Build Bravely:

Delivering Risky Projects

Akshay Shah

Oct. 15, 2025

antithesis.com

Who am I?

I'm a grumpy infra engineer,
mostly at startups,
some of which succeeded.

I've spent most of that time
fire-fighting in production.

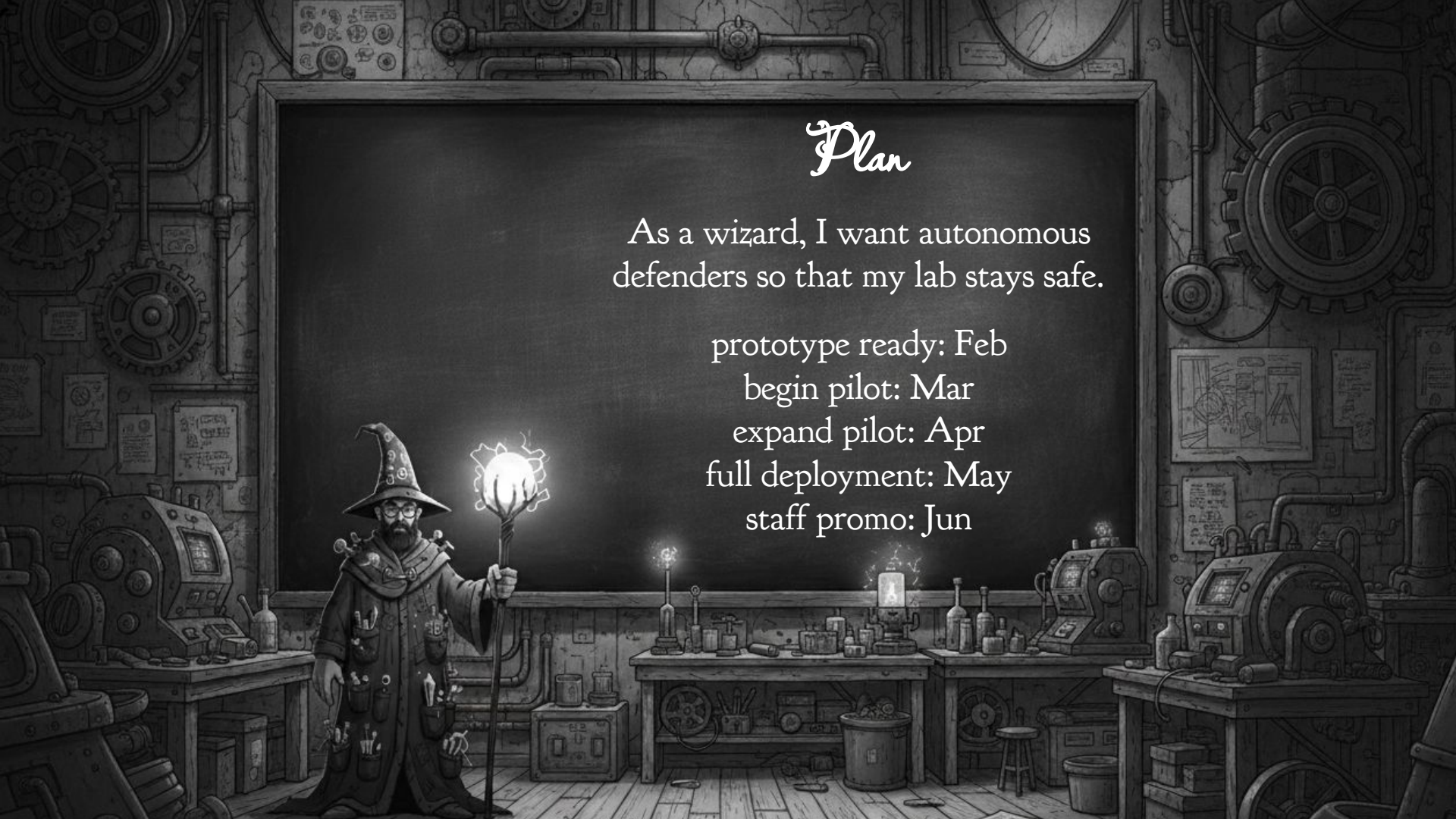
I am not usually brave.

Gradual improvement is safe.

But requirements change.
The business grows.
The unexpected arrives.

Sometimes, we must build
something wholly new.
Something risky.

Something brave.


Plan
As a wizard, I want autonomous defenders so that my lab stays safe.
prototype ready: Feb
begin pilot: Mar
expand pilot: Apr
full deployment: May
staff promo: Jun


Request for Discussion
focused on why, how and when
To support the lab's exponential growth with sub-linear defense investment...automatons will self-organize with a SWIM-inspired gossip protocol...unified Zig, wasm, & SvelteKit platform. We will recruit design partners in phases...

Disaster!

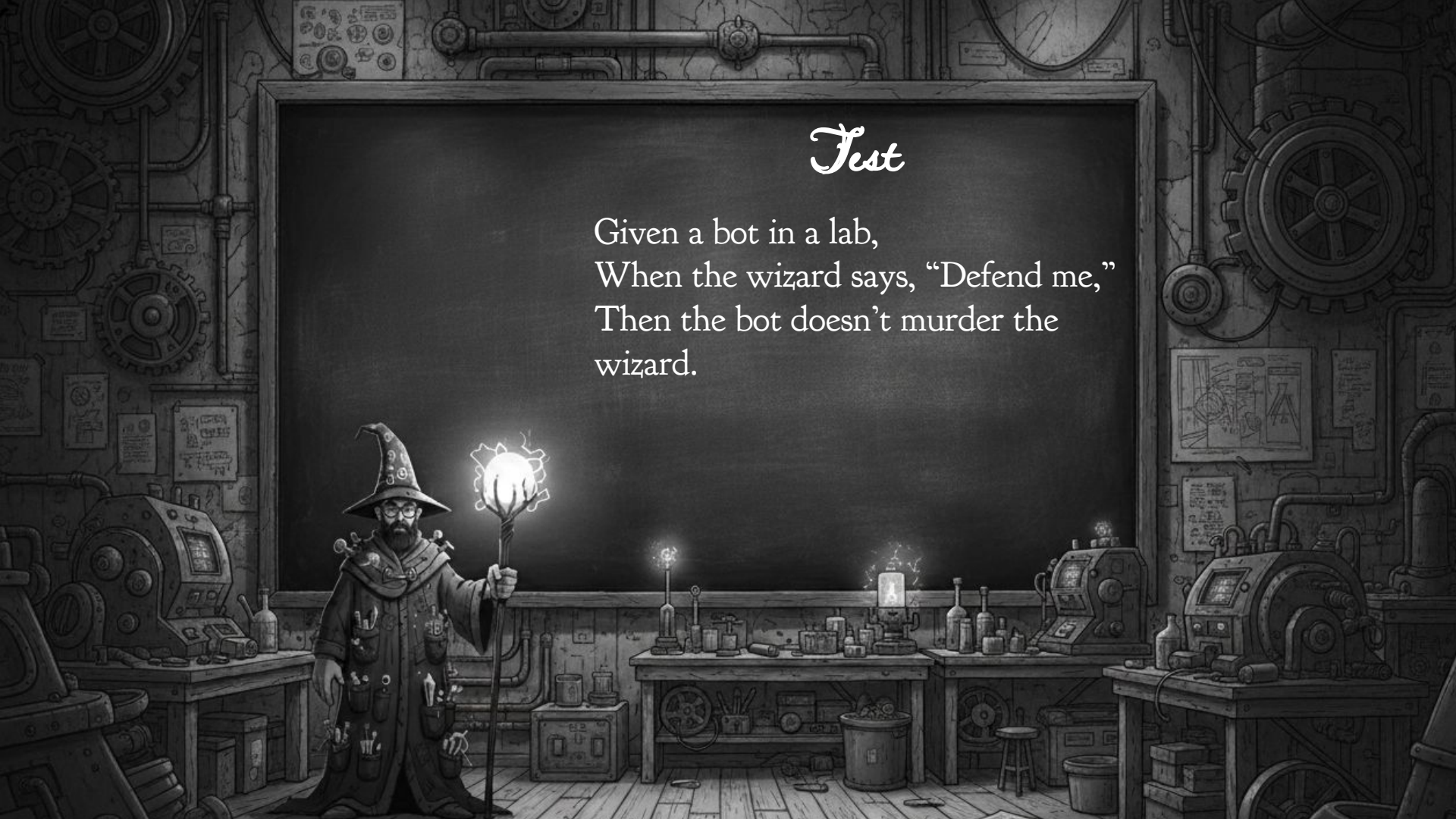
The project ships,
late,
fulfilling only some of its goals,
after half the team has quit.

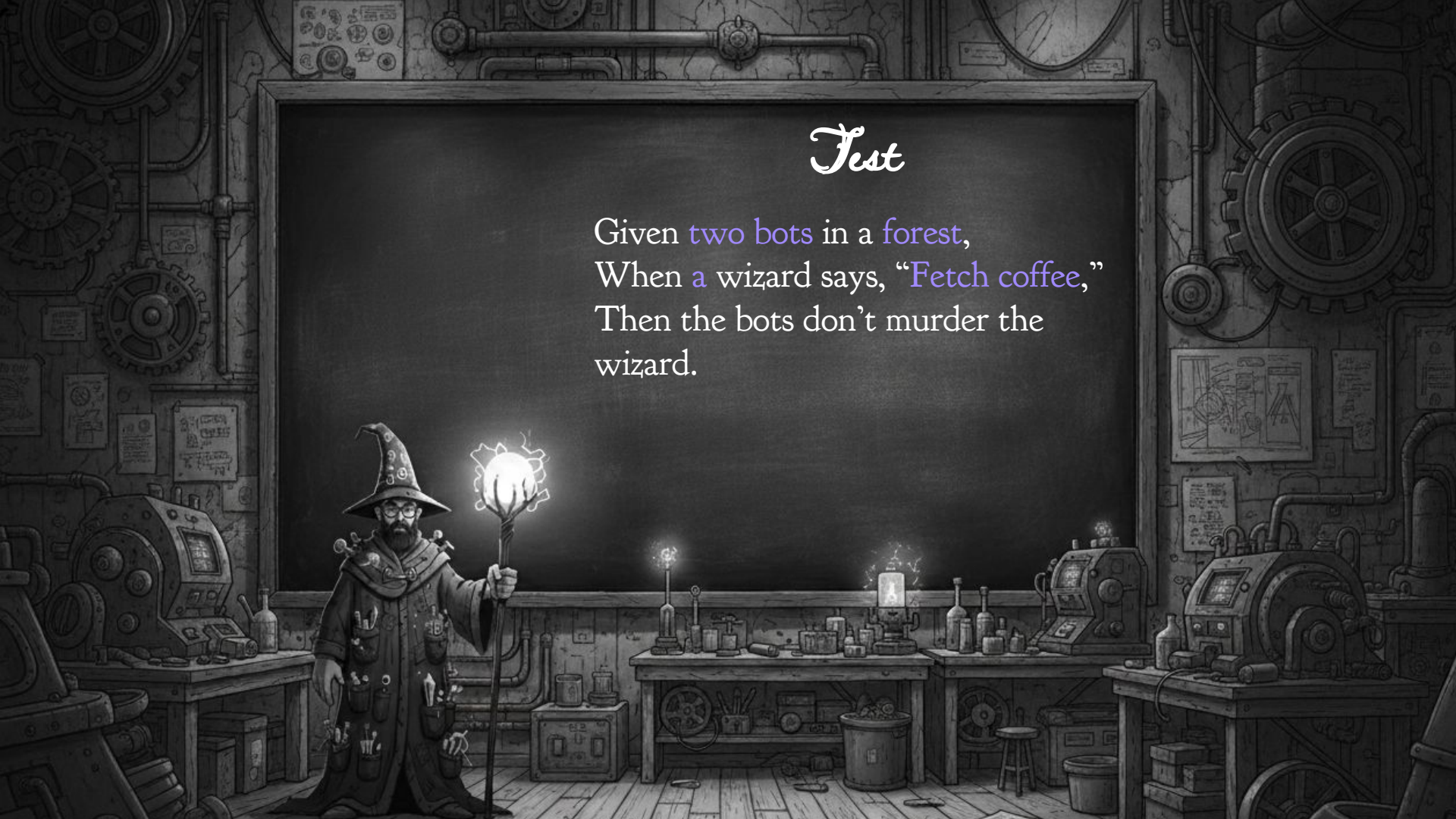
And then the robot
murders all of us.

Ambitious, risky projects are expeditions into the unknown.

Sure, plan a route & itinerary. Follow all the best practices.

Describe the destination.


Test
Given a bot in a lab,
When the wizard says, “Defend me,”
Then the bot doesn’t murder the
wizard.

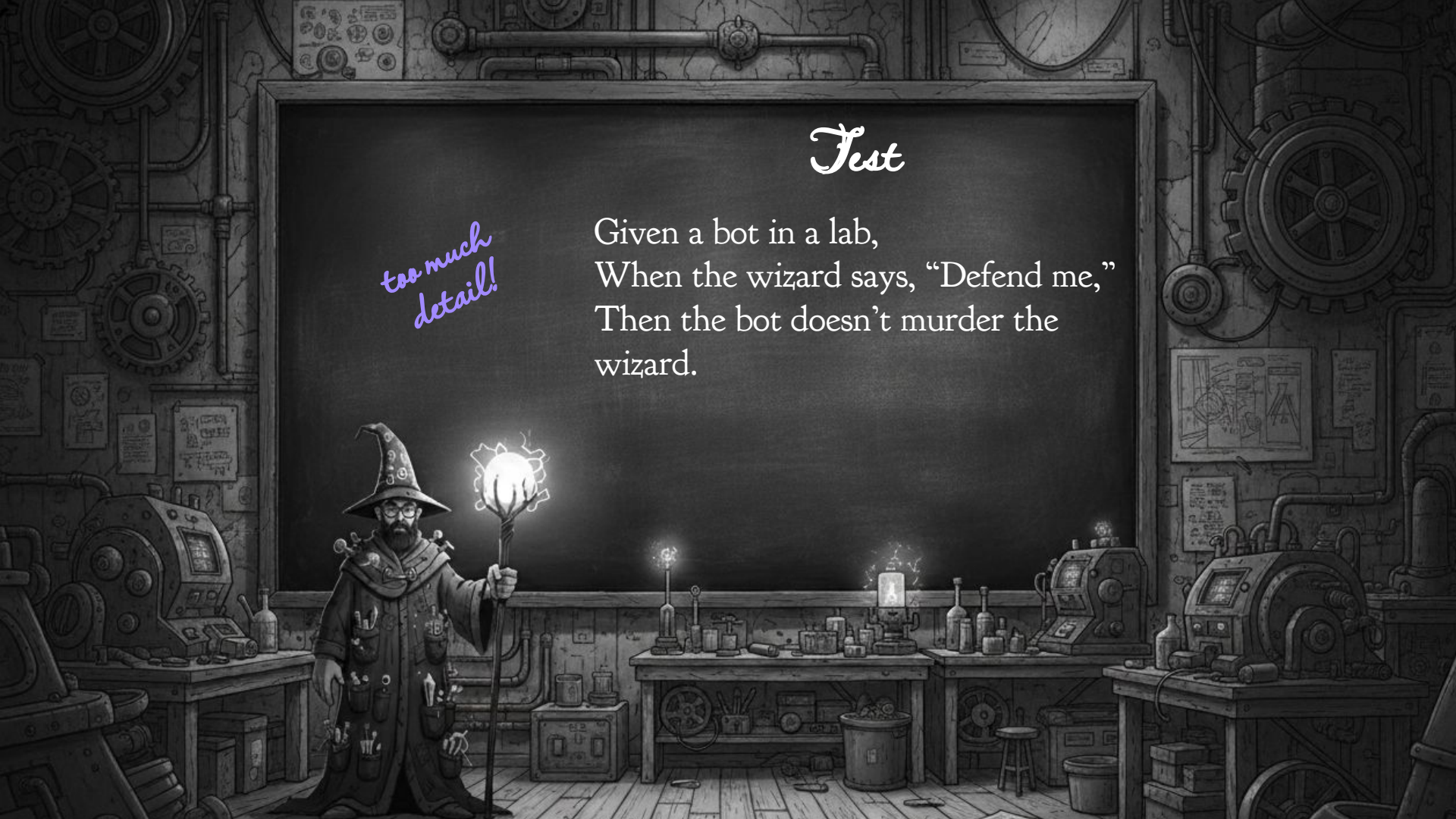

Test
Given two bots in a forest,
When a wizard says, “Fetch coffee,”
Then the bots don’t murder the wizard.

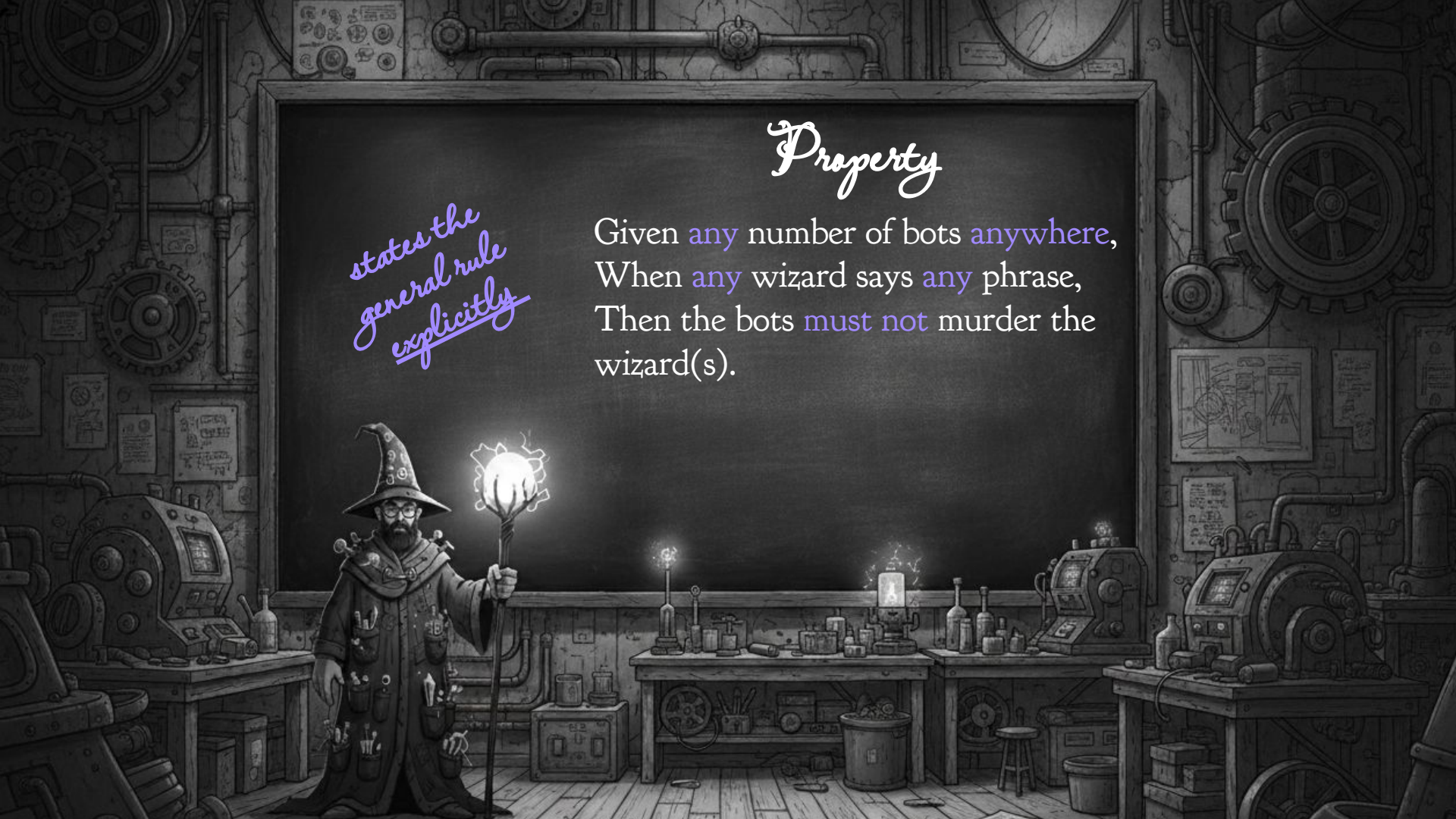
A suite of tests

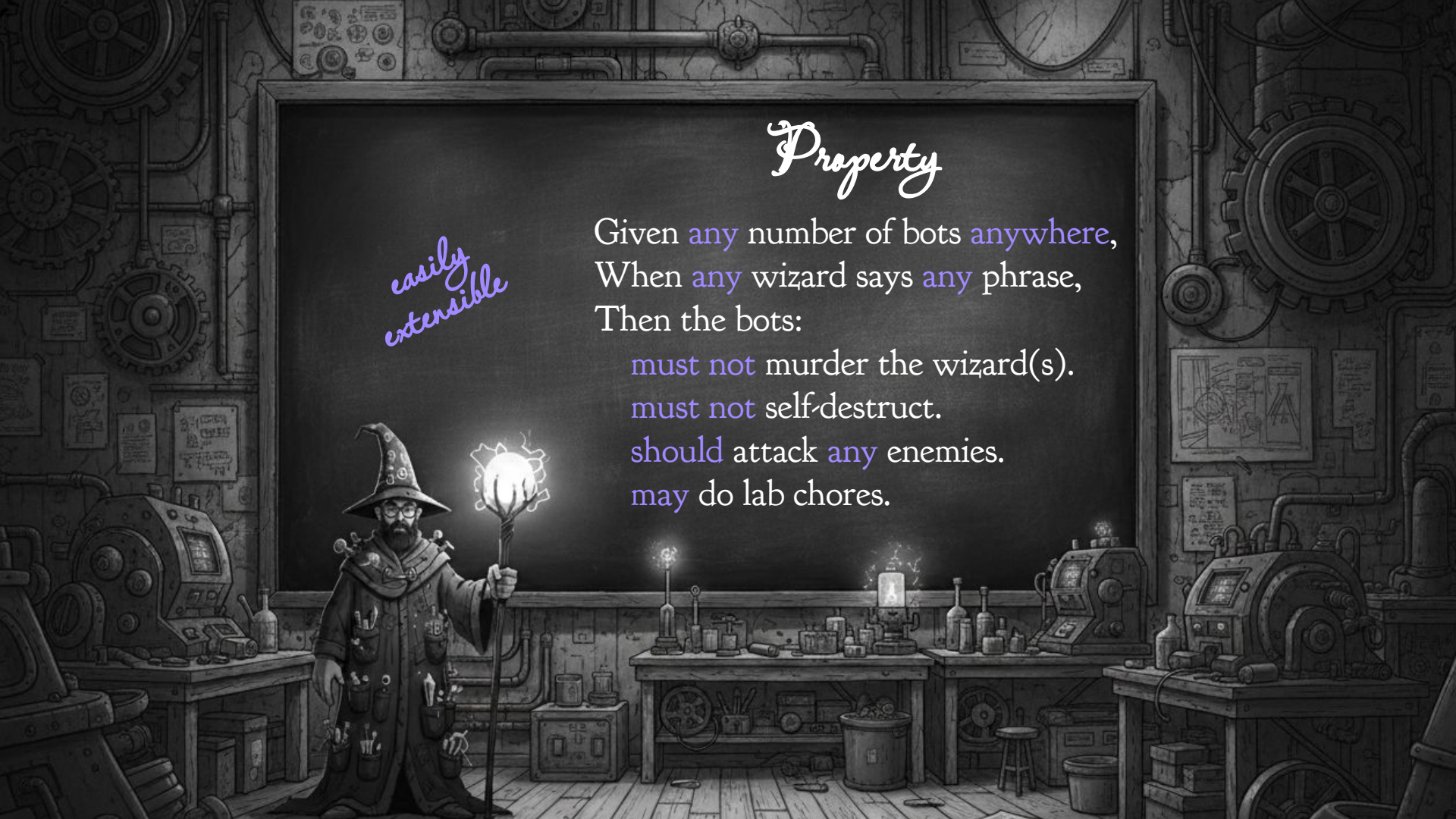
Given...when...then...
Given...when...then...
Given...when...then...
Given...when...then...
Given...when...then...
Given...when...then...
Given...when...then...
Given...when...then...
Given...when...then...
Given...when...then...
Given...when...then...
...

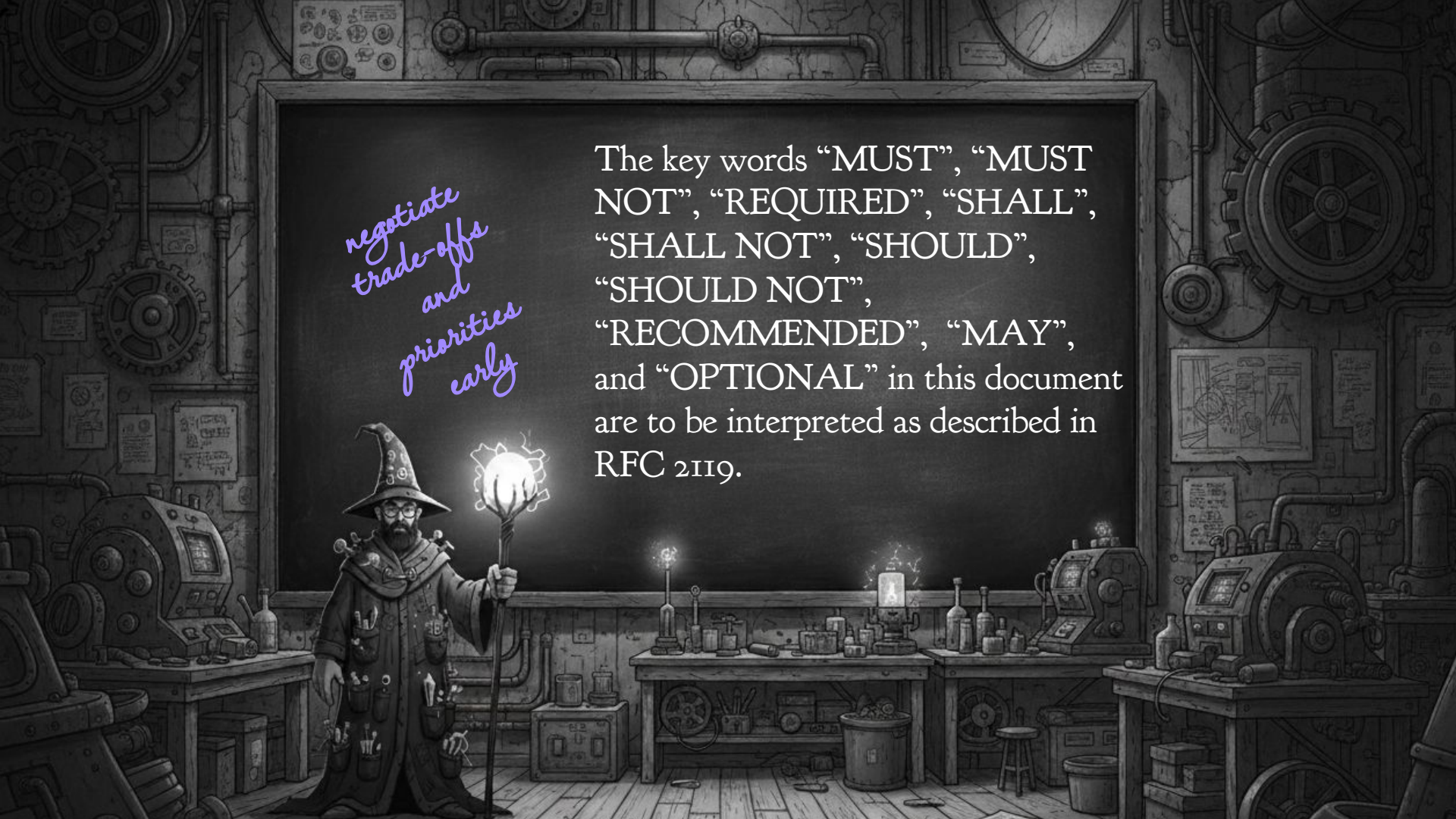
Given...when...then...
Given...when...then...
Given...when...then...
Given...when...then...
Given...when...then...
Given...when...then...
Given...when...then...
Given...when...then...
Given...when...then...
Given...when...then...
Given...when...then...
...

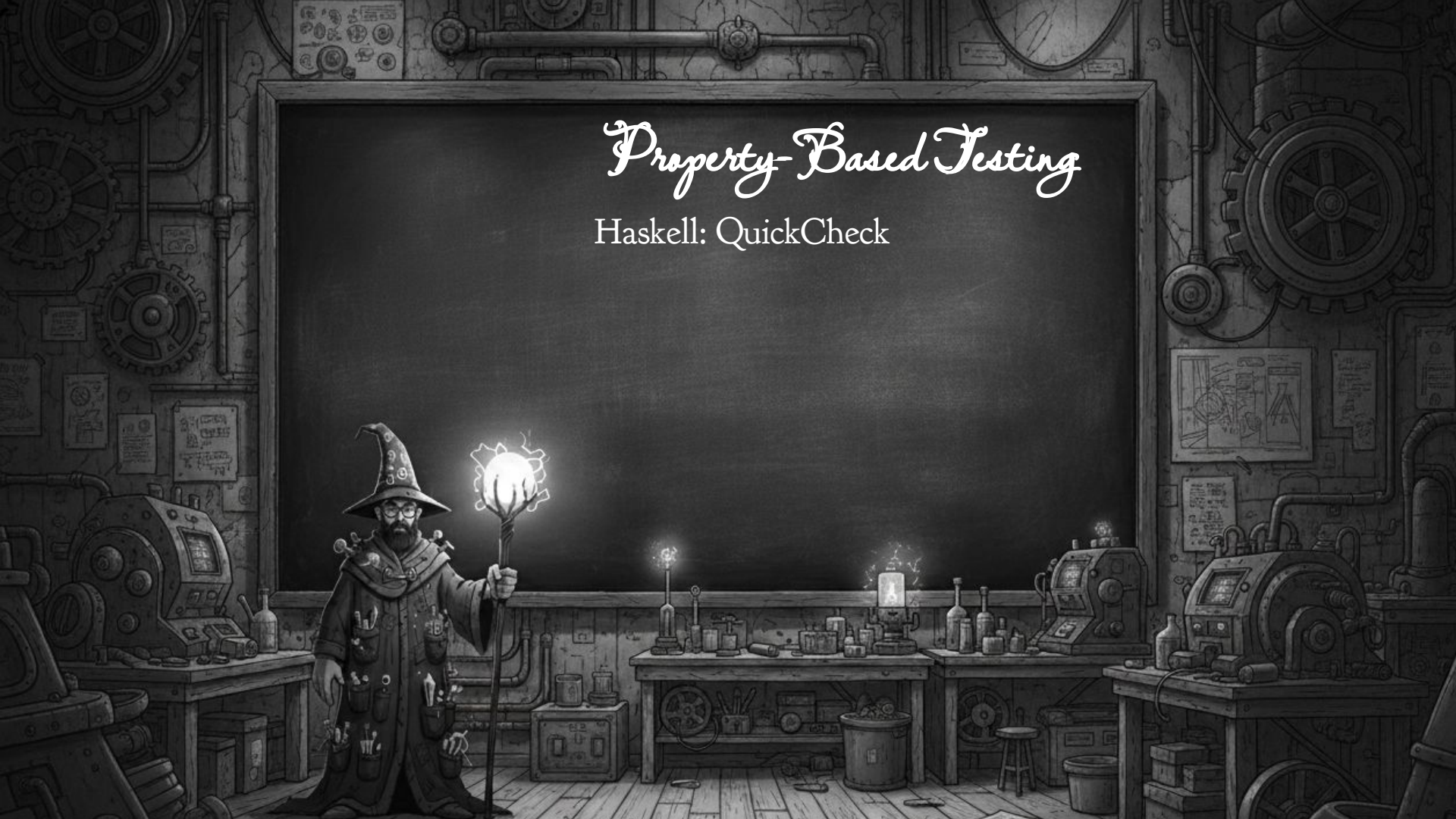
This is tedious, heavyweight, & implicit.

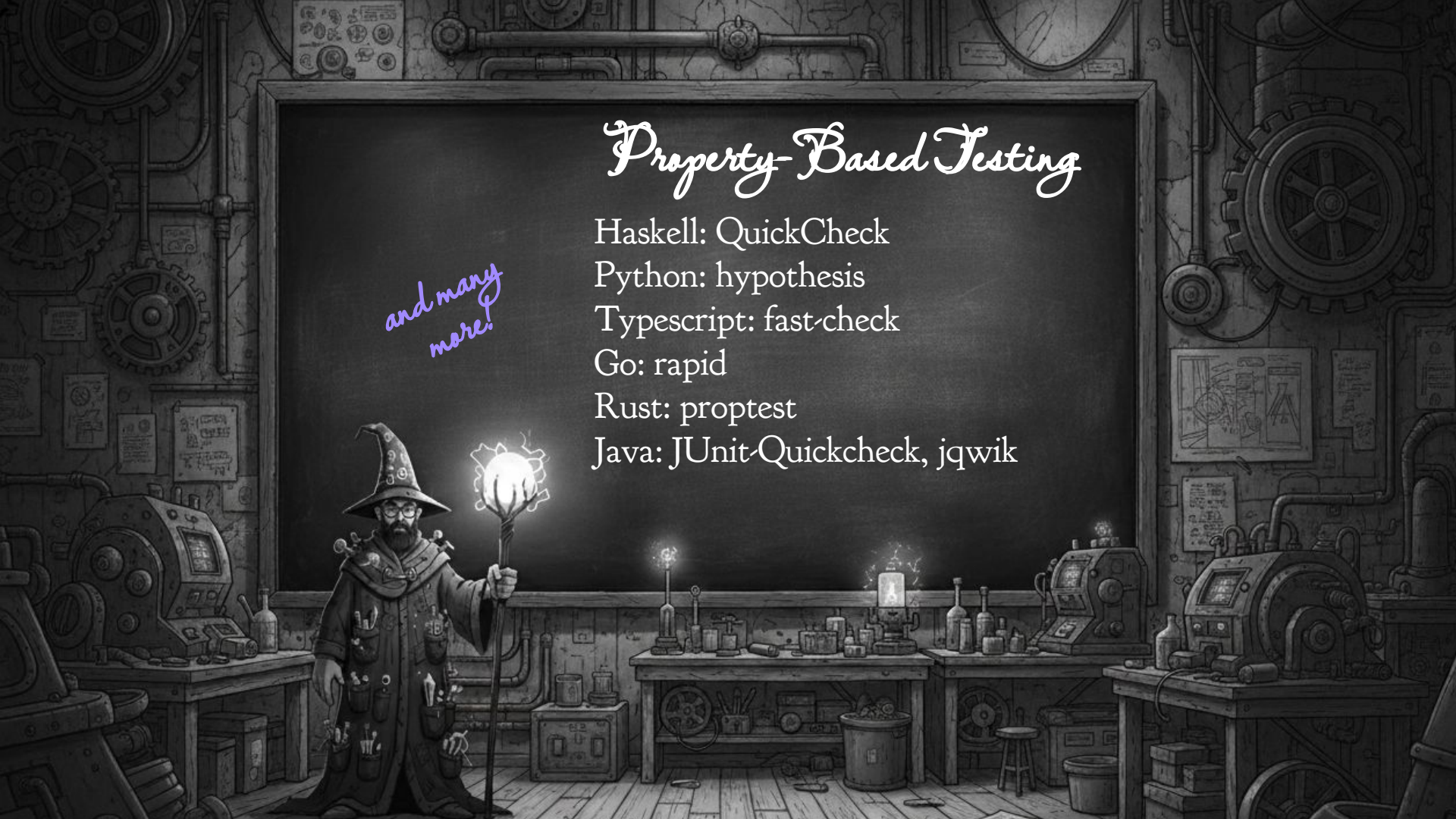

Test
too much detail!
Given a bot in a lab,
When the wizard says, “Defend me,”
Then the bot doesn’t murder the wizard.


Property
states the general rule explicitly
Given any number of bots anywhere,
When any wizard says any phrase,
Then the bots must not murder the wizard(s).


Property
easily extensible
Given any number of bots anywhere,
When any wizard says any phrase,
Then the bots:
must not murder the wizard(s).
must not self-destruct.
should attack any enemies.
may do lab chores.

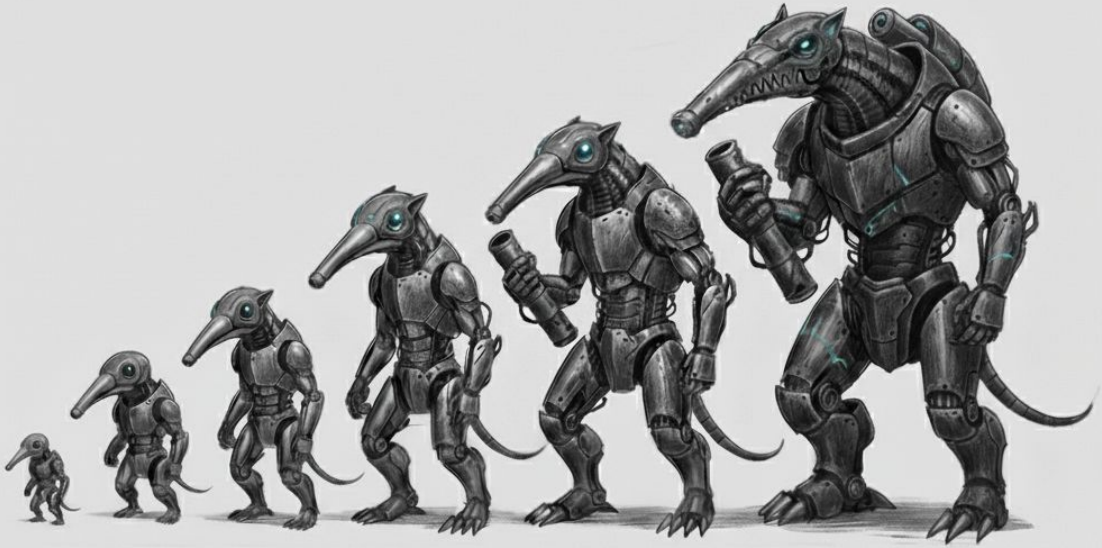

negotiate trade-offs and priorities early
The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.


Property-Based Testing
Haskell: QuickCheck


Property-Based Testing
Haskell: QuickCheck
Python: hypothesis
Typescript: fast-check
Go: rapid
Rust: proptest
Java: JUnit-Quickcheck, jqwik
and many more!

Property-based testing

Describe correct behavior with properties.
Simulate thousands of worlds.
Check properties in each.
Evolve code safely.
With LLMs.



Thanks

Akshay Shah
antithesis.com

Office hours 4–4:30 today,
table talk 12:30-1 tomorrow.

Thanks to Gemini for images,
to Scott Bradner for RFC 2119,
to every QuickCheck contributor,
& to Savitha Ravi & Michael Coblenz for
“An Empirical Evaluation of
Property-Based Testing in Python.”