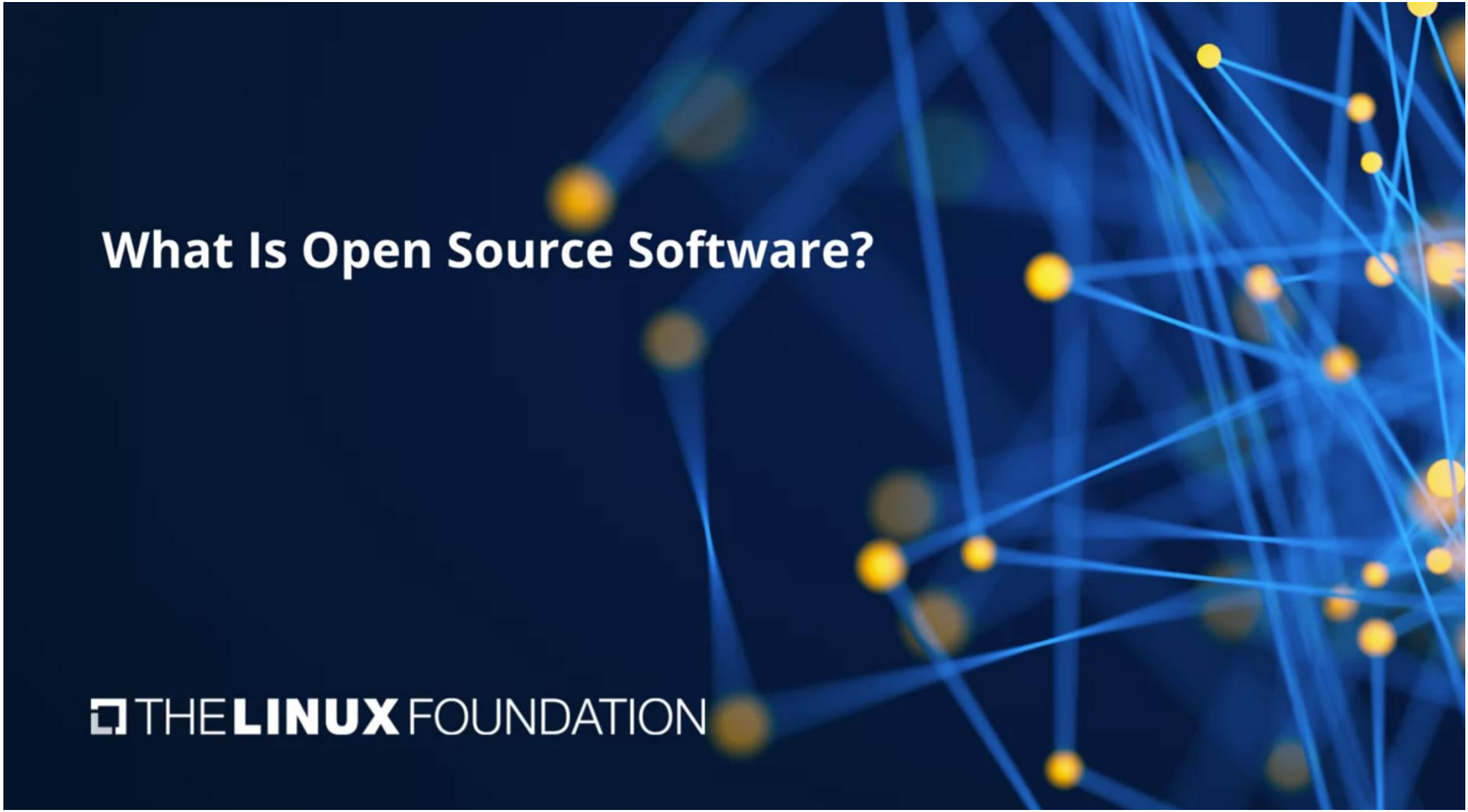

What Is Open Source Software?
THE LINUX FOUNDATION

Open Source Software

Source code is made available with a license which provides rights to:

- **examine**
- **modify**
- **redistribute**

without restriction on the user's identity or purpose.

Licensing Classifications

Permissive

- Any code changes need not be available to recipients
- Often preferred by companies

Example: BSD-licensed software

Restrictive

- Any code changes must be available to all recipients
- Sometimes called “Copyleft”

Example: GPL-licensed software

What Is Proprietary Software?

Proprietary Software

- Historically, the only real model used by commercial projects until the recent rise of OSS
- Only software owners have full legal access to the source code
- Trusted partners can be granted inspection rights if they sign a non-disclosure agreement (NDA) ⟶ Closed Source
- Software owners may or may not be the authors of the code

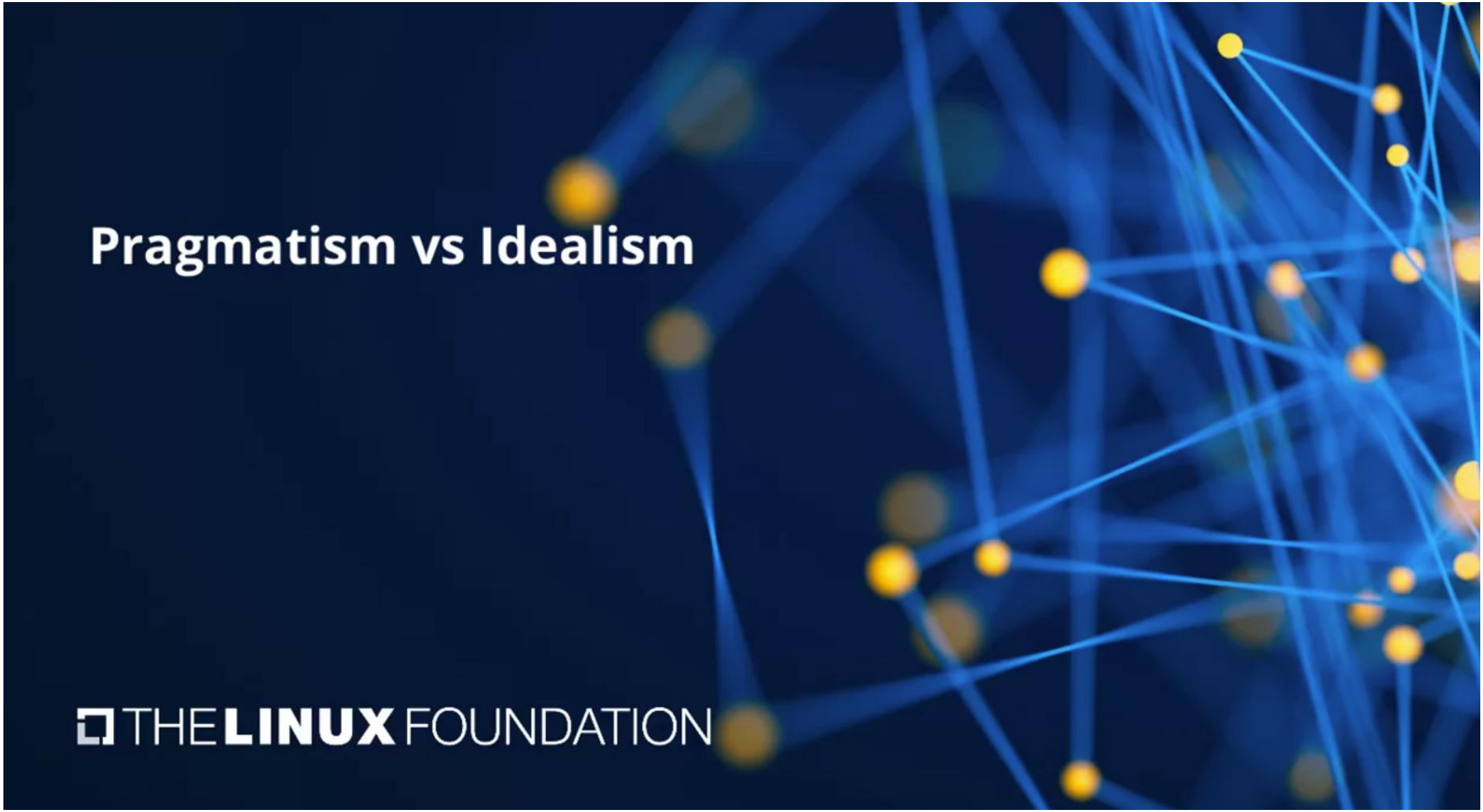
Using Proprietary Software

- To use proprietary software, end users must accept a license restricting their rights
- Such licenses generally:
  - Restrict the user's rights of re-distribution
  - Indemnify the product from damages due to either malfunction or misuse
  - Prohibit trying to reconstruct source code or use inside another product
- They may also sometimes restrict how a product is used

Price Is Not the Point!

- The difference with Proprietary and OSS models has nothing to do with price
- The license differences have to do with redistribution, modification, reuse of code, etc.

Pragmatism vs Idealism



Pragmatism vs Idealism

Pragmatism	Idealism
• The primary considerations are technical: - faster and better development (e.g. more contributors and review) - easier debugging, etc.	• "Free" as in freedom, not beer • Software open for ideological and ethical reasons, not just technological ones

Examples to Consider

- The objectives of both perspectives in many cases coincide
- Consider the following examples:
  - Should the software powering a life-saving medical device be secret?
  - Should the software powering voting machines be closed?
- The conflicts between these two attitudes towards open source has often been acrimonious and destructive

Open Source Governance Models

THE LINUX FOUNDATION

What Is a Governance Model?

- Any project needs organization to achieve its purpose
- How decisions get made and who makes them requires careful thought
- A project is still an open source one, whether or not:
  - Anyone can contribute or only a selected few
  - Decisions are made democratically by an authority
  - Plans and discussions are made public before releases
- How this is done determines the **Governance Model** for a project

Company-led: Mostly Closed Process

- Software design, development and releases are controlled by one entity
- External contributions may or may not be solicited
- Plans and release dates may not be described openly
- Internal discussions/controversies may not be aired
- Upon release, software is in the open
- Examples: Google Android, Red Hat Enterprise Linux

Benevolent Dictatorship: Strong Leadership

- One individual has overriding influence
- Project quality and success depend heavily on the dictator's wisdom and management capability
- Dictator's role may be social and political, not structural
- Maintainers write less and less code as projects mature
- Can avoid endless discussions and lead to a quicker pace of development
- Examples: Linux kernel, Wikipedia

Governing Board: Tighter Control by Smaller Groups

- A body (group) carries out discussions on open mailing lists and all decisions are made collectively
- Decisions about who can contribute, and how patches and new software are accepted, are made by governing body
- Much variation in governing structures, rules of organization, degree of consensus required, etc.
- Releases are less frequent, but well-debugged
- Examples: FreeBSD, Debian

Advantages of Open Source Software

Collaborative Development

- Enables software projects to build better software
- When progress is shared, not everyone has to solve the same problems and make the same mistakes - faster progress and reduced costs
- Having more eyeballs viewing code and more groups testing it leads to stronger and more secure code
- It is often hard for competitors to get used to the idea of sharing, and grasping that the benefits can be greater than the costs
- Competitors can compete on user-facing interfaces - users see plenty of differentiation and have varying experiences

Security and Quality of Source Code

- Coding standards and styles tend to be cleaner and more consistent on community projects:
  - It's embarrassing to show ugly, sloppy code
  - More people have to understand and work on the code
- More eyeballs examining code looking for security weaknesses before they are discovered by bad actors
- More input in original design to avoid bad ideas
- No "security through obscurity"
- No "just trust me"
- Potentially faster bug repair

Stakeholders

Some of the stakeholders benefiting from Open Source Software include:

- Users
- Businesses
- Education
- Developers

Users: Flexibility

- Mix and match software from different sources
- Save money on buying or leasing software
- Avoid vendor lock-in, maintain choice
- Look under the hood - “trust but verify”
- Have more fun!

Business: Collaborative Development

- Lowers total cost of development
- Speeds up time to market
- Encourages community feedback: criticism, suggestions, contributions
- Supports upstreaming to reduce future costs for new products that reuse code
- Uses well-delineated APIs

Business: Marketing

- Customers know what they are getting, can have confidence in quality, no secrets
- Product is seen as being part of a large ecosystem of related products
- More flexible, possible modular construction
- Adoption by larger community can lead to customer confidence product is here to stay

Education: Elementary-High (El-Hi) School, Public Systems

- Very large amount of teaching resources at little or no cost
- Very wide range of areas available for: using, operating, system administration, and programming
- Students do not become locked into vendor products
- School systems don't have to pay for expensive software, even at a discount
- Lower hardware costs and easier to use old hardware
- Skills that students will need in the workforce
- Unleashes student creativity: more fun!

Education: University

In addition to already discussed advantages (El-Hi school level) students can:

- Study and work on the internals of operating systems, applications and libraries, and system administration utilities
- Enter the workforce where they are most needed
- Develop good habits
- Be easily evaluated by prospective employers since their work is publicly accessible

Developers

- Makes it easier to not have to re-invent everything
- Helps to make good early decisions on product design
- More eyeballs on code find and fix bugs faster
- Allows for suggestions/contributions from a larger group
- Helps to find the next job
  - Code is readily available for evaluation
  - Shows how well you work and play with others
  - Shows how good you are at mentoring and maintaining projects and sub-projects
- Builds community (you are not alone!)


Contributing to Open
Source Software Projects
THE LINUX FOUNDATION

How to Contribute Properly to Open Source Software?

- Investigate the project, understand its workflow and style, identify the scope and nature of your work
- Identify how the project communicates:
  - Review mailing lists
  - Study archives
  - Join an Internet Relay Chat (IRC) Channel, if one exists
- Understand how contributions are submitted:
  - Mailing lists
  - Email
  - Revision control system (e.g. **git** or **subversion**)
- Study previous history

How to Contribute Properly to Open Source Software? (Cont.)

- Check if the project offers veteran contributors as mentors
- Offer your services for testing, finding bugs, etc., before you begin submitting code
- Make sure you are competent at whatever programming or scripting language the project uses
- Find the right balance between asking for review and suggestions early in the process, and waiting too long and dumping a lot of work on people at once
- Be polite and respectful, and avoid obscenities, flaming and trolling

Study and Understand the Project DNA

- Remember that unless you are starting a new project, there will already be a community established ethos and formal or informal leadership structure
- Ask questions:
  - Why does this project exist and why was it started?
  - Has it diverged far from its original purpose?
  - Is the contributing community large or small? Continuously active or only sporadically so?
  - Does it have a collective or singular governance structure?
  - What kind of licenses does it adopt? Is there a Contributor License Agreement (CLA) you have to agree to?
- Keep in mind that the majority of OSS project never really take off

Figure Out What Itch You Want to Scratch

- Most contributors get involved in open source projects because:
  - There is a bug/problem that interferes with their use of the project
  - They want to add additional functionality to the project
  - They want to learn something new
- Some projects have a "janitorial" list of things to do; these relatively straightforward tasks can help you get your feet wet
- Beginning contributors who go on to become important to a project rarely start with patches that do not affect functionality


Tips for Successful Contributions
THE LINUX FOUNDATION

Identify Maintainers, Their Work and Methods

- There are projects that have one maintainer (perhaps the benevolent dictator) or subsystem maintainers (e.g. the Linux kernel community)
- The maintainers have to:
  - Understand and review all submissions
  - Make sure they add something other than complication and do not introduce bugs
  - Make sure they don't conflict with other patches
  - Complete their work in a timely manner
  - Do everything respectfully
- Have a good relationship with relevant maintainers; be respectful and patient; offer help reviewing other contributors' submissions, debugging new features or fixing bugs

Get Early Input and Work In the Open

- As a new member of the community you probably don't know the history - your bright new idea might actually be an old one (discussed and rejected/brought to life and petered out)
- This doesn't mean your idea is a bad one; maybe it couldn't have been done in the past, but is feasible now as software and hardware evolved
- Never just introduce an idea and suggest someone else do it
- Propose your ideas to the community before you go too far with it; get opinions, suggestions and input; take advantage of institutional memory
- If you are uncomfortable having other people look at your work often, OSS is not for you!

Contribute Incremental Bits, Not Large Code Dumps

- Large code dumps overload maintainers and mailing lists, and make it difficult to process the individual parts
- Code contributions are best digested in small, unit-sized bites; for example, you may have a separate patch for each file you are modifying (sequential patches)
- It is possible that your functionality will not work until all patches are included - that's fine

Leave Your Ego at the Door: Don't Be Thin-Skinned

- OSS projects tend to attract some rather interesting characters, some of them can be quite irritating, even offensive
- If that's the case, rely on community members with long history to try and calm things down and resolve issues (occasionally, contributors may be purged from a project); don't feed the trolls
- When making criticism or engaging in any discussion thread, be polite and respect other people opinions; maybe your approach is not the best one
- Learn when to yield if you want to get code upstream

Do Not Discriminate or Offend

- Do not engage in behaviors that involve discrimination and offensive conduct with respect to:
  - Race
  - Sex
  - Sexual preferences
  - Religion
  - National origin
  - Politics
  - etc.
- People from rather far sides of the spectrum often work together successfully

Be Patient, Develop Long-Term Relationships, Be Helpful

- Most projects do not appreciate drive by, one-time contributors, and look for long-term maintenance
- Sometimes good code gets rejected, unless someone offers long-term commitment (think about the future when doing submissions)
- Develop meaningful relationships in the community by offering help in areas that are not directly related to your main interests
- Be viewed as a good citizen and real member of the community, not just someone exploiting it; this is particularly important if you are contributing as part of a commercial organization


Continuous Integration
THE LINUX FOUNDATION

Why Continuous Integration?

- Once upon a time, most software was written by a relatively small group of developers, often working in the same location and in frequent contact; coordination and division of responsibilities was straightforward
- **Revision control systems** were developed to accommodate more than one contributor working on a project
  - Central **repository** stores the master copy of the project; one or more developers possess the ability to make changes and then check them in
- The Linux kernel was the first really huge distributed development project, and its creator, Linus Torvalds, invented the **git** system for rationalizing distributed development

Why Continuous Integration? (Cont.)

- A revision control system does not solve the problem of making sure what a diverse group of contributors is doing actually works together; that one set of new code or bug fixes does not conflict with another - this can only be done by **testing**
- Testing requires the following considerations:
  - Can overlapping sets of changes be applied simultaneously, or do they conflict?
  - Does the project compile when all changes are applied?
  - Does it work on all possible targets?
  - What does working mean?
  - Are there non-trivial test suites that can exercise a representative workload enough to give confidence things are fine?
- Continuous integration techniques ensure that testing is so frequent that any problems cannot persist for long; distributed developers stay on the same page.

Continuous Integration, Continuous Delivery, Continuous Deployment

We can distinguish three separate steps/stages:

- **Continuous integration** - changes merged into the main branch ("master") as often as possible; automated builds run on as many variations of software and hardware as possible; conflicts are resolved as soon as they arise
- **Continuous delivery** - the release process is automated and projects are ready to be delivered to consumers of the build; thorough testing is done on all relevant platforms
- **Continuous deployment** - the product is released to customers; again, in an automated fashion

Note: Continuous integration can be considered to include both delivery and deployment.

Continuous Integration, Continuous Delivery, Continuous Deployment (Cont.)

The time gap between these steps is meant to be as close to zero as possible. In a perfect word, developer changes can reach end user customers the same day or even in minutes.

Costs and Benefits

Costs	Benefits
Changes have to be merged very often, putting a possible strain on developers	Developers don't go down the wrong path and compound fixable mistakes, or get in each other's way
The repository must be monitored by a continuous integration server; staff has to be allocated to do this	The build steps are fully automated; all the work has been done up front, instead of each time build testing needs to be done
Scripts and other tools have to be run to perform automated tests, report their results and take appropriate actions - it can be a lot of work to prepare this infrastructure	Regressions (bugs which break working product) may be minimized; releases should have fewer bugs

Tools

- There are many well-developed continuous integration software tools including:
  - **Jenkins** (the most widely used)
  - Travis
  - TeamCity
  - GO CD
  - GitLab CI
  - Bamboo
  - Codeship
  - CircleCI
- Some of these products are free in cost, others are not

Open Source Software Licenses and Legal Issues

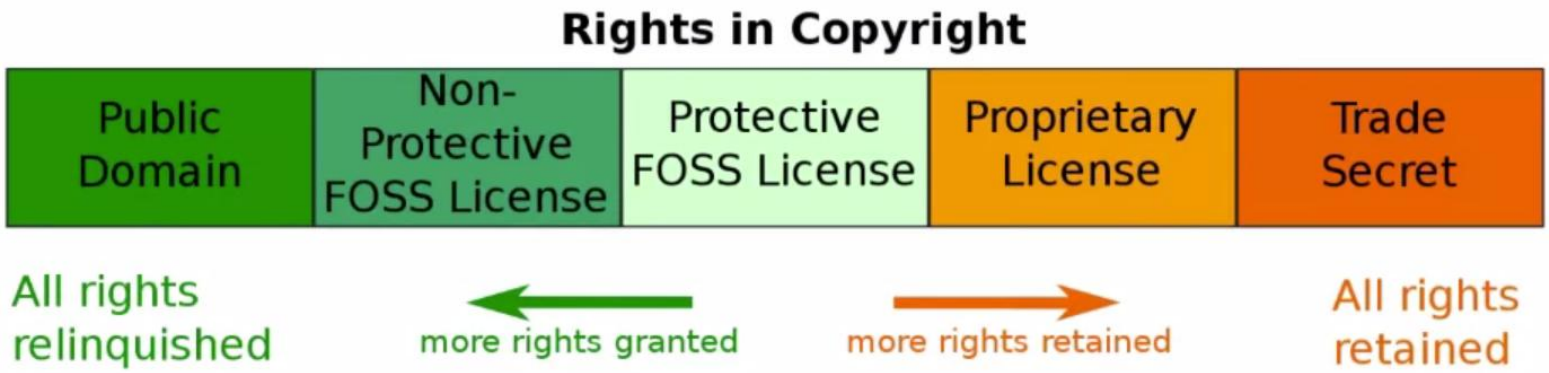
THE LINUX FOUNDATION

Restrictive vs Permissive Licensing

- There is an almost infinite variety of available open source licenses
- Choice should depend on both needs and philosophy
- Two basic categories:
  - **Restrictive licenses** - software remains open; strong limitations on any attempt to make proprietary closed products; changes to the code available to future recipients; e.g. GPL (a copyleft license)
  - **Permissive licenses** - don't require modifications and enhancements to be generally available; e.g. BSD and Apache license

Choosing a License

- Choosing the best license is a very important decision and requires careful thought
- Switching to a different license later during the life of the project can be difficult if not impossible (especially, if there are many contributors with rights)



Fear, Uncertainty and Doubt (FUD)

- First usage of the term Fear, Uncertainty and Doubt goes back as far as the 1920s; use of the acronymic form seems to date from the 1970s
- FUD means disseminating misinformation to influence recipients to avoid certain strategies, products or classes of products by appealing to fear
- Microsoft was widely accused of spreading FUD about Linux in the 1990s; in present day Microsoft has stopped doing so and it is actually employing OSS widely

FUD statement	Why it is not true
OSS is a virus	One has to be careful about respecting licenses but many prominent companies heave learned how to combine open and closed software in their offerings; there are companies and organizations dedicated to helping ensure this is done properly
OSS infringes on software and forces you to grant patent rights to others	Once again proper legal analysis is required
OSS products leave nowhere to turn when they break or to get technical help	Many open source products are supported by serious, long-living companies (such as Red Hat Enterprise Linux), as well as smaller organizations, plus there is a lot of freely available help online, and there are many consultants that can be hired (there is more competition available for such help than there would be with vendor lock-in on a product)
OSS requires a lot of legal help to avoid the above pitfalls and is thus very expensive	Even proprietary software requires significant legal analysis to properly avoid copyright and patent infringement, etc.; OSS is no different and not more expensive, and having all the software being available in source form expedites the auditing process.

Legal Issues

- Companies require interaction with lawyers, either on staff or external, to make sure they do not violate copyrights and licenses
- There are many kinds of licenses and one has to be careful, but once an organization develops proper reasonable procedures, it is just a standard part of any project
- Part of this is training the developers to understand the dos and don'ts of working with OSS

Patents and Licenses

THE LINUX FOUNDATION

Software Patents

- A software patent gives exclusionary rights to material such as a computer program, library, interfaces or even techniques and algorithms
- Software patents must be filed in for each nation (or trading block such as the European Union) in which coverage is desired ⟶ an expensive and time-consuming project
- Exactly what a patent can or cannot cover varies from jurisdiction to jurisdiction
  - USA ⟶ exclusion of "abstract ideas"

Do We Need Software Patents?

- Software patents vs copyright and trademark laws
- Software patents have been often used defensively, with corporations cross-licensing each other's work to avoid litigation

Open Invention Network (OIN)

- Founded in 2005 (over 3000 members in 2019, including Google, IBM, NEC, Philips, Red Hat, Sony, SUSE and Toyota)
- Created as a global patent pool
  - Companies and other entities enter in a mutual non-aggression agreement within the Linux-based ecosystem
  - Members agree in return for not suing each other over patent issues
- Microsoft Sells Out
  - Microsoft joined OIN in October 2018, opening up over 60000 patents

openinventionnetwork

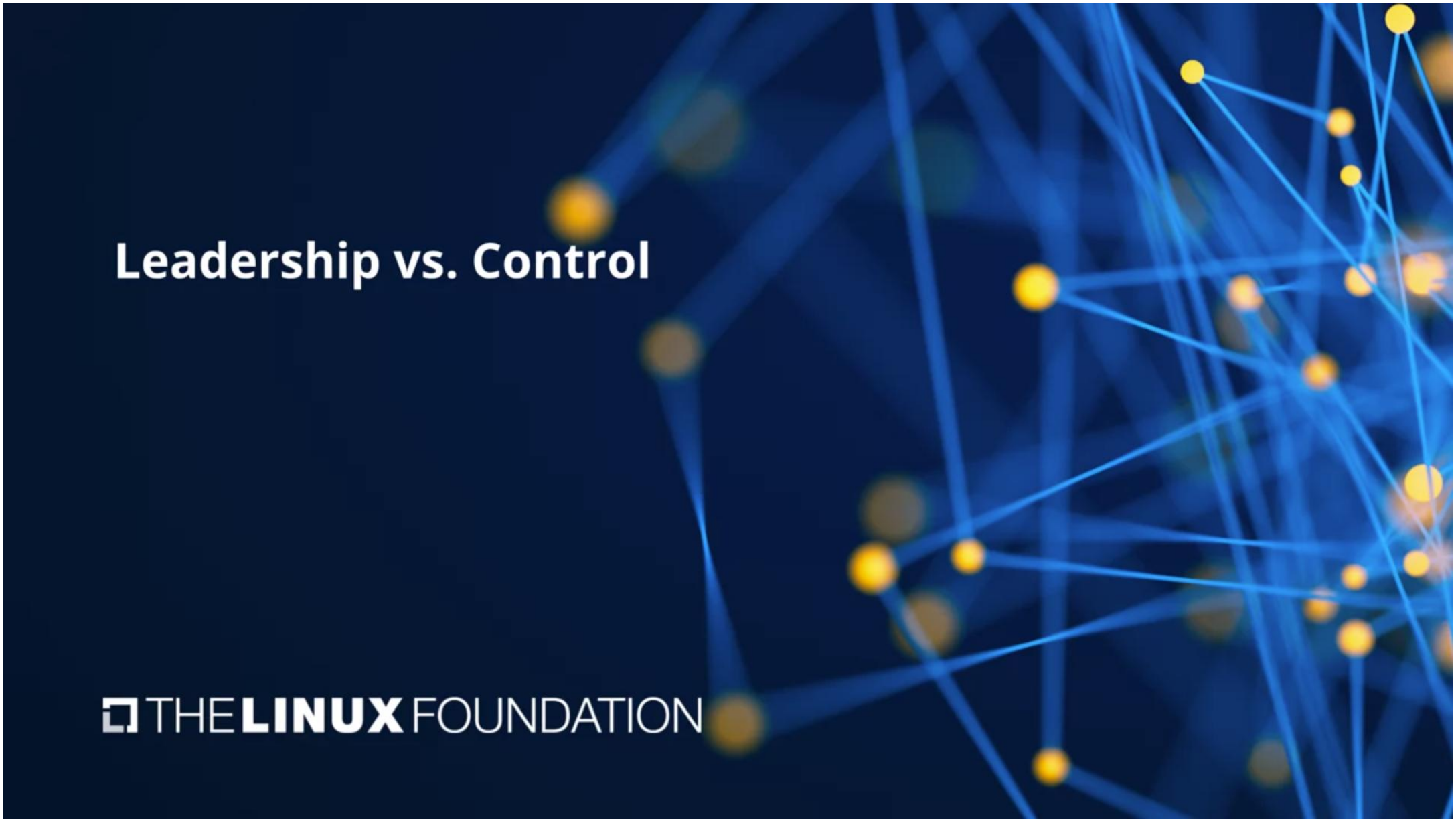
Patents and Licenses

- The intersection of software licenses and patents is rather complicated
- A partial table of properties about whether or not a license conveys explicit patent rights from contributors to users:

Yes	No
Apache 2.0 GPL 3.0 and LGPL 3.0 MPL 1.0, 1.1, 2.0 EPL 1.0 CDDL 1.0	BSD 2-Clause, 3-Clause MIT GPL 2.0 and LGPL 2.1

- Sometimes, projects will use a license that does not provide an express grant of patents, but then add a grant of patents in a separate file (e.g. **`LICENSE`** and **`PATENTS`** file in the source code)

Leadership vs. Control



Loosen the Chains

- A good leader listens and empowers all participants to present their ideas, and contribute higher quality and imaginative work
- Popular leadership paradigm - **Benevolent Dictator for Life** (BDFL)
  - Can make the final decision (generally more efficiently than a majority vote or consensus of a committee), but that does not alleviate them from listening and giving proper consideration to debates in the community and even occasionally including changes and contributions they personally have problems with
- If the controllers of a project take and do not give back by mentoring and moderating, they are limiting what a project can accomplish

Mentoring

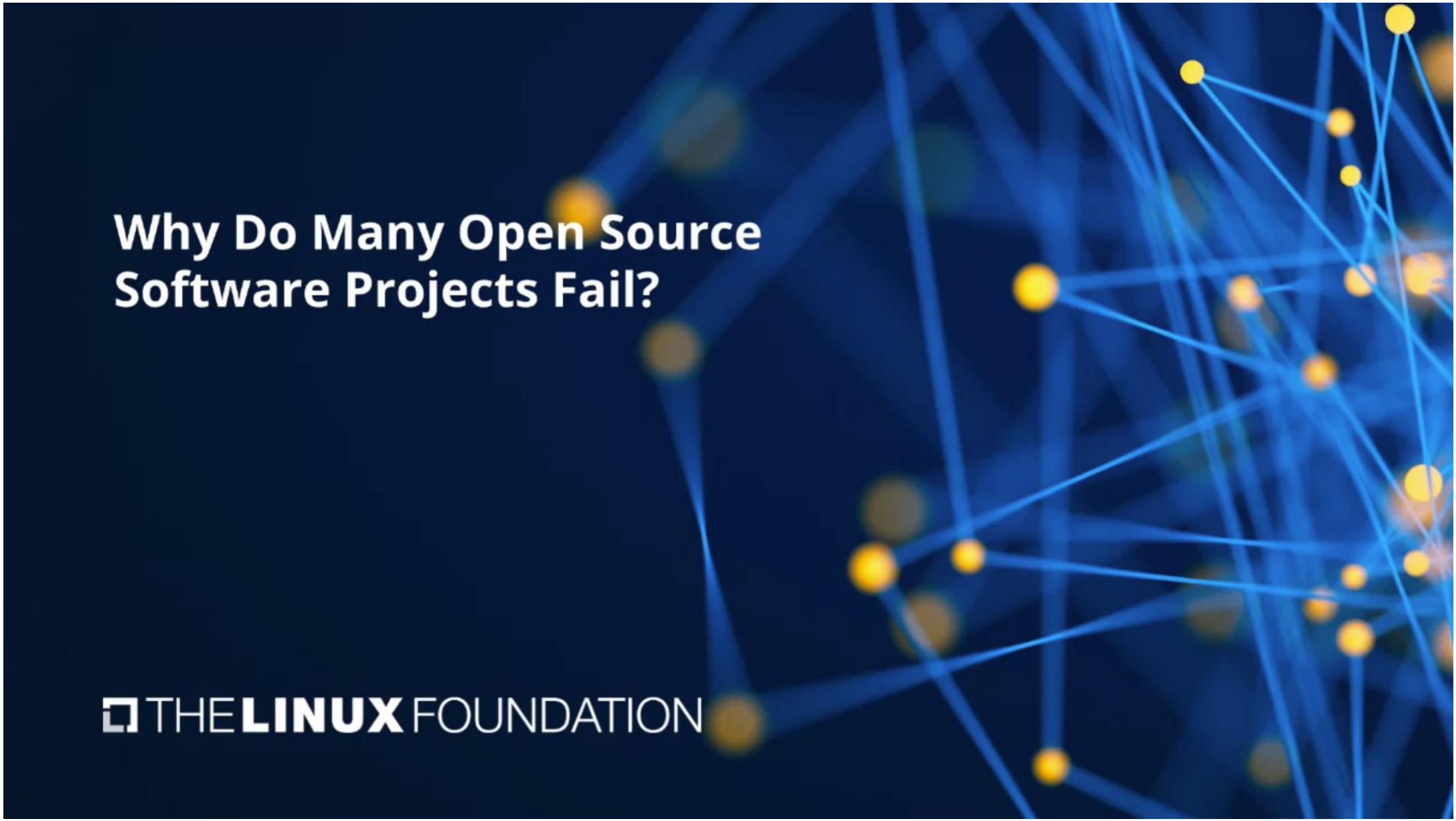
- Depending on the size of the project, there may be one level of leadership or more, including subsystem maintainers whose responsibilities include:
  - Handling contributions efficiently
  - Knowing when to send patches back for revision
  - Sending them up the chain with or without revision
  - Discarding them with proper explanation
- Obtaining knowledge and skills necessary to become a good maintainer requires training, or more precisely **mentoring**, from those who have done it successfully before

Mentoring (Cont.)

- Another level of mentoring involves new members of a project - if they can attach themselves to more experienced contributors, who may or may not be maintainers, they can learn the ropes with less pain:
  - How to produce patches in the proper form
  - How to phrase questions and proposals
  - How to acquire realistic expectations of how quickly their work might be considered and whether it has a chance of success
- Proper mentoring empowers people, leads to a more efficient workflow, and contributes to long-term vitality and growth of a project

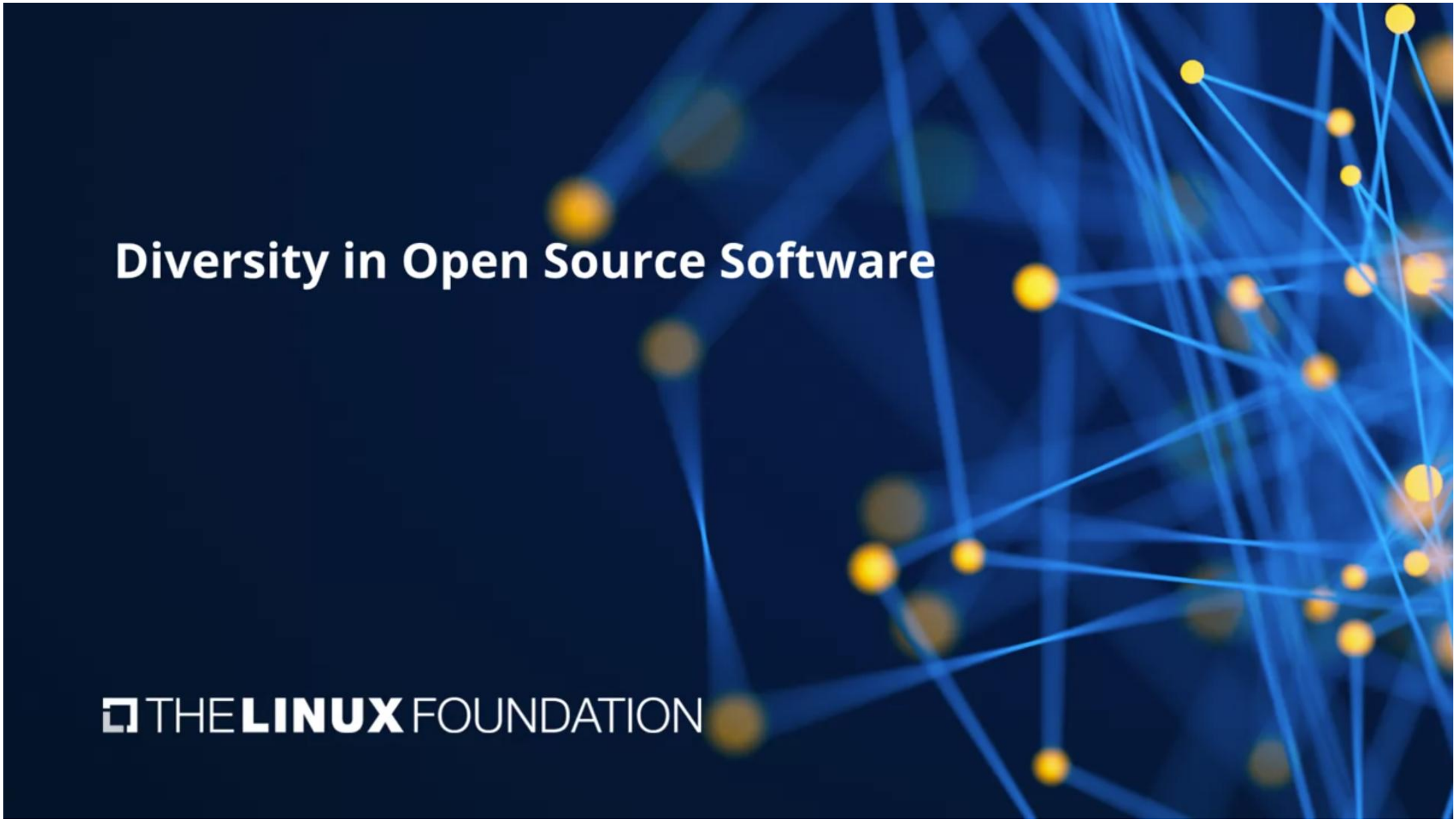
Building Trust

- Without **trust** an open source project cannot function:
  - Contributors must trust that their submissions will be treated with respect and given reasonably prompt consideration
  - Project leaders must trust subsystem maintainers are qualified and capable of doing their job; if they are not living up to their responsibilities, they can be tutored on doing a better job or be asked to transition out of the role
- Trust must be earned and must go beyond the benefit of the doubt
- As a project matures, reputations are earned and new members should be aware of the history


Why Do Many Open Source Software Projects Fail?
THE LINUX FOUNDATION

Why Projects Fail?

- The vast majority of OSS projects do not succeed; some of the reasons why include:
  - Insufficient interest
  - Competition from more established projects that duplicate the intended functionality (even if the new project seems to have features the older one does not)
  - Poor leadership
  - Not enough developers
  - Insufficient funding
  - Insufficient or uninformed attention to licensing issues
  - Low barrier to entry - projects are very easy to start
- The low survival rate should not be seen as a bad thing; most (but not all) successful open source projects start small and it can be quite difficult to predict which ones will survive and thrive


Diversity in Open Source Software
THE LINUX FOUNDATION

Diversity and OSS

- The word "**Open**" that appears in OSS might be taken to indicate a friendly, welcoming environment; unfortunately, unless a proper atmosphere permeates a project, this may be a false promise
- **Diversity** in an OSS project can mean many different things:
  - Race and national origin
  - Sex and gender identity issues
  - Geographical/regional issues (including both language and cultural differences)
  - Religious beliefs and political views
  - Acceptance of different opinions and methods about how the project should take shape and develop in the future
- While it is just *the right thing to do* to accept contributors and reviewers from divergent backgrounds, diversity also leads to a better project due to unleashing more sources of new ideas, approaches, and contribution

Share

Sex and Gender Identity

- In general, software engineering, system administration, and IT have been dominated by men
- Incidents of unwelcome behavior have long been a problem at some conferences
  - The Linux Foundation staff has very stringent policies regarding avoiding and dealing with any improper behavior
- Conference speakers should avoid marketing with sexual pictures, jokes and innuendos, and should try to use sex-neutral terms and pronouns, etc.
- Besides the fact that discrimination is wrong, it also shrinks the potential pool of contributors and users
- Any misogynistic and/or homophobic statements in mailing lists, discussion groups, chats, etc., should be promptly beaten down and criticized; don't contribute if a community accepts demeaning or unwelcoming behavior

THE LINUX FOUNDATION

0:56 / 4:31

Race, National Origin, Geography and Language

- Discrimination based on race and national origin is obviously wrong but is rarely explicit
- Geographical and language differences can lead to misunderstandings and insults
  - The vast majority of open source projects are primarily done in English and developers tend to have at least a working familiarity with it, but that is quite different than everyone being on the same page
  - Even among English speakers, words can have quite different meanings and usage in various countries
  - We can choose our words with sufficient care, try to be inoffensive and clear (e.g. be careful when using idioms and metaphors)
- Localization/internationalization - providing translated documents, using standard methods to provide help and error messages in multiple languages

Tone and Criticism

- Another aspect is tone; in some cultures cursing is rather routine and coarse and not offensive in the way it is in others
- It is important how criticism is offered; adopting the proper criticism methods is a two way street
  - Those used to a very direct approach should continue to do so but should avoid unnecessary nastiness
  - Those not used to such an approach need to develop a thicker skin and pay attention to the content as much as possible, not the wrapping it comes in
- The goal is to promote healthy and rapid development from as large a pool of contributors and users as possible, not just to behave nicely because it is good manners

Religion and Politics

- Discussion of religion is generally inappropriate (denigration of anyone's beliefs and proselytizing in favor of any religion) and has nothing to do with either technical or governance aspects of an open source project
- The same is true with political opinions extraneous to the project (e.g. national or international affairs)
- OSS is built on the concept of freely available intellectual product; once freedom is in the air, it becomes hard to avoid political discussions
- Once a discussion with political aspects veers away from direct bearing on the project it should be moved into other channels
- One should not dissuade anyone from participating in a project based on their political or religious views - this limits the potential pool of contributors and hurts a project

Share

Differences of Opinion

- Collaboration involves:
  - Dividing work among more than one group or individual, to be merged later
  - Reviewing it, accepting as is, sending back for revisions, or changing and sending further along
  - Competition between differing approaches
- Besides the usual advice about being civil and respectful, it is also very important to keep in mind that different people do things in different ways
  - Accept someone else's contribution that has the same effect as yours even if you think yours is better; if you do not prevail in the discussions of which approach to merge, you may just have to accept your less desired outcome
  - Or you can think more of things your approach can take care of the other cannot and add increased functionality and try again
- With more debate and more opinions, often a better project can be built