ΟΙΚΟΝΟΜΙΚΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΑΘΗΝΩΝ

ATHENS UNIVERSITY
OF ECONOMICS
AND BUSINESS

TU Delft

Macros Gone Wild:
The Usage of the C Preprocessor in the Linux Kernel

Diomidis Spinellis

Department of Management Science and Technology
Athens University of Economics and Business

Department of Software Technology
Delft University of Technology

www.spinellis.gr

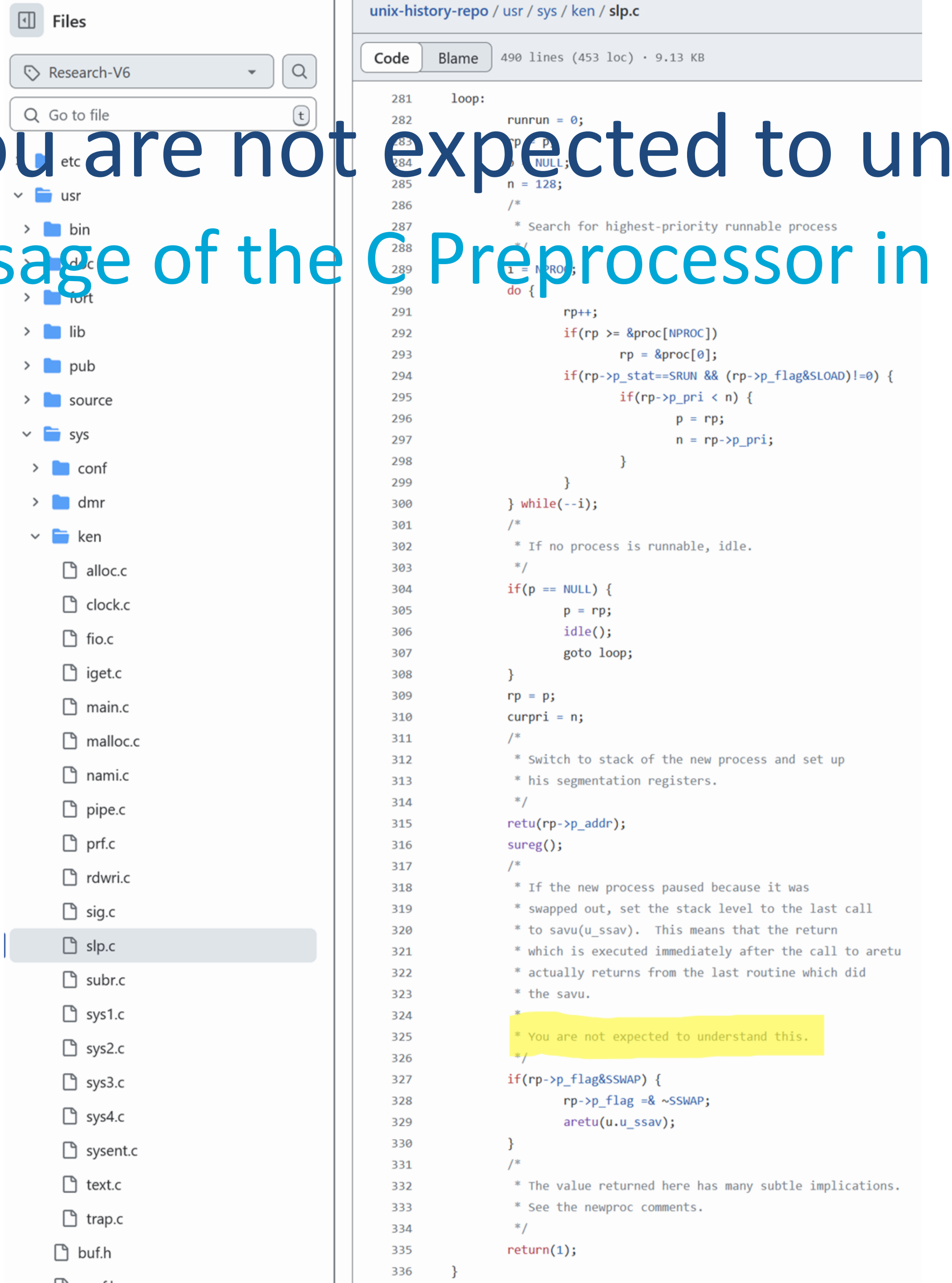
🦋 𝕏 @CoolSWEng

@CoolSWEng@mastodon.acm.org

Business
Analytics Lab

You are not expected to understand this:

The Usage of the C Preprocessor in the Linux Kernel



```
loop:
        runrun = 0;
        rp = p;
        p = NULL;
        n = 128;
        /*
         * Search for highest-priority runnable process
         */
        i = NPROC;
        do {
                rp++;
                if(rp >= &proc[NPROC])
                        rp = &proc[0];
                if(rp->p_stat==SRUN && (rp->p_flag&SLOAD)!=0) {
                        if(rp->p_pri < n) {
                                p = rp;
                                n = rp->p_pri;
                        }
                }
        } while(--i);
        /*
         * If no process is runnable, idle.
         */
        if(p == NULL) {
                p = rp;
                idle();
                goto loop;
        }
        rp = p;
        curpri = n;
        /*
         * Switch to stack of the new process and set up
         * his segmentation registers.
         */
        retu(rp->p_addr);
        sureg();
        /*
         * If the new process paused because it was
         * swapped out, set the stack level to the last call
         * to savu(u_ssav).  This means that the return
         * which is executed immediately after the call to aretu
         * actually returns from the last routine which did
         * the savu.
         *
         * You are not expected to understand this.
         */
        if(rp->p_flag&SSWAP) {
                rp->p_flag =& ~SSWAP;
                aretu(u.u_ssav);
        }
        /*
         * The value returned here has many subtle implications.
         * See the newproc comments.
         */
        return(1);
}
```

C Preprocessor 101

C preprocessor: Source file inclusion

```c
#include <linux/irq.h>
#include <linux/delay.h>
#include <linux/property.h>
#include <linux/spi/spi.h>
#include <linux/regmap.h>
#include <linux/skbuff.h>
#include <linux/ieee802154.h>

#include <net/mac802154.h>
#include <net/cfg802154.h>

#include "at86rf230.h"
```

C preprocessor: Macro replacement

```
#define KB         1024
#define MB         (1024*KB)
#define GB         (1024*MB)

if (block_size != (16 * GB))

#define FL_BASE_MASK         0x0007
#define FL_GET_BASE(x)       (x & FL_BASE_MASK)

bar = FL_GET_BASE(board->flags);
```

C preprocessor: conditional compilation

```
#ifdef CONFIG_KEYS
        .keyring_name_list = LIST_HEAD_INIT(init_user_ns.keyring_name_list),
        .keyring_sem = __RWSEM_INITIALIZER(init_user_ns.keyring_sem),
#endif

#ifdef CONFIG_ARCH_USES_CFI_TRAPS
static inline unsigned long trap_address(s32 *p)
{
        return (unsigned long)((long)p + (long)*p);
}
[…]
#endif
```

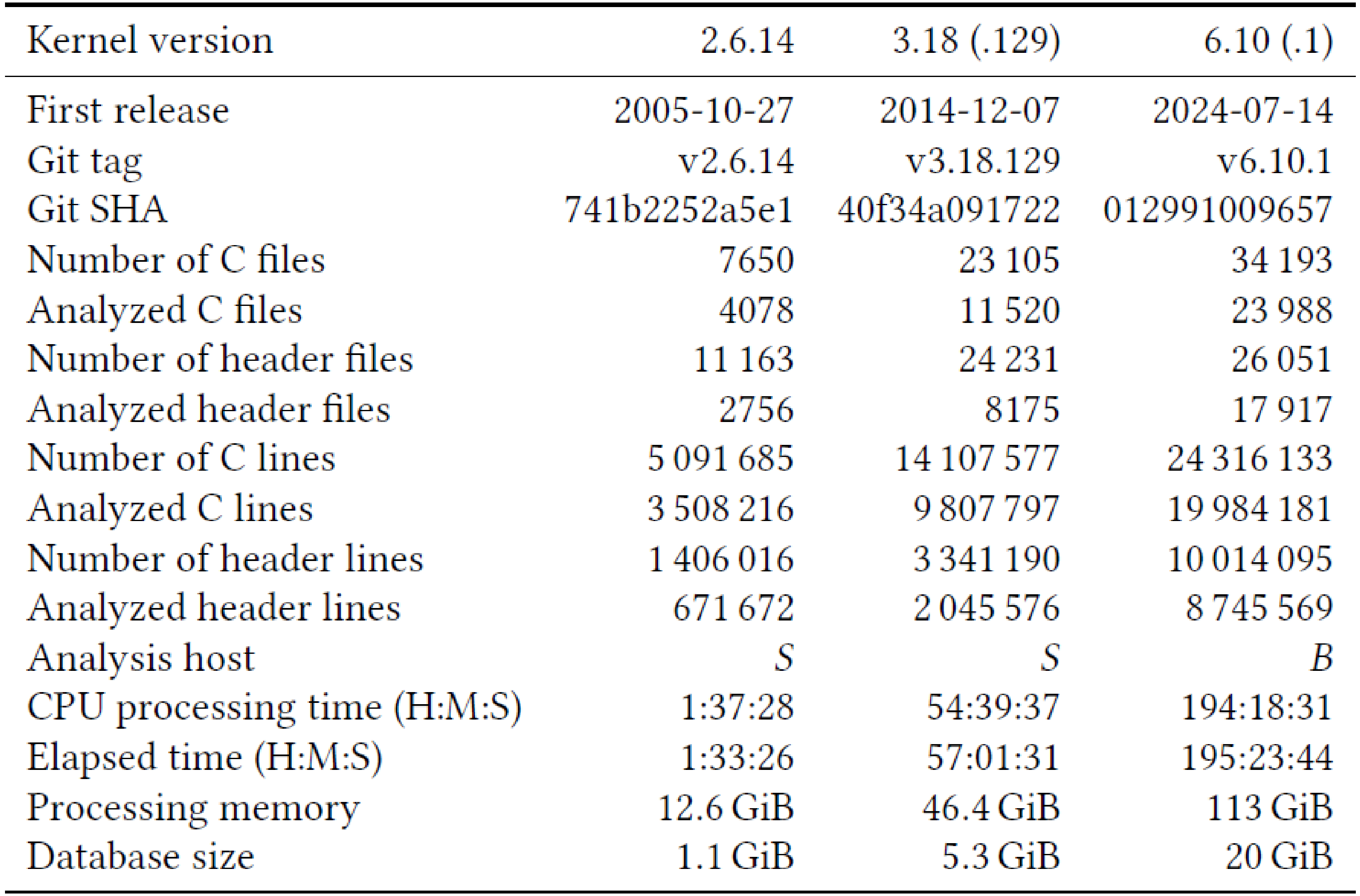
Outline

1. Preprocessor’s usage characteristics
2. Introduced technical debt
3. Usage evolution
4. Feasibility of reducing incurred technical debt (esp. via Rust)

CScout and its extensions

- Refactoring browser for C code
- Performs semantic & syntactic analysis of C code, taking into account the C preprocessor
- Extended to
  - Collect pre/post expansion metrics
  - At the level of functions & files
  - Size, keywords, Halstead volume, cyclomatic complexity

Linux kernel analysis



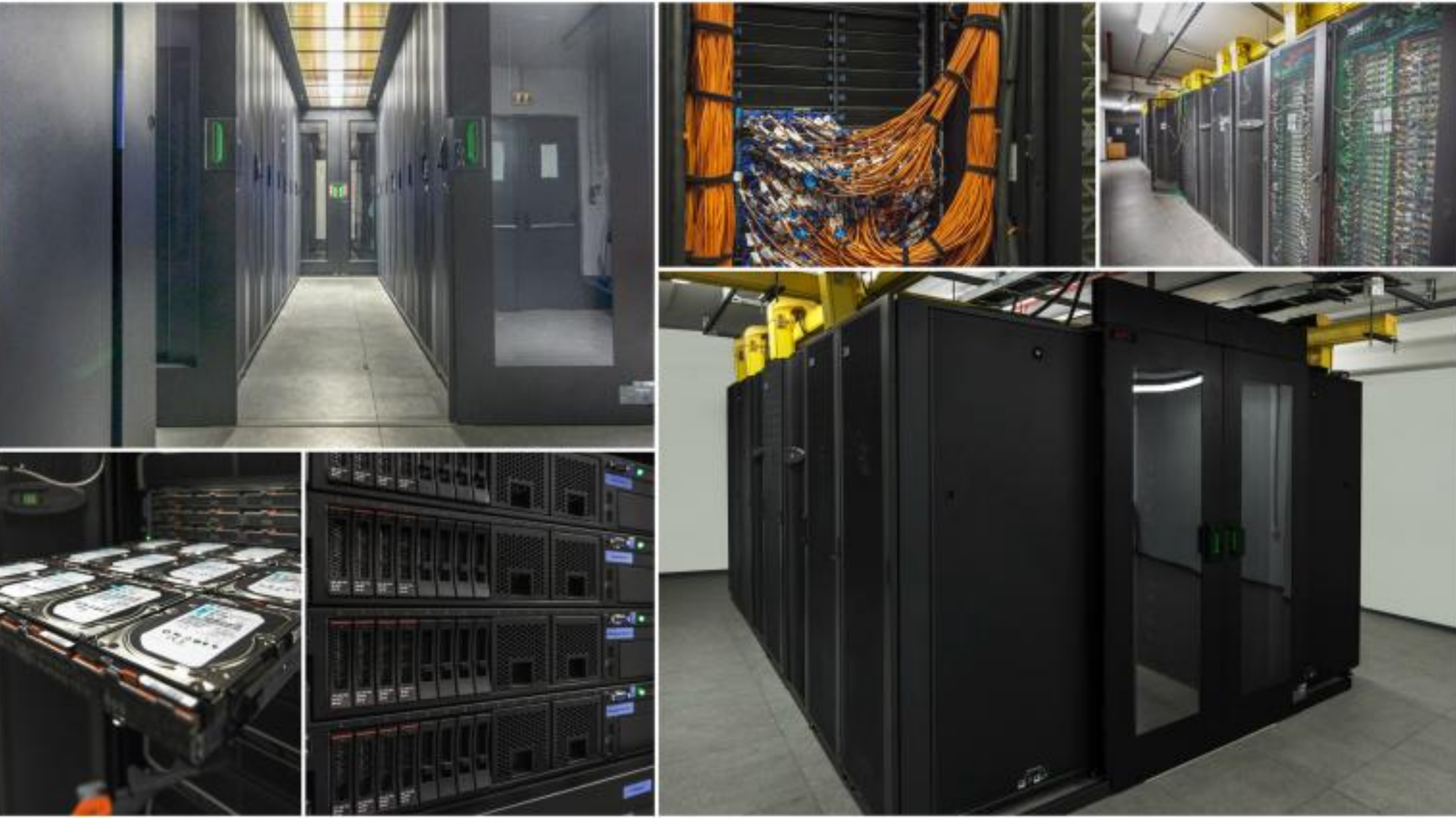
Kernel version	2.6.14	3.18 (.129)	6.10 (.1)
First release	2005-10-27	2014-12-07	2024-07-14
Git tag	v2.6.14	v3.18.129	v6.10.1
Git SHA	741b2252a5e1	40f34a091722	012991009657
Number of C files	7650	23 105	34 193
Analyzed C files	4078	11 520	23 988
Number of header files	11 163	24 231	26 051
Analyzed header files	2756	8175	17 917
Number of C lines	5 091 685	14 107 577	24 316 133
Analyzed C lines	3 508 216	9 807 797	19 984 181
Number of header lines	1 406 016	3 341 190	10 014 095
Analyzed header lines	671 672	2 045 576	8 745 569
Analysis host	*S*	*S*	*B*
CPU processing time (H:M:S)	1:37:28	54:39:37	194:18:31
Elapsed time (H:M:S)	1:33:26	57:01:31	195:23:44
Processing memory	12.6 GiB	46.4 GiB	113 GiB
Database size	1.1 GiB	5.3 GiB	20 GiB

Analysis challenges

- 2.6.14 (2005)
  - Unable to compile with modern GCC
  - Installing old GCC on modern Linux impractical
  - 32-bit RAM capacity insufficient for Cscout
- Solution
  - Run kernel on QEMU, Windows Hypervisor accelerator
  - Force use of deprecated crypto, archived packages
  - Compile under QEMU, analyze on powerful host

Analysis challenges

- 6.10 (2024)
  - Requires more than a week of processing
  - Requires more than 100 GB of RAM



Analysis challenges

- 6.10 (2024) Solution
  - Split into 32 tasks
  - Analyze in parallel on a supercomputer’s nodes
  - Develop procedure to merge the results on a powerful node
    - ~~SQL recursive queries~~
    - ~~Graph connected components~~
    - Develop CScout merge command

Binary tournament merge (0:00:00)

```
slurm_script───merge.sh───merge.sh─┬─merge.sh─┬─merge.sh─┬─merge.sh─┬─merge.sh─┬─merge.sh
                                   │          │          │          │          └─sqlite3
                                   │          │          │          └─merge.sh─┬─merge.sh
                                   │          │          │                     └─sqlite3
                                   │          │          └─merge.sh─┬─merge.sh─┬─merge.sh
                                   │          │                     │          └─sqlite3
                                   │          │                     └─merge.sh─┬─merge.sh
                                   │          │                                └─sqlite3
                                   │          └─merge.sh─┬─merge.sh─┬─merge.sh─┬─merge.sh
                                   │                     │          │          └─sqlite3
                                   │                     │          └─merge.sh─┬─merge.sh
                                   │                     │                     └─sqlite3
                                   │                     └─merge.sh─┬─merge.sh─┬─merge.sh
                                   │                                │          └─sqlite3
                                   │                                └─merge.sh─┬─merge.sh
                                   │                                           └─sqlite3
                                   └─merge.sh─┬─merge.sh─┬─merge.sh─┬─merge.sh─┬─merge.sh
                                              │          │          │          └─sqlite3
                                              │          │          └─merge.sh─┬─merge.sh
                                              │          │                     └─sqlite3
                                              │          └─merge.sh─┬─merge.sh─┬─merge.sh
                                              │                     │          └─sqlite3
                                              │                     └─merge.sh─┬─merge.sh
                                              │                                └─sqlite3
                                              └─merge.sh─┬─merge.sh─┬─merge.sh─┬─merge.sh
                                                         │          │          └─sqlite3
                                                         │          └─merge.sh─┬─merge.sh
                                                         │                     └─sqlite3
                                                         └─merge.sh─┬─merge.sh─┬─merge.sh
                                                                    │          └─sqlite3
                                                                    └─merge.sh─┬─merge.sh
                                                                               └─sqlite3
```

Binary tournament merge (1:29:14)

```
slurm_script───merge.sh───merge.sh─┬─merge.sh─┬─merge.sh─┬─merge.sh─┬─merge.sh
                                   │          │          │          └─sqlite3
                                   │          │          └─merge.sh─┬─merge.sh
                                   │          │                     └─sqlite3
                                   │          └─merge.sh─┬─merge.sh───merge.sh─┬─merge.sh
                                   │                     │                     └─sqlite3
                                   │                     └─merge.sh─┬─merge.sh
                                   │                                └─sqlite3
                                   └─merge.sh─┬─merge.sh─┬─merge.sh─┬─merge.sh
                                              │          │          └─sqlite3
                                              │          └─merge.sh─┬─merge.sh
                                              │                     └─sqlite3
                                              └─merge.sh─┬─merge.sh─┬─merge.sh
                                                         │          └─sqlite3
                                                         └─merge.sh─┬─merge.sh
                                                                    └─sqlite3
```

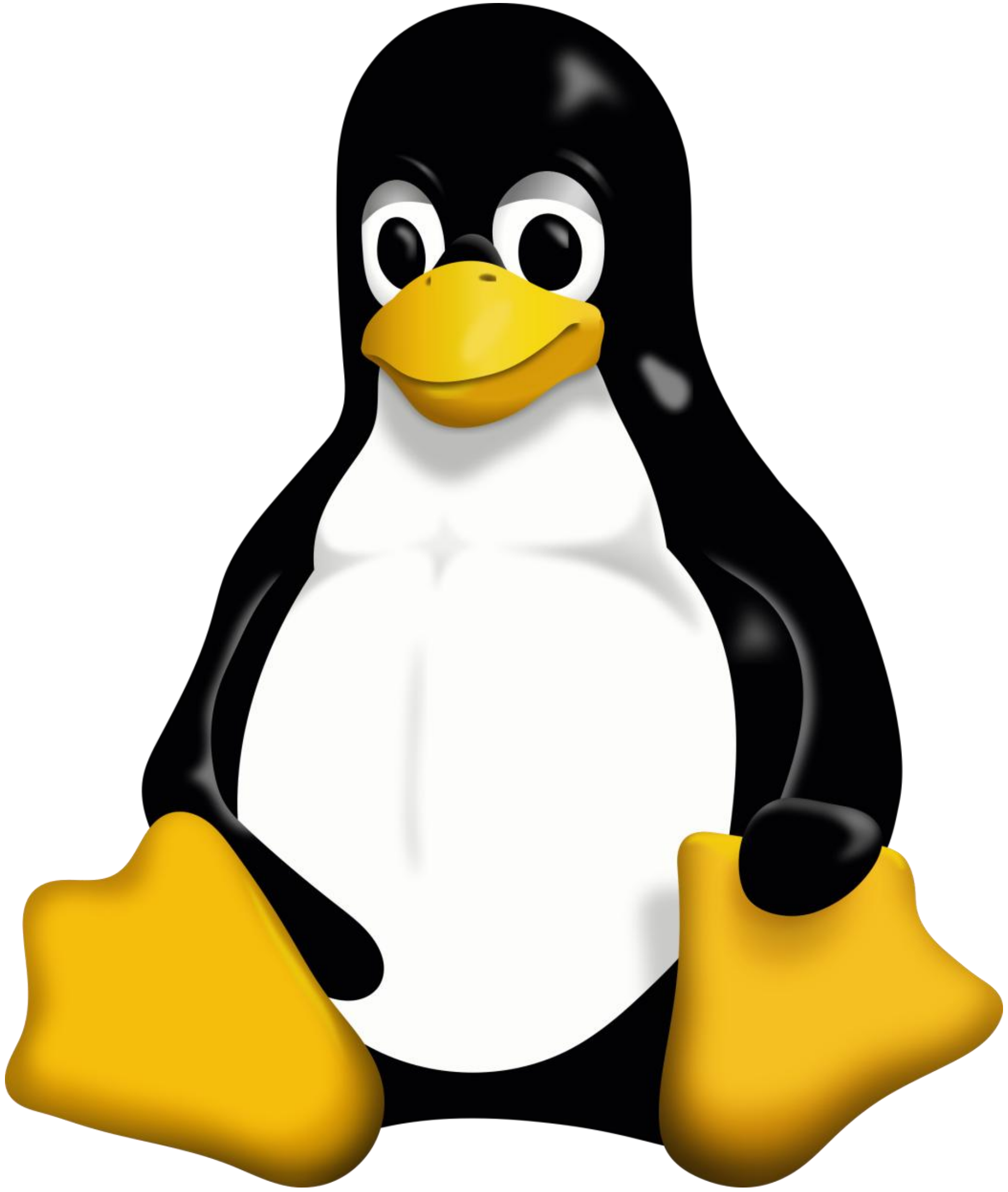
Binary tournament merge (2:04:50)

```
slurm_script───merge.sh───merge.sh─┬─merge.sh─┬─merge.sh─┬─merge.sh
                                   │          │          └─sqlite3
                                   │          └─merge.sh─┬─merge.sh───merge.sh─┬─merge.sh
                                   │                     │                     └─sqlite3
                                   │                     └─merge.sh─┬─merge.sh
                                   │                                └─sqlite3
                                   └─merge.sh─┬─merge.sh───merge.sh─┬─merge.sh
                                              │                     └─sqlite3
                                              └─merge.sh─┬─merge.sh
                                                         └─sqlite3
```

Binary tournament merge (3:57:35)

```
slurm_script───merge.sh───merge.sh─┬─merge.sh───merge.sh─┬─merge.sh
                                   │                     └─sqlite3
                                   └─merge.sh─┬─merge.sh
                                              └─sqlite3
```

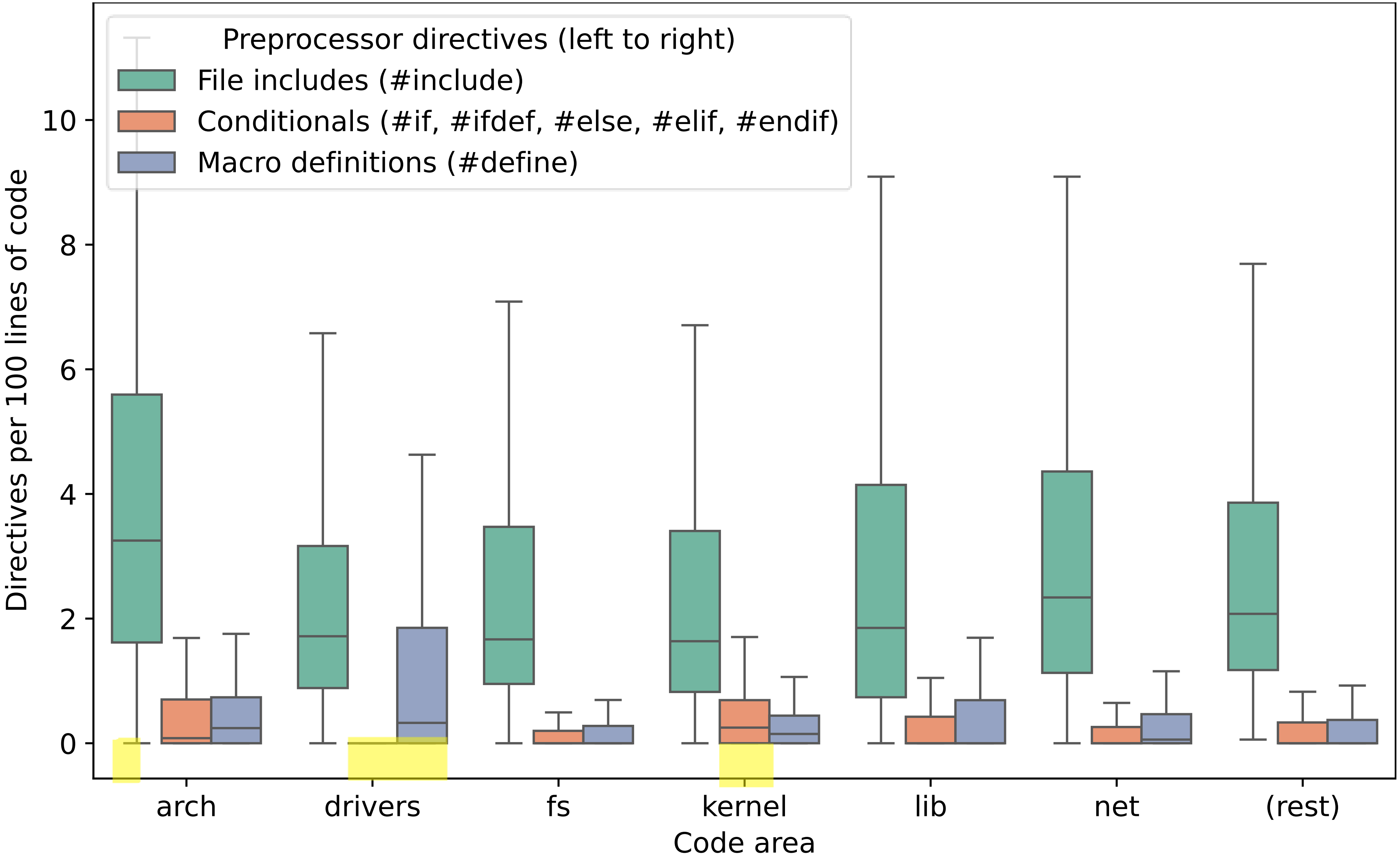
v6.10.1 findings



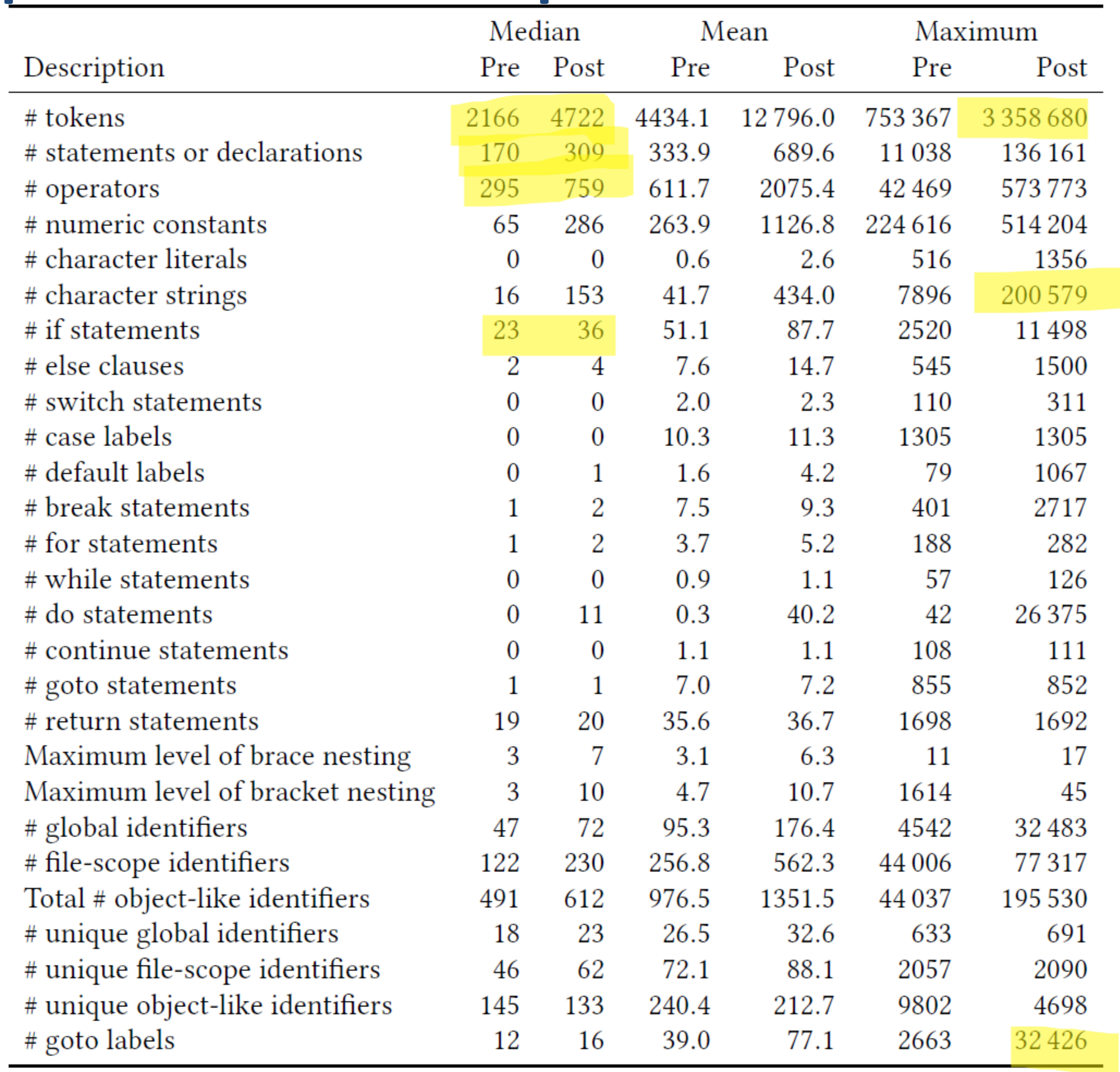
Usage characteristics

- Extensively used:
  - 33% of defined functions
  - 72% of defined identifiers
  - 44% of function identifiers
  - 44% of all identifiers
- 94% of macro identifiers are never used

Variant distribution of C preprocessor directives



Preprocessor expansion in C files



	Median		Mean		Maximum	
Description	Pre	Post	Pre	Post	Pre	Post
# tokens	2166	4722	4434.1	12 796.0	753 367	3 358 680
# statements or declarations	170	309	333.9	689.6	11 038	136 161
# operators	295	759	611.7	2075.4	42 469	573 773
# numeric constants	65	286	263.9	1126.8	224 616	514 204
# character literals	0	0	0.6	2.6	516	1356
# character strings	16	153	41.7	434.0	7896	200 579
# if statements	23	36	51.1	87.7	2520	11 498
# else clauses	2	4	7.6	14.7	545	1500
# switch statements	0	0	2.0	2.3	110	311
# case labels	0	0	10.3	11.3	1305	1305
# default labels	0	1	1.6	4.2	79	1067
# break statements	1	2	7.5	9.3	401	2717
# for statements	1	2	3.7	5.2	188	282
# while statements	0	0	0.9	1.1	57	126
# do statements	0	11	0.3	40.2	42	26 375
# continue statements	0	0	1.1	1.1	108	111
# goto statements	1	1	7.0	7.2	855	852
# return statements	19	20	35.6	36.7	1698	1692
Maximum level of brace nesting	3	7	3.1	6.3	11	17
Maximum level of bracket nesting	3	10	4.7	10.7	1614	45
# global identifiers	47	72	95.3	176.4	4542	32 483
# file-scope identifiers	122	230	256.8	562.3	44 006	77 317
Total # object-like identifiers	491	612	976.5	1351.5	44 037	195 530
# unique global identifiers	18	23	26.5	32.6	633	691
# unique file-scope identifiers	46	62	72.1	88.1	2057	2090
# unique object-like identifiers	145	133	240.4	212.7	9802	4698
# goto labels	12	16	39.0	77.1	2663	32 426

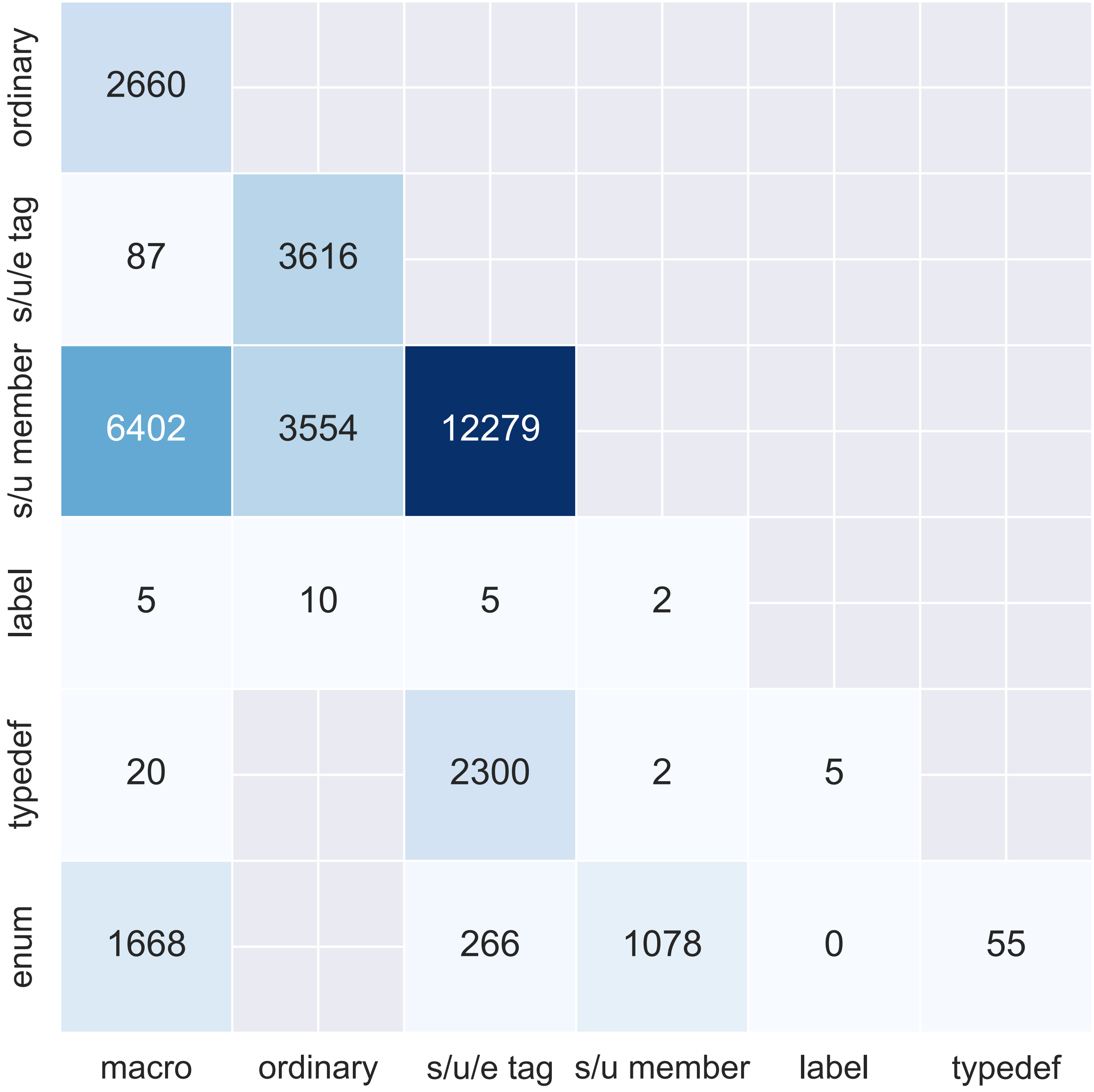
Technical debt: namespace pollution

- 106 363 (median) global namespace occupants at visible the top of each function
- Each macro is used in 81 (median) files
- Ten most frequently defined (full or partial) macro names:
  - defined 3316 times
  - used 152 998 times in 2387 files.

Technical debt: namespace confusion

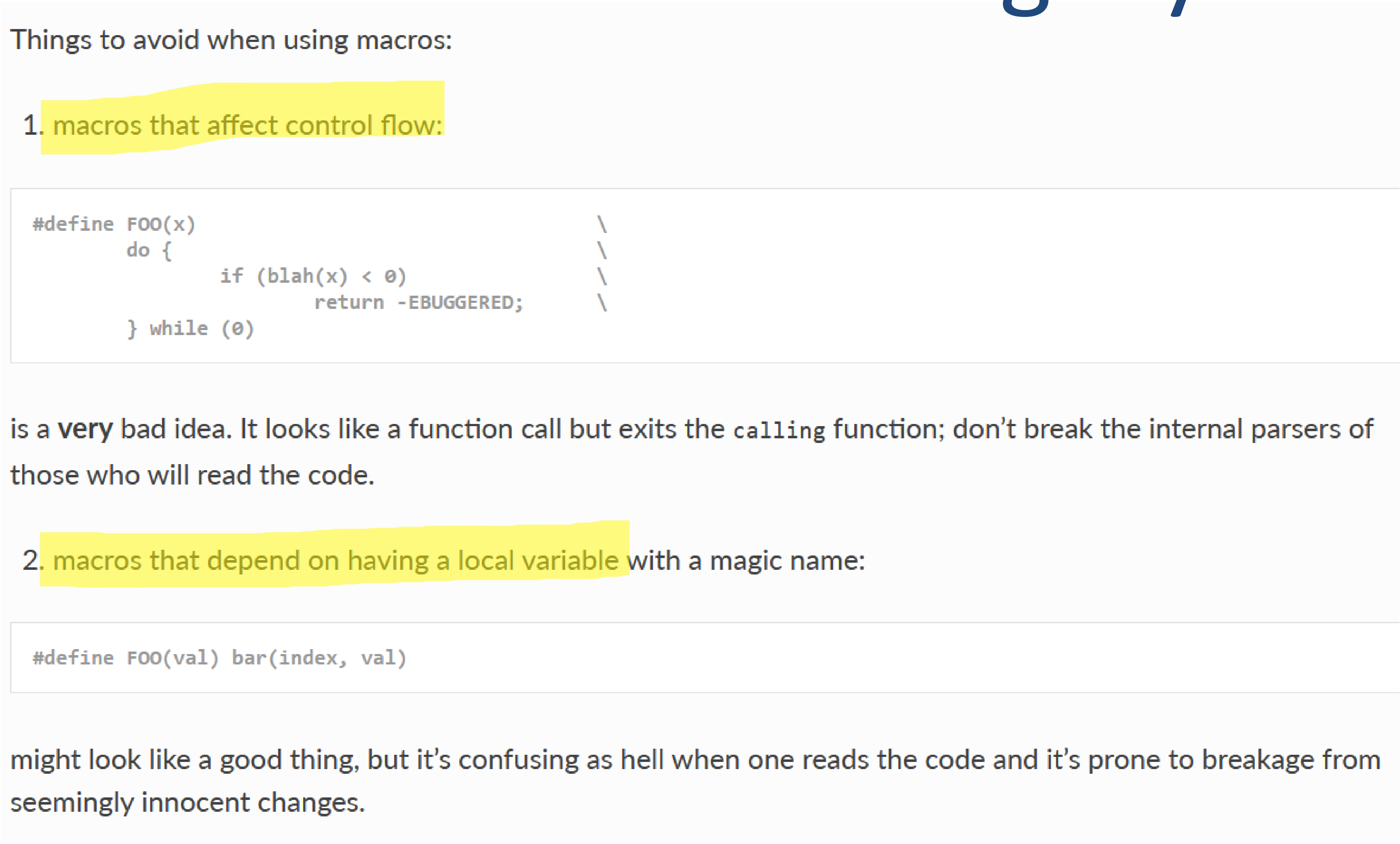
```
#define BCH_ALLOC_FIELDS_V1()                    \
        x(read_time,            16)          \
        x(write_time,           16)          \
        x(data_type,            8)           \
[...]
enum {
#define x(name, _bits) BCH_ALLOC_FIELD_V1_##name,
        BCH_ALLOC_FIELDS_V1()
#undef x
};
[...]
#define x(_name, _bits) out->_name = alloc_field_v1_get(in, &d, idx++);
        BCH_ALLOC_FIELDS_V1()
```

Namespace confusion



	macro	ordinary	s/u/e tag	s/u member	label	typedef
ordinary	2660					
s/u/e tag	87	3616				
s/u member	6402	3554	12279			
label	5	10	5	2		
typedef	20		2300	2	5	
enum	1668		266	1078	0	55

Linux kernel coding style



Things to avoid when using macros:

1. macros that affect control flow:

```
#define FOO(x)                                  \
        do {                                    \
                if (blah(x) < 0)                \
                        return -EBUGGERED;      \
        } while (0)
```

is a **very** bad idea. It looks like a function call but exits the `calling` function; don’t break the internal parsers of those who will read the code.

2. macros that depend on having a local variable with a magic name:

```
#define FOO(val) bar(index, val)
```

might look like a good thing, but it’s confusing as hell when one reads the code and it’s prone to breakage from seemingly innocent changes.

Technical debt: scoping confusion

```
#define BTREE_CACHE_NOT_FREED_INCREMENT(counter) \
do {                                             \
        if (shrinker_counter)                    \
                bc->not_freed_##counter++;       \
} while (0)
// 224 lines omitted
static int __btree_node_reclaim(struct bch_fs *c, struct btree *b, bool flush, bool
shrinker_counter)
{
        struct btree_cache *bc = &c->btree_cache;
// 21 lines omitted
                BTREE_CACHE_NOT_FREED_INCREMENT(dirty);
```

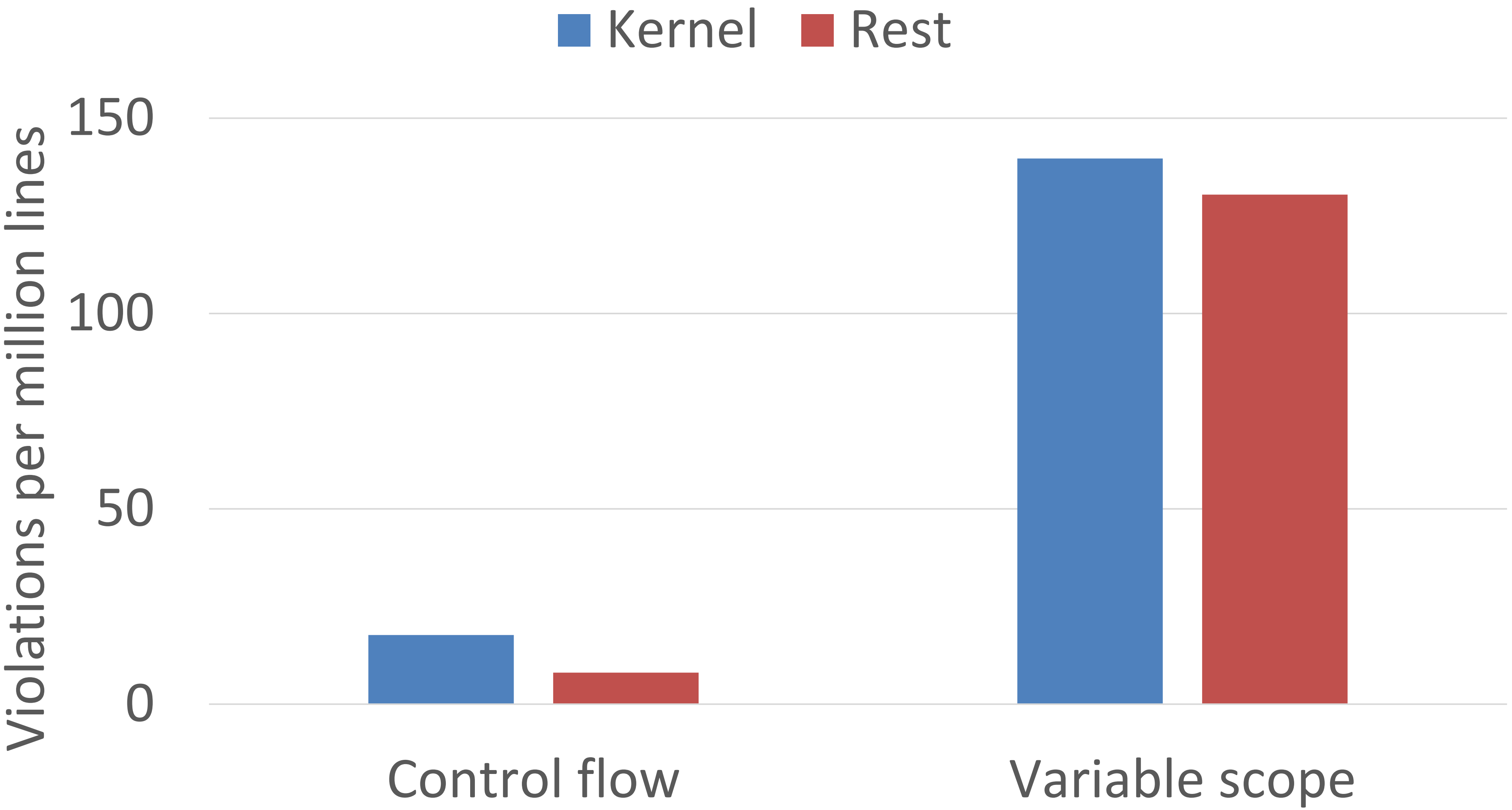
3722 macros defined outside a function contain identifiers local to a C-proper function

Technical debt: control-flow confusion

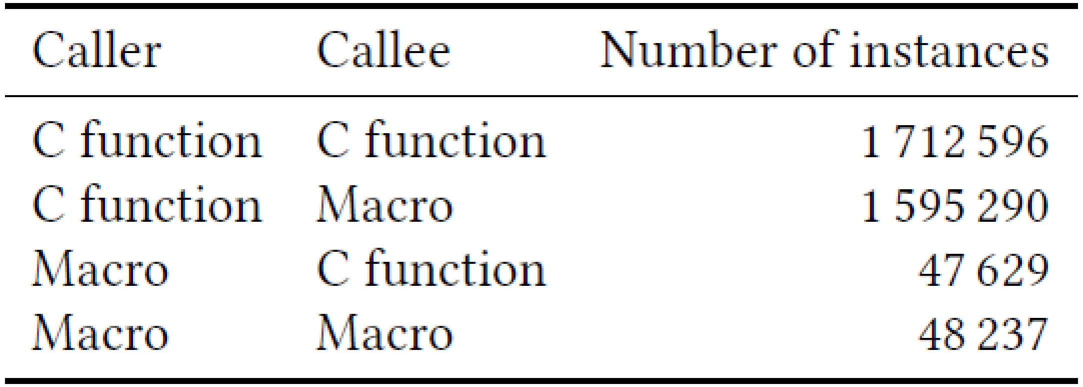
```
#define ENHANCEMENT(name, NAME) do { \
    if (enhancements.name) { \
        if (!psb_intel_sdvo_get_value(psb_intel_sdvo, SDVO_CMD_GET_MAX_##NAME,
&data_value, 4) || \
            !psb_intel_sdvo_get_value(psb_intel_sdvo, SDVO_CMD_GET_##NAME, &response, 2)) \
            return false; \
// 11 limes omitted
} while(0)

static bool
psb_intel_sdvo_create_enhance_property_tv(struct psb_intel_sdvo *psb_intel_sdvo,
    struct psb_intel_sdvo_connector *psb_intel_sdvo_connector,
    struct psb_intel_sdvo_enhancements_reply enhancements)
{
// 77 limes omitted
        ENHANCEMENT(vpos, VPOS);
        ENHANCEMENT(saturation, SATURATION);
```

Small: 12 continue, 42 break, 83 goto, 97 return

Violation density



Technical debt: hybrid call paths



Caller	Callee	Number of instances
C function	C function	1 712 596
C function	Macro	1 595 290
Macro	C function	47 629
Macro	Macro	48 237

412 695 length-3 chains of C functions calling another C function via a macro.

Technical debt: expansion explosion

- 491 files expand by 1393% (median) against 87%
- 29 outliers take 14 s (median) to compile against 1.8

arch/x86/xen/setup.c

944 lines 26 kB

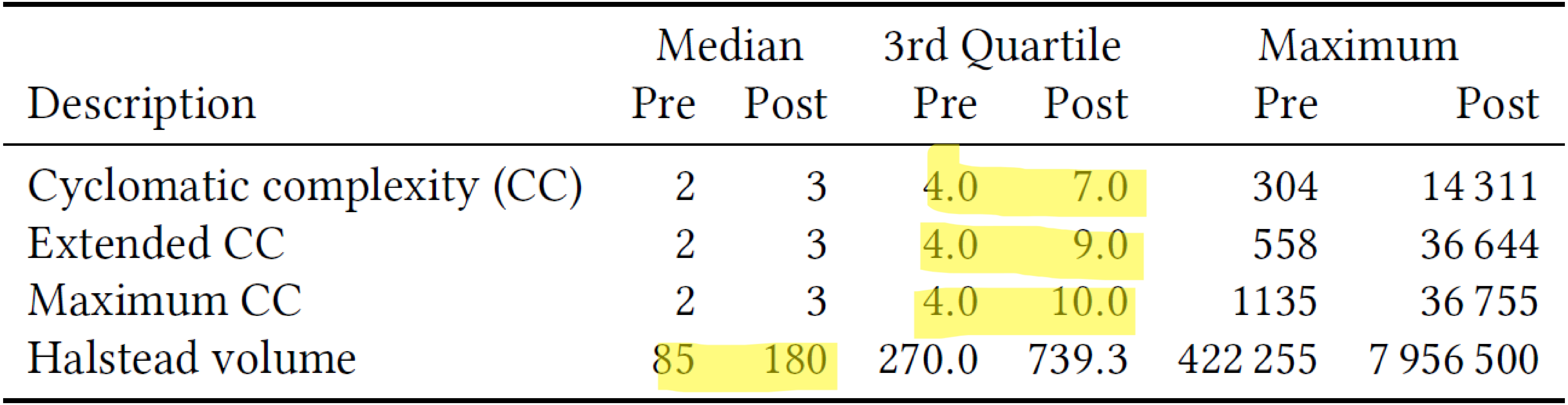
setup.i

88,343 lines 49 MB!

Compilation cost:
7’36”
2.7 GiB RAM

```
commit 21b136cc63d2a9ddd60d4699552b69c214b32964
Author: Linus Torvalds <torvalds@linux-foundation.org>
Date:   Tue Jul 30 15:44:16 2024 -0700

    minmax: fix up min3() and max3() too

    David Laight pointed out that we should deal with the min3() and max3()
    mess too, which still does excessive expansion.
```

Technical debt: complexity metrics

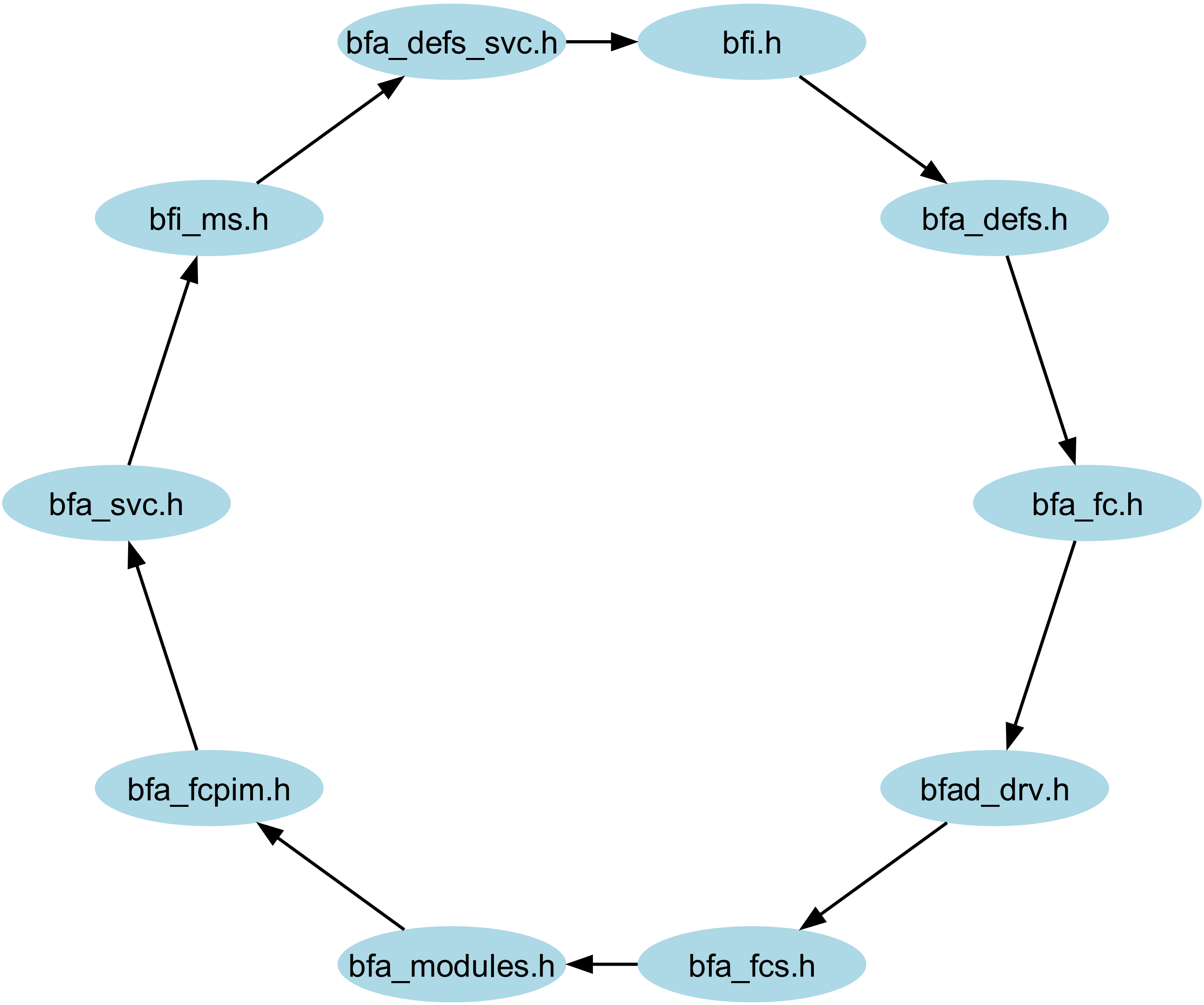


	Median		3rd Quartile		Maximum	
Description	Pre	Post	Pre	Post	Pre	Post
Cyclomatic complexity (CC)	2	3	4.0	7.0	304	14 311
Extended CC	2	3	4.0	9.0	558	36 644
Maximum CC	2	3	4.0	10.0	1135	36 755
Halstead volume	85	180	270.0	739.3	422 255	7 956 500

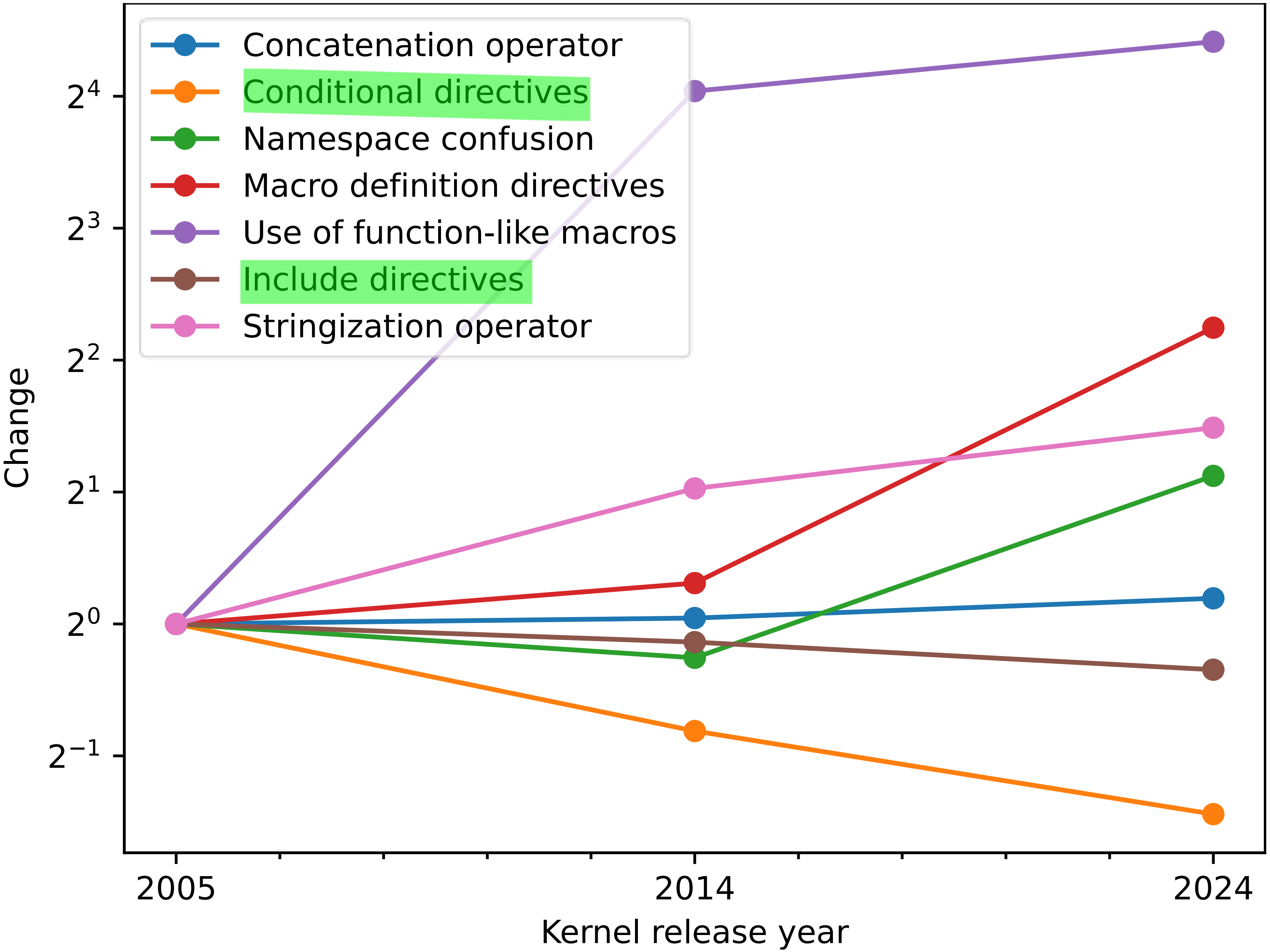
Technical debt: More!

- Composite identifiers: 143 017 concatenation operators
- Extensive include hierarchies
  - 84 outlier compilation units include 1,6 M (median) lines
  - Each compilation unit includes 2156 (median) files
  - 36 603 include file outliers with depth-12 (median) nesting
- Cyclic include file dependencies
  - 177 489 total; 7.5 (mean) per compilation unit
  - Longest consists of 10 elements

drivers/scsi/bfa/



Usage evolution



Reducing C preprocessor’s technical debt

- 4 977 706 object-like macro identifiers (out of 5 094 759)

```
#define WRITE 1
→
static const int WRITE = 1;

enum { WRITE = 1 };
```

- Possible for about 77%. Rest:
  - Value is (probably) not a compile-time constant — 1M
  - Value used in token concatenation (a ## b), stringization (# a) — 90k
  - Value used in preprocessor context (#if, #ifdef, defined — 23k)

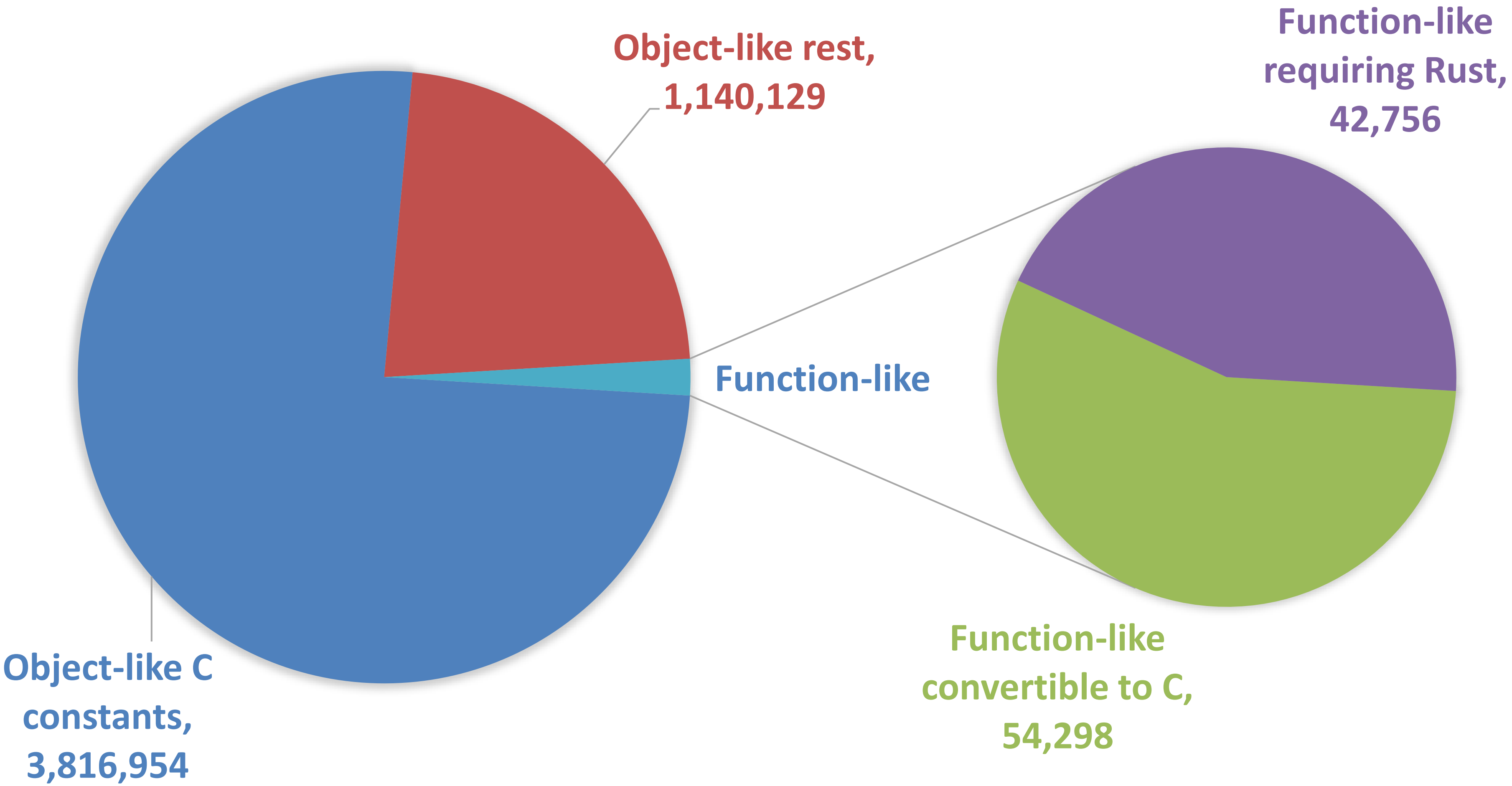
Reducing C preprocessor’s technical debt

- What about the 97 054 function-like macros?
- 54 298 could be converted to C
- 42 756 could be written in Rust
  - More powerful type system
  - Typed, syntactically complete macros
  - Can process code declaratively and by manipulating syntax tree

In Rust we trust?

- 32 415 function-like macros not used as C functions: const functions (initialize data structures), Rust macros
- 9693 use token concatenation: Rust `concat_idents!` (evolving)
- 5219 use non-object parameters: Rust macro metavariables
- 3722 access local variables: refactor to pass parameters
- 1766 have stringifications: Rust `stringify!`
- 234 affect control flow: Rust macros / refactor
- 43 use typeof: Rust traits, generic parameters
- 28 have incomplete syntax: refactor
- **Overall**: 42 756 macros (44%) would require Rust

Ditching C preprocessor macros



Conclusions: C preprocessor usage

- Extensive
- Introducing technical debt in all preprocessor dimensions
- Still growing in some areas
- Expensive to address

// TODO

- Short term:
  - – Fix macro explosions.
  - – Correct frequent cyclic dependencies.
  - – Convert 77% object-like macros into C constants.
- Long term:
  - – Prioritize refactoring of function-line macros into C/Rust.

ΟΙΚΟΝΟΜΙΚΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΑΘΗΝΩΝ

ATHENS UNIVERSITY
OF ECONOMICS
AND BUSINESS

TU Delft

Thank you!

🌐 www.spinellis.gr

🦋 𝕏 @CoolSWEng

m @CoolSWEng@mastodon.acm.org

✉ dds@aueb.gr

Business
Analytics Lab

Backup slides

Goto label aliasing

```
#define emulator_try_cmpxchg_user(t, ptr, old, new) \
        (__try_cmpxchg_user((t __user *)(ptr), (t *)(old), *(t *)(new), efault ## t))
```

```
switch (bytes) {
    case 1:
        r = emulator_try_cmpxchg_user(u8, hva, old, new);
        break;
    case 2:
        r = emulator_try_cmpxchg_user(u16, hva, old, new);
        break;
    case 4:
        r = emulator_try_cmpxchg_user(u32, hva, old, new);
        break;
    case 8:
        r = emulator_try_cmpxchg_user(u64, hva, old, new);
        break;
```

```
#define __try_cmpxchg_user(_ptr, _oldp, _nval, _label)  ({          \
    int __ret = -EFAULT;                                            \
    __uaccess_begin_nospec();                                       \
    __ret = !unsafe_try_cmpxchg_user(_ptr, _oldp, _nval, _label);   \
_label:                                                             \
    __uaccess_end();                                                \
    __ret;                                                          \
                                                })
```

Predefined variable macros

Name	Occurrences
__func__	52436
__LINE__	2740
__FILE__	876
__PRETTY_FUNCTION__	8
__DATE__	2
__FUNCTION__	2
__TIME__	2

Frequently defined macros

Name	Definitions
OTG	532
_MASK	455
DRV_NAME	439
CM	371
__SHIFT	308
DRIVER_NAME	288
DP	262
DRIVER_DESC	257
DIG	225
HUBPREQ	179