# Identification and Analysis of unsafe.Pointer Usage Patterns in Open-Source Go Code

Johannes Lauinger

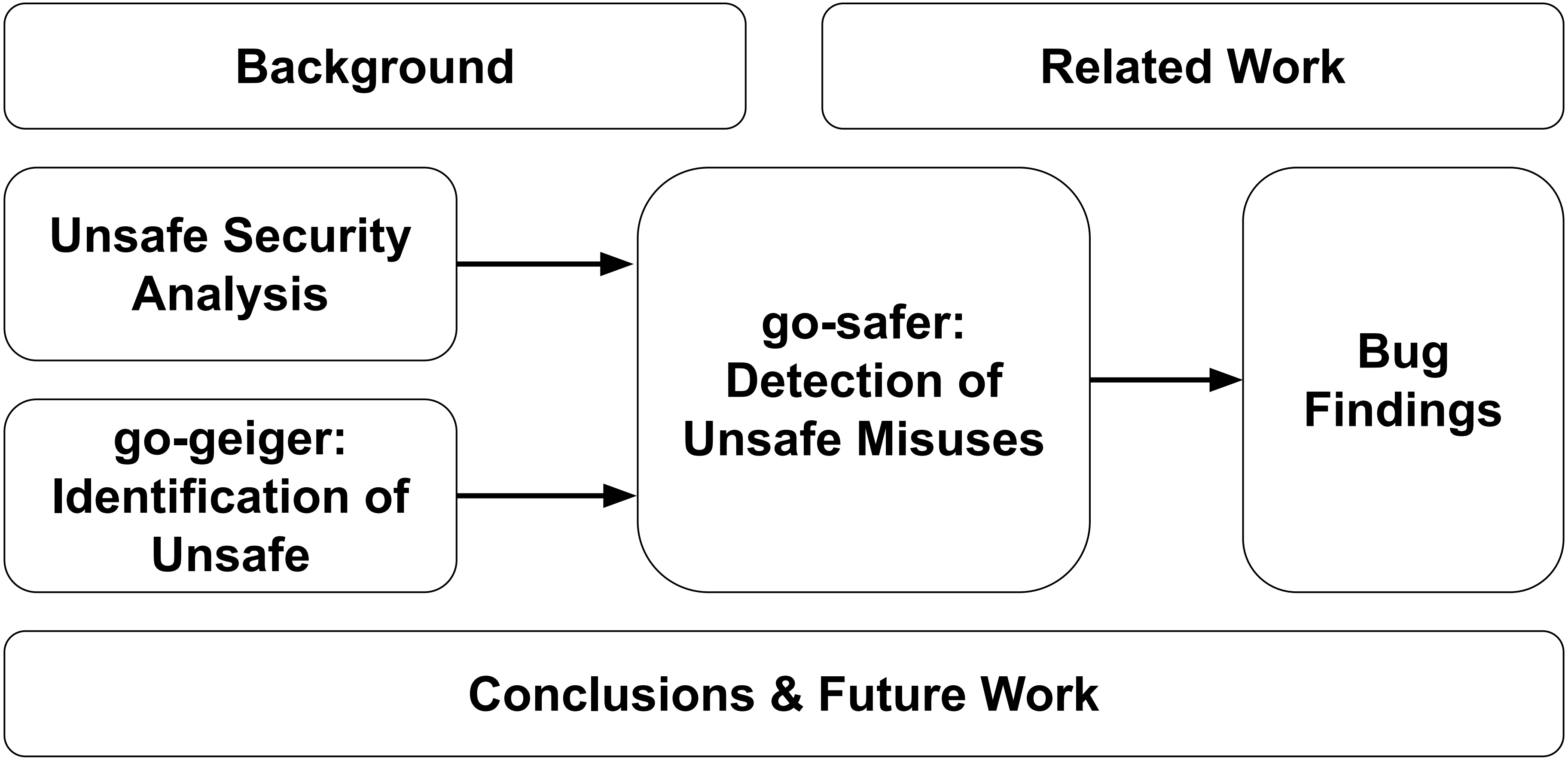
TECHNISCHE UNIVERSITÄT DARMSTADT

SOFTWARE TECHNOLOGY GROUP

# Motivation

- Memory safety: big source of bugs [8]
- Go language countermeasures: automatic memory management, type system restrictions
- Escape hatch: *unsafe* API
- How dangerous is it? How often is it used in the real world?

# Outline


Background

Related Work

Unsafe Security Analysis

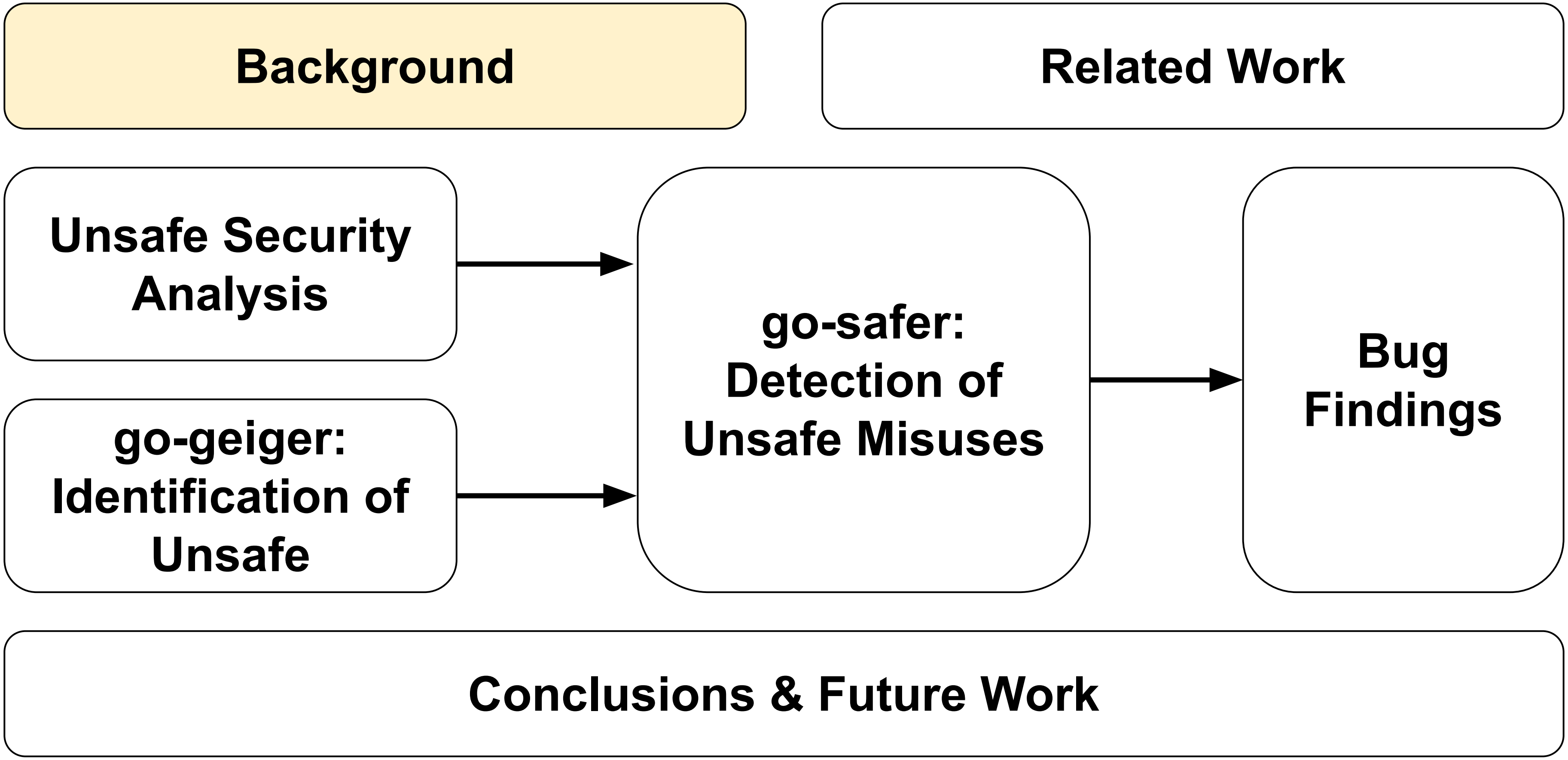
go-geiger: Identification of Unsafe

go-safer: Detection of Unsafe Misuses

Bug Findings

Conclusions & Future Work

# Outline



Background

Related Work

Unsafe Security Analysis

go-geiger: Identification of Unsafe

go-safer: Detection of Unsafe Misuses

Bug Findings

Conclusions & Future Work

# Go Unsafe API

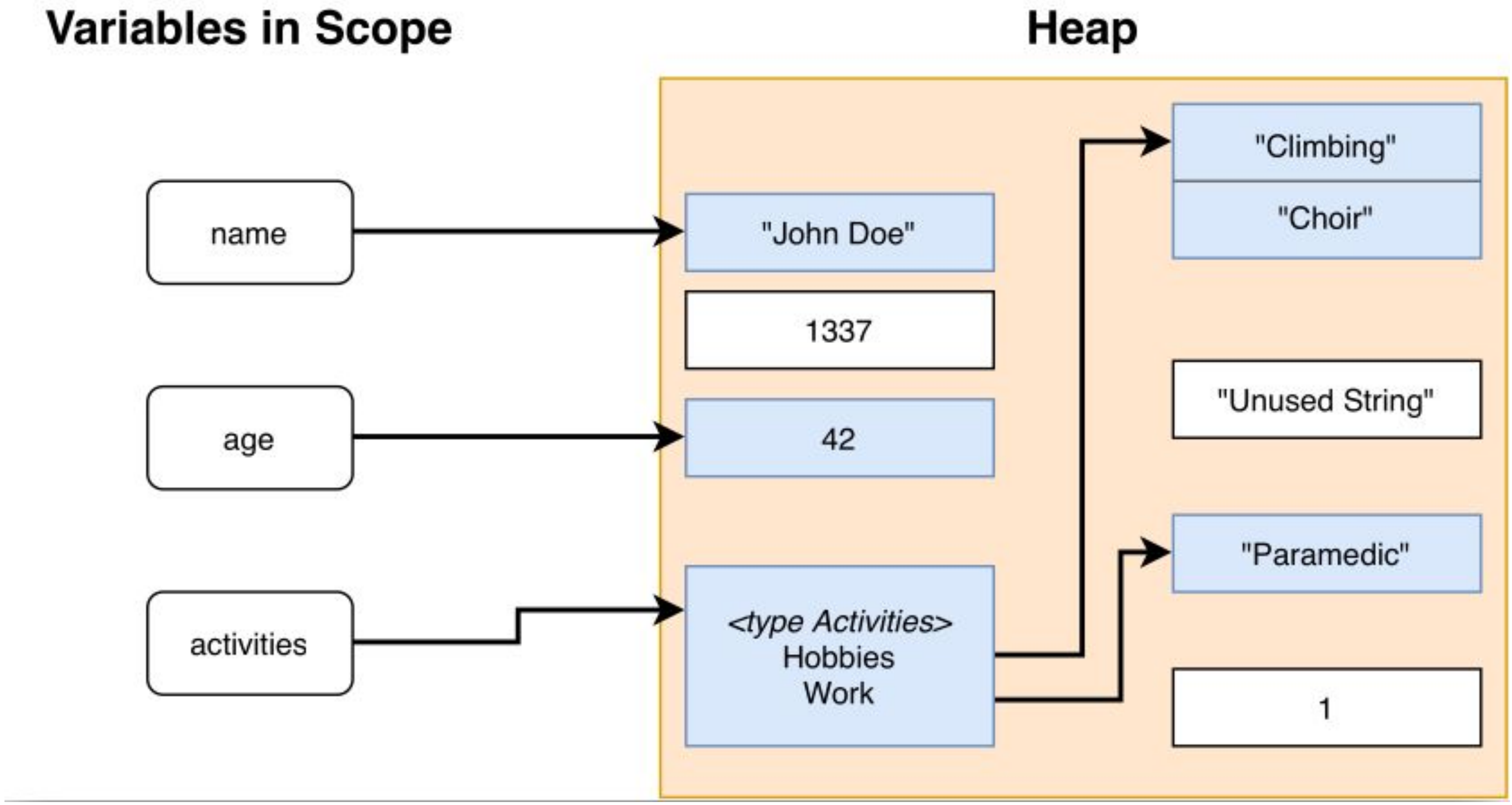
```
package unsafe

type Pointer
```

Allows:

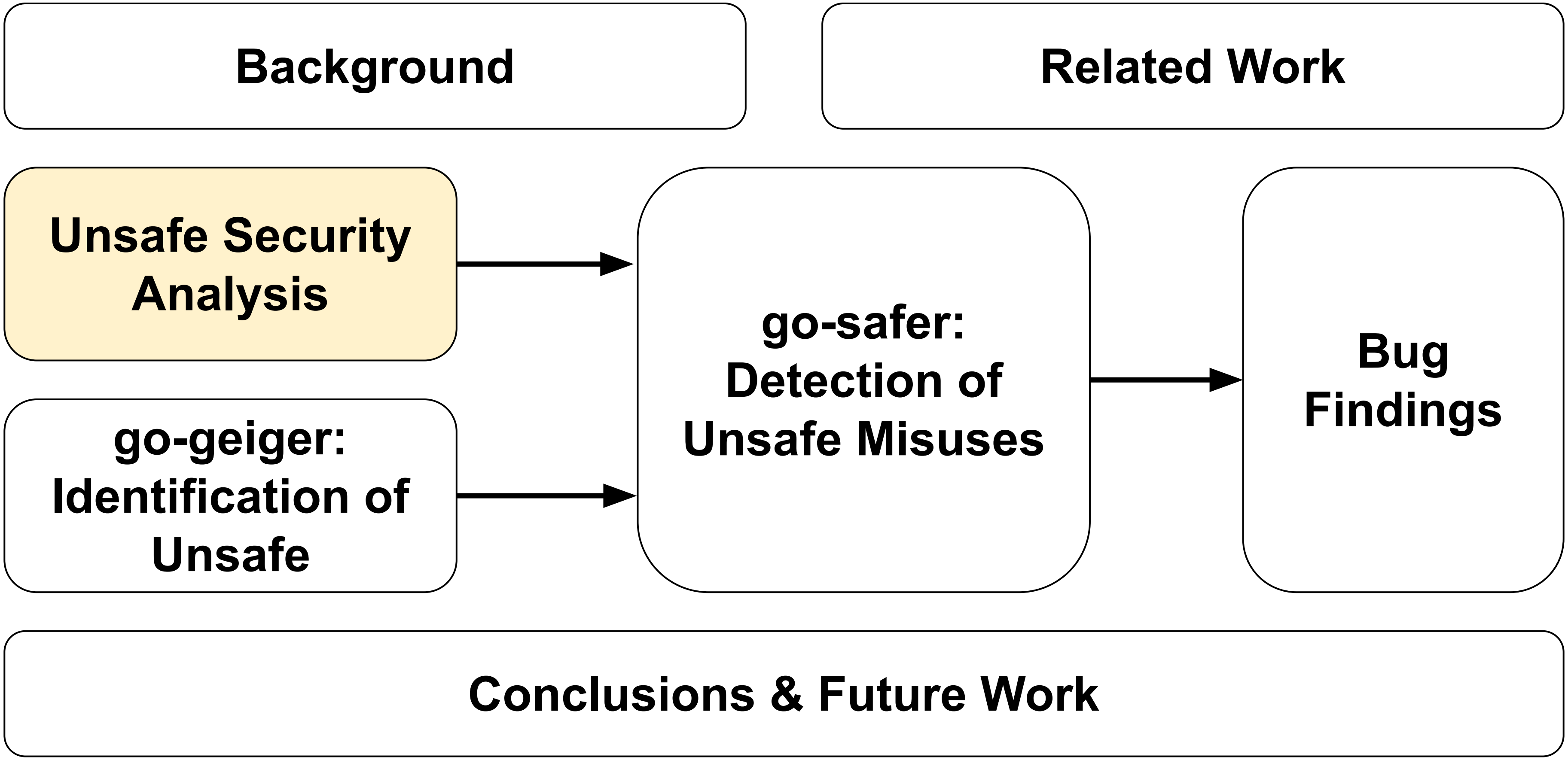
- Arbitrary type casts
- Pointer arithmetic

# Memory Management: GC

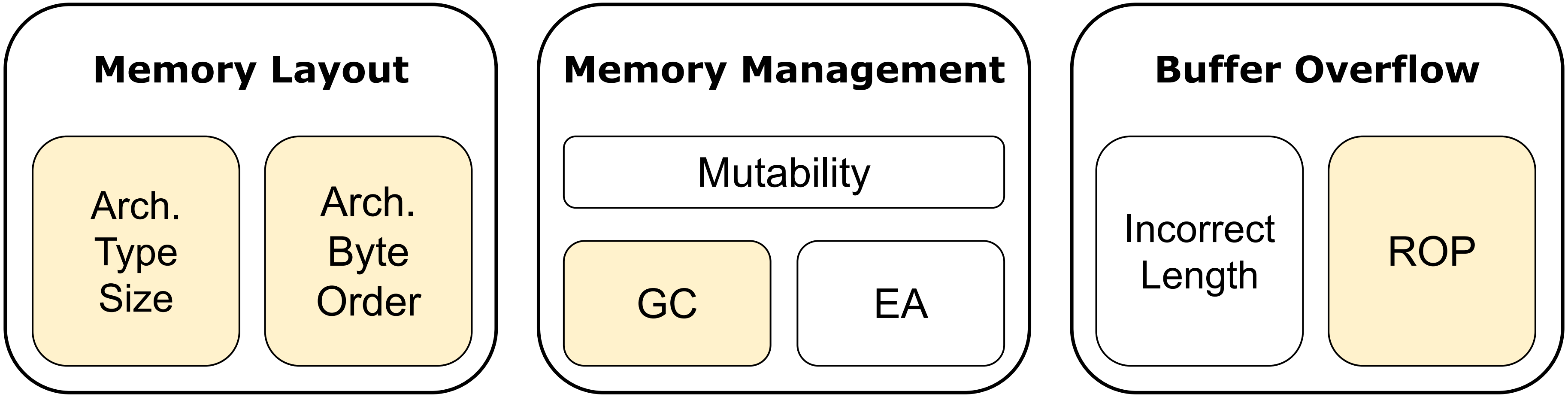
## Garbage Collection



# Outline


Background
Related Work
Unsafe Security Analysis
go-geiger: Identification of Unsafe
go-safer: Detection of Unsafe Misuses
Bug Findings
Conclusions & Future Work

# Security Analysis of Unsafe



Memory Layout

Arch. Type Size

Arch. Byte Order

Memory Management

Mutability

GC

EA

Buffer Overflow

Incorrect Length

ROP

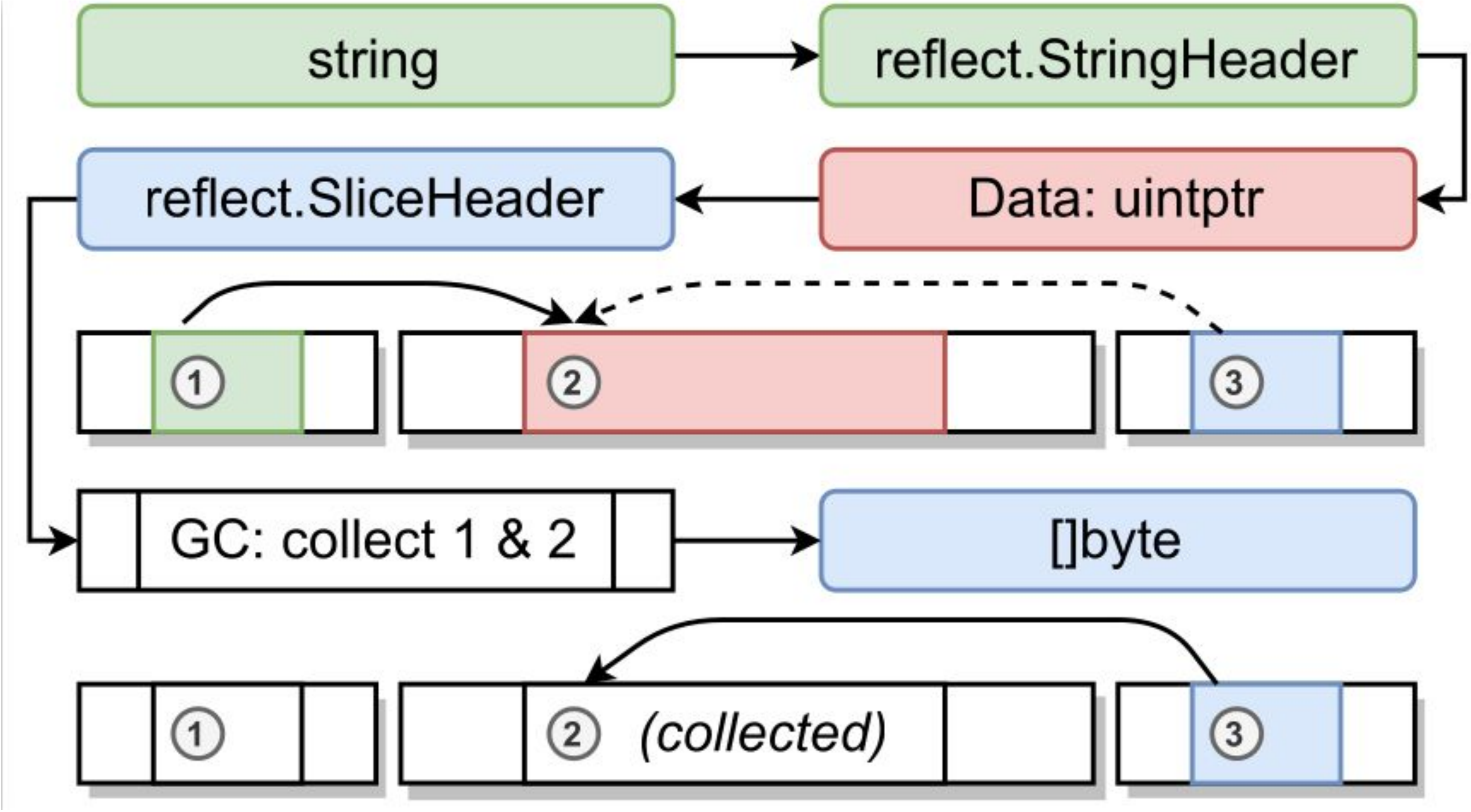
# Architecture-Dependent Types

```
type PinkStruct struct {
   A int
   B uint8
}
type VioletStruct struct {
   A int64
   B uint8
}

func main() {
   pink := PinkStruct{A: 1, B: 42}
   violet := *(*VioletStruct)(unsafe.Pointer(&pink))
}
```

# Incorrect Slice Casts

```
func StringToBytes(s string) []byte {

   strHeader := (*reflect.StringHeader)
                    (unsafe.Pointer(&s))

   bytesHeader := reflect.SliceHeader{
      Data: strHeader.Data,
      Cap: strHeader.Len,
      Len: strHeader.Len,
   }

   return *(*[]byte)(unsafe.Pointer(&bytesHeader))
}
```

# Incorrect Slice Casts

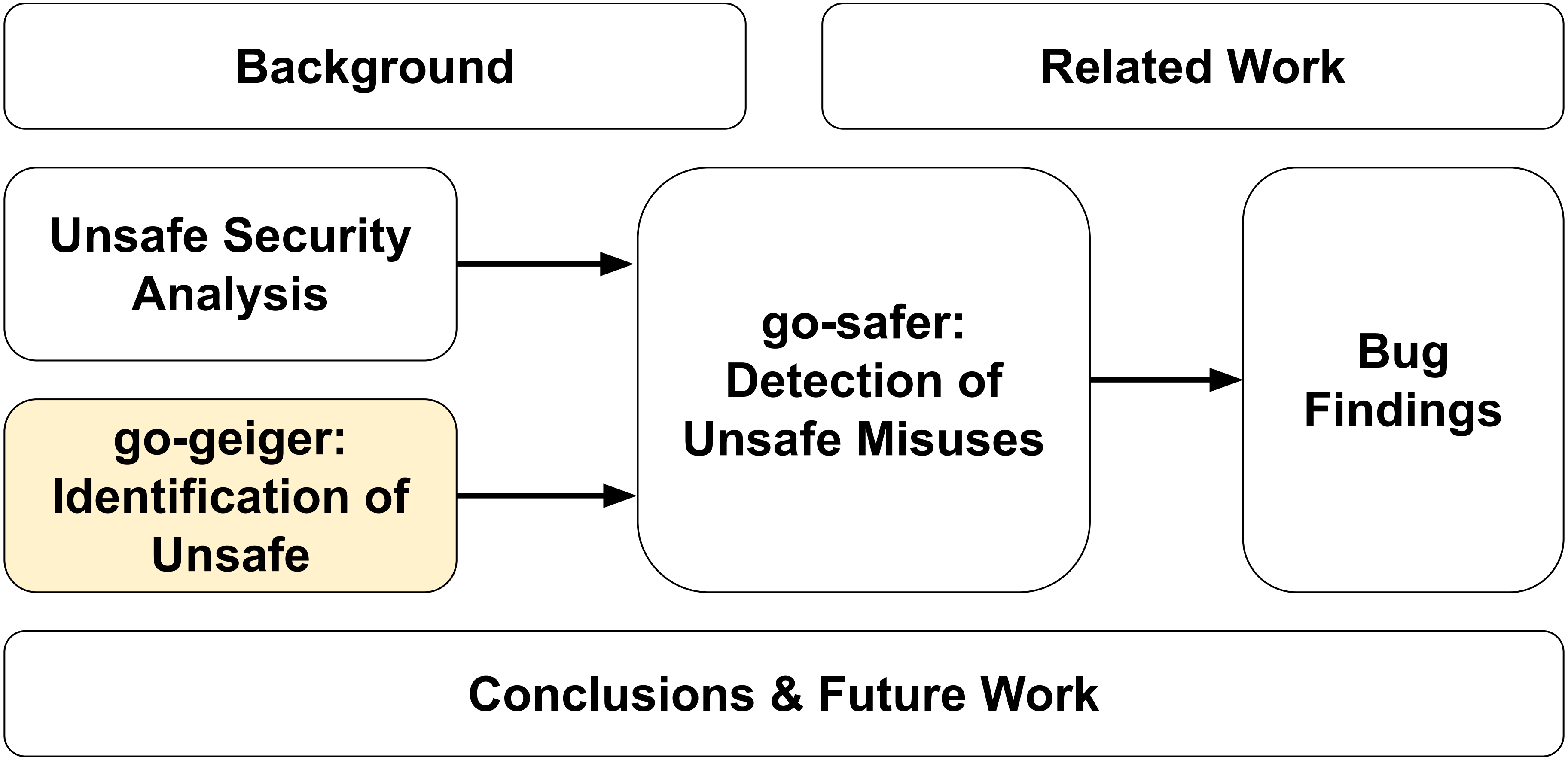


# Buffer Overflow

- Incorrect Slice Lengths
- Dangling Pointers

ROP Exploit

# Outline



Background

Related Work

Unsafe Security Analysis

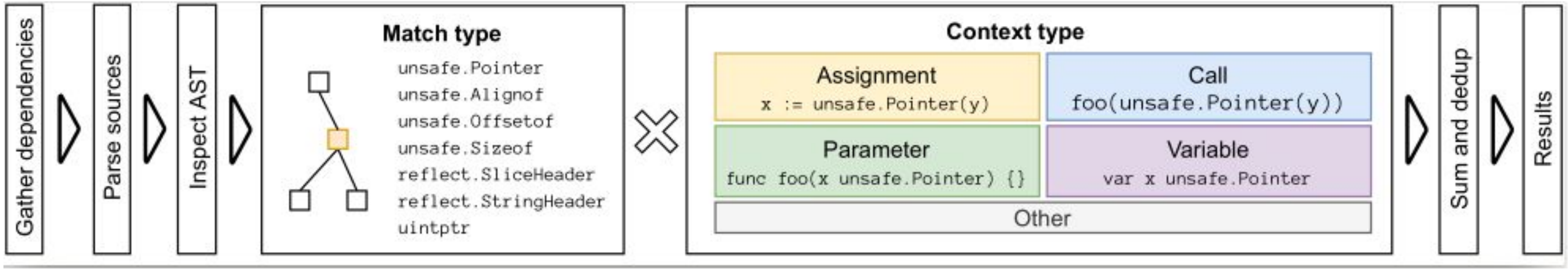
go-geiger: Identification of Unsafe

go-safer: Detection of Unsafe Misuses

Bug Findings

Conclusions & Future Work

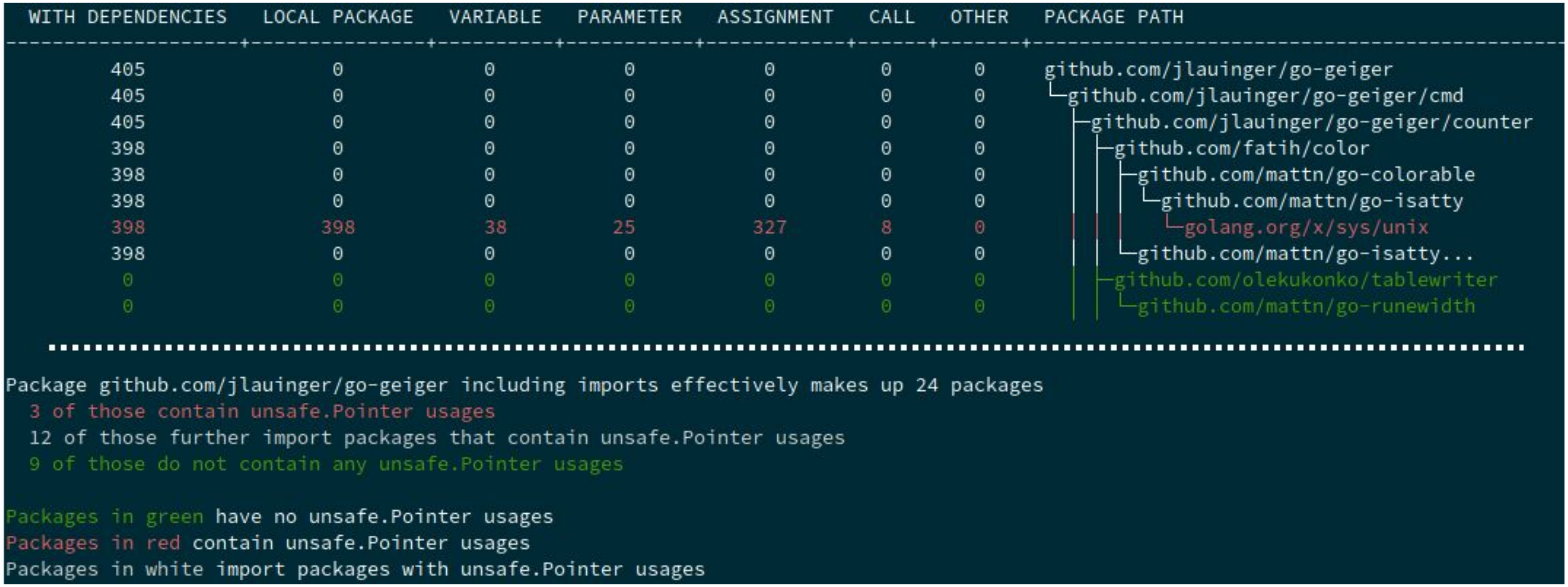
# *go-geiger*: Identification of Unsafe Usages



Key Features:

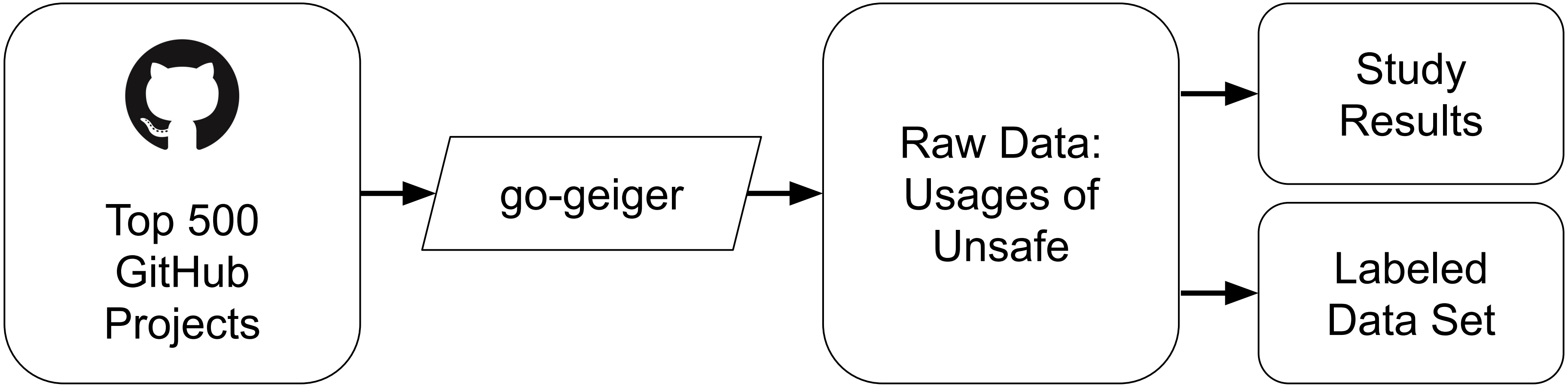
- Includes dependencies
- Counts divided by match and context types

# *go-geiger*: Screenshot



```
  WITH DEPENDENCIES   LOCAL PACKAGE   VARIABLE   PARAMETER   ASSIGNMENT   CALL   OTHER   PACKAGE PATH
--------------------+---------------+----------+-----------+------------+------+-------+------------------------------------------
        405                0             0          0            0         0       0     github.com/jlauinger/go-geiger
        405                0             0          0            0         0       0     └github.com/jlauinger/go-geiger/cmd
        405                0             0          0            0         0       0      ├github.com/jlauinger/go-geiger/counter
        398                0             0          0            0         0       0      │ ├github.com/fatih/color
        398                0             0          0            0         0       0      │ │ ├github.com/mattn/go-colorable
        398                0             0          0            0         0       0      │ │ │ └github.com/mattn/go-isatty
        398               398           38         25           327        8       0      │ │ │   └golang.org/x/sys/unix
        398                0             0          0            0         0       0      │ │ └github.com/mattn/go-isatty...
         0                 0             0          0            0         0       0      │ ├github.com/olekukonko/tablewriter
         0                 0             0          0            0         0       0      │ │ └github.com/mattn/go-runewidth

   ..........................................................................................

Package github.com/jlauinger/go-geiger including imports effectively makes up 24 packages
  3 of those contain unsafe.Pointer usages
  12 of those further import packages that contain unsafe.Pointer usages
  9 of those do not contain any unsafe.Pointer usages

Packages in green have no unsafe.Pointer usages
Packages in red contain unsafe.Pointer usages
Packages in white import packages with unsafe.Pointer usages
```

# Evaluation



Top 500 GitHub Projects

go-geiger

Raw Data: Usages of Unsafe

Study Results

Labeled Data Set

# Research Questions

- RQ1: How prevalent is *unsafe* in Go projects?
- RQ2: How deep is *unsafe* code buried in dependencies?
- RQ3: Which *unsafe* keywords are used most?
- RQ4: Does *unsafe* usage correlate with project metrics?
- RQ5: How does *unsafe* usage change in code’s lifetime?
- RQ6: How does go-geiger compare to existing linters?
- RQ7: What unsafe operations are used in practice, and for which purpose?

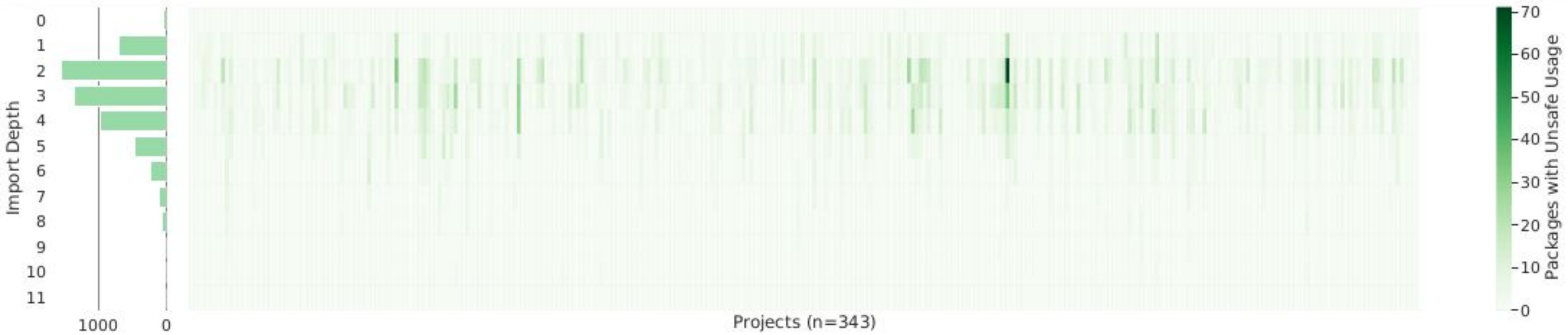
# RQ1: Prevalence of Unsafe

Prevalence in Projects:

- No usage: 31 (9%)
- Direct usage: 131 (38%)
- Transitive usage: 312 (91%)

343 out of 500 projects analyzed

# RQ2: Unsafe Dependency Depth



Most packages using *unsafe* imported around a depth of 3.

# RQ7: Purpose of Unsafe
# Labeled Data Set of Usages

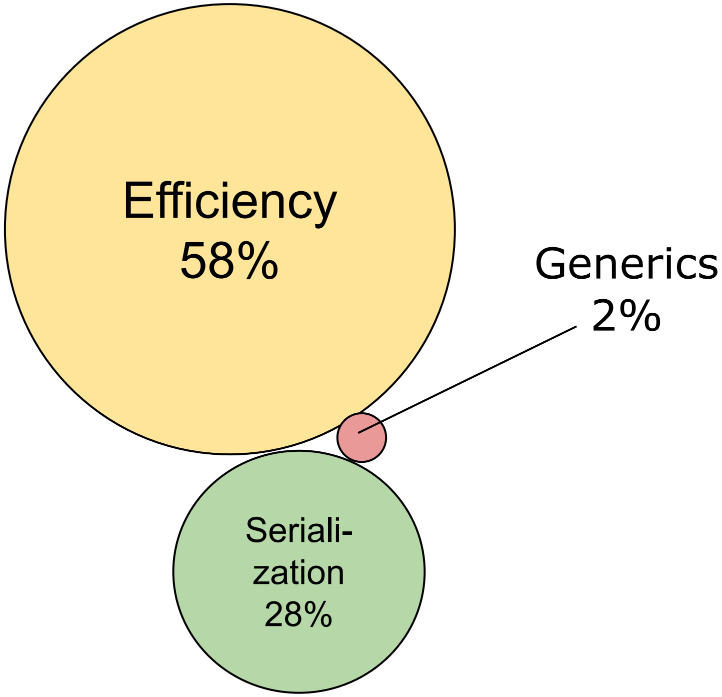
1,400 Code samples (one line)
Taken from 10 projects incl. dependencies

Labeled in two dimensions

What is done? × For what purpose?

# RQ7: Purpose of Unsafe

## Application Code

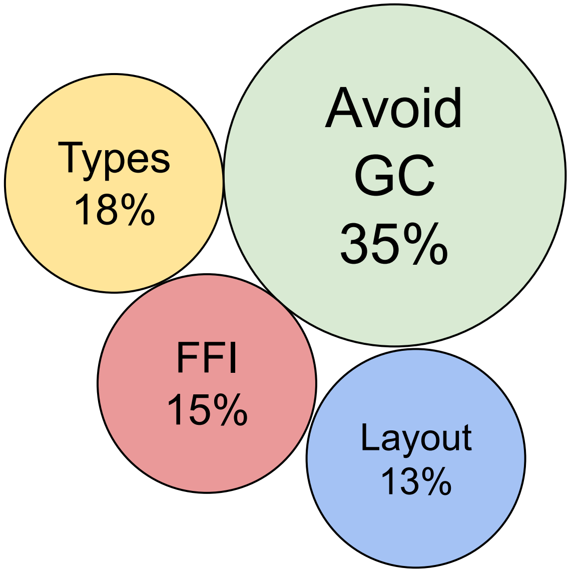


Efficiency 58%

Generics 2%

Serialization 28%

## Standard Library



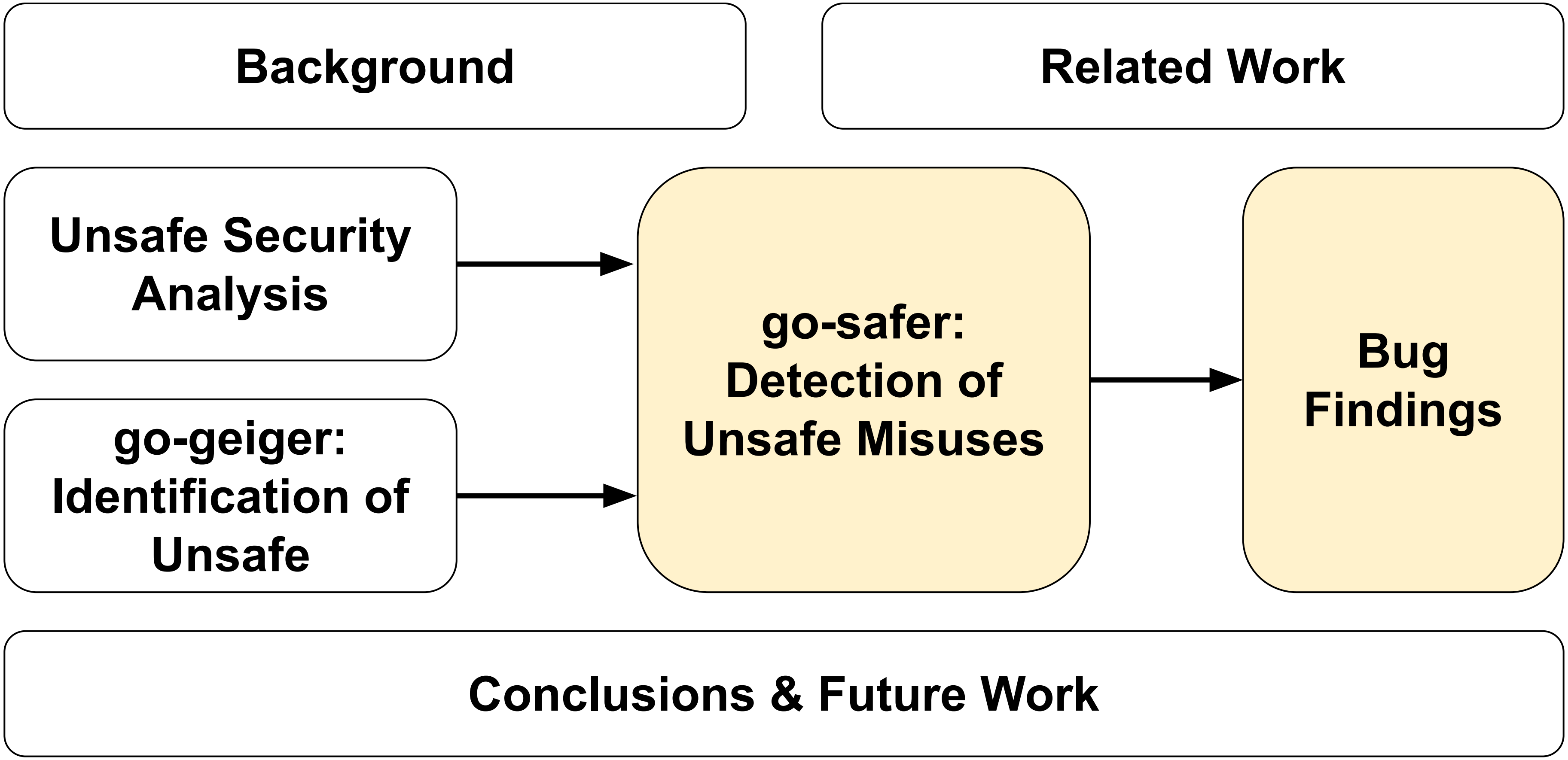
Types 18%

Avoid GC 35%

FFI 15%

Layout 13%

# Outline



Background

Related Work

Unsafe Security Analysis

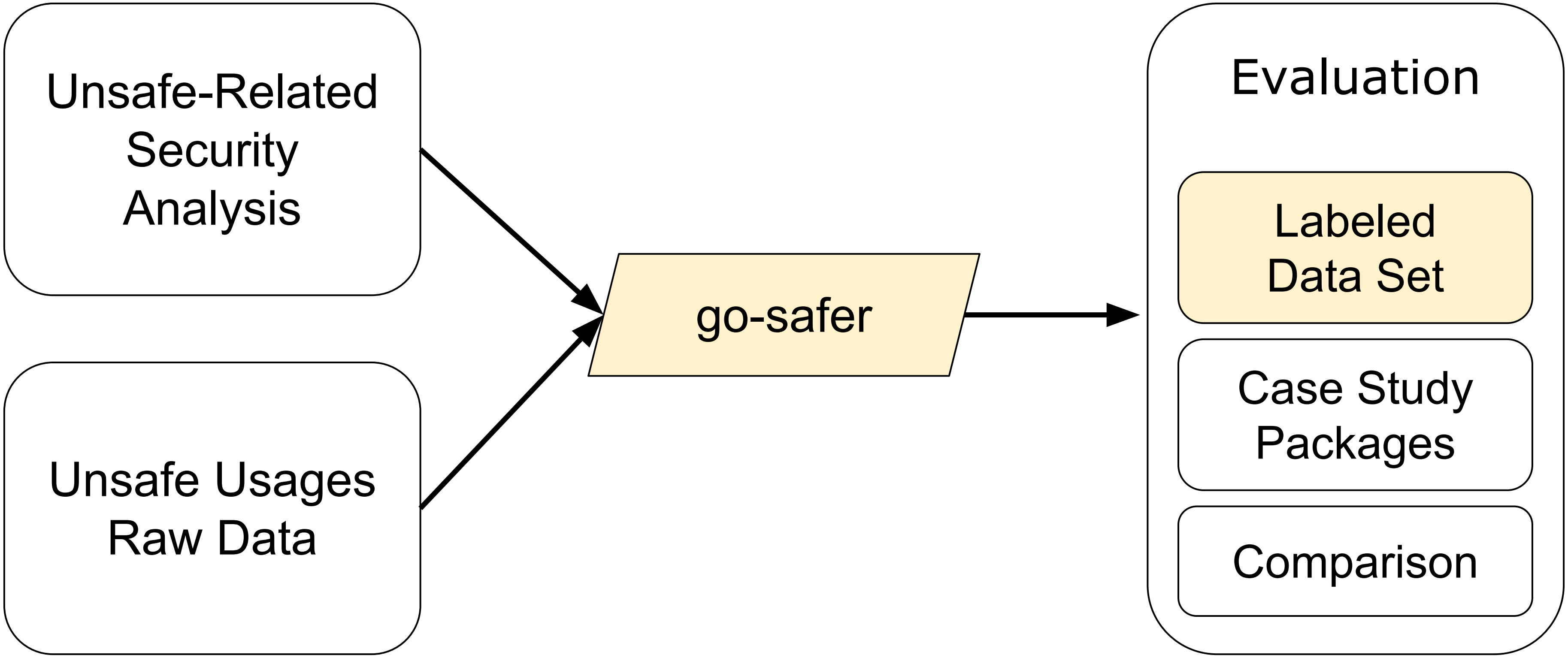
go-geiger: Identification of Unsafe

go-safer: Detection of Unsafe Misuses

Bug Findings

Conclusions & Future Work

# *go-safer*: Detecting Misuses of *Unsafe*



# Detected Pattern 1

```
type A struct {
   x int
}
type B struct {
   y int64
}
func unsafeFunction(a A) B {
   return *(*B)(unsafe.Pointer(&a))
}
```

Conversions between architecture-dependent types

# Detected Pattern 2

```
func unsafeFunction(s string) []byte {
   sH := (*reflect.StringHeader)(unsafe.Pointer(&s))
   bH := &reflect.SliceHeader{
      Data: sH.Data,
      Len: sH.Len,
      Cap: sH.Len,
   }
   return *(*[]byte)(unsafe.Pointer(bH))
}
```

Invalid constructions of slice and string headers

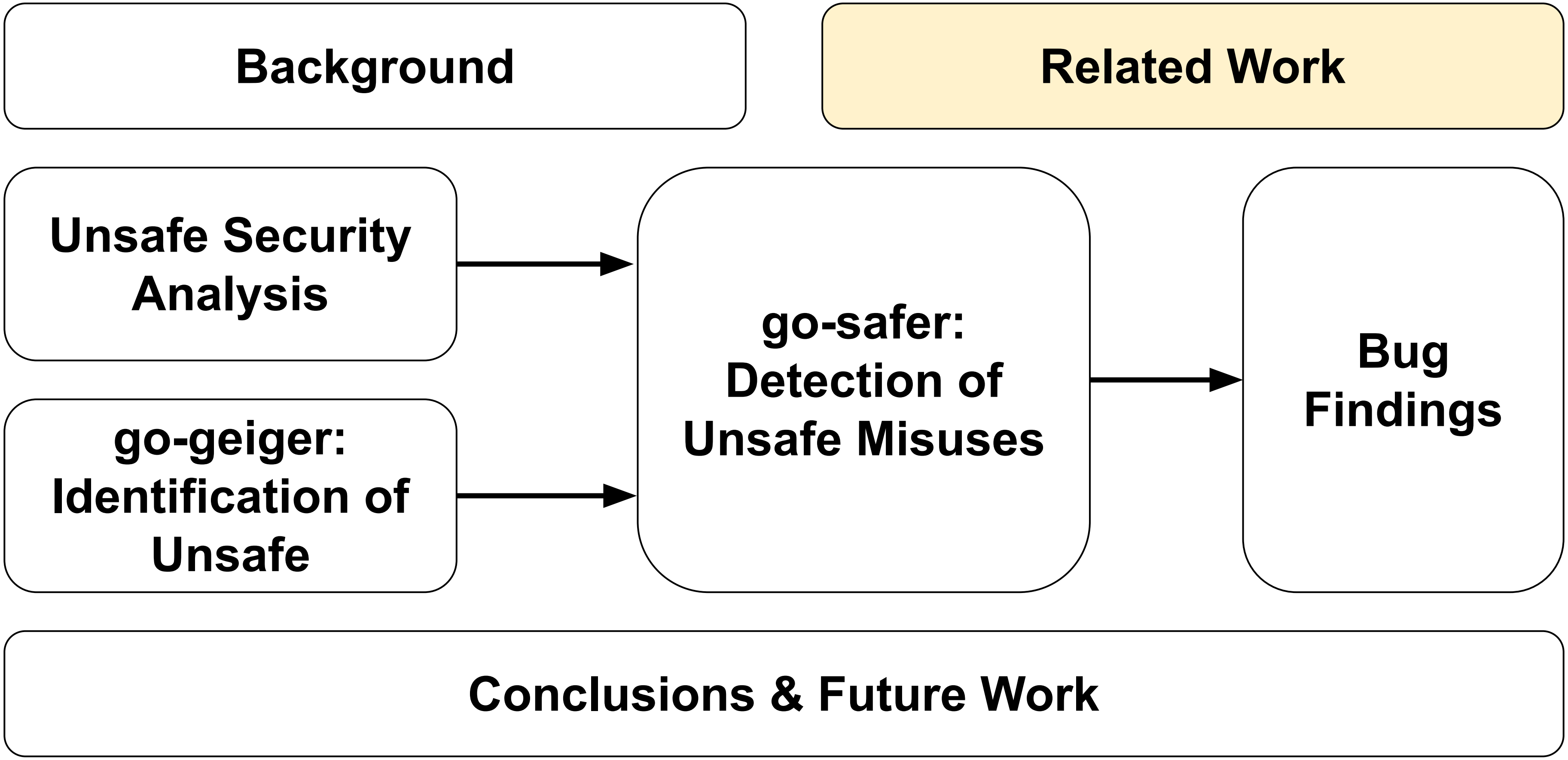
# Evaluation on Labeled Data Set

| TP | FP | TN | FN |
|---|---|---|---|
| 29 | 1 | 13 | 1 |
| Precision | Recall | Accuracy | F1-Score |
| 0.967 | 0.967 | 0.955 | 0.967 |

# Bug Findings

| | Project | Popularity | Bugs | Merged |
|---|---|---|---|---|
| 1 | hanwen/go-fuse | 3 project(s) | 1 | no |
| 2 | buger/jsonparser | 4 project(s) | 1 | yes |
| 3 | elastic/go-structform | 1 project(s) | 2 | yes |
| 4 | go-fiber/utils | 1 project(s) | 1 | yes |
| 5 | influxdata/influxdb | 8 project(s) | 1 | no |
| 6 | influxdata/influxdb1-client | 4 project(s) | 1 | no |
| 7 | modern-go/reflect2 | 71 project(s) | 1 | yes |
| 8 | savsgio/gotils | 1 project(s) | 2 | yes |
| 9 | valyala/fasttemplate | 10 project(s) | 1 | yes |
| 10 | weaveworks/ps | 1 project(s) | 1 | no |
| 11 | yuin/goldmark | 5 project(s) | 1 | yes |
| 12 | yuin/gopher-lua | 6 project(s) | 1 | yes |
| 13 | gorgonia/tensor | 1 project(s) | 1 | yes |
| 14 | gorgonia/tensor (second PR) | 1 project(s) | 48 | yes |
| 15 | mailru/easyjson | 42 project(s) | 1 | no |
| | **Total** | | **64** | **10 / 15** |

# Outline



Background

Related Work

Unsafe Security Analysis

go-geiger: Identification of Unsafe

go-safer: Detection of Unsafe Misuses

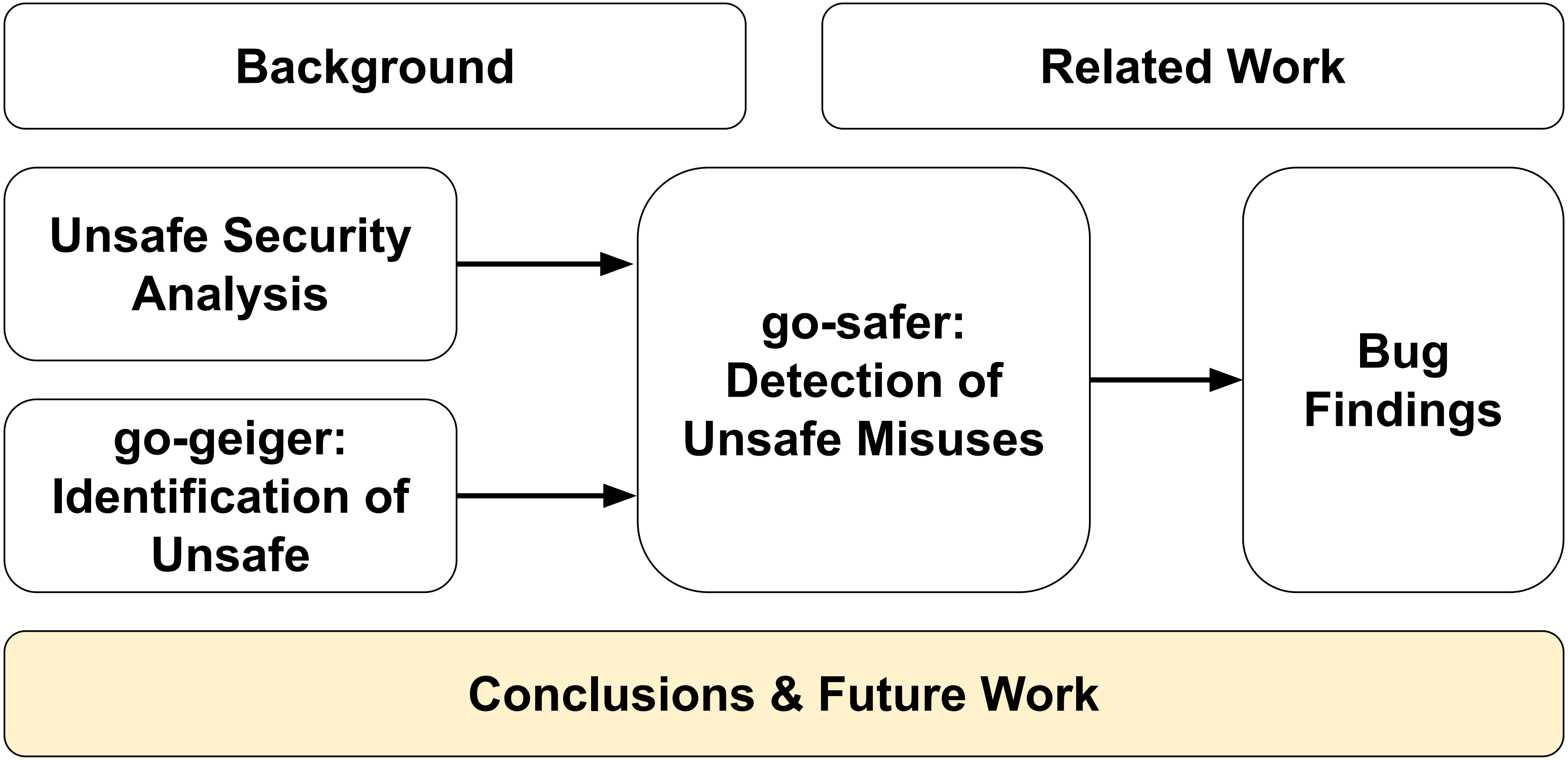
Bug Findings

Conclusions & Future Work

# Related Work

- Concurrent Study on Unsafe Go by Costa et al. [1]: similar results
- Unsafe APIs in other languages: Rust [2,3], Java [4]. About 30% of projects use directly, >50% use transitively
- Dependencies often introduce vulnerabilities [5], and are updated slowly [6,7]

# Outline



Background

Related Work

Unsafe Security Analysis

go-geiger: Identification of Unsafe

go-safer: Detection of Unsafe Misuses

Bug Findings

Conclusions & Future Work

# Improvements in Future Go Versions

- New compiler flag: -d=checkptr
- New type for slice headers
- Support for generics

# Conclusion

- Unsafe API in Go can introduce vulnerabilities
- Use after free, buffer overflow => possible code injection and information leak
- Unsafe API used commonly in open-source projects
- go-geiger: detect unsafe in dependencies
- go-safer: find some misuses of unsafe

# Future Work

- Quantify performance gain offered by unsafe
- ML on labeled data set
- Static verification techniques
- Public documentation of good and bad unsafe usage examples

# Thanks for your attention!

Questions?

# Sources

Literature:

- [1] Diego Elias Costa, Suhaib Mujahid, Rabe Abdalkareem, and Emad Shihab. Breaking Type-Safety in Go: An Empirical Study on the Usage of the unsafe Package. arXiv:2006.09973 [cs], June 2020
- [2] Ana Nora Evans, Bradford Campbell, and Mary Lou Soffa. Is Rust Used Safely by Software Developers? In 42nd International Conference on Software Engineering (ICSE ’20), Seoul, Republic of Korea, May 2020. Association for Computing Machinery, New York, NY, USA
- [3] Boqin Qin, Yilun Chen, Zeming Yu, Linhai Song, and Yiying Zhang. Understanding Memory and Thread Safety Practices and Issues in Real-World Rust Programs. In Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation, pages 763–779, New York, NY, USA, June 2020. Association for Computing Machinery
- [4] Luis Mastrangelo, Luca Ponzanelli, Andrea Mocci, Michele Lanza, Matthias Hauswirth, and Nathaniel Nystrom. Use at your own risk: The Java unsafe API in the wild. ACM SIGPLAN Notices, 50(10):695–710, October 2015
- [5] Takuya Watanabe, Mitsuaki Akiyama, Fumihiro Kanei, Eitaro Shioji, Yuta Takata, Bo Sun, Yuta Ishi, Toshiki Shibahara, Takeshi Yagi, and Tatsuya Mori. Understanding the Origins of Mobile App Vulnerabilities: A Large-scale Measurement Study of Free and Paid Apps. In 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), pages 14–24, Buenos Aires, Argentina, May 2017. IEEE

# Sources

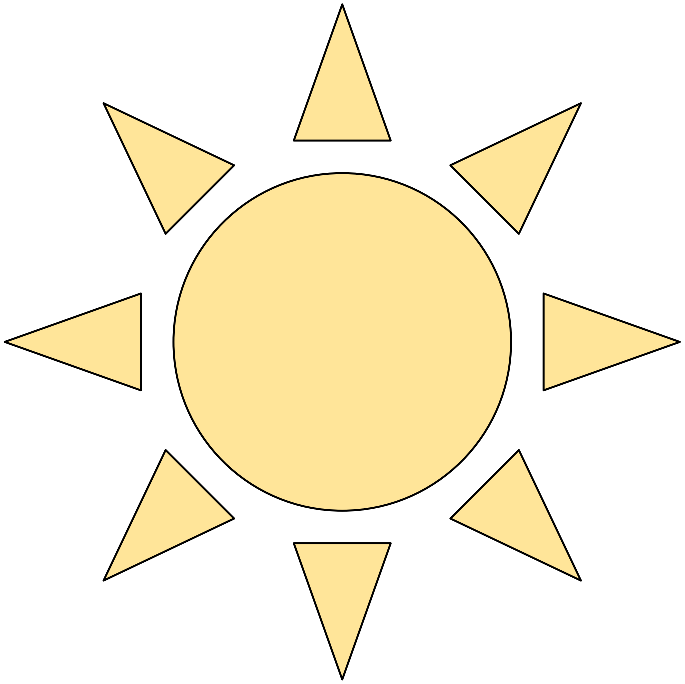
Literature:

- [6] Raula Gaikovina Kula, Daniel M. German, Ali Ouni, Takashi Ishio, and Katsuro Inoue. Do developers update their library dependencies? In Empirical Software Engineering (2018), pages 384–417, New York, May 2017. Springer Science+Business Media
- [7] Samim Mirhosseini and Chris Parnin. Can automated pull requests encourage software developers to upgrade out-of-date dependencies? In 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), pages 84–94, Urbana, IL, USA, October 2017. IEEE
- [8] https://msrc-blog.microsoft.com/2019/07/16/a-proactive-approach-to-more-secure-code

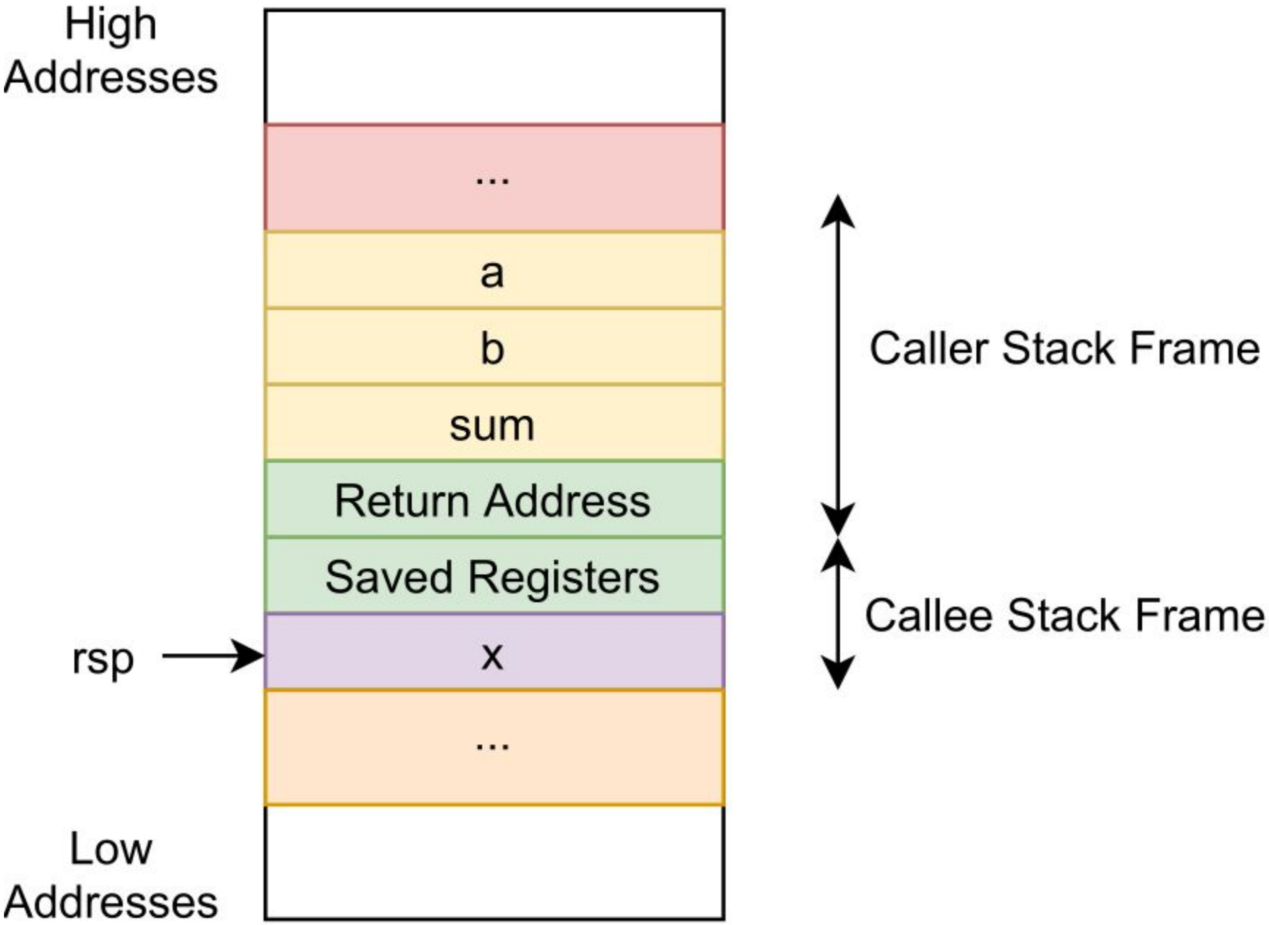
Images:

- Title Image: https://medium.com/a-journey-with-go/go-what-is-the-unsafe-package-d2443da36350
- GitHub Logo: https://github.com/logos

# Backup Slides



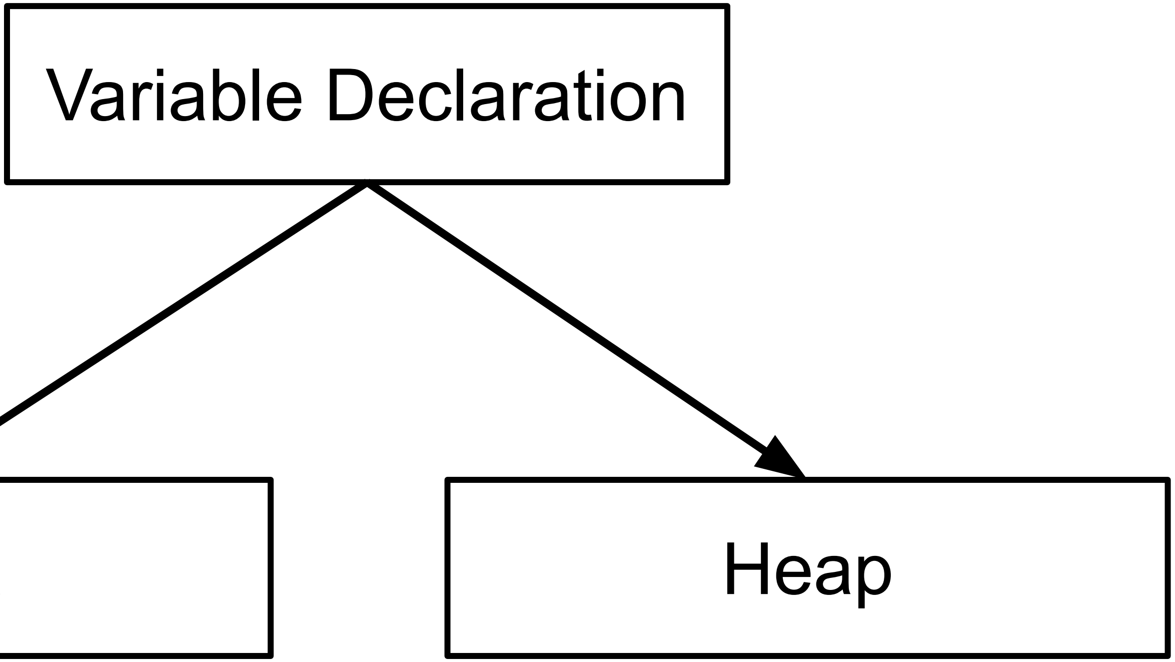
# Memory Management: Stack



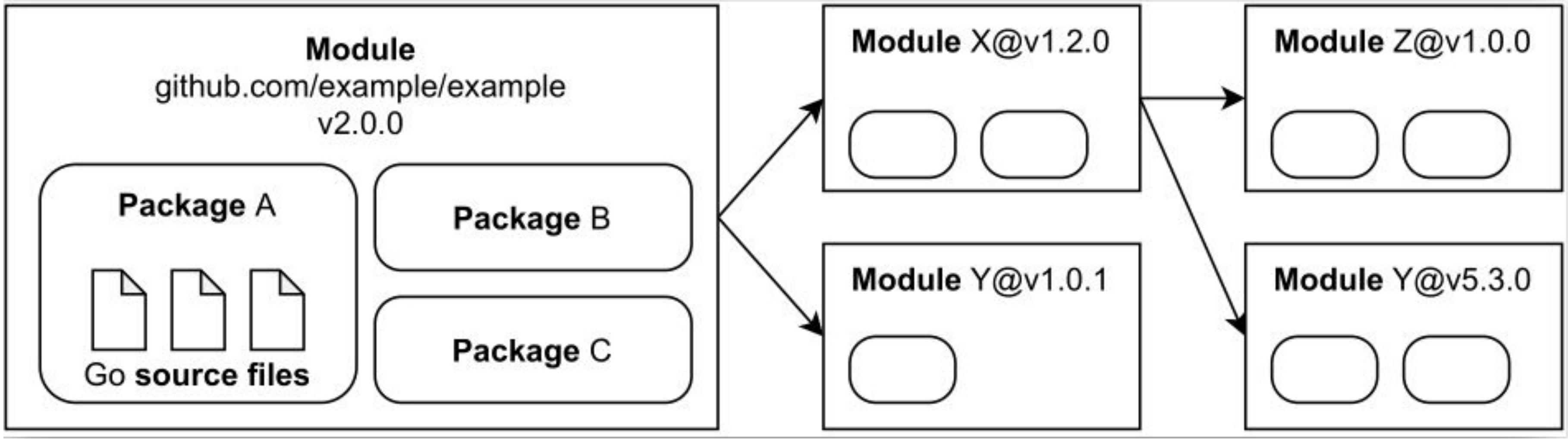
# Memory Management: EA

TECHNISCHE UNIVERSITÄT DARMSTADT

## Escape Analysis

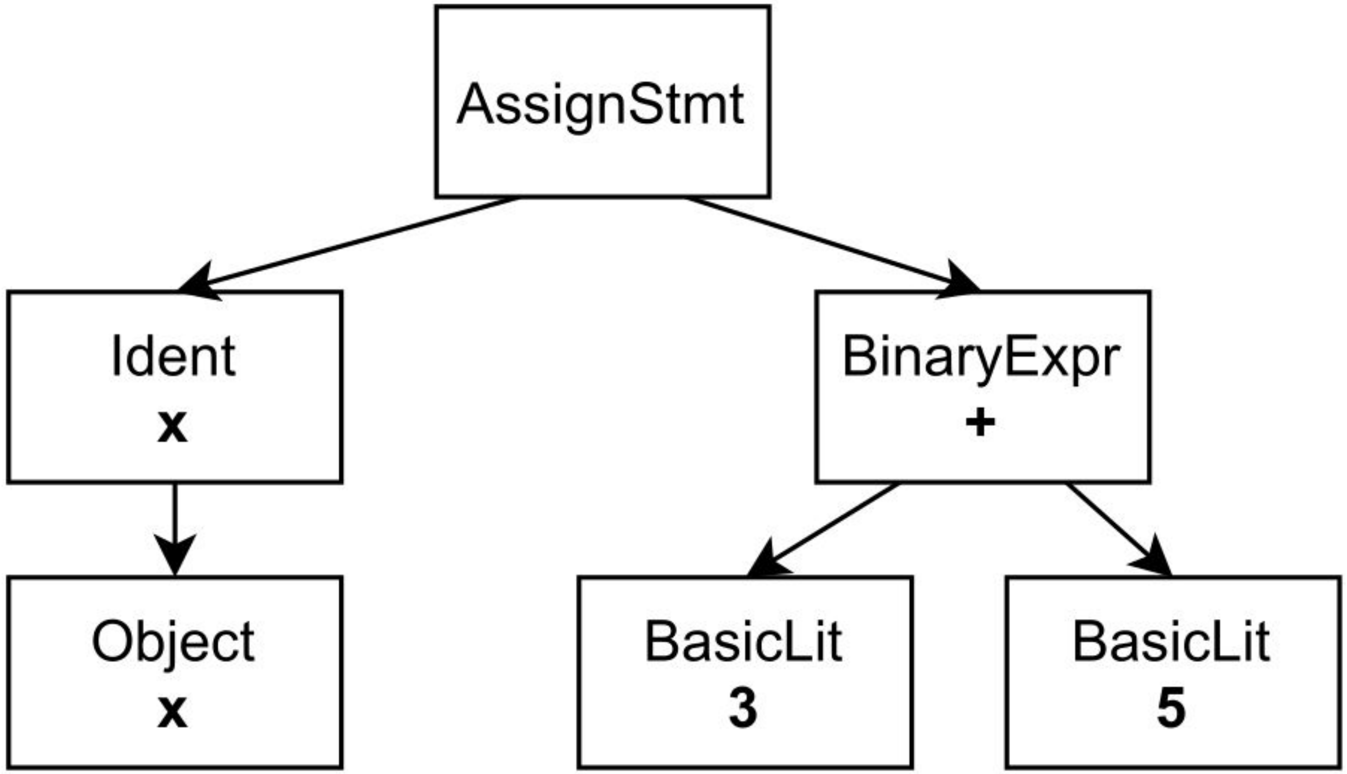


# Dependency Management in Go



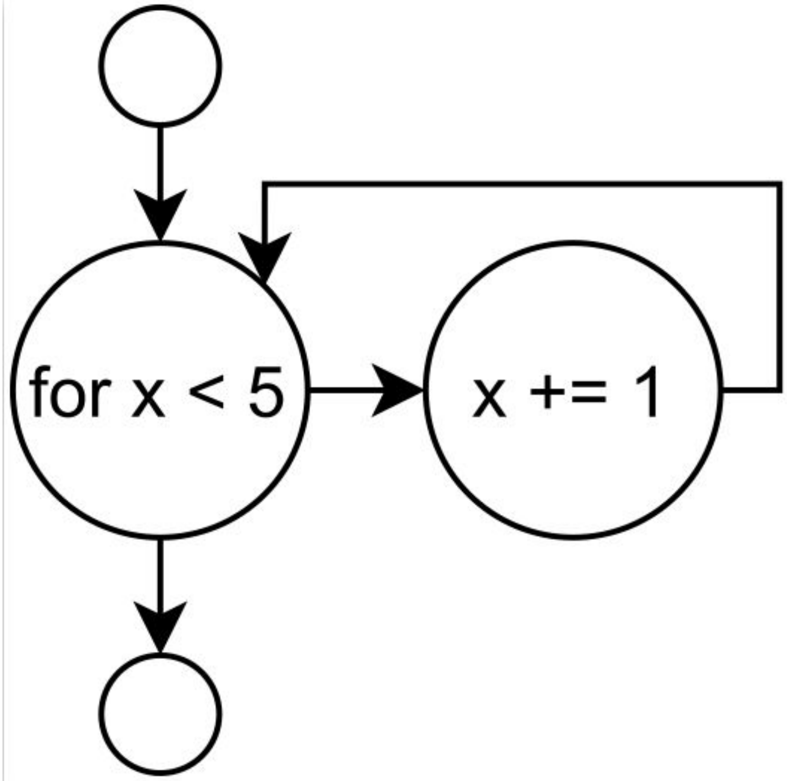
# Abstract Syntax Tree

x := 3 + 5

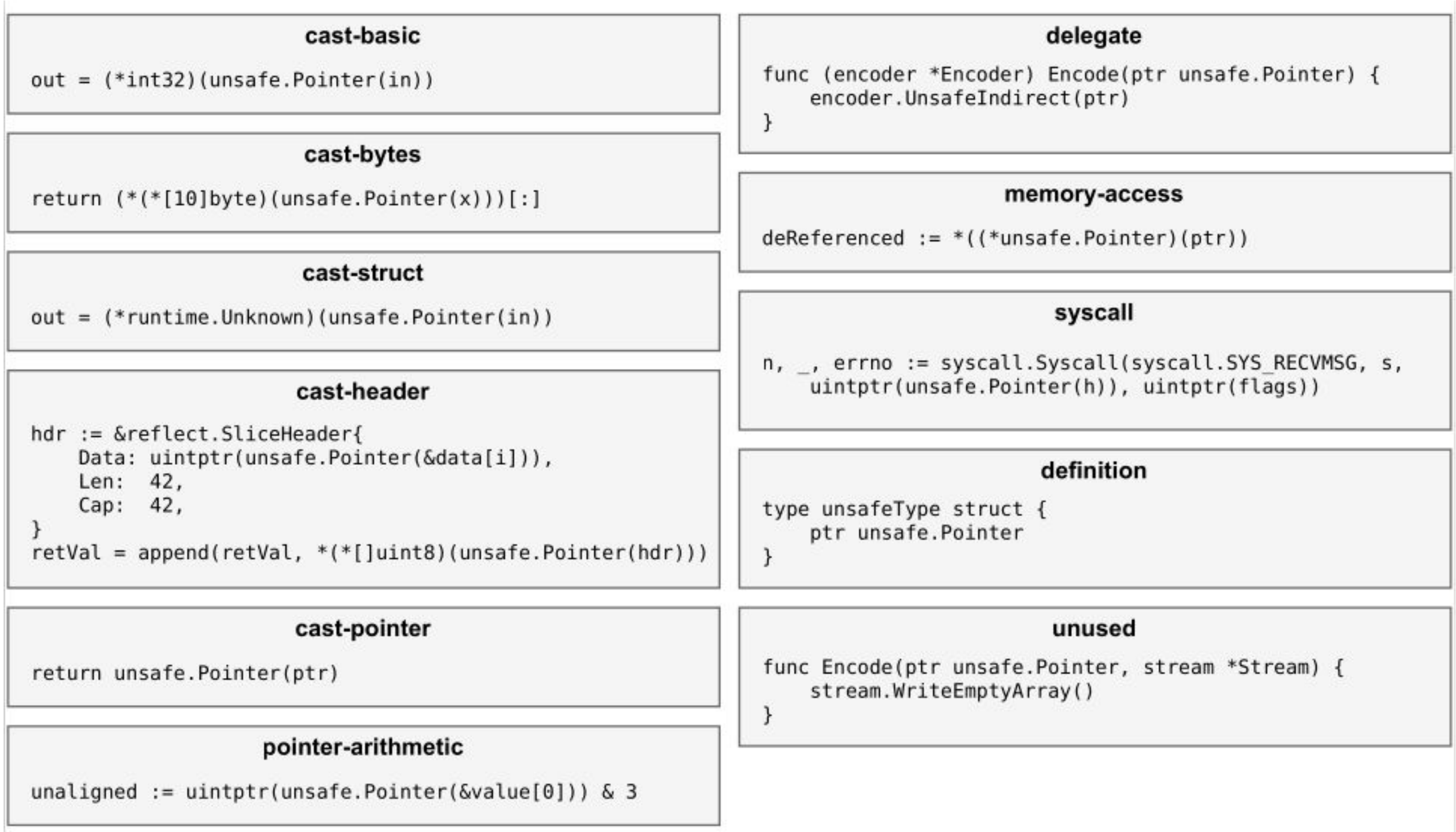


# Control Flow Graph

for x < 5 { x += 1 }



# Labeled Data Set
# What is done dimension



**cast-basic**

```
out = (*int32)(unsafe.Pointer(in))
```

**cast-bytes**

```
return (*(*[10]byte)(unsafe.Pointer(x)))[:]
```

**cast-struct**

```
out = (*runtime.Unknown)(unsafe.Pointer(in))
```

**cast-header**

```
hdr := &reflect.SliceHeader{
    Data: uintptr(unsafe.Pointer(&data[i])),
    Len:  42,
    Cap:  42,
}
retVal = append(retVal, *(*[]uint8)(unsafe.Pointer(hdr)))
```

**cast-pointer**

```
return unsafe.Pointer(ptr)
```

**pointer-arithmetic**

```
unaligned := uintptr(unsafe.Pointer(&value[0])) & 3
```

**delegate**

```
func (encoder *Encoder) Encode(ptr unsafe.Pointer) {
    encoder.UnsafeIndirect(ptr)
}
```

**memory-access**

```
deReferenced := *((*unsafe.Pointer)(ptr))
```

**syscall**

```
n, _, errno := syscall.Syscall(syscall.SYS_RECVMSG, s,
    uintptr(unsafe.Pointer(h)), uintptr(flags))
```

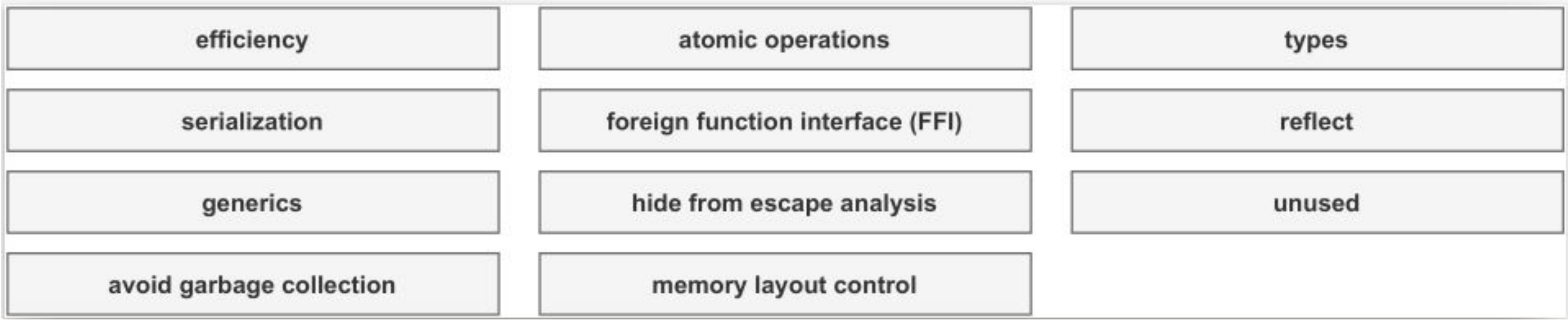
**definition**

```
type unsafeType struct {
    ptr unsafe.Pointer
}
```

**unused**

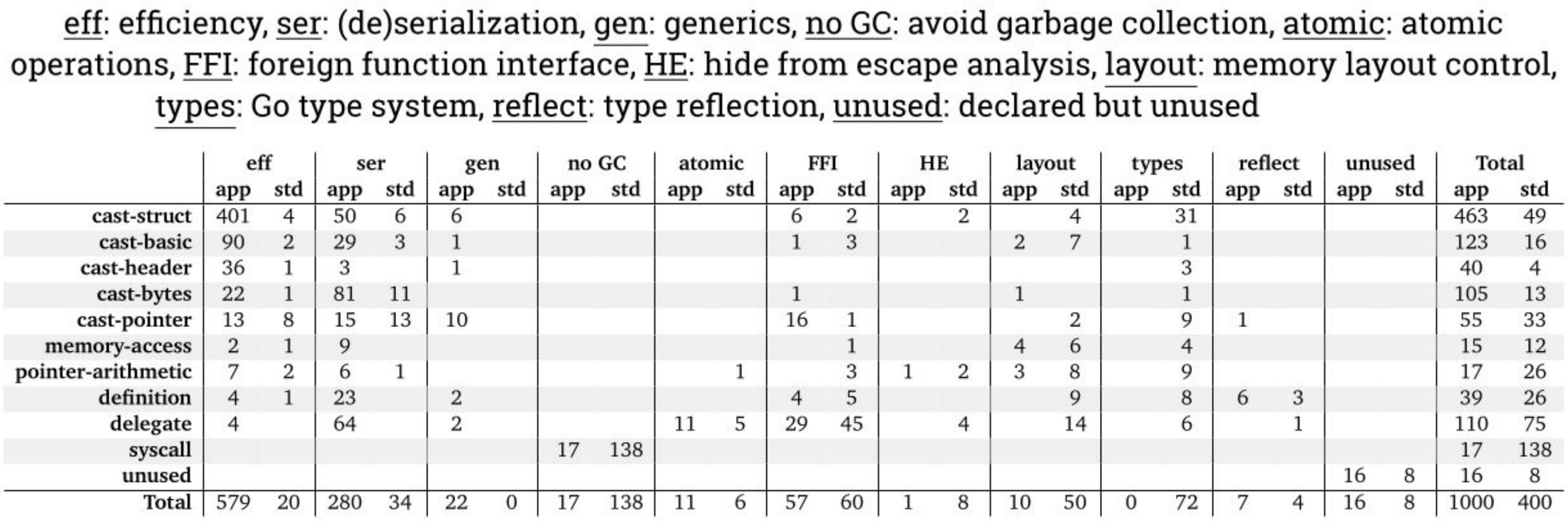
```
func Encode(ptr unsafe.Pointer, stream *Stream) {
    stream.WriteEmptyArray()
}
```

# Labeled Data Set
# Purpose dimension



- efficiency
- serialization
- generics
- avoid garbage collection
- atomic operations
- foreign function interface (FFI)
- hide from escape analysis
- memory layout control
- types
- reflect
- unused

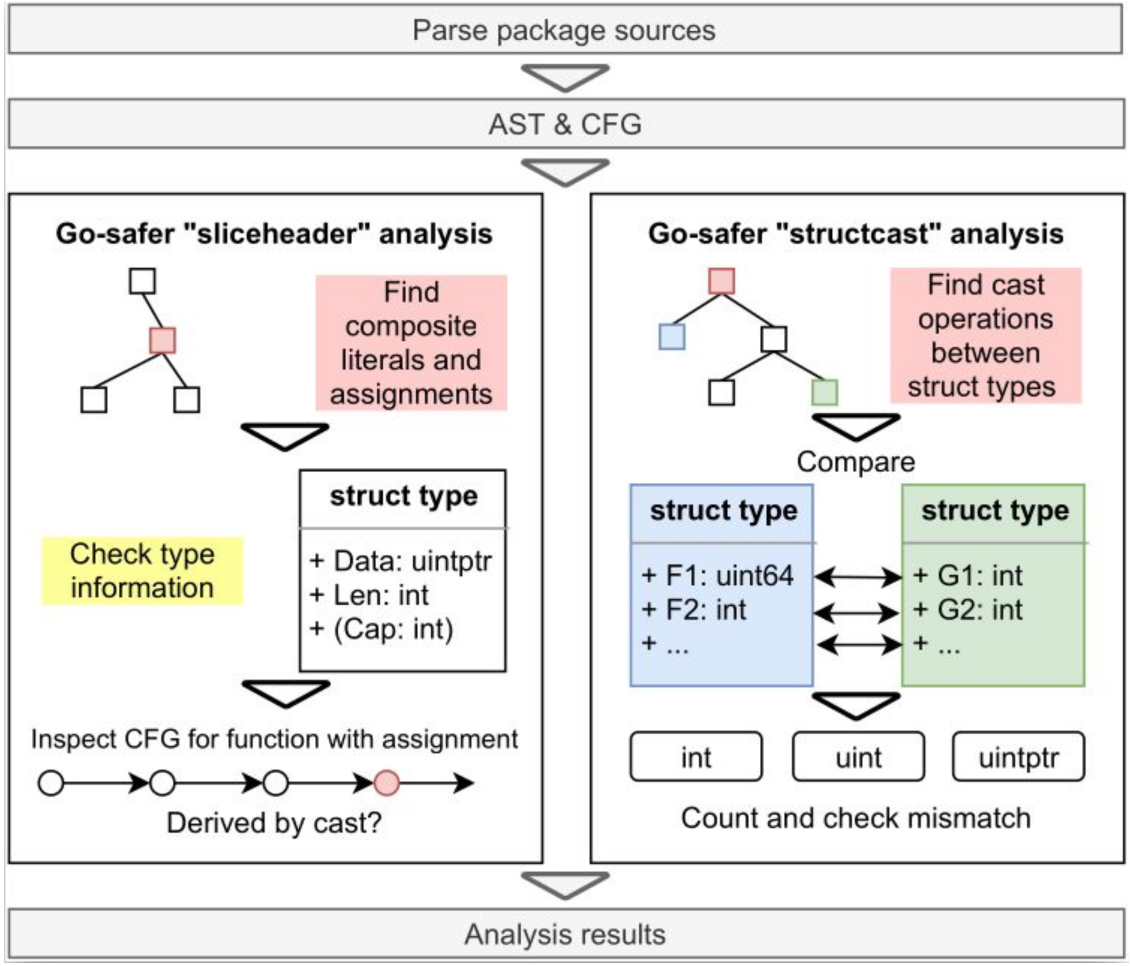
# Labeled Data Set
# Classes Counts



eff: efficiency, ser: (de)serialization, gen: generics, no GC: avoid garbage collection, atomic: atomic operations, FFI: foreign function interface, HE: hide from escape analysis, layout: memory layout control, types: Go type system, reflect: type reflection, unused: declared but unused

| | eff | | ser | | gen | | no GC | | atomic | | FFI | | HE | | layout | | types | | reflect | | unused | | Total | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | app | std | app | std | app | std | app | std | app | std | app | std | app | std | app | std | app | std | app | std | app | std | app | std |
| cast-struct | 401 | 4 | 50 | 6 | 6 | | | | | | 6 | 2 | | 2 | | 4 | | 31 | | | | | 463 | 49 |
| cast-basic | 90 | 2 | 29 | 3 | 1 | | | | | | 1 | 3 | | | 2 | 7 | | 1 | | | | | 123 | 16 |
| cast-header | 36 | 1 | 3 | | 1 | | | | | | | | | | | | | 3 | | | | | 40 | 4 |
| cast-bytes | 22 | 1 | 81 | 11 | | | | | | | 1 | | | | 1 | | | 1 | | | | | 105 | 13 |
| cast-pointer | 13 | 8 | 15 | 13 | 10 | | | | | | 16 | 1 | | | | 2 | | 9 | 1 | | | | 55 | 33 |
| memory-access | 2 | 1 | 9 | | | | | | | | | 1 | | | 4 | 6 | | 4 | | | | | 15 | 12 |
| pointer-arithmetic | 7 | 2 | 6 | 1 | | | | | | 1 | | 3 | 1 | 2 | 3 | 8 | | 9 | | | | | 17 | 26 |
| definition | 4 | 1 | 23 | | 2 | | | | | | 4 | 5 | | | | 9 | | 8 | 6 | 3 | | | 39 | 26 |
| delegate | 4 | | 64 | | 2 | | | | 11 | 5 | 29 | 45 | | 4 | | 14 | | 6 | | 1 | | | 110 | 75 |
| syscall | | | | | | | 17 | 138 | | | | | | | | | | | | | | | 17 | 138 |
| unused | | | | | | | | | | | | | | | | | | | | | 16 | 8 | 16 | 8 |
| Total | 579 | 20 | 280 | 34 | 22 | 0 | 17 | 138 | 11 | 6 | 57 | 60 | 1 | 8 | 10 | 50 | 0 | 72 | 7 | 4 | 16 | 8 | 1000 | 400 |

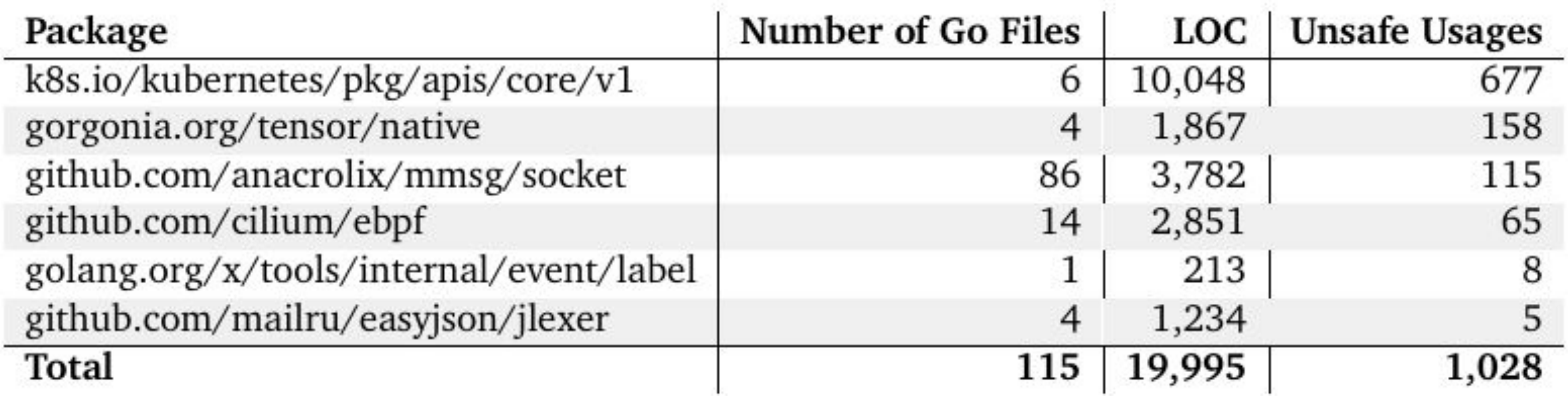
# go-safer Implementation



# Evaluation on Case Study Packages

| TP | FP | TN | FN |
|---|---|---|---|
| 49 | 10 | 969 | 0 |
| Precision | Recall | Accuracy | F1-Score |
| 0.831 | 1 | 0.990 | 0.907 |

# Evaluation on Case Study Packages



| Package | Number of Go Files | LOC | Unsafe Usages |
|---|---:|---:|---:|
| k8s.io/kubernetes/pkg/apis/core/v1 | 6 | 10,048 | 677 |
| gorgonia.org/tensor/native | 4 | 1,867 | 158 |
| github.com/anacrolix/mmsg/socket | 86 | 3,782 | 115 |
| github.com/cilium/ebpf | 14 | 2,851 | 65 |
| golang.org/x/tools/internal/event/label | 1 | 213 | 8 |
| github.com/mailru/easyjson/jlexer | 4 | 1,234 | 5 |
| **Total** | **115** | **19,995** | **1,028** |

# Threats to Validity

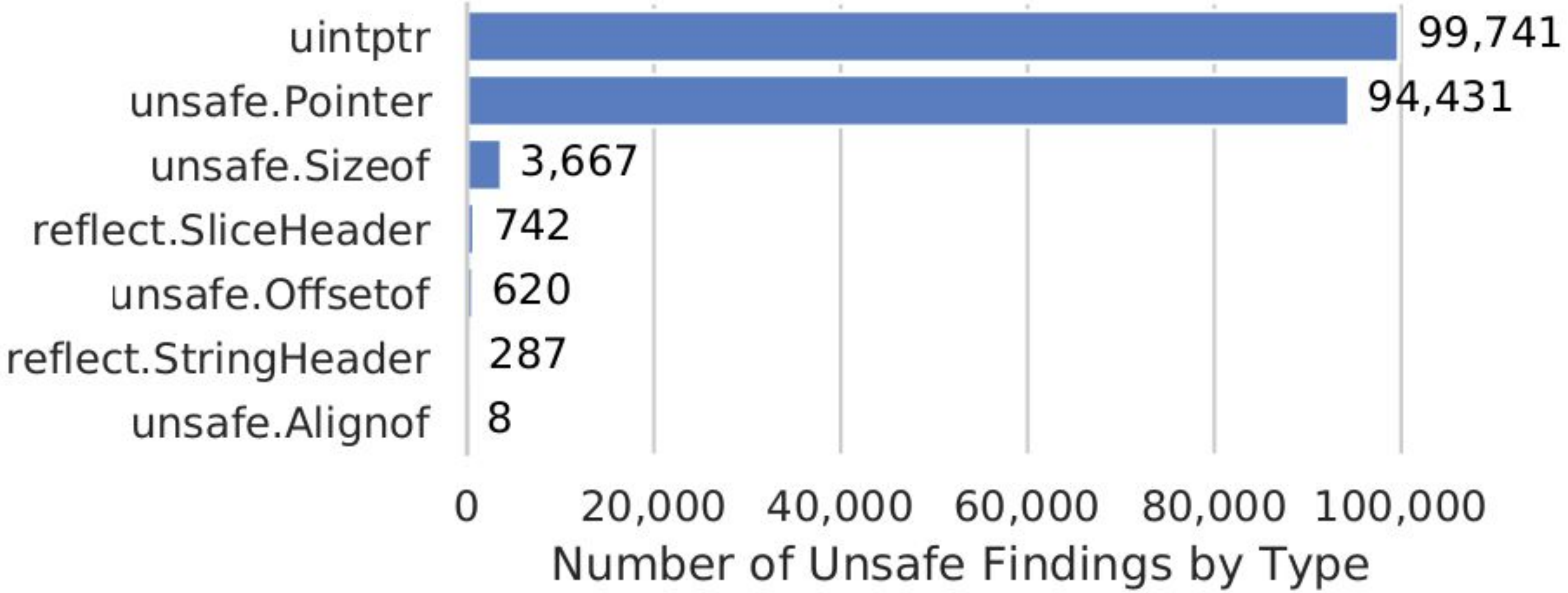
Internal

- amd64 architecture
- Manual vulnerability analysis

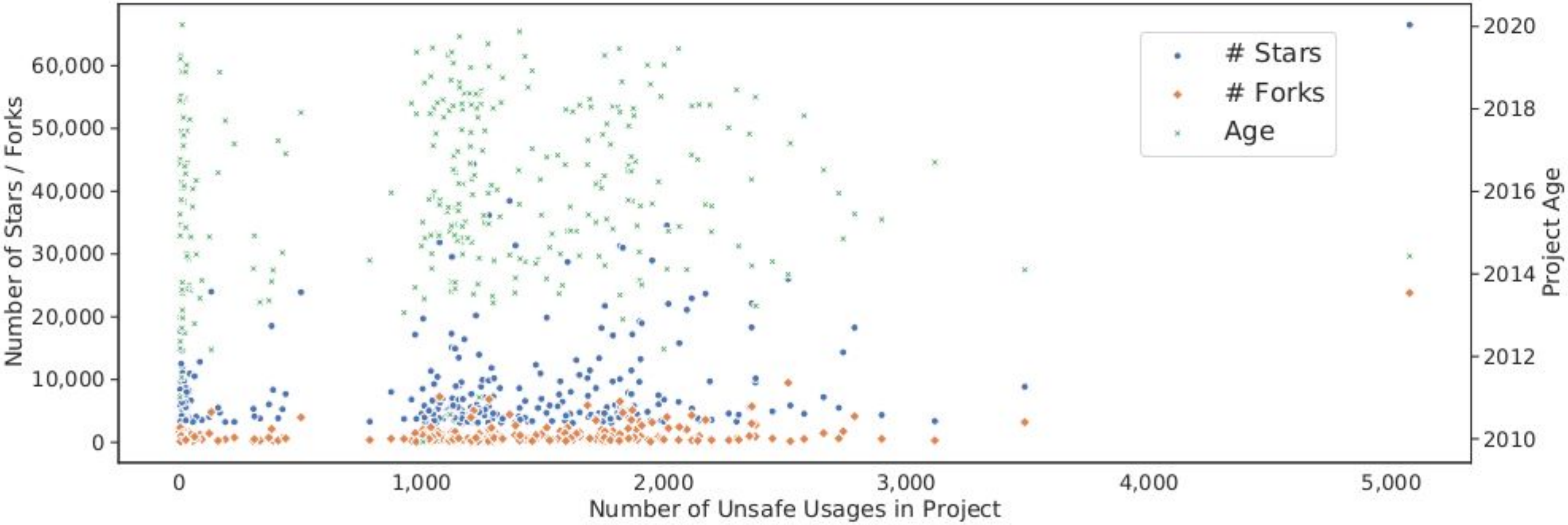
External

- Project selection
- Go module support

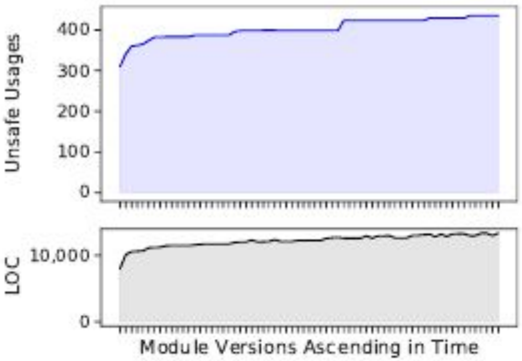
# RQ3: Which unsafe keywords are used the most?



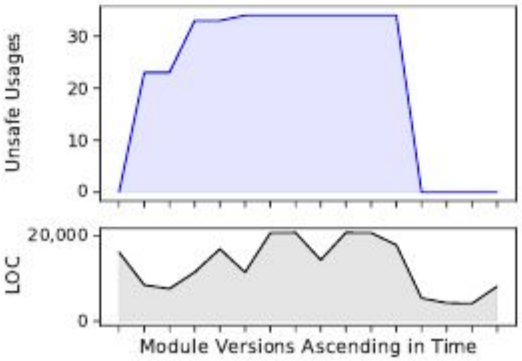
# RQ4: Correlation of unsafe with project metrics (age, popularity)



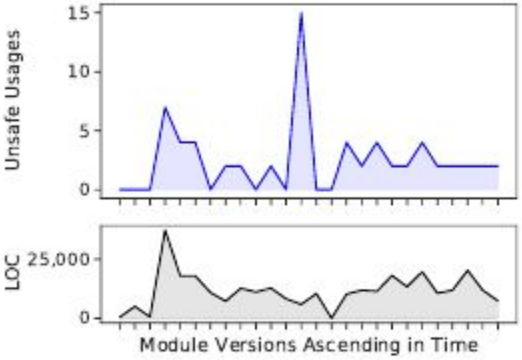
# RQ5: Change of unsafe usage over code lifetime



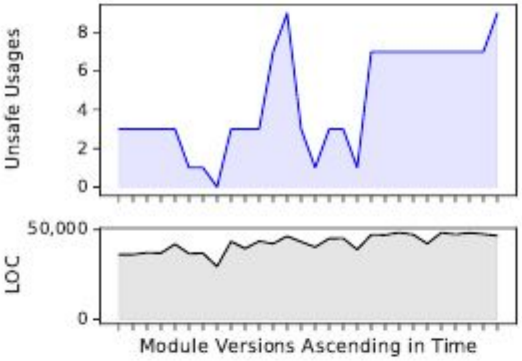
(a) golang.org/x/sys



(b) github.com/golang/protobuf

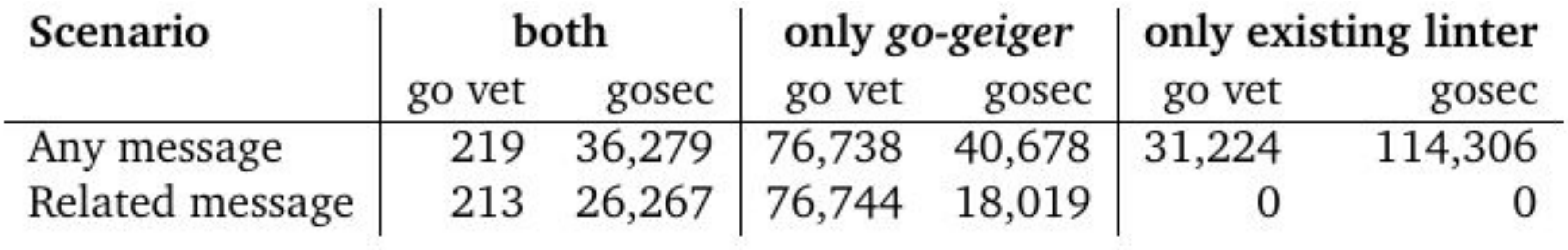


(c) github.com/docker/docker



(d) k8s.io/apimachinery

# RQ6: Comparison with existing tools (go vet, gosec)



| Scenario | both | | only *go-geiger* | | only existing linter | |
|---|---|---|---|---|---|---|
| | go vet | gosec | go vet | gosec | go vet | gosec |
| Any message | 219 | 36,279 | 76,738 | 40,678 | 31,224 | 114,306 |
| Related message | 213 | 26,267 | 76,744 | 18,019 | 0 | 0 |