ECE444: Software Engineering

Open Source

Shurui Zhou

The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

Learning Goals

- Understand the terminology “free software” and explain open source culture and principles.
- Reason about the tradeoffs of the open source model on issues like quality and risk

https://www.youtube.com/watch?v=a8fHgx9mE5U

The culture

GNU/Linux

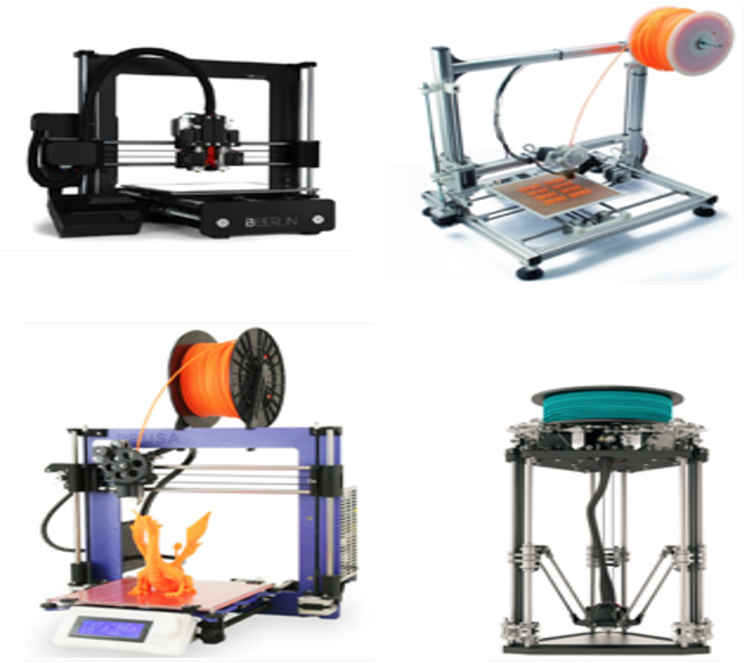
- "I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones."

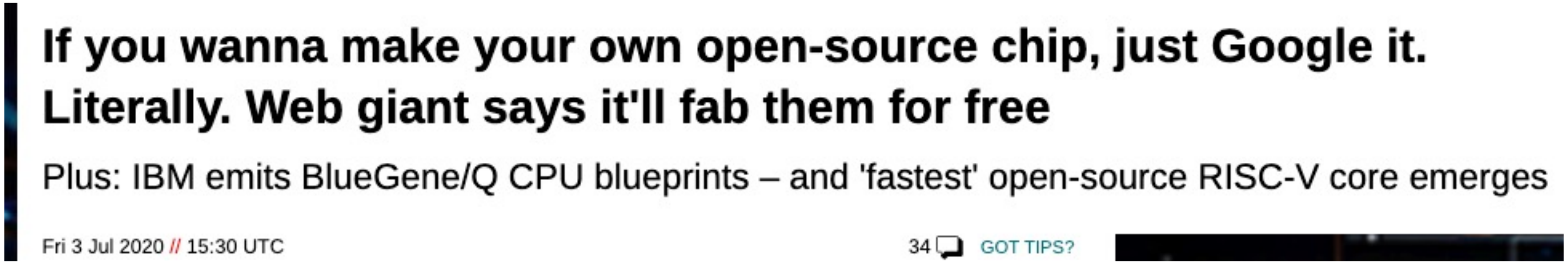
  -- Linus Torvalds

- In the second quarter of 2013, 187.4 million Android (a Linux derivative) and 1.8 million Linux phones shipped compromising over 80% of the shipping market share for smartphones.

open source

- Software
- Hardware
- What else?





If you wanna make your own open-source chip, just Google it. Literally. Web giant says it'll fab them for free

Plus: IBM emits BlueGene/Q CPU blueprints – and 'fastest' open-source RISC-V core emerges

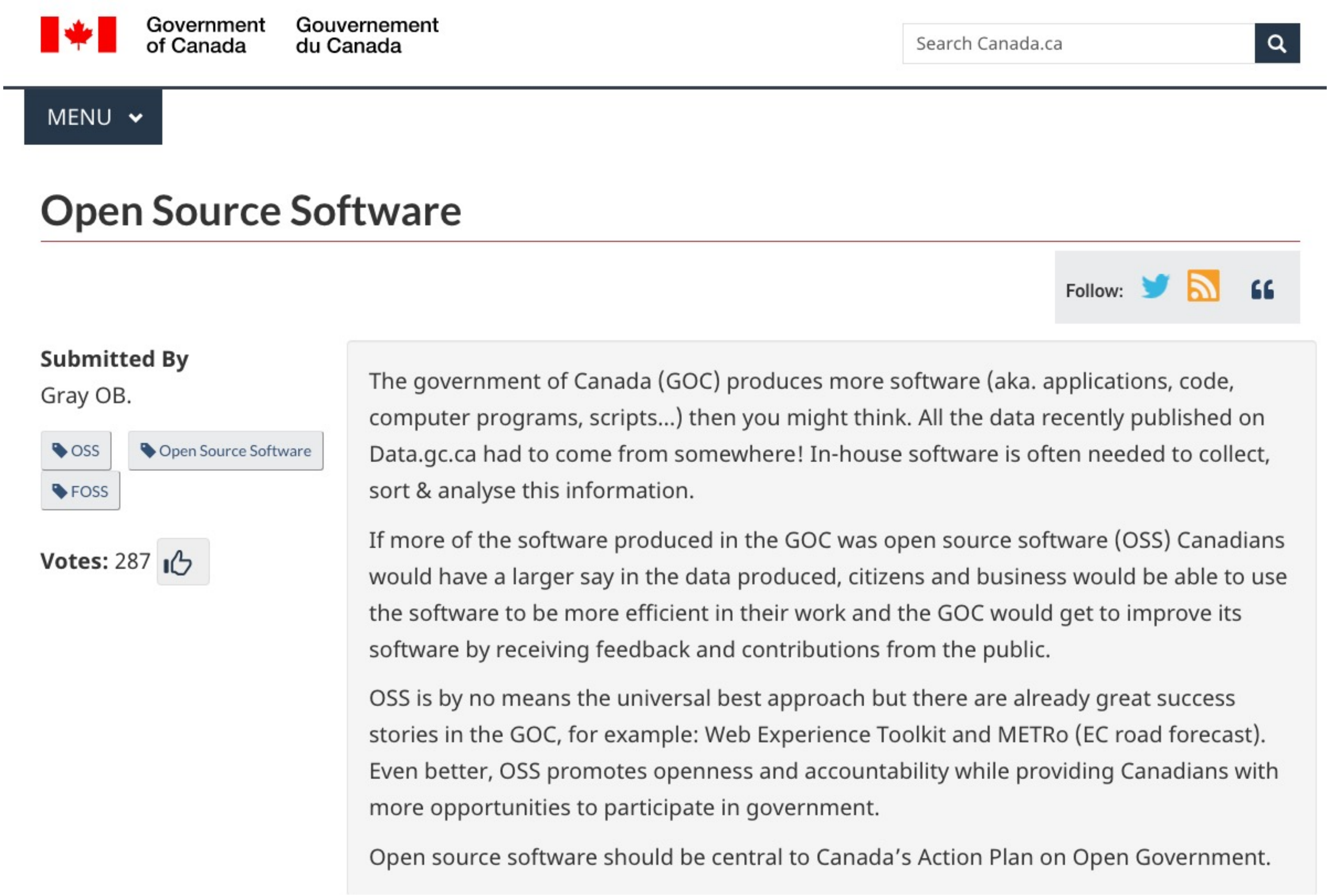
Fri 3 Jul 2020 // 15:30 UTC

34 GOT TIPS?

OpenOffice.org
fedora
blender
mozilla
Firefox
Linux
Java
php
MySQL
debian
WordPress
Apache
ubuntu

Motivation to understand open source.

- Companies work on open source projects.
- Companies use open source projects.
- Companies are based around open source projects.
- Principles percolate throughout industry.
- Political/philosophical debate and being informed is healthy.



Open Source Software

Submitted By
Gray OB.

Votes: 287

The government of Canada (GOC) produces more software (aka. applications, code, computer programs, scripts...) then you might think. All the data recently published on Data.gc.ca had to come from somewhere! In-house software is often needed to collect, sort & analyse this information.

If more of the software produced in the GOC was open source software (OSS) Canadians would have a larger say in the data produced, citizens and business would be able to use the software to be more efficient in their work and the GOC would get to improve its software by receiving feedback and contributions from the public.

OSS is by no means the universal best approach but there are already great success stories in the GOC, for example: Web Experience Toolkit and METRo (EC road forecast). Even better, OSS promotes openness and accountability while providing Canadians with more opportunities to participate in government.

Open source software should be central to Canada’s Action Plan on Open Government.

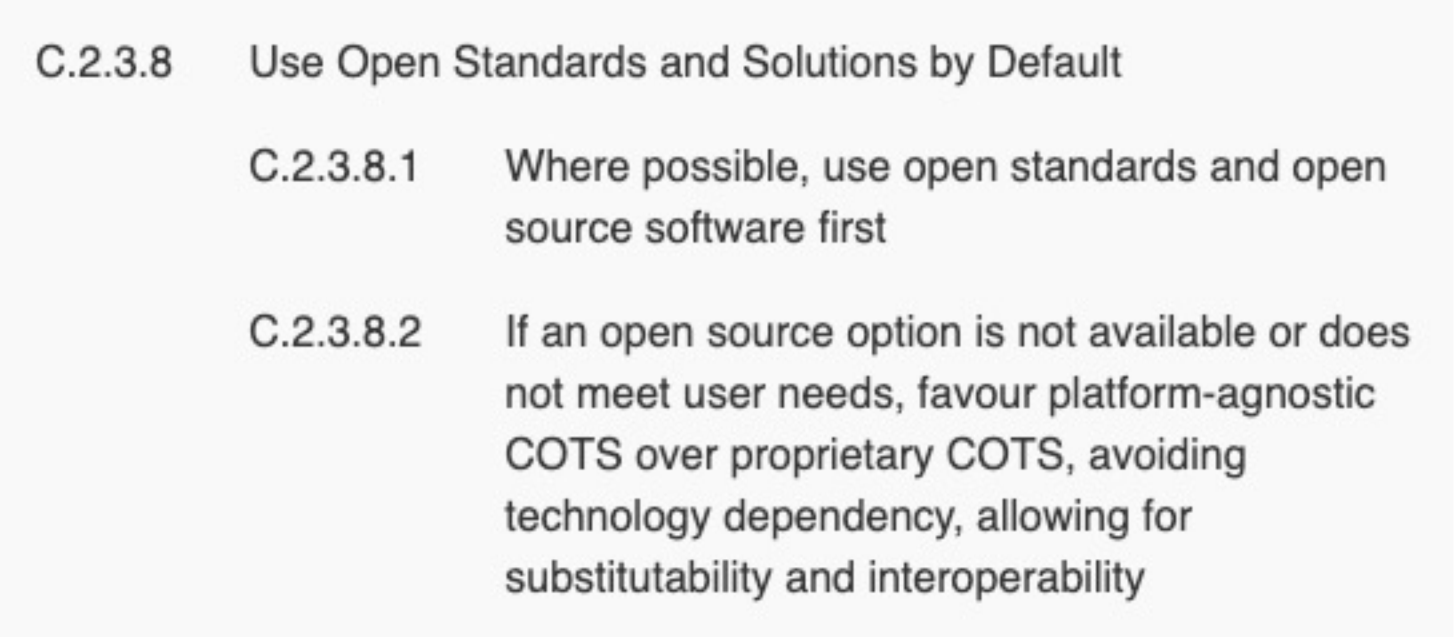
https://open.canada.ca/en/open_source_software

Canada adopts open source mandate for government software

December 10, 2018 By Luke Fretwell

https://govfresh.com/2018/12/canada-adopts-open-source-mandate-for-government-software/

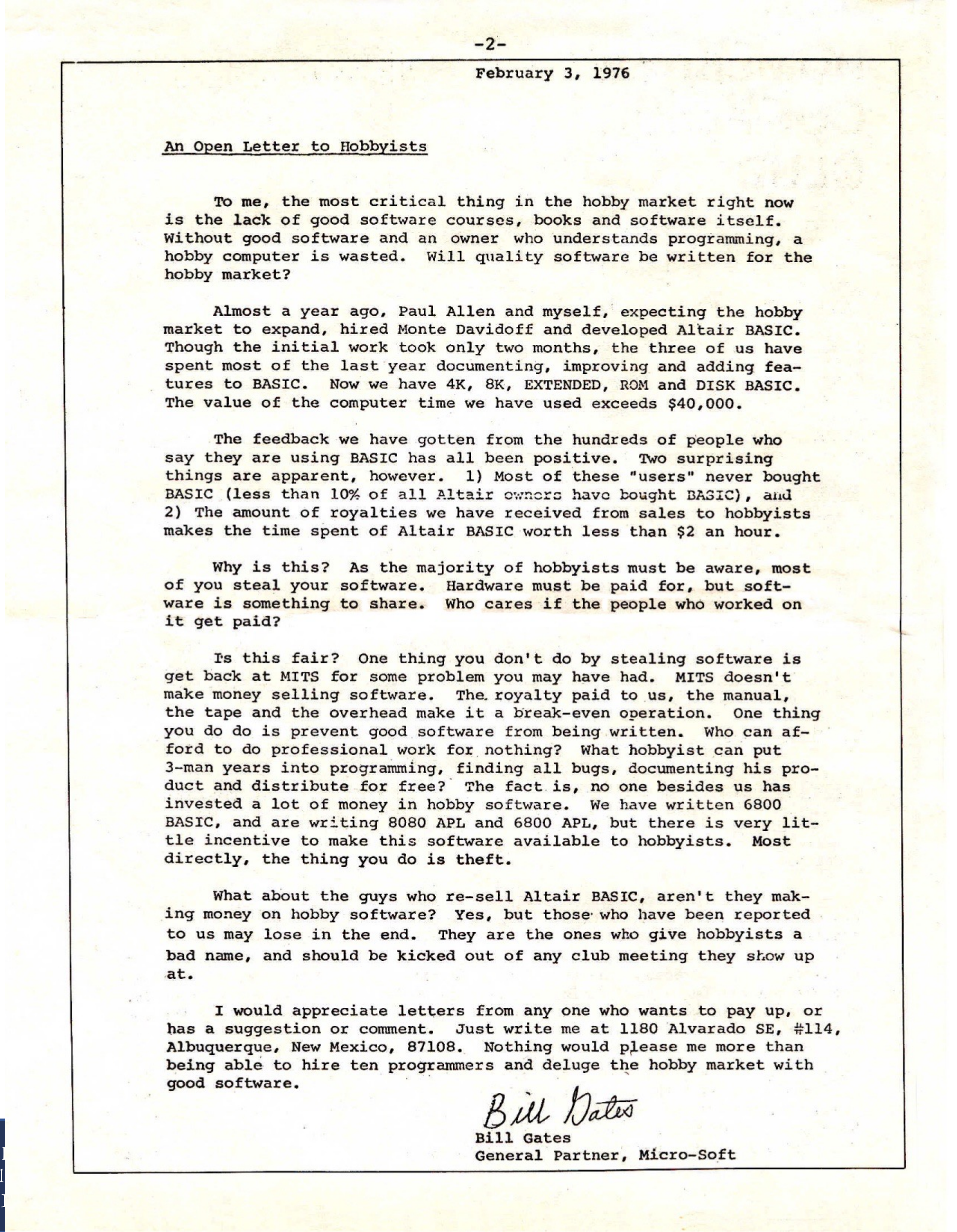
Canada Federal Government publishes a new IT directive that mandates the use of open source software first before considering proprietary software



C.2.3.8 Use Open Standards and Solutions by Default

C.2.3.8.1 Where possible, use open standards and open source software first

C.2.3.8.2 If an open source option is not available or does not meet user needs, favour platform-agnostic COTS over proprietary COTS, avoiding technology dependency, allowing for substitutability and interoperability



-2-

February 3, 1976

An Open Letter to Hobbyists

To me, the most critical thing in the hobby market right now is the lack of good software courses, books and software itself. Without good software and an owner who understands programming, a hobby computer is wasted. Will quality software be written for the hobby market?

Almost a year ago, Paul Allen and myself, expecting the hobby market to expand, hired Monte Davidoff and developed Altair BASIC. Though the initial work took only two months, the three of us have spent most of the last year documenting, improving and adding features to BASIC. Now we have 4K, 8K, EXTENDED, ROM and DISK BASIC. The value of the computer time we have used exceeds $40,000.

The feedback we have gotten from the hundreds of people who say they are using BASIC has all been positive. Two surprising things are apparent, however. 1) Most of these "users" never bought BASIC (less than 10% of all Altair owners have bought BASIC), and 2) The amount of royalties we have received from sales to hobbyists makes the time spent of Altair BASIC worth less than $2 an hour.

Why is this? As the majority of hobbyists must be aware, most of you steal your software. Hardware must be paid for, but software is something to share. Who cares if the people who worked on it get paid?

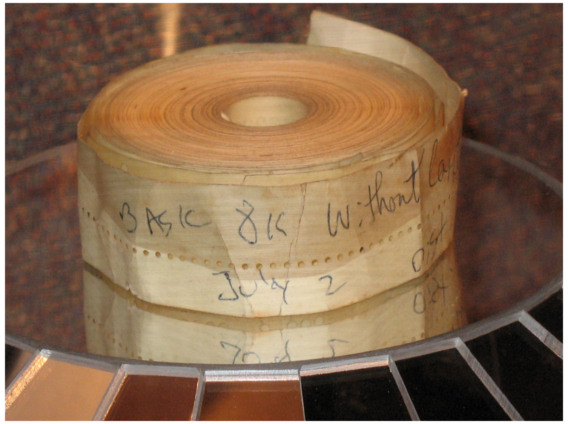
Is this fair? One thing you don't do by stealing software is get back at MITS for some problem you may have had. MITS doesn't make money selling software. The royalty paid to us, the manual, the tape and the overhead make it a break-even operation. One thing you do do is prevent good software from being written. Who can afford to do professional work for nothing? What hobbyist can put 3-man years into programming, finding all bugs, documenting his product and distribute for free? The fact is, no one besides us has invested a lot of money in hobby software. We have written 6800 BASIC, and are writing 8080 APL and 6800 APL, but there is very little incentive to make this software available to hobbyists. Most directly, the thing you do is theft.

What about the guys who re-sell Altair BASIC, aren't they making money on hobby software? Yes, but those who have been reported to us may lose in the end. They are the ones who give hobbyists a bad name, and should be kicked out of any club meeting they show up at.

I would appreciate letters from any one who wants to pay up, or has a suggestion or comment. Just write me at 1180 Alvarado SE, #114, Albuquerque, New Mexico, 87108. Nothing would please me more than being able to hire ten programmers and deluge the hobby market with good software.

Bill Gates

Bill Gates
General Partner, Micro-Soft



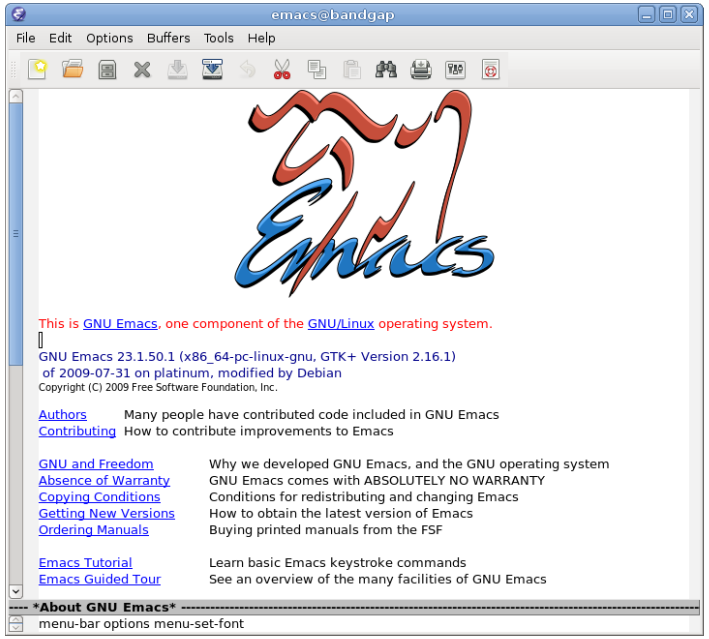
Altair BASIC， 1970

https://pt.wikipedia.org/wiki/Altair_BASIC

"Open source is an intellectual-property destroyer, I can't imagine something that could be worse than this for the software business and the intellectual-property business."

-- former Windows chief Jim Allchin in 2001.

“Free as in free speech.”



This is GNU Emacs, one component of the GNU/Linux operating system.

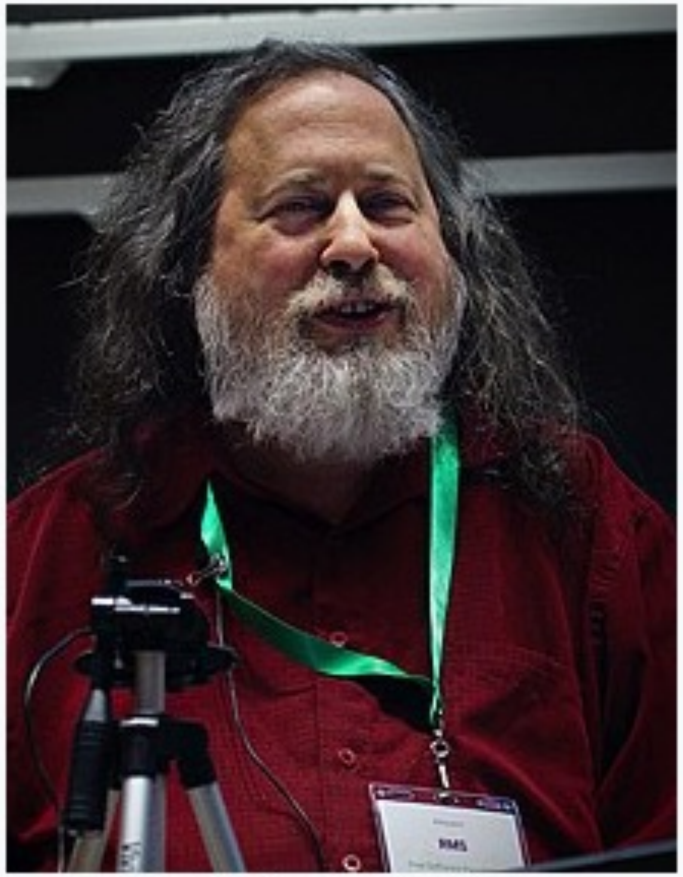
GNU Emacs 23.1.50.1 (x86_64-pc-linux-gnu, GTK+ Version 2.16.1)
of 2009-07-31 on platinum, modified by Debian
Copyright (C) 2009 Free Software Foundation, Inc.

Authors	Many people have contributed code included in GNU Emacs
Contributing	How to contribute improvements to Emacs
GNU and Freedom	Why we developed GNU Emacs, and the GNU operating system
Absence of Warranty	GNU Emacs comes with ABSOLUTELY NO WARRANTY
Copying Conditions	Conditions for redistributing and changing Emacs
Getting New Versions	How to obtain the latest version of Emacs
Ordering Manuals	Buying printed manuals from the FSF
Emacs Tutorial	Learn basic Emacs keystroke commands
Emacs Guided Tour	See an overview of the many facilities of GNU Emacs

```
---- *About GNU Emacs* ----
menu-bar options menu-set-font
```

GNU/Linux

Richard Stallman



Stallman in 2019

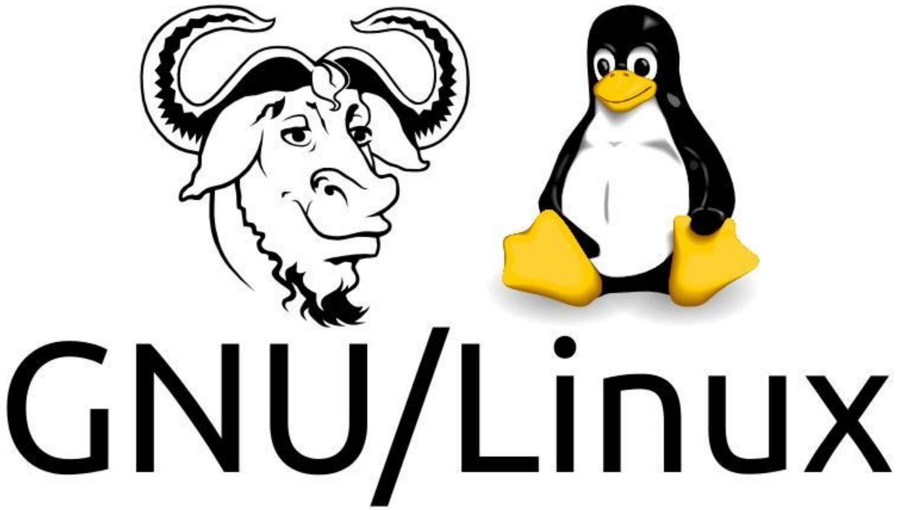
Stallman launched the GNU Project, founded the Free Software Foundation, developed the GNU Compiler Collection and GNU Emacs, and wrote the GNU General Public License.

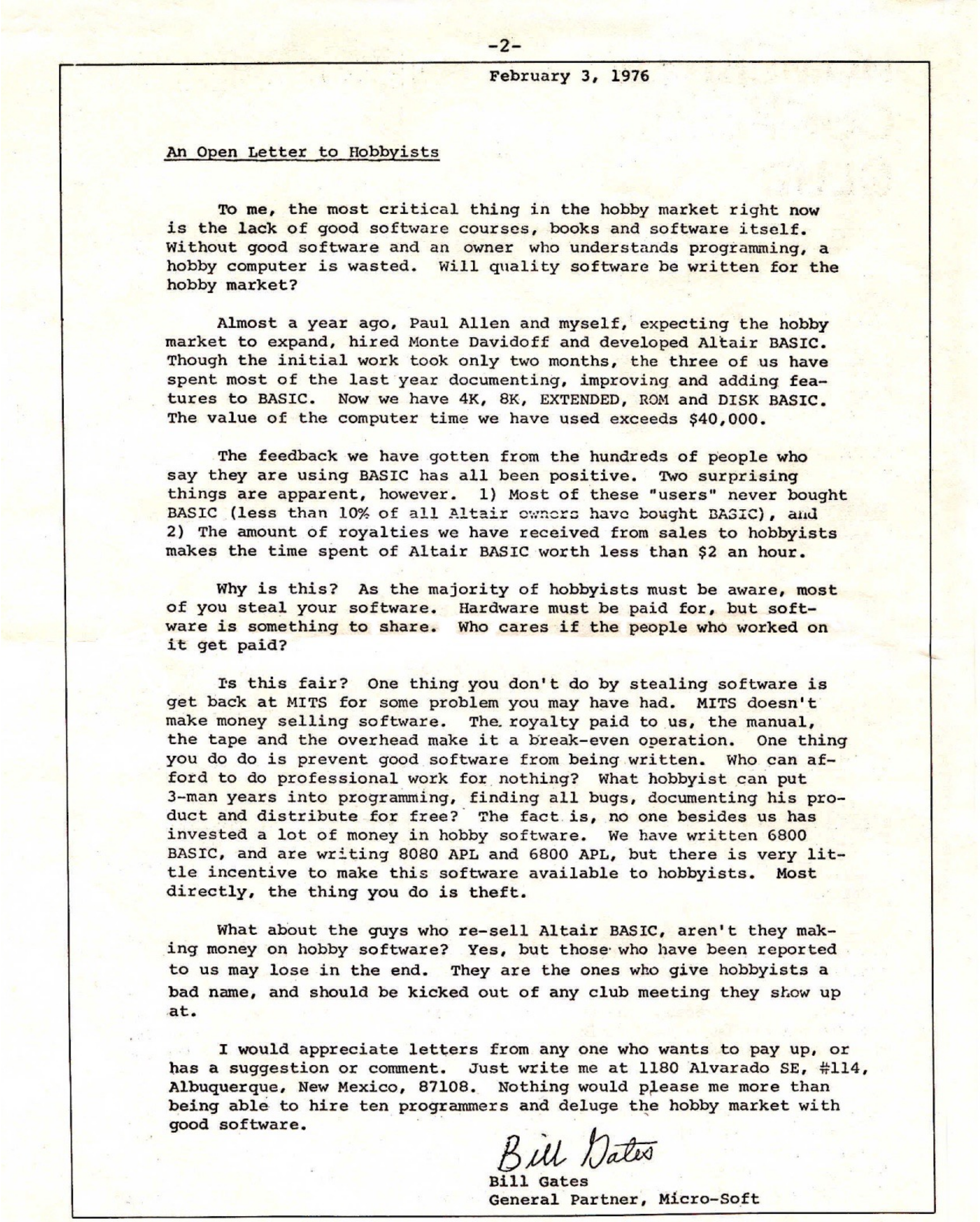
https://www.youtube.com/watch?v=Ag1AKIl_2GM

ON THIS DAY IN 1983
PROJECT GNU
WAS ANNOUNCED BY
RICHARD STALLMAN
WATCH HOW HE LED THE
FREE SOFTWARE
MOVEMENT

https://www.youtube.com/watch?v=djHxEcGLqN8

Stallman vs. Gates





-2-

February 3, 1976

An Open Letter to Hobbyists

To me, the most critical thing in the hobby market right now is the lack of good software courses, books and software itself. Without good software and an owner who understands programming, a hobby computer is wasted. Will quality software be written for the hobby market?

Almost a year ago, Paul Allen and myself, expecting the hobby market to expand, hired Monte Davidoff and developed Altair BASIC. Though the initial work took only two months, the three of us have spent most of the last year documenting, improving and adding features to BASIC. Now we have 4K, 8K, EXTENDED, ROM and DISK BASIC. The value of the computer time we have used exceeds $40,000.

The feedback we have gotten from the hundreds of people who say they are using BASIC has all been positive. Two surprising things are apparent, however. 1) Most of these "users" never bought BASIC (less than 10% of all Altair owners have bought BASIC), and 2) The amount of royalties we have received from sales to hobbyists makes the time spent of Altair BASIC worth less than $2 an hour.

Why is this? As the majority of hobbyists must be aware, most of you steal your software. Hardware must be paid for, but software is something to share. Who cares if the people who worked on it get paid?

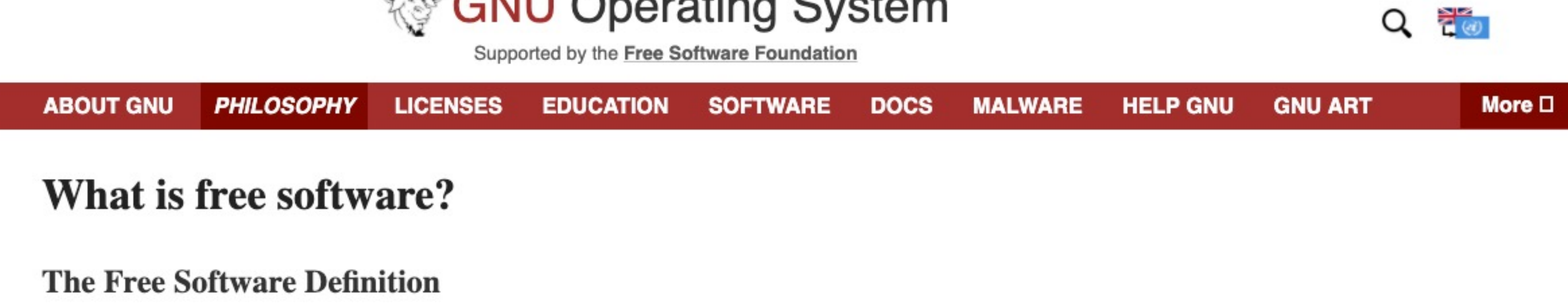
Is this fair? One thing you don't do by stealing software is get back at MITS for some problem you may have had. MITS doesn't make money selling software. The royalty paid to us, the manual, the tape and the overhead make it a break-even operation. One thing you do do is prevent good software from being written. Who can afford to do professional work for nothing? What hobbyist can put 3-man years into programming, finding all bugs, documenting his product and distribute for free? The fact is, no one besides us has invested a lot of money in hobby software. We have written 6800 BASIC, and are writing 8080 APL and 6800 APL, but there is very little incentive to make this software available to hobbyists. Most directly, the thing you do is theft.

What about the guys who re-sell Altair BASIC, aren't they making money on hobby software? Yes, but those who have been reported to us may lose in the end. They are the ones who give hobbyists a bad name, and should be kicked out of any club meeting they show up at.

I would appreciate letters from any one who wants to pay up, or has a suggestion or comment. Just write me at 1180 Alvarado SE, #114, Albuquerque, New Mexico, 87108. Nothing would please me more than being able to hire ten programmers and deluge the hobby market with good software.

Bill Gates

Bill Gates
General Partner, Micro-Soft



What is free software?

The Free Software Definition

freedom 0 : The freedom to run the program as you wish, for any purpose

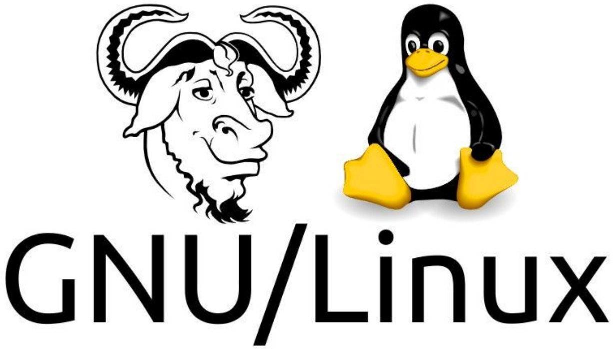
freedom 1: The freedom to study how the program works, and change it so it does your computing as you wish

freedom 2: The freedom to redistribute copies so you can help others

freedom 3: The freedom to distribute copies of your modified versions to others

Free Software vs Open Source

- Free software origins (70-80s ~Stallman)
  - Political goal
  - Software part of free speech
    - free exchange, free modification
    - proprietary software is unethical
    - security, trust
  - GNU project, Linux, GPL license
- Open source (1998 ~ O'Reilly)
  - Rebranding without political legacy
  - Emphasis on internet and large dev./user involvement
  - Openness toward proprietary software/coexist
  - (Think: Netscape becoming Mozilla)



Licenses

- The MIT (Massachusetts Institute of Technology) License: This is a permissive license that places limited restrictions on software reuse.
- The GNU General Public License v2: This copyleft license gives users the freedom to run, study, and make improvements to the software.
- The Apache License v2: This is a permissive license that mandates preservation of the copyright notice and disclaimer.
- The BSD Licenses: They are a set of non-copyleft licenses that gives minimal restrictions on the use and redistribution of the software.

Why learn about licenses?

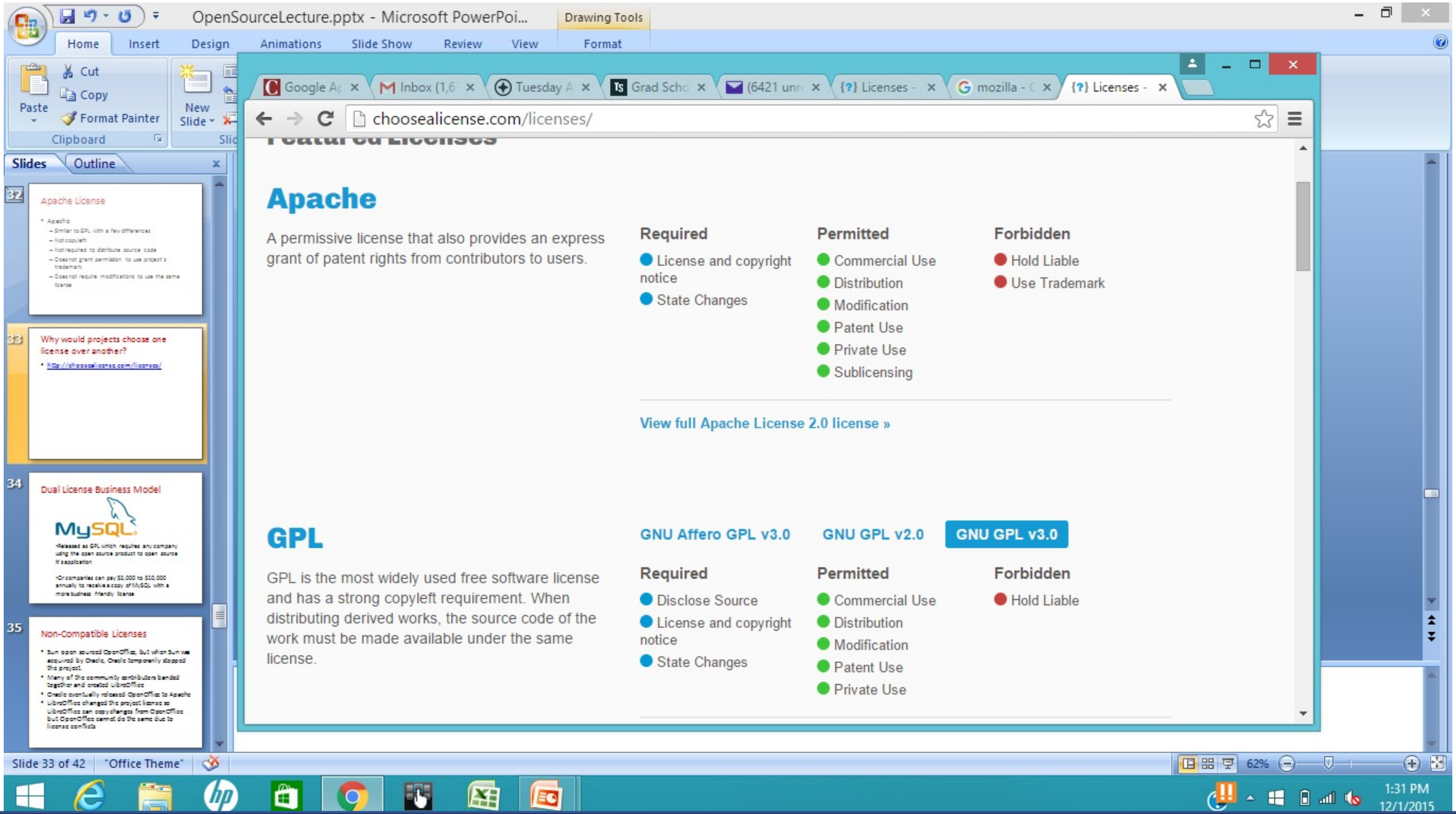
- Companies will avoid certain licenses – commonly the copyleft licenses
- Specific licenses may provide competitive advantages
- You may eventually want to release open source software or become more involved in an open source project

Open Source Licenses

Software	Percentage
MIT License	24%
GNU General Public License (GPL) 2.0	23%
Apache License 2.0	16%
GNU General Public License (GPL) 3.0	9%
BSD License 2.0 (3-clause, New or Revised) License	6%
GNU Lessor General Public License (LGPL) 2.1	5%
Artistic License (Perl)	4%
GNU Lesser General Public License (LGPL) 3.0	2%
Microsoft Public License	2%
Eclipse Public License	2%

List from: https://www.blackducksoftware.com/resources/data/top-20-open-source-licenses

Why would projects choose one license over another?



32 Apache License

- Apache
  - Similar to GPL with a few differences
  - Not copyleft
  - Not required to distribute source code
  - Does not grant permission to use project's trademark
  - Does not require modifications to use the same license

33 Why would projects choose one license over another?

- http://choosealicense.com/licenses/

34 Dual License Business Model

•Released as GPL which requires any company using the open source product to open source it's application

•Or companies can pay $2,000 to $10,000 annually to receive a copy of MySQL with a more business friendly license

35 Non-Compatible Licenses

- Sun open sourced OpenOffice, but when Sun was acquired by Oracle, Oracle temporarily stopped the project.
- Many of the community contributors banded together and created LibreOffice
- Oracle eventually released OpenOffice to Apache
- LibreOffice changed the project license so LibreOffice can copy changes from OpenOffice but OpenOffice cannot do the same due to license conflicts

choosealicense.com/licenses/

Apache

A permissive license that also provides an express grant of patent rights from contributors to users.

Required	Permitted	Forbidden
● License and copyright notice	● Commercial Use	● Hold Liable
● State Changes	● Distribution	● Use Trademark
	● Modification	
	● Patent Use	
	● Private Use	
	● Sublicensing	

View full Apache License 2.0 license »

GPL

GNU Affero GPL v3.0 GNU GPL v2.0 GNU GPL v3.0

GPL is the most widely used free software license and has a strong copyleft requirement. When distributing derived works, the source code of the work must be made available under the same license.

Required	Permitted	Forbidden
● Disclose Source	● Commercial Use	● Hold Liable
● License and copyright notice	● Distribution	
● State Changes	● Modification	
	● Patent Use	
	● Private Use	

Dual License Business Model

- Released as GPL which requires a company using the open source product to open source it’s application
- Or companies can pay $2,000 to $10,000 annually to receive a copy of MySQL with a more business friendly license

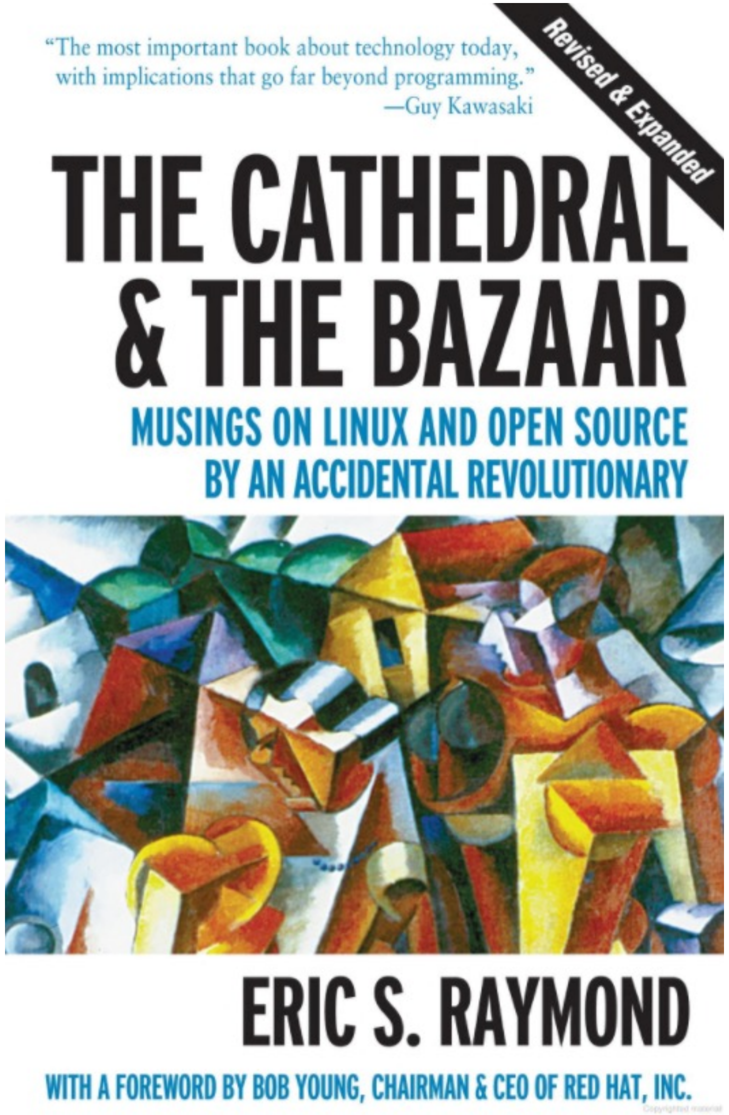
Risk: Incompatible Licenses

- Sun open sourced OpenOffice, but when Sun was acquired by Oracle, Oracle temporarily stopped the project.
- Many of the community contributors banded together and created LibreOffice
- Oracle eventually released OpenOffice to Apache
- LibreOffice changed the project license so LibreOffice can copy changes from OpenOffice but OpenOffice cannot do the same due to license conflicts

The Cathedral and the Bazaar

- Centralized vs. decentralized
- Planned vs. unplanned

- Fetchmail


"The most important book about technology today, with implications that go far beyond programming."
—Guy Kawasaki
Revised & Expanded
THE CATHEDRAL & THE BAZAAR
MUSINGS ON LINUX AND OPEN SOURCE BY AN ACCIDENTAL REVOLUTIONARY
ERIC S. RAYMOND
WITH A FOREWORD BY BOB YOUNG, CHAIRMAN & CEO OF RED HAT, INC.

Roles

- Leader

  Develops initial system

  Does what nobody else does

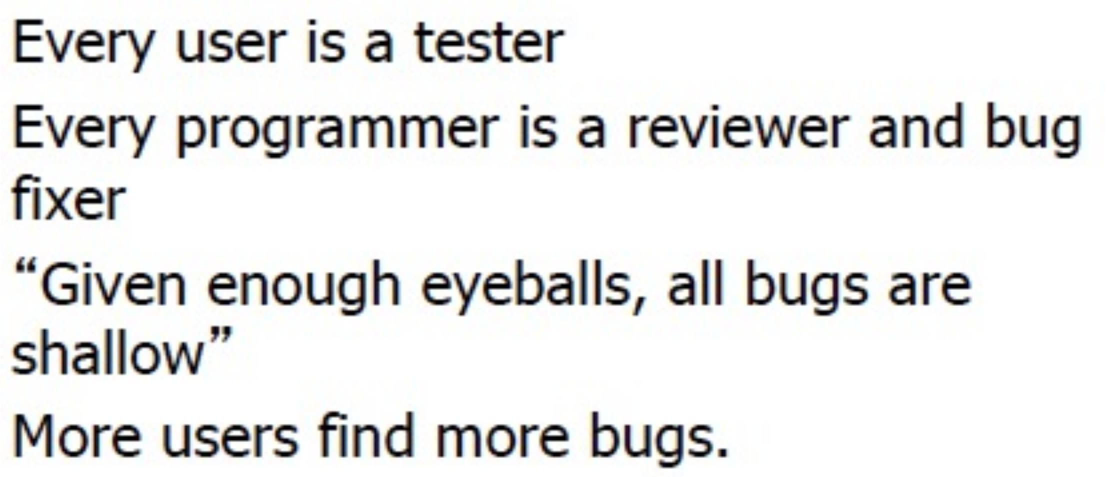
  Makes final decisions

- User/programmer
  - Dose most of the work

Requirements

- Who decides what features get added?
    - Programmers
        - who want to use the feature (scratch an itch)
        - who are persuaded to add it
- Must be a way to distribute changes for a feature (Version Control)
- Must be way to talk about desired features (mailing lists, forums)

Testing



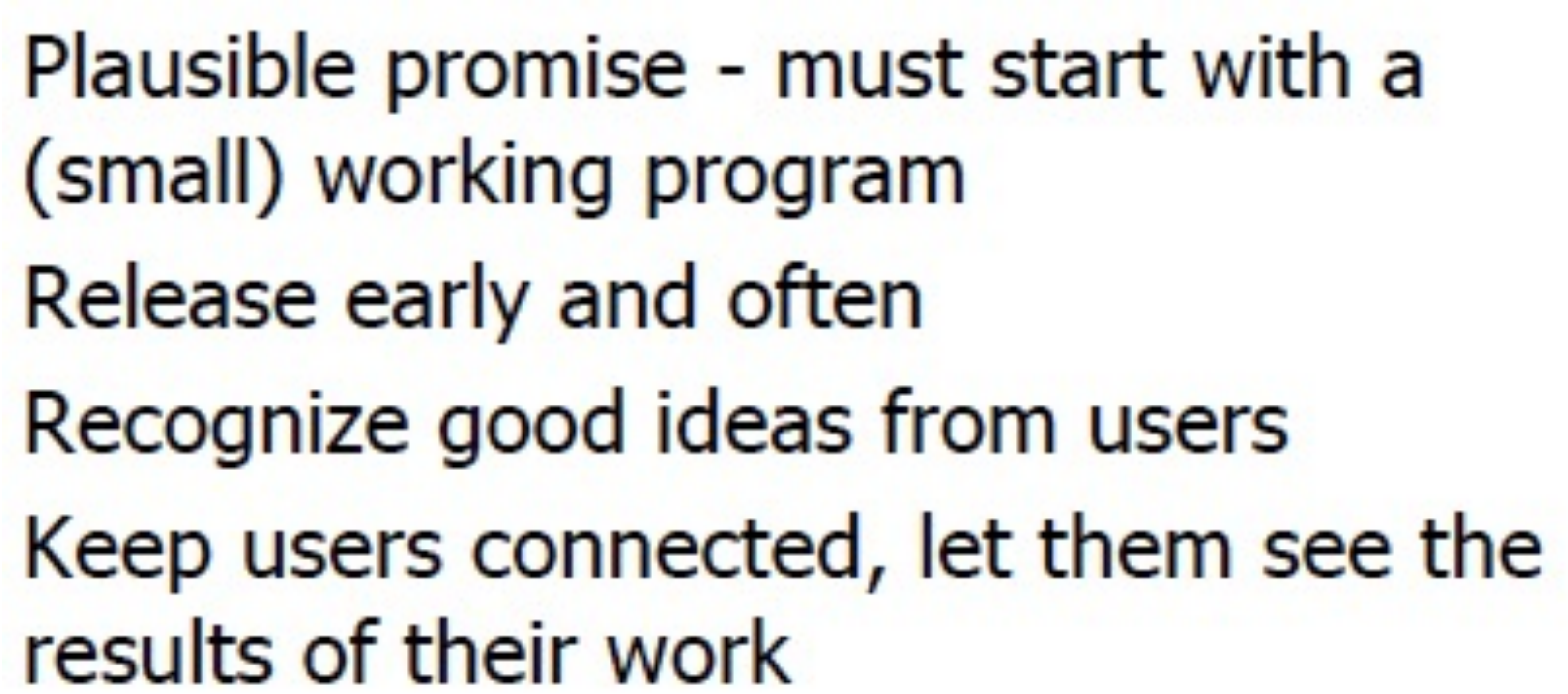
Every user is a tester

Every programmer is a reviewer and bug fixer

“Given enough eyeballs, all bugs are shallow”

More users find more bugs.

Lifecycle



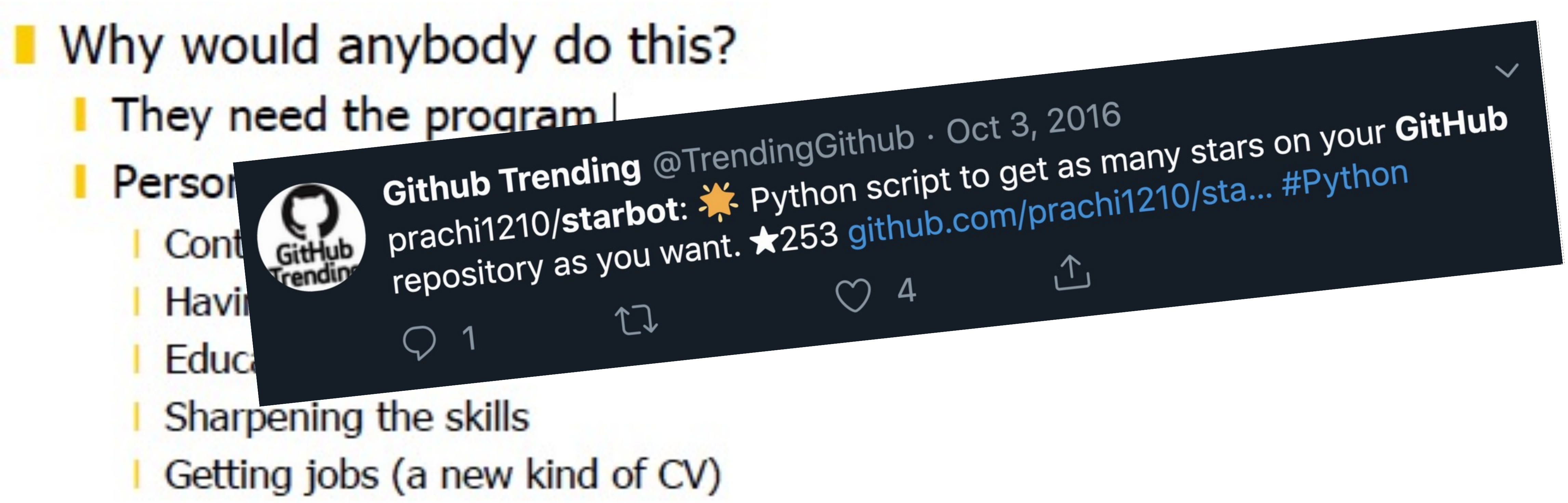
Plausible promise - must start with a (small) working program

Release early and often

Recognize good ideas from users

Keep users connected, let them see the results of their work

Rewards



- Why would anybody do this?
  - They need the program
  - Perso
    - Cont
    - Havi
    - Educ
    - Sharpening the skills
    - Getting jobs (a new kind of CV)

Github Trending @TrendingGithub · Oct 3, 2016

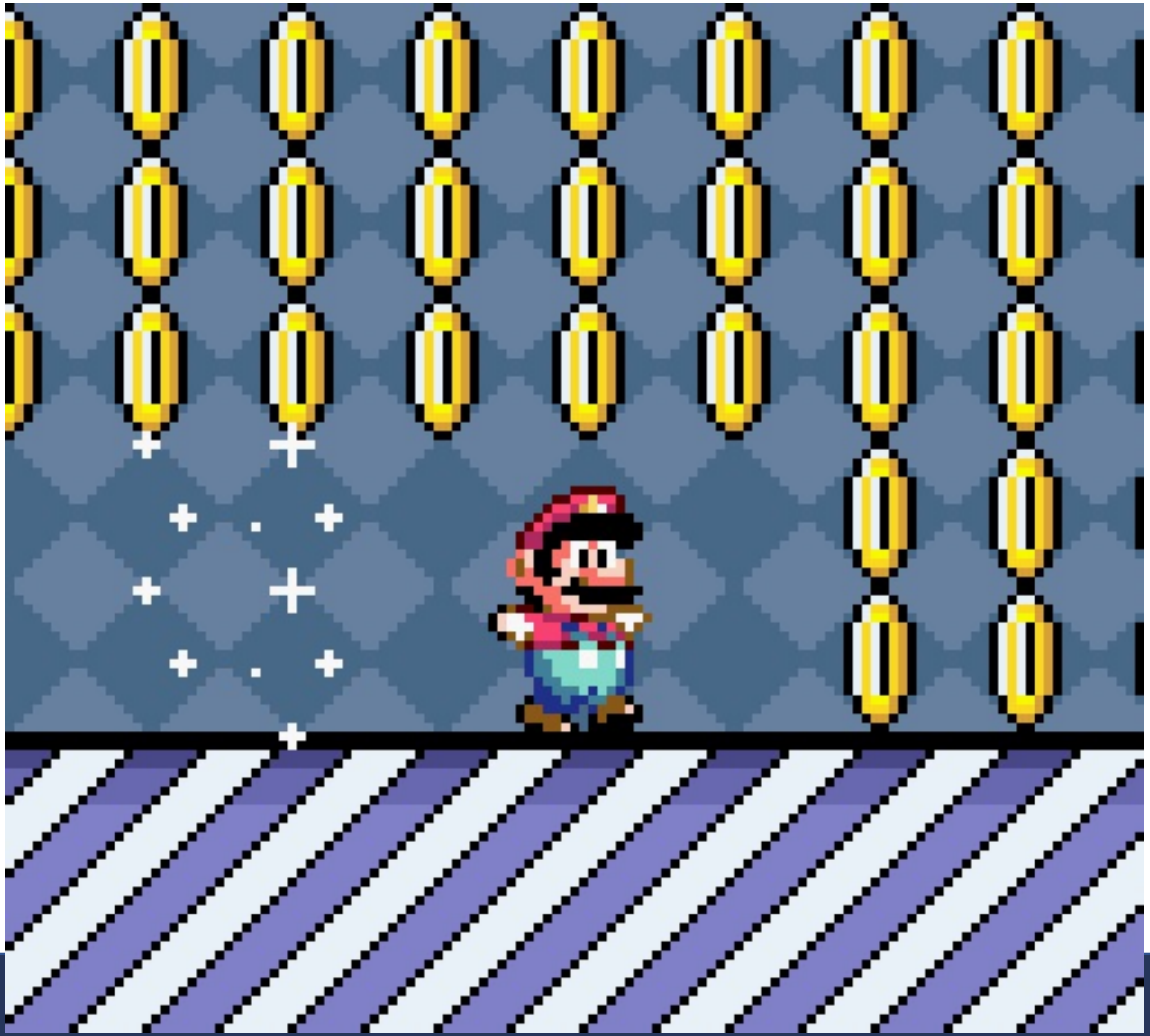
prachi1210/**starbot**: 🌟 Python script to get as many stars on your **GitHub** repository as you want. ★253 github.com/prachi1210/sta... #Python

Costs

- Need a leader
  - a lot of work over a long time
  - must communicate
  - an organizer as much as a designer

Open Source in a proprietary context (benefits vs. Risk)

How do open source companies make money?



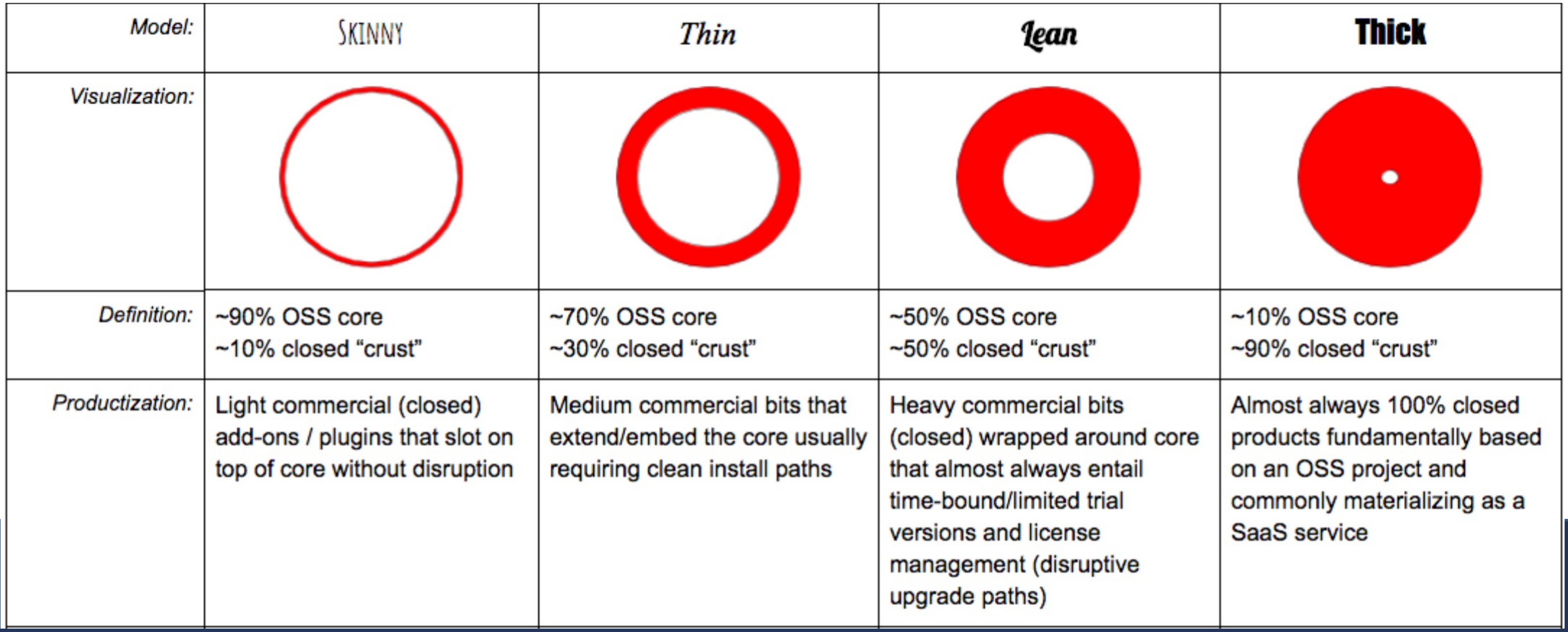
Open Source Business Models

- Support
- Hosting
- Open-core
- Restrictive Licensing
- Hybrid Licensing

Red Hat

databricks

Acquia Cloud



Model:	SKINNY	Thin	Lean	Thick
Visualization:				
Definition:	~90% OSS core ~10% closed "crust"	~70% OSS core ~30% closed "crust"	~50% OSS core ~50% closed "crust"	~10% OSS core ~90% closed "crust"
Productization:	Light commercial (closed) add-ons / plugins that slot on top of core without disruption	Medium commercial bits that extend/embed the core usually requiring clean install paths	Heavy commercial bits (closed) wrapped around core that almost always entail time-bound/limited trial versions and license management (disruptive upgrade paths)	Almost always 100% closed products fundamentally based on an OSS project and commonly materializing as a SaaS service

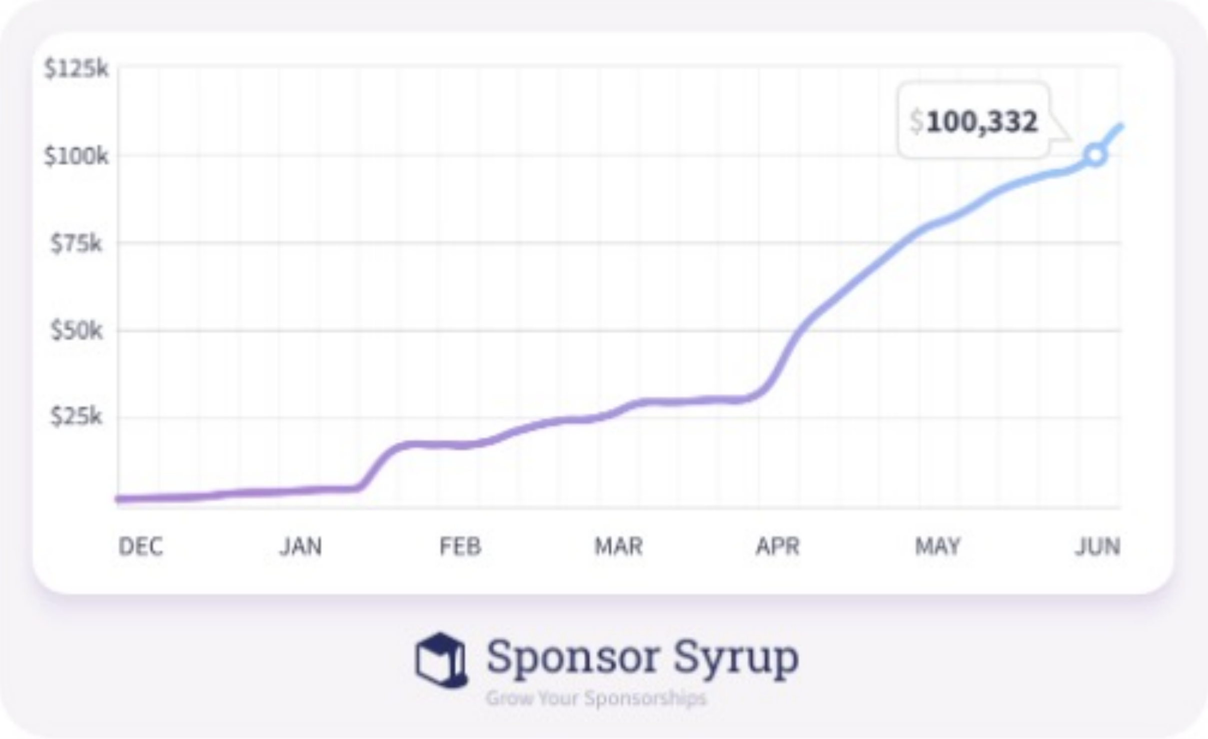
Other Open Source Business Models

- Companies dedicate resources to projects which help them and the community
    - Apache receives donations
- Selling merchandise – Canonical (Ubuntu)
- Selling advertising or customer traffic – Mozilla

- **But: Sustainability is a risk/problem!**

I Just Hit $100k/yr On GitHub Sponsors! 🎉❤️ (How I Did It)

Jun 2020



Caleb Porzio
https://hubs.ly/H0rR4_X0

Quality?!

“There are no technical requirements for the plugins aside from them being able to be installed on a fresh Eclipse platform. We leave it to the community to find and report bugs related to technical features and conflicts.”

--Eclipse Marketplace, Dec 2014

Open Source Famous Phrases

Linus’s Law - Many eyes make all bugs shallow

Collaboration over Competition

…is open source code of higher quality?

- How would we be able to tell?

A Case Study of Open Source Software Development: The Apache Server

Measure	Apache	Proprietary System A	Proprietary System C	Proprietary System D
Post-release defects/KLOCA	2.64	0.11	0.1	0.7
Post-release defects/KDelta	40.8	4.3	14	2.8
Post-feature test Defects/KLOCA	2.64	*	5.7	6.0
Post-feature test Defects/KLOCA	40.8	*	164	196

Open SSL/Heartbleed.

- In 2013, OpenSSL made $2,000 in donations (and some from other sources)
- One full time programmer
- Heartbleed (2014): Vulnerability was found that effected about 17.5% of web servers (half a million)
- Used by Yahoo, Twitter, Google
- Who is responsible?

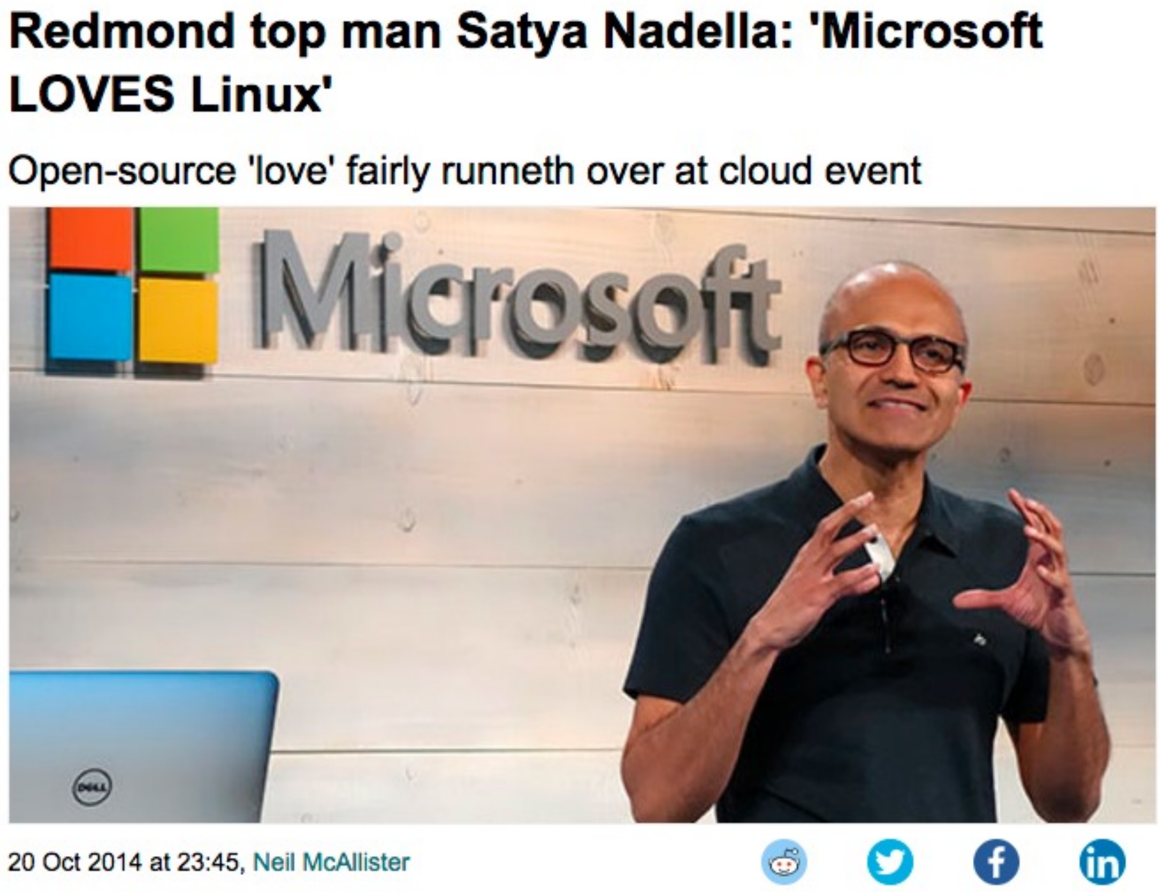
Case Study: OpenSSL

- When HeartBleed occurred, Google reported the bug and later submitted a patch
- After the HeartBleed bug, more than 17 companies agreed to each contribute $100,000 annually for 3 year to the Core Infrastructure Initiative.
- Core Infrastructure Initiative distributes funds to needy but important projects

Bug Bounties

- Facebook, Google, Yahoo, Microsoft, and other companies have rewards for finding bugs and reporting them
- Usually $100 or more for simple bugs and higher rewards for more serious bugs
- Bounties can save the company from malicious exploits, which can cost the company much more.
  - Ponemon Institute reports average cost of $3.79 million per company data breech (2014)

Hilarious irony



Redmond top man Satya Nadella: 'Microsoft LOVES Linux'

Open-source 'love' fairly runneth over at cloud event

20 Oct 2014 at 23:45, Neil McAllister

Summary

- Human beings take pleasure in a task when it falls in an optimal-challenge zone; not so easy as to be boring, not too hard to achieve.
- A happy programmer is one who is neither underutilized nor weighed down with ill-formulated goals and stressful process friction.
- Enjoyment predicts efficiency.

Open Source Reality

- Aggressive collaborative tool use
    - version control, CI, issue tracker, reviews, …
- Careful management of people
- Process rigor
- Often aimed at expert users

similar to industrial practices

- Intellectual property
- Often industry supported
- Often addressing common assets

Traditional Collaboration Model

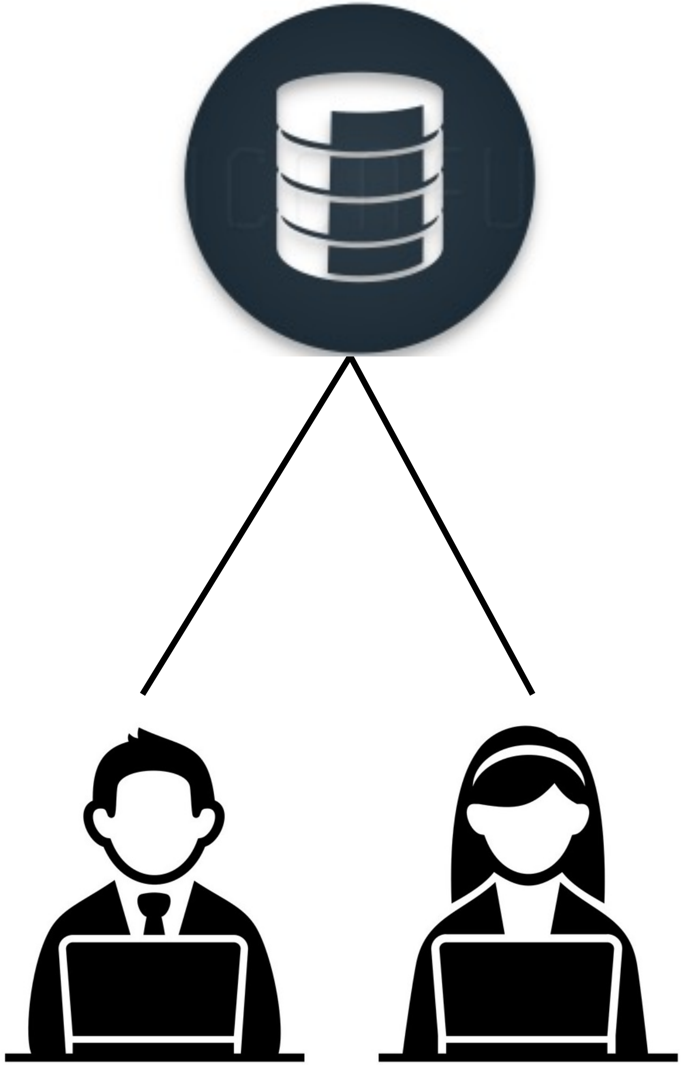
90'S

open source

Traditional Collaboration Model

90's

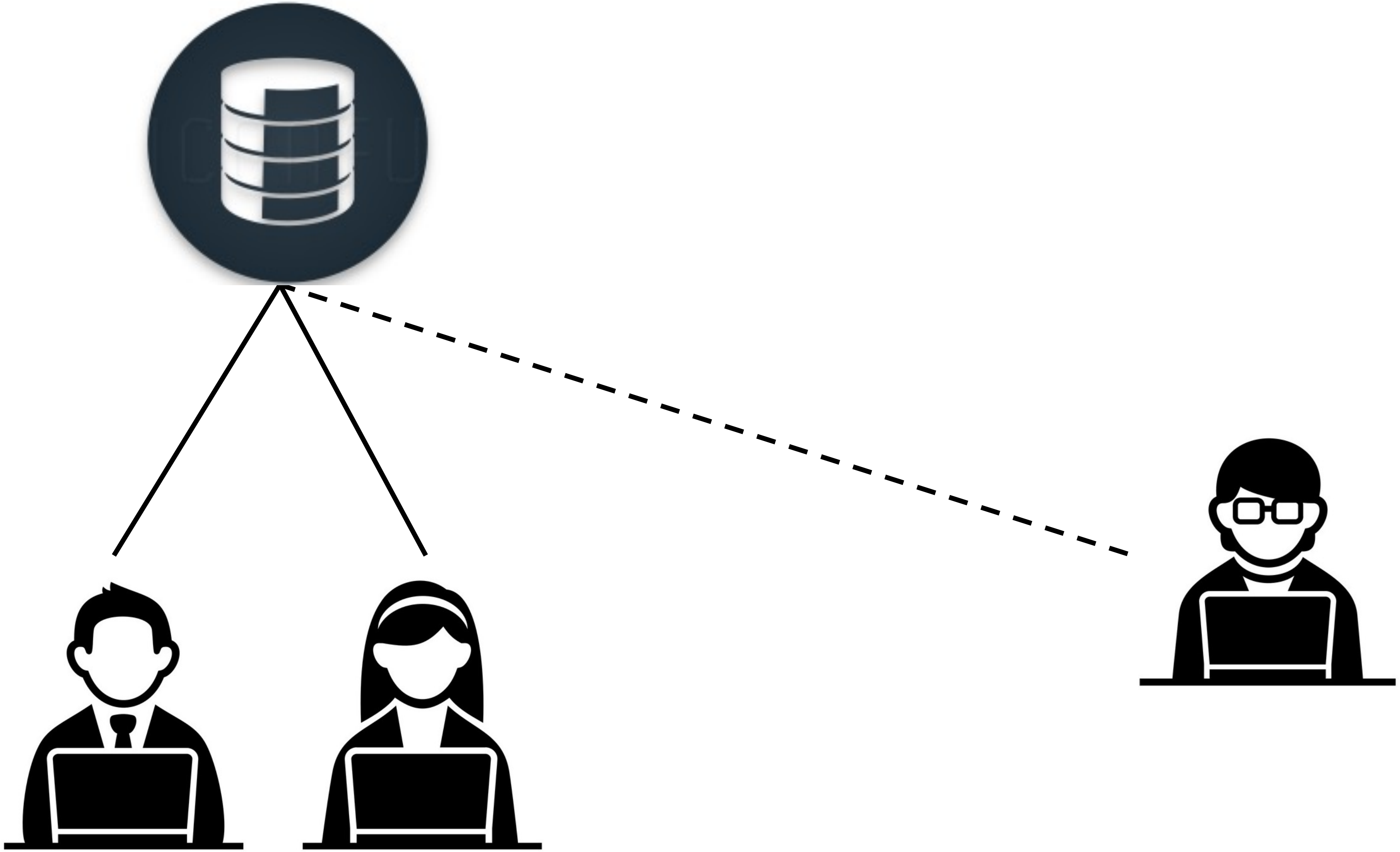
open source



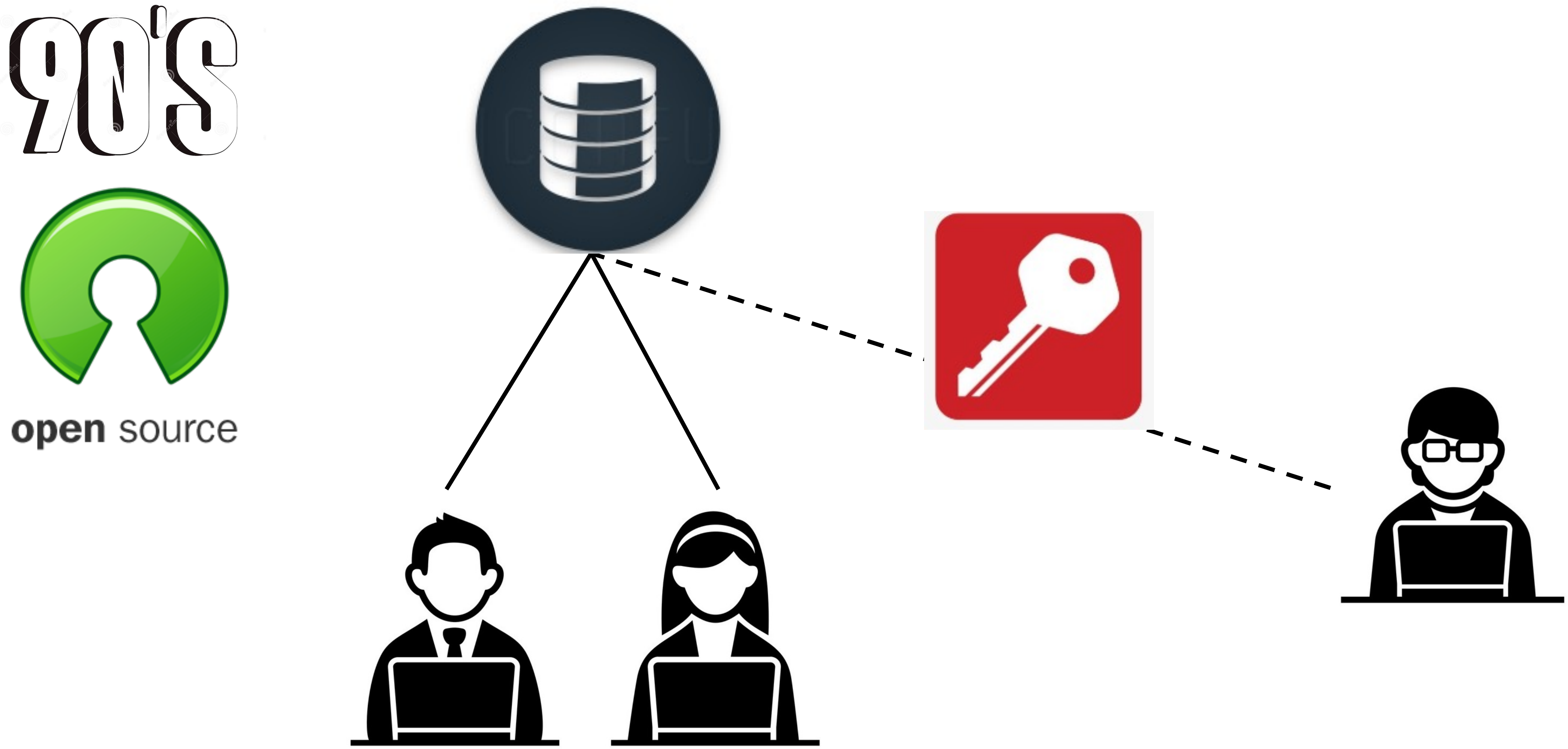
Traditional Collaboration Model

90's

open source



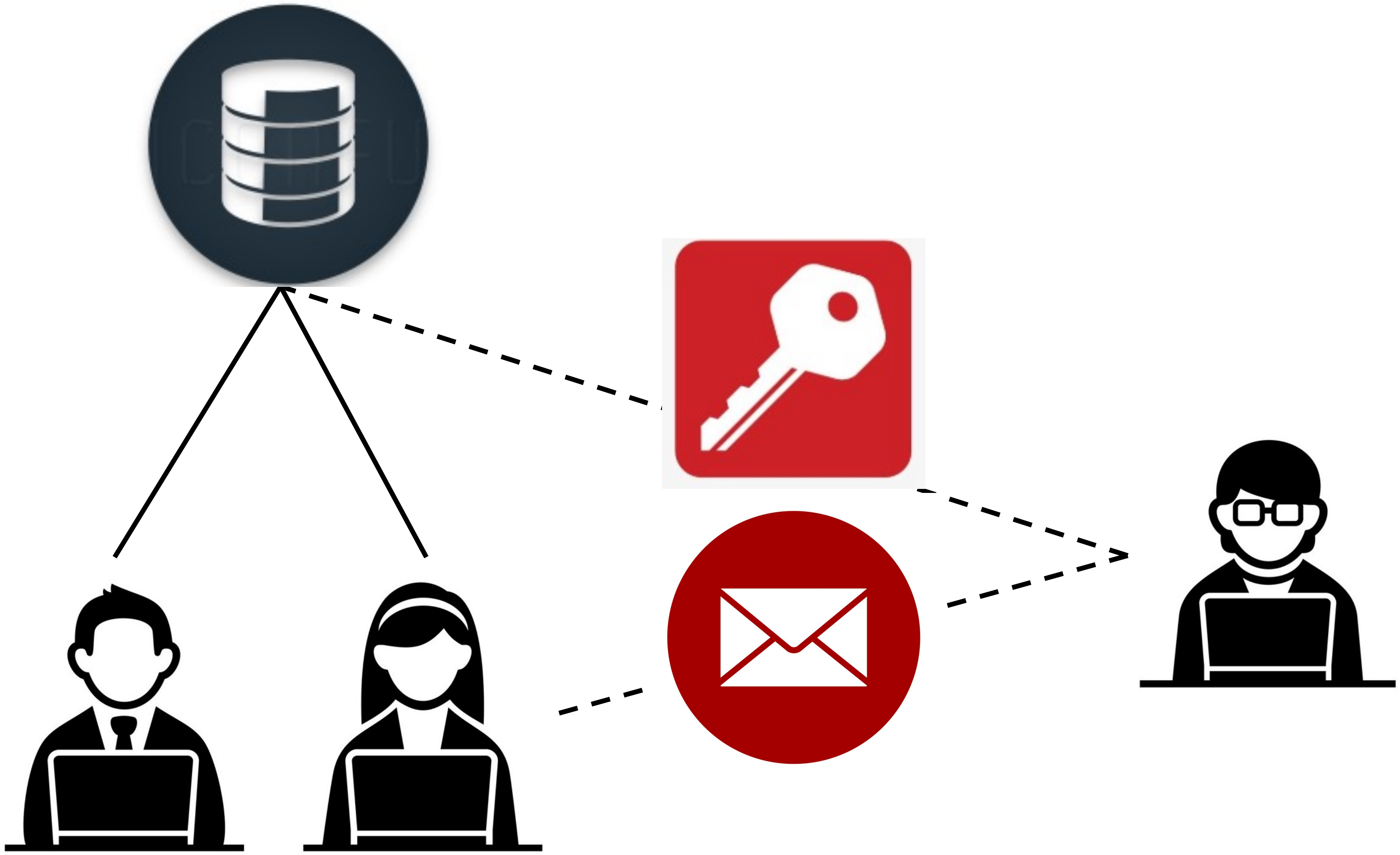
Traditional Collaboration Model



Traditional Collaboration Model

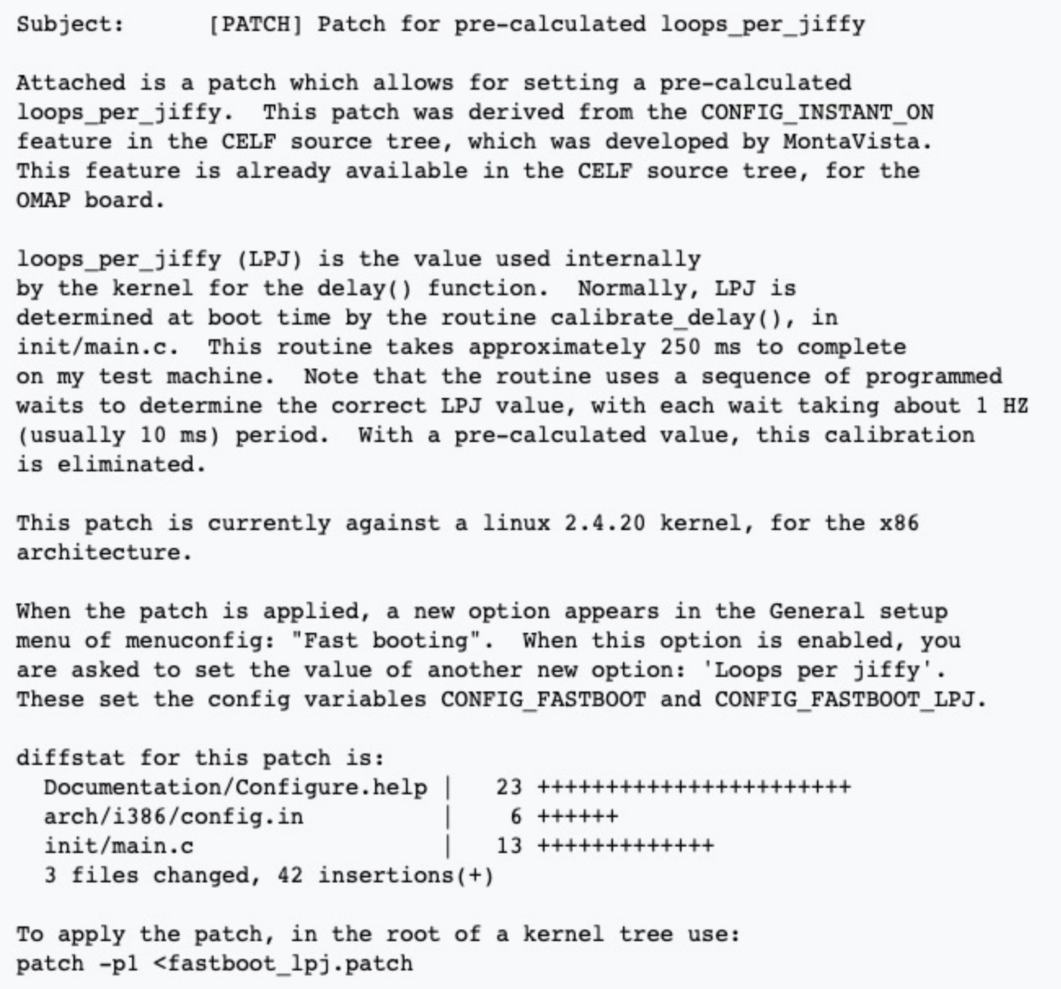
90's

open source



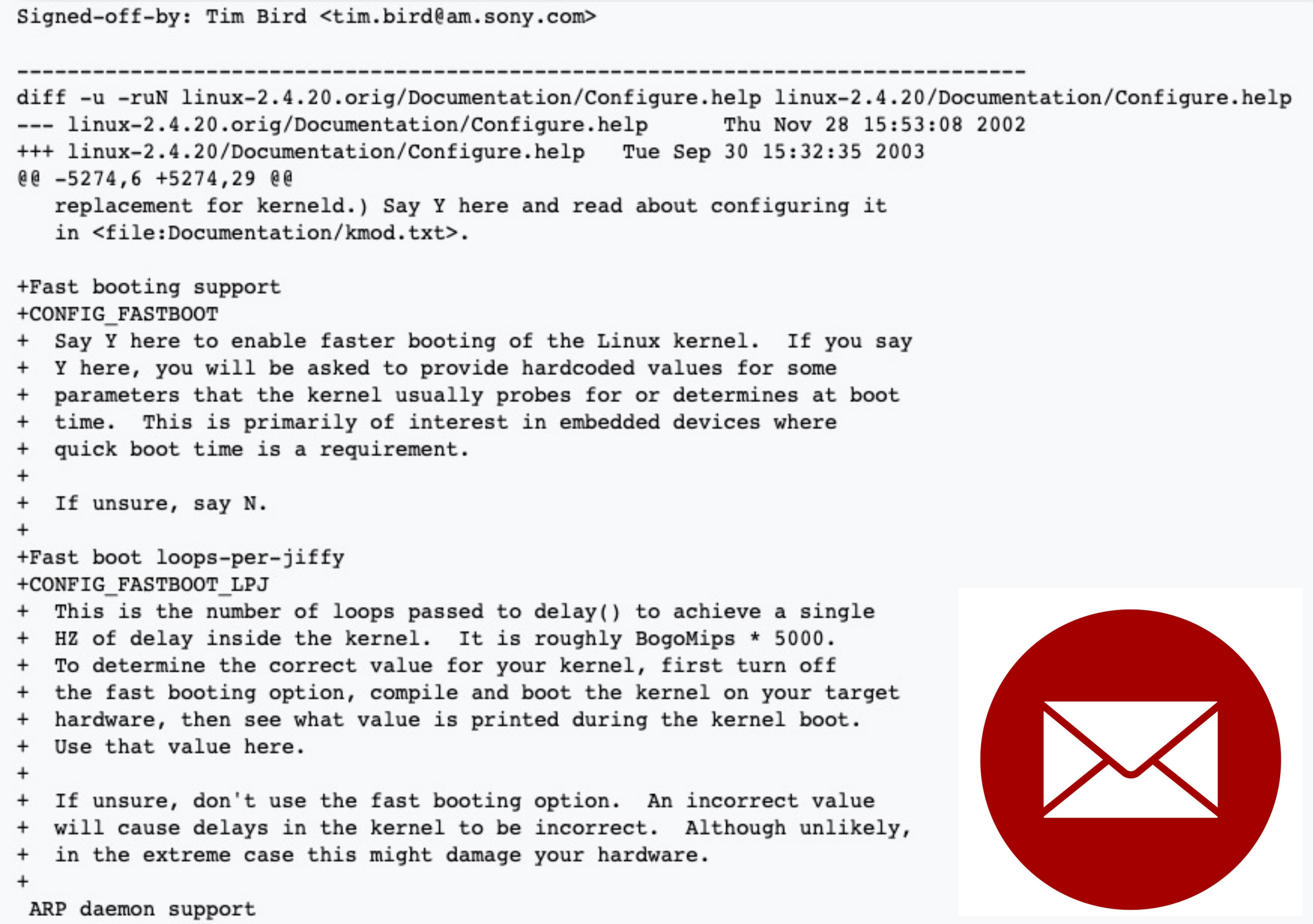
Traditional Collaboration Model

Description



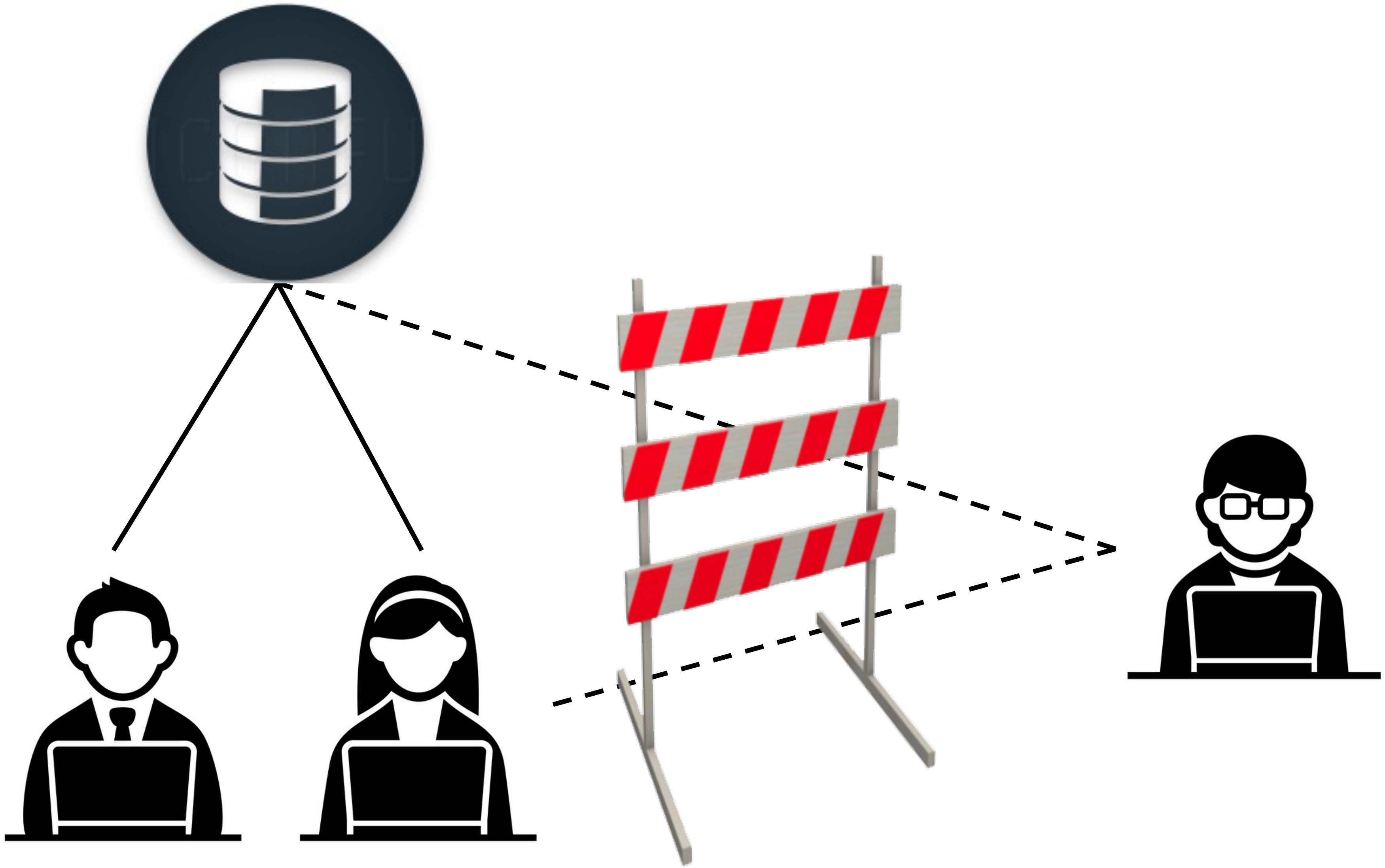
```
Subject:      [PATCH] Patch for pre-calculated loops_per_jiffy

Attached is a patch which allows for setting a pre-calculated
loops_per_jiffy.  This patch was derived from the CONFIG_INSTANT_ON
feature in the CELF source tree, which was developed by MontaVista.
This feature is already available in the CELF source tree, for the
OMAP board.

loops_per_jiffy (LPJ) is the value used internally
by the kernel for the delay() function.  Normally, LPJ is
determined at boot time by the routine calibrate_delay(), in
init/main.c.  This routine takes approximately 250 ms to complete
on my test machine.  Note that the routine uses a sequence of programmed
waits to determine the correct LPJ value, with each wait taking about 1 HZ
(usually 10 ms) period.  With a pre-calculated value, this calibration
is eliminated.

This patch is currently against a linux 2.4.20 kernel, for the x86
architecture.

When the patch is applied, a new option appears in the General setup
menu of menuconfig: "Fast booting".  When this option is enabled, you
are asked to set the value of another new option: 'Loops per jiffy'.
These set the config variables CONFIG_FASTBOOT and CONFIG_FASTBOOT_LPJ.

diffstat for this patch is:
  Documentation/Configure.help |   23 +++++++++++++++++++++++
  arch/i386/config.in          |    6 ++++++
  init/main.c                  |   13 +++++++++++++
  3 files changed, 42 insertions(+)

To apply the patch, in the root of a kernel tree use:
patch -p1 <fastboot_lpj.patch
```

Source code



```
Signed-off-by: Tim Bird <tim.bird@am.sony.com>

-----------------------------------------------------------------------------
diff -u -ruN linux-2.4.20.orig/Documentation/Configure.help linux-2.4.20/Documentation/Configure.help
--- linux-2.4.20.orig/Documentation/Configure.help      Thu Nov 28 15:53:08 2002
+++ linux-2.4.20/Documentation/Configure.help   Tue Sep 30 15:32:35 2003
@@ -5274,6 +5274,29 @@
   replacement for kerneld.) Say Y here and read about configuring it
   in <file:Documentation/kmod.txt>.

+Fast booting support
+CONFIG_FASTBOOT
+  Say Y here to enable faster booting of the Linux kernel.  If you say
+  Y here, you will be asked to provide hardcoded values for some
+  parameters that the kernel usually probes for or determines at boot
+  time.  This is primarily of interest in embedded devices where
+  quick boot time is a requirement.
+
+  If unsure, say N.
+
+Fast boot loops-per-jiffy
+CONFIG_FASTBOOT_LPJ
+  This is the number of loops passed to delay() to achieve a single
+  HZ of delay inside the kernel.  It is roughly BogoMips * 5000.
+  To determine the correct value for your kernel, first turn off
+  the fast booting option, compile and boot the kernel on your target
+  hardware, then see what value is printed during the kernel boot.
+  Use that value here.
+
+  If unsure, don't use the fast booting option.  An incorrect value
+  will cause delays in the kernel to be incorrect.  Although unlikely,
+  in the extreme case this might damage your hardware.
+
 ARP daemon support
```

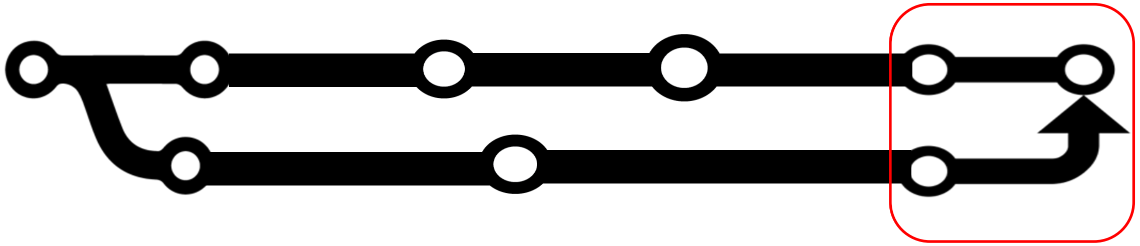
Traditional Collaboration Model



Social Coding

- Github, Bitbucket, GitLab, etc.
- Add social networking features to coding
    - Follow users
    - Watch repositories
- Allows team structure to emerge as opposed to previous planning

Fork-based Development



Upstream

Fork/Branch

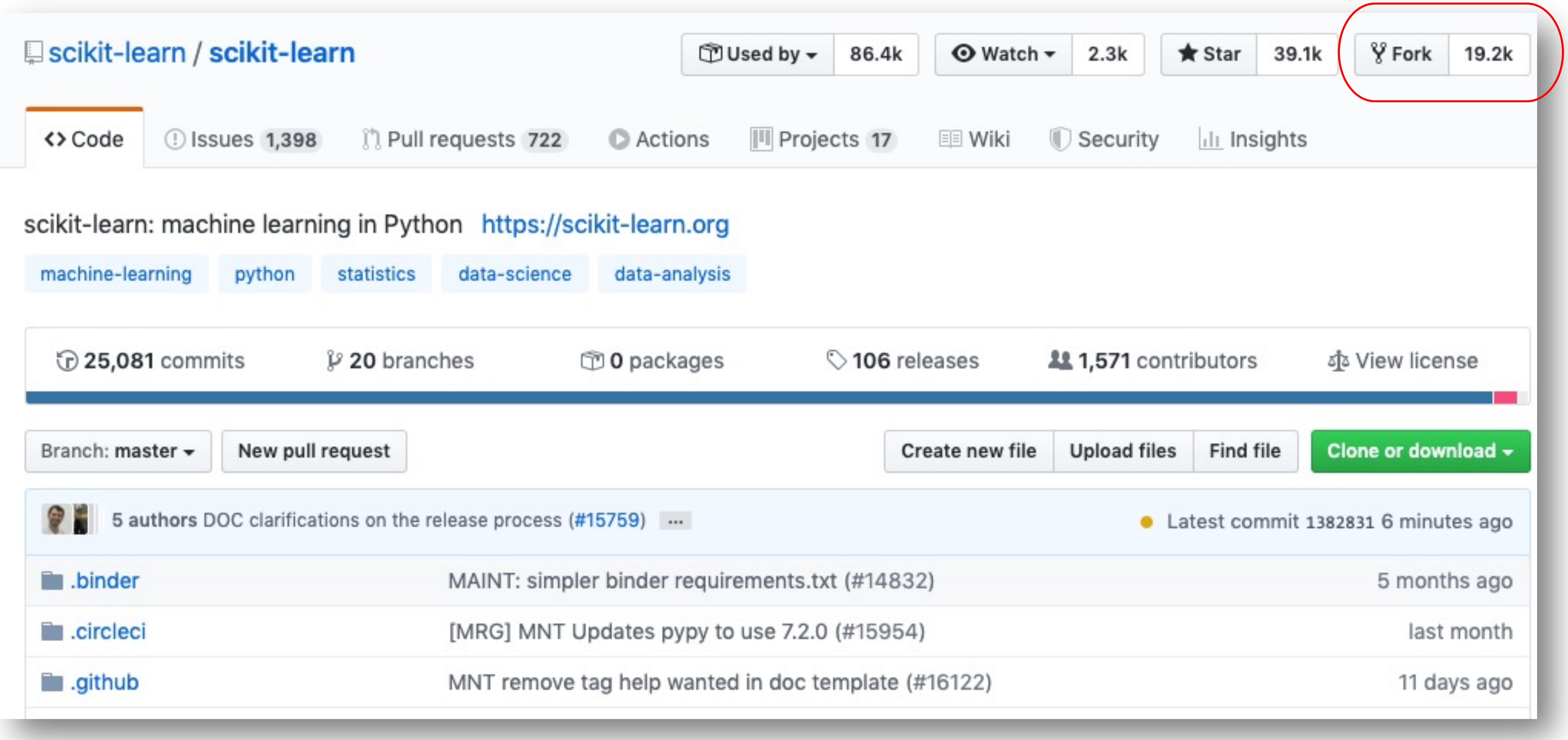
Commit

Pull Request (PR)

Fork-based / Branch-based / Pull-based Dev.

Pull Request / Merge Request

Fork-based Dev. Lowers Entry Barriers



scikit-learn / scikit-learn

scikit-learn: machine learning in Python https://scikit-learn.org

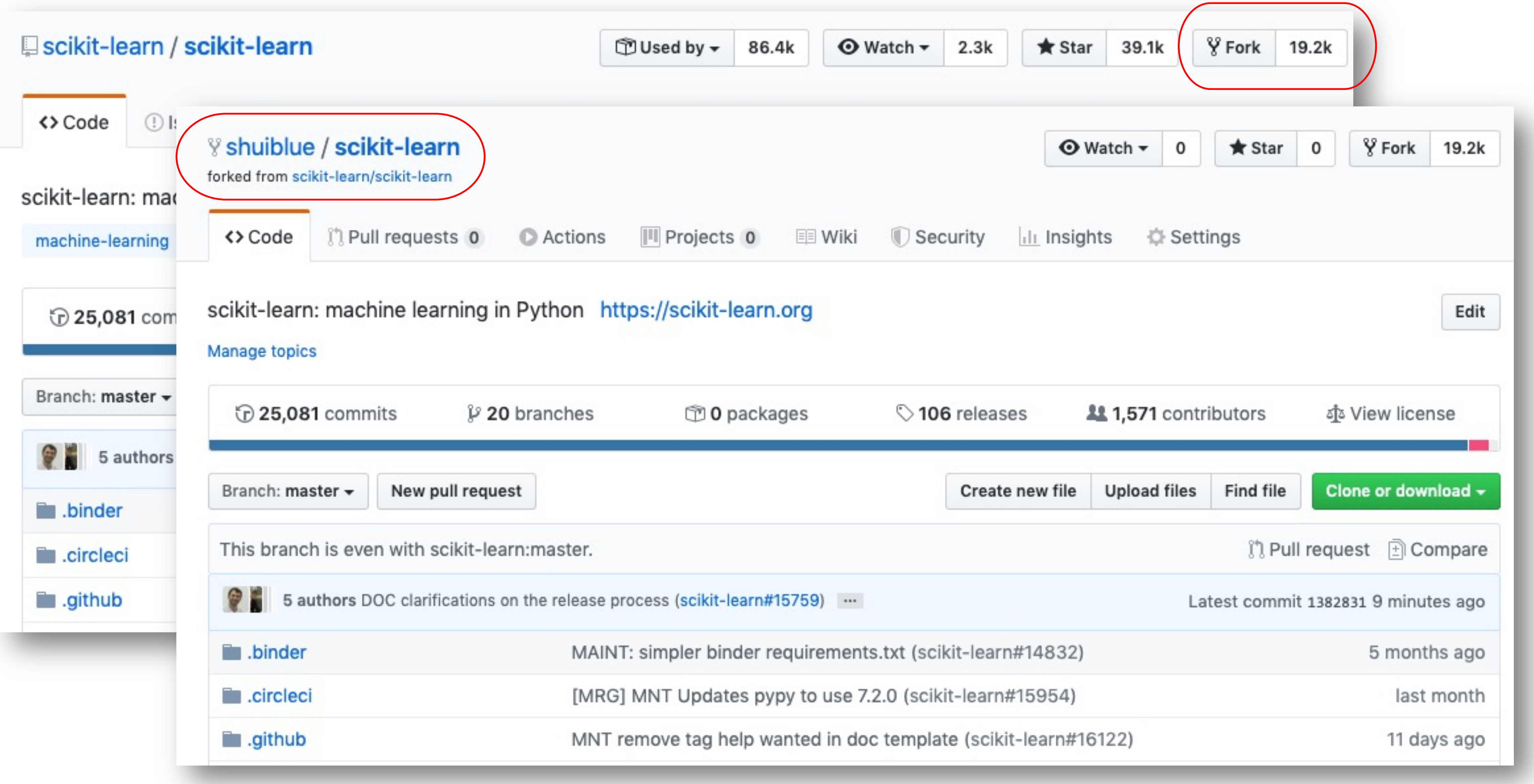
machine-learning python statistics data-science data-analysis

25,081 commits 20 branches 0 packages 106 releases 1,571 contributors View license

5 authors DOC clarifications on the release process (#15759) … Latest commit 1382831 6 minutes ago

.binder	MAINT: simpler binder requirements.txt (#14832)	5 months ago
.circleci	[MRG] MNT Updates pypy to use 7.2.0 (#15954)	last month
.github	MNT remove tag help wanted in doc template (#16122)	11 days ago

Fork-based Dev. Lowers Entry Barriers



scikit-learn: machine learning in Python https://scikit-learn.org

25,081 commits | 20 branches | 0 packages | 106 releases | 1,571 contributors | View license

This branch is even with scikit-learn:master.

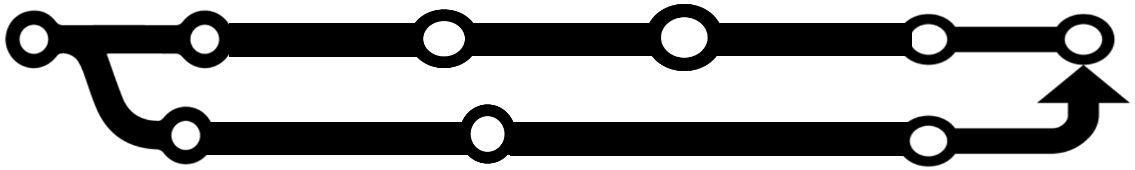
5 authors DOC clarifications on the release process (scikit-learn#15759) ... Latest commit 1382831 9 minutes ago

.binder	MAINT: simpler binder requirements.txt (scikit-learn#14832)	5 months ago
.circleci	[MRG] MNT Updates pypy to use 7.2.0 (scikit-learn#15954)	last month
.github	MNT remove tag help wanted in doc template (scikit-learn#16122)	11 days ago

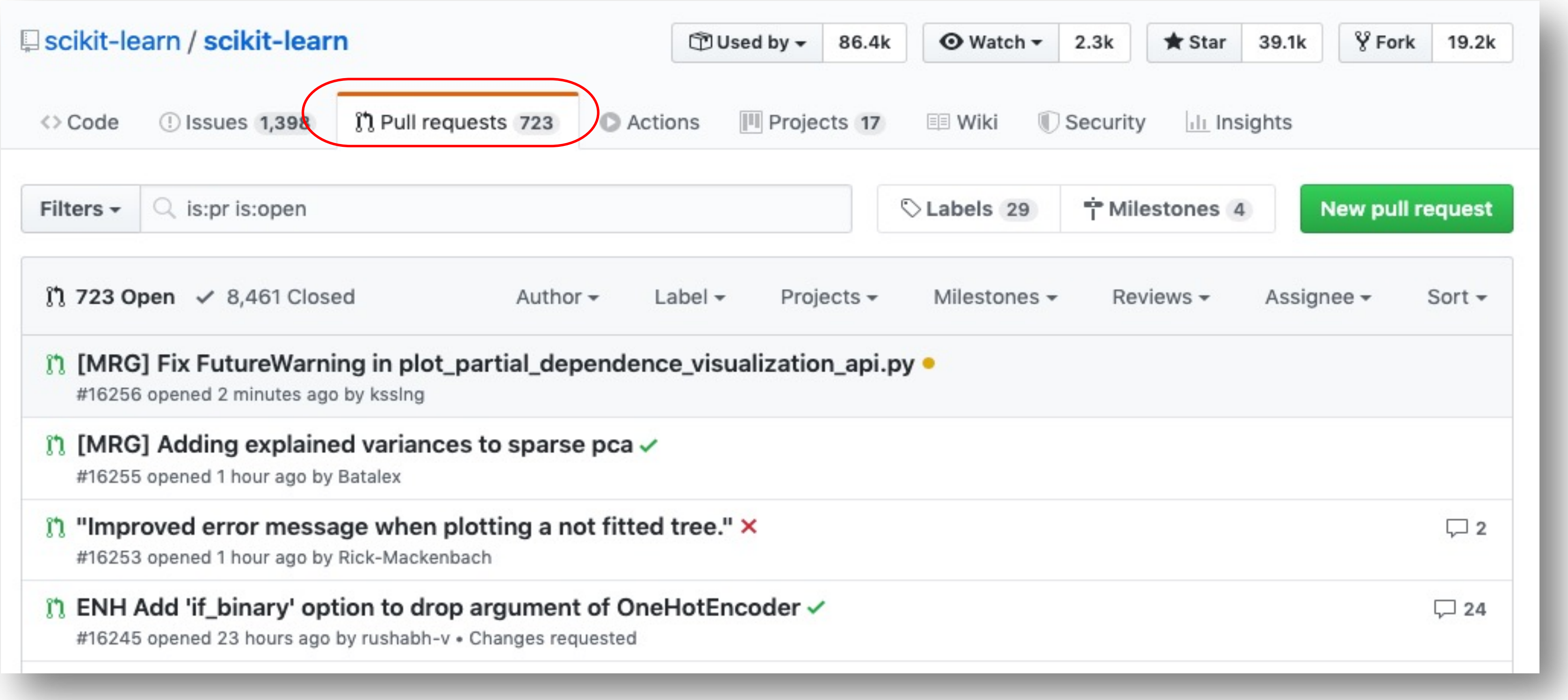
Fork-based Dev. Lowers Entry Barriers

Upstream

Fork/Branch



Pull Request (PR)



scikit-learn / scikit-learn

723 Open ✓ 8,461 Closed

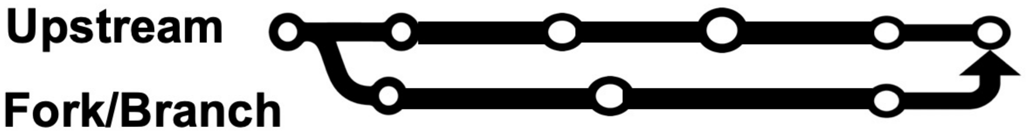
[MRG] Fix FutureWarning in plot_partial_dependence_visualization_api.py ●
#16256 opened 2 minutes ago by ksslng

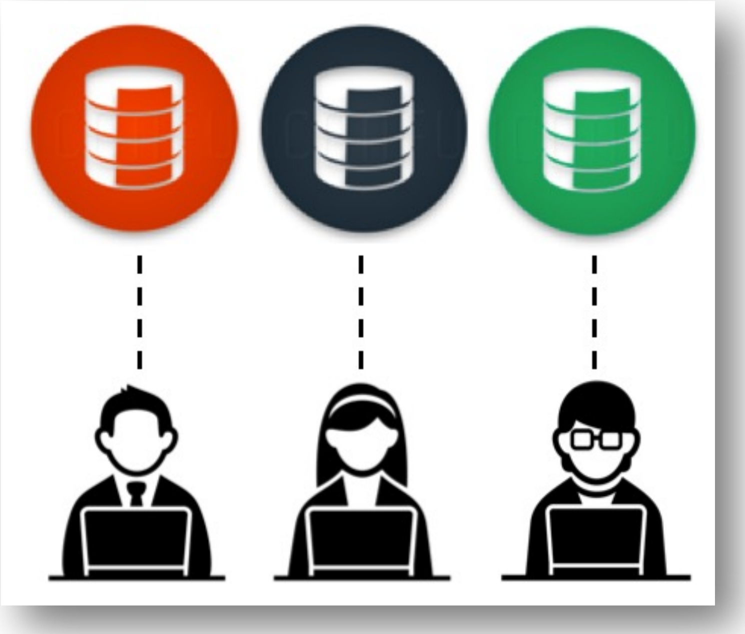
[MRG] Adding explained variances to sparse pca ✓
#16255 opened 1 hour ago by Batalex

"Improved error message when plotting a not fitted tree." ✗ 2
#16253 opened 1 hour ago by Rick-Mackenbach

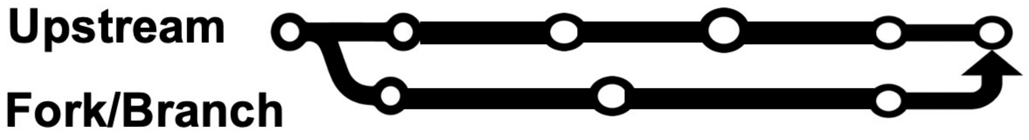
ENH Add 'if_binary' option to drop argument of OneHotEncoder ✓ 24
#16245 opened 23 hours ago by rushabh-v • Changes requested

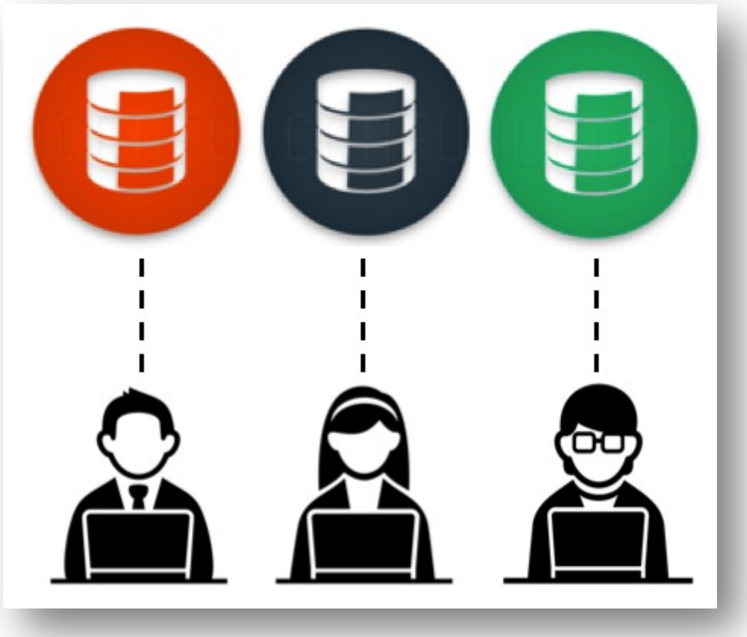
Fork-based Development



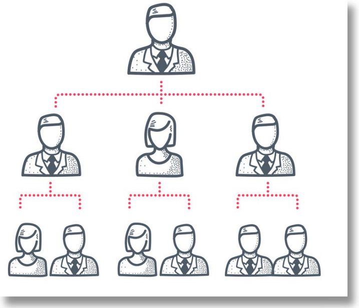


Fork-based Development





VS



90'S
open source

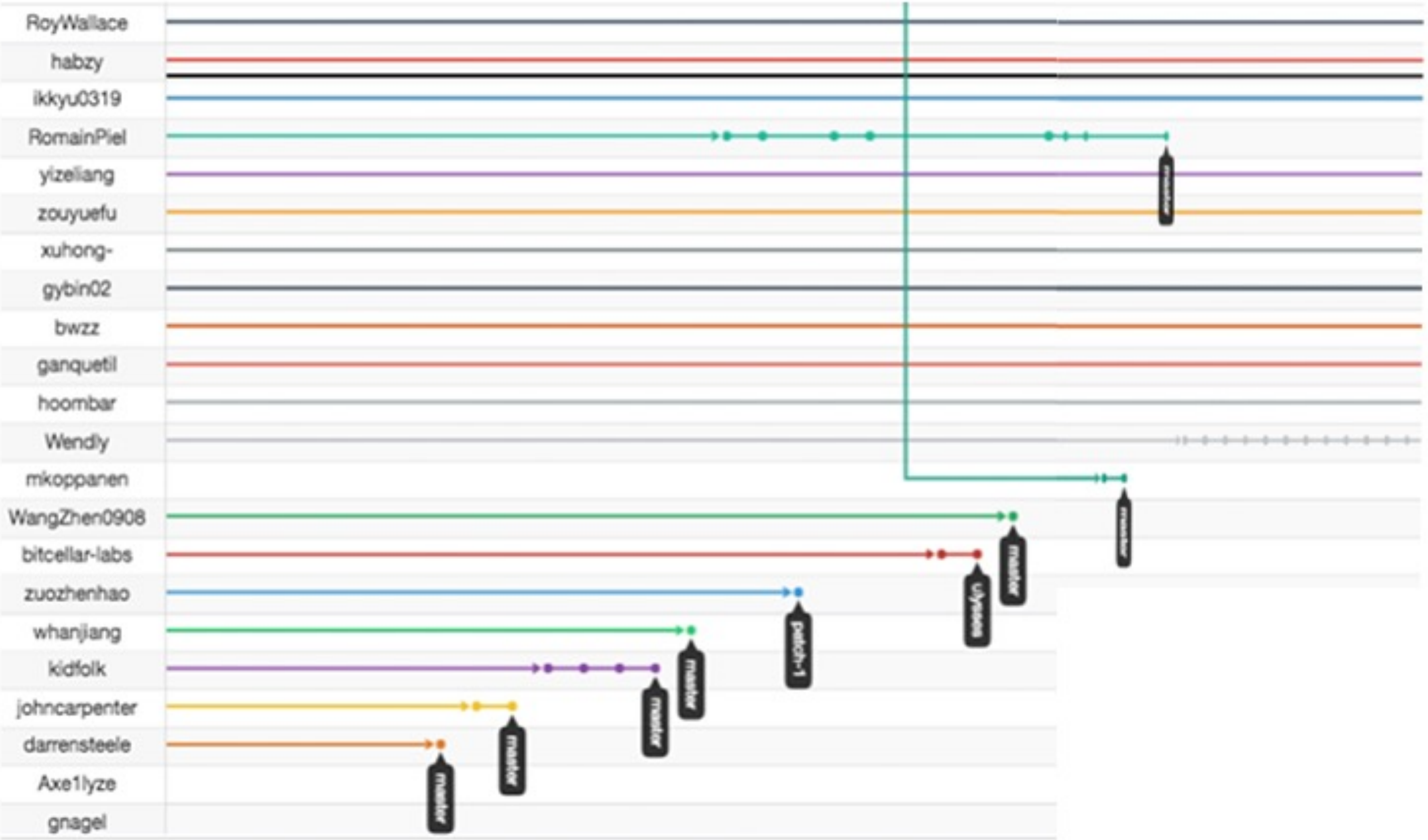
Fork-based Dev. Becomes Popular

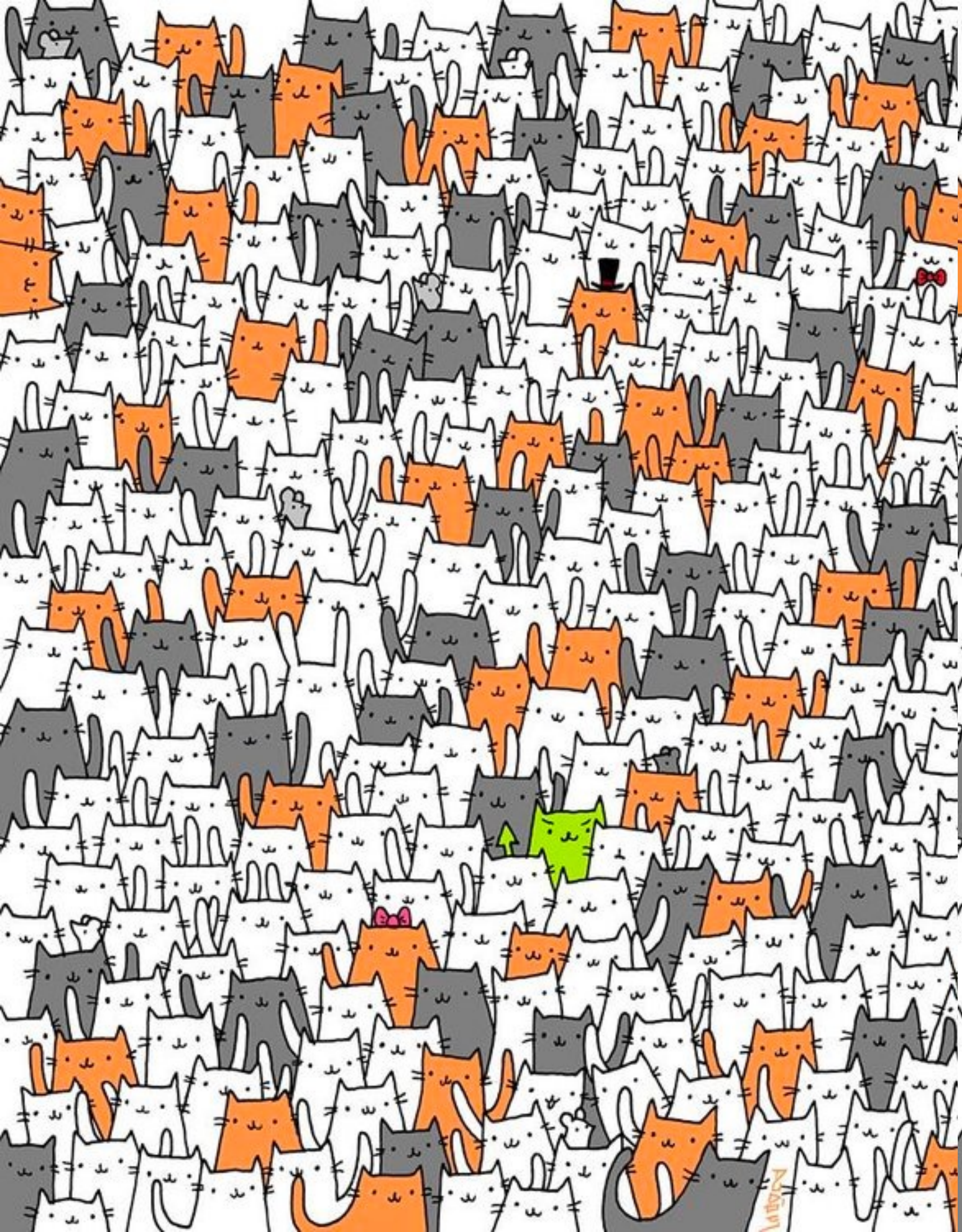
NETFLIX
Ford
American Airlines
GROUPON®
Qualcomm
Linked in
Spotify®

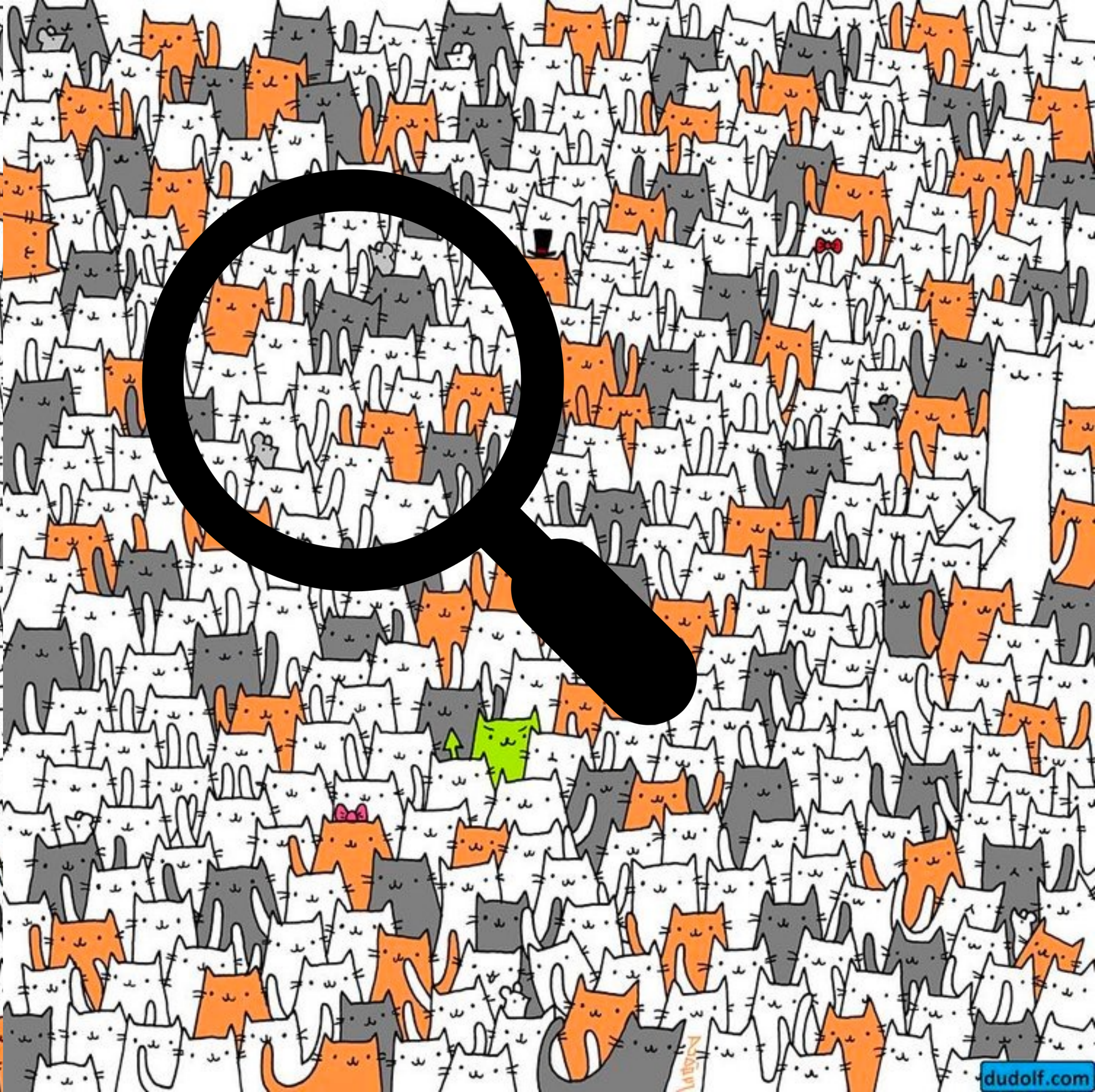
Companies

https://github.com/customer-stories?type=enterprise

Problem -- Lost Contributions







dudolf.com

Guest Lecture – Mike Hoye

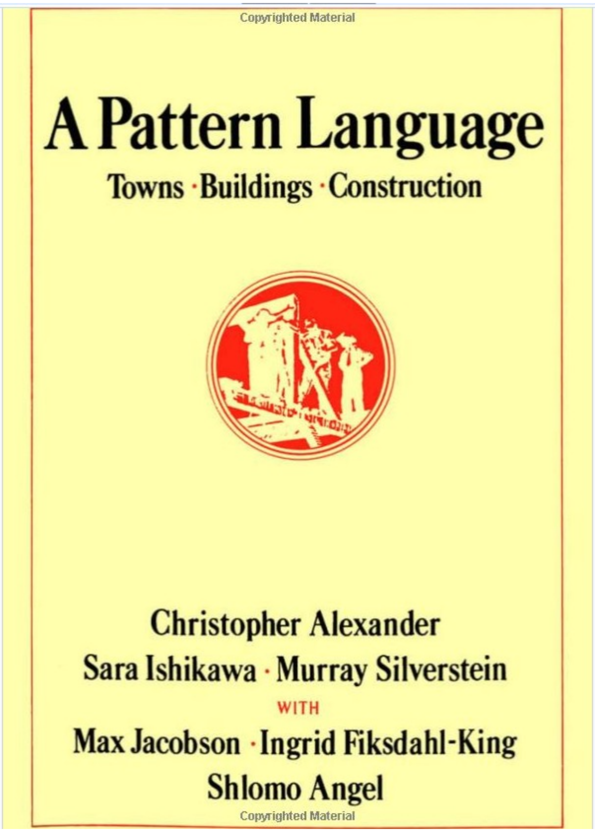
"Mike Hoye is a senior staff project and community manager at Mozilla, creators of Firefox, the web browser you actually want. He's here today to talk about the evolution of open source - the ideas, practices, licenses, values, how they're changing and what's coming next."

Design Patterns 1: SOLID

Shurui Zhou

The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

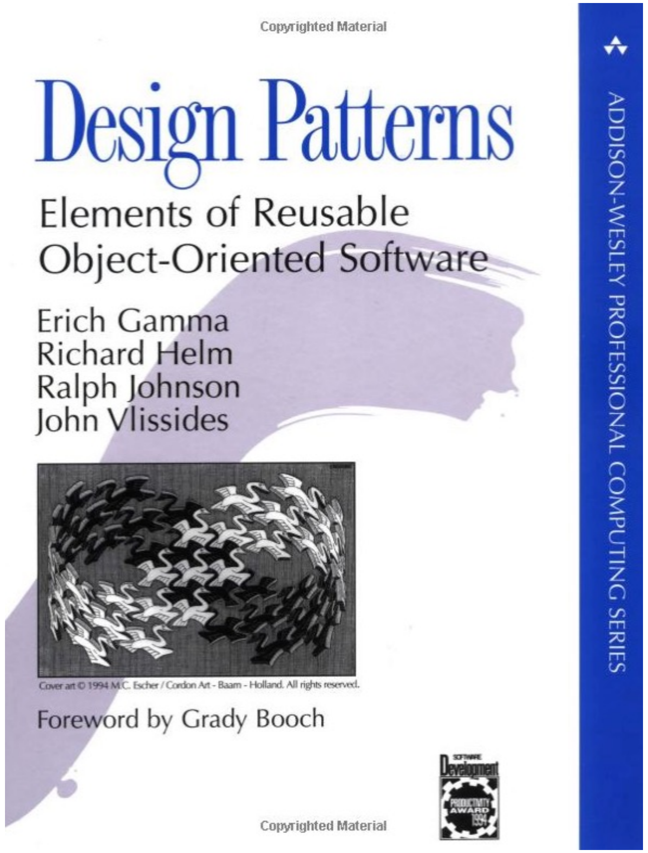
History of Patterns



A Pattern Language

Towns · Buildings · Construction

Christopher Alexander
Sara Ishikawa · Murray Silverstein
WITH
Max Jacobson · Ingrid Fiksdahl-King
Shlomo Angel



Design Patterns

Elements of Reusable
Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

- Elements of Reusable Object-Oriented Software
- 23 OO patterns

Design Patterns

- When used strategically, they can make a programmer significantly more efficient by allowing them to avoid reinventing the proverbial wheel, instead using methods refined by others already
- Provide a useful common language to conceptualize repeated problems and solutions when discussing with others or managing code in larger teams.

Levels of Abstraction

- Requirements
  - high-level “what” needs to be done
- Architecture (High-level design)
  - high-level “how”, mid-level “what”
- OO-Design (Low-level design, e.g. design patterns)
  - mid-level “how”, low-level “what”
- Code
  - low-level “how”

Design vs. Architecture

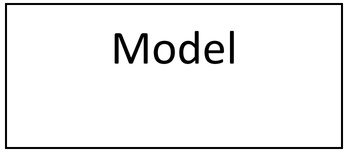
Design Questions

- How do I add a menu item in Eclipse?
- How can I make it easy to add menu items in Eclipse?
- What lock protects this data?
- How does Google rank pages?
- What encoder should I use for secure communication?
- What is the interface between objects?

Architectural Questions

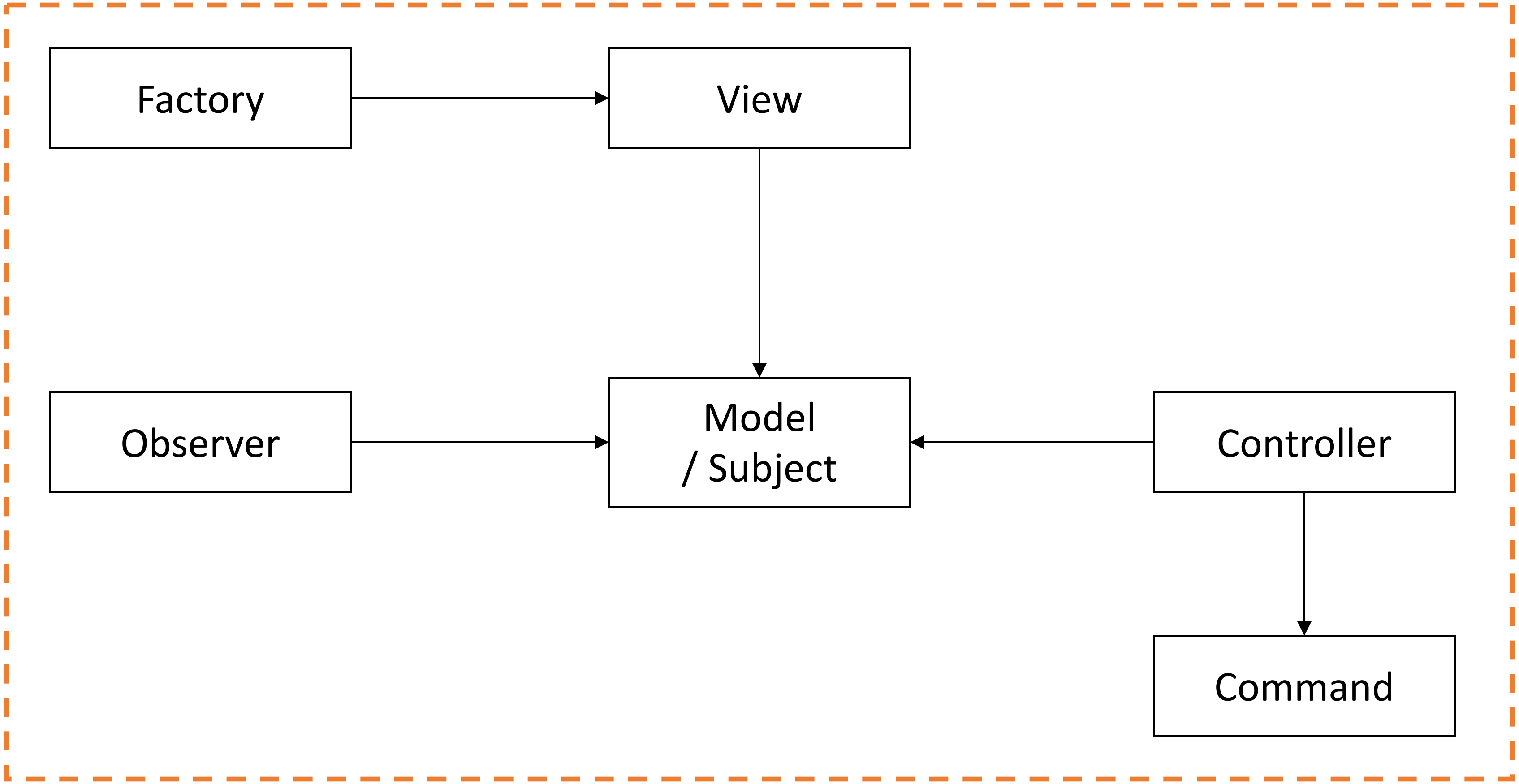
- How do I extend Eclipse with a plugin?
- What threads exist and how do they coordinate?
- How does Google scale to billions of hits per day?
- Where should I put my firewalls?
- What is the interface between subsystems?

Objects

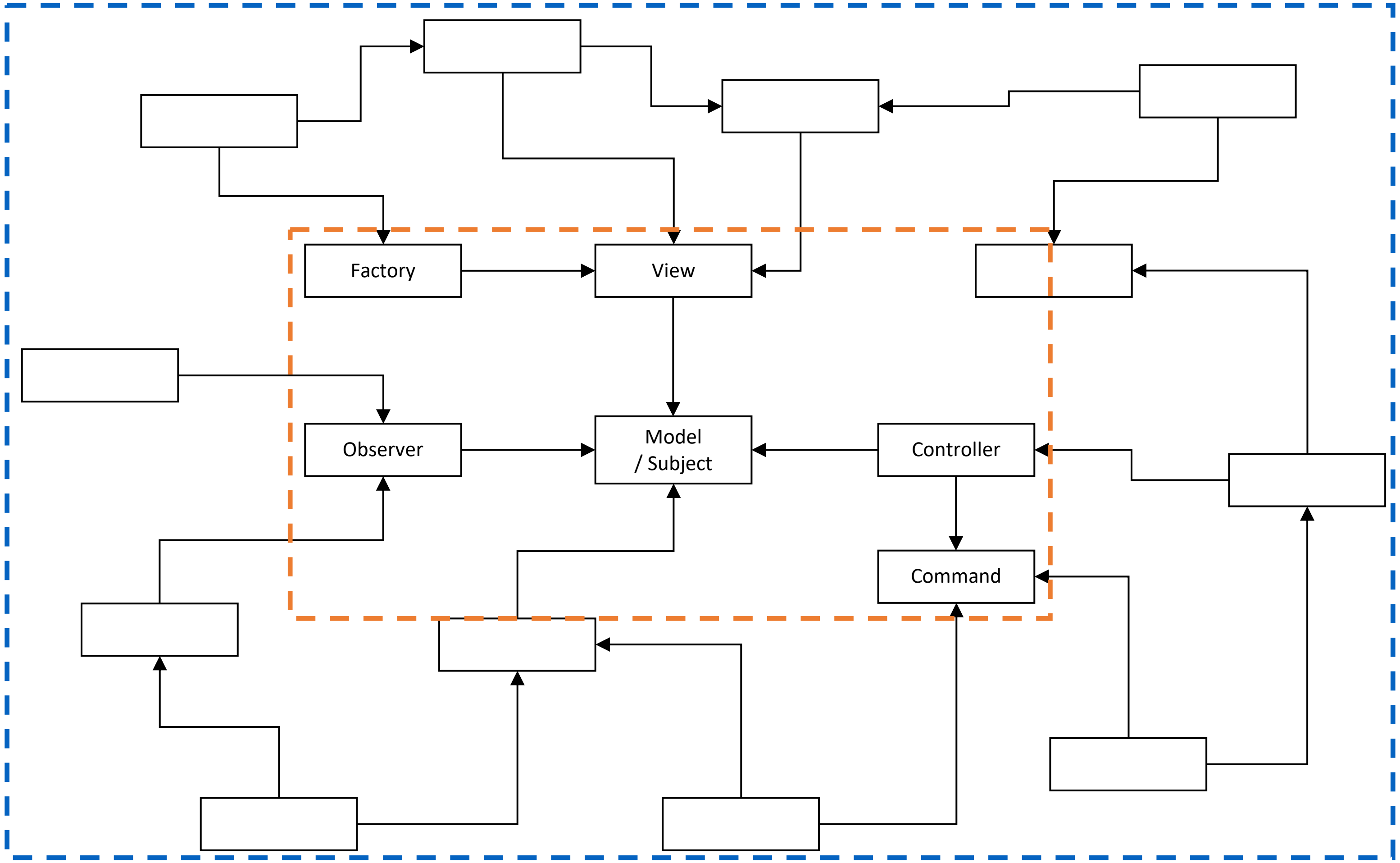


Model

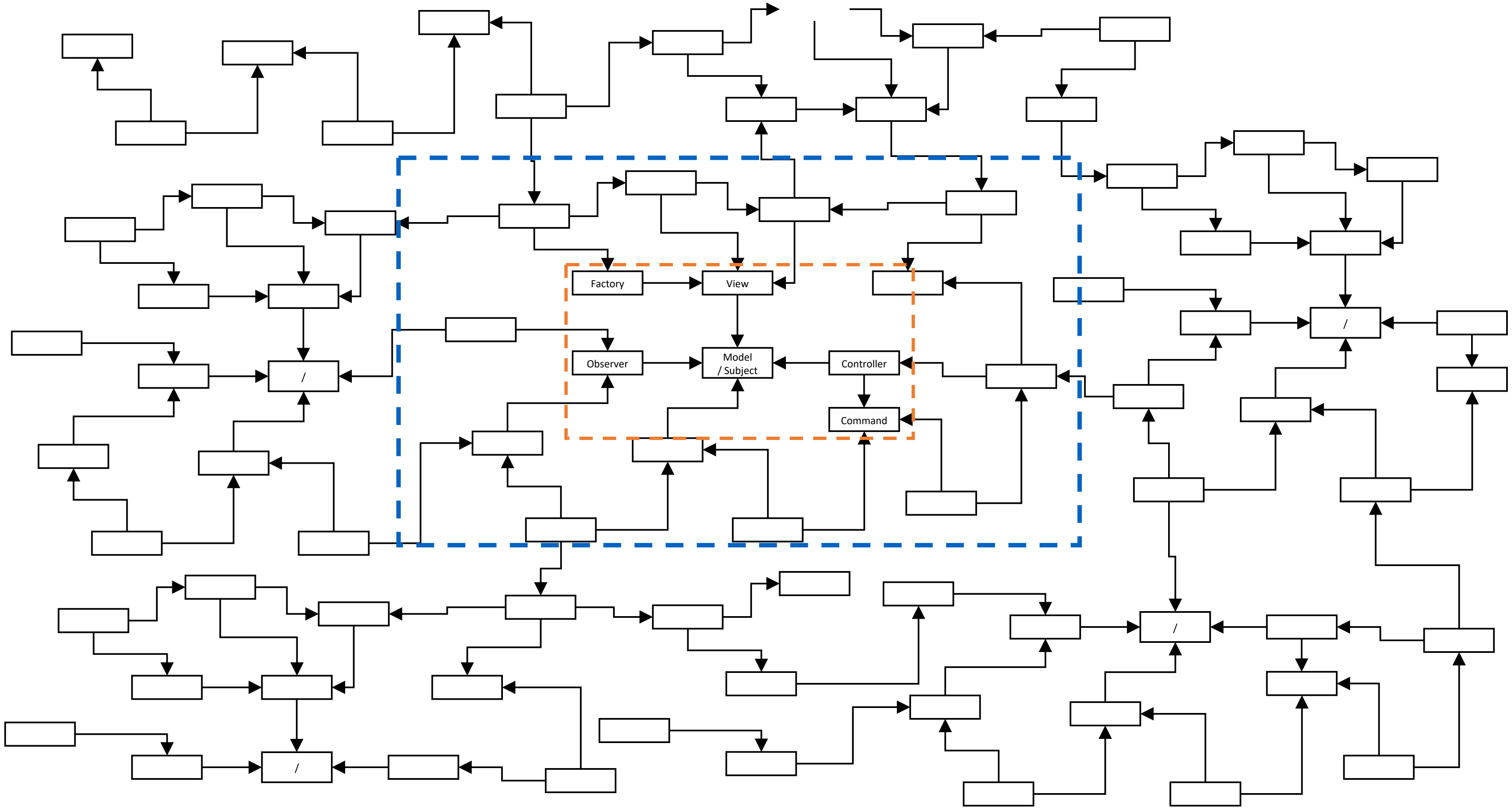
Design Patterns



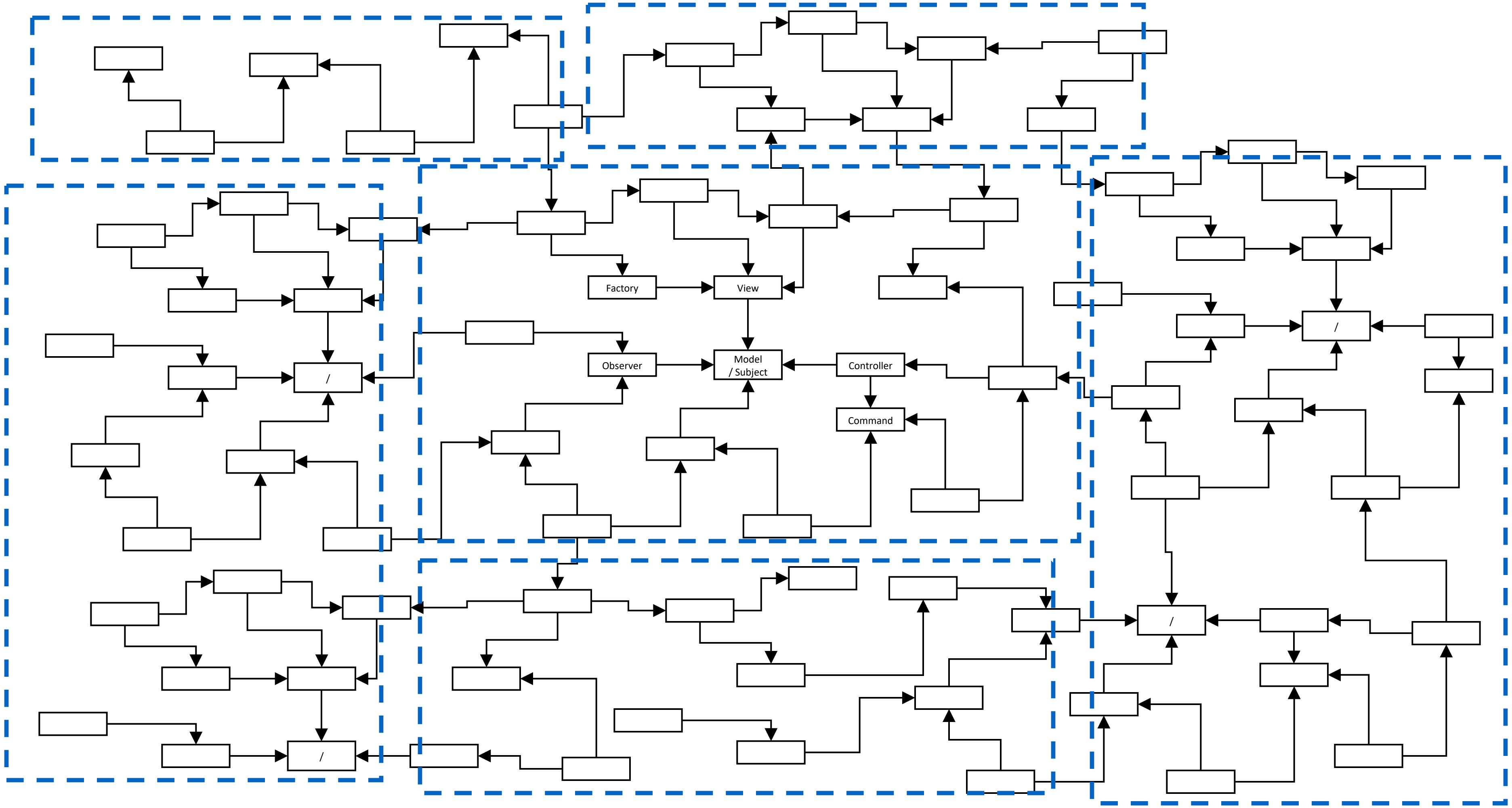
Design Patterns



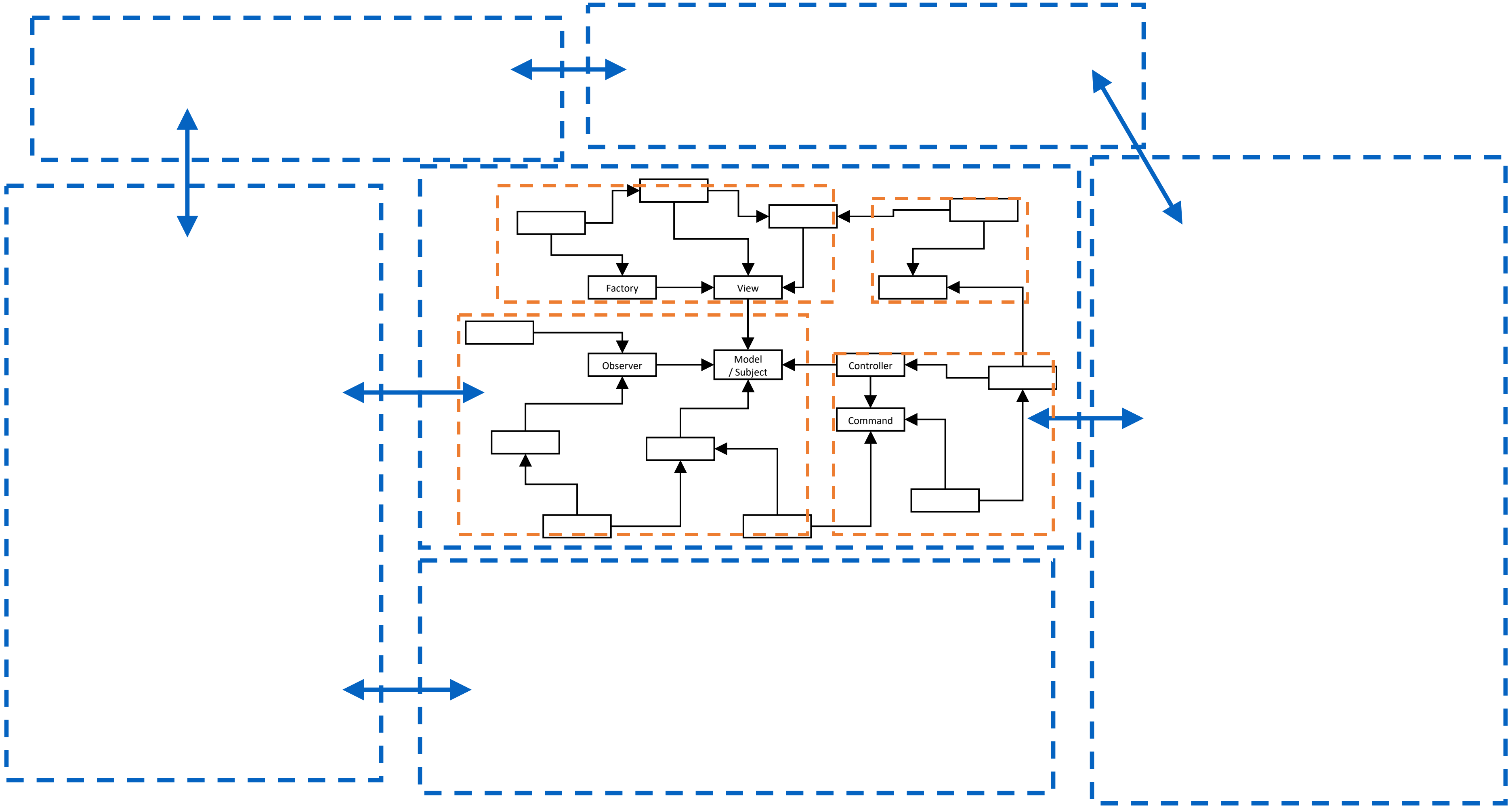
Design Patterns



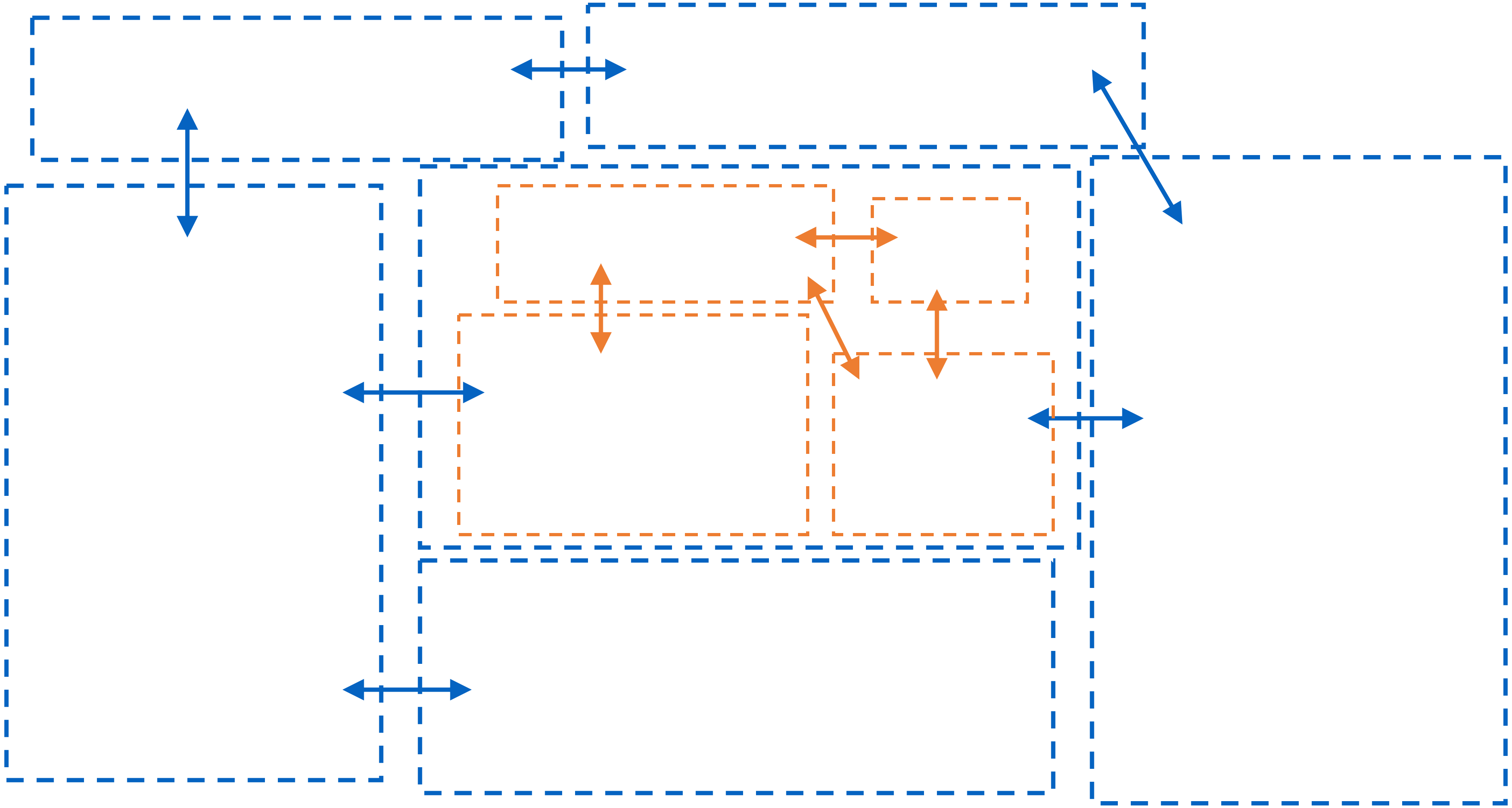
Architecture



Architecture



Architecture



Classification of patterns

- **Creational patterns** provide object creation mechanisms that increase flexibility and reuse of existing code.
- **Structural patterns** explain how to assemble objects and classes into larger structures, while keeping the structures flexible and efficient.
- **Behavioral patterns** take care of effective communication and the assignment of responsibilities between objects.

Criticism of Design Patterns

- **Kludges for a weak programming language**

Usually the need for patterns arises when people choose a programming language or a technology that lacks the necessary level of abstraction.

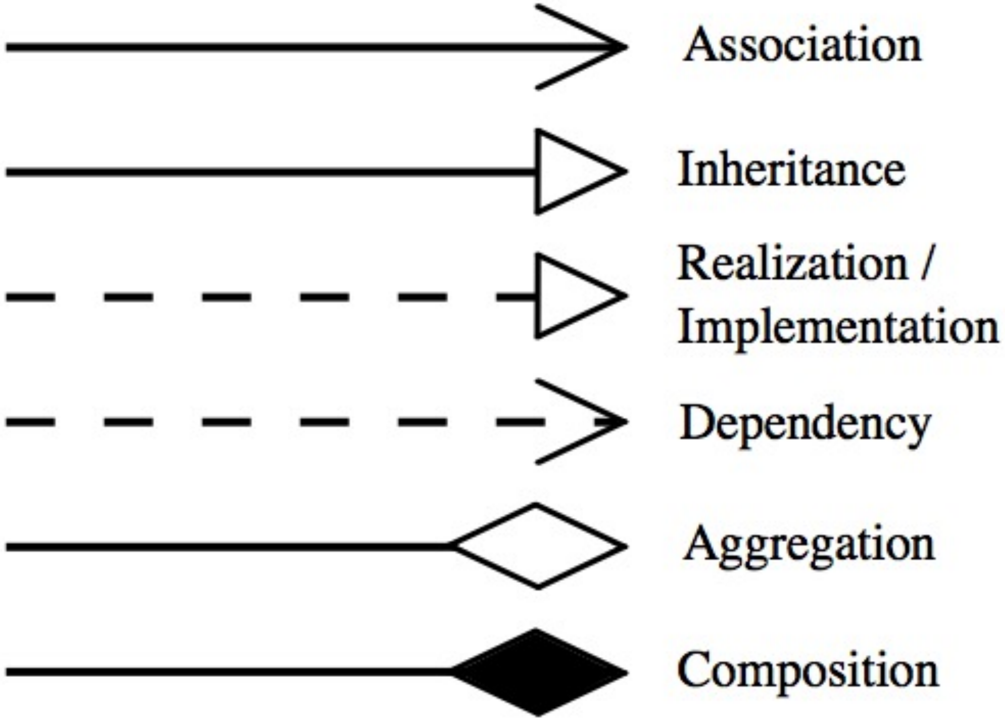
- **Inefficient solutions**

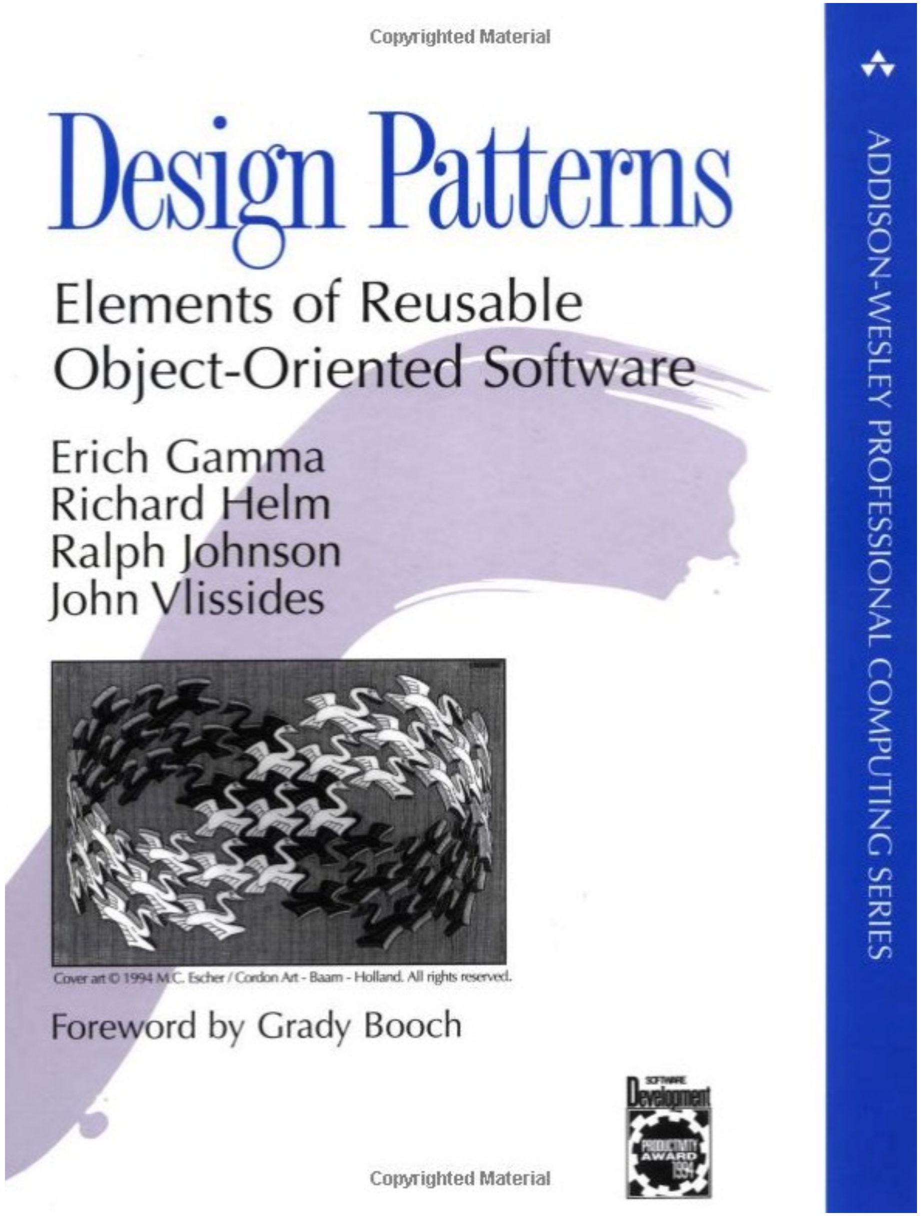
Patterns try to systematize approaches that are already widely used.

- **Unjustified use**

If all you have is a hammer, everything looks like a nail.

UML Relationships





Copyrighted Material

Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch

Copyrighted Material

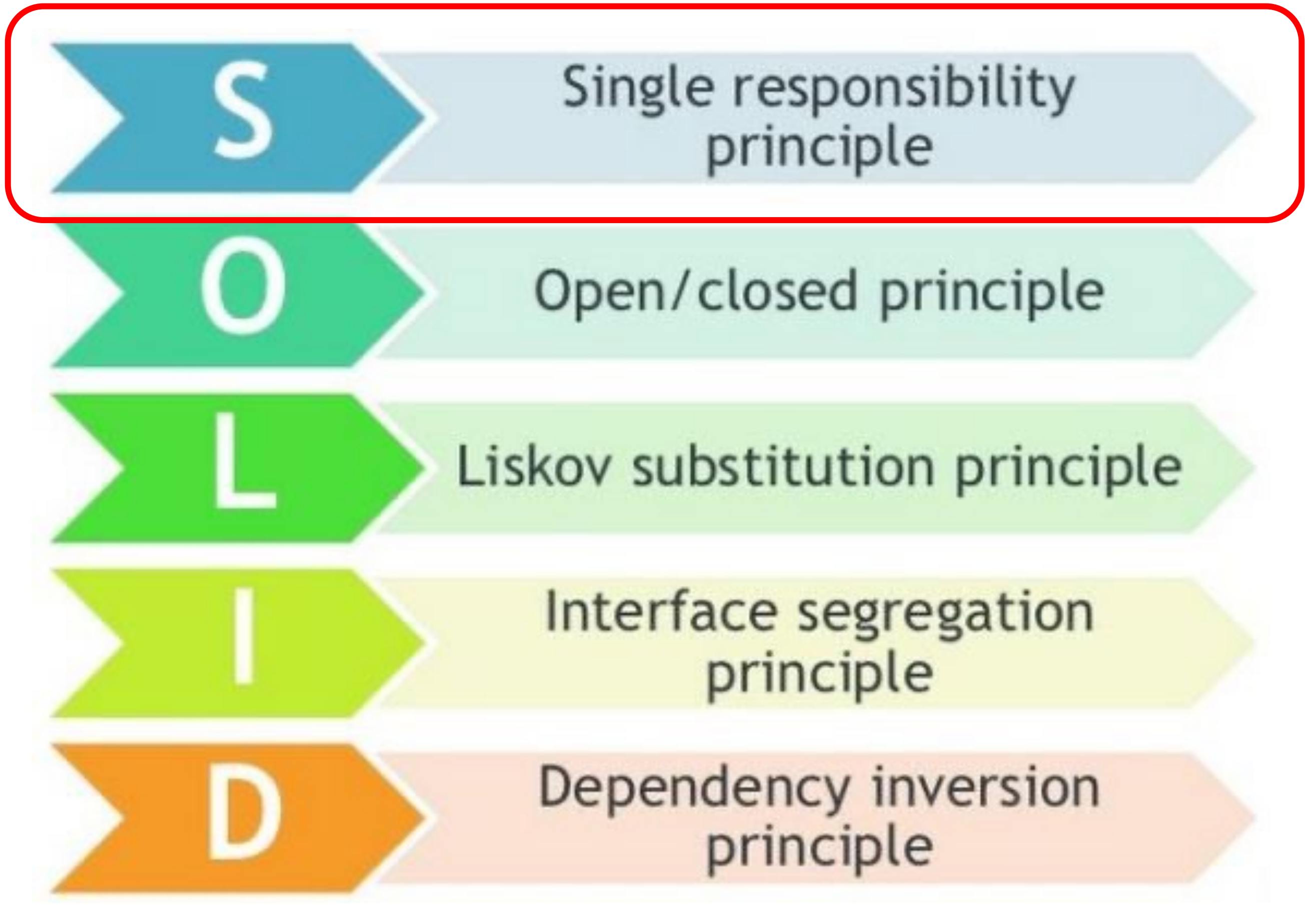
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

- Elements of Reusable Object-Oriented Software
- 23 OO patterns

Why Patterns?

- They offer solutions for specific problems
- They are easily applicable because the purpose and application are consistently described
- They make work more efficient
- They can be adapted to specific contexts
- They make communication between developers easier
- Goal: Understandable, reusable, testable, maintainable and flexible

OO Design Principles



S Single responsibility principle

O Open/closed principle

L Liskov substitution principle

I Interface segregation principle

D Dependency inversion principle

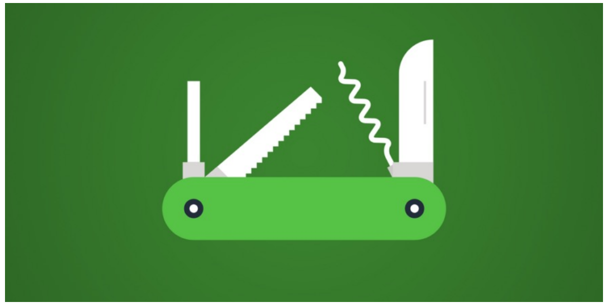
Single Responsibility Principle

Guildelines to partition your logic into classes

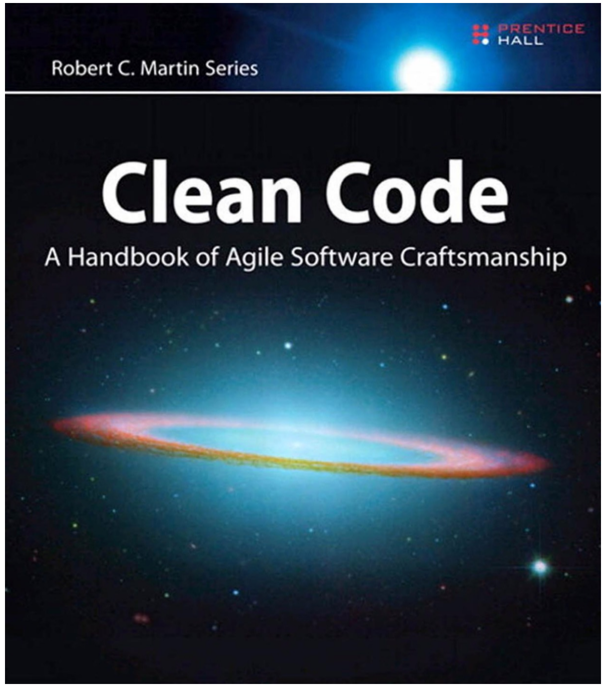
OO Design Principles

CARL QUIT. HE'S THE ONLY ONE WHO KNOWS HOW TO PROGRAM THE LEGACY SYSTEM.
scottadams@aol.com
www.dilbert.com
IT CAN'T BE THAT HARD. GO FIGURE IT OUT.
12-8-06 ©2006 Scott Adams, Inc./Dist. by UFS, Inc.
?
FRACK.

Single Responsibility Principle



A class should have one, and only one, reason to change.
Just because you can, doesn’t mean you should

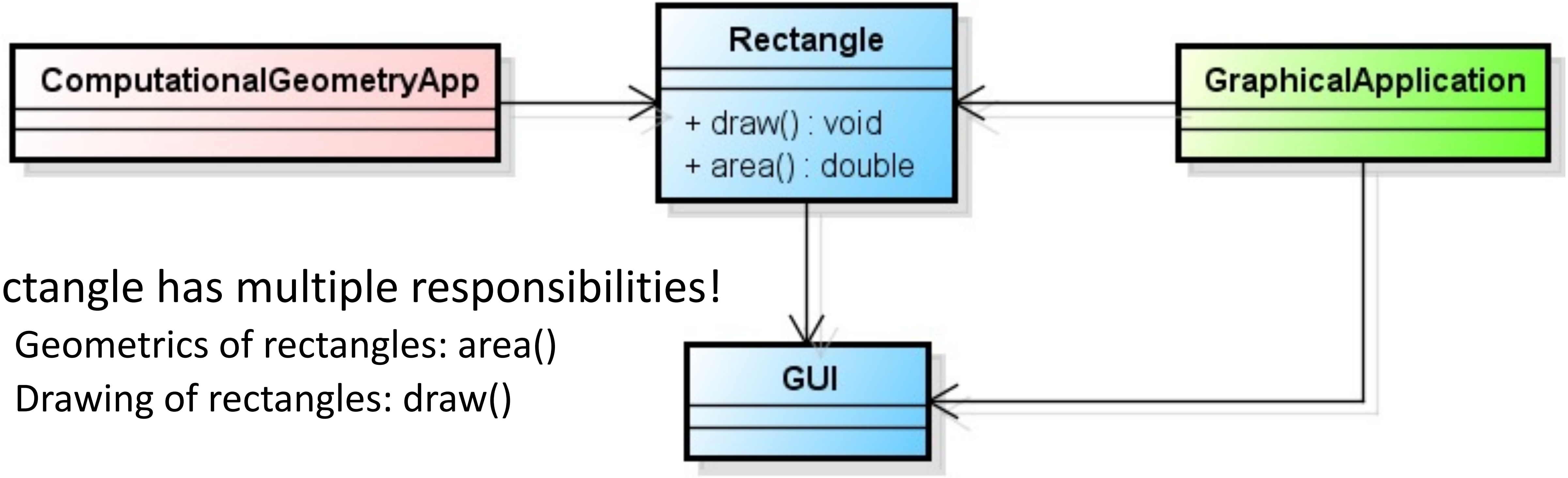


Benefits:

- Frequency and Effects of Changes
- Easier to Understand

Q: What is the responsibility of your class/component/microservice?

Single Responsibility Principle



Rectangle has multiple responsibilities!

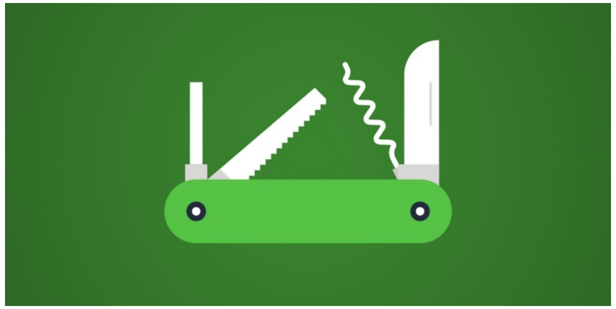
- Geometrics of rectangles: area()
- Drawing of rectangles: draw()

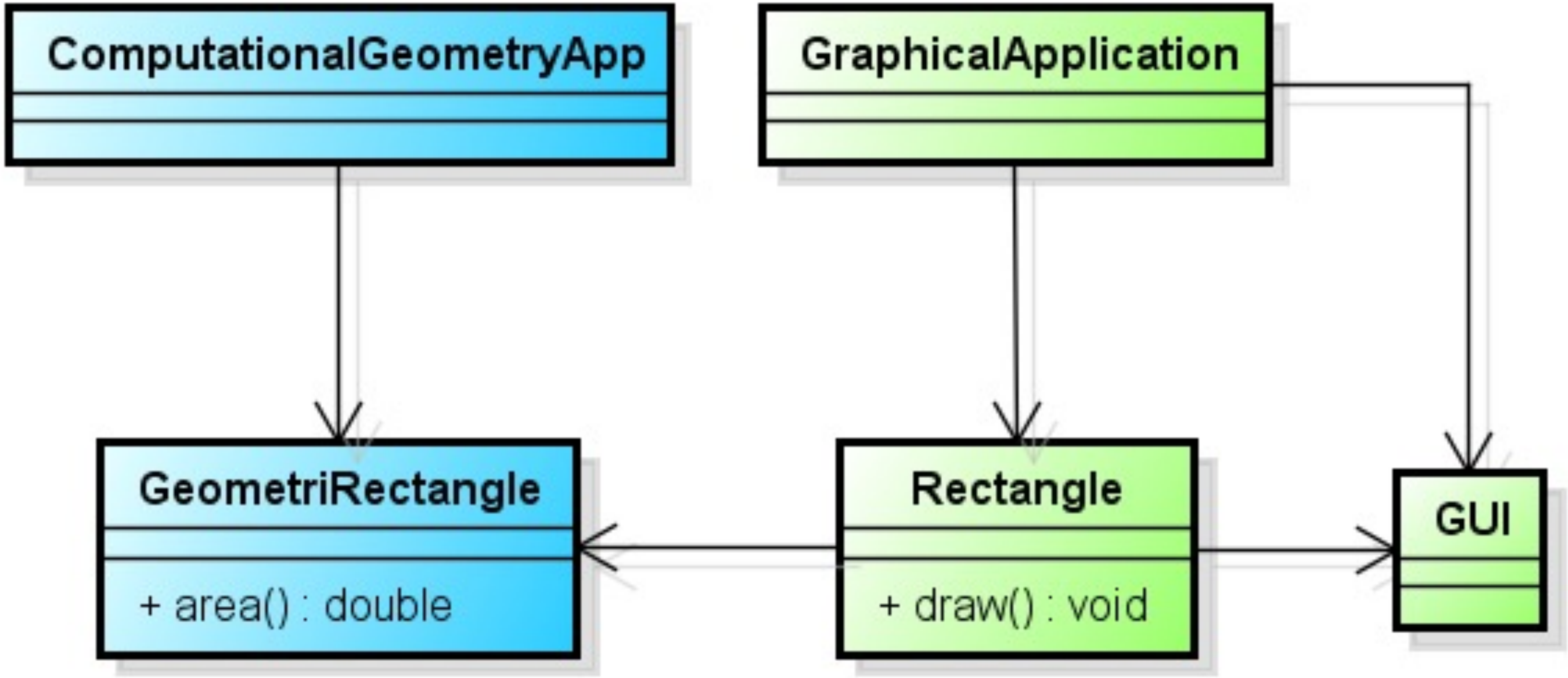
Dependency in UML

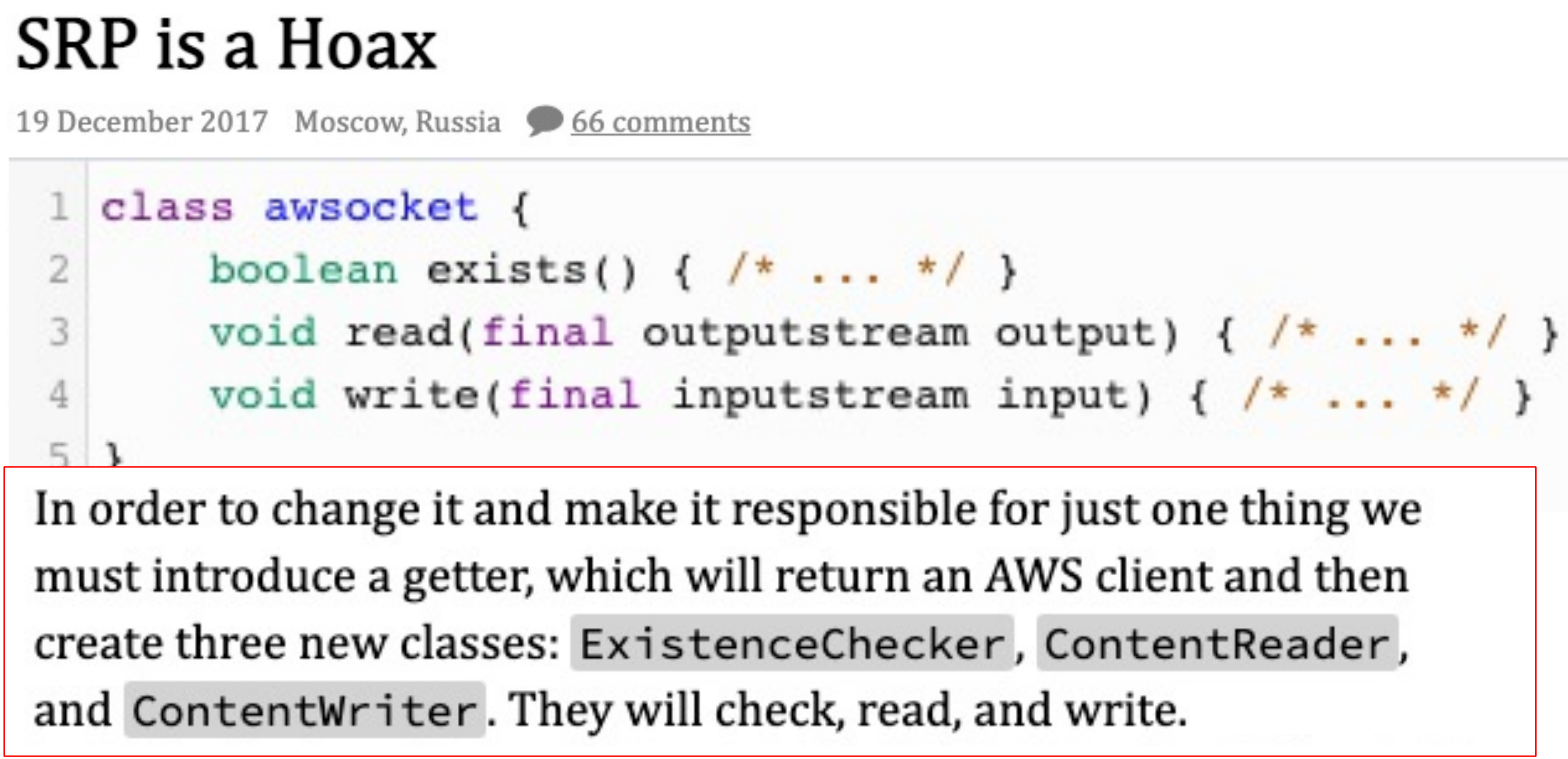
- a **directed relationship** -- some UML element or a set of elements requires, needs or depends on other model elements for **specification** or **implementation**.
- Also called a **supplier** - **client** relationship. Modification of the supplier may impact the client elements.

https://www.uml-diagrams.org/dependency.html

Single Responsibility Principle







SRP is a Hoax

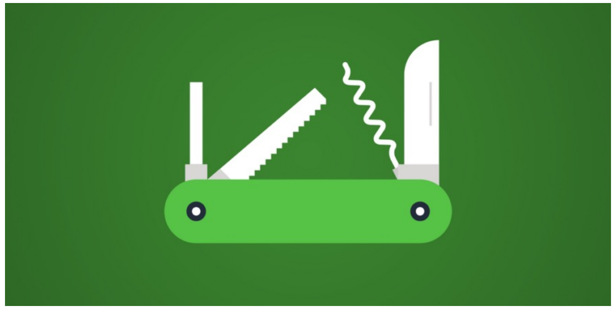
19 December 2017 Moscow, Russia 66 comments

```
class awsocket {
    boolean exists() { /* ... */ }
    void read(final outputstream output) { /* ... */ }
    void write(final inputstream input) { /* ... */ }
}
```

In order to change it and make it responsible for just one thing we must introduce a getter, which will return an AWS client and then create three new classes: `ExistenceChecker`, `ContentReader`, and `ContentWriter`. They will check, read, and write.

https://www.yegor256.com/2017/12/19/srp-is-hoax.html

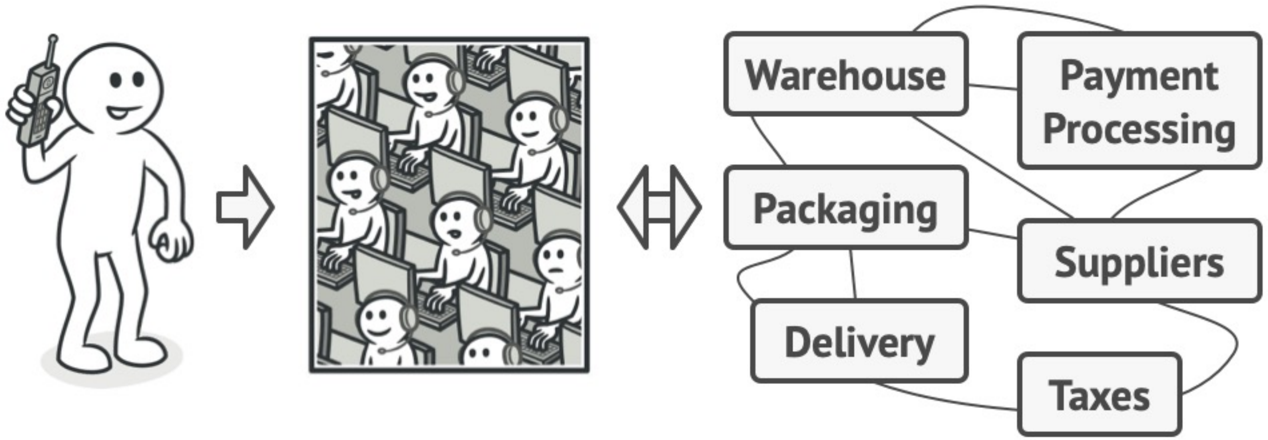
Single Responsibility Principle (SRP)



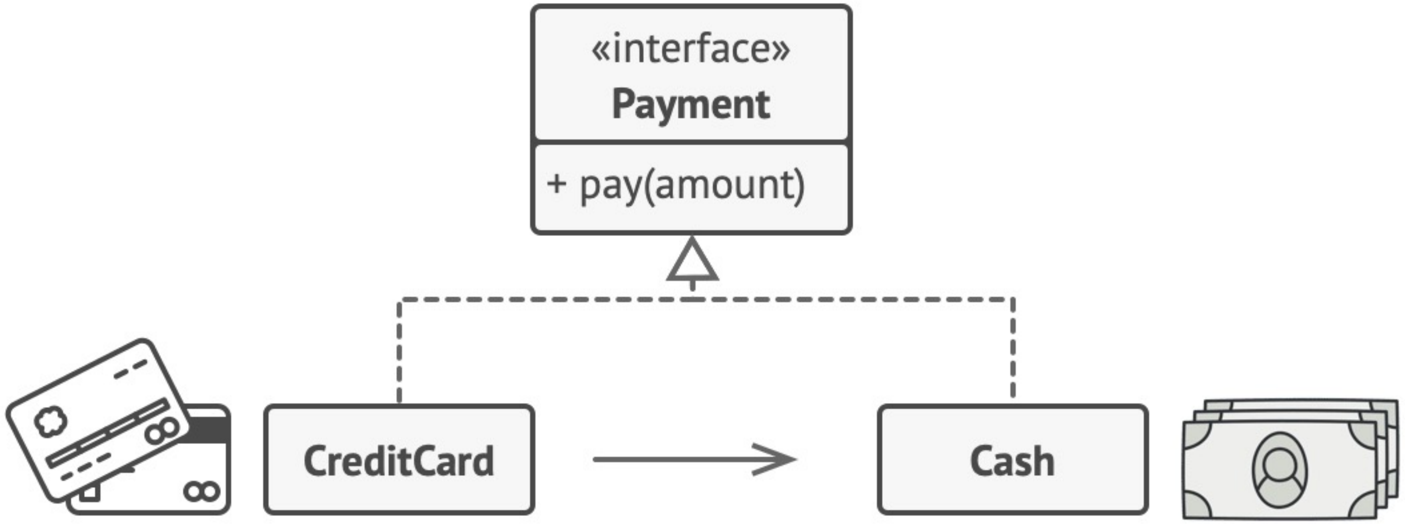
- It is all about **cohesion --** the degree to which the elements inside a module belong together. in one sense, it is a measure of the strength of relationship between the methods and data of a class and some unifying purpose or concept served by that class (wikipedia)
- if a client of a class tends to always use all the functions of that class, then the class is probably highly cohesive

Corresponding Design Patterns

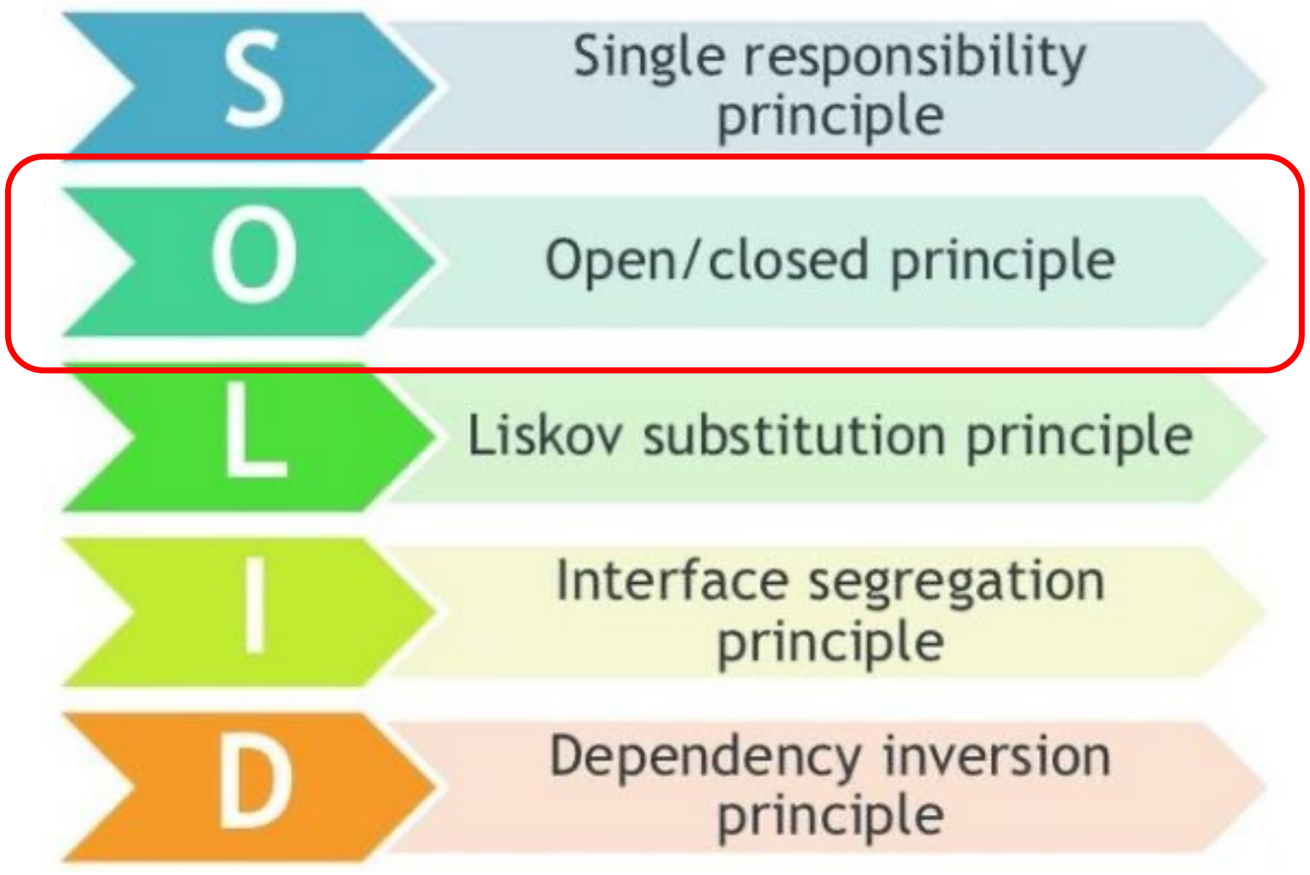
- Façade



- Proxy



OO Design Principles



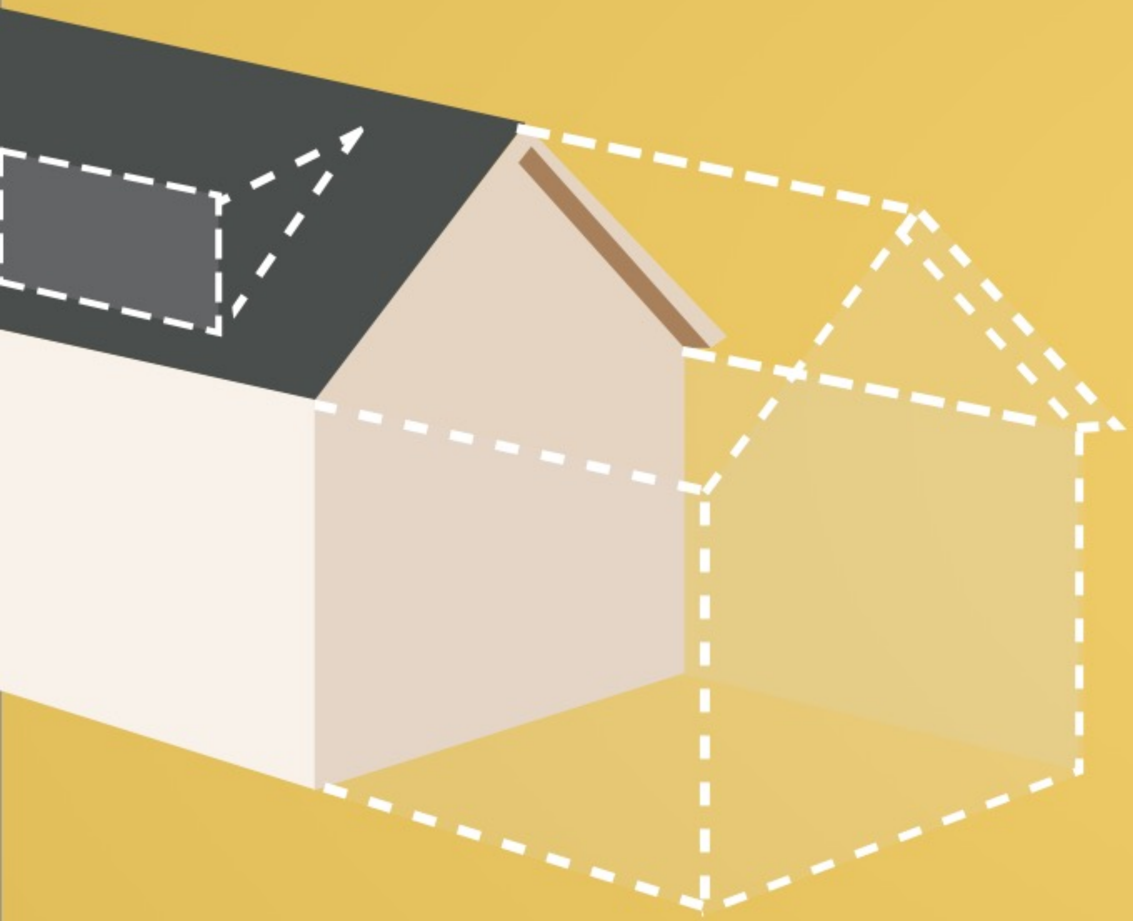
S Single responsibility principle

O Open/closed principle

L Liskov substitution principle

I Interface segregation principle

D Dependency inversion principle



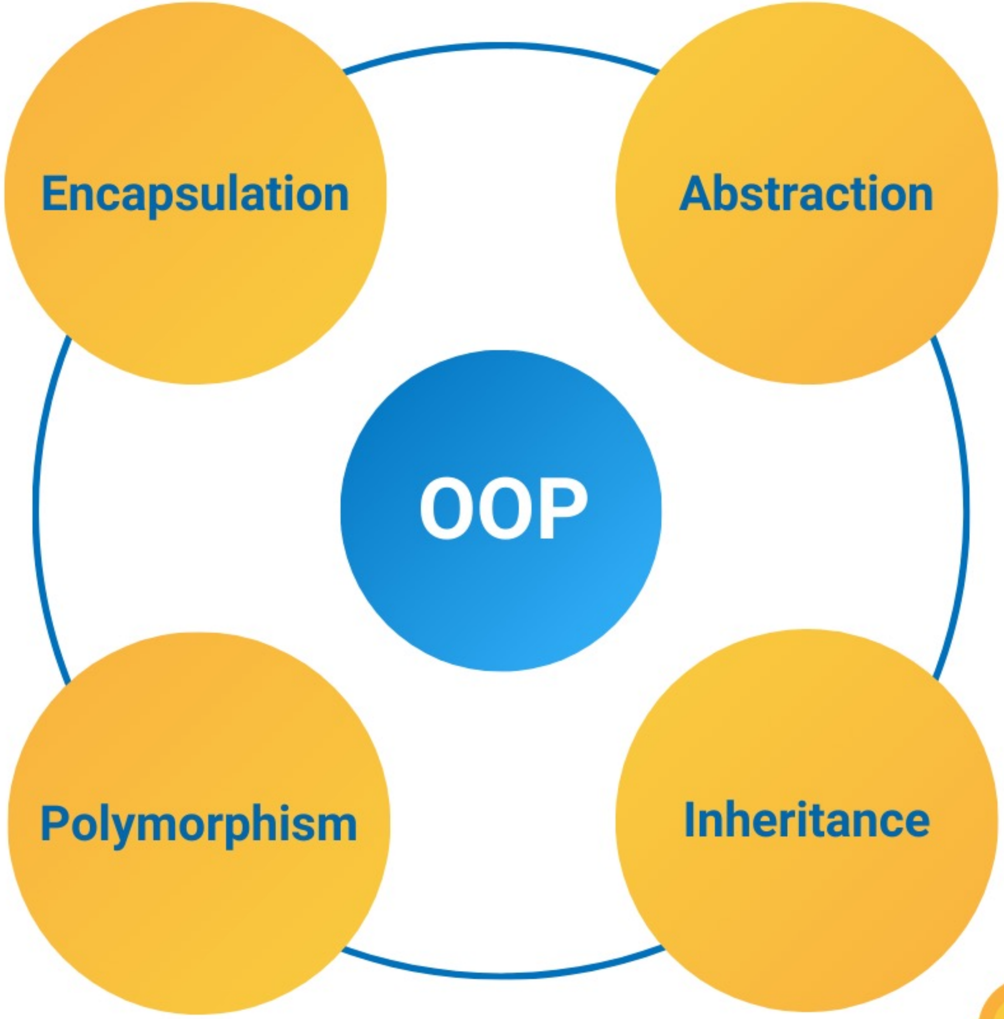
The Open-Closed Principle

Extending Your Entities Correctly

Open-Closed Principle (OCP)

- Software entities should be open for extension, but closed for modification.
- better stability, maintainability

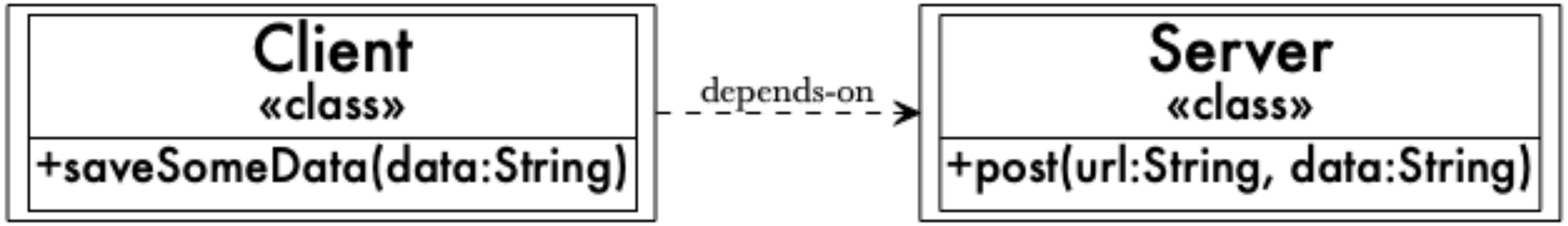
Fundamental Object-Oriented Design Principle



Open-Closed Principle (OCP)

- Implementation:
  - <u>inheritance</u>
  - <u>composition</u>
- Benefits:
  - extend a component's logic without breaking backward compatibility
  - test different component implementations (that have the same logic) against each other.

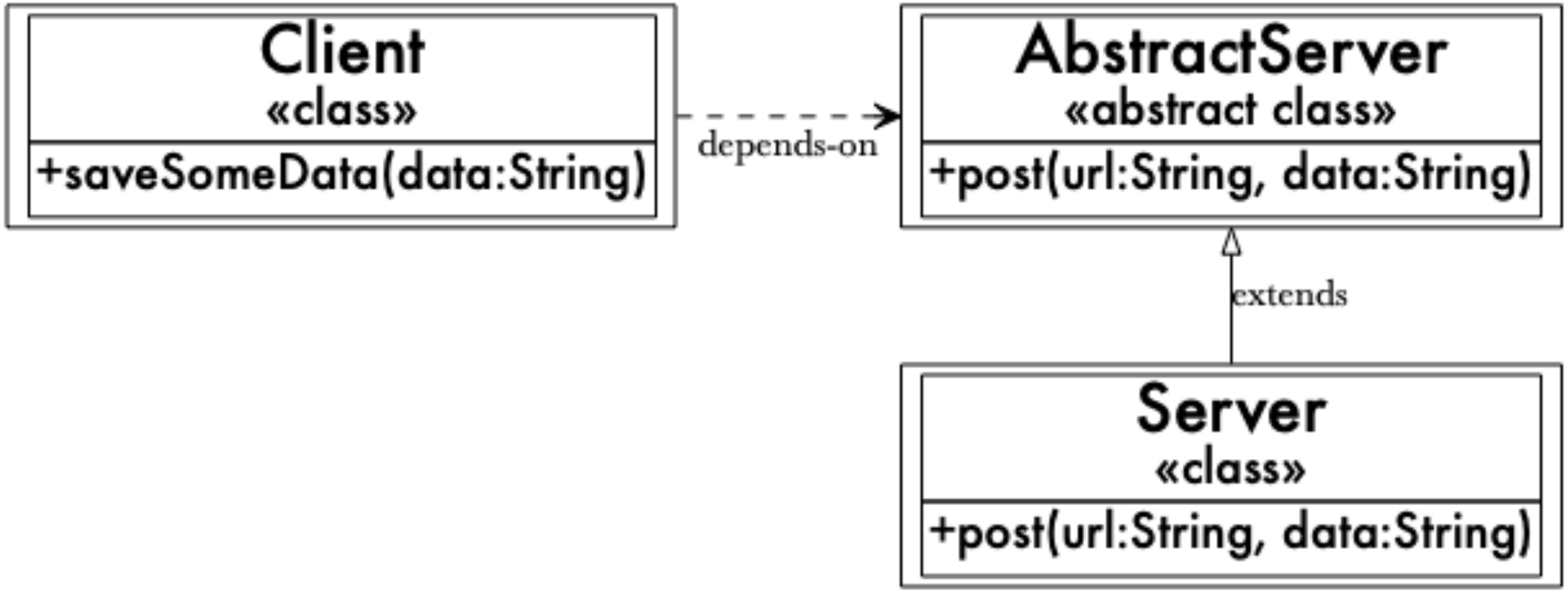
Open-Closed Principle (Example: Client&Server)



The class is:

- not open for extension, since we always use a concrete Server instance
- not closed for modification, because if we wish to change to another type of server, we must change the source code.

Open-Closed Principle (Example: Client&Server)



Client
«class»
+saveSomeData(data:String)

depends-on

AbstractServer
«abstract class»
+post(url:String, data:String)

extends

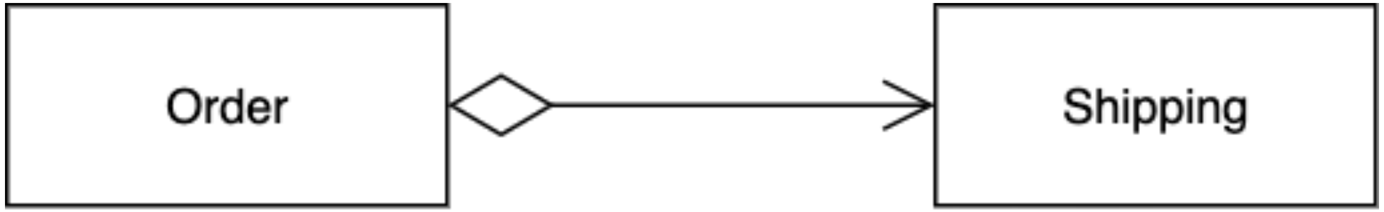
Server
«class»
+post(url:String, data:String)

Abstract Class in Java

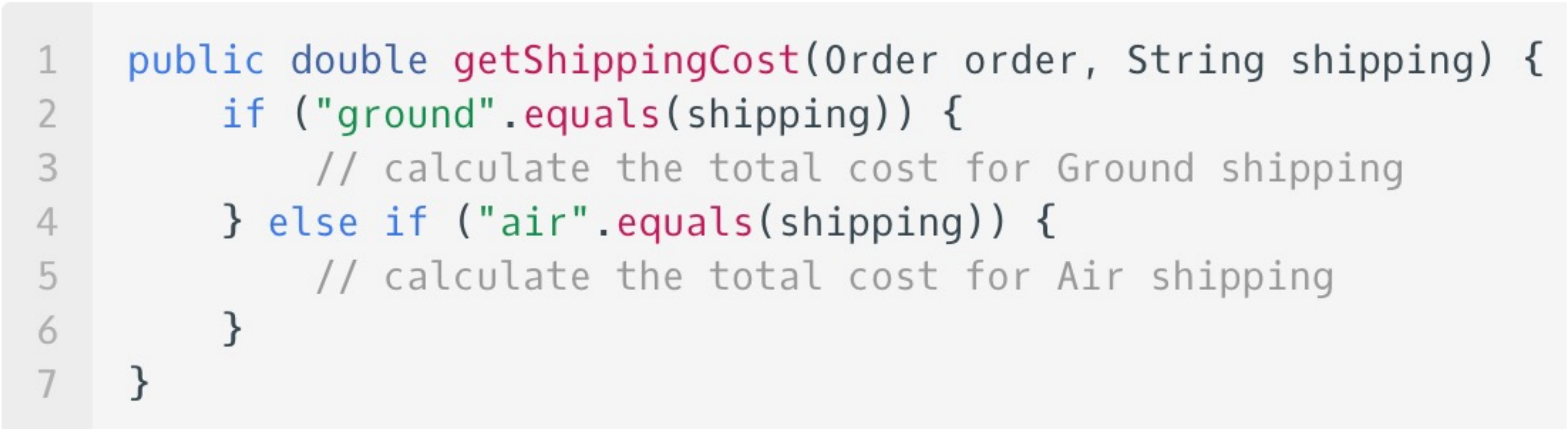
- An *abstract method* is a method that is declared without an implementation
- Consider using **abstract classes** if any of these statements apply to your situation:
  - You want to share code among several closely related classes.
  - You expect that classes that extend your abstract class have many common methods or fields, or require access modifiers other than public (such as protected and private).
  - You want to declare non-static or non-final fields. This enables you to define methods that can access and modify the state of the object to which they belong.
- Consider using **interfaces** if any of these statements apply to your situation:
  - You expect that unrelated classes would implement your interface. For example, the interfaces Comparable and Cloneable are implemented by many unrelated classes.
  - You want to specify the behavior of a particular data type, but not concerned about who implements its behavior.
  - You want to take advantage of multiple inheritance of type.

https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html

Open-Closed Principle (Example: Order&Shipping)

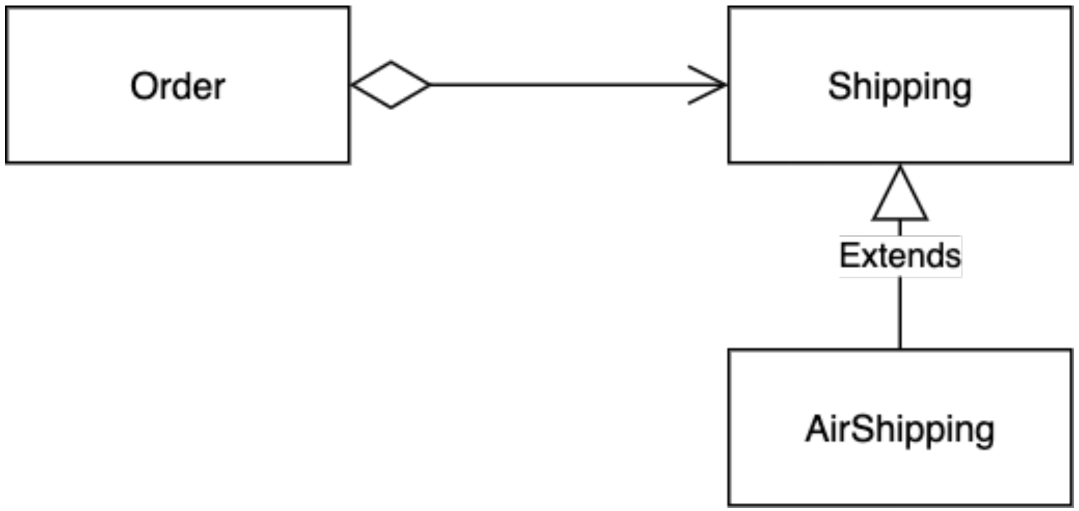


Ground shipping

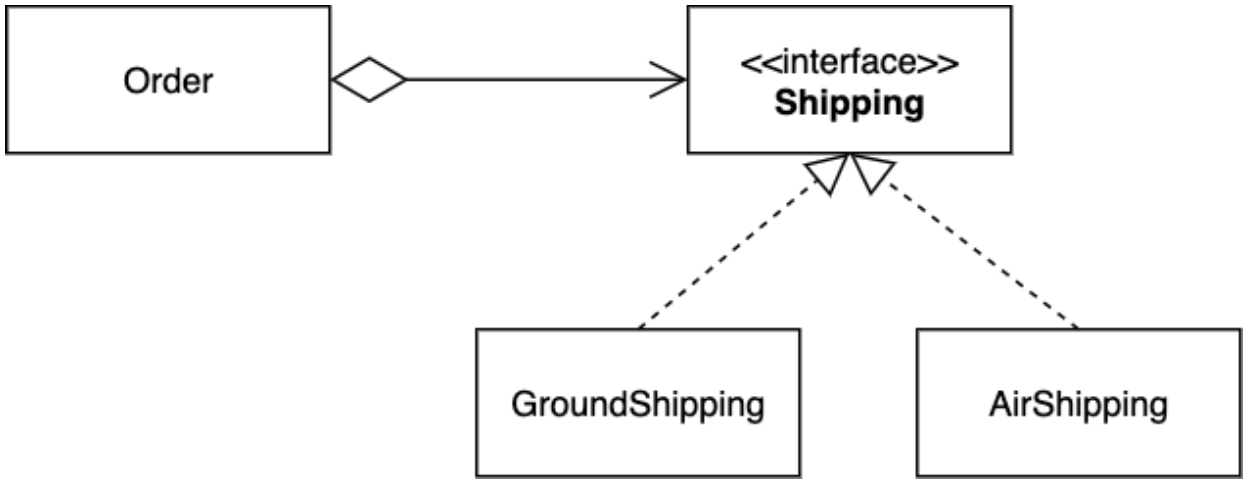


```java
public double getShippingCost(Order order, String shipping) {
    if ("ground".equals(shipping)) {
        // calculate the total cost for Ground shipping
    } else if ("air".equals(shipping)) {
        // calculate the total cost for Air shipping
    }
}
```

Open-Closed Principle (Example: Order&Shipping)



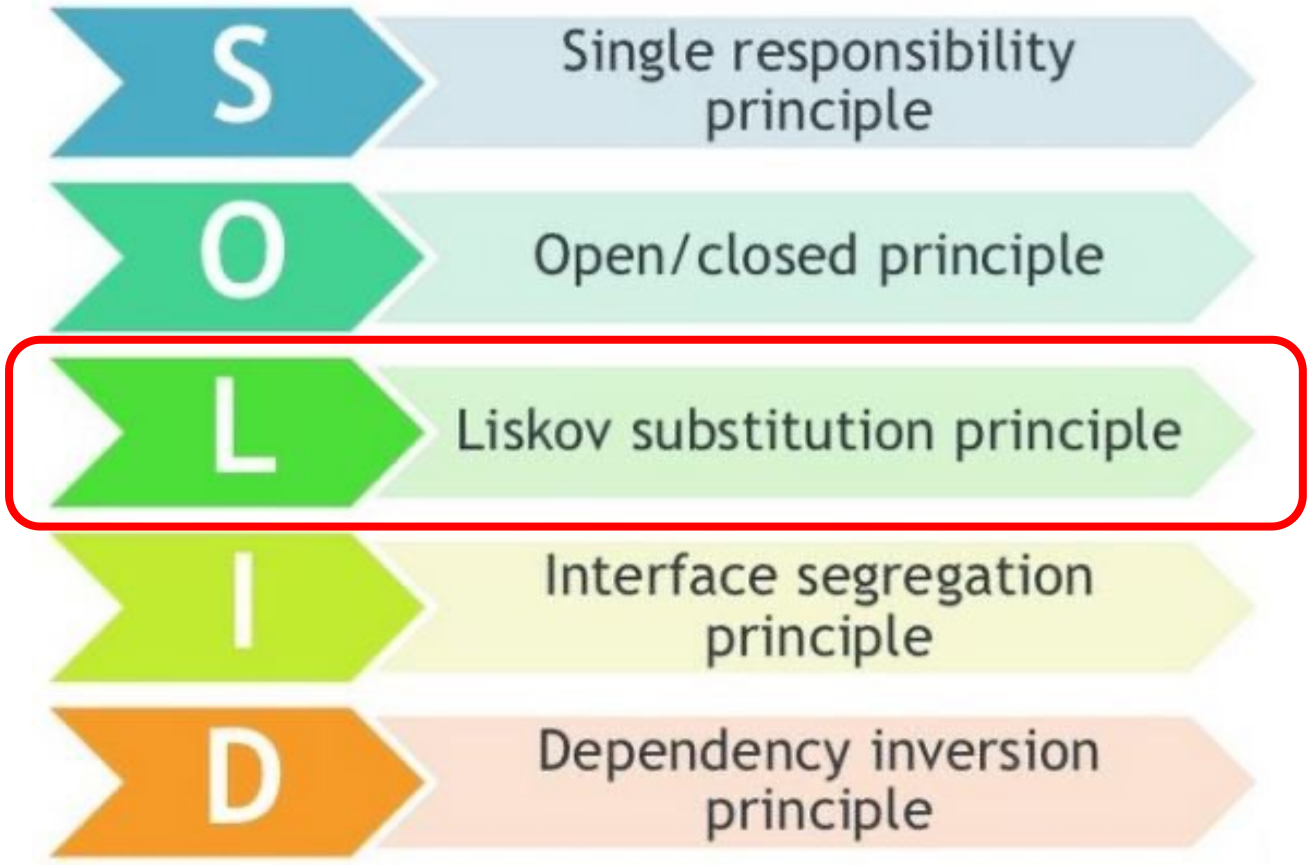
better



Thoughts? Critiques on OCP

- Adding un-needed flexibility to code (to make it open for extension) breeds complexity and carrying cost.
- It requires imagining all sorts of use-cases that don’t exist in order to make it ultimately flexible.
- Principle != you should always do this

OO Design Principles



S — Single responsibility principle

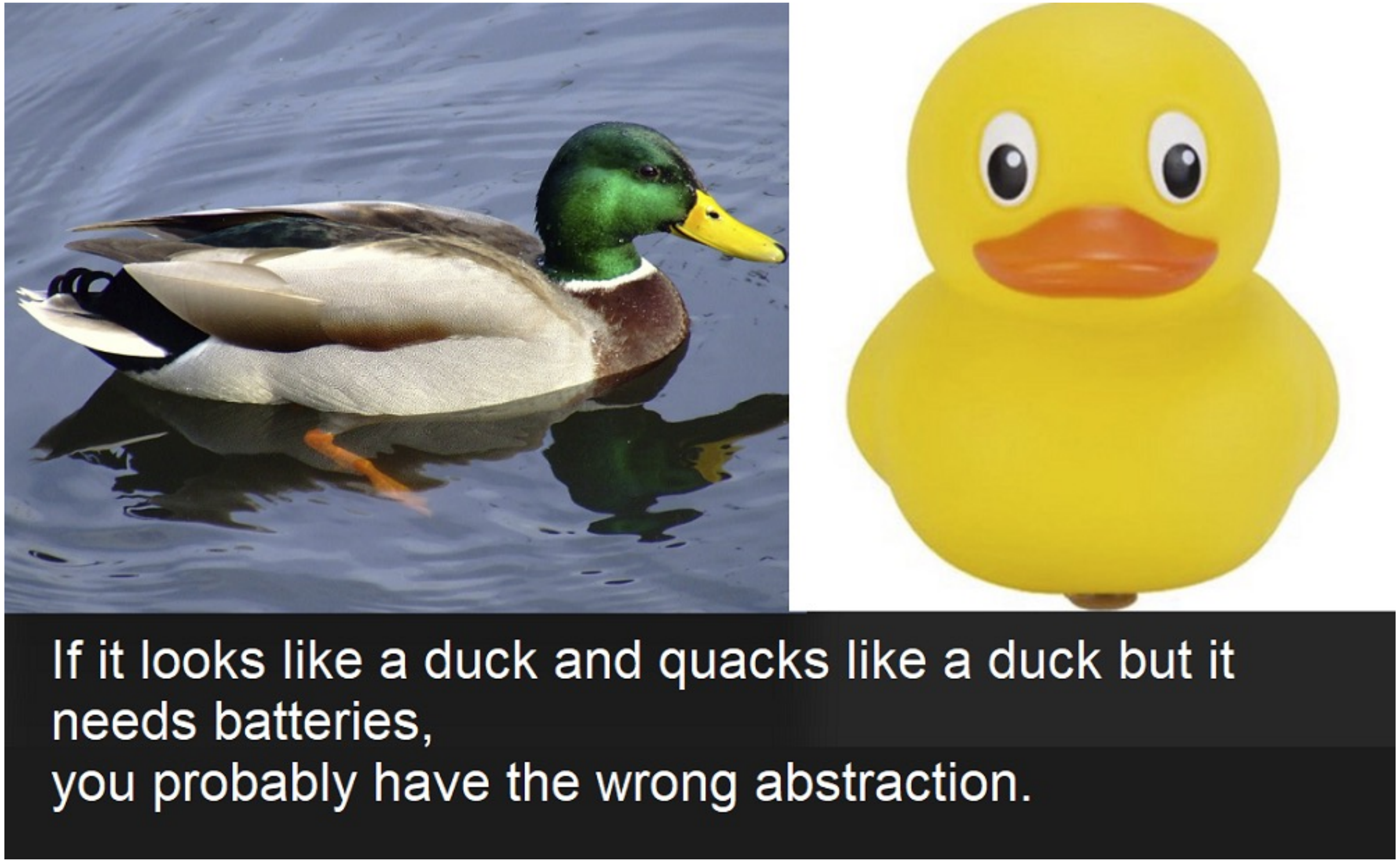
O — Open/closed principle

L — Liskov substitution principle

I — Interface segregation principle

D — Dependency inversion principle

Duck Tesk



If it looks like a duck and quacks like a duck but it needs batteries,
you probably have the wrong abstraction.

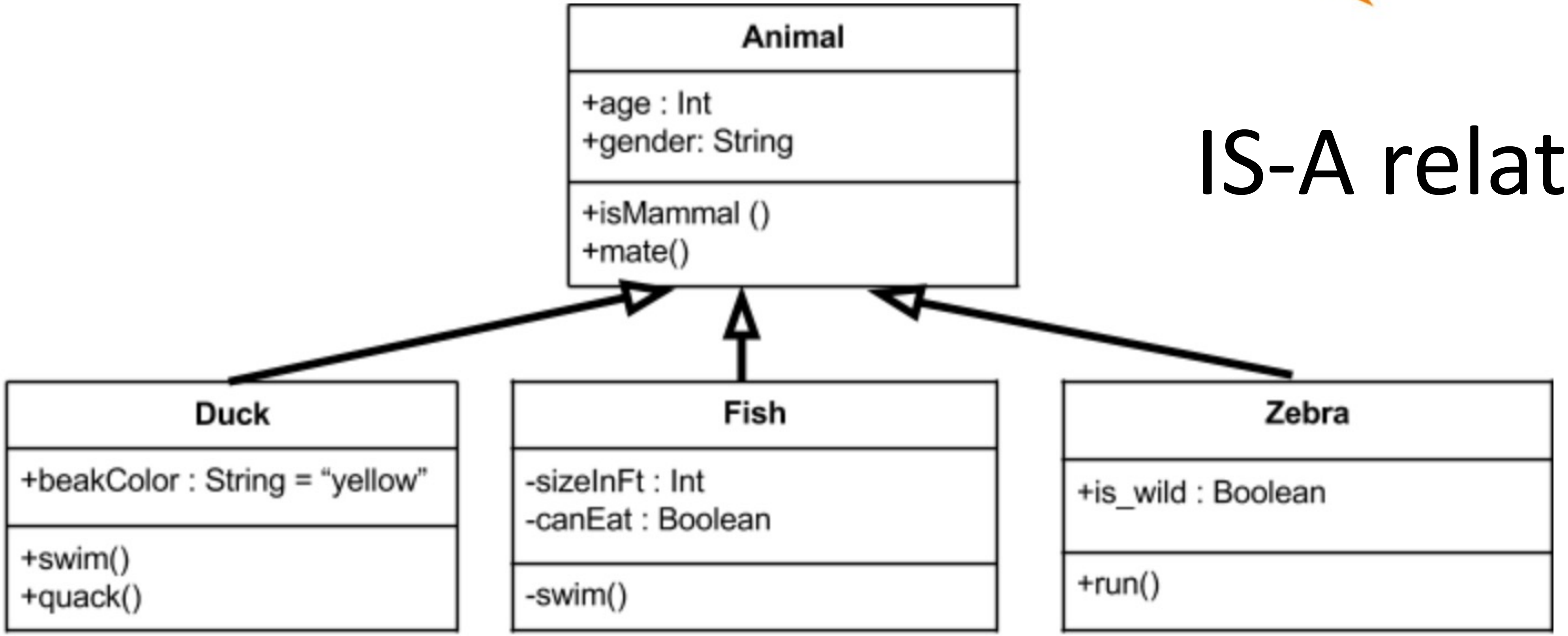
How can I send an SMS?
Wrong abstraction, bro!

Liskov Substitution Principle (LSP)

- The object of a derived class should be able to replace an object of the base class without bringing any errors in the system or modifying the behavior of the base class.
- **Benefit**: Code that adheres to LSP is loosely dependent to each other and encourages code reusability.

Inheritance

REMINDER



Animal

+age : Int
+gender: String

+isMammal ()
+mate()

Duck

+beakColor : String = “yellow”

+swim()
+quack()

Fish

-sizeInFt : Int
-canEat : Boolean

-swim()

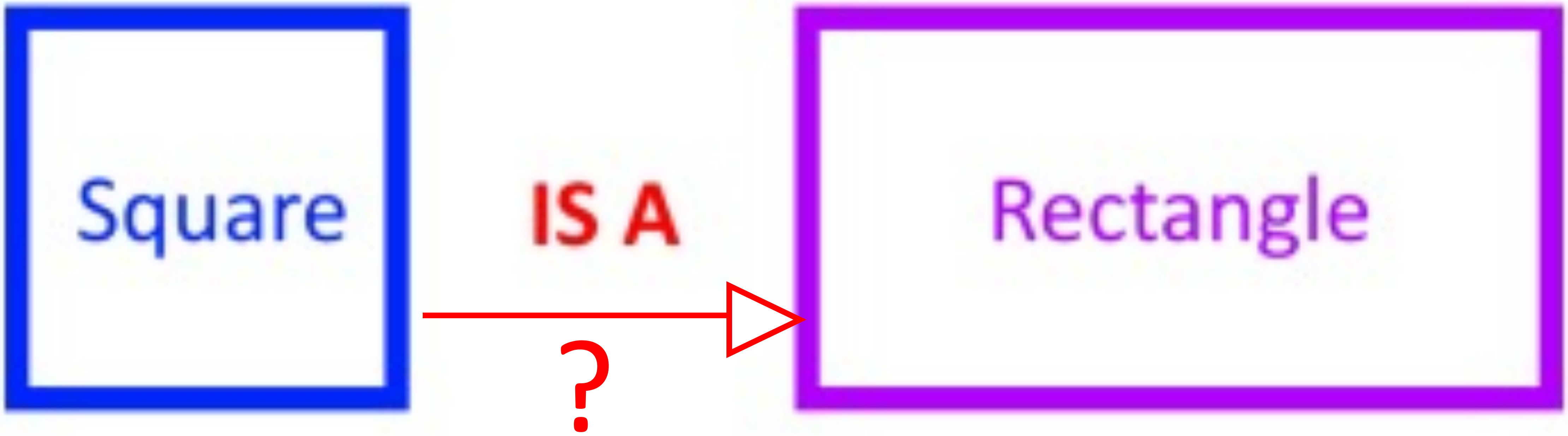
Zebra

+is_wild : Boolean

+run()

IS-A relationship

But...



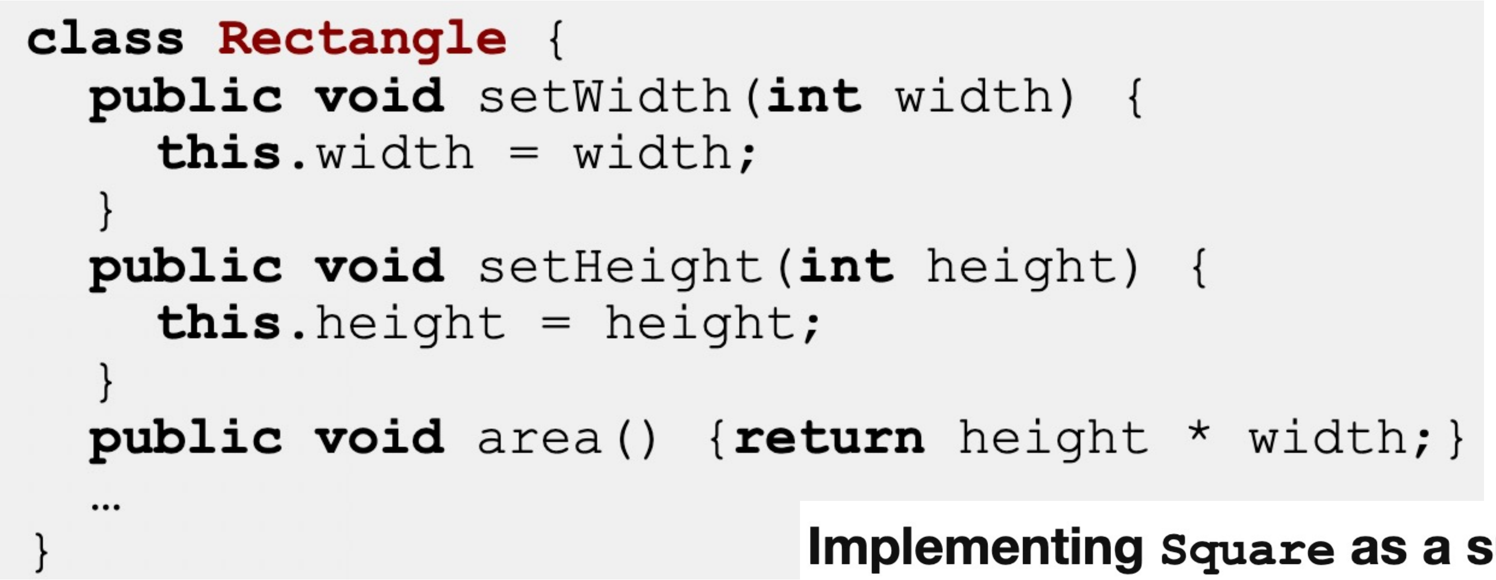
Square

IS A

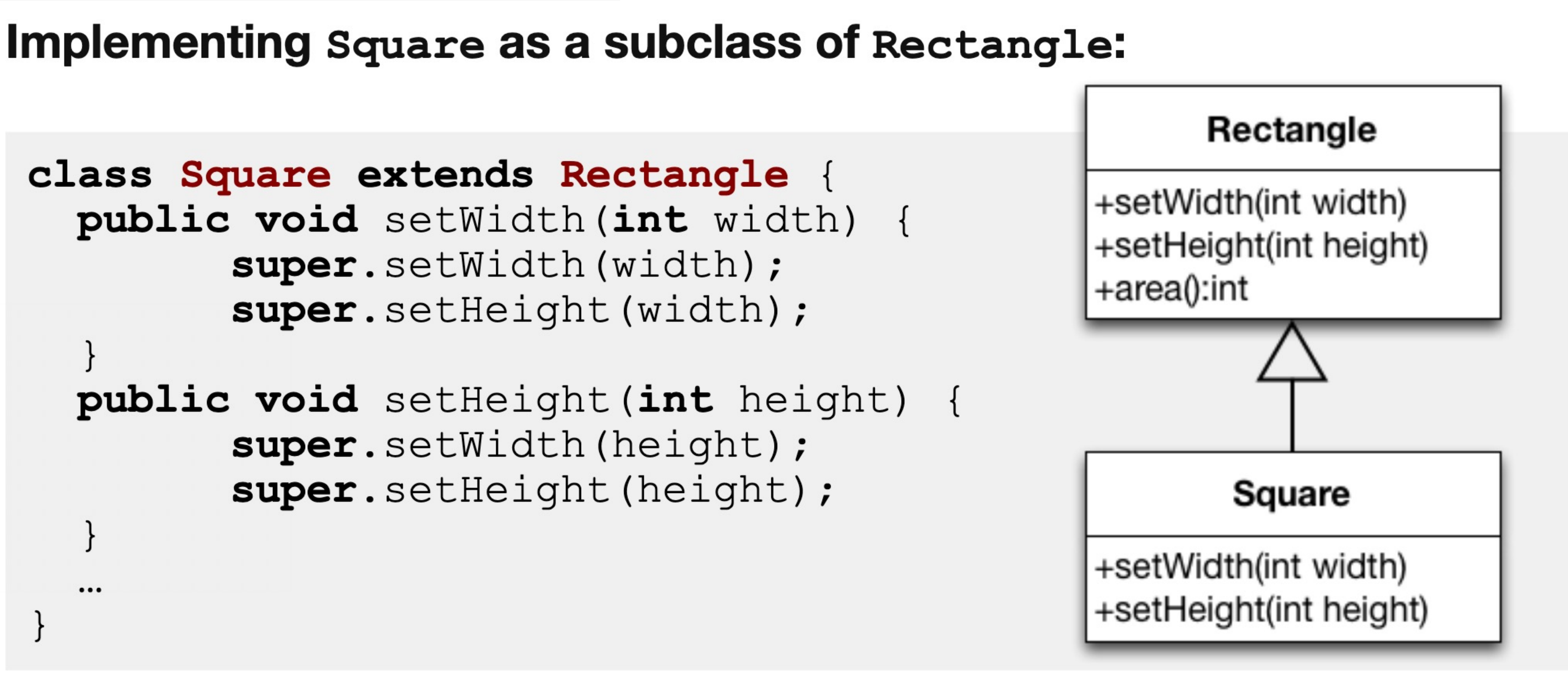
?

Rectangle

Violating the Liskov Substitution Principle



```
class Rectangle {
  public void setWidth(int width) {
    this.width = width;
  }
  public void setHeight(int height) {
    this.height = height;
  }
  public void area() {return height * width;}
  …
}
```

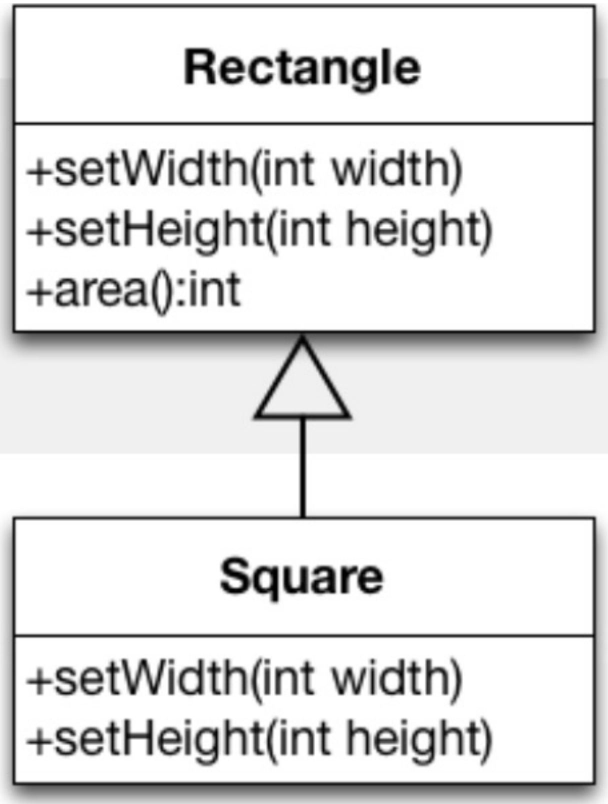


Implementing Square as a subclass of Rectangle:

```
class Square extends Rectangle {
  public void setWidth(int width) {
        super.setWidth(width);
        super.setHeight(width);
  }
  public void setHeight(int height) {
        super.setWidth(height);
        super.setHeight(height);
  }
  …
}
```

Violating the Liskov Substitution Principle

```
void clientMethod(Rectangle rec) {
  rec.setWidth(5);
  rec.setHeight(4);
  assert(rec.area() == 20);
}
```



Rectangle

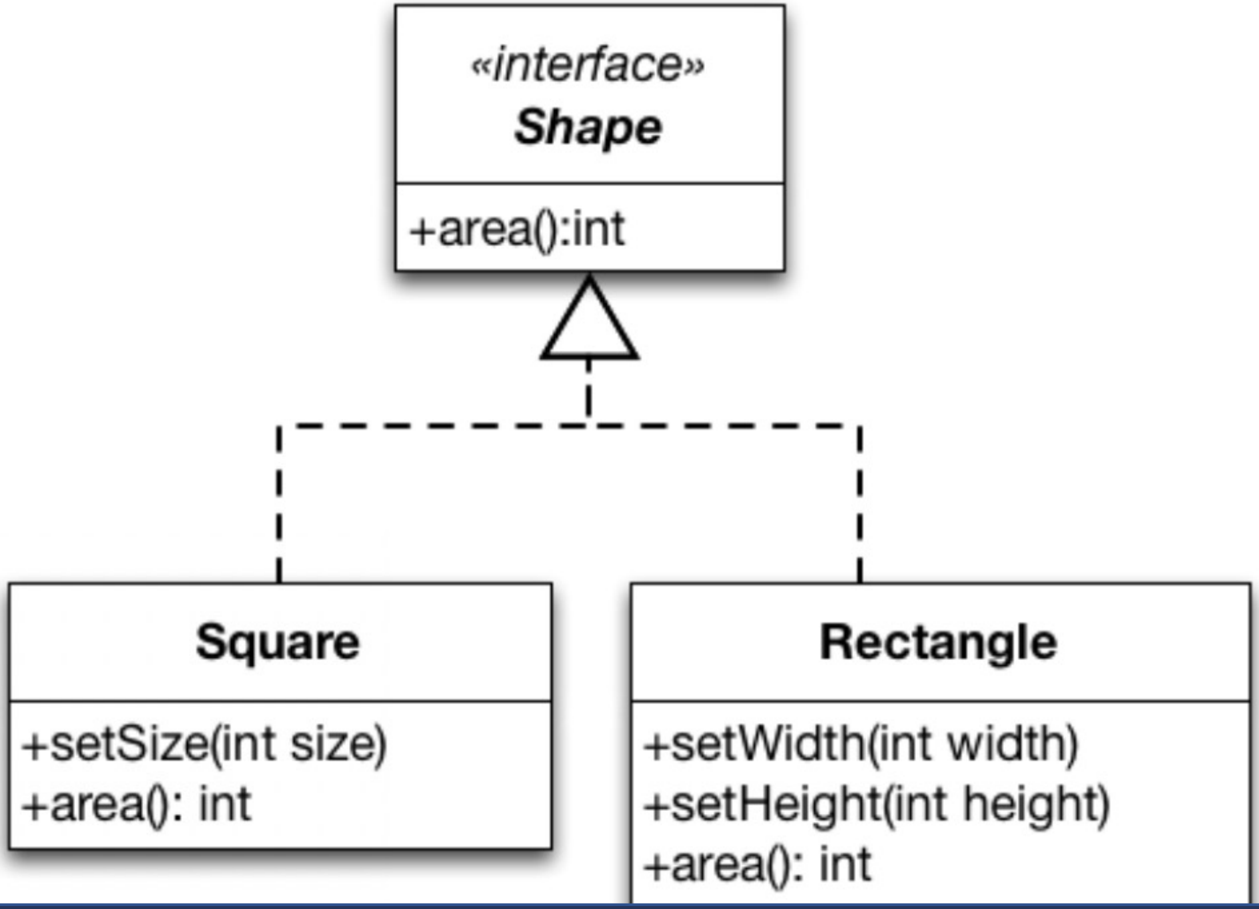
+setWidth(int width)
+setHeight(int height)
+area():int

Square

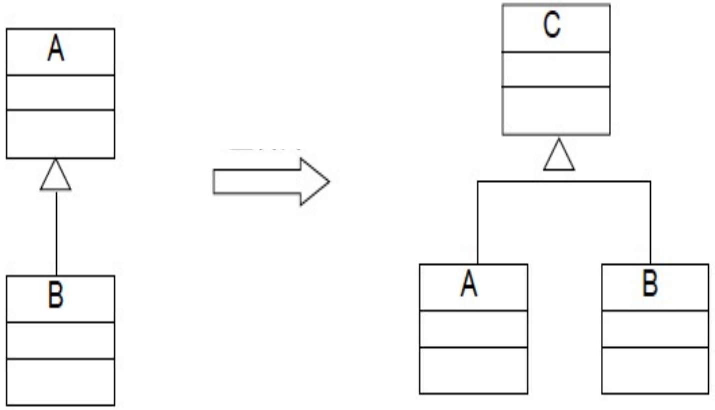
+setWidth(int width)
+setHeight(int height)

Liskov Substitution Principle (LSP)

- A LSP compliant solution
- Introduce the interface Shape



Solution



- To encapsulate what varies and to provide a generic interface we introduce an abstract Shape class.

Takeaway is to formulate the abstractions based on the logical structure of the code and not fall into the trap of letting real world relationships force its way into the design decisions of the application.

Disadvantages to violating the LSP

- Code that does not adhere to the LSP is tightly coupled and creates unnecessary entanglements.
- E.g. when a subclass can not substitue its parent class there would have to be multiple conditional statements to determine the class or type to handle certain cases differently.

Liskov Substitution Principle (LSP)

- Think twice before applying the IS-A trick
- Use polymorphism with great caution
- When writing an API first take the point of view of the client of your API
- Test-Driven Development (TDD), where client code must be written for test and design purposes before writing the code itself.

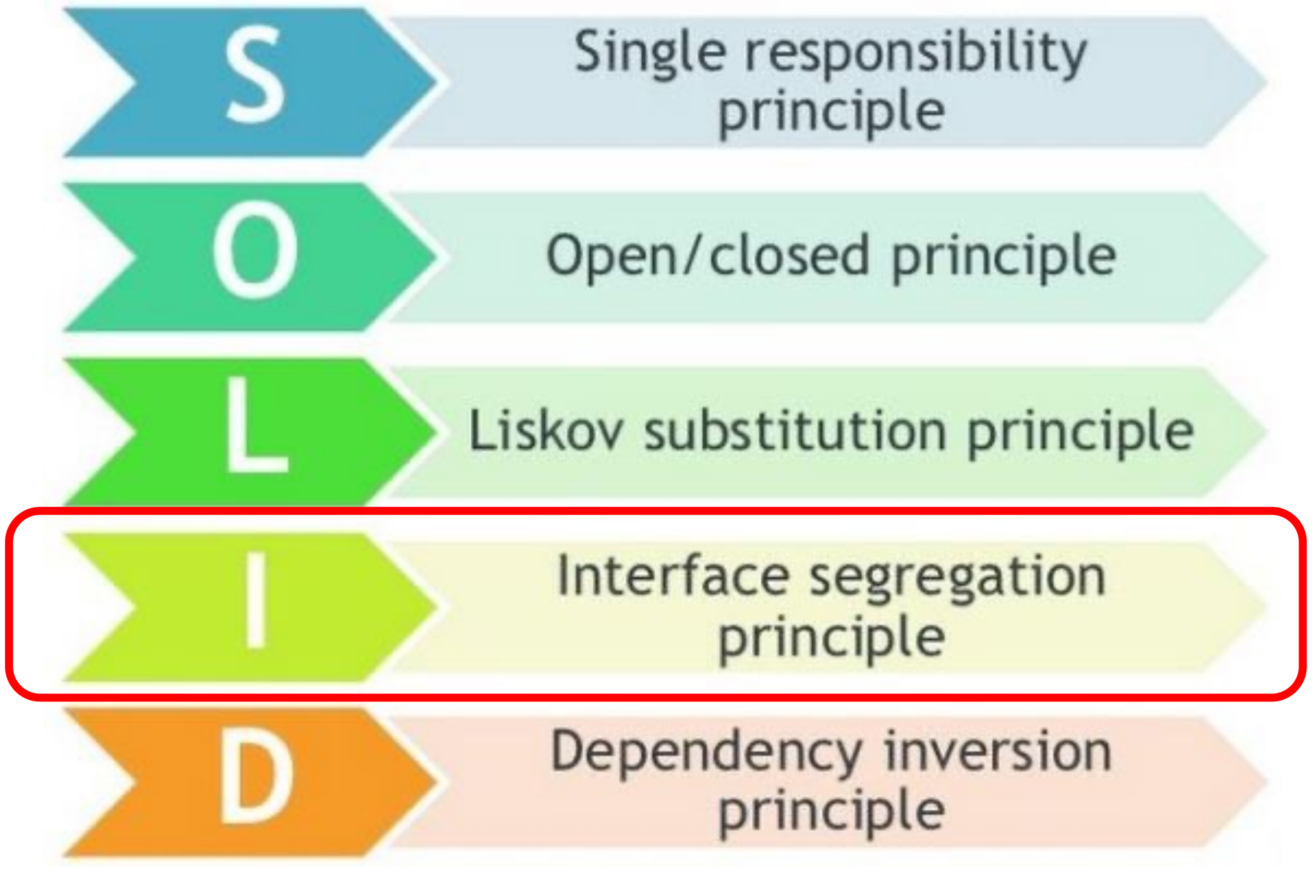
Do withdrawal applies to all bank account? What happen if we try to withdraw money from a locked long term deposit account?

Do really all birds can fly? What happen if I try to call *Fly()* on a bird that cannot fly?

Corresponding Design Patterns

- Strategy
- Composite
- Proxy

OO Design Principles



S Single responsibility principle

O Open/closed principle

L Liskov substitution principle

I Interface segregation principle

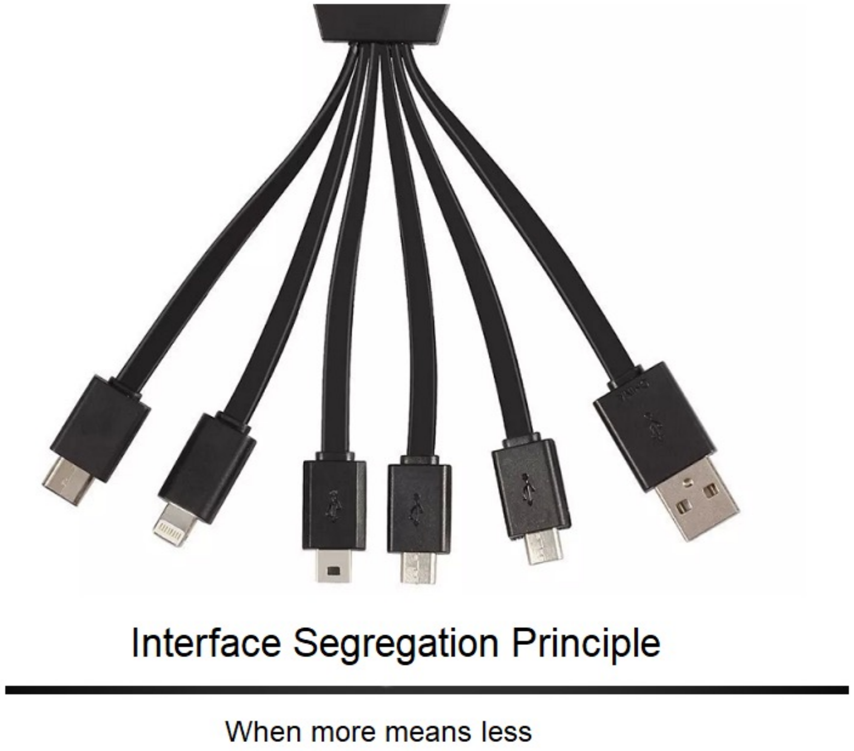
D Dependency inversion principle

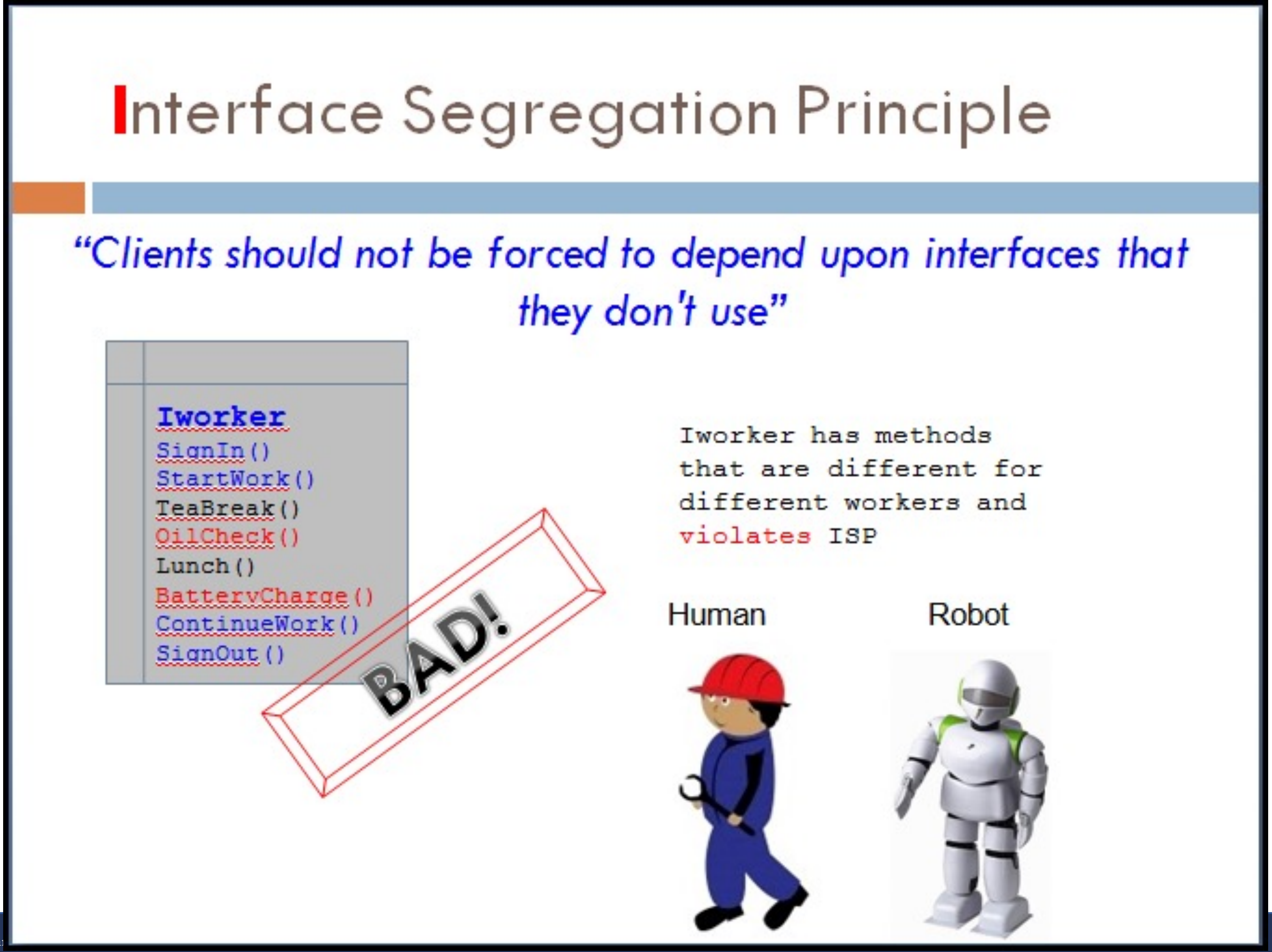
Interface Segregation Principle

When more means less

Interface Segregation Principle (ISP)

- No client should be forced to depend on methods it does not use.
- The goal of ISP is similar to Single Responsibility principle : to reduce the side effects and frequency of required changes by splitting the software into multiple, independent parts.
- "fat" interfaces!



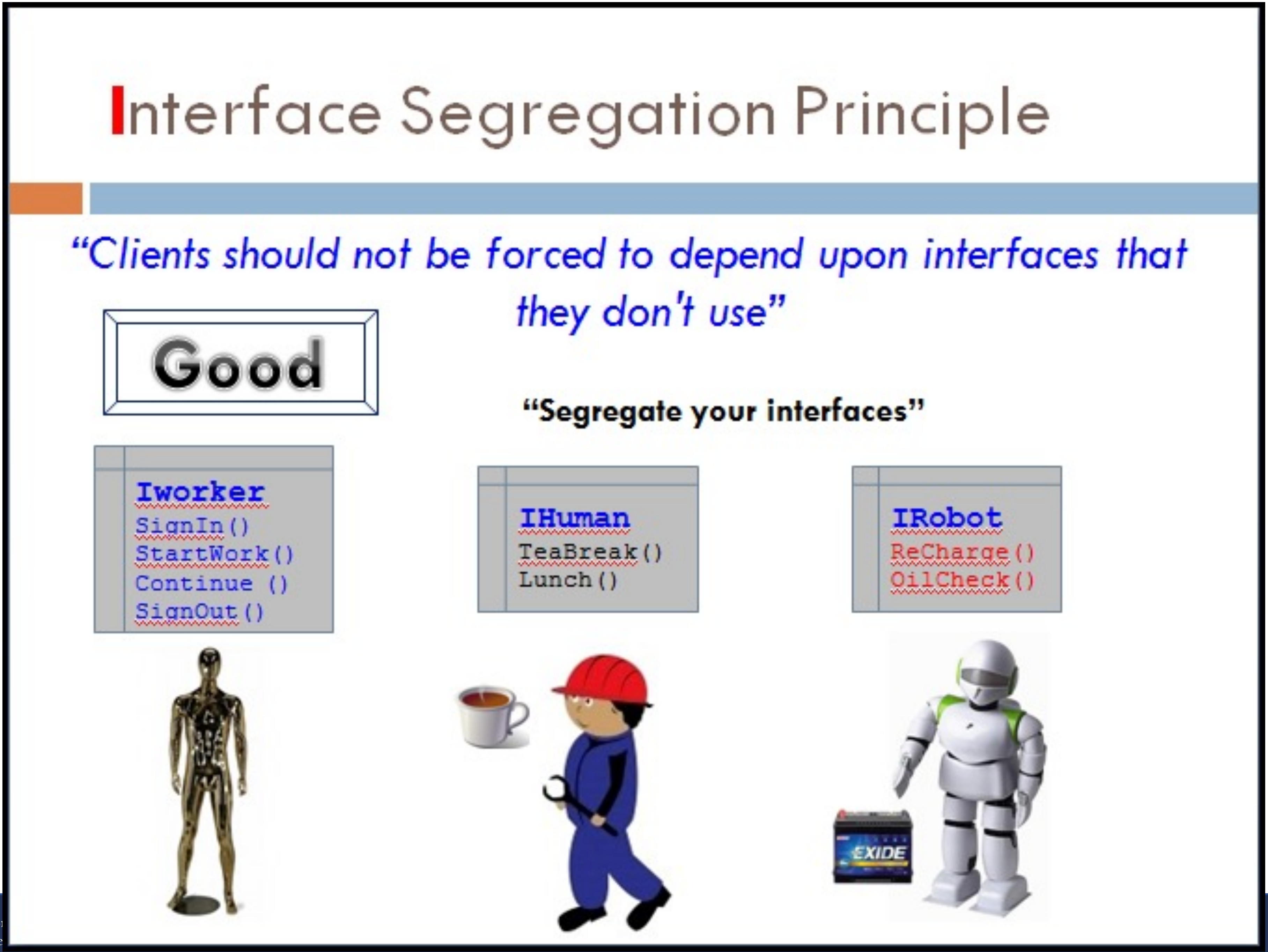


Interface Segregation Principle

“Clients should not be forced to depend upon interfaces that they don't use”

```
Iworker
SignIn()
StartWork()
TeaBreak()
OilCheck()
Lunch()
BatteryCharge()
ContinueWork()
SignOut()
```

Iworker has methods that are different for different workers and violates ISP



Interface Segregation Principle

“Clients should not be forced to depend upon interfaces that they don't use”

Good

“Segregate your interfaces”

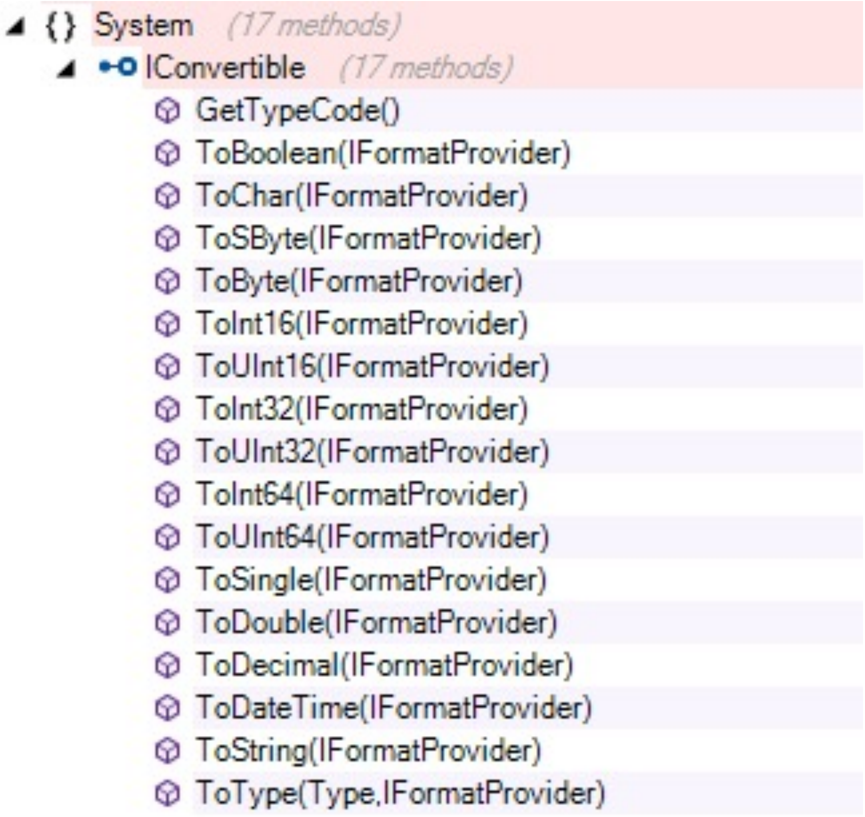
```
Iworker
SignIn()
StartWork()
Continue ()
SignOut()
```

```
IHuman
TeaBreak()
Lunch()
```

```
IRobot
ReCharge()
OilCheck()
```

Interface Segregation Principle (ISP)

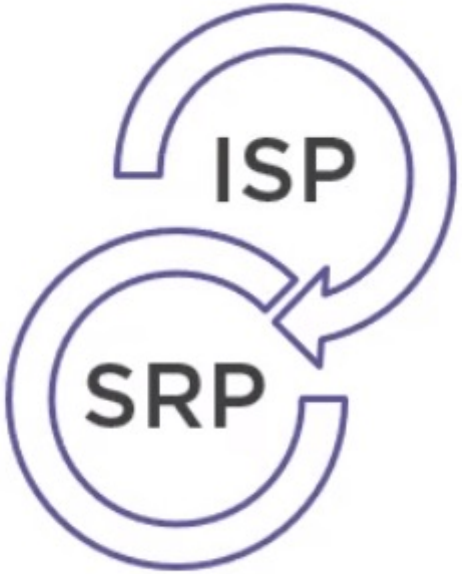
- A fat interface is not necessarily a design flaw



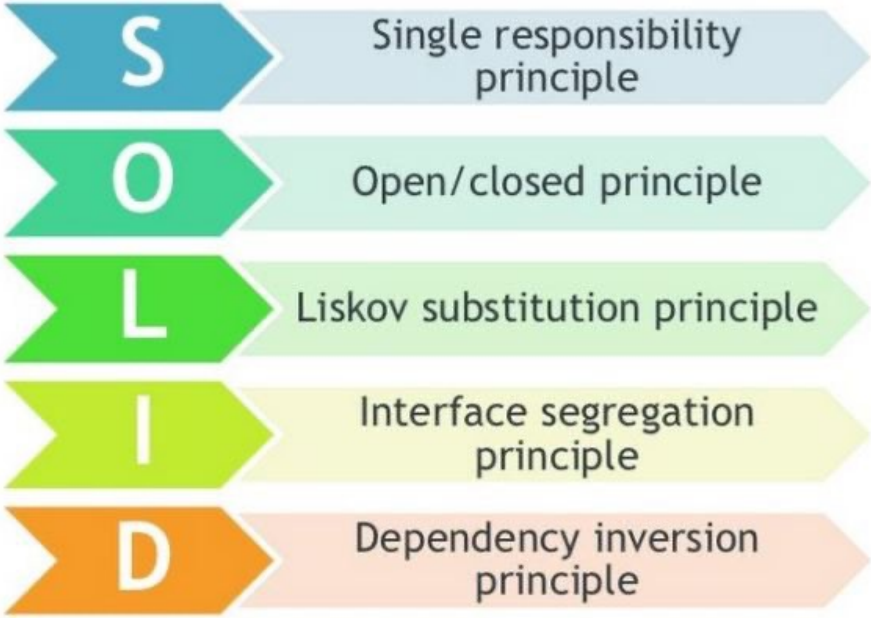
```
{} System (17 methods)
    IConvertible (17 methods)
        GetTypeCode()
        ToBoolean(IFormatProvider)
        ToChar(IFormatProvider)
        ToSByte(IFormatProvider)
        ToByte(IFormatProvider)
        ToInt16(IFormatProvider)
        ToUInt16(IFormatProvider)
        ToInt32(IFormatProvider)
        ToUInt32(IFormatProvider)
        ToInt64(IFormatProvider)
        ToUInt64(IFormatProvider)
        ToSingle(IFormatProvider)
        ToDouble(IFormatProvider)
        ToDecimal(IFormatProvider)
        ToDateTime(IFormatProvider)
        ToString(IFormatProvider)
        ToType(Type,IFormatProvider)
```

```
[SuppressMessage("NDepend",
"ND1200:AvoidInterfacesTooBig",
Justification="This interface is fat because it needs to
support all primitive types")]
public interface IConvertible {
   ...
```

https://www.ndepend.com/docs/suppress-issues?_ga=2.63469095.983202201.1601605450-1723910178.1601605450



Classes that implement small interfaces are more focused and tend to have a single purpose



S Single responsibility principle

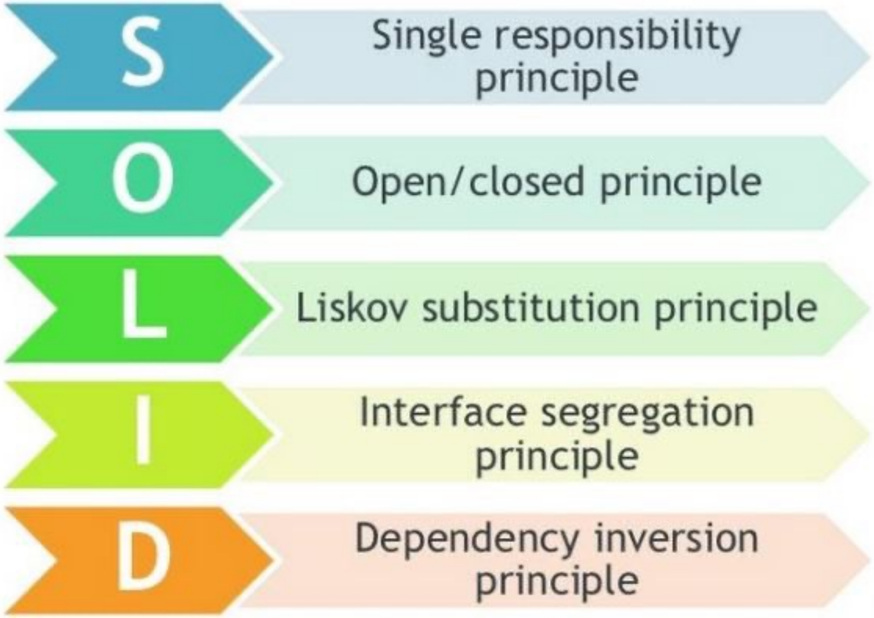
O Open/closed principle

L Liskov substitution principle

I Interface segregation principle

D Dependency inversion principle

By keeping interfaces small, the classes that implement them have a higher chance to fully substitute the interface

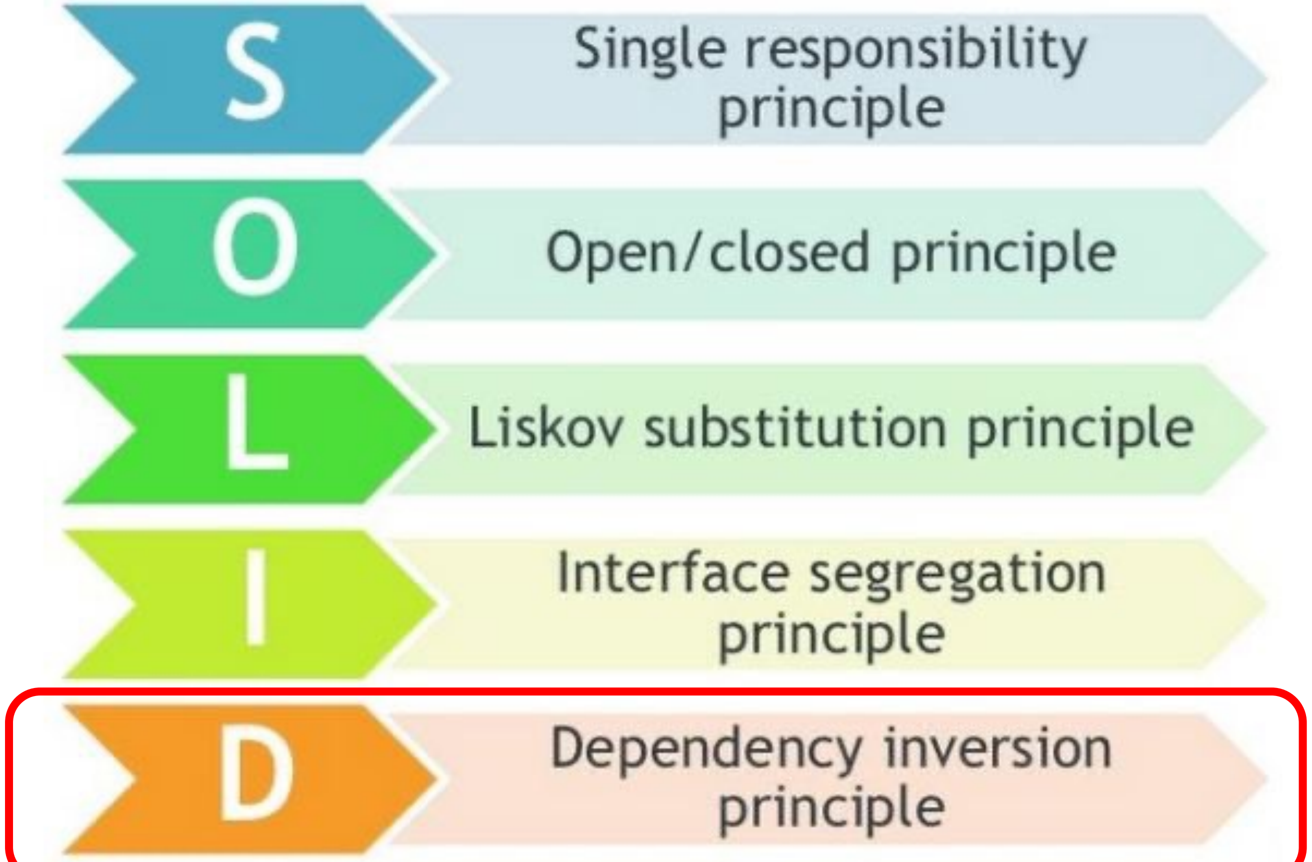


- **S** Single responsibility principle
- **O** Open/closed principle
- **L** Liskov substitution principle
- **I** Interface segregation principle
- **D** Dependency inversion principle

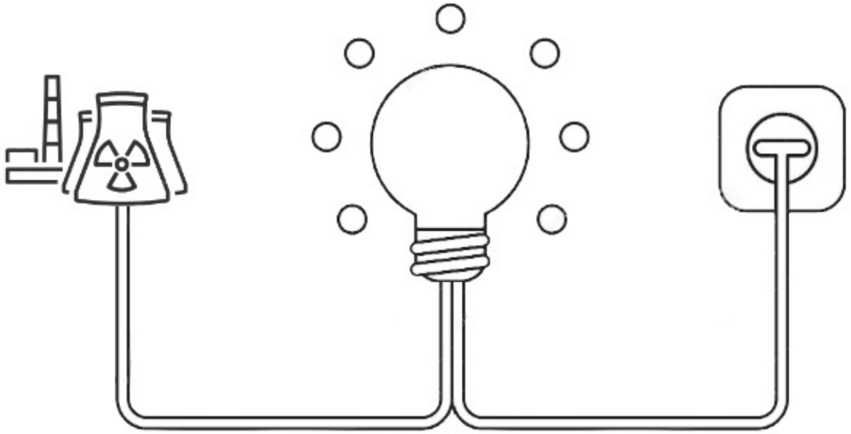
Corresponding Design Patterns

- Memento
- Iterator

OO Design Principles



- **S** Single responsibility principle
- **O** Open/closed principle
- **L** Liskov substitution principle
- **I** Interface segregation principle
- **D** Dependency inversion principle



Dependency Inversion Principle

When knowing how things work becomes a burden

Dependency Inversion Principle (DIP)

High-level modules, which provide complex logic, should be easily reusable and unaffected by changes in low-level modules, which provide utility features.

maintainability and *reusability*

SONY
Cyber-shot

Port doesn't define device

Dependency Inversion Principle (DIP)

- High-level modules should not depend on low-level modules. Both should depend on abstractions.
- Abstractions should not depend on details *(concrete implementation)*. Details should depend on abstractions.

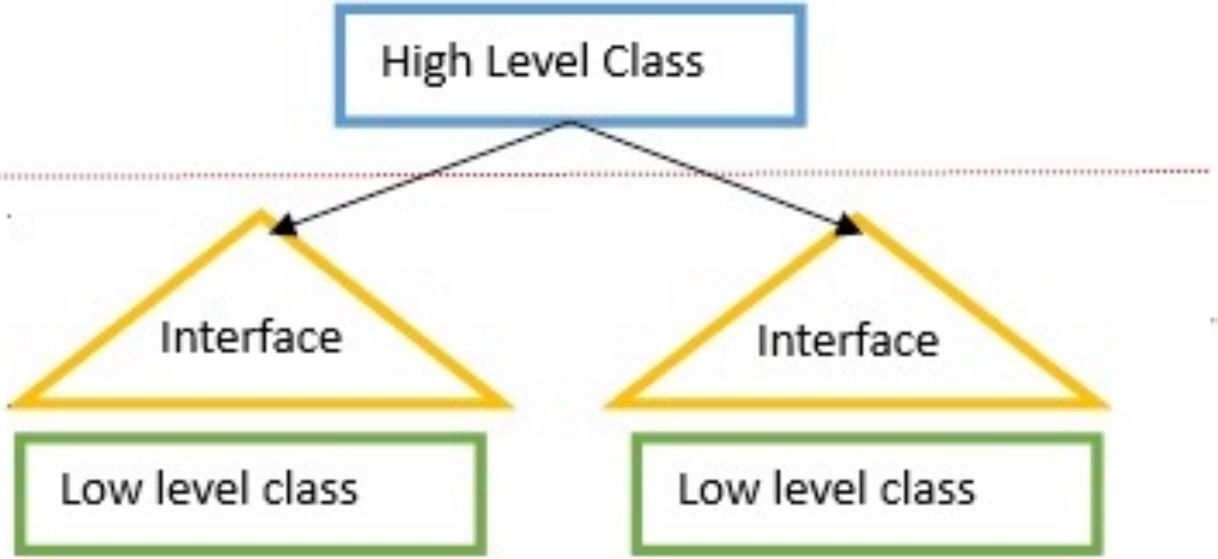


fig: When Dependency was not inverted

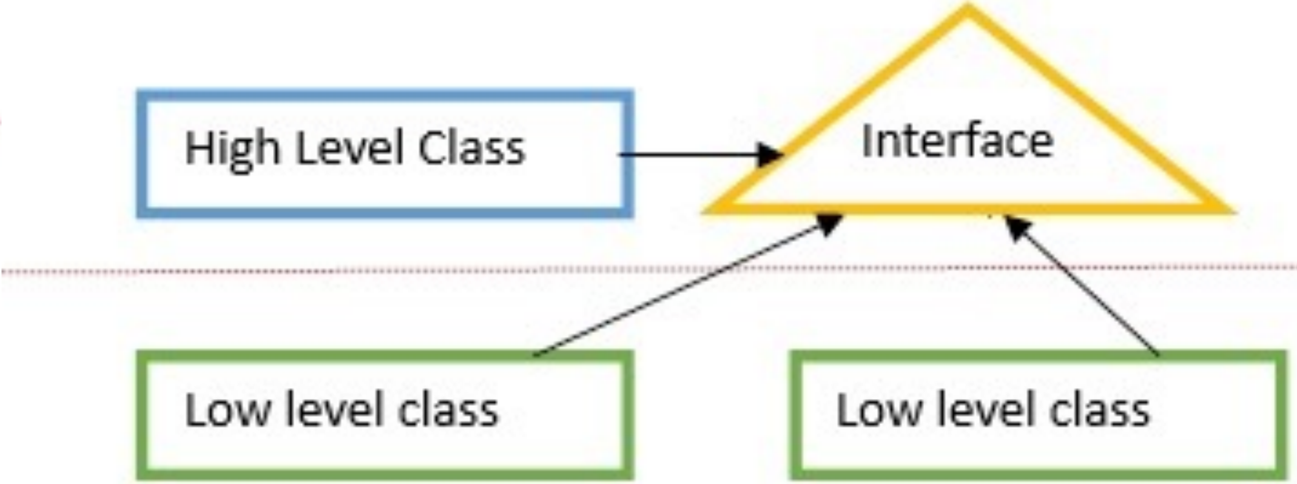


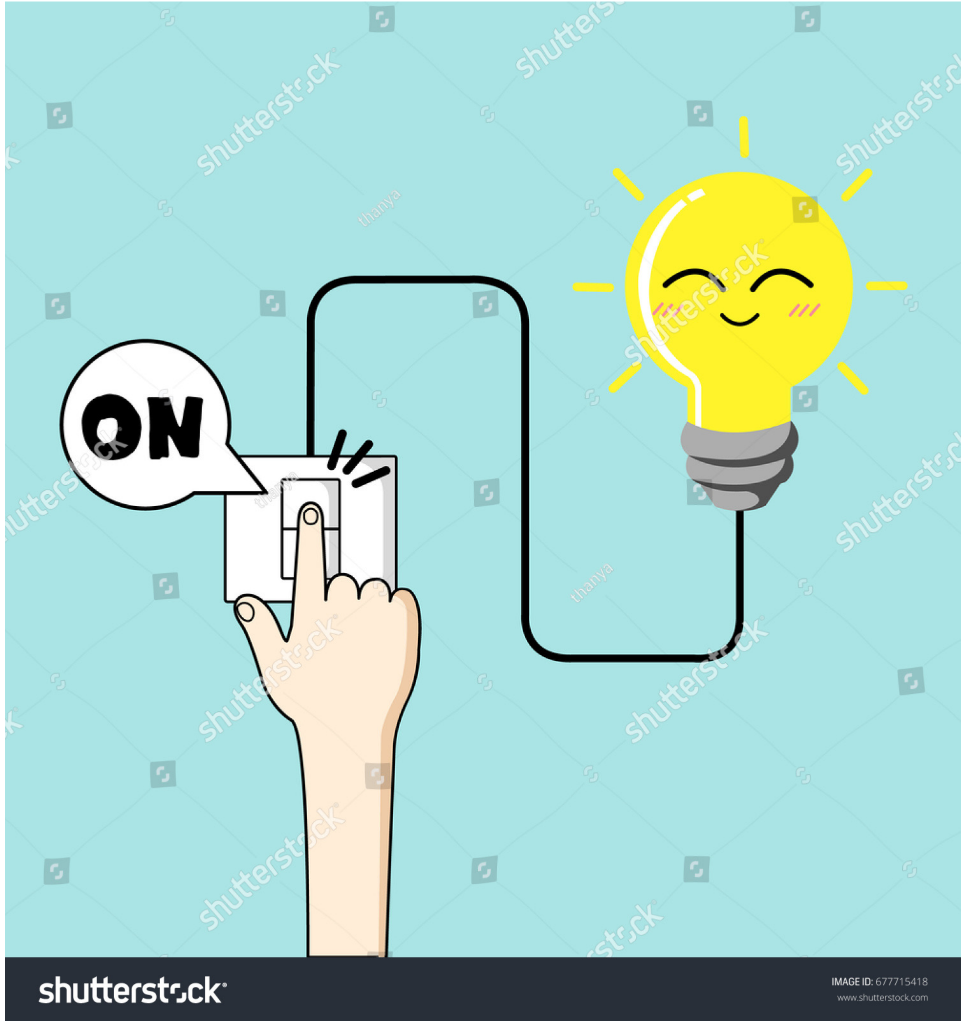
fig: Interface was defined by high level class

https://www.codeproject.com/Articles/538536/A-curry-of-Dependency-Inversion-Principle-DIP-Inve

Dependency Inversion Principle (DIP)

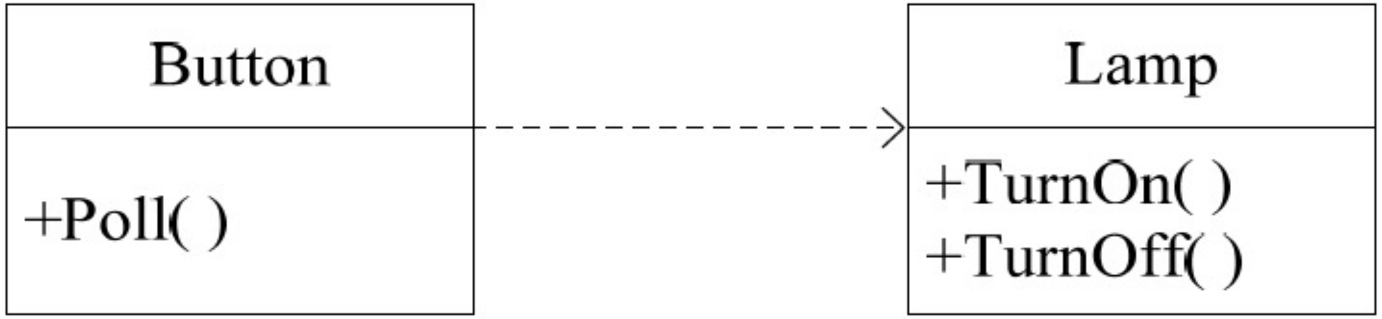
- A **High level module** is any module that contains the policy decisions and business model of an application.
- **Low level modules** are modules that contains detailed implementation that are required to execute the decisions and business policies.

Dependency Inversion Principle (DIP)



higher level module

lower level module



Button objects control Lamp objects and only Lamp objects.

Dependency Inversion Principle (DIP)

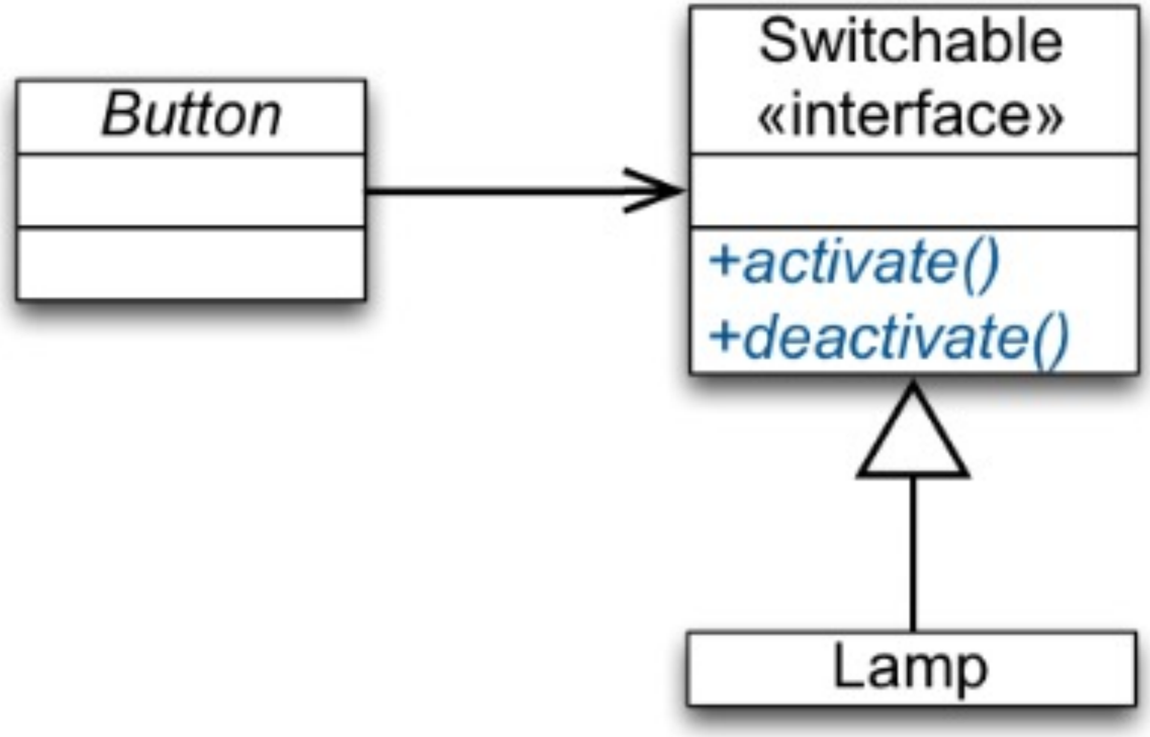
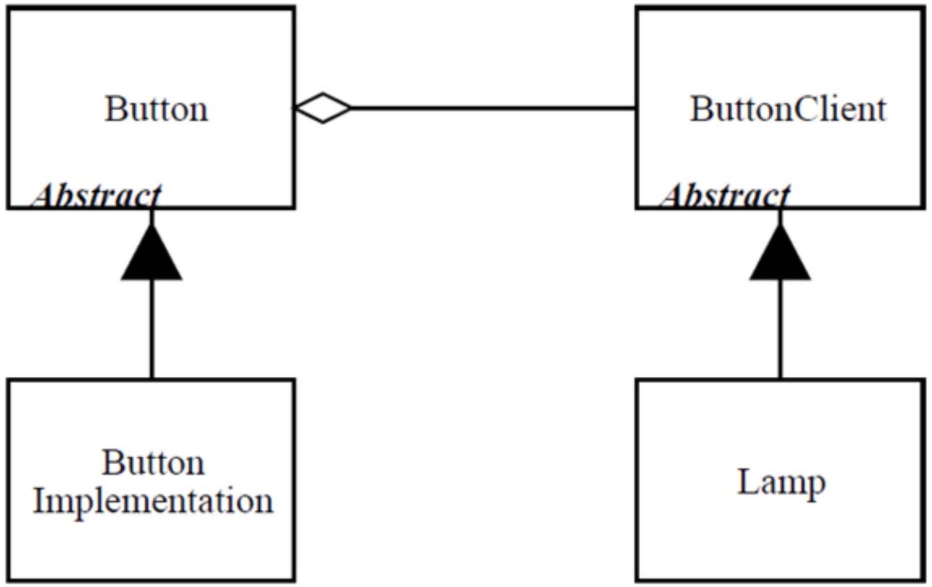


Figure 6: Inverted Button Model



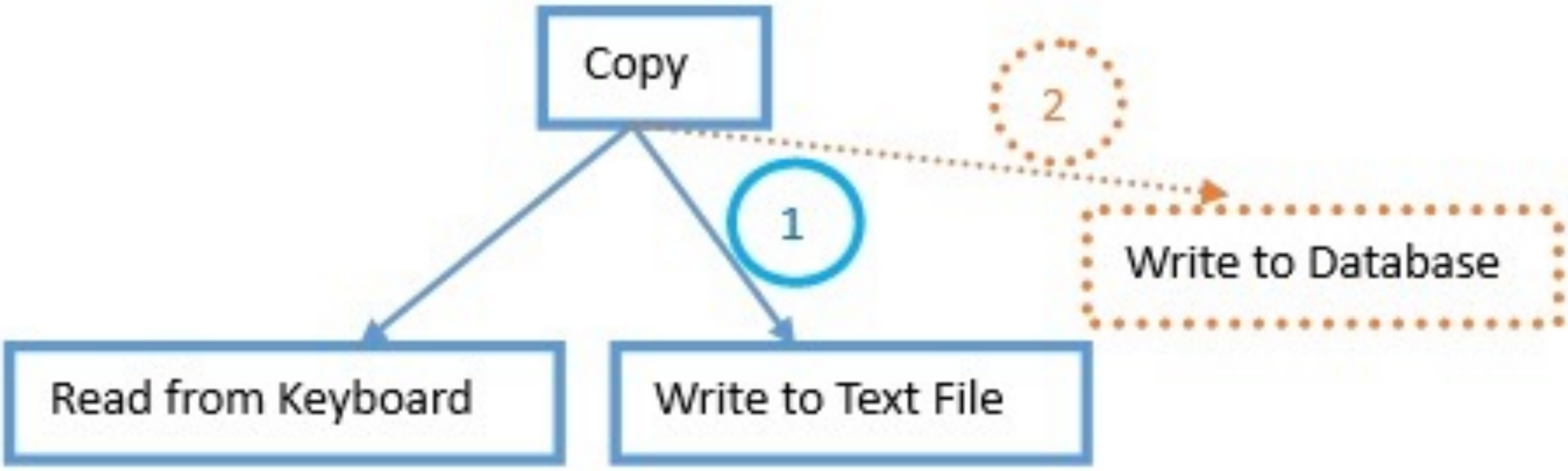


fig: Copy Program without maintaining DIP

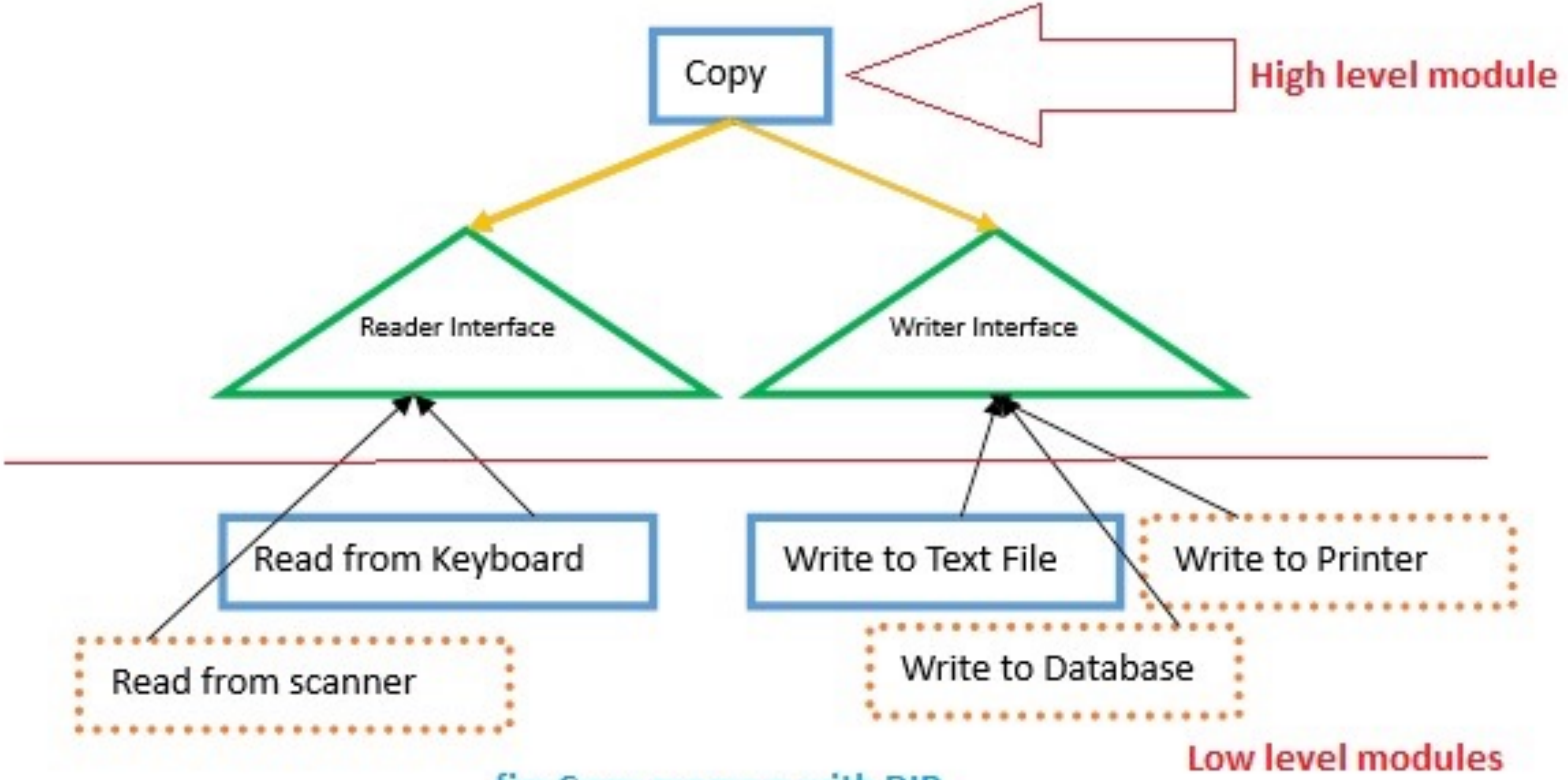


fig: Copy program with DIP

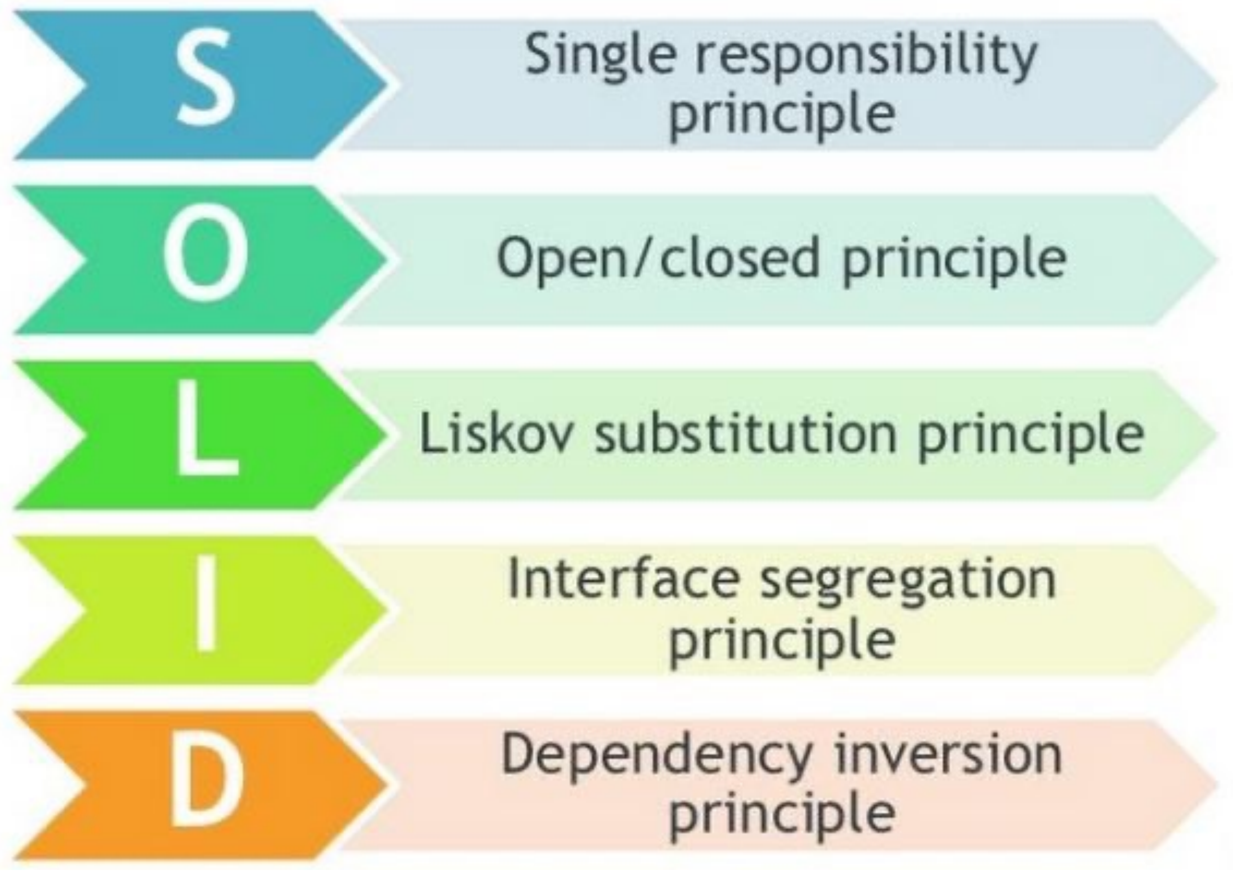
Problem if we DO NOT maintain DIP

- **System will rigid:** It will be difficult to change a part of the system without affection too many other parts of the system.
- **System will fragile:** When we will make a change, unexpected parts of the system will break.
- **System or component will be immobile:** It will be difficult to reuse it in another application because it cannot be disentangled from the current application.
- And so on……

Corresponding Design Patterns

- Factory Method
- Prototype
- Iterator

OO Design Principles



S Single responsibility principle

O Open/closed principle

L Liskov substitution principle

I Interface segregation principle

D Dependency inversion principle

Building stable and flexible systems

Cargo cult programming

Are SOLID principles Cargo Cult?

It looks like a plane, but will it fly?

https://blog.ndepend.com/are-solid-principles-cargo-cult/