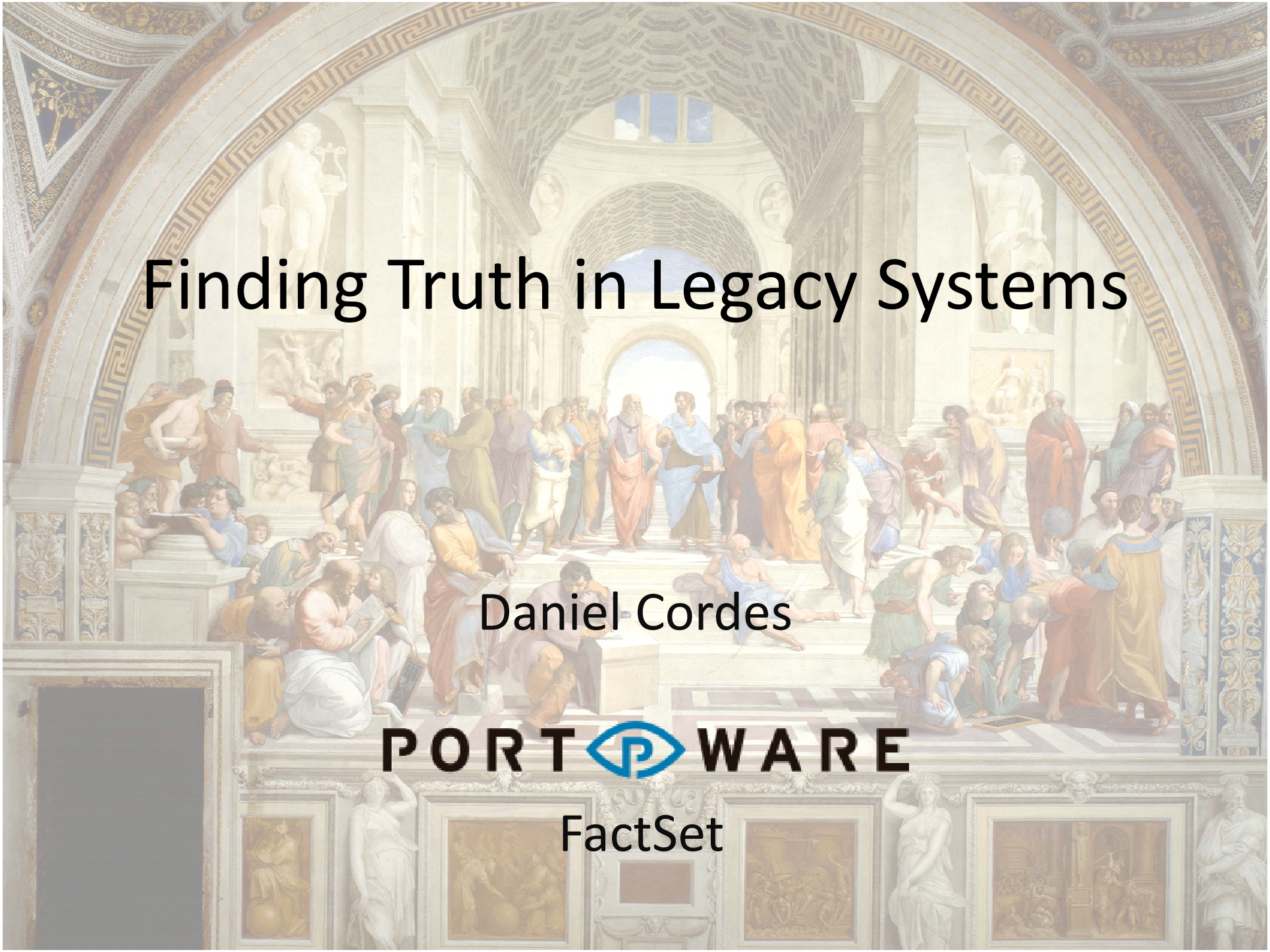
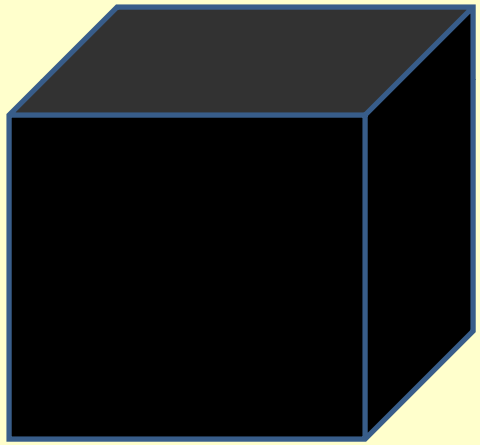


# Finding Truth in Legacy Systems

Daniel Cordes

PORTWARE

FactSet

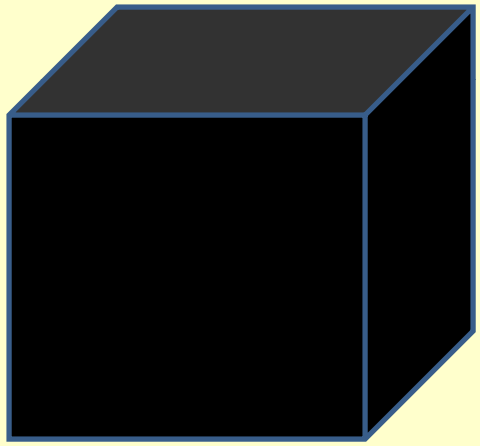







if you
SEE
something
SAY
something™
Report suspicious activity.
Call 720-913-2000
In case of emergency call 9-1-1
DENVER
THE MILE HIGH CITY
If You See Something Say Something™ use
thority.

# Let’s CD it!



Quid est veritas?

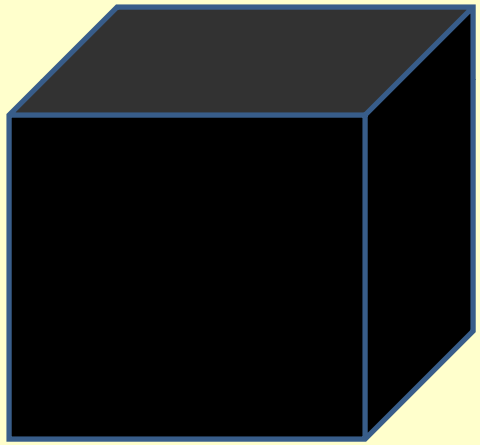





(Top-Down
Prescriptive Truth)
The SPEC
says
What SHOULD be TRUE


PRACTICAL NEEDS
CONTEXTS
TOOLS
ROLES
PROCESSES
EMOTIONS
SIMPLE AND EASY - BILL
FAMILY GARDEN - JANE
MICROWAVE - LIZ
COOKS FOR FAMILY - JANE
CONVENIENT - JANE
LOCAL MARKETS - BILL
FROZEN FISHSTICKS - BILL
FEEDS CHILDREN - SARAH
MUST BE CHEAP - LIZ
DORMITORY - LIZ
FROZEN VEGGIES - JANE
LIVES ALONE - BILL
QUICK AND HEALTHY - SARAH
A LOT OF FROZEN FOOD - SARAH
TV DINERS - BILL



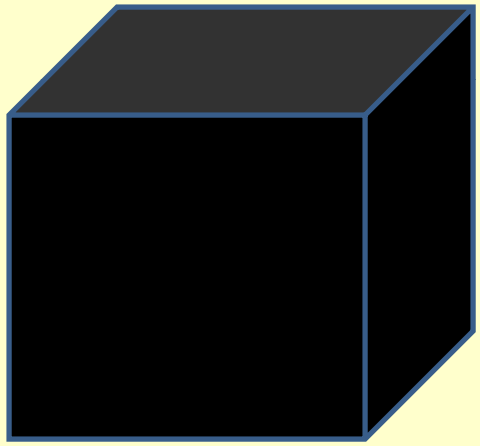




# Maybe it’s good enough as it is?

1) Hard to add new features

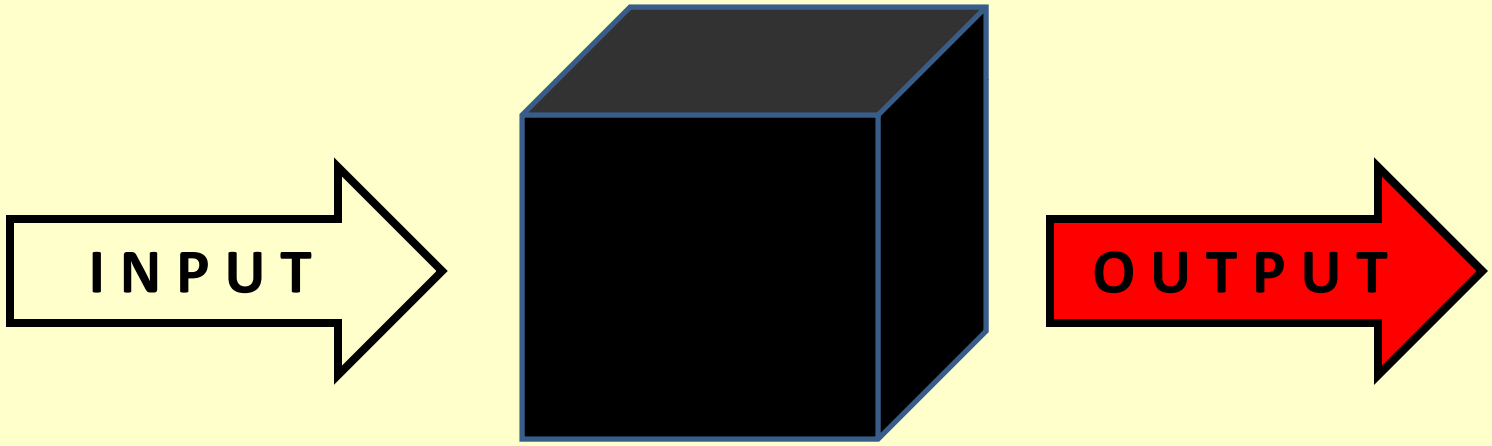
3) Hard/slow to deploy

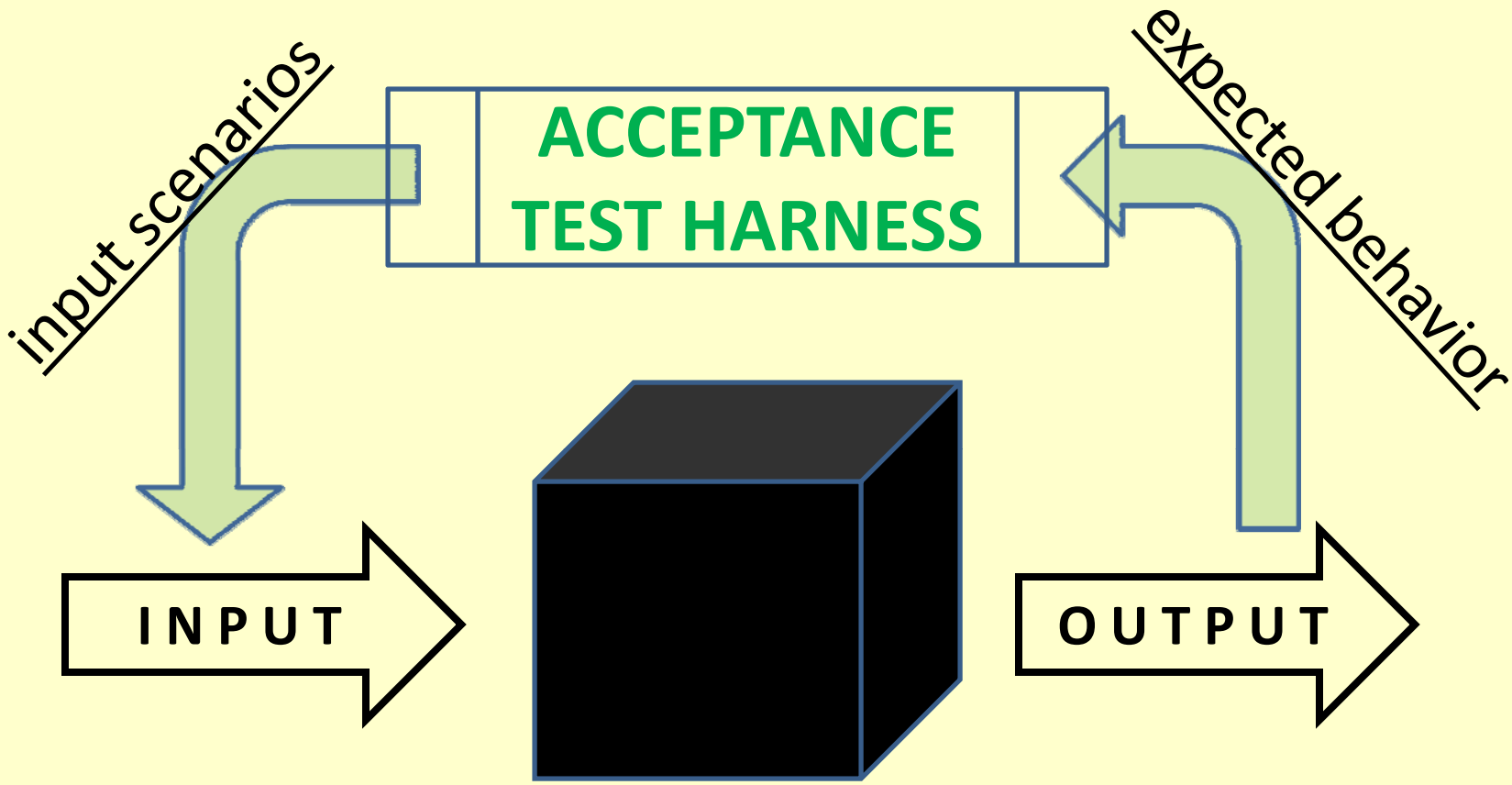


2) Hard to debug

4) One day it’s going to break

# Let’s CD it!

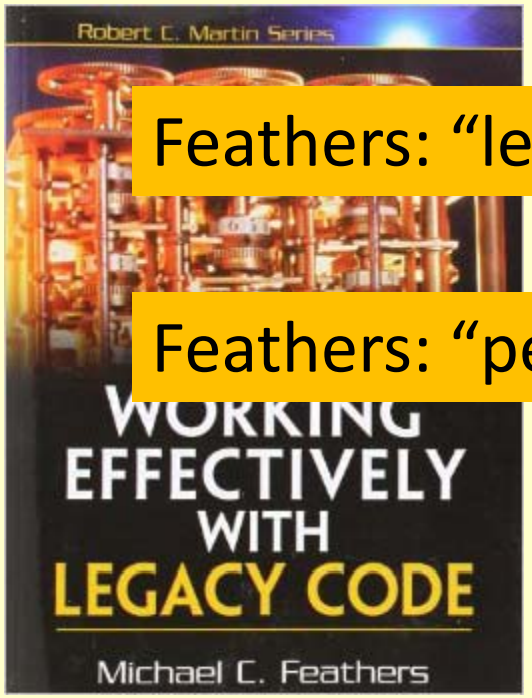


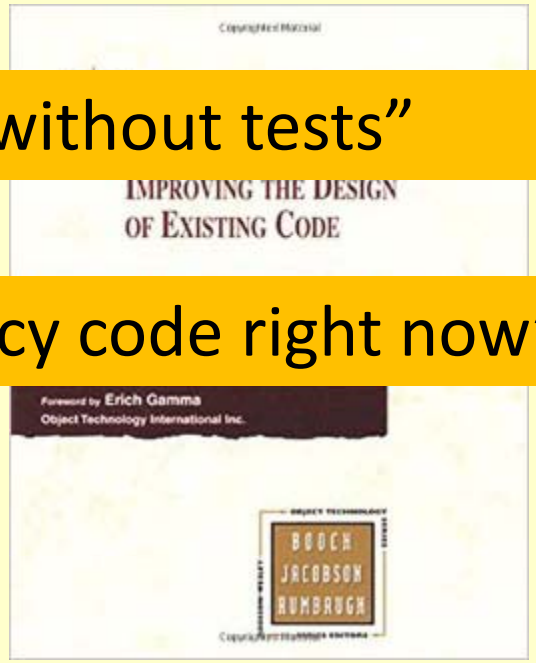


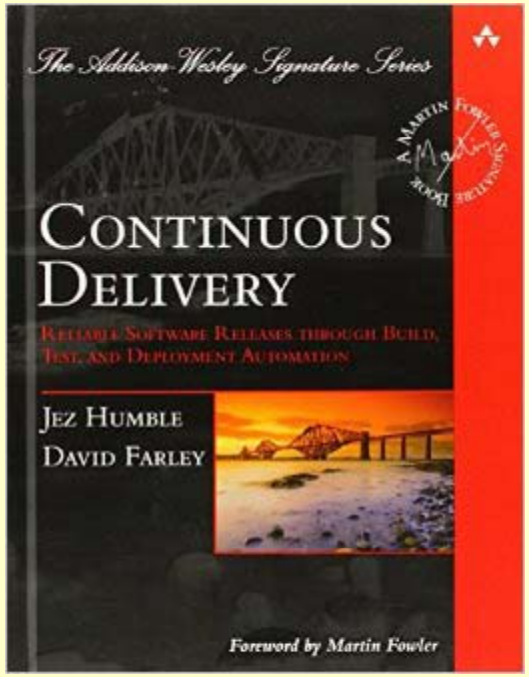

4) One day it's going to break

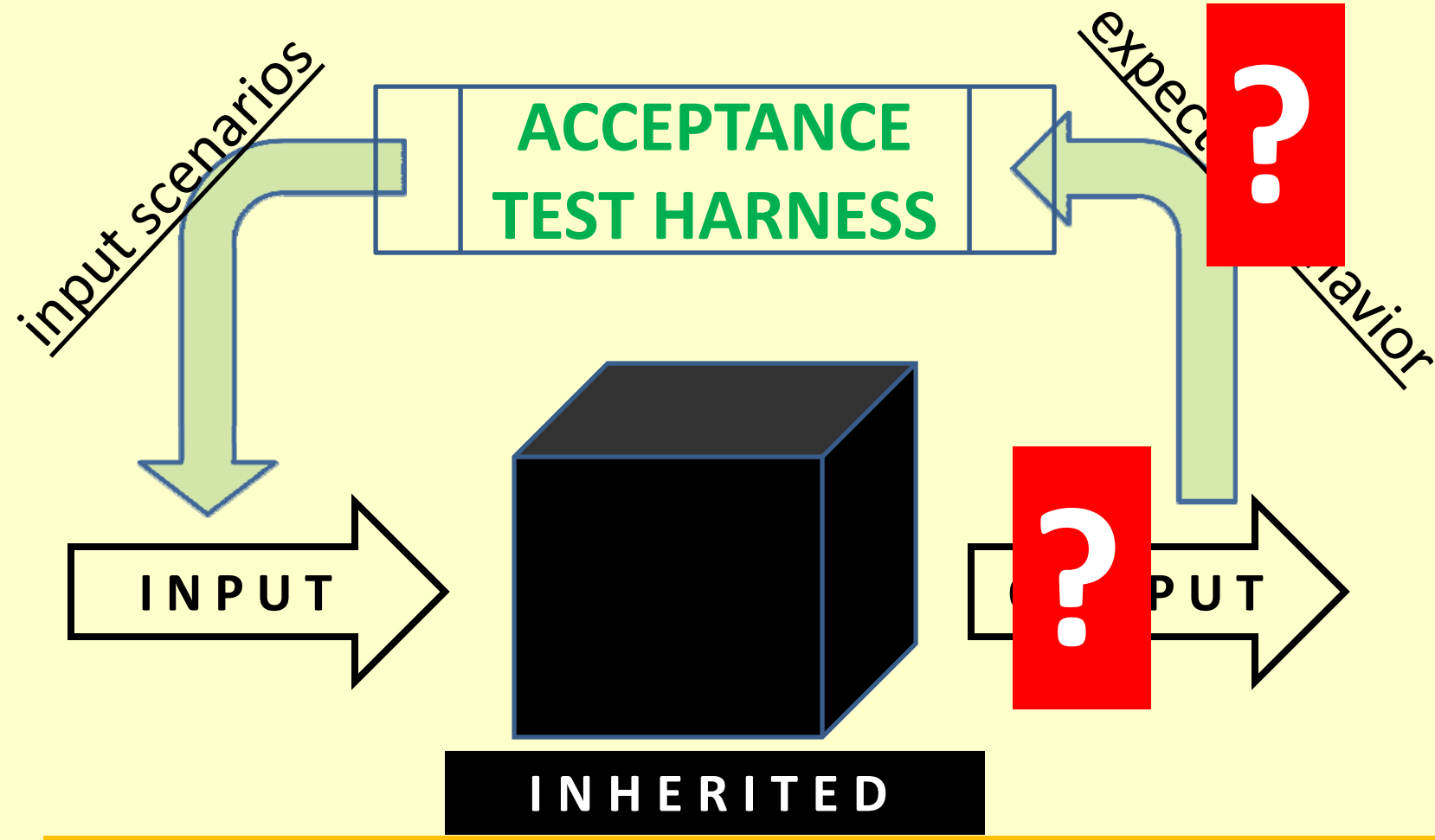
Feathers: “legacy code” == “code without tests”

Feathers: “people are writing legacy code right now”

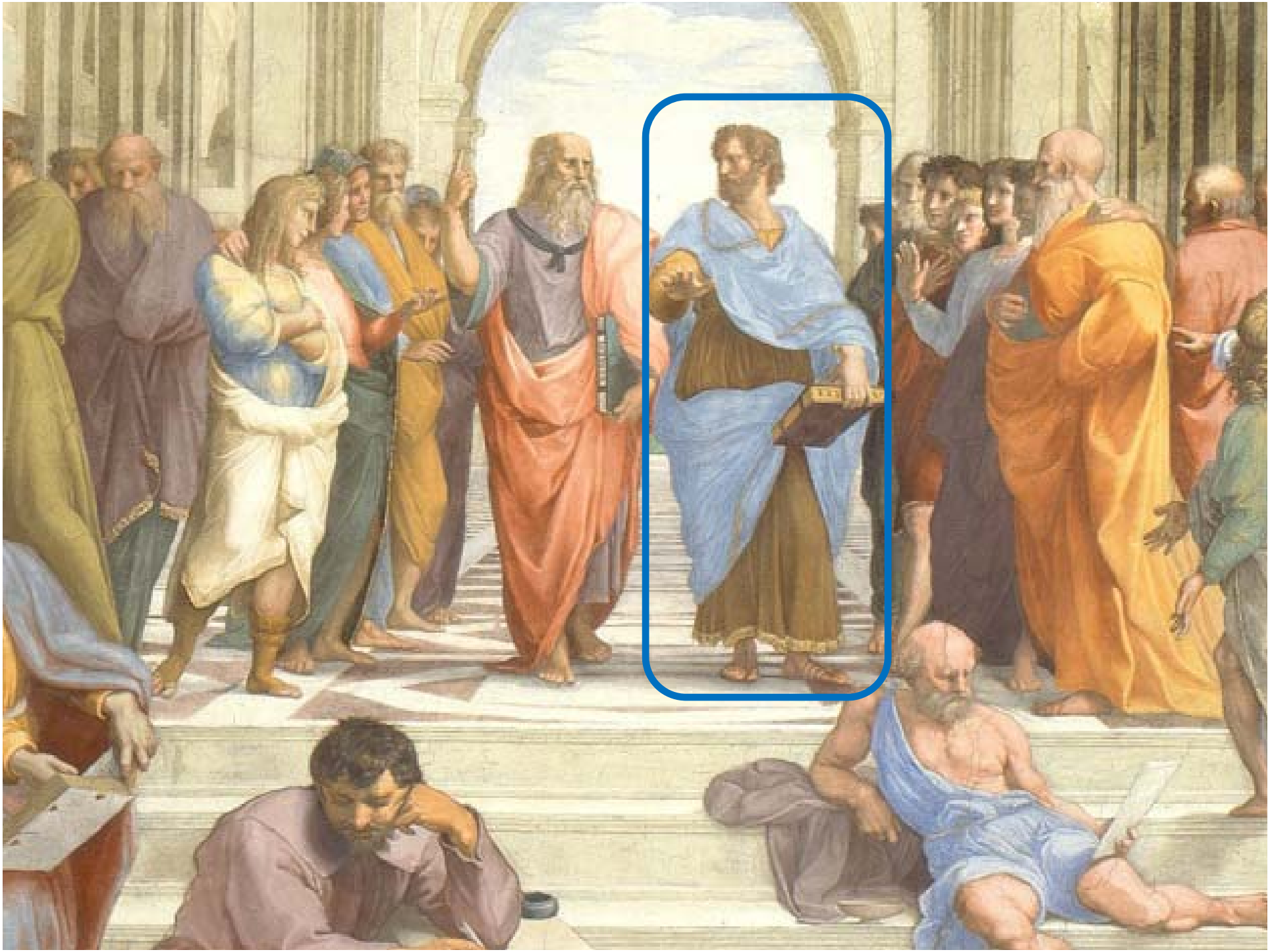


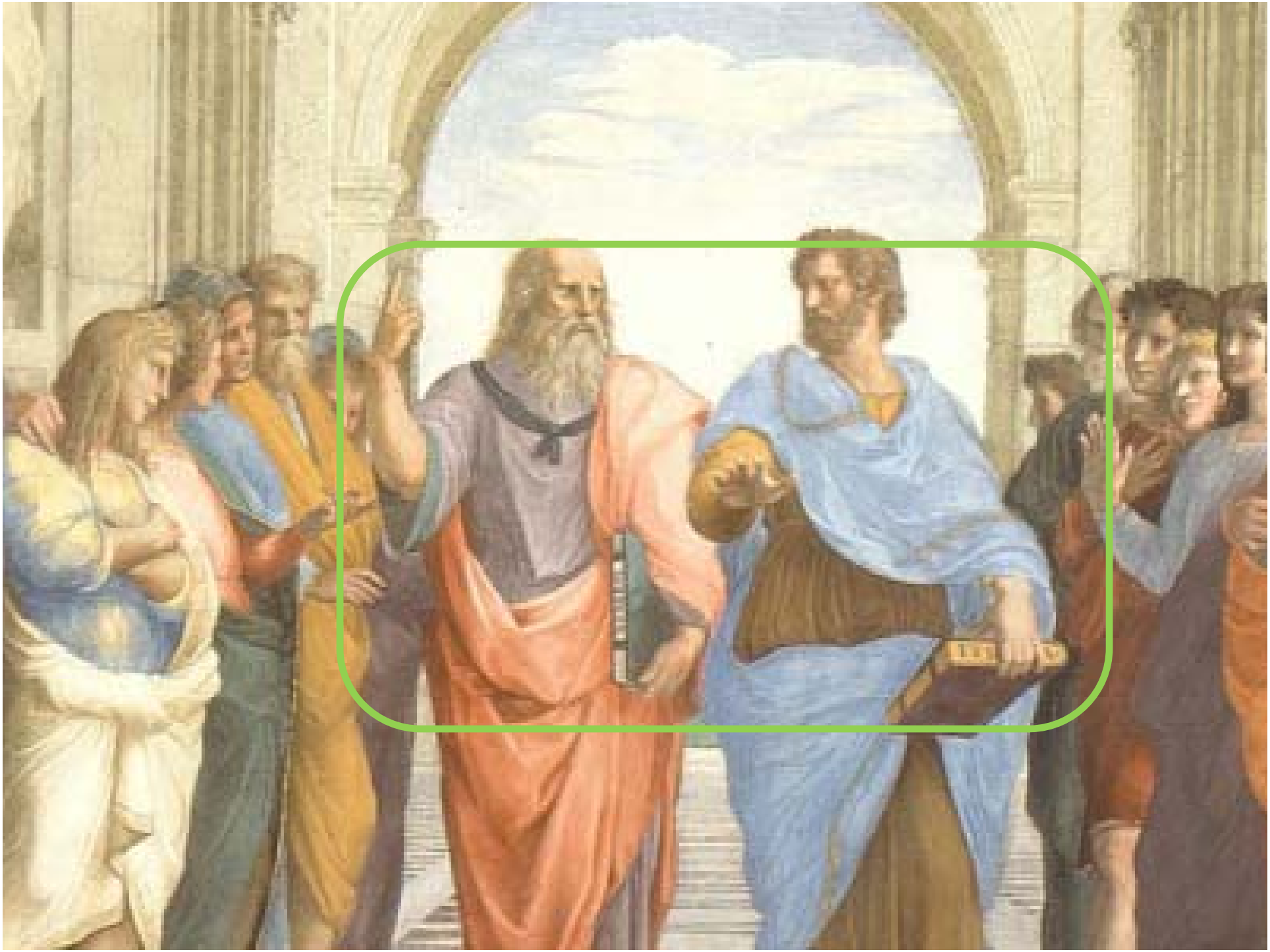






~~Feathers: "people are writing legacy code right now"~~





# Hiera for Pouncers and Stalkers

Share

April 26, 2013 by Mike Hall in Product, Hiera

At my last job, and before I knew much about Puppet, I had a configuration challenge I solved the way a lot of us have: I wrote a script. I got tired of passing lots of configuration parameters to the script, so I created a YAML file. Over time, that YAML file became a source of truth upon which all my scripts depended. If you're using Puppet, you already understand why the scripts had to go. If you've heard about Hiera, you might already know that I got to keep the YAML file.

With Hiera, you create a hierarchy of YAML or JSON files keyed to node characteristics related by Facter. That hierarchy becomes a source of truth for your parameterized classes. You get to drop conditional logic, your site manifests shrink, and you have a single place you can go to feed your Puppet modules with data. Lots of people think that sounds exciting and useful, but the formal documentation on how to use Hiera was a little slow to appear, so users were left piecing together fragments from Github repos, PuppetConf talks, and the occasional blog entry. In the past month or so, we have pushed a number of

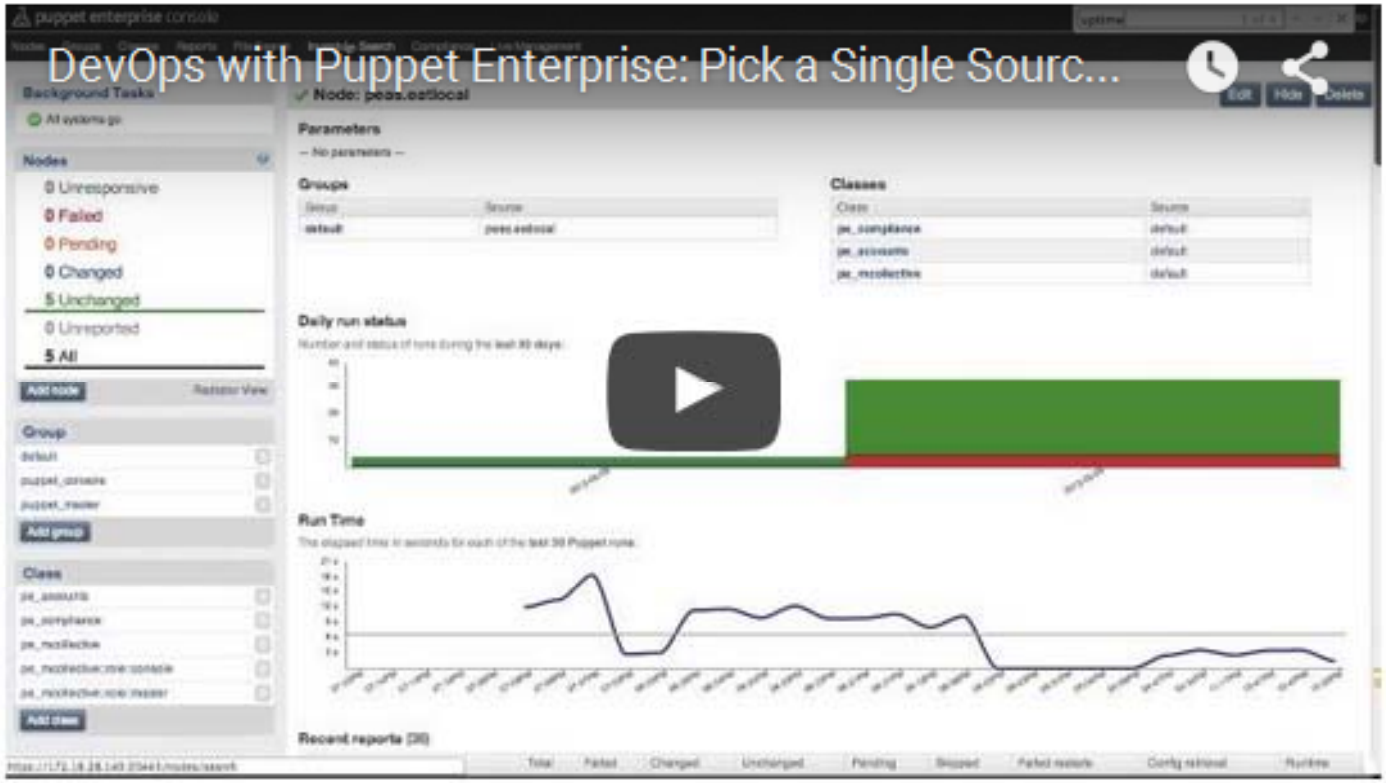
puppet labs

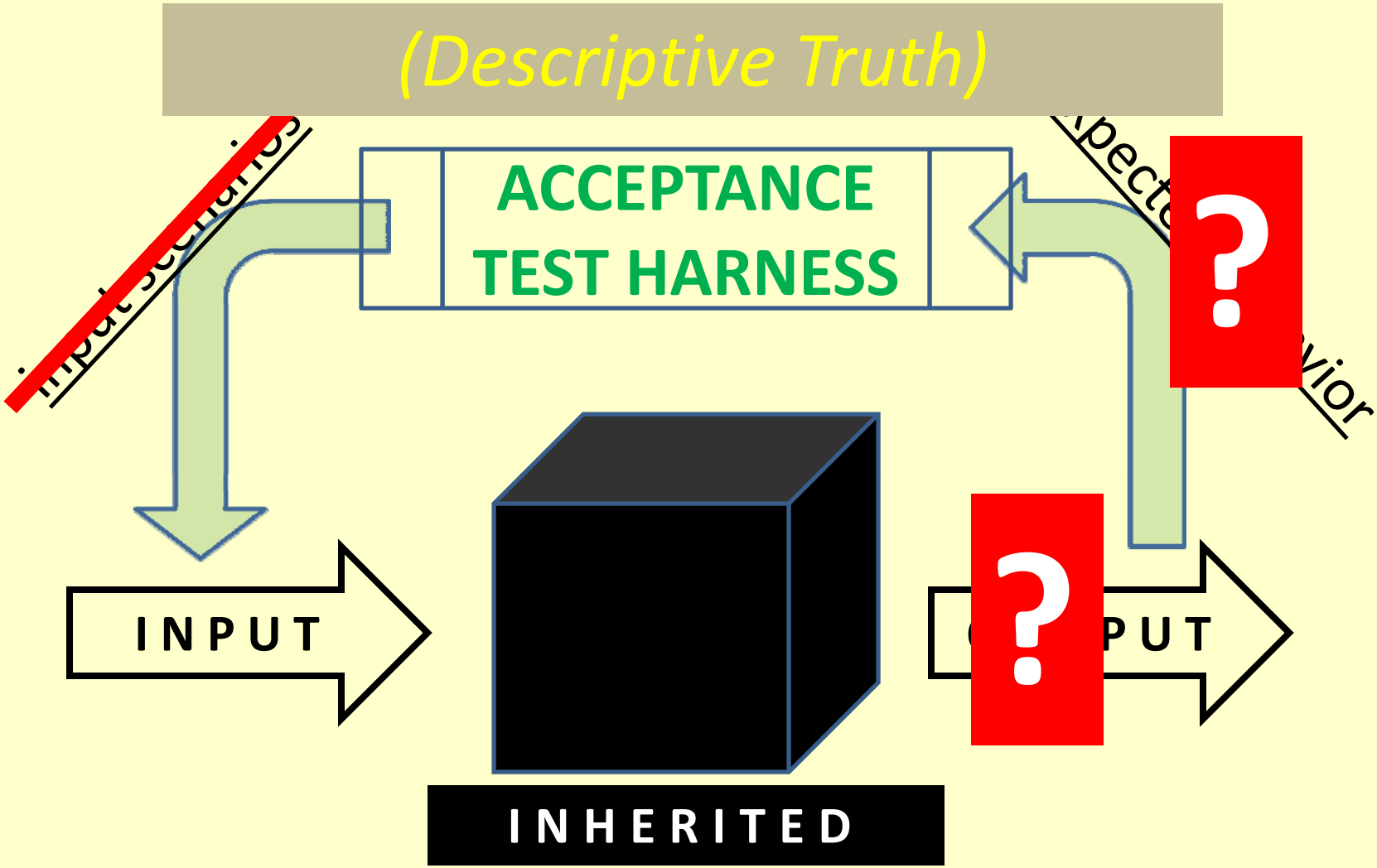
Menu

search

# DevOps with Puppet Enterprise: Pick a Single Source of Truth

Get rid of the spreadsheet with Puppet Enterprise. (2:21)

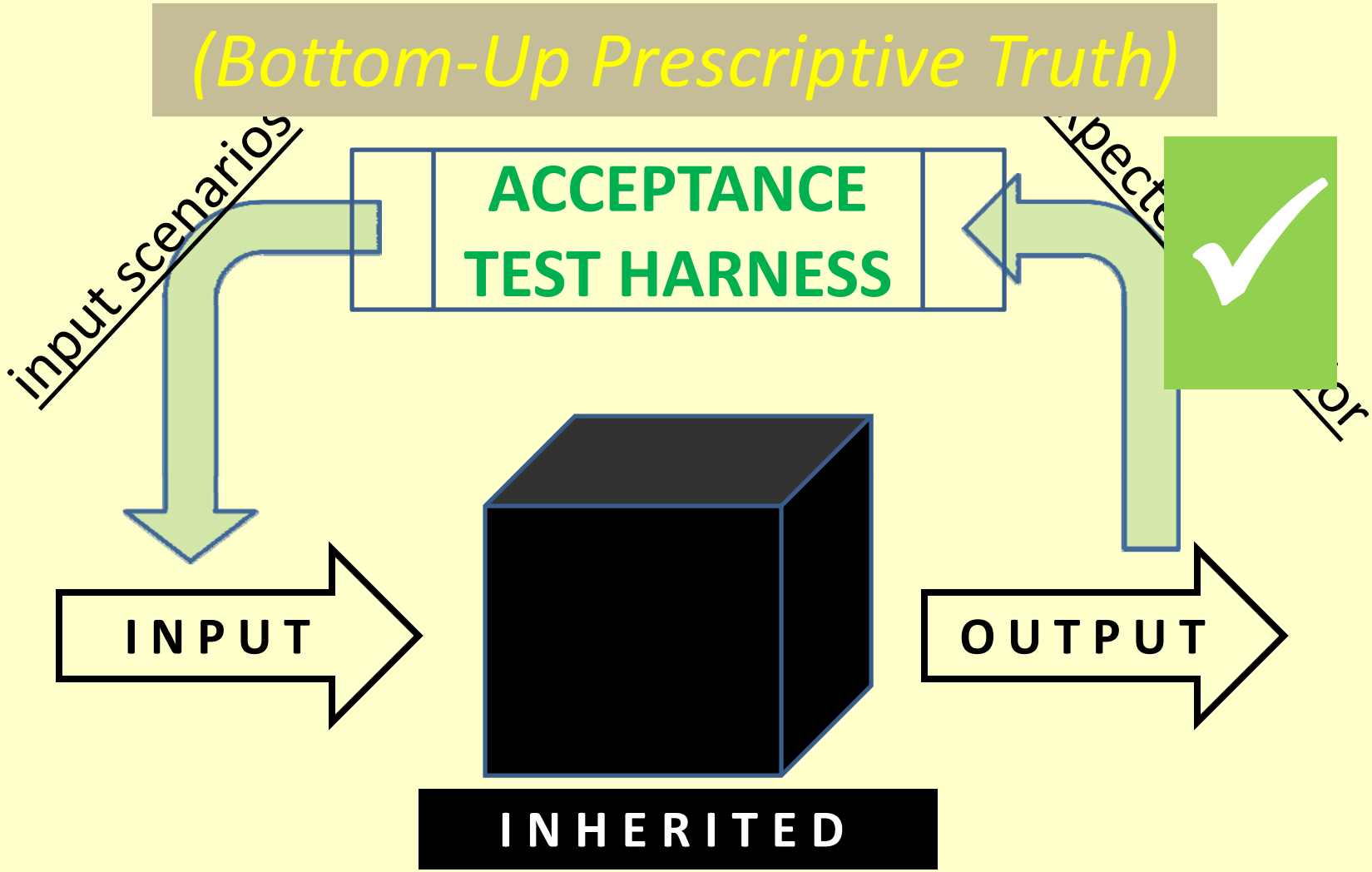


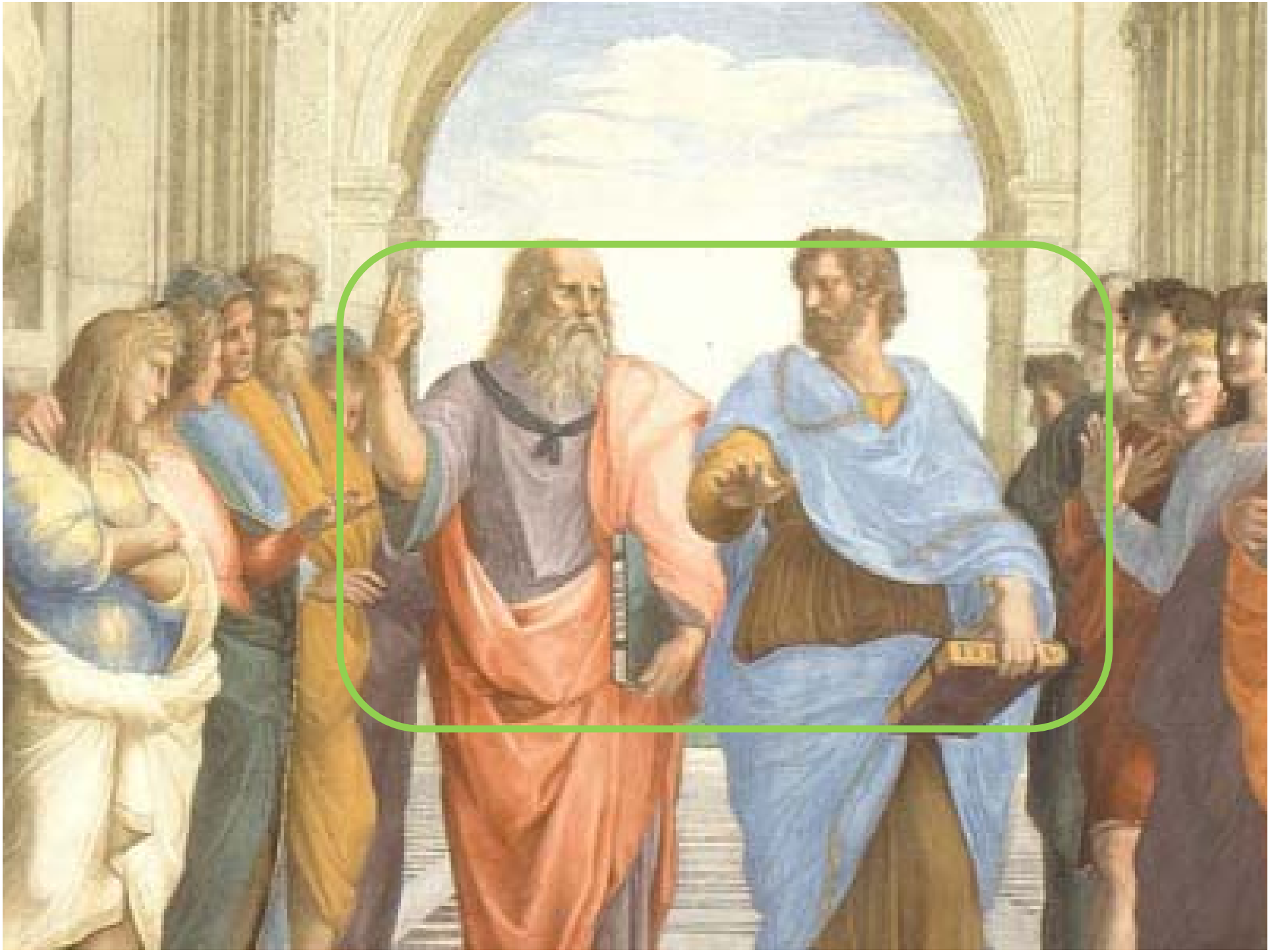

(Descriptive Truth)
ACCEPTANCE
TEST HARNESS
?
INPUT
?
INHERITED
Facts on the Ground say
What IS TRUE

Is this enough?

"… correct behavior is defined by what the set of classes did yesterday, not by any external standard of correctness... It is quite literally the case that no one 'knows' whether the code is correct or not. "

Feathers


(Bottom-Up Prescriptive Truth)
input scenarios
ACCEPTANCE
TEST HARNESS
INPUT
OUTPUT
INHERITED
Facts on the Ground say
What SHOULD be TRUE



*(Bottom-Up Prescriptive Truth)*

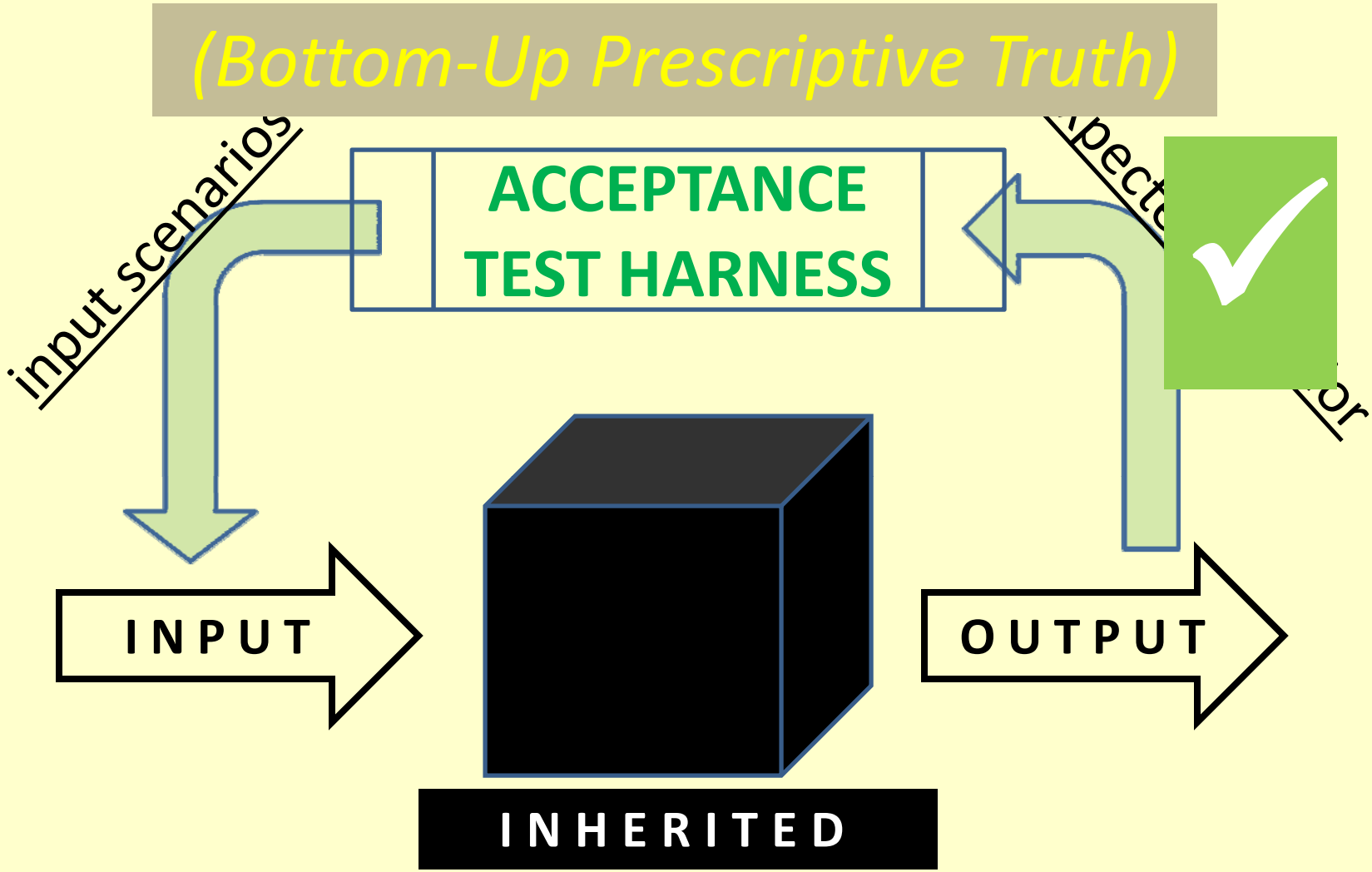
* What developers treat as normative / prescriptively true, when troubleshooting

E.g.,

* System “facts” guiding the spec derived from checks of attributes of existing DB / installed SW

* Emergent (desired) behaviors

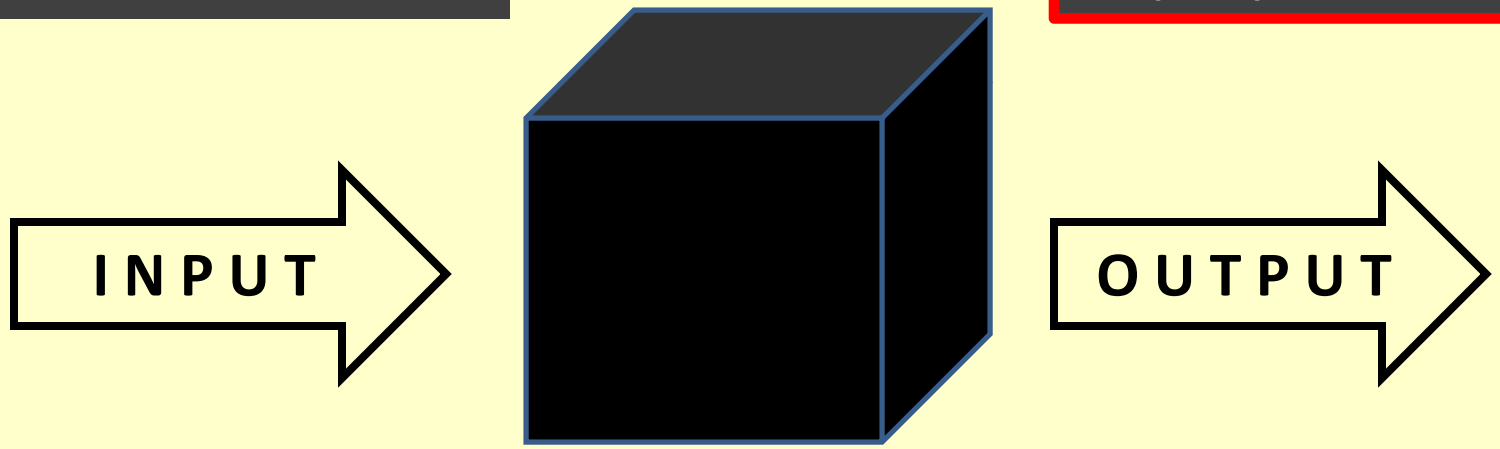
Facts on the Ground say
What SHOULD be TRUE


(Bottom-Up Prescriptive Truth)
input scenarios
ACCEPTANCE
TEST HARNESS
INPUT
OUTPUT
INHERITED
Facts on the Ground say
What SHOULD be TRUE

# *Maybe it's good enough as it is?*

1) Hard to add new features

3) Hard/slow to deploy

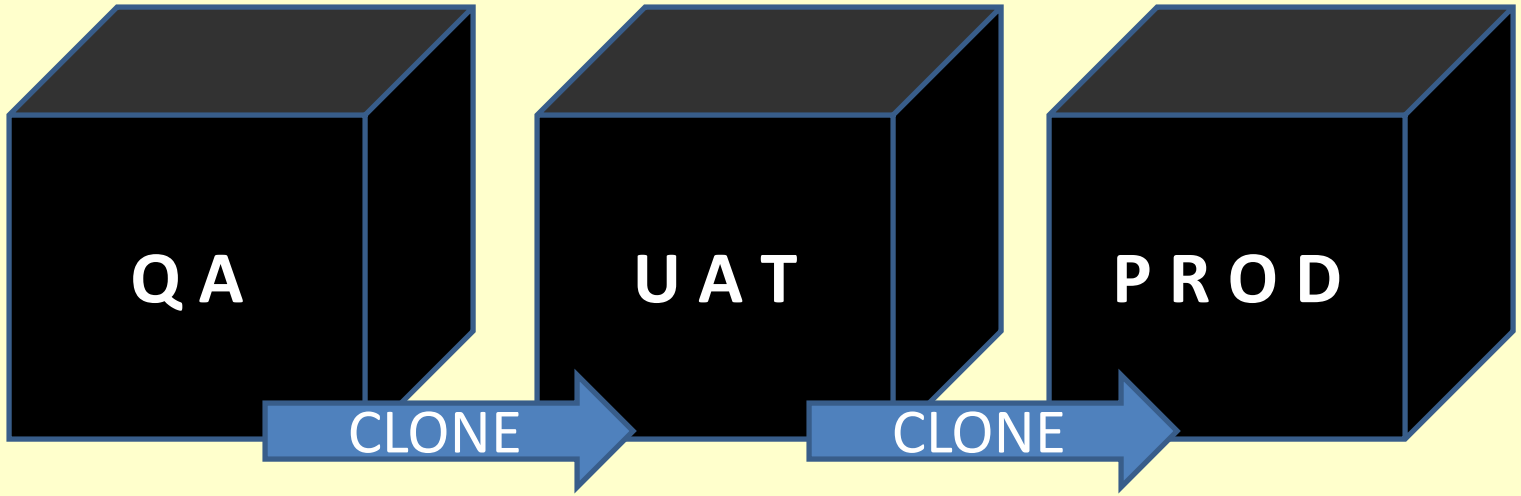



2) Hard to debug

4) One day it's going to break

# *Maybe it's good enough as it is?*

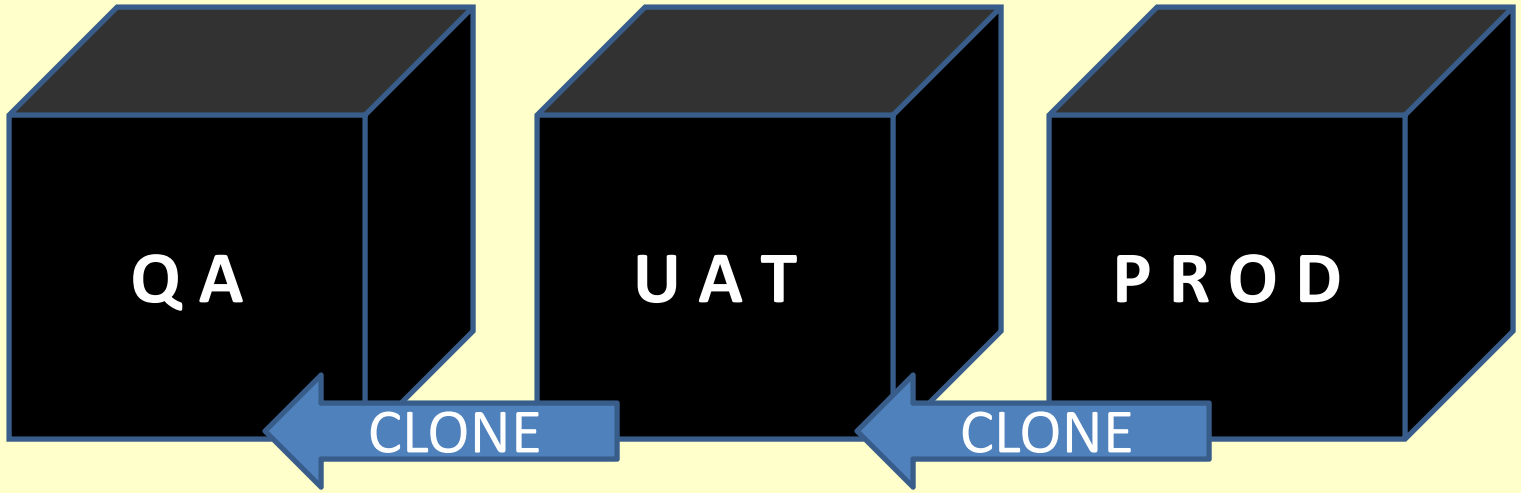
~~3) Ha[illegible]d/[illegible]ow to deplo[illegible]~~




→ VM /Primitive Container Clone Pipeline →

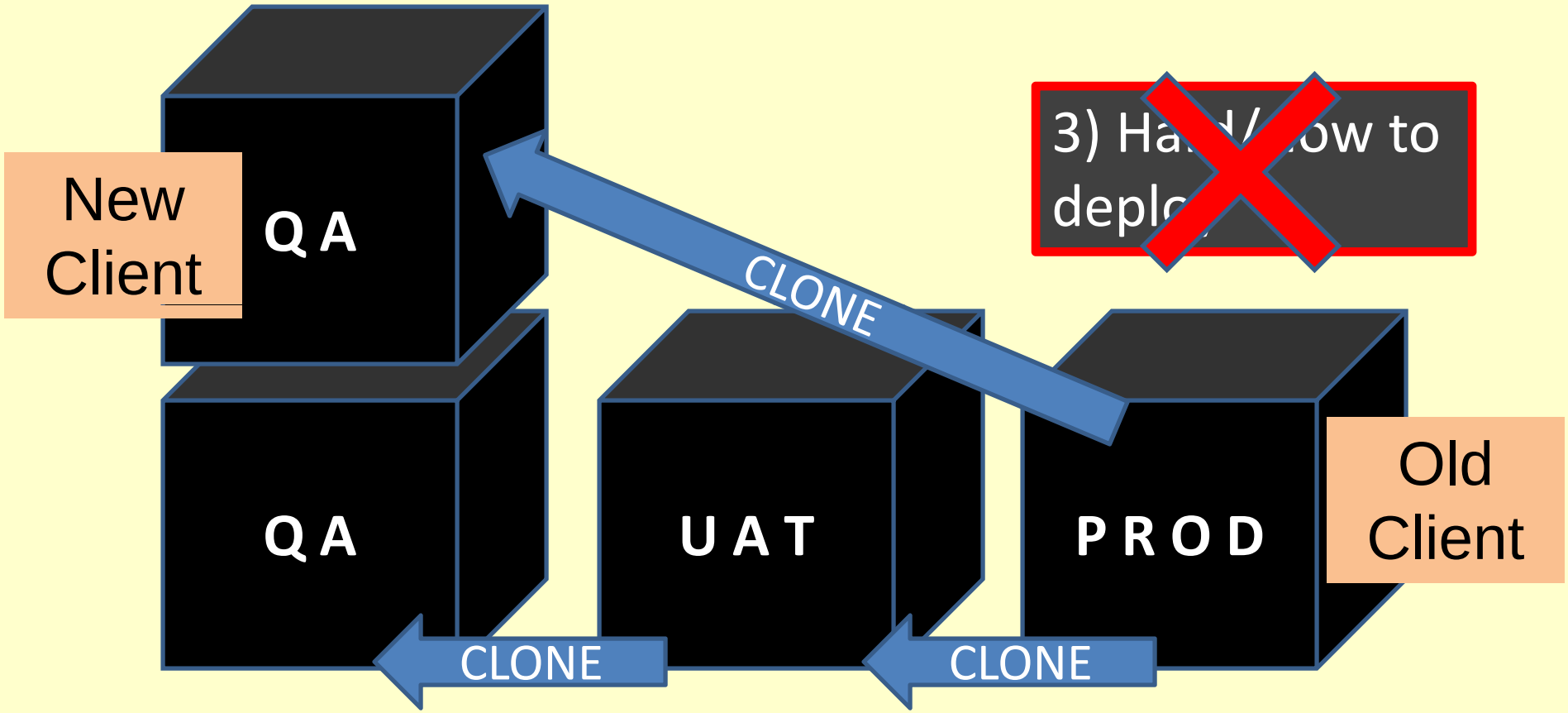
# *Maybe it's good enough as it is?*

3) Ha[illegible]d/[illegible]ow to deplo[illegible]




←VM /Primitive Container Clone Pipeline ←

# *Maybe it's good enough as it is?*

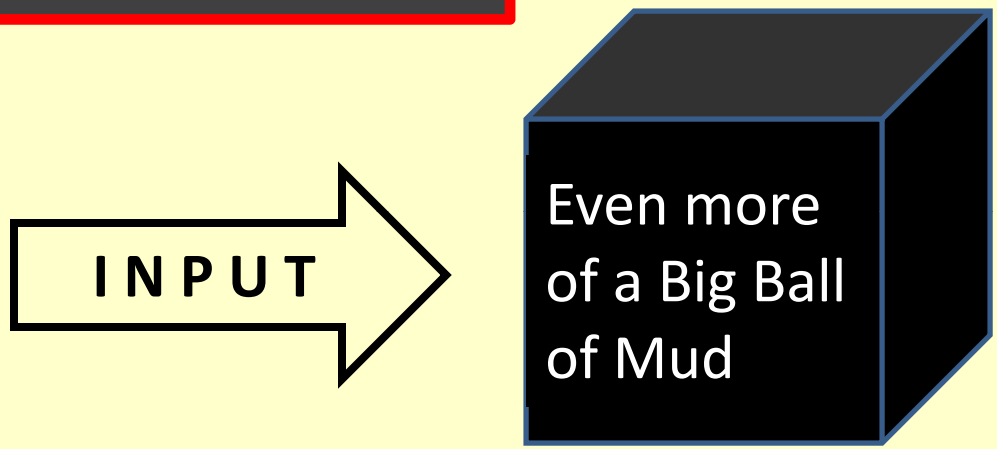



←VM /Primitive Container Clone Pipeline ←

# *Maybe it's good enough as it is?*

1) Hard to add new features

3) Ha[illegible]d/[illegible]ow to deplo[illegible]




2) Hard to debug

4) On[illegible]day it's going to [illegible]reak

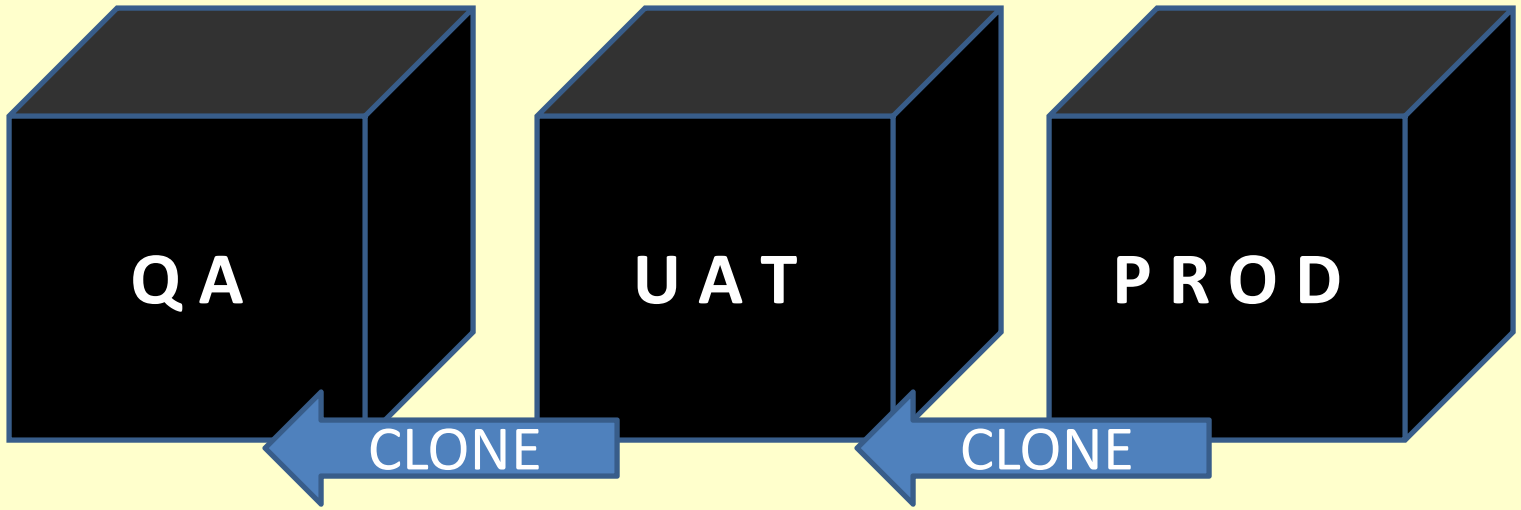
* External entropy; e.g. network and library changes

* Internal entropy a result of feature enhancements

* Current quality is frozen, but lousy nonetheless

# *Black box replication not good enough!*

1. First step, stabilize existing system behavior and introduce whole black box replication
2. But goal is to be able to build new box or deploy new features without having to clone




←VM /Primitive Container Clone Pipeline ←

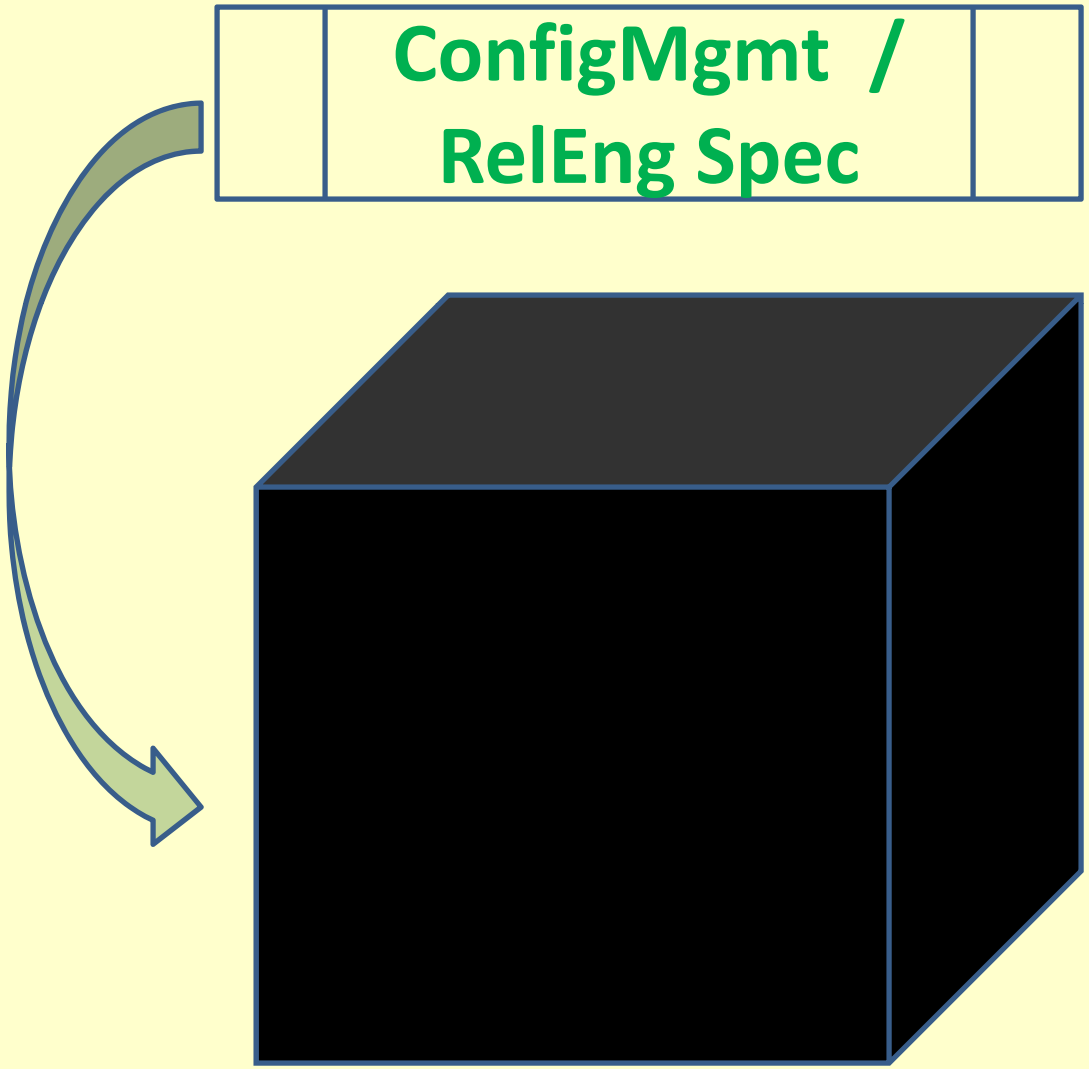
# *Black box replication not good enough!*

1. First step, stabilize existing system behavior and introduce whole black box replication
2. But goal is to be able to build new box or deploy new features without having to clone

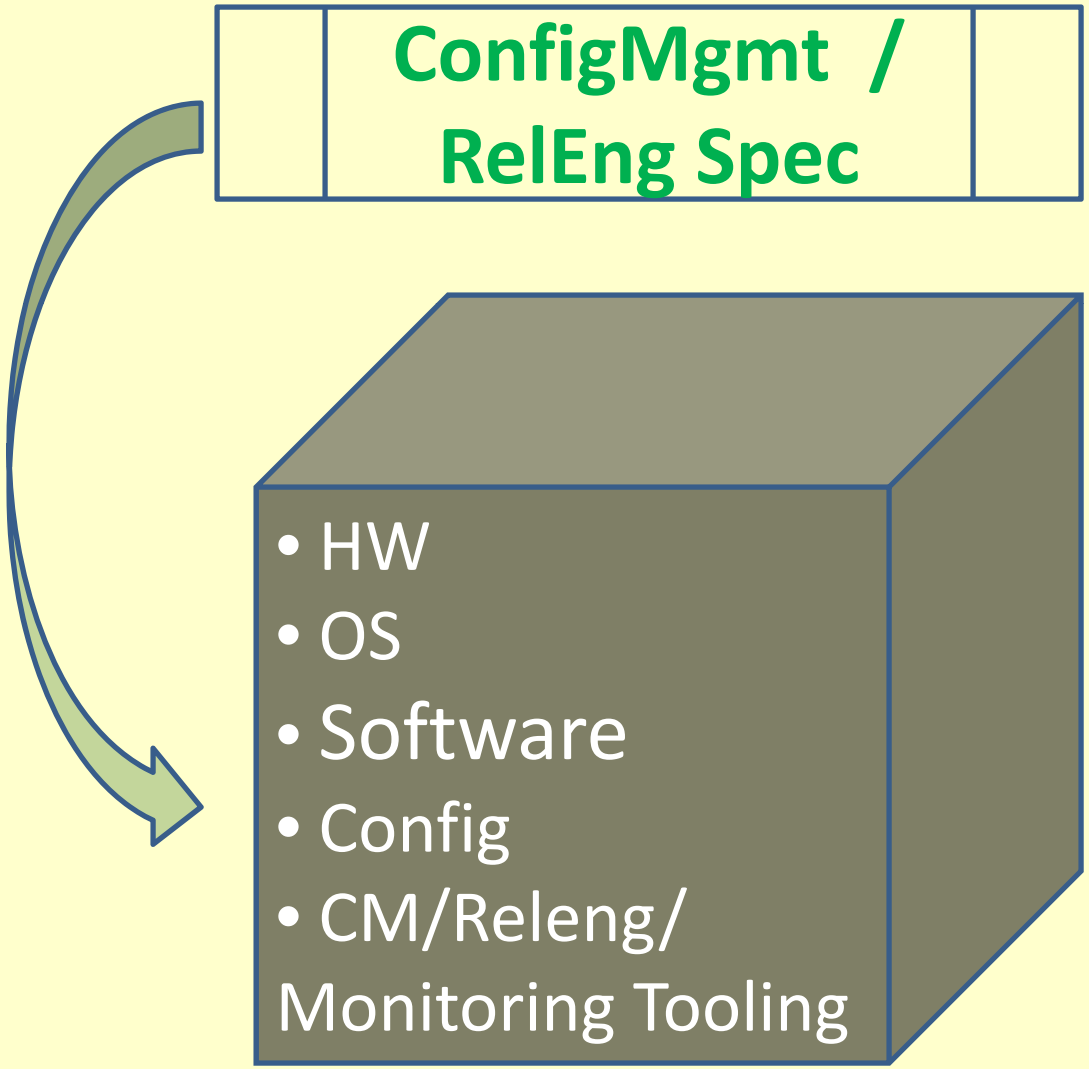
-- Can you deploy a new system without cloning?

-- Legacy System == A system that is CD-resistant

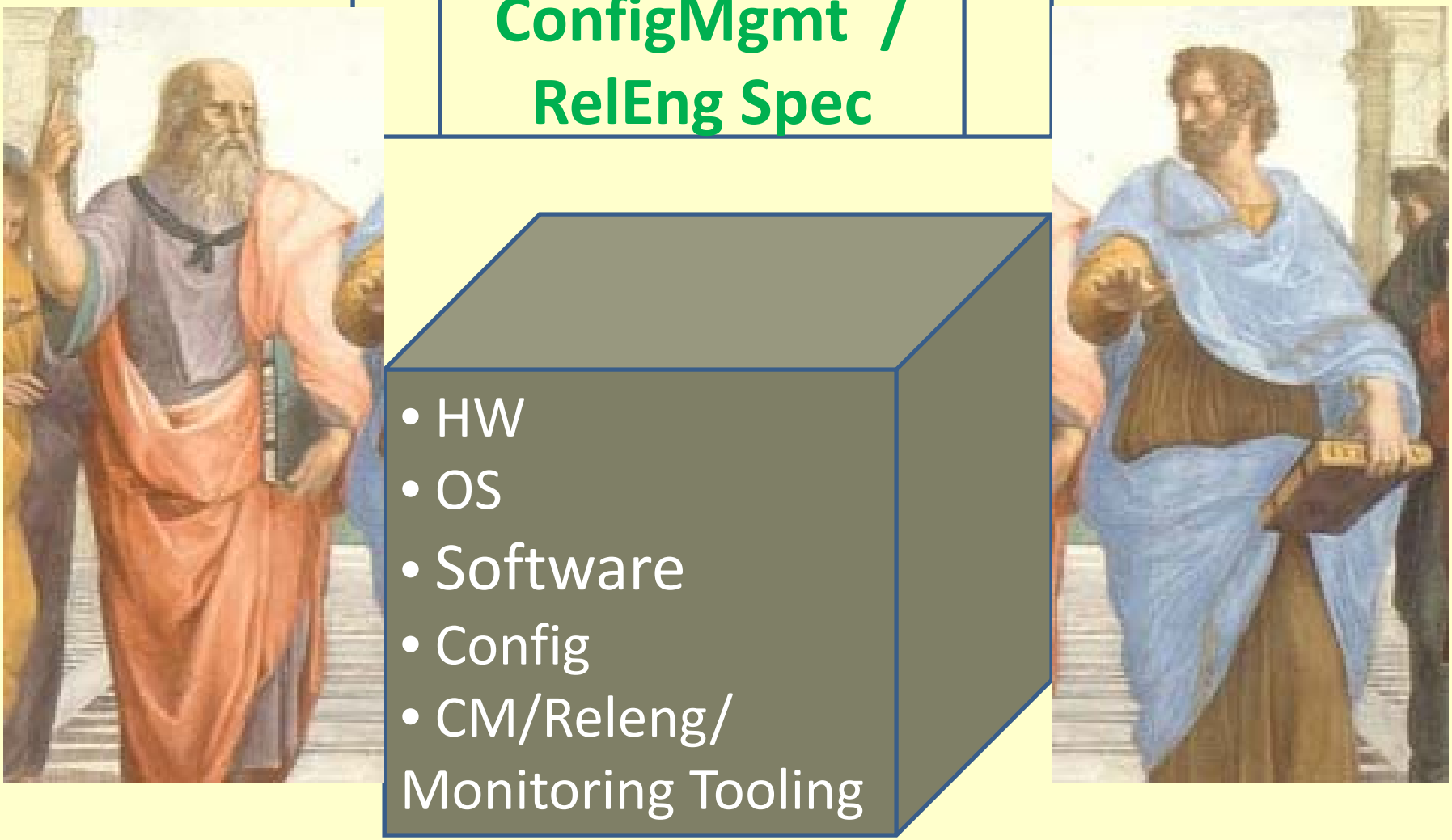
# *Making the legacy system CD-permeable*



# *Making the legacy system CD-permeable*



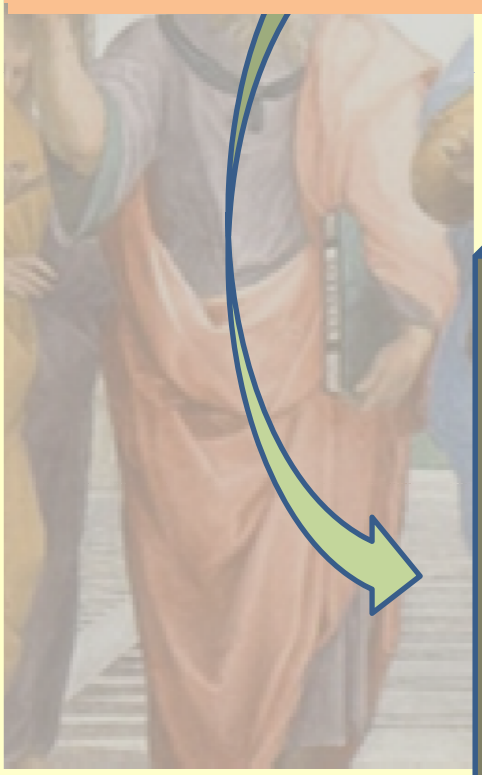
# *Making the legacy system CD-permeable*



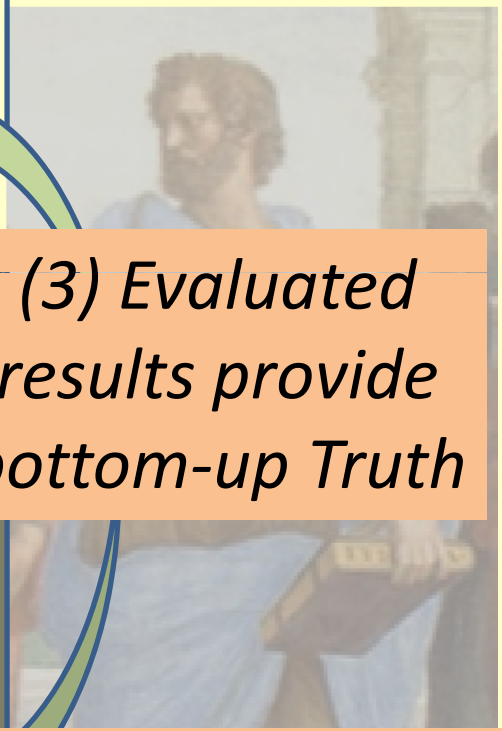
# *Making the legacy system CD-permeable*

*(1) <u>Refactor</u> to enhance top-down Truth spec*

**ConfigMgmt / RelEng Spec**



- HW
- OS
- Software
- Config
- CM/Releng/ Monitoring Tooling



*(3) Evaluated results provide bottom-up Truth*

*(2) By tooling up for better descriptive Truth*

# *Making the legacy system CD-permeable*

(1/4) Refactor to enhance top-down Truth spec

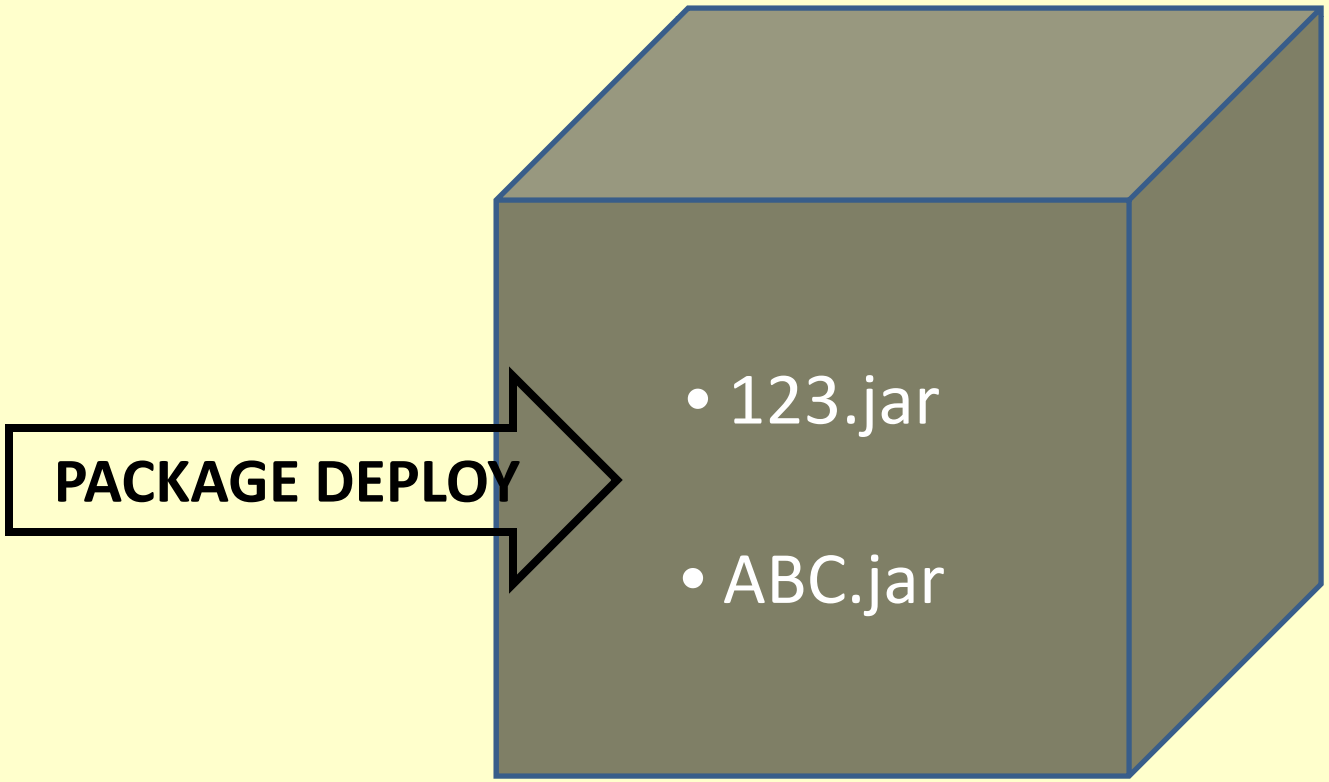
(2) By tooling up for better descriptive Truth

(3) Evaluated results provide bottom-up Truth

# *Making the legacy system CD-permeable*

(2) By tooling up for better descriptive Truth

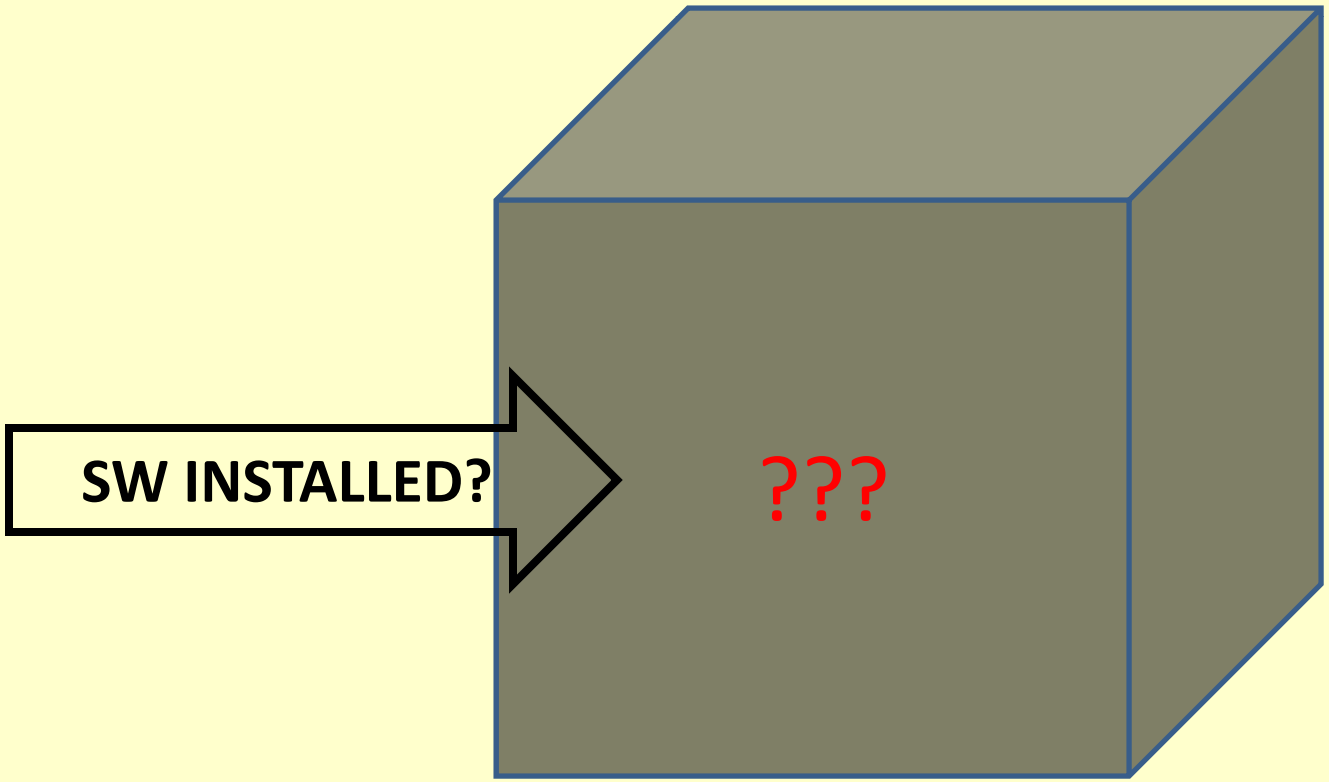
-- enhance tools to reveal system-relevant facts



# *Making the legacy system CD-permeable*

(2) By tooling up for better descriptive Truth

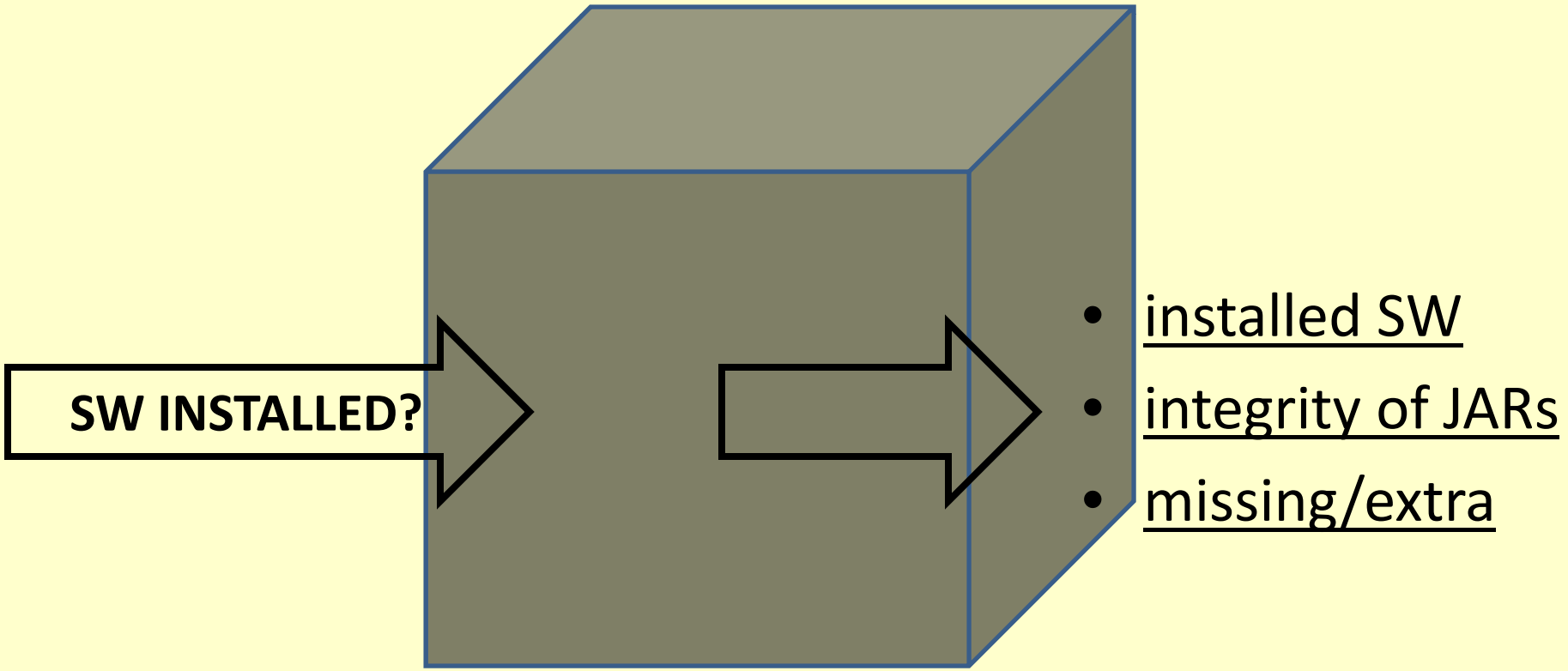
-- enhance tools to reveal system-relevant facts



# *Making the legacy system CD-permeable*

## (2) By tooling up for better descriptive Truth

-- enhance tools to reveal system-relevant facts



# *Making the legacy system CD-permeable*

(2) By tooling up for better descriptive Truth

-- enhance tools to reveal system-relevant facts

-- company-wide metrics to track degree of CD implementation

company wide % of auto-installed SW

-- ideally tied to bug rates, # of rollouts, time from commit to production, client onboarding frequency, etc.

# *Making the legacy system CD-permeable*

## (3) Evaluated results provide bottom-up Truth

- talk to:
  - developers
  - product managers
  - account managers / customer reps
    - the user experience
  - tech support / NOC
- watch practices / cultural “muscle memory”

# *Making the legacy system CD-permeable*

## (1/4) Refactor to enhance top-down Truth spec

- Wrap more specific acceptance tests around pieces of system to be made more CD-permeable
- Problem: entangled dependencies make it hard to reliably isolate one portion of the system
- And if you can’t isolate it, you can’t focus on improving just that piece

# *Making the legacy system CD-permeable*

## (1/4) Refactor to enhance top-down Truth spec

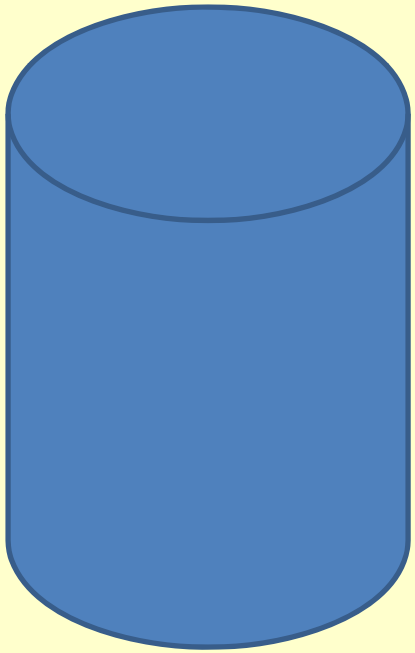
Code-Level Solution to Entangled Dependencies

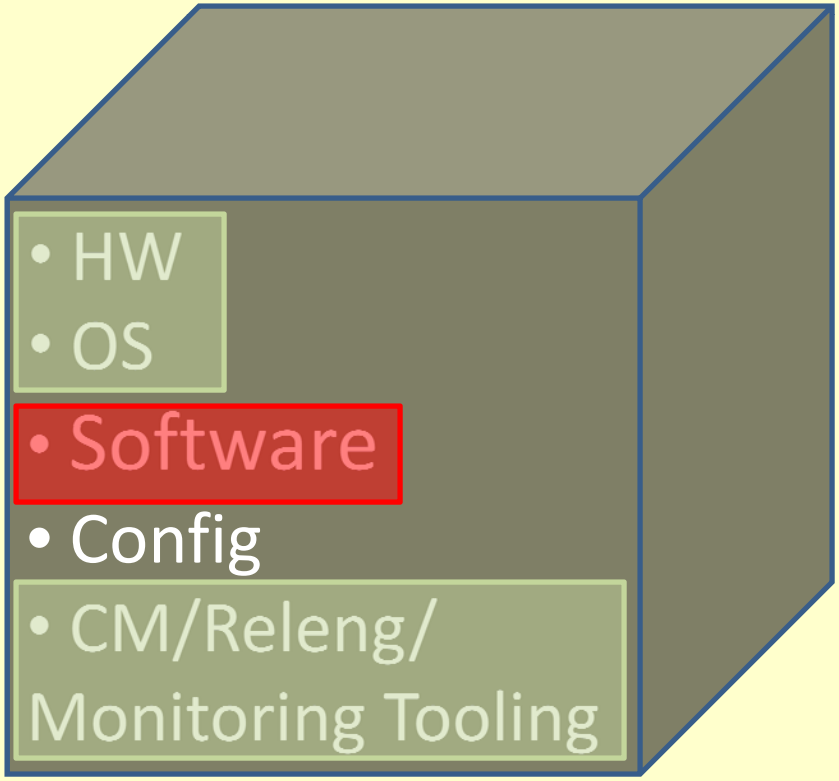
1. Find “inflection point”, important link between two components of system
2. Figure out the “contract” for the data flow
3. Break dependencies between components by providing simulated input
4. Write acceptance tests around the isolated component using the simulated input
5. Refactor that component at will

# *Making the legacy system CD-permeable*

**Ticketing**

**ConfigMgmt / RelEng Spec**

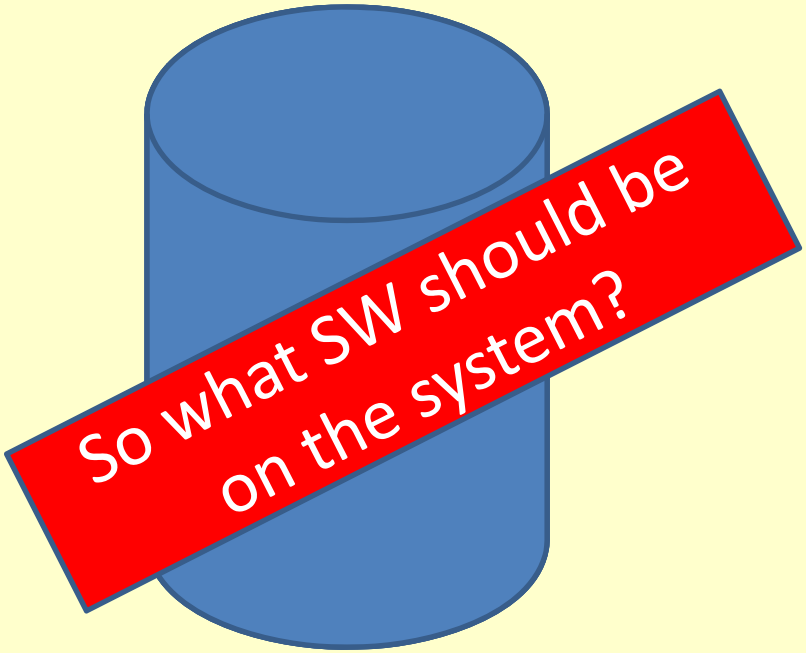


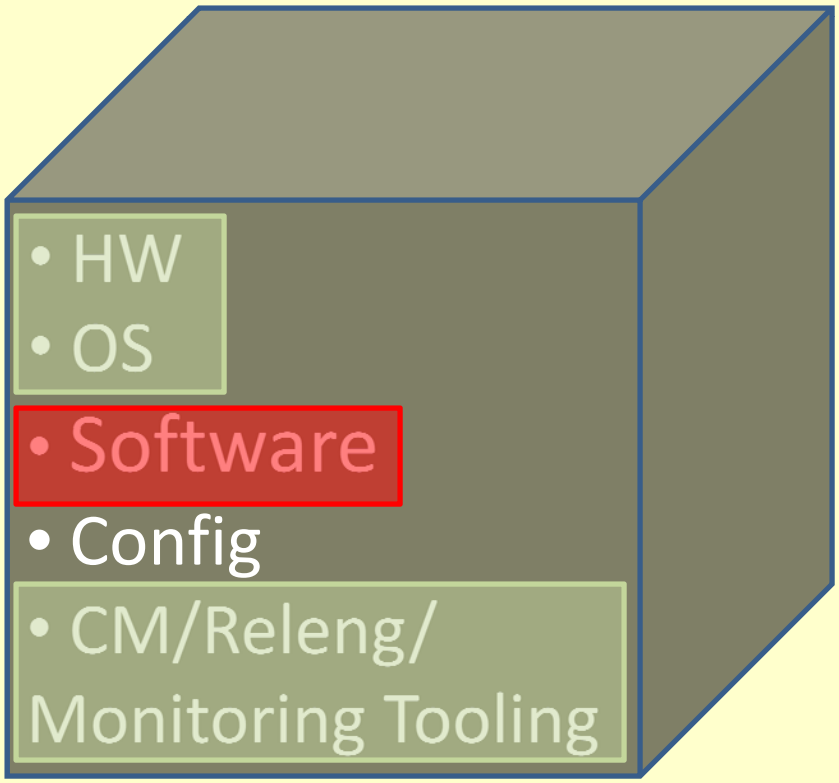


# *Making the legacy system CD-permeable*

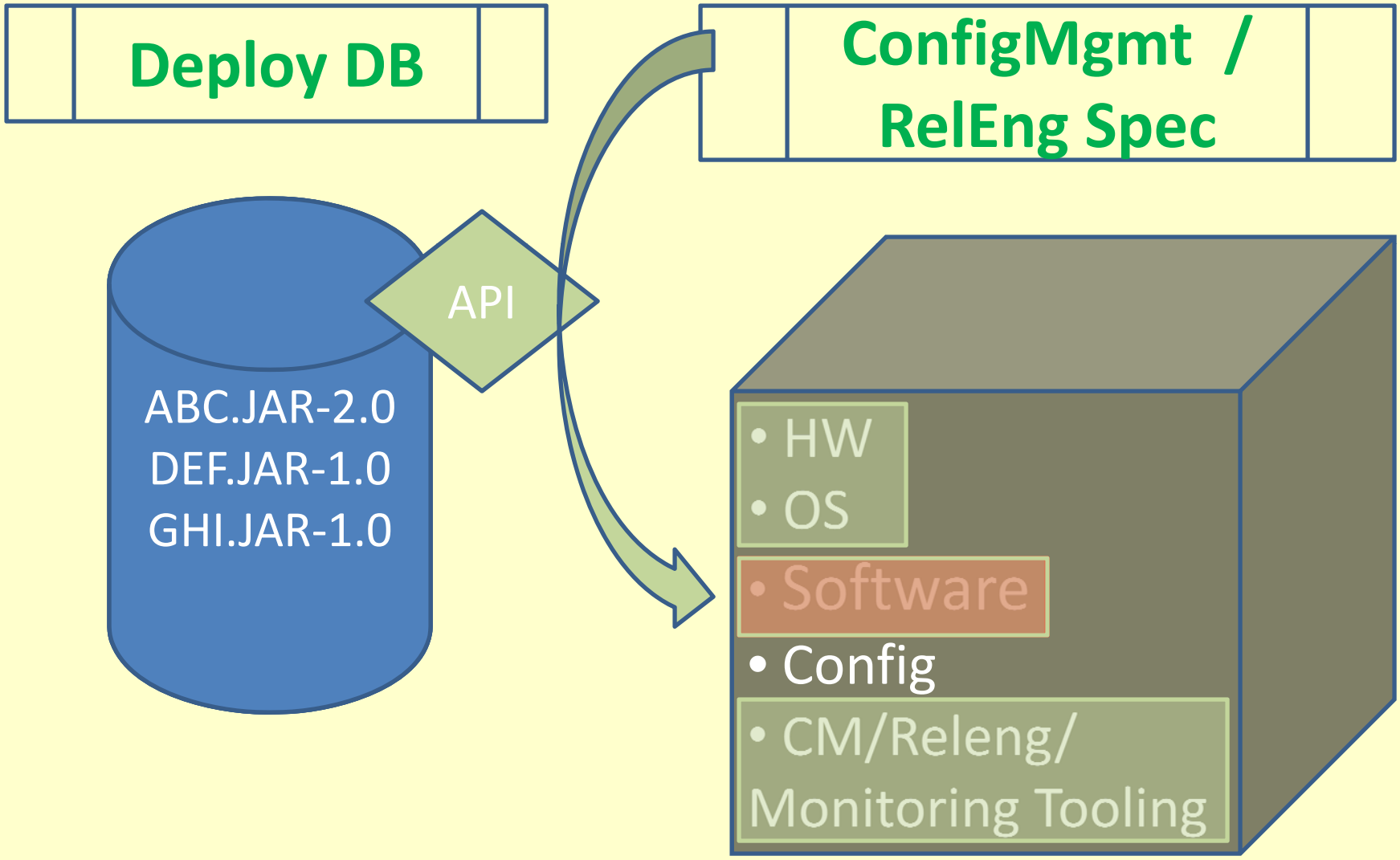
**Ticketing**

**ConfigMgmt / RelEng Spec**

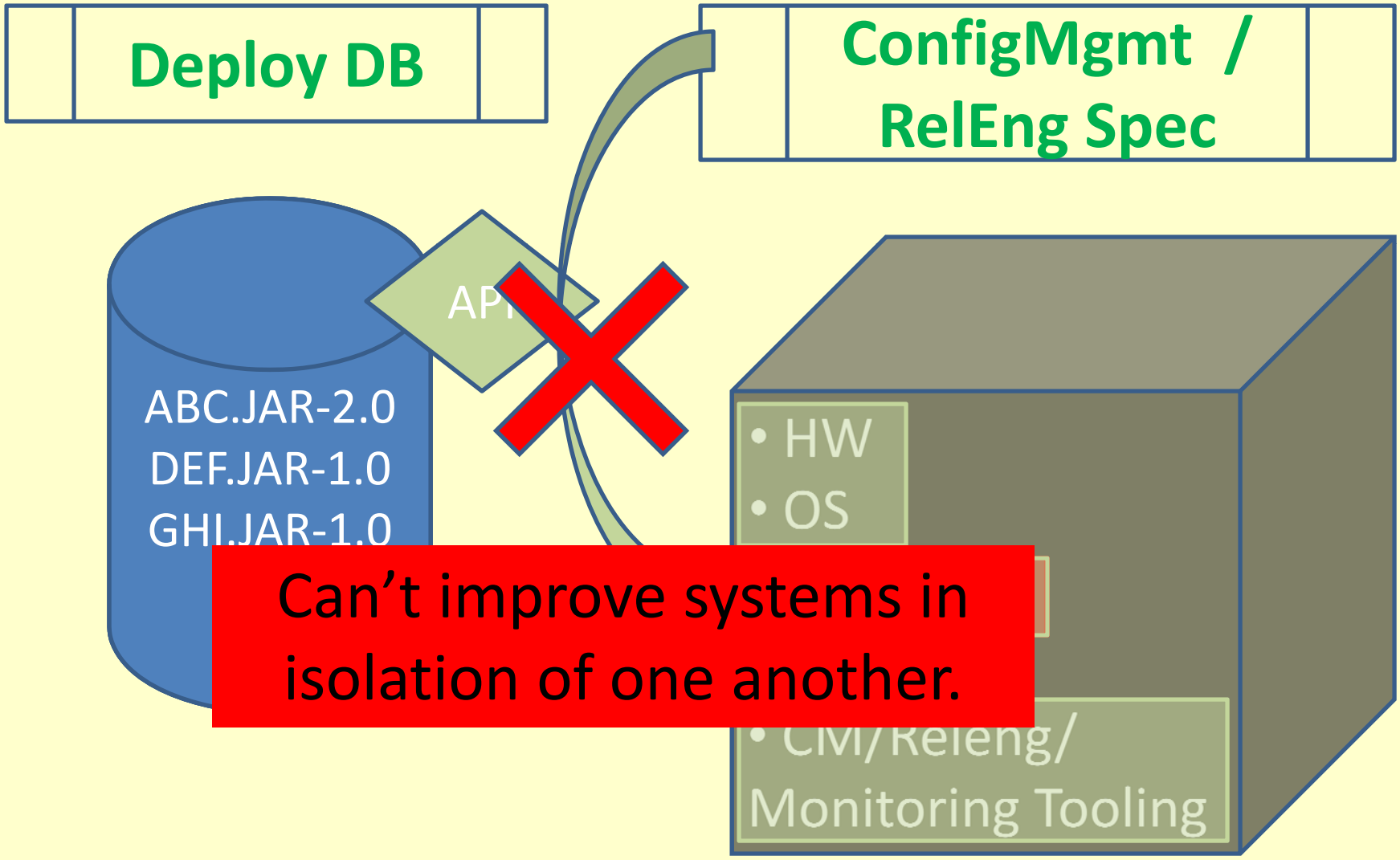


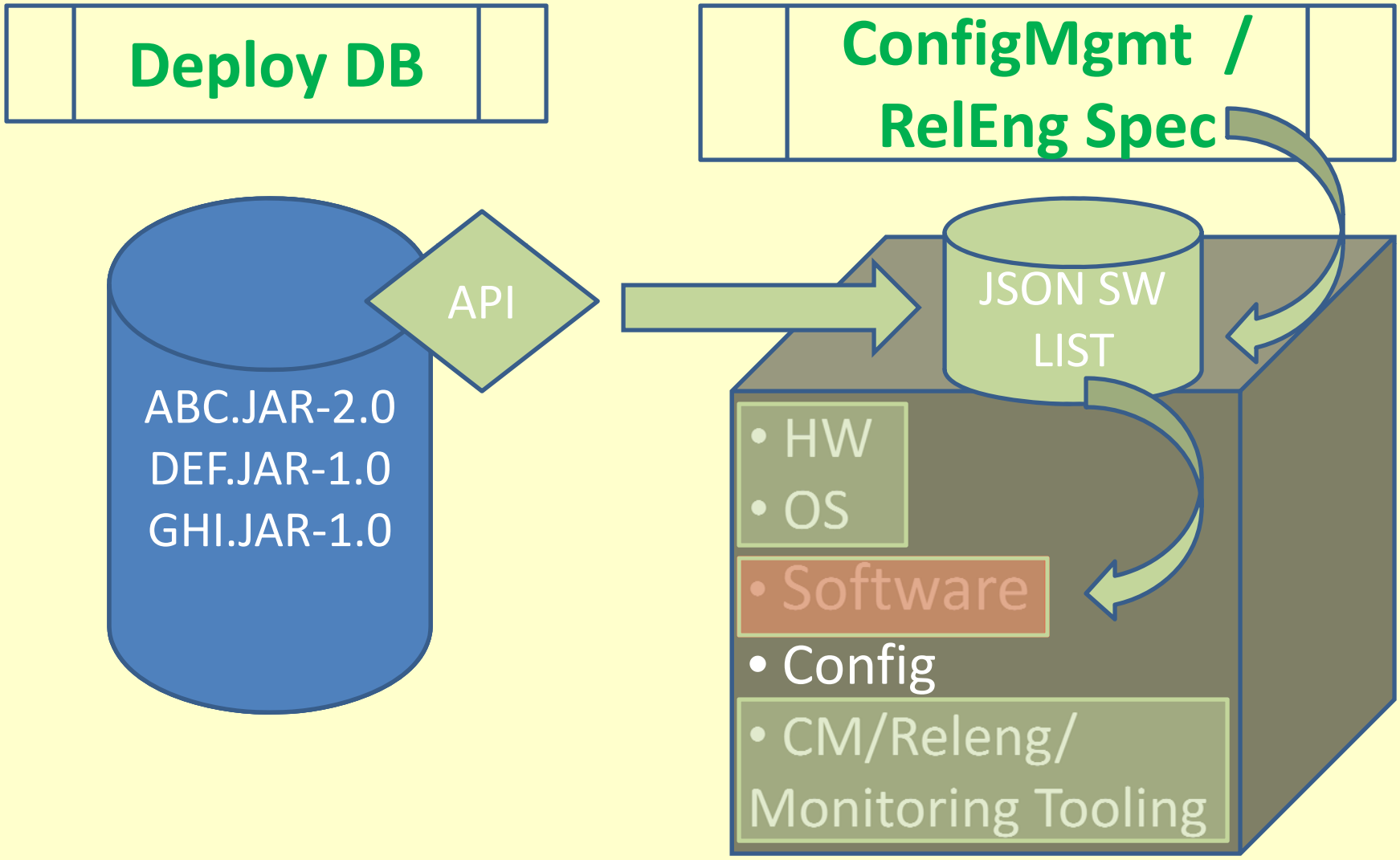
# *Making the legacy system CD-permeable*



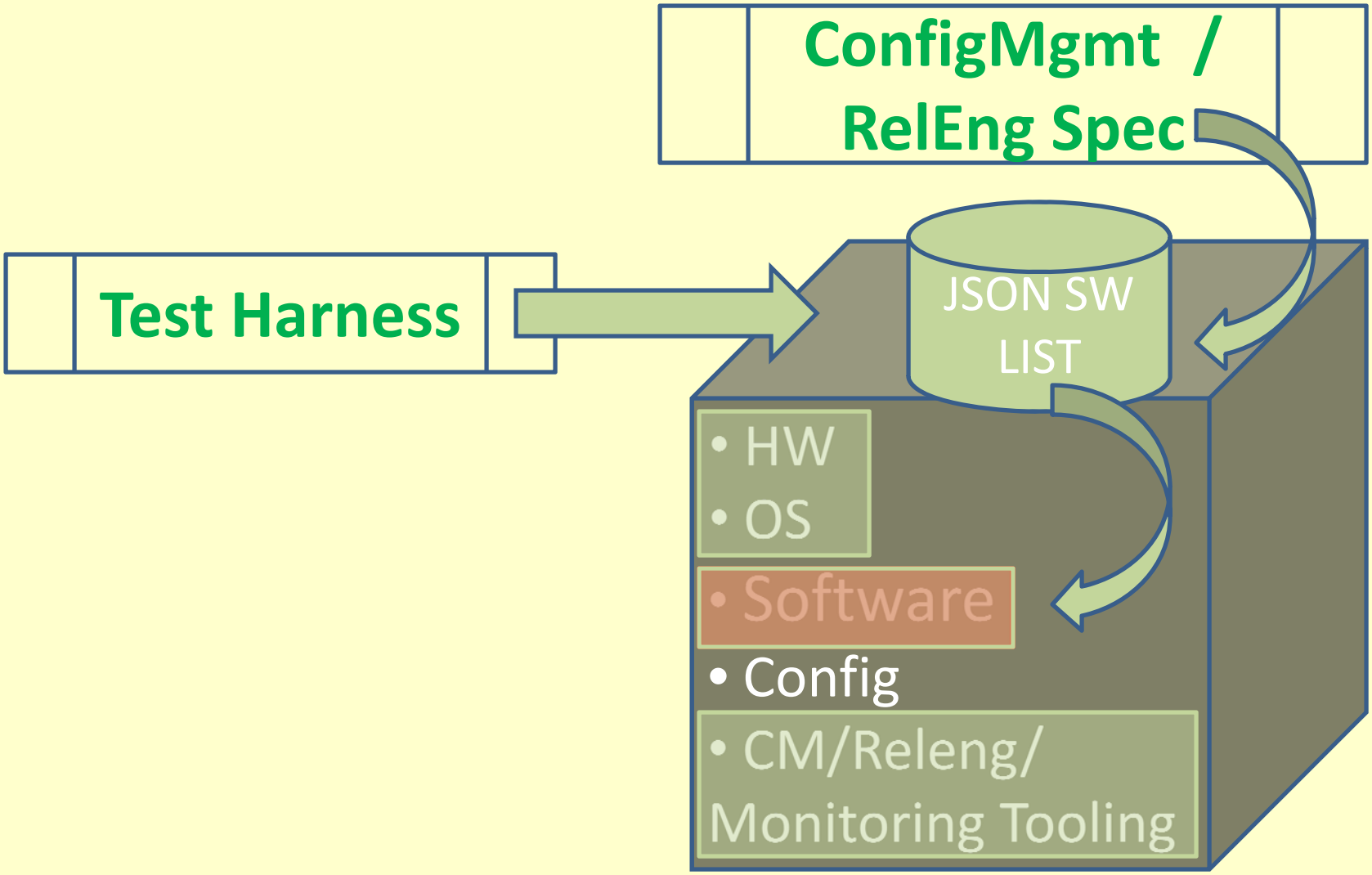
# *Making the legacy system CD-permeable*



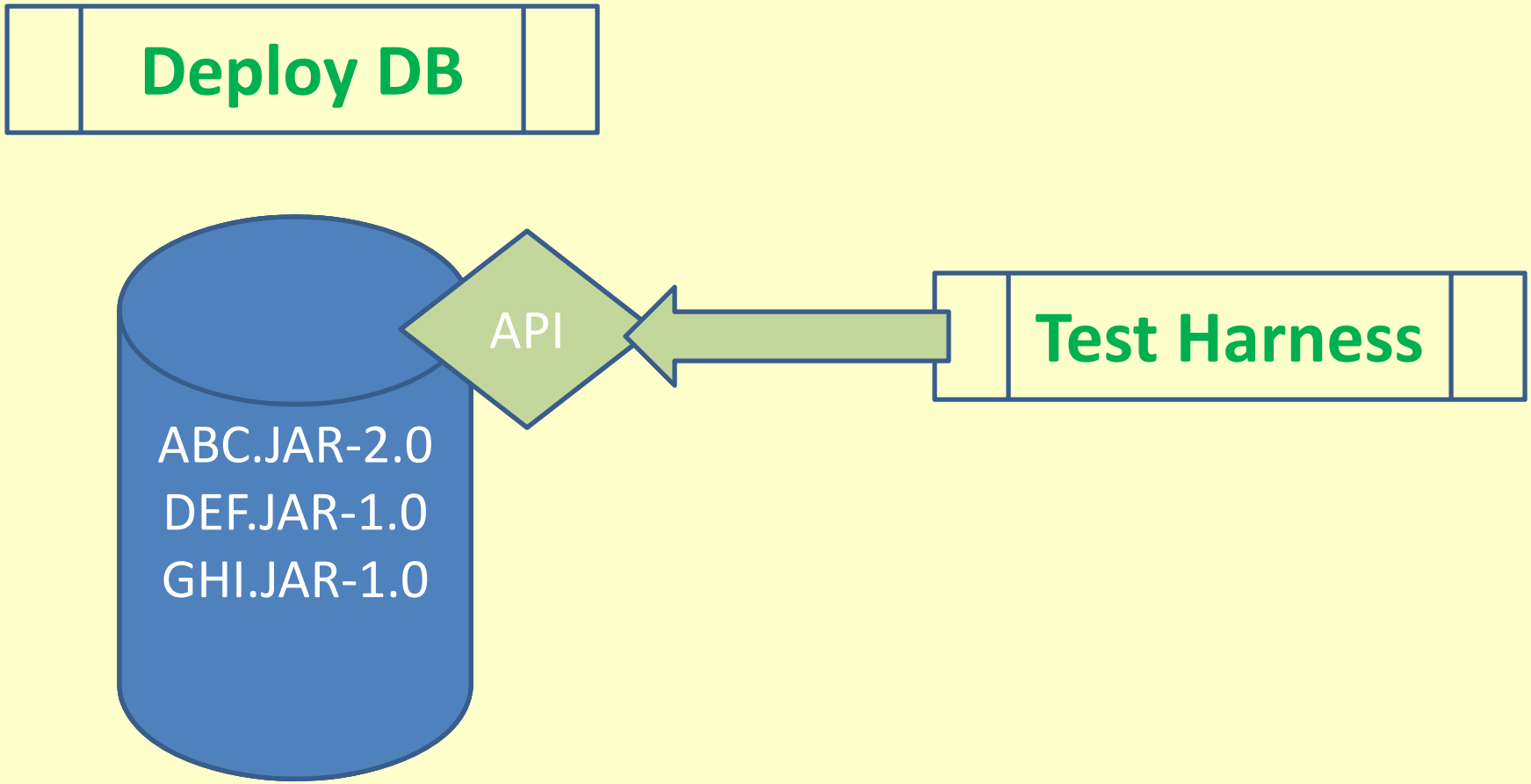
# *Making the legacy system CD-permeable*



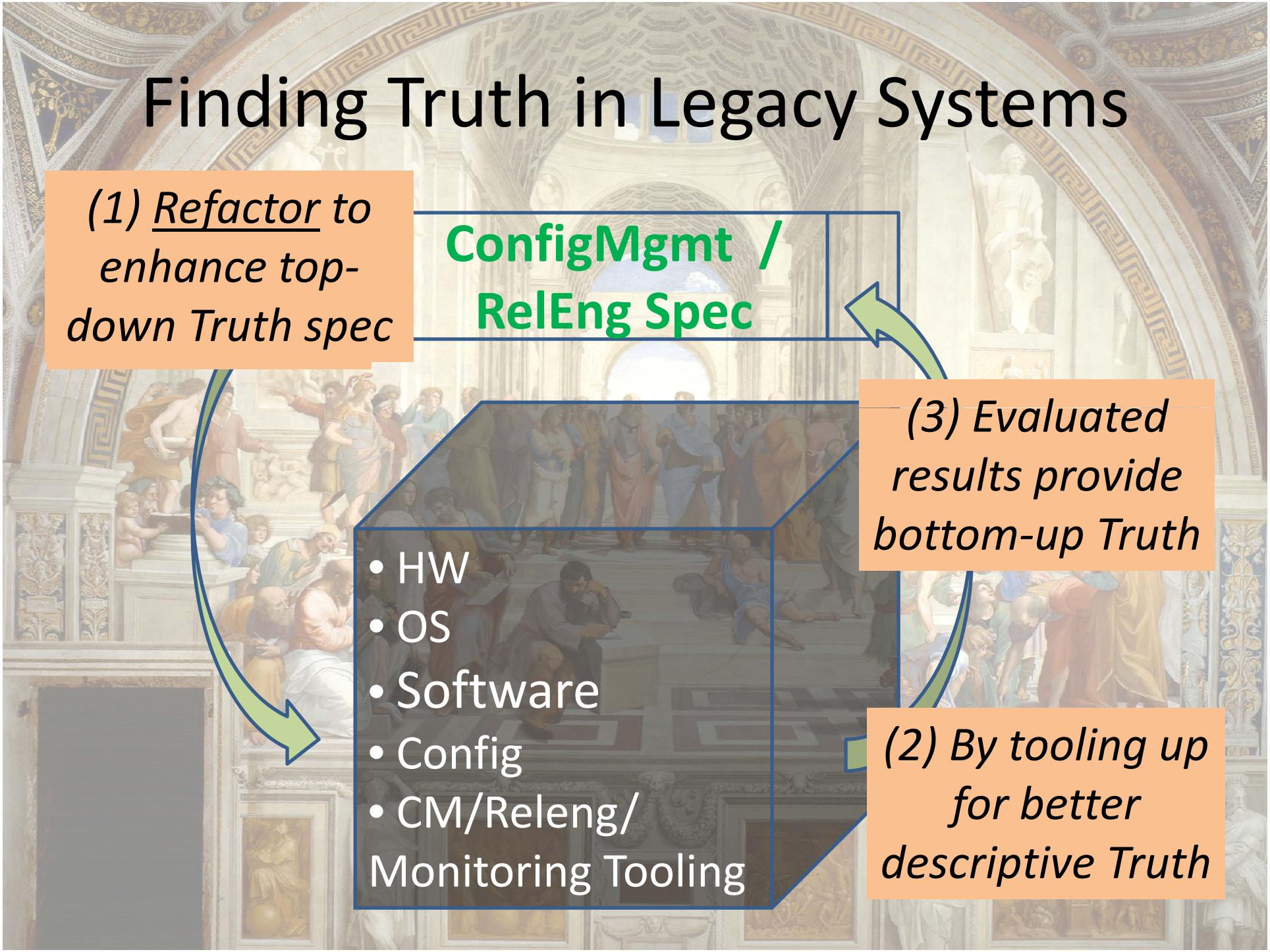
# *Making the legacy system CD-permeable*



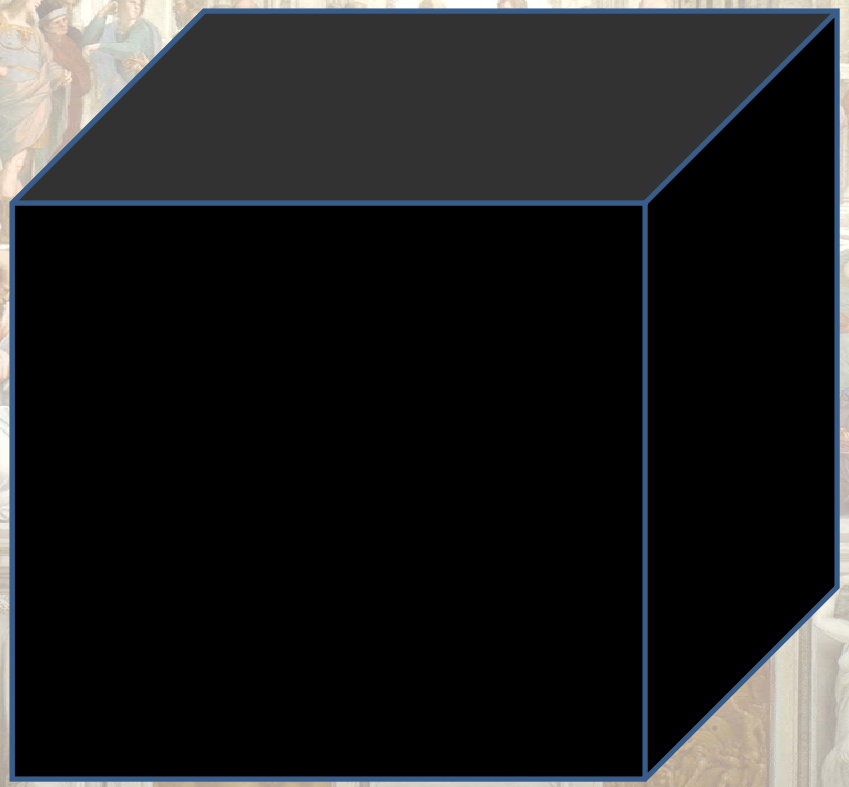
# *Making the legacy system CD-permeable*

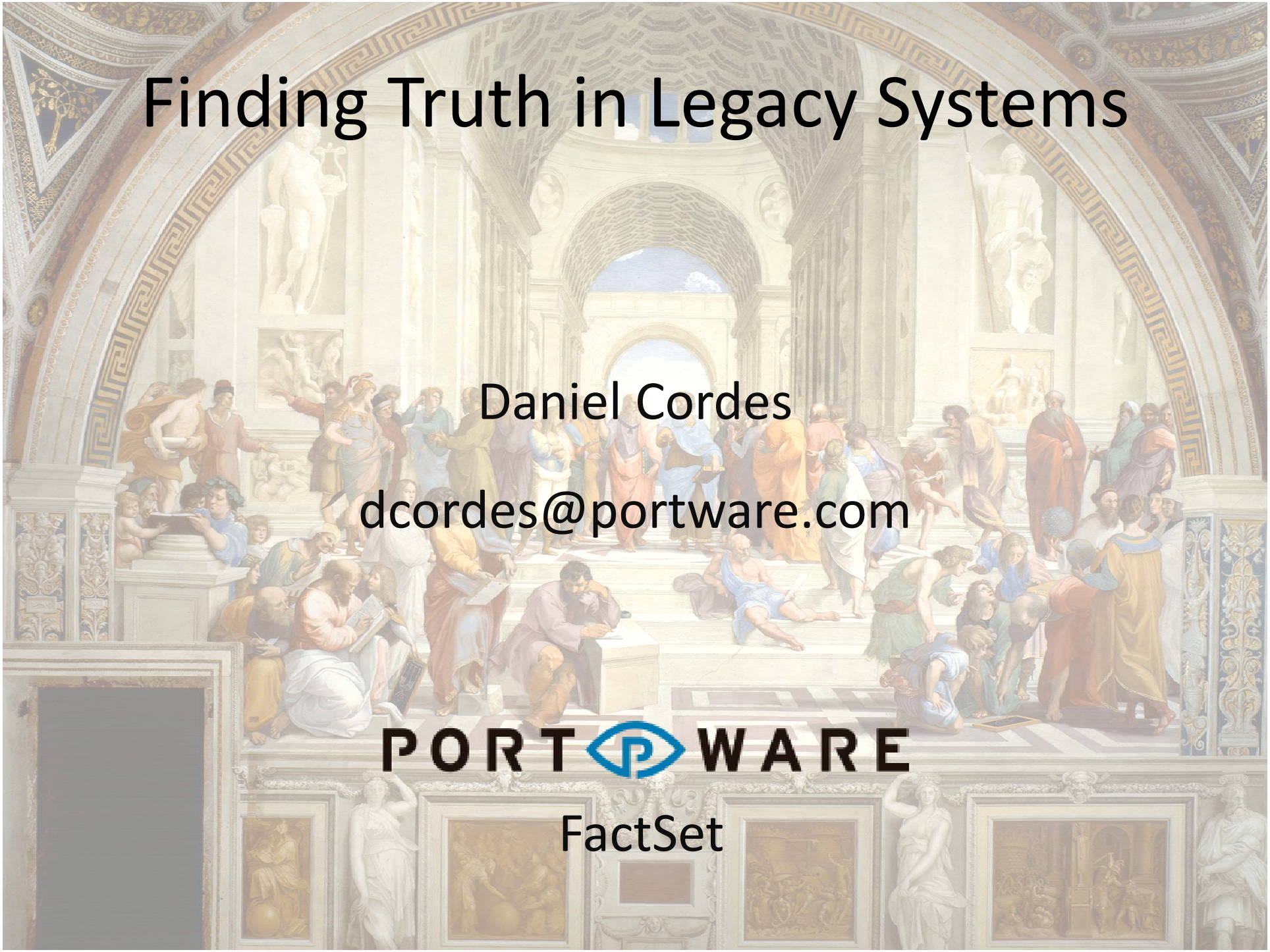


# Finding Truth in Legacy Systems



# Finding Truth in Legacy Systems





# Finding Truth in Legacy Systems

Daniel Cordes

dcordes@portware.com

PORTWARE

FactSet