

CEN 5016:
Software
Engineering

Fall 2024



University of
Central Florida

Dr. Kevin Moran

Week 7 - Class II:
Open Source
Software





- *Assignment 4*
 - Due Monday, October 7th
 - Exploring Static Analysis Tools and CI with a simple Python app
 - Accept the Assignment on GitHub Classroom
- *SDE Project Part 2*
 - Due Tuesday, October 15th (updated deadline!)
 - You should have already received feedback on your plan!
 - Two parts:
 - Process & Implementation Snapshot
 - Checkpoint Presentation

Midterm Exam Format



- 2 Parts, In-class exam, closed book, 200 points total
 - *Part 1:* Multiple Choice
 - 12-15 questions
 - Will test basic knowledge of concepts, select the best answer for each question
 - *Part 2:* Short Answer Questions
 - 4-5 questions
 - Concepts from class, SE scenarios, answer in a paragraph
- Covers material from Weeks 1-6
- You will have the **entire** class period to complete the exam
- Please bring your UCF ID to the exam

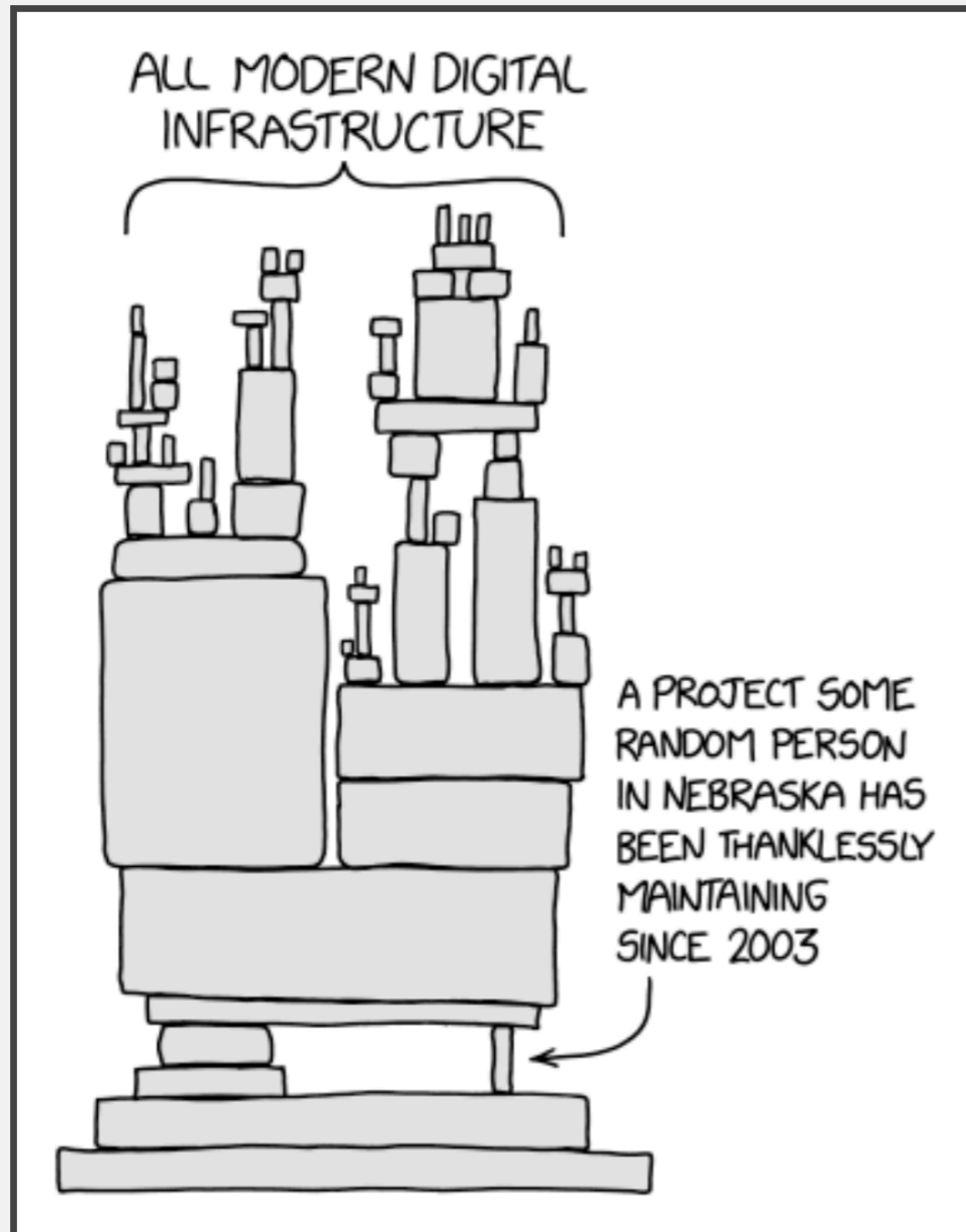
Open-Source Software





- Distinguish between open-source software, free software, and commercial software.
- Identify the common types of software licenses and their implications.
- Distinguish between copyright and intellectual property.
- Express an educated opinion on the philosophical/political debate between open source and proprietary principles.
- Describe how open-source ecosystems work and evolve, in terms of maintainers, community contribution, and commercial backing
- Identify various concerns of commercial entities in leveraging open-source, as well as strategies to mitigate these.

The Importance of Open-Source



Why did the commercial software get a promotion?

Because it knew how to "package" open-source code and sell it for a profit!

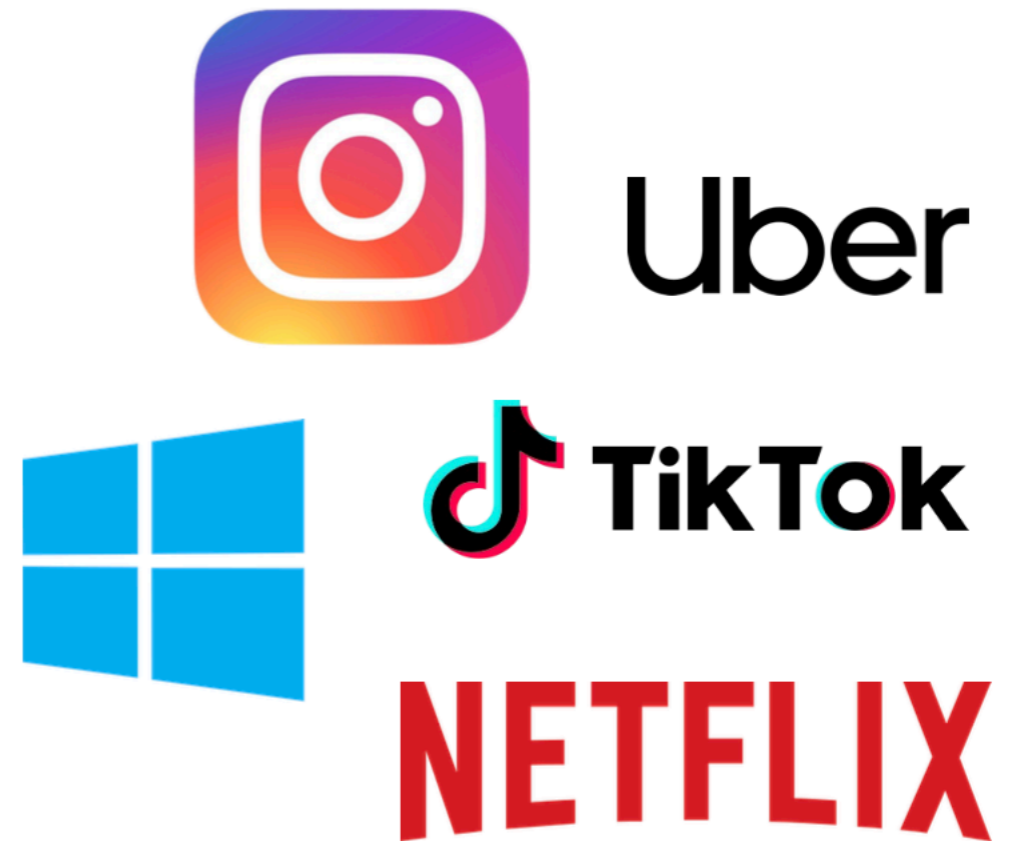
What is Open Source Software?



Open-source



Proprietary



What is Open Source Software?



- Source code availability
- Right to modify and create derivative works
- (Often) Right to redistribute derivative works



- **Early Days:** Roots trace back to the collaborative nature of software development in the 1950s and 60s.
- **GNU Project (1983):** Initiated by Richard Stallman to create a completely free operating system.
- **Open Source Initiative (1998):** Founded to promote open source software after Netscape released its browser source code.



Open Source: Focuses on the practical benefits of sharing code.

Free Software: Emphasizes ethical and moral aspects of software freedom.

Common Goal: Both aim to empower users and developers.



- **Quality**

- **Myth:** Lower quality than proprietary
- **Reality:** Often meets or exceed industry standards

- **Support and Maintenance**

- **Myth:** Lack of Professional support
- **Reality:** Robust support community

- **Security**

- **Myth:** Less secure because code is public
- **Reality:** Transparency allows quicker identification and fixing



- **For Individuals**
 - Learning opportunities
 - Customization
 - Cost Saving
- **For Business**
 - Flexibility
 - Security
 - Community Support



- **Copyleft:** Requires derivatives to maintain the same license (e.g., GPL)
- **Permissive:** Allows proprietary use of modified code (e.g., MIT, Apache)



Type: Copyleft

Key Terms:

- Any derivative work must be distributed with the same GPL license.
- The source code must be made available to users, enabling them to modify and redistribute it.
- Commercial use is allowed, but any distributed version of the software (including commercial ones) must adhere to the GPL terms.



Type: Permissive

Key Terms:

- The software can be used for personal, commercial, or open-source purposes.
- There's no requirement to release derivative works as open source.
- The original copyright notice and license must be included in all copies or substantial portions of the software.
- No warranties or liability are provided by the authors of the software.



Type: Permissive with additional patent rights

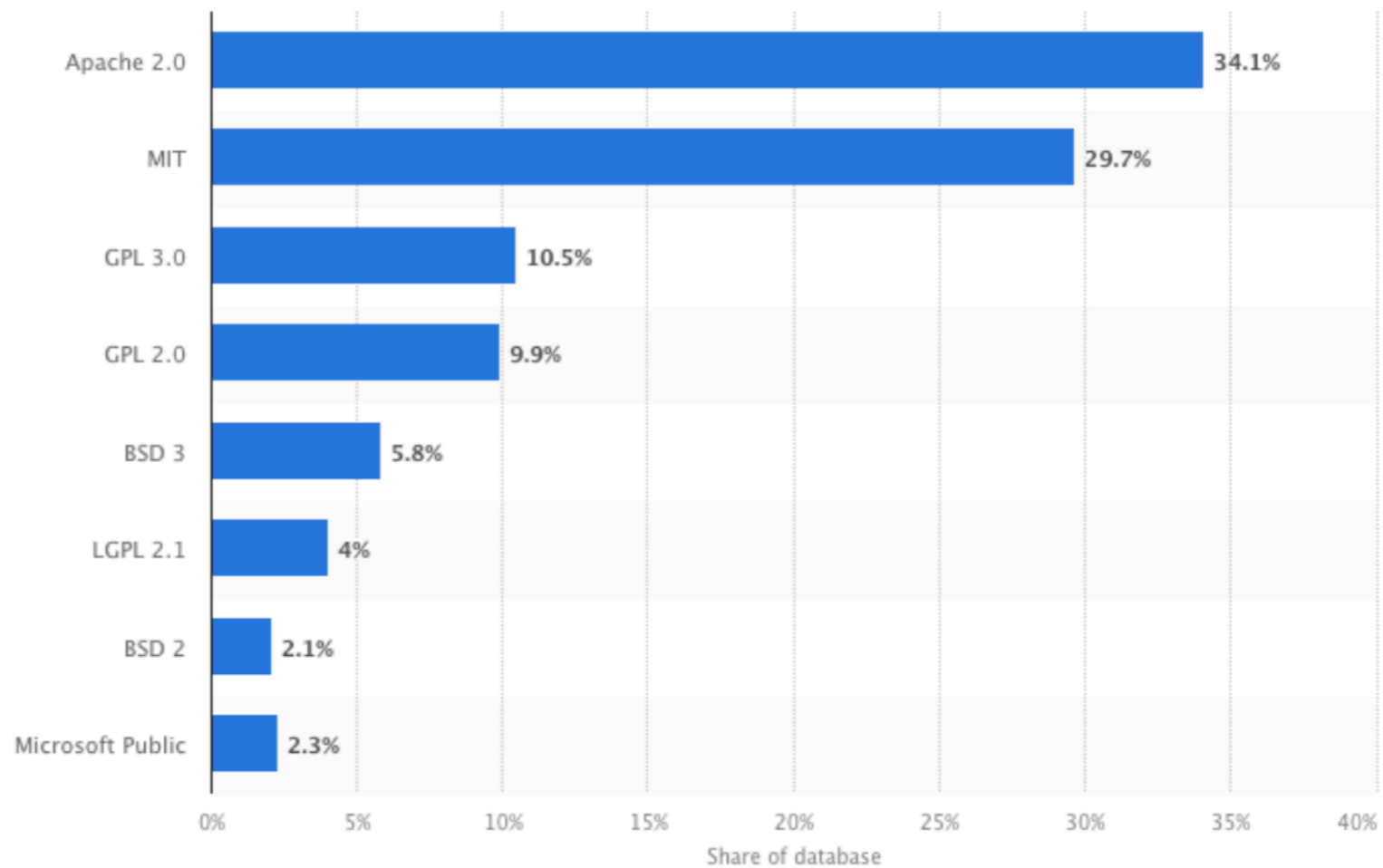
Key Terms:

- Users can use the software for both open-source and proprietary purposes.
- The license includes an express grant of patent rights, ensuring that contributors cannot sue users for patent infringement related to their contributions.
- Modifications to the original software must be clearly marked.
- The original copyright notice and license must be included in any derivative works.

Most popular Software Licenses



Most popular open source licenses worldwide in 2021



© Statista 2023

[Show source](#)

[Additional Information](#)

Which License to Choose?




choosealicense.com

Choose an open source license


An open source license protects contributors and users. Businesses and savvy developers won't touch a project without this protection.

{ Which of the following best describes your situation? }


I need to work in a community.


Use the [license preferred by the community](#) you're contributing to or depending on. Your project will fit right in.

If you have a dependency that doesn't have a license, ask its maintainers to [add a license](#).


I want it simple and permissive.

The [MIT License](#) is short and to the point. It lets people do almost anything they want with your project, like making and distributing closed source versions.

[Babel](#), [.NET](#), and [Rails](#) use the MIT License.


I care about sharing improvements.

The [GNU GPLv3](#) also lets people do almost anything they want with your project, *except* distributing closed source versions.

[Ansible](#), [Bash](#), and [GIMP](#) use the GNU GPLv3.

{ What if none of these work for me? }

My project isn't software.

[There are licenses for that.](#)

I want more choices.

[More licenses are available.](#)

I don't want to choose a license.

[Here's what happens if you don't.](#)



- Nobody should be restricted by the software they use. There are four freedoms that every user should have:
 - the freedom to use the software for any purpose,
 - the freedom to change the software to suit your needs,
 - the freedom to share the software with your friends and neighbors, and
 - the freedom to share the changes you make.
- Code must be made available
- Any modifications must be relicensed under the same license (copyleft)



- Nobody should be restricted by the software they use. There are four freedoms that every user should have:
 - the freedom to use the software for any purpose,
 - the freedom to change the software to suit your needs,
 - the freedom to share the software with your friends and neighbors, and
 - the freedom to share the changes you make.
- Code must be made available
- Any modifications must be relicensed under the same license (copyleft)



- Software must be a library
- Similar to GPL but does not consider dynamic binding as “derivative work”
- So, proprietary code can depend on LGPL libraries as long as they are not being modified
- See also: GPL with classpath exception (e.g., Oracle JDK)



- 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



Collaboration and Community

- **Distributed Teams:** Developers contribute from around the world.
- **Communication Tools:** Use of mailing lists, forums, and version control systems.

Project Management

- **Governance Structures:** Maintainers, committers, and contributors.
- **Decision-Making:** Often merit-based and consensus-driven.

Contribution Guidelines

- **Code of Conduct:** Sets expectations for participant behavior.
- **Contribution Process:** Guidelines for submitting code, reporting issues, and documentation.



Contributing to Projects

- **Identify Interests:** Choose projects aligned with your skills or passions.
- **Start Small:** Begin with documentation, bug fixes, or feature requests.
- **Engage with the Community:** Participate in discussions and follow project guidelines.

Starting Your Own Project

- **Planning:** Define the project's purpose and goals.
- **Licensing:** Select an appropriate open source license.
- **Promotion:** Build a community through outreach and collaboration.

Participating in Communities

- **Events and Conferences:** Attend or speak at open source events.
- **Online Platforms:** Join forums, mailing lists, and social media groups.



Platforms for Collaboration

- **GitHub:** Hosts repositories and facilitates collaboration.
- **GitLab:** Provides integrated DevOps lifecycle tools.
- **SourceForge:** Long-standing platform for open source projects.

Learning Resources

- **Documentation:** Official project docs, wikis, and READMEs.
- **Tutorials and Courses:** Online platforms like Coursera, edX, and freeCodeCamp.
- **Community Support:** Forums like Stack Overflow and community chats.

Tools for Development

- **Version Control Systems:** Git, Mercurial.
- **Integrated Development Environments (IDEs):** VSCode, Eclipse.



Linux Operating System

- **Overview:** Kernel that forms the basis of various operating systems.
- **Impact:** Powers servers, desktops, and mobile devices (Android).

Apache HTTP Server

- **Overview:** Widely used web server software.
- **Significance:** Serves a large portion of the world's websites.

Mozilla Firefox

- **Overview:** Open source web browser.
- **Contribution:** Advocates for internet privacy and open standards.
-



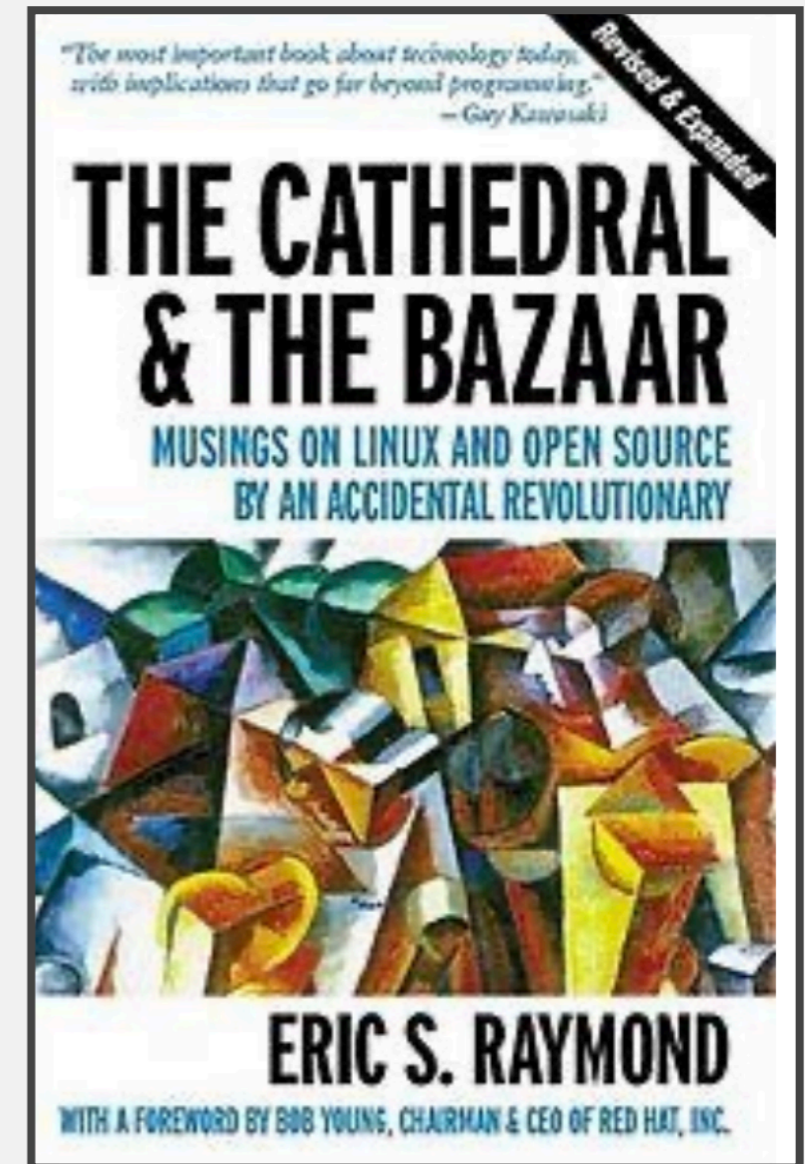
Perception (from some):

- Anarchy
- Demagoguery
- Ideology
- Altruism

Open-Source Ecosystems



The Cathedral and the Bazaar





Cathedral

- Developed centrally by a core group of members
- Available for all once complete (or at releases)
- Examples: GNU Emacs, GCC (back in the 1990s)
- “Sort of” examples today: Chrome
Intellij

Bazaar

- Developed openly and organically
- Wide participation (in theory, anyone can contribute)
Examples: Linux

OSS has many stakeholders / contributors



- **Core members**
 - Often (but not always) includes the original creators
 - Direct push access to main repository
 - May be further split into admin roles and developers
- **External contributors**
 - File bug reports and report other issues
 - Contribute code and documentation via pull requests
- **Other supporters**
 - Beta testers (users)
 - Sponsors (financial or platform)
 - Steering committees or public commenters (for standards and RFCs)
- **Spin-offs**
 - Maintainers of forks of the original repository



- Some OSS projects are managed by for-profit firms
 - Examples: Chromium (Google), Moby (Docker), Ubuntu (Canonical), TensorFlow (Google), PyTorch (Meta), Java (Oracle)
 - Contributors may be a mix of employees and community volunteers
 - Corporations often fund platforms (websites, test servers, deployments, repository hosting, etc.)
 - Corporations usually control long-term vision and feature roadmap
- Many OSS projects are managed by non-profit foundations or ad-hoc communities
 - Examples: Apache Hadoop/Spark/Hbase/Kafka/Tomcat (ASF), Firefox (Mozilla), Python (PSF), NumPy (community)
 - Foundations fund project infrastructure via charitable donations
 - Long-term vision often developed via a collaborative process (e.g., Apache) or by benevolent dictators (e.g., Python, Linux)
- Corporations still heavily rely on community-owned OSS projects • Many OSS non-profits are funded by Big Tech (e.g., Mozilla by Google)



MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

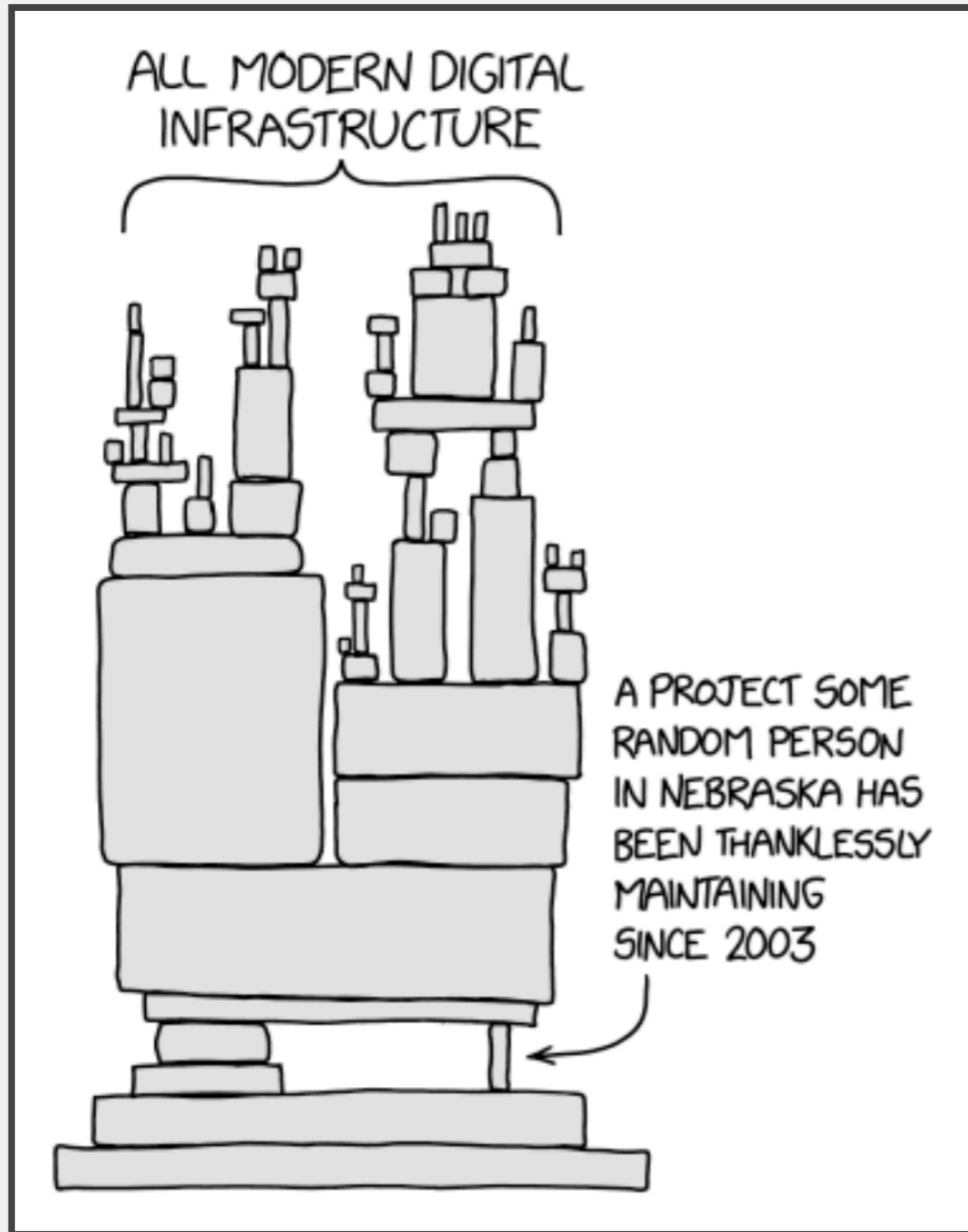
Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling ma-

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp



Use of Open-Source Software in Companies



QUARTZ | Make business better.™

HOME LATEST BUSINESS NEWS MONEY & MARKETS TECH & INNOVATION LIFESTYLE LEADERSHIP EMAILS PODCASTS EN ESPAÑOL

MEMBERSHIP

TECH & INNOVATION

NPM ERR!

How one programmer broke the internet by deleting a tiny piece of code

```
1 module.exports = leftpad;
2 function leftpad (str, len, ch) {
3   str = String(str);
4   var i = -1;
5   if (!ch && ch !== 0) ch = ' ';
6   len = len - str.length;
7   while (++i < len) {
8     str = ch + str;
9   }
10  return str;
11 }
12
13
```

Software Licenses



Note: I am not a lawyer (this is not legal advice)

Copyright vs. Intellectual Property



- IP and Patents cover an idea for solving a problem
 - Examples: Machine designs, pharma processes to manufacture certain drugs, (controversially) algorithms
 - Have expiry dates. IP can be licensed or sold/transferred for \$\$\$.
- Copyrights cover particular expressions of some work
 - Examples: Books, music, art, source code
 - Automatic copyright assignment to all new work unless a license authorizes alternative uses.
- Exceptions for trivial works and ideas.

Security & Privacy





Computer security refers to the practice of protecting computer systems and networks from unauthorized access, theft, damage, or disruption.

Confidentiality: Ensuring that sensitive information is only accessible to authorized individuals.

Integrity: Safeguarding the accuracy and completeness of data, preventing unauthorized modifications.

Availability: Ensuring that systems and data are accessible when needed by authorized users, often through measures like redundancy, fault tolerance, and disaster recovery



OWASP - **O**pen **W**eb **A**pplication **S**ecurity **P**roject

- Founded in 2001
- Mission to make the software vulnerability visible
- No profit organization

<https://owasp.org/www-project-top-ten/>



- 1. Broken Access Control**
- 2. Cryptographic Failures**
- 3. Injection**
- 4. Insecure Design**
- 5. Security Misconfiguration**
- 6. Vulnerable and Outdated Components**
- 7. Identification and Authentication Failures**
- 8. Software and Data Integrity Failures**
- 9. Security Logging and Monitoring Failures**
- 10. Server-Side Request Forgery (SSRF)**

Broken Access Control



Example Attack Scenarios

Scenario #1: The application uses unverified data in a SQL call that is accessing account information:

```
pstmt.setString(1, request.getParameter("acct"));  
ResultSet results = pstmt.executeQuery( );
```

An attacker simply modifies the browser's 'acct' parameter to send whatever account number they want. If not correctly verified, the attacker can access any user's account.

```
https://example.com/app/accountInfo?acct=notmyacct
```

Scenario #2: An attacker simply forces browses to target URLs. Admin rights are required for access to the admin page.

```
https://example.com/app/getappInfo  
https://example.com/app/admin_getappInfo
```

If an unauthenticated user can access either page, it's a flaw. If a non-admin can access the admin page, this is a flaw.

Mitigation: Implement strong access controls, always check user roles and permissions.

Cryptographic Failures



Example: An application encrypts credit card numbers in a database using automatic database encryption. However, this data is automatically decrypted when retrieved, allowing a SQL injection flaw to retrieve credit card numbers in clear text.

Mitigation: Use modern encryption standards like AES and secure key management.



Example Attack Scenarios

Scenario #1: An application uses untrusted data in the construction of the following vulnerable SQL call:

```
query = "SELECT \* FROM accounts WHERE custID='" + request.getParameter("id") + "'";
```

Scenario #2: Similarly, an application's blind trust in frameworks may result in queries that are still vulnerable, (e.g., Hibernate Query Language (HQL)):

```
.on.createQuery("FROM accounts WHERE custID='" + request.getParameter("id") + "'");
```

In both cases, the attacker modifies the 'id' parameter value in their browser to send: ' UNION SLEEP(10);--. For example:

```
http://example.com/app/accountView?id=' UNION SELECT SLEEP(10);--
```

This changes the meaning of both queries to return all the records from the accounts table. More dangerous attacks could modify or delete data or even invoke stored procedures.



Example: A cinema chain allows group booking discounts and has a maximum of fifteen attendees before requiring a deposit. Attackers could threat model this flow and test if they could book six hundred seats and all cinemas at once in a few requests, causing a massive loss of income.

Mitigation: Include secure design principles like threat modeling and secure design patterns.

Security Misconfiguration



```
public void ConfigureServices(IServiceCollection services)
{
    // Add CORS services
    services.AddCors(options =>
    {
        options.AddPolicy("AllowAllOrigins",
            builder =>
            {
                builder.AllowAnyOrigin()
                    .AllowAnyMethod()
                    .AllowAnyHeader();
            });
    });

    services.AddControllers();
}
```

Vulnerable and Outdated Components



Description: Using components with known vulnerabilities can open your application to attack.

Example: An old library with an unpatched security flaw being exploited by attackers.

Mitigation: Regularly update libraries and third-party software.

Identification and Authentication Failures



Example: Application session timeouts aren't set correctly. A user uses a public computer to access an application. Instead of selecting "logout," the user simply closes the browser tab and walks away. An attacker uses the same browser an hour later, and the user is still authenticated.

Mitigation: Enforce strong password policies, implement MFA, and secure sessions properly.



Description: Trusting insecure software updates or using code without verifying integrity.

Example: Compromised CI/CD pipeline delivering malicious code in a software update.

Mitigation: Implement code signing, secure CI/CD pipelines, and verify updates.

Security Logging and Monitoring Failures



Description: Insufficient logging and monitoring can prevent detection of breaches or attacks.

Example: Failing to log suspicious activities like failed login attempts.

Mitigation: Enable comprehensive logging and actively monitor for anomalies.



Description: SSRF occurs when an attacker tricks the server into making unintended requests to other systems.

Example: Exploiting a web application to send unauthorized requests to internal services.

Mitigation: Validate and sanitize all external requests, restrict internal server access.