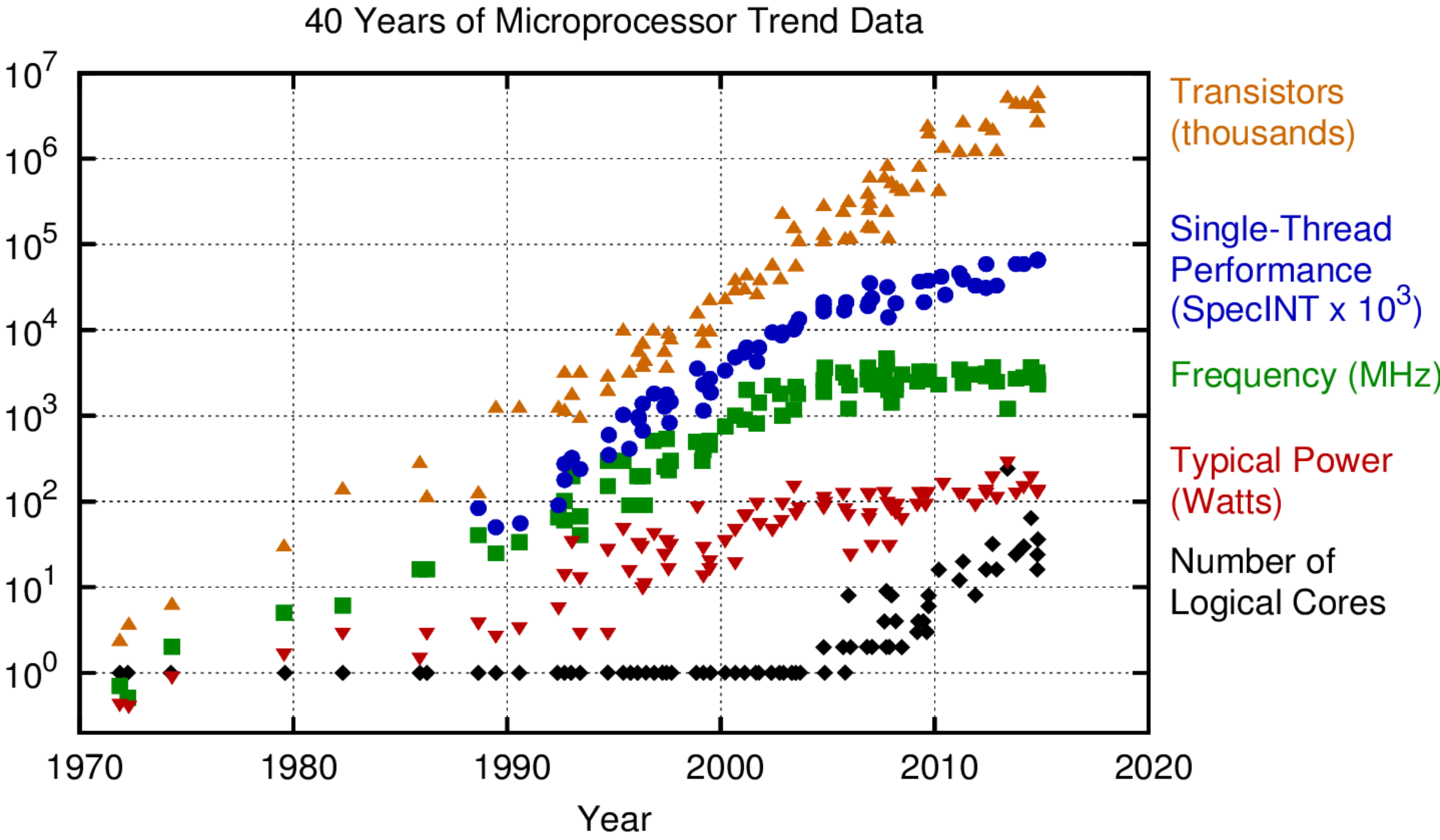
Introduction to Heterogenous Computing Systems

Henk Corporaal

www.ics.ele.tue.nl/~heco

ASCI Spring School

Soesterberg, May 29 - June 1, 2017

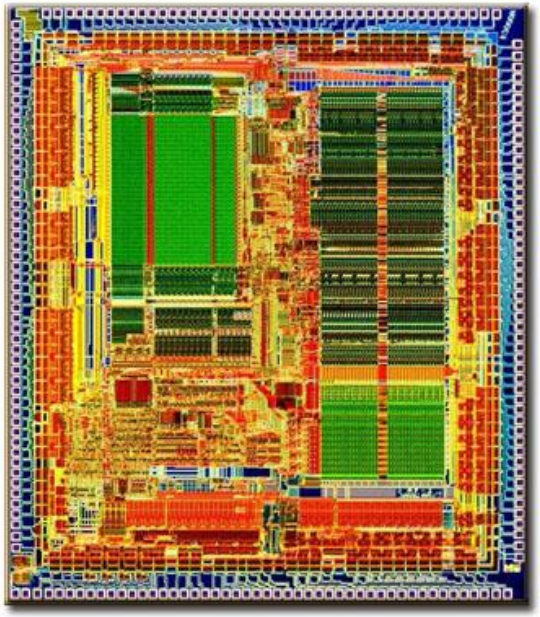


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

Overview

- Processor basics
  - RISC, pipelining, etc.
- Energy optimizations
  - inefficiencies
- Going parallel
  - the 4-D model
- Going heterogeneous: why?
- How far are we?
  - some examples
- Conclusions
  - Research topics

Processor basics: How to build a RISC



MIPS_3000

RISC characteristics:

- Reduced number of instructions
- Limited addressing modes
  - load-store architecture
  - enables pipelining
- Large, uniform register set
- One instruction size (32 bits)
  - know directly where the following instruction starts
- Limited number of instruction formats
- Memory alignment restrictions
- ..... keep it simple ....
- Based on quantitative analysis
  - " the famous MIPS one percent rule": don't even think about it when its not used more than one percent

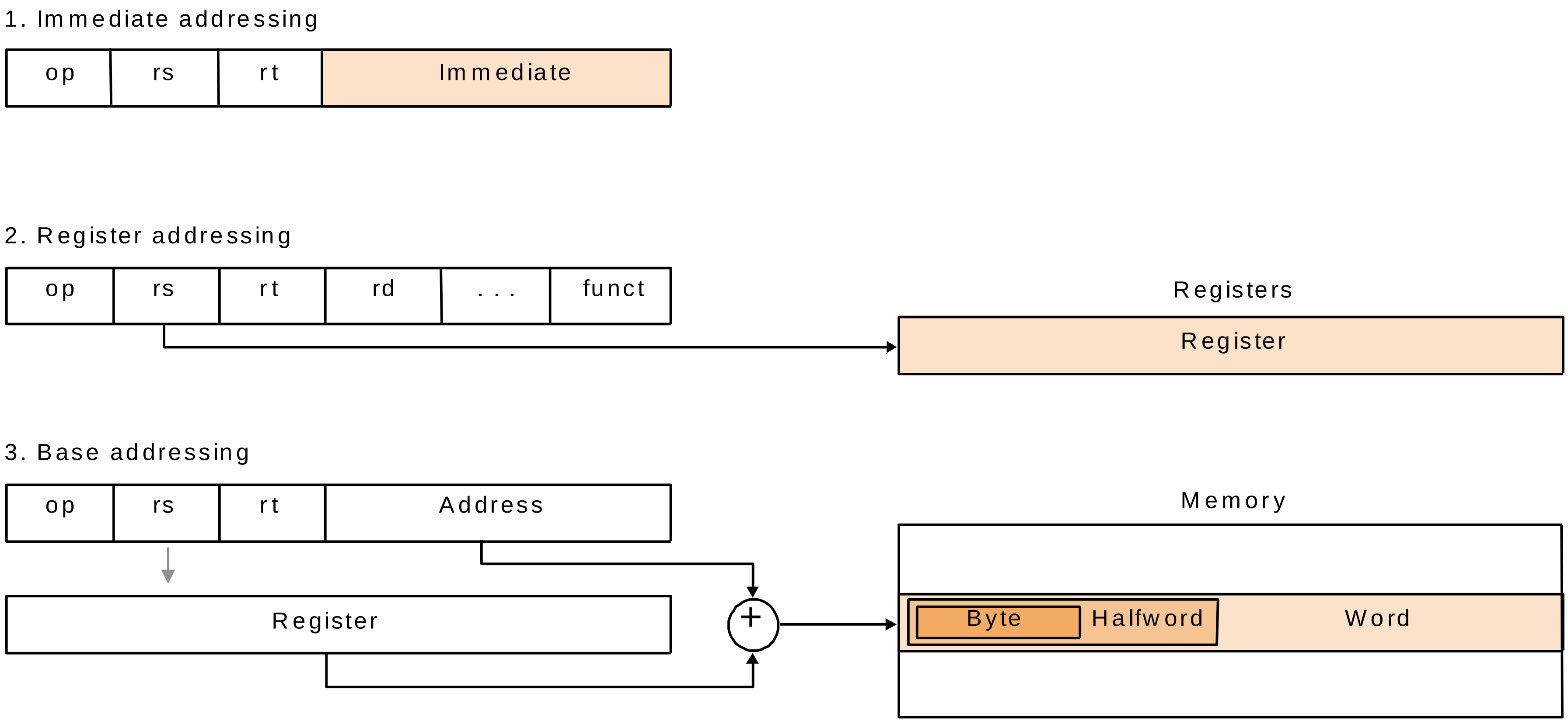
Instruction types: only 3 classes !!

- Arithmetic
  - Integer arithmetic/logic instructions
    - ADD, SUB, MULT, ADDU, ...
    - OR, AND, NOR, NAND, ...
  - Floating point instructions
    - FADD, FMUL, FDIV
- Memory transfer
  - Loads and Stores
  - TEST-and-SET, and SWAP
  - various operand sizes: bytes, half-words, words, doubles, etc.
- Control instructions
  - Branches, Jumps, Returns, Exceptions

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1, $s2, $s3	$s1 = $s2 + $s3	Three operands; data in registers
	subtract	sub $s1, $s2, $s3	$s1 = $s2 - $s3	Three operands; data in registers
	add immediate	addi $s1, $s2, 100	$s1 = $s2 + 100	Used to add constants
Data transfer	load word	lw $s1, 100($s2)	$s1 = Memory[$s2 + 100	Word from memory to register
	store word	sw $s1, 100($s2)	Memory[$s2 + 100] = $s1	Word from register to memory
	load byte	lb $s1, 100($s2)	$s1 = Memory[$s2 + 100	Byte from memory to register
	store byte	sb $s1, 100($s2)	Memory[$s2 + 100] = $s1	Byte from register to memory
	load upper immediate	lui $s1, 100	$\$s1 = 100 * 2^{16}$	Loads constant in upper 16 bits
Conditional branch	branch on equal	beq $s1, $s2, 25	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne $s1, $s2, 25	if ($s1 != $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt $s1, $s2, $s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; for beq, bne
	set less than immediate	slti $s1, $s2, 100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare less than constant
Uncondi-tional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr $ra	go to $ra	For switch, procedure return
	jump and link	jal 2500	$ra = PC + 4; go to 10000	For procedure call

MIPS: 3 addressing modes

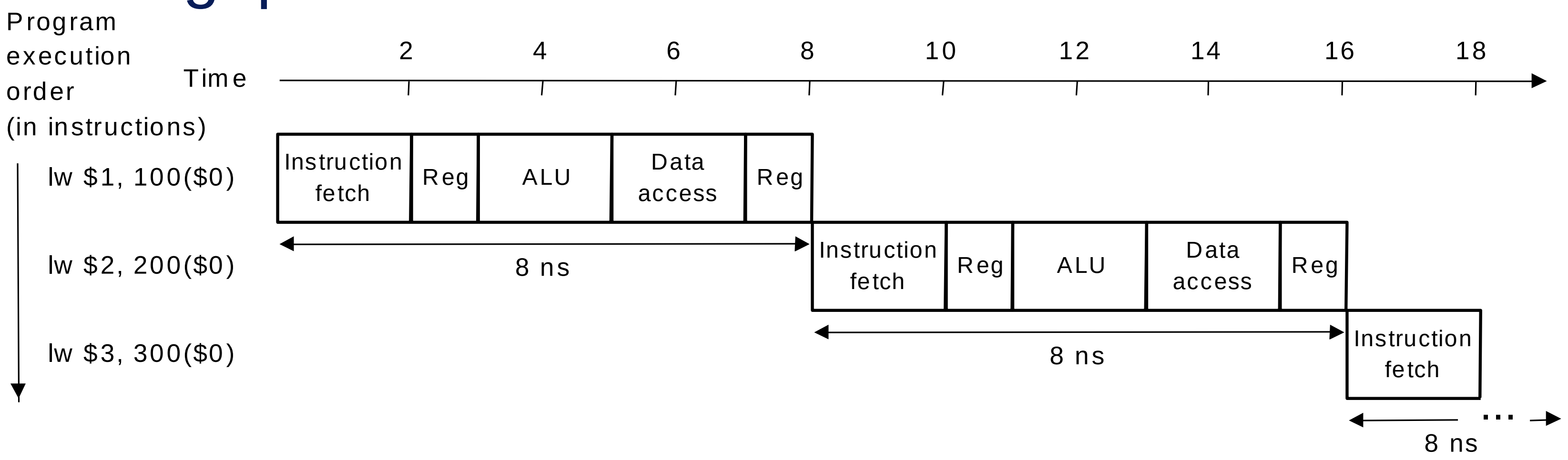


Pipelining

- Why: Ideal speedup = number of stages
- Do we achieve this? NO! HAZARDS
- Let’s look at Pipeline implementation (= processor organization)

Pipelining

Improve performance by increasing instruction throughput



Program execution order (in instructions)

Time

2 4 6 8 10 12 14 16 18

lw $1, 100($0)

Instruction fetch | Reg | ALU | Data access | Reg

lw $2, 200($0)

8 ns

Instruction fetch | Reg | ALU | Data access | Reg

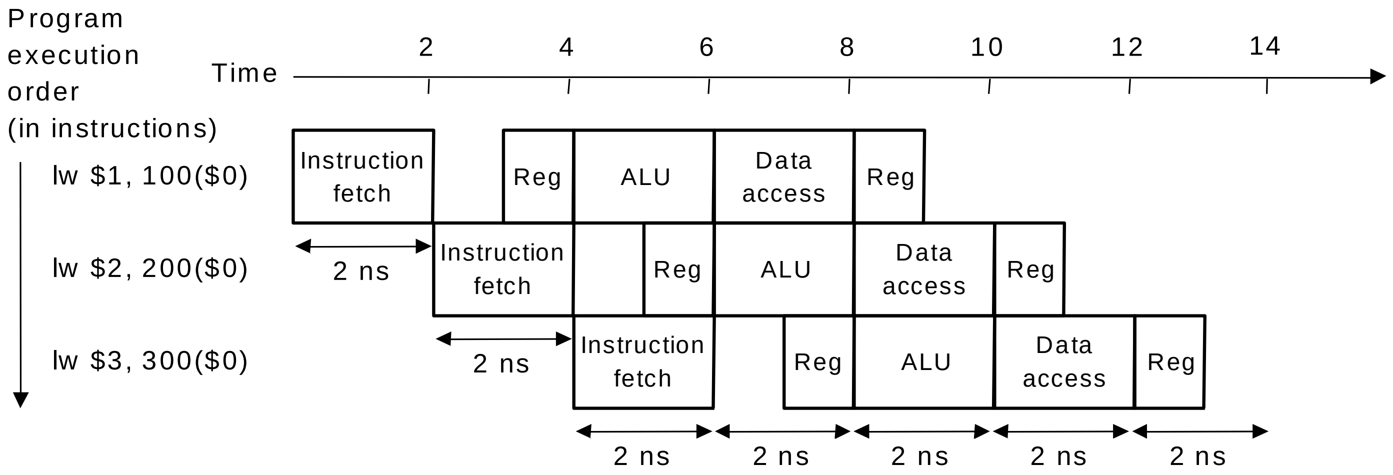
lw $3, 300($0)

8 ns

Instruction fetch

...

8 ns



Program execution order (in instructions)

Time

2 4 6 8 10 12 14

lw $1, 100($0)

Instruction fetch | Reg | ALU | Data access | Reg

lw $2, 200($0)

2 ns

Instruction fetch | Reg | ALU | Data access | Reg

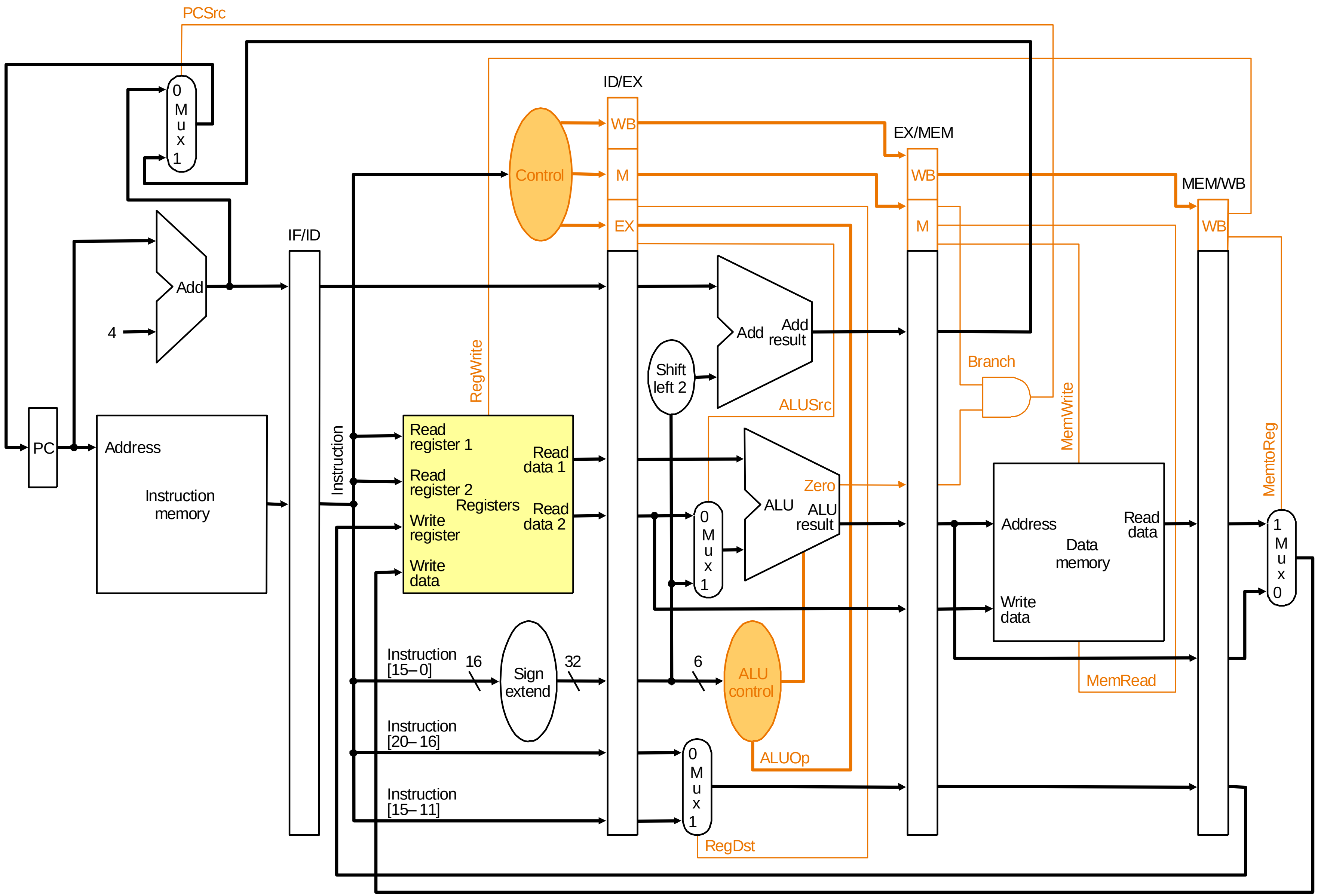
lw $3, 300($0)

2 ns

Instruction fetch | Reg | ALU | Data access | Reg

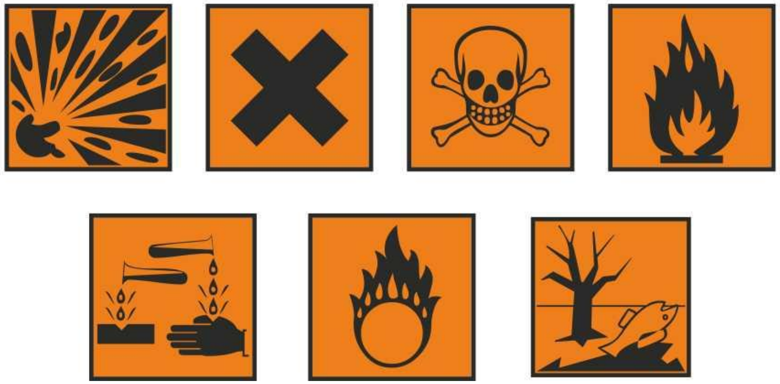
2 ns 2 ns 2 ns 2 ns 2 ns

Datapath with Control

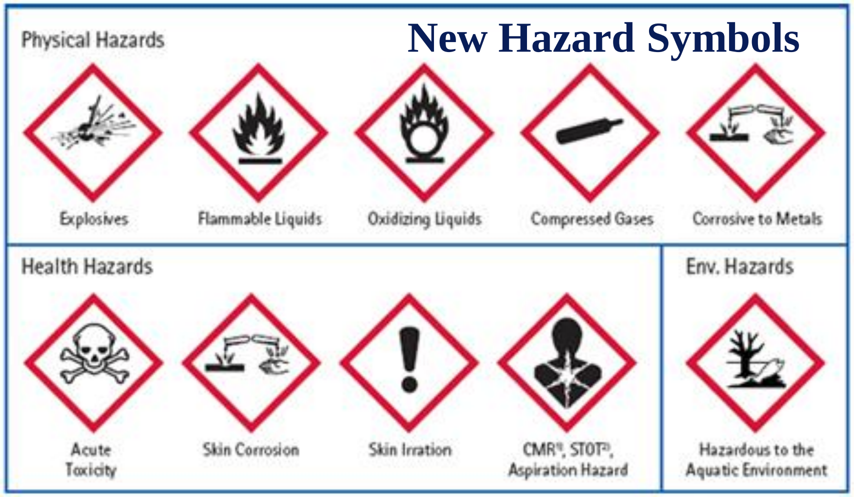


Hazards

Current CHIP Hazard Symbols valid until 1st June 2015



New Hazard Symbols



Hazards: problems due to pipelining

What is CPI (average # cycles / instruction) ?

Hazard types:

- Structural
  - same resource is needed multiple times in the same cycle
- Data
  - data dependencies limit pipelining
- Control
  - next executed instruction may not be the next specified instruction

3 Data dependences types:

RaW, WaR, WaW

Examples:

```
add r1, r2, 5      ; r1 := r2+5
sub r4, r1, r3     ; RaW of r1

add r1, r2, 5
sub r2, r4, 1      ; WaR of r2

add r1, r2, 5
sub r1, r1, 1      ; WaW of r1

st  r1, 5(r2)      ; M[r2+5] := r1
ld  r5, 0(r4)      ; RaW if 5+r2 = 0+r4
```

Forwarding example

Time (in clock cycles) →

	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register $2 :	10	10	10	10	10/– 20	– 20	– 20	– 20	– 20
Value of EX/MEM :	X	X	X	– 20	X	X	X	X	X
Value of MEM/WB :	X	X	X	X	– 20	X	X	X	X

Program
execution order
(in instructions)

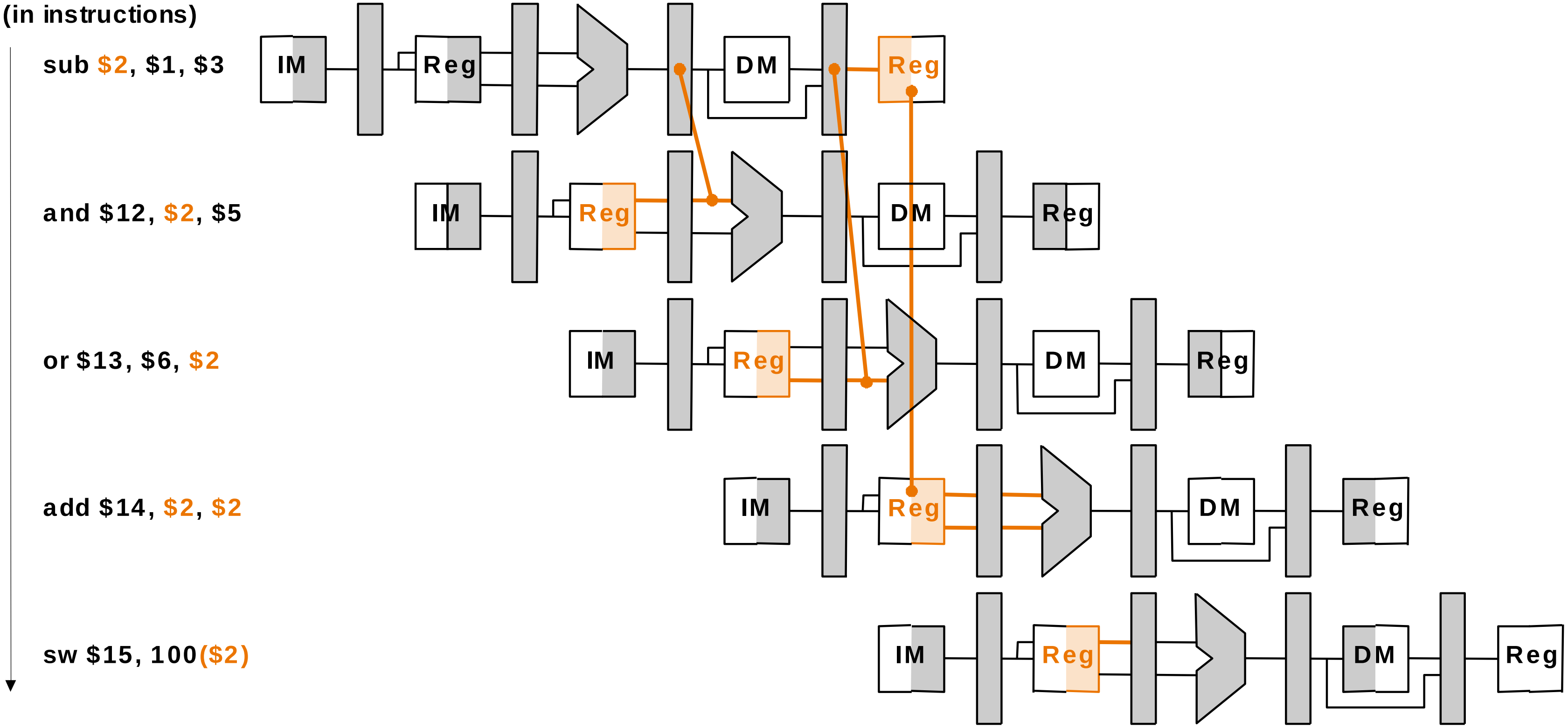
sub $2, $1, $3

and $12, $2, $5

or $13, $6, $2

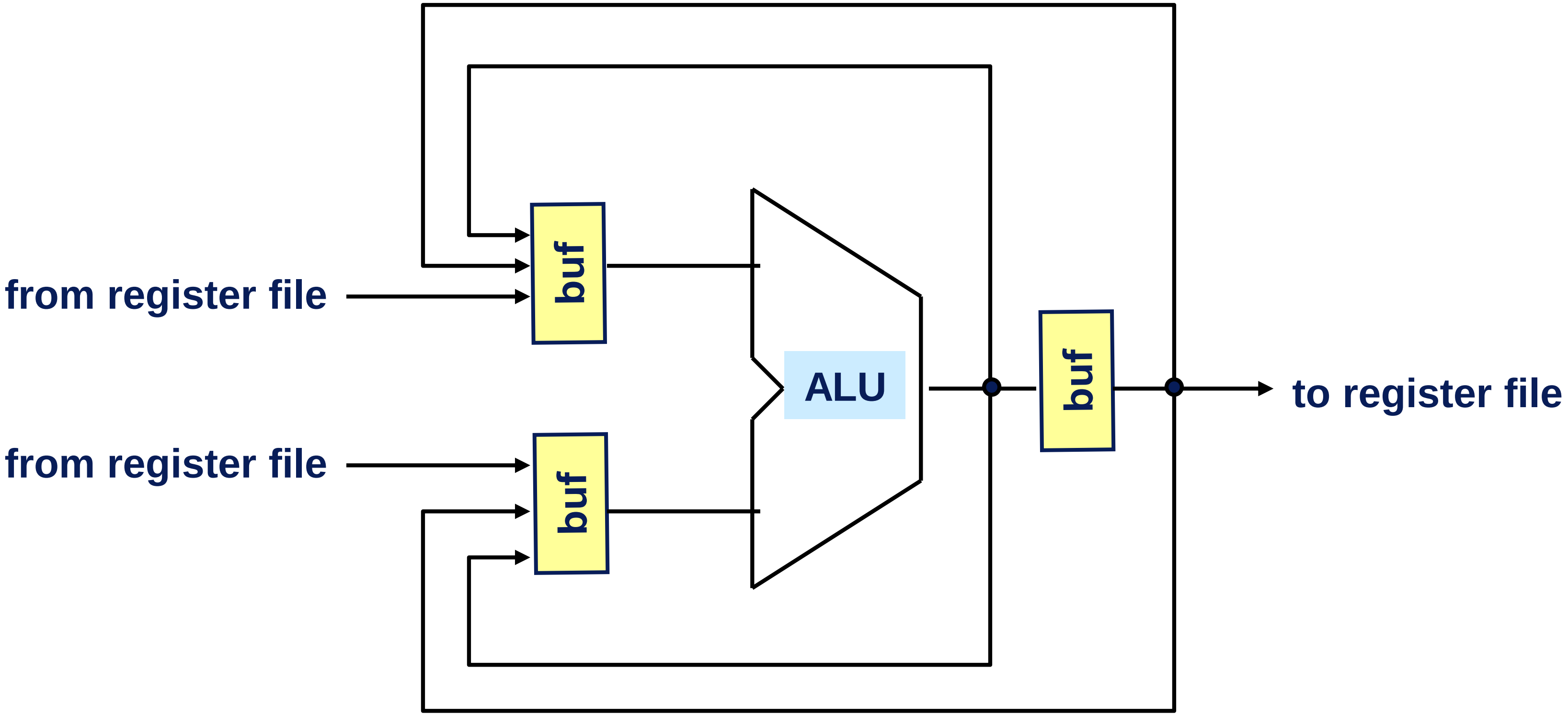
add $14, $2, $2

sw $15, 100($2)



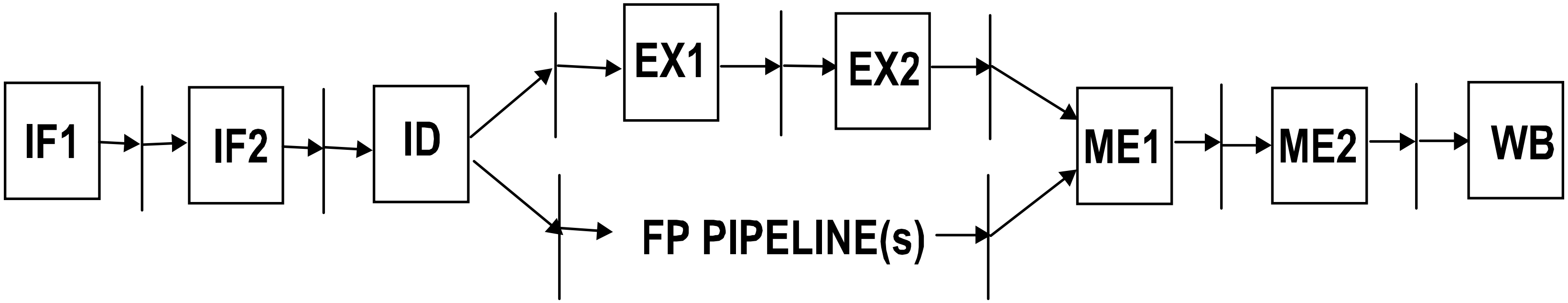
Use forwarding hardware

- Do not wait till the result is in the register file



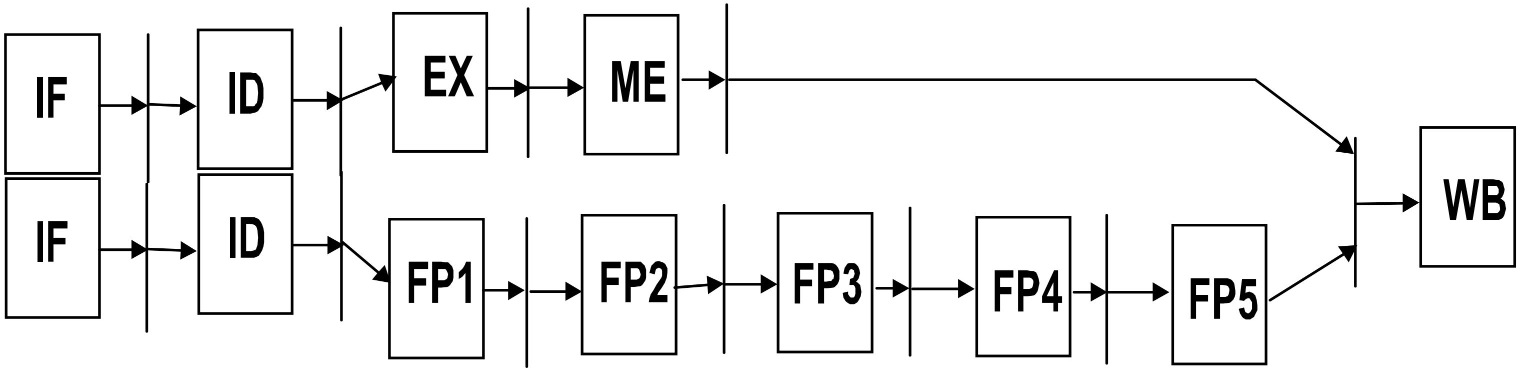
Note: not all Processors implement Forwarding: why / why not??

Superpipelined CPU



- Some stages can be further pipelined
  - To increase the clock rate
    - Clock is 2x as fast (hopefully)
  - Here if, EX, and Memory (ME) are now 2 pipeline stages
  - Not “free”
    - Branch penalty when taken is now 3 clocks
    - Latencies in clock are higher

Superscalar CPU: multi issue



- Fetch, decode and execute e.g. up to 2 instructions per clock
  - Same clock rate as basic pipeline
    - Issue a pair of instructions
      - Pair must be integer/branch/memory and FP pair (compiler)
      - Instructions in pair must be independent

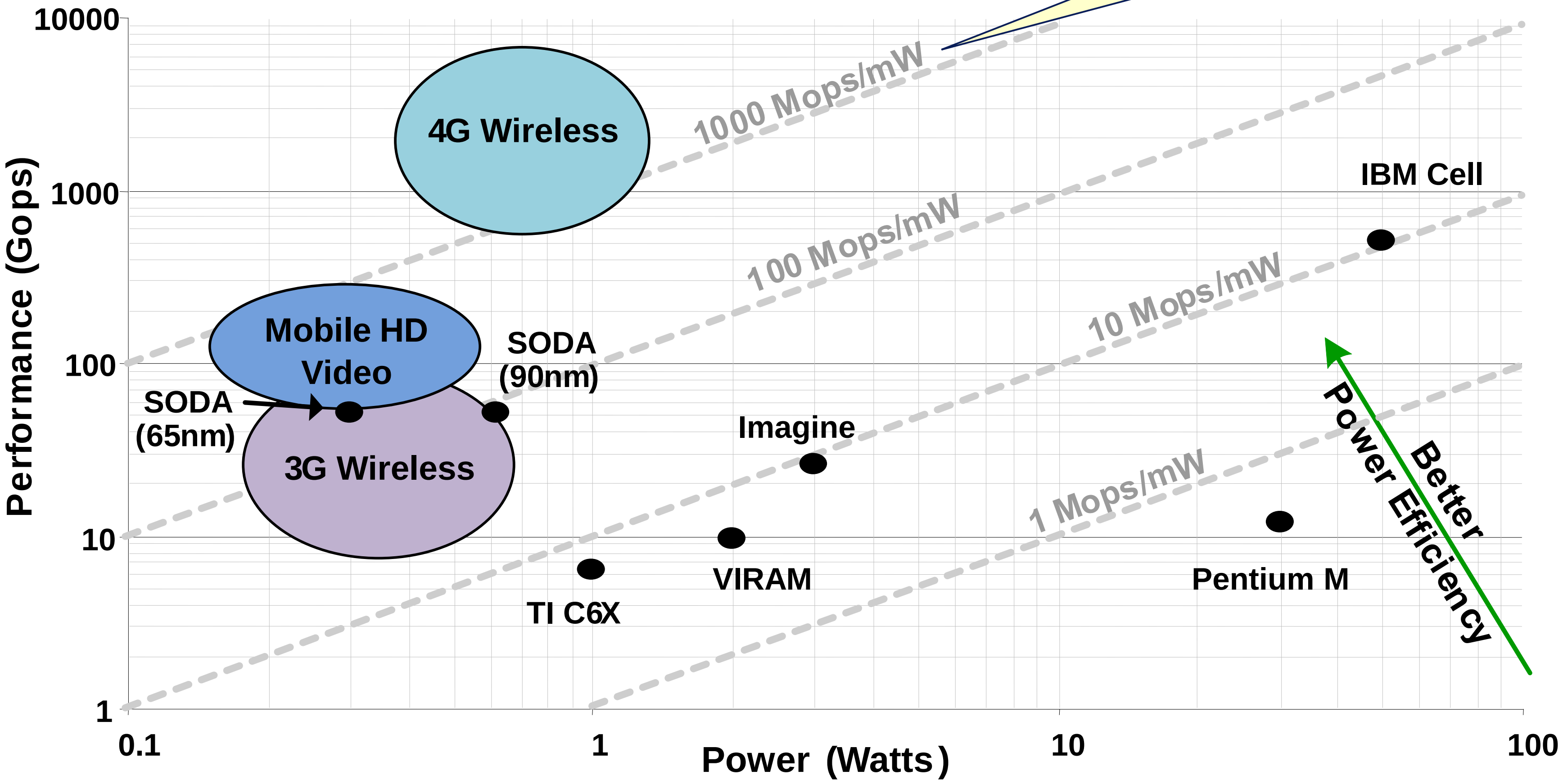
- Today: Typical upto 4-issue

Power versus Energy

- Power P = $\alpha f C V_{dd}^2$
  - α switching activity (<1); f frequency; C switching capacitance, V_{dd} supply voltage
  - heat / temperature constraint
  - wear-out
  - peak power delivery constraint
- Energy E = P*t or, for time varying P: $\int P(t).dt$
  - battery life
  - cost: electricity bill
- Note: lowering f reduces P, but not necessarily E; E may even increase due to leakage (static power dissipation)

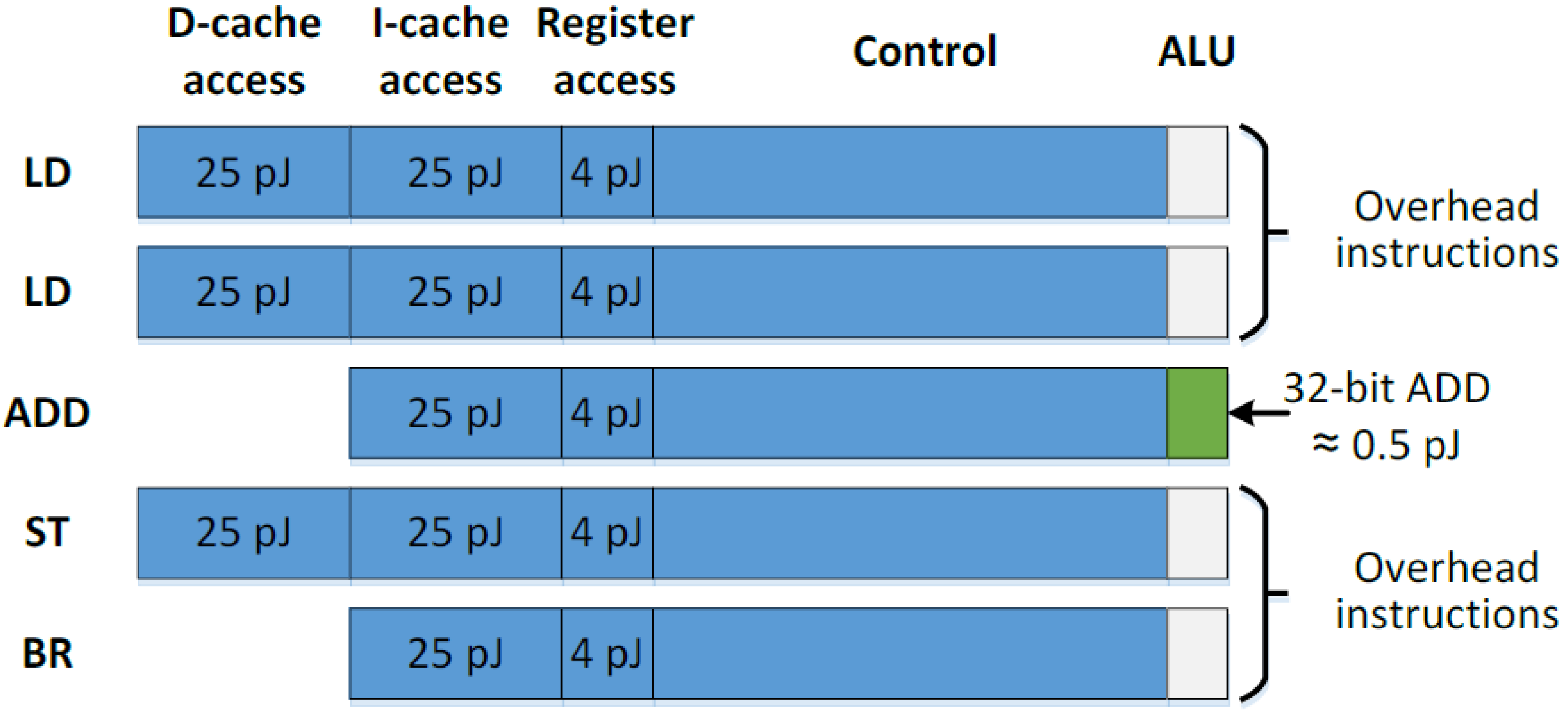
Computational efficiency (Mops/mW): what do we need?

This means
1 pJ / operation
or 1 TeraOp/Watt



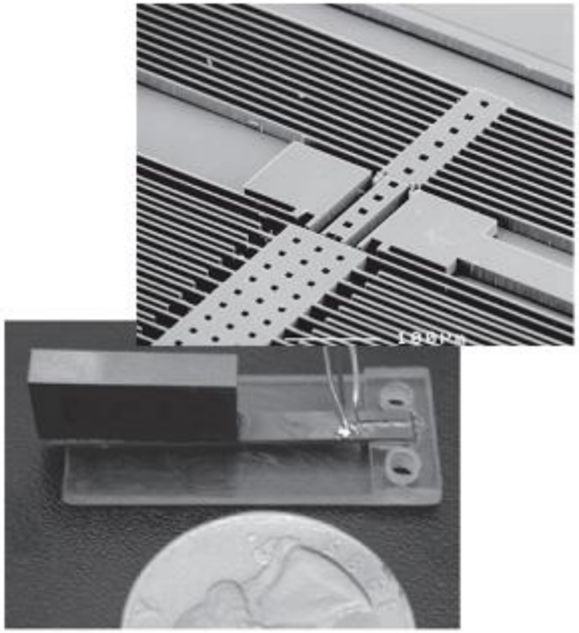
Woh e.a., ISCA 2009

Energy, energy, energy: where does it go?

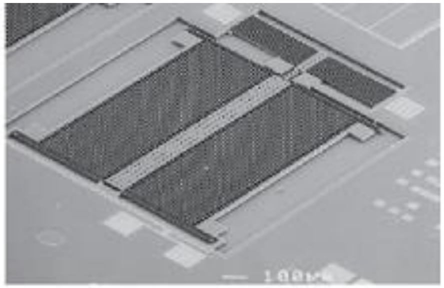


- RISC @ 45 nm, 09 V
- ADD op. 0.5 pJ out of 70 pJ for the ADD instruction
- Overall efficiency 1 / 850 = 0.12 %

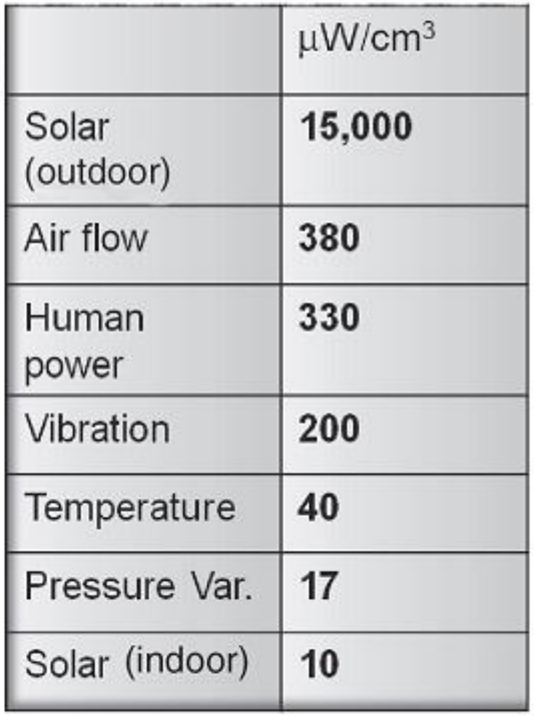
Low end: How much energy in the air?



Vibrations

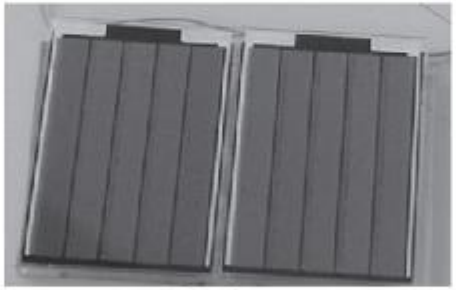


Thermal



	μW/cm^3
Solar (outdoor)	**15,000**
Air flow	**380**
Human power	**330**
Vibration	**200**
Temperature	**40**
Pressure Var.	**17**
Solar (indoor)	**10**

Air Flow



Solar

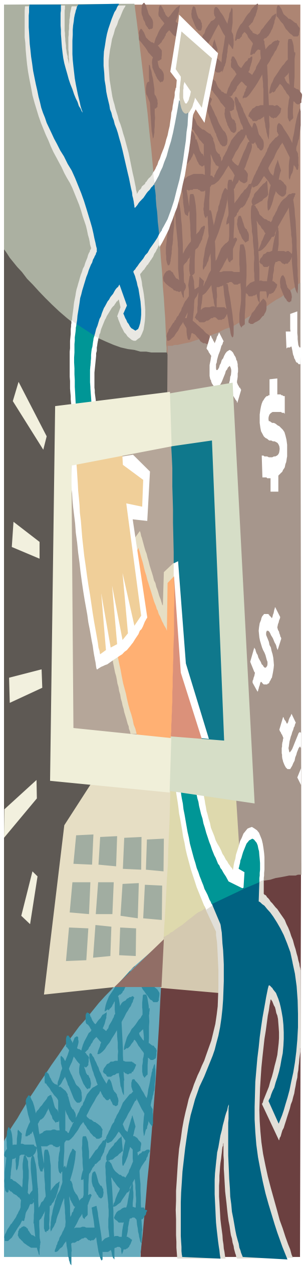
[Rabaey 2009]

Reducing power @ all design levels

$$P = \alpha.f.C.V_{dd}^2$$

$$E = \int P.dt \Rightarrow$$
$$E/cycle = \alpha.C.V_{dd}^2$$

- Algoritmic level
- Compiler level
- Architecture level
- Organization level
- Circuit level
- Silicon level
- Important concepts:
  - Lower Vdd and freq. (even if errors occur) / dynamically adapt Vdd and freq.
  - Reduce circuit
  - Exploit locality
  - Reduce switching activity, glitches, etc.



Algoritmic level

- The best indicator for energy is …..
  …. the number of cycles

- Try alternative algorithms with lower complexity
  - E.g. quick-sort, O(n log n) ⇔ bubble-sort, O (n^2)
  - … but be aware of the 'constant' : O(n log n) ⇒ **c***(n log n)

- Heuristic approach
  - Go for a good solution, not the best !!

Biggest gains at this level !!

Compiler level

- Source-to-Source transformations
  - loop trafo's to improve locality
- Strength reduction
  - E.g. replace Const * A with Add's and Shift's
  - Replace Floating point with Fixed point
- Reduce register pressure / number of accesses to register file
  - Use software bypassing
- Scenarios: current workloads are highly dynamic
  - Determine and predict execution modes
  - Group execution modes into scenarios
  - Perform special optimizations per scenario
    - DFVS: Dynamic Voltage and Frequency Scaling
    - More advanced loop optimizations
- Reorder instructions to reduce bit-transistions

Architecture level

- Going parallel
- Going heterogeneous
  - tune your architecture, exploit SFUs (special function units)
  - trade-off between flexibility / programmability / genericity and efficiency
- Add local memories
  - prefer scratchpad i.s.o. cache
- Cluster FUs and register files (see next slide)
- Reduce bit-width
  - sub-word parallelism (SIMD)

Organization (micro-arch.) level

- Enabling Vdd reduction
  - Super Pipelining
    - cheap way of parallelism
  - Enabling lower freq. ⇒ lower V_{dd}
  - Note 1: don't pipeline if you don't need the performance
  - Note 2: don't exaggerate (like the 31-stage Pentium 4)
- Reduce register traffic
  - avoid unnecessary reads and write
  - make bypass registers visible

Circuit level

- Clock gating
- Power gating
- Multiple Vdd modes
- Reduce glitches: balancing digital path's
- Exploit Zeros
- Special SRAM cells
  - normal SRAM can not scale below Vdd = 0.7 - 0.8 Volt
- Razor method; replay
- Allow errors and add redundancy to architectural invisible structures
  - branch predictor
  - caches
- .. and many more ..

Silicon level

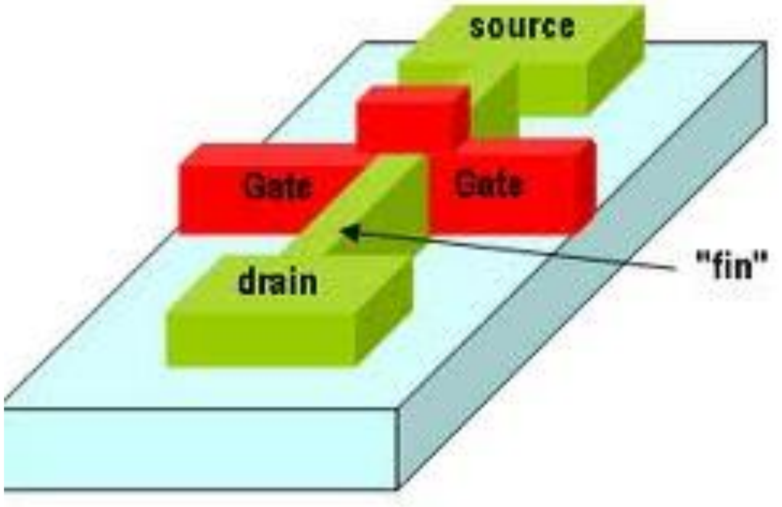
- Higher V_t (V_threshold)
  - Back Biasing control
    - see thesis Maurice Meijer (2011)

- Sub/Near-threshold Vdd

- SOI (Silicon on Insulator)
  - silicon junction is above an electric insulator (silicon dioxide)
  - lowers parasitic device capacitance

- Better transistors: Finfet
  - multi-gate
  - reduce leakage (off-state curent)

- .. and many more



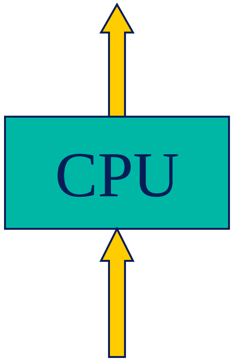
Going parallel

- Running into the
  - Frequency wall
  - ILP wall
  - Memory wall
  - Energy wall

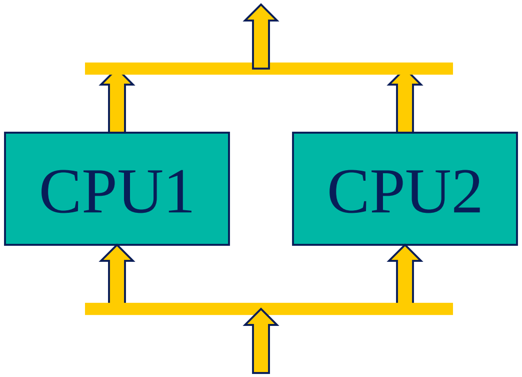
- Chip area enabler: Moore's law goes well below 14 nm
  - What to do with all this area?
  - Multiple processors fit easily on a single die
- Application demands
- Cost effective
  - Reusue: just connect existing processors or processor cores
- Low power: parallelism may allow lowering Vdd

Low power through parallelism

- Sequential Processor
  - Switching capacitance C
  - Frequency f
  - Voltage V
  - $P_1 = \alpha f C V^2$



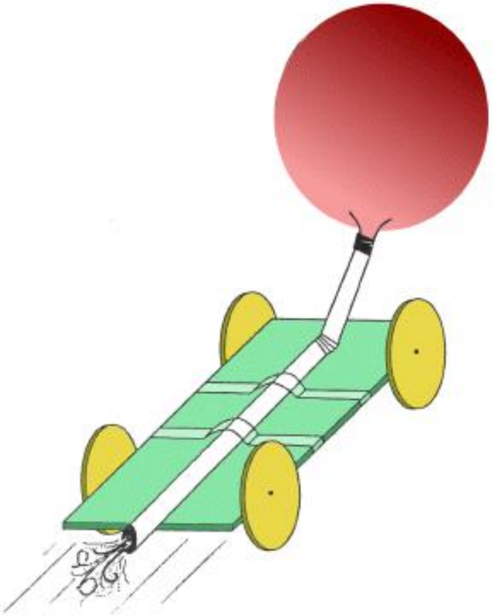
- Parallel Processor (two times the number of units)
  - Switching capacitance 2C
  - Frequency f/2
  - Voltage V’ < V
  - $P_2 = \alpha f/2 * 2CV'^2 = \alpha f C V'^2 < P_1$



- Check yourself whether this works for pipelining as well !

4-D model of parallel architectures

How to speedup your favorite processor?



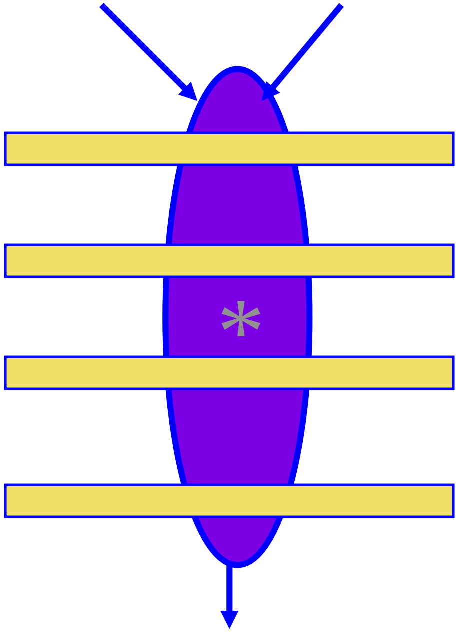
1. Super-pipelining
2. Powerful instructions
   - MD-technique
     - multiple data operands per operation
   - MO-technique
     - multiple operations per instruction
3. Multiple instruction issue
   - Single stream: Superscalar
   - Multiple streams
     - Single core, multiple threads: Simultaneously Multi-Threading
     - Multiple cores

Architecture methods
1. Super pipelining

- Superpipelining:
  - Split one or more of the critical pipeline stages

- Superpipelining degree S:

$$\boldsymbol{S}(\text{architecture}) = \sum_{\forall Op \in I\_set} \boldsymbol{f(Op) * lt\ (Op)}$$



where:

f(op) is *<u>frequency</u>* *of operation op*

lt(op) is *<u>latency</u>* *of operation op*

Architecture methods
2. Powerful Instructions (1)

- MD-technique
  - Multiple data operands per operation
  - SIMD: Single Instruction Multiple Data

Vector instruction:

```
for (i=0, i++, i<64)
   c[i] = a[i] + 5*b[i];
```

or

$\vec{\mathbf{c}} = \vec{\mathbf{a}} + 5*\vec{\mathbf{b}}$

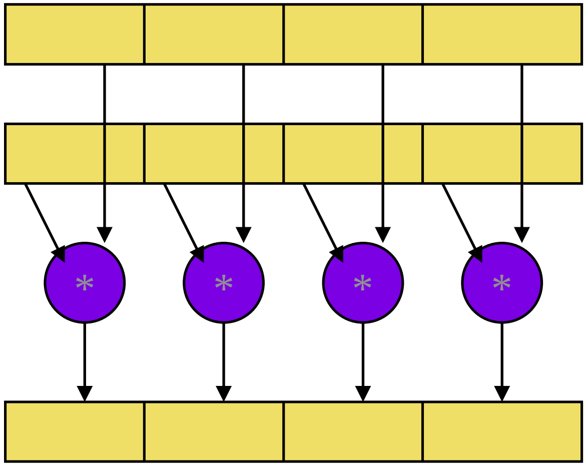
Assembly:

```
set    v1,64
ldv    v1,0(r2)
mulvi  v2,v1,5
ldv    v1,0(r1)
addv   v3,v1,v2
stv    v3,0(r3)
```

Architecture methods
2. Powerful Instructions (1)

- Sub-word parallelism
  - SIMD on restricted scale:
  - Used for Multi-media instructions
  - Many processors support this

- Examples
  - MMX, SSE, SUN-VIS, HP MAX-2, AMD-K7/Athlon 3Dnow, Trimedia II
  - Example: $\Sigma_{i=1..4} |a_i-b_i|$



Architecture methods
2. Powerful Instructions (2)

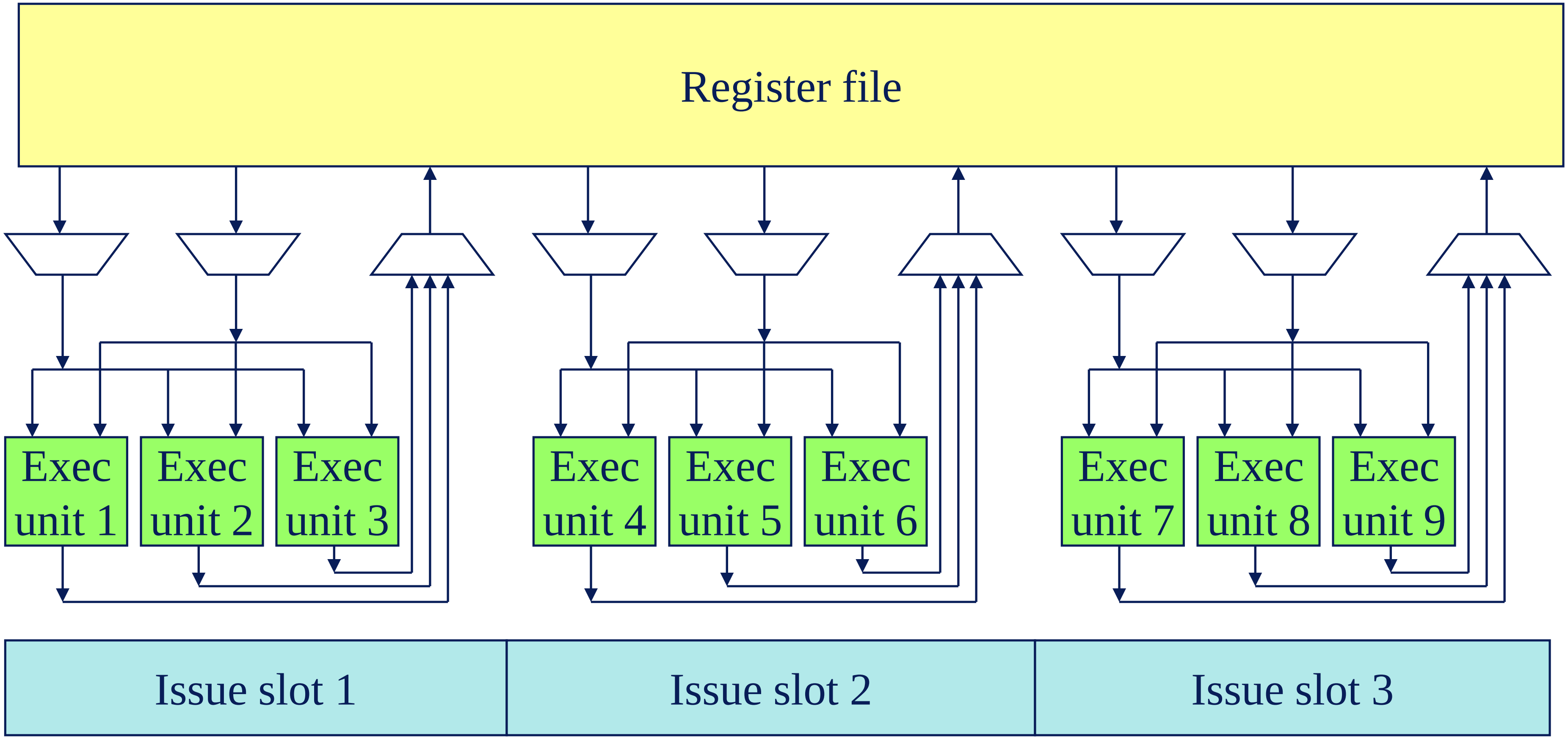
- MO-technique: multiple operations per instruction

- Two options:
  - CISC (Complex Instruction Set Computer)
    - this is what we did in the 'old' days of *microcoded processors*
  - VLIW (Very Long Instruction Word)

field →	FU 1	FU 2	FU 3	FU 4	FU 5
instruction →	sub r8, r5, 3	and r1, r5, 12	mul r6, r5, r2	ld r3, 0(r5)	bnez r5, 13

VLIW instruction example

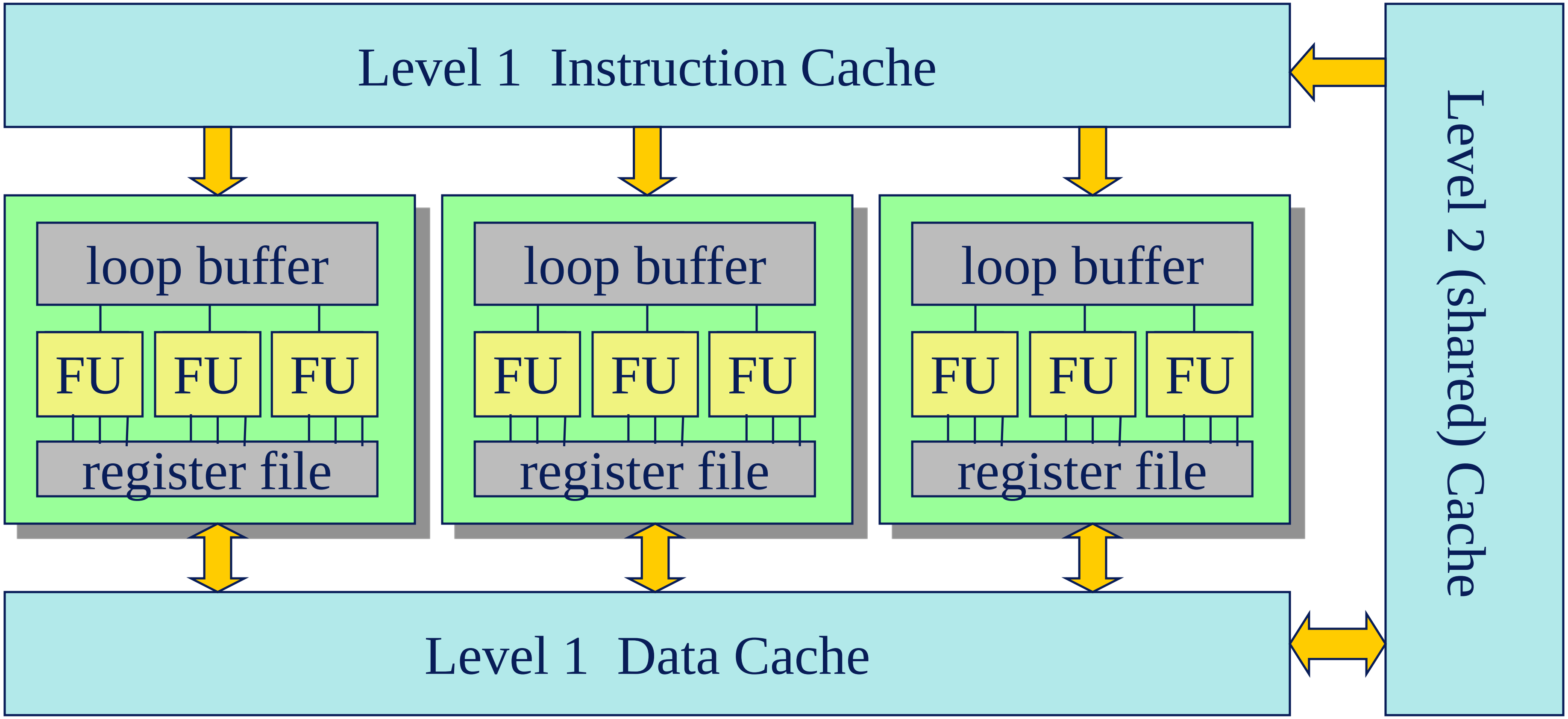
VLIW architecture: central Register File



Q: ***How many ports does the registerfile need for n-issue?***

Clustered VLIW

- Clustering = Splitting up the VLIW data path
  - same can be done for the instruction path –
- Exploit locality @ Level 0, for Instructions and Data



Architecture methods
3. Multiple instruction issue (per cycle)

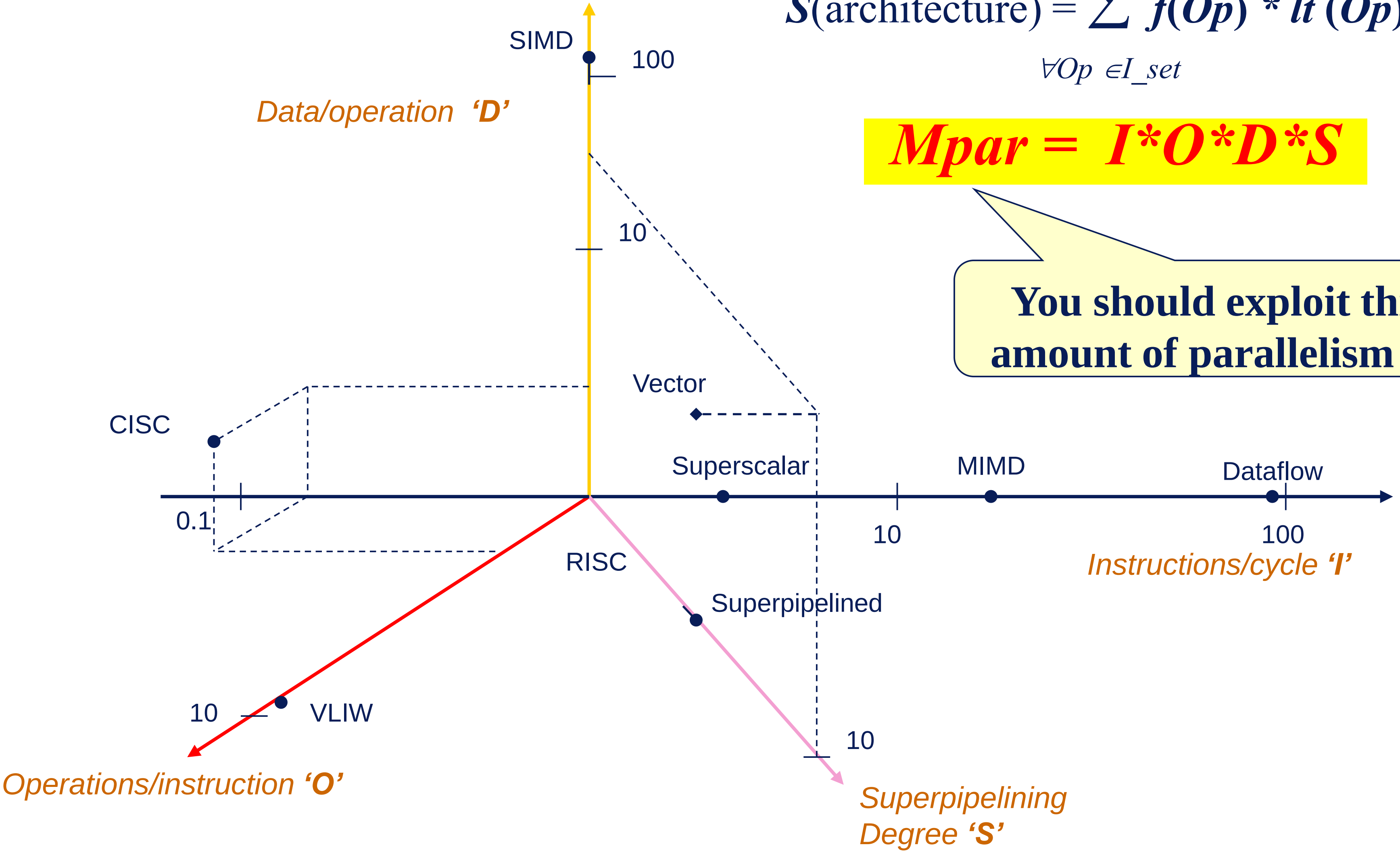
- Who guarantees semantic correctness?
  - can instructions be executed in parallel
- User: he specifies multiple instruction streams
  - Multi-processor: MIMD (Multiple Instruction Multiple Data)
- HW: Run-time detection of ready instructions
  - Superscalar, single instruction stream
- Compiler: Compile into dataflow representation
  - Dataflow processors
  - Multi-threaded processors

Four dimensional representation of the architecture design space <I, O, D, S>

$$S(\text{architecture}) = \sum_{\forall Op \in I\_set} f(Op) * lt\,(Op)$$

Mpar = I*O*D*S

You should exploit this amount of parallelism !!!



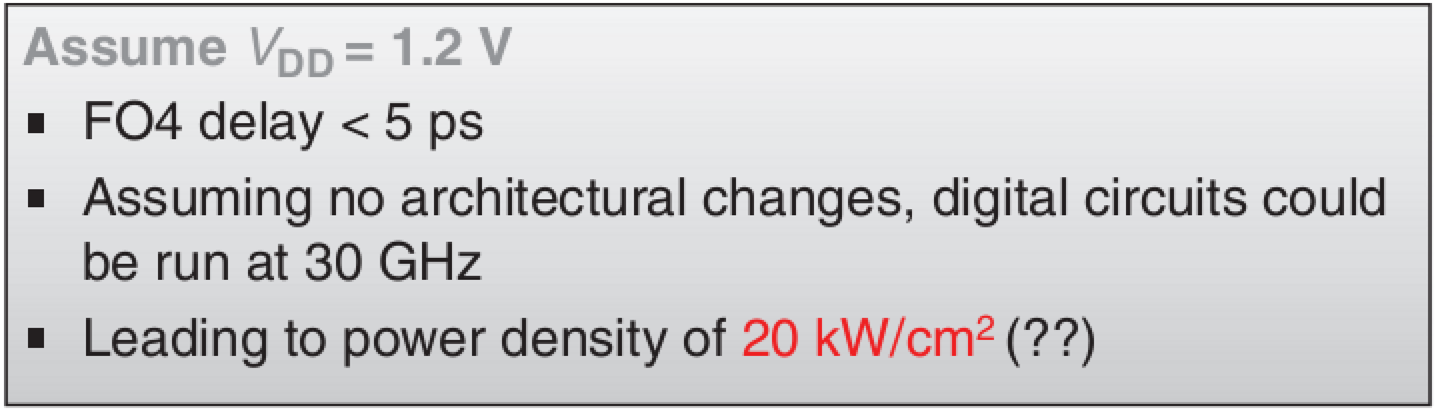
Parallelism Everywhere

- SIMD / vector
  - Xetal (320 PEs), Imap (128 PEs), AnySP (Michigan Univ), GPUs
  - subword support (SSX, NEON, etc.)
- VLIW
  - ADRES, TriMedia, Liquid, TTA, CGRA
  - more dynamic:
    Itanium (static sched., rt mapping), TRIPS/EDGE (rt scheduling)
- Multi-threaded
  - idea: hide long latencies
  - Denelcor HEP (1982), SUN Niagara (2005), GPUs
- Multi-processor
  - RaW, PicoChip, ARM, Intel/AMD, GRID, Farms, even GPUs
- **Hybrid: actually, most are hybrid !!**

Going Heterogeneous

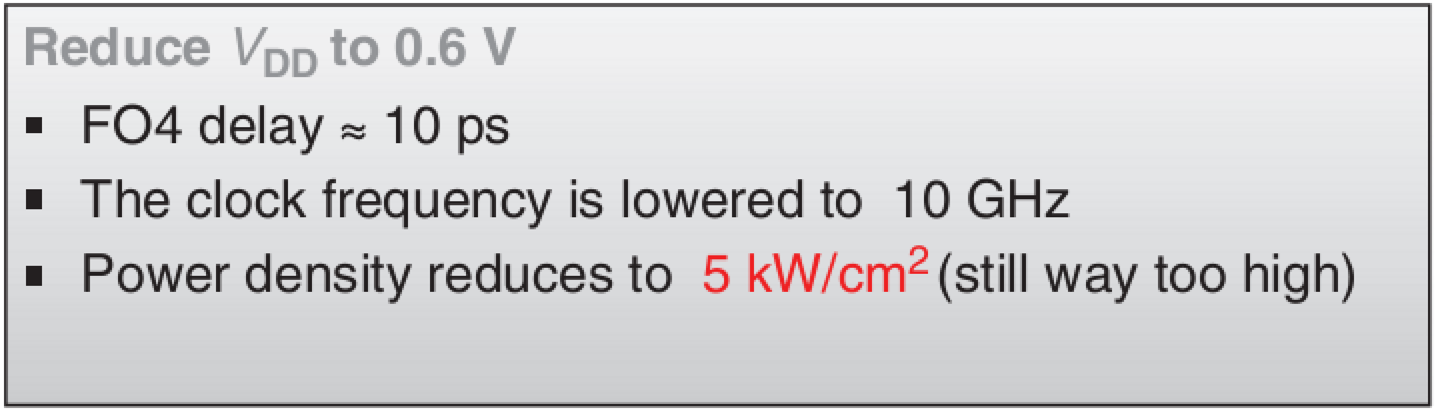
- Why would we do this?
  - Energy Efficiency (pJ / operation or MOPS / Watt)
  - Area Efficiency (MOPS / area)
- Technology favors heterogeneous
  - Why?

A 20nm scenario (high end processor)



Assume V_{DD} = 1.2 V

- FO4 delay < 5 ps
- Assuming no architectural changes, digital circuits could be run at 30 GHz
- Leading to power density of 20 kW/cm^2 (??)



Reduce V_{DD} to 0.6 V

- FO4 delay ≈ 10 ps
- The clock frequency is lowered to 10 GHz
- Power density reduces to 5 kW/cm^2 (still way too high)

[Ref: S. Borkar, Intel]

This means:

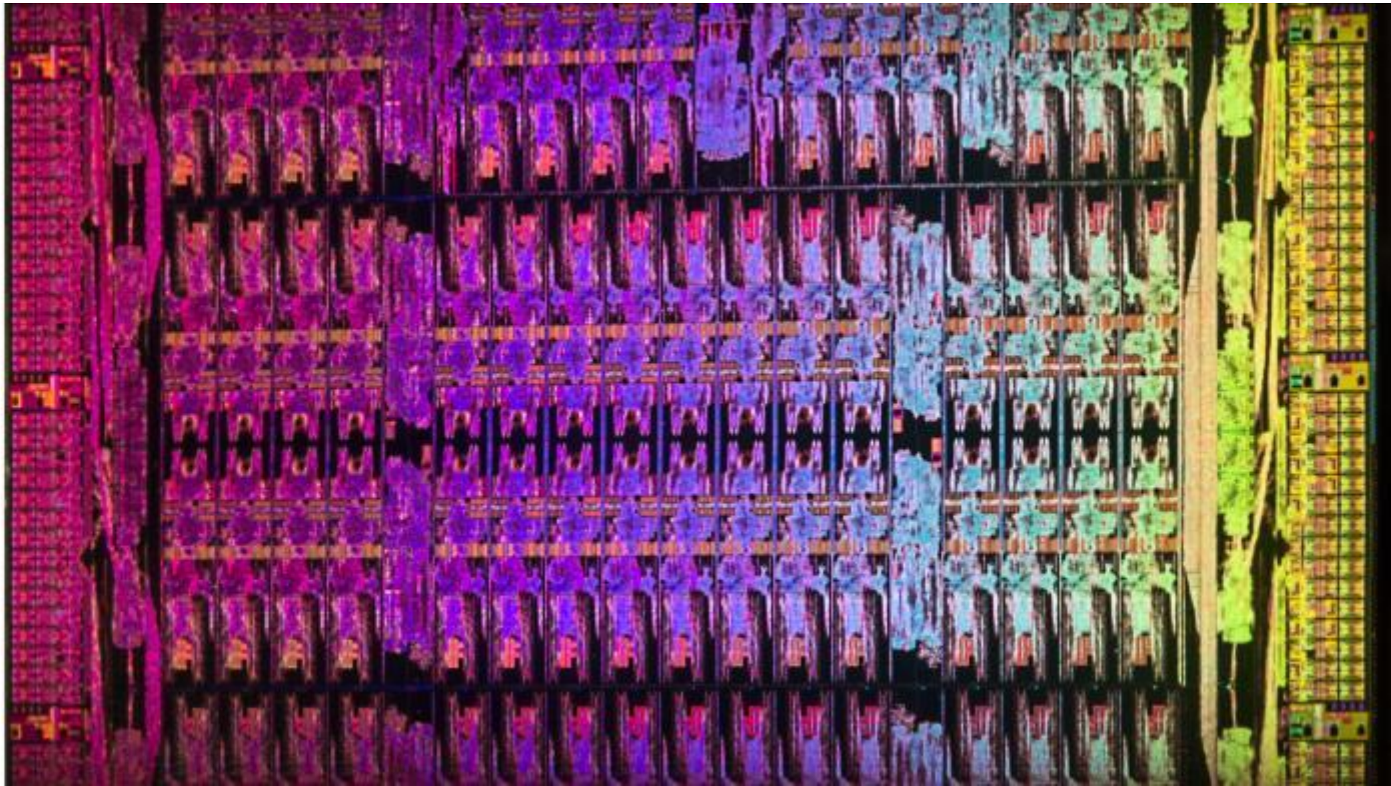
- a 2cm^2 processor consumens 10 kW
- a bound of 100W requires only 1% to be active ⇒ **dark silicon**

Where are we?

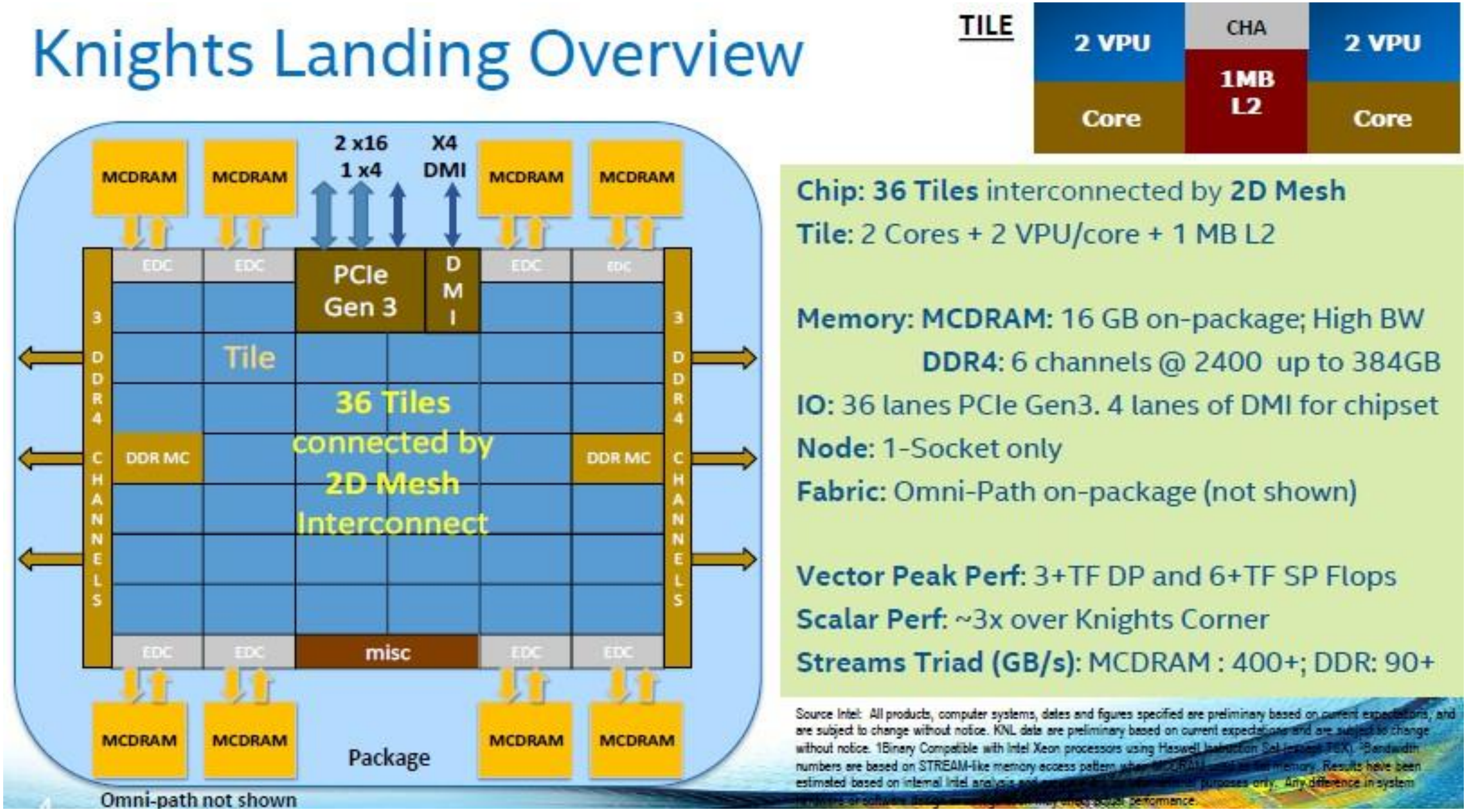
- 3 examples
- Multi-core: Intel KNL (Knight Landing)
- Many-core: GPU
- Dedicated: TPU

Xeon Phi Knight Landing

- 72 cores: Silvermont cores
  - 4 Threads/core, 32KB L1 + 1MB L2, 16 way associative
- 2D-mesh connected
- 8 channels to Hybrid Cubes of Micron (DDR4)



KNL details



Knights Landing Overview

Omni-path not shown

Chip: 36 Tiles interconnected by **2D Mesh**

Tile: 2 Cores + 2 VPU/core + 1 MB L2

Memory: MCDRAM: 16 GB on-package; High BW

DDR4: 6 channels @ 2400 up to 384GB

IO: 36 lanes PCIe Gen3. 4 lanes of DMI for chipset

Node: 1-Socket only

Fabric: Omni-Path on-package (not shown)

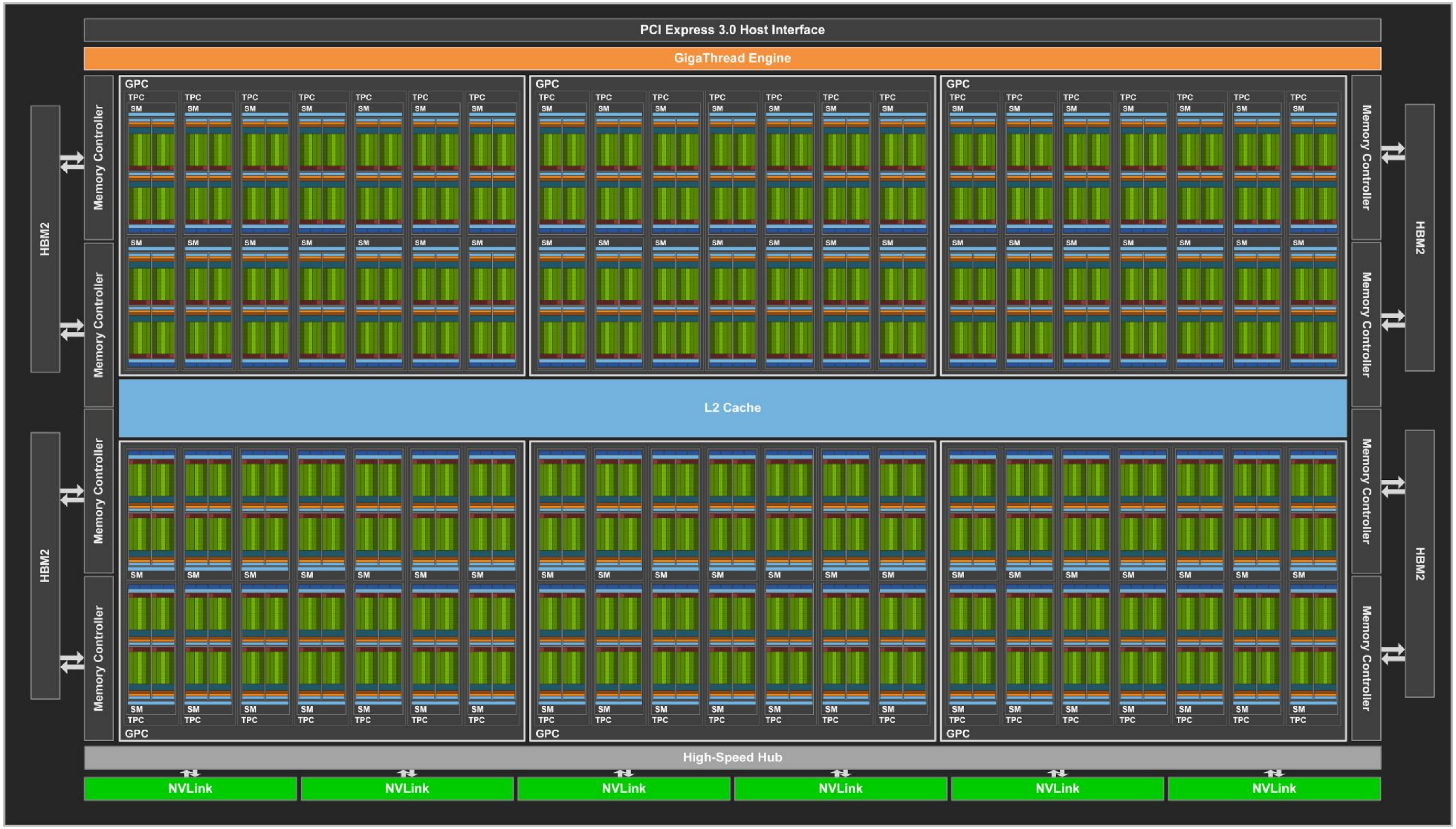
Vector Peak Perf: 3+TF DP and 6+TF SP Flops

Scalar Perf: ~3x over Knights Corner

Streams Triad (GB/s): MCDRAM : 400+; DDR: 90+

Source Intel: All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice. KNL data are preliminary based on current expectations and are subject to change without notice. 1Binary Compatible with Intel Xeon processors using Haswell Instruction Set (except TSX). 2Bandwidth numbers are based on STREAM-like memory access pattern when MCDRAM used as flat memory. Results have been estimated based on internal Intel analysis and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.

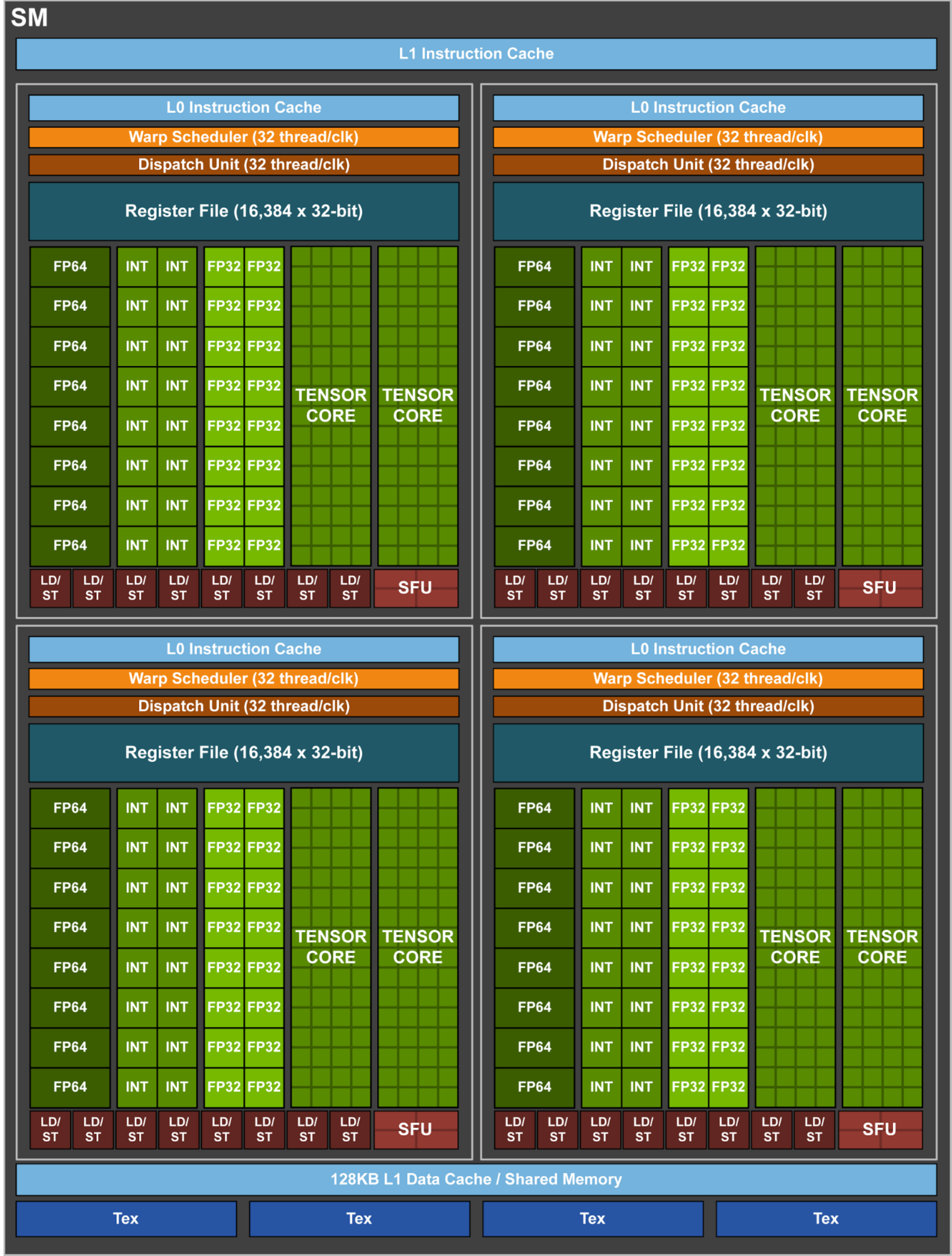
GPU: NVIDIA Volta V100 (GTC May 2017)



- up to 80 cores, 5120 PEs (FP32), 20 MB register space
- 815 mm^2, 21.1 Btransistors, 12 nm, 300 W
- peak: 120 TFlops/s (FP16) => 2.5 pJ/op

1 SM core

- Units:
  - 8 tensor cores/SM
  - 64 Int units
  - 64 FP32
  - 32 FP64
  - 32 Ld/St
  - 4 SFUs
- 128 LB L1 Data $
- 4 warp schedulers


SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
FP64
INT
FP32
TENSOR CORE
LD/ ST
SFU
128KB L1 Data Cache / Shared Memory
Tex

Tensor core operation

- D = AxB + C, all 4x4 matrices
- 64 floating point MAC operations per clock

$$\mathbf{D} = \left(\begin{array}{cccc} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{array}\right) \left(\begin{array}{cccc} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{array}\right) + \left(\begin{array}{cccc} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{array}\right)$$

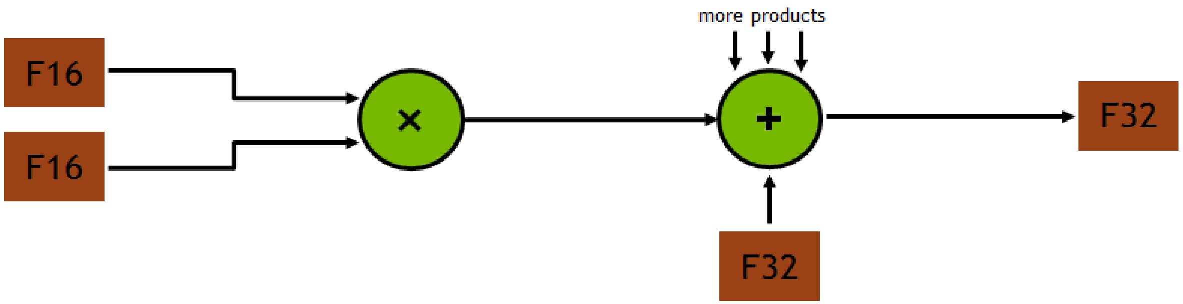
FP16 or FP32 FP16 FP16 FP16 or FP32

FP16 storage/input

Full precision product

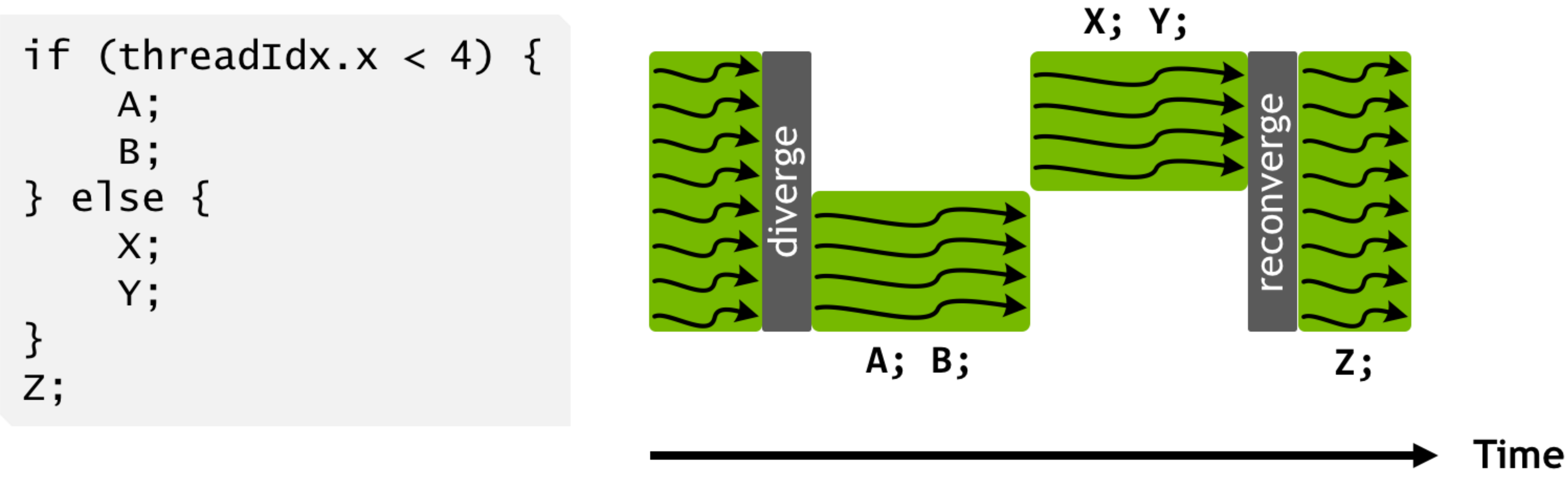
Sum with FP32 accumulator

Convert to FP32 result



GPU remarks

- Ext memory Bandwith / Perf ratio low
- Less cache space compared to m-CPUs
- SIMD model => Divergence problem
- average << peak performance
- see https://devblogs.nvidia.com/parallelforall/inside-volta/

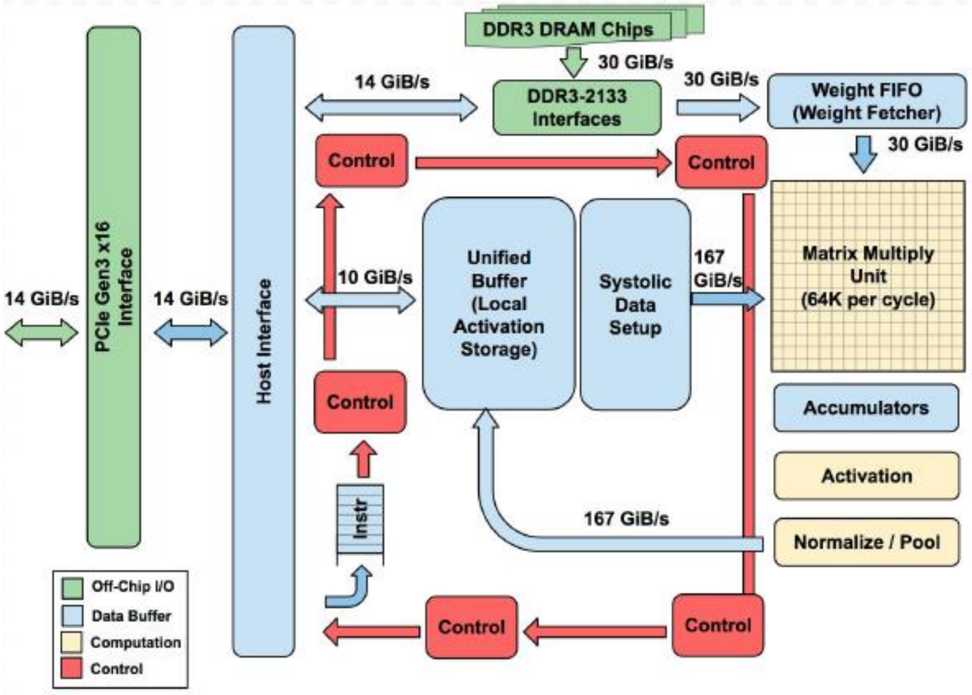


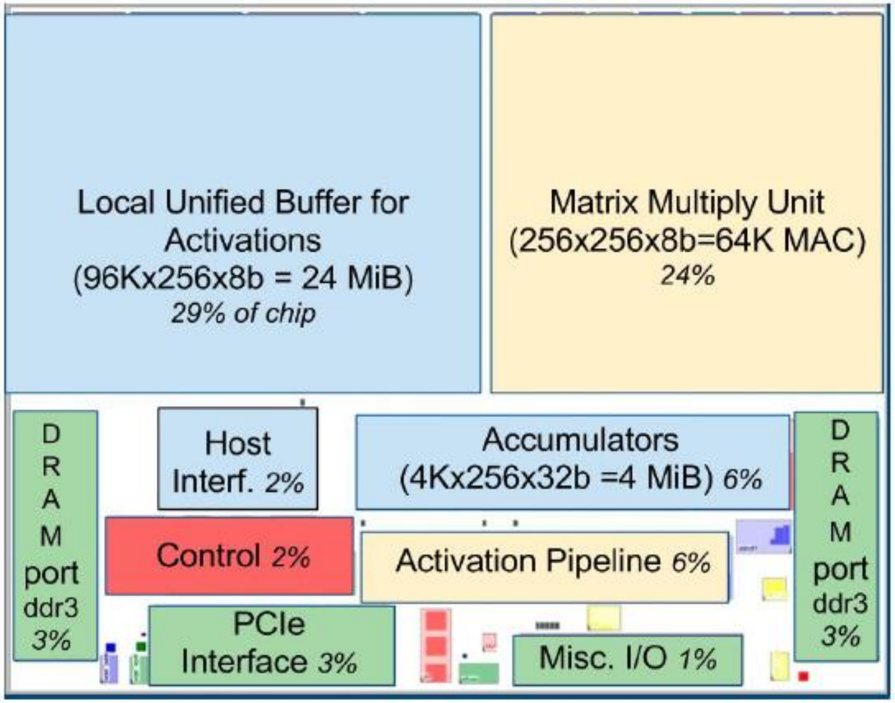
```
if (threadIdx.x < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
```

TPU: Tensor Processing Unit (Google)

- Google TPU, board and cloud usage

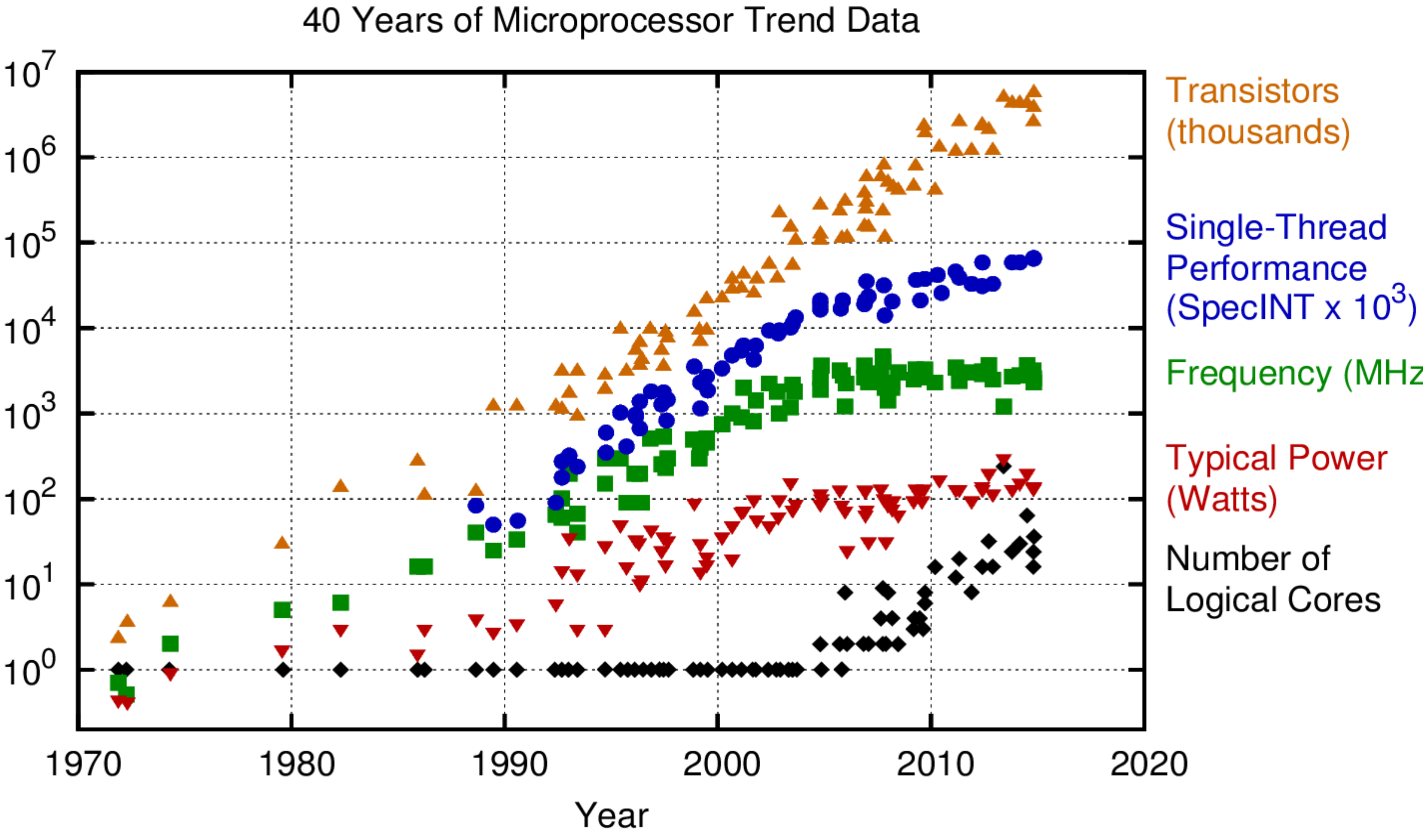
TPU details




Local Unified Buffer for Activations (96Kx256x8b = 24 MiB) 29% of chip
Matrix Multiply Unit (256x256x8b=64K MAC) 24%
DRAM port ddr3 3%
Host Interf. 2%
Accumulators (4Kx256x32b =4 MiB) 6%
DRAM port ddr3 3%
Control 2%
Activation Pipeline 6%
PCIe Interface 3%
Misc. I/O 1%

- Block diagram and Floor plan
- work horse: systolic array of 256x256 8-bit MACs
  - 40 W, 28 nm
  - 92 TOPS/s => 0.43 pJ/8-bit Operation

Conclusions

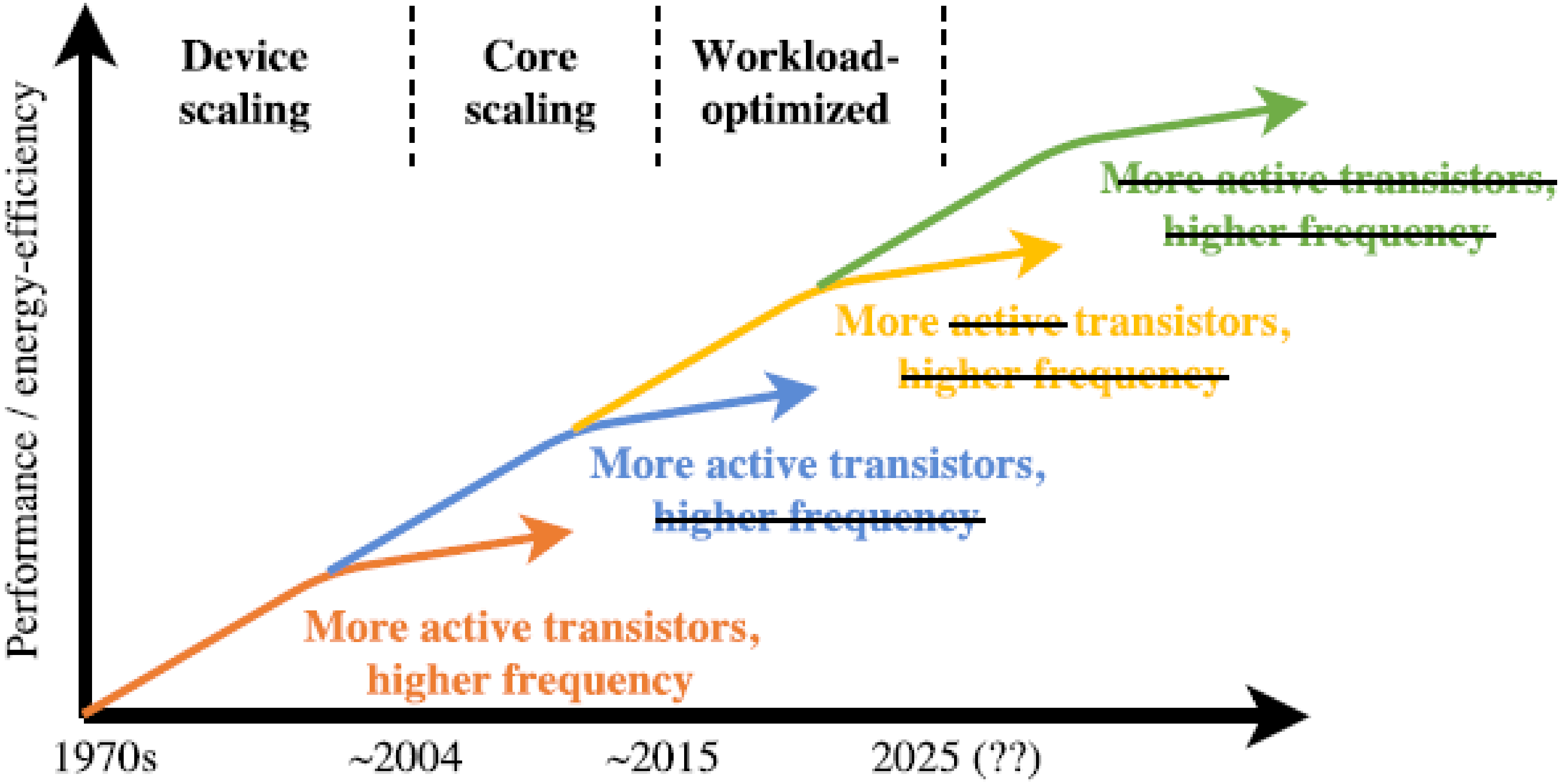


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

Key Observations

- Energy and Performance are main design drivers
- Moore still alive (although somewhat slower)
    - Billions of transistors
    - Where does it stop? And thereafter?
- Power limit reached
    - Frequency limited to ~3GHz
    - Dark Silicon
- Single thread performance hardly increases
- Going multicore since 2005

Conclusions



Research

- Architecture and implementation
  - Flexibility : the CISC vs RISC debate is open again
  - Why exact: Go approximate
  - Near/Sub threshold

- Compiler / Mapping
  - Tiling, Fusion
  - Partitioning
  - Vectorization
  - Using templates / species

JUST
FOR
YOU

Research

- Programming related
  - VM support
  - Coherence
  - Data Flow model
  - OpenMP4
- New paradigms
  - Near Memory Computing
  - CIM: Computing In Memory
  - Quantum
  - and many others
    - Adiabatic, Nano, DNA, Optic, Analog...
- THE SKY IS THE LIMIT

Finally: Can we match your brain ???

- **Performance** = 100 Billion (10^{11}) Neurons * 1000 (10^3) Connections/Neuron * 200 ($2 * 10^2$) Calculations Per Second Per Connection =

  2 * 10^16 Calculations Per Second

- **Memory** = 100 Billion (10^{11}) Neurons * 1000 (10^3) Connections/Neuron * 10 bytes (information about connection strength and adress of output neuron, type of synapse) = **10^{15} bytes = 1 PB = 1000 TB**

How far off are we?

Brain needs only 20 Watt

and processors need MegaWatts

How come????