OPENCHAIN

Reference Automation Slides

Open Source Training for OpenChain 2.1 (ISO/IEC 5230:2020)

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 are the Reference Automation Slides?

OPENCHAIN

- Explanation…

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

Contents

1. Automation Use Cases
2. Automation Types

CHAPTER XX

Automation 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

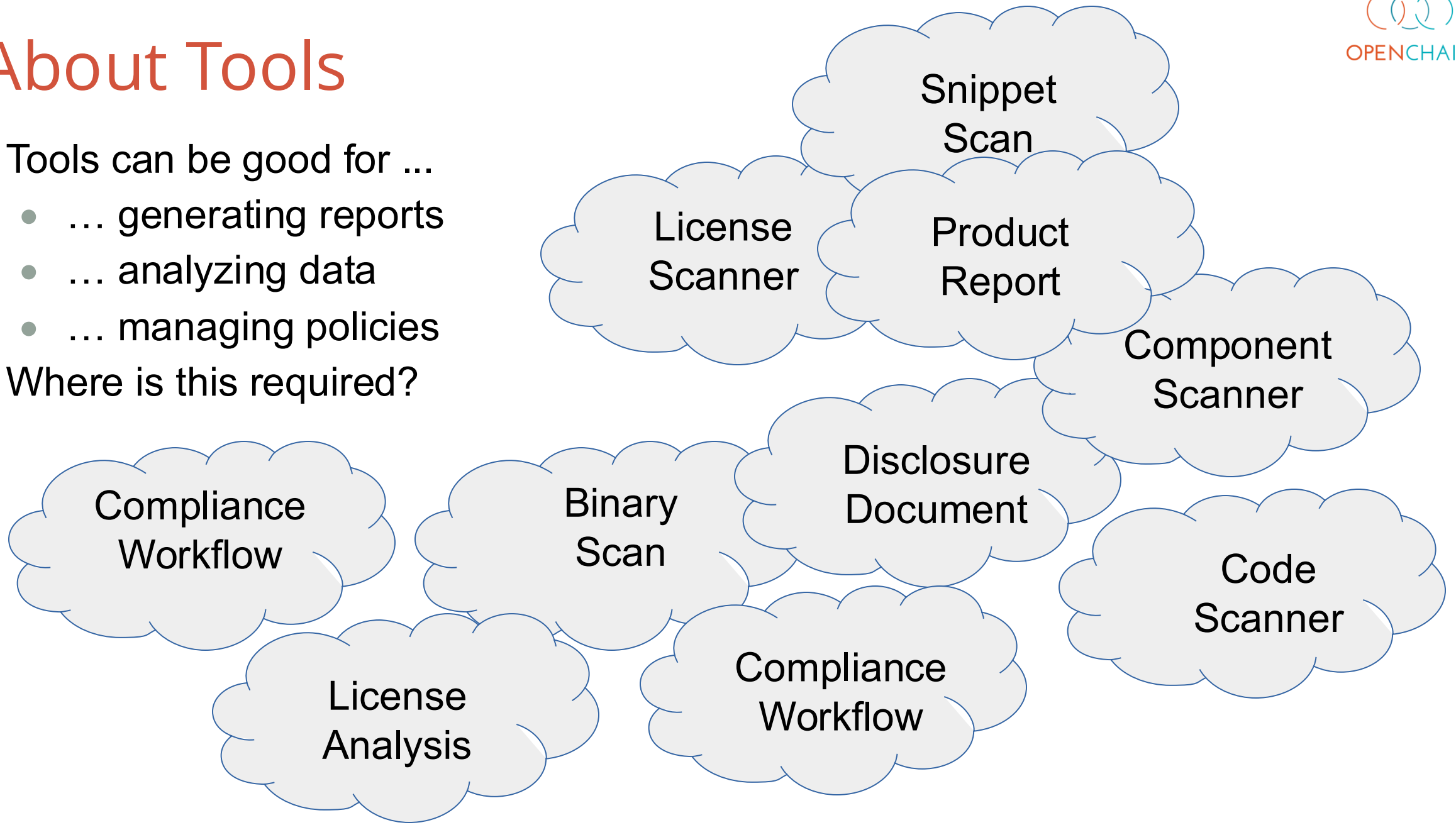
“A fool with a tool is still a fool” (from the hardware world)

About Tools

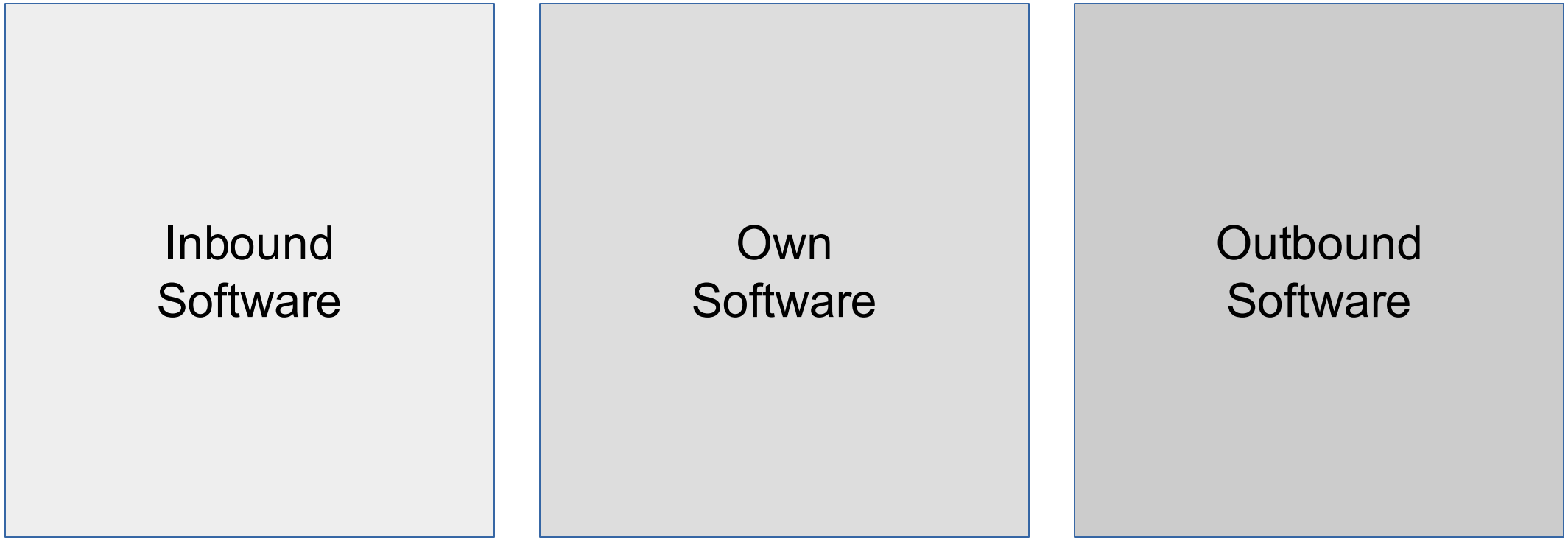
Tools can be good for ...

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

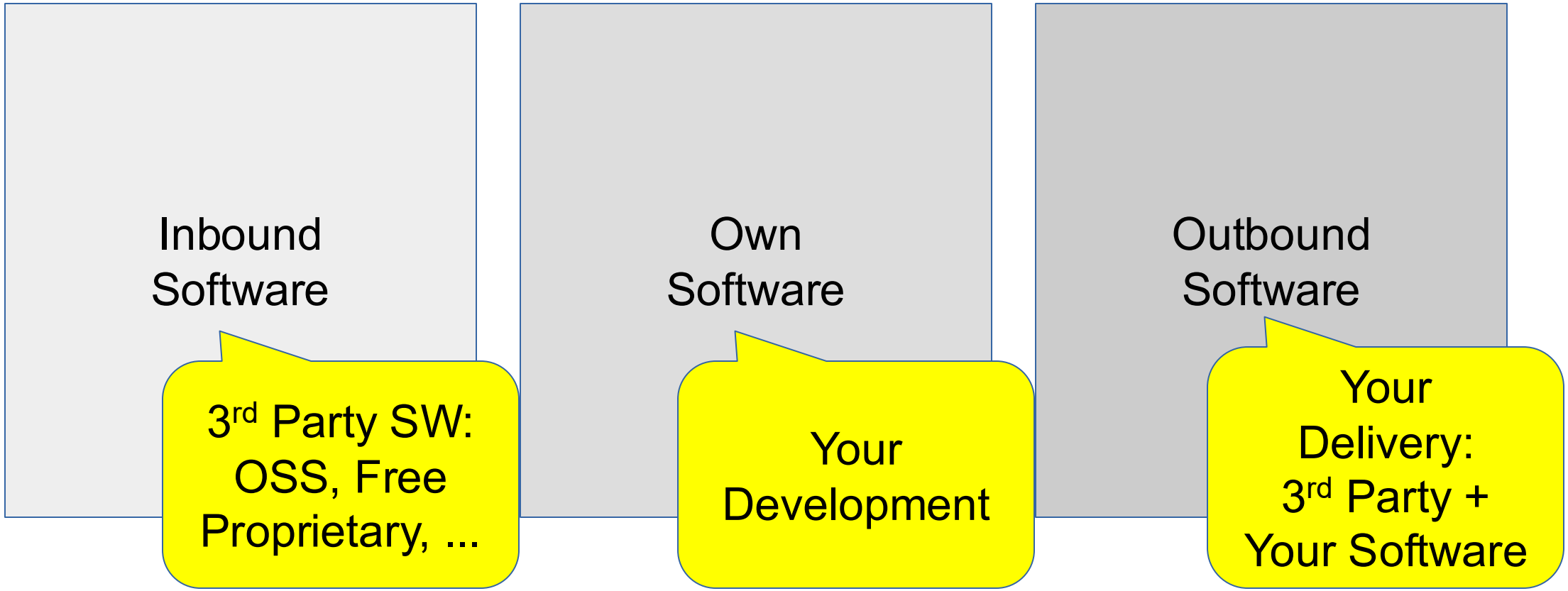
Where is this required?



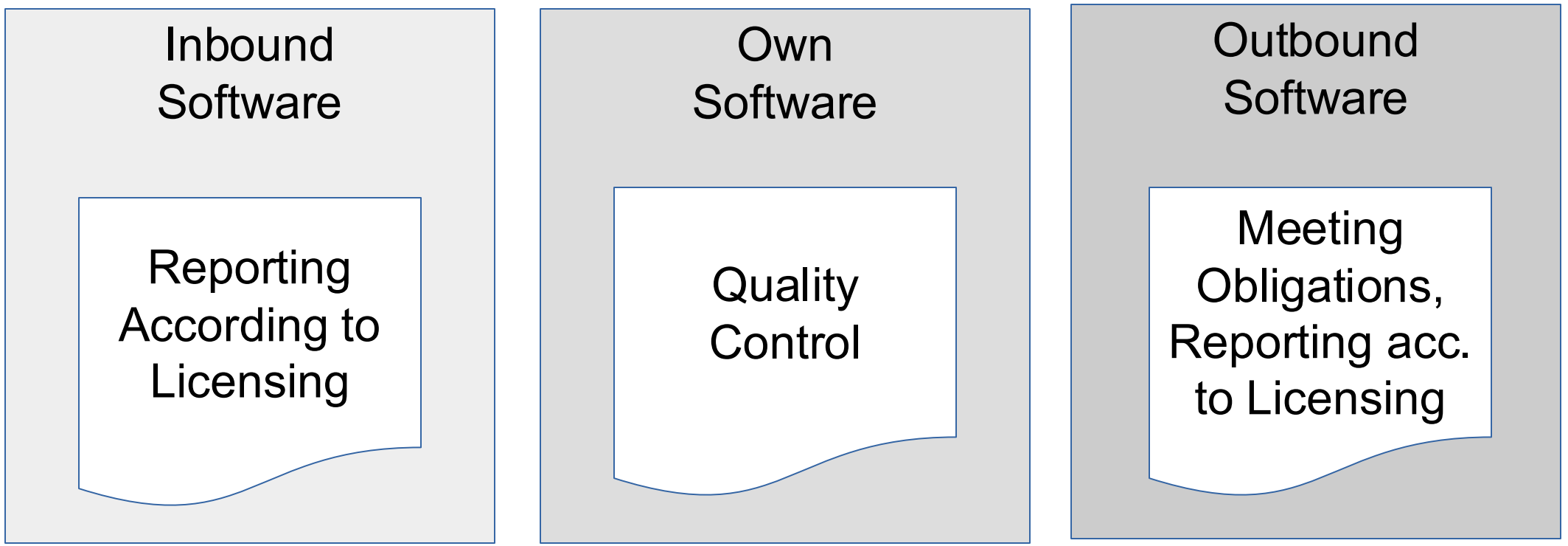
Software Situation



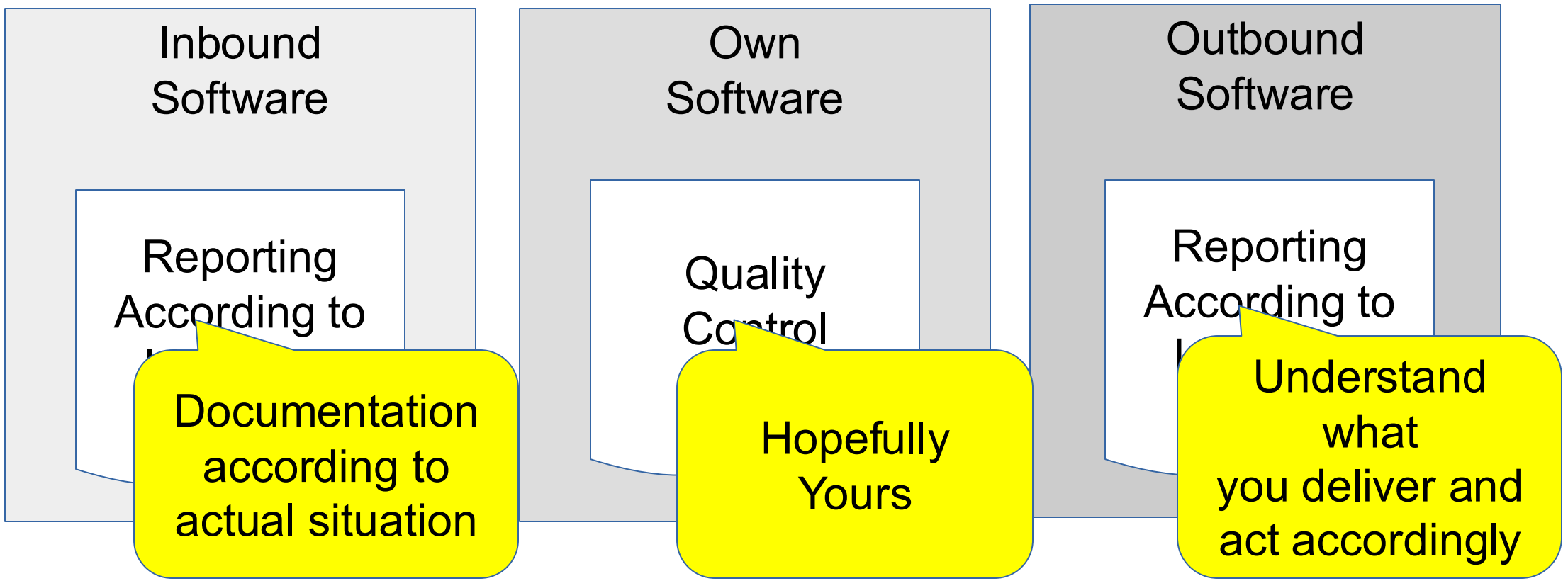
Software Situation – What it Means



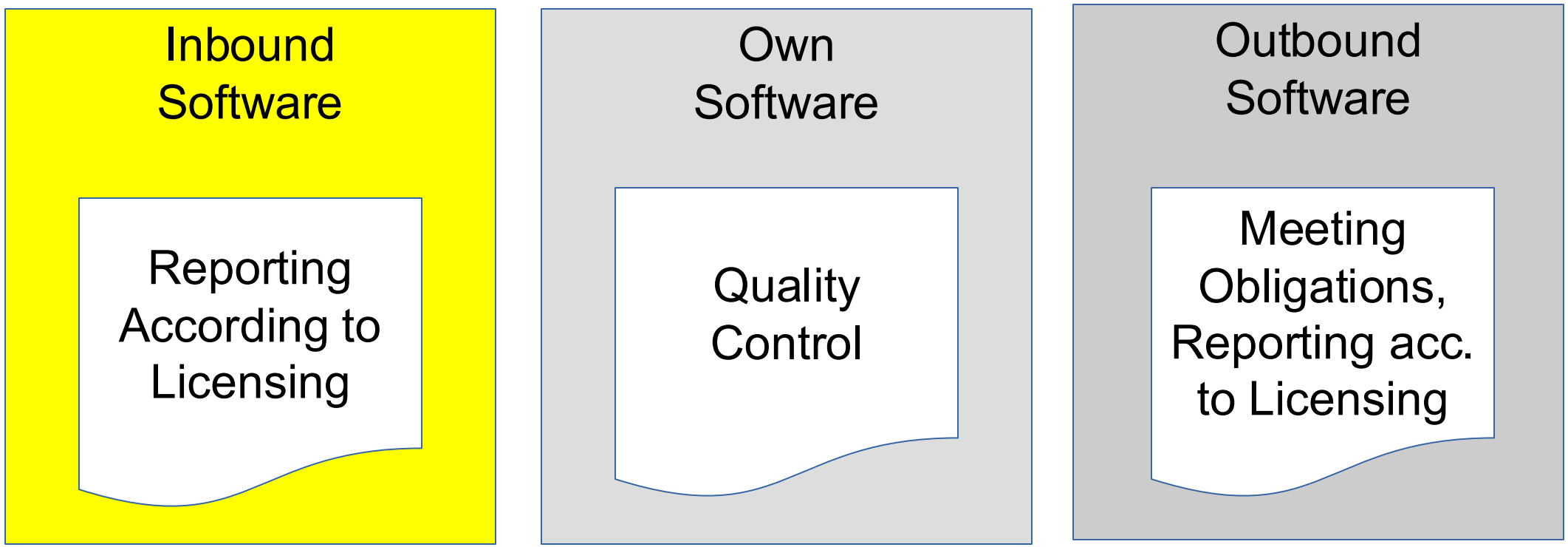
OSS License Compliance from 10k Feet



Again What this Means



Part I: Analysing Inbound



Understanding Inbound

OPENCHAIN

- 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)

OPENCHAIN

… 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

OPENCHAIN

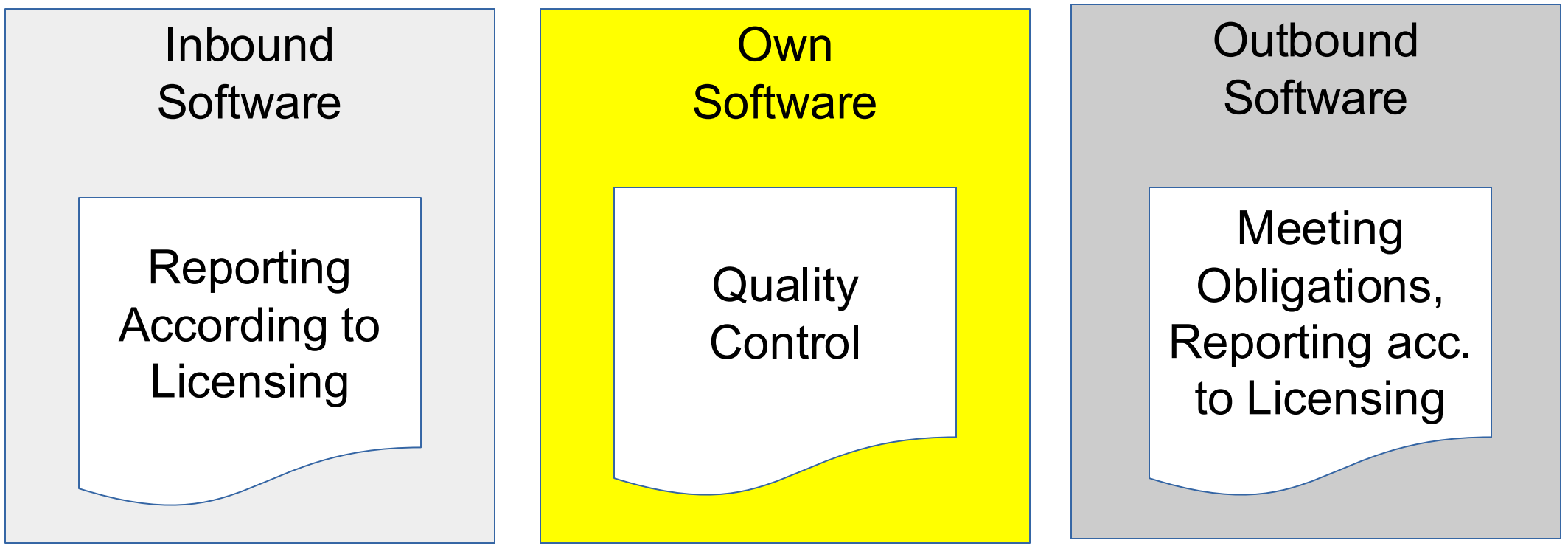
Binaries are compiled applications, libraries, software that can be used

- 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?

OPENCHAIN

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

OPENCHAIN

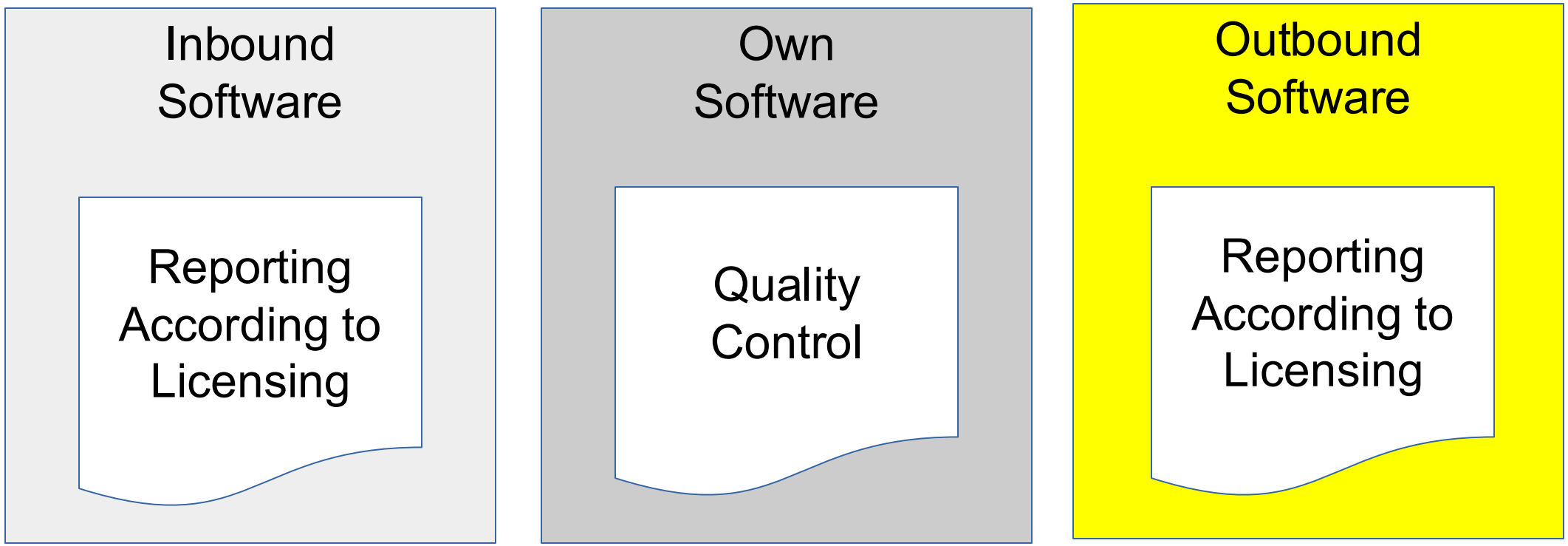
Good education and engineering codex can be solution

- 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

OPENCHAIN

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

OPENCHAIN

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)

OPENCHAIN

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)

OPENCHAIN

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)

OPENCHAIN

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 XX

Automation 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

OPENCHAIN

- 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

OPENCHAIN

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

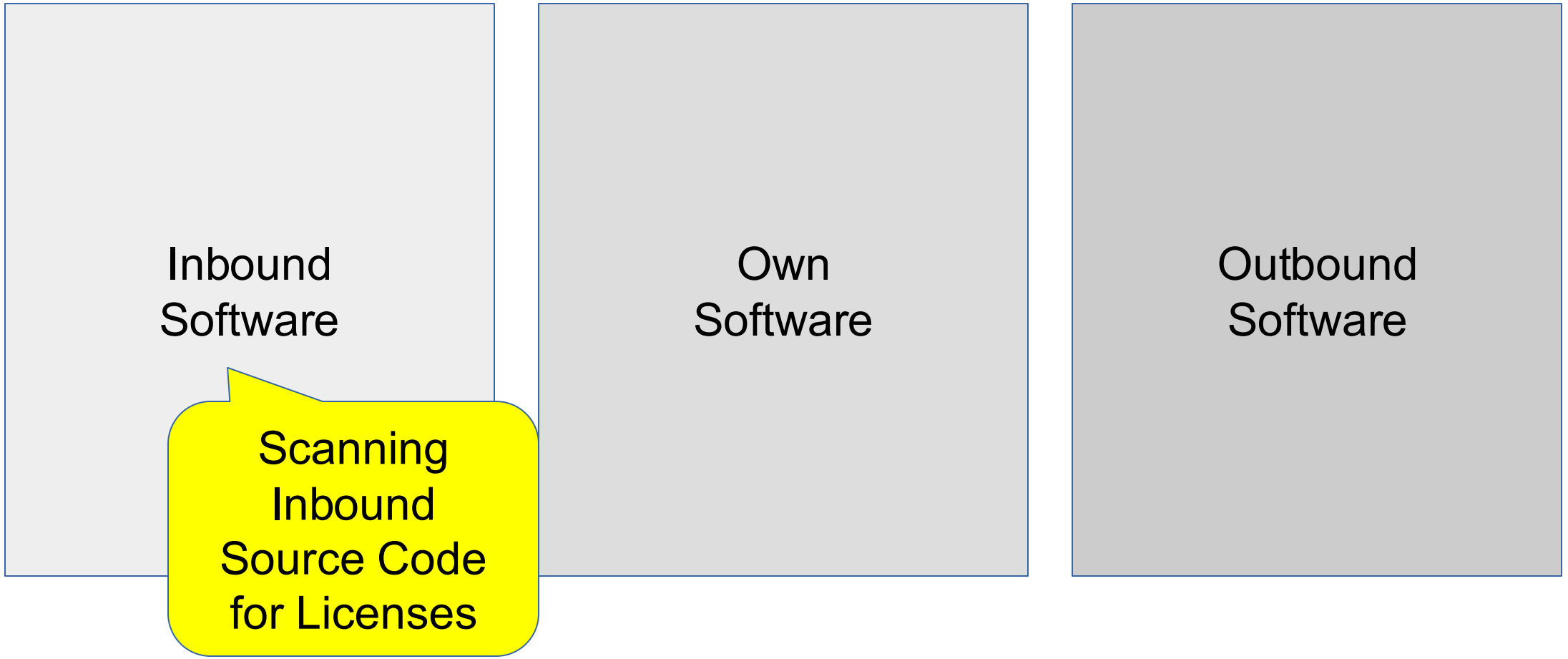
- Searching in every file of software uploaded: requires source code distribution
- Different approaches can be applied: regular expressions, text comparison, phrase collection
- Requires database of license texts, licensing statements
- Comparison with existing license texts enables exact identification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

License Scanner: More Remarks

OPENCHAIN

- 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

OPENCHAIN

- 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

OPENCHAIN

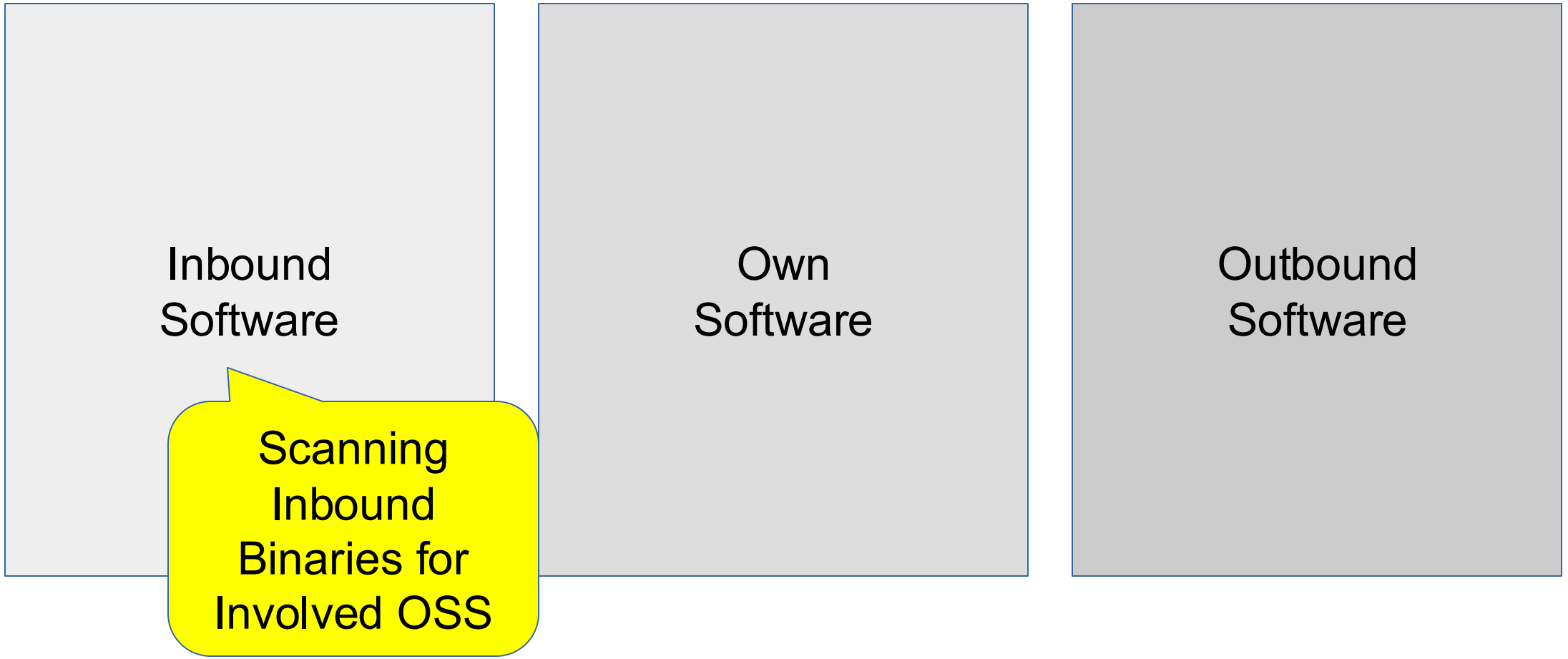
- Compiled machine language can contain characteristic elements
- For example used string variables (=text)
  or other content compiled into the binary
- Simpler method: capturing file names,
  or for run-time code (e.g. Java): method and field names
- Requires database of mapping
  from source code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

Binary Scanner: More Remarks

OPENCHAIN

- 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

OPENCHAIN

- 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)

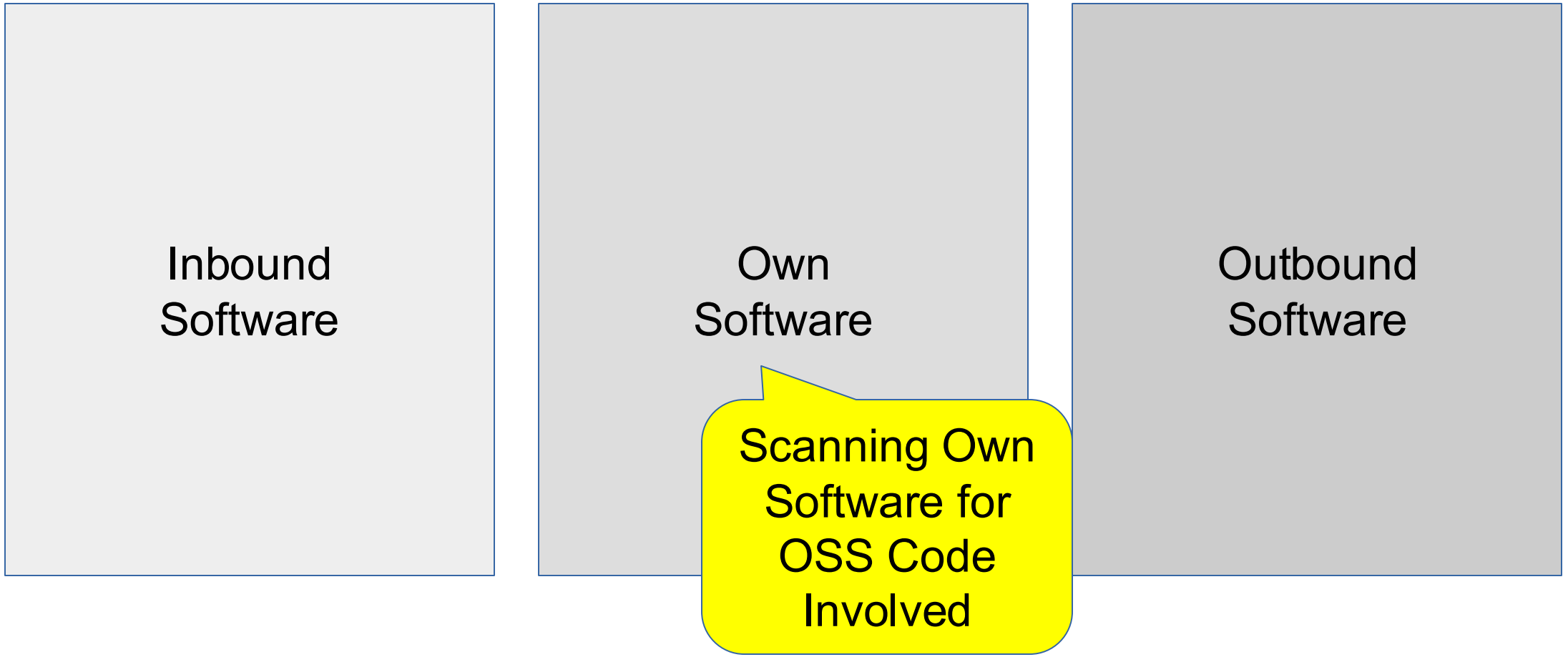
Source Code Scanner: More Remarks

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

OPENCHAIN

- 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

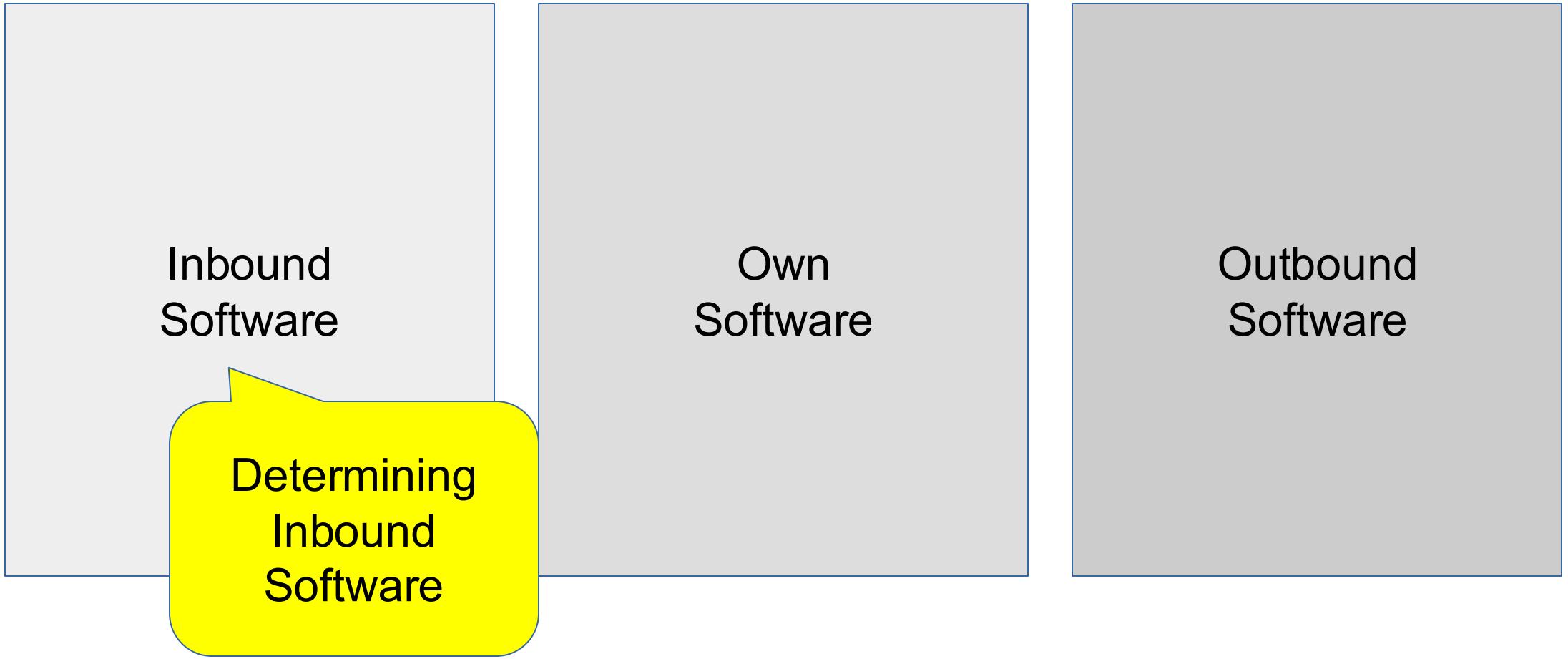
- Integration into Dev Ops tooling requires customization
  - Building software depends on used technology as well as individually setup tooling
  - Additional efforts, if software is comprised of different technologies
  - Today, building environments sometimes contain already metadata about licensing 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

OPENCHAIN

- 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

OPENCHAIN

- 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

OPENCHAIN

- 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

OPENCHAIN

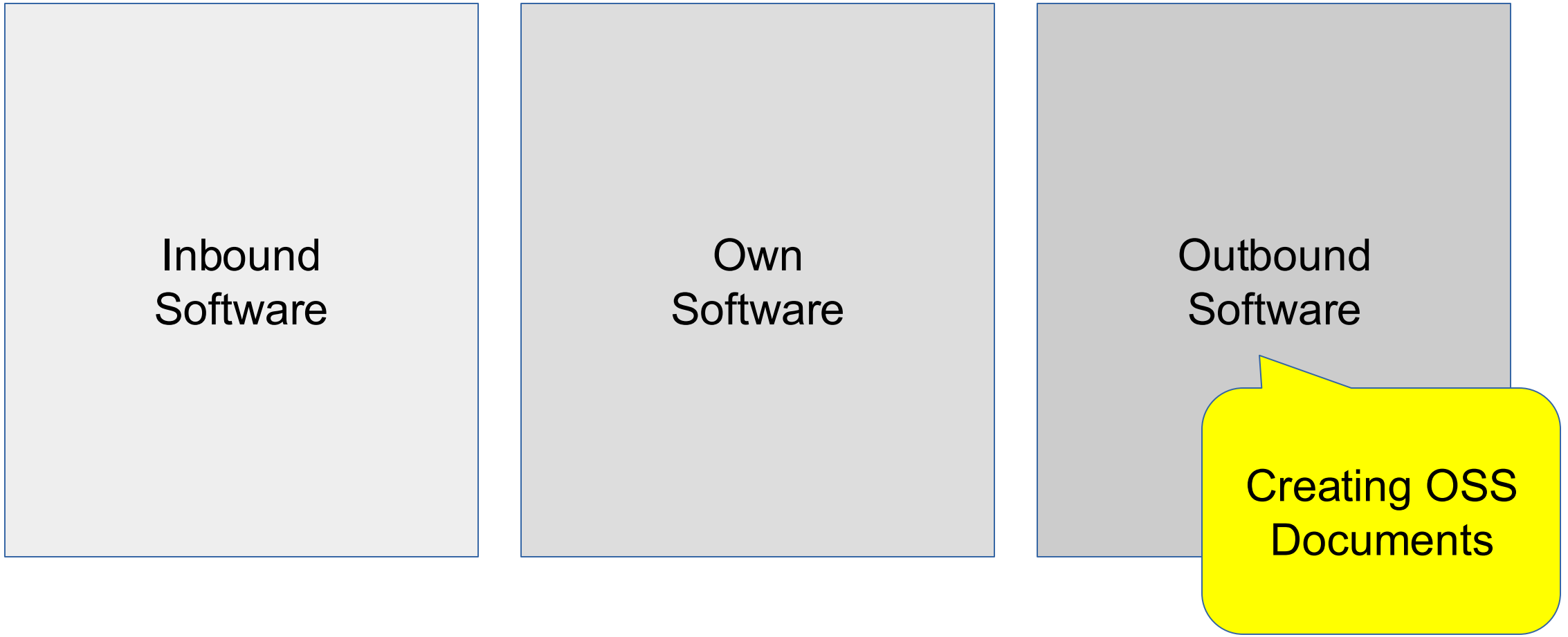
- A component catalogue can be viewed as a portal
- Database holding the catalogue information
- Another use case is archiving OSS distributions / source code
- Storing also multiple other files,
  for example license analysis reports, SPDX files
- Provides reporting output, for example OSS product documentation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

Component Catalogue: More Remarks

OPENCHAIN

- 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



OPENCHAIN

Reference Automation Slides

Open Source Training for DIS 5230 (ISO Number Pending)

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.

What are the Reference Automation Slides?

OPENCHAIN

- Explanation…

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

Contents

1. Tools Use Cases
2. Tooling Types

OPENCHAIN

CHAPTER 1

Tooling Use Cases

Introduction

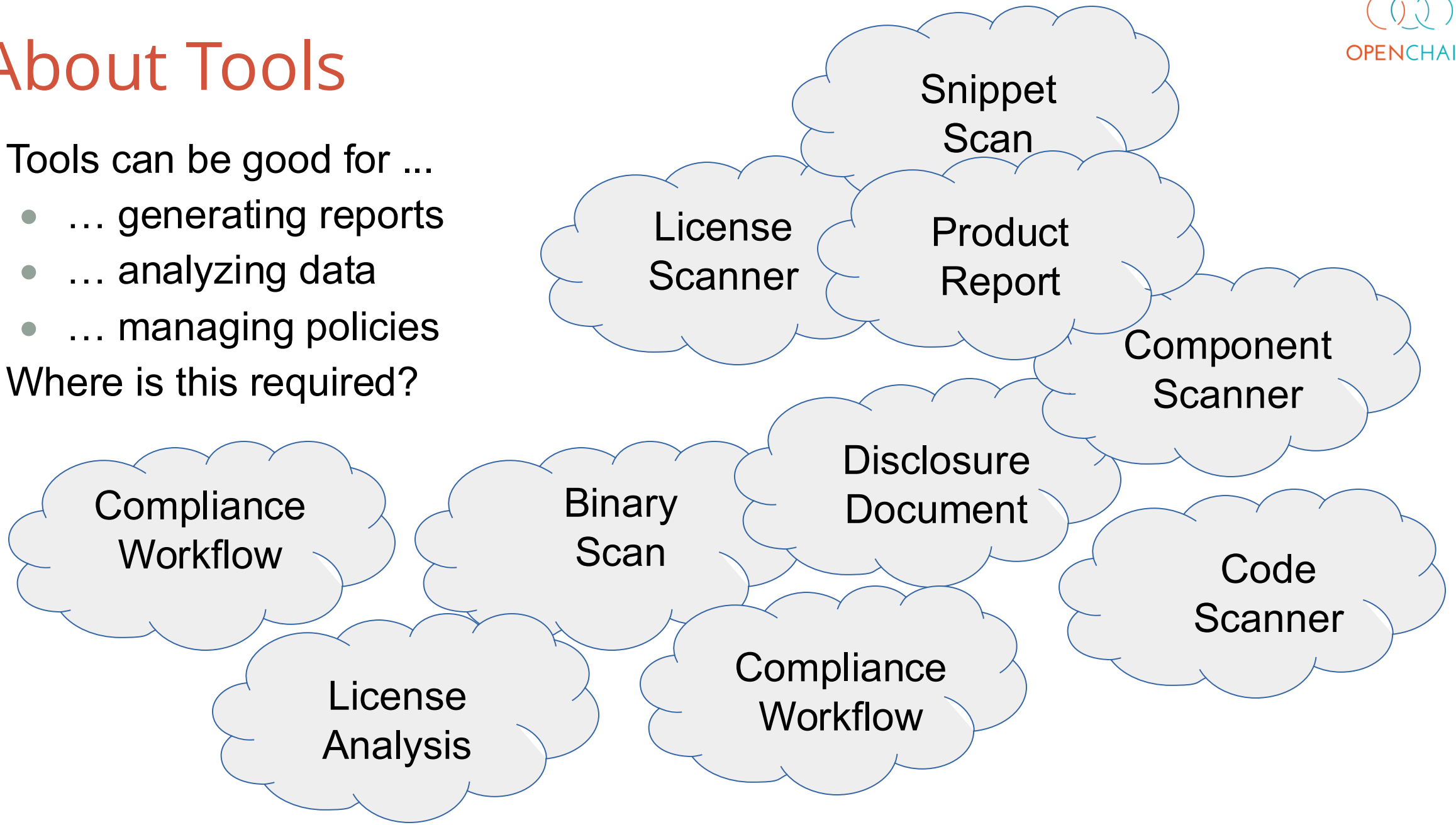
- Why we would need tools?
- First demand and process, then the tool
- A tool cannot provide (difficult) decisions
- Only data for decisions
- Many cases where expert knowledge is required

About Tools

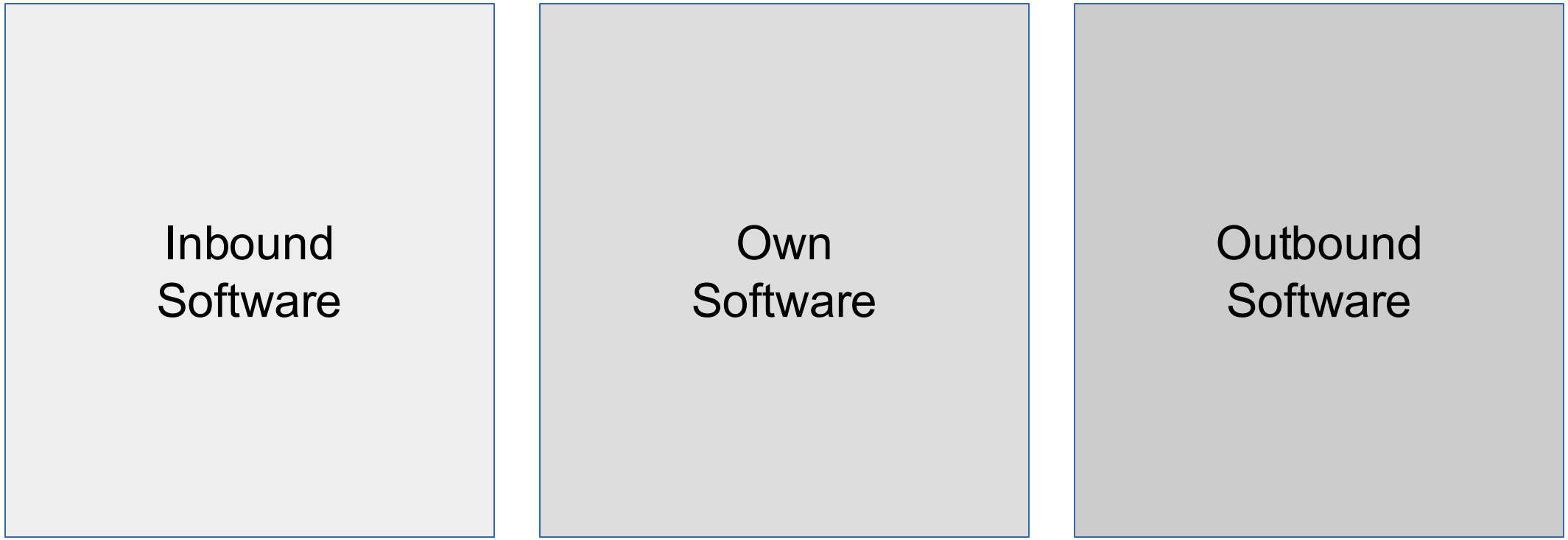
Tools can be good for ...

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

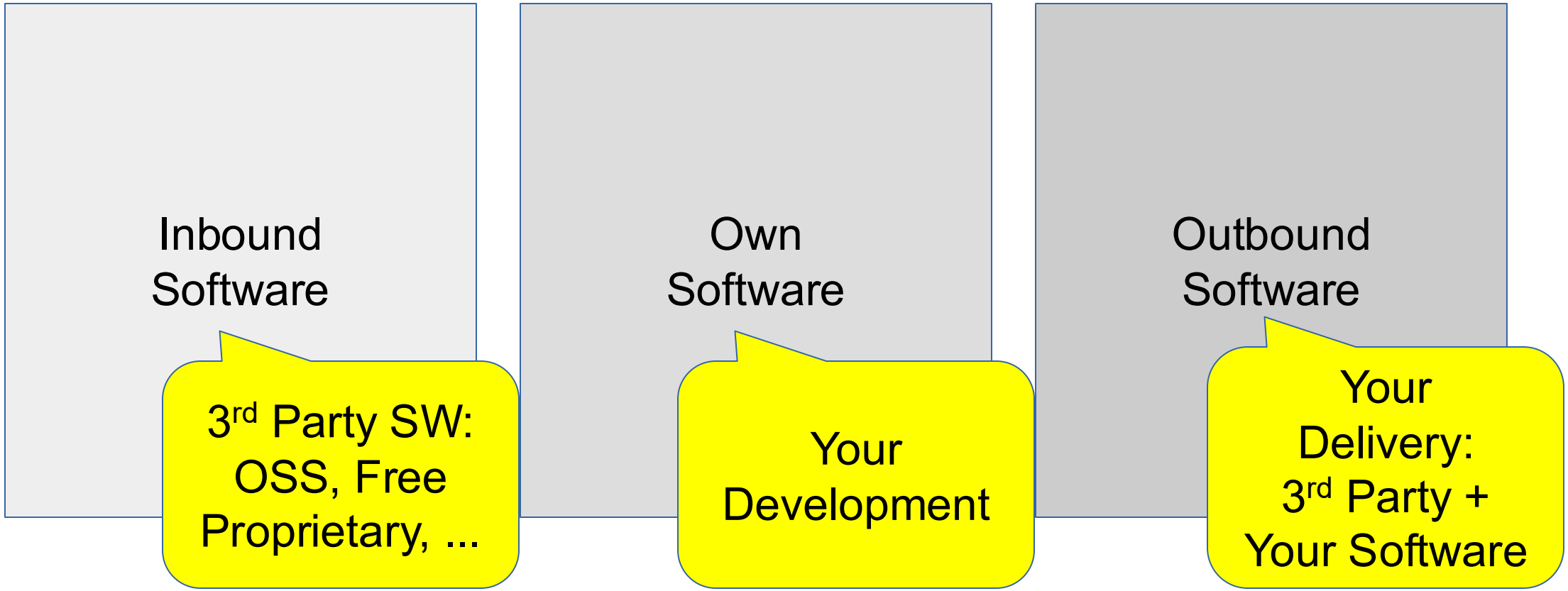
Where is this required?



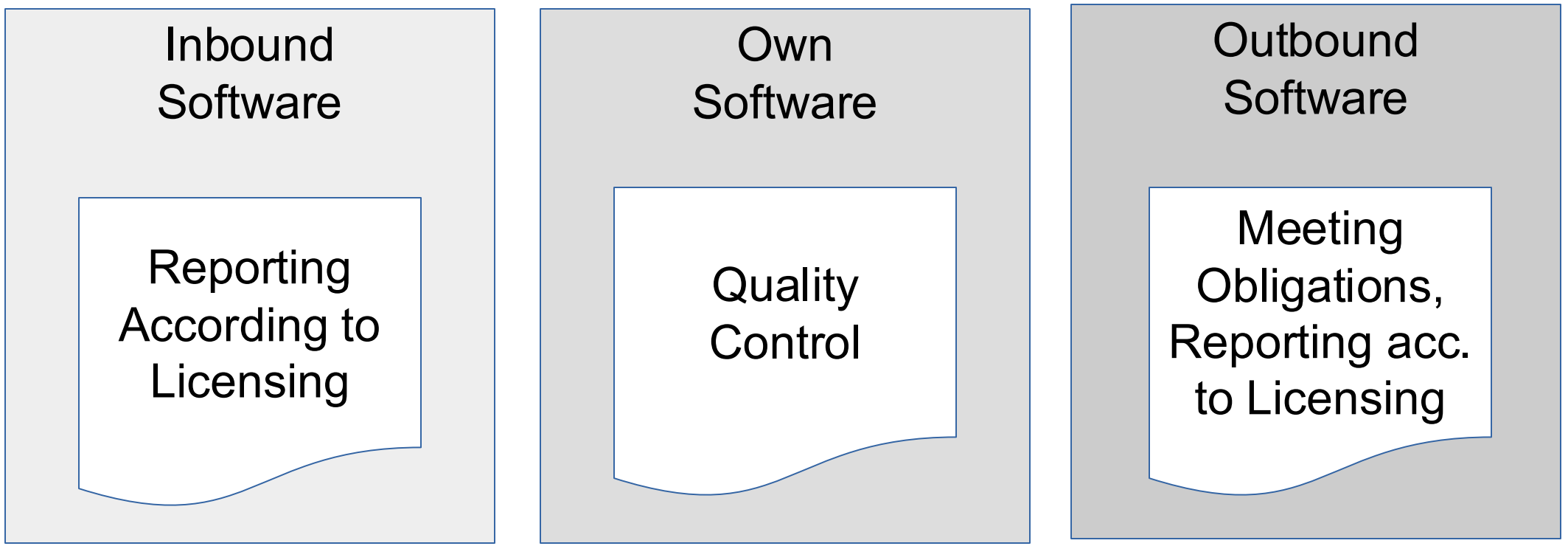
Software Situation



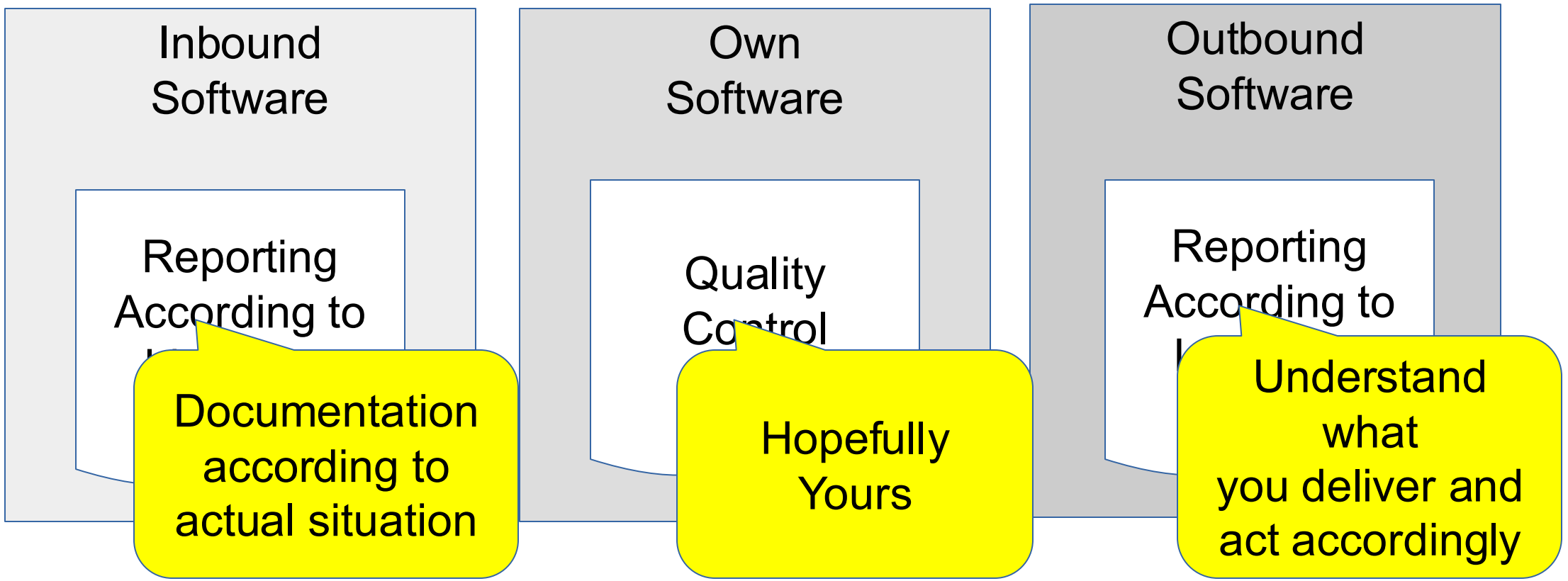
Software Situation – What it Means



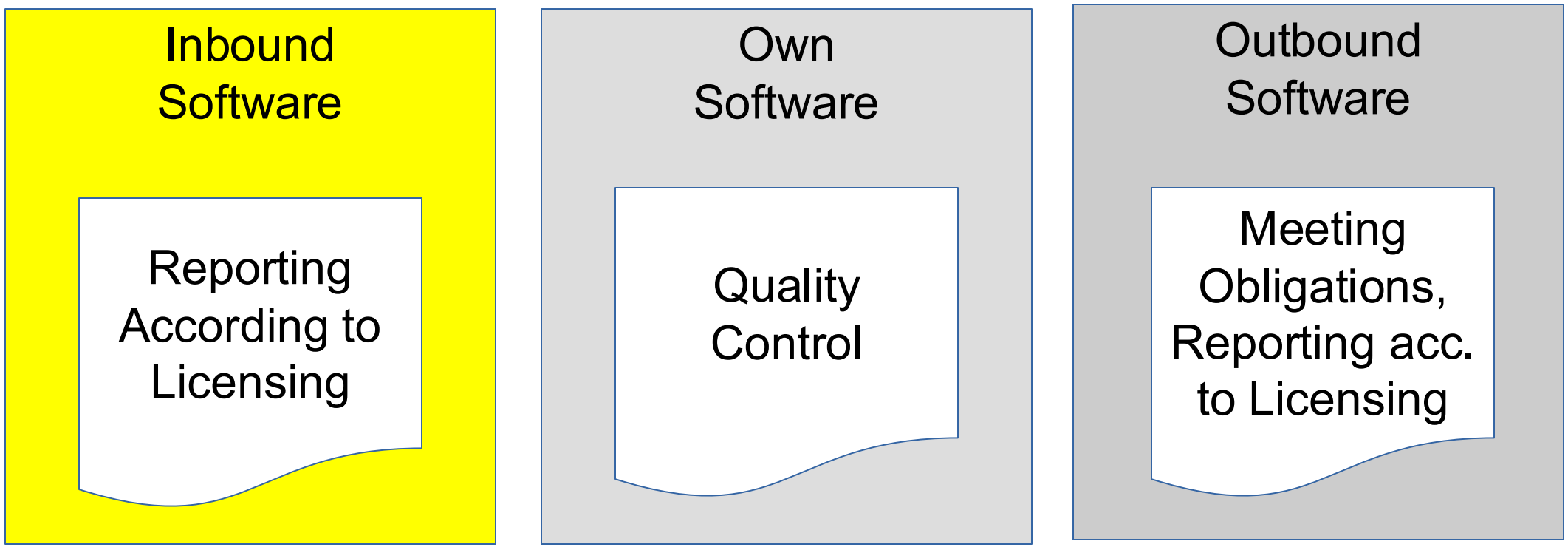
OSS License Compliance from 10k Feet



Again What this Means



Part I: Analysing Inbound



Understanding Inbound

OPENCHAIN

- 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)

OPENCHAIN

… 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

OPENCHAIN

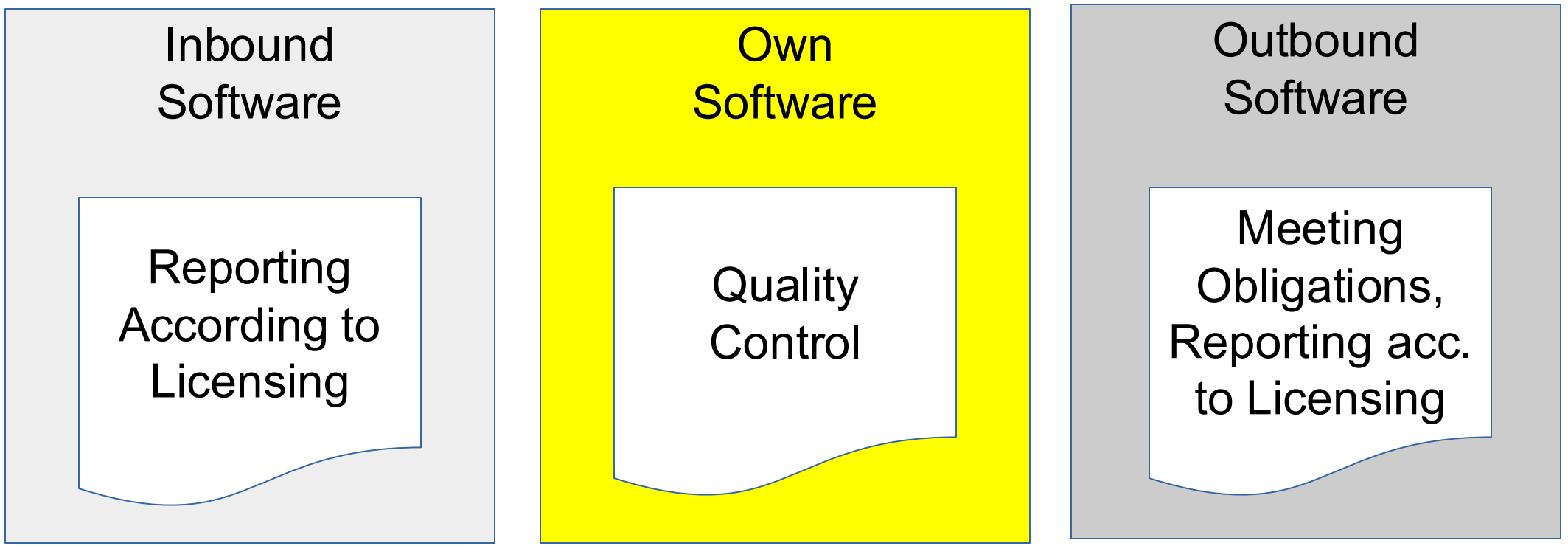
Binaries are compiled applications, libraries, software that can be used

- 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?

OPENCHAIN

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

OPENCHAIN

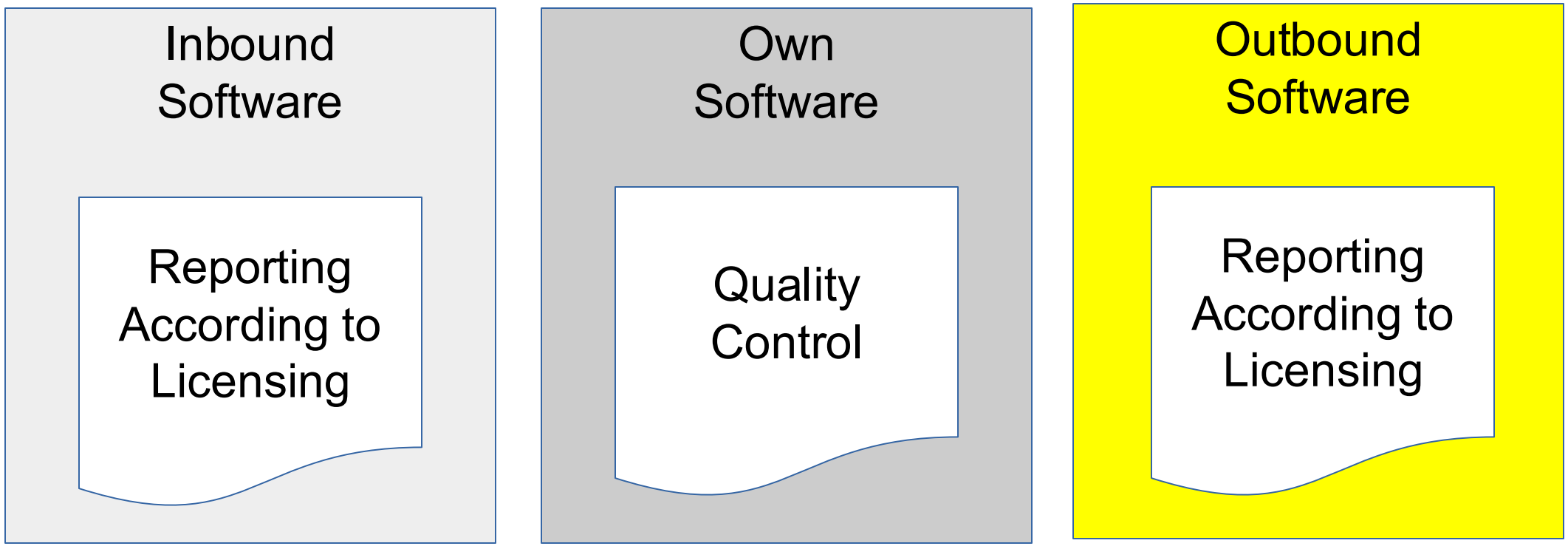
Good education and engineering codex can be solution

- 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

OPENCHAIN

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

OPENCHAIN

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)

OPENCHAIN

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)

OPENCHAIN

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)

OPENCHAIN

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

OPENCHAIN

CHAPTER 2

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

OPENCHAIN

- 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

OPENCHAIN

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

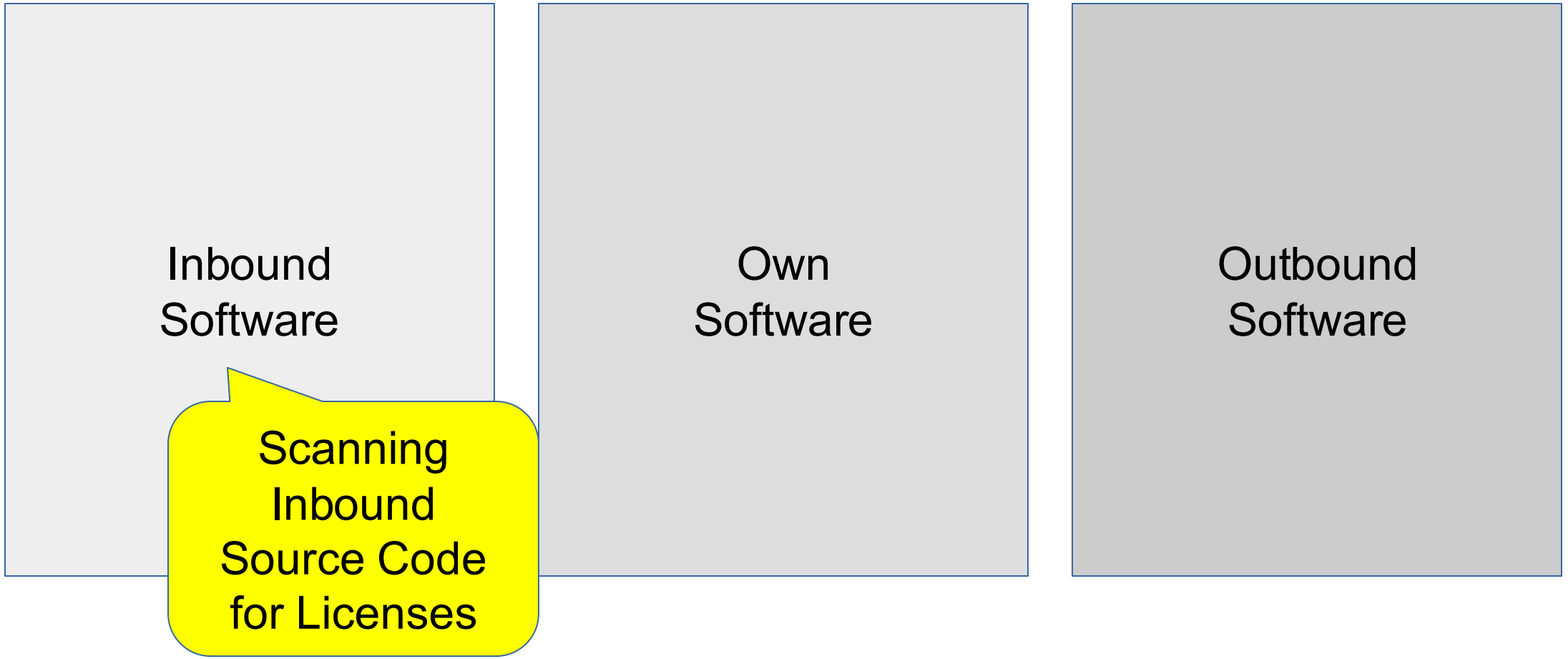
- Searching in every file of software uploaded: requires source code distribution
- Different approaches can be applied: regular expressions, text comparison, phrase collection
- Requires database of license texts, licensing statements
- Comparison with existing license texts enables exact identification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

License Scanner: More Remarks

OPENCHAIN

- 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

OPENCHAIN

- 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

OPENCHAIN

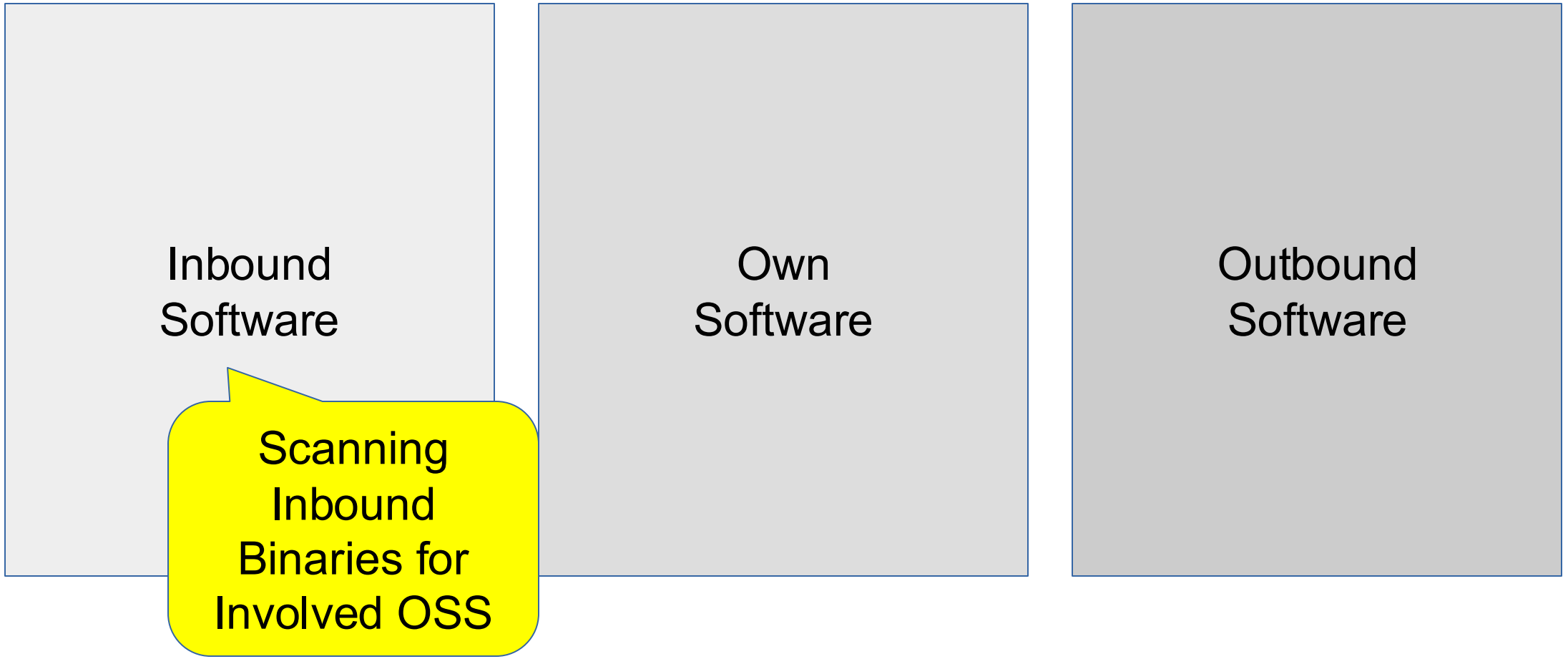
- Compiled machine language can contain characteristic elements
- For example used string variables (=text)
  or other content compiled into the binary
- Simpler method: capturing file names,
  or for run-time code (e.g. Java): method and field names
- Requires database of mapping
  from source code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

Binary Scanner: More Remarks

OPENCHAIN

- 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

OPENCHAIN

- 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)

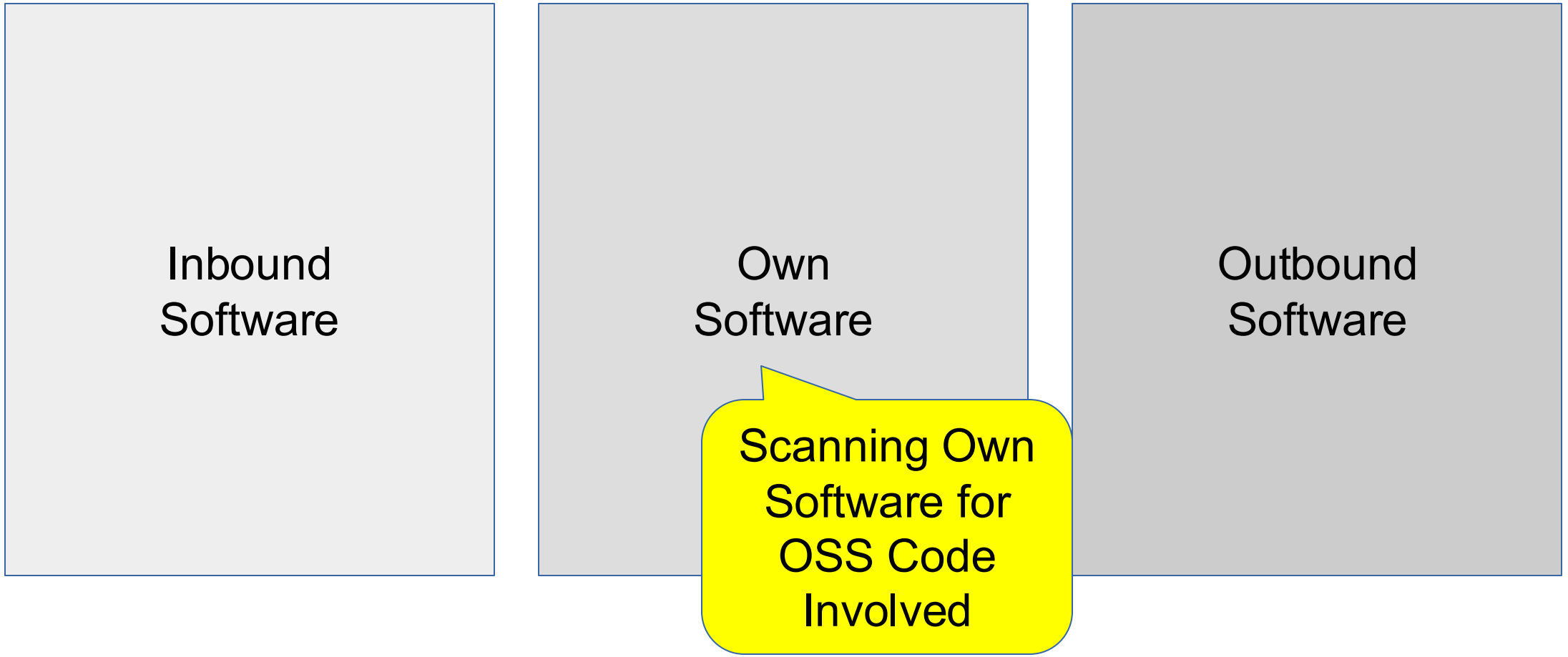
Source Code Scanner: More Remarks

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

OPENCHAIN

- 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

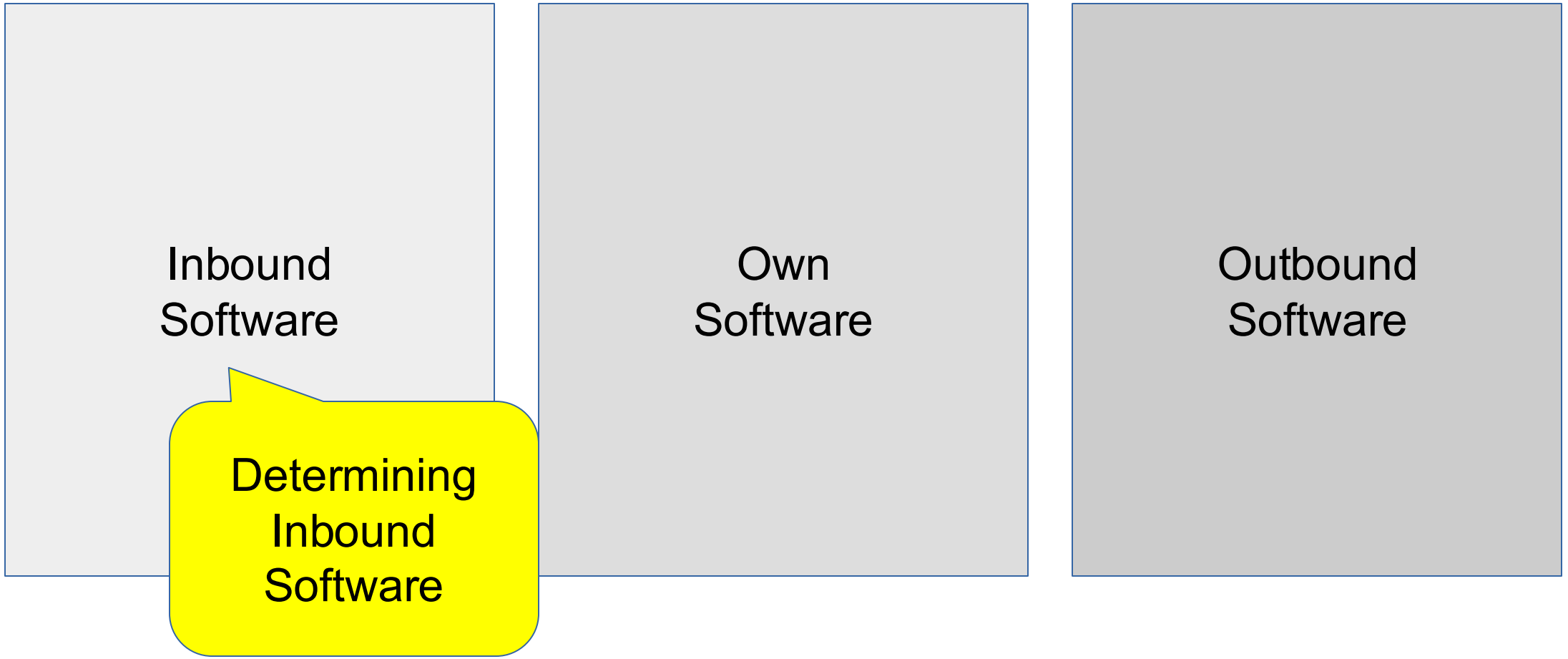
- Integration into Dev Ops tooling requires customization
  - Building software depends on used technology as well as individually setup tooling
  - Additional efforts, if software is comprised of different technologies
  - Today, building environments sometimes contain already metadata about licensing 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

OPENCHAIN

- 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

OPENCHAIN

- 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

OPENCHAIN

- 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

OPENCHAIN

- 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

OPENCHAIN

- 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