Pipelining Persistent Kernels

Paweł Szczerbuk OpenAI pawel.szczerbuk@openai.com

Agenda

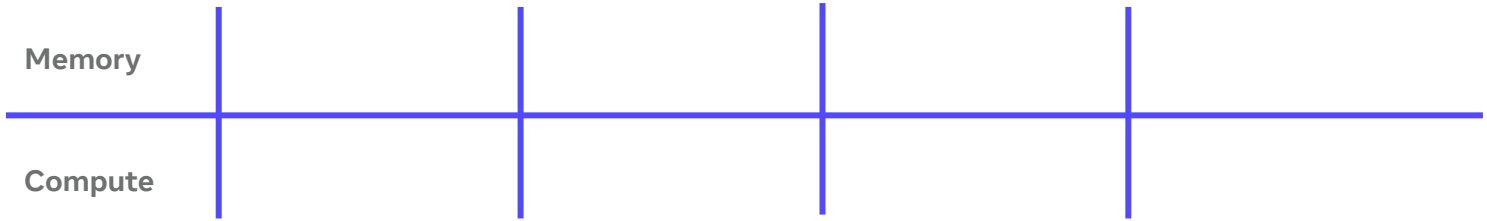
- Software Pipelining
- Persistent Kernels
- Putting the Two Together…
- … Efficiently

Software Pipelining

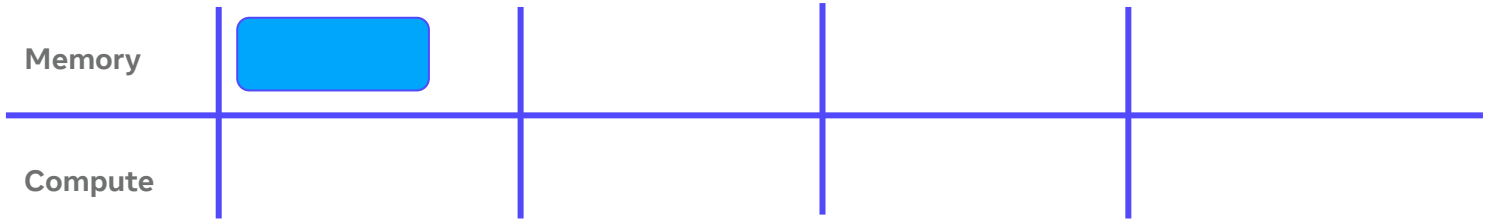
SOFTWARE PIPELINING

```
for k in range(0, k_tiles):
  a = tl.load(a_ptrs)
  b = tl.load(b_ptrs)
  c = tl.dot(a, b, c)
  a_ptrs, b_ptrs = …
tl.store(c_ptrs, c)
```

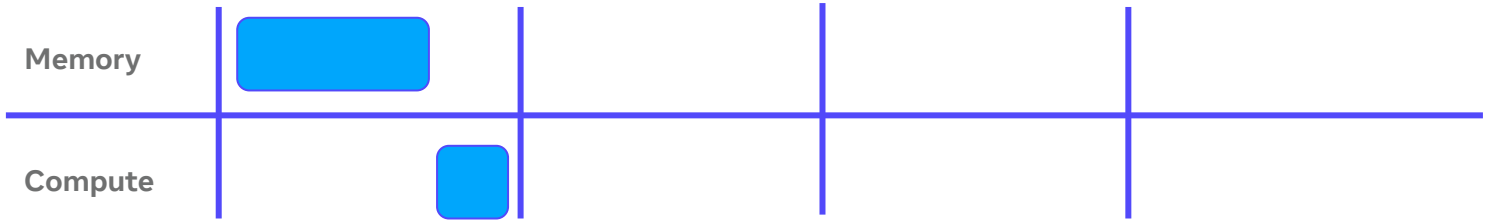
SOFTWARE PIPELINING



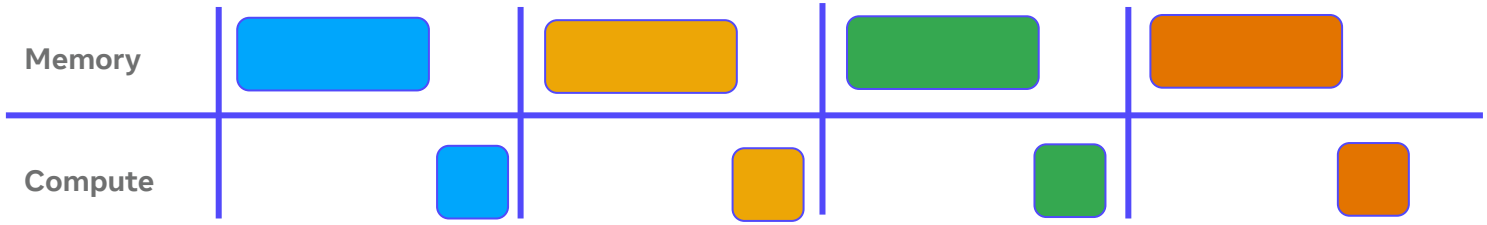
SOFTWARE PIPELINING



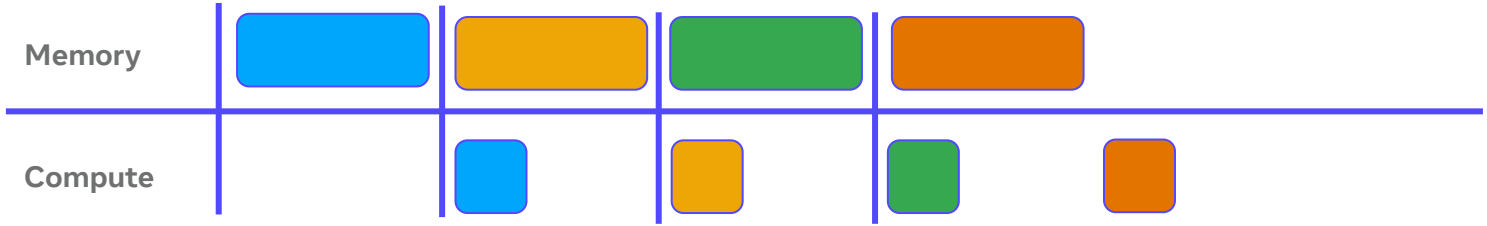
SOFTWARE PIPELINING



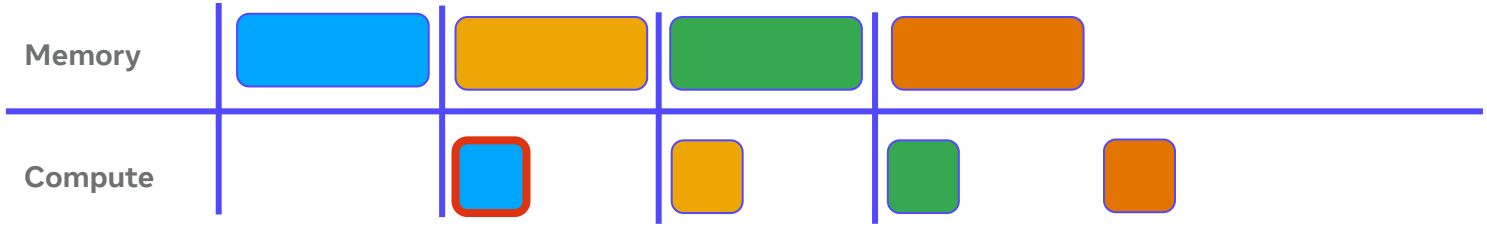
SOFTWARE PIPELINING



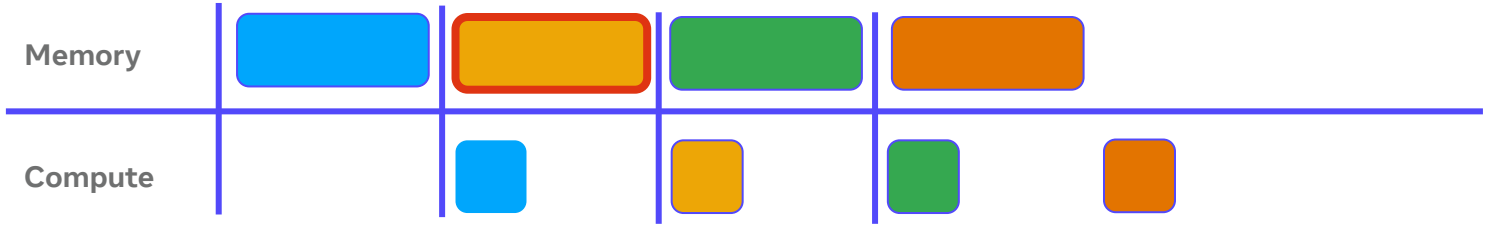
SOFTWARE PIPELINING



SOFTWARE PIPELINING

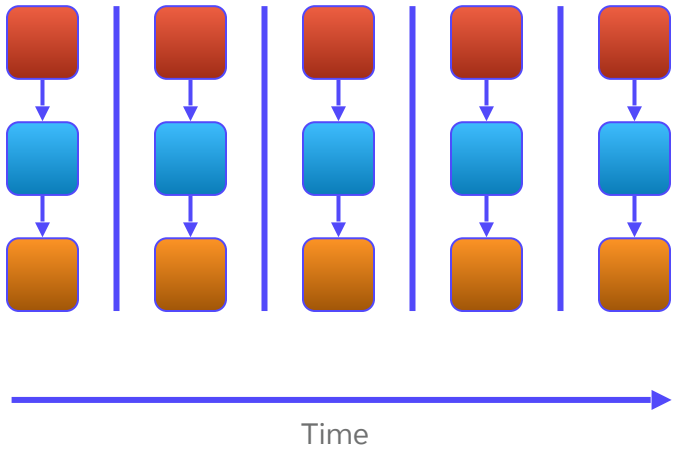


SOFTWARE PIPELINING

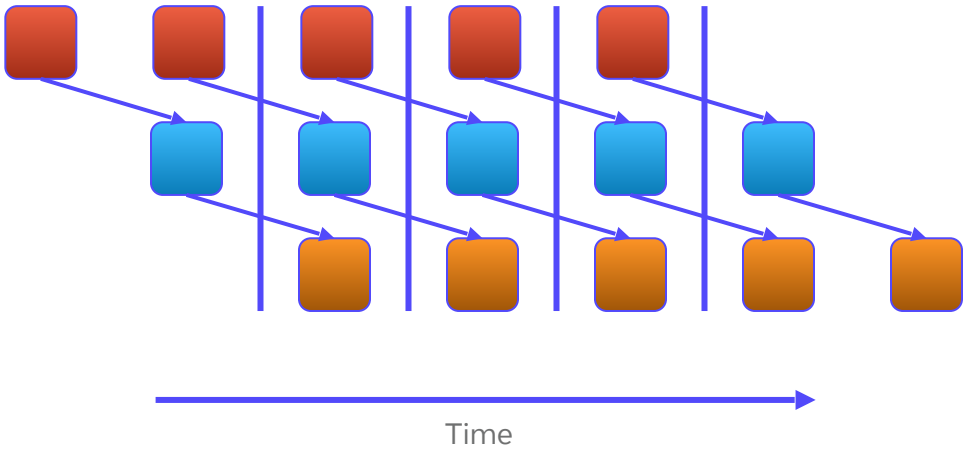


SOFTWARE PIPELINING

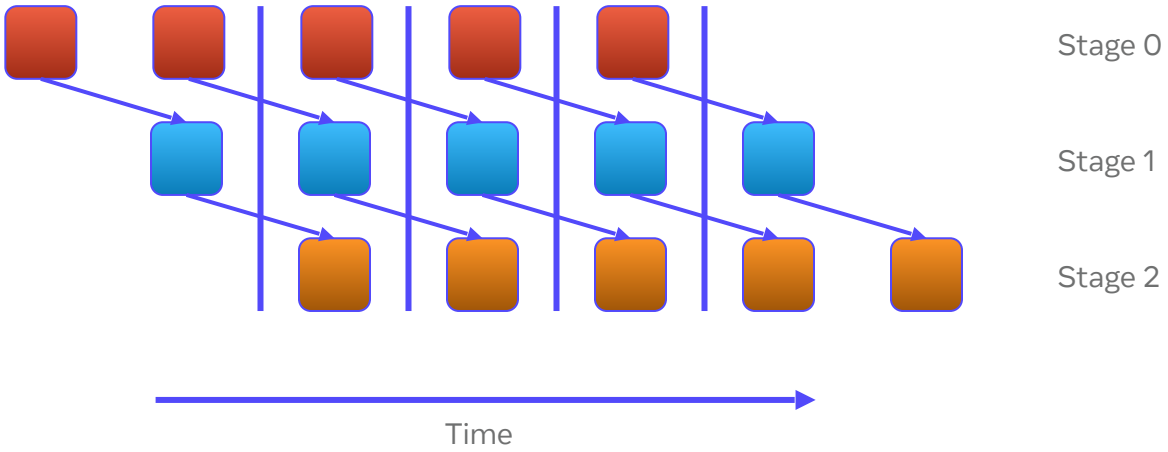
SOFTWARE PIPELINING



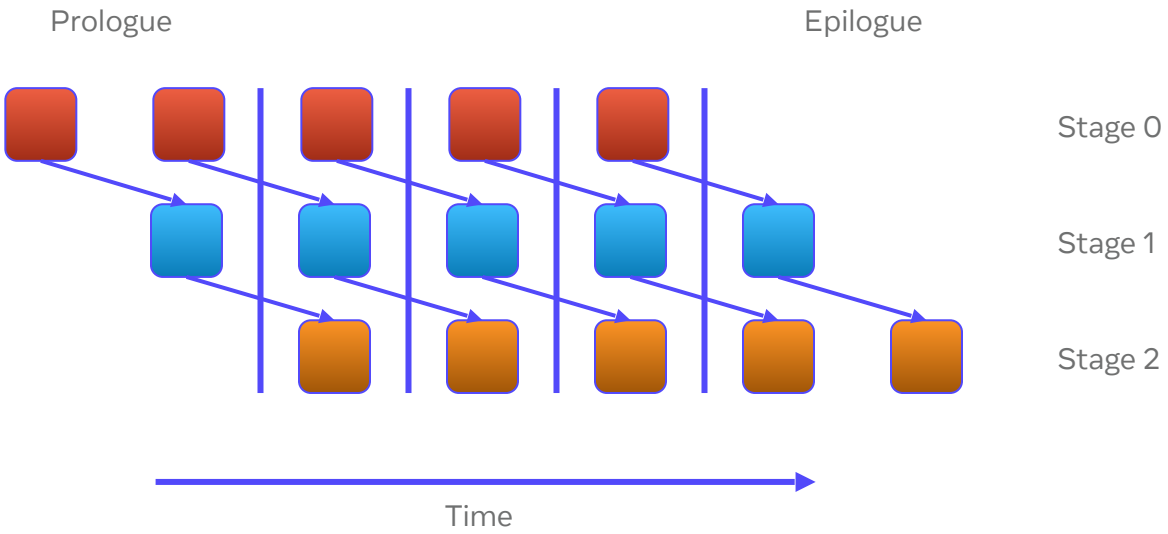
SOFTWARE PIPELINING



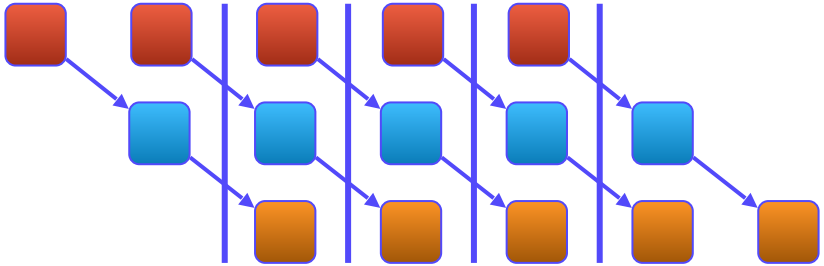
SOFTWARE PIPELINING



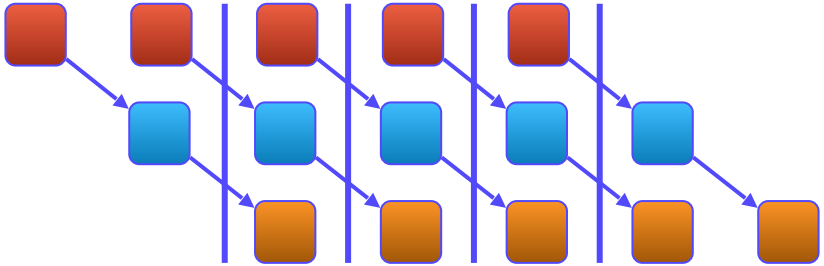
SOFTWARE PIPELINING

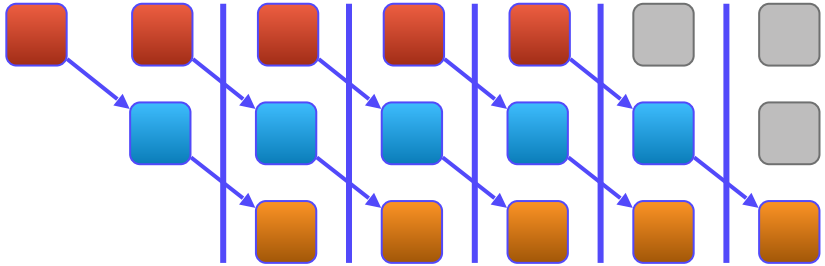


SOFTWARE PIPELINING - PROLOGUE/EPILOGUE PEELING/MASKING

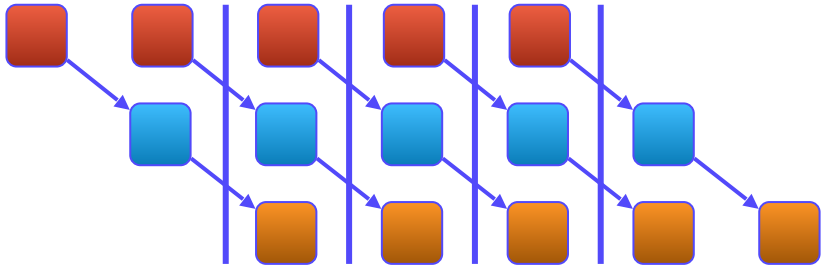


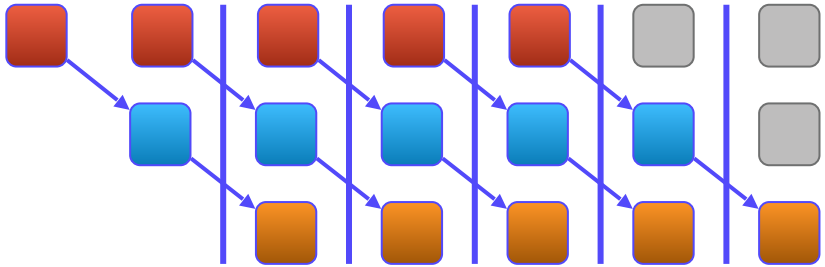
SOFTWARE PIPELINING - PROLOGUE/EPILOGUE PEELING/MASKING

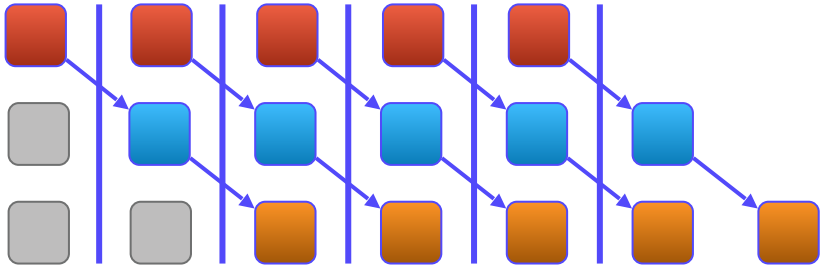




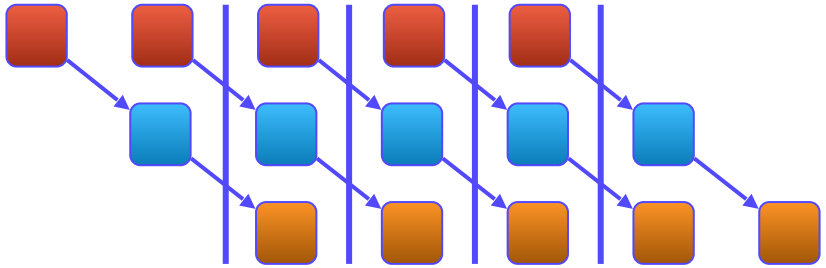
SOFTWARE PIPELINING - PROLOGUE/EPILOGUE PEELING/MASKING

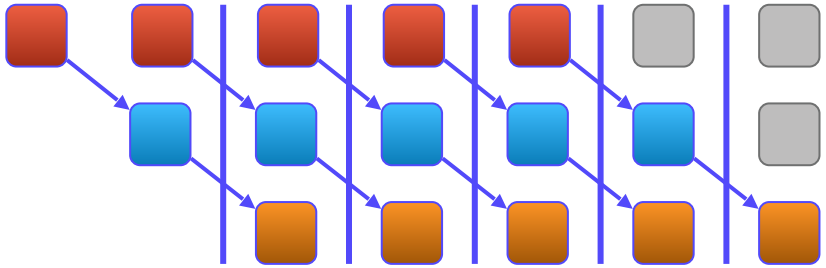


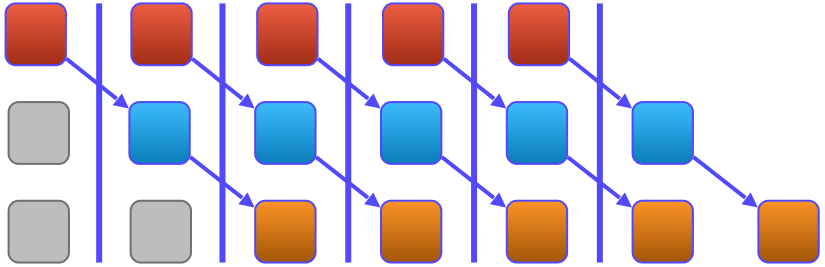


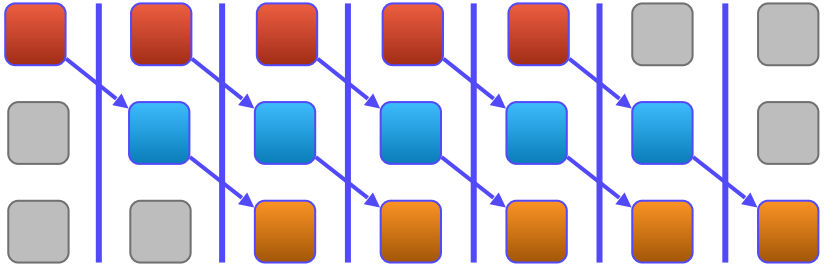


SOFTWARE PIPELINING - PROLOGUE/EPILOGUE PEELING/MASKING

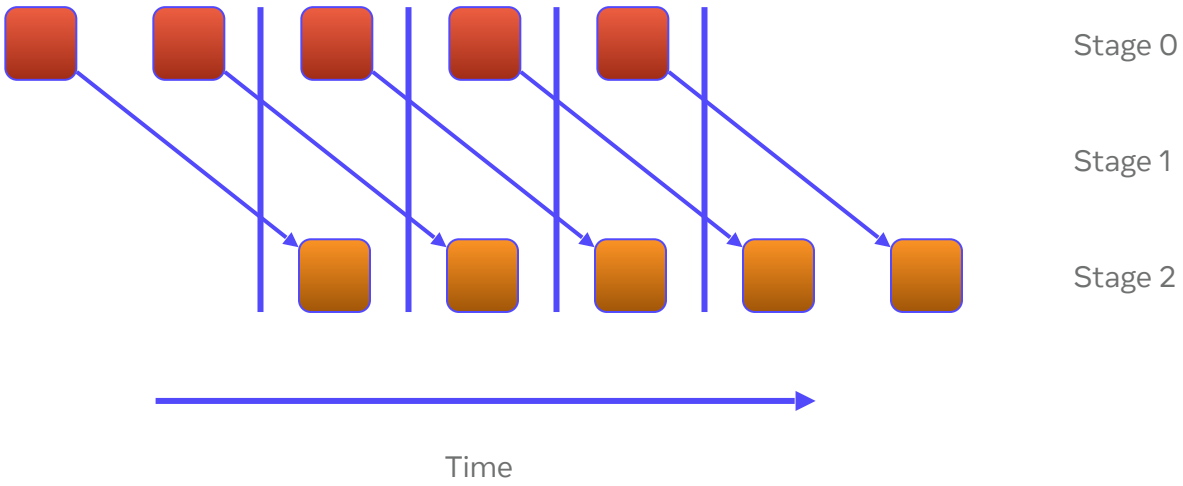








SOFTWARE PIPELINING

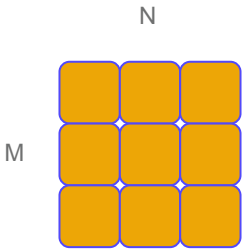


Software Pipelining

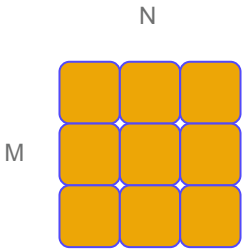
- Additional transformations required:
  - Multi buffering
  - Sync → async operations

- Amazing Pipeline Expander in MLIR upstream!

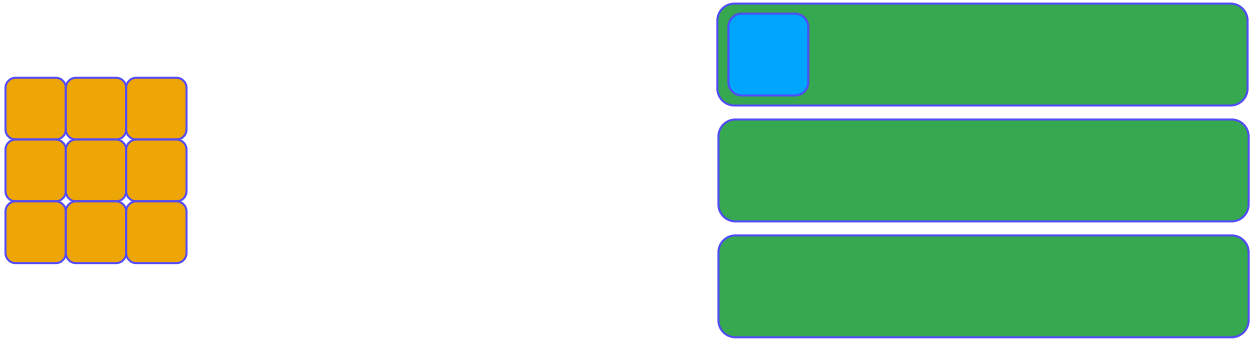
PERSISTENT KERNELS



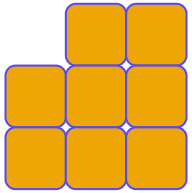
PERSISTENT KERNELS



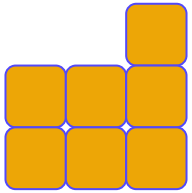
PERSISTENT KERNELS



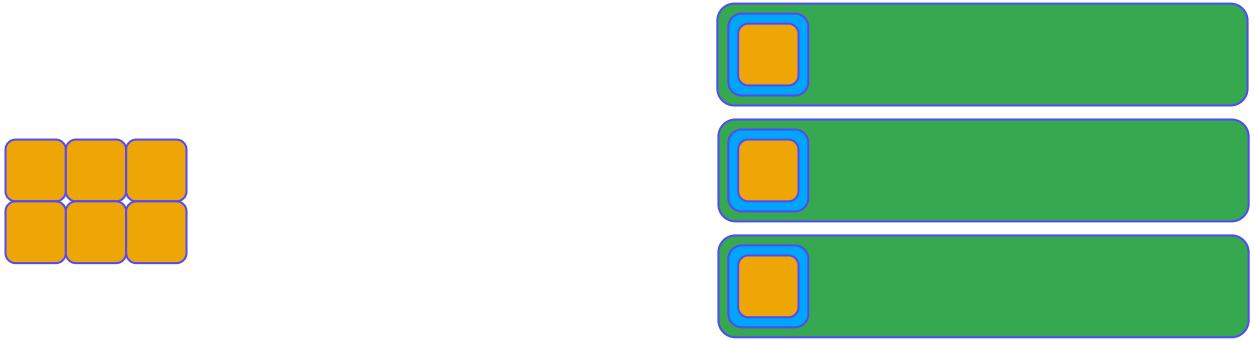
PERSISTENT KERNELS



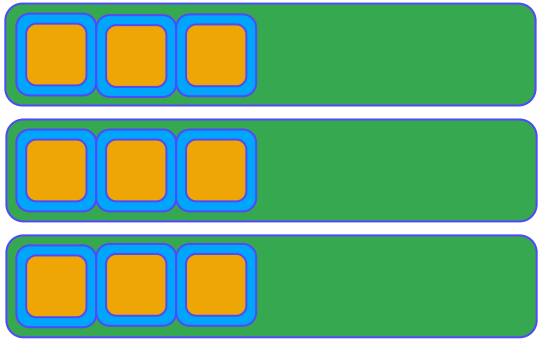
PERSISTENT KERNELS



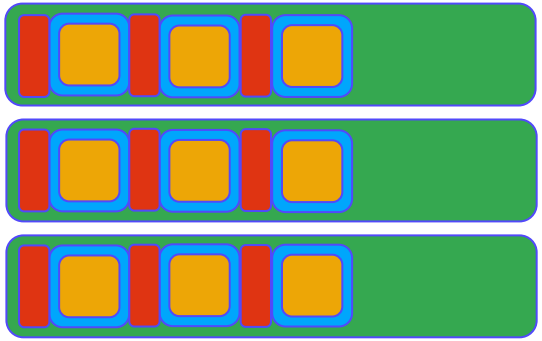
PERSISTENT KERNELS



PERSISTENT KERNELS

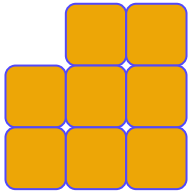


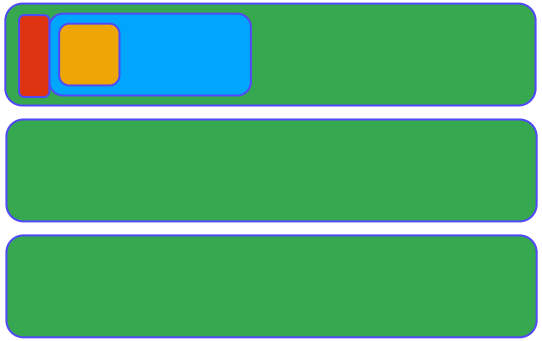
PERSISTENT KERNELS



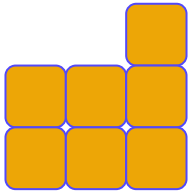
PERSISTENT KERNELS

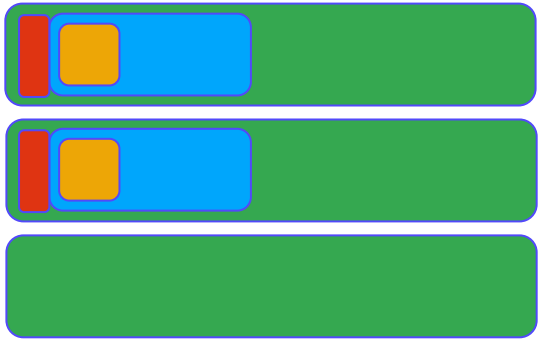
PERSISTENT KERNELS



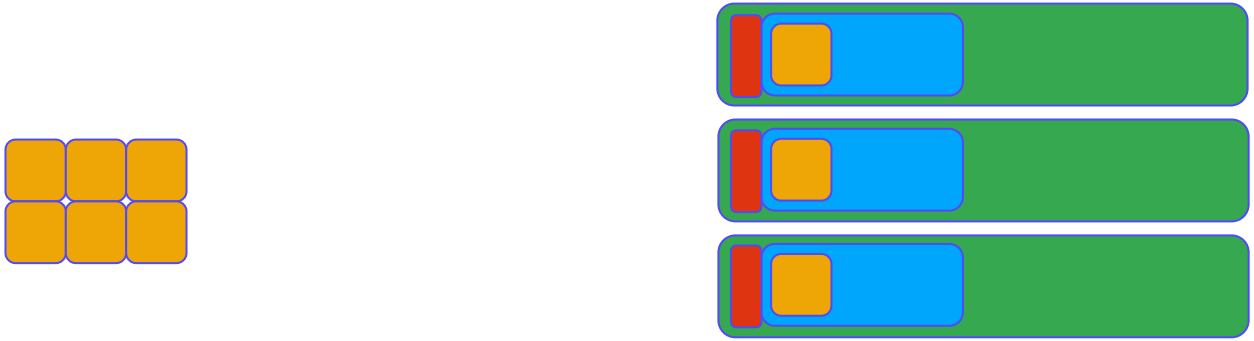


PERSISTENT KERNELS

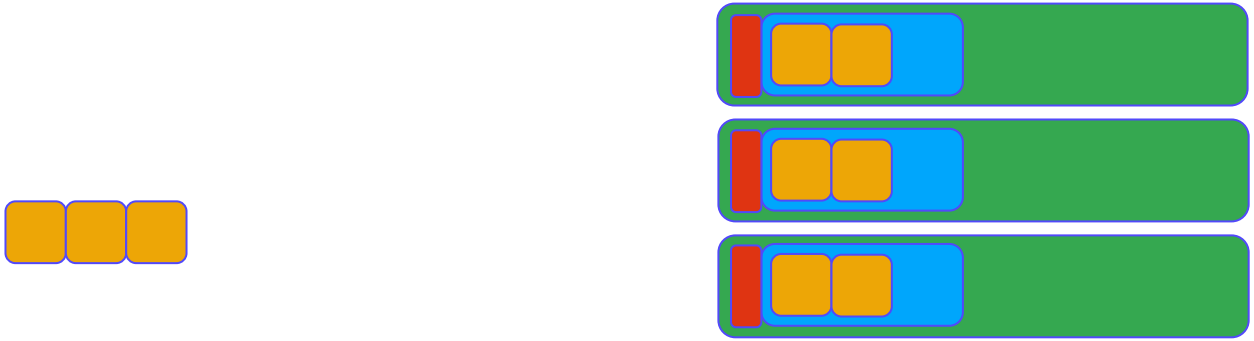




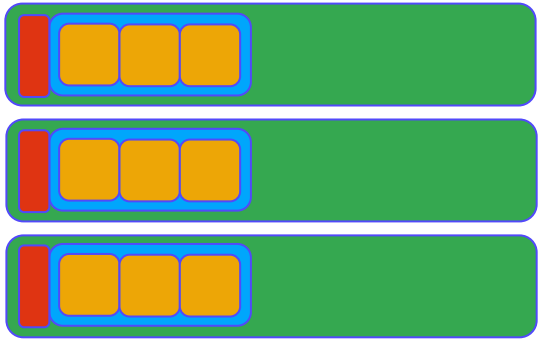
PERSISTENT KERNELS



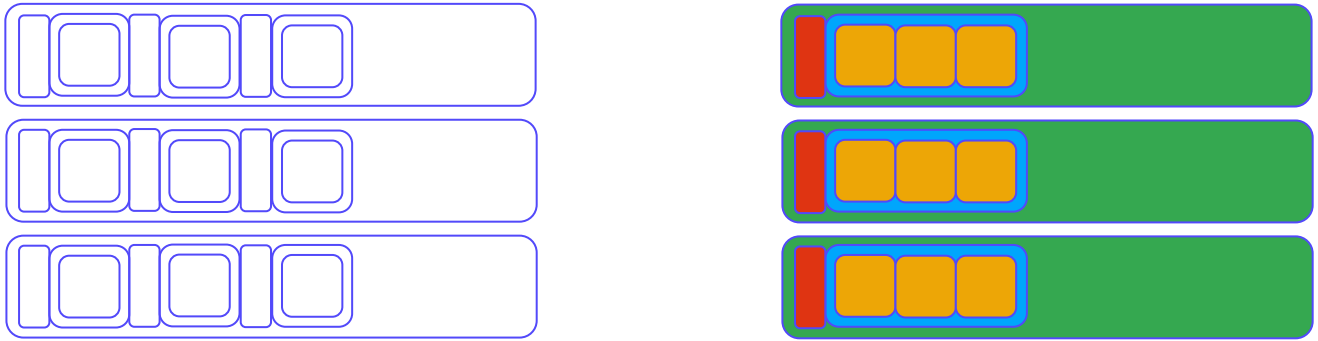
PERSISTENT KERNELS



PERSISTENT KERNELS



PERSISTENT KERNELS



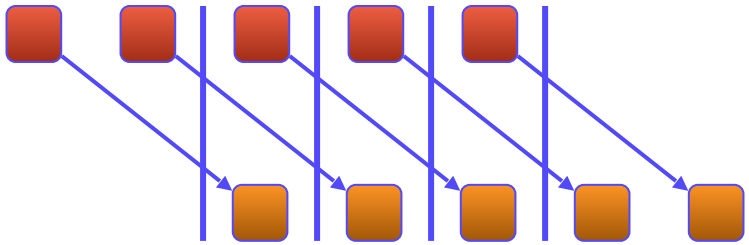
PERSISTENT KERNELS

```
for k in range(0, k_tiles):
  a = tl.load(a_ptrs)
  b = tl.load(b_ptrs)
  c = tl.dot(a, b, c)
  a_ptrs, b_ptrs = …
tl.store(c_ptrs, c)
```

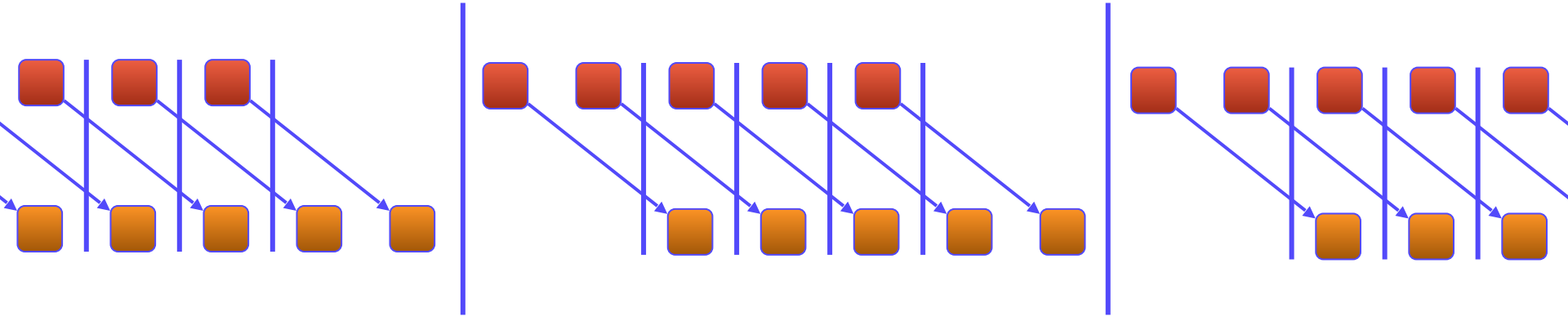
PERSISTENT KERNELS

```
for tile_id in range(start, num_tiles, NUM_SMS):
  a_ptrs, b_ptrs = …
  c = tl.zeros()
  for k in range(0, k_tiles):
    a = tl.load(a_ptrs)
    b = tl.load(b_ptrs)
    c = tl.dot(a, b, c)
    a_ptrs, b_ptrs = …
  tl.store(c_ptrs, c)
```

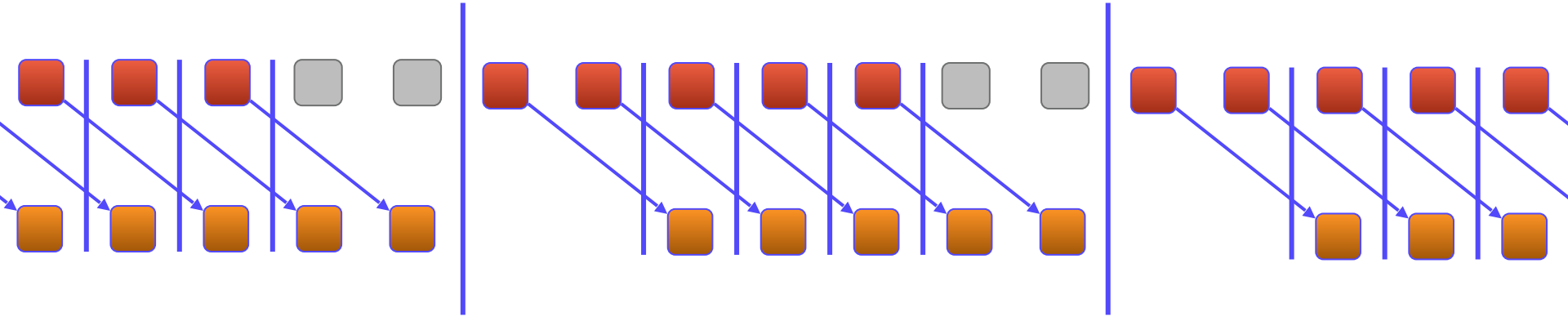
PUTTING THE TWO TOGETHER…



PUTTING THE TWO TOGETHER…



PUTTING THE TWO TOGETHER...



PUTTING THE TWO TOGETHER… EFFICIENTLY

```
for tile_id in range(start, num_tiles, NUM_SMS):
  a_ptrs, b_ptrs = …
  c = tl.zeros()
  for k in range(0, k_tiles):
    a = tl.load(a_ptrs)
    b = tl.load(b_ptrs)
    c = tl.dot(a, b, c)
    a_ptrs, b_ptrs = …
  tl.store(c_ptrs, c)
```

PUTTING THE TWO TOGETHER… EFFICIENTLY

```
for tile_id in range(start, num_tiles, NUM_SMS):
  a_ptrs, b_ptrs = …
  c = tl.zeros()
  for k in range(0, k_tiles):
    a = tl.load(a_ptrs)
    b = tl.load(b_ptrs)
    c = tl.dot(a, b, c)
    a_ptrs, b_ptrs = …
  tl.store(c_ptrs, c)
```

```
for tile_id in range(start, tiles_per_SM * k_tiles):
```

PUTTING THE TWO TOGETHER… EFFICIENTLY

```
for tile_id in range(start, num_tiles, NUM_SMS):
  a_ptrs, b_ptrs = …
  c = tl.zeros()
  for k in range(0, k_tiles):
    a = tl.load(a_ptrs)
    b = tl.load(b_ptrs)
    c = tl.dot(a, b, c)
    a_ptrs, b_ptrs = …
  tl.store(c_ptrs, c)
```

```
for tile_id in range(start, tiles_per_SM * k_tiles):
  tile_id, k = …
  if (k == 0):
    a_ptrs, b_ptrs = …
    c = tl.zeros()
```

PUTTING THE TWO TOGETHER… EFFICIENTLY

```
for tile_id in range(start, num_tiles, NUM_SMS):
  a_ptrs, b_ptrs = …
  c = tl.zeros()
  for k in range(0, k_tiles):
    a = tl.load(a_ptrs)
    b = tl.load(b_ptrs)
    c = tl.dot(a, b, c)
    a_ptrs, b_ptrs = …
  tl.store(c_ptrs, c)
```

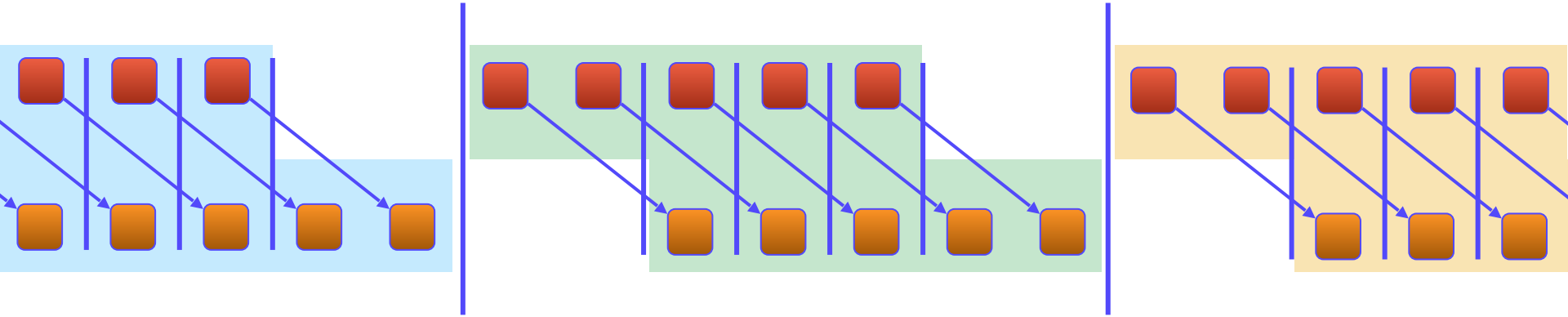
```
for tile_id in range(start, tiles_per_SM * k_tiles):
  tile_id, k = …
  if (k == 0):
    a_ptrs, b_ptrs = …
    c = tl.zeros()
  a_ptrs, b_ptrs = …
    c = tl.zeros()
  a = tl.load(a_ptrs)
  b = tl.load(b_ptrs)
  c = tl.dot(a, b, c)
  a_ptrs, b_ptrs = …
```

PUTTING THE TWO TOGETHER… EFFICIENTLY

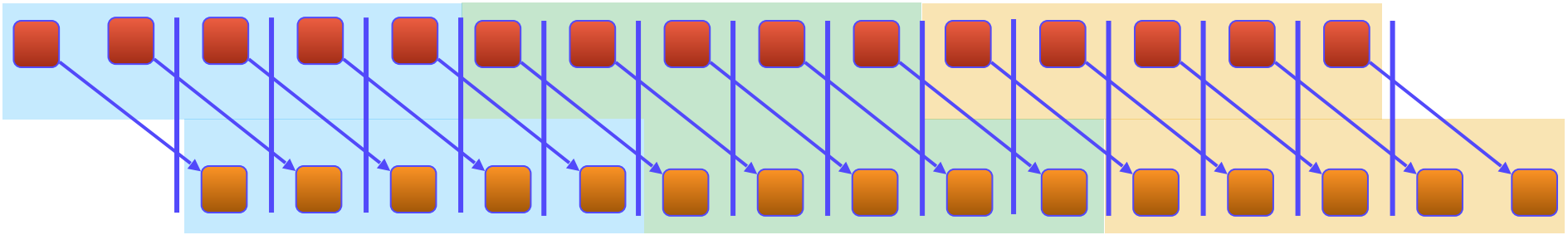
```
for tile_id in range(start, num_tiles, NUM_SMS):
  a_ptrs, b_ptrs = …
  c = tl.zeros()
  for k in range(0, k_tiles):
    a = tl.load(a_ptrs)
    b = tl.load(b_ptrs)
    c = tl.dot(a, b, c)
    a_ptrs, b_ptrs = …
  tl.store(c_ptrs, c)
```

```
for tile_id in range(start, tiles_per_SM * k_tiles):
  tile_id, k = …
  if (k == 0):
    a_ptrs, b_ptrs = …
    c = tl.zeros()
  a_ptrs, b_ptrs = …
    c = tl.zeros()
  a = tl.load(a_ptrs)
  b = tl.load(b_ptrs)
  c = tl.dot(a, b, c)
  a_ptrs, b_ptrs = …
if (k == k_tiles-1):
    tl.store(c_ptrs, x)
```

PUTTING THE TWO TOGETHER… EFFICIENTLY



PUTTING THE TWO TOGETHER… EFFICIENTLY



We do it not because it is easy…

We do it not because it is easy…

…But because we thought it would be easy.

Challenges

Hand-fused loops in matmul kernel, multitude of functional and performance problems

- Matmul optimized path not ready for branching in the main loop

Challenges

Hand-fused loops in matmul kernel, multitude of functional and performance problems

- Matmul optimized path not ready for branching in the main loop
    - WGMMA pipelining 💀

Challenges

Hand-fused loops in matmul kernel, multitude of functional and performance problems

- Matmul optimized path not ready for branching in the main loop
    - WGMMA pipelining 💀
    - Layout conversion optimizations 😵

Challenges

Hand-fused loops in matmul kernel, multitude of functional and performance problems

- Matmul optimized path not ready for branching in the main loop
    - WGMMA pipelining 💀
    - Layout conversion optimizations 😵
    - Axis Analysis 🪦

Challenges

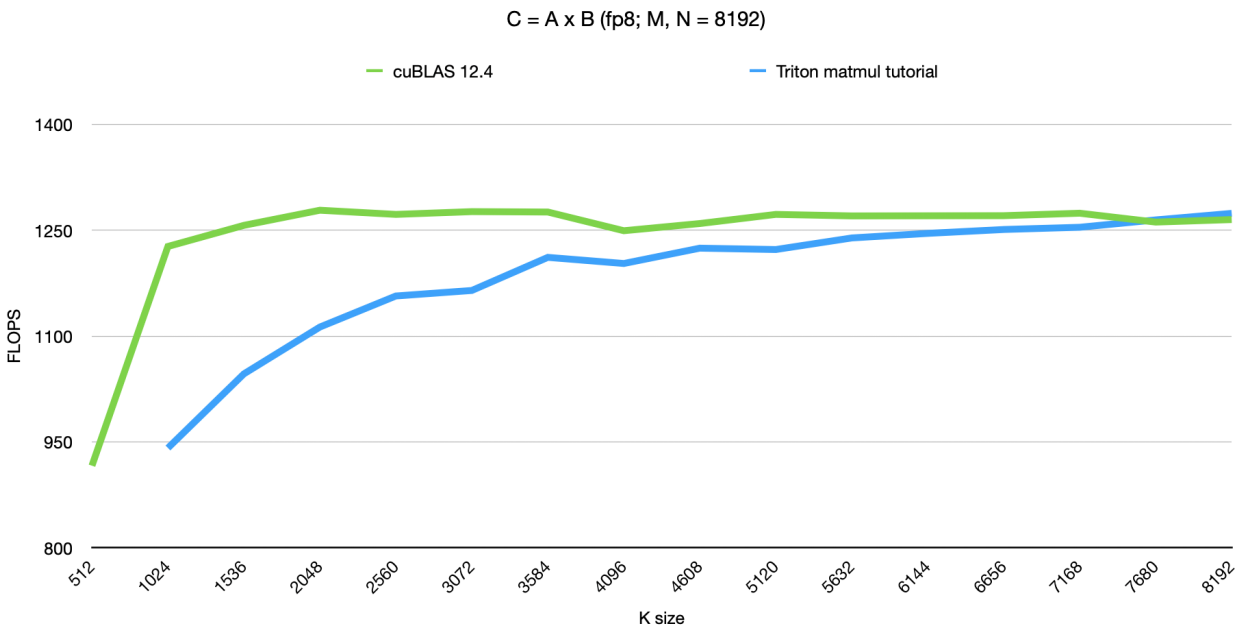
Hand-fused loops in matmul kernel, multitude of functional and performance problems

- Matmul optimized path not ready for branching in the main loop
    - WGMMA pipelining 💀
    - Layout conversion optimizations 😵
    - Axis Analysis 🪦
- IfOp placement in the loop - major performance problem
    - Placed in the middle of the loop
    - Instruction scheduling off the window
    - Introduced CoarseSchedule, allowing for better control over ops placement

- **Final ingredient: TMA**
  - Less arithmetic in main loop because of HW-handled OOB

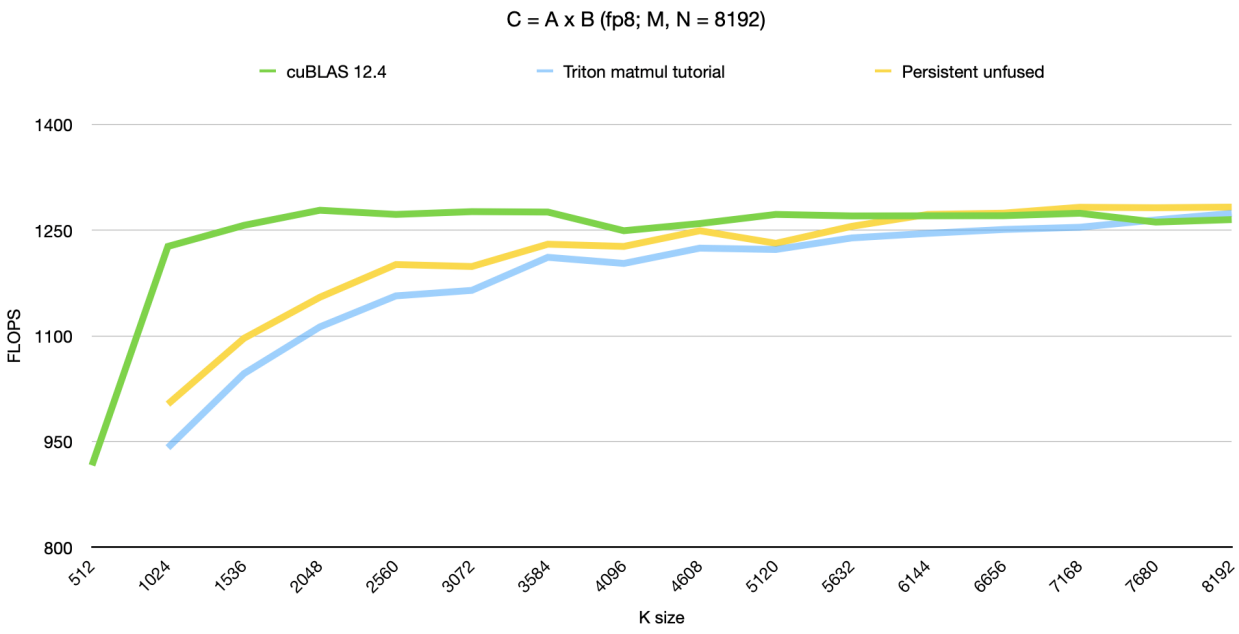
- **Final ingredient: TMA**
  - Less arithmetic in main loop because of HW-handled OOB
  - Async stores to global memory at the end of the tile

PUTTING THE TWO TOGETHER... EFFICIENTLY



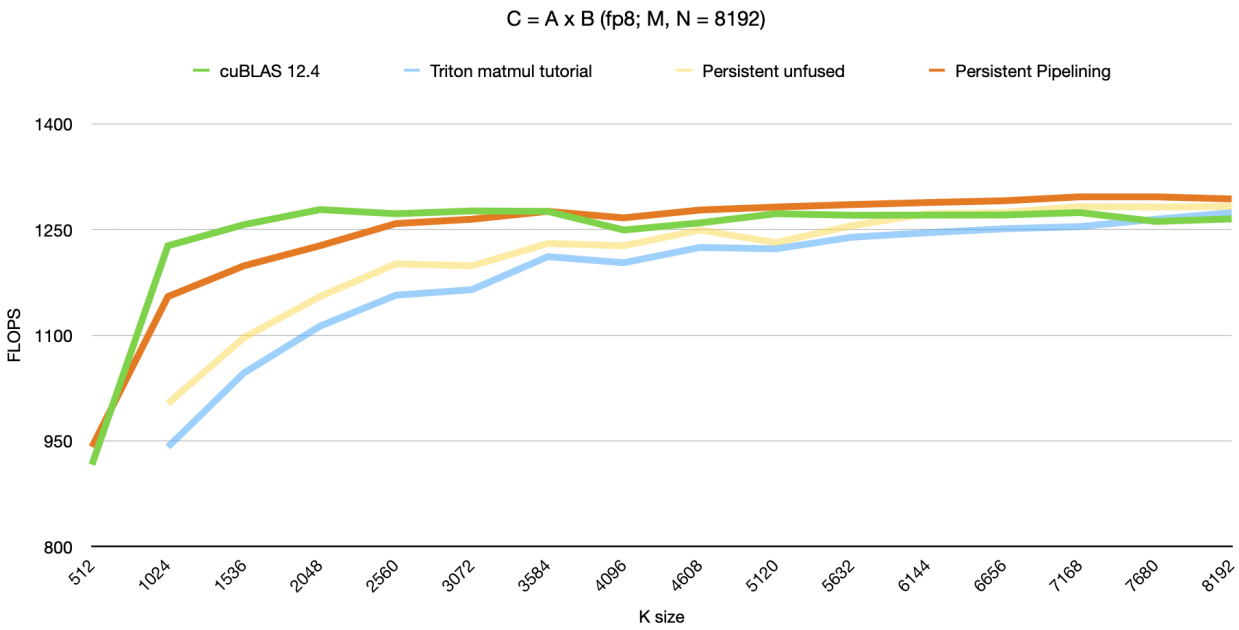
* random A, B; 10000 warm-up, 1000 measurements avg; no fixed clocks; minimal startup overhead; measured with:
branch: `pawel/perf_data_collection`
cmd: `python 09-persistent-matmul.py --prec=fp8 --K_range 512 8192; proton-viewer -m "avg_time/us" matmul.hatchet`

PUTTING THE TWO TOGETHER… EFFICIENTLY



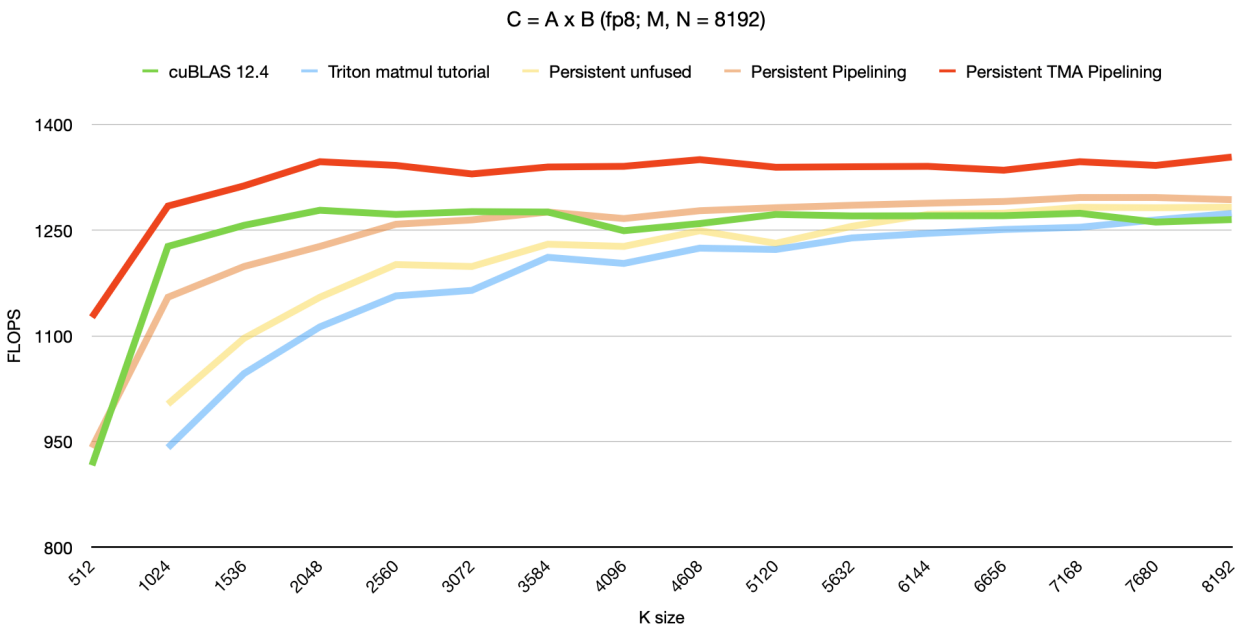
* random A, B; 10000 warm-up, 1000 measurements avg; no fixed clocks; minimal startup overhead; measured with:
branch: `pawel/perf_data_collection`
cmd: `python 09-persistent-matmul.py --prec=fp8 --K_range 512 8192; proton-viewer -m "avg_time/us" matmul.hatchet`

PUTTING THE TWO TOGETHER… EFFICIENTLY



* random A, B; 10000 warm-up, 1000 measurements avg; no fixed clocks; minimal startup overhead; measured with:
branch: `pawel/perf_data_collection`
cmd: `python 09-persistent-matmul.py --prec=fp8 --K_range 512 8192; proton-viewer -m "avg_time/us" matmul.hatchet`

PUTTING THE TWO TOGETHER… EFFICIENTLY



* random A, B; 10000 warm-up, 1000 measurements avg; no fixed clocks; minimal startup overhead; measured with:
branch: `pawel/perf_data_collection`
cmd: `python 09-persistent-matmul.py --prec=fp8 --K_range 512 8192; proton-viewer -m "avg_time/us" matmul.hatchet`

Further Work

- TMA descriptor passing via grid constant by Elliot Gorokhovsky (Meta)
- TMA descriptor on-device creation and update by Peter Bell (OpenAI)
- Pipeliner refactoring and support for multiple schedules led by Manman Ren (Meta)

- Further further work
    a. Layering schedules
    b. Epilogue tiling
    c. Automatic loop fusion
    d. …

Thank you!