The SOFTWARE ESSENTIALIST

FA^{2}STR — How to Decompose, Architect, Design, Implement & Test Anything On Any Side of the Stack

Let’s see where all of this leads. Remember: we’re developing your problem decomposition skills. This is the main skill.

The Ideal Developer Workflow is great, but...

Examples:

- How do I handle playlist re-ordering? How do I design the feature on the backend?
- How do I design auth on the frontend properly?
- Basically… when a problem seems out of your wheelhouse just because you haven't done that particular thing before.
  - You still need to have a way to think through how you'd do it — and you want to feel confident in your thinking

How do you consistently go from

A non-trivial design challenge →

To

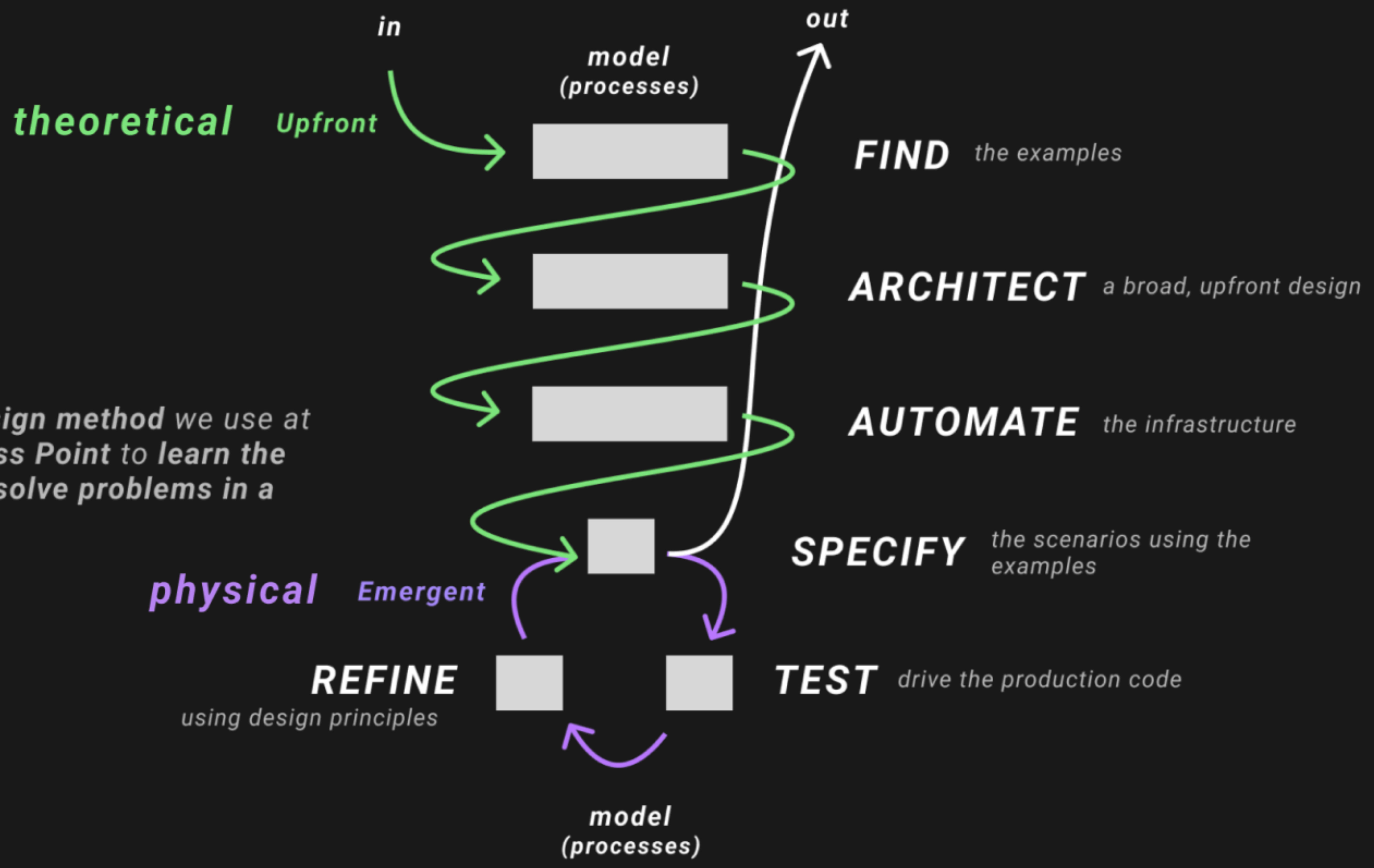
Decomposed action steps →

Arch, design, implemented, tested code →

This community needs a framework (or **design method**) to think **through, communicate**, and **consistently** decompose and solve design challenges.

FA^{2}STR

The repeatable **design method** we use at each **Physical Guess Point** to **learn the 12 Essentials** and **solve problems in a consistent way**.



in

out

model
(processes)

theoretical Upfront

FIND the examples

ARCHITECT a broad, upfront design

AUTOMATE the infrastructure

SPECIFY the scenarios using the examples

physical Emergent

REFINE using design principles

TEST drive the production code

model
(processes)

What is the FA^{2}STR Design Method?

- ***The culmination of everything I know about problem decomposition using the 12 Essentials***
- ***The best of TDD, BDD, RDD.***
- ***A way to remember the steps of The Ideal Developer Workflow***
- ***A way to think through unique design challenges and end up with testable code***
- ***A way to work Outside-In, guiding your work with tests at each level of The Guess Points***

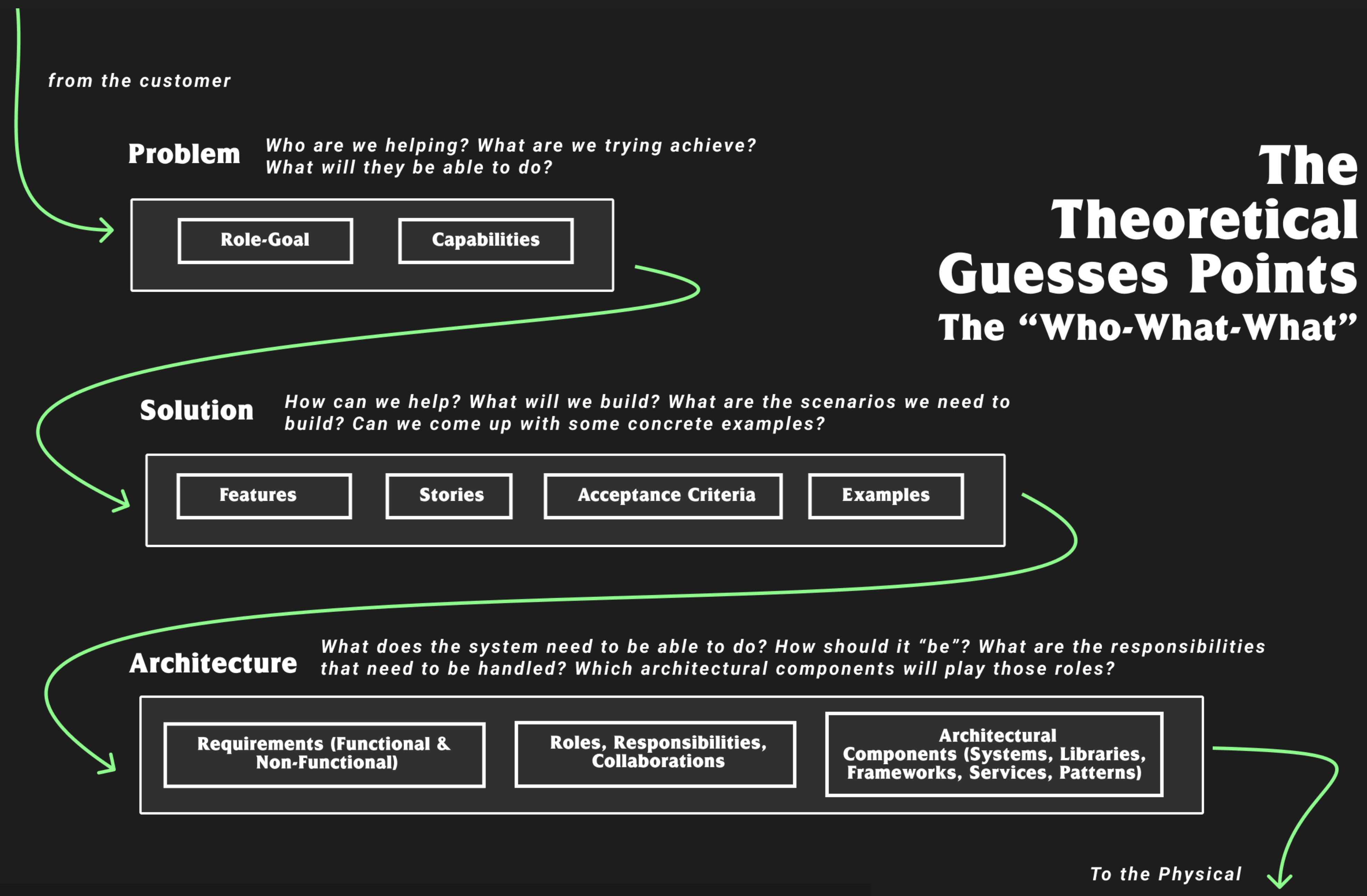
What does it stand for?

- ***FIND the theoretical guesses & get concrete examples***
- ***ARCHITECT the broad stroke solution***
- ***AUTOMATE the system under test infra***
- ***SPECIFY the acceptance criteria***
- ***TEST the implementation***
- ***REFINE the implementation***

Example: Notion to Google Calendar Sync

Find

- *Who, What, Why, Concrete Examples*
- *Scenarios:*
  - *No Change*
  - *New Tasks*
  - *New Calendar Event*
  - *Task Completed Before Event*
- *Examples for each*



The Theoretical Guesses Points
The “Who-What-What”

from the customer

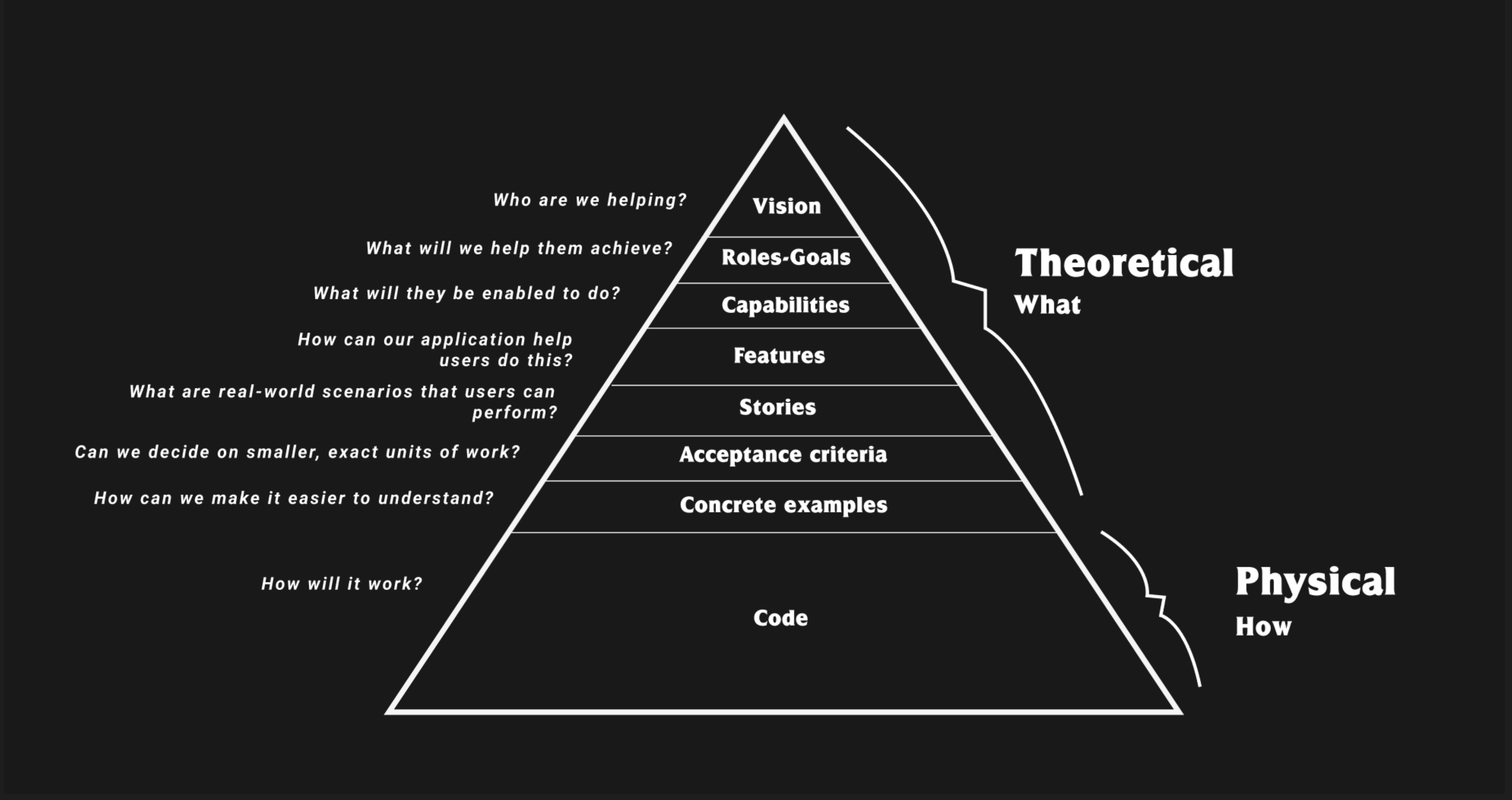
Problem *Who are we helping? What are we trying achieve? What will they be able to do?*

Solution *How can we help? What will we build? What are the scenarios we need to build? Can we come up with some concrete examples?*

Architecture *What does the system need to be able to do? How should it “be”? What are the responsibilities that need to be handled? Which architectural components will play those roles?*

Requirements (Functional & Non-Functional)
Roles, Responsibilities, Collaborations
Architectural Components (Systems, Libraries, Frameworks, Services, Patterns)

To the Physical

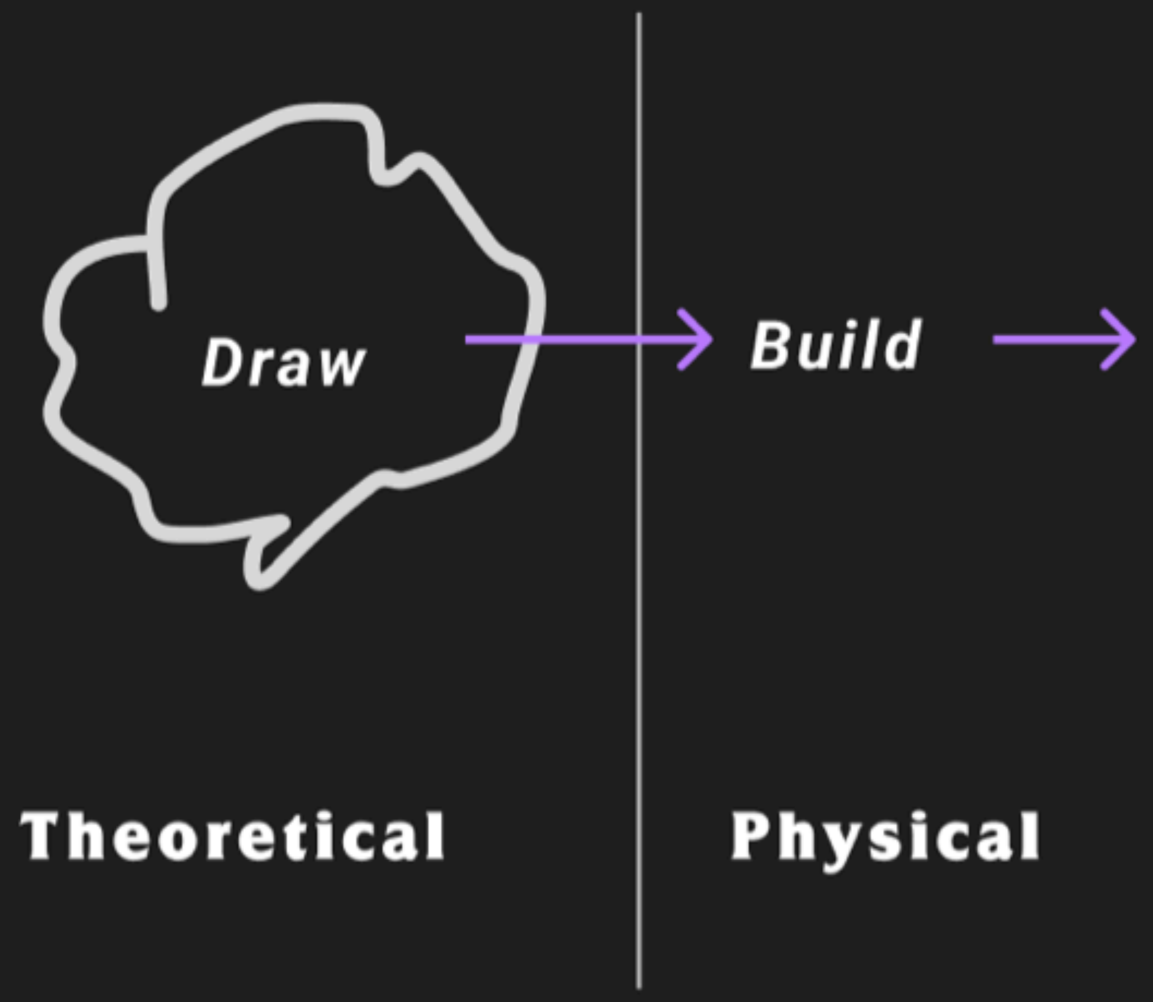


Who are we helping?
Vision
What will we help them achieve?
Roles-Goals
What will they be enabled to do?
Capabilities
How can our application help users do this?
Features
What are real-world scenarios that users can perform?
Stories
Can we decide on smaller, exact units of work?
Acceptance criteria
How can we make it easier to understand?
Concrete examples
How will it work?
Code
Theoretical
What
Physical
How

essentialist.dev

Architect

- *Determine the Patterns we'll use*
- *Determine any new Components we need*
- *Determine the Testing Strategy we'll use to test at which Guess Points*
- *If we already have an architecture, testing strategy, well established patterns & the necessary components, this is much easier, but we need to move through The Guess Points like a checklist*
- *Create a broad stroke architecture!*

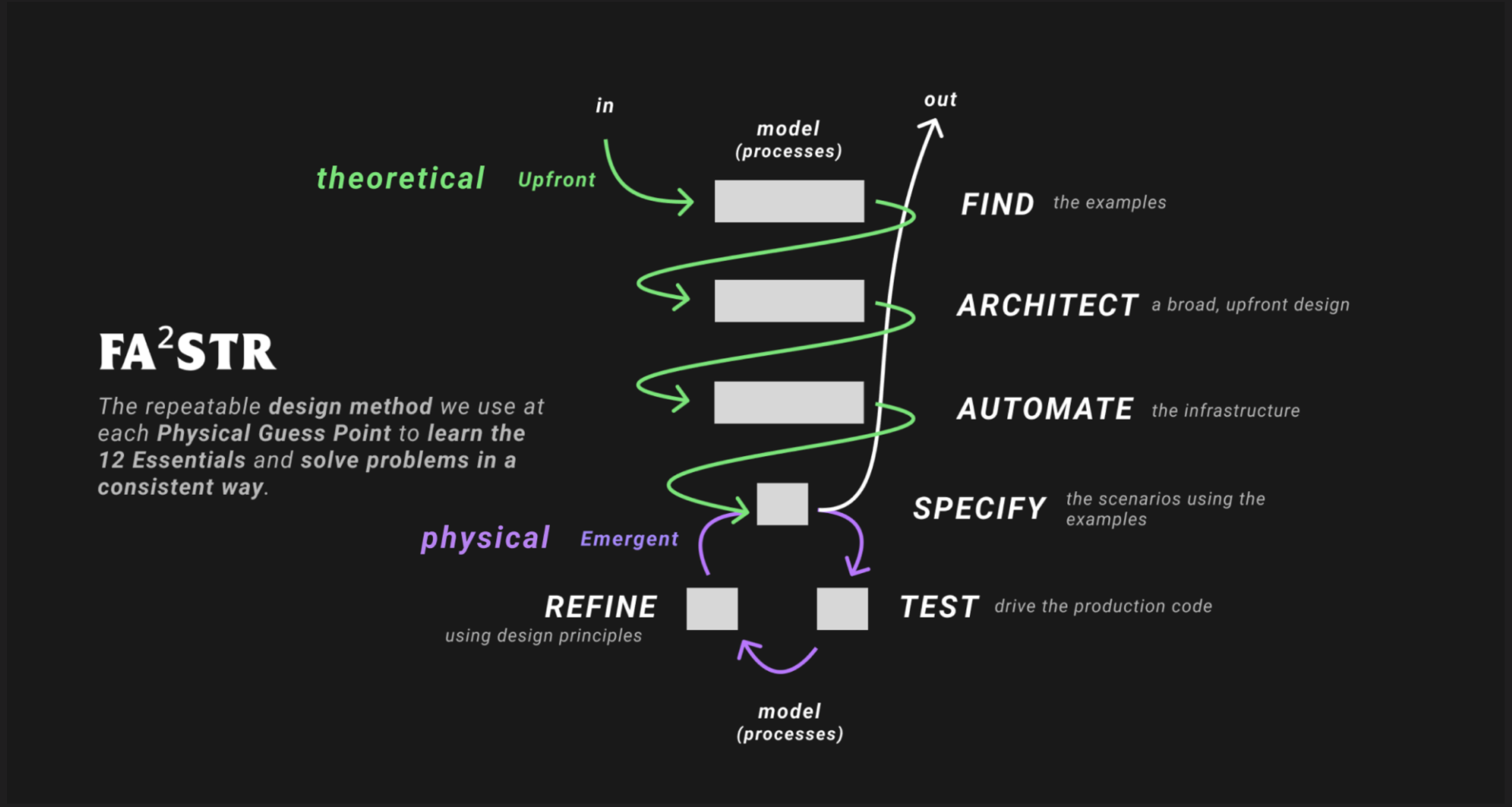


Automate

- Forces me to think through the test infrastructure and if I’m going to be able to verify the acceptance criteria (and what that even is) before I write any code
- Think Backwards; start at the end; define success before we start
- “How will I test this?”
- Make a decision

Specify-Test-Refine

- *If you're familiar with TDD, it's Red-Green-Refactor*
- *Write the acceptance criteria at whichever Guess Point we're currently working at, make it pass, then refine*
- *We'll discuss how this works in more detail in The Phases of Craftship*



FA^{2}STR

*The repeatable **design method** we use at each **Physical Guess Point** to **learn the 12 Essentials** and **solve problems in a consistent way**.*

in
model (processes)
out
theoretical
Upfront
FIND the examples
ARCHITECT a broad, upfront design
AUTOMATE the infrastructure
SPECIFY the scenarios using the examples
physical
Emergent
REFINE using design principles
TEST drive the production code
model (processes)

essentialist.dev

FA^{2}STR: a combination of what we need from design, testing, architecture & strategy in a single mental model

What we covered