TESTING

SIMPLE RULES

inspired by
Sandi Metz and Katrina Owen
talks

R. CIATTI
L. GIORDANI

TWO TYPES OF TESTS

UNIT TEST

- Near to the core functionality
- The whole universe is the single object to test
- Proves that every cell of an organism behaves correctly

INTEGRATION TEST

- Evaluate distant side-effects
- Acts on different parts of an entire ecosystem
- Proves that the beast is alive

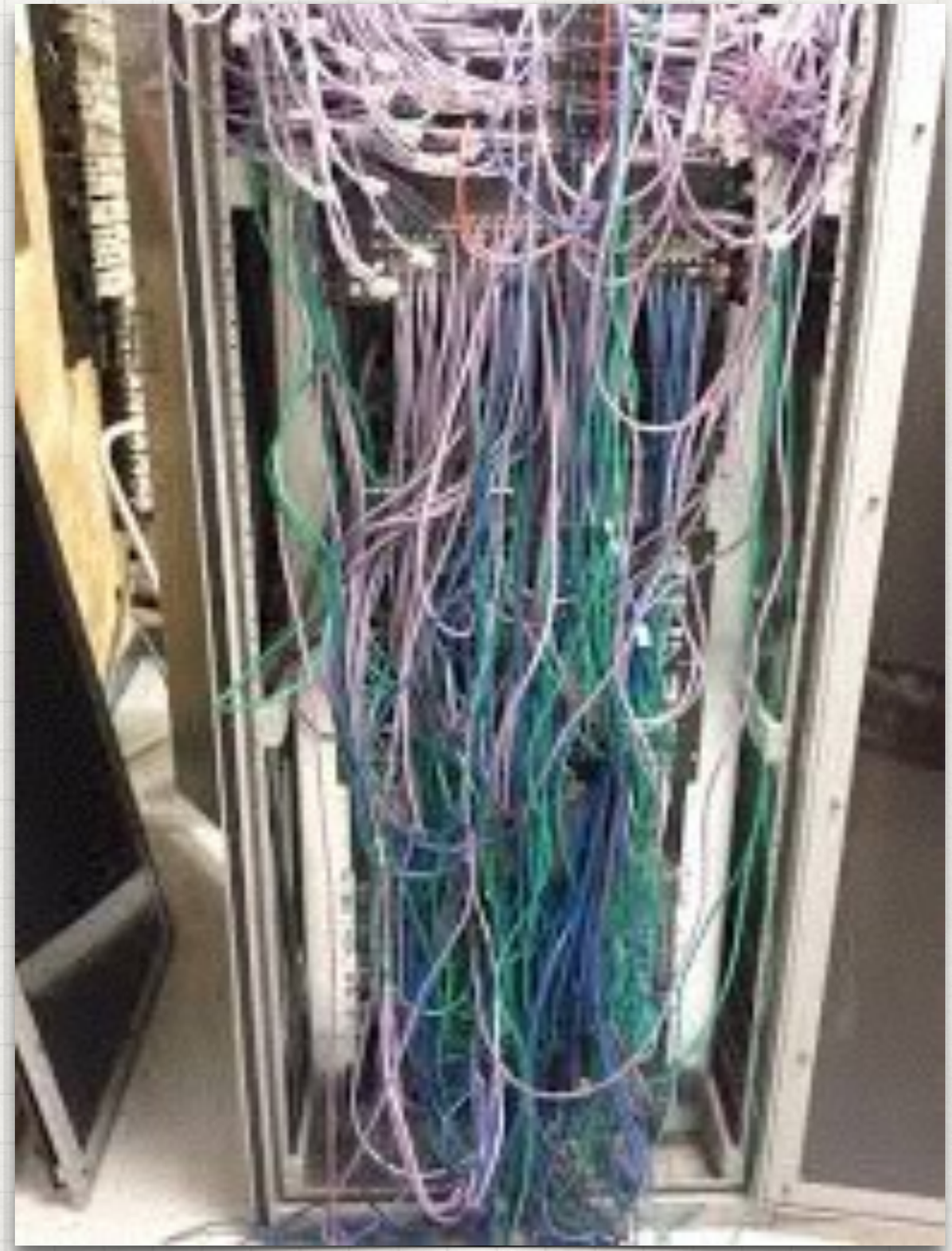
WHAT WE WANT FROM UNIT TESTS

- Thorough
- Stable
- Fast
- Few

BAD DESIGNED APP

↓

HARD TO TEST



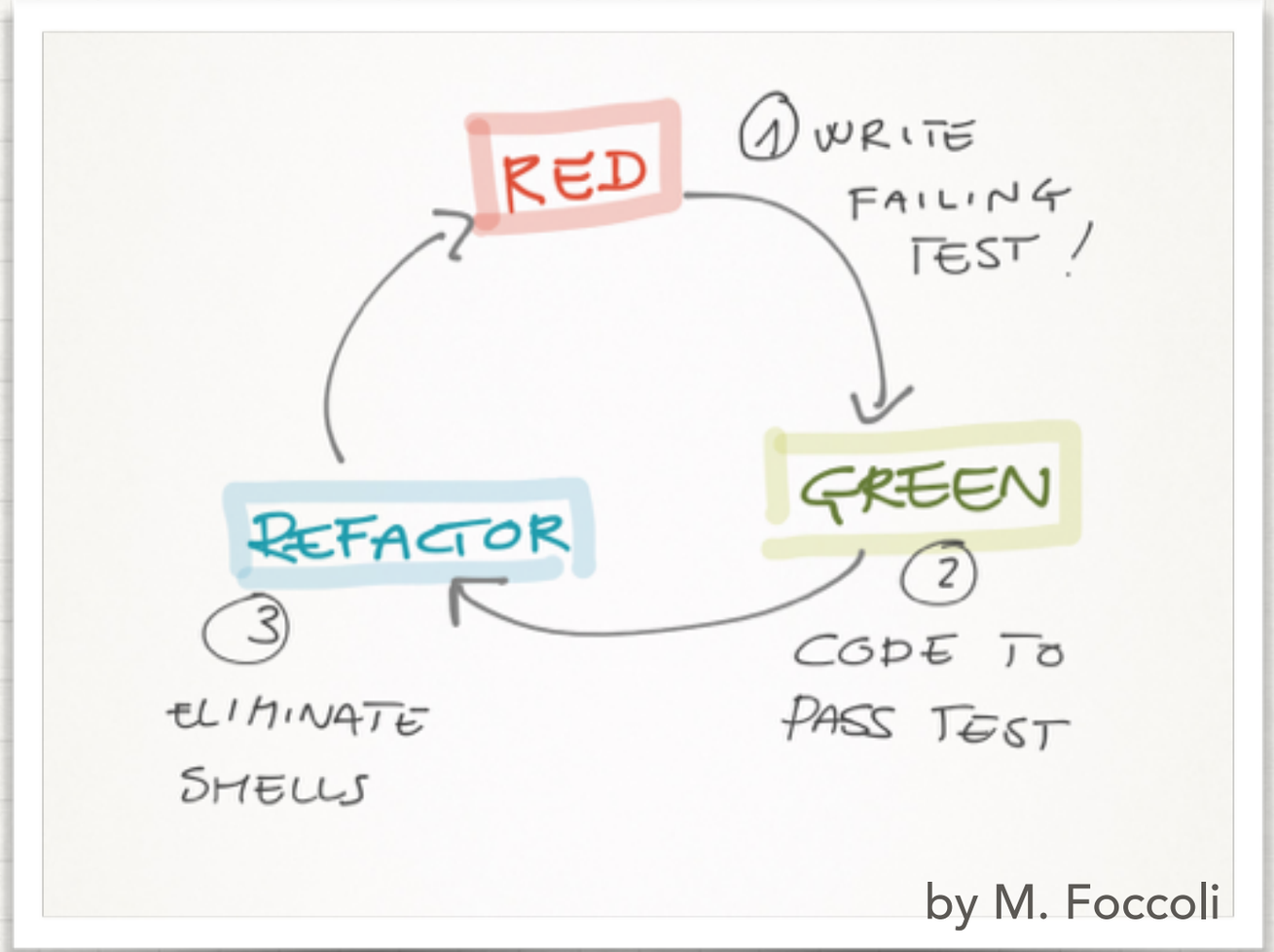
GOOD TESTING PRACTICES

GOOD APP DESIGN

TDD

Test Driven Development

- Software development process
- Repetition of very **short cycle**
- Based on **minimalism**
- It works well with **Unit Tests**



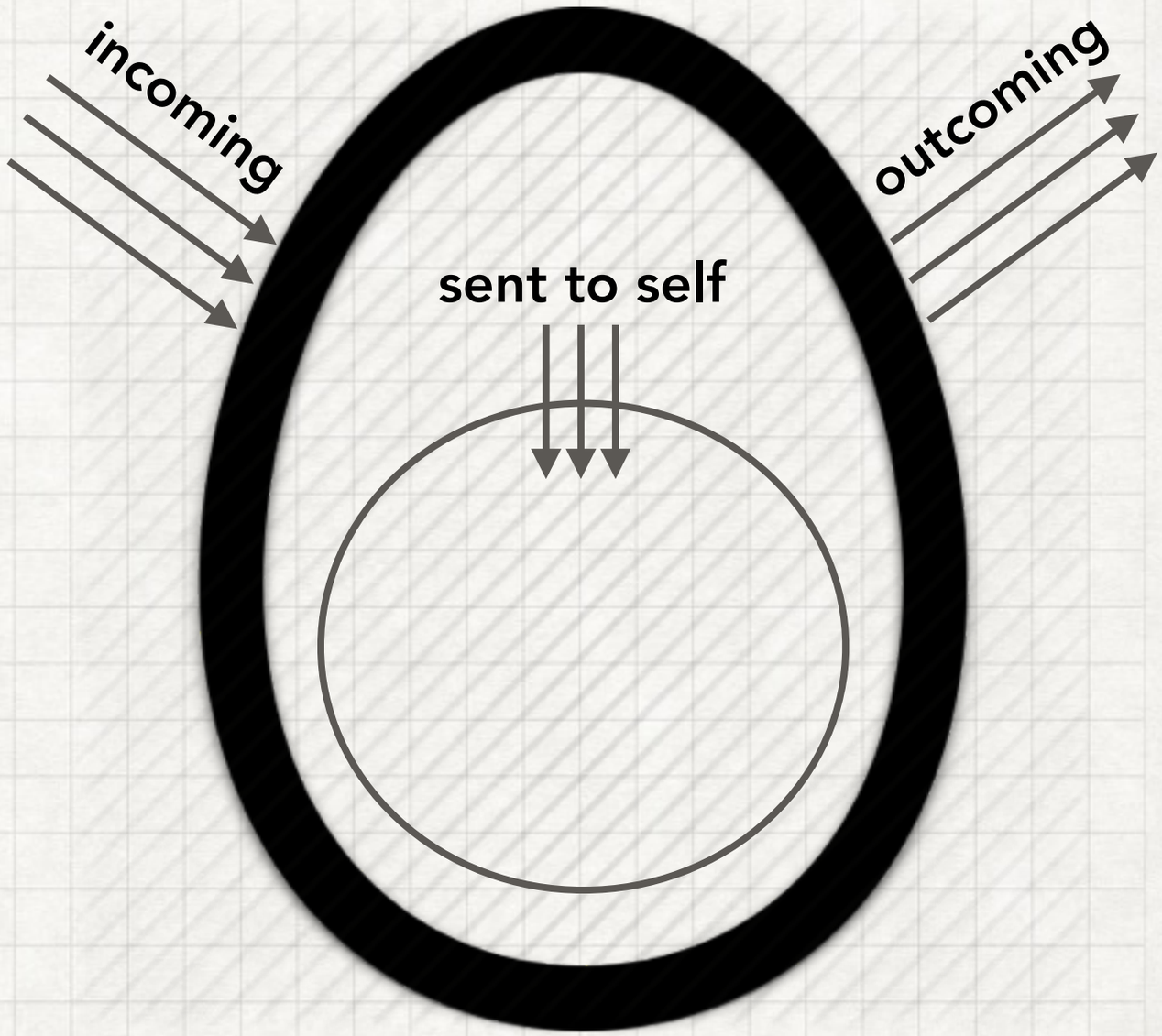
I HATE MY TESTS

- Slow
- Fragile
- Too many

FOCUS ON MESSAGES

An object

- receives message from others (**INCOMING**)
- sends message to others (**OUTCOMING**)
- send message to itself (**SENT TO SELF)**
  invisible from outside



Think to an object as an **egg** that has:

- **inside**
- **outside**

TWO FLAVOURS OF MESSAGE

QUERY	COMMAND
• does not have no side-effects • cares about what get back (result or **state**)	• has side-effects • returns nothing

MIXING QUERIES AND COMMANDS

- Often it's a smell
- Different ways to test (commands, queries or both)

BUT IT'S NOT EVIL

e.g.:
Pop an item from a queue

- retrieves an item (query part)
- changes the queue removing that item (command part)

SO...

WHAT WE HAVE TO TEST

INCOMING MESSAGES

QUERY

```
class Item:
    def __init__(self, name, total_length):
        self.name = name
        self.total_length = total_length
        self.completed_length = 0

    def remaining_length(self):
        return self.total_length - self.completed_length
```

no side-effects

```
def test_retrieve_remaining_length():
    item = Item('S1A_0SDV_20151015T202014', 4567)

    assert item.remaining_length() == 4567
```

incoming query

RULE

Test **INCOMING QUERY**

messages by

making assertions about **what they send back**

INCOMING MESSAGES

QUERY (WITH SENT TO SELF)

```
class Item:
    def __init__(self, name, total_length):
        self.name = name
        self.total_length = total_length
        self.completed_length = 0

    def get_name(self):
        return self._real_name()

    def _real_name(self):
        if self.name is not None and len(self.name) > 0:
            return self.name
        else:
            return 'NO NAME'
```

call private method

```
def test_retrieve_name():
    item = Item('S1A_0SDV_20151015T202014', 4567)

    assert item.get_name() == 'S1A_0SDV_20151015T202014'
```

test only the public method

Test only the **INTERFACE**

NOT the **IMPLEMENTATION**

INCOMING MESSAGES

COMMAND

```
class Item:
    def __init__(self, name, total_length):
        self.name = name
        self.total_length = total_length
        self.completed_length = 0

    def update_completed_length(self, length):
        self.completed_length = length

    def remaining_length(self):
        return self.total_length - self.completed_length
```

side-effects

test direct public side-effects

```
def test_retrieve_remaining_length():
    item = Item('S1A_0SDV_20151015T202014', 4567)
    item.update_completed_length(2567)

    assert item.remaining_length() == 2000
```

RULE

Test **INCOMING COMMAND**

messages by

making assertions

about

direct public effects

direct = responsibility of the last object taking part in the interaction

Receiver of Incoming message

has sole **responsibility**

for asserting the

result direct public side-effects

SENT TO SELF MESSAGES

```
class Item:
    def __init__(self, name, total_length):
        self.name = name
        self.total_length = total_length
        self.completed_length = 0

    def get_name(self):
        return self._real_name()

    def _real_name(self):
        if self.name is not None and len(self.name) > 0:
            return self.name
        else:
            return 'NO NAME'
```

sent to self

NO TESTS
FOR SELF MESSAGES

RULE

Do not test private methods

Do not make assertions about their result

Do not expect to send them

CAVEAT

Break the last rule:

- for highly complicated private method
- if there are immediate benefits

Remember that tests on private are fragile

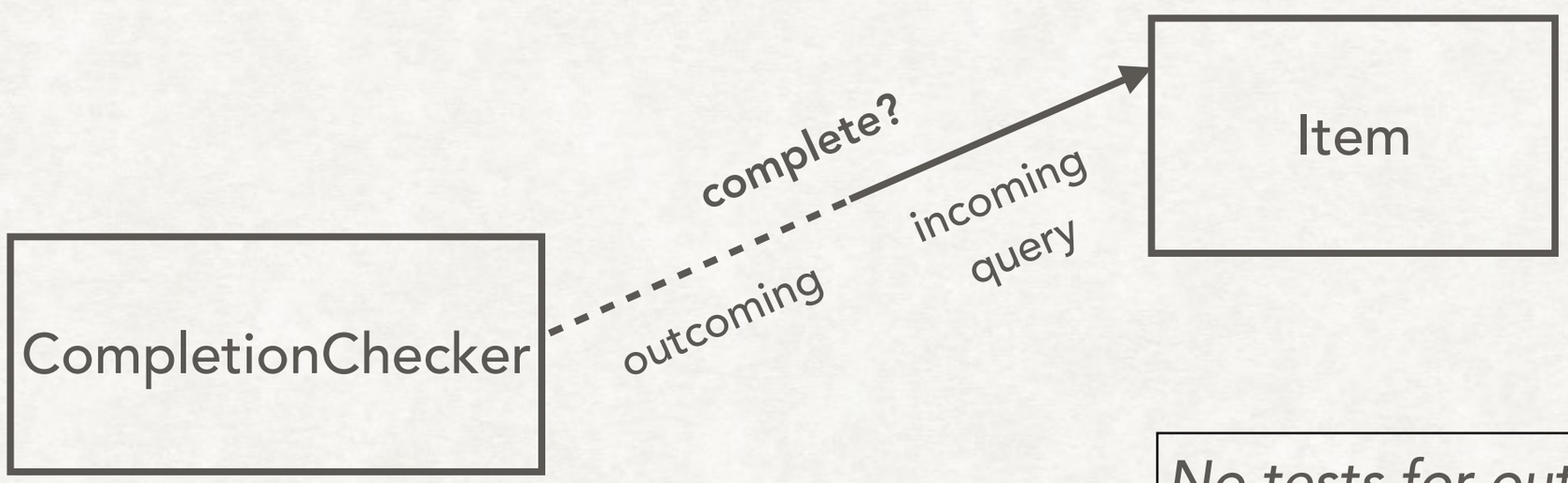
OUTGOING MESSAGES

QUERY

```
class CompletionChecker:
  …
  def execute(self):
    …
    item.is_complete()
    …
```

```
class Item:
  …
  def is_complete(self):
    return self.remaining_length == 0

  …
```



No tests for outcoming queries, they are redundant. Only the receiver must make assertions on incoming messages

RULE

Do **NOT** test

OUTGOING QUERY MESSAGES

- Do not make assertions about the result
- Do not expect to send

Outgoing query messages and sent to self messages are invisible (no visible side-effects)

If a **MESSAGE** has
NO VISIBLE SIDE-EFFECTS,
THE SENDER SHOULD NOT TEST IT

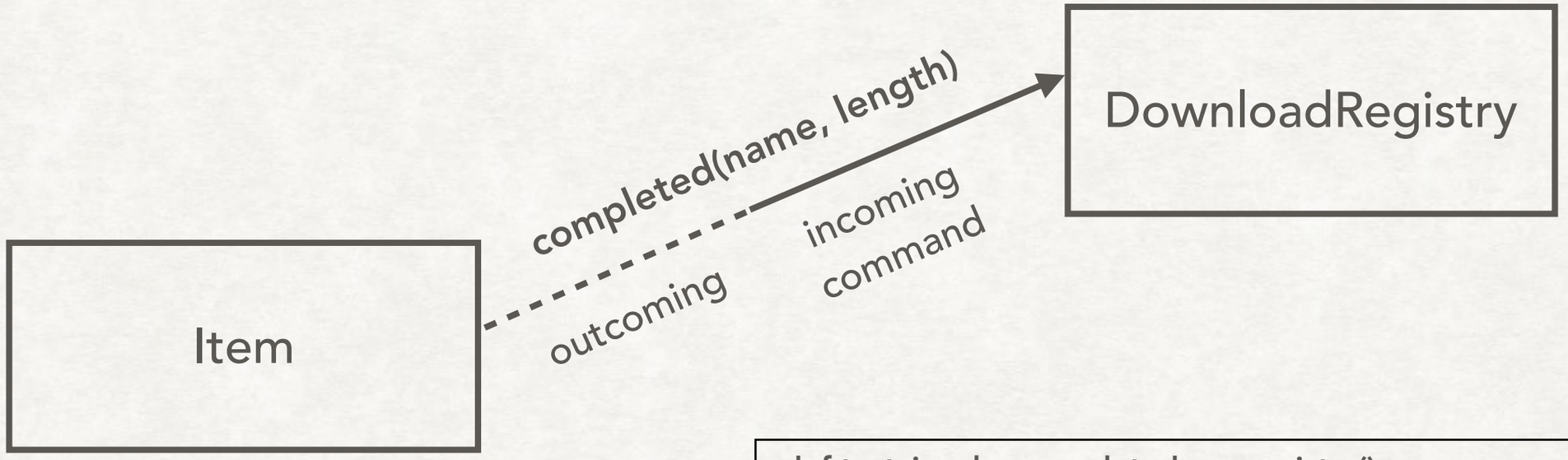
OUTGOING MESSAGES

COMMAND

```
class Item:
  …
  def update_download_length(self, length):
    self.update_completed_length(length)
    if self.remaining_length == 0:
      registry.completed(self.name, self.total_length)

  …
```

```
class DownloadRegistry:
  …
  def completed(self, name, length):
    self.completed_list.append(
      {"n": name, "l": length}
    )
  …
```




```
def test_invoke_completed_on_registry():
  registry = mock.Mock()
  i = Item('S1A_0SDV_20151015T202014', 4567, registry)
  i.update_downloaded_length(4567)

  registry.completed.assert_called_with(i.name, i.total_length)
```

RULE

Expect to send outgoing command messages

Do not make assertions on distant side-effects

Remember that Mocks usage makes your tests more fragile due to the API drift of the code you are mocking

CAVEAT

Break the last rule if side effects are stable and cheap.

Message	QUERY	COMMAND
INCOMING	**Assert** result	**Assert** direct public side-effects
SENT TO SELF		
OUTCOMING		**Expect** to Send

SUMMARY

- Be minimalist
- Use good judgement
- Test interfaces
- Trust collaborators
- Insist on simplicity

I LOVE MY TESTS

BIBLIOGRAPHY & LINKS

Videos:

- "The magic Tricks of Testing" by Sandi Metz
  https://www.youtube.com/watch?v=URSWYvyc42M&sns=em
- "467 tests, 0 failures, 0 confidence" by Katrina Owen
  https://vimeo.com/68730418

Books

- *"Practical Object-Oriented Design in Ruby"* by Sandi Metz
- *"Test-Driven Development By Example"* by Kent Beck
- *"Agile Testing A Practical Guide for Testers and Agile Teams"* by Lisa Crispin and Janet Gregory

Thanks for your attention and …

… let’s go to work !!!