CPSC 313: Computer Hardware and Operating Systems

Unit 2: Pipelining

Other Forms of Parallelism

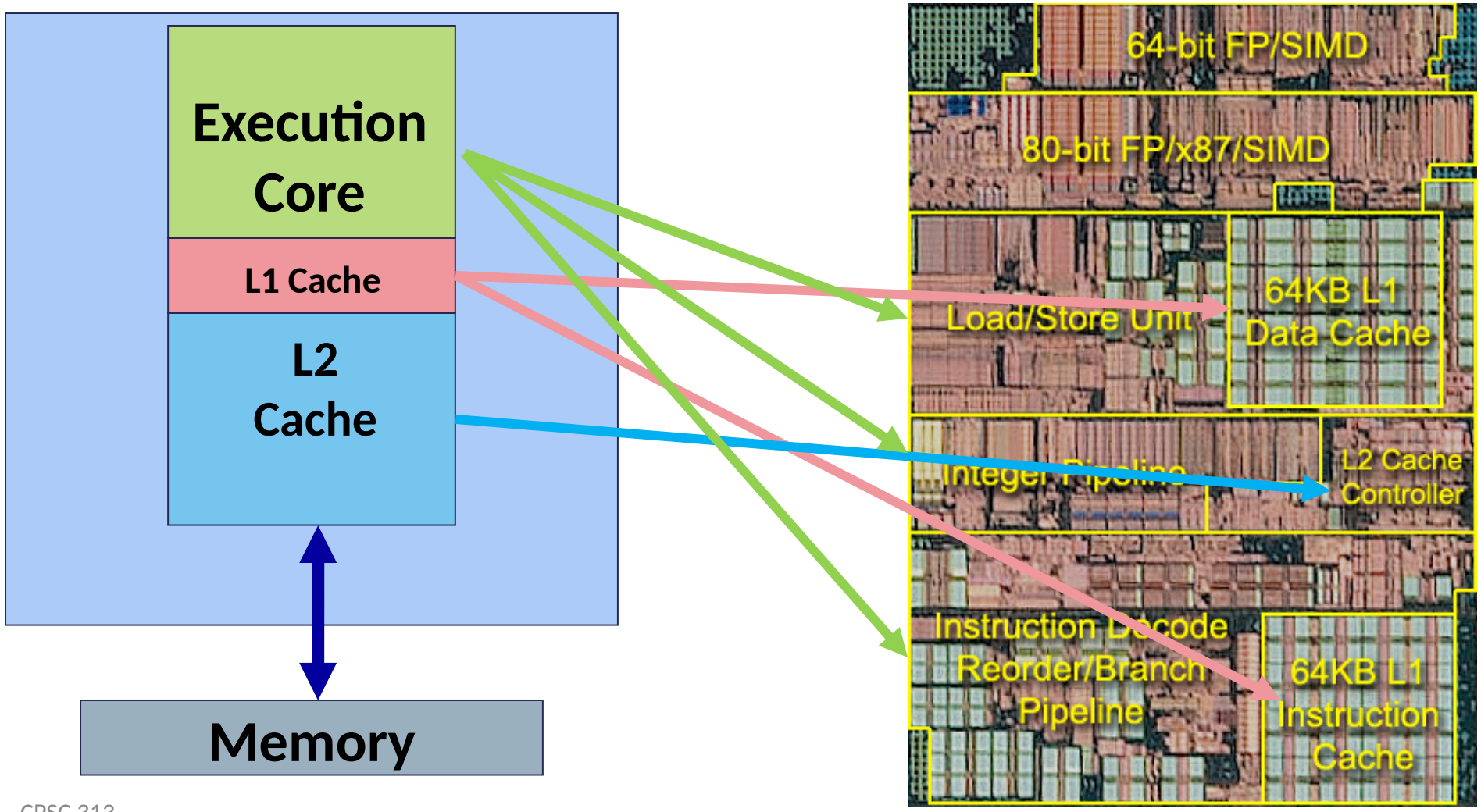
Today

- Learning outcomes
  - Place pipelining in the context of the many forms of hardware parallelism
- How we'll get there
  - Talk about how we slice and dice processors to get different forms of parallelism.
  - Talk about different models of parallel computation and how they map to different kinds of hardware.

Administration

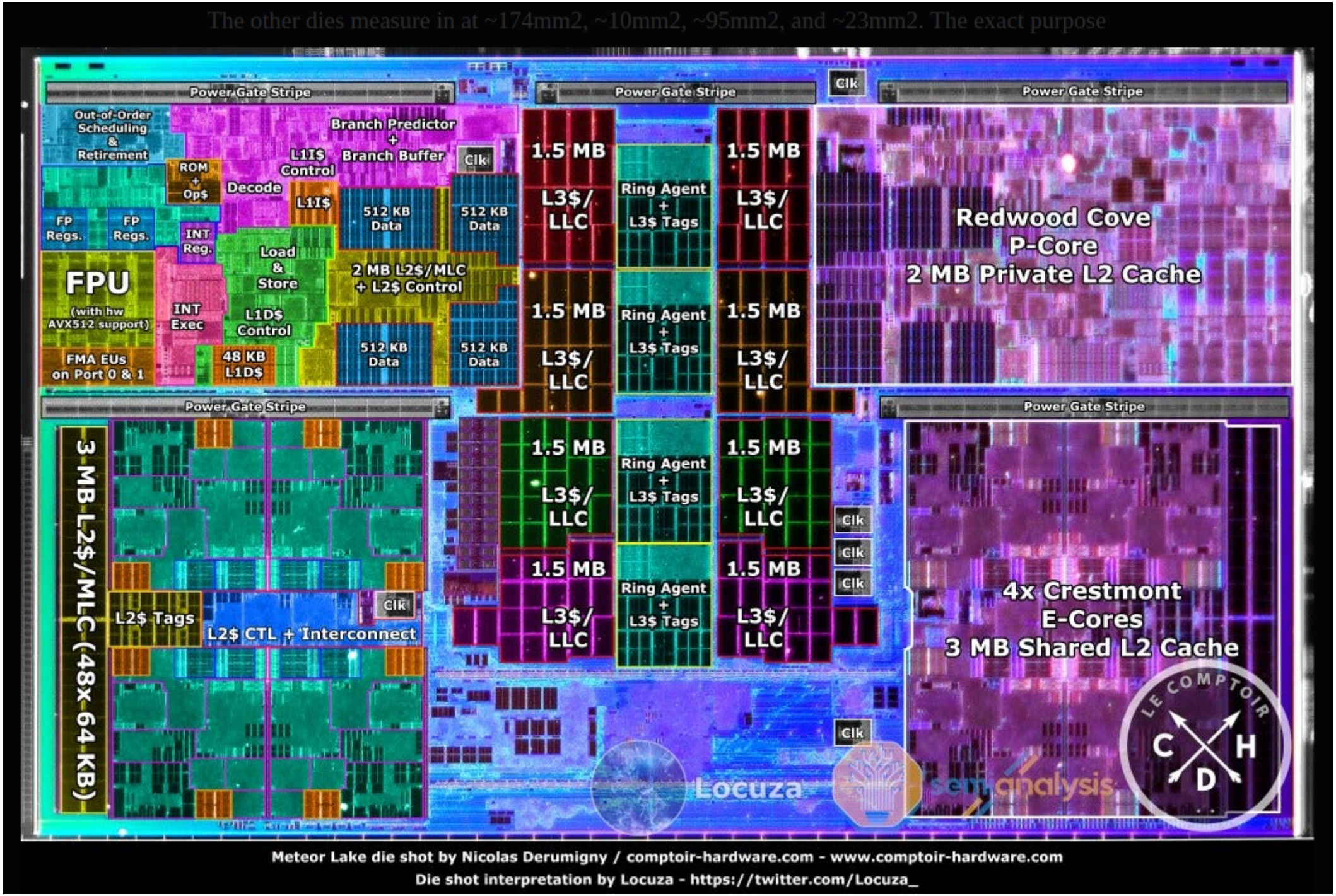
- Quiz 2: Don’t miss your time!
  - Quiz 2 information and practice quiz were released on Friday.
  - This lecture is not assessable on Quiz 2 (but it is on Quizzes 3-5 + the Final).
- Lab 5:
  - Due Sunday October 20th (in **two** weeks).
- No class Friday because of Quiz 2
  - but we’ll be here to answer questions.

What is really inside a single chip?

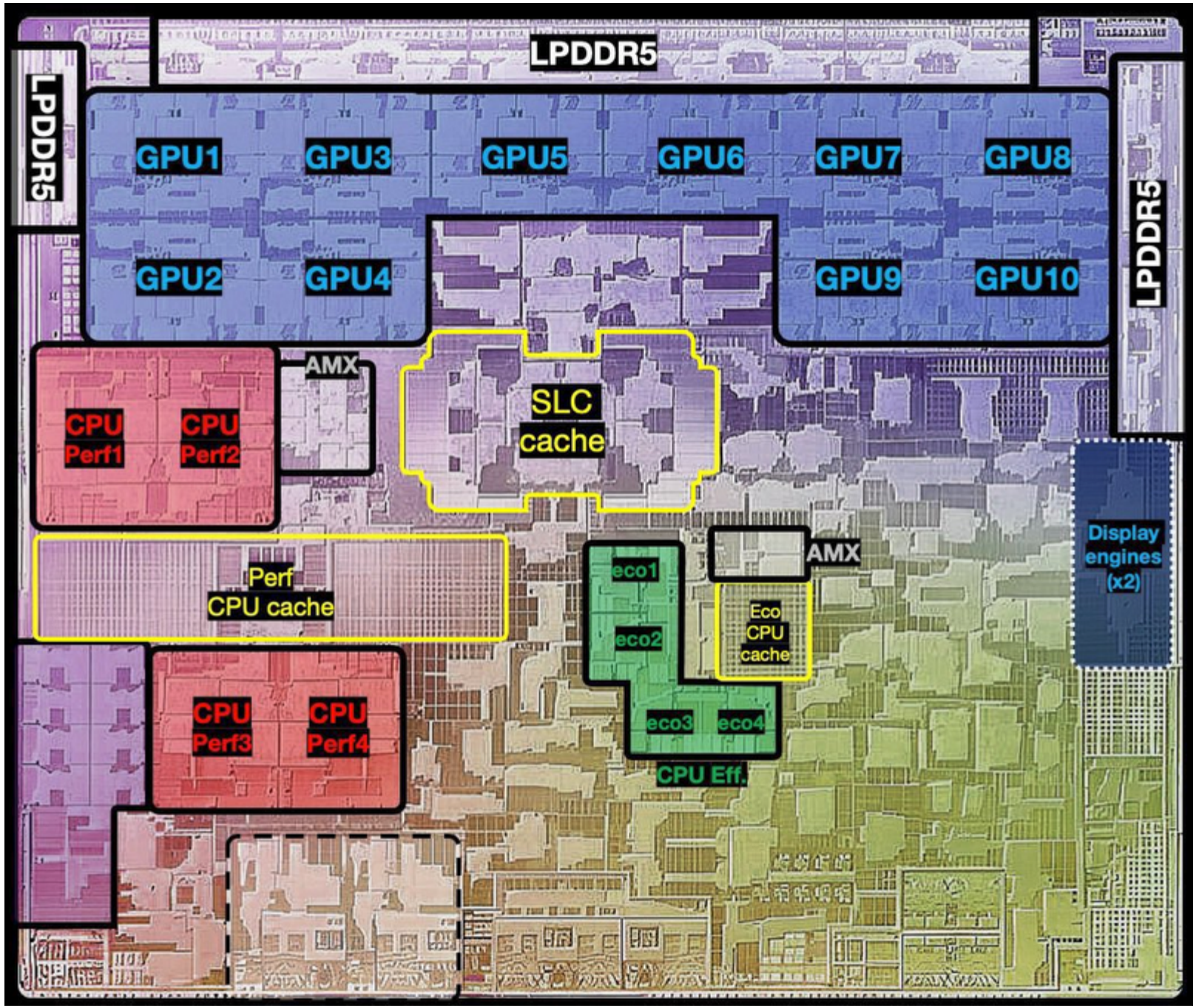


L2 controller is here,
But the actual cache is
On a separate die.

Intel Meteor Lake Die Map



Apple M4 Die Map



Apple M-series Comparison

Apple M-Series (Vanilla) SoCs			
SoC	M4	M3	M2
CPU Performance	4-core	4-core 16MB Shared L2	4-core (Avalanche) 16MB Shared L2
CPU Efficiency	6-core	4-core 4MB Shared L2	4-core (Blizzard) 4MB Shared L2
GPU	10-Core Same Architecture as M3	10-Core New Architecture - Mesh Shaders & Ray Tracing	10-Core 3.6 TFLOPS
Display Controller	2 Displays?	2 Displays	2 Displays
Neural Engine	16-Core 38 TOPS (INT8)	16-Core 18 TOPS (INT16)	16-Core 15.8 TOPS (INT16)
Memory Controller	LPDDR5X-7500 8x 16-bit CH 120GB/sec Total Bandwidth (Unified)	LPDDR5-6250 8x 16-bit CH 100GB/sec Total Bandwidth (Unified)	LPDDR5-6250 8x 16-bit CH 100GB/sec Total Bandwidth (Unified)
Max Memory Capacity	24GB?	24GB	24GB
Encode/ Decode	8K H.264, H.265, ProRes, ProRes RAW, AV1 (Decode)	8K H.264, H.265, ProRes, ProRes RAW, AV1 (Decode)	8K H.264, H.265, ProRes, ProRes RAW
USB	USB4/Thunderbolt 3 ? Ports	USB4/Thunderbolt 3 2x Ports	USB4/Thunderbolt 3 2x Ports
Transistors	28 Billion	25 Billion	20 Billion
Mfc. Process	TSMC N3E	TSMC N3B	TSMC N5P

Our pipeline is very much simplified …

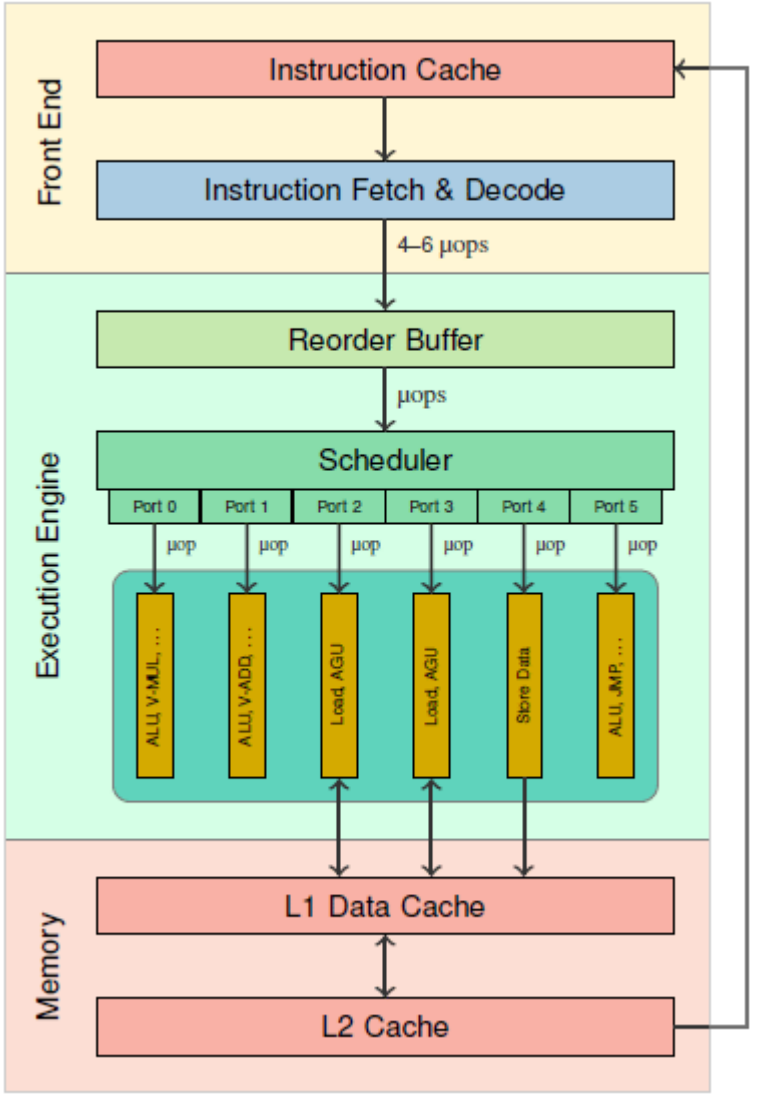
- Not all ALU operations take a single cycle
  - Division is particularly difficult
- There can be multiple Functional Units
  - More than one adder/multiplier/divider/whateverer
- Super Scalar architectures
  - Issue more than one instruction per cycle

Our pipeline is very much simplified …

- Out-Of-Order Execution and other tricks
  - Reorder instructions so they execute faster
  - Move memory reads earlier or dependent instructions later
  - Renaming registers on the fly to avoid “output” dependencies
- Memory accesses can take many clock cycles; not all accesses take the same time.

What a real Intel execution unit looks like …

- How instructions are executed:
  - Multiple instructions are fetched at once
  - Instructions can be reordered; registers can be renamed to avoid name conflicts.
  - Instructions wait in the scheduler (also called reservation station) until operands are ready.
  - Then they are sent through one of multiple execution units.



Limits to the pipeline

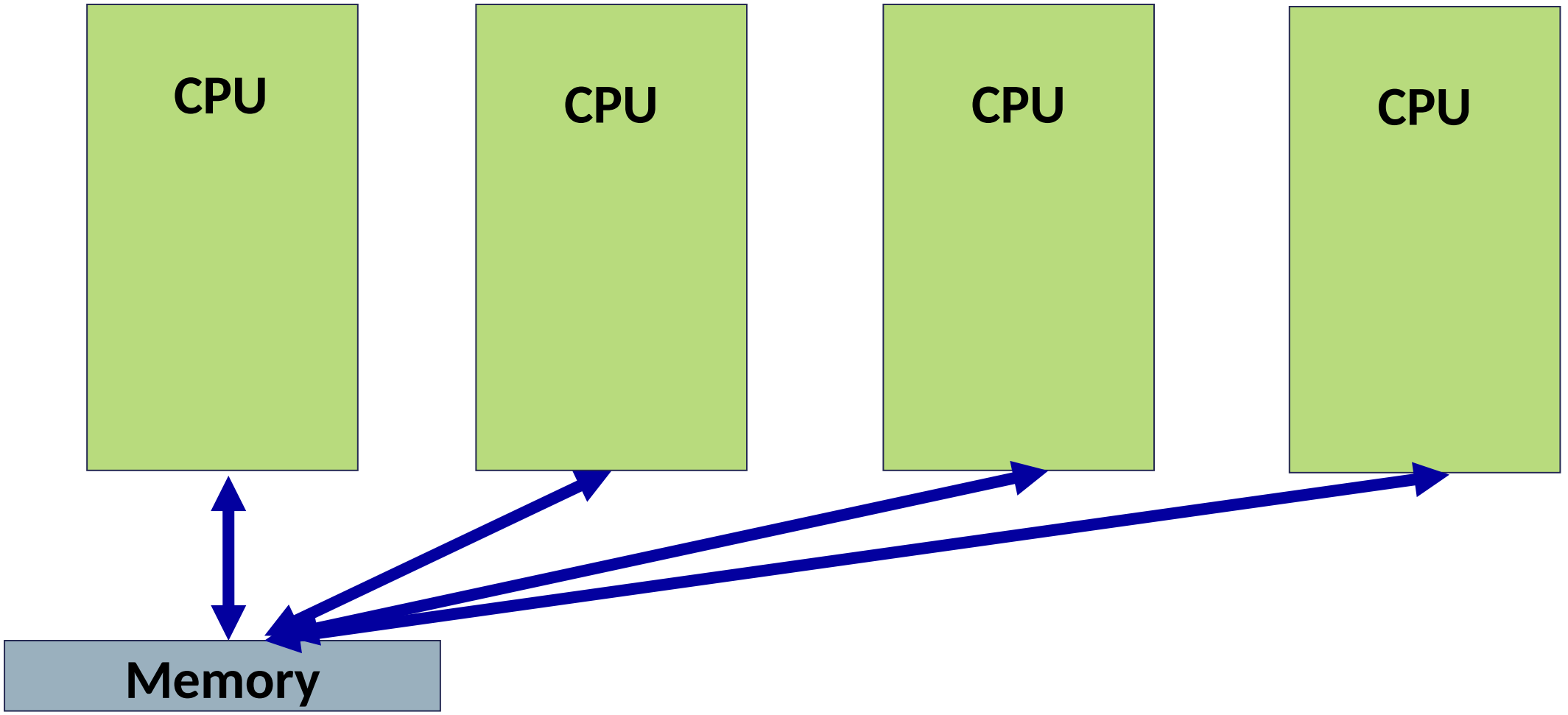
- clock frequency … as high as possible
  - pipeline depth
  - heat
  - clock skew
- cpi … as low as possible
  - pipeline tricks
  - amount of ILP (instruction-level parallelism)
  - 1?
- Other types of parallelism?

Some Other Kinds of Parallelism

- **Multitasking (or multiprogramming)**:
  - This is basically an illusion.
  - Software makes it look like many things are happening at once, but quite likely the operating system is simply rapidly switching among many different jobs.

- **Multiprocessing**:
  - Place multiple processing units (CPUs) on a machine so you can run different programs on each unit.

Multiprocessing



Some Other Kinds of Parallelism

- Parallel runtimes that can leverage multiprocessing or other forms of parallelism:
  - The parallelism can be figured out either manually or automatically.
  - Then you need a runtime system to dispatch the different tasks to different processing elements
- **Multithreading**: one way to implement a parallel runtime
  - Again, the idea is to allow different threads to run on different processing units.
  - But you can have multiple software threads that still run on a single HW core.

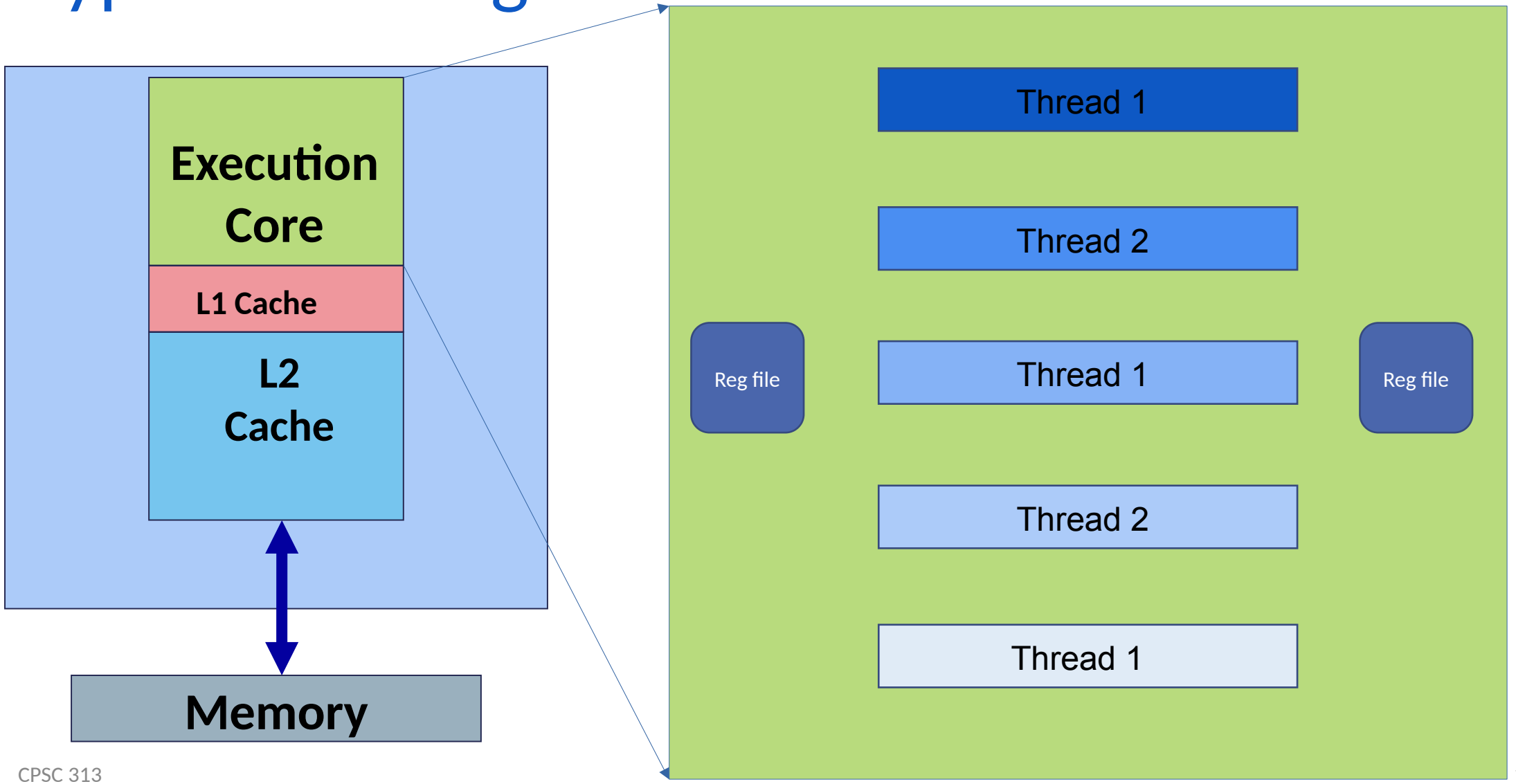
Thread-Level Parallelism

- Programmers Express this Explicitly
  - requires software API or language support
  - threads or other things (async/await …)
- Granularity
  - coarse grain, limited by fixed overheads (create, schedule, etc)
- Exploited by
  - multi-core processors (ubiquitous)
    - also multi-chip processors (less common and expensive)
  - also … asynchronous communication with other devices

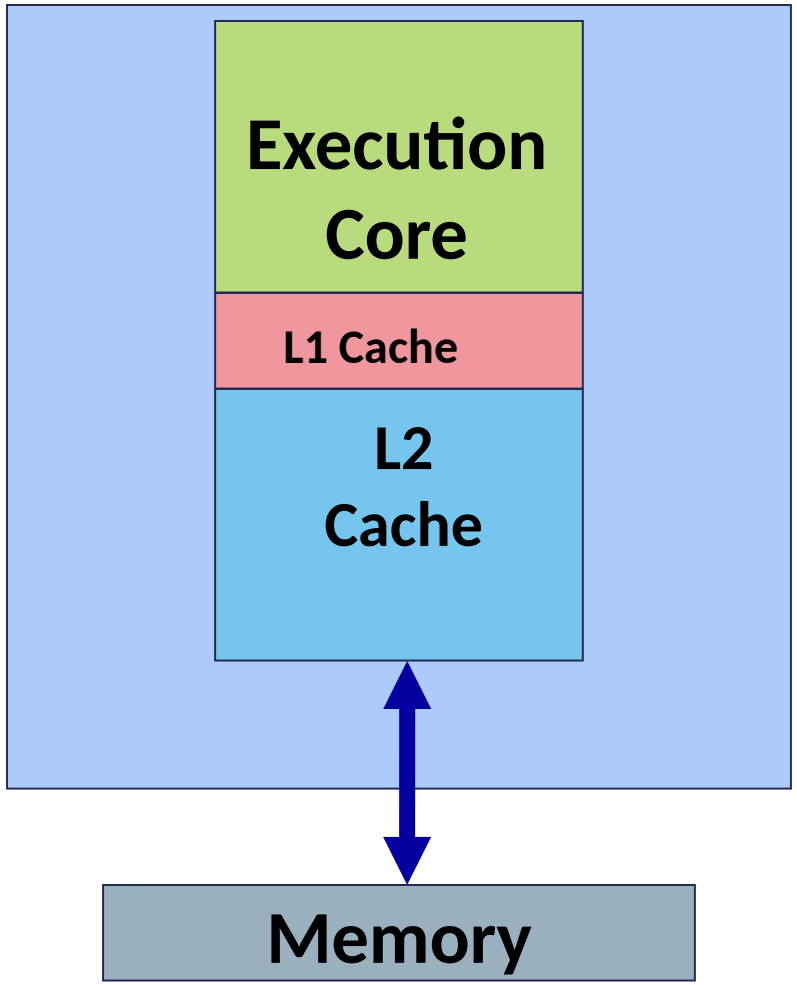
Hyper Threading

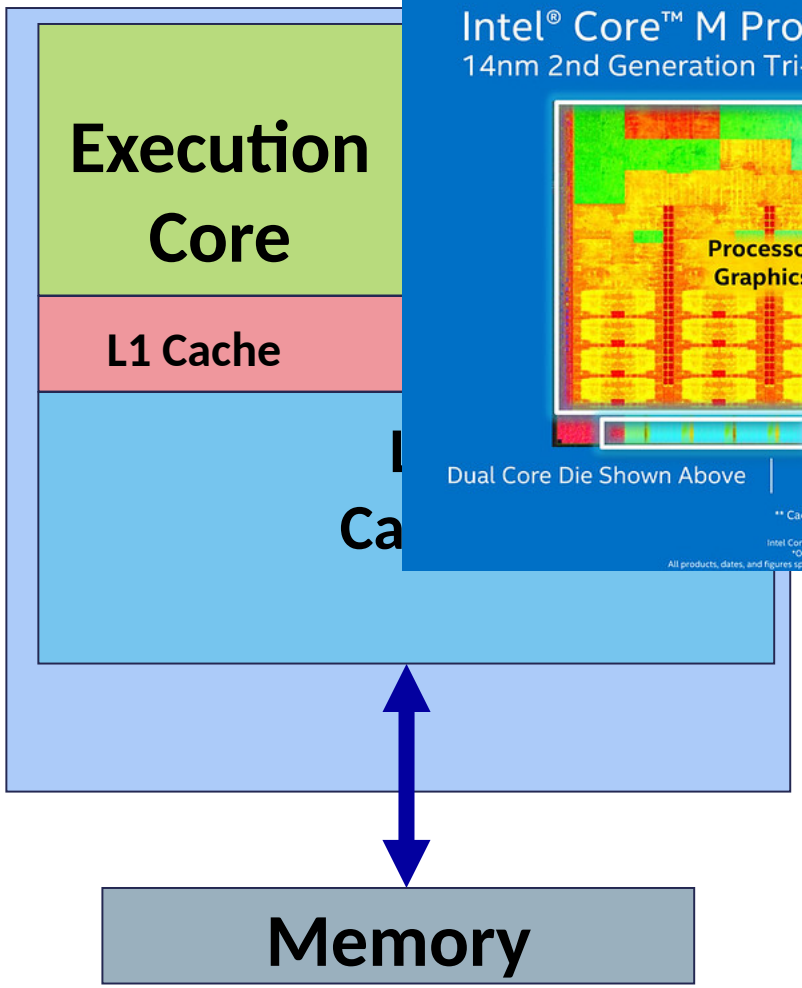
- Memory latency is a problem (That M stage is expensive even when it's cheap!)
- Fast, hardware-implemented, thread switch while waiting for memory
  - similar to software thread waiting for disk read, for example
- Each has N hyper-threads (usually 2)
  - implemented by having N copies of all registers
  - exploited by virtual cores (# cores X # of hyper threads)

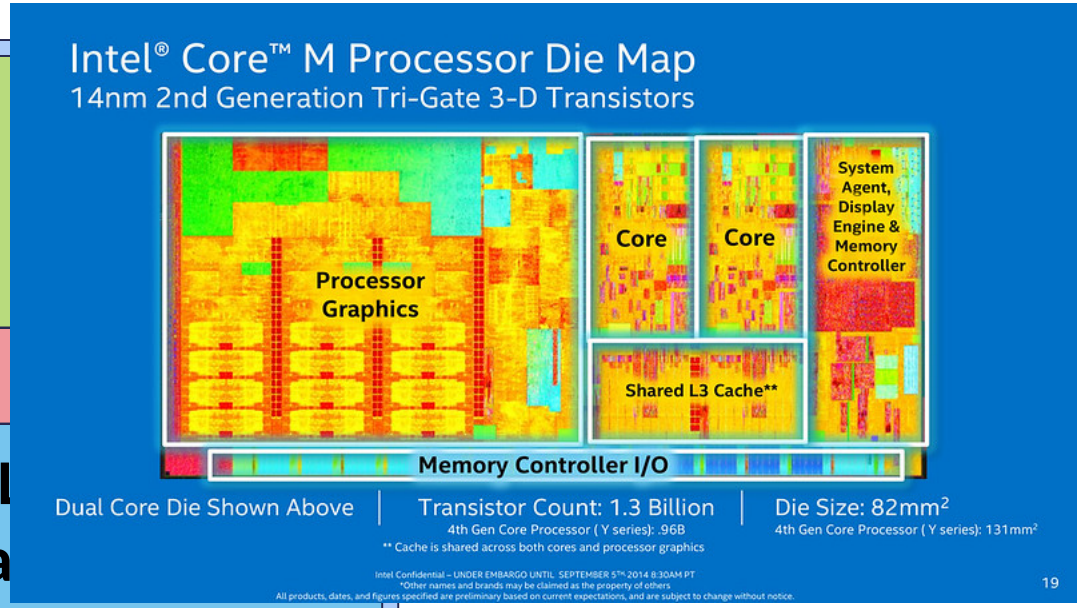
Hyperthreading CPU



From Multiprocessing to Multicore







Distributed Systems

- A data centre, for example
- Multiple machines communicating over a network
  - they don’t share memory … each machine runs a different program
  - each machine has its own OS, managing threads locally
- Distributed ensemble is organized using higher-level API’s
  - standard or roll your own
- Compared to multi-core?

Still Other Types of Parallelism

- What about matrix multiply?
  - fine-grain parallelism: one instruction applied to many values in ||
  - Intel AVX, vector instructions
- What about machine learning networks?
  - and other, massive, fine-grain, one-function-to-multiple data things
  - video games to the rescue … GPUs

Flynn's Taxonomy: xIxD for x = M(ultiple)/S(ingle)

		Data Stream	
		Single	Multi
Instruction Stream	Single	SISD	SIMD
	Multi	MISD	MIMD

Flynn's Taxonomy: xIxD for x = M(ultiple)/S(ingle)

- Conventional Uniprocessor: Single Instruction Single Data (SISD)
  - A single execution unit fetches a single instruction and executes it to completion.
  - No parallelism
- Multicore/Multiprocessors: Multiple Instruction Multiple Data (MIMD)
  - Each core/processor has a stream of instructions
  - Each instruction stream operates on its own data

More Exotic Architectures

- Single Instruction, Multiple Data (SIMD)
  - You apply the same set of instructions to many data items in parallel
  - Can you think of applications where this makes sense?

- Multiple Instruction, Single Data (MISD)
  - This time, different functional units do different things to the same data.
  - These are less common structures.

Limits

- We have to know how to build it
- We have to actually be able to build it
- We have to be able to afford to build it
- Much more potential parallelism than we can exploit right now
  - neural networks
  - specialized hardware
  - quantum computers

Summary

- Hardware for Parallelism
  - pipelines
  - super scalar
  - hyper threading
  - multi core
  - networks
- Software
  - compilers
  - threads, async/await
  - massively scalable data-structures and frame works etc
  - using language embedding OR APIs