0x5

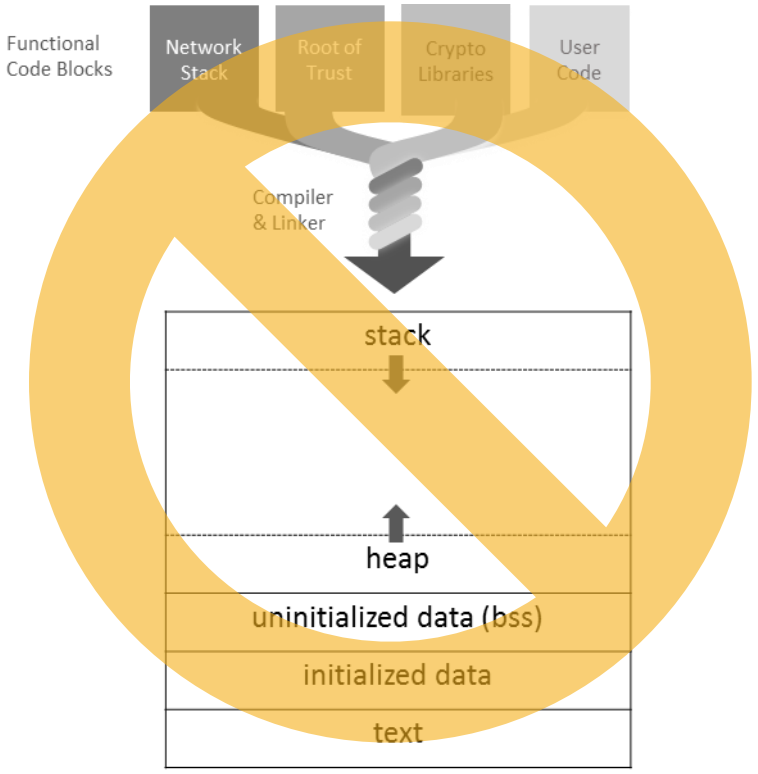
HEX-Five™

Making RISC-V The Most Secure Platform

Cesare Garlati, CEO Hex Five Security

Andes RISC-V Con Beijing 2019

Embedded Computing Thread Model



- MCU-based lack basic hardware security primitives like MMU & Virtual Memory
  - ➡ any line of code can break the CIA Confidentiality Integrity Availability
- Linux-based have MMU & VM but can't be trusted either
  - ➡ 17M+ lines of code attack surface and non-free kernel drivers
- Untrusted software: 3rd party libraries, open source, proprietary binaries
  - ➡ Supply chain security: 100+ libraries in a typical IoT stack
- Secure Elements & TPM Modules: secure data at rest, can't run programs
- Arm has TrustZone®: too complex and expensive for mainstream adoption
- RISC-V: free and open (good) but no TrustZone® at all (bad)

... while regulators increasingly mandate "isolation" built into any device

RISC-V ISA Security Building Blocks

Privilege Levels & Control and Status Registers

- Machine – always present, highest privilege mode
- Supervisor – Linux, supports MMU / virtual memory
- Reserved (Hypervisor) – work in progress
- User / Application – unprivileged lowest level
- Trusted Execution Environment runs at highest privilege
- Note: Interrupts always M mode (unless "N" implemented)

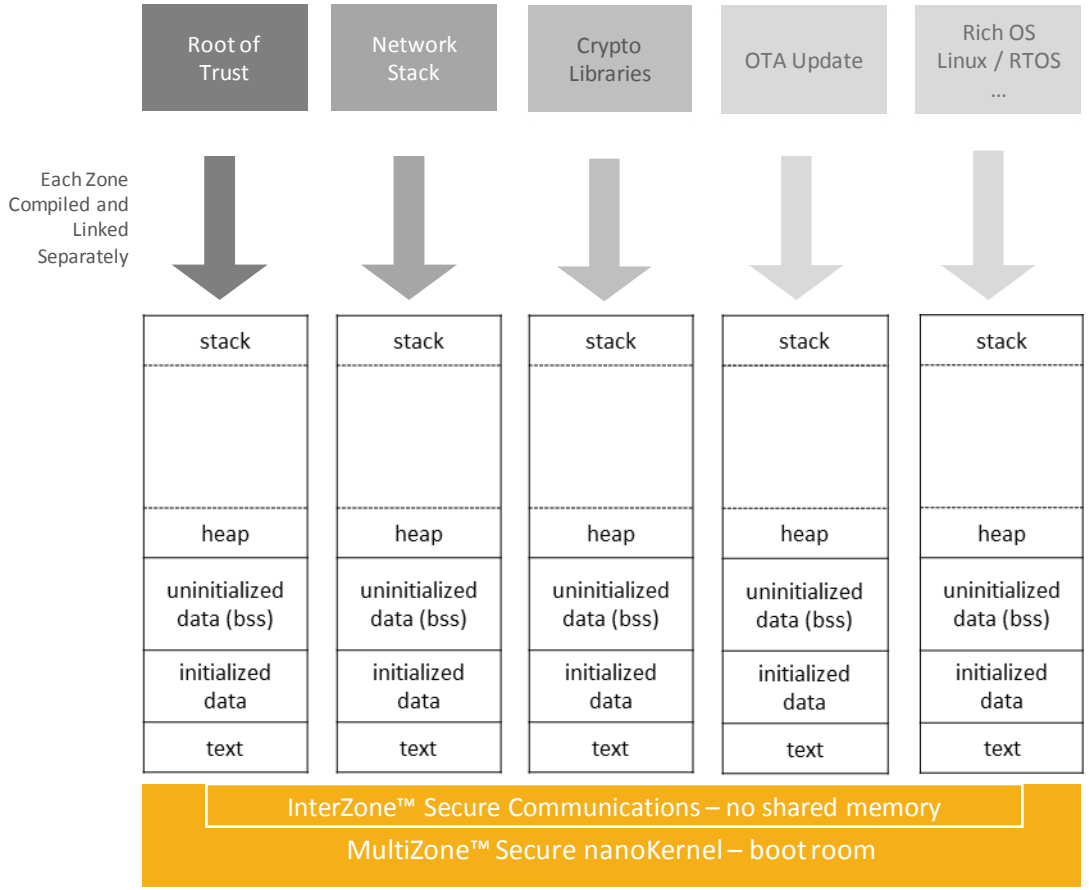
Physical Memory Protection

- Hardware enforced – 4 ranges * 4 config reg (if implemented)
- Policy R/W/X => synchronous exception mechanism (trap)
- Overlapping OK, ranges can be locked down
- Top of range (TOR) or naturally aligned power of two (NAPOT)
- Trusted Execution Environment manages PMP context at runtime
- Note: enforced per core – no ISA spec for multi-core / platform

Rings	Modes	Intended Usage
1	M	Unsecured embedded
2	M,U	Secure embedded
3	M,S,U	Linux

A	Name	Description
1	TOR	Top of range
2	NA4	Naturally aligned 4-byte
3	NAPOT	Naturally aligned power of 2

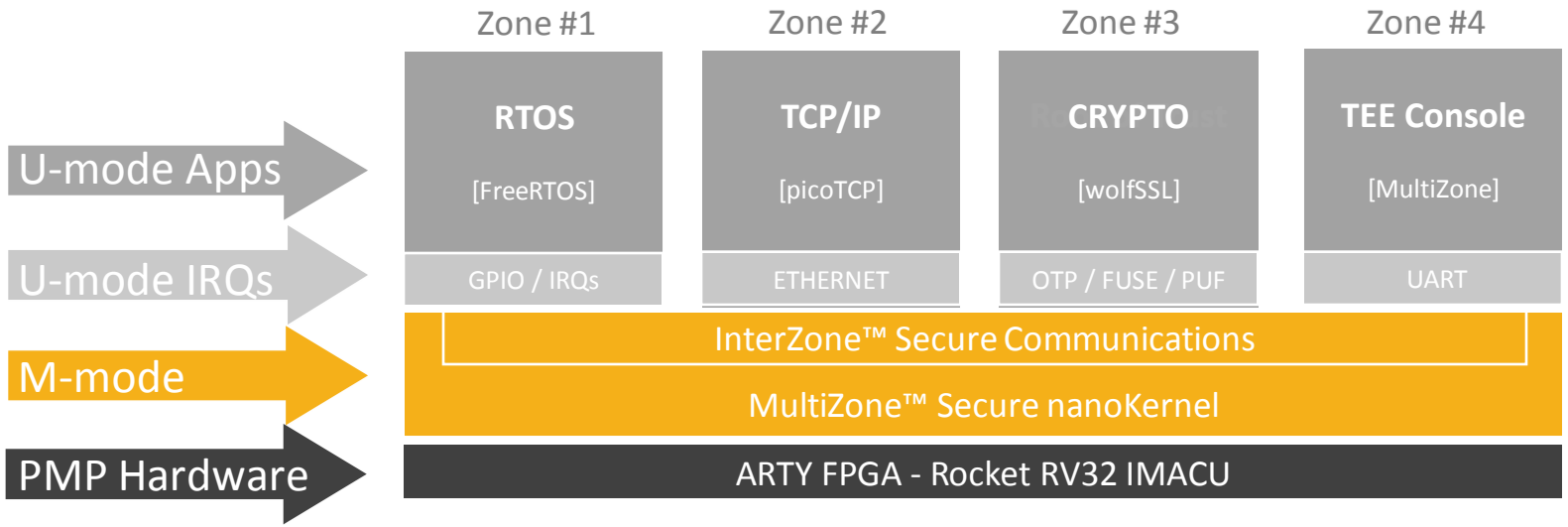
MultiZone™ Trusted Execution Environment



- ☑ Multiple equally secure zones for programs, data, i/o
- ☑ Hardware-enforced Software-defined Policy-driven RWX
- ☑ Minimal attack surface (<2KB), Formally verifiable

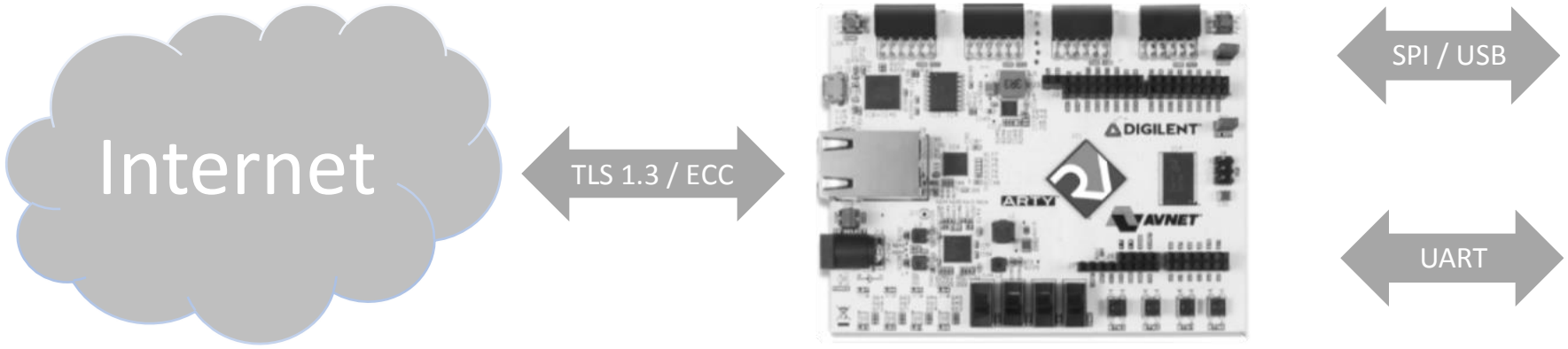
- ➡ It's like TrustZone® for RISC-V
- ➡ Runs on any RISC-V core with PMP & U-Mode
- ➡ No need to modify application software or toolchain
- ➡ Use cases: TLS, secure boot, remote firmware update

Reference Application – Secure IoT Stack



Crypto

TLSv1.3, Cipher TLS_AES_128_GCM_SHA256
Peer signing digest: SHA256
Peer signature type: ECDSA
Server Temp Key: ECDH, P-256, 256 bits
Server public key is 256 bit
Private Key ASN1 OID: prime256v1
Private Key NIST CURVE: P-256

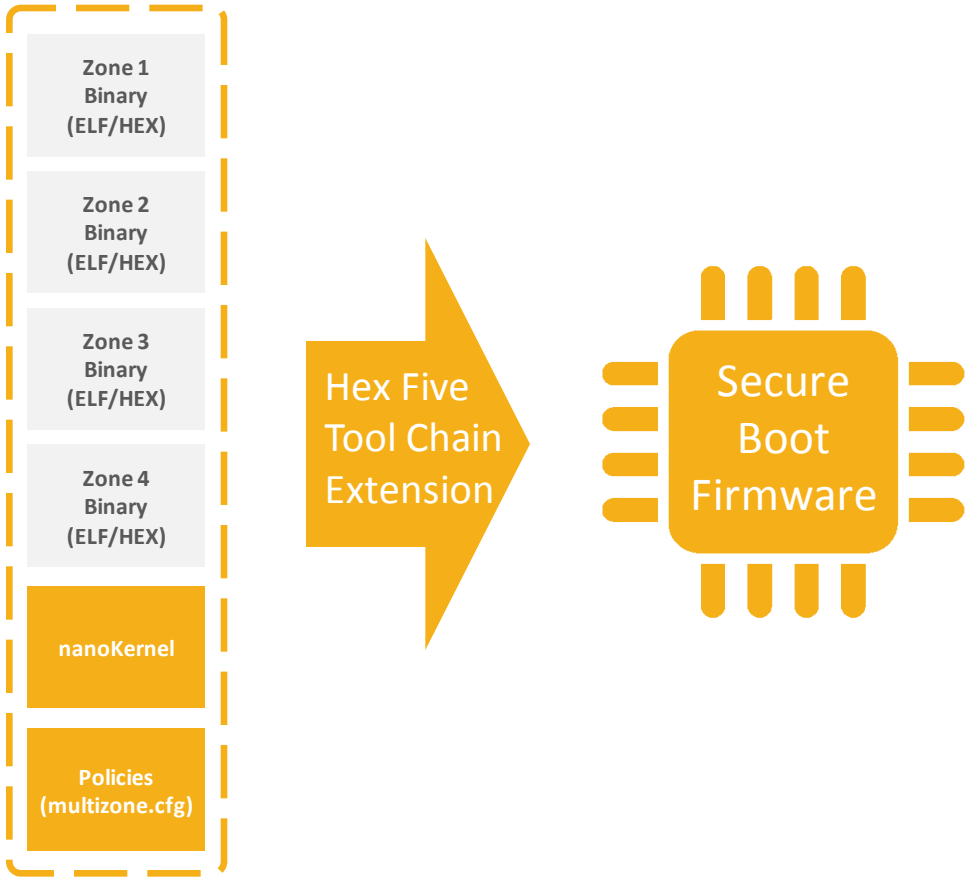


MultiZone™ Security – How It Works

Open | multizone.cfg ~/eclipse-cdt-ws/hexfive-conf | Save

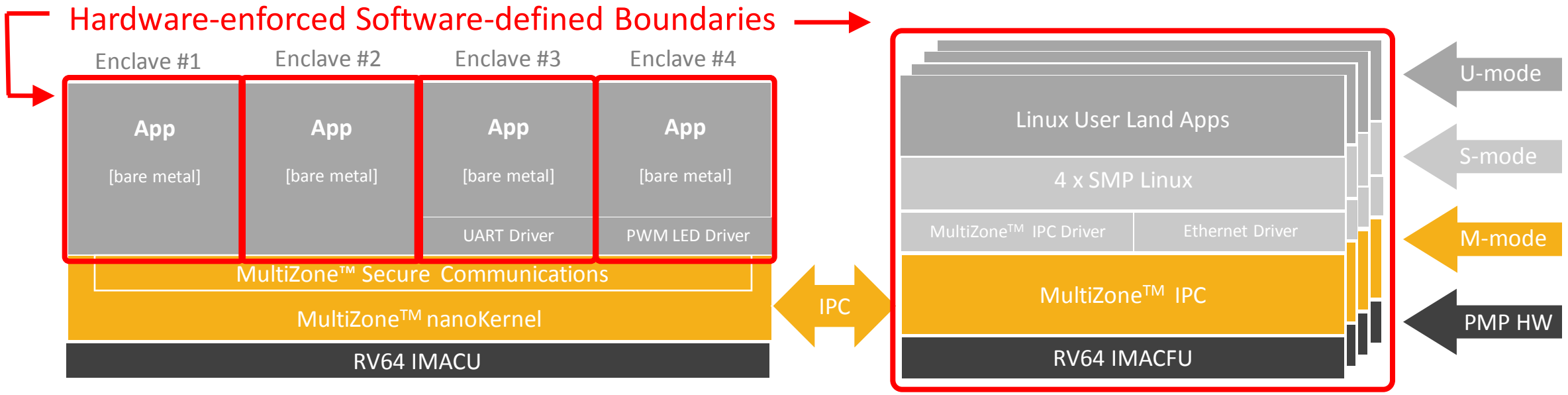
```
Tick = 10 # ms

Zone = 1
               irq  = 16 # BTN0
               base = 0x20410000; size =      64K; rwx = rx # FLASH
               base = 0x80001000; size =      16K; rwx = rw # RAM
               base = 0x10025000; size =    0x100; rwx = rw # PWM
               base = 0x10012000; size =    0x100; rwx = rw # GPIO
               base = 0x0C000000; size = 0x400000; rwx = rw # PLIC

Zone = 2
               irq  = 17, 18 # BTN1, BTN2
               base = 0x20420000; size =   64K; rwx = rx # FLASH
               base = 0x80005000; size =   16K; rwx = rw # RAM
               base = 0x60000000; size =    8K; rwx = rw # XEMACLITE

Zone = 3
               base = 0x20430000; size =   64K; rwx = rx # FLASH
               base = 0x80009000; size =    4K; rwx = rw # RAM

Zone = 4
               base = 0x20440000; size =   64K; rwx = rx # FLASH
               base = 0x8000A000; size =    4K; rwx = rw # RAM
               base = 0x10013000; size = 0x100; rwx = rw # UART
```

Plain Text | Tab Width: 3 | Ln 10, Col 1 | INS



Patent pending US 16450826, PCT US1938774 – Configuring, Enforcing, And Monitoring Separation Of Trusted Execution Environments.

MultiZone™ For Linux – Enclave Concept



- ☑ Multiple statically defined enclaves – ram, rom, i/o, irq
- ☑ Secure messaging with no shared mem – secure buffers for Linux IPC
- ☑ Secure interrupt handlers mapped to enclaves and executed in U-mode
- ☑ Trap & emulation of privileged instructions, Soft-timers, Secure boot

Takeaways

- Embedded systems – with or without MMU – are inherently not secure as all code can access all data and peripherals
- The RISC-V ISA defines some security building blocks including privileged modes and physical memory protection
- The design complexity associated with properly implementing security primitives often results in them not being used at all

1. *MultiZone™ security provides multiple equally secure execution environments*
2. *MultiZone™ provides hardware-enforced software-defined separation for programs data and I/O*
3. *MultiZone™ Security doesn't require additional cores, specialized IP or changes to existing applications*

Hex Five MultiZone™ Security

Hex Five Security, Inc. is the creator of MultiZone™ Security, the first Trusted Execution Environment for RISC-V. Hex Five open standard technology provides software-defined hardware-enforced separation for multiple security domains, with full isolation of data, programs and peripherals. Contrary to traditional solutions, MultiZone™ Security requires no additional hardware or changes to existing software: open source libraries, third party binaries and legacy code can be configured in minutes to achieve unprecedented levels of safety and security.

MultiZone™ Open Standard API – C Library

```
/* Copyright(C) 2019 Hex Five Security, Inc.

Permission to use, copy, modify, and/or distribute this software for
any purpose with or without fee is hereby granted, provided that the
above copyright notice and this permission notice appear in all copies.

*/

#ifndef LIBHEXFIVE_H_
#define LIBHEXFIVE_H_

void ECALL_YIELD();
void ECALL_WFI();

int ECALL_SEND(int, void *);
int ECALL_RECV(int, void *);

void ECALL_TRP_VECT(int, void *);
void ECALL_IRQ_VECT(int, void *);

void ECALL_CSRS_MIE();
void ECALL_CSRC_MIE();

void ECALL_CSRW_MTIMECMP(uint64_t);

uint64_t ECALL_CSRR_MTIME();
uint64_t ECALL_CSRR_MCYCLE();
uint64_t ECALL_CSRR_MINSTR();
uint64_t ECALL_CSRR_MHPMC3();
uint64_t ECALL_CSRR_MHPMC4();

uint64_t ECALL_CSRR_MISA();
uint64_t ECALL_CSRR_MVENDID();
uint64_t ECALL_CSRR_MARCHID();
uint64_t ECALL_CSRR_MIMPID();
uint64_t ECALL_CSRR_MHARTID();

#endif /* LIBHEXFIVE_H_ */
```

Permissive Licensing – “any purpose”

Hardware threads (zones) management

Inter zone messaging – zone0 SMP Linux

Traps & IRQs handlers registration (U-mode)

Traps & IRQs enable / disable – per zone

Hardware thread timer – per zone

Trap & emulation helpers
Read-only, selected CSRs
Completely optional – just for speed / latency

Virtual Memory (MMU) Uncomfortable Truth

```
~/linux-4.18.6$ cloc --exclude-lang=DTD,Lua,make .
   60965 text files.
   60546 unique files.
   14391 files ignored.

-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C                            25782        2554166        2248398       12965944
C/C++ Header                 18693         484773         892818        3629746
Assembly                      1318          47155         105960         232515
JSON                           189              0              0         102201
Perl                            55           5414           3994          27294
Bourne Shell                   346           5633           4983          24450
Python                         108           3055           3337          17427
HTML                             5            669              0           5492
yacc                             9            701            375           4648
lex                              8            326            314           2007
C++                              7            285             77           1844
Bourne Again Shell              51            351            318           1711
awk                             11            170            155           1384
Markdown                         1            220              0           1077
TeX                              1            108              3            915
NAnt script                      2            156              0            599
Windows Module Definition        2             14              0            102
m4                               1             15              1             95
XSLT                             5             13             26             61
CSS                              1             18             27             44
vim script                       1              3             12             27
Ruby                             1              4              0             25
INI                              1              1              0              6
sed                              1              2              5              5
-------------------------------------------------------------------------------
SUM:                         46599        3103252                     17019619
-------------------------------------------------------------------------------
```

(a) Industry Average: "about 15 - 50 errors per 1000 lines of delivered code." He further says this is usually representative of code that has some level of structured programming behind it, but probably includes a mix of coding techniques.

(b) Microsoft Applications: "about 10 - 20 defects per 1000 lines of code during in-house testing, and 0.5 defect per KLOC (KLOC IS CALLED AS 1000 lines of code) in released product (Moore 1992)." He attributes this to a combination of code-reading techniques and independent testing (discussed further in another chapter of his book).

(c) "Harlan Mills pioneered 'cleanroom development', a technique that has been able to achieve rates as low as 3 defects per 1000 lines of code during in-house testing and 0.1 defect per 1000 lines of code in released product (Cobb and Mills 1990).

17,019,619 * 10^{-4} = **1,701** disasters waiting to happen

Credits: Al Danial https://github.com/AlDanial/cloc, Dan Mayer's development blog https://www.mayerdan.com/ruby/2012/11/11/bugs-per-line-of-code-ratio