

## 1 Problem & motivation

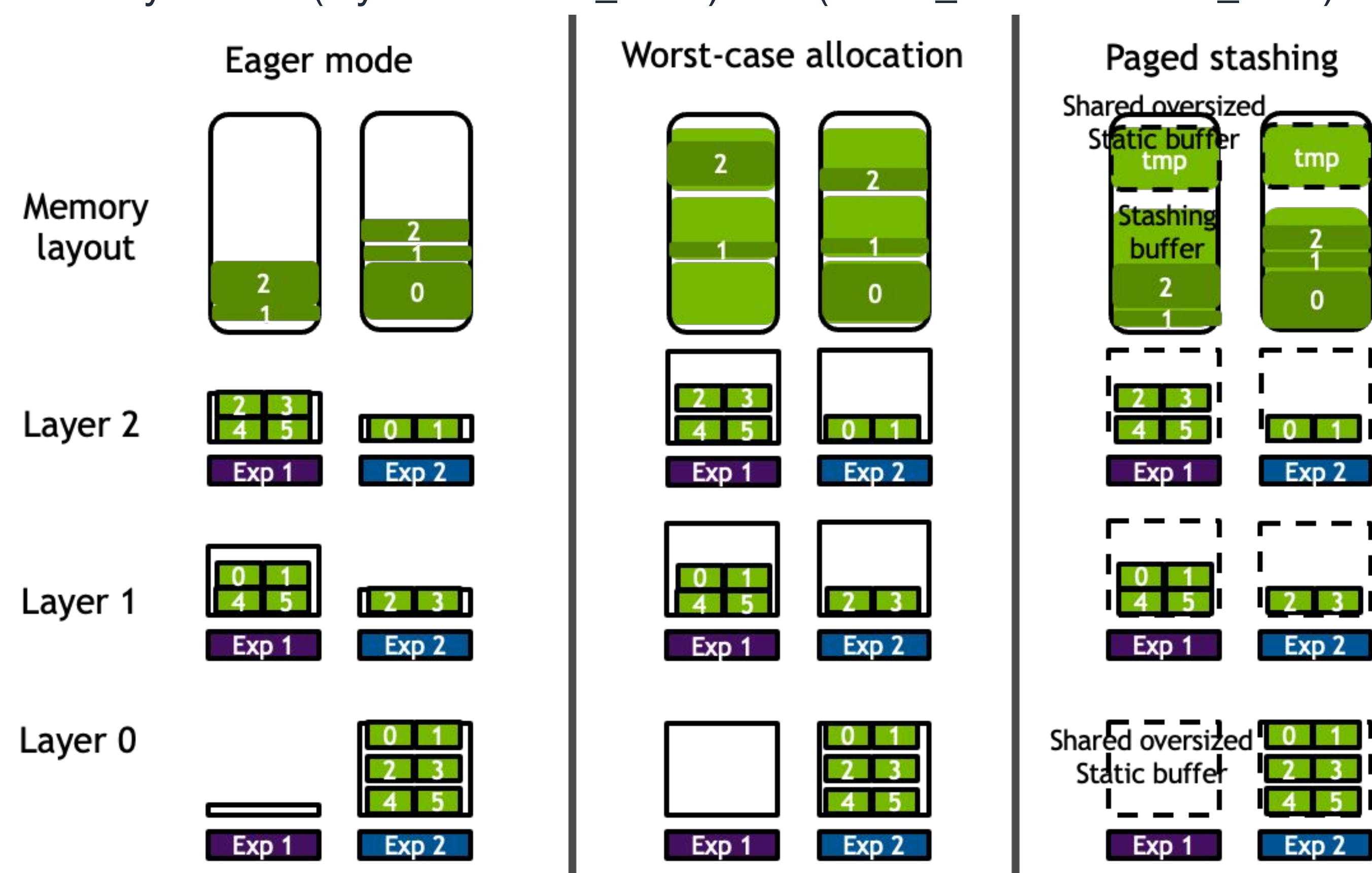
CUDA graph replay requires fixed buffer addresses and shapes, but MoE routing produces variable token counts. Conventional communication and grouped GEMM kernels require the host to retrieve these counts from the GPU, introducing synchronization that prevents graph capture.

Device-initiated grouped GEMM reads token counts directly from GPU memory, while HybridEP uses preallocated buffers for sync-free communication, removing these synchronization barriers.

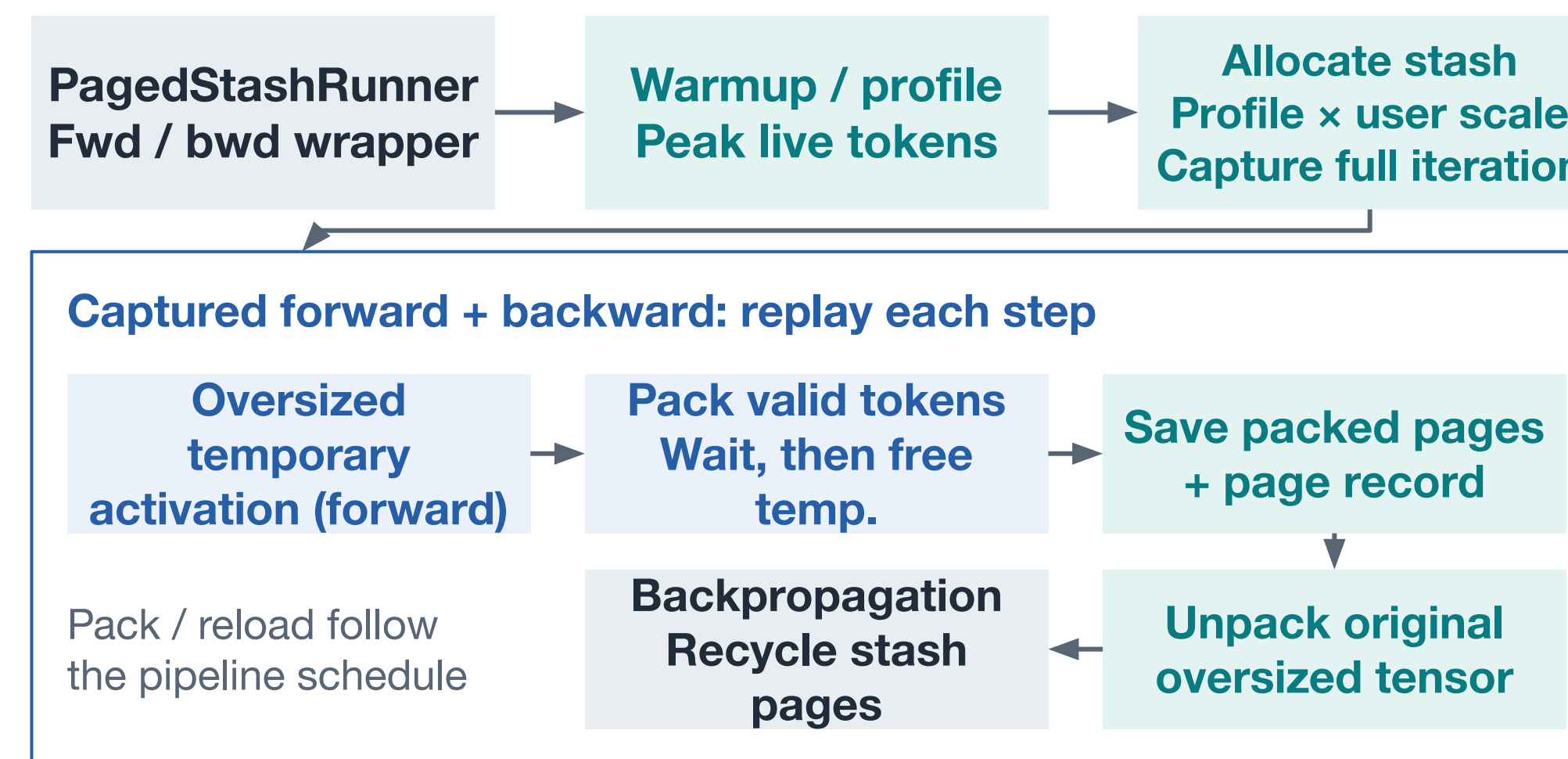
Sync-free execution still requires preallocated buffers. Sizing each layer's activation buffer for the worst case and retaining it for backward leads to  $O(\text{layers} \times \text{worst\_case})$  activation memory usage, even when actual token counts are much lower. Making MoE training CUDA-graphable therefore requires both sync-free kernels and memory-efficient activation storage.

## 2 Paged stashing: bounded temporary memory

Keep 1 oversized temporary activation buffer. Pack routed data into a shared stash pool and release the temporary for reuse, reducing activation memory from  $O(\text{layers} \times \text{worst\_case})$  to  $O(\text{worst\_case} + \text{actual\_total})$ .

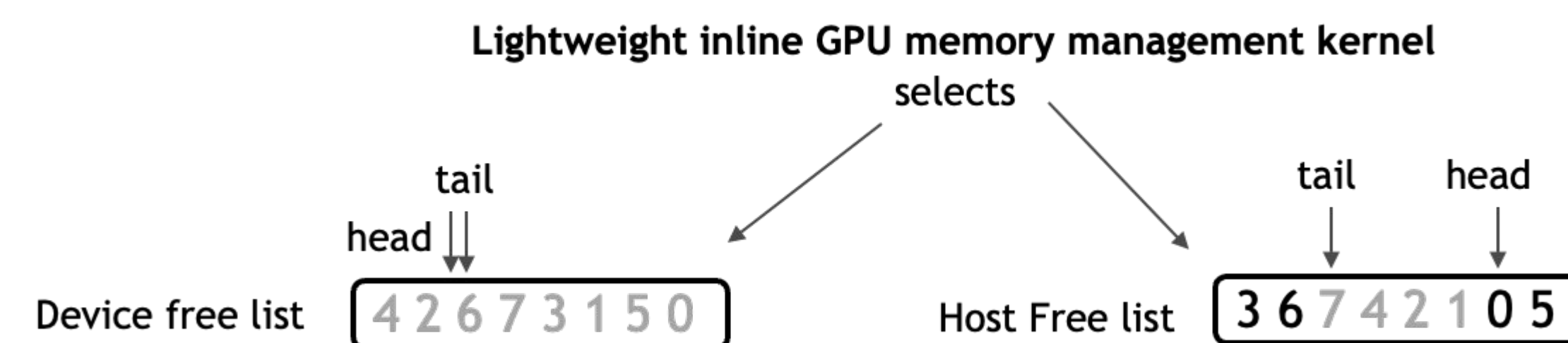


## 3 Megatron paged stashing flow



## 5 On-device light-weight memory management

Lightweight CUDA kernels manage memory pages using two circular buffers that track free pages in device and host memory. Activations are stashed in either memory pool according to the selected policy.



## 6 Megatron optimizations shipped to TorchTitan

We have integrated these Megatron optimizations into TorchTitan's GraphTrainer, enabling CUDA Graph capture of workloads using HybridEP and Paged Stashing through an FX graph pass that applies the paged stashing memory policy. The full integration composes with GraphTrainer's new GraphPP pipeline parallel mode.

