Move Fast and *Fix* Things

gholzmann@acm.org

nimble research

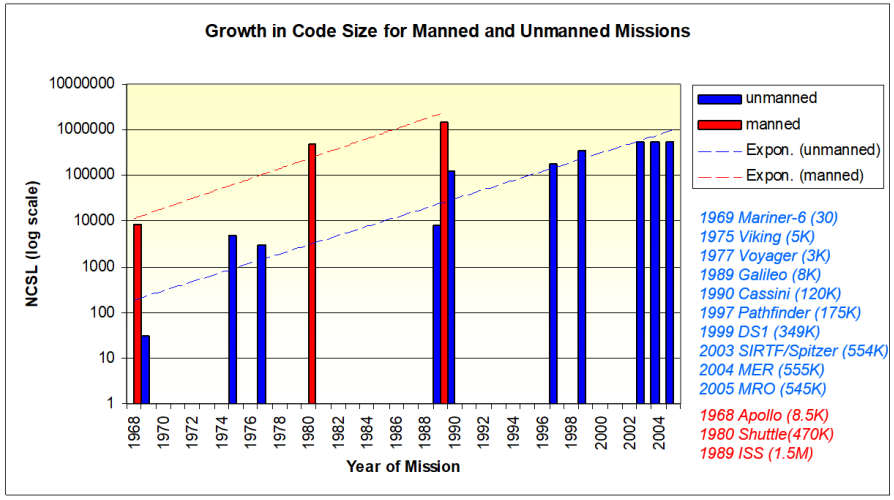
formerly: LARS
NASA/JPL Laboratory
for Reliable Software

problem: software breaks, and we should be able to find out *why* quickly, and *fix* it

do we have the right tools?

two unfortunate facts

- software grows to fill whatever space is available to it
  - corollary 1: software grows with time
  - corollary 2: it keeps getting harder to navigate all that code
- software *always* has undiscovered residual defects after testing
  - just like every issue of even the best newspapers list corrections to the previous issue
  - for an exceptional software process: ~0.1 per 1KLOC



	residual defects per KLOC	reference
NASA space shuttle avionics	0.1	[1]
Leading-edge software companies	0.2	[2]
Reliability survey	1.4	[3]
Military system survey	5 – 55	[4]

sources:

[1] E. Joyce, *Is error-free software possible?,* Datamation, Feb. 18, 1989.
[2] C. Jones, *Applied software measurement,* McGraw-Hill, 1991, p. 177.
[3] J.D. Musa et al., *Software reliability: measurement, prediction, application,* McGraw-Hill, 1990, p. 116.
[4] J.P. Cavano, F.S. LaMonica, *Quality assurance in future development environments,* IEEE Software, Sept. 1987, pp. 26-34.

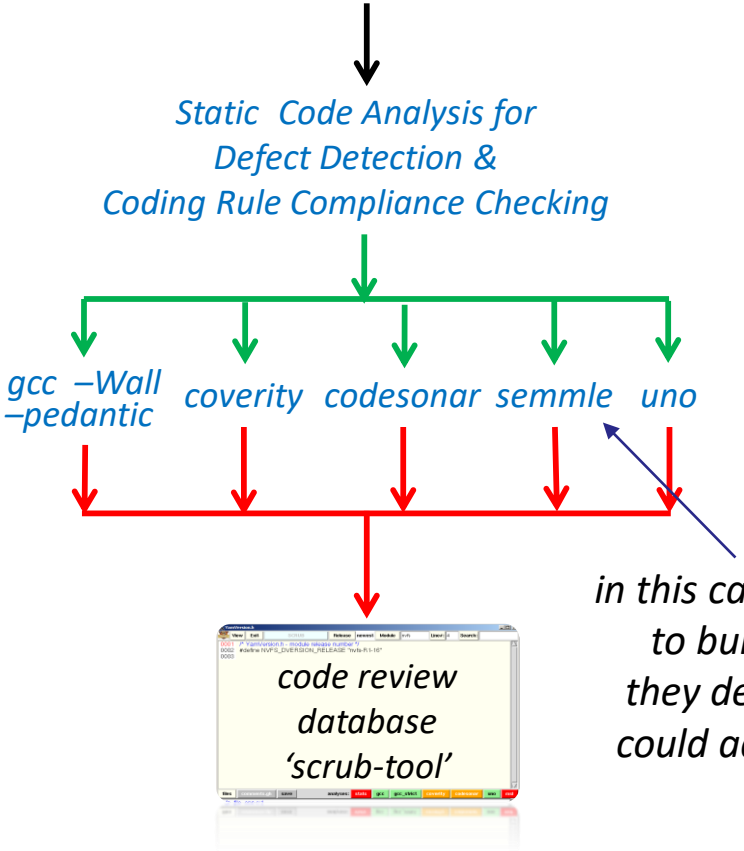
an example

- the MSL mission has ~2.8 Million lines of flight code
  - more than *all* previous missions to Mars combined
- a software anomaly occurred in the cruise phase
  - manual analysis revealed:
    - a function call passed a bad array argument
    - the function expected an array of *16* elements
    - the caller passed an array of *8* elements
    - *data corruption* resulted (compilers don't catch this)
- *Q:* does this happen anywhere else in the code?
  - old method: develop new checkers for a static analyzer
  - wait hours for the static analysis to be completed…
  - meanwhile, a few million miles away…..

MSL's code review process

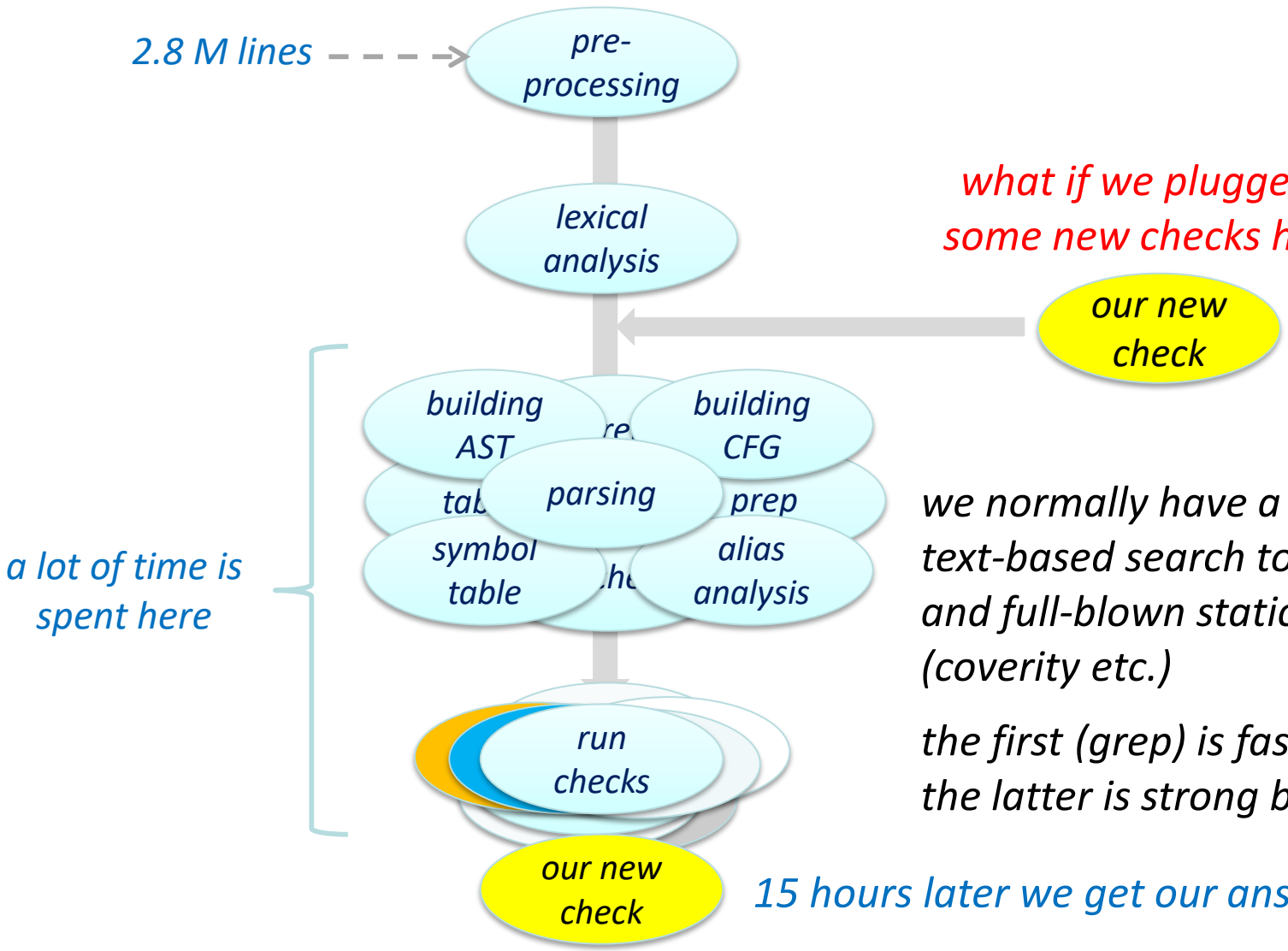
Nightly Build Log, 2.8 M lines
~3K compiler calls extracted

analysis time
~15hrs



in this case, we asked Semmle researchers to build a new checker for us, which they delivered the next day, so that we could add it to the nightly build & check

why does the analysis take so long?



what if we plugged in some new checks here?

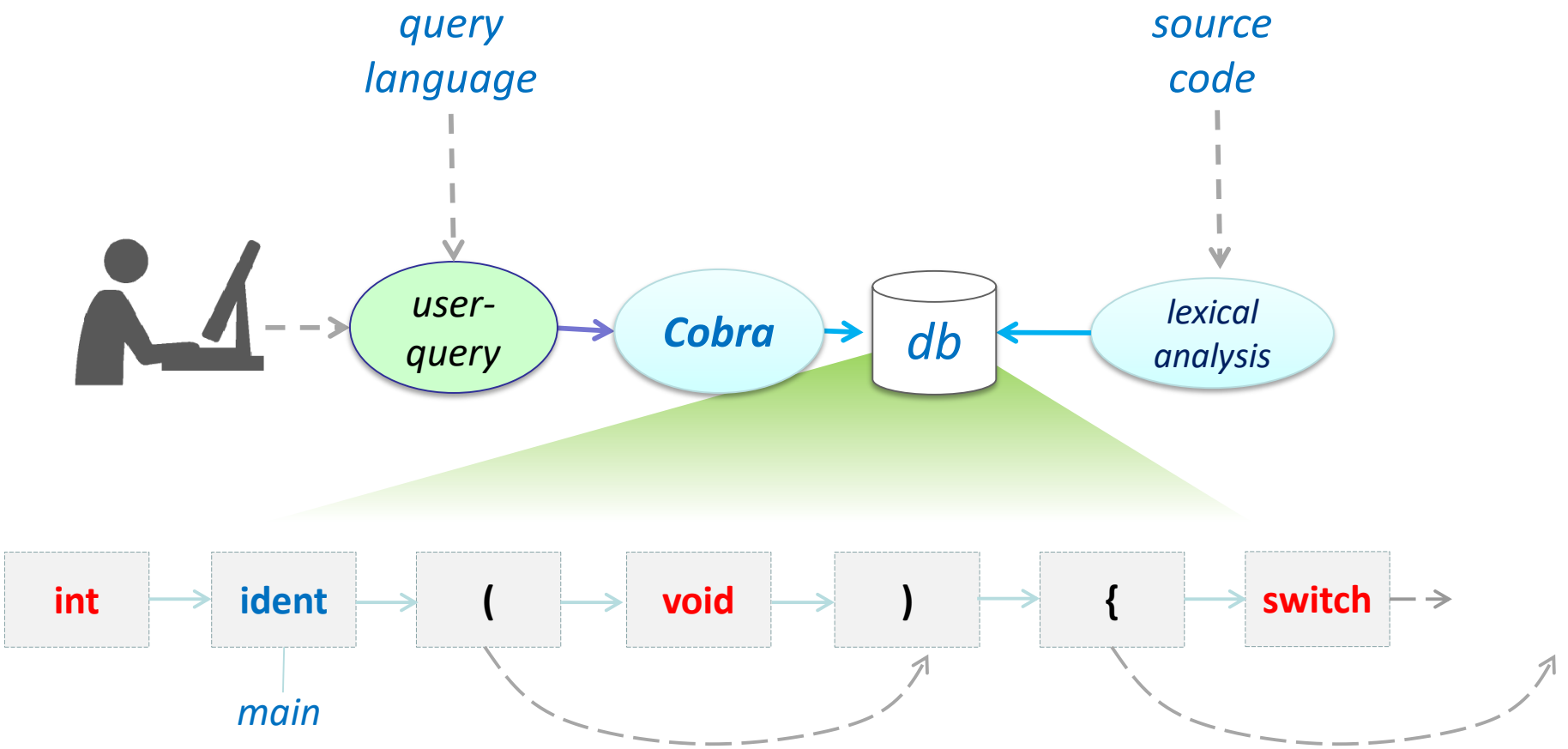
a lot of time is spent here

we normally have a choice between text-based search tools (grep etc.) and full-blown static analyzers (coverity etc.)

the first (grep) is fast but too weak
the latter is strong but too slow

15 hours later we get our answer….

interactive source code analysis



a linked list of lexical tokens with annotations

(token types, ranges, levels of nesting for parentheses, brackets, and braces, etc.)

this can give us super-fast pattern matching on tokenized source code

- token matching:

```
$ cobra –pat j *.c
```

*(compare with grep –e j *.c)*

- pattern matching:

```
$ cobra –pat 'switch ( .* ) { ^default* }' *.c
```

(find switch statements without default clause)

- name binding:

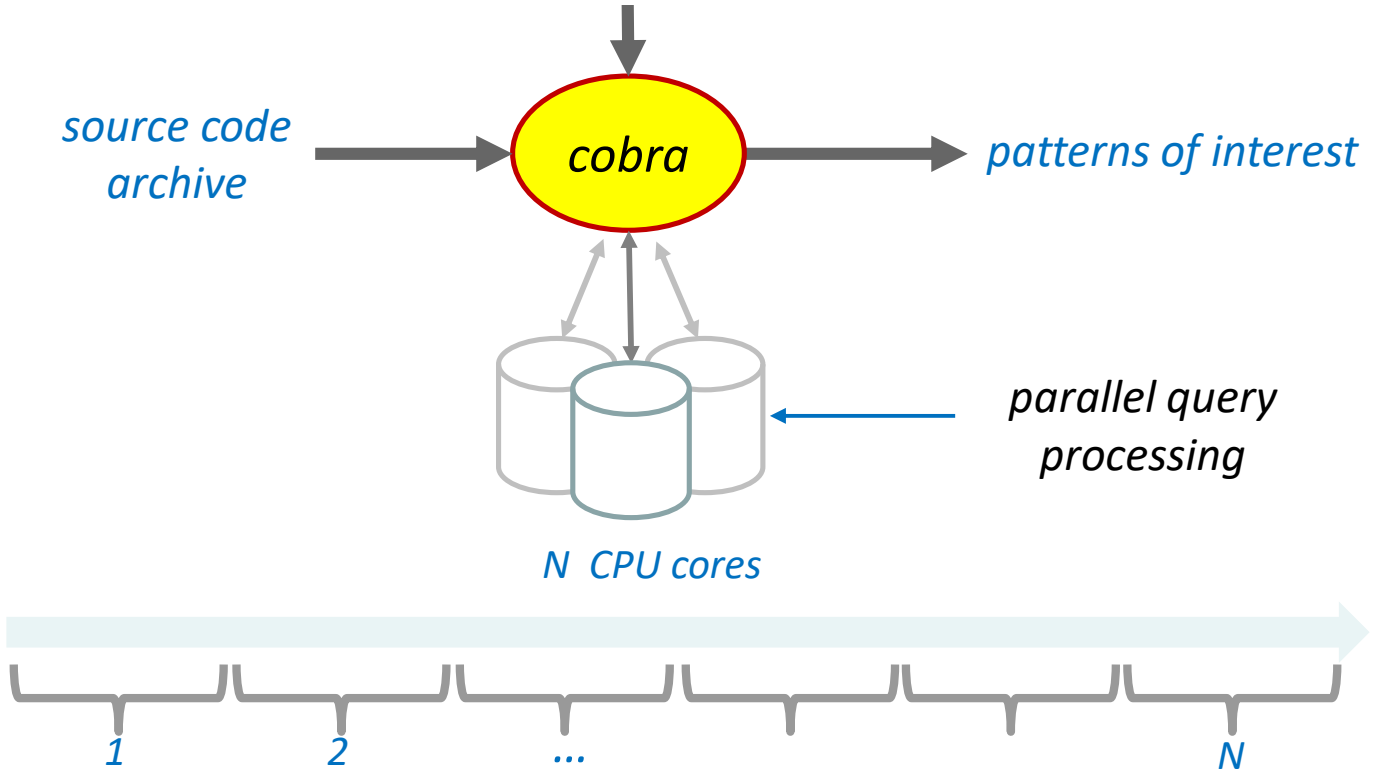
```
$ cobra –pat '{ .* @type x:@ident ^:x* }' *.c
```

(find redundant variable declarations)

interactively querying large code archives

lots of different query methods:

- *pattern matching on lexical tokens*
- *interactive queries (using sets, ranges, regular expressions)*
- *inline programs (+ recursive functions and associative arrays)*
- *standalone compiled checkers linked to the Cobra front-end*



quickly finding vulnerabilities

Undefined Behavior

```
$ cobra -pat '[-- ++] ^[, ;]* [-- ++]' *.c
```

sample match in a safety-critical code base:

```
val = (j++ << 16) | j++;
```

Code Injection & Remote Code Execution

```
$ cobra -pat 'x:@ident += snprintf ( ^,* :x .* /%s .* )' *.c
```

sample match in rsyslog/librelp/src/tcp.c:1216-1217 (version 1.2.14)

```
iAllNames += snprintf(allNames+iAllNames, sizeof(allNames)-iAllNames,
    "DNSname: %s; ", szAltName);
```

Suspicious Coding Patterns (likely bugs)

```
$ cobra -pat 'for ( .* ; .* [< <=] .* ; .* ^[++ +=] )' *.c
```

sample match in the Linux 4.3 distribution (18.6 M lines of code):

```
timeconv.c:120:   for (y = 11; days < ip[y]; y--)
```

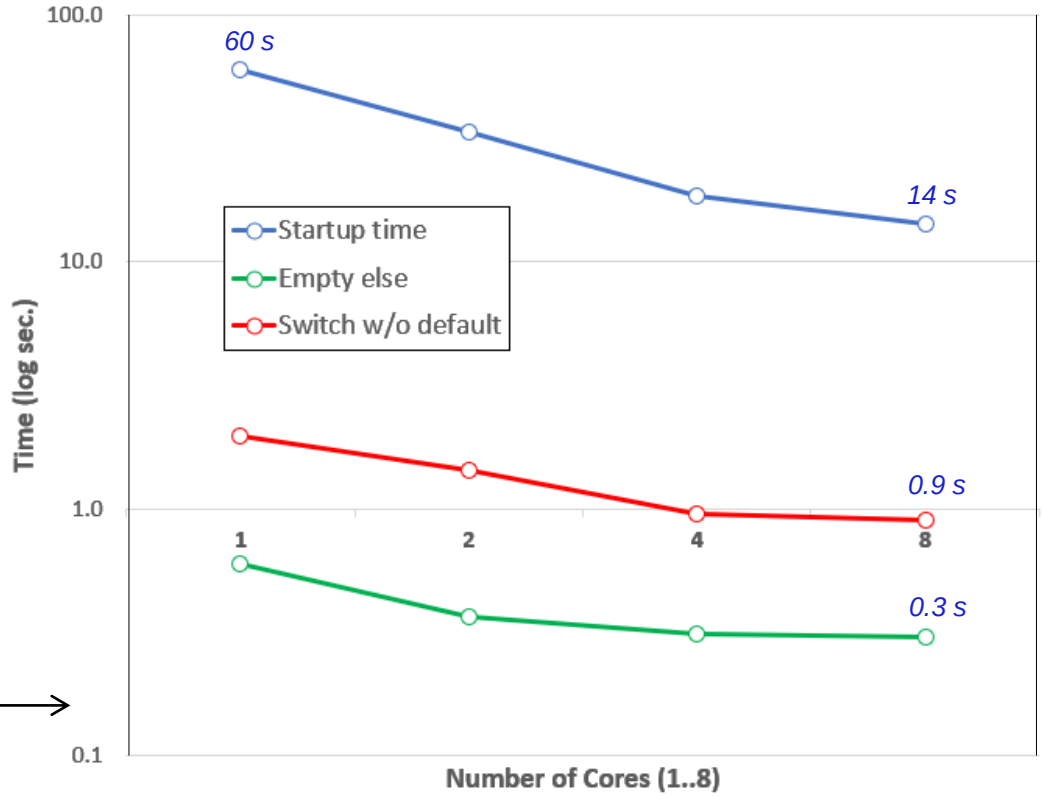
interactive query processing using multiple cores on a standard desktop

18,633,817 lines of code from the Linux 4.3 distribution, (39,144 .c and .h files)

checking 2 types of queries:

- *find empty else stmnts*
- *find all switch stmnts without default clause*

using 1..8 CPU cores



when using 4 or more cores:
query processing ≤ 1 sec.

move fast and find things!

manual pages, tutorials, papers:
http://www.spinroot.com/cobra

source code, query libraries, binaries:
https://github.com/nimble-code/Cobra