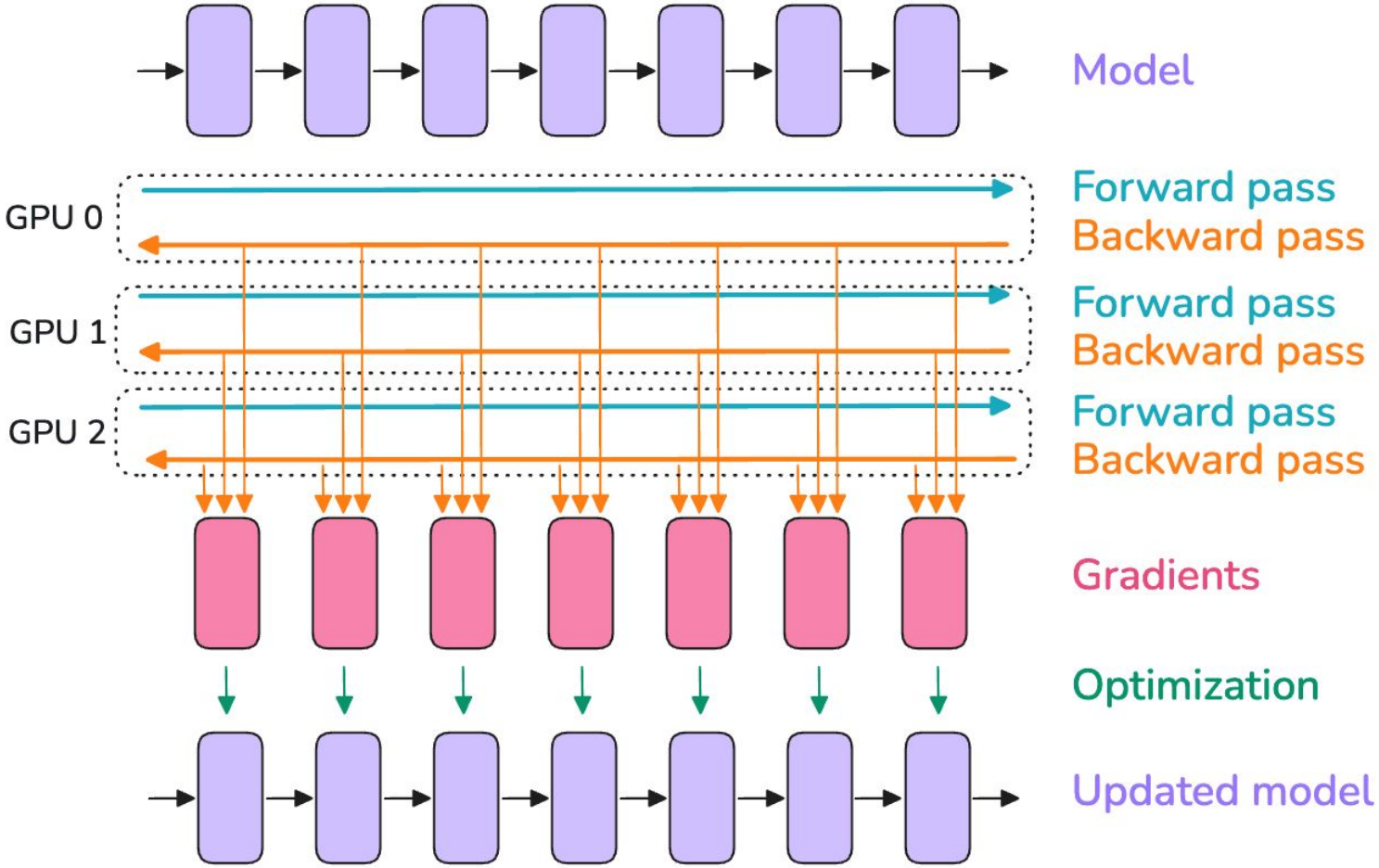
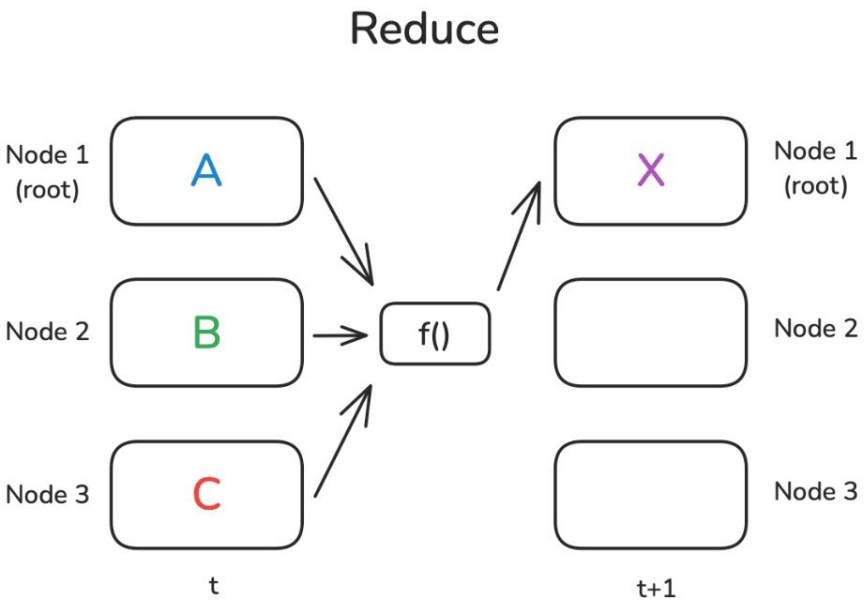
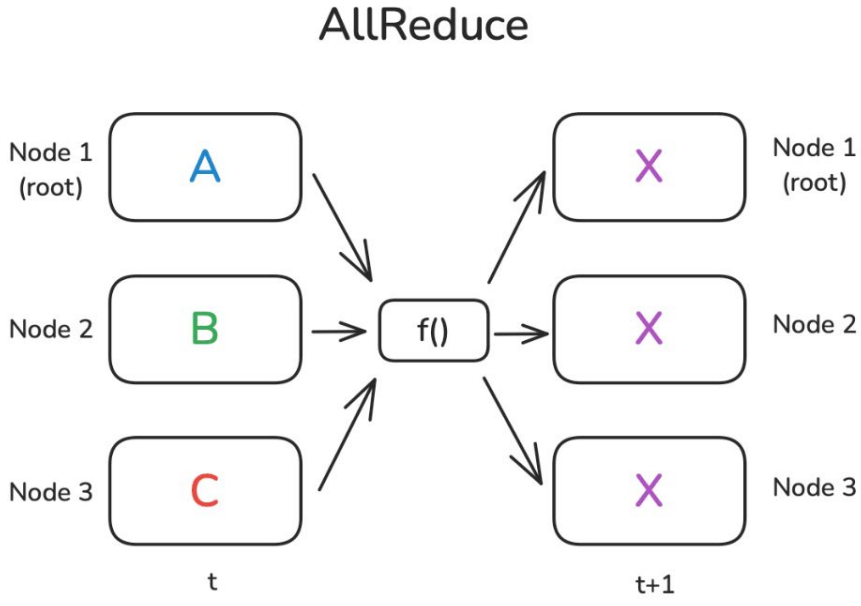
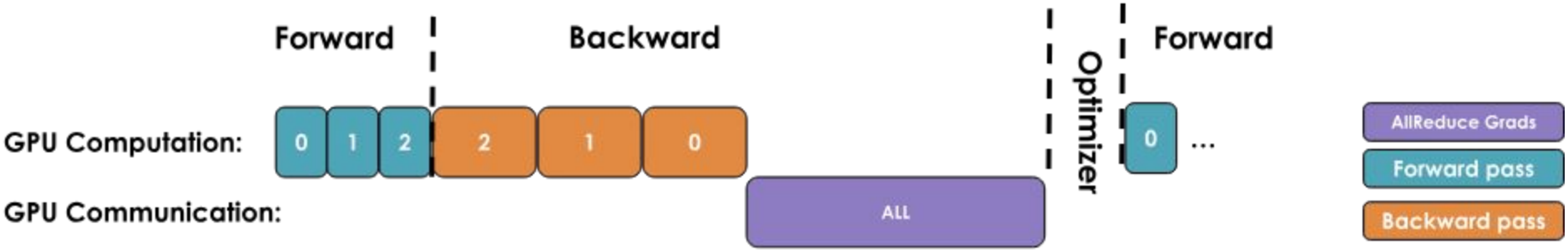
Data Parallelism [:ZERO]

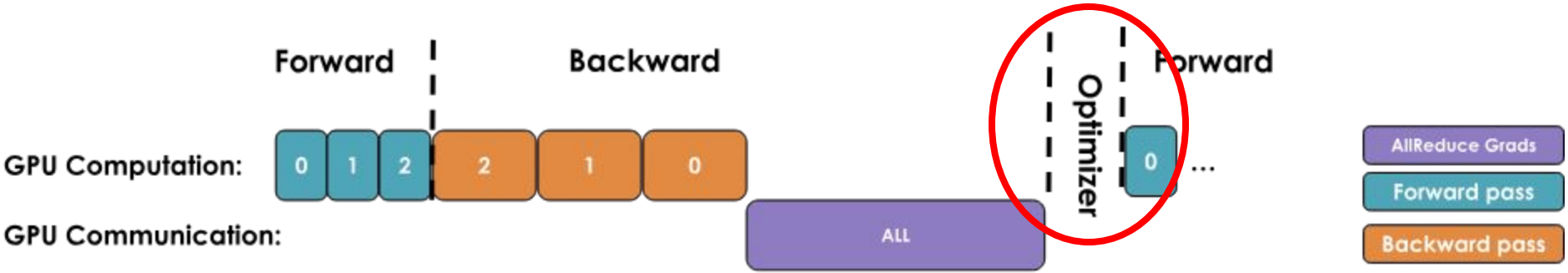
made with ❤️ for “Little ML book club”

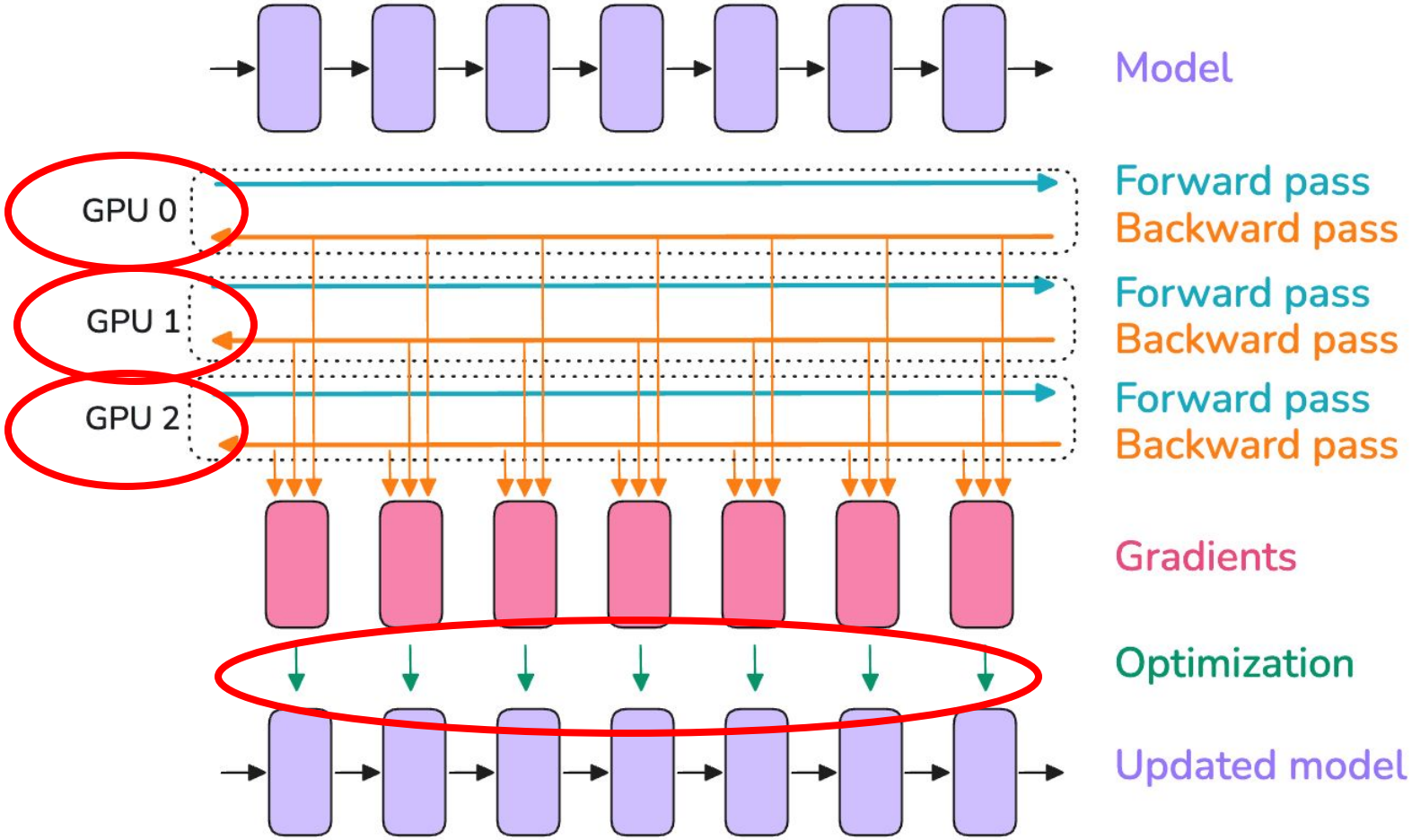

Model
GPU 0
Forward pass
Backward pass
GPU 1
Forward pass
Backward pass
GPU 2
Forward pass
Backward pass
Gradients
Optimization
Updated model

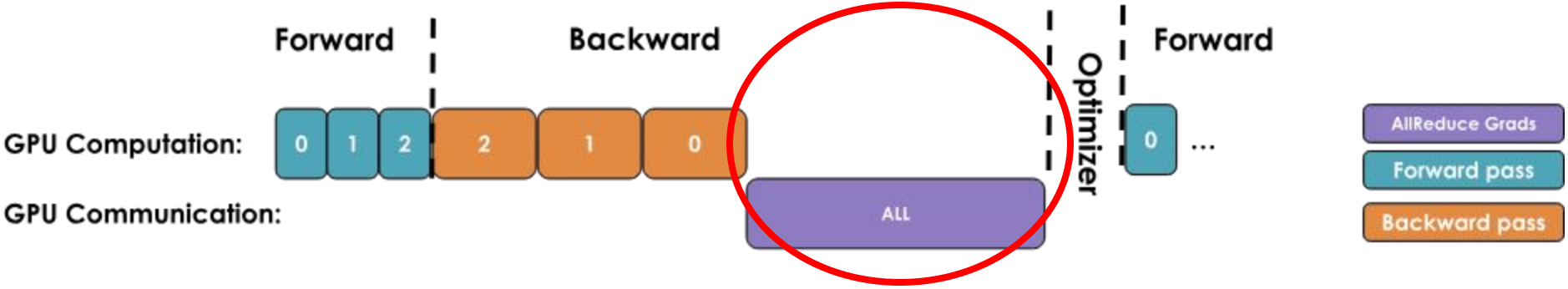

Reduce
Node 1
(root)
A
Node 2
B
Node 3
C
f()
X
Node 1
(root)
Node 2
Node 3
t
t+1

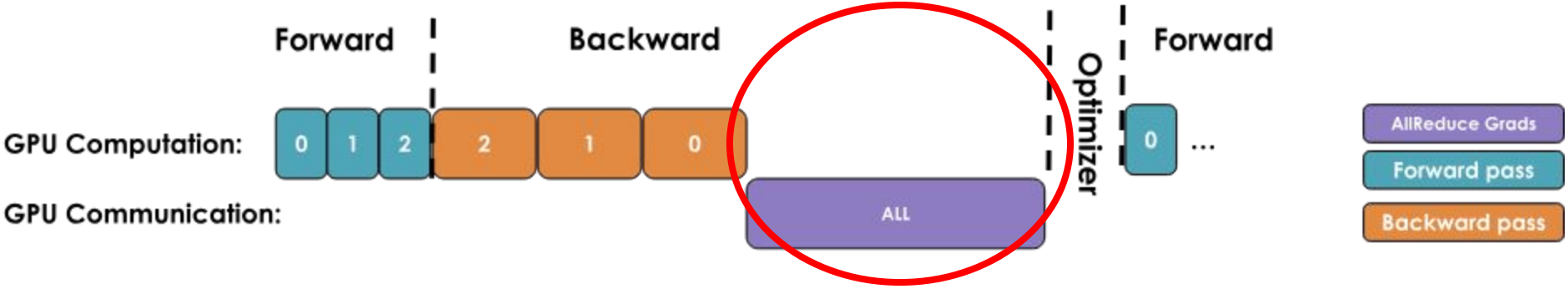

AllReduce
Node 1
(root)
A
Node 2
B
Node 3
C
f()
X
X
X
Node 1
(root)
Node 2
Node 3
t
t+1


Forward
Backward
Optimizer
Forward
GPU Computation:
GPU Communication:
0
1
2
2
1
0
ALL
0
...
AllReduce Grads
Forward pass
Backward pass

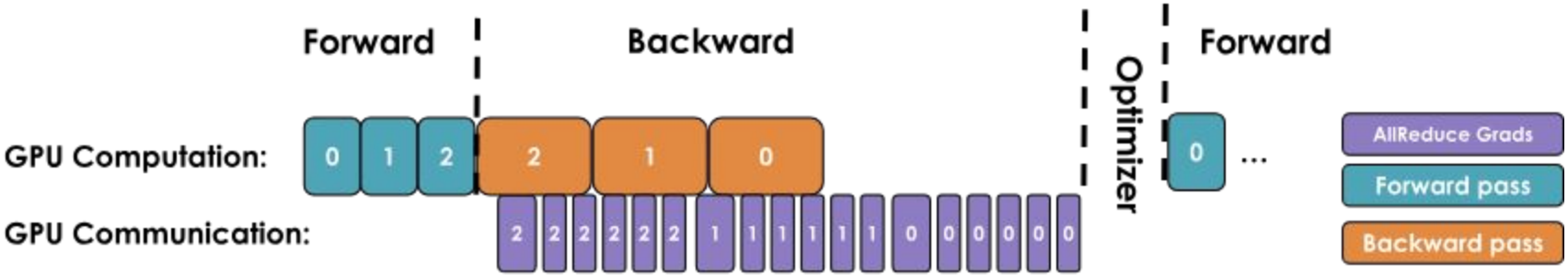

Forward
Backward
Optimizer
Forward
GPU Computation:
0
1
2
2
1
0
0
...
GPU Communication:
ALL
AllReduce Grads
Forward pass
Backward pass

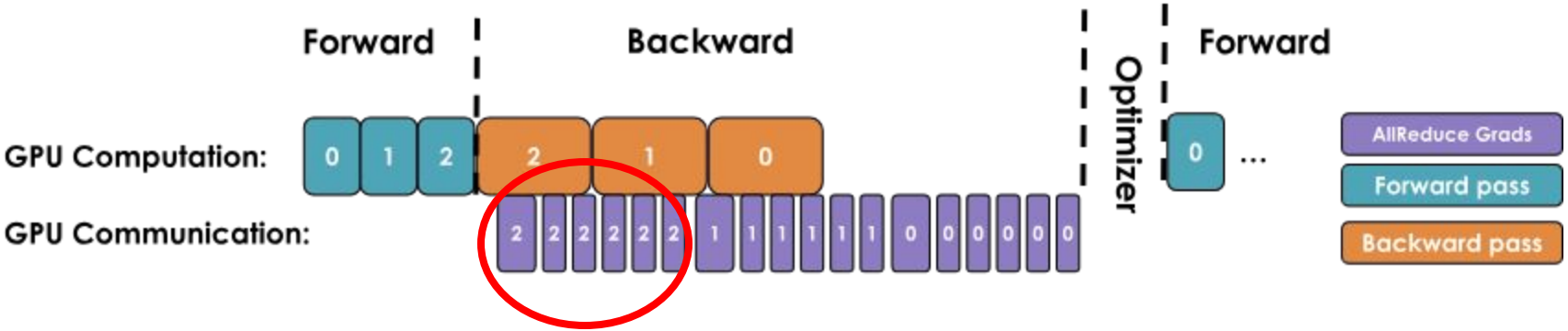

Model
GPU 0
GPU 1
GPU 2
Forward pass
Backward pass
Forward pass
Backward pass
Forward pass
Backward pass
Gradients
Optimization
Updated model

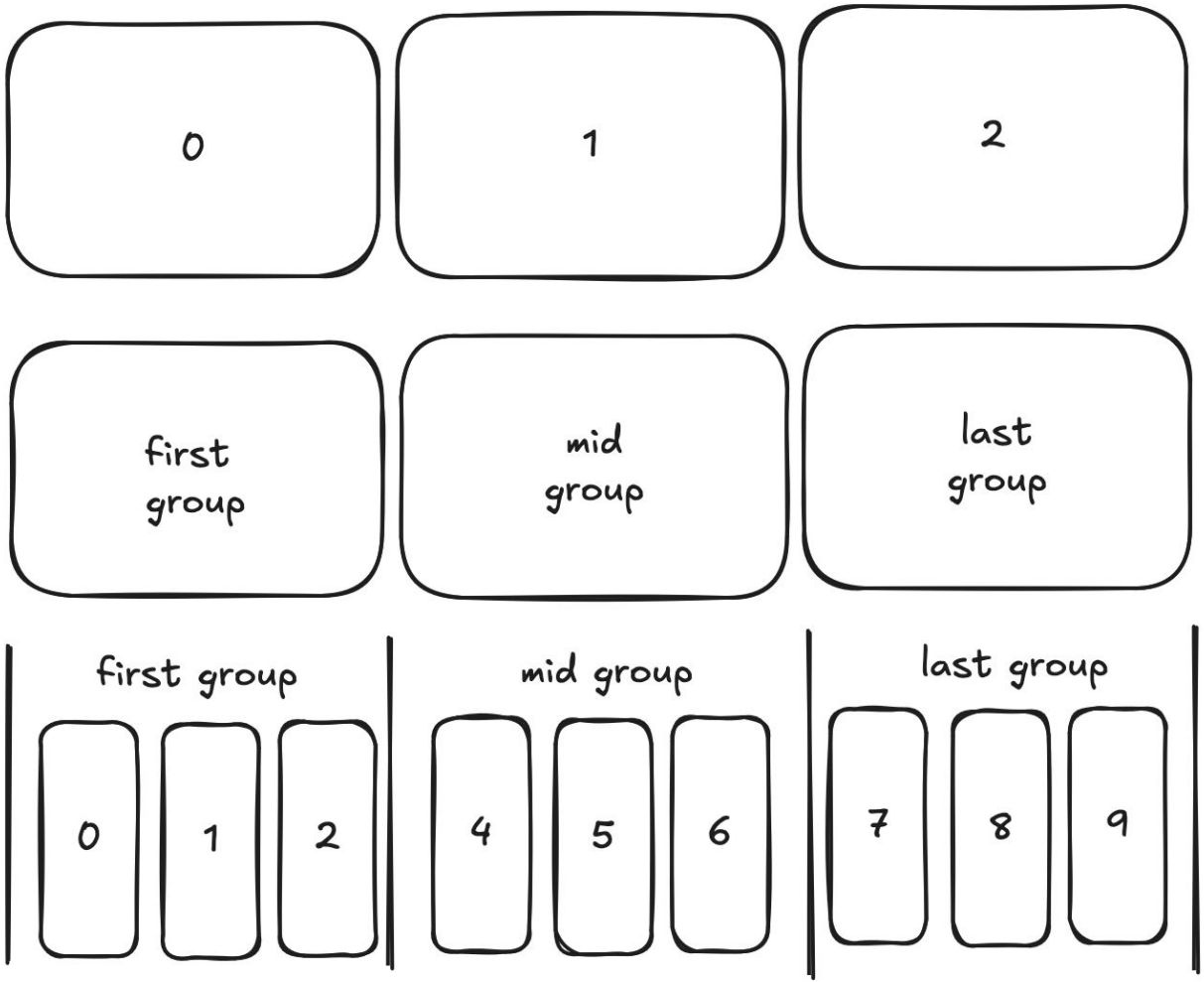

Forward
Backward
Optimizer
Forward
GPU Computation:
0
1
2
2
1
0
0
...
GPU Communication:
ALL
AllReduce Grads
Forward pass
Backward pass



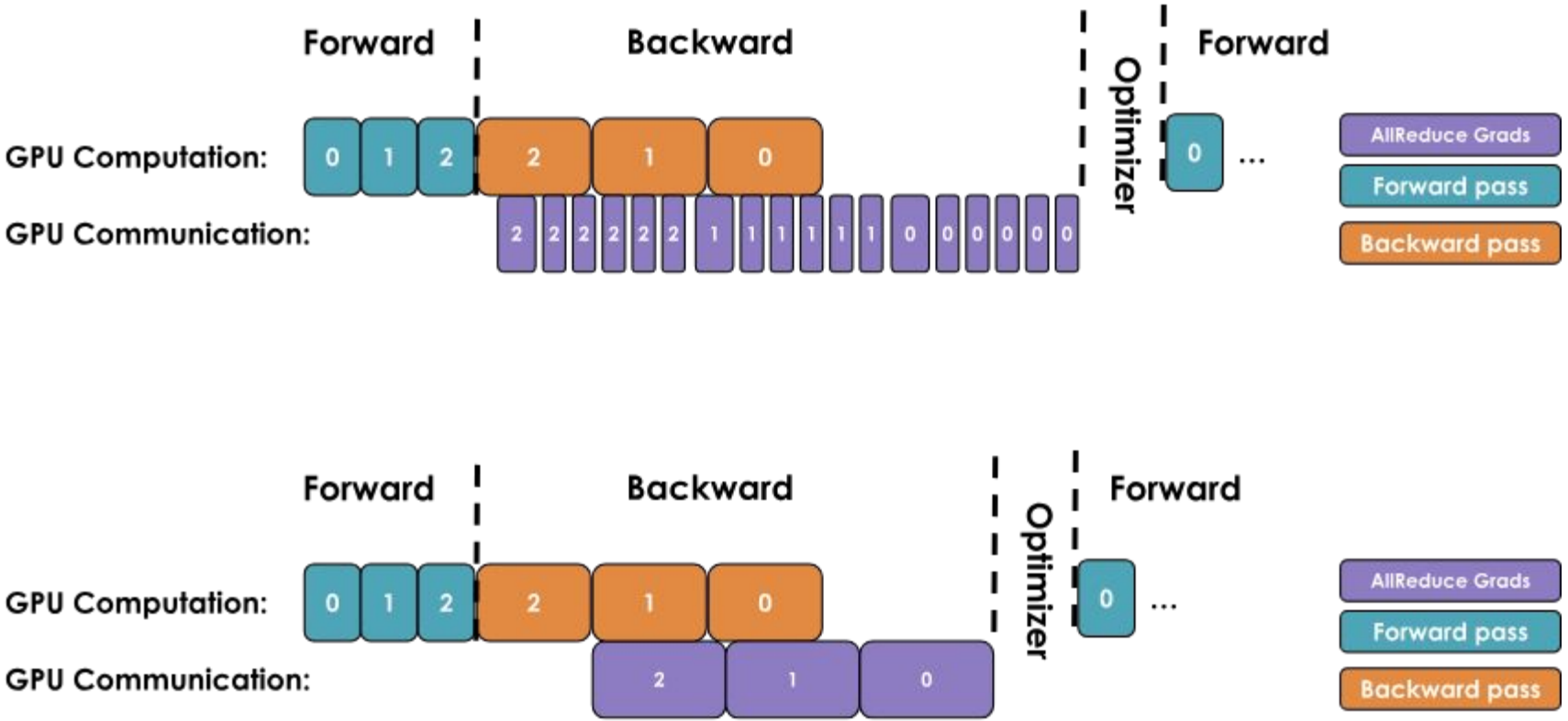
NOTE: Each GPU needs memory for all grads!

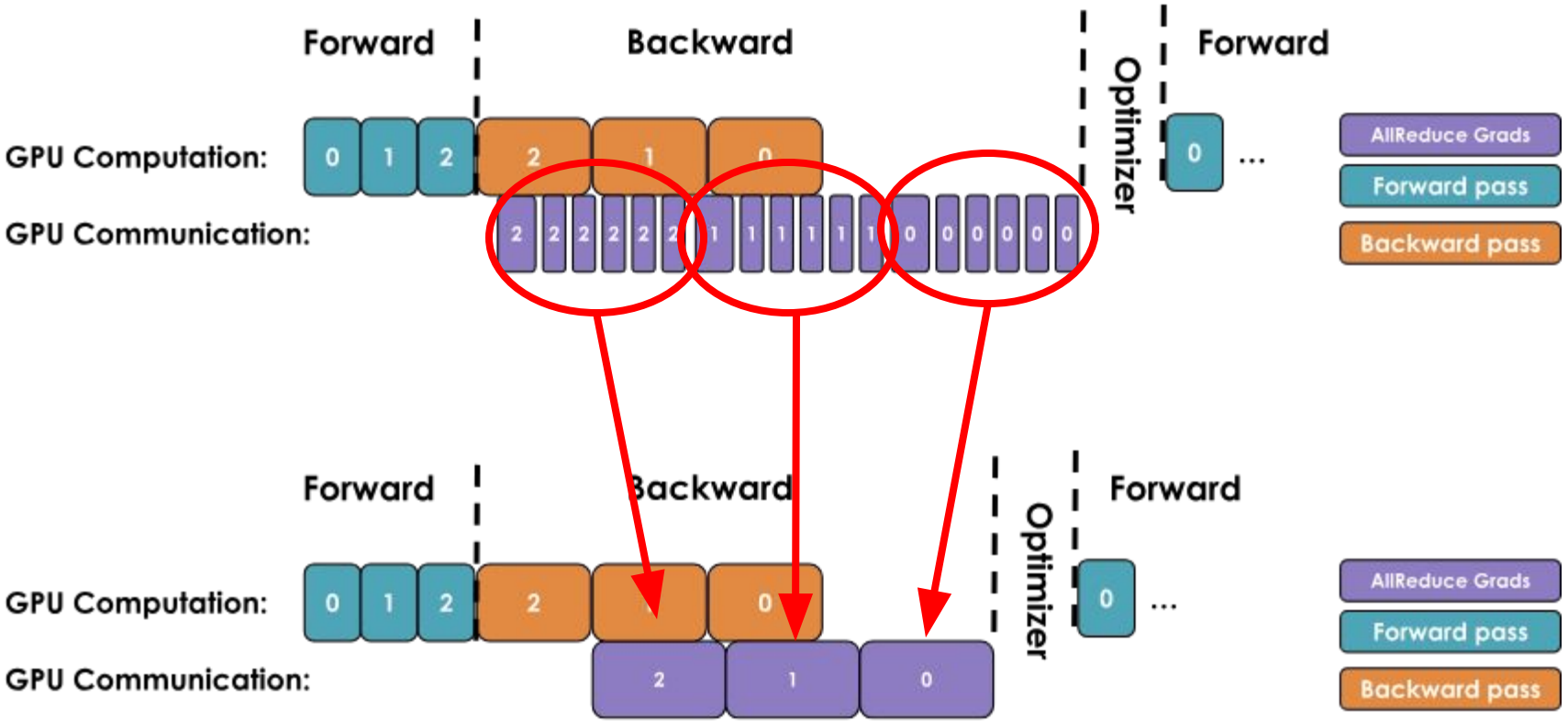

Forward
Backward
Optimizer
Forward
GPU Computation:
GPU Communication:
0 1 2 2 1 0 0 ...
2 2 2 2 2 2 1 1 1 1 1 1 0 0 0 0 0 0
AllReduce Grads
Forward pass
Backward pass


Forward
Backward
Optimizer
Forward
GPU Computation:
GPU Communication:
0
1
2
2
1
0
0
...
AllReduce Grads
Forward pass
Backward pass

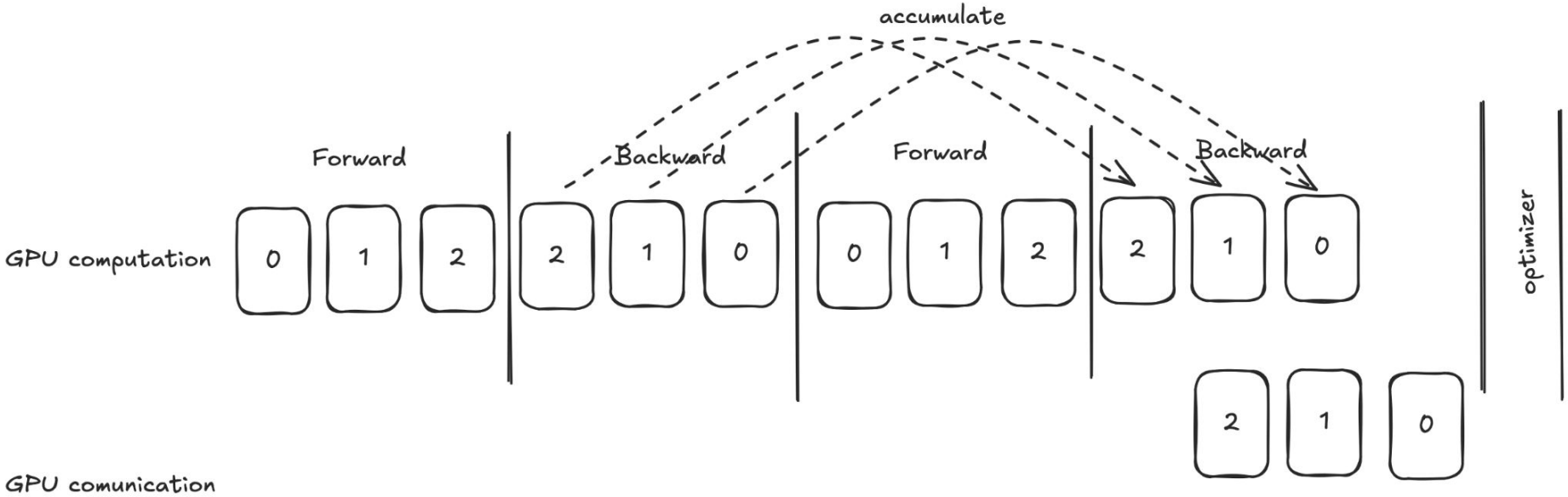

0
1
2
first group
mid group
last group
first group
0
1
2
mid group
4
5
6
last group
7
8
9

```python
def _allreduce_grads(self, grad):
    """
    Performs an all-reduce operation to synchronize gradients across multiple processes.
    """
    # No synchronization needed during gradient accumulation, except at the final accumulation step.
    if self.require_backward_grad_sync:
        dist.all_reduce(
            grad, op=dist.ReduceOp.SUM, group=pgm.process_group_manager.cp_dp_group
        )
        grad /= pgm.process_group_manager.cp_dp_world_size
    return grad
```


Forward
Backward
Optimizer
Forward
GPU Computation:
GPU Communication:
0 1 2 2 1 0 0 …
2 2 2 2 2 2 1 1 1 1 1 1 0 0 0 0 0 0
AllReduce Grads
Forward pass
Backward pass
Forward
Backward
Optimizer
Forward
GPU Computation:
GPU Communication:
0 1 2 2 1 0 0 …
2 1 0
AllReduce Grads
Forward pass
Backward pass


Forward
Backward
Optimizer
Forward
GPU Computation:
GPU Communication:
AllReduce Grads
Forward pass
Backward pass
Forward
Backward
Optimizer
Forward
GPU Computation:
GPU Communication:
AllReduce Grads
Forward pass
Backward pass

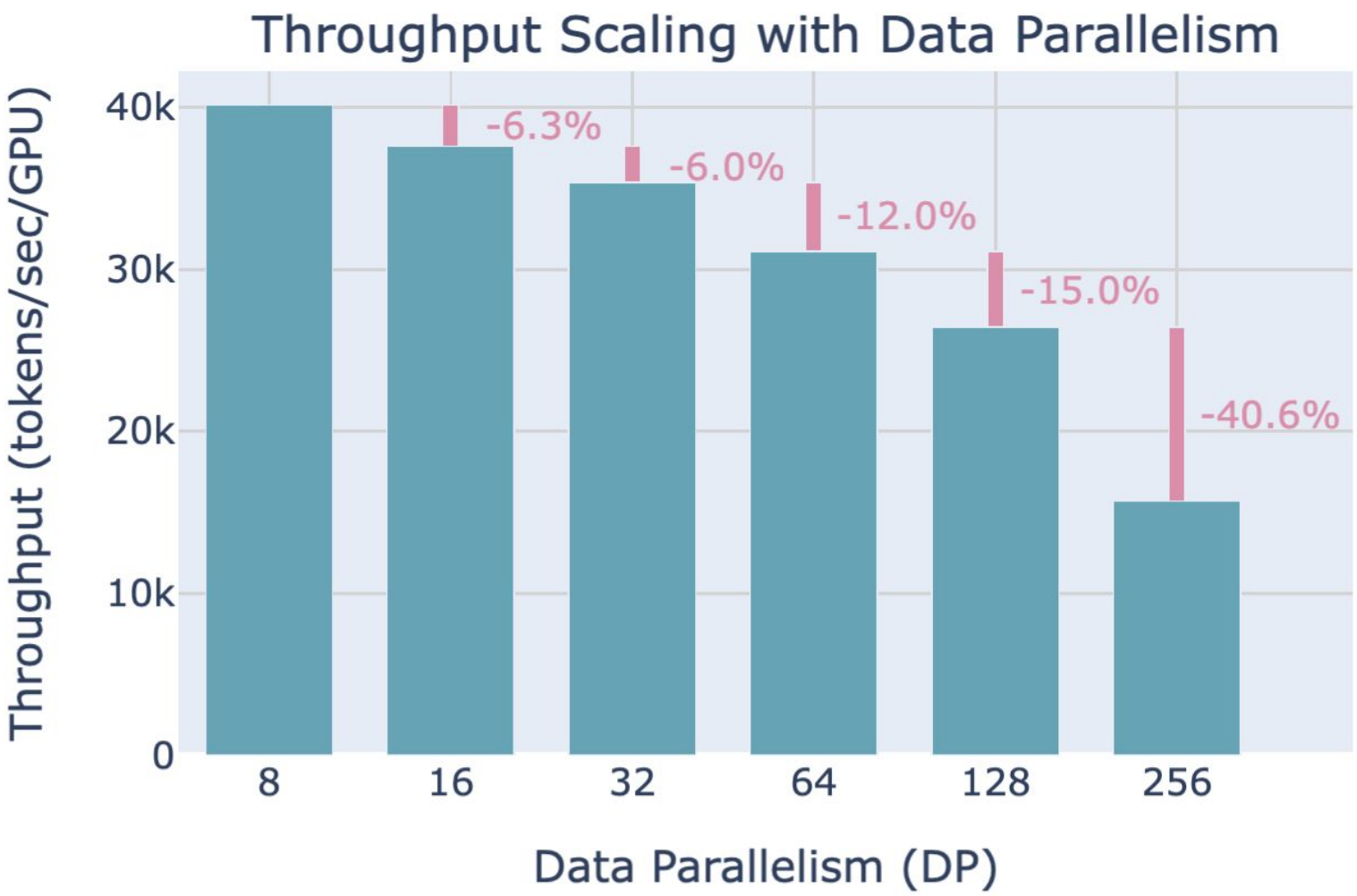
mom,
what if my batch size is
not big enough
for optimal
convergence?


accumulate
Forward
Backward
Forward
Backward
optimizer
GPU computation
0
1
2
2
1
0
0
1
2
2
1
0
2
1
0
GPU comunication

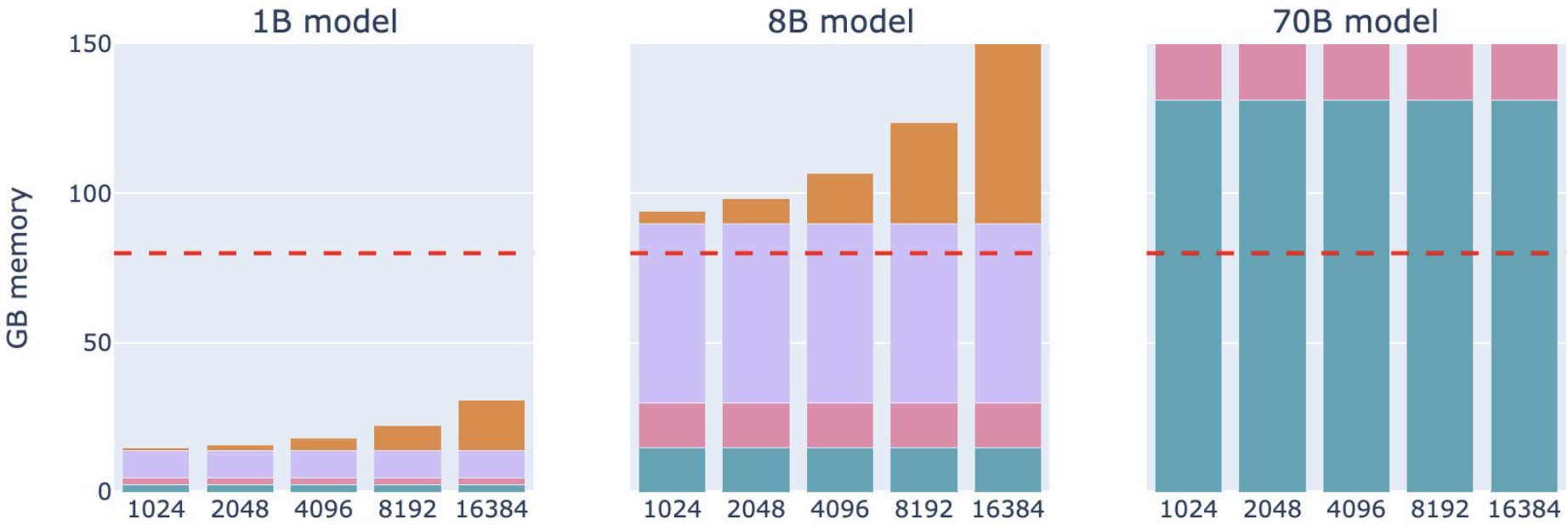
$$bs = gbs = mbs \times grad\_acc \times dp$$

Recipe

1. Find best (global) batch size in tokens (sacred empirical knowledge)
2. Find a sequence length (sacred empirical knowledge). Generally, 2-8k tokens.
3. Find the maximum local batch size on a single GPU by increasing the local batch size until we run out of memory.
4. Find the number of available GPUs for our target (dp). The ratio of gbs to dp gives us the remaining number of gradient accumulation steps needed for the desired gbs .


Throughput Scaling with Data Parallelism
Throughput (tokens/sec/GPU)
40k
30k
20k
10k
0
-6.3%
-6.0%
-12.0%
-15.0%
-40.6%
8
16
32
64
128
256
Data Parallelism (DP)

mom,
what if my model is too large?


1B model
8B model
70B model
GB memory
150
100
50
0
1024
2048
4096
8192
16384

see you next week
to solve it