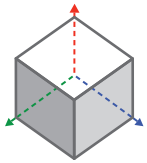

# A Free and Open ISA

## Enabling a Diversity of CPU Cores and Accelerators

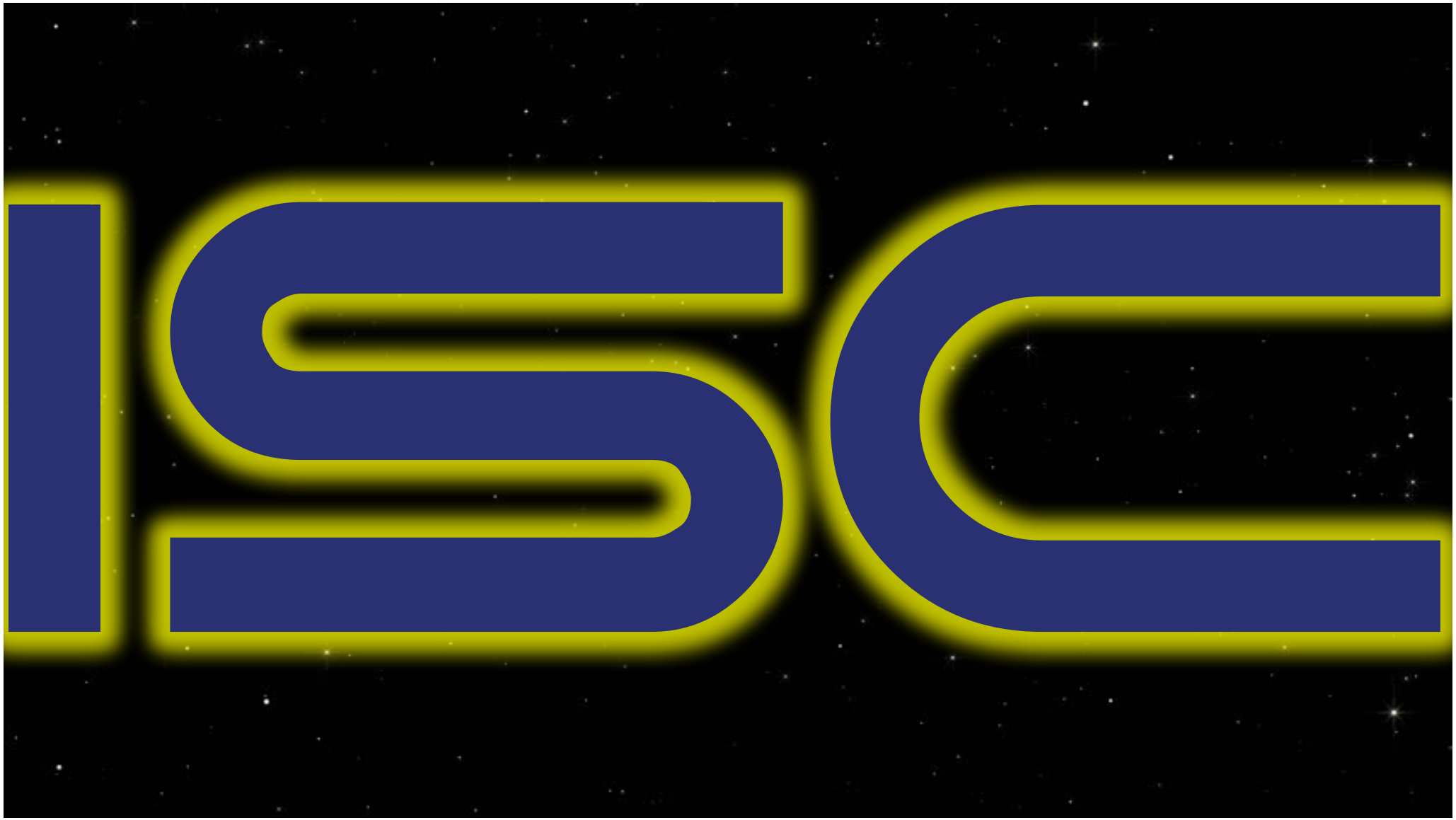
Guy Lemieux CEO



VectorBlox
embedded supercomputing

Professor

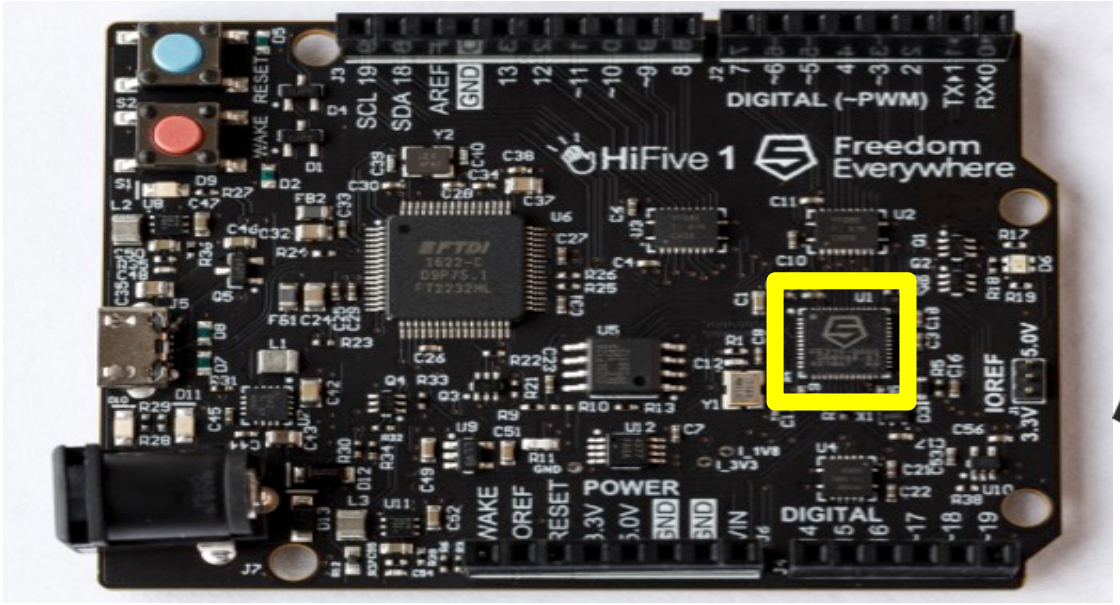
a place of mind
THE UNIVERSITY OF BRITISH COLUMBIA


ISC


RISC-V
Not so long ago in
academia far, far away,
researchers at UC Berkeley
started a 3 month project
to design a new open ISA...

# What is RISC-V?

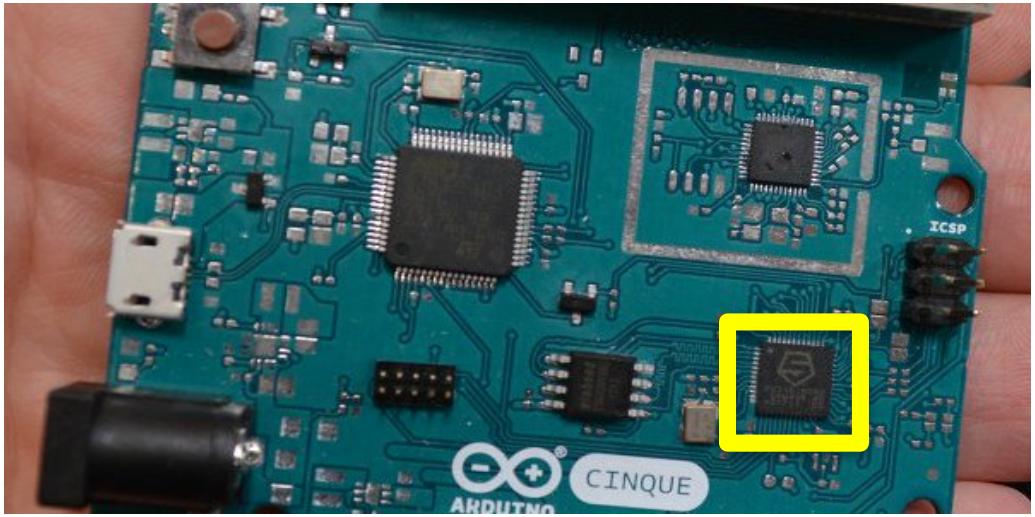
- 5th generation RISC Instruction Set Architecture (UC Berkeley)
  - Andrew Waterman, Yunsup Lee, Dave Patterson, Krste Asanovic
  - First public specification released in May 2011
- High-quality, license-free, royalty-free ISA spec.
  - Microcontrollers to supercomputers
- Standard maintained by *RISC-V Foundation*




SiFive HiFive 1

# Arduino Cinque

Announced at Bay Area Maker Faire ,
May 20, 2017



# RISC-V ISA "Green Card"

RISC-V ① ② ③ **RISC-V Reference Card ④**

## Base Integer Instructions (32|64|128)

| Category | Name | Fmt | RV{32\|64\|128}I Base | |
|---|---|---|---|---|
| **Loads** | Load Byte | I | LB | rd,rs1,imm |
| | Load Halfword | I | LH | rd,rs1,imm |
| | Load Word | I | L{W\|D\|Q} | rd,rs1,imm |
| | Load Byte Unsigned | I | LBU | rd,rs1,imm |
| | Load Half Unsigned | I | L{H\|W\|D}U | rd,rs1,imm |
| **Stores** | Store Byte | S | SB | rs1,rs2,imm |
| | Store Halfword | S | SH | rs1,rs2,imm |
| | Store Word | S | S{W\|D\|Q} | rs1,rs2,imm |
| **Shifts** | Shift Left | R | SLL{\|W\|D} | rd,rs1,rs2 |
| | Shift Left Immediate | I | SLLI{\|W\|D} | rd,rs1,shamt |
| | Shift Right | R | SRL{\|W\|D} | rd,rs1,rs2 |
| | Shift Right Immediate | I | SRLI{\|W\|D} | rd,rs1,shamt |
| | Shift Right Arithmetic | R | SRA{\|W\|D} | rd,rs1,rs2 |
| | Shift Right Arith Imm | I | SRAI{\|W\|D} | rd,rs1,shamt |
| **Arithmetic** | ADD | R | ADD{\|W\|D} | rd,rs1,rs2 |
| | ADD Immediate | I | ADDI{\|W\|D} | rd,rs1,imm |
| | SUBtract | R | SUB{\|W\|D} | rd,rs1,rs2 |
| | Load Upper Imm | U | LUI | rd,imm |
| | Add Upper Imm to PC | U | AUIPC | rd,imm |
| **Logical** | XOR | R | XOR | rd,rs1,rs2 |
| | XOR Immediate | I | XORI | rd,rs1,imm |
| | OR | R | OR | rd,rs1,rs2 |
| | OR Immediate | I | ORI | rd,rs1,imm |
| | AND | R | AND | rd,rs1,rs2 |
| | AND Immediate | I | ANDI | rd,rs1,imm |
| **Compare** | Set < | R | SLT | rd,rs1,rs2 |
| | Set < Immediate | I | SLTI | rd,rs1,imm |
| | Set < Unsigned | R | SLTU | rd,rs1,rs2 |
| | Set < Imm Unsigned | I | SLTIU | rd,rs1,imm |
| **Branches** | Branch = | SB | BEQ | rs1,rs2,imm |
| | Branch ≠ | SB | BNE | rs1,rs2,imm |
| | Branch < | SB | BLT | rs1,rs2,imm |
| | Branch ≥ | SB | BGE | rs1,rs2,imm |
| | Branch < Unsigned | SB | BLTU | rs1,rs2,imm |
| | Branch ≥ Unsigned | SB | BGEU | rs1,rs2,imm |
| **Jump & Link** | J&L | UJ | JAL | rd,imm |
| | Jump & Link Register | I | JALR | rd,rs1,imm |
| **Synch** | Synch thread | I | FENCE | |
| | Synch Instr & Data | I | FENCE.I | |
| **System** | System CALL | I | SCALL | |
| | System BREAK | I | SBREAK | |
| **Counters** | ReaD CYCLE | I | RDCYCLE | rd |
| | ReaD CYCLE upper Half | I | RDCYCLEH | rd |
| | ReaD TIME | I | RDTIME | rd |
| | ReaD TIME upper Half | I | RDTIMEH | rd |
| | ReaD INSTR RETired | I | RDINSTRET | rd |
| | ReaD INSTR upper Half | I | RDINSTRETH | rd |

## RV Privileged Instructions (32|64|128)

| Category | Name | Fmt | RV mnemonic | |
|---|---|---|---|---|
| **CSR Access** | Atomic R/W | R | CSRRW | rd,csr,rs1 |
| | Atomic Read & Set Bit | R | CSRRS | rd,csr,rs1 |
| | Atomic Read & Clear Bit | R | CSRRC | rd,csr,rs1 |
| | Atomic R/W Imm | R | CSRRWI | rd,csr,imm |
| | Atomic Read & Set Bit Imm | R | CSRRSI | rd,csr,imm |
| | Atomic Read & Clear Bit Imm | R | CSRRCI | rd,csr,imm |
| **Change Level** | Env. Call | R | ECALL | |
| | Environment Breakpoint | R | EBREAK | |
| | Environment Return | R | ERET | |
| **Trap Redirect** | to Superviso | R | MRTS | |
| | Redirect Trap to Hypervisor | R | MRTH | |
| | Hypervisor Trap to Supervisor | R | HRTS | |
| **Interrupt** | Wait for Interrup | R | WFI | |
| **MMU** | Supervisor FENCE | R | SFENCE.VM | rs1 |

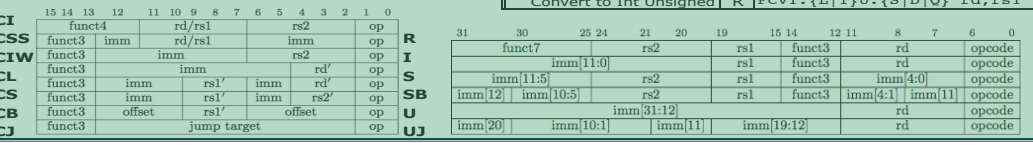
## Optional Multiply-Divide Extension: RV32M

| Category | Name | Fmt | RV32M (Mult-Div) | |
|---|---|---|---|---|
| **Multiply** | MULtiply | R | MUL{\|W\|D} | rd,rs1,rs2 |
| | MULtiply upper Half | R | MULH | rd,rs1,rs2 |
| | MULtiply Half Sign/Uns | R | MULHSU | rd,rs1,rs2 |
| | MULtiply upper Half Uns | R | MULHU | rd,rs1,rs2 |
| **Divide** | DIVide | R | DIV{\|W\|D} | rd,rs1,rs2 |
| | DIVide Unsigned | R | DIVU | rd,rs1,rs2 |
| **Remainder** | REMainder | R | REM{\|W\|D} | rd,rs1,rs2 |
| | REMainder Unsigned | R | REMU{\|W\|D} | rd,rs1,rs2 |

## Optional Atomic Instruction Extension: RVA

| Category | Name | Fmt | RV{32\|64\|128}A (Atomic) | |
|---|---|---|---|---|
| **Load** | Load Reserved | R | LR.{W\|D\|Q} | rd,rs1 |
| **Store** | Store Conditiona | R | SC.{W\|D\|Q} | rd,rs1,rs2 |
| **Swap** | SWAP | R | AMOSWAP.{W\|D\|Q} | rd,rs1,rs2 |
| **Add** | ADD | R | AMOADD.{W\|D\|Q} | rd,rs1,rs2 |
| **Logical** | XOR | R | AMOXOR.{W\|D\|Q} | rd,rs1,rs2 |
| | AND | R | AMOAND.{W\|D\|Q} | rd,rs1,rs2 |
| | OR | R | AMOOR.{W\|D\|Q} | rd,rs1,rs2 |
| **Min/Max** | MINimum | R | AMOMIN.{W\|D\|Q} | rd,rs1,rs2 |
| | MAXimum | R | AMOMAX.{W\|D\|Q} | rd,rs1,rs2 |
| | MINimum Unsigned | R | AMOMINU.{W\|D\|Q} | rd,rs1,rs2 |
| | MAXimum Unsigned | R | AMOMAXU.{W\|D\|Q} | rd,rs1,rs2 |

## 16-bit (RVC) and 32-bit Instruction Formats




## 3 Optional FP Extensions: RV32{F|D|Q}

| Category | Name | Fmt | RV{F\|D\|Q} (HP/SP,DP,QP) | |
|---|---|---|---|---|
| **Load** | Load | I | FL{W,D,Q} | rd,rs1,imm |
| **Store** | Store | S | FS{W,D,Q} | rs1,rs2,imm |
| **Arithmetic** | ADD | R | FADD.{S\|D\|Q} | rd,rs1,rs2 |
| | SUBtract | R | FSUB.{S\|D\|Q} | rd,rs1,rs2 |
| | MULtiply | R | FMUL.{S\|D\|Q} | rd,rs1,rs2 |
| | DIVide | R | FDIV.{S\|D\|Q} | rd,rs1,rs2 |
| | SQuare RooT | R | FSQRT.{S\|D\|Q} | rd,rs1 |
| **Mul-Add** | Multiply-ADD | R | FMADD.{S\|D\|Q} | rd,rs1,rs2,rs3 |
| | Multiply-SUBtract | R | FMSUB.{S\|D\|Q} | rd,rs1,rs2,rs3 |
| | Negative Multiply-SUBtract | R | FMNSUB.{S\|D\|Q} | rd,rs1,rs2,rs3 |
| | Negative Multiply-ADD | R | FMNADD.{S\|D\|Q} | rd,rs1,rs2,rs3 |
| **Sign Inject** | SiGN source | R | FSGNJ.{S\|D\|Q} | rd,rs1,rs2 |
| | Negative SiGN source | R | FSGNJN.{S\|D\|Q} | rd,rs1,rs2 |
| | Xor SiGN source | R | FSGNJX.{S\|D\|Q} | rd,rs1,rs2 |
| **Min/Max** | MINimum | R | FMIN.{S\|D\|Q} | rd,rs1,rs2 |
| | MAXimum | R | FMAX.{S\|D\|Q} | rd,rs1,rs2 |
| **Compare** | Compare Float = | R | FEQ.{S\|D\|Q} | rd,rs1,rs2 |
| | Compare Float < | R | FLT.{S\|D\|Q} | rd,rs1,rs2 |
| | Compare Float ≤ | R | FLE.{S\|D\|Q} | rd,rs1,rs2 |
| **Categorize** | Classify Type | R | FCLASS.{S\|D\|Q} | rd,rs1 |
| **Move** | Move from Integer | R | FMV.S.X | rd,rs1 |
| | Move to Integer | R | FMV.X.S | rd,rs1 |
| **Convert** | Convert from Int | R | FCVT.{S\|D\|Q}.W | rd,rs1 |
| | Convert from Int Unsigned | R | FCVT.{S\|D\|Q}.WU | rd,rs1 |
| | Convert to Int | R | FCVT.W.{S\|D\|Q} | rd,rs1 |
| | Convert to Int Unsigned | R | FCVT.WU.{S\|D\|Q} | rd,rs1 |
| **Configuration** | Read Stat | R | FRCSR | rd |
| | Read Rounding Mode | R | FRRM | rd |
| | Read Flags | R | FRFLAGS | rd |
| | Swap Status Reg | R | FSCSR | rd,rs1 |
| | Swap Rounding Mode | R | FSRM | rd,rs1 |
| | Swap Flags | R | FSFLAGS | rd,rs1 |
| | Swap Rounding Mode Imm | I | FSRMI | rd,imm |
| | Swap Flags Imm | I | FSFLAGSI | rd,imm |

## 3 Optional FP Extensions: RV{64|128}{F|D|Q}

| Category | Name | Fmt | RV{F\|D\|Q} (HP/SP,DP,QP) | |
|---|---|---|---|---|
| **Move** | Move from Integer | R | FMV.{D\|Q}.X | rd,rs1 |
| | Move to Integer | R | FMV.X.{D\|Q} | rd,rs1 |
| **Convert** | Convert from Int | R | FCVT.{S\|D\|Q}.{L\|T} | rd,rs1 |
| | Convert from Int Unsigned | R | FCVT.{S\|D\|Q}.{L\|T}U | rd,rs1 |
| | Convert to Int | R | FCVT.{L\|T}.{S\|D\|Q} | rd,rs1 |
| | Convert to Int Unsigned | R | FCVT.{L\|T}U.{S\|D\|Q} | rd,rs1 |

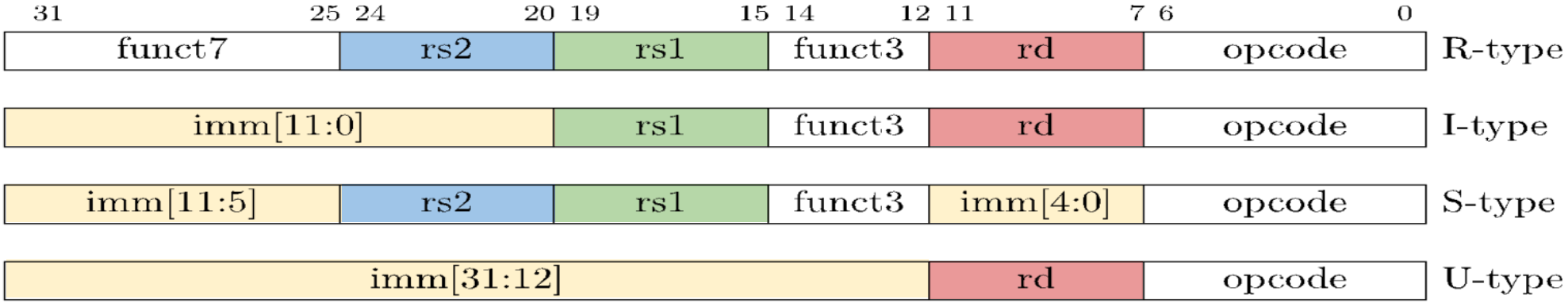
## Optional Compressed Instructions: RVC

| Category | Name | Fmt | RVC | |
|---|---|---|---|---|
| **Loads** | Load Word | CL | C.LW | rd′,rs1′,imm |
| | Load Word SP | CI | C.LWSP | rd,imm |
| | Load Double | CL | C.LD | rd′,rs1′,imm |
| | Load Double SP | CI | C.LWSP | rd,imm |
| | Load Quad | CL | C.LQ | rd′,rs1′,imm |
| | Load Quad SP | CI | C.LQSP | rd,imm |
| | Load Byte Unsigned | CL | C.LBU | rd′,rs1′,imm |
| | Float Load Word | CL | C.FLW | rd′,rs1′,imm |
| | Float Load Double | CL | C.FLD | rd′,rs1′,imm |
| | Float Load Word SP | CI | C.FLWSP | rd,imm |
| | Float Load Double SP | CI | C.FLDSP | rd,imm |
| **Stores** | Store Word | CS | C.SW | rs1′,rs2′,imm |
| | Store Word SP | CSS | C.SWSP | rs2,imm |
| | Store Double | CS | C.SD | rs1′,rs2′,imm |
| | Store Double SP | CSS | C.SDSP | rs2,imm |
| | Store Quad | CS | C.SQ | rs1′,rs2′,imm |
| | Store Quad SP | CSS | C.SQSP | rs2,imm |
| | Float Store Word | CSS | C.FSW | rd′,rs1′,imm |
| | Float Store Double | CSS | C.FSD | rd′,rs1′,imm |
| | Float Store Word SP | CSS | C.FSWSP | rd,imm |
| | Float Store Double SP | CSS | C.FSDSP | rd,imm |
| **Arithmetic** | ADD | CR | C.ADD | rd,rs1 |
| | ADD Word | CR | C.ADDW | rd',rs2' |
| | ADD Immediate | CI | C.ADDI | rd,imm |
| | ADD Word Imm | CI | C.ADDIW | rd,imm |
| | ADD SP Imm * 16 | CI | C.ADDI16SP | x0,imm |
| | ADD SP Imm * 4 | CIW | C.ADDI4SPN | rd',imm |
| | Load Immediate | CI | C.LI | rd,imm |
| | Load Upper Imm | CI | C.LUI | rd,imm |
| | MoVe | CR | C.MV | rd,rs1 |
| | SUB | CR | C.SUB | rd',rs2' |
| | SUB Word | CR | C.SUBW | rd',rs2' |
| **Logical** | XOR | CS | C.XOR | rd',rs2' |
| | OR | CS | C.OR | rd',rs2' |
| | AND | CS | C.AND | rd',rs2' |
| | AND Immediate | CB | C.ANDI | rd',rs2' |
| **Shifts** | Shift Left Imm | CI | C.SLLI | rd,imm |
| | Shift Right Immediate | CB | C.SRLI | rd',imm |
| | Shift Right Arith Imm | CB | C.SRAI | rd',imm |
| **Branches** | Branch=0 | CB | C.BEQZ | rs1′,imm |
| | Branch≠0 | CB | C.BNEZ | rs1′,imm |
| **Jump** | Jump | CJ | C.J | imm |
| | Jump Register | CR | C.JR | rd,rs1 |
| **Jump & Link** | J&L | CJ | C.JAL | imm |
| | Jump & Link Register | CR | C.JALR | rs1 |
| **System** | Env. BREAK | CI | C.EBREAK | |

# RISC-V Base + Standard Extensions

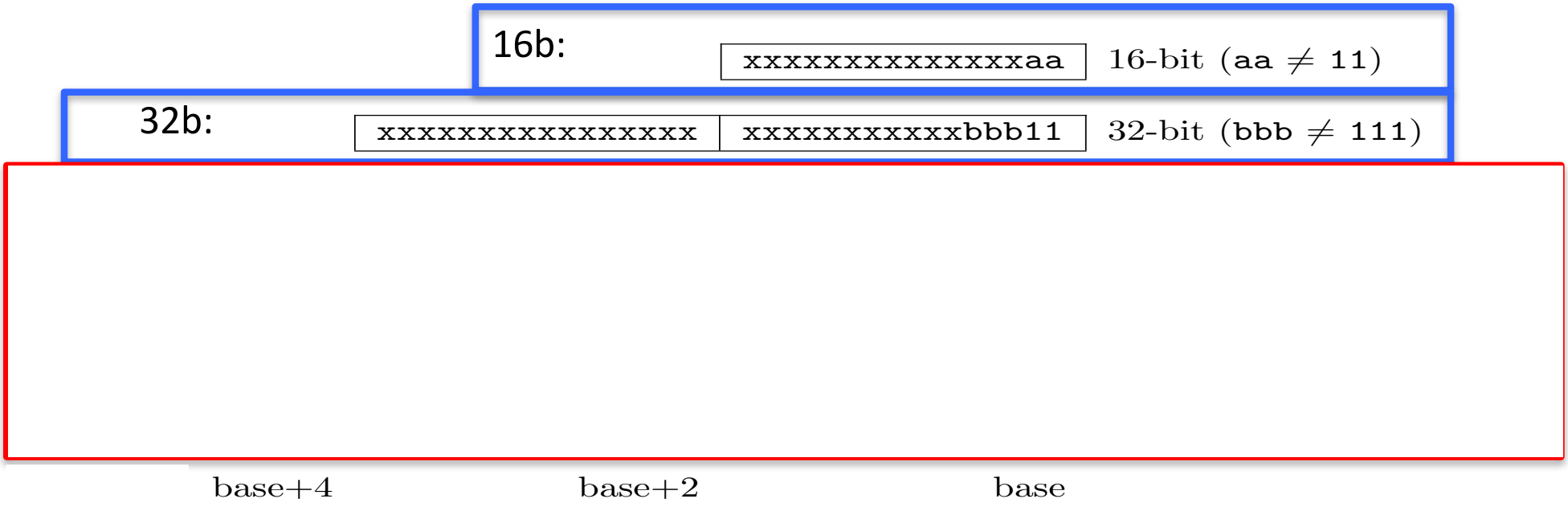
- Base < 50 instructions, 4 variants
  - 16 registers: RV32E
  - 32 registers: RV32I, RV64I, RV128I
- Standard ISA extensions
  - M: Integer multiply/divide
  - A: Atomic memory operations (AMOs + LR/SC)
  - F: Single-precision floating-point
  - D: Double-precision floating-point
  - Q: Quad-precision floating-point
- Fairly standard RISC encoding 32-bit instruction format (eg, MIPS)

# Base ISA Encoding: Always 32 Bits



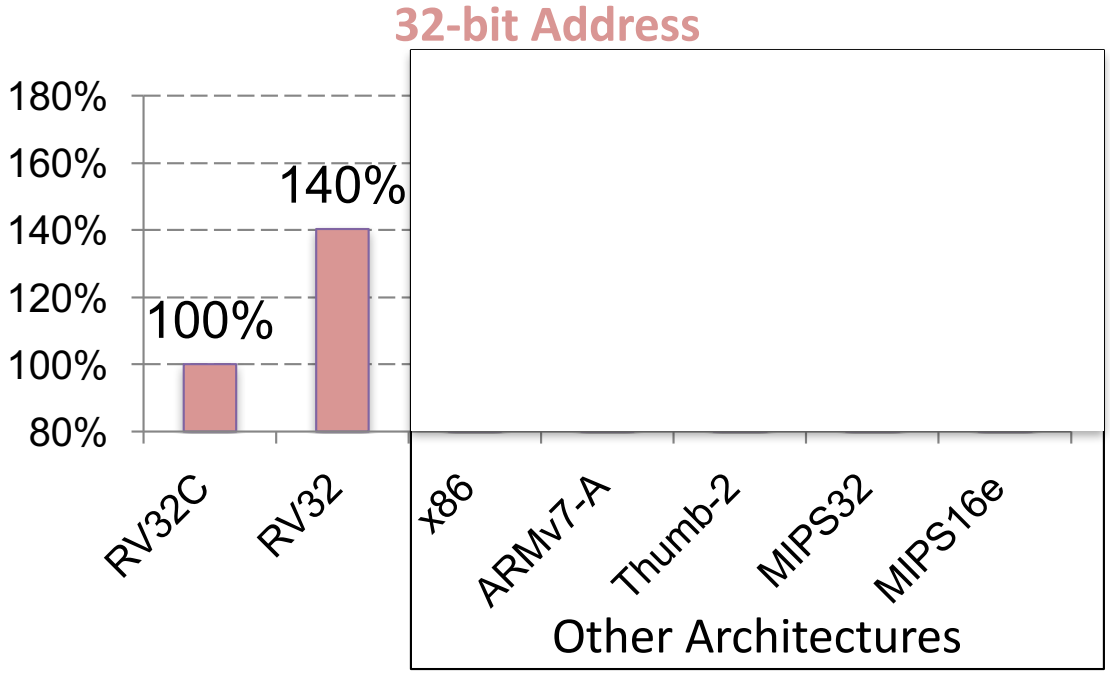

- 32b x 32 registers (32b x 16 in “embedded”)
  - 64b, 128b variants
- rd/rs1/rs2 in fixed location, no implicit registers
- Immediate field (instr[31]) always sign-extended

# Rich Instruction Encoding Space



# Compressed Instructions

- 16b encoding
  - Expands into one 32b instr.
  - 2-address forms (32 reg.)
  - 3-address forms (8 reg.)
- Implementation
  - Decoder ~700 gates
  - 16-bit instruction alignment
  - Compiler-oblivious

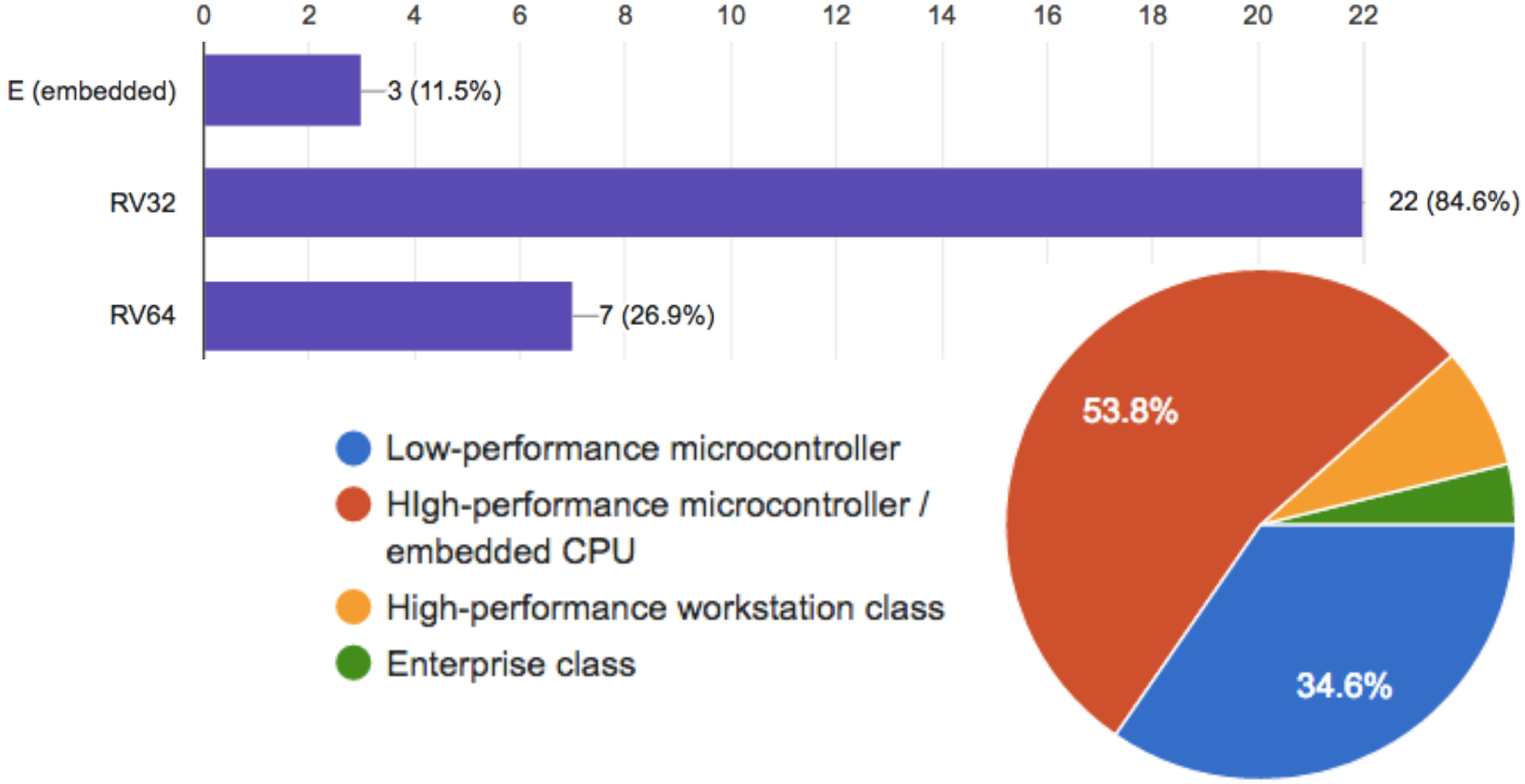



RISC-V smallest on SPECint2006

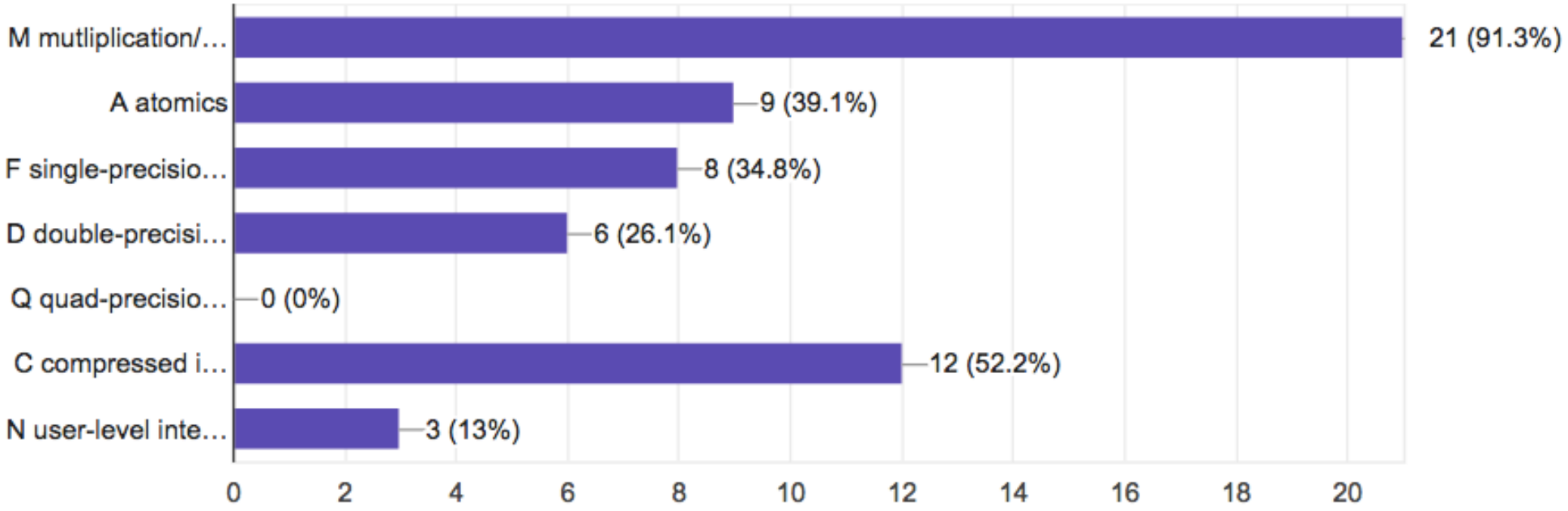
# RISC-V Growth

- Rapid uptake in industry + academia
  - Google, Micron, Samsung, Qualcomm, NVIDIA, NXP, WD, …
- Growing open software ecosystem
- Variety of proprietary + open-source cores
  - Seeded by Rocket SoC Generator (open source)

# Survey: 26 RISC- V CPU Designs

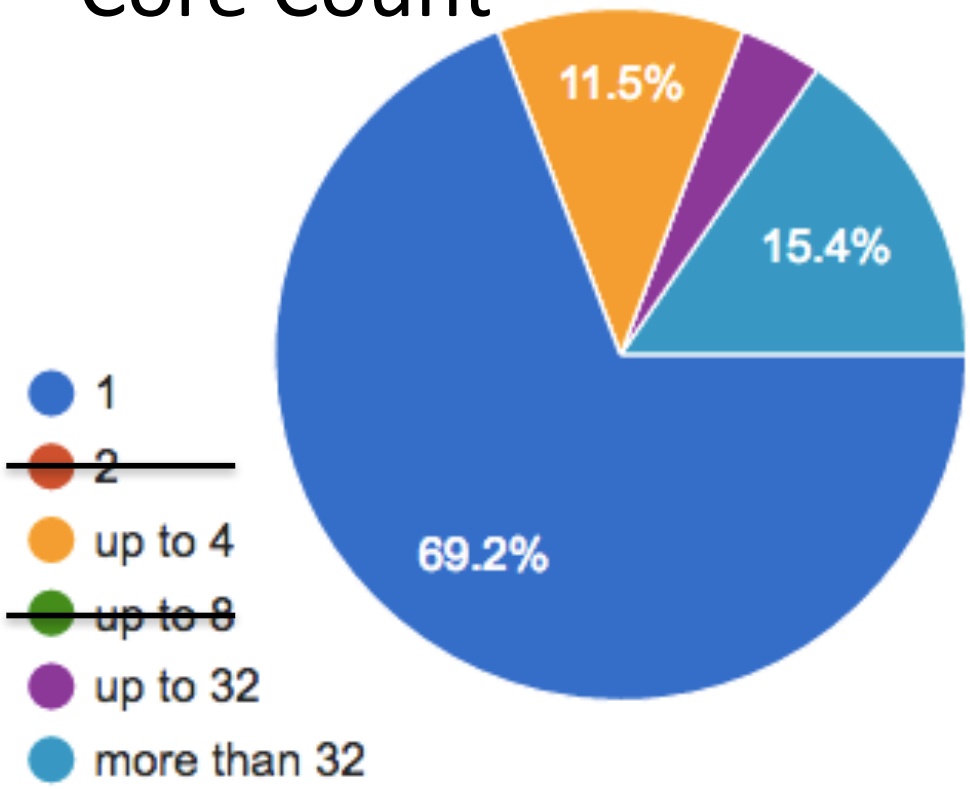


# Survey: Standard Extensions

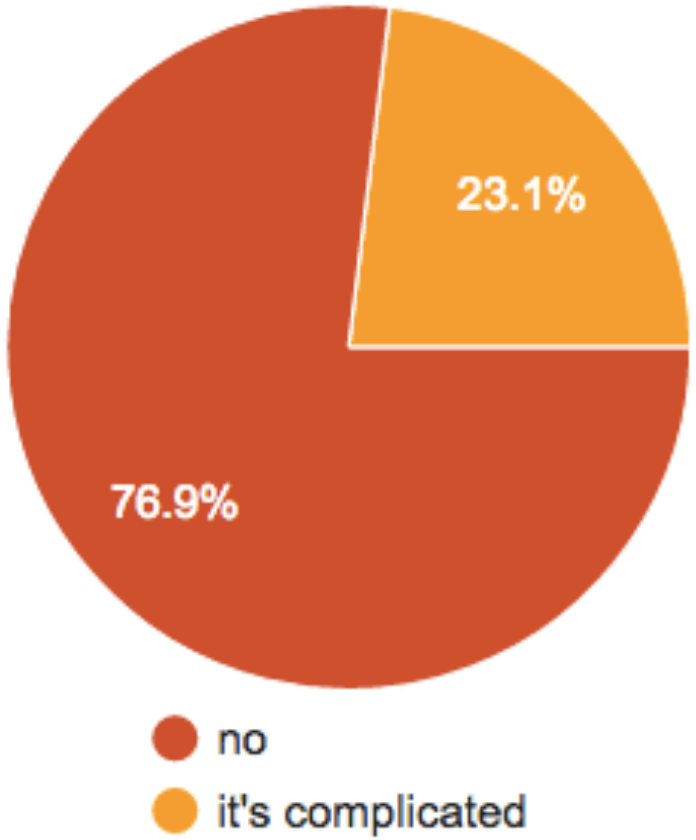


# Survey: Multicore

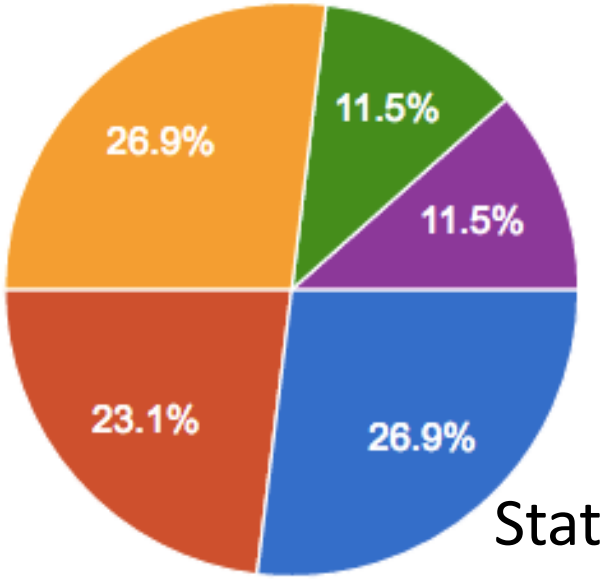
## Core Count




## big.LITTLE

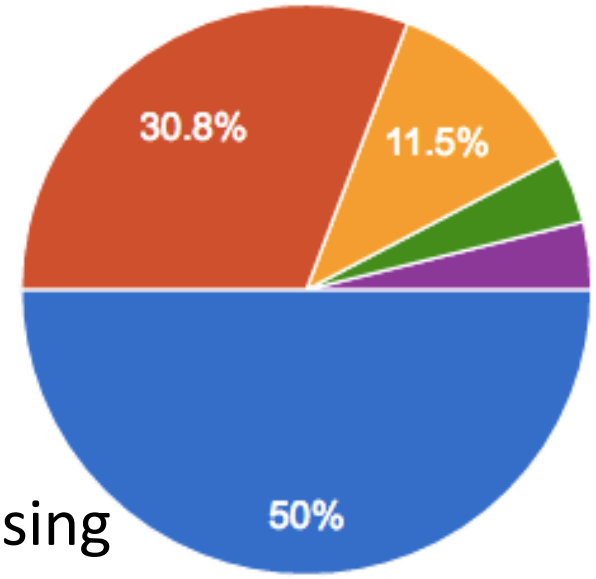


# Survey: Availability




Status

- In development
- Works in research lab
- In a select few customer labs
- In many customer labs
- Fully shipping




Licensing

- open source
- for fee under license
- privately held / may be available at some point
- privately held / unavailable / please don't ask us
- it's complicated

# Core and Accelerator Choices

RISC-V enables two kinds of choices…

1. Choices as a user
   - Variety of RISC-V providers/products
   - Best selection / performance / availability / competitive prices

2. Choices a provider (SoC designer, cpu architect, …)
   - Variety of design/implementation options
   - More value to users

# FPGA Soft Processor Choices

| | ISA | Interconnect | Tool |
|---|---|---|---|
| **Intel/Altera** | Nios II | Avalon | Qsys |
| **Xilinx** | MicroBlaze | AXI | IPI |
| **Lattice** | Mico32 | Wishbone3 | MicoSystemBuilder |
| **Microsemi** | ARM M1 | APB/AHP/AXI | SystemBuilder |

- Highly fractured
  - No common ISA, closed-source CPUs
  - No common interconnect
  - No common system build tools
  - No common IP or software

How is any **IP Ecosystem** going to thrive?

# FPGAs + RISC-V

- **Healthy shared ecosystem, cross-platform potential (all vendors)**

- Members of RISC-V Foundation
  - Microsemi  Rocket chip + SoftConsole IDE
  - Lattice

- FPGA-optimized soft cores
  - ORCA  VectorBlox  200 MHz pipelined (all vendors)
  - PicoRV32  Clifford Wolf  300+ MHz multicycle (Xilinx)
  - GRVI  Jan Gray  1,000+ pipelined cores (Xilinx)

# Case Study: VectorBlox

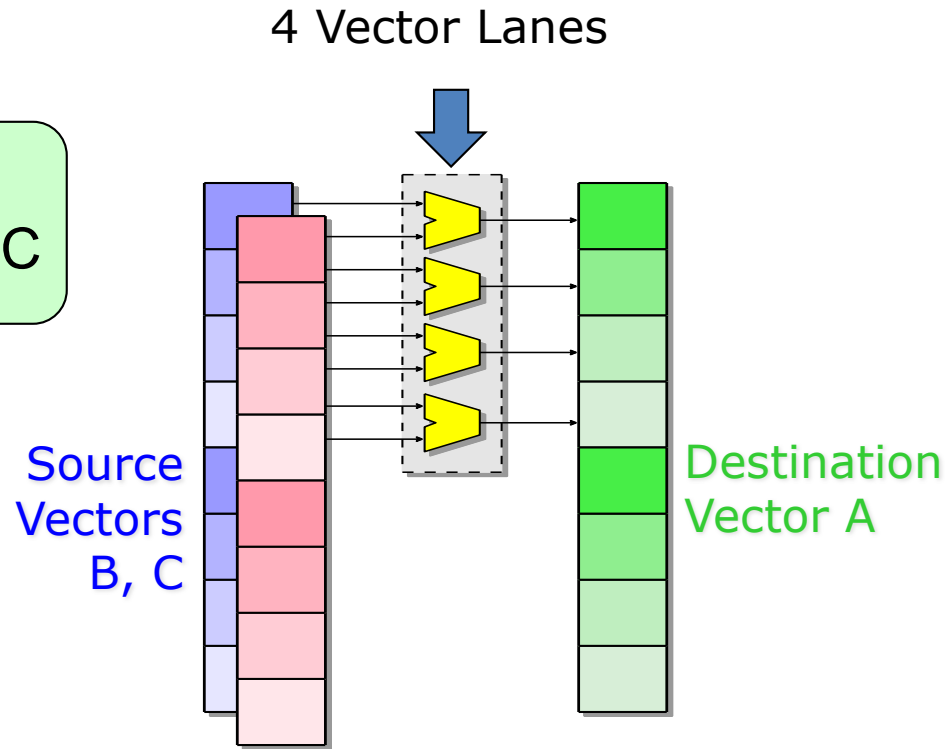
- VectorBlox mission
  - Design custom vector accelerators
- But…
  - No access to soft processor source code
  - Hard ARM cores are inflexible
  - Fragmented ecosystems – very costly/risky for small IP Vendors

- VectorBlox ORCA http://www.github.com/vectorblox/orca
  - Open source RISC-V
  - Optional proprietary extensions
    - Lightweight vector: share the RISC-V ALU
    - Full vector: up to 256 parallel ALUs

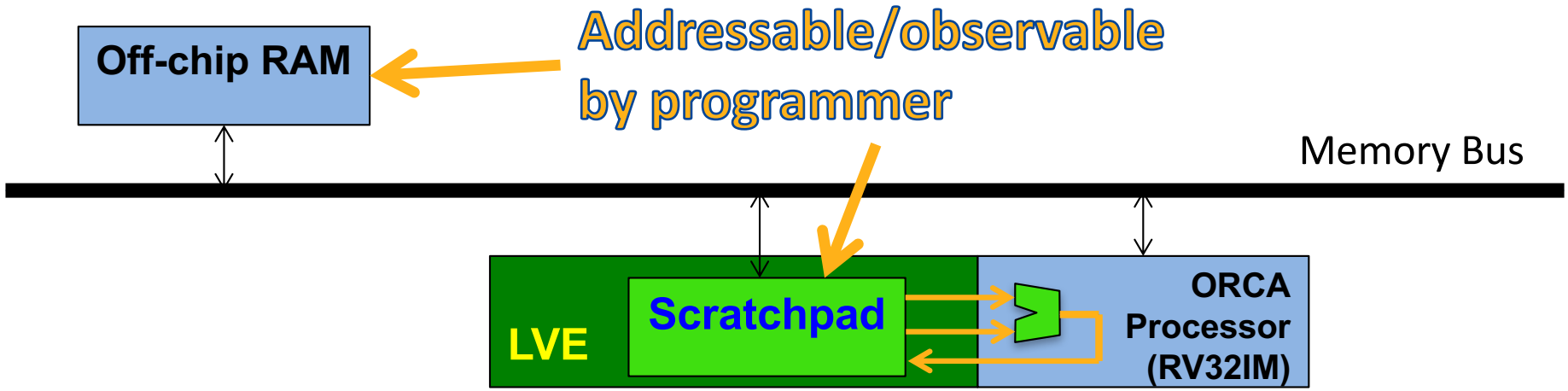
# Vector Programming

**Data-level parallelism**

```
for ( i=0; i<8; i++ )        →   set VL, 8
  A[i] = B[i] * C[i];            vmult  A, B, C
```



# System Model




- Vector instructions operate only on scratchpad
  - Stream data through RISC V ALU
  - Address generators in hardware

# Lightweight Vector Extension (LVE)

```
Vadd    Vsub            // RV32I base (vectorized)
Vsll    Vsrl    Vsra
Vxor    Vor     Vand
Vslt    Vsltu

Vmul    Vmulh           // RV32M multiplication (vectorized)
Vdiv    Vrem

Vcmv_z  Vcmv_nz         // VectorBlox: conditional move
Vtype                   // VectorBlox: data type, vector length
```

# FIR filter (12x speedup)

- RV32IM

```
00000030 <scalar_fir(long*, long*, long*, int, int)>:
  30:   40e686b3            sub     a3,a3,a4
  34:   06d05263            blez    a3,98 <.L6>
  38:   00269693            slli    a3,a3,0x2
  3c:   00271e13            slli    t3,a4,0x2
  40:   00d50eb3            add     t4,a0,a3
  44:   01c60e33            add     t3,a2,t3
  48:   00100f13            li      t5,1

0000004c <.L10>:
  4c:   0005a683            lw      a3,0(a1)
  50:   00062803            lw      a6,0(a2)
  54:   00460793            addi    a5,a2,4
  58:   00058893            mv      a7,a1
  5c:   03068833            mul     a6,a3,a6
  60:   01052023            sw      a6,0(a0)
  64:   02ef5263            ble     a4,t5,88 <.L11>

00000068 <.L13>:
  68:   0048a683            lw      a3,4(a7)
  6c:   0007a303            lw      t1,0(a5)
  70:   00478793            addi    a5,a5,4
  74:   00488893            addi    a7,a7,4
  78:   026686b3            mul     a3,a3,t1
  7c:   00d80833            add     a6,a6,a3
  80:   01052023            sw      a6,0(a0)
  84:   fefe12e3            bne     t3,a5,68 <.L13>

00000088 <.L11>:
  88:   00450513            addi    a0,a0,4
  8c:   00458593            addi    a1,a1,4
  90:   faae9ee3            bne     t4,a0,4c <.L10>
  94:   00008067            ret
```

No vectors
8 instructions, N stalls
72 bytes code for 2 loops

- RV32IM + LVE

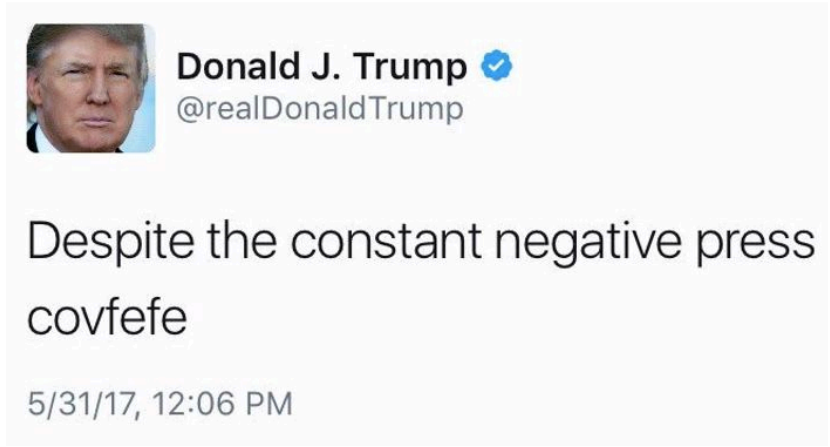
```
00000000 <vector_fir(long*, long*, long*, int, int)>:
   0:   000007b7            lui     a5,0x0
   4:   00e7a023            sw      a4,0(a5)
   8:   40e686b3            sub     a3,a3,a4
   c:   02d05063            blez    a3,2c <.L1>
  10:   00269693            slli    a3,a3,0x2
  14:   00d586b3            add     a3,a1,a3

00000018 <.L3>:
  18:   08c5fe2b            vtype.www   a1,a2
  1c:   a6e50cab            vmul.vv.1d.sss.acc  a0,a4
  20:   00458593            addi    a1,a1,4
  24:   00450513            addi    a0,a0,4
  28:   fed598e3            bne     a1,a3,18 <.L3>

0000002c <.L1>:
  2c:   00008067            ret
```

Vectors
1 instruction, 0 stalls
20 bytes code for 2 loops

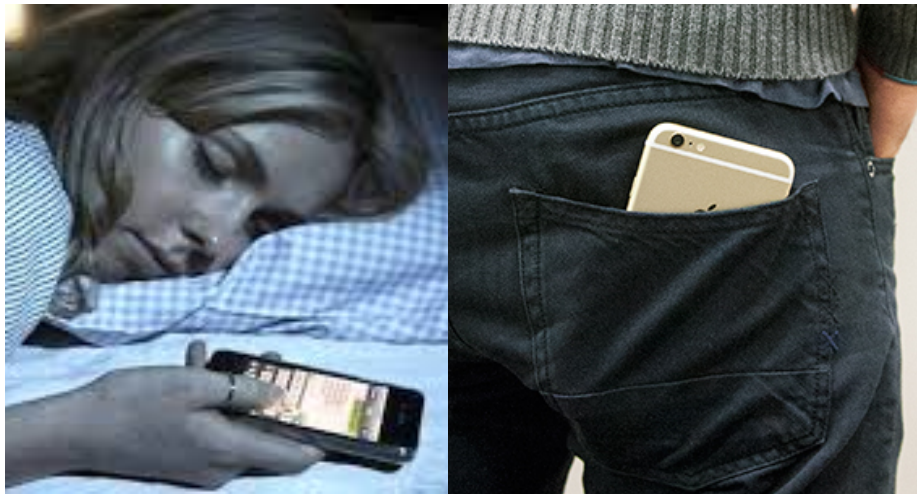
Real application:

“butt-posting”

“butt-tweeting”




cause?



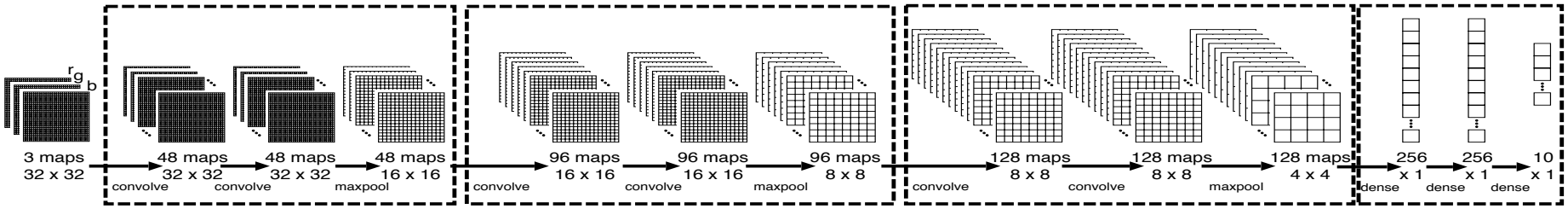
solution!

# Deep Learning: Face Detector

Built using
ORCA RISC-V
+
Lightweight
Vectors
+
BNN Accel.

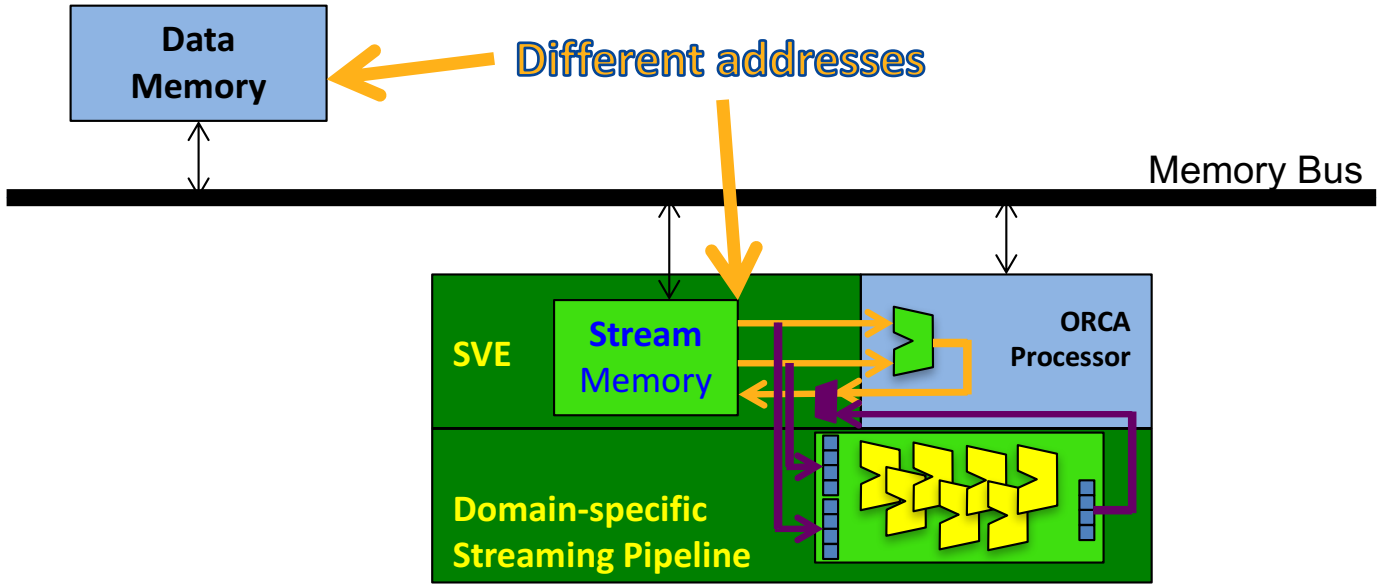
https://www.youtube.com/watch?v=_0rWDrOGqrk

# Deep Learning w/ Binary Weights

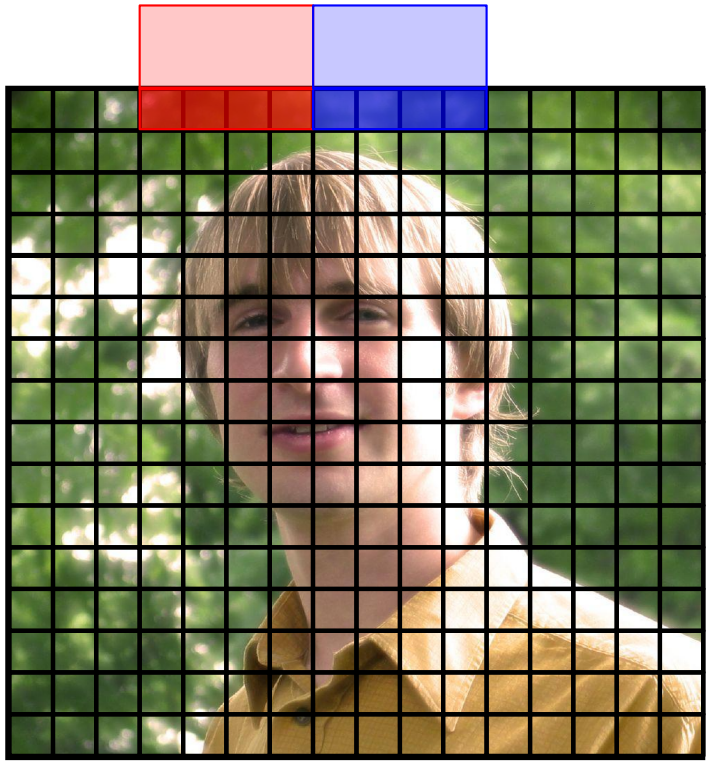


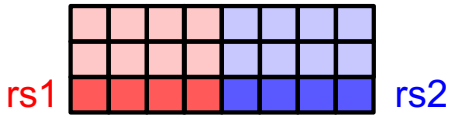

- Binary weights:  only +1/-1,  no *
- Database > 75,000 face images
- Trained 99% accurate

# Custom Streaming Accelerators

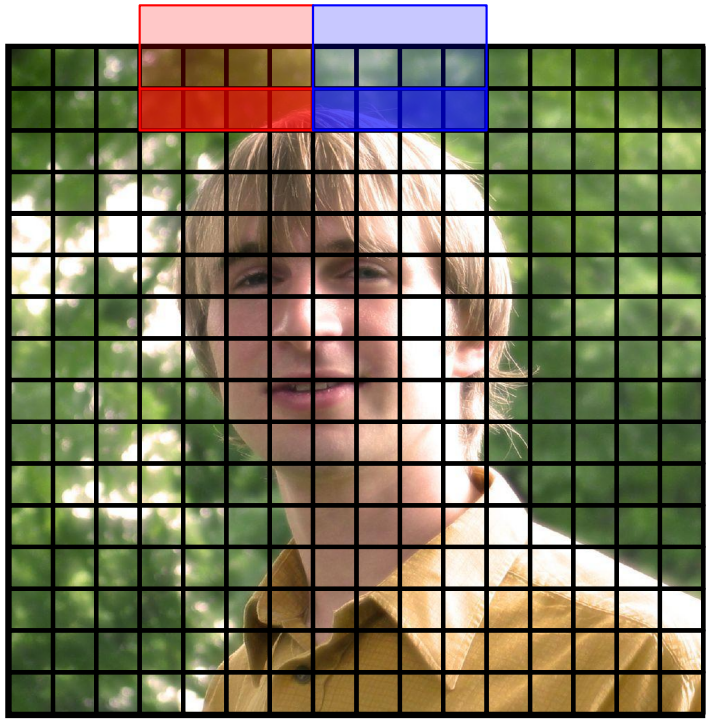


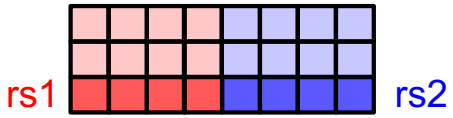
# Streaming Convolution Accelerator



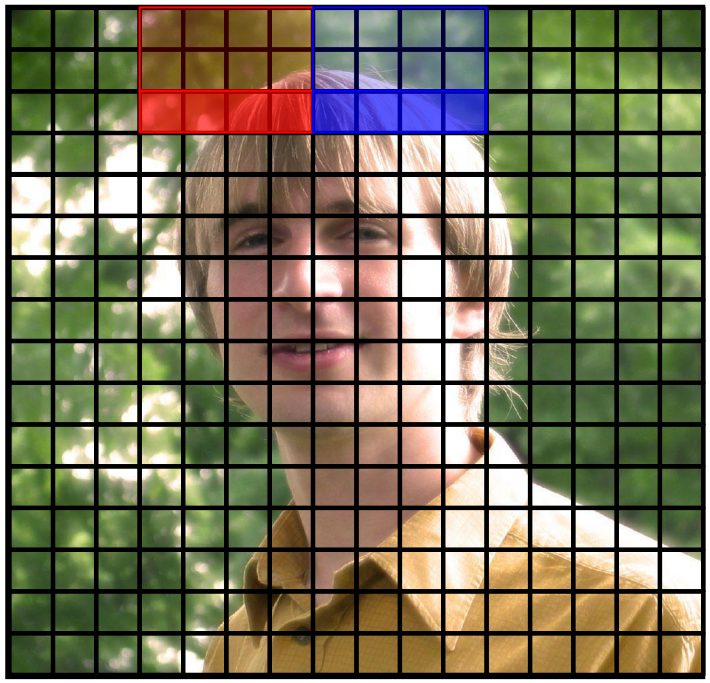


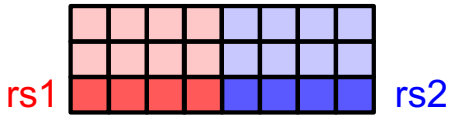
# Streaming Convolution Accelerator



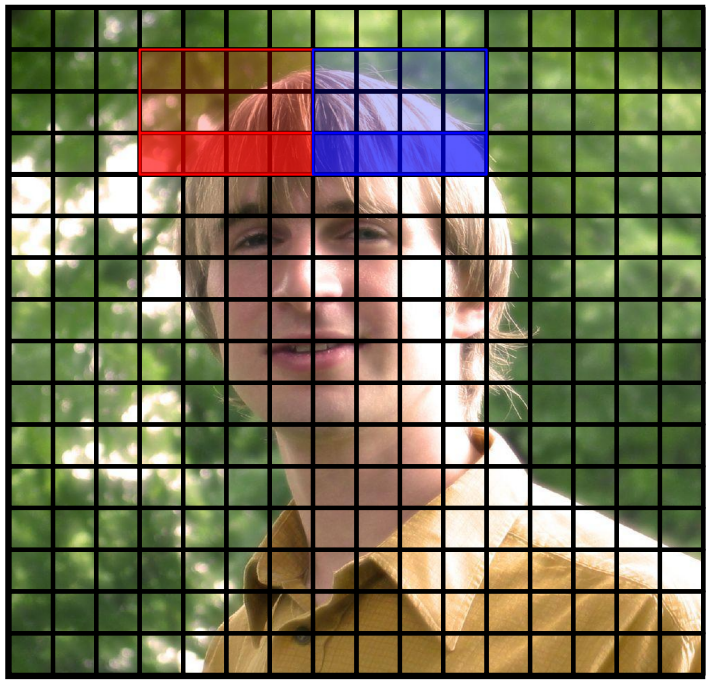


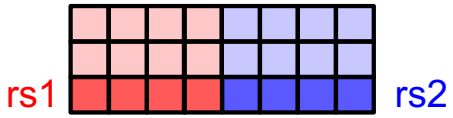
# Streaming Convolution Accelerator



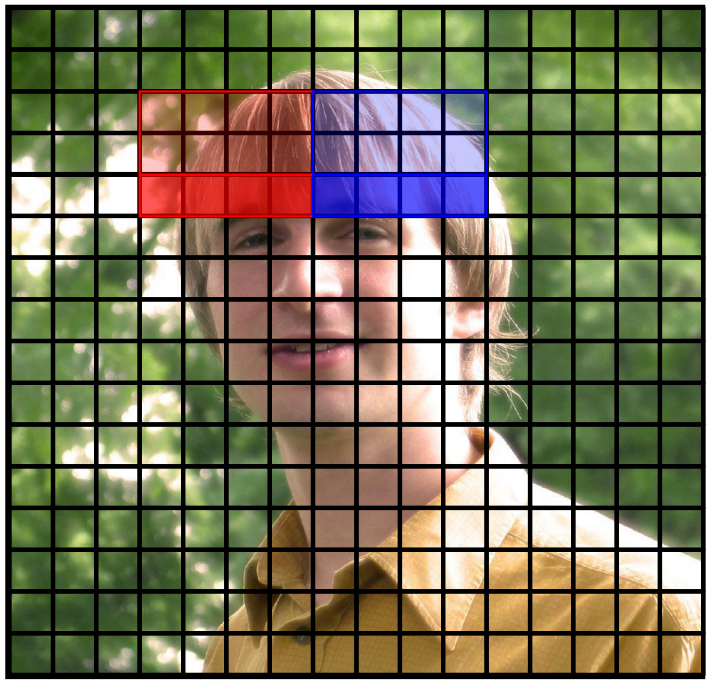


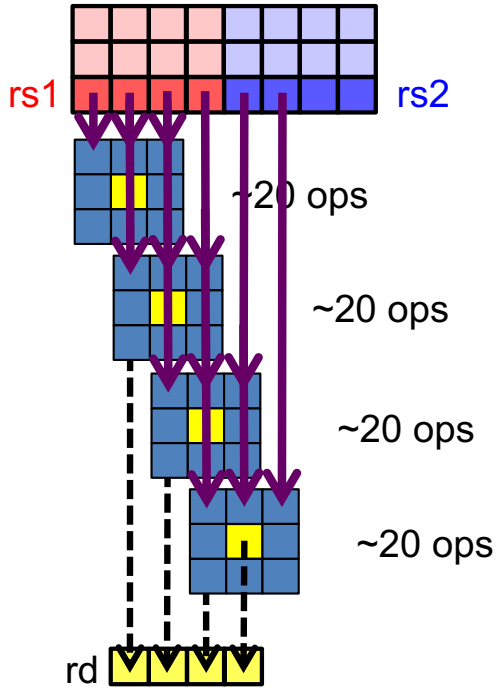
# Streaming Convolution Accelerator



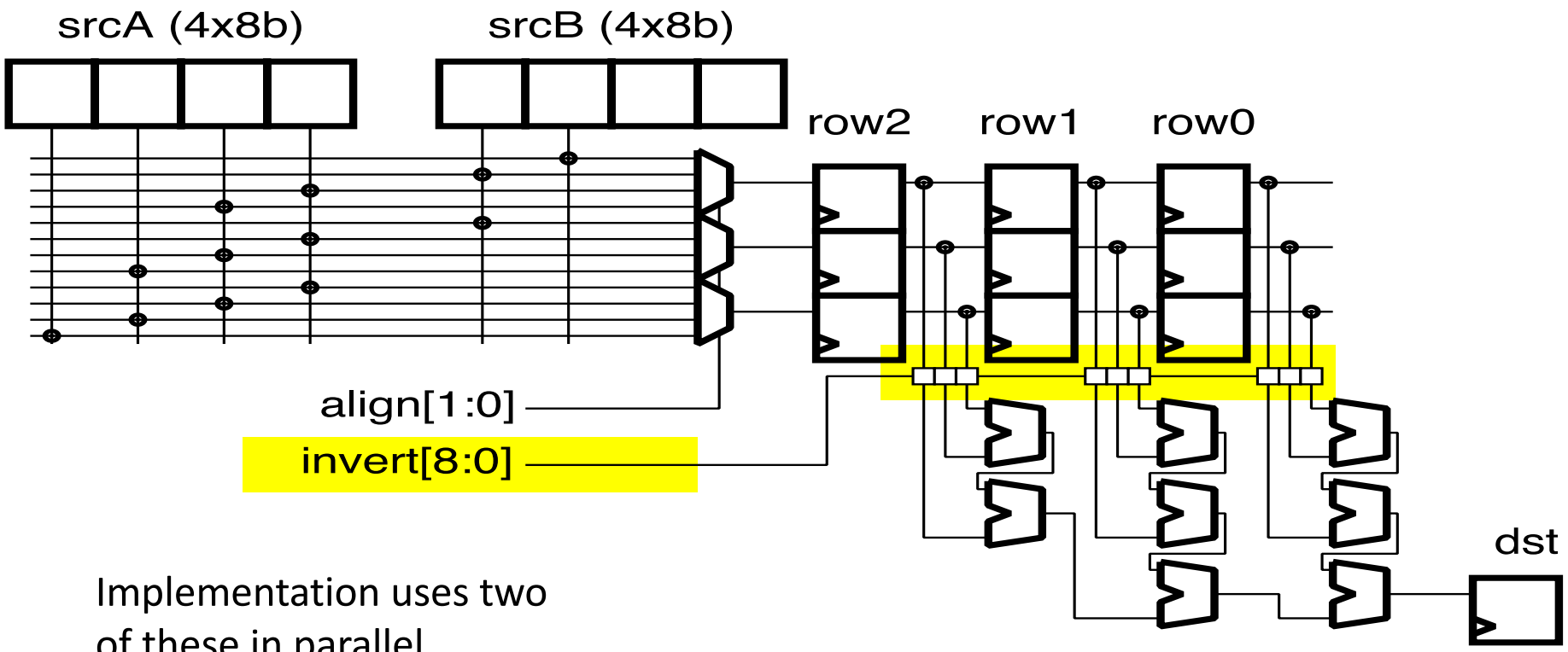


# Streaming Convolution Accelerator





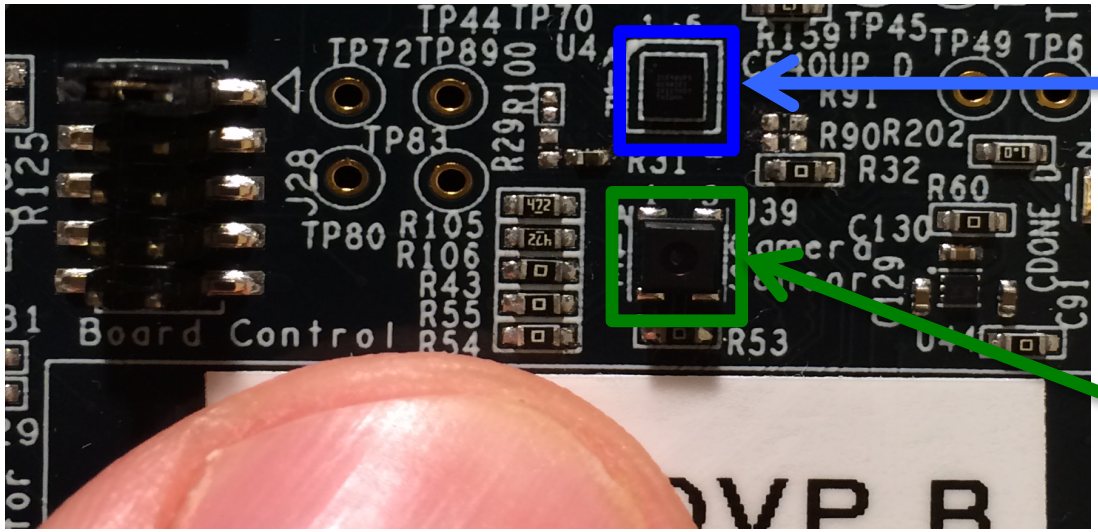
# Binary-weight NN Accelerator




Implementation uses two of these in parallel.

# Face Detector Performance

- 24 MHz, 1 cycle per instruction (4 stages)
- Effectively a 1.7 GHz RISC-V processor
  - Vectors 8x speedup. \
  - Binary CNN 73x speedup. / Overall 71x speedup
- Power ~15mW

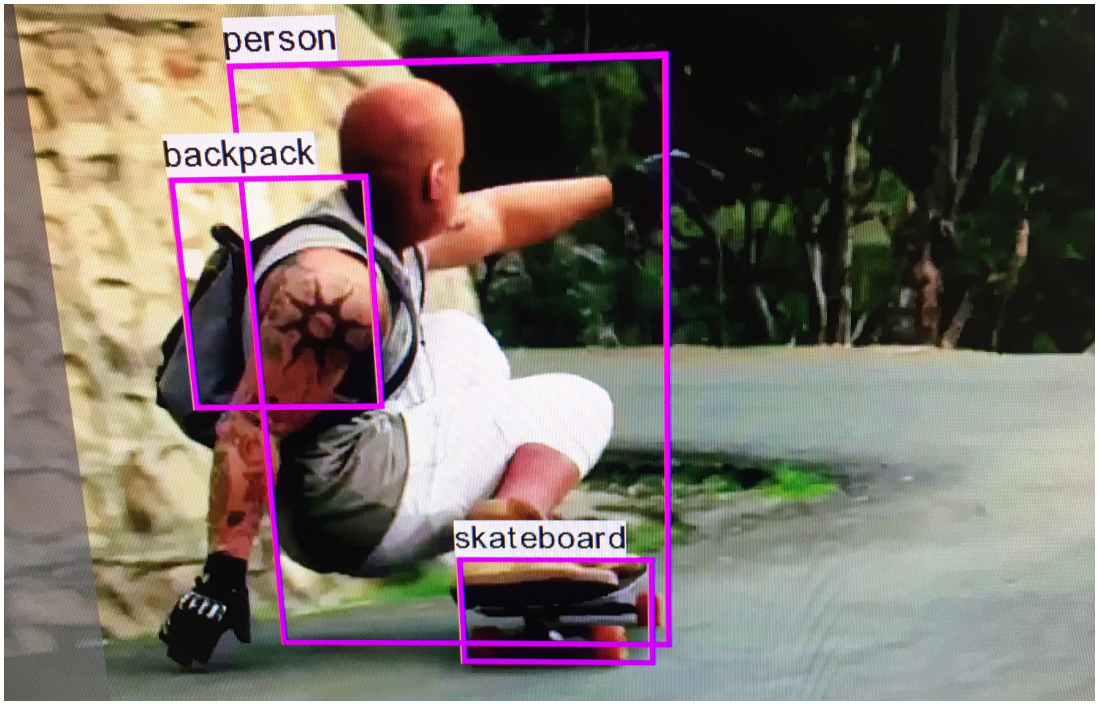


# Advantage: RISC-V

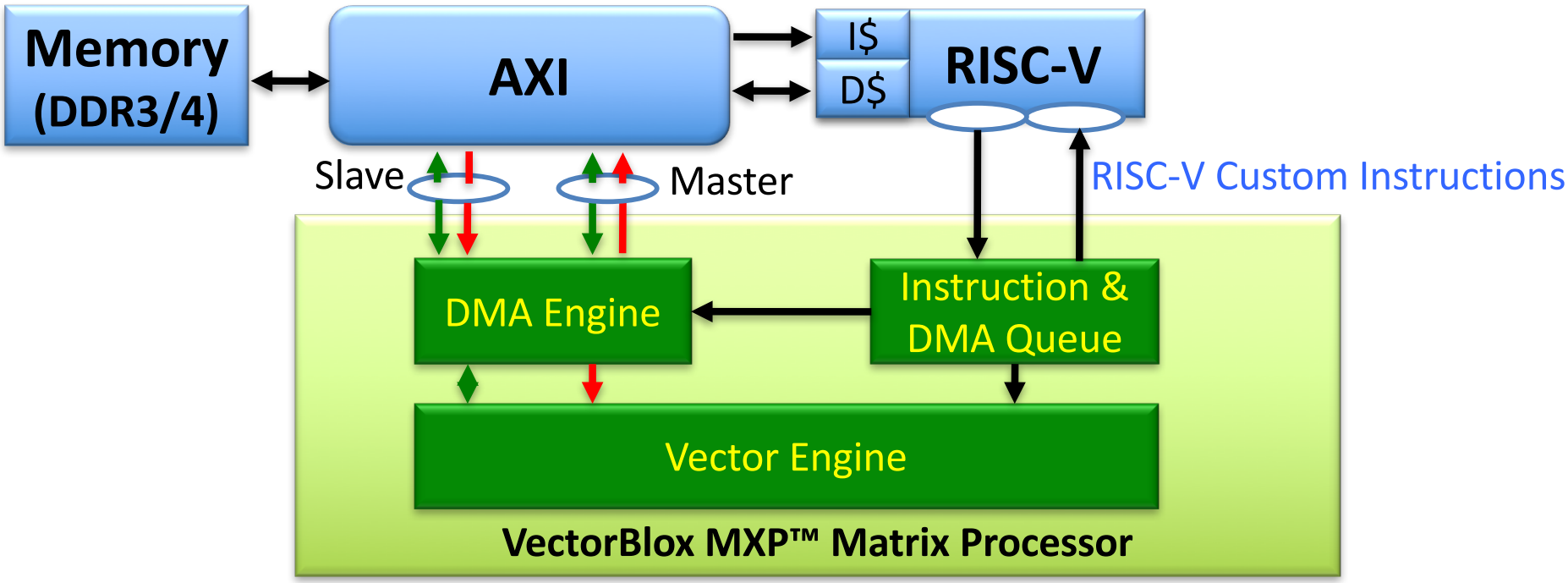
- RISC-V made this possible!
  - Open ISA → FPGA-based CPU + gcc
  - Lightweight vectors → 8x performance \
  - BNN accelerator → 73x performance / combined 71x
  - Power optimizations → about 15x lower power

- Not possible with closed-source CPUs
  - Could not add lightweight vectors
  - Could not add BNN accelerator
  - Could not make power optimizations

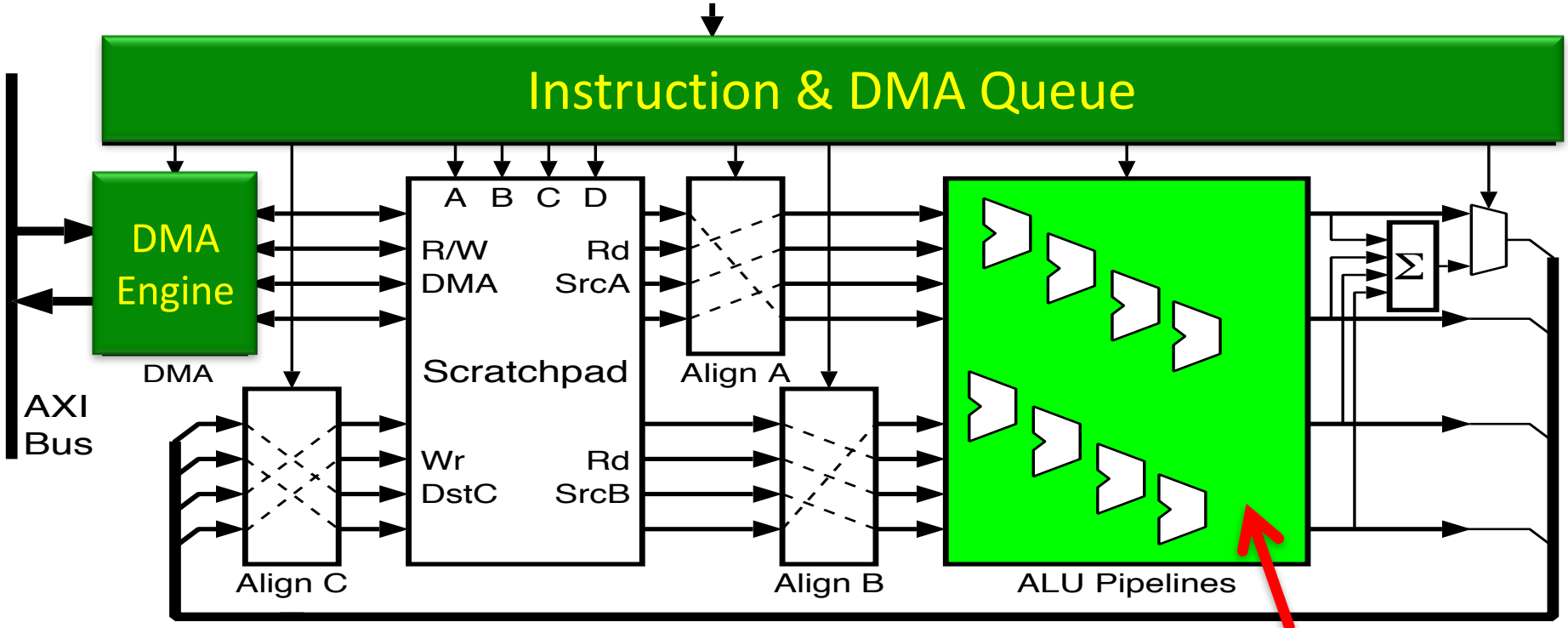
# Deep Learning Application: YOLO




# Full Vector Extension (MXP)



# Inside the VectorBlox MXP (2)




**+ CNN Accelerator**

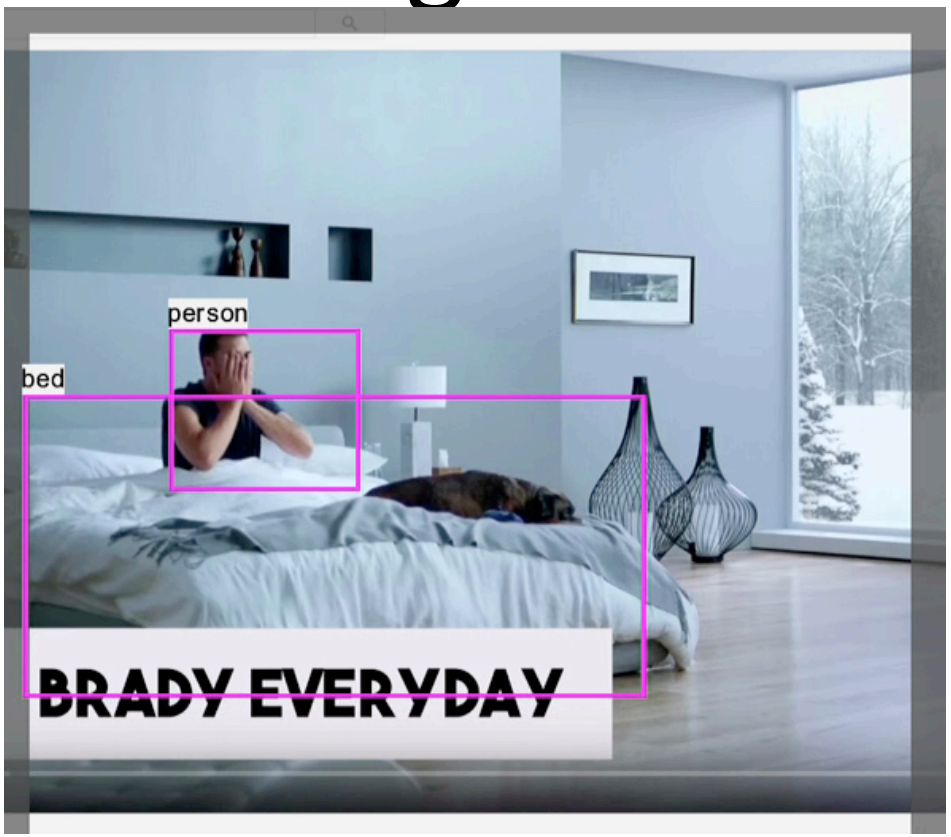
**2000 ops/cycle**

# Deep Learning Performance

- Same 28nm FPGA
  - ARM Cortex A9 667 MHz
  - VectorBlox MXP 160 MHz (1/4 of Cortex A9)

- ~300-500x speedup over ARM A9
  - RISC-V Custom Instructions: Vectors + CNN Accel.
  - Parallelism: 2,000 operations / clock cycle

# Deep Learning on VectorBlox




https://www.youtube.com/watch?v=SS2Rc5zpwxs

# Summary

- RISC-V is a free and open ISA
  - Enables healthy software (and hardware) ecosystems
  - Variety of cores available / in development

- Impact on VectorBlox
  - Single ecosystem for all FPGA Vendors
    - Shared SW + HW costs, leverage open source contributions
  - Access to CPU internals → performance + power
  - Full vectors (MXP) →10x to 10,000x performance (vs. soft core)
    - CNN application → about 500x performance (vs. hard ARM)
  - Lightweight vectors (LVE) → 8x performance
    - BNN application → 71x performance
    - Power optimizations → about 15x lower power

# RISC-V CPU Survey Respondents

| | |
|---|---|
| Rocket | github.com/freechipsproject/ |
| Freedom Everywhere | SiFive.com |
| Mythic IPU | mythic-ai.com |
| riscV | n/a |
| PulpTR | ankasys.com |
| proc_rv32ec_p2 | astc-design.com |
| proc_rv32ic_p5 | astc-design.com |
| KCP53000 | kestrelcomputer.github.io/kestrel |
| RV12 | roalogic.com |
| PicoRV32 | github.com/cliffordwolf/picorv32 |
| Klessydra processing core | en.uniroma1.it |
| BOOM | ucb-bar.github.io/riscv-boom |
| PULP | github.com/pulp-platform |

| | |
|---|---|
| RV01 | n/a |
| IntenCore | intensivate.com |
| YARVI | github.com/tommythorn/yarvi |
| RISCVBusiness | purduesoceet.github.io |
| GRVI Phalanx | fpga.org/gray-research-llc |
| SCR1 | syntacore.com |
| SCR2 | syntacore.com |
| SCR3 | syntacore.com |
| SCR4 | syntacore.com |
| SCR5 | syntacore.com |
| Celerity | n/a |
| riscv-lanzones | github.com/e19293001/riscv-lanzones |
| ORCA | github.com/vectorblox/orca |