OPENCHAIN

CURRICULUM

FOSS Training Reference Slides for the OpenChain Specification 1.1

These slides follow US law. Different legal jurisdictions may have different legal requirements.
This should be taken into account when using these slides as part of a compliance training program.

These slides do not contain legal advice

What is the OpenChain Curriculum?

OPENCHAIN

- The OpenChain Project helps to identify and share the core components of a Free and Open Source Software (FOSS) compliance program.
- The core of the OpenChain Project is the **Specification**. This identifies and publishes the core requirements a FOSS compliance program should satisfy.
- The OpenChain **Curriculum** supports the Specification by providing freely available training material.
- These slides help companies satisfy the requirements of the Specification Section 1.2. They can also be used for general compliance training.

Learn more at: `https://www.openchainproject.org`

Contents

FOSS Policy

- <<This is a placeholder slide to identify where your FOSS policy can be found (OpenChain Specification 1.1, section 1.1.1)>>

- You can get an example FOSS policy via the Linux Foundation Open Compliance Program at: https://www.linux.com/publications/generic-foss-policy

CHAPTER 1

What is Intellectual Property?

What is “Intellectual Property”?

OPENCHAIN

- Copyright: protects original works of authorship
  - Protects expression (not the underlying idea)
  - It covers software, books, and similar works
- Patents: useful inventions that are novel and non-obvious
  - Limited monopoly to incentivize innovation
- Trade secrets: protects valuable confidential information
- Trademarks: protects marks (word, logos, slogans, color, etc.) that identify the source of the product
  - Consumer and brand protection; avoid consumer confusion and brand dilution

This chapter will focus on copyright and patents,
the areas most relevant to FOSS compliance.

Copyright Concepts in Software

OPENCHAIN

- Basic rule: copyright protects creative works
- Copyright generally applies to literary works, such as books, movies, pictures, music, maps
- Software is protected by copyright
  - Not the functionality (that's protected by patents) but the expression (creativity in implementation details)
  - Includes Binary Code and Source Code
- The copyright owner only has control over the work that he or she created, not someone else's independent creation
- Infringement may occur if copying without the permission of the author

Copyright Rights Most Relevant to Software

OPENCHAIN

- The right to *reproduce* the software – making copies
- The right to create "*derivative works*" – making modifications
  - The term derivative work comes from the US Copyright Act
  - It is a "term of art" meaning that it has a particular meaning based on the statute and not the dictionary definition
  - In general it refers to a new work based upon an original work to which enough original creative work has been added so that the new work represents an original work of authorship rather than a copy
- The right to *distribute*
  - Distribution is generally viewed as the provision of a copy of a piece of software, in binary or source code form, to another entity (an individual or organization outside your company or organization)

Note: The interpretation of what constitutes a "derivative work" or a "distribution" is subject to debate in the FOSS community and within FOSS legal circles

Patent Concepts in Software

OPENCHAIN

- Patents protect functionality – this can include a method of operation, such as a computer program
  - Does not protect abstract ideas, laws of nature
- A patent application must be made in a specific jurisdiction in order to obtain a patent in that country. If a patent is awarded, the owner has the right to stop anybody from exercising its functionality, regardless of independent creation
- Other parties who want to use the technology may seek a patent license (which may grant rights to use, make, have made, sell, offer for sale, and import the technology)
- Infringement may occur even if other parties independently create the same invention

Licenses

- A “license” is the way a copyright or patent holder gives permission or rights to someone else
- The license can be limited to:
  - Types of use allowed (commercial / non-commercial, distribution, derivative works / to make, have made, manufacture)
  - Exclusive or non-exclusive terms
  - Geographical scope
  - Perpetual or time limited duration
- The license can have conditions on the grants, meaning you only get the license if you comply with certain obligations
  - E.g, provide attribution, or give a reciprocal license
- May also include contractual terms regarding warranties, indemnification, support, upgrade, maintenance

Check Your Understanding

OPENCHAIN

- What type of material does copyright law protect?
- What copyright rights are most important for software?
- Can software be subject to a patent?
- What rights does a patent give to the patent owner?
- If you independently develop your own software, is it possible that you might need a copyright license from a third party for that software? A patent license?

CHAPTER 2

Introduction to FOSS Licenses

FOSS Licenses

OPENCHAIN

- FOSS licenses by definition make source code available under terms that allow for modification and redistribution
- FOSS licenses may have conditions related to providing attributions, copyright statement preservation, or a written offer to make the source code available
- One popular set of licenses are those approved by the Open Source Initiative (OSI) based on their Open Source Definition (OSD). A complete list of OSI-approved licenses is available at http://www.opensource.org/licenses/

Permissive FOSS Licenses

OPENCHAIN

- Permissive FOSS license: a term used often to describe minimally restrictive FOSS licenses
- Example: BSD-3-Clause
  - The BSD license is an example of a permissive license that allows unlimited redistribution for any purpose in source or object code form as long as its copyright notices and the license's disclaimers of warranty are maintained
  - The license contains a clause restricting use of the names of contributors for endorsement of a derived work without specific permission
- Other examples: MIT, Apache-2.0

License Reciprocity & Copyleft Licenses

OPENCHAIN

- Some licenses require that if derivative works (or software in the same file, same program or other boundary) are distributed, the distribution is under the same terms as the original work
- This is referred to as a “copyleft” or “reciprocal” effect
- Example of license reciprocity from the GPL version 2.0:

  You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed […] under the terms of this License.

- Licenses that include reciprocity or Copyleft clauses include all versions of the GPL, LGPL, AGPL, MPL and CDDL

Proprietary License or Closed Source

- A proprietary software license (or commercial license or EULA) has restrictions on the usage, modification and/or distribution of the software
- Proprietary licenses are unique to each vendor – there are as many variations of proprietary licenses as there are vendors and each must be evaluated individually
- FOSS developers often use the term “proprietary” to describe a commercial non-FOSS license, even though both FOSS and proprietary licenses are based on intellectual property and provide a license grant to that property

Other Non-FOSS Licensing Situations

- Freeware – software distributed under a proprietary license at no or very low cost
  - The source code may or may not be available, and creation of derivative works is usually restricted
  - Freeware software is usually fully functional (no locked features) and available for unlimited use (no locking on days of usage)
  - Freeware software licenses usually impose restrictions in relation to copying, distributing, and making derivative works of the software, as well as restrictions on the type of usage (personal, commercial, academic, etc.)
- Shareware – proprietary software provided to users on a trial basis, for a limited time, free of charge and with limited functionalities or features
  - The goal of shareware is to give potential buyers the opportunity to use the program and judge its usefulness before purchasing a license for the full version of the software
  - Most companies are very leery of Shareware, because Shareware vendors often approach companies for large license payments after the software has freely propagated within their organizations.

Other Non-FOSS Licensing Situations

- “Non-commercial” – some licenses have most of the characteristics of a FOSS license, but are limited to non-commercial use (e.g. CC-BY-NC).
  - FOSS by definition cannot limit the field of use of the software
  - Commercial use is a field of use so any restriction prevents the license from being FOSS

Public Domain

OPENCHAIN

- The term **public domain** refers to software not protected by law and therefore usable by the public without requiring a license
- Developers may include a *public domain declaration* with their software
  - E.g., “All of the code and documentation in this software has been dedicated to the public domain by the authors.”
  - The public domain declaration is not the same as a FOSS license

- A public domain declaration attempts to waive or eliminate any intellectual property rights that the developers may have in the software to make it clear that it can be used without restriction, but the enforceability of these declarations is subject to dispute within the FOSS community and its effectiveness at law varies from jurisdiction to jurisdiction
- Often the public domain declaration is accompanied by other terms, such as warranty disclaimers; in such cases, the software may be viewed as being under a license rather than being in the public domain

License Compatibility

- License compatibility is the process of ensuring that license terms do not conflict.
- If one license requires you to do something and another prohibits doing that, the licenses conflict and are not compatible if the combination of the two software modules trigger the obligations under a license.
  - GPL-2.0 and EPL-1.0 each extend their obligations to “derivative works” which are distributed.
  - If a GPL-2.0 module is combined with an EPL-1.0 module and the merged module is distributed, that module must
    - (according to GPL-2.0) be distributed under GPL-2.0 only, and
    - (according to EPL-1.0) under EPL-1.0 only.
  - The distributor cannot satisfy both conditions at once so the module may not be distributed.
  - This is an example of *license incompatibility.*

The definition of “derivative work” is subject to different views in the FOSS community and its interpretation in law is likely to vary from jurisdiction to jurisdiction.

Notices

OPENCHAIN

Notices, such as text in comments in file headers, often provide authorship and licensing information. FOSS licenses may also require the placement of notices in or alongside source code or documentation to give credit to the author (an attribution) or to make it clear the software includes modifications.

- **Copyright notice** – an identifier placed on copies of the work to inform the world of copyright ownership. Example: `Copyright © A. Person (2016)`
- **License notice** – a notice that specifies and acknowledges the license terms and conditions of the FOSS included in the product.
- **Attribution notice** – a notice included in the product release that acknowledges the identity of the original authors and / or sponsors of the FOSS included in the product.
- **Modification notice** – a notice that you have made modifications to the source code of a file, such as adding your copyright notice to the top of the file.

Multi-Licensing

- Multi-licensing refers to the practice of distributing software under two or more different sets of terms and conditions simultaneously
  - E.g., when software is “dual licensed,” the copyright owner gives each recipient the choice of two licenses
- Note: This should not be confused for situations in which a licensor imposes more than one license, and you must comply with *all* of them

Check Your Understanding

OPENCHAIN

- What is a FOSS license?
- What are typical obligations of a permissive FOSS license?
- Name some permissive FOSS licenses.
- What does license reciprocity mean?
- Name some copyleft-style licenses.
- What needs to be distributed for code used under a copyleft license?
- Are Freeware and Shareware software considered FOSS?
- What is a multi-license?
- What information may you find in FOSS Notices, and how may the notices be used?

CHAPTER 3

Introduction to FOSS Compliance

FOSS Compliance Goals

OPENCHAIN

- **Know your obligations.** You should have a process for identifying and tracking FOSS components that are present in your software

- **Satisfy license obligations.** Your process should be capable of handling FOSS license obligations that arise from your organization's business practices

What Compliance Obligations Must Be Satisfied?

OPENCHAIN

Depending on the FOSS license(s) involved, your compliance obligations may consist of:

- **Attribution and Notices.** You may need to provide or retain copyright and license text in the source code and/or product documentation or user interface, so that downstream users know the origin of the software and their rights under the licenses. You may also need to provide notices regarding modifications, or full copies of the license.
- **Source code availability.** You may need to provide source code for the FOSS software, for modifications you make, for combined or linked software, and scripts that control the build process.
- **Reciprocity.** You may need to maintain modified versions or derivative works under the same license that governs the FOSS component.
- **Other terms.** The FOSS license may restrict use of the copyright holder name or trademark, may require modified versions to use a different name to avoid confusion, or may terminate upon any breach.

FOSS Compliance Issues: Distribution

OPENCHAIN

- Dissemination of material to an outside entity
  - Applications downloaded to a user's machine or mobile device
  - JavaScript, web client, or other code that is downloaded to the user's machine
- For some FOSS licenses, access via a computer network can be a "trigger" event
  - Some licenses define the trigger event to include permitting access to software running on a server (e.g., all versions of the Affero GPL if the software is modified) or in the case of "users interacting with it remotely through a computer network"

FOSS Compliance Issues: Modification

OPENCHAIN

- Changes to the existing program (e.g., additions, deletions of code in a file, combining components together)
- Under some FOSS licenses, modifications may cause additional obligations upon distribution, such as:
  - Providing notice of modification
  - Providing accompanying source code
  - Licensing modifications under the same license that governs the FOSS component

FOSS Compliance Program

OPENCHAIN

Organizations that have been successful at FOSS compliance have created their own *FOSS Compliance Programs* (consisting of policies, processes, training and tools) to:

1. Facilitate effective usage of FOSS in their products (commercial or otherwise)
2. Respect FOSS developer/owner rights and comply with license obligations
3. Contribute to and participate in FOSS communities

Implementing Compliance Practices

OPENCHAIN

Prepare business processes and sufficient staff to handle:

- Identification of the origin and license of all internal and external software
- Tracking FOSS software within the development process
- Performing FOSS review and identifying license obligations
- Fulfillment of license obligations when product ships
- Oversight for FOSS Compliance Program, creation of policy, and compliance decisions
- Training

Compliance Benefits

OPENCHAIN

Benefits of a robust FOSS Compliance program include:

- Increased understanding of the benefits of FOSS and how it impacts your organization
- Increased understanding of the costs and risks associated with using FOSS
- Increased knowledge of available FOSS solutions
- Reduction and management of infringement risk, increased respect of FOSS developers/owners' licensing choices
- Fostering relationships with the FOSS community and FOSS organizations

Check Your Understanding

OPENCHAIN

- What does FOSS compliance mean?
- What are two main goals of a FOSS Compliance Program?
- List and describe important business practices of a FOSS Compliance Program.
- What are some benefits of a FOSS Compliance Program?

CHAPTER 4

Key Software Concepts for FOSS Review

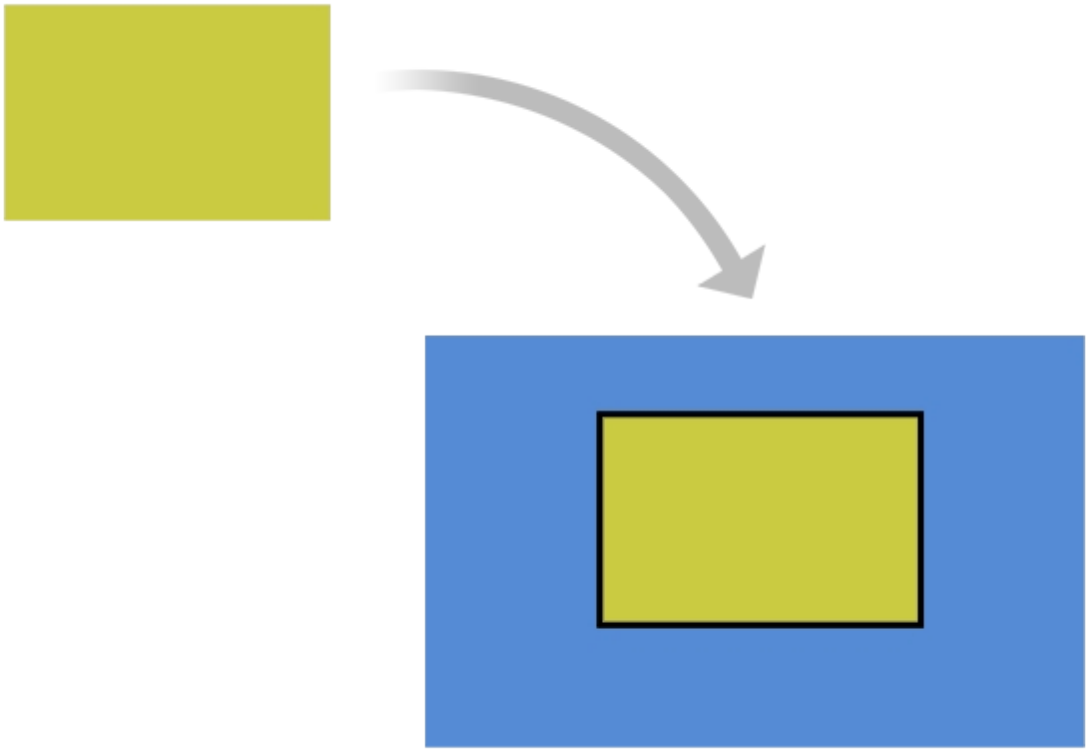
How do you want to use a FOSS component?

Common scenarios include:

- Incorporation
- Linking
- Modification
- Translation

Incorporation

A developer may copy portions of a FOSS component into your software product.



Relevant terms include:

- Integrating
- Merging
- Pasting
- Adapting
- Inserting

Linking

A developer may link or join a FOSS component with your software product.

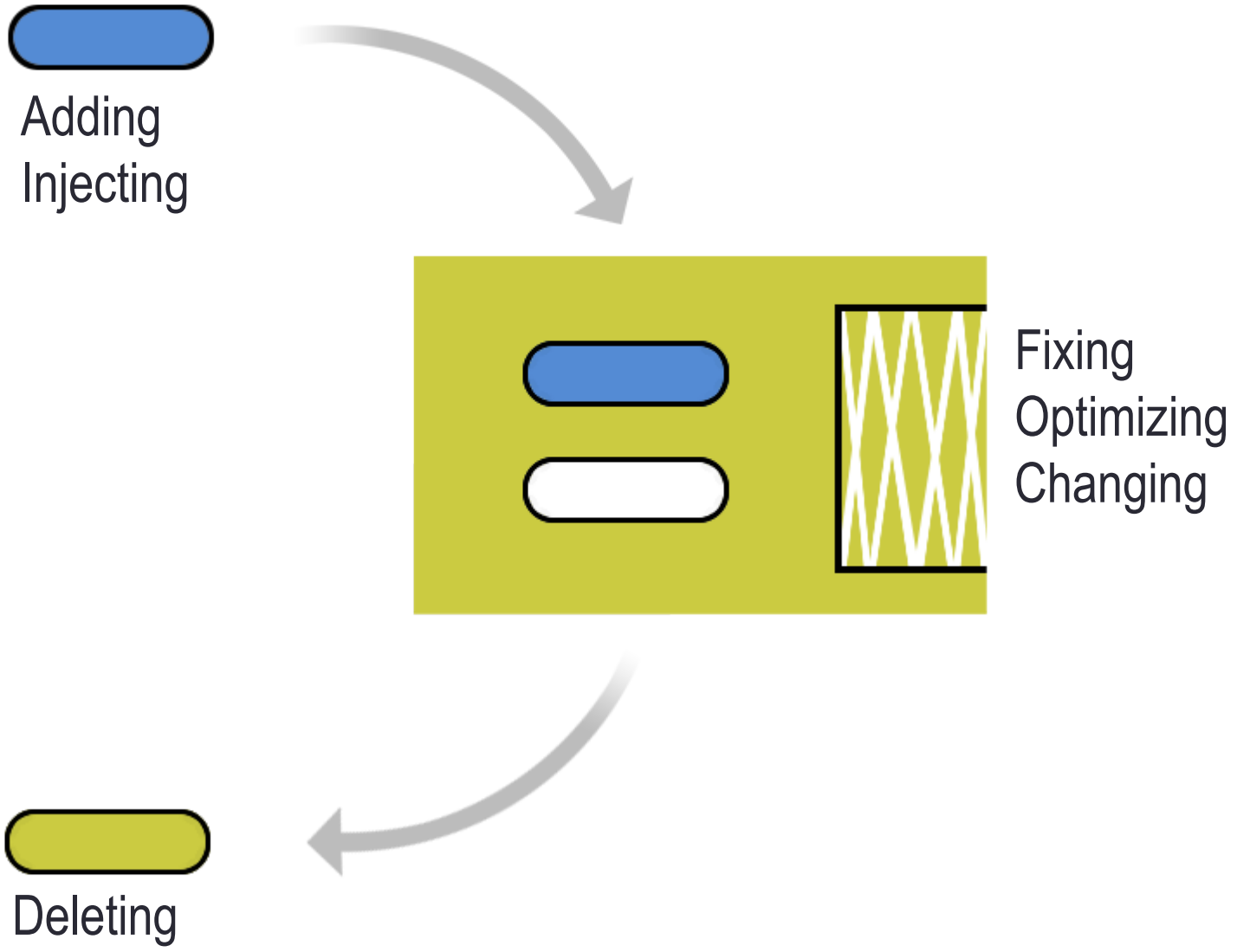
Relevant terms include:

- Static/Dynamic Linking
- Pairing
- Combining
- Utilizing
- Packaging
- Creating interdependency

Modification

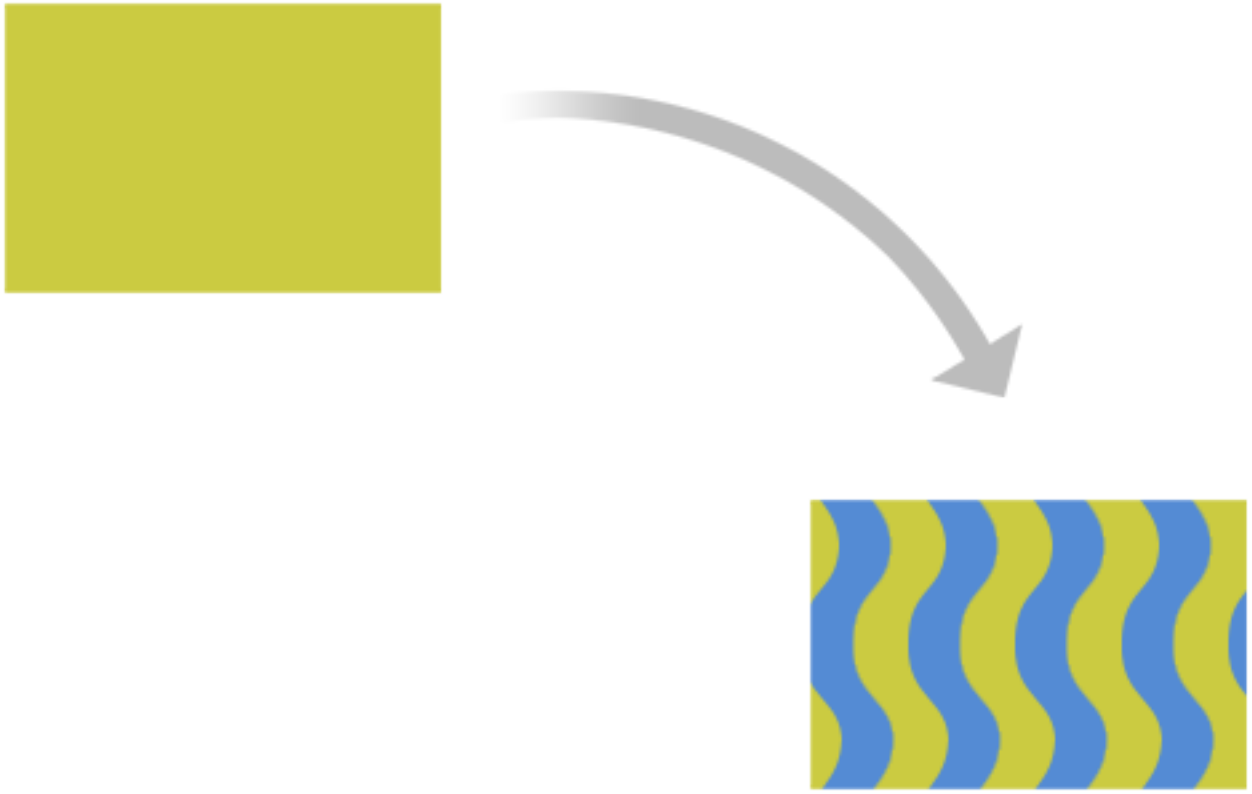
A developer may make changes to a FOSS component, including:

- Adding/injecting new code into the FOSS component
- Fixing, optimizing or making changes to the FOSS component
- Deleting or removing code



Translation

A developer may transform the code from one state to another.



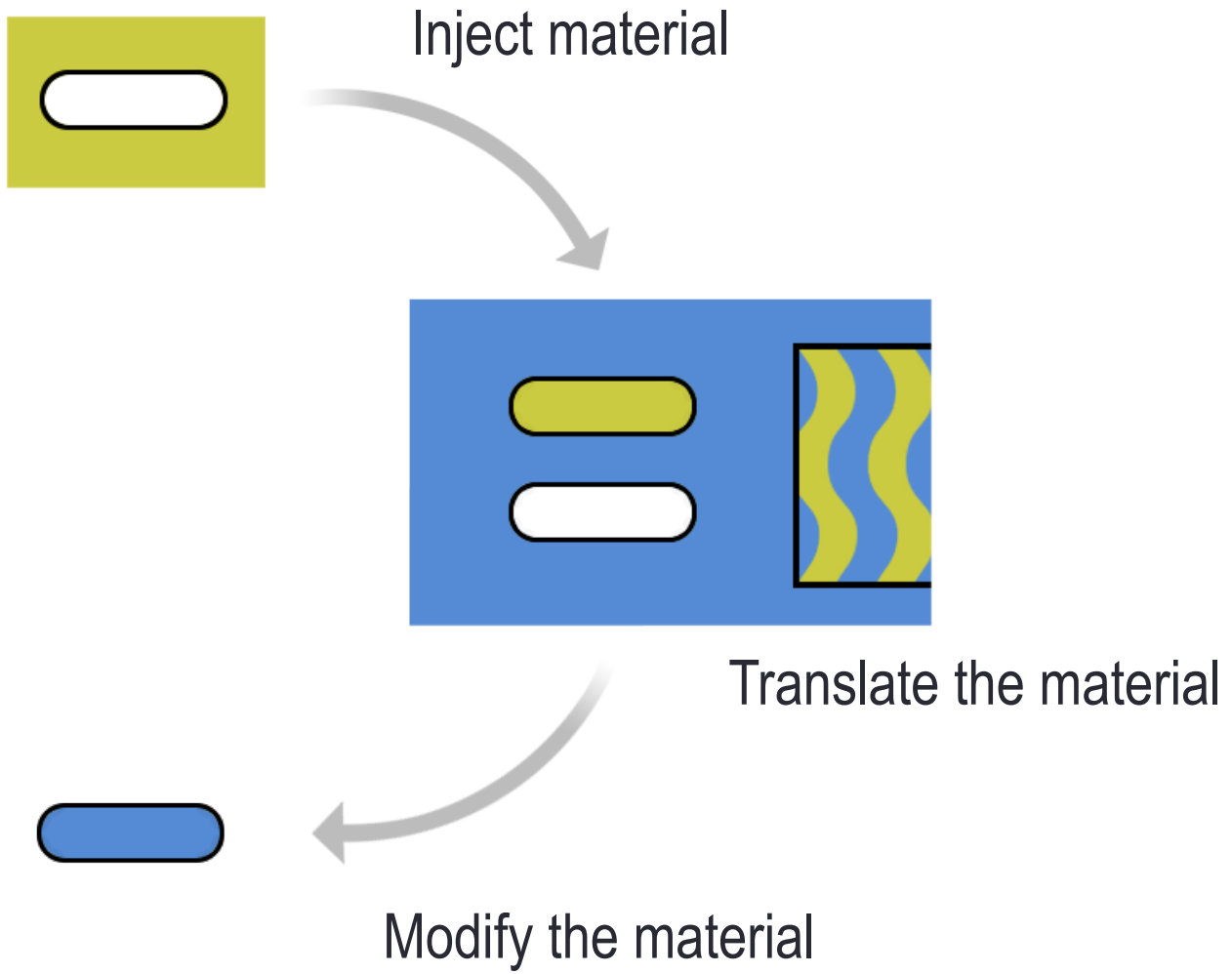
Examples include:

- Translating Chinese to English
- Converting C++ to Java
- Compiling into binary

Development Tools

Development tools may perform some of these operations behind the scenes.

For example, a tool may inject portions of its own code into output of the tool.



How is the FOSS component distributed?

OPENCHAIN

- Who receives the software?
  - Customer/Partner
  - Community project
  - Another legal entity within the business group (this may count as distribution)

- What format for delivery?
  - Source code delivery
  - Binary delivery
  - Pre-loaded onto hardware

Check Your Understanding

OPENCHAIN

- What is incorporation?
- What is linking?
- What is modification?
- What is translation?
- What factors are important in assessing a distribution?

CHAPTER 5

Running a FOSS Review

FOSS Review

OPENCHAIN

- After Program and Product Management and Engineers have reviewed proposed FOSS components for usefulness and quality, a review of the rights and obligations
  associated with the use of the selected components should be initiated
- A key element to a FOSS Compliance Program is a *FOSS Review* process. This process is where a company can analyze the FOSS software it uses and understand its rights and obligations
- The FOSS Review process includes the following steps:
  - Gather relevant information
  - Analyze and understand license obligations
  - Provide guidance compatible with company policy and business objectives

Initiating a FOSS Review

Anyone working with FOSS in the company should be able to initiate a FOSS Review, including Program or Product Managers, Engineers, and Legal.

Note: The process often starts when new FOSS-based software is selected by engineering or outside vendors.

What information do you need to gather?

OPENCHAIN

When analyzing FOSS usage, collect information about the identity of the FOSS component, its origin, and how the FOSS component will be used. This may include:

- Package name
- Status of the community around the package (activity, diverse membership, responsiveness)
- Version
- Download or source code URL
- Copyright owner
- License and License URL
- Attribution and other notices and URLs
- Description of modifications intended to be made

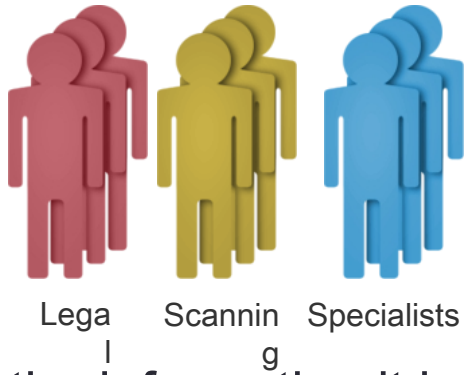
- List of dependencies
- Intended use in your product
- First product release that will include the package
- Location where the source code will be maintained
- Possible previous approvals in another context
- If from an external vendor:
- Development team's point of contact
- Copyright notices, attribution, source code for vendor modifications if needed to satisfy license obligations

FOSS Review Team

A FOSS Review team includes the company representatives that support, guide, coordinate and review the use of FOSS. These representatives may include:

- Legal to identify and evaluate license obligations
- Source code scanning and tooling support to help identify and track FOSS usage
- Engineering Specialists working with business interests, commercial licensing, export compliance, etc., who may be impacted by FOSS usage

Analyzing Proposed FOSS Usage



The FOSS Review team should assess the information it has gathered before providing guidance for issues. This may include scanning the code to confirm the accuracy of the information.

The FOSS Review team should consider:

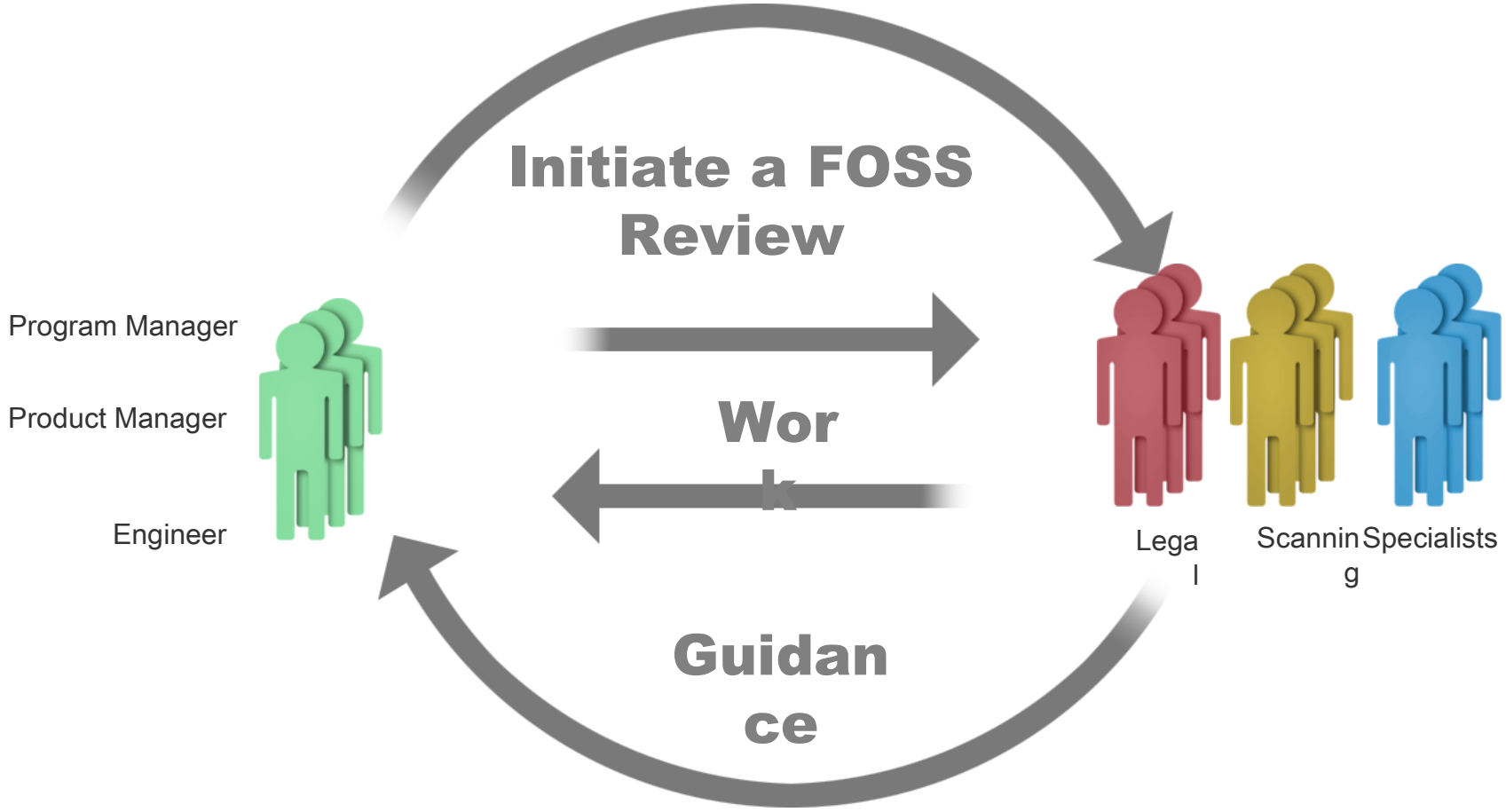
- Is the code and associated information complete, consistent and accurate?
- Does the declared license match what is in the code files?
- Does the license permit use with other components of the software?

Source Code Scanning Tools

OPENCHAIN

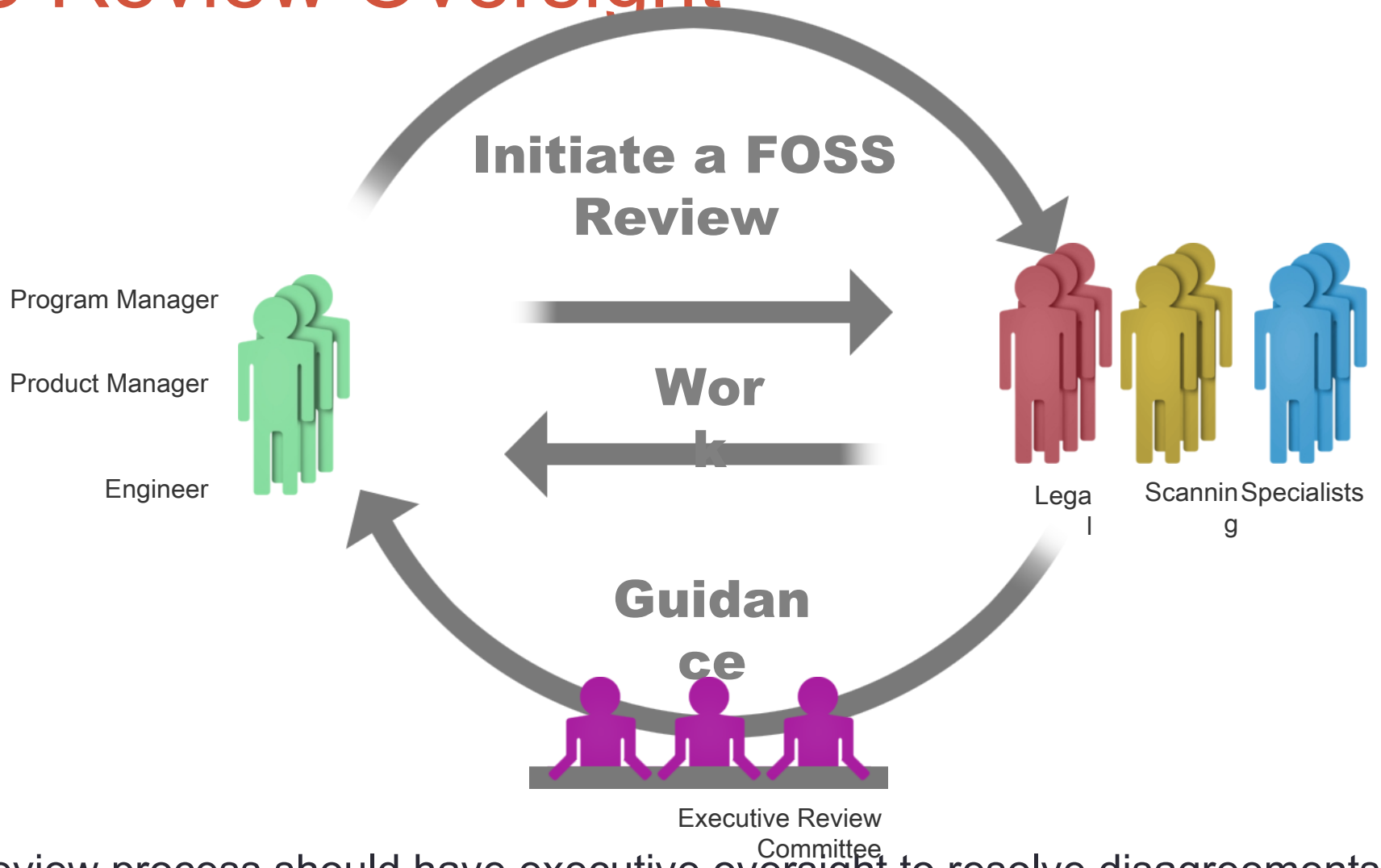
- There are many different automated source code scanning tools.
- All of the solutions address specific needs and - for that reason - none will solve all possible challenges
- Companies pick the solution most suited to their specific market area and product
- Many companies use both an automated tool and manual review
- A good example of freely available source code scanning tool is FOSSology, a project hosted by the Linux Foundation:
  https://www.fossology.org

Working through the FOSS Review



The FOSS Review process crosses disciplines, including engineering, business, and legal teams. It should be interactive to ensure all those groups correctly understand the issues and can create clear, shared guidance.

FOSS Review Oversight



The FOSS Review process should have executive oversight to resolve disagreements and approve the most important decisions.

Check Your Understanding

OPENCHAIN

- What is the purpose of a FOSS Review?
- What is the first action you should take if you want to use FOSS components?
- What should you do if you have a question about using FOSS?
- What kinds of information might you collect for a FOSS review?
- What information helps identify who is licensing the software?
- What additional information is important when reviewing a FOSS component from an outside vendor?
- What steps can be taken to assess the quality of information collected in a FOSS Review?

CHAPTER 6

End to End Compliance Management (Example Processes)

Introduction

- Compliance management is a set of actions that manages FOSS components used in products. Companies may have similar processes in place for proprietary components. FOSS components are called "Supplied Software" in the OpenChain specification.
- Such actions often include:
  - Identifying all the FOSS components used in Supplied Software
  - Identifying and tracking all obligations created by those components
  - Confirming that all obligations have been or will be met
- Small companies may use a simple checklist and enterprises a detailed process.

Example Small to Medium Company Checklist

OPENCHAIN

Ongoing Compliance Tasks:

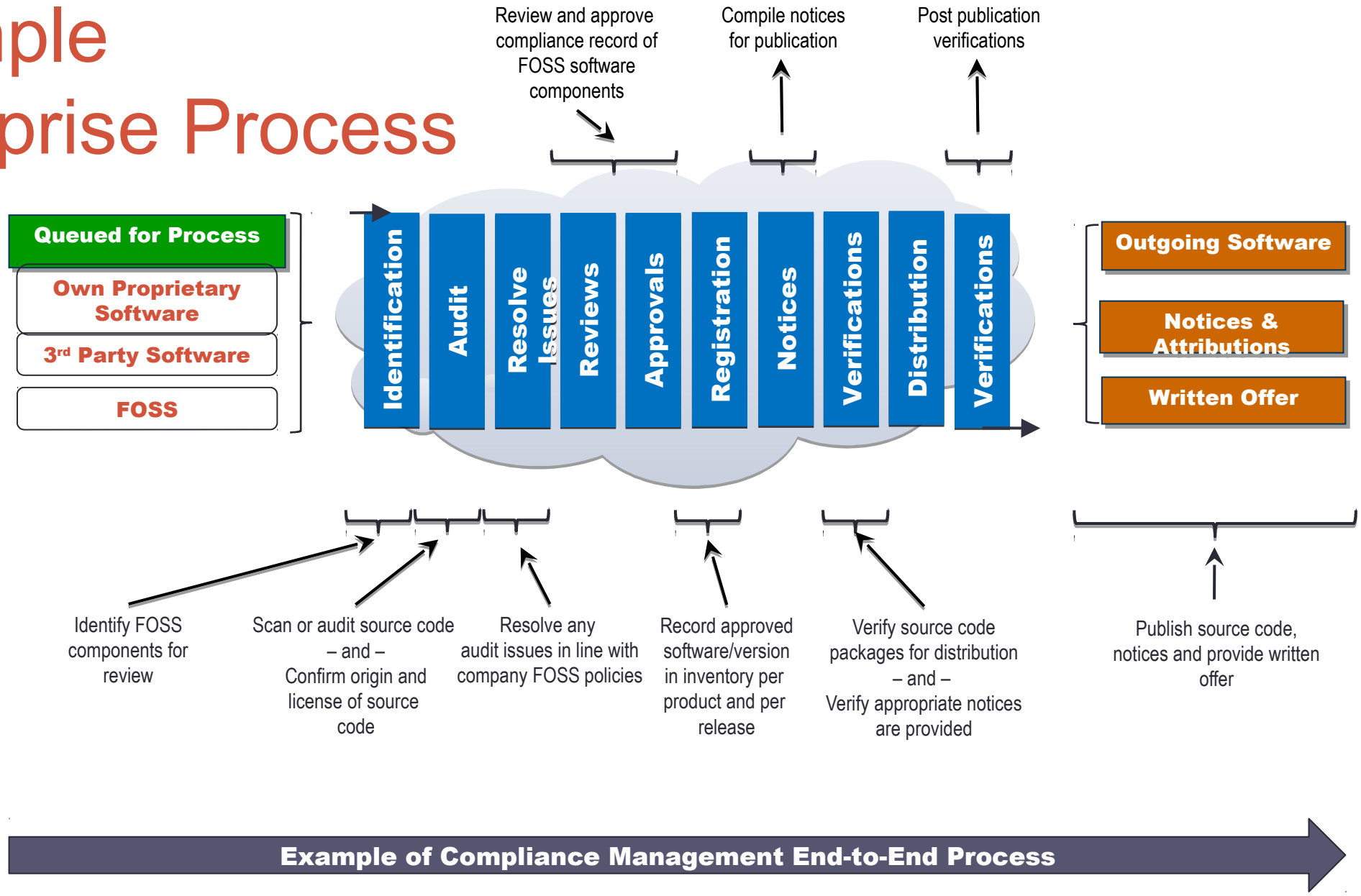
1. Discover all FOSS early in the procurement/development cycle
2. Review and Approve all FOSS components used
3. Verify the information necessary to satisfy FOSS obligations
4. Review and approve any outbound contributions to FOSS projects

Support Requirements:

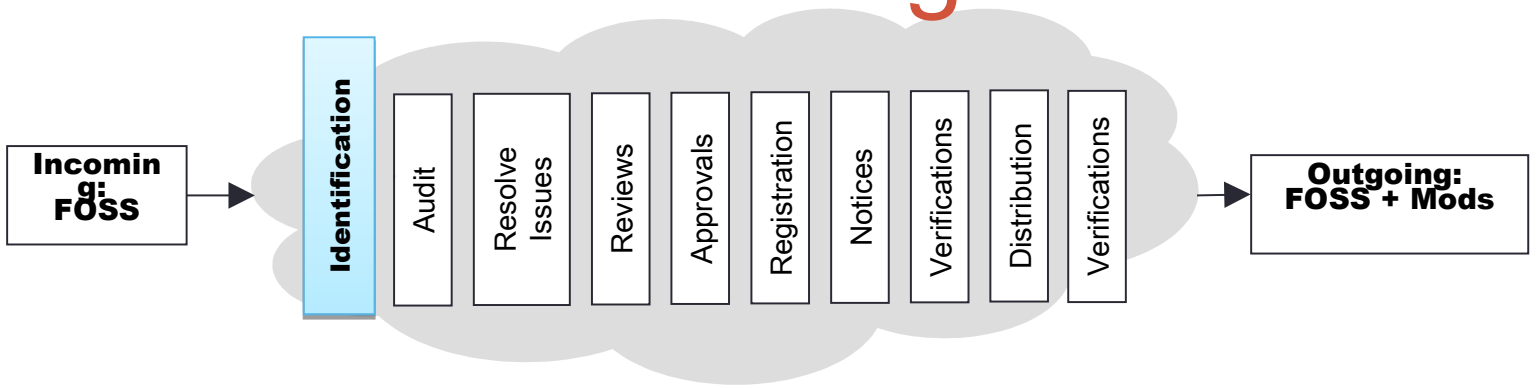
5. Ensure adequate compliance staffing and designate clear lines of responsibility
6. Adapt existing Business Processes to support the FOSS compliance program
7. Have training on the organization's FOSS policy available to everyone
8. Track progress of all FOSS compliance activities

You can get detailed checklists for these items here: `https://www.linuxfoundation.org/projects/opencompliance/self-assessment-compliance-checklist`

Example Enterprise Process



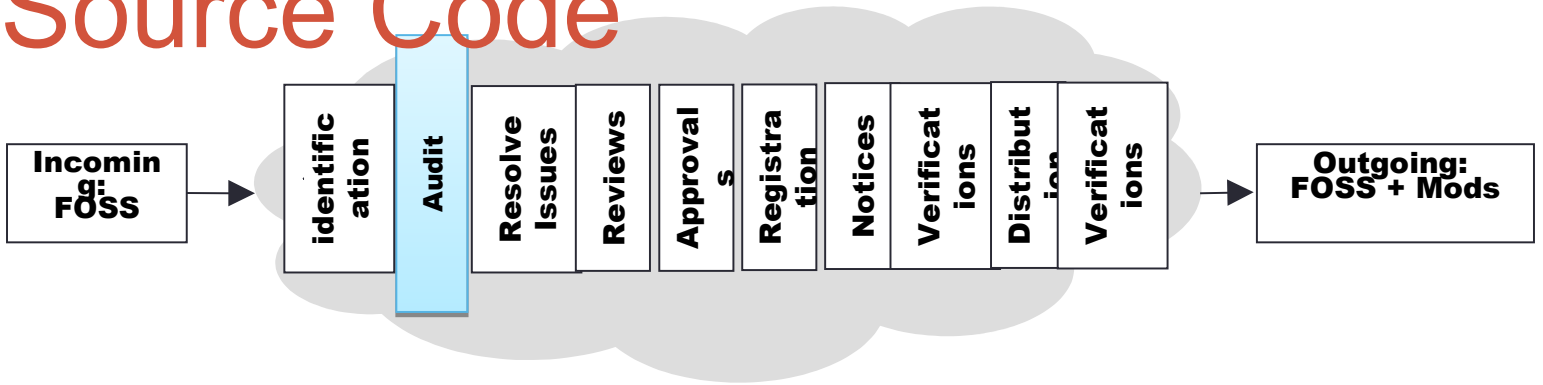
Identify and Track FOSS Usage



Identify FOSS components

- Steps:
  - Incoming requests from engineering
  - Scans of the software
  - Due diligence of 3rd-party software
  - Manual recognition of new components added to the repository
- Outcome:
  - A compliance record is created (or updated) for the FOSS
  - An audit is requested to review the source code with a scope a defined as exhaustive or limited according to FOSS policy requirements.

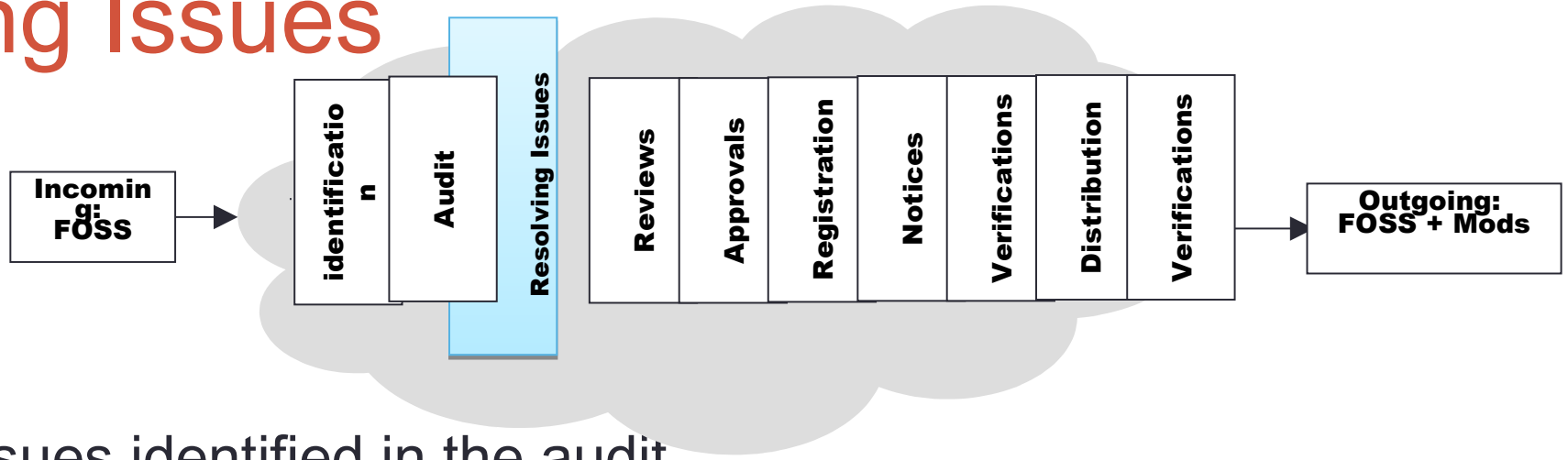
Auditing Source Code



Identify and audit FOSS licenses

- Steps:
  - Source code for the audit is identified
  - Source may be scanned by a software tool
  - “Hits” from the audit or scan are reviewed and verified as to the proper origin of the code
  - Audits or scans are performed iteratively based on the software development and release lifecycles
- Outcome:
  - An audit report identifying:
    1. The origins and licenses of the source code
    2. Issues that need resolving

Resolving Issues

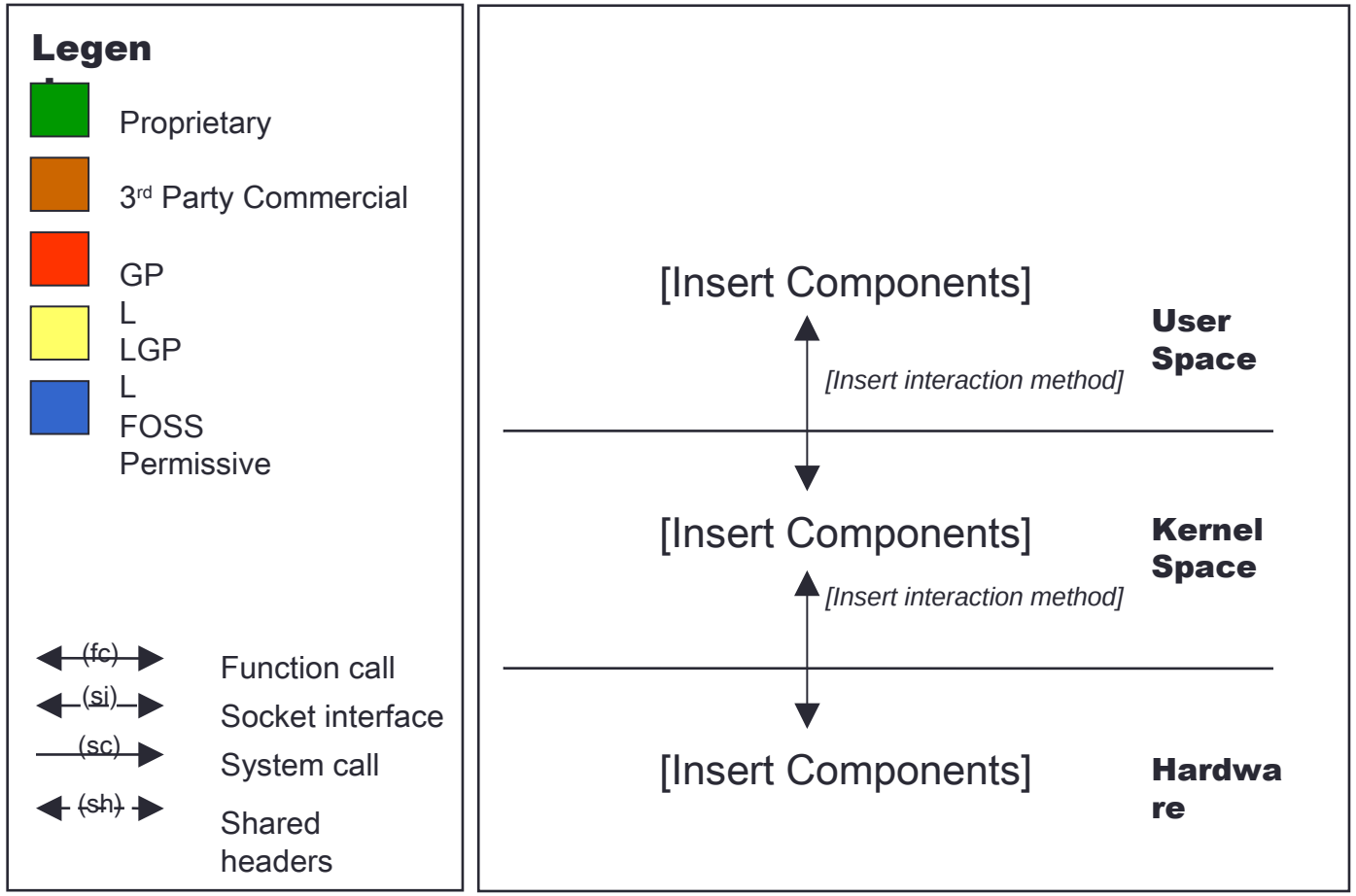


Resolve all issues identified in the audit

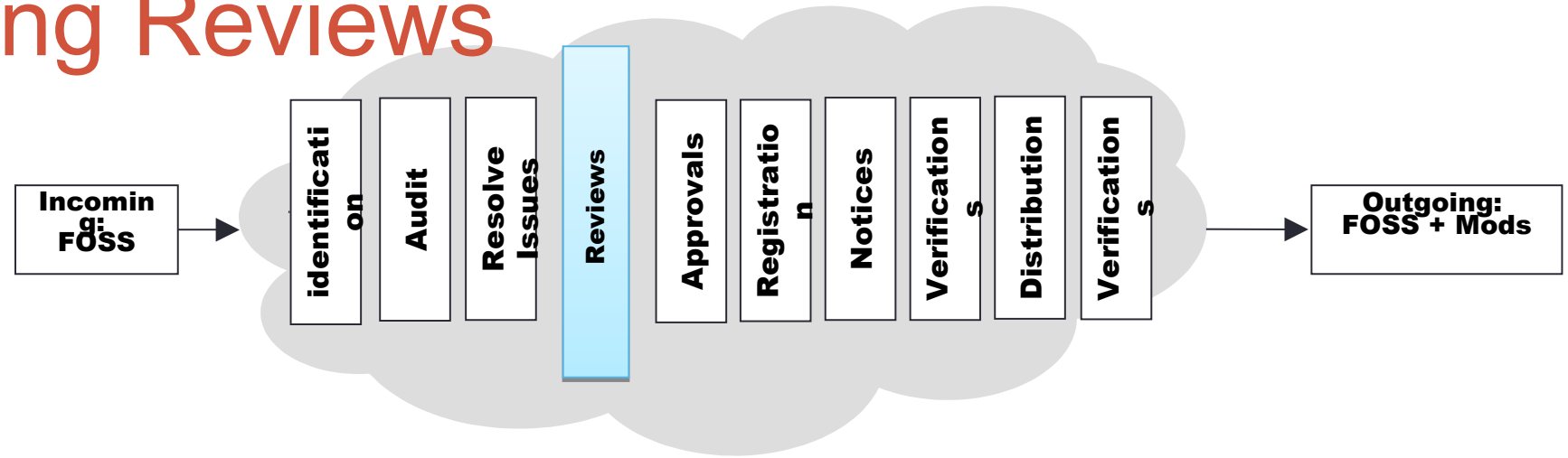
- Steps:
  - Provide feedback to the appropriate engineers to resolve issues in the audit report that conflict with your FOSS policy
  - The appropriate engineers then conduct FOSS Reviews on the relevant source code (see next slide for template)
- Outcome:

  A resolution for each of the flagged files in the report and a resolution for any flagged license conflict

Architecture Review (Example Template)



Performing Reviews



Review the resolved issues to confirm it matches your FOSS policy

Steps:

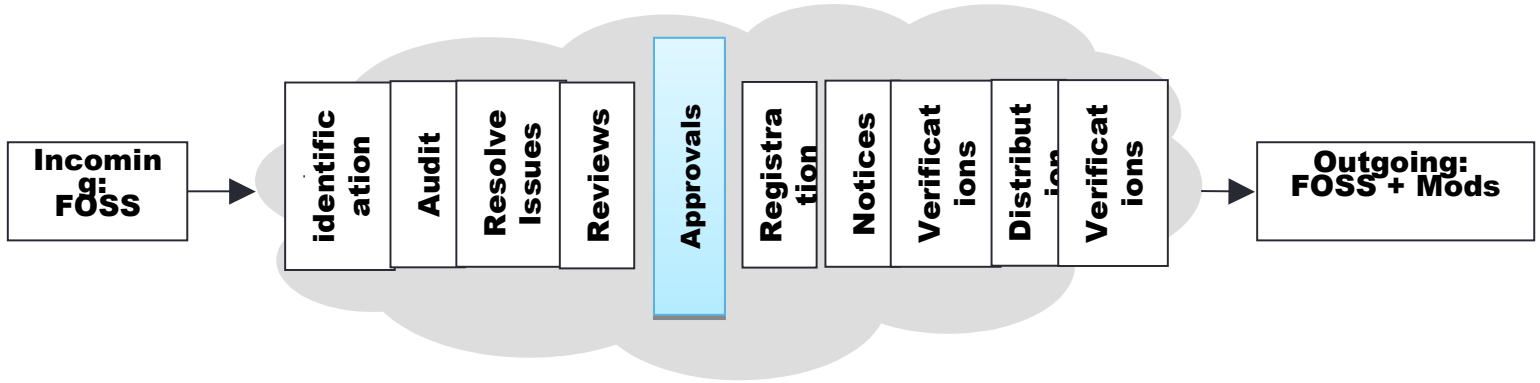
- Include appropriate authority levels in review staff
- Conduct review with reference to your FOSS policy

Outcome:

- Ensure the software in the audit report conforms with FOSS policies
- Preserve audit report findings and mark resolved issues as ready for the next step (i.e. Approval)

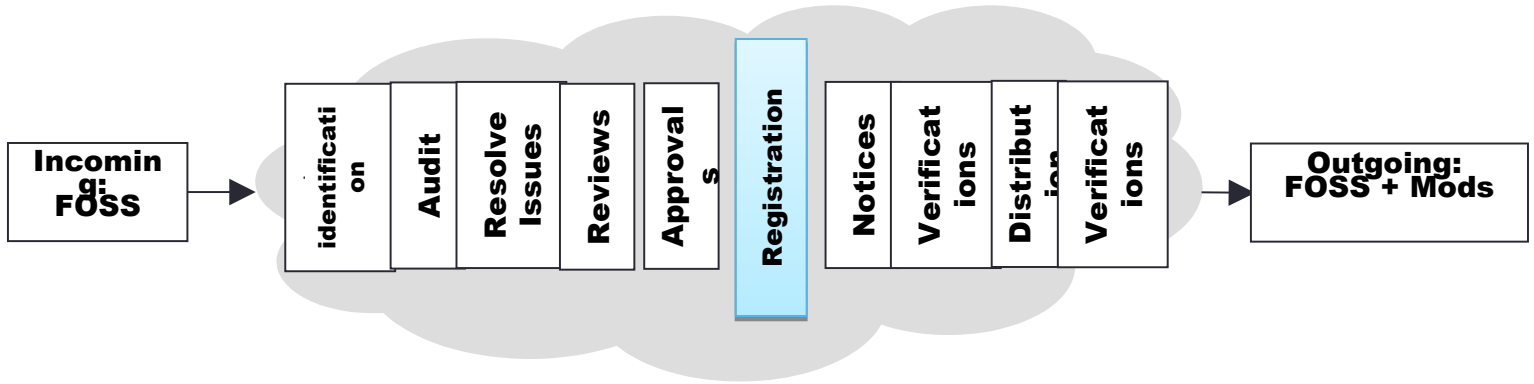
Approvals

- Based on the results of the software audit and review in previous steps, software may or may not be approved for use
- The approval should specify versions of approved FOSS components, the approved usage model for the component, and any other applicable obligations under the FOSS license
- Approvals should be made at appropriate authority levels

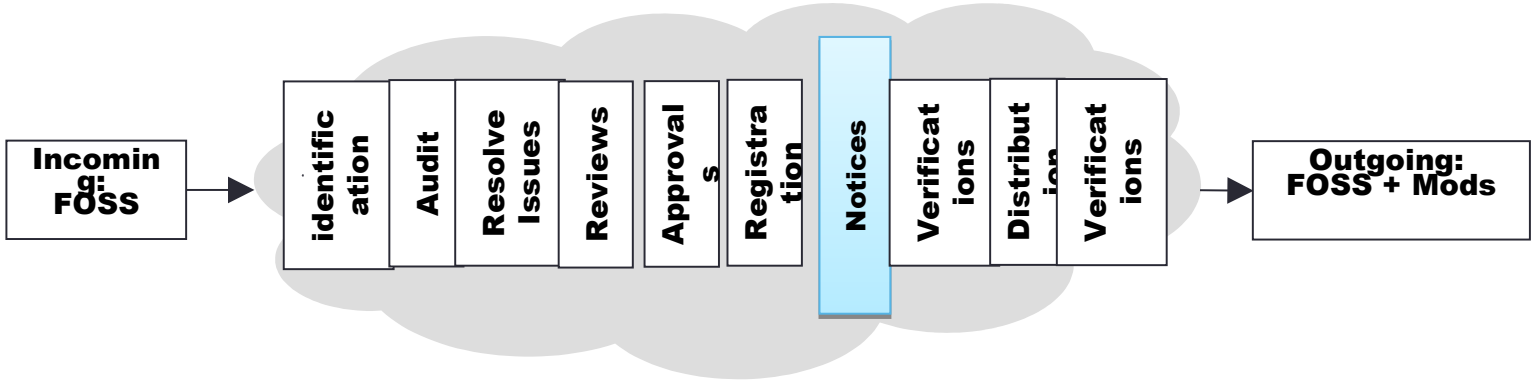


Registration / Approval Tracking

- Once a FOSS component has been approved for usage in a product, it should be added to the software inventory for that product
- The approval and its conditions should be registered in a tracking system
- The tracking system should make it clear that a new approval is needed for a new version of a FOSS component or if a new usage model is proposed

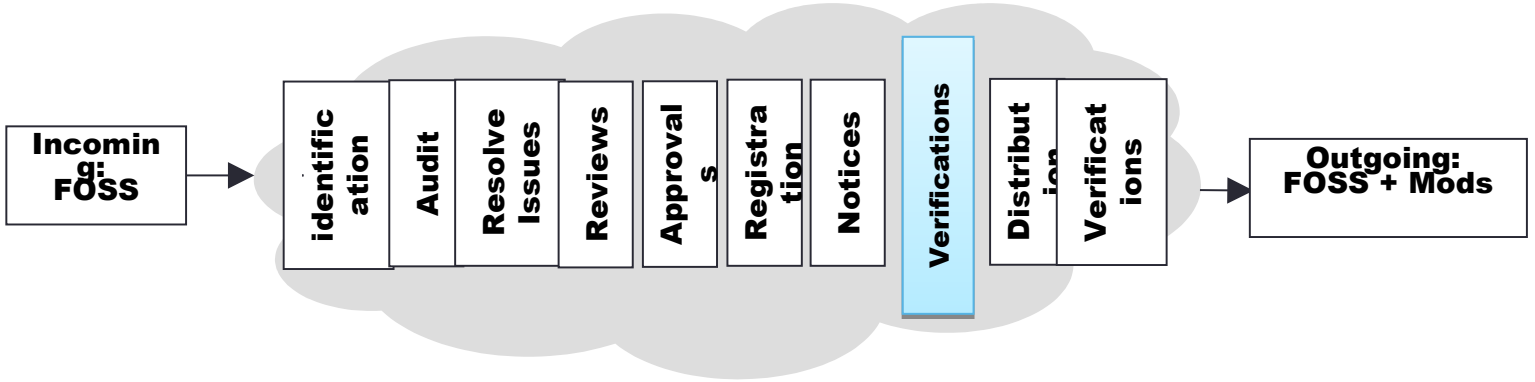


Notices



- Prepare appropriate notices for any FOSS used in a product release:
  - Acknowledge the use of FOSS by providing full copyright and attribution notices
  - Inform the end user of the product on how to obtain a copy of the FOSS source code (when applicable, for example in the case of GPL and LGPL)
  - Reproduce the entire text of the license agreements for the FOSS code included in the product as needed

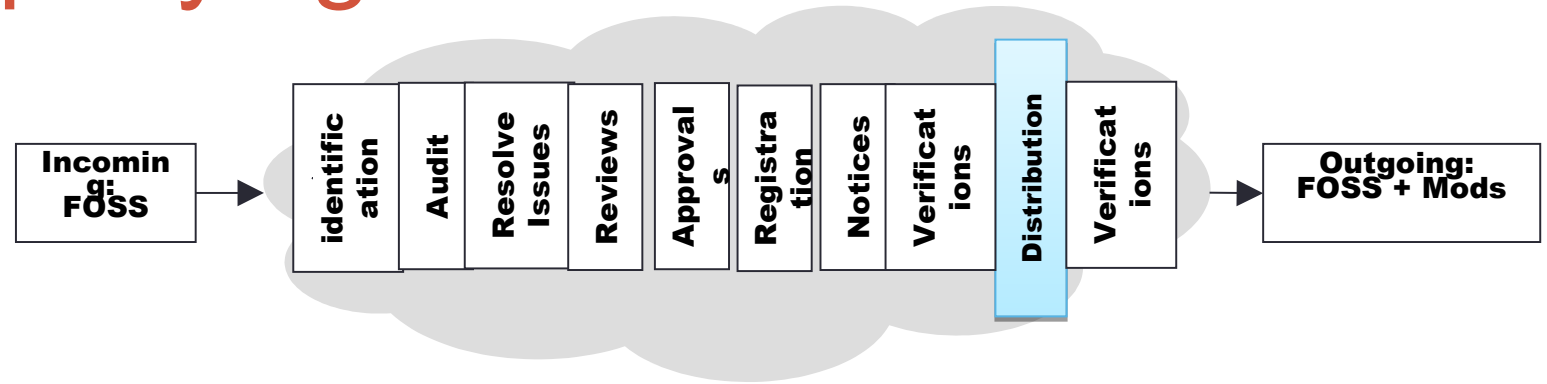
Pre-Distribution Verifications



Verify that distributed software has been reviewed and approved

- Steps:
  - Verify FOSS packages destined for distribution have been identified and approved
  - Verify the reviewed source code matches the binary equivalents shipping in the product
  - Verify all appropriate notices have been included to inform end-users of their right to request source code for identified FOSS
  - Verify compliance with other identified obligations
- Outcome:
  - The distribution package contains only software that has been reviewed and approved
  - "Distributed Compliance Artifacts" (as defined in the OpenChain specification), including appropriate notice files are included in the distribution package or other delivery method

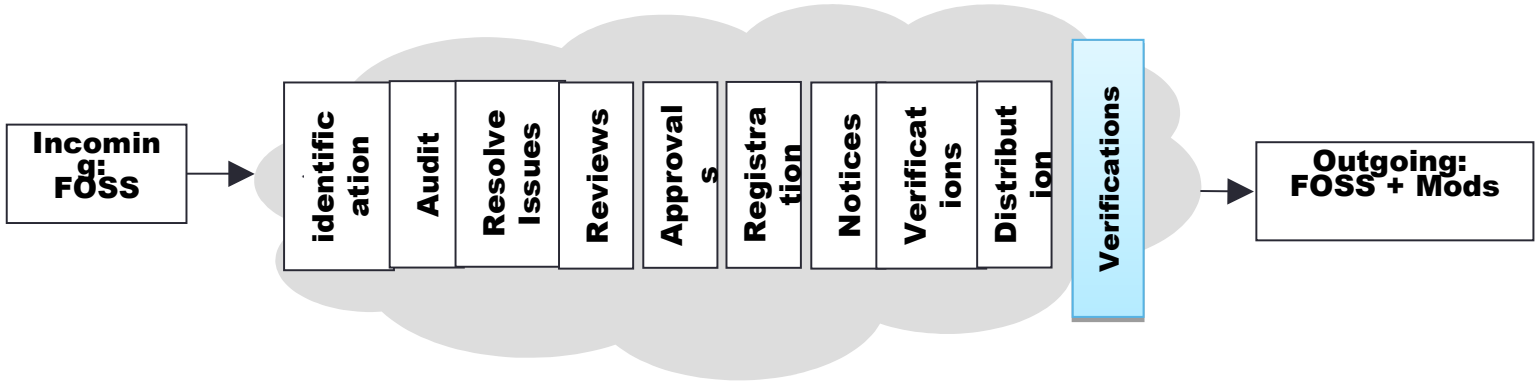
Accompanying Source Code Distribution



Provide accompanying source code as required

- Steps:
  - Provide accompanying source code along with any associated build tools and documentation (e.g., by uploading to a distribution website or including in the distribution package)
  - Accompanying source code is identified with labels as to which product and version to which it corresponds
- Outcome:
  - Obligations to provide accompanying source code are met

Final Verifications



Validate compliance with license obligations

- Steps:
  - Verify accompanying source code (if any) has been uploaded or distributed correctly
  - Verify uploaded or distributed source code corresponds to the same version that was approved
  - Verify notices have been properly published and made available
  - Verify other identified obligations are met
- Outcome:
  - Verified Distributed Compliance Artifacts are appropriately provided

Check Your Understanding

OPENCHAIN

- What is involved in compliance due diligence (for our example process, describe the steps at a high level)?
  - Identification
  - Audit source code
  - Resolving issues
  - Performing reviews
  - Approvals
  - Registration/approval tracking
  - Notices
  - Pre-distribution verifications
  - Accompanying source code distribution
  - Verification
- What does an architecture review look for?

CHAPTER 7

Avoiding Compliance Pitfalls

Compliance Pitfalls

OPENCHAIN

This chapter will describe some potential pitfalls to avoid in the compliance process:

1. Intellectual Property (IP) pitfalls
2. License Compliance pitfalls
3. Compliance Process pitfalls

Intellectual Property Pitfalls

OPENCHAIN

Type & Description	Discovery	Avoidance
Unplanned inclusion of copyleft FOSS into proprietary or 3rd party code: This type of failure occurs during the development process when engineers add FOSS code into source code that is intended to be proprietary in conflict with the FOSS policy.	This type of failure can be discovered by scanning or auditing the source code for possible matches with: • FOSS source code • Copyright notices Automated source code scanning tools may be used for this purpose	This type of failure can be avoided by: • Offering training to engineering staff about compliance issues, the different types of FOSS licenses and the implications of including FOSS in proprietary source code • Conducting regular source code scans or audits for all the source code in the build environment.

Intellectual Property Pitfalls

OPENCHAIN

Type & Description	Discovery	Avoidance
Unplanned linking of copyleft FOSS and proprietary source code: This type of failure occurs as a result of linking software with conflicting or incompatible licenses. The legal effect of linking is subject to debate in the FOSS community.	This type of failure can be discovered using a dependency tracking tool that shows any linking between different software components.	This type of failure can be avoided by: 1. Offering training to engineering staff to avoid linking software components with licenses that conflict with you FOSS policies which will take a position on these legal risks 2. Continuously running the dependency tracking tool over your build environment
Inclusion of proprietary code into copyleft FOSS through source code modifications	This type of failure can be discovered using the audits or scans to identify and analyze the source code you introduced to the FOSS component.	This type of failures can be avoided by: 1. Offering training to engineering staff 2. Conducting regular code audits

License Compliance Pitfalls

OPENCHAIN

Type & Description	Avoidance
Failure to Provide Accompanying Source Code/ appropriate license, attribution or notice information	This type of failure can be avoided by making source code capture and publishing a checklist item in the product release cycle before the product becomes available in the market place.
Providing the Incorrect Version of Accompanying Source Code	This type of failure can be avoided by adding a verification step into the compliance process to ensure that the accompanying source code for the binary version is being published.
Failure to Provide Accompanying Source Code for FOSS Component Modifications	This type of failure can be avoided by adding a verification step into the compliance process to ensure that source code for modifications are published, rather than only the original source code for the FOSS component

License Compliance Pitfalls

OPENCHAIN

Type & Description	Avoidance
Failure to mark FOSS Source Code Modifications: Failure to mark FOSS source code that has been changed as required by the FOSS license (or providing information about modifications which has an insufficient level of detail or clarity to satisfy the license)	This type of failure can be avoided by: 1. Adding source code modification marking as a verification step before releasing the source code 2. Offering training to engineering staff to ensure they update copyright markings or license information of all FOSS or proprietary software that is going to be released to the public

Compliance Process Failures

Description	Avoidance	Prevention
Failure by developers to seek approval to use FOSS	This type of failure can be avoided by offering training to Engineering staff on the company's FOSS policies and processes.	This type of failure can be prevented by: 1. Conducting periodic full scan for the software platform to detect any "undeclared" FOSS usage 2. Offering training to engineering staff on the company's FOSS policies and processes 3. Including compliance in the employees performance review
Failure to take the FOSS training	This type of failure can be avoided by ensuring that the completion of the FOSS training is part of the employee's professional development plan and it is monitored for completion as part of the performance review	This type of failure can be prevented by mandating engineering staff to take the FOSS training by a specific date

Compliance Process Failures

Description	Avoidance	Prevention
Failure to audit the source code	This type of failure can be avoided by: 1. Conducting periodic source code scans/audits 2. Ensuring that auditing is a milestone in the iterative development process	This type of failure can be prevented by: 1. Providing proper staffing as to not fall behind in schedule 2. Enforcing periodic audits
Failure to resolve the audit findings (analyzing the "hits" reported by a scan tool or audit)	This type of failure can be avoided by not allowing a compliance ticket to be resolved (i.e. closed) if the audit report is not finalized.	This type of failure can be prevented by implementing blocks in approvals in the FOSS compliance process
Failure to seek review of FOSS in a timely manner	This type of failure can be avoided by initiating FOSS Review requests early even if engineering did not yet decide on the adoption of the FOSS source code	This type of failure can be prevented through education

Ensure Compliance Prior to Product Shipment

- Companies must make compliance a priority before any product (in whatever form) ships
- Prioritizing compliance promotes:
  - More effective use of FOSS within your organization
  - Better relations with the FOSS community and FOSS organizations

Establishing Community Relationships

OPENCHAIN

As a company that uses FOSS in a commercial product, it is best to create and maintain a good relationship with the FOSS community - in particular, with the specific communities related to the FOSS projects you use and deploy in your commercial products.

In addition, good relationships with FOSS organizations can be very helpful in advising on best way to be compliant and also help out if you experience a compliance issue.

Good relationships with the software communities may also be helpful for two-way communication: upstreaming improvements and getting support from the software developers.

Check Your Understanding

- What types of pitfalls can occur in FOSS compliance?
- Give an example of an intellectual property failure.
- Give an example of a license compliance failure.
- Give an example of a compliance process failure.
- What are the benefits of prioritizing compliance?
- What are the benefits of maintaining a good community relationship?

CHAPTER 8

Developer Guidelines

Developer Guidelines

OPENCHAIN

- Select code from high quality, well supported FOSS communities
- Seek guidance
  - Request formal approval for each FOSS component you are using
  - Do not check un-reviewed code into any internal source tree
  - Request formal approval for outside contributions to FOSS projects
- Preserve existing licensing information
  - Do not remove or in any way disturb existing FOSS licensing copyrights or other licensing information from any FOSS components that you use. All copyright and licensing information is to remain intact in all FOSS components
  - Do not re-name FOSS components unless you are required to under the FOSS license (e.g., required renaming of modified versions)
- Gather and retain FOSS project information required for your FOSS review process

Anticipate Compliance Process Requirements

- Include time required to follow established FOSS policy in work plans
  - Follow the developer guidelines for using FOSS software, particularly incorporating or linking FOSS code into proprietary or third party source code or vice versa
  - Review architecture plans and avoid mixing components governed by incompatible FOSS licenses
- Always update compliance verification - for every product
  - Verify compliance on a product-by-product basis: Just because a FOSS package is approved for use in one product does not necessarily mean it will be approved for use in a second product
- And for every upgrade to newer versions of FOSS
  - Ensure that each new version of the same FOSS component is reviewed and approved
  - When you upgrade the version of a FOSS package, make sure that the license of the new version is the same as the license of the older used version (license changes can occur between version upgrades)
  - If a FOSS project's license changes, ensure that compliance records are updated and that the new license does not create a conflict

Compliance Process Applies to all FOSS components

OPENCHAIN

- In-bound software
  - Take steps to understand what FOSS is included in software delivered by suppliers
  - Evaluate your obligations for all of the software that will be included in your products
  - Always audit source code you received from your software providers or alternatively make it a company policy that software providers must deliver you a source code audit report for any source code you receive

Check Your Understanding

OPENCHAIN

- Name some general guidelines developers can practice when working with FOSS.
- Should you remove or alter FOSS license header information?
- Name some important steps in a compliance process.
- How can a new version of a previously-reviewed FOSS component create new compliance issues?
- What risks should you address with in-bound software?

Learn more through the free Compliance Basics for Developers hosted by the Linux Foundation at:
https://training.linuxfoundation.org/linux-courses/open-source-compliance-courses/ compliance-basics-for-developers

CHAPTER 9

Tooling Use Cases

Introduction

- Why we would need tools?
- First demand and process, then the tool
- A tool cannot provide (difficult) decisions
- Only data for decisions
- Many cases where expert knowledge is required

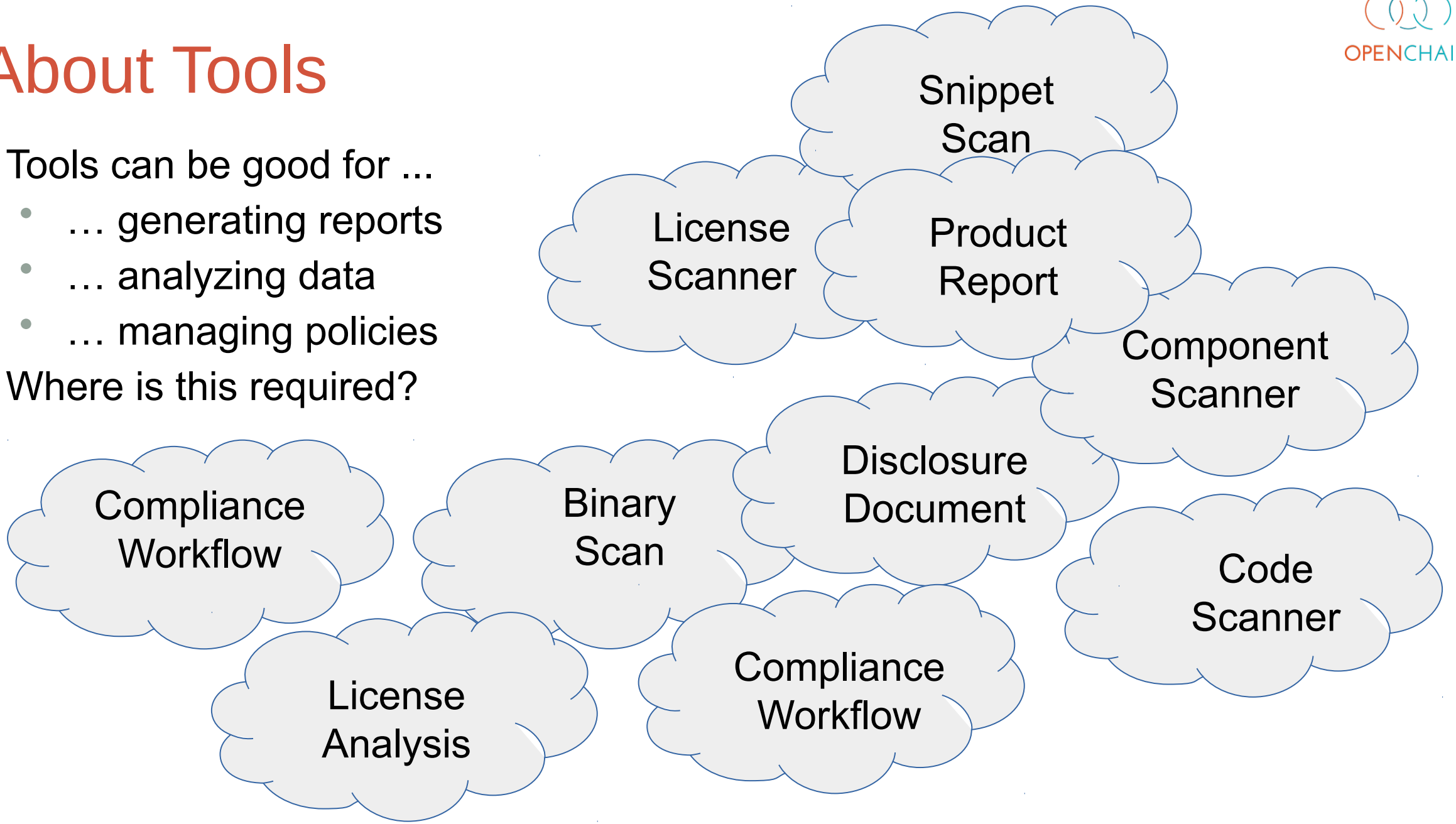
“A fool with a tool is still a fool” (from the hardware world)

About Tools

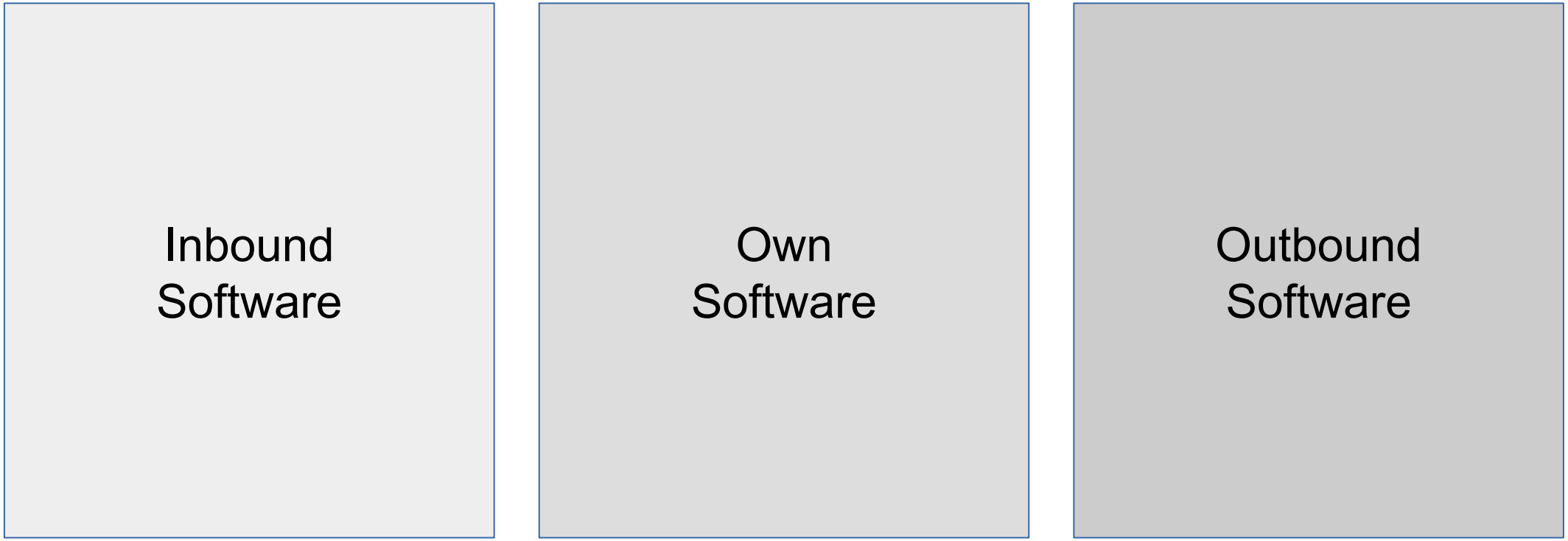
Tools can be good for ...

- … generating reports
- … analyzing data
- … managing policies

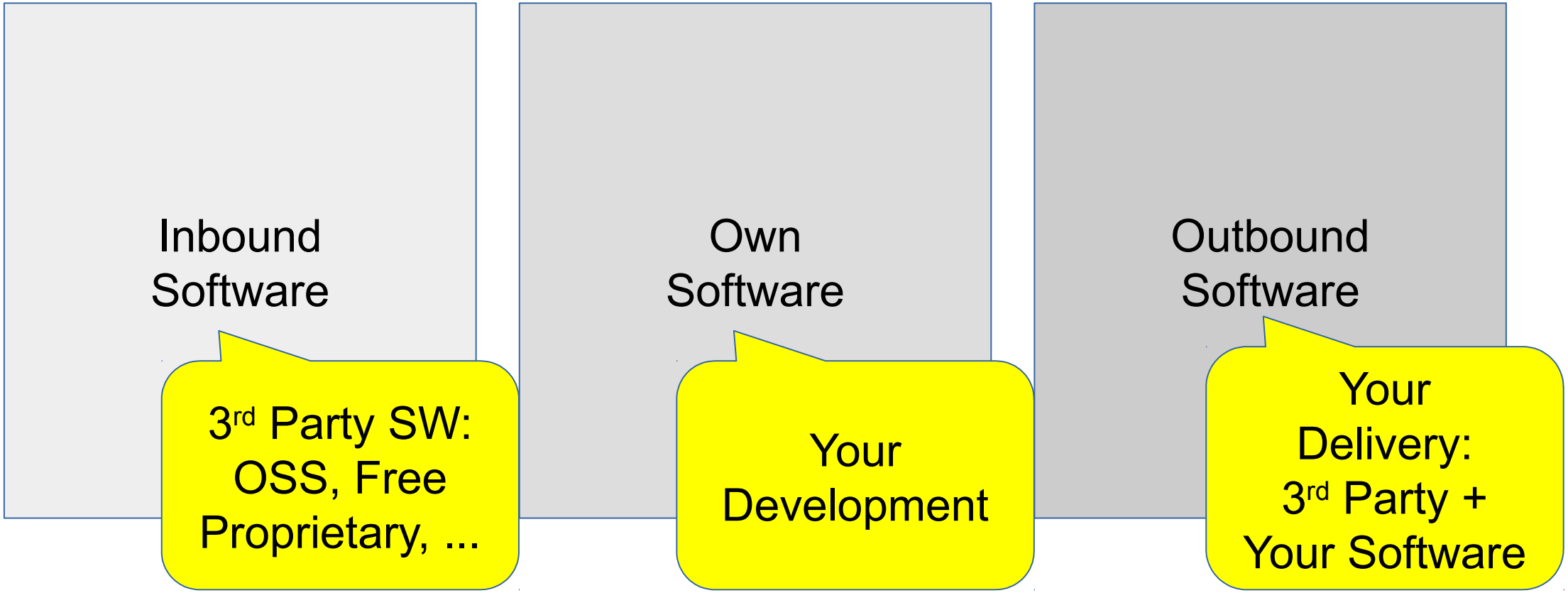
Where is this required?



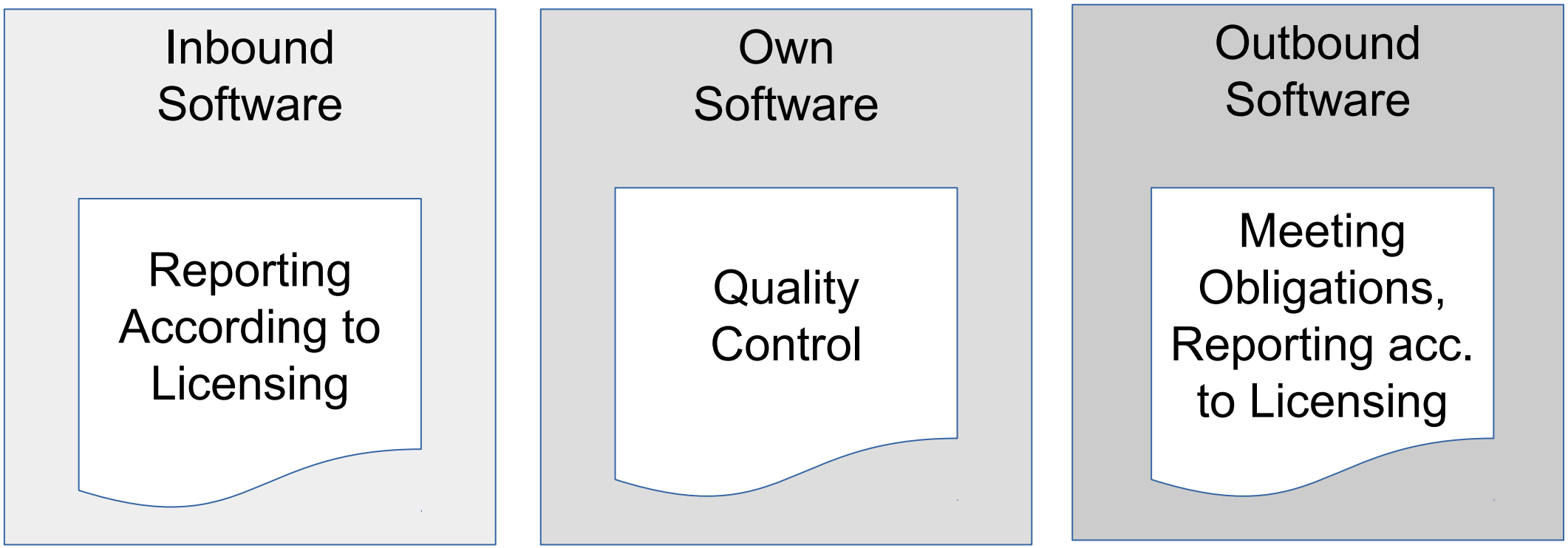
Software Situation



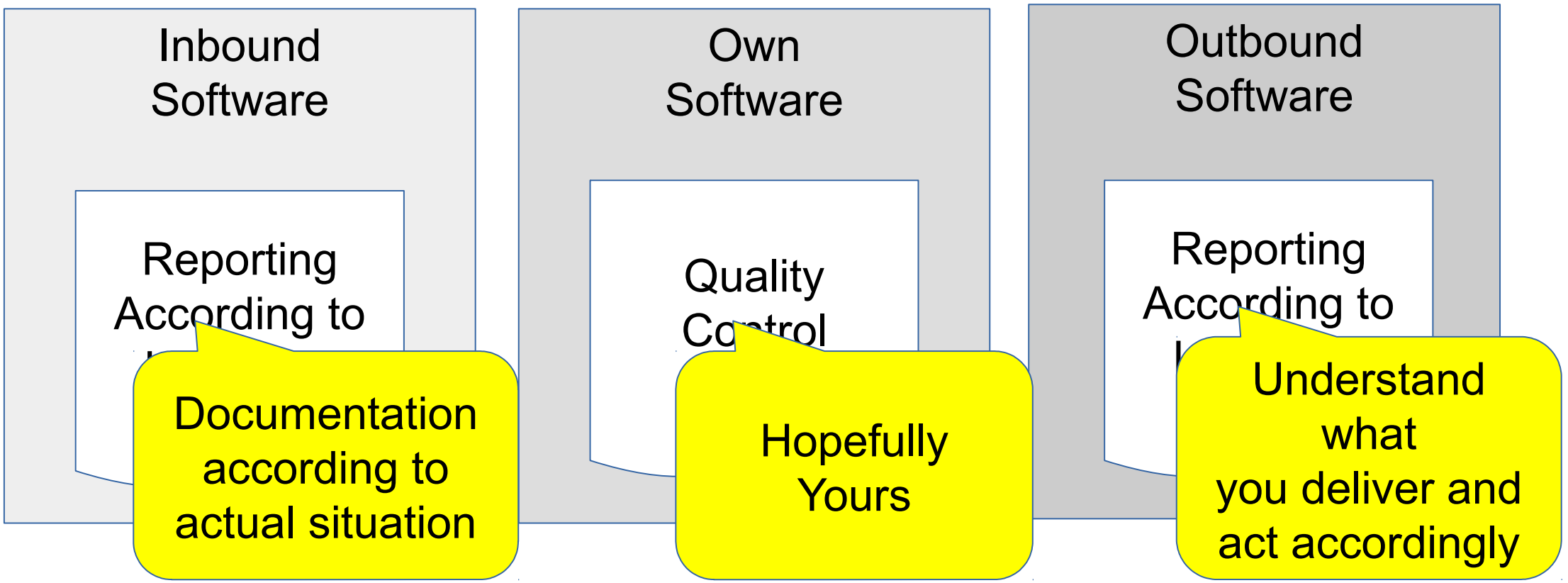
Software Situation – What it Means



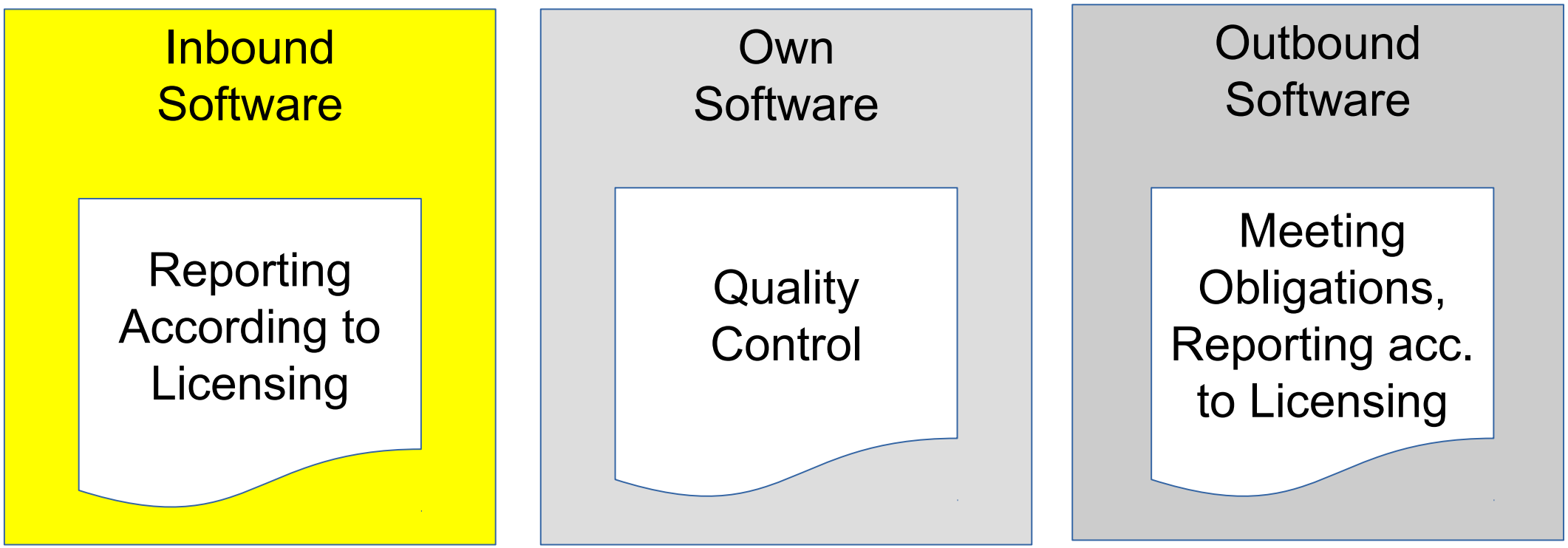
OSS License Compliance from 10k Feet



Again What this Means



Part I: Analysing Inbound



Understanding Inbound

- Determining which software is used (commercial + OSS actually)
- Because commercial software can contain OSS as well!
- OSS components involved and their involved licensing
- Identifying licenses
- Identifying authorships and copyrights
- Determining any further points from licensing obligations

How to Understand What is Inbound

- Depends on the software technology used
- Modern software projects use dependency management
  - Declaration of imports, dependencies, used libraries, etc.
  - Defined dependencies can be extracted
  - In some cases for OSS, used component source code can be extracted
- However, involved software can be also in form of binaries
  - Origin and contents of binaries must be determined
- “Manual dependencies”: commercial software added

Identifying Licensing within Inbound Software: Easy Cases

OPENCHAIN

- License, copying or notice document provided along with software
- At infrastructure, home page or project pages
  - e.g. Github or Sourceforge metadata
- Project definition file
  - e.g. in Java pom.xml
- Already provided license info
  - e.g debian-copyright or SPDX documentation

Identifying Licenses within Inbound Software: The Problem (1)

OPENCHAIN

- License proliferation
  - About 350 „main“ licenses exist
  - A lot more out there
  - Existing licenses come at new versions
- Licenses in different languages (e.g. the French CeCILL)
- License obligations must be understood
- Commercial licenses such as an EULA lack standardization

Identifying Licenses within Inbound Software: The Problem (2)

OPENCHAIN

- OSS = reuse
  - OSS components are not (always) homogeneous
  - If OSS exists, pull it from elsewhere
  - Code from many sources, different licensing
- Main license does not apply to all contents
  - If project does not enforce common licensing for all contributions
  - CLA: contributor license agreements

Identifying Licenses: The Fun (1)

OPENCHAIN

Identifying license statements is not straightforward ...

* See README and **LICENS**E files in bz/ directory
* for more information
* about bzip2 library code.
*/

This file is part of Jam - see jam.**c** for **Copyright** information.

---

* See **LICENS**E.qla2xxx for **copyright** and **licens**ing details.

/* **Licens**ing details are in the COPYING file accompanying popt source **distribut**ions, available from ftp://ftp.rpm.org/pub/rpm/dist. */

Copyright (c) Insight Software Consortium. All rights reserved.
See ITK**Copyright**.txt or http://www.itk.org/HTML/**Copyright**.htm for details.

---

* See wps_upnp.**c** for more details on **licens**ing and code history.

Identifying Licenses: The Fun (2)

… or just very difficult statements

```
* Copyright (c) 1998-1999 Some Company, Inc. All Rights Reserved.
*
* This software is the confidential and proprietary information of Some
* Company, Inc. ("Confidential Information").  You shall not
* disclose such Confidential Information and shall use it only in
* accordance with the terms of the license agreement you entered into
* with Some Company.
*
* Some Company  MAKES NO REPRESENTATIONS
* OR WARRANTIES ABOUT THE SUITABILITY OF THE
* SOFTWARE, EITHER EXPRESS OR IMPLIED,
* INCLUDING BUT NOT LIMITED TO THE ….
```

Identifying Copyright

Some licenses ask for copyright notice or author listing

- Resulting obligation of providing these
- Generally, there is software for these problems
- Challenge: wrongly expressed copyright statements

Identifying Copyright: Fun (again)

OPENCHAIN

Identifying copyright statements is not less fun:

```
Copyright by many contributors; see http://babel.eclipse.org/
---
 *  Original Code <s>Copyright (C) 1994, Jeff Hostetler, Spyglass, Inc.</s>
 *  Portions of Content-MD5 code <s>Copyright (C) 1993, 1994 by Carnegie Mellon
 *     University</s> (see Copyright below).
 *  Portions of Content-MD5 code <s>Copyright (C) 1991 Bell Communications
 *     Research, Inc. (Bellcore</s>) (see Copyright below).
 *  Portions extracted from mpack, John G. Myers - jgm+@cmu.edu
 *  Content-MD5 Code <s>contributed by Martin Hamilton (martin@net.lut.ac.uk)</s>
```

Identifying Licenses: Binaries

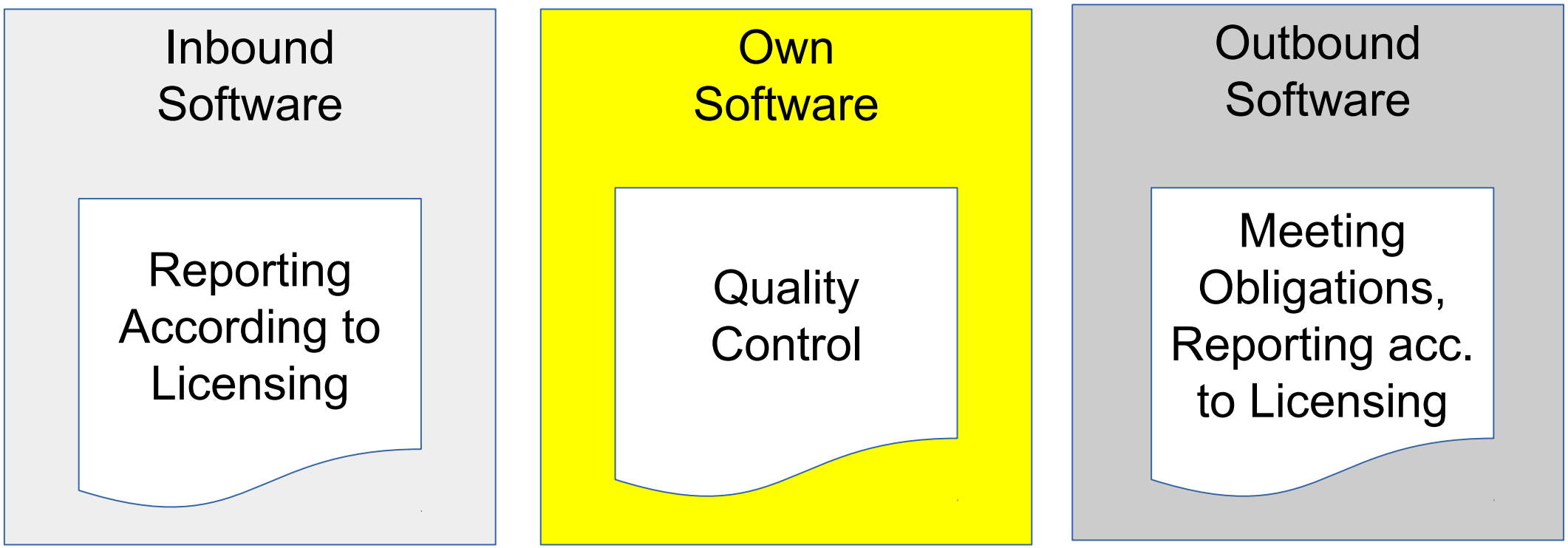
Binaries are compiled applications, libraries, software that can be used

- Binary = code translated from programming language to executable code by processor → information encoded
- Binaries can be part of an OSS component distribution
- Binaries can include OSS

How to understand what is contained in a binary?

- Main problem 1: different binary technologies
- Main problem 2: small variations, new binary

Part II: Your Own Software



What is the Issue with Your Software?

Sometimes, genuinely written software is expected
but “copy & paste” solution can be very near

- Open source projects are publicly available
- But also other files are valuable: scripts, icons, images, css files
- and code copied from Web sites for best practices and snippets

Copy paste of source code from the Internet in your code can be done:

- Respecting the author’s interests required: licensing, copyright
- Generally, reuse is good - opposed to reinventing the wheel

Code Scanning

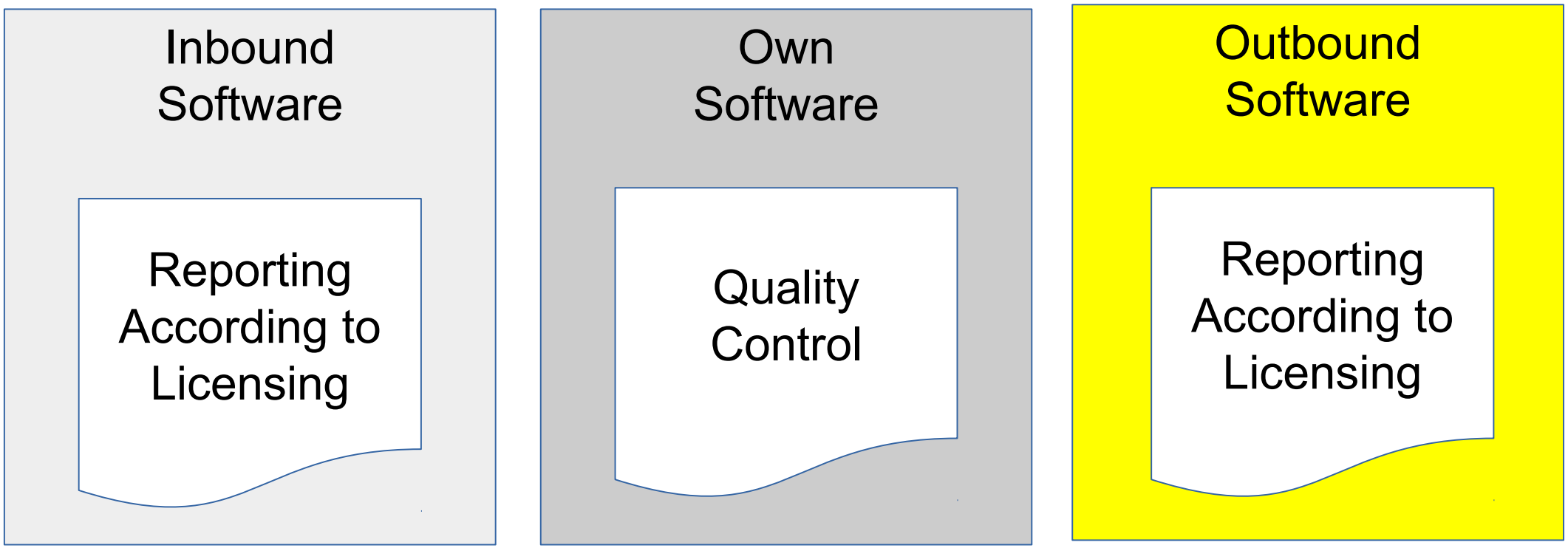
Good education and engineering codex can be solution

- Plain “copy & paste” of source code is bad practice anyway today
- Duplicated code reduces maintainability
- Engineers like clean dependency management

For all other cases

- Scanning tools for source code based on comparing text portions
- Using a database of already published source code (by other party)
- What is in Internet, tutorial code from vendors, Github
- Licensing: scan for licensing statements again

Part III: Outbound Software



Case 1: Distribution of OSS (1)

OPENCHAIN

Distributing OSS as part of product or project

- E.g. requires notice file
  - Listing all licenses, listing copyright notice
  - … as a basic and common license obligation
- E.g. written offer to provide the OSS code

Builds upon knowledge on

- Which OSS components are in (here comes the BOM!)
- Which licenses in there, copyright notices

Case 2: Quality Management

Project or product documentation can require, e.g.

- All tests passed
- But as well: all licenses checked?
  - For their obligations, for their compatibility
- Or: All OSS required material ready for distribution

Requires (as well)

- Which OSS components are in
- Which licenses in there, copyright notices

Case 3: Ensuring Distribution Rights

OPENCHAIN

Some licenses are not compatible

- That is life, for example GPL <-> EPL incompatibility
- *Distribution based on GPL works and EPL works: maybe a problem*

Some license statements are ambiguous

- For example „Licensed under BSD”
- *Requires legal decision how did you decide this statement*

Besides Delivering, Internal Work

Some license statements need documentation

- For example: „for license conditions, see Web site”
- *Web site needs to be archived*

Some licenses are not compatible with the business case

- E.g. Start up implements medical analysis algorithm after years of research, danger of being copied by market leaders
- *License obligations need to be compatible with business goals*

Excursus: Not OSS only, all 3rd Parties

Also with commercial software, appropriate licensing must be ensured:

- Does contract cover rights for intended commercial use?
- Where is the contract by the way?

Ensuring distribution obligations is required, for example:

- Documentation of distribution
- Time- / volume-limited licensing
- Logo printed on box necessary

BOM Documentation (1)

BOM: „Bill of Material”

- It is a general question what is in the delivery
- Understand the nature of the delivery (How much OSS?)
- Understand potential issues (IP)
- How else to ensure license compliance?
- Basics of supply chain issues actually apply also to software
- Software Package Data Exchange (SPDX) specifies one implementation how to express a BOM of a software package [1]

[1] https://spdx.org/

BOM Documentation (2)

Bill of material can be general obligation, for example at:

- USA: Cyber Supply Chain Management and Transparency Act of 2014
- Germany: KRITIS: BSI-Kritisverordnung [2]
  - Obliged to report service disturbances
  - Obliged to implement information security
  - Requires knowledge about BOM

[2] https://www.bmi.bund.de/SharedDocs/pressemitteilungen/DE/2017/06/nis-richtlinie.html

Your Own Software as OSS (1)

OPENCHAIN

Yes, it is true: sometimes software developers want to publish their work

- Excursus: Motivation 3.0 [3]
- How to publish? - A process topic
- But documentation is required (besides the publication)
  - What are the involved licenses
  - What is the own license
  - Are formal aspects met?

[3] https://www.youtube.com/watch?v=u6XAPnuFjJc

Your Own Software as OSS (2)

Analysis here has the goal to

- Confirm involved OSS licensing, business compatible?
- Identify dependencies and binaries
- Checking if all the source code is of our origin?

General quality points (including, but not limited to):

- Do all files have headers? (disclaimers for config files)
- Do all files have copyright and authorship statements
- Is the documentation of the licensing appropriate?

Summary of Tool Support

OPENCHAIN

Tools are there, but requirements and purpose require understanding

- First comes the definition of what is needed and then the tool
- Tools are there for analysis, reporting and management

Different tools serve different purposes

- Requires integration of different functions
- Integration poses classic IT problems
- Interfaces must be understood to avoid manual effort

CHAPTER 10

Tooling Types

Overview

Main types of tools in the area of license compliance include (but are not limited to):

- Source code scanning
- License scanning
- Binary scanning
- Dev Ops integration
- Component management

1. License Scanner

OPENCHAIN

License Scanner: Introduction

- Purpose:
  - Identifies licenses and license relevant statements
- Other Identifications:
  - Copyright statements, author statements, acknowledgements
- Also of interest:
  - Export control statements, more static code analysis

License Scanner: Solved Problem

OPENCHAIN

Problem: Identify licensing in Open Source Software packages

- Licensing in Open Source Software
  - Licensing of OSS can be heterogeneous, different licensing applies to parts of OSS
  - Licensing statements are not uniform
  - Many licenses exist, number growing

Tool based licensing identification required for complicated licensing situations

License Scanner: Technical

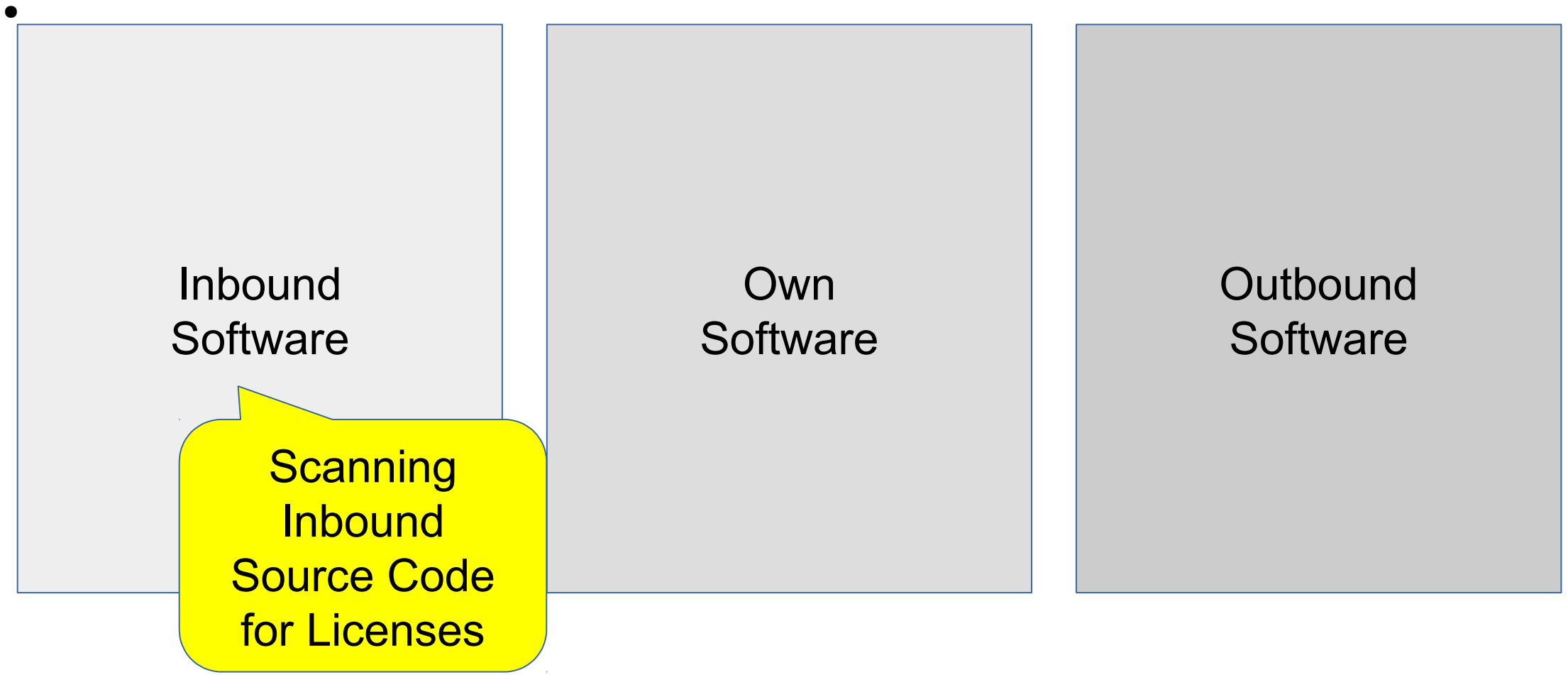
Mode of operation: Tool searches in content
for license relevant keywords, phrases, license texts

- Searching in every file of software uploaded: requires source code distribution
- Different approaches can be applied: regular expressions, text comparison, phrase collection
- Requires database of license texts, licensing statements
- Comparison with existing license texts enables exact identification
- Licensing information can summarized for open source packages

License Scanner: More Remarks

- License scanning does not require huge database
- However, updates are necessary as licensing statements evolve and new licenses are still created
- Identified licensing information of a software package can be exchanged using SPDX files
- Approach makes sense for OSS licenses, commercial licensing is even more heterogeneous
- License identification precision depends on available licensing information and may require expert knowledge for analysis

License Scanner: Main Usage



2. Binary Scanner

OPENCHAIN

Binary Scanner: Introduction

- Purpose:
  - Identifies used software packages in software binaries
- Other identifications:
  - Can also determine the versions of software packages
- Also of interest:
  - Identifying used software packages for creating the binary also enables identification of vulnerabilities

Binary Scanner: Solved Problem

OPENCHAIN

Problem: A binary is comprised of different software packages, but if not declared, not obvious to determine

- Applies in compiled programming languages: programming language code is translated (=compiled) into machine executable code (machine = processor)
- Script languages (e.g. JavaScript) are not compiled
- Binaries are usually not readable, understanding contents difficult
- However, identification of contents can be inevitable for understanding required license compliance tasks

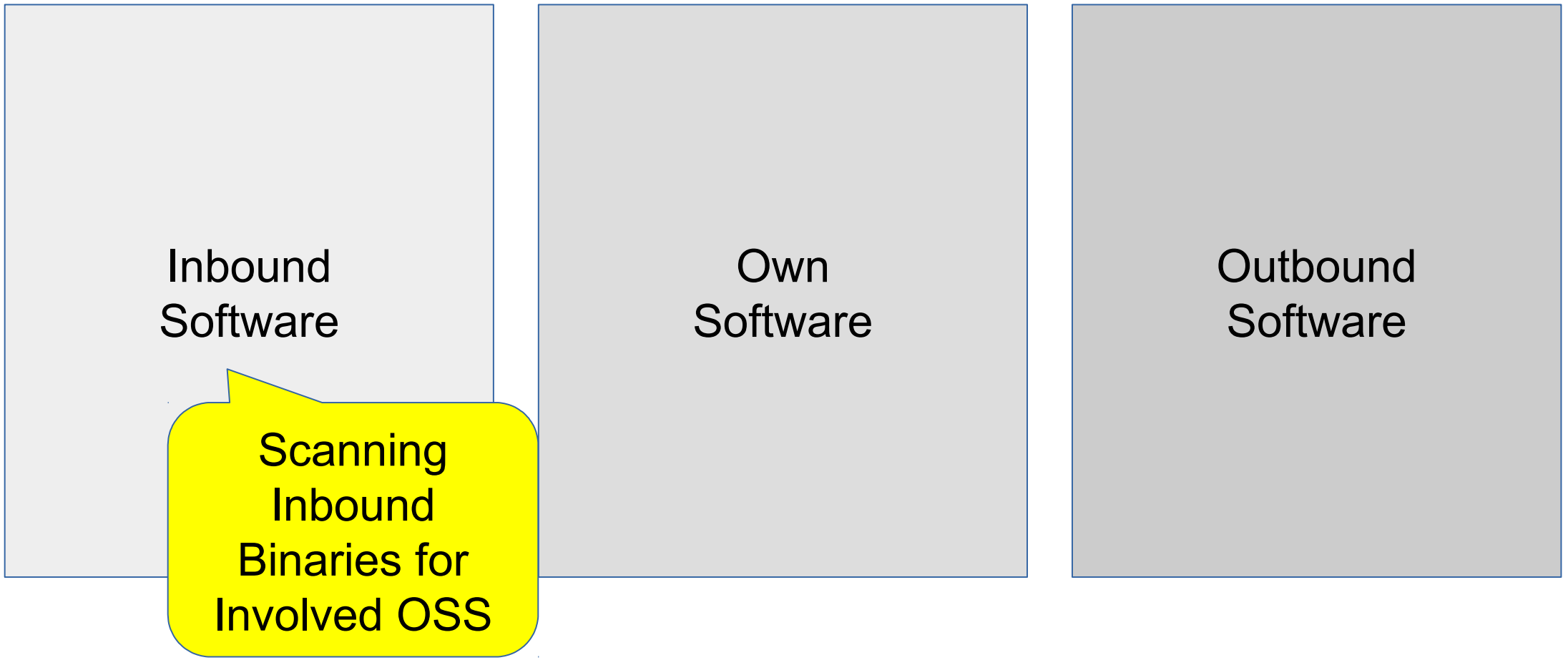
Binary Scanner: Technical

- Compiled machine language can contain characteristic elements
- For example used string variables (=text)
  or other content compiled into the binary
- Simpler method: capturing file names,
  or for run-time code (e.g. Java): method and field names
- Requires database of mapping
  from source code to resulting artefacts in binary

Binary Scanner: More Remarks

- Binary scanning is a heuristic,
  secure mapping not supported for every possible binary
- Topic connected with reproducible builds
  - (then, binaries can be compared more efficiently)
- Database requires updates because,
  because new software is published every day
  - (similar with source code scanning)

Binary Scanner: Main Usage



3. Source Code Scanner

OPENCHAIN

Source Code Scanner: Introduction

OPENCHAIN

- Purpose:
  - Can identify published origin of source code and other files
- Other Identifications:
  - Icons, images, style descriptions, XML schemes, documentation
- Also of interest:
  - Programming examples, from blogs and best practise Websites

Source Code Scanner: Solved Problem

Problem: how to understand that source code or other files have been taken from elsewhere, not self-created, and not declared

- If "own" software is not entirely own software and not understood:
  - Missing rights for business case in "own" software
  - But distribution requires distribution rights are available
  - Identification of origin is first step to understand available rights

Source Code Scanner: Technical

OPENCHAIN

Mode of operation: upload source code or just files or fingerprints of it, get origin in case it is captured by database

- File contents are compared
  with contents from (huge) database of published contents
- Fingerprinting of file contents (“hashing”)
  allow for accelerated search and storage in database
- Not only coverage of entire files, but fragments of it
- Database requires updates: every day new published OSS
- Content is large (e.g. the entire GitHub)

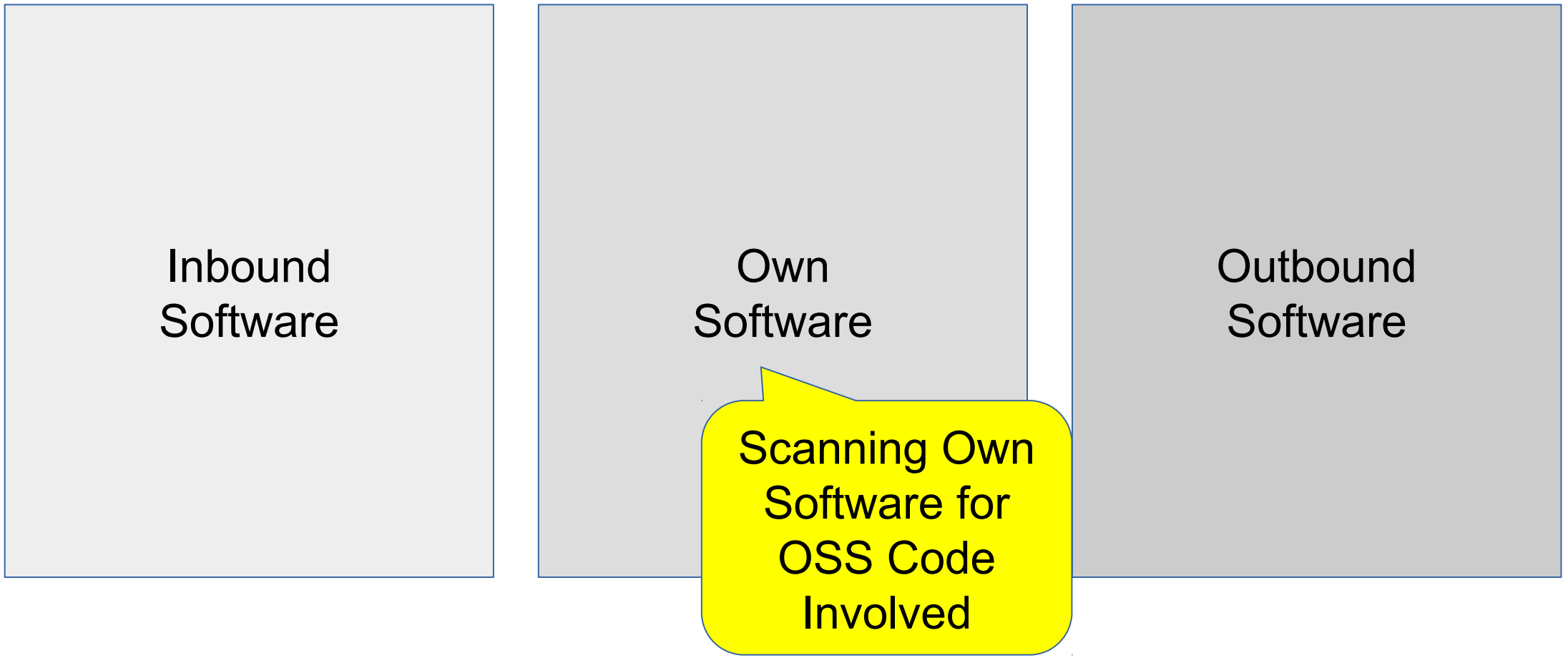
Source Code Scanner: More Remarks

OPENCHAIN

- Once origin of source is identified, more metadata can be made available:
  - Licensing
  - Vulnerabilities
- Potential for integration:
  - Development toolchain
  - Reporting, BOM

  - Matched content may require expert knowledge to determine relevance

Source Code Scanner: Main Usage



4. Dev Ops Integration

OPENCHAIN

Dev Ops Integration: Introduction

- Purpose:
  - Uses the information from building the software to determine OSS used

- Other identifications:
  - Can be combined with source code scanning, license scanning, binary scanning

- Also of interest:
  - Resulting identification of elements during building the software enables the creation of a bill of material (BOM)

Dev Ops Integration: Solved Problem

OPENCHAIN

Problem: for larger software projects a tool based approach is inevitable to understand involved OSS

- Modern software building environments have defined dependencies
- During compilation, dependencies can be captured to understand used dependencies
- License compliance integrated into the Dev Ops tooling implements automation
- Reporting as part of Dev Ops tooling reduces manual efforts
- Enables short release cycles in an agile environment

Dev Ops Integration: Technical

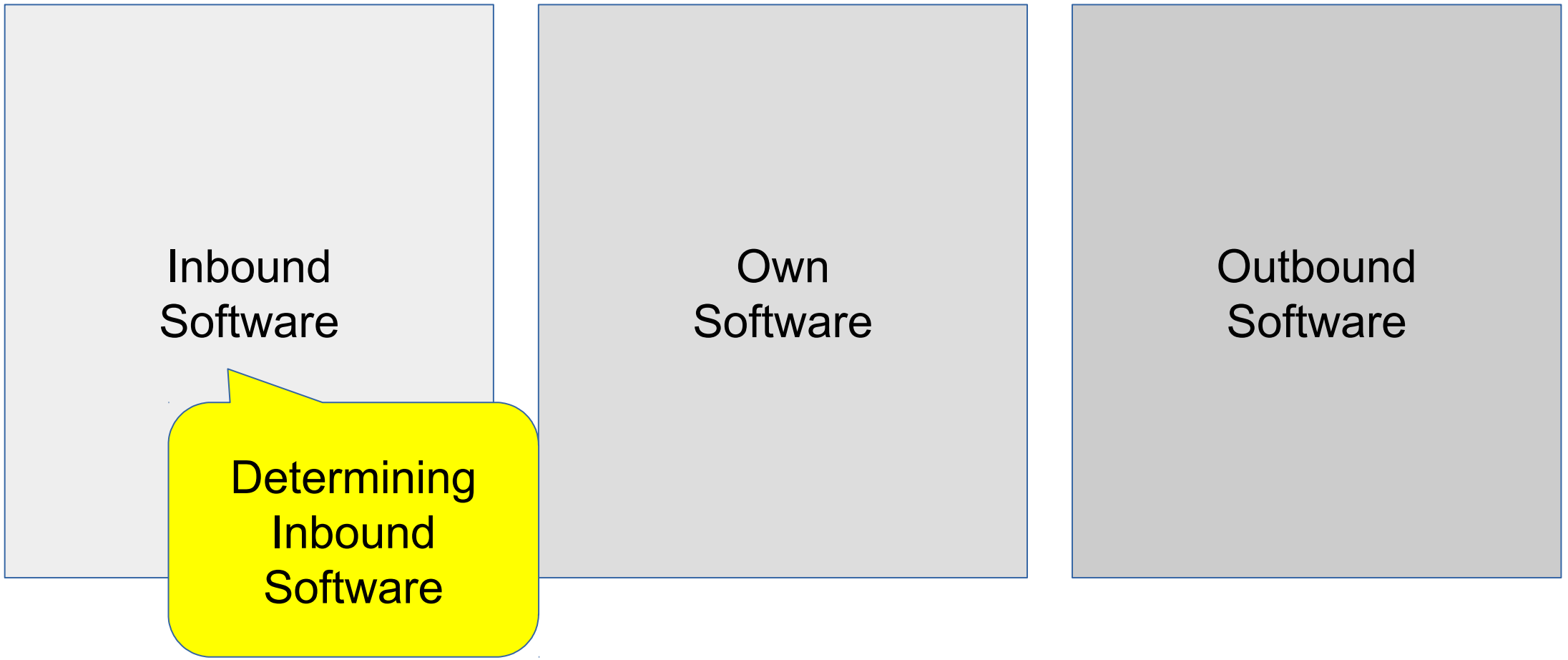
OPENCHAIN

- Integration into Dev Ops tooling requires customization
  - Building software depends on used technology as well as individually setup tooling
  - Additional efforts, if software is comprised of different technologies
  - Today, building environments sometimes contain already metadata about licensing of involved OSS software
  - Identified software elements may require additional checks to determine actual licensing information
    - (in case of heterogeneous licensing)

Dev Ops Integration: More Remarks

- Today, a custom task, nothing to "download and double-click"
- Tooling approach allows for differential approach: once setup and checked, only new dependencies require additional coverage

Dev Ops Integration: Main Usage



5. Component Catalogue

OPENCHAIN

Component Catalogue: Introduction

- Purpose:
  - Collect information about used software components and their use in products or projects is centrally collected and can be reused

- Other purposes:
  - A component catalogue captures also the used components in a product or project, maintains a so-named BOM

- Also interesting:
  - Enables also vulnerability management or reuse of export classifications

Component Catalogue: Solved Problem

Problem: Once analysed component w.r.t. license compliance shall not require repeated analyses, but reuse of information shall be possible

- Component catalogue:
  - Maps component usage in products or projects
  - Makes sense if an organisation has actually multiple products
  - Shows organisation the important software components
  - Allows for a comprehensive overview about involved licensing per product

Component Catalogue: Technical

- A component catalogue can be viewed as a portal
- Database holding the catalogue information
- Another use case is archiving OSS distributions / source code
- Storing also multiple other files,
  for example license analysis reports, SPDX files
- Provides reporting output, for example OSS product documentation
- Component catalogue can be implemented as Web portal, thus accessible from various client computers in organisation

Component Catalogue: More Remarks

- Component catalogue can be integrated with other license compliance tooling: scanners can directly feed the analyses
- Also integration in Dev Ops tooling is useful to automatically create BOM of products
- Component catalogues can also serve uses cases for vulnerability management
- Another related topic is license management and license metadata

Component Catalogue: Main Usage