Questions

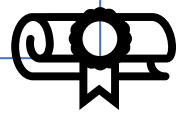
Nobody has responded yet.

Hang tight! Responses are coming in.

Can we change the way we program?

- Prove programs correct!

```
i = 1;
while(i <= n){
  r = r * i;
  i++;
}
```



This course:

How can we write these proofs?

How can we write programs so they're easier to prove?

- This program will:
  - never overflow its stack
  - never dereference a null pointer
  - never call a function without meeting its preconditions
  - always return the right result!

What did we learn?

- Introduction to theorem proving with Coq/Rocq
  - Notation, pattern-matching and recursion, tactics, proof strategies

- Separation logic with Iris
  - Basic separation logic and tactics
  - Specs for programs: pre- and postconditions
  - Data structures
  - Concurrency: invariants, threads, locks, ghost state
  - Iris for real programming languages

What *should* we prove correct?

- Program verification is a lot of work!

- Where is it worth the effort?
  - Tradeoff: how bad would it be if something went wrong?
  - Critical infrastructure – safety, security
  - High-value code
  - Widely-used code, e.g. libraries

Can it be less work?

- Program verification is a lot of work!

- Just stating a *specification* is already an important step
  - What does this function always do? What should be true in the caller before it's called?
  - Just writing this down in the code can help correctness!

- Spectrum of tools between Coq proofs and just writing a spec:
  - Lightweight formal methods: Dafny, Verifast, …
  - Even lighter-weight formal methods: Infer, …

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.