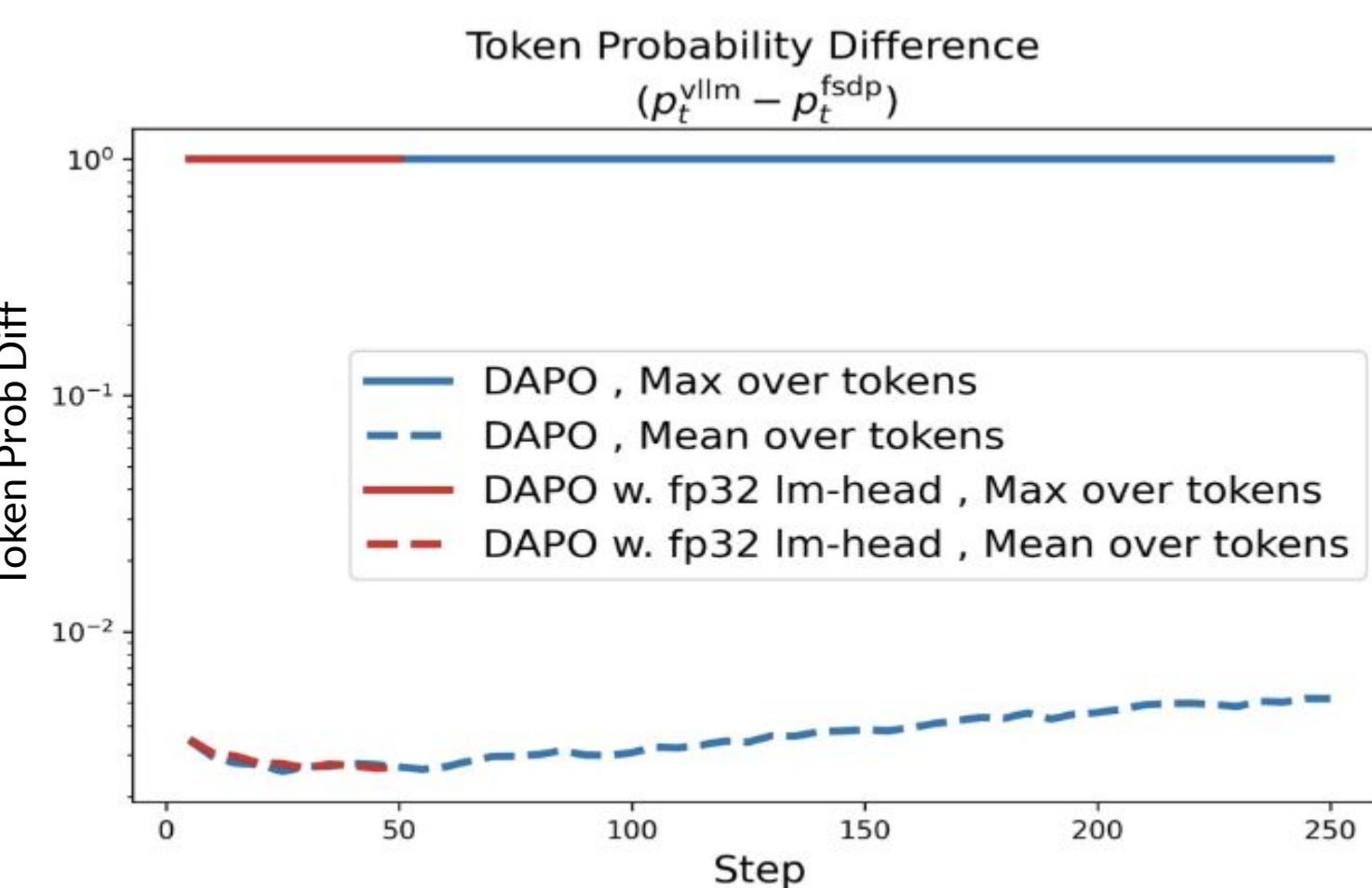


Abstract

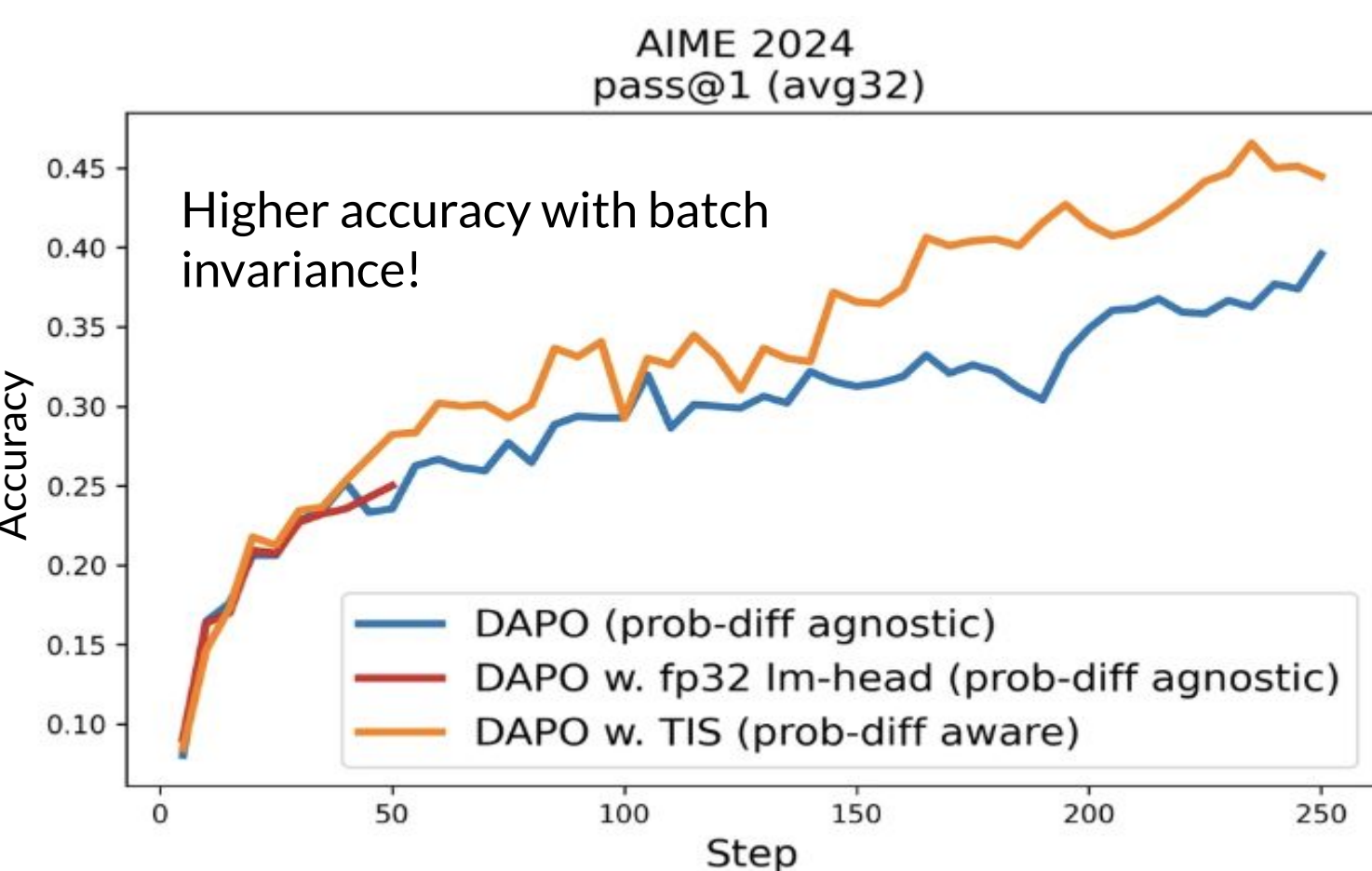
Reinforcement learning (RL) requires trainer-generator parity to achieve stable training. However, differences in attention for training vs inference cause varying floating point reduction ordering.

Let's explore how to prevent this for some of the most popular attention variants.

Trainer-generator Parity

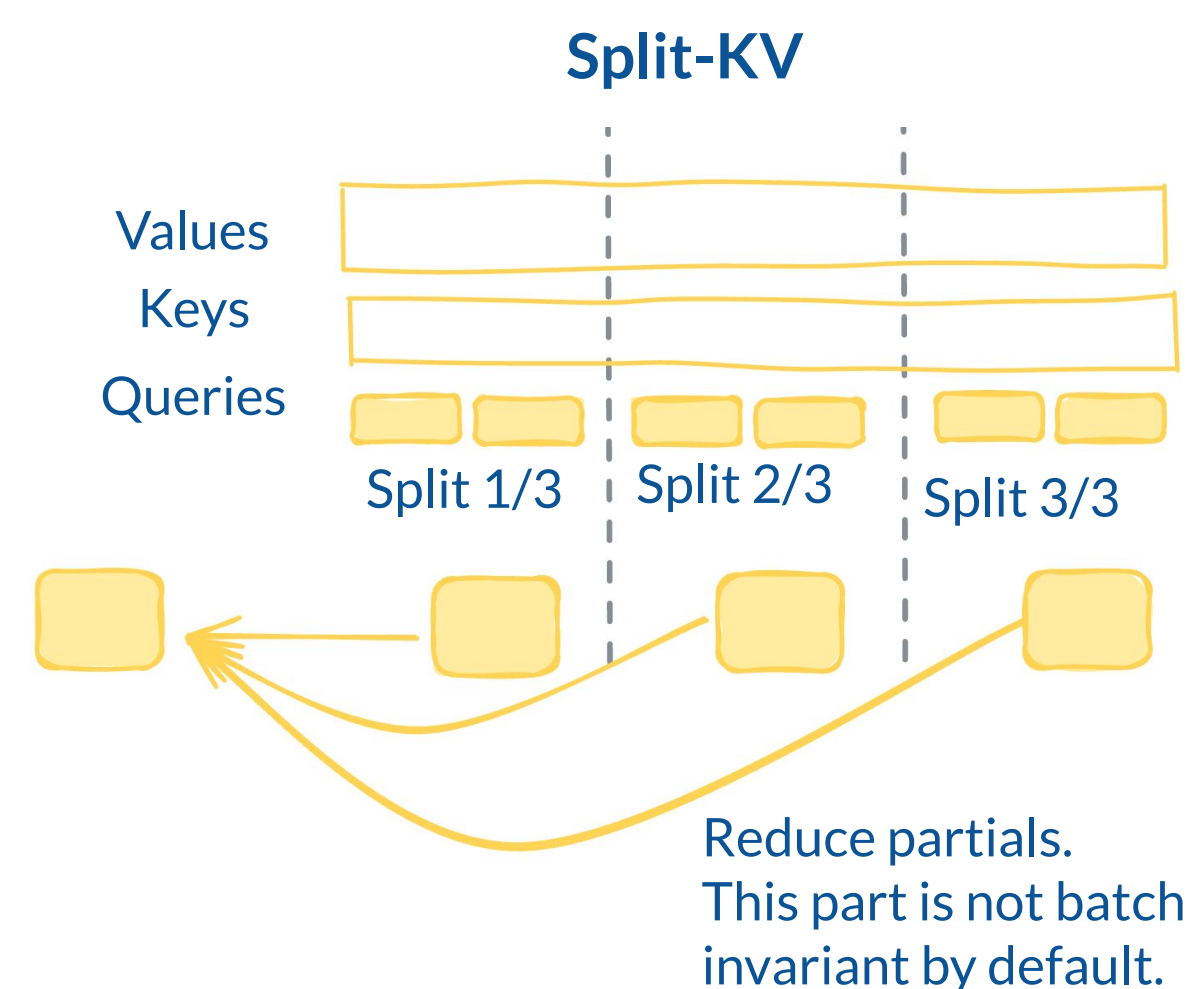


If an algorithm assumes on-policy RL but there's numerical noise, you are now unintentionally off-policy. This leads to larger KL-divergence and worse results.



To achieve batch invariance for attention, we need **one unified definition** between the trainer and the generator.

Varlen Attention



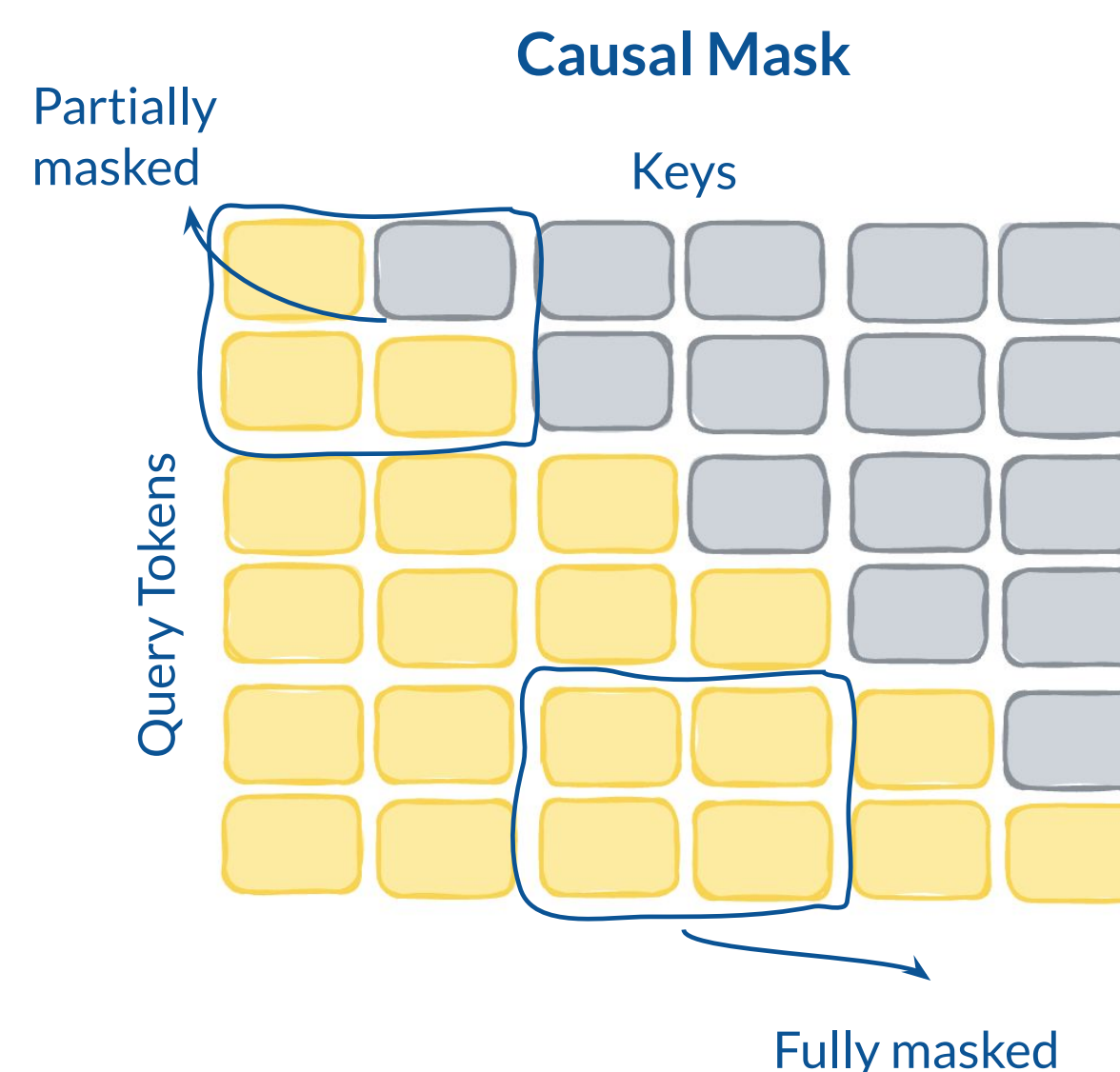
Reduction order of a token with split-KV cannot depend on how many other tokens are being processed.

For batch invariance, either disable split-KV or use fixed split sizes and reduction order.

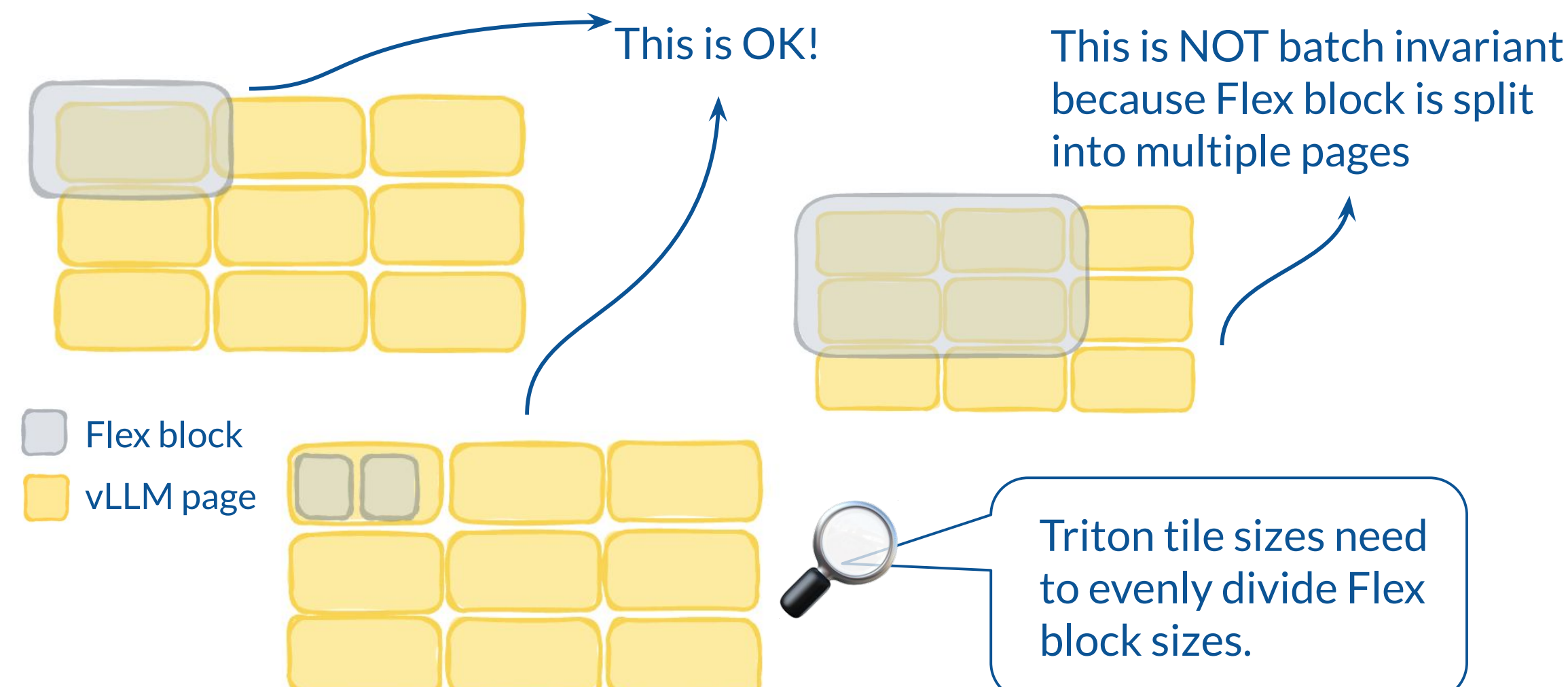
Flex Attention

- For batch invariance, Flex blocks cannot be categorized by batch dependent characteristics.

For example, we can't group partial/full blocks.



- Divisibility between Flex block, Triton tile, and vLLM page sizes must be preserved for batch invariant results.



Linear Attention (GDN & KDA)

The linear attention kernels in attention-gym were designed to be deterministic.

Recurrent:

```
state *= exp(g_t)
delta = (v_t - sum_over_K(k_t * state)) * beta_t
state += k_t * delta
o_t = sum_over_K(q_t * state)
```

One CTA per work item. Each program executes its sequence in order.

Chunked:

```
token_start = begin + local_chunk * chunk_size
valid_tokens = cutlass.min(chunk_size, end - token_start)
```

Each sequence's chunks start at its t0. No sum crosses a chunk or CTA so results are not batch dependent.

Performance: TorchTitan with Kimi K3

Recurrent vs chunked kernels aren't bitwise identical, so we need to use the same kernel for prefill and decode. We observed that replaying the chunked kernel for decode is **63%** faster than using the recurrent kernel for prefill.

Decode Algorithm with Chunked Kernel

