Test-Driven Development:

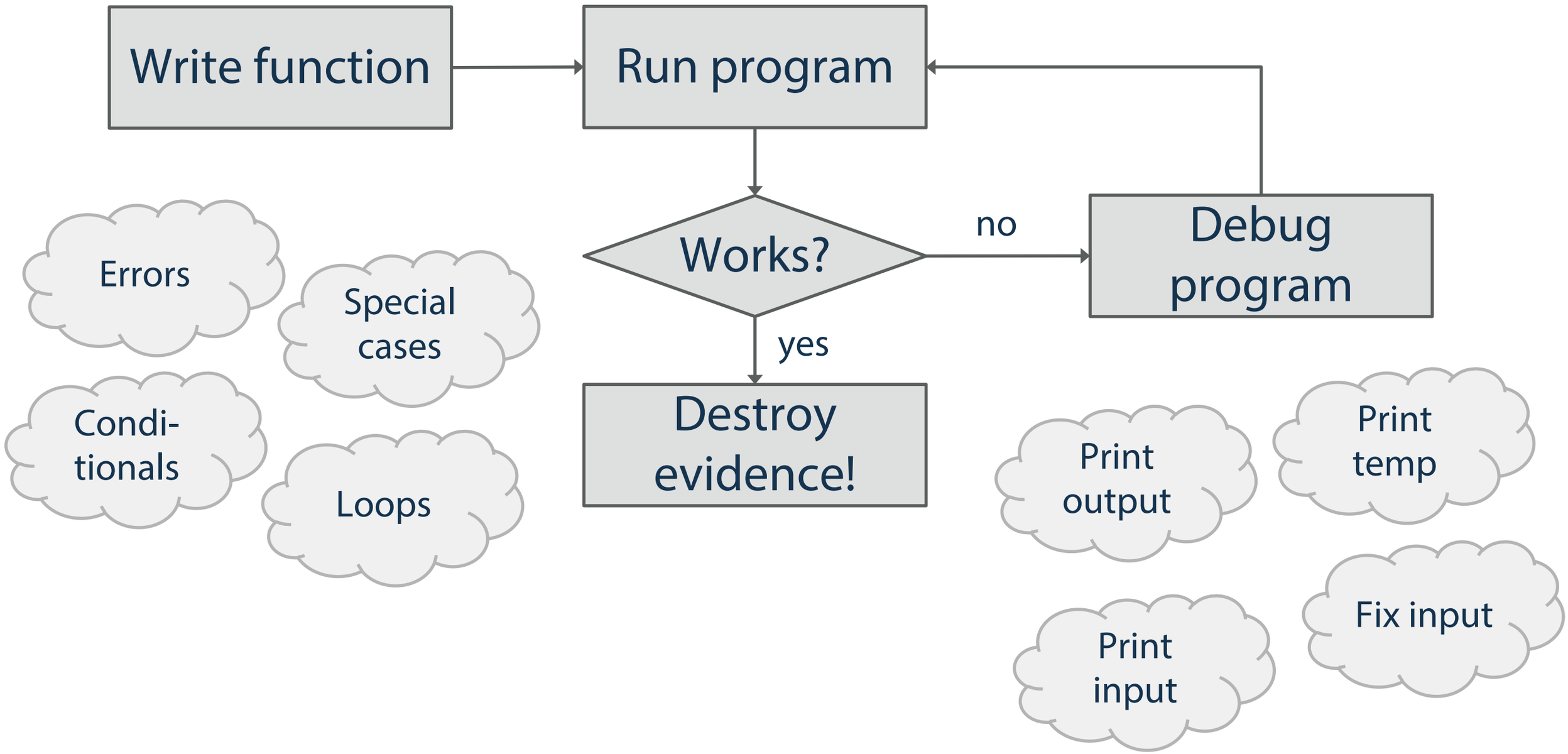
Wann? Wie? Wieso?

XPDays 2016

Michael König, Blue Yonder GmbH

blueyonder

Confessions of a Physicist



THERE'S TRIBBLES EVERYWHERE!

Would you kindly…

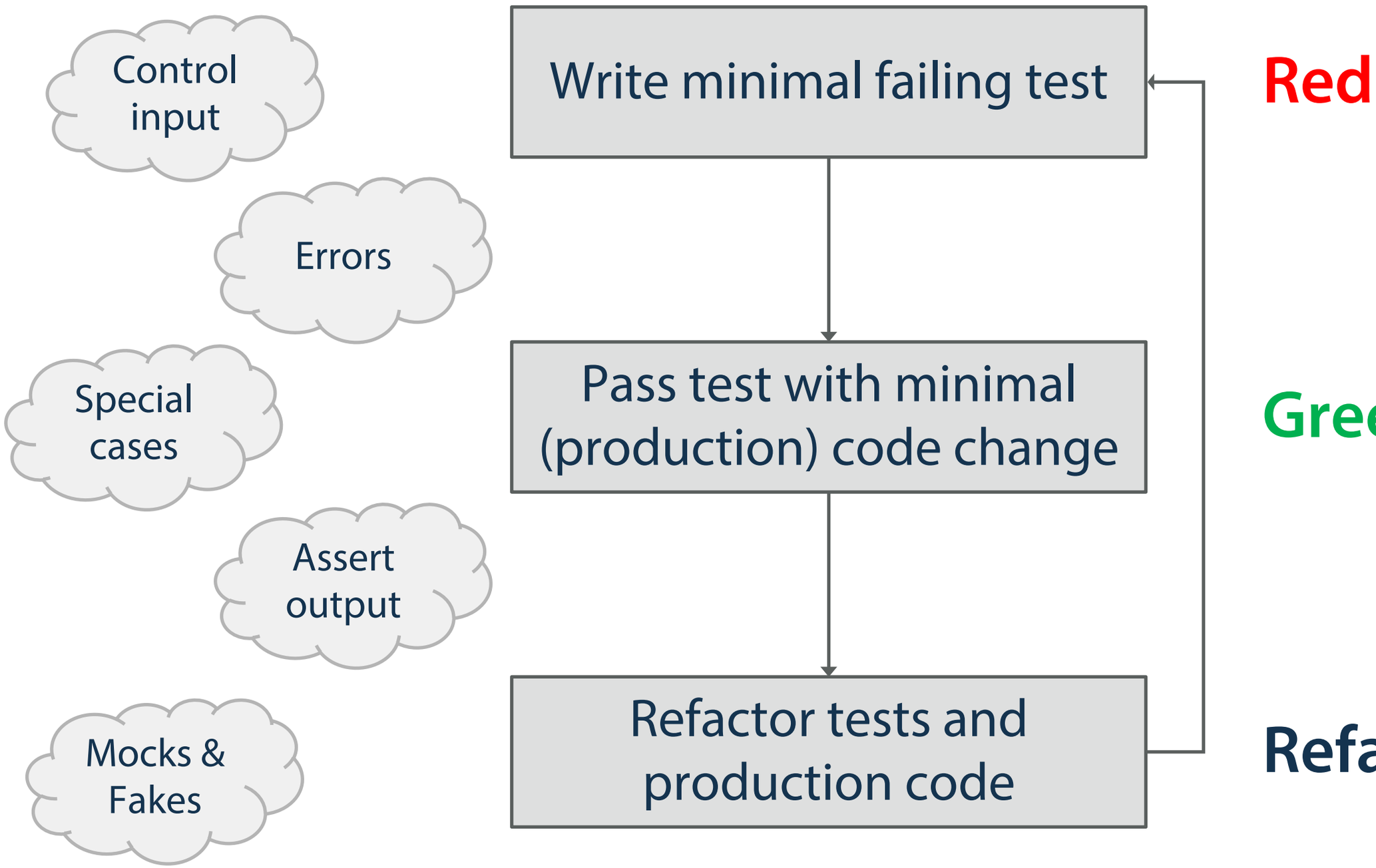
- Change!
  - Bug
  - Feature
  - Performance
- Mommy!
  - Unclear behavior
  - New & old bugs
  - Manual testing
  - Not my code!
  - Wait, that was me?!

"Never change a running system"

"Fix nothing which ain't broken"

THIS FAR,
NO FURTHER!

Test-driven development



Red: Write minimal failing test

- Minimal!
  - Prevents complexity
- Execute all tests
  - Prevents slow tests
- Assert new test fails
  - Prevents inactive tests
  - Prevents bugs in tests
  - Prevents complexity

Minimal means:

- Missing import
- Missing class
- Missing function
- One assertion a time
- Simple to complex
  - Error cases first
  - Corner cases next
  - General behavior last

Green: Pass test with minimal change

- Minimal!
  - Prevents missing tests
- Execute all tests
  - Prevents slow tests
- Assert all test succeed
  - Prevents bugs in code
  - Prevents bugs in tests

Minimal means:

- Add file stub
- Add class stub
- Add function stub
- Unconditionally raise
- Hard-coded results
- Correctly sized results
- Defer conditionals
- Defer loops

Refactor: Clean up test/production code

- Remove superseded tests
  - Better signal/noise ratio
- Clean code principles
  - Reduce complexity
- Execute all tests
  - Prevents slow tests
  - Prevents refactoring bugs
  - Prevents brittle tests

Principles

- DRY
- SRP
- SLA
- KISS
- POLA
- LoD

Hands-on session

Hands-on session: Roman numerals

- Task description & Python quickstart at
  `https://github.com/blue-yonder/tdd_exercise`
- Virtual environment recommended

```
> git clone https://github.com/blue-yonder/tdd_exercise
> cd tdd_exercise
> python setup.py test
```

I'M SENSING
CONFUSING EMOTIONS

Is TDD that *painfully* slow?

- Babysteps… really?
  - Not necessarily
  - Write failing test
  - Write obvious implementation

- TDD lets you work as fast as you can

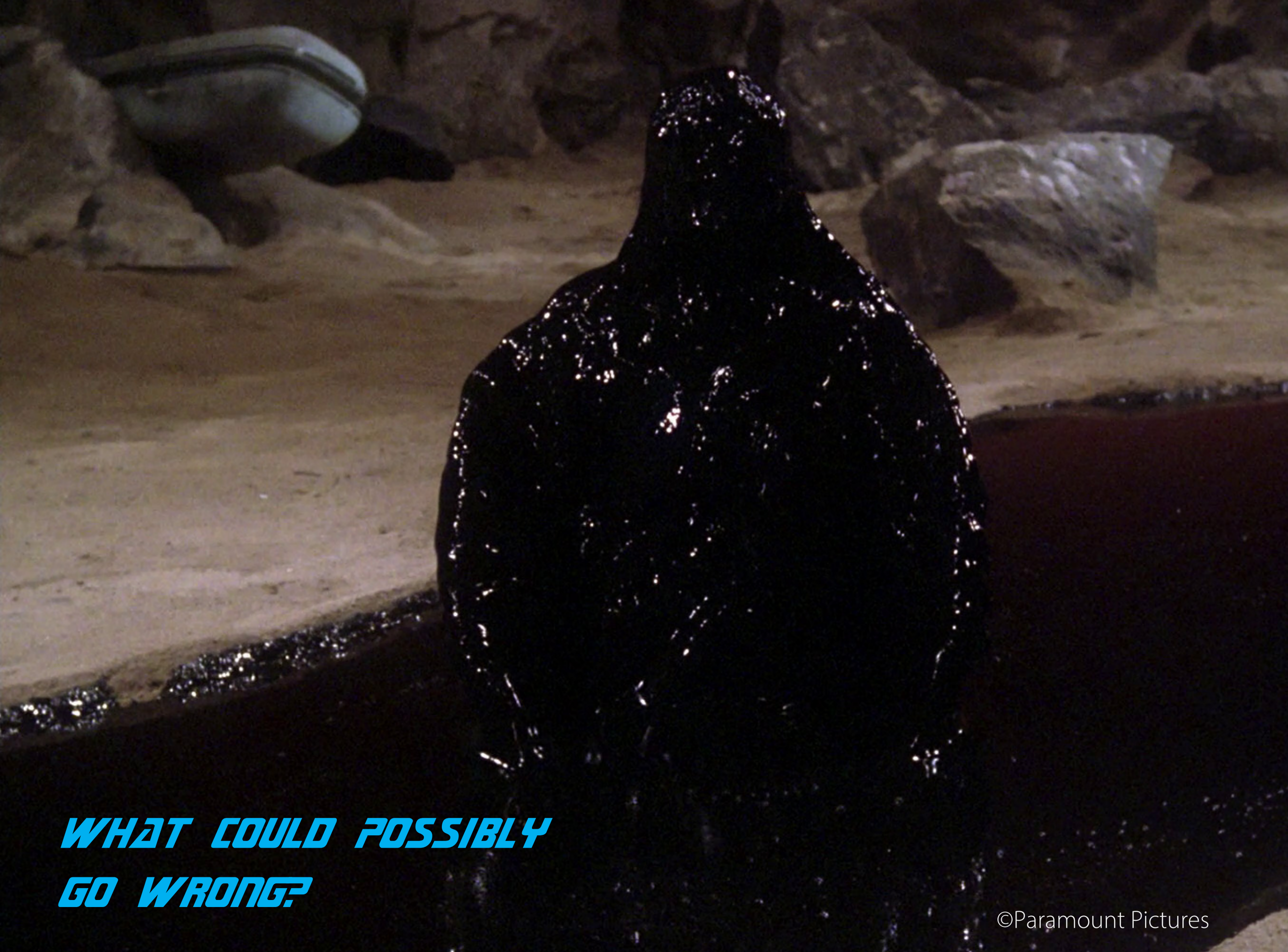
> "The best race drivers know when to *brake*"

TDD boosts your code

- Impact on code
  - Modular design
  - Cleaner code
  - Less bugs
- Impact on tests
  - Full automation
  - 100% coverage
  - Executable specs

TDD boosts your work life

- Steady sense of progress
- Ease of mind
- Courage



Lots of things, apparently

- Die 10 goldenen Regeln für schlechte Tests
  - *Tilmann Glaser, Peter Fichtner*

- Wann soll ich mocken?
  - *David Völkel*

Tests to avoid

- Can’t say no
- Overly complex tests
- Parrots
- Riddles
- Nitroglycerine
- Mocking hell
- Refactoring clamps

Countermeasures

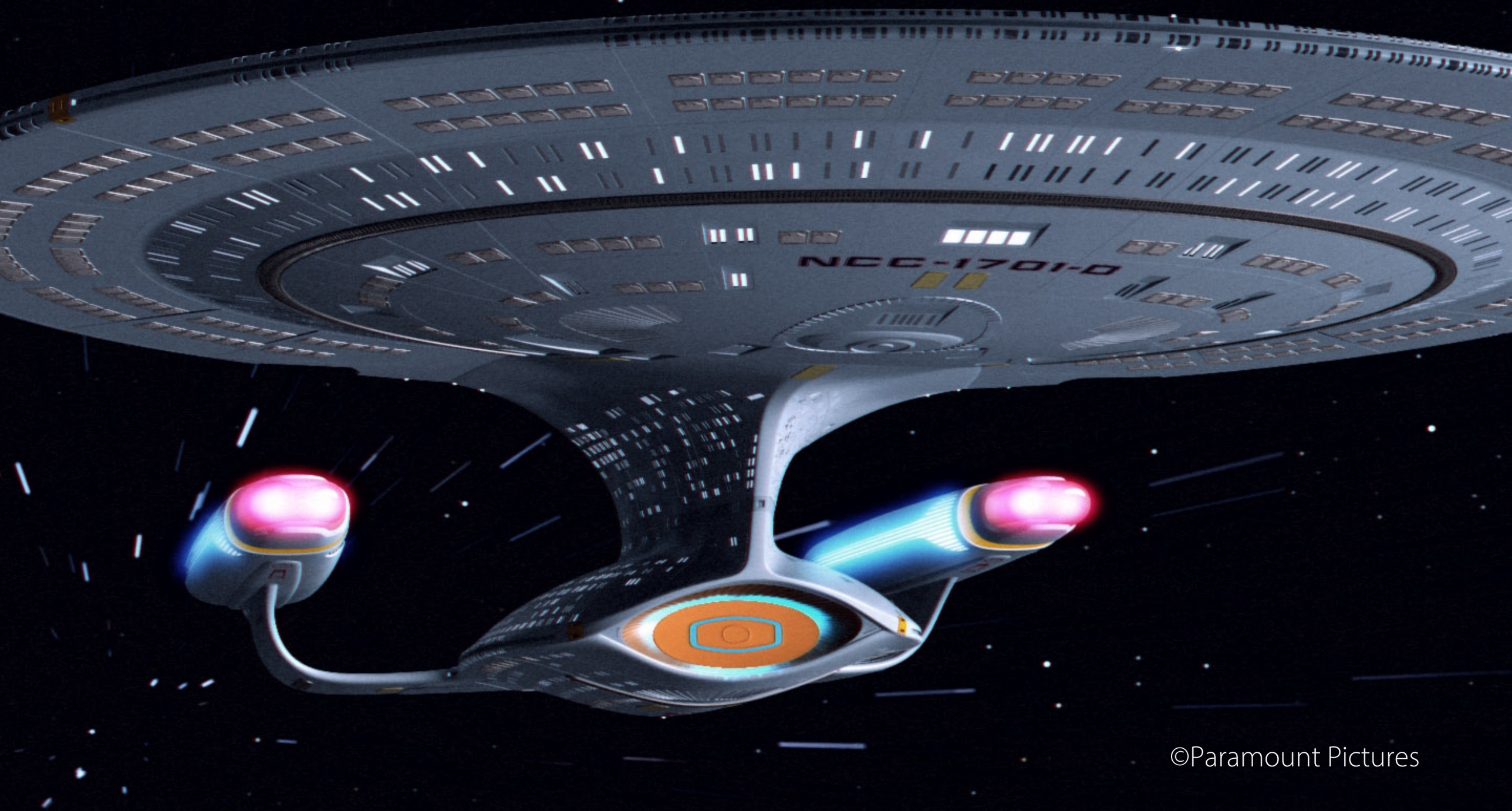
- Split
- Helpers for setup / assertions
- Hand-picked examples
- Express intent in names
- Eliminate *all* randomness
- Prefer fakes/stubs over mocks

- Improve production code design

- Last resort: Drop

TO BOLDLY DO TDD

WHERE NO ONE HAS DONE BEFORE



Be honest: Is TDD perfect for anything?

- Prototype code
  - Quickly moving target without perspective
- Performance optimizations
  - Non-functional, without prior expectation
- Concurrent programming
  - *Hard* to control
- Declarative code
  - *How* more complex than *what*

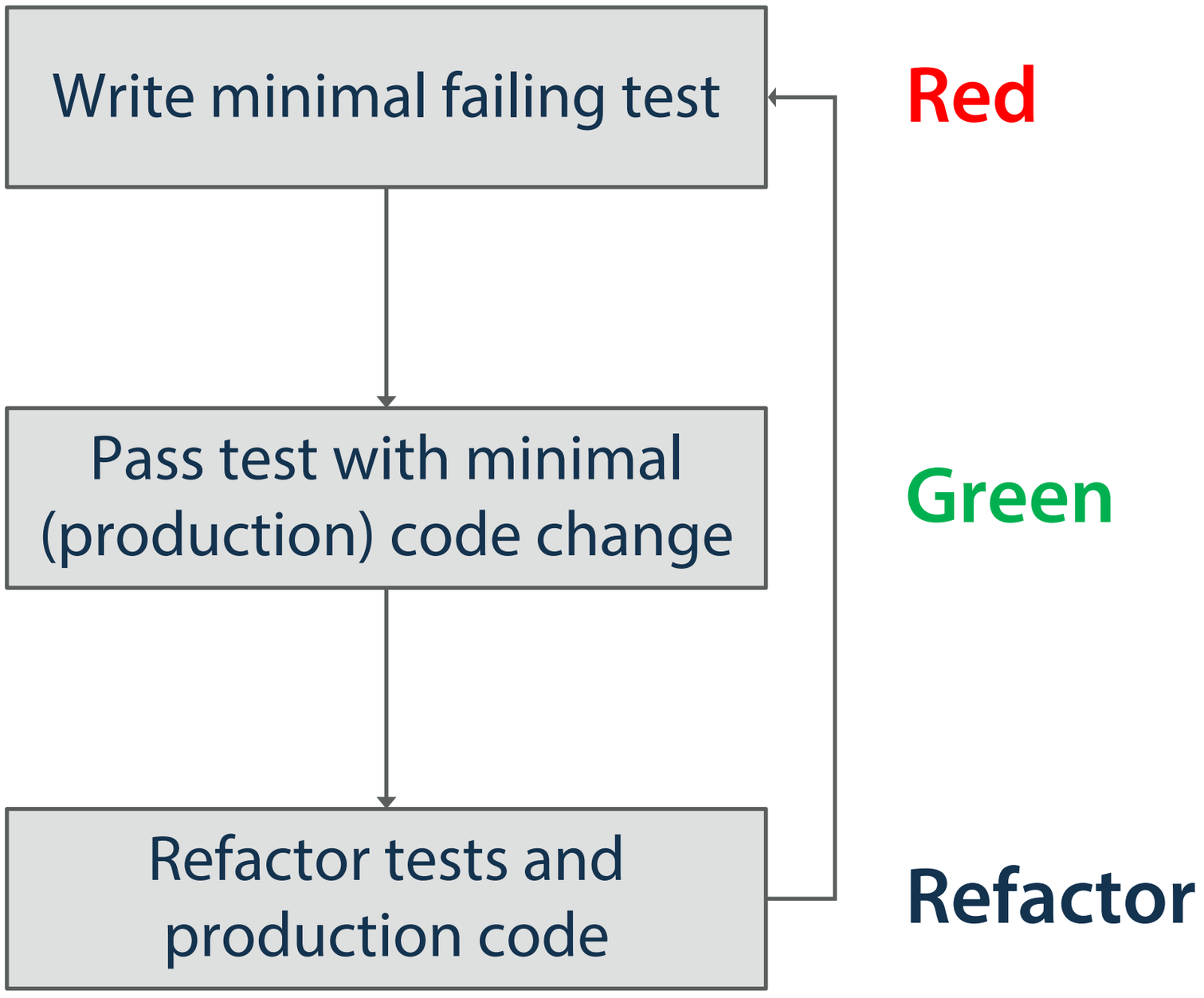
DEBRIEFING

TDD: One tool in your belt

- Great for
  - Functional correctness
  - Black/white situations
  - Production code
  - Single-threaded code
  - Non-declarative code

- Don’t be dogmatic about it

Test-driven development



Further material

- Test-driven development by example
  - *by Kent Beck*

- Code Katas:
  - Poker hand classification
  - Hangman game
  - Roman to Arabic numerals
  - …