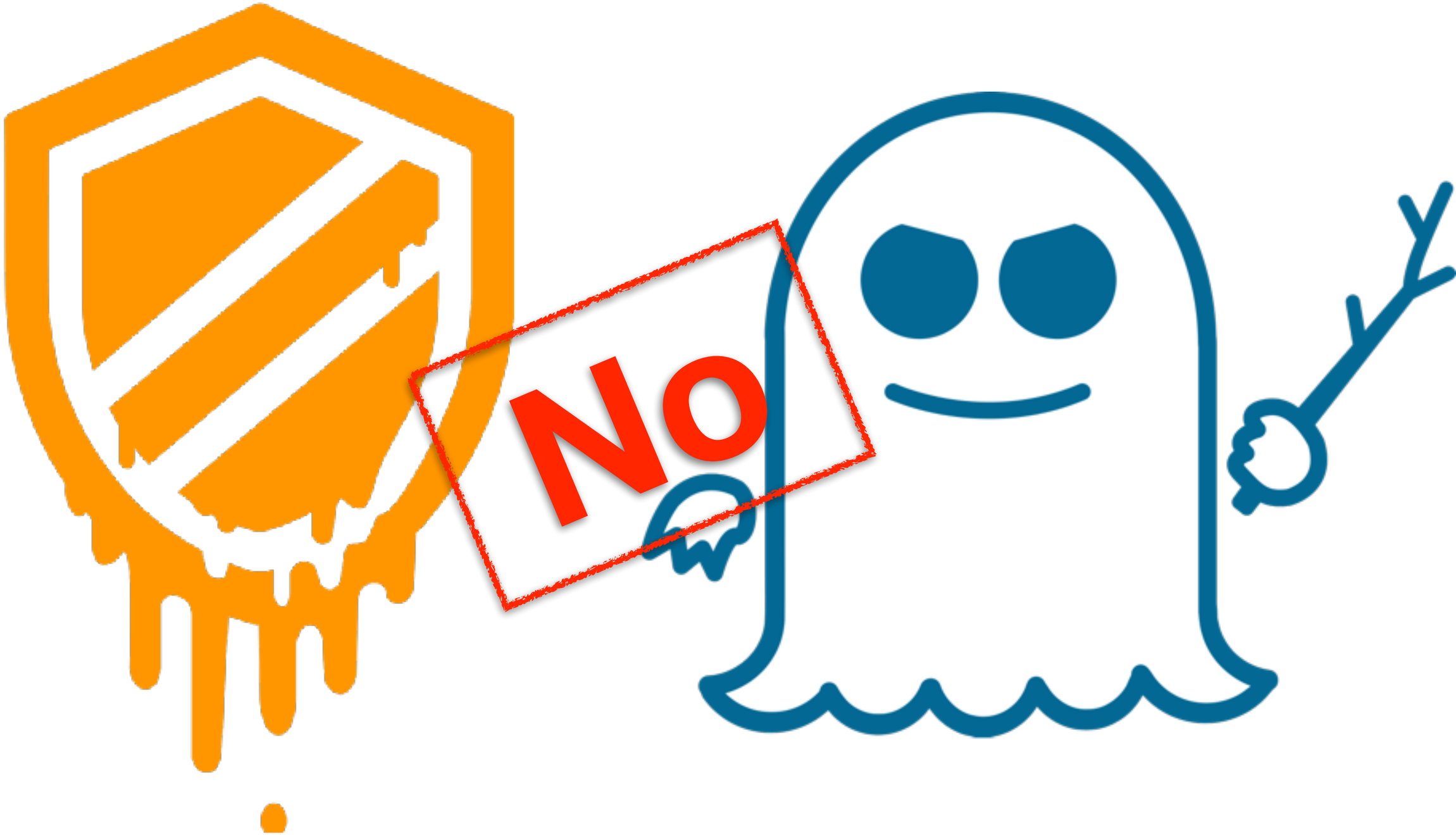
Why hardware and operating systems engineers need to talk

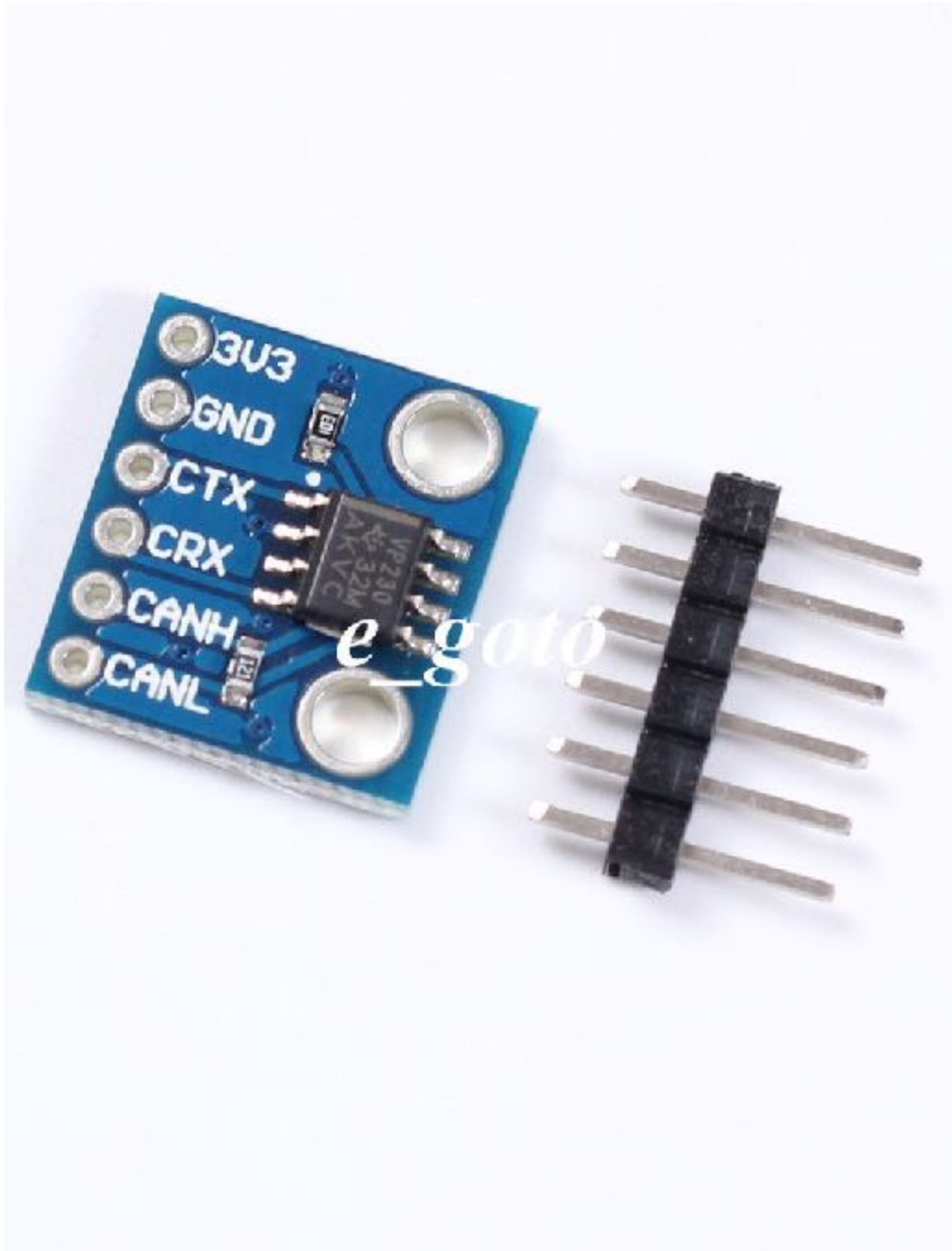
Matthias Lange, Kernkonzept GmbH, FOSDEM 2018


No

#1

i.MX6 Flexcan

Broken hardware design



(c) Aliexpress

Writing a driver for the i.MX6 Flexcan controller

- Download ~6.000 pages TRM
- Read IP core's Initialization / Application section
- Look into register definitions

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	MDIS	FRZ	RFEN	HALT	NOT_RDY	WAK_MSK	SOFT_RST	FRZ_ACK	SUPV	SLF_WAK	WRN_EN	LPM_ACK	WAK_SRC	-	SRX_DIS	IRMQ
W																
Reset	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	-		LPRIO_EN	AEN	-		IDAM		-	MAXMB						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Module Config Register

24 FRZ_ACK	…
23 SUPV	This bit configures some of the FLEXCAN registers to be either in Supervisor or User Mode. Reset value of this bit is ‘1’ …
22 SLF_WAK	…

Module Config Register

Writing a driver for the i.MX6 Flexcan controller

- Start reading again
- Disable the bit in the kernel (hack!)
- Read MCR in driver: SUPV = 0
- Next register read hangs the system

Writing a driver for the i.MX6 Flexcan controller

- Lot's of trial and error: async, external data aborts
  - Converted to pagefault by kernel
- Tried reading registers in the kernel
- Back to the manual
- CSU
  - Central Security Unit

“Instances of hackers and pirates breaking into portable devices and stealing private information and copyright content are becoming increasingly common.”

–iMX6 TRM, Chapter 20.1

Writing a driver for the i.MX6 Flexcan controller

- No publicly available document
  - Got it from the “dark net”
- Each device has an access matrix
  - Can only be changed in secure supervisor mode

Non-secure User	Non-secure Supervisor	Secure User	Secure Supervisor
None	read	read/write	read/write

Writing a driver for the i.MX6 Flexcan controller

- Set bits in u-boot bootloader
  - (because we are lucky that u-boot runs on secure side)
- Yay!
- Read/write from non-secure user
  - even with SUPV set

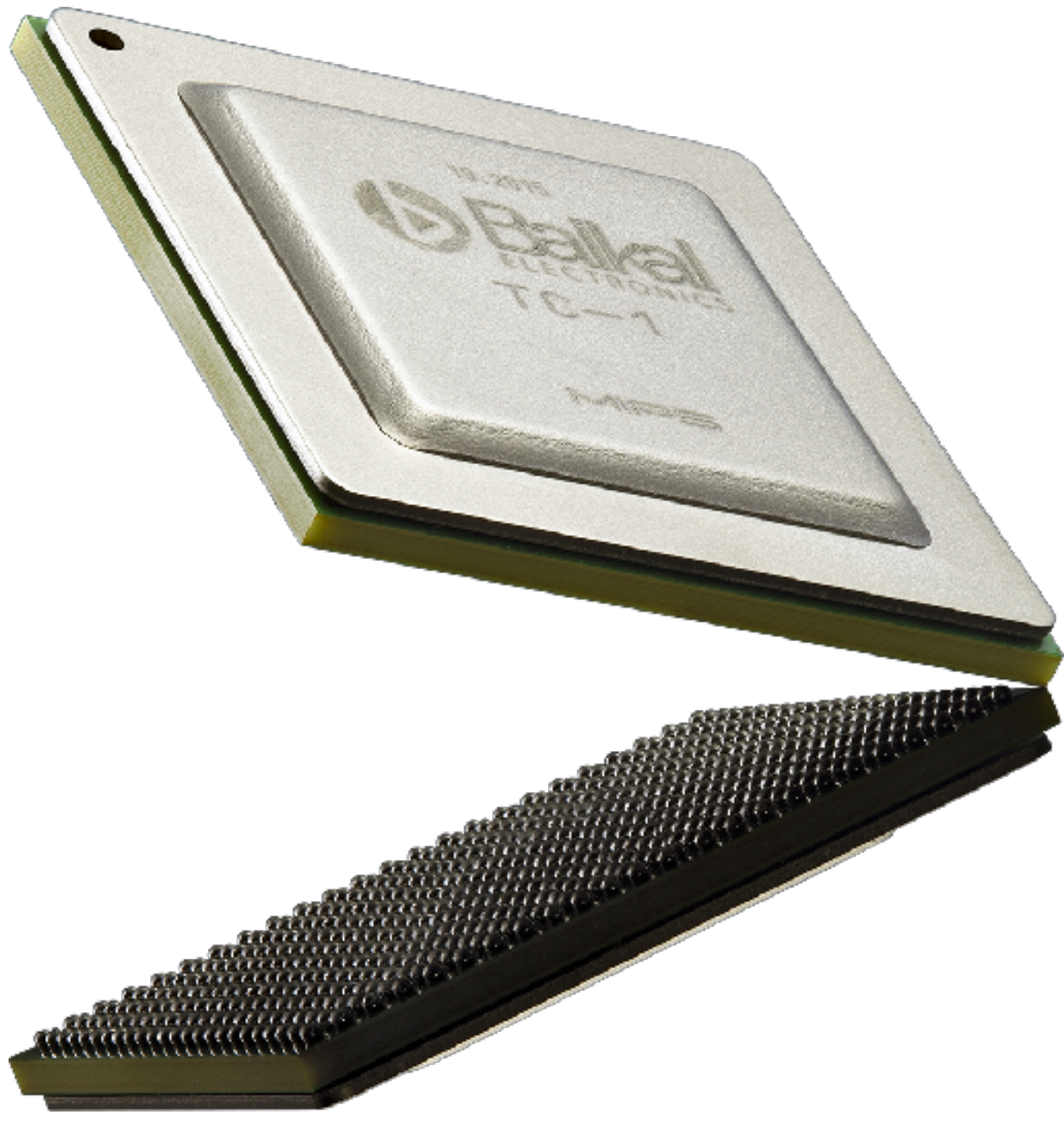
Take away #1

- Annoying and even useless security feature
- Security by obscurity
- HW design makes software implementation complex
- No obvious benefits

#2

MIPS Cache Architecture

Unfortunate hardware design



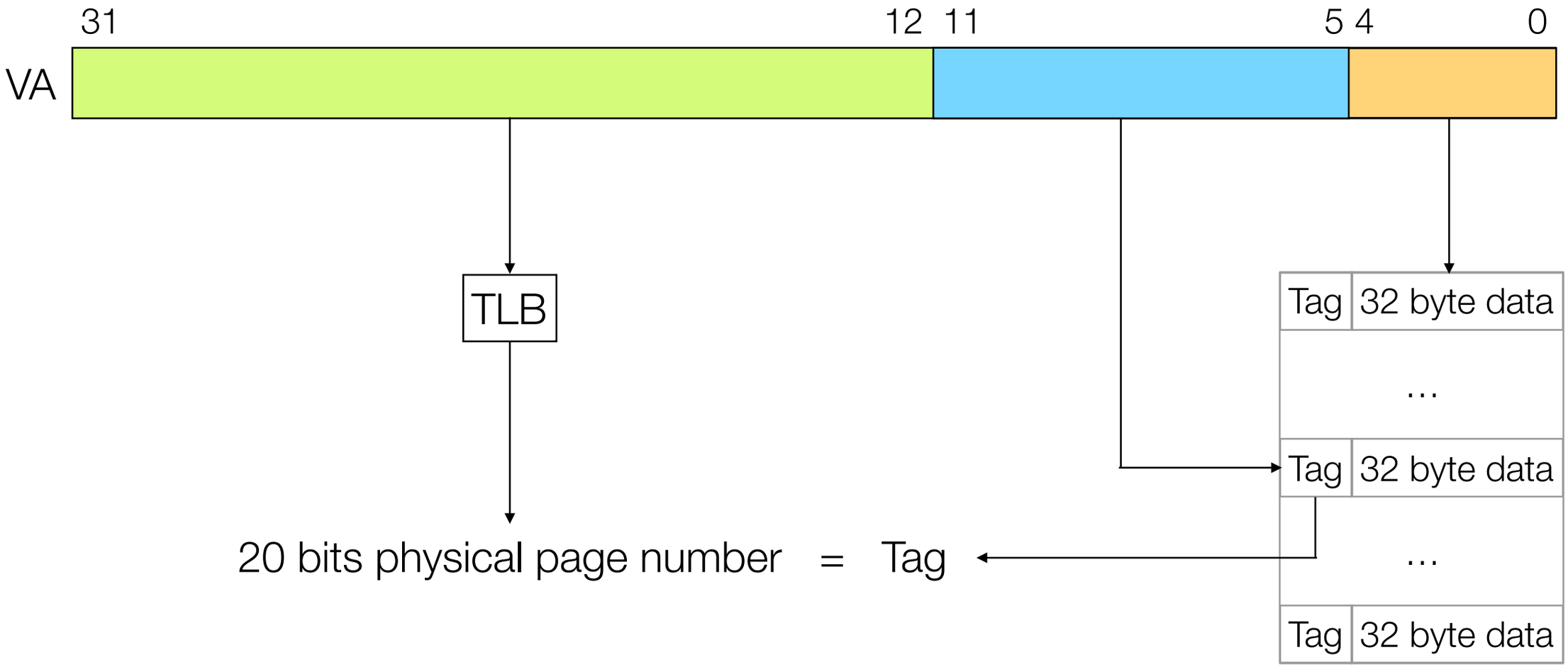
MIPS L1 Cache Design

- Separate data and instruction cache
- Virtually indexed, physically tagged (VIPT)
- 32 bytes / line
- 4-way
- 64, 128, 256 or 512 lines
    - Cache sizes of 8kb, 16kb, 32kb or 64kb

4kb way size, 4kb page size

Index bits into cache line: 5

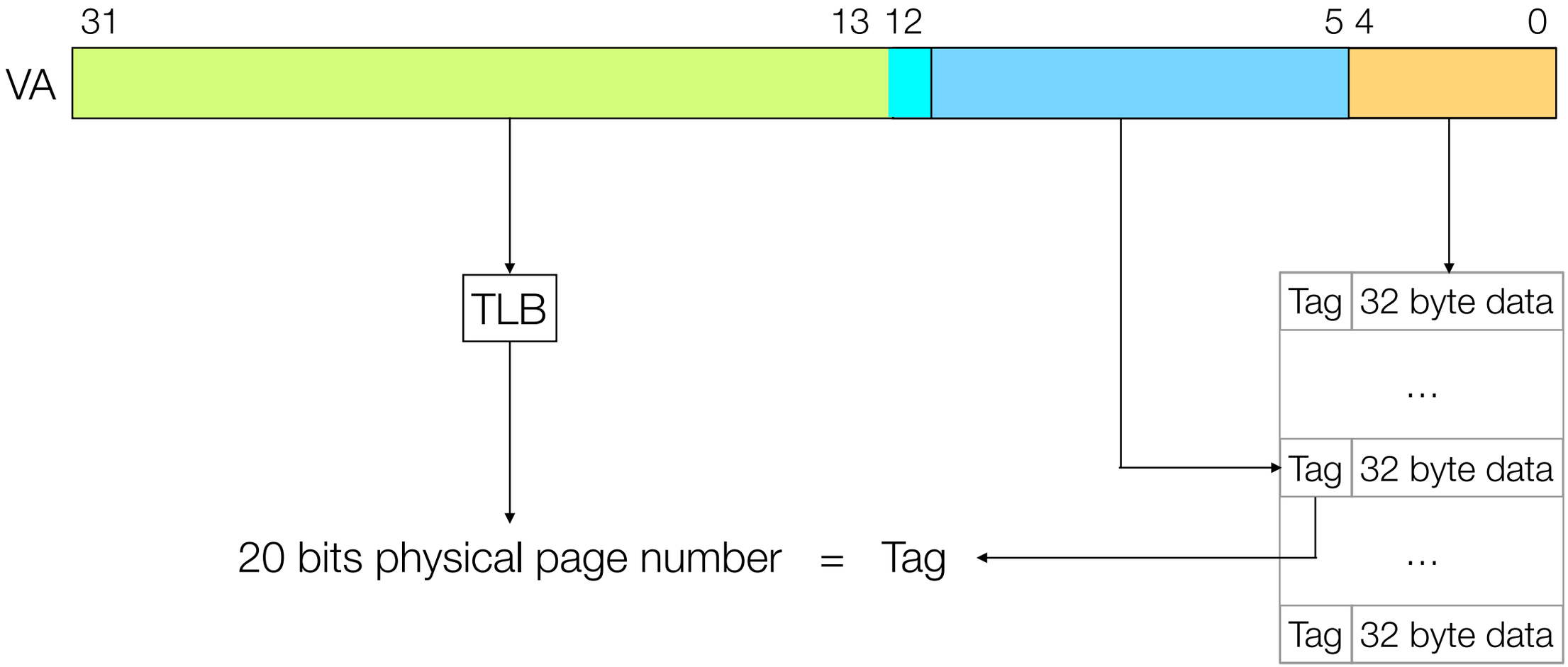
Cache line index bits: 7 (4096 / 32 = 128)



8kb way size, 4kb page size

Index bits into cache line: 5

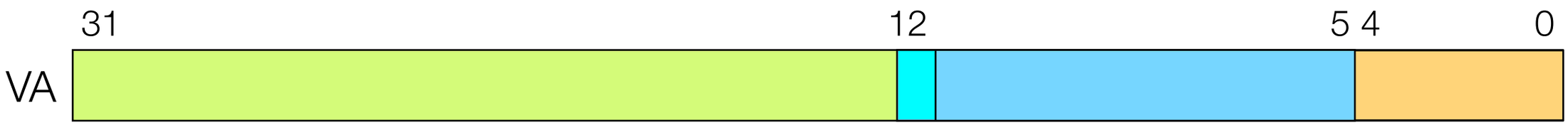
Cache line index bits: 8 (8192 / 32 = 256)



8kb way size, 4kb page size

Index bits into cache line: 5

Cache line index bits: 8 (8192 / 32 = 256)



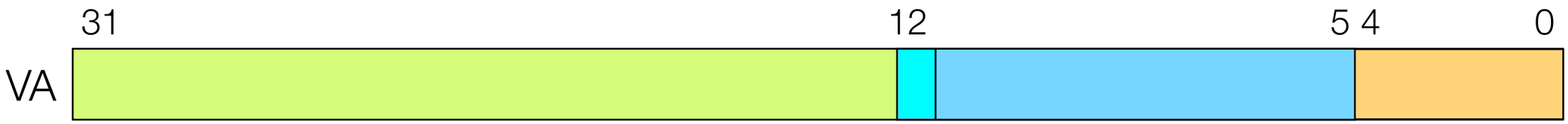
VA1 0000_0000_0000_0000_0000_0000_0000_0000

Index	Tag	Data
0	0	32 byte data
…		
128	0	32 byte data
…		
256	Tag	32 byte data

8kb way size, 4kb page size

Index bits into cache line: 5

Cache line index bits: 8 (8192 / 32 = 256)



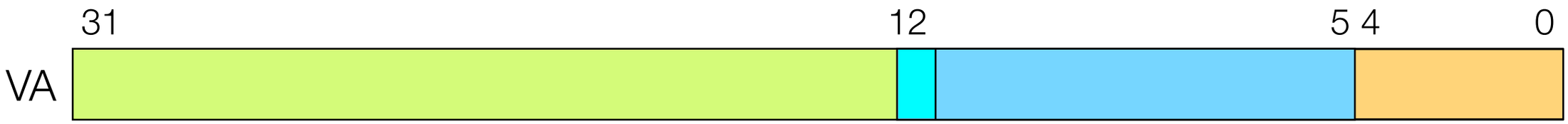
VA1 0000_0000_0000_0000_0000_0000_0000_0000

Index	Tag	Data
0	0	32 byte data
…		
128	0	32 byte data
…		
256	Tag	32 byte data

8kb way size, 4kb page size

Index bits into cache line: 5

Cache line index bits: 8 (8192 / 32 = 256)



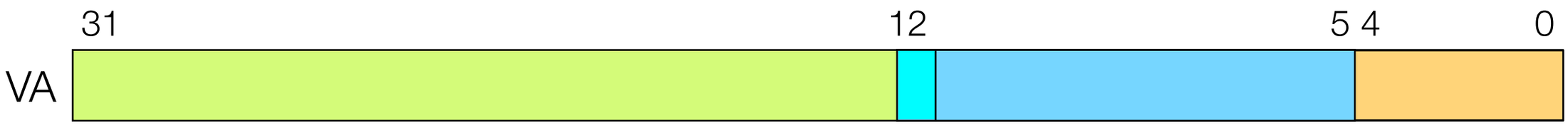
VA1 0000_0000_0000_0000_0000_0000_0000_0000

Index	Tag	Data
0	0	32 byte data
…		
128	0	32 byte data
…		
256	Tag	32 byte data

8kb way size, 4kb page size

Index bits into cache line: 5

Cache line index bits: 8 (8192 / 32 = 256)



VA1 0000_0000_0000_0000_0000_0000_0000_0000

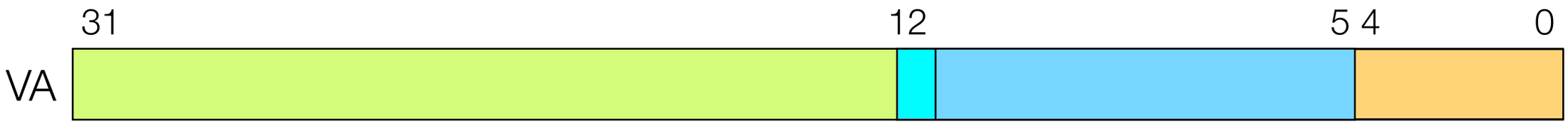
VA2 0000_0000_0000_0000_0001_0000_0000_0000

Index	Tag	Data
0	0	32 byte data
…		
128	0	32 byte data
…		
256	Tag	32 byte data

8kb way size, 4kb page size

Index bits into cache line: 5

Cache line index bits: 8 (8192 / 32 = 256)



VA1 0000_0000_0000_0000_0000_0000_0000_0000

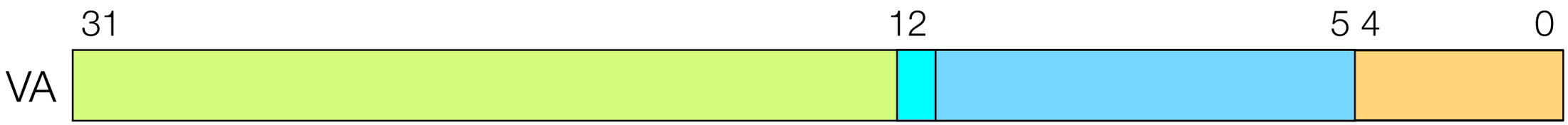
VA2 0000_0000_0000_0000_0001_0000_0000_0000

Index	Tag	Data
0	0	32 byte data
…		
128	0	32 byte data
…		
256	Tag	32 byte data

8kb way size, 4kb page size

Index bits into cache line: 5

Cache line index bits: 8 (8192 / 32 = 256)



VA1 0000_0000_0000_0000_0000_0000_0000_0000

VA2 0000_0000_0000_0000_0001_0000_0000_0000

Index	Tag	Data
0	0	32 byte data
…		
128	0	32 byte data
…		
256	2	32 byte data

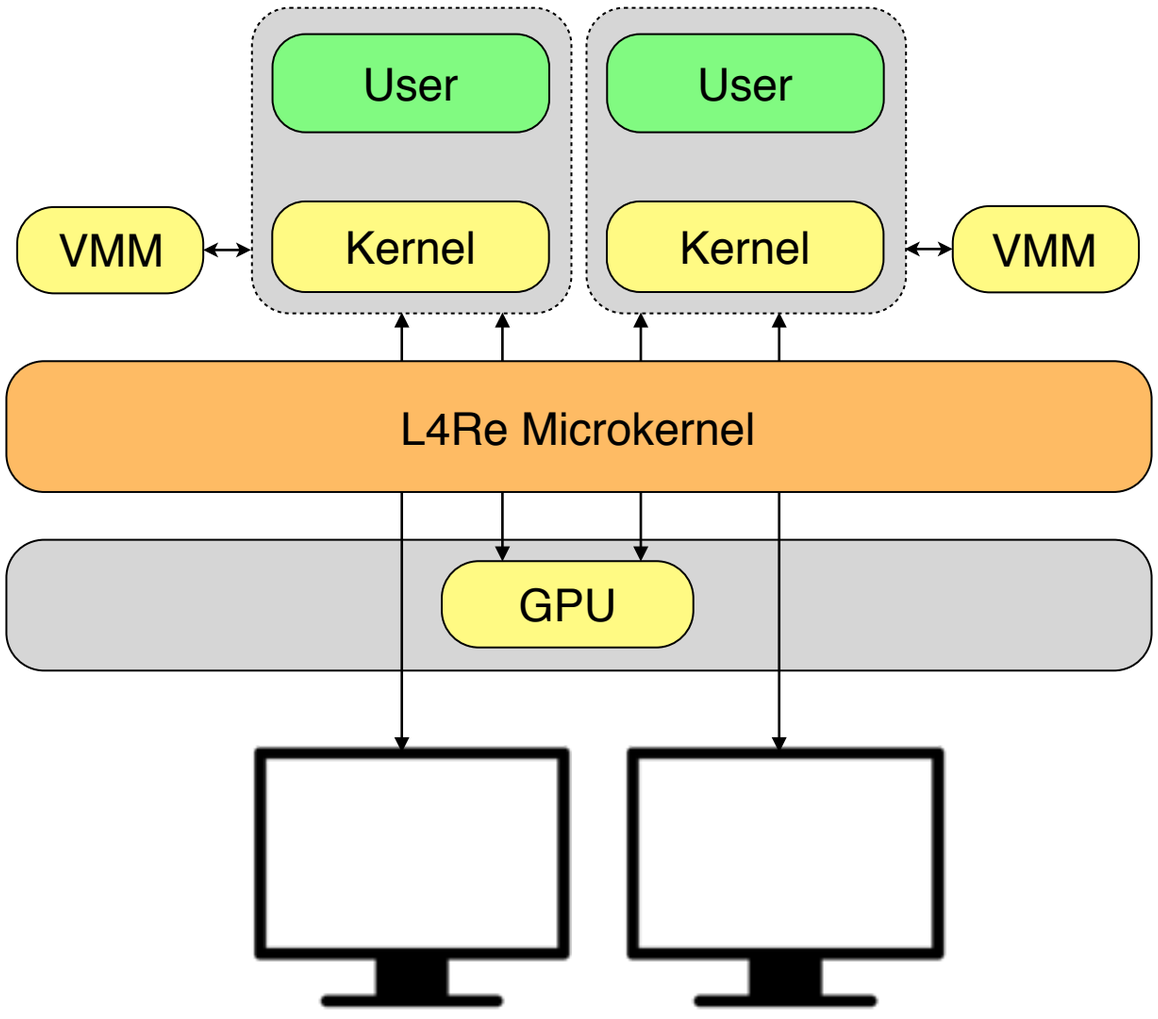
Take away #2

- Hardware implementation choices induce design complexity in software
- Hardware implementation choices may even hinder certain use-cases

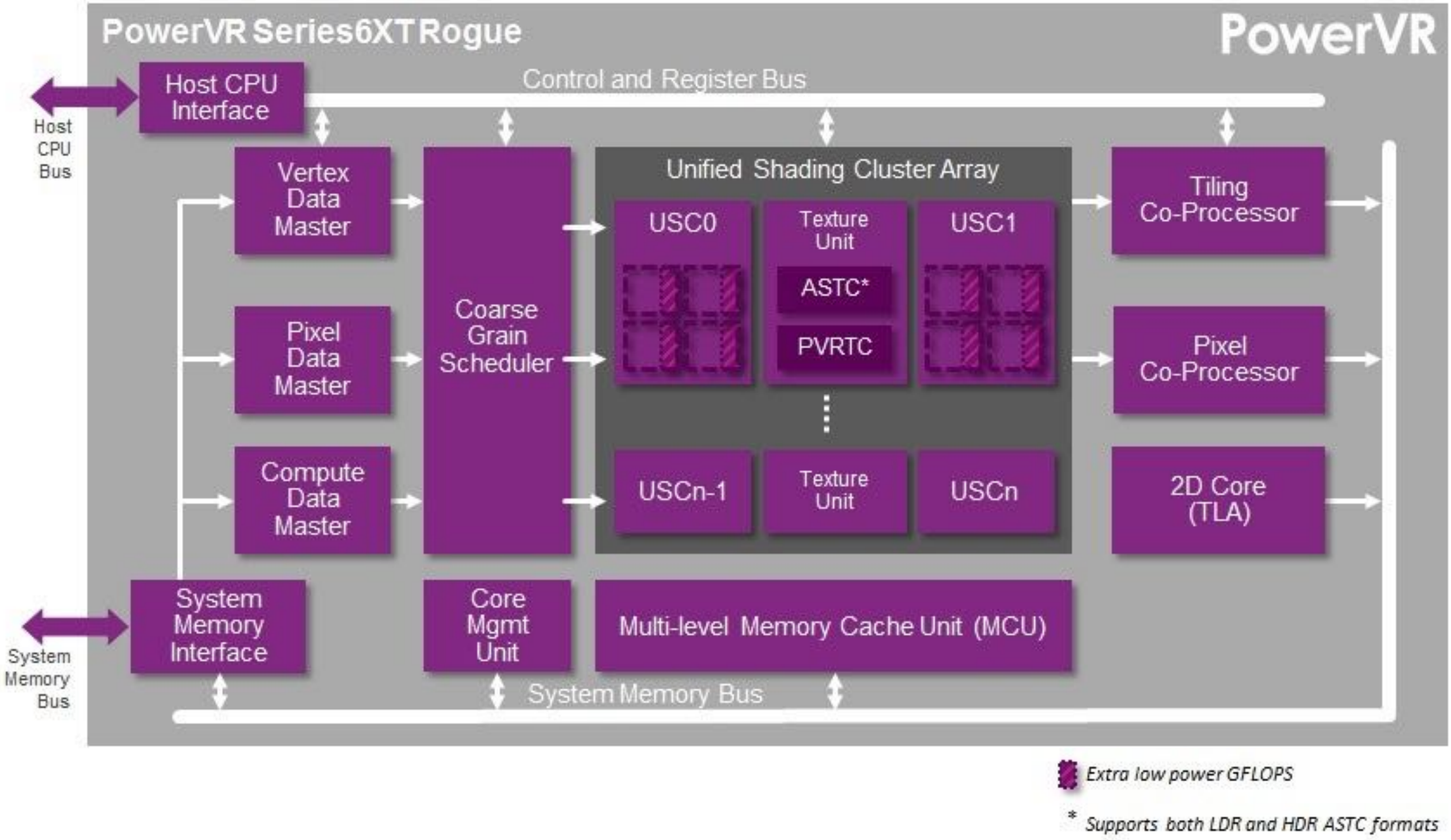
#3

Graphics Virtualization

Incomplete architecture and poor software

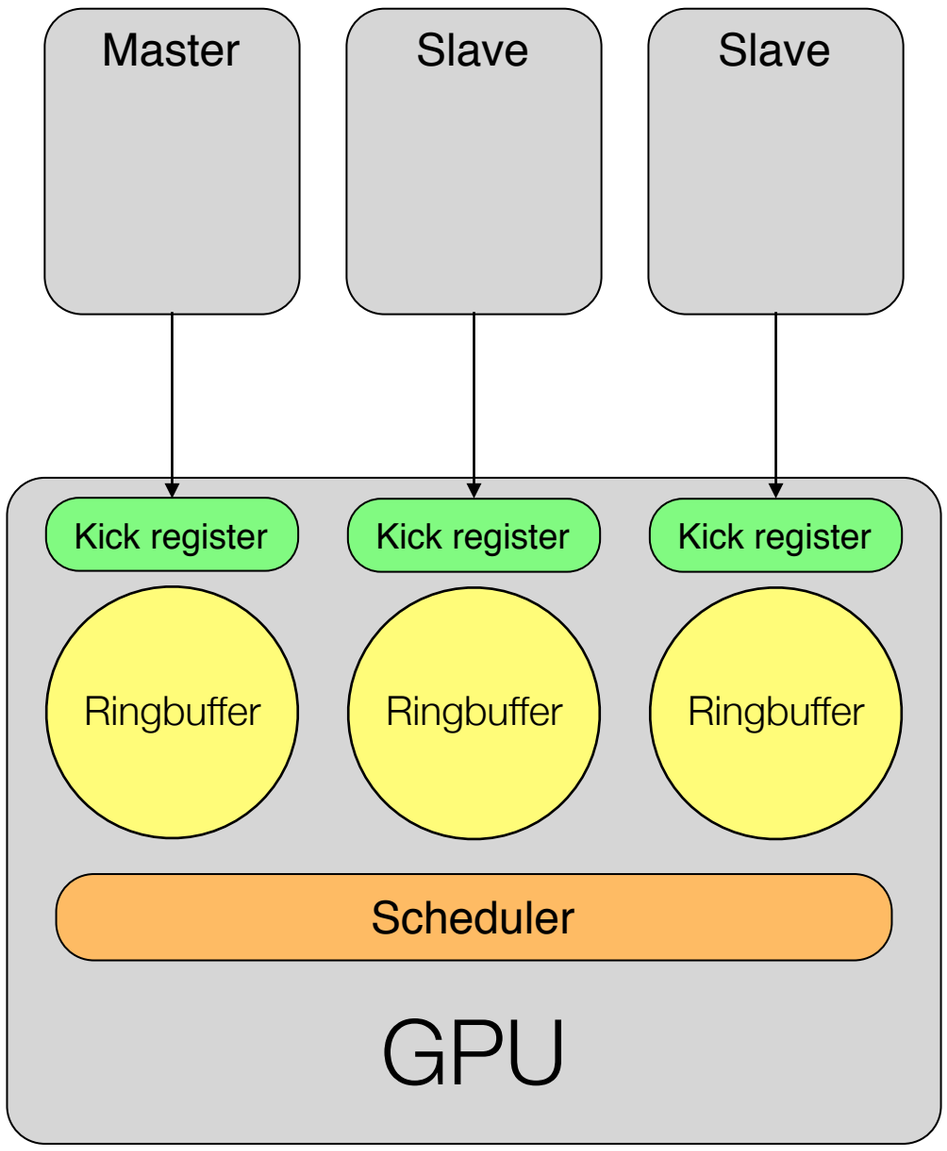


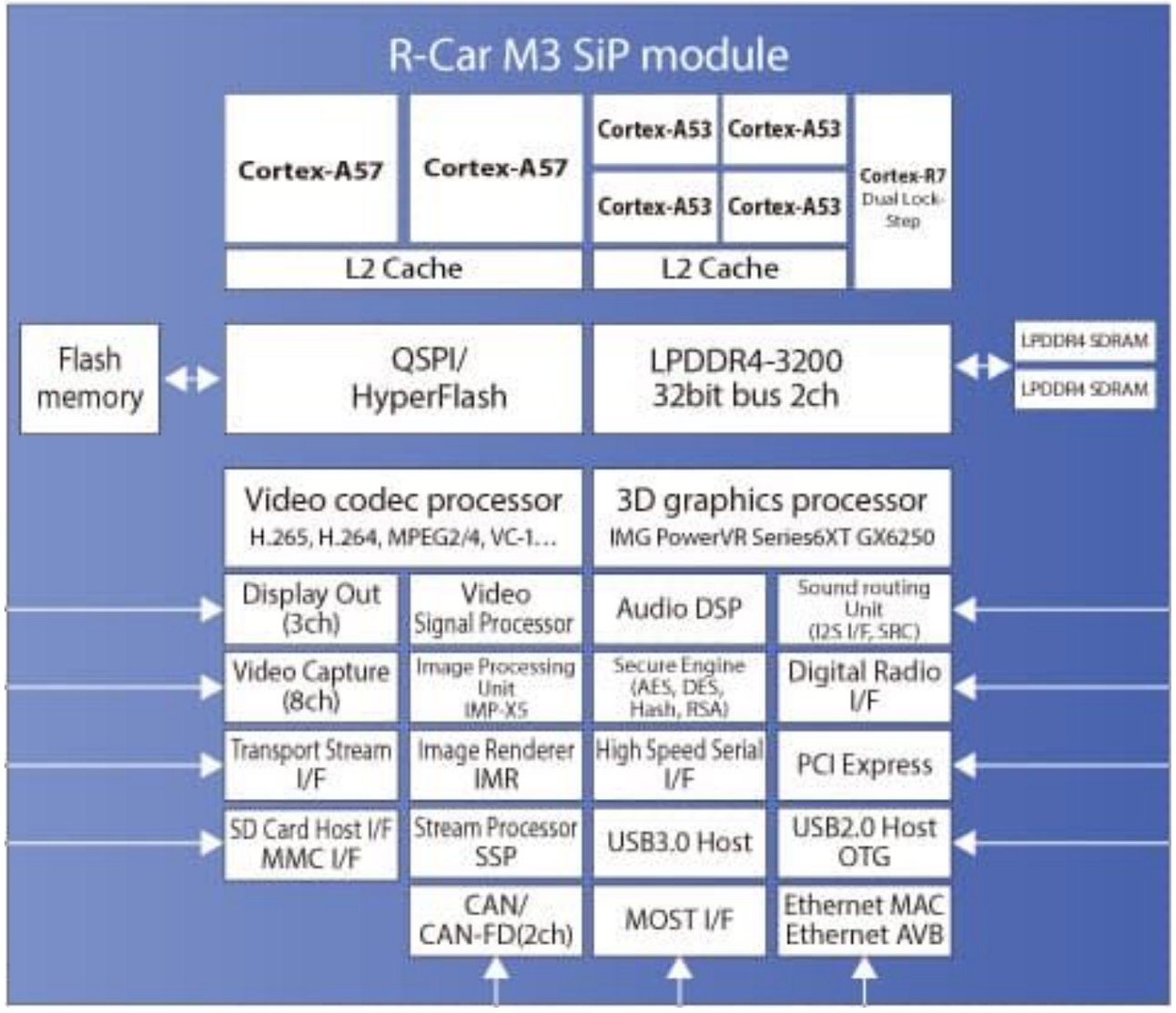
2 VMs and 2 displays



ImgTec PowerVR architecture

High-level architecture





Hardware architecture

Ready! Steady! Go!

- Run native Linux first
- Run Linux in VM
  - Full device pass-through
- Run master and slave VM
  - That’s where the fun starts!

Encountered Problems (curated list)

- Assigning dedicated display units to VM
  - Linux crashes
- Clock controller cannot be securely partitioned
  - We need to save power right?
- Graphics stack needs significant amount of memory <4GB

Take away #3

- Many things done right in hardware
  - But clock controller
  - Memory constraints
- Software / Linux drivers are not ready
- Documentation

“How many programmers does it take to change a lightbulb?”

–None! It’s a hardware problem.

The End