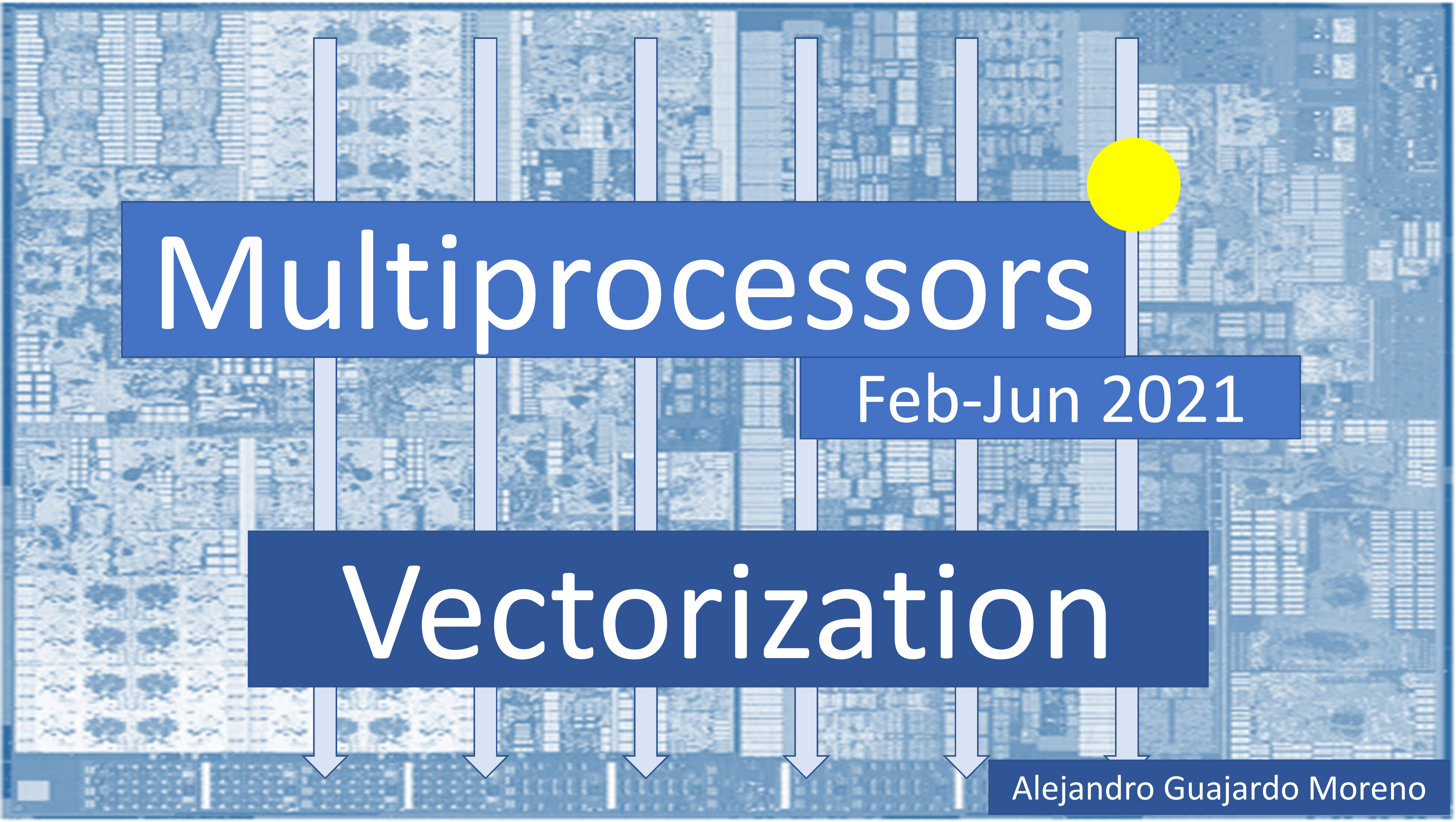
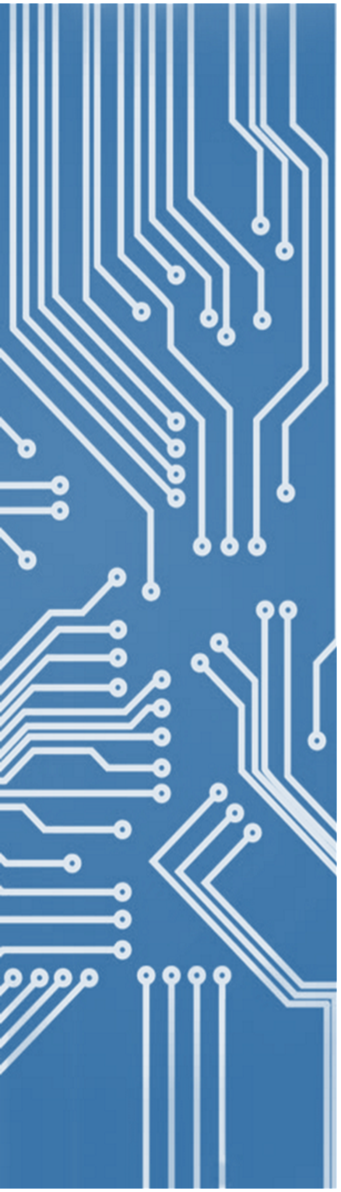


Multiprocessors

Feb-Jun 2021

Vectorization

Alejandro Guajardo Moreno



OBJECTIVES

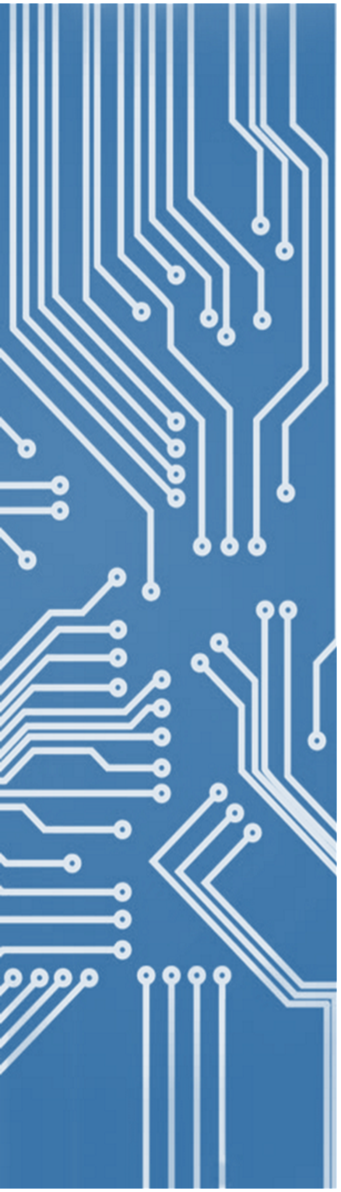
How do computers get faster and walls encountered?

What is & why Vectorization?

Instruction sets w/support for Vectorization

Vector processors

Vectorization & Coding in C

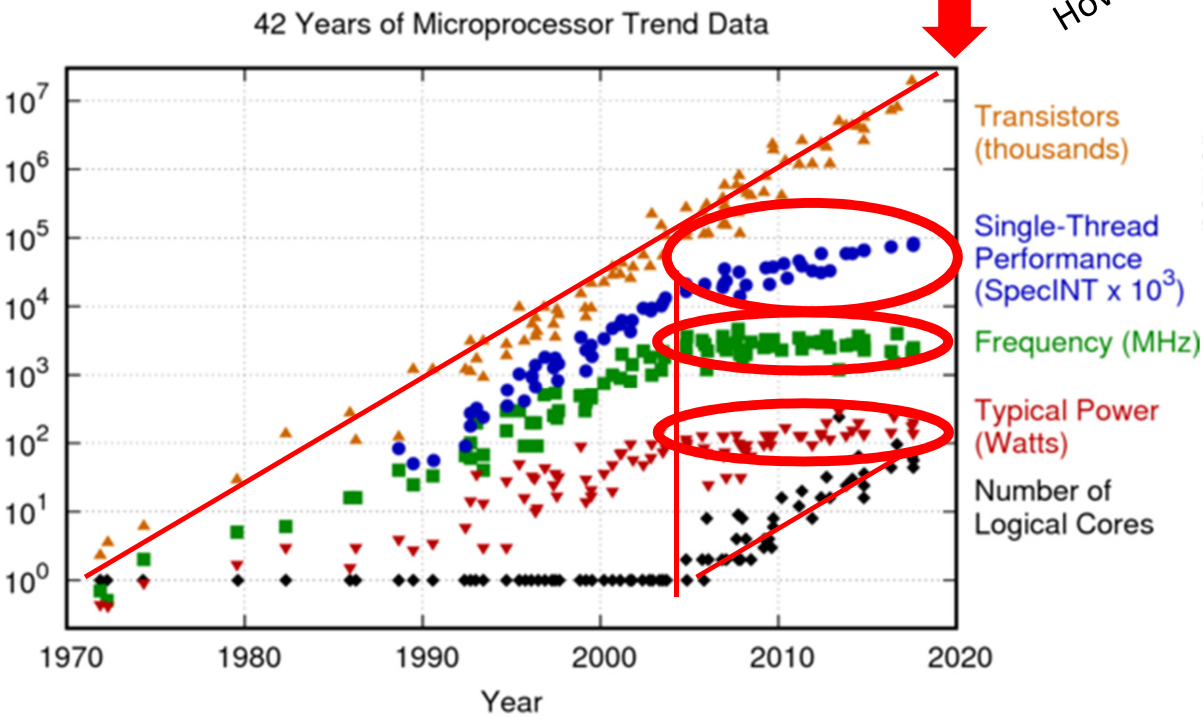


How do computers get faster and walls encountered?

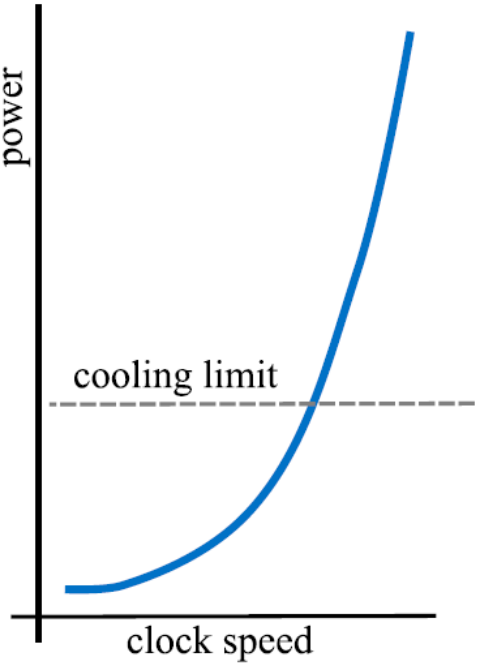
- Power Wall
    - Frequency Wall

Power & Frequency Wall

Epyc Rome (64C, 256MB)
Why so many transistors?
How are they used?

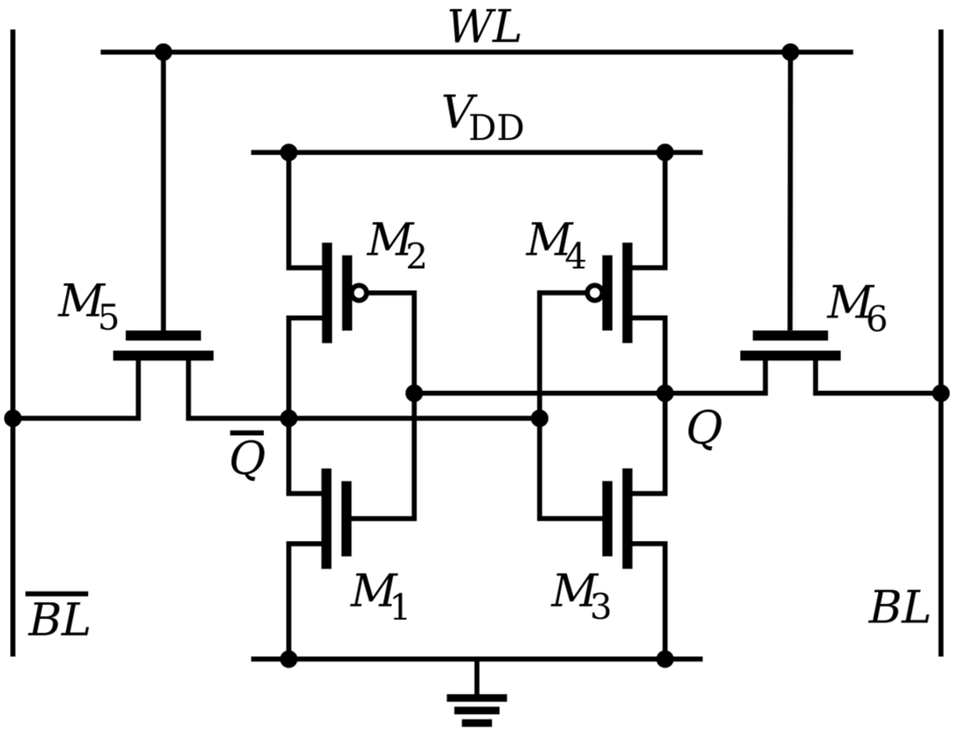


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp



Power & Frequency Wall

Why so many transistors? L2-L3 Caches



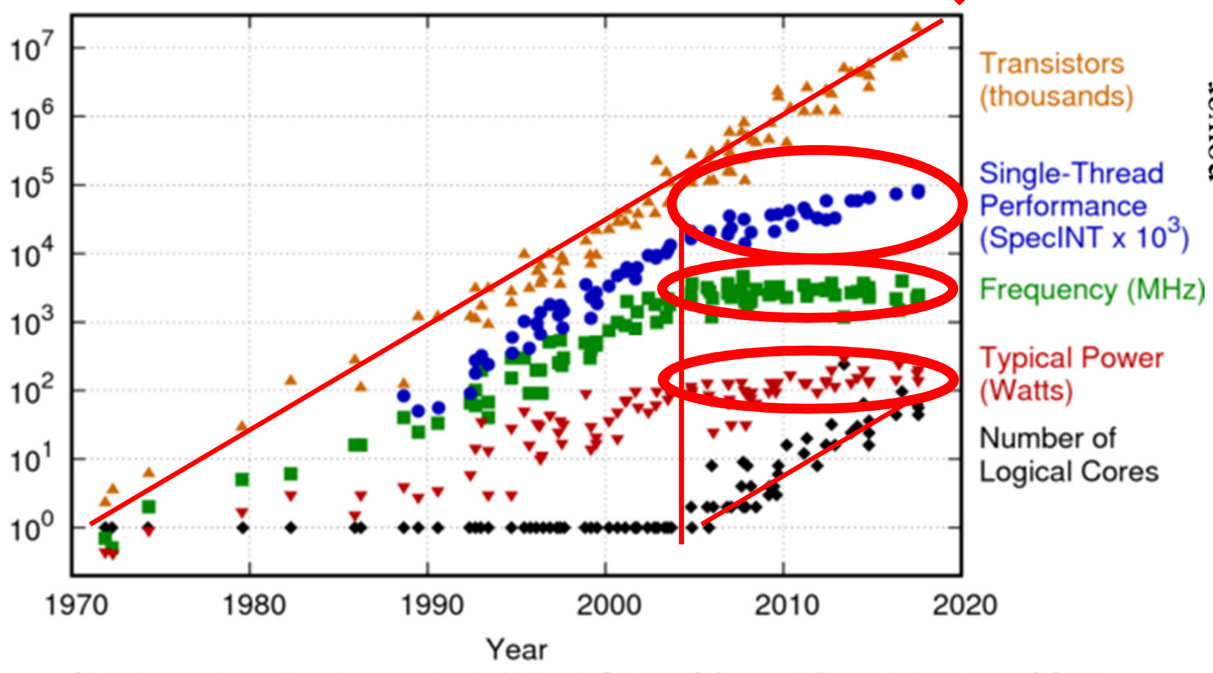
- 1 bit of SRAM
- Static RAM, not SDRAM!
- Typically based on MOSFET transistors.
- 4 transistors needed to hold the bit of data.
- 2 transistors as gates.
- 6 transistors per cell

256 MB
268,435,456 B
2,147,483,648 b
12,884,901,888 transistors

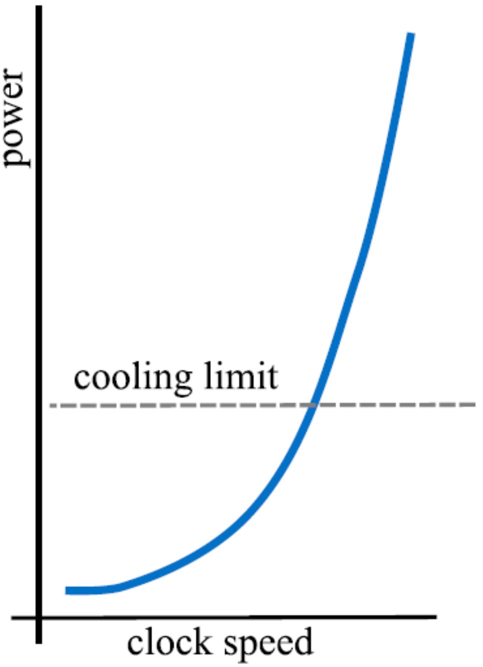
Power & Frequency Wall

Epyc Rome (64C, 256MB)
Transistors needed for cache?
12.88 Billion!

42 Years of Microprocessor Trend Data

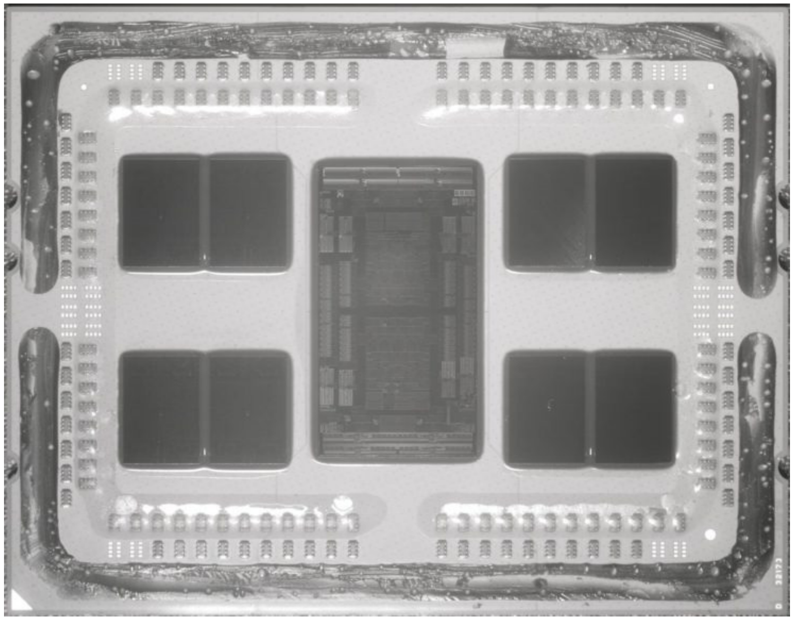


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp



Power & Frequency Wall

AMD’s 2nd Gen EPYC Rome



- 39.54 Billion on The Entire Chip

- 9 die design MCM (Multi-Chip-Module)
- 8 Compute Core dies (CCD)
  - Each CCD has 2 Compute Core Complexes (CCX)
- 1 I/O die

https://wccftech.com/amd-2nd-gen-epyc-rome-iod-ccd-chipshots-39-billion-transistors/

Power & Frequency Wall

Does it really matter?

“The Ring of Death”

Some analysts claim: “The failures resulted from the multicore IBM Cell derivative causing solder connections to melt and separate from the circuit board.

XBOX 360 lesson learned!

How do computers get faster and walls encountered?

- Power Wall
- Frequency Wall
- ILP Wall

ILP – Instruction Level Parallelism Wall

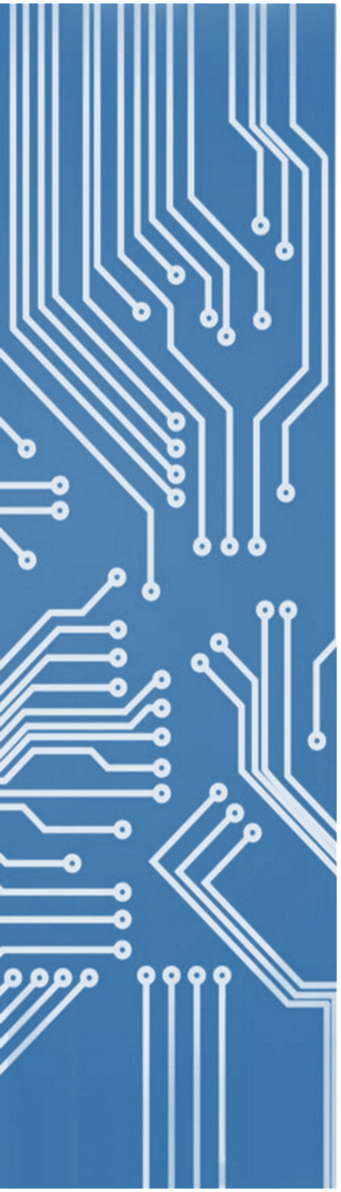
Scalar

Pipeline Stage

FETCH	DECODE	EXECUTE	STORE					
	FETCH	DECODE	EXECUTE	STORE				
		FETCH	DECODE	EXECUTE	STORE			
			FETCH	DECODE	EXECUTE	STORE		
				FETCH	DECODE	EXECUTE	STORE	
					FETCH	DECODE	EXECUTE	STORE

Clock Cycles

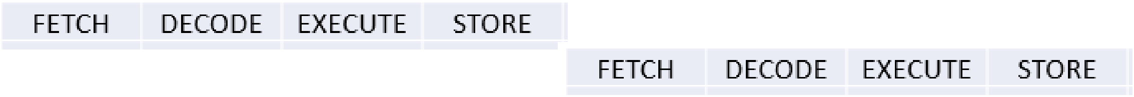
Scalar (pipelined)



ILP – Instruction Level Parallelism Wall

Important concepts

CPI = Cycles Per (Low Level) Instruction



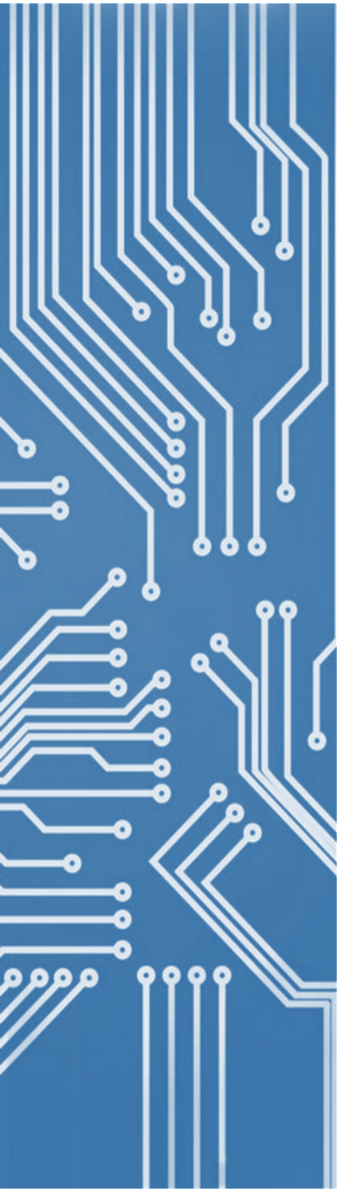
FETCH DECODE EXECUTE STORE

FETCH DECODE EXECUTE STORE

CPI = 4

Pipeline Depth # of stages in the pipeline.

Pipeline Depth = 4



ILP – Instruction Level Parallelism Wall

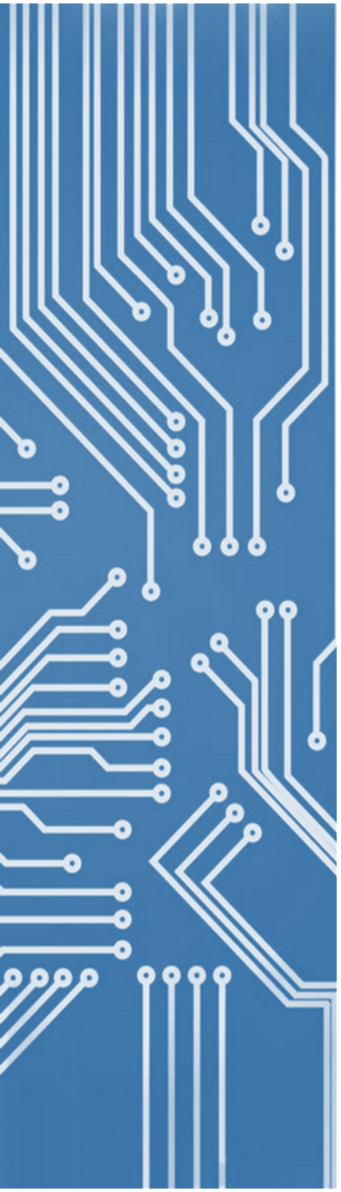
Important concepts

FETCH	DECODE	EXECUTE	STORE					
	FETCH	DECODE	EXECUTE	STORE				
		FETCH	DECODE	EXECUTE	STORE			
			FETCH	DECODE	EXECUTE	STORE		
				FETCH	DECODE	EXECUTE	STORE	
					FETCH	DECODE	EXECUTE	STORE

CPI = 1

Pipeline Depth = 4

ILP – Instruction Level Parallelism Wall

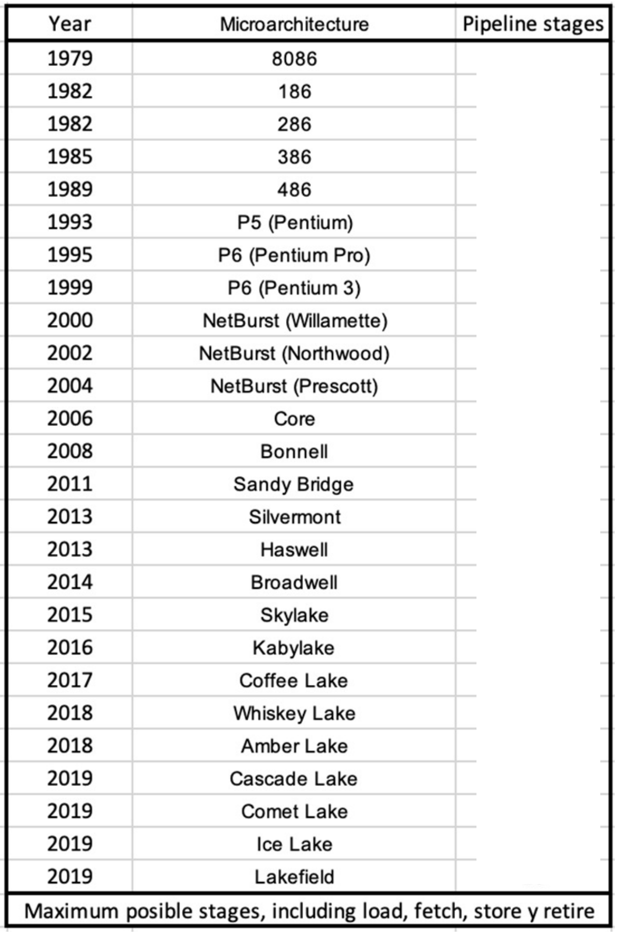


How many pipeline stages are in an Intel® CPUs?

A = Depends…

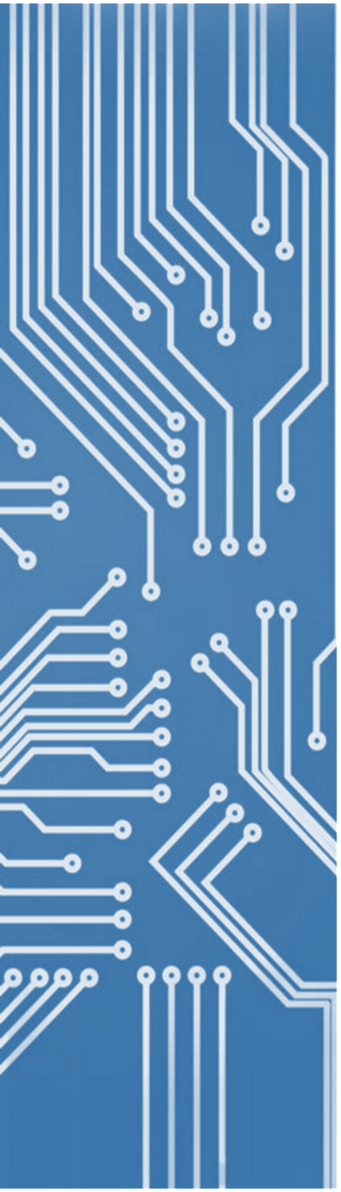
On the Instruction Set

Implementation



Year	Microarchitecture	Pipeline stages
1979	8086	
1982	186	
1982	286	
1985	386	
1989	486	
1993	P5 (Pentium)	
1995	P6 (Pentium Pro)	
1999	P6 (Pentium 3)	
2000	NetBurst (Willamette)	
2002	NetBurst (Northwood)	
2004	NetBurst (Prescott)	
2006	Core	
2008	Bonnell	
2011	Sandy Bridge	
2013	Silvermont	
2013	Haswell	
2014	Broadwell	
2015	Skylake	
2016	Kabylake	
2017	Coffee Lake	
2018	Whiskey Lake	
2018	Amber Lake	
2019	Cascade Lake	
2019	Comet Lake	
2019	Ice Lake	
2019	Lakefield	

Maximum posible stages, including load, fetch, store y retire



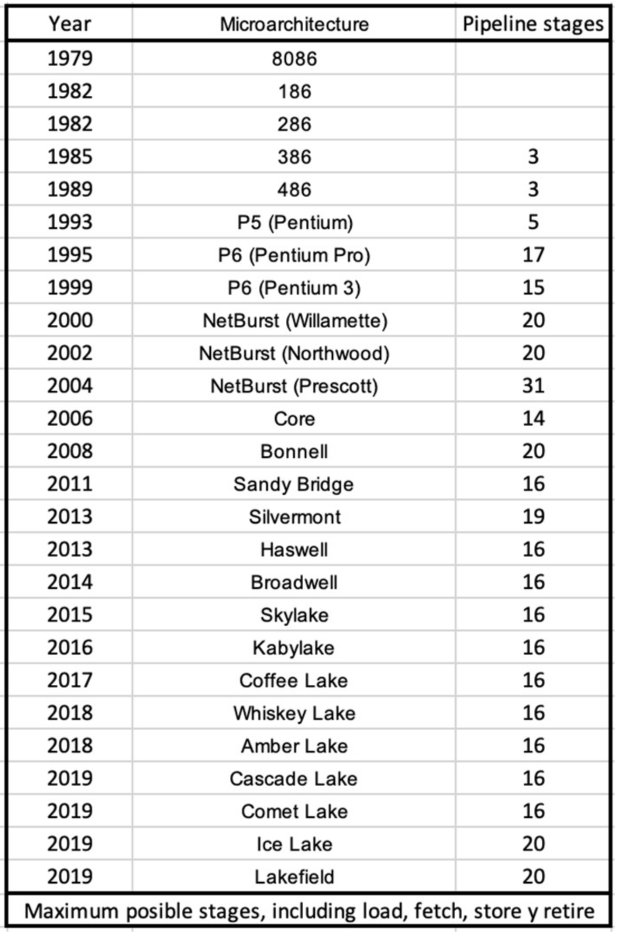
ILP – Instruction Level Parallelism Wall

How many pipeline stages are in an Intel® CPUs?

A = Depends…

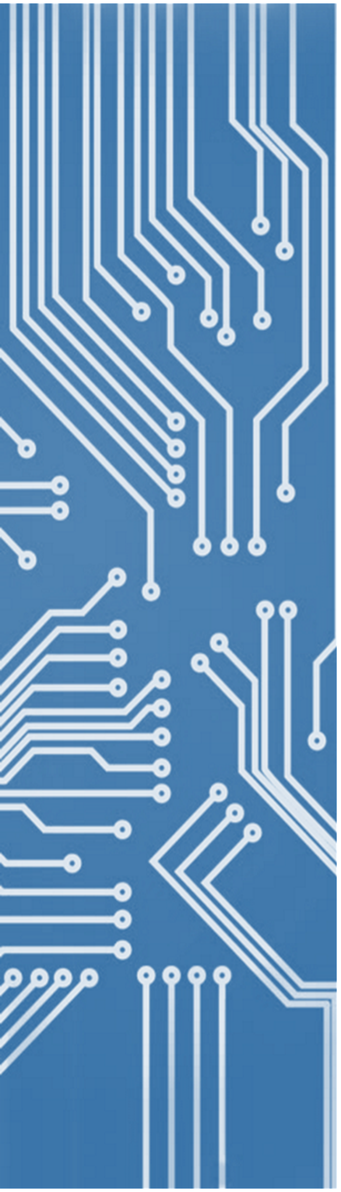
On the Instruction Set

Implementation



Year	Microarchitecture	Pipeline stages
1979	8086	
1982	186	
1982	286	
1985	386	3
1989	486	3
1993	P5 (Pentium)	5
1995	P6 (Pentium Pro)	17
1999	P6 (Pentium 3)	15
2000	NetBurst (Willamette)	20
2002	NetBurst (Northwood)	20
2004	NetBurst (Prescott)	31
2006	Core	14
2008	Bonnell	20
2011	Sandy Bridge	16
2013	Silvermont	19
2013	Haswell	16
2014	Broadwell	16
2015	Skylake	16
2016	Kabylake	16
2017	Coffee Lake	16
2018	Whiskey Lake	16
2018	Amber Lake	16
2019	Cascade Lake	16
2019	Comet Lake	16
2019	Ice Lake	20
2019	Lakefield	20

Maximum posible stages, including load, fetch, store y retire



ILP – Instruction Level Parallelism Wall

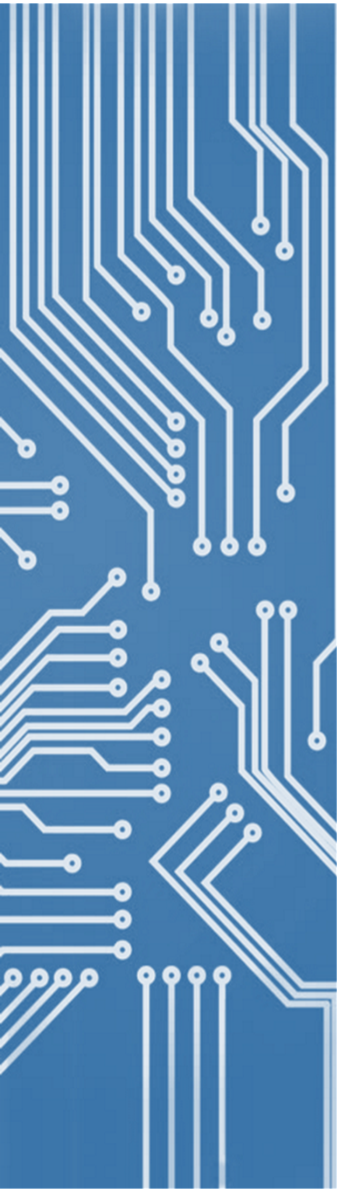
Scalar

Pipeline Stage

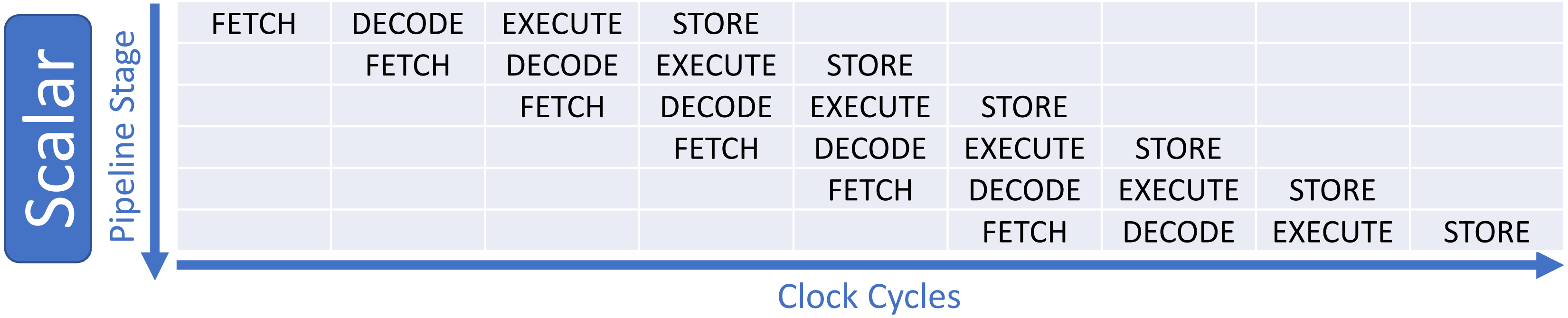
FETCH	DECODE	EXECUTE	STORE					
	FETCH	DECODE	EXECUTE	STORE				
		FETCH	DECODE	EXECUTE	STORE			
			FETCH	DECODE	EXECUTE	STORE		
				FETCH	DECODE	EXECUTE	STORE	
					FETCH	DECODE	EXECUTE	STORE

Clock Cycles

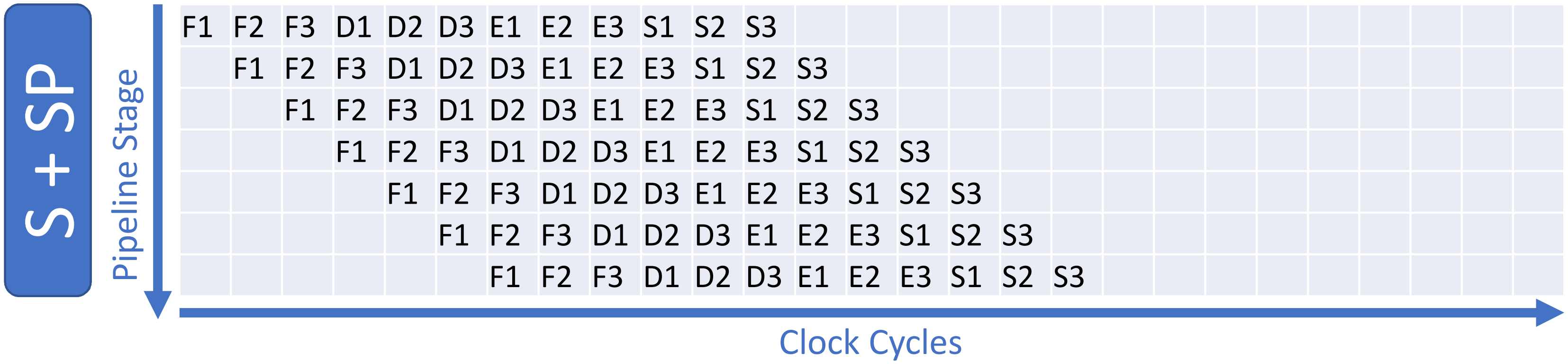
Scalar (pipelined)



ILP – Instruction Level Parallelism Wall



FETCH	DECODE	EXECUTE	STORE					
	FETCH	DECODE	EXECUTE	STORE				
		FETCH	DECODE	EXECUTE	STORE			
			FETCH	DECODE	EXECUTE	STORE		
				FETCH	DECODE	EXECUTE	STORE	
					FETCH	DECODE	EXECUTE	STORE



F1	F2	F3	D1	D2	D3	E1	E2	E3	S1	S2	S3															
	F1	F2	F3	D1	D2	D3	E1	E2	E3	S1	S2	S3														
		F1	F2	F3	D1	D2	D3	E1	E2	E3	S1	S2	S3													
			F1	F2	F3	D1	D2	D3	E1	E2	E3	S1	S2	S3												
				F1	F2	F3	D1	D2	D3	E1	E2	E3	S1	S2	S3											
					F1	F2	F3	D1	D2	D3	E1	E2	E3	S1	S2	S3										
						F1	F2	F3	D1	D2	D3	E1	E2	E3	S1	S2	S3									

Scalar + Superpipelined

ILP – Instruction Level Parallelism Wall

Scalar

Pipeline Stage

FETCH	DECODE	EXECUTE	STORE					
	FETCH	DECODE	EXECUTE	STORE				
		FETCH	DECODE	EXECUTE	STORE			
			FETCH	DECODE	EXECUTE	STORE		
				FETCH	DECODE	EXECUTE	STORE	
					FETCH	DECODE	EXECUTE	STORE

Clock Cycles

SScalar

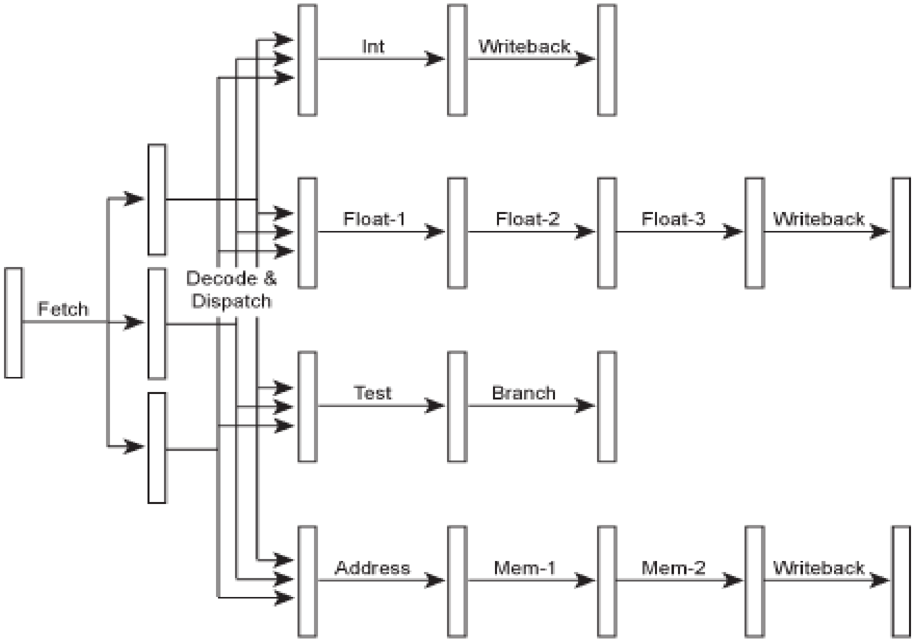
Pipeline Stage

FETCH	DECODE	EXECUTE	STORE					
FETCH	DECODE	EXECUTE	STORE					
	FETCH	DECODE	EXECUTE	STORE				
	FETCH	DECODE	EXECUTE	STORE				
		FETCH	DECODE	EXECUTE	STORE			
		FETCH	DECODE	EXECUTE	STORE			

Clock Cycles

Superscalar

ILP – Instruction Level Parallelism Wall



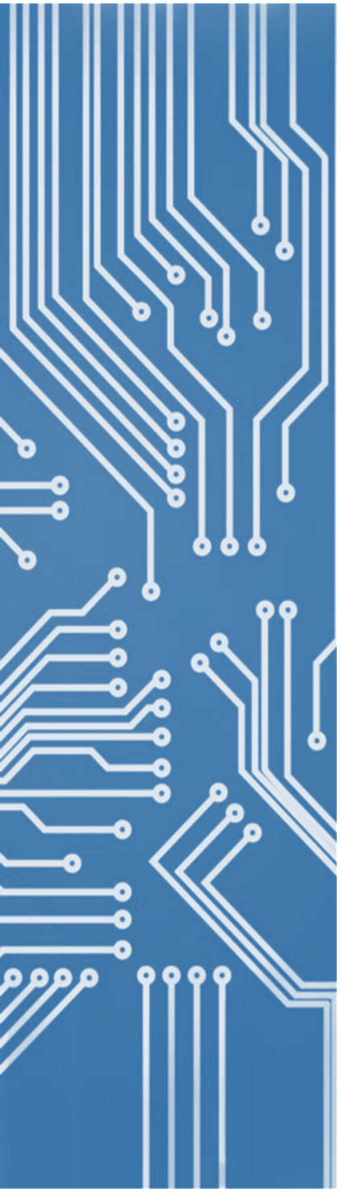
Superscalar microarquitecture
Source: Jason Carey

Note:

❖ Multiple functional units.

❖ More complicated logic.

Superscalar



ILP – Instruction Level Parallelism Wall

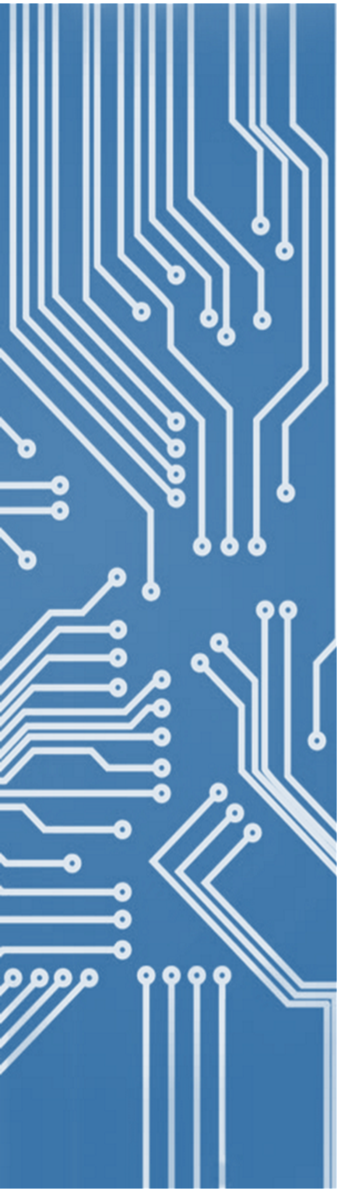
More important concepts…

FETCH	DECODE	EXECUTE	STORE		
FETCH	DECODE	EXECUTE	STORE		
FETCH	DECODE	EXECUTE	STORE		
	FETCH	DECODE	EXECUTE	STORE	
	FETCH	DECODE	EXECUTE	STORE	
	FETCH	DECODE	EXECUTE	STORE	
		FETCH	DECODE	EXECUTE	STORE
		FETCH	DECODE	EXECUTE	STORE
		FETCH	DECODE	EXECUTE	STORE

CPI= 0.33

IPC Instructions Per Cycle = 1/CPI = 3

ILP Instruction Level Parallelism = IPC = 3



ILP – Instruction Level Parallelism Wall

Scalar

Pipeline Stage

FETCH	DECODE	EXECUTE	STORE					
	FETCH	DECODE	EXECUTE	STORE				
		FETCH	DECODE	EXECUTE	STORE			
			FETCH	DECODE	EXECUTE	STORE		
				FETCH	DECODE	EXECUTE	STORE	
					FETCH	DECODE	EXECUTE	STORE

Clock Cycles

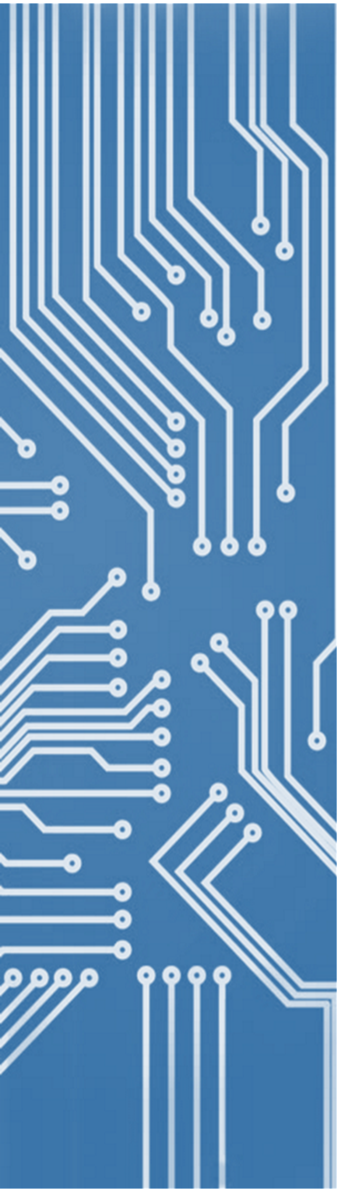
SS + SP

Pipeline Stage

F1	F2	F3	D1	D2	D3	E1	E2	E3	S1	S2	S3															
F1	F2	F3	D1	D2	D3	E1	E2	E3	S1	S2	S3															
	F1	F2	F3	D1	D2	D3	E1	E2	E3	S1	S2	S3														
	F1	F2	F3	D1	D2	D3	E1	E2	E3	S1	S2	S3														
		F1	F2	F3	D1	D2	D3	E1	E2	E3	S1	S2	S3													
		F1	F2	F3	D1	D2	D3	E1	E2	E3	S1	S2	S3													

Clock Cycles

Superscalar + Superpipelined



Teams

3

➢ Given…

FETCH	DECODE	EXECUTE	STORE	FETCH	DECODE	EXECUTE	STORE	FETCH	DECODE	EXECUTE	STORE

Scalar? Pipelined? Superscalar? Superpipelined? Scalar wo/pipeline.

CPI = 4

IPC (ILP) = 0.25

Now consider a Scalar (Pipelined)…

How would the pipeline look with 3 complete instructions executed (consider 4-stage instructions)?

FETCH	DECODE	EXECUTE	STORE		
	FETCH	DECODE	EXECUTE	STORE	
		FETCH	DECODE	EXECUTE	STORE

CPI = 1

IPC (ILP) = 1

Is it Superscalar + ~~Superpipelined?~~

CPI = 0.5

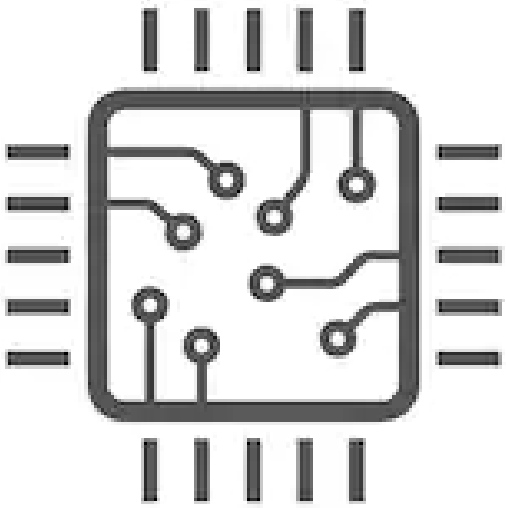
IPC (ILP) = 2

FETCH	DECODE	EXECUTE	STORE		
FETCH	DECODE	EXECUTE	STORE		
	FETCH	DECODE	EXECUTE	STORE	
	FETCH	DECODE	EXECUTE	STORE	
		FETCH	DECODE	EXECUTE	STORE
		FETCH	DECODE	EXECUTE	STORE

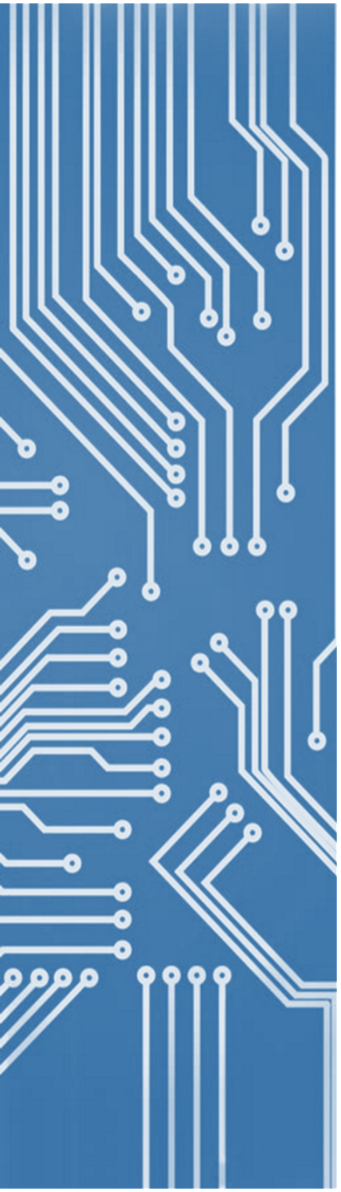
ILP – Instruction Level Parallelism Wall

- Superpiplined =>
  - Frequency => Power => Heat
- Superscalar =>
  - Instruction independence
  - More functional units requiered
  - More logic and intelligence needed
  - Complexity grows exponentially.

How do we use Instruction Level Parallelism?



- CPU does the job.
- At runtime.
- No code needed.
- Special logic implemented in CPU.



Teams 3

- ➢ Comparing scientific application code vs. mainstream application code, which one do you think exhibits more ILP and why?

  ILP Mainstream = 2 - 3

  ILP Scientific > 3

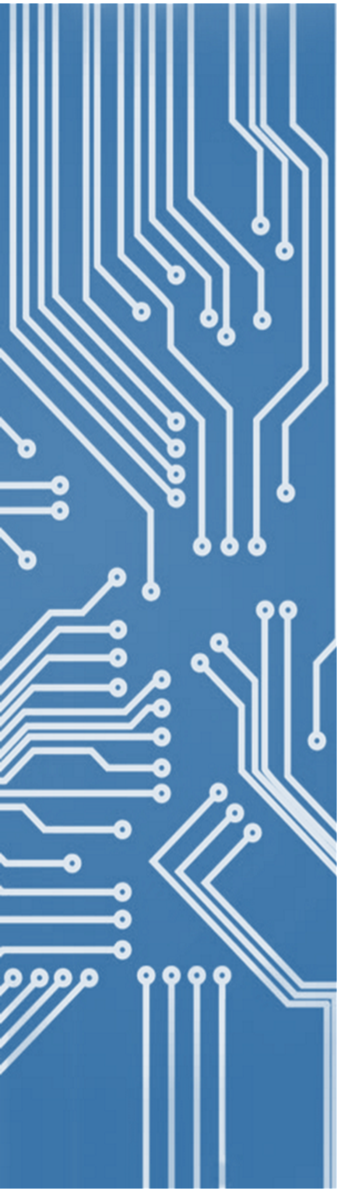
- ➢ What is "Pointer Chasing Code"?

- ➢ What kind of ILP (at most) can we expect from an application using Pointer Chasing Code?

  ILP <= 1

- ➢ Does the programming style of a developer affect parallelism? Explain.

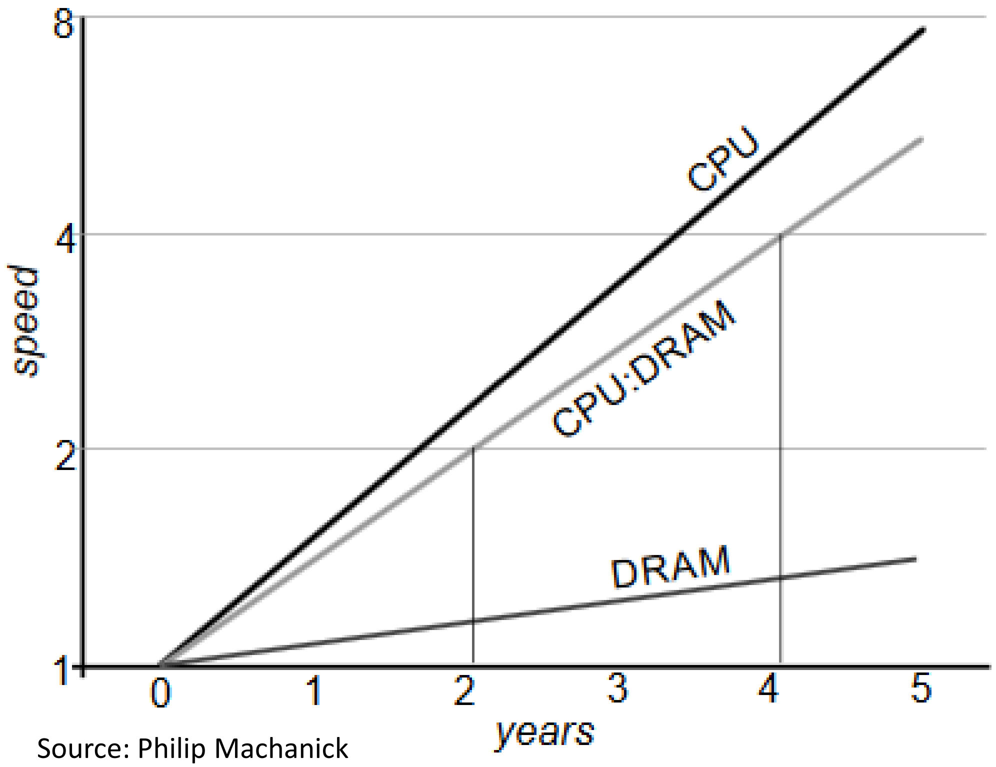
  YES!



How do computers get faster and walls encountered?

- Power Wall
- Frequency Wall
- ILP Wall
- Memory Wall

Memory Wall

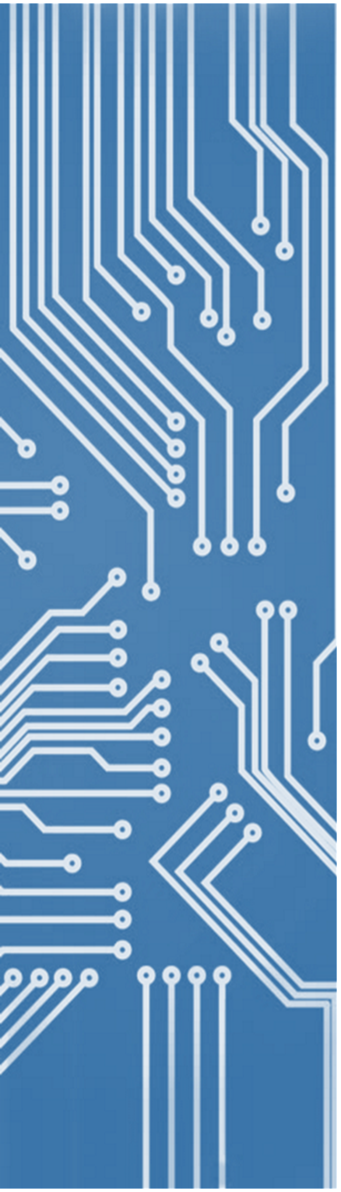


Source: Philip Machanick

CPU

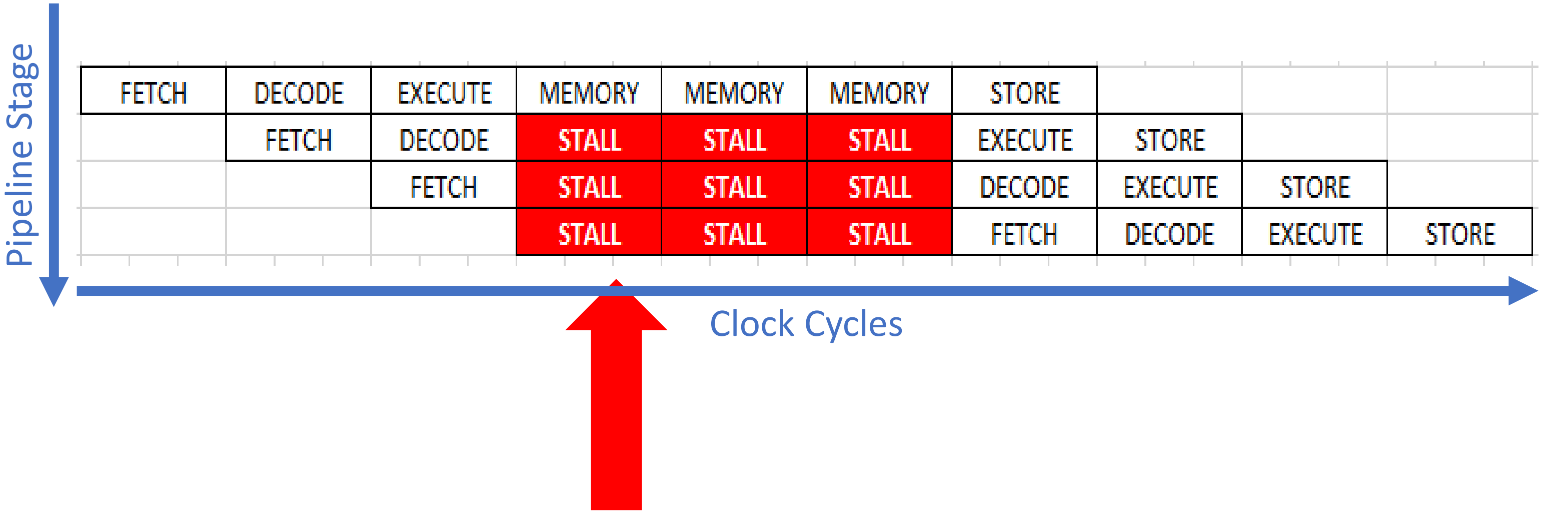
VS.

RAM



Memory Wall

Memory Wall in Pipelines….

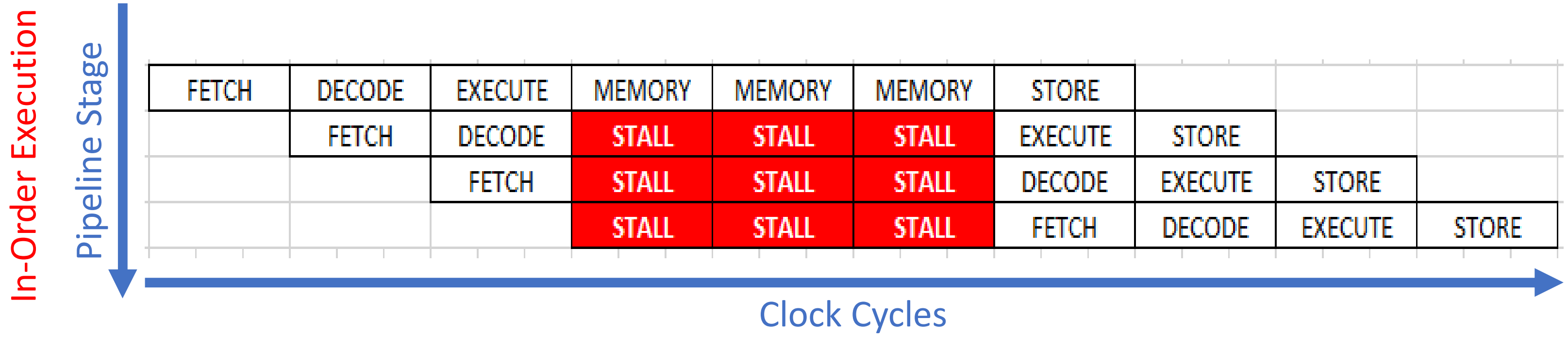


FETCH	DECODE	EXECUTE	MEMORY	MEMORY	MEMORY	STORE			
	FETCH	DECODE	STALL	STALL	STALL	EXECUTE	STORE		
		FETCH	STALL	STALL	STALL	DECODE	EXECUTE	STORE	
			STALL	STALL	STALL	FETCH	DECODE	EXECUTE	STORE

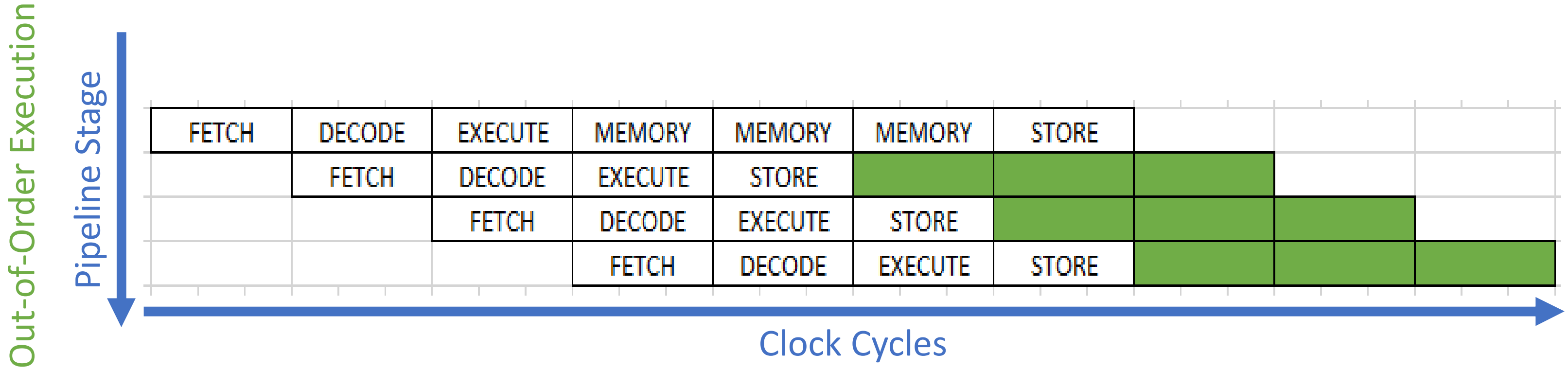
Memory Access stalls the pipeline until value is retrieved.

Memory Wall

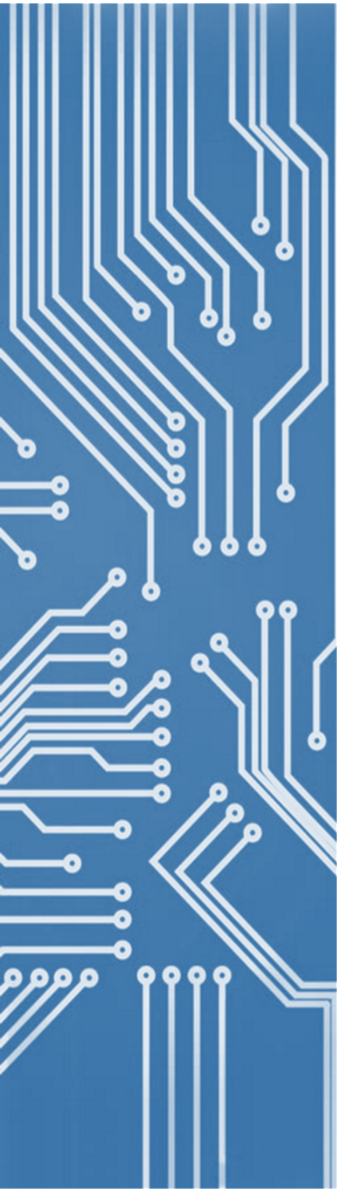
Memory Wall in Pipelines….



FETCH	DECODE	EXECUTE	MEMORY	MEMORY	MEMORY	STORE			
	FETCH	DECODE	STALL	STALL	STALL	EXECUTE	STORE		
		FETCH	STALL	STALL	STALL	DECODE	EXECUTE	STORE	
			STALL	STALL	STALL	FETCH	DECODE	EXECUTE	STORE



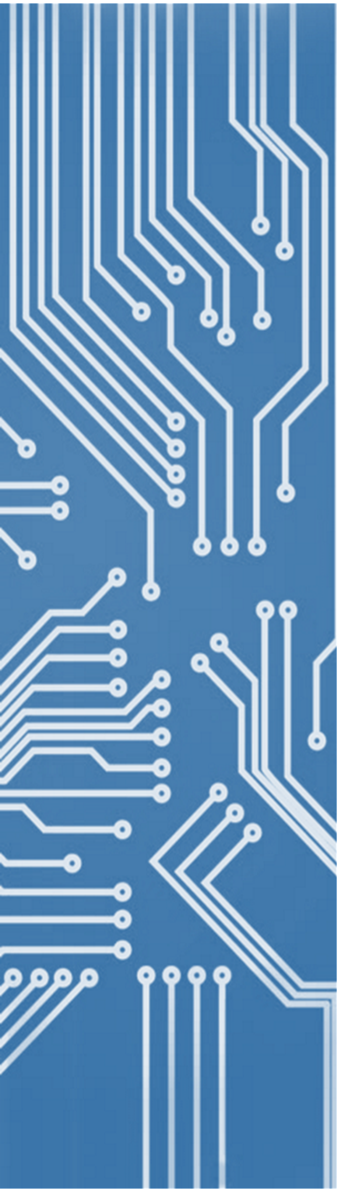
FETCH	DECODE	EXECUTE	MEMORY	MEMORY	MEMORY	STORE			
	FETCH	DECODE	EXECUTE	STORE	🟩	🟩	🟩		
		FETCH	DECODE	EXECUTE	STORE	🟩	🟩	🟩	
			FETCH	DECODE	EXECUTE	STORE	🟩	🟩	🟩



Memory Wall

- Even with advances
  - DRAM Speed.
  - DRAM organization.
  - Improved aproaches to memory hierarchy.
  - Out-of-order execution.
    - Even more complex logic.
    - HW Implementation.

We hit the memory wall!

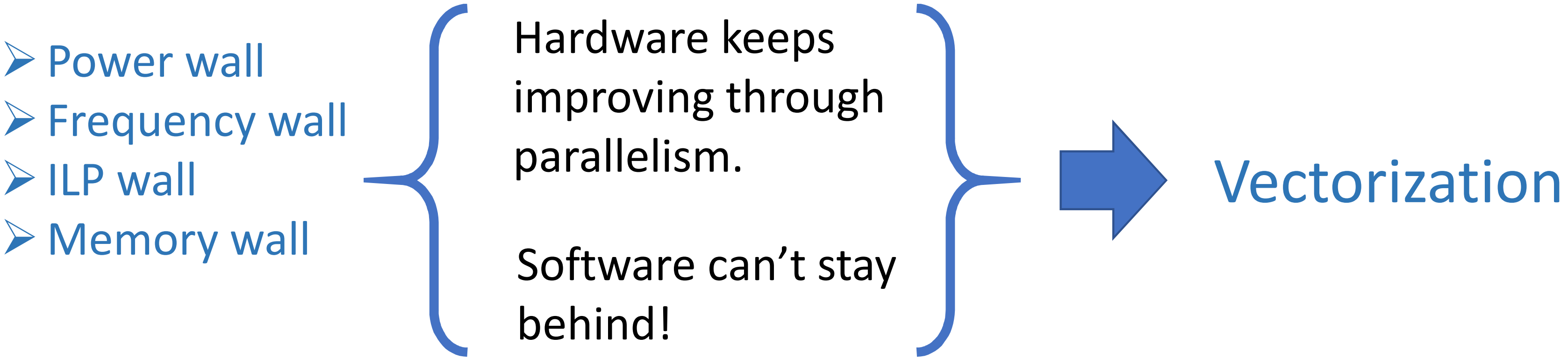


How can we improve performance?

Threads, as seen in previous lectures, are an option…

, but….

they are NOT the only option…



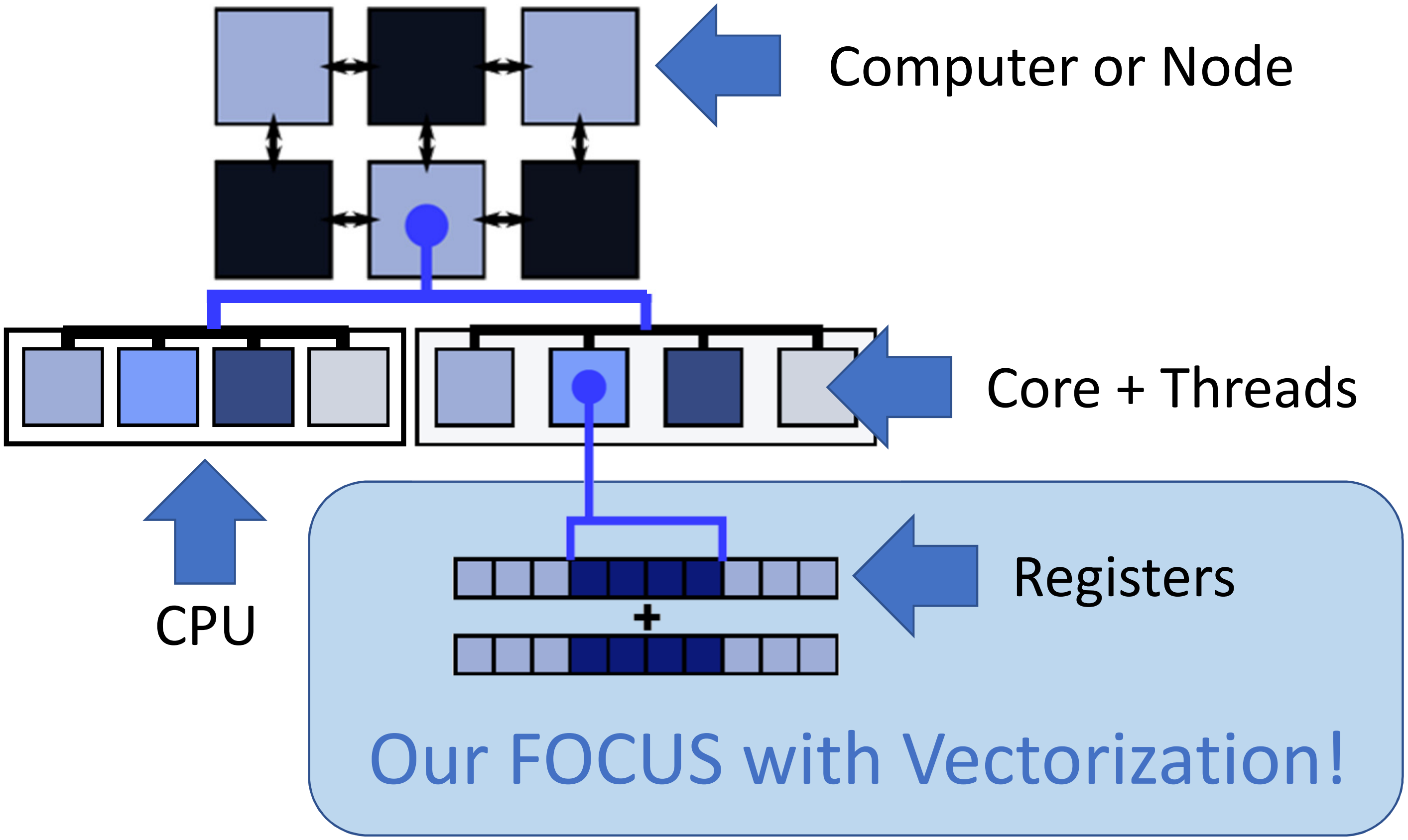
- Power wall
- Frequency wall
- ILP wall
- Memory wall

Hardware keeps improving through parallelism.

Software can’t stay behind!

Vectorization

Where we stand?



Computer or Node

Core + Threads

CPU

Registers

Our FOCUS with Vectorization!

Thank you!