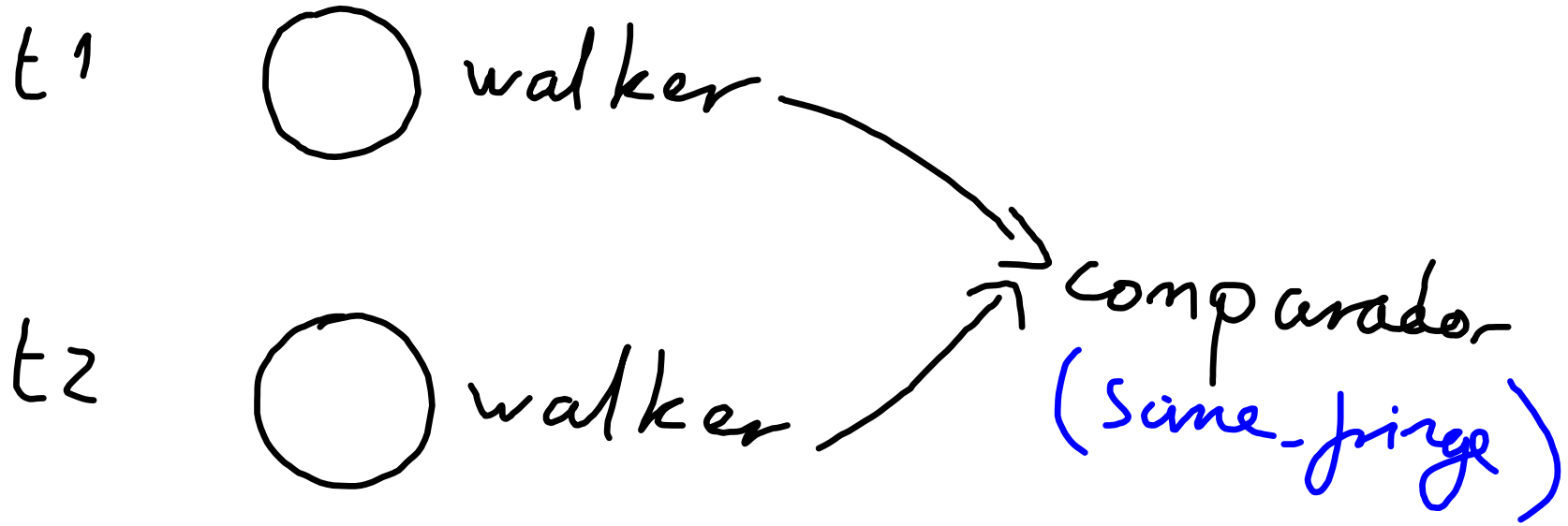
```
more(calculan_id)
{
while(true) {
   data = resume(calculan_id);
   printf( \
   i++; i=0;
   if(i>20){wait_for_enter();}
}}
```

¿Cómo empieza?

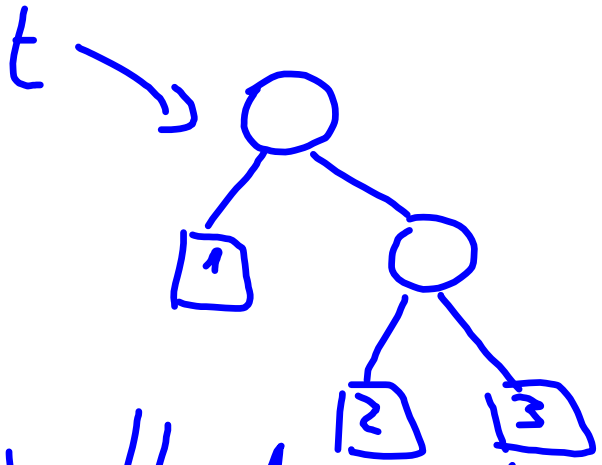
```
id=make-coroutine();
 start_coroutine(id, funcion)
 resume_coroutine(id, val);
 current coroutine()
 exit_resume(id, val);
```

→ ppto: calc /more completo + término (EOF + 'quit')

Ej 2: same_fringe:



¿ podríamos construir :

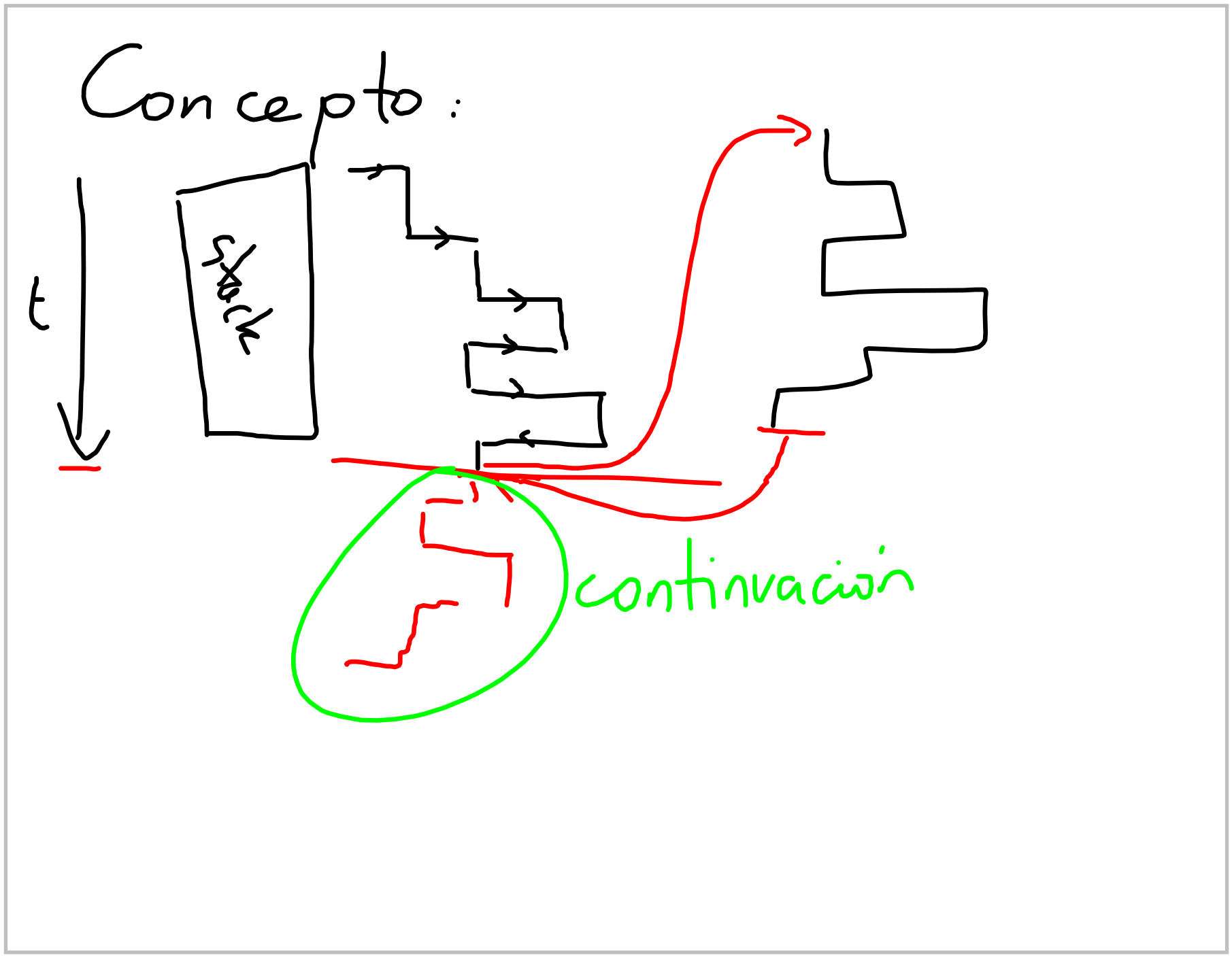


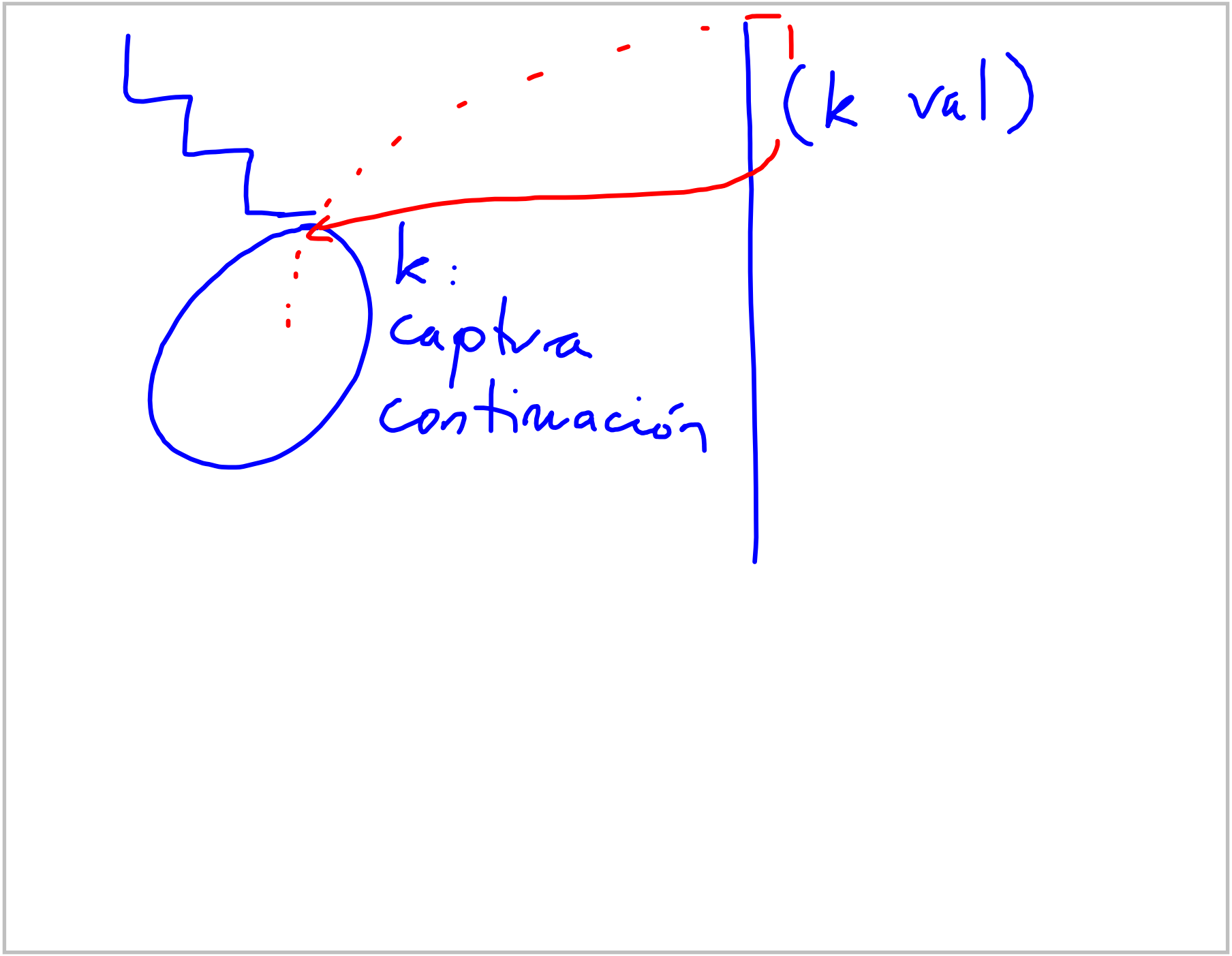
(define walker1 (make-walker t))

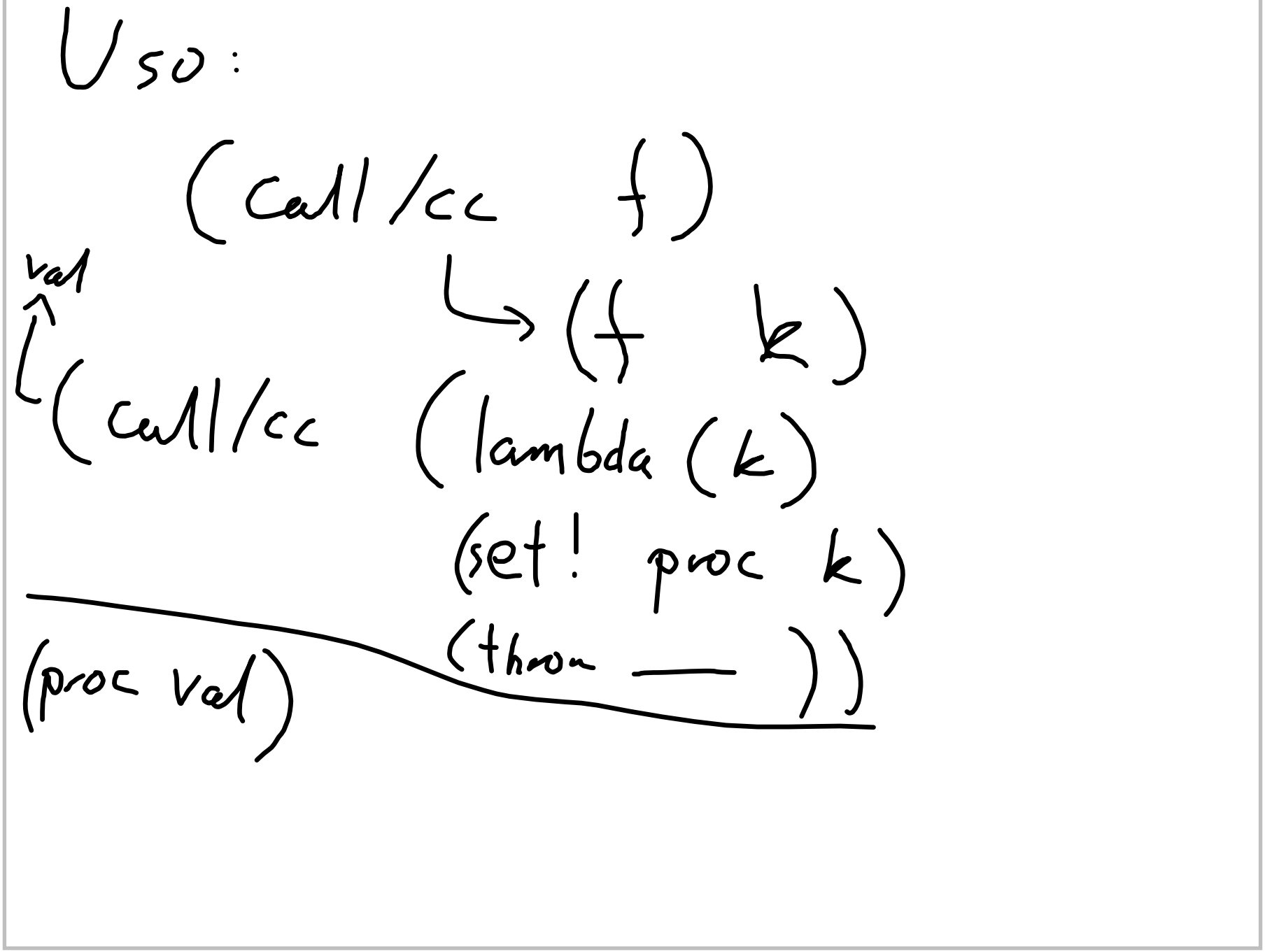
(walker1) → 1

(walker1) → 2

?


Concepto:
t
stack
continuación


(k val)
k:
captura
continuación


Uso:
(call/cc f)
(f k)
val
(call/cc (lambda (k)
(set! proc k)
(throw —))
(proc val)