



EADS INNOVATION WORKS
Penjili: static analysis for security of embedded soft.
Finding all bugs in the haystack...



Charles Hymans
EADS IW SE IA
charles.hymans@eads.net
Xavier Allamigeon, Jean-Loup Carré, Wenceslas Godard, Sarah Zennou



Penjili: static analysis tool for embedded software

- Penjili automatically **proves** the absence of bugs
- Penjili targets:
 - **memory manipulation bugs** (array, pointer and string out-of-bounds)
 - **arithmetic bugs** (integer overflows, division by zero)
- These bugs are security breaches that make the code vulnerable to attacks, such as system takeover



Some errors detected by Penjili

- Array out-of-bounds

```
char a[10];  
a[11] = 1;
```
- Pointer out-of-bounds

```
char a[10];  
char *ptr;  
ptr = &a[0];  
*(ptr + 11) = 1;
```
- Integer overflows

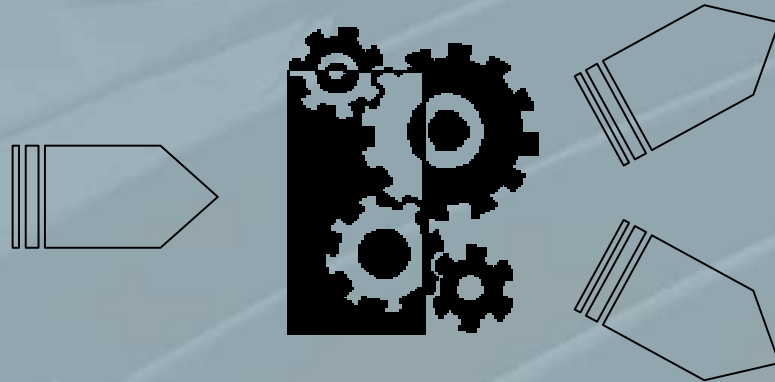
```
int i;  
i = 2147483647;  
i = i + 1;
```



Principle of the tool

- Inputs C source code

```
i = 0;
while (i < 10) {
  x = t[i];
  if (x >= 10)
    return;
  u[x] = 1;
  i++;
}
```

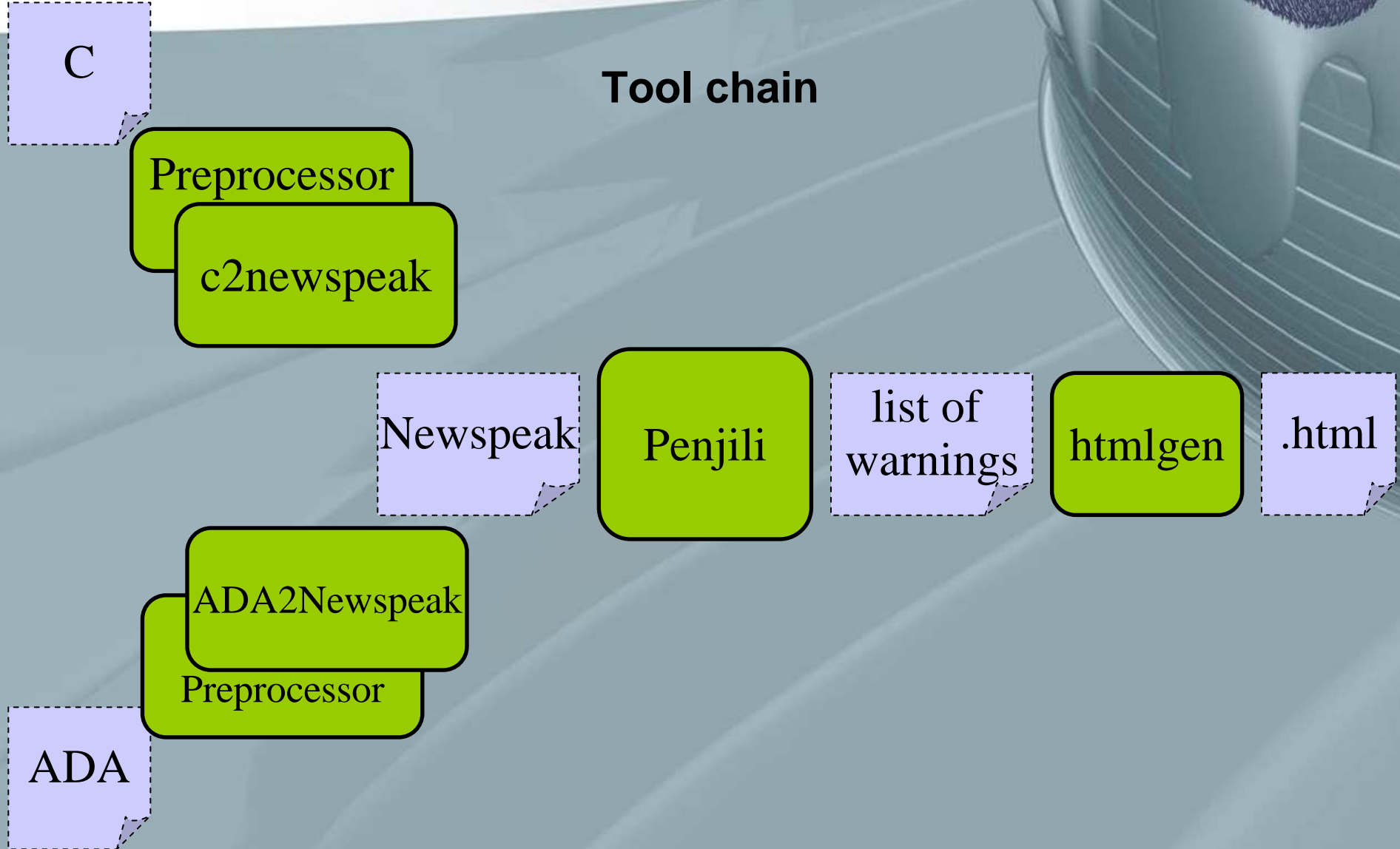


line 1073: pointer out-of-bounds
line 3206: array out-of-bounds

- Outputs no warnings: the code is proved correct
- Outputs some warnings:
 - either real bugs
 - or false alarms



Tool chain





Newspeak design rationale and paradigm

- **Precise:** its semantics is formally defined
- **Expressive:** it should be possible to translate most C into Newspeak
- **Simple:** few primitives, as classical and concise as possible
- **Minimal:** no primitives or fragment of primitive should be expressible as a combination of other primitives
- **Explicit:** all primitives can be executed without context
- **Analysis-oriented:** primitives have annotations necessary to perform correctness checks
- **Architecture-independent:** all architecture dependent features (sizes, offsets...) should be made explicit

High level assembly language with annotations for analysis



Newspeak whole syntax

- $t = \text{int}(\text{sign}, n), \text{float } n$
 | ptr, fptr
 | $t[n]$
 | $\{ (o_1: t_1) \dots (o_n: t_n) \}$
- $\text{blk} = \text{cmd}_1 \dots \text{cmd}_n$
- $\text{cmd} =$
 | $\text{lv} = (t) e$
 | $\text{lv}_1 := \text{lv}_2$
 | $t \ x; \text{blk}$
 | $e_1 \rightarrow \text{blk}_1 + e_2 \rightarrow \text{blk}_2$
 | $\text{forever } \text{blk}$
 | $\text{do } \text{blk} \text{ with } \text{lbl}$
 | $\text{goto } \text{lbl}$
 | $\text{call } \text{fun}$
- $e = i, f, \text{null}$
 | $\text{lv} : t$
 | $\&(\text{lv}, n)$
 | $\&\text{fid}$
 | $\text{op } e \mid e_1 \text{ op } e_2$
- $\text{lv} = x- \mid g$
 | $[e]n$
 | $\text{lv} + e$
- $\text{fun} = \text{fid} \mid [e](t_1..t_n \rightarrow t)$
- $\text{op} = \text{belongs}[l; u]$
 | $\text{coerce}[l; u]$
 | $\text{not} \mid > \mid ==$
 | $\text{bnot} \mid \text{bor} \mid \text{band} \mid \text{bxor} \mid \ll \mid \gg$
 | $(t_1 \rightarrow t_2)$
 | $+ \mid - \mid * \mid / \mid \%$
 | $+. \mid -. \mid *. \mid /.$
 | $+\text{ptr} \mid -\text{ptr}$



Newspeak distributed under LGPL

- Compiles most ANSI C (no backward gotos)
 - conditional expression, bitfields, variable number of arguments
- Compiles some GNU C
 - flexible array members, transparent unions, __typeof, ...
- Rejects dirty code by defaults, options to accept:
 - dirty casts, dirty syntax, missing prototypes, multiple definitions
- Several utilities:
 - c2newspeak, ada2newspeak
 - npkstats
 - npksimplify, npkstrip
 - npkbugfind, npkpointer
- Compiles 40 million lines of macro-expanded C in 9 minutes



Translation example

```
int x;  
void main() {  
    int y;  
    int z;  
    x = y;  
    x = z;  
}
```

```
int4 x = 0;  
main() {  
    int4;  
    int4;  
    x =(int4) 1-_int4;  
    x =(int4) 0-_int4;  
}
```

- Integer size is made explicit
- Local variables are pushed on a stack



Translation example

```
int x, y, z;  
x = y + z;
```

```
int4;  
int4;  
int4;  
2- =(int4)  
coerce[-2147483648,2147483647] (1-_int4 + 0-_int4);
```

- Integer coercion after any arithmetic operation is explicit



Translation example

```
struct {
    int a;
    int b;
} x;
int t[10];
int i;
x.b = t[i];
```

```
{
    int4 0;
    int4 4;
}8;
int4[10];
int4;
2- + 4 =(uint4) (1- + (belongs[0,9] 0-_int4 * 4))_int4;
```

- Structure size is computed
- Field access is performed with shift
- Annotation for array bound check is added
- Offset computation is explicit



Translation example

```
int* x;  
int t[100];
```

```
x = &t[3];  
x = x + 5;  
*x = 3
```

```
ptr;  
int4[100];  
  
1- =(ptr) (&_400(0-) + (3*4));  
1- =(ptr) (1-ptr + (5*4));  
[1-_ptr]4 =(int4) 3;
```

- Annotations are added to check pointer manipulations



Translation example

```
int x;  
x = 0;  
while (x < 10) {  
    x++;  
}
```

```
int4;  
0- =(int4) 0;  
do { forever {  
    choose {  
        | (10 > 0-_int4) ->  
        | !(10 > 0-_int4) -> goto lbl1;  
    }  
    0- =(int4) coerce[-2147483648,2147483647] (0-_int4 + 1);  
} with lbl1:
```

- Loops are infinite
- Breaks are explicit with forward gotos and labels
- They are necessarily well-structured and forward

http://www.penjili.org/newspeak.html

Fichier Edition Affichage Favoris Outils ?

Newspeak



Newspeak

In the end the whole notion of goodness and badness will be covered by only six words – in reality, only one word. Don't you see the beauty of that, Winston?

Nineteen eighty-four, George Orwell.

Home
Newspeak
Publications
Links

Newspeak is a simplified programming language, well-suited for the purpose of static analysis. **C2Newspeak** compiles C programs into Newspeak. C2Newspeak is distributed under the LGPL.

C2Newspeak v. 0.9 source code: [C2Newspeak-0.9.tgz](#). It uses CIL v. 1.3.5, which can be downloaded at <http://sourceforge.net/projects/cil>.

Examples

http://www.penjili.org

Legend

Here are a few compilation examples from C to Newspeak. In the following, the C code will be on the left side and the corresponding Newspeak code on the right side:

C code	Newspeak code
--------	---------------

Types

Integer types are normalized according to their size and sign. Their size, which is architecture dependent, is made explicit

C code	Newspeak code
<pre>int i1; unsigned int i2; char i3; unsigned char i4;</pre>	<pre>int4 i1; uint4 i2; int1 i3; uint1 i4;</pre>

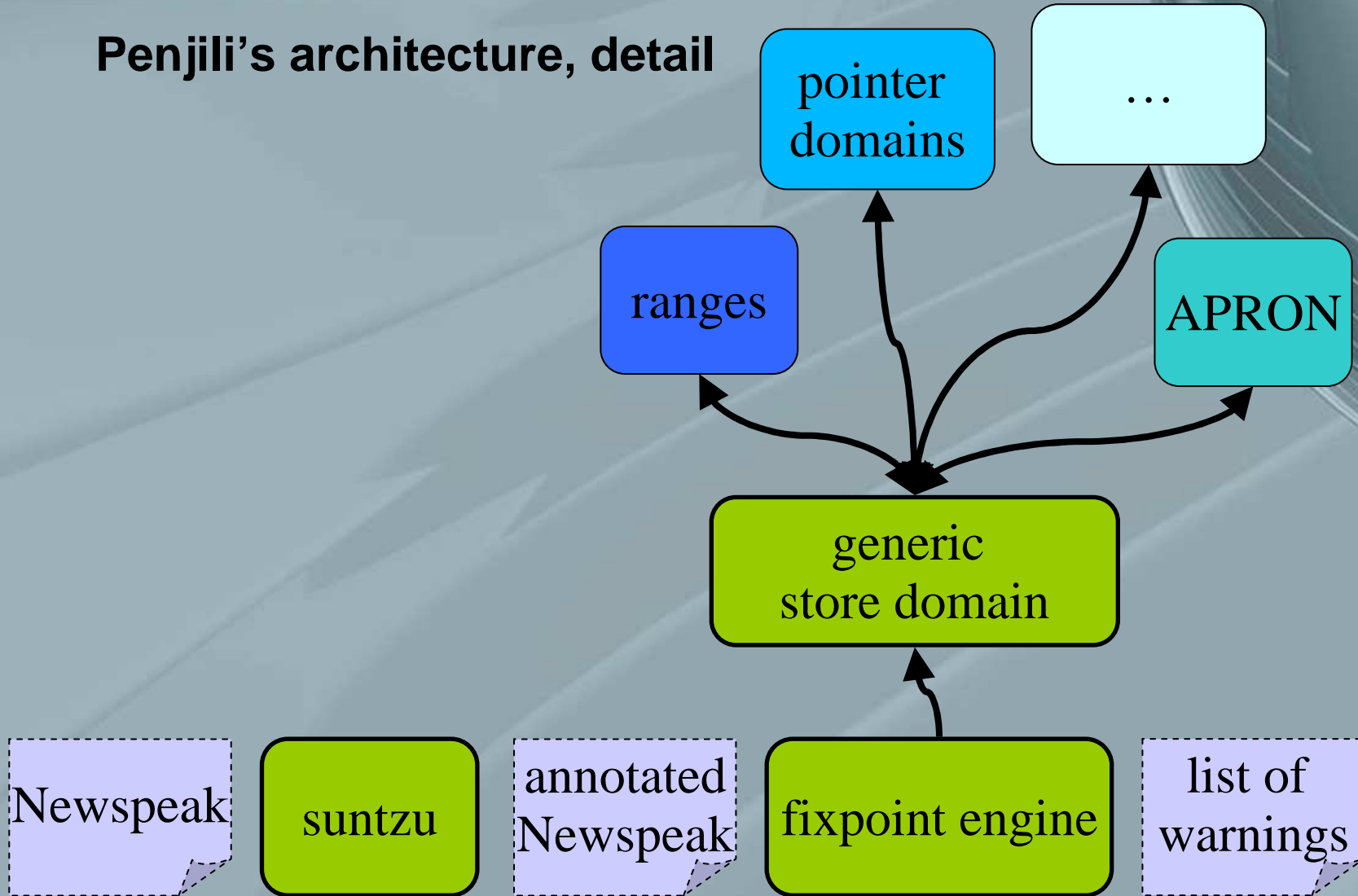
Casts (and unions) in C allow programmers to manipulate sequences of bytes with any type. Consequently, Newspeak distinguishes only two types of pointers: data and function pointers.

C code	Newspeak code
<pre>int *p1;</pre>	<pre>ptr p1;</pre>

Internet 100%



Penjili's architecture, detail





Some Penjili internal features

- Several fixpoint algorithms
 - Unroll, union, widening, narrowing
 - Techniques to avoid unnecessary re-analyses of code
- Several domains coexist
 - Ranges, equality/difference with zero
 - Several kinds of pointer and function pointer domains
 - Several kinds of string domains
 - Boolean expression propagation
 - APRON, weaker relational domains
 - Automatic and manual case splitting
 - Variable packing
 - ...
- Domains interact and communicate information to each other
- Store is made compact by merging cells together
 - Array smashing, ...
- Several implementation optimizations
 - Several optimized versions of sets and maps
 - Sharing via hash consing



Performance metrics for static analysis

- Precision:
 - O: number of potentially dangerous operations
 - A: number of alarms
 - Precision is: $(O - A)/O$
- Scalability: size of the code analysed within
 - A fixed amount of time (48h)
 - A fixed amount of memory (4Gb)



Precision: exemple

```
int t[10]; int i, x;
```

```
t[9] = t[1];
```

```
for (i = 1; i < 10; i++) {
```

```
    t[i-1] = t[i];
```

```
    t[i] = i;
```

```
}
```

```
x = t[9];
```

```
t[x] = 0;
```

```
t[11] = 0;
```

O=8 array accesses

A=2 alarms

precision: $(O-A)/O$

$6/8 = 75\%$

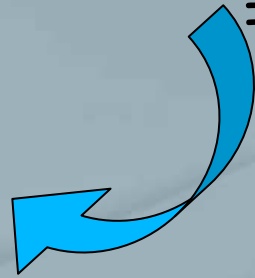
Note there is:

- 1 false alarm
- 1 error

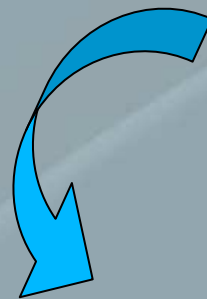


precision

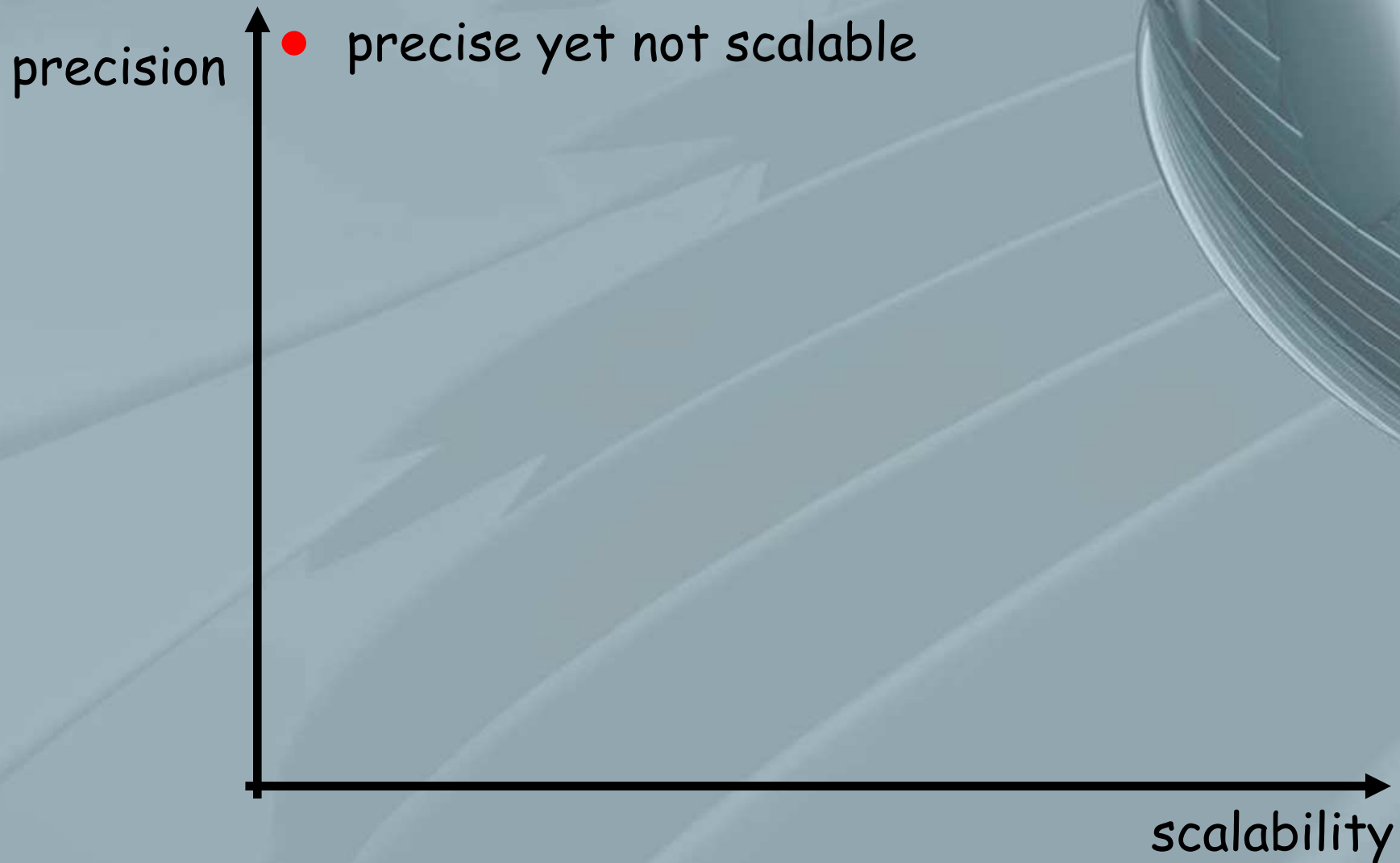
proved correct / dangerous operations
= 1 - alarms / dangerous operations

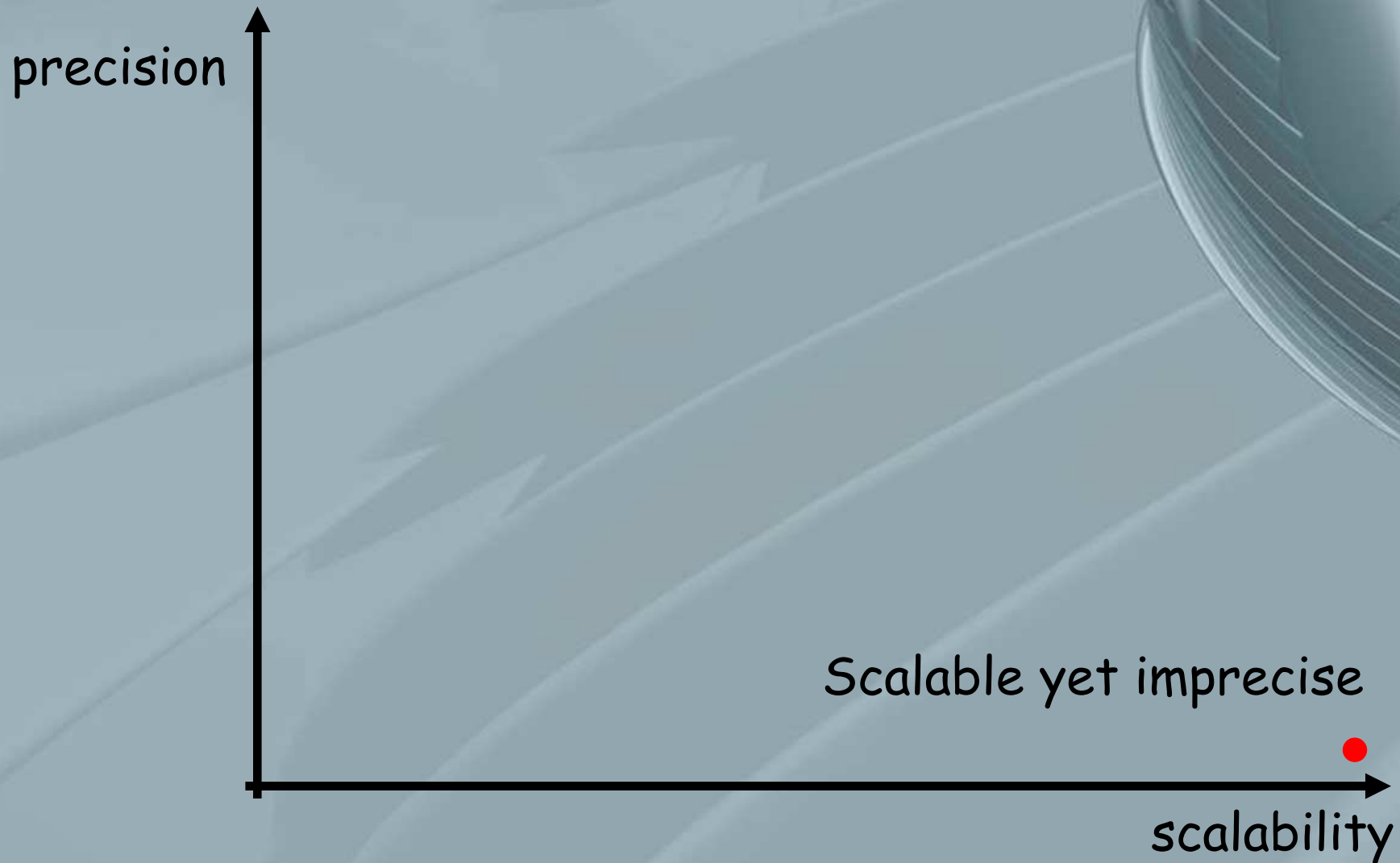


number of lines of code



scalability







Holy grail



precision

scalability



NIST database of 1690 C code

The purpose of the dataset is to provide with a set of known security flaws, for end users to evaluate tools and tool developers to test their methods

- Number of codes within the scope of Penjili: 1161
 - 289 OK / 872 with a vulnerability
- The tool results:
 - 291 codes raise 0 alarm
 - 848 codes raise 1 alarm
 - 10 codes raise 2 alarms
 - 12 codes raise 3 alarms or more

?



NIST database, conclusions

- At least 6 codes are wrongly classified in the database!
 - detected by the analyser
- Soundness of the tool is 100%
- Precision of the tool is at least 98%



Analysis of some EADS embedded software

Size Newspeak loc	2 870	9 314	14 017	33 769	757 755
Alarms reported	0	2	18	690	76
Anomalies found	0	2	> 6	–	> 5
Overall precision	100%	100%	99.6%	93.2%	99.3%

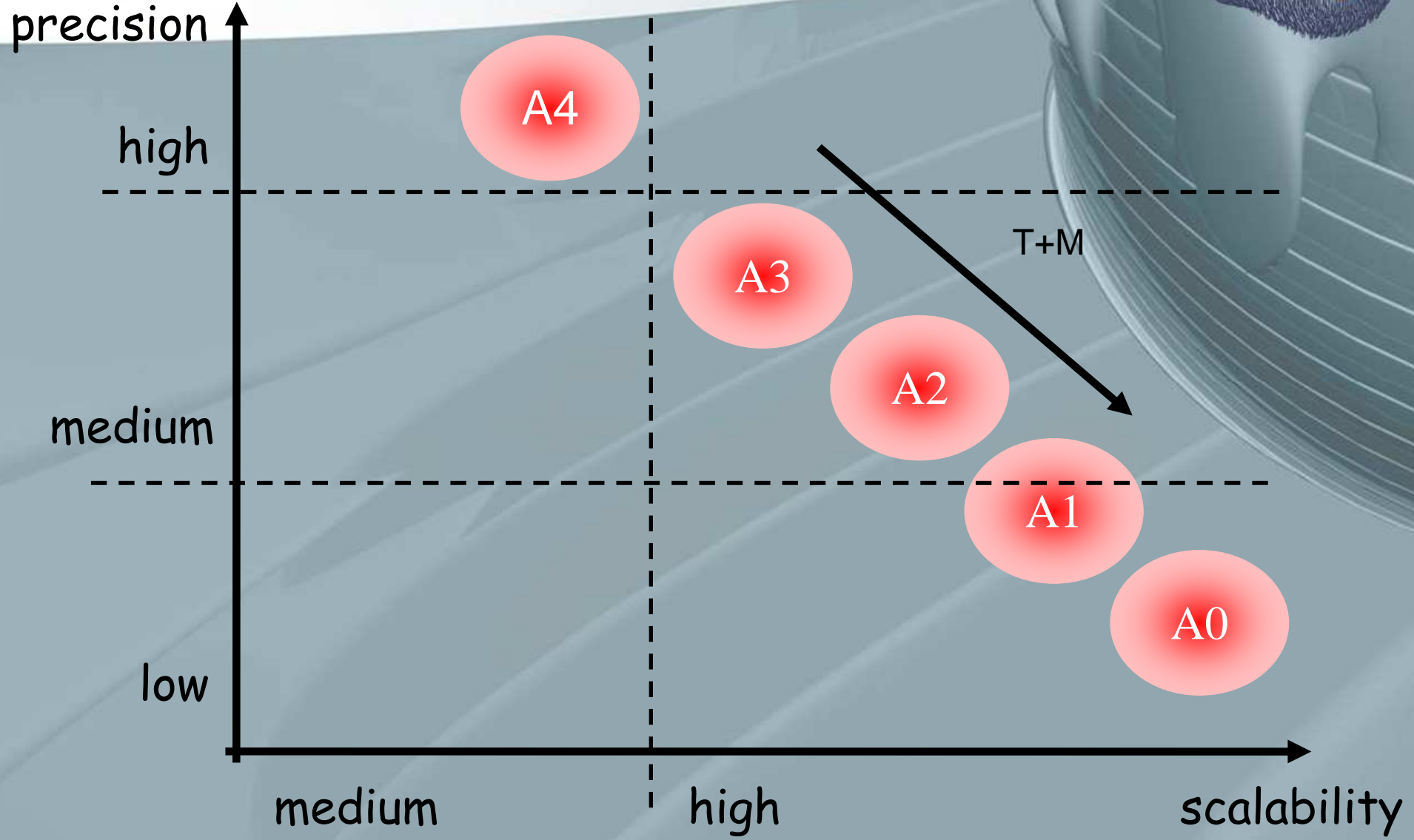
Currently, experiments on a very large and mature code

	Mini version	Complete version
Size (Kloc)	100 Kloc	1 726 Kloc
Number of threads	5 + init	15 + init
Number of globals	1805	108466
Number of functions	1153	14159
Number of loops	328	1882
Maximum depth of function calls	25	28
Maximum nesting depth of loops	7	7

Default analysis does not scale!



Penjili: several analyses, several options





Penjili: several analyses, several options

Analysis	Array precision	Pointer precision	Scalability
A0	Medium	None	High
A1	Medium	Low	High
A2	High	Low	High
A3	High	Medium	High
A4	High	High	Medium



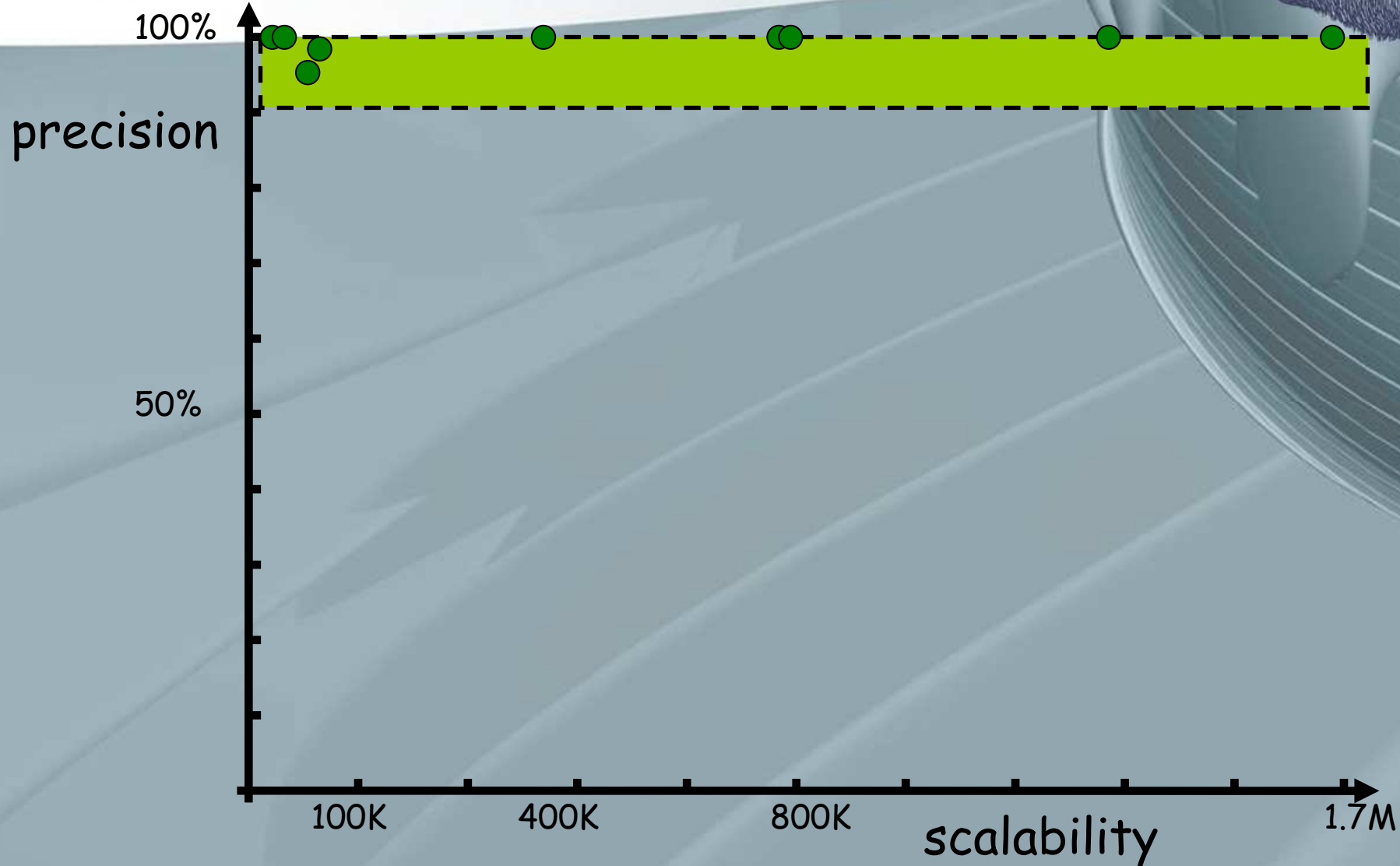
Options	Scalability Gain	Precision loss
T	Time	Small
M	Memory	Small-Medium

Results

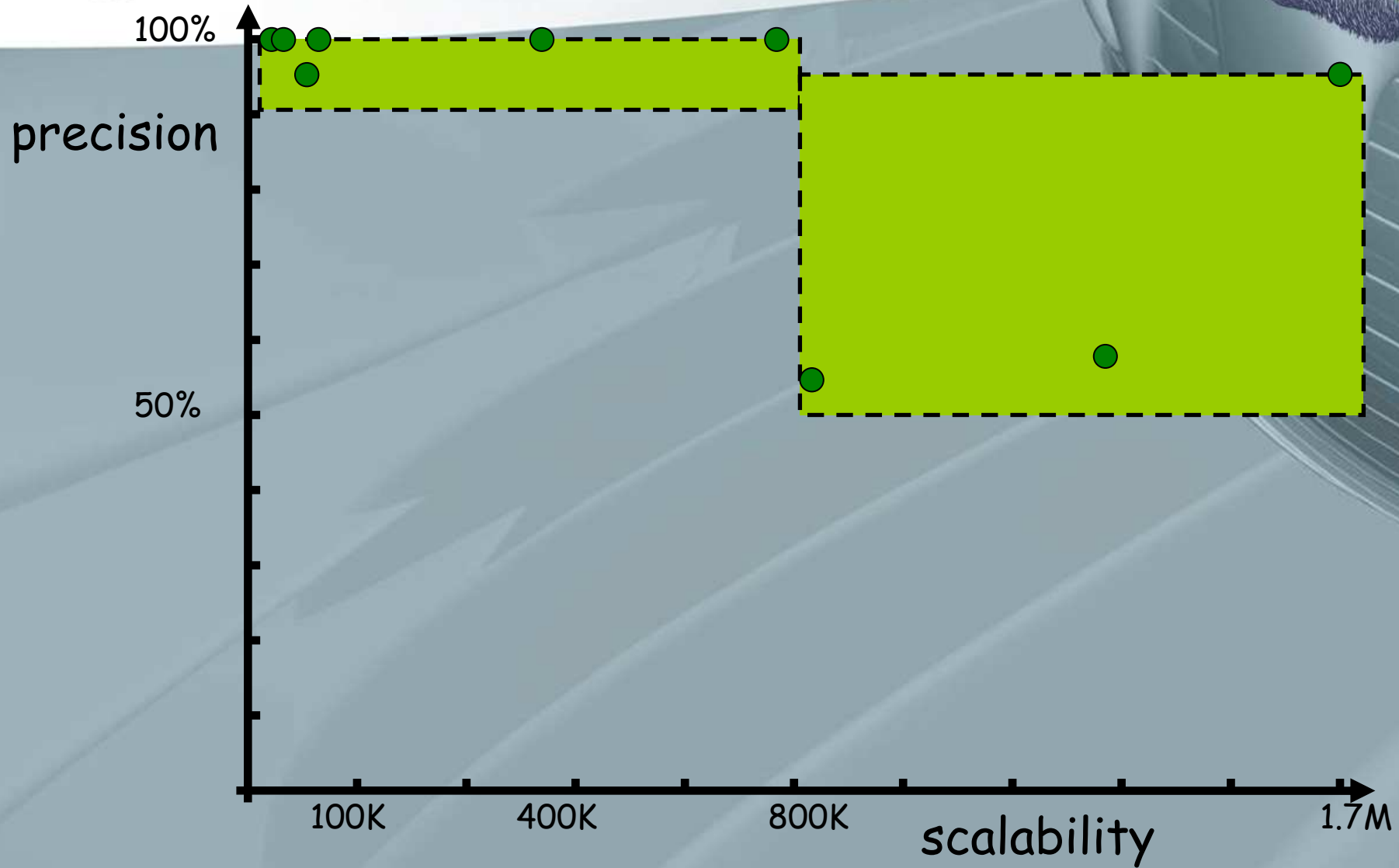
	Mini version	Complete version
Analysis	A4	A3
Options		T+M
Array precision (%)	98.42%	99.82%
Pointer precision (%)	99.69%	95.65%
Time (hms)	13m23s	11h35m52s
Memory (Mb)	98 Mb	943 Mb



Precision for array out of bounds



Precision for invalid pointer dereferences





Conclusion: Penjili

- Achieved great scalability:
 - > 1 million lines of code surpassed
- While retaining great precision:
 - > 99% for arrays, > 95% for pointers
- Was able to find anomalies in a real and mature software
- Is security experts wrapped in a box!