

GPUs: from CUDA to the gigawatt rack

Xuan Jiang | Prof. Sengupta and Demmel's PhD student who graduated many years ago | Oct 2026

Agenda and timing

	1	2	break	3	4	5
	0:00	0:25	1:05	1:15	2:30	3:00 3:10
0:00 • 25 min	1 Why GPUs exist	and why every LLM runs on one: von Neumann vs SIMT, latency vs throughput				
0:25 • 40 min	2 Inside one GPU	threads, memory, roofline, then how fast an H200 answers, fewer bits, the KV cache				
1:05 • 10 min	Break					
1:15 • 75 min	3 Writing CUDA	what changes from CPU code, in 8 short examples. Moving 1 GB inside vs across servers				
2:30 • 30 min	4 The AI factory	five GPU generations, training time, 72-GPU racks and MoE, failures, power				
3:00 • 10 min	5 What is next	the public roadmap, the software stack, and where you fit				
3:10	Questions					

GB/s are bytes, Gb/s are bits. All numbers are public. The source line on each slide names the page. Slides titled 'Your turn' are questions for you.

Three numbers to carry through the talk

4.8 TB/s vs 50 GB/s. 72. 2.5 kW.

- **4.8 TB/s vs 50 GB/s:** HBM bandwidth of one H200 against the 400 Gb/s network port behind it. A factor of about 100. Everything in distributed GPU computing is about this gap.
- **72:** GPUs in one NVLink domain since GB200 NVL72 (2024). It was 8 for the ten years before. Rubin keeps 72 packages per rack, Kyber targets 144 packages (576 dies) in 2027.
- **2.5 kW:** NVIDIA's own factory-level figure for Rubin: 100 MW buys about 40,000 GPUs. The GPU itself is a fraction of that. Power, not silicon, is the budget.

Source: nvidia.com/en-us/data-center/h200; aws.amazon.com/ec2/instance-types/p5; nvidia.com/en-us/data-center/vera-rubin-nvl72 (GPUs in 100 MW)

GPUs: from CUDA to the gigawatt rack | October 2026

Your turn: quick poll



Raise your hand if you have...

1

used a GPU from PyTorch or JAX

2

written a CUDA kernel

3

run a job on more than one node

4

paid a GPU bill

There is no right answer. It tells me where to slow down and where to skip.



Chapter 1

Why GPUs exist, and why every LLM runs on one

25 minutes. From von Neumann to 270,000 threads in flight

Your turn: guess first

? How many threads can one H200 keep in flight at the same time?

A 112

B 2,048

C about 27,000

D about 270,000

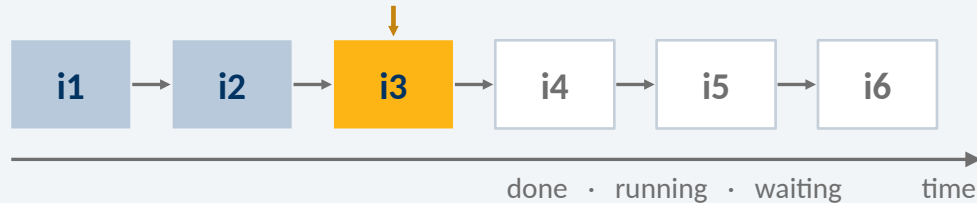
D. $132 \text{ SMs} \times 2,048 \text{ resident threads} = 270,336$. A 56-core server CPU runs 112 hardware threads.

Click to show the answer. Sources on the next page.

The von Neumann model is serial. A GPU is parallel.

von Neumann (1945): serial

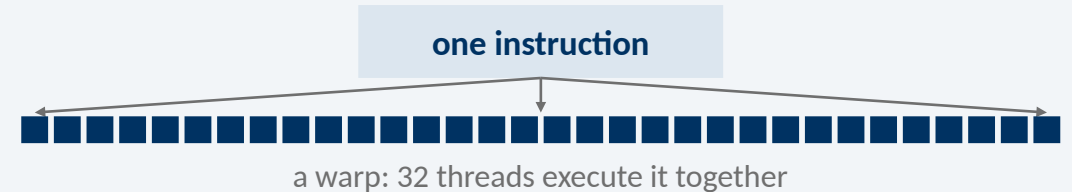
one program counter, one instruction at a time



- One instruction stream, executed in order (EDVAC, 1945)
- Flynn (1966): SISD. Backus (1977): 'word-at-a-time' thinking
- CPU answer: caches, branch prediction, out-of-order, all to make one stream fast

VS

GPU (CUDA, 2006): parallel



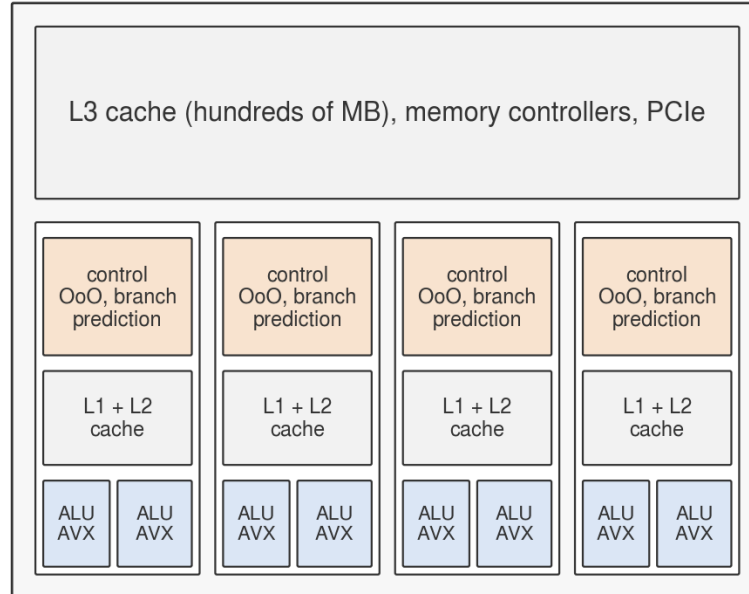
$$32 \text{ threads per warp} \times 64 \text{ warps per SM} \times 132 \text{ SMs per H200} = 270,336 \text{ threads in flight}$$

- SIMT: one instruction, 32 threads. Each may take its own branch
- GPU answer: while some threads wait for memory, others compute

Both are stored-program machines. The difference that matters is serial vs parallel: one instruction stream in order, or tens of thousands of threads in flight at once.

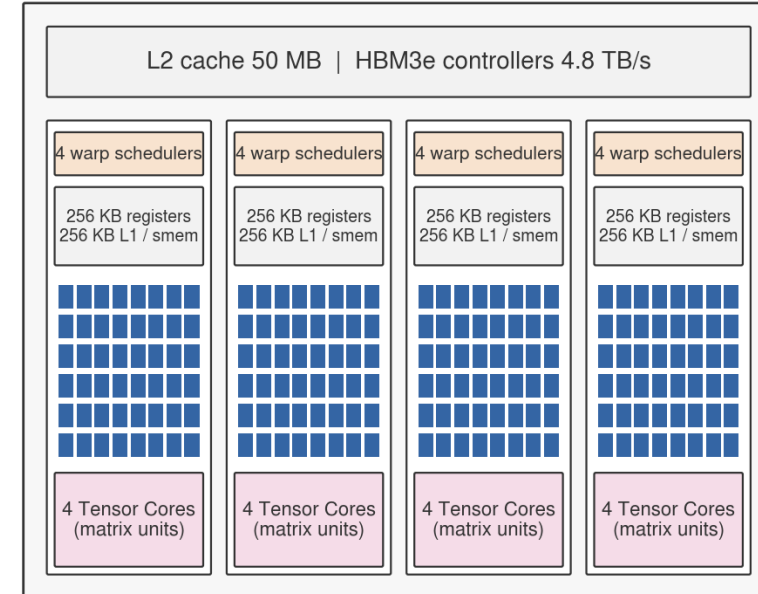
Why GPUs exist: a latency machine and a throughput machine

CPU core complex: few threads, each fast



transistors buy latency: caches, prediction, out-of-order issue

GPU, 4 of 132 SMs: many threads, each slow



transistors buy throughput: ALUs and matrix units, parallelism hides latency

- A CPU spends its transistors on making one thread fast: big caches, branch prediction, out-of-order issue. A GPU spends them on arithmetic and hides memory latency with tens of thousands of threads.
- Same silicon budget, opposite goal: how fast can one thread finish, or how much work finishes per second.

Source: schematic, not to scale. SM contents from the NVIDIA H100 whitepaper: nvidia.com/en-us/data-center/technologies/hopper-architecture

GPUs: from CUDA to the gigawatt rack | October 2026

Latency hiding: the CPU shortens the wait, the GPU fills it

CPU thread: compute, then wait for DRAM (about 100 ns = hundreds of cycles)



cache, prefetchers and out-of-order execution shorten the stalls, they do not remove them

GPU SM: 8 of up to 64 resident warps, one scheduler issues from whichever warp is ready



- Little's law: bytes in flight = bandwidth x latency. 4.8 TB/s x 600 ns is about 2.9 MB in flight, about 22,000 outstanding 128-byte requests. No cache hierarchy delivers that. Threads do.
- So a GPU program must expose tens of thousands of independent operations at all times. Code with little parallelism runs on a GPU, slowly.

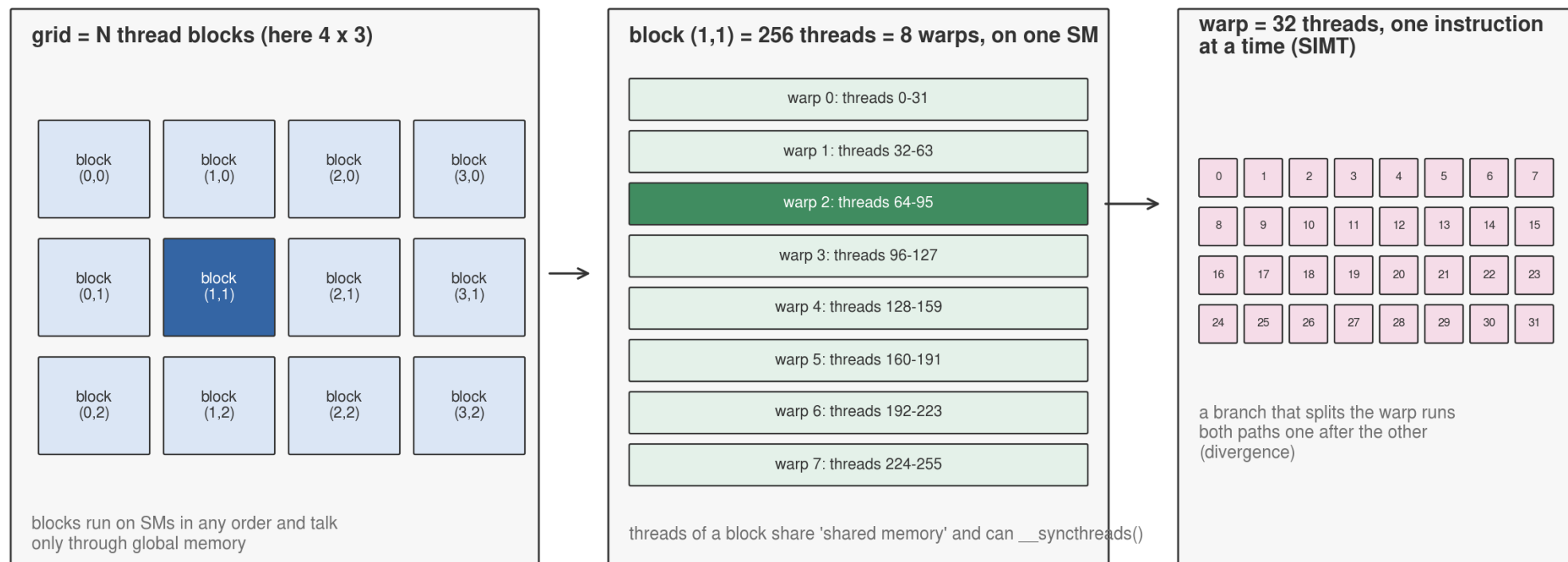


Chapter 2

Inside one GPU: threads, memory, and an LLM

40 minutes. Execution model, memory, roofline, three worked examples

The execution model: grid, blocks, warps, threads

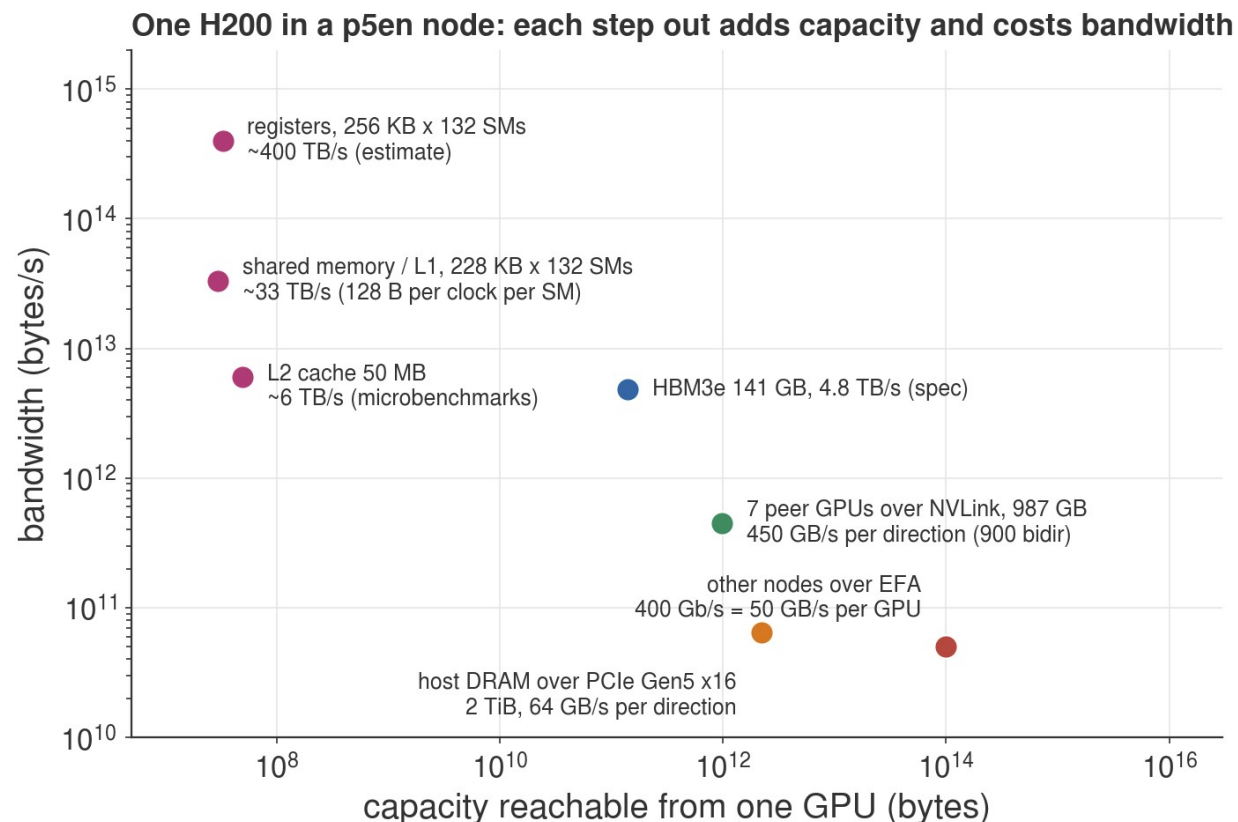


- You write one function (the kernel) for one thread and launch it for millions of threads. Threads form blocks of up to 1,024. Blocks are independent and run on SMs in any order.
- Hardware runs threads in warps of 32 with one instruction stream (SIMT), so branches inside a warp serialize. Threads in a block share fast memory and can synchronize. Blocks cannot.

Source: CUDA C++ Programming Guide, chapter 'Programming Model': docs.nvidia.com/cuda/cuda-c-programming-guide

The memory hierarchy is the program

- CUDA exposes memory spaces explicitly: registers, shared memory (per block), global memory (HBM), plus constant and texture paths. L1 and L2 are transparent.
- Every step outward trades bandwidth for capacity: registers at hundreds of TB/s, shared memory at tens, HBM at 4.8 TB/s, NVLink at 450 GB/s per direction, PCIe at 64, the network at 50.
- A fast kernel moves data as few steps as possible: load once from HBM, reuse from shared memory and registers, write once.
- Across GPUs the same rule holds: keep traffic inside the NVLink domain when you can, and know that crossing to another node costs 10x in bandwidth and 10x in latency.



Source: nvidia.com/en-us/data-center/h200/; nvidia.com/en-us/data-center/h100/; aws.amazon.com/ec2/instance-types/p5/. Register and L2 bandwidths are estimates

Your turn: guess first

? An LLM generates one token at a time for one user. What share of an H200's peak compute (BF16) does it use?

A about 50%

B about 10%

C about 5%

D about 0.5%

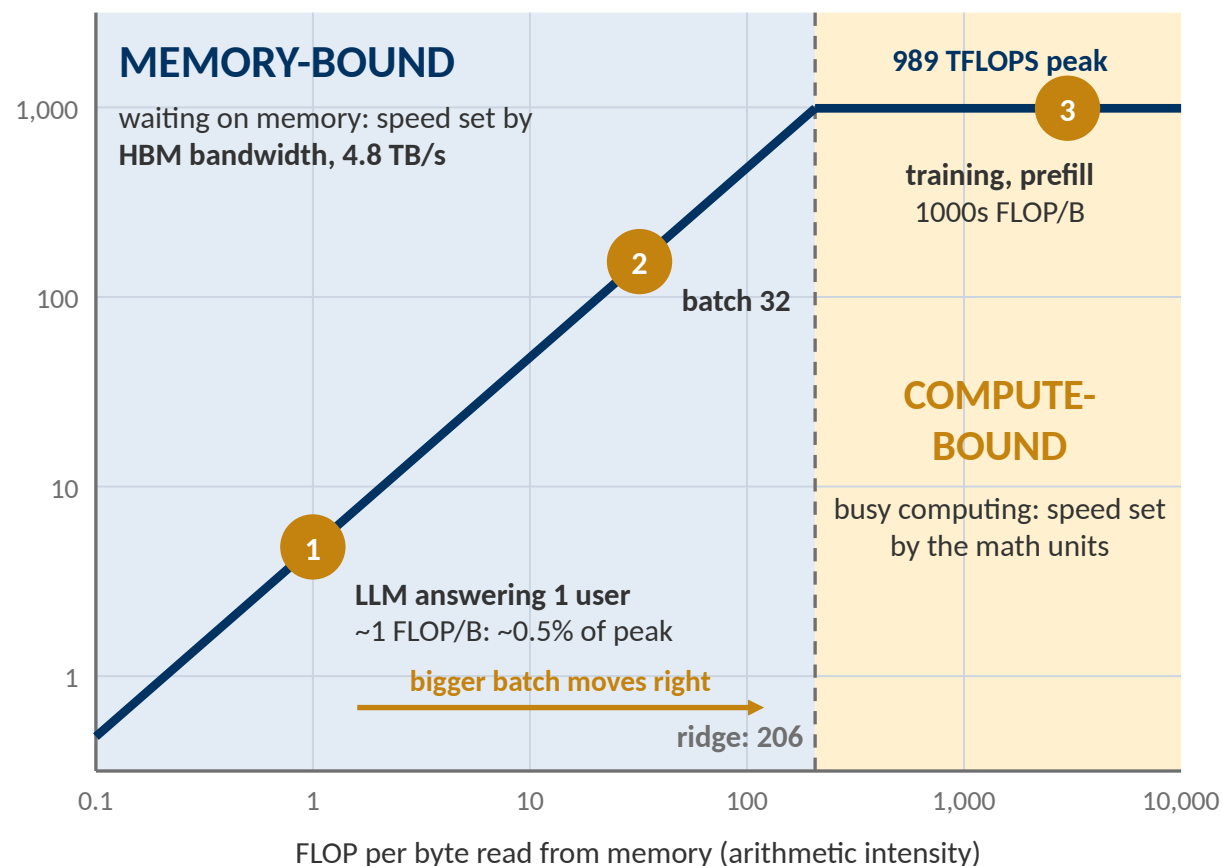
D. Every new token reads every weight once. A 70B model is 70 billion weights $\times$ 2 bytes = 140 GB, and the math is 1 multiply + 1 add per weight = 140 GFLOP. $140 \text{ GFLOP} \div 140 \text{ GB} = 1 \text{ FLOP per byte}$, so $4.8 \text{ TB/s} \times 1 = 4.8 \text{ TFLOPS}$ of 989: about 0.5%.

Click to show the answer. Sources on the next page.

Roofline: which bound are you under?

- Arithmetic intensity = FLOP per byte moved from HBM. Below the ridge point the kernel is memory-bound, above it compute-bound.
- Ridge point = peak $\div$ bandwidth: 989 TFLOPS $\div$ 4.8 TB/s = 206 FLOP per byte (H200, BF16).
- One new token for one user reads every weight once. A 70B model: 140 GB read (2 bytes per weight) for 140 GFLOP (1 multiply + 1 add per weight): 1 FLOP per byte. Attainable = 4.8 TB/s $\times$ 1 = 4.8 TFLOPS, 0.5% of the 989 TFLOPS peak.
- Prefill and training GEMMs have intensities in the thousands and are compute-bound. The same GPU is a different machine for the two phases.
- Consequence for buying: for decode, HBM bandwidth and capacity are the spec, not TFLOPS.

TFLOPS you can reach (H200, BF16)



Source: peak numbers from nvidia.com/en-us/data-center/h200 (dense = sparse / 2)

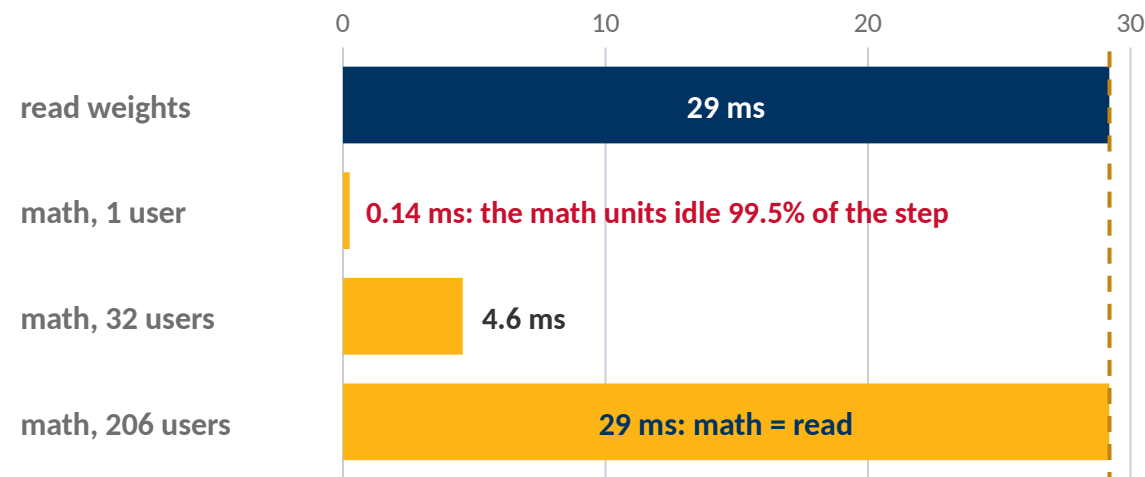
GPUs: from CUDA to the gigawatt rack | October 2026

Worked example: how fast can one H200 answer you?

Setup: Llama 3.1 70B, 70 billion weights $\times$ 2 bytes (BF16), on one H200 (4.8 TB/s, 989 TFLOPS). One user, one new token.

- 1 **Every new token reads all 70 billion weights once**
70 billion $\times$ 2 bytes = **140 GB** (the H200 holds 141 GB)
- 2 **Reading takes time**
140 GB $\div$ 4.8 TB/s = **29 ms** $\rightarrow$ at most **34 tokens/s**
- 3 **The math is tiny: 1 multiply + 1 add per weight**
2 FLOP $\times$ 70 billion = 140 GFLOP $\div$ 989 TFLOPS = **0.14 ms**
- 4 **32 users share one read of the weights**
4.6 ms of math hides inside the 29 ms read $\rightarrow$ **$\approx$ 1,100 tokens/s**
- 5 **206 users: the math fills the whole 29 ms**
206 $\times$ 0.14 ms $\approx$ 29 ms: balanced, the ridge point (206)

One decode step: reading and math run at the same time (ms)



1 user
34 tokens/s

32 users
 $\approx$ 1,100 tokens/s

206 users
 $\approx$ 7,000 tokens/s

One user: the GPU mostly waits for memory. 206 users at once: the same GPU makes about 200x the tokens. That is why serving systems batch many users of the same model together.

Source: [Llama 3.1 70B config](#); H200: nvidia.com/en-us/data-center/h200

GPUs: from CUDA to the gigawatt rack | October 2026

Fewer bits per weight: the same H200 answers 4x faster

Same 70B model, same H200 (141 GB, 4.8 TB/s). Only the number of bits per weight changes.

bits per weight	the H200's 141 GB of memory	one token takes	one user gets
16-bit (BF16) 2 bytes	weights 140 GB · 1 GB left	29 ms	34 tokens/s
8-bit (FP8) 1 byte	weights 70 GB 71 GB free	15 ms	69 tokens/s
4-bit half a byte	35 GB 106 GB free	7.3 ms	137 tokens/s

Why it works: decode is memory-bound, so time $\approx$ bytes $\div$ bandwidth. Half the bytes, half the time.

The catch: fewer bits, less precision. 8-bit is common in serving. Test 4-bit on your own task first.

Same chip: 4x the tokens per second and 105 GB more room for conversations, only by storing each weight in fewer bits.

Source: [Llama 3.1 70B config](#); H200: nvidia.com/en-us/data-center/h200

GPUs: from CUDA to the gigawatt rack | October 2026

Worked example: the KV cache decides how many users fit

Llama 3.1 70B with 8-bit weights on one H200: **70 GB** of weights, **71 GB** free. Where does the free memory go?

1 Each token leaves a note in every layer: its K and V

$$\begin{array}{ccccccc}
 \boxed{2} & \times & \boxed{80} & \times & \boxed{8} & \times & \boxed{128} \times \boxed{2} = \boxed{327,680 \text{ bytes} \approx 320 \text{ KB}} \\
 \text{K and V} & & \text{layers} & & \text{KV heads} & & \text{numbers per head} \quad \text{bytes per number}
 \end{array}$$

From the model's config: 80 layers, 8 KV heads of 128 numbers each (head size = 8,192 hidden ÷ 64 attention heads), BF16.

2 One chat of 8,000 tokens

$$8,000 \times 320 \text{ KB} = 2.6 \text{ GB}$$

3 How many chats fit in 71 GB?

71 GB ÷ 2.6 GB ≈ **27 users**, far below the 206 that keep the math busy

4 One long session: 128,000 tokens

128,000 × 320 KB = **42 GB**: one user takes 60% of the free memory

The H200's 141 GB



8,000-token chats: 27 fit



128,000-token sessions: 1 fits



Memory, not math, decides how many users share a GPU. Long contexts make it worse, that is why each new GPU adds memory.



Chapter 3

Writing CUDA: what changes from CPU code

75 minutes, 8 short code examples

Writing CUDA: what actually changes

The language is C++ with three additions. The hard part is the model: explicit parallel decomposition, explicit memory spaces, explicit synchronization.

- A kernel is a function run by every thread. You compute your index, you do your part.
- You manage two address spaces (host and device) and the copies between them.
- You choose the launch shape (blocks and threads) and it changes performance by 10x.
- Bulk-synchronous thinking: phases separated by barriers, not fine-grained locks.
- Eight examples follow. Each is short on purpose. Together they are most of what a kernel author does.

Your turn: guess first



Adding two arrays of 67 million floats takes 0.17 ms on the GPU. How long does copying the data in and out over PCIe take?



0.1 ms



1.6 ms



16 ms



1.6 s

C. About 16 ms: 805 MB over PCIe Gen5 at about 50 GB/s, roughly 100x the kernel time. Keep data on the GPU across many kernels.

Click to show the answer. Sources on the next page.

Example 1: vector add. The loop disappears, the index appears

C on the CPU

```
// CPU: one thread, one loop
void add(int n, const float* a,
        const float* b, float* c) {
    for (int i = 0; i < n; i++)
        c[i] = a[i] + b[i];
}

add(n, a, b, c);           // ~25 ms, 1 core
```

CUDA on the GPU

```
// GPU: one thread per element
__global__ void add(int n, const float* a,
                   const float* b, float* c) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n) c[i] = a[i] + b[i];
}

// a, b, c: device buffers from cudaMalloc
cudaMemcpy(a, h_a, n * 4, cudaMemcpyHostToDevice);
cudaMemcpy(b, h_b, n * 4, cudaMemcpyHostToDevice);
int block = 256, grid = (n + block - 1) / block;
add<<<grid, block>>>(n, a, b, c); // async, ~0.2 ms
cudaMemcpy(h_c, c, n * 4, cudaMemcpyDeviceToHost);
```

CPU: one thread, one loop



$i = 0, 1, 2, \dots$ one at a time

one element after another
~25 ms on one core

GPU: one thread per element



all elements at the same time
~0.2 ms for the kernel

Which element is mine? $i = \text{blockIdx.x} \times \text{blockDim.x} + \text{threadIdx.x} = 2 \times 4 + 1 = 9$ (4 threads per block in this picture, 256 in the code)

Example 2: the launch is async, and the index is yours to compute

```
// Grid-stride loop: any n, any grid size
__global__ void add(int n, const float* a,
                  const float* b, float* c) {
    int stride = blockDim.x * blockDim.x;
    for (int i = blockIdx.x * blockDim.x + threadIdx.x;
         i < n; i += stride)
        c[i] = a[i] + b[i];
}
// fixed grid: 132 SMs x 8 blocks of 256 threads
add<<<132 * 8, 256>>>(n, a, b, c);

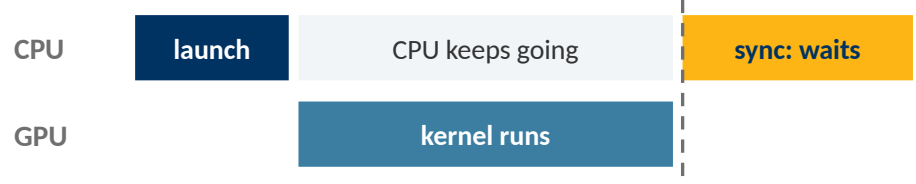
// Always check: launch errors are synchronous,
// execution errors arrive later
#define CK(x) do { cudaError_t e = (x); \
    if (e != cudaSuccess) { printf("%s:%d %s\n", \
        __FILE__, __LINE__, cudaGetErrorString(e)); \
        exit(1); } } while (0)
CK(cudaGetLastError());           // bad launch config
CK(cudaDeviceSynchronize());     // illegal address, etc.
```

A fixed grid covers any n

	t0	t1	t2	t3	t4	t5	t6	t7
pass 1	0	1	2	3	4	5	6	7
pass 2	8	9	10	11	12	13	14	15
pass 3	16	17	18	19	20	21	22	23

each thread does $i, i + 8, i + 16$: $\text{stride} = \text{gridDim.x} \times \text{blockDim.x}$

The launch returns at once



An error inside the kernel shows up at the sync, not at the launch line.

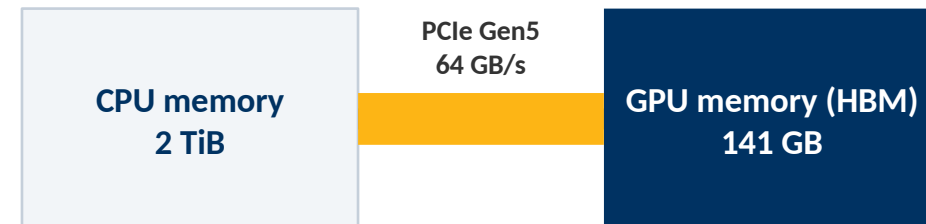
CK(cudaGetLastError()): bad launch config
 CK(cudaDeviceSynchronize()): illegal address

Example 3: two address spaces, and what a copy costs

```
// Pinned host memory: DMA-able, ~2x faster copies
float* h; cudaMallocHost(&h, bytes);

cudaStream_t s; cudaStreamCreate(&s);
cudaMemcpyAsync(d_in, h, bytes, cudaMemcpyHostToDevice, s);
kernel<<<grid, block, 0, s>>>(d_in, d_out);
cudaMemcpyAsync(h, d_out, bytes, cudaMemcpyDeviceToHost, s);
cudaStreamSynchronize(s);           // the only blocking call

// Grace / Vera + NVLink-C2C: the CPU memory is a
// coherent 480-1,500 GB extension, no copies at all
```



The GPU reads its own HBM at 4.8 TB/s, 75x faster than this pipe. Copying 2 TiB takes about 30 s.

Pinned memory + cudaMemcpyAsync: the copy engine moves data while the GPU computes, and copies run about 2x faster

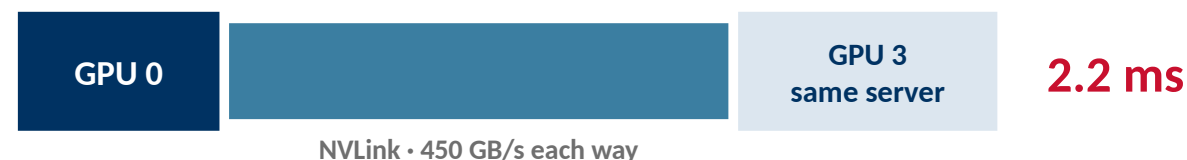
Grace / Vera CPUs: NVLink-C2C at 900 GB/s and one coherent memory. No copies at all

Moving 1 GB: inside one server vs across the network

One H200 server (like AWS p5en): 8 GPUs, a CPU, network cards. Move **1 GB** out of GPU 0.

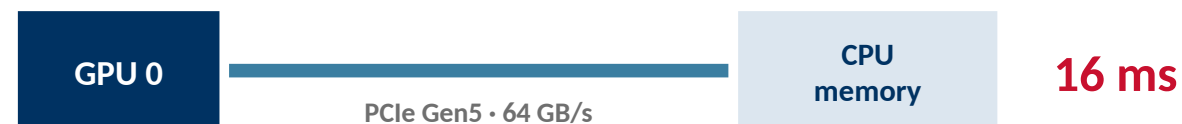
1 To another GPU in the same server: NVLink

$1 \text{ GB} \div 450 \text{ GB/s (each way)} = \mathbf{2.2 \text{ ms}}$



2 To the CPU's memory: PCIe Gen5

$1 \text{ GB} \div 64 \text{ GB/s} = \mathbf{16 \text{ ms}}$, the copy cost from Example 3



3 To a GPU in another server: the network

$400 \text{ Gb/s} = 50 \text{ GB/s} \rightarrow \mathbf{20 \text{ ms}}$, 9x slower than NVLink



4 So place the work by how much it talks

split one layer across GPUs $\rightarrow$ inside one server
each server trains its own copy $\rightarrow$ across servers

Inside one server, GPUs talk about 9x faster than across servers. Where the communication happens decides the speed.

Source: [H200 \(NVLink, PCIe Gen5\)](#); [AWS p5en \(EFA per GPU\)](#)

GPUs: from CUDA to the gigawatt rack | October 2026

Example 4: access patterns. Coalescing is the first optimization

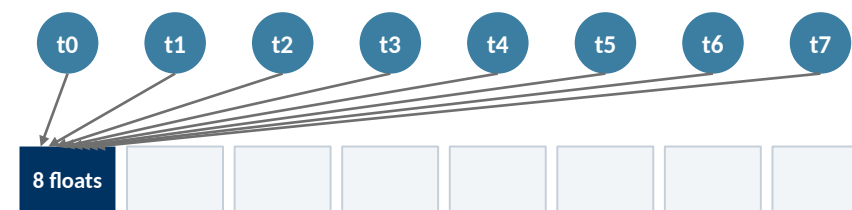
```
// 32 threads of a warp issue one load instruction.
// Consecutive addresses -> 4 x 32 B sectors, one request
out[i] = in[i];

// Stride N floats -> 32 separate sectors, 1/8 efficiency
out[i] = in[i * N];

// Array of structs: x,y,z interleaved, 3x the traffic
struct P { float x, y, z; } p[n];
float r = p[i].x;

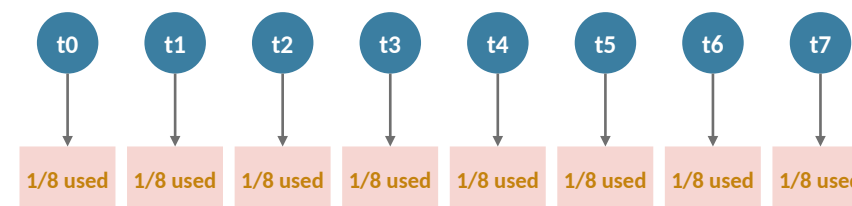
// Struct of arrays: each field contiguous
struct SoA { float *x, *y, *z; } q;
float r = q.x[i];
```

Coalesced: thread k reads float k



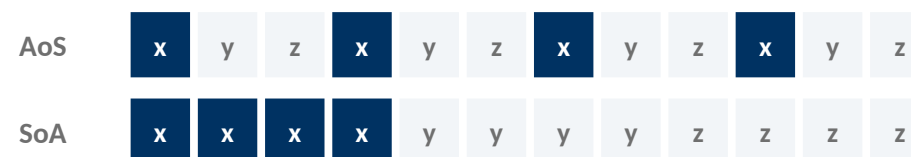
1 sector of 32 B: one memory transaction

Strided: thread k reads float 8k



8 sectors for the same 8 floats: 8x the traffic

Layout: keep each field contiguous



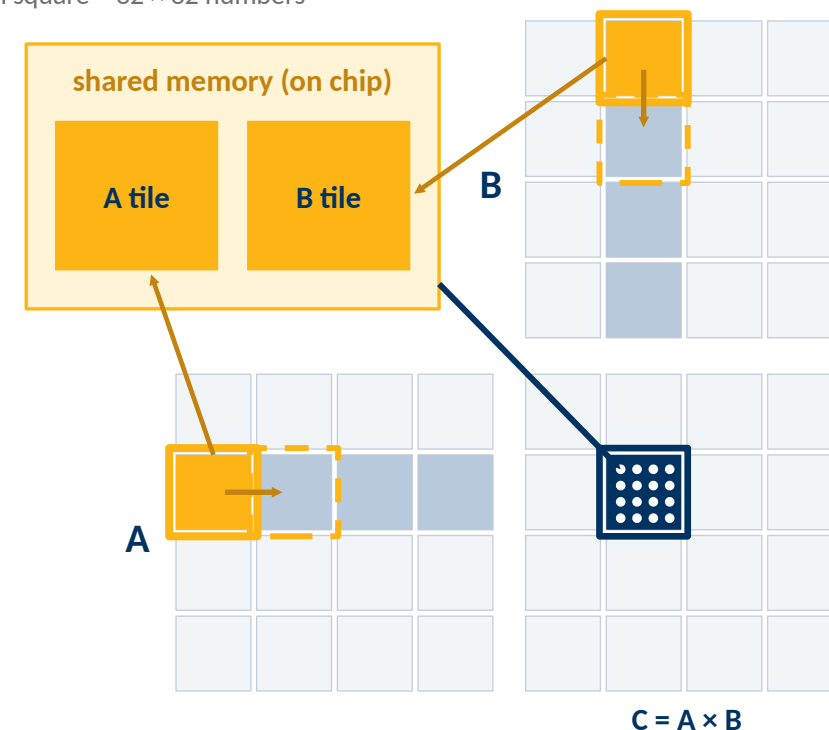
Example 5: shared memory tiling. Load once, reuse 32 times

```
#define T 32
__global__ void matmul(const float* A, const float* B,
                      float* C, int n) {    // n % T == 0

    __shared__ float As[T][T], Bs[T][T];
    int row = blockIdx.y * T + threadIdx.y;
    int col = blockIdx.x * T + threadIdx.x;
    float acc = 0.f;
    for (int k0 = 0; k0 < n; k0 += T) {
        As[threadIdx.y][threadIdx.x] = A[row * n + k0 + threadIdx.x];
        Bs[threadIdx.y][threadIdx.x] = B[(k0 + threadIdx.y) * n + col];
        __syncthreads();                // tile is complete
        for (int k = 0; k < T; k++)
            acc += As[threadIdx.y][k] * Bs[k][threadIdx.x];
        __syncthreads();                // before overwriting
    }
    C[row * n + col] = acc;
}
dim3 block(T, T), grid(n / T, n / T);
matmul<<<grid, block>>>(A, B, C, n);
```

One thread block = one tile of C

each square = 32×32 numbers



- 1 one block = one tile of C: $32 \times 32 = 1,024$ threads
- 2 it needs one row of A tiles and one column of B tiles
- 3 copy 1 A tile + 1 B tile into shared memory, then `__syncthreads()`
- 4 all 1,024 threads read both tiles: each value used 32 times
- 5 `__syncthreads()`, move to the next pair, repeat $n \div 32$ times

Without shared memory: every loop step fetches one value of A and one of B from GPU memory (HBM), 2×4 bytes, for 2 FLOP = **0.25 FLOP per byte**

With the tiles: each thread copies 2 values (8 bytes) once, then does 32 steps on the tiles (64 FLOP) = **8 FLOP per byte: 32x less memory traffic**

Example 6: warps are the real unit. Shuffles, atomics, divergence

```

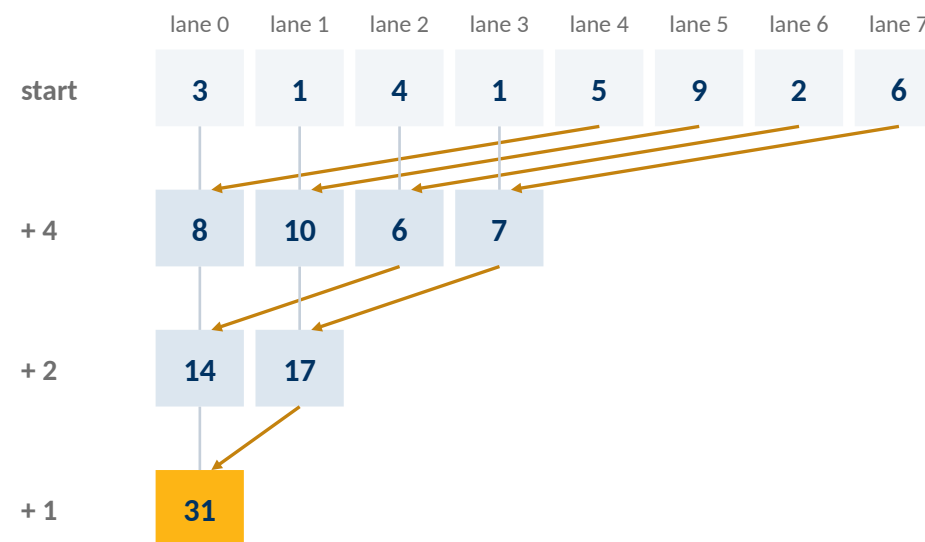
__device__ float warp_sum(float v) {
    for (int o = 16; o > 0; o >>= 1)    // 5 steps
        v += __shfl_down_sync(0xffffffff, v, o);
    return v;                            // lane 0 has it
}

__global__ void sum(const float* x, float* out, int n) {
    float v = 0.f;
    for (int i = blockIdx.x * blockDim.x + threadIdx.x;
         i < n; i += gridDim.x * blockDim.x)
        v += x[i];
    v = warp_sum(v);
    if ((threadIdx.x & 31) == 0) atomicAdd(out, v);
}

// Divergence: both paths run, the other lanes idle
if (threadIdx.x % 2) f(); else g();    // 2x slower
if (blockIdx.x % 2)  f(); else g();    // free

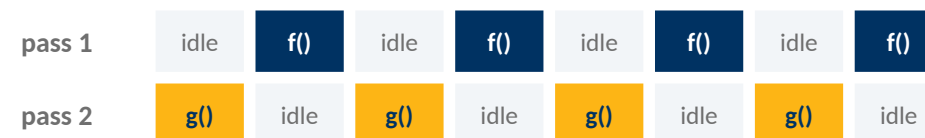
```

warp_sum: 8 of the 32 lanes shown



lane 0 now holds the sum (31). 32 lanes need 5 steps: 16, 8, 4, 2, 1

if (threadIdx.x % 2): two paths in one warp



2 passes: 2x slower. if (blockIdx.x % 2): whole warp agrees, free

Example 7: occupancy and launch shape

```
// The compiler reports registers per thread:
// nvcc --ptxas-options=-v kernel.cu
// ptxas info: Used 96 registers, 0 bytes spill
__global__ void kernel(float* x) { ... }

// Ask it to fit 3 blocks of 256 threads per SM:
__global__ void __launch_bounds__(256, 3)
kernel(float* x) { ... } // now at most 80 registers
```

1 One block: 256 threads × 96 registers each

= **24,576 registers**. One SM has 65,536.

2 How many blocks fit on one SM?

$65,536 \div 24,576 = 2.7 \rightarrow$ **2 blocks** = 16 of 64 warps: **25% occupancy**

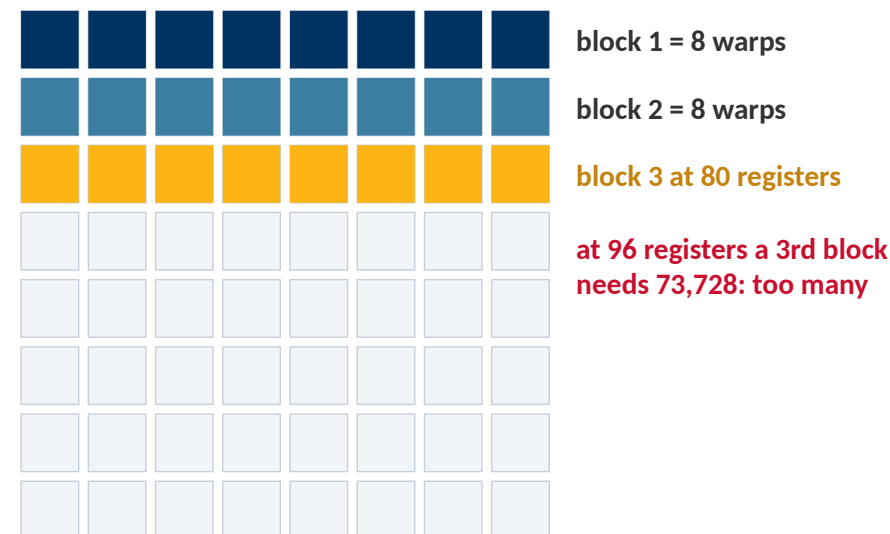
3 `__launch_bounds__(256, 3)`: ask for 3 blocks

$65,536 \div (3 \times 256) = 85 \rightarrow$ **80** (handed out in steps of 8) $\rightarrow$ 24 warps: **37.5%**

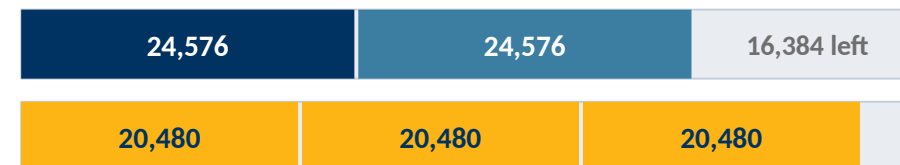
4 Which is better: more warps, or more registers per thread?

Both use all 65,536 registers. Squeeze too hard and values spill to slow memory.
Next slide: which one hides the memory wait better?

One SM: 64 warp slots



registers in one SM: 65,536



at 80 registers: 3 blocks use 61,440

Example 8: streams, events and graphs. The CPU is still in charge

```
// Overlap copy and compute with two streams
for (int k = 0; k < 2; k++) {
    cudaMemcpyAsync(d[k], h[k], B, H2D, s[k]);
    work<<<g, b, 0, s[k]>>>(d[k]);
    cudaMemcpyAsync(h[k], d[k], B, D2H, s[k]);
}

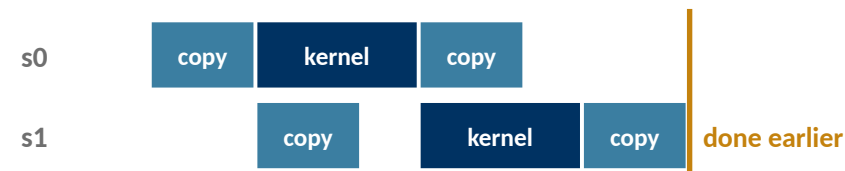
// Time GPU work with events, not with the CPU clock
cudaEventRecord(t0, s[0]); work<<<g, b, 0, s[0]>>>(d[0]);
cudaEventRecord(t1, s[0]); cudaEventSynchronize(t1);
cudaEventElapsedTime(&ms, t0, t1);

// CUDA Graphs: capture 60 launches, replay with one call
cudaStreamBeginCapture(s[0], cudaStreamCaptureModeGlobal);
for (int l = 0; l < 60; l++) layer<<<g, b, 0, s[0]>>>(...);
cudaStreamEndCapture(s[0], &graph);
cudaGraphInstantiate(&exec, graph, 0);
cudaGraphLaunch(exec, s[0]); // ~1 launch overhead
```

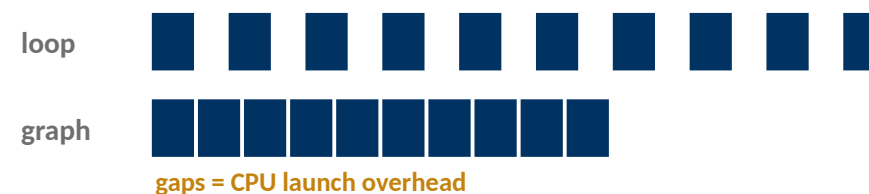
One stream: everything waits in line



Two streams: copies overlap with compute



CUDA Graph: record 60 launches, replay once



Small decode kernels can spend more time waiting to launch than running. One graph launch removes most of the gaps.

CPU vs CUDA: guess the CUDA column, one row at a time

	CPU (C++, OpenMP)	CUDA: what changes? Guess first
Unit of parallelism	a thread per core, tens of them	a thread per element, millions, in warps of 32
Memory	one address space, caches are implicit	host and device, shared memory is explicit, caches help but do not save you
Latency	hidden by caches, OoO, prefetch	hidden by other warps. Low occupancy = exposed latency
Branches	cheap, predicted	cheap if uniform per warp, 2x cost if divergent
Synchronization	locks, atomics, futures	block barriers, warp shuffles, atomics, grid sync via cooperative launch
Performance model	cache misses, branch mispredicts, IPC	bytes moved per FLOP, occupancy, coalescing, launch count
Failure modes	segfault with a stack trace	illegal address reported later, hangs in barriers, silent corruption
Libraries	BLAS, STL, your own loops	cuBLAS, cuDNN, CUTLASS, Thrust/CUB, NCCL, Triton. Write a kernel last



Chapter 4

From one GPU to the AI factory

30 minutes. Five generations, training time, 72-GPU racks, failures, power

One model, five GPU generations: what each step bought

Same model on every GPU: Llama 3.1 70B with 8-bit weights (**70 GB**), chats of 8,000 tokens (**2.6 GB** each).

GPU	tokens/s for one user = bandwidth ÷ 70 GB	users that fit = (memory – 70 GB) ÷ 2.6 GB	what changed
H100 • 2022 80 GB • 3.35 TB/s	 48	 3 users	baseline
H200 • 2024 141 GB • 4.8 TB/s	 69	 27 users	capacity + speed
B200 • 2024–25 180 GB • 8 TB/s	 114	 41 users	speed
B300 • 2025 288 GB • 8 TB/s	 114	 83 users	Capacity + speed
Rubin • 2026 288 GB • 19–22 TB/s	 270–310	 83 users	speed (HBM4)

Bandwidth sets how fast one user is served. Capacity sets how many users share the GPU. Each generation moved one of the two, or both.

Source: [H100](#), [H200](#), [HGX B200 and B300](#), [Vera Rubin NVL72](#) (nvidia.com); model: [Llama 3.1 70B config](#)

GPUs: from CUDA to the gigawatt rack | October 2026

Your turn: guess first



Meta trained Llama 3 405B on 16,384 H100s. What share of the H100's peak FLOPS did the training actually use?



about 90%



about 70%



about 40%



about 10%

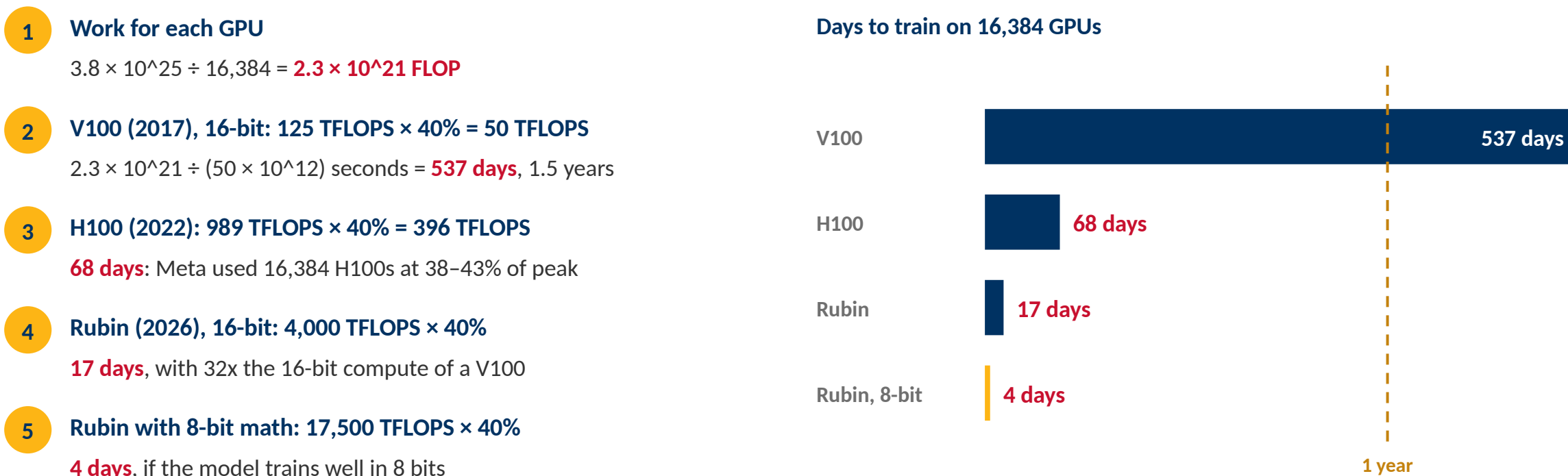
C. 38–43% MFU: 380–430 of 989 TFLOPS per GPU. Many jobs run at 25–35%.

Click to show the answer. Sources on the next page.

GPUs: from CUDA to the gigawatt rack | October 2026

What 32x more compute buys: training Llama 3 405B

Training Llama 3 405B took 3.8×10^{25} FLOP (Meta). Same **16,384 GPUs** each time, each running at **40%** of its peak.



The work is fixed. Faster chips (32x) and fewer bits (4x more) shrink the calendar from 1.5 years to a few days.

Source: [Meta, The Llama 3 Herd of Models \(arxiv.org/abs/2407.21783\)](https://arxiv.org/abs/2407.21783); peaks: [V100](#), [H100](#), [Vera Rubin NVL72](#)

GPUs: from CUDA to the gigawatt rack | October 2026

Your turn: guess first



Inside an NVL72 rack, NVLink gives each GPU 900 GB/s per direction. How much more is that than one 400 Gb/s network port?



about the same



2x



18x



100x

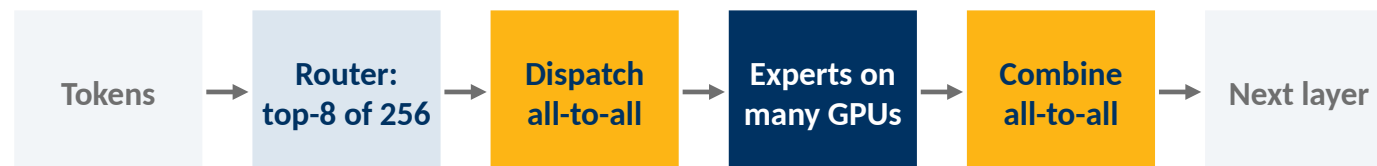
C. 400 Gb/s is 50 GB/s (bits, not bytes), so NVLink is 18x. That is why MoE expert traffic stays inside the rack.

Click to show the answer. Sources on the next page.

GPUs: from CUDA to the gigawatt rack | October 2026

Mixture of experts turns the network into a kernel

One MoE layer: the network sits inside every layer



One dispatch, by the numbers (DeepSeek-V3 sizes):



...twice per layer (dispatch and combine), in each of 61 layers.

Why NVL72 helps

NVLink inside an NVL72 rack



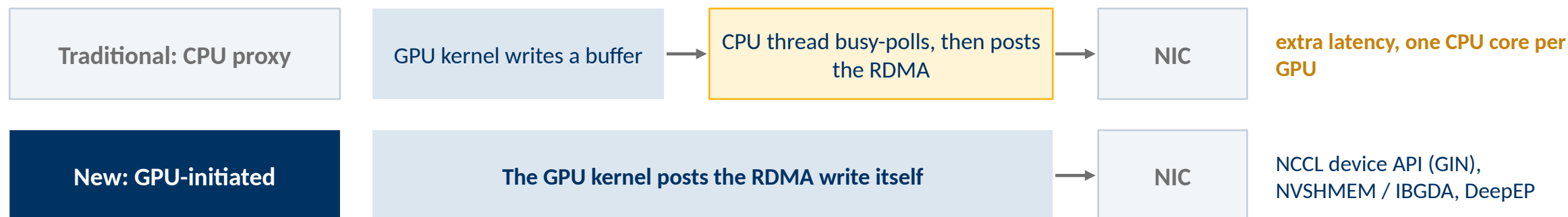
One 400 Gb/s network port



18x

more bandwidth per direction for the hottest collective in the model

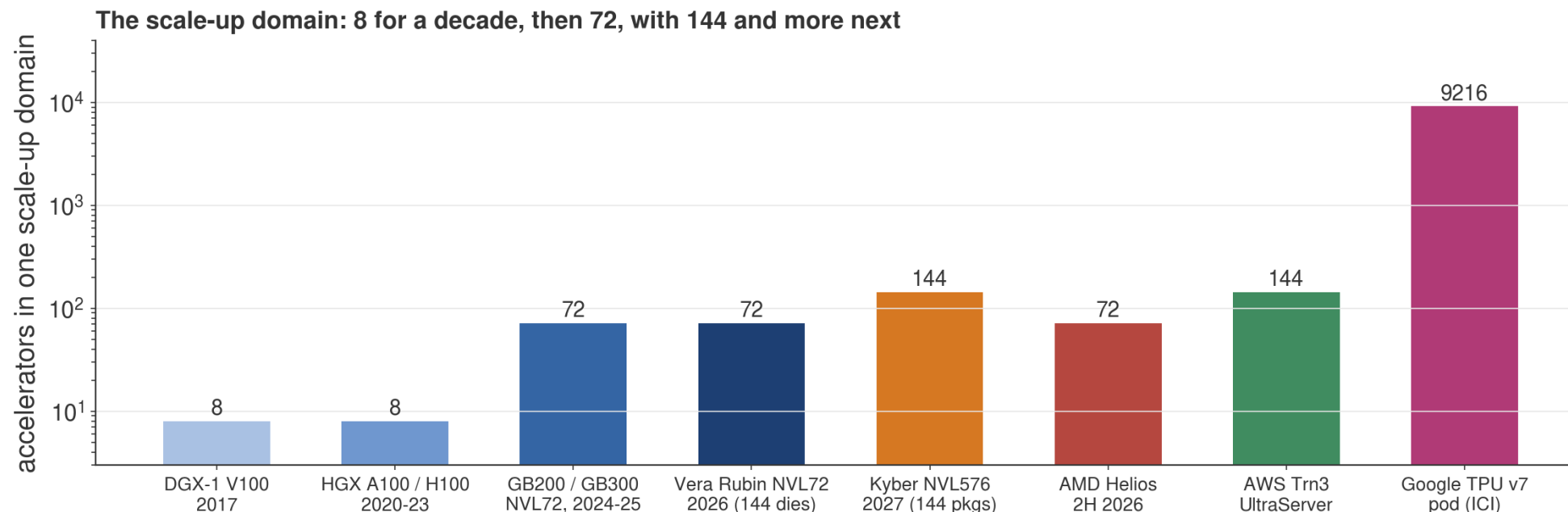
Who posts the network write?



Source: DeepSeek-V3 technical report arxiv.org/abs/2412.19437; NCCL docs.nvidia.com/deeplearning/nccl/; DeepEP github.com/deepseek-ai/DeepEP

GPUs: from CUDA to the gigawatt rack | October 2026

The scale-up domain: 8 for a decade, then 72, then 144 and more



- NVLink domains stayed at 8 GPUs from DGX-1 (2017) to HGX H100. GB200 NVL72 made 72 the unit in 2024, Rubin keeps 72 packages (144 dies), Kyber targets 144 packages in 2027.
- Others chose the same direction with different fabrics: AMD Helios 72 on UALink, Trainium3 UltraServer 144 on NeuronSwitch, TPU pods of 9,216 on ICI. The rack is the product.

Source: nvidia.com/en-us/data-center/hgx/, [gb200-nvl72/](https://nvidia.com/en-us/data-center/gb200-nvl72/), [vera-rubin-nvl72/](https://nvidia.com/en-us/data-center/vera-rubin-nvl72/); OCP 2025 blog (Kyber); amd.com/en/products/accelerators/instinct/mi400.html; aws.amazon.com/ai/machine-learning/trainium/; Ironwood (blog.google)

Your turn: guess first



Meta trained Llama 3 405B on 16,384 H100s for 54 days. How many times was the job interrupted?



about 5



about 50



about 470



about 5,000

C. 466 interruptions, 419 of them unexpected: about one every 3 hours. Automation still kept effective training time above 90%.

Click to show the answer. Sources on the next page.

GPUs: from CUDA to the gigawatt rack | October 2026

Failures: once in 6 years per GPU, every 3 hours per job

Meta's Llama 3 405B run: 16,384 H100s, a 54-day period, 419 unexpected interruptions.

1 For the whole job

54 days × 24 h ÷ 419 = one every **3.1 hours**

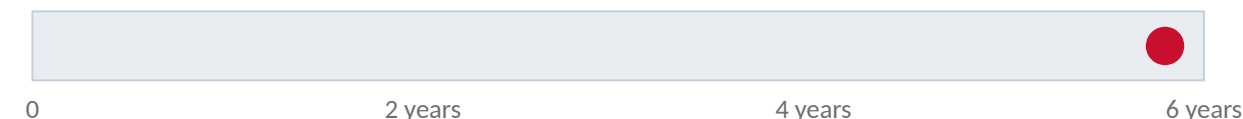
one day of the 16,384-GPU job: about 8



2 For one GPU

16,384 × 54 = 884,736 GPU-days ÷ 419 = one per **5.8 years**

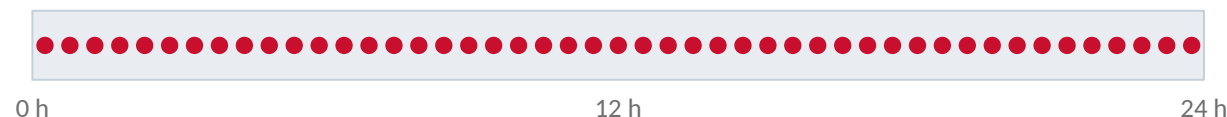
one GPU over six years: about 1



3 With 100,000 GPUs

one per 2,100 GPU-days → **one every 30 minutes**

one day with 100,000 GPUs: about 47



4 So operators automate

health checks before each job, frequent checkpoints, spare nodes, automatic restart: Meta kept over 90% of the time productive

At scale, failure is a rate, not an event. The software that detects, drains and restarts is part of the machine.

Source: [Meta, The Llama 3 Herd of Models, section 3.3.4](#)

GPUs: from CUDA to the gigawatt rack | October 2026

Power is the budget: 100 MW buys about 40,000 GPUs

NVIDIA's planning figure for a Rubin AI factory: **100 MW** of power for about **40,000 GPUs**.

1 Power per GPU, whole factory

100 MW ÷ 40,000 GPUs = **2.5 kW** per GPU

Power per GPU

Rubin, whole factory  **2.5 kW**

GB300 chip  **1.4 kW**

H200 chip  **0.7 kW**

2 The chip is only part of it

H200 chip 0.7 kW, GB300 chip 1.4 kW. The rest: CPUs, network, cooling, power conversion

3 A 1-gigawatt site: 1 GW = 1,000 MW

1,000 MW = 1,000,000 kW ÷ 2.5 kW = **400,000 GPUs**

1 GW = 1,000 MW = 10 blocks of 100 MW, about 40k GPUs each



= 400,000 GPUs

4 So the question changes

not 'how many GPUs can I buy?' but 'how many megawatts can I connect, and how many tokens does each one make?'

Power, not chips, is what a data center runs out of. A GPU that is 30% slower but 50% more efficient per watt can be the better buy.

Source: [Vera Rubin NVL72 \(40,000 GPUs per 100 MW\)](#); [H200](#); [GB300 NVL72](#)

GPUs: from CUDA to the gigawatt rack | October 2026

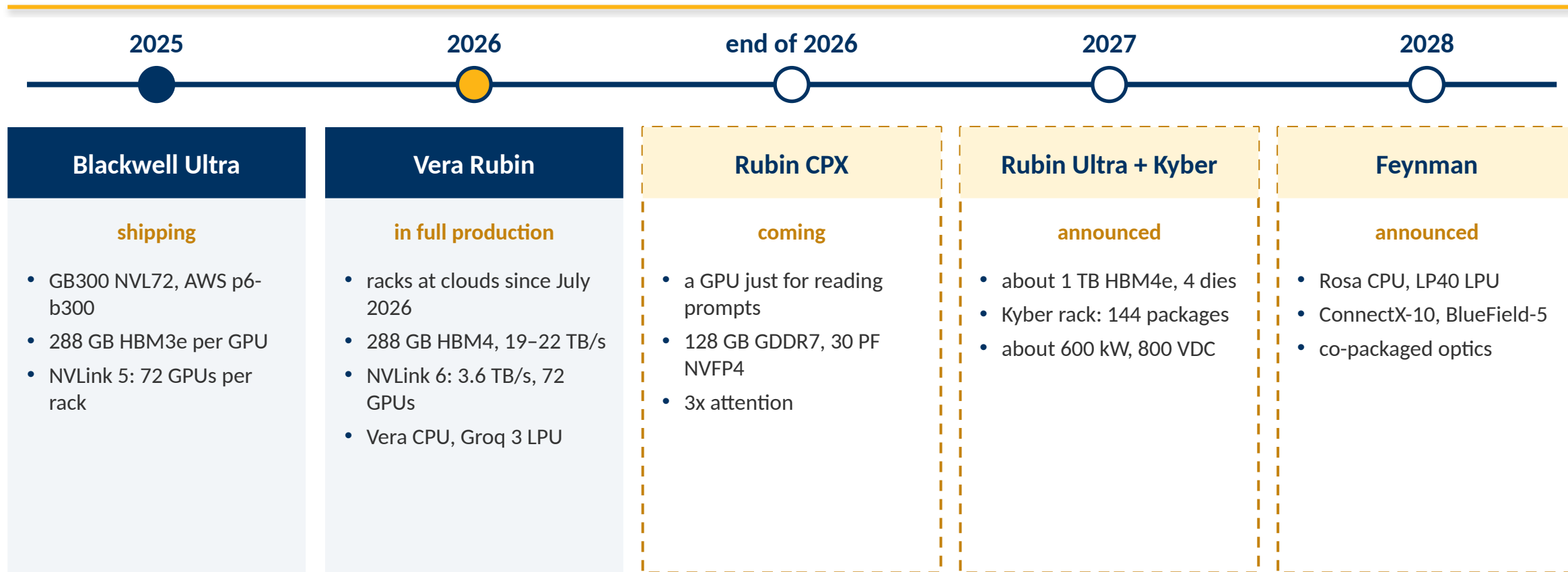


Chapter 5

Where it is going, and where you fit

10 minutes. The public roadmap, the software stack, skills

The public roadmap, October 2026



Right of Vera Rubin: announced, not shipped. Numbers can still change (NVL144 became NVL72).

Source: nvidianews.nvidia.com/news/nvidia-vera-rubin-platform; [GTC 2026 blog](#), [OCP 2025 blog](#), [OCP 2026 blog](#), [GTC 2025 keynote blog](#) (blogs.nvidia.com); [Rubin CPX release](#)

GPUs: from CUDA to the gigawatt rack | October 2026

Software: twenty years of CUDA, and where the stack is moving

Serving	TensorRT-LLM, Dynamo (NVIDIA) · vLLM, SGLang (community): the new battleground
Frameworks	PyTorch with torch.compile (Inductor) · JAX
Kernel authoring	Triton : write a kernel in Python one tile at a time, and the compiler handles threads, shared memory and syncs · CUTLASS · raw CUDA for the last 20%
Libraries: the moat	cuBLAS, cuDNN, CUTLASS, NCCL, TensorRT, Nsight, 1,000+ CUDA-X libraries
Communication	NCCL 2.28+ device API: LSA over NVLink, GIN over the network · DeepEP, NVSHMEM
CUDA, 20 years old	day-one support for every new GPU generation (CUDA turned 20 at GTC 2026)

Portable

vLLM and SGLang on ROCm and Trainium, Triton on NVIDIA and AMD, JAX on TPUs

1.6x

up to 1.6x on the same GB300 NVL72 rack, same Qwen3-VL test, April → September 2026 (MLPerf v6.0 → v6.1). Only the software changed: lower-precision KV cache, fused and better kernels, prefill/decode split.

Vendor-specific

custom kernels, GPU-initiated communication

Source: blogs.nvidia.com/blog/gtc-2026-news; [MLPerf v6.1 blog](#); docs.nvidia.com/deeplearning/nccl/release-notes; [vLLM](#), [SGLang](#), [Triton](#) on GitHub

Strategy for a developer

1 Think in bytes per FLOP

Place the kernel on the roofline first. Most LLM serving is memory-bound.



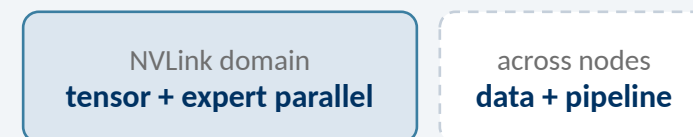
2 Use the stack. write CUDA last

Write CUDA when the profile says so, then use CUTLASS/CuTe, not raw PTX.



3 Design for the rack

Know the 9–18x bandwidth gap per direction, and the 10x latency gap.



4 Learn the precision recipes

FP8 with per-tensor or per-block scales today, NVFP4 with block-16 scales next. Validate on your tasks, not on perplexity alone.

5 Keep the CPU out of the loop

Launch and Python overhead starve decode-size kernels. CUDA Graphs, fusion, persistent kernels, GPU-initiated communication.

6 Build for heterogeneity

H200 today, Rubin next year. Framework- and Triton-level code outlives the last 10% on one part.

Your turn: remember?



Fill in the three numbers from the start of the talk

4.8 TB/s vs ____ GB/s

____ GPUs in one NVLink
domain

____ kW per GPU, factory
level

50 GB/s · 72 GPUs · 2.5 kW. Memory vs network, the rack, and power: the three threads of this talk.

Click to show the answers. Numbers from page 3.

GPUs: from CUDA to the gigawatt rack | October 2026

Takeaways

1 Why GPUs

A CPU makes one instruction stream fast. A GPU keeps 270,000 threads in flight, so waiting on memory never stops the work.

2 One GPU

An LLM answering one user is a memory-reading machine: $\text{tokens/s} \approx \text{bandwidth} \div \text{model bytes}$. Batching and fewer bits raise it.

3 Writing CUDA

One thread per element, data placed on purpose, the CPU kept out of the loop. Use the libraries first. Write kernels where the profile says so.

4 The AI factory

Bandwidth sets speed, capacity sets users, NVLink sets how many GPUs act as one, and power sets how big the fleet can be.

5 Where you fit

Every layer needs engineers: kernels and compilers, serving systems, networks, reliability, data-center power.

Three numbers: **4.8 TB/s vs 50 GB/s** (memory vs network) · **72 GPUs** in one NVLink domain · **2.5 kW** per GPU

Questions

Three numbers: 4.8 TB/s vs 50 GB/s. 72 GPUs per domain. 2.5 kW per GPU.

- Sources: the source line on each slide, and the full list of public pages in A3.
- All figures are public NVIDIA, AWS, AMD, Google or Meta material fetched on 2026-10-16.



Backup

Reference slides
for questions

What Hopper and Blackwell added, and why you will mostly use them through libraries

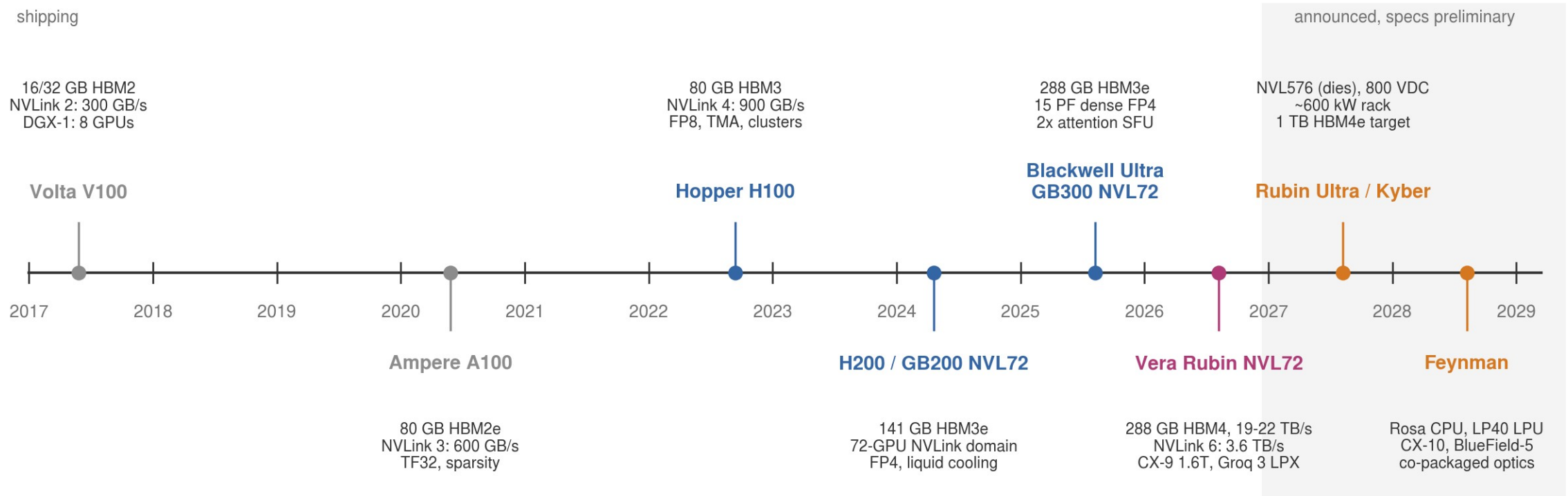
```
// Hopper: thread block clusters + distributed shared memory
__global__ void __cluster_dims__(2, 1, 1) k(float* g) {
    namespace cg = cooperative_groups;
    cg::cluster_group cluster = cg::this_cluster();
    __shared__ float buf[1024];
    buf[threadIdx.x] = g[...];
    cluster.sync();
    // read the neighbour block's shared memory directly
    float* peer = cluster.map_shared_rank(buf, 1);
    float v = peer[threadIdx.x];
    cluster.sync();
}

// Hopper: TMA copies whole tiles HBM <-> shared memory,
//   one thread issues, hardware completes (cp.async.bulk)
// Hopper: wgmma, warp-group (128 thread) tensor core MMA
// Blackwell: tcgen05 MMA, 256 KB Tensor Memory per SM,
//   two SMs cooperating on one MMA (pair mode), FP4 / FP6
```

- **Thread block clusters:** up to 16 blocks scheduled together on one GPC, with each other's shared memory addressable (DSMEM). A block-to-block exchange without going through HBM.
- **TMA (Tensor Memory Accelerator):** a copy engine that moves multi-dimensional tiles between HBM and shared memory asynchronously. Frees the warps to compute.
- **Warp-group MMA (wgmma) and Blackwell tcgen05:** tensor core instructions issued by 128 threads with operands in shared memory and, on Blackwell, results in Tensor Memory.
- These are what CUTLASS 3/4, CuTe, Triton and the vendor GEMMs are built from. Writing them by hand is expert work. Knowing they exist tells you why a Hopper GEMM is written differently from an Ampere one.

Source: CUDA guide (clusters, TMA): docs.nvidia.com/cuda/cuda-c-programming-guide; Inside NVIDIA Blackwell Ultra (TMEM, pair MMA): developer.nvidia.com/blog

Nine years in one line: Volta to Feynman



- Three things grew: memory per GPU (16 GB to 288 GB, 1 TB targeted), the NVLink domain (8 to 72 and beyond), power per rack (10 kW boxes to 600 kW liquid-cooled racks).
- The cadence went from one architecture every two years to one every year since Blackwell, with an 'Ultra' refresh in between. Items right of 2026 are announced targets.

Source: nvidia.com/en-us/data-center/hgx; [GTC 2026 blog](#), [OCP 2025 blog](#), [OCP 2026 blog](#) (blogs.nvidia.com). Items right of 2026 are announced targets

H200: Hopper with the memory it should have had

	H200 SXM (GH100 die, HBM3e)
Compute	132 SMs, 16,896 FP32 lanes, 528 4th-gen Tensor Cores, 80 B transistors, TSMC 4N
Dense tensor	BF16 989 TFLOPS, FP8 1,979 TFLOPS, TF32 495, FP64 TC 67. No FP4
Memory	141 GB HBM3e at 4.8 TB/s (H100: 80 GB at 3.35 TB/s). 1.76x capacity, 1.43x bandwidth
Interconnect	NVLink 4: 900 GB/s bidirectional, 8 GPUs per NVSwitch domain. PCIe Gen5 x16
Power	700 W configurable, air or liquid
On AWS	p5en.48xlarge: 8x H200 (1,128 GB), 3.2 Tb/s EFAv3, 192 vCPU, 2 TiB, 8x 3.84 TB NVMe
Role in 2026	the workhorse for training at BF16/FP8 and for serving models that fit 1.1 TB per node. Same compute as H100, so it wins exactly where H100 was memory-bound: decode, long context

- NVIDIA's own numbers: Llama 2 70B inference 1.9x faster than H100 at the same 700 W, from memory alone.
- Where it hurts: no FP4, half the FP8 of Blackwell, 8-GPU NVLink domain. A 671B MoE at FP8 needs the whole node and then some.

Source: nvidia.com/en-us/data-center/h200; aws.amazon.com/ec2/instance-types/p5

GPUs: from CUDA to the gigawatt rack | October 2026

B200 and GB200: two dies, FP4, and the 72-GPU rack

	B200 (HGX, x86 host)	GB200 (NVL72, Grace host)
Silicon	Blackwell: 2 reticle-limited dies, 208 B transistors, 10 TB/s NV-HBI, TSMC 4NP	same GPU, higher clocks and power
Dense tensor per GPU	FP4 9 PF, FP8 4.5 PF, BF16 2.25 PF, FP64 37 TF	FP4 10 PF, FP8 5 PF, BF16 2.5 PF, FP64 40 TF
Memory per GPU	180 GB HBM3e, 8 TB/s	186 GB HBM3e, 8 TB/s
Interconnect	NVLink 5: 1.8 TB/s, 8 GPUs per node	NVLink 5 switch: 72 GPUs, 130 TB/s total
Host	2x Xeon (Emerald Rapids), PCIe Gen5	36 Grace CPUs, NVLink-C2C 900 GB/s, 17 TB LPDDR5X
Power	up to 1,000 W per GPU, 14.3 kW per 8-GPU system, air-cooled	1,200 W per GPU, about 120 kW per rack, liquid-cooled
On AWS	p6-b200.48xlarge: 8 GPUs, 1,440 GB, 3.2 Tb/s EFAv4	p6e-gb200 UltraServer: 72 GPUs, 13.4 TB, 28.8 Tb/s EFAv4

- What Blackwell changed: the second-generation Transformer Engine with FP4 (micro-tensor scaling), 2x the NVLink bandwidth, a decompression engine, confidential computing over NVLink, a RAS engine.
- What the rack changed: 72 GPUs in one NVLink domain makes expert parallelism and distributed KV cache a one-rack problem. That is the design center for inference since 2024.

Source: nvidia.com/en-us/data-center/hgx, [/dgx-b200](#), [/gb200-nvl72](#), [/technologies/blackwell-architecture](#); aws.amazon.com/ec2/instance-types/p6

B300 and GB300 (Blackwell Ultra): more memory, more FP4, less of everything else

	B300 (HGX)	GB300 (NVL72)
Silicon	same 208 B transistor dual die, 160 SMs, 640 5th-gen Tensor Cores, 256 KB Tensor Memory per SM	same
Dense tensor per GPU	FP4 13.5 PF (1.5x B200), FP8 4.5 PF, BF16 2.25 PF	FP4 15 PF, FP8 5 PF, BF16 2.5 PF
What got cut	FP64 1.25 TF (B200: 37), INT8 0.2 POPS dense (B200: 4.5)	FP64 1.4 TF per GPU
Attention	2x SFU throughput for exp and softmax: 2x attention-layer speed	same
Memory per GPU	288 GB HBM3e (1.6x B200), 8 TB/s	288 GB, 8 TB/s. 20.7 TB per rack
Network	ConnectX-8: 800 Gb/s per GPU (2x)	ConnectX-8: 800 Gb/s per GPU
Power	about 1.1 to 1.4 kW per GPU (configurable, partner figures)	1.4 kW per GPU, rack about 135 to 140 kW
On AWS	p6-b300.48xlarge: 8 GPUs, 2.1 TiB HBM, 6.4 Tb/s EFA, 4 TB DRAM, 300 Gb/s ENA	p6e-gb300 UltraServer: 1.5x memory and FP4 of p6e-gb200

- Blackwell Ultra is an inference part by design: the transistor budget moved from FP64 and INT8 to FP4, HBM capacity and attention math. HPC centers buy B200 or wait for the AMD MI430X. Serving fleets buy B300.
- The 2x network per GPU matters as much as the memory: at 800 Gb/s the all-to-all of an MoE layer takes half the time it took on B200 nodes.

Source: nvidia.com/en-us/data-center/hgx, [/gb300-nvl72](https://nvidia.com/en-us/data-center/gb300-nvl72); Inside Blackwell Ultra: developer.nvidia.com/blog; aws.amazon.com/ec2/instance-types/p6

Side by side: what each step bought (per GPU, dense)

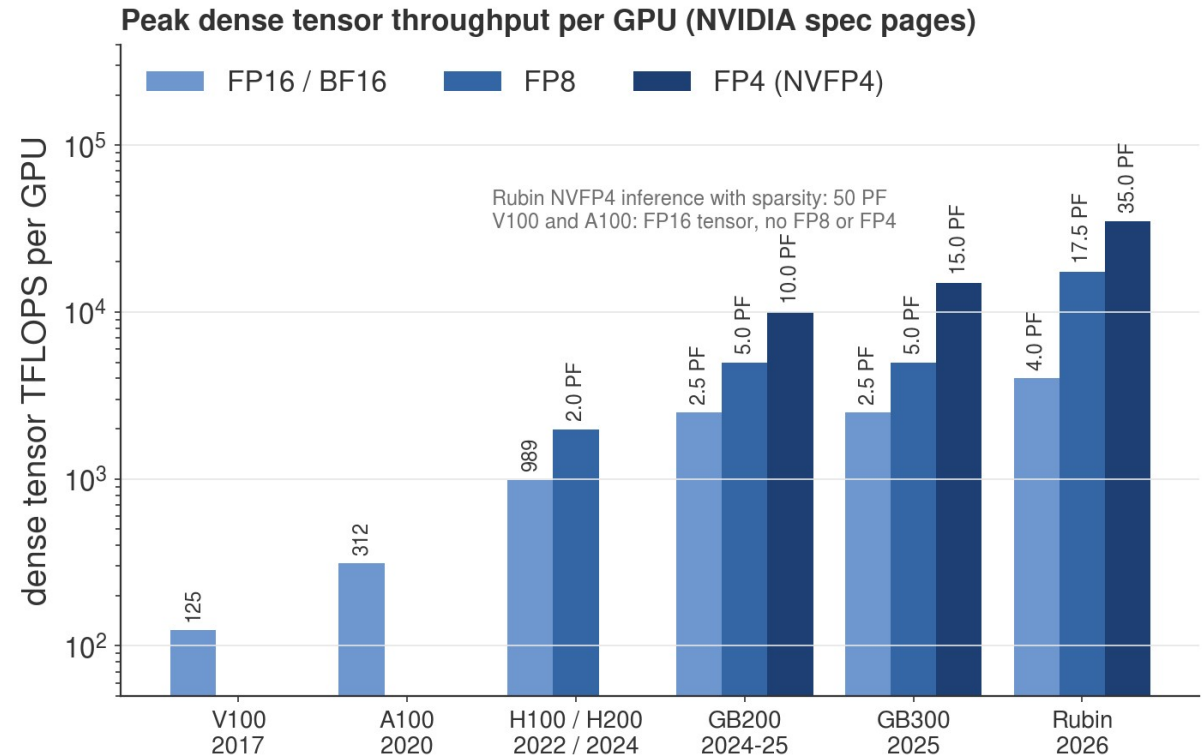
	H100	H200	B200 (HGX)	B300 (HGX)	Rubin (NVL72)
Year	2022	2024	2024-25	2025	2026
HBM	80 GB, 3.35 TB/s	141 GB, 4.8 TB/s	180 GB, 8 TB/s	288 GB, 8 TB/s	288 GB HBM4, 19-22 TB/s
BF16 dense	989 TF	989 TF	2.25 PF	2.25 PF	4 PF
FP8 dense	1.98 PF	1.98 PF	4.5 PF	4.5 PF	17.5 PF
FP4 dense	-	-	9 PF	13.5 PF	35 PF (50 sparse)
FP64	34 / 67 TF	34 / 67 TF	37 TF	1.25 TF	33 TF
NVLink per GPU	900 GB/s	900 GB/s	1.8 TB/s	1.8 TB/s	3.6 TB/s
Network per GPU (AWS)	400 Gb/s	400 Gb/s	400 Gb/s	800 Gb/s	1.6 Tb/s (CX-9)
TDP	700 W	700 W	1,000 W	1.1-1.4 kW	not published
Scale-up domain	8	8	8 (72 as GB200)	8 (72 as GB300)	72

- H100 to H200: memory only. H200 to B200: compute 2.3x, memory bandwidth 1.7x, FP4 arrives. B200 to B300: memory 1.6x and FP4 1.5x, FP64 gone. B300 to Rubin: HBM bandwidth 2.4x, FP8 3.9x, NVLink 2x.

Source: nvidia.com/en-us/data-center/h200/, [/hgx/](https://nvidia.com/en-us/data-center/hgx/), [/gb200-nvl72/](https://nvidia.com/en-us/data-center/gb200-nvl72/), [/gb300-nvl72/](https://nvidia.com/en-us/data-center/gb300-nvl72/), [/vera-rubin-nvl72/](https://nvidia.com/en-us/data-center/vera-rubin-nvl72/); aws.amazon.com/ec2/instance-types/p5/, [/p6/](https://aws.amazon.com/ec2/instance-types/p6/)

Compute per GPU: 280x in nine years, mostly by changing the number format

- BF16 dense: 125 TF (V100, FP16) to 4 PF (Rubin), 32x. FP8 added 2x on Hopper, FP4 another 2x on Blackwell. Rubin's FP8 is 3.5x its predecessor and its BF16 only 1.6x.
- Read the slope per format, not across formats. Per format the gain is about 1.5 to 2x per generation. The rest is precision.
- This is why FP8 training (Hopper) and FP4 inference (Blackwell) became mainstream so fast: they were the only way to use the new hardware.
- Rubin's spec sheet puts BF16 at 4 PF next to 17.5 PF FP8 and 35 PF NVFP4. The hardware assumes you have moved down the ladder.

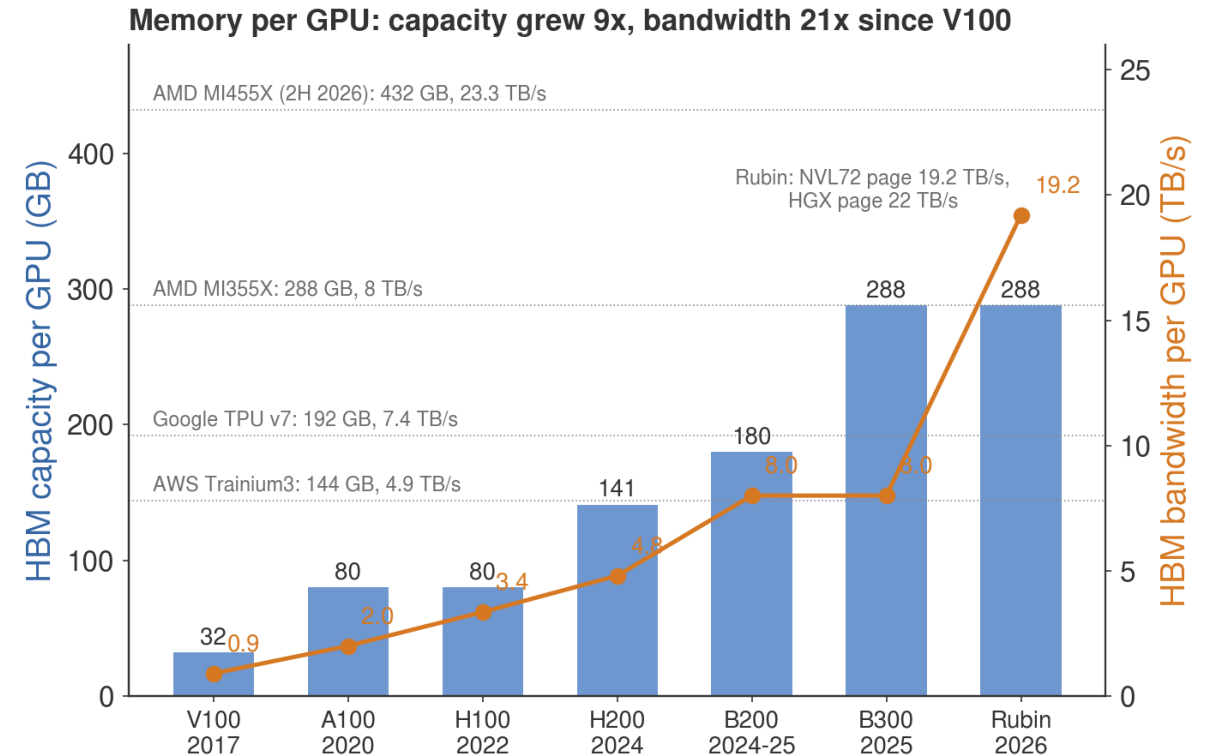


Source: nvidia.com/en-us/data-center/h200/, [/hgx/](#), [/gb200-nvl72/](#), [/gb300-nvl72/](#), [/vera-rubin-nvl72/](#), [/technologies/rubin](#)

GPUs: from CUDA to the gigawatt rack | October 2026

Memory per GPU: the spec that decides what fits

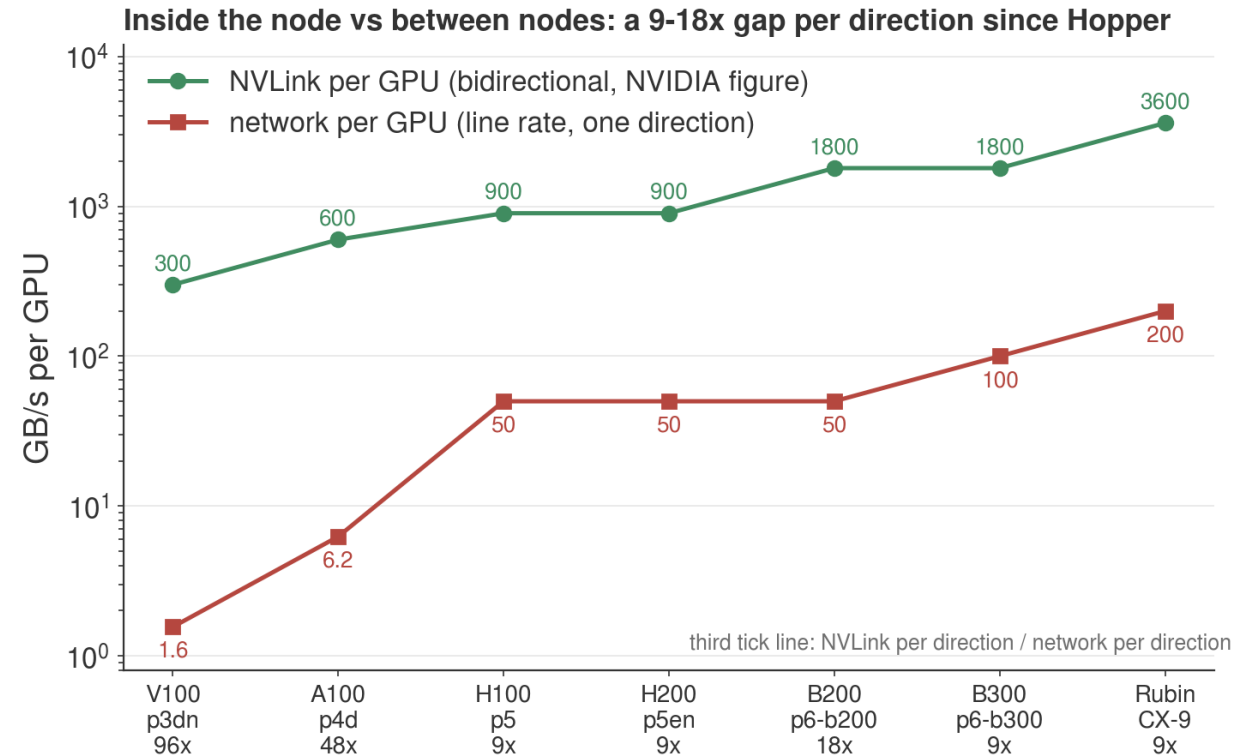
- Capacity: 32 GB (V100) to 288 GB (B300 and Rubin), with 1 TB targeted for Rubin Ultra. Bandwidth: 0.9 to 8 TB/s through Blackwell, then 19 to 22 TB/s with HBM4 on Rubin.
- The B200 to B300 step added capacity and no bandwidth. Rubin adds bandwidth and no capacity. Each step serves a different bottleneck: B300 fits more context and bigger batches, Rubin decodes faster.
- Competitors lead on capacity: AMD MI455X 432 GB HBM4 at 23.3 TB/s (2H 2026). Google and AWS chips carry less per chip and more chips per scale-up domain.
- For an operator, GB of HBM per dollar and per watt is the first filter for an inference fleet.



Source: nvidia.com/en-us/data-center/h200/, [hgx](https://nvidia.com/en-us/data-center/hgx/), [vera-rubin-nvl72](https://nvidia.com/en-us/data-center/vera-rubin-nvl72/); amd.com/en/products/accelerators/instinct/mi350.html, [mi400.html](https://amd.com/en/products/accelerators/instinct/mi400.html); Ironwood: blog.google; aws.amazon.com/ai/machine-learning/trainium

Operator view 1: the node is the unit, and its links are not equal

- Inside the node: 8 GPUs on an NVSwitch fabric at 900 GB/s (H200) or 1.8 TB/s (B200/B300) bidirectional. One hop, microseconds, load/store semantics.
- Out of the node: one 400 Gb/s port per GPU (p5en, p6-b200), 800 Gb/s on p6-b300. RDMA, tens of microseconds, 50 to 100 GB/s per direction.
- The ratio per direction has stayed at 9 to 18x since Hopper. Rubin keeps it at 9x. Software must place communication accordingly: tensor and expert parallelism inside, data and pipeline parallelism across.
- Topology inside the node matters too: a GPU and its NICs hang off the same PCIe switch or root port. Traffic to the host competes with traffic to the network.

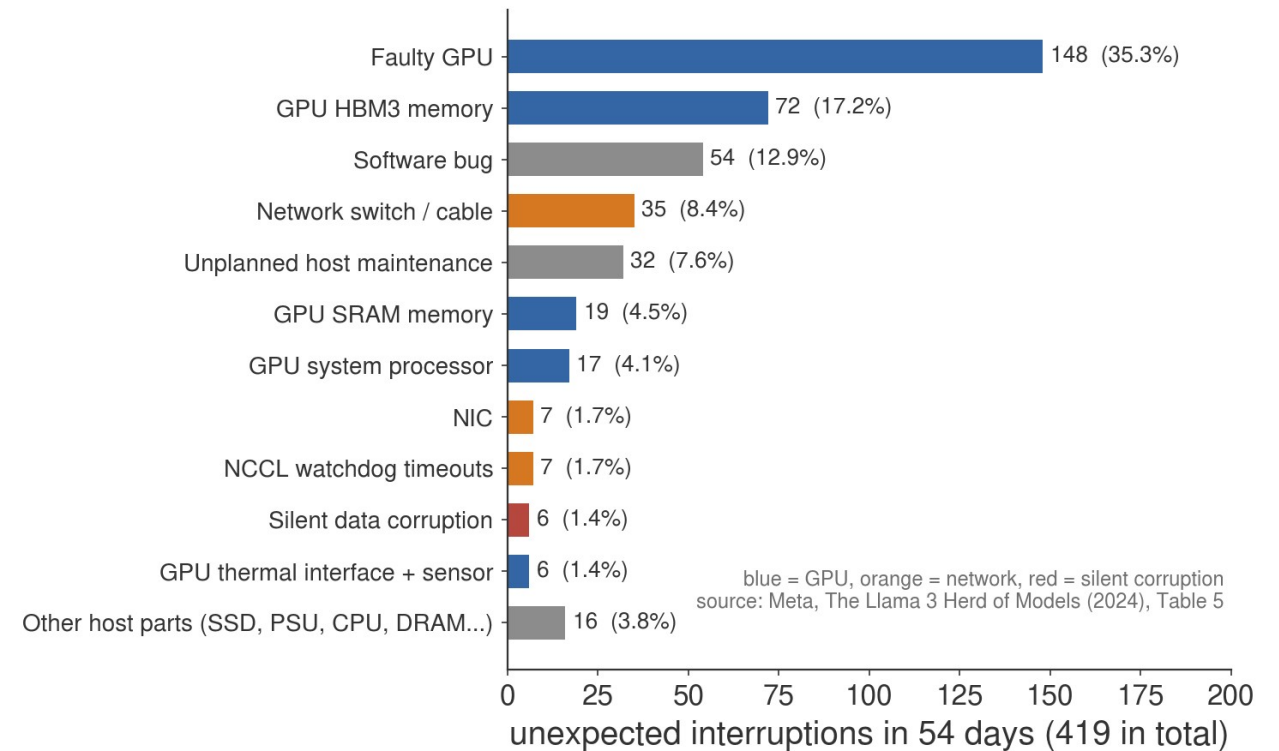


Source: nvidia.com/en-us/data-center/nvlink; aws.amazon.com/ec2/instance-types/p4, [/p5](#), [/p6](#); Spectrum-X / CX-9 for Rubin: developer.nvidia.com/blog

Operator view 2: at scale, something fails every three hours

- Meta's Llama 3 405B run: 16,384 H100s, 54 days, 466 interruptions. 419 were unexpected, 78% of those hardware. GPUs and their HBM were the largest group. Effective training time stayed above 90% through automation: 3 manual interventions in 54 days.
- Same paper: a 1 to 2% diurnal throughput swing from midday temperatures, and tens of megawatts of power swing when all GPUs pause for a checkpoint.
- The operator's tools: health checks before every job, checkpoint fast, detect stragglers, keep spares, and treat silent data corruption as a real category.

Llama 3 405B on 16,384 H100s, 54 days: GPUs caused most interruptions



Source: Meta, The Llama 3 Herd of Models, section 3.3.4: arxiv.org/abs/2407.21783

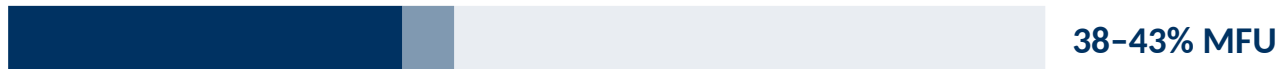
Operator view 3: utilization is the product you sell

Share of peak FLOPS actually used

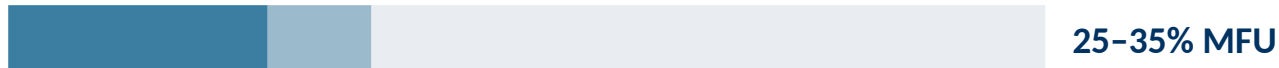
H100 peak, BF16 dense



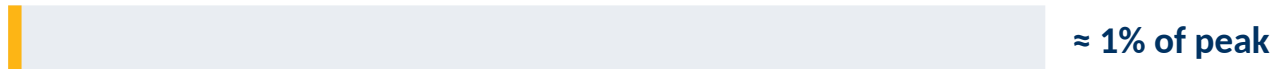
Llama 3 405B, well tuned (8K–16K H100s)



Many training jobs



Decode serving at batch 8



Llama 3 405B: 380–430 of 989 TFLOPS per GPU. Serving utilization rises with batch size, prefix caching and disaggregation.

Goodput, not throughput

≥ 90% effective time

lost

16K-GPU Llama 3 run. Lost time: restarts, checkpoint writes, stragglers, failed health checks.

≈ \$60M

per MW of AI factory (NVIDIA)

Idle is the enemy: a GPU waiting on the host, network or a checkpoint still draws its share.

What operators do

Health check before every allocation

Interleaved A/B on the same nodes

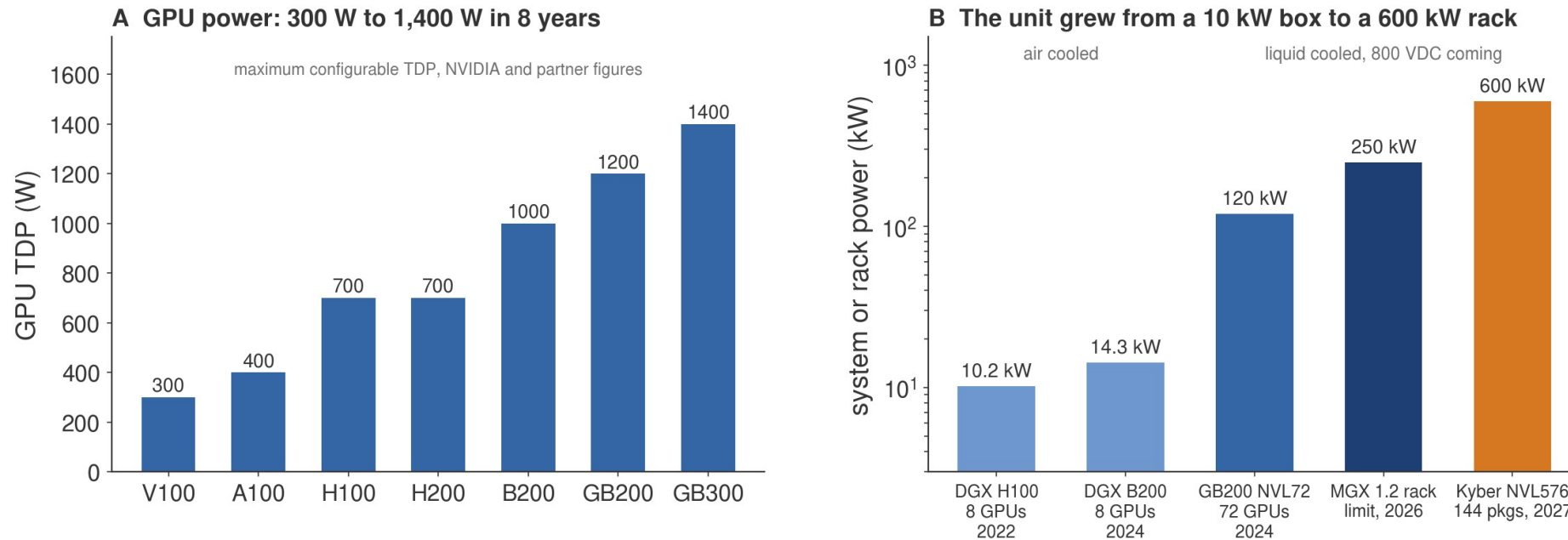
Budget for spare nodes

Auto-drain nodes that wedge

Source: Llama 3 paper (MFU, effective time): arxiv.org/abs/2407.21783; NVIDIA 'Productive, Durable, Fungible' (cost per MW): blogs.nvidia.com/blog/productive-durable-fungible-ai-factories

GPUs: from CUDA to the gigawatt rack | October 2026

Operator view 4: power and cooling set the shape of the data center



- GPU TDP went 300 W to 1.4 kW in eight years. The system went from a 10 kW air-cooled box to a 120 kW liquid-cooled rack, 250 kW in the MGX 1.2 spec, and 600 kW for Kyber in 2027.
- NVIDIA's factory-level figure for Rubin: 2.5 kW per GPU (40,000 GPUs per 100 MW), 45 C inlet water, 800 V DC distribution. Rack power, not floor space, is what you run out of.

Source: nvidia.com/en-us/data-center/dgx-b200 (14.3 kW); [GTC 2024](#) and [GTC 2025](#) keynote blogs (120 kW, 600 kW); [OCP 2025](#) and [2026](#) blogs (MGX 1.2, 800 VDC)

Operator view 5: the economics NVIDIA itself now argues with

Quantity	Figure	Source
Cost of an AI factory	about 60 million dollars per megawatt	NVIDIA blog, Oct 2026
The metric	tokens per second per megawatt, then cost per million tokens	NVIDIA, CoreWeave, SemiAnalysis AgentX
Useful life	5 to 6 years for an 8x H100 system, 9 to 10 years for GB300 NVL72 (resale-based)	Barkr, quoted by NVIDIA
Old GPUs keep earning	A100 (2020) still rented, bookings through 2029; worth 25% of cost after 6 years	CoreWeave, Silicon Data via NVIDIA
Agentic token demand	about 15x the tokens of a chat request per task	OpenRouter data via NVIDIA
Generation step (inference)	Vera Rubin NVL72 vs GB300 NVL72: 2.5 to 3.7x (MLPerf v6.1), 30x per MW on agentic traces (AgentX)	MLCommons; SemiAnalysis
Power as the lever	DSX MaxLPS: 19 nodes in the power of 16 (+24% tokens) at Lambda; up to 40% more GPUs per MW claimed for Rubin	NVIDIA AI Infra Summit 2026

- Two consequences for an operator. First, an H200 fleet does not become worthless when B300 ships: it moves to the workloads whose roofline it still fits (BF16 training, moderate-context serving). Second, the per-MW metric means a slower GPU with more memory per watt can win the inference bid.
- Treat the multipliers as claims under specific workloads and precisions. The MLPerf numbers are peer-reviewed, the per-MW numbers are vendor or analyst measurements.

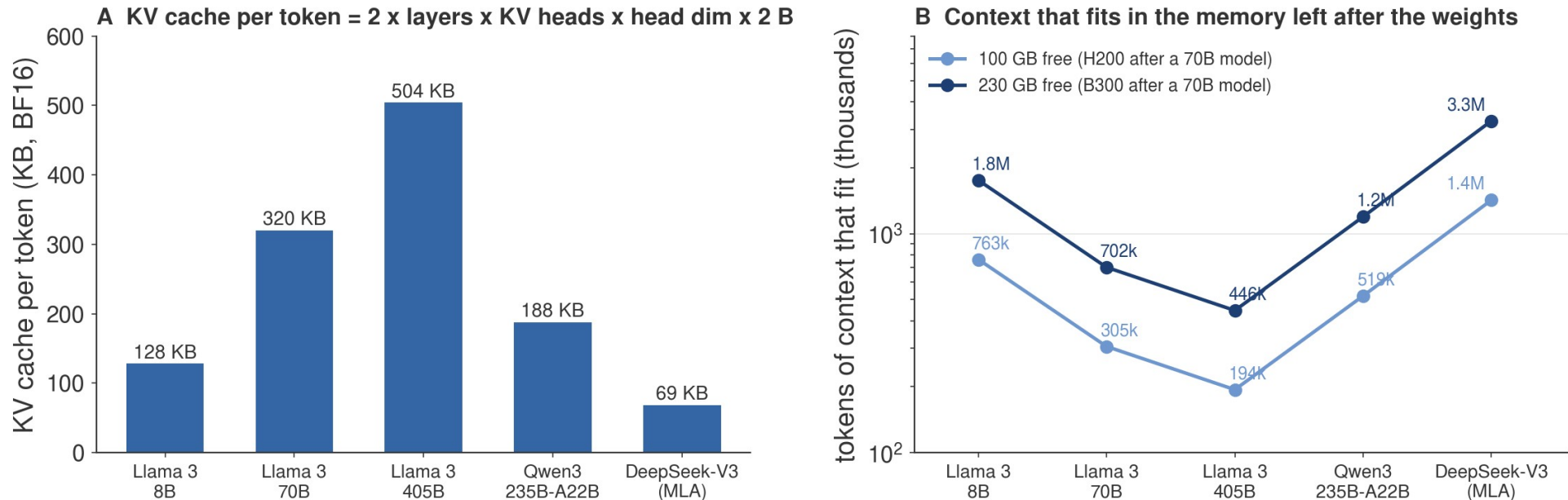
Source: blogs.nvidia.com/blog/productive-durable-fungible-ai-factories/, [/vera-rubin-nvl72-mlperf-inference](#), [AI Infra Summit 2026 blog](#)

Four shifts in the workload since 2023

SHIFT		EVIDENCE	HARDWARE ANSWER	
Training	➡	Inference	The installed base now serves more than it trains. Blackwell Ultra traded FP64 for FP4; Rubin CPX is prefill-only; Groq LPX is decode-only.	FP4 and memory
Chat	➡	Agents	≈ 15x the tokens per task (OpenRouter data). Context grows to hundreds of thousands of tokens across tool calls.	Tokens per MW, Vera CPU
Dense	➡	Mixture of experts	DeepSeek-V3: 671B parameters, 37B active, 256 experts, top-8. Compute per token stays flat. memory and all-to-all grow.	NVL72 scale-up, GPU-initiated communication
Answer directly	➡	Reason first	Models think in tokens before answering. Decode speed per user and KV cache per request become the limits.	HBM capacity (288 GB, HBM4), KV offload

Source: nvidia.com/en-us/data-center/vera-rubin-nvl72; Inside Blackwell Ultra: developer.nvidia.com/blog; DeepSeek-V3 report arxiv.org/abs/2412.19437; Qwen3 qwenlm.github.io/blog/qwen3

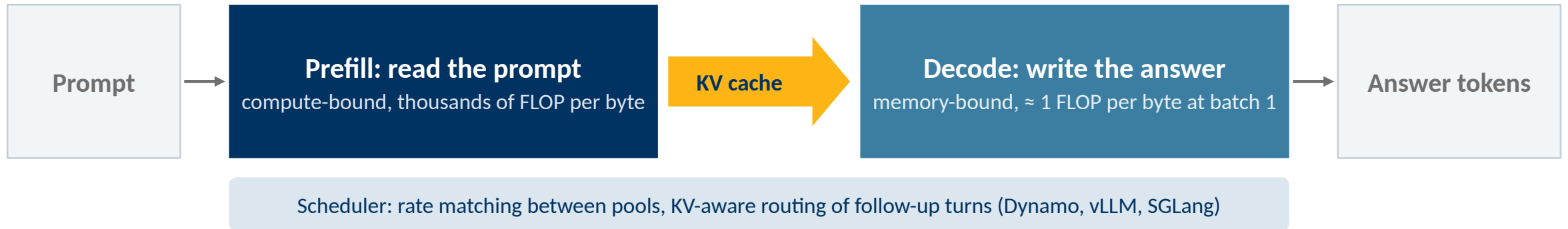
Decode is a memory problem: the KV cache



- Every generated token reads the whole KV cache of its request: $2 \times \text{layers} \times \text{KV heads} \times \text{head dim} \times \text{bytes per token}$. 320 KB for Llama 3 70B at BF16, 69 KB for DeepSeek-V3 with MLA.
- After a 70B model's weights an H200 has about 100 GB left: 300k tokens of Llama 3 70B context or 1.4 M of DeepSeek-V3, before batching. This arithmetic is why 288 GB and HBM4 sell.

Source: model configurations (config.json): huggingface.co/meta-llama/Llama-3.1-405B, deepseek-ai/DeepSeek-V3, Qwen/Qwen3-235B-A22B

Disaggregated serving: the GPU splits into roles



NVIDIA now builds hardware per role

Prefill	Low-latency decode	Everything in between	Agents' tool calls	KV-cache storage
Rubin CPX 30 PF NVFP4 128 GB GDDR7, no HBM 3x attention, end of 2026	Groq 3 LPX 256 LPUs per rack, SRAM 640 TB/s scale-up 3,400 tok/s (31B, 100K ctx)	Rubin GPU the general case	Vera CPU CPU racks for tools and sandboxes	BlueField-4 STX storage racks as a shared context tier

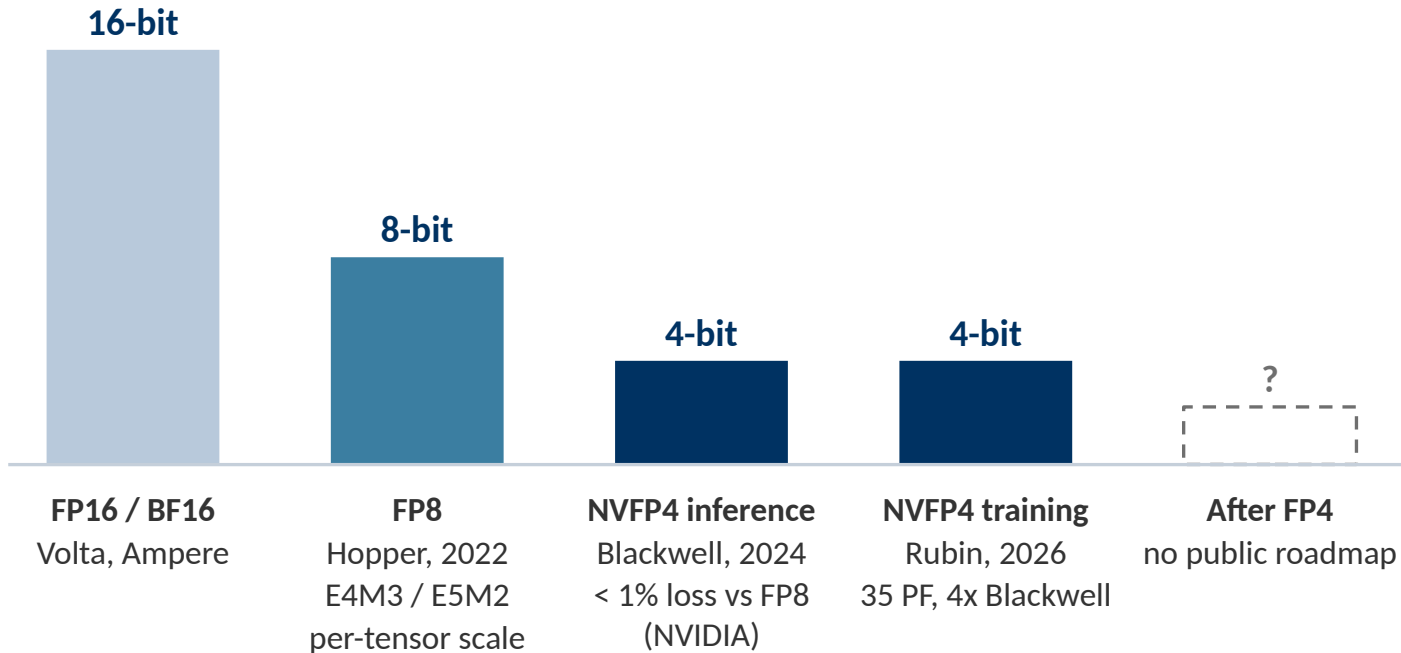
A factory is a mix of pools with different power, memory and failure profiles. 'How many GPUs' becomes 'how many of each'.

Source: nvidianews.nvidia.com/news/nvidia-vera-rubin-platform; [Rubin CPX press release](#); [Groq 3 LPX, Aug 2026](#) (nvidianews.nvidia.com)

GPUs: from CUDA to the gigawatt rack | October 2026

Precision: the ladder keeps going down

Bits per value, generation by generation



NVFP4: how 4 bits keep accuracy



FP8 scale

one E4M3 scale per block of 16

FP32 scale

one scale per tensor

Weights and KV cache: **1.8x** smaller than FP8, **3.5x** smaller than FP16

Open question for 2026–27: will frontier labs pretrain at FP4? NVIDIA claims MoE pretraining with a quarter of the GPUs.

Given up: FP64 went 67 TF (H200) → 1.4 TF (GB300) → 33 TF (Rubin). INT8 fell to 0.2 POPS on Blackwell Ultra.

After FP4, the next doublings come from sparsity, specialized units (attention, LPU) and memory.

Source: [Inside Blackwell Ultra](#) and [NVFP4](#) (developer.nvidia.com/blog); [nvidia.com/en-us/data-center/hgx](#) (Rubin training FLOPS); [amd.com/en/products/accelerators/instinct/mi400.html](#)

GPUs: from CUDA to the gigawatt rack | October 2026

The competition, 2026 (per accelerator, vendor peak figures)

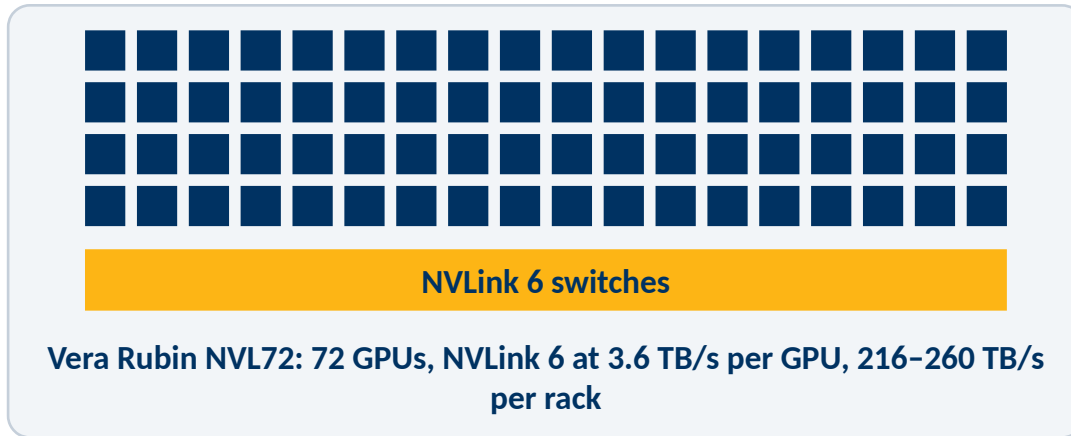
	Memory	FP8 dense	FP4 dense	Scale-up domain	Status
NVIDIA Rubin	288 GB HBM4, 19-22 TB/s	17.5 PF	35 PF (50 sparse)	NVLink 6, 72 per rack, 3.6 TB/s per GPU	ramping, racks at clouds since July 2026
NVIDIA GB300	288 GB HBM3e, 8 TB/s	5 PF	15 PF	NVLink 5, 72 per rack, 1.8 TB/s	shipping, p6e-gb300
AMD MI455X (Helios)	432 GB HBM4, 23.3 TB/s	20 PF (AMD peak)	40 PF (MXFP4)	UALink, 72 per rack, 3.6 TB/s per GPU, ORW double-wide rack	volume 2H 2026 (reference design)
AMD MI355X	288 GB HBM3e, 8 TB/s	about 5 PF	about 10 PF (MXFP4)	Infinity Fabric, 8 per node	shipping
Google TPU v7 Ironwood	192 GB HBM3e, 7.4 TB/s	4.6 PF	-	ICI with optical circuit switches, pods of 9,216	GA 2025, Google Cloud only
AWS Trainium3	144 GB HBM3e, 4.9 TB/s	2.5 PF (MXFP8)	W4A8 in hardware	NeuronSwitch, up to 144 per UltraServer	trn3, 2025-26
NVIDIA Groq 3 LPU (LP30)	SRAM, 128 GB per 256-LPU rack	-	-	direct chip-to-chip, 640 TB/s per rack	LPX racks in production, Aug 2026

- AMD's own comparison lists Rubin at 35 PF FP4 and 22 TB/s and claims 1.26x on peak FLOPS and 1.5x on memory. Peak is not throughput: software maturity, rack availability and power decide deployments.
- The hyperscaler chips compete on cost per token for their own workloads and on scale-up domain size, not on per-chip peak. Trainium3's hardware MoE routing and W4A8 are aimed at the same inference shift.

Source: amd.com/en/products/accelerators/instinct/mi350.html, [/mi400.html](https://amd.com/en/products/accelerators/instinct/mi400.html); [Ironwood, Apr 2025 \(blog.google\)](https://ironwood.blog.google); aws.amazon.com/ai/machine-learning/trainium; nvidia.com/en-us/data-center/vera-rubin-nvl72

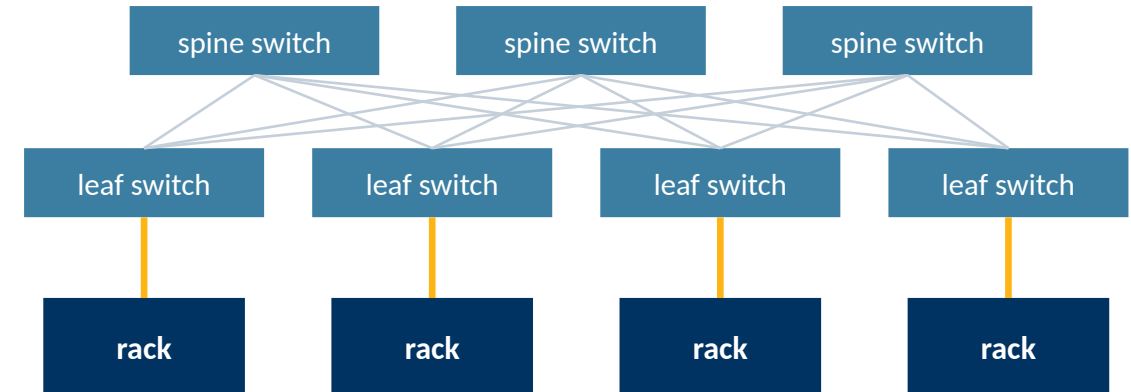
Networking: scale-up is NVIDIA's, scale-out is the operator's choice

Scale-up: one NVLink domain (NVIDIA)



- $\approx 10\times$ packet rate, 3x lower latency than off-the-shelf Ethernet (NVIDIA)
- NVLink Fusion: Intel x86, d-Matrix, Samsung Foundry parts can join

Scale-out: the operator's network



Each GPU has a 1.6 Tb/s port (ConnectX-9); one Spectrum-6 switch carries 102.4 Tb/s

- Multiplane: up to 8 planes, 512,000 GPUs on two tiers
- Spectrum-XGS: one job across sites; Meta adopted Spectrum-6 (OCP 2026)

Scale-out choices

Spectrum-X
Ethernet

InfiniBand

AWS EFA (SRD)

Ultra Ethernet

Google Falcon

Broadcom
Tomahawk

Optics: co-packaged (200G per lane, 5x power efficiency) vs pluggable USFP (400G per lane). **The work:** tail latency, congestion control, multipath.

Source: [Vera Rubin](#), [NVLink 6](#), [Spectrum-X](#) blogs (nvidia.com, 2026); [OCP 2026 blog](#); AWS EFA docs.aws.amazon.com/AWSEC2/latest/UserGuide/efa.html

GPUs: from CUDA to the gigawatt rack | October 2026

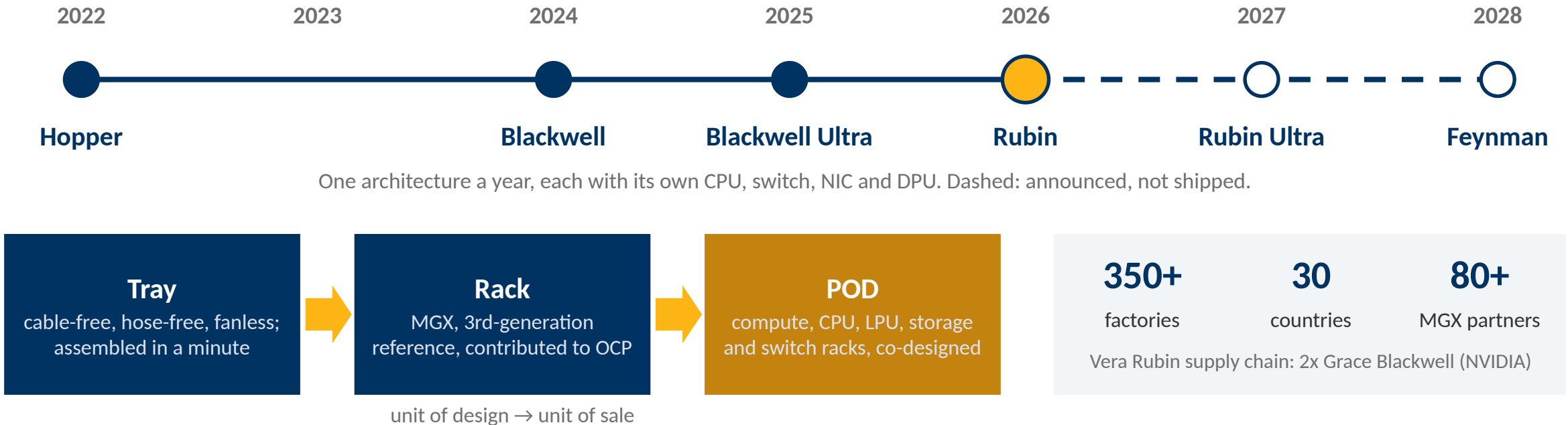
Vera Rubin NVL72 by the numbers (NVIDIA's official page)

	Per rack (72 GPUs, 36 CPUs)	Per GPU	vs GB300 NVL72
NVFP4 inference (sparse)	3,600 PF	50 PF	2.5x
NVFP4 training (dense)	2,520 PF	35 PF	2.3x
FP8 (dense)	1,260 PF	17.5 PF	3.5x
FP16 / BF16 (dense)	288 PF	4 PF	1.6x
FP64	2,400 TF	33 TF	24x (Blackwell Ultra had cut it)
HBM4	20.7 TB at 1,400 TB/s	288 GB at 19.2 TB/s (HGX page: 22)	same capacity, 2.4x bandwidth
NVLink 6	216 TB/s (blog: 260 TB/s all-to-all)	3 TB/s (HGX page: 3.6)	1.7 to 2x
CPU	3,168 Olympus cores, up to 54 TB LPDDR5X	88 cores, 1.5 TB per CPU	about 3x CPU memory
Network	32.4 TB/s bidirectional	1.6 Tb/s (ConnectX-9)	2x
Chips per rack	1,296 NVIDIA and HBM4 dies	-	-
Power	'40K GPUs in 100 MW' = 2.5 kW per GPU at factory level, 45 C inlet	-	-

- Measured so far: MLPerf v6.1 preview 2.5x (DeepSeek-R1) to 3.7x (Qwen3-VL) over GB300 NVL72, CoreWeave 10x tokens per MW over GB200 NVL72, AgentX up to 30x per MW. The spread is workload, precision and denominator.

Source: nvidia.com/en-us/data-center/vera-rubin-nvl72, [/hgx](#); blogs.nvidia.com/blog/vera-rubin-nvl72-mlperf-inference; mlcommons.org/benchmarks/inference-datacenter

Trend 1: one architecture per year, and the rack is the product



For operators

A fleet is always four generations deep: H200, B200, B300 and Rubin side by side. Useful life (5–10 years) outlasts the cadence, so fungibility decides utilization.

For developers

Write to the library and framework layer and let the hardware move under you. Hand-tuned Hopper kernels (wgmma, TMA) already differ on Blackwell (tcgen05, TMEM).

Source: blogs.nvidia.com/blog/nvidia-gtc-taipei-computex-2026-news/; [OCP 2026 blog](#); ['Productive, Durable, Fungible' blog](#)

GPUs: from CUDA to the gigawatt rack | October 2026

Your turn: discuss with your neighbor



Your site has a fixed power budget. Which GPU do you buy?

GPU X

30% slower, but 50% more efficient per watt

GPU Y

the faster one

2 minutes: pick one and say why.

One view: GPU X. When power is the budget you sell tokens per MW, and X gets 1.5x the tokens out of the same megawatts.

Click to show one view. Sources on the next page.

GPUs: from CUDA to the gigawatt rack | October 2026

Trend 2: power is the budget, tokens per megawatt is the metric

Power per GPU

H200, chip TDP



GB300, chip TDP



Rubin, whole factory per GPU



Power per rack

GB200 NVL72



MGX 1.2 limit



Kyber (2027)



At 2.5 kW per GPU

100 MW ≈ 40,000 GPUs

1 GW ≈ 400,000 GPUs

The response is infrastructure, not silicon

800 V DC: 150% more power per copper cross-section

45 °C warm-water cooling, no chillers

5,000 A liquid-cooled busbars

Energy storage in the rack smooths tens-of-MW swings

Power management became software

DSX MaxLPS: Lambda ran 19 nodes in the power of 16, +24% tokens

DSX Flex: the factory sheds load on a grid signal

OCP grid spec: back to near-full load within 2 seconds

How many GPUs can I buy?



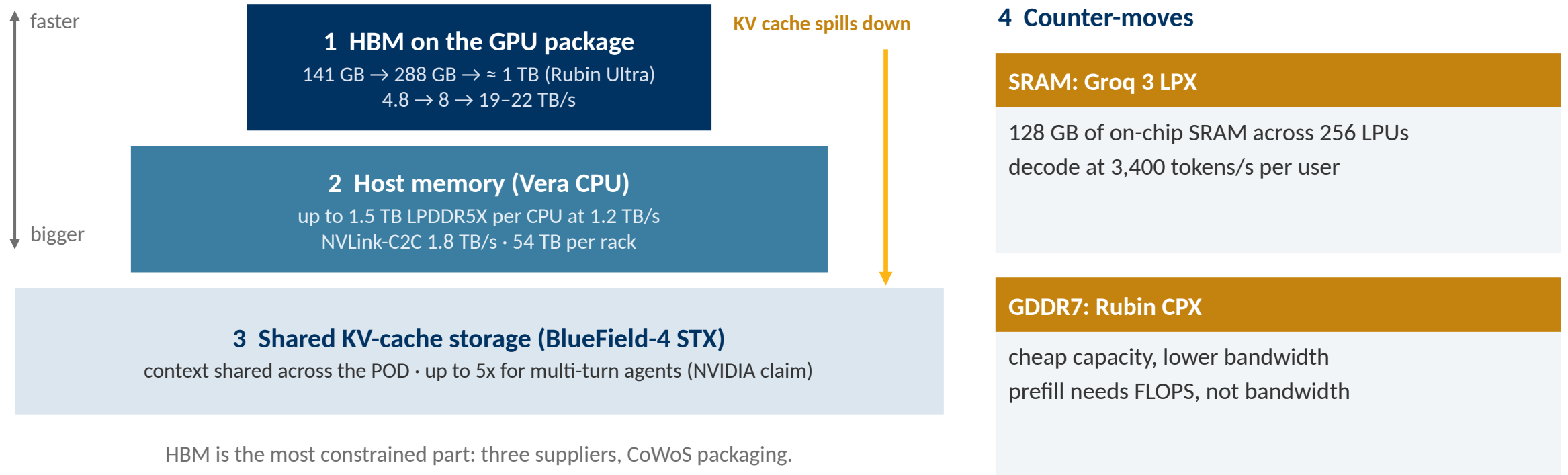
How many MW can I energize, and how many tokens per MW?

A GPU 30% slower but 50% more efficient per watt wins a power-limited site.

Source: nvidia.com/en-us/data-center/vera-rubin-nvl72; [OCP 2025](#), [OCP 2026](#), [800 VDC](#) and [AI Infra Summit 2026 \(Lambda, DSX\)](#) blogs on [blogs.nvidia.com](#)

GPUs: from CUDA to the gigawatt rack | October 2026

Trend 3: memory becomes a tiered system, and context becomes a storage class



'How much HBM' is still the first question, but memory is now a four-level hierarchy with a routing decision at each level. Million-token agent contexts force it.

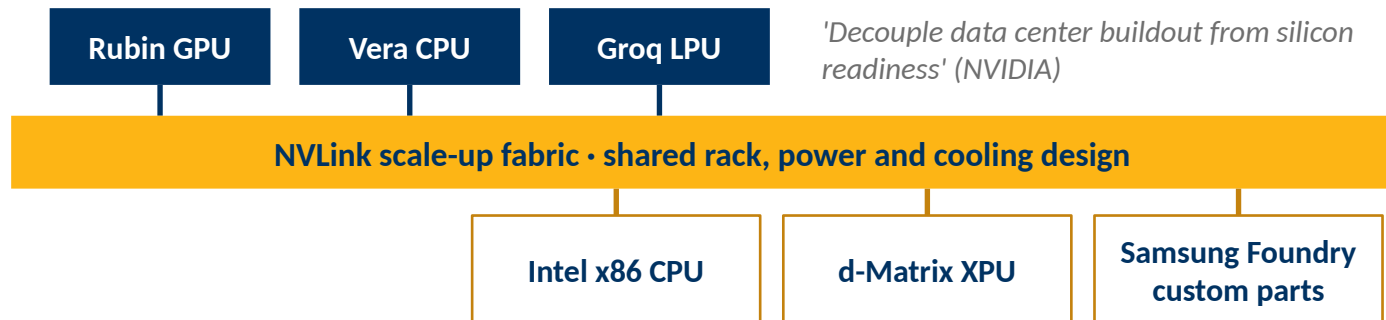
Trend 4: specialization inside the factory, and NVLink opens to others

The GPU is no longer one part: one rack type per role

Rubin GPU	Rubin CPX	Groq 3 LPU	Vera CPU	BlueField-4
general	prefill · GDDR7 video codecs on die	decode · SRAM deterministic	agents, tool calls 2x single-thread (claim)	storage and security

Feynman (2028) continues the pattern with the Rosa CPU and the LP40 LPU.

NVLink Fusion: other silicon joins the scale-up fabric



Hyperscalers run the same play

Trainium3: MoE routing in hardware, W4A8
 TPU v7: optical circuit-switched pods
 AWS + NVIDIA (Mar 2026): > 1M NVIDIA GPUs plus LPUs for AWS

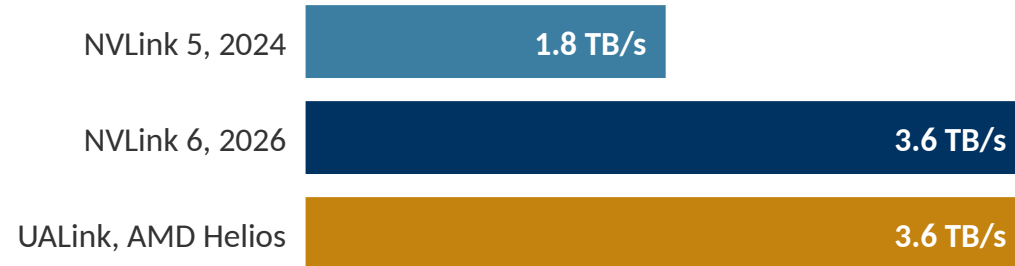
'Which GPU' becomes 'which mix for which traffic'. Schedulers, not chips, set the token cost.

Source: nvidianews.nvidia.com/news/nvidia-vera-rubin-platform; [GTC 2026 blog](#); [Groq 3 LPX, Hot Chips 2026](#); [LPX / NVLink Fusion blog](#); aws.amazon.com/ai/machine-learning/trainium

GPUs: from CUDA to the gigawatt rack | October 2026

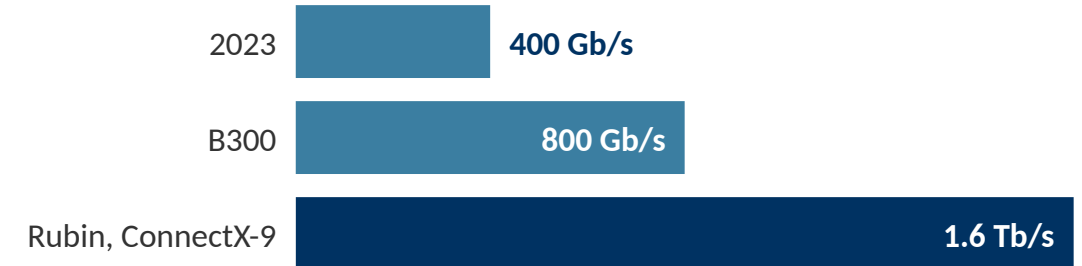
Trend 5: networks scale across, go optical, and move into the GPU

Scale-up per GPU (both directions)



72 GPUs per domain today; Kyber (2027) adds copper plus co-packaged optics.

Scale-out per GPU (one direction)

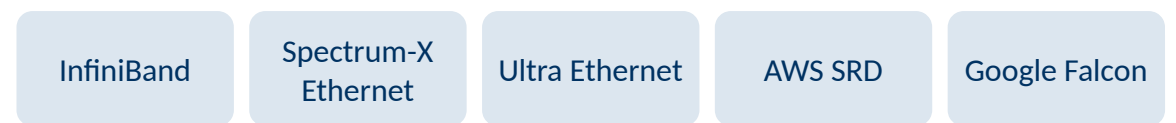


Each step doubles optics count and optics power per GPU. Multiplane: 512,000 GPUs on two tiers.

Optics



Transports differ; RDMA verbs stay the common API



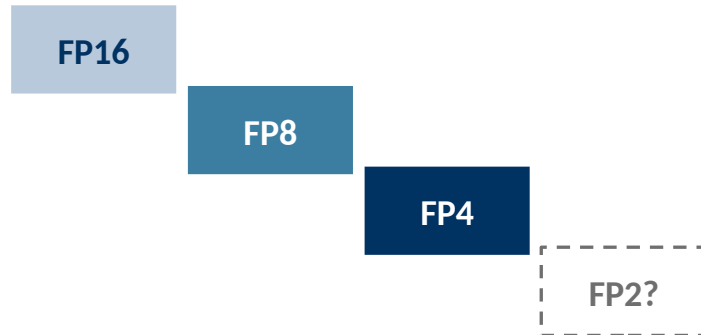
Congestion control and multipath are where they differ.



Source: [Spectrum-X photonics](#) and [NVLink 6](#) blogs (developer.nvidia.com); [OCP 2026 blog](#); [amd.com/en/products/accelerators/instinct/mi400.html](#); NCCL [docs.nvidia.com/deeplearning/nccl](#)

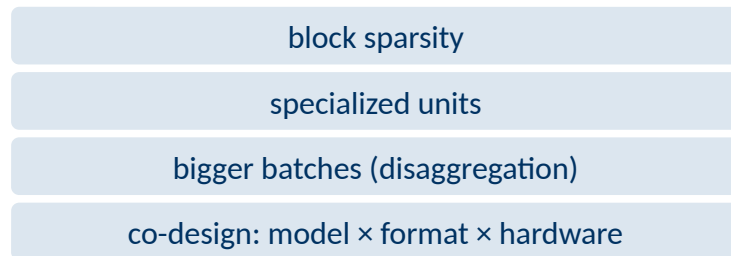
Trend 6: numerics and architecture after the easy precision gains

The easy rungs are used up

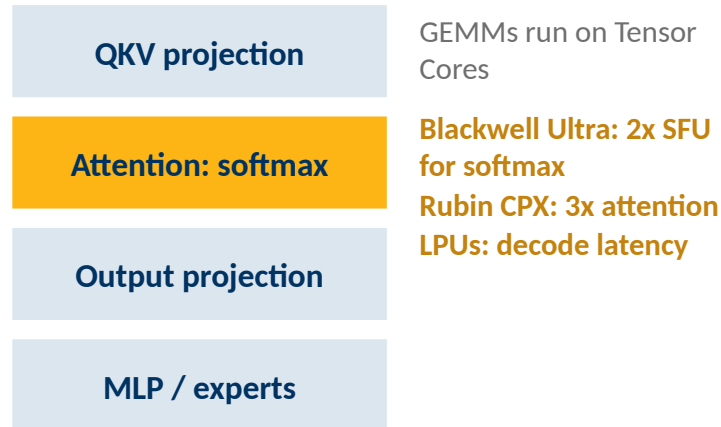


2x per rung so far; no public FP2 roofline

The next 2x comes from



Attention is the new hot spot

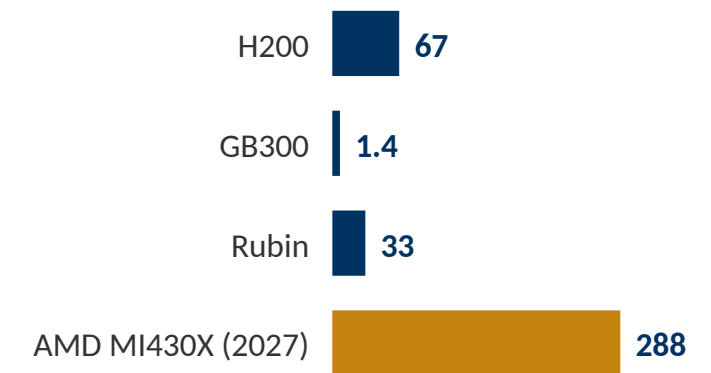


Expect more fixed-function help for the parts of the transformer that are not a GEMM.

GEMMs run on Tensor Cores

Blackwell Ultra: 2x SFU for softmax
Rubin CPX: 3x attention LPUs: decode latency

HPC and AI diverge: FP64 TFLOPS



MI430X targets sovereign and scientific sites. Operators serving both markets will run two fleets.

Software has to keep up: NVFP4 needs calibration and block scaling, and MLPerf v6.1 gains came from lower-precision KV cache and kernel fusion. Test FP4 accuracy on your own models before pricing it in.

Trend 7: open standards for the plumbing, proprietary for the fabric

What is open, layer by layer

Software	Open: PyTorch, Triton and vLLM run everywhere	Per vendor: the fastest kernels and GPU-initiated communication libraries
Fabric and formats	Proprietary: scale-up fabric (NVLink 6, UALink, ICI, NeuronLink), number formats (NVFP4, MXFP4), serving integration. NVLink Fusion: you can join NVIDIA's fabric; the fabric is not open.	
The building	Open: MGX rack and tray designs in OCP • Open Data Center Spec v0.7 (NVIDIA, AMD, Google, Meta) • grid resilience spec • DSX OS components open-sourced (Oct 2026) • AMD Helios on Open Rack Wide	

open
 proprietary

Operator reading

- 1

Standardize the building (power, cooling, racks) so compute can change.
- 2

Keep workloads portable at the framework layer.
- 3

Accept vendor-specific code only where the profile shows it pays.

Source: blogs.nvidia.com/blog/ocp-global-summit-vera-rubin-scale-ai-factories; amd.com/en/products/accelerators/instinct/mi400.html (ORW, UALink, UEC); opencompute.org

GPUs: from CUDA to the gigawatt rack | October 2026

Your turn: vote



Which part of this picture is the least certain?

A

FP4 training

B

Decode moving to
SRAM chips

C

Custom chips on
NVLink Fusion

D

Power and the grid

E

Demand itself

No single right answer. This deck's view: the roadmap is the most certain part; power and demand are the least.

Click to show the deck's view. Sources on the next page.

GPUs: from CUDA to the gigawatt rack | October 2026

Risks and uncertainties (what could change this picture)

Area	What we know	What we do not know
Supply	HBM from three vendors, CoWoS packaging at TSMC, NVIDIA says Rubin supply chain is 2x Blackwell's	whether HBM4 and CoWoS keep up with an annual cadence at 1M+ GPU per year volumes
Power and sites	gigawatt sites announced, 800 VDC and liquid cooling becoming standard, grid programs starting	grid connection lead times, water, local permitting. The 100 GW 'stranded power' claim is a vendor figure
Demand	agentic traffic 15x chat per task, token prices falling 10x per generation (vendor claims)	whether token demand grows faster than token cost falls. NVIDIA argues yes, the ROI debate says maybe
Useful life	5-10 years by resale data, A100 still earning	accounting vs physical life if a generation step is 10x per MW
Numerics	FP4 inference mainstream, FP4 training claimed on Rubin	accuracy at FP4 for long-context agents, adoption by frontier labs
Competition	AMD Helios 2H 2026 with more HBM, TPUs and Trainium at scale inside their clouds	software maturity on ROCm, actual rack-level throughput and reliability of new fabrics
Geopolitics	export controls shape which parts ship where (H20-class parts, sovereign AI programs)	rule changes, which can move demand by regions within a quarter

Strategy for an operator (one view)

1

Buy memory and interconnect, not peak FLOPS

Decode and MoE are bound by HBM and the scale-up domain. FP4 counts only if your models hold accuracy at FP4.

from: roofline, KV cache

2

Plan for power first

2.5 kW per GPU, 120–600 kW per rack, liquid cooling, 800 VDC. Sites take longer than a GPU generation.

from: power and cooling

3

Own the scale-out network

Transport, congestion control, topology, NIC placement. Line rate is the floor. tail latency is the work.

from: operator views 1–2, networking

4

Run the fleet as pools

H200: BF16 training • B300: FP4, long context • NVL72: MoE, disaggregated • Rubin as it arrives. Price per MW-hour too.

from: economics

5

Measure interleaved, on the same nodes

Node state can move results as much as an optimization does. Health preflights, A/B pairing, failure taxonomies.

from: reliability, tools

6

GPU-initiated communication is production code

It carries the throughput and the rarest bugs. Test under load and congestion. keep stress tests in CI.

from: MoE, networking

Source: opinion, grounded in the public figures of Parts 2 and 3

GPUs: from CUDA to the gigawatt rack | October 2026

What 2027 and 2028 probably look like

2027

announced roadmaps

Rubin NVL72 is the volume cloud platform

Rubin CPX racks for million-token prefill

Kyber + Rubin Ultra: $\approx$ 1 TB HBM4e per GPU, 800 VDC, 600 kW, late 2027

AMD Helios: 432 GB per GPU, in volume

1.6 Tb/s per GPU is the standard port

2028

direction only (GTC 2026 keynote)

Feynman with the Rosa CPU and LP40 LPU

ConnectX-10 and BlueField-5

Kyber with co-packaged optics for scale-up

Over 3 kW per GPU at factory level

■ announced ■ likely □ uncertain

Likely, whatever the vendor

Scale-up domains of 144–576 dies

1 TB-class memory per package

Disaggregated serving by default

FP4 as the inference default

Agents as the main token consumer

Power as the unit of capacity planning

Uncertain

FP4 training

decode on SRAM chips

custom XPU's on NVLink Fusion

grid build-out pace

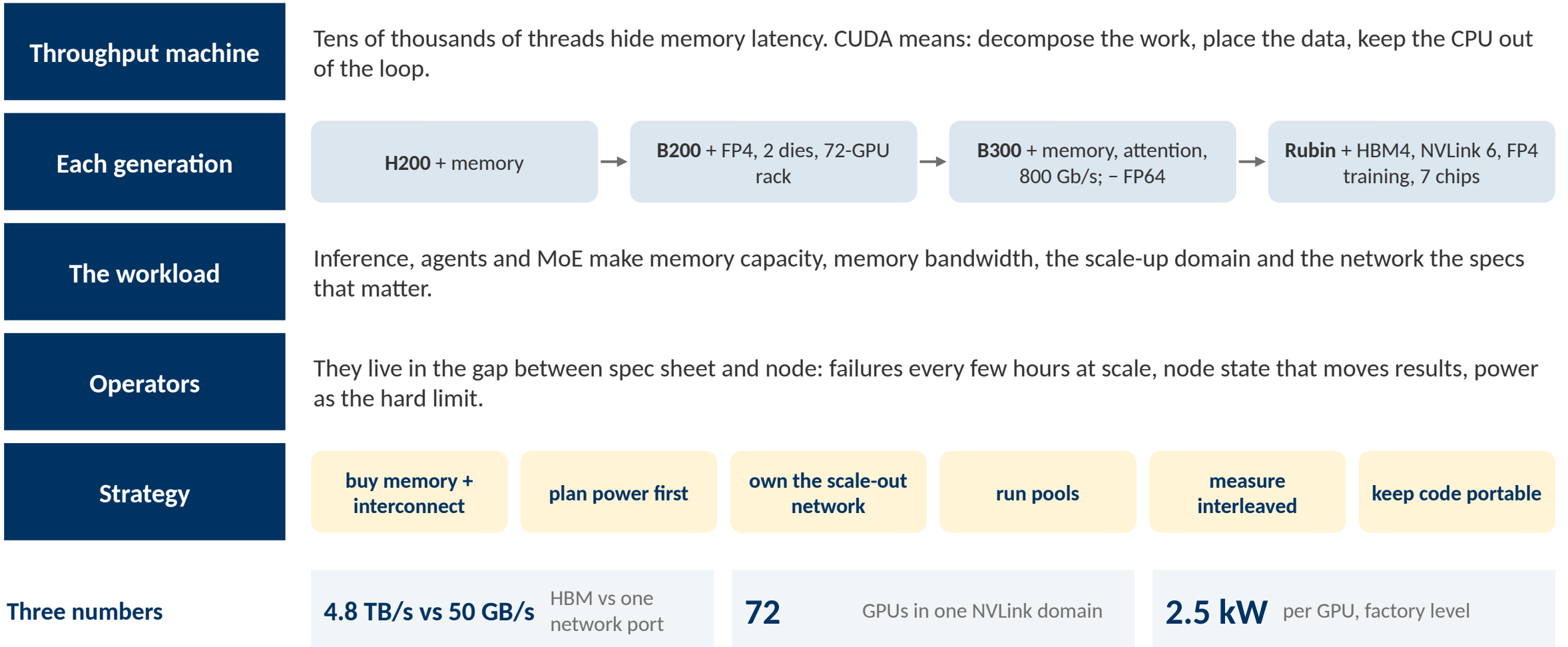
demand itself

Today's H200 and B200 fleets: 5+ years of useful life. software keeps improving them (1.6x in one MLPerf round on GB300). Keep them busy.

Source: [GTC 2026](#), [OCP 2025](#), [OCP 2026](#) blogs; [Rubin CPX release](#); amd.com/en/products/accelerators/instinct/mi400.html; the rest is the speaker's reading

GPUs: from CUDA to the gigawatt rack | October 2026

Takeaways



A1. Per-GPU specification table (dense tensor figures)

GPU	Year	HBM	BF16	FP8	FP4	FP64	NVLink	TDP	AWS instance, net per GPU
V100 SXM2	2017	32 GB HBM2, 0.9 TB/s	125 TF (FP16)	-	-	7.8 TF	300 GB/s	300 W	p3dn, 12.5 Gb/s
A100 SXM 80GB	2020	80 GB HBM2e, 2.0 TB/s	312 TF	-	-	19.5 TF	600 GB/s	400 W	p4d, 50 Gb/s
H100 SXM5	2022	80 GB HBM3, 3.35 TB/s	989 TF	1.98 PF	-	34 / 67 TF	900 GB/s	700 W	p5, 400 Gb/s
H200 SXM	2024	141 GB HBM3e, 4.8 TB/s	989 TF	1.98 PF	-	34 / 67 TF	900 GB/s	700 W	p5en, 400 Gb/s
B200 (HGX)	2024-25	180 GB HBM3e, 8 TB/s	2.25 PF	4.5 PF	9 PF	37 TF	1.8 TB/s	1,000 W	p6-b200, 400 Gb/s
GB200 (NVL72)	2024-25	186 GB HBM3e, 8 TB/s	2.5 PF	5 PF	10 PF	40 TF	1.8 TB/s	1,200 W	p6e-gb200, 400 Gb/s
B300 (HGX)	2025	288 GB HBM3e, 8 TB/s	2.25 PF	4.5 PF	13.5 PF	1.25 TF	1.8 TB/s	1.1-1.4 kW	p6-b300, 800 Gb/s
GB300 (NVL72)	2025	288 GB HBM3e, 8 TB/s	2.5 PF	5 PF	15 PF	1.4 TF	1.8 TB/s	1.4 kW	p6e-gb300, 800 Gb/s
Rubin (NVL72)	2026	288 GB HBM4, 19.2-22 TB/s	4 PF	17.5 PF	35 PF (50 sparse)	33 TF	3-3.6 TB/s	n/a	CX-9, 1.6 Tb/s
Rubin CPX	end 2026	128 GB GDDR7	-	-	30 PF	-	-	n/a	prefill only
Rubin Ultra	2027	about 1 TB HBM4e	-	-	about 100 PF	-	NVLink 7	n/a	Kyber NVL576 (targets)

Source: NVIDIA product pages ([h200](#), [hgx](#), [gb200-nvl72](#), [gb300-nvl72](#), [vera-rubin-nvl72](#) under [nvidia.com/en-us/data-center](#)). 'Sparse' spec-sheet values were halved

A2. Glossary

Term	Meaning
SM, warp, block, grid	streaming multiprocessor (the GPU's core), 32 threads in lockstep, a group of up to 1,024 threads on one SM, the whole launch
HBM, HBM3e, HBM4	high-bandwidth memory stacked next to the GPU die. HBM3e: 8 TB/s class. HBM4: 2,048-bit interface, 19-22 TB/s on Rubin
NVLink, NVSwitch, NVL72	NVIDIA's scale-up link, the switch that joins GPUs, a 72-GPU rack-scale domain
Scale-up vs scale-out	load/store fabric inside a rack (NVLink, UALink, ICI) vs RDMA network between racks (InfiniBand, Ethernet, EFA)
EFA, SRD	AWS's network interface for HPC/ML and its multipath reliable datagram transport
Tensor Core, TMA, TMEM	matrix unit per SM partition, Hopper's tile copy engine, Blackwell's 256 KB per-SM tensor memory
FP8, NVFP4, MXFP4	8-bit float (E4M3/E5M2), NVIDIA's 4-bit float with FP8 block scales, the OCP micro-scaling 4-bit format
MoE, EP, dispatch, combine	mixture of experts, expert parallelism, the all-to-all that routes tokens to experts and back
KV cache	per-token keys and values kept for attention during decode. Memory grows with context x batch
MFU, goodput	achieved model FLOPs over peak. Useful work after restarts, checkpoints and stragglers
Disaggregated serving	prefill and decode on separate GPU pools, KV cache streamed between them
CPO, 800 VDC, MGX, Kyber	co-packaged optics, 800 V direct-current power distribution, NVIDIA's modular rack reference, the 2027 rack generation
GIN, GPU-initiated communication	NCCL's GPU-initiated networking device API, and the general idea of posting RDMA from a kernel

A3. Sources (all public, fetched 2026-10-16)

NVIDIA H200: nvidia.com/en-us/data-center/h200
 NVIDIA H100: nvidia.com/en-us/data-center/h100
 NVIDIA HGX (B200, B300, Rubin NVL8): nvidia.com/en-us/data-center/hgx
 NVIDIA DGX B200: nvidia.com/en-us/data-center/dgx-b200
 GB200 NVL72: nvidia.com/en-us/data-center/gb200-nvl72
 GB300 NVL72: nvidia.com/en-us/data-center/gb300-nvl72
 Vera Rubin NVL72: nvidia.com/en-us/data-center/vera-rubin-nvl72
 Rubin platform: nvidia.com/en-us/data-center/technologies/rubin
 Blackwell architecture: nvidia.com/en-us/data-center/technologies/blackwell-architecture
 Hopper architecture, H100 whitepaper: nvidia.com/en-us/data-center/technologies/hopper-architecture
 NVLink: nvidia.com/en-us/data-center/nvlink
 Groq 3 LPX: nvidia.com/en-us/data-center/lpx
 CUDA C++ Programming Guide: docs.nvidia.com/cuda/cuda-c-programming-guide
 Nsight Systems: docs.nvidia.com/nsight-systems
 Nsight Compute: docs.nvidia.com/nsight-compute
 NCCL documentation: docs.nvidia.com/deeplearning/nccl
 NCCL release notes: docs.nvidia.com/deeplearning/nccl/release-notes
 nccl-tests: github.com/NVIDIA/nccl-tests
 DeepEP: github.com/deepseek-ai/DeepEP
 vLLM: github.com/vllm-project/vllm
 SGLang: github.com/sgl-project/sglang
 Triton: github.com/triton-lang/triton
 Inside NVIDIA Blackwell Ultra (developer blog): developer.nvidia.com/blog/inside-nvidia-blackwell-ultra-the-chip-powering-the-ai-factory-era
 Introducing NVFP4 (developer blog): developer.nvidia.com/blog/introducing-nvfp4-for-efficient-and-accurate-low-precision-inference
 NVLink 6 resiliency (developer blog): developer.nvidia.com/blog/how-nvidia-nvlink-6-delivers-multi-layer-resiliency-for-ai-factories
 Spectrum-X Ethernet Photonics (developer blog): developer.nvidia.com/blog/scaling-power-efficient-ai-factories-with-nvidia-spectrum-x-ethernet-photonics
 Vera Rubin platform, GTC 2026 (press release): nvidianews.nvidia.com/news/nvidia-vera-rubin-platform
 Rubin, six new chips, CES 2026 (press release): nvidianews.nvidia.com/news/rubin-platform-ai-supercomputer
 CUDA Programming Guide: warps and SIMT: docs.nvidia.com/cuda/cuda-programming-guide/01-introduction/programming-model.html
 CUDA Programming Guide: tile kernels: docs.nvidia.com/cuda/cuda-programming-guide/02-basics/writing-tile-kernels.html
 PTX ISA (wgmma, tcgen05): docs.nvidia.com/cuda/parallel-thread-execution/index.html
 cuBLAS documentation (math modes): docs.nvidia.com/cuda/cublas/index.html
 NVIDIA blog: Programming Tensor Cores in CUDA 9: developer.nvidia.com/blog/programming-tensor-cores-cuda-9

Vera Rubin in full production, GTC Taipei 2026: nvidianews.nvidia.com/news/vera-rubin-full-production-agentic-ai-factory
 Rubin CPX, Sept 2025 (press release): nvidianews.nvidia.com/news/nvidia-unveils-rubin-cpx-a-new-class-of-gpu-designed-for-massive-context-inference
 Groq 3 LPX, Hot Chips 2026 (press release): nvidianews.nvidia.com/news/nvidia-groq-3-lpx-now-in-full-production-with-world-class-speed-for-agentic-ai
 GTC 2026 live blog: blogs.nvidia.com/blog/gtc-2026-news
 GTC Taipei 2026 live blog: blogs.nvidia.com/blog/nvidia-gtc-taipei-computex-2026-news
 GTC 2025 keynote blog: blogs.nvidia.com/blog/nvidia-keynote-at-gtc-2025-ai-news-live-updates
 GTC 2024 keynote blog: blogs.nvidia.com/blog/2024-gtc-keynote
 MLPerf Inference v6.1 (blog): blogs.nvidia.com/blog/vera-rubin-nvl72-mlperf-inference
 AgentX efficiency (blog): blogs.nvidia.com/blog/vera-rubin-nvl72-efficiency-ai-agents
 AI Infra Summit 2026 (blog): blogs.nvidia.com/blog/ai-infra-summit-vera-rubin-dsx-energy-efficiencies-tokens-per-watt-ai-factories
 LPX, Spectrum-X, NVLink Fusion (blog): blogs.nvidia.com/blog/vera-rubin-lpx-spectrum-x-nvlink-fusion
 OCP Global Summit 2025 (blog): blogs.nvidia.com/blog/gigawatt-ai-factories-ocp-vera-rubin
 OCP Global Summit 2026 (blog): blogs.nvidia.com/blog/ocp-global-summit-vera-rubin-scale-ai-factories
 800 VDC power architecture (blog): blogs.nvidia.com/blog/800-vdc-power-architecture-ai-factory
 'Productive, Durable, Fungible', Oct 2026 (blog): blogs.nvidia.com/blog/productive-durable-fungible-ai-factories
 AWS EC2 P4, P5, P6 instance pages: aws.amazon.com/ec2/instance-types/p5
 AWS Trainium: aws.amazon.com/ai/machine-learning/trainium
 AWS EFA documentation: docs.aws.amazon.com/AWSEC2/latest/UserGuide/efa.html
 AMD Instinct MI350: amd.com/en/products/accelerators/instinct/mi350.html
 AMD Instinct MI400: amd.com/en/products/accelerators/instinct/mi400.html
 Google Cloud Ironwood TPU, Apr 2025: blog.google/products/google-cloud/ironwood-tpu-age-of-inference
 Meta, The Llama 3 Herd of Models: arxiv.org/abs/2407.21783
 DeepSeek-V3 technical report: arxiv.org/abs/2412.19437
 Qwen3 release: qwenlm.github.io/blog/qwen3
 Model configs: Llama 3.1 405B, DeepSeek-V3, Qwen3-235B-A22B: huggingface.co/meta-llama/Llama-3.1-405B
 Intel ARK, Xeon Platinum 8570: intel.com/content/www/us/en/products/sku/237264
 MLCommons MLPerf Inference results: mlcommons.org/benchmarks/inference-datacenter
 Open Compute Project: opencompute.org
 von Neumann, First Draft of a Report on the EDVAC (1945): web.mit.edu/STS.035/www/PDFs/edvac.pdf
 Backus, Can programming be liberated from the von Neumann style? (CACM 1978): doi.org/10.1145/359576.359579
 Flynn, Very high-speed computing systems (Proc. IEEE 1966): doi.org/10.1109/PROC.1966.5273
 CUDA Programming Guide: warp matrix functions (WMMMA): docs.nvidia.com/cuda/cuda-programming-guide/05-appendices/cpp-language-extensions.html
 PyTorch CUDA semantics: TF32: docs.pytorch.org/docs/stable/notes/cuda.html