


# Unlocking the Full Potential of NAND Flash for AI Era

**GB KIM**
SK hynix / AI-N Design

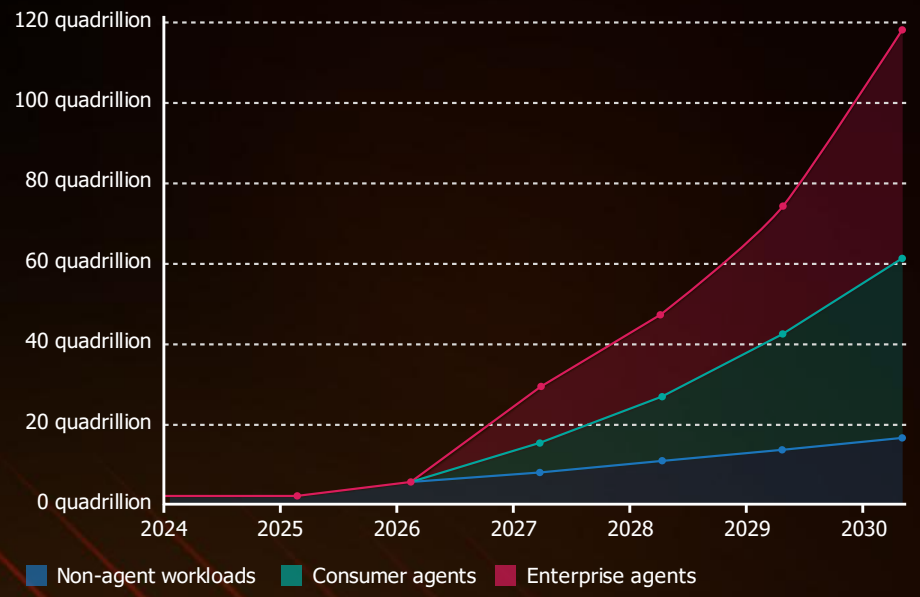
# Introduction

# Introduction – Agentic AI system trend

- Token processing volumes are surging at an unprecedented rate
- As multi-agent systems evolve, compute requirement and interactivity skyrocket

## Token Usage

✓ Token use by AI agents is expected to multiply 24 times by 2030




*Source: Goldman Sachs Research Estimates as of May 2026

## Memory-centric era

**Memory Bandwidth** → Gating → **System Performance**

Tokens/sec, Latency, GPU Utilization

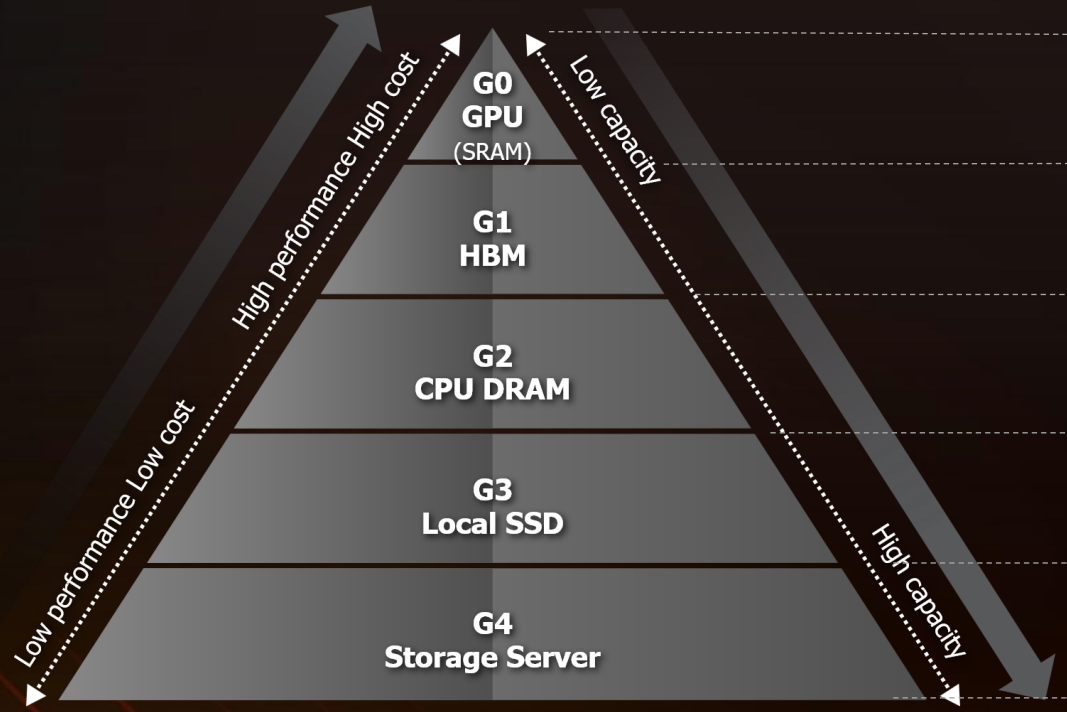
**Memory Capacity** → Gating → **Model & Context Scale**

Model Size, KV Cache, Context Length

SK hynix

# Introduction – Current Memory Hierarchy

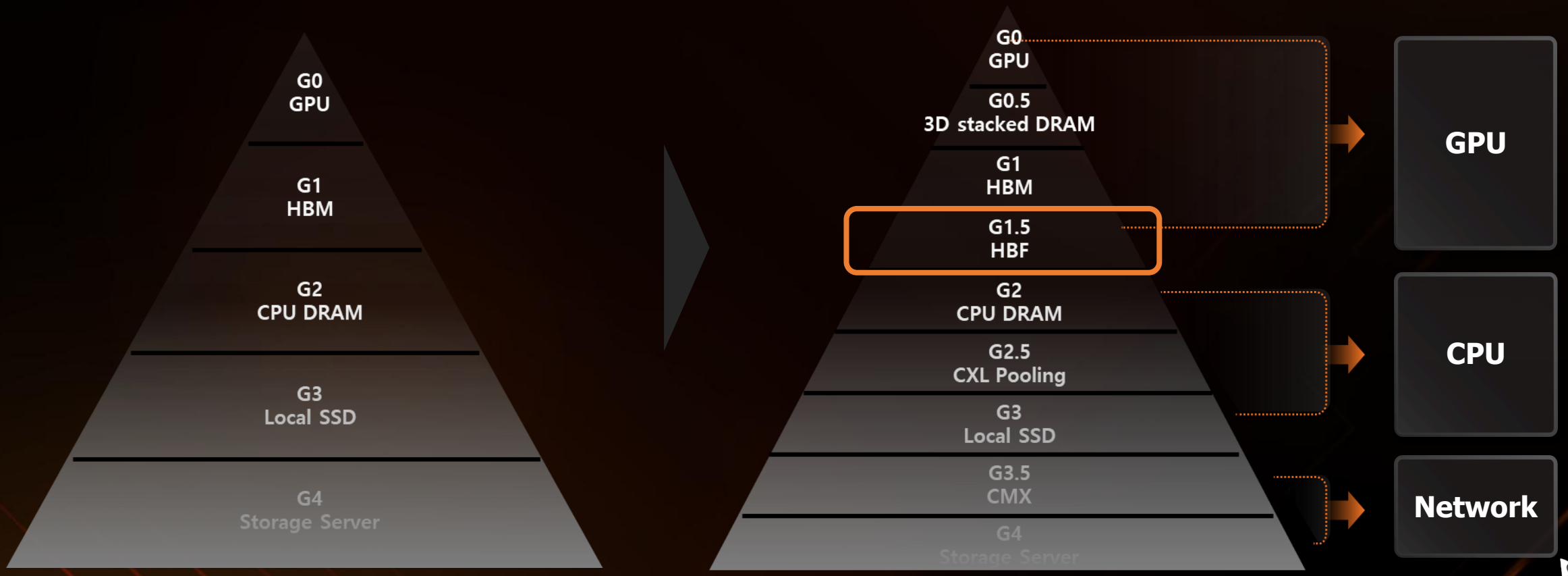
- From HBM to SSD, memory hierarchies are defined to maximize overall system performance
  - With rapid growth in performance requirements, additional memory and storage solutions may be needed




| | Latency | BW | Capacity |
|---|---|---|---|
| G0 GPU (SRAM) | <Tens of ns | > Tens of TB/s | Few MBs |
| G1 HBM | ~Tens of ns | Few TB/s | Tens of GBs |
| G2 CPU DRAM | ~Tens of ns | Hundreds of GBs/s | Hundreds of GBs |
| G3 Local SSD | ~Few us | ~Few GB/s | Few TBs |
| G4 Storage Server | ~Few ms | ~MB/s | Tens of TBs |

# Introduction – New Memory Hierarchy

- Introducing new tiers across the memory hierarchy enhances data efficiency and enables tier-specific optimization based on customer priorities and technological innovation



# Efficient Utilization of existing SSD

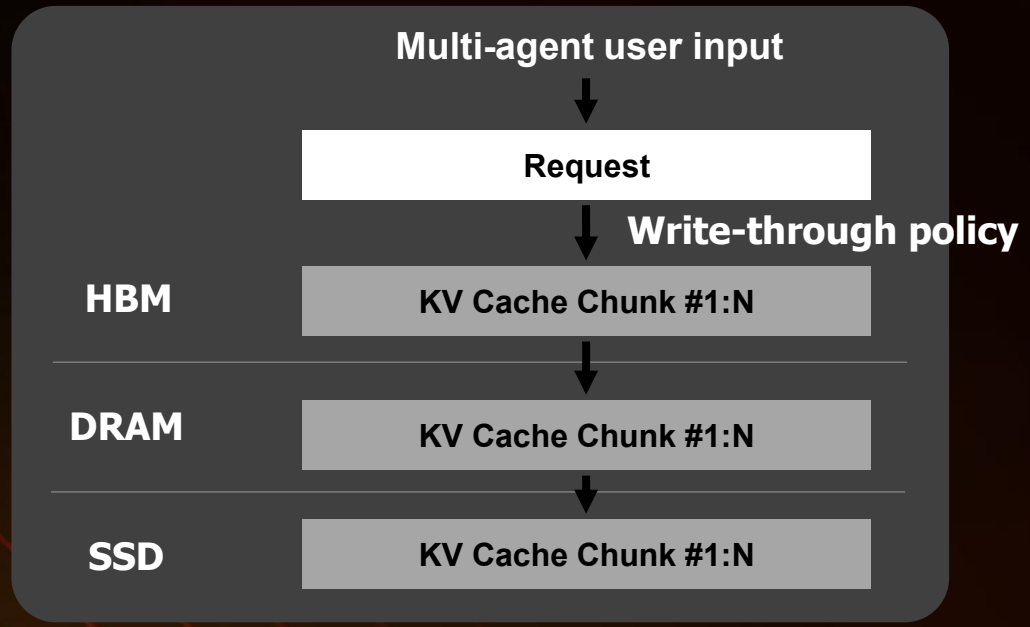
# Efficient Utilization of Conventional Storage

- Current : Storing all KV Cache in multiple memory tiers

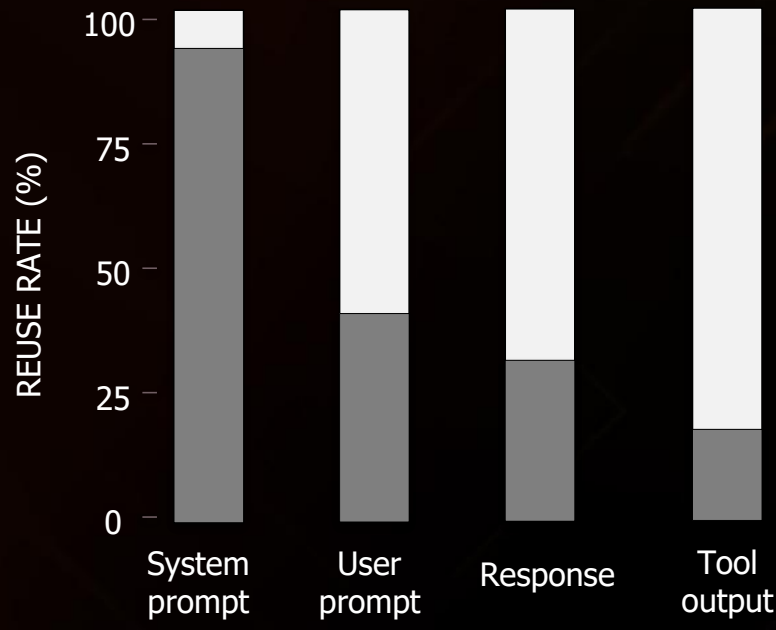
  → Since Different reuse rate, user can define KV Cache segment based on the reuse rate.

**KV Cache management policy**

✓ When the capacity of each tier is full, an LRU-based drop occurs.




✓ Reuse rate varies by Segment




Reference: arXiv:2605.18825v1 ↗

SK hynix

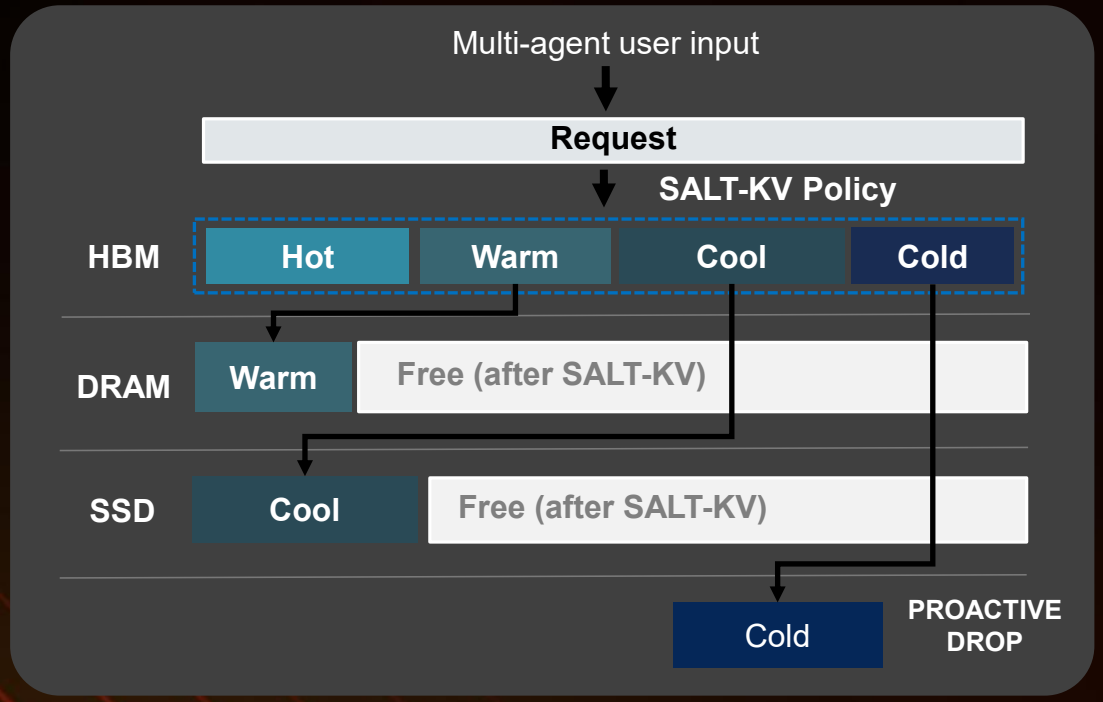
# Segment-Aware Lifecycle Tiering for KV (SALT-KV)

- Reuse rate : HBM(hot segment), DRAM(warm), SSD(cool) and drop(Cold segment)

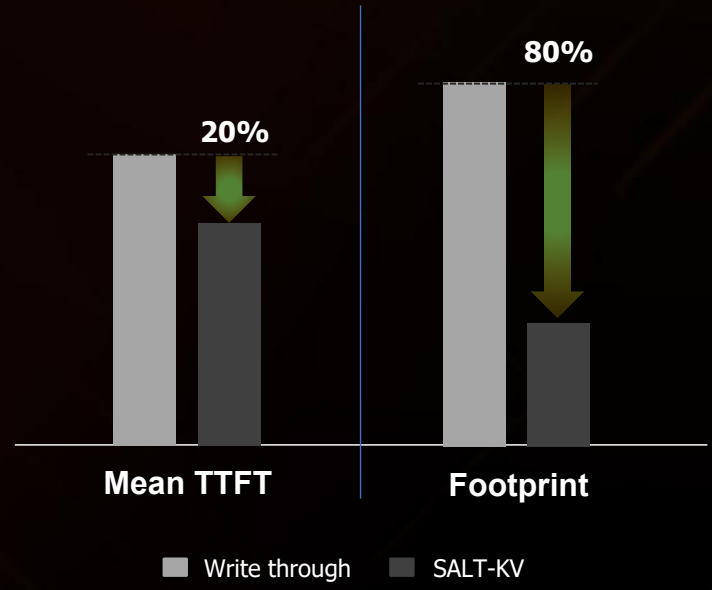
  → Improving TTFT and Reducing footprint of DRAM and SSD

## SALT-KV management policy

✓ Tier, Drop : determined based on the reuse rate



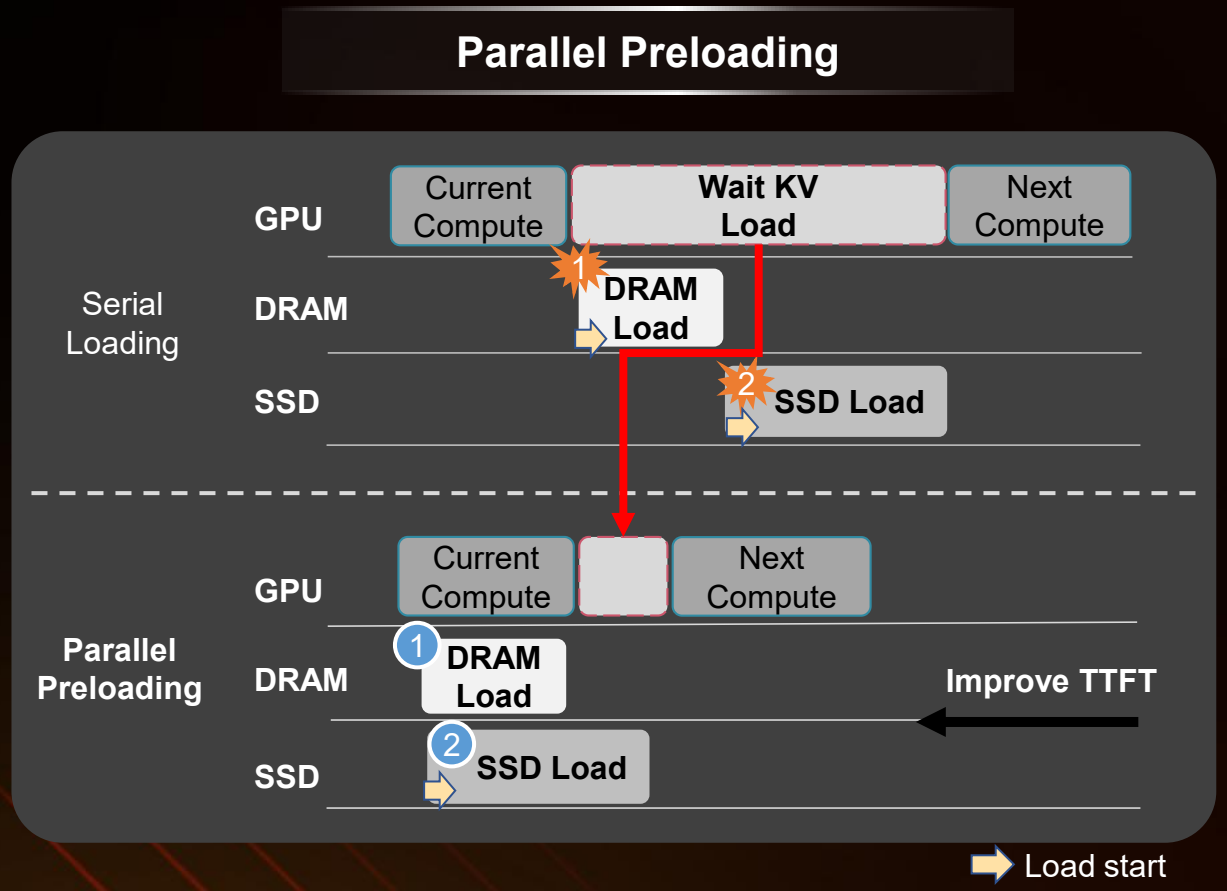

✓ TTFT and footprint : Write-through vs. SALT-KV



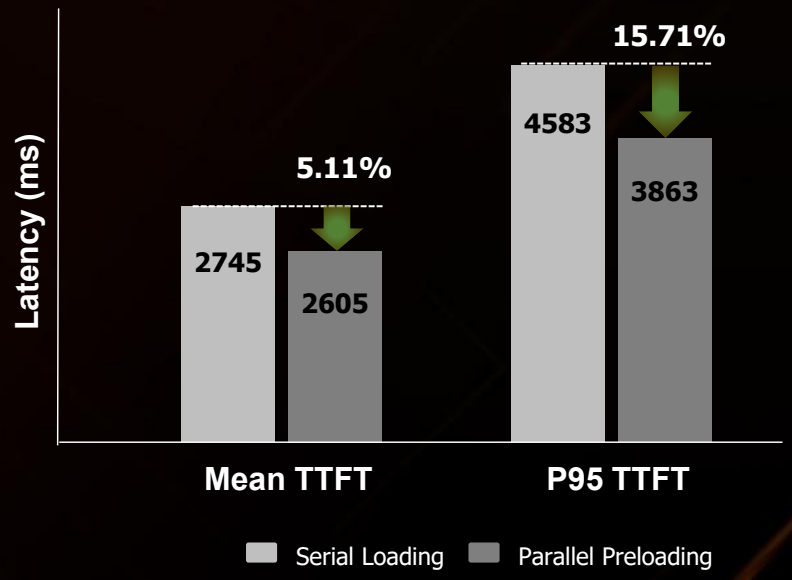
# Parallel Preload scheduler optimization

- Optimized vLLM scheduler to enable parallel preload of the KV cache

  → Improving TTFT by addressing overlap between compute and loading



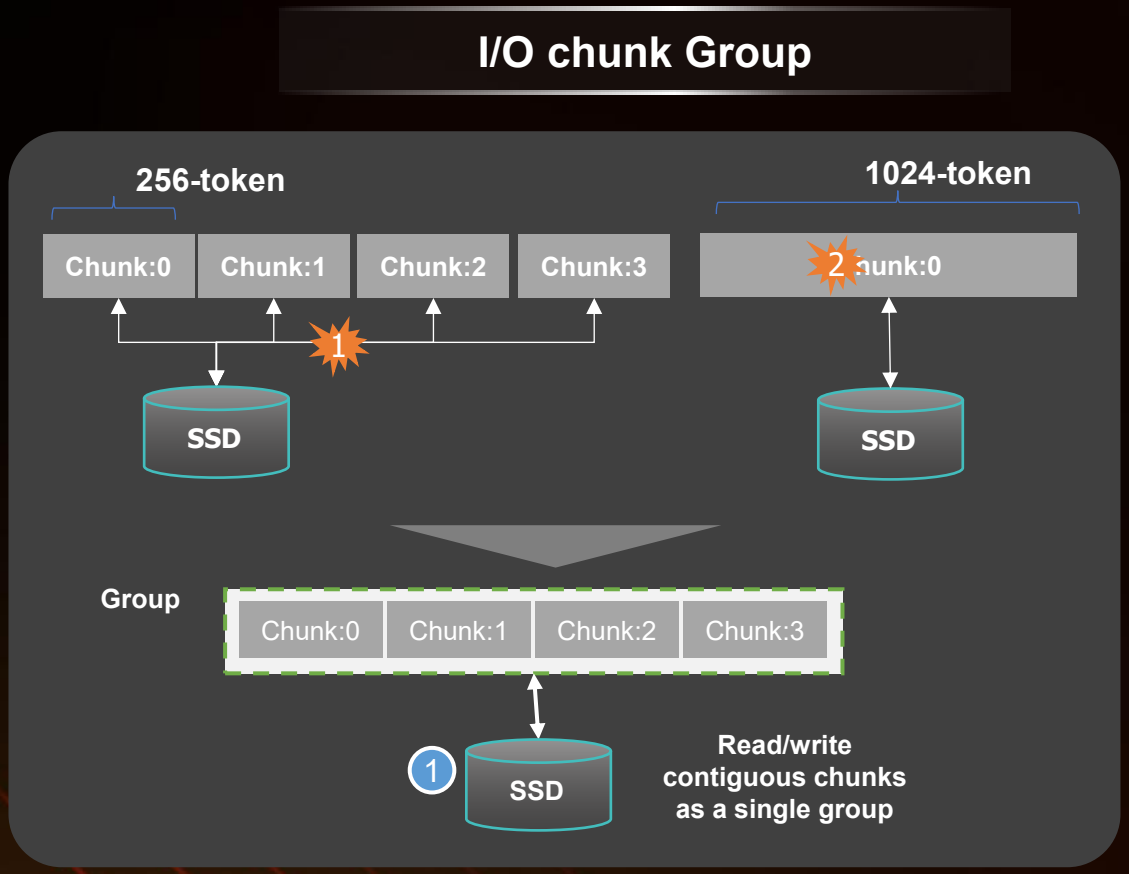

✓ TTFT : Serial loading vs. KV Parallel Preloading



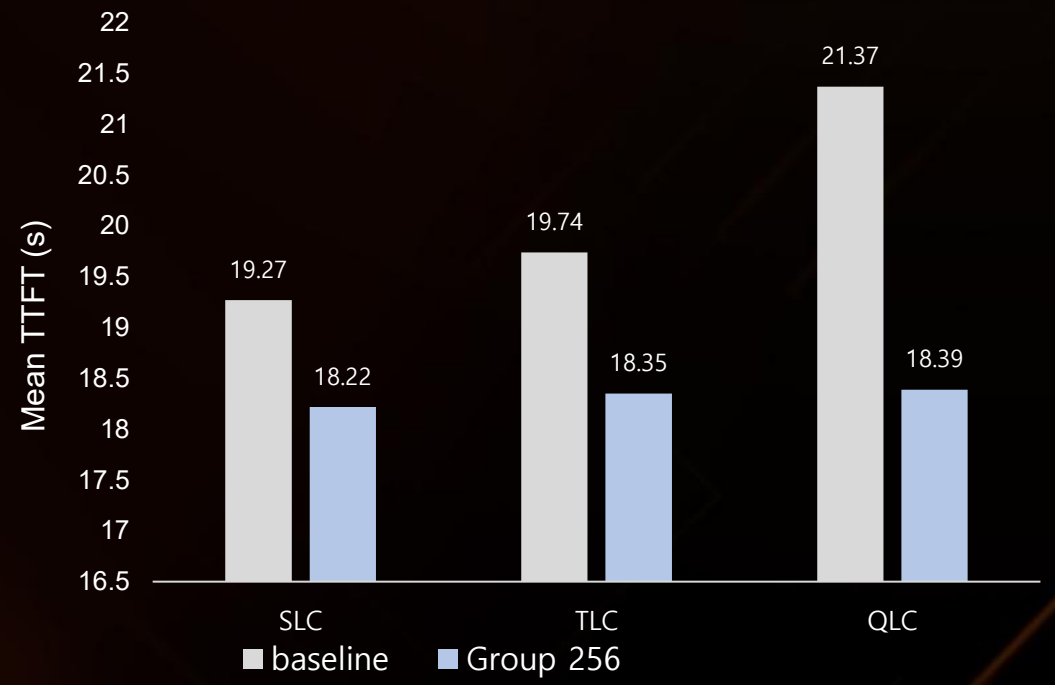
# SSD chunk grouping optimization

- Local KV chunk : fine-grained, Physical SSD I/O : group chunk

  → Maintaining hit-rate and reducing I/O overhead




✓ TTFT : Non-chunk grouping vs Chunk grouping




(최대 chunk 256개를 grouping)

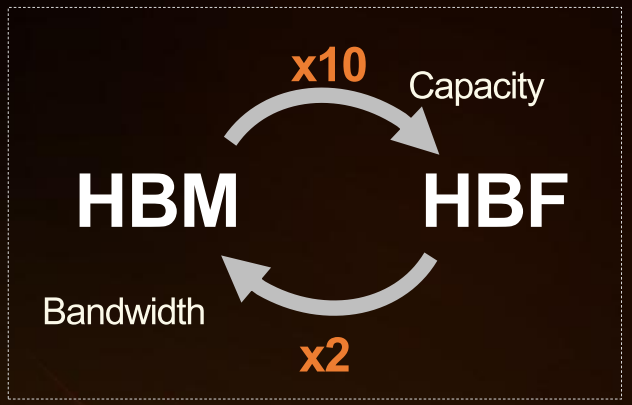
# NAND Flash, new wave - HBF

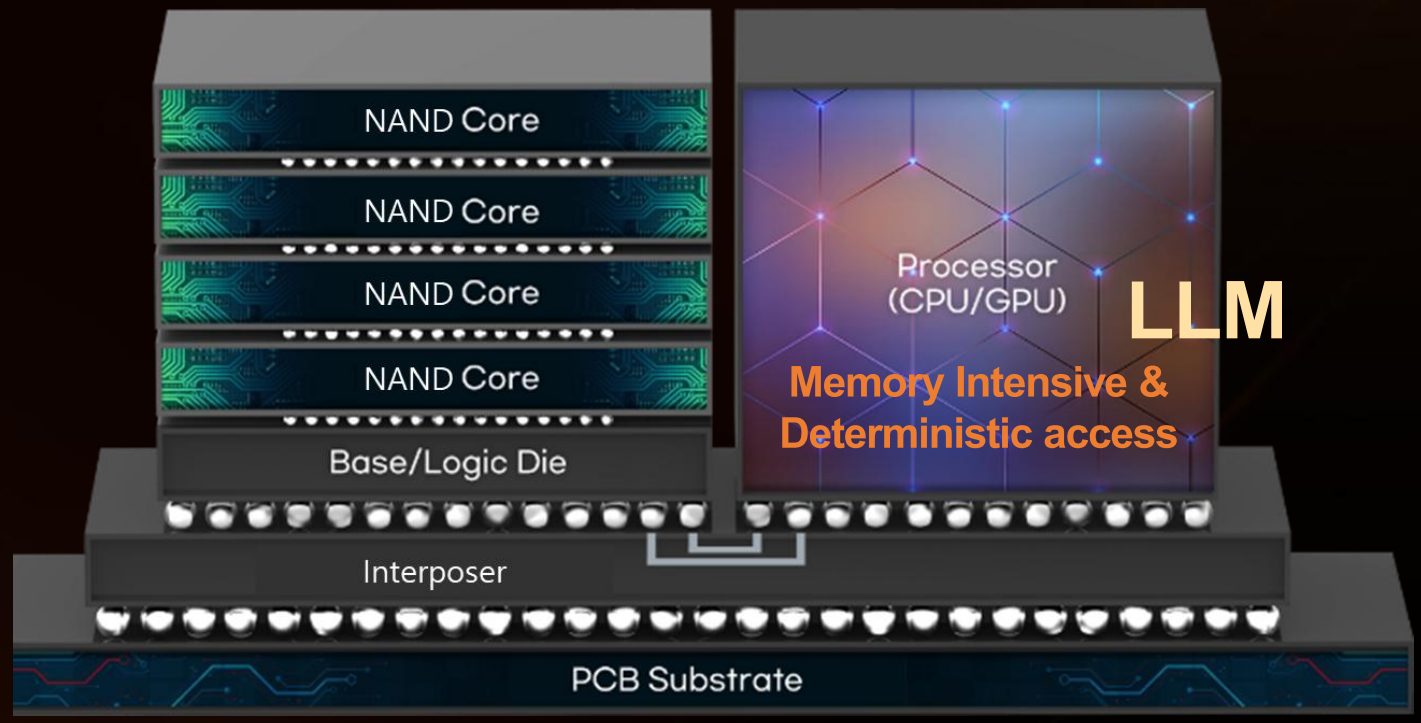
# What is ‘HBF’ ?

- HBM like Bandwidth, SSD like Capacity

✓ Throughput (like HBM)
**SSD << HBF ≈ HBM**

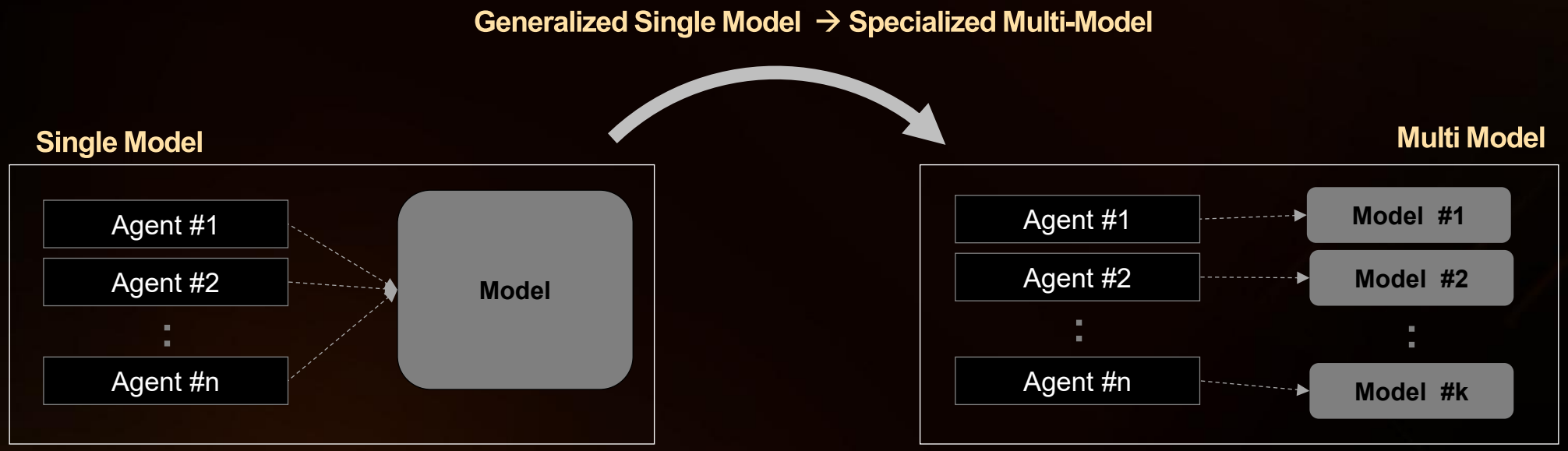
✓ Capacity (like SSD)
**HBM << HBF ≈ SSD**



# Why ‘HBF’? AI Service evolution : Multi Model

- Agentic AI services : several small & specialized multi-model > single large model
- Operational cost & Energy efficiency




Specialized Model for each Agent

- Various Multi-models → Larger capacity
- Service → Model switching

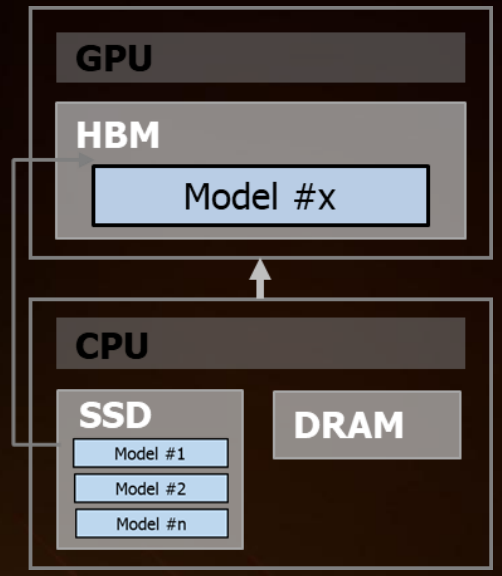
# Why ‘HBF’? AI Service evolution : Multi Model

- Multi-Model without Model loading

**Model loading from storage**

✓ Due to HBM capacity limitations, multi-models are stored in CPU DRAM or SSD

– models are loaded whenever needed.

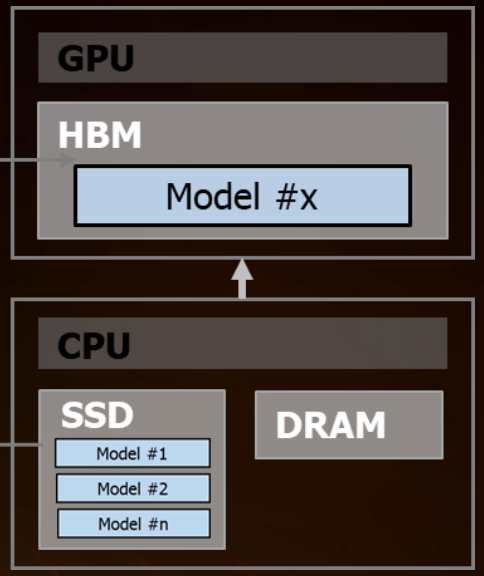


# Why ‘HBF’? AI Service evolution : Multi Model

- Multi-Model without Model loading

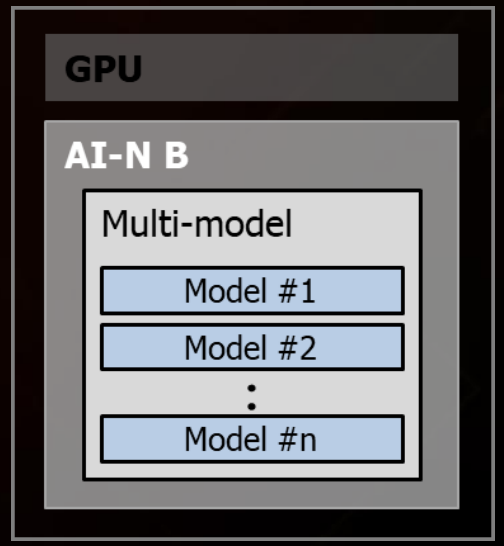
## Model loading from storage

- ✓ Due to HBM capacity limitations, multi-models are stored in CPU DRAM or SSD
  - models are loaded whenever needed.



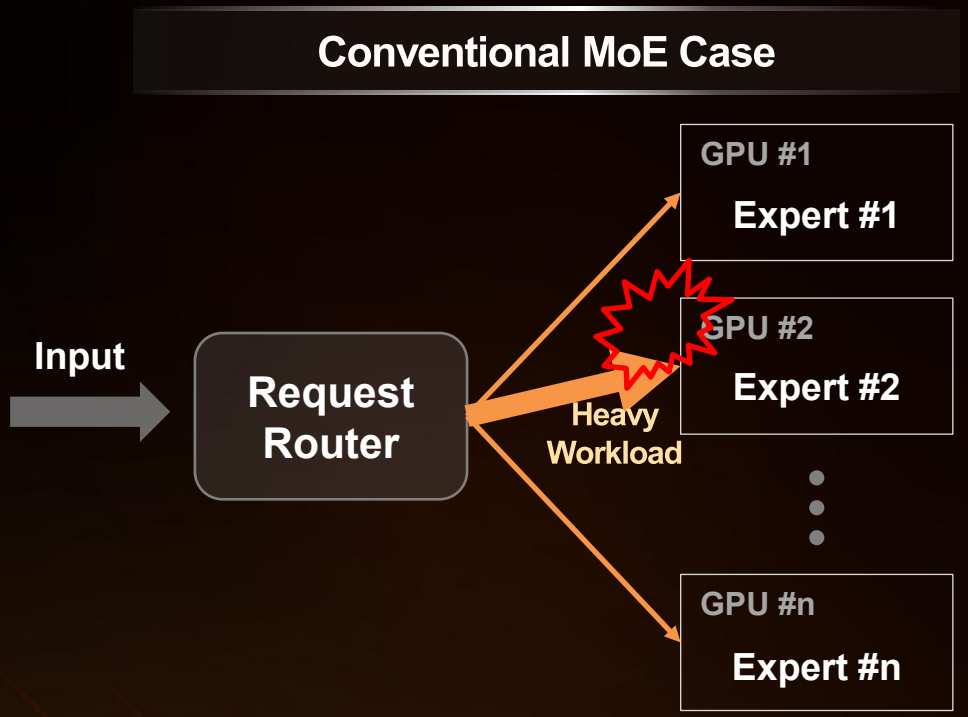

## Model resident in HBF

- ✓ Store multiple models in HBF
  - Service without loading models.



# Why ‘HBF’? Workload Balance : MoE Model

- one expert per GPU card : With the MoE model, workload balance issues
- Enabling multiple expert stores per GPU with HBF



# Why ‘HBF’? Workload Balance : MoE Model

- one expert per GPU card : With the MoE model, workload balance issues
- Enabling multiple expert stores per GPU with HBF

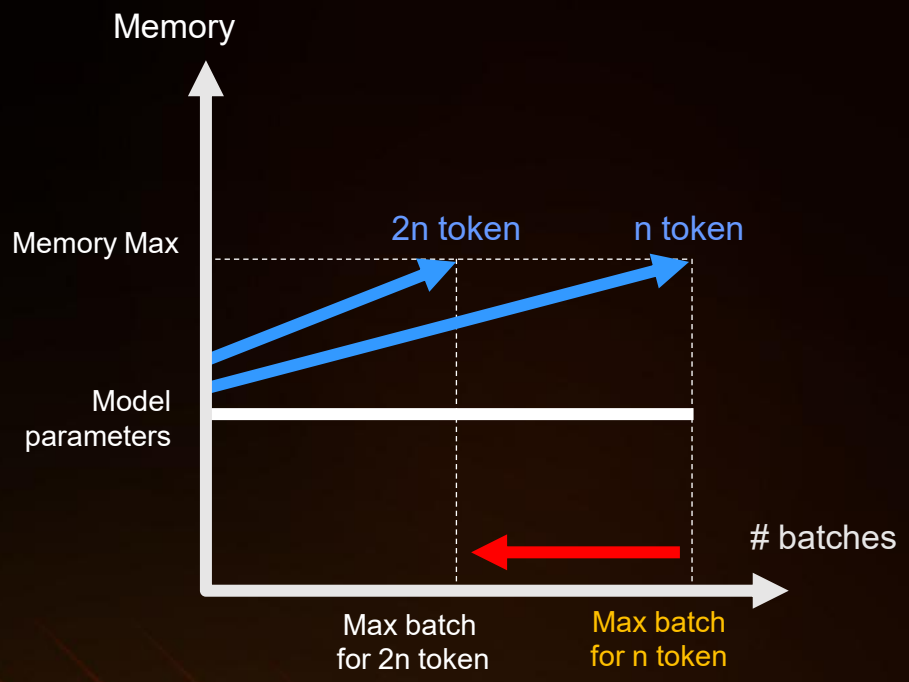


# Why ‘HBF’? More token needs more memory capacity

- Token length vs Batch size trade-off issue

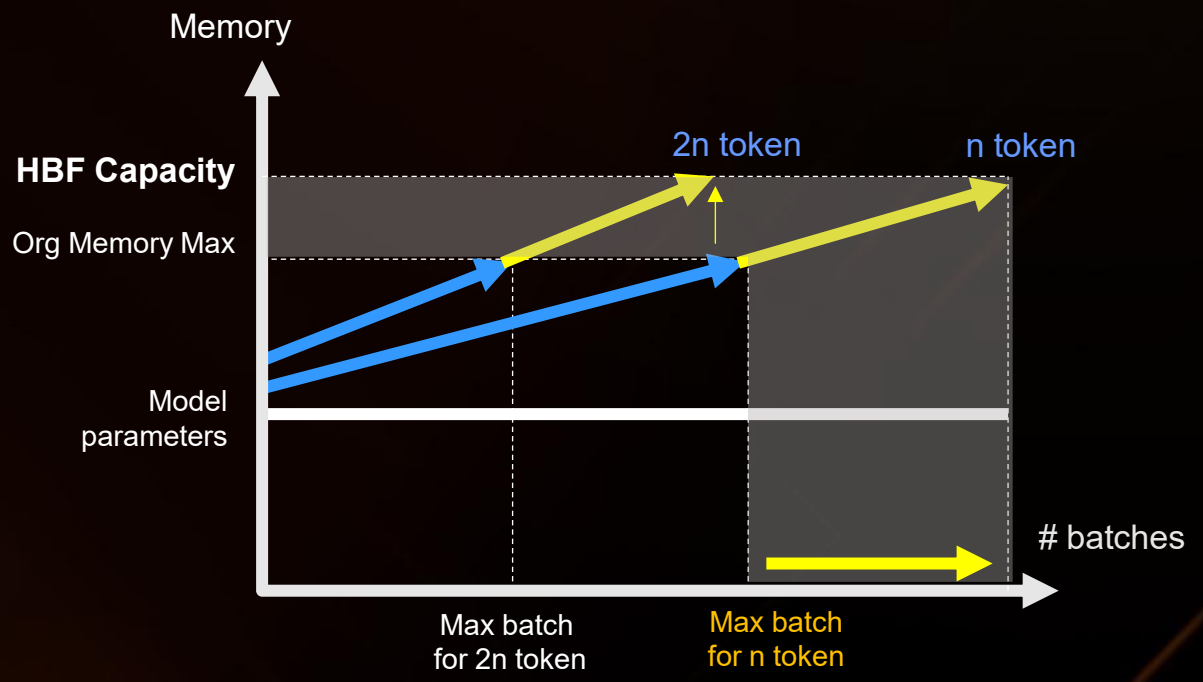
→ By supplying sufficient capacity with HBF

**As Token increases, batch size decreases**



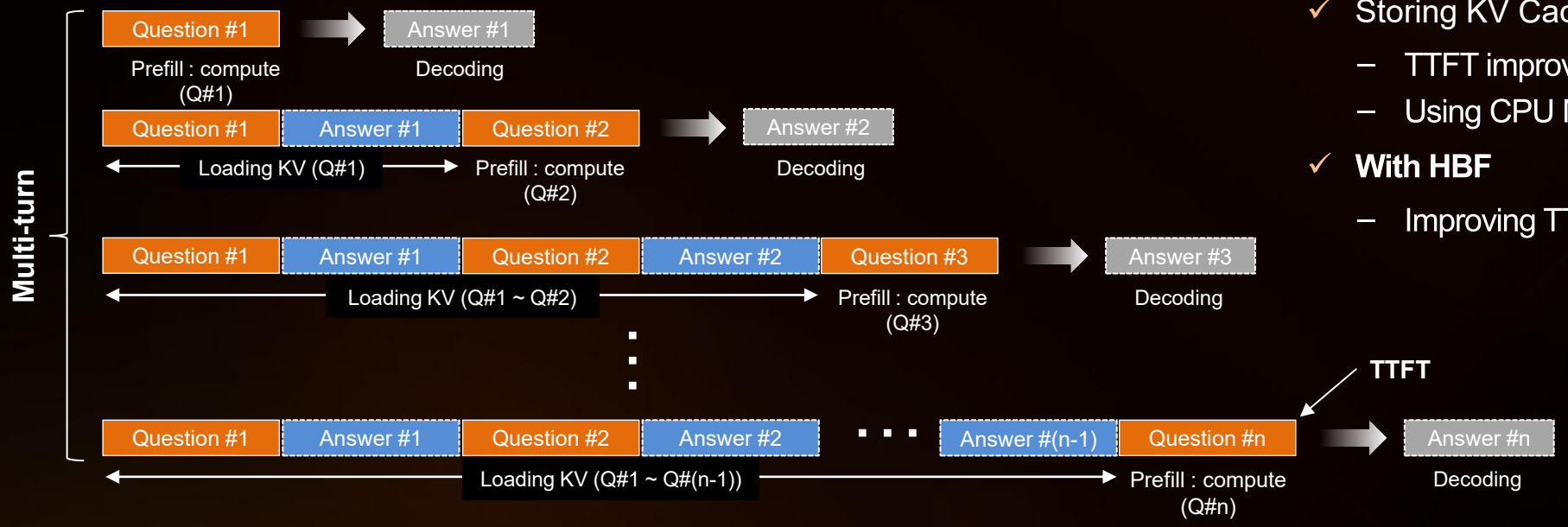

Reducing batch size → lower GPU utilization

**With HBF, token and batch can increase**



# Why ‘HBF’? KV cache re-use improves multi-turn performance

- In Multi-turn, Maintaining existing context issue




- ✓ Storing KV Cache for Context
  - TTFT improved, but Memory Capacity issue
  - Using CPU Memory or spill-out to SSD
- ✓ **With HBF**
  - Improving TTFT without CPU Memory and/or SSD

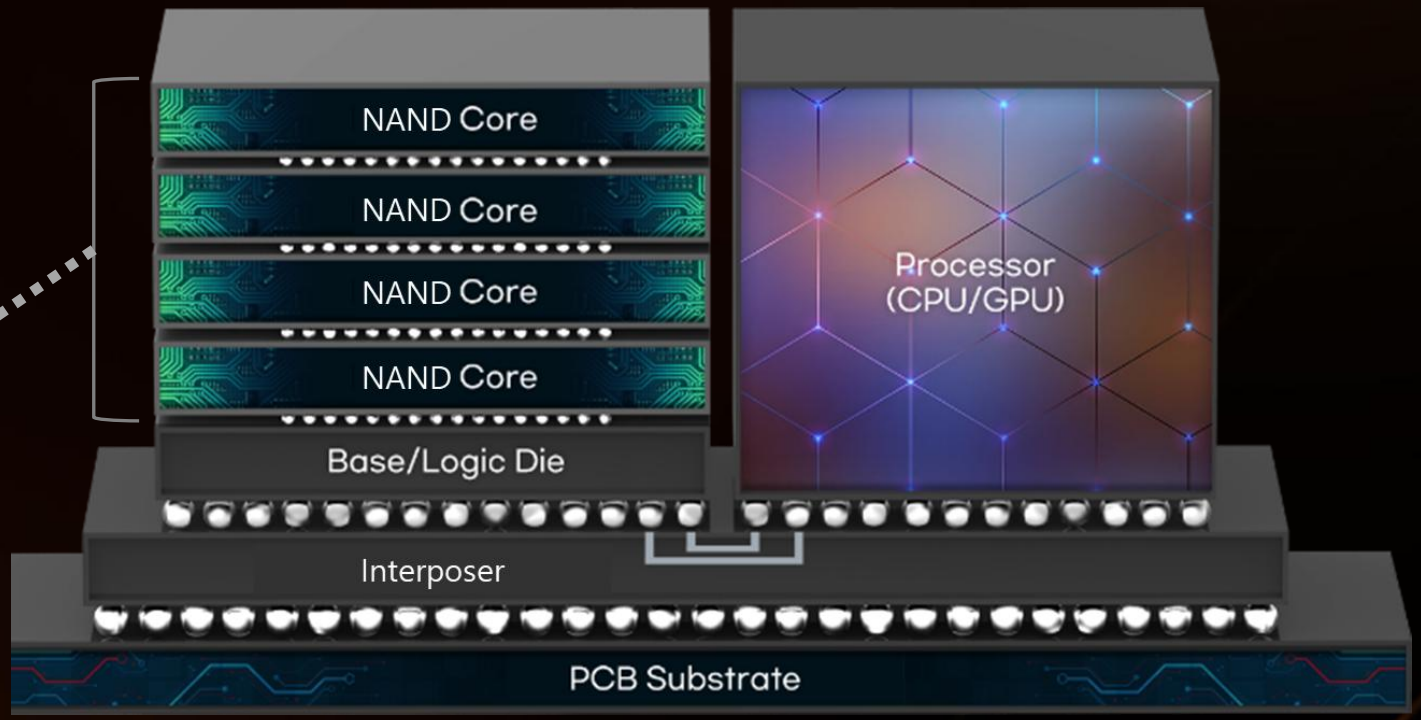
# How ‘HBF’? Constraints

- NAND Flash : Various constraints

✓ NANAD Media characteristics
- **Access Latency**
- **Endurance**
- **Operation Granularity**
- **Error Recovery Scheme**

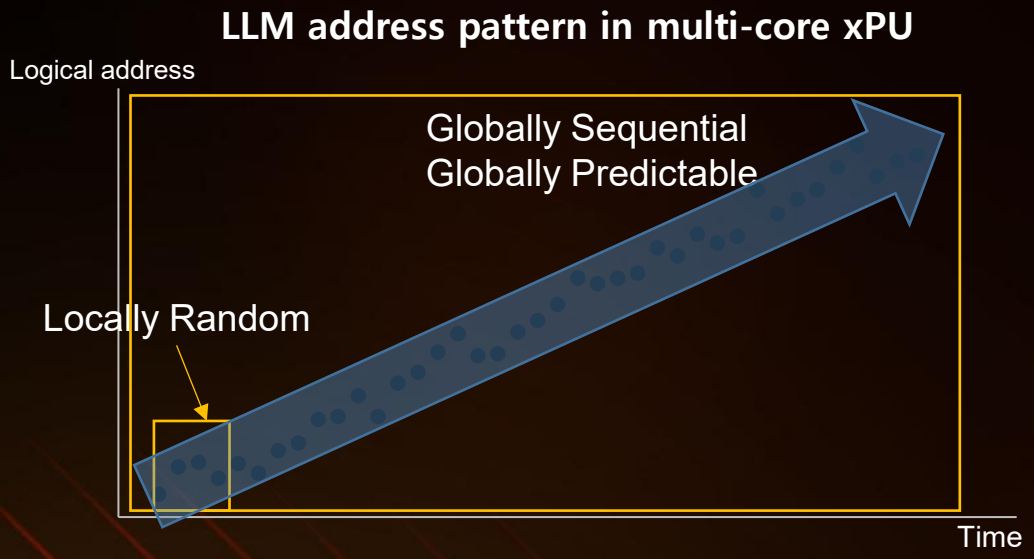
…
- **Wear Leveling? GC?**

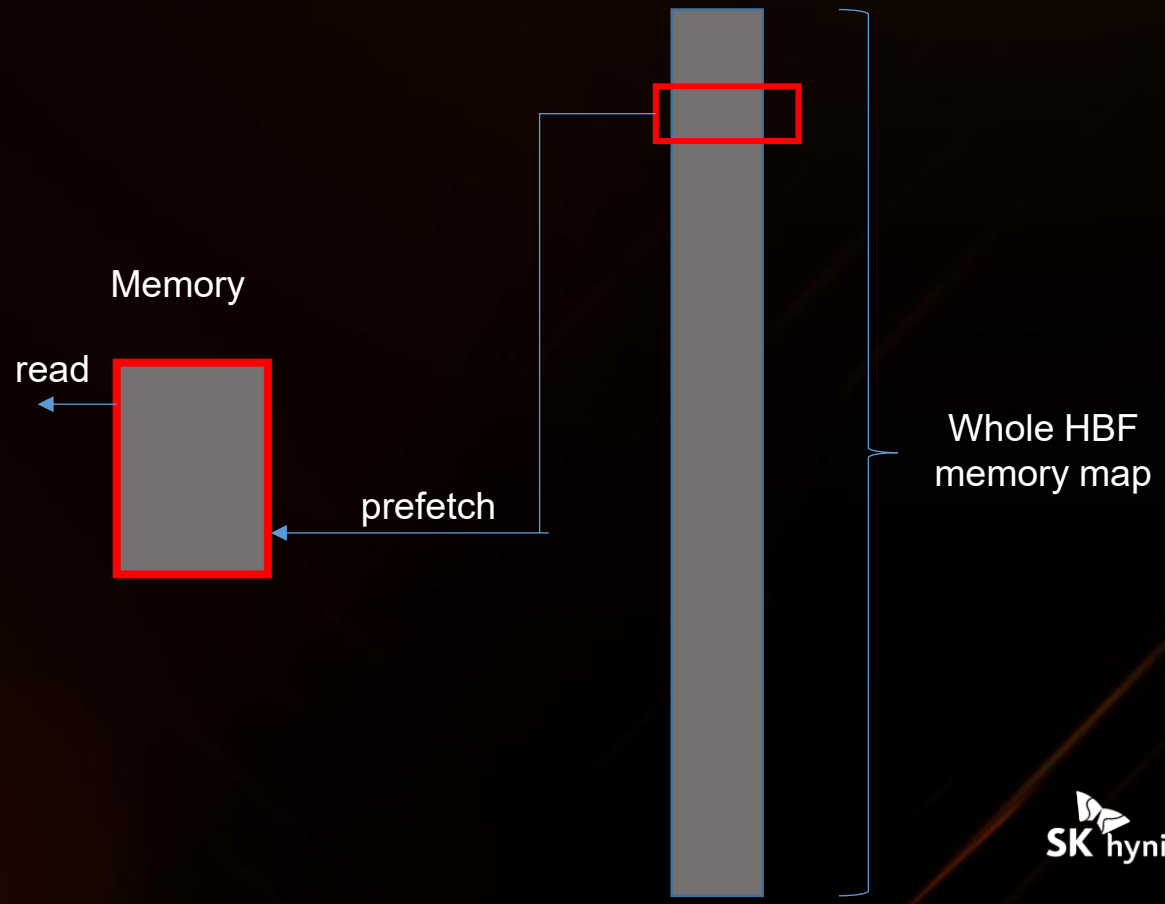
…



# How ‘HBF’? Access Latency hiding

- LLM in multi-core xPU has Locally Random, Globally Sequential address pattern
  - ✓ Predictable address pattern due to LLM algorithm
- Prefeching Memory buffer
  - ✓ NAND latency hiding



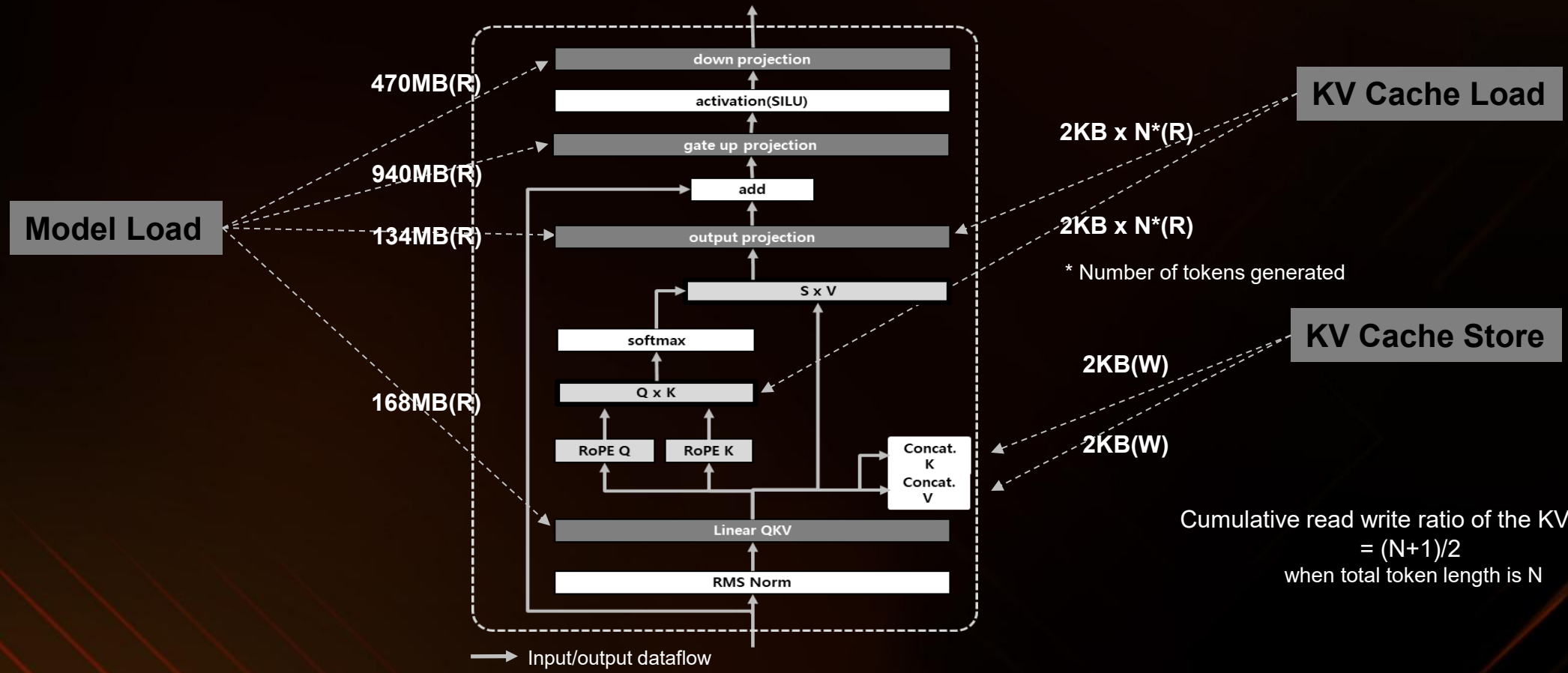
# How ‘HBF’? Read/Write amount of LLM Weight & KV Cache

- Amount of memory access required to generate 1 token
  - ✓ Model weight read: 141GB (1712MB * 80 block+ etc)
  - ✓ KV Cache write : 320KB (4KB * 80 block)
  - ✓ KV Cache read : 1.6GB (320KB * 5Kth token)

**Llama 70B Inference Blocks**

**80-blocks → 1-token generation**

**(R) : Read**
**(W) : Write**



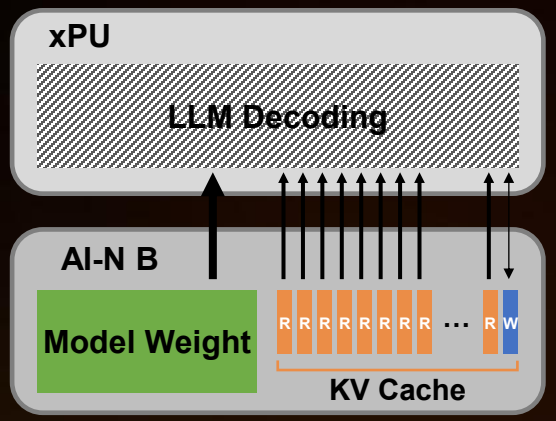

Cumulative read write ratio of the KV cache
= (N+1)/2
when total token length is N

# How ‘HBF’? Endurance Problem?

- The LLM inference : Read Intensive

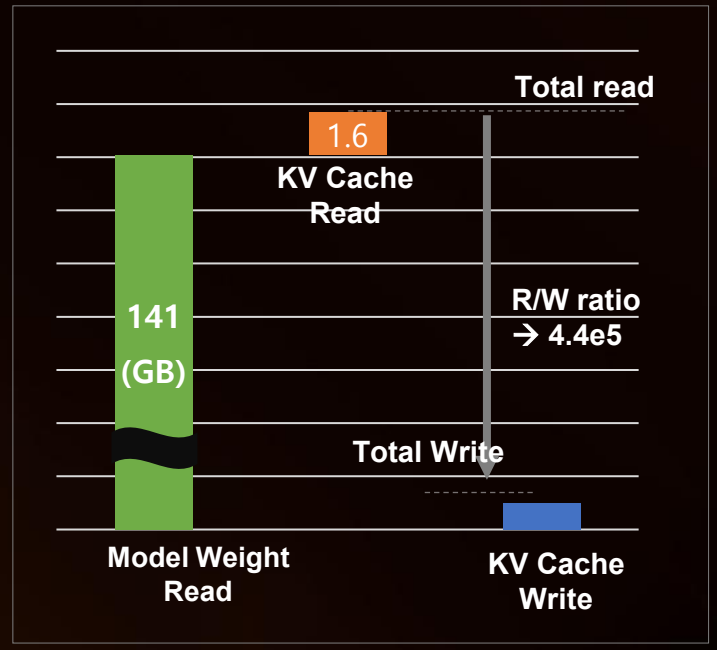
## Memory Access on Token Generation

On Token Generation :
Model Weight Read >> KV Cache Read > KV Cache Write



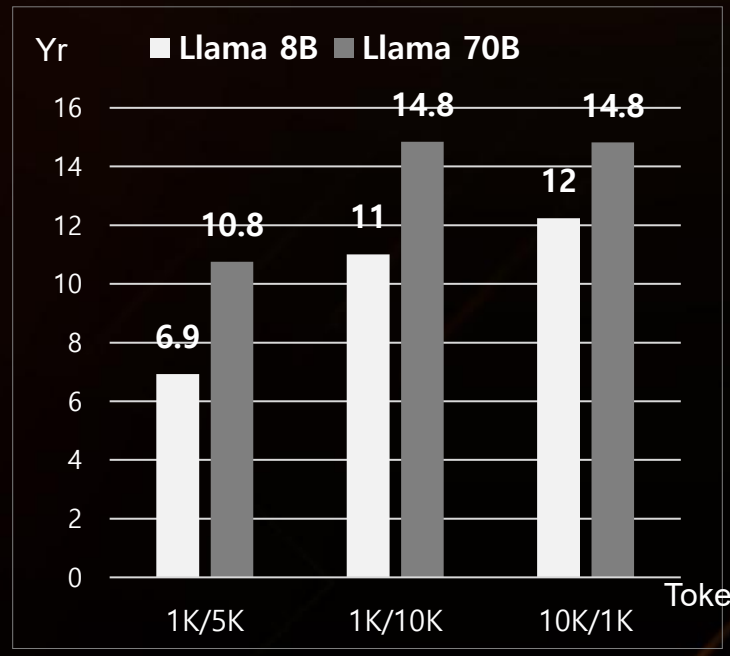

## Read/Write amount

5K-th Token Generation (Llama 70B)




## Expected Life-Span

512GB, 2TB/s BW, 200K PE Cycle, 128 Batch



# How 'HBF'? Data Characteristics?

- HBF only? or HBF + DRAMs? Mapping to HBF or DRAM depending on data characteristics
  - ✓ Read Intensive, Deterministic Access Pattern Data : Model weight & KV cache → HBF
  - ✓ Random Access, Immediate Data including code memory → DRAM

Llama 70B Operation size (1 Decoder block)

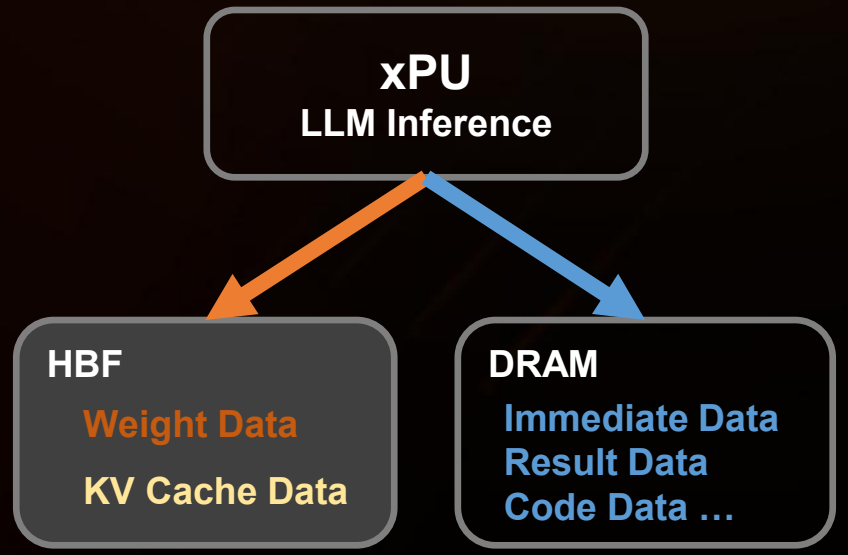
| R/W | Layer / Data | RMS Norm | Attention | | | | | | | | | add | MLP | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Linear (QKV) | ROPE Q | ROPE K | Q × K | softmax | S × V | Linear (out_proj) | Concat. K | Concat. V | | FFN (up) | act | FFN (down) |
| Read | Input | 16KB | 16KB | 16KB | 2KB | 16KB | 128KB* | 128KB* | 16KB | 2KB | 2KB | 32KB | 16KB | 56KB | 56KB |
| | Weight | **16KB** | **160MB** | **256B** | **256B** | | | | **128MB** | | | | **896MB** | | **448MB** |
| | KV Cache | | | | | **10MB*** | | **10MB*** | | | | | | | |
| Write | Output | 16KB | 20KB | 16KB | 16KB | 2MB | 2MB | 16KB | 16KB | **2KB** | **2KB** | 16KB | 56KB | 56KB | 16KB |

**Linear, FFN (Feed-Forward Network), Attention (Q × K , S × V) Layer**

- ✓ Sequential Read Pattern, Read Intensive, Deterministic access
- ✓ These are the majority of total memory accesses

: Intermediate result data
: Fixed large weight data
: K, V Cache data

- Data size depends on the # of tokens generated
- Input 1K, output 5K




High capacity & High BW

143GB @ Llama 70B (1 batch, 1K/5K tokens)

Low capacity & Low BW

1.3 GB @ Llama 70B (1 batch, 1K/5K tokens)

SK hynix

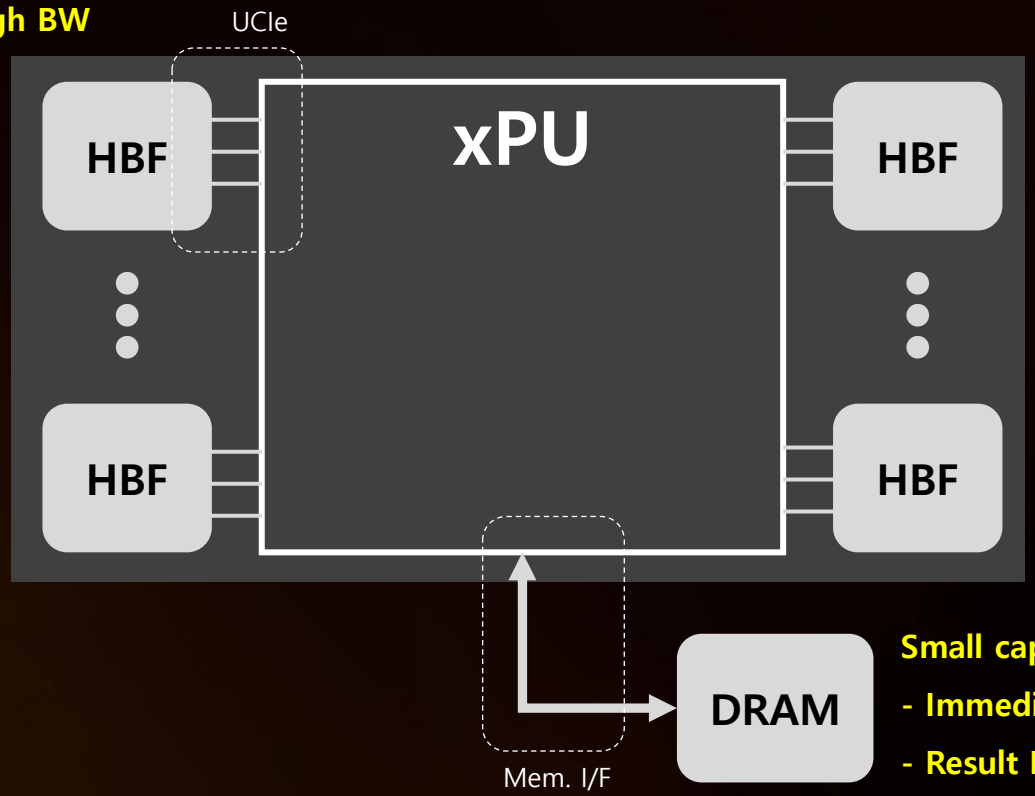
# How ‘HBF’? XPU + HBF architecture?

- HBF together with DRAM
  - ✓ HBM, HBF and DRAM each have different strength
  - ✓ All three can be used for more optimal AI inference services

**Large capacity & High BW**

**- Weight Data**

**- KV Cache Data**




**Small capacity & Low BW**

**- Immediate Data**

**- Result Data**

**- Code Data …**

SK hynix

# End of Document