Cross Translational Unit Analysis in Clang Static Analyzer: Prototype and Measurements

Gabor Horvath (xazax[1]), Peter Szecsi (ps95[1]), Zoltan Gera (gerazo[1])
Daniel Krupp (daniel.krupp[2]), Zoltan Porkolab (zoltan.porkolab[2])

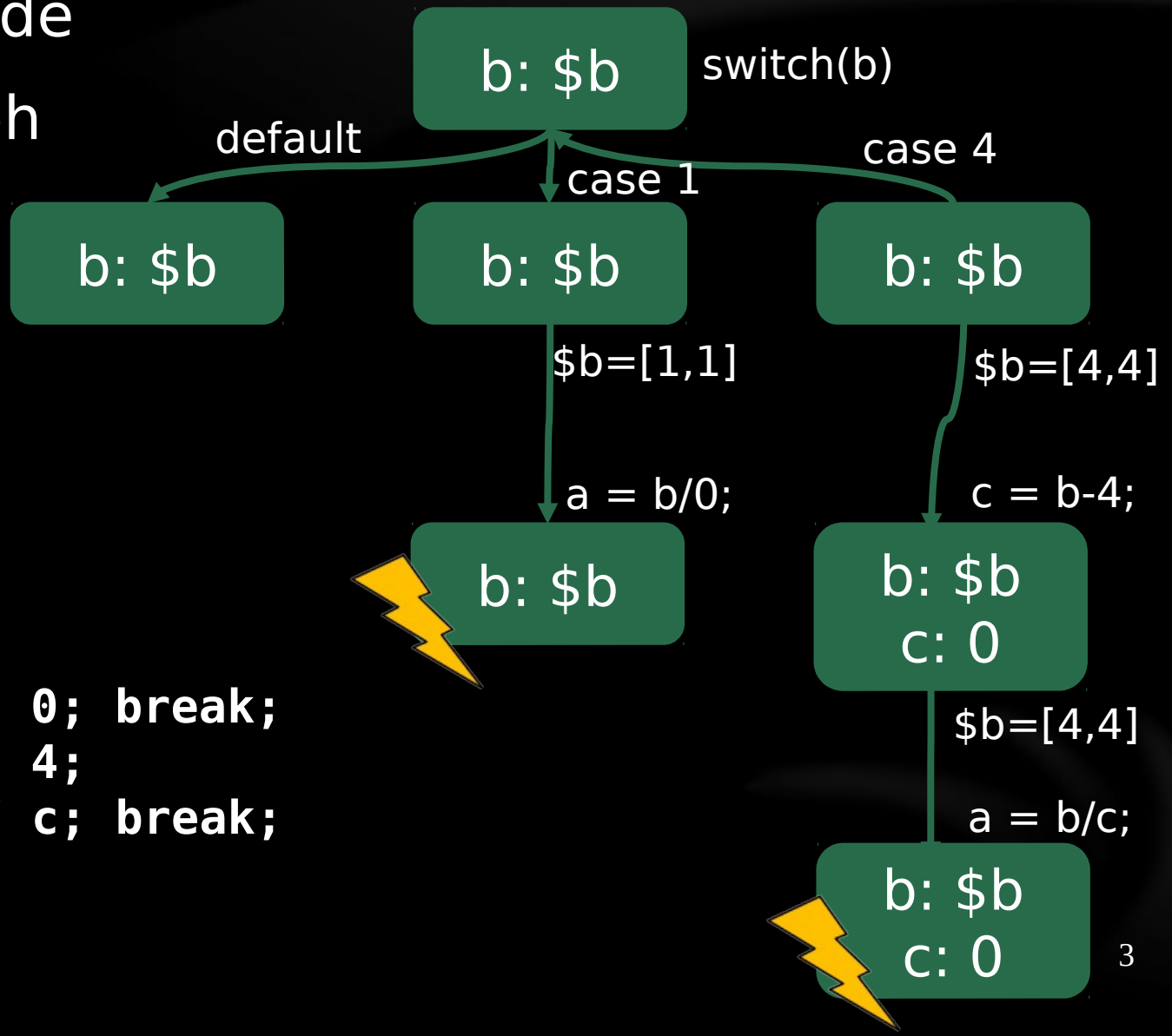
[1] @caesar.elte.hu
[2] @ericsson.com

Outline

- Motivation
- Overview of the Cross Translation Unit Analysis architecture
- Evaluation on open source projects
  - Findings
  - Performance
- Design questions
  - How to organize CTU related code
  - What to reanalyze, how to scale
- Future work

Clang Static Analyzer – Symbolic Execution

- Find bugs without running the code
- Exploded Graph




```
void test(int b) {
  int a,c;
  switch (b){
    case 1: a = b / 0; break;
    case 4: c = b - 4;
            a = b / c; break;
  }
}
```

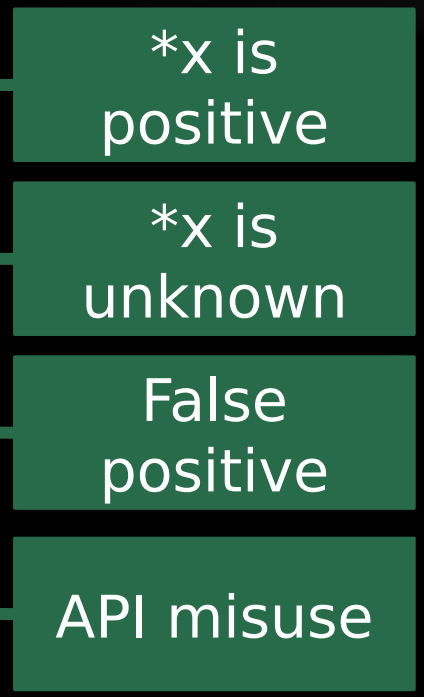
Motivation

A.cpp

```
void neg(int *x);
void g(int *x) {
  if (*x > 0)
    neg(x);
  if (*x > 0)
    *x / 0;
  neg(NULL);
}
```

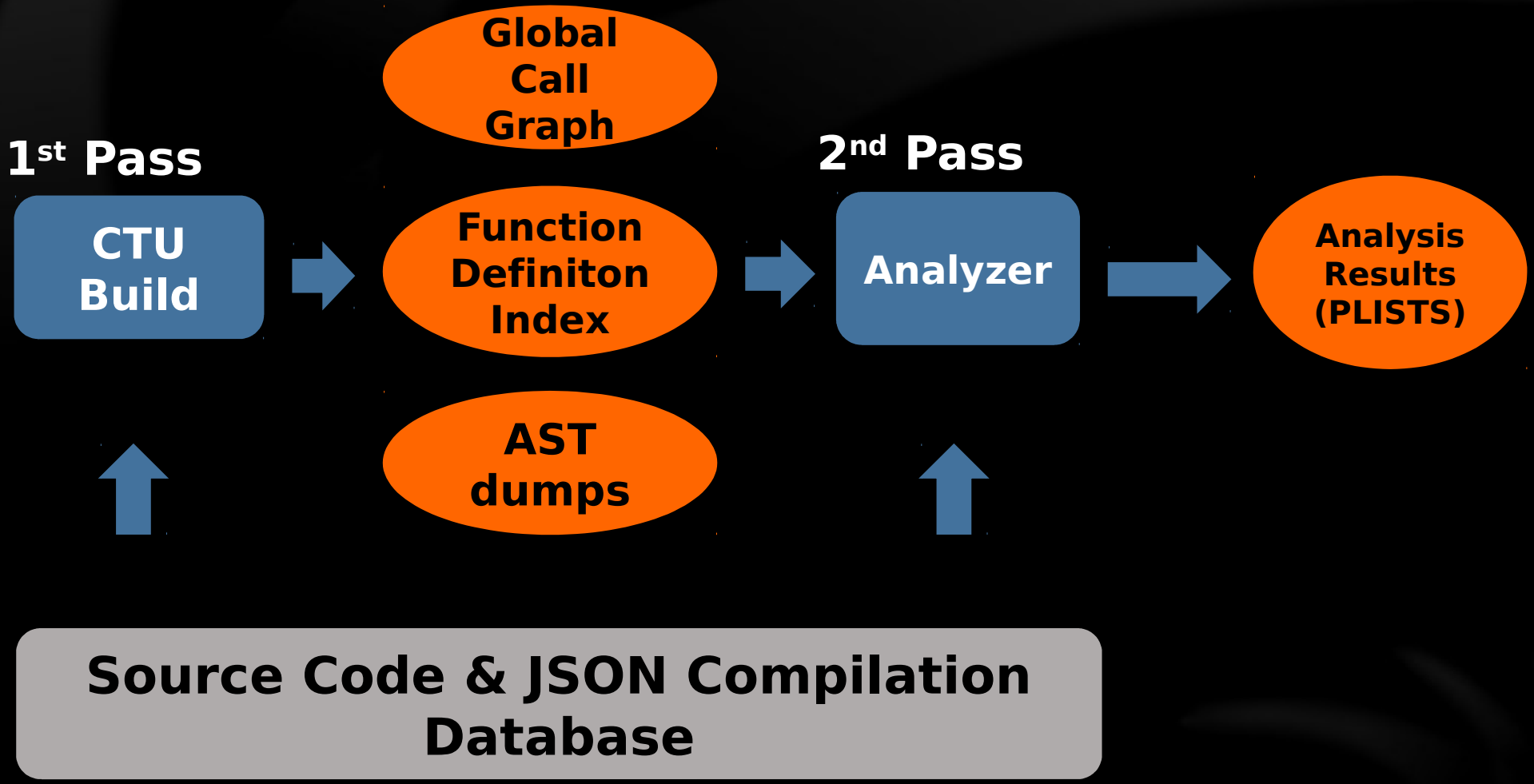
B.cpp

```
void neg(int *x) {
   *x = -(*x);
}
```



- We saw useful CTU results reported by closed source analysis tools
- Can we achieve the same using Clang SA?

High Level Architecture



Evaluation

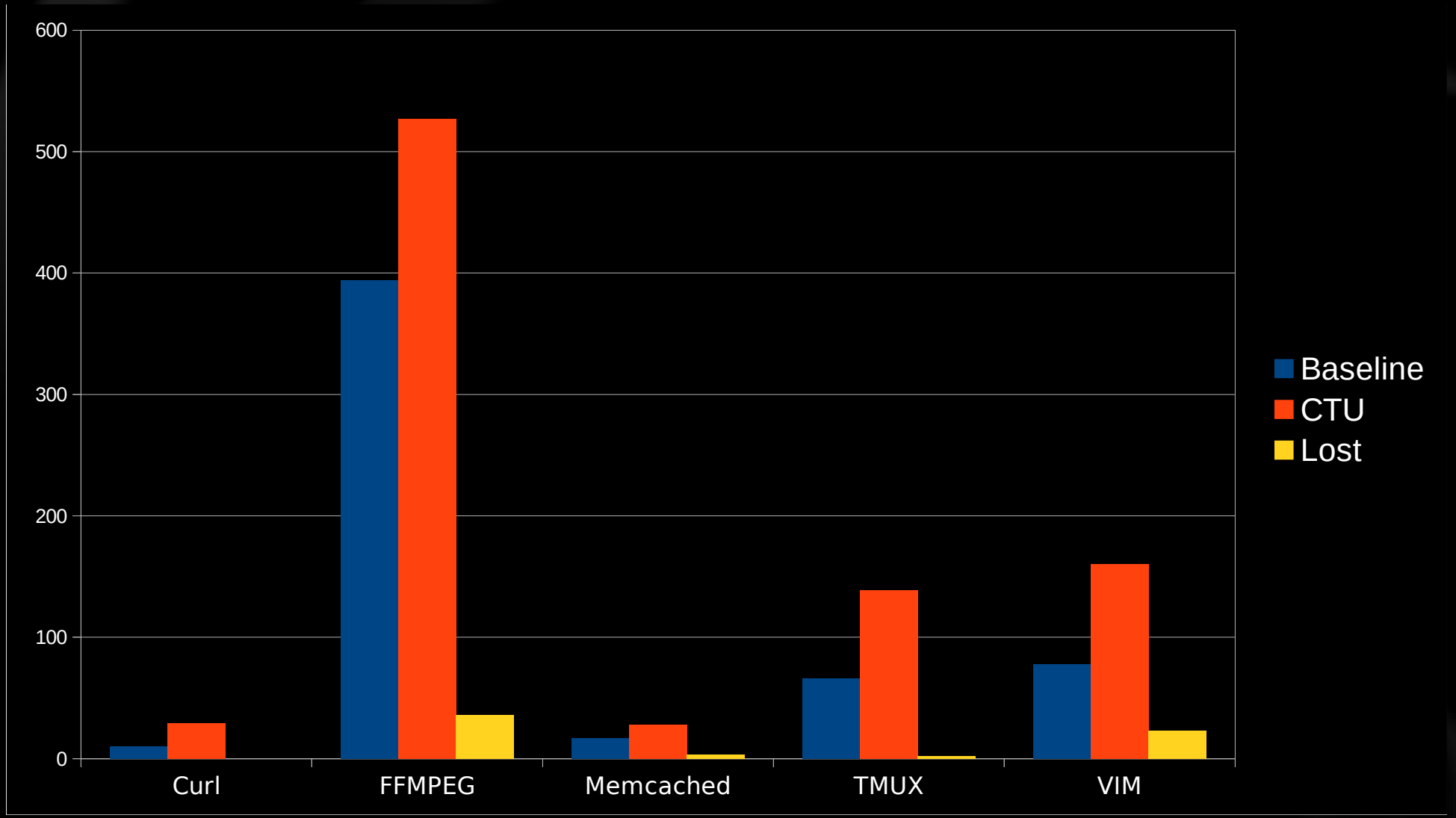
- Open source C projects:
  - OpenSSL, Curl, Vim, Memcached, ffmpeg, PostgreSQL, ...
  - Full details at: http://cc.inf.elte.hu
  - Improvements needed for C++ support
- Metrics:
  - Number of new bugs reported
  - Number of lost bug reports
  - Quality of new bug reports
  - Analysis time
  - Peak memory usage (per process)

Run ID (project + timestamp)	Total files of project	Files XTU Passed	Files XTU Failed	Time of XTU (sec)	Time of noXTU (sec)	Time of XTU-BUILD (sec)	Max heap usage of NOXTU (B)	Max Heap usage of XTU (B))	Analysis Coverage
curl_2017-03-23_23:50:26	293	293	0	644.54	362.41	8.37	97912954	243880678	No XTU Coverage XTU Coverage Diff Coverage
ffmpeg_2017-03-24_00:10:12	1563	1559	4	6260.98	2941.50	57.89	157030682	202068780	No XTU Coverage XTU Coverage Diff Coverage
memcached_2017-03-24_03:10:12	35	35	0	372.16	230.57	2.20	72261684	95777660	No XTU Coverage XTU Coverage Diff Coverage
nginx_2017-03-24_03:21:37	114	114	0	766.14	336.93	5.28	81624439	181235414	No XTU Coverage XTU Coverage Diff Coverage

File	Lines			Branches	
lib/base64.c		90.7 %	107 / 118	100.0 %	0 / 0
lib/conncache.c		100.0 %	139 / 139	100.0 %	0 / 0
lib/connect.c		85.2 %	439 / 515	100.0 %	0 / 0
lib/content_encoding.c		100.0 %	160 / 160	100.0 %	0 / 0
lib/cookie.c		62.7 %	381 / 608	100.0 %	0 / 0
lib/curl_addrinfo.c		96.7 %	174 / 180	100.0 %	0 / 0
lib/curl_endian.c		100.0 %	32 / 32	100.0 %	0 / 0
lib/curl_fnmatch.c		91.0 %	213 / 234	100.0 %	0 / 0
lib/curl_gethostname.c		100.0 %	11 / 11	100.0 %	0 / 0
lib/curl_memrchr.c		100.0 %	8 / 8	100.0 %	0 / 0
lib/curl_ntlm_core.c		98.4 %	122 / 124	100.0 %	0 / 0
lib/curl_ntlm_wb.c		100.0 %	196 / 196	100.0 %	0 / 0
lib/curl_sasl.c		98.2 %	224 / 228	100.0 %	0 / 0
lib/dict.c		100.0 %	127 / 127	100.0 %	0 / 0
lib/dotdot.c		84.1 %	53 / 63	100.0 %	0 / 0
lib/easy.c		98.0 %	251 / 256	100.0 %	0 / 0
lib/escape.c		90.0 %	81 / 90	100.0 %	0 / 0
lib/file.c		100.0 %	234 / 234	100.0 %	0 / 0
lib/fileinfo.c		100.0 %	11 / 11	100.0 %	0 / 0
lib/formdata.c		66.4 %	417 / 628	100.0 %	0 / 0

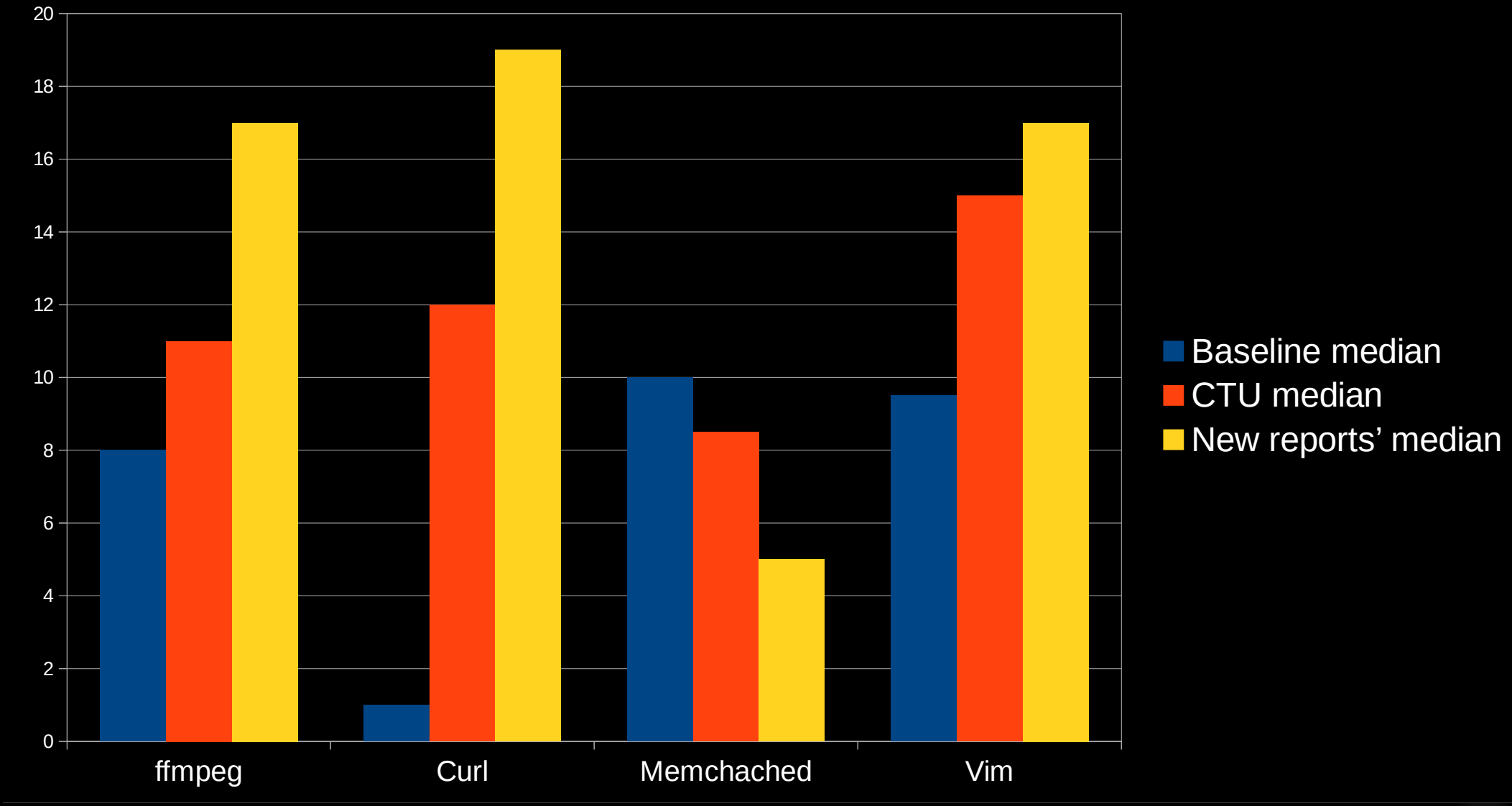
```
681     12      if(strncmp(lineptr, "#HttpOnly_", 10) == 0) {
682             lineptr += 10;
683             co->httponly = TRUE;
684             }
685
686     2       if(lineptr[0]=='#') {
687             /* don't even try the comments */
688             free(co);
689             return NULL;
690             }
691             /* strip off the possible end-of-line characters */
692     10      ptr=strchr(lineptr, '\r');
693     2       if(ptr)
694             *ptr=0; /* clear it */
695     10      ptr=strchr(lineptr, '\n');
696     2       if(ptr)
697             *ptr=0; /* clear it */
698
699     8       firstptr=strtok_r(lineptr, "\t", &tok_buf); /* tokenize it on the TAB */
700
701             /* Now loop through the fields and init the struct we already have
702                allocated */
703     6306    for(ptr=firstptr, fields=0; ptr && !badcookie;
704     17908       ptr=strtok_r(NULL, "\t", &tok_buf), fields++) {
705     820     switch(fields) {
706             case 0:
707     2         if(ptr[0]=='.') /* skip preceding dots */
708               ptr++;
709     2         co->domain = strdup(ptr);
710     2         if(!co->domain)
711               badcookie = TRUE;
712             break;
```

Evaluation – Bug Reports



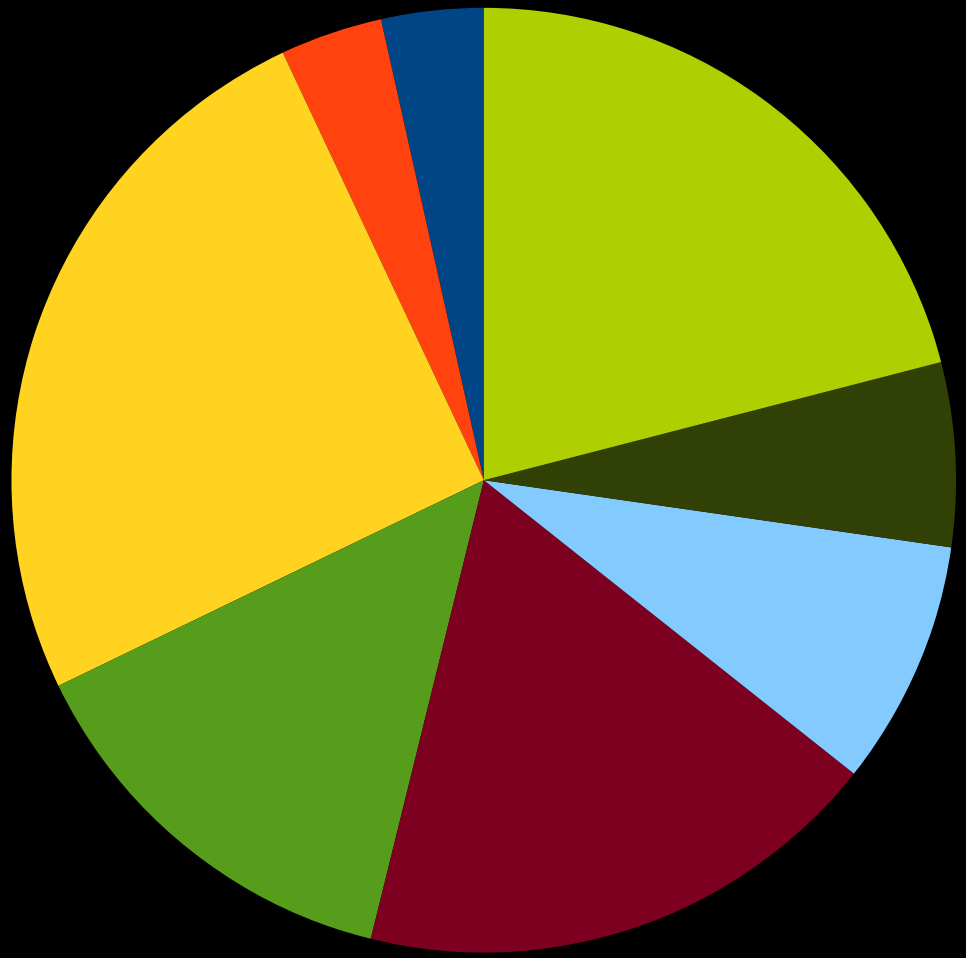
- 2.4X average, 2.1X median, 5X peak

Bug Path Length of Bug Reports



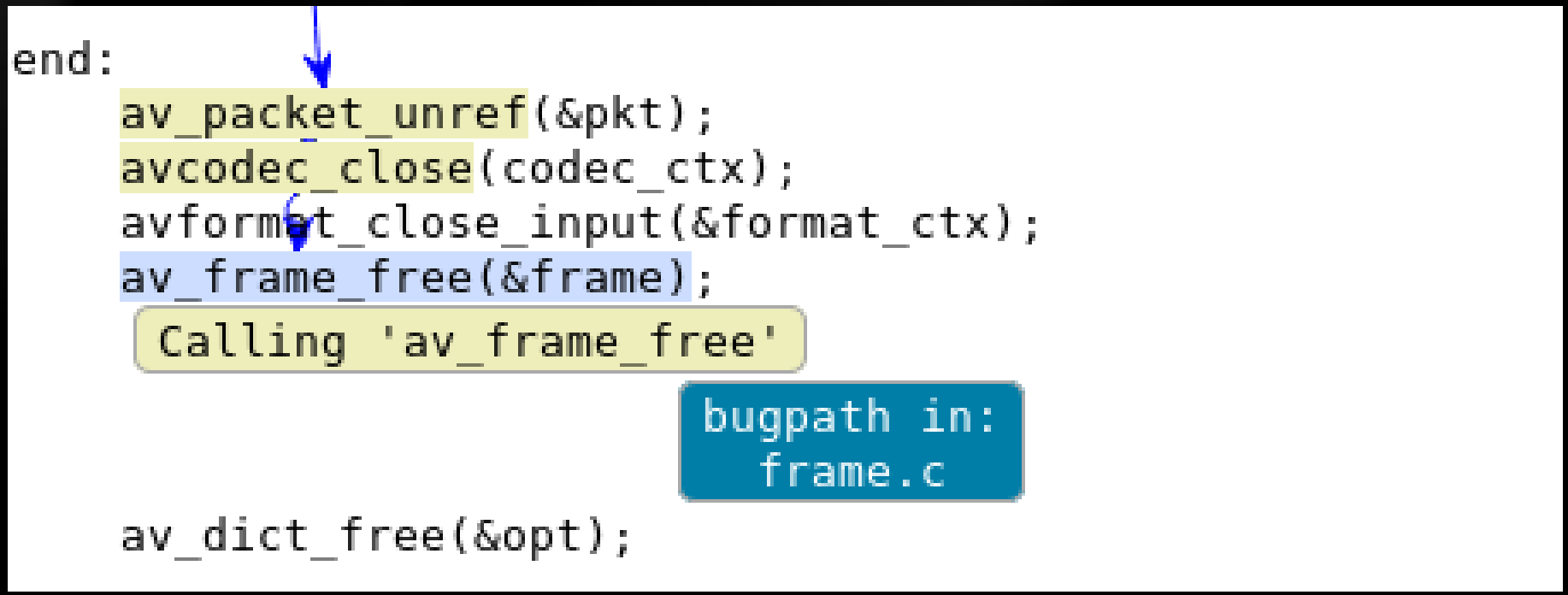
- The reason for false positives was never the CTU

FFMPEG - Quality of New Bug Reports



- True positive example: http://cc.inf.elte.hu:8080/#baseline=177&newcheck=178&report=17539
- One Definition Rule violation found

```
AVInputFormat *iformat = NULL;
AVFormatContext *format_ctx = NULL;
AVCodec *codec;
AVCodecContext *codec_ctx;
AVFrame *frame;
int frame_decoded, ret = 0;
AVPacket pkt;
AVDictionary *opt=NULL;

av_init_packet(&pkt);

av_register_all();

iformat = av_find_input_format("image2");
if ((ret = avformat_open_input(&format_ctx, filename, iformat, NULL)) < 0) {
    Assuming the condition is false
    av_log(log_ctx, AV_LOG_ERROR,
           "Failed to open input file '%s'\n", filename);
    return ret;
}

if ((ret = avformat_find_stream_info(format_ctx, NULL)) < 0) {
    Assuming the condition is false
    av_log(log_ctx, AV_LOG_ERROR, "Find stream info failed\n");
    return ret;
}

codec_ctx = format_ctx->streams[0]->codec;
codec = avcodec_find_decoder(codec_ctx->codec_id);
if (!codec) {
    av_log(log_ctx, AV_LOG_ERROR, "Failed to find codec\n");
    ret = AVERROR(EINVAL);
    goto end;
}

av_dict_set(&opt, "thread_type", "slice", 0);
if ((ret = avcodec_open2(codec_ctx, codec, &opt)) < 0) {
    av_log(log_ctx, AV_LOG_ERROR, "Failed to open codec\n");
    goto end;
}

if (!(frame = av_frame_alloc()) ) {
```


end:
av_packet_unref(&pkt);
avcodec_close(codec_ctx);
avformat_close_input(&format_ctx);
av_frame_free(&frame);
Calling 'av_frame_free'
bugpath in:
frame.c
av_dict_free(&opt);

Same bug multiple times?

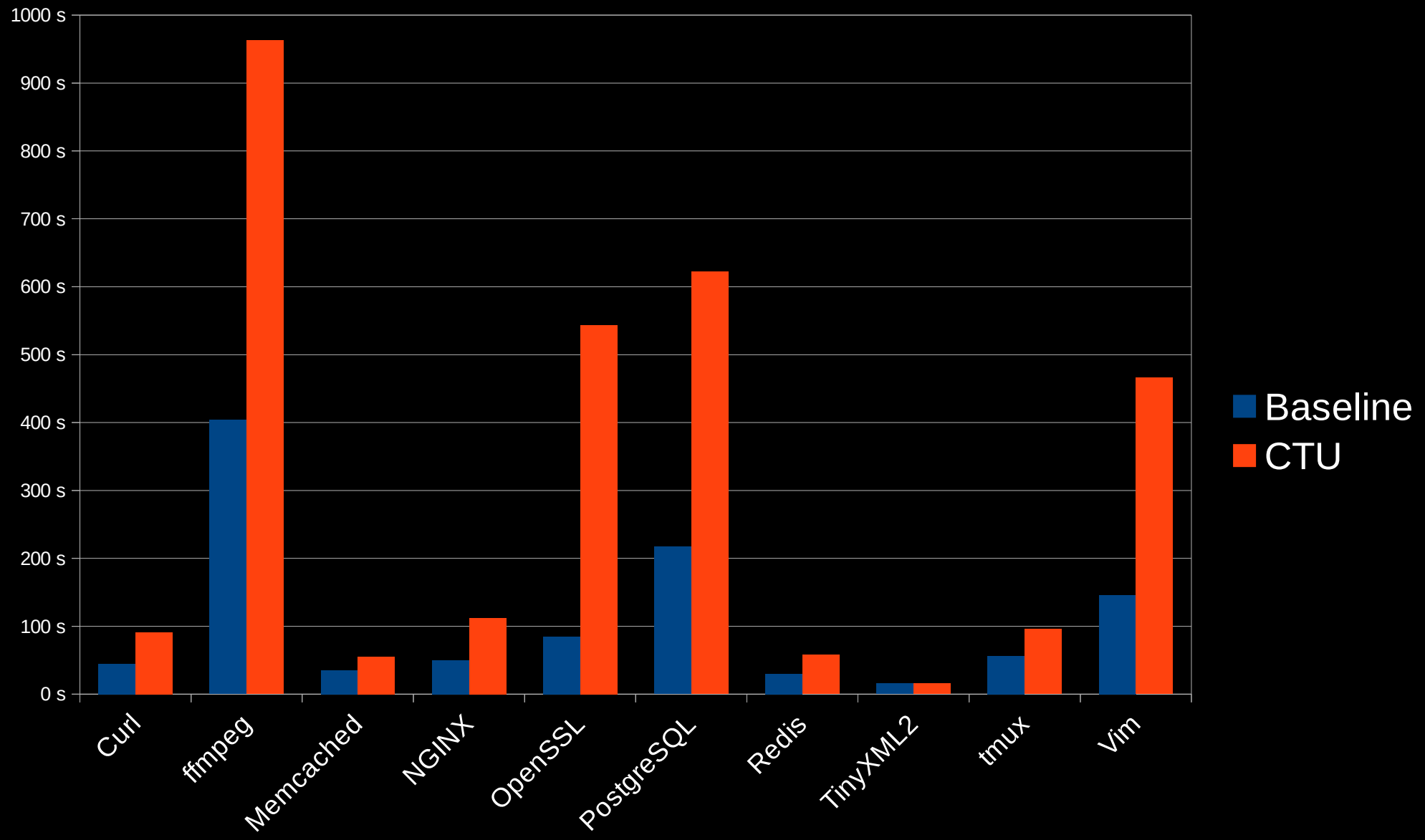
A.cpp

```
void neg(int *x);
void g(int *x) {
  ...
  neg(NULL);
  ...
}
void h(int *x) {
  ...
  neg(NULL);
  ...
}
```

B.cpp

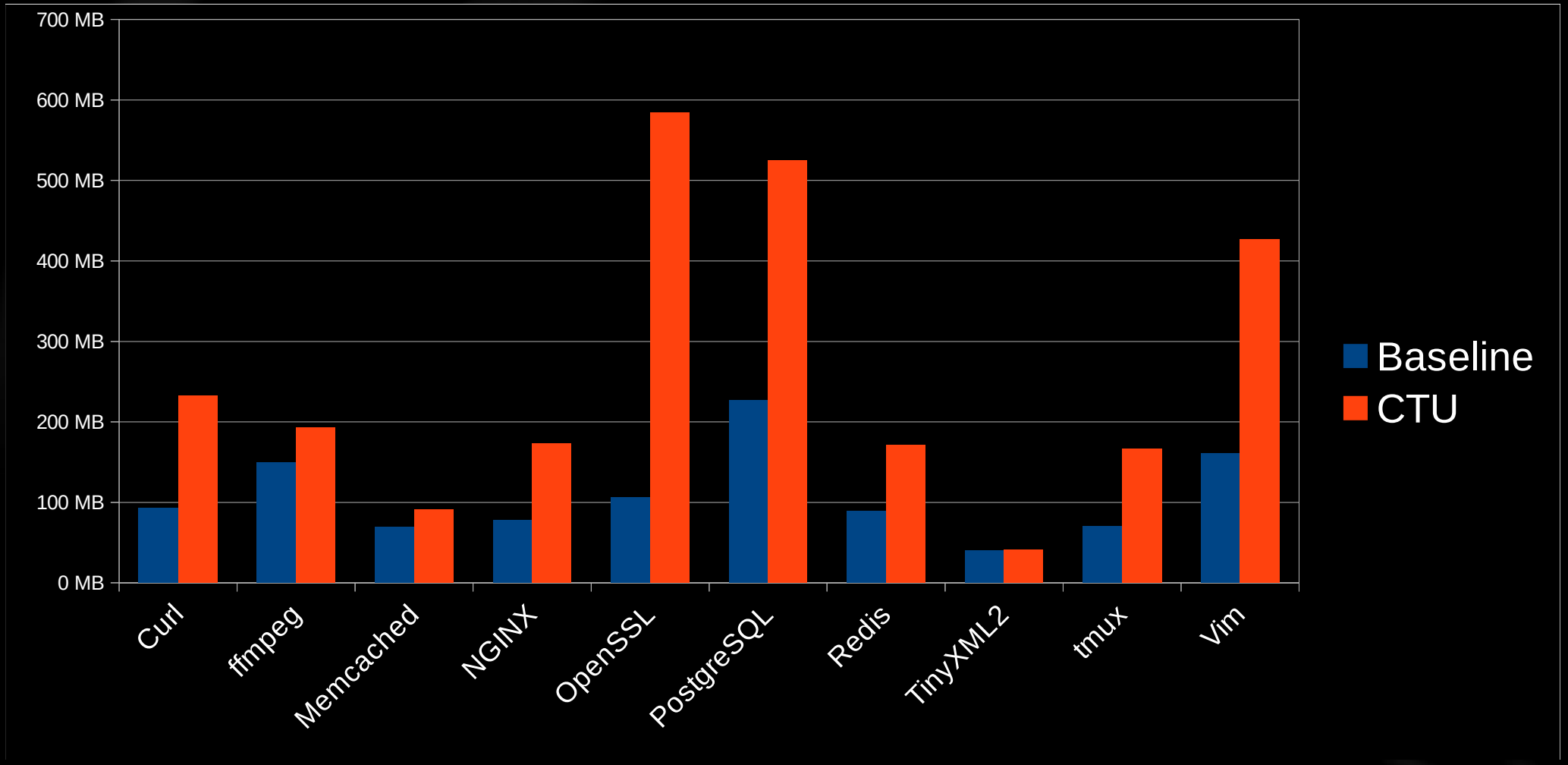
```
void neg(int *x) {
   *x = -(*x);
}
```

Evaluation – Analysis time



- 2.5X average, 2.1X median, 6.4X peak

Evaluation - Memory



- 2.3X average, 2.3X median, 5.5X peak
- AST dumps consume disk space temporarily
  - ~40GB for LLVM

Current Implementation

- Artem Dergachev, Aleksei Sidorin, et al.
  - Prototype: both for naive CTU and summary based interprocedural analysis, based on Clang 3.4
  - http://lists.llvm.org/pipermail/cfe-dev/2015-October/045730.html
- Improved version contributed by Ericsson, only contains the CTU part, ready for review
  - https://reviews.llvm.org/D30691
  - Patch is relatively small, CTU off by default
  - **No changes required to checker implementations**

Order of the Analysis of Functions

A.cpp

```
void f(int x);
void g(int x) {
  f(x);
}
void h(int x) {
  g(x);
}
```

B.cpp

```
void i(int x) {
}
void f(int x) {
  i(x);
}
```

Incrementality

A.cpp

```
void f(int *x);
void g(int *x) {
  f(NULL);
}
```

B.cpp

```
void f(int *x) {
   *x = -(*x);
}
```

Incrementality

A.cpp

```
void f(int *x);
void g(int *x) {
  f(NULL);
}
```

B.cpp

```
void f(int *x) {
  *x = -(*x);
  if (x == 0)
    return;
}
```

- **We need to reanalyze A.cpp too**

AST Importer

- Import (merge) one AST into another
- Can import one function/type a time
- Caches the results to avoid multiple imports
- Used by LLDB
- Not a mature component of Clang yet

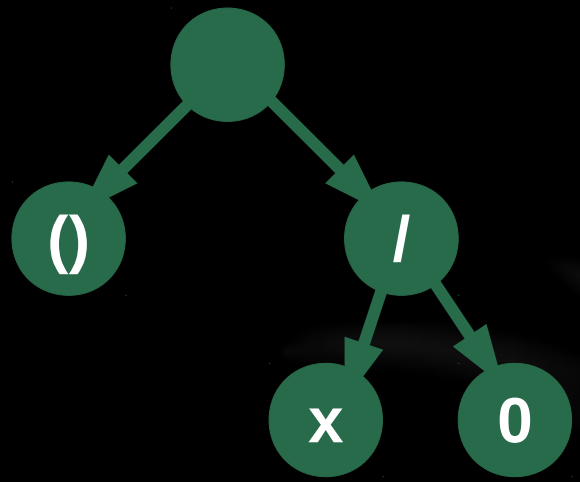
AST Importer

- Issues with importing source locations from macros
- Suboptimal results for C++ projects
  - We concentrated on C projects
  - Fixed C related bugs in the importer
- The analysis can find AST Importer bugs
  - Running analysis on the imported AST can trigger asserts
  - Found invariant violations on imported AST that otherwise very challenging to write a test for

Coverage

- Increased for some files
  - Functions evaluated in more contexts
- Decreased for others
  - Analysis budget runs out due to DFS
  - Prune more infeasible paths
  - More issues reported implies stops
- Small overall decrease

```
void external(int x);
void g(int x) {
  external(x);
  x / 0;
}
```

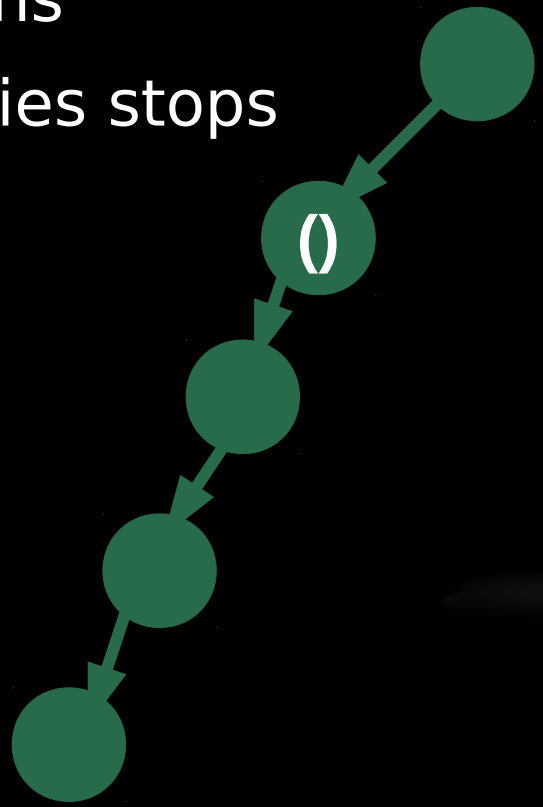


Coverage

- Increased for some files
  - Functions evaluated in more contexts
- Decreased for others
  - Analysis budget runs out due to DFS
  - Prune more infeasible paths
  - More issues reported implies stops
- Small overall decrease

```
void external(int x);
void g(int x) {
  external(x);
  x / 0;
}
```

Might exhaust budget



Getting Started

- Run CTU on your project if interested in additional results
- Run both CTU and non-CTU to get maximal coverage
- Give us feedback about the quality of reports
  - Analysis errors
  - True positives
  - False positives
- CodeChecker supports viewing CTU results!
  - https://github.com/Ericsson/codechecker

Future Work

- Extend the C++ support of ASTImporter
- New strategies to build an exploded graph with good shape?
- Tune default budget for CTU
- Incremental CTU analysis
- Make the binary AST dumps smaller

- Grouping of bug paths in viewers (CodeChecker, XCode, ...)

Summary

- Improved the CTU prototype
- Evaluated the results on open source projects
  - CTU found many new potential bugs
  - Analysis time scales well with CPUs
  - Bug/time, bug/memory ratio is good
  - Coverage, quality of reports satisfying
- Works well for C programs
- Improvements needed for C++
- Prepared a patch for upstreaming

Thank you! Questions?