The Future of Exabyte Scale Inference

Arthur Rasmusson

Grand Challenge

Internet-scale Analytics with AI Inference.

Goals of This Presentation

- Give you the tools to run **internet-scale analytics** with **AI Inference** using LLM sequence length extending techniques.
- Expand GPU & accelerator cache tiers with new page fault and speculative page migration techniques.
- **Improve performance.**
- **Eliminate wasted compute.**
- **Enable efficient scaling of production AI inference.**

Goals of This Presentation

- Supply you with useful code to **analyze** the **cache internals** in an AI inference environment.
- Help you quantify how **analysis** of **cache internals** can be used to enhance your analytics capabilities today.
- Help you quantify how **optimization of current technology** can greatly enhance the volume of data you can process with AI today.

This Presentation is **Cache Analysis Software**

Inference Cache Analysis Software + Slides / Formulas = **The Future of Exabyte Scale Inference** (CLSAC)

This Presentation
is produced by
human inputs
&
Open Source
Software.

This Software
bundles an
Open Source
Inference Server
which emits
analysis of inference data

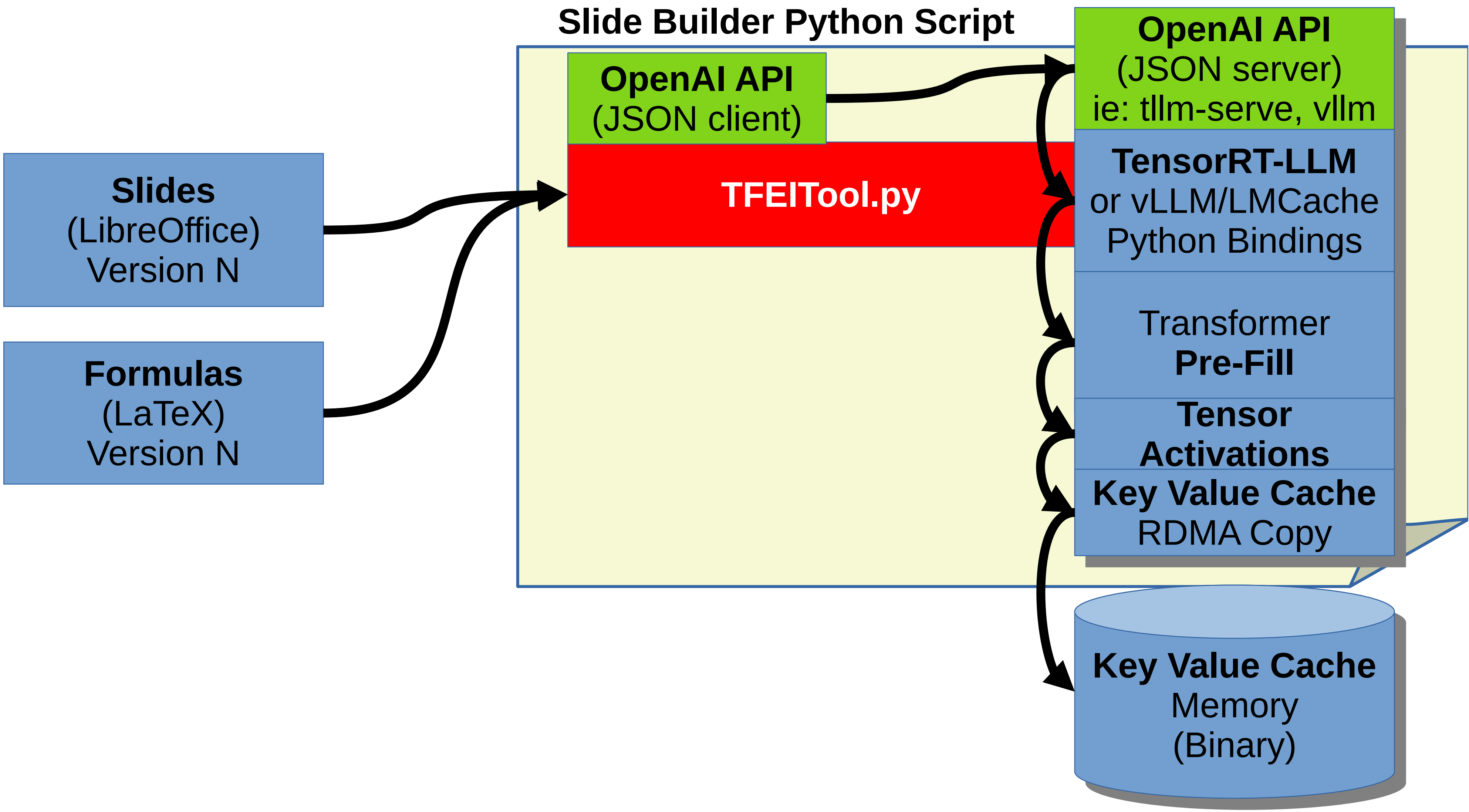
We will model the type system hierarchy & data transformations in AI Inference.

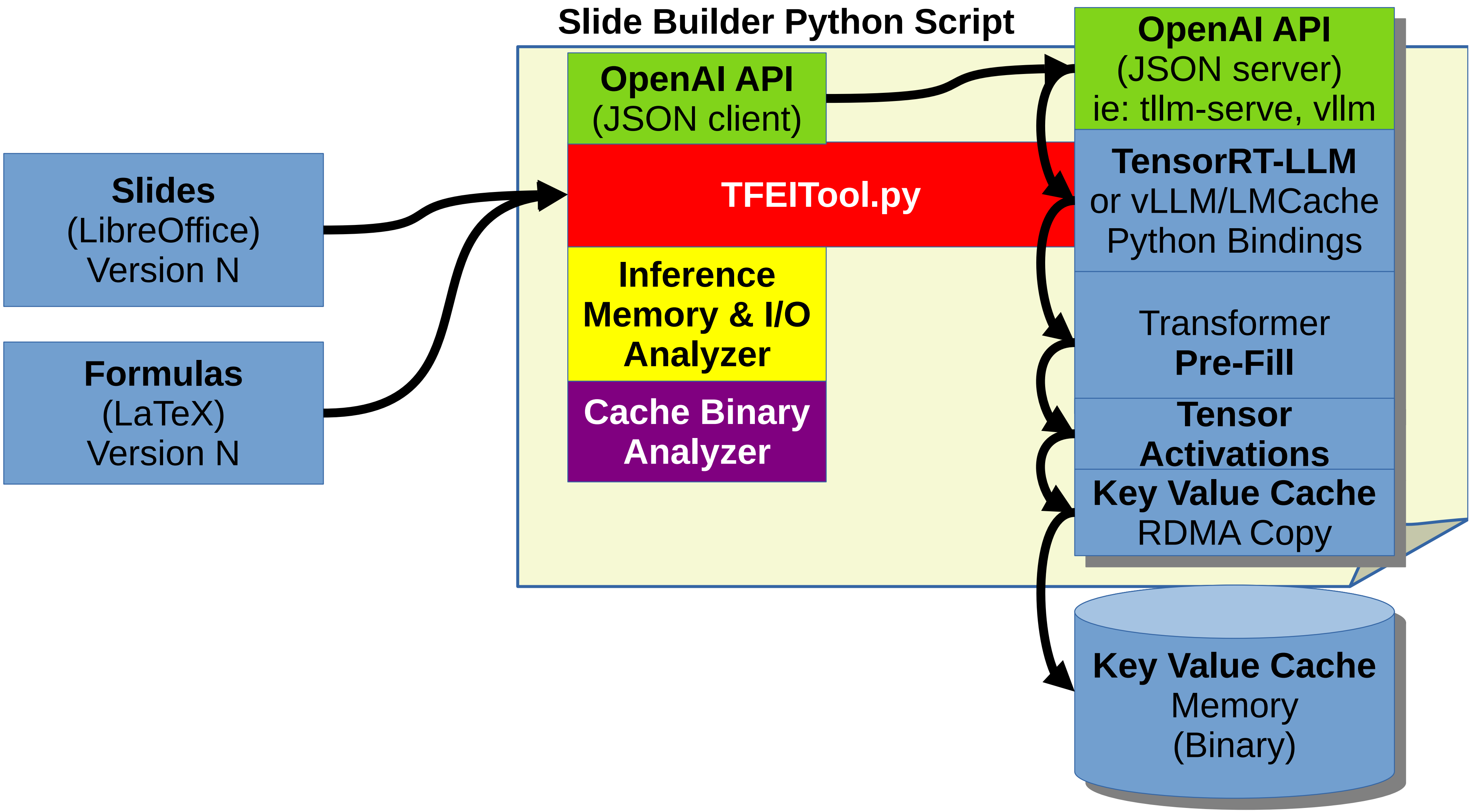
What you will learn.

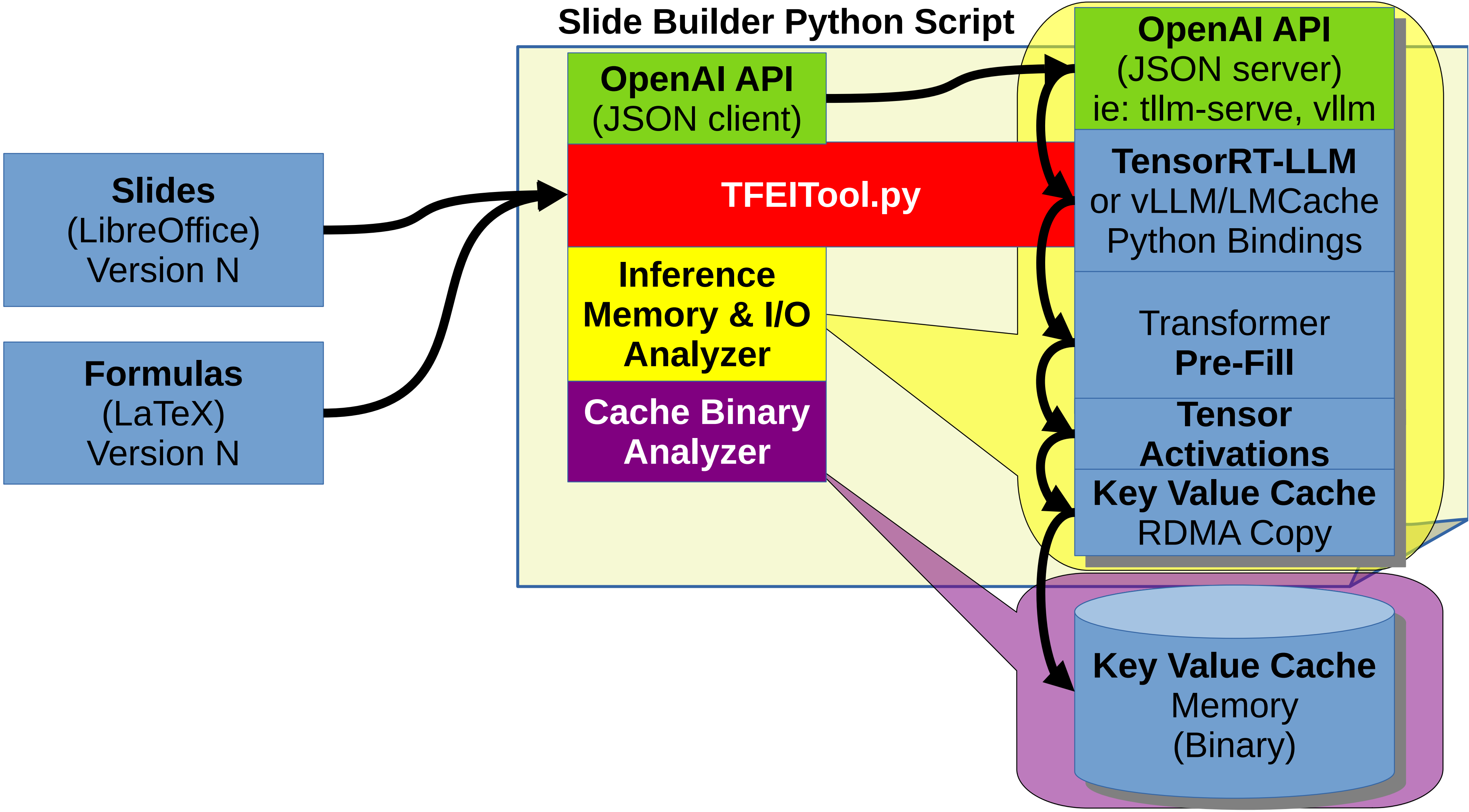
- **How plain text input maps to** internal type systems.
- How those type systems are used for tensor activations to predict tokens.
- **You learn to read a labeled hexadecimal view** of **binary vector cache memory stored on GPU** or in external memory tier known as a “Token Warehouse”.

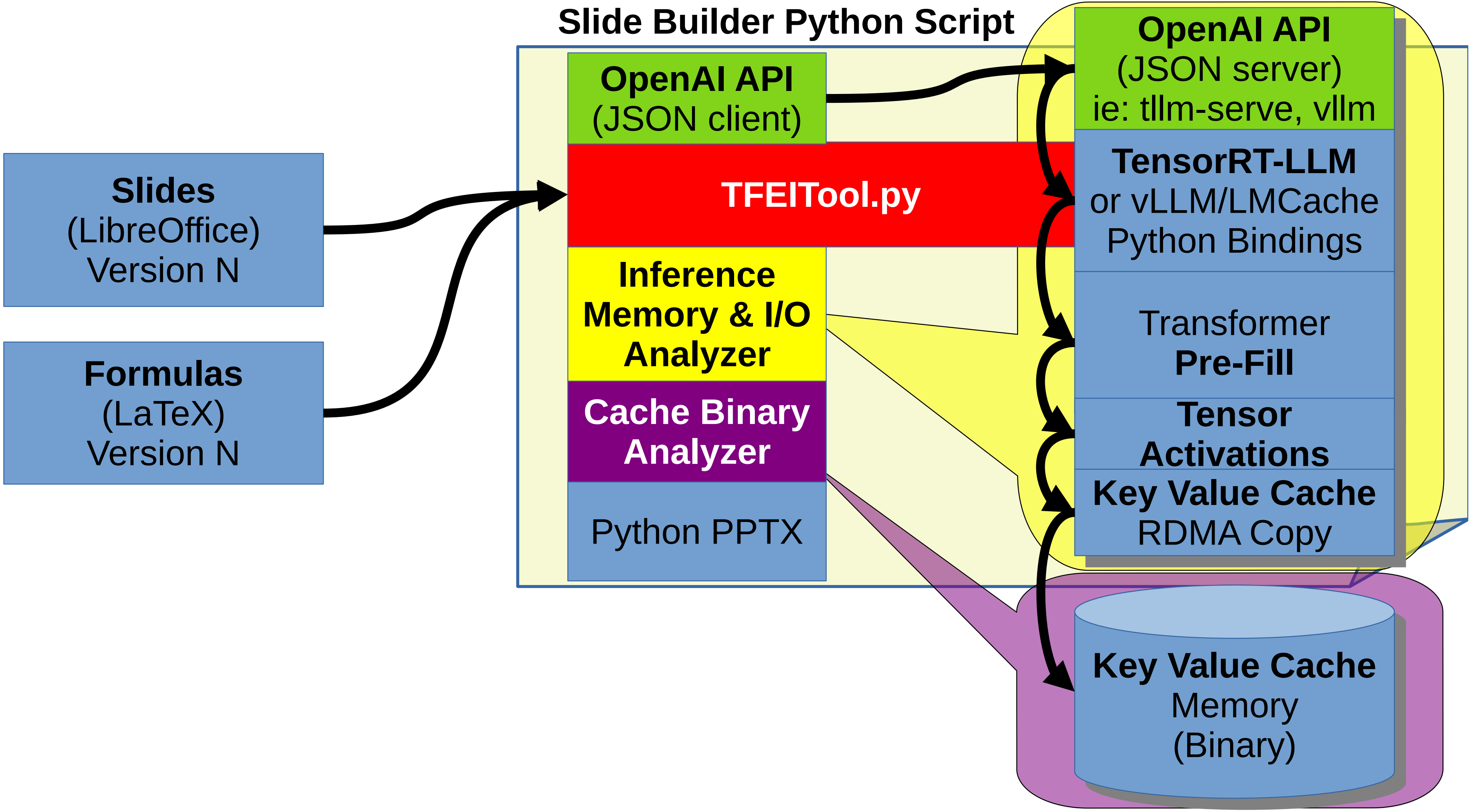
Slides
(LibreOffice)
Version N

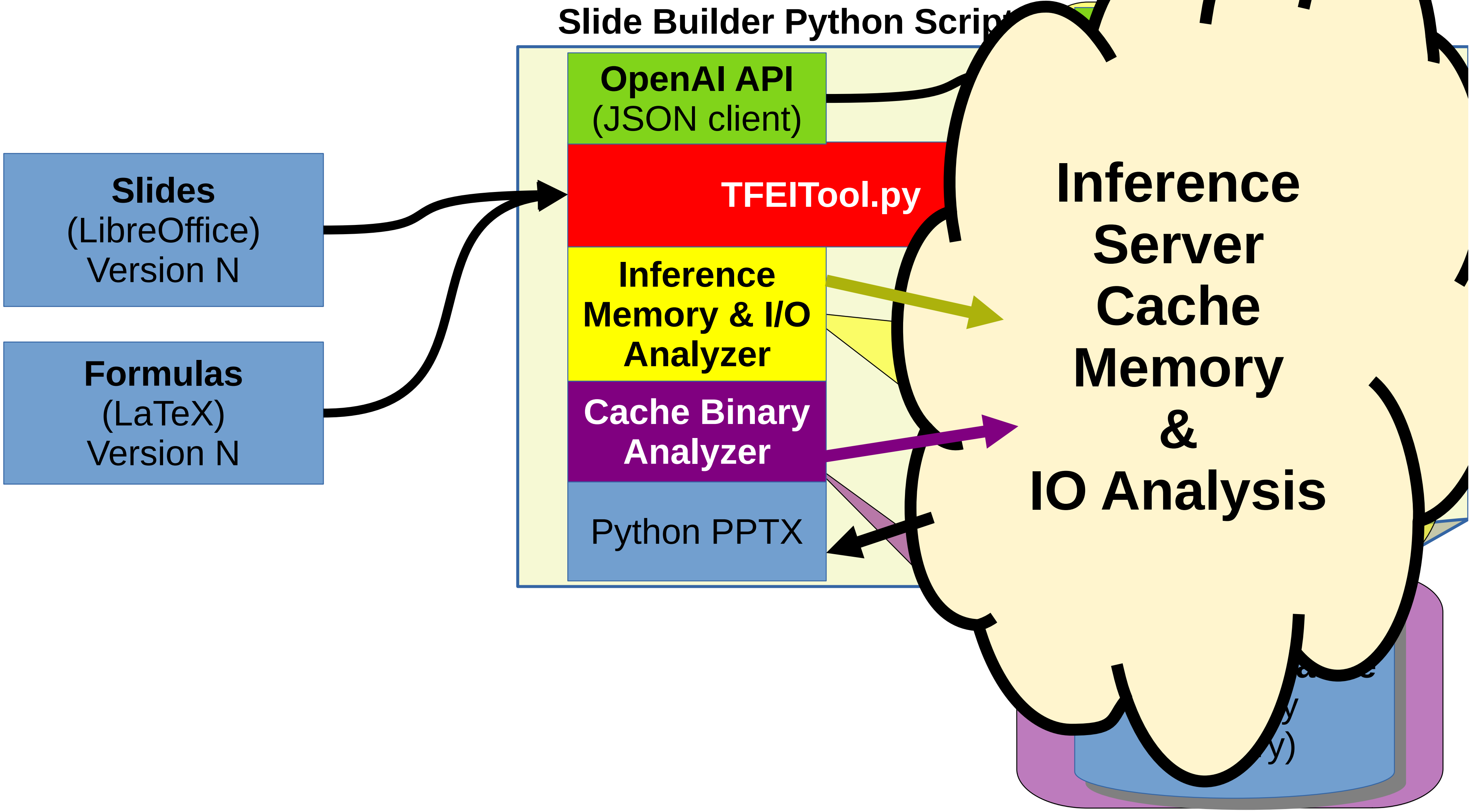
Formulas
(LaTeX)
Version N

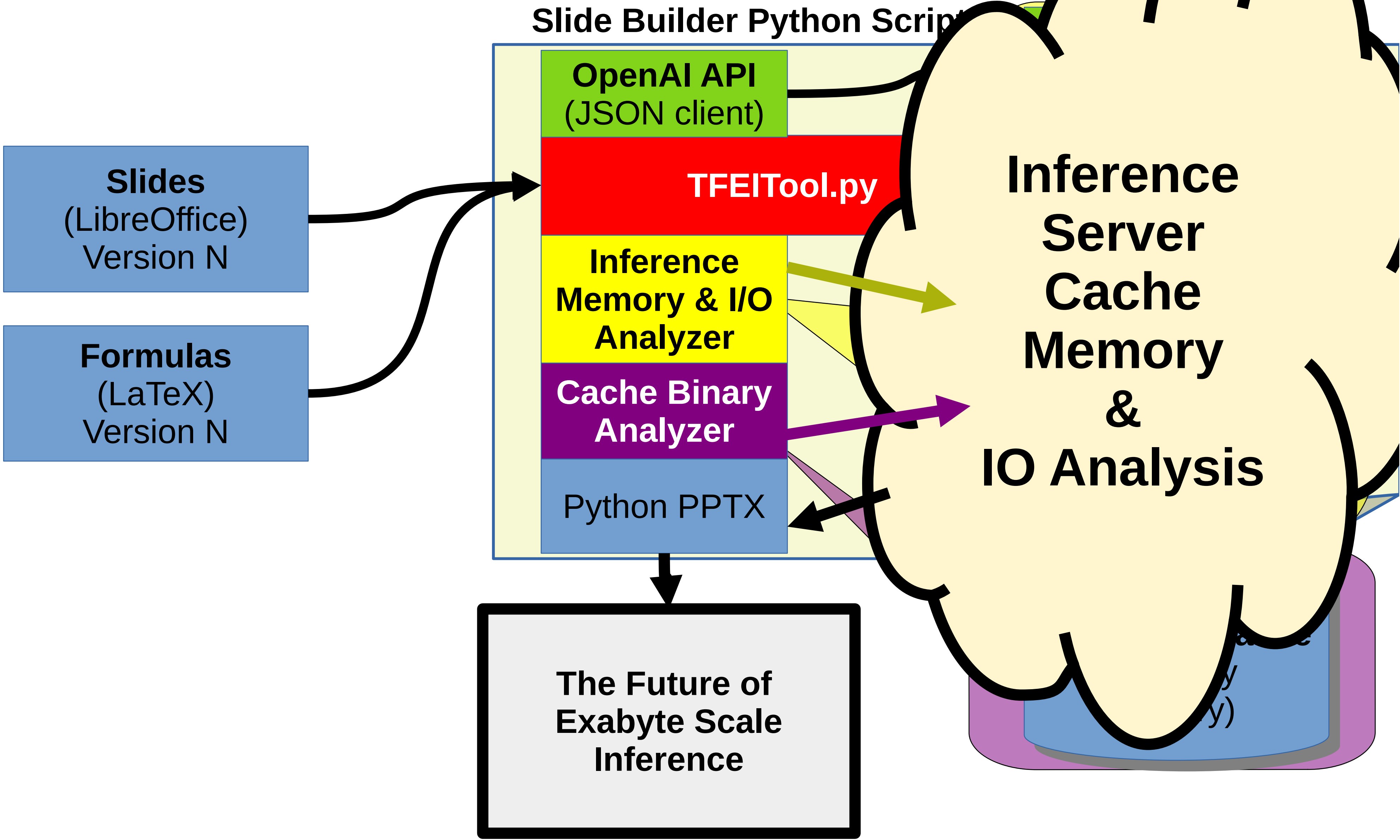

Slide Builder Python Script
OpenAI API
(JSON client)
TFEITool.py
OpenAI API
(JSON server)
ie: tllm-serve, vllm
TensorRT-LLM
or vLLM/LMCache
Python Bindings
Transformer
Pre-Fill
Tensor
Activations
Key Value Cache
RDMA Copy
Key Value Cache
Memory
(Binary)
Slides
(LibreOffice)
Version N
Formulas
(LaTeX)
Version N

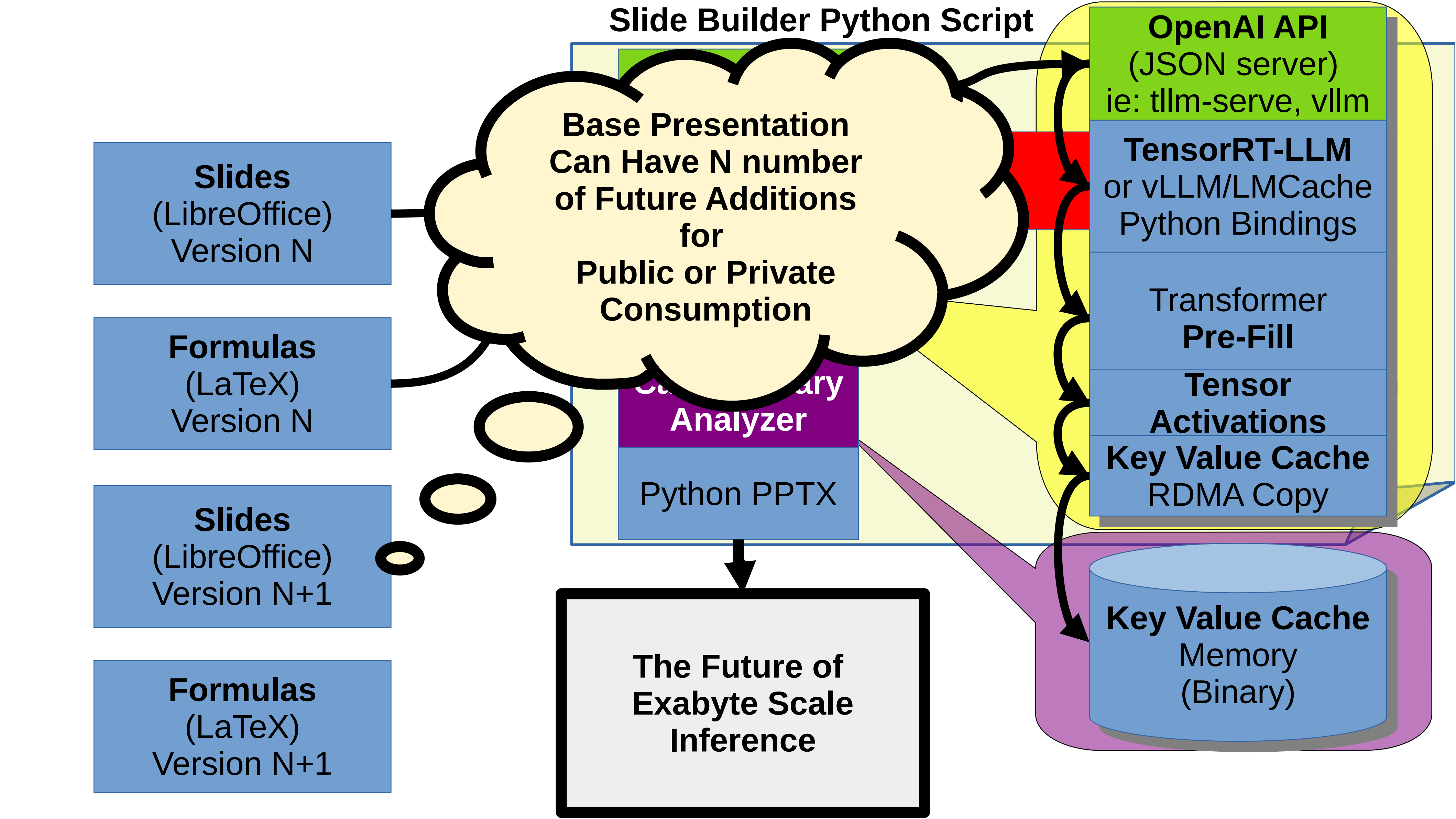

Slide Builder Python Script
Slides
(LibreOffice)
Version N
Formulas
(LaTeX)
Version N
OpenAI API
(JSON client)
TFEITool.py
Inference
Memory & I/O
Analyzer
Cache Binary
Analyzer
OpenAI API
(JSON server)
ie: tllm-serve, vllm
TensorRT-LLM
or vLLM/LMCache
Python Bindings
Transformer
Pre-Fill
Tensor
Activations
Key Value Cache
RDMA Copy
Key Value Cache
Memory
(Binary)


Slide Builder Python Script
Slides
(LibreOffice)
Version N
Formulas
(LaTeX)
Version N
OpenAI API
(JSON client)
TFEITool.py
Inference
Memory & I/O
Analyzer
Cache Binary
Analyzer
OpenAI API
(JSON server)
ie: tllm-serve, vllm
TensorRT-LLM
or vLLM/LMCache
Python Bindings
Transformer
Pre-Fill
Tensor
Activations
Key Value Cache
RDMA Copy
Key Value Cache
Memory
(Binary)


Slide Builder Python Script
Slides
(LibreOffice)
Version N
Formulas
(LaTeX)
Version N
OpenAI API
(JSON client)
TFEITool.py
Inference
Memory & I/O
Analyzer
Cache Binary
Analyzer
Python PPTX
OpenAI API
(JSON server)
ie: tllm-serve, vllm
TensorRT-LLM
or vLLM/LMCache
Python Bindings
Transformer
Pre-Fill
Tensor
Activations
Key Value Cache
RDMA Copy
Key Value Cache
Memory
(Binary)


Slide Builder Python Script
Slides
(LibreOffice)
Version N
Formulas
(LaTeX)
Version N
OpenAI API
(JSON client)
TFEITool.py
Inference
Memory & I/O
Analyzer
Cache Binary
Analyzer
Python PPTX
Inference
Server
Cache
Memory
&
IO Analysis


Slide Builder Python Script
OpenAI API
(JSON client)
TFEITool.py
Inference
Memory & I/O
Analyzer
Cache Binary
Analyzer
Python PPTX
Slides
(LibreOffice)
Version N
Formulas
(LaTeX)
Version N
The Future of
Exabyte Scale
Inference
Inference
Server
Cache
Memory
&
IO Analysis


Slide Builder Python Script
OpenAI API
(JSON server)
ie: tllm-serve, vllm
TensorRT-LLM
or vLLM/LMCache
Python Bindings
Transformer
Pre-Fill
Tensor
Activations
Key Value Cache
RDMA Copy
Key Value Cache
Memory
(Binary)
Slides
(LibreOffice)
Version N
Formulas
(LaTeX)
Version N
Slides
(LibreOffice)
Version N+1
Formulas
(LaTeX)
Version N+1
Base Presentation
Can Have N number
of Future Additions
for
Public or Private
Consumption
Analyzer
Python PPTX
The Future of
Exabyte Scale
Inference

Cache Analysis Slides
are
Reproducible.

How To
Reproduce Cache Analysis Slides

1) **Download** TFEITool, will be published at:
   https://github.com/arthurrasmusson/TFEITool

2) **Install** Inference Server Python bindings
   (e.g. TensorRT-LLM).

3) **Run** TFEITool in Python3.

Open your PowerPoint slides to see graphs.

Agenda

1. **Thesis**
(AI Re-Learning OS, Hypervisor, & Compiler Efficiency)

2. **How Transformers Work**
(The Basics)

3. **Trends in Long Context**
(Sequence Length and AI’s Moore’s Law)

4. **Cache Memory Structure**
(How LLMs Store & Load Cache Memory)

5. **Internals of Long Context Inference**
(Solutions for 30-40 Million Character Inputs & Beyond)

6. **Long Context Uses**
(Analyze Chats, High Resolution Video,
Audio, Documents, Decompiled Binaries, ect..)

Arthur Rasmusson

Creator of **Paged Attention over RDMA**

- **Arc Compute**
  Engineering Co-Founder
- **Open-IOV.org**
  Community Founder,
  Open Source Code & Documentation Contributor
- **Cohere**
  Model Efficiency Engineer
  GPU Infrastructure Engineer
- **Weka**
  Principal AI Engineer
  Started Augmented Memory Grid project
- AI Open Source Contributor

Open Source Community, Founding Engineer: Open-IOV.org

- Encouraging adoption of **GPU virtualization**.
- **Open Documentation of vGPU internals** (Hardware, Firmware, Drivers, Userspace).
- **Open Source programs** for GPU Virtualization.
- Encouraging collaboration between open source and closed source **industry stakeholders**.

Open Source, PAoR

- Created **Paged Attention over RDMA (PaoR)** as CC- By-4.0 creative commons technique for inference.
- **Open Source** code used in NVIDIA Dynamo & **NVIDIA NIXL, NVIDIA TensorRT-LLM**, & **LMCache** project (vLLM).
- **Merged TensorRT-LLM PR 3209 in NVIDIA GitHub upstream:** Feature: KV Cache GPUDirect Storage
- Wrote **Python-Native-libCuFile LMCache** dependency for PAoR.
- PAoR is the core IP for **NVIDIA Dynamo**.
- NVIDIA moved **KV Cache GPUDirect Storage** feature from PR 3029 in TensorRT-LLM to Dynamo’s NVIDIA Inference Xfer Library (NIXL).
- NIXL alternative dependency for Python-Native-libCuFile in LMCache.

Select Commercial Work

- **Just-In-Time Training** & **Just-In-Time Inference** at Cohere.
- Created **Augmented Memory Grid (AMG)** feature for NeuralMesh by Weka.
- Compile time and runtime validation of **Nvidia TensorRT** and **TensorRT-LLM** on **NVIDIA Grace CPU platform** (Cohere early ARM64 CPU adoption for AI inference, led to scale deployment in Coreweave for Cohere).

1
Thesis

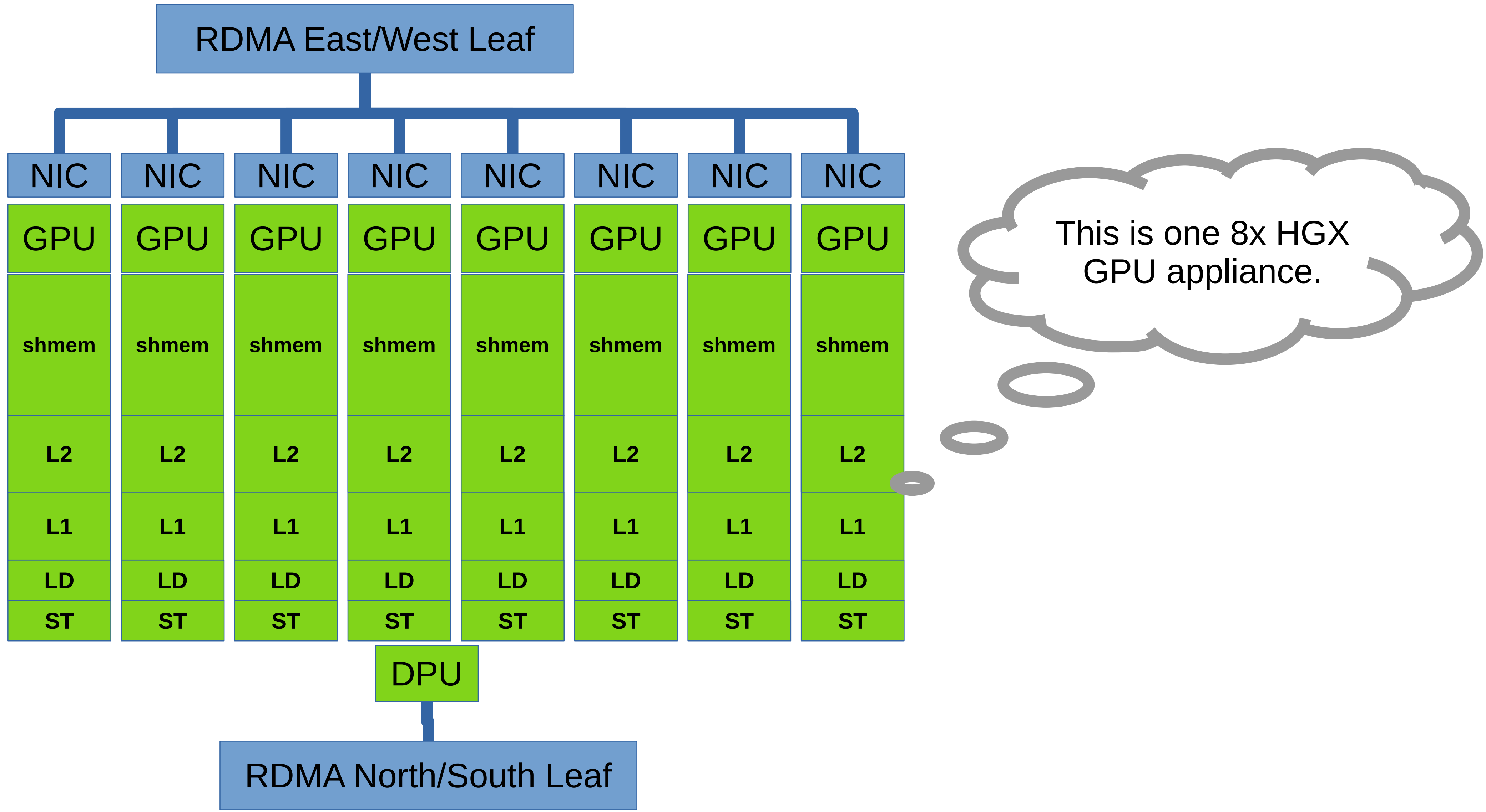
“AI is relearning
Operating System,
Hypervisor,
& **Compiler**
efficiency techniques.”

“AI is relearning
Operating System,
Hypervisor,
& **Compiler**
efficiency techniques.”

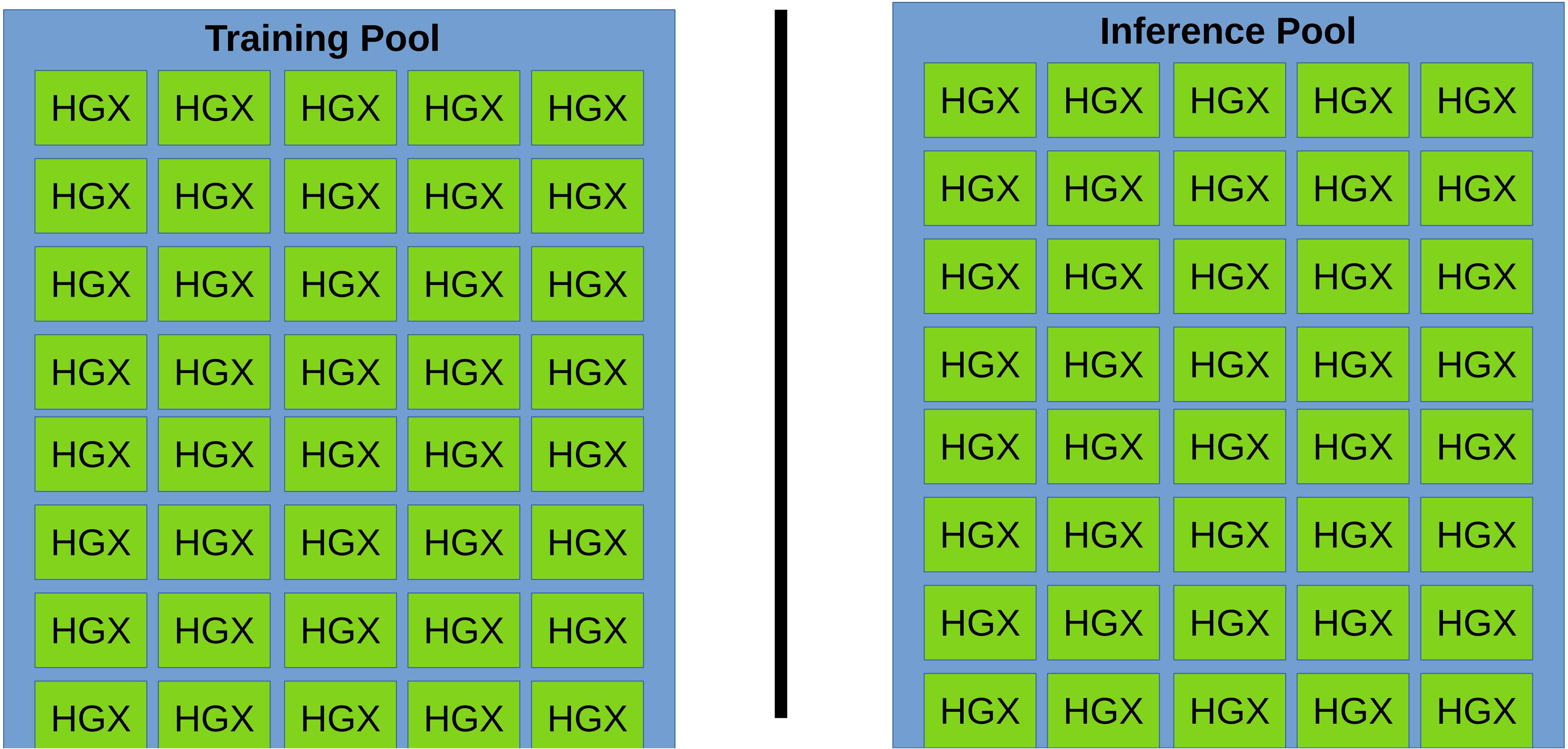
Example:
GPU Workload
Live Migration

Problem at Cohere

- **“One thousand GPU servers”** (8x GPUs) running in hybrid training and inference cluster. (fictitious example)
- **GPU nodes allocated statically** to training and inference to serve **top 1%** of customer API traffic & to **meet customer SLAs** for **real-time requests**.
- **Millions of dollars wasted monthly** on cloud GPUs measurable in wasted cycles.


RDMA East/West Leaf
NIC
GPU
shmem
L2
L1
LD
ST
DPU
RDMA North/South Leaf
This is one 8x HGX GPU appliance.

Training and Inference



Root Cause

- **Slow auto-scaling.**
- S3 and **TCP/IP** in the data path for downloading inference models.
- NVMe to CPU to GPU **bounce buffer** used for staging data to GPU once downloaded.

Solution at Cohere
Stolen From OS & Hypervisor World

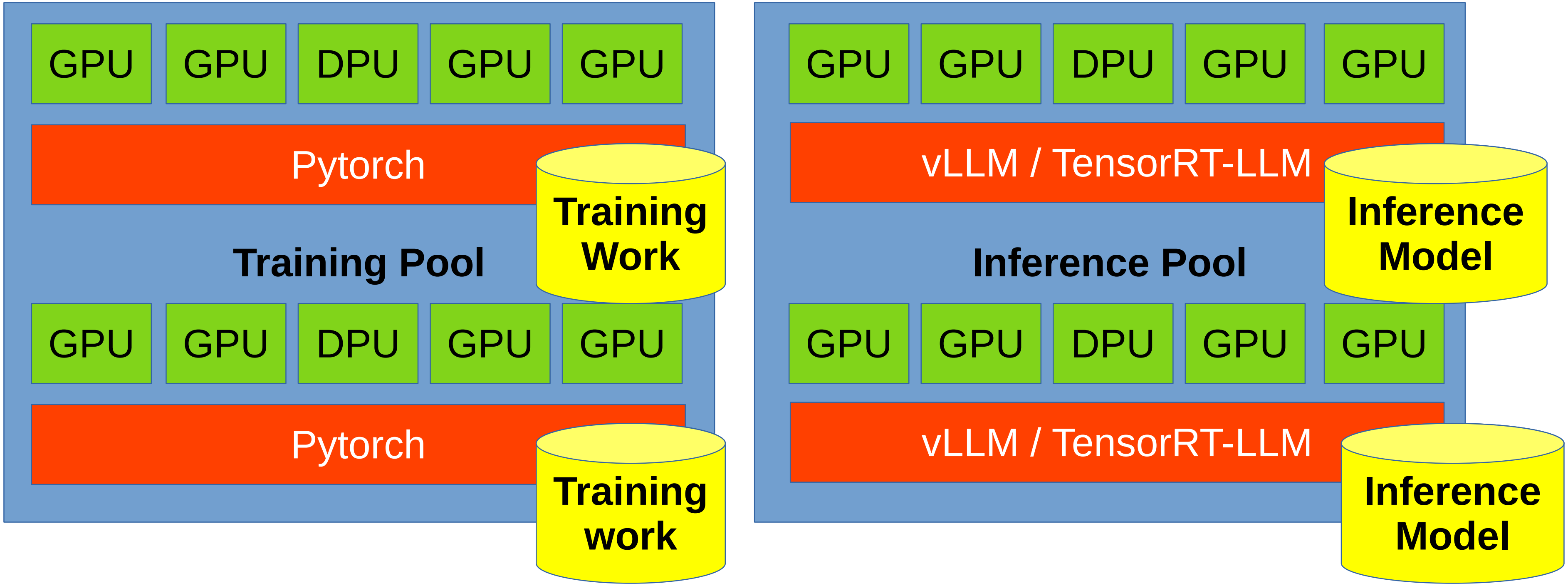
Use **RDMA** for GPU workload **live migration** to efficiently auto-scale training and inference.

Solution at Cohere
Stolen From OS & Hypervisor World

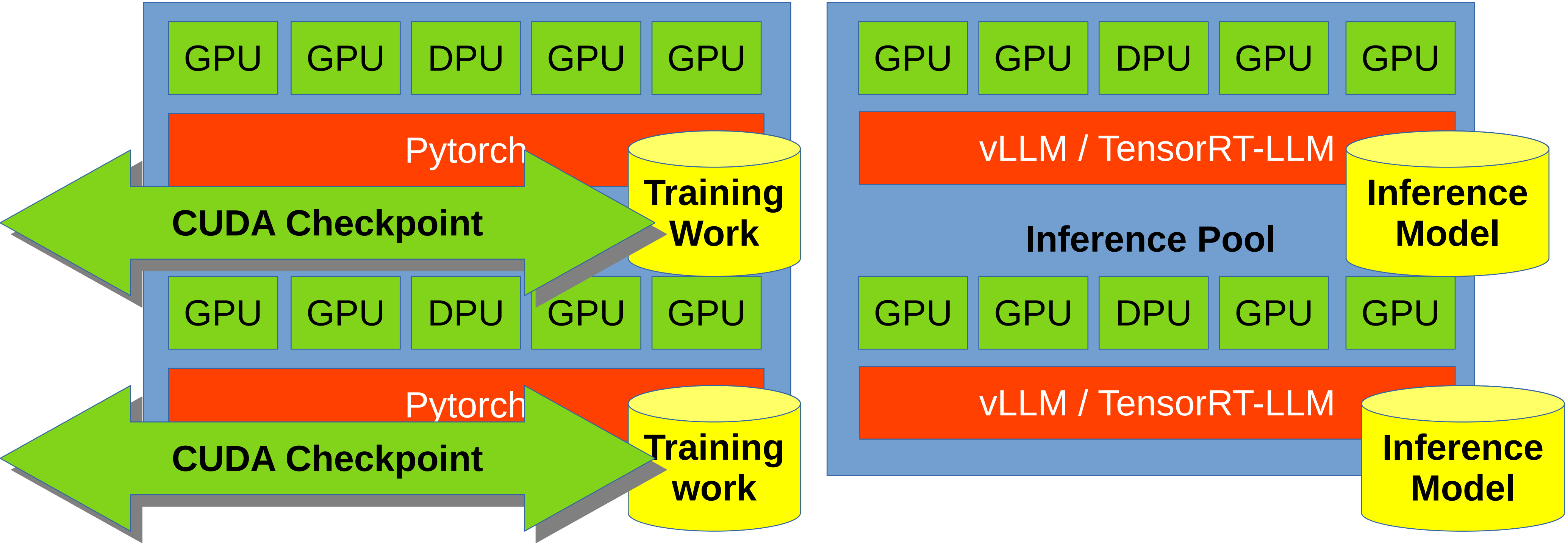
Use **RDMA** for GPU workload **live migration** to efficiently auto-scale training and inference.

Just-In-Time-Inference
Just-In-Time-Training

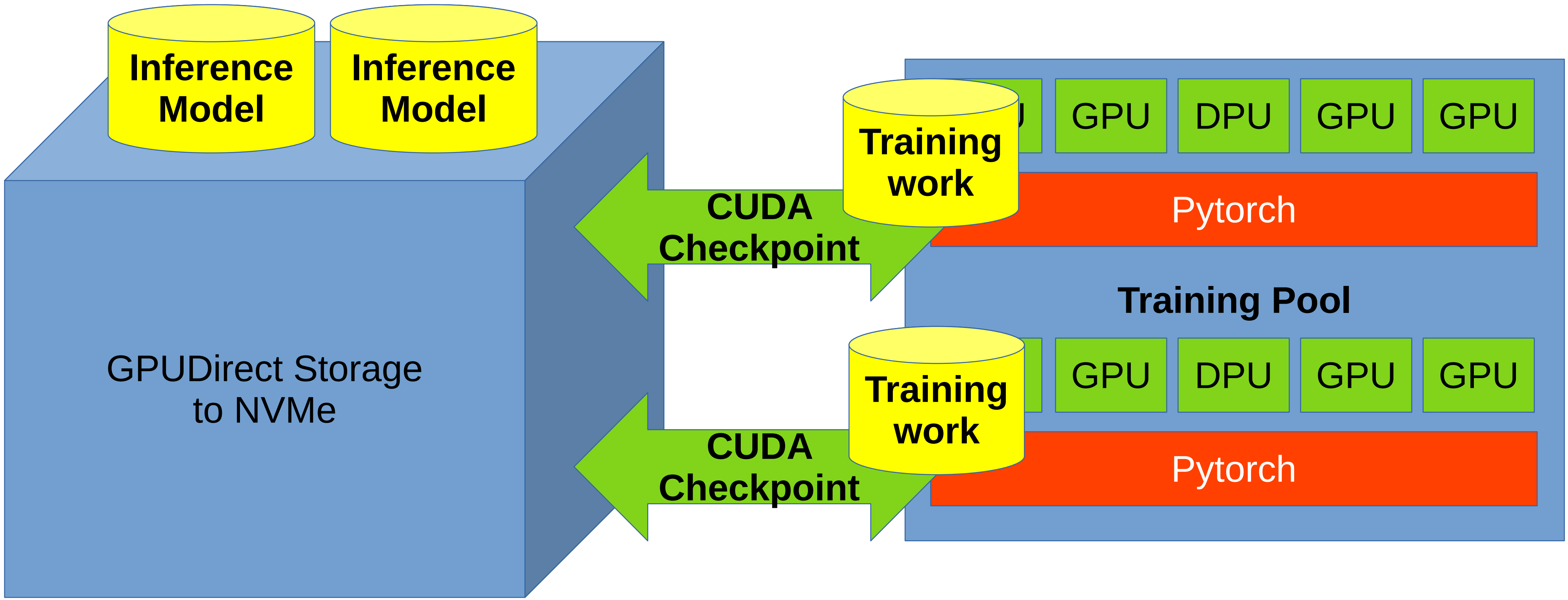
Just-in-Time-Inference (JITI)
Just-in-Time-Training (JITT)



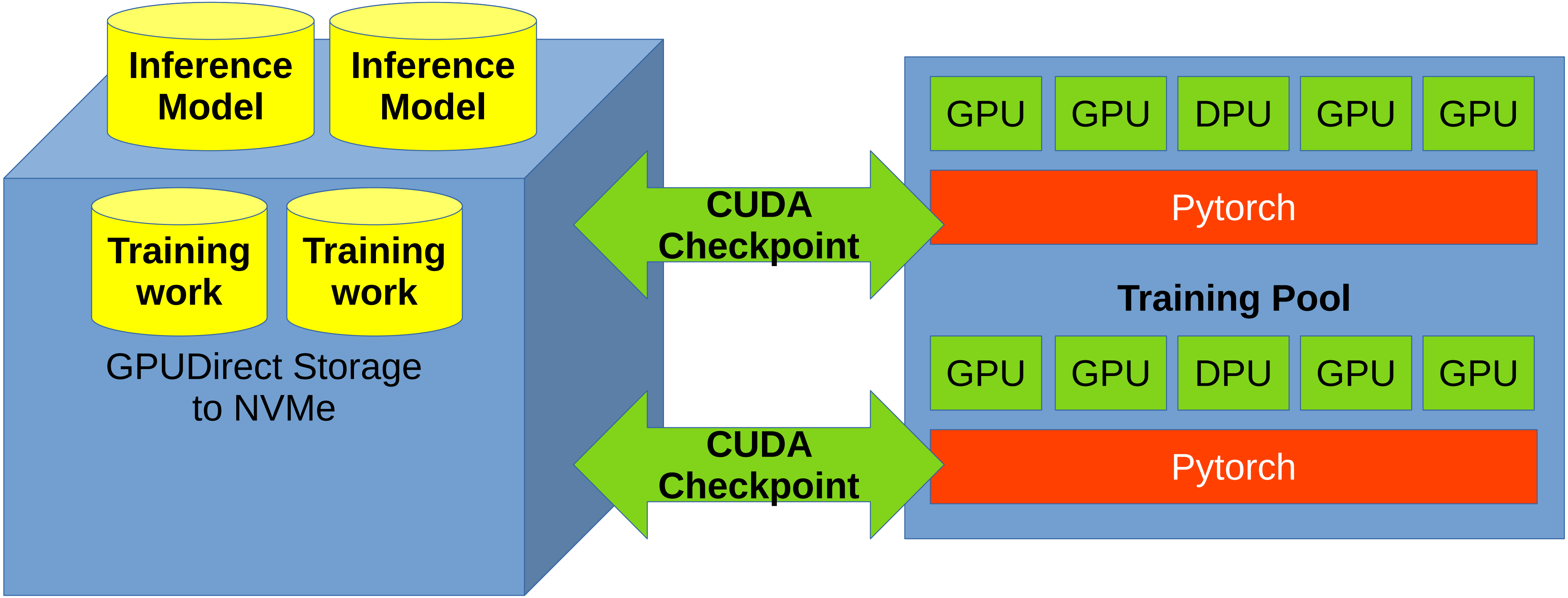
Just-in-Time-Inference (JITI)
Just-in-Time-Training (JITT)



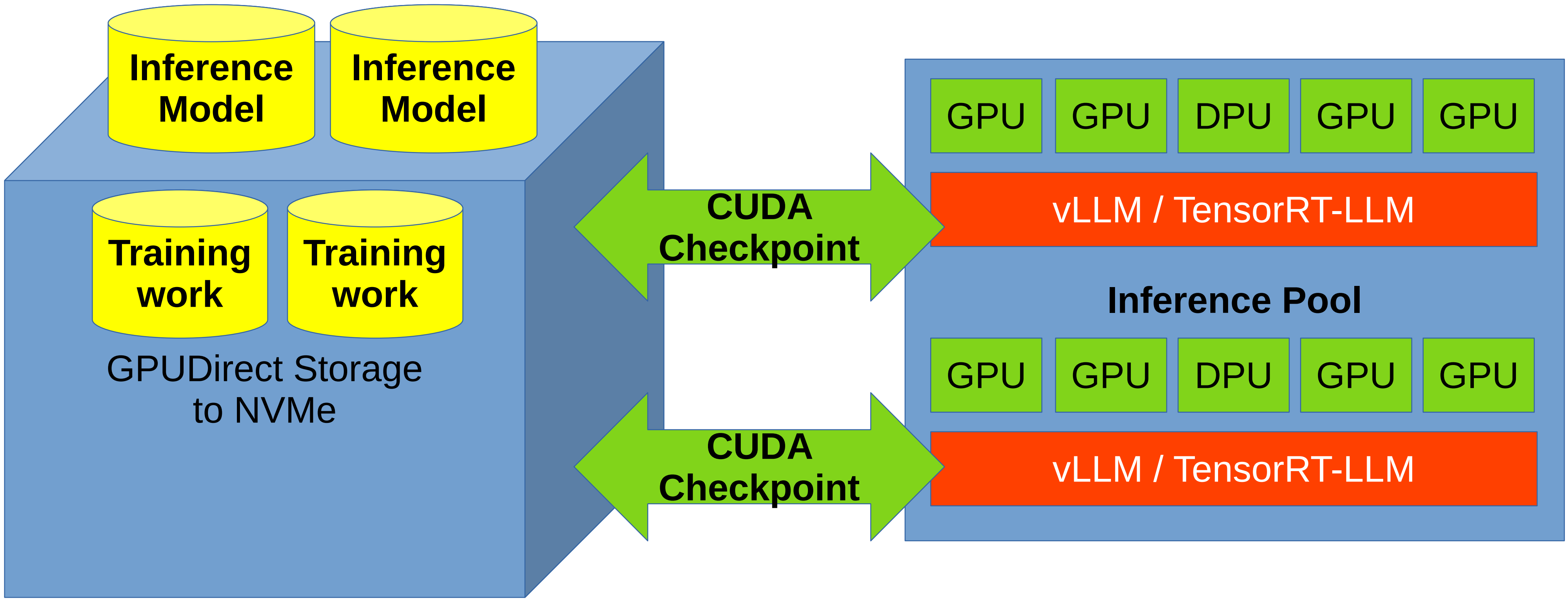
Just-in-Time-Inference (JITI)
Just-in-Time-Training (JITT)



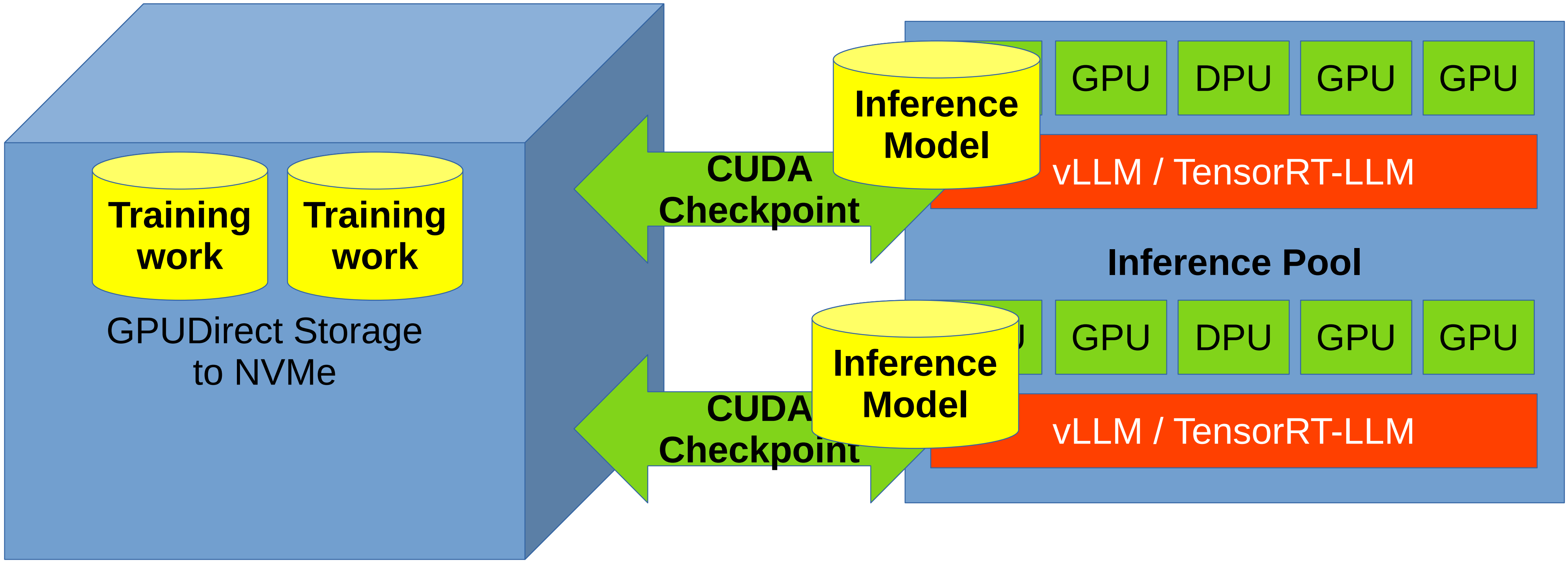
Just-in-Time-Inference (JITI)
Just-in-Time-Training (JITT)



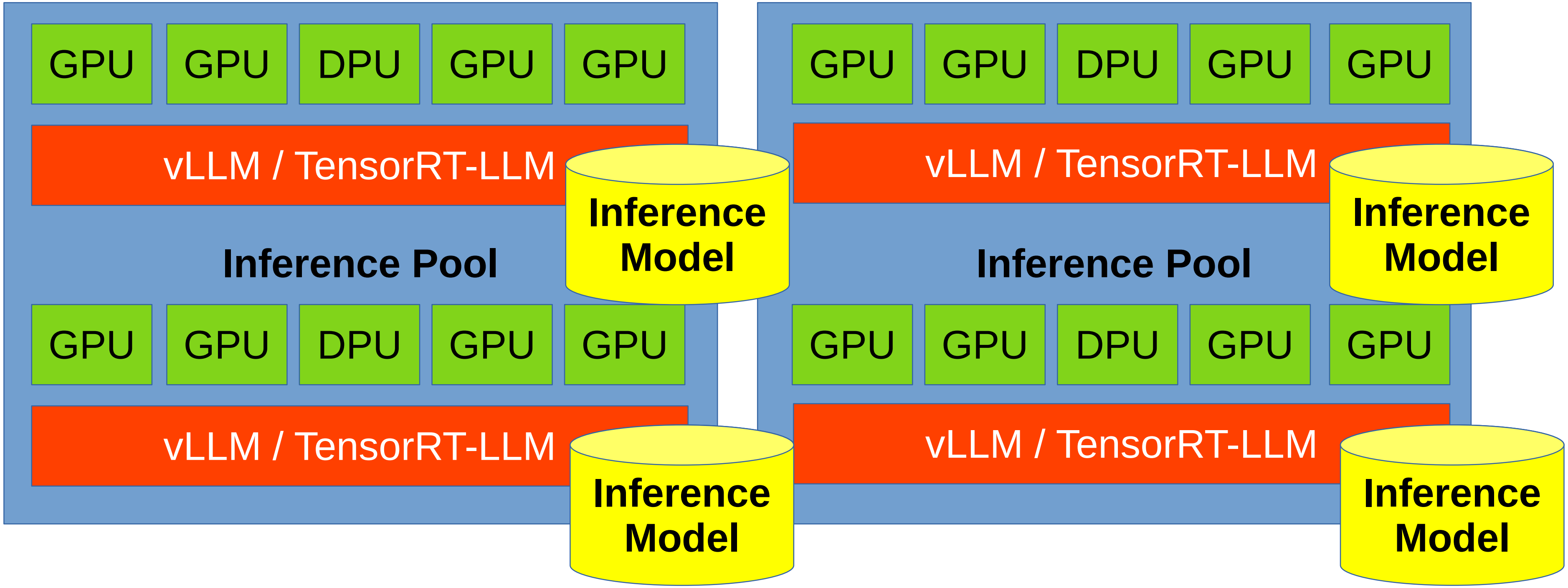
Just-in-Time-Inference (JITI)
Just-in-Time-Training (JITT)



Just-in-Time-Inference (JITI)
Just-in-Time-Training (JITT)



Just-in-Time-Inference (JITI)
Just-in-Time-Training (JITT)



Just-In-Time Auto-Scaling at Cohere

Before

10 Minutes

(S3 & TCP/IP Slow Data Path)

After

13 Seconds

(GPUDirect RDMA Live Migration)

Blog Post

Lessons Learned Scaling LLM Training and Inference Using Direct Memory Access (DMA)

Read more at **https://vgpu.io/**

2
How Transformers Work

Transformers, the basics.

- “Attention is All You Need” introduces **Transformer**.
- Transformers perform text prediction based on input sequences.
- Perform arbitrary information processing tasks with human instructions.

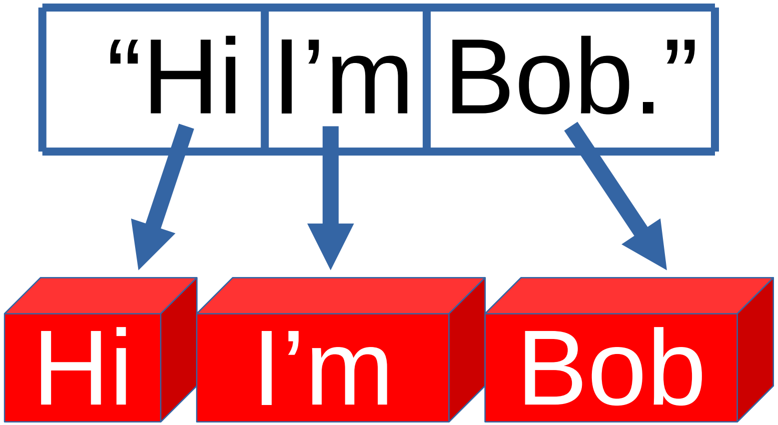
Token

- The **Token** is a **fundamental unit of measurement** in Transformers.
- A **token** is typically **3-4 characters** but can be less or more.
- A token is converted to a **vector, represented as a key and a value or K V.**

Tokens

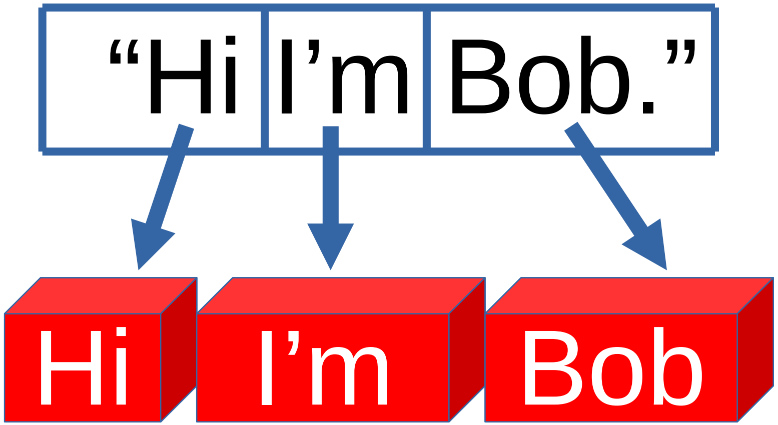
“Hi I’m Bob.”

Tokens



Keys: 0 1 2

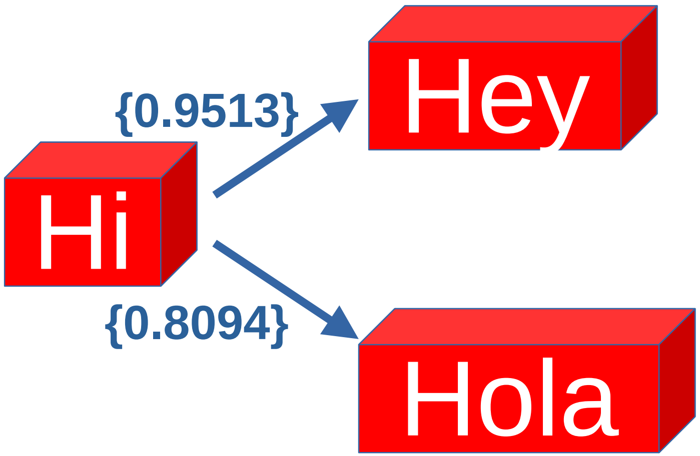
Tokens



Values:

$$\begin{pmatrix}0.4321\\0.3456\\0.5345\\0.3452\\0.9523\\0.6546\end{pmatrix} \quad \begin{pmatrix}0.6454\\0.1235\\0.6547\\0.3576\\0.3247\\0.1237\end{pmatrix} \quad \begin{pmatrix}0.6435\\0.1235\\0.7455\\0.1325\\0.1235\\0.6457\end{pmatrix}$$

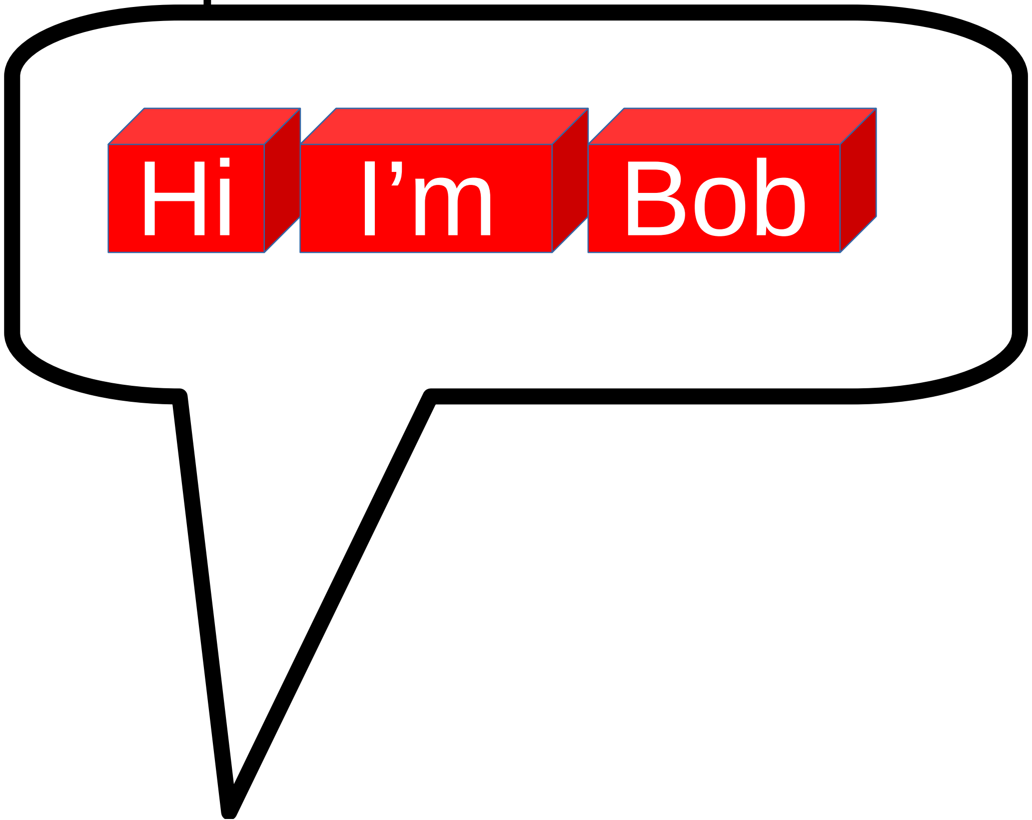
Word2Vec



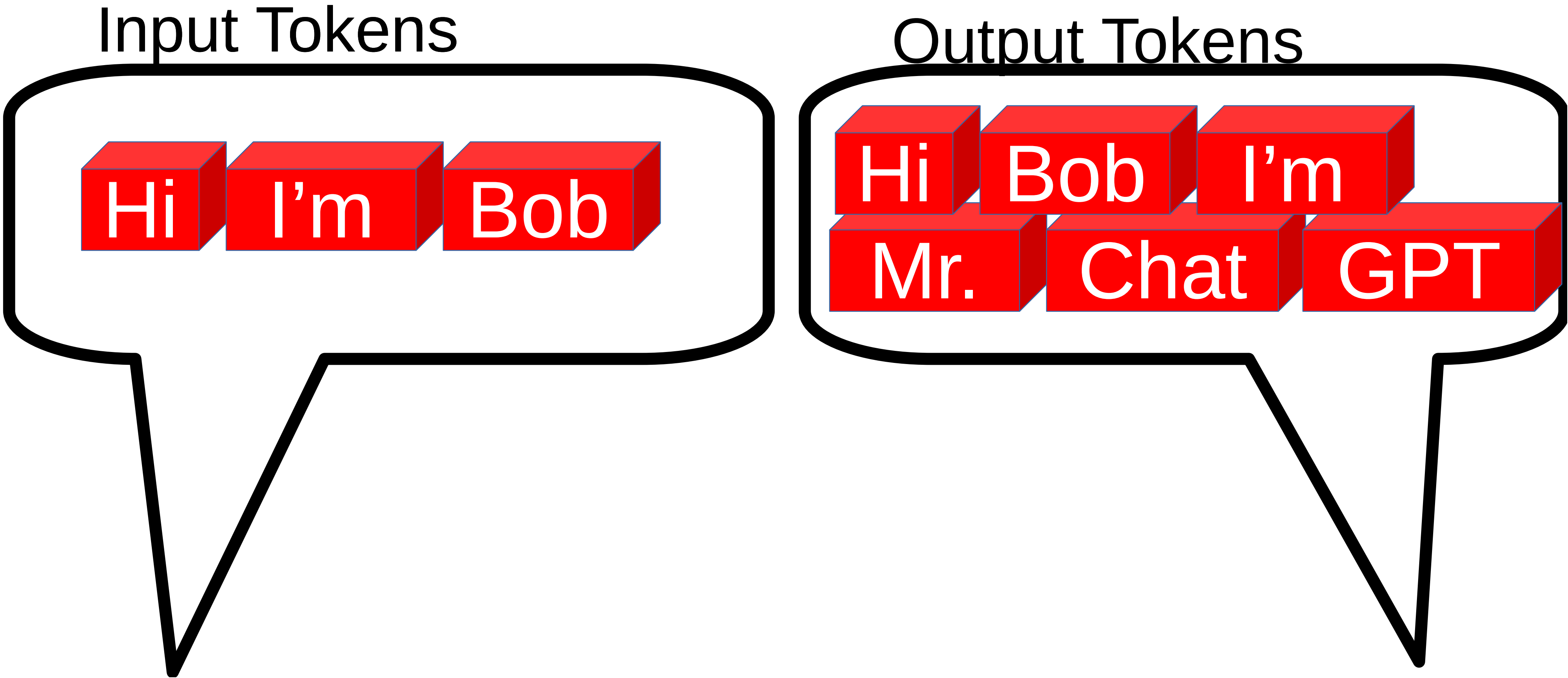
“Stregth of connection to frequently neighboring tokens.”

Input and Output Tokens

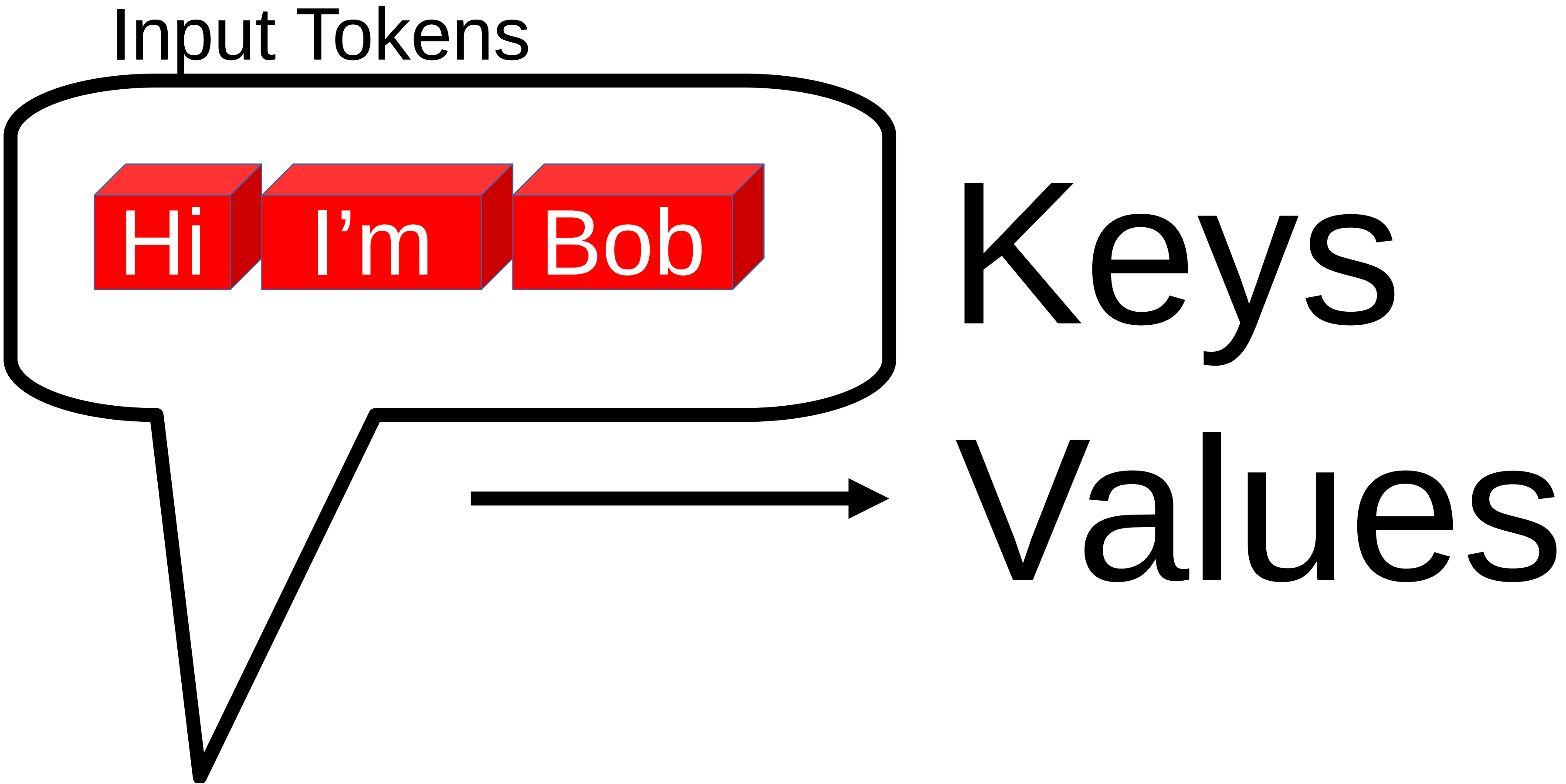
Input Tokens



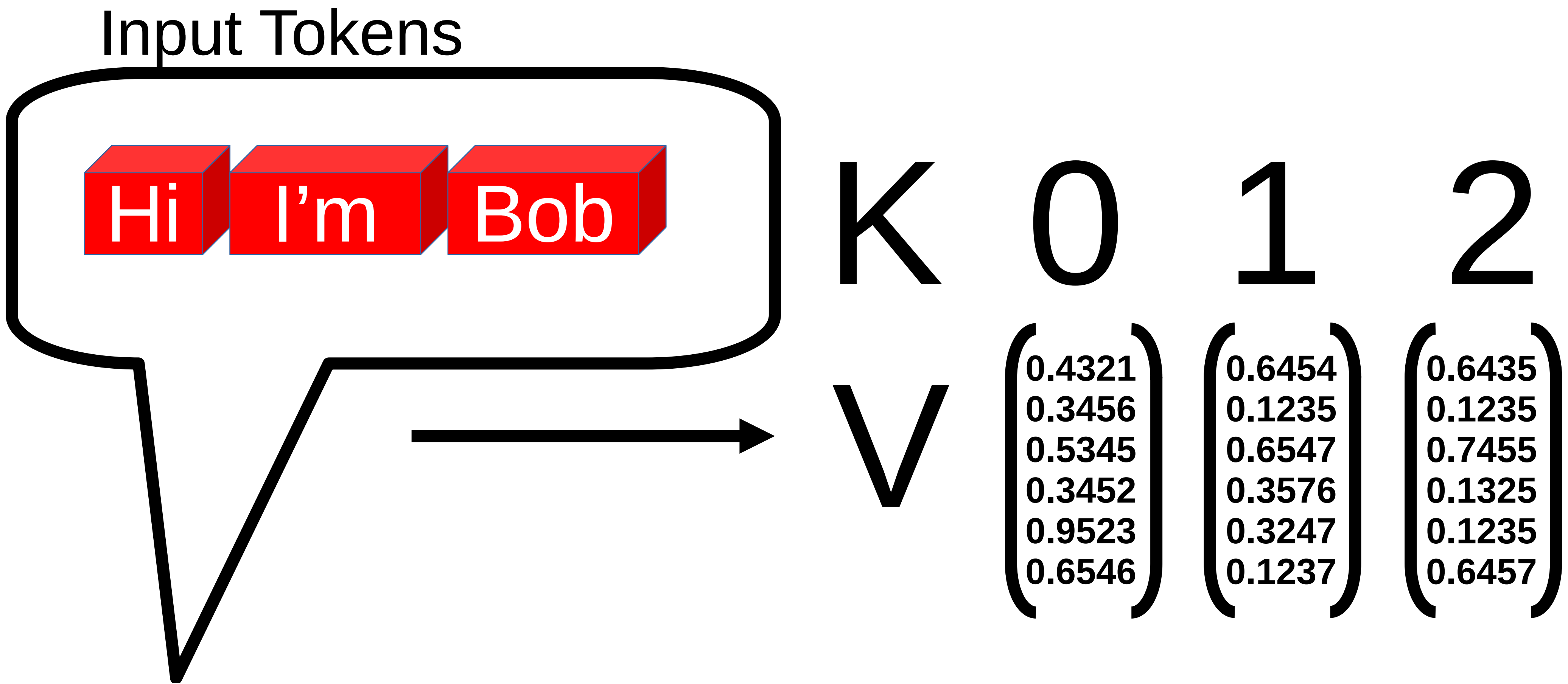
Input and Output Tokens



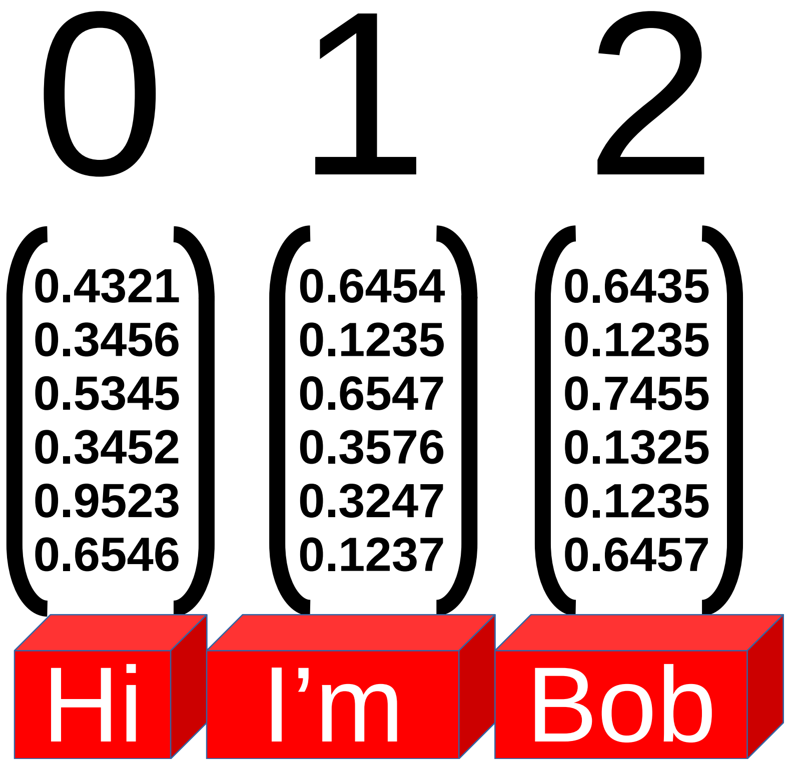
Pre-Fill, Cache Generation



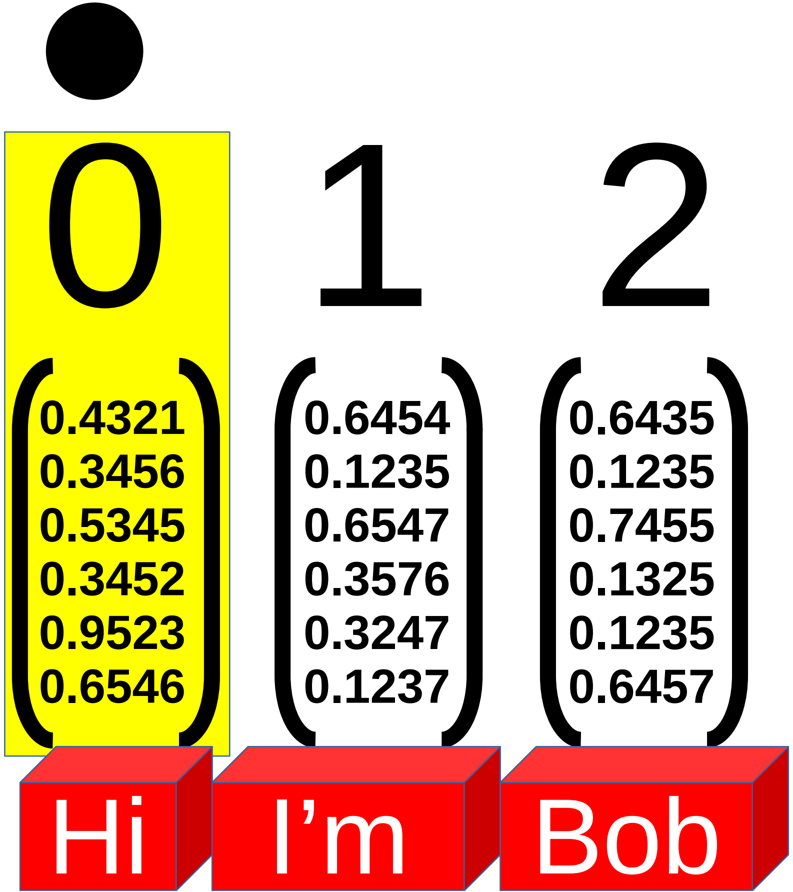
Pre-Fill, Cache Generation



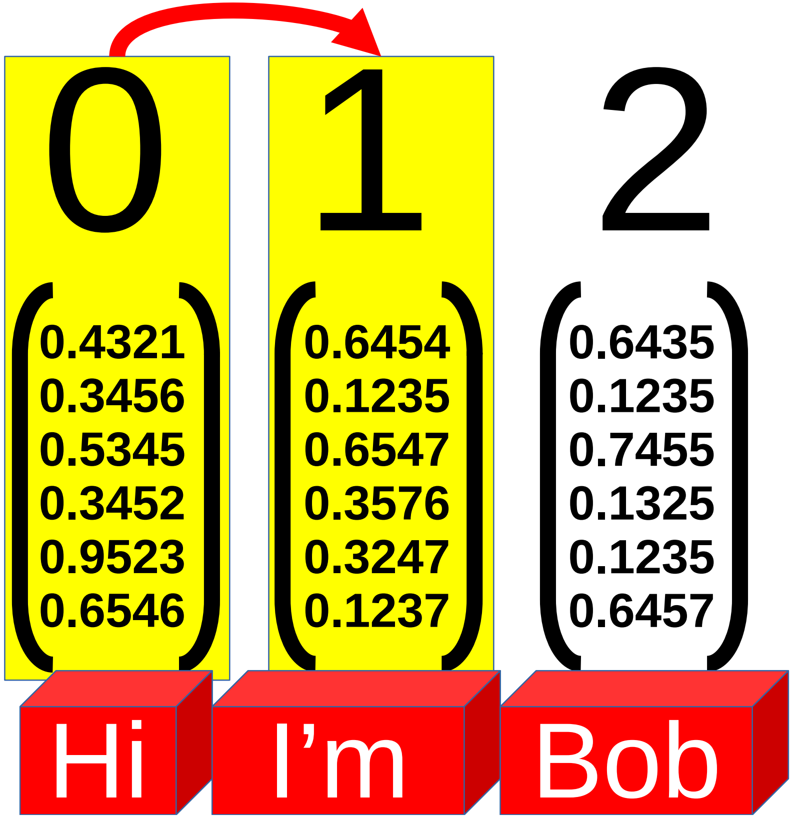
Auto-Regressive Token Prediction



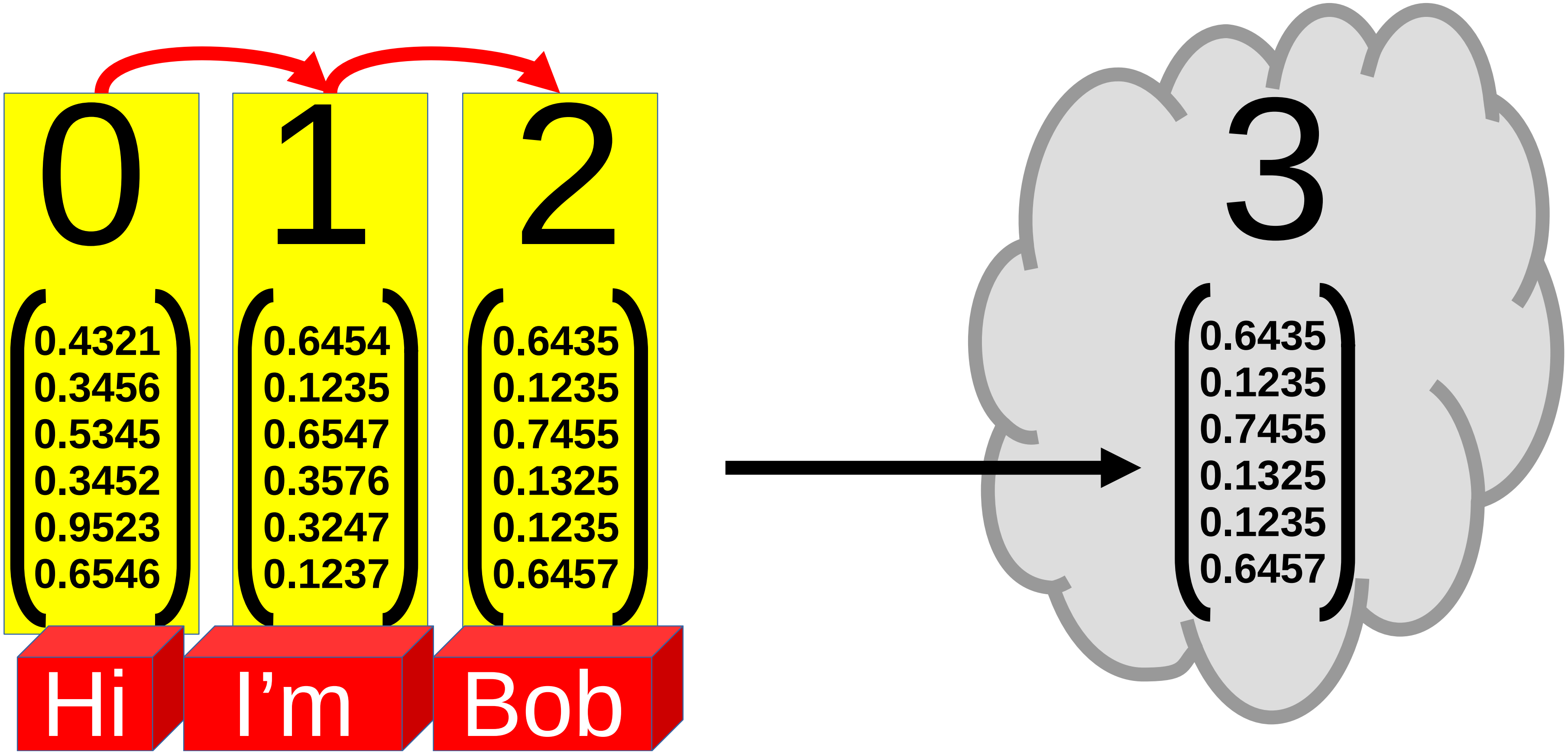
Auto-Regressive Token Prediction



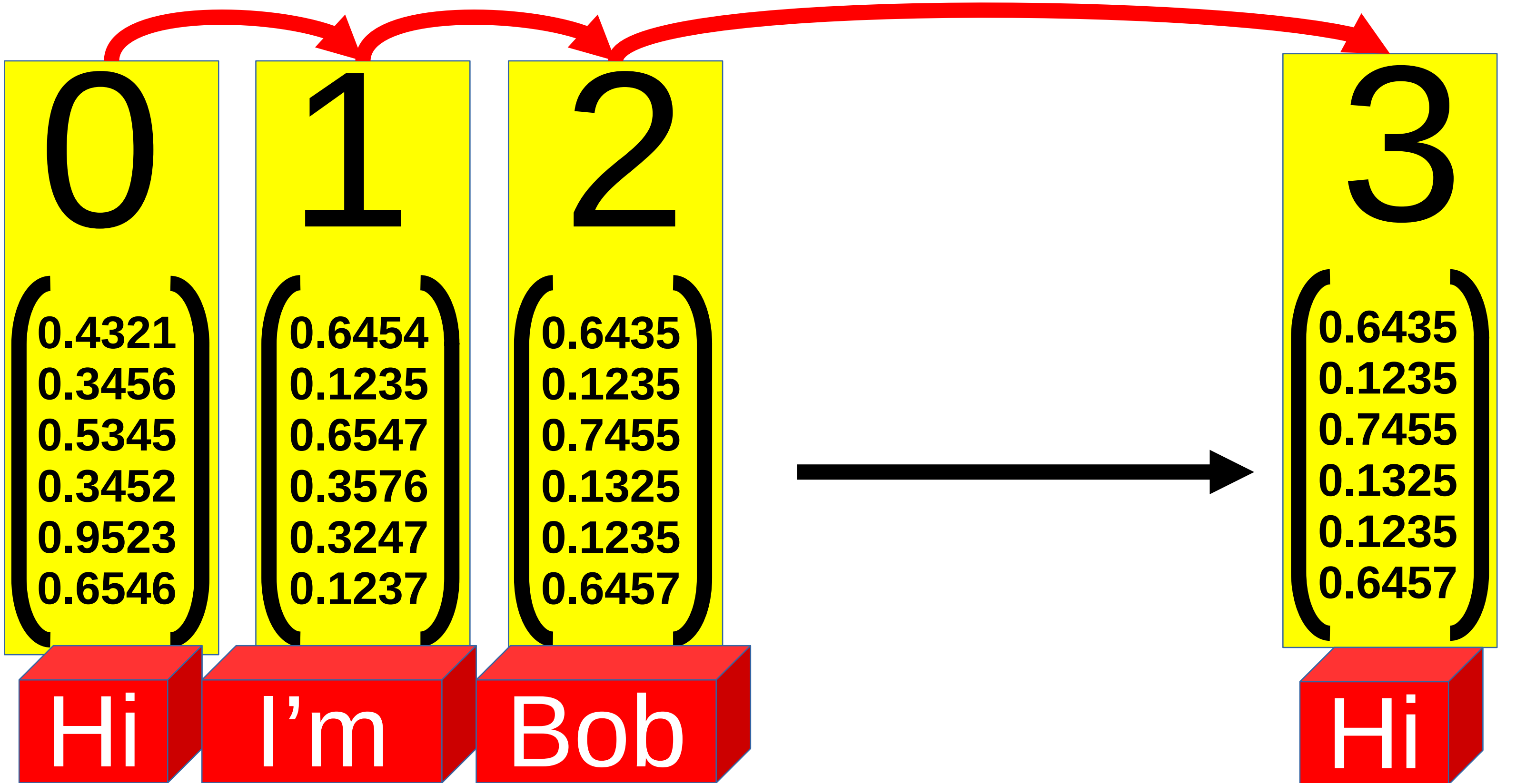
Auto-Regressive Token Prediction



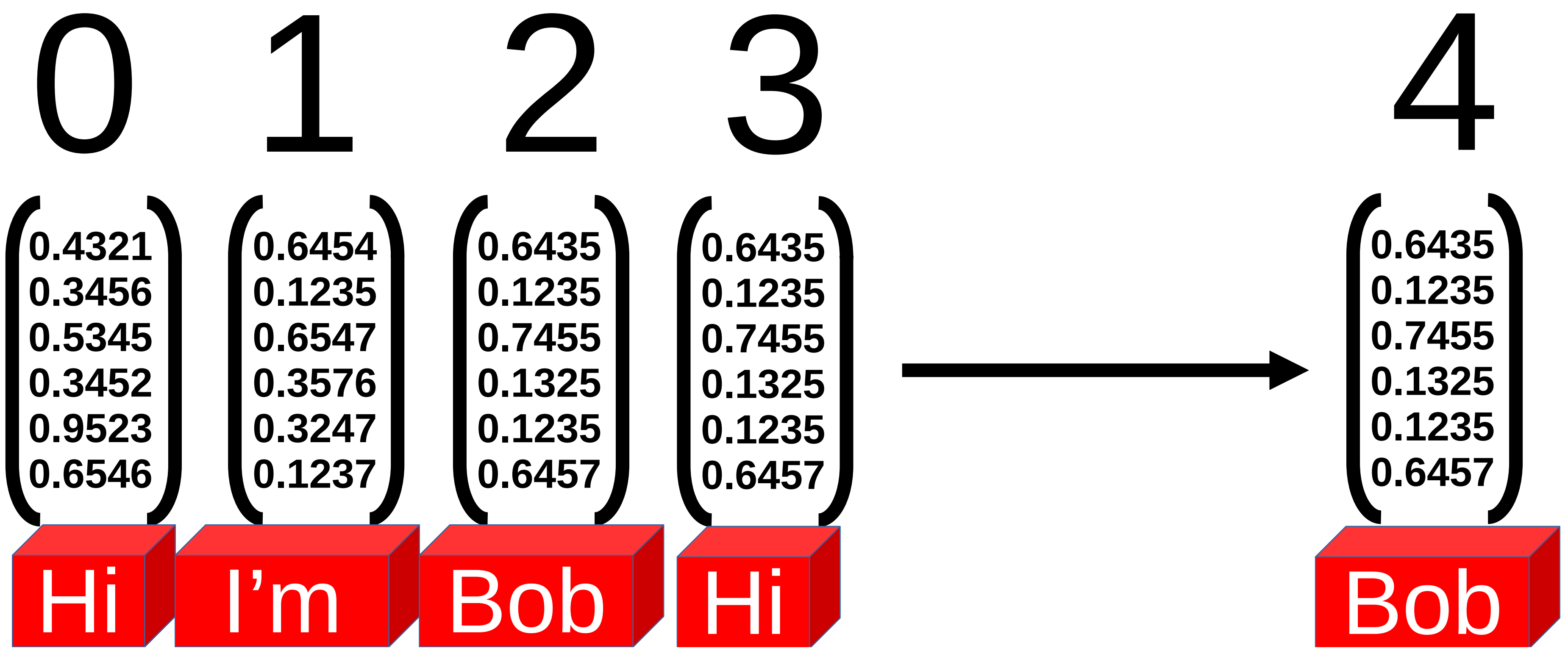
Auto-Regressive Token Prediction



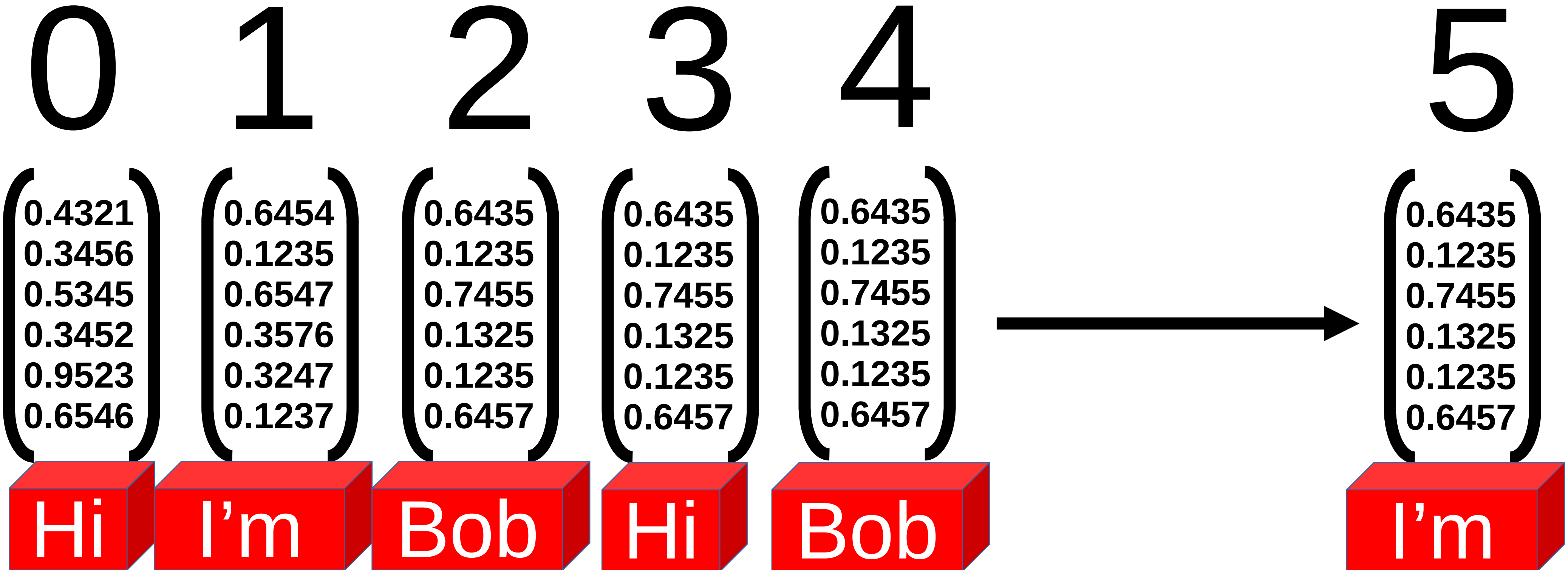
Auto-Regressive Token Prediction



Auto-Regressive Token Prediction



Auto-Regressive Token Prediction



$$\overset{3}{\begin{pmatrix} 0.4321 \\ 0.3456 \\ 0.5345 \\ 0.3452 \\ 0.9523 \\ 0.6546 \end{pmatrix}} \quad \overset{4}{\begin{pmatrix} 0.6454 \\ 0.1235 \\ 0.6547 \\ 0.3576 \\ 0.3247 \\ 0.1237 \end{pmatrix}} \quad \overset{5}{\begin{pmatrix} 0.6435 \\ 0.1235 \\ 0.7455 \\ 0.1325 \\ 0.1235 \\ 0.6457 \end{pmatrix}}$$

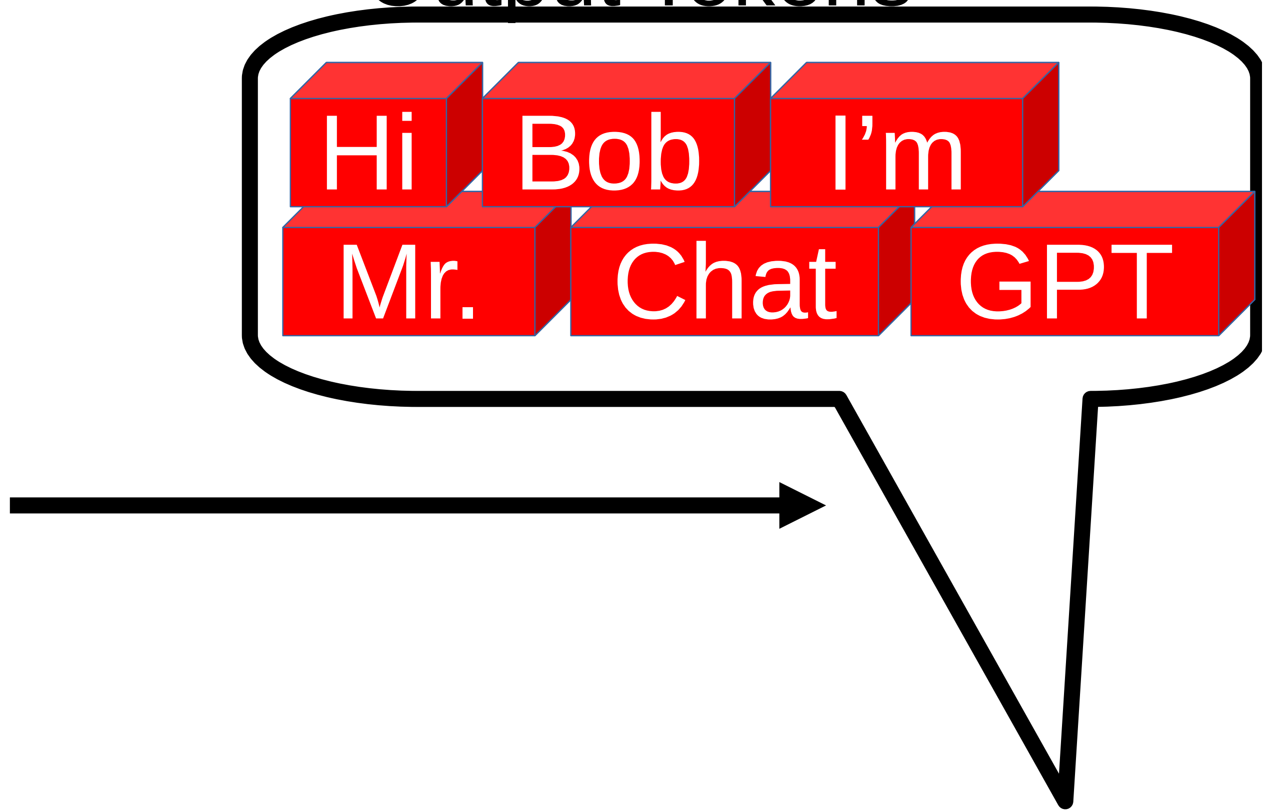
$$\overset{6}{\begin{pmatrix} 0.4321 \\ 0.3456 \\ 0.5345 \\ 0.3452 \\ 0.9523 \\ 0.6546 \end{pmatrix}} \quad \overset{7}{\begin{pmatrix} 0.6454 \\ 0.1235 \\ 0.6547 \\ 0.3576 \\ 0.3247 \\ 0.1237 \end{pmatrix}} \quad \overset{8}{\begin{pmatrix} 0.6435 \\ 0.1235 \\ 0.7455 \\ 0.1325 \\ 0.1235 \\ 0.6457 \end{pmatrix}}$$

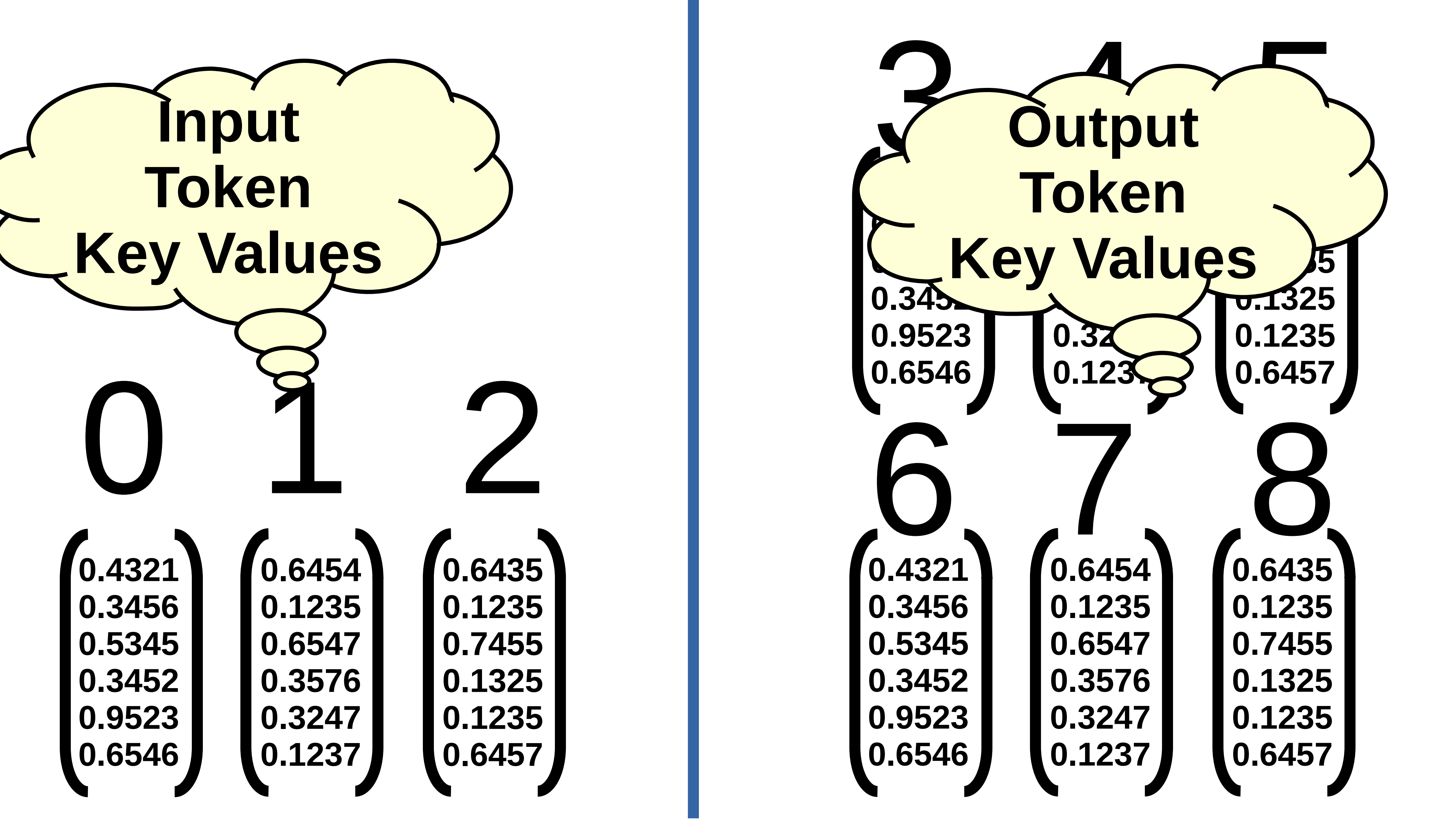
Predicted Tokens

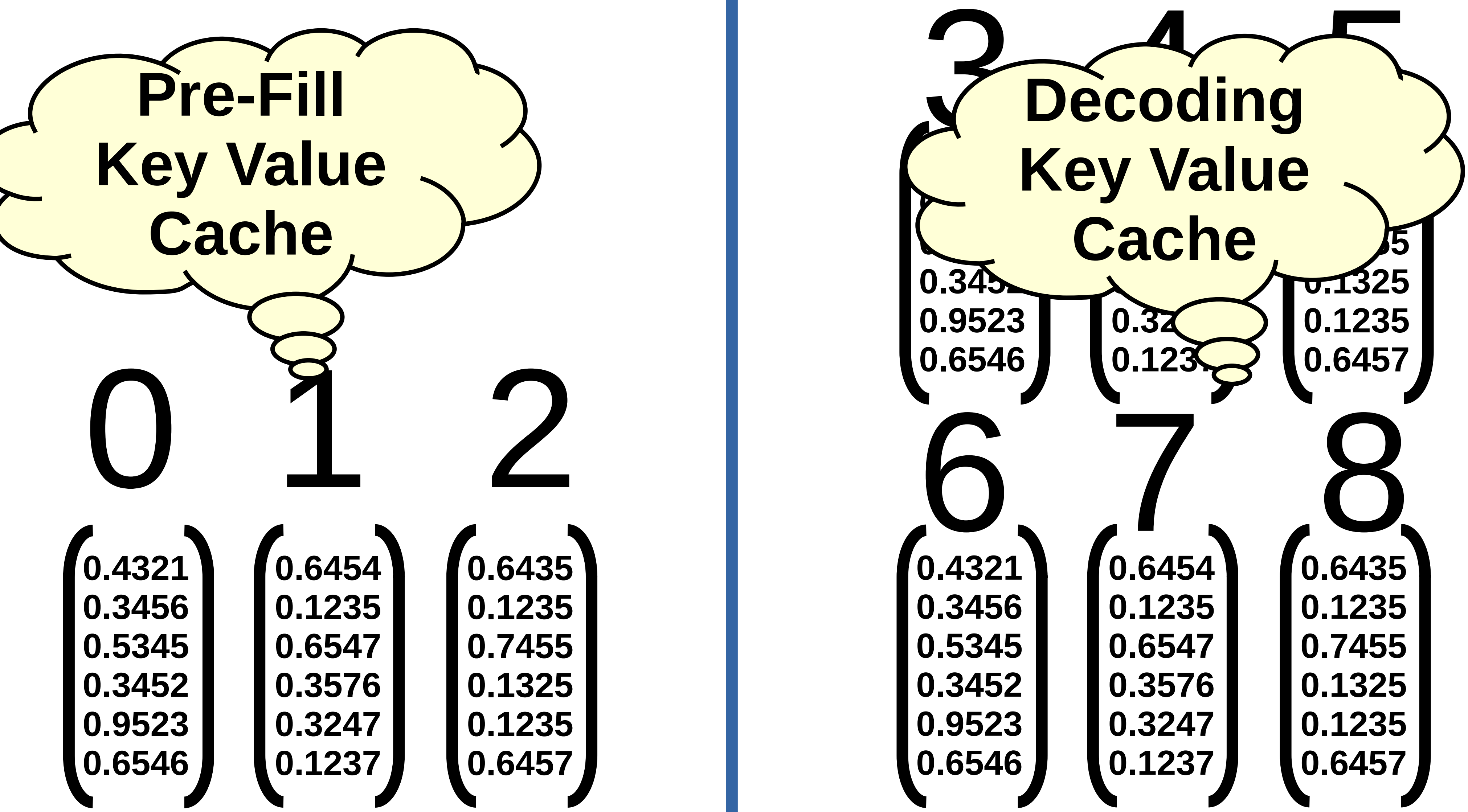
Decoding
Convert KV to Plain Text

$$\begin{array}{ccc} 3 & 4 & 5 \\ \begin{pmatrix} 0.4321 \\ 0.3456 \\ 0.5345 \\ 0.3452 \\ 0.9523 \\ 0.6546 \end{pmatrix} & \begin{pmatrix} 0.6454 \\ 0.1235 \\ 0.6547 \\ 0.3576 \\ 0.3247 \\ 0.1237 \end{pmatrix} & \begin{pmatrix} 0.6435 \\ 0.1235 \\ 0.7455 \\ 0.1325 \\ 0.1235 \\ 0.6457 \end{pmatrix} \\ 6 & 7 & 8 \\ \begin{pmatrix} 0.4321 \\ 0.3456 \\ 0.5345 \\ 0.3452 \\ 0.9523 \\ 0.6546 \end{pmatrix} & \begin{pmatrix} 0.6454 \\ 0.1235 \\ 0.6547 \\ 0.3576 \\ 0.3247 \\ 0.1237 \end{pmatrix} & \begin{pmatrix} 0.6435 \\ 0.1235 \\ 0.7455 \\ 0.1325 \\ 0.1235 \\ 0.6457 \end{pmatrix} \end{array}$$

Output Tokens




Input Token Key Values
0
1
2
0.4321 0.3456 0.5345 0.3452 0.9523 0.6546
0.6454 0.1235 0.6547 0.3576 0.3247 0.1237
0.6435 0.1235 0.7455 0.1325 0.1235 0.6457
3
4
5
Output Token Key Values
0.9523 0.6546
0.1325 0.1235 0.6457
6
7
8
0.4321 0.3456 0.5345 0.3452 0.9523 0.6546
0.6454 0.1235 0.6547 0.3576 0.3247 0.1237
0.6435 0.1235 0.7455 0.1325 0.1235 0.6457



Pre-Fill
Key Value
Cache
0
1
2
0.4321
0.3456
0.5345
0.3452
0.9523
0.6546
0.6454
0.1235
0.6547
0.3576
0.3247
0.1237
0.6435
0.1235
0.7455
0.1325
0.1235
0.6457
3
4
5
Decoding
Key Value
Cache
0.9523
0.6546
0.1235
0.6457
6
7
8
0.4321
0.3456
0.5345
0.3452
0.9523
0.6546
0.6454
0.1235
0.6547
0.3576
0.3247
0.1237
0.6435
0.1235
0.7455
0.1325
0.1235
0.6457

Model Parameter Count

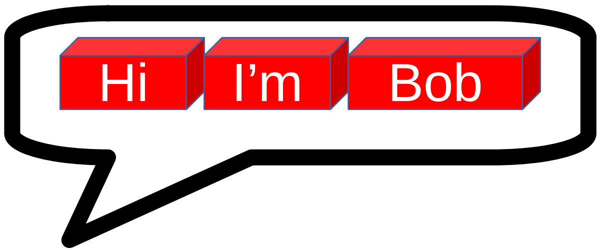
- **Each token's** number of **vector dimensions** varies based on the number of **Parameters in a Model**.
- The number of parameters in the model is determined by the amount of data the model has trained on.
- Llama-3.1-**405B** contains between **13,000** and **17,000 hidden dimensions**.

Terms to Understand

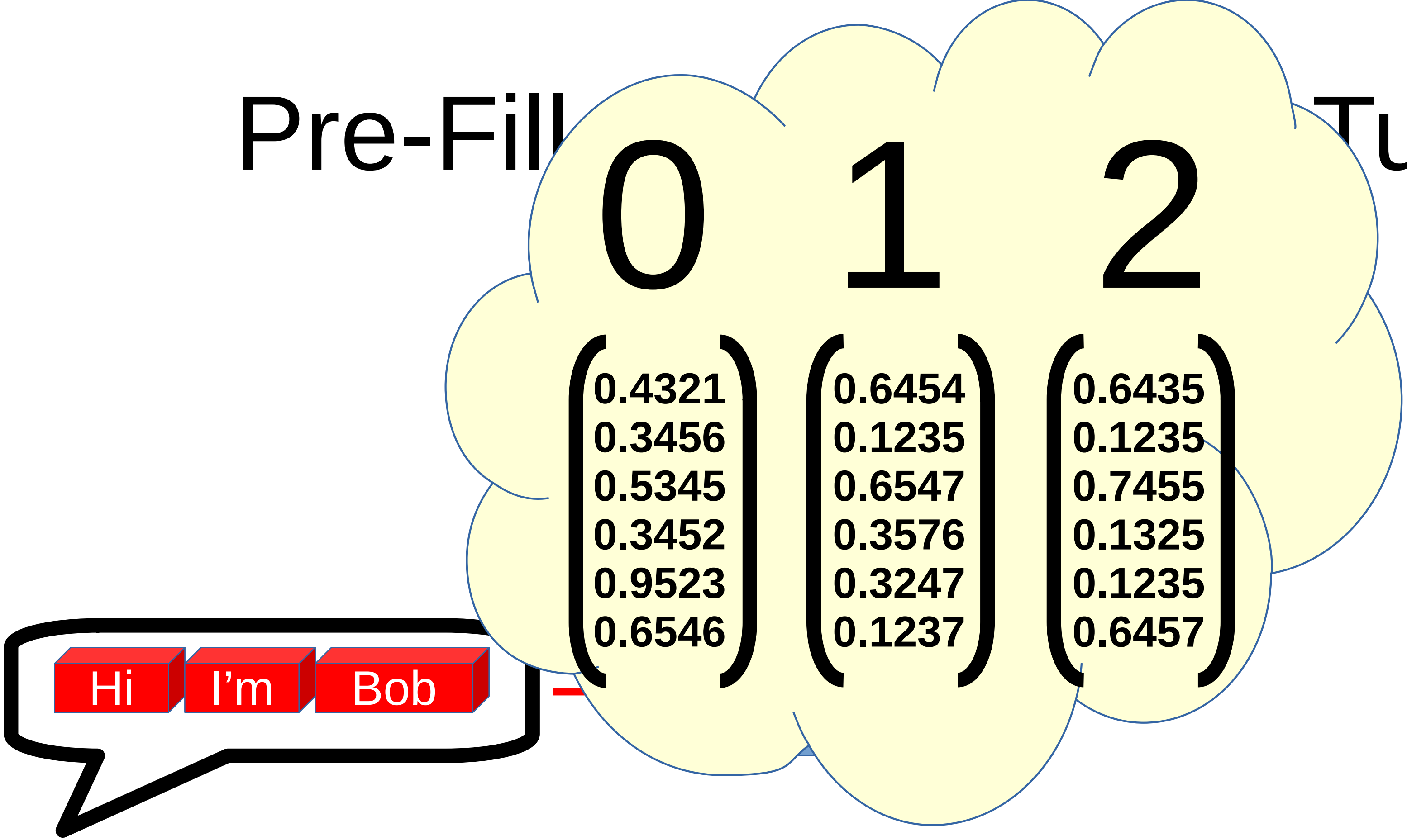
- Multi-Round QA:
  ”**Multiple questions** and **answers** in an
  AI inference conversation.”
- Turn 1:
  ”Your first QA round.”
- Turn 2:
  ”Your second QA round.”

The Full Picture

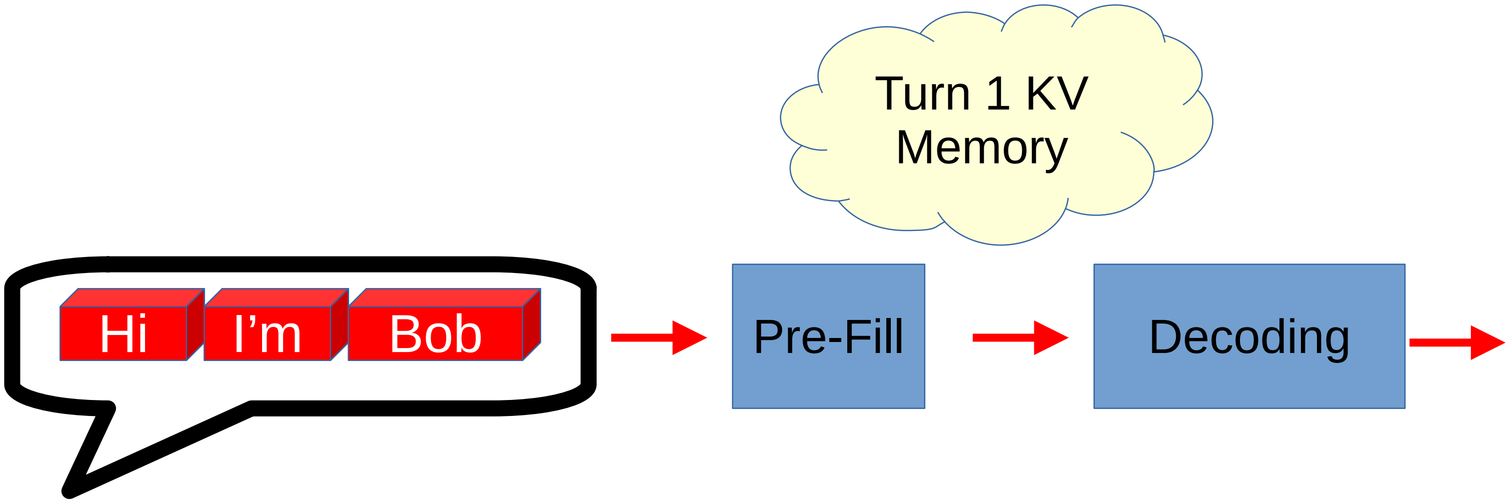
Pre-Fill and Decoding: Turn 1



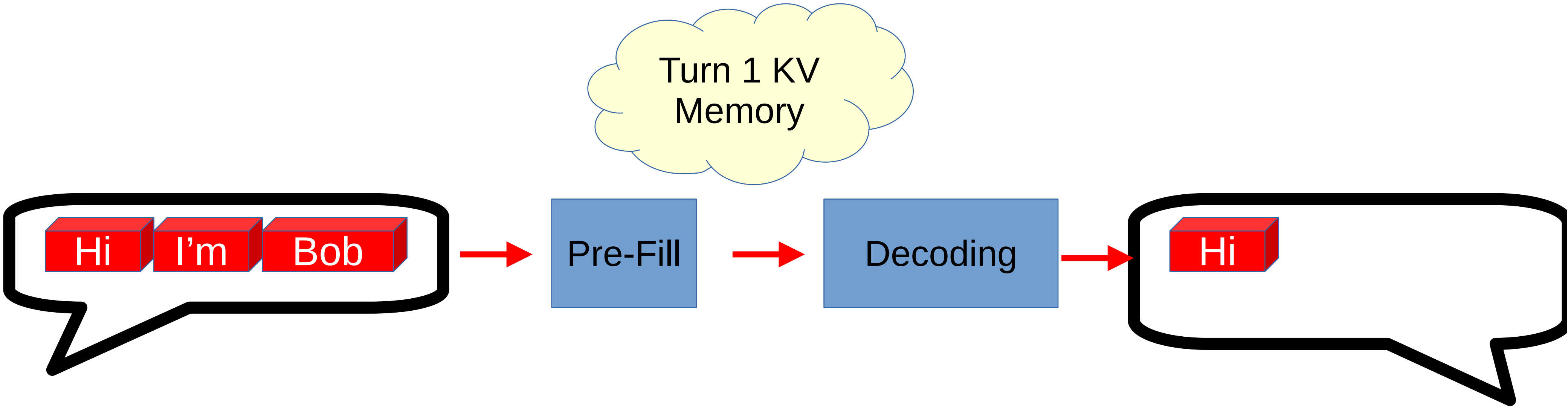
Pre-Fill Turn 1



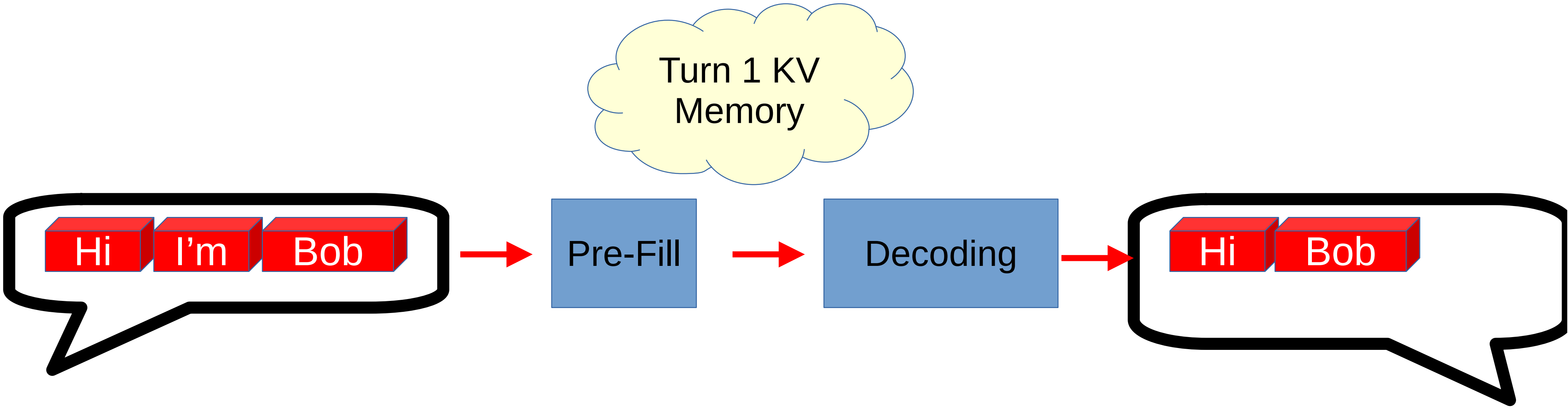
Pre-Fill and Decoding: Turn 1



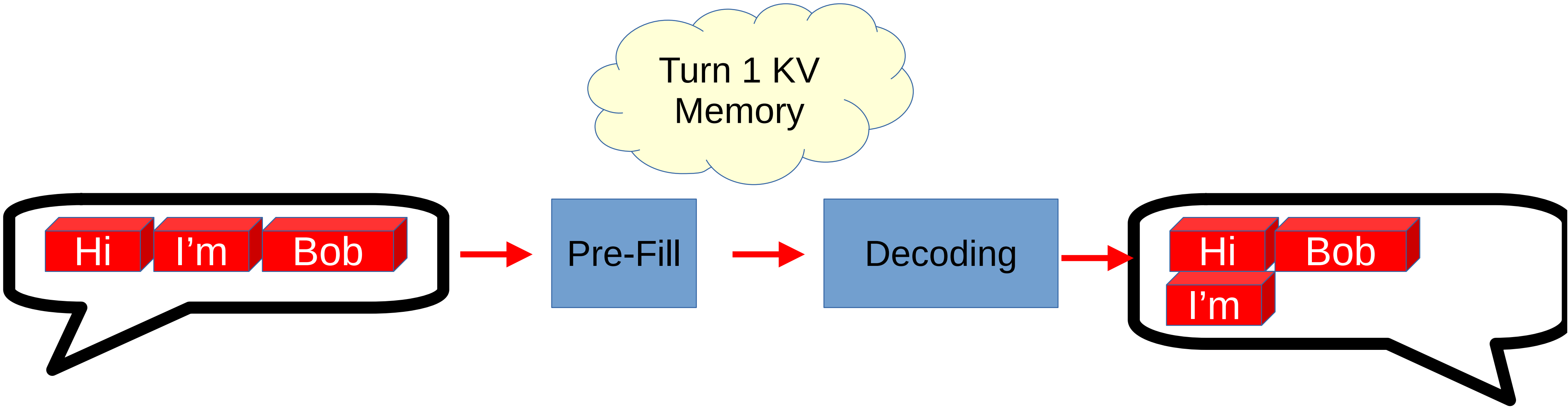
Pre-Fill and Decoding: Turn 1



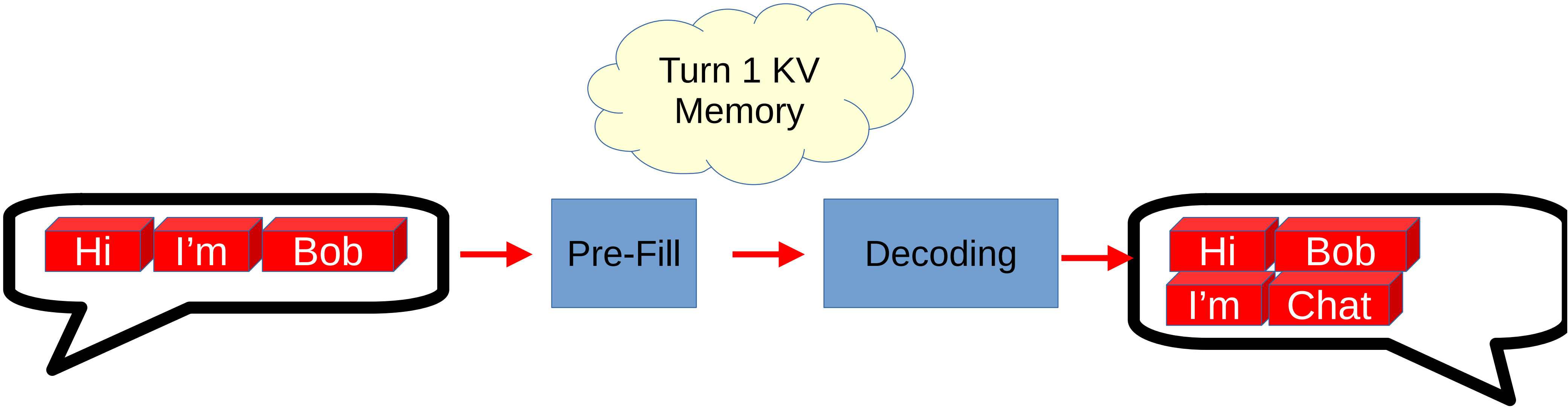
Pre-Fill and Decoding: Turn 1



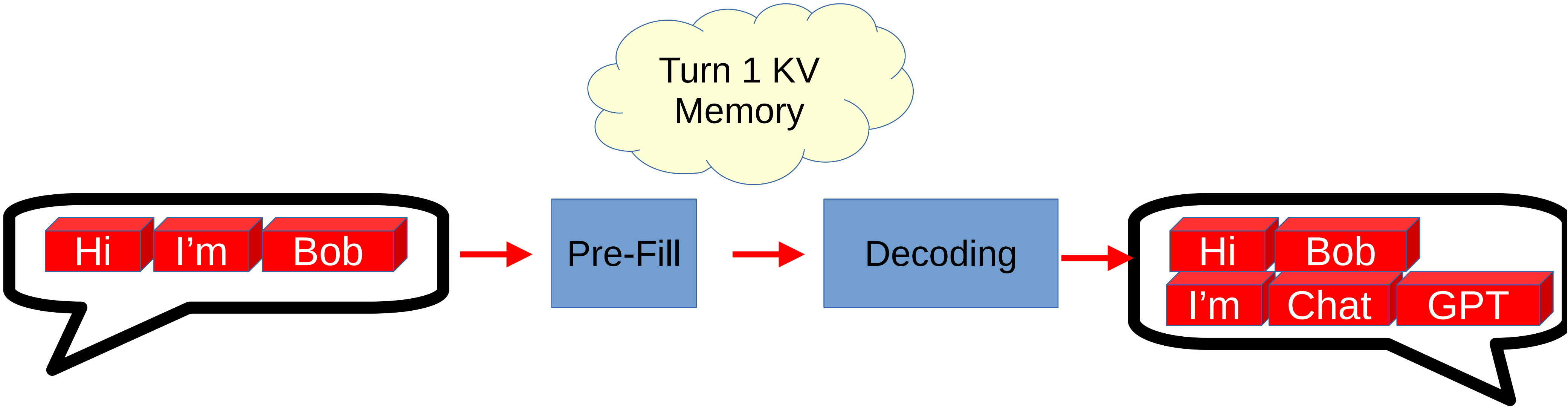
Pre-Fill and Decoding: Turn 1



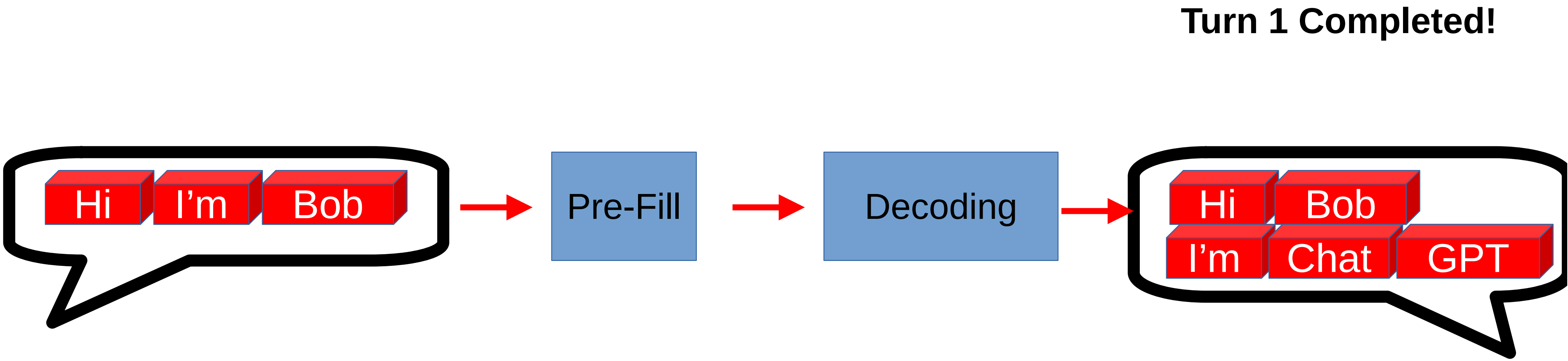
Pre-Fill and Decoding: Turn 1



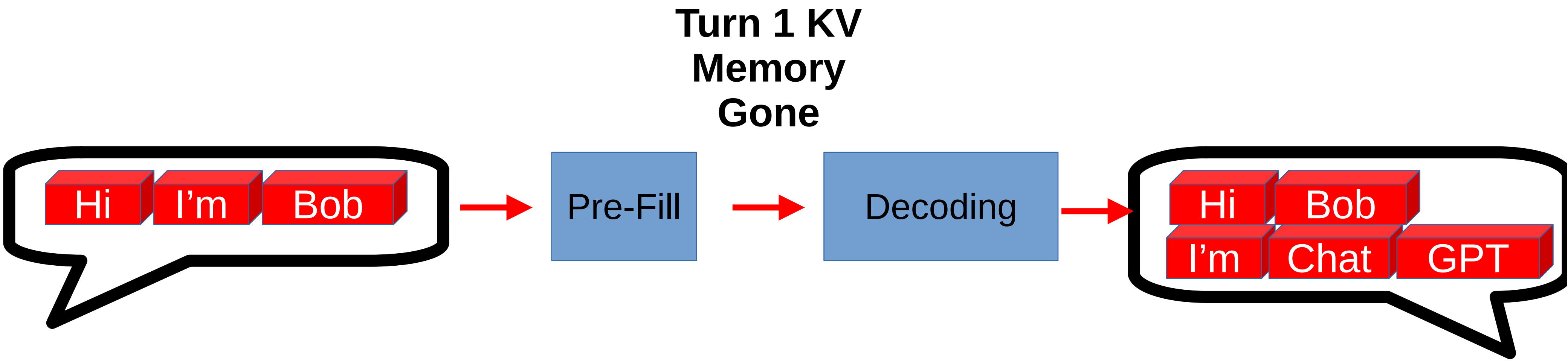
Pre-Fill and Decoding: Turn 1



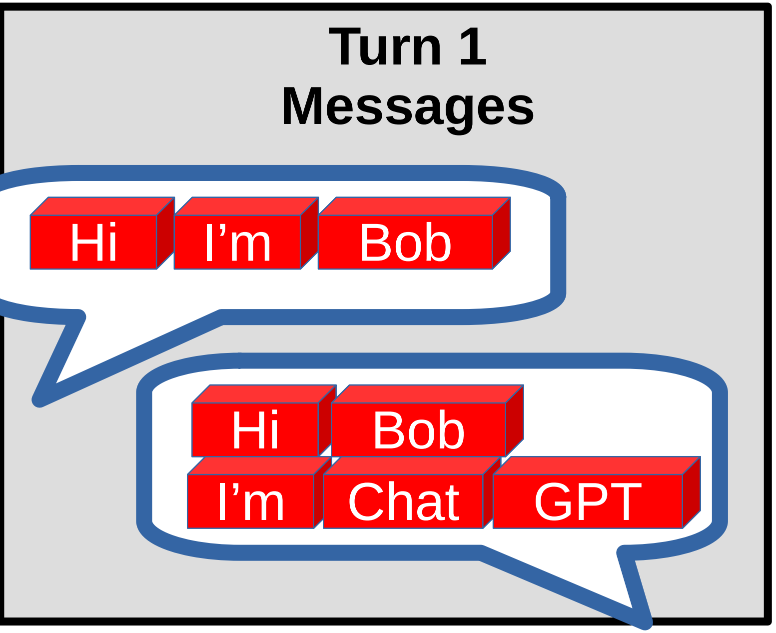
Pre-Fill and Decoding: Turn 1



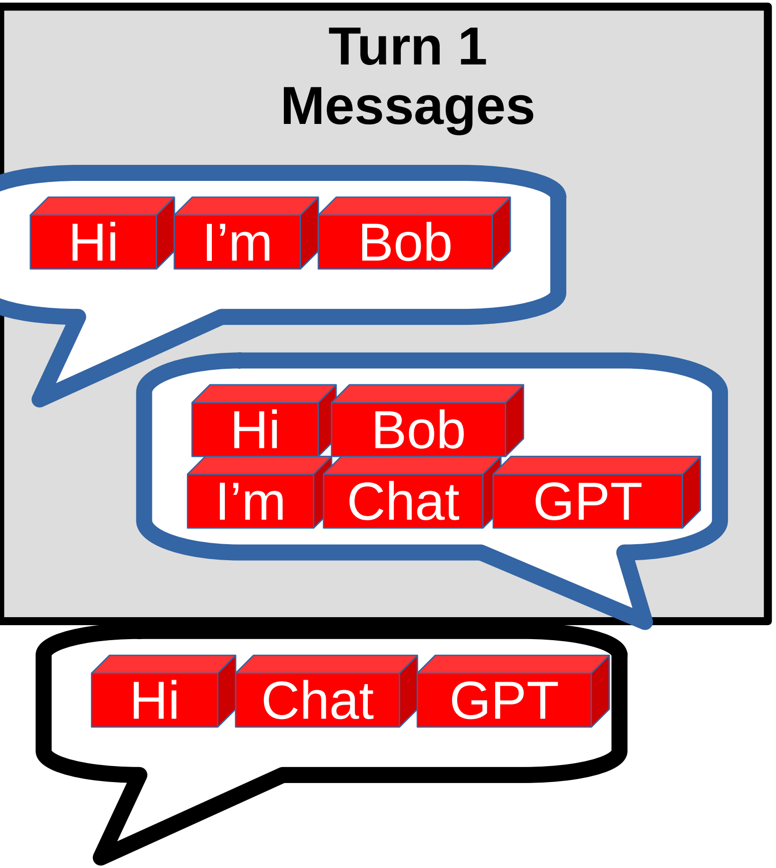
Pre-Fill and Decoding: Turn 1



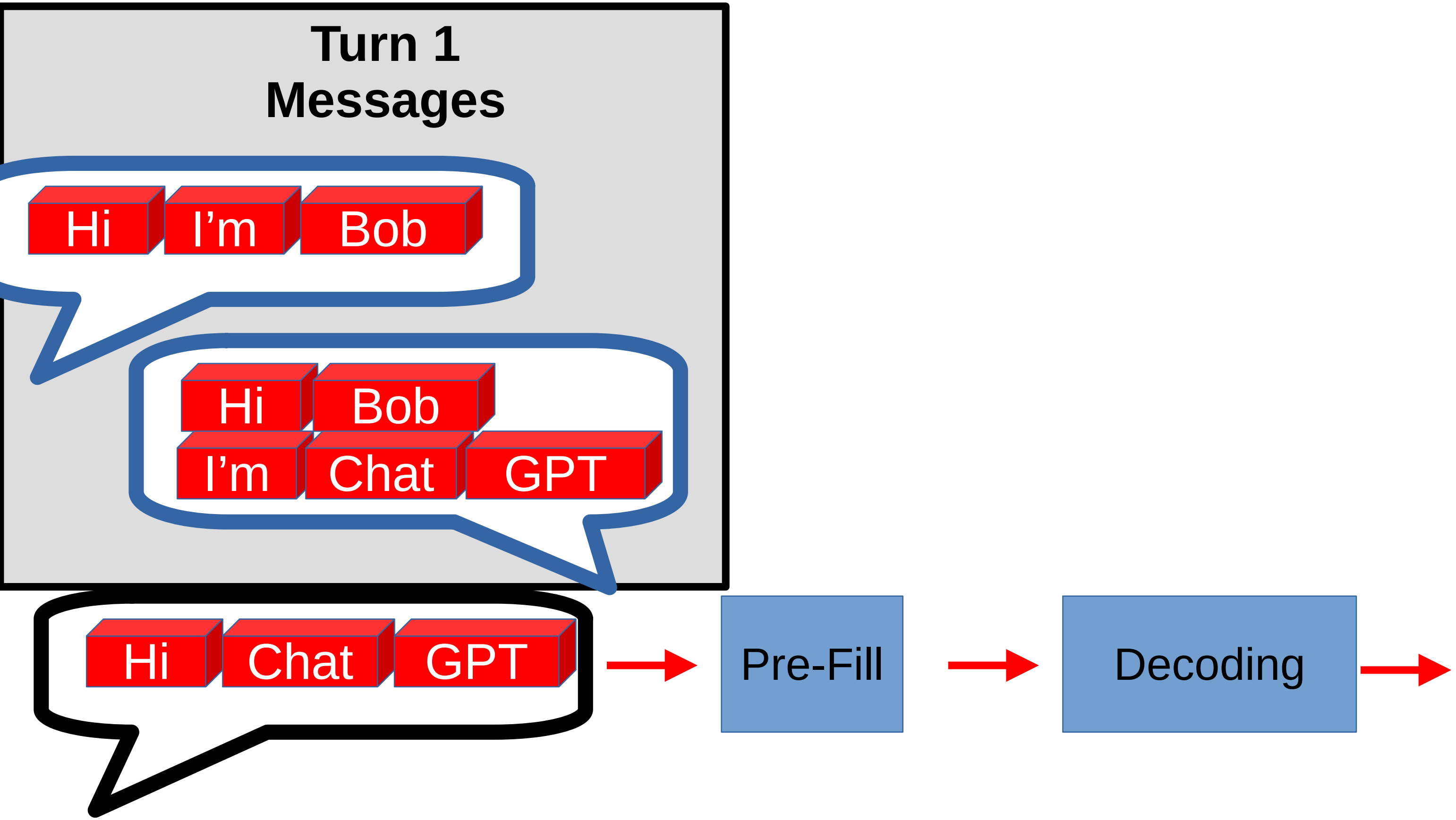
Pre-Fill and Decoding: Turn 2



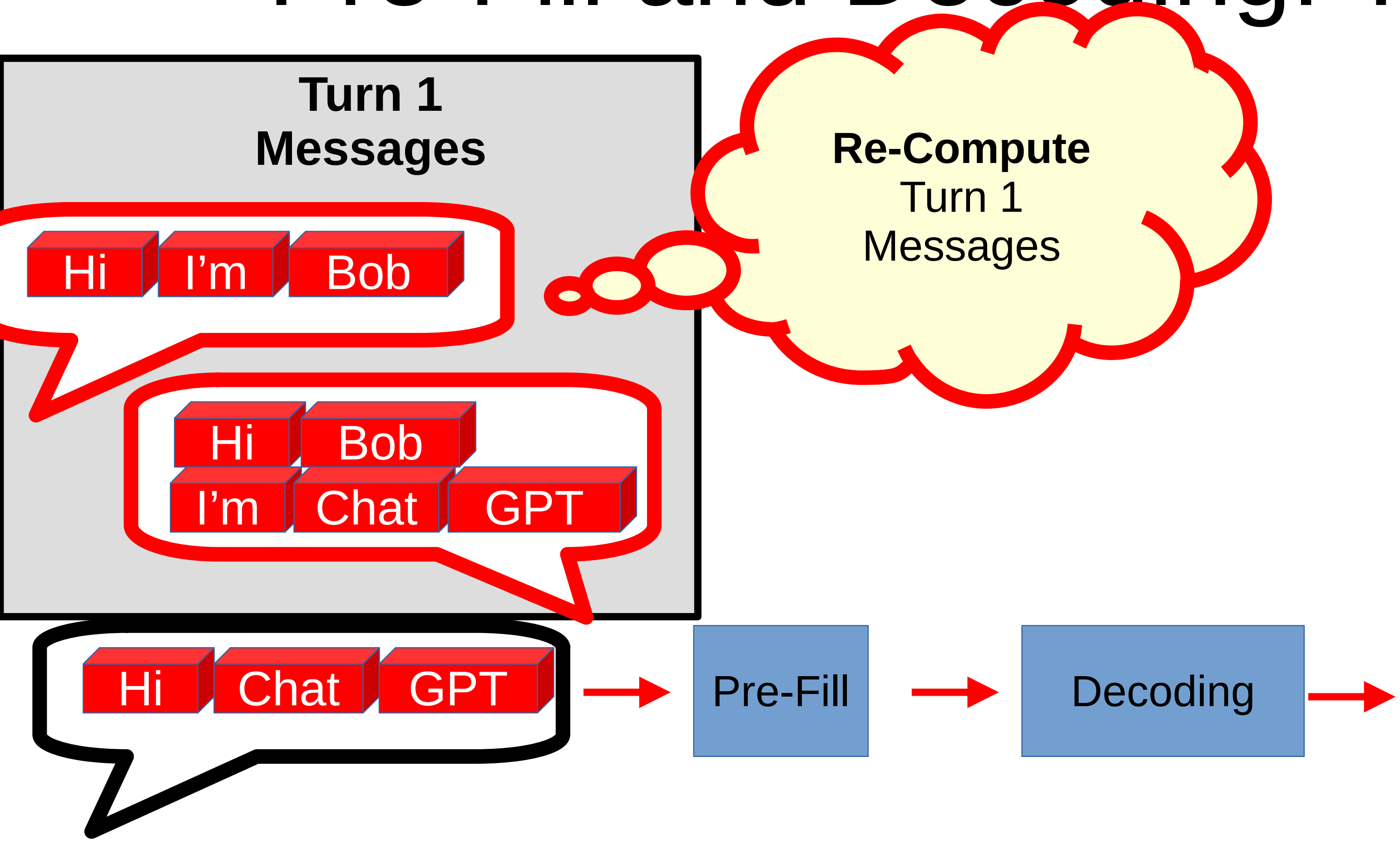
Pre-Fill and Decoding: Turn 2

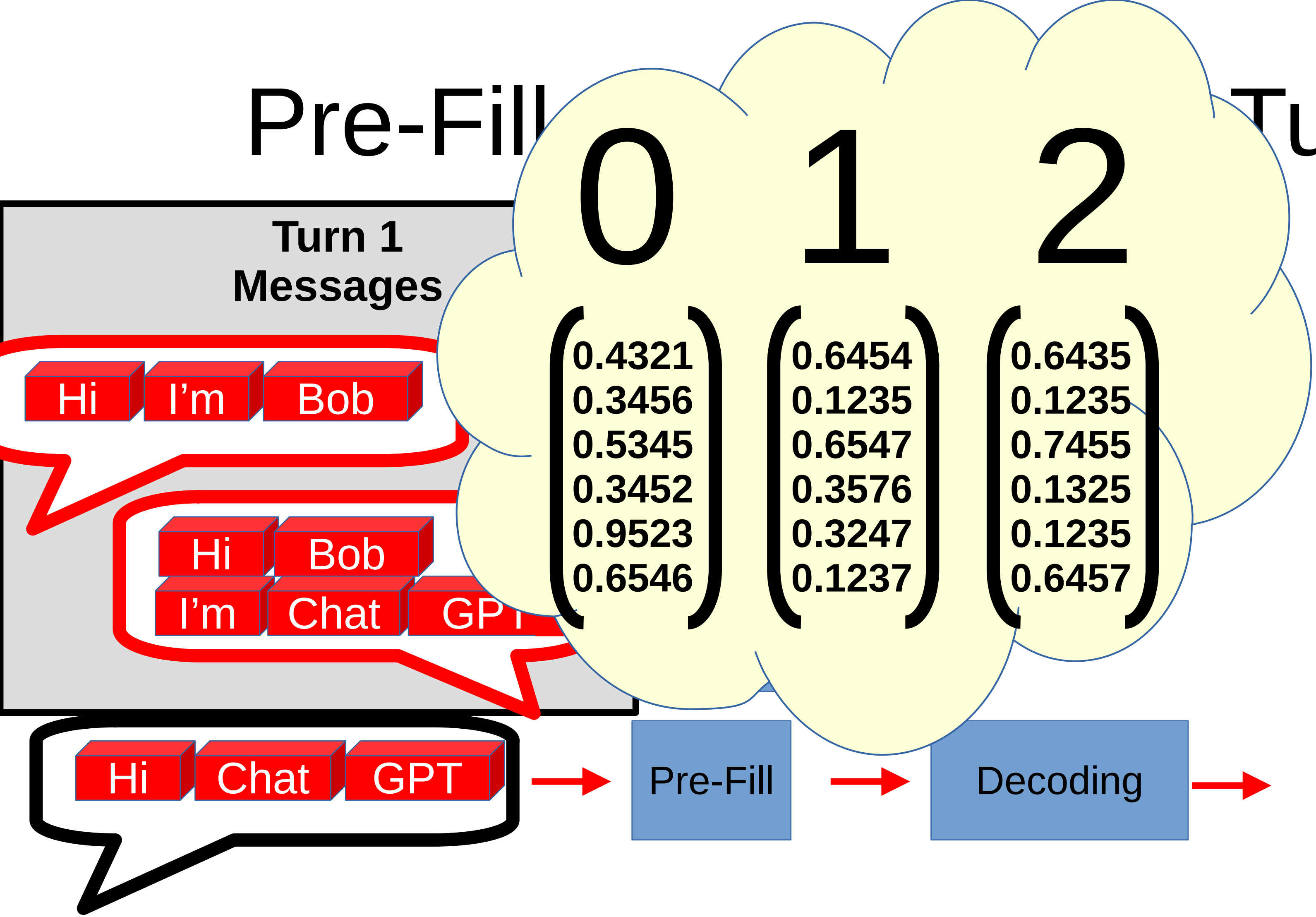


Pre-Fill and Decoding: Turn 2

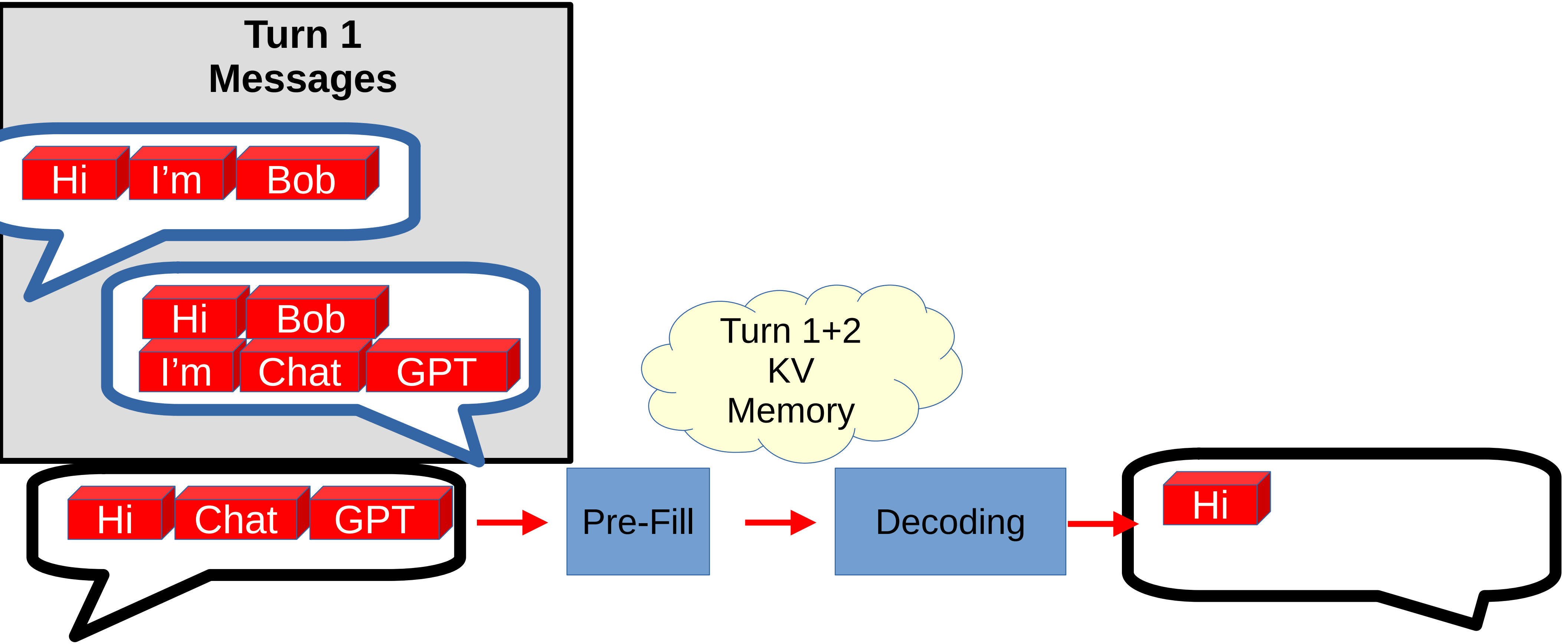


Pre-Fill and Decoding: Turn 2

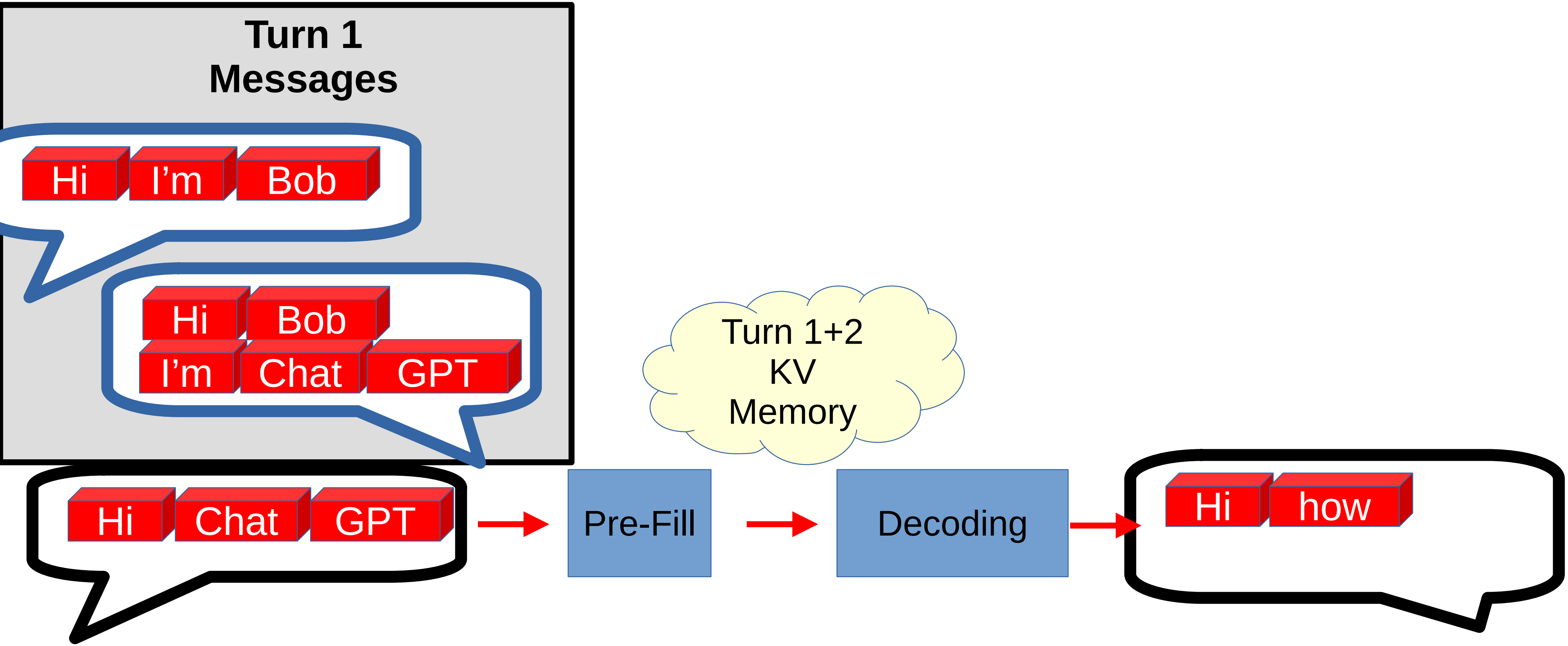



Pre-Fill
Turn 2
Turn 1
Messages
Hi
I'm
Bob
Hi
Bob
I'm
Chat
Hi
Chat
GPT
Pre-Fill
Decoding
0
1
2
0.4321
0.3456
0.5345
0.3452
0.9523
0.6546
0.6454
0.1235
0.6547
0.3576
0.3247
0.1237
0.6435
0.1235
0.7455
0.1325
0.1235
0.6457

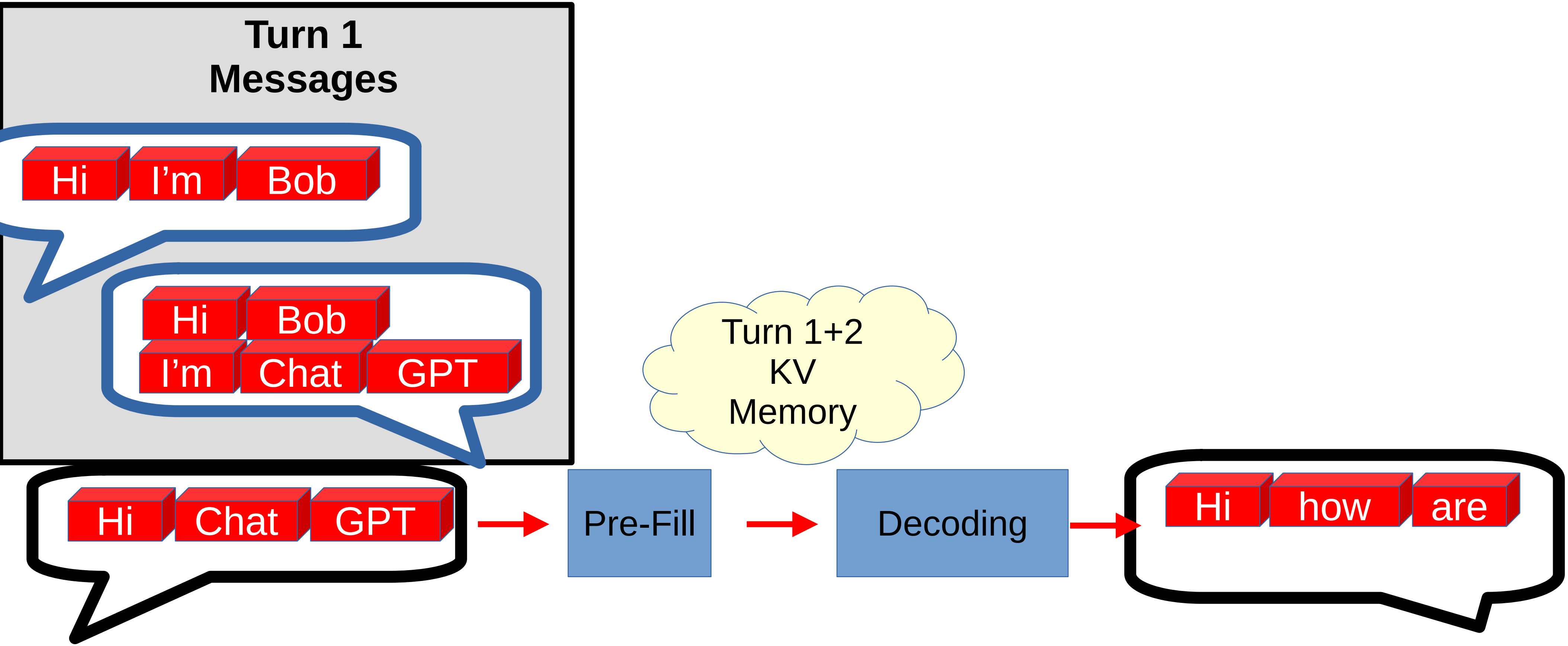
Pre-Fill and Decoding: Turn 2



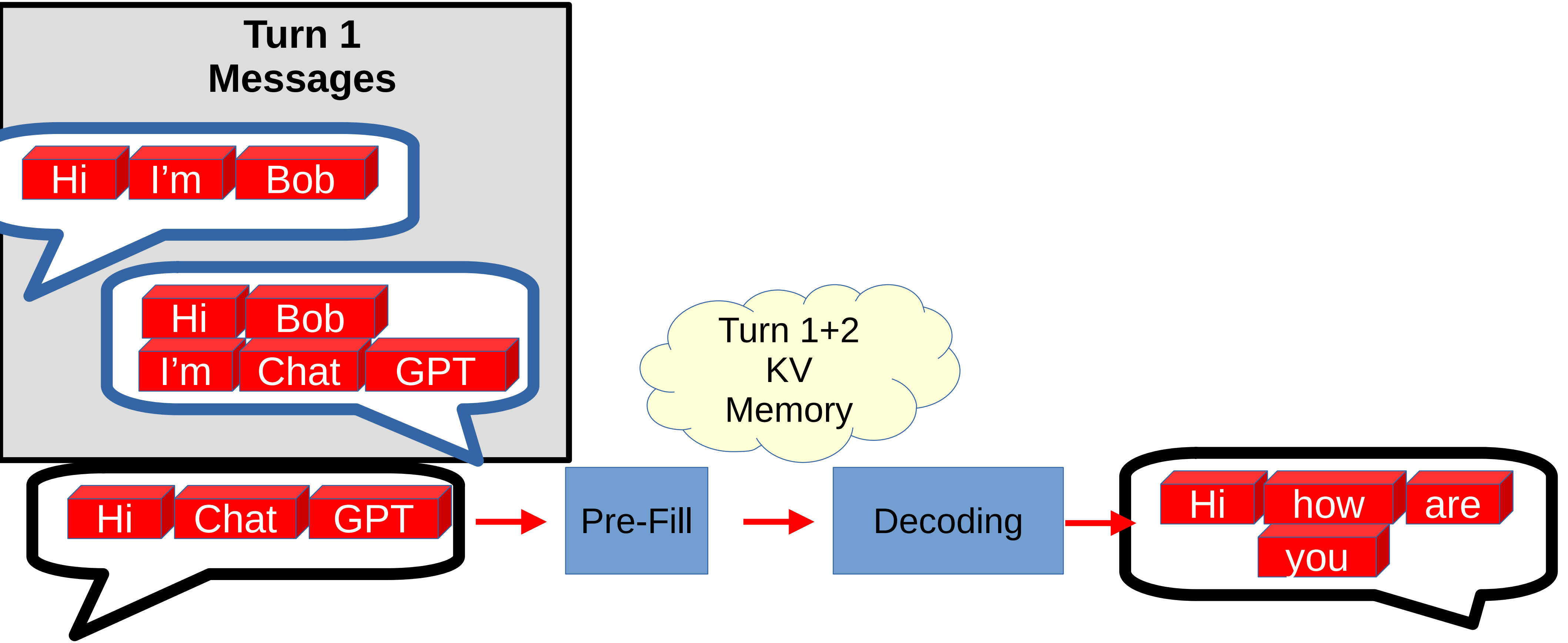
Pre-Fill and Decoding: Turn 2



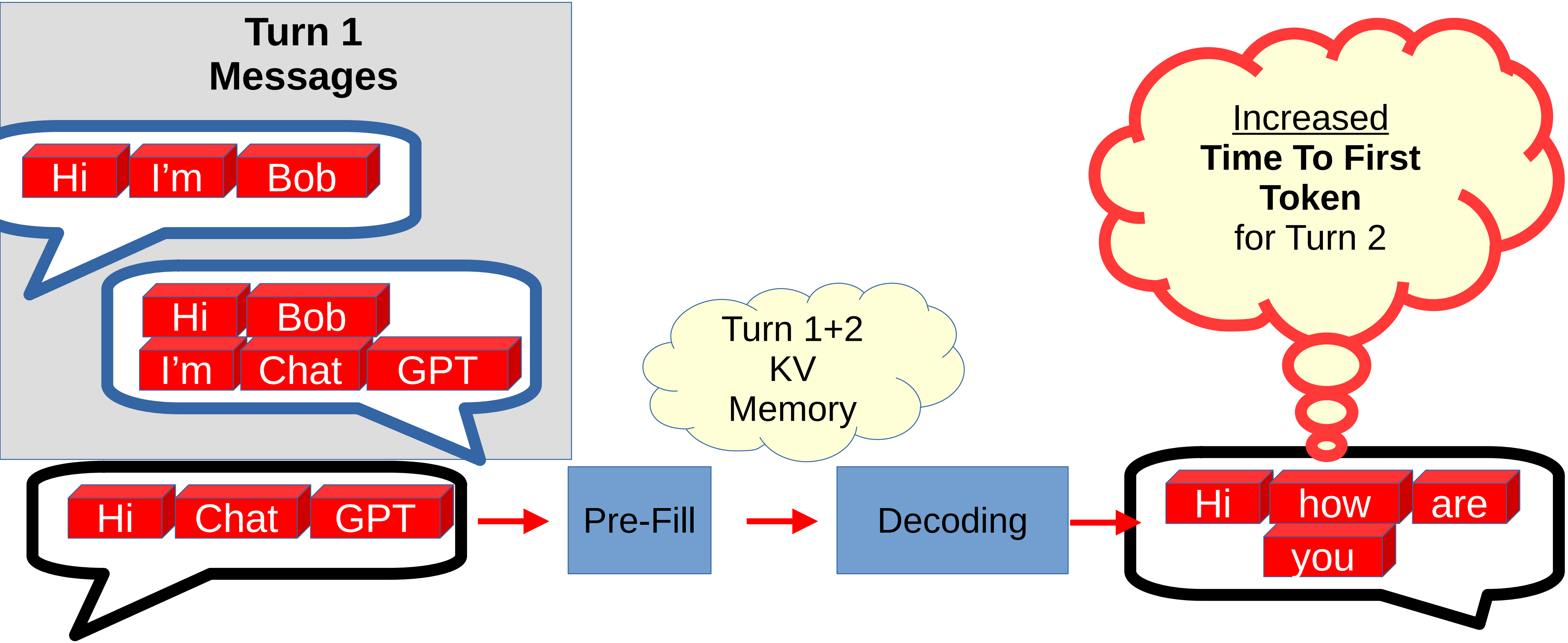
Pre-Fill and Decoding: Turn 2



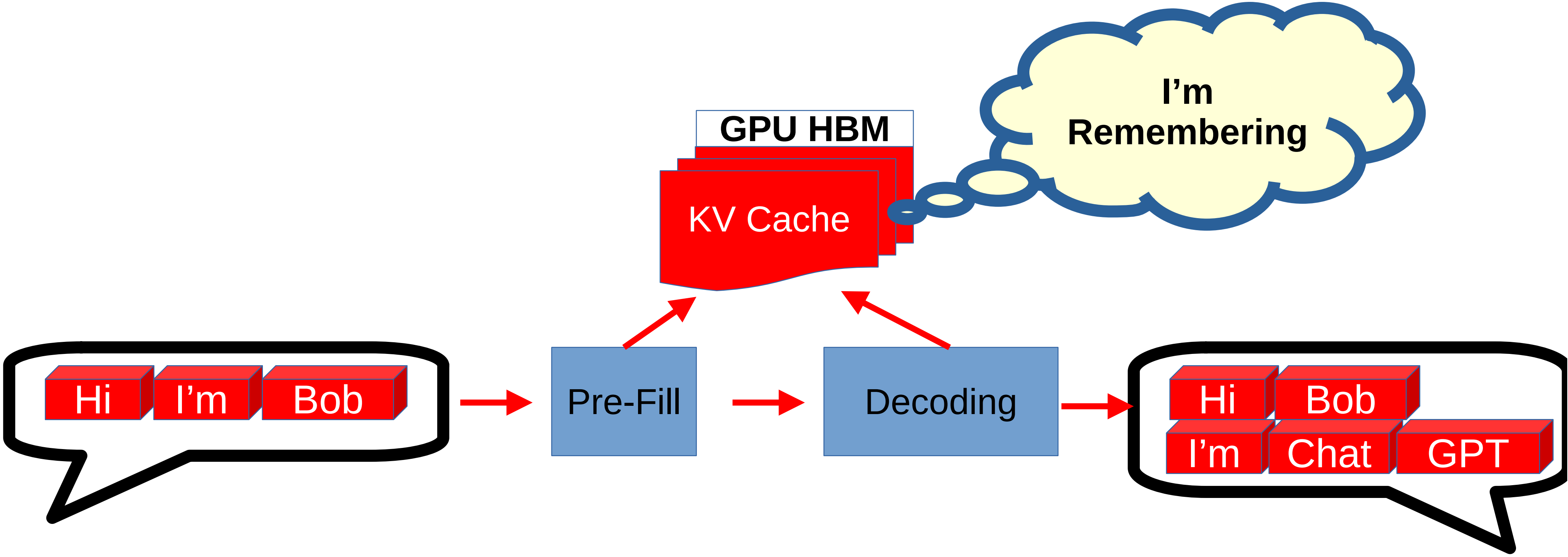
Pre-Fill and Decoding: Turn 2



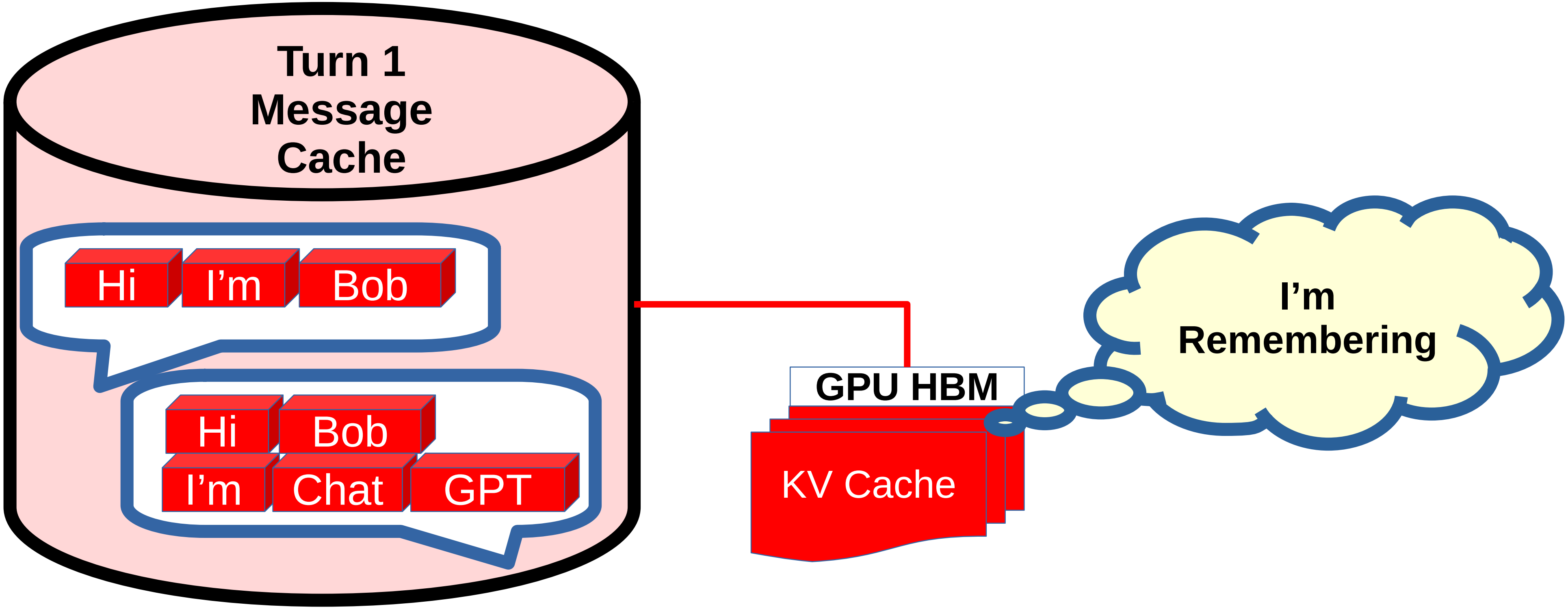
Pre-Fill and Decoding: Turn 2



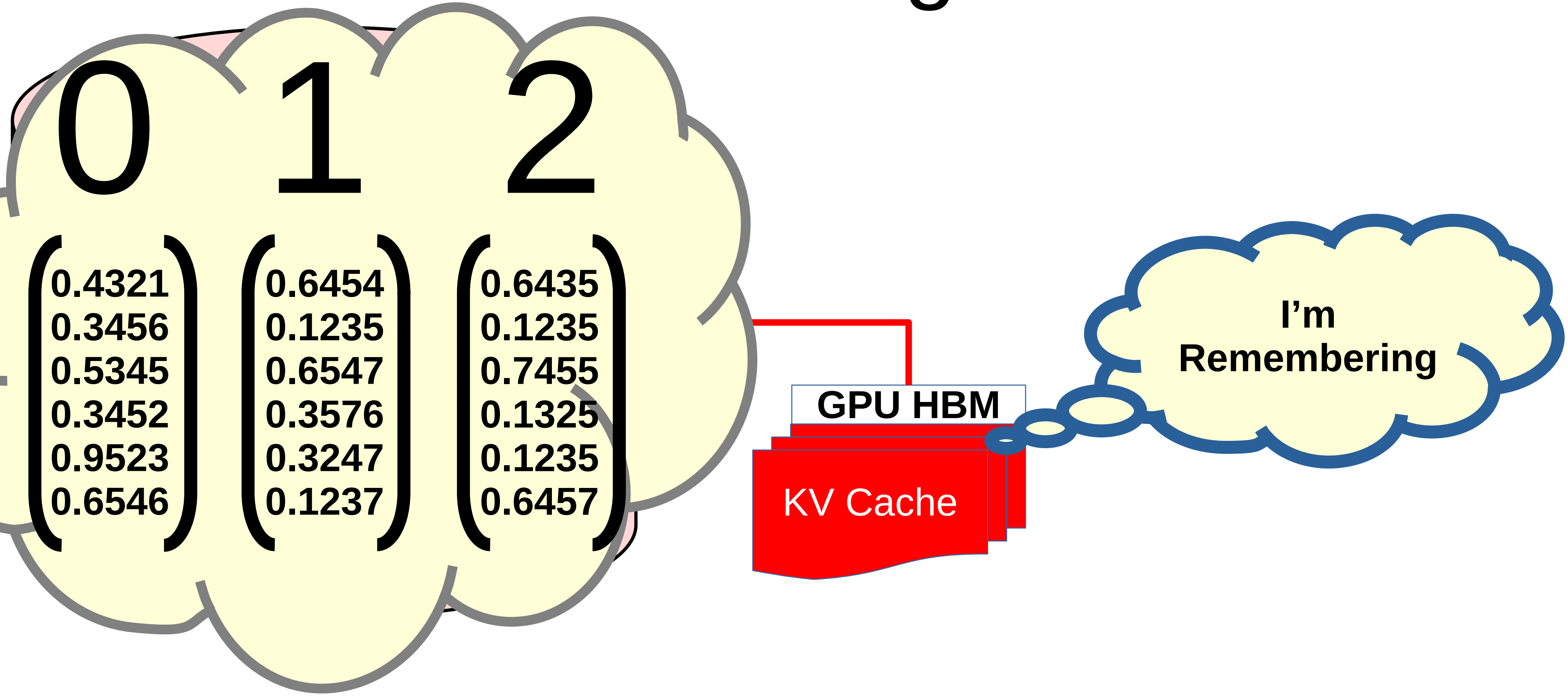
Pre-Fill and Decoding **+ Cache**: Turn 1



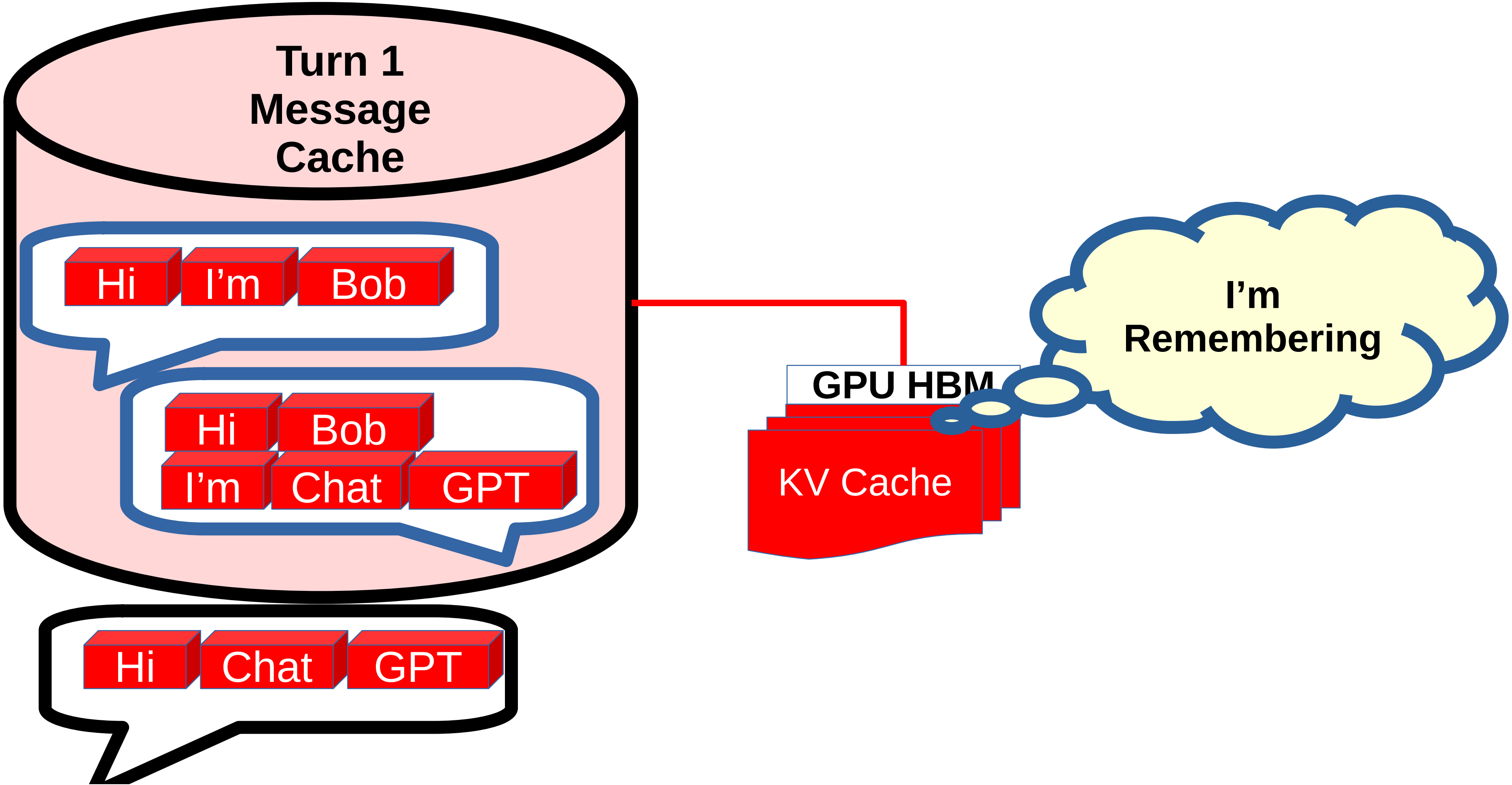
Pre-Fill and Decoding **+ Cache**: Turn 2



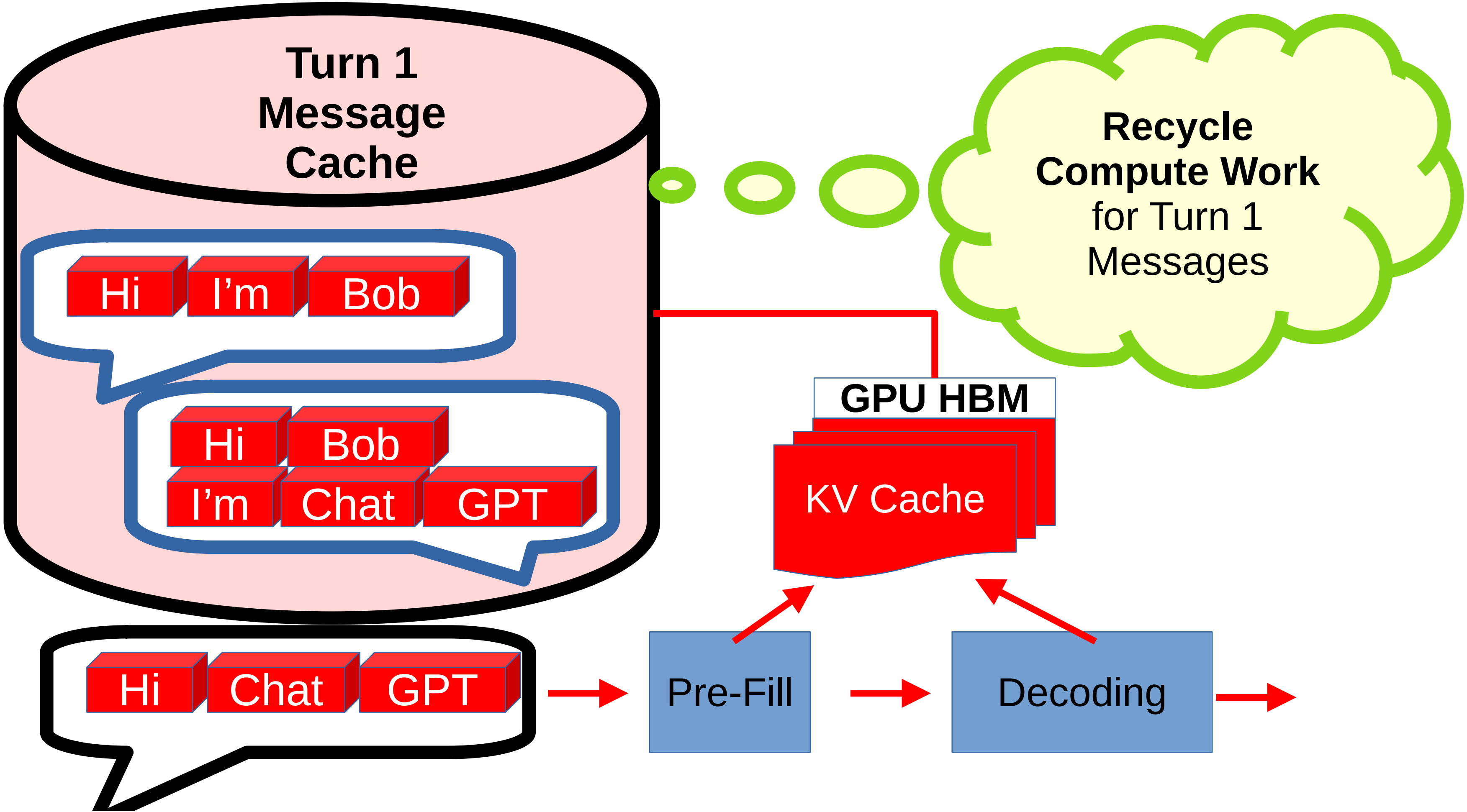
Pre-Fill and Decoding **+ Cache**: Turn 2



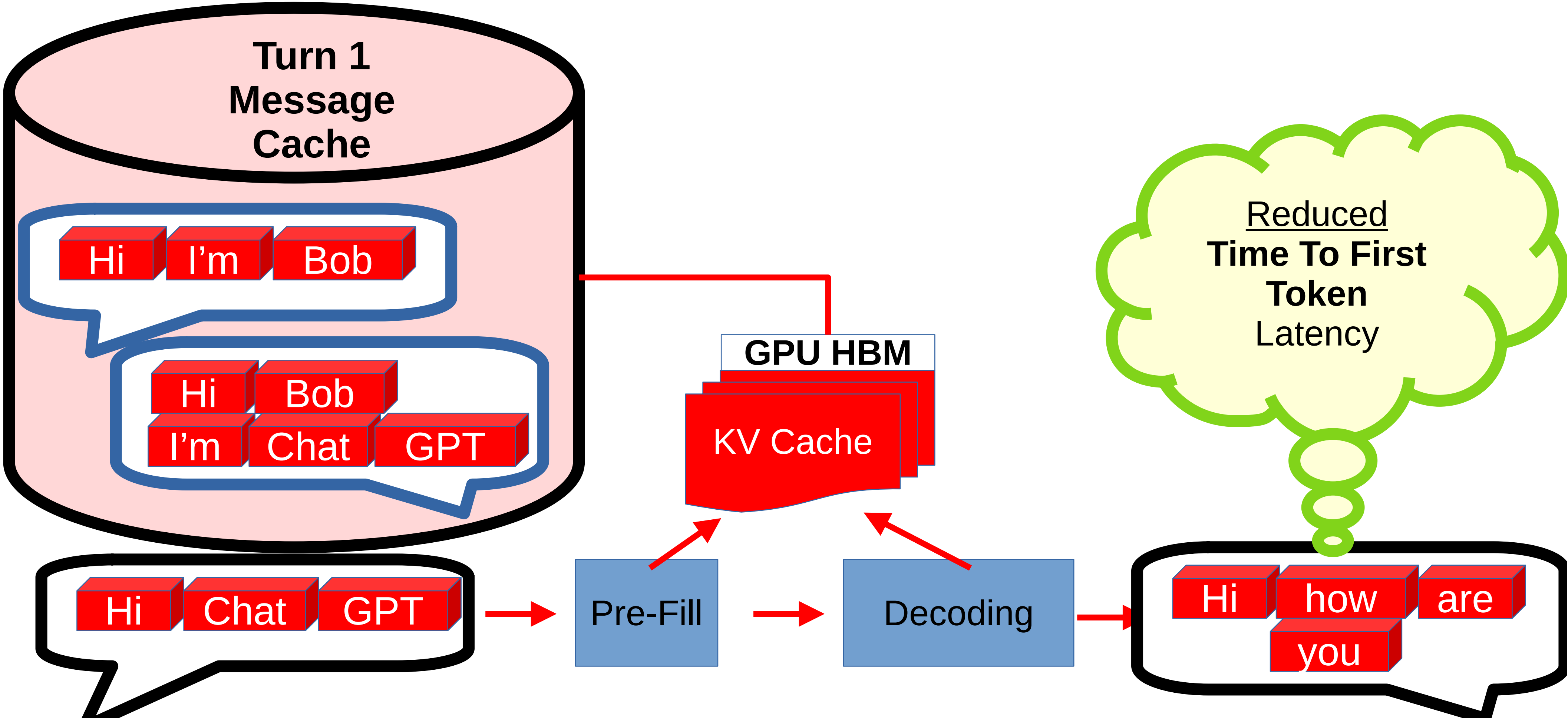
Pre-Fill and Decoding **+ Cache**: Turn 2



Pre-Fill and Decoding + **Cache**: Turn 2



Pre-Fill and Decoding + **Cache**: Turn 2

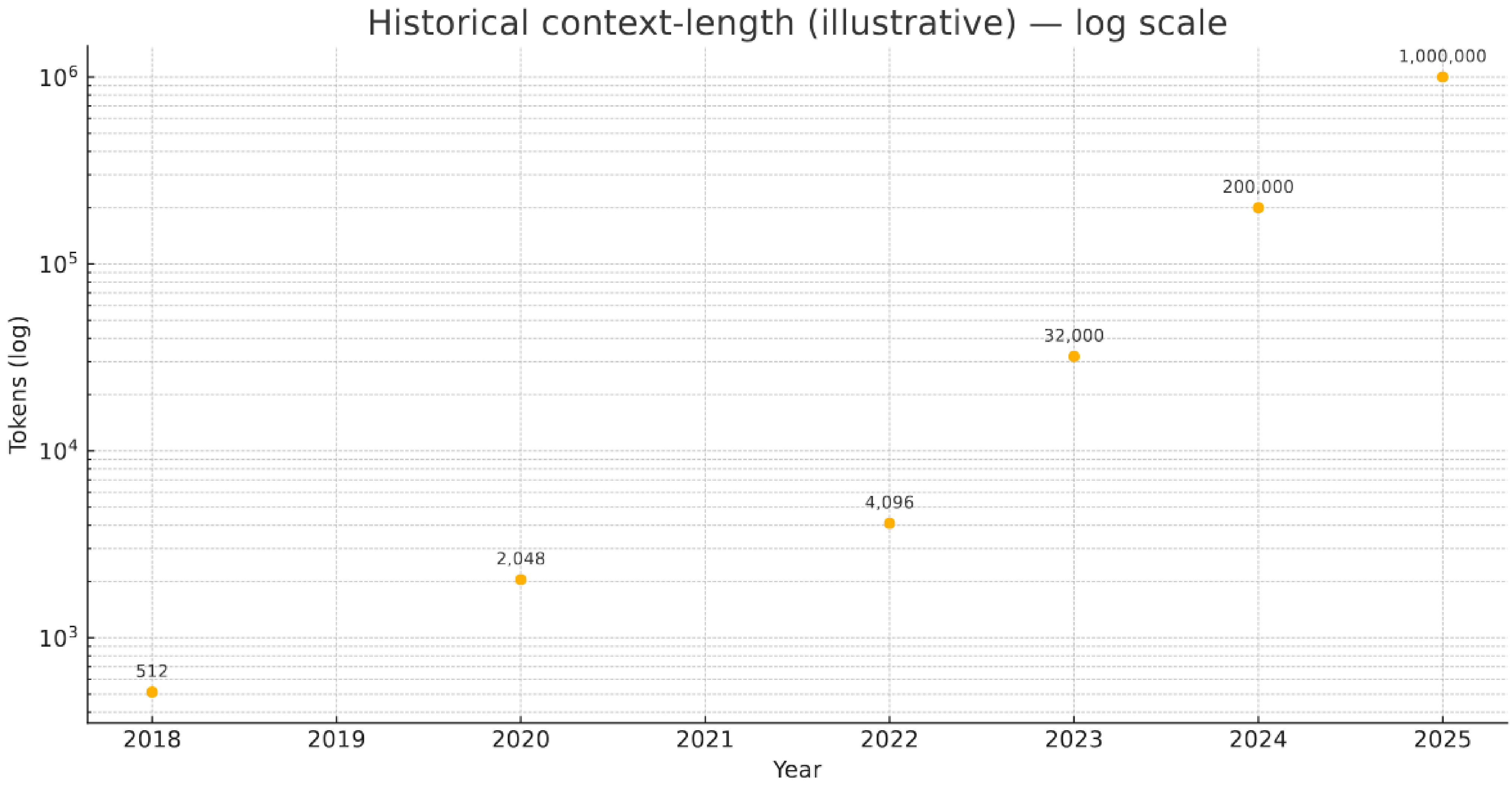


3

Trends in Long Context

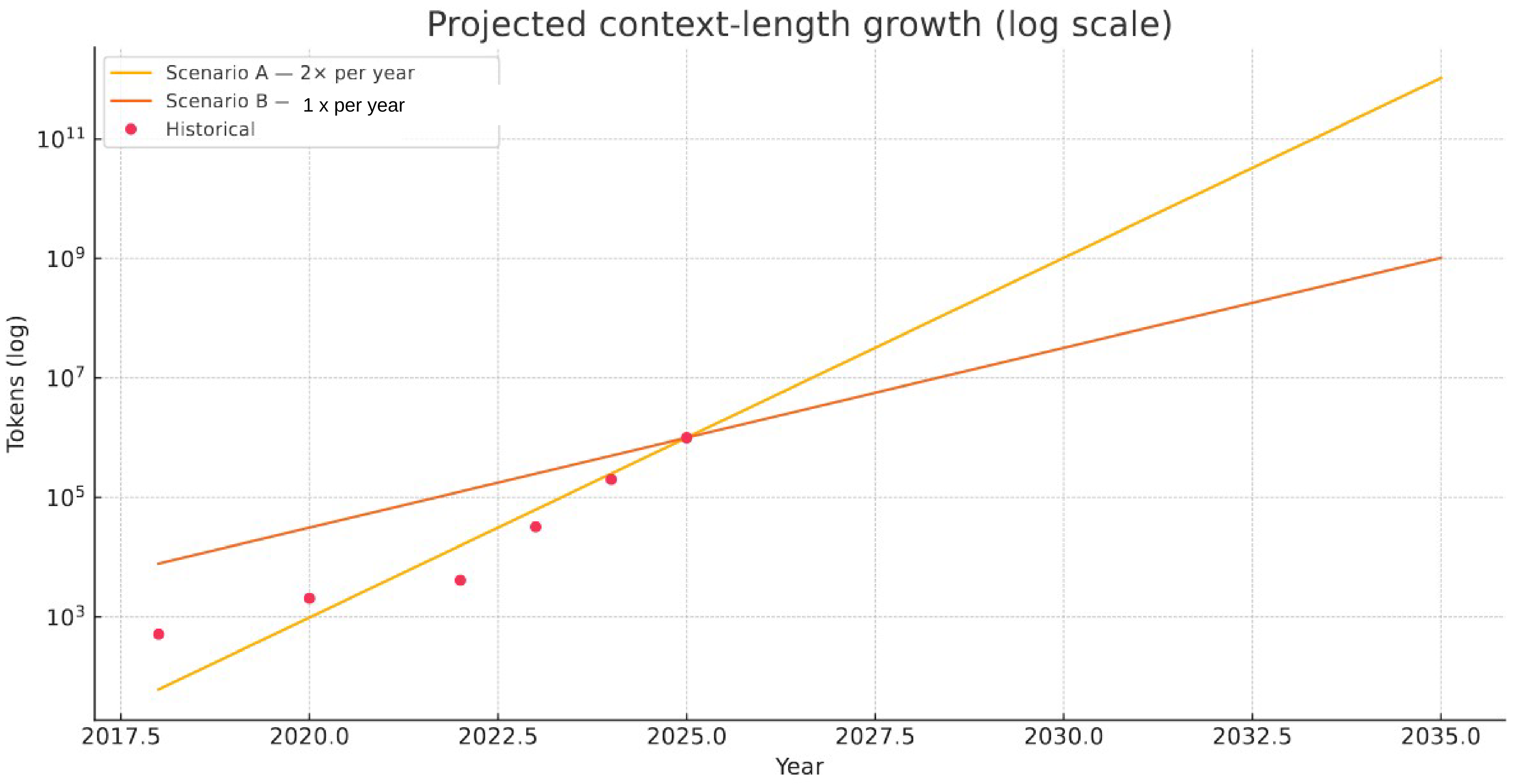
LLM Context Windows

- LLM **Context Windows double** in **sequence length** on average **twice per year**.
- **KV Cache scales with Context Length.**


Historical context-length (illustrative) — log scale
Tokens (log)
Year
10^6
10^5
10^4
10^3
2018
2019
2020
2021
2022
2023
2024
2025
512
2,048
4,096
32,000
200,000
1,000,000

Recent updates

- Gemini 1.5 Pro claims to support **2,000,000 input token** sequence lengths.
- Algorithms exist for **10,000,000 input token** sequence lengths, but no managed model service offers APIs based on this.



Projected context-length growth (log scale)
Scenario A — 2× per year
Scenario B — 1 x per year
Historical
10^{11}
10^9
10^7
10^5
10^3
Tokens (log)
2017.5
2020.0
2022.5
2025.0
2027.5
2030.0
2032.5
2035.0
Year

Milestone Years — Scenario A (doubling twice per year)

Target tokens → estimated crossing year				
• Personal chat history (power user)	:	1e+06	→	2025.0
• Large org chat/year	:	1e+08	→	2028.3
• English Wikipedia (text)	:	5e+09	→	2031.1
• All public GitHub issues+PRs (est.)	:	1e+11	→	2033.3
• All human chats – conservative	:	1e+14	→	2038.3
• All human chats – mid	:	1e+15	→	2039.9
• All human chats – aggressive	:	1e+16	→	2041.6

Parameters: S_0=1,000,000 tokens at t_0=2025, f=2.0 doublings/year

Milestone Years — Scenario B (doubling once per year)

```
Target tokens → estimated crossing year
• Personal chat history (power user)   :          1e+06  →  2025.0
• Large org chat/year                  :          1e+08  →  2031.6
• English Wikipedia (text)             :          5e+09  →  2037.3
• All public GitHub issues+PRs (est.)  :          1e+11  →  2041.6
• All human chats — conservative       :          1e+14  →  2051.6
• All human chats — mid                :          1e+15  →  2054.9
• All human chats — aggressive         :          1e+16  →  2058.2
```

Parameters: S_0=1,000,000 tokens at t_0=2025, f=1.0 doublings/year

Methodology

1) Assume exponential growth in sequence length S(t):

$S(t) = S_0 \cdot 2^{f(t - t_0)}$\n • S_0 = baseline context today (tokens)

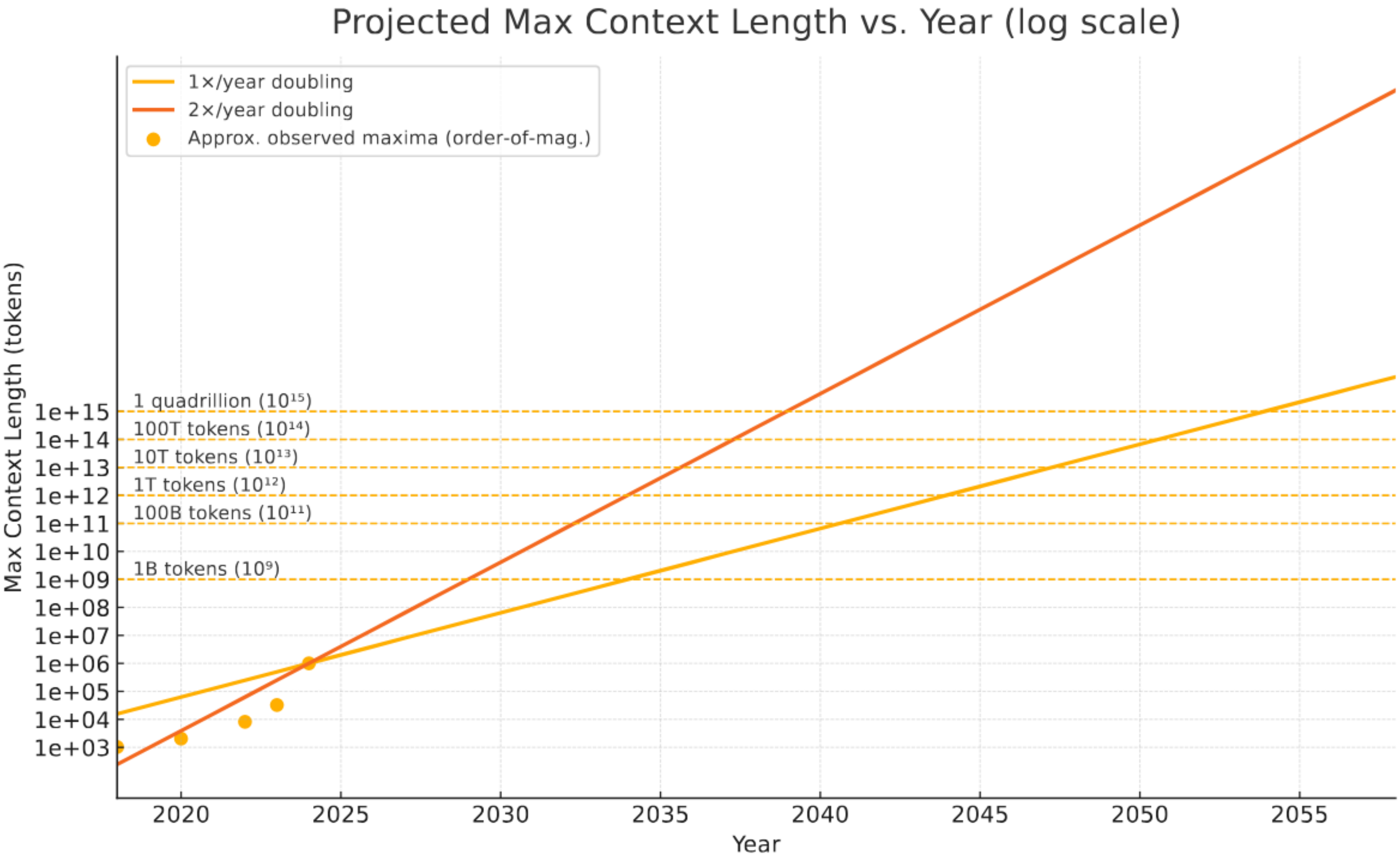
• t_0 = baseline year

• f = doublings/year

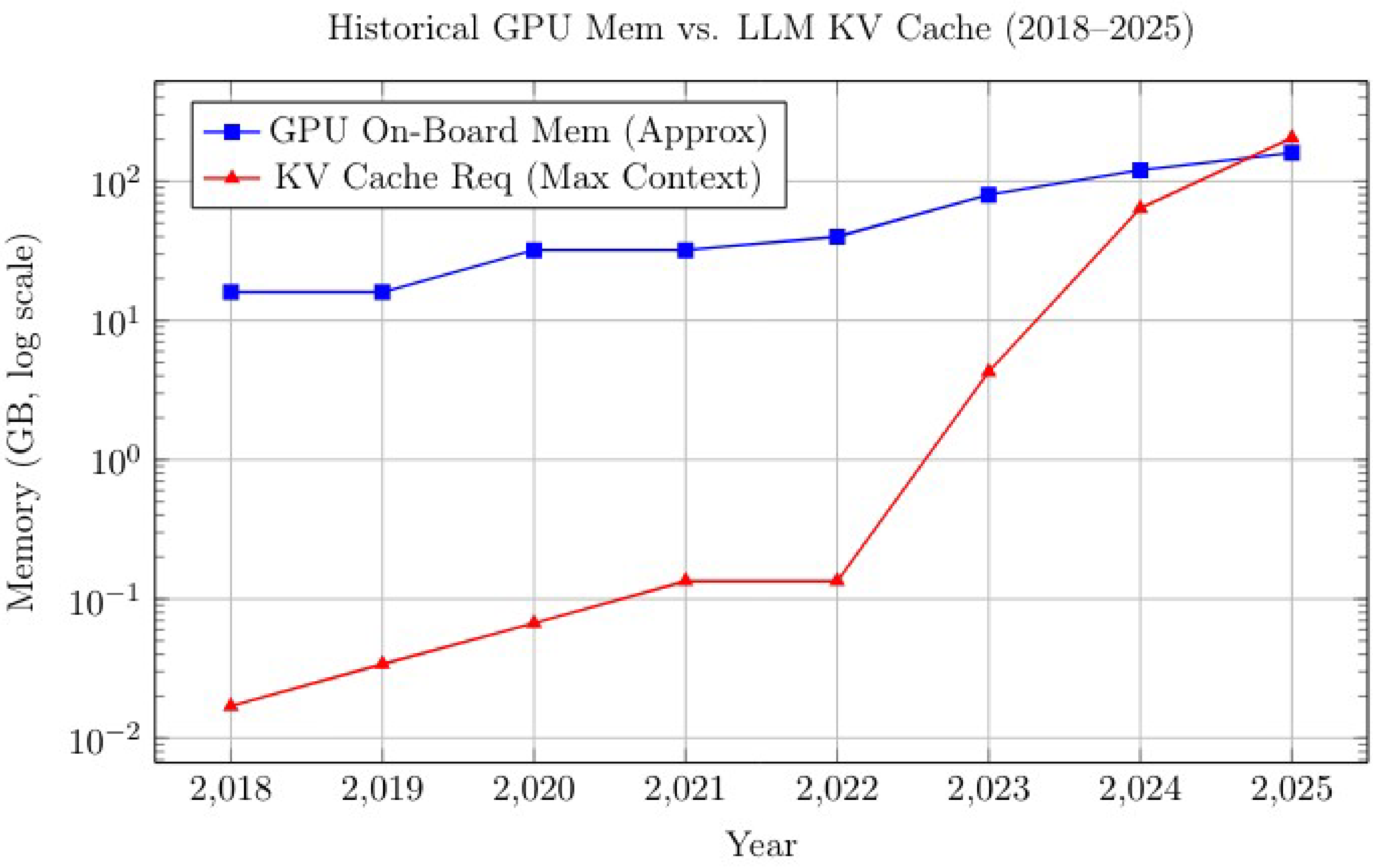
2) Solve for the year when a target corpus of N tokens fits in-window:

$t(N) = t_0 + \frac{\log_2 (N/S_0)}{f}$\n\n3) Historical points provide a qualitative check.

4) Use log-scale plots to visualize multiple orders of magnitude.



Projected Max Context Length vs. Year (log scale)
1×/year doubling
2×/year doubling
Approx. observed maxima (order-of-mag.)
1 quadrillion (10¹⁵)
100T tokens (10¹⁴)
10T tokens (10¹³)
1T tokens (10¹²)
100B tokens (10¹¹)
1B tokens (10⁹)
Max Context Length (tokens)
1e+15
1e+14
1e+13
1e+12
1e+11
1e+10
1e+09
1e+08
1e+07
1e+06
1e+05
1e+04
1e+03
2020
2025
2030
2035
2040
2045
2050
2055
Year


Historical GPU Mem vs. LLM KV Cache (2018–2025)
GPU On-Board Mem (Approx)
KV Cache Req (Max Context)
Memory (GB, log scale)
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
2,018
2,019
2,020
2,021
2,022
2,023
2,024
2,025
Year

Llama-4, KV Cache Problem

- **Interleaved Rotary Positional Embeddings** (**IRoPE**) attention algorithm.
- Alternates between **No Positional Encoding (NoPE)** global attention layers with chunked local-attention layers that use RoPE.
- Attention algorithm makes it **theoretically capable** of supporting **10M token sequence lengths**.
- 10M tokens is roughly **38 of the longest books** on Project Gutenberg.
- **32Tb** of working memory KV Cache required for a **single 10M token inference query**.
- Today's GPU Memory constraints mean this sequence length is **not practical**.

From

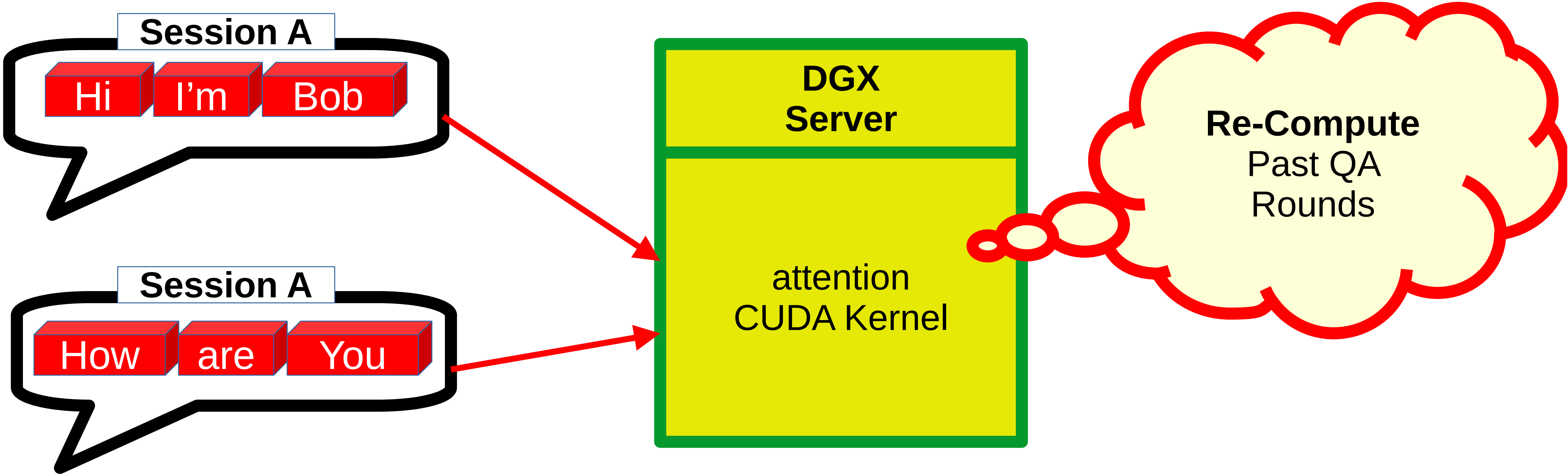
Theory

to

Production

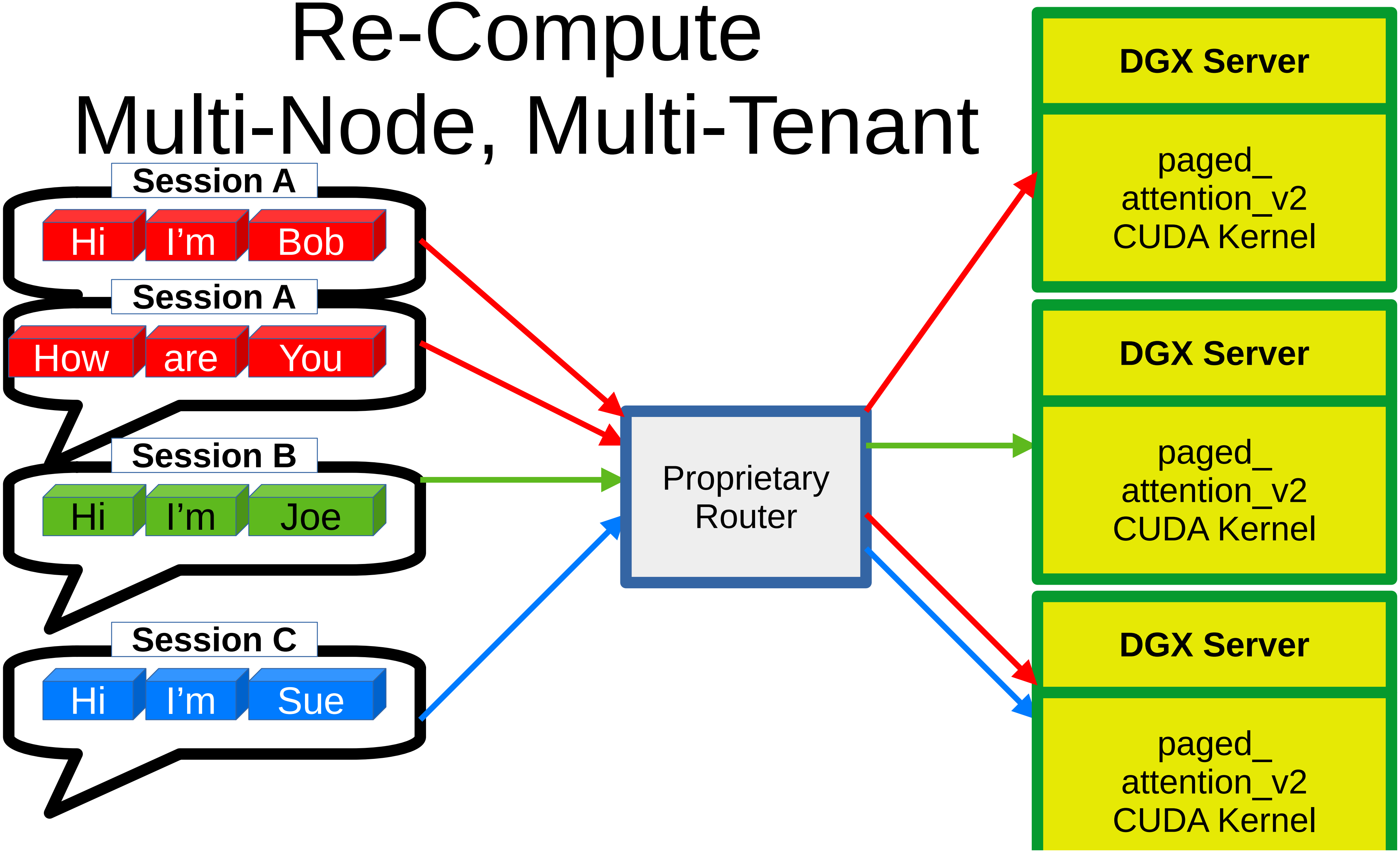
Recompute, **Single User** on **Single Node**.

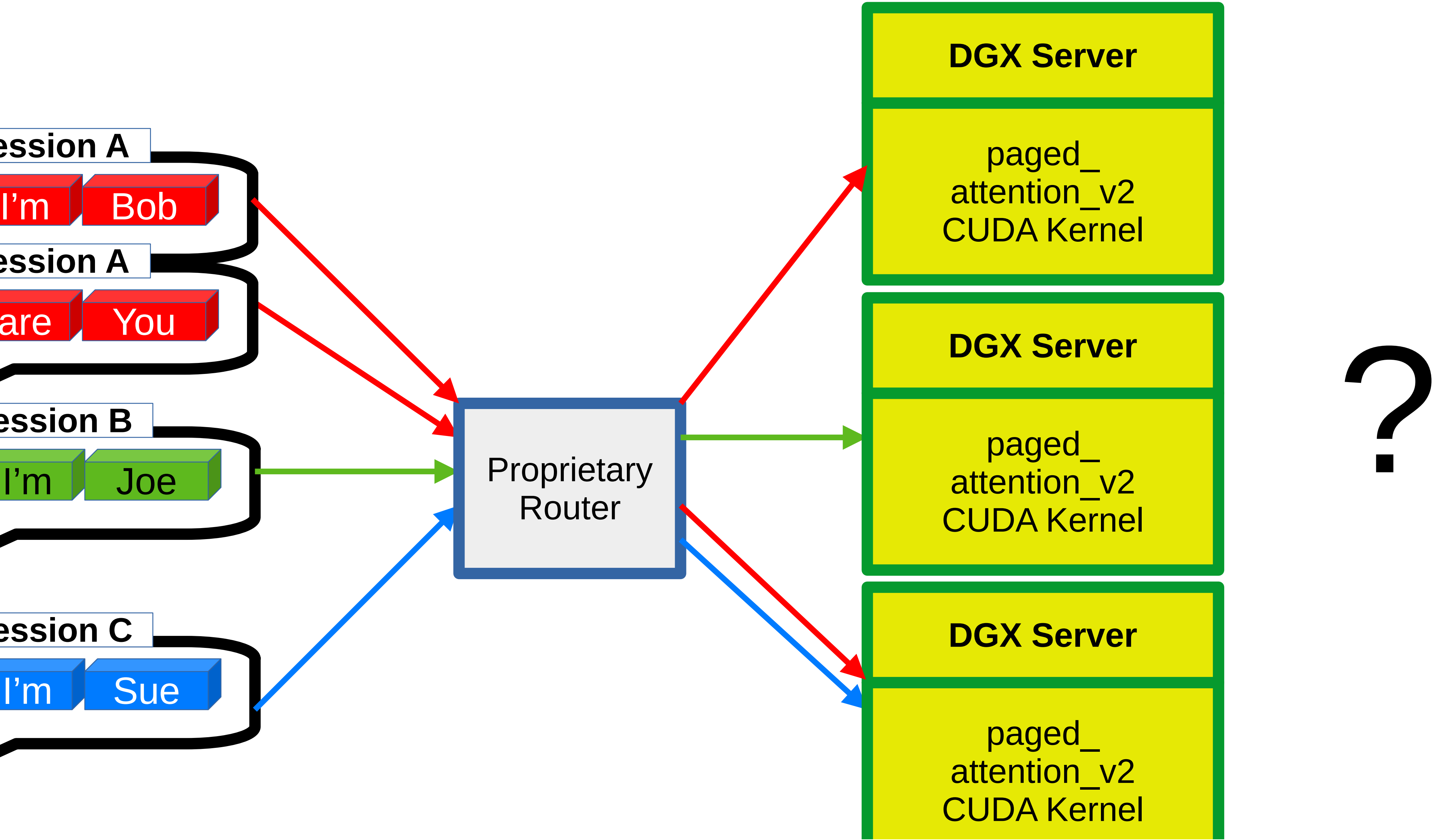
Transformer, Attention: Single Node, Single Tenant



LLM Production… At Scale

Re-Compute
Multi-Node, Multi-Tenant

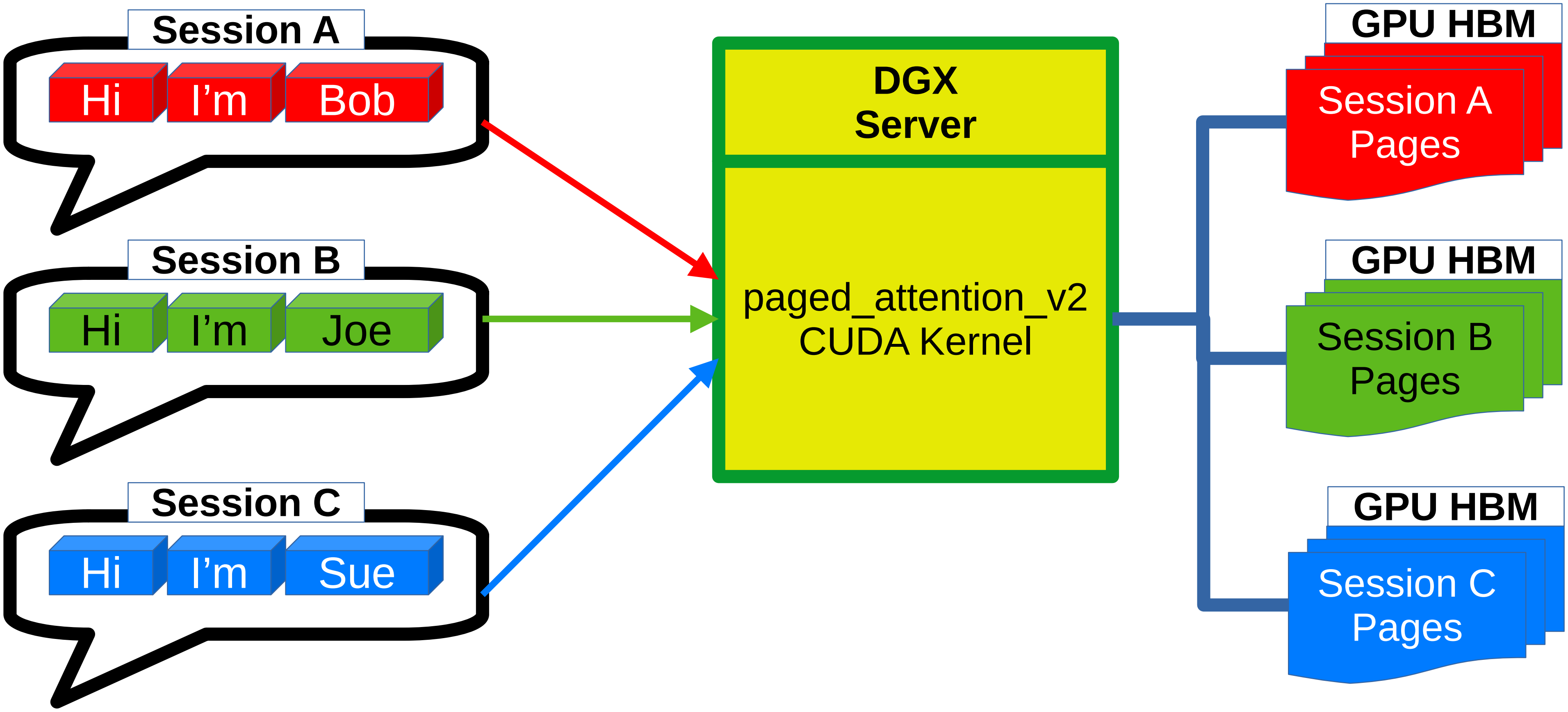



ession A
I'm
Bob
ession A
are
You
ession B
I'm
Joe
ession C
I'm
Sue
Proprietary Router
DGX Server
paged_ attention_v2 CUDA Kernel
DGX Server
paged_ attention_v2 CUDA Kernel
DGX Server
paged_ attention_v2 CUDA Kernel
?

Efficient Memory Management with Paged Attention.

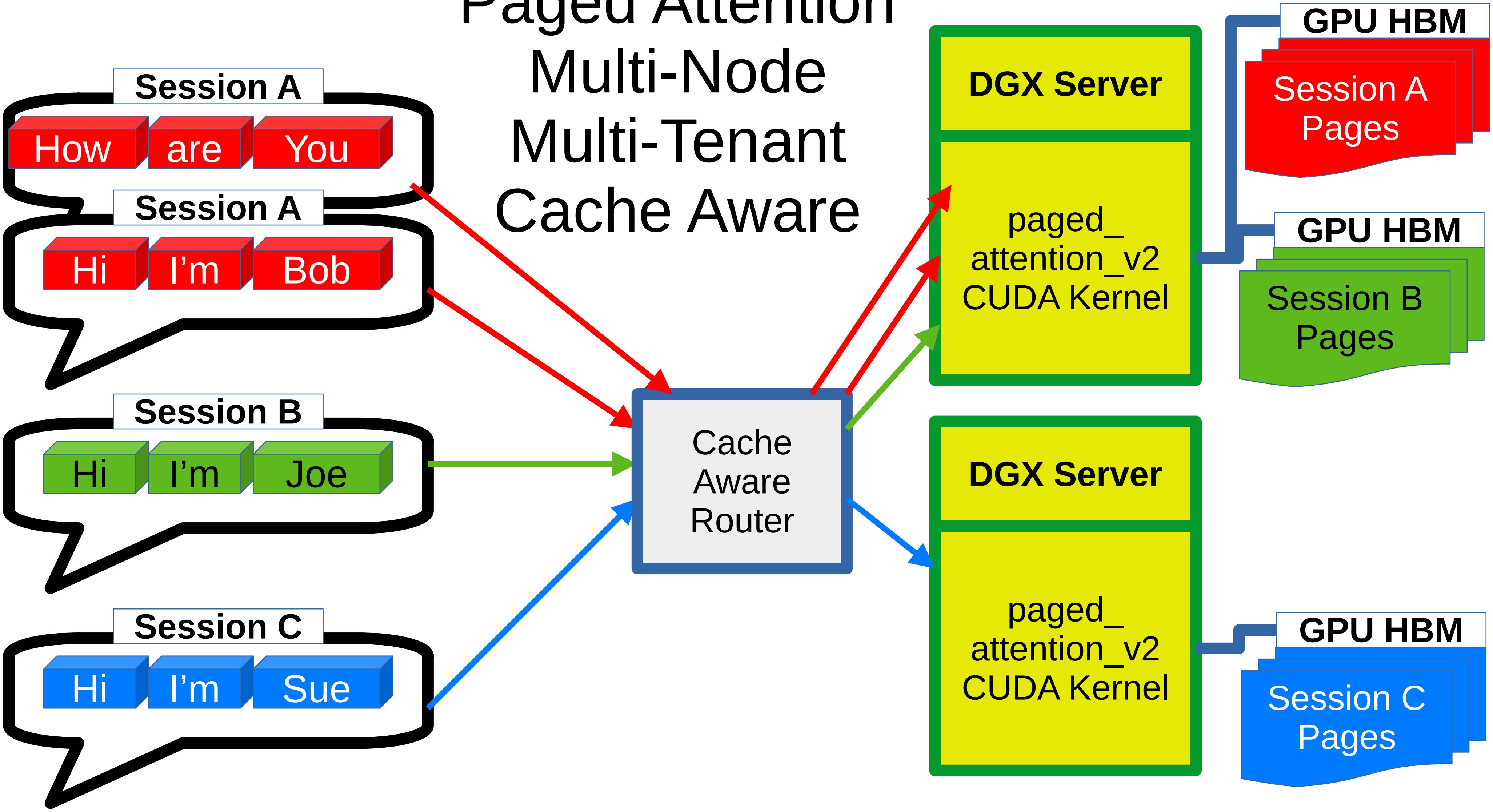
(introduces vLLM)

Paged Attention: Single Node, Multiple Tenants

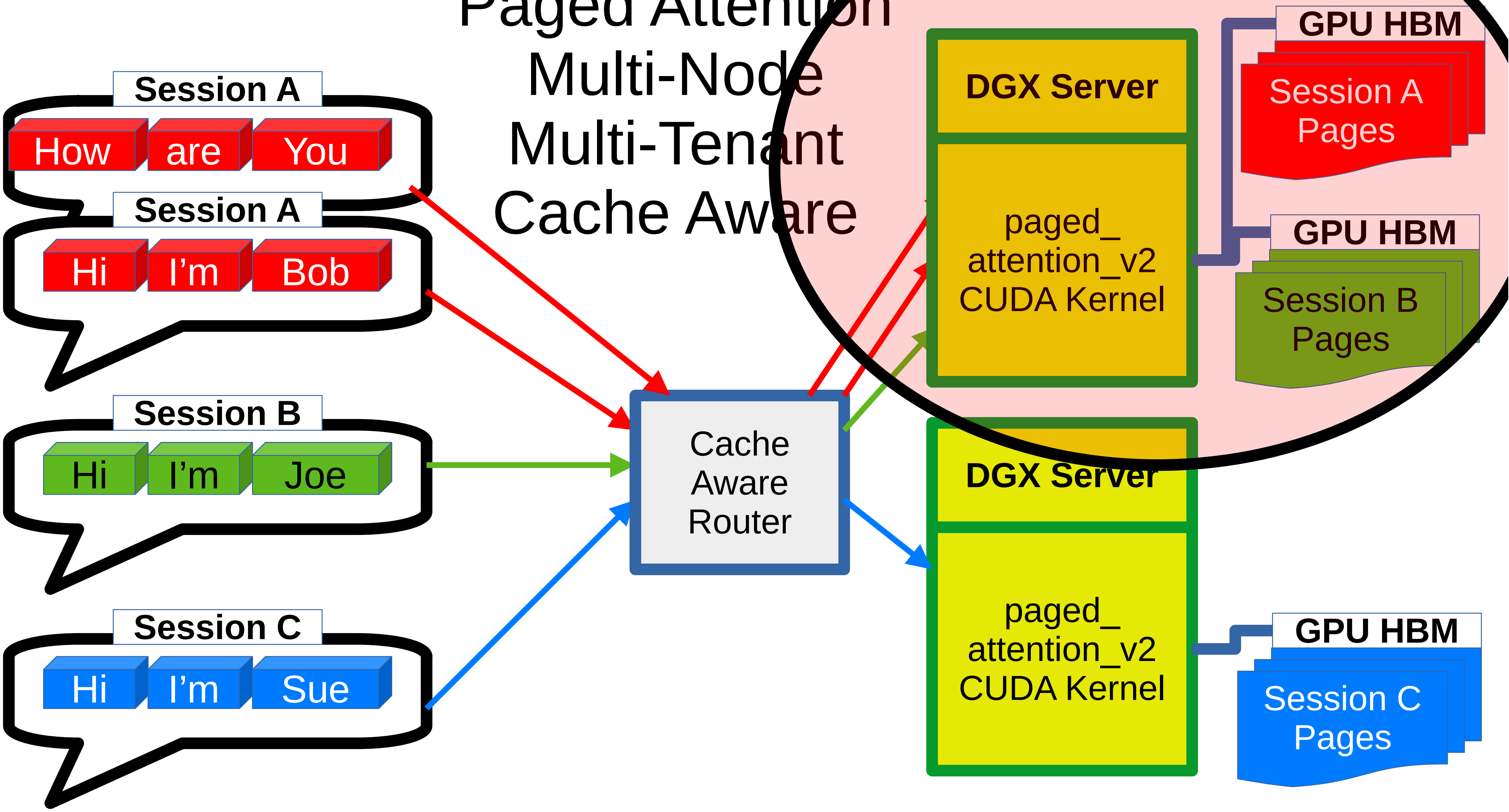


Cache Aware Routing.

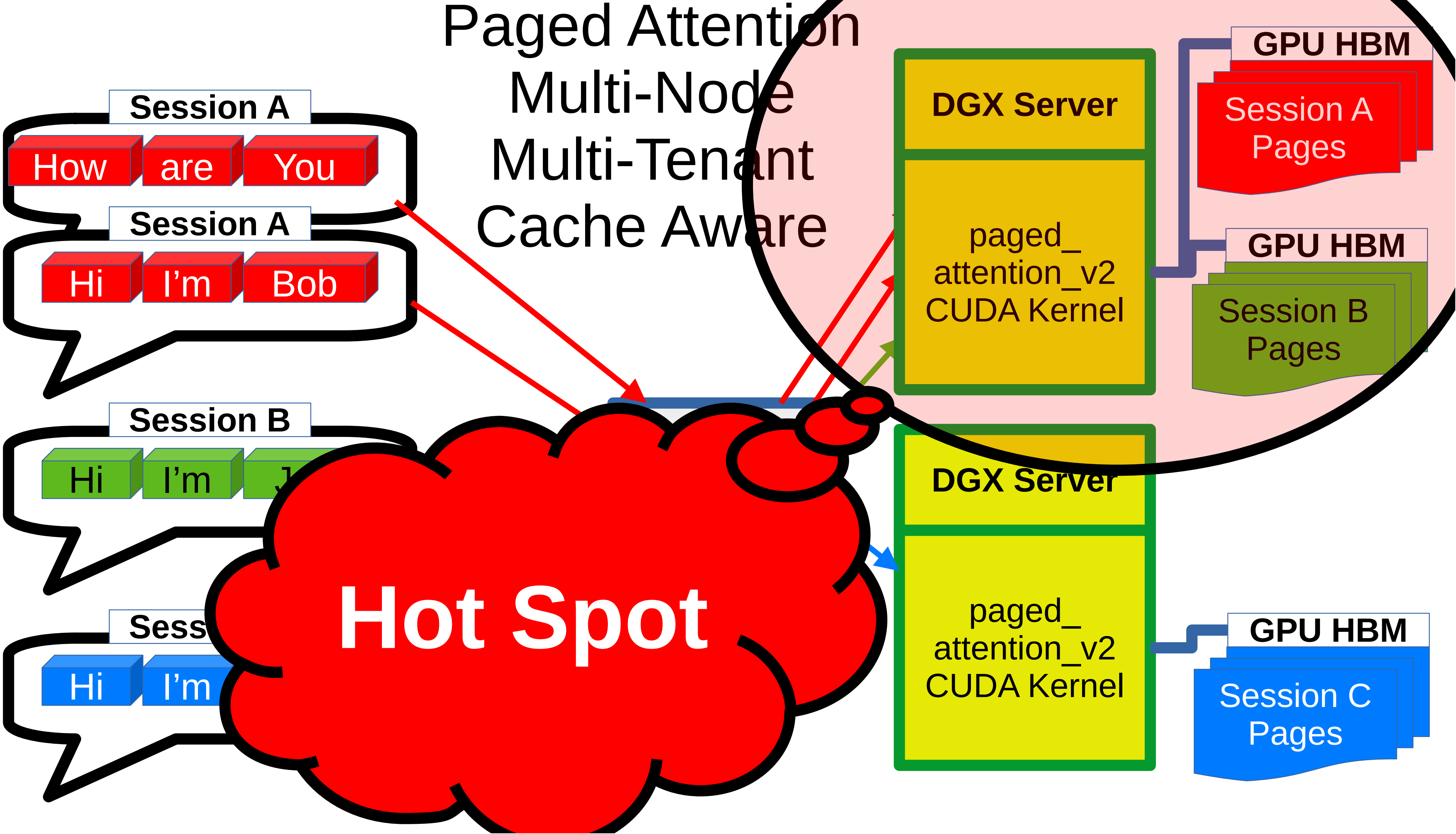
Paged Attention Multi-Node Multi-Tenant Cache Aware



Paged Attention
Multi-Node
Multi-Tenant
Cache Aware



Paged Attention Multi-Node Multi-Tenant Cache Aware



KV Cache Requirements:

1. Tiered Buffer Capacity

2. High Performance

3. Multi-Tenant Live Migration of KV Cache

KV Cache Requirements:

1. Tiered Buffer Capacity
Long Context / Long Time to Eviction

2. High Performance
Minimize GPU Pipeline Stalls
While Waiting For Page Copies

3. Multi-Tenant Live Migration of KV Cache
Handle Node Outages

KV Cache Requirements:

3. Multi-Tenant Live Migration of KV Cache

Prevent Cache Hot Spots
By
Re-Balancing Requests
From **High Traffic Nodes**
To **Low Traffic Nodes**

KV Cache Requirements:

1. Tiered Buffer Capacity
(Support multiple KvCacheTransferModes)

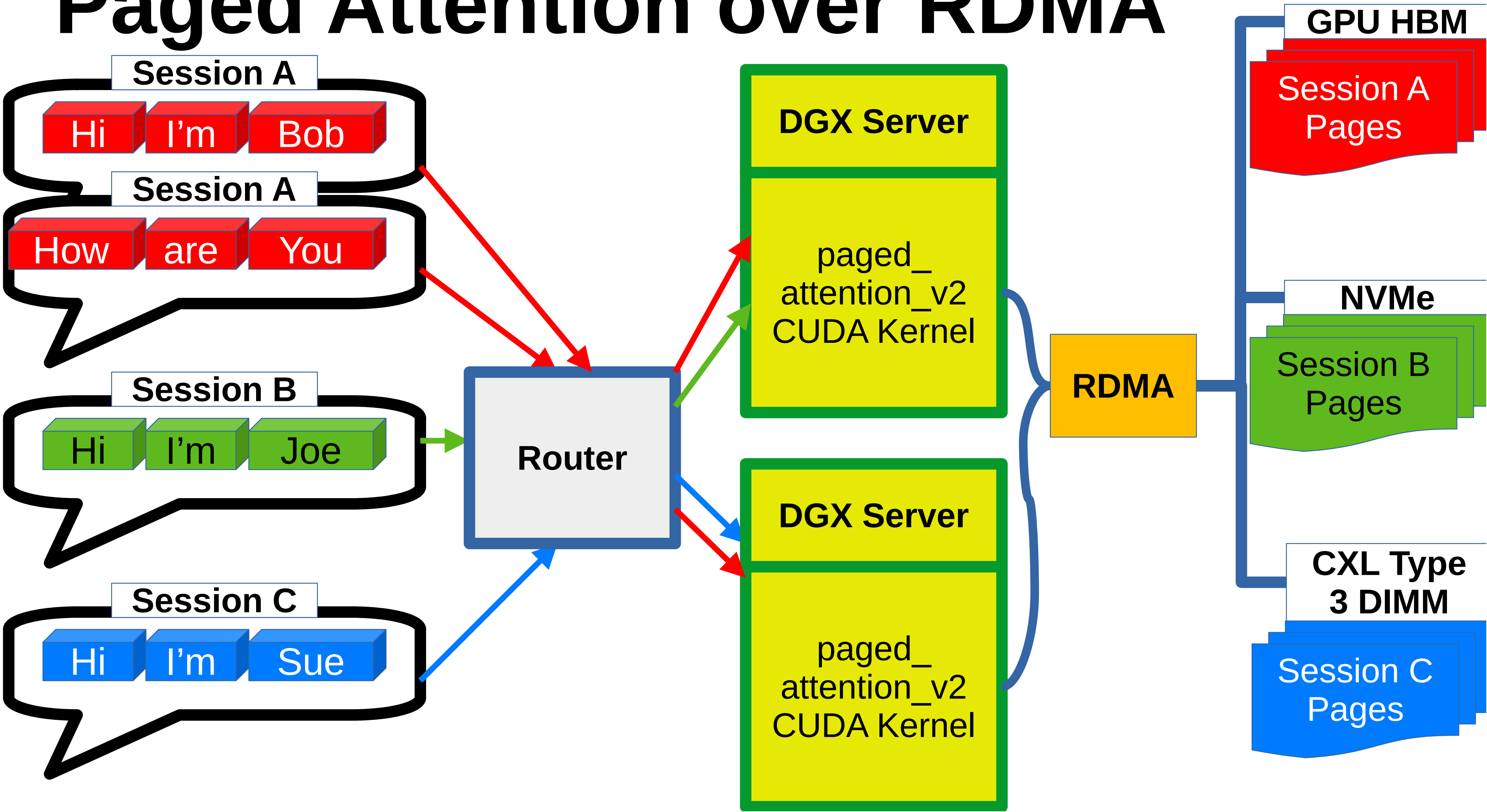
2. High Performance
(UCX, MPI, GPUDirect Storage)

3. Multi-Tenant Live Migration of KV Cache
(RDMA Page Migration)

KV Cache over Fabric

Paged Attention over RDMA (PAoR)

Paged Attention over RDMA

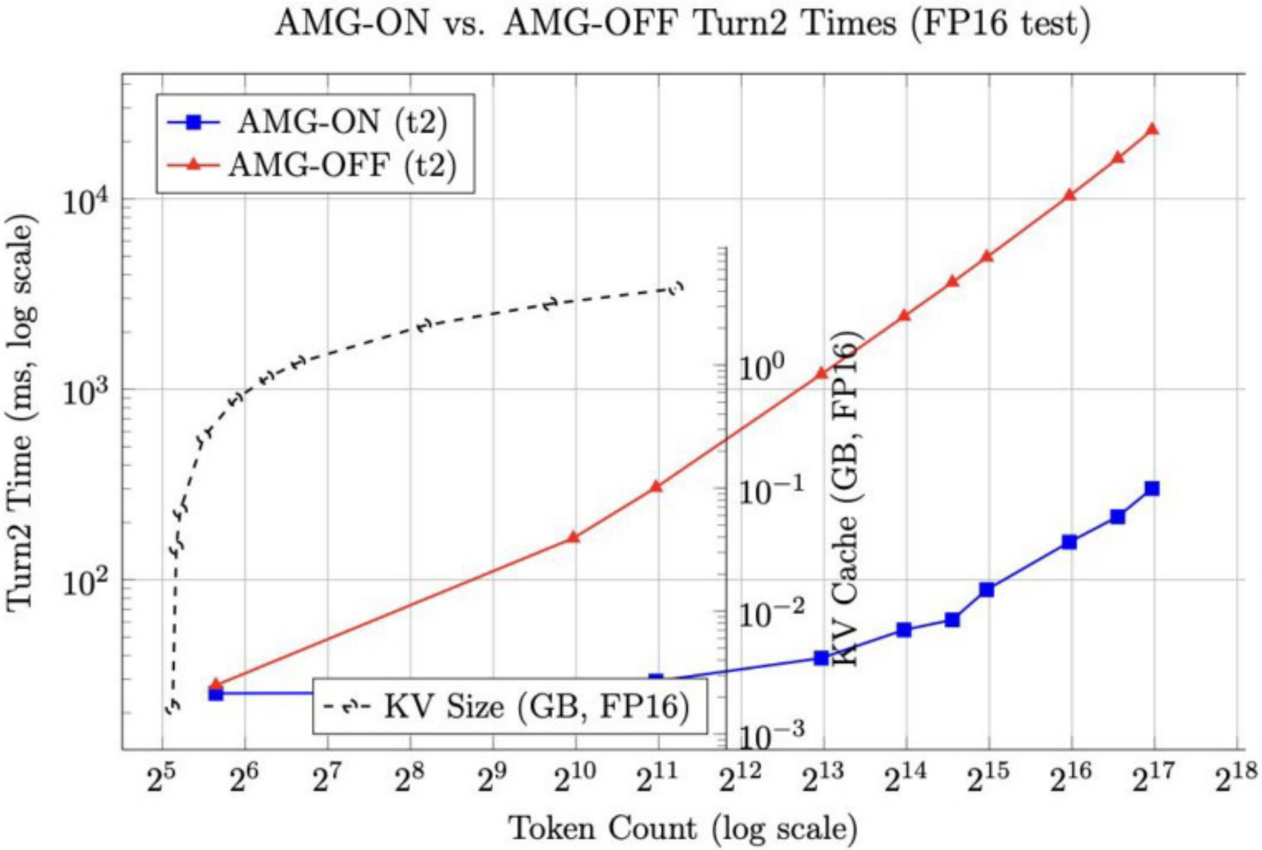


Milestones

- I wrote a blog post and introduced PAoR, CC-By-4.0 in *“**Lessons Learned Scaling LLM Training and Inference with Direct Memory Access**” in October 2024.*
- ***“Blueprint for Supercharging LLM Inference with Paged Attention over RDMA”*** ***NVIDIA, GPU Technology Conference (GTC) 2025***
- Wrote **”feature: KV Cache GPUDirect Storage” Pull Request 3209** in **TensorRT-LLM** (merged)
- Wrote **Python-Native-libCuFile** (PyPi packaged fork used in **LMCache**)
- PR 3209 used for **NVIDIA NIXL enablement** in Nvidia’s TensorRT-LLM inference server.

Public Benchmarks

- Public inference performance benchmark for LLM inference.
- Benchmark Measures:
  A. Time to First Token with **cache generation**.
  B. Time to First Token with **RDMA on page fault**.
- Paged Attention over RDMA page migration to NVMe using Nvidia cuFile userspace on NVIDIA DGX H100.
- Distributed Filesystem with GPUDirect Storage over RDMA running at 270GiB/second.
- **No Speculative Page Fetching.**
- **Use of cuFile sync APIs cuFileWrite/cuFileRead** due to FS interaction with cuFile Userspace blobs. NVIDIA stable API use of async/await cuFileWrite and cuFileRead requires asynchronous RDMA userspace IOCTL (userspace write kernel read)**.**



[Public Source] World Summit AI, June 2025:

https://4149027.fs1.hubspotusercontent-na1.net/hubfs/4149027/WSAIUSA25%20speaker%20ppts/TRACK%202/11_20%20-%2012_20%2c%20Arthur%20Rasmusson.pptx.pdf

[Public Source] New Industry Bar TensorRT-LLM blog post:

https://www.weka.io/blog/ai-ml/weka-sets-a-new-bar-with-75x-faster-time-to-first-token-ttft/

Tokens	AMG-ON t1	AMG-ON t2	AMG-OFF t1	AMG-OFF t2	% diff (t2)
50	36.894	25.328	29.081	27.925	10.25
1000	165.612	25.538	166.805	165.260	547.12
2000	305.454	29.350	304.692	304.856	938.69
8000	1189.103	38.787	1199.765	1199.311	2992.03
16000	2394.289	54.519	2410.260	2412.058	4324.28
24000	3612.110	61.598	3652.862	3648.305	5822.81
32000	4893.061	88.661	4931.505	4934.670	5465.75
64000	10288.430	157.749	10360.978	10369.098	6473.17
96000	16256.742	214.237	16304.672	16312.039	7514.03
128000	22981.840	301.481	22994.476	22998.583	7528.53

Table 1: **FP16 test, AMG-ON vs. AMG-OFF Turn2 Times.** Gains up to 7528.53% at 128k tokens.

[Public Source] World Summit AI, June 2025:

https://4149027.fs1.hubspotusercontent-na1.net/hubfs/4149027/WSAIUSA25%20speaker%20ppts/TRACK%202/11_20%20-%2012_20%2c%20Arthur%20Rasmusson.pptx.pdf

[Public Source] New Industry Bar TensorRT-LLM blog post:

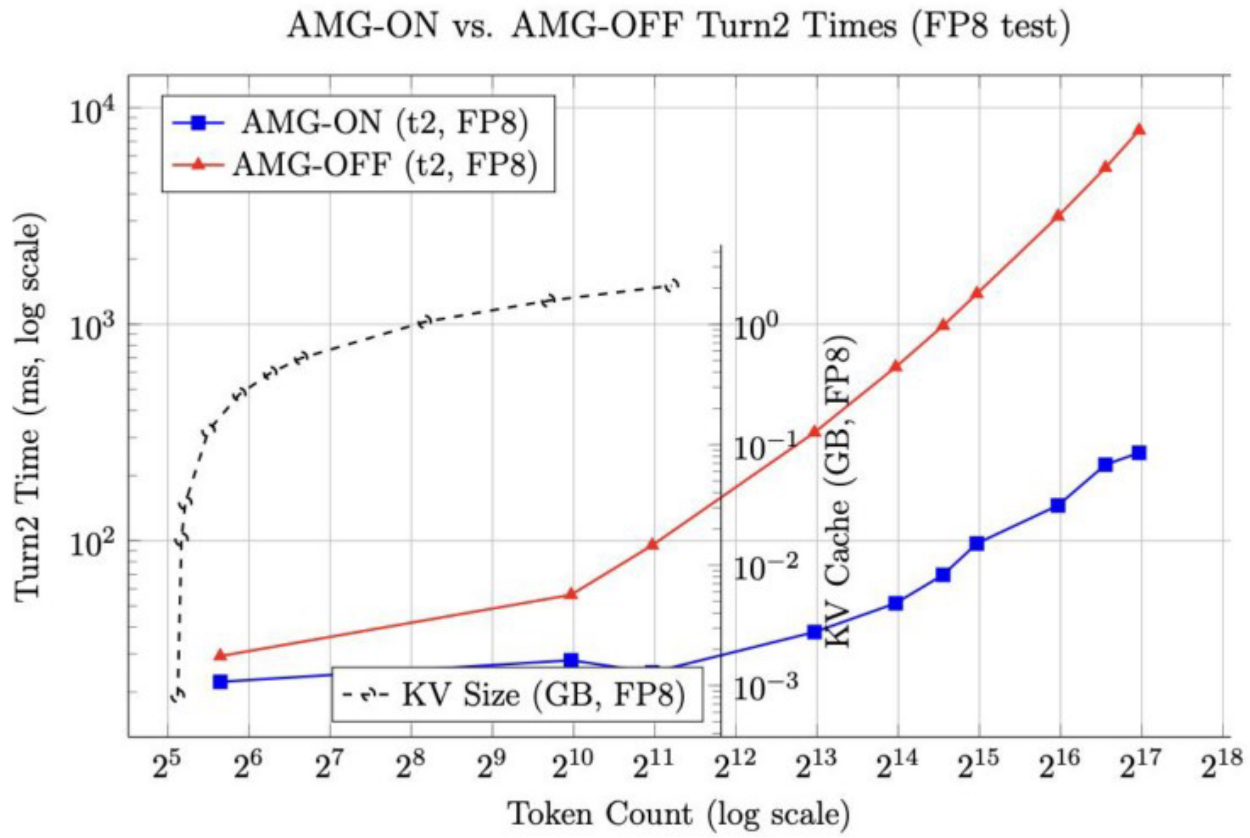
https://www.weka.io/blog/ai-ml/weka-sets-a-new-bar-with-75x-faster-time-to-first-token-ttft/

Paged Attention over RDMA:
TensorRT-LLM PR 3209: sync IOCTL

128,000 input tokens.
22981.840 ms TTFT for KV Cache generation.
301.481 ms TTFT on RDMA page fetch.

<u>**7,528.53% faster**</u> at FP16 at
in Llama-3.1-70B.

[Public Source] World Summit AI, June 2025:
https://4149027.fs1.hubspotusercontent-na1.net/hubfs/4149027/WSAIUSA25%20speaker%20ppts/TRACK%202/11_20%20-%2012_20%2c%20Arthur%20Rasmusson.pptx.pdf



[Public Source] World Summit AI, June 2025:

https://4149027.fs1.hubspotusercontent-na1.net/hubfs/4149027/WSAIUSA25%20speaker%20ppts/TRACK%202/11_20%20-%2012_20%2c%20Arthur%20Rasmusson.pptx.pdf

[Public Source] New Industry Bar TensorRT-LLM blog post:

https://www.weka.io/blog/ai-ml/weka-sets-a-new-bar-with-75x-faster-time-to-first-token-ttft/

Tokens	AMG-ON t1	AMG-ON t2	AMG-OFF t1	AMG-OFF t2	% diff (t2)
50	39.439	22.308	43.725	29.327	31.47
1000	58.850	28.046	56.905	56.246	100.55
2000	97.152	24.662	95.072	95.206	286.05
8000	317.714	37.862	316.145	316.258	735.30
16000	641.439	51.424	633.103	634.030	1132.96
24000	1007.591	69.586	984.308	983.485	1313.34
32000	1367.059	97.098	1371.541	1381.308	1322.60
64000	3156.745	145.569	3143.190	3149.996	2063.92
96000	5290.473	224.575	5277.509	5279.433	2250.86
128000	7860.608	254.781	7846.349	7859.110	2984.65

Table 2: **FP8 test, AMG-ON vs. AMG-OFF Turn2 Times.** Gains up to 2984.65% at 128k tokens.

[Public Source] World Summit AI, June 2025:

https://4149027.fs1.hubspotusercontent-na1.net/hubfs/4149027/WSAIUSA25%20speaker%20ppts/TRACK%202/11_20%20-%2012_20%2c%20Arthur%20Rasmusson.pptx.pdf

[Public Source] New Industry Bar TensorRT-LLM blog post:

https://www.weka.io/blog/ai-ml/weka-sets-a-new-bar-with-75x-faster-time-to-first-token-ttft/

Paged Attention over RDMA:
TensorRT-LLM PR 3209: sync IOCTL

128,000 input tokens.
7860.608 ms TTFT for KV Cache generation.
254.781 ms TTFT on RDMA page fetch.

2,984.65% faster at FP8 at
in Llama-3.1-70B.

[Public Source] World Summit AI, June 2025:
https://4149027.fs1.hubspotusercontent-na1.net/hubfs/4149027/WSAIUSA25%20speaker%20ppts/TRACK%202/11_20%20-%2012_20%2c%20Arthur%20Rasmusson.pptx.pdf

GPUs Utilized?
Are they **Effectively Utilized**?

- High GPU utilization?
- High GPU *effective utilization*?
- “Utilized GPUs” can be wasting cycles on redundancies like recalculating kv cache.
- Think in inference cluster-wide **total token throughput** or **maximum sequence length**, not utilization.

$37,000 per H100
10,000 GPU inference cluster.

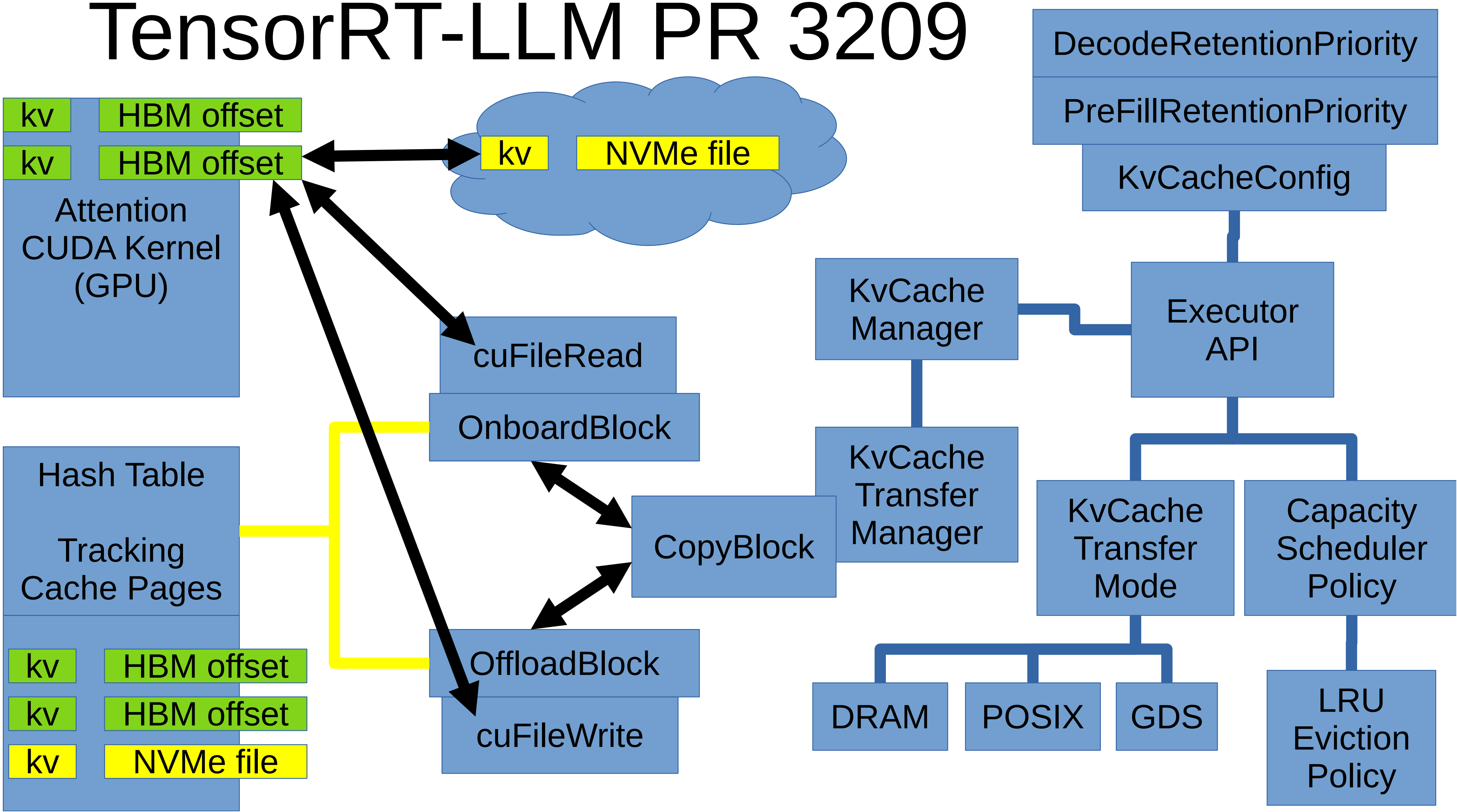
Roughly $370,000,000
(on GPUs alone).

$37,000 per H100
10,000 GPU inference cluster.

With 30% performance improvement generate the same total token throughput for artificial intelligence inference with **7,000 GPUs** as with **10,000 GPUs**.

30% performance increase **saves** **<u>$111,000,000</u>**

TensorRT-LLM PR 3209



Nvidia Dynamo Product

- Product for KV Cache over Fabric / **Paged Attention over RDMA**.
- Product for **Just-in-Time-Inference** (Auto-Scale Inference with RDMA)

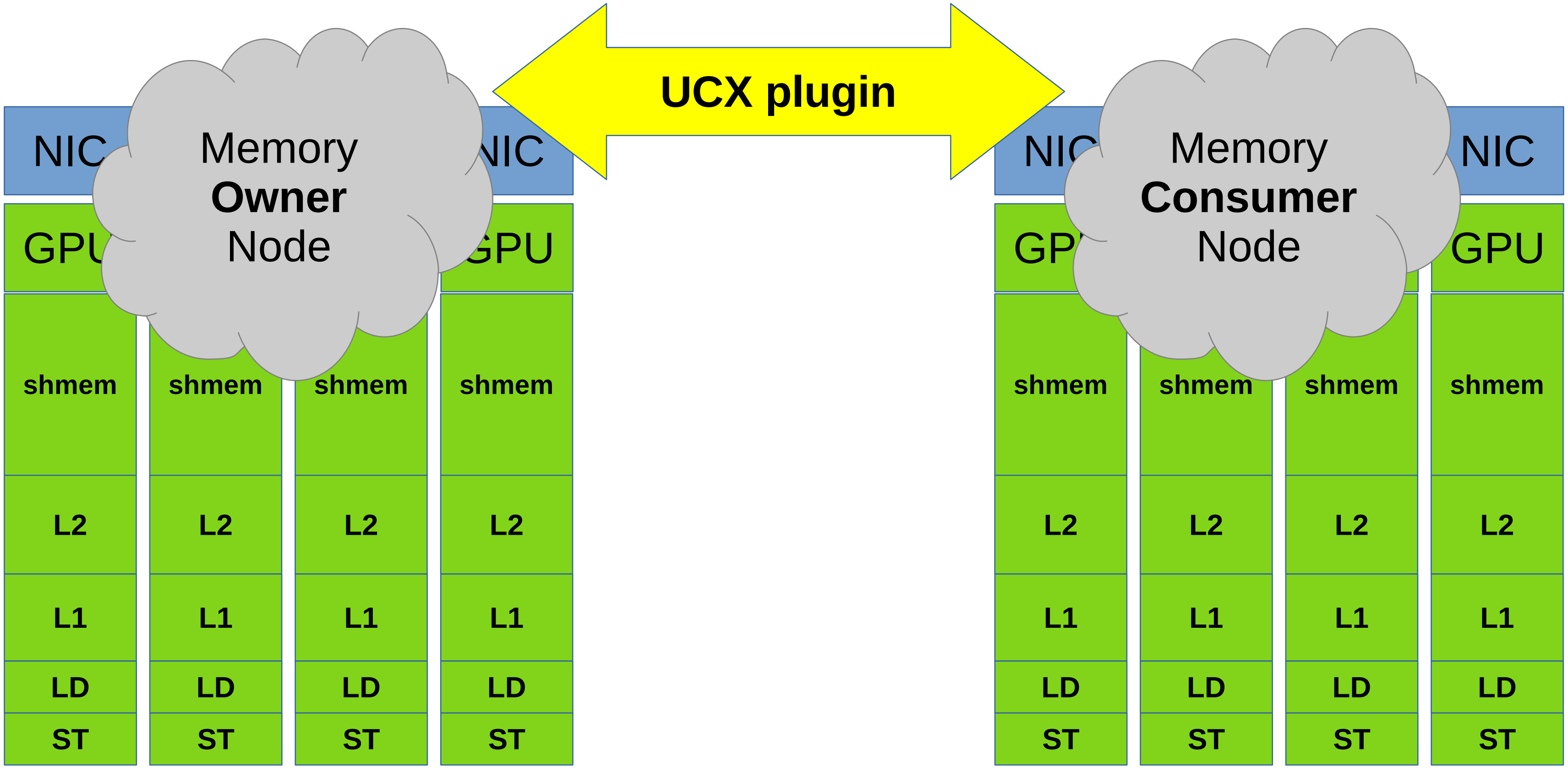
NIXL

- **Modular** plugin architecture for KV transfers.
- Supports **open source** and **proprietary** plugins for **different commercialization models** of KV Cache and Metadata transfers.
- **Removes the burden of support complexity for vendor specific diffs in** multiple inference servers (LMCache/vLLM, TensorRT-LLM, SG-Lang) shifted to NIXL core maintainers, **not plugin authors**.
- Vendors can focus on **speed & reliability of put and get operations** over complexity of maintaining diffs in multiple inference servers.
- Uses MPI driven distributed hash table instead of metadata files.

NIXL / Dynamo

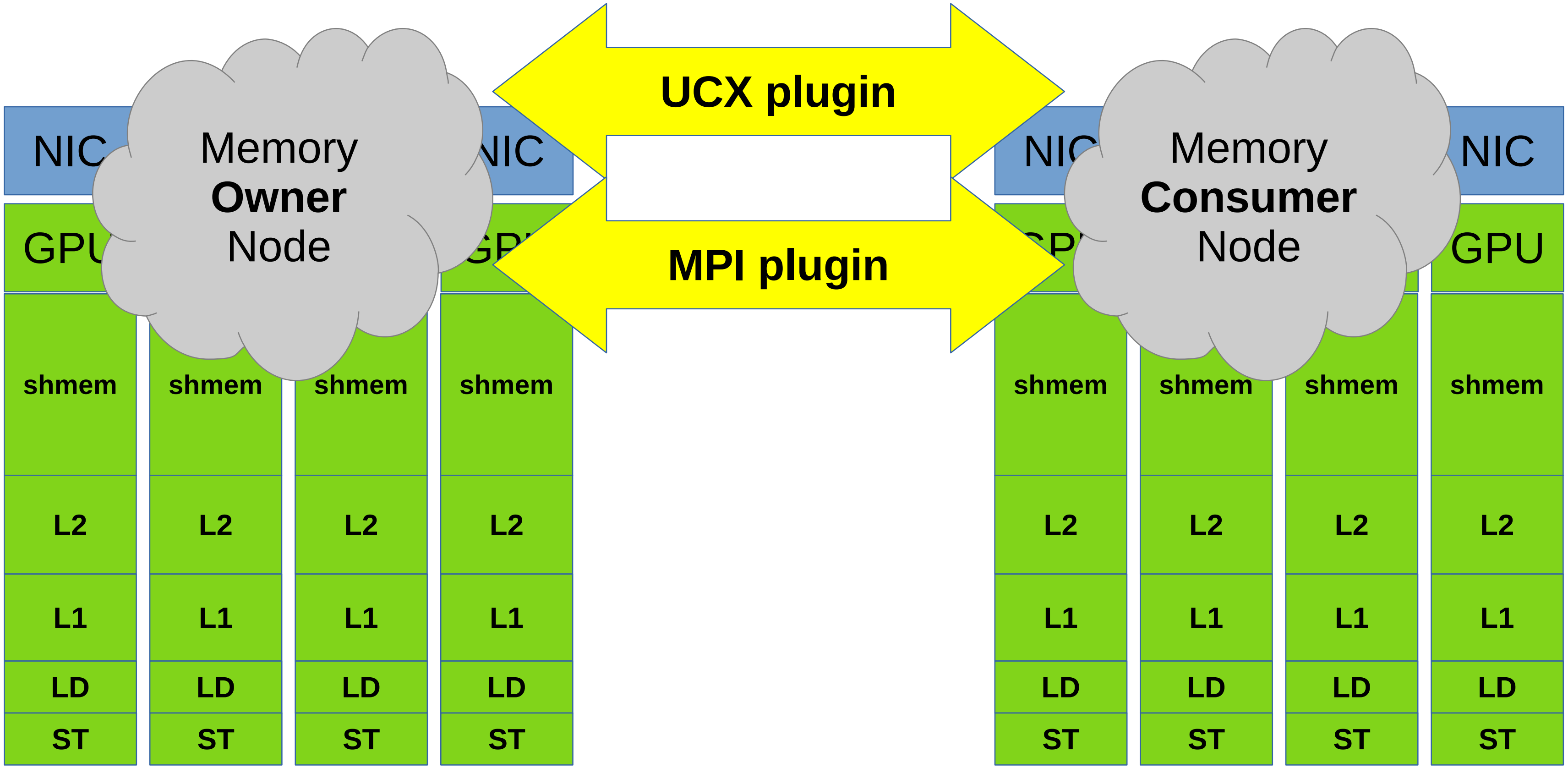
Memory
Owner
Node
Memory
Consumer
Node
NIC
NIC
NIC
NIC
GPU
GPU
GPU
GPU
shmem
shmem
shmem
shmem
shmem
shmem
shmem
shmem
L2
L1
LD
ST

NIXL / Dynamo

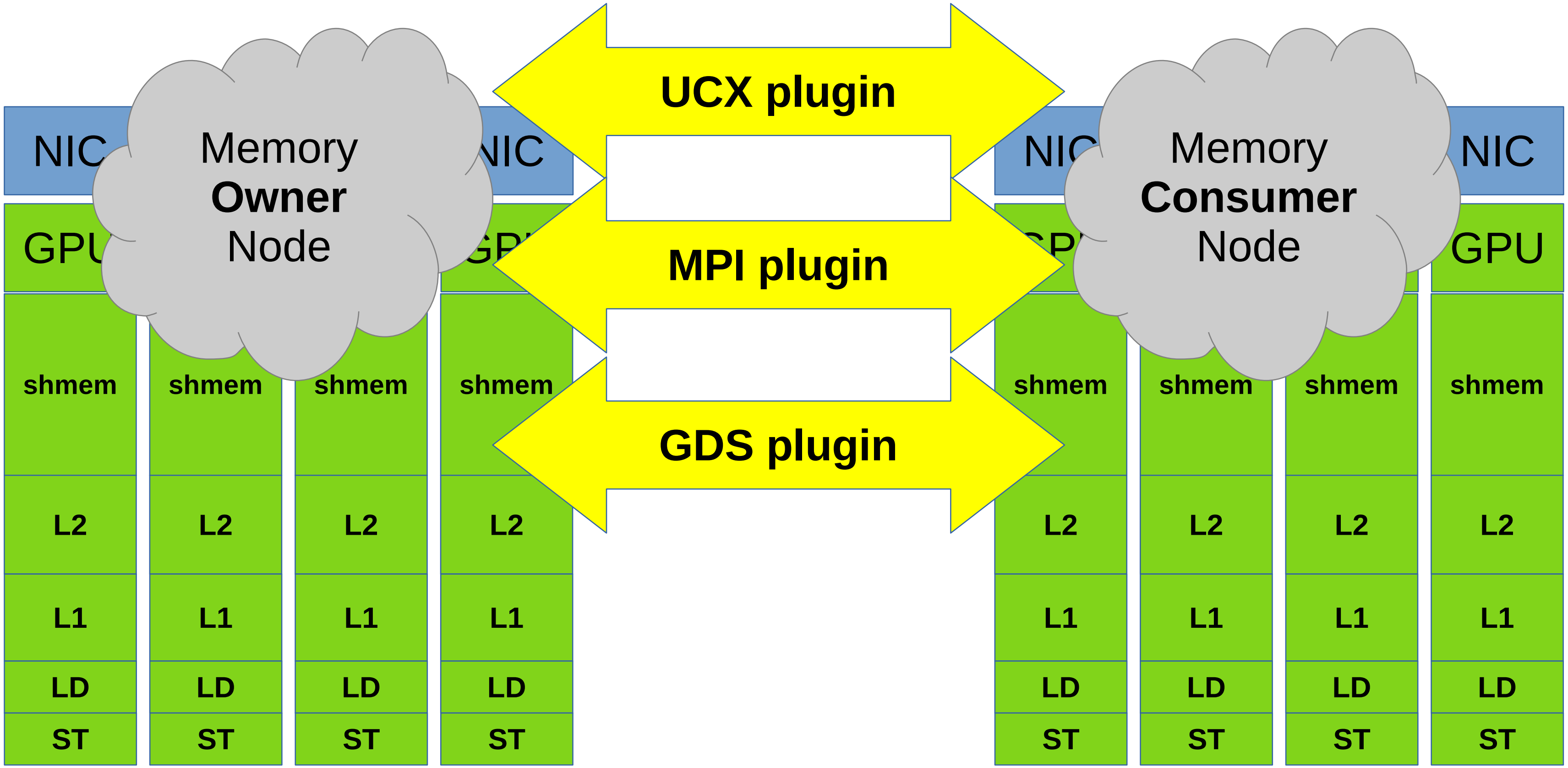


UCX plugin
Memory
Owner
Node
Memory
Consumer
Node
NIC
GPU
shmem
L2
L1
LD
ST

NIXL / Dynamo

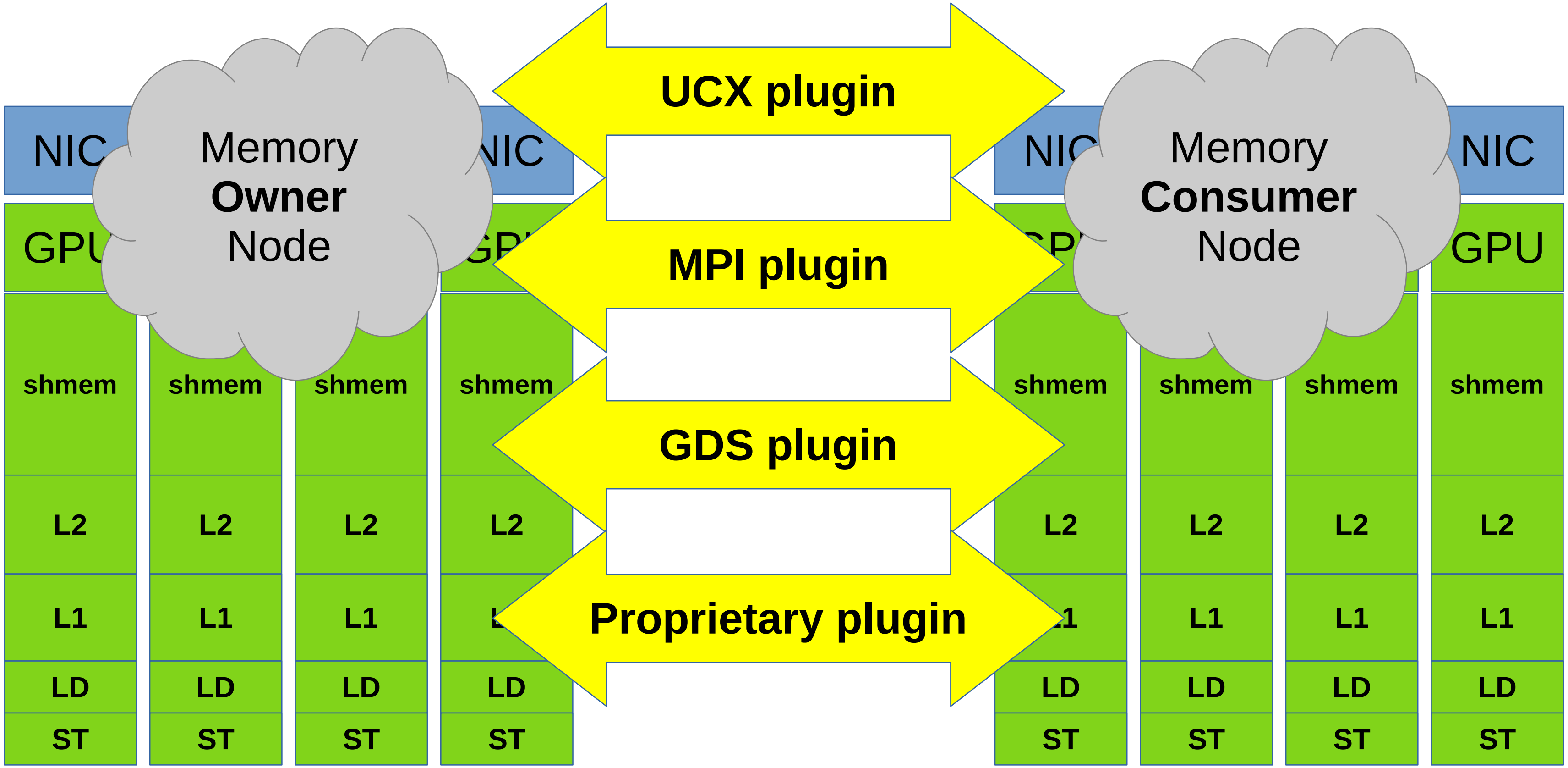

UCX plugin
MPI plugin
Memory
Owner
Node
Memory
Consumer
Node
NIC
GPU
shmem
L2
L1
LD
ST

NIXL / Dynamo

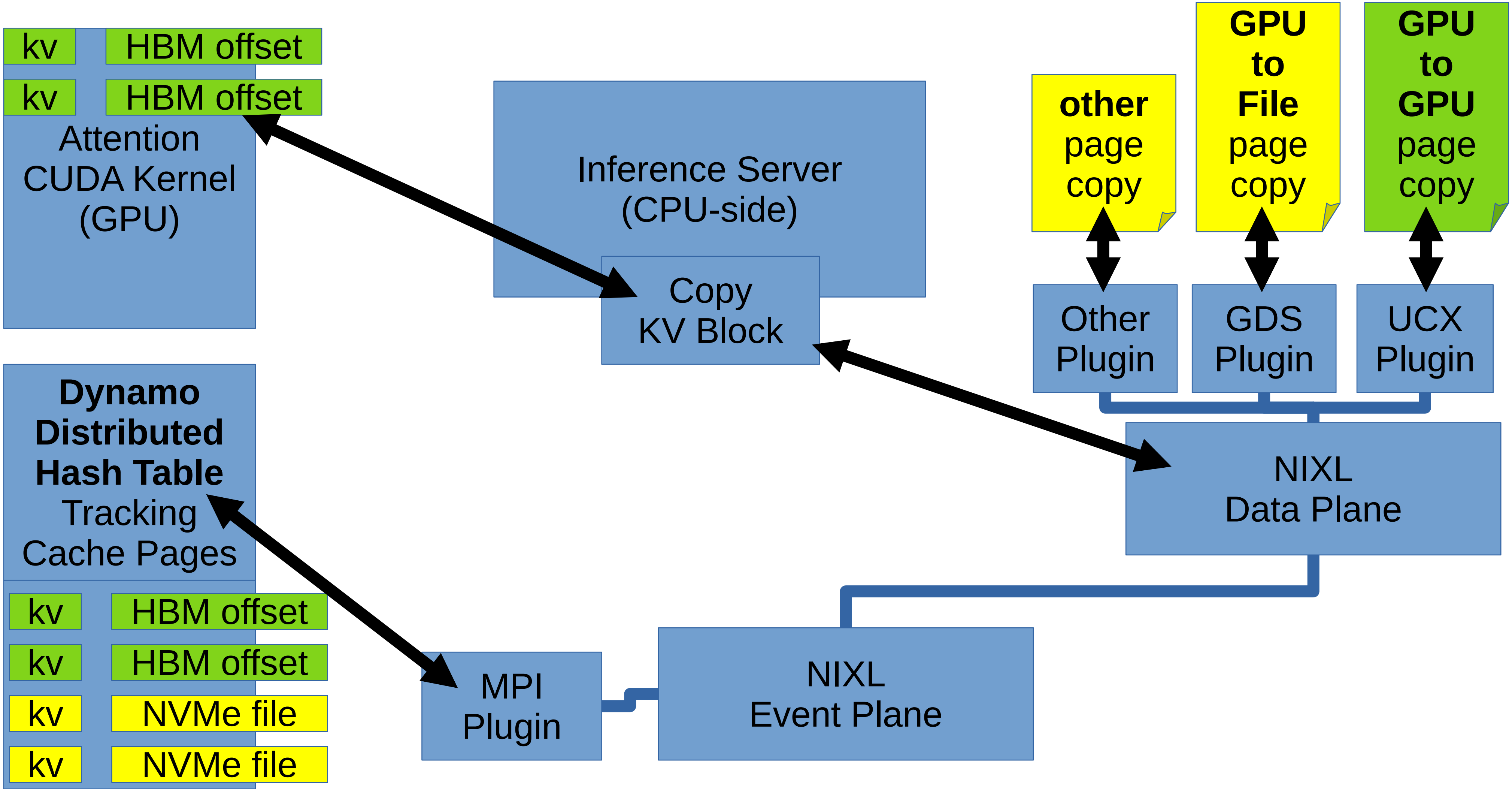


Memory
Owner
Node
Memory
Consumer
Node
UCX plugin
MPI plugin
GDS plugin
NIC
GPU
shmem
L2
L1
LD
ST

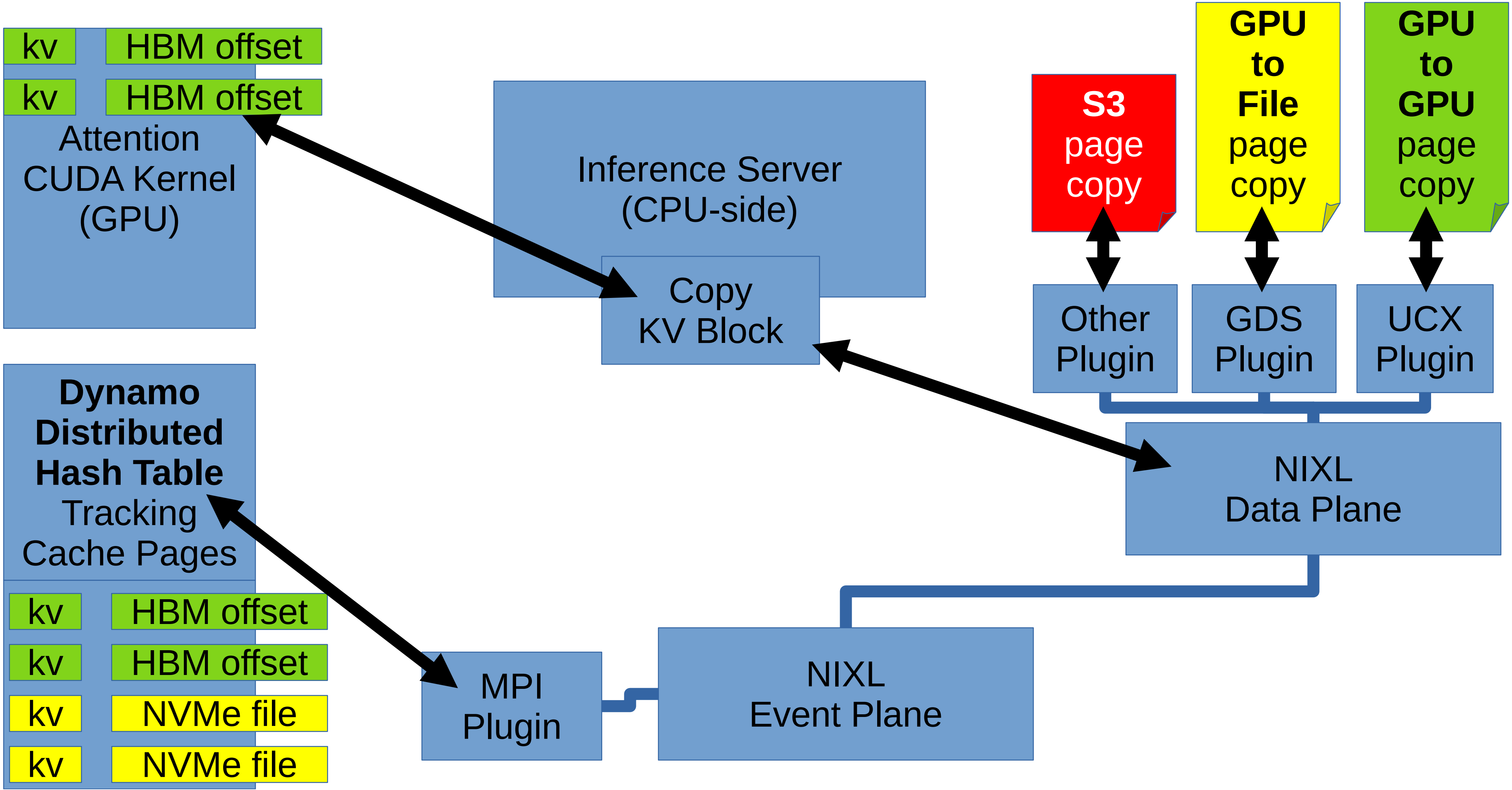
NIXL / Dynamo


Memory
Owner
Node
Memory
Consumer
Node
UCX plugin
MPI plugin
GDS plugin
Proprietary plugin
NIC
GPU
shmem
L2
L1
LD
ST

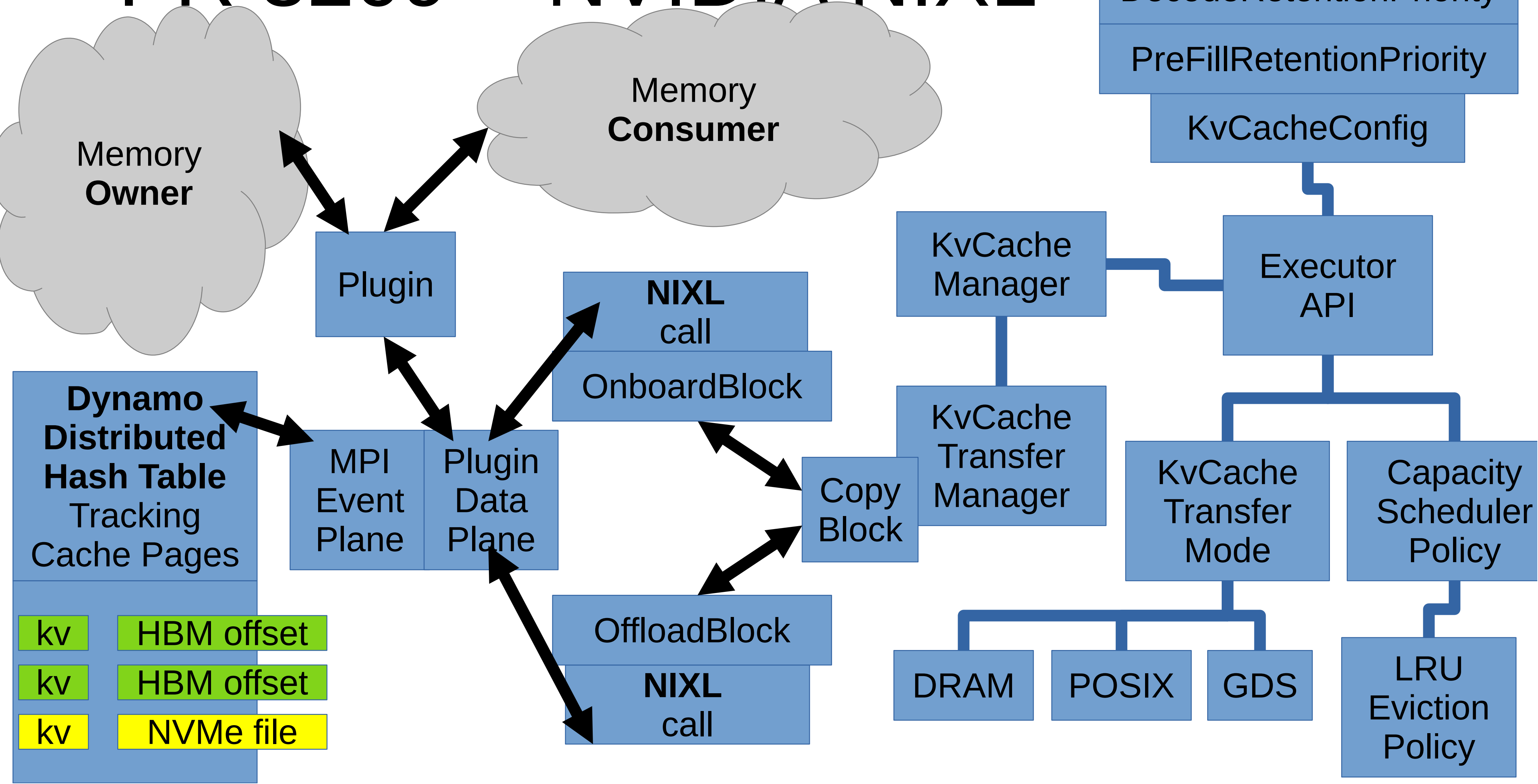
NIXL Architecture



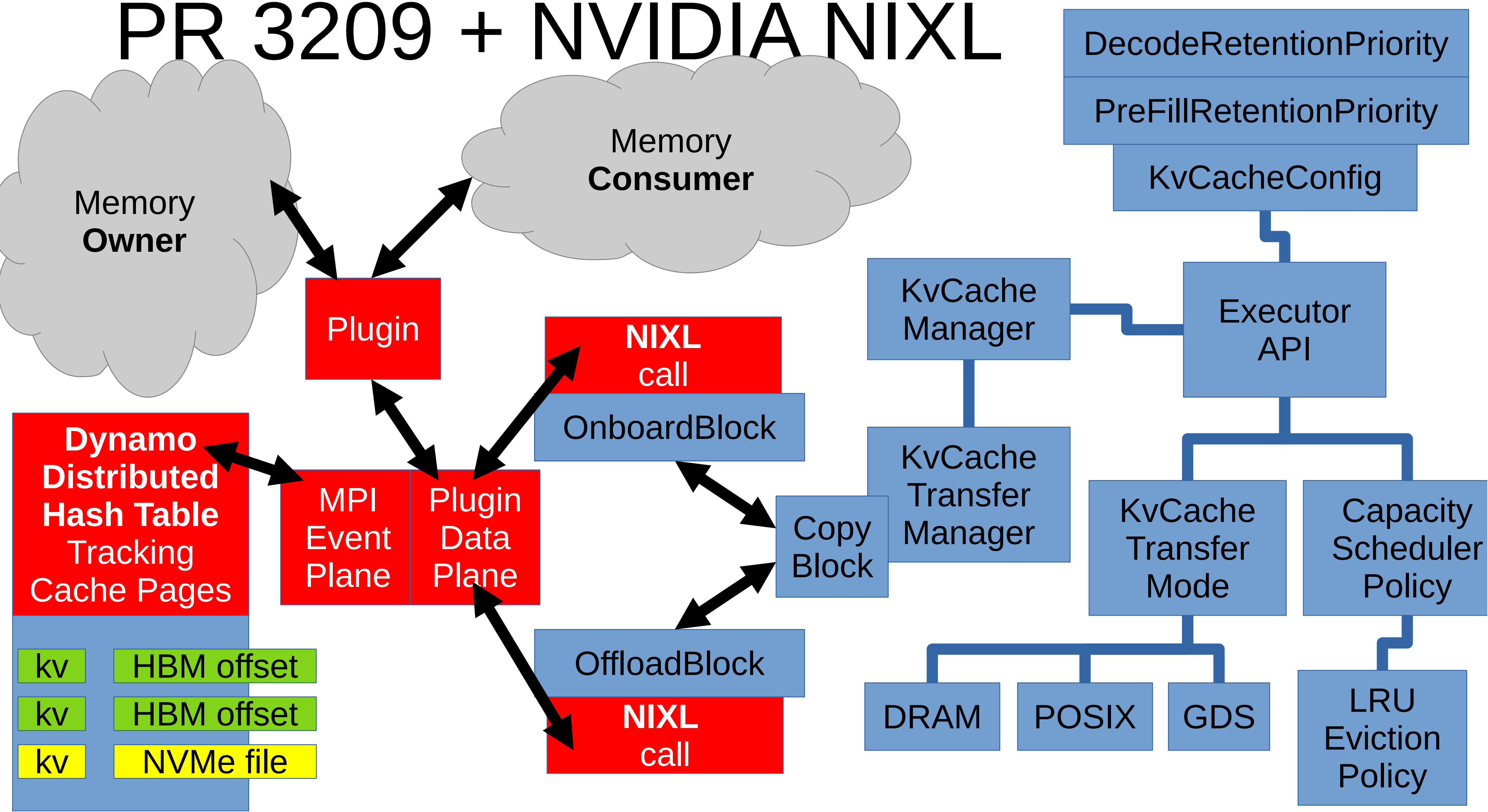
NIXL Architecture

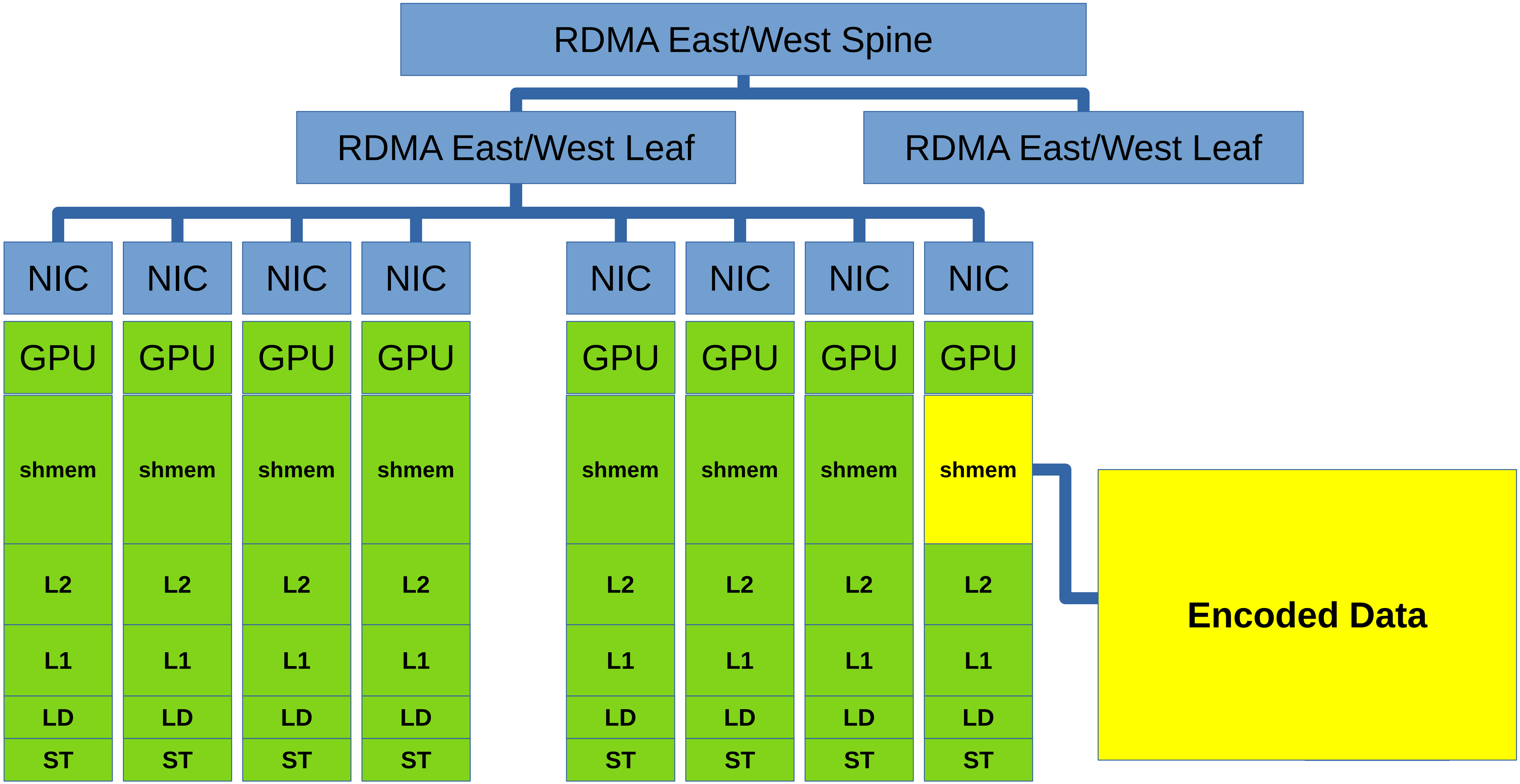


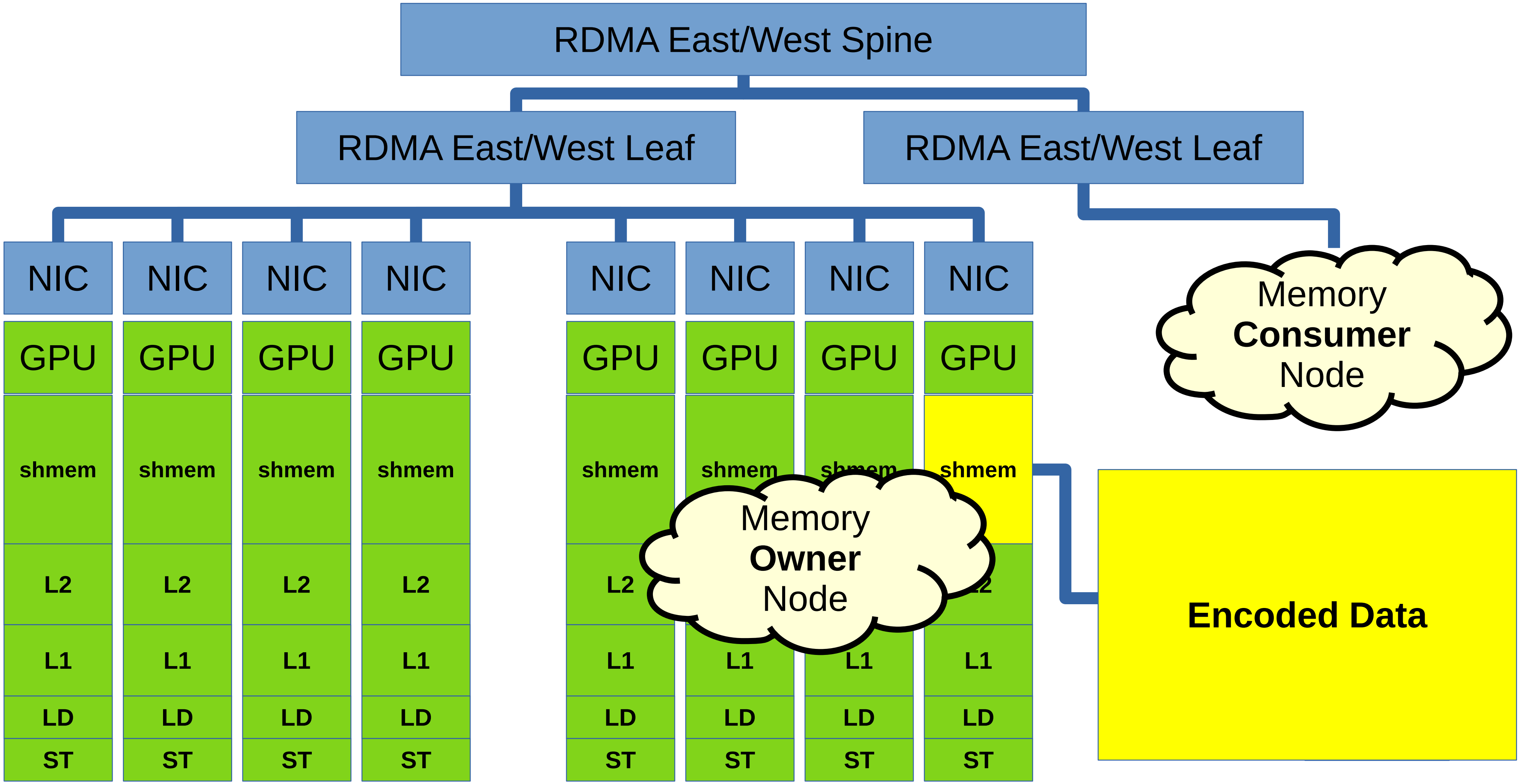
PR 3209 + NVIDIA NIXL



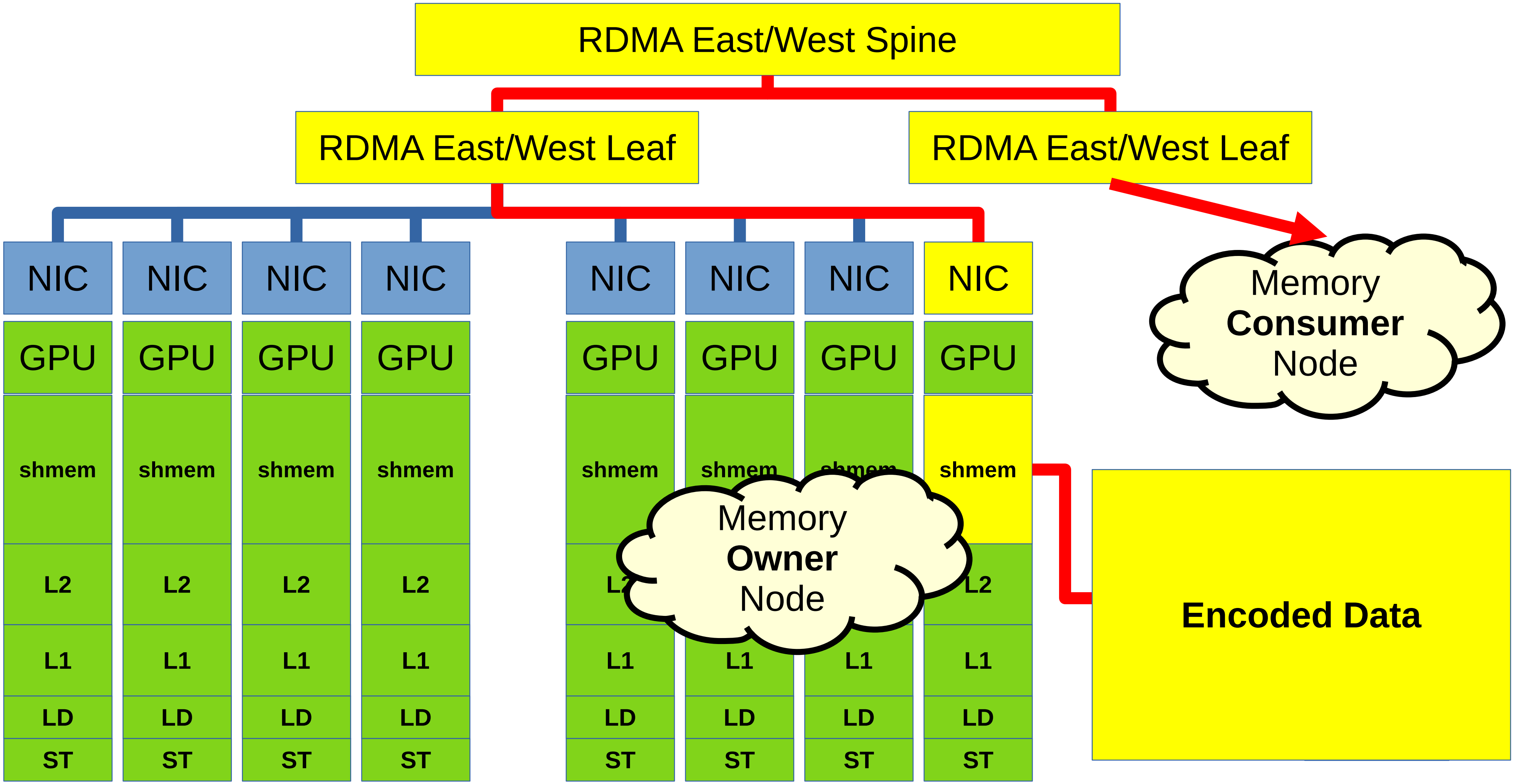
PR 3209 + NVIDIA NIXL




RDMA East/West Spine
RDMA East/West Leaf
RDMA East/West Leaf
NIC
GPU
shmem
L2
L1
LD
ST
Encoded Data


RDMA East/West Spine
RDMA East/West Leaf
RDMA East/West Leaf
NIC
GPU
shmem
L2
L1
LD
ST
Memory Consumer Node
Memory Owner Node
Encoded Data

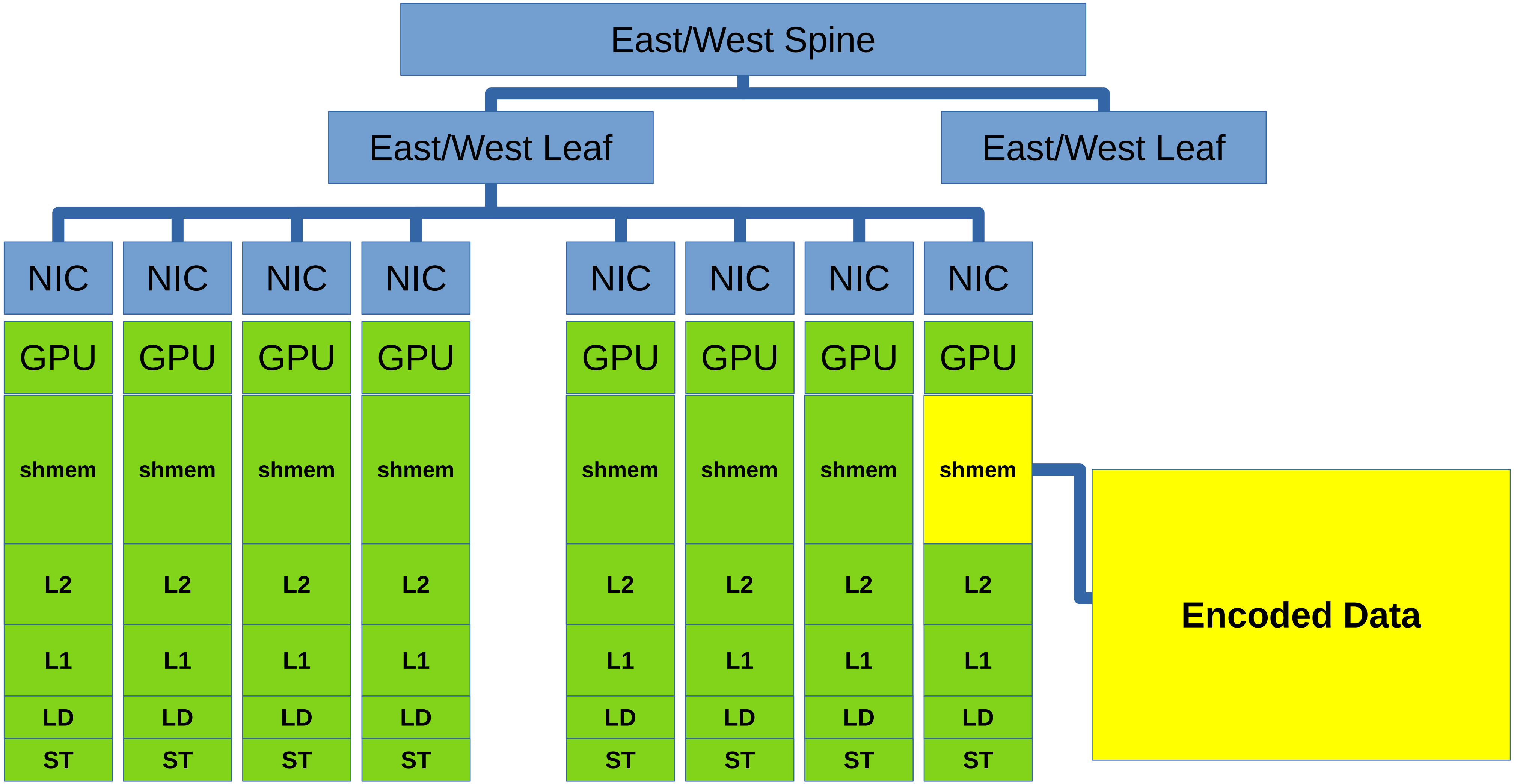
UCX or GDS Example



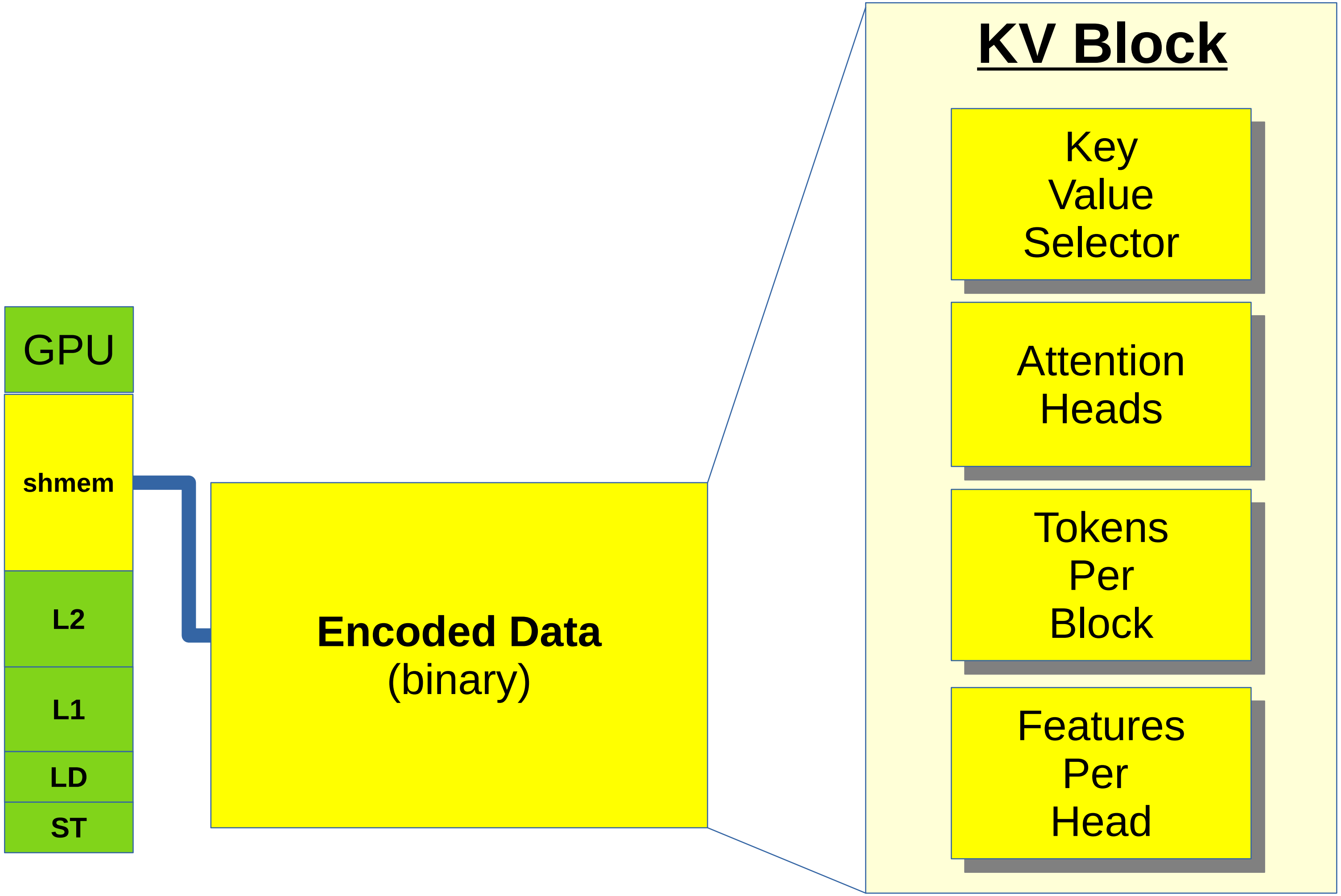
4

Cache Memory Structure

Lessons Learned Scaling LLMs with DMA



Data Structure



KV Block

- A **KV block** stores a chunk ("page") of attention **K**eys and **V**alues.
- Shape: 2, Hk, T, D
  - 2 → selects **K** or **V**
  - Hk → number of **KV heads**
  - T → tokens per block
  - D → **head dimension**
- One block packs **both** K and V for **the total number of KV** heads across the **total number of tokens per block**.

KV Cheat Sheet

Axis	**Meaning**	**Values / Notes**
0	K/V Selector	0 = K, 1 =V
1	KV Heads	Hk (attention heads under GQA/MQA)
2	Tokens in block	T (e.g. 16/32/64/128)
3	Hidden Dimensions	D (features per head)

Hex Map Binary Cache

Full Hex Map
(Lines 1008..1023 / 1023)

block_pool.bin in any memory tier.

kv_dims=[2,8,32,16], dtype=16-bit • Dry-run prompt: 'example audience prompt'

```
00003F00:  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00003F10:  10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
00003F20:  20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
00003F30:  30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
00003F40:  40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
00003F50:  50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
00003F60:  60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F
00003F70:  70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F
00003F80:  80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
00003F90:  90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
00003FA0:  A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
00003FB0:  B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF
00003FC0:  C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF
00003FD0:  D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF
00003FE0:  E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF
00003FF0:  F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
```

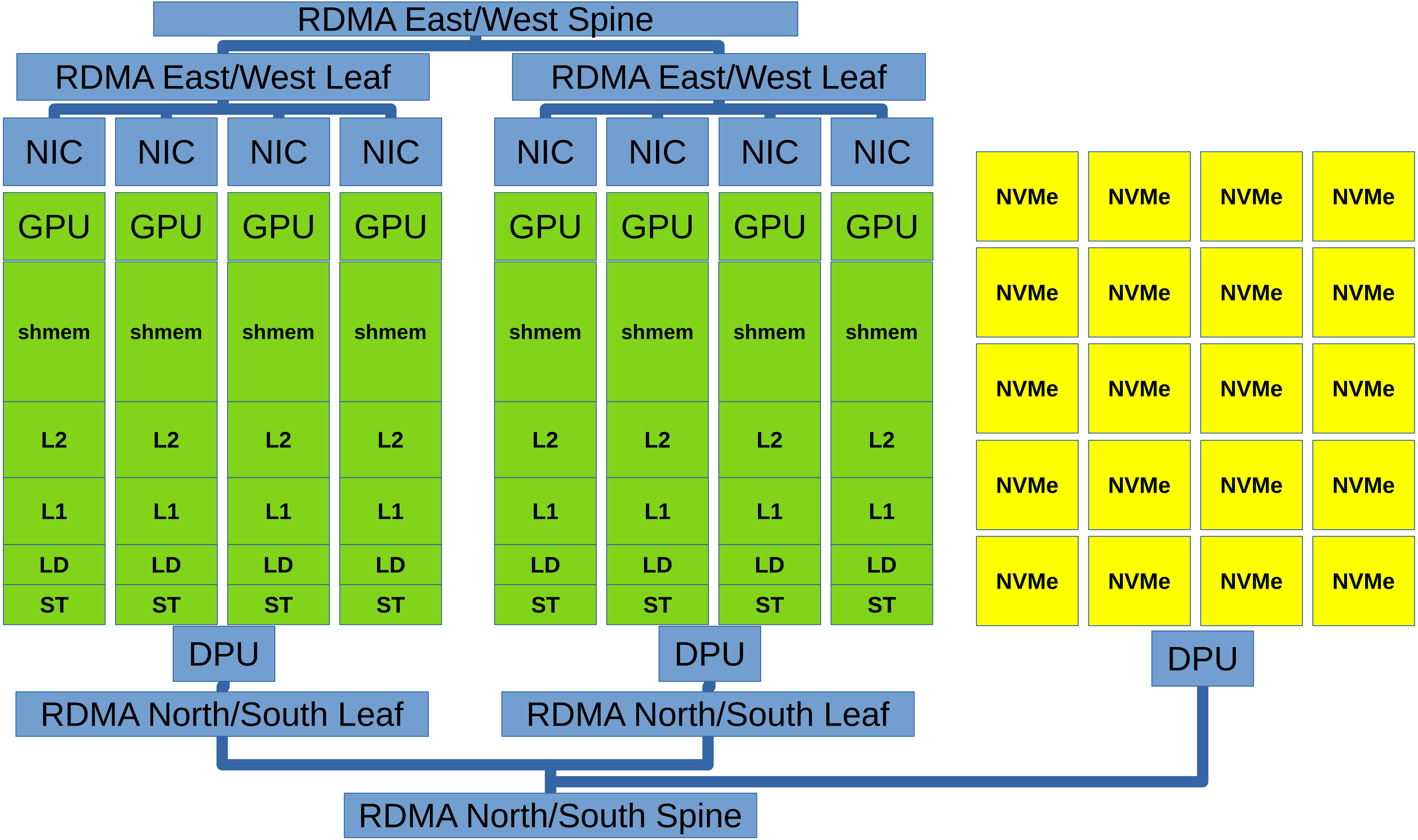
Full Hex Map (Lines 1008..1023 / 1023)

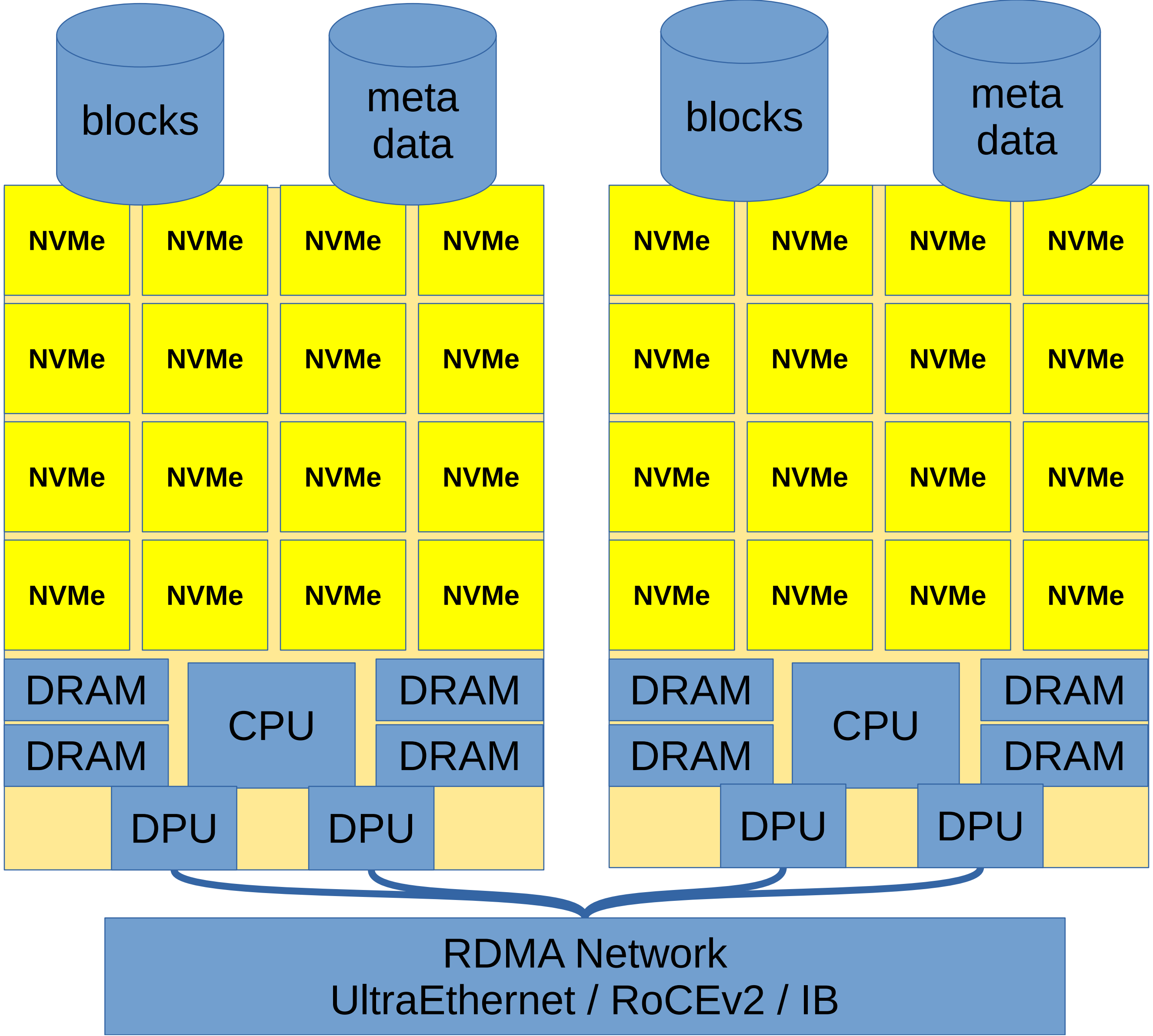
block_pool.bin in any memory tier.

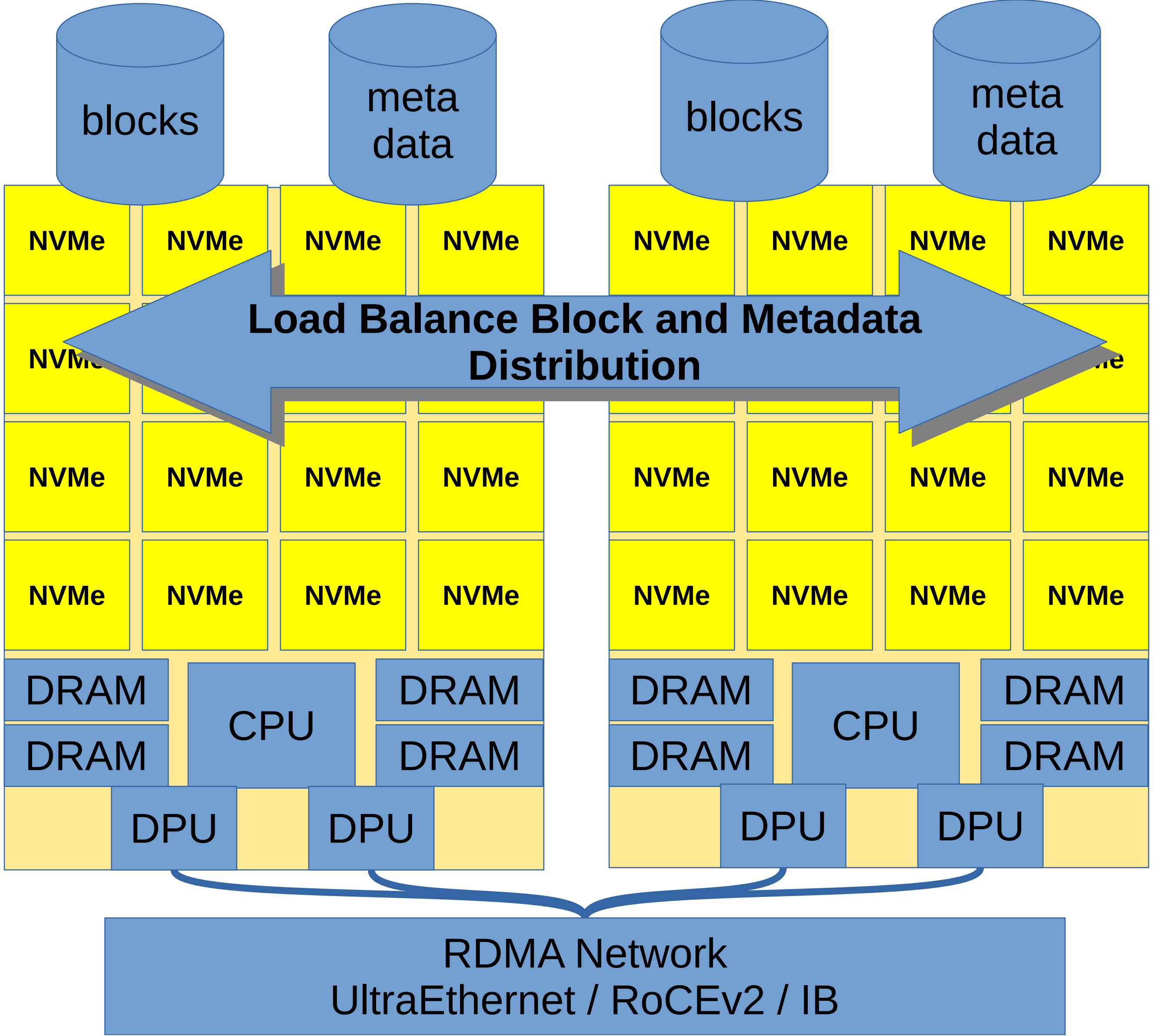
Labeled binary content.

```
Block mapping (per two lines)
00003F00..00003F1F  →  kv=V, h=7, t=24, d=0..7
00003F10..00003F1F  →  d=0..7
00003F20..00003F3F  →  kv=V, h=7, t=25, d=0..7
00003F30..00003F3F  →  d=0..7
00003F40..00003F5F  →  kv=V, h=7, t=26, d=0..7
00003F50..00003F5F  →  d=0..7
00003F60..00003F7F  →  kv=V, h=7, t=27, d=0..7
00003F70..00003F7F  →  d=0..7
00003F80..00003F9F  →  kv=V, h=7, t=28, d=0..7
00003F90..00003F9F  →  d=0..7
00003FA0..00003FBF  →  kv=V, h=7, t=29, d=0..7
00003FB0..00003FBF  →  d=0..7
00003FC0..00003FDF  →  kv=V, h=7, t=30, d=0..7
00003FD0..00003FDF  →  d=0..7
00003FE0..00003FFF  →  kv=V, h=7, t=31, d=0..7
00003FF0..00003FFF  →  d=0..7
```

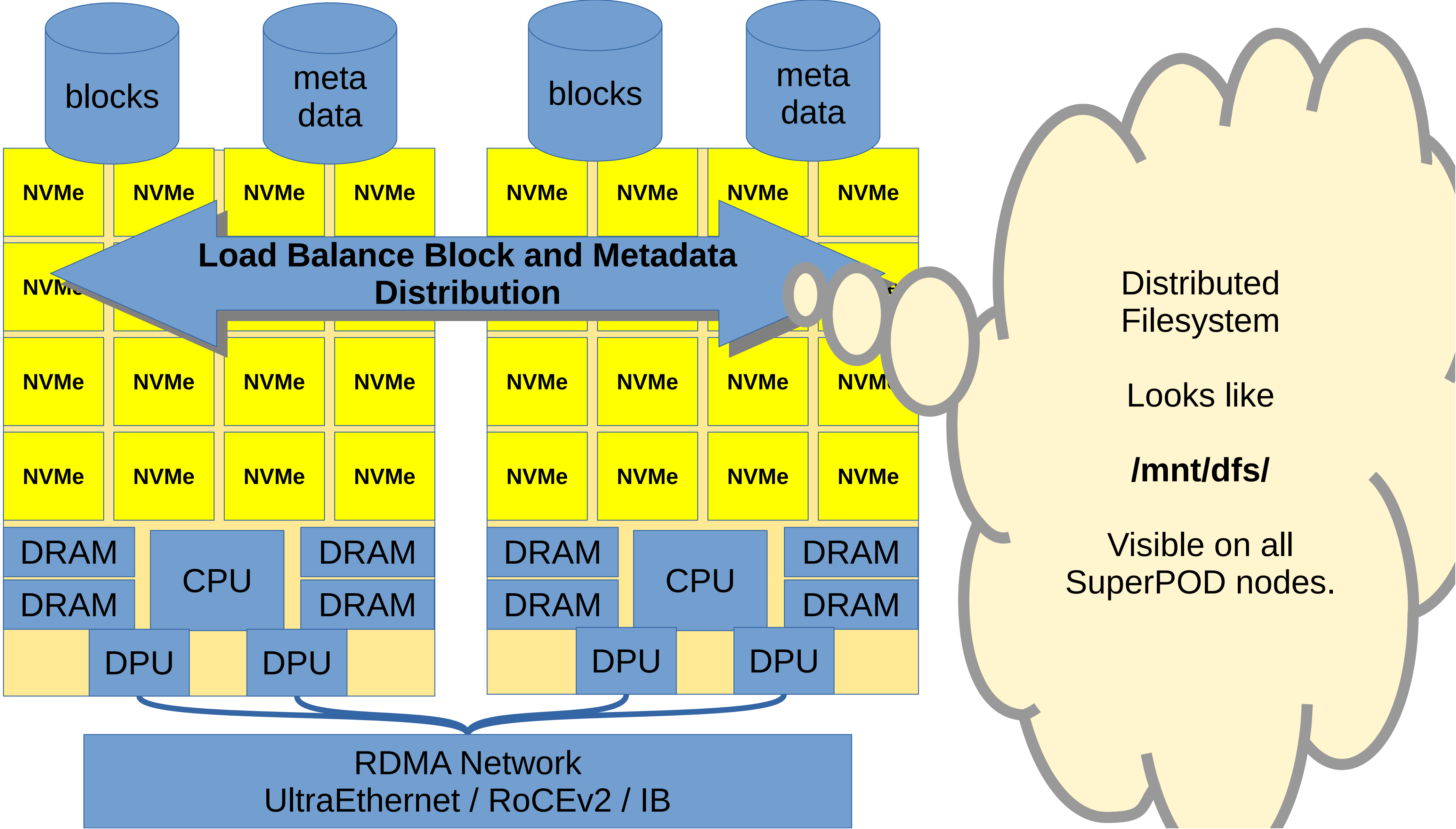
kv_dims=[2,8,32,16], dtype=16-bit • Dry-run prompt: 'example audience prompt'

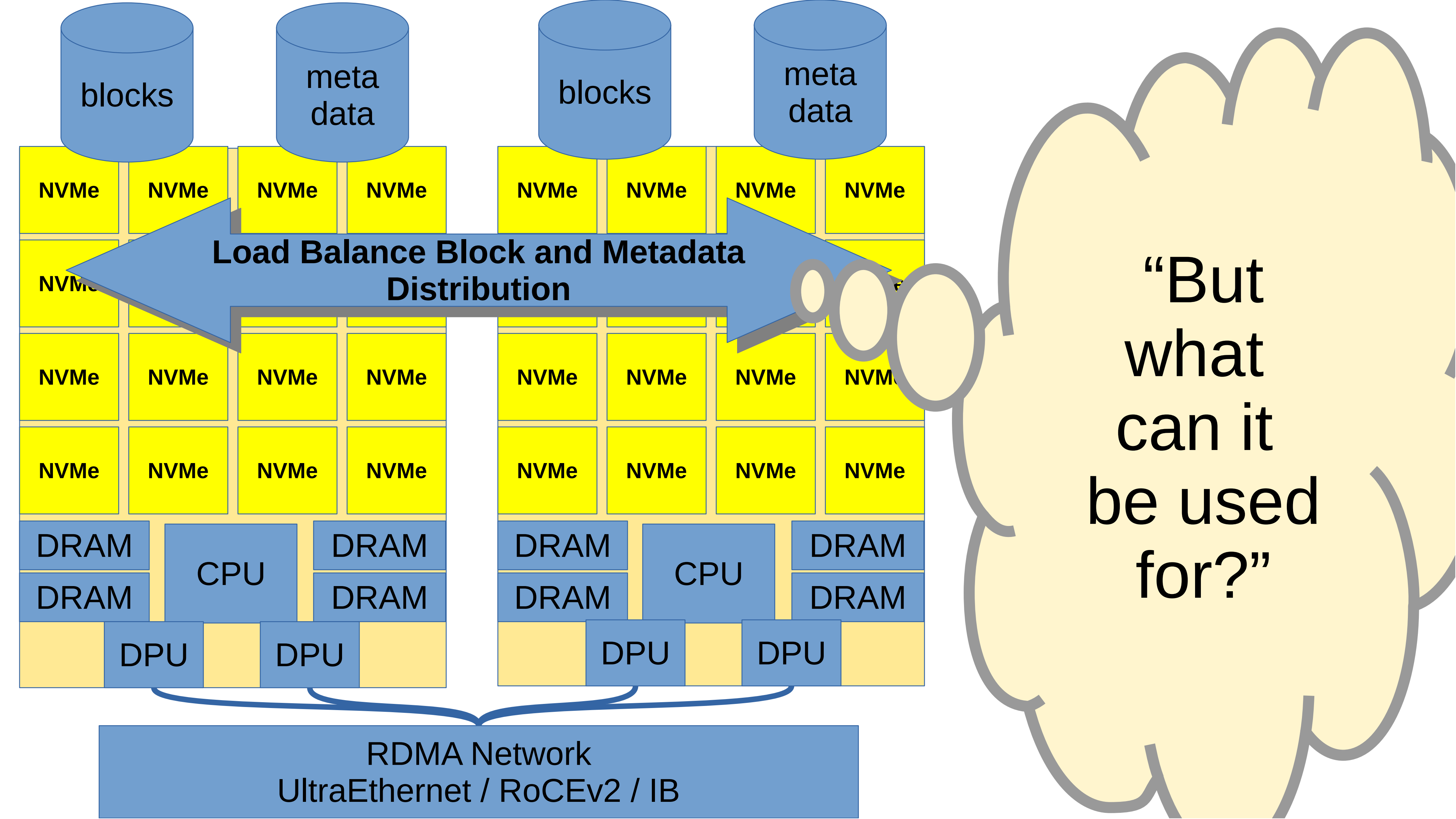

RDMA East/West Spine
RDMA East/West Leaf
RDMA East/West Leaf
NIC
GPU
shmem
L2
L1
LD
ST
NVMe
DPU
DPU
DPU
RDMA North/South Leaf
RDMA North/South Leaf
RDMA North/South Spine


blocks
meta data
blocks
meta data
NVMe
DRAM
CPU
DRAM
DPU
DPU
DRAM
CPU
DRAM
DPU
DPU
RDMA Network
UltraEthernet / RoCEv2 / IB

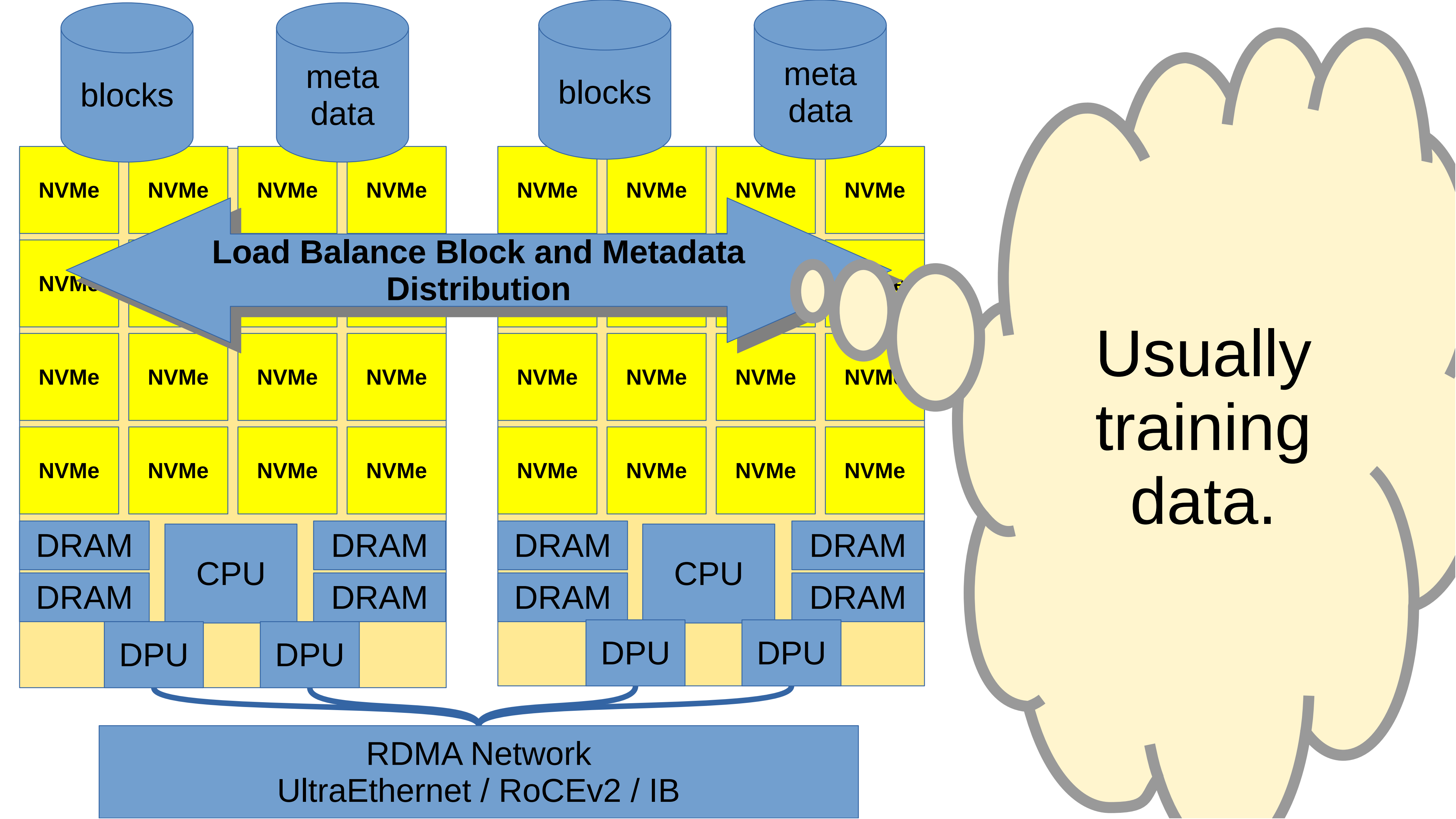


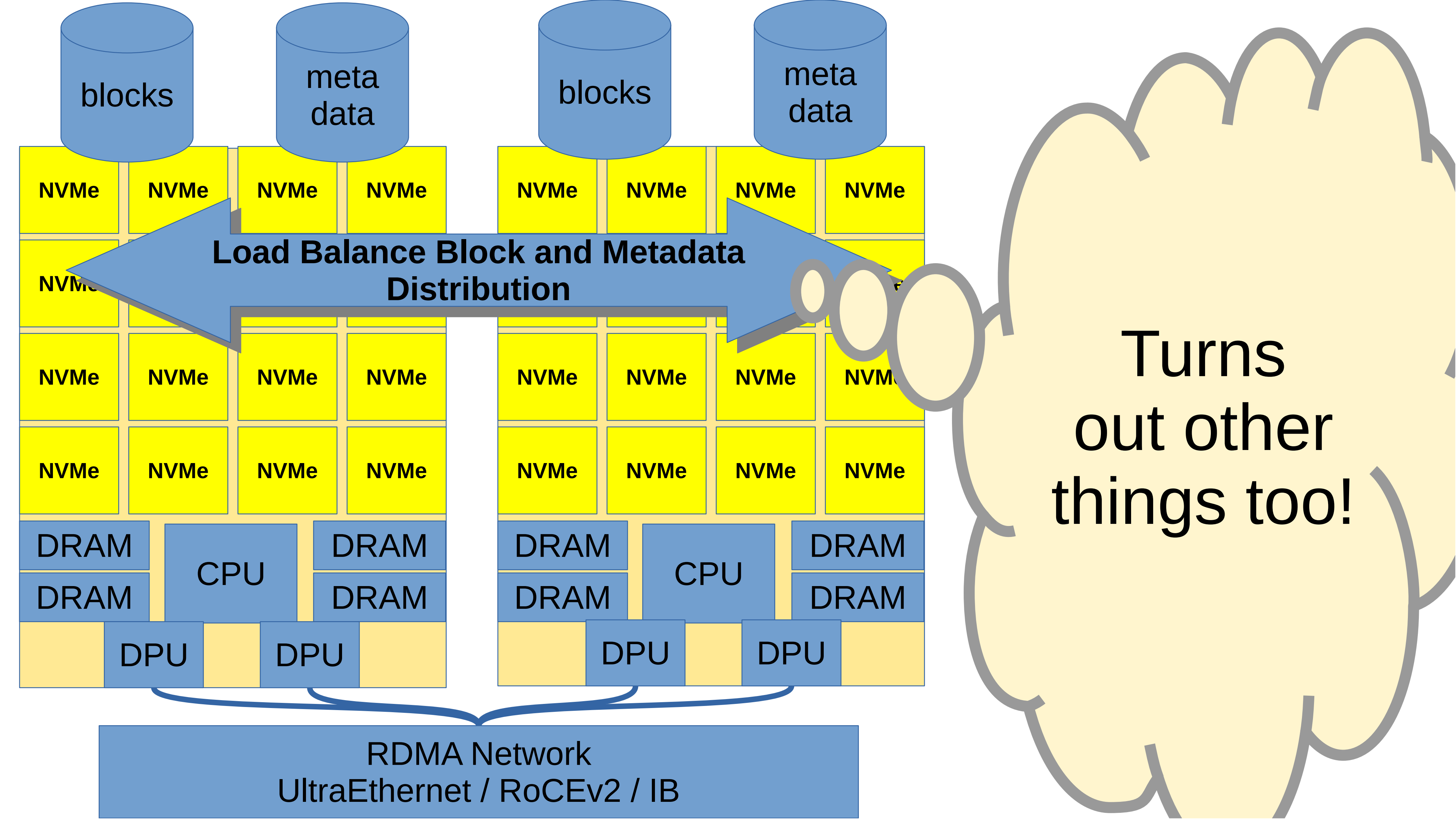
blocks
meta data
blocks
meta data
NVMe
Load Balance Block and Metadata Distribution
DRAM
CPU
DPU
RDMA Network
UltraEthernet / RoCEv2 / IB

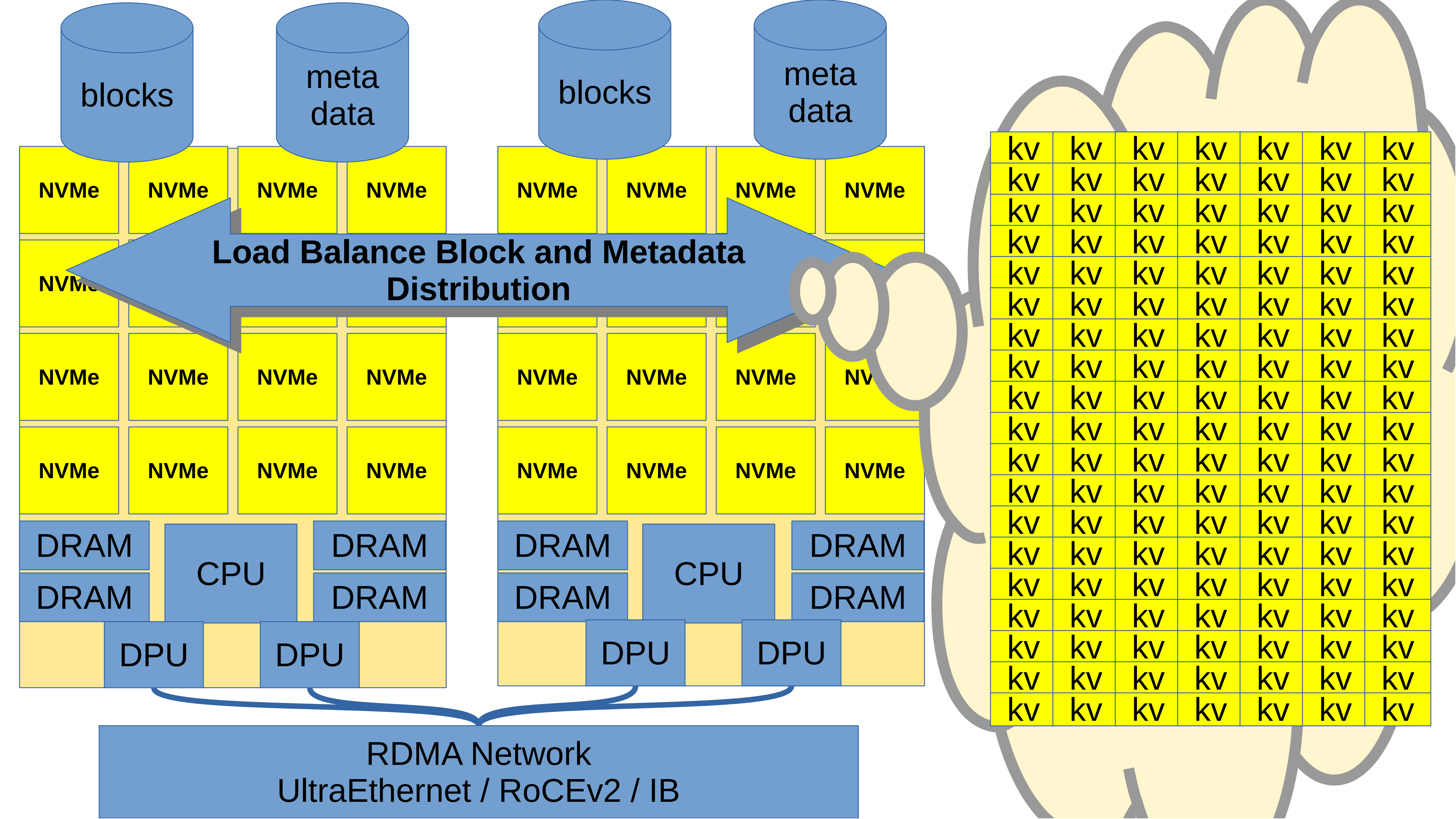

blocks
meta data
blocks
meta data
NVMe
Load Balance Block and Metadata Distribution
DRAM
CPU
DPU
RDMA Network
UltraEthernet / RoCEv2 / IB
Distributed Filesystem
Looks like
/mnt/dfs/
Visible on all SuperPOD nodes.



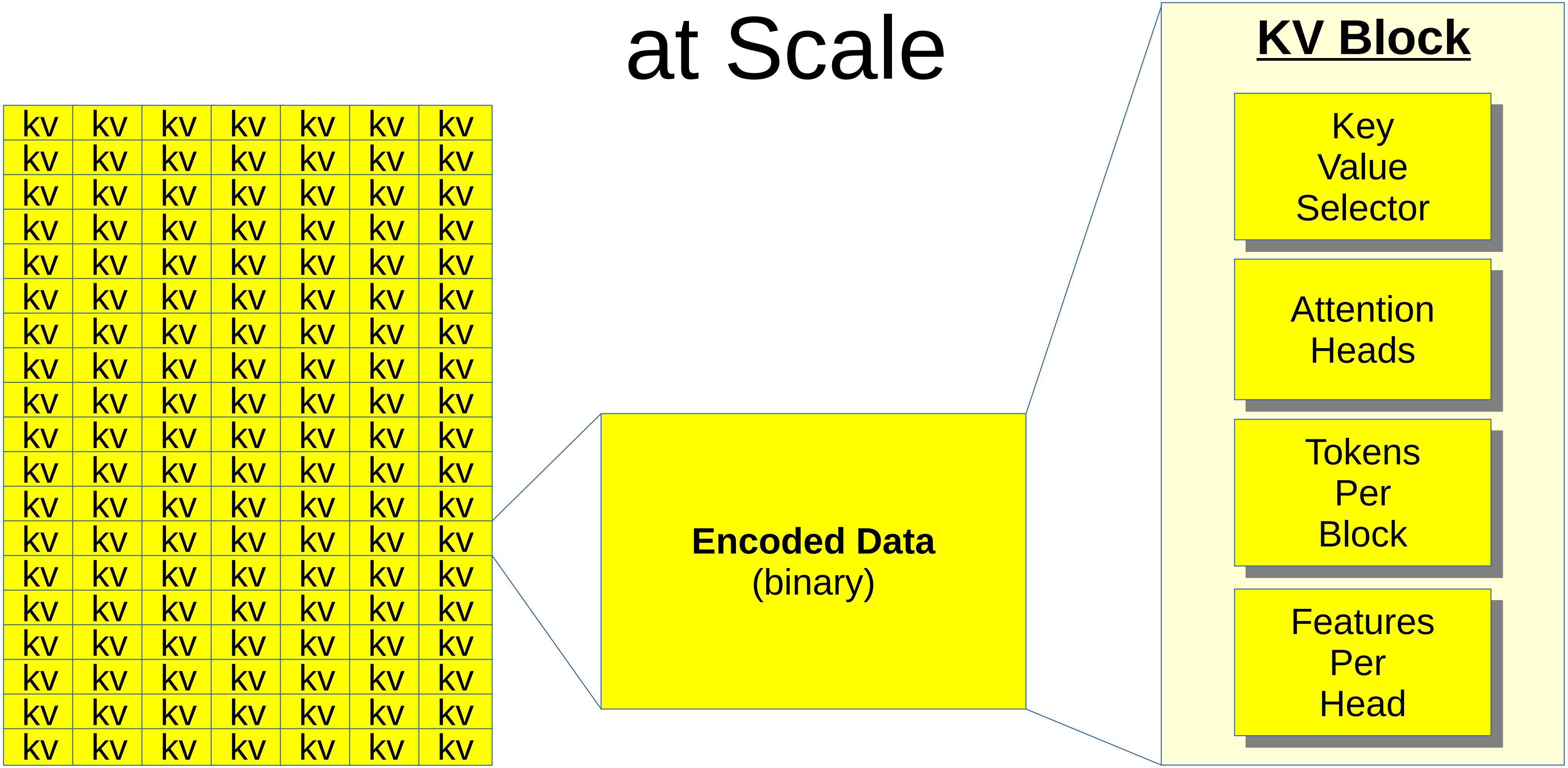
blocks
meta data
blocks
meta data
NVMe
Load Balance Block and Metadata Distribution
DRAM
CPU
DPU
RDMA Network
UltraEthernet / RoCEv2 / IB
“But what can it be used for?”

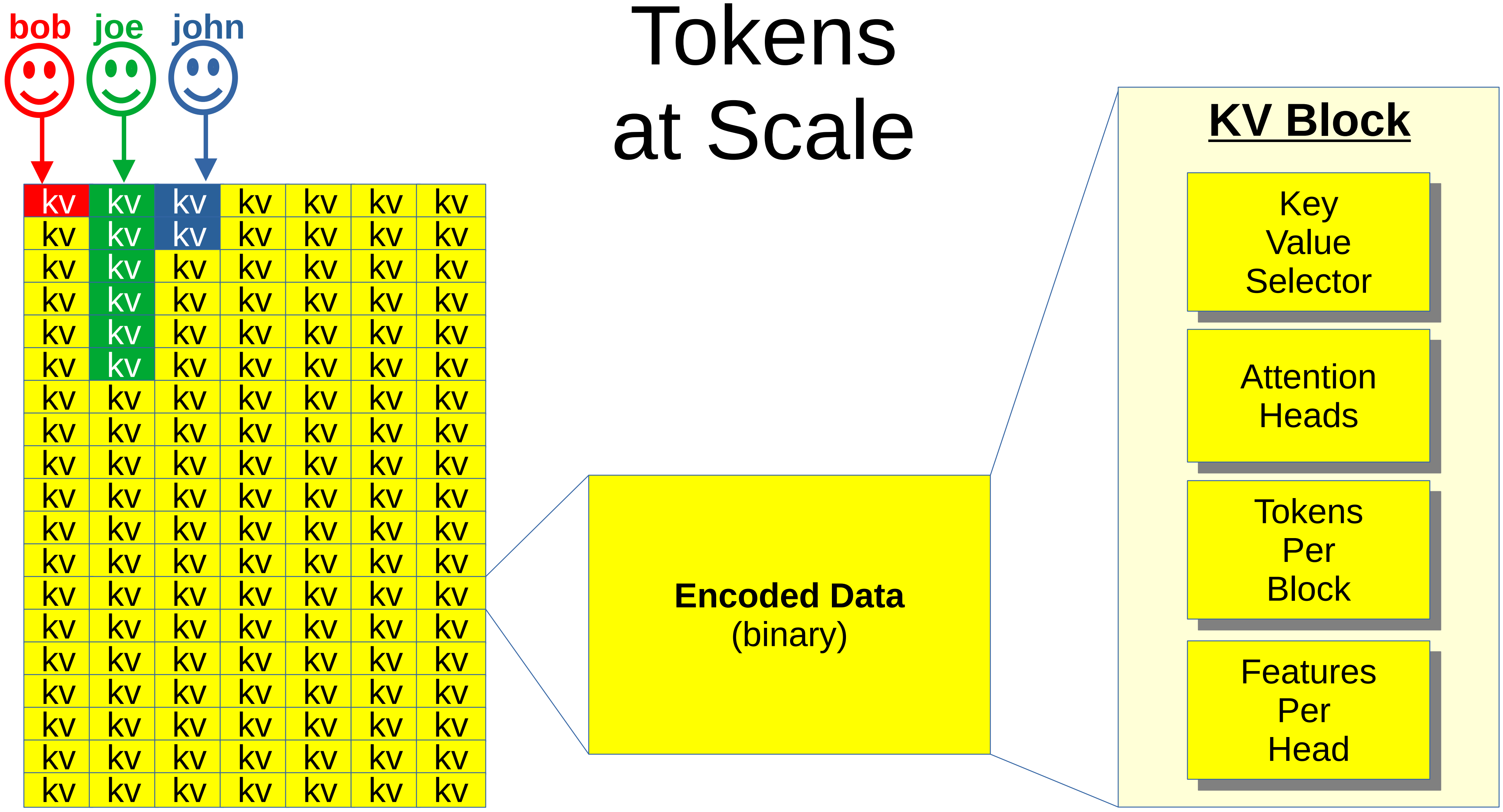

blocks
meta data
blocks
meta data
NVMe
NVMe
NVMe
NVMe
NVMe
NVMe
NVMe
NVMe
Load Balance Block and Metadata Distribution
NVMe
NVMe
NVMe
NVMe
NVMe
NVMe
NVMe
NVMe
NVMe
NVMe
NVMe
NVMe
NVMe
NVMe
NVMe
NVMe
NVMe
NVMe
DRAM
DRAM
CPU
DRAM
DRAM
DRAM
DRAM
CPU
DRAM
DRAM
DPU
DPU
DPU
DPU
RDMA Network
UltraEthernet / RoCEv2 / IB
Usually training data.


blocks
meta data
blocks
meta data
NVMe
Load Balance Block and Metadata Distribution
DRAM
CPU
DPU
RDMA Network
UltraEthernet / RoCEv2 / IB
Turns out other things too!


blocks
meta data
blocks
meta data
NVMe
Load Balance Block and Metadata Distribution
DRAM
CPU
DRAM
DPU
DPU
DRAM
CPU
DRAM
DPU
DPU
RDMA Network
UltraEthernet / RoCEv2 / IB
kv

Tokens at Scale




Tokens
at Scale
bob
joe
john
kv
Encoded Data
(binary)
KV Block
Key
Value
Selector
Attention
Heads
Tokens
Per
Block
Features
Per
Head

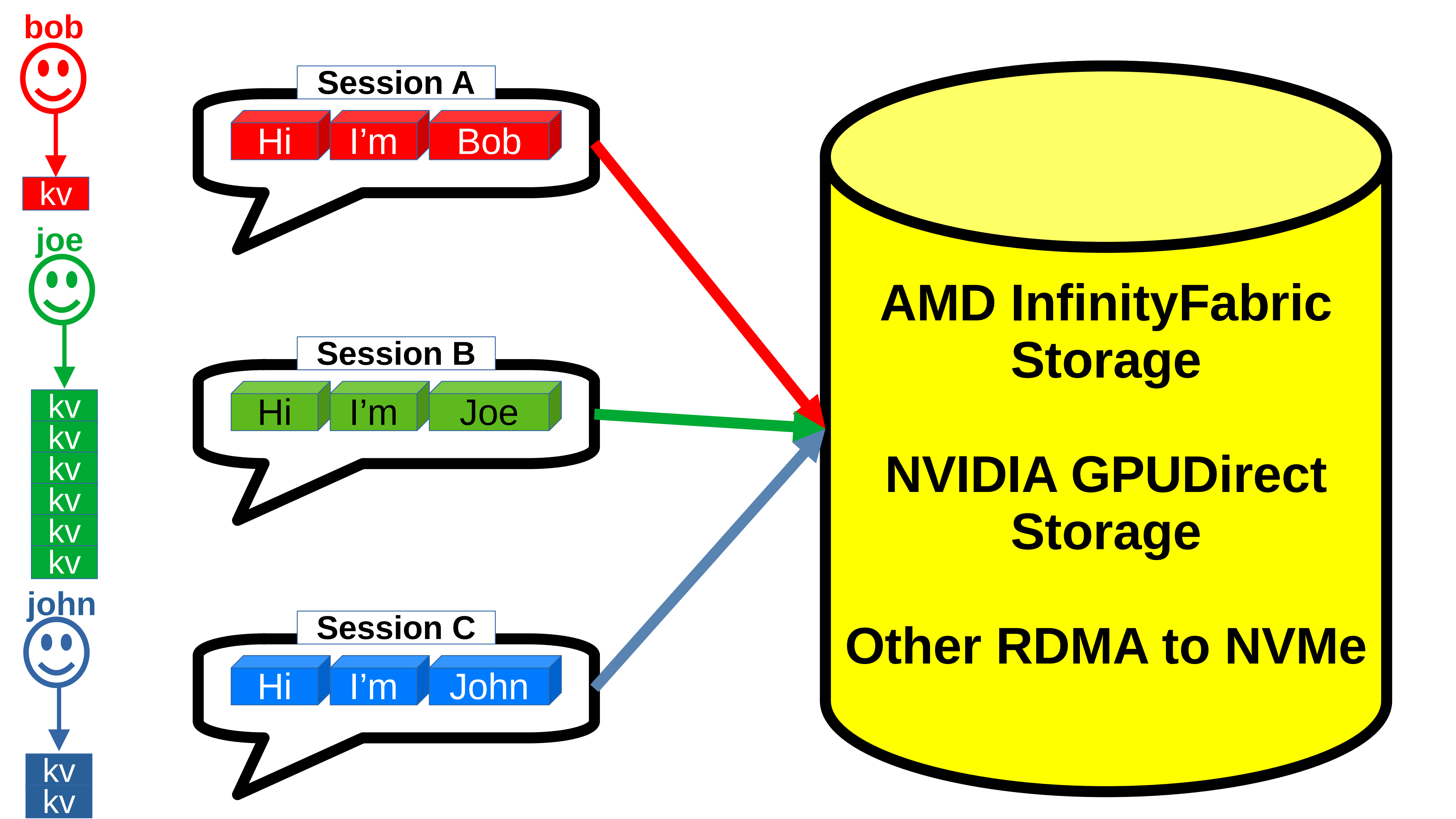
KV Block

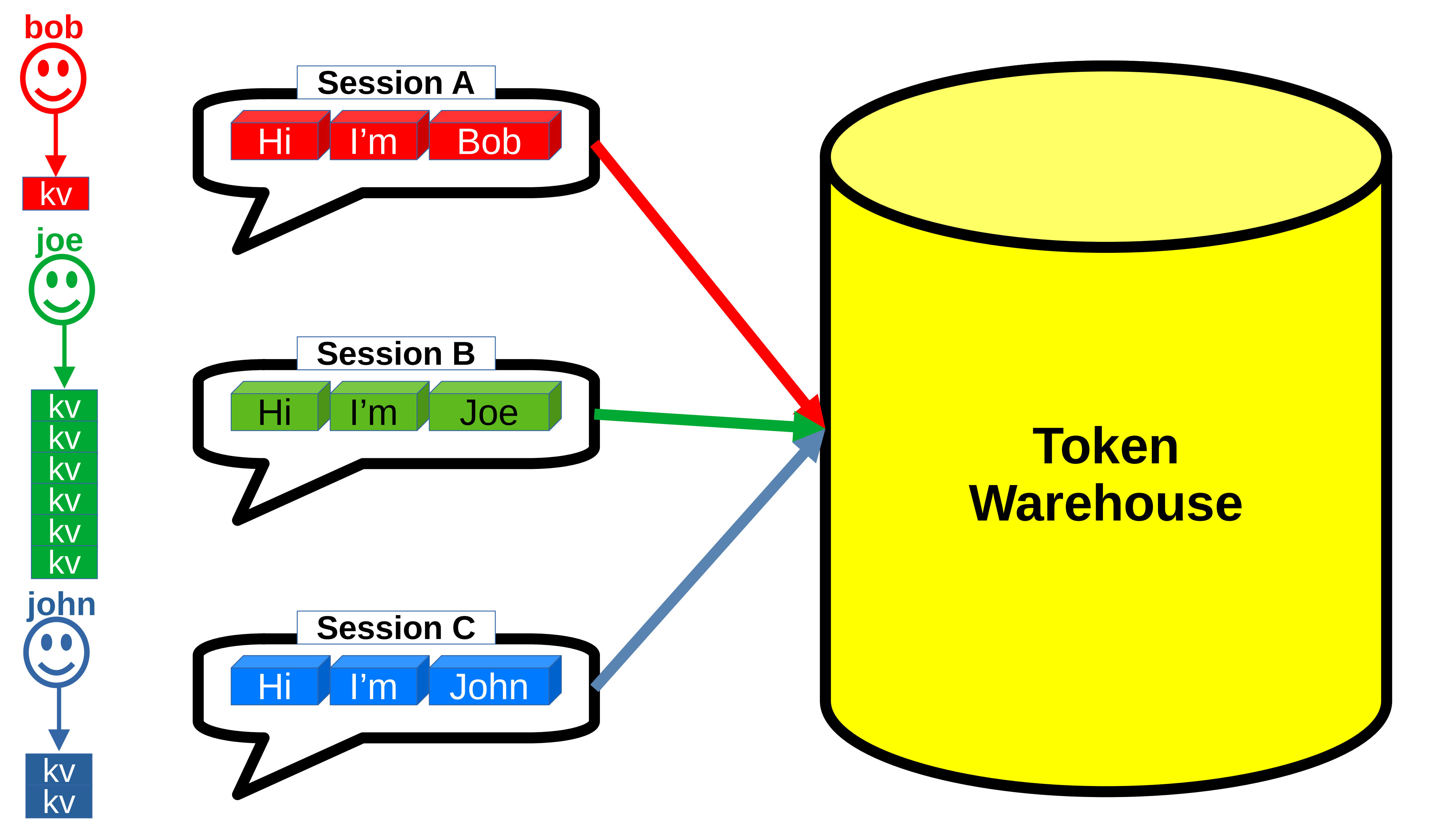
- Key Value Selector
- Attention Heads
- Tokens Per Block
- Features Per Head

```
00003F00:  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00003F10:  10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
00003F20:  20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
00003F30:  30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
00003F40:  40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
00003F50:  50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
00003F60:  60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F
00003F70:  70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F
00003F80:  80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
00003F90:  90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
00003FA0:  A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
00003FB0:  B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF
00003FC0:  C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF
00003FD0:  D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF
00003FE0:  E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF
00003FF0:  F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
```

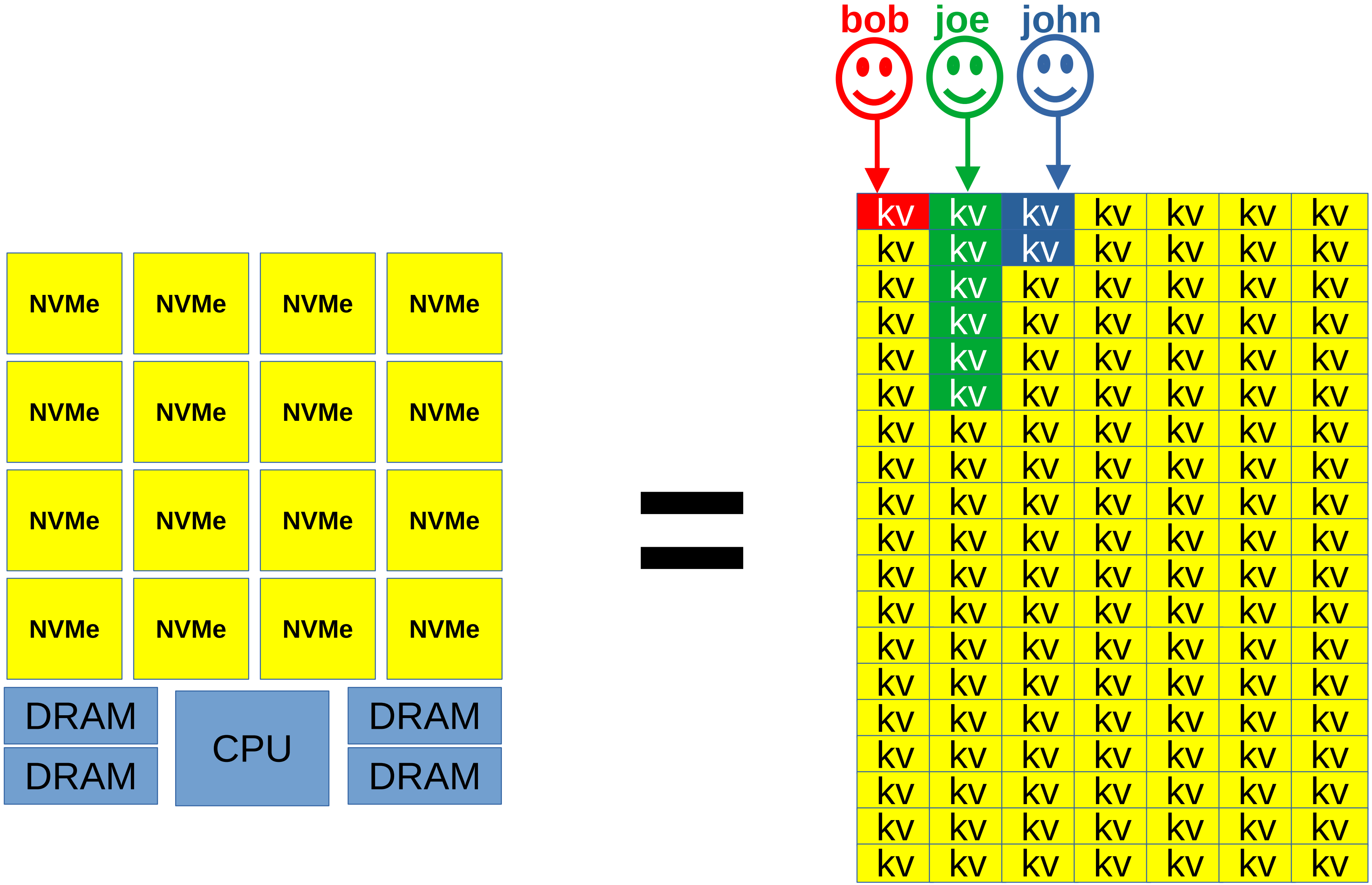
```
00003F00:  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00003F10:  10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F

00003F20:  20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F

00003F30:  30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F

00003F40:  40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F

00003F50:  50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F

00003F60:  60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F

00003F70:  70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F

00003F80:  80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F

00003F90:  90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F

00003FA0:  A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF

00003FB0:  B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF

00003FC0:  C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF

00003FD0:  D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF

00003FE0:  E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF

00003FF0:  F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
```

```
Block mapping (per two lines)
00003F00..00003F1F  →  kv=V, h=7, t=24, d=0..7
00003F10..00003F1F  →  d=0..7
00003F20..00003F3F  →  kv=V, h=7, t=25, d=0..7
00003F30..00003F3F  →  d=0..7
00003F40..00003F5F  →  kv=V, h=7, t=26, d=0..7
00003F50..00003F5F  →  d=0..7
00003F60..00003F7F  →  kv=V, h=7, t=27, d=0..7
00003F70..00003F7F  →  d=0..7
00003F80..00003F9F  →  kv=V, h=7, t=28, d=0..7
00003F90..00003F9F  →  d=0..7
00003FA0..00003FBF  →  kv=V, h=7, t=29, d=0..7
00003FB0..00003FBF  →  d=0..7
00003FC0..00003FDF  →  kv=V, h=7, t=30, d=0..7
00003FD0..00003FDF  →  d=0..7
00003FE0..00003FFF  →  kv=V, h=7, t=31, d=0..7
00003FF0..00003FFF  →  d=0..7
```


bob
kv
joe
kv
kv
kv
kv
kv
kv
john
kv
kv
Session A
Hi
I'm
Bob
Session B
Hi
I'm
Joe
Session C
Hi
I'm
John
AMD InfinityFabric Storage
NVIDIA GPUDirect Storage
Other RDMA to NVMe



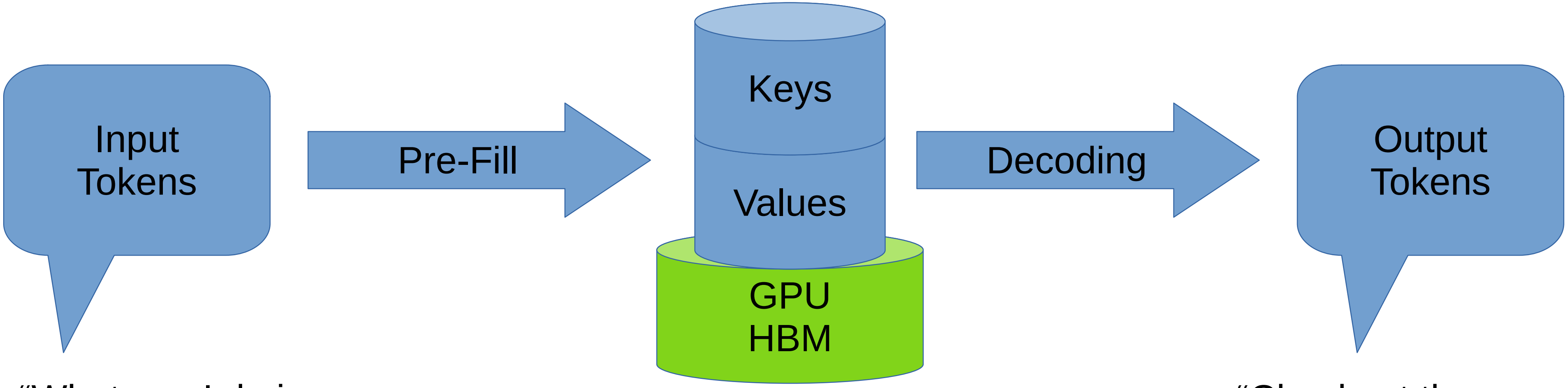
bob
kv
joe
kv
kv
kv
kv
kv
kv
john
kv
kv
Session A
Hi I’m Bob
Session B
Hi I’m Joe
Session C
Hi I’m John
Token Warehouse


NVMe NVMe NVMe NVMe
NVMe NVMe NVMe NVMe
NVMe NVMe NVMe NVMe
NVMe NVMe NVMe NVMe
DRAM
DRAM
CPU
DRAM
DRAM
=
bob
joe
john
kv kv kv kv kv kv kv
kv kv kv kv kv kv kv
kv kv kv kv kv kv kv
kv kv kv kv kv kv kv
kv kv kv kv kv kv kv
kv kv kv kv kv kv kv
kv kv kv kv kv kv kv
kv kv kv kv kv kv kv
kv kv kv kv kv kv kv
kv kv kv kv kv kv kv
kv kv kv kv kv kv kv
kv kv kv kv kv kv kv
kv kv kv kv kv kv kv
kv kv kv kv kv kv kv
kv kv kv kv kv kv kv
kv kv kv kv kv kv kv
kv kv kv kv kv kv kv
kv kv kv kv kv kv kv
kv kv kv kv kv kv kv

5

Internals of Long Context Inference

Short Context



“What can I do in Maryland after the conference?”

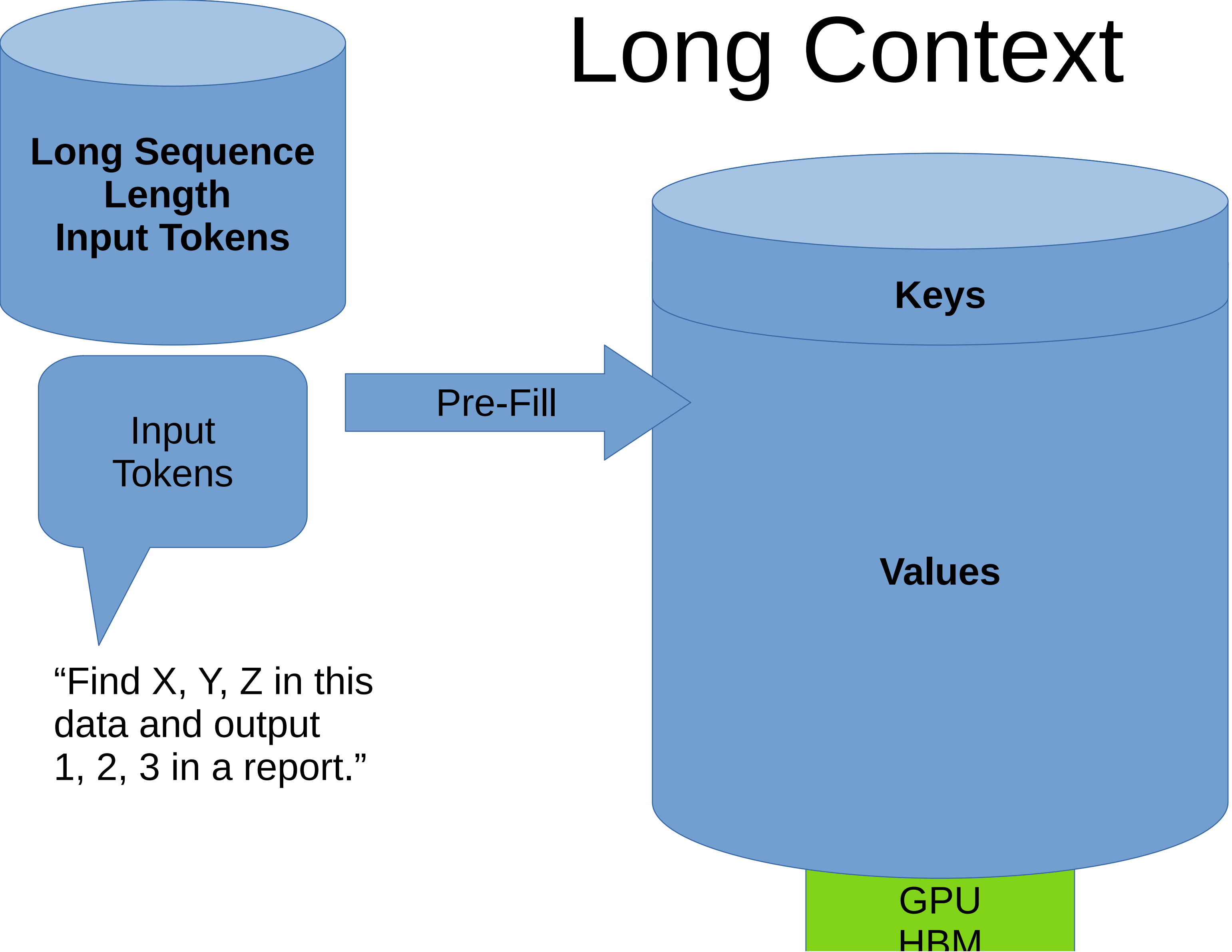
Size:
Hidden Dimensions
Number of KV Heads

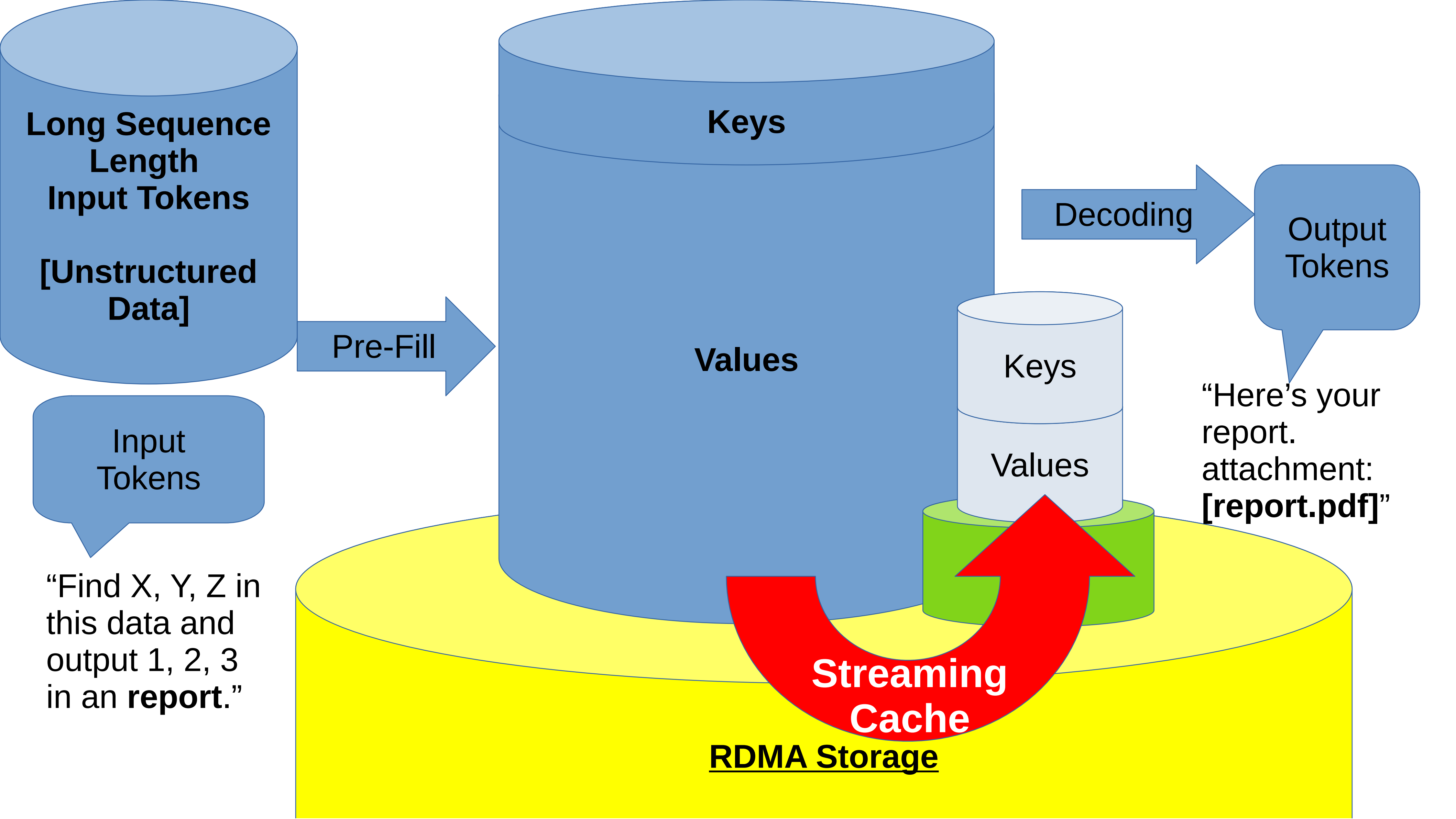
“Checkout the cryptologic museum.”

Revisiting Llama-4-Scout

- **Interleaved Rotary Positional Embeddings** (**IRoPE**) attention algorithm.
- Alternates between **No Positional Encoding (NoPE)** global attention layers with chunked local-attention layers that use RoPE.
- Attention algorithm makes it **theoretically capable** of supporting **10M token sequence lengths**.
- 10M tokens is roughly **38 of the longest books** on Project Gutenberg.
- **32Tb** of working memory KV Cache required for a **single 10M token inference query**.
- Today's GPU Memory constraints mean this sequence length is **not practical**.

Long Context




Long Sequence Length Input Tokens

[Unstructured Data]
Input Tokens
“Find X, Y, Z in this data and output 1, 2, 3 in an report.”
Pre-Fill
Keys
Values
Decoding
Output Tokens
“Here’s your report. attachment: [report.pdf]”
Keys
Values
Streaming Cache
RDMA Storage

ReBOOT

https://github.com/arthurrasmusson/reboot

Ghidra on HPC clusters.

Long context AI with Ghidrascript.

Infer on full codebases worth of decompiled code with single long context inference queries using Paged Attention over RDMA.

ReBOOT

Long context documentation generation for decompiled binaries.

Single shot codebase refactors for open source codebases or decompiled binaries.

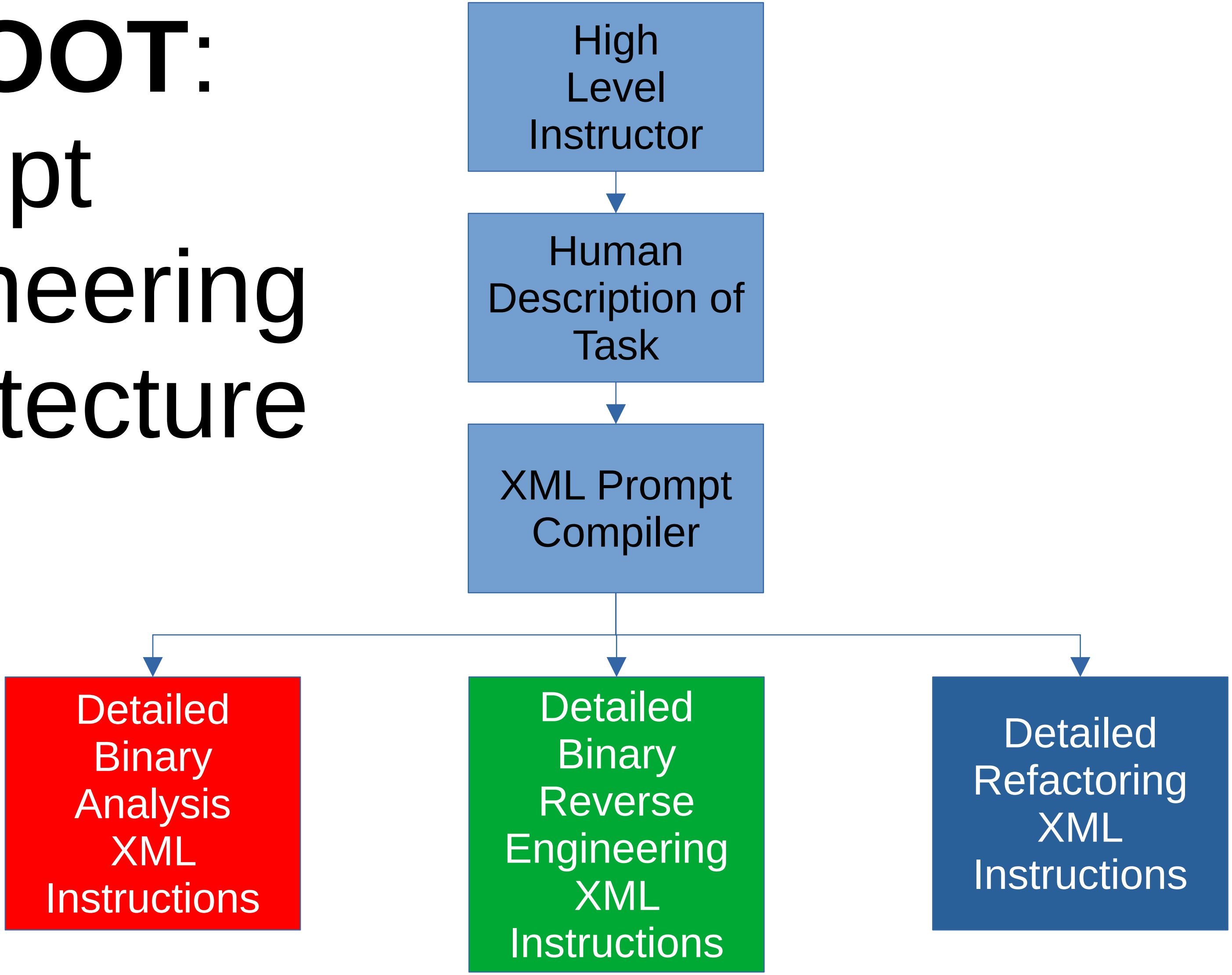
Agentic workflows for writing and executing Ghidrascript.

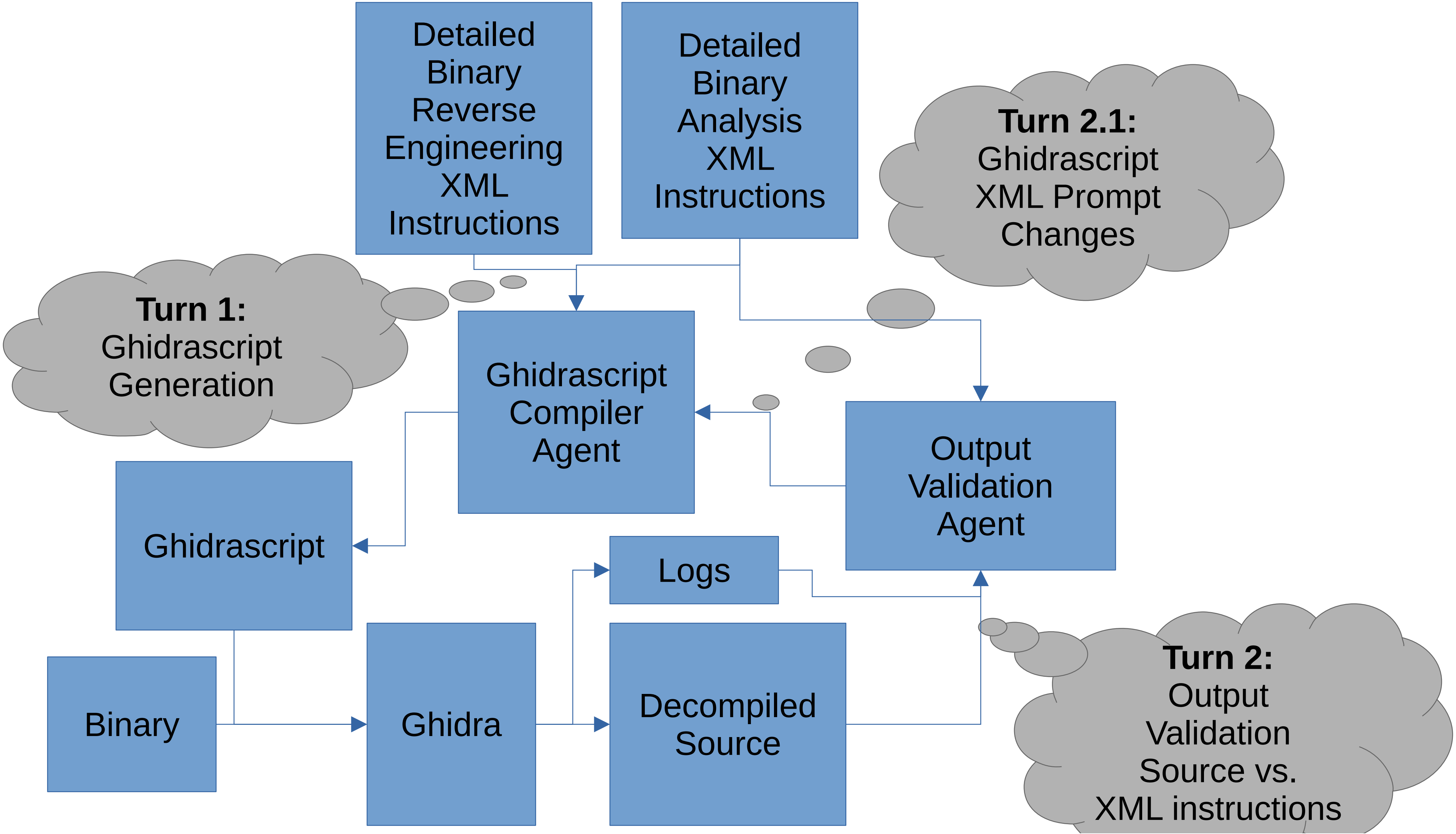
Ghidra,
meet Paged Attention over RDMA

- Shorten Observation / Orientation time in reverse engineering constrained OODA Loops.
- Long context input sequence (able to process entire codebases for large decompiled binaries).

ReBOOT:
High
Level
Architecture

ReBOOT: Prompt Engineering Architecture




Detailed Binary Reverse Engineering XML Instructions
Detailed Binary Analysis XML Instructions
Turn 2.1: Ghidrascript XML Prompt Changes
Turn 1: Ghidrascript Generation
Ghidrascript Compiler Agent
Output Validation Agent
Ghidrascript
Logs
Binary
Ghidra
Decompiled Source
Turn 2: Output Validation Source vs. XML instructions

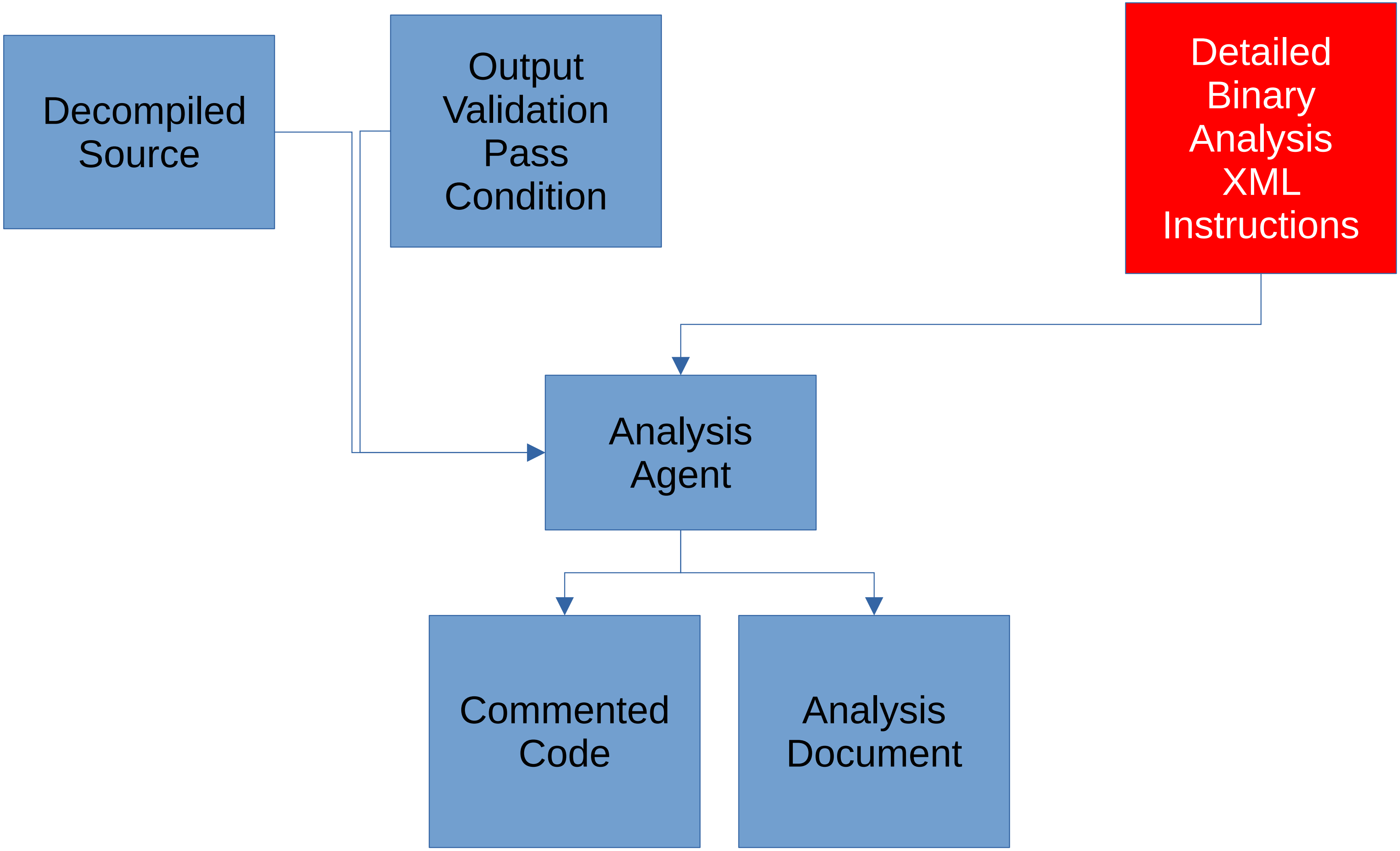
ReBOOT for Binary and Open Source Code Analysis:

HPC analysis of binaries using

Ghidra

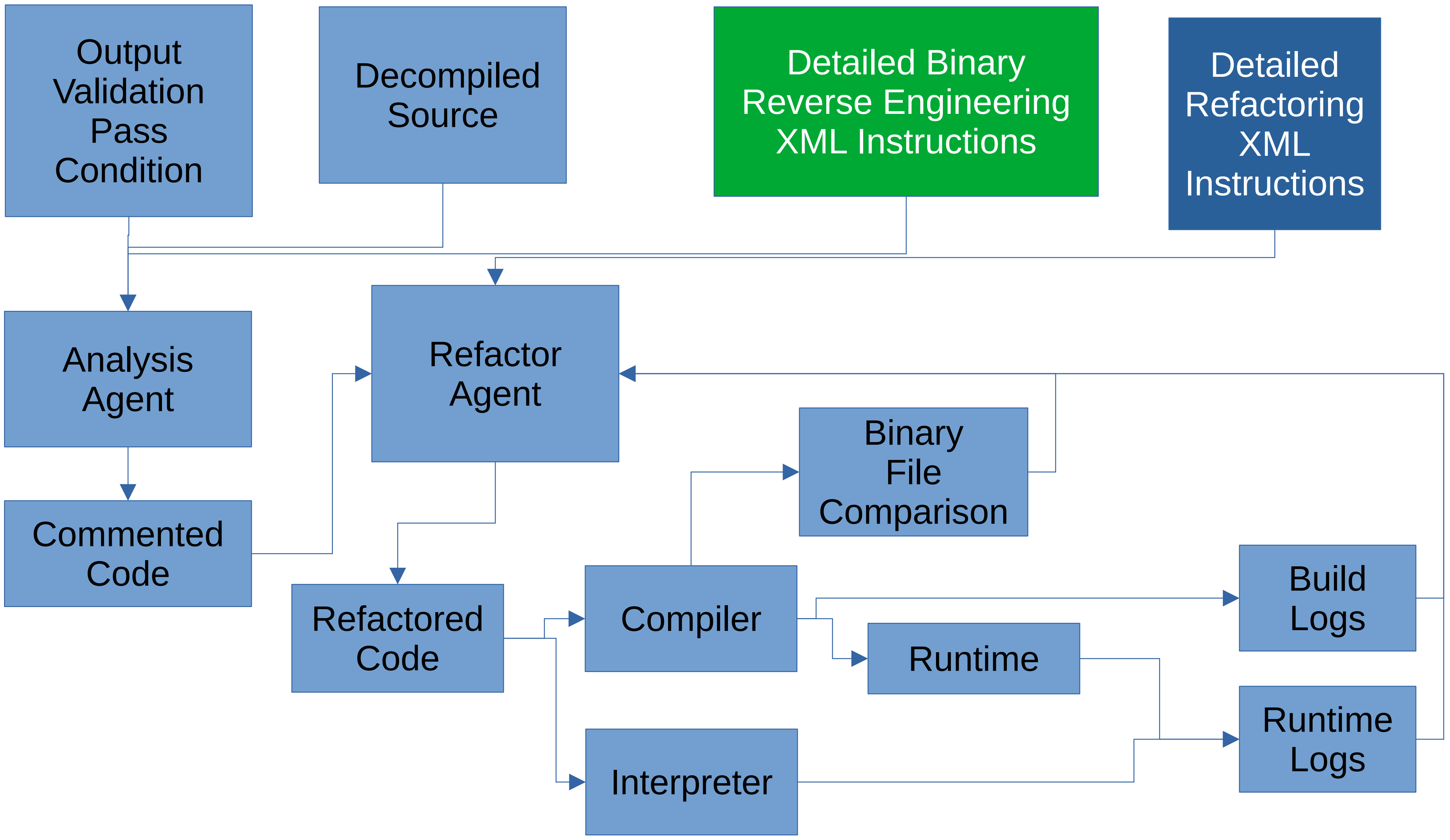
and

Paged Attention over RDMA.


Decompiled Source
Output Validation Pass Condition
Detailed Binary Analysis XML Instructions
Analysis Agent
Commented Code
Analysis Document

ReBOOT for Open Source Re-Implementation:

Re-Implementation
of binaries or open source programs with
Paged Attention over RDMA.



Output Validation Pass Condition
Decompiled Source
Detailed Binary Reverse Engineering XML Instructions
Detailed Refactoring XML Instructions
Analysis Agent
Refactor Agent
Commented Code
Binary File Comparison
Refactored Code
Compiler
Build Logs
Runtime
Runtime Logs
Interpreter

Unsolved Problems

- Big Problem:
  We use **Page Faults**, not
  page fetch speculation.

Unsolved Problems

- **Ramp up latency** with GPU exit to CPU to configure NICs for staging RDMA page copies.
  Negotiation of RDMA inbox & outbox results in Paged Attention pipeline stall, increasing Time to First Token (TTFT), and Inter-Token Latency (ITL).

  Issue affects:

  UCX for GPU to GPU

  GPUDirect Storage for GPU to NVMe.

  (Test with DPUs to check for latency reduction)
- MIO instructions in cubin (SASS CUDA binary) for GPU handle memory movement with predictable CPI latency per cache tier / put and fetch size (based on floating point / int precision).
- No such predictive latency for RDMA.

Unsolved Problems

- **MIO instruction** in Streaming Assembly (SASS), ELF format CUDA binary optimizes data pre-fetching in GPU assembly code.
- PTXas and NVCC compiler rely on predictable memory transfer Cycle Per Instruction (CPI).
- **GPU is constantly overlapping memory copies with compute** to pre-fetch data. This is foundational to CUDA kernel performance.
- Memory movement with predictable CPI latency per cache tier / put and fetch size (based on floating point / int precision) is optimized by the CUDA programmer and CUDA compiler (PTXas and NVCC).
- **No fixed Cycle Per Instruction (CPI) budget for RDMA to NVMe.**

Summary of Presentation

- Demonstrated how Paged Attention over RDMA enables long sequence lengths for attention algorithms that exceed memory limitations of GPUs.
- Demonstrated how Paged Attention over RDMA technique provides long term growth for future long context attention software.

Summary of Presentation

- Example code shared to run PAoR on and analyze infrastructure suited for Paged Attention over RDMA so that you can both practice with long sequence lengths and understand how they work.
- ReBOOT available at: **https://github.com/arthurrasmusson/reboot**
- **TFEITool Scripts will be at**: **https://github.com/arthurrasmusson/TFEITool**

Thank You!

Arthur Rasmusson

Email:
arthur@vgpu.io

Blog:
https://vgpu.io

GitHub:
https://github.com/arthurrasmusson