Read distraction-free on Substack

在 Substack 上享受无干扰阅读

2026: The Year of Dataflow

2026：数据流之年

FlashAttention. Megakernels. Boardfly TPU. Taalas. Extreme co-design. The biggest hardware progressions of the last few years are all converging on dataflow.

FlashAttention、巨型内核（Megakernels）、Boardfly TPU、Taalas、极致协同设计。过去几年硬件领域最重大的进展都在向数据流（Dataflow）汇聚。

DAVID DONG

AUG 19, 2026

Listen

I was watching Eric Vishria explain why Benchmark invested in Cerebras after a decade of avoiding hardware. His argument was simple: as you add more compute, you also need more memory nearby and more bandwidth between the compute units. Keep scaling all three, and eventually the chip boundary gets in the way. Cerebras's answer was to make the chip as large as a wafer.

70 6

chmark 在回避硬件领域十年后投资了能力的增加，你也需要在附近增加更多的内存，并在计算单元之间提供更大的带宽。如果这三者持续扩展，芯片边界最终会成为阻碍。Cerebras 的解决方案是将芯片做得像晶圆一样大。

That was 2016, and at the time it looked like an extreme position. In May 2026 Cerebras went public above $56 billion, with a multi-year agreement to deploy 750 megawatts of wafer-scale systems for OpenAI.

那是 2016 年，在当时看来这是一个极端的立场。2026 年 5 月，Cerebras 以超过 560 亿美元的估值上市，并签署了一项多年协议，为 OpenAI 部署 750 兆瓦的晶圆级系统。

The bet aged well because it was never really about wafers. It was about a boundary. Compute kept getting faster, the things feeding it did not, and the cost of crossing between them became the design problem. Cerebras took that observation to its physical limit early. Everyone else has been arriving at the same place ever since, from other directions.

这次豪赌之所以成功，是因为它从来不只是关乎晶圆，而是关乎边界。计算速度在不断提升，但为其提供数据的环节却停滞不前，跨越两者之间边界的成本成了设计的核心难题。Cerebras 很早就将这一观察推向了物理极限。从那时起，其他所有人也都在从不同方向汇聚到同一个终点。

Around 2017, the reason to build accelerators was deep learning. In 2026, it is inference. Both push computer systems toward dataflow.

2017 年左右，构建加速器的动力是深度学习。到了 2026 年，动力则变成了推理。两者都在推动计算机系统向数据流（dataflow）架构演进。

Deep learning gave hardware an unusually regular workload: large tensor operations, known dependencies, repeated layers, and enough reuse to justify specialized hardware and expensive compilation. Inference pushes this further. The same model may execute billions of times, while latency increasingly depends on where weights and KV cache live, how activations move between stages, and whether communication can overlap with computation.

深度学习为硬件提供了异常规律的工作负载：大型张量运算、明确的依赖关系、重复的层结构，以及足够的复用率，这使得专用硬件和昂贵的编译过程变得物有所值。推理则进一步推高了这一需求。同一个模型可能会执行数十亿次，而延迟越来越取决于权重和 KV 缓存的存放位置、激活值在各阶段间的移动方式，以及通信是否能与计算重叠。

Once those costs matter, optimizing each operation independently is no longer enough. The producer-consumer edges have to stay visible: which operation produces a value, where that value lives, how it moves, and when the consumer can run.

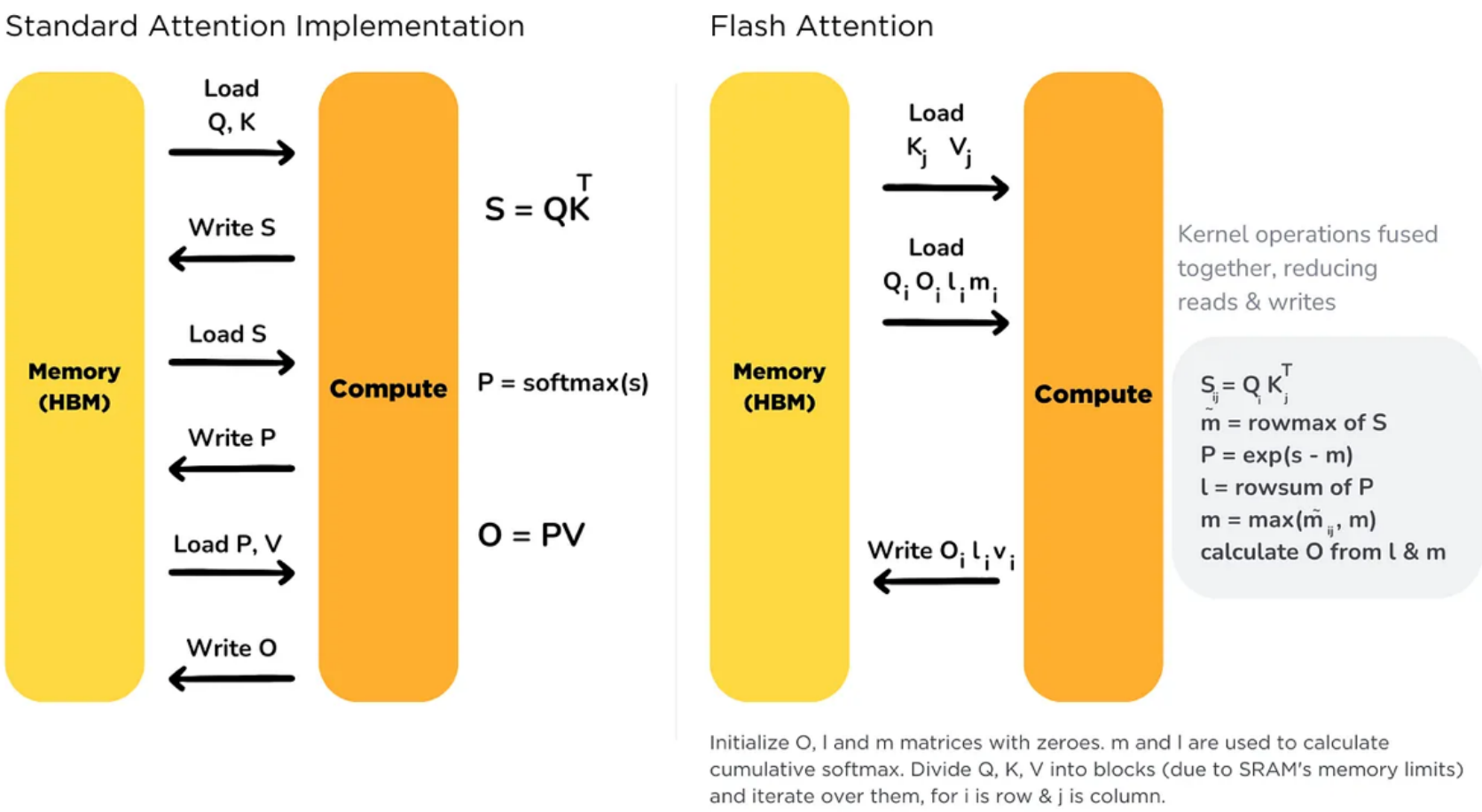
一旦这些成本变得至关重要，独立优化每个操作就再也行不通了。生产者-消费者（producer-consumer）之间的连接必须保持透明：哪个操作产生了值、该值存放在哪里、它是如何移动的，以及消费者何时可以运行。

That is the version of dataflow this series is about. The clearest way to see it is to watch one algorithm get rewritten around a single boundary.

这就是本系列文章所探讨的数据流版本。最直观的理解方式是观察同一个算法如何围绕单一边界进行重构。

Same math, same silicon, different execution

同样的数学原理，同样的芯片，不同的执行方式



A straightforward implementation writes the attention score matrix to HBM, reads it back for softmax, then writes and reads another large intermediate before multiplying by V. FlashAttention works on tiles instead. Q, K, and V blocks are brought into SRAM, a block of scores is computed, softmax state is updated, and the result is consumed while the data is still on chip. The full attention matrix never needs to exist in HBM. On the backward pass it recomputes values rather than storing and reloading them, because on a modern accelerator extra arithmetic can be cheaper than another trip through memory.

常规实现会将注意力评分矩阵写入 HBM（高带宽内存），再读取回来进行 softmax 运算，接着写入并读取另一个巨大的中间变量，最后与 V 相乘。而 FlashAttention 则采用分块（tiles）处理。Q、K 和 V 的数据块被调入 SRAM，计算出一块评分，更新 softmax 状态，并在数据仍在芯片上时就消耗掉结果。完整的注意力矩阵从未需要在 HBM 中存在。在反向传播过程中，它选择重新计算数值而非存储并重新加载，因为在现代加速器上，额外的算术运算往往比再次访问内存更廉价。

The important point is not that data stops moving. It is that the producer-consumer chain is reorganized around the memory hierarchy: values stay close to the computation long enough to be reused, and are consumed before they have to written back. Same attention, different physical execution.

https://davidhdong.substack.com/p/2026-the-year-of-dataflow?r=2l9xu3&utm_medium=ios

核心要点并非数据停止了流动，而是生产者-消费者链条围绕内存层级进行了重组：数值在计算单元附近停留足够长的时间以供复用，并在必须写回之前就被消耗掉。同样的注意力机制，不同的物理执行过程。

That is one boundary: HBM to SRAM. There are four others.

这是一个边界：从 HBM 到 SRAM。此外还有另外四个边界。

Thanks for reading! Subscribe for free to receive new posts and support my work.

Type your email... Subscribe

The seam ladder 接缝阶梯

A seam is a producer-consumer edge that crosses a physical boundary. The boundary may sit inside an operation, between memory tiers, between operators, between chips, or between racks. For each edge, the system has to decide where the value lives, how it moves, and when its consumer can run. Placement and scheduling are coupled: where the value lives affects when it arrives, and when it is consumed affects how long it must stay live.

接缝（Seam）是跨越物理边界的生产者-消费者边缘。该边界可能位于算子内部、内存层级之间、算子之间、芯片之间或机架之间。对于每一个边缘，系统必须决定数值存储在哪里、如何移动，以及消费者何时可以运行。放置（Placement）与调度（Scheduling）是耦合的：数值存储的位置影响其到达的时间，而消费的时间则影响其必须保持存活的时长。

Five seams organize almost everything that follows. I will refer to them by number for the rest of this series.

接下来的内容几乎全部由五个接缝组织而成。在本系列的后续部分中，我将通过编号来指代它们。

Seam	Edge	Boundary	Representative systems
S1	operand → arithmetic	inside a single operation	TPU systolic array
S2	HBM → SRAM	the on-chip memory hierarchy	FlashAttention
S3	operator → operator	kernel and scheduling boundaries	SambaNova, Groq, TileRT
S4	chip → chip	the scale-up domain	Cerebras, NVLink, ICI
S5	rack → rack	scale-out, and the building	TPU pod topology, DSX, Etched

I use *seam* because no single existing term spans all five boundaries. The individual ideas already have their own literatures — memory I/O, communication-avoiding algorithms, placement and routing — but the producer-consumer path is the common object.

我使用“接缝”（seam）这个词，是因为没有任何一个现有术语能跨越所有这五个边界。这些独立的理念各自都有相关的文献——内存 I/O、规避通信算法（communication-avoiding algorithms）、布局与布线——但生产者-消费者的路径才是它们共同的研究对象。

Follow these seams outward and the design target gets larger: from movement inside an operation, to communication between operators and chips, and eventually to the path a token takes through several kinds of machines.

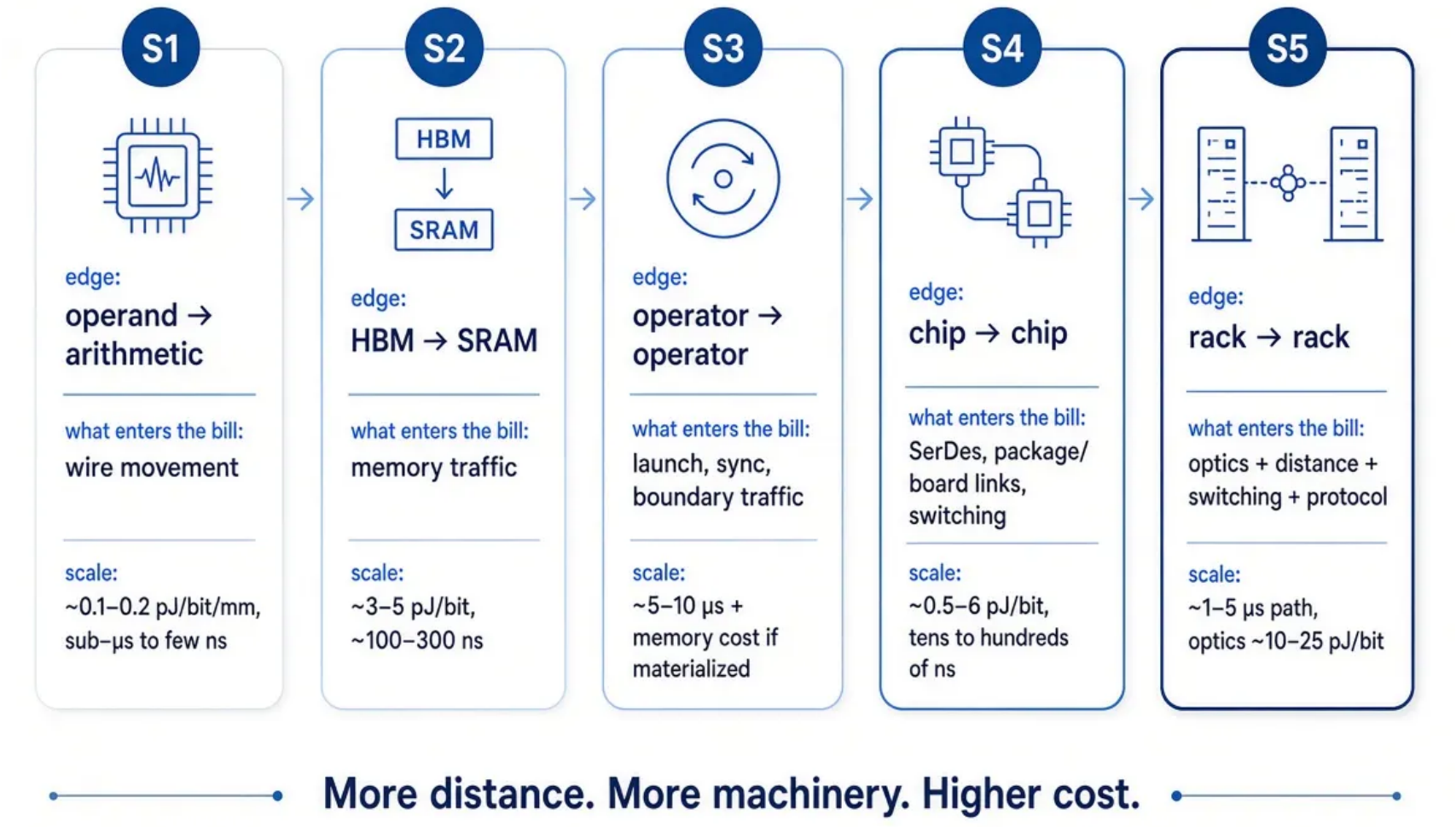
沿着这些接缝向外延伸，设计目标会变得越来越大：从操作内部的数据移动，到算子与芯片之间的通信，最终延伸到一个 Token 穿过数种机器的路径。

Why the seams matter 为什么接缝至关重要

Every time a dependency crosses a seam, it acquires a physical bill. The farther the seam reaches, the more things can enter that bill: distance, serialization, buffering,

retiming, switching, protocol, and synchronization.

每当一个依赖关系跨越接缝时，都会产生一份物理账单。接缝跨度越远，账单中包含的项目就越多：距离、序列化、缓冲、重定时、交换、协议以及同步。



The cost does not increase perfectly with the seam number—a good chip-to-chip link can be cheaper than an HBM access. What does increase is **how much machinery sits between producer and consumer**.

成本并不会随着接缝数量的增加而线性增长——一个优秀的芯片间链路（chip-to-chip link）甚至可以比一次 HBM 访问更便宜。真正增加的是生产者与消费者之间所存在的机制复杂度。

As arithmetic gets faster, those boundaries matter more. That is why AI systems keep pulling the expensive seams into the architecture.

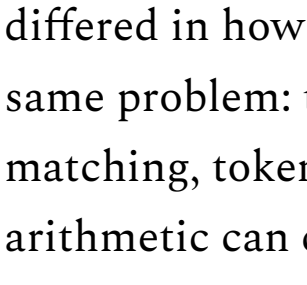
随着算力变得越来越快，这些边界变得愈发重要。这就是为什么 AI 系统不断地将昂贵的接缝引入到架构之中。

Why deep learning made an old idea economical

为什么深度学习让一个古老的想法变得经济可行

Dataflow is much older than the current accelerator industry. In the classical model, an operation becomes executable when its inputs are available. Execution follows producer-consumer dependencies rather than a program counter walking through a fixed sequence of instructions.

数据流（Dataflow）的历史远比当前的加速器行业悠久。在经典模型中，当输入就绪时，操作即可执行。执行过程遵循生产者-消费者的依赖关系，而不是由程序计数器按固定指令序列逐步运行。



That firing rule is a statement about causality: ADD cannot run until A and B exist. A dataflow machine carries that dependency into execution itself. Values are produced, routed, stored, matched with their consumers, and then enable the next work.

这条触发规则是对因果关系的陈述：在 A 和 B 存在之前，ADD 无法运行。数据流机将这种依赖关系带入执行过程本身。数值被产生、路由、存储、与它们的消费者匹配，然后开启下一项工作。

Researchers built machines around this idea decades ago. Dennis's static dataflow model, the MIT tagged-token architecture, and the Manchester Dataflow Machine differed in how they represented and scheduled dependencies, but they shared the same problem: the unit of work was small. If every scalar operation needs operand matching, token storage, routing, and scheduling, the machinery around the arithmetic can cost as much as the arithmetic itself.

几十年前，研究人员就围绕这一理念构建了机器。Dennis 的静态数据流模型、MIT 的标记令牌（tagged-token）架构以及曼彻斯特数据流机（Manchester Dataflow Machine）在如何表示和调度依赖关系上有所不同，但它们面临着共同的问题：工作单元太小。如果每个标量操作都需要操作数匹配、令牌存储、路由和调度，那么围绕算术运算构建的机制成本可能与算术运算本身一样高。

Deep learning changed that economics. The unit governed by one placement or scheduling decision can now be a tensor tile, a fused operator, or an entire pipeline stage containing millions of operations. The graphs are unusually regular and repeat at an enormous scale. Spending minutes or hours compiling a model makes sense when the resulting mapping runs billions of times.

深度学习改变了这种经济模式。现在，由一次放置或调度决策所控制的单元可以是一个张量分片（tensor tile）、一个融合算子，或者是包含数百万次操作的整个流水线阶段。这些图具有异常的规律性，并以巨大的规模重复。当生成的映射运行数十亿次时，花费数分钟或数小时来编译模型就变得非常有意义。

Another way to say the same thing is to change perspective. Instead of starting with a unit and asking what it executes, follow the value: where it is produced, how it is transformed and routed, and where it is consumed next.

另一种表达方式是转换视角。与其从一个单元开始并询问它执行什么，不如跟随值的足迹：它在哪里产生，如何被转换和路由，以及接下来在哪里被消耗。

So the working definition for this series: a dataflow-oriented system keeps producer-consumer dependencies explicit into execution, so placement, storage, movement, and scheduling can be organized around them. The seam ladder tracks how far that dependency reaches physically.

因此，本系列文章的实用定义是：面向数据流（dataflow-oriented）的系统将生产者-消费者依赖关系显式保留至执行阶段，以便围绕这些关系来组织放置、存储、移动和调度。接缝阶梯（seam ladder）则追踪这种依赖关系在物理上能触达多远。

Four architectures, four taxes

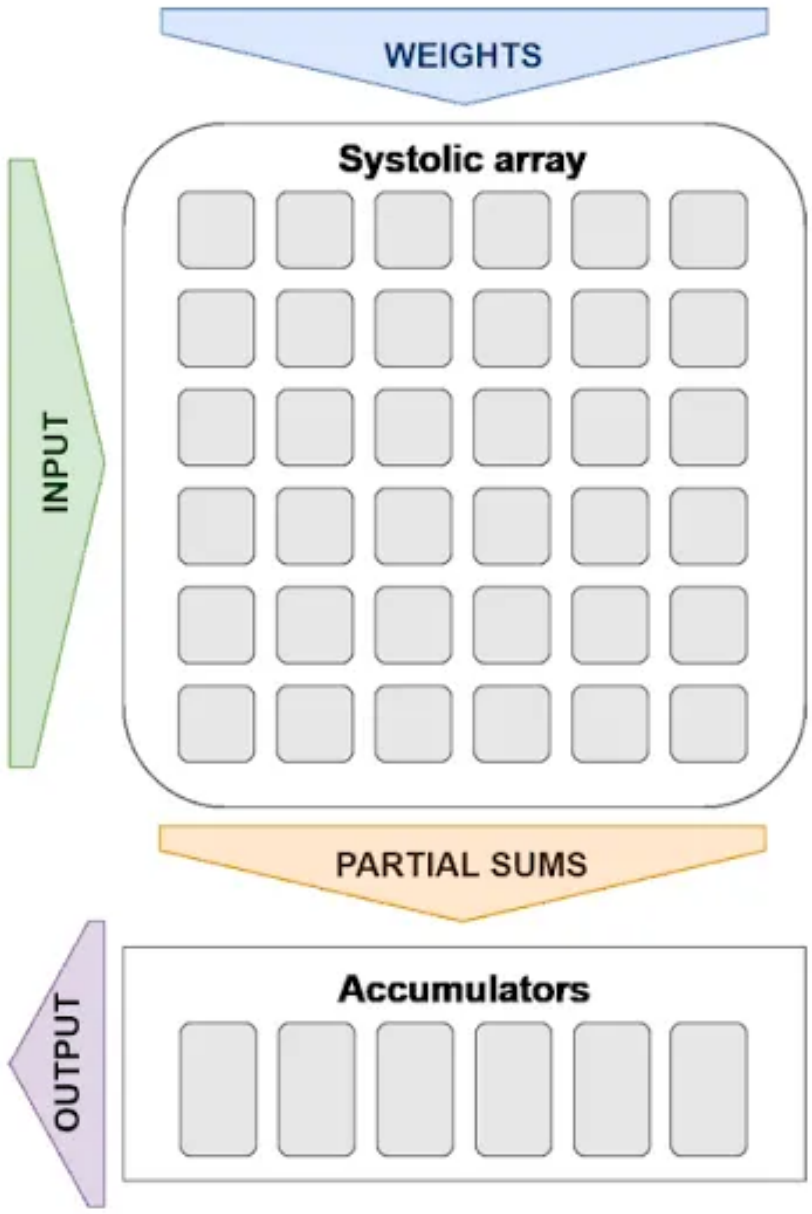
四种架构，四种税收

The chips that emerged after 2016 get grouped together as dataflow accelerators. That hides the interesting part. Each attacks a different tax that a conventional machine pays.

2016 年后出现的芯片被统称为数据流加速器。但这掩盖了有趣的部分。每种架构都在针对传统机器所支付的不同“税收”发起攻击。

TPU hard-wires the inner loop.

TPU 将内层循环固化在了硬件中。

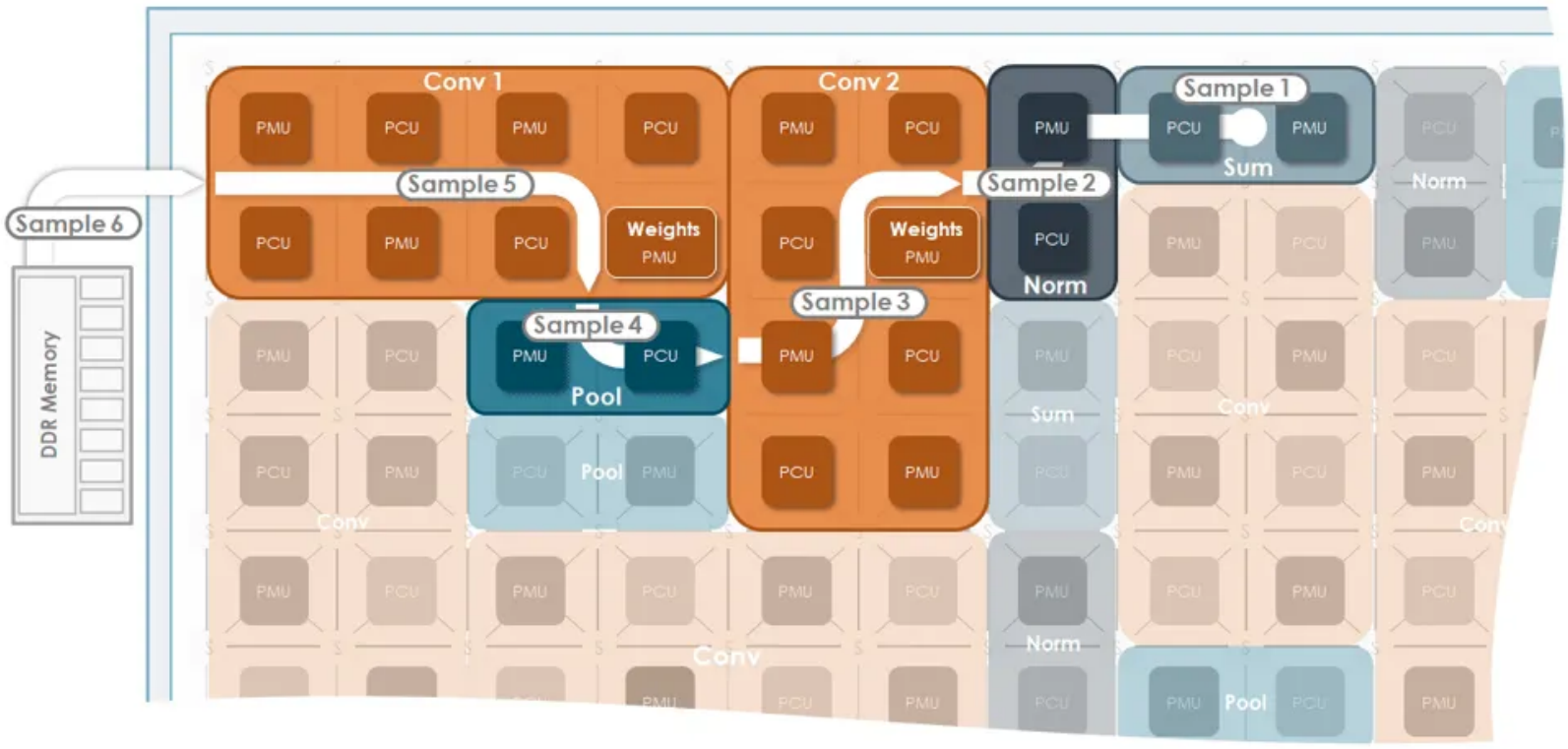


A TPU core is built around Matrix Multiply Units. Inside each MXU, operands flow through a systolic array and are reused as they move from one multiply-accumulate cell to the next. The compiler tiles larger tensor operations onto those units and keeps them fed. The tax removed is operand movement inside dense matrix multiplication: values stay close to the arithmetic instead of repeatedly traveling through a general memory hierarchy. TPU made S1 architectural.

TPU 核心围绕矩阵乘法单元（MXU）构建。在每个 MXU 内部，操作数流经脉动阵列（systolic array），并在从一个乘累加单元移动到下一个单元的过程中被重复利用。编译器将较大的张量操作拆解并平铺到这些单元上，并保持数据供给。这种方式消除了稠密矩阵乘法内部的操作数移动开销：数值始终靠近算术逻辑单元，而不是反复穿梭于通用的存储层级。TPU 实现了 S1 级别的架构化。

SambaNova maps the model onto a spatial machine.

SambaNova 将模型映射到空间计算架构上。

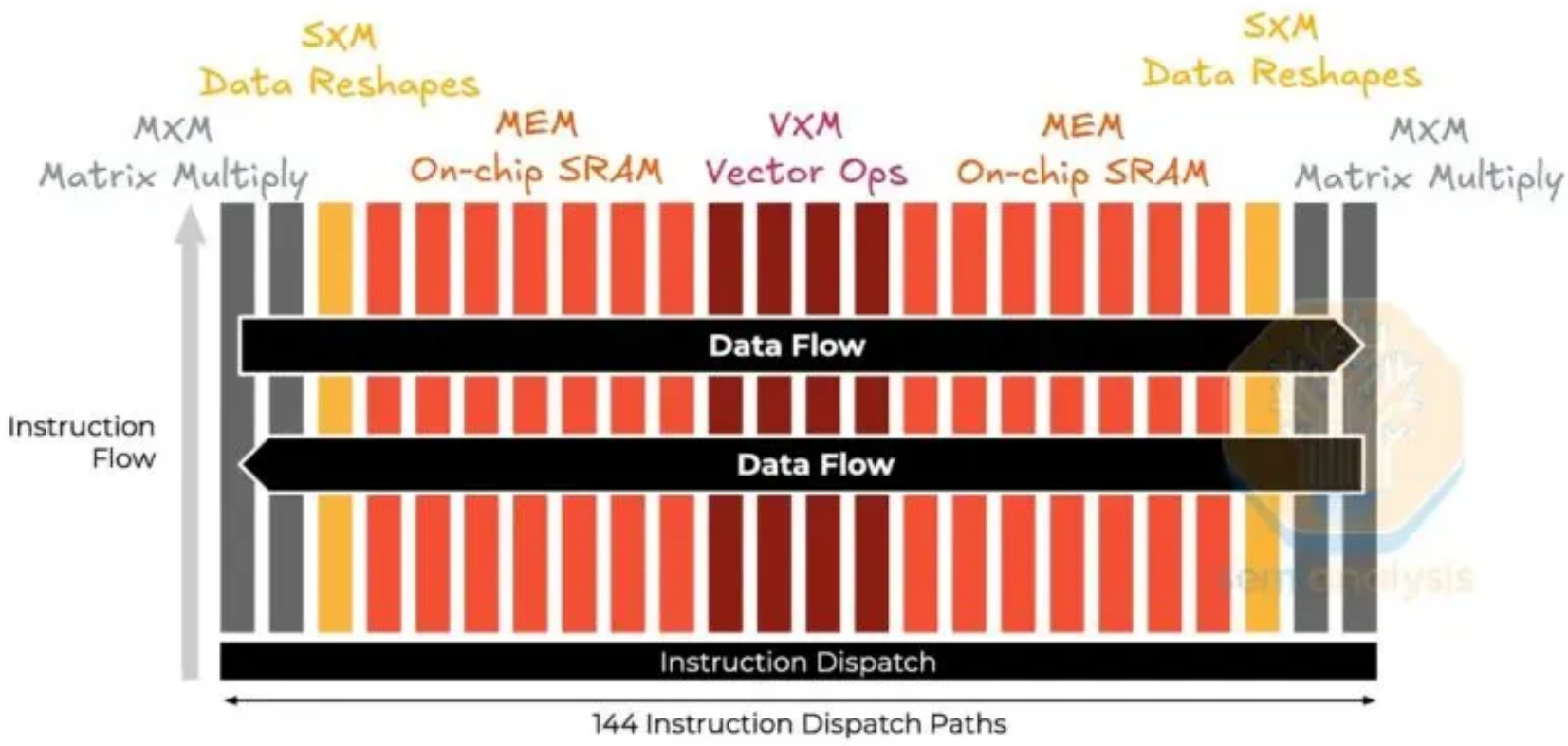


An RDU is a fabric of compute units, SRAM memory units, and configurable switches. The compiler decides where each operation runs, where intermediate tensors live, and how values move between them. Different regions of the chip execute different parts of the model at the same time; an activation can stream directly into its consumer instead of becoming a kernel output that later has to be read back. TPU fixed the flow inside one dominant operator. SambaNova made the compiler responsible for the pipeline across operators — which buys cross-operator locality and overlap, and makes physical placement part of the compile. SambaNova made S3 a compiler target.

RDU 是由计算单元、SRAM 存储单元和可配置交换机组成的织网。编译器决定每个操作在哪里运行、中间张量存储在哪里，以及数值如何在它们之间移动。芯片的不同区域同时执行模型的不同部分；激活值可以直接流向其消费者，而无需先作为算子输出写入内存再被重新读取。TPU 固化了单个主导算子内部的数据流。SambaNova 则让编译器负责跨算子的流水线——这带来了跨算子的局部性和重叠执行，并将物理布局纳入了编译过程。SambaNova 将 S3 变成了编译目标。

Groq turns the model into a timetable.

Groq 将模型转化为一张时间表。



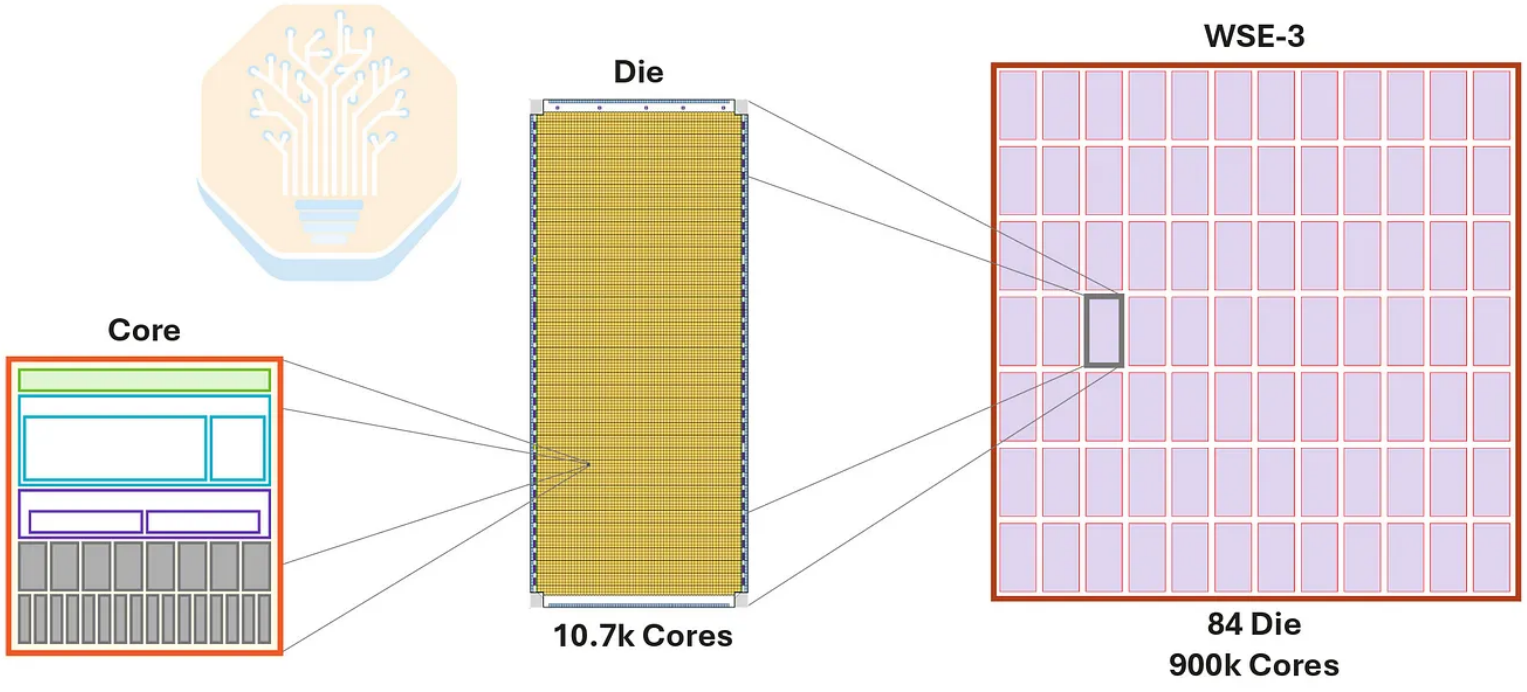
Groq gives the compiler similar authority and extends it to exact timing. SRAM is primary storage rather than a cache, and computation, memory access, and inter-chip communication are scheduled before execution. The compiler decides when a value is read, when a matrix operation starts, when a result is transmitted, and when the next chip consumes it. The result is deterministic latency: cache misses, dynamic arbitration, and much of runtime scheduling leave the critical path. SambaNova makes placement and routing explicit; Groq goes further and fixes the arrival schedule ahead of time.

Groq 赋予了编译器类似的权限，并将其扩展到精确的时序控制。SRAM 被用作主存储器而非缓存，计算、内存访问和芯片间通信在执行前就已调度完成。编译器决定了何时读取数值、何时开始矩阵运算、何时传输结果以及下一颗芯片何时接收。其结果是确定性的延迟：缓存缺失、动态仲裁以及大部分运行时调度都从关键路径中消失了。SambaNova 实现了布局和布线的显式化；而 Groq 则更进一步，提前固定了数据的到达时间表。

Cerebras moves inter-chip communication onto the wafer.

Cerebras 将芯片间通信转移到了晶圆上。

From Small Core to Massive Wafer



Cerebras also does compile-time placement and routing, but the substrate is different. The Wafer-Scale Engine is a large two-dimensional mesh of processing elements, each with local SRAM and a router. A conventional accelerator cluster repeatedly leaves one package, crosses SerDes and switches, and enters another. Cerebras stretches the on-chip mesh across the wafer, turning a large class of transfers that would have been network communication into short on-wafer movement. Cerebras made S4 a silicon decision.

Cerebras 同样进行编译时的布局和布线，但其底层架构有所不同。晶圆级引擎（Wafer-Scale Engine）是一个由处理单元组成的大型二维网格，每个单元都配有本地 SRAM 和路由器。传统的加速器集群需要反复离开封装，穿过 SerDes 和交换机，再进入另一个封装。Cerebras 将片上网格延伸至整个晶圆，将大量原本属于网络通信的传输转化为短距离的晶圆内移动。Cerebras 将 S4 变成了一个芯片层面的决策。

Worth noting what Cerebras did not absorb. WSE-3 carries around 44 GB of on-chip SRAM, which is not enough for a frontier model’s weights, so weights live in an external MemoryX appliance and stream onto the wafer layer by layer while activations stay resident. The most aggressive S4 answer in the industry deliberately keeps an S2-style seam. Hold onto that; it is the whole question the last post is about.

值得注意的是 Cerebras 没有吸收的部分。WSE-3 携带约 44 GB 的片上 SRAM，这不足以容纳前沿模型的权重，因此权重存储在外部的 MemoryX 设备中，并逐层流式传输到晶圆上，而激活值则保留在片上。业界最激进的 S4 方案刻意保留了 S2 式的接缝。记住这一点；这正是上一篇文章所讨论的核心问题。

The four taxes: **四种税：**

- **TPU** → reduce operand movement around matrix compute

  TPU → 减少矩阵计算周围的操作数移动

- **SambaNova** → reduce the memory and scheduling cost between operators

  SambaNova → 减少算子之间的内存和调度开销

- **Groq** → remove runtime timing uncertainty

  Groq → 消除运行时的时间不确定性

- **Cerebras** → remove package and network distance

  Cerebras → 消除封装和网络距离

The seams are different. The reason for pulling them into the design is the same.

虽然切入点不同，但将这些因素纳入设计的初衷是一致的。

(This is not about hardware architecture, so there are a lot of details I am not going into. For deeper dive, Irrational Analysis is always a good source. Like the following)

（本文并非讨论硬件架构，因此许多细节不再赘述。如需深入研究，Irrational Analysis 始终是很好的信息源。例如以下内容）

Irrational Analysis **非理性分析**

[unfinished draft] It's the dataflow, stupid.

[未完成草案] 笨蛋，问题在于数据流。

Irrational Analysis is heavily invested in the semiconductor industry...

> **stable compute graph + dynamic state and demand**
>
> **静态计算图 + 动态状态与需求**

This creates a natural tension. The compiler can specialize tensor layouts, kernel pipelines, model partitioning, communication schedules, and hardware placement. The runtime must still handle request admission, batch composition, KV allocation, cache routing, expert imbalance, and failures.

这产生了一种天然的张力。编译器可以对张量布局、算子流水线、模型分区、通信调度以及硬件放置进行专门优化。而运行时系统仍必须处理请求准入、批次组合、KV 缓存分配、缓存路由、专家负载不均以及故障恢复。

The industry is not moving that boundary in one direction. It is redrawing it, and the workload is doing the drawing.

业界并非在单一方向上移动这一边界，而是在重新划定它，且工作负载本身正是划定边界的画笔。

What that cost us at SambaNova

这让我们在 SambaNova 付出了什么代价

I spent years on these problems at SambaNova, where our compiler placed computation, memory, and communication routes onto a physical fabric. Spatial place-and-route worked best with static shapes. Production inference brought variable sequence lengths, batch sizes, and parallelism choices.

我在 SambaNova 工作的数年间一直致力于解决这些问题，当时我们的编译器负责将计算、内存和通信路由部署到物理架构上。空间布局与布线（place-and-route）在处理静态形状时效果最佳。然而，生产环境中的推理带来了多变的序列长度、批处理大小以及并行策略的选择。

We covered the gap with a library of precompiled mappings. Every new model and context length multiplied the combinations we had to generate and validate. Eventually we rebuilt around dynamic tensors and moved more binding and scheduling into the runtime, trading static efficiency for a system that could adapt.

我们通过预编译映射库来弥补这一差距。每增加一个新模型或上下文长度，我们需要生成并验证的组合就会成倍增加。最终，我们围绕动态张量进行了重构，并将更多的绑定和调度工作移至运行时，以牺牲静态效率为代价，换取了一个具备适应能力的系统。

That is the honest version of this story. Pulling a seam into the machine is not free, and production dynamism can push it back out. What was unusual then now shows up everywhere, from GPU kernels to rack-scale systems.

这是这个故事的真实版本。在机器内部引入“接缝”（seam）并非没有代价，生产环境的动态性可能会将其排挤出去。当时非同寻常的做法，如今已随处可见，从 GPU 内核到机架级系统皆是如此。

The seam moved inside the GPU

接缝移到了 GPU 内部

The GPU remains a SIMT machine. Around its performance-critical path, NVIDIA has steadily turned data movement and tensor execution into explicit producer-consumer pipelines.

GPU 仍然是 SIMT 架构。在其性能关键路径上，NVIDIA 已稳步将数据移动和张量执行转变为显式的生产者-消费者流水线。

On Hopper, the Tensor Memory Accelerator moves multidimensional tiles between global and shared memory asynchronously; one thread initiates movement while the rest of the block keeps working, and warps specialize into producers and consumers. Blackwell extends the separation: fifth-generation Tensor Core operations can be issued by a single thread, execute asynchronously, and store accumulators in dedicated Tensor Memory, with adjacent CTAs cooperating on larger matrix operations.

在 Hopper 架构上，张量内存加速器（TMA）在全局内存和共享内存之间异步移动多维分片（tiles）；一个线程发起移动，而线程块的其余部分继续工作，且 Warp 细分为生产者和消费者。Blackwell 扩展了这种分离：第五代 Tensor Core 操作可由单个线程发布，异步执行，并将累加器存储在专用的张量内存中，相邻的 CTA 则协作完成更大规模的矩阵运算。

A small number of threads increasingly issue asynchronous memory and tensor operations while dedicated hardware engines execute them. The programming model is following. CuTe and CUDA Tile expose tiles, layouts, copies, MMA operations, and

pipelines as explicit concepts, and Tile IR lets the compiler choose warp specialization, TMA use, prefetch depth, register pressure, and shared-memory staging. The useful question moved from what does this thread execute to how does this tile travel through the machine. Post 2 is entirely about this.

少量的线程越来越多地发布异步内存和张量操作，而专用硬件引擎负责执行它们。编程模型也随之演进。CuTe 和 CUDA Tile 将分片、布局、拷贝、MMA 操作和流水线暴露为显式概念，而 Tile IR 让编译器能够选择 Warp 专业化、TMA 使用、预取深度、寄存器压力以及共享内存暂存。核心问题已从“这个线程执行什么”转变为“这个分片如何在机器中流转”。第二篇文章将完全围绕这一点展开。

Software is converging on the same answer from above. TileRT — from the group behind the TileLang DSL — decomposes LLM operators into tile-level tasks and schedules compute, I/O, and communication together across multiple GPUs, compiling the decode graph into a single persistent megakernel. Work that used to be coordinated between kernel launches is now scheduled inside one long-running GPU program. It currently targets specific frontier models on eight-B200 systems, and has adopted a split where vLLM handles prefill and TileRT handles latency-sensitive decode.

软件层面正从上至下趋向于同样的答案。TileRT（源自 TileLang DSL 背后的团队）将 LLM 算子分解为分片级任务，并将多 GPU 间的计算、I/O 和通信统一调度，将解码图编译为单个持久化的 megakernel。过去需要在内核启动之间进行协调的工作，现在被调度在单个长时间运行的 GPU 程序中。它目前针对八卡 B200 系统上的特定前沿模型，并采用了一种拆分方案：由 vLLM 处理 Prefill，而 TileRT 处理对延迟敏感的 Decode。

This is worth being precise about, because it is the point of the whole series. TileRT is not a dataflow chip. It is a dataflow representation, on a SIMT substrate, doing what a spatial compiler does: making the operator-to-operator edge explicit and scheduling movement against computation. The operator boundary still exists; it no longer has to be a kernel-launch boundary. It is still an S3 edge, now controlled from inside the GPU.

这一点值得精确界定，因为它是整个系列的核心所在。TileRT 并非一款数据流芯片。它是一种在 SIMT 底层架构上的数据流表示，执行着空间编译器所做的工作：使算子间的边缘显式化，并针对计算调度数据移动。算子边界依然存在，但它不再必须是内核启动（kernel-launch）的边界。它仍然是一个 S3 边缘，只是现在由 GPU 内部进行控制。

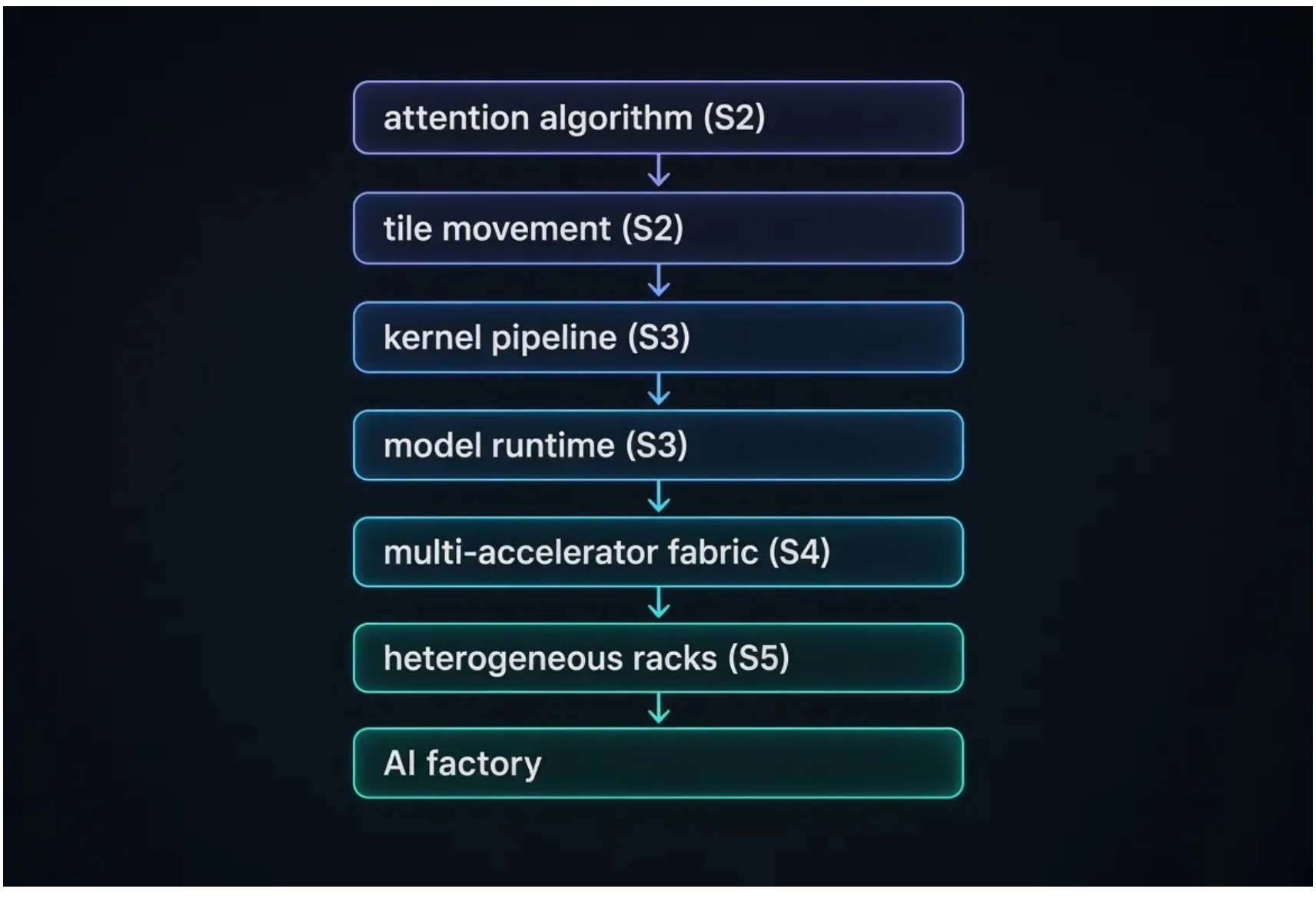
The same shift reached the network earlier than most people assume. SHARP in-network reduction arrived in the NVLink domain with third-generation NVSwitch on Hopper in 2022 — portions of all-reduce, reduce-scatter, and all-gather executing inside the switch fabric. NVLink 6 extends it. The switch no longer only forwards collective traffic; it performs part of the reduction.

同样的转变在网络领域发生的时间比大多数人预想的要早。SHARP 网络内归约技术于 2022 年随 Hopper 架构上的第三代 NVSwitch 进入了 NVLink 领域——all-reduce、reduce-scatter 和 all-gather 的部分操作在交换矩阵内部执行。NVLink 6 进一步扩展了这一功能。交换机不再仅仅转发集合通信流量，它还承担了部分归约计算。

Then the design boundary reaches the rack and the building. NVIDIA's Vera Rubin DSX reference architecture treats compute, networking, storage, power, cooling, simulation, and operations as one co-designed system, with tokens per watt and time to production as the target metrics.

随后，设计边界延伸到了机架和建筑层面。NVIDIA 的 Vera Rubin DSX 参考架构将计算、网络、存储、电力、冷却、仿真和运营视为一个协同设计的整体系统，并将每瓦特 Token 数和投产时间作为目标衡量指标。

Stack the progression: **将这一演进过程进行叠加：**



SIMT remains the flexible control substrate, handling irregular code, dynamic behavior, and the long tail of operations. Dataflow mechanisms shape the hot path, where locality, overlap, and predictable dependencies dominate. The convergence is layered, not wholesale.

SIMT 仍然是灵活的控制基座，负责处理不规则代码、动态行为以及长尾操作。数据流机制则塑造了热路径（hot path），即局部性、重叠和可预测依赖性占主导地位的部分。这种融合是分层的，而非全盘替代。

The graph now crosses processor types

图结构现在跨越了处理器类型

Once the producer-consumer path is explicit, there is no reason every stage has to run on the same kind of processor. Prefill values dense computation and memory capacity. Decode attention repeatedly reads a growing KV cache. MoE feed-forward execution values low-latency weight access and fast expert communication. Agentic workloads add CPU-based tools, retrieval, storage, and long-lived context. One homogeneous accelerator can execute all of these. Their physical needs are increasingly different.

一旦生产者-消费者路径变得明确，就没有理由要求每个阶段都在同一种处理器上运行。Prefill（预填充）阶段重视密集计算和内存容量。Decode（解码）注意力机制需要反复读取不断增长的 KV 缓存。MoE（混合专家模型）的前馈执行则看重低延迟的权重访问和快速的专家间通信。智能体（Agentic）工作负载还增加了基于 CPU 的工具调用、检索、存储和长效上下文。虽然单一的同构加速器可以执行所有这些任务，但它们的物理需求正变得日益迥异。

Google's eighth-generation TPUs show this at the product level. TPU 8t targets large-scale training. TPU 8i targets sampling, serving, and reasoning. Dense training favors the regular neighbor communication of a torus; MoE serving and reasoning benefit from a flatter network with fewer hops. So 8i triples on-chip SRAM to 384 MB — enough to hold much of a reasoning model's KV working set on silicon — doubles ICI bandwidth to 19.2 Tb/s, adds a dedicated Collectives Acceleration Engine in place of the previous generation's SparseCores, and abandons the 3D torus for a hierarchical, Dragonfly-inspired topology called Boardfly. Four-chip building blocks aggregate into groups over copper; groups connect through optical circuit switches. Maximum network diameter drops from sixteen hops to seven across a 1,024-active-chip pod. That is S5 being pulled into the compilation target: the topology was chosen by the shape of the model's communication graph.

谷歌的第八代 TPU 在产品层面展示了这一点。TPU 8t 针对大规模训练，而 TPU 8i 则针对采样、推理服务和推理。密集训练青睐环形拓扑（Torus）中规则的邻居通信；MoE 推理服务和推理则受益于跳数更少的扁平化网络。因此，8i 将片上 SRAM 增加了两倍，达到 384 MB——足以在芯片上容纳推理模型的大部分 KV 工作集；将 ICI 带宽翻倍至 19.2 Tb/s；增加了一个专门的集合通信加速引擎（Collectives Acceleration Engine）来取代上一代的 SparseCores；并放弃了 3D Torus，转而采用一种受 Dragonfly 启发的分层拓扑结构，称为 Boardfly。四芯片构建模块通过铜缆聚合为组，组与组之间通过光电路交换机连接。在 1,024 个活跃芯片的集群（Pod）中，最大网络直径从 16 跳降至 7 跳。这就是 S5 被纳入编译目标的过程：拓扑结构是根据模型通信图的形状来选择的。

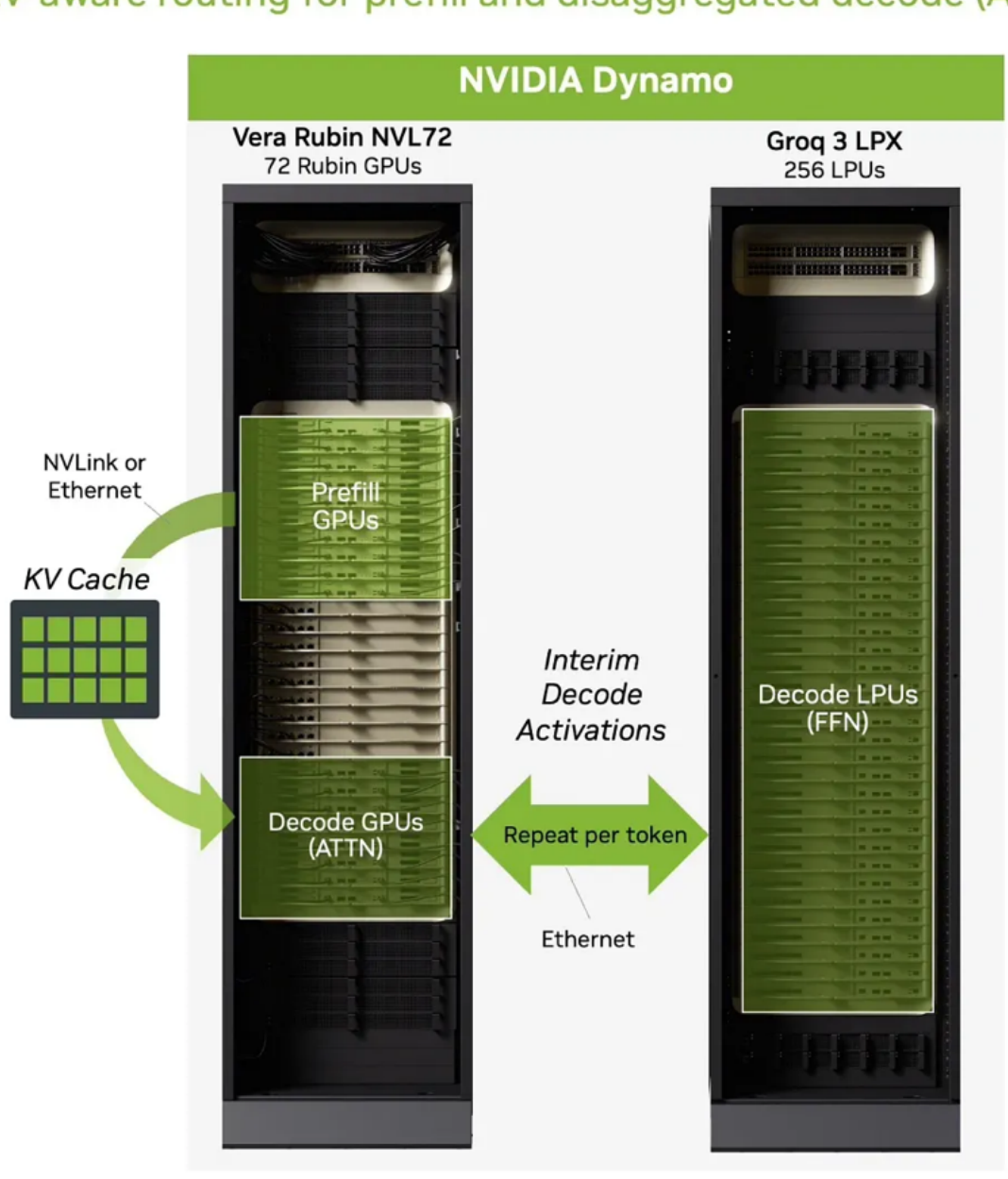
The clearest signal is in what 8i gave up. It has lower peak FP4 throughput than 8t, 10.1 PFLOPs against 12.6, and more HBM, 288 GB against 216. The inference chip traded arithmetic for memory and collectives.

最明显的信号体现在 8i 所做出的取舍上。它的 FP4 峰值吞吐量低于 8t（10.1 PFLOPs 对比 12.6 PFLOPs），但拥有更大的 HBM 容量（288 GB 对比 216 GB）。这款推理芯片用算力换取了内存容量和集合通信能力。

NVIDIA’s Vera Rubin platform splits work across processor types inside a single decode loop. Rubin GPUs handle prefill and decode attention, where memory capacity and dense throughput matter. The LPX system — 256 LPUs across 32 liquid-cooled trays, derived from Groq’s LPU — handles latency-sensitive FFN and MoE expert execution. NVIDIA calls it attention-FFN disaggregation, and describes GPUs and LPUs as jointly computing every layer of the model for every output token. For a forty-layer model, that is forty round trips per token across the seam: NVIDIA is betting that FFN weights resident in SRAM at rack scale beats the cost of crossing that boundary eighty times. Look at the energy table and you can see exactly what is being

traded against what.

NVIDIA 的 Vera Rubin 平台在单个解码循环中跨处理器类型分配任务。Rubin GPU 处理 Prefill（预填充）和 Decode Attention（解码注意力），这些环节对内存容量和稠密吞吐量要求极高。而 LPX 系统——由 32 个液冷托盘组成的 256 个 LPU，源自 Groq 的 LPU 技术——则处理对延迟敏感的 FFN（前馈网络）和 MoE（混合专家模型）专家执行。NVIDIA 将其称为“Attention-FFN 解耦”，并描述为 GPU 和 LPU 共同为每个输出 Token 计算模型的每一层。对于一个 40 层的模型，这意味着每个 Token 在两者衔接处要进行 40 次往返：NVIDIA 押注于“机架级 SRAM 中驻留的 FFN 权重”所带来的收益，将超过跨越该边界 80 次的成本。查看能效表，你可以清楚地看到究竟是什么在与什么进行交换。



Two details make the bet legible. Rubin CPX, NVIDIA’s own long-context inference GPU announced in 2025, was deprioritized once LPX arrived — the homegrown phase-specialized part lost to licensed dataflow silicon. And inside the LPX rack, eight-chip all-to-all clusters form the local groups of a dragonfly topology. Google reached for Dragonfly for MoE all-to-all in the same year, independently. When two teams with nothing in common pick the same topology, the workload is doing the

choosing.

有两个细节让这一押注变得清晰可见。Rubin CPX 是 NVIDIA 在 2025 年发布的自有长文本推理 GPU，但在 LPX 出现后便被降低了优先级——这款自研的阶段专用部件输给了授权的数据流（Dataflow）芯片。而在 LPX 机架内部，八芯片全连接集群构成了 Dragonfly 拓扑结构的局部组。同年，Google 也独立地为 MoE 的全对全（All-to-all）通信选择了 Dragonfly 拓扑。当两个毫无共同点的团队选择了相同的拓扑结构时，说明是工作负载本身做出了选择。

The unit of design is no longer one accelerator. It is the path taken by one token through several kinds of machines.

设计的单元不再是单个加速器，而是一个 Token 穿过多种机器所经过的路径。

The product boundary keeps moving

产品边界在不断移动

Etched's product reflects the same movement. It is now a "frontier inference cluster," with chips, packages, boards, cooling, interconnects, memory hierarchy, software, and manufacturing co-designed for prefill and decode — low-voltage inference and a cluster-scale HBM/SRAM memory system offered as system-level answers to power density, memory latency, and large-model capacity. These are company claims and still need independent validation, but the boundary of the product is the revealing part. Etched is not selling a chip that drops into a conventional rack.

Etched 的产品反映了同样的趋势。它现在是一个"前沿推理集群"，其芯片、封装、电路板、冷却、互连、内存层级、软件和制造都针对预填充（prefill）和解码（decode）进行了协同设计——将低电压推理和集群级 HBM/SRAM 内存系统作为应对功率密度、内存延迟和大模型容量的系统级方案。这些是公司的说法，仍需独立验证，但产品的边界才是揭示真相的部分。Etched 销售的不是那种可以直接放入传统机架的芯片。

At the far extreme, Taalas turned an individual model into custom silicon. Its thesis was "the model is the computer": the compiler's output becomes hard-wired hardware, with some adaptation retained through fine-tuning, and only a couple of metal layers customized per model. In August 2026, AMD agreed to acquire it.

在另一个极端，Taalas 将单个模型转化为定制硅片。其核心论点是“模型即计算机”：编译器的输出变成了硬连线的硬件，通过微调保留一定的适应性，且每个模型仅需定制几层金属层。2026 年 8 月，AMD 同意收购该公司。

What AMD actually wants is not yet clear, and the ambiguity is the interesting part. It might be the chips — hardwired decode silicon sitting alongside Instinct GPUs in a Helios rack, a third instance of the same heterogeneous split. Or it might be the flow: a design methodology that turns a trained model into taped-out silicon in roughly two months. The second reading is the consequential one, because it would mean the asset was never the hardware. It was the compiler, extended all the way through fabrication.

AMD 究竟想要什么尚不明确，而这种模糊性正是耐人寻味之处。它可能想要的是芯片——即在 Helios 机架中与 Instinct GPU 并排运行的硬连线解码硅片，这是同类异构拆分的第三个实例。或者它想要的是流程：一种能在约两个月内将训练好的模型转化为流片硅片的设计方法论。第二种解读更具深远意义，因为这意味着核心资产从来不是硬件，而是延伸至整个制造环节的编译器。

Either way, within eight months two aggressive forms of specialization — compile-time scheduling of computation and communication, and model-specific silicon — were absorbed by GPU vendors and aimed at inference. NVIDIA took Groq’s team, assets, and a license rather than the company. AMD bought Taalas outright.

无论如何，在八个月内，两种激进的专业化形式——计算与通信的编译时调度，以及模型专用芯片——都被 GPU 厂商吸收并瞄准了推理领域。NVIDIA 接收了 Groq 的团队、资产和许可，而非收购整家公司。AMD 则直接买下了 Taalas。

Where does dataflow stop?

数据流的边界在哪里?

Not every seam should be pulled into the machine. And how to pull also matters significantly. Tighter coupling improves locality and predictability, and costs flexibility. Inference makes that trade messy: sequence lengths are unknown, MoE routing is token-dependent, speculative decoding is stochastic, architectures keep changing. Cerebras keeps an S2 seam on purpose. We gave back static efficiency at SambaNova to survive production. The stopping condition should be computable from the frequency, volume, and predictability of each edge — which is the question for the

last post, and the reason the series has one.

并非每一个接缝都应该被拉入机器内部。如何拉入也至关重要。更紧密的耦合提高了局部性和可预测性，但牺牲了灵活性。推理让这种权衡变得棘手：序列长度未知、MoE 路由依赖于 Token、投机采样具有随机性、架构也在不断变化。Cerebras 特意保留了 S2 接缝。我们在 SambaNova 放弃了静态效率，以换取在生产环境中的生存。停止条件应当根据每个边缘的频率、容量和可预测性来计算——这是最后一篇文章要讨论的问题，也是本系列之所以有最后一篇的原因。

What makes dataflow a useful systems lens in 2026 is that it no longer identifies a family of accelerators. It describes the repeated process through which AI turns logical dependencies into physical architecture.

数据流之所以在 2026 年成为一个有用的系统视角，是因为它不再仅仅代表某一类加速器。它描述了一个重复的过程，通过这个过程，AI 将逻辑依赖转化为物理架构。

The first accelerator era asked which chip should run the graph. The next one asks how far the graph should extend into the machine.

第一代加速器时代探讨的是应该由哪种芯片来运行计算图。而下一个时代探讨的则是计算图应该延伸至机器的何种深度。

The series

Five posts, following the seams from the chip to the datacenter.

共五篇系列文章，顺着从芯片到数据中心的脉络展开。

S2 — the GPU. How the thread stopped being the unit of work: TMA, warp specialization, Tensor Memory, persistent kernels, and what tile-level programming models actually change.

S2 —— GPU。探讨线程如何不再是基本工作单元：TMA（张量内存加速器）、Warp 特化、张量内存、持久化算子（Persistent Kernels），以及分块级（Tile-level）编程模型究竟改变了什么。

S3 — the programming model. Why AI compilers increasingly place computation, memory, and communication before execution, what can be decided statically, and what has to remain dynamic.