CRYPTOGRAPHICALLY TAGGED MEMORY

SEC4DEV, FEBRUARY 2020

VIENNA, AUSTRIA

PASCAL NASAHL
ROBERT SCHILLING
MARIO WERNER
JAN HOOGERBRUGGE
STEFAN MANGARD
MARCEL MEDWED

SECURE CONNECTIONS FOR A SMARTER WORLD

26,000+ Customers

Employees in
30+ Countries
Headquartered in Eindhoven, Netherlands

~30,000
Employees

9,000
Patent Families

$9.41B
Annual Revenue[1]

60+
Year History

~9,000
R&D Engineers

[1] Posted revenue for 2018 – Please refer to the Financial Information page of the Investor Relations section of our website at www.nxp.com/investor for additional information

Secure Connections for a Smarter World Is Becoming a Reality

NXP has all enabling technologies

Outlook

This talk is about automatic run-time protection for unsafe languages

- Introduction
- Tagged memory
- Cryptographically tagged memory
- Conclusions

INTRO

Motivation

Secure Connections for a Smarter World

Everything Connected

1B+ additional consumers online, **30B+** connected devices

Connectivity

Everything Smart

40B+ devices with intelligence shipped in **2020**

Processing

Everything Secure

Potential savings to economy up to **half trillion dollars**

Security

Source: Euromonitor; Gartner; ARM Holdings; UBS; Center for Strategic and International Studies; McAfee, NXP analysis, International Telecommunications Union

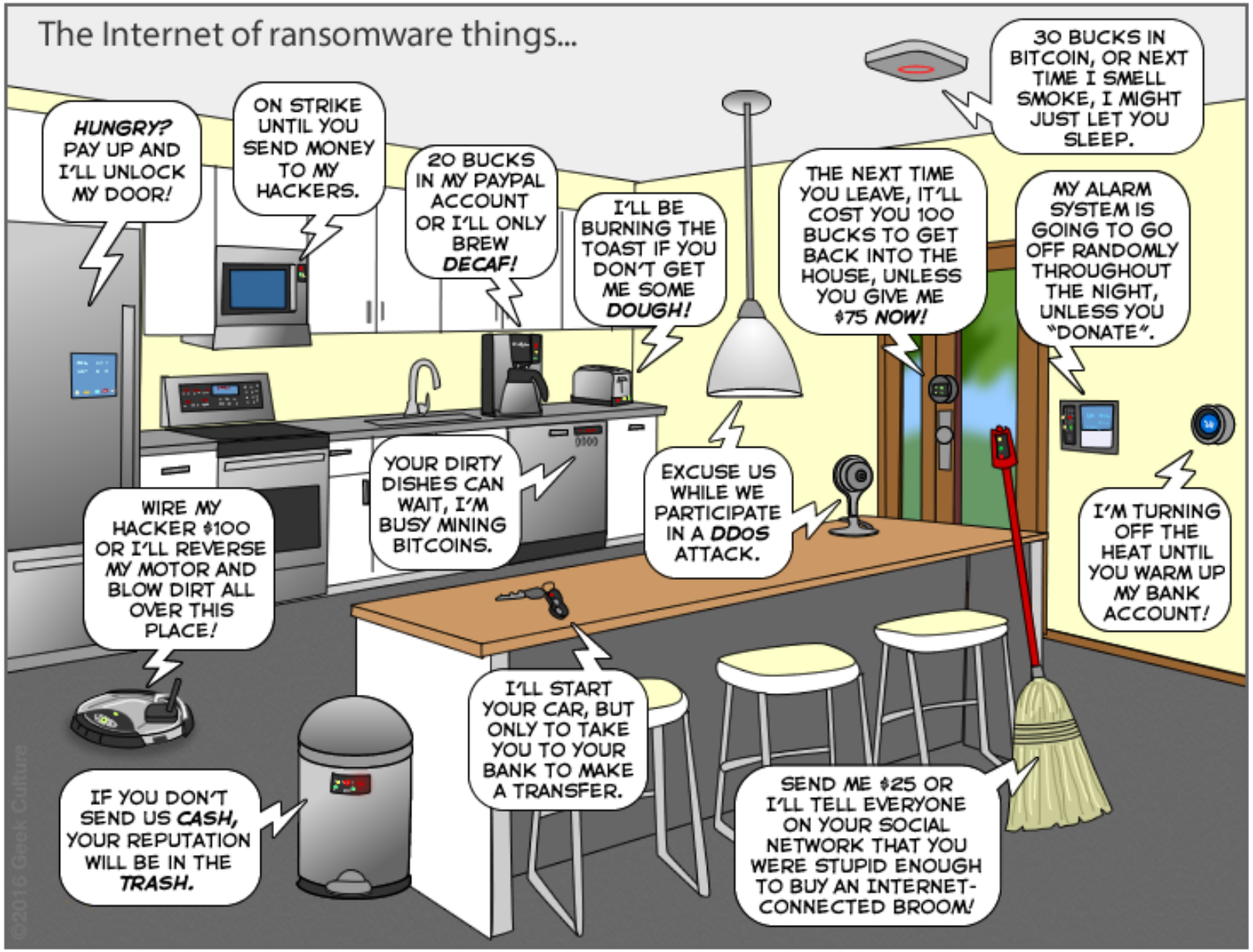
Remote attacks

- Remote attacks are known for PCs/servers
  - but IoT devices are a much easier target nowadays

- Attack types
  - Stack overflows/execution → till 2004… but not every IoT device has an MMU
  - ROP attacks since 2010
  - JOP, DOP,…

- Consequences
  - DoS
  - Botnets (steal computational power, DDoS, ransomware, spamming, ...)
  - Controlling fleets (control critical infrastructure, congest cities)

The Joy of Tech™ by Nitrozac & Snaggy

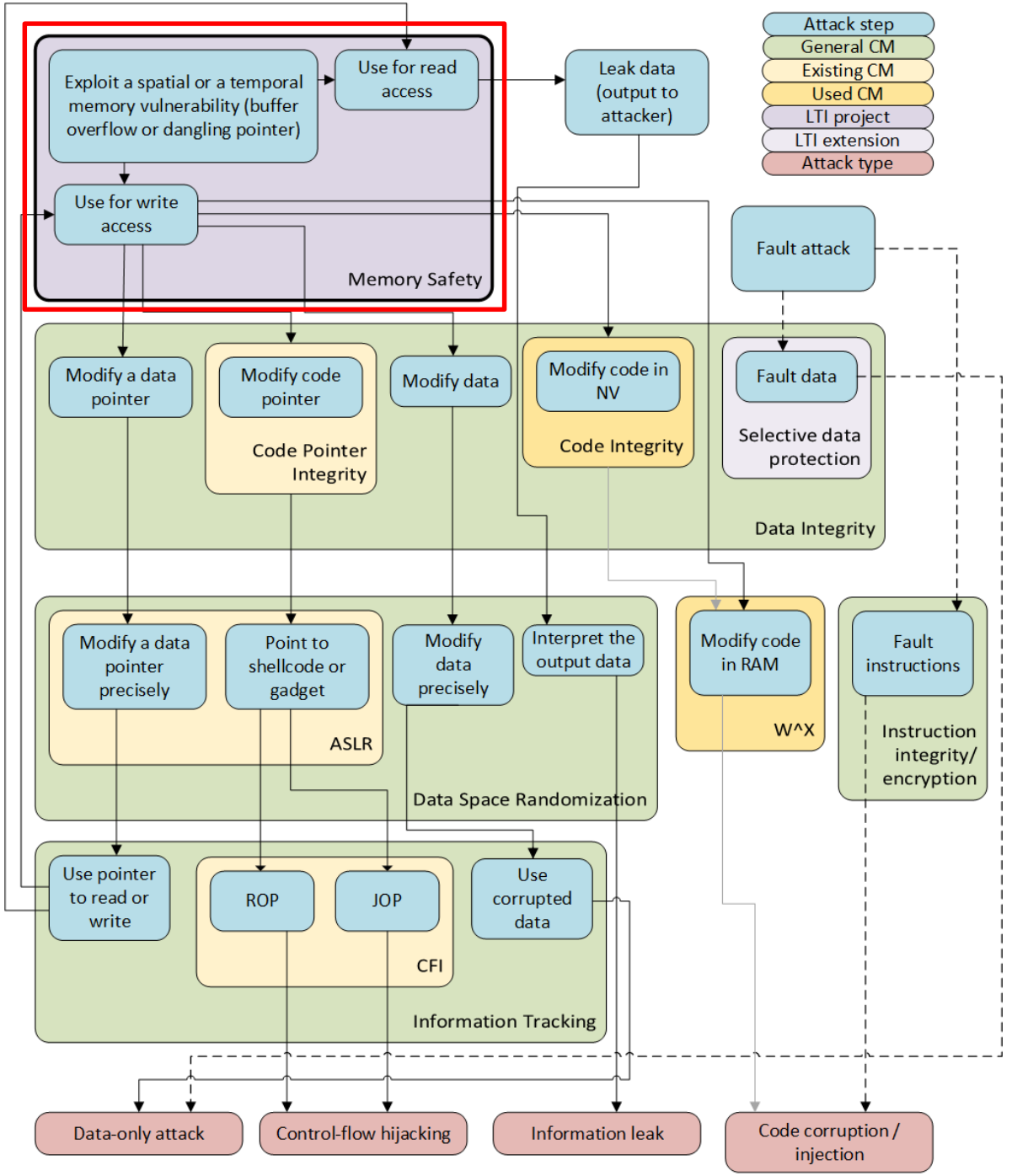


You can help us keep the comics coming by becoming a patron!
www.patreon/joyoftech

joyoftech.com

Mitigation of remote attacks

- All remote attacks need a memory vulnerability (except for weak passwords, protocol bugs, etc. and XSS and …)
- Memory safety prevents the exploitation of software bugs for buffer overflows, use-after-free, …
- **Strong memory safety is costly**
- **Tagged memory is the most promising approach**



BACKGROUND

Motivation and Use cases for tagged memory

- Store meta info next to data
  - Already in the 70s for debugging, data type indication
- Simple taint tracking
  - One bit tag size
  - f(tainted, safe) → tainted
  - No control flow based on tainted data → sanitize
- Full blown policy engines
  - e.g. f(instr, tag(src1), tag(src2), tag(*dst))
  - Up to 32-bit tags
  - NXP & Dover Microsystems CoreGuard
  - Taint tracking, landing pads, memory coloring
- Memory coloring

Data (machine word)	Tag (1 bit)
Ext User	Tainted
Input\0\0\0	Tainted
Function pointer 1	Safe
Function pointer 2	Safe

Memory coloring (Setup)

- Each pointer carries a tag (aka color) in the unused bits

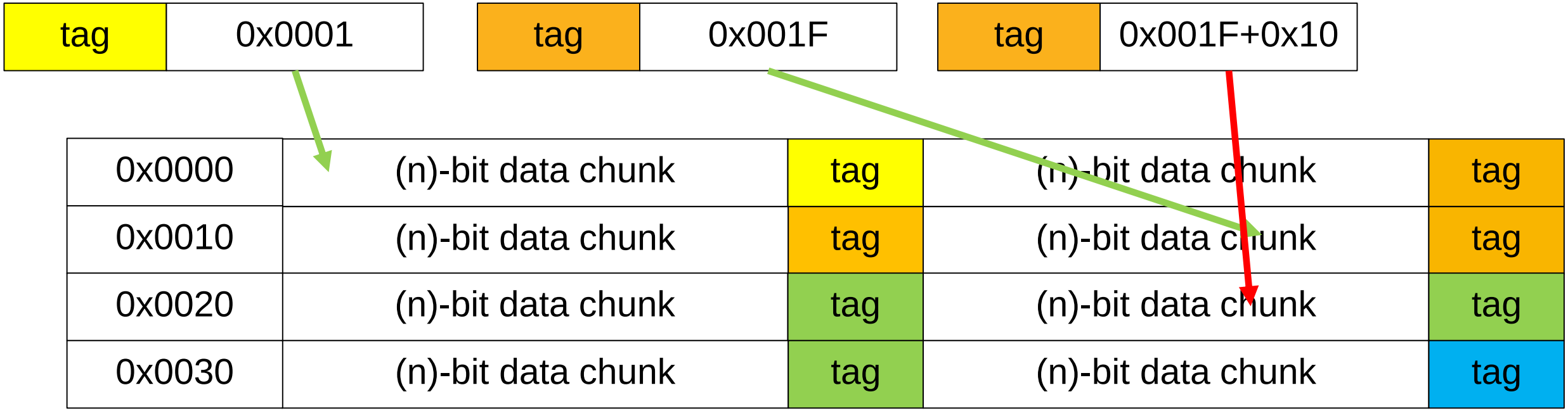
(k)-bit tag	(64-k)-bit address (typically 48 or 56 bits)

- Also data in memory has tags associated with it

(n)-bit data chunk	tag	(n)-bit data chunk	tag
(n)-bit data chunk	tag	(n)-bit data chunk	tag
(n)-bit data chunk	tag	(n)-bit data chunk	tag
(n)-bit data chunk	tag	(n)-bit data chunk	tag

Memory coloring (Accesses)

- Tags in dereferenced pointers must match the target data tag



Store tags and data separately

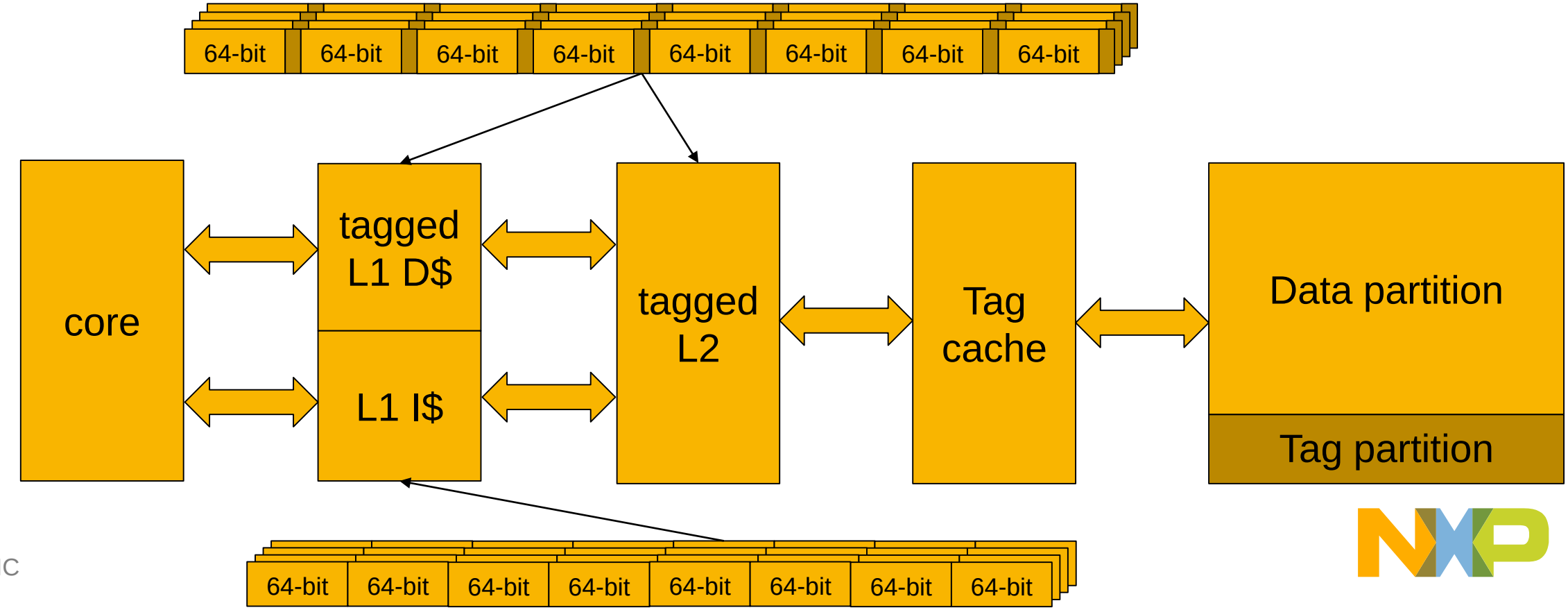
- Tags are stored in different data structure to keep the same memory layout
  - Usually a hash map
- The data granularity could be 32 bits, 64 bits or multiples thereof (overhead!)
- 16 < x < 2^11 bytes for ARM64 v8.5

0x0000	(n)-bit data chunk	(n)-bit data chunk
0x0010	(n)-bit data chunk	(n)-bit data chunk
0x0020	(n)-bit data chunk	(n)-bit data chunk
0x0030	(n)-bit data chunk	(n)-bit data chunk

tag	tag
tag	tag
tag	tag
tag	tag

Tagged memory (Data management)

- Tag cache outside last level cache
- Tags and data get spliced in cache
- Extra RAM traffic



Programming model and behavior

- A pointer needs to be tagged by setting the most significant bits
- Pointer arithmetic implements tag inheritance ({t,p} + offset → {t,p‘})

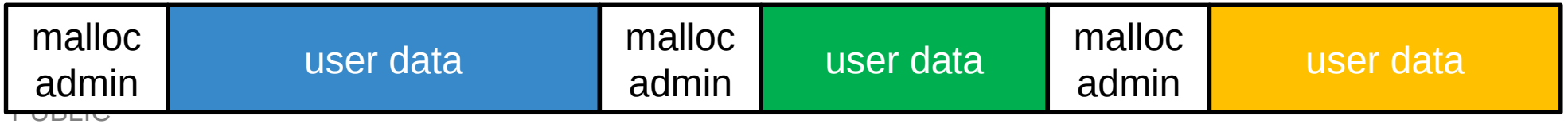
- If memory is accessed with a wrong tag an exception is triggered

- Special instructions needed
  - Freshly allocated memory needs to be tagged
  - Untagging for deallocation

Sketch of malloc and free wrappers

```
char *malloc(size_t s)
{
  p = system_malloc(s);

  p = rand_tag(p);

  for(i = 0; i < s; i += granularity)
    tag_mem(p + i);

  return p;
}
```

```
void free(void *p)
{
  p = clear_tag(p);
  s = malloc_usable_size(p);

  for(i = 0; i < s; i += granularity)
    tag_mem(p + i);

  system_free(p);
}
```



Properties

- Tags are almost always chosen randomly
  - Otherwise hard to manage
  - ARM MTE & SPARC ADI
- Extra RAM traffic, **large tag caches** can mitigate this (>=8x as many entries)
- Usually small tags for efficiency reasons
  - Smaller tag caches
  - ARM & SPARC use 4 bits
- Malicious access succeeds only with **1/16**
- Works for **attack/bug prevention and detection**

- **If you can, use it!**

CTM

This work

- Avoid storing the tags
  - No RAM traffic overhead
  - No large tag caches

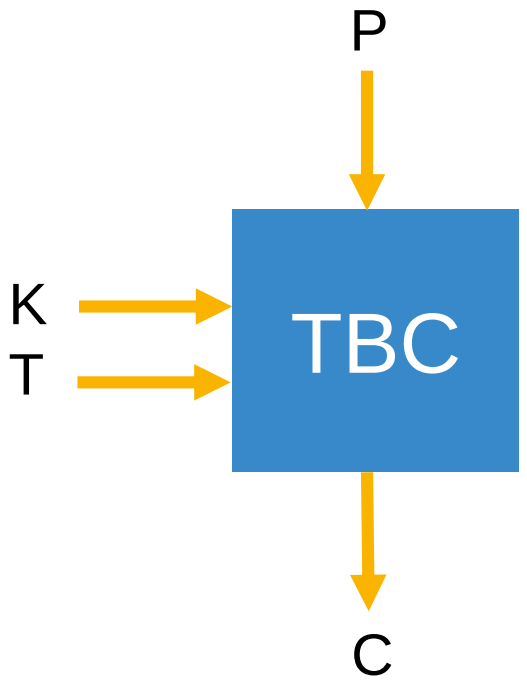
- Increase probabilistic security to >> 1/16
  - On many systems a 39-bit memory space is sufficient
  - This would actually allow for 25 tag bits

- Show results on RISC-V Rocket Core
  - FPGA implementation
  - LLVM support
  - Running Linux

Low latency tweakable block ciphers

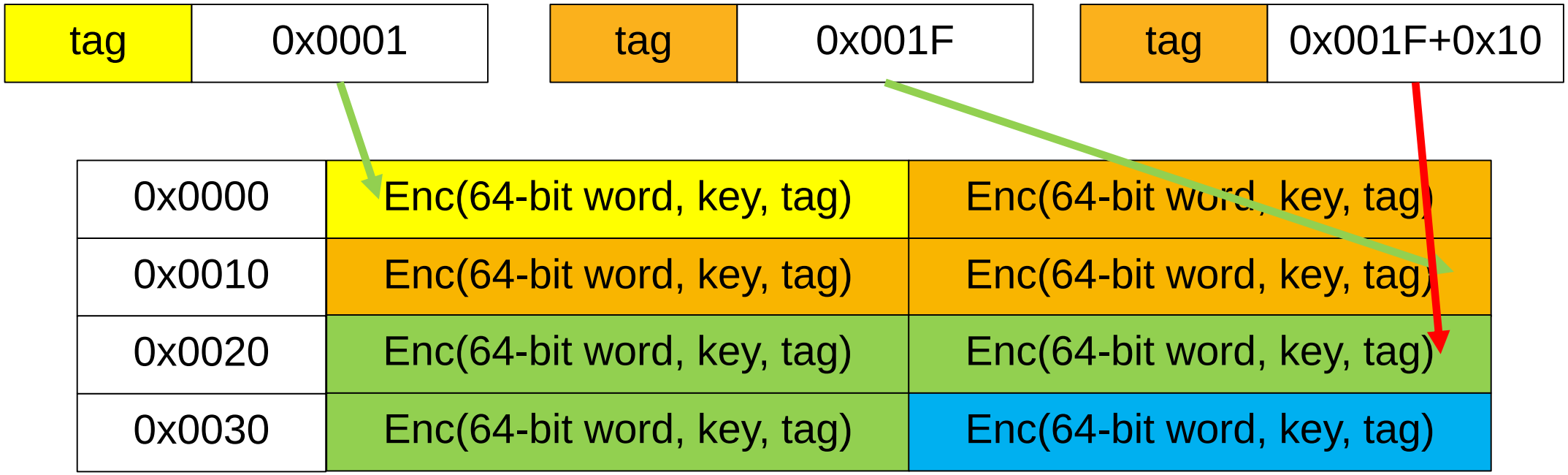
- Low latency cipher 1-2 cycles

- Tweakable low latency block cipher
  - Add additional input that “changes the cipher”
  - Either use modes or dedicated design (e.g. QARMA)



Cryptographically tagged memory (CTM)

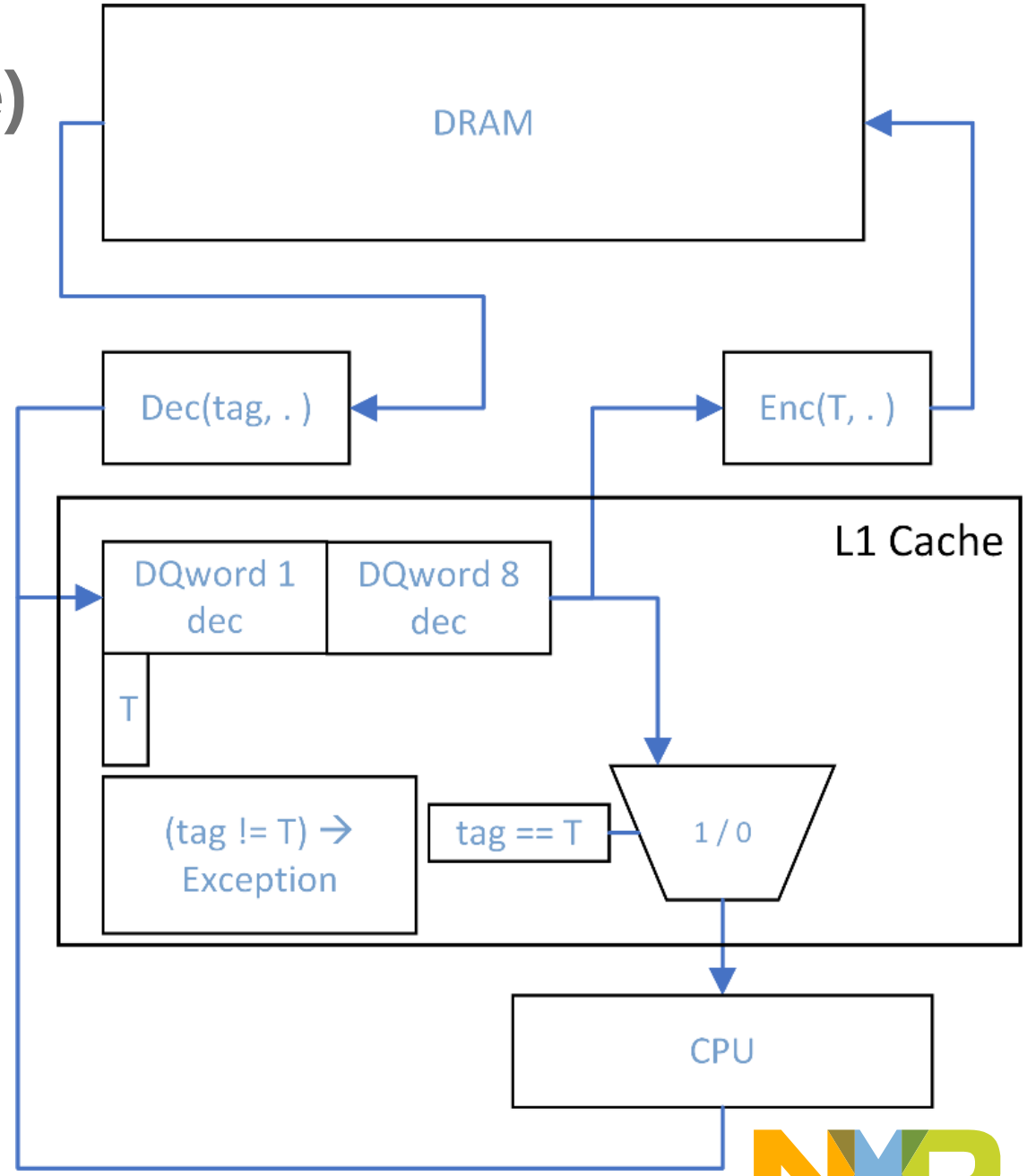
- Don‘t store tags in RAM, but use them to encrypt RAM (or L2).
- Wrong tag reads or writes garbage
- Zero (and all-one) tag means no encryption



Simple implementation (Ariane core)

- Dereference *p tagged with *tag*
  - location is not cached
  - data is decrypted using *tag*
- The used tag is stored in *T*
- Afterwards *tag* needs to match *T*
  - otherwise an exception is raised

- Are we done?
  - No, there is prefetching



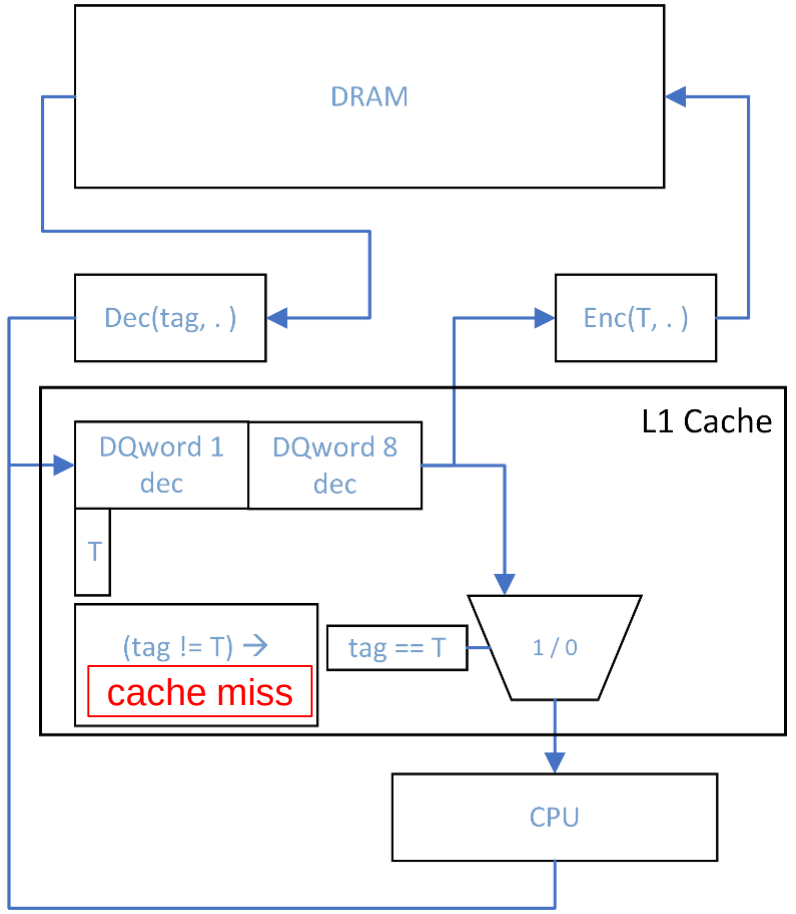
Problematic cases

w0	w1	w2	w3	w4	w5	w6	w7
T	T	T	T	T	T	T	T

- Large cache lines
  - 64 byte cache lines are usually prefetched upon first access
  - The tag used for prefetching might be wrong
- Speculative execution
  - Pointer might access invalid address → T is incorrectly stored
- Exceptions are not possible
  - No redundancy in memory → tags are not known
  - Either add redundancy or…

Probabilistic approach

- Re-encryption
  - If a tag != T, just load the machine word again
- Code pointers are very sparse in the 64-bit space
  - Random access leads to SIGSEGV
- Integer and Boolean values
  - Could be supported by compilers later



Prefetching and speculative execution: A performance problem?

- Actually does not occur that often in practice
  - Memory fragmentation
  - Supported by low overhead numbers

- Related research showed that a granularity of 16 bytes is optimal
  - malloc might anyways already use such granularity
  - 0-6% RAM overhead for heap
  - 3.5-9% RAM overhead for stack

Combine with memory encryption

- Overhead still in the higher single digit percentage

- Essentially, the only overhead comes from encryption
- Many systems already offer memory encryption (Intel, AMD, …)

- On such systems CTM comes almost for free
  - Just use a tweakable cipher
  - Extend the caches to support tags
  - Less than 1% runtime and hardware overhead

Authenticated encryption

- Probabilistic approach might also not suit everybody

- However, some systems use authenticated encryption per default (e.g. Intel SGX)

- In such setting a decryption error implies a wrong tag and we can actually trigger exceptions
  - In this setting CTM has strictly better security properties than tagged memory.

CONCLUSIONS

Conclusions

- Cryptographically tagged memory provides memory safety
- Implemented on a RISC-V Rocket platform running Linux on an FPGA
  - Memory encryption with QARMA
  - Authenticated memory encryption with ASCON
- In a 39-bit memory model, 25 bits are left for the tags
  - allows for strong security
- When combining it with encrypted memory, the overhead stays below 1% for runtime and hardware

Practical take away

- Use Sanitizers during your tests (e.g. ASAN)
- HWASAN
- ARM MTE, Pointer Authentication, Intel CET,…
- And yes, recompiling your libraries is annoying

SECURE CONNECTIONS
FOR A SMARTER WORLD