The RISC Journey from One to a Million Processors

Dave Ditzel

Founder and CTO

Esperanto Technologies, Inc.

dave@esperanto.ai

October 3, 2022

MICRO-55 Keynote

A Personal View of Computer History

Designing computers has been a lifelong passion

Fortunate to have been at the right place a number of times

- Iowa State University: SYMBOL High Level Language Computer & first 8-bit uP chips
- UC Berkeley: At transition from HLLCA to RISC, under prof. D. Patterson
- Bell Labs: At the creation of Unix, C, C++, early RISC processors (C-Machines)
- Sun Microsystems: At transition from M68K to RISC (SPARC)
- Transmeta: Binary translation onto VLIW-RISC
- Intel: Binary translation onto OOO-RISC
- Esperanto: Massively-parallel energy-efficient RISC-V processors

Got to see the evolution of early architectures from 8-bit to 64-bit superscalar monsters

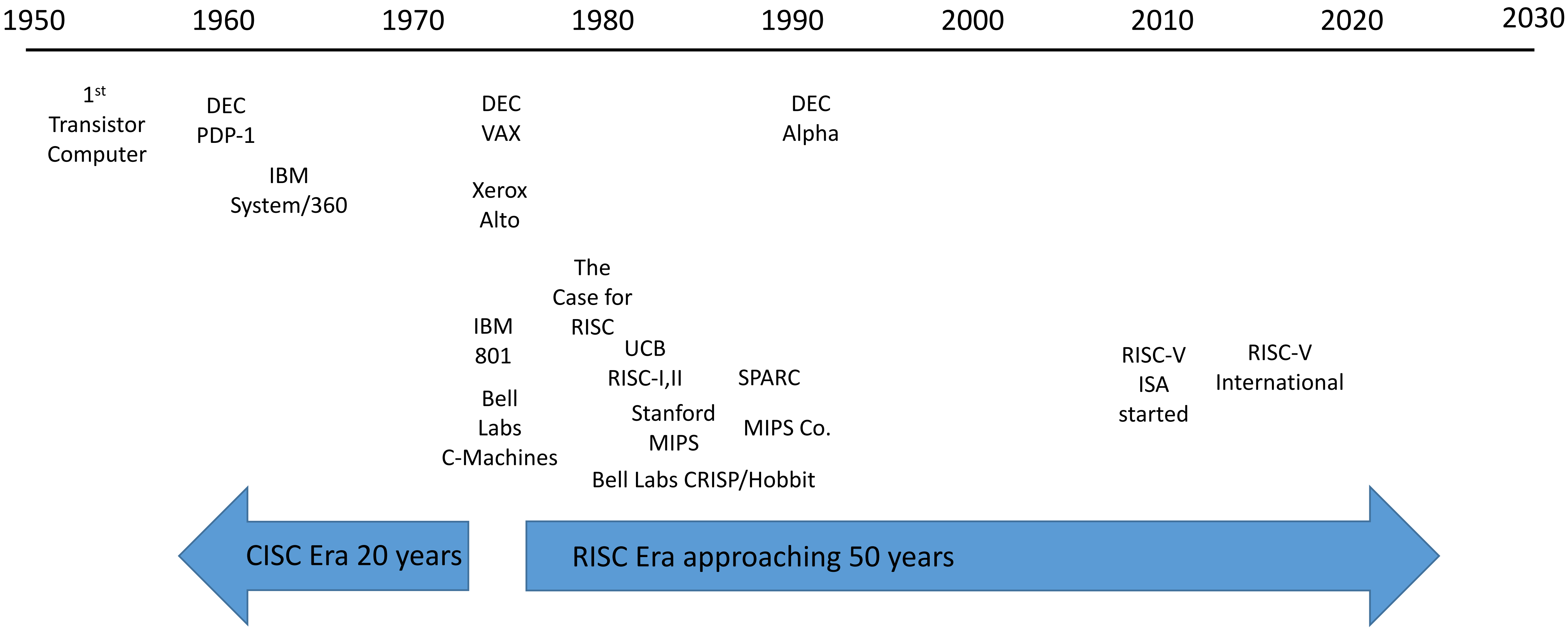
One trend in design seems to have withstood the test of time more than any other: RISC

Useful quote for computer architects

“Those who cannot remember the past are condemned to repeat it.”

George Santayana,
The Life of Reason,
1905.

Computer Development Timeline of Notable Designs



1950 1960 1970 1980 1990 2000 2010 2020 2030

1st Transistor Computer

DEC PDP-1

IBM System/360

DEC VAX

Xerox Alto

IBM 801

Bell Labs C-Machines

The Case for RISC

UCB RISC-I,II

Stanford MIPS

Bell Labs CRISP/Hobbit

SPARC

MIPS Co.

DEC Alpha

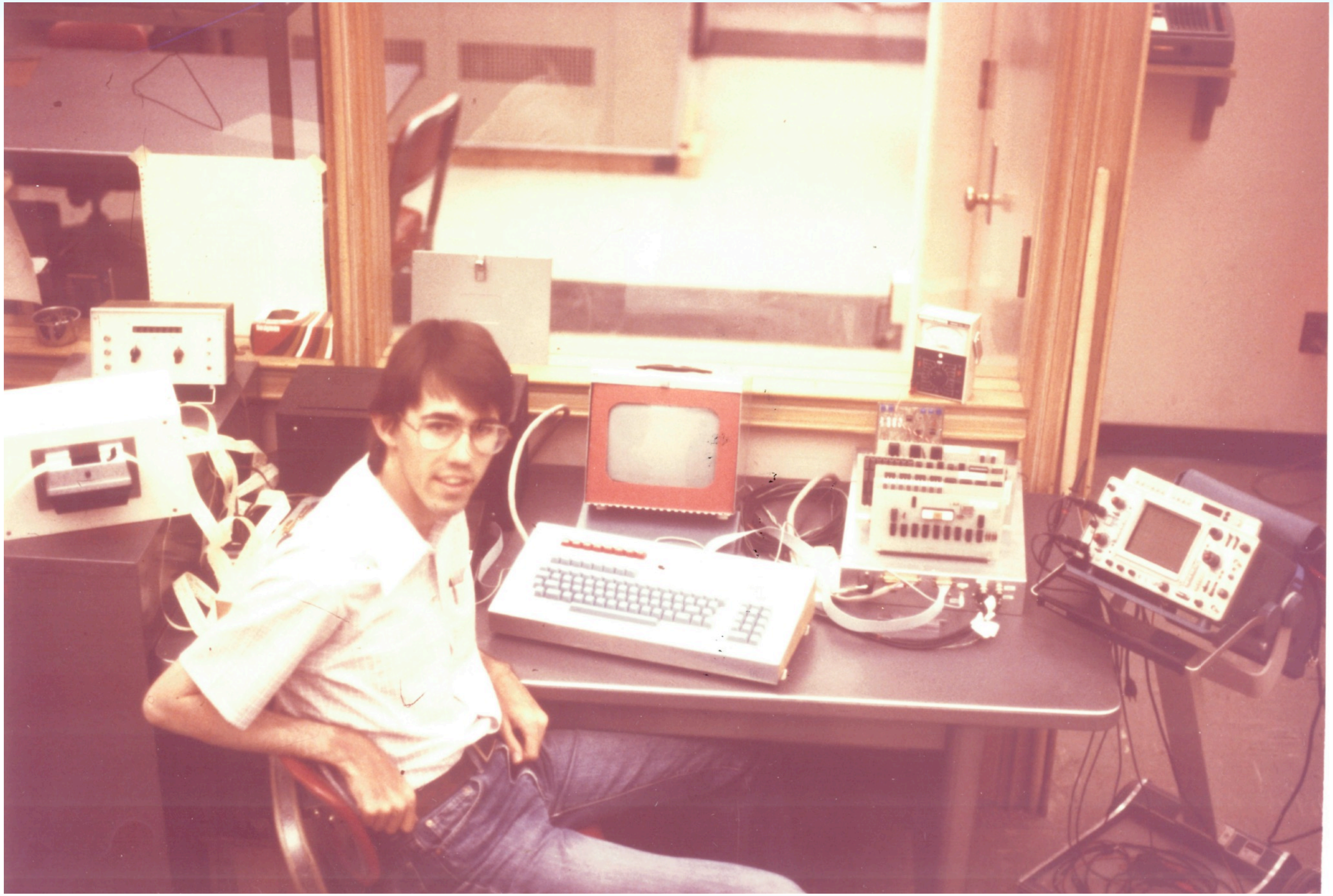
RISC-V ISA started

RISC-V International

CISC Era 20 years

RISC Era approaching 50 years

1978 Photo: My career started at the same time as the first microprocessors



My personal hobby computer

8-bit 6502 Microprocessor

4 MHz

4K Bytes of main memory

Hand “wire-wrapped”

Paper tape reader

Hand wound transformer on power supply.

Note – base of SYMBOL computer in background

Era of
High Level Language Computer Architecture

later called CISC (Complex Instruction-set Computers)

DEC VAX-11/780: a 32-bit CISC

A real 70’s architecture:

- VAX: Transitioned from 16-bit PDP-11 assembly to 32-bit addresses and compiled code
- ISA made it easy to take high level language statements and cast into VAX assembly code

Instruction set heavily influenced by availability of 8-bit wide integrated circuits

VAX ISA composed of variable length instructions

- Instruction opcode (1 or 2 bytes) followed by up to 6 operands
- First operand descriptor (1 byte)
- First operand data (0-4 bytess
- Second operation descriptor
- Second operand data
- Third operand descriptor
- Third operand data (0-4 bytes)
- …

This was great when decoding serially one byte at a time,

but a nightmare for pipelined or later superscalar implementations

Xerox Palo Alto Research Center: Alto

First Graphical User Interface with mouse

5.8 MHz microcoded CPU

User loadable instruction sets led to several

HLLCA type bytecoded instruction sets for languages like BCPL, Smalltalk and MESA

Enabled exploration of highly tailored instruction sets

But raised the question
“Why not compile to lowest microcode level?”

SYMBOL:
The Ultimate CISC

SYMBOL: The ultimate Complex Instruction Set Computer

SYMBOL was a High Level Language Computer announced in 1971 by Fairchild Semiconductor

Supported by Gordon Moore and Robert Noyce at Fairchild until they left to form Intel

Goal was to reduce cost of software by using hardware instead, literally “programmers cost too much”.

Implemented Compiler, Text Editor and Operating System entirely in logic gates

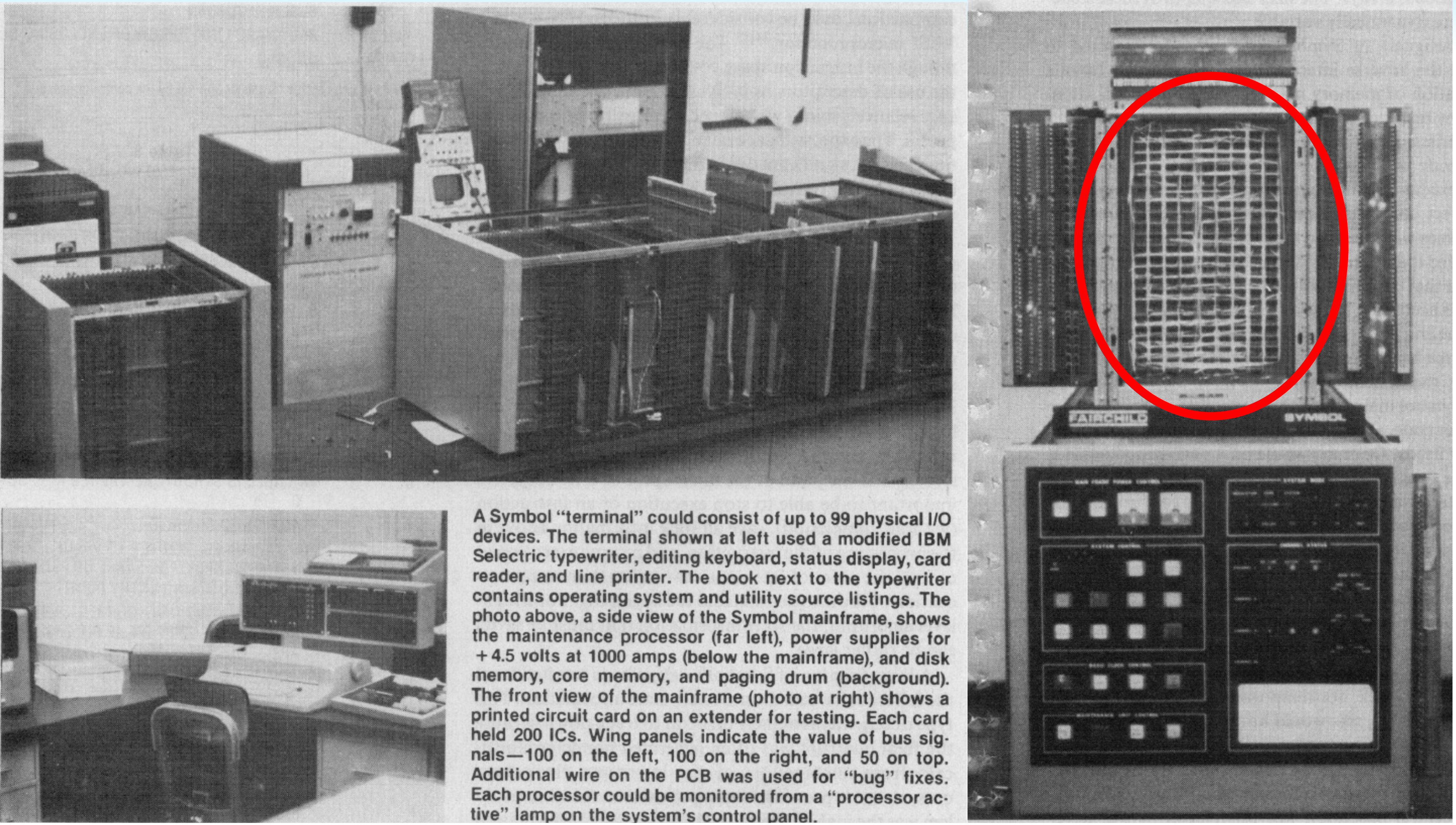
- 2 gates or 1 Flip-Flop per 14-pin DIP package, no ROM
- 20 thousand chips – took **several years** to debug at ISU
- Instruction set was bytecode mapped almost 1-1 from high level ALGOL/LISP like typeless language
- Tagged architecture with descriptors, and “logical memory” that was not linearly addressable.
- Five years of my life thinking about CISC vs RISC, i.e. better ways to use 20K chips, programming this machine.

I got to work on this computer while a student for 5 years, after it was donated to ISU

Lessons:

- Don’t build your compiler or OS in logic gates
- Use right combination of Hardware, Software or μCode

Debugging SYMBOL, the inspiration for RISC



A Symbol "terminal" could consist of up to 99 physical I/O devices. The terminal shown at left used a modified IBM Selectric typewriter, editing keyboard, status display, card reader, and line printer. The book next to the typewriter contains operating system and utility source listings. The photo above, a side view of the Symbol mainframe, shows the maintenance processor (far left), power supplies for +4.5 volts at 1000 amps (below the mainframe), and disk memory, core memory, and paging drum (background). The front view of the mainframe (photo at right) shows a printed circuit card on an extender for testing. Each card held 200 ICs. Wing panels indicate the value of bus signals—100 on the left, 100 on the right, and 50 on top. Additional wire on the PCB was used for "bug" fixes. Each processor could be monitored from a "processor active" lamp on the system's control panel.

Inside red circle are 220 chips.

In order to fix a bug, white wires added on top of blue pc board to make logic changes.

Many of the 100 boards were covered with white bug fixes.

Source: D. Ditzel, Reflections on the High Level Language SYMBOL Computer Systems, IEEE Computer, July 1981.

HLLCA Lessons

Putting functions in hardware does not necessarily make them higher performance

Putting functions in hardware always makes them harder to debug, modify and fix

High level opcodes often precluded the ability to make code generation optimizations

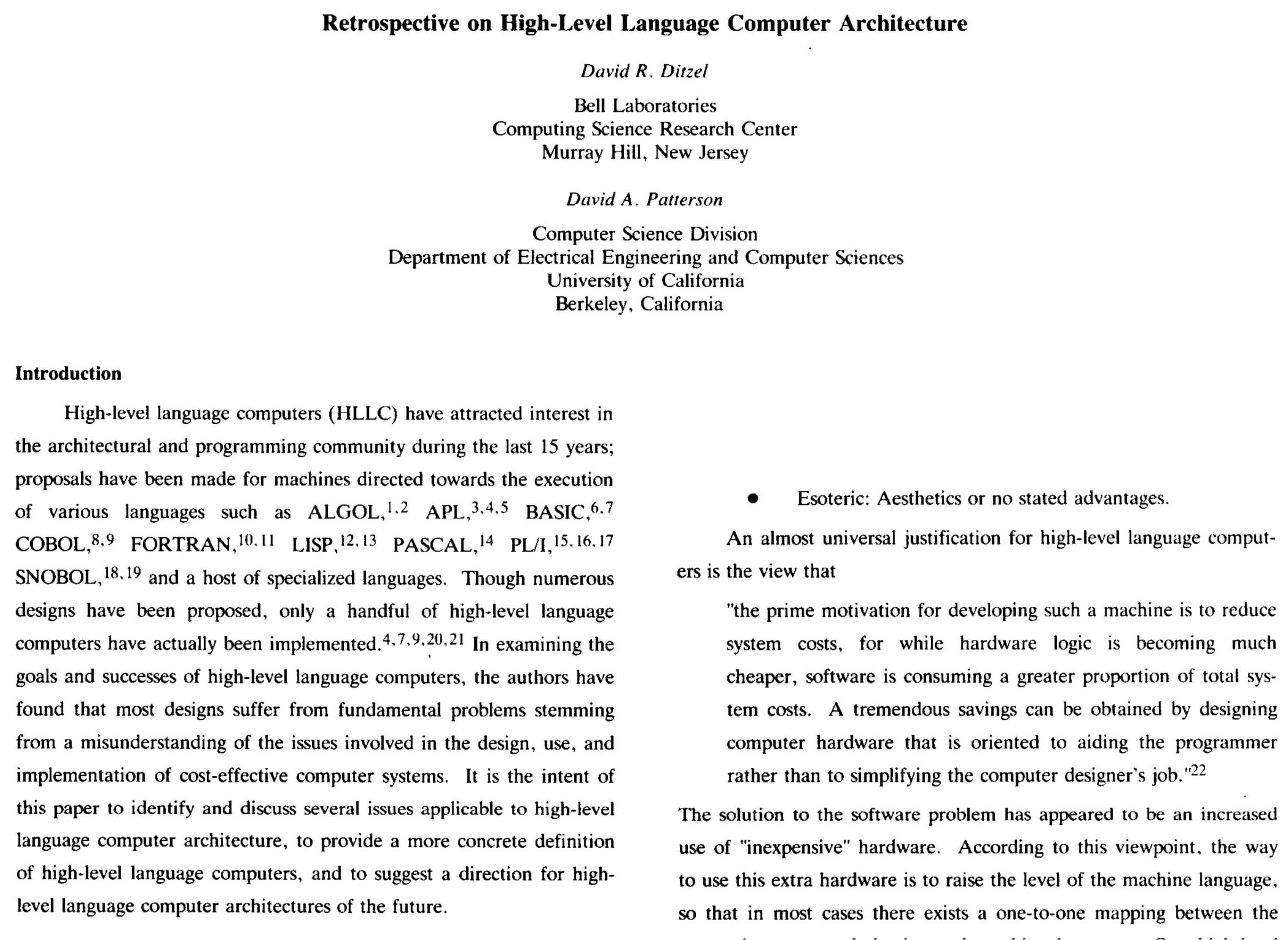
Example: Symbol precluded any use of pointer arithmetic, array accesses quite slow.

Better to provide a low level instruction set with HLL Compilers

Use logic gates to make that simple instruction set go very fast

Fascination studying SYMBOL (Ditzel/Patterson) eventually led both to religious conversion to RISC

Mood was changing against HLLCA by end of 70's



Retrospective on High-Level Language Computer Architecture

David R. Ditzel

Bell Laboratories
Computing Science Research Center
Murray Hill, New Jersey

David A. Patterson

Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California
Berkeley, California

Introduction

High-level language computers (HLLC) have attracted interest in the architectural and programming community during the last 15 years; proposals have been made for machines directed towards the execution of various languages such as ALGOL,[1,2] APL,[3,4,5] BASIC,[6,7] COBOL,[8,9] FORTRAN,[10,11] LISP,[12,13] PASCAL,[14] PL/I,[15,16,17] SNOBOL,[18,19] and a host of specialized languages. Though numerous designs have been proposed, only a handful of high-level language computers have actually been implemented.[4,7,9,20,21] In examining the goals and successes of high-level language computers, the authors have found that most designs suffer from fundamental problems stemming from a misunderstanding of the issues involved in the design, use, and implementation of cost-effective computer systems. It is the intent of this paper to identify and discuss several issues applicable to high-level language computer architecture, to provide a more concrete definition of high-level language computers, and to suggest a direction for high-level language computer architectures of the future.

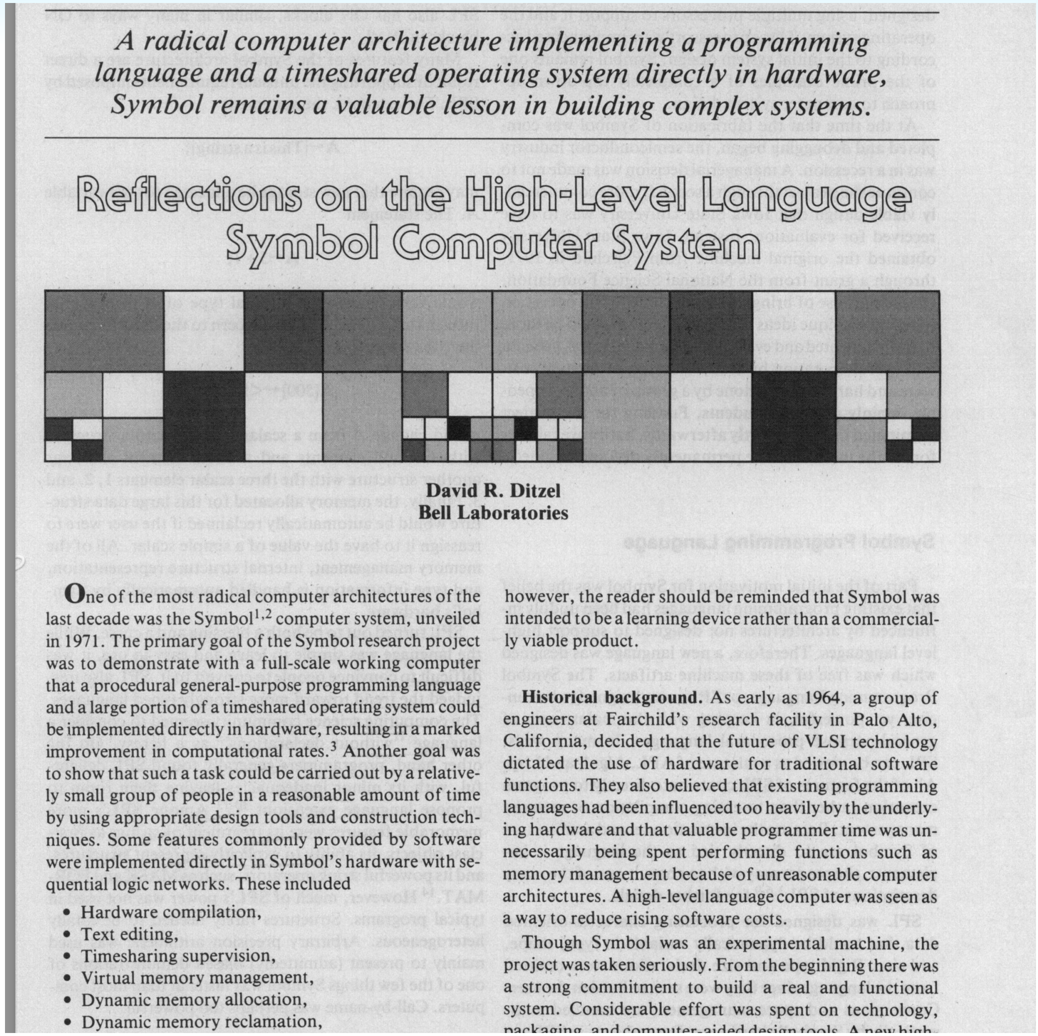
- Esoteric: Aesthetics or no stated advantages.

An almost universal justification for high-level language computers is the view that

> "the prime motivation for developing such a machine is to reduce system costs, for while hardware logic is becoming much cheaper, software is consuming a greater proportion of total system costs. A tremendous savings can be obtained by designing computer hardware that is oriented to aiding the programmer rather than to simplifying the computer designer's job."[22]

The solution to the software problem has appeared to be an increased use of "inexpensive" hardware. According to this viewpoint, the way to use this extra hardware is to raise the level of the machine language, so that in most cases there exists a one-to-one mapping between the

Our "Pre-RISC" paper call for change
ISCA May 1980 (written in 1979)



A radical computer architecture implementing a programming language and a timeshared operating system directly in hardware, Symbol remains a valuable lesson in building complex systems.

Reflections on the High-Level Language Symbol Computer System

David R. Ditzel
Bell Laboratories

One of the most radical computer architectures of the last decade was the Symbol[1,2] computer system, unveiled in 1971. The primary goal of the Symbol research project was to demonstrate with a full-scale working computer that a procedural general-purpose programming language and a large portion of a timeshared operating system could be implemented directly in hardware, resulting in a marked improvement in computational rates.[3] Another goal was to show that such a task could be carried out by a relatively small group of people in a reasonable amount of time by using appropriate design tools and construction techniques. Some features commonly provided by software were implemented directly in Symbol's hardware with sequential logic networks. These included

- Hardware compilation,
- Text editing,
- Timesharing supervision,
- Virtual memory management,
- Dynamic memory allocation,
- Dynamic memory reclamation,

however, the reader should be reminded that Symbol was intended to be a learning device rather than a commercially viable product.

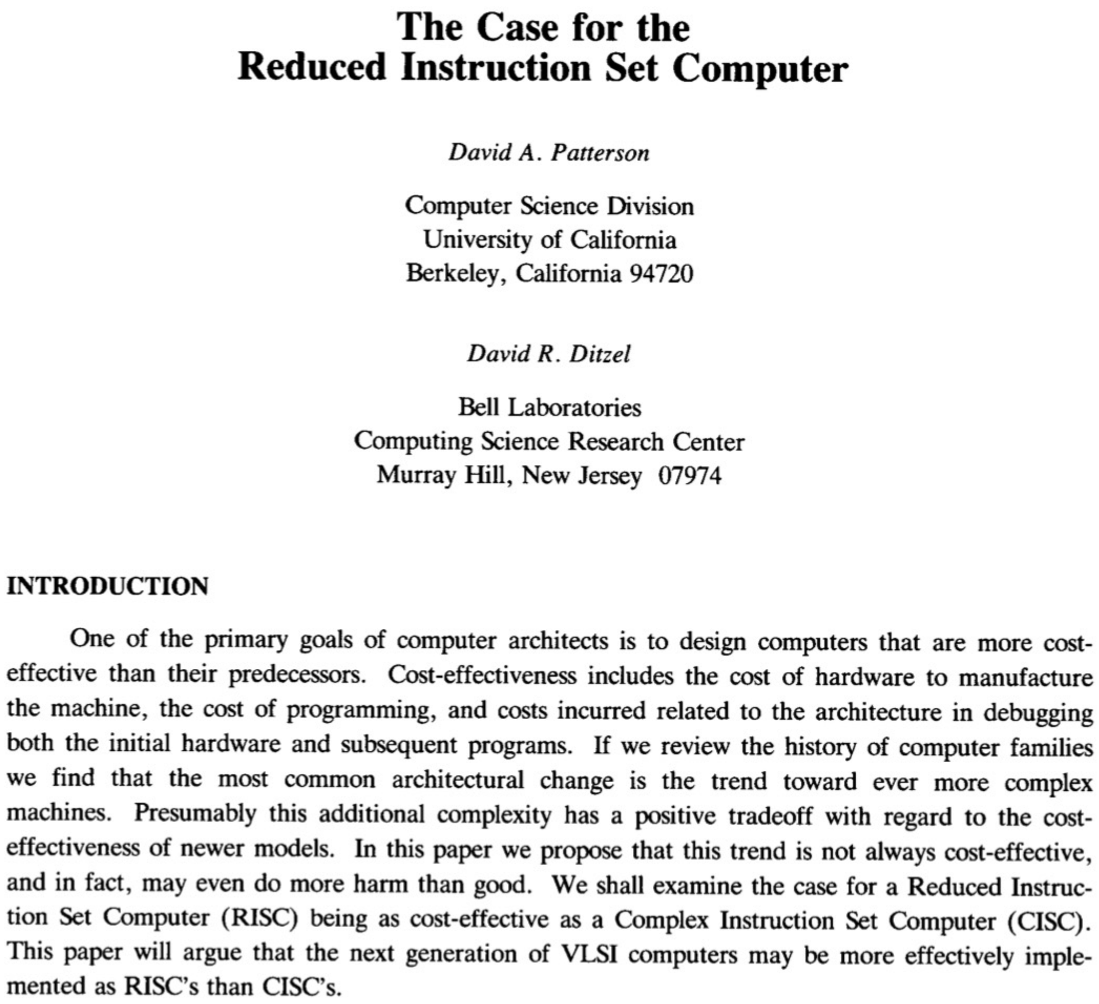
Historical background. As early as 1964, a group of engineers at Fairchild's research facility in Palo Alto, California, decided that the future of VLSI technology dictated the use of hardware for traditional software functions. They also believed that existing programming languages had been influenced too heavily by the underlying hardware and that valuable programmer time was unnecessarily being spent performing functions such as memory management because of unreasonable computer architectures. A high-level language computer was seen as a way to reduce rising software costs.

Though Symbol was an experimental machine, the project was taken seriously. From the beginning there was a strong commitment to build a real and functional system. Considerable effort was spent on technology, packaging, and computer-aided design tools. A new high-

Recognition that SYMBOL/HLLCA was not right direction
IEEE Computer magazine 1981

Reduced Instruction Set Computing

1980: The Case for RISC Published



The Case for the Reduced Instruction Set Computer

David A. Patterson

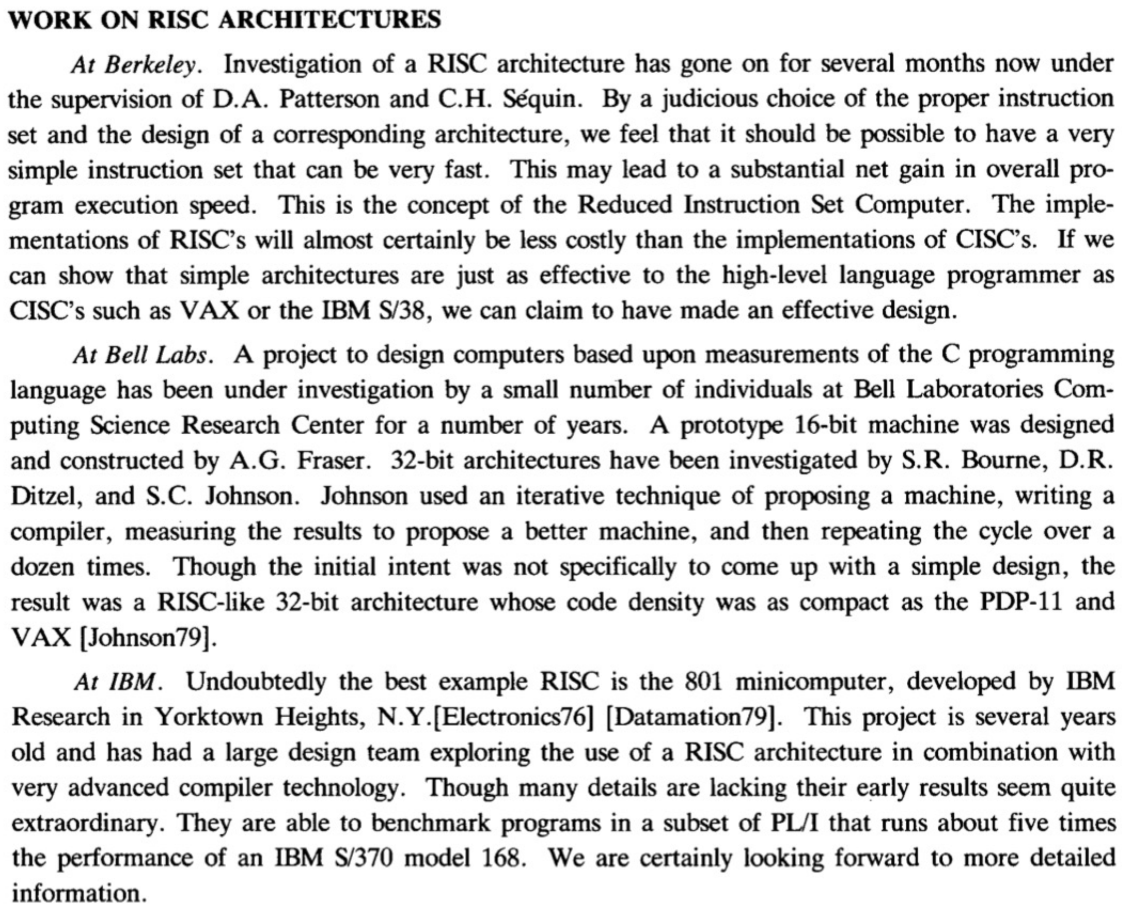
Computer Science Division
University of California
Berkeley, California 94720

David R. Ditzel

Bell Laboratories
Computing Science Research Center
Murray Hill, New Jersey 07974

INTRODUCTION

One of the primary goals of computer architects is to design computers that are more cost-effective than their predecessors. Cost-effectiveness includes the cost of hardware to manufacture the machine, the cost of programming, and costs incurred related to the architecture in debugging both the initial hardware and subsequent programs. If we review the history of computer families we find that the most common architectural change is the trend toward ever more complex machines. Presumably this additional complexity has a positive tradeoff with regard to the cost-effectiveness of newer models. In this paper we propose that this trend is not always cost-effective, and in fact, may even do more harm than good. We shall examine the case for a Reduced Instruction Set Computer (RISC) being as cost-effective as a Complex Instruction Set Computer (CISC). This paper will argue that the next generation of VLSI computers may be more effectively implemented as RISC's than CISC's.



WORK ON RISC ARCHITECTURES

At Berkeley. Investigation of a RISC architecture has gone on for several months now under the supervision of D.A. Patterson and C.H. Séquin. By a judicious choice of the proper instruction set and the design of a corresponding architecture, we feel that it should be possible to have a very simple instruction set that can be very fast. This may lead to a substantial net gain in overall program execution speed. This is the concept of the Reduced Instruction Set Computer. The implementations of RISC's will almost certainly be less costly than the implementations of CISC's. If we can show that simple architectures are just as effective to the high-level language programmer as CISC's such as VAX or the IBM S/38, we can claim to have made an effective design.

At Bell Labs. A project to design computers based upon measurements of the C programming language has been under investigation by a small number of individuals at Bell Laboratories Computing Science Research Center for a number of years. A prototype 16-bit machine was designed and constructed by A.G. Fraser. 32-bit architectures have been investigated by S.R. Bourne, D.R. Ditzel, and S.C. Johnson. Johnson used an iterative technique of proposing a machine, writing a compiler, measuring the results to propose a better machine, and then repeating the cycle over a dozen times. Though the initial intent was not specifically to come up with a simple design, the result was a RISC-like 32-bit architecture whose code density was as compact as the PDP-11 and VAX [Johnson79].

At IBM. Undoubtedly the best example RISC is the 801 minicomputer, developed by IBM Research in Yorktown Heights, N.Y.[Electronics76] [Datamation79]. This project is several years old and has had a large design team exploring the use of a RISC architecture in combination with very advanced compiler technology. Though many details are lacking their early results seem quite extraordinary. They are able to benchmark programs in a subset of PL/I that runs about five times the performance of an IBM S/370 model 168. We are certainly looking forward to more detailed information.

Why RISC

Trend to “reduce the semantic gap” was the wrong approach.

Turns out a few really simple instructions is a good match for compilers

It’s also much better for the hardware

A simple regular RISC instruction set has simpler control and datapaths and:

- Facilitates efficient instruction pipelining
- Leads to higher operating speed and performance
- Leaves more room for larger caches, for higher performance
- Logic that is “left out” doesn’t have bugs, hence fewer bugs. (lately Musk “the best part is no part”)

But in the early 1980’s RISC vs CISC was not so clear.

- Led to many lively conference debates

We had to build real RISC chips to prove the benefits.

Early RISC Principles

RISC: Reduced Instruction Set Computer

- RISC and CISC terms coined by UC Berkeley professors Dave Patterson and Carlo Sequin

Simple orthogonal instructions

- Often fixed 32-bit length
- Easy for compiler code generation
- Easy to implement with small (10-20) gates per cycle
- Easily pipelined

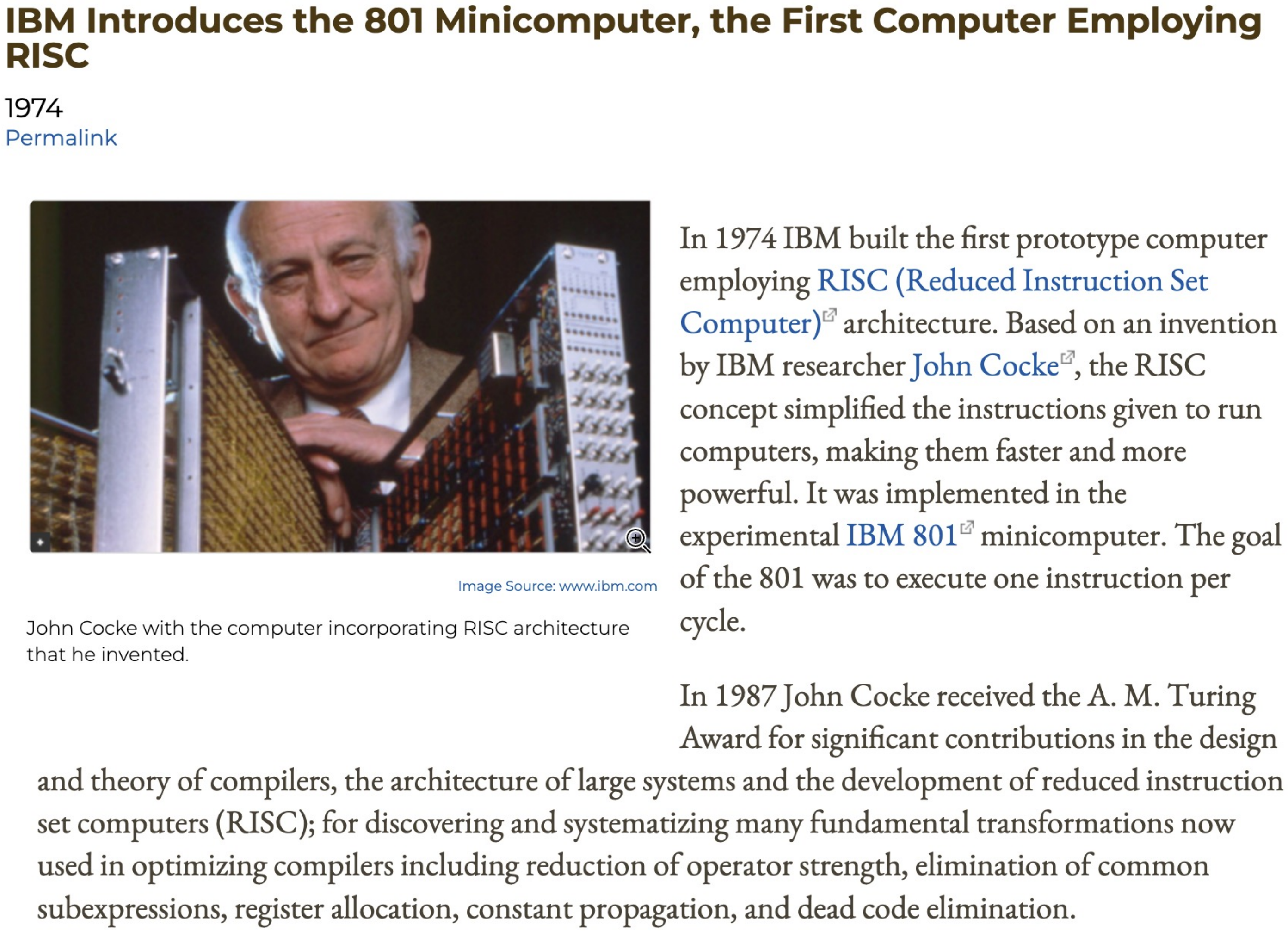
General purpose register file

- Enough to keep data in registers for current and another called subroutine
- Variation in register file style: Flat, Windowed, or invisible (stack cache)

Single load or store per instruction

Register to register arithmetic operations

IBM 801: In my view, the first true RISC



IBM Introduces the 801 Minicomputer, the First Computer Employing RISC

1974
Permalink

Image Source: www.ibm.com

John Cocke with the computer incorporating RISC architecture that he invented.

In 1974 IBM built the first prototype computer employing RISC (Reduced Instruction Set Computer) architecture. Based on an invention by IBM researcher John Cocke, the RISC concept simplified the instructions given to run computers, making them faster and more powerful. It was implemented in the experimental IBM 801 minicomputer. The goal of the 801 was to execute one instruction per cycle.

In 1987 John Cocke received the A. M. Turing Award for significant contributions in the design and theory of compilers, the architecture of large systems and the development of reduced instruction set computers (RISC); for discovering and systematizing many fundamental transformations now used in optimizing compilers including reduction of operator strength, elimination of common subexpressions, register allocation, constant propagation, and dead code elimination.

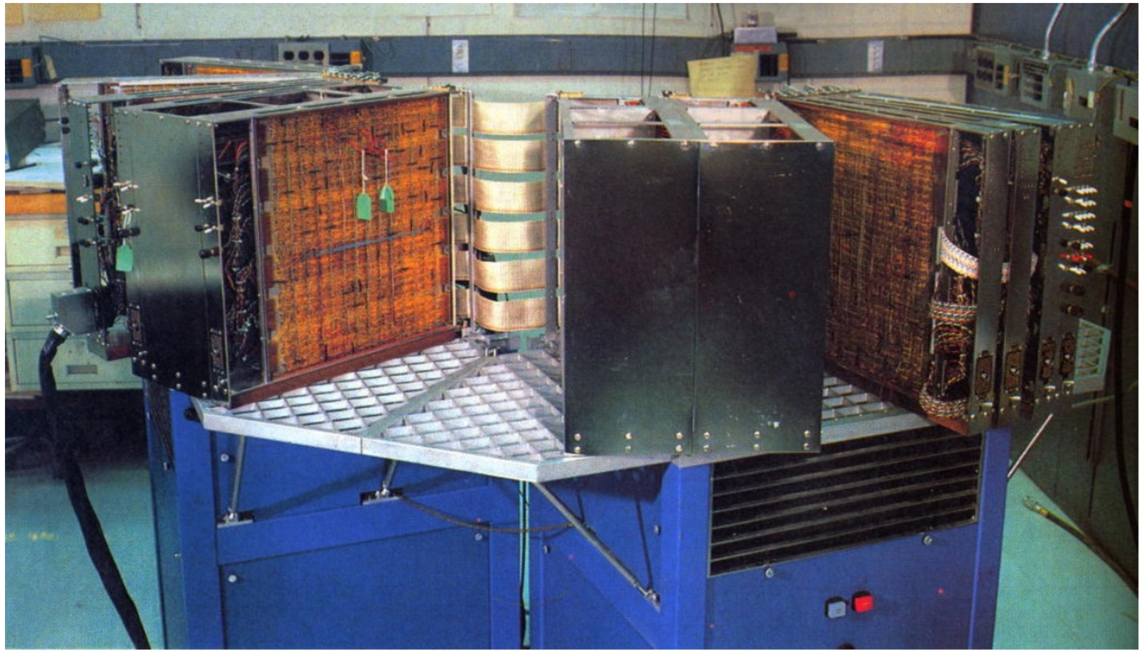
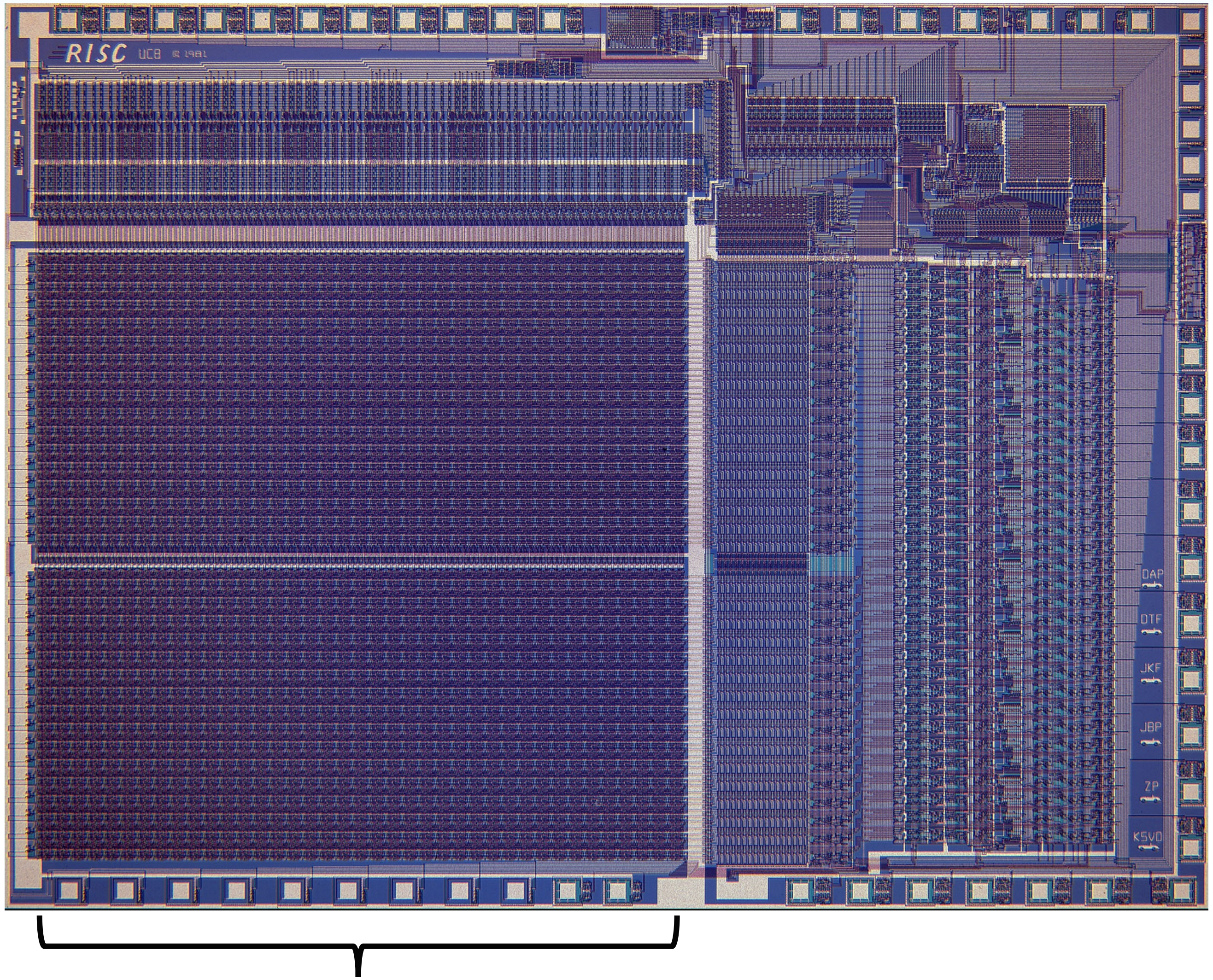


Photo of IBM 801 minicomputer.

The name 801 was from the IBM building number
of the T.J. Watson Research Center in Yorktown Heights

First described at ASPLOS-1 in 1982

UC Berkeley RISC-I ~1981



Register Windows

Most of die consumed with register windows

More registers to keep more operands on die

Single load or store to/from a register

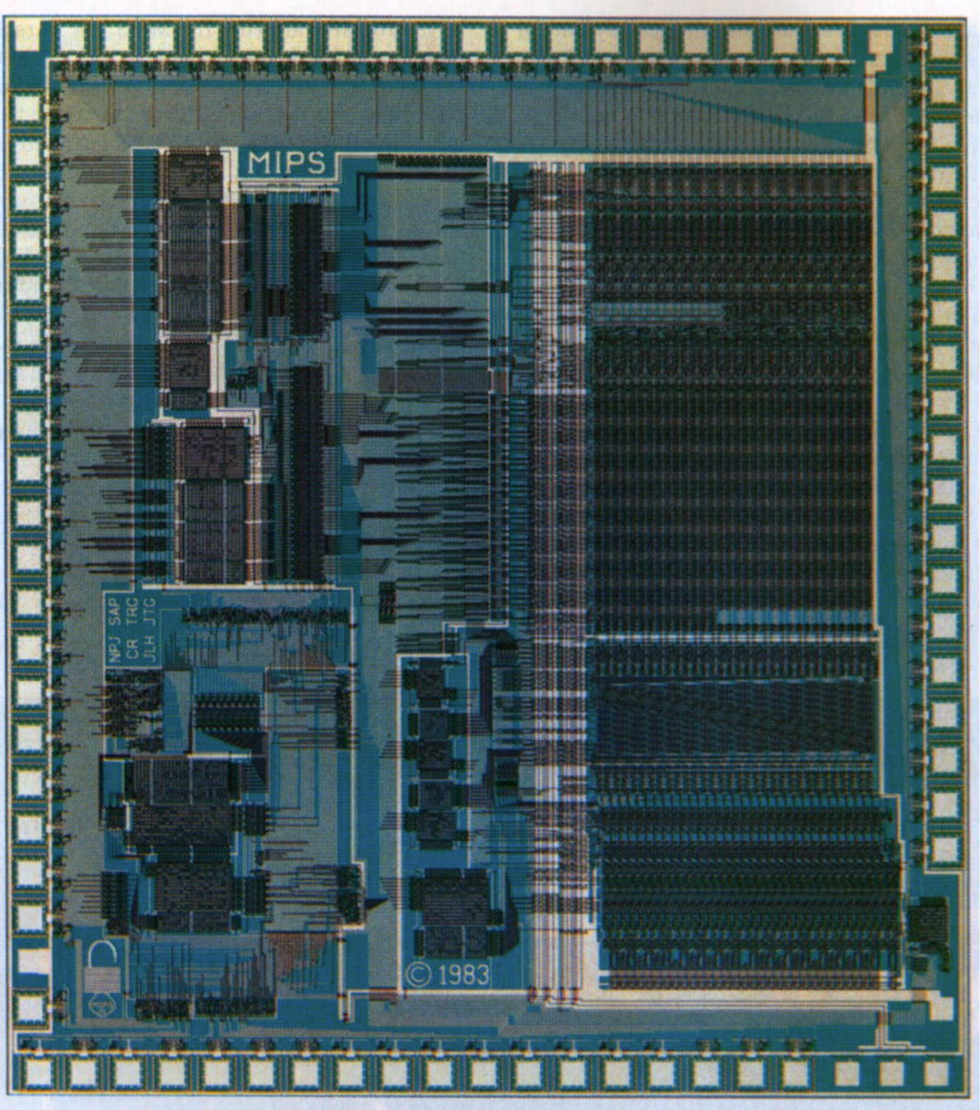
3-address register to register operations

Integer only

No caches

But simple enough for students to design!

Stanford MIPS ~1983



The MIPS instruction set consists of about 111 total instructions

Each instruction encoded in 32-bits

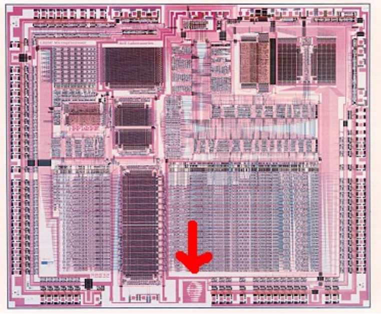
Pipelined to execute one instruction per clock

The instruction set includes:

- 21 arithmetic instructions (+, -, *, /, %)
- 8 logic instructions (&, |, ~)
- 8 bit manipulation instructions
- 12 comparison instructions (>, <, =, >=, <=, ¬)
- 25 branch/jump instructions
- 15 load instructions
- 10 store instructions
- 8 move instructions
- 4 miscellaneous instructions

Delayed branch with 2 delay slots

Early RISC Processors 1978-87: Bell Labs C Machines, including CRISP and Hobbit



AT&T CRISP Microprocessor

C-Language Reduced Instruction Set Processor 1.75 micron CMOS

Announced 1987, 1st CMOS superscalar (more than 1 instr/clock) chip

Hardware translated compact instr into 180-bit wide decoded micro-Op cache

Branch folding eliminated the overhead of branches

“Stack Cache” performed automatic register allocation in hardware

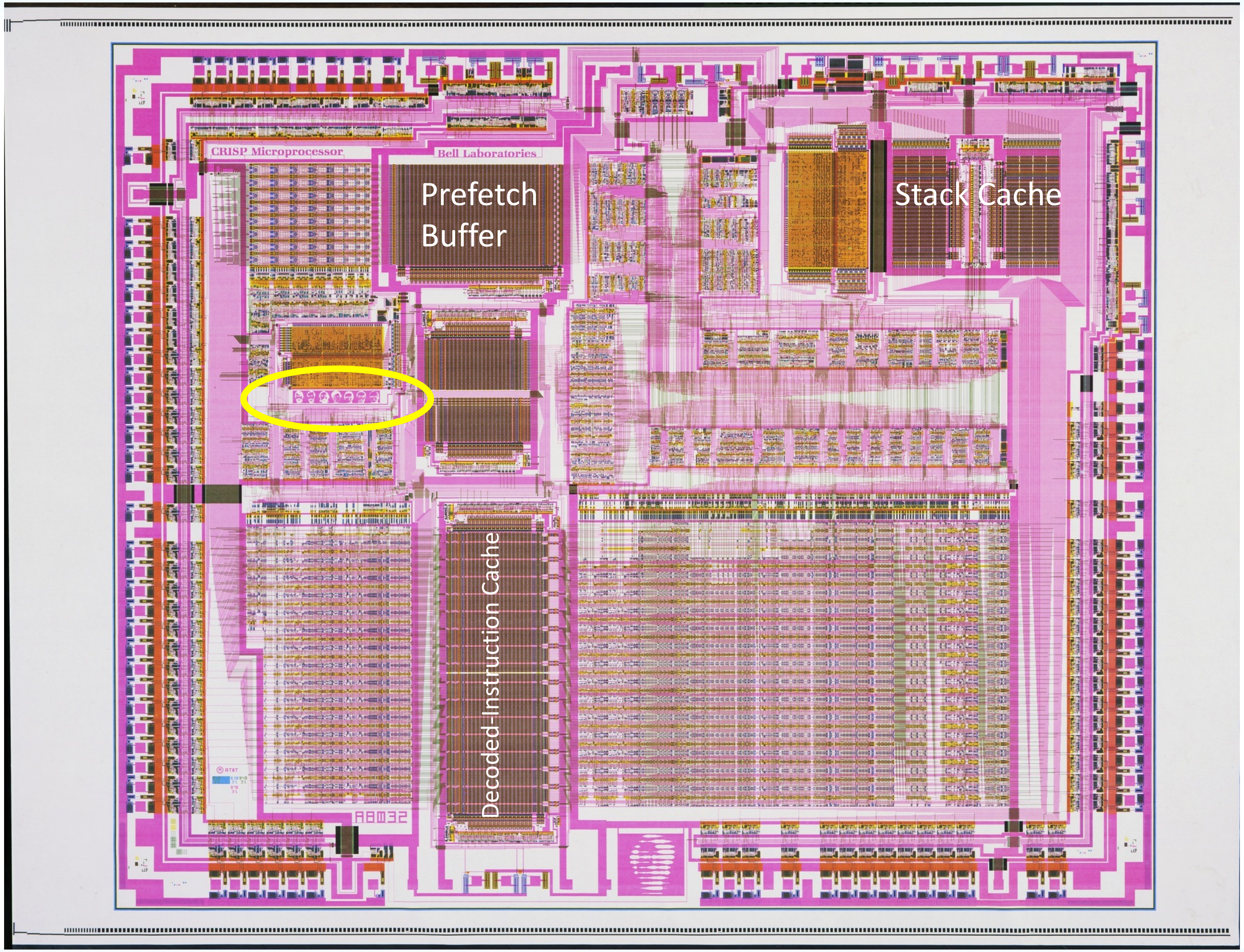
Low power chip, used in EO Tablet computer

Software binary translator used to move software from CISC WE32100

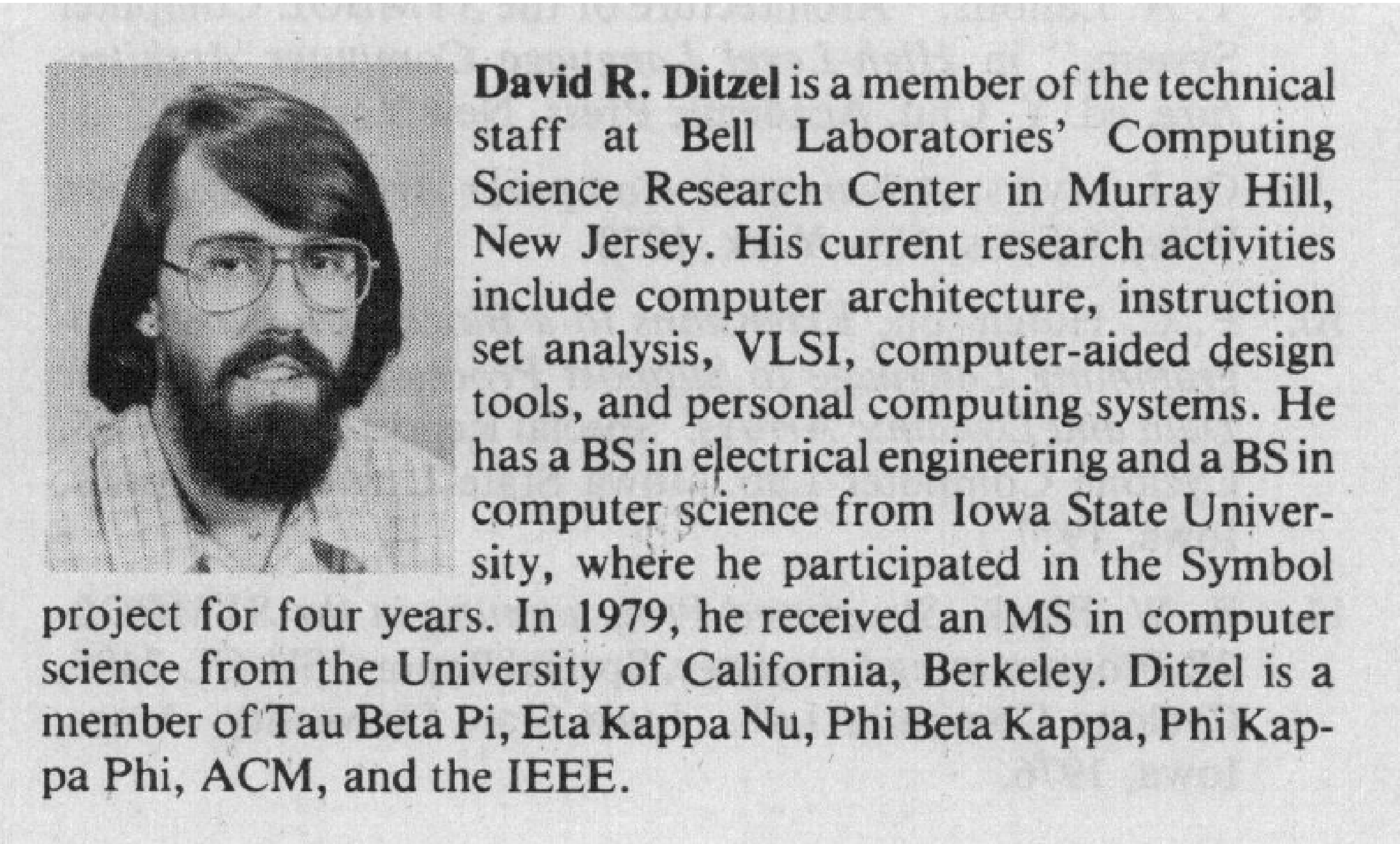
Lessons

- External to internal translation worked well
- Decouple external and internal ISA

AT&T Bell Labs CRISP: C language Reduced Instruction Set Processor

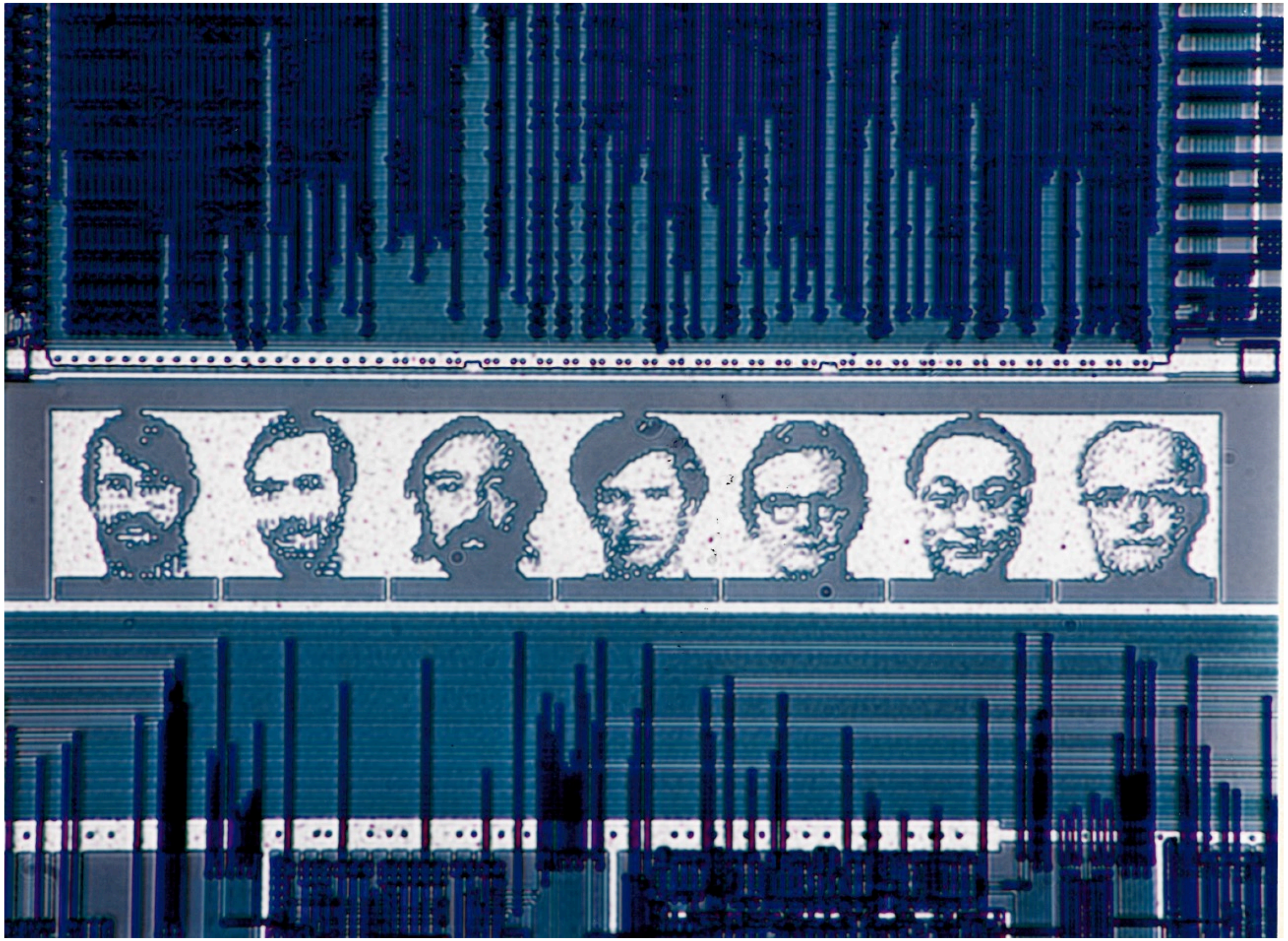


Transition to Bell Labs for 10 years: A more serious look



David R. Ditzel is a member of the technical staff at Bell Laboratories' Computing Science Research Center in Murray Hill, New Jersey. His current research activities include computer architecture, instruction set analysis, VLSI, computer-aided design tools, and personal computing systems. He has a BS in electrical engineering and a BS in computer science from Iowa State University, where he participated in the Symbol project for four years. In 1979, he received an MS in computer science from the University of California, Berkeley. Ditzel is a member of Tau Beta Pi, Eta Kappa Nu, Phi Beta Kappa, Phi Kappa Phi, ACM, and the IEEE.

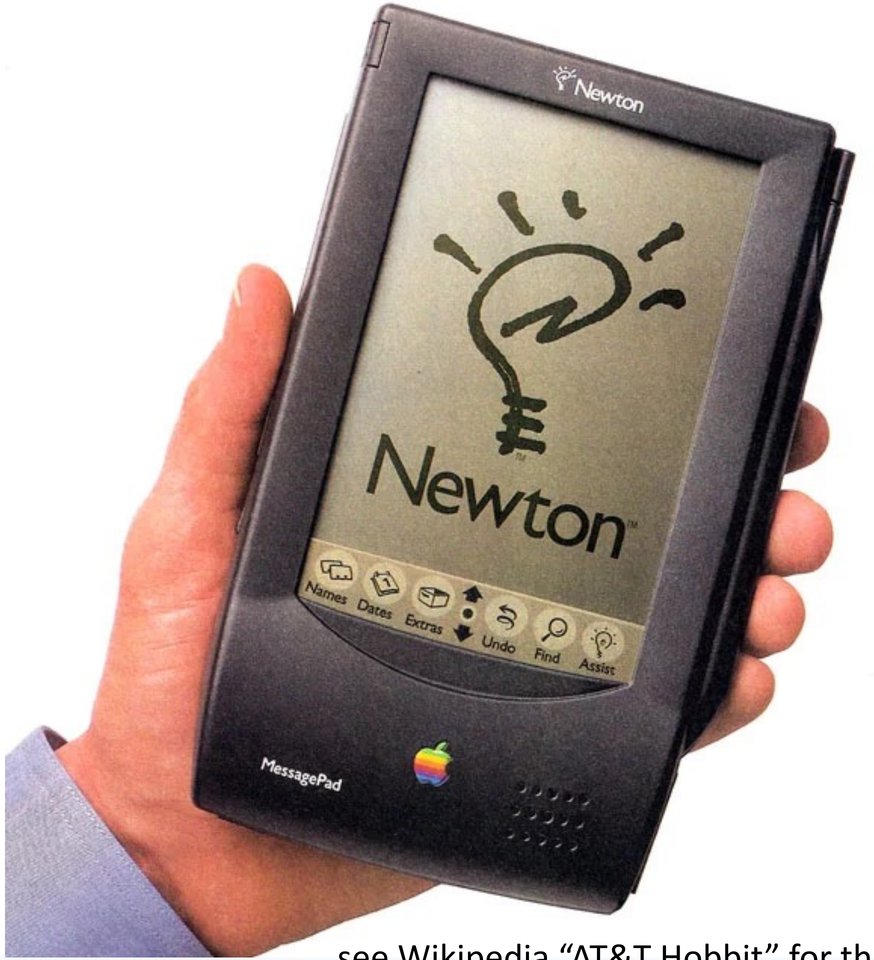
Instead of putting our initials onto the chip…..



CRISP: Wins and losses in early tablets

CRISP made it into AT&T EO Tablet in 1993

Did NOT make it into Apple Newton
…. AT&T fumbled the opportunity
and caused Apple to fund ARM



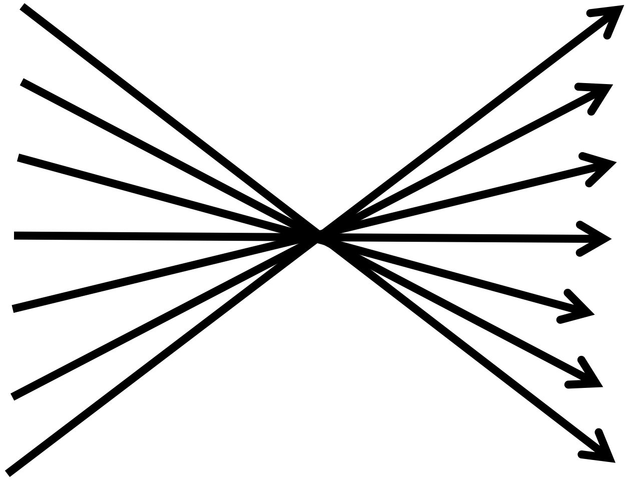
see Wikipedia "AT&T Hobbit" for the story

Lessons in Technical vs Market Success in Hot Chips Presentation (from 2008)

Rank order –

My ISA admiration list

1. DEC Alpha
2. MIPS
3. HP-PA
4. Itanium
5. SPARC
6. IBM Power
7. x86



Rank order –

Recent Market Success

1. x86
2. IBM Power
3. SPARC
4. Itanium
5. HP-PA
6. MIPS
7. DEC Alpha

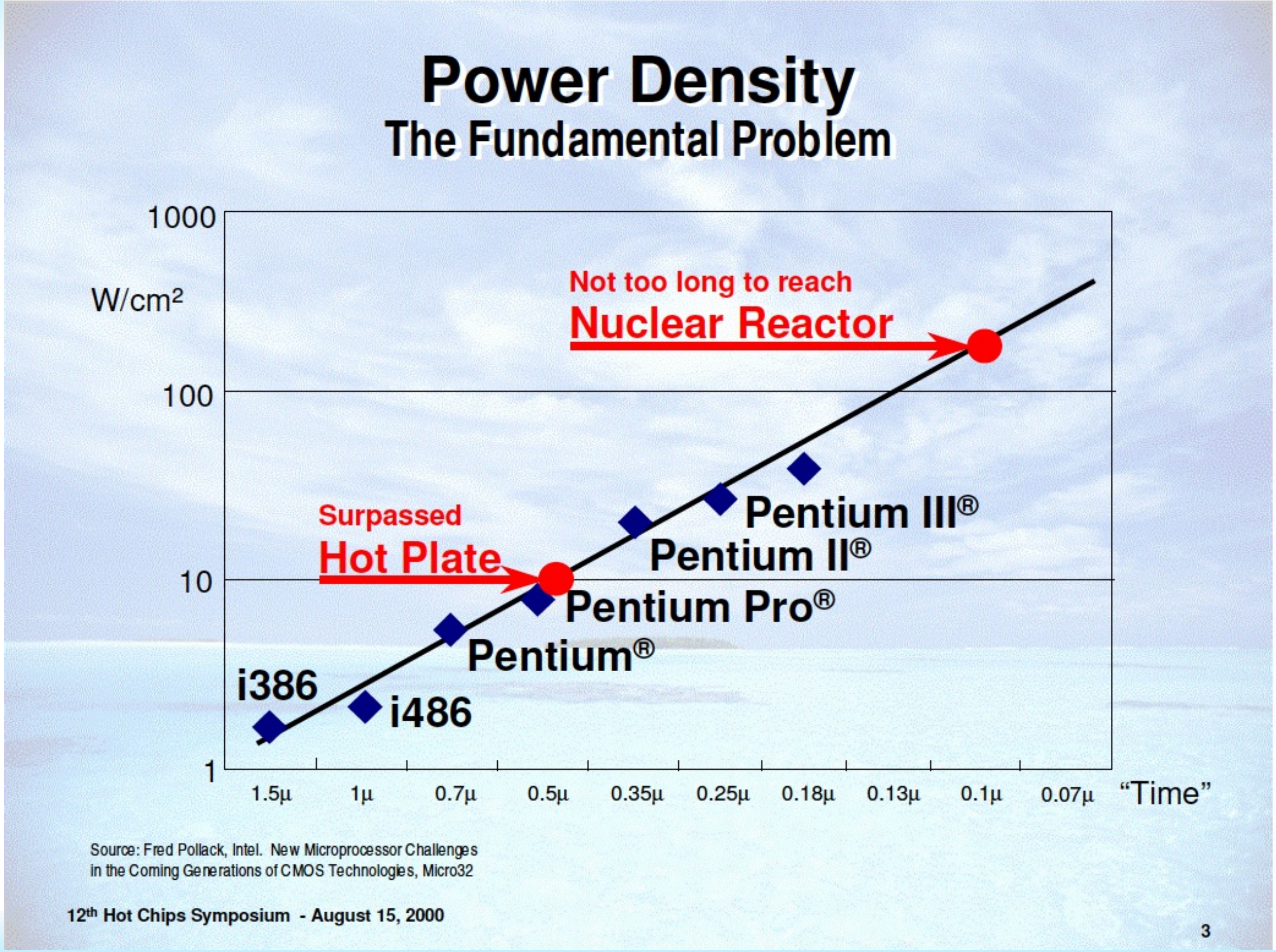
Transmeta 1995-2007:

First to provide full x86 compatibility using software binary translation (Code Morphing) to a simpler processor.

Many low power tricks

We proved that doing low power right can be very exciting

Slide that crystallized the concern over power growth



Power Density

The Fundamental Problem

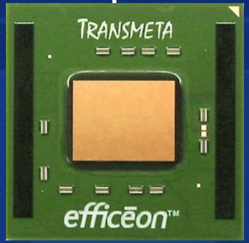
Source: Fred Pollack, Intel. New Microprocessor Challenges in the Coming Generations of CMOS Technologies, Micro32

12th Hot Chips Symposium - August 15, 2000

3

Transmeta low-power x86 Compatible CPU

Efficeon is the sum of

x86 Code Morphing Software +  =

Low Power

x86 PC Compatibility

High Performance

Code Morphing Software

- Provides Compatibility
- Translates the 1’s and 0’s of x86 instructions to equivalent 1’s and 0’s for a simple VLIW processor
- Learns and improves with time

VLIW Hardware

- Very Long Instruction Word processor
- Simple and fast
- Fewer transistors

Bill Gates at Comdex 2000 Keynote announced the Tablet would be the Future of Computing, and held up a Transmeta Crusoe based protoytype.

Microsoft's Tablet PC technology enables any Windows-based application to take advantage of pen-based input.

With software developed by Microsoft, the Tablet PC can function as a sheet of paper.

Handwriting is captured as rich digital ink for immediate or later manipulation, including reformatting and editing.

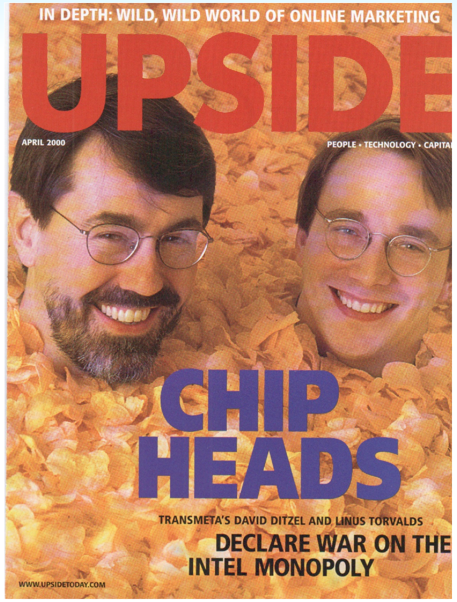
The Tablet PC requires x86 compatibility as it needs to run Windows XP.

This Tablet PC prototype developed by Microsoft demonstrates the concept of tablet computing.

People got really excited about low power Transmeta processors!!

SPECIAL REPORT: THE NET AND SMALL BUSINESS
RED HERRING
THE BUSINESS OF TECHNOLOGY
THIS COULD CHANGE EVERYTHING
Behind Transmeta's bid to power the post-PC era.
(The Crusoe chip, actual size)
RH NO. 76
MARCH 2000
WWW.REDHERRING.COM

NE
3-13
2000
LongRun
Power
Control
Crusoeの船出
ポスト・パソコン時代のマイコン像
解説 プレイステーション2を徹底解剖
解説 「ケータイ」に照準定め，立ち上がる有機EL
NETs特集 「80米ドル，20Gバイト」，AV機器向けHDDが勢ぞろい


IN DEPTH: WILD, WILD WORLD OF ONLINE MARKETING
UPSIDE
APRIL 2000
PEOPLE • TECHNOLOGY • CAPITAL
CHIP HEADS
TRANSMETA'S DAVID DITZEL AND LINUS TORVALDS
DECLARE WAR ON THE INTEL MONOPOLY
WWW.UPSIDETODAY.COM

Hitachi

NEC

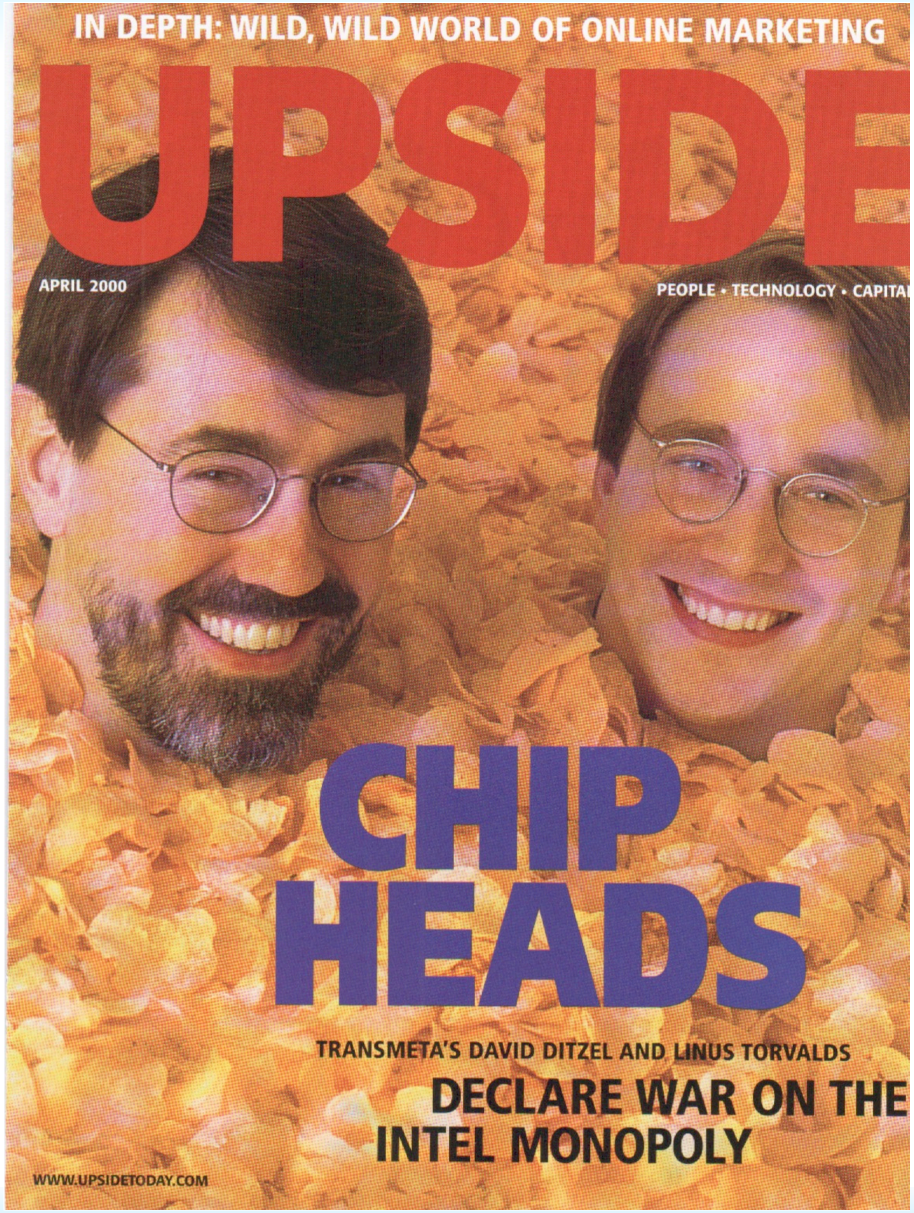
Sharp

Flipstart

OQO

Compaq Tablet PC

Linus Torvalds led the Transmeta Code Morphing Software team


IN DEPTH: WILD, WILD WORLD OF ONLINE MARKETING
UPSIDE
APRIL 2000
PEOPLE • TECHNOLOGY • CAPITAL
CHIP HEADS
TRANSMETA'S DAVID DITZEL AND LINUS TORVALDS
DECLARE WAR ON THE INTEL MONOPOLY
WWW.UPSIDETODAY.COM

Intel 2008-2013

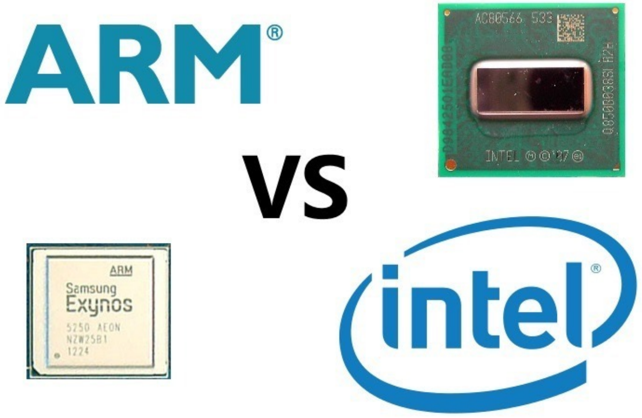
This slide intentionally left blank

x86 tax = 2x

RISC-V

A free and open instruction set

Which instruction set?



Neither ARM nor x86 is very attractive

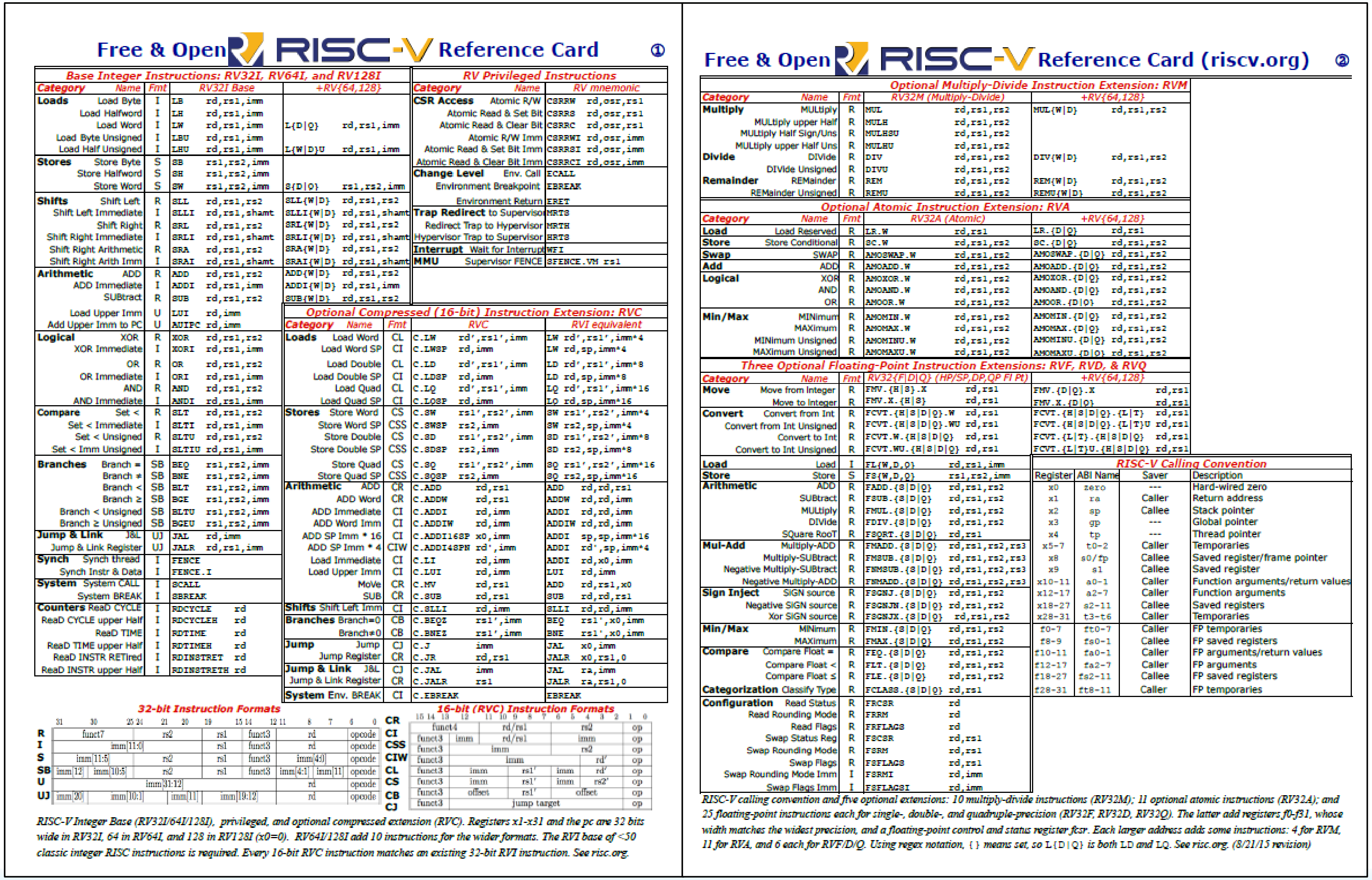
- Proprietary and expensive
- Not very energy efficient

RISC-V is a better choice

New free and open CPU instruction set

- Managed by non-profit RISC-V International
- Over 3000 members
- Simpler ISA, hence more energy efficient
- Allows free or proprietary implementations
  - Already competitive in area and performance
  - Room for improvement
- Growing ecosystem, like early days of Linux

RISC-V Instructions Described on 2 pages
64-bit ARM Document is ~5000 pages



Free & Open RISC-V Reference Card ①

Base Integer Instructions: RV32I, RV64I, and RV128I				
Category	**Name**	**Fmt**	**RV32I Base**	**+RV{64,128}**
Loads	Load Byte	I	LB rd,rs1,imm	
	Load Halfword	I	LH rd,rs1,imm	
	Load Word	I	LW rd,rs1,imm	L{D|Q} rd,rs1,imm
	Load Byte Unsigned	I	LBU rd,rs1,imm	
	Load Half Unsigned	I	LHU rd,rs1,imm	L{W|D}U rd,rs1,imm
Stores	Store Byte	S	SB rs1,rs2,imm	
	Store Halfword	S	SH rs1,rs2,imm	
	Store Word	S	SW rs1,rs2,imm	S{D|Q} rs1,rs2,imm
Shifts	Shift Left	R	SLL rd,rs1,rs2	SLL{W|D} rd,rs1,rs2
	Shift Left Immediate	I	SLLI rd,rs1,shamt	SLLI{W|D} rd,rs1,shamt
	Shift Right	R	SRL rd,rs1,rs2	SRL{W|D} rd,rs1,rs2
	Shift Right Immediate	I	SRLI rd,rs1,shamt	SRLI{W|D} rd,rs1,shamt
	Shift Right Arithmetic	R	SRA rd,rs1,rs2	SRA{W|D} rd,rs1,rs2
	Shift Right Arith Imm	I	SRAI rd,rs1,shamt	SRAI{W|D} rd,rs1,shamt
Arithmetic	ADD	R	ADD rd,rs1,rs2	ADD{W|D} rd,rs1,rs2
	ADD Immediate	I	ADDI rd,rs1,imm	ADDI{W|D} rd,rs1,imm
	SUBtract	R	SUB rd,rs1,rs2	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	UJ	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		
Category	**Name**	**RV mnemonic**
CSR Access	Atomic R/W	CSRRW rd,csr,rs1
	Atomic Read & Set Bit	CSRRS rd,csr,rs1
	Atomic Read & Clear Bit	CSRRC rd,csr,rs1
	Atomic R/W Imm	CSRRWI rd,csr,imm
	Atomic Read & Set Bit Imm	CSRRSI rd,csr,imm
	Atomic Read & Clear Bit Imm	CSRRCI rd,csr,imm
Change Level	Env. Call	ECALL
	Environment Breakpoint	EBREAK
	Environment Return	ERET
Trap Redirect	to Supervisor	MRTS
	Redirect Trap to Hypervisor	MRTH
	Hypervisor Trap to Supervisor	HRTS
Interrupt	Wait for Interrupt	WFI
MMU	Supervisor FENCE	SFENCE.VM rs1

Optional Compressed (16-bit) Instruction Extension: RVC				
Category	**Name**	**Fmt**	**RVC**	**RVI equivalent**
Loads	Load Word	CL	C.LW rd′,rs1′,imm	LW rd′,rs1′,imm*4
	Load Word SP	CI	C.LWSP rd,imm	LW rd,sp,imm*4
	Load Double	CL	C.LD rd′,rs1′,imm	LD rd′,rs1′,imm*8
	Load Double SP	CI	C.LDSP rd,imm	LD rd,sp,imm*8
	Load Quad	CL	C.LQ rd′,rs1′,imm	LQ rd′,rs1′,imm*16
	Load Quad SP	CI	C.LQSP rd,imm	LQ rd,sp,imm*16
Stores	Store Word	CS	C.SW rs1′,rs2′,imm	SW rs1′,rs2′,imm*4
	Store Word SP	CSS	C.SWSP rs2,imm	SW rs2,sp,imm*4
	Store Double	CS	C.SD rs1′,rs2′,imm	SD rs1′,rs2′,imm*8
	Store Double SP	CSS	C.SDSP rs2,imm	SD rs2,sp,imm*8
	Store Quad	CS	C.SQ rs1′,rs2′,imm	SQ rs1′,rs2′,imm*16
	Store Quad SP	CSS	C.SQSP rs2,imm	SQ rs2,sp,imm*16
Arithmetic	ADD	CR	C.ADD rd,rs1	ADD rd,rd,rs1
	ADD Word	CR	C.ADDW rd,rs1	ADDW rd,rd,imm
	ADD Immediate	CI	C.ADDI rd,imm	ADDI rd,rd,imm
	ADD Word Imm	CI	C.ADDIW rd,imm	ADDIW rd,rd,imm
	ADD SP Imm * 16	CI	C.ADDI16SP x0,imm	ADDI sp,sp,imm*16
	ADD SP Imm * 4	CIW	C.ADDI4SPN rd′,imm	ADDI rd′,sp,imm*4
	Load Immediate	CI	C.LI rd,imm	ADDI rd,x0,imm
	Load Upper Imm	CI	C.LUI rd,imm	LUI rd,imm
	MoVe	CR	C.MV rd,rs1	ADD rd,rs1,x0
	SUB	CR	C.SUB rd,rs1	SUB rd,rd,rs1
Shifts	Shift Left Imm	CI	C.SLLI rd,imm	SLLI rd,rd,imm
Branches	Branch=0	CB	C.BEQZ rs1′,imm	BEQ rs1′,x0,imm
	Branch≠0	CB	C.BNEZ rs1′,imm	BNE rs1′,x0,imm
Jump	Jump	CJ	C.J imm	JAL x0,imm
	Jump Register	CR	C.JR rd,rs1	JALR x0,rs1,0
Jump & Link	J&L	CJ	C.JAL imm	JAL ra,imm
	Jump & Link Register	CR	C.JALR rs1	JALR ra,rs1,0
System	Env. BREAK	CI	C.EBREAK	EBREAK

32-bit Instruction Formats

	31	30 25	24 21	20	19 15	14 12	11 8	7	6 0
R	funct7		rs2		rs1	funct3	rd		opcode
I	imm[11:0]				rs1	funct3	rd		opcode
S	imm[11:5]		rs2		rs1	funct3	imm[4:0]		opcode
SB	imm[12]	imm[10:5]	rs2		rs1	funct3	imm[4:1]	imm[11]	opcode
U	imm[31:12]						rd		opcode
UJ	imm[20]	imm[10:1]		imm[11]	imm[19:12]		rd		opcode

16-bit (RVC) Instruction Formats

	15 14 13	12	11 10	9 8 7	6 5	4 3 2	1 0
CR	funct4		rd/rs1		rs2		op
CI	funct3	imm	rd/rs1		imm		op
CSS	funct3	imm			rs2		op
CIW	funct3	imm				rd′	op
CL	funct3	imm		rs1′	imm	rd′	op
CS	funct3	imm		rs1′	imm	rs2′	op
CB	funct3	offset		rs1′	offset		op
CJ	funct3	jump target					op

RISC-V Integer Base (RV32I/64I/128I), privileged, and optional compressed extension (RVC). Registers x1-x31 and the pc are 32 bits wide in RV32I, 64 in RV64I, and 128 in RV128I (x0=0). RV64I/128I add 10 instructions for the wider formats. The RVI base of <50 classic integer RISC instructions is required. Every 16-bit RVC instruction matches an existing 32-bit RVI instruction. See risc.org.

Free & Open RISC-V Reference Card (riscv.org) ②

Optional Multiply-Divide Instruction Extension: RVM				
Category	**Name**	**Fmt**	**RV32M (Multiply-Divide)**	**+RV{64,128}**
Multiply	MULtiply	R	MUL rd,rs1,rs2	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 rd,rs1,rs2	DIV{W|D} rd,rs1,rs2
	DIVide Unsigned	R	DIVU rd,rs1,rs2	
Remainder	REMainder	R	REM rd,rs1,rs2	REM{W|D} rd,rs1,rs2
	REMainder Unsigned	R	REMU rd,rs1,rs2	REMU{W|D} rd,rs1,rs2

Optional Atomic Instruction Extension: RVA				
Category	**Name**	**Fmt**	**RV32A (Atomic)**	**+RV{64,128}**
Load	Load Reserved	R	LR.W rd,rs1	LR.{D|Q} rd,rs1
Store	Store Conditional	R	SC.W rd,rs1,rs2	SC.{D|Q} rd,rs1,rs2
Swap	SWAP	R	AMOSWAP.W rd,rs1,rs2	AMOSWAP.{D|Q} rd,rs1,rs2
Add	ADD	R	AMOADD.W rd,rs1,rs2	AMOADD.{D|Q} rd,rs1,rs2
Logical	XOR	R	AMOXOR.W rd,rs1,rs2	AMOXOR.{D|Q} rd,rs1,rs2
	AND	R	AMOAND.W rd,rs1,rs2	AMOAND.{D|Q} rd,rs1,rs2
	OR	R	AMOOR.W rd,rs1,rs2	AMOOR.{D|Q} rd,rs1,rs2
Min/Max	MINimum	R	AMOMIN.W rd,rs1,rs2	AMOMIN.{D|Q} rd,rs1,rs2
	MAXimum	R	AMOMAX.W rd,rs1,rs2	AMOMAX.{D|Q} rd,rs1,rs2
	MINimum Unsigned	R	AMOMINU.W rd,rs1,rs2	AMOMINU.{D|Q} rd,rs1,rs2
	MAXimum Unsigned	R	AMOMAXU.W rd,rs1,rs2	AMOMAXU.{D|Q} rd,rs1,rs2

Three Optional Floating-Point Instruction Extensions: RVF, RVD, & RVQ				
Category	**Name**	**Fmt**	**RV32{F|D|Q} (HP/SP,DP,QP Fl Pt)**	**+RV{64,128}**
Move	Move from Integer	R	FMV.{H|S}.X rd,rs1	FMV.{D|Q}.X rd,rs1
	Move to Integer	R	FMV.X.{H|S} rd,rs1	FMV.X.{D|Q} rd,rs1
Convert	Convert from Int	R	FCVT.{H|S|D|Q}.W rd,rs1	FCVT.{H|S|D|Q}.{L|T} rd,rs1
	Convert from Int Unsigned	R	FCVT.{H|S|D|Q}.WU rd,rs1	FCVT.{H|S|D|Q}.{L|T}U rd,rs1
	Convert to Int	R	FCVT.W.{H|S|D|Q} rd,rs1	FCVT.{L|T}.{H|S|D|Q} rd,rs1
	Convert to Int Unsigned	R	FCVT.WU.{H|S|D|Q} rd,rs1	FCVT.{L|T}U.{H|S|D|Q} rd,rs1
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	FNMSUB.{S|D|Q} rd,rs1,rs2,rs3	
	Negative Multiply-ADD	R	FNMADD.{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	
Categorization	Classify Type	R	FCLASS.{S|D|Q} rd,rs1	
Configuration	Read Status	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	

RISC-V Calling Convention			
Register	**ABI Name**	**Saver**	**Description**
x0	zero	---	Hard-wired zero
x1	ra	Caller	Return address
x2	sp	Callee	Stack pointer
x3	gp	---	Global pointer
x4	tp	---	Thread pointer
x5-7	t0-2	Caller	Temporaries
x8	s0/fp	Callee	Saved register/frame pointer
x9	s1	Callee	Saved register
x10-11	a0-1	Caller	Function arguments/return values
x12-17	a2-7	Caller	Function arguments
x18-27	s2-11	Callee	Saved registers
x28-31	t3-t6	Caller	Temporaries
f0-7	ft0-7	Caller	FP temporaries
f8-9	fs0-1	Callee	FP saved registers
f10-11	fa0-1	Caller	FP arguments/return values
f12-17	fa2-7	Caller	FP arguments
f18-27	fs2-11	Callee	FP saved registers
f28-31	ft8-11	Caller	FP temporaries

RISC-V calling convention and five optional extensions: 10 multiply-divide instructions (RV32M); 11 optional atomic instructions (RV32A); and 25 floating-point instructions each for single-, double-, and quadruple-precision (RV32F, RV32D, RV32Q). The latter add registers f0-f31, whose width matches the widest precision, and a floating-point control and status register fcsr. Each larger address adds some instructions: 4 for RVM, 11 for RVA, and 6 each for RVF/D/Q. Using regex notation, {} means set, so L{D|Q} is both LD and LQ. See risc.org. (8/21/15 revision)

RISC-V likely to flourish

RISC-V is likely to be the first highly successful open instruction set

Already lots of implementations, some open source, some not

Very successful already for cost sensitive embedded applications

Expect high performance multi-issue out of order versions for high end: SiFive, Ventana, etc

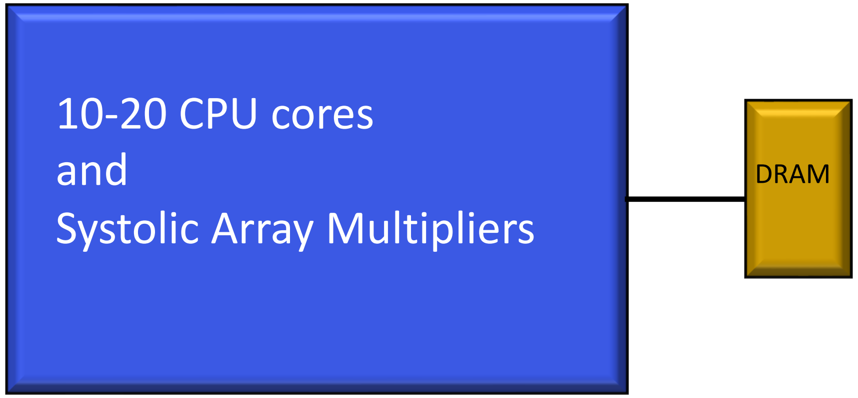
The open nature of the ISA is attractive to many

Present day state of the art:

Esperanto puts over a thousand RISC-V cores on a chip

Esperanto's approach is different... and we think better for ML Recommendation

Other ML Chip approaches



One Giant Hot Chip uses up power budget

Limited I/O pin budget limits memory BW

Dependence on systolic array multipliers

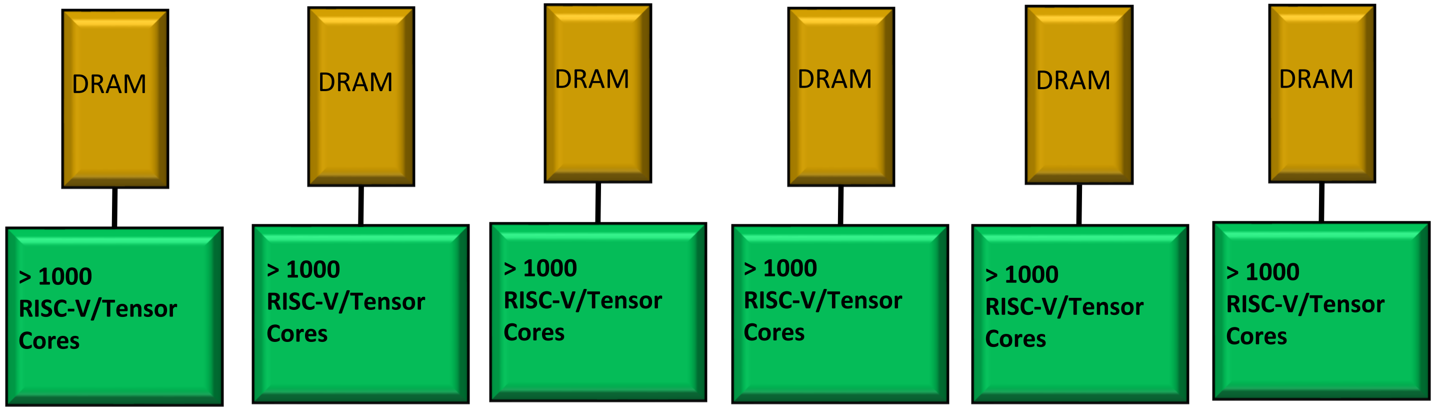
- Great for high ResNet50 score
- Not so good with large sparse memory

Only a **handful (10-20) of CPU cores**

- **Limited parallelism** with CPU cores when problem doesn't fit onto array multiplier

Standard voltage: Not energy efficient

Esperanto's better approach


DRAM
> 1000 RISC-V/Tensor Cores
DRAM
> 1000 RISC-V/Tensor Cores
DRAM
> 1000 RISC-V/Tensor Cores
DRAM
> 1000 RISC-V/Tensor Cores
DRAM
> 1000 RISC-V/Tensor Cores
DRAM
> 1000 RISC-V/Tensor Cores

Use **multiple low-power** chips that still fit within power budget

Performance, pins, memory, bandwidth **scale up with more chips**

Thousands of general-purpose RISC-V/tensor cores

- **Far more programmable** than overly-specialized (eg systolic) hw
- **Thousands of threads** help with large sparse memory latency

Full parallelism of thousands of cores always available

Low-voltage operation of transistors **is more energy-efficient**

- Lower voltage operation also reduces power
- Requires both **circuit and architecture innovations**

Challenge: How to keep the power of each chip to < 20 watts?

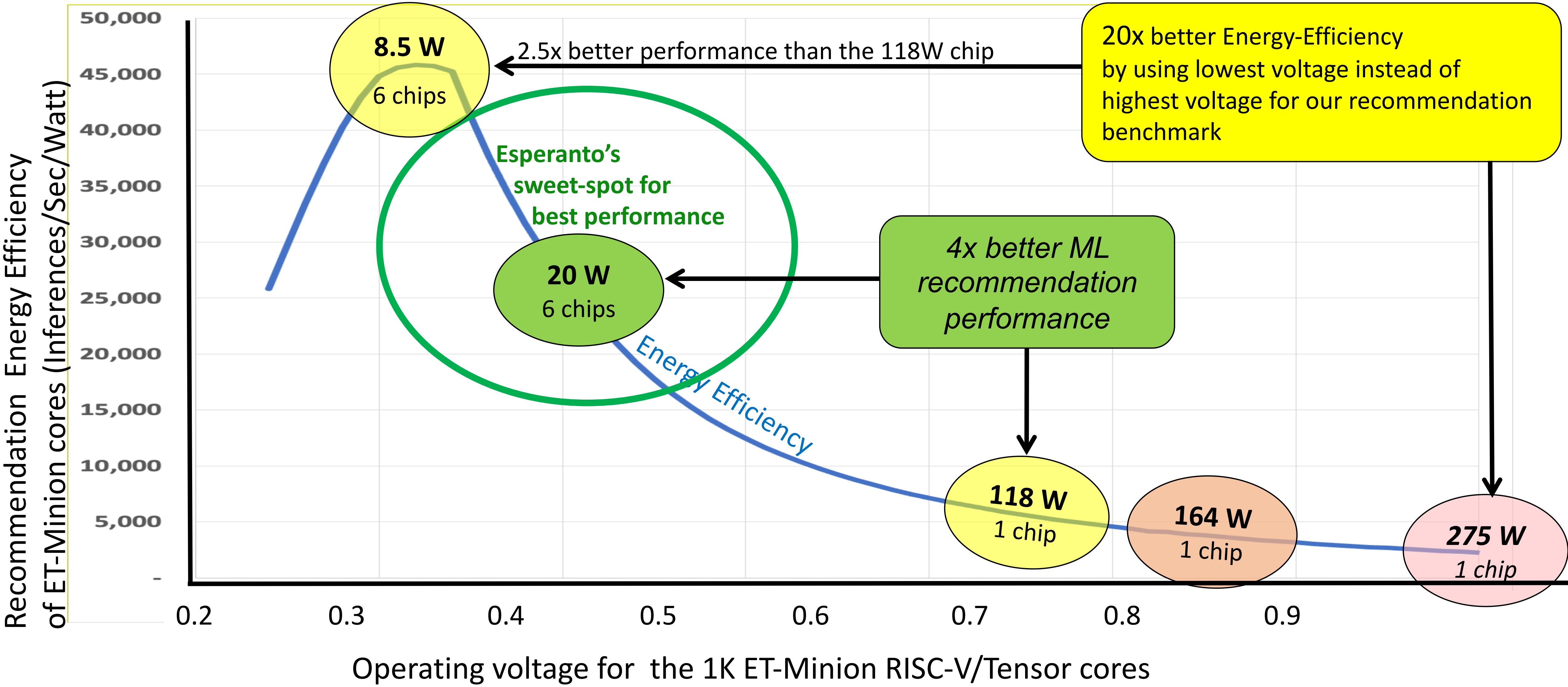
Challenge was to put >1000 RISC-V Cores in a 20 Watt chip

Assumed half of 20W power for 1K RISC-V cores, so only 10 mW per core!

$$\textbf{Power (Watts)} \quad = \quad \mathbf{C_{dynamic}} \ \mathbf{x} \ \mathbf{Voltage^{2}} \ \mathbf{x} \ \mathbf{Frequency + Leakage}$$

	Power/core	Frequency	Voltage	Cdynamic
Generic x86 Server core (165W for 24 cores)	7 W	3 GHz	0.850v	2.2nF
10mW ET-Minion core (~10W for 1K cores)	0.01 W	1 GHz	0.425v	0.04nF
Reductions needed to hit goals	~700x	3x	4x	58x
		Easy	Hard Circuit/SRAM	Very Hard Architecture

Study of energy-efficiency and number of chips to get best ML Performance in 120 watts (six 20W chips)



2.5x better performance than the 118W chip

20x better Energy-Efficiency by using lowest voltage instead of highest voltage for our recommendation benchmark

Esperanto's sweet-spot for best performance

4x better ML recommendation performance

Cluster of CPUs (Shires) to Become Next Unit of Compute

For Machine Learning or any highly parallel application wanting to use hundreds to thousands of CPUs, much better to consider the *basic unit of compute to be a cluster of CPUs and memory*, rather than just isolated individual processors.

Gain dramatic advantages by designing CPUs to work together on large problems: new "Parallel CPU"

- Increase performance
- Reduce area with smaller cores
- Reduce power by enabling simpler design
- Further reduce power (~4x) through cooperation between cores, both data and instruction work

Need to consider memory issues

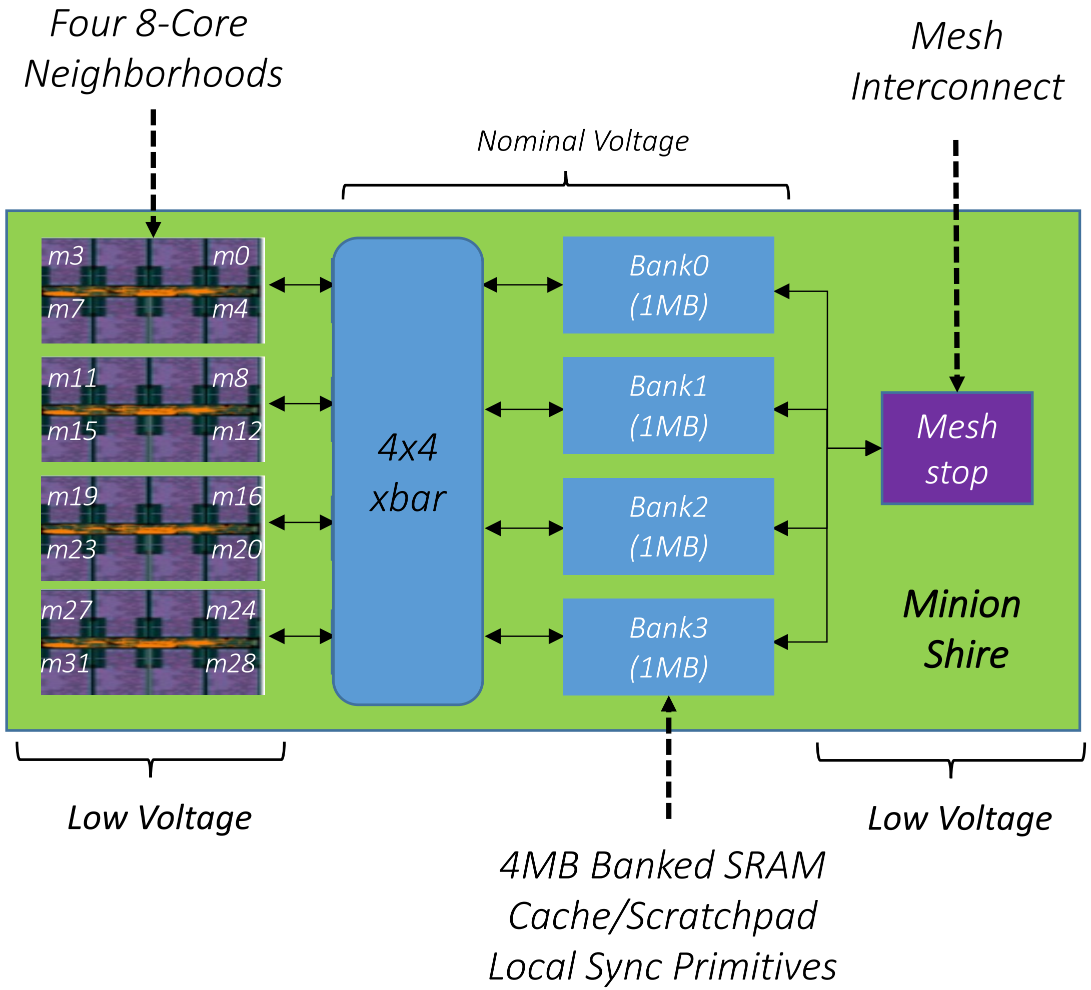
- L2, L2, L3 cache and path the main memory

Need to consider interconnect

- NoC-to-NoC interconnect between clusters

Esperanto has a highly optimized solution working in silicon today as a great new "unit of compute"

32 ET-Minion CPUs and 4 MB Memory form a “Minion Shire” cluster



32 ET-MINION RISC-V CORES PER MINION SHIRE

Arranged in four 8-core neighborhoods

SOFTWARE CONFIGURABLE MEMORY HIERARCHY

L1 data cache can also be configured as scratchpad

Four 1MB SRAM banks can be partitioned as private L2, shared L3 and scratchpad

SHIRES CONNECTED WITH MESH NETWORK

NEW SYNCHRONIZATION PRIMITIVES

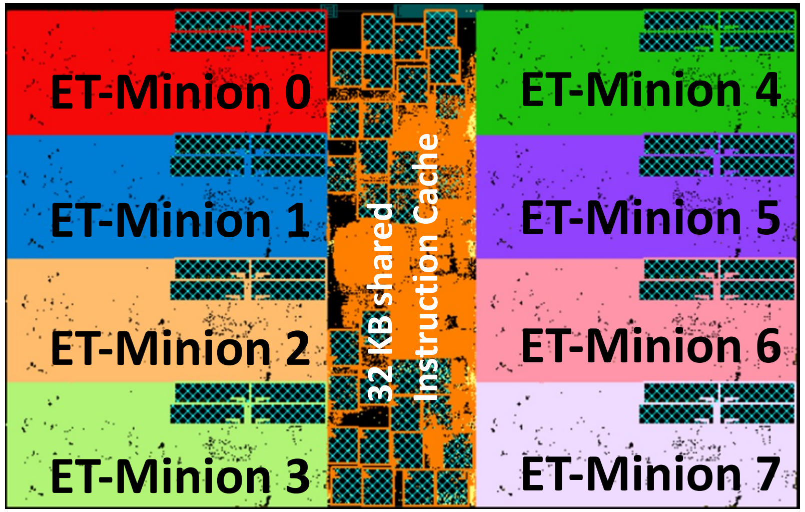
Fast local atomics

Fast local barriers

Fast local credit counter

IPI support

8 ET-Minions form a “Neighborhood”



NEIGHBORHOOD CORES WORK CLOSELY TOGETHER

- Architecture improvements capitalize on physical proximity of 8 cores
- Take advantage that almost always running highly parallel code

OPTIMIZATIONS FROM CORES RUNNNING THE SAME CODE

- 8 ET-Minions share single large instruction cache,
  this is more energy efficient than many separate instruction caches.
- “Cooperative loads” substantially reduce memory traffic to L2 cache

NEW INSTRUCTIONS MAKE COOPERATION MORE EFFICIENT

- New Tensor instructions dramatically cut back on instruction fetch bandwidth
- New instructions for fast local synchronization within group
- New Send-to-Neighbor instructions
- New Receive-from-Neighbor instructions

ET-Minion is an Energy-Efficient RISC-V CPU with a Vector/Tensor Unit

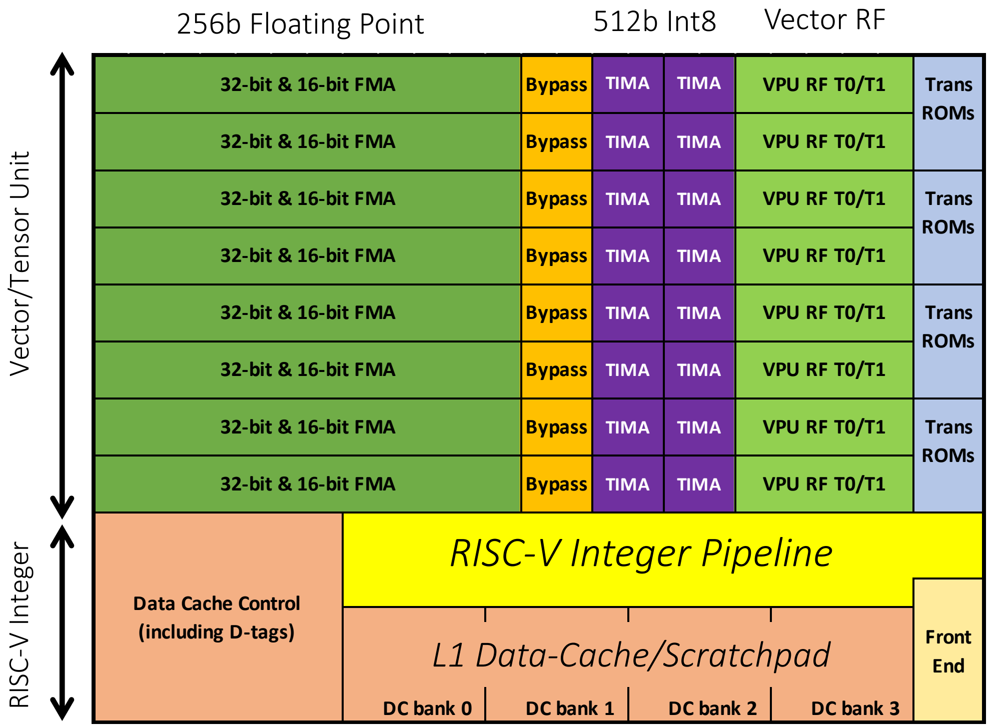
ET-MINION IS A CUSTOM BUILT 64-BIT RISC-V PROCESSOR

- In-order pipeline with low gates/stage to improve MHz at low voltages
- Architecture and circuits optimized to enable low-voltage operation
- Two hardware threads of execution
- Software configurable L1 data-cache and/or scratchpad

ML OPTIMIZED VECTOR/TENSOR UNIT

- 512-bit wide integer per cycle
  - 128 8-bit integer operations per cycle, accumulates to 32-bit Int
- 256-bit wide floating point per cycle
  - 16 32-bit single precision operations per cycle
  - 32 16-bit half precision operations per cycle
- New multi-cycle Tensor Instructions
  - Can run for up to 512 cycles with one tensor instruction (32K ops)
  - Reduces instruction fetch bandwidth and reduces power
  - RISC-V integer pipeline put to sleep during tensor instructions
- Vector transcendental instructions

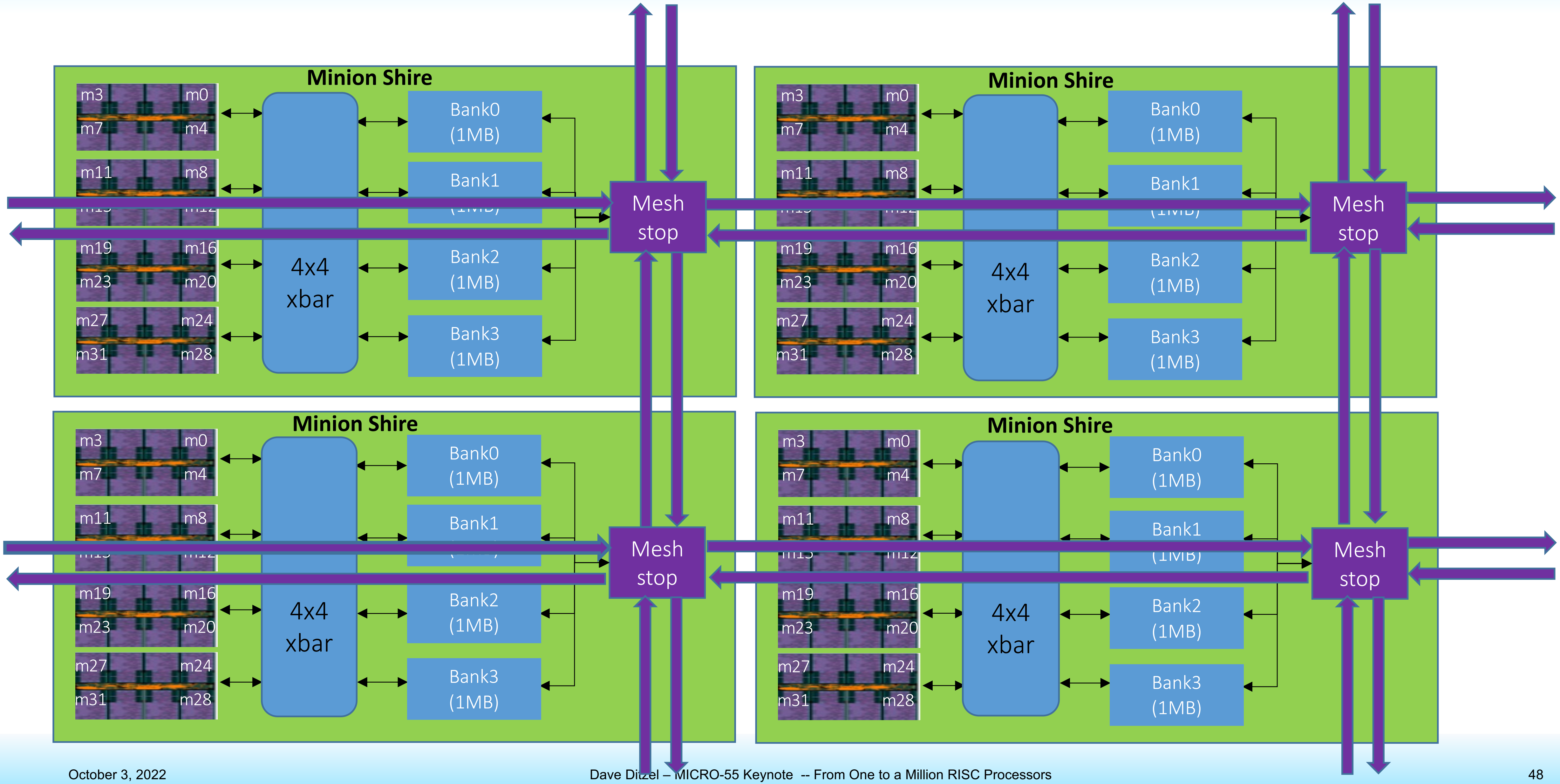
OPERATING RANGE: 300 MHz TO 2 GHz



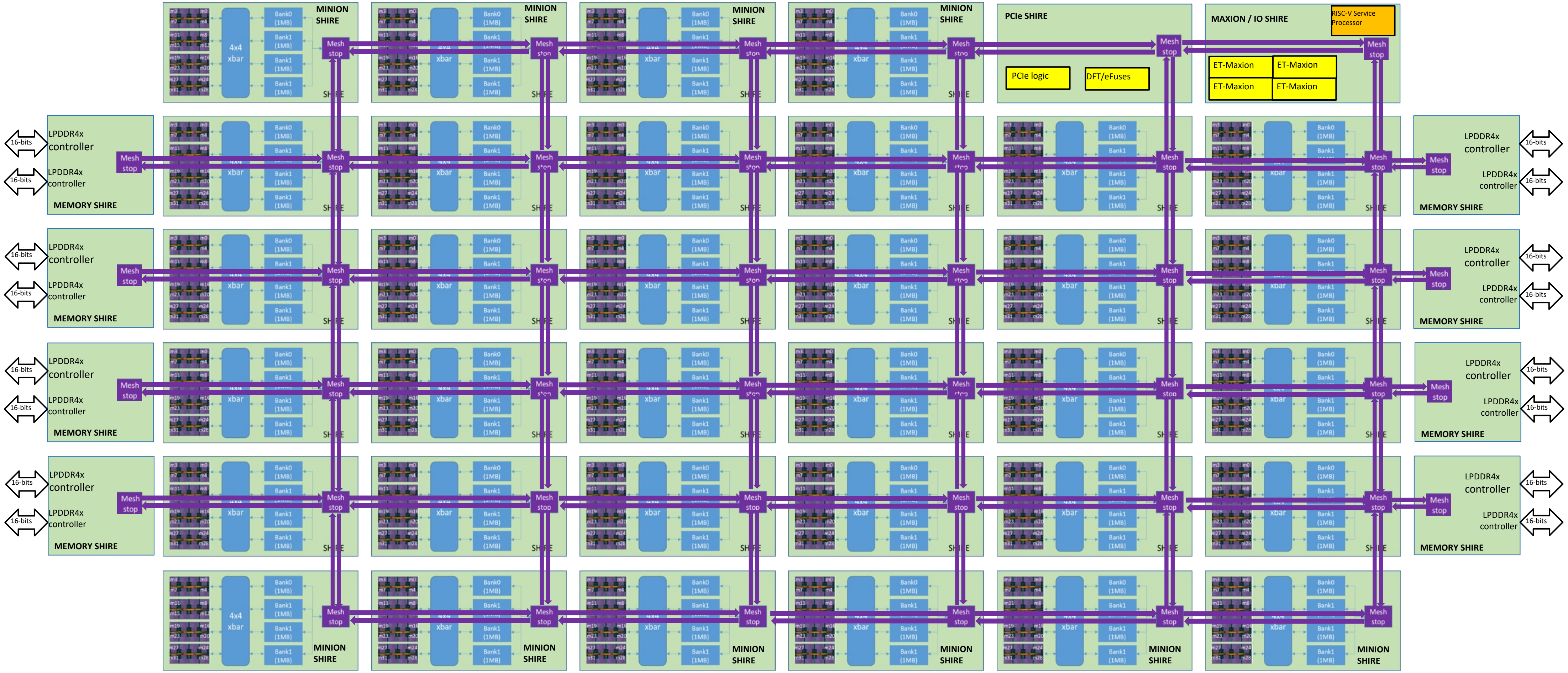
ET-Minion RISC-V Core and Tensor/Vector unit optimized for low-voltage operation
to improve energy-efficiency

Optimized for energy-efficient ML operations. Each ET-Minion can deliver peak of 128 Int8 GOPS per GHz

Shires are connected to each other and to external memory through Mesh Network



ET-SoC-1: Full chip internal block diagram



34 MINION SHIRES

- 1088 ET-Minions

8 MEMORY SHIRES

- **LPDDR4x DRAM controllers**

1 MAXION / IO SHIRE

- **4 ET-Maxions**
- **1 RISC-V Service Processor**

PCIe SHIRE

160 million bytes of on-die SRAM

x8 PCIe Gen4

Secure Root of Trust

ET-SoC-1 External Chip Interfaces

8-bit PCIe Gen4

- Root/endpoint/both

256-bit wide LPDDR4x

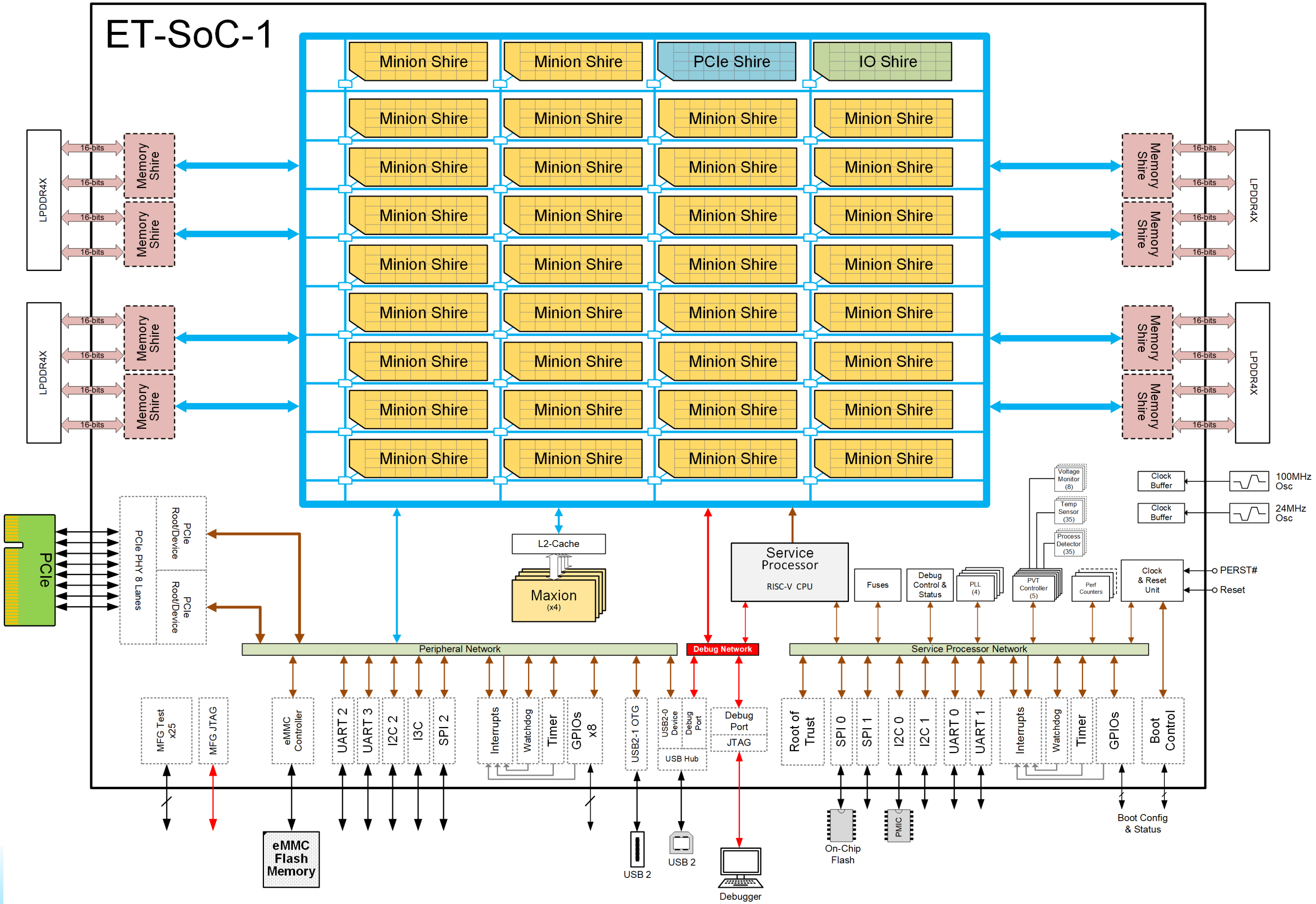
- 4267 MT/s
- 137 GB/s
- ECC support

RISC-V SERVICE PROCESSOR

- Secure Boot
- System Management
- Watchdog timers
- eFuse

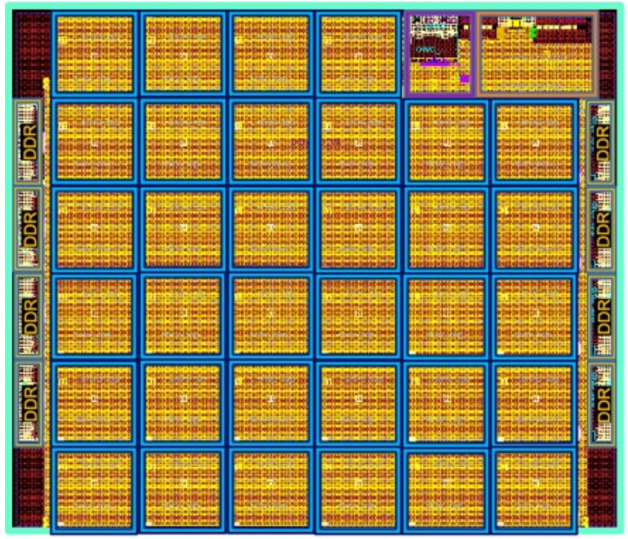
EXTERNAL IO

- SMBus
- Serial – I2C/SPI/UART
- GPIO
- FLASH

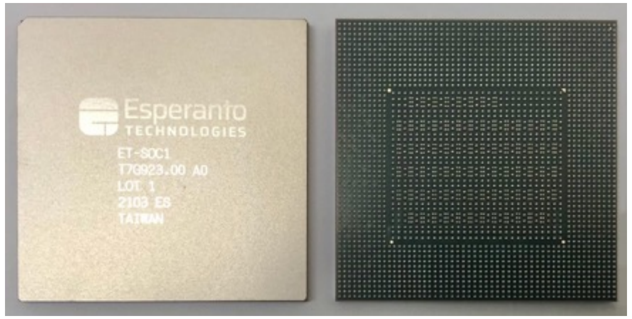


Summary Statistics and Status of ET-SoC-1

- **ET-SoC-1 is fabricated in TSMC 7nm**
  - 24 billion transistors
  - Die-area: 570 mm^2
- **1088 ET-Minion energy-efficient 64-bit RISC-V processors**
  - Each with an attached vector/tensor unit
  - Typical operation 300 MHz to 1 GHz
- **4 ET-Maxion 64-bit high-performance RISC-V out-of-order processors**
  - Typical operation 500 MHz to 1.0 GHz
- **Over 160 million bytes of on-die SRAM used for caches and scratchpad memory**
- **ET-SoC-1 Power ~ 20 watts, can be adjusted for 10 to 60+ watts under SW control**
- **Package: 45x45mm with 2494 balls to PCB, over 30,000 bumps to die**
- **Status**
  - First silicon is healthy and has been shipped to paying customers

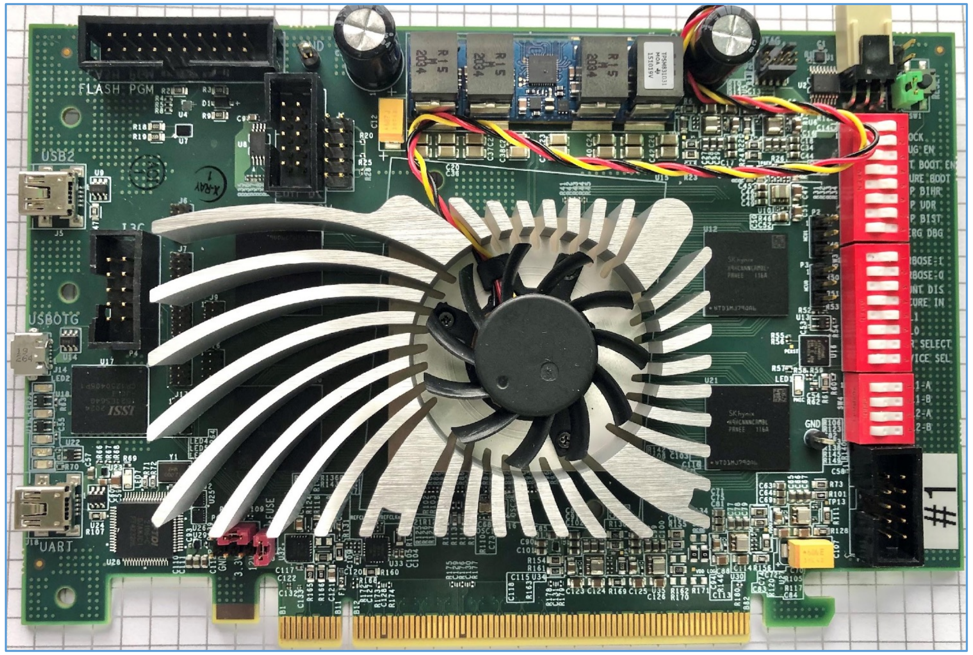


ET-SoC-1 Die Plot



ET-SoC-1 Package

Esperanto's ET-SoC-1 PCIe Evaluation Card

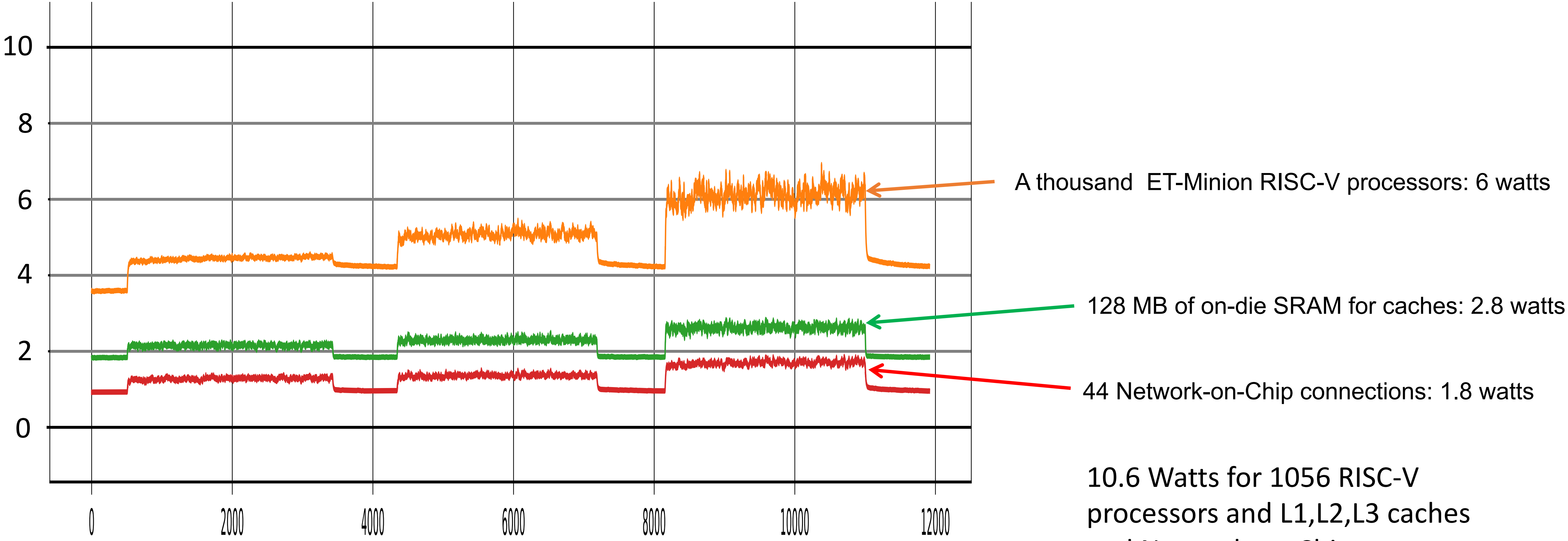


Esperanto PCI Express Accelerator Card:

- Single ET-SoC-1 chip running at 300 MHz to 1 GHz
- 1088 ET-Minion cores and 4 ET-Maxion cores
- 1056 ET-Minion cores provide acceleration
- 8 lanes of PCIe Gen 4
- ET-SoC-1 power can be configured from 10W to to 60W per chip depending on customer requirements
- 16GB or 32GB DRAM
- Typically used as accelerator to x86 host
- Now being sold in servers for customer evaluations

Esperanto Minion Shire Power with ML Recommendation Benchmarks

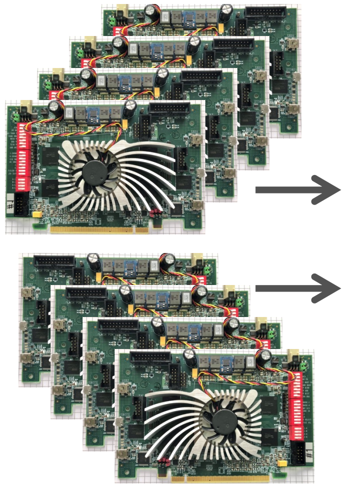
Power in Watts for 33 Minion Shires (1056 RISC-V cores)

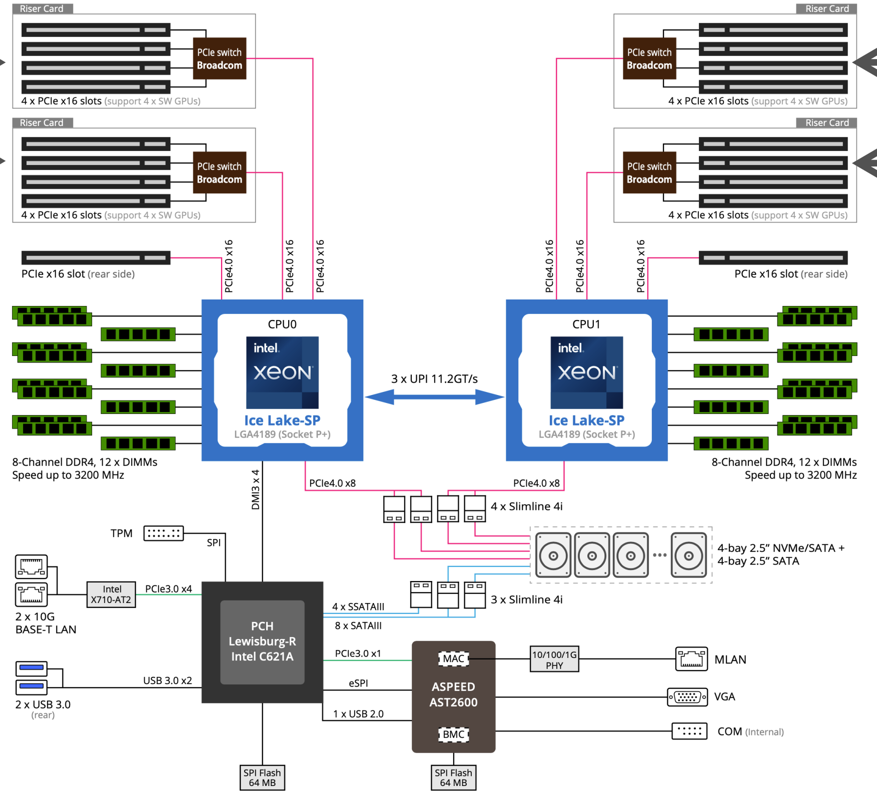


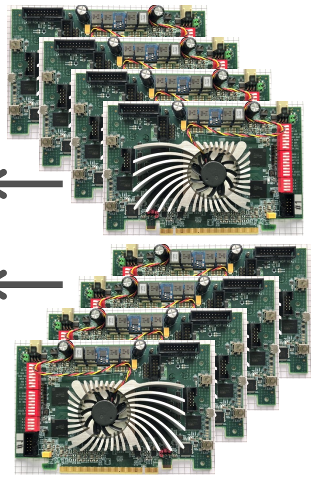
10.6 Watts for 1056 RISC-V processors and L1,L2,L3 caches and Network-on-Chip averages 10.0 mW per RISC-V core

Today: Over 300,000 RISC-V Vector Processors in a single rack

16 Esperanto cards can go into each 2U high server







This server has 2 x86 host CPUs and holds 16 Esperanto PCIe cards, each card with one ET-SoC-1 chip with 1088 RISC-V processors, for a total of 17,408 RISC-V processors.

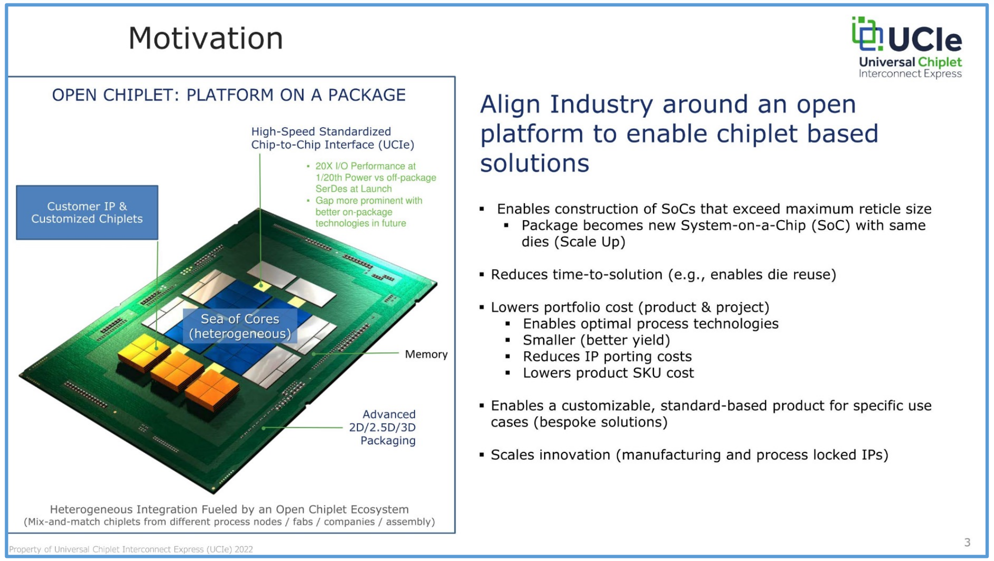
Put 20 servers in a single rack, and that would be 348,160 RISC-V processors.

320 Esperanto PCIe cards in a standard 42U high rack using 20 server chassis, leaving 2U for TOR switch.

What’s next for RISC?

Chiplets and a lot more cores

Industry standard UCIe bus promoting chiplet interoperability announced March 2, 2022



Motivation

UCIe Universal Chiplet Interconnect Express

OPEN CHIPLET: PLATFORM ON A PACKAGE

High-Speed Standardized Chip-to-Chip Interface (UCIe)

- 20X I/O Performance at 1/20th Power vs off-package SerDes at Launch
- Gap more prominent with better on-package technologies in future

Customer IP & Customized Chiplets

Sea of Cores (heterogeneous)

Memory

Advanced 2D/2.5D/3D Packaging

Heterogeneous Integration Fueled by an Open Chiplet Ecosystem
(Mix-and-match chiplets from different process nodes / fabs / companies / assembly)

Align Industry around an open platform to enable chiplet based solutions

- Enables construction of SoCs that exceed maximum reticle size
  - Package becomes new System-on-a-Chip (SoC) with same dies (Scale Up)
- Reduces time-to-solution (e.g., enables die reuse)
- Lowers portfolio cost (product & project)
  - Enables optimal process technologies
  - Smaller (better yield)
  - Reduces IP porting costs
  - Lowers product SKU cost
- Enables a customizable, standard-based product for specific use cases (bespoke solutions)
- Scales innovation (manufacturing and process locked IPs)

Property of Universal Chiplet Interconnect Express (UCIe) 2022

3

UCIe Universal Chiplet Interconnect Express

PROMOTERS

Leaders in semiconductors, packaging, IP suppliers, foundries, and cloud service providers are joining together to drive a new open chiplet ecosystem.

JOIN US!

AMD arm ASE GROUP

Google Cloud intel. Meta

Microsoft Qualcomm SAMSUNG tsmc

Property of Universal Chiplet Interconnect Express (UCIe) 2022

8

Achronix Data Acceleration ADVANTEST 芯耀辉 | AkroStar alchip

Alphawave IP ANALOG DEVICES apmemory ARTERIS IP

AsteraLabs 芯云凌 Atrontech AyarLabs AVERY design systems

BLUE CHEETAH ANALOG DESIGN Brite semiconductor BROADCOM cadence

cxmt 长鑫存储技术有限公司 ChangXin Memory Technologies, Inc. DECA etopus we accelerate FUTUREWEI Technologies

GUC The Advanced ASIC Leader INNOSILICON IBM 辉羲智能 RHINO

InPsytech JCET JUNIPER NETWORKS kiwi moore

KNIULINK 牛芯半导体 M² SEMI 超摩科技 MARVELL MEDIATEK

Micron MICROCHIP nepes nepes corporation NEURON IP

oppo proteanTecs Rivos SOFICS Solutions For ICs

SK hynix socionext 希姆计算 STREAM COMPUTING SYNOPSYS Silicon to Software

TF 通富微電 TONGFU MICROELECTRONICS CO., LTD. VeriSilicon XPEEDIC ACCELERATE YOUR IC DESIGN 浙江力积存储科技有限公司

芯云凌 Atrontech BlueOcean Smart cxmt 长鑫存储技术有限公司

DECA einfochips An Arrow Company The Solutions People KNIULINK 牛芯半导体

M² InPsytech PHISON

PSMC 力晶積成電子 SmartDV

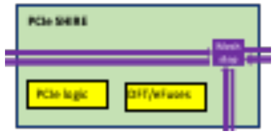
Standardized Chiplet bus is a big deal.

Will accelerate Chiplet development.

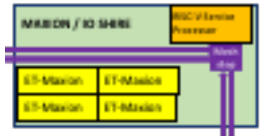
Chiplets will affect your future, so get ready.

Esperanto's Next Generation will use a Chiplet-based implementation

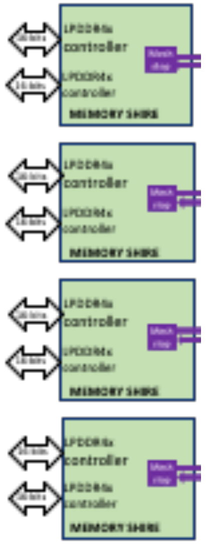
Easy for Esperanto to make chiplet products, cut into pieces and shrink from 7nm to 3nm



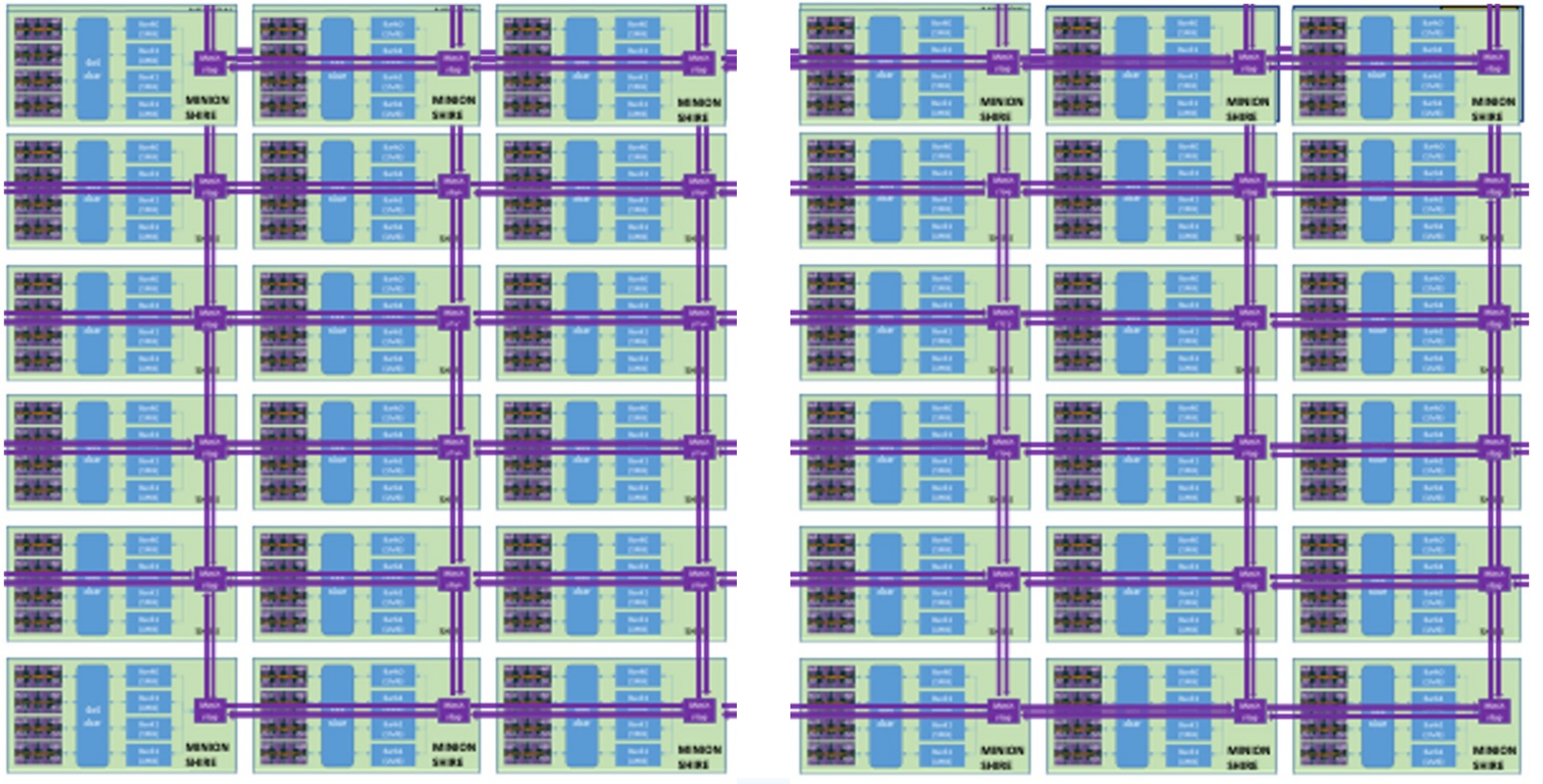
I/O Chiplet

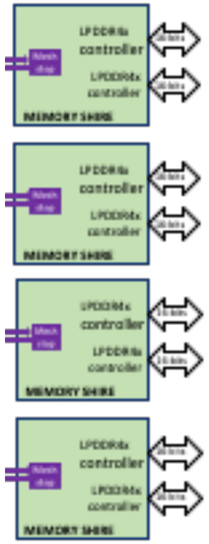


General Purpose CPU Chiplet



DRAM Controller Chiplet



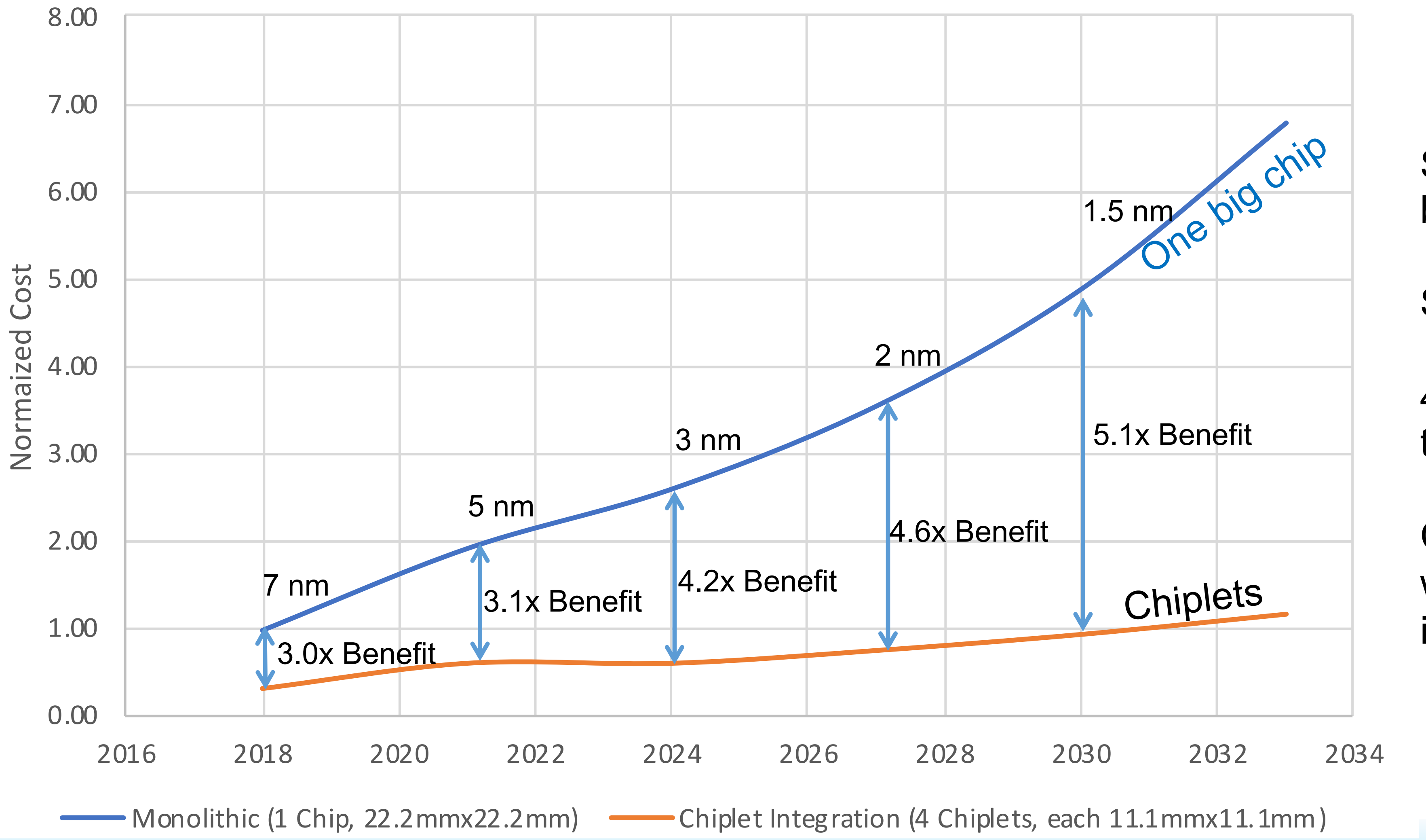


DRAM Controller Chiplet

ET-Minion Core Massively Parallel Compute Chiplets

Cost Benefit Using Chiplets Gets More Compelling Every Foundry Node

Example: Relative cost of ONE monolithic 500 mm^2 die vs implementing as FOUR 125mm^2 chiplets



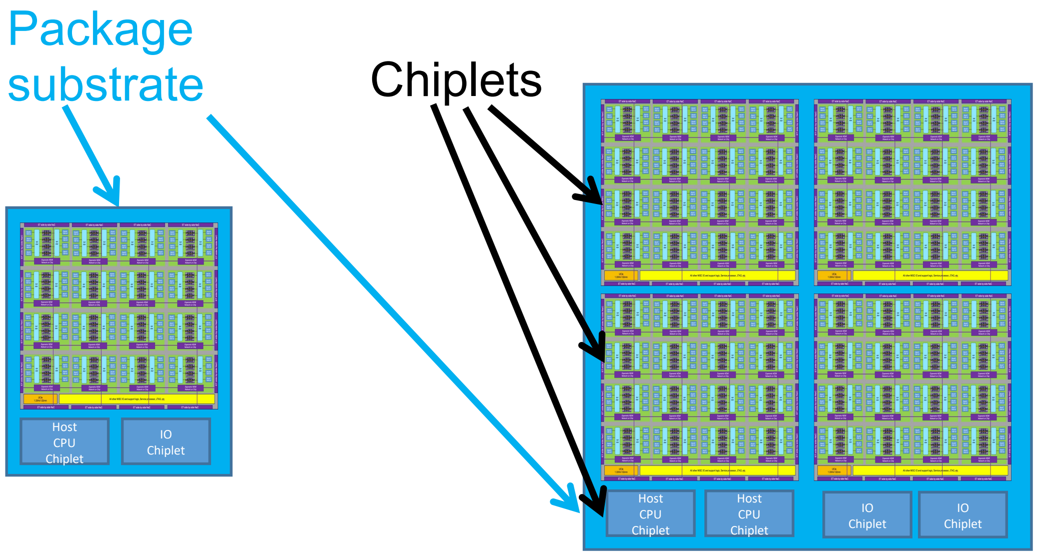
Same design,
but on 4 chips vs 1 chip

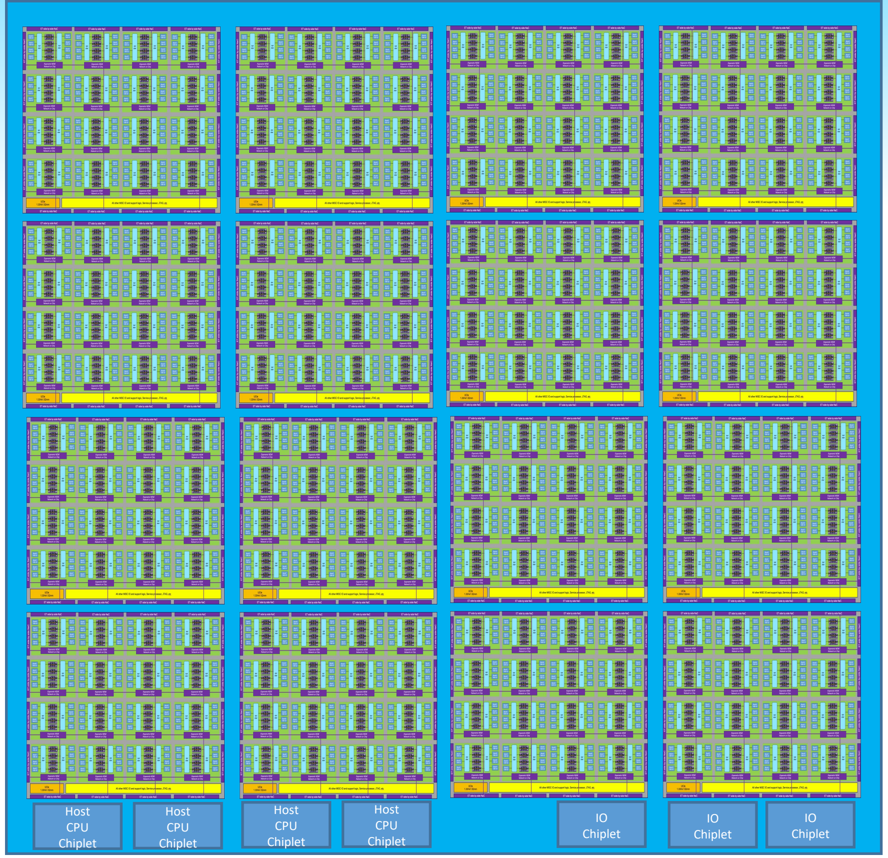
Same total chip area

4 chips includes extra
testing and assembly cost

Chiplet gains could be more
when some pieces done
in older less expensive nodes.

What might chiplets in a package look like in 2030?
4x today in a future node?





Small

1 Host CPU chiplet
1 IO Chiplet
1 Compute Chiplet with:

- 4K RISC-V+vector cores
- 3D Memory

Medium

2 Host CPU chiplet
2 IO Chiplet
4 Compute Chiplets with:

- 16K RISC-V compute cores
- 3D Memory

Large

4 Host CPU chiplets
3 IO Chiplet
16 Compute Chiplets with:

- 64K RISC-V compute cores
- 3D Memory
- Optical IO

Prediction: Over 1,000,000 cores in a small server by 2030

Today in 7nm we have over 1,000 cores per chip

Moore's law is slowing down, but not dead

By 2030 expect to have over 4,000 cores in a chiplet

Expect to put 16 chiplets together in a single package, ie 64,000 cores in a package

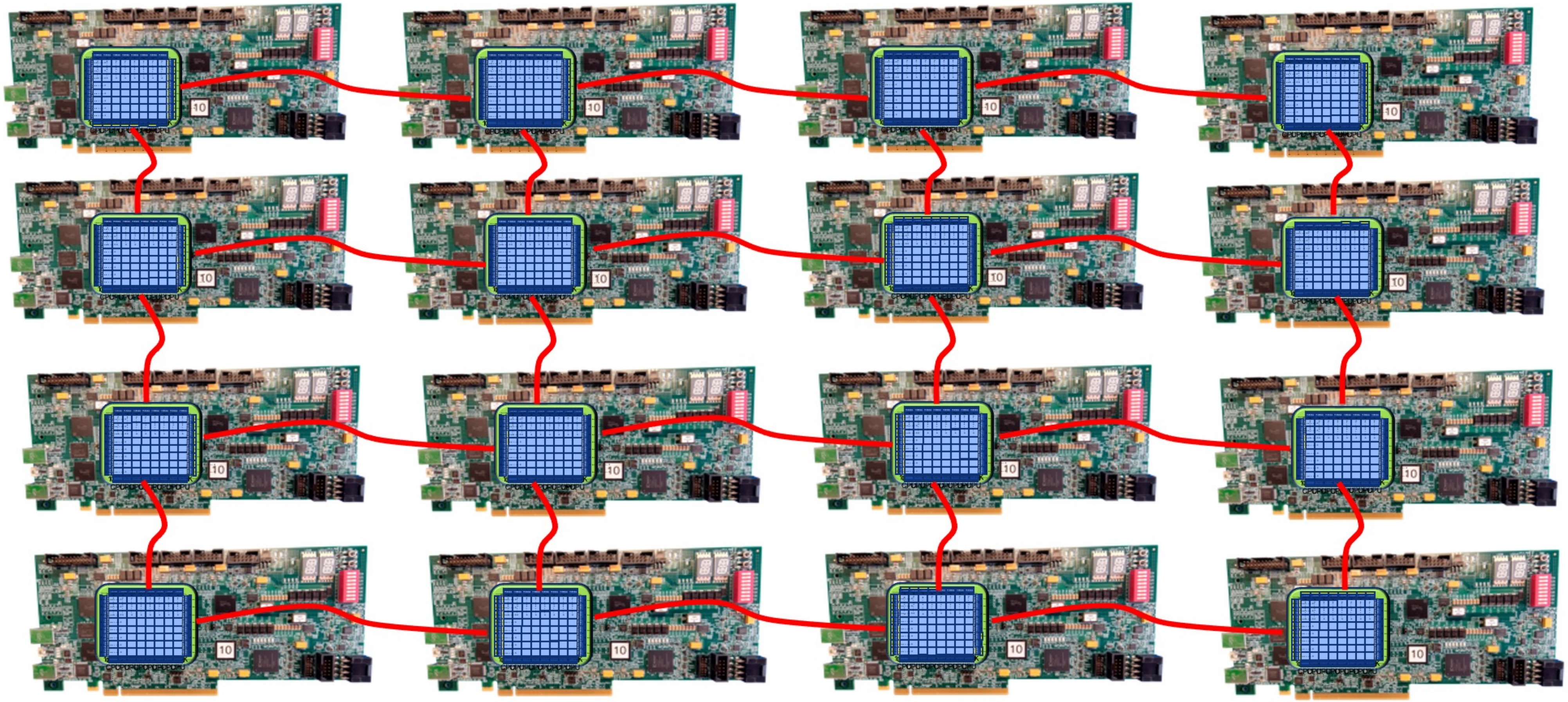
Put each package on a single accelerator board

Like we do today, put 16 of these boards in a small sever

That is over one million RISC-V cores, each with a vector unit, ready to run your workload

What will you do with over a million cores at your disposal. This is the next generation challenge.

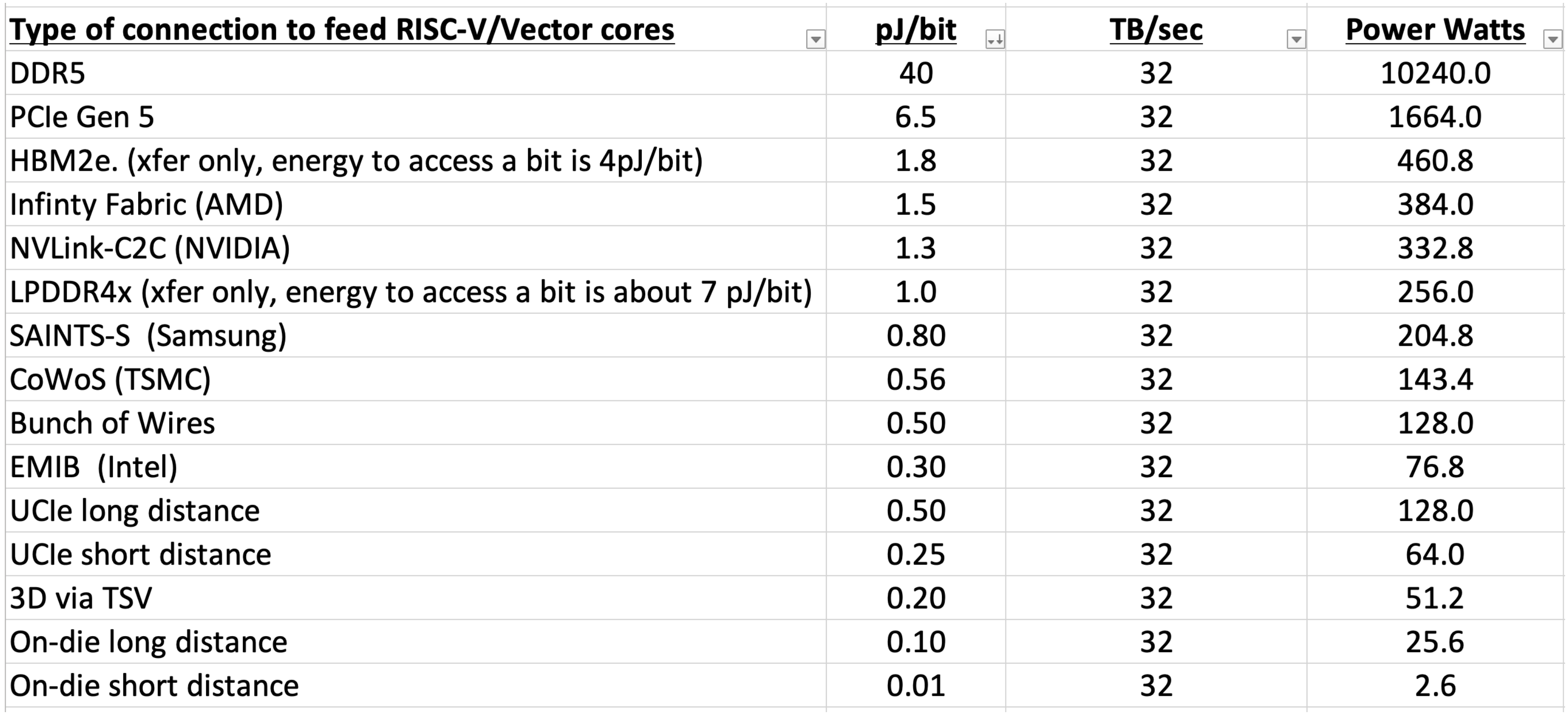
One million cores might fit on just a few cards. Challenge is using them efficiently.



Example: 1 Million cores on 16 boards, each board with one package with 64K cores.
If boards fit in similar size server as today, that’s 20 million RISC-V cores in a rack.

Energy cost to move data is the key challenge

Today’s ET-SoC-1 vector units can consume ~100TB/s, caches feed them at 8 TB/s.
If future chips are 4x as capable, how do we feed 4x, ie 32 TB/second as data set sizes grow?



Type of connection to feed RISC-V/Vector cores	pJ/bit	TB/sec	Power Watts
DDR5	40	32	10240.0
PCIe Gen 5	6.5	32	1664.0
HBM2e. (xfer only, energy to access a bit is 4pJ/bit)	1.8	32	460.8
Infinty Fabric (AMD)	1.5	32	384.0
NVLink-C2C (NVIDIA)	1.3	32	332.8
LPDDR4x (xfer only, energy to access a bit is about 7 pJ/bit)	1.0	32	256.0
SAINTS-S (Samsung)	0.80	32	204.8
CoWoS (TSMC)	0.56	32	143.4
Bunch of Wires	0.50	32	128.0
EMIB (Intel)	0.30	32	76.8
UCIe long distance	0.50	32	128.0
UCIe short distance	0.25	32	64.0
3D via TSV	0.20	32	51.2
On-die long distance	0.10	32	25.6
On-die short distance	0.01	32	2.6

Data movement power may be far greater than compute power.
Seems like stacking memory in 3D on top of compute chiplets
may be a way to reduce excessive chip to chip data movement power.

Summary

RISC vs CISC is no longer a popular debate, RISC has won

- RISC techniques are ubiquitous in most types of processors
- RISC-V is likely to be the common instruction set for much new RISC innovation
- RISC-V doesn’t carry years of baggage, providing advantages in power, area, cost
- RISC-V started with simple embedded applications
- Will soon see 8+ issue RISC-V out-of-order cores rivaling the best of other ISA implementations
- For massively parallel compute, energy efficiency is key, and slower in-order with vector wins

Our future is likely to continue to use RISC techniques:

- Easy to program general-purpose RISC-V likely to win over specialized ISA & architectures
- Next big challenge likely to be methods to program millions of RISC-V cores efficiently
- Big opportunities in CPU micro-architecture to design more efficient clusters of RISC-V cores

Thanks!

End of Presentation

Questions?