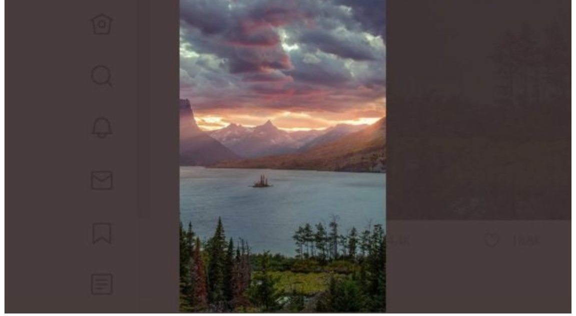
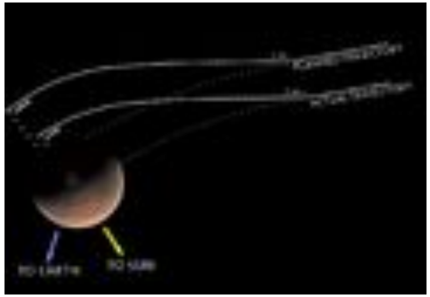
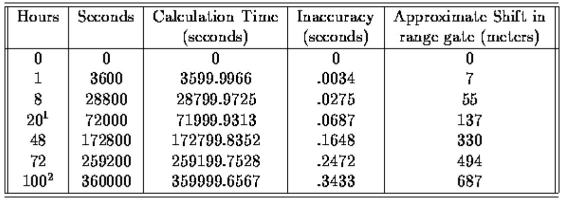
Verification for working developers

Martin Kellogg
University of Washington

Bugs in software





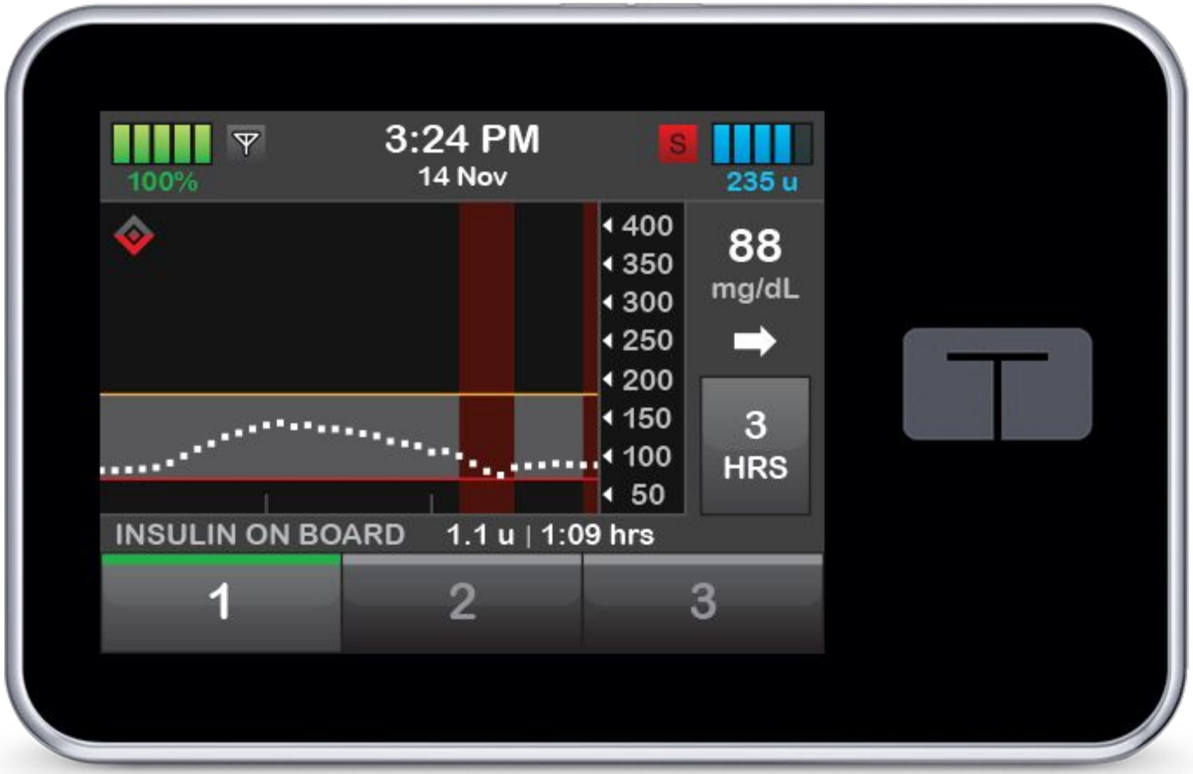


Hours	Seconds	Calculation Time (seconds)	Inaccuracy (seconds)	Approximate Shift in range gate (meters)
0	0	0	0	0
1	3600	3599.9966	.0034	7
8	28800	28799.9725	.0275	55
20[1]	72000	71999.9313	.0687	137
48	172800	172799.8352	.1648	330
72	259200	259199.7528	.2472	494
100[2]	360000	359999.6567	.3433	687

EQUIFAX

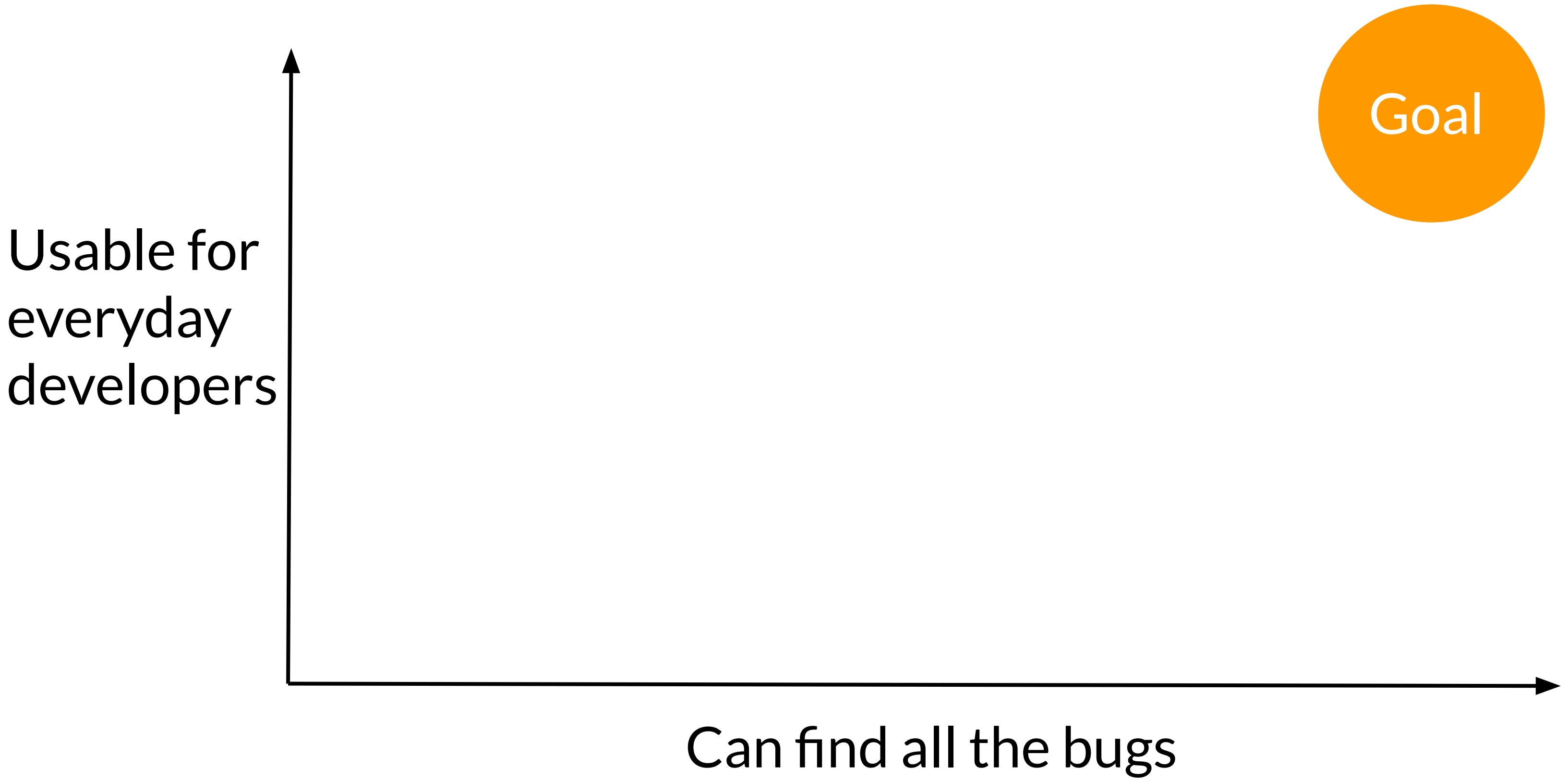
etc.

Bugs in software

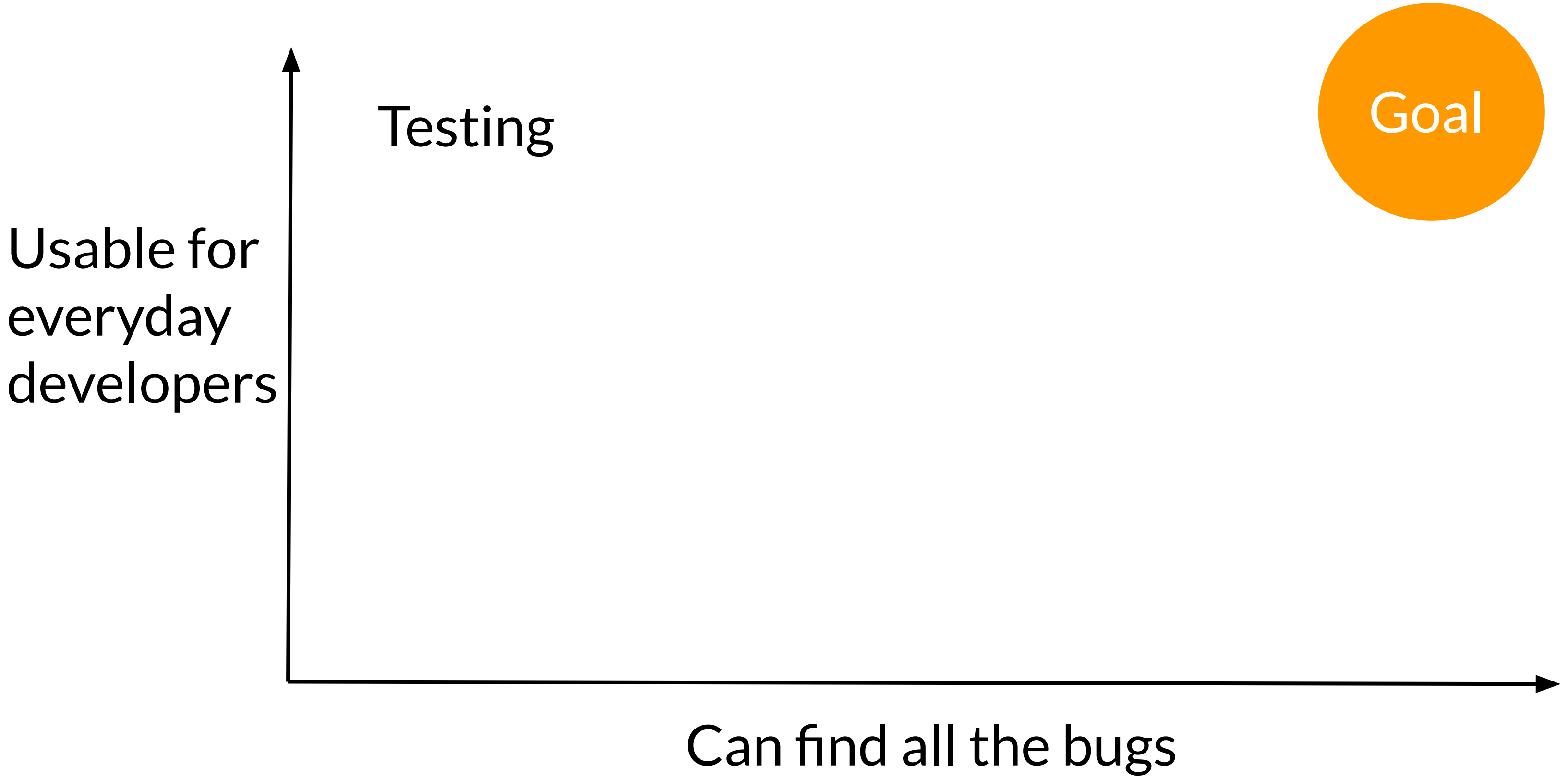


Goal: every developer uses verification

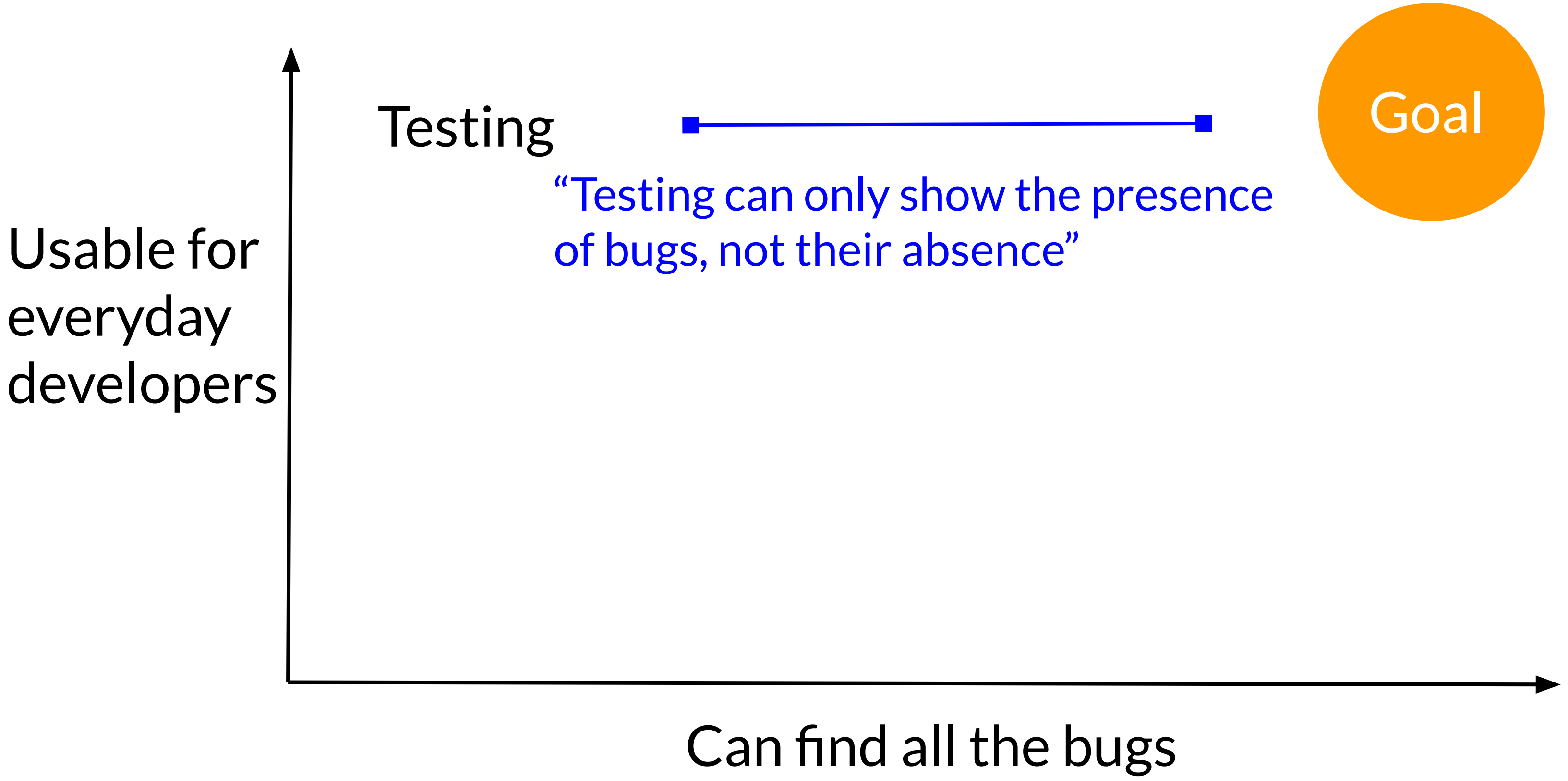
Preventing bugs: a gross oversimplification



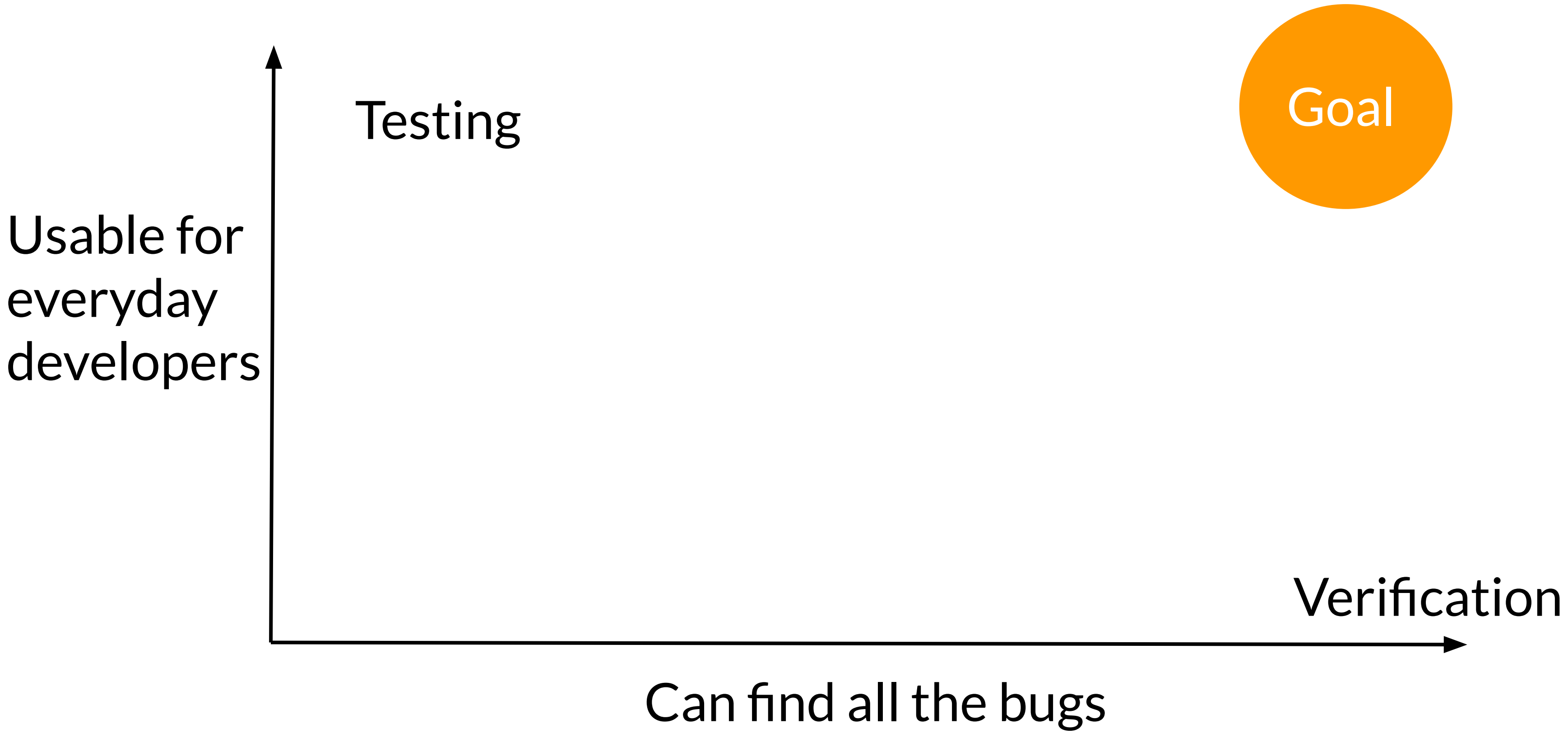
Preventing bugs: a gross oversimplification



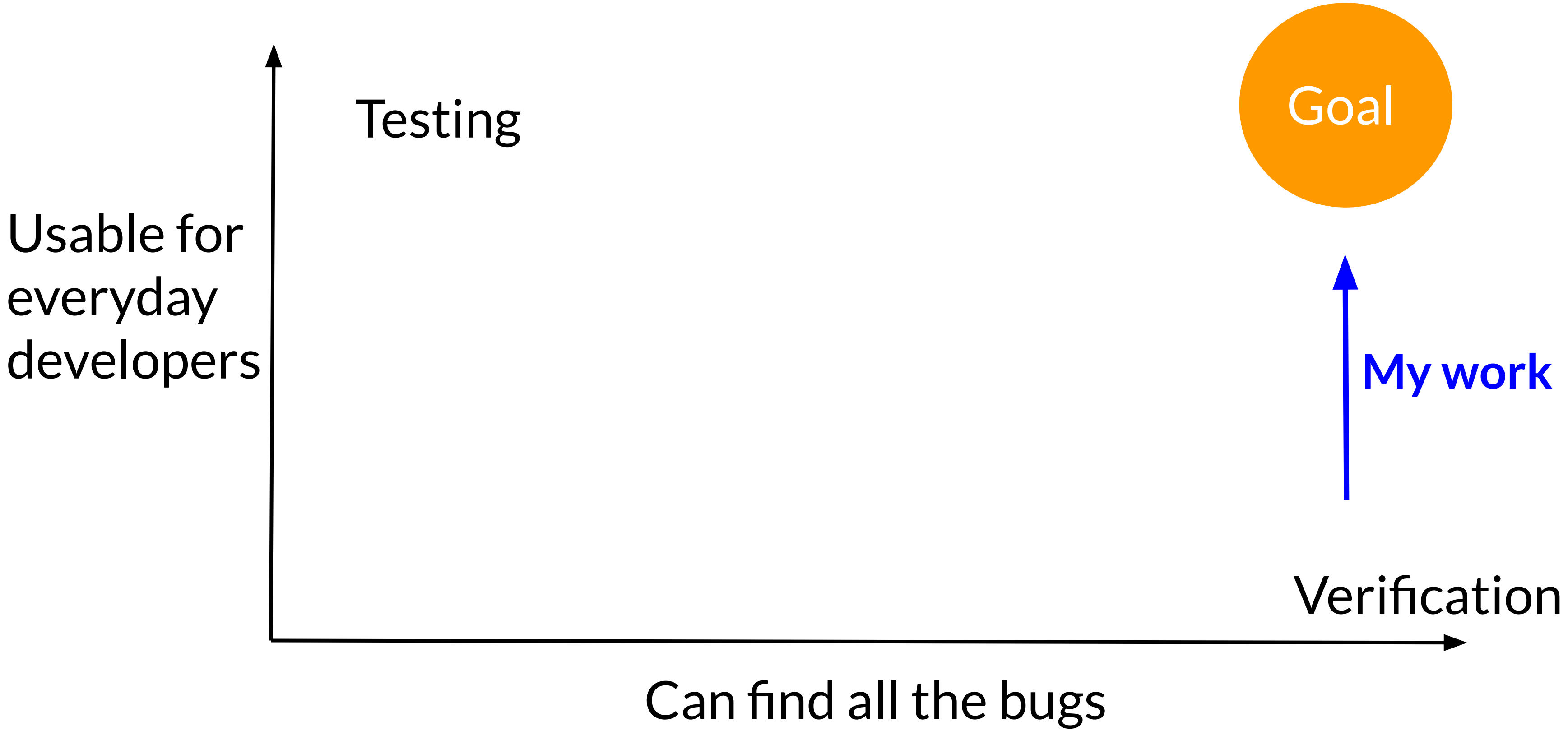
Preventing bugs: a gross oversimplification


Testing
“Testing can only show the presence of bugs, not their absence”
Goal
Usable for everyday developers
Can find all the bugs

Preventing bugs: a gross oversimplification



Preventing bugs: a gross oversimplification



Verification for working developers

Approach #1: make verification technologies ***more expressive***

Verification for working developers

Approach #1: make verification technologies ***more expressive***

“find clever ways to solve hard problems using simple techniques”

Verification for working developers

This talk: accumulation typestates

Approach #1: make verification technologies ***more expressive***

"find clever ways to solve hard problems using simple techniques"

Verification for working developers

Approach #1: make verification technologies ***more expressive***

“find clever ways to solve hard problems using simple techniques”

Approach #2: convince developers to use verification

Verification for working developers

Approach #1: make verification technologies ***more expressive***

"find clever ways to solve hard problems using simple techniques"

Approach #2: convince developers to use verification

- find new applications
- improve the usability

Verification for working developers

Approach #1: make verification technologies ***more expressive***

"find clever ways to solve hard problems using simple techniques"

Approach #2: convince developers to use verification

- find new applications **This talk: compliance**
- improve the usability

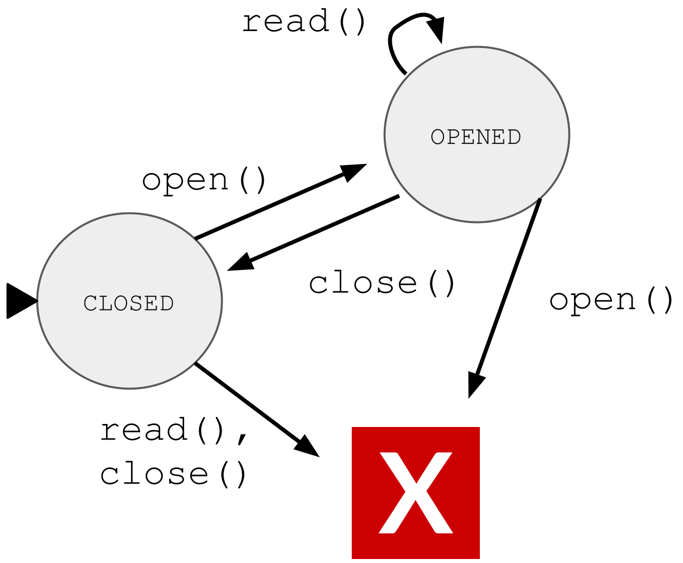
Talk outline

- **Expressivity**: accumulation typestate automata
  - theory: what is an accumulation typestate?
  - practice: is accumulation analysis useful?
- **Convincing developers**: compliance verification

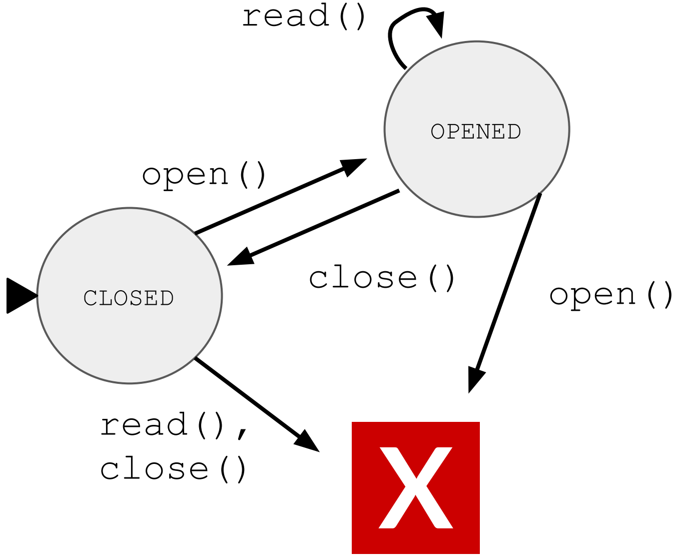
Typestate analysis

- Classic static program analysis technique
- First proposed by Strom & Yemeni (1986)
- Extensive literature: **over 18,000** hits on Google Scholar
- Sound typestate analysis is **expensive** due to aliasing

Typestate specification via FSM

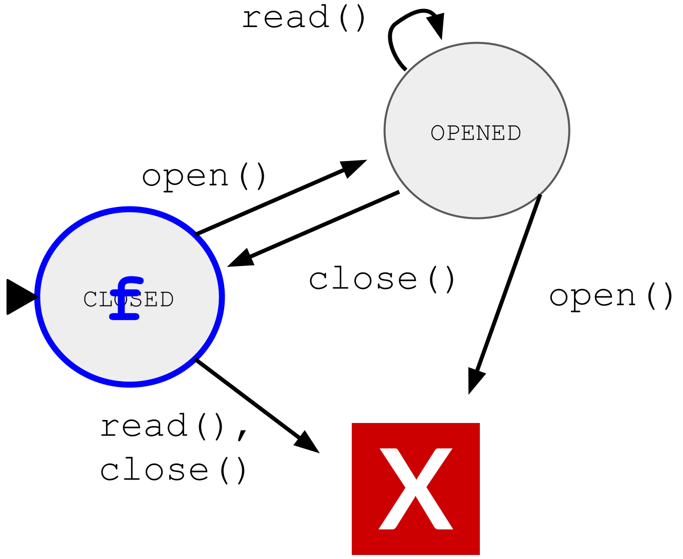


Typestate specification via FSM



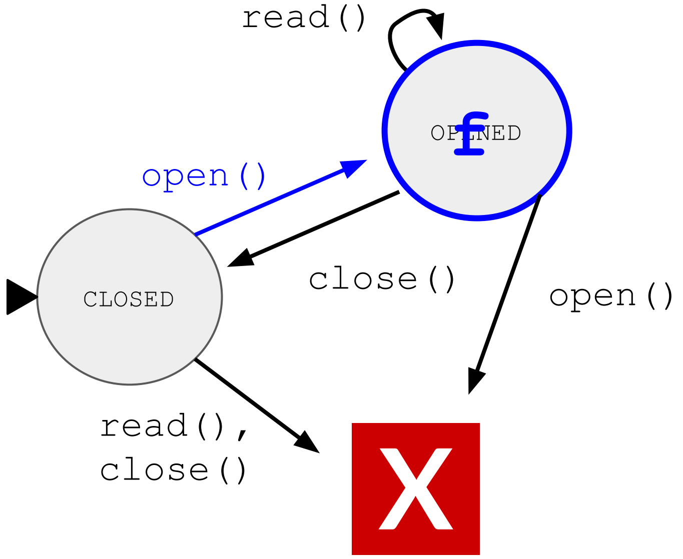
```
File f = …;
f.open();
f.close();
f.read();
```

Typestate specification via FSM



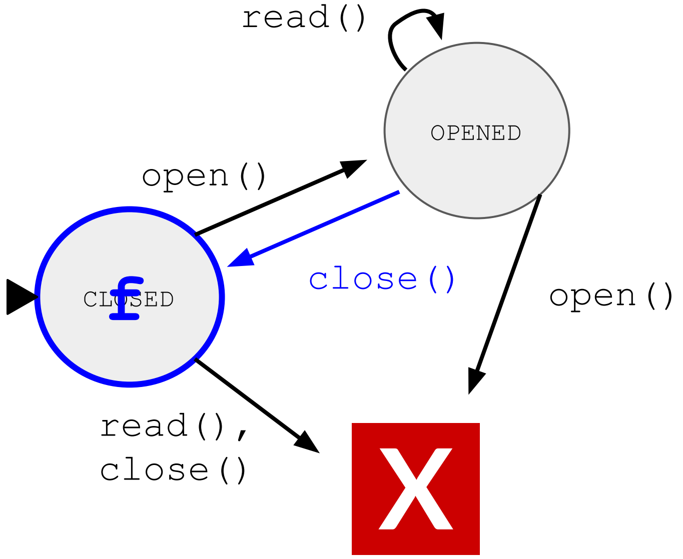
```
File f = …;
f.open();
f.close();
f.read();
```

Typestate specification via FSM



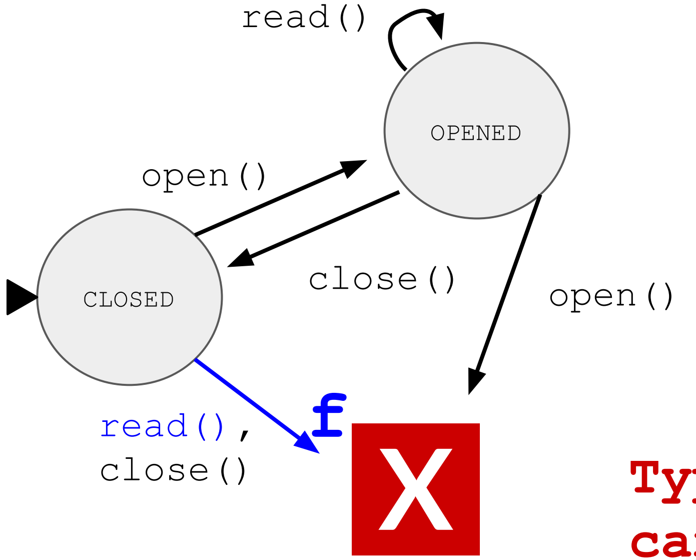
```
File f = …;
f.open();
f.close();
f.read();
```

Typestate specification via FSM



```
File f = …;
f.open();
f.close();
f.read();
```

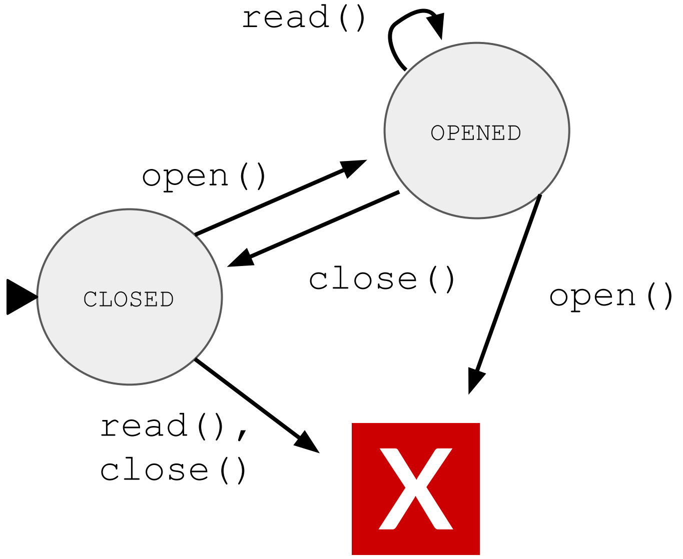
Typestate specification via FSM



```
File f = …;
f.open();
f.close();
f.read();
```

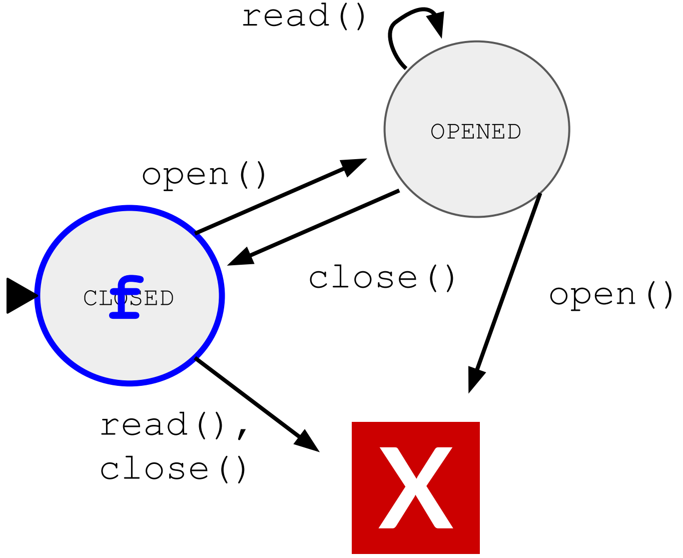
Typestate error: f cannot read() in state CLOSED

Why is typestate expensive?



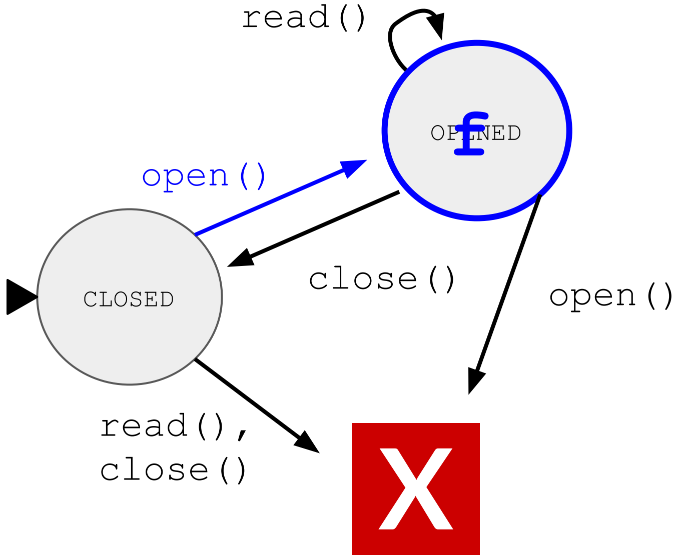
```
File f = …;
f.open();
File g = f;
f.close();
g.read();
```

Why is typestate expensive?



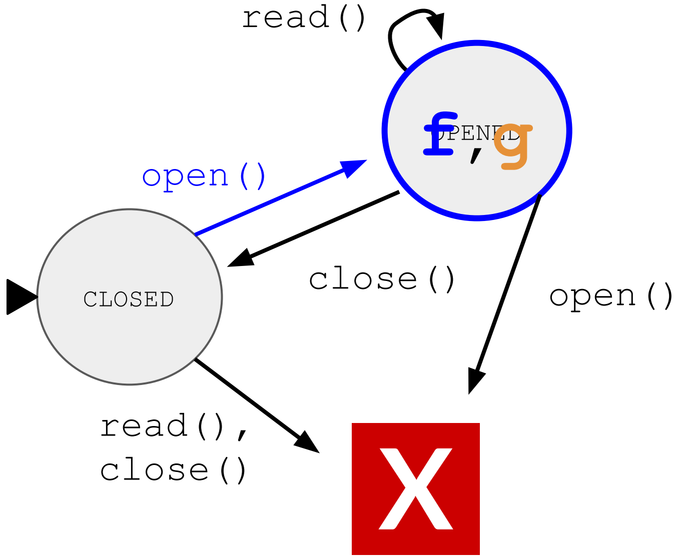
```
File f = …;
f.open();
File g = f;
f.close();
g.read();
```

Why is typestate expensive?



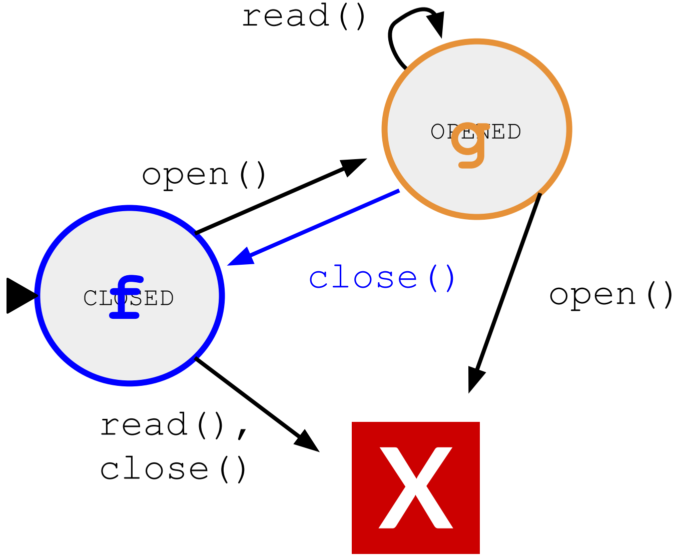
```
File f = …;
f.open();
File g = f;
f.close();
g.read();
```

Why is typestate expensive?



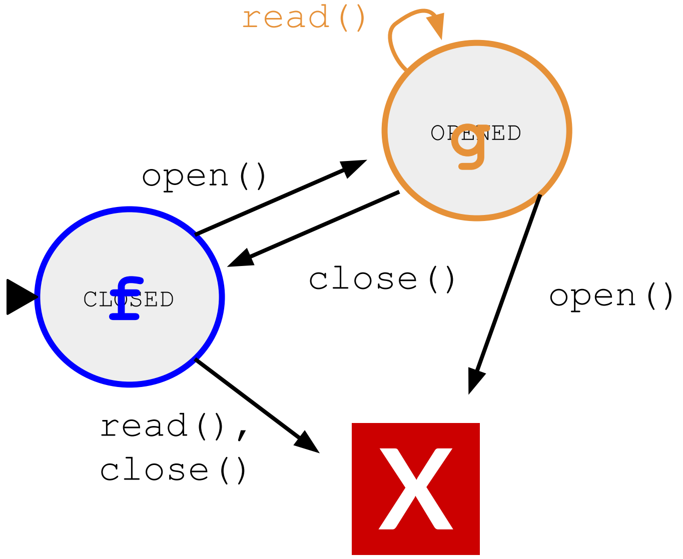
```
File f = …;
f.open();
File g = f;
f.close();
g.read();
```

Why is typestate expensive?



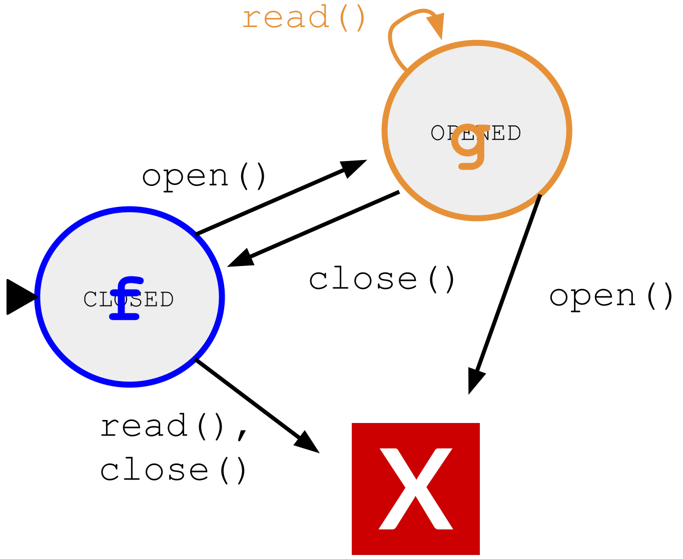
```
File f = …;
f.open();
File g = f;
f.close();
g.read();
```

Why is typestate expensive?



```
File f = …;
f.open();
File g = f;
f.close();
g.read();
```

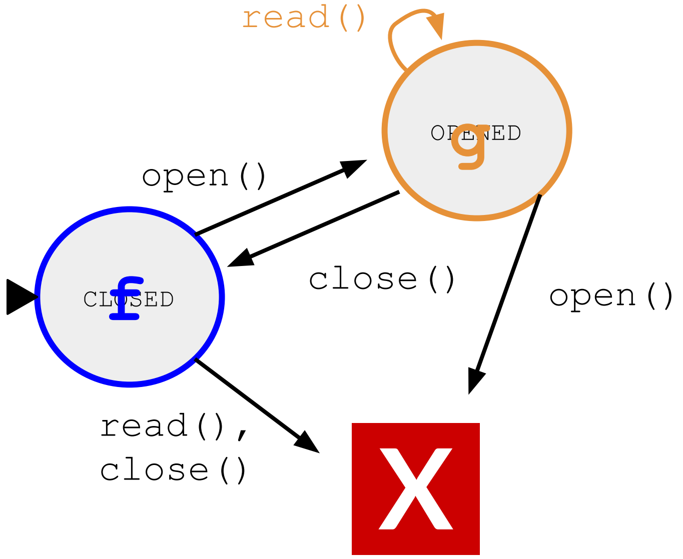
Why is typestate expensive? Aliasing.



```
File f = …;
f.open();
File g = f;
f.close();
g.read();
```

No error?

Why is typestate expensive? Aliasing.

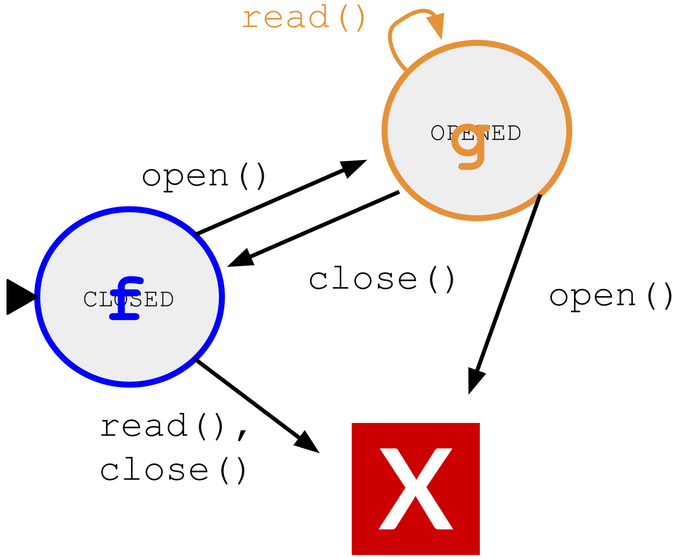


```
File f = …;
f.open();
File g = f;
f.close();
g.read();
```

No error?

“false negative”

Why is typestate expensive? Aliasing.



```
File f = …;
f.open();
File g = f;
f.close();
g.read();
```

No error?

“false negative”

Sound typestate requires aliasing information

- A **sound** typestate analysis must **track all aliases** to keep FSMs in sync

Sound typestate requires aliasing information

- A **sound** typestate analysis must **track all aliases** to keep FSMs in sync
- Three prior approaches:
  1. **ignore aliasing** and be unsound (e.g., Emmi et al. 2021)

Sound typestate requires aliasing information

- A **sound** typestate analysis must **track all aliases** to keep FSMs in sync
- Three prior approaches:
  1. **ignore aliasing** and be unsound (e.g., Emmi et al. 2021)
  2. **restrict aliasing** (e.g., via ownership types)

     ↳ e.g., Bierhoff et al. 2009, Clark et al. 2013, Rust

Sound typestate requires aliasing information

- A **sound** typestate analysis must **track all aliases** to keep FSMs in sync
- Three prior approaches:
  1. **ignore aliasing** and be unsound (e.g., Emmi et al. 2021)
  2. **restrict aliasing** (e.g., via ownership types)
  3. **whole-program** may-alias analysis (expensive)

     ↳ Tan et al. 2021 report hours for real programs

Sound typestate requires aliasing information

- A **sound** typestate analysis must **track all aliases** to keep FSMs in sync
- Three prior approaches:
  1. **ignore aliasing** and be unsound (e.g., Emmi et al. 2021)
  2. **restrict aliasing** (e.g., via ownership types)
  3. **whole-program** may-alias analysis (expensive)

Key question: does typestate analysis *always* need aliasing information?

Insight: aliasing information is only required for some typestate automata

Insight: aliasing information is only required for some typestate automata

Which ones?

Accumulation typestates

accumulation typestate automaton:

for any **error-inducing sequence** $S = t_1, \ldots, t_i$,

all **subsequences** of S that end in t_i

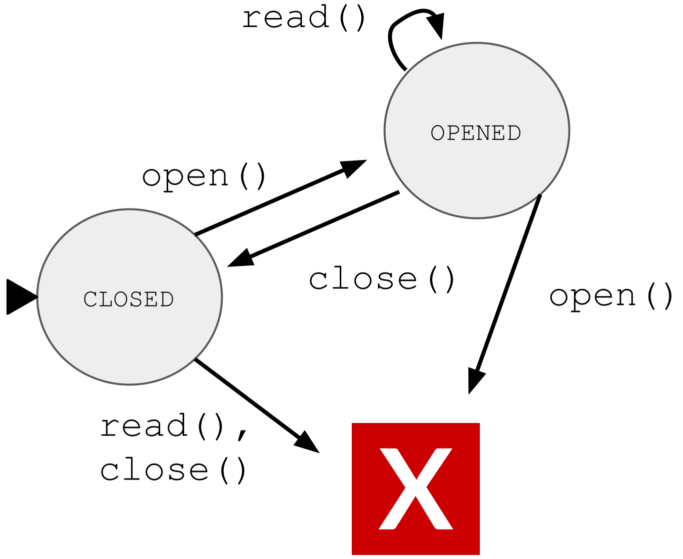
are also **error-inducing**

Accumulation typestates

accumulation typestate automaton:
for any **error-inducing sequence** $S = t_1, \ldots, t_i$,
all **subsequences** of S that end in t_i
are also **error-inducing**

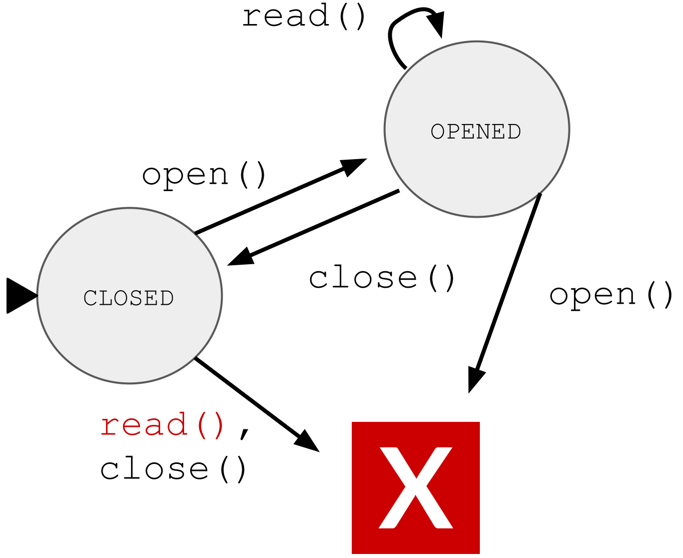
Key theorem: Accumulation typestates are **exactly** those that can be checked soundly **without aliasing information**

Is it an accumulation typestate automaton?



for any **error-inducing sequence** $S = t_1, \ldots, t_i$,
all **subsequences** of S that end in t_i
are also **error-inducing**

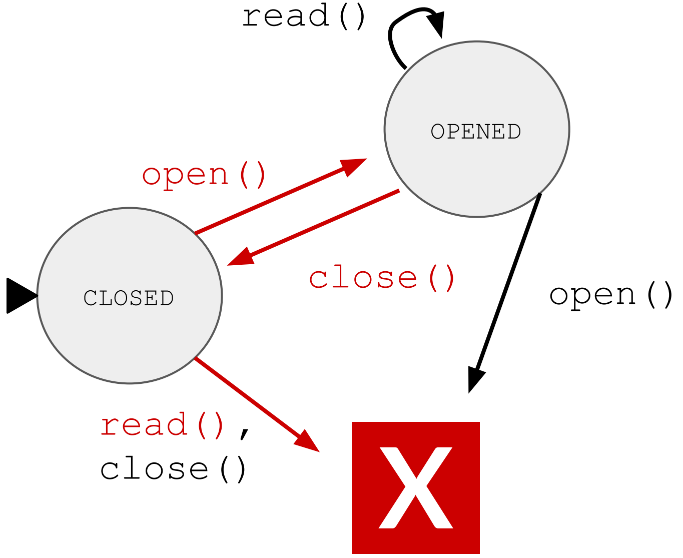
Is it an accumulation typestate automaton?



for any **error-inducing sequence** *S* = $t_1, \ldots, t_i$,
all **subsequences** of *S* that end in t_i
are also **error-inducing**

S = `read()`

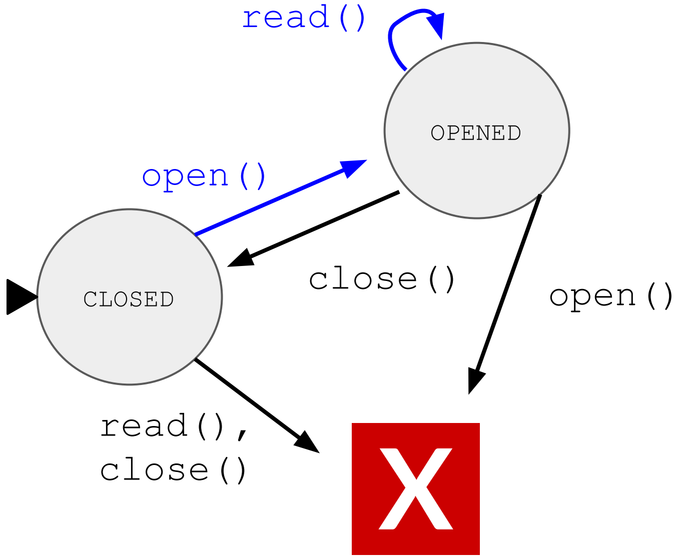
Is it an accumulation typestate automaton?



for any **error-inducing sequence** *S* = $t_1, \ldots, t_i$,
all **subsequences** of *S* that end in t_i
are also **error-inducing**

S = `open(),close(),read().`

Is it an accumulation typestate automaton?



for any **error-inducing sequence** $S = t_1, \ldots, t_i$,
all **subsequences** of S that end in t_i
are also **error-inducing**

No!

S = `open(), close(), read().`

S' = `open(),` ~~`close()`~~, `read()`
is not error-inducing!

Is it an accumulation typestate automaton?

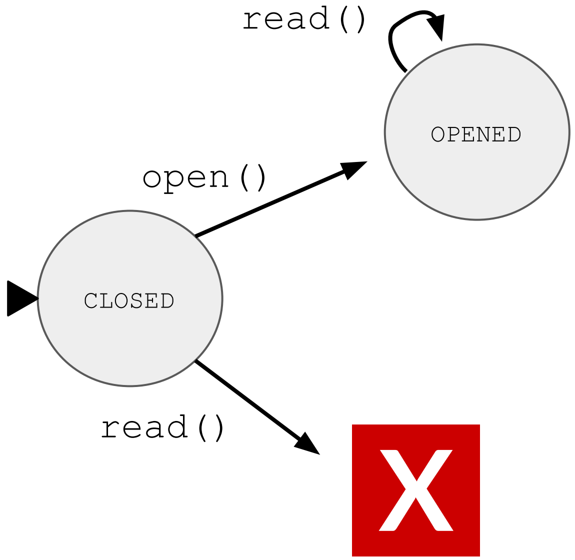
“only call `read()` after calling `open()` at least once”

for any **error-inducing sequence** $S = t_1, \ldots, t_i$,
all **subsequences** of S that end in t_i
are also **error-inducing**

Is it an accumulation typestate automaton?

"only call `read()` after calling `open()` at least once"

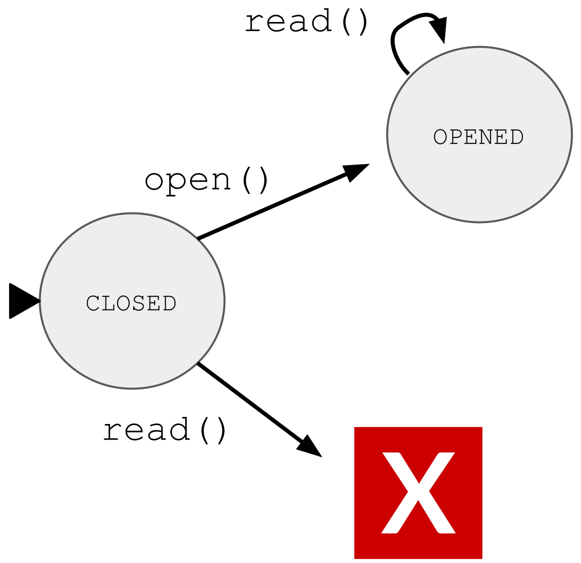
for any **error-inducing sequence** $S = t_1, \ldots, t_i$,
all **subsequences** of S that end in t_i
are also **error-inducing**



Is it an accumulation typestate automaton?

"only call `read()` after calling `open()` at least once"

for any **error-inducing sequence** $S = t_1, \ldots, t_i$, all **subsequences** of S that end in t_i are also **error-inducing**



Yes!

Aside: how hard is it to decide if a typestate automaton is accumulation?

Aside: how hard is it to decide if a typestate automaton is accumulation?

- As easy as checking DFA equivalence
  - Result due to **Higman's Theorem** (1952)

Aside: how hard is it to decide if a typestate automaton is accumulation?

- As easy as checking DFA equivalence
  - Result due to **Higman's Theorem** (1952)

"The subsequence language of *any language whatsoever* over a finite alphabet is regular."

Accumulation typestates

accumulation typestate automaton:

for any **error-inducing sequence** $S = t_1, \ldots, t_i$,

all **subsequences** of S that end in t_i

are also **error-inducing**

Key theorem: Accumulation typestates are **exactly** those that can be checked soundly **without aliasing information**

Accumulation typestates

accumulation typestate automaton:

for any **error-inducing sequence** $S = t_1, \ldots, t_i$,

all **subsequences** of S that end in t_i

are also **error-inducing**

Key theorem: Accumulation typestates are **exactly** those that can be checked soundly **without aliasing information**

Proof intuition

Key theorem: Accumulation typestates are **exactly** those that can be checked soundly **without aliasing information**

Proof intuition

Key theorem: Accumulation typestates are **exactly** those that can be checked soundly **without aliasing information**

Intuition for ⇒:

1. without aliasing information, analysis **observes a subsequence** of actual transitions

Proof intuition

Key theorem: Accumulation typestates are **exactly** those that can be checked soundly **without aliasing information**

Intuition for ⇒:

1. without aliasing information, analysis **observes a subsequence** of actual transitions
2. if analysis observes a transition that leads to an error at run time, the final transition **must be error-inducing**

Proof intuition

Key theorem: Accumulation typestates are exactly those that can be checked soundly without aliasing information

for any **error-inducing sequence** $S = t_1, \ldots, t_i$,
all **subsequences** of S that end in t_i
are also **error-inducing**

Intuition for ⇒:

1. without aliasing information, analysis **observes a subsequence** of actual transitions
2. if analysis observes a transition that leads to an error at run time, the final transition **must be error-inducing**

A brief review

- An **accumulation typestate automaton** is **closed under error-inducing subsequences** with the same error-inducing transition
- Accumulation typestate automata are **exactly** those that can be checked **without aliasing information**
- Higman’s theorem is cool

Measuring success

Goal: every developer uses verification

Measuring success

Goal: every developer uses verification

"Are the resulting analyses useful & usable for developers?"

Measuring success

Goal: every developer uses verification

"Are the resulting analyses useful & usable for developers?"

Implementation **Evaluation**

Implementation: accumulation analysis

- Directly tracks the **sequence of transitions** each variable has observed rather than the FSM
- **Modular**: can analyze each method independently
- Can be implemented as a **type system**, abstract interpretation, dataflow analysis, etc.

Implementation: aliasing

- Accumulation is **sound** without aliasing information
- But it might not be **precise**: false positives

Implementation: aliasing

- Accumulation is **sound** without aliasing information
- But it might not be **precise**: false positives

→ **Prune false positives using cheap, local alias analysis**

Accumulation analysis: example

"Before using an object of type *T*, set the *f* and *g* fields."

Accumulation analysis: example

"Before using an object of type *T*, set the *f* and *g* fields."

```
T t = …;
t.f = …;
t.g = …;

use(t);
```

Accumulation analysis: example

"Before using an object of type *T*, set the *f* and *g* fields."

```
T t = …; ◀
t.f = …;
t.g = …;

use(t);
```

Initialized Fields

[ ]

Accumulation analysis: example

“Before using an object of type *T*, set the *f* and *g* fields.”

```
T t = …;
t.f = …; ◀
t.g = …;

use(t);
```

Initialized Fields

```
[     ]
[  f  ]
```

Accumulation analysis: example

"Before using an object of type *T*, set the *f* and *g* fields."

```
T t = …;
t.f = …;
t.g = …; ◀

use(t);
```

Initialized Fields

```
[       ]
[   f   ]
[  f,g  ]
```

Accumulation analysis: example

"Before using an object of type *T*, set the *f* and *g* fields."

```
T t = …;
t.f = …;
t.g = …; ◀

use(t);
```

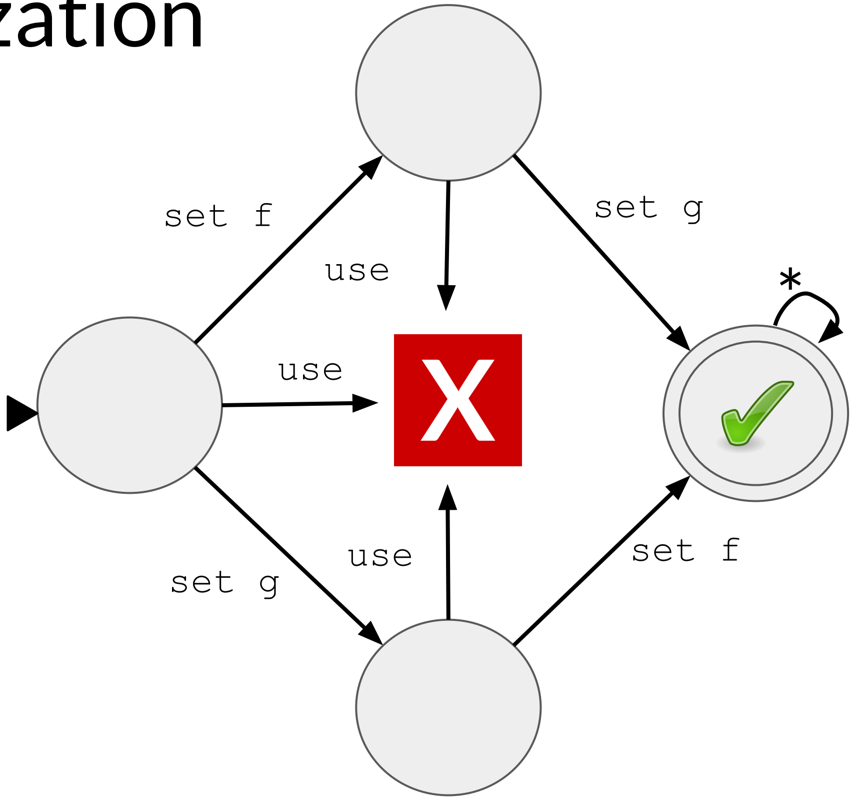
Initialized Fields

```
[       ]
[  f    ]
[  f,g  ]
```

✓

Accumulation for initialization

"Before using an object of type *T*, set the *f* and *g* fields."



Accumulation for initialization

"Before using an object of type *T*, set the *f* and *g* fields."

```
T t = …;
t.f = …;
t.g = …; ◀

use(t);
```

Initialized Fields

```
[       ]
[  f    ]
[  f,g  ]
```

✓

Accumulation: evaluation overview

- Initialization (ICSE 2020)
    - **User study** with real engineers
    - Detection & prevention of machine-image sniping **security vulnerabilities**
- Detection & prevention of **resource leaks** (ESEC/FSE 2021)

Accumulation for initialization: user study

Task: add a new required field to a builder

  Control: existing tests only

  Treatment: accumulation analysis + existing tests

Design: factorial with 2 tasks/subject, randomized order and condition

Subjects: 6 professional software engineers

Accumulation for initialization: user study

Task: add a new required field to a builder

- **Control**: existing tests only
- **Treatment**: accumulation analysis + existing tests

Design: factorial with 2 tasks/subject, randomized order and condition

Subjects: 6 professional software engineers

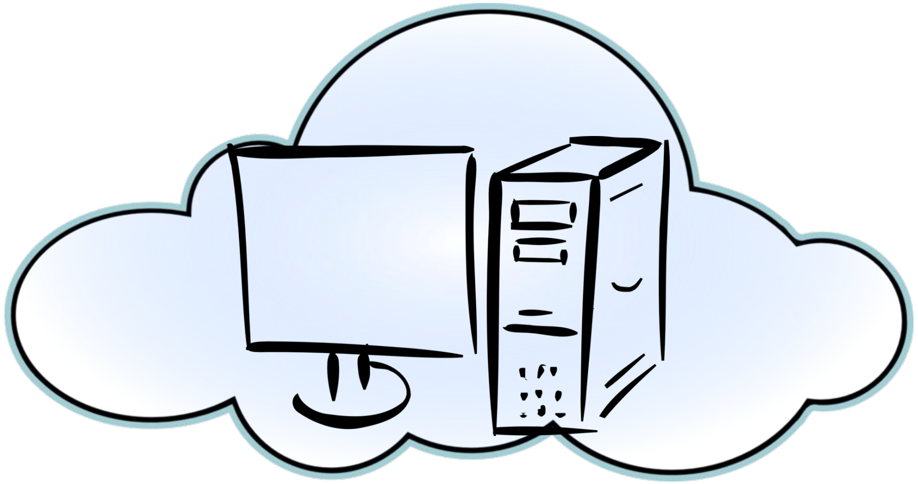
Results:

- +50% **success rate**
- ~50% **faster**

Accumulation for initialization: security

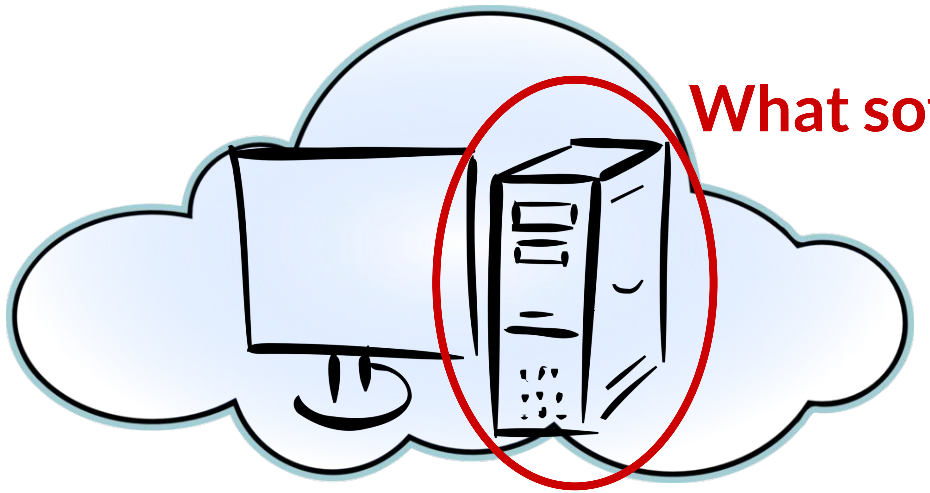
- Security vulnerabilities: **machine image sniping**

What is a machine image?

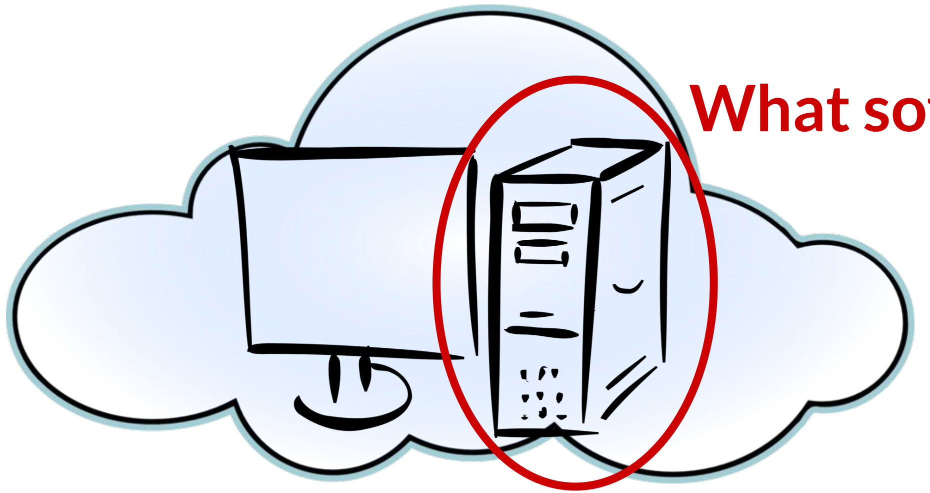


cloud computer

What is a machine image?



What is a machine image?



What software to run?

cloud computer

“machine image”

How to choose a machine image:

Look it up in a repository.

- By unique id:

```
aws ec2 describe-images --imageIds ami-5731123e
```

- By owner and name:

```
aws ec2 describe-images --owners myOrg \
  --filters "Name=myName,Values=ubuntu16.04-*"
```

- By name alone:

```
aws ec2 describe-images \
  --filters "Name=myName,Values=ubuntu16.04-*"
```

How to choose a machine image:

Look it up in a repository.

- By **unique id**:

```
aws ec2 describe-images --imageIds ami-5731123e
```

✓

- By owner and name:

```
aws ec2 describe-images --owners myOrg \
  --filters "Name=myName,Values=ubuntu16.04-*"
```

- By name alone:

```
aws ec2 describe-images \
  --filters "Name=myName,Values=ubuntu16.04-*"
```

How to choose a machine image:

Look it up in a repository.

- By **unique id**:

  ```
  aws ec2 describe-images --imageIds ami-5731123e
  ```

- By **owner** and **name**:

  ```
  aws ec2 describe-images --owners myOrg \
    --filters "Name=myName,Values=ubuntu16.04-*"
  ```

- By name alone:

  ```
  aws ec2 describe-images \
   --filters "Name=myName,Values=ubuntu16.04-*"
  ```

How to choose a machine image:

Look it up in a repository.

- By **unique id**:

```
aws ec2 describe-images --imageIds ami-5731123e
```

✓

- By **owner** and **name**:

```
aws ec2 describe-images --owners myOrg \
  --filters "Name=myName,Values=ubuntu16.04-*"
```

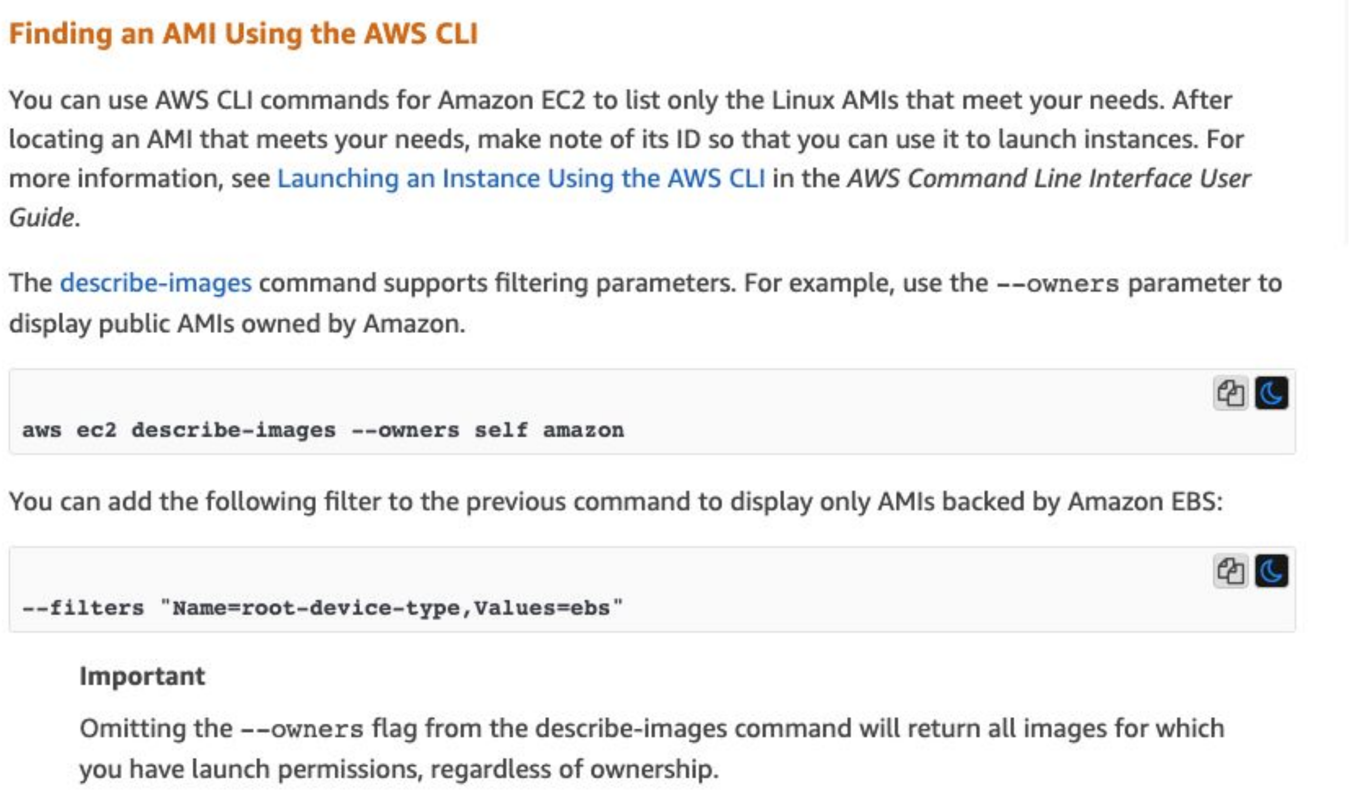
✓

- By **name alone**:

```
aws ec2 describe-images \
  --filters "Name=myName,Values=ubuntu16.04-*"
```

X

Unsafe client



Finding an AMI Using the AWS CLI

You can use AWS CLI commands for Amazon EC2 to list only the Linux AMIs that meet your needs. After locating an AMI that meets your needs, make note of its ID so that you can use it to launch instances. For more information, see Launching an Instance Using the AWS CLI in the *AWS Command Line Interface User Guide*.

The describe-images command supports filtering parameters. For example, use the `--owners` parameter to display public AMIs owned by Amazon.

```
aws ec2 describe-images --owners self amazon
```

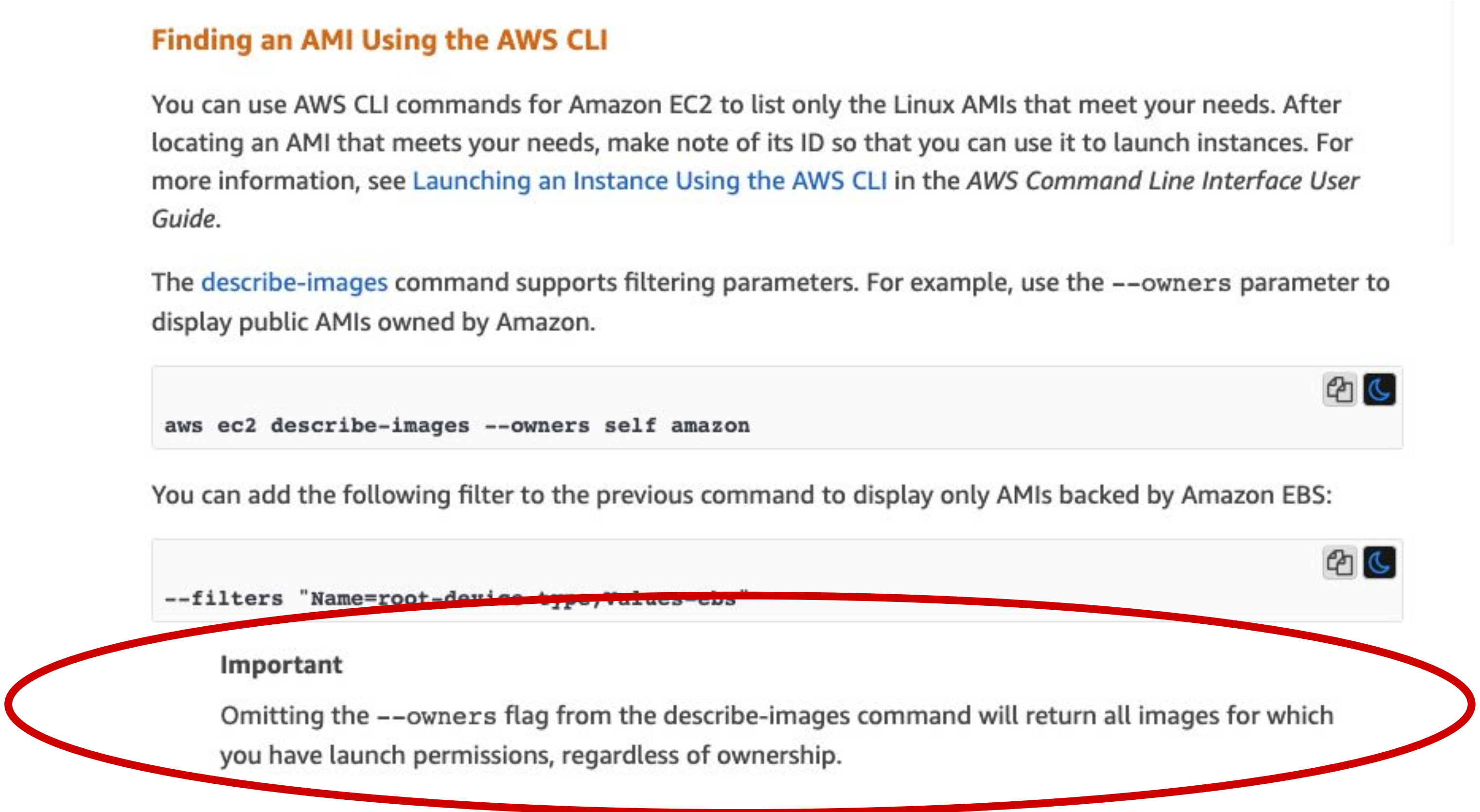
You can add the following filter to the previous command to display only AMIs backed by Amazon EBS:

```
--filters "Name=root-device-type,Values=ebs"
```

Important

Omitting the `--owners` flag from the describe-images command will return all images for which you have launch permissions, regardless of ownership.

Unsafe client



Finding an AMI Using the AWS CLI

You can use AWS CLI commands for Amazon EC2 to list only the Linux AMIs that meet your needs. After locating an AMI that meets your needs, make note of its ID so that you can use it to launch instances. For more information, see Launching an Instance Using the AWS CLI in the *AWS Command Line Interface User Guide*.

The describe-images command supports filtering parameters. For example, use the `--owners` parameter to display public AMIs owned by Amazon.

```
aws ec2 describe-images --owners self amazon
```

You can add the following filter to the previous command to display only AMIs backed by Amazon EBS:

```
--filters "Name=root-device-type,Values=ebs"
```

Important

Omitting the `--owners` flag from the describe-images command will return all images for which you have launch permissions, regardless of ownership.

Unsafe client

```
DescribeImagesRequest request = new DescribeImagesRequest();
request.withFilters(new Filter("myName", "RHEL-7.5_HVM_GA"));

api.describeImages(request);
```

Unsafe client

```
DescribeImagesRequest request = new DescribeImagesRequest();
request.withFilters(new Filter("myName", "RHEL-7.5_HVM_GA"));

api.describeImages(request);
```

Unsafe: returns all images with that name from public repo!

How to make this client safe?

```
DescribeImagesRequest request = new DescribeImagesRequest();
request.withFilters(new Filter("myName", "RHEL-7.5_HVM_GA"));

api.describeImages(request);
```

How to make this client safe?

```
DescribeImagesRequest request = new DescribeImagesRequest();
request.withFilters(new Filter("myName", "RHEL-7.5_HVM_GA"));
request.withOwners(“myOrg”);
api.describeImages(request);
```

Requirement: call `withOwners()` or `withImageIds()` before calling `describeImages()`

Experimental results

No. projects	545
Source LoC	~9.1M
True positives	16
False positives	3
Annotations	34

Experimental results

Non-comment, non-blank

No. projects	545
Source LoC	~9.1M
True positives	16
False positives	3
Annotations	34

Experimental results

No. projects	545
Source LoC	~9.1M
True positives	16
False positives	3
Annotations	34

Real RCE vulnerabilities

Example: Netflix/SimianArmy

```java
public List<Image> describeImages(String... imageIds) {
    DescribeImagesRequest request =
              new DescribeImagesRequest();

    if (imageIds != null) {
        request.setImageIds(Arrays.asList(imageIds));
    }

    DescribeImagesResult result =
              ec2client.describeImages(request);

    return result.getImages();
}
```

Accumulation: evaluation overview

- Initialization (ICSE 2020)
  - **User study** with real engineers
  - Detection & prevention of machine-image sniping **security vulnerabilities**
- Detection & prevention of **resource leaks** (ESEC/FSE 2021)

Accumulation for resource leaks

```
try {
    Socket s = new Socket(address, port);
    ...
    s.close();
} catch (IOException e) {


}
```

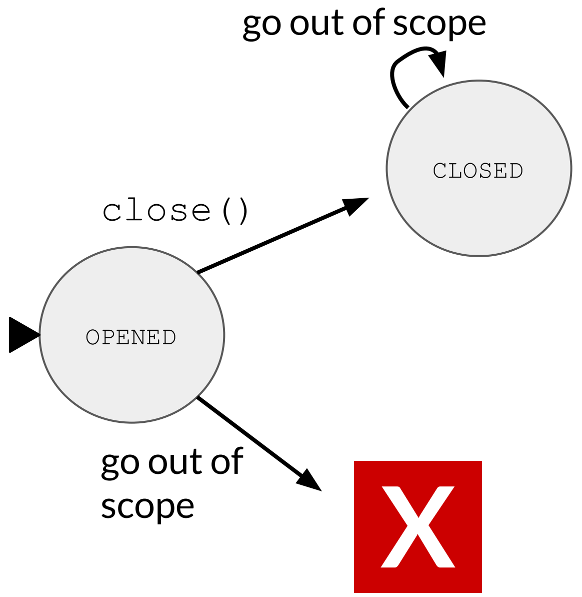
Accumulation for resource leaks

```
try {
    Socket s = new Socket(address, port);
    ...
    ~~s.close();~~
} catch (IOException e) {


}
```

Missing call to close()

Accumulation for resource leaks

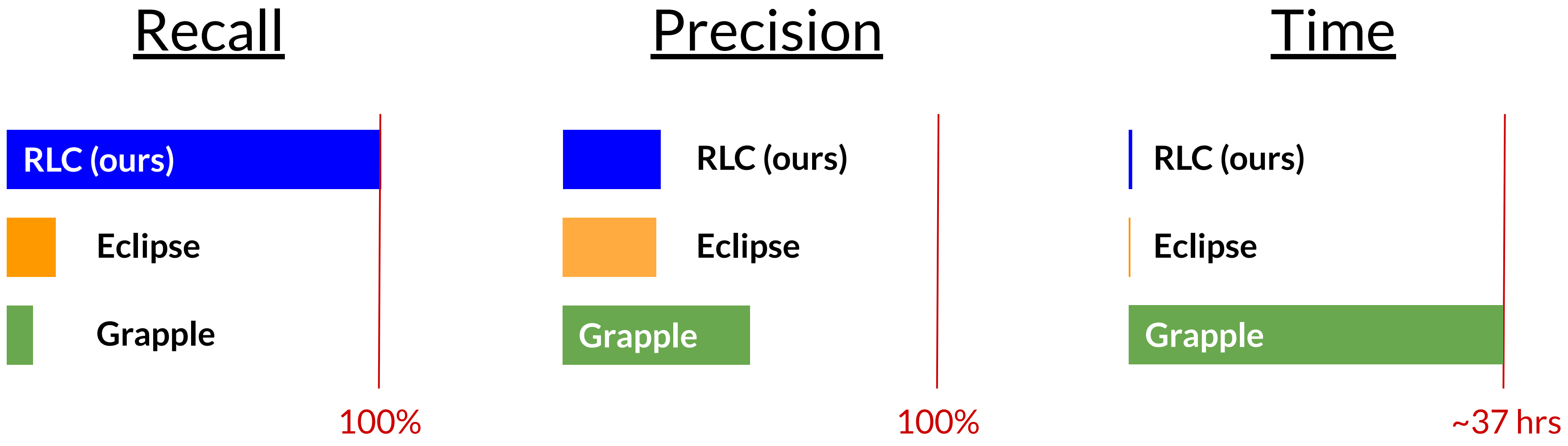


Accumulation for resource leaks

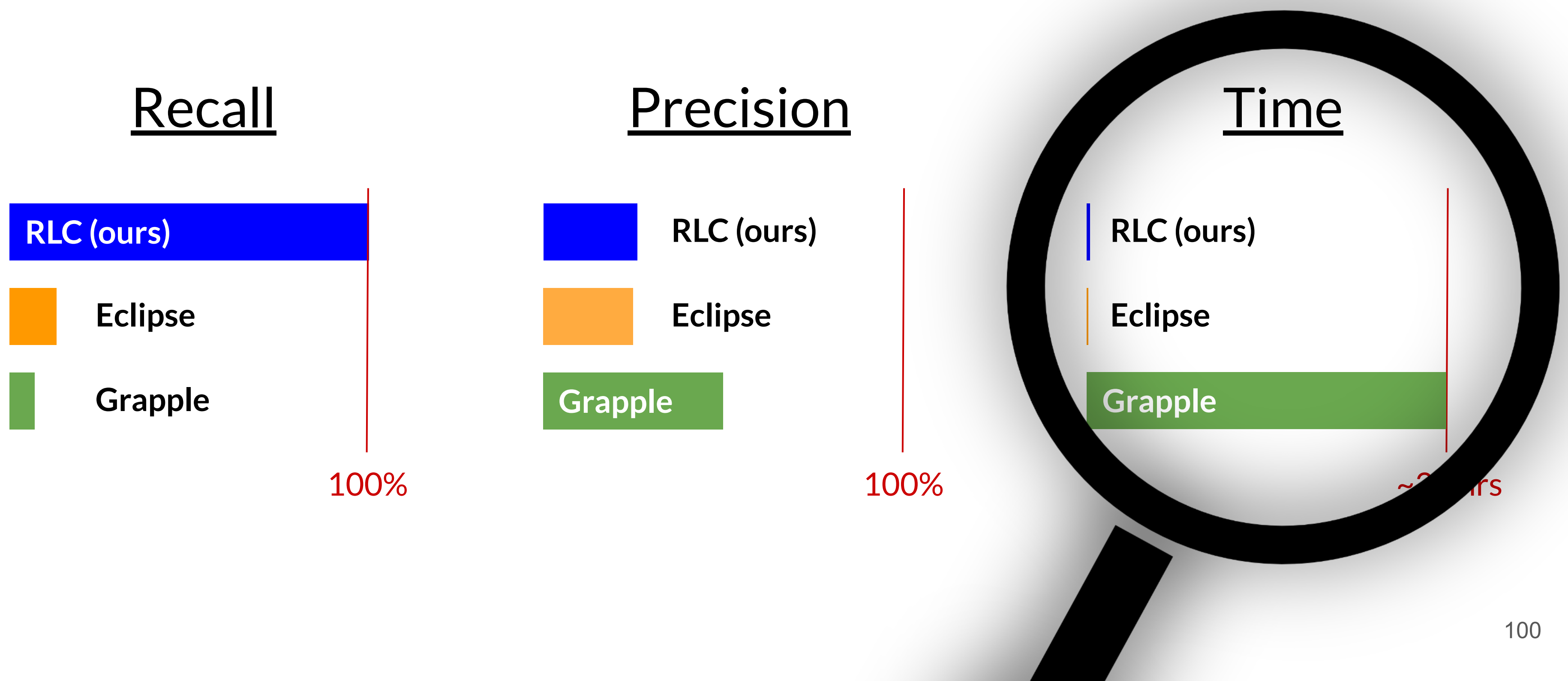
3-stage checker:

1. taint-tracker over-approximates methods that **need to be called**
2. accumulation under-approximates methods that **have been called**
3. dataflow analysis **compares** the two at “going out-of-scope” points

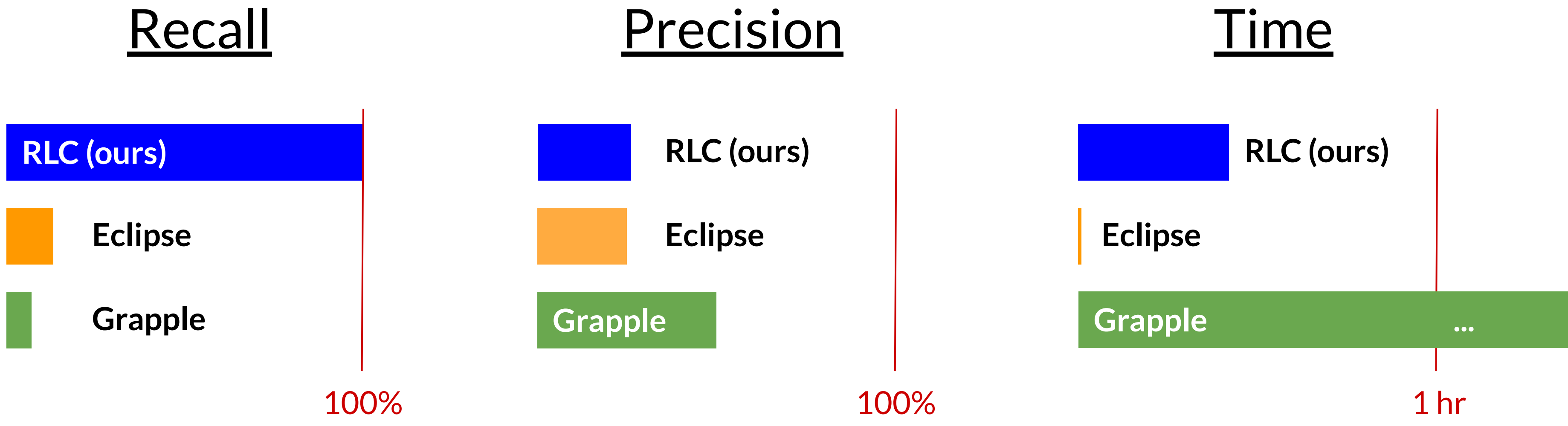
Accumulation for resource leaks: results



Accumulation for resource leaks: results



Accumulation for resource leaks: results



Accumulation summary

- **Accumulation typestate automata** are exactly those that can be checked **without aliasing information**
- Accumulation typestate automata include **important problems** like resource leaks, security vulnerabilities, and initialization
- For accumulation typestate problems, an accumulation analysis is **sound, precise, and fast**

Other projects

- **Array bounds** checking without SMT (ISSTA 2018)
- Other **verifiers deployed** at AWS
- **Push-button** verification via type inference
- Replacing manual **compliance** with verification (ASE 2020)

Other projects

- **Array bounds** checking without SMT (ISSTA 2018)
- Other **verifiers deployed** at AWS
- **Push-button** verification via type inference
- Replacing manual **compliance** with verification (ASE 2020)

Replacing compliance checks with verification

- Certificates that a company follows a ruleset
  - PCI DSS for credit card transactions
  - HIPAA for healthcare information
  - FedRAMP for US government cloud vendors
  - SOC for information security vendors
  - etc.

Replacing compliance checks with verification

- Certificates that a company follows a ruleset
  - PCI DSS for credit card transactions
  - HIPAA for healthcare information
  - FedRAMP for US government cloud vendors
  - SOC for information security vendors
  - etc.
- State-of-the-practice is **manual audits** of source code

Replacing compliance checks with verification

- Certificates that a company follows a ruleset
  - PCI DSS for credit card transactions
  - HIPAA for healthcare information
  - FedRAMP for US government cloud vendors
  - SOC for information security vendors
  - etc.
- State-of-the-practice is manual audits of source code

Developers hate doing this work

Replacing compliance checks with verification

- Certificates that a company follows a ruleset
    - PCI DSS for credit card transactions
    - HIPAA for healthcare information
    - FedRAMP for US government cloud vendors
    - SOC for information security vendors
    - etc.
- State-of-the-practice is **manual audits** of source code
- **Insight:** specialized checkers can replace manual audits

Replacing compliance checks with verification

- Certificates that a company follows a ruleset
  - PCI DSS for credit card transactions
  - HIPAA for healthcare information
  - FedRAMP for US government cloud vendors
  - SOC for information security vendors
  - etc.
- State-of-the-practice is manual audits of source code
- **Insight:** specialized checkers can replace manual audits

$\hookrightarrow$ **Developers love this, because it saves work**
Auditors love this, because it reduces human error

Specialized compliance checkers, industry

Run on ~76,000,000 NCNB LoC

Verified	37,315 pkgs
True pos.	173 pkgs
False pos.	1 pkg

Specialized compliance checkers, industry

Only 23 handwritten annotations

Run on ~76,000,000 NCNB LoC

Verified	37,315 pkgs
True pos.	173 pkgs
False pos.	1 pkg

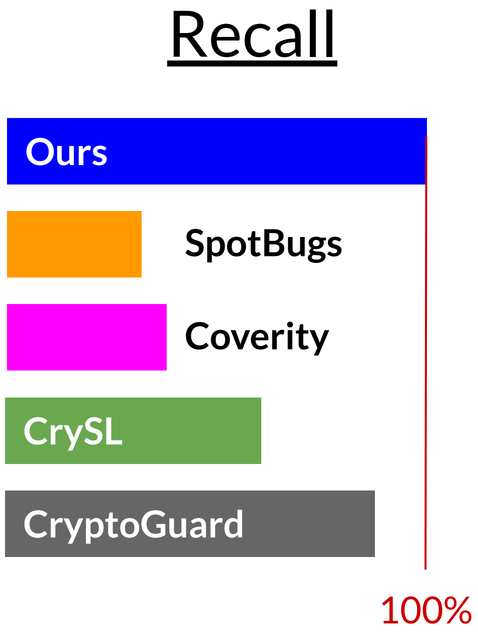
Specialized compliance checkers, industry

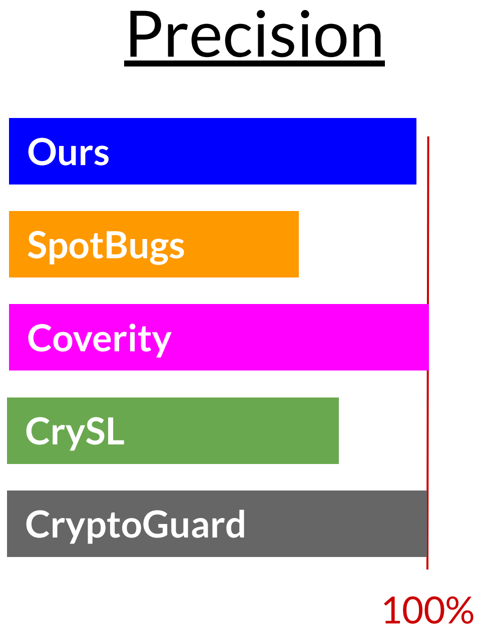
Run on ~76,000,000 NCNB LoC

Verified	37,315 pkgs
True pos.	173 pkgs
False pos.	1 pkg

- Auditors accepted output of checkers as evidence during a **real audit**
- Checkers **integrated** into build process

Our checkers vs. other approaches





Future work: short-term plans

- **accumulation**: **41% of typestates** in the scientific literature since 1999 are accumulation
    - e.g., authorization, connect sockets before send, etc.
    - improved accumulation analysis algorithms

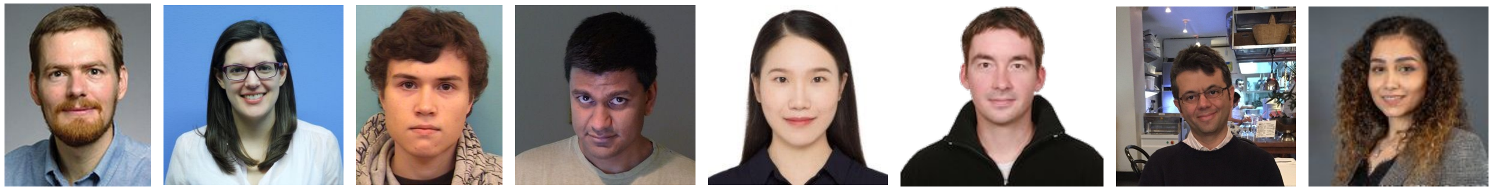
Future work: short-term plans

- **accumulation**: **41% of typestates** in the scientific literature since 1999 are accumulation
  - e.g., authorization, connect sockets before send, etc.
  - improved accumulation analysis algorithms
- **compliance verification**
  - collaborate with management science or operations research and with industry

Future work: long-term vision

- **Verification-by-parts**: split apart the codebase by **commits** rather than by files, classes, methods, etc.
- **Push-button verification**: use specification inference techniques to verify simple properties automatically
- **Continued industrial collaboration** to find good problems to work on

Thanks to my fantastic collaborators!



...

Summary

- **My goal: verification** for **working developers**
- **My approach:** design and build verification systems **that developers can use**
    - **expressivity**: accumulation makes it easier to verify initialization, resource leaks, etc.
    - **convince**: compliance shows how verification can fit into an everyday developer’s workflow