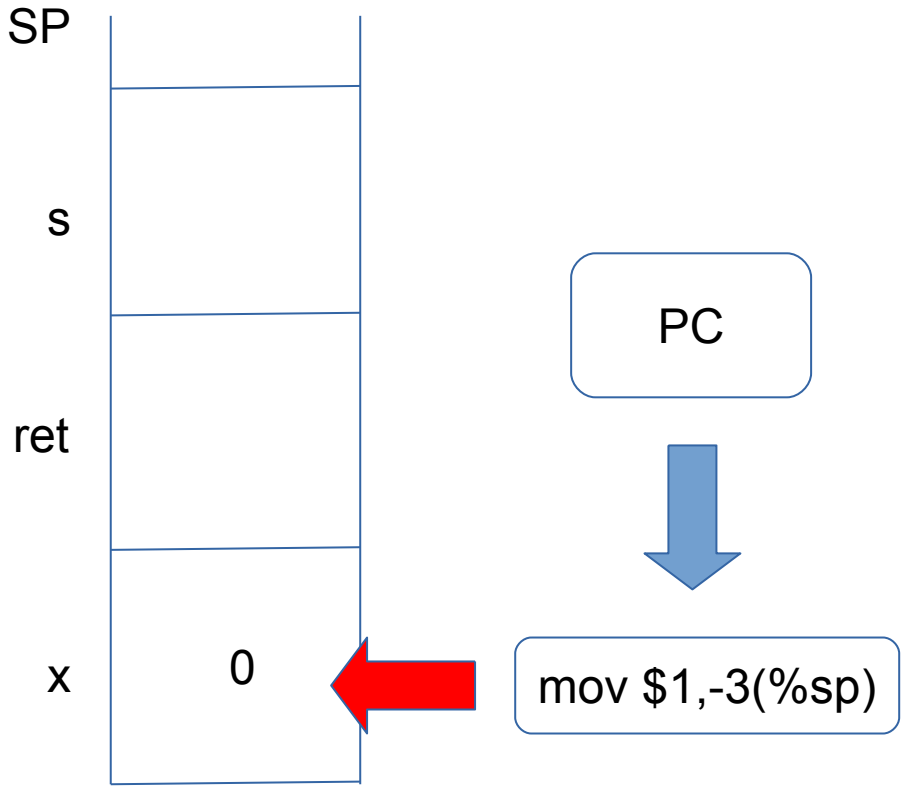
Security Properties for Stack Safety

We know stack un-safety when we see it

```
g() {
    int s;
    *(&s-2) = 1;
}

f() {
    int x = 0;
    g();
    assert(x==0);
}
```

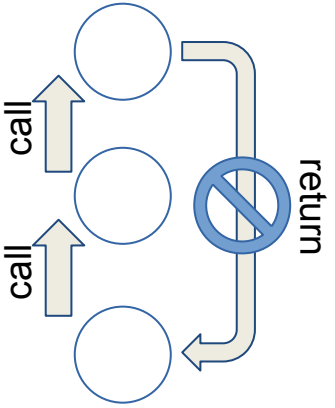


But we define stack safety as...

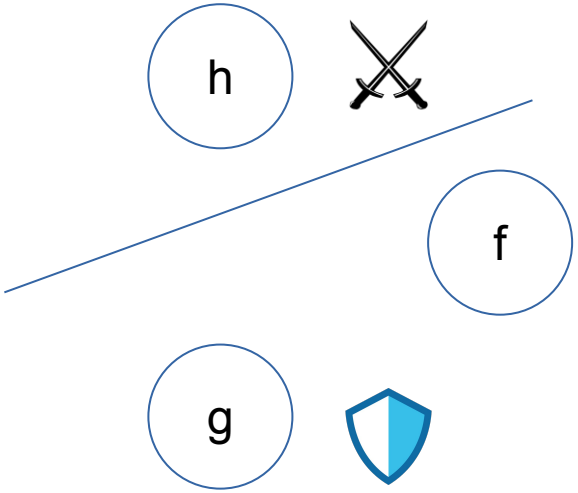
Confidentiality

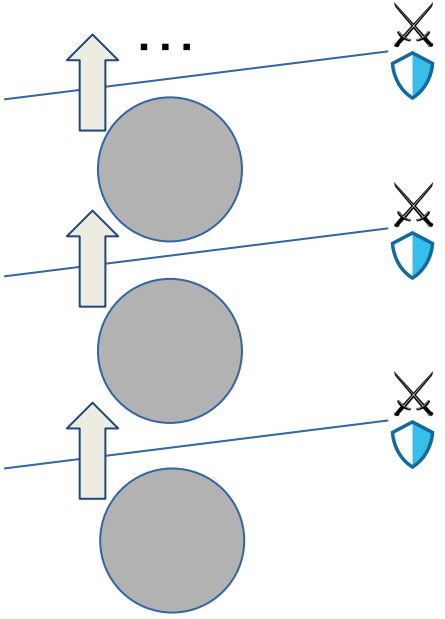
Integrity

Well-bracketedness

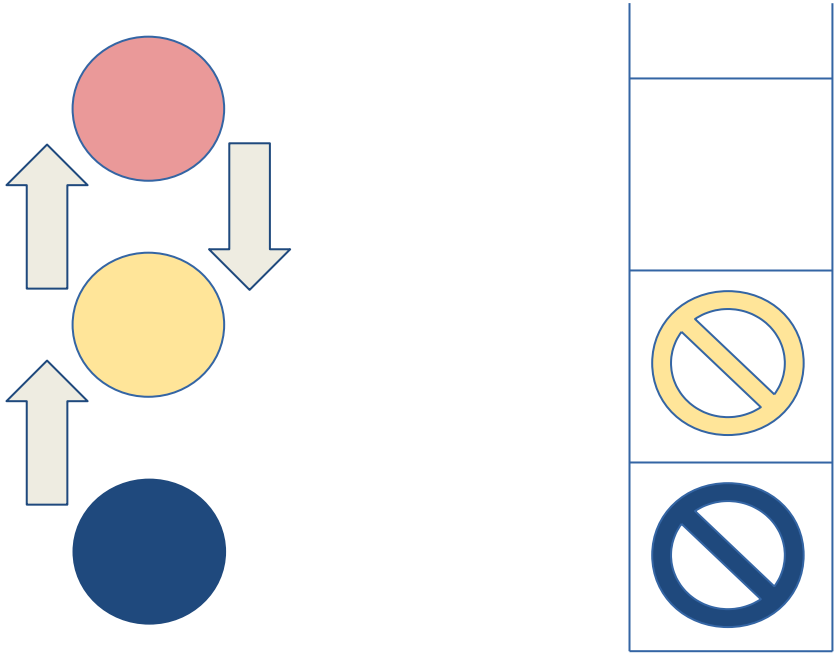


Every Caller Deserves Stack Safety

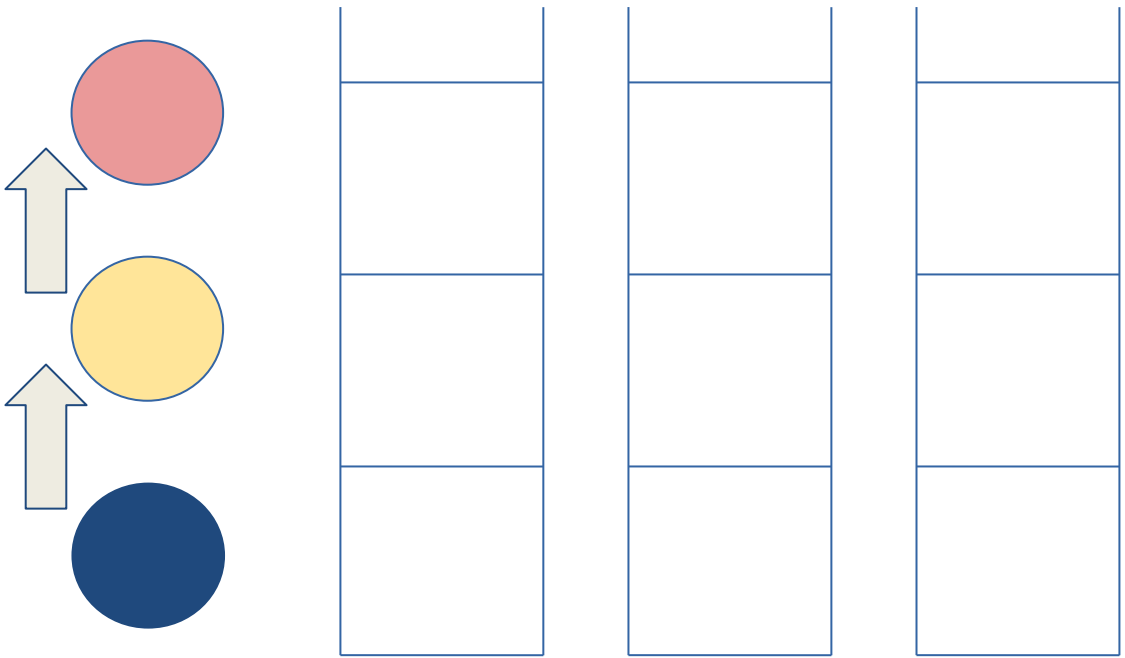




Integrity



Confidentiality is non-interference



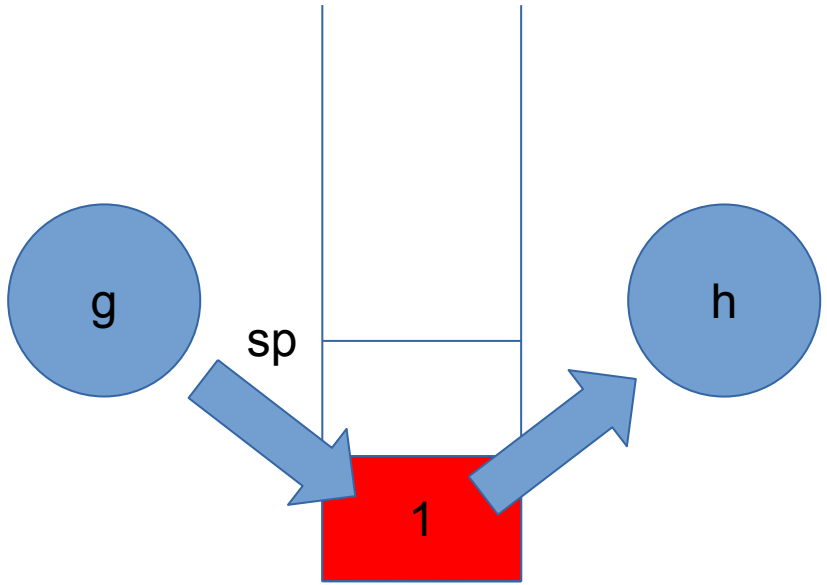
I haven’t done the animating yet, but the idea is:
each of these calls will have a variant stack and I’ll
talk about how we use noninterference in a nested setting.

Testing

- Randomized property based testing with Quickchick
- Tested Roessler and DeHon's "depth isolation" micro-policy successfully
- Roessler and Dehon's "lazy tagging" fails tests as expected

Lazy Tagging Leaks

```
f() {
    int x = 0;
    g();
    h();
}
g: ...
   mov $1,-2(%sp)
h: ...
   mov -2(%sp),r1
```



Additional Features

- Call-by-reference and stack-allocated call-by-value
- Simple coroutine model
- Observational property variants

Ongoing and Future Work

- Testing Cheri-esque capability models
- Expanding tests to handle arguments, observational properties
- Low-level separation logic

See our preprint – link