15-884: Machine Learning Systems

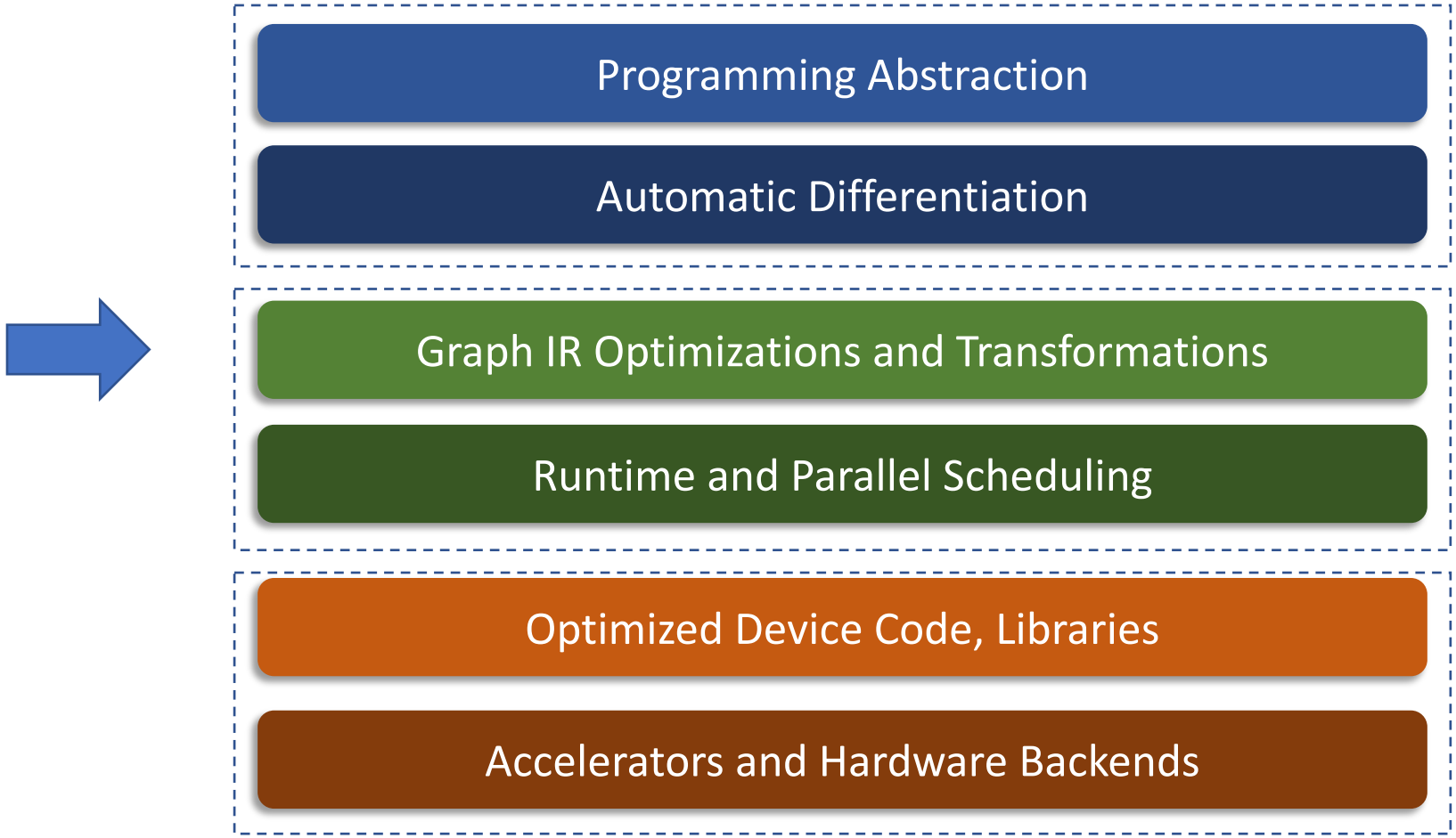
High-level Optimizations

Instructor: Tianqi Chen

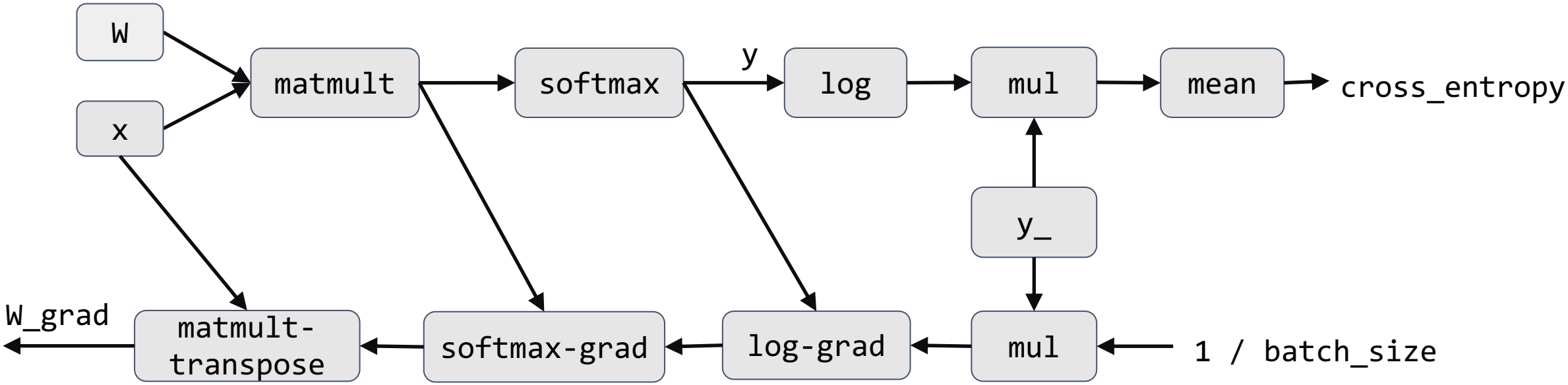
Carnegie Mellon University
School of Computer Science

catalyst

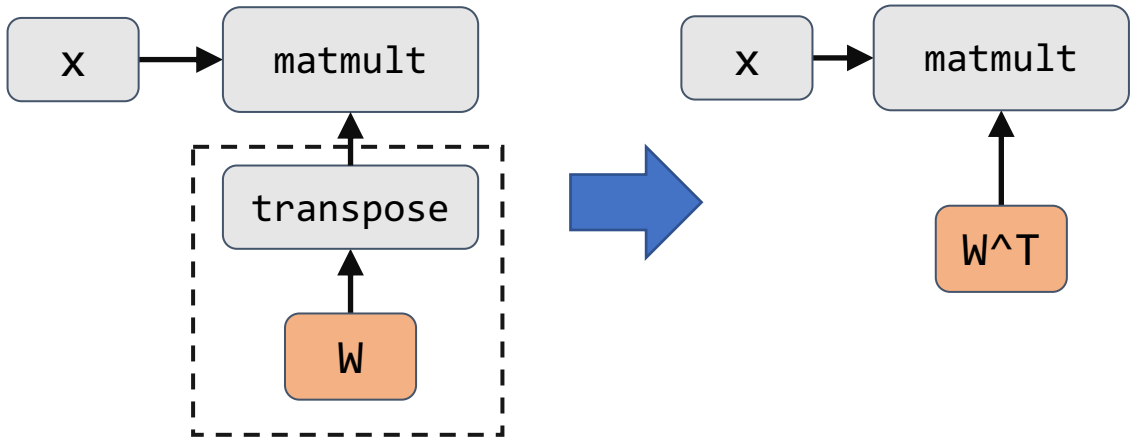
A Typical Deep Learning System Stack



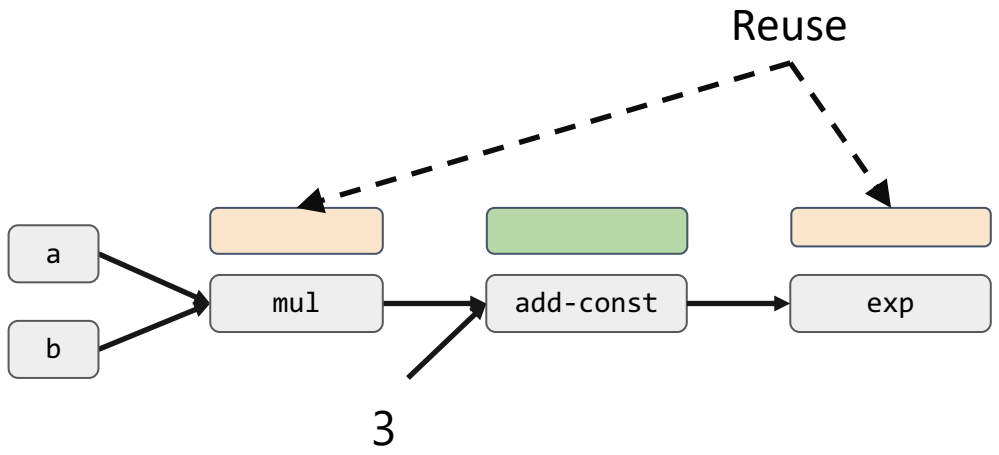
Computational Graph



Optimizations as Graph Rewrite and Annotation



Constant Folding



Memory Planning

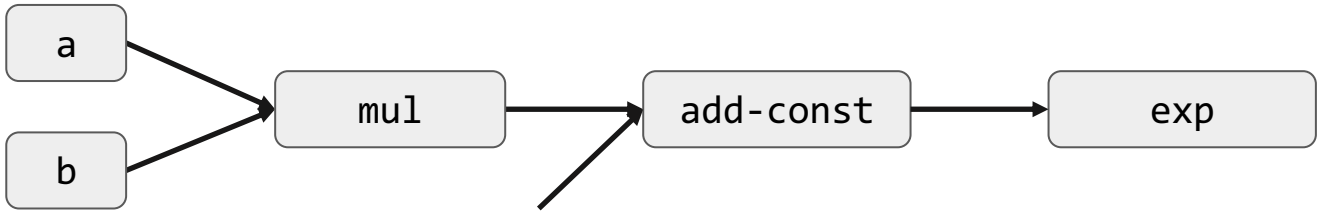
Discussion

What are possible optimizations we can do on a computational graph?

Memory Reuse Planning

Executing a Computation Graph

Computational Graph for `exp(a * b + 3)`

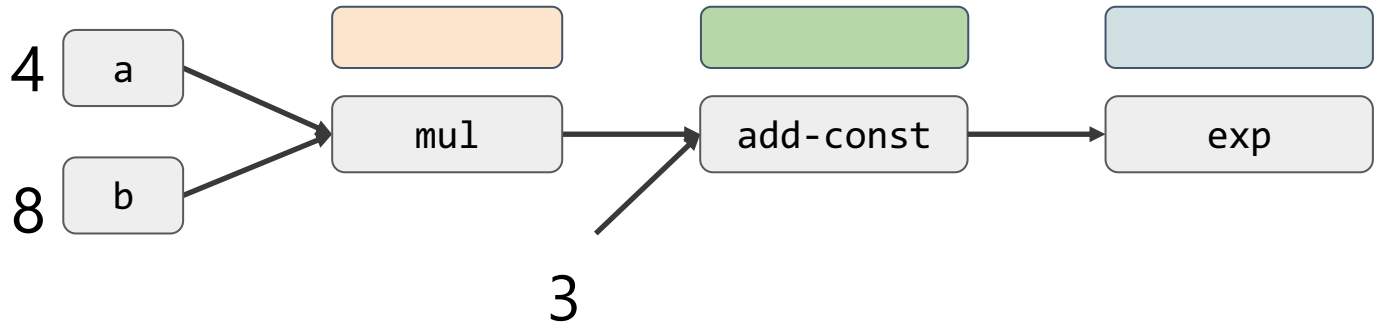


Executing a Computation Graph

- **Allocate** temp memory for intermediate computation

Same color represent same piece of memory

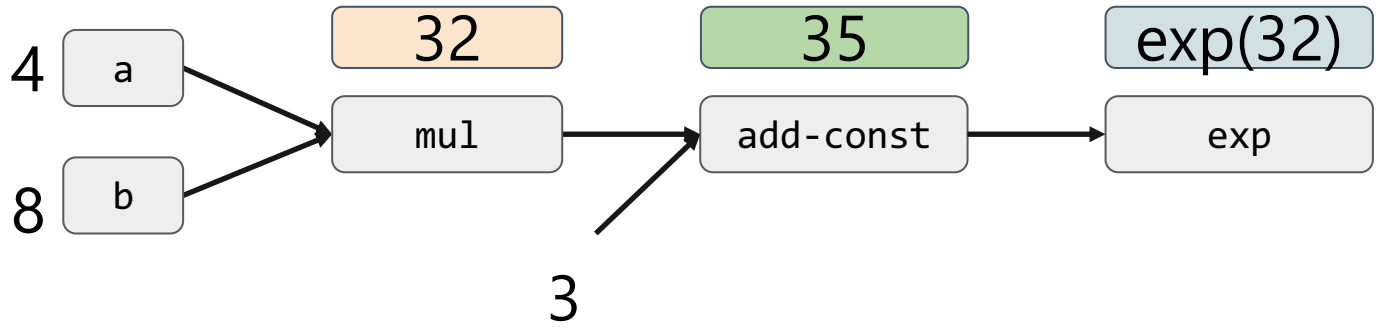
Computational Graph for `exp(a * b + 3)`



Executing a Computation Graph

- **Allocate** temp memory for intermediate computation
- **Traverse and execute** the graph in topo order.

Computational Graph for `exp(a * b + 3)`

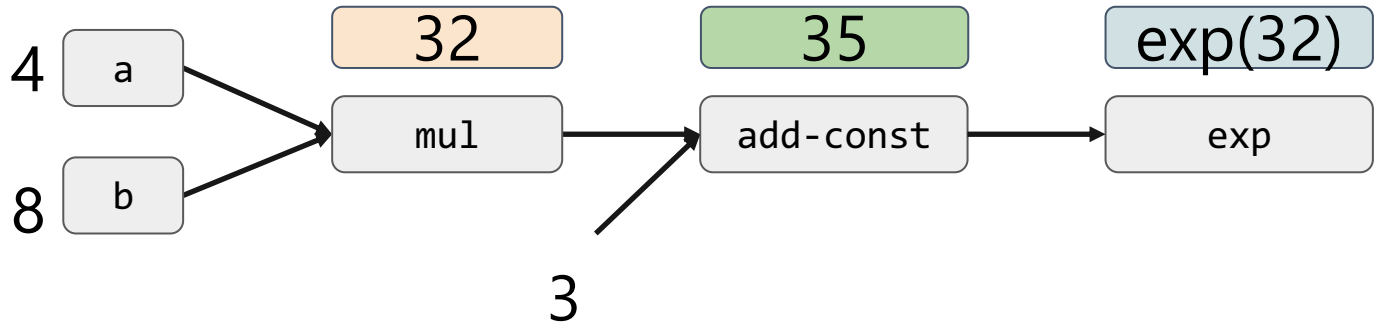


Executing a Computation Graph

- **Allocate** temp memory for intermediate computation
- **Traverse and execute** the graph by topo order.

No memory reuse

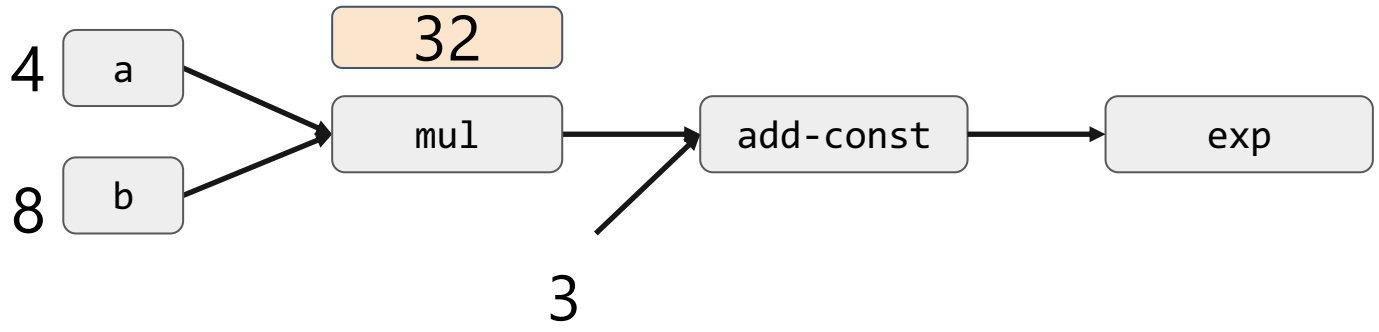
Computational Graph for `exp(a * b + 3)`



Dynamic Memory Allocation

- **Allocate** when needed
- **Recycle** when a memory is not needed.
- Useful for both declarative and imperative executions

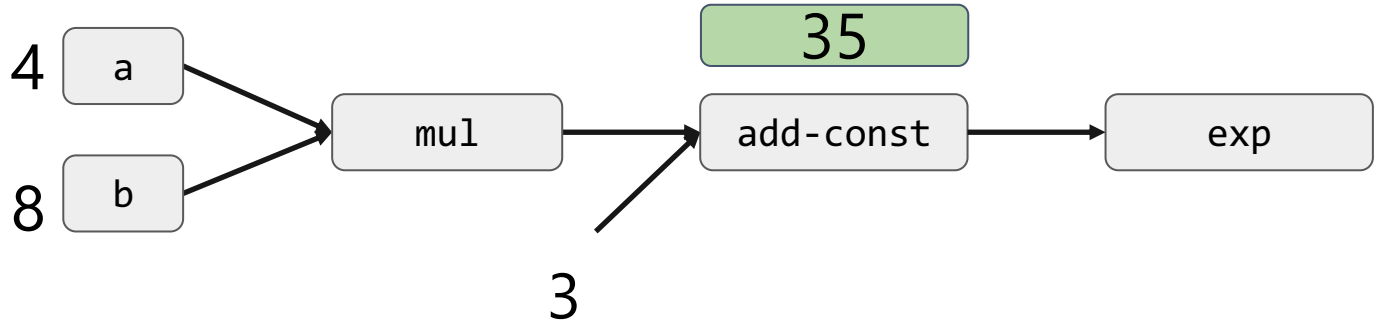
Memory Pool



Dynamic Memory Allocation

- **Allocate** when needed
- **Recycle** when a memory is not needed.
- Useful for both declarative and imperative executions

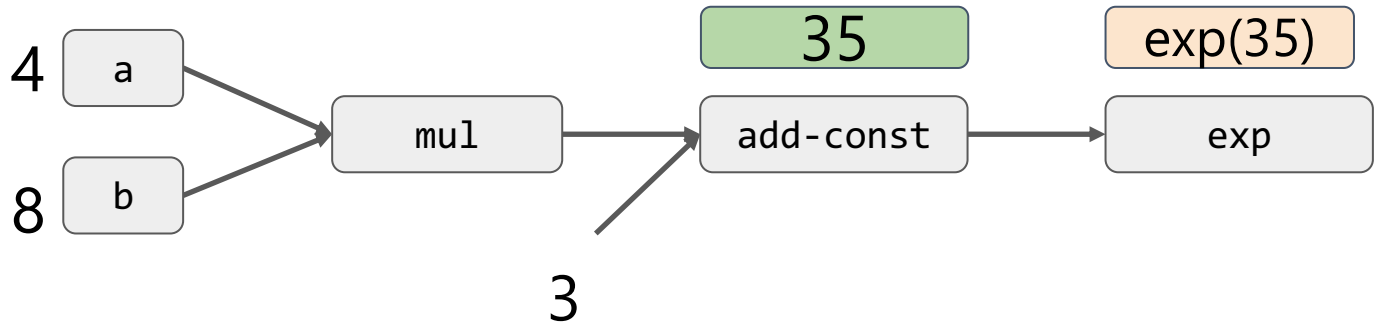
Memory Pool



Dynamic Memory Allocation

- **Allocate** when needed
- **Recycle** when a memory is not needed.
- Useful for both declarative and imperative executions

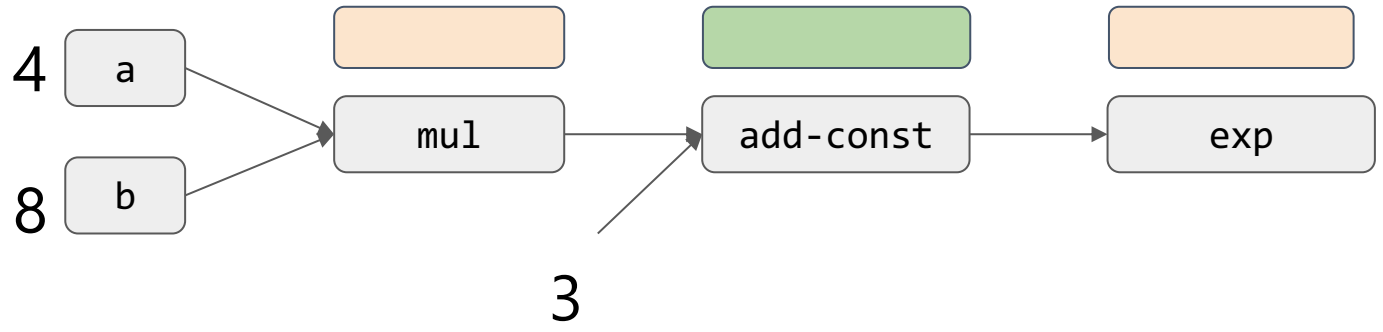
Memory Pool



Static Memory Planning

- Plan for reuse **ahead of time**
- Similar to register allocation algorithm in compilers

Same color represent same piece of memory



Common Patterns of Memory Planning

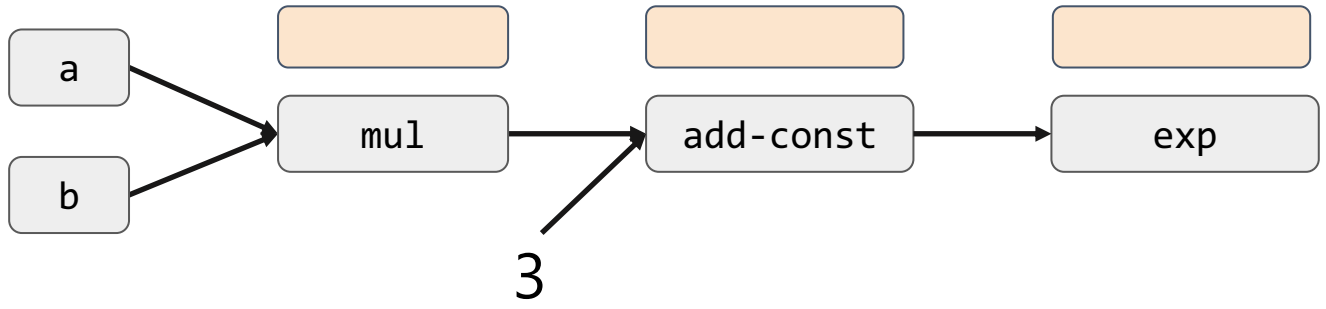
- **Inplace** store the result to the same input memory
- **Memory Reuse** reuse memory that are no longer needed.

Inplace Optimization

- Store the result to the input memory
- Works if we only care about the final result

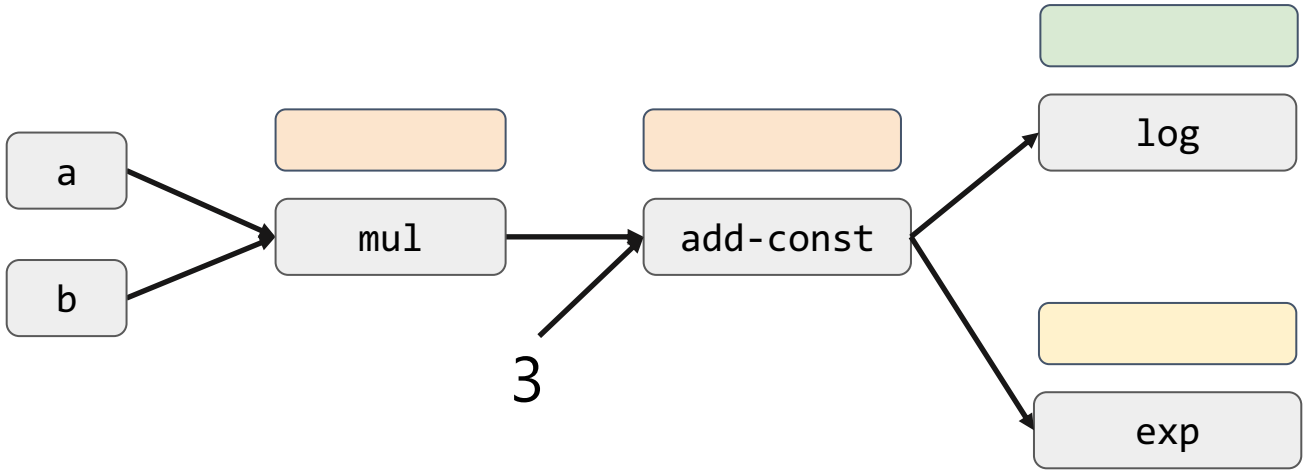
- Question: what operation cannot be done inplace ?

Computational Graph for `exp(a * b + 3)`

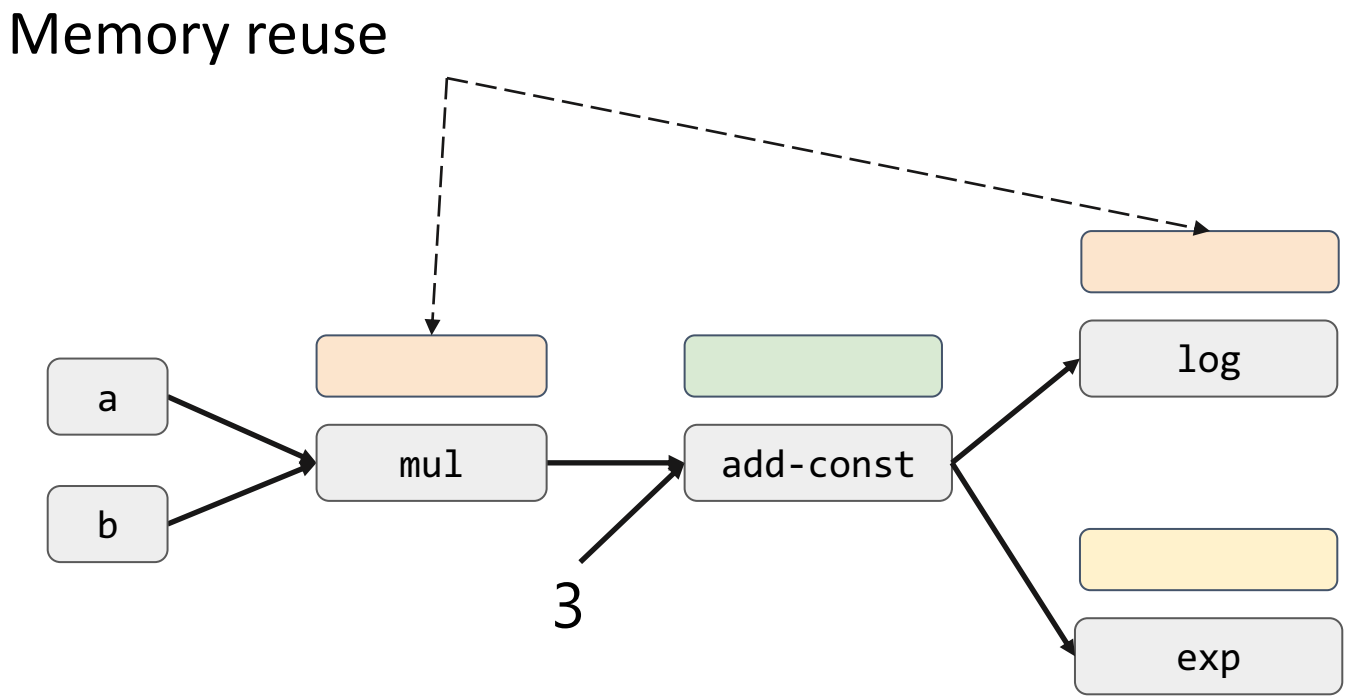


Inplace Pitfalls

We can only do inplace store if result op is the only consumer of the current value

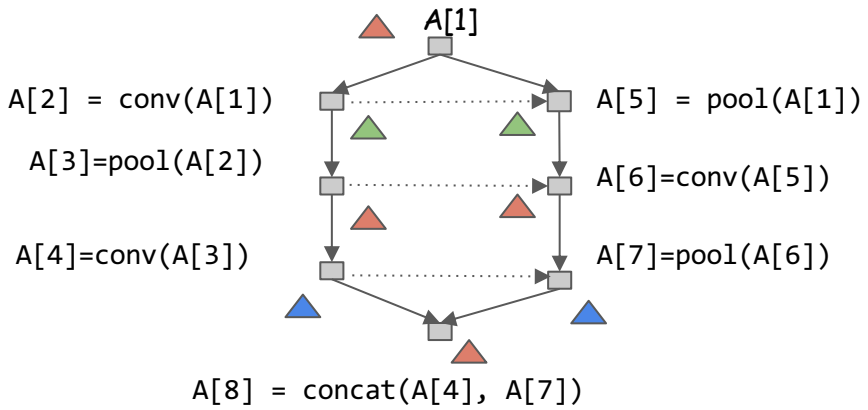


Memory Reuse

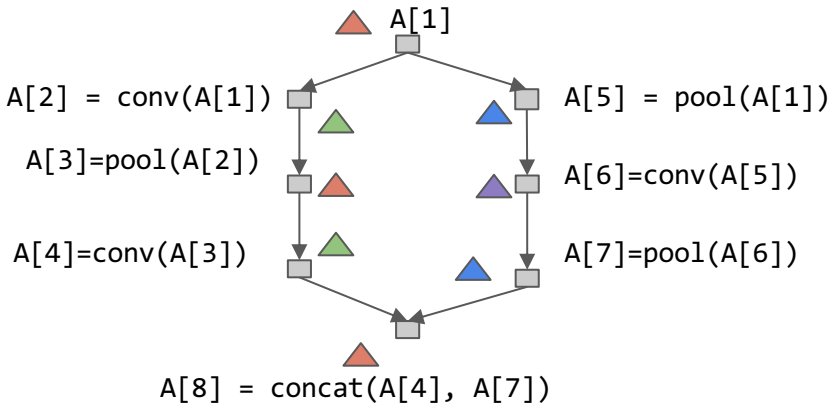


Concurrency vs Memory Optimization

Cannot Run in Parallel



Enables Two Parallel Paths



internal arrays

Memory allocation for result, same color indicates shared memory.

data dependency

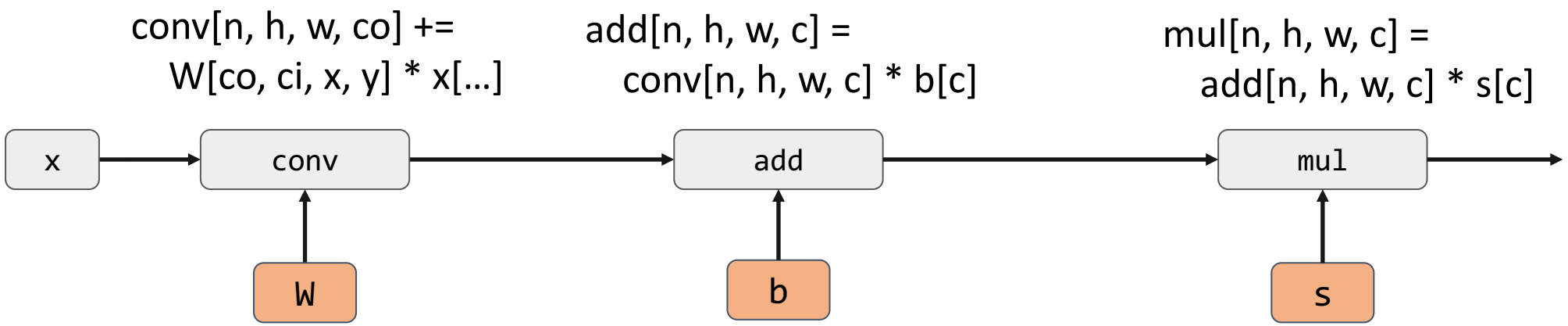
implicit dependency introduced due to allocation

More about Memory Planning

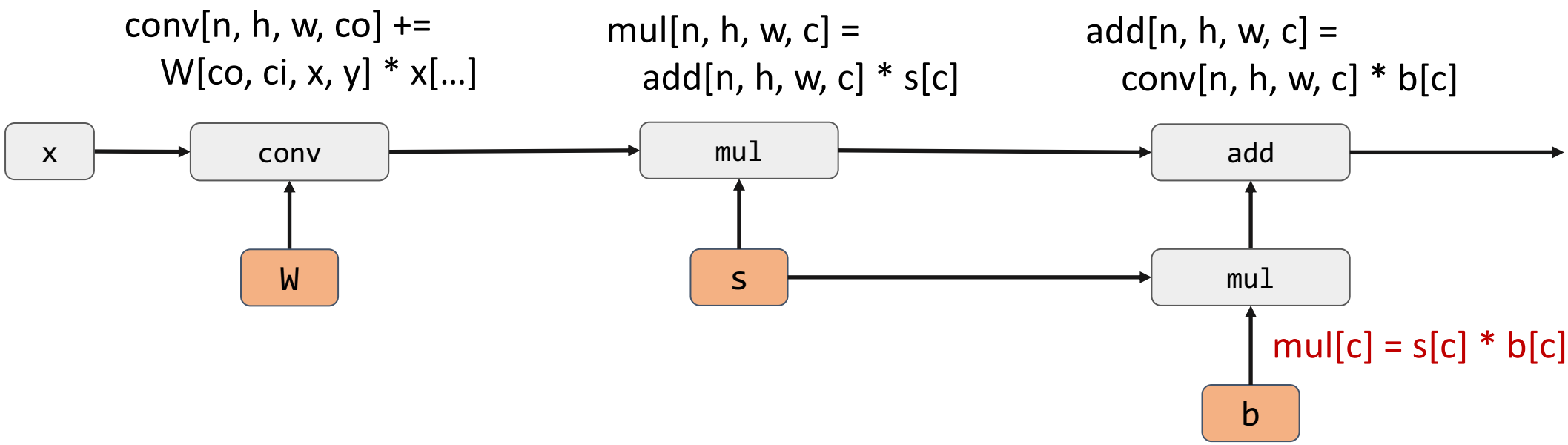
- Re-computation vs materialization (covered in the last lecture)
- More advanced reuse
  - Split a memory into two regions
  - Avoid copy during reshape
- More techniques that saves memory
  - Pruning and quantizing model weights

Rewrite Optimizations

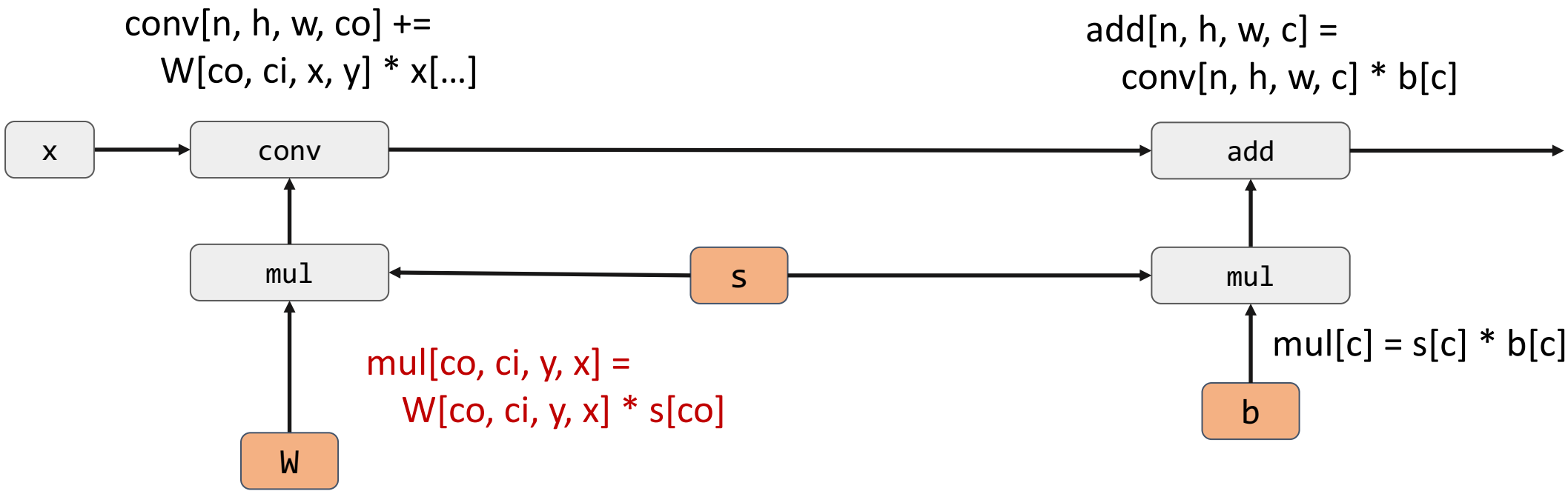
Axis Scale Re-adjustment and Folding



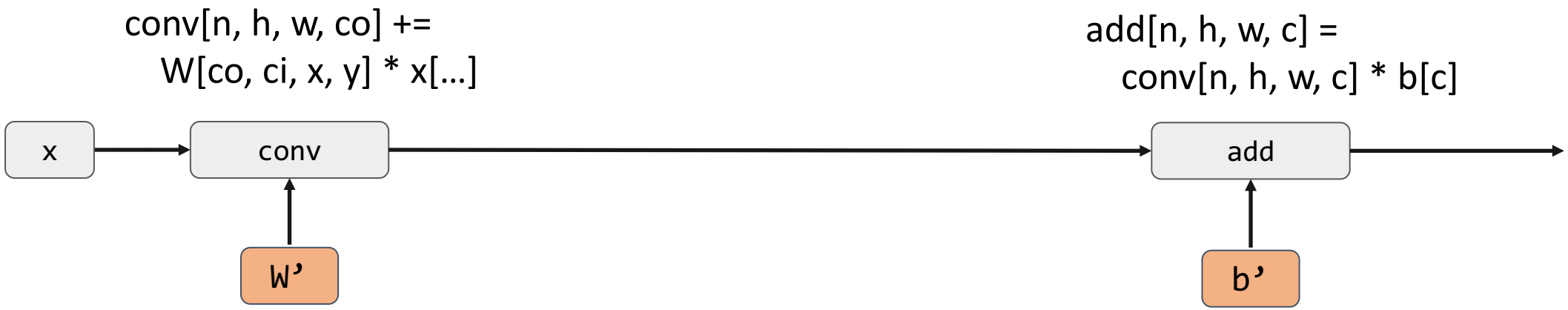
Axis Scale Re-adjustment and Folding



Axis Scale Re-adjustment and Folding



Axis Scale Re-adjustment and Folding



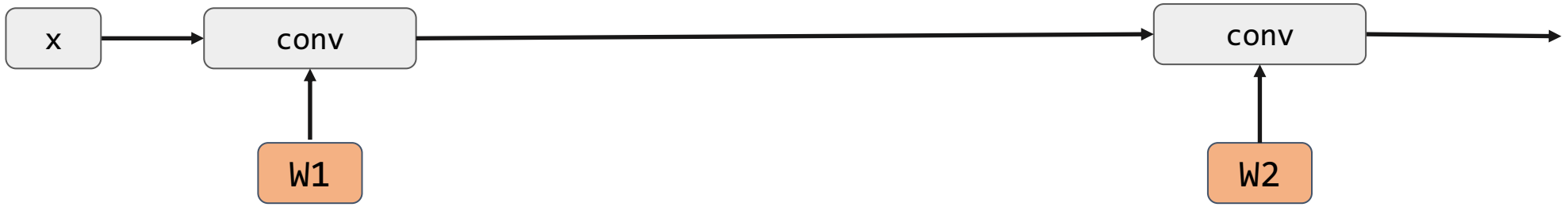
Fold scaling of tensor axis into input weights

Axis Scale Re-adjustment and Folding

- The current example is about folding backwards

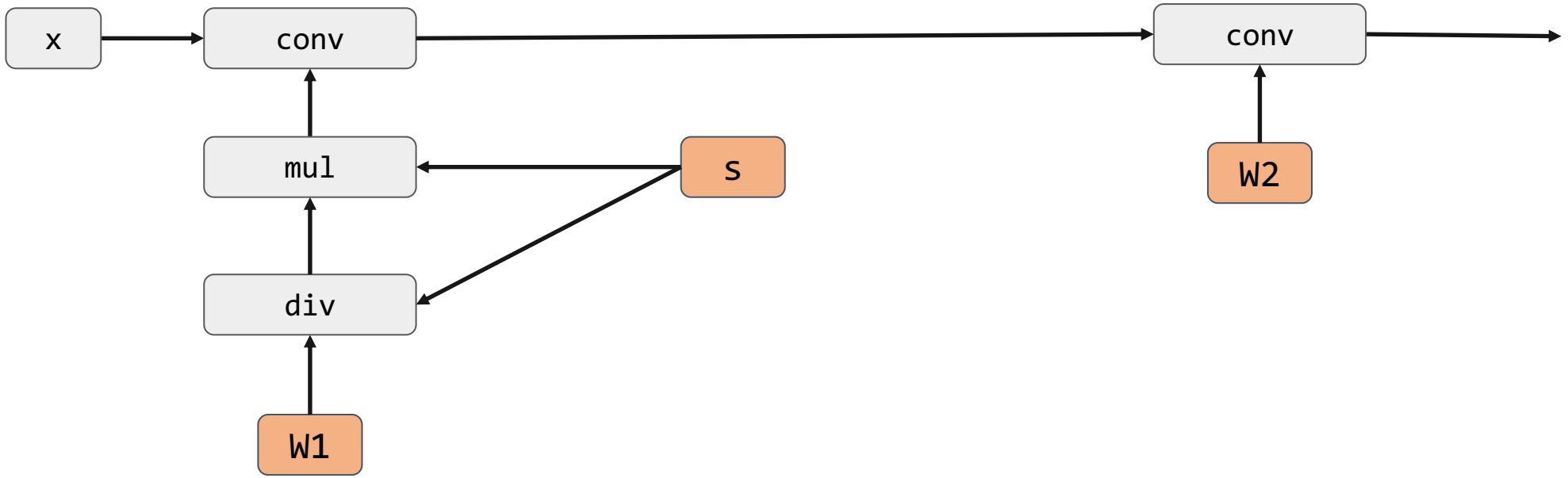
- Similar rules can be applied to fold scaling forward to the weight of next matmul conv

Axis Rebalancing

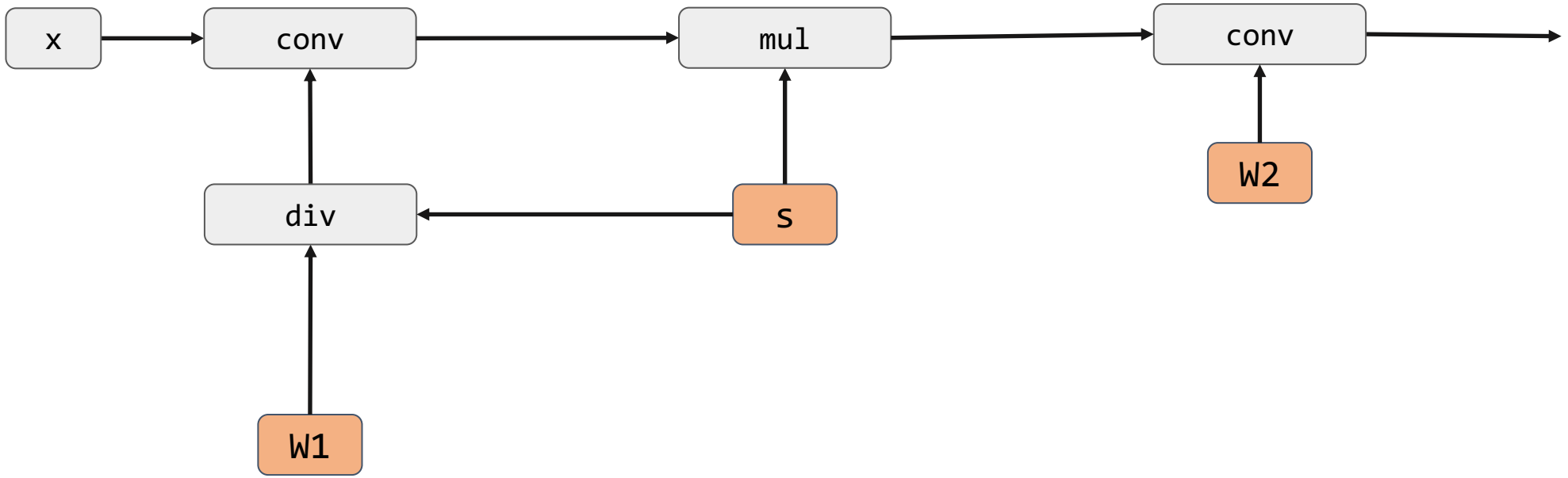


Intermediate scale and activation simplified

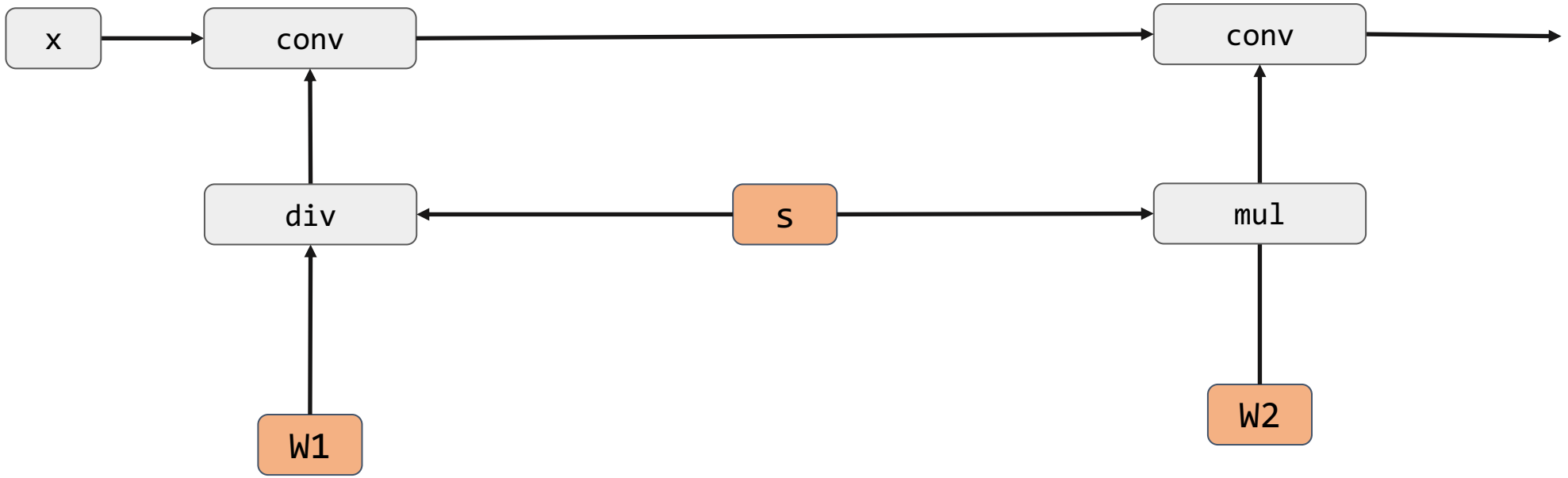
Axis Rebalancing



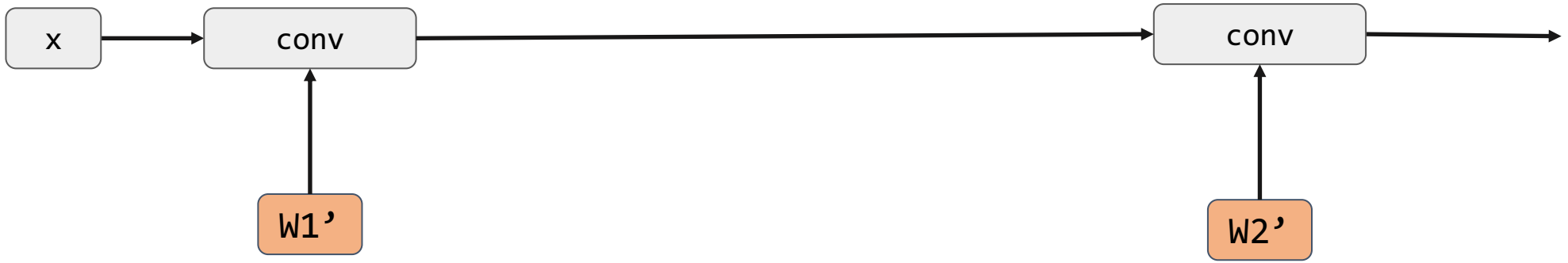
Axis Rebalancing



Axis Rebalancing

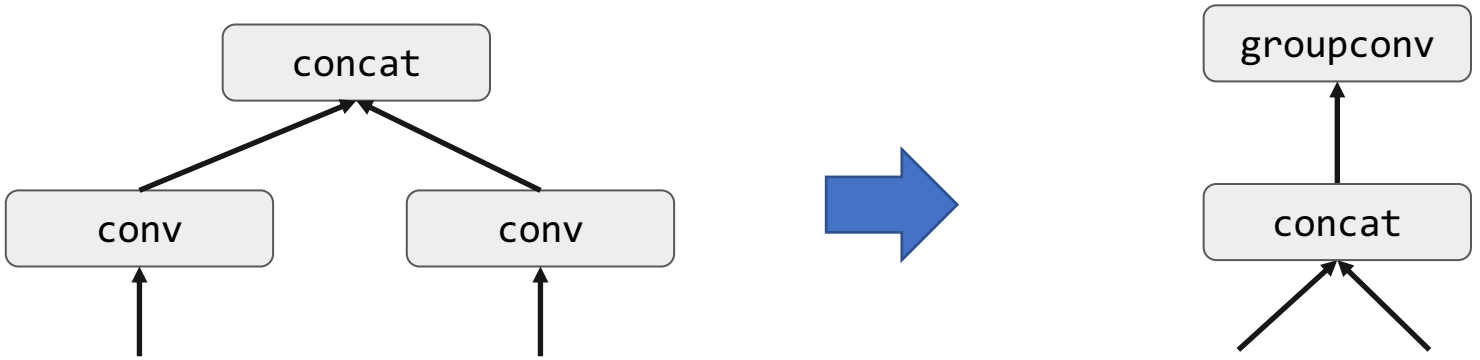


Axis Rebalancing



Sometime useful to balance the magnitude of each channels

Equivalence Rewriting



Discussion

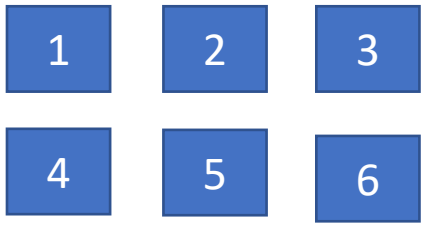
What are other possible rewrite rules?

How to choose which rewrite to apply?

Data Layout

How do we store each intermediate tensors

Original Matrix



Row major

Col major

Typical Data Layout in Vision Workloads

NCHW: X[n, c, h, w]

NHWC: X[n, h, w, c]

- n: batch
- h: height
- w: width
- c: channel

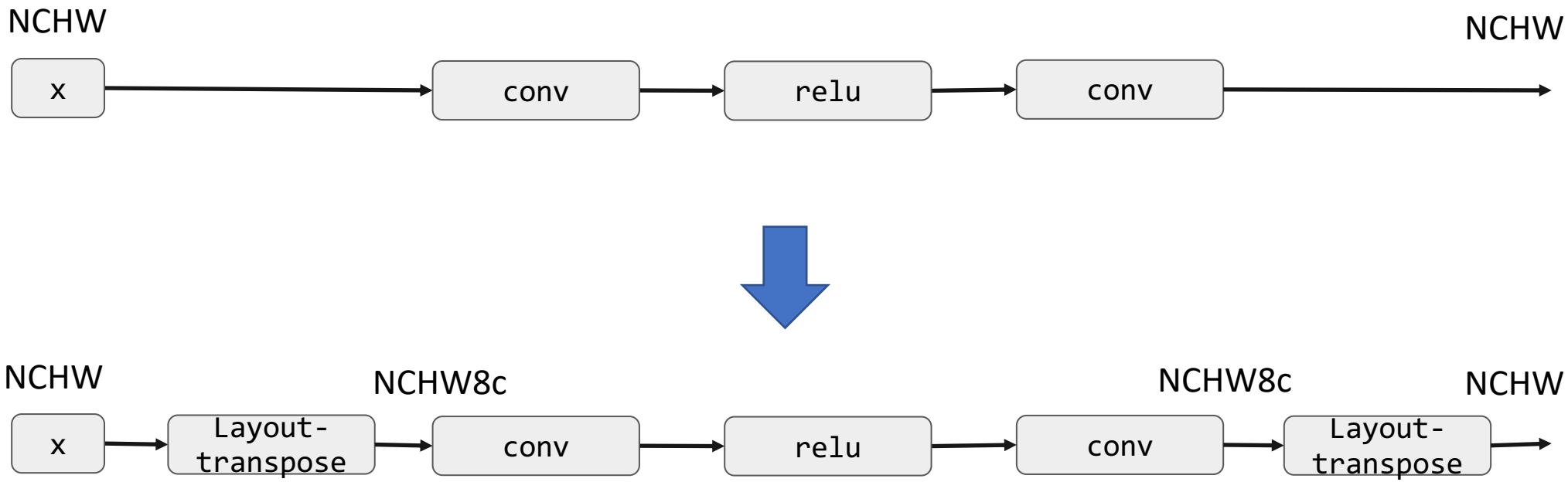
Packed Data Layout

NCHW8c: X[n, c / 8, h, w, c % 8]

- n: batch
- h: height
- w: width
- c: channel

Useful for accelerators with vector unit of 8.

Layout Conversion



Beyond Computational Graphs

Beyond Computational Graph

- State updates
- Recursive function calls
- Data structures

Example: Relay IR

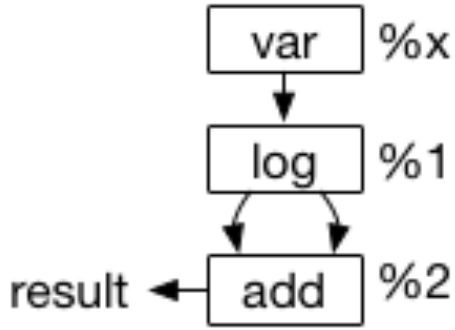
Python Code

```
x = relay.var("x")
v1 = relay.log(x)
v2 = relay.add(v1, v1)
f = relay.Function([x], v2)
```

Text Form

```
fn (%x) {
  %1 = log(%x)
  %2 = add(%1, %1)
  %2
}
```

AST Structure



Example: Relay IR

```
def @muladd(%x, %y, %z) {
   %1 = mul(%x, %y)
   %2 = add(%1, %z)
   %2
}

def @myfunc(%x) {
   %1 = @muladd(%x, 1, 2)
   %2 = @muladd(%1, 2, 3)
   %2
}
```

Multiple function calls

Let-binding Form and Data Flow Form

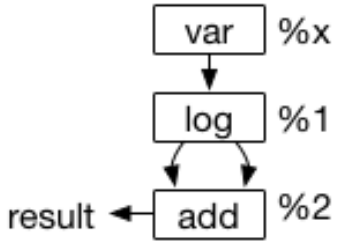
Python Code

```
x = relay.var("x")
v1 = relay.log(x)
v2 = relay.add(v1, v1)
f = relay.Function([x], v2)
```

Text Form

```
fn (%x) {
  %1 = log(%x)
  %2 = add(%1, %1)
  %2
}
```

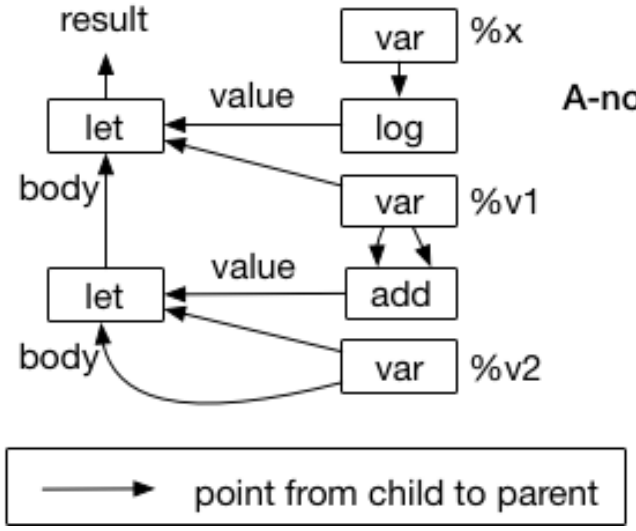
AST Structure



Dataflow

```
x = relay.var("x")
sb = relay.ScopeBuilder()
v1 = sb.let("v1", relay.log(x))
v2 = sb.let("v2", relay.add(v1, v1))
sb.ret(v2)
f = relay.Function([x], sb.get())
```

```
fn (%x) {
  let %v1 = log(%x)
  let %v2 = add(%v1, %v1)
  %v2
}
```



A-normal Form

Discussion

How would additional features (e.g. state, recursive calls) affect optimizations?

Logistics

- More on ML Compilation Later
- No class next Tuesday
- Find your team mates and start to work on proposals