Apple Silicon

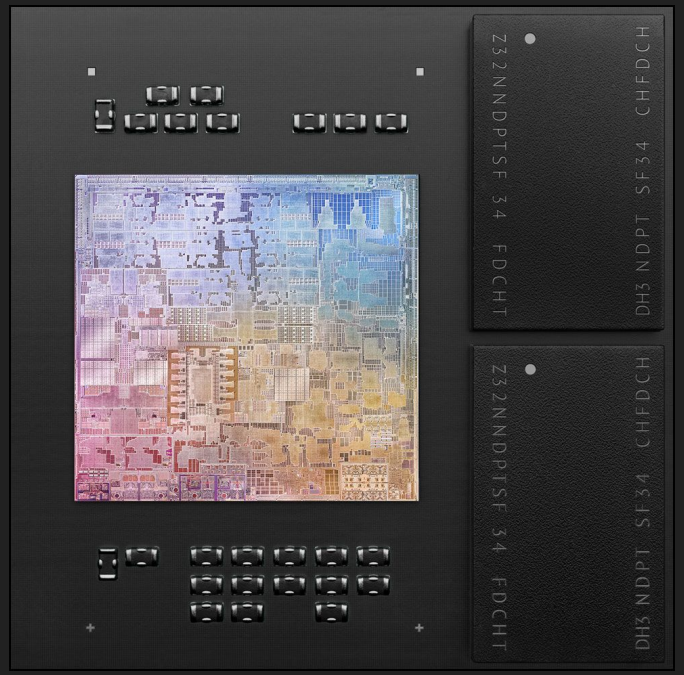
Fan

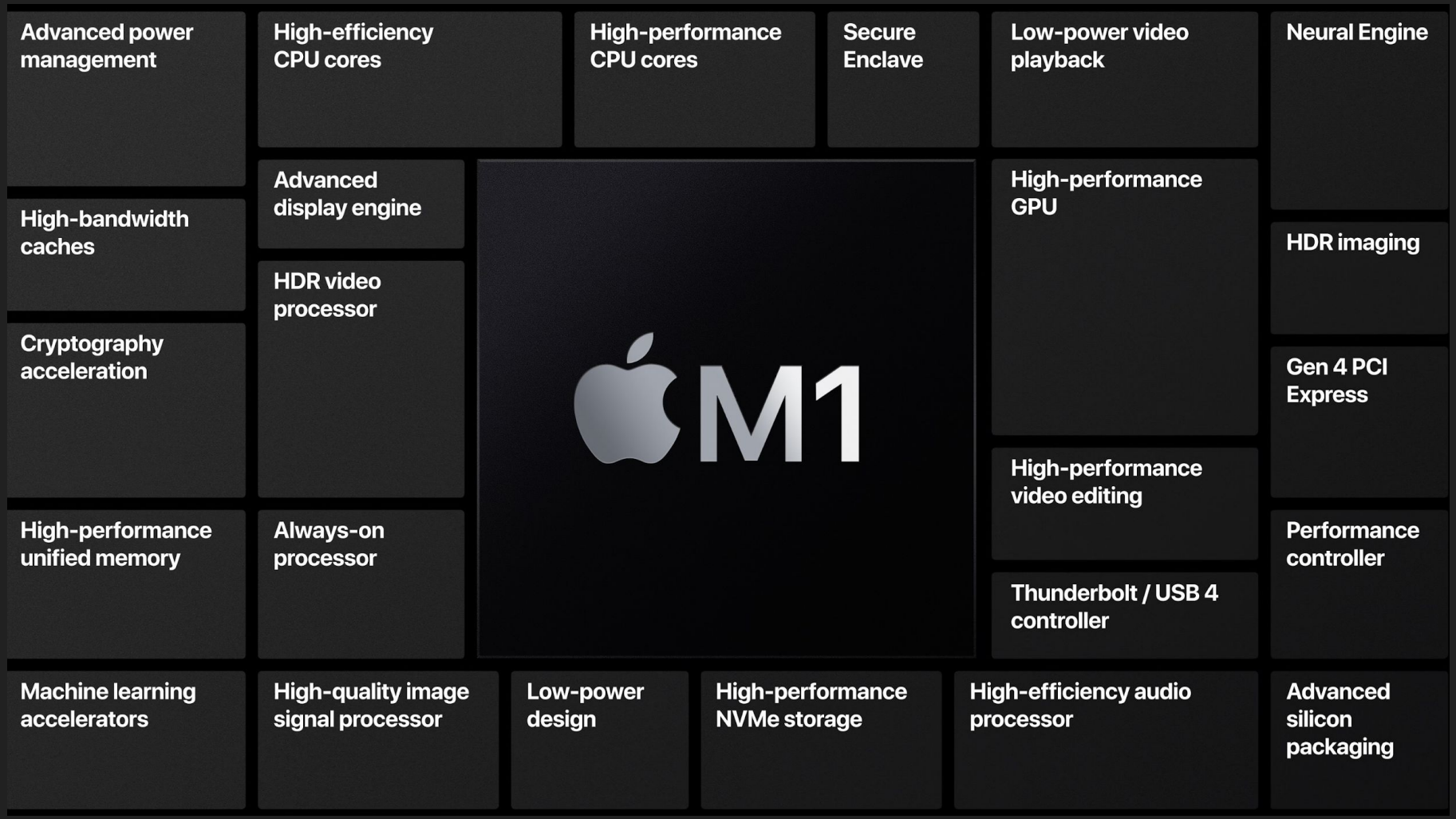
01/06/2021


"People who are really serious about software
should make their own hardware."
– Alan Kay

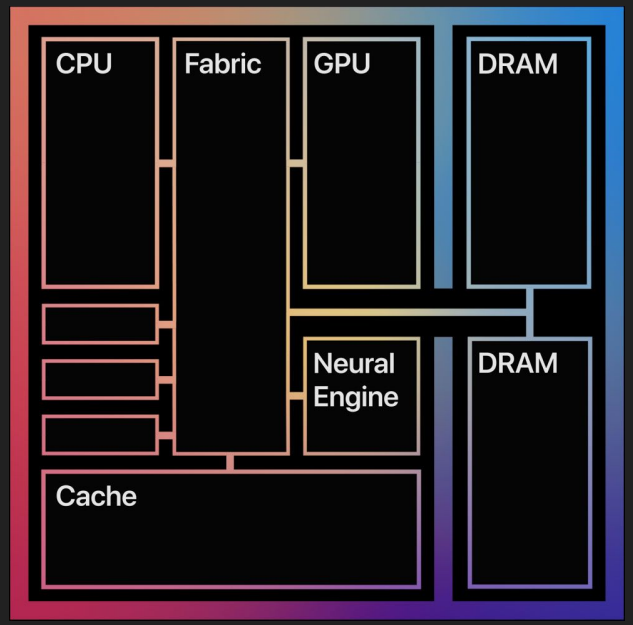
Agenda

- An overview of Apple M1 SoC
  - Micro-architecture
  - Compatibility (from x86 to ARM)
- Why is Apple M1 so fast?
  - The cores
  - Unified Memory Architecture (UMA)
  - Rosetta 2 with Dual memory models
  - Apple Neural Engine (ANE)
- Security
  - Secure Enclave (T2 style)



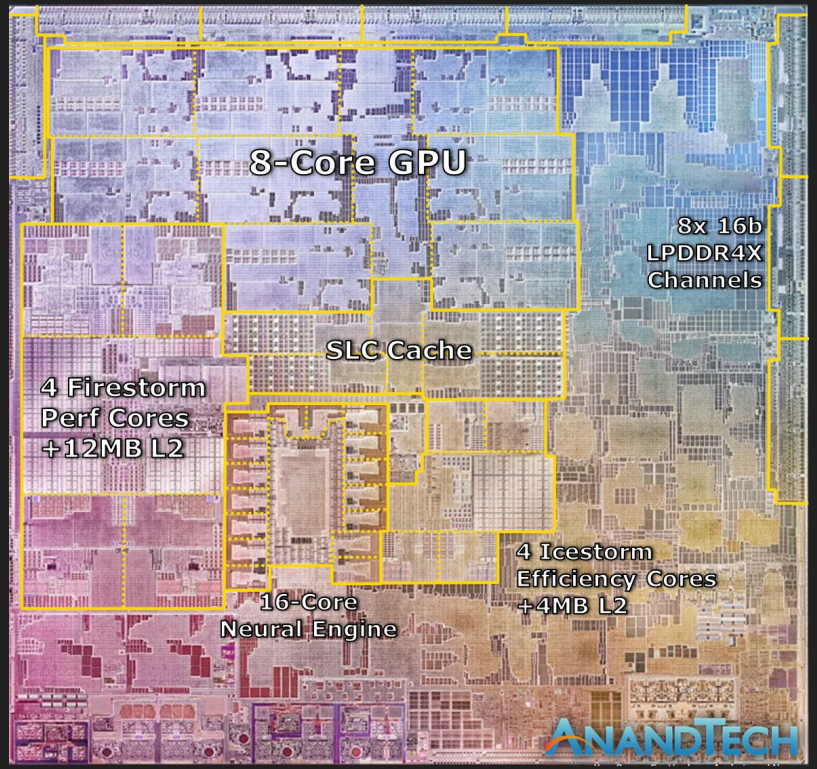

Advanced power management
High-efficiency CPU cores
High-performance CPU cores
Secure Enclave
Low-power video playback
Neural Engine
High-bandwidth caches
Advanced display engine
High-performance GPU
HDR imaging
HDR video processor
Cryptography acceleration
M1
Gen 4 PCI Express
High-performance video editing
High-performance unified memory
Always-on processor
Performance controller
Thunderbolt / USB 4 controller
Machine learning accelerators
High-quality image signal processor
Low-power design
High-performance NVMe storage
High-efficiency audio processor
Advanced silicon packaging

The Apple M1 SoC: An A14X for Macs



- First System on a Chip (SoC) for the Mac
- Unified memory architecture
- 8-core CPU
- 8-core GPU
- 16-core Apple Neural Engine
- Secure Enclave

Micro-architecture



- Frees up die space by keeping the LPDDR4X DRAM close at hand
- 4 Firestorm performance cores
- 4 Icestorm efficiency cores
- Big GPU die size

Compatibility (from x86 to ARM)

Universal apps

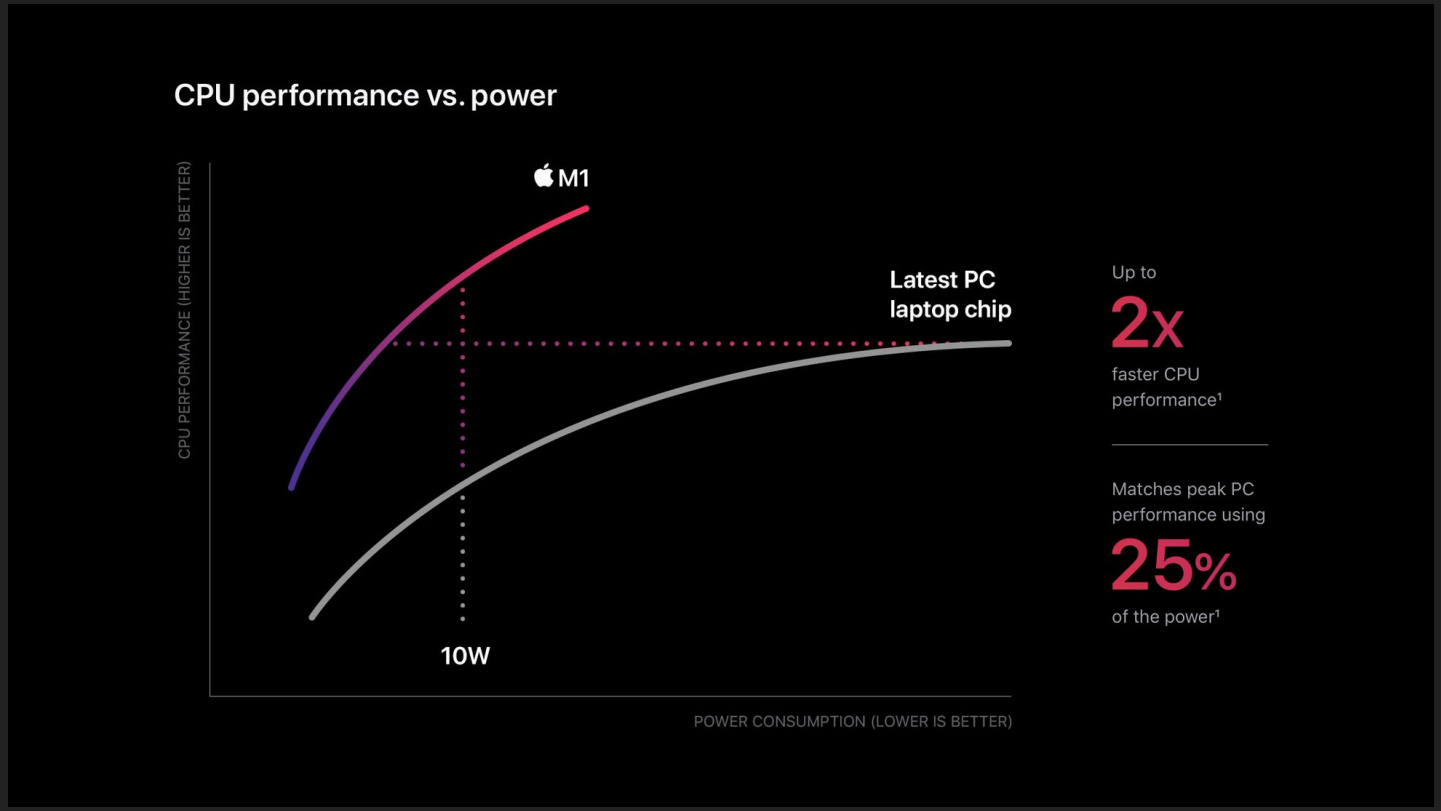
Universal apps support both Intel-based and Apple silicon–based Mac systems.

Rosetta 2

A binary translation software allows most x86 programs to be able to execute after an initial translation step.

iPhone and iPad apps can directly run on Mac

Why is Apple M1 so fast?



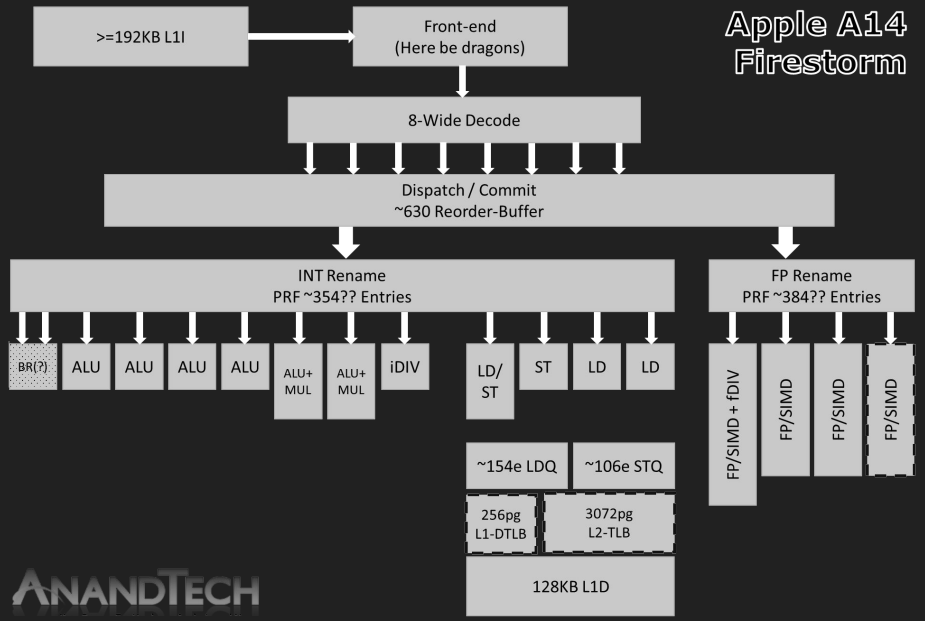
Firestorm Cores (starting A14)

Beat most Intel cores and almost beats the fastest AMD Ryzen cores

- Perform more instructions in a sequence faster.
- Perform lots of instructions in parallel.

Refill the instruction buffer quickly relies on decoding code instruction into micro-ops.

- 4 decoders (Intel and AMD) vs.
- 8 decoders (Apple)



Firestorm Cores (starting A14)

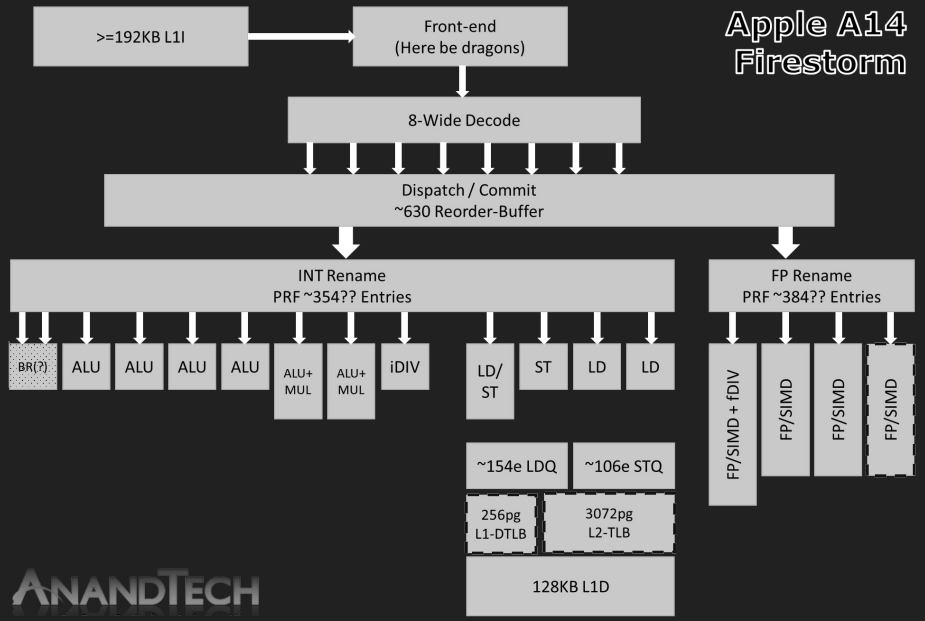
A win for RISC vs. CISC

- Various length x86 instruction (1-15 bytes)
- Fixed length ARM instruction (4 bytes)

Intel and AMD attempt to decode instructions at every possible starting point

- Convoluted and complicated decoder stage
- Hard to add more

Twice as many instructions as AMD and Intel CPUs at the **same clock frequency**

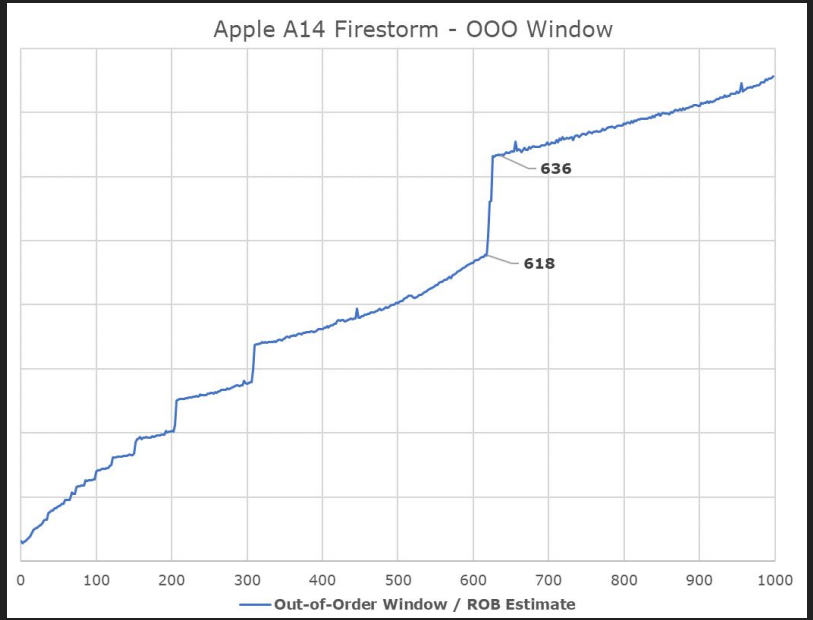


Firestorm Cores (starting A14)

Huge out-of-order window

- Apple - 630 ROB
- Intel Sunny Cove and Willow Cove - 352 ROB
- AMD Zen3 - 256 ROB
- Arm Cortex-X1 - 224 ROB

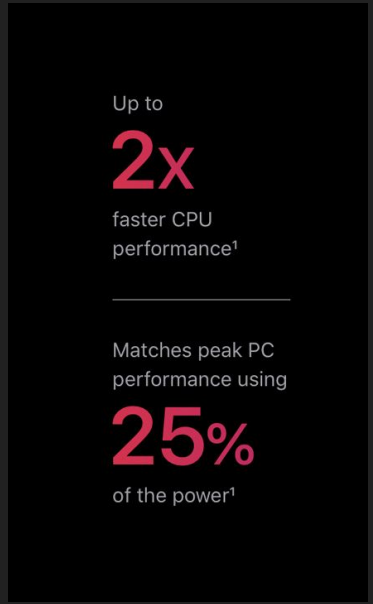
High ILP (Instruction level-parallelism) with many, many Execution Units.



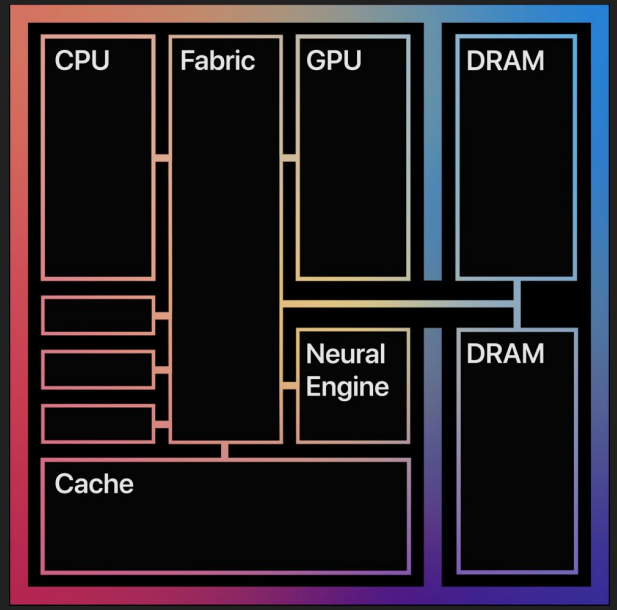
Firestorm Cores (starting A14)

Others

- 256 pages L1 TLB and 3072 pages L2 TLB
- Massive 192KB instruction cache
- Fast L1D with 3-cycle load-use latency
- A shared huge 12MB L2 cache
- 4 128-bit SIMD units
- etc.



Unified Memory Architecture (UMA)



A single pool that's accessible by any portion of the processor

- Closer to components
- No need to copy data around
- Access at the same memory address
- Dynamic distribution of memory (e.g. GPU vs CPU)
- 8x 16-bit LPDDR4X-4266 offer 68.25GB/s memory bandwidth

Rosetta 2

Ahead-of-time binary translation system

- About 70-80% of native performance

Memory-ordering could be the biggest hurdle

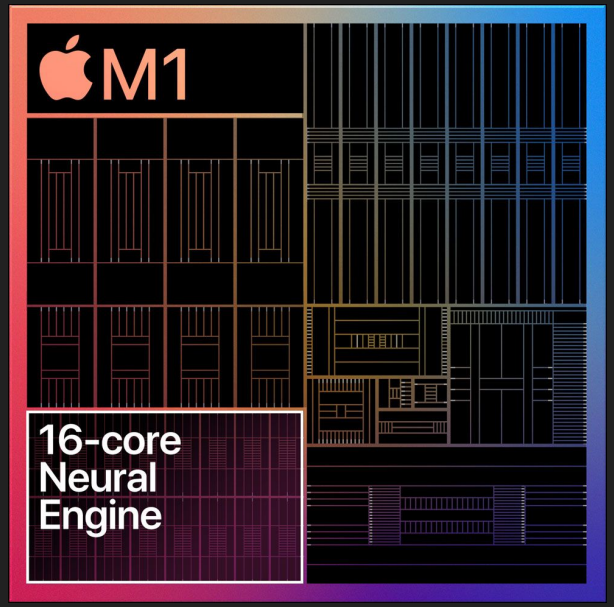
- Intel strong model vs. ARM weaker model
- Microsoft's emulation of x86 on Arm-based Surface laptops

In package Intel memory model

- Switch CPU memory mode to Intel when running translated x86

	Loads Reordered After Loads?	Loads Reordered After Stores?	Stores Reordered After Stores?	Stores Reordered After Loads?	Atomic Instructions Reordered With Loads?	Atomic Instructions Reordered With Stores?	Dependent Loads Reordered?	Incoherent Instruction Cache/Pipeline?
Alpha	Y	Y	Y	Y	Y	Y	Y	Y
AMD64				Y				
IA64	Y	Y	Y	Y	Y	Y		Y
(PA-RISC)	Y	Y	Y	Y				
PA-RISC CPUs								
POWER™	Y	Y	Y	Y	Y	Y		Y
(SPARC RMO)	Y	Y	Y	Y	Y	Y		Y
(SPARC PSO)			Y	Y		Y		Y
SPARC TSO				Y				Y
x86				Y				Y
(x86 OOStore)	Y	Y	Y	Y				Y
zSeries®				Y				Y

Apple Neural Engine (ANE)



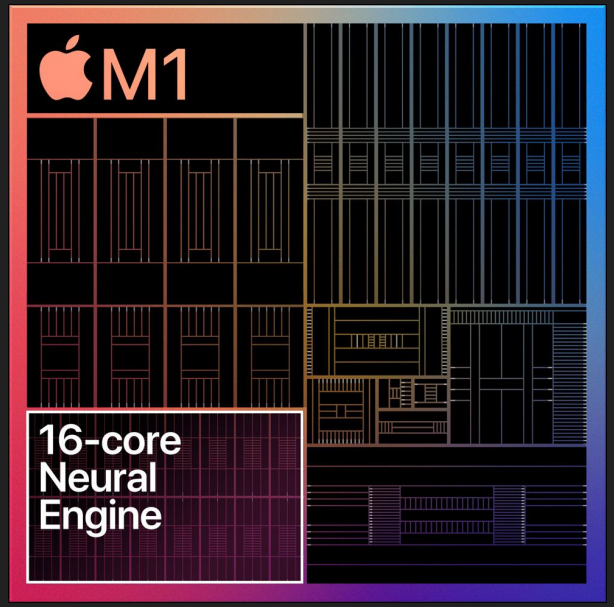
A special processor that makes machine learning models (Core ML) really fast

- Accelerates neural network operations (e.g., convolutions and matrix multiplies)
- Huge improvement on image and signal processing
- Assist GPU processes
  - e.g. keep image features while compressing the data

Not much is publicly known : (

George Hotz is reverse-engineering it :)
https://github.com/geohot/tinygrad/tree/master/ane

Apple Neural Engine (ANE)

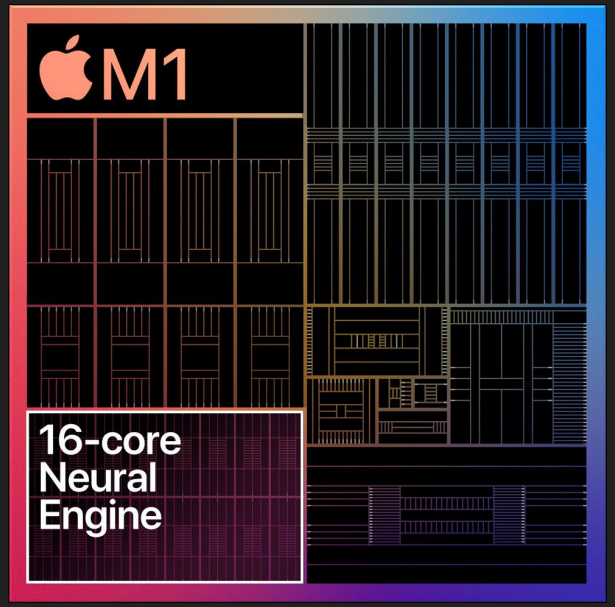


16 wide Kernel DMA engine

Works with 5D Tensors

- Column (width)
- Row (height)
- Plane (channels)
- Depth
- Group (batch)

Apple Neural Engine (ANE)



The ops have several parts

- Header - base addresses of the DMA engines
- KernelDMASrc - 16x wide DMA engine
- Common - parameters for the convolution
- TileDMASrc - Input DMA engine
- L2 - Use the L2 cache
- NE - Configure Kernel/MAC/Post
- TileDMADst - Output DMA engine

Work with 8 base addresses for the DMA streams per op

Apple Matrix coprocessor

AMX machine learning on-die accelerators

- Apple's custom ARM "NEON" or "SVE"

A superset of the ARM ISA that is running on the CPU cores.

- Not publicly exposed to developers
- And not included in Apple's public compilers.

Accelerate.framework (Apple's vector processing framework) takes advantage of AMX

aarch64_amx.py

```
# IDA (disassembler) and Hex-Rays (decompiler) plugin for Apple AMX
#
# WIP research. (This was edited to add more info after someone posted it to
# Hacker News. Click "Revisions" to see full changes.)
#
# Copyright (c) 2020 dougallj


# Based on Python port of VMX intrinsics plugin:
# Copyright (c) 2019 w4kfu - Synacktiv

# Based on AArch64 8.3-A Pointer Authentication plugin:
# Copyright (c) 2018 Eloi Benoist-Vanderbeken - Synacktiv
# Copyright (c) 2018 xerub

# TODO: XZR can be an operand, but I don't handle that correctly in
# the decompuler yet.


# AMX: Apple Matrix coprocessor
#
# This is an undocumented arm64 ISA extension present on the Apple M1. These
# instructions have been reversed from Accelerate (vImage, libBLAS, libBNNS,
# libvDSP and libLAPACK all use them), and by experimenting with their
# behaviour on the M1. Apple has not published a compiler, assembler, or
# disassembler, but by callling into the public Accelerate framework
# APIs you can get the performance benefits (fast multiplication of big
# matrices). This is separate from the Apple Neural Engine.
#
# Warning: This is a work in progress, some of this is going to be incorrect.
#
# This may actually be very similar to Intel Advanced Matrix Extension (AMX),
# making the name collision even more confusing, but it's not a bad place to
# look for some idea of what's probably going on.
#
#
# WIP simulator/hardware tests are at:
# https://gist.github.com/dougallj/7cba721da1a94da725ee37c1e9cd1f21
```

https://gist.github.com/dougallj/7a75a3be1ec69ca550e7c36dc75e0d6f

Secure Enclave on M1

Same as the Secure Enclave in A14

- Previously on T2 chip on Intel MACs

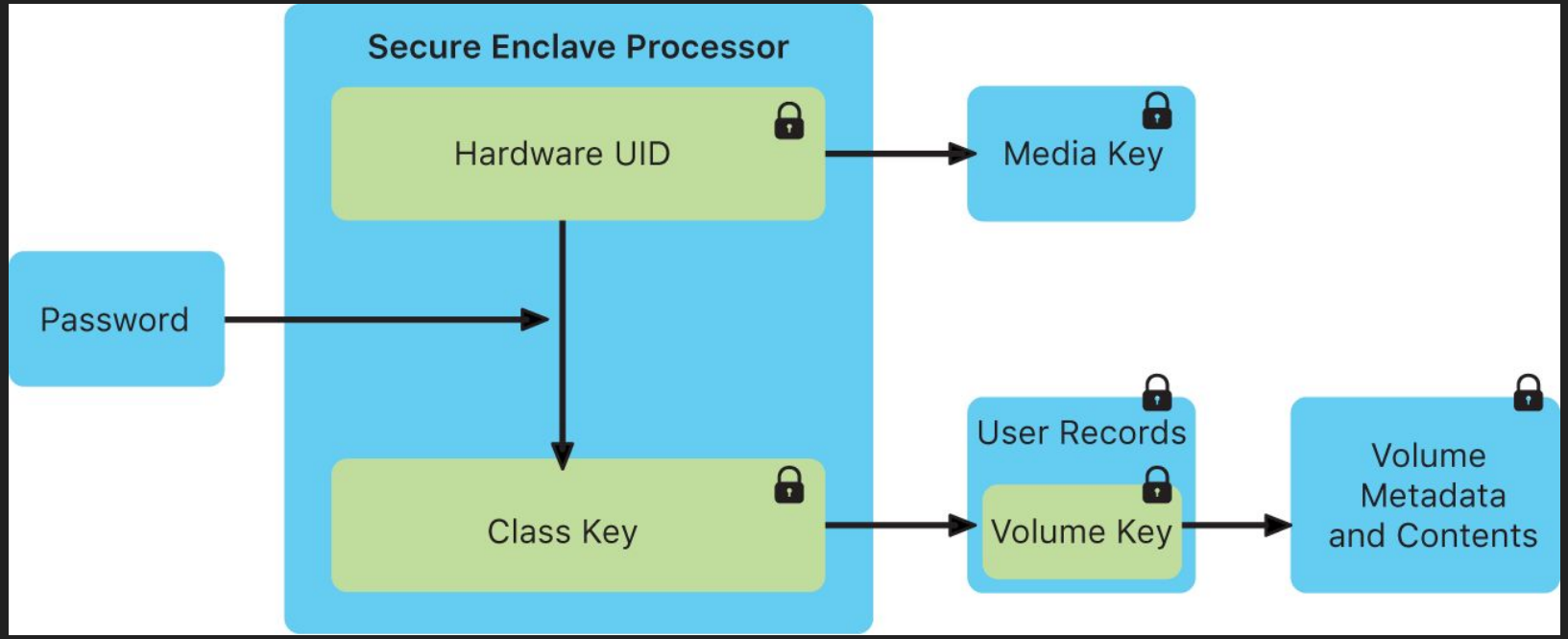
A high-performance storage controller with AES encryption hardware

- Biometrics (e.g., TouchID and FaceID)
- Derives APFS and FileVault keys
- Hardware-verified secure boot

Similar to Intel SGX technology

- Encrypted memory (inline AES engine)
- Compromised macOS kernel

Secure Enclave Key Derivation



Secure Enclave coprocessor (SEP)

L4 family of microkernels

- Runs Secure Enclave coprocessor Operating System (SEPOS)
  - Supplied by application processor during boot time
- Own peripherals, drivers, services and apps
  - Crypto Engine
  - Random Number Generator
  - Fuses
  - GID/UID
- It shares RAM with the AP, but its portion of the RAM is encrypted
  - Specified by TZ0 register
  - Enforced by Apple’s Memory Cache Controller (AMCC)

Secure Enclave coprocessor (SEP)

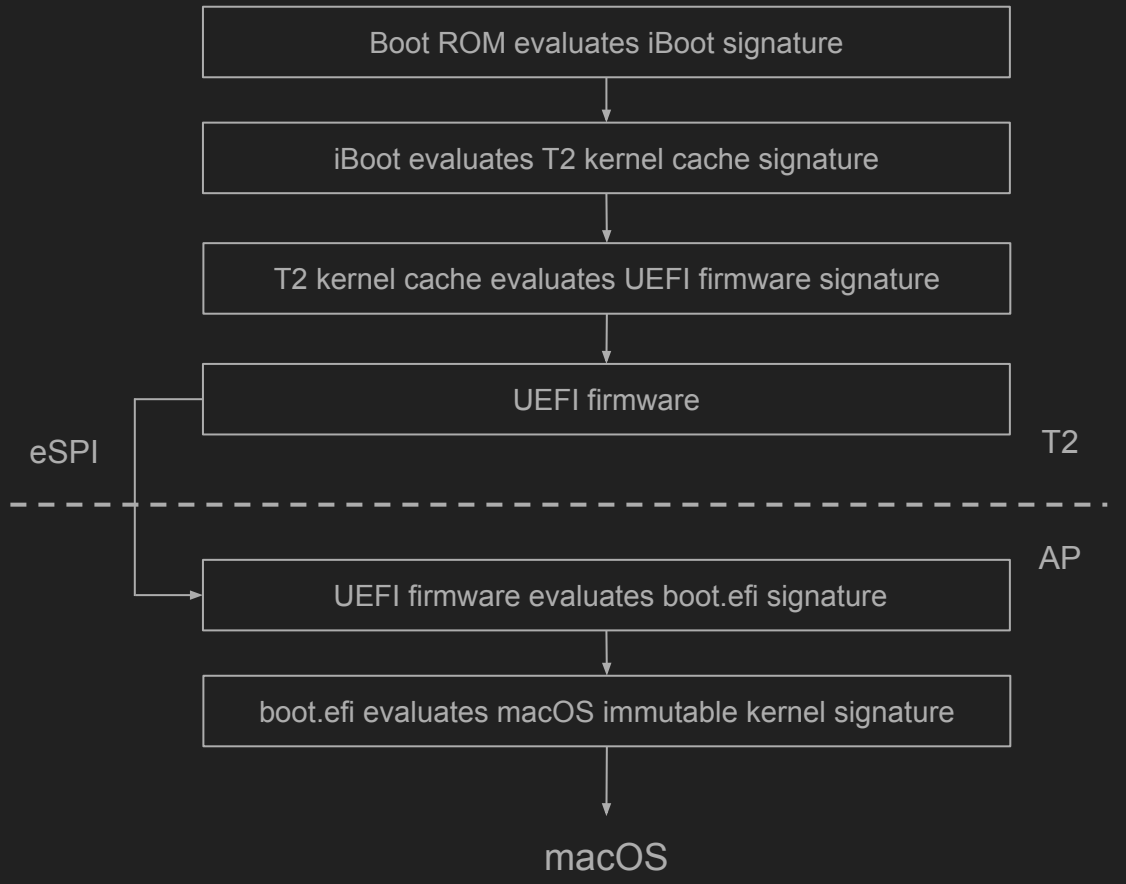
Application processor (AP) and the SEP are seperate

Communicate via a secure mailbox

- A series of registers shared between the processors
- Use interrupts for signalling

Secure Boot

A chain of trust rooted in hardware, including the UEFI firmware, bootloaders, kernel, and kernel extensions necessary for boot.



Thanks!