EXPLAINED FROM ZERO

The KV Cache

Why an AI's short-term memory has become the most expensive real estate in the data center — and why it is reshaping the memory and storage industry.

Parts 1–3 assume nothing at all. No maths, no computing background.
Part 4 gets into hardware and names some companies — it's the industry map, and you're told where it starts.

WHAT YOU'LL BE ABLE TO EXPLAIN

Six things, by the end

In the order the deck teaches them, because each one leans on the one before it.

- **What a "layer" is** — and why a model stacks 80 of them, one after another
- **What Q, K and V mean** — the three things every word produces, and what each is for
- **Why K and V are saved but Q is binned** — the lopsided bit the whole cache rests on
- **Why the cache exists** — what would happen without it, in concrete numbers
- **How big it gets** — gigabytes per person, and the sum that produces them
- **Why storage companies suddenly care** — and what they're building about it

> **There is one equation in this deck.** It arrives on **slide 11**, in plain English, with no symbols at all. The written-down version sits on **slide 12**, on its own, and you are free to skip it.

Every metaphor here is chosen to be **useful and directionally true**, not literally exact. Where one bends, the slide says so.

PART 1 • THE GROUND FLOOR

Five words first, so nothing sneaks up on you

These five turn up constantly from here on. Each is one sentence. Nothing else is assumed.

model

The AI itself — the thing you type into. A single enormous file of numbers, plus the code that runs it.

weights

The numbers inside that file, learned during training. **A model essentially *is* its weights** — billions of them. When people say a model is "70 billion", they mean 70 billion weights.

layer

One stage of the model's processing. Text passes through all of them in order. **Slide 6 is entirely about this.**

GPU

The specialised chip that runs the model. Very fast at arithmetic, and it carries its own onboard memory. A data-centre one costs roughly **$30,000**.

byte, KB, MB, GB, TB — the sizes

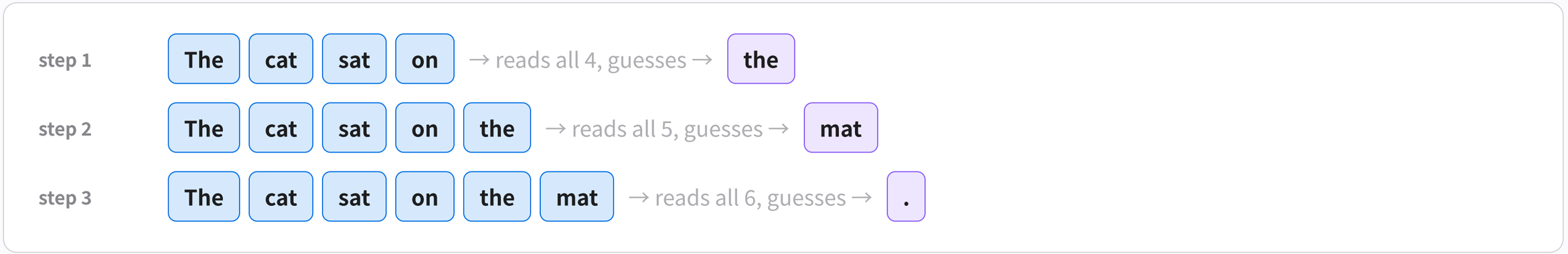
A **byte** is one character of text. A **KB** is a thousand bytes — a short email. An **MB** is a thousand KB — one photo. A **GB** is a thousand MB — an hour of video. A **TB** is a thousand GB — a laptop's whole drive. **Each step up is roughly a thousand times bigger.** Every size in this deck sits on that ladder.

One useful fact to park now: **one of the model's numbers takes 2 bytes.** That figure comes back on slide 18.

Anything else that turns up gets explained on the slide where it appears.

A model writes exactly one word at a time

It does not compose a sentence and then type it. It guesses one word, adds it to what it already has, then **re-reads the whole thing from the start** and guesses the next one.



To write word number **1,001**, the model re-reads words 1 through 1,000. **Every single time.** That one fact is the reason everything else in this deck exists.

Models actually work in **tokens** — roughly ¾ of an English word each. We'll say "word" throughout; nothing important changes.

Inside the model there are no words, only lists of numbers

The first thing that happens is that each word is swapped for a long list of numbers. After that the model only ever handles numbers.

cat becomes

```
[ 0.21, −1.07, 0.66, 0.02, −0.44,
  1.13, 0.37, −0.90, ⋯ ]
```

Shown: the first eight. The real list is **8,192 numbers long** — about four pages of numbers, for the single word *cat*.

- Think of it as a **coordinate on a giant map of meaning**. Words that mean similar things sit near each other on that map.
- "cat" and "kitten" land close together. "cat" and "Tuesday" land far apart.
- This list is what the model refines as it works. **The word itself never changes** — its list of numbers does.
- By the time it comes out the far end of the model, the list attached to a word has absorbed the meaning of everything around it.

If you only remember one thing: **a word inside the model is a long row of numbers.**

A model is a stack of layers — an assembly line for meaning

A layer takes one list of numbers per word and hands back one list of numbers per word, **the same size going out as coming in** — which is exactly why the next one can pick up where the last left off.

Each station works on the same document, then passes it along. Each keeps its own private drawers — nothing is shared sideways.

- **A big model has 80 layers.** Every word passes through all 80, in order. Nothing skips ahead.
- Each layer has **its own weights** — its own set of learned habits. Layer 3 and layer 40 are different specialists doing different jobs.
- Roughly: early layers handle surface things — spelling, is this a noun or a verb. Later layers handle meaning — what does "it" refer to, what is this text actually asking for.
- The output of layer 39 **is** the input of layer 40. **It's a relay, not a committee.**

Where the metaphor bends: the stations aren't literally "spelling, then grammar, then meaning." That division of labour is learned, messy and overlapping. **But the relay structure is exact** — 80 passes over the same data.

PART 1 • WHAT A LAYER ACTUALLY IS

Inside one layer there are two stations

All 80 layers are built the same way. Only the first station has anything to do with the cache — and that is where the rest of this deck lives.

STATION A — ATTENTION

Talk to the room

Every word **looks at the other words** and pulls in whatever context it needs. This is the only place in the whole model where words are allowed to see each other.

"it" reaches back and discovers it means "the mat".

This station needs the cache.

STATION B — THE THINKING BIT

Go back to your desk

Each word now sits alone and **digests what it just gathered**. It does not look left or right; it cannot see any other word.

Most of the model's raw knowledge lives here — the bulk of the weights are in this station.

No cache needed. Nothing to remember.

Station B is the bigger of the two and holds more of what the model knows. But because it never looks sideways, it has no memory problem. **Every expensive thing in this deck traces back to Station A.**

So the shape of the whole model is: **[talk to the room → go back to your desk] × 80**. That's it.

PART 2 • Q, K AND V

The problem Station A has to solve

Read this and answer instantly — you'll do it without noticing.

The cat sat on the mat because **it** was warm.

What is **it**?

The mat. Not the cat.

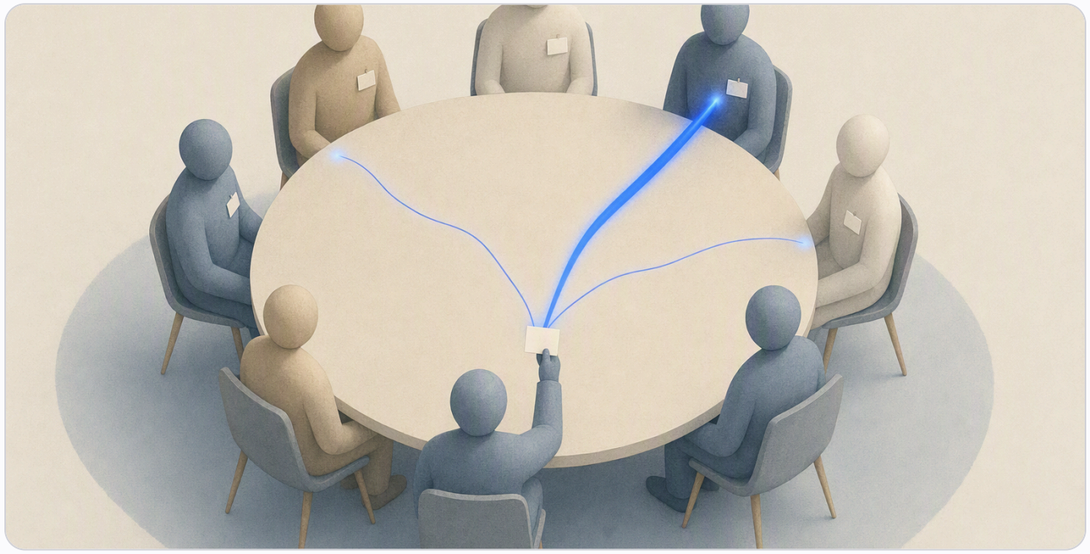
You resolved that in a fraction of a second using knowledge about warmth, surfaces and causation. The model has none of that intuition. It has to reach the same answer **using nothing but arithmetic on lists of numbers**.

Notice the answer depends on the word **warm**. Change it to "because it was tired" and the answer flips to the cat. The model must handle both — so it can't use a fixed rule.

PART 2 • Q, K AND V

Q, K and V: a meeting where everyone wears a name tag

Every word, at every layer, produces **three** lists of numbers from its own list of numbers. Here's what each is for — and why **a librarian's index card is not the book**.



One person asks. Everyone else is wearing a tag. The asker scans the tags, decides who's worth listening to, and blends what they say.

Q — Query

The question I'm holding up

"I'm the word *it*. Who here is a warm, flat thing you could sit on?"

K — Key

The name tag I'm wearing

How I advertise myself to anyone looking. *mat* wears: "flat surface, floor, indoors, object."

V — Value

What I actually say when called on

The **index card says where to find me; the book is what you get.** Being findable and being useful are different jobs.

Concretely, dropping the metaphor for one sentence: **a name tag is not words — it is another row of numbers**, 128 of them, exactly like the row on slide 5. All three rows come from that same source row, each squeezed through a different set of **weights the model learned** — three ways of looking at the same word.

PART 2 • Q, K AND V

The same sentence, one word at a time

Read this first: the tags below are written in English and the percentages are invented, so that you can see the machinery. Real tags are lists of numbers nobody wrote down. **The mechanism is exact; the labels are a translation.**

WORD IN THE ROOM	ITS NAME TAG (K)	MATCH WITH *IT*'S QUESTION	SHARE OF ATTENTION
The	function word, no content	very low	1%
cat	animal, furry, alive, subject of the sentence	medium	22%
sat	verb, past tense, posture	very low	2%
on	preposition, spatial relation	low	1%
the	function word, no content	very low	1%
mat	**flat surface, floor covering, indoors, can be sat on**	**very high**	**68%**
because	connective, signals cause	low	5%

Result: **68% of what "mat" says + 22% of what "cat" says + a little of the rest.** The numbers attached to *it* now carry mostly mat-ness. It has resolved the reference — and it did so by **blending**, never by picking a single winner.

What just happened, in three steps

This is the whole of attention. No symbols on this slide — just the three things that happen, in order, every time any word looks at the room.

STEP 1

Compare

Hold your **question** up against **every name tag in the room**, one at a time. Each comparison produces a raw score — how well that tag answers your question.

That's the "match" column you just read.

STEP 2

Turn scores into shares

Convert those raw scores into **percentages that add up to 100%**. A high score becomes a big share; a low one becomes a sliver.

That's the "share of attention" column.

STEP 3

Blend

Use those percentages to take a **weighted mixture of what everyone says** — 68% of mat, 22% of cat, and so on.

One new list of numbers comes out. That's the answer.

Note where each of the three lives: **your question (Q)** is used in step 1 only. **The name tags (K)** are used in step 1 only. **What people say (V)** is used in step 3 only. Hold on to that — it's the whole argument on slide 13.

That's it. Every other complication in a real model is a variation on these three steps.

 SKIP THIS SLIDE FREELY

And here is that written down, if you ever meet it

Nothing new happens on this slide. It is the previous slide in symbols, so that the notation isn't strange the first time you see it somewhere else.

$$\text{output} = \text{softmax}\left(Q \cdot K^{\top} / \sqrt{d} \right) \cdot V$$

SYMBOL	SAID OUT LOUD	WHAT IT IS
Q	"queue"	Your question — step 1
K	"kay"	Everyone's name tags — step 1
V	"vee"	What everyone says — step 3
·	"dot"	Compare these two — the multiplication that produces a match score
ᵀ	"transpose"	A bookkeeping mark. It turns the list of tags on its side so the comparison lines up. **Carries no meaning of its own.**
softmax	"soft max"	Turn raw scores into percentages that add to 100% — step 2
√d	"root dee"	Volume control. Keeps scores from growing so large that one word drowns out all the others

PART 3 • WHY WE CACHE

Why K and V are kept, and Q is thrown away

This lopsidedness is the foundation of the cache. It is **not** a cost decision — Q isn't binned to save room. It's binned because nothing will ever want it again.

PRODUCED WHEN THE MODEL PROCESSED…	WHO READS IT LATER?	NEEDED AGAIN?	SO…
K V of **"mat"**	Every word that comes after it — "because", "it", "was", "warm", and every word the model generates from here to the end of the answer.	**Yes — forever**	**Cache it**
Q of **"it"**	Only the step that created it. That step already produced its answer, and the model never revisits it.	**No — never**	**Bin it**

THE MEETING VERSION

Your question is asked once and answered once. Nobody re-asks it.

Your name tag stays on your chest all meeting — because the next person to speak will also need to scan the room.

THE CLINCHER

A cache saves you from **recomputing** something.

But every step needs a **brand-new** question, from a word that **didn't exist yet** at the previous step. There is nothing there to have cached.

And size isn't the argument either. One Q is a few KB — next to a cache measured in GB, that's **a millionth** of it. Storing it would still be storing something with zero future readers.

Every layer runs its own meeting, with its own name tags

This is the piece that makes the cache big. It isn't one table of name tags. **It's 80 tables** — one per layer, each completely different from the others.

THE WORD "MAT", AS IT TRAVELS UP THE STACK

layer 0	tag: "three letters, a noun, singular"
layer 20	tag: "object of the preposition *on*"
layer 55	tag: "a physical surface, in a room"
layer 79	tag: "the thing that is warm — likely referent"

Same word. Eighty different tags. All eighty must be kept, because all eighty meetings run again for the next word.

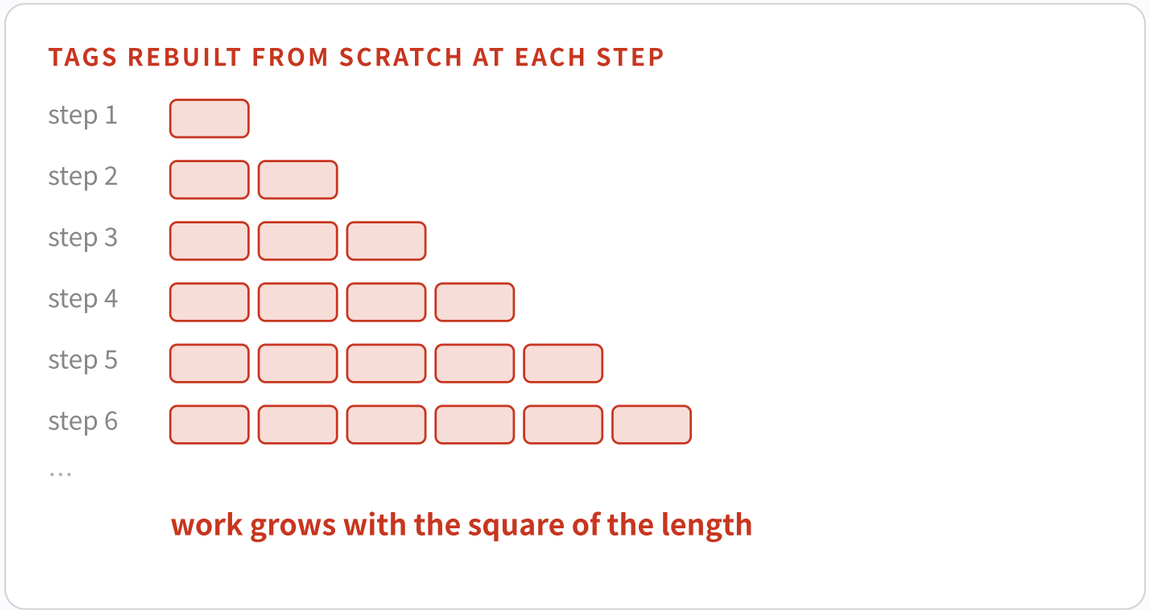
- Layer 40's attention only ever sees **layer 40's** tags. It cannot use layer 39's. They describe different things.
- So the cache is filed **per layer**.
- And one level down again: each layer runs **several meetings side by side** — one hunting for the grammatical subject, one for the topic, one for tone. These are called **heads**. A 70B model runs **64 heads per layer**.
- But those 64 heads do **not** keep 64 sets of tags. They are built to **share just 8 sets between them** — the single biggest saving in the whole story, and **slide 18 is where that 8 shows up**.
- So storing one word means storing **2 × 80 layers × 8 shared sets** worth of numbers. That multiplication is the whole cost story.

This is what people mean when they draw a KV cache as a stack of rows labelled **layer 0, layer 1, … layer 79**. Each row is a separate meeting keeping separate notes.

PART 3 • WHY WE CACHE

Without a cache: the whole room re-introduces itself, every time

Remember slide 4 — the model re-reads everything to write each new word. If it kept nothing, it would have to **re-make every name tag and every answer from scratch**, at all 80 layers, at every step.

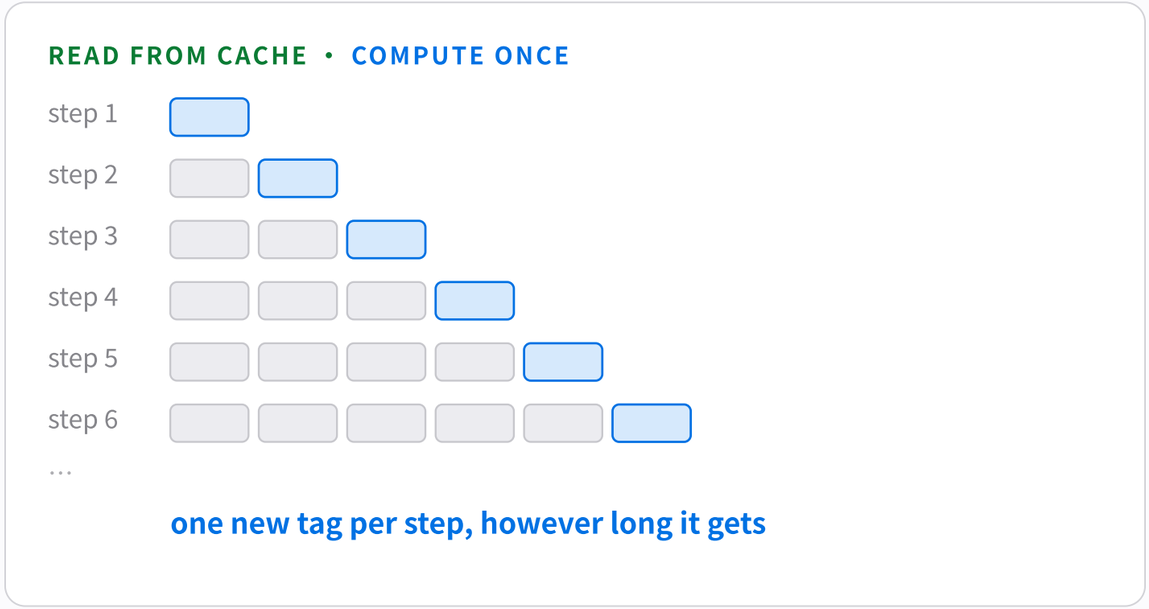


- Step 1 rebuilds 1 tag. Step 2 rebuilds 2. Step 500 rebuilds 500. Over a 500-word answer: **125,000 tag-rebuilds instead of 500**.
- And here's what makes it absurd: **the rebuilt tags are byte-for-byte identical to last time's.** What "mat" advertises at layer 55 depends only on the text before it, which hasn't changed.
- You are paying to compute the exact same numbers over and over until the answer ends.
- A long answer would slow down **quadratically** — the 500th word costing 500× the first.

This is what a naive implementation genuinely does. It's why the cache is the first thing anyone building a system that **runs** a model — the industry calls that **inference** — reaches for.

With a cache: write each name tag down, once

Keep every word's K and V the moment you first compute them. From then on each new word computes only **its own** — one new tag per step, forever.



- The trade is the oldest one in computing: **spend memory to save time.**
- Work per step stops growing. One new tag, then look up the rest.
- Nothing is approximated. The saved numbers are **exactly** what recomputing would have produced, so the output is identical. This is a pure win.
- Which is why every real system does it. It isn't an optimisation you switch on — **it's the default, and "no cache" is the hypothetical.**

> And now the bill arrives: **that memory has to physically live somewhere.**

PART 3 • WHY WE CACHE

Two halves of every request

Your prompt goes in as one big batch. After that the model runs **one word per pass**, reading the whole cache each time. The two halves behave — and cost — completely differently.

PHASE 1 — READING YOUR PROMPT

Filling the cache

All your prompt's words go through the model **together, in one pass**. Every layer writes tags for all of them at once.

Held up by: raw calculating speed
Grows with: how long your prompt is
The chip is: fully busy, doing big efficient sums

This is the half that gets skipped when you send the same document twice — the tags are already written. That's what **"prompt caching"** means on an **API** bill (an API is the paid, programmatic door into a model — what apps and developers use instead of a chat window).

PHASE 2 — WRITING THE ANSWER

Feeding off the cache

One word per pass. Each pass adds **one** tag per layer — but must **read every tag already stored**, at all 80 layers.

Held up by: how fast the cache can be hauled out of memory
Grows with: how big the cache has got
The chip is: mostly **waiting for data**

The arithmetic per step is trivial. The step is dominated by **moving the cache** to where the sums happen.

The sentence to keep: **generating text is not a calculating problem, it's a fetching problem.** A $30,000 chip spends most of this phase idle, waiting for numbers to arrive.

One picture of this: **paste a 50-page contract and ask one question.** Phase 1 swallows all 50 pages in seconds, flat out. Phase 2 then types the answer **one word at a time**, each word re-reading the whole contract's tags first. That's why answers start fast, then arrive at a drip.

PART 4 • WHERE THE CACHE LIVES

So how big does this thing actually get?

Every ingredient has already appeared in this deck. Multiply them together and you get the memory bill.

```
memory per word = 2 × layers × heads × numbers per head × bytes each
```

INGREDIENT	WHAT IT IS, IN THIS DECK'S LANGUAGE	70B MODEL	WHERE IT CAME FROM
2	A name tag **and** an answer — K and V — for every word	2	Slide 9
layers	Every layer runs its own meeting with its own tags	80	Slide 14
heads	**Sets of tags kept** per layer — not the number of meetings. 64 meetings run; they share 8 sets	8	Slide 14
numbers per head	How long one tag is	128	Slide 5 — that word's 8,192 numbers, split 64 ways
bytes each	How much space one number takes	2	Slide 3

`2 × 80 × 8 × 128 × 2 = 327,680` bytes — **320 KB for every single word**, prompt and answer alike. **And that 8 is already a discount:** without the sharing trick it would be 64, and every number here would be **eight times worse**. Next slide shows that bill.

Sanity check: the word "mat" is 3 letters — **3 bytes** of text. Its presence in the conversation costs **320,000 bytes** of memory. **A hundred thousand to one.**

PART 4 • WHERE THE CACHE LIVES

Real numbers — and the trick that saved us from 320 GB

Two levers keep this affordable. Both work by cutting a number in the sum you just saw.

LEVER 1 — SHARING THE TAGS (THE 8 ON SLIDE 18)

64 meetings per layer, but only **8 sets of tags between them**. Eight times less to store, for a small quality cost. Its name is **grouped-query attention**, and it is what made long conversations commercially possible.

LEVER 2 — COARSER NUMBERS

Store each number in **1 byte instead of 2** and the whole cache halves again: **40 GB → 20 GB**. This one *does* change the output slightly — a real quality-for-capacity trade, and one every serving team argues about.

MODEL, AT A FULL 128,000-WORD CONVERSATION	PER WORD	WHOLE CONVERSATION	WHAT THAT MEANS
8B model 32 layers, shared tags	128 KB	16 GB	About the same size as the model itself
70B model 80 layers, 8 shared sets	320 KB	**40 GB**	Over a quarter of a $30,000 chip's entire memory — for **one** conversation
70B, if nobody had invented the sharing trick 64 sets, not 8	2.5 MB	**320 GB**	**Impossible — more than any single chip made**

Now multiply by people. A hundred users each mid-conversation on the 70B model is **4 terabytes** of name tags that must all be instantly reachable. **No chip has 4 TB.**

PART 4 • WHERE THE CACHE LIVES

The memory ladder: desk, shelf, filing room, warehouse

Computers have always had this hierarchy. What's new is that AI put an **enormous, constantly-read pile of notes** on top of it — and it doesn't fit on the desk.

Same notes. The only thing that changes is how far you have to walk.

TIER	THE WALK	ROOM, VS THE DESK	SPEED, VS THE DESK
HBM memory stacked on the chip	**The open notebook on your desk.** Instant. Tiny. Extremely expensive.	—	—
DRAM the computer's main memory	**The shelf behind you.** Swivel round and reach.	~10× more	~100× narrower
Fast flash low-latency SSD	**The filing room down the hall.** A short walk, but a real one.	~20× more	~300× narrower
Ordinary SSD a normal big drive	**The warehouse across town.** Vast, cheap, and far.	~300× more	~500× narrower

Ratios rounded to orders of magnitude, against a current data-centre chip. The point is the shape of the ladder, not the exact rungs.

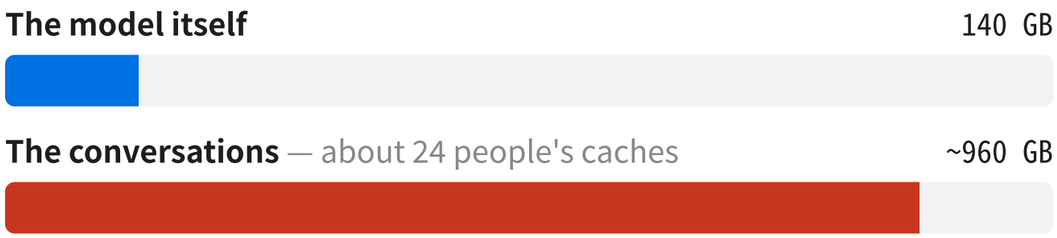
Per gigabyte, the desk costs roughly **a hundred times** what the warehouse does. That price gap is the entire commercial reason anyone wants to push the cache downwards.

PART 4 • WHERE THE CACHE LIVES

Why the desk is the wall

The surprising part, once you do the arithmetic: on a busy server the conversations take up **far more room than the model does**.

ONE 8-CHIP SERVER • 70B MODEL • 8 × 141 GB = 1,128 GB OF ON-CHIP MEMORY



The cache ends up roughly **seven times** the size of the model. Almost everyone's mental picture has this backwards.

- How many people you can serve at once is simply **how many caches fit in memory**. Twenty-four, here. The industry calls that number the **batch size**.
- And that number sets the price. The expensive part of each step — hauling the model's weights out of memory — is **shared across everyone being served together**. Half as many people, roughly twice the cost each.
- So a bigger cache doesn't just risk running out of room. **It directly raises what each word costs to produce.**
- Speed bites too: producing every word requires **reading the entire cache**. That's 960 GB hauled out of memory, per round of words.

Hence three ways out, in order of preference: **shrink it** (share tags, coarser numbers), **reuse it** (don't rebuild what you built last time), or **move it down the ladder**.

PART 4 • WHERE THE CACHE LIVES

Moving it down the ladder: the problem is punctuality, not speed

This is the counter-intuitive bit, and the reason a fast ordinary drive doesn't solve it. **The drives are wide enough. They're just not punctual enough.**

WHAT DRIVES ARE BUILT FOR

One big haul

Every benchmark measures how fast a drive moves a huge load in one go. The numbers look excellent — and they are.

It's the removal-van question: **how much can you shift in an afternoon?**

WHAT THIS ACTUALLY NEEDS

A small parcel, every few hundredths of a second, on a deadline

It's the school-run question: **not how fast the car goes, but whether it is ever late for pickup.**

Miss once and a **$30,000 chip** sits idle. Average speed is irrelevant.

And the sting: everyone in the batch advances together, so **the single latest delivery sets the pace for all twenty-four people.** A drive that is brilliant on average and occasionally late is worse, here, than one that is mediocre and never late.

Which is why "AI SSD" doesn't mean a faster drive. **It means a drive that knows what the model is going to ask for next** — so it can have it ready before the deadline.

 MECHANICS — SKIP IF YOU LIKE

Three reasons flash is occasionally late

You don't need these to follow the story. They're here because they explain why the fix has to happen inside the drive rather than in the flash chips.

- **Flash does housekeeping, and it can't be interrupted.** Memory chips can only be wiped in big chunks, so every drive constantly tidies up in the background. Every so often a request arrives just behind that tidying and waits. Rare — but rare is enough when there's a deadline every few hundredths of a second.
- **The drive has no idea what the model wants next.** To a drive, a storage location is just a number. Nothing tells it "this one is head 3 of layer 17, and it'll be wanted in 20 milliseconds" — so related notes end up scattered, and it can't fetch them ahead of time.
- **The parcels are the wrong size.** The chip wants pieces of about 512 bytes. Ordinary drives deal in pieces of 4,096. You drag **eight times more** than you need across the wire and throw most of it away.

> All three are fixable — but by the drive's **controller** (the small computer inside every drive that decides what to fetch and when), not by the flash chips themselves. **That's the whole reason a new category of drive had to be invented.**

INDUSTRY MAP — NAMES AND PART NUMBERS

Which is why the storage industry just re-organised itself

Everything on the last twenty slides is why NVIDIA went to the drive makers and asked for a new category of device. **This slide is the vendor landscape** — skip it unless you work in this world.

BOTTOM — THE FLASH ITSELF

Kioxia, Samsung, SK hynix, Micron

Makes the raw chips. Kioxia's **XL-FLASH** is a low-latency variant sitting in the gap between main memory and ordinary flash — roughly **13 µs** to first byte instead of ~80. (µs = a millionth of a second.)

Supplies faster raw material.

MIDDLE — CONTROLLER & FIRMWARE

InnoGrit, Phison, Silicon Motion

Makes the drive **understand what the model is doing** — scheduling, predicting, taming the late deliveries.

This is where the hard part lives, and where the differentiation will be.

TOP — THE SOFTWARE ABOVE

vLLM, Mooncake, NVIDIA CMX

Decides **what to keep, what to throw away, what to fetch early**. Treats the whole ladder as one pool rather than four separate things.

Where the policy lives.

NVIDIA Storage-Next asks drive vendors to build for direct access by the chip — 512-byte pieces, flash addressable as an extension of on-chip memory. Kioxia announced its **GP Series** answer on **16 March 2026** at GTC and shipped the first product, **GP1**, on **3 August 2026**: PCIe 6.0, XL-FLASH gen 2, **up to 10 million random reads per second** at 512 bytes. Samples to selected customers by end of 2026.

One number to keep straight: the widely-quoted **100 million** figure is a **future-generation target** and NVIDIA's stated ask — not the shipping GP1, which does 10 million. Sources: Kioxia press releases 2026-03-16 and 2026-08-03; Blocks & Files 2025-09-15.

THE WHOLE DECK

Seven sentences

If you remember nothing else, remember these — in this order, because each one causes the next.

1. A model writes **one word at a time**, re-reading everything before it to write each one.
2. So every word wears a **name tag (K)** and holds **something to say (V)**, letting later words find it.
3. Its **question (Q)** is asked once and answered once, so it's binned — **nothing will ever read it again**.
4. The tags come out identical every time, so we save them instead of rebuilding them. **That's the KV cache.**
5. Every layer keeps its own — **80 layers, 80 sets of tags** — which is why it reaches **gigabytes per person**.
6. On a busy server the cache is **bigger than the model**, and it sets both how many people you can serve and what each word costs.
7. So it's being pushed onto cheaper memory and flash — where the hard part isn't speed, it's **arriving on time**.

The one-line version: an AI's short-term memory is enormous, must be reachable in milliseconds, and grows with every word of every conversation — and the entire memory and storage industry is now rebuilding itself around that fact.

IF THIS WAS USEFUL

Three places to go next

I write and build in public about exactly this — how these systems actually work, and how to make them do real work.
Point a phone camera at any code below, or tap the link.

THE WRITING

ai-nate.com

Where explainers like this one live, plus what I'm building on **OPC**, the open runtime underneath it all.

THE PRODUCT

clawbot-ai.app

ClawBot — an agent that runs your actual operations, not just a chat window. Built on the stack described here.

THE COURSE

maven.com/ai-nate

Live cohort. We build these systems hands-on — the parts a deck can only point at.

Built by **AI Nate**. Corrections welcome — the numbers here are worked from published model configurations and vendor announcements, and every one of them is checkable.