The Snake in the Tar Pit

Complex Systems with Python

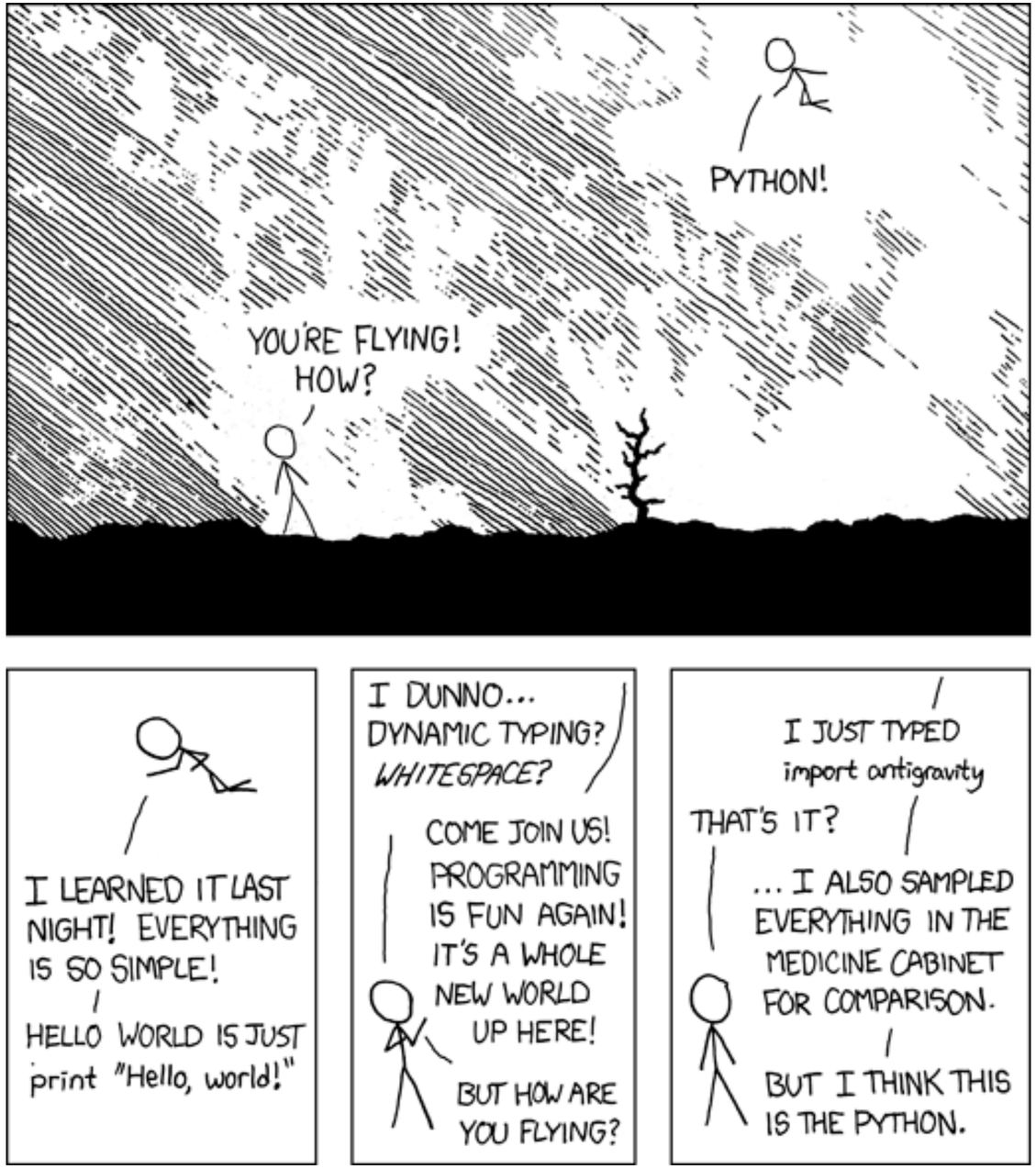
27.10.2017
PyConDE

Stephan Erb
@ErbStephan

Blue Yonder GmbH

Photo by Tormod Sandtorv https://flic.kr/p/asN2QD

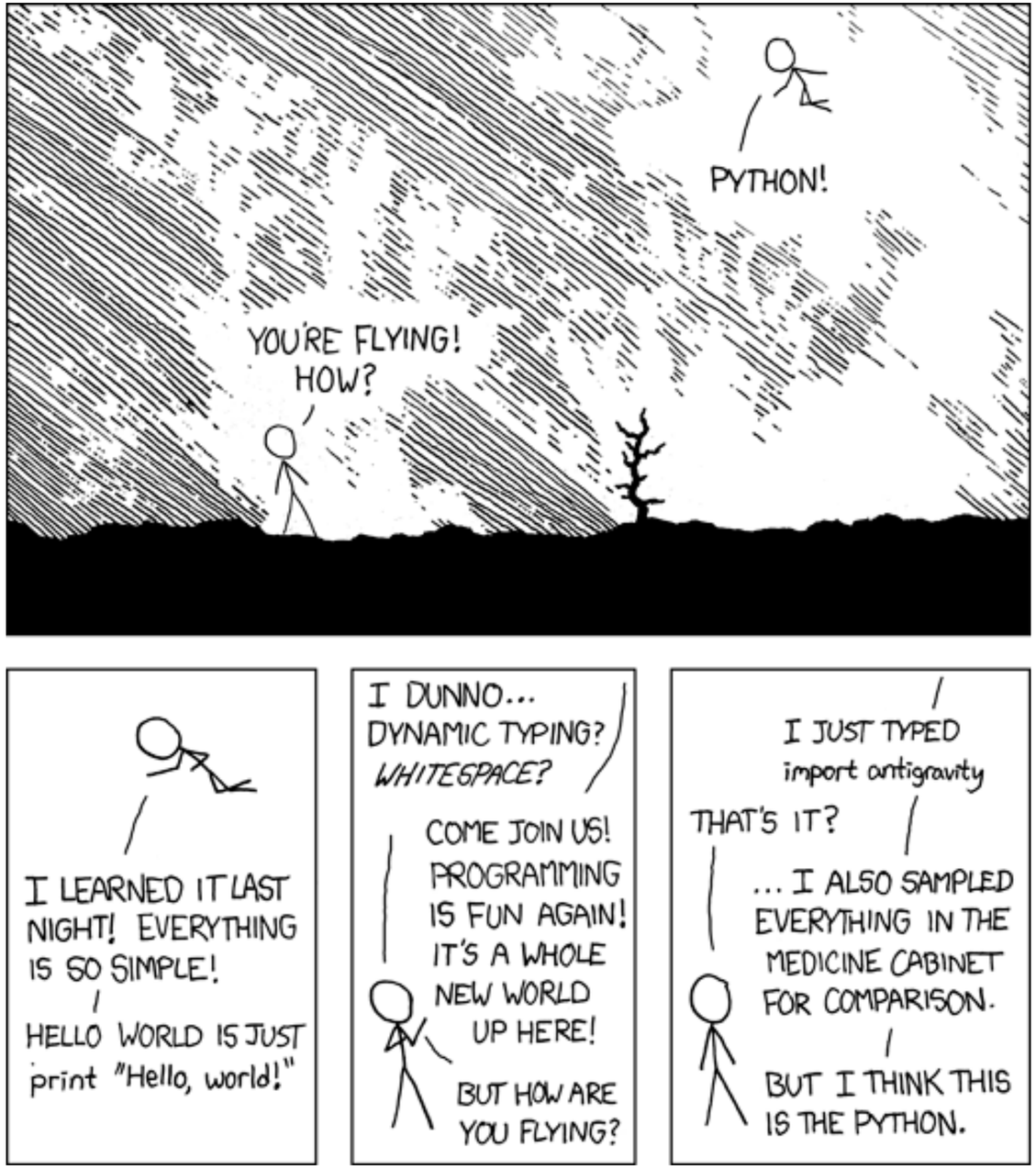
import **antigravity**



https://www.xkcd.com/353/

import **antigravity**

import **greenfield**



https://www.xkcd.com/353/

Axioms of Engineering

(by @dmontauk)

- most engineering time is spent **debugging**
- most debugging time is spend **looking for information**
- most time spent looking for information is **because the code / system is unfamiliar**

https://www.youtube.com/watch?v=yXkOgnyK4Hw

Axioms of Engineering

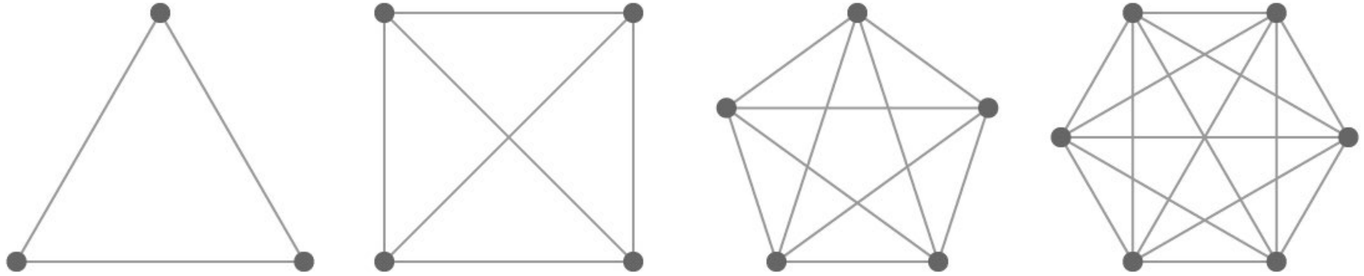
(by @dmontauk)

- most engineering time is spent **debugging**
- most debugging time is spend **looking for information**
- most time spent looking for information is **because the code / system is unfamiliar**

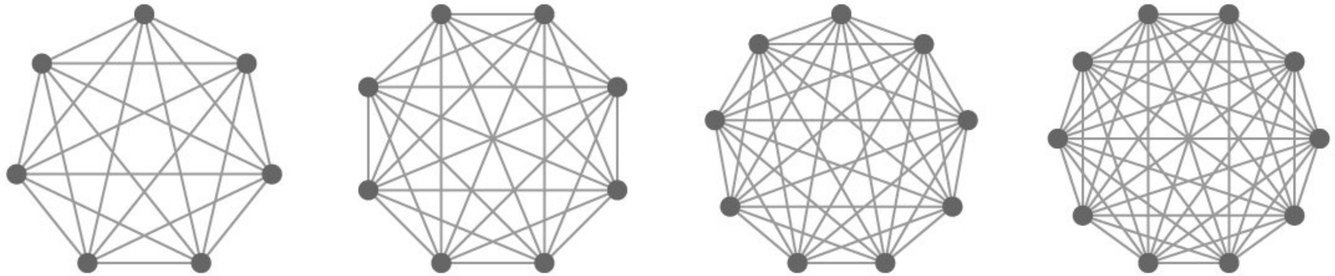
https://www.youtube.com/watch?v=yXkOgnyK4Hw

Examples:

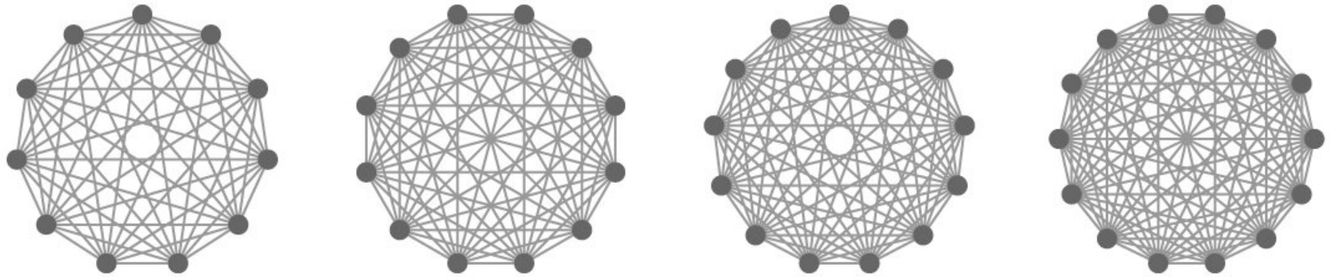
- Test fails on Jenkins but works fine locally
- Service slow in production today but you are not aware of any changes



3 people, 3 lines | 4 people, 6 lines | 5 people, 10 lines | 6 people, 15 lines



7 people, 21 lines | 8 people, 28 lines | 9 people, 36 lines | 10 people, 45 lines



11 people, 55 lines | 12 people, 66 lines | 13 people, 78 lines | 14 people, 91 lines

https://twitter.com/RichRogersIoT/status/914577613300965377

Building Understanding

Informal Reasoning

- “run code your head”
- code and design reviews
- pair programming
- …

Testing

- unit tests, integration tests, …
- load tests
- chaos engineering

Building Understanding

Informal Reasoning

- “run code your head”
- code and design reviews
- pair programming
- …

Testing

- unit tests, integration tests, …
- load tests
- chaos engineering

by getting better we
create less bugs

by getting better we
find more bugs

complexity
destroys
understanding

Photo by Tormod Sandtorv https://flic.kr/p/asN2QD

```python
def get_double(self, key):
    return self._elements[key] +
           self._elements[key]
```

Scenarios

- returns `2 * value`
- throws `KeyError`
- returns something *wrong*
- crashes when the network is down
- …

Law of Leaky Abstractions

All non-trivial abstractions are leaky:

- key value store as dict
- reliable TCP
- declarative SQL
- …

They help us time building but not time learning

https://www.joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/

Applies to **dependencies** too…

```python
import requests

def query_json(self, path):
    resp = requests.get("https://api.foo.com" + path)
    return resp.json()
```

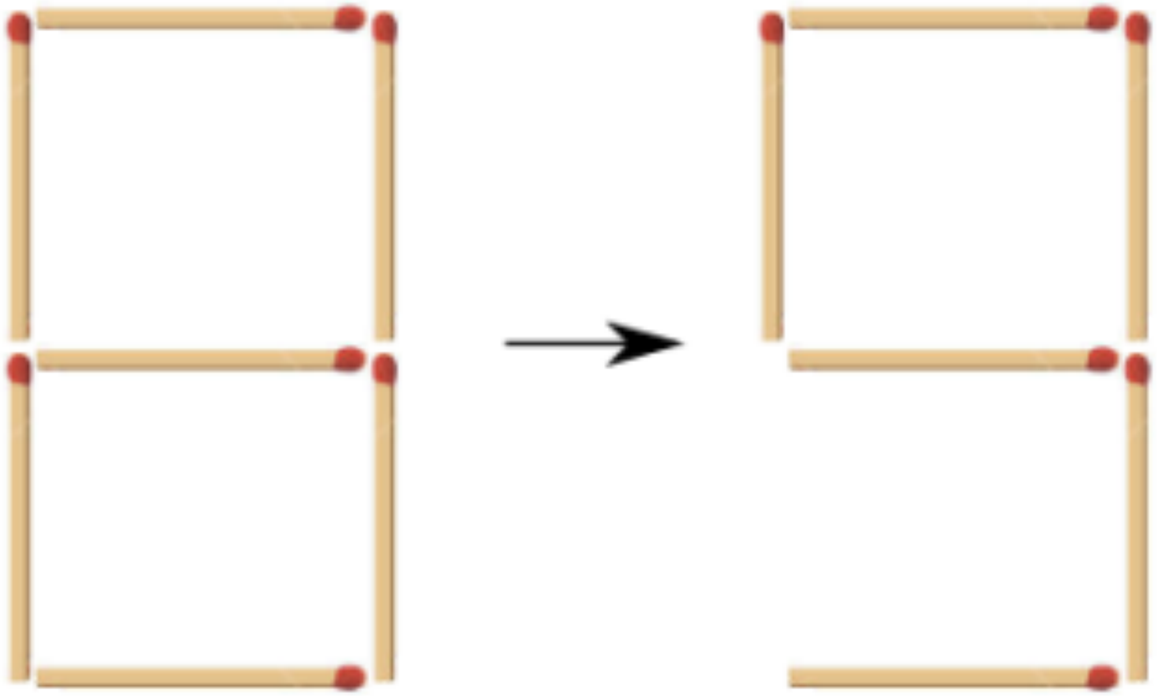
Scenarios

- returns response data as expected
- returns response but with HTTP error code
- fails to connect
- slow to connect
- hangs forever
- …

```python
from prometheus_client import Summary
from pybreaker import CircuitBreaker
import requests

REQUEST_TIME = Summary(
    'api_request_seconds',
    'Time spend on API request')

api_breaker = CircuitBreaker(
    fail_max=5,
    reset_timeout=60)

@api_breaker
@REQUEST_TIME.time()
def query_json(self, path, timeout=10):
    resp = requests.get(
        url="https://api.foo.com" + path,
        timeout=timeout)
    resp.raise_for_status()
    return resp.json()
```

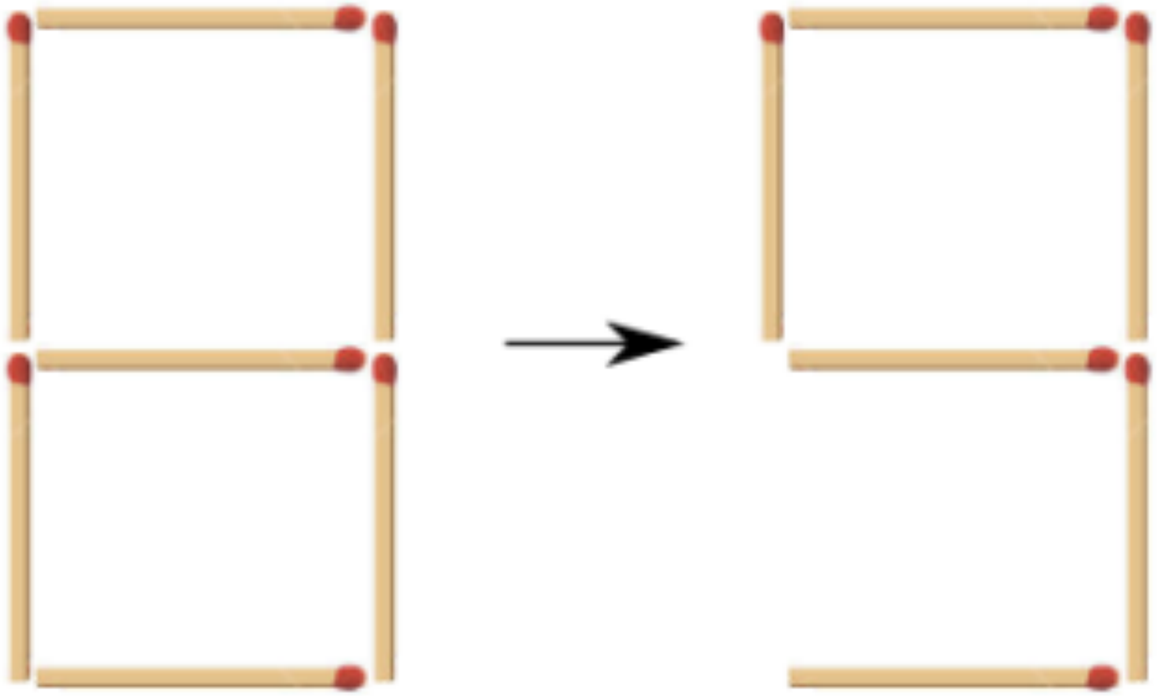
State & Mutability

(by @sebineubauer)



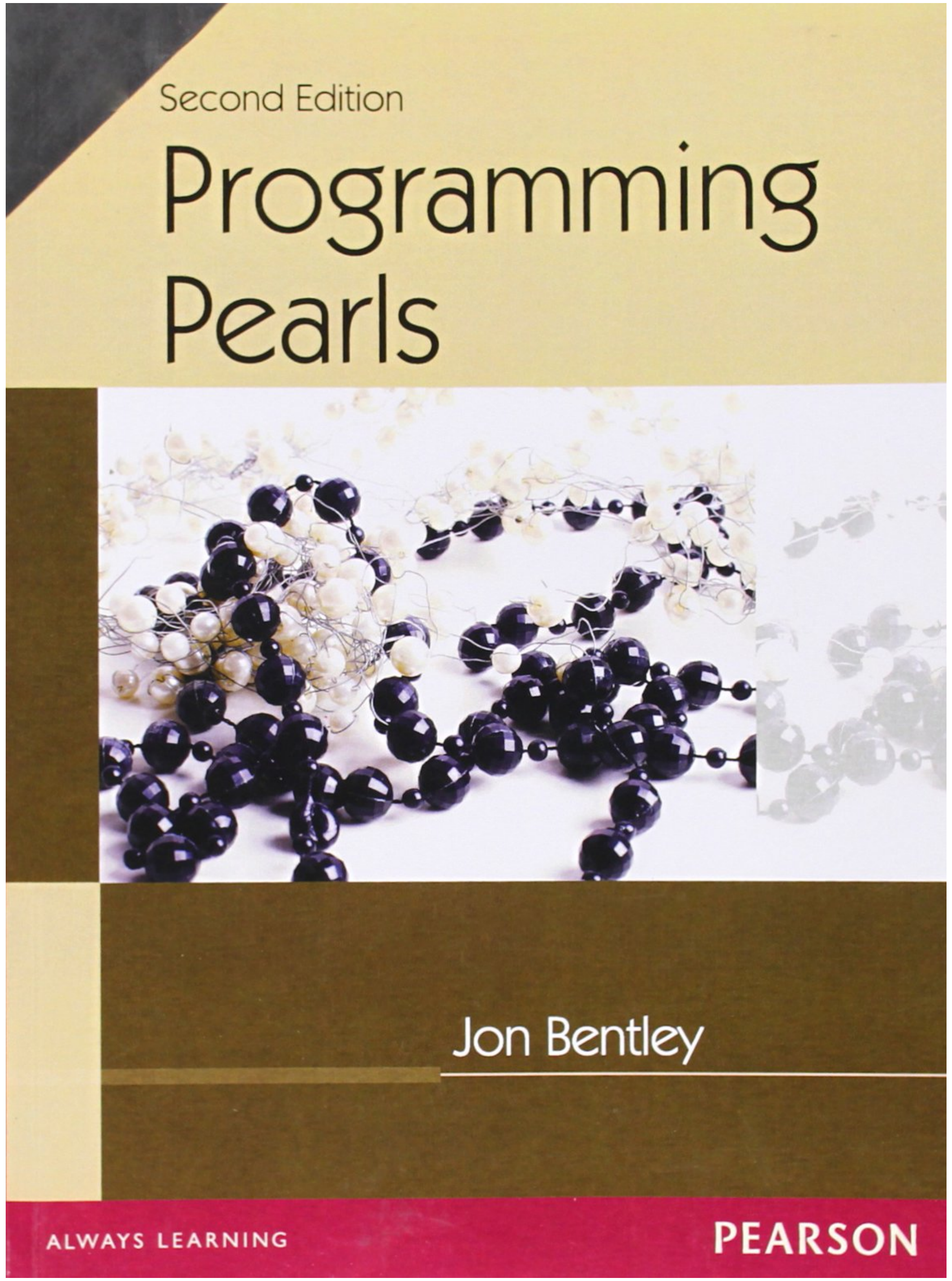
State & Mutability

(by @sebineubauer)



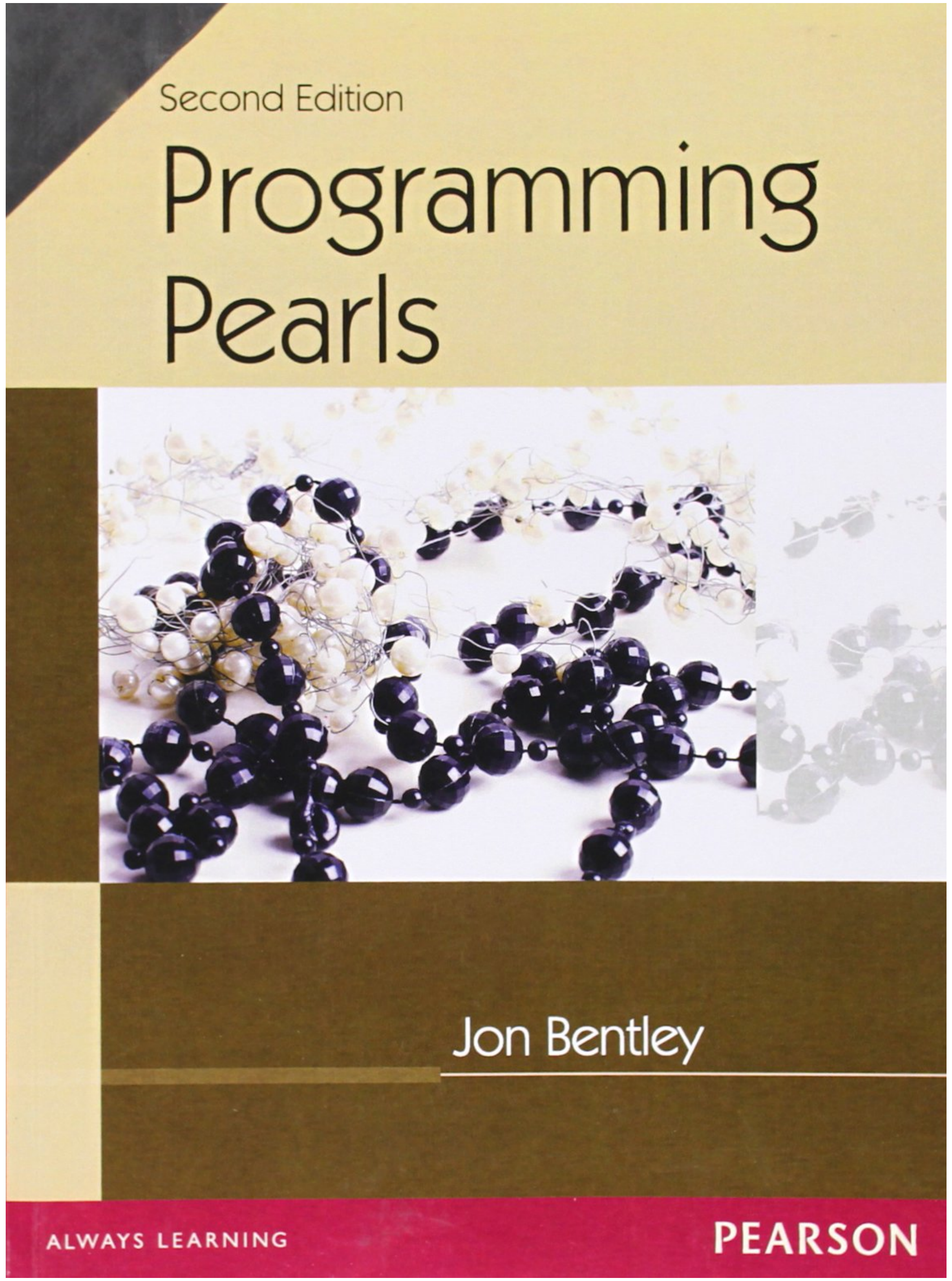
mutable approach
45 combinations

immutable approach
10 combinations



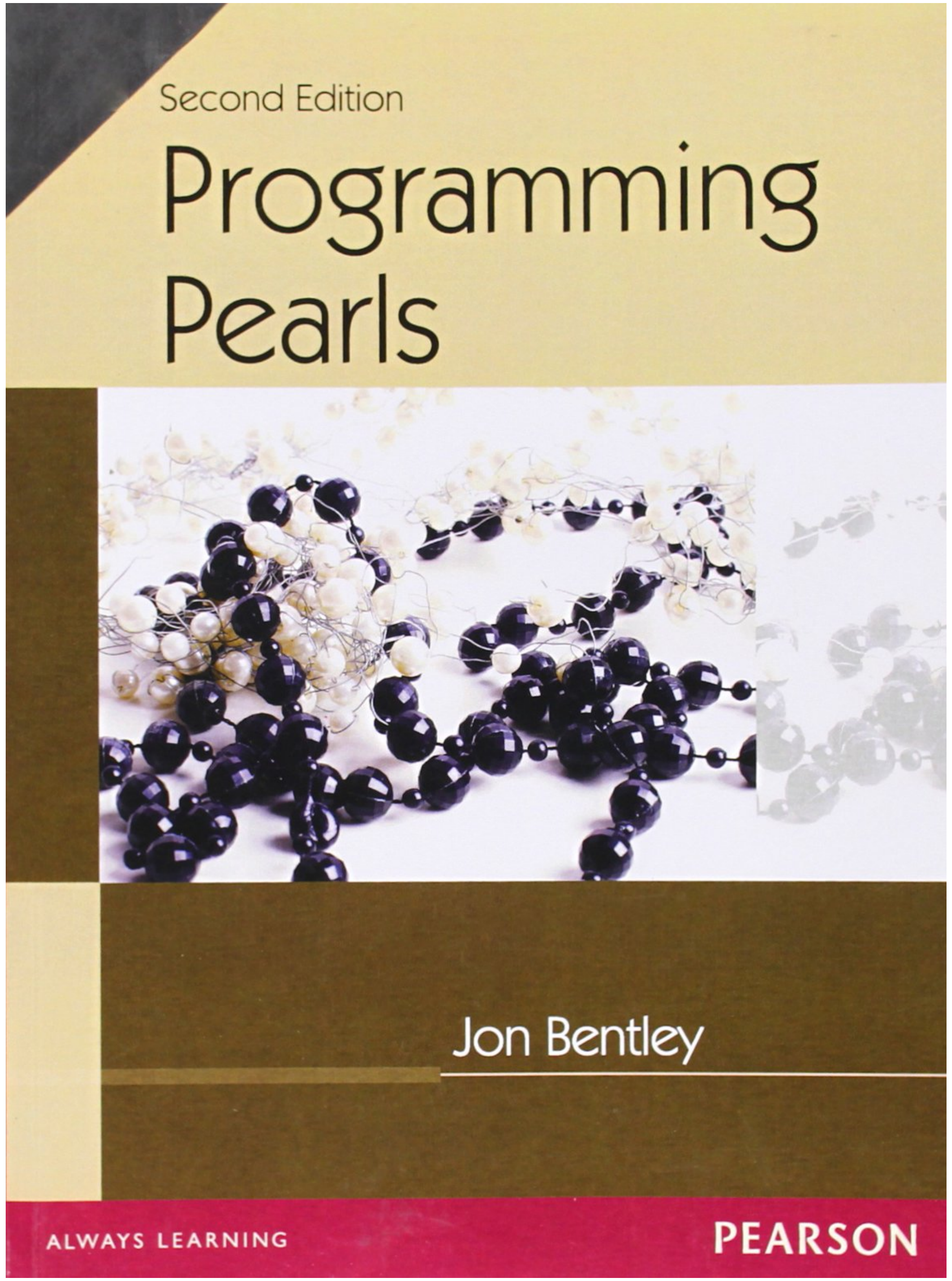
Performance Tuning

- Problem Definition
- System Structure (Architecture)
- Algorithms and Data Structures
- Code Tuning
- System Software (libs, DB, …)
- Hardware



Performance Tuning

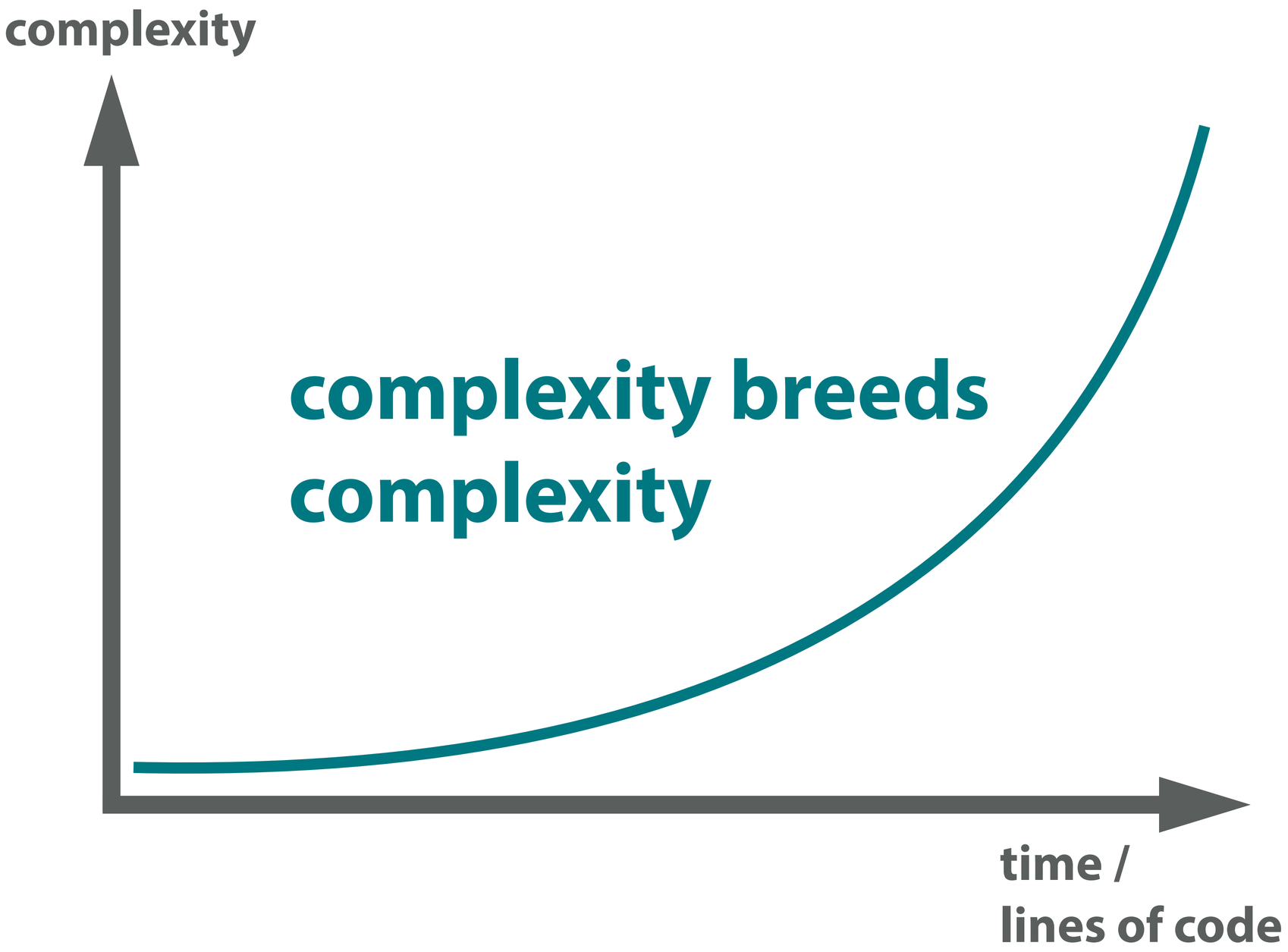
- Problem Definition
- System Structure (Architecture)
- Algorithms and Data Structures
- Code Tuning
- System Software (libs, DB, …)
- Hardware



Performance Tuning

- Problem Definition
- System Structure (Architecture)
- Algorithms and Data Structures
- Code Tuning
- System Software (libs, DB, …)
- Hardware

Photo by Rob https://flic.kr/p/7NtGzn


complexity
complexity breeds complexity
time /
lines of code

build more with less

8 chords
120 million records sold

End-To-End Arguments in System Design

J. H. SALTZER, D. P. REED, and D. D. CLARK
Massachusetts Institute of Technology Laboratory for Computer Science

This paper presents a design principle that helps guide placement of functions among the modules of a distributed computer system. The principle, called the end-to-end argument, suggests that functions placed at low levels of a system may be redundant or of little value when compared with the cost of providing them at that low level. Examples discussed in the paper include bit-error recovery, security using encryption, duplicate message suppression, recovery from system crashes, and delivery acknowledgment. Low-level mechanisms to support these functions are justified only as performance enhancements.

CR Categories and Subject Descriptors: C.0 [**General**] Computer System Organization—*system architectures*; C.2.2 [**Computer-Communication Networks**]: Network Protocols—*protocol architecture*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems; D.4.7 [**Operating Systems**]: Organization and Design—*distributed systems*

General Terms: Design

Additional Key Words and Phrases: Data communication, protocol design, design principles

1. INTRODUCTION

Choosing the proper boundaries between functions is perhaps the primary activity of the computer system designer. Design principles that provide guidance in this choice of function placement are among the most important tools of a system designer. This paper discusses one class of function placement argument that has been used for many years with neither explicit recognition nor much conviction. However, the emergence of the data communication network as a computer system component has sharpened this line of function placement argument by making more apparent the situations in which and the reasons why it applies. This paper articulates the argument explicitly, so as to examine its nature and to see how general it really is. The argument appeals to application requirements and provides a rationale for moving a function upward in a layered system closer to the application that uses the function. We begin by considering the communication network version of the argument.

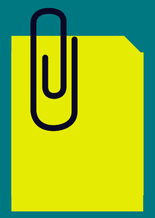
This is a revised version of a paper adapted from End-to-End Arguments in System Design by J. H. Saltzer, D.P. Reed, and D.D. Clark from the 2nd International Conference on Distributed Systems (Paris, France, April 8–10) 1981, pp. 509–512. © IEEE 1981
This research was supported in part by the Advanced Research Projects Agency of the U.S. Department of Defense and monitored by the Office of Naval Research under contract N00014-75-C-0661.
Authors' address: J. H. Saltzer and D. D. Clark, M.I.T. Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139. D. P. Reed, Software Arts, Inc., 27 Mica Lane, Wellesley, MA 02181.
Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
© 1984 ACM 0734-2071/84/1100-0277 $00.75
ACM Transactions on Computer Systems, Vol. 2, No. 4, November 1984, Pages 277–288.

End-To-End Arguments in System Design

- [...] a rationale for **moving function upward in a layered system, closer to the application** that uses the function.
- [...] **functions placed at low levels** of a system may be **redundant or of little value** when compared with the cost of providing them at that low level
- **Low level** mechanisms [...] are justified only as a **performance enhancement**.

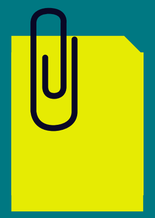
Distributed Systems Example

Send a file from one node to another



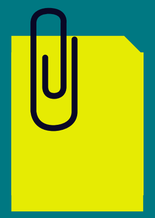
Distributed Systems Example

Send a file from one node to another



Distributed Systems Example

Send a file from one node to another



Gzip file (incl checksum)

Thought Experiment

"

If I would **extend the scope** of my task or problem a little bit - **would it still make sense?**

Will my current approach solve the problem end-to-end?

"

Summary

- engineers spend lot of time debugging because they don't understand their systems
- understanding is killed by complexity
- **more code** (often) **means more complexity**
- we therefore have to **be more like modern talking** and get **more done with less** (code)
- "end-to-end" argument can be a way to achieve that

TLDR: Go home, deprecate some "features", and delete some code

References

(on top of the other links within the talk)

- **Out of the Tar Pit**
  - http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.93.8928
- **End-to-End Arguments in System Design**
  - http://web.mit.edu/Saltzer/www/publications/endtoend/endtoend.pdf
- **How to build stable systems**
  - https://medium.com/@jlouis666/how-to-build-stable-systems-6fe9dcf32fc4
- **Modern Software Over-Engineering Mistakes**
  - https://medium.com/@rdsubhas/10-modern-software-engineering-mistakes-bc67fbef4fc8
- **Law of Leaky Abstractions**
  - https://www.joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/

Thank you!

27.10.2017
PyConDE

Stephan Erb
@ErbStephan

Blue Yonder GmbH

BlueYonder
Best decisions, delivered daily

Blue Yonder
Best decisions, delivered daily

Blue Yonder GmbH
Ohiostraße 8
76149 Karlsruhe
Germany
+49 721 383117 0

Blue Yonder Software Limited
19 Eastbourne Terrace
London, W2 6LG
United Kingdom
+44 20 3626 0360

Blue Yonder Analytics, Inc.
5048 Tennyson Parkway
Suite 250
Plano, Texas 75024
USA

BlueYonder
Best decisions, delivered daily