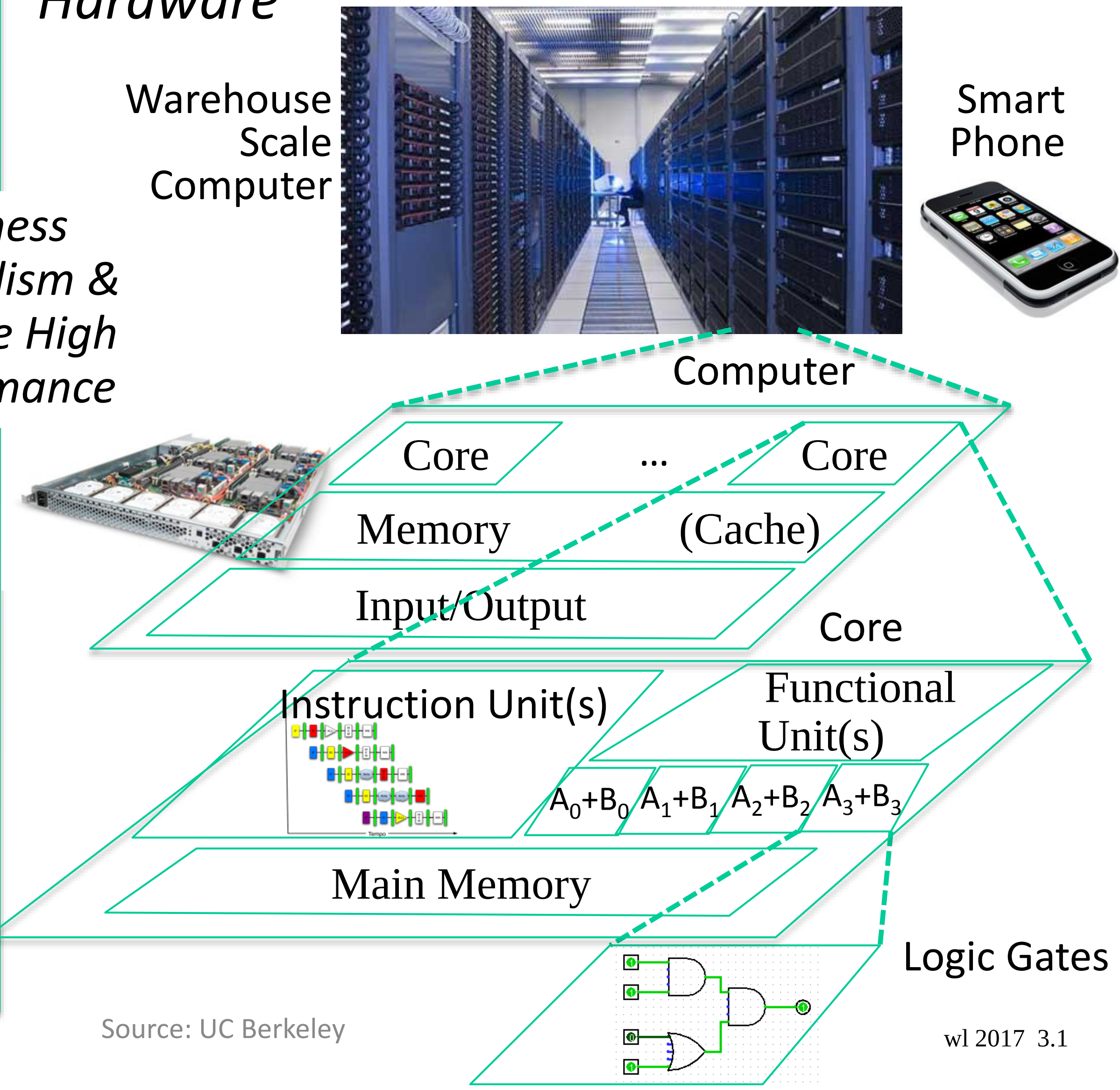
Hardware/software: levels of abstraction

Software | *Hardware*

- Parallel Requests
  Assigned to computer
  e.g. Search “Imperial”
- Parallel Threads
  Assigned to core
  e.g. Lookup, Ads
- Parallel Instructions
  >1 instruction @ one time
  e.g. 5 pipelined instructions
- Parallel Data
  >1 data item @ one time
  e.g. Add of 4 pairs of words
- Hardware descriptions
  All gates functioning in
  parallel at same time

Harness Parallelism & Achieve High Performance



Source: UC Berkeley

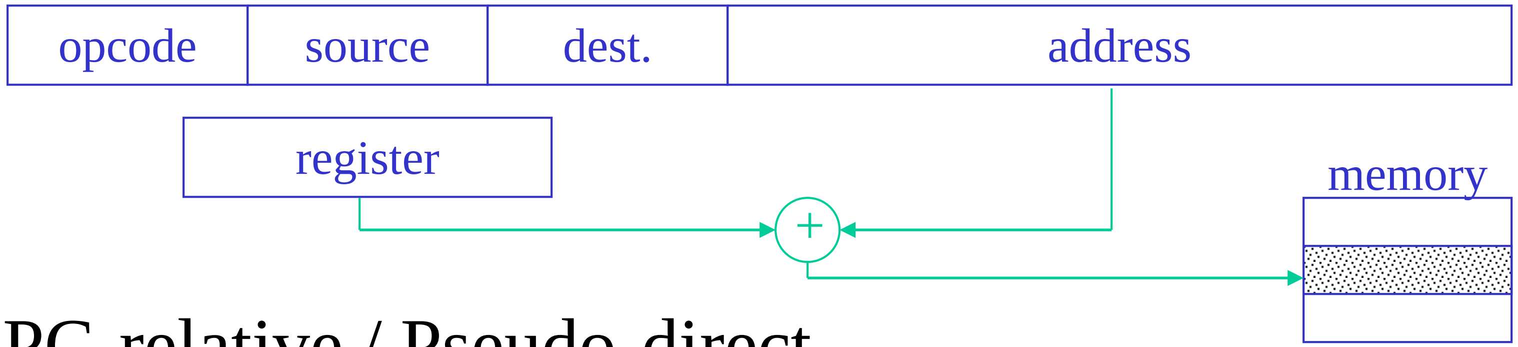
MIPS instruction format

	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
R	opcode	source 1	source 2	dest.	shift amt	fn code
I	opcode	source 1	**source 2 / dest.**	address / data		
J	opcode	address				

- R: register-based operations: arithmetic, compare
- I : immediate data/address: load/store, branch
- J : jumps: involving memory

MIPS addressing modes

- register addressing: data in registers
- immediate addressing: data in instruction itself (I-type)
- base addressing: data in memory: load/store instructions



- PC-relative / Pseudo-direct

MIPS and 68000

- MIPS is typical RISC
  - SPARC similar but more complex
- 68000 is typical CISC
  - VAX: more complex but regular
  - x86/Pentium *N*: more complex and irregular!
- differences: reflect technology advances
  - 68000: less registers, less general
- differences: reflect different views
  - CISC: reduce “semantic gap”

Compare 68000 and MIPS

- 8 data registers, 8 address registers
- 12 addressing modes   data reg dir, addr reg dir/indir
- limited number of arithmetic instructions operate directly on address registers
- speed: benchmark SPECint92: 21 (4.2 times slower)
  68040 SPECfp92 : 15 (6.5 times slower)
  (than MIPS R4400)
- cost: $233 (4.7 times cheaper than R4400)
- cost effectiveness?

Addressing comparison

- 68000 has auto-increment mode

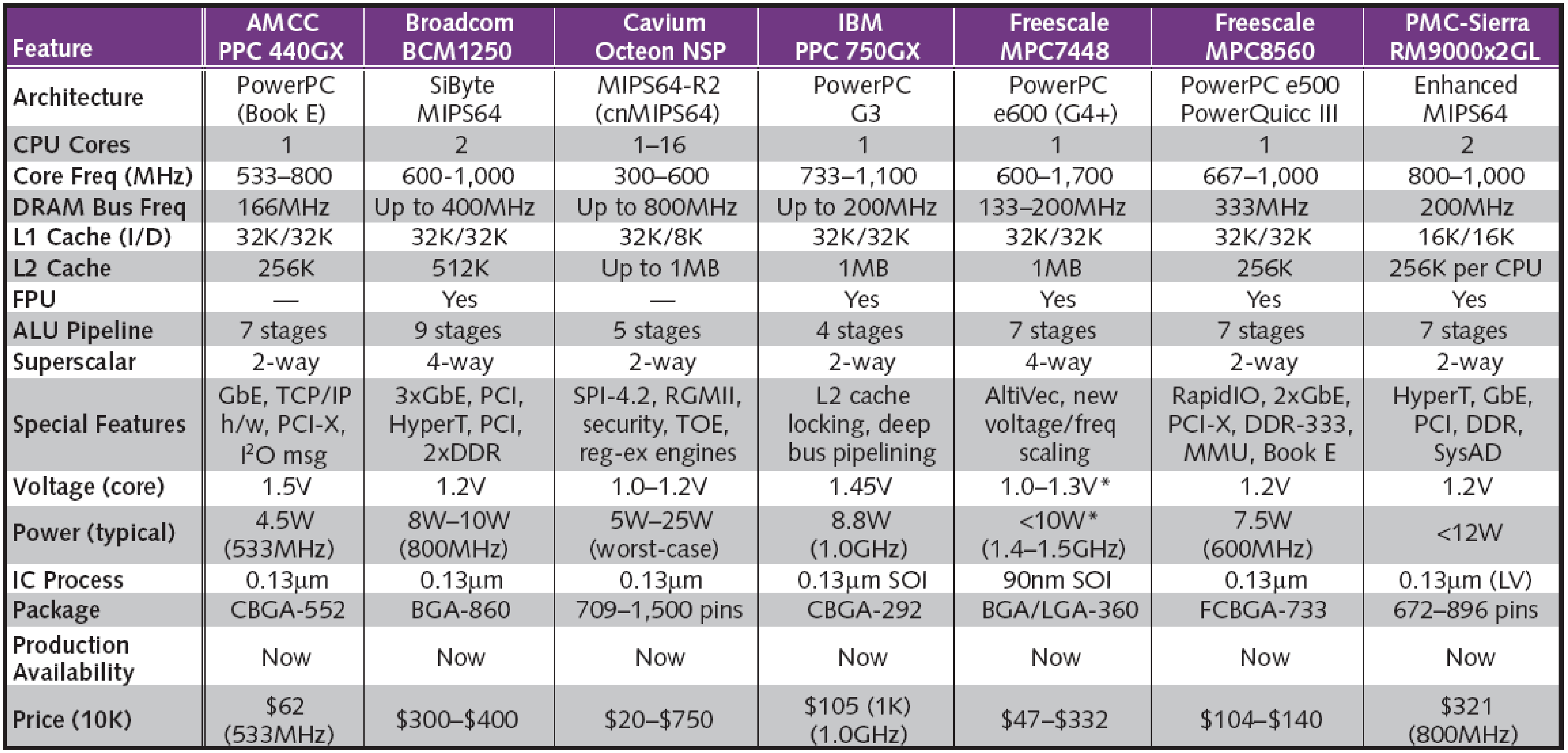
```
    (a0)+                        M[a0]  before  a0 = a0 + 4
so   add.l  d0  (a0)+            d0 = d0 + M[a0], a0 = a0+4
takes 16 bits
```

- MIPS require 3 instructions

```
lw   $9,   0, ($7)    # reg9 = M[reg7]
add $7, $7,   4       # reg7 = reg7 + 4
add $8, $8, $9        # reg8 = reg8 + reg9
```

- … and how many bits?

Embedded processors comparison



Feature	AMCC PPC 440GX	Broadcom BCM1250	Cavium Octeon NSP	IBM PPC 750GX	Freescale MPC7448	Freescale MPC8560	PMC-Sierra RM9000x2GL
Architecture	PowerPC (Book E)	SiByte MIPS64	MIPS64-R2 (cnMIPS64)	PowerPC G3	PowerPC e600 (G4+)	PowerPC e500 PowerQuicc III	Enhanced MIPS64
CPU Cores	1	2	1–16	1	1	1	2
Core Freq (MHz)	533–800	600-1,000	300–600	733–1,100	600–1,700	667–1,000	800–1,000
DRAM Bus Freq	166MHz	Up to 400MHz	Up to 800MHz	Up to 200MHz	133–200MHz	333MHz	200MHz
L1 Cache (I/D)	32K/32K	32K/32K	32K/8K	32K/32K	32K/32K	32K/32K	16K/16K
L2 Cache	256K	512K	Up to 1MB	1MB	1MB	256K	256K per CPU
FPU	—	Yes	—	Yes	Yes	Yes	Yes
ALU Pipeline	7 stages	9 stages	5 stages	4 stages	7 stages	7 stages	7 stages
Superscalar	2-way	4-way	2-way	2-way	4-way	2-way	2-way
Special Features	GbE, TCP/IP h/w, PCI-X, I^{2}O msg	3xGbE, PCI, HyperT, PCI, 2xDDR	SPI-4.2, RGMII, security, TOE, reg-ex engines	L2 cache locking, deep bus pipelining	AltiVec, new voltage/freq scaling	RapidIO, 2xGbE, PCI-X, DDR-333, MMU, Book E	HyperT, GbE, PCI, DDR, SysAD
Voltage (core)	1.5V	1.2V	1.0–1.2V	1.45V	1.0–1.3V*	1.2V	1.2V
Power (typical)	4.5W (533MHz)	8W–10W (800MHz)	5W–25W (worst-case)	8.8W (1.0GHz)	<10W* (1.4–1.5GHz)	7.5W (600MHz)	<12W
IC Process	0.13µm	0.13µm	0.13µm	0.13µm SOI	90nm SOI	0.13µm	0.13µm (LV)
Package	CBGA-552	BGA-860	709–1,500 pins	CBGA-292	BGA/LGA-360	FCBGA-733	672–896 pins
Production Availability	Now	Now	Now	Now	Now	Now	Now
Price (10K)	$62 (533MHz)	$300–$400	$20–$750	$105 (1K) (1.0GHz)	$47–$332	$104–$140	$321 (800MHz)

* vendor estimate

(source: Chart Watch, Microprocessor report)

Classifying architectures

- addressing temporary storage
- stack: operands specified implicitly at top of stack

  C = A+B $\rightarrow$ `push A; push B; add; pop C`
- accumulator: one operand in accumulator

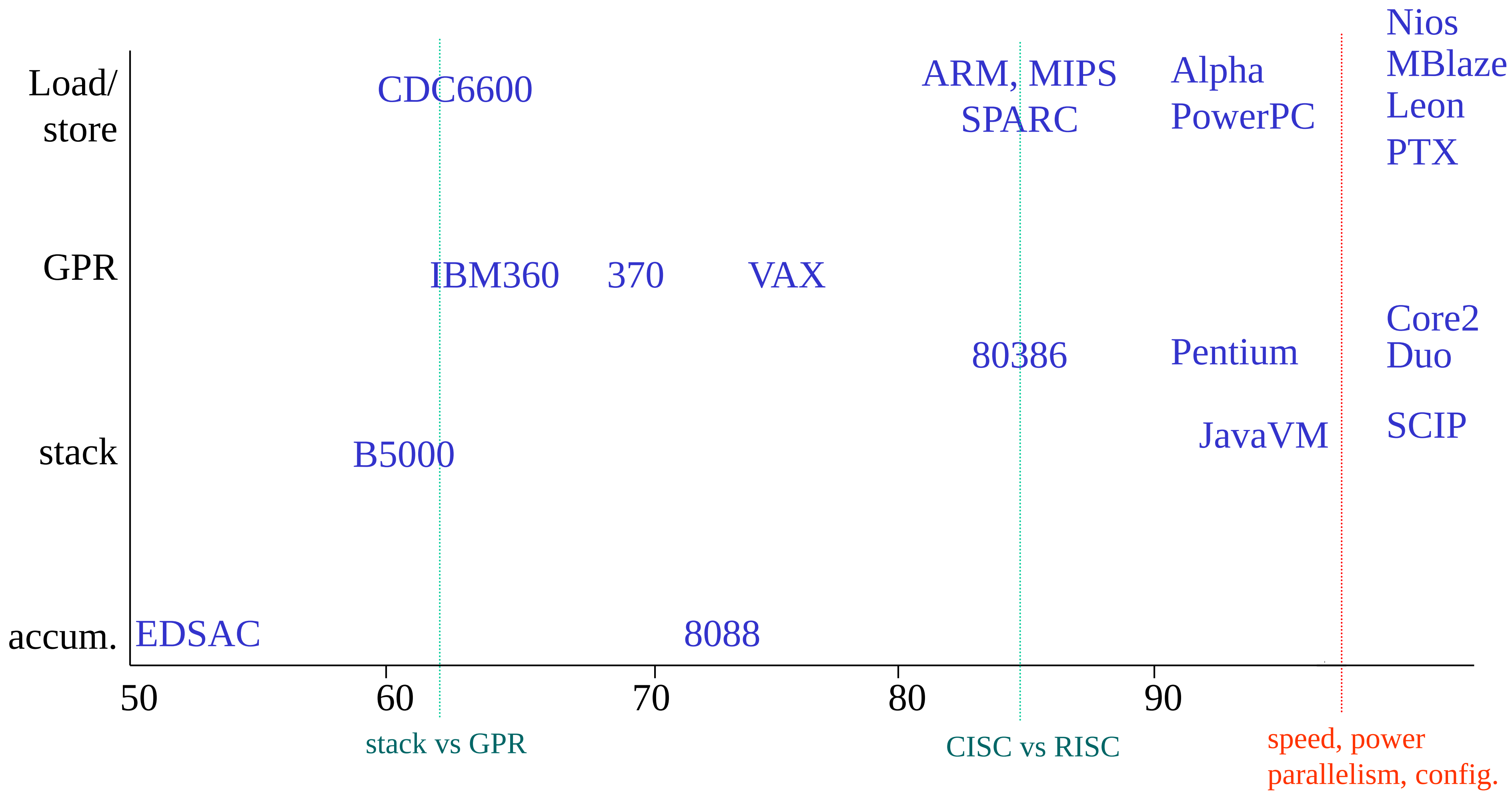
  C = A+B $\rightarrow$ `load A; add B; store C`
- register: explicit operands

  C = A+B $\rightarrow$ `load R1 A; add R2,R1,B; store C,R2`
- register advantages: faster than memory, reduce memory traffic, compiler friendly, improve code density

Comparing architectures

Examples	temporary storage	example	pros	cons
B5500 HP 3000/70	stack	add top pair on stack	simple eval model; dense code	less flexible: no random access; slow if stack in memory
PDP 8 M6809	accumulator	add accum. and memory	min. internal store; short instr.	freq. memory access: slow
VAX MIPS	registers or memory	add 2 registers	general model for code gen., fast reg. access	name all operands; long instr.

The Great Instruction Set Debates

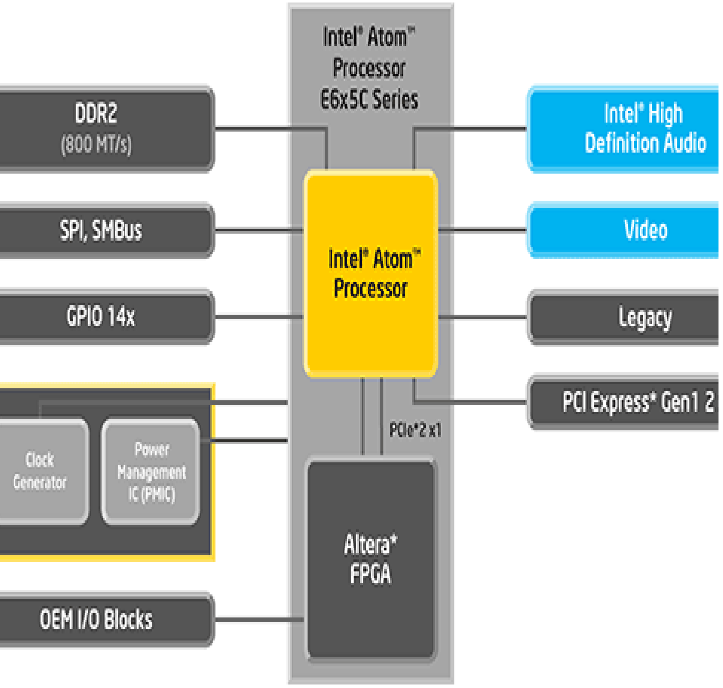


criteria: instruction count? code size? performance? parallelism: fixed/customisable? power/energy? evolve with changing requirements, application and environment?

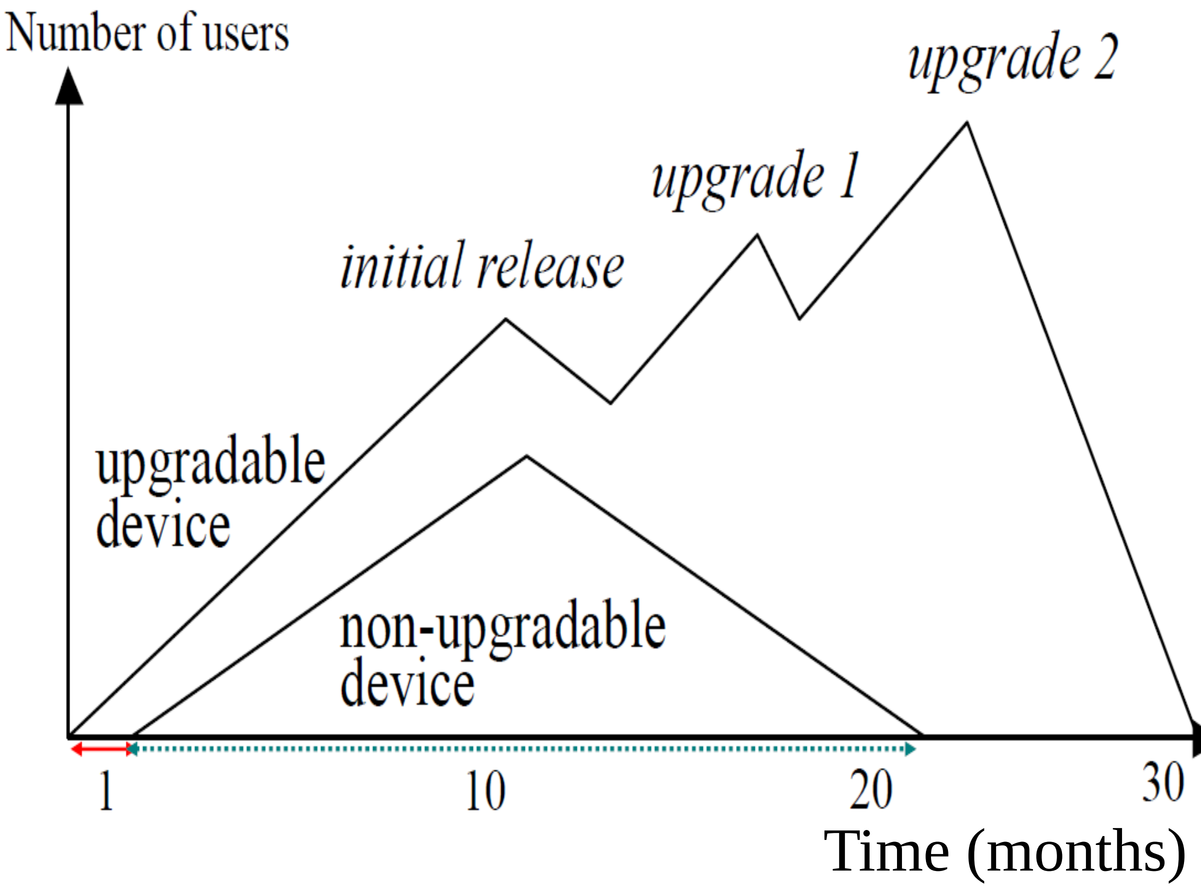
PTX: nVidia Parallel Thread Execution

SCIP: Scalable Configurable Instrument Processor

Current/future: upgradable hardware!



Intel Atom Processor E6x5C Series



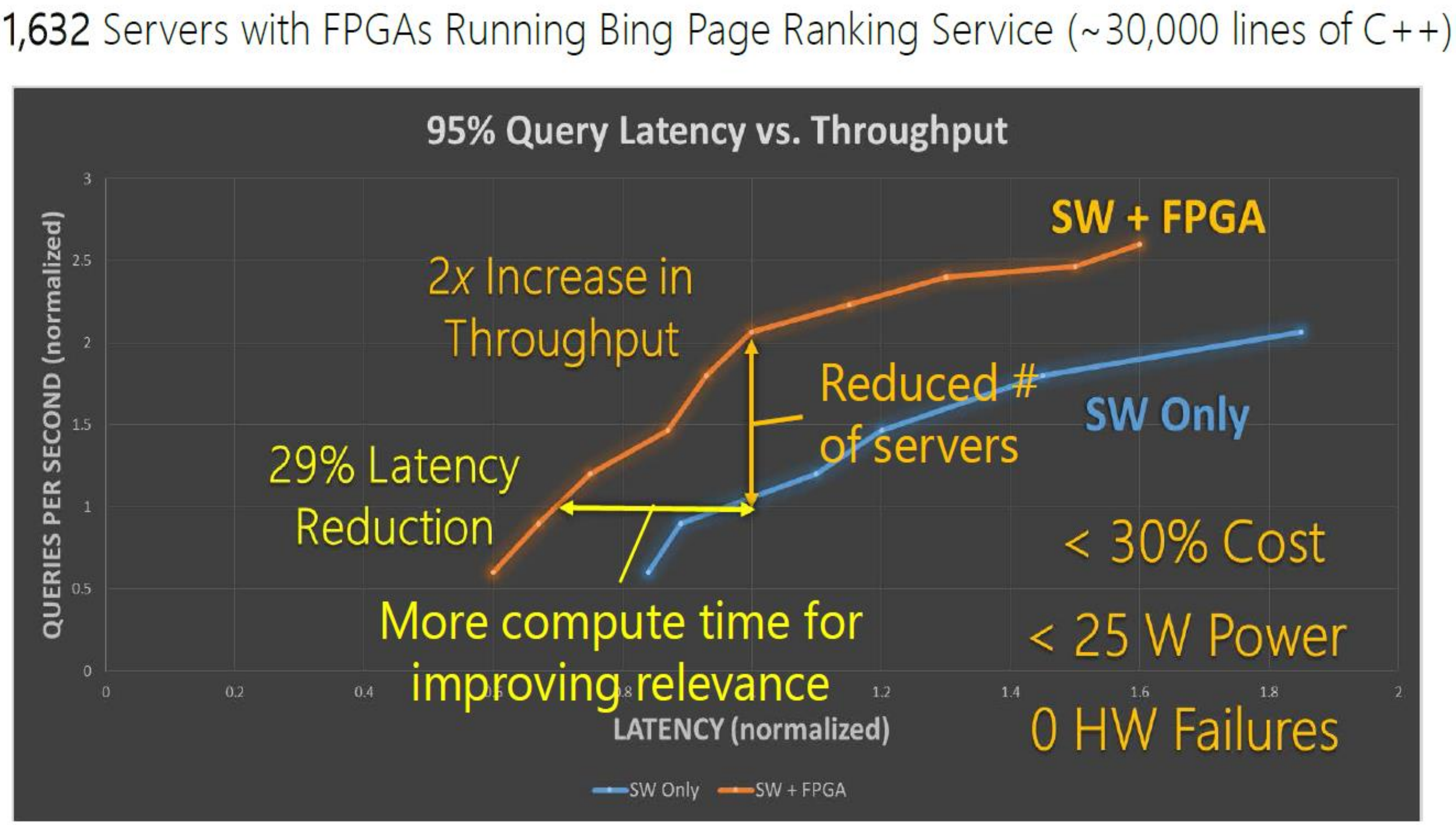
upgradability: minimise time-to-market ⟷

maximise time-in-market ⟷

upgrade frequently? Run Time Reconfiguration

Source: Algotronix Consulting / Xilinx / Intel

Bottleneck example: Bing page ranking



source: Microsoft