ACRoBat: Optimizing Auto-Batching of Dynamic Deep Learning at Compile Time

Pratik Fegade[1]*,
Tianqi Chen[1,2], Phillip B. Gibbons[1], Todd C. Mowry[1]

[1]Carnegie Mellon University [2]OctoAI

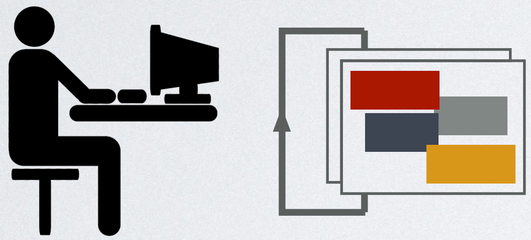
* Now at Google

Carnegie Mellon University
Computer Science Department

catalyst

ACRoBat: Efficient Auto-Batching for Dynamic Control Flow

Unbatched dynamic model impl.

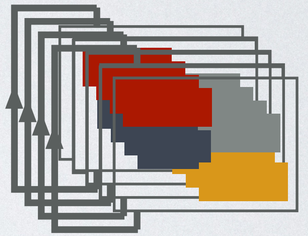


Compilation

Input batch

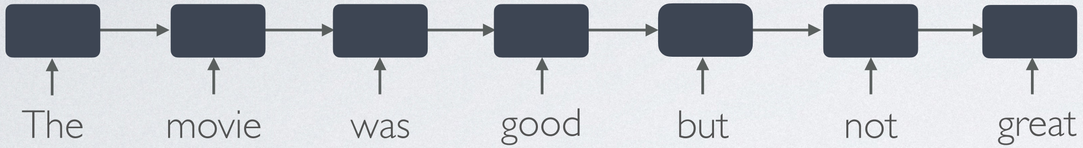
Runtime

Batched model execution

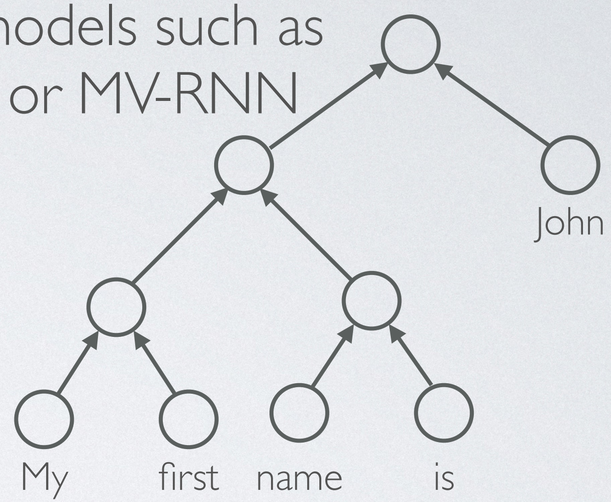


Wide Variety of Control Flow in DL Computations

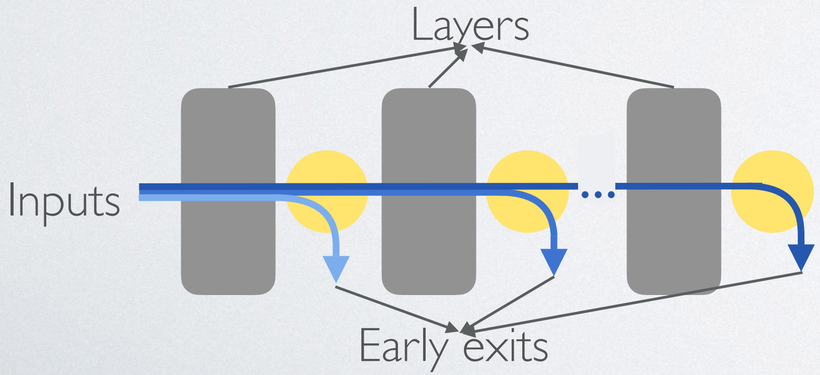
Recurrent neural networks (RNNs)



Recursive models such as TreeLSTM or MV-RNN

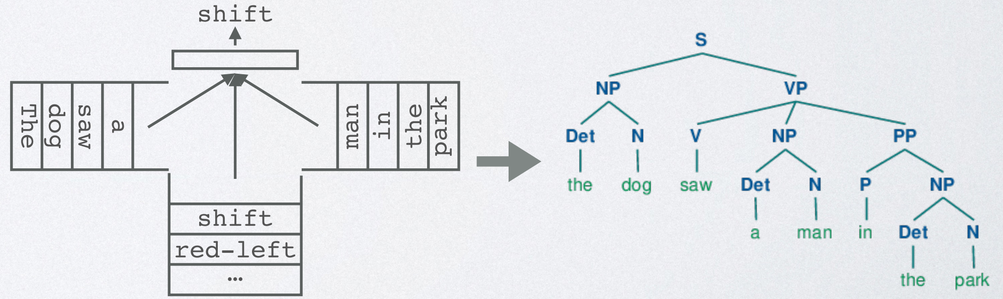


Early exit models have tensor dependent control flow



StackLSTM: Complex control flow

The dog saw a man in the park



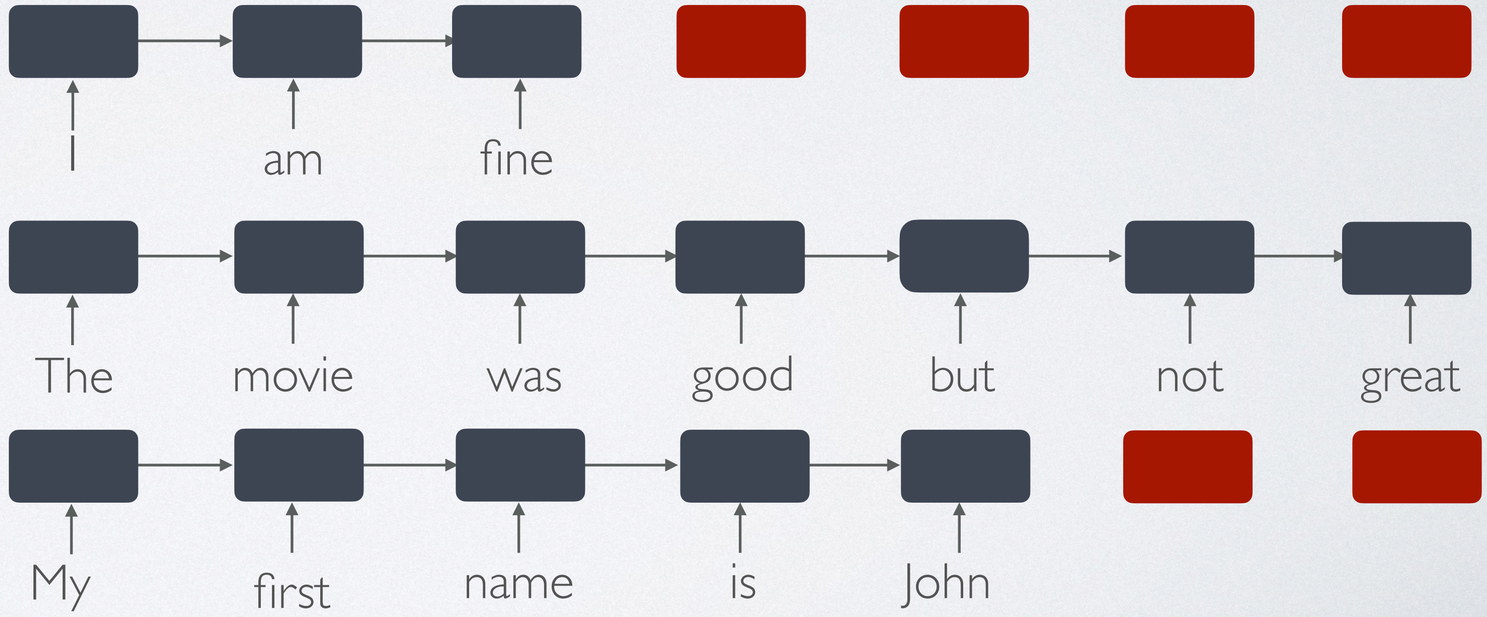
Batching Is Difficult for Models With Dynamic Control Flow

- Recurrent neural networks
  - Variable sentence length → non-uniformity in number of iterations

```
state = initial_state
for word in sentence:
    state = RNNCell(state, word)
```

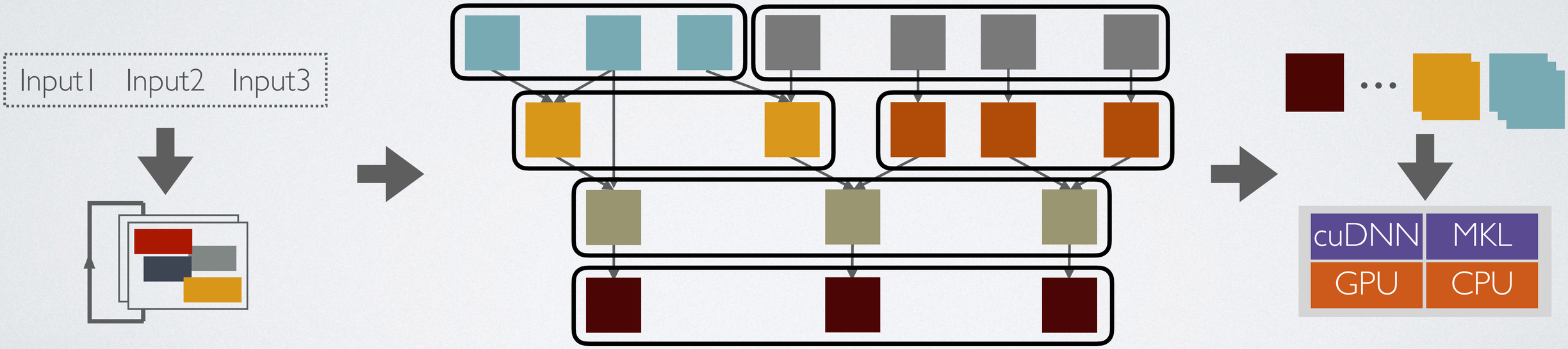
Iteration over words: surrounding control flow

Simple padding wasteful!

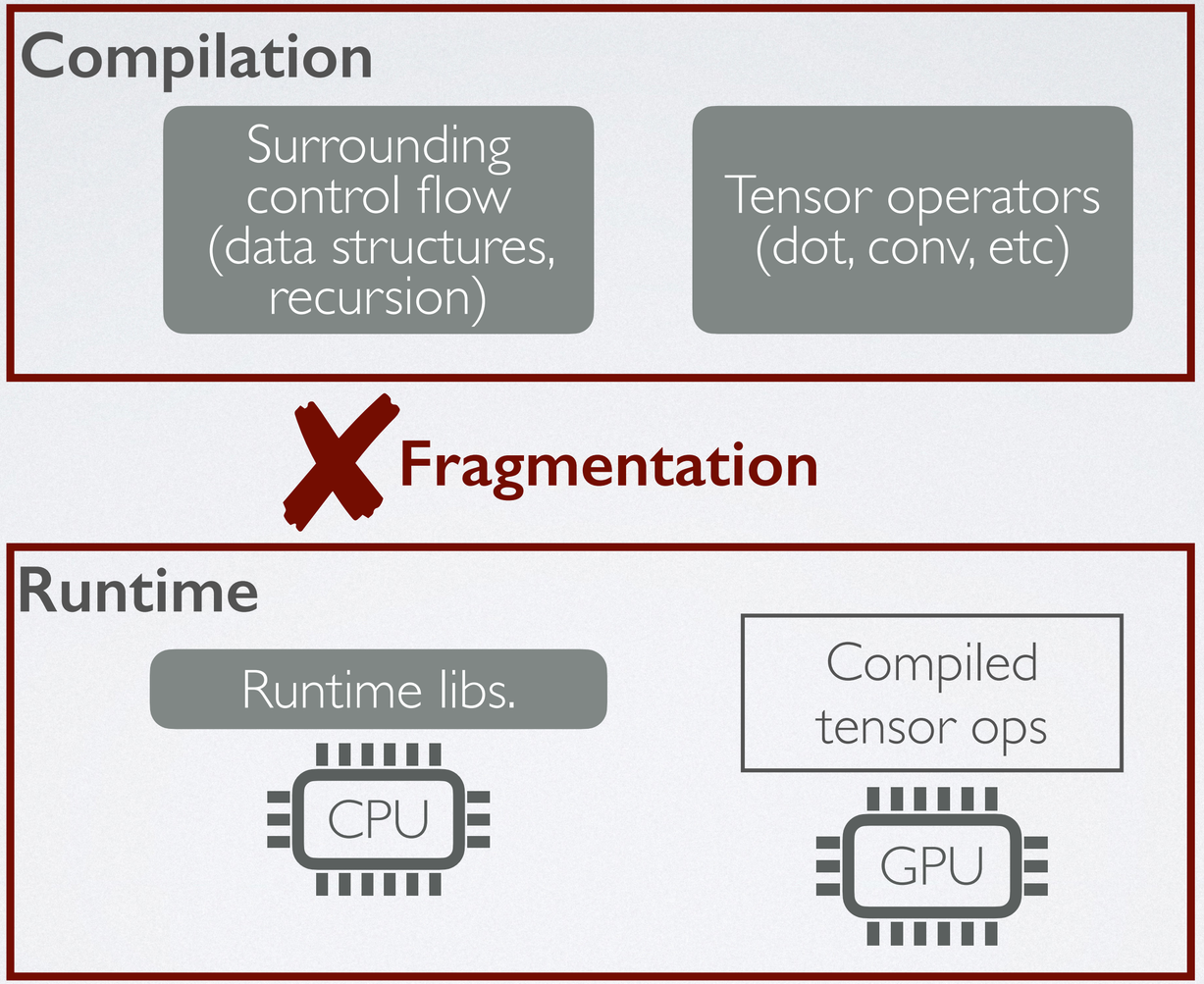


Dynamic Batching, a Prior, Fully Runtime Approach

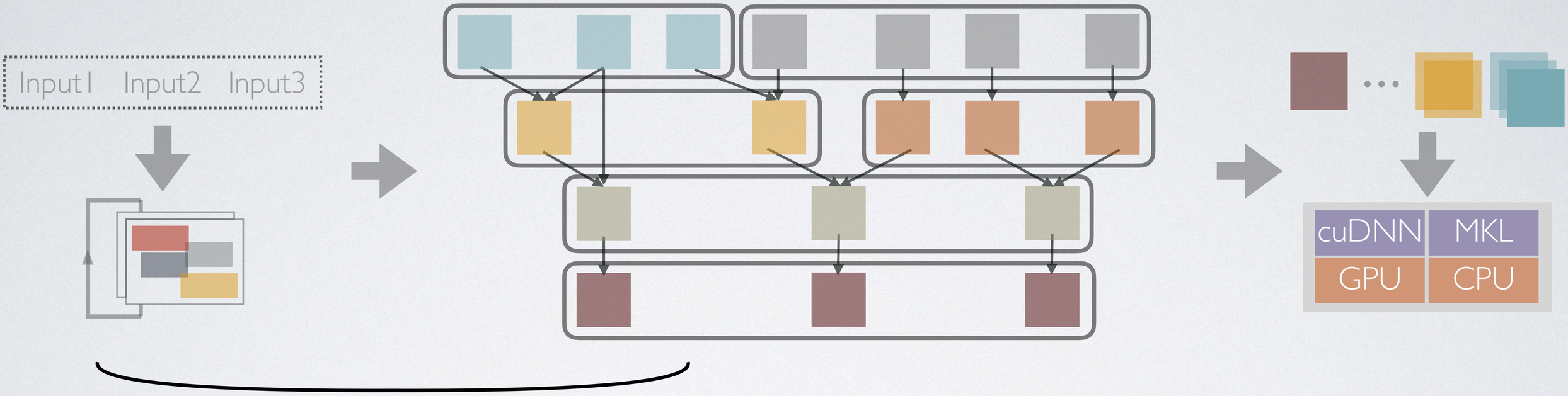
- Construct dataflow graphs (DFGs) for each input in mini-batch
- Traverse graphs to determine which operators can be batched
- Invoke batched kernels



Past Work: Compiler-Runtime Fragmentation→Suboptimal Performance

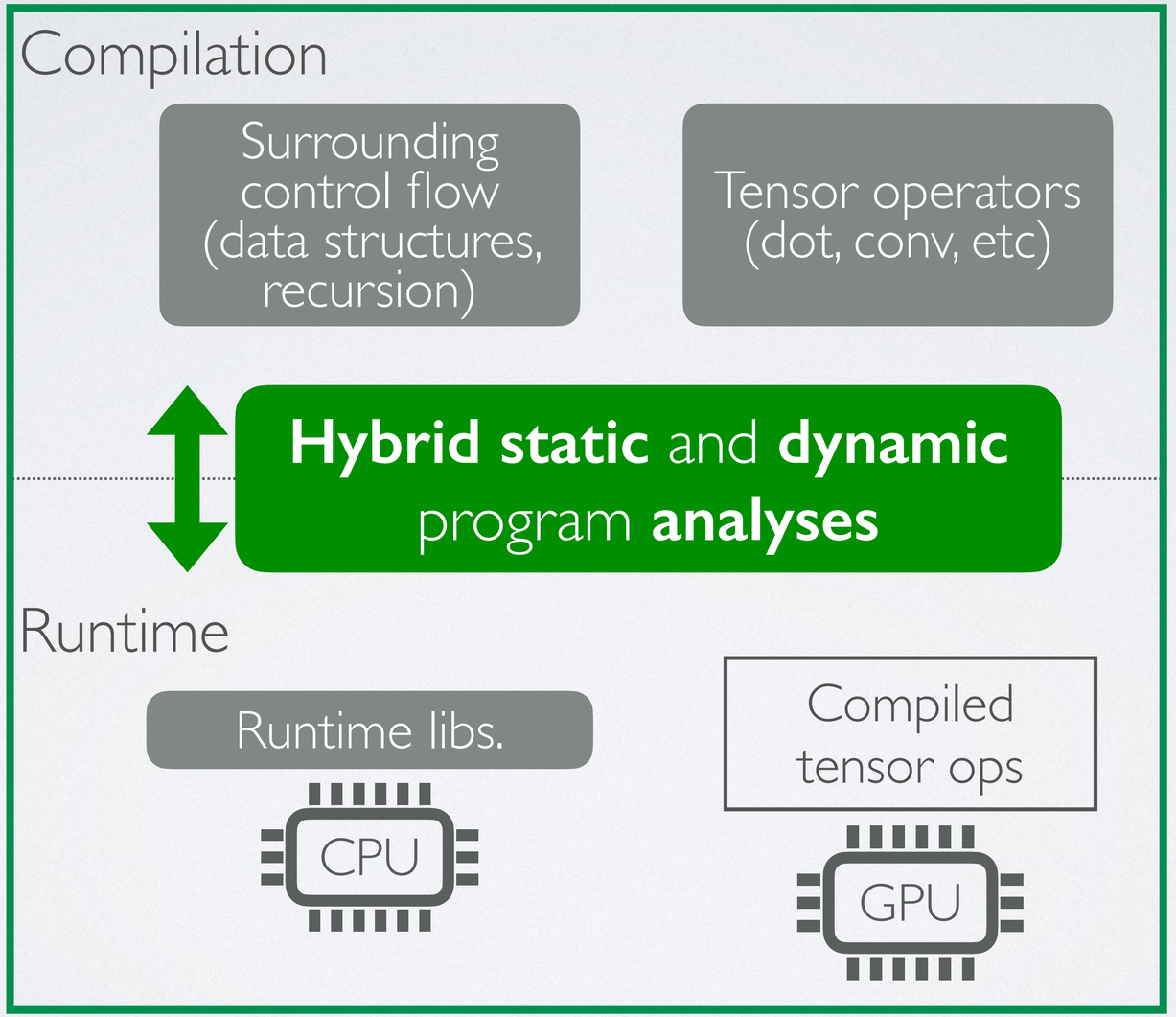


Fully Dynamic Auto-Batching→Execution Overheads

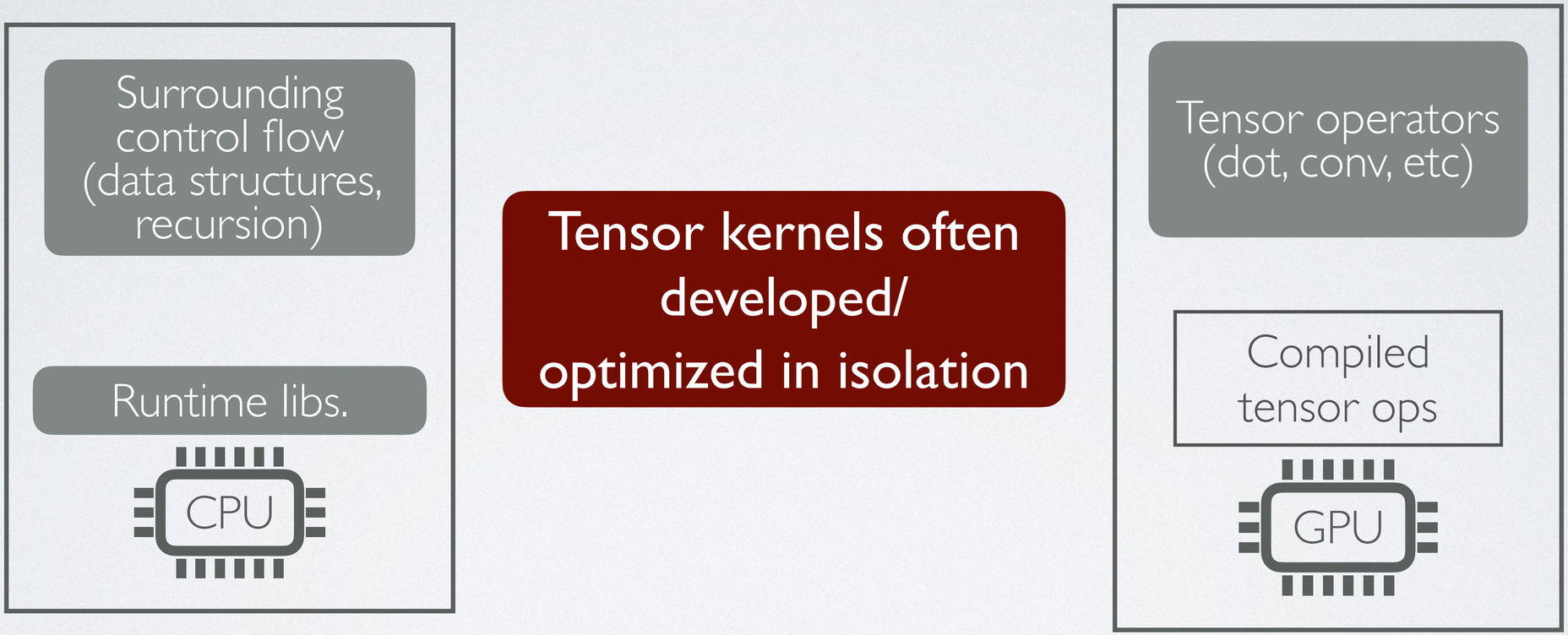


- DFG construction
- DFG scheduling

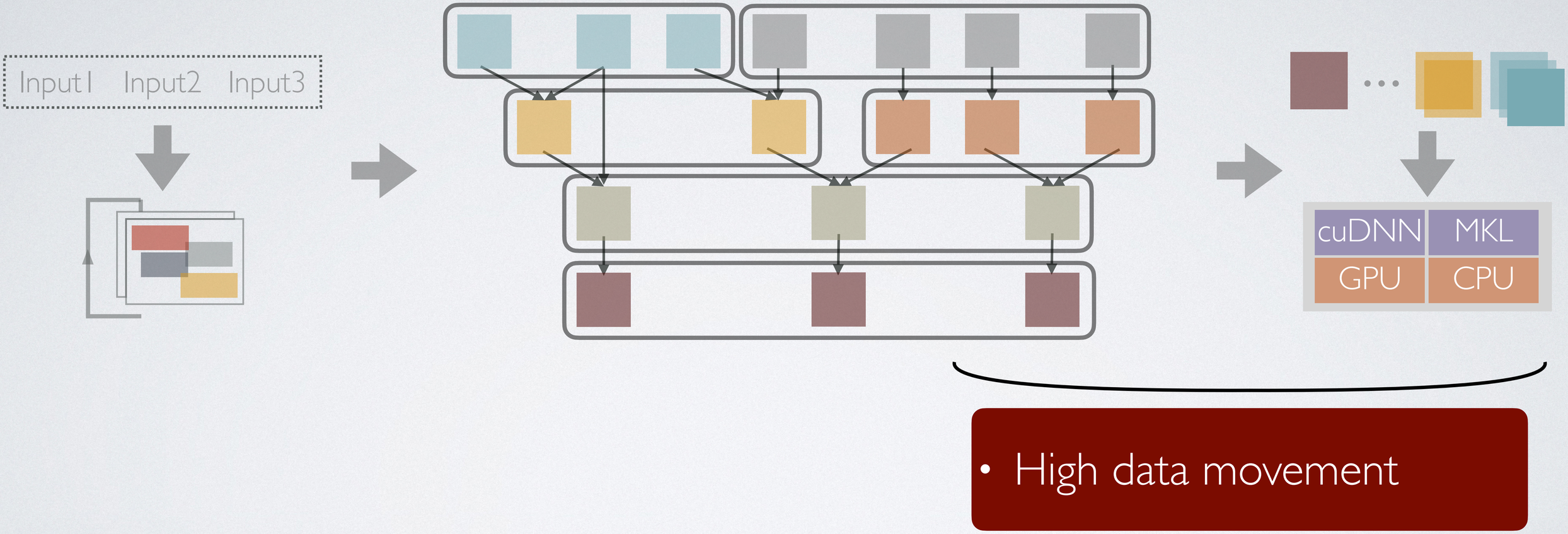
ACRoBat Uses Hybrid Static and Dynamic Program Analyses



Past Work: Compiler-Runtime Fragmentation→Suboptimal Performance

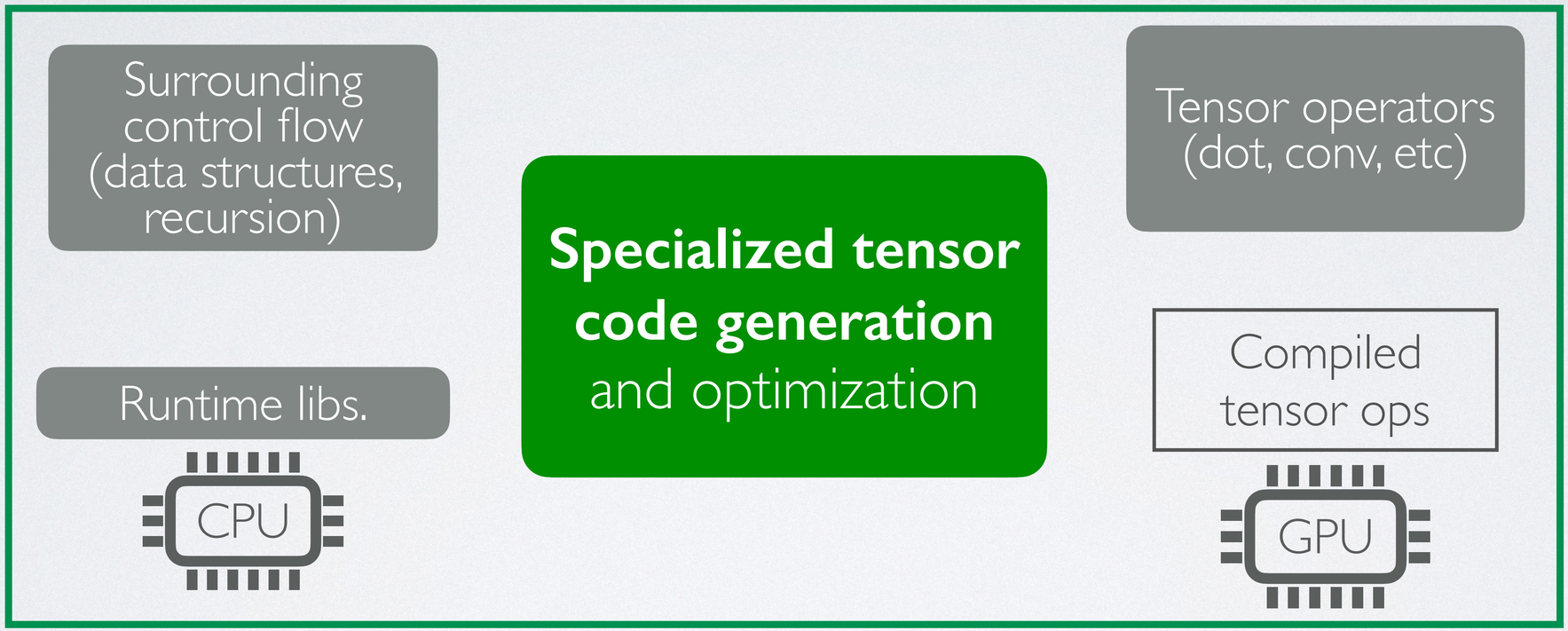


Fully Dynamic Auto-Batching→Execution Overheads



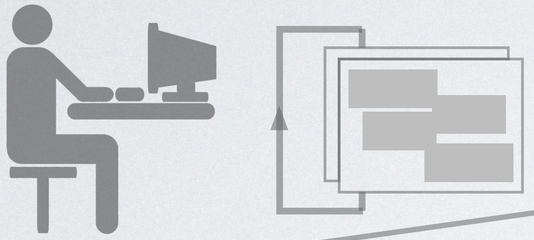
- High data movement

ACRoBat Uses Specialized Tensor Code Generation



ACRoBat: Efficient Auto-Batching for Dynamic Control Flow

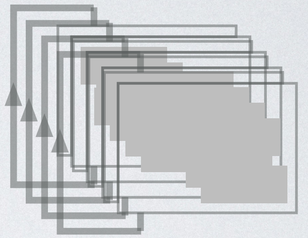
Unbatched model implementation



Input batch

Compilation → Runtime

Batched model execution



Dynamic batching

Past fully dynamic auto-batching technique

+

Hybrid static & dynamic analyses

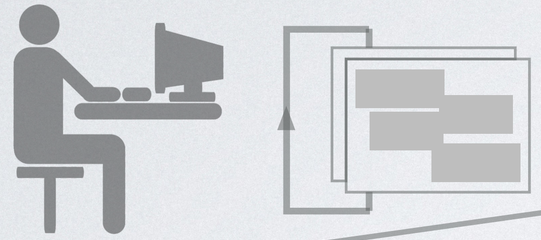
- **Inline scheduling**
- Grain size coarsening…

Specialized tensor codegen.

- **Memory gather fusion**
- Profile info. for prioritizing auto-scheduling…

ACRoBat: Efficient Auto-Batching for Dynamic Control Flow

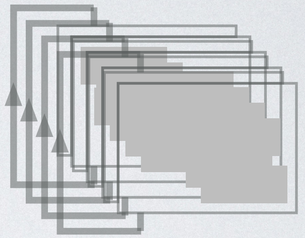
Unbatched model implementation



Input batch

Compilation → Runtime

Batched model execution



Dynamic batching

Past fully dynamic auto-batching technique

+

Hybrid static & dynamic analyses

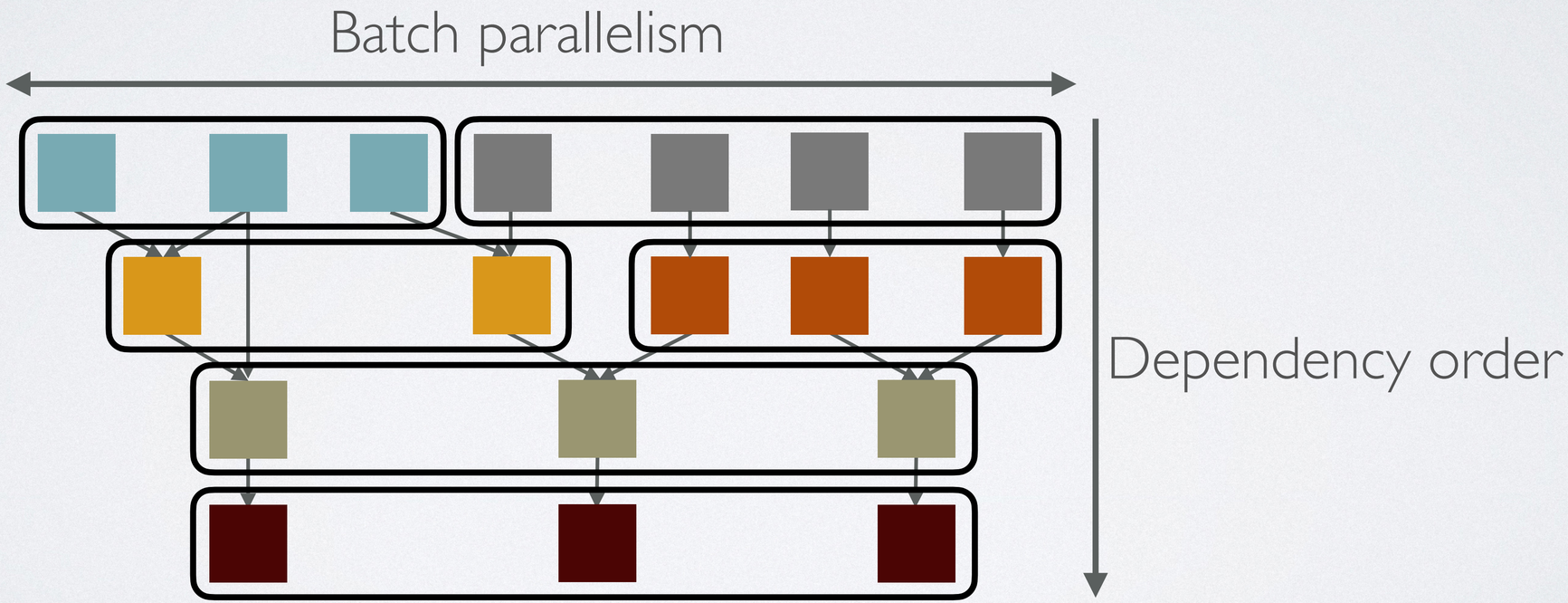
- **Inline scheduling**
- Grain size coarsening…

Specialized tensor codegen.

- **Memory gather fusion**
- Profile info. for prioritizing auto-scheduling…

Inline Scheduling: Goals of Scheduling/Batching

- Correctness: Respect operator dependences
- Performance: Identify opportunities for parallelism



Inline Scheduling: Unifying DFG Construction and Scheduling

- Correctness: Respect operator dependences

Model computation invokes tensor ops in dependency order → DFGs are already constructed in dependency order

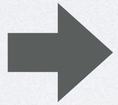
- Performance: Identify opportunities for parallelism

Parallelism often expressed via recursion or the list map function → Knowledge of parallelism often available statically

We can perform scheduling during graph construction! → Up to 2.5X better perf. for some model configs

Inline Scheduling: Example

```
def treeFC(n):
  if isleaf(n):
    return Emb[words[n]]
  else:
    lh = treeFC(n.left)
    rh = treeFC(n.right)
    return W * (lh + rh)
```



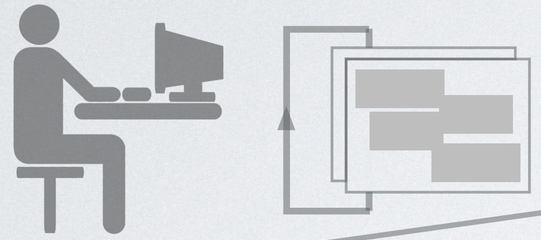
```
def treeFC(n, n_idx):
  if isleaf(n):
    // Node Idx: n_idx
    return Emb[words[n]]
  else:
    lh = treeFC(n.left, n_idx + 1)
    rh = treeFC(n.right, n_idx + 1)
    // Node Idx: n_idx
    return W * (lh + rh)
```

Static knowledge of parallelism

Nodes with the same node_idx are independent and can be batched!

ACRoBat: Efficient Auto-Batching for Dynamic Control Flow

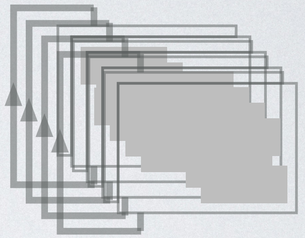
Unbatched model implementation



Input batch

Compilation → Runtime

Batched model execution



Dynamic batching

Past fully dynamic auto-batching technique

+

Hybrid static & dynamic analyses

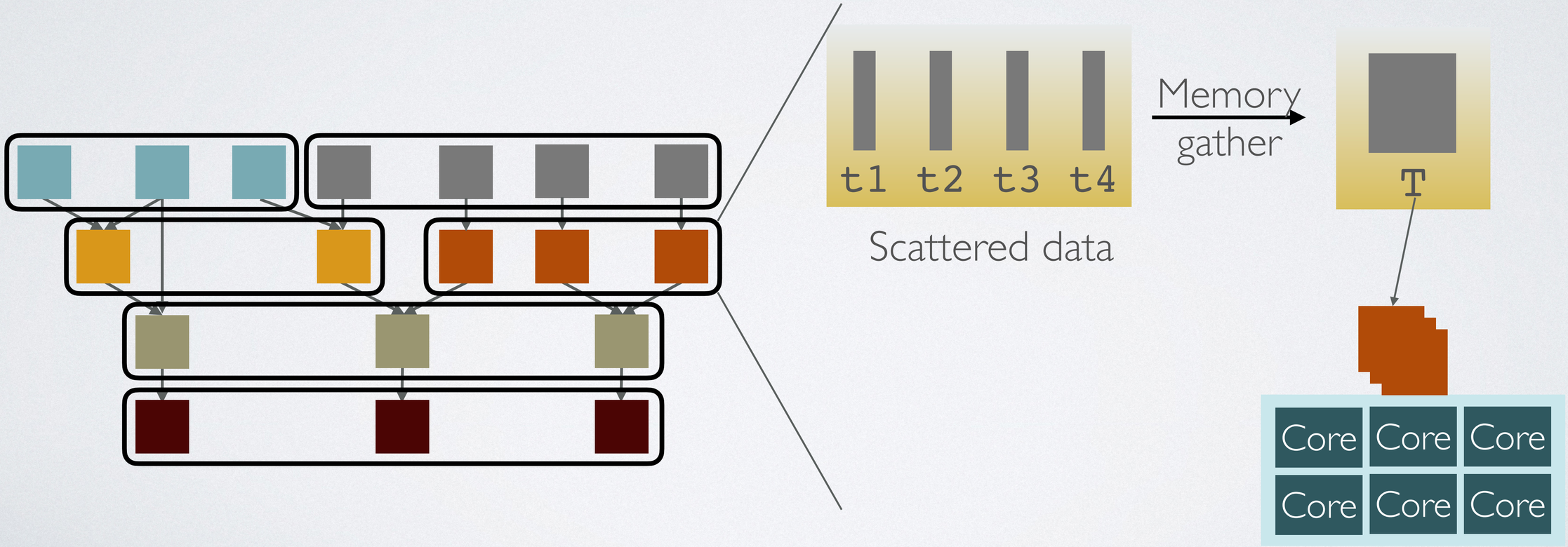
- **Inline scheduling**
- Grain size coarsening…

Specialized tensor codegen.

- **Memory gather fusion**
- Profile info. for prioritizing auto-scheduling…

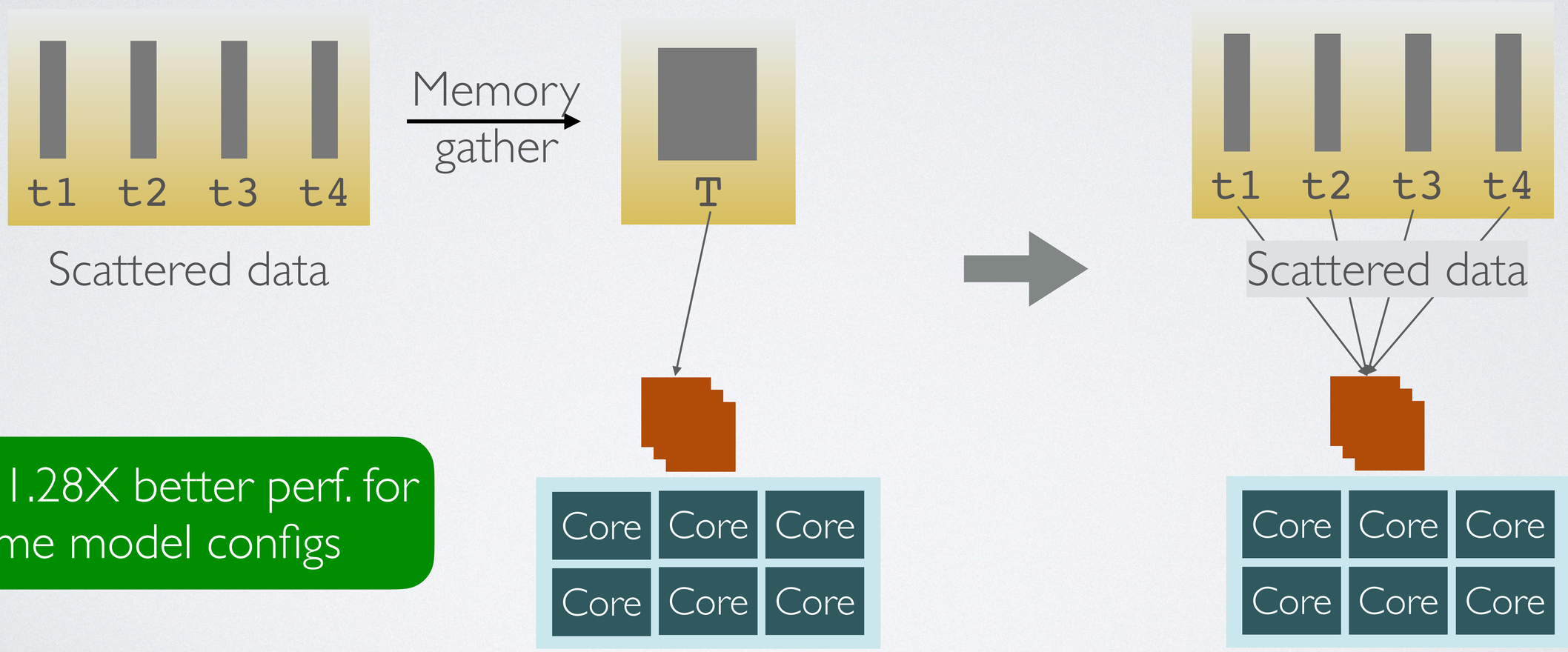
Memory Gather To Ensure Input Contiguity

- Dynamic scheduling → input tensors to batched kernels scattered in memory
  - Perform expensive memory gather before kernel call



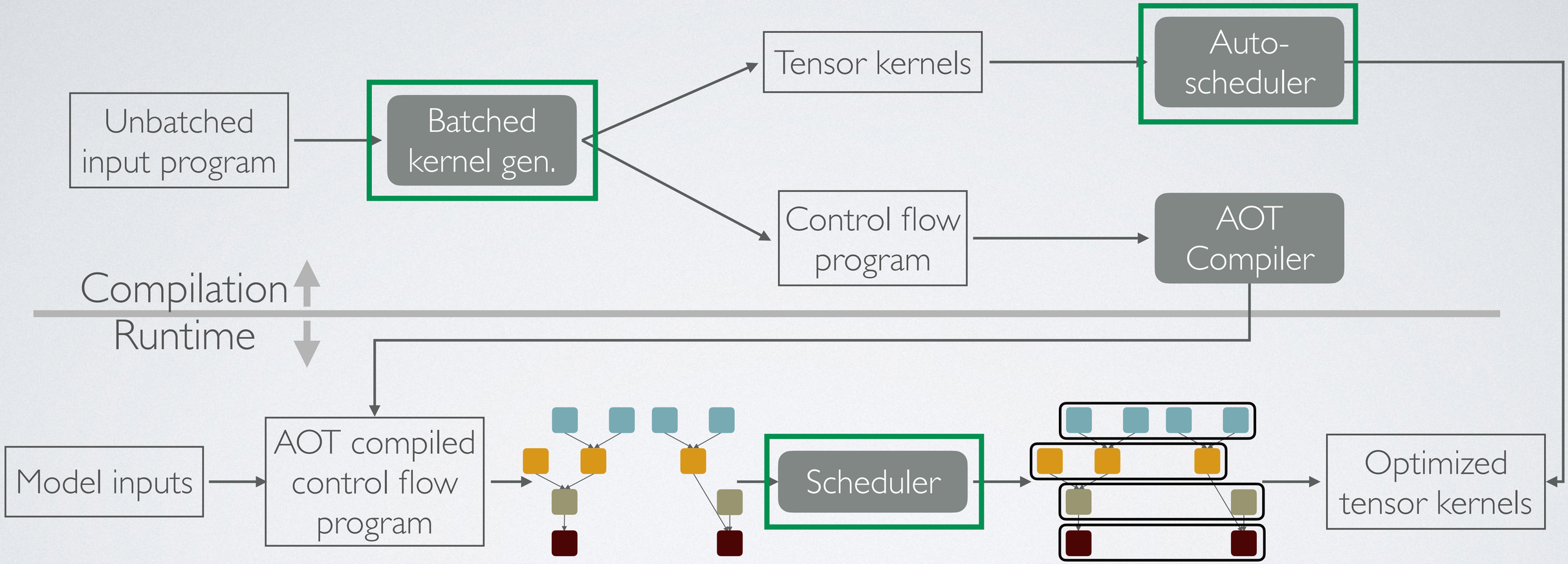
Avoiding Data Movement: Fuse Memory Gather Op

- Generate kernels to directly operate on scattered data

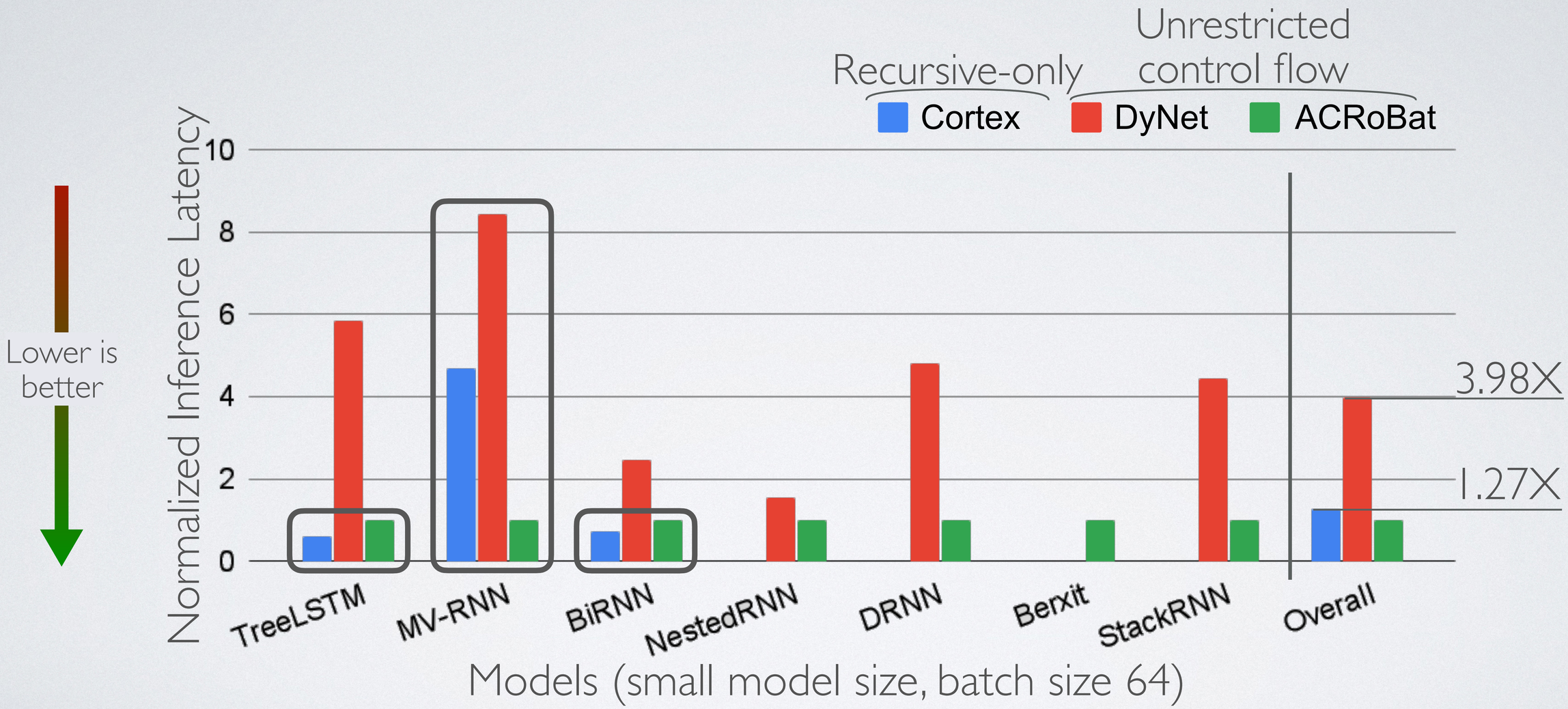


Up to 1.28X better perf. for some model configs

ACRoBat: Compilation and Runtime Workflow



Evaluation: Inference Latencies on Nvidia RTX 3070 GPU



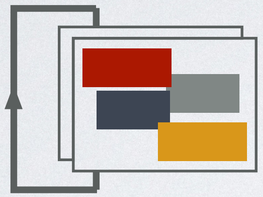
ACRoBat Has Low Execution Overheads

Includes memory gather time

Model	Framework	Scheduling time (ms)	Kernels called	Memory mgmt. time (ms)	GPU kernel time
TreeLSTM (small, BS 64)	DyNet	18.5	1653	3.1	6.1
	ACRoBat	1.9 ↓	183 ↓	0.1 ↓	4.0 ↓
BiRNN (large, BS 64)	DyNet	7.8	580	2.3	6.6
	ACRoBat	1.4 ↓	380 ↓	0.2 ↓	11.2 ↑

ACRoBat: Efficient Auto-Batching for Dynamic Control Flow

Unbatched dynamic model impl.

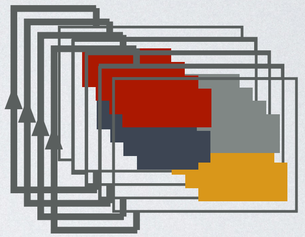


Compilation

Input batch

Runtime

Batched model execution



Hybrid static & dynamic analyses

Specialized tensor codegen.

Overall ~4X faster!