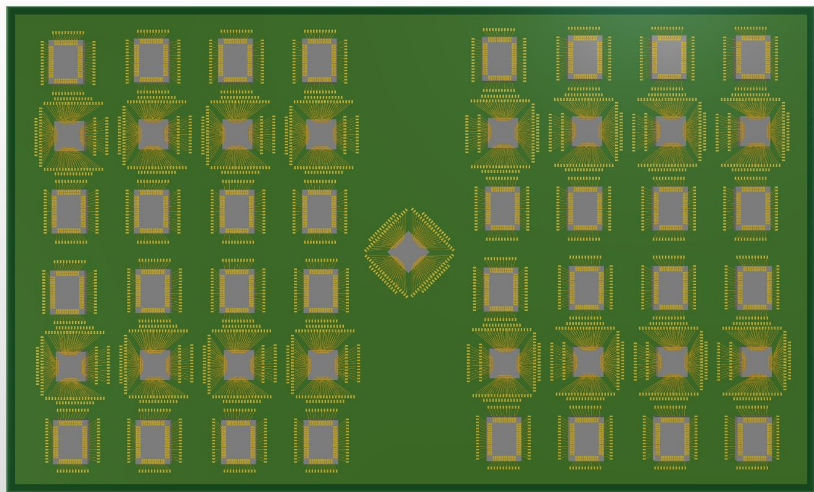
µVLA: Designing a Multi-Chiplet 16nm SoP with Heterogeneous Memory on Package (HMoP) to Enable Real-Time Physical AI on the Edge

Christian Kubicka

Stanford University

Executive Summary

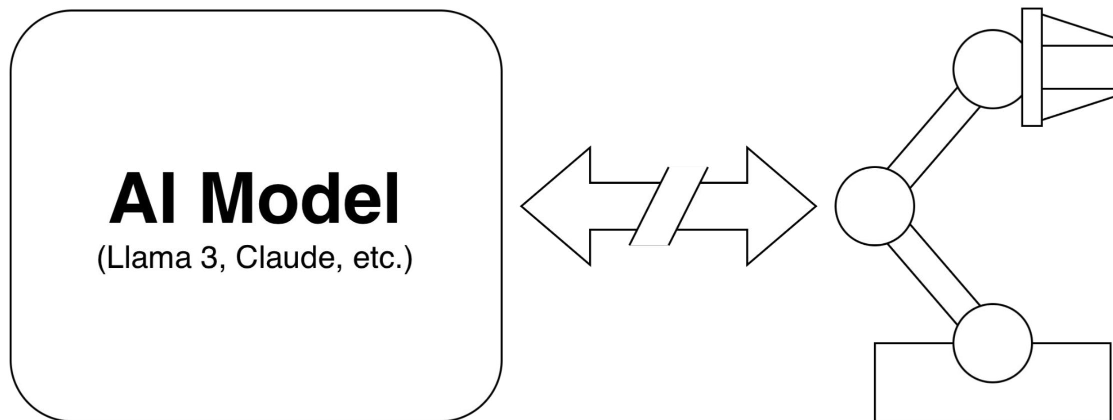


- **Summary:** µVLA is a heterogeneous multi-chiplet system targeted at Physical AI.
- **Tapeout Date:** December 2026
- **49 Chiplets Incorporated:**
  1. 16x 1T1C DRAM
  2. 16x NOR Flash
  3. 16x Custom TSMC 16nm ASIC (Spoke Chiplet)
  4. 1x Custom TSMC 16nm ASIC (Hub Chiplet) with 3T GCRAM
- **Main Problems Addressed:**
  - Low performance of existing solutions due to high end-to-end latency on Physical AI workloads.
  - High power consumption of existing solutions on Physical AI workloads.
- **Our Proposed Solution:**
  - Integration of differentiated memory technologies that align with data persistency.
  - Custom compute chiplets specifically designed to minimize latency.
  - Custom chiplet topology optimized for end to end latency.

What makes VLA-style Physical AI unique as a workload?

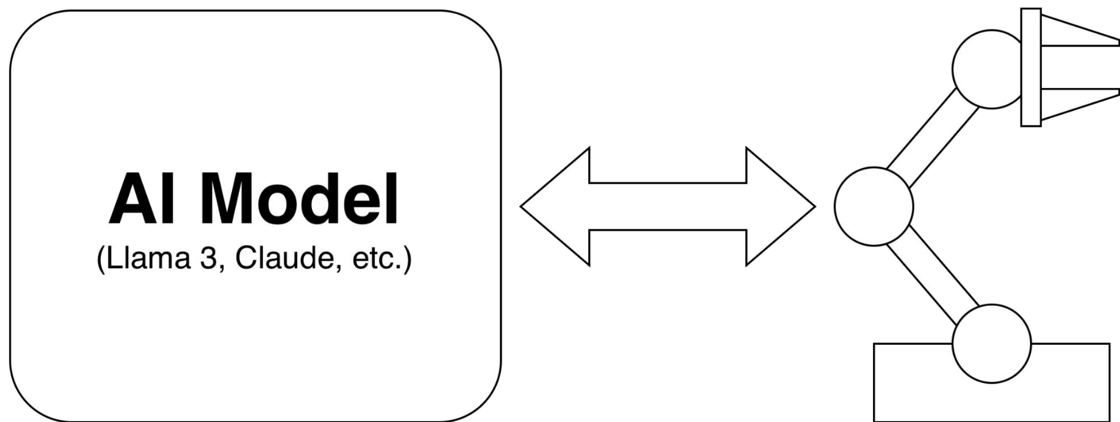
What is Physical AI?

- AI Models are great at a variety of tasks: coding, data retrieval, etc. However, by themselves, they lack the ability to interact with the real world.



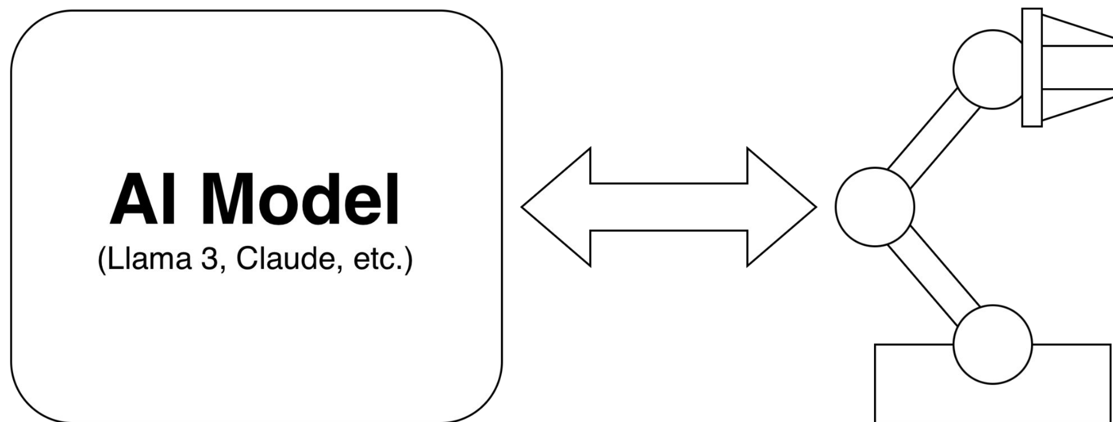
What is Physical AI?

- AI Models are great at a variety of tasks: coding, data retrieval, etc. However, by themselves, they lack the ability to interact with the real world.
- Physical AI is the concept of allowing AI Models to interact with real-world environments for the purpose of accomplishing some task.



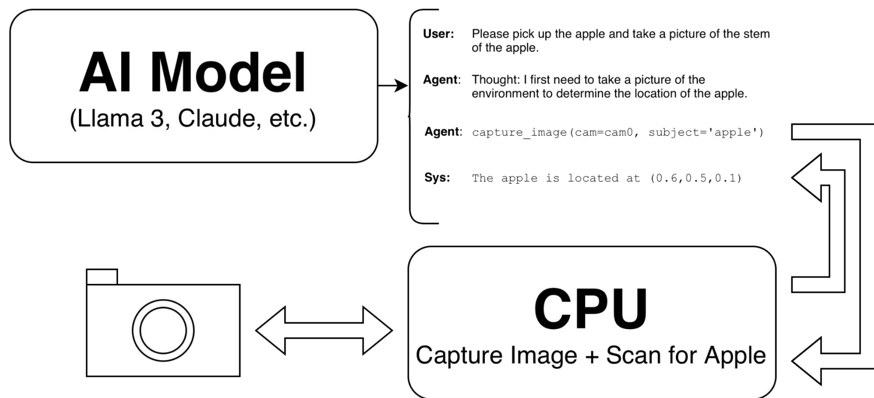
What is Physical AI?

- AI Models are great at a variety of tasks: coding, data retrieval, etc. However, by themselves, they lack the ability to interact with the real world.
- Physical AI is the concept of allowing AI Models to interact with real-world environments for the purpose of accomplishing some task.
- How can this be accomplished?



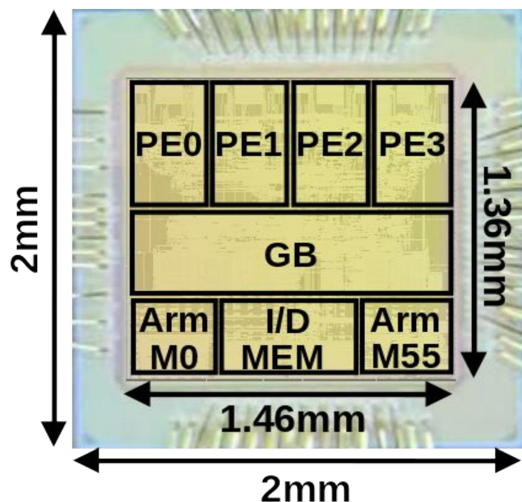
In the beginning, there was Agentic AI for Robotics

- The first method used to allow AI Models to interact with the real world was Agentic AI.
- The AI Model is trained to initiate a “function call” when it needs to complete a task in the real world.
- This “function call” is then executed by the CPU which executes a predesigned program to accomplish the requested task



Agentic AI for Robotics Cont.

- This was the approach which was the focus of our 7nm Tapeout: **µAgent** (Presented at VLSI 2026)



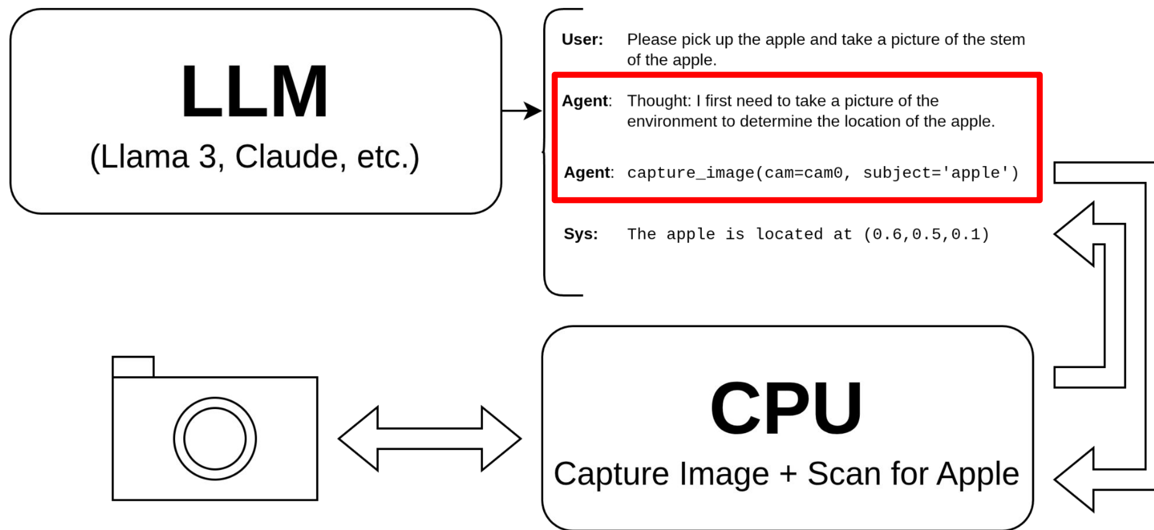
Process		TSMC 7nm
Area	Die	$4mm^2$
	Accel.	$1.36\ mm^2$
On-chip memory		780KB
Off-chip memory	Type	HyperRAM
	Size	64MB
Supply Voltage		0.51-0.9V
Clock Frequency		26-1220MHz

Clock Frequency	26-1220MHz
Data Precision	Block-wise[1)] **4b(A) 2-4b(W)**
Block size	**8,16,32,64**
Power	20-257mW
[2)4)]Peak Throughput	0.63TOPS
[2)3)5)]Peak Energy Efficiency	6.50TOPS/W (SoC)
[2)4)]Peak Area Efficiency	$0.16TOPS/mm^2$ (SoC)

What are the Problems with Agentic AI for Robotics?

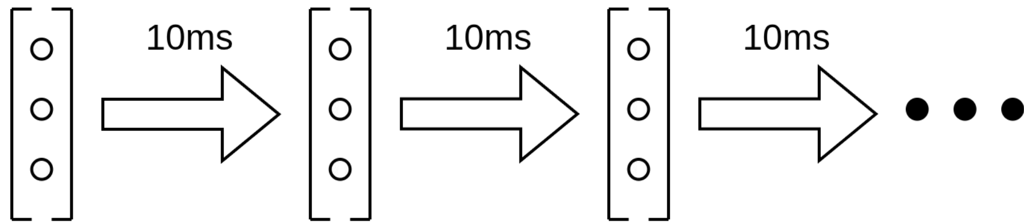
What are the Problems with Agentic AI for Robotics?

- Many tokens are being generated to take a single action. For µAgent one action took on average 30.8 tokens.



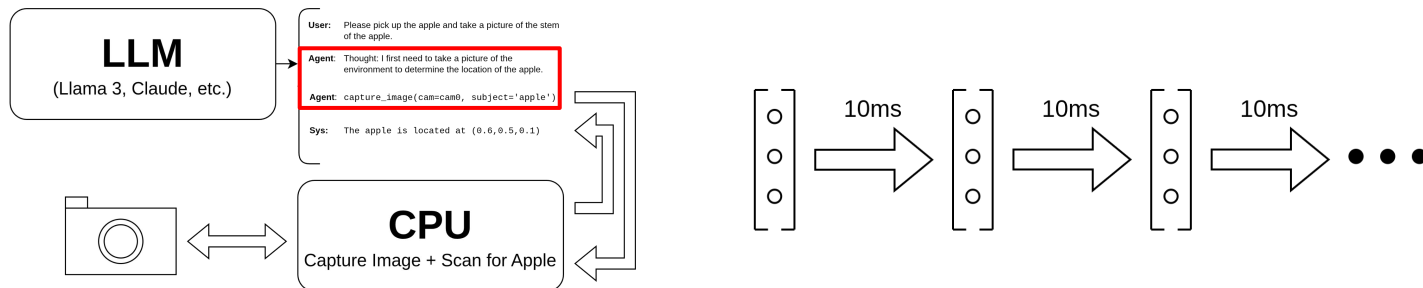
What are the Problems with Agentic AI for Robotics?

- Many tokens are being generated to take a single action. For µAgent one action took on average 30.8 tokens.
- The overall algorithm uses autoregressive token generation resulting in significant latency issues arising from memory bandwidth limitations.



What are the Problems with Agentic AI for Robotics?

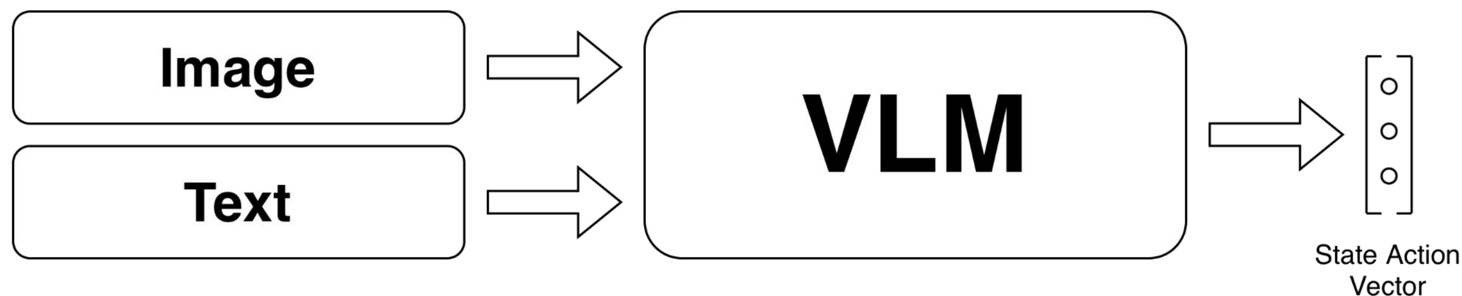
- Many tokens are being generated to take a single action. For µAgent one action took on average 30.8 tokens.
- The overall algorithm uses autoregressive token generation resulting in significant latency issues arising from memory bandwidth limitations.
- Visual Language Action (VLA) models specifically address these problems.



What is a Visual-Language-Action (VLA) Model?

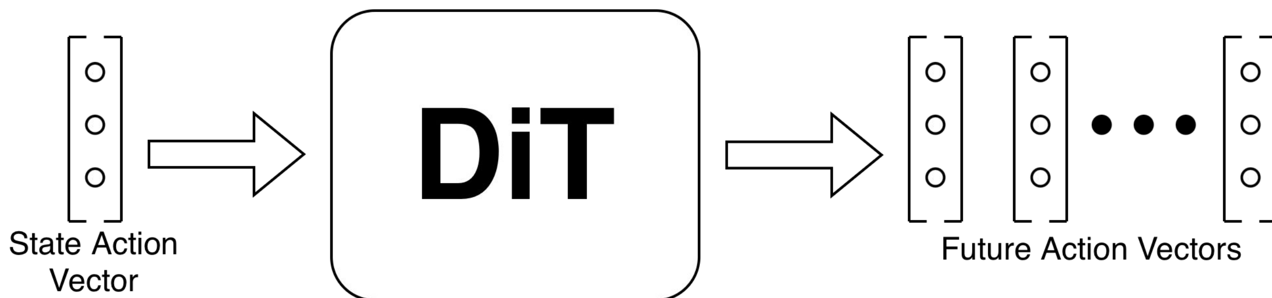
What is a Visual-Language-Action (VLA) Model?

- VLA models use a pre-trained **Vision-Language-Model (VLM)** generate a single high-level action vector from a combined prompt and state.



What is a Visual-Language-Action (VLA) Model?

- VLA models use a pre-trained **Vision-Language-Model (VLM)** generate a single* high-level action vector from a combined prompt and state.
- This high level action vector is then used as an input to a **Diffusion Transformer (DiT)** to extrapolate multiple new actions from higher-level vector.



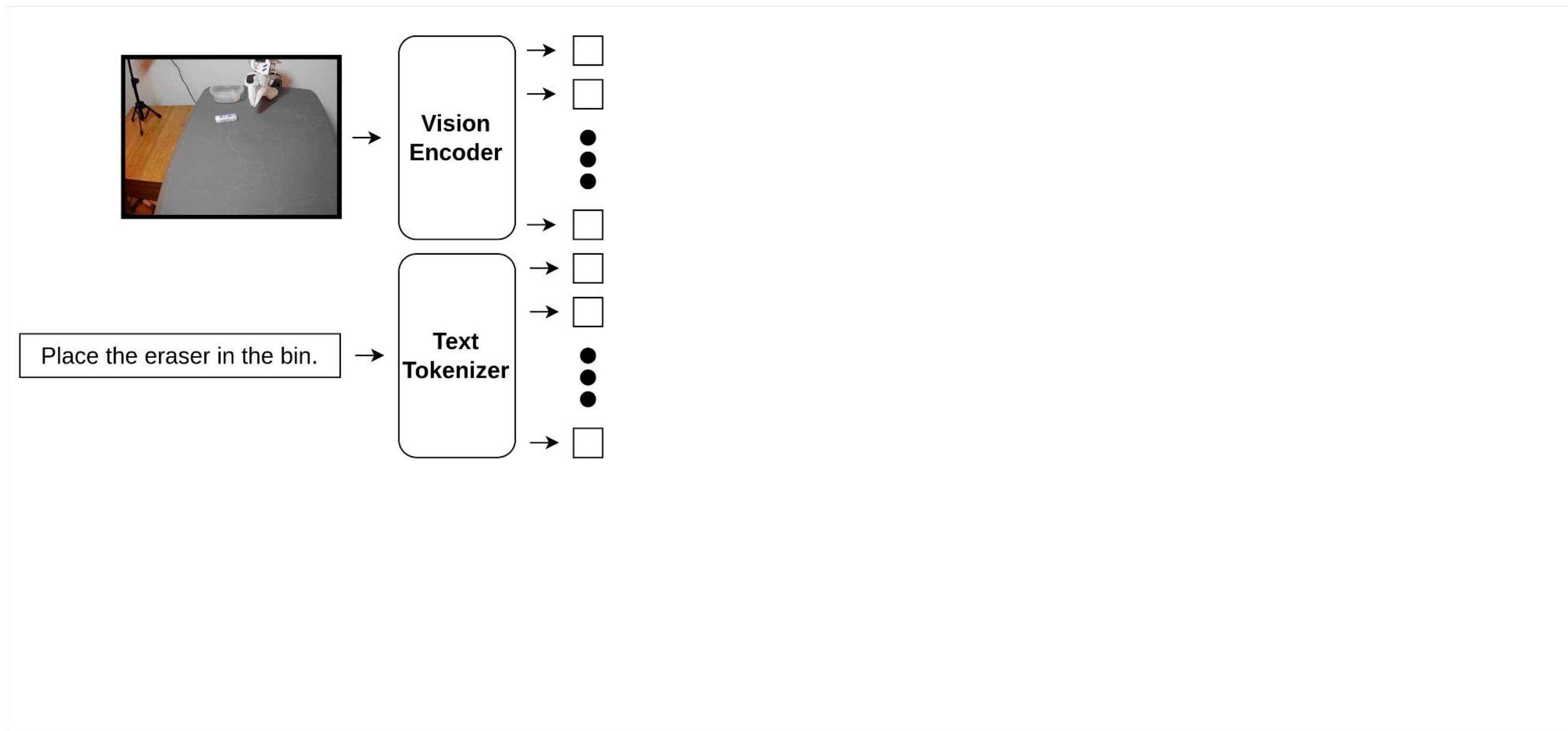
What is a Visual-Language-Action (VLA) Model?

- VLA models use a pre-trained **Vision-Language-Model (VLM)** generate a single* high-level action vector from a combined prompt and state.
- This high level action vector is then used as an input to a **Diffusion Transformer (DiT)** to extrapolate multiple new actions from higher-level vector.
- What does this actually look like?

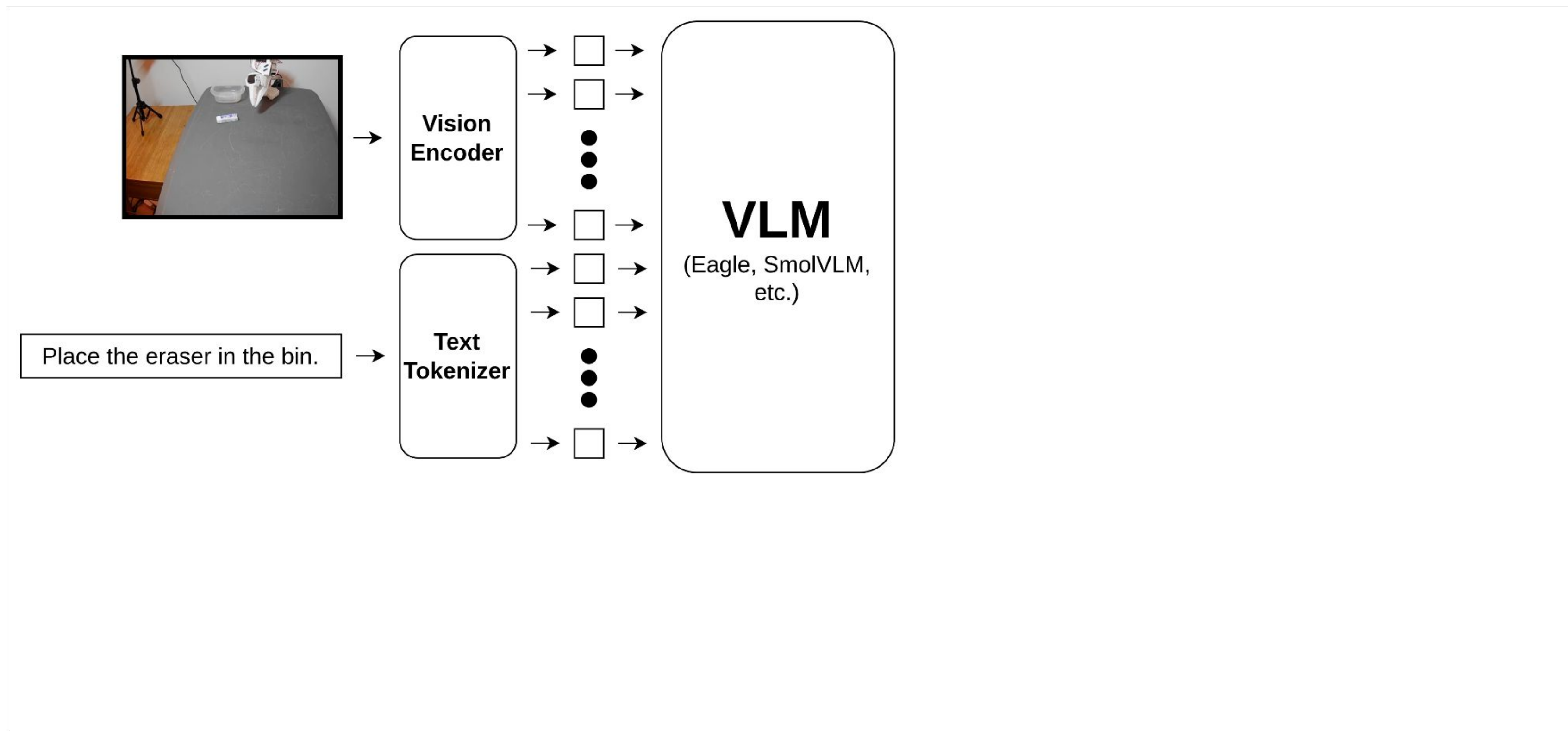
What is a Visual-Language-Action (VLA) Model?

Place the eraser in the bin.

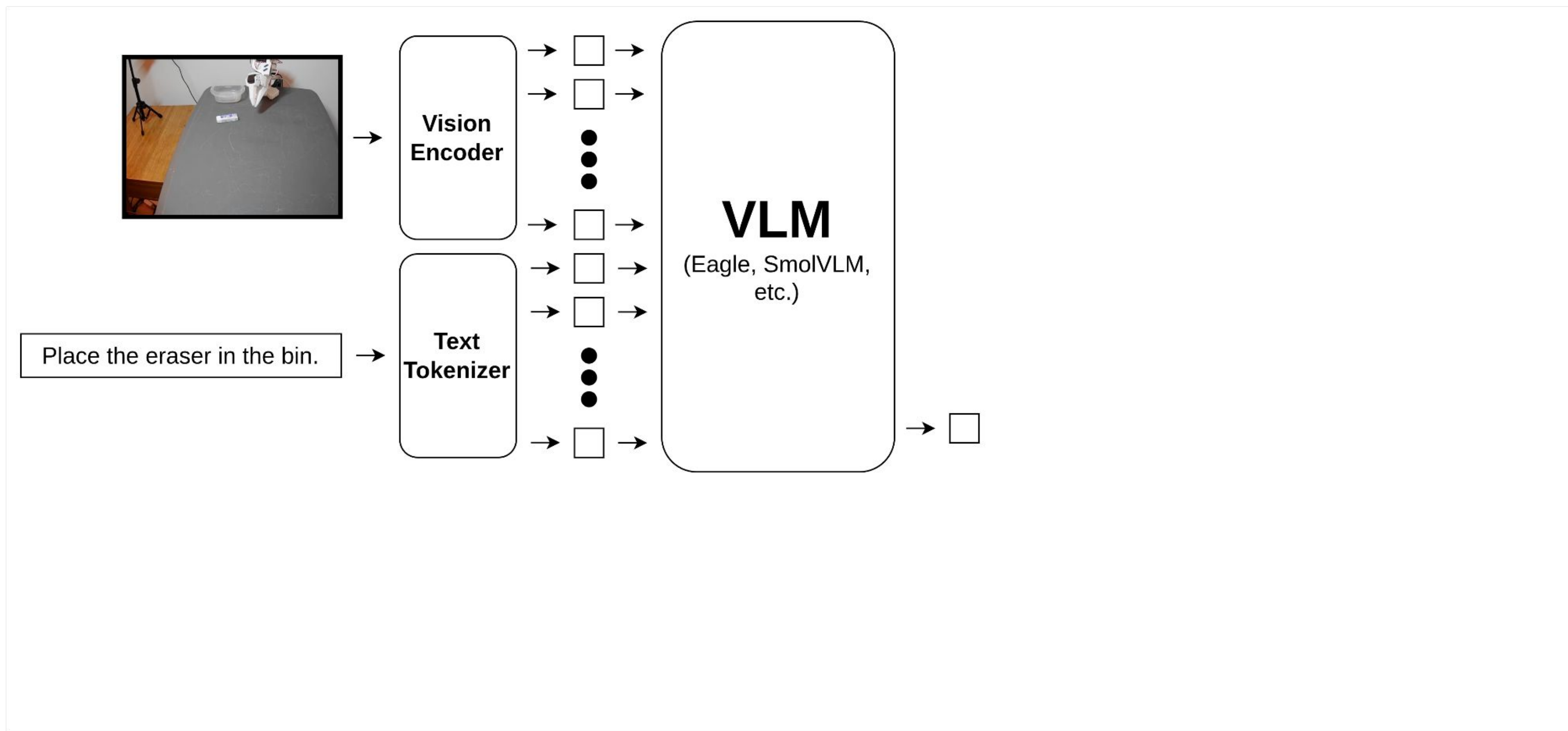
What is a Visual-Language-Action (VLA) Model?



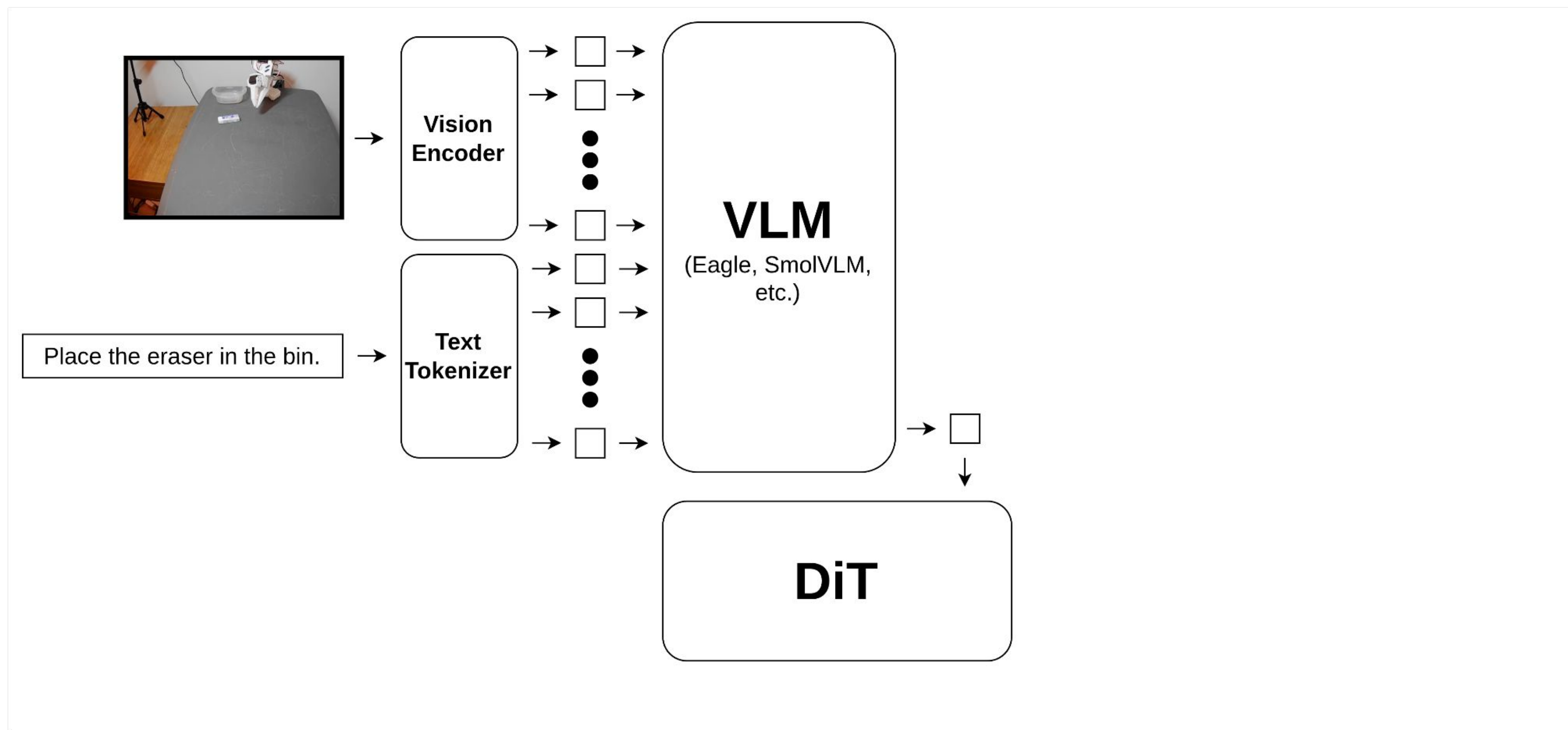
What is a Visual-Language-Action (VLA) Model?



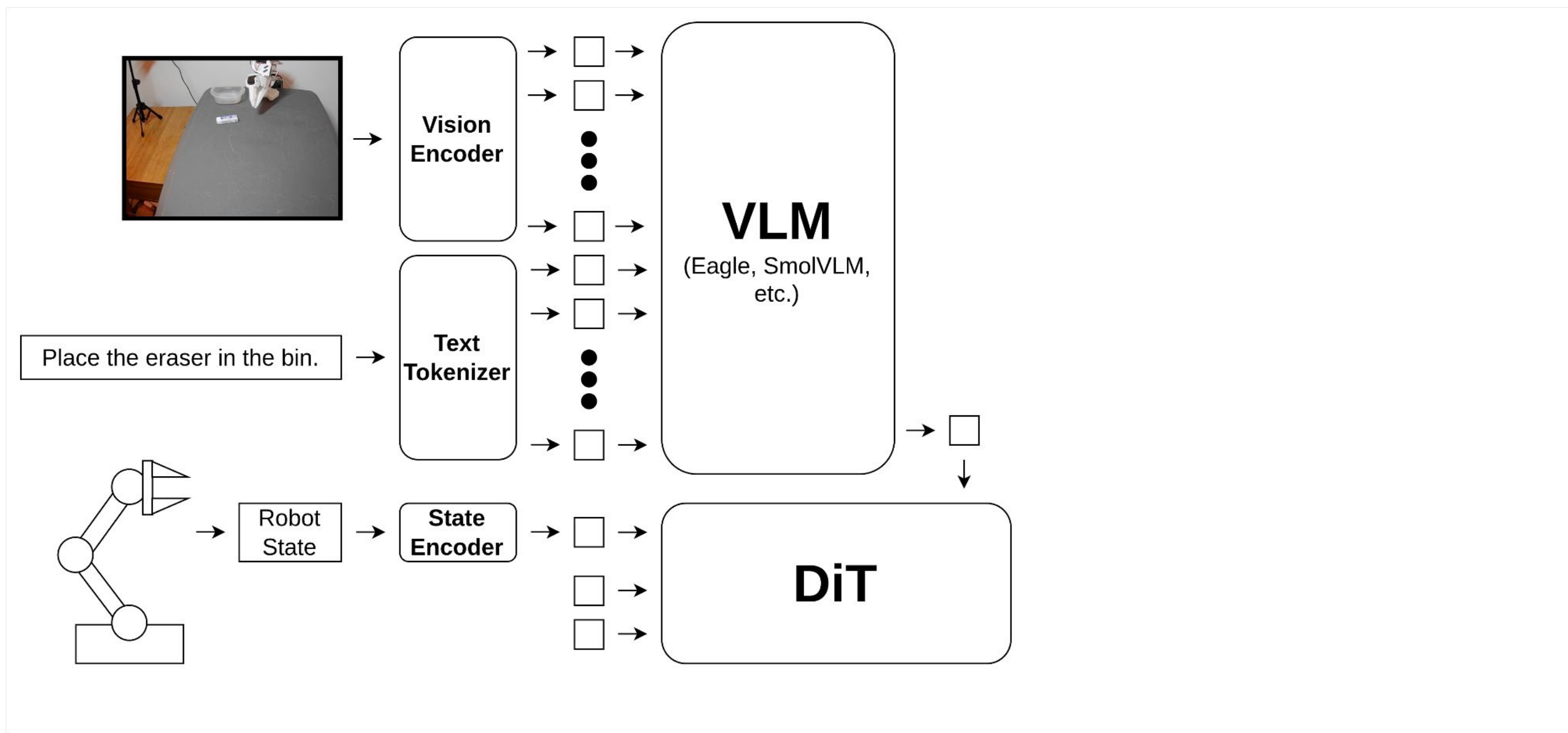
What is a Visual-Language-Action (VLA) Model?



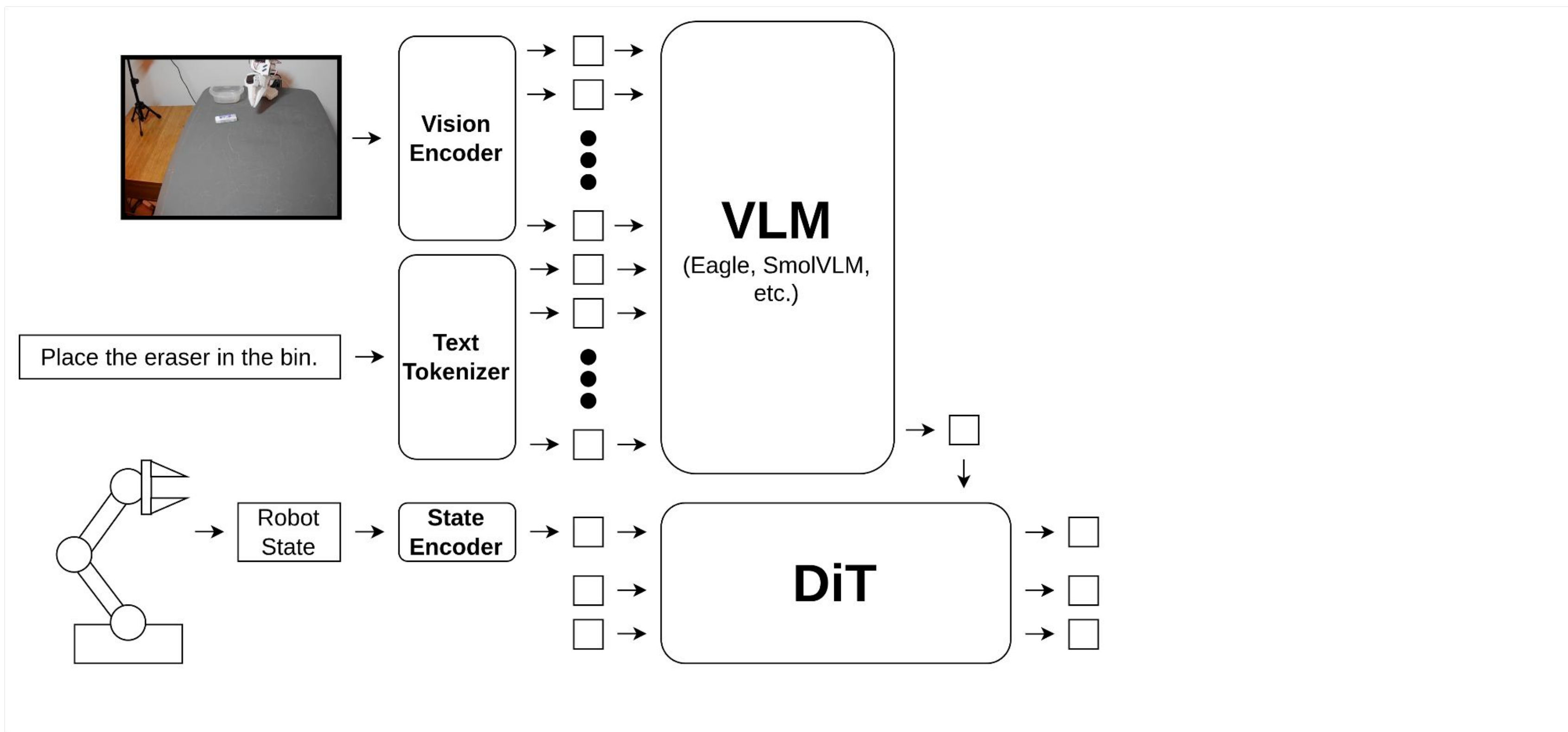
What is a Visual-Language-Action (VLA) Model?



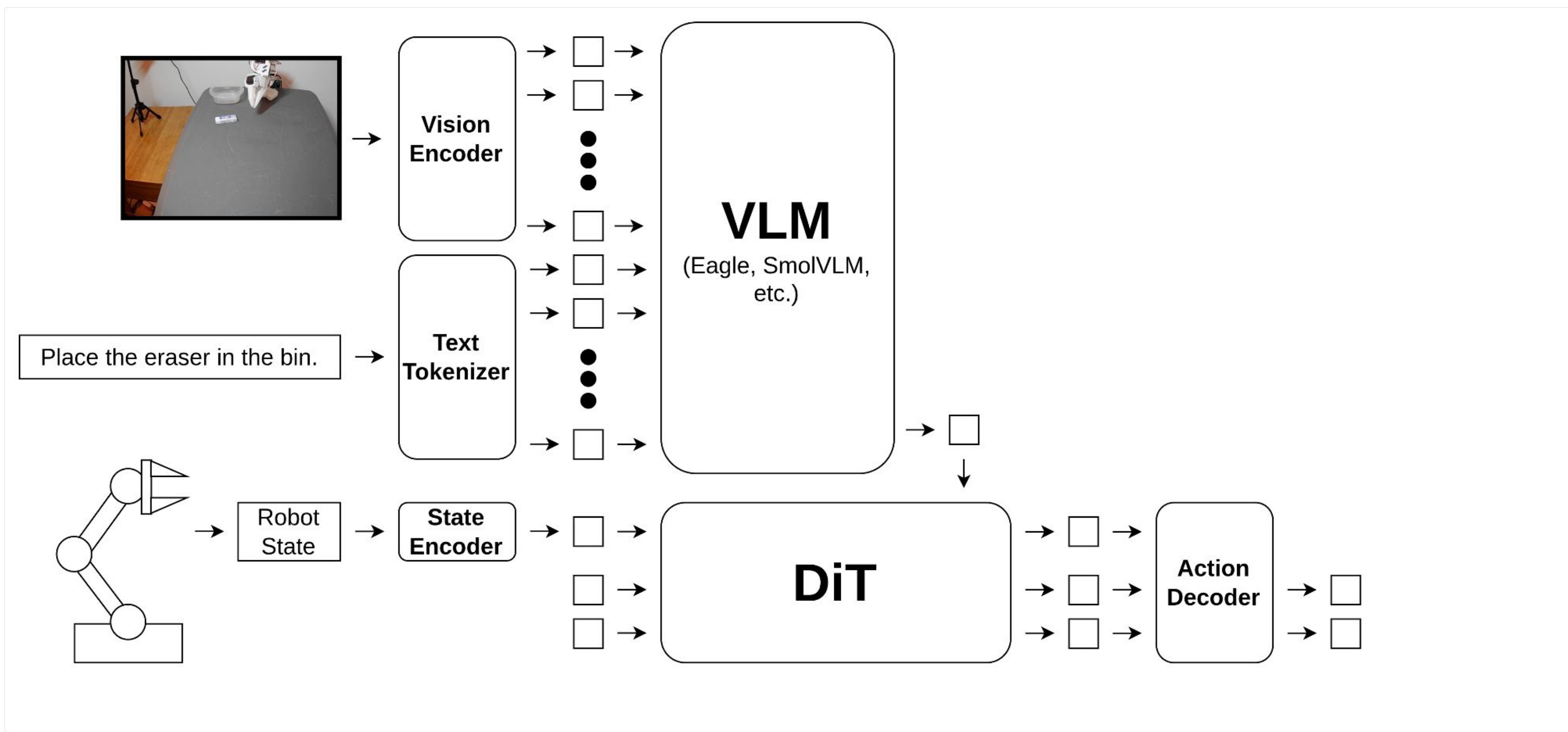
What is a Visual-Language-Action (VLA) Model?



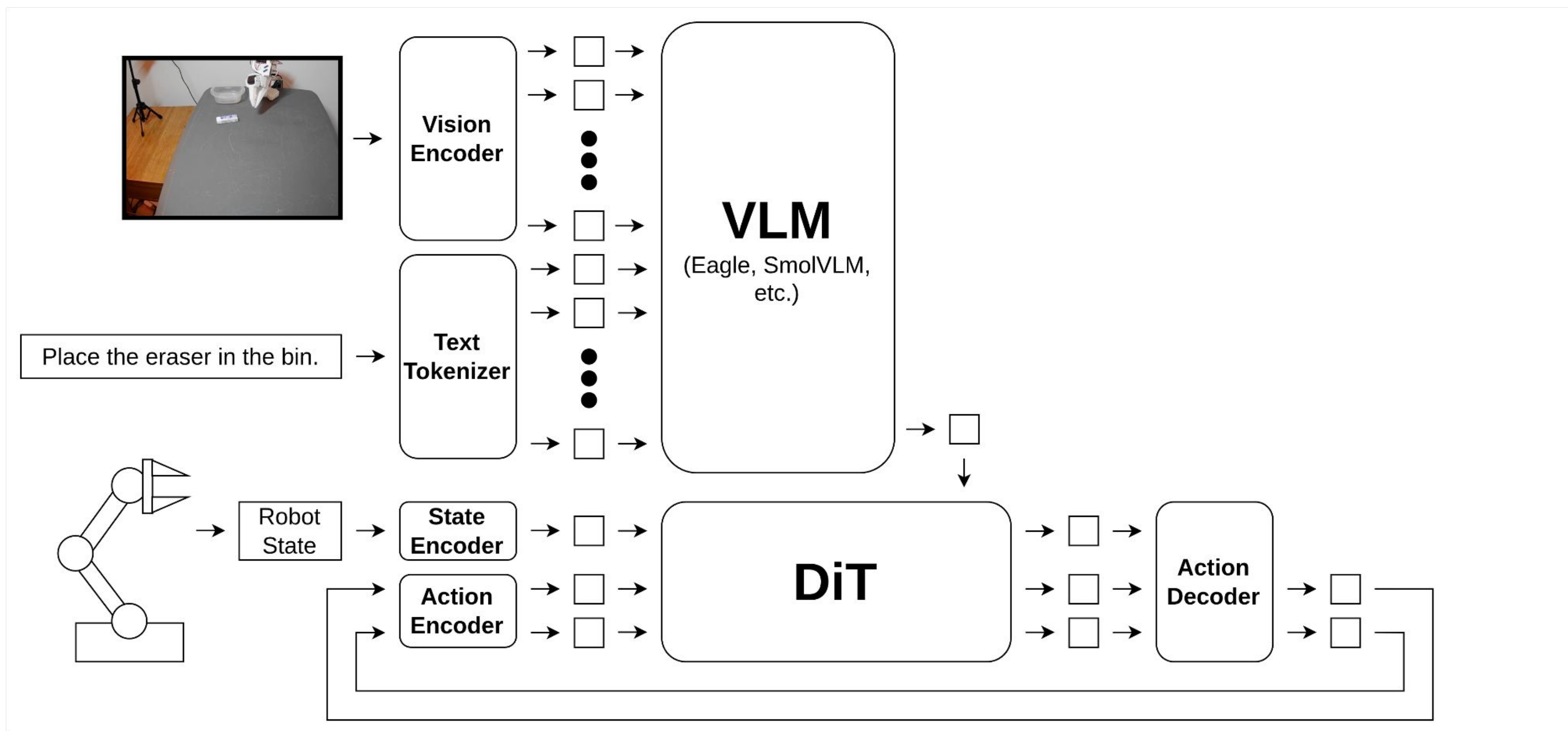
What is a Visual-Language-Action (VLA) Model?



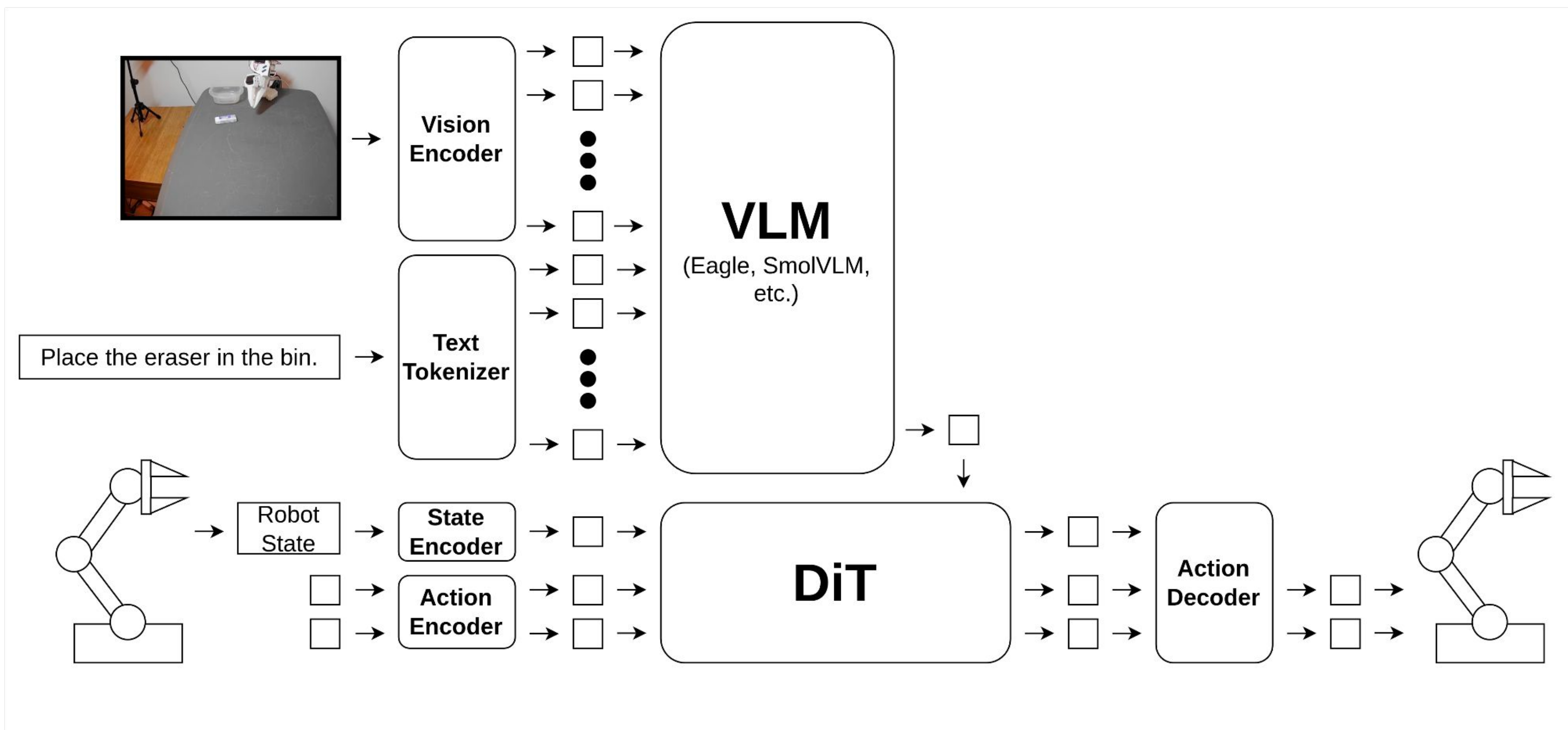
What is a Visual-Language-Action (VLA) Model?



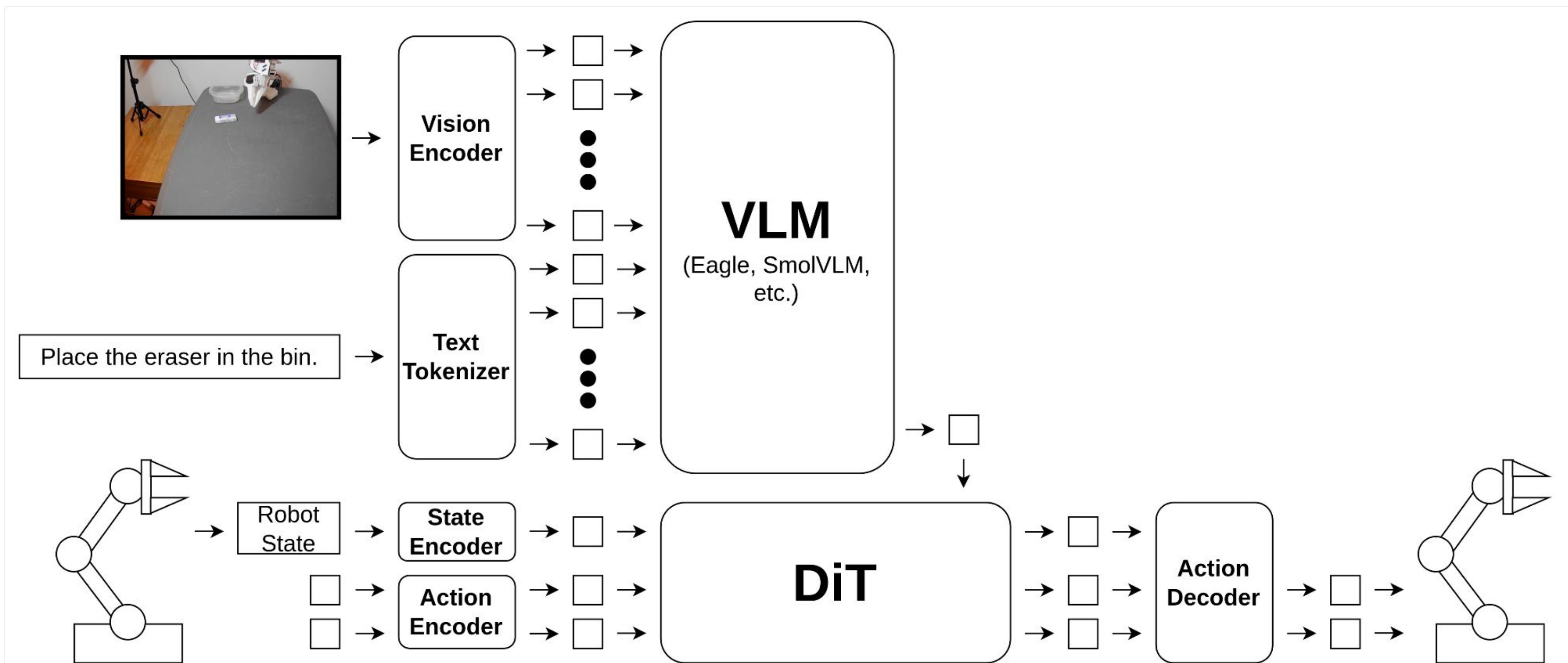
What is a Visual-Language-Action (VLA) Model?



What is a Visual-Language-Action (VLA) Model?



What is a Visual-Language-Action (VLA) Model?

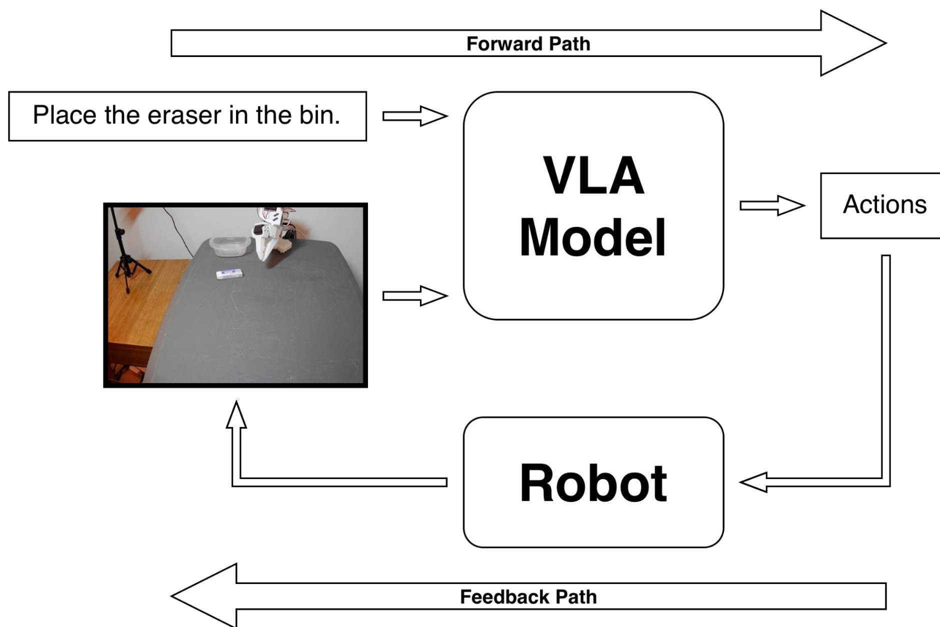


- This solves many of the original issues with Agentic AI for robotics.

What is the Catch?

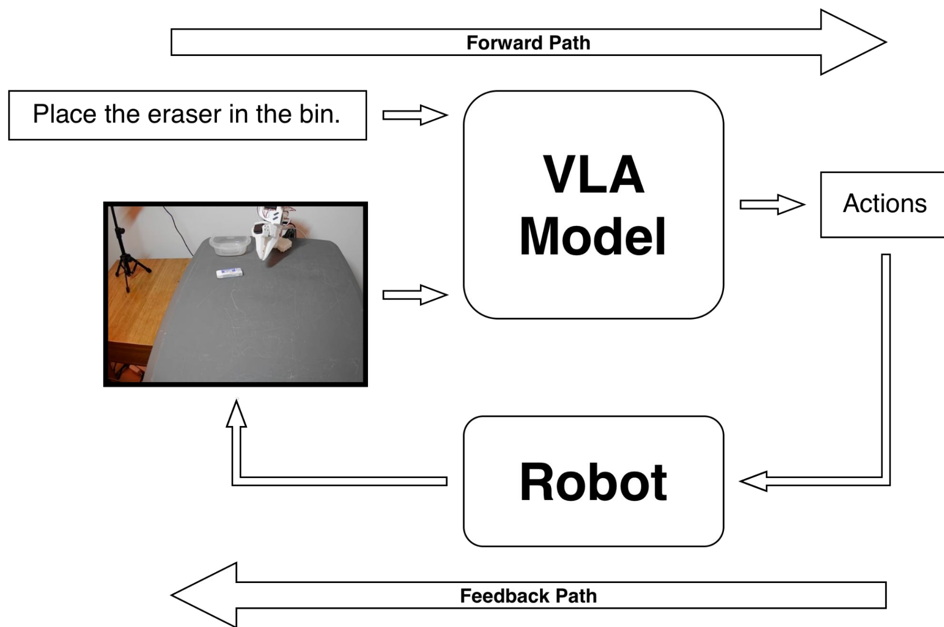
What is the Catch?

- This approach essentially implements closed-loop control.



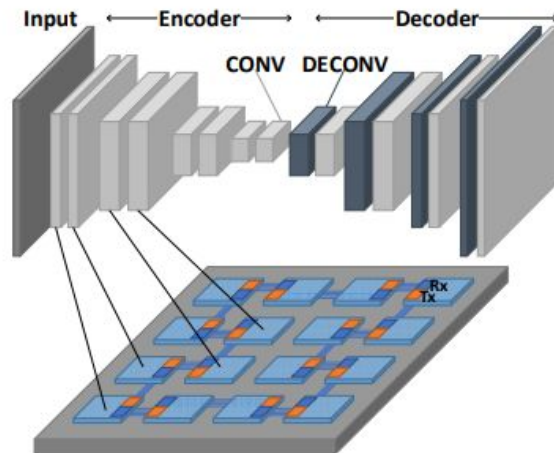
What is the Catch?

- This approach essentially implements closed-loop control.
- **Single-batch latency** is the primary design target in terms of performance, not throughput.



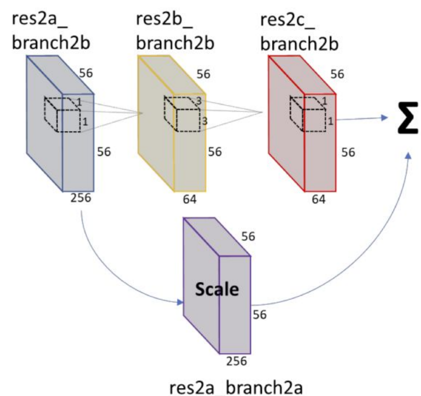
What does this mean for our preferable chiplet structure?

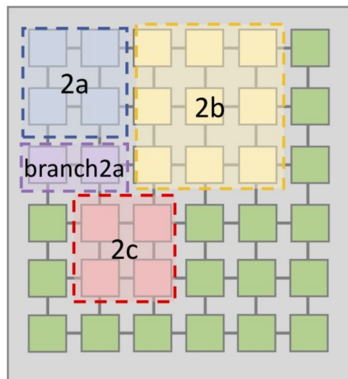
What do other works do?

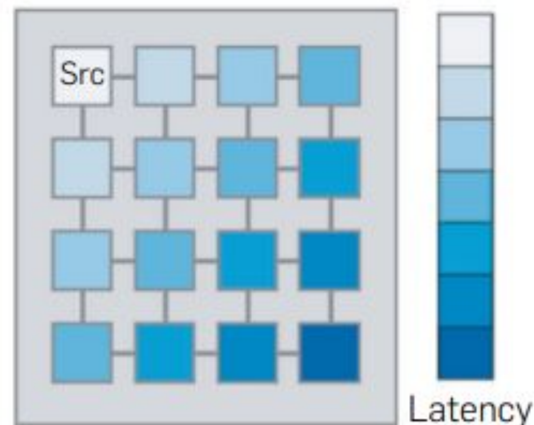


- Works such as NetFlex (VLSI 2022) map a single layer of a given DNN to one chiplet, then pass the activations through each chiplet sequentially. (Spatial Pipelining)

What do other works do?



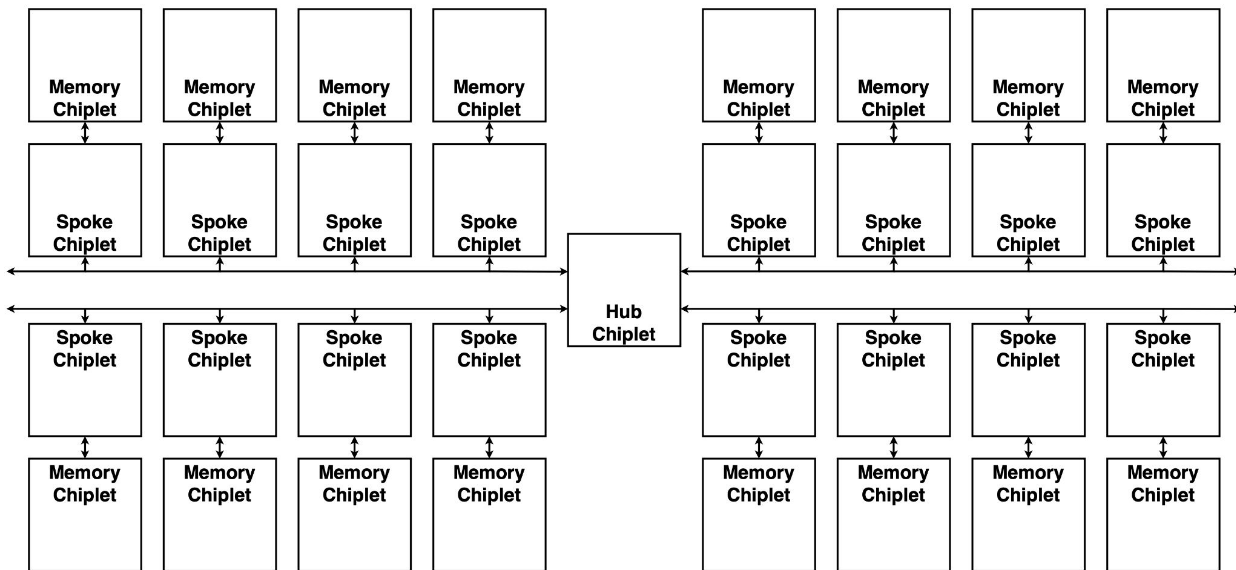




- SIMBA (VLSI 2019) has similar direct chip-to-chip connections. However, workload mapping is handled slightly differently. (Partial Spatial Pipelining)

Our Proposed Architecture

- Single batch utilization issues are addressed by allowing the Hub Chiplet to broadcast work directly to multiple chiplets at once.
- Lack of available memory is addressed through off-chip memories associated with each die.



How can we make use of a variety of memory technologies to enhance this specific chiplet topology?

Creating the Memory Architecture

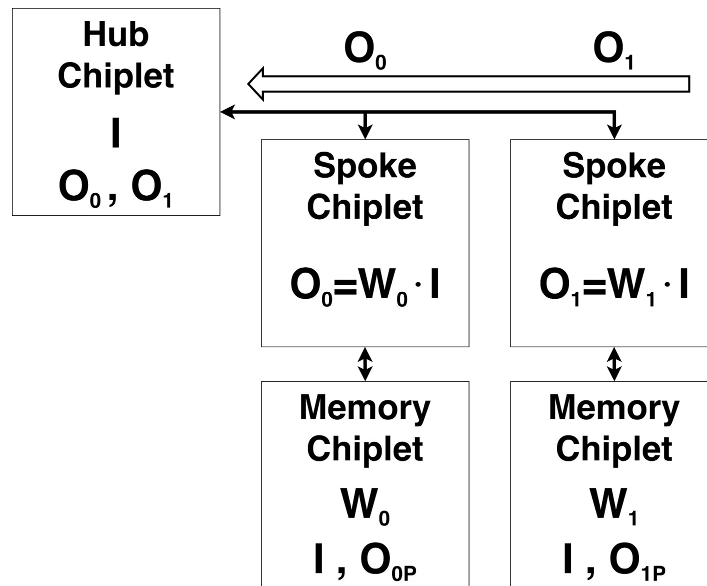
- When using this topology to run machine-learning-inference-type workloads, we can derive the following three classes of data.

Creating the Memory Architecture

- When using this topology to run machine-learning-inference-type workloads, we can derive the following three classes of data.
  - Class 1 - Local Read Only

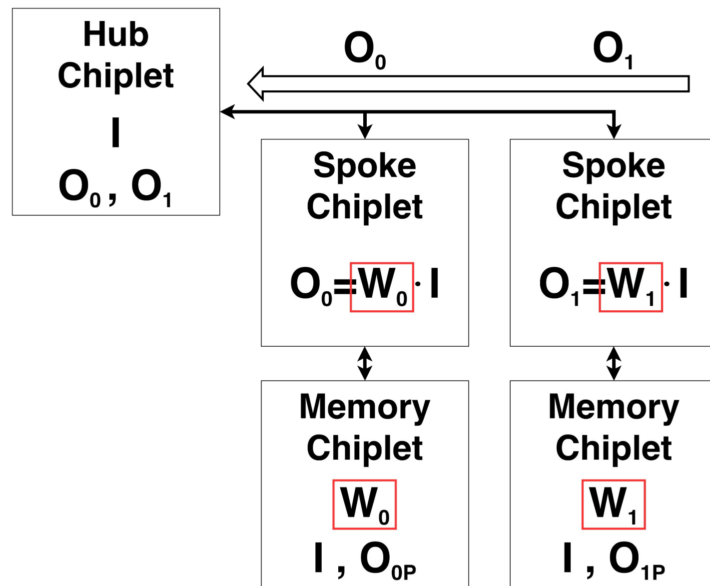
Creating the Memory Architecture

- When using this topology to run machine-learning-inference-type workloads, we can derive the following three classes of data.
  - Class 1 - Local Read Only



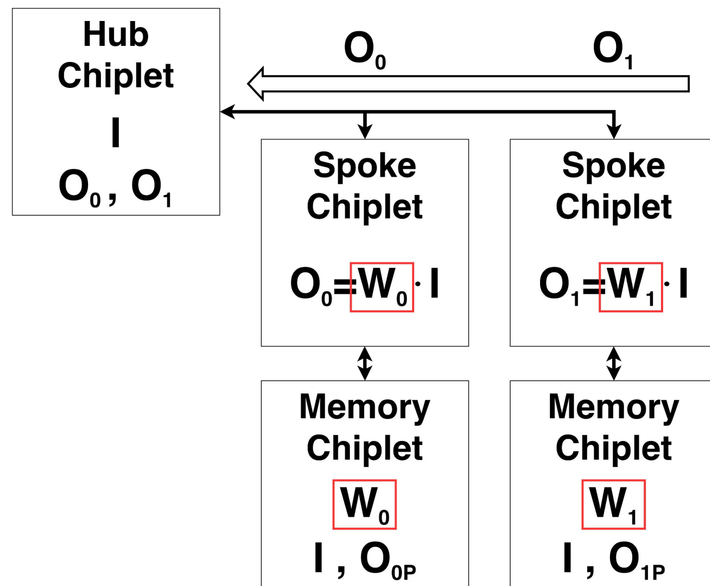
Creating the Memory Architecture

- When using this topology to run machine-learning-inference-type workloads, we can derive the following three classes of data.
  - Class 1 - Local Read Only



Creating the Memory Architecture

- When using this topology to run machine-learning-inference-type workloads, we can derive the following three classes of data.
  - Class 1 - Local Read Only **(NOR Flash)**

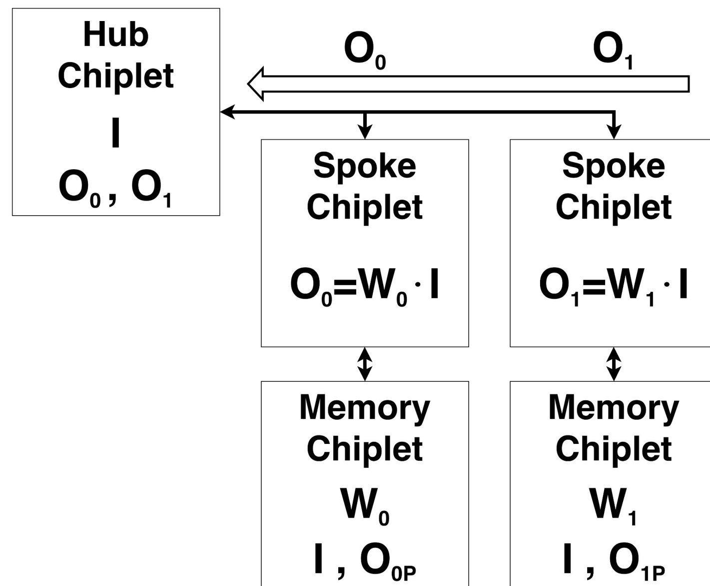


Creating the Memory Architecture

- When using this topology to run machine-learning-inference-type workloads, we can derive the following three classes of data.
  - Class 1 - Local Read Only **(NOR Flash)**
  - Class 2 - Local Scratchpad

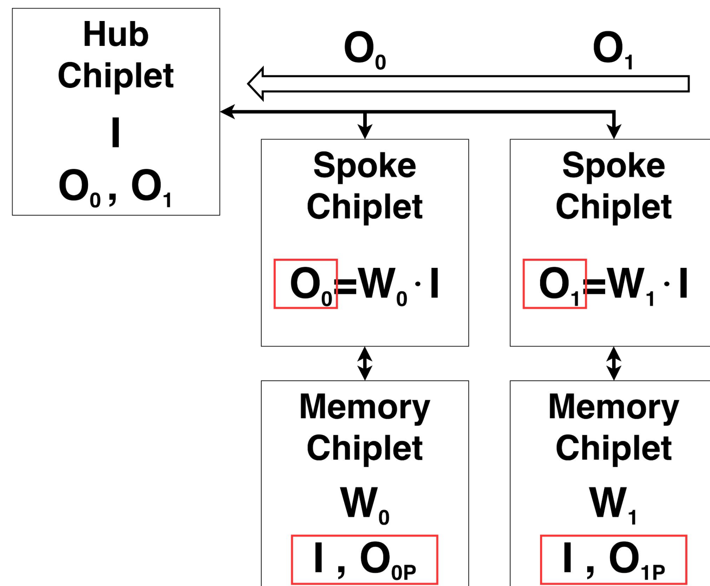
Creating the Memory Architecture

- When using this topology to run machine-learning-inference-type workloads, we can derive the following three classes of data.
    - Class 1 - Local Read Only **(NOR Flash)**
    - Class 2 - Local Scratchpad



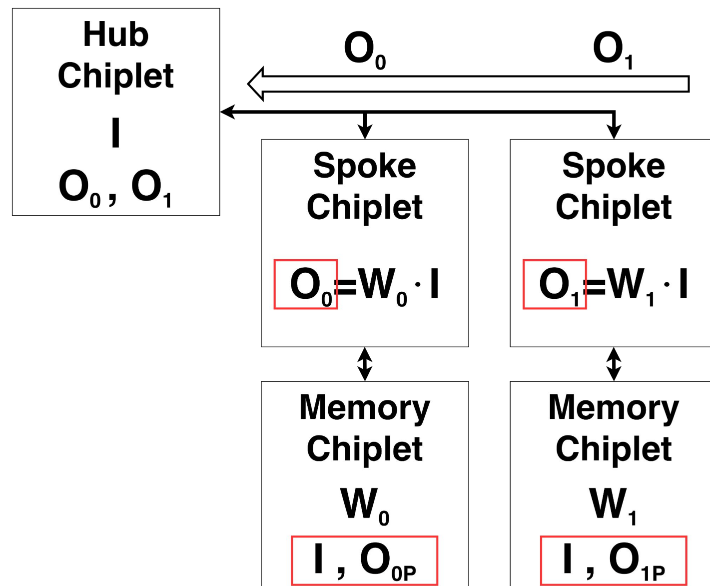
Creating the Memory Architecture

- When using this topology to run machine-learning-inference-type workloads, we can derive the following three classes of data.
  - Class 1 - Local Read Only **(NOR Flash)**
  - Class 2 - Local Scratchpad



Creating the Memory Architecture

- When using this topology to run machine-learning-inference-type workloads, we can derive the following three classes of data.
  - Class 1 - Local Read Only **(NOR Flash)**
  - Class 2 - Local Scratchpad **(1T1C DRAM)**

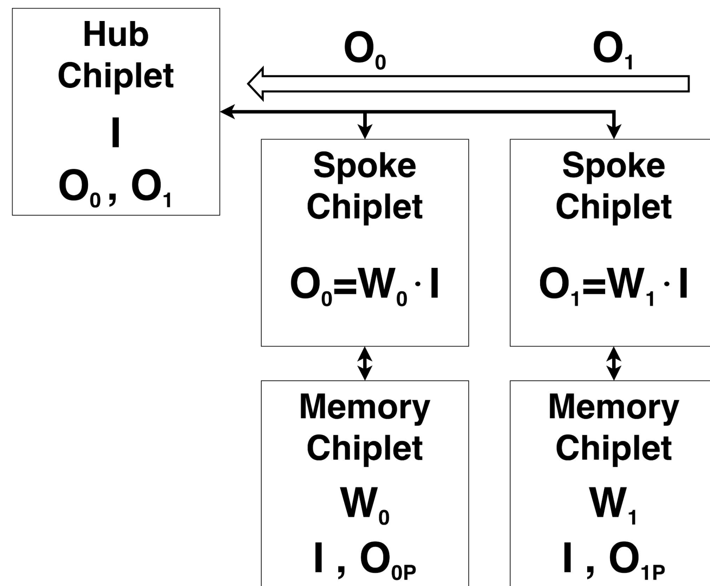


Creating the Memory Architecture

- When using this topology to run machine-learning-inference-type workloads, we can derive the following three classes of data.
  - Class 1 - Local Read Only **(NOR Flash)**
  - Class 2 - Local Scratchpad **(1T1C DRAM)**
  - Class 3 - Global Intermediates

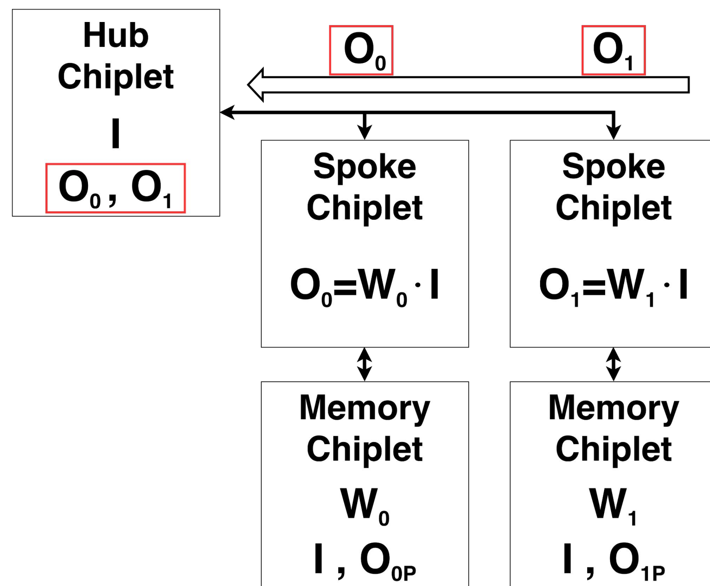
Creating the Memory Architecture

- When using this topology to run machine-learning-inference-type workloads, we can derive the following three classes of data.
  - Class 1 - Local Read Only **(NOR Flash)**
  - Class 2 - Local Scratchpad **(1T1C DRAM)**
  - Class 3 - Global Intermediates



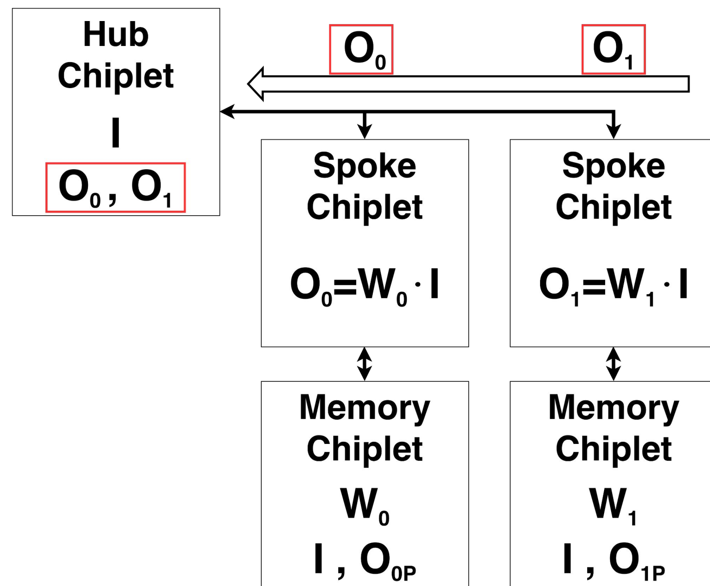
Creating the Memory Architecture

- When using this topology to run machine-learning-inference-type workloads, we can derive the following three classes of data.
  - Class 1 - Local Read Only **(NOR Flash)**
  - Class 2 - Local Scratchpad **(1T1C DRAM)**
  - Class 3 - Global Intermediates



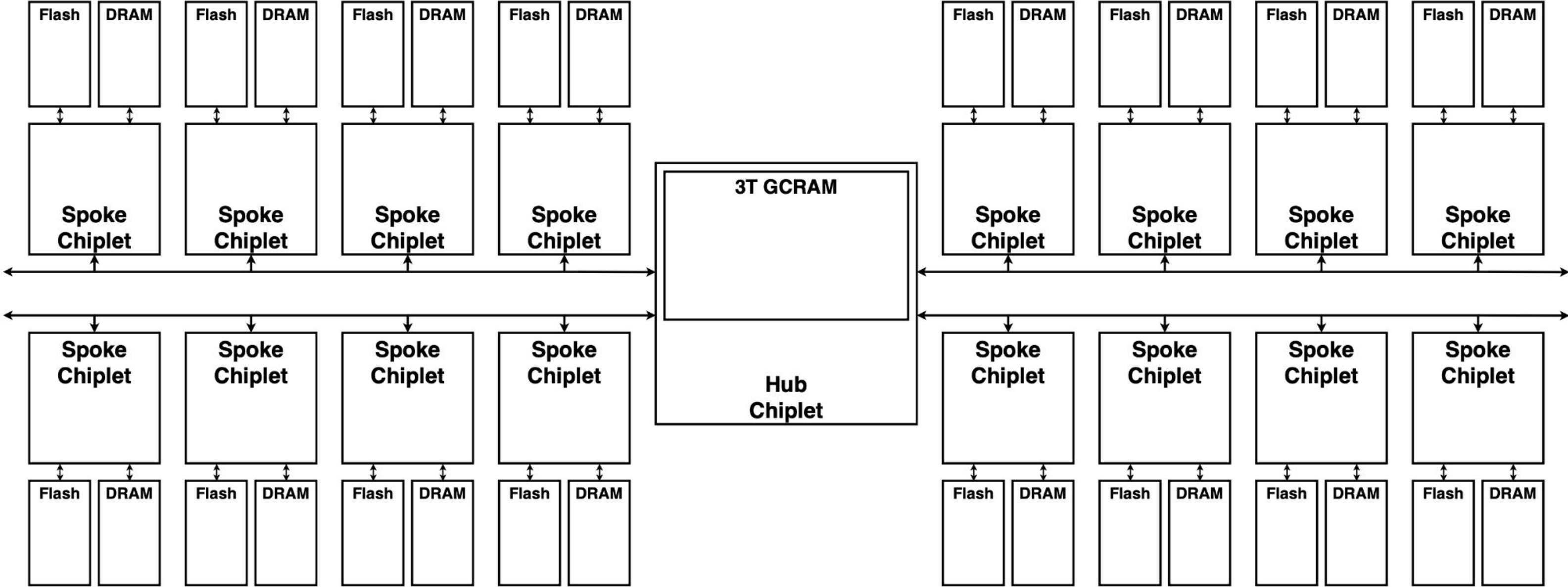
Creating the Memory Architecture

- When using this topology to run machine-learning-inference-type workloads, we can derive the following three classes of data.
  - Class 1 - Local Read Only **(NOR Flash)**
  - Class 2 - Local Scratchpad **(1T1C DRAM)**
  - Class 3 - Global Intermediates **(3T GCRAM)**



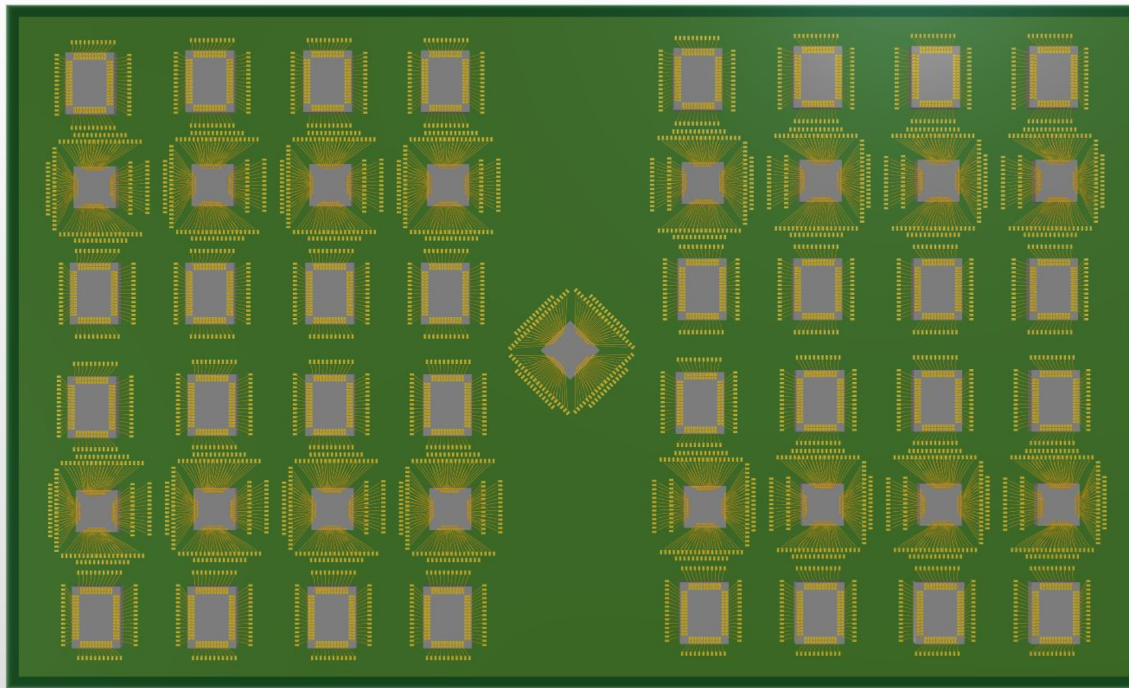
Creating the Memory Architecture

- When annotated over the original, we get the following architecture:



Packaging Considerations

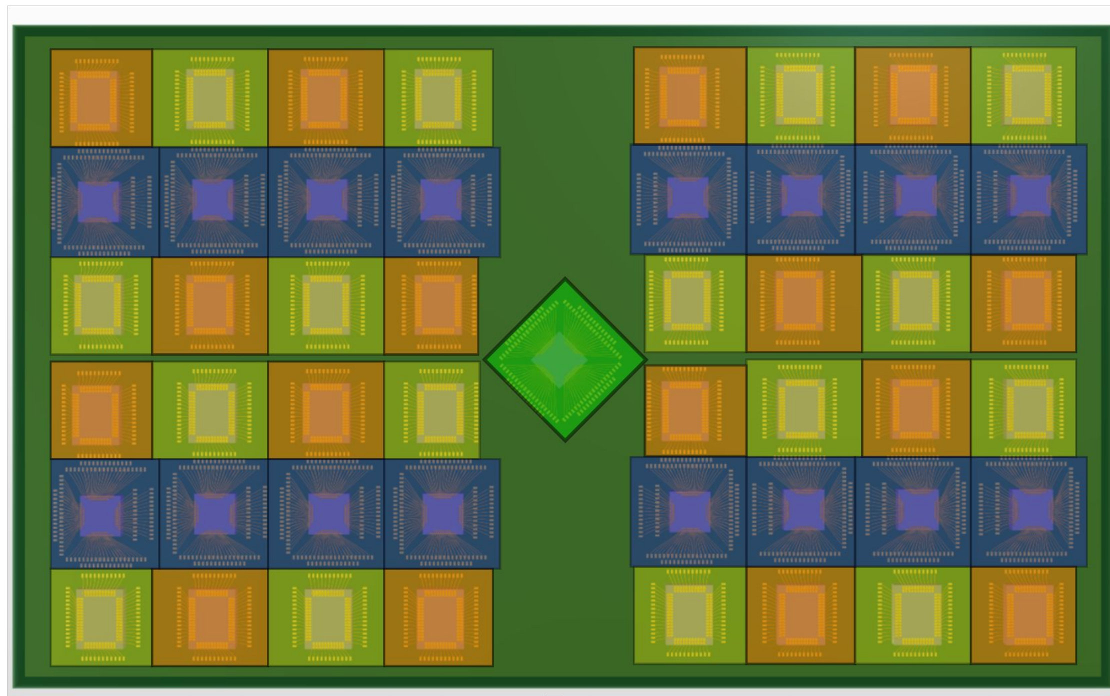
- Here is an **early** prototype of what this looks like at the scale of the entire package:



Packaging Considerations

- Here is an **early** prototype of what this looks like at the scale of the entire package:

NOR Flash
1T1C DRAM
Spoke
Hub



How can we map a transformer workload to this architecture?

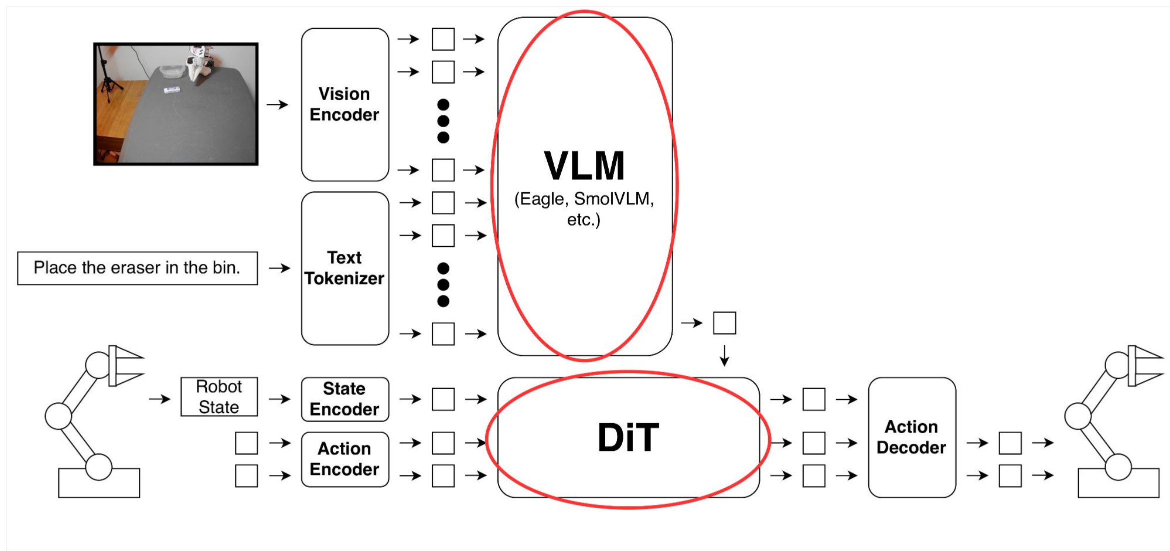
Why a transformer workload?

Why a transformer workload?

- Transformer layers are the backbone for VLA-style physical AI.

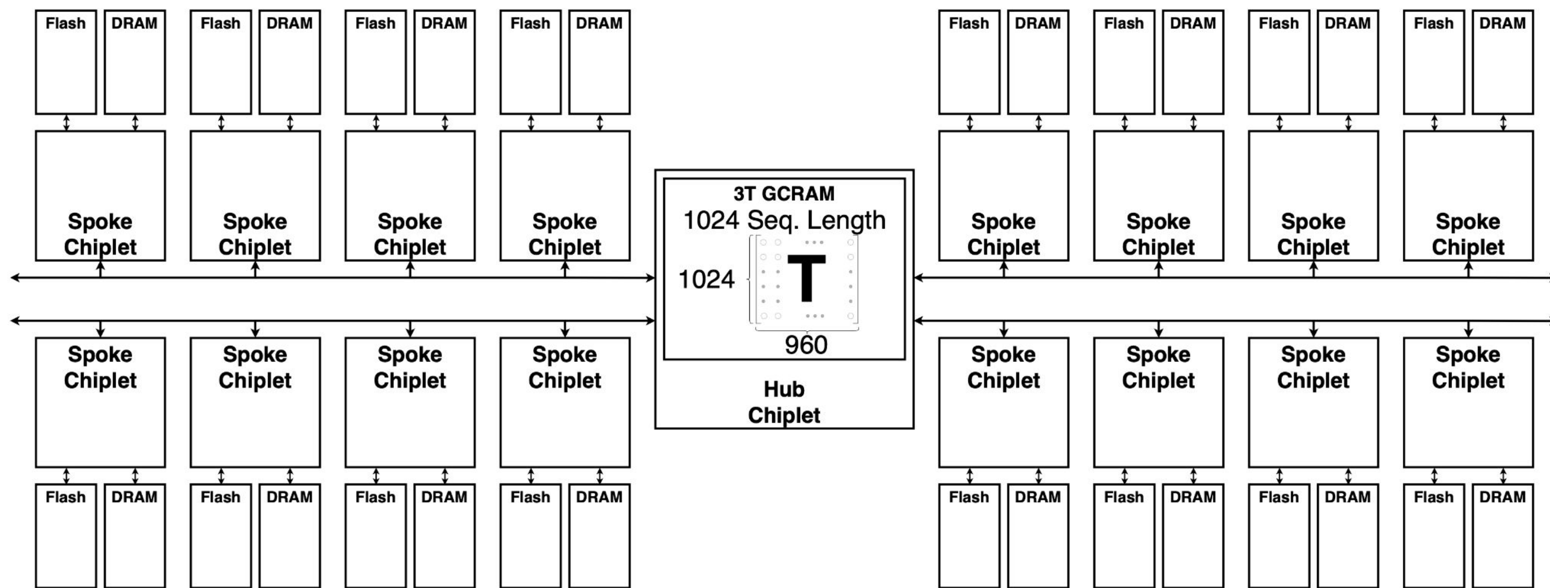
Why a transformer workload?

- Transformer layers are the backbone for VLA-style physical AI.
- This ML workload is used extensively in both the VLM and the DiT blocks.



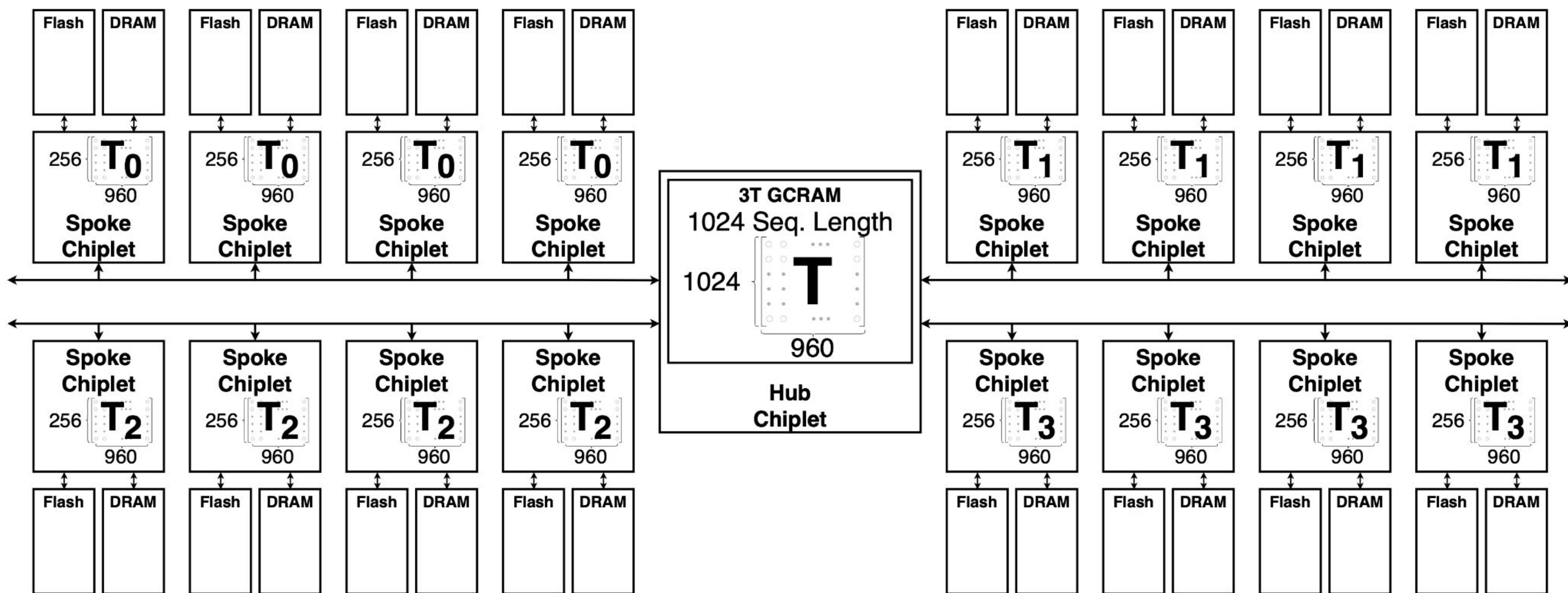
Dataflow for Group Query Attention

- The tokens to be processed will be deposited into the central chiplet through the off-package interface.
- In this example there are 1024 tokens with a hidden dimension size of 960.



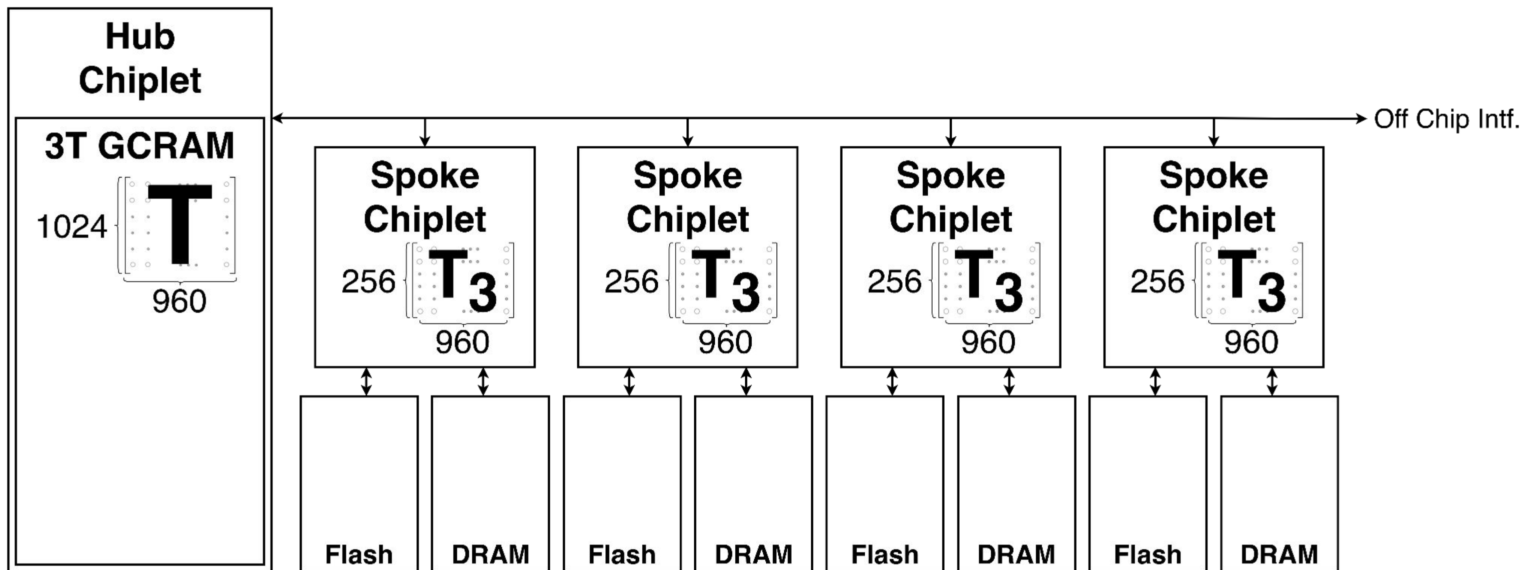
Dataflow for Group Query Attention Cont.

- The tokens are distributed evenly branch-wise across the network. In this example, this means that 256 tokens are sent to each branch.



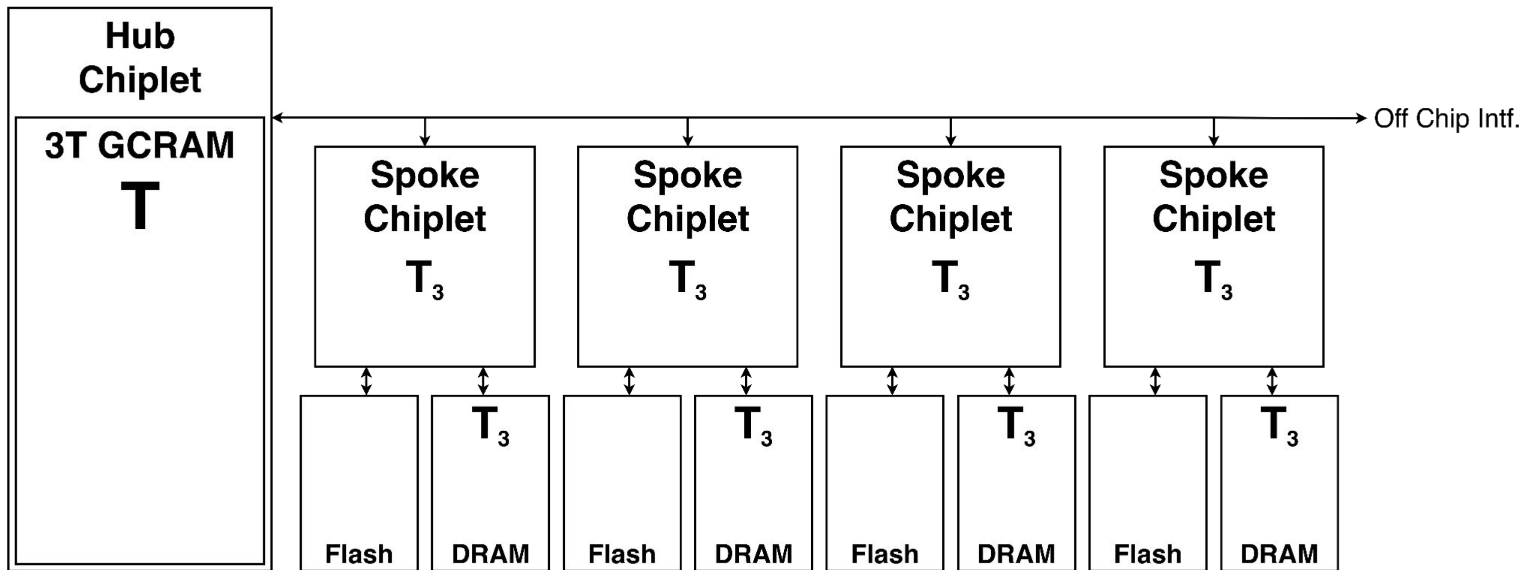
Dataflow for Group Query Attention Cont.

- For the sake of simplicity, we will focus on only a single branch for this walkthrough as, in the mapping we will be describing, the operations performed on each branch are exactly the same.



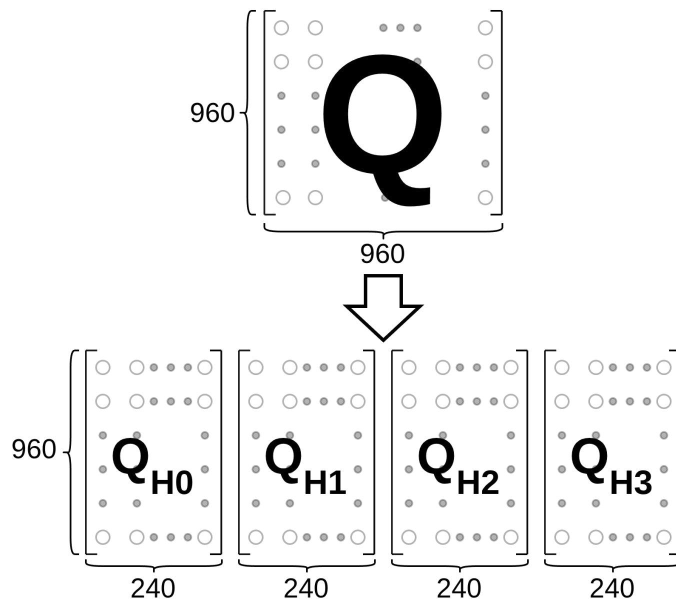
Dataflow for Group Query Attention Cont.

- To make the notation simpler, the term *T* denotes the whole token sequence while the term T_3 denotes the partial token sequence dispatched to branch 3.



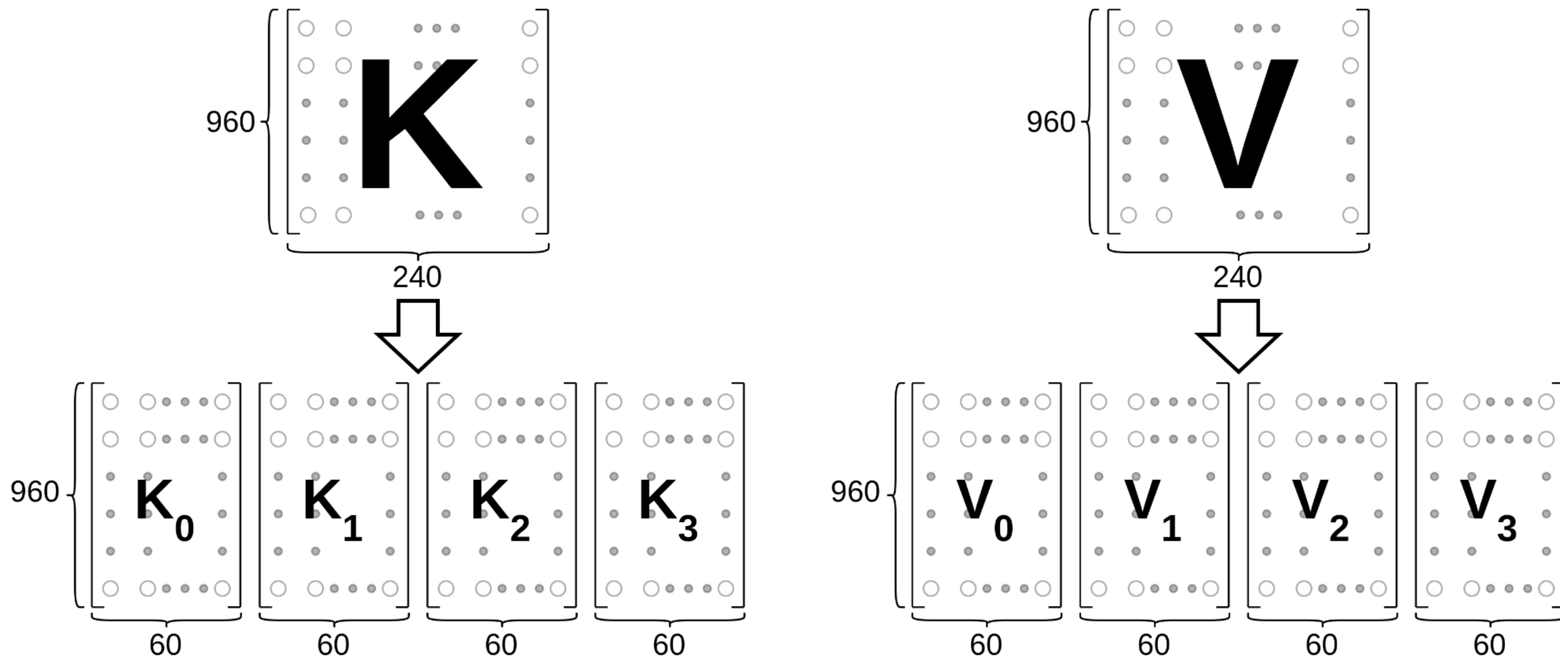
Dataflow for Group Query Attention Cont.

- Next, consider how should the weights be distributed in memory
- Given that we are using GQA, we will have multiple Q heads per KV head. In this example we will be assuming 4 Q heads and 1 KV head.



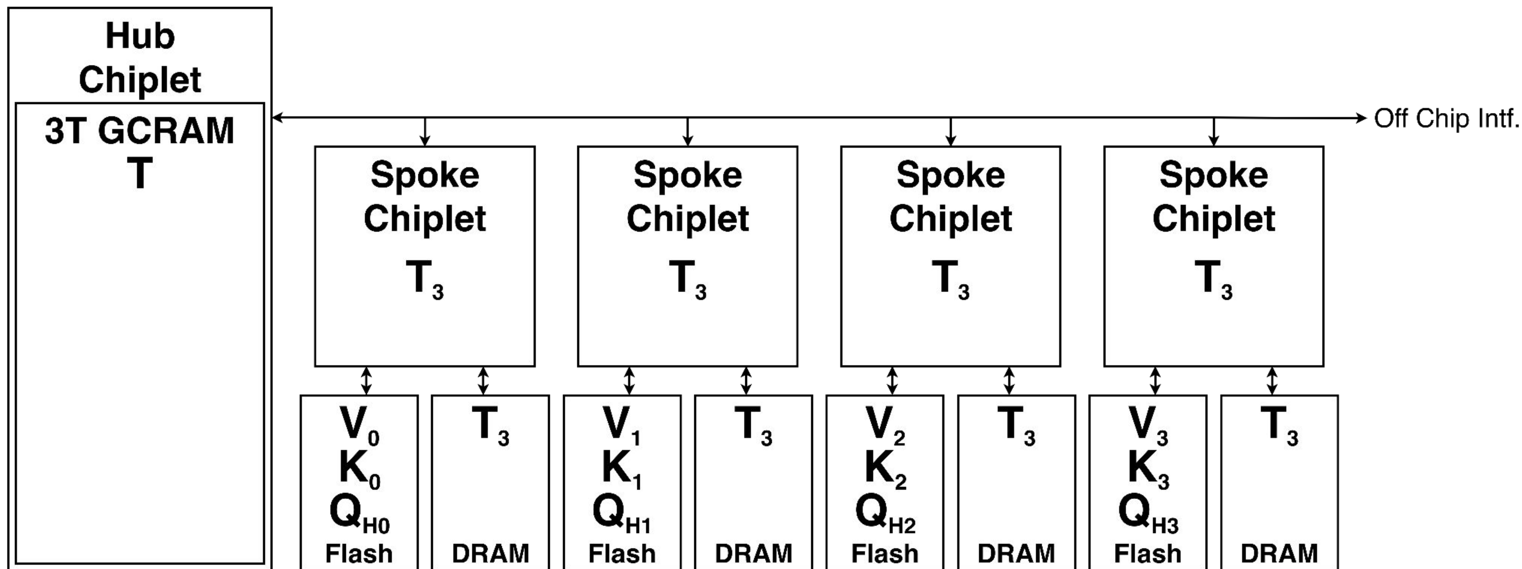
Dataflow for Group Query Attention Cont.

- For the K and V heads, we will simply divide up each head equally among the four chiplets.



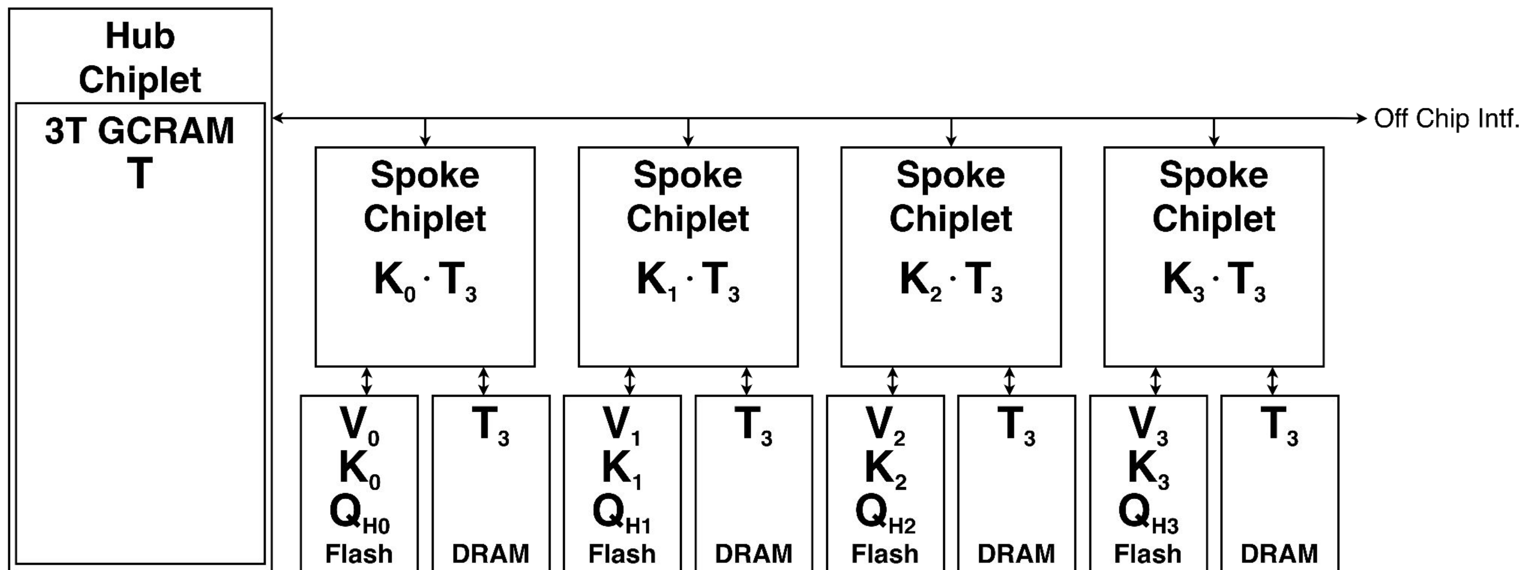
Dataflow for Group Query Attention Cont.

- The weight matrices are mapped to NOR flash.



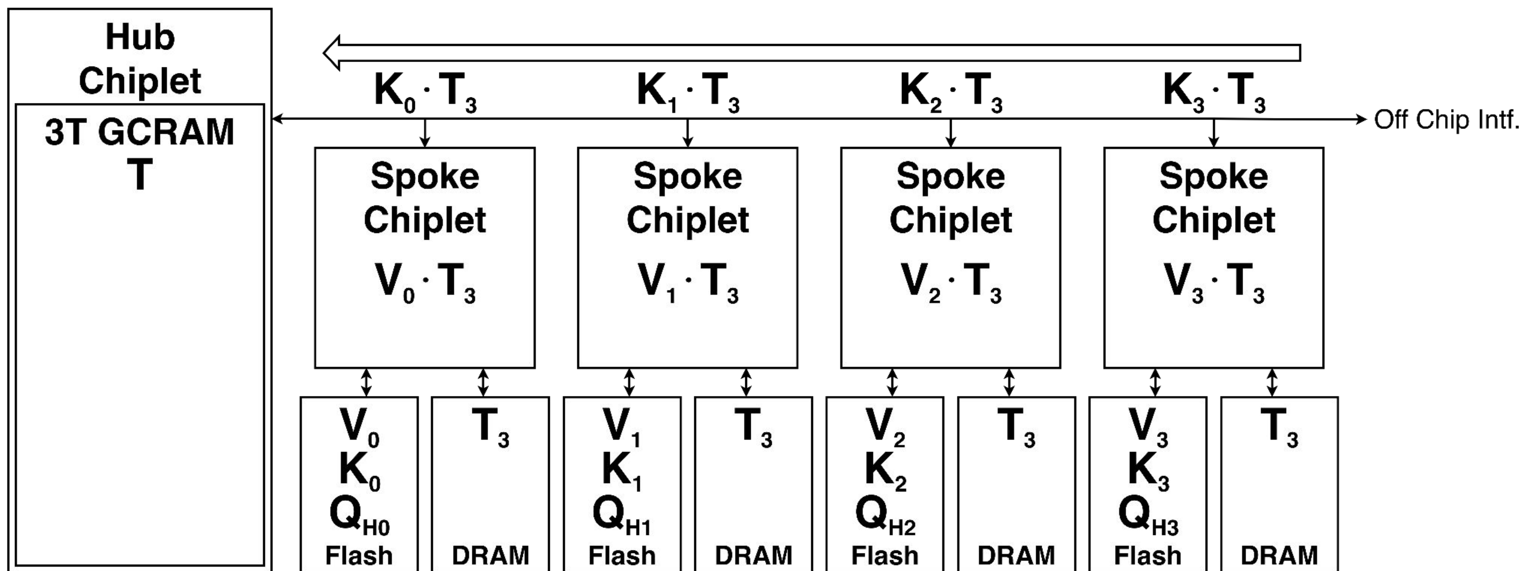
Dataflow for Group Query Attention Cont.

- The first computation we perform is the K matrix multiplication.



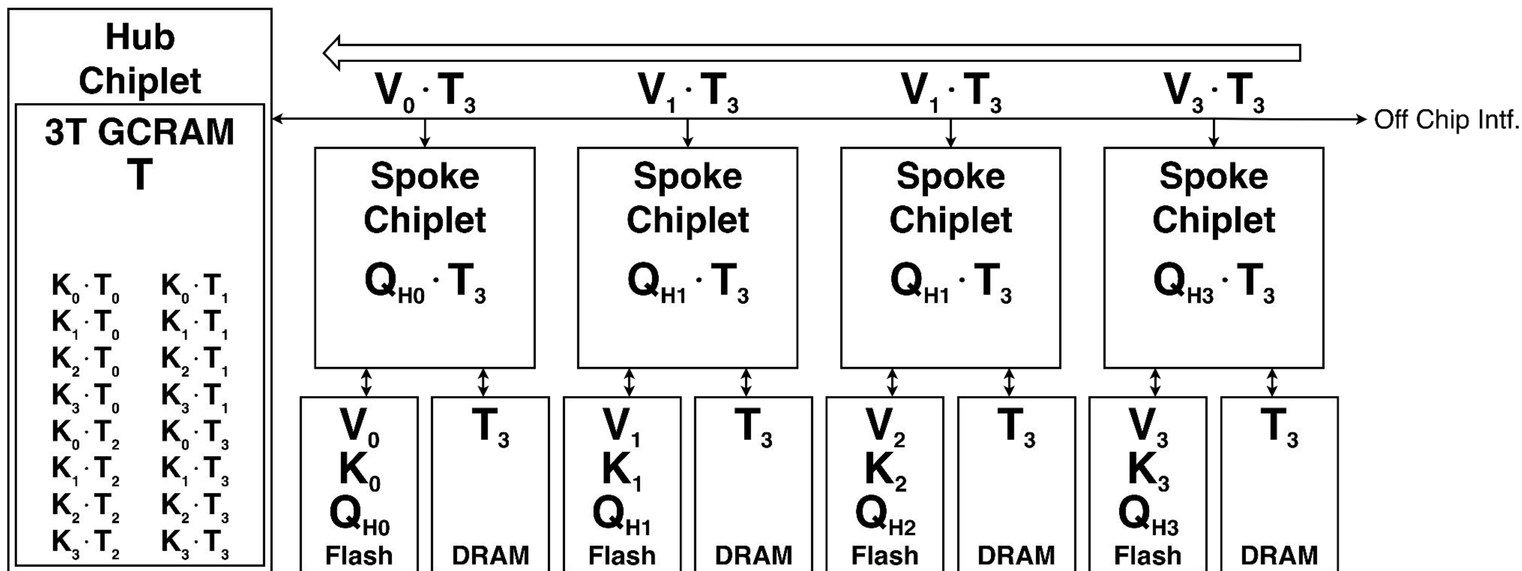
Dataflow for Group Query Attention Cont.

- We then overlap the following:
  - Data-transfer of the K matrix multiplication.
  - Computation of the V matrix multiplication.



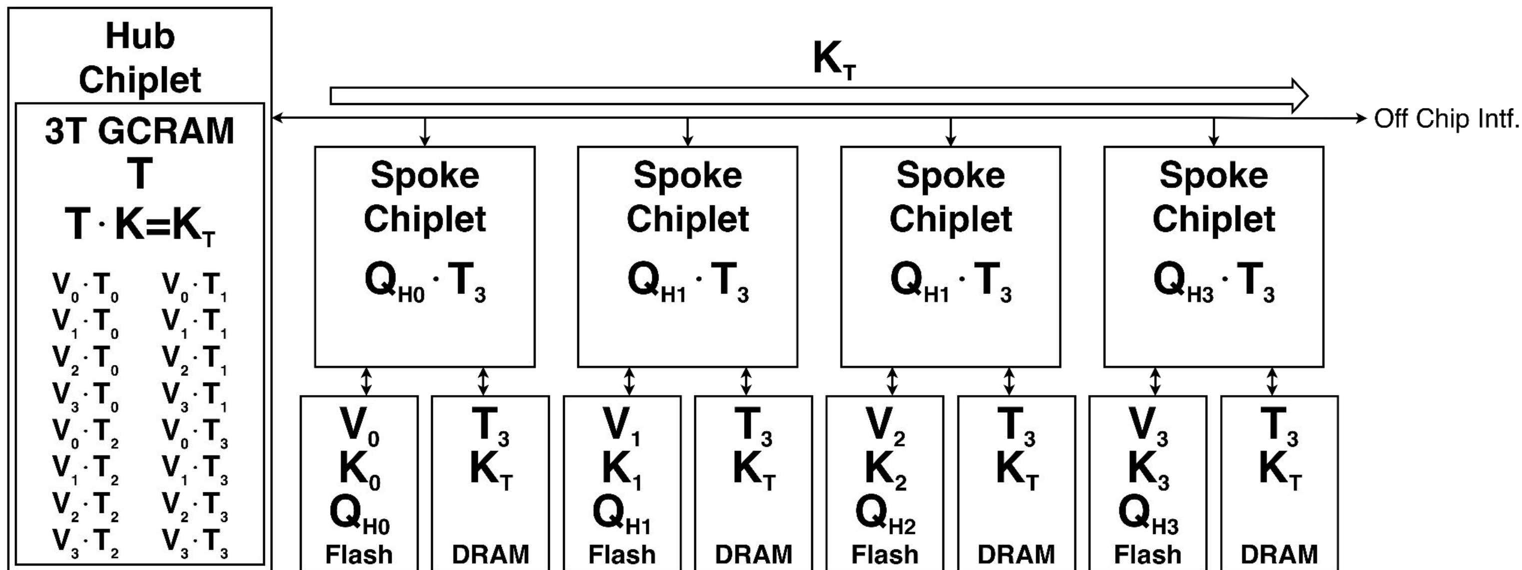
Dataflow for Group Query Attention Cont.

- We then overlap the following:
  - Data-transfer of the V matrix multiplication.
  - Computation of the Q matrix multiplication.



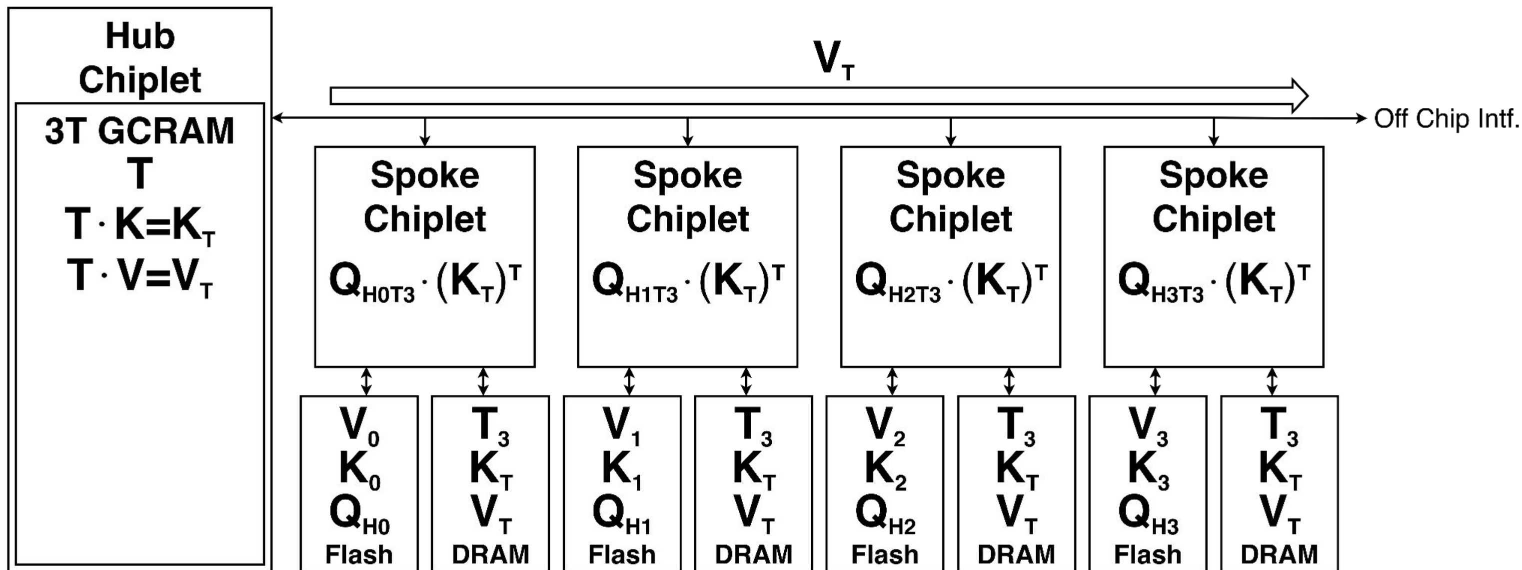
Dataflow for Group Query Attention Cont.

- Now, the partial K matrix results are reorganized within the hub chiplet. (This process is also overlapped with the V matrix output data-transfer and Q matrix computation.)
- Once this is complete this is broadcast back out to all of the connected chiplets.



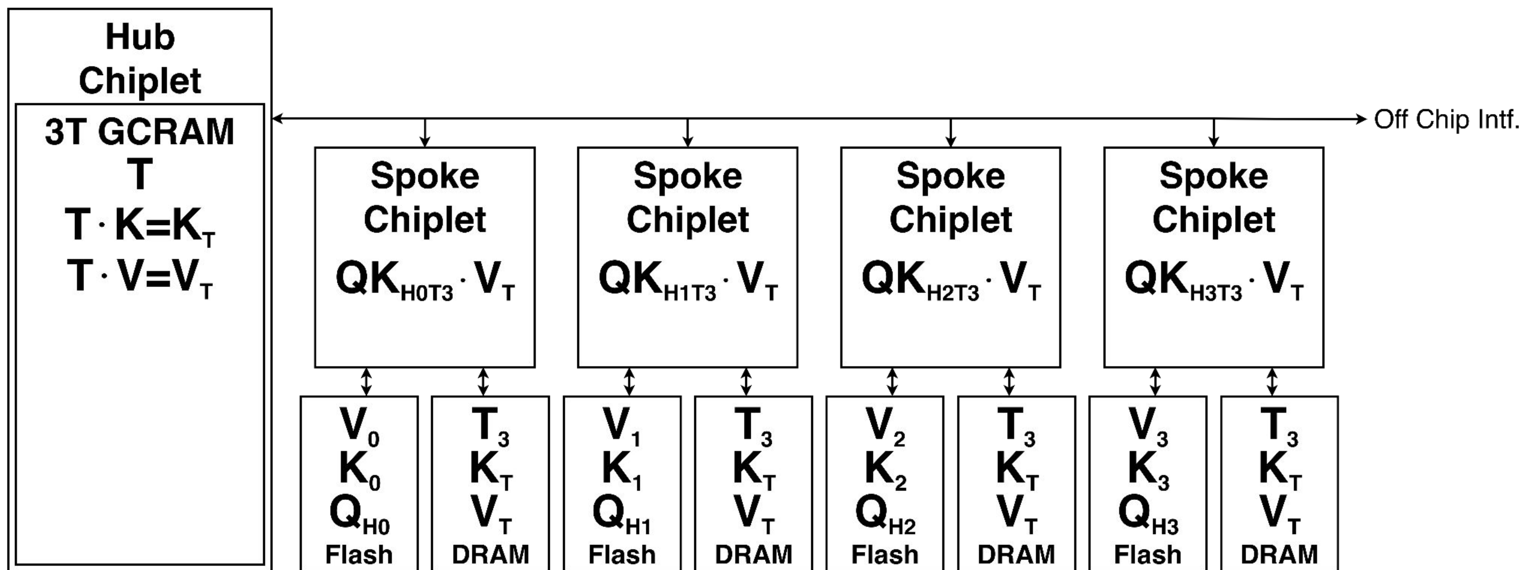
Dataflow for Group Query Attention Cont.

- The V matrix outputs are then reorganized, transposed and broadcast out to the target chiplets. This process is overlapped with the QK computation.



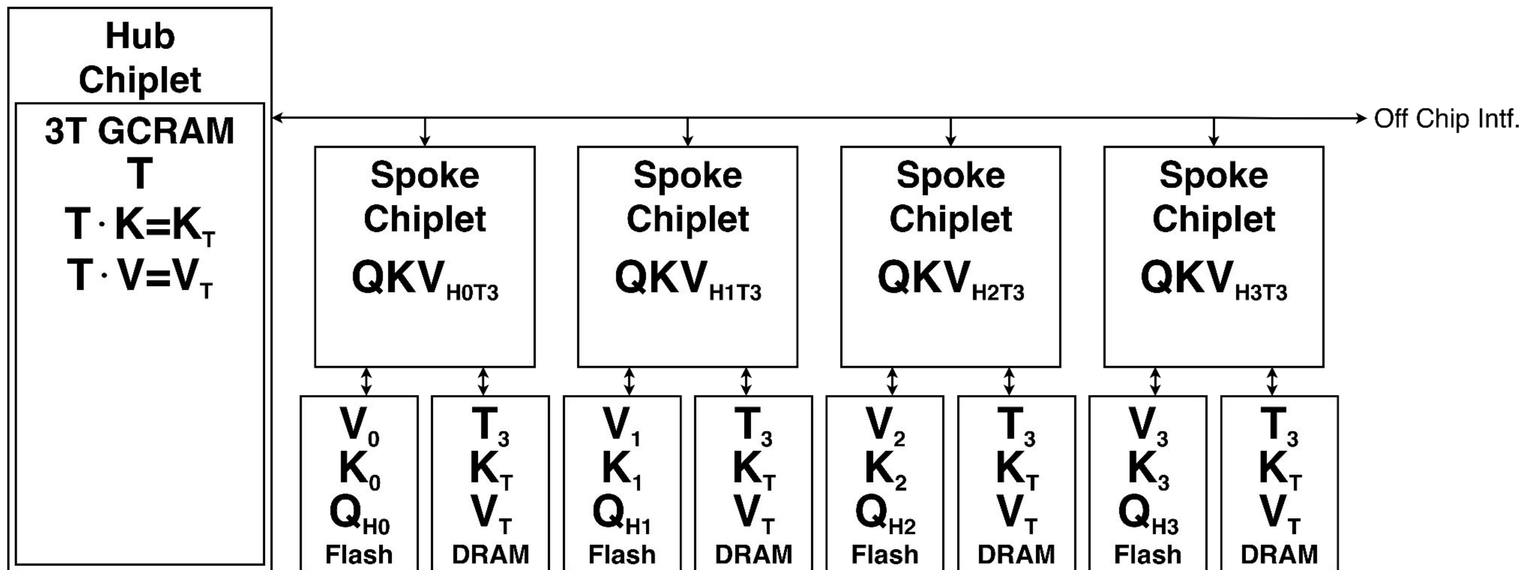
Dataflow for Group Query Attention Cont.

- The per-head QKV result is computed.



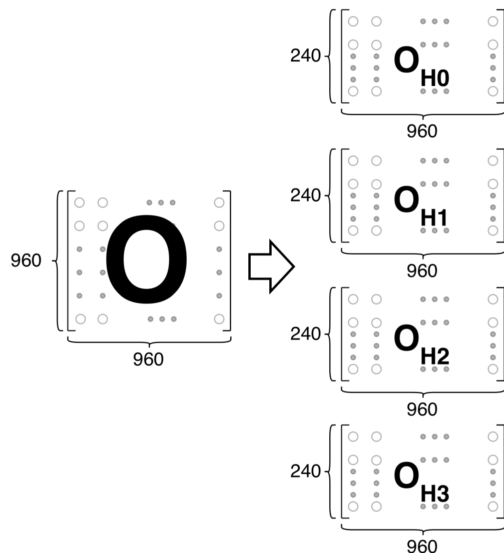
Dataflow for Group Query Attention Cont.

- We then arrive in this state, where the QKV product is complete and stored locally in each chip.



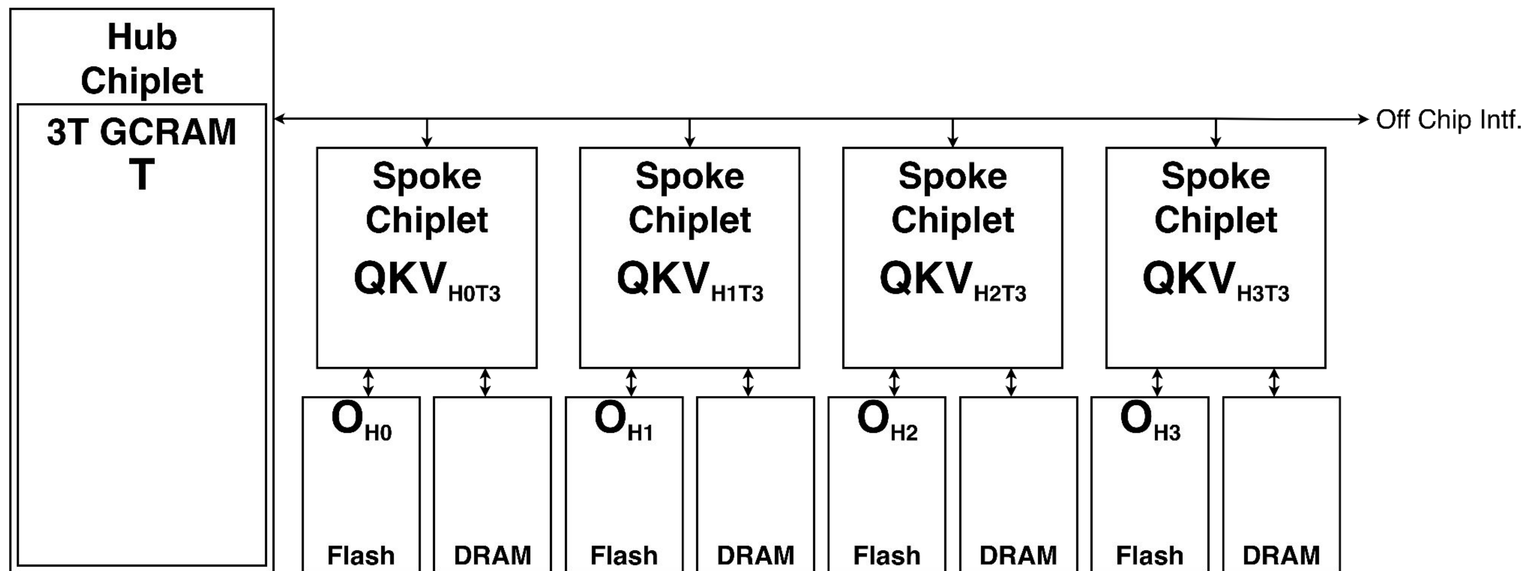
Dataflow for Group Query Attention Cont.

- Next the output matrix must be applied to these QKV partial results. There are two ways we can map this operation. (We will only be discussing a single option)
- The following notation is useful in discussing these operations.



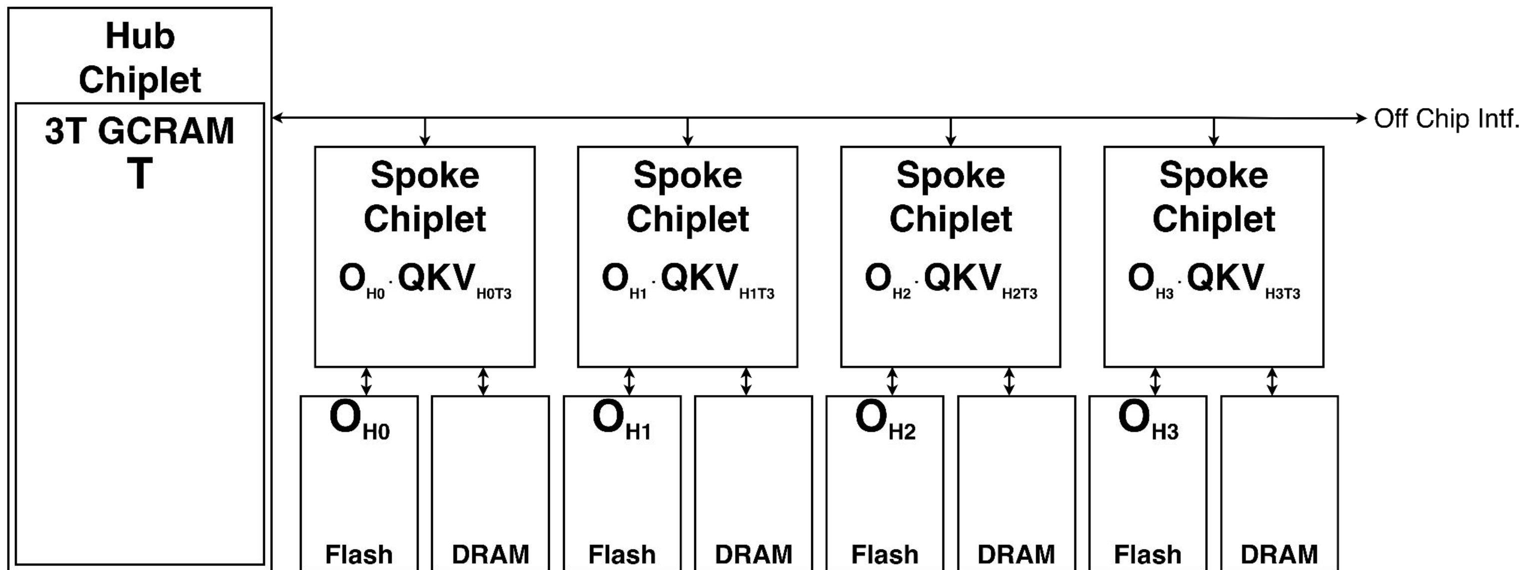
Attention Dataflow for GQA Cont.

- Each portion of the output matrix is mapped to the flash of the associated chiplet.



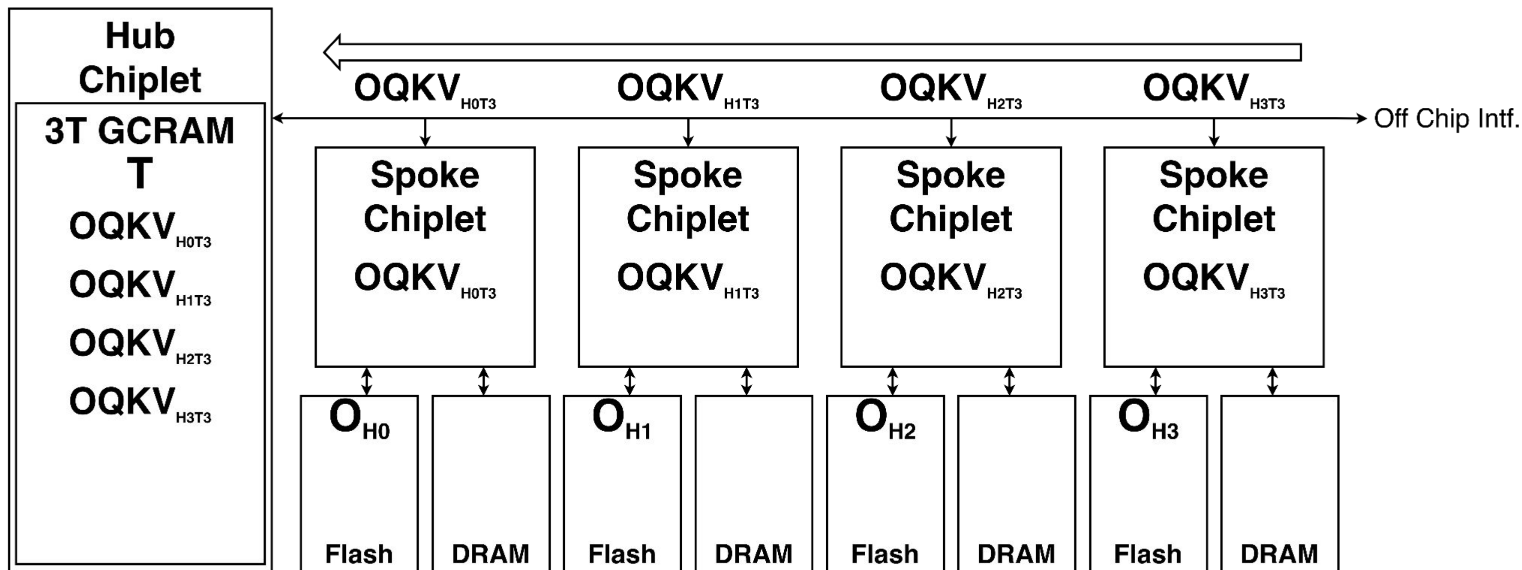
Dataflow for Group Query Attention Cont.

- The associated QKV partial result is then multiplied with the partial O matrix to create a partial OQKV result.



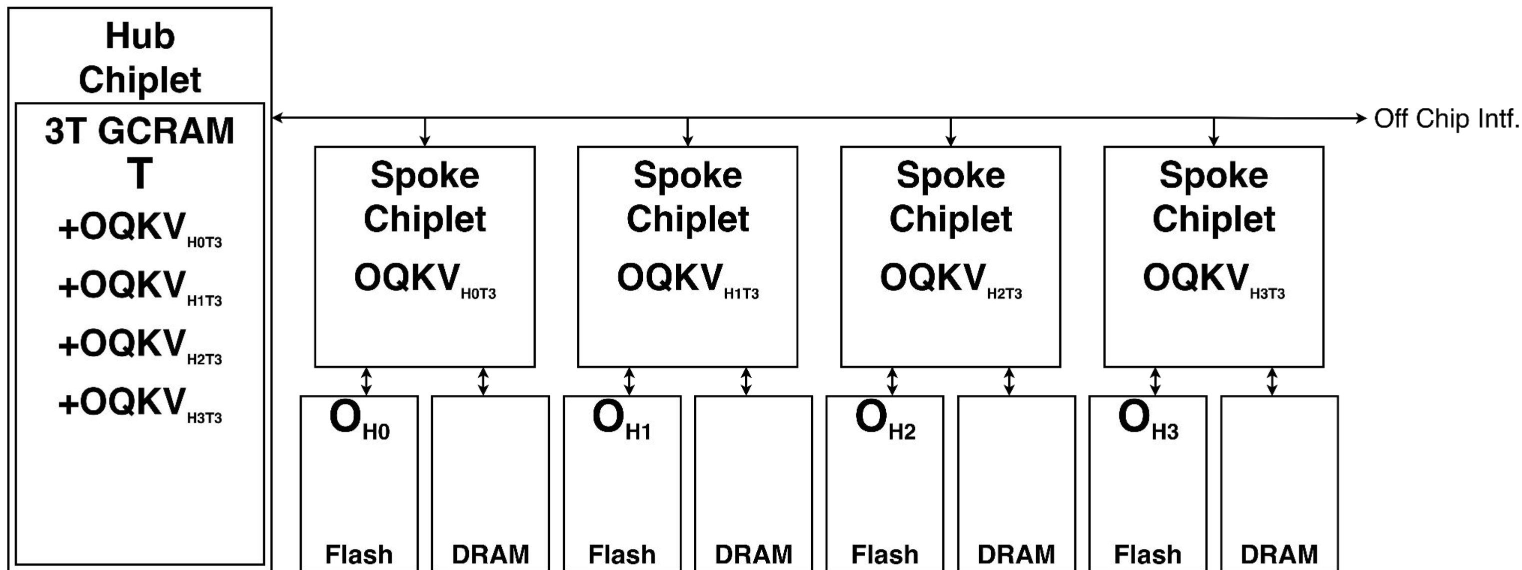
Dataflow for Group Query Attention Cont.

- The partial OQKV result is then gathered by the hub-chiplet.



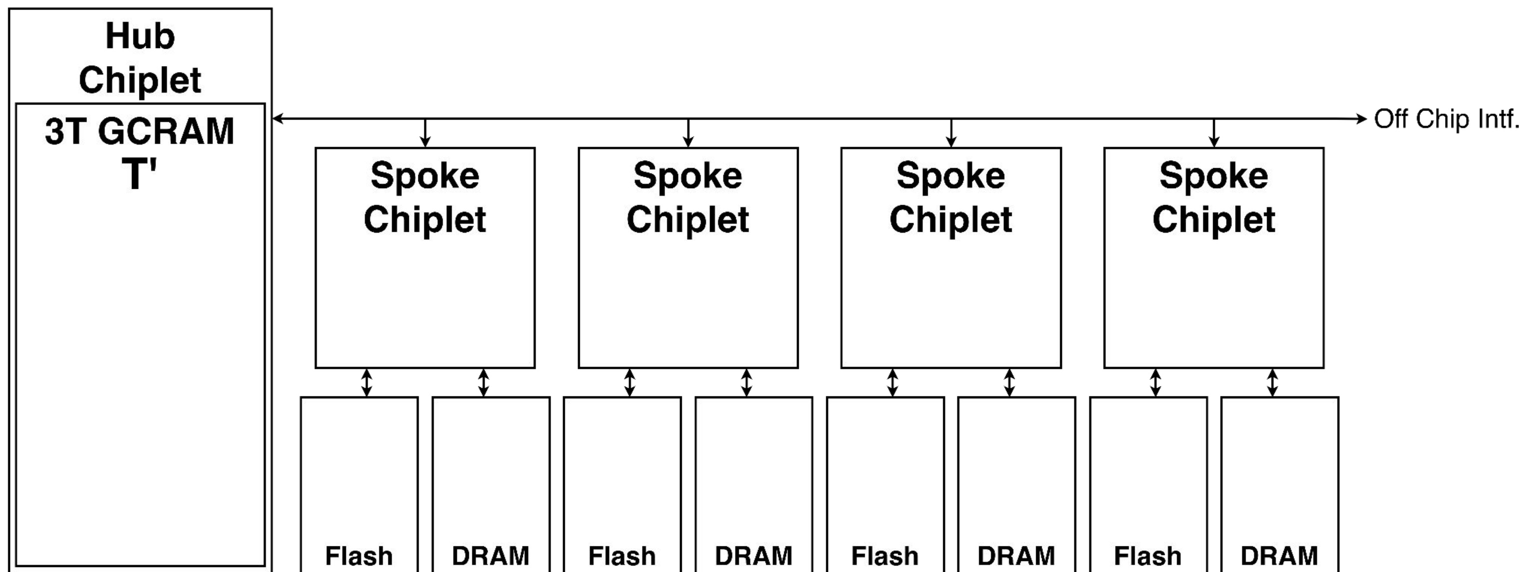
Dataflow for Group Query Attention Cont.

- These partial results are then summed in the Hub chiplet.



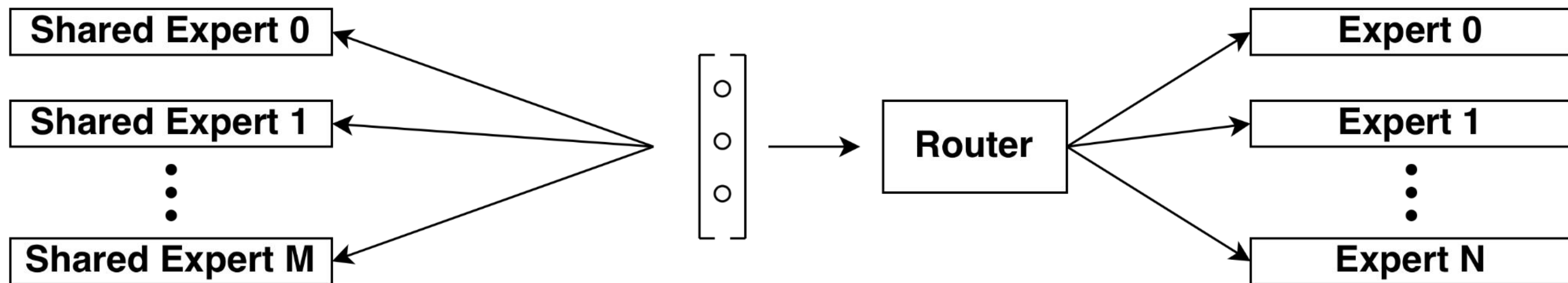
Dataflow for Group Query Attention Cont.

- There is now a post-attention set of tokens located in the 3T GCRAM of the Hub Chiplet.



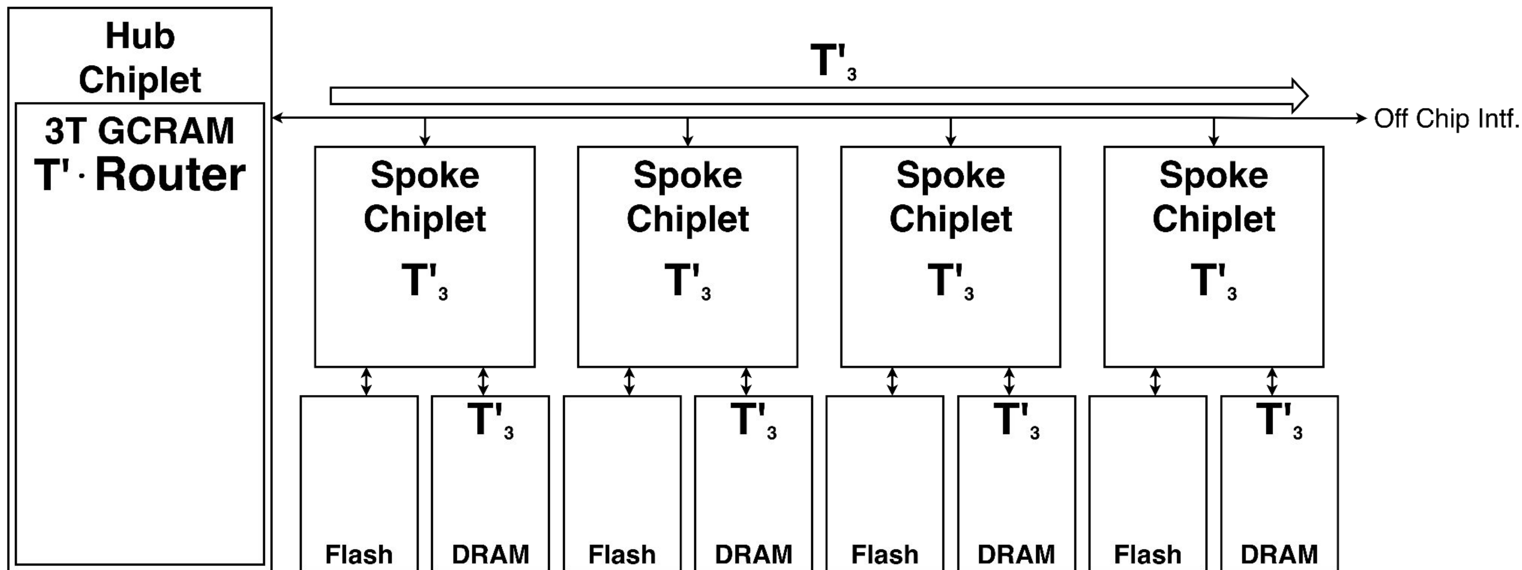
Mixture of Experts (MoE) FFN Dataflow

- MoE-style transformer layers route a given token through a series of experts.
- While there are very few existing MoE models at our target parameter count, we have developed techniques that allow us to generate an MoE FFN from an existing dense FFN.



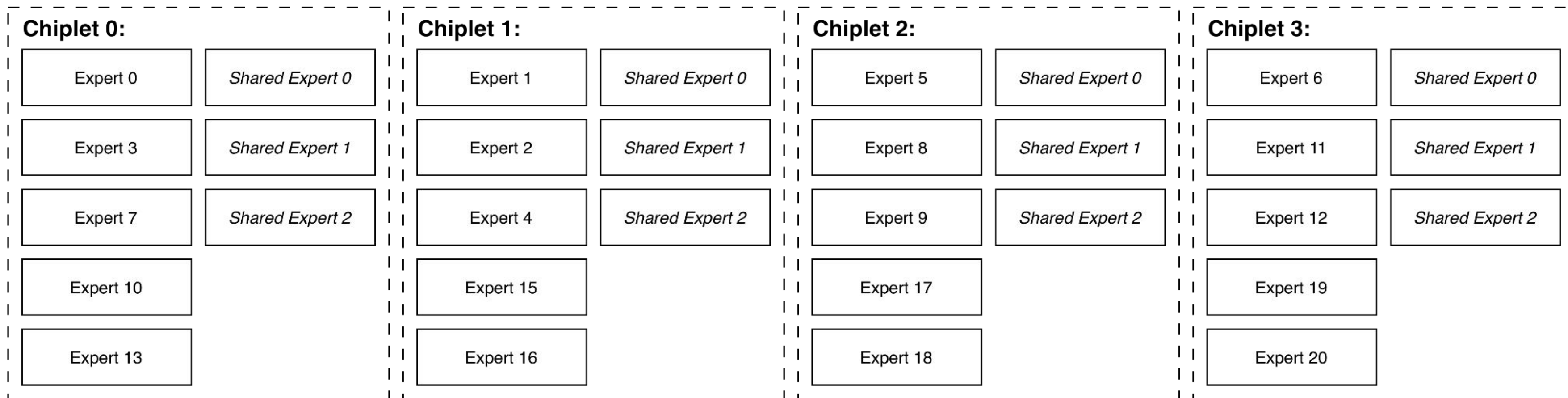
Mixture of Experts (MoE) FFN Dataflow

- The hub-chiplet overlaps the router computation with the token broadcast.



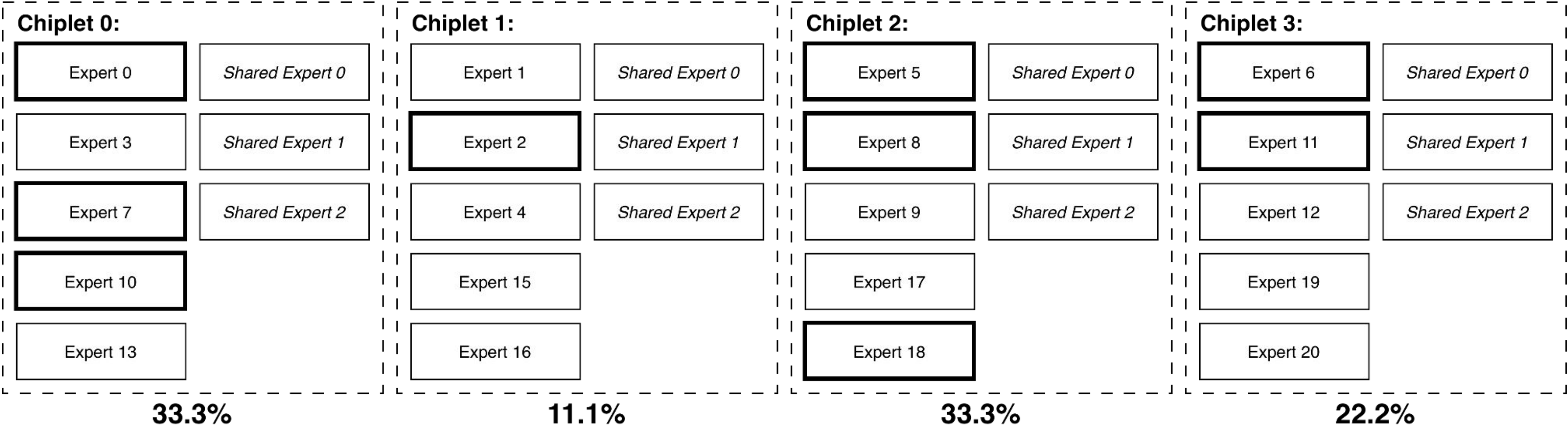
Mixture of Experts (MoE) FFN Dataflow

- For our MoE dataflow, we first profile our workload and determine which experts are commonly activated together, then place those experts in the local memory of **different** chiplets.



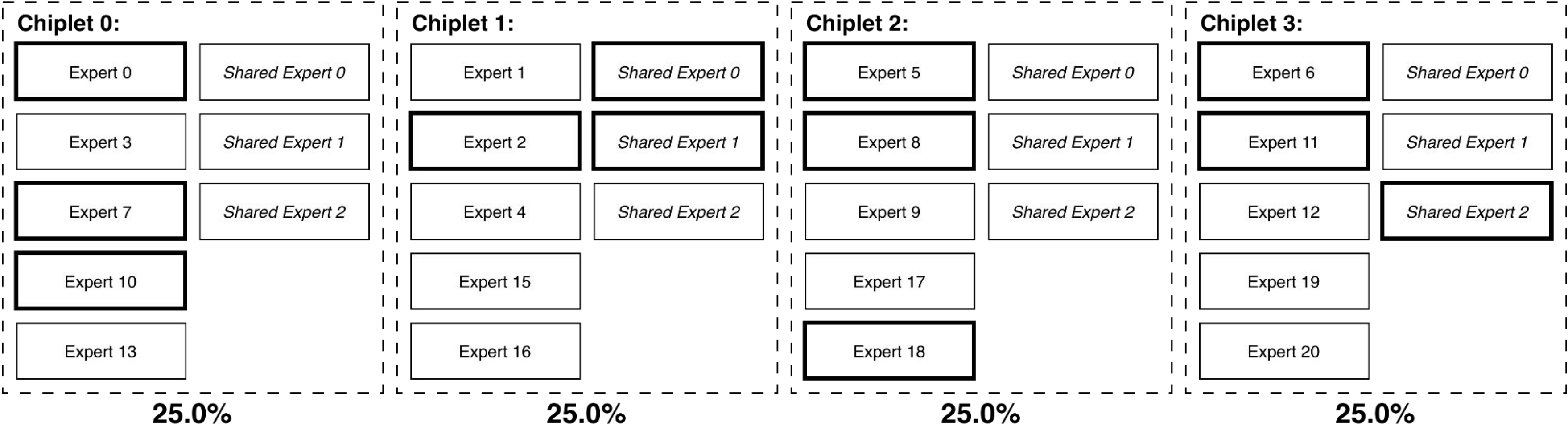
Mixture of Experts (MoE) FFN Dataflow

- After the routing computation, the hub-chiplet knows which tokens require which unique experts. Due to random chance, this usage may be uneven.



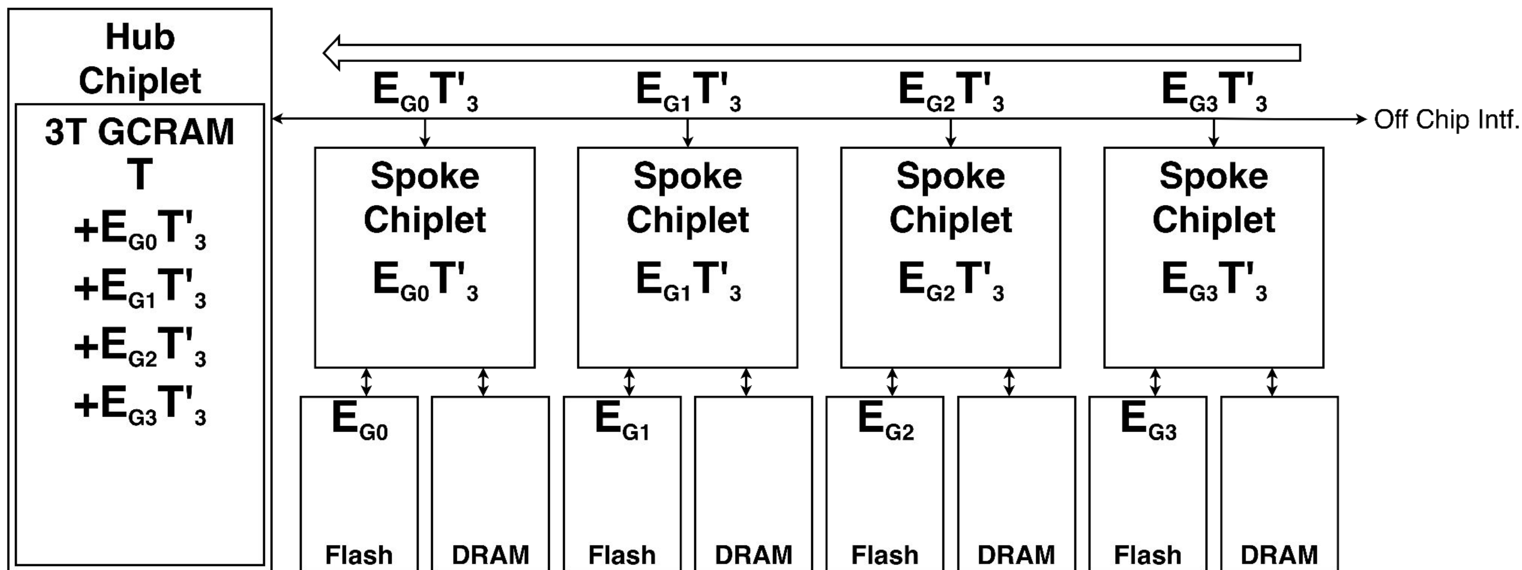
Mixture of Experts (MoE) FFN Dataflow

- In order to balance the compute load on each chiplet, the hub will assign the under-utilized spoke chiplets to perform the shared expert computations.



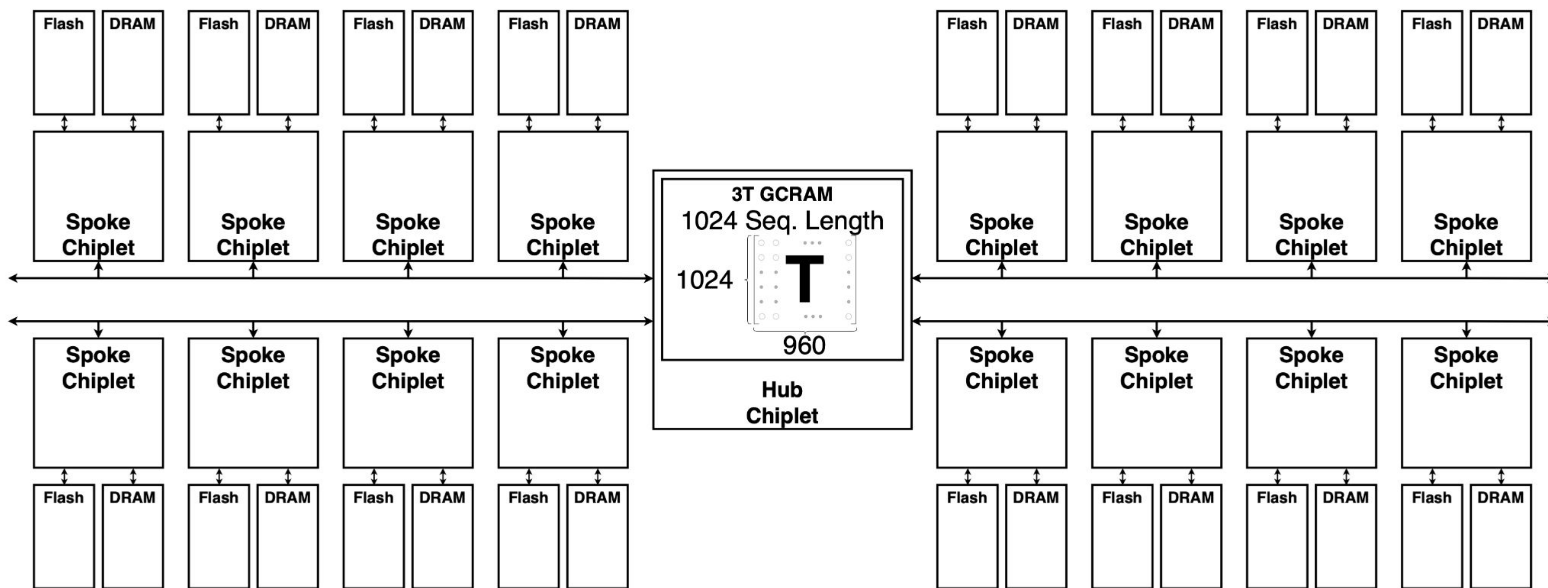
Mixture of Experts (MoE) FFN Dataflow

- Once the expert computations are complete, the results are gathered back into the hub and added to the token residual.



Mixture of Experts (MoE) FFN Dataflow

- The transformer layer is now complete and the process is ready to be repeated for any subsequent transformer layers.



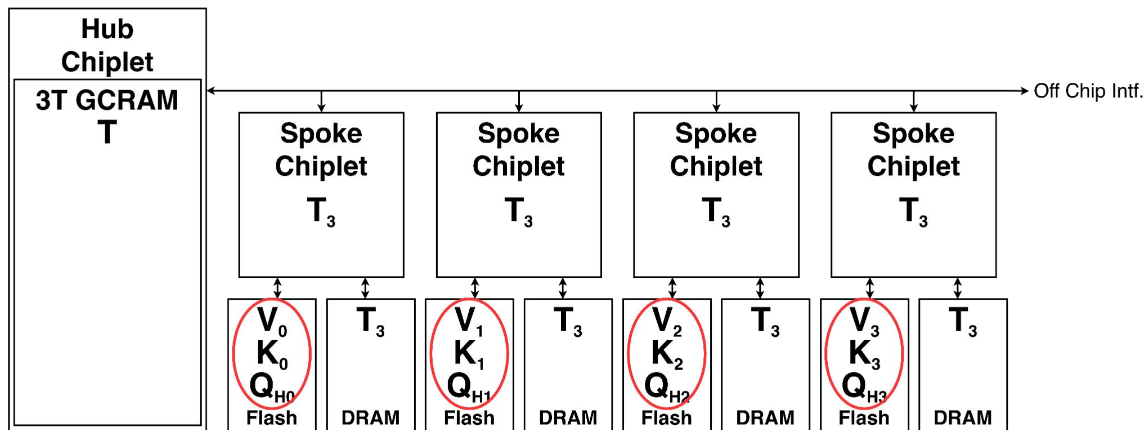
How does our transformer workload mapping fit into our classes of data?

Creating the Memory Architecture

- If you recall, we had the following classes of data:
  - Class 1 - Local Read Only (NOR Flash)
  - Class 2 - Local Scratchpad (Read/Write) (1T1C DRAM)
  - Class 3 - Global Intermediates (3T GCRAM)

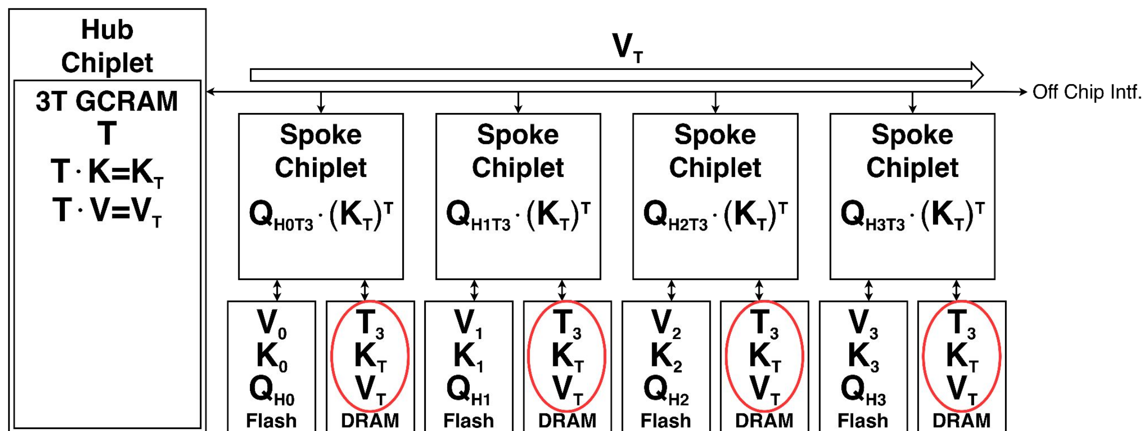
Creating the Memory Architecture

- If you recall, we had the following classes of data:
  - **Class 1 - Local Read Only (NOR Flash)**
  - Class 2 - Local Scratchpad (Read/Write) (1T1C DRAM)
  - Class 3 - Global Intermediates (3T GCRAM)



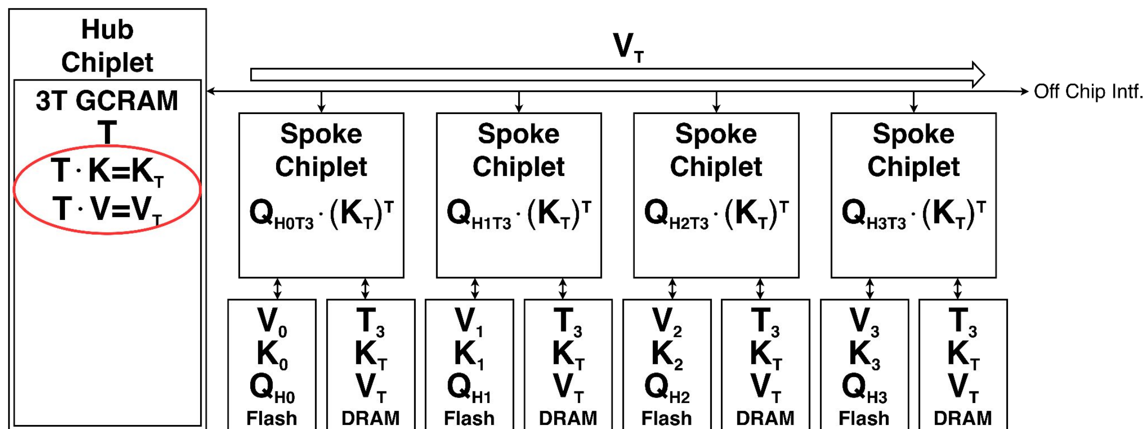
Creating the Memory Architecture

- If you recall, we had the following classes of data:
  - Class 1 - Local Read Only (NOR Flash)
  - **Class 2 - Local Scratchpad (Read/Write) (1T1C DRAM)**
  - Class 3 - Global Intermediates (3T GCRAM)



Creating the Memory Architecture

- If you recall, we had the following classes of data:
  - Class 1 - Local Read Only (NOR Flash)
  - Class 2 - Local Scratchpad (Read/Write) (1T1C DRAM)
  - **Class 3 - Global Intermediates (3T GCRAM)**



Recap

Summary

- The uniqueness of Physical AI as a workload
- What makes a desirable chiplet structure for Physical AI
- Our proposed chiplet architecture
  - Packaging-level considerations
  - Workload Mapping
  - Memory Architecture

Thank You