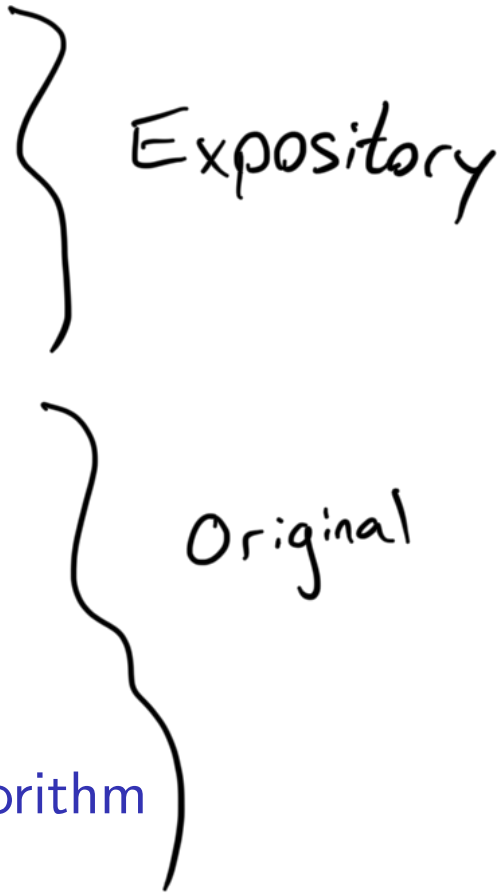
Fundamentals of CuTe Layout Algebra and Category-theoretic Interpretation

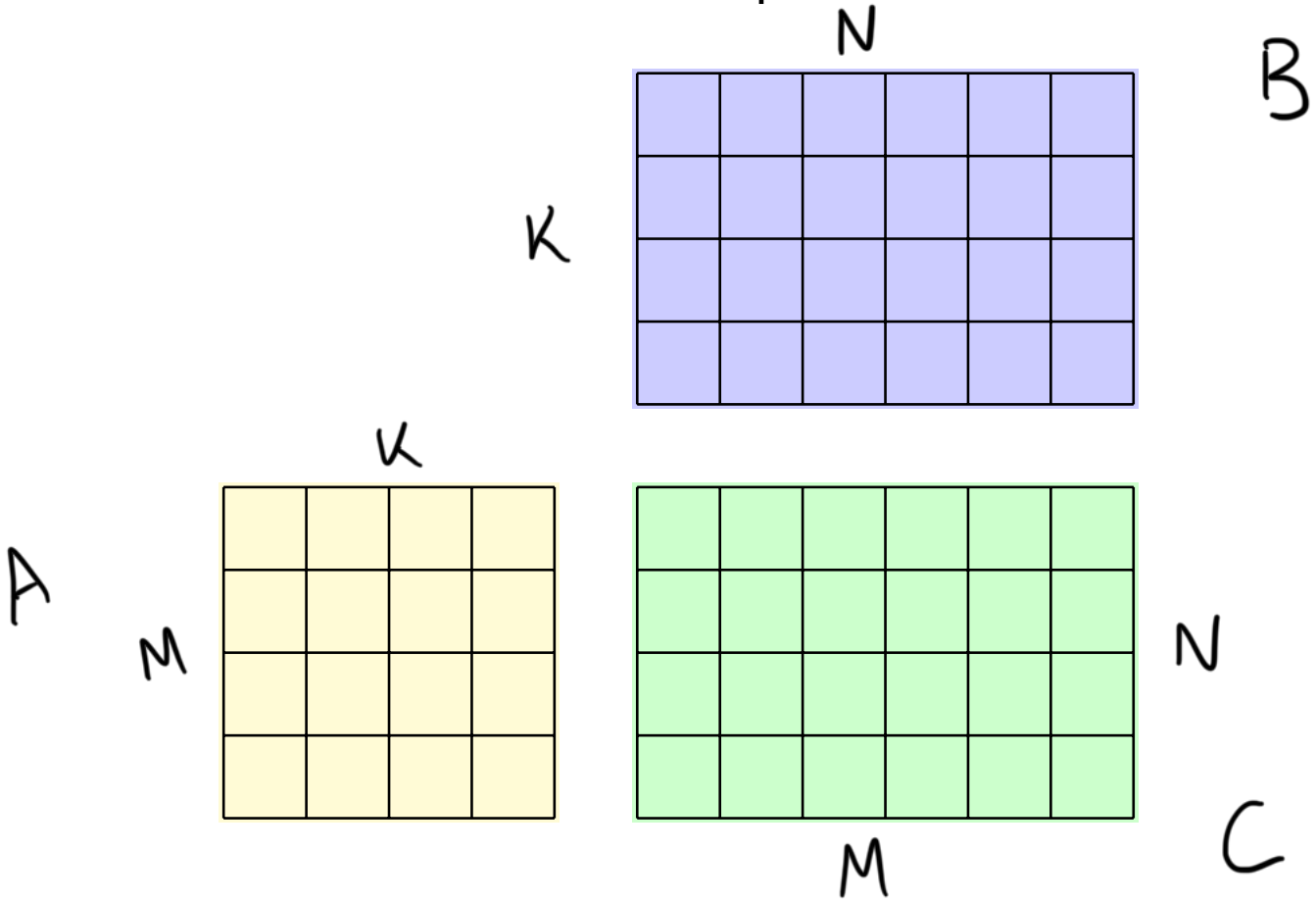
Colfax Research:

Jack Carlisle, Jay Shah, Reuben Stern, Paul VanKoughnett

April 18, 2026



Consider a GEMM kernel which computes $C = AB$.



The operands A, B and output C are *tensors*.

Tensor	Shape
A	(M, K)
B	(K, N)
C	(M, N)

} Matrices

Higher rank tensors are common, too:

Tensor	Shape
Q	(B, M, H, D)
K	(B, N, H, D)
V	(B, N, H, D)

← headdim

batch, seqlen, nheads

In CuTe, a tensor is specified by an *iterator* and a *layout*:

$$L = S : D$$

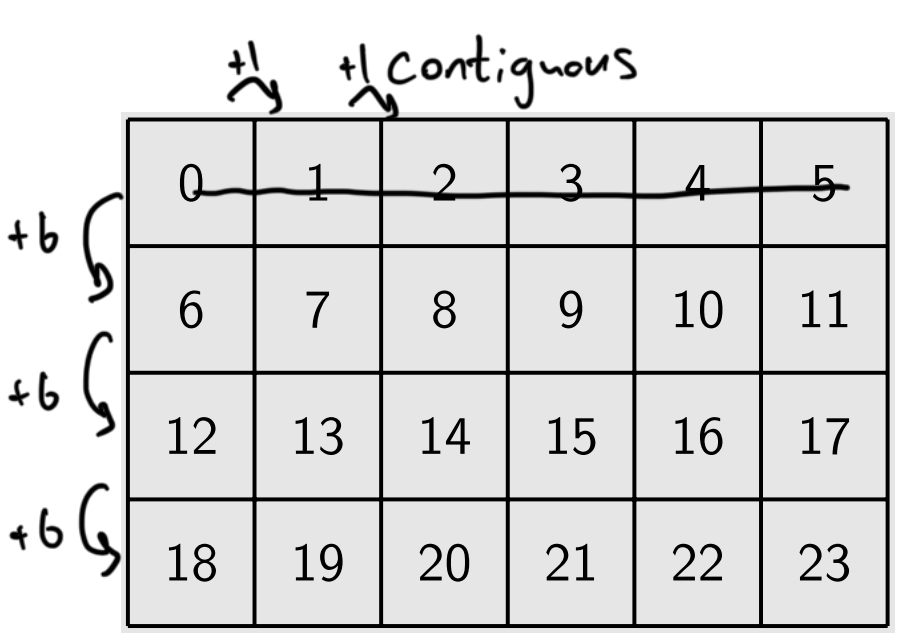
Shape Stride

Example:

4×6 matrix

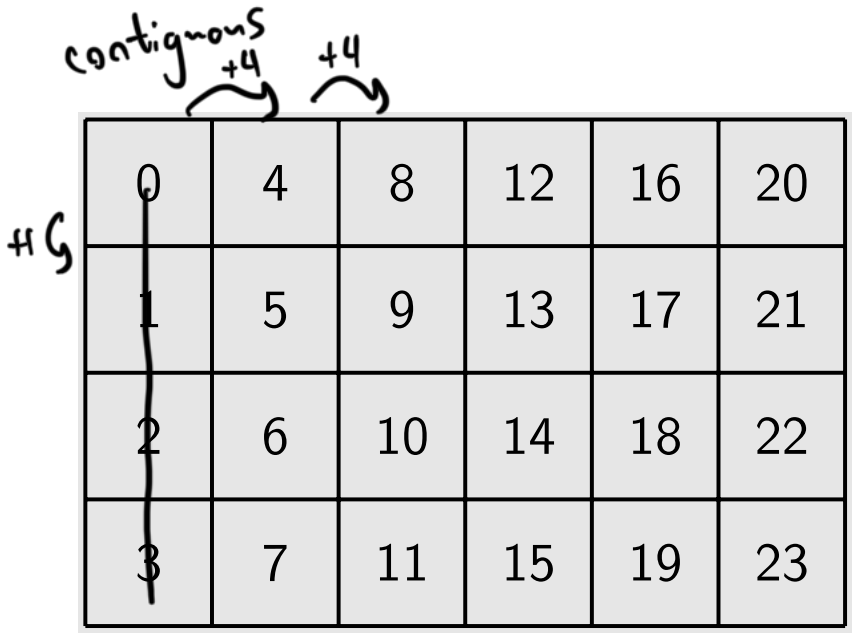
$$L = (4, 6) : (6, 1)$$
$$L = (8, 8) : (1, 0)$$
$$L = (5, 5) : (20, 2)$$
$$L = ((2, 3), (4, 2)) : ((4, 16), (1, 8))$$

Row major



$L = (4, 6) : (6, 1)$

Column Major



$L = (4, 6) : (1, 4)$

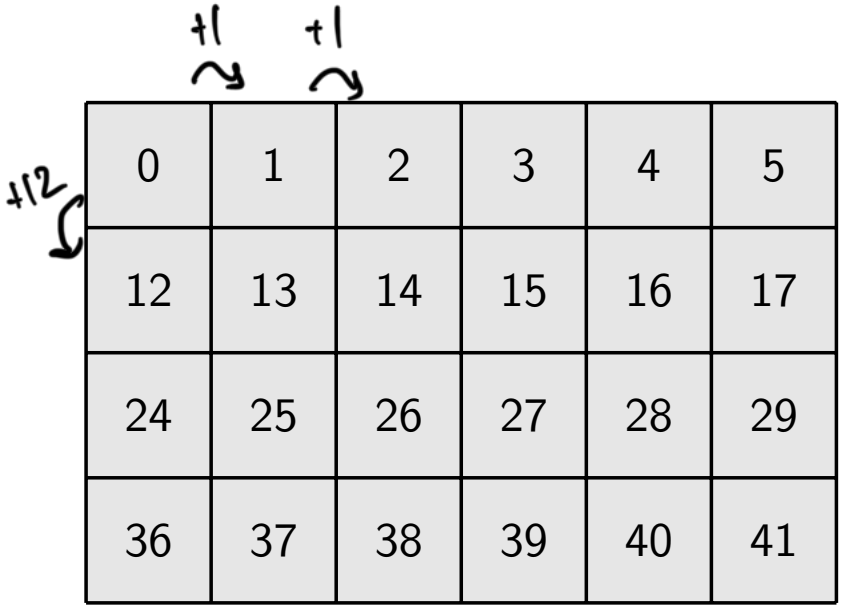
Broadcast layout

0	0	0	0	0	0
1	1	1	1	1	1
2	2	2	2	2	2
3	3	3	3	3	3

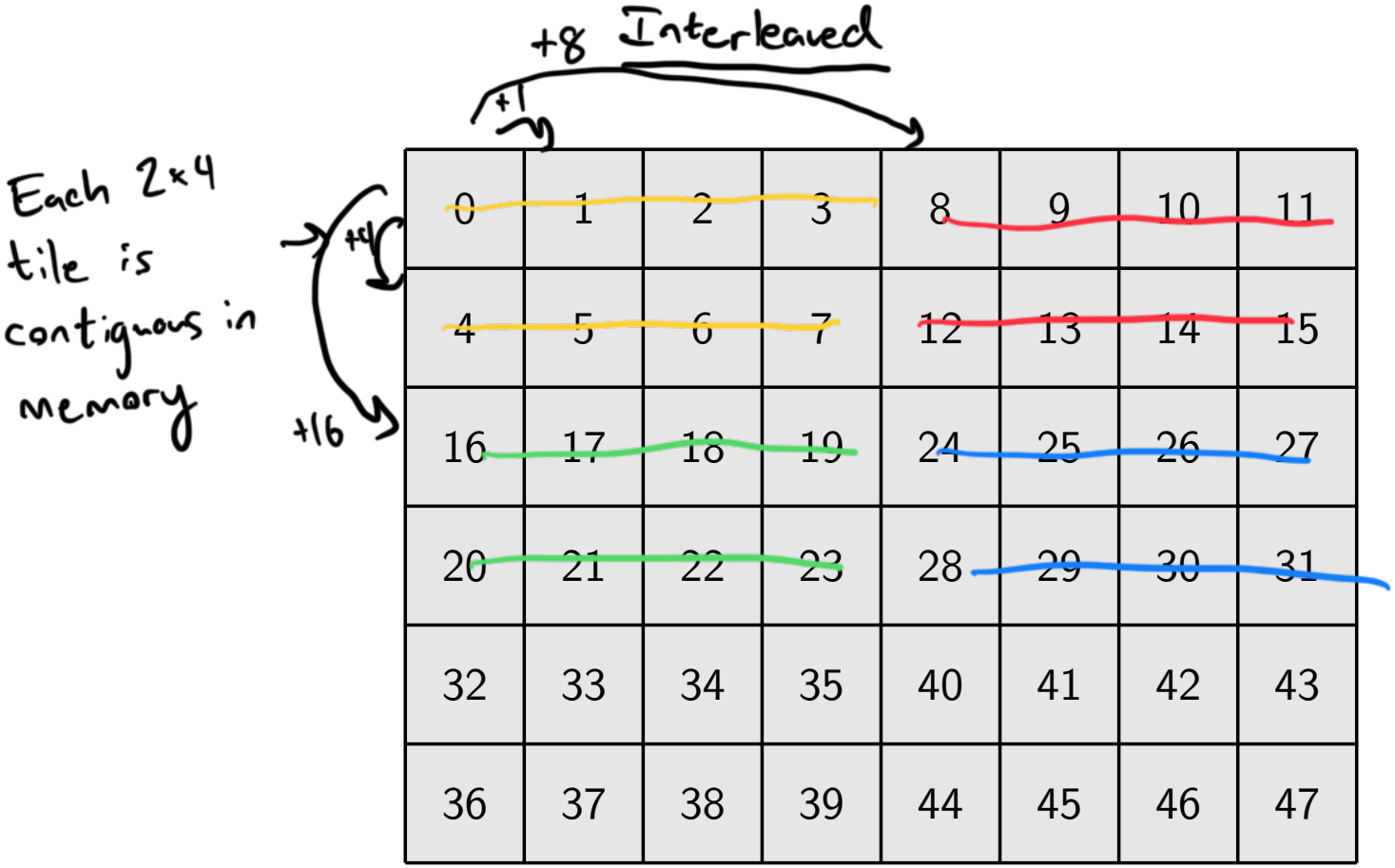
$L = (4, 6) : (1, 0)$

vector of shape 4 ⇝ matrix of shape 4×6

Strided layout



$L = (4, 6) : (12, 1)$



$$L = ((2,3),(4,2)) : ((4,16),(1,8))$$

"Logical shape 6×8"

32 threads 4 values

$$L = ((\overbrace{4,8}), (\overbrace{2,2})) : ((32,1),(16,8))$$

Row\Col	0 1	2 3	4 5	6 7
0	T0: {c0, c1}	T1: {c0, c1}	T2: {c0, c1}	T3: {c0, c1}
1	T4: {c0, c1}	T5: {c0, c1}	T6: {c0, c1}	T7: {c0, c1}
2				
..				
7	T28: {c0, c1}	T29: {c0, c1}	T30: {c0, c1}	T31: {c0, c1}
8	T0:{c2, c3}	T1:{c2, c3}	T2:{c2, c3}	T3:{c2, c3}
9	T4:{c2, c3}	T5:{c2, c3}	T6:{c2, c3}	T7:{c2, c3}
10				
..				
15	T28:{c2, c3}	T29:{c2, c3}	T30:{c2, c3}	T31:{c2, c3}

%laneid:{fragments}

Figure: Accumulator layout for a `mma.sync` instruction

128 threads

$$L = ((4, 8, 4), (2, 2, 8)) : ((128, 1, 16), (64, 8, 512))$$

(Thread, Value) ⇝ logical (M, N)

Thread 0

R\C	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...	N-8	N-7	N-6	N-5	N-4	N-3	N-2	N-1
0	T0:{d0, d1}		T1:{d0, d1}		T2:{d0, d1}		T3:{d0, d1}		T0:{d4, d5}		T1:{d4, d5}		T2:{d4, d5}		T3:{d4, d5}		...	T0:{dX,dY}		T1:{dX,dY}		T2:{dX,dY}		T3:{dX,dY}	
1	T4:{d0, d1}		T5:{d0, d1}		T6:{d0, d1}		T7:{d0, d1}		T4:{d4, d5}		T5:{d4, d5}		T6:{d4, d5}		T7:{d4, d5}		...	T4:{dX,dY}		T5:{dX,dY}		T6:{dX,dY}		T7:{dX,dY}	
...																	...								
7	T28:{d0, d1}		T29:{d0, d1}		T30:{d0, d1}		T31:{d0, d1}		T28:{d4, d5}		T29:{d4, d5}		T30:{d4, d5}		T31:{d4, d5}		...	T28:{dX,dY}		T29:{dX,dY}		T30:{dX,dY}		T31:{dX,dY}	
8	T0:{d2, d3}		T1:{d2, d3}		T2:{d2, d3}		T3:{d2, d3}		T0:{d6, d7}		T1:{d6, d7}		T2:{d6, d7}		T3:{d6, d7}		...	T0:{dZ, dW}		T1:{dZ, dW}		T2:{dZ, dW}		T3:{dZ, dW}	
.																	...								
15	T28:{d2, d3}		T29:{d2, d3}		T30:{d2, d3}		T31:{d2, d3}		T28:{d6, d7}		T29:{d6, d7}		T30:{d6, d7}		T31:{d6, d7}		...	T28:{dZ, dW}		T29:{dZ, dW}		T30:{dZ, dW}		T31:{dZ, dW}	
16	T32:{d0, d1}		T33:{d0, d1}		T34:{d0, d1}		T35:{d0, d1}		T32:{d4, d5}		T33:{d4, d5}		T34:{d4, d5}		T35:{d4, d5}		...	T32:{dX, dY}		T33:{dX, dY}		T34:{dX, dY}		T35:{dX, dY}	
..																	...								
31	T60:{d2, d3}		T61:{d2, d3}		T62:{d2, d3}		T63:{d2, d3}		T60:{d6, d7}		T61:{d6, d7}		T62:{d6, d7}		T63:{d6, d7}		...	T60:{dZ, dW}		T61:{dZ, dW}		T62:{dZ, dW}		T63:{dZ, dW}	
32	T64:{d0, d1}		T65:{d0, d1}		T66:{d0, d1}		T67:{d0, d1}		T64:{d4, d5}		T65:{d4, d5}		T66:{d4, d5}		T67:{d4, d5}		...	T64:{dX, dY}		T65:{dX, dY}		T66:{dX, dY}		T67:{dX, dY}	
..																	...								
47	T92:{d2, d3}		T93:{d2, d3}		T94:{d2, d3}		T95:{d2, d3}		T92:{d6, d7}		T93:{d6, d7}		T94:{d6, d7}		T95:{d6, d7}		...	T92:{dZ, dW}		T93:{dZ, dW}		T94:{dZ, dW}		T95:{dZ, dW}	
48	T96:{d0, d1}		T97:{d0, d1}		T98:{d0, d1}		T99:{d0, d1}		T96:{d4, d5}		T97:{d4, d5}		T98:{d4, d5}		T99:{d4, d5}		...	T96:{dX, dY}		T97:{dX, dY}		T98:{dX, dY}		T99:{dX, dY}	
..																	...								
63	T124:{d2, d3}		T125:{d2, d3}		T126:{d2, d3}		T127:{d2, d3}		T124:{d6, d7}		T125:{d6, d7}		T126:{d6, d7}		T127:{d6, d7}		...	T124:{dZ, dW}		T125:{dZ, dW}		T126:{dZ, dW}		T127:{dZ, dW}	

Warp 0 | Warp 1 | Warp 2 | Warp 3

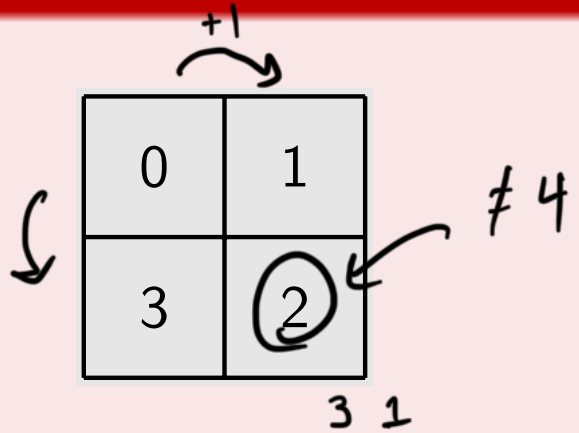
(tid % 128) : fragments

Figure: Accumulator layout for a `wgmma` instruction

Not all labellings correspond to a valid layout.

Non-example:

Consider



$$L = (2,2) : (?,?)$$

Need layout with swizzle function to represent this mapping.

"Composed Layout"

These images depict the *coordinate function* of L.

Notation:

If S is a nested tuple, we write $[0, S)$

$$prof\Big(((3,2),2)\Big) = prof\Big(((1,1),4)\Big)$$

$$[0, S) = \{X \mid prof(X) = prof(S) \text{ and } 0 \leq x_i < s_i\}.$$

Definition: (Coordinate function)

If $L = S : D$ is a layout, then the *coordinate function* of L is given by

$$[0, S) \xrightarrow{D\cdot} \mathbb{Z}$$

$$X \longmapsto D \cdot X$$

$((4,0),2)$.

$L = ((3,2),2) : ((4,0),2)$

$[0,S)$

$$[0,S) \xrightarrow{D\cdot} \mathbb{Z}$$

$$\begin{aligned}
((0,0),0) &\longmapsto 4*0+0*0+2*0 = 0\\
((1,0),0) &\longmapsto 4*1+0*0+2*0 = 4\\
((2,0),0) &\longmapsto 4*2+0*1+2*0 = 8\\
((0,1),0) &\longmapsto 4*0+0*1+2*0 = 0\\
((1,1),0) &\longmapsto 4*1+0*0+2*0 = 4\\
((2,1),0) &\longmapsto 4*2+0*0+2*0 = 8\\
((0,0),1) &\longmapsto 4*0+0*1+2*1 = 2\\
((1,0),1) &\longmapsto 4*1+0*1+2*1 = 6\\
((2,0),1) &\longmapsto 4*2+0*0+2*1 = 10\\
((0,1),1) &\longmapsto 4*0+0*0+2*1 = 2\\
((1,1),1) &\longmapsto 4*1+0*1+2*1 = 6\\
((2,1),1) &\longmapsto 4*2+0*1+2*1 = 10
\end{aligned}$$

Remark:

Coordinate function has

- Inputs: nested tuples,
- Outputs: integers.

Can't compose these!

Can provide more general inputs. First, we need a vocabulary for working with nested tuples.

Definition (by example):

Consider $S = (4, (5, 5), (2, (3, 3)))$:

1. The *modes* of S are $4, \quad (5, 5), \quad (2, (3, 3))$.
2. The *entries* of S are $4, \quad 5, \quad 5, \quad 2, \quad 3, \quad 3,$
3. The *rank* of S is the number of modes $rank(S) = 3$
4. The *length* of S is the number of entries: $len(S) = 6$.
5. The *size* of S is the product of its entries: $size(S) = 1800$.

"compatible with"

Definition:

1. We say S *refines* S', and write

$$S \twoheadrightarrow S'$$

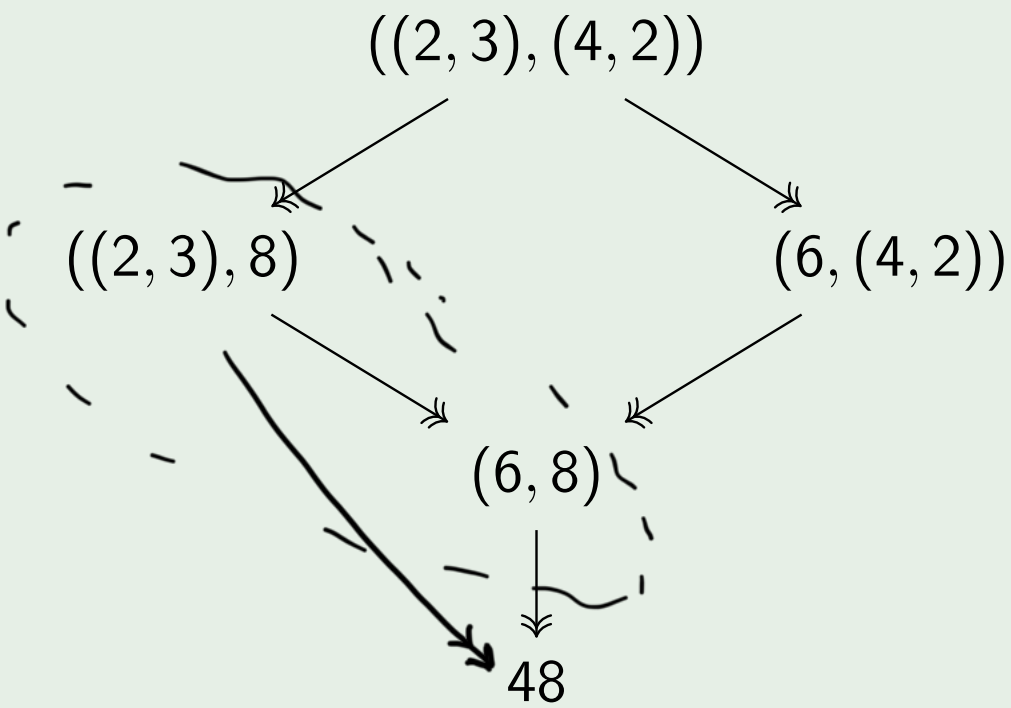
if S may be obtained from S' by replacing some of its entries with nested tuples of the same size.

$$((2,2),(5,5)) \twoheadrightarrow (4,25)$$

$$S \twoheadrightarrow S'$$

Example: (Nested tuple refinements)

Nested tuples refined by $S = ((2,3),(4,2))$:



Definition:

If $S = (s_1, \ldots, s_m)$ is a (nested) tuple of size N, then the *colexicographic isomorphism* is

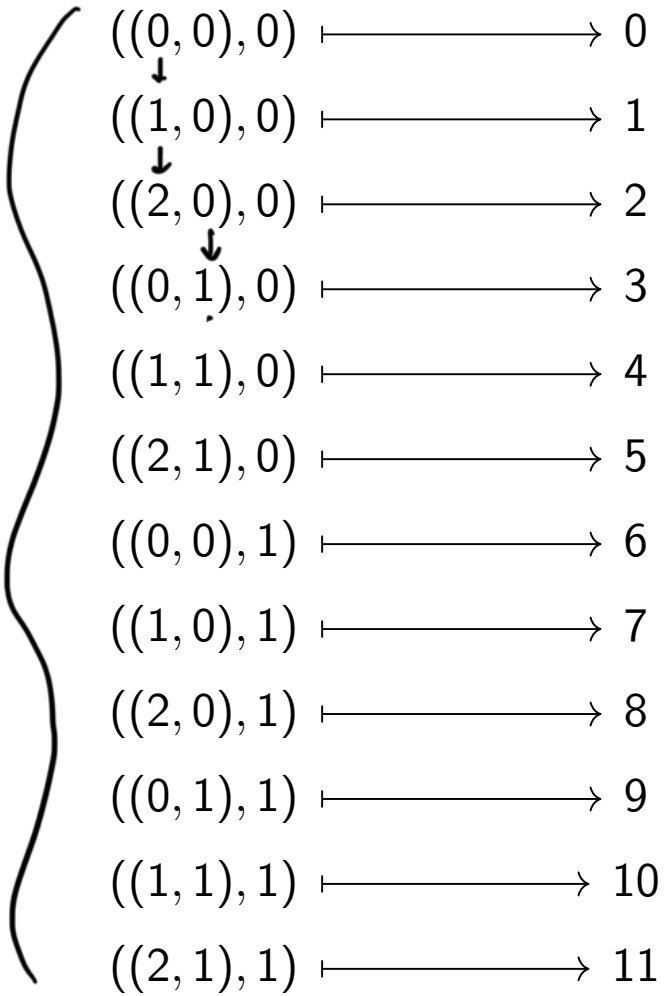
$$[0, S) \xrightarrow{\cong} [0, N)$$

$$(x_1, \ldots, x_m) \xmapsto{colex} \sum_{i=1}^{m} s_1 \cdots s_{i-1} x_i$$

$$\left(\left\lfloor \frac{x}{s_1 \cdots s_{i-1}} \right\rfloor \mod s_i \right)_{i=1}^{m} \xmapsto{colex^{-1}} x$$

$S = ((3, 2), 2)$

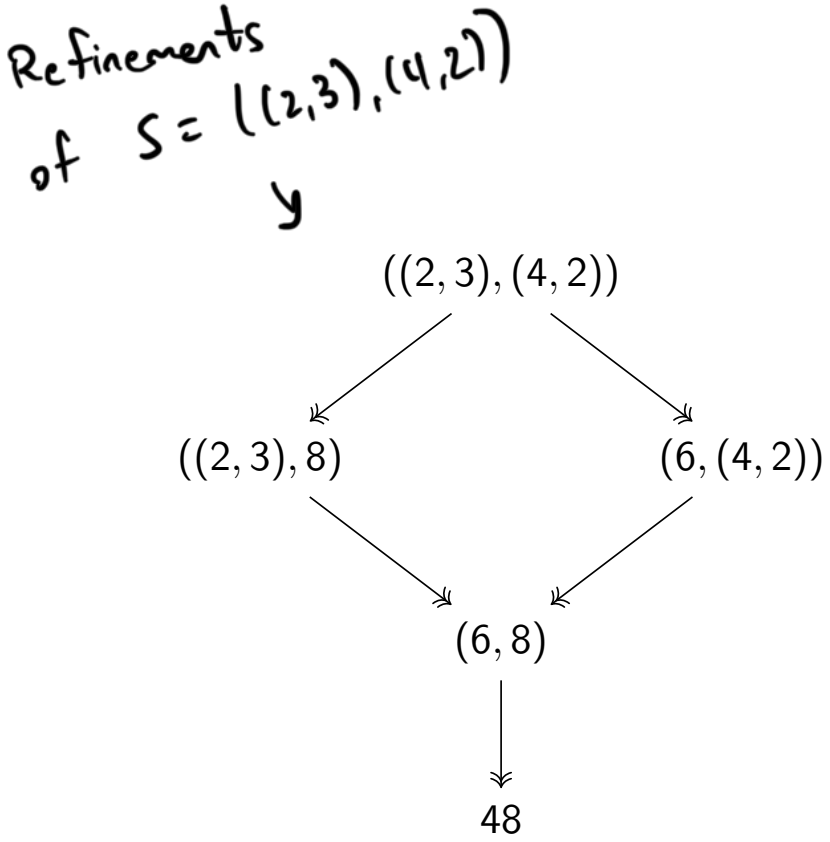
$$[0, S) \xrightarrow{\text{colex}} [0, 12)$$

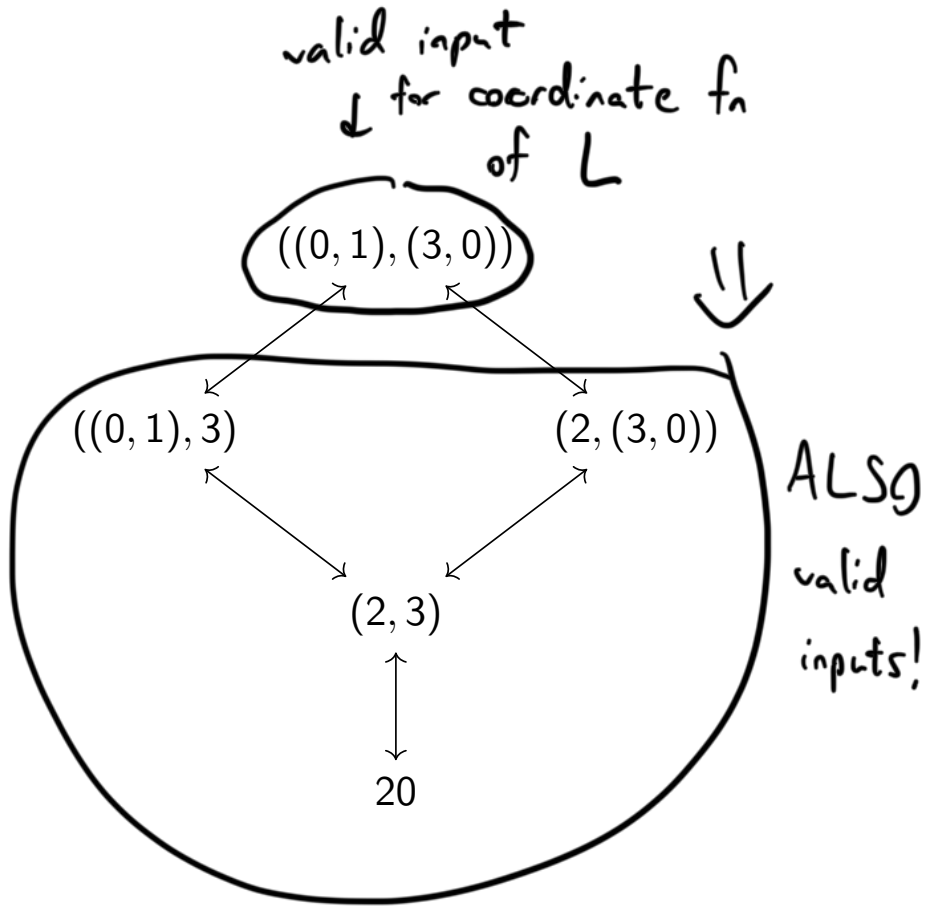


More generally, if S refines S', then there is a colexicographic isomorphism

$$[0, S) \xrightarrow[\text{colex}]{\cong} [0, S')$$

Same formula applied "entrywise".

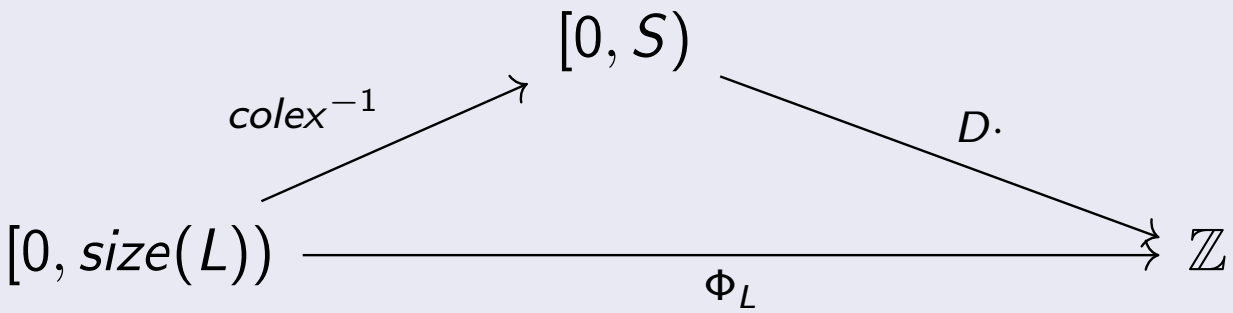

Refinements
of S = ((2,3),(4,2))
y
((2, 3), (4, 2))
((2, 3), 8)
(6, (4, 2))
(6, 8)
48

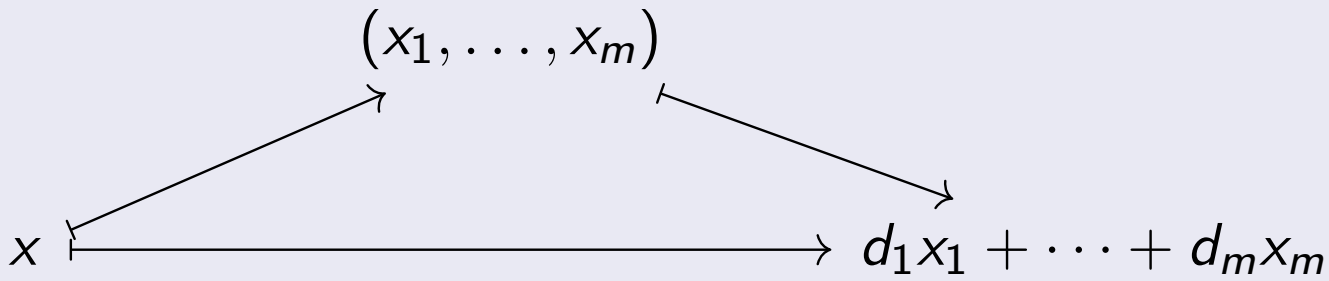

valid input
for coordinate fn
of L
((0, 1), (3, 0))
((0, 1), 3)
(2, (3, 0))
(2, 3)
20
ALSO
valid
inputs!

We especially care about integer inputs.

Definition:

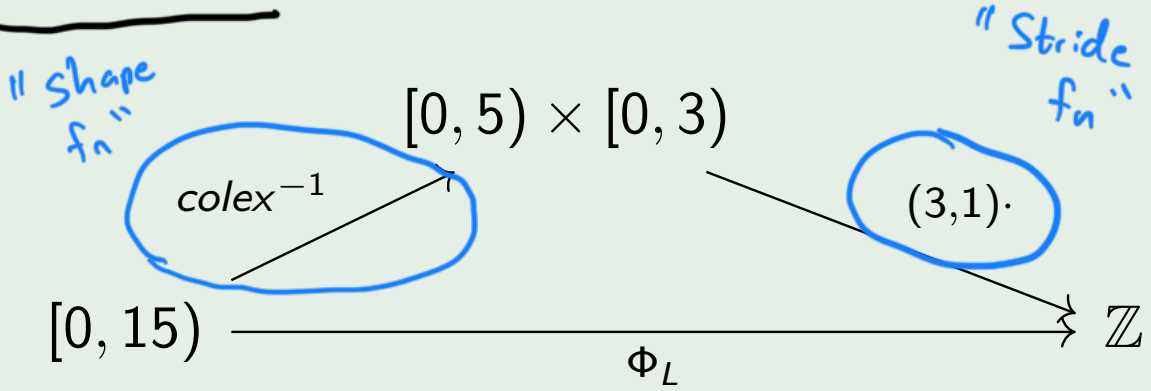
The *layout function* of $L = S : D$ is the composite

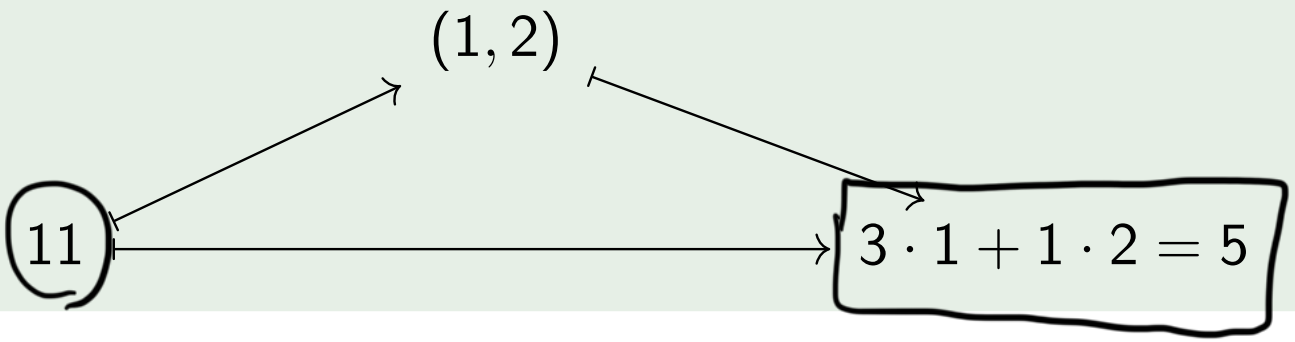




Example:

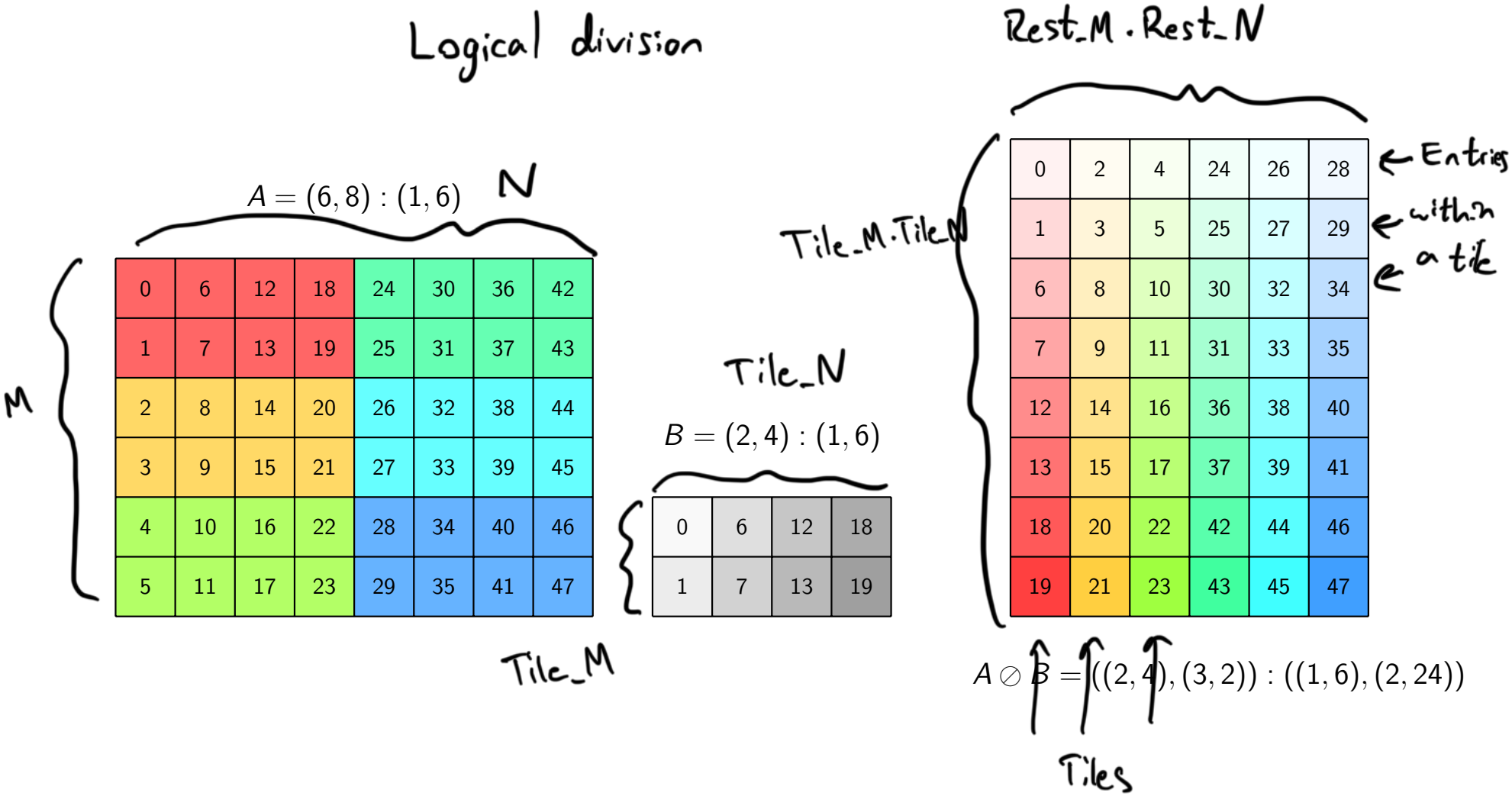
$L = (5, 3) : (3, 1)$





Interleaved layouts arise naturally via algebraic operations.

1. Logical division $A \oslash B$
2. Logical product $A \otimes B$
3. Variants of these


Logical division
Rest_M . Rest_N
$A = (6, 8) : (1, 6)$
N
M
Tile_M . Tile_N
Entries
within a tile
Tile_N
$B = (2, 4) : (1, 6)$
Tile_M
$A \oslash B = ((2, 4), (3, 2)) : ((1, 6), (2, 24))$
Tiles

Tile ↓

$A = (3,3) : (3,1)$

0	1	2
3	4	5
6	7	8

Arrangements of Tiles ↓

$B = (2,2) : (1,4)$

0	4
1	5

0	9	36	45
3	12	39	48
6	15	42	51
1	10	37	46
4	13	40	49
7	16	43	52
2	11	38	47
5	14	41	50
8	17	44	53

← entries
← within
← each tile

$A \otimes B = ((3,3),(2,2)) : ((3,1),(9,36))$

Tiles

These are defined in terms of more fundamental layout operations.

Definition:

1. The logical division of A by B is

$$A \oslash B = (A \circ B, A \circ B^c)$$

where $B^c = comp(B, size(A))$

2. The logical product of A and B is

$$A \otimes B = (A, A^c \circ B)$$

where $A^c = comp(A, size(A) * cosize(B))$.

Need to understand composition, complements, coalesce, concatenation.

What is composition?

Definition:

If A and B are layouts, then the composition of A and B is a layout

"a group composition" -Cris

$$B \circ A$$

satisfying

1. The shape of $B \circ A$ refines the shape of A, and
2. $\Phi_{B \circ A} = \Phi_B \circ \Phi_A$.

Example: (Composition)

$$A = (4, 6) : (6, 1)$$ ← Row major

$$B = (12, 4) : (4, 1)$$ ← Row major

$$\boxed{B \circ A = ((2, 2), 6) : ((24, 1), 4)}$$

Check

Φ_A colex^{-1}

$$\Phi_A(3) = (3, 0) \cdot (6, 1) = 18$$

colex^{-1}

$$\Phi_B(18) = (6, 1) \cdot (4, 1) = \boxed{25}$$

$$\Phi_{B \circ A}(3) = ((1, 1), 0) \cdot ((24, 1), 4) = \boxed{25}$$ ✓

colex^{-1}

$(L, L', L'') \neq (L, (L', L''))$

$\neq ((L, L'), L'')$

What is concatenation?

Definition:

If $L = S : D$ and $L' = S' : D'$ are layouts, then their concatenation is

$$(L, L') = (S, S') : (D, D').$$

Example: (Concatenation)

If $L = (2, 2) : (2, 1)$ and $L' = (3, 5) : (1, 3)$, then

$$(L, L') = ((2, 2), (3, 5)) : ((2, 1), (1, 3)).$$

$\text{size}((L, L')) = \text{size}(L) \cdot \text{size}(L')$.

What is complementation?

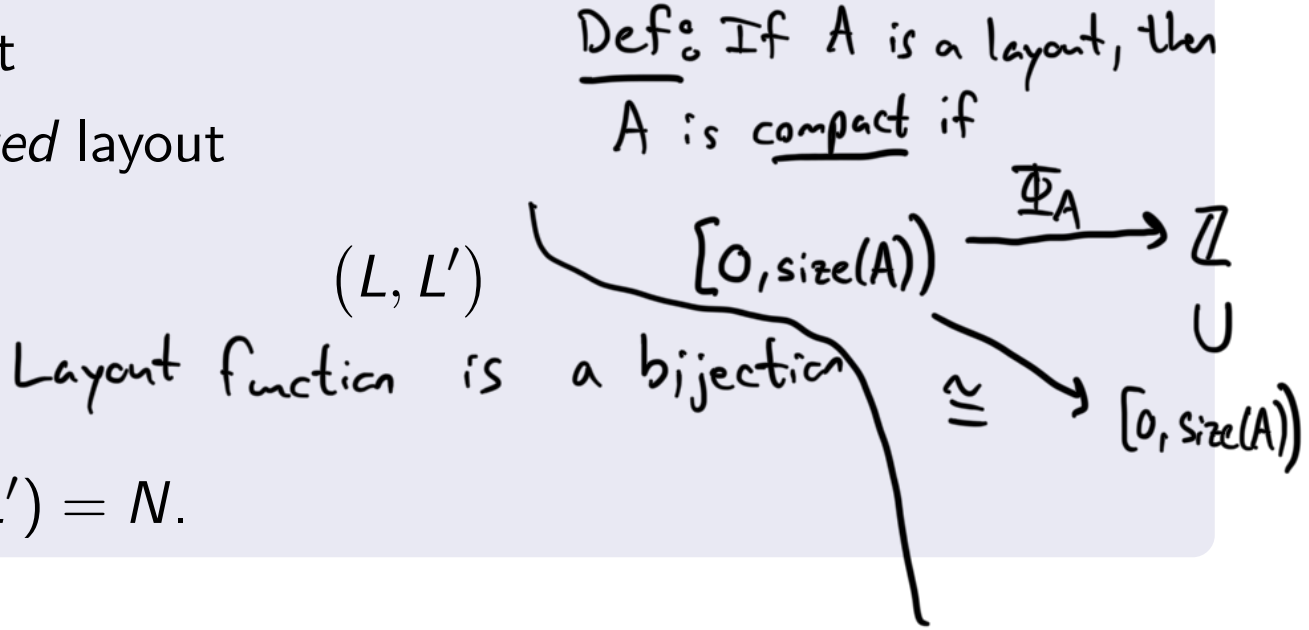
Definition:

If L is a layout and N is an integer, then

$$L' = comp(L, N)$$

is a layout such that

1. the *concatenated* layout (L, L') is *compact*.
2. $size(L) * size(L') = N$.



Example: (Complement)

If $L = (3,7) : (2,30)$ and $N = 210$, then

$$L' = comp(L, N) = (2,5) : (1,6).$$

since

$$(L, L') = ((2,5),(3,7)) : ((1,6),(2,30))$$

is compact of size 210.

What is coalesce?

Definition:

If L is a layout, then $coal(L)$ is obtained by

1. flattening L, and
2. consolidating adjacent modes when possible:

$$s_i, s_{i+1} : d_i, d_{i+1} \rightsquigarrow s_i s_{i+1} : d_i$$

when $s_i d_i = d_{i+1}$.

This "simplifies" the layout without altering the layout function Φ_L.

Example: (Coalesce)

If $L = (2, 3, \underline{2}, 3) : (12, 6, \underline{1}, 2)$, then

$$coal(L) = (2, 3, 6) : (12, 6, 1)$$

$2 \cdot 1 = 2$

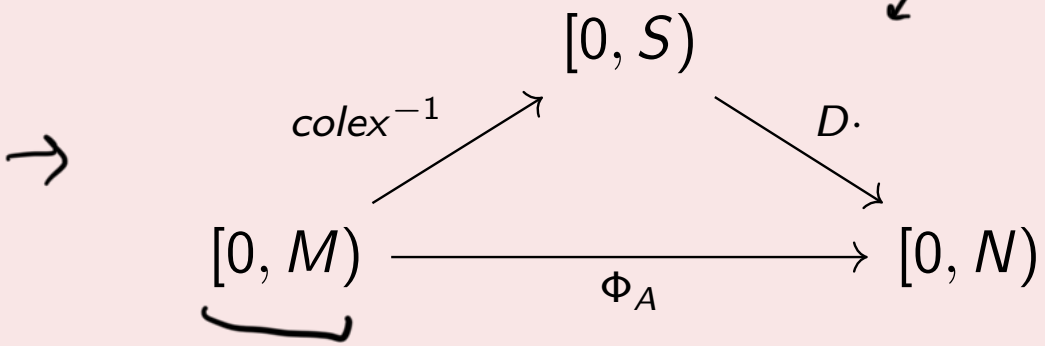
$s_i \cdot d_i = d_{i+1}$ $\rightsquigarrow$ combine these modes.

Computing these layout operations, especially composition, is subtle:

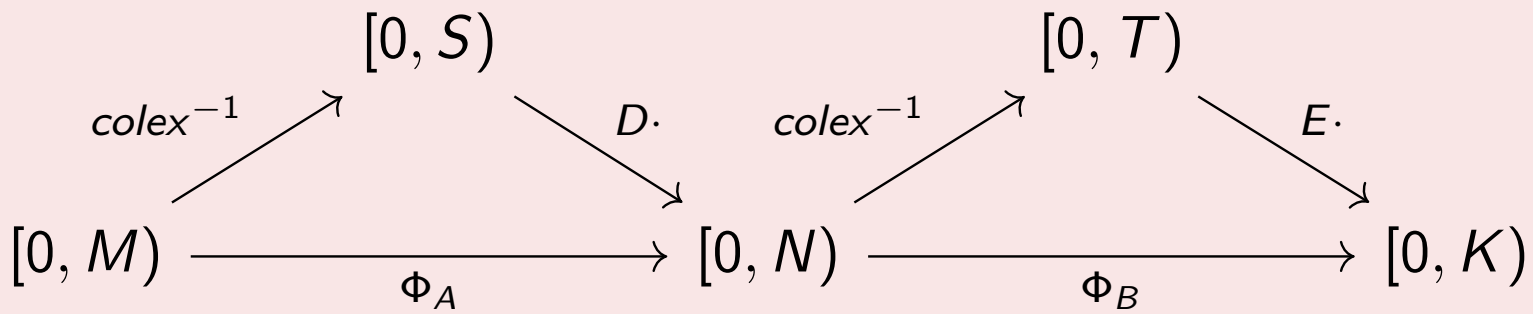
Composition subtleties:

If A is a layout, then Φ_A has the form

multilinear

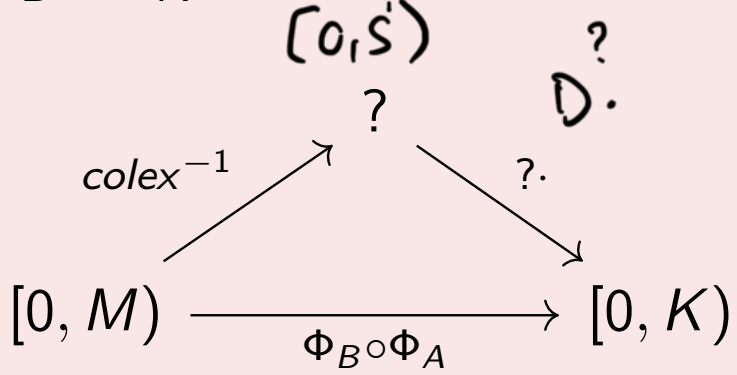


If we compose Φ_A with some other Φ_B we get



Composition subtleties:

Q: Can we factor $\Phi_B \circ \Phi_A$ as shown below?



A: Not always! Want to identify conditions that guarantee that $B \circ A$ exists.

Recap:

1. Layouts are important:
   - Specify how data is stored in memory
   - Specify how data is partitioned among agents
2. Need to work with
   - Logical division,
   - Logical product,
   - Composition,
   - Complement.

Can develop an intuitive and powerful mathematical theory of layouts by restricting to *tractable* layouts. These are

1. Expressive enough to support the operations we care about (permutations, slicing, broadcasting, partitioning, etc.), and
2. Simple enough to admit a clean mathematical theory.

Definition:

We say a layout L is *tractable* if L may be obtained from a column major layout by

1. permuting modes,
2. removing modes,
3. inserting modes with stride 0,
4. reparenthesizing.

Remark:

Can also give an explicit divisibility condition which characterizes tractable layouts.

Example: (Compact layouts)

If L is *compact*, then L is tractable. For instance,

$$L = ((2,2),(2,2)) : ((8,1),(4,2))$$

Example: (Strided layouts)

If L is obtained from a tractable layout by removing modes, then L is tractable. For instance,

$$L = (8,8) : (1,16).$$

Example: (Broadcast layouts)

If L is obtained from a tractable layout by appending modes with stride 0, then L is tractable. For instance,

$$L = (128,128) : (1,0)$$

Our goal is to connect the theory of layouts to some well-established concepts in abstract algebra, especially *categories*. These arise naturally in mathematics, physics, and computer science.

Definition:

A **category C** consists of

1. a collection of objects

$$ob(\mathbf{C}) = \{X, Y, Z, \dots\},$$

2. a collection of morphisms

$$mor(\mathbf{C}) = \{f, g, h, \dots\},$$

3. a composition rule

$$X \xrightarrow{f} Y \xrightarrow{g} Z \quad \rightsquigarrow \quad X \xrightarrow{g \circ f} Z$$

domain of f

codomain of f

Example 1:

In the category **FinSet**,

1. an object is a finite set $[0, n) = \{0, \ldots, n-1\}$ $(n \geq 0)$,
2. a morphism is a function

$$[0, n) \xrightarrow{f} [0, m)$$

$[0,3) \xrightarrow{f} [0,4)$

$f(0) = 2$
$f(1) = 2$
$f(2) = 3$

3. Composition is functional composition

$$[0, n) \xrightarrow{f} [0, m) \xrightarrow{g} [0, p) \quad \rightsquigarrow \quad [0, n) \xrightarrow{g \circ f} [0, p)$$

$(g \circ f)(x) = g(f(x))$

Example 2:

In the category **Vect**,

1. an object is a vector space $\mathbb{R}^n$ $(n \geq 0)$,
2. a morphism is a matrix

$$\mathbb{R}^n \xrightarrow{A} \mathbb{R}^m$$

3. Composition is matrix multiplication

$$\mathbb{R}^n \xrightarrow{A} \mathbb{R}^m \xrightarrow{B} \mathbb{R}^p \quad \rightsquigarrow \quad \mathbb{R}^n \xrightarrow{BA} \mathbb{R}^p$$

Example 3:

In the category $\mathbf{Z}_{>0}$,

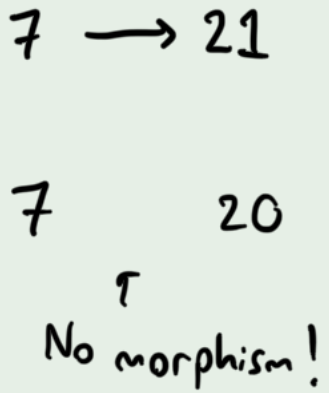
1. an object is a positive integer a.
2. There is a unique morphism

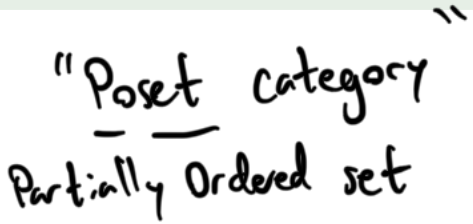
$$a \to b$$

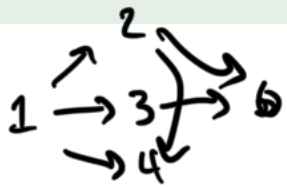
if a divides b, otherwise none.

3. Composition $\Leftrightarrow$ divisibility is transitive.

$$a \longrightarrow b \longrightarrow c \quad \rightsquigarrow \quad a \longrightarrow c$$







There are also morphisms between categories called *functors*.

Definition:

If **C** and **D** are categories, then a *functor* is an assignment

$$\mathbf{C} \xrightarrow{\quad F \quad} \mathbf{D}$$

$$\begin{array}{ccc} X & \longmapsto & FX \\ f \downarrow & \longmapsto & \downarrow Ff \\ Y & \longmapsto & FY \end{array}$$

that is compatible with composition:

$$F(g \circ f) = Fg \circ Ff$$

compose in C — Compose in D.

Example

$$\mathbf{FinSet} \xrightarrow{\quad F \quad} \mathbf{Vect}$$

$$\begin{array}{ccc} [0,n) & \longmapsto & \mathbb{R}^n \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle Ff} \\ [0,m) & \longmapsto & \mathbb{R}^m \end{array}$$

an $m \times n$ matrix!

$$(Ff)_{i,j} = \begin{cases} 1 & f(j) = i \\ 0 & \text{else} \end{cases}$$

Exercise: Check

f, g composable

$F(g \circ f) = Fg\,Ff$ (matrix mult.)

functional composition

Want to define a category whose morphisms encode layouts.
We begin with the flat case.
Later, we will extend to the nested case.

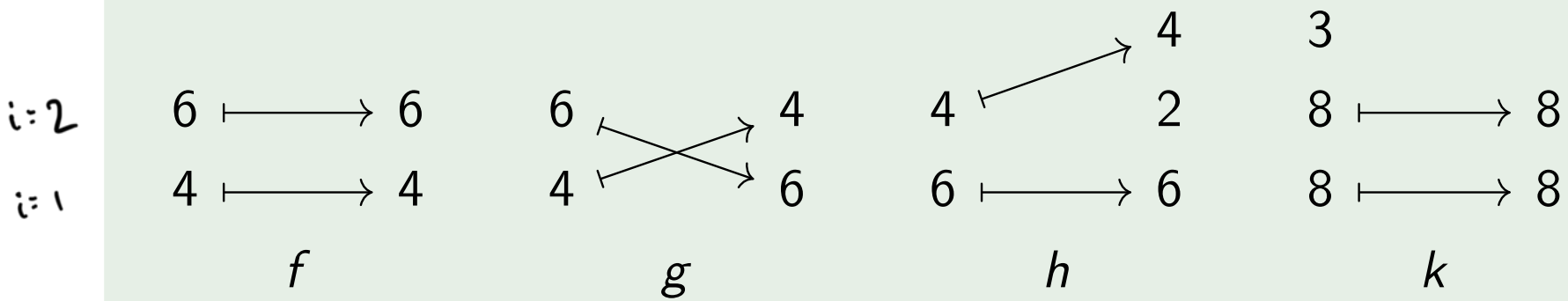
Definition:

In the category **Tuple**, positive integers

1. an object is a tuple $(s_1, \ldots, s_m)$.
2. A morphism $f : (s_1, \ldots, s_m) \to (t_1, \ldots, t_n)$ is specified by a function
$$\alpha : \{*, 1, \ldots, m\} \to \{*, 1, \ldots, n\}$$
satisfying
   - $\alpha(*) = *$,
   - If $\alpha(i) = \alpha(i') \neq *$, then $i = i'$,
   - If $\alpha(i) = j$, then $s_i = t_j$.
3. Composition is functional composition of underlying maps:
$$S \xrightarrow[\alpha]{f} T \xrightarrow[\beta]{g} U \qquad \rightsquigarrow \qquad S \xrightarrow[\beta \circ \alpha]{g \circ f} U$$

Example:

Here are some **Tuple** morphisms.



$(4,6) \xrightarrow{f} (4,6)$

lies over

$\alpha(1) = 1$

$\alpha(2) = 2$

$(6,4) \xrightarrow{h} (6,2,4)$

lies over

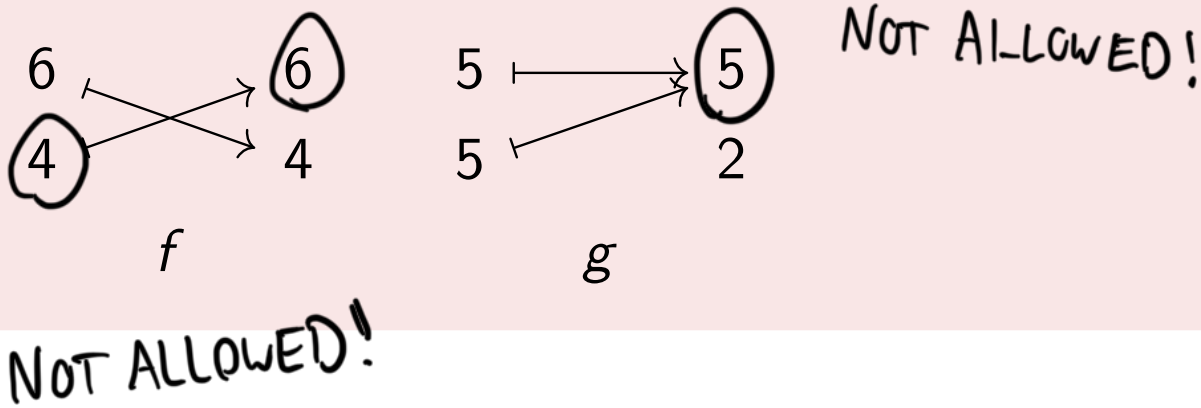
$\alpha : \{*,1,2\} \to \{*,1,2,3\}$

$\alpha(1) = 1$

$\alpha(2) = 3$

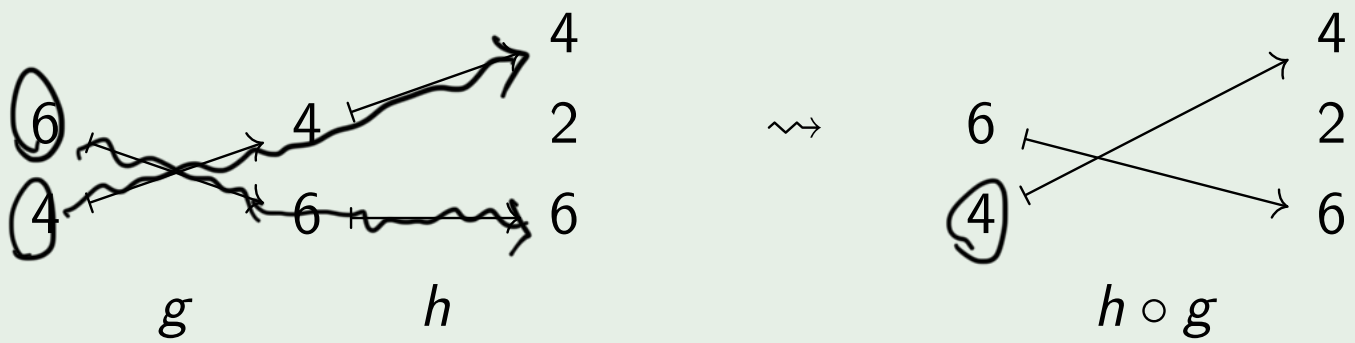
Non-example:

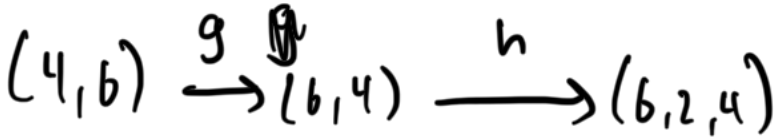
These diagrams do not depict valid **Tuple**-morphisms.



We compose morphisms by pasting diagrams.

Example:





Mathematical Remark:

Construction of **Tuple** is motivated by the theory of *operads*.

1. Associative operad **Assoc**,
2. Commutative operad **Comm**,
3. Lie operad **Lie**.

More precisely, our category **Tuple** (and variants we will consider later) occur naturally as subcategories of the category of operators of certain (colored) operads.

Each **Tuple**-morphism encodes a layout:

Construction:

If

$$(s_1, \ldots, s_m) \xrightarrow[\alpha]{f} (t_1, \ldots, t_n)$$

is a **Tuple**-morphism we define

$$L_f = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$

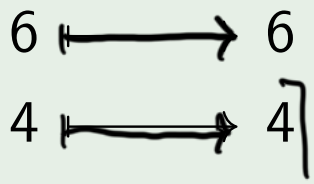
prefix product

where

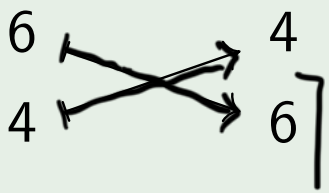
$$d_i = \begin{cases} t_1 \cdots t_{\alpha(i)-1} & \alpha(i) \neq * \\ 0 & \alpha(i) = *. \end{cases}$$

zero

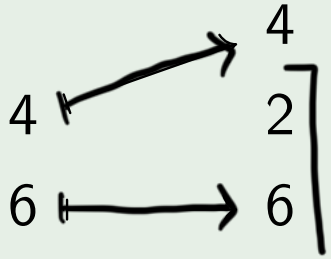
Example:



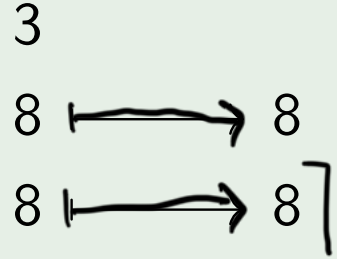
f



g



h



k

$L_f = (4,6) : (1,4)$

Col Major

$L_g = (4,6) : (6,1)$

Row major

$L_h = (6,4) : (1,12)$

Strided layout

$\alpha : \{*_{1..1}3\} \to \{*_{1..1}2\}$

$\alpha(3) = *$

$L_k = (8,8,3) : (1,8,0)$

Broadcast layouts

Not all layouts arise from this construction.

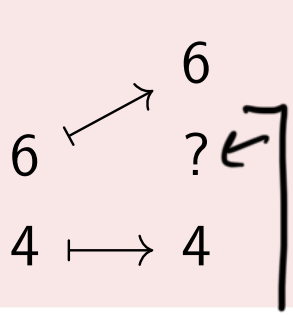
Example:

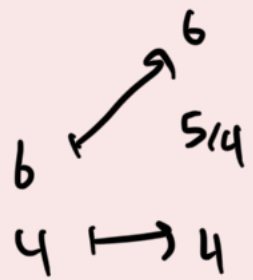
The layout

$$L = (4, 6) : (1, 5)$$

will never arise from a tuple morphism!

does not arise as L_f for a **Tuple**-morphism f:





Want to characterize which layouts arise.

Recall the notion of a tractable layout:

Definition:

We say a layout L is *tractable* if L may be obtained from a column major layout by

1. permuting modes,
2. removing modes,
3. inserting modes with stride 0,
4. reparenthesizing.

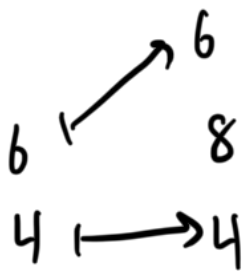
Correspondence Theorem:

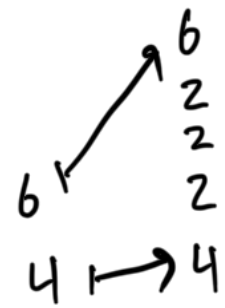
1. The layouts which arise from **Tuple**-morphisms are exactly the flat tractable layouts.
2. There is a one-to-one correspondence

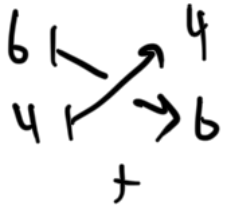
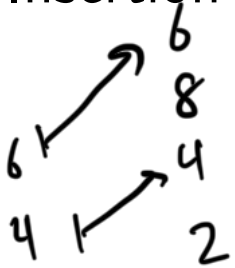
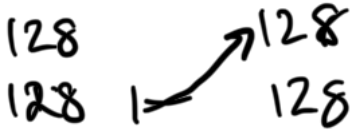
shape = 1 => stride = 0

$$\left\{\begin{array}{c}\text{Non-degenerate flat}\\ \text{tractable layouts}\end{array}\right\} \leftrightarrow \left\{\begin{array}{c}\text{Non-degenerate}\\ \textbf{Tuple}\text{-morphisms of}\\ \text{standard form}\end{array}\right\}$$

(4,6):(1,32)



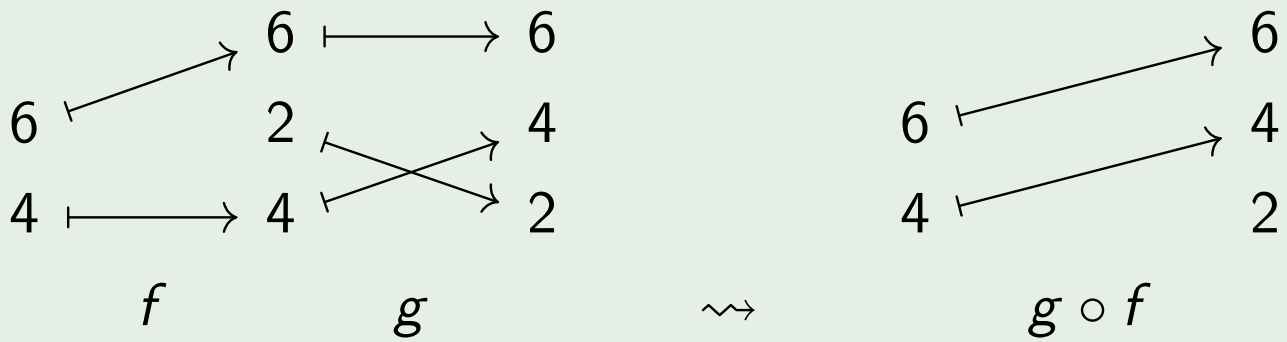


Compact layout	$\leftrightarrow$	Permutation
$L = (4, 6) : (6, 1)$	$\leftrightarrow$	
Strided layout	$\leftrightarrow$	Insertion
$L = (4, 6) : (2, 64)$	$\leftrightarrow$	
Broadcast layout	$\leftrightarrow$	Projection
$L = (128, 128) : (128, 0)$	$\leftrightarrow$	

Why do we care?

Operations on **Tuple**-morphisms $\leftrightsquigarrow$ Operations on layouts

Example: (Composition)



$L_f = (4,6):(1,8)$

$L_g = (4,2,6):(2,1,8)$

$\rightsquigarrow$ $L_{g\circ f} = (4,6):(2,8)$

Theorem:

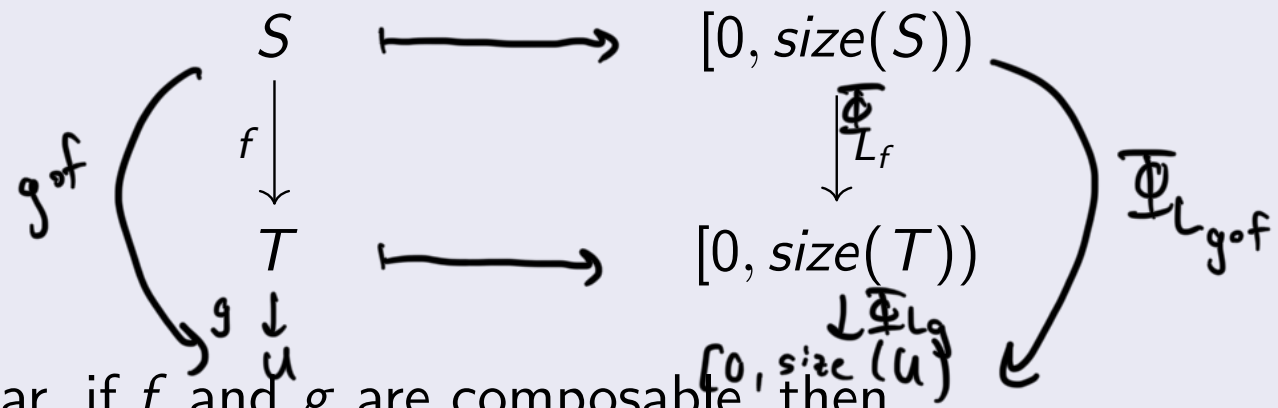
If f and g are composable, then $L_{g\circ f} = L_g \circ L_f$.

Categorical formulation:

Theorem:

There is a realization functor

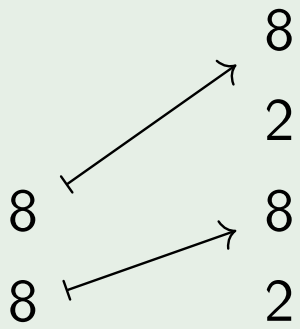
$$\textbf{Tuple} \xrightarrow{\text{Fin}} \textbf{Set}$$



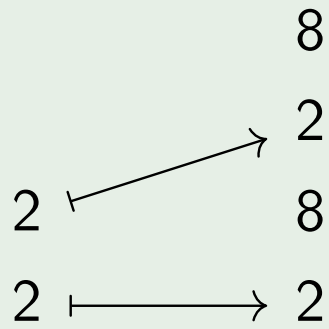
In particular, if f and g are composable, then

$$L_g \circ L_f = L_{g \circ f}.$$

Example: (Complement)



f



f^c

$L_f = (8,8):(2,32)$

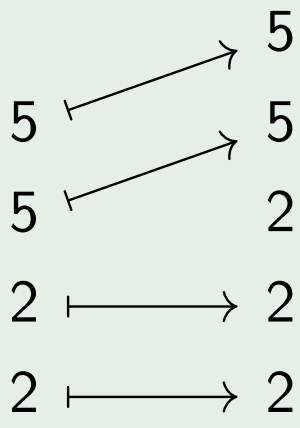
$L_{f^c} = (2,2):(1,16)$

Theorem:

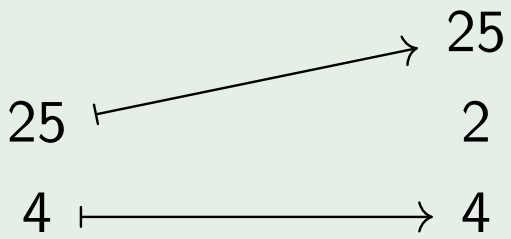
If $f : S \to T$ is an injective **Tuple**-morphism, then

$$L_{f^c} = comp(L_f, size(T)).$$

Example: (Coalesce)



f



$coal(f)$

$L_f = (2,2,5,5):(1,2,8,40)$

$L_{coal(f)} = (4,25):(1,8)$

Theorem:

If $f : S \to T$ is a **Tuple**-morphism, then

$$L_{coal(f)} = coal(L_f).$$

Want to express nested layouts, too.

Notation:

If S is a nested tuple, we write

$$S = (s_1, \ldots, s_m)_P$$

where $(s_1, \ldots, s_m) = S^\flat$ is the flattening of S, and P is the profile of S.

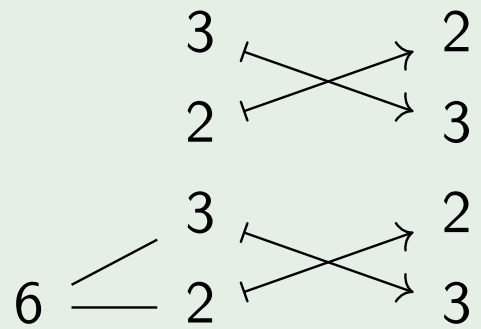
Definition:

In the category **Nest**,

positive integers

1. an object is a nested tuple $S = (s_1, \ldots, s_m)_P$.
2. A morphism $f : (s_1, \ldots, s_m)_P \to (t_1, \ldots, t_n)_Q$ is specified by a function
$$\alpha : \{*, 1, \ldots, m\} \to \{*, 1, \ldots, n\}$$
satisfying
   1. $\alpha(*) = *$,
   2. if $\alpha(i) = \alpha(i') \neq *$, then $i = i'$,
   3. if $\alpha(i) = j$, then $s_i = t_j$.
3. Composition is functional composition of underlying maps.

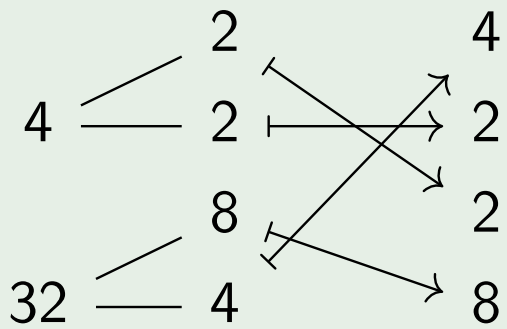
Example:



f

$((2,3),2,3) \xrightarrow{f} (3,2,3,2)$

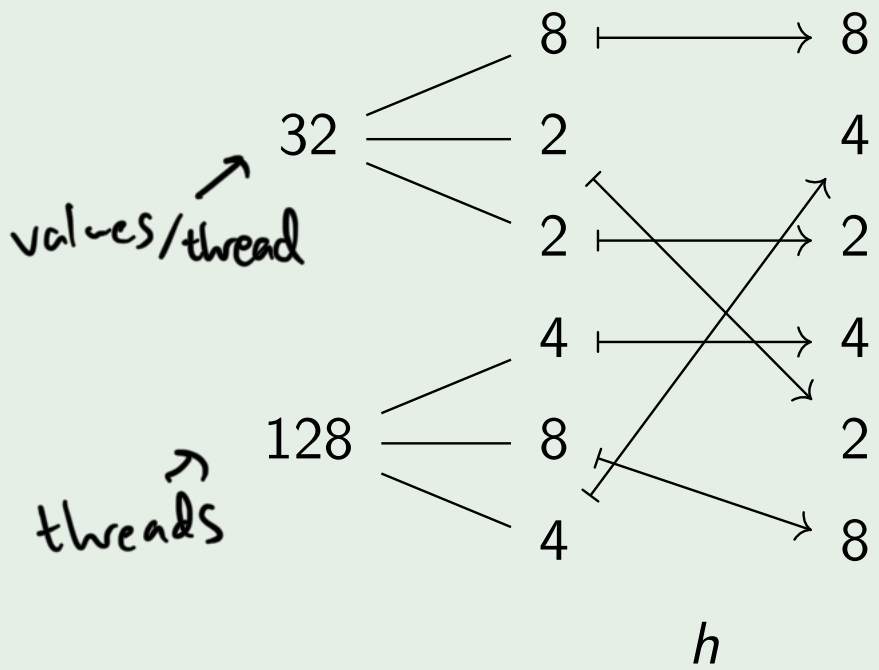
Example: (`mma.sync` accumulator thread-value layout)



g

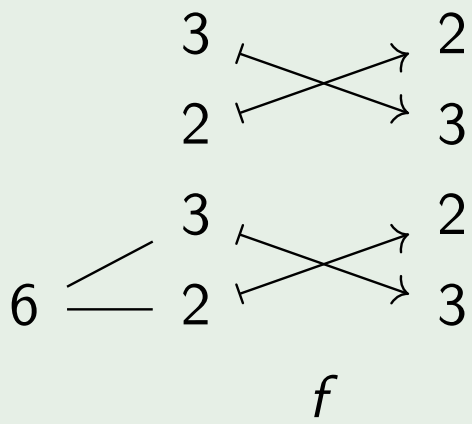
$((4,8),(2,2)) \longrightarrow (8,2,2,4)$

Example: (wgmma accumulator thread-value layout)



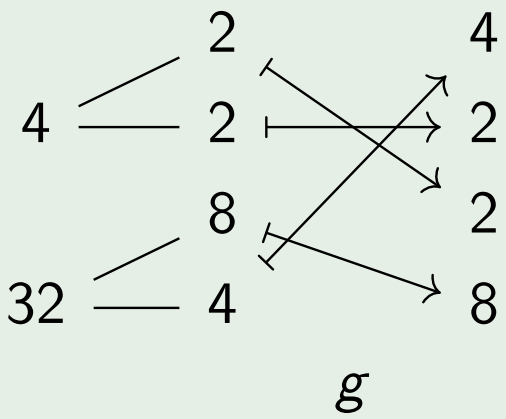
We can use the same formula as before to define the layout encoded by any **Nest**-morphism.

Example:



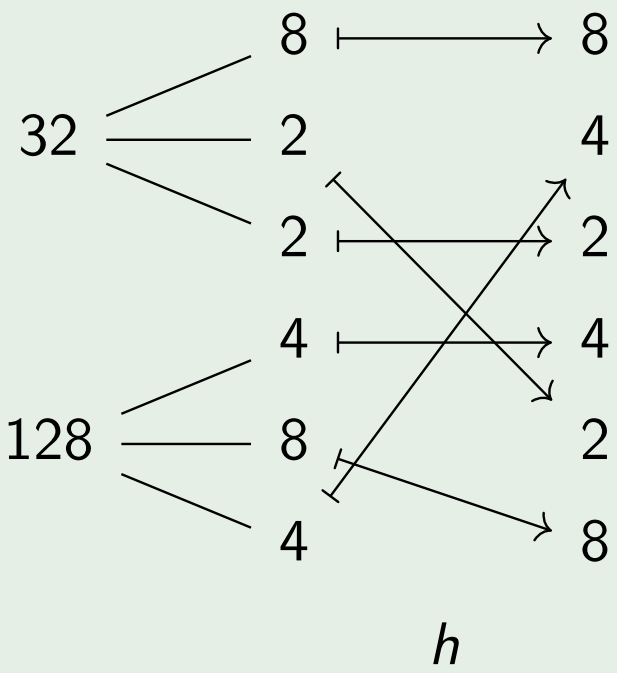
$$L = ((2,3),2,3) : ((3,1),18,6)$$

Example: (`mma.sync` accumulator thread-value layout)



$$L = ((4, 8), (2, 2)) : ((32, 1), (16, 8))$$

Example: (`wgmma` accumulator thread-value layout)



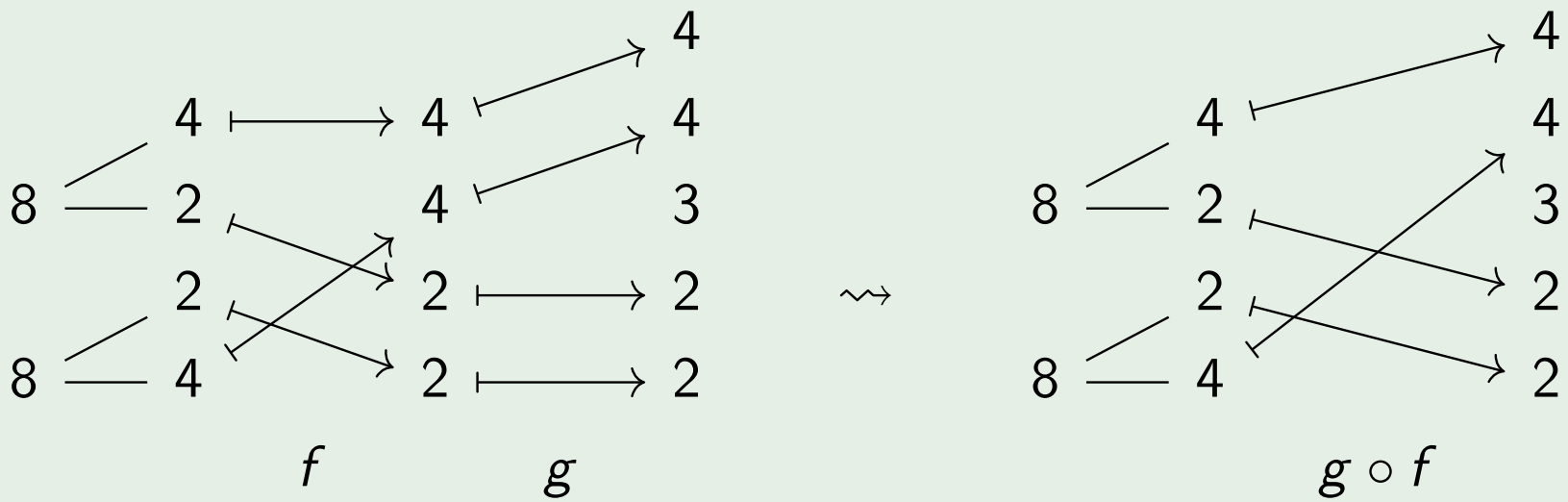
$$L = ((4, 8, 4), (2, 2, 8)) : ((128, 1, 16), (64, 8, 512))$$

Theorem:

1. The layouts which arise from **Nest**-morphisms are exactly the tractable layouts.
2. There is a one-to-one correspondence

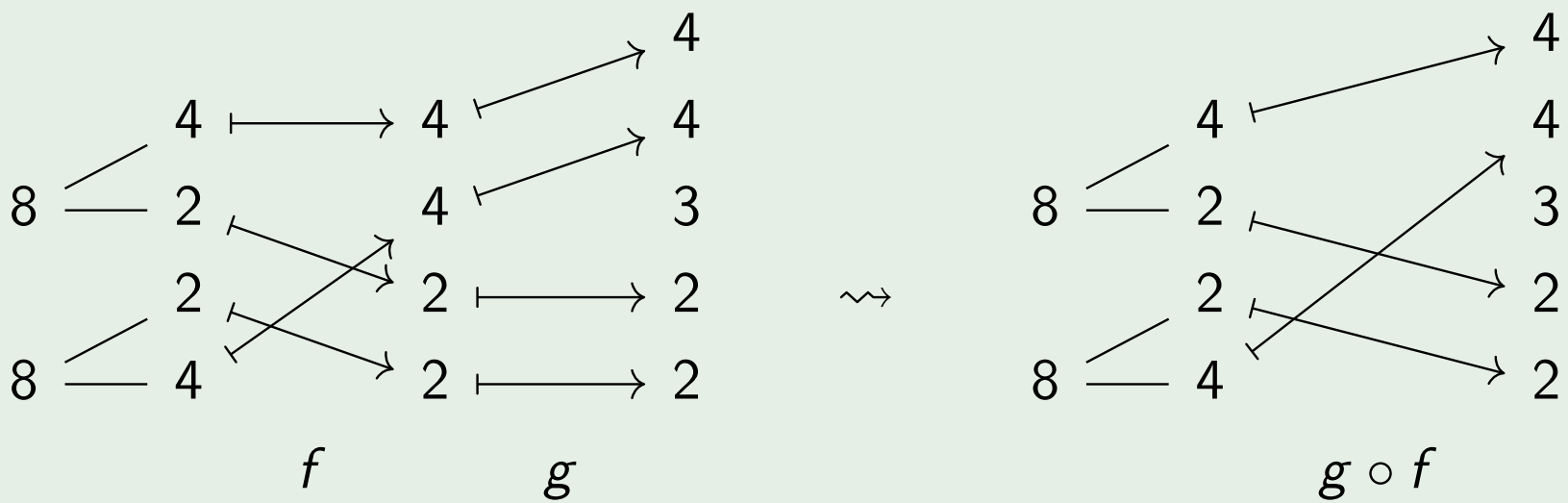
$$\left\{\begin{array}{c}\text{Non-degenerate}\\ \text{tractable layouts}\end{array}\right\} \leftrightarrow \left\{\begin{array}{c}\text{Non-degenerate}\\ \textbf{Nest}\text{-morphisms of}\\ \text{standard form}\end{array}\right\}$$

Example: (Composition)



$$L_f = ((4,2),(2,4)) : ((4,1),(2,16))$$
$$L_g = (2,2,4,4) : (1,2,12,48)$$
$$\rightsquigarrow L_{g\circ f} = ((4,2),(2,4)) : ((12,1),(2,48))$$

Example: (Composition)



Theorem:

If $f : S \to T$ and $g : T \to U$ are non-degenerate **Nest**-morphisms, then

$$L_g \circ L_f = L_{g \circ f}.$$

Categorical formulation:

Theorem:

There is a realization functor

$$\begin{array}{ccc} \textbf{Nest} & \longrightarrow & \textbf{Set} \\ S & & [0, size(S)) \\ \Big\downarrow f & & \Big\downarrow L_f \\ T & & [0, size(T)) \end{array}$$

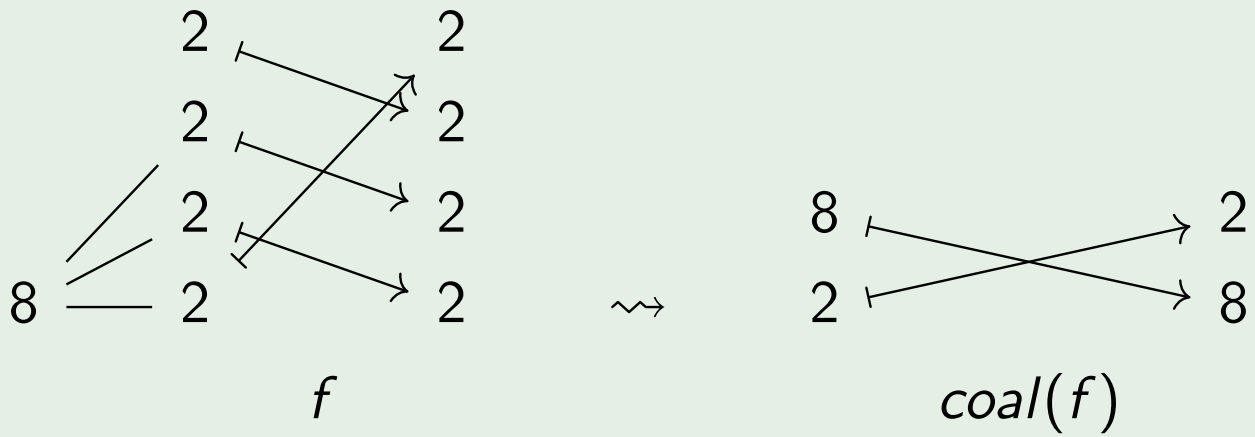
In particular, if f and g are composable, then

$$L_g \circ L_f = L_{g \circ f}.$$

Example: (Coalesce)

$$L_f = ((2,2,2),2) : ((8,1,2),4)$$
$$L_{coal(f)} = (2,8) : (8,1)$$

Example: (Coalesce)

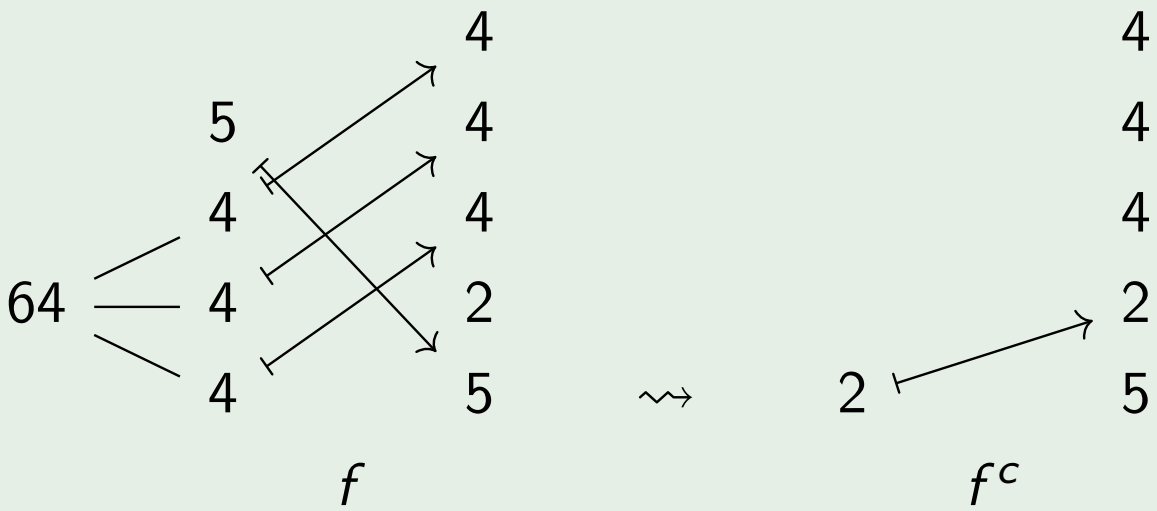


Theorem:

If $f : S \to T$ is a **Nest**-morphism, then

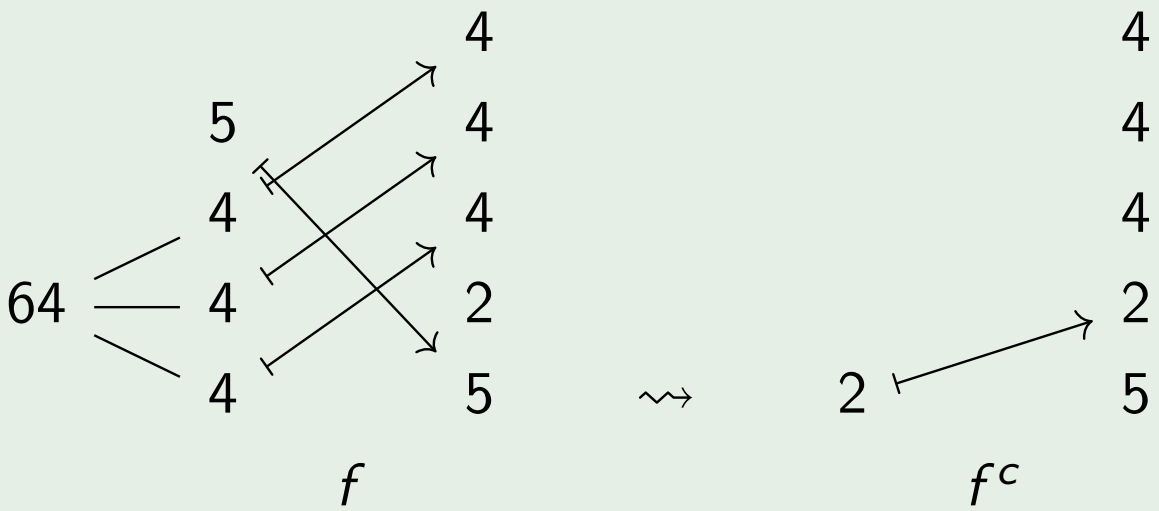
$$L_{coal(f)} = coal(L_f).$$

Example: (Complement)



$$L_f = ((4,4,4),5) : ((10,40,160),1)$$
$$L_{f^c} = (2) : (5)$$

Example: (Complement)

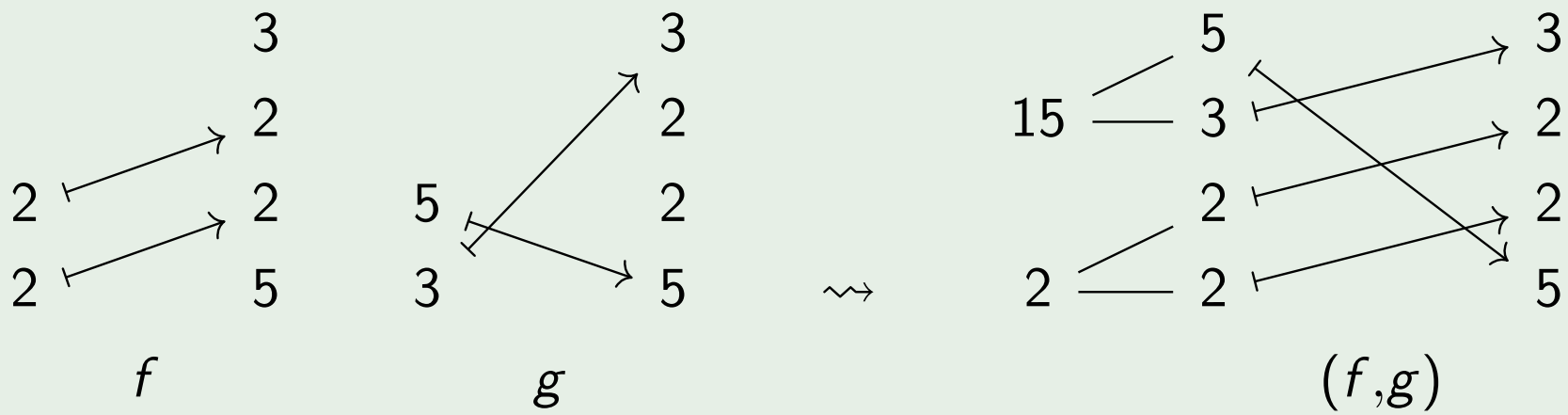


Theorem:

If $f : S \to T$ is an injective **Nest**-morphism and $N = size(T)$, then

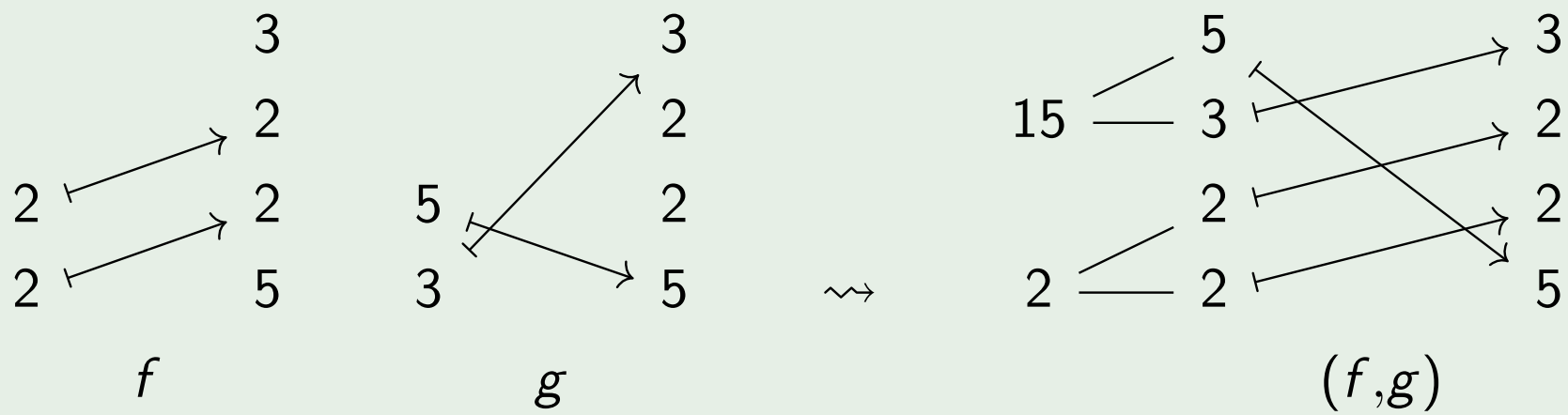
$$coal(L_{f^c}) = comp(L_f, N).$$

Example: (Concatenation)



$$L_f = (2,2) : (5,10)$$
$$L_g = (3,5) : (20,1)$$
$$L_{(f,g)} = ((2,2),(3,5)) : ((5,10),(20,1))$$

Example: (Concatenation)



Theorem:

If $f : S \to T$ and $g : U \to T$ are **Nest**-morphisms with disjoint images, then

$$L_{(f,g)} = (L_f, L_g)$$

When defined, we can set

$$f \oslash g = (f \circ g, f \circ g^c)$$
$$f \otimes g = (f, f^c \circ g)$$

Theorem:

If f and g are non-degenerate **Nest**-morphisms, and g divides f, then

$$coal(L_{f \oslash g}) = coal(L_f \oslash L_g)$$

Theorem:

If f and g are non-degenerate **Nest**-morphisms and f and g are product admissible, then

$$L_{f \otimes g} = L_f \otimes L_g$$

We can use our theory to compute the composition $B \circ A$ of tractable layouts A and B.

Example 1: (Composition Algorithm)

Consider the layouts

$$A = (4, 6) : (6, 1) \qquad B = (12, 4) : (4, 1)$$

Example 1: (Composition Algorithm)

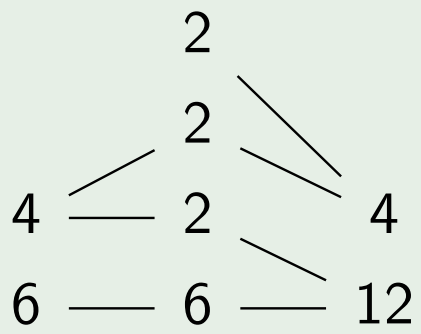
Take their standard representations

$$A = (4, 6) : (6, 1) \qquad B = (12, 4) : (4, 1)$$

factor these compatibly

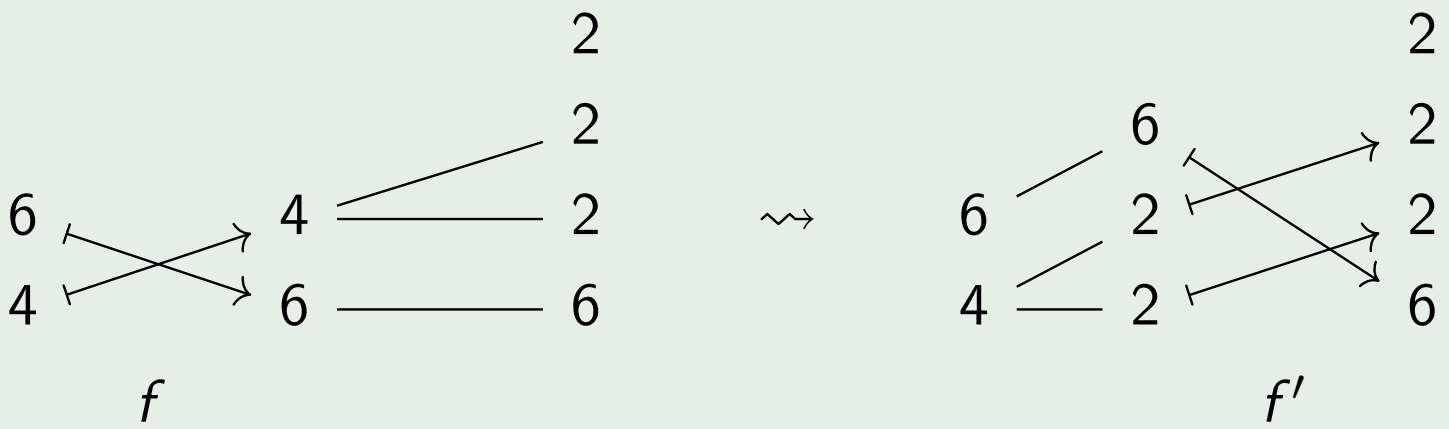
Example 1: (Composition Algorithm)

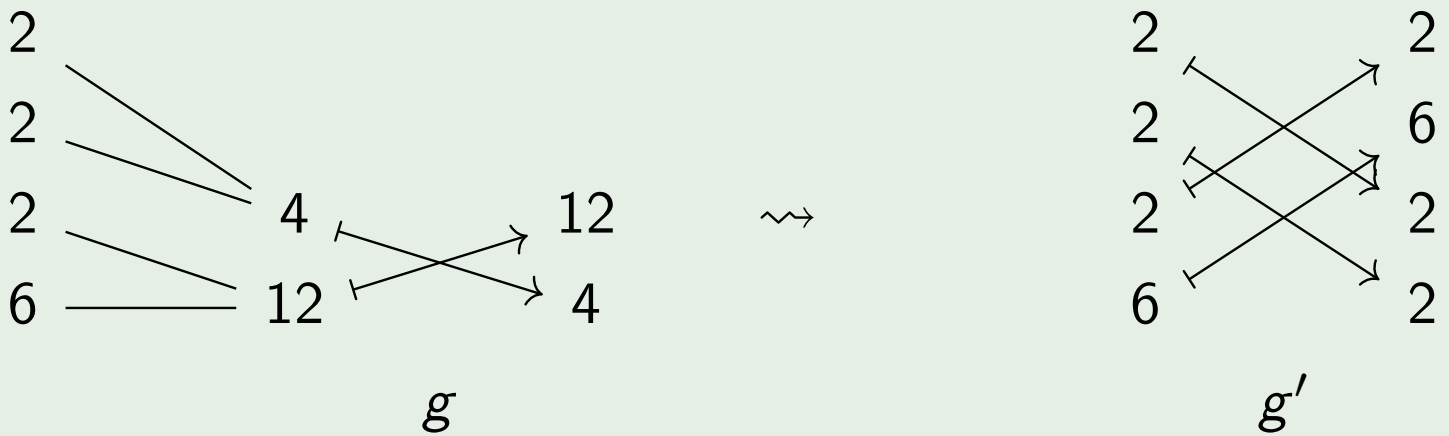
Modify the codomain of f and domain of g:



Example 1: (Composition Algorithm)

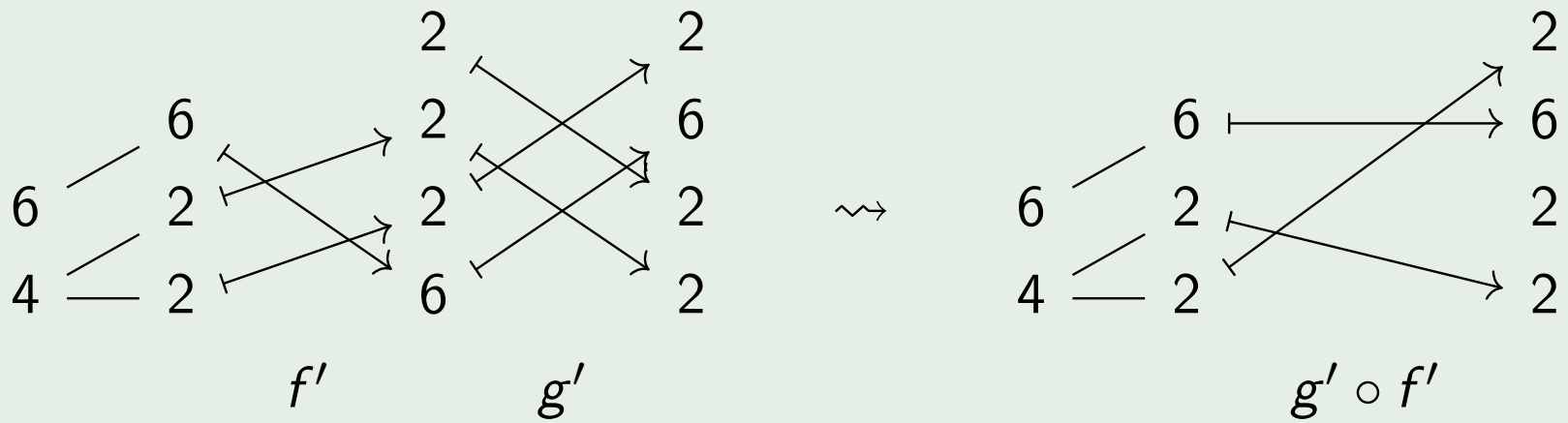
Make a corresponding modification to f and g:





Example 1: (Composition Algorithm)

Compose!



$$B \circ A = L_{g' \circ f'} = ((2,2),6) : ((24,1),4).$$

Example 2: (Composition Algorithm)

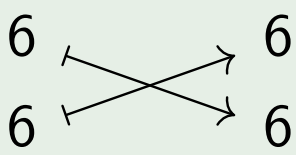
Consider the layouts

$$A = (6, 6) : (6, 1) \qquad B = (12, 3, 6) : (1, 72, 12).$$

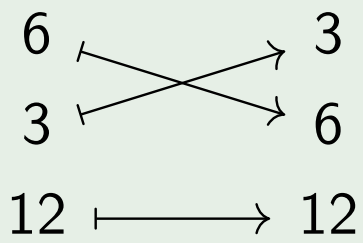
Example 2: (Composition Algorithm)

Take the standard representations

$$A = (6, 6) : (6, 1) \qquad B = (12, 3, 6) : (1, 72, 12).$$



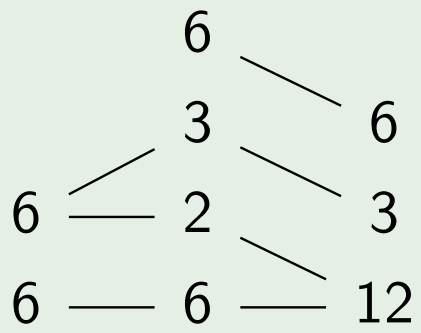
f



g

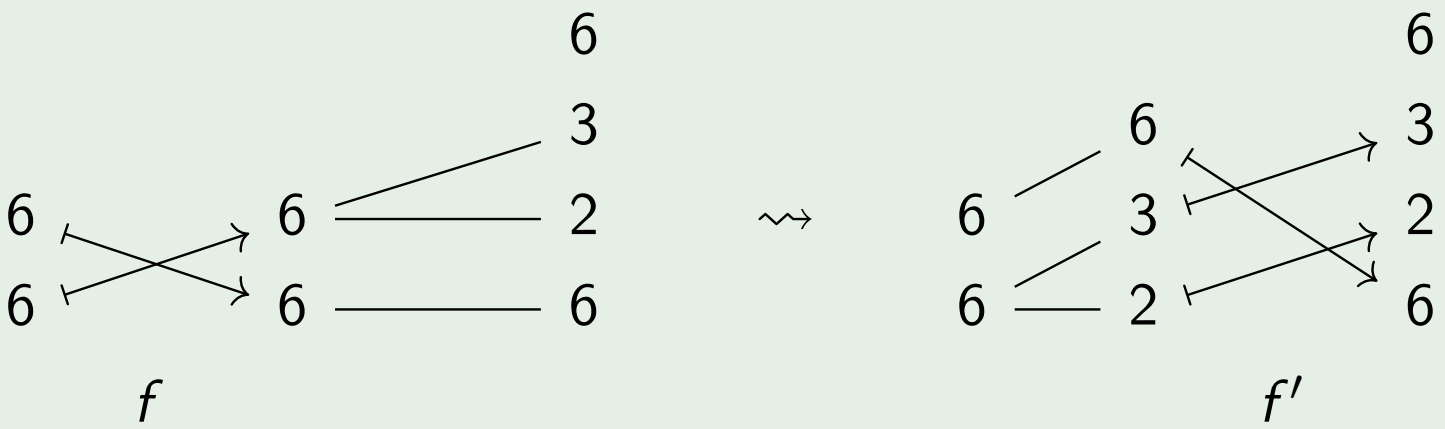
Example 2: (Composition Algorithm)

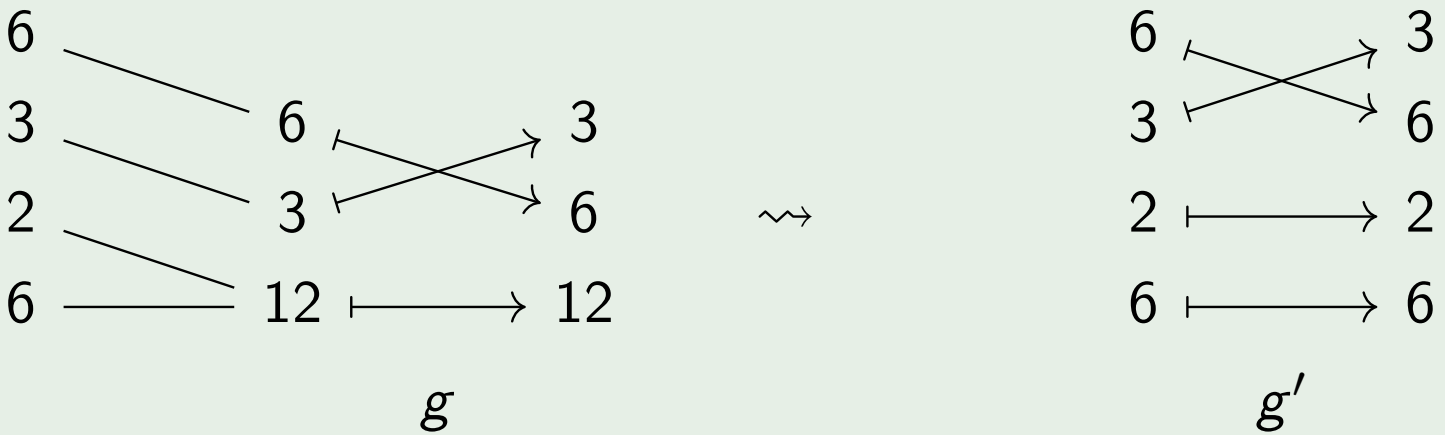
Modify the codomain of f and domain of g



Example 2: (Composition Algorithm)

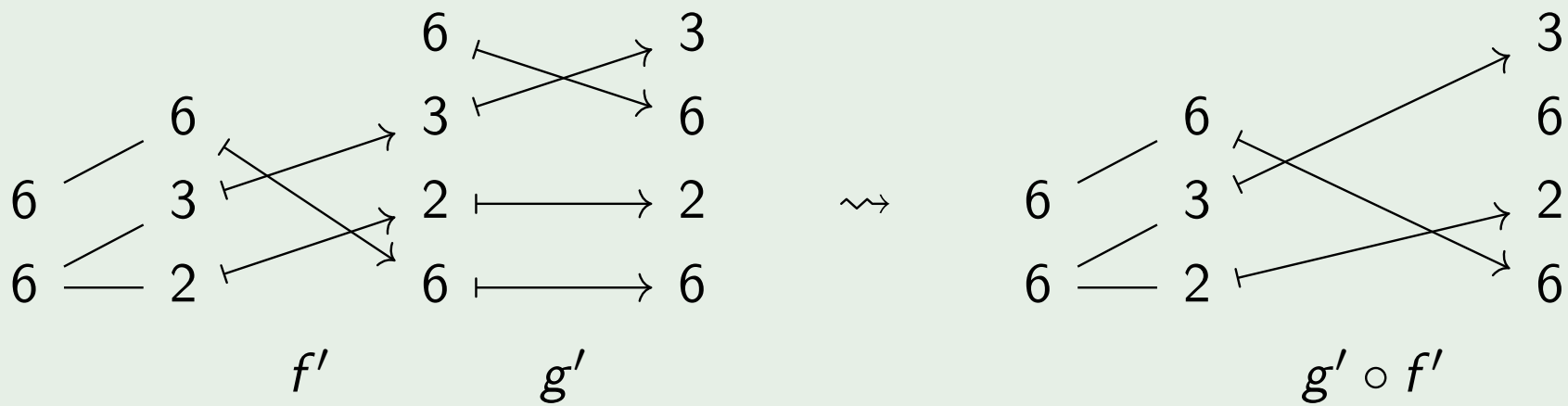
Make a corresponding modification to f and g





Example 2: (Composition Algorithm)

Compose!



$$B \circ A = L_{g' \circ f'} = ((2,3),6) : ((6,72),1).$$

The preceding examples are instances of a more general algorithm.

Composition Algorithm

Input: Tractable layouts A and B.

- Take the standard representations of A and $coal(B)$.

$$f : S \to T, \quad g : U \to V$$

- Compute a mutual refinement (T', U') of T and U,
- Pull back f along $T' \to T$ to obtain $f' : S' \to T' \to U'$,
- Push forward $U' \to U$ along g to obtain $g' : U' \to V'$,

Composition Algorithm (cont.)

- Compose f' and g' to obtain

$$S' \xrightarrow{g' \circ f'} V'.$$

- Take the encoded layout $C = L_{g' \circ f'}$.

Output: Layout $\boxed{C = B \circ A.}$

Remark:

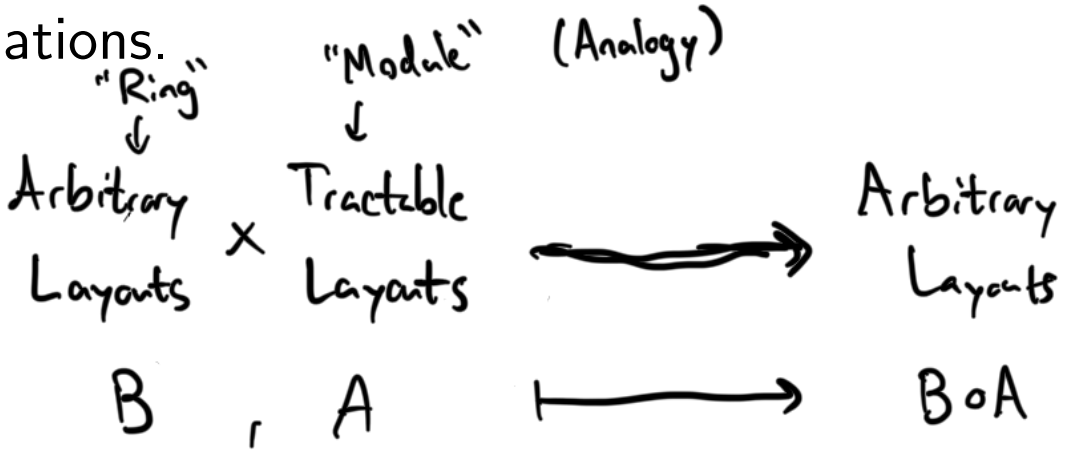
With appropriate modifications, we can use the same algorithm to compute $B \circ A$ when A is tractable and B is *arbitrary*.

Theorem: (Composition algorithm)

The algorithm described above is well-defined, and is guaranteed to produce the composition $B \circ A$ of tractable layouts A and B, if it exists.

Future directions:

- Span category formulation, further connections to operad theory.
- Characterize algebraic relations satisfied by logical division, logical product, etc.
- Interaction with other layout abstractions: swizzles, linear layouts, integer set relations.



References and related work:

- *CuTe Layout Representation and Algebra* (Cecka) `https://arxiv.org/abs/2603.02298`
- *Categorical foundations for CuTe layouts* (Colfax) `https://arxiv.org/abs/2601.05972` (blog post: `https://research.colfax-intl.com/categorical-foundations-for-cute-layouts/`)
- *Modeling Layout Abstractions using Integer Set Relations* (Bhaskaracharya et. al.) `https://arxiv.org/html/2511.10374v1`
- *Linear Layouts: Robust Code Generation of Efficient Tensor Computation Using* $\mathbb{F}_2$ (Zhou et. al.) `https://arxiv.org/abs/2505.23819`
- CuTe documentation: (`https://docs.nvidia.com/cutlass/latest/media/docs/cpp/cute/index.html`)
- Cris Cecka's GPU mode talk (`https://www.youtube.com/watch?v=ufa4pmBOBT8`)

Thanks for listening!