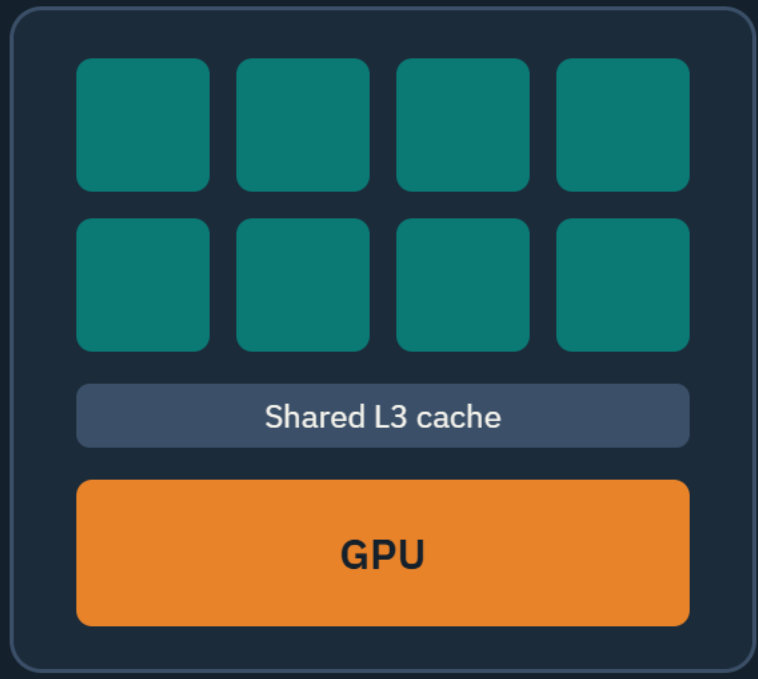
OPERATING SYSTEMS · PROCESSES AND THREADS

CPU and GPU scheduling on a modern chip

How threads get onto cores, and how the OS shares a GPU among them



Cores and multithreading

Kernel and user threads

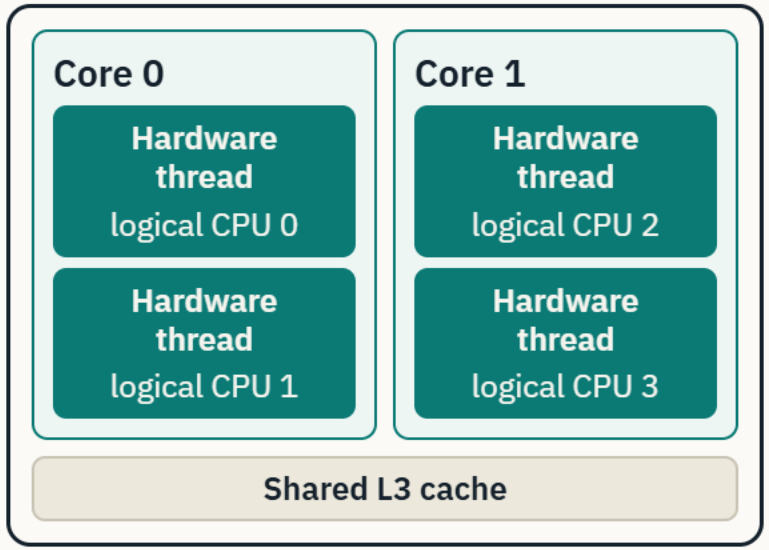
GPU queues and memory

Interrupts and locks

VOCABULARY 1 OF 4 · PROCESSORS

Cores, hardware threads and logical CPUs

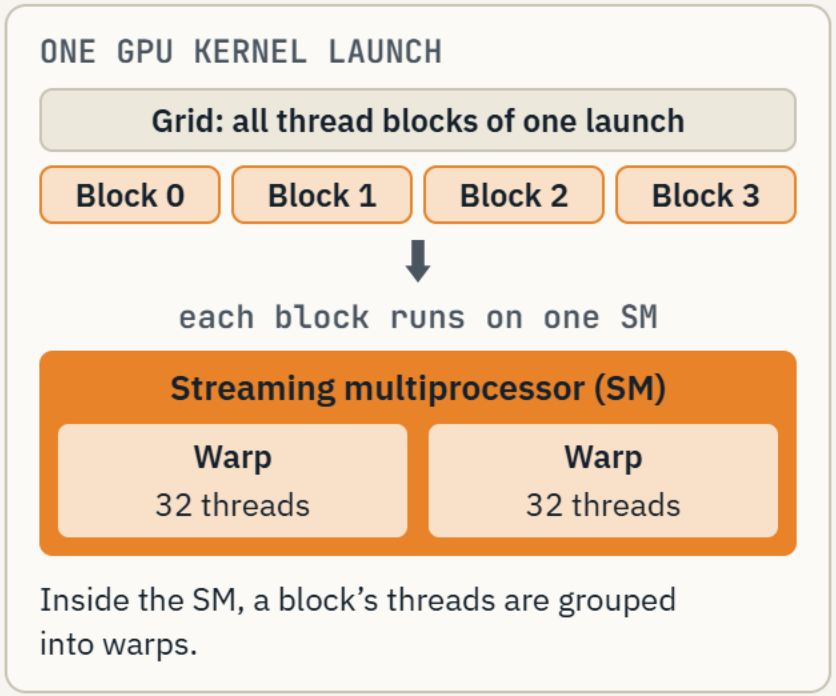
ONE CPU CHIP



The OS sees 4 logical CPUs and picks one for every thread it runs.

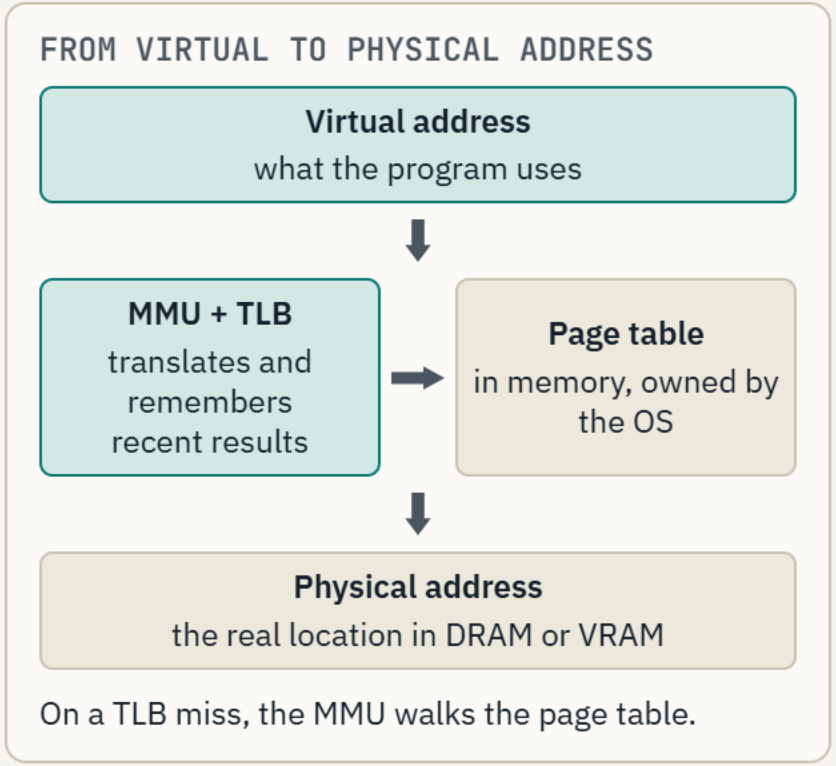
Core	One complete processor that runs instructions. A chip holds several.
Hardware thread	One instruction stream that a core can hold at once. The OS sees each as a CPU.
SMT	Simultaneous multithreading: one core runs two hardware threads. Intel calls it Hyper-Threading.
Logical CPU	A CPU as the OS sees it: one per hardware thread (2 cores × 2 threads = 4).
Kernel thread	A thread the OS kernel knows about and schedules onto a logical CPU.
User thread	A thread managed by a library or runtime inside the program. The kernel cannot see it.

GPU terms: kernels, thread blocks and warps



GPU	Graphics processing unit: a separate processor that runs thousands of threads at once.
GPU kernel	A function the CPU launches to run on the GPU. Not the same as the OS kernel.
Thread block	A group of GPU threads that run together on one SM and share fast memory.
Streaming multiprocessor (SM)	One of the GPU's many cores. AMD calls it a compute unit (CU).
Warp	A group of 32 GPU threads that execute the same instruction together. AMD: wavefront.
SIMT	Single instruction, multiple threads: one instruction drives all 32 threads of a warp.

Memory terms: addresses, translation and caches



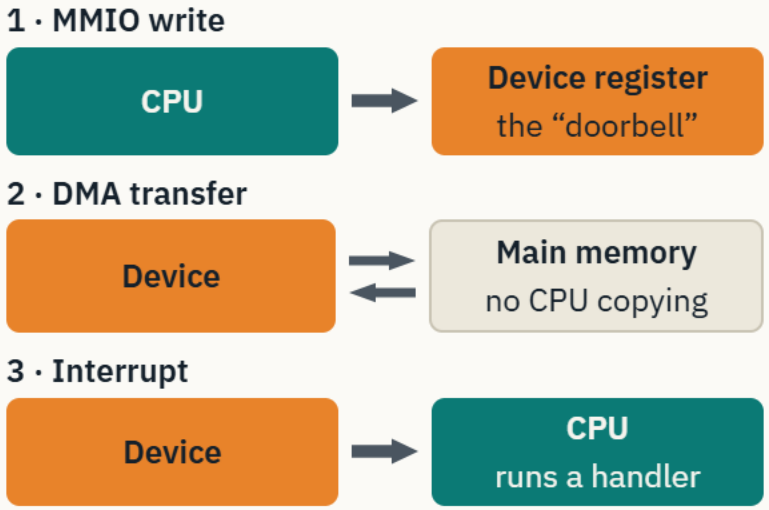
Virtual address	The address a program uses. Each process has its own private address space.
Page table	An OS-owned table that maps each virtual page (usually 4 KB) to a physical page.
Memory management unit (MMU)	Hardware that translates virtual addresses to physical addresses on every access.
Translation lookaside buffer (TLB)	A small cache of recent translations, so most accesses skip the page table.
Cache	Small, fast memory beside the cores that holds recently used data. L1 is the smallest and fastest; L3 is the largest and shared.
DRAM and VRAM	Main memory (dynamic RAM) and the GPU's own video RAM, built from HBM (high-bandwidth memory) or GDDR (graphics double data rate) chips.

VOCABULARY 4 OF 4 · DEVICES

Device terms: how the CPU and devices talk

THREE WAYS CPU AND DEVICE TALK

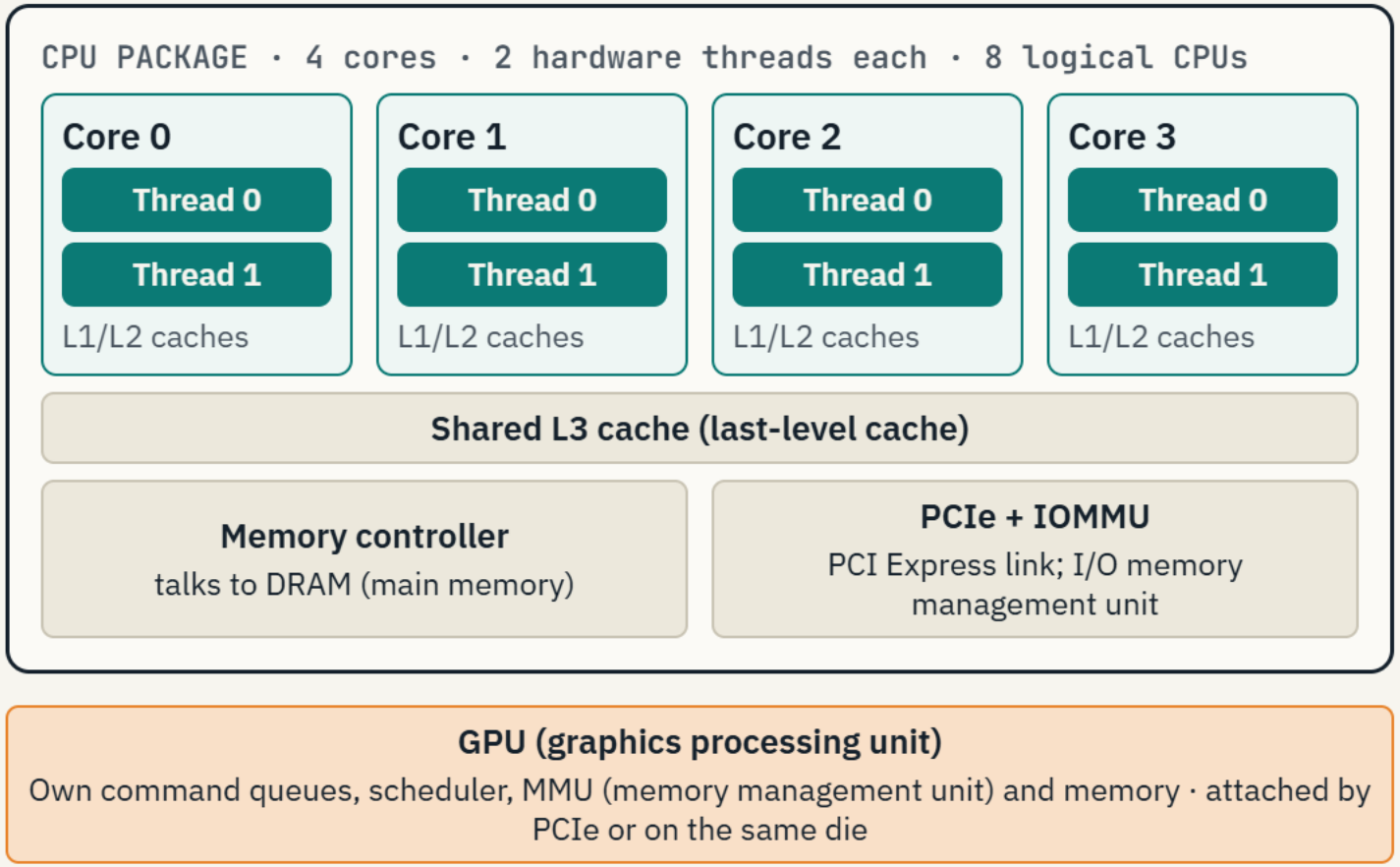
PCIe (PCI Express) is the fast link between the CPU chip and devices such as GPUs.



Memory-mapped I/O (MMIO)	Device registers appear as memory addresses; writing one controls the device.
Direct memory access (DMA)	A device reads or writes main memory itself, without the CPU copying the data.
I/O memory management unit (IOMMU)	Translates and restricts the addresses that devices use for DMA.
Interrupt	A signal that makes the CPU pause and run a short handler (an interrupt service routine).
Fence	A marker the GPU signals when earlier work is finished. Threads can wait on it.
Driver	Operating system code that controls one kind of device, such as a GPU.

THE HARDWARE

What the scheduler sees on a modern chip



Logical CPU

Each hardware thread looks like one CPU to the OS. 4 cores × 2 threads (SMT) = 8 logical CPUs.

Shared resources

Threads on one core share its execution units and L1/L2 caches; all cores share L3 and memory bandwidth.

A separate device

The GPU has its own queues, memory management and memory. The OS controls it through a driver.

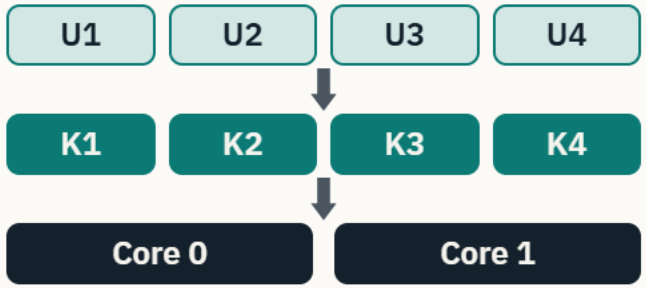
THREAD MODELS

Kernel threads and user threads

User thread Kernel thread (the OS schedules these) CPU core

1:1, kernel-supported

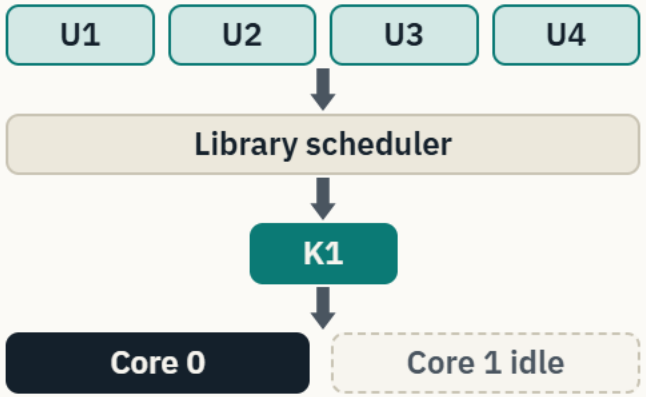
Linux, Windows, macOS



The OS schedules every thread, so blocking is safe and threads run in parallel. Cost: a system call (a request to the kernel) to create or switch.

N:1, user threads only

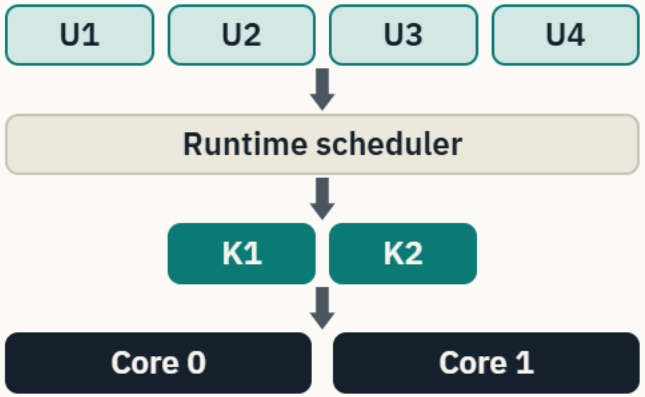
Green threads, early Java



Switches are cheap and need no kernel call. But one blocking system call stalls every thread, and only one core is used.

M:N, hybrid

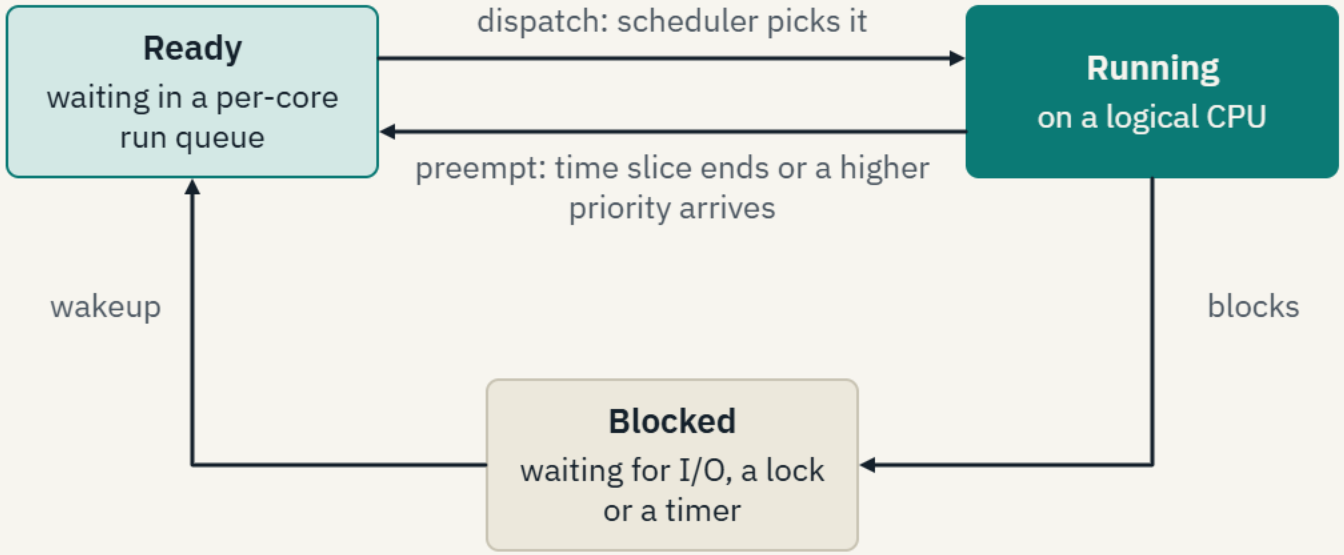
Go, Java virtual threads, Erlang



A runtime multiplexes M user threads over N kernel threads (M larger): cheap and parallel, but blocking calls need care.

CPU SCHEDULING

Getting on and off a CPU



What triggers a switch

- Timer interrupt ends the time slice
- Thread blocks on I/O, a lock or a sleep
- A higher-priority thread becomes ready
- Thread yields or exits

Per-core run queues

Each logical CPU has its own queue, so cores rarely fight over one lock. Idle cores pull work from busy ones. A switch saves registers and swaps page tables.

SIMULTANEOUS MULTITHREADING (SMT) AND MULTI-CORE

Where the scheduler places threads

Naive placement

Slower per thread

Core 0	Core 1
Thread A	idle
Thread B	idle

A and B share one core's execution units and caches. SMT (simultaneous multithreading) typically adds 20–30% more throughput, not 2×.

SMT-aware placement

Full speed

Core 0	Core 1
Thread A	Thread B
idle	idle

A and B each get a whole core. Spread across physical cores first; use SMT siblings only when cores run out.

Scheduling domains

CPUs are grouped by what they share: SMT siblings, cores on one L3 cache, sockets. Balance inside a group first.

Affinity

Affinity: keep a thread on the core where its data is cached. Migrate only if the imbalance outweighs a cold cache.

Hybrid cores

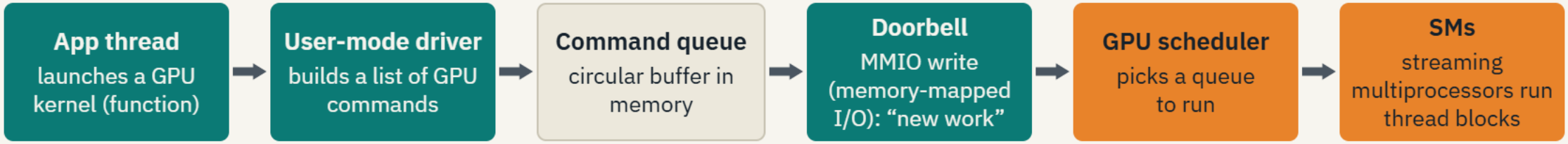
Fast performance cores (P) and slower efficiency cores (E) share a chip. Intel Thread Director hints guide placement.

SMT and security

Siblings share core state, so one thread can spy on the other through timing. Some systems disable SMT.

GPU EXECUTION

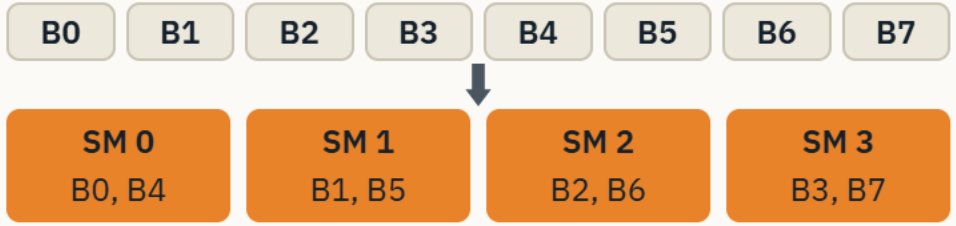
How CPU threads hand work to a GPU



Teal: runs on the CPU. Orange: runs on the GPU.

A kernel launch is a grid of thread blocks

B0–B7: thread blocks. SM: streaming multiprocessor.



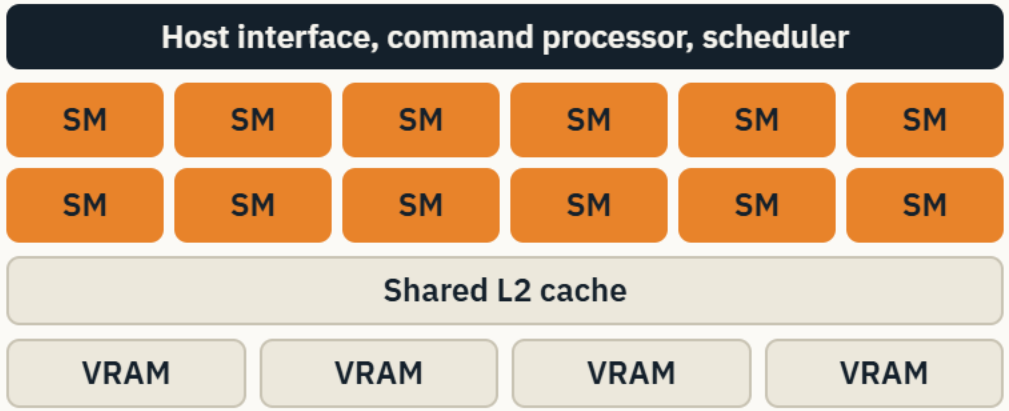
Key points

- Submission is asynchronous: the CPU thread returns at once and later waits on a fence (a completion marker).
- A thread block runs to completion on one SM. Its threads run in warps of 32, in lockstep.
- Older GPUs could not interrupt a running kernel. Modern ones can preempt it (pause it and save its state).
- The OS never schedules individual GPU threads. It schedules the CPU thread and arbitrates the queues.

GPU INTERNALS

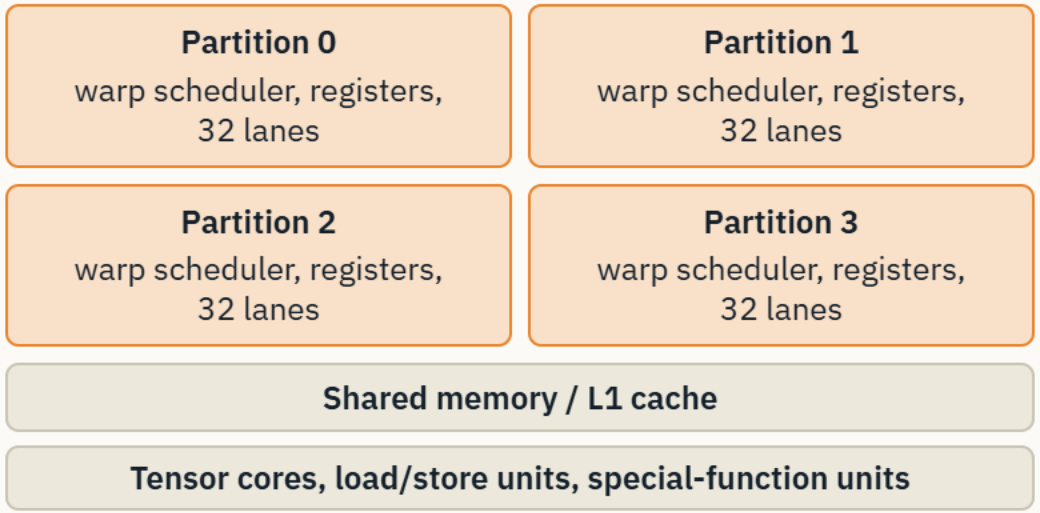
What is inside a GPU

The chip



SM = streaming multiprocessor (AMD: compute unit). L2 = second-level cache. VRAM = video RAM.

One SM, zoomed in



Lane: one arithmetic unit serving one thread of a warp (a group of 32 threads). Tensor core: unit specialized for matrix multiplication. Recent datacenter GPU: 100+ SMs, up to 64 resident warps and about 64K registers per SM.

GPU INTERNALS

Warps: running thousands of threads

One warp, one instruction at a time

A warp is a group of 32 GPU threads sharing one instruction stream. Each box is one lane (one thread); 8 of 32 are drawn.

if branch	on	on	on	on	off	off	off	off
else branch	off	off	off	off	on	on	on	on

Both paths run one after the other. Masked lanes do no work, so divergence wastes cycles.

Switching warps hides memory latency

Each warp runs one step, then waits on memory.

Warp 0	run	wait	wait	wait	run	wait	wait	wait
Warp 1	wait	run	wait	wait	wait	run	wait	wait
Warp 2	wait	wait	run	wait	wait	wait	run	wait
Warp 3	wait	wait	wait	run	wait	wait	wait	run

The scheduler issues from a ready warp every cycle, so the SM (streaming multiprocessor) never sits idle.

CPU switch vs warp switch

A CPU context switch saves registers and swaps page tables. A warp switch costs nothing: every resident warp keeps its registers on chip.

Occupancy

Occupancy is the number of warps resident on an SM at once. More warps hide more latency, but each block's registers and shared memory limit how many fit.

GPU INTERNALS

GPU memory hierarchy and coalescing

Levels of GPU memory

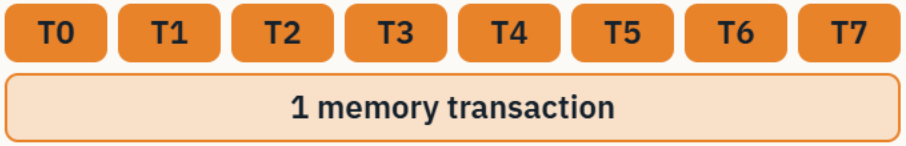
Level	Shared by	Size	Cycles
Registers	one thread	~64K per SM	~1
Shared memory / L1	one block	~200 KB per SM	~30
L2 cache	whole GPU	tens of MB	~200
VRAM (HBM, GDDR)	whole GPU	tens of GB	400–800
Host memory	CPU and GPU	system RAM	over PCIe: 1000s

Approximate values for recent datacenter GPUs. VRAM = video RAM. HBM = high-bandwidth memory (stacked chips). GDDR = graphics double data rate memory. PCIe = PCI Express. Cycle = one tick of the GPU clock. Other warps can hide latency, but nothing hides a bandwidth limit.

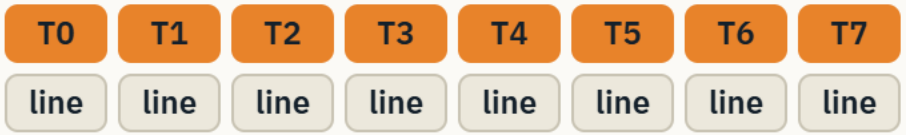
Coalescing

Eight threads (T0–T7) each read one word.

Adjacent addresses



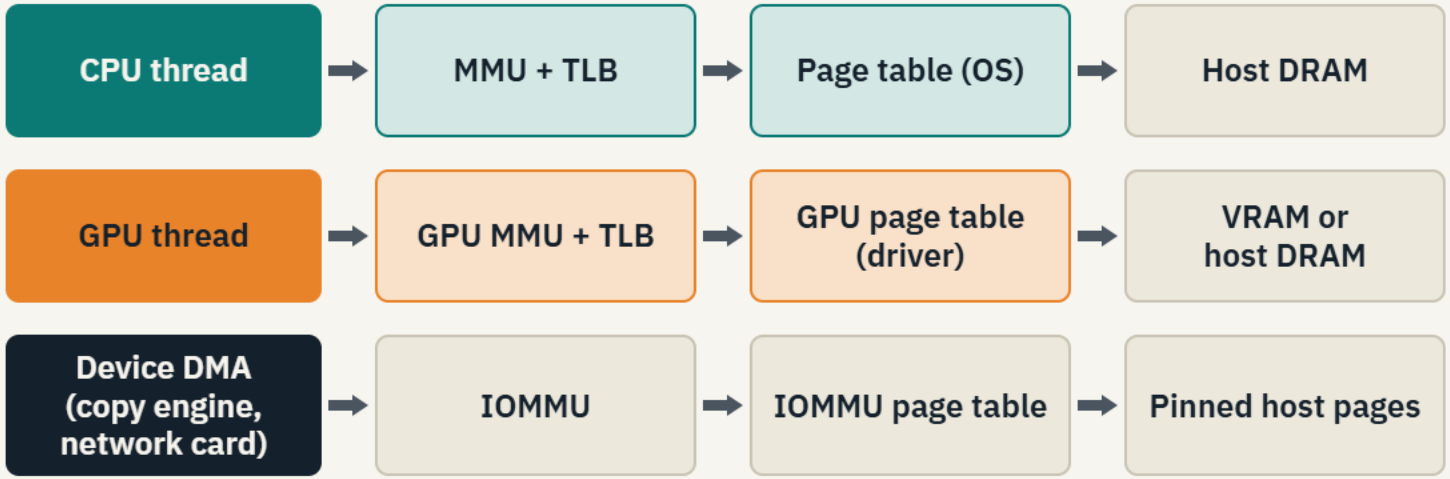
Addresses far apart



8 transactions for the same data. Coalescing = merging neighbouring reads into one wide memory transaction.

GPU MEMORY

Memory access and translation with GPUs



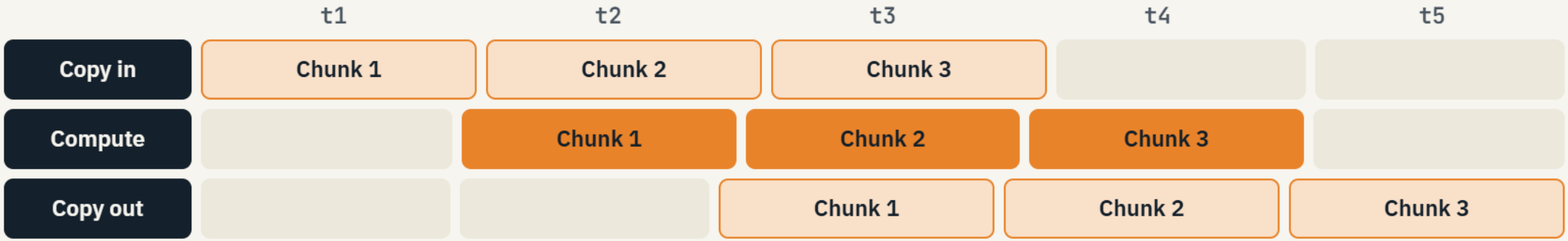
Same virtual address, different page tables: the driver keeps them consistent (unmapping a page on the CPU also invalidates the GPU's TLB entry). MMU = memory management unit. TLB = translation lookaside buffer. IOMMU = I/O memory management unit. DMA = direct memory access. DRAM = main memory. VRAM = video RAM.

What to remember

- Each GPU context has its own virtual address space.
- Unified virtual memory lets CPU and GPU use the same pointers; pages migrate on demand.
- A GPU page fault stalls the faulting warp while the driver maps or migrates the page.
- DMA goes through the IOMMU, which also isolates devices. Pages are normally pinned (locked in place).

OVERLAPPING WORK

Streams: copy and compute in parallel



Streams

A stream is a queue of GPU work that runs in order. Different streams may run concurrently.

Events and fences

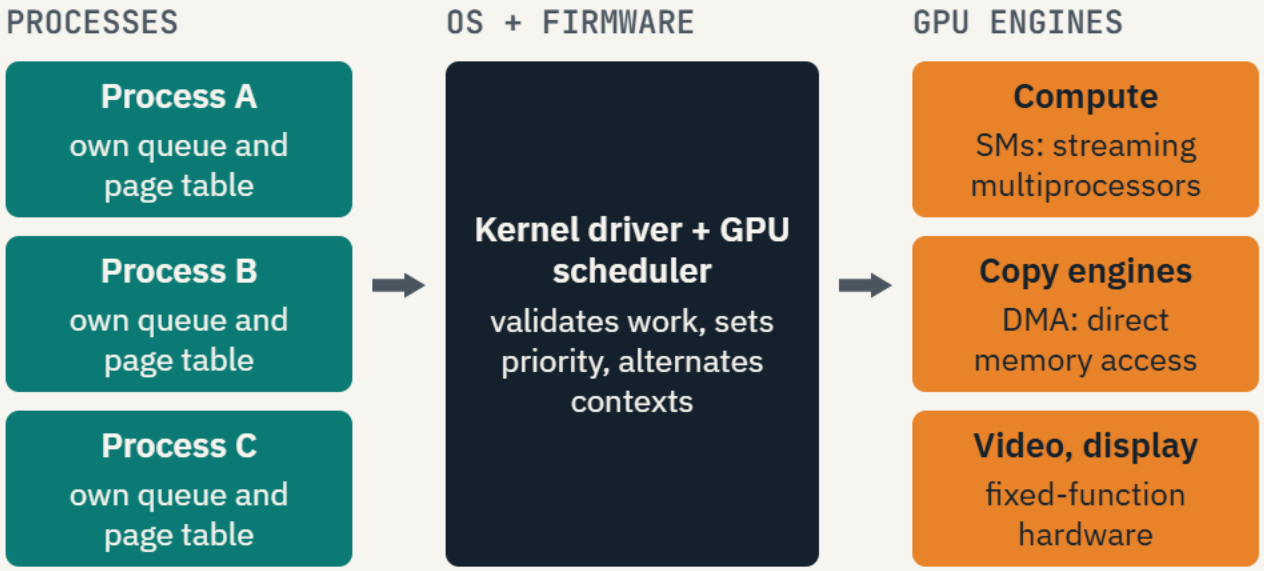
An event marks a point in a stream. Other streams, or the CPU, can wait on it. This orders GPU work.

Pinned memory

Asynchronous copies need pinned (page-locked) host memory, so the DMA (direct memory access) engine can read it while the OS cannot move the pages.

SHARING THE GPU

Sharing one GPU among many processes



Each process submits to its own queue. Firmware (software stored on the GPU) and the driver decide which queue each engine serves next, so a copy can overlap with compute.

Time slicing

Contexts take turns on the whole GPU. Preemption bounds the wait, but each switch saves GPU state.

Spatial sharing

Kernels from different streams or MPS (Multi-Process Service) clients share the GPU's streaming multiprocessors (SMs) at once. Weaker isolation.

Hardware partitioning

MIG (Multi-Instance GPU) or SR-IOV (single-root I/O virtualization) give each user dedicated SMs and memory. Fixed sizes.

SHARING THE GPU

Context switches and preemption on a GPU

Time slicing between processes



Each switch saves one context’s state and restores the next. That time does no useful work.

How finely can the GPU stop work?

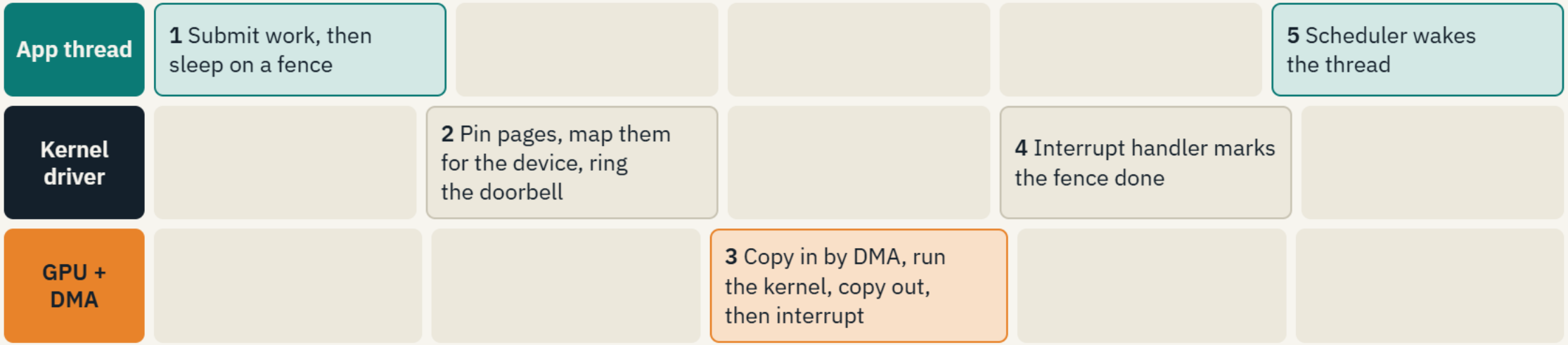
Granularity	Waits for	State saved
Whole kernel or draw	kernel to finish	nothing
Thread block	running blocks to end	little
Instruction level	almost nothing	all thread state

Hang protection

A kernel that never ends would freeze the display. Windows resets the GPU after about 2 seconds without progress (Timeout Detection and Recovery, TDR). Linux drivers use job timeouts and GPU reset.

COMPLETING A GPU REQUEST

Interrupts, locks, and DMA: finishing GPU work



Interrupt handlers

An interrupt pauses the CPU to run a short handler. The top half acknowledges the device and never sleeps; a deferred bottom half wakes waiting threads.

Locks

Spinlocks (busy-waiting locks) guard data shared with handlers; mutexes (sleeping locks) guard other paths. Waiting for the GPU uses a fence (completion marker), not a lock.

DMA (direct memory access)

Copy engines move data without the CPU. Pages are pinned (locked in place) and mapped through the IOMMU (I/O memory management unit).

SUMMARY

Key ideas from today

1. The scheduler assigns logical CPUs (hardware threads). SMT (simultaneous multithreading) siblings are not full cores.
2. The OS schedules kernel threads. User threads need a runtime to map onto them.
3. Per-core run queues, timer interrupts and load balancing decide who runs where and when.
4. A GPU runs warps (groups of 32 threads) on streaming multiprocessors, hiding latency by switching between warps. The driver arbitrates its queues between processes.
5. Interrupts, fences and DMA (direct memory access) turn GPU completion back into ordinary thread wakeups.

Questions to think about

- Why might a scheduler leave an SMT sibling idle while threads wait?
- What should happen to a running GPU kernel if the CPU unmaps one of its pages?
- Why can an interrupt handler not take a mutex?