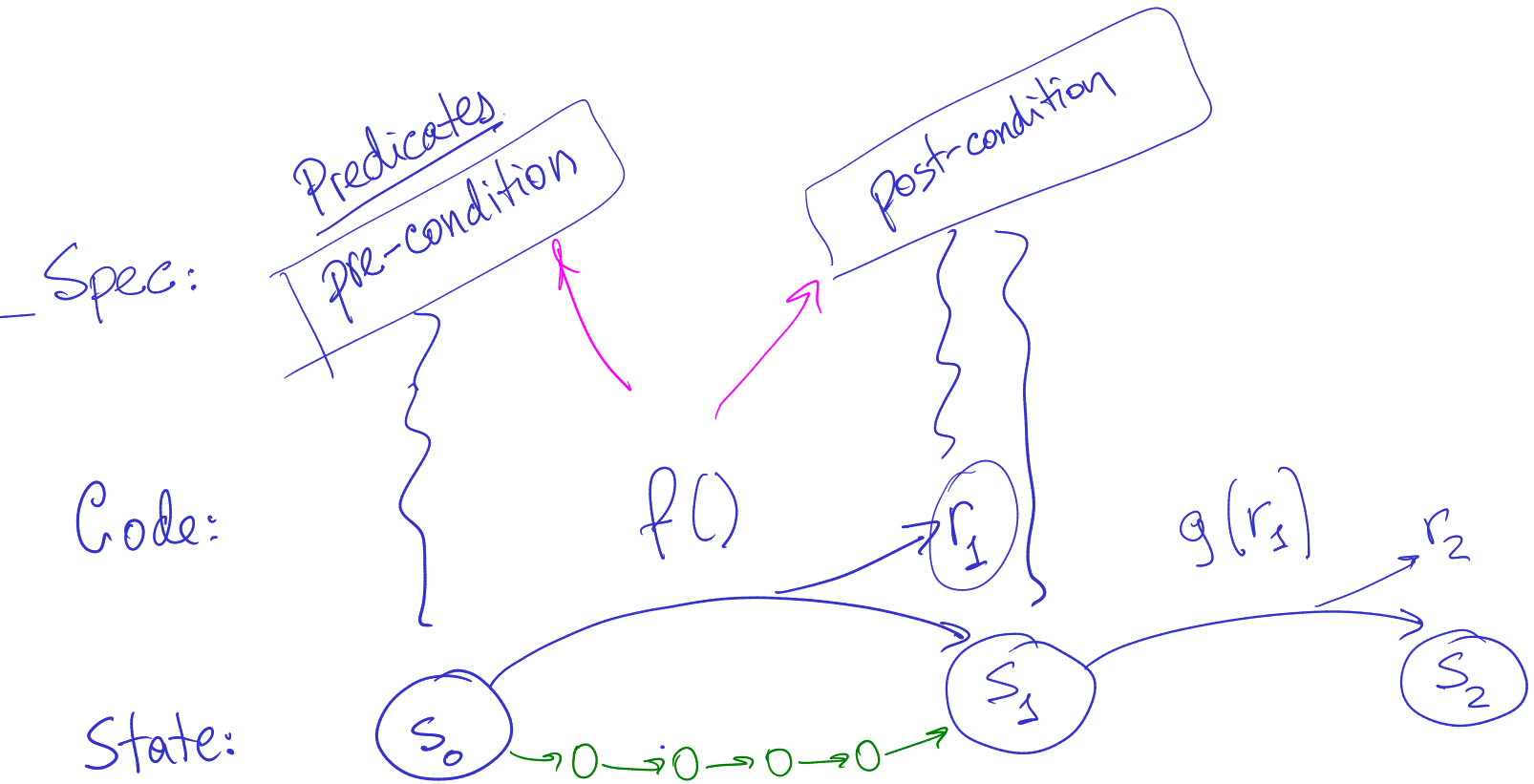
Specifications + Abstractions.

Goal: reason about possible execution.

Butler: general view: concurrency
crashes
distributed

Today: specialized for sequential.

Hoare logic



State-machine view.

No calls, no return values.

↳ calls ∈ state

↳ return values ∈ state, externally visible

Intermediate states.

→ concurrency.

→ crashes

Example: StatDB

```
# add(x) adds element x.
def add(x):
    total = total + x
    count = count + 1

# avg() returns average so far.
def avg():
    return total / count
```

Spec:

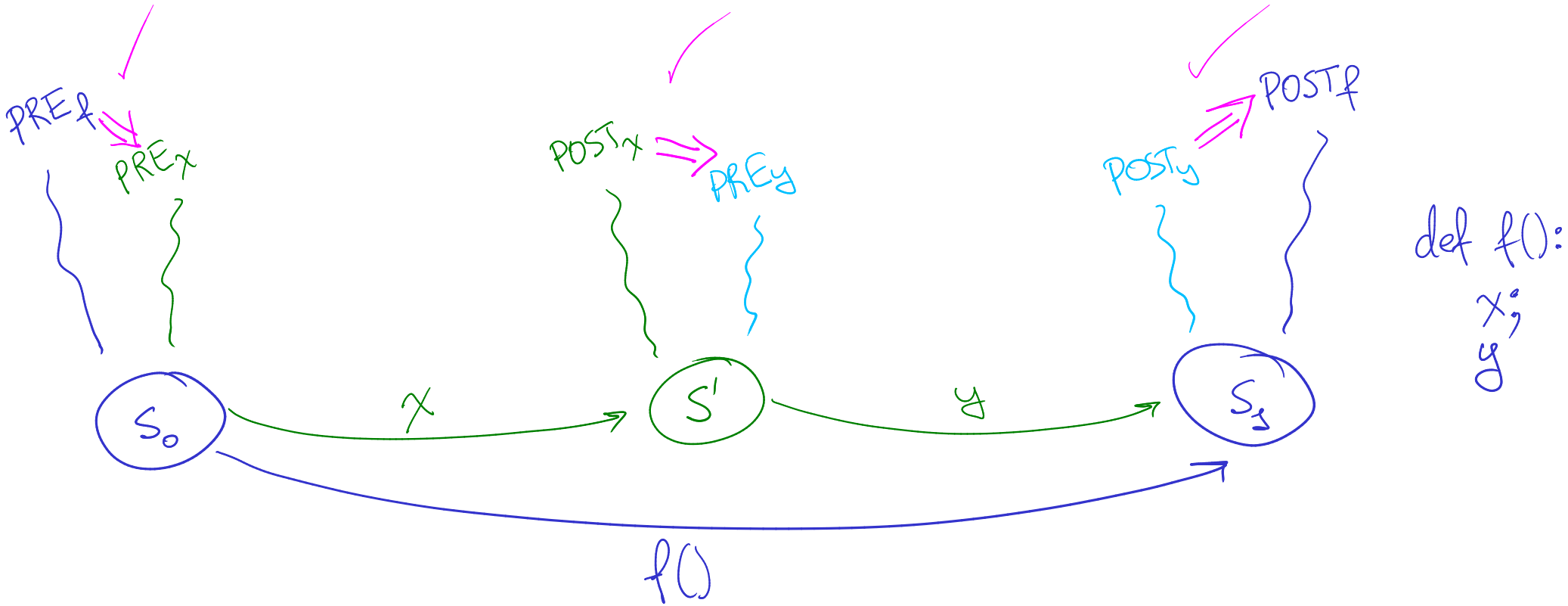
add(x): PRE(s): True

POST(s', r): $s'.total = s.total + x \wedge$
$s'.count = s.count + 1$.

avg(): PRE(s): $s.count \neq 0$

POST(s', r): $s' = s \wedge$
$r = s.total / s.count$.

Sequencing rule




```
def f():
    x;
    y
```

Applying Hoare logic seq. rule.

```
# add(x) adds element x.
def add(x):
    total = total + x
    count = count + 1
```

True (S_0)

```
tmp := R(total)
```

$S_1 = S_0 \wedge tmp = S_0.total$

```
W(total, tmp + x)
```

$S_2.total = S_0.total + x \wedge S_2.count = S_1.count$

```
tmp := R(count)
```

$S_3 = S_2 \wedge tmp = S_2.count$

```
W(count, tmp+1)
```

$S_4.count = S_2.count + 1 \wedge S_4.total = S_3.\cancel{count}\ total$

$S_4.count = S_0.count + 1 \wedge S_4.total = S_0.total + x.$

Specs (POST).

R(total): POST(s', r):
→ $s' = s \wedge r = s.total$.

R(count): POST(s', r):
→ $s' = s \wedge r = s.count$

W(total, v): $s'.total = v \wedge s'.count = s.count$
→

W(count, v): $s'.count = v \wedge s'.total = s.total$

Abstractions

Simple spec for StatDB.

add(x): append x to history.

avg(): ret. avg of history.

State abstraction.

- Define new type of spec state.
- Write spec using spec state.
- Connect code state, spec state.

Example

State = list nat

add(x): POST(s'): $s' = s ++ [x]$.

avg(): PRE(s) len(s) > 0

POST(r): $r = \frac{sum(s)}{len(s)}$. ^

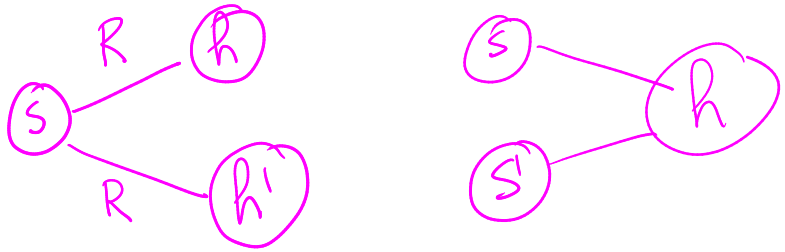
s';

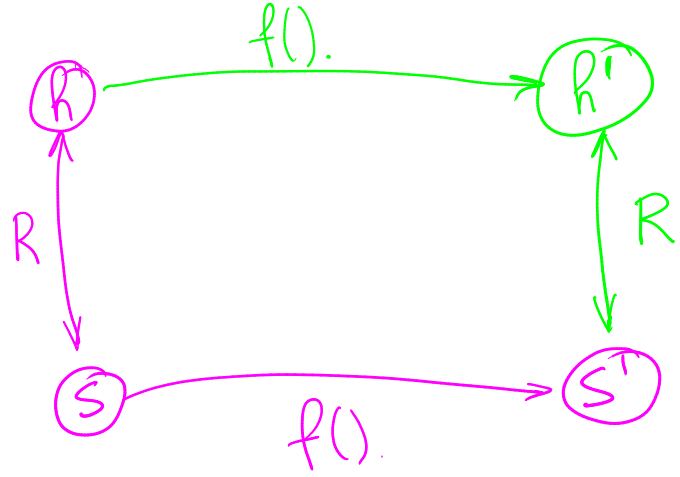
$s' = s$.

Abstraction relation.

R (code state s, spec state h)

$$R(s, h) := s.\text{total} = \text{sum}(h) \wedge s.\text{count} = \text{len}(h).$$





↳ add(x).

add(x): PRE(s):

$\exists h,\ R(s, h)$.

POST(s', r):

$\exists h',\ R(s', h') \wedge$

$h' = h +\!+ [x]$.

Layering.