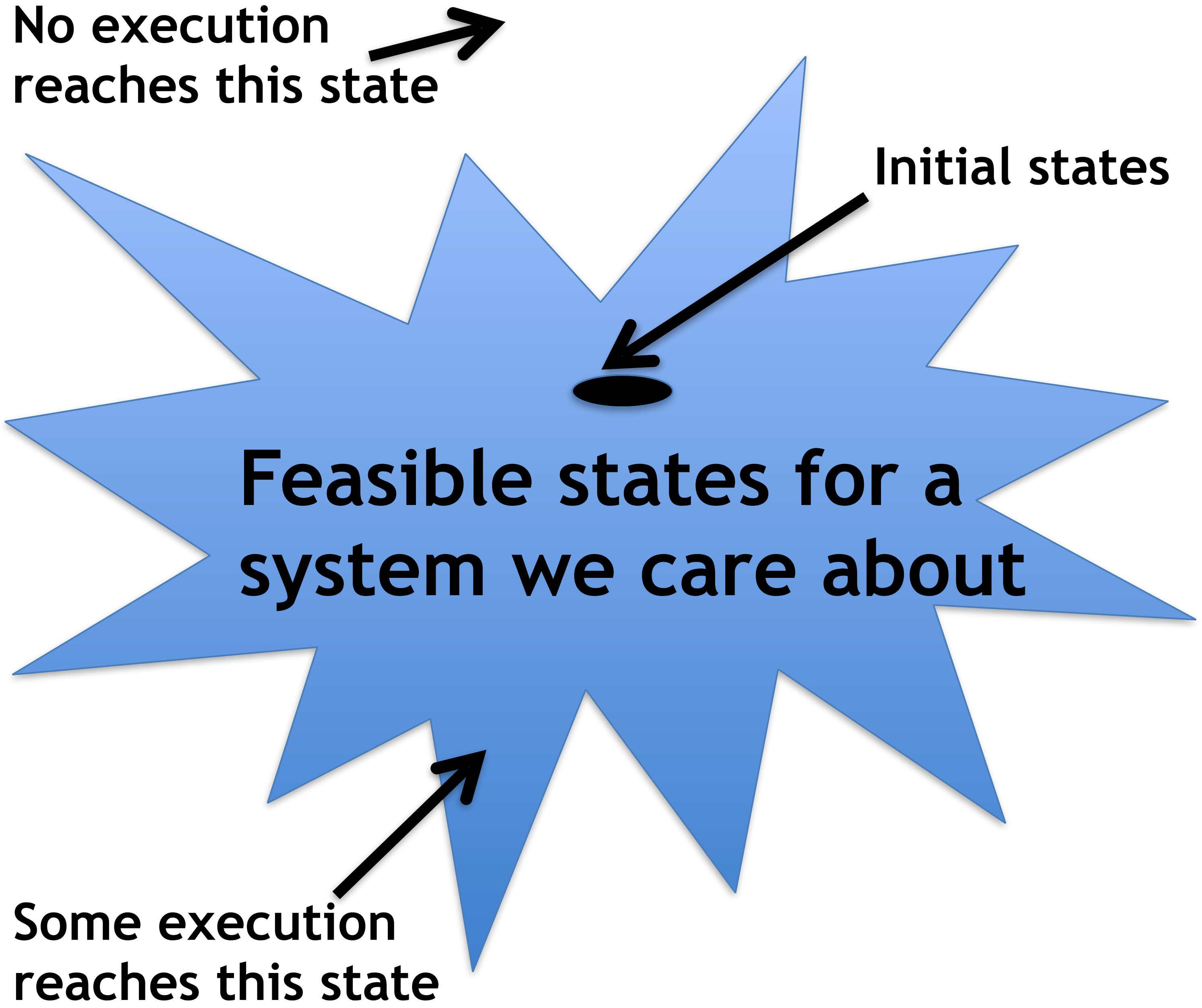
SQLite au peigne fin

Pascal Cuoq
John Regehr

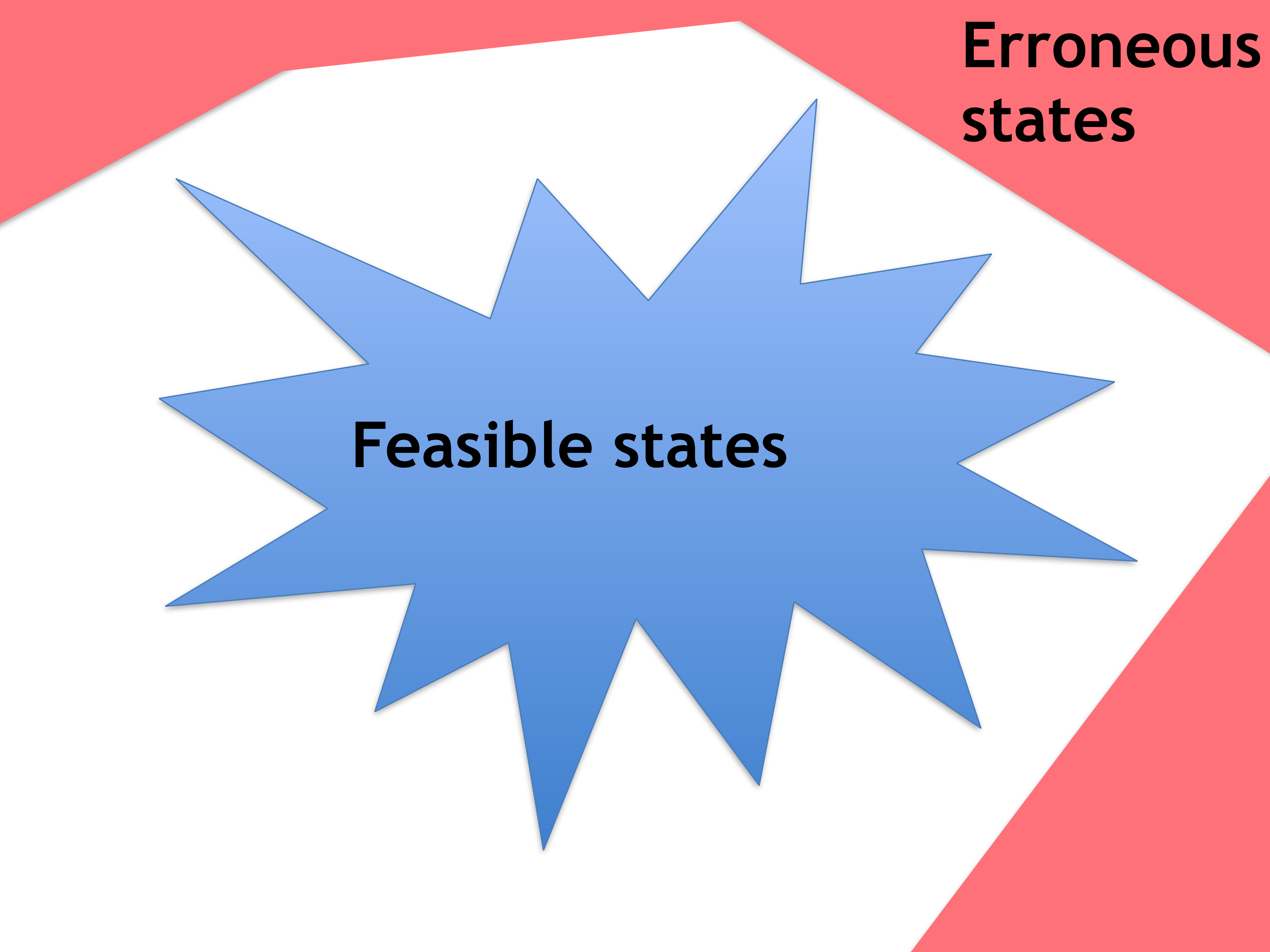
Feasible states for a system we care about

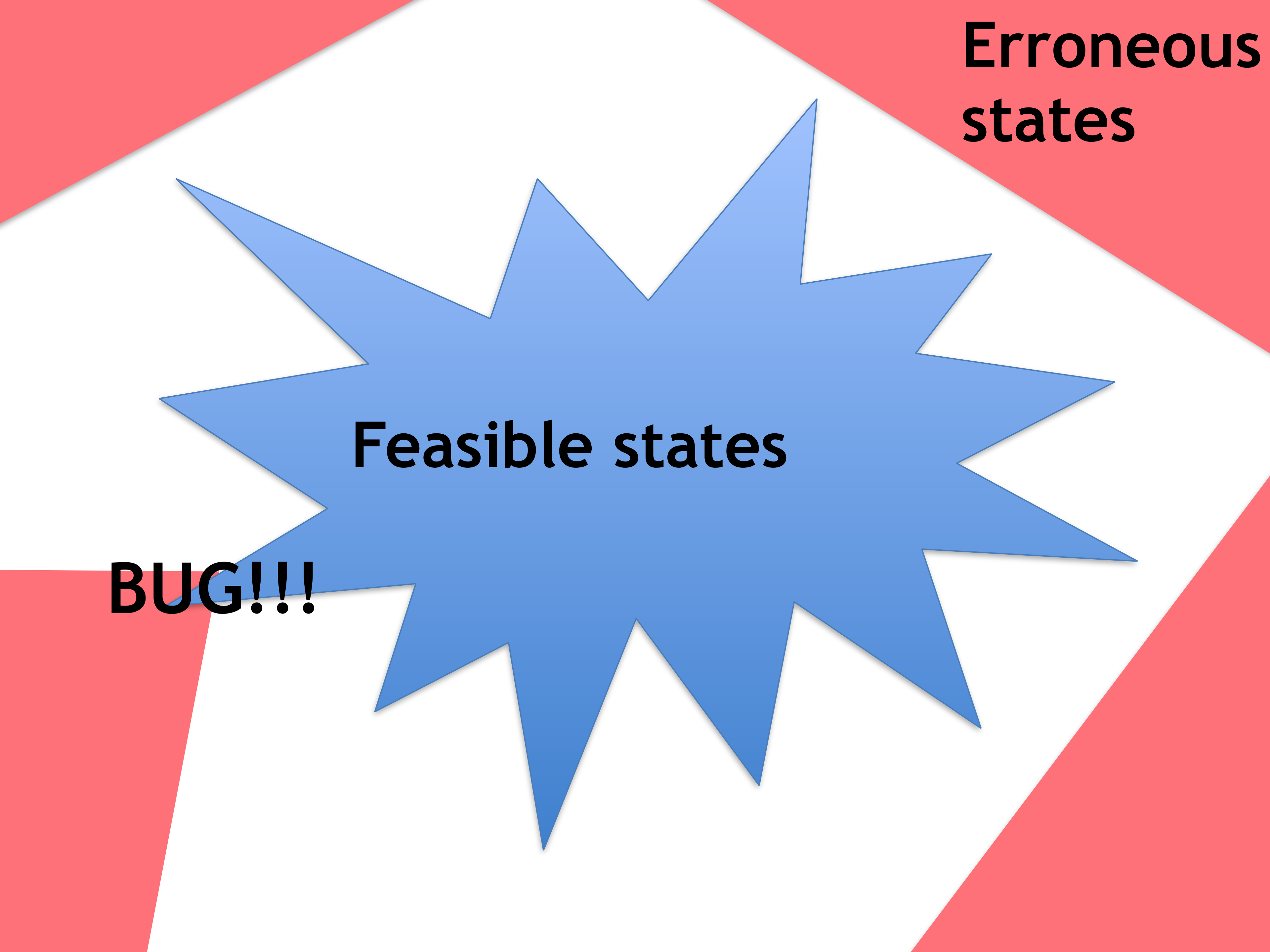

No execution reaches this state
Initial states
Feasible states for a system we care about
Some execution reaches this state

Feasible states

Figuring out whether an arbitrary state is feasible is very, very hard

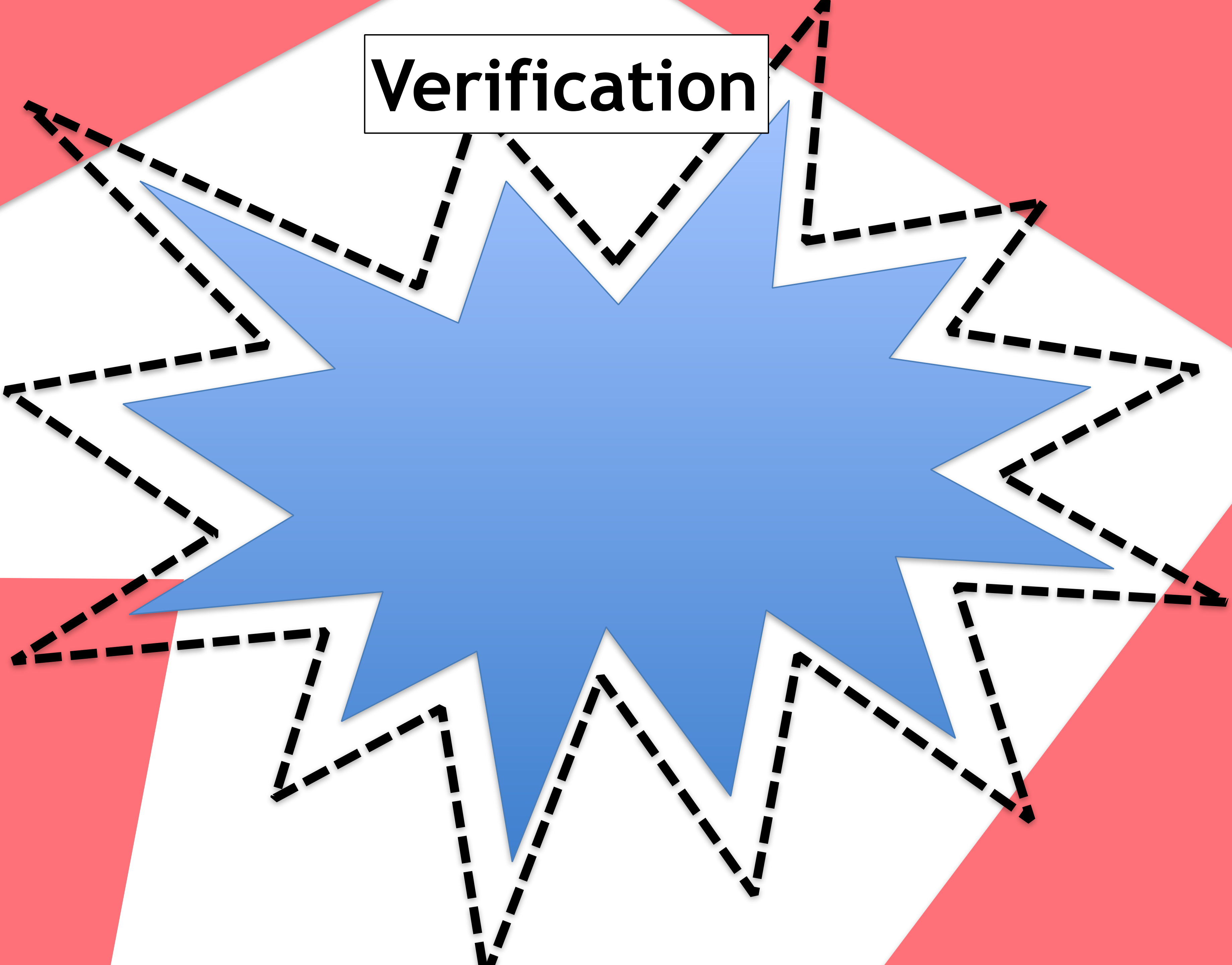
Feasible states

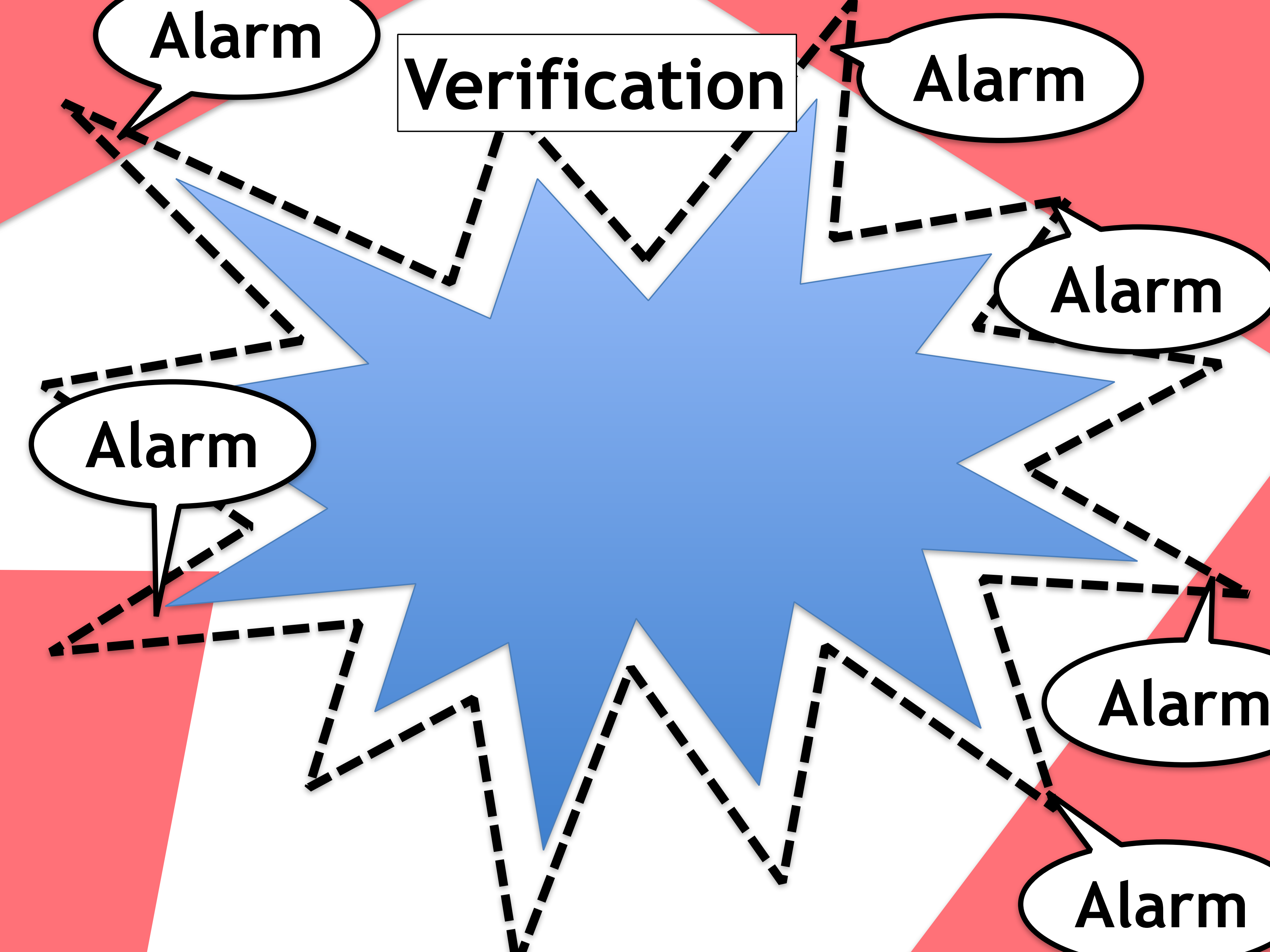

Erroneous states
Feasible states


Erroneous states
Feasible states
BUG!!!

Verification

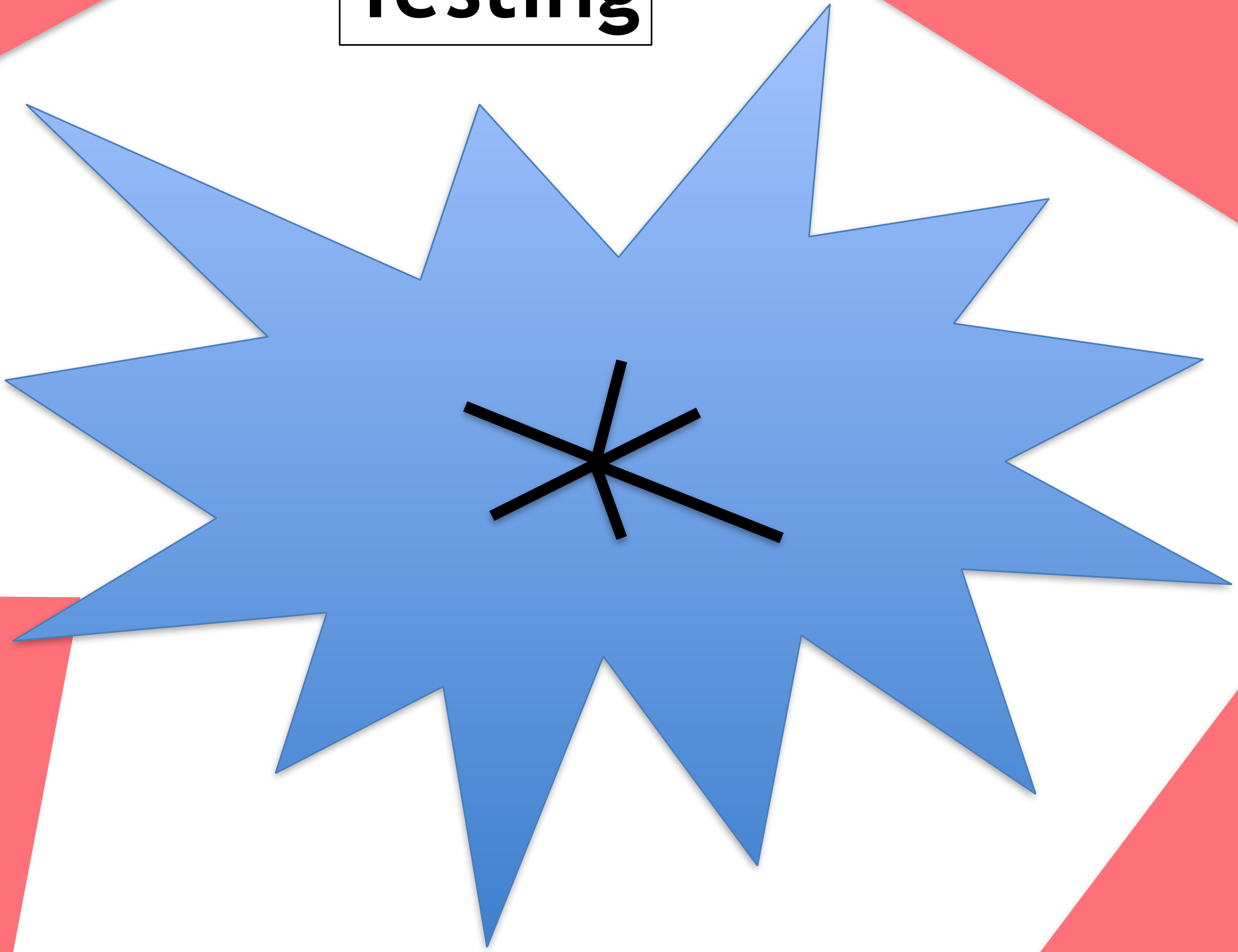
Verification



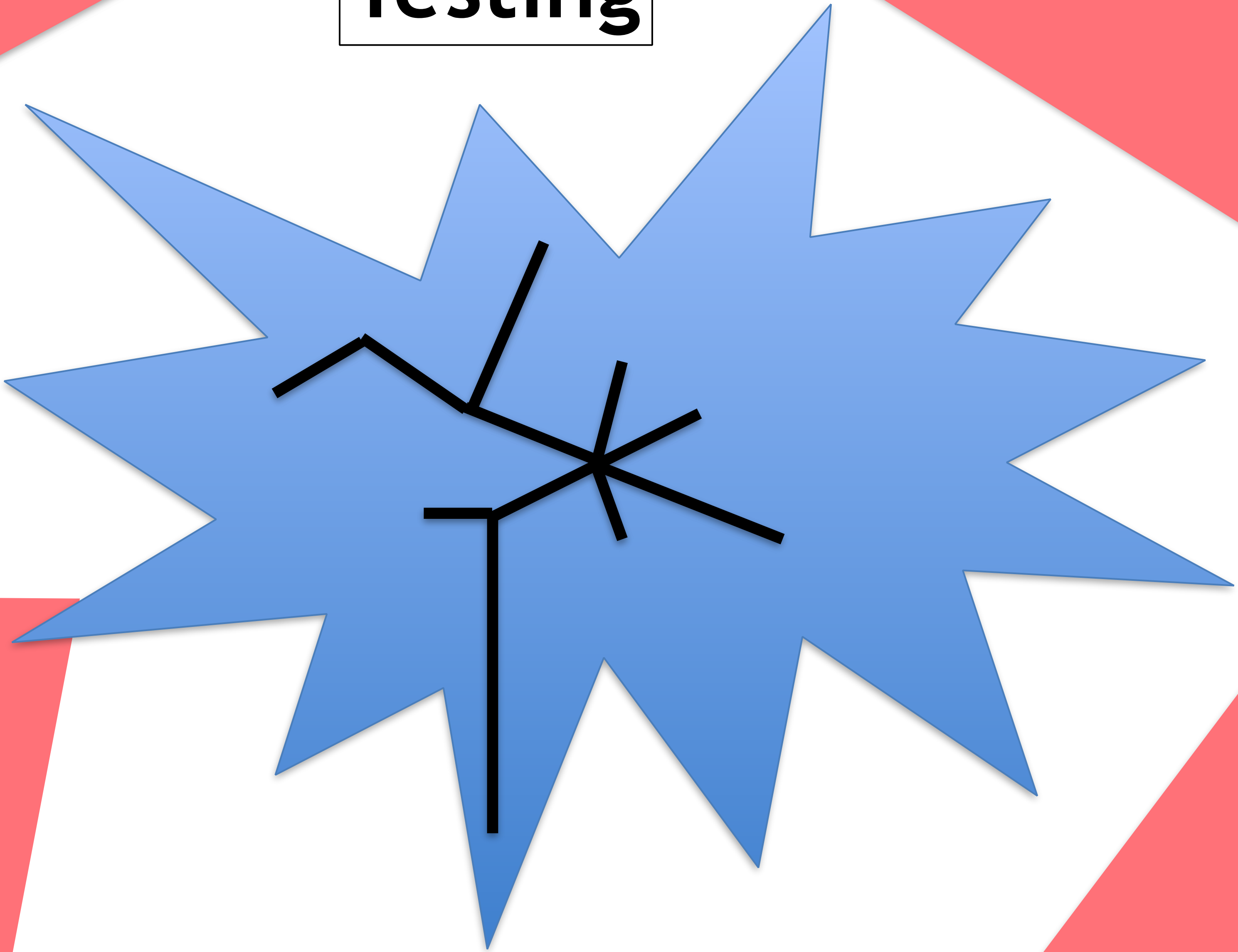

Alarm
Verification
Alarm
Alarm
Alarm
Alarm
Alarm

Testing

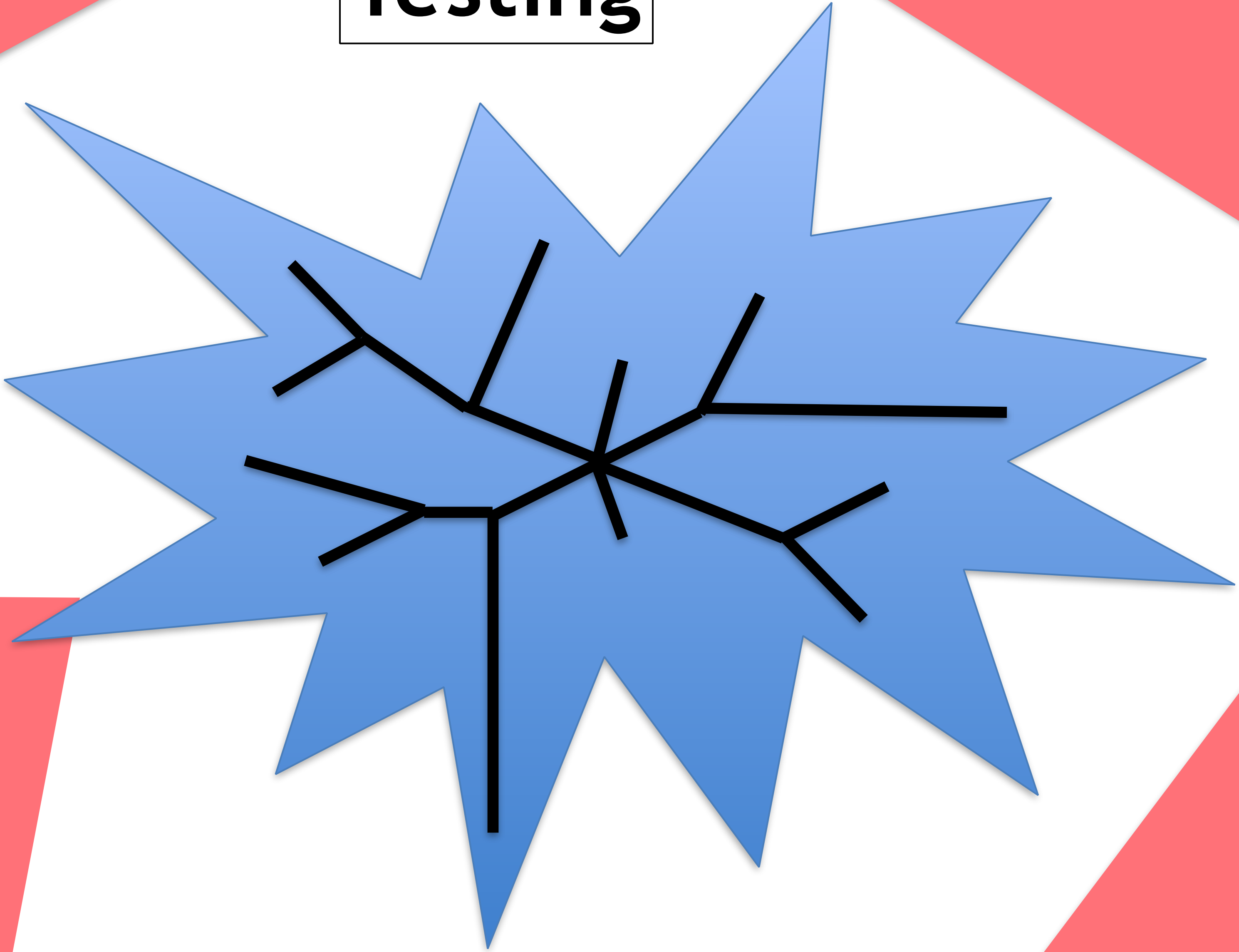
Testing



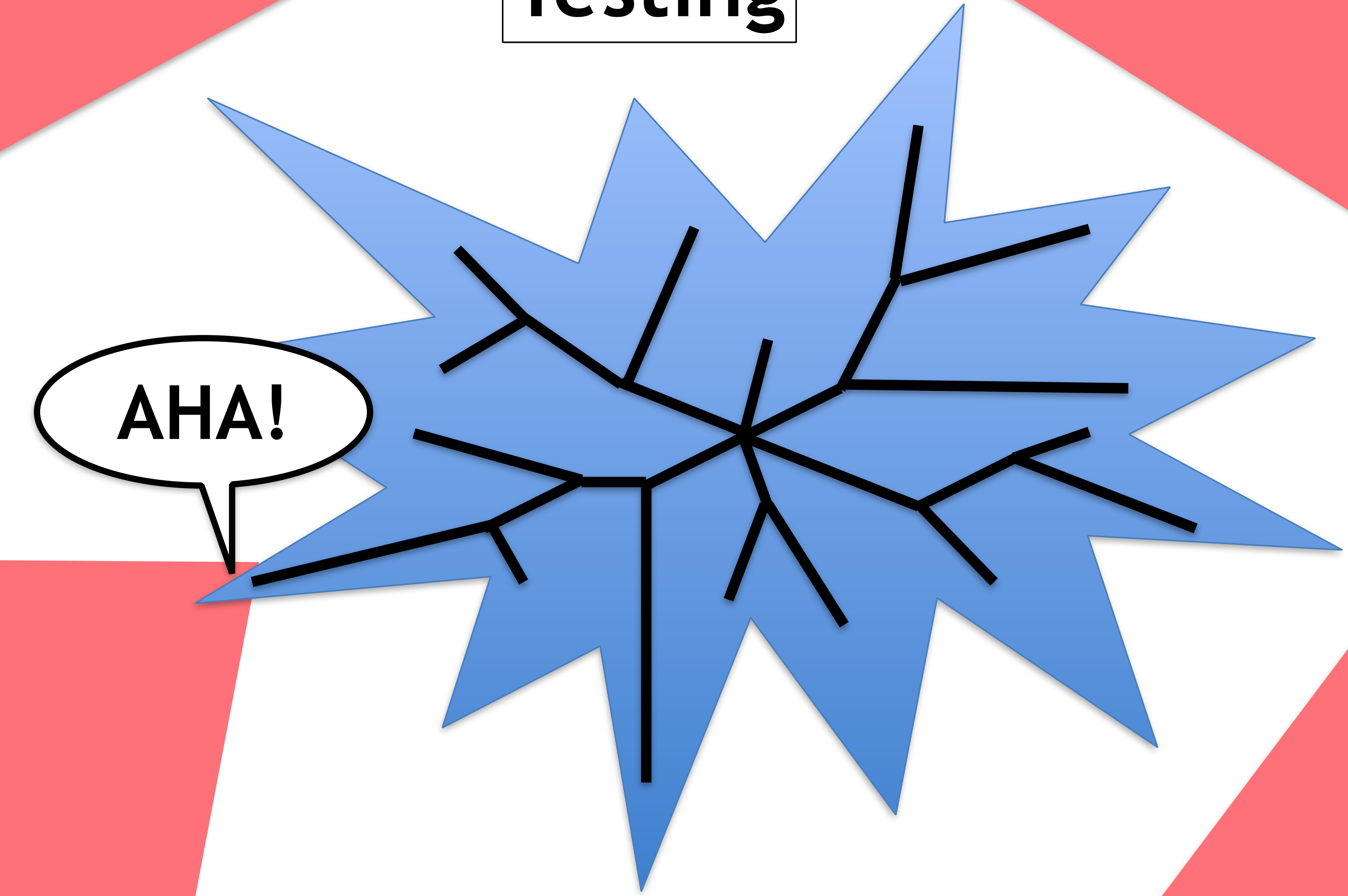
Testing



Testing



Testing



- Testing is unsatisfying: no guarantees
  - In practice, testing almost invariably misses critical bugs
  - Even microprocessors and rockets ship with nasty bugs

However, it can make sense to do testing first, verification second

- legacy security-critical code probably comes with some tests
- Finding bugs during verification makes verification more difficult
  - We want verification to be about proving absence of bugs, not about finding bugs

Restricted mode of Frama-C's value analysis

- enforces abstract states that model single concrete states
- no join
- initiated for Csmith testing
- continued at TrustInSoft with CII funding
- tis-interpreter lets us detect a wide variety of very subtle C undefined behaviors as a side effect of normal testing

An **undefined behavior** in C and C++ (and other languages) is a program error that

- is not caught by the compiler or runtime library
- is assumed to not happen by the compiler
- invalidates all guarantees made by the compiler

Basically all non-trivial C and C++ programs execute undefined behaviors

- Thus, according to the standards, almost all C and C++ programs are meaningless
- Including, for example, most of the SPEC CPU 2006 benchmarks

- This function executes undefined behavior:

```
int foo(int x, int y) {
  return (x + y) >> 32;
}
```

- This function executes undefined behavior:

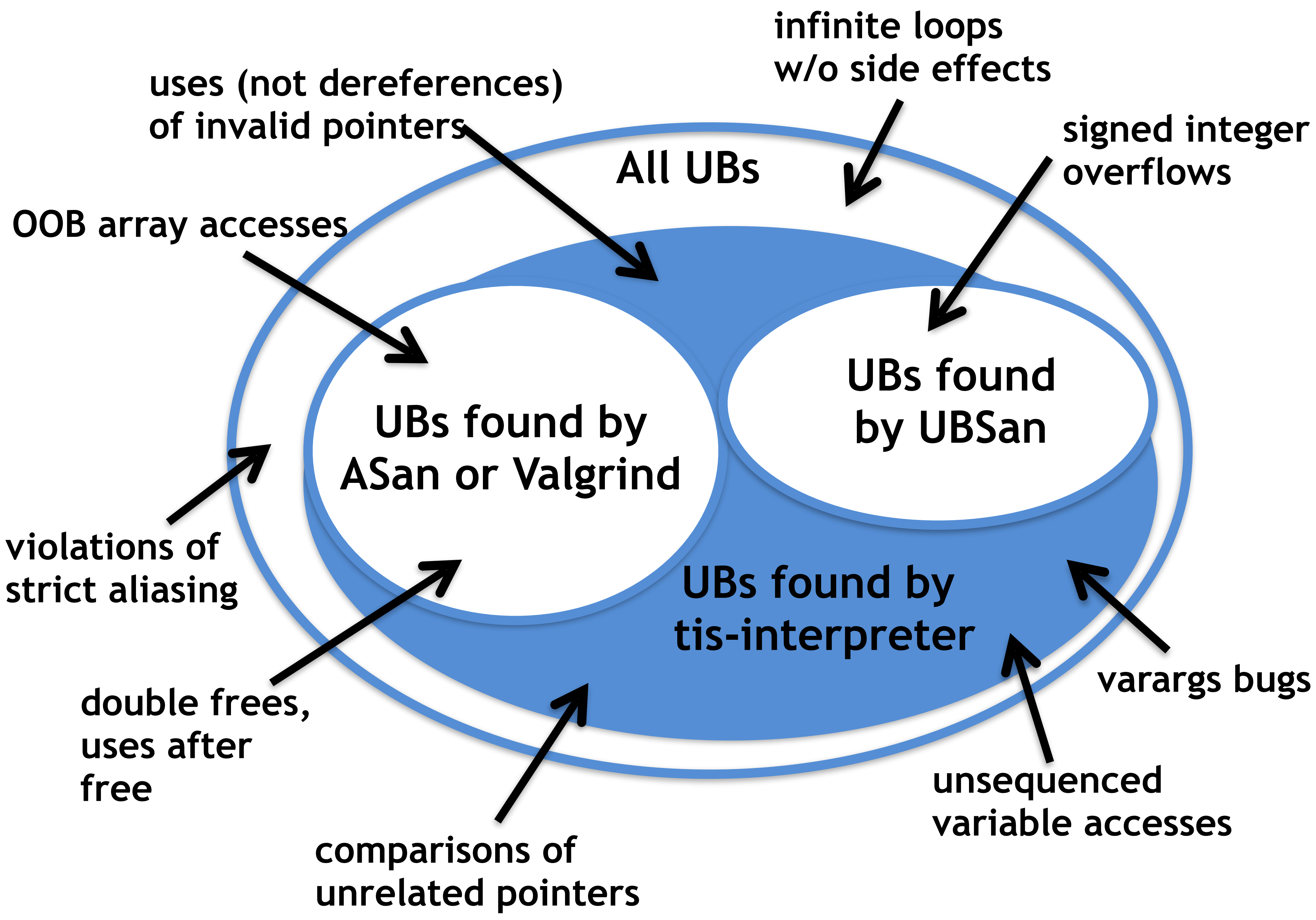
```
int foo(int x, int y) {
  return (x + y) >> 32;
}
```

Latest version of LLVM emits:

```
foo:
    retq
```

- Most safety-critical and security critical software is written in C and C++
- Undefined behavior is a huge problem
  - Responsible for a large fraction of major security problems over the last 20 years

- The solution is tools
  - Static analysis to find bugs at compile time
  - Dynamic analysis to find bugs at runtime


infinite loops
w/o side effects
uses (not dereferences)
of invalid pointers
signed integer
overflows
All UBs
OOB array accesses
UBs found by
ASan or Valgrind
UBs found
by UBSan
violations of
strict aliasing
UBs found by
tis-interpreter
varargs bugs
double frees,
uses after
free
unsequenced
variable accesses
comparisons of
unrelated pointers

We've been applying tis-interpreter to widely used, security-critical open source libraries

- Crypto
  - PolarSSL, OpenSSL, LibreSSL, s2n
- File processing
  - libjpeg, libpng, libwebp, bzip, zlib
- Databases
  - SQLite

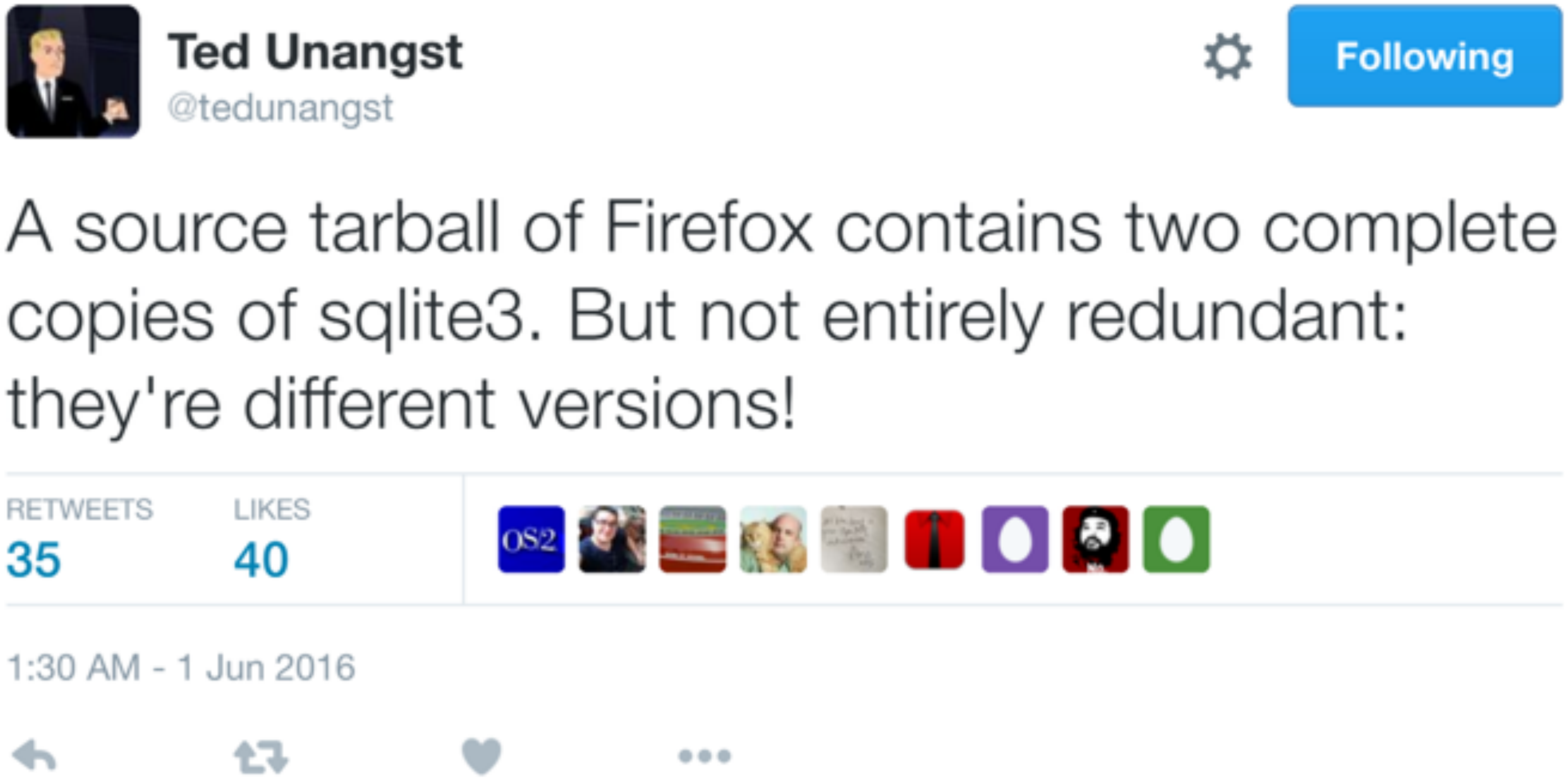
Where do we get test cases?

- Test suites
- afl-fuzz

SQLite

- Open source embedded SQL database
- ~113,000 lines of C
- Most widely deployed SQL database (probably by multiple orders of magnitude)
- One of the most widely deployed software packages period
  - Most phones, web browser instances, smart TVs, set top boxes contain at least one instance
- https://www.sqlite.org

SQLite in Firefox


Ted Unangst
@tedunangst

Following

A source tarball of Firefox contains two complete copies of sqlite3. But not entirely redundant: they're different versions!

RETWEETS
35

LIKES
40

1:30 AM - 1 Jun 2016

SQLite is extensively tested

- Test cases written by hand
  - 100% MC/DC coverage!
  - Every entry and exit point is invoked
  - Every decision takes every outcome
  - Every condition in a decision takes every outcome
  - Every condition in a decision is shown to independently affect the outcome of the decision
- Test cases generated automatically by fuzzers
- https://www.sqlite.org/testing.html
- Executions are examined by checking tools such as Valgrind

Are there problems in SQLite left for us to find?

memcpy(b1, b2, s) must be passed buffers b1 and b2 valid for the full length s even if they differ early.

http://trust-in-soft.com/memcmp-requires-pointers-to-fully-valid-buffers/

- SQLite sometimes uses the pattern:

```
memcmp(e, "unix-excl", 10)
```

where e's validity can be shorter than 10

Library functions such as memcpy() and memset() assume that their pointer arguments are non-null

- SQLite sometimes calls these functions with null arguments

```
void foo(char *p1, char *p2, size_t n) {
  memcpy(p1, p2, n);
  if (!p1)
    error_handler();
}
```

Library functions such as memcpy() and memset() assume that their pointer arguments are non-null

- SQLite sometimes calls these functions with null arguments

```
void foo(char *p1, char *p2, size_t n) {
  memcpy(p1, p2, n);
  if (!p1)
    error_handler();
}
```

Code generated by GCC:

```
foo:
    jmp memcpy
```

```
int sqlite3_config(int op, ...) {
  …
  var1 = va_arg(ap, void *);
  var2 = va_arg(ap, void *);
  …
}
```

OK to call like this?

```
void *pLog = …;
sqlite3_config(CONFIG_LOG, 0, pLog);
```

```
int sqlite3_config(int op, ...) {
  …
  var1 = va_arg(ap, void *);
  var2 = va_arg(ap, void *);
  …
}
```

Correct call:

```
sqlite3_config(CONFIG_LOG, (void *)0, pLog);
```

How can this kind of bug go undetected?

```
int sqlite3_config(int op, ...) {
   …
}
```

Co

sq g);

Ho

On x86:

- int and pointer are the same size
- Integer 0 and null pointer have the same representation
- No problem!

On x86-64:

- int has size 4 and pointer has size 8
- First six integer arguments are passed in registers
- No problem!

On other platforms, memory corruption is possible

- Many occurrences of integer zero values being passed as null pointers
- Also, a few other bugs such as more arguments being popped than pushed
- Are varargs bugs common?
  - We don't know
  - Bugs in calls to variadic standard library functions are caught by custom compiler warnings
  - Bugs in user-written variadic code get no checking whatsoever

C does not initialize automatic variables.

Valgrind tracks initialization at bit level, allowing detection of accesses to uninitialized storage

- But Valgrind analyzes compiled code
- The compiler can hide errors, for example by reusing stack memory that was already initialized

tis-interpreter always finds these bugs

  – Including several in SQLite

```
int dummy;
some sort of loop {
   ...
   // we don't care about function()’s
   // return value (but its other
   // callers might)
   dummy += function();
   ...
}
// dummy is not used again
```

A pointer in C becomes illegal to use once the storage to which it points is freed

- We found many locations where SQLite frees memory and then continues to use the invalid pointers

```
req1_malloc02_alignment(p, z);
sqlite3_realloc(z, 0);
th3testCheckTrue(p, z!=0);
```

Creating a pointer ahead of or more than one element past the end of a block of storage is illegal in C

```
int a[10];
int *p1 = &a[-1];  // illegal
int *p2 = &a[9];   // pointer to last element
int *p3 = &a[10];  // OK (one past the end)
int *p4 = &a[11];  // illegal
```

SQLite computed illegal pointers…

- On purpose: systematic use of pointers to array[-1]
  - 1-based array indexing w/o wasting RAM
- Accidentally, as part of input validation
  - This error is seen in almost all C code

Results testing SQLite using tis-interpreter:

- Many bugs fixed
- Developers are now more aware of subtleties of the C standard
  - They had been writing “1990s C code” which ignores many undefined behaviors

tis-interpreter improvements:

- Recursion
- va_list
- Built-in support for many standard functions
- especially file access: open(), read(), …
- invalid pointer arithmetic (pinpoint problem)

Missing from tis-interpreter to support SQLite better:

- mmap()
- mkdir()
- fcntl()

Missing to support other packages:

- setjmp() / longjmp()
- Intrinsics and inline assembly

- The C language is full of subtle undefined behaviors
  - Some are directly harmful
  - Others matter because compilers assume they will not happen
- tis-interpreter uses existing test cases to find these bugs
- Testing using tis-interpreter is a very useful prelude to formal verification
- tis-interpreter is open source
  - http://trust-in-soft.com/tis-interpreter/