6.5930/1

Hardware Architectures for Deep Learning

# Calculating Data Motion

March 18, 2026

Joel Emer and Vivienne Sze

Acknowledgements: Angshuman Parashar/Michael Gilbert

Massachusetts Institute of Technology
Electrical Engineering & Computer Science

# 1-D Convolution – Output Stationary

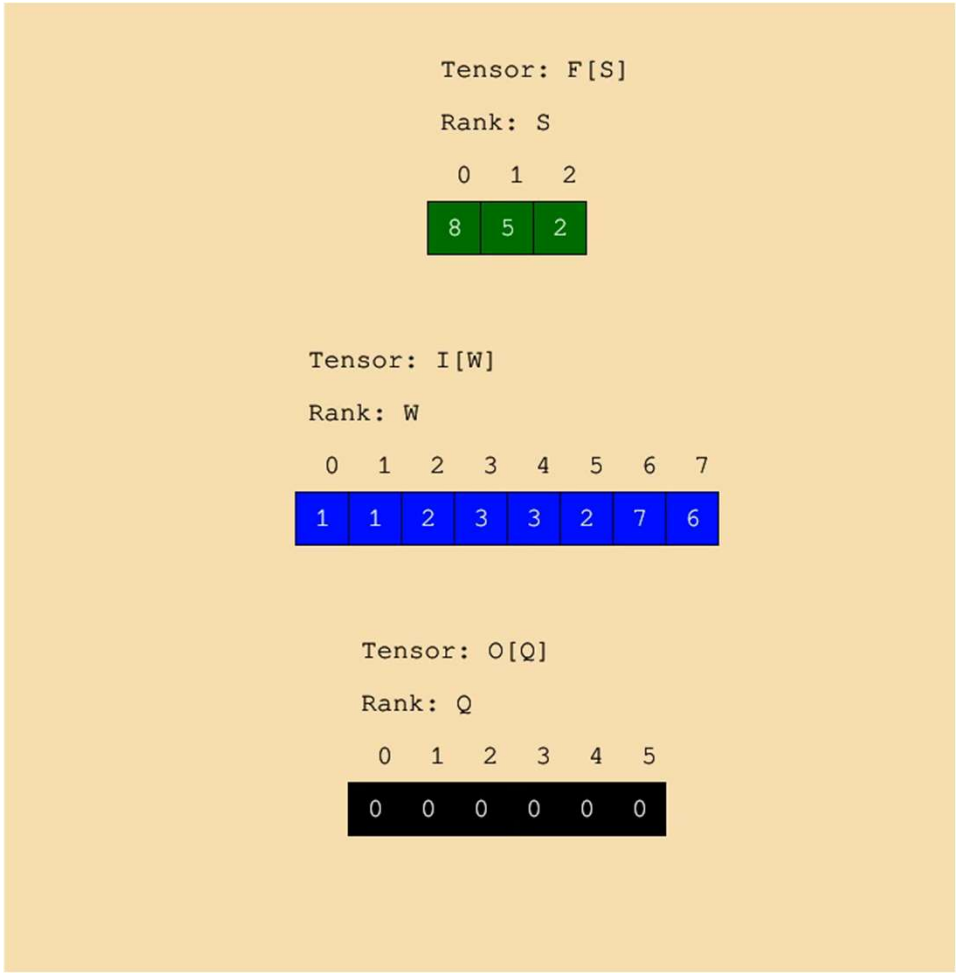

$$O_q = I_{q+s} \times F_s$$

Traversal order (fastest to slowest): S, Q

```
int i[W];      # Input activations
int f[S];      # Filter weights
int o[Q];      # Output activations

for q in [0, Q):
       for s in (0, S):
          o[q] += i[q+s]*f[s]
```

# Output Stationary



# An ISL Set

Input:

```
X = 5
Y = 3

set1 = isl.Set(f'{{ SetXY[x, y] :
       0 <= x <= {X-1} and
       0 <= y <= {Y-1} }}')

printSet(set1)
```

Space Name

Coordinates

Coordinate specification/ constraints

Output:

```
set1 = { SetXY[x, y] : 0 <= x <= 4 and 0 <= y <= 2 }
```

set1

```
{ SetXY[0, 0] }
{ SetXY[1, 0] }
{ SetXY[2, 0] }
{ SetXY[3, 0] }
{ SetXY[4, 0] }
{ SetXY[0, 1] }
{ SetXY[1, 1] }
{ SetXY[2, 1] }
{ SetXY[3, 1] }
{ SetXY[4, 1] }
{ SetXY[0, 2] }
{ SetXY[1, 2] }
{ SetXY[2, 2] }
{ SetXY[3, 2] }
{ SetXY[4, 2] }
```

# An ISL Set

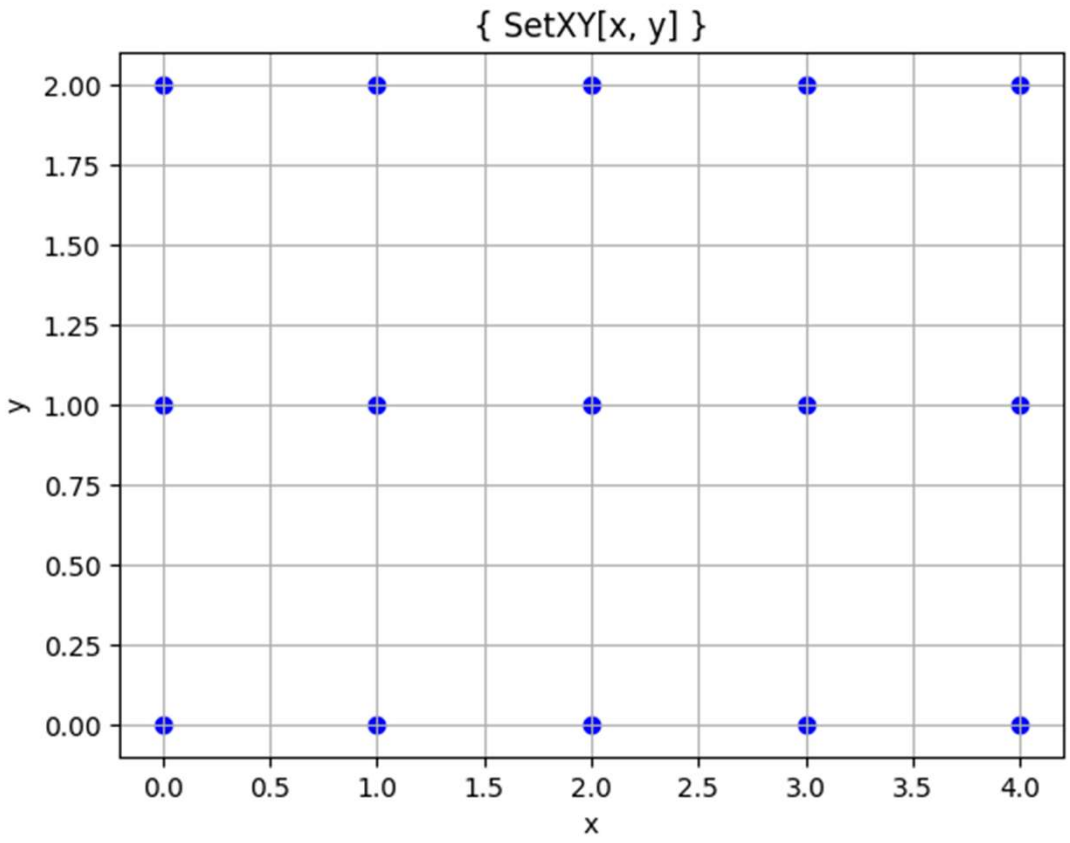
Input:

```
X = 5
Y = 3

set1 = isl.Set(f'{{ SetXY[x, y] :
       0 <= x <= {X-1} and
       0 <= y <= {Y-1} }}')

plotSet(set1)
```

set1 = { SetXY[x, y] : 0 <= x <= 4 and 0 <= y <= 2 }



# A different ISL Set

```
set1 = isl.Set(f'{{ SetXY[x, y] :
       0 <= x <= {X-1} and
       0 <= y <= {Y-1} }}'
```

Input:

```
set2 = isl.Set(f'{{ SetWX[x, y] :
       0 <= x <= {X-1} and
       0 <= y <= {Y-1} }}')
```

Output:

SetXY = { SetWX[x, y] : 0 <= x <= 4 and 0 <= y <= 2 }

set2

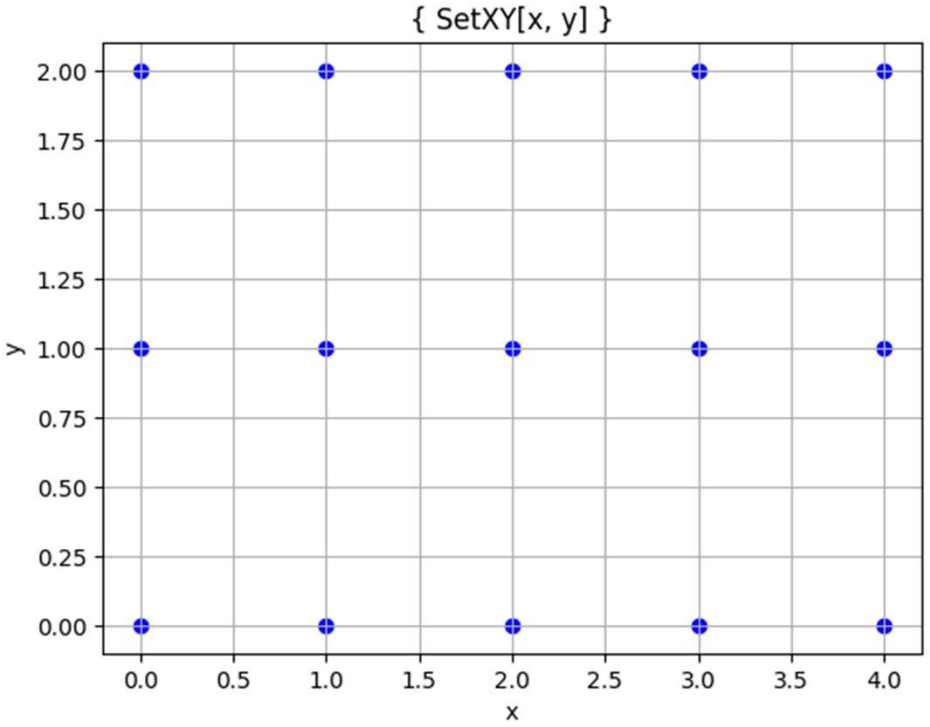
{ SetWX[0, 0] }
{ SetWX[1, 0] }
{ SetWX[2, 0] }
{ SetWX[3, 0] }
{ SetWX[4, 0] }
{ SetWX[0, 1] }
{ SetWX[1, 1] }
{ SetWX[2, 1] }
{ SetWX[3, 1] }
{ SetWX[4, 1] }
{ SetWX[0, 2] }
{ SetWX[1, 2] }
{ SetWX[2, 2] }
{ SetWX[3, 2] }
{ SetWX[4, 2] }

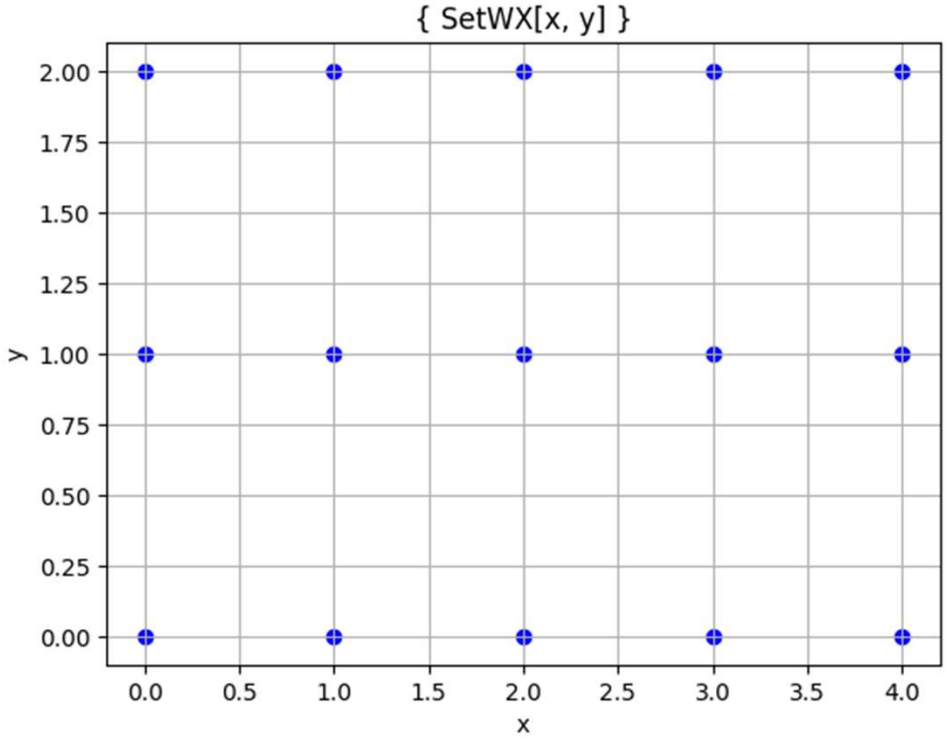
Set has a different “spacename”

# A different ISL Set

set1 = { SetXY[x, y] : 0 <= x <= 4 and 0 <= y <= 2 }

set2 = { SetWX[x, y] : 0 <= x <= 4 and 0 <= y <= 2 }




Not the same!

# Different Coordinate Order

```
set1 = isl.Set(f'{{ SetXY[x, y] :
       0 <= x <= {X-1} and
       0 <= y <= {Y-1} }}'
```

Input:

```
set3 = isl.Set(f'{{ SetXY[y, x] :
       0 <= x <= {X-1} and
       0 <= y <= {Y-1} }}')
```

Output:

```
set3 = { SetXY[y, x] : 0 <= y <= 2 and 0 <= x <= 4 }
```

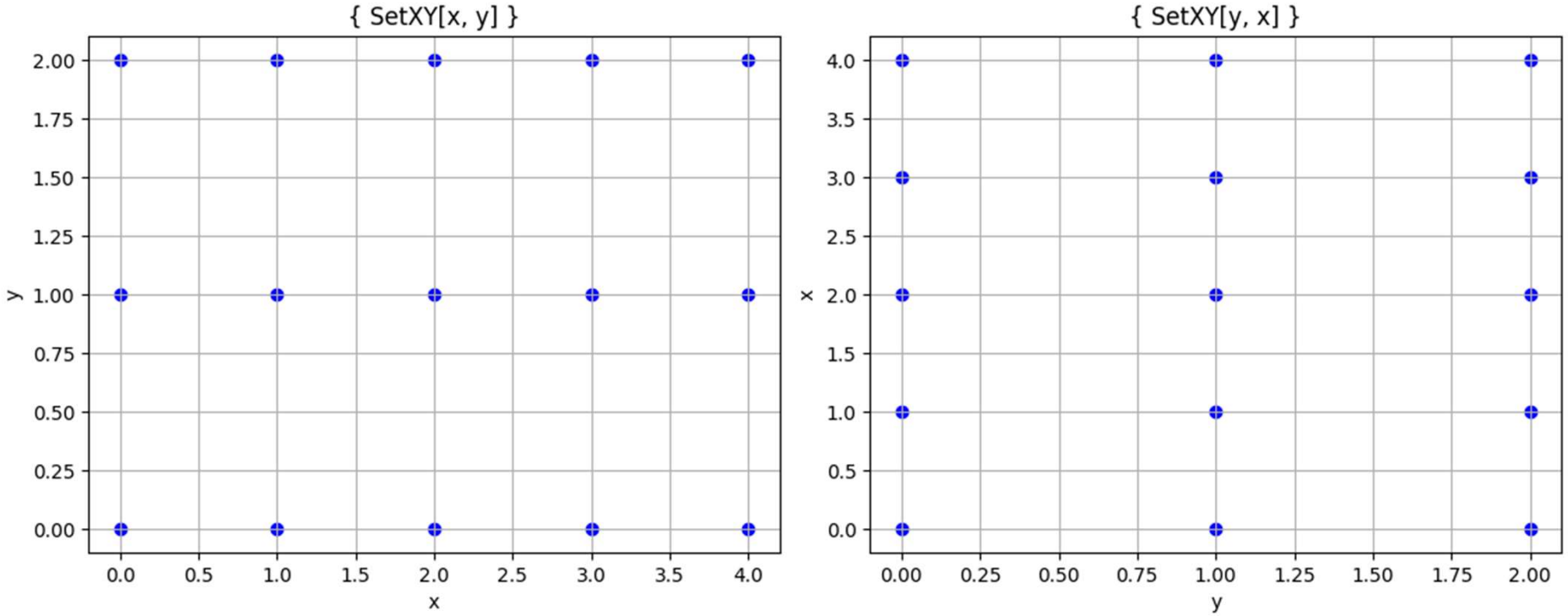
set3

```
{ SetXY[0, 0] }
{ SetXY[0, 1] }
{ SetXY[0, 2] }
{ SetXY[0, 3] }
{ SetXY[0, 4] }
{ SetXY[1, 0] }
{ SetXY[1, 1] }
{ SetXY[1, 2] }
{ SetXY[1, 3] }
{ SetXY[1, 4] }
{ SetXY[2, 0] }
{ SetXY[2, 1] }
{ SetXY[2, 2] }
{ SetXY[2, 3] }
{ SetXY[2, 4] }
```

# ISL coordinate order matters

set1 = { SetXY[x, y] : 0 <= x <= 4 and 0 <= y <= 2 }

set3 = { SetXY[y, x] : 0 <= y <= 2 and 0 <= x <= 4 }




Not the same!

# Different Constraints

Input:

```
set1 = isl.Set(f'{{ SetXY[x, y] :
       0 <= x <= {X-1} and
       0 <= y <= {Y-1} }}'
```

Output:

```
set1 = { SetXY[x, y] : 0 <= x <= 4 and 0 <= y <= 2 }
```

Input:

```
set4 = isl.Set(f'{{ SetXY[x, y] :
       0 <= x < {X} and
       0 <= y < {Y} }}')
```

Output:

```
set4 = { SetXY[x, y] : 0 <= x <= 4 and 0 <= y <= 2 }
```

Same

set1

```
{ SetXY[0, 0] }
{ SetXY[1, 0] }
{ SetXY[2, 0] }
{ SetXY[3, 0] }
{ SetXY[4, 0] }
{ SetXY[0, 1] }
{ SetXY[1, 1] }
{ SetXY[2, 1] }
{ SetXY[3, 1] }
{ SetXY[4, 1] }
{ SetXY[0, 2] }
{ SetXY[1, 2] }
{ SetXY[2, 2] }
{ SetXY[3, 2] }
{ SetXY[4, 2] }
```

set4

```
{ SetXY[0, 0] }
{ SetXY[1, 0] }
{ SetXY[2, 0] }
{ SetXY[3, 0] }
{ SetXY[4, 0] }
{ SetXY[0, 1] }
{ SetXY[1, 1] }
{ SetXY[2, 1] }
{ SetXY[3, 1] }
{ SetXY[4, 1] }
{ SetXY[0, 2] }
{ SetXY[1, 2] }
{ SetXY[2, 2] }
{ SetXY[3, 2] }
{ SetXY[4, 2] }
```

set1 and set4 are the same

# More Complex Constraints

```
set1 = isl.Set(f'{{ SetXY[x, y] :
       0 <= x <= {X-1} and
       0 <= y <= {Y-1} }}’
```

Input:

```
set5 = isl.Set(f'{{ SetXY[x, y] :
       0 <= x <= {X-1} and
       0 <= y <= x }}')
```

Output:

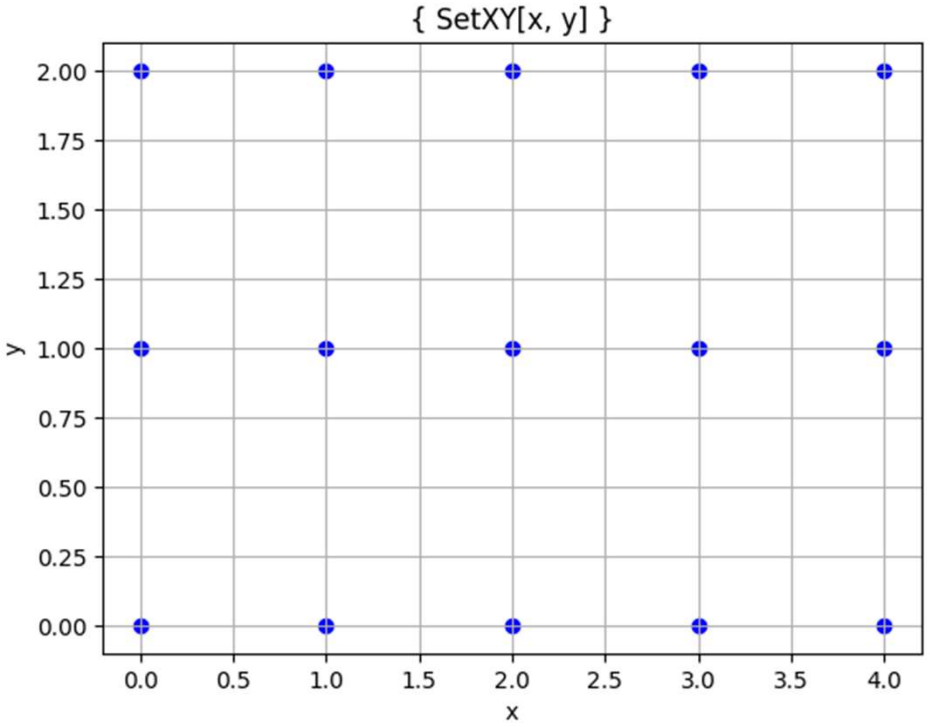
```
set5 = { SetXY[x, y] : 0 <= x <= 4 and 0 <= y <= x }
```

set5

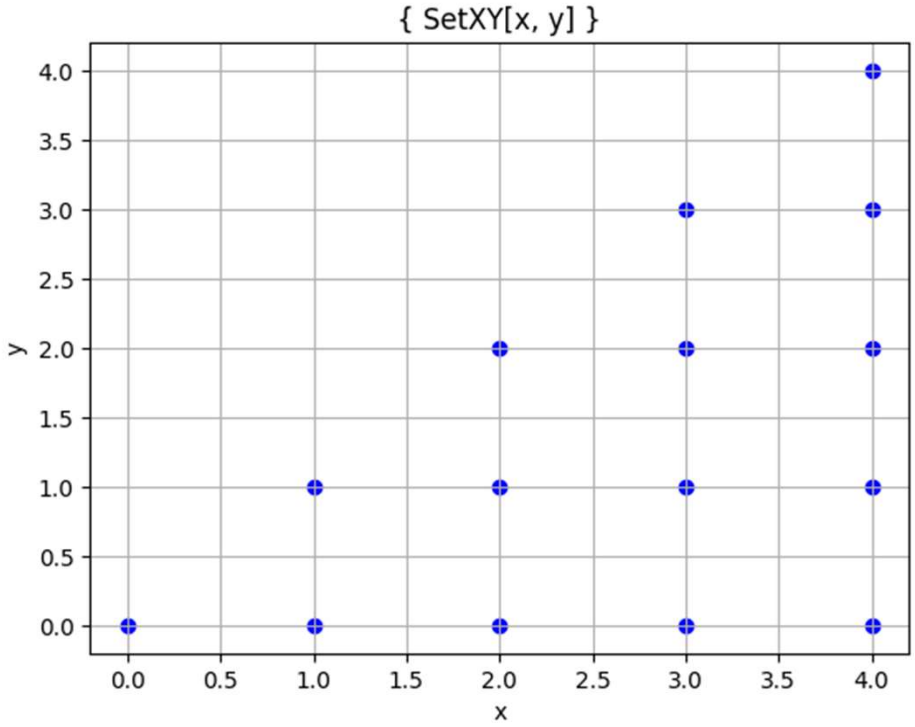
{ SetXY[0, 0] }
{ SetXY[1, 0] }
{ SetXY[1, 1] }
{ SetXY[2, 0] }
{ SetXY[2, 1] }
{ SetXY[2, 2] }
{ SetXY[3, 0] }
{ SetXY[3, 1] }
{ SetXY[3, 2] }
{ SetXY[3, 3] }
{ SetXY[4, 0] }
{ SetXY[4, 1] }
{ SetXY[4, 2] }
{ SetXY[4, 3] }
{ SetXY[4, 4] }

# More Complex Constraints

set1 = { SetXY[x, y] : 0 <= x <= 4 and 0 <= y <= 2 }




set5 = { SetXY[x, y] : 0 <= x <= 4 and 0 <= y <= x }



# Defining an Iteration Space

Input:

```
X = 5
Y = 3

ispaceXY = isl.Set(f'{{ IterationSpace[x,y] :
                  0 <= x < {X} and
                  0 <= y < {Y} }}')
```

Output:

```
ispaceXY = { IterationSpace[x, y] : 0 <= x <= 4 and 0 <= y <= 2 }
```

ispaceXY

```
{ IterationSpace[0, 0] }
{ IterationSpace[1, 0] }
{ IterationSpace[2, 0] }
{ IterationSpace[3, 0] }
{ IterationSpace[4, 0] }
{ IterationSpace[0, 1] }
{ IterationSpace[1, 1] }
{ IterationSpace[2, 1] }
{ IterationSpace[3, 1] }
{ IterationSpace[4, 1] }
{ IterationSpace[0, 2] }
{ IterationSpace[1, 2] }
{ IterationSpace[2, 2] }
{ IterationSpace[3, 2] }
{ IterationSpace[4, 2] }
```

# Iteration Space

$$O_q = I_{q+s} \times F_s$$

Tensor: Ispace[S, Q]

Rank: Q

| Rank: S | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 1 | 1 | 1 | 1 |

What is the iteration space? Cross product of Q x S

# Defining an Iteration Space

```
X = 5
Y = 3

ispaceXY = isl.Set(f'{{ IterationSpace[x,y] :
                  0 <= x < {X} and
                  0 <= y < {Y} }}')
```

Input:

```
Q = 5
S = 3

ispace = isl.Set(f'{{ IterationSpace[q,s] :
                0 <= q < {Q} and
                0 <= s < {S} }}')
```

Output:

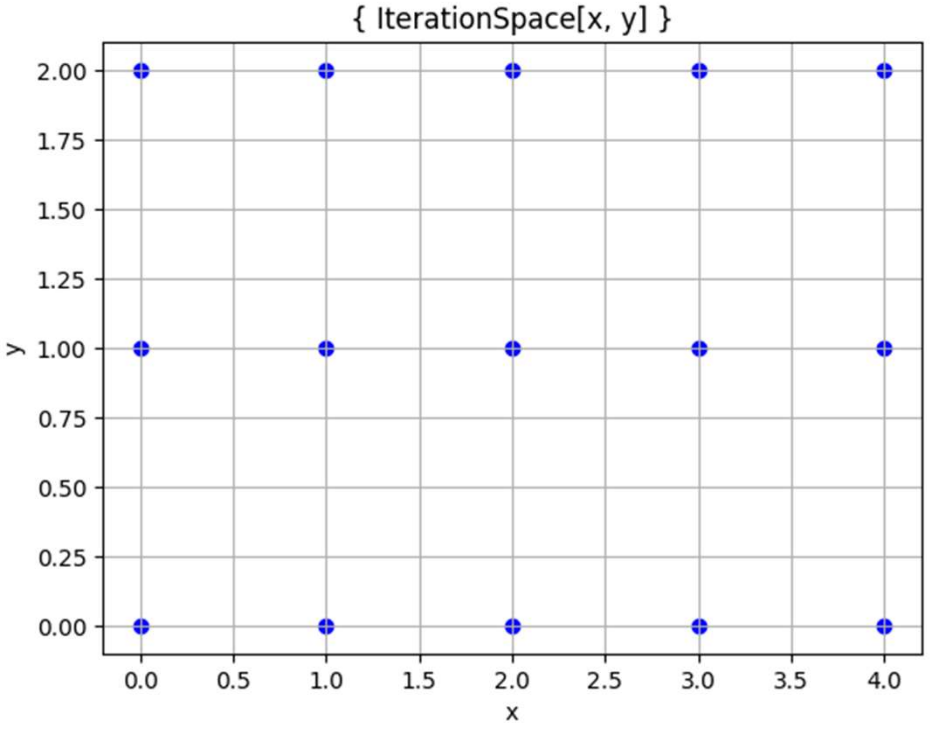
```
ispace  = { IterationSpace[q, s] : 0 <= q <= 4 and 0 <= s <= 2 }
```

ispace

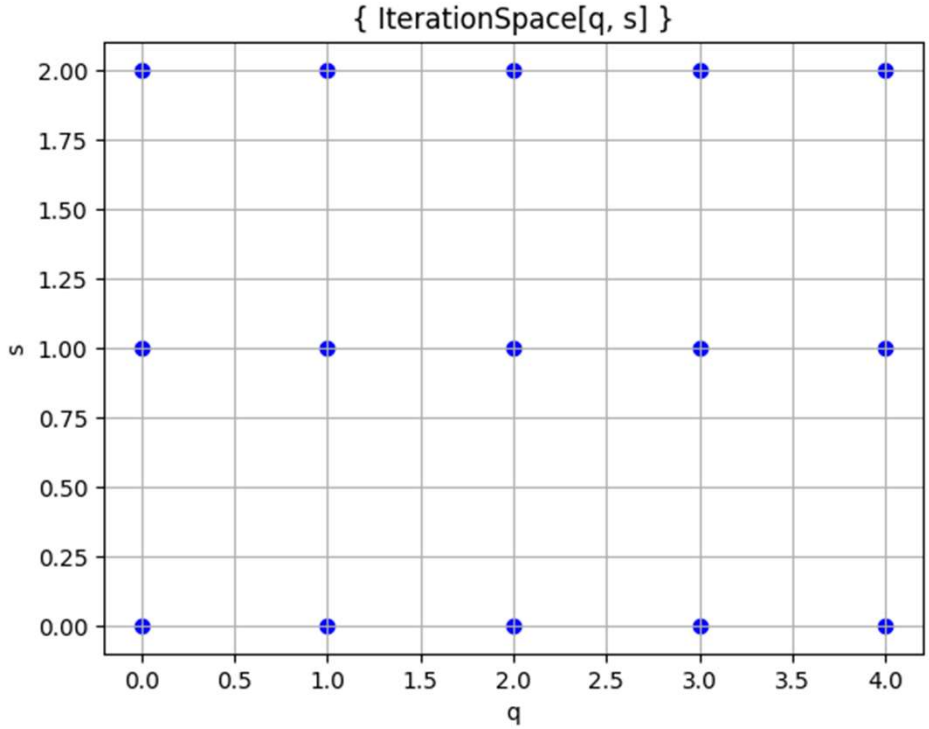
```
{ IterationSpace[0, 0] }
{ IterationSpace[1, 0] }
{ IterationSpace[2, 0] }
{ IterationSpace[3, 0] }
{ IterationSpace[4, 0] }
{ IterationSpace[0, 1] }
{ IterationSpace[1, 1] }
{ IterationSpace[2, 1] }
{ IterationSpace[3, 1] }
{ IterationSpace[4, 1] }
{ IterationSpace[0, 2] }
{ IterationSpace[1, 2] }
{ IterationSpace[2, 2] }
{ IterationSpace[3, 2] }
{ IterationSpace[4, 2] }
```

# Coordinate Naming

{ IterationSpace[x, y] : 0 <= x <= 4 and 0 <= y <= 2 }




{ IterationSpace[q, s] : 0 <= q <= 4 and 0 <= s <= 2 }




Spaces are the same!

# Maps – Relations Between Two Sets

```
unbounded_map = isl.Map(f'{{
                SetXY[x,y] ->
                SetXY_range[x1,y1] :
                x1=x+1 and
                y1=y+10}}')
```

Domain

Range

Relation

```
bounded_map = unbounded_map.
                intersect_domain(set1)
```

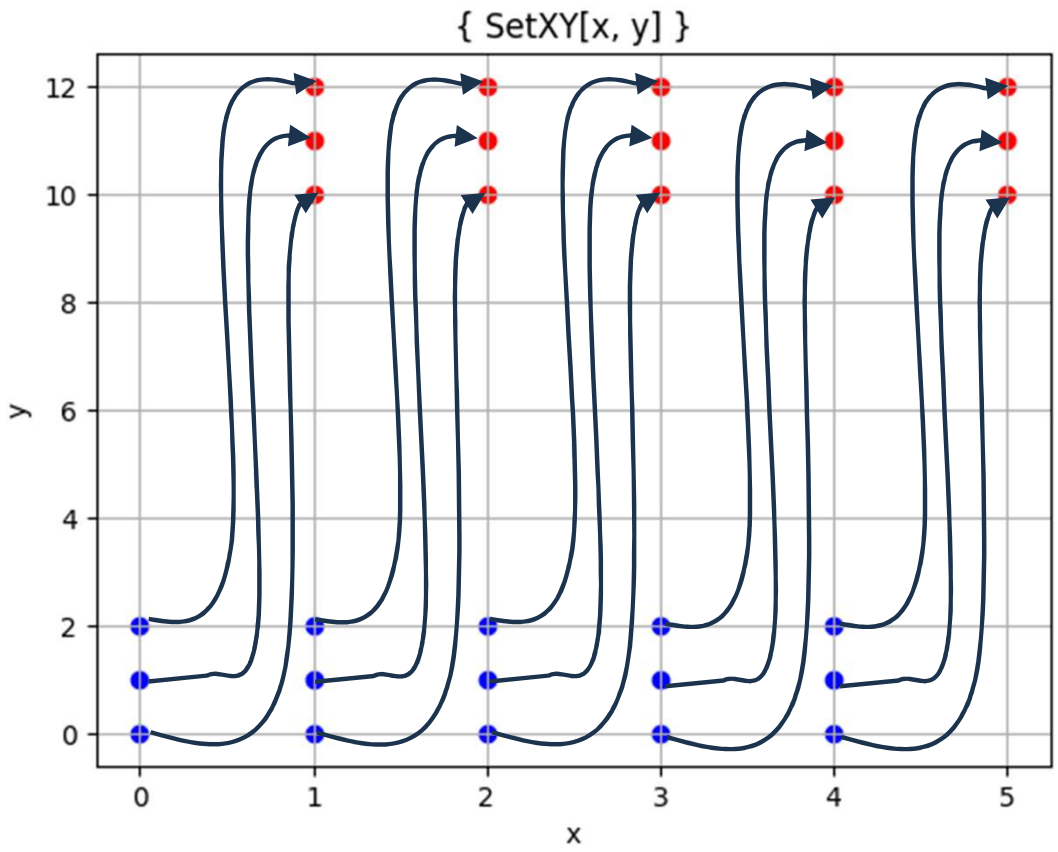
bounded_map

```
{ [SetXY[x = 0, y = 0] -> SetXY_range[x1 = 1, y1 = 10]] }
{ [SetXY[x = 0, y = 1] -> SetXY_range[x1 = 1, y1 = 11]] }
{ [SetXY[x = 0, y = 2] -> SetXY_range[x1 = 1, y1 = 12]] }
{ [SetXY[x = 1, y = 0] -> SetXY_range[x1 = 2, y1 = 10]] }
{ [SetXY[x = 1, y = 1] -> SetXY_range[x1 = 2, y1 = 11]] }
{ [SetXY[x = 1, y = 2] -> SetXY_range[x1 = 2, y1 = 12]] }
{ [SetXY[x = 2, y = 0] -> SetXY_range[x1 = 3, y1 = 10]] }
{ [SetXY[x = 2, y = 1] -> SetXY_range[x1 = 3, y1 = 11]] }
{ [SetXY[x = 2, y = 2] -> SetXY_range[x1 = 3, y1 = 12]] }
{ [SetXY[x = 3, y = 0] -> SetXY_range[x1 = 4, y1 = 10]] }
{ [SetXY[x = 3, y = 1] -> SetXY_range[x1 = 4, y1 = 11]] }
{ [SetXY[x = 3, y = 2] -> SetXY_range[x1 = 4, y1 = 12]] }
{ [SetXY[x = 4, y = 0] -> SetXY_range[x1 = 5, y1 = 10]] }
{ [SetXY[x = 4, y = 1] -> SetXY_range[x1 = 5, y1 = 11]] }
{ [SetXY[x = 4, y = 2] -> SetXY_range[x1 = 5, y1 = 12]] }
```

# Maps

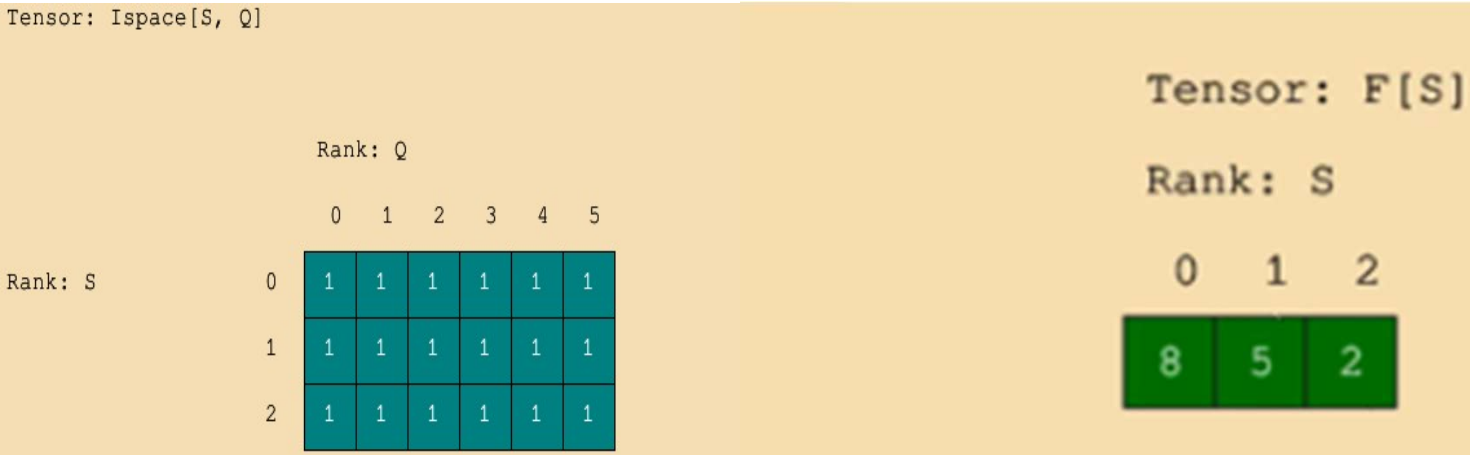
```
unbounded_map = isl.Map(f'{{
                        SetXY[x,y] ->
                        SetXY_range[x1,y1] :
                        x1=x+1 and
                        y1=y+10}}')
```

```
bounded_map = unbounded_map.
                    intersect_domain(set1)
```



# Iteration Space – Projections to Data

$$O_q = I_{q+s} \times F_s$$




Given a point (s,q) in the iteration space, what weight does it map to?

$F_s$

# Weight Projection

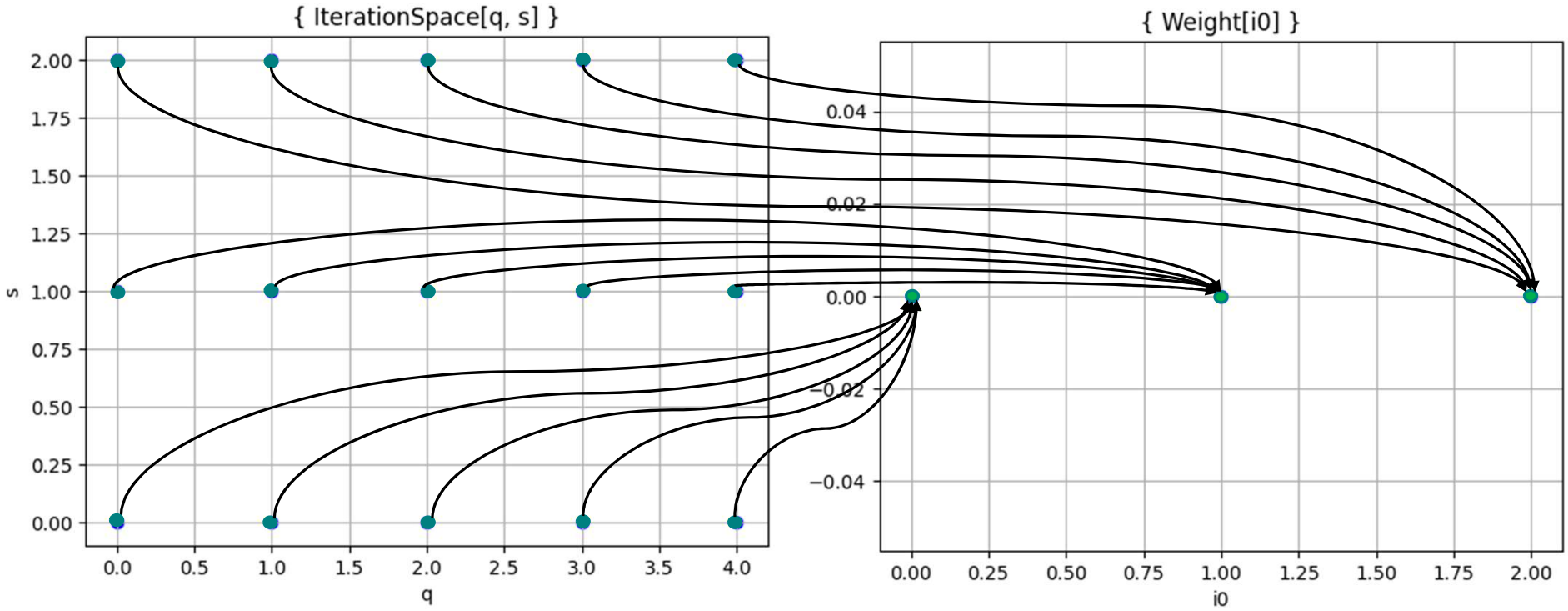
$$O_q = I_{q+s} \times F_s$$

```
is2weight = isl.Map("{ IterationSpace[q,s] -> Weight[s] }")
```

```
{ [IterationSpace[q = 0, s = 0] -> Weight[0]] }
{ [IterationSpace[q = 0, s = 1] -> Weight[1]] }
{ [IterationSpace[q = 0, s = 2] -> Weight[2]] }
{ [IterationSpace[q = 1, s = 0] -> Weight[0]] }
{ [IterationSpace[q = 1, s = 1] -> Weight[1]] }
{ [IterationSpace[q = 1, s = 2] -> Weight[2]] }
{ [IterationSpace[q = 2, s = 0] -> Weight[0]] }
{ [IterationSpace[q = 2, s = 1] -> Weight[1]] }
{ [IterationSpace[q = 2, s = 2] -> Weight[2]] }
{ [IterationSpace[q = 3, s = 0] -> Weight[0]] }
{ [IterationSpace[q = 3, s = 1] -> Weight[1]] }
{ [IterationSpace[q = 3, s = 2] -> Weight[2]] }
{ [IterationSpace[q = 4, s = 0] -> Weight[0]] }
{ [IterationSpace[q = 4, s = 1] -> Weight[1]] }
{ [IterationSpace[q = 4, s = 2] -> Weight[2]] }
```

# Weight Projection

$O_q = I_{q+s} \times F_s$ ⟹ is2weight = isl.Map("{ IterationSpace[q,s] -> Weight[s] }")




Note: Many-to-one relationship

# Output Projection

$O_q = I_{q+s} \times F_s$ ⟹ is2output = isl.Map("{ IterationSpace[q,s] -> Output[q] }")

{ [IterationSpace[q = 0, s = 0] -> Output[0]] }
{ [IterationSpace[q = 0, s = 1] -> Output[0]] }
{ [IterationSpace[q = 0, s = 2] -> Output[0]] }
{ [IterationSpace[q = 1, s = 0] -> Output[1]] }
{ [IterationSpace[q = 1, s = 1] -> Output[1]] }
{ [IterationSpace[q = 1, s = 2] -> Output[1]] }
{ [IterationSpace[q = 2, s = 0] -> Output[2]] }
{ [IterationSpace[q = 2, s = 1] -> Output[2]] }
{ [IterationSpace[q = 2, s = 2] -> Output[2]] }
{ [IterationSpace[q = 3, s = 0] -> Output[3]] }
{ [IterationSpace[q = 3, s = 1] -> Output[3]] }
{ [IterationSpace[q = 3, s = 2] -> Output[3]] }
{ [IterationSpace[q = 4, s = 0] -> Output[4]] }
{ [IterationSpace[q = 4, s = 1] -> Output[4]] }
{ [IterationSpace[q = 4, s = 2] -> Output[4]] }

# Input Projection

$O_q = I_{q+s} \times F_s$ ⟹ is2input = isl.Map("{ IterationSpace[q,s] -> Input[w] : w=q+s }")

{ [IterationSpace[q = 0, s = 0] -> Input[w = 0]] }
{ [IterationSpace[q = 0, s = 1] -> Input[w = 1]] }
{ [IterationSpace[q = 0, s = 2] -> Input[w = 2]] }
{ [IterationSpace[q = 1, s = 0] -> Input[w = 1]] }
{ [IterationSpace[q = 1, s = 1] -> Input[w = 2]] }
{ [IterationSpace[q = 1, s = 2] -> Input[w = 3]] }
{ [IterationSpace[q = 2, s = 0] -> Input[w = 2]] }
{ [IterationSpace[q = 2, s = 1] -> Input[w = 3]] }
{ [IterationSpace[q = 2, s = 2] -> Input[w = 4]] }
{ [IterationSpace[q = 3, s = 0] -> Input[w = 3]] }
{ [IterationSpace[q = 3, s = 1] -> Input[w = 4]] }
{ [IterationSpace[q = 3, s = 2] -> Input[w = 5]] }
{ [IterationSpace[q = 4, s = 0] -> Input[w = 4]] }
{ [IterationSpace[q = 4, s = 1] -> Input[w = 5]] }
{ [IterationSpace[q = 4, s = 2] -> Input[w = 6]] }

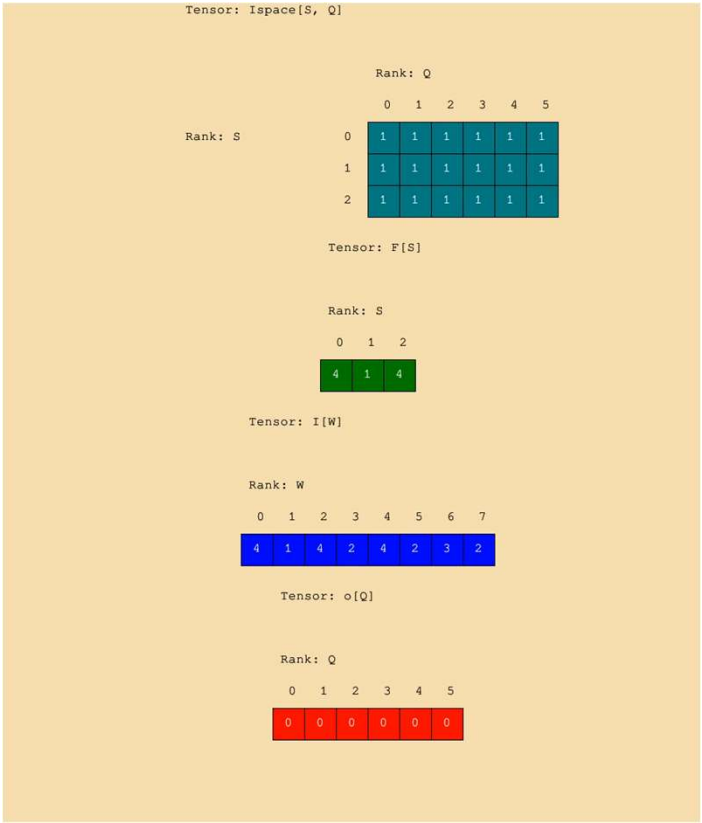
# Compute Projection

$$O_q = I_{q+s} \times F_s$$

is2compute = isl.Map("{ IterationSpace[q,s] -> MACC[q,s] }")

```
{ [IterationSpace[q = 0, s = 0] -> MACC[0, 0]] }
{ [IterationSpace[q = 0, s = 1] -> MACC[0, 1]] }
{ [IterationSpace[q = 0, s = 2] -> MACC[0, 2]] }
{ [IterationSpace[q = 1, s = 0] -> MACC[1, 0]] }
{ [IterationSpace[q = 1, s = 1] -> MACC[1, 1]] }
{ [IterationSpace[q = 1, s = 2] -> MACC[1, 2]] }
{ [IterationSpace[q = 2, s = 0] -> MACC[2, 0]] }
{ [IterationSpace[q = 2, s = 1] -> MACC[2, 1]] }
{ [IterationSpace[q = 2, s = 2] -> MACC[2, 2]] }
{ [IterationSpace[q = 3, s = 0] -> MACC[3, 0]] }
{ [IterationSpace[q = 3, s = 1] -> MACC[3, 1]] }
{ [IterationSpace[q = 3, s = 2] -> MACC[3, 2]] }
{ [IterationSpace[q = 4, s = 0] -> MACC[4, 0]] }
{ [IterationSpace[q = 4, s = 1] -> MACC[4, 1]] }
{ [IterationSpace[q = 4, s = 2] -> MACC[4, 2]] }
```

# Iteration Space Traversal



# Iteration Space to Timestamp

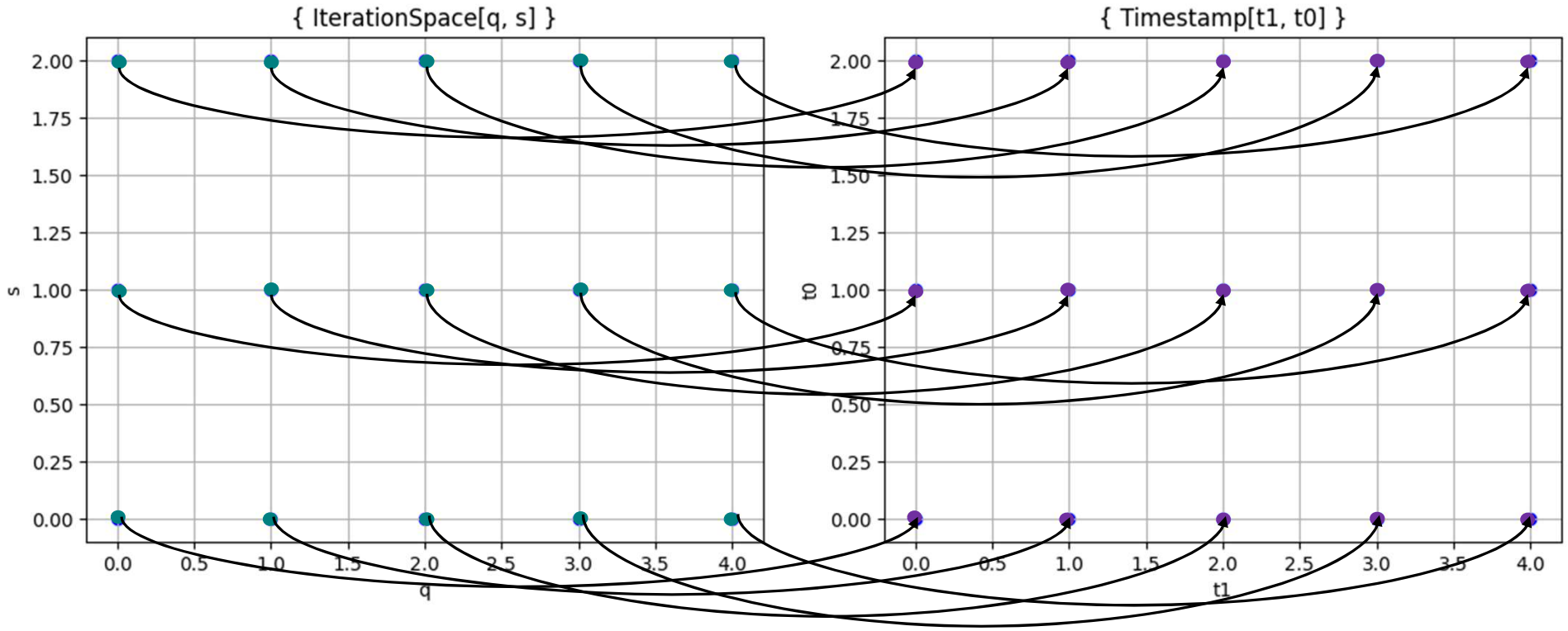
```
df_is2ts_unbounded = isl.Map(f'{{ IterationSpace[q,s] -> Timestamp[t1,t0] : t1=q and t0=s }}')
```

```
df_is2ts = df_is2ts_unbounded.intersect_domain(ispace)
```

```
{ [IterationSpace[q = 0, s = 0] -> Timestamp[t1 = 0, t0 = 0]] }
{ [IterationSpace[q = 0, s = 1] -> Timestamp[t1 = 0, t0 = 1]] }
{ [IterationSpace[q = 0, s = 2] -> Timestamp[t1 = 0, t0 = 2]] }
{ [IterationSpace[q = 1, s = 0] -> Timestamp[t1 = 1, t0 = 0]] }
{ [IterationSpace[q = 1, s = 1] -> Timestamp[t1 = 1, t0 = 1]] }
{ [IterationSpace[q = 1, s = 2] -> Timestamp[t1 = 1, t0 = 2]] }
{ [IterationSpace[q = 2, s = 0] -> Timestamp[t1 = 2, t0 = 0]] }
{ [IterationSpace[q = 2, s = 1] -> Timestamp[t1 = 2, t0 = 1]] }
{ [IterationSpace[q = 2, s = 2] -> Timestamp[t1 = 2, t0 = 2]] }
{ [IterationSpace[q = 3, s = 0] -> Timestamp[t1 = 3, t0 = 0]] }
{ [IterationSpace[q = 3, s = 1] -> Timestamp[t1 = 3, t0 = 1]] }
{ [IterationSpace[q = 3, s = 2] -> Timestamp[t1 = 3, t0 = 2]] }
{ [IterationSpace[q = 4, s = 0] -> Timestamp[t1 = 4, t0 = 0]] }
{ [IterationSpace[q = 4, s = 1] -> Timestamp[t1 = 4, t0 = 1]] }
{ [IterationSpace[q = 4, s = 2] -> Timestamp[t1 = 4, t0 = 2]] }
```

# Iteration Space to Timestamp

```
df_is2ts_unbounded = isl.Map(f'{{ IterationSpace[q,s] -> Timestamp[t1,t0] : t1=q and t0=s }}')
df_is2ts = df_is2ts_unbounded.intersect_domain(ispace)
```



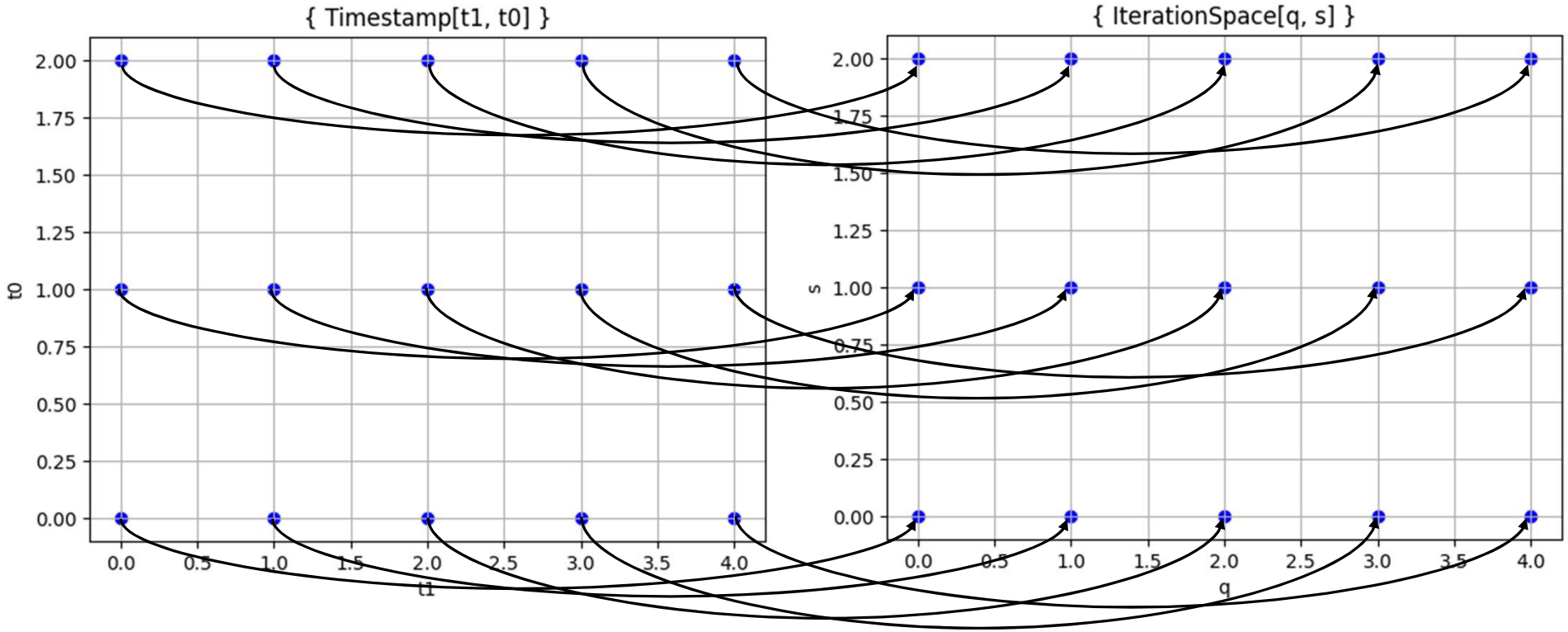
# Timestamp to Iteration Space

```
df_ts2is = df_is2ts.reverse()
```

```
{ [Timestamp[t1 = 0, t0 = 0] -> IterationSpace[q = 0, s = 0]] }
{ [Timestamp[t1 = 0, t0 = 1] -> IterationSpace[q = 0, s = 1]] }
{ [Timestamp[t1 = 0, t0 = 2] -> IterationSpace[q = 0, s = 2]] }
{ [Timestamp[t1 = 1, t0 = 0] -> IterationSpace[q = 1, s = 0]] }
{ [Timestamp[t1 = 1, t0 = 1] -> IterationSpace[q = 1, s = 1]] }
{ [Timestamp[t1 = 1, t0 = 2] -> IterationSpace[q = 1, s = 2]] }
{ [Timestamp[t1 = 2, t0 = 0] -> IterationSpace[q = 2, s = 0]] }
{ [Timestamp[t1 = 2, t0 = 1] -> IterationSpace[q = 2, s = 1]] }
{ [Timestamp[t1 = 2, t0 = 2] -> IterationSpace[q = 2, s = 2]] }
{ [Timestamp[t1 = 3, t0 = 0] -> IterationSpace[q = 3, s = 0]] }
{ [Timestamp[t1 = 3, t0 = 1] -> IterationSpace[q = 3, s = 1]] }
{ [Timestamp[t1 = 3, t0 = 2] -> IterationSpace[q = 3, s = 2]] }
{ [Timestamp[t1 = 4, t0 = 0] -> IterationSpace[q = 4, s = 0]] }
{ [Timestamp[t1 = 4, t0 = 1] -> IterationSpace[q = 4, s = 1]] }
{ [Timestamp[t1 = 4, t0 = 2] -> IterationSpace[q = 4, s = 2]] }
```

# Timestamp to Iteration Space

```
df_ts2is = df_is2ts.reverse()
```

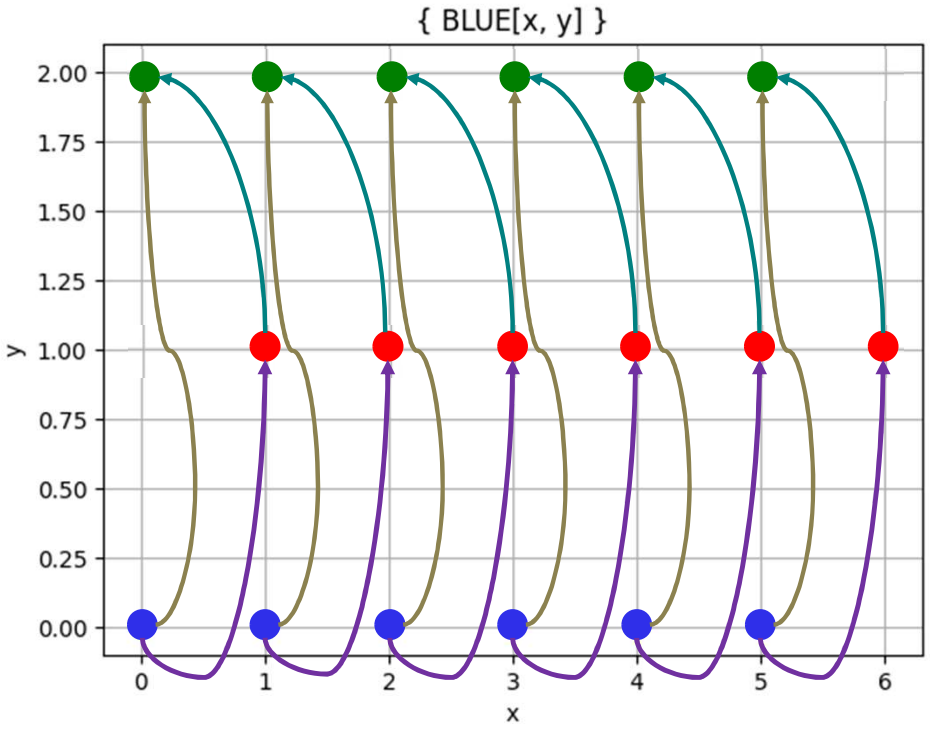


# Composing Maps

```
blue2red = isl.Map("{ BLUE[x,y] -> RED[x1,y1] :
                      x1=x+1 and y1=y+1 }")
```

```
red2green = isl.Map("{ RED[x,y] -> GREEN[x1,y1] :
                       x1=x-1 and y1=y+1 }")
```

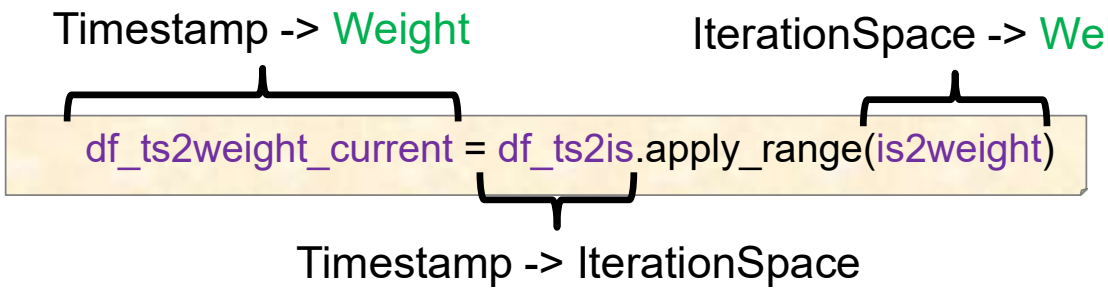
```
blue2green = blue2red.apply_range(red2green)
```




Output:

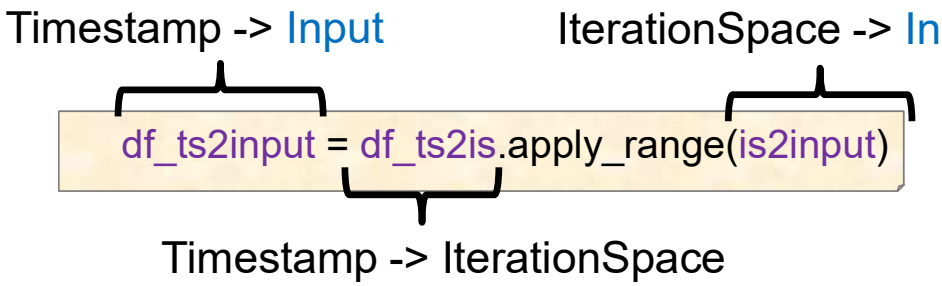
```
blue2green = { BLUE[x, y = 0] -> GREEN[x1 = x, y1 = 2] : 0 <= x <= 5 }
```

# Timestamp to Weight



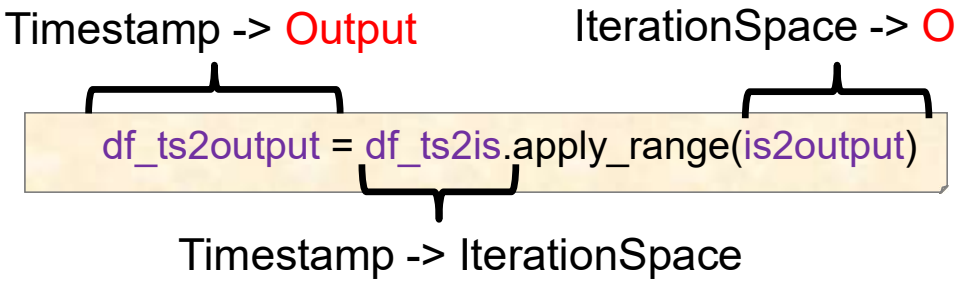

{ [Timestamp[t1 = 0, t0 = 0] -> Weight[0]] }
{ [Timestamp[t1 = 0, t0 = 1] -> Weight[1]] }
{ [Timestamp[t1 = 0, t0 = 2] -> Weight[2]] }
{ [Timestamp[t1 = 1, t0 = 0] -> Weight[0]] }
{ [Timestamp[t1 = 1, t0 = 1] -> Weight[1]] }
{ [Timestamp[t1 = 1, t0 = 2] -> Weight[2]] }
{ [Timestamp[t1 = 2, t0 = 0] -> Weight[0]] }
{ [Timestamp[t1 = 2, t0 = 1] -> Weight[1]] }
{ [Timestamp[t1 = 2, t0 = 2] -> Weight[2]] }
{ [Timestamp[t1 = 3, t0 = 0] -> Weight[0]] }
{ [Timestamp[t1 = 3, t0 = 1] -> Weight[1]] }
{ [Timestamp[t1 = 3, t0 = 2] -> Weight[2]] }
{ [Timestamp[t1 = 4, t0 = 0] -> Weight[0]] }
{ [Timestamp[t1 = 4, t0 = 1] -> Weight[1]] }
{ [Timestamp[t1 = 4, t0 = 2] -> Weight[2]] }

# Timestamp to Input




{ [Timestamp[t1 = 0, t0 = 0] -> Input[w = 0]] }
{ [Timestamp[t1 = 0, t0 = 1] -> Input[w = 1]] }
{ [Timestamp[t1 = 0, t0 = 2] -> Input[w = 2]] }
{ [Timestamp[t1 = 1, t0 = 0] -> Input[w = 1]] }
{ [Timestamp[t1 = 1, t0 = 1] -> Input[w = 2]] }
{ [Timestamp[t1 = 1, t0 = 2] -> Input[w = 3]] }
{ [Timestamp[t1 = 2, t0 = 0] -> Input[w = 2]] }
{ [Timestamp[t1 = 2, t0 = 1] -> Input[w = 3]] }
{ [Timestamp[t1 = 2, t0 = 2] -> Input[w = 4]] }
{ [Timestamp[t1 = 3, t0 = 0] -> Input[w = 3]] }
{ [Timestamp[t1 = 3, t0 = 1] -> Input[w = 4]] }
{ [Timestamp[t1 = 3, t0 = 2] -> Input[w = 5]] }
{ [Timestamp[t1 = 4, t0 = 0] -> Input[w = 4]] }
{ [Timestamp[t1 = 4, t0 = 1] -> Input[w = 5]] }
{ [Timestamp[t1 = 4, t0 = 2] -> Input[w = 6]] }

# Timestamp to Output




{ [Timestamp[t1 = 0, t0 = 0] -> Output[0]] }
{ [Timestamp[t1 = 0, t0 = 1] -> Output[0]] }
{ [Timestamp[t1 = 0, t0 = 2] -> Output[0]] }
{ [Timestamp[t1 = 1, t0 = 0] -> Output[1]] }
{ [Timestamp[t1 = 1, t0 = 1] -> Output[1]] }
{ [Timestamp[t1 = 1, t0 = 2] -> Output[1]] }
{ [Timestamp[t1 = 2, t0 = 0] -> Output[2]] }
{ [Timestamp[t1 = 2, t0 = 1] -> Output[2]] }
{ [Timestamp[t1 = 2, t0 = 2] -> Output[2]] }
{ [Timestamp[t1 = 3, t0 = 0] -> Output[3]] }
{ [Timestamp[t1 = 3, t0 = 1] -> Output[3]] }
{ [Timestamp[t1 = 3, t0 = 2] -> Output[3]] }
{ [Timestamp[t1 = 4, t0 = 0] -> Output[4]] }
{ [Timestamp[t1 = 4, t0 = 1] -> Output[4]] }
{ [Timestamp[t1 = 4, t0 = 2] -> Output[4]] }

# Probing the Schedule

```
print('last timestamp =', df_is2ts.range().lexmax())
```

```
last timestamp = { Timestamp[t1 = 4, t0 = 2] }
```

```
print(isl.Set('{ Timestamp[1,2] }').apply(df_ts2weight))
```

```
{ Weight[2] }
```

```
print(df_ts2weight.intersect_domain(isl.Set('{ Timestamp[1,2] }')))
```

```
{ Timestamp[t1 = 1, t0 = 2] -> Weight[2] }
```

# Calculating Data Movement

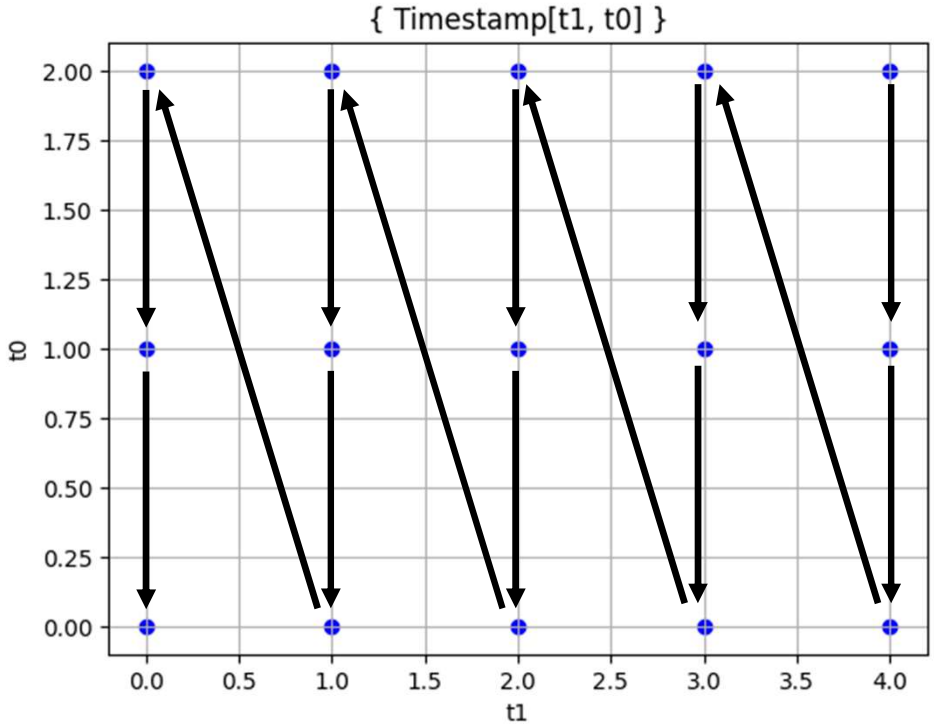
- Create a map from a timestamp to the previous timestamp

- Determine weight access in each timestep
- Determine weight access in previous timestep
- Find changes

- Repeat for inputs and outputs

# Previous Timestamp

```
T0_MAX = S # Ideally should get this from lexmax
timestamp_previous = isl.Map(f"{{ Timestamp[t1',t0'] -> Timestamp[t1,t0] :
t1'=t1 and t0'=t0+1 or t1'=t1+1 and t0'=0 and t0={T0_MAX}-1 }}")
```



# Previous Timestamp Map

```
T0_MAX = S # Ideally should get this from lexmax
timestamp_previous = isl.Map(f"{{ Timestamp[t1',t0'] -> Timestamp[t1,t0] :
t1'=t1 and t0'=t0+1 or t1'=t1+1 and t0'=0 and t0={T0_MAX}-1 }}")
```

```
{ [Timestamp[t1 = 0, t0 = 1] -> Timestamp[t1' = 0, t0' = 0]] }
{ [Timestamp[t1 = 0, t0 = 2] -> Timestamp[t1' = 0, t0' = 1]] }
{ [Timestamp[t1 = 1, t0 = 0] -> Timestamp[t1' = 0, t0' = 2]] }
{ [Timestamp[t1 = 1, t0 = 1] -> Timestamp[t1' = 1, t0' = 0]] }
{ [Timestamp[t1 = 1, t0 = 2] -> Timestamp[t1' = 1, t0' = 1]] }
{ [Timestamp[t1 = 2, t0 = 0] -> Timestamp[t1' = 1, t0' = 2]] }
{ [Timestamp[t1 = 2, t0 = 1] -> Timestamp[t1' = 2, t0' = 0]] }
{ [Timestamp[t1 = 2, t0 = 2] -> Timestamp[t1' = 2, t0' = 1]] }
{ [Timestamp[t1 = 3, t0 = 0] -> Timestamp[t1' = 2, t0' = 2]] }
{ [Timestamp[t1 = 3, t0 = 1] -> Timestamp[t1' = 3, t0' = 0]] }
{ [Timestamp[t1 = 3, t0 = 2] -> Timestamp[t1' = 3, t0' = 1]] }
{ [Timestamp[t1 = 4, t0 = 0] -> Timestamp[t1' = 3, t0' = 2]] }
{ [Timestamp[t1 = 4, t0 = 1] -> Timestamp[t1' = 4, t0' = 0]] }
{ [Timestamp[t1 = 4, t0 = 2] -> Timestamp[t1' = 4, t0' = 1]] }
```

# Weight Use in Previous Timestamp

```
df_ts2weight_previous = timestamp_previous.apply_range(df_ts2weight_current)
```

```
{ [Timestamp[t1 = 0, t0 = 1] -> Weight[0]] }
{ [Timestamp[t1 = 0, t0 = 2] -> Weight[1]] }
{ [Timestamp[t1 = 1, t0 = 0] -> Weight[2]] }
{ [Timestamp[t1 = 1, t0 = 1] -> Weight[0]] }
{ [Timestamp[t1 = 1, t0 = 2] -> Weight[1]] }
{ [Timestamp[t1 = 2, t0 = 0] -> Weight[2]] }
{ [Timestamp[t1 = 2, t0 = 1] -> Weight[0]] }
{ [Timestamp[t1 = 2, t0 = 2] -> Weight[1]] }
{ [Timestamp[t1 = 3, t0 = 0] -> Weight[2]] }
{ [Timestamp[t1 = 3, t0 = 1] -> Weight[0]] }
{ [Timestamp[t1 = 3, t0 = 2] -> Weight[1]] }
{ [Timestamp[t1 = 4, t0 = 0] -> Weight[2]] }
{ [Timestamp[t1 = 4, t0 = 1] -> Weight[0]] }
{ [Timestamp[t1 = 4, t0 = 2] -> Weight[1]] }
```

Previous

```
{ [Timestamp[t1 = 0, t0 = 0] -> Weight[0]] }
{ [Timestamp[t1 = 0, t0 = 1] -> Weight[1]] }
{ [Timestamp[t1 = 0, t0 = 2] -> Weight[2]] }
{ [Timestamp[t1 = 1, t0 = 0] -> Weight[0]] }
{ [Timestamp[t1 = 1, t0 = 1] -> Weight[1]] }
{ [Timestamp[t1 = 1, t0 = 2] -> Weight[2]] }
{ [Timestamp[t1 = 2, t0 = 0] -> Weight[0]] }
{ [Timestamp[t1 = 2, t0 = 1] -> Weight[1]] }
{ [Timestamp[t1 = 2, t0 = 2] -> Weight[2]] }
{ [Timestamp[t1 = 3, t0 = 0] -> Weight[0]] }
{ [Timestamp[t1 = 3, t0 = 1] -> Weight[1]] }
{ [Timestamp[t1 = 3, t0 = 2] -> Weight[2]] }
{ [Timestamp[t1 = 4, t0 = 0] -> Weight[0]] }
{ [Timestamp[t1 = 4, t0 = 1] -> Weight[1]] }
{ [Timestamp[t1 = 4, t0 = 2] -> Weight[2]] }
```

Current

# Timesteps Using a New Weight

```
df_ts2weight_delta = df_ts2weight_current.subtract(df_ts2weight_previous)
```

```
{ [Timestamp[t1 = 0, t0 = 0] -> Weight[0]] }
{ [Timestamp[t1 = 0, t0 = 1] -> Weight[1]] }
{ [Timestamp[t1 = 0, t0 = 2] -> Weight[2]] }
{ [Timestamp[t1 = 1, t0 = 0] -> Weight[0]] }
{ [Timestamp[t1 = 1, t0 = 1] -> Weight[1]] }
{ [Timestamp[t1 = 1, t0 = 2] -> Weight[2]] }
{ [Timestamp[t1 = 2, t0 = 0] -> Weight[0]] }
{ [Timestamp[t1 = 2, t0 = 1] -> Weight[1]] }
{ [Timestamp[t1 = 2, t0 = 2] -> Weight[2]] }
{ [Timestamp[t1 = 3, t0 = 0] -> Weight[0]] }
{ [Timestamp[t1 = 3, t0 = 1] -> Weight[1]] }
{ [Timestamp[t1 = 3, t0 = 2] -> Weight[2]] }
{ [Timestamp[t1 = 4, t0 = 0] -> Weight[0]] }
{ [Timestamp[t1 = 4, t0 = 1] -> Weight[1]] }
{ [Timestamp[t1 = 4, t0 = 2] -> Weight[2]] }
```

# Input Use in Previous Timestamp

```
df_ts2input_previous = timestamp_previous.apply_range(df_ts2input_current)
```

```
{ [Timestamp[t1 = 0, t0 = 1] -> Input[w = 0]] }
{ [Timestamp[t1 = 0, t0 = 2] -> Input[w = 1]] }
{ [Timestamp[t1 = 1, t0 = 0] -> Input[w = 2]] }
{ [Timestamp[t1 = 1, t0 = 1] -> Input[w = 1]] }
{ [Timestamp[t1 = 1, t0 = 2] -> Input[w = 2]] }
{ [Timestamp[t1 = 2, t0 = 0] -> Input[w = 3]] }
{ [Timestamp[t1 = 2, t0 = 1] -> Input[w = 2]] }
{ [Timestemp[t1 = 2, t0 = 2] -> Input[w = 3]] }
{ [Timestamp[t1 = 3, t0 = 0] -> Input[w = 4]] }
{ [Timestamp[t1 = 3, t0 = 1] -> Input[w = 3]] }
{ [Timestamp[t1 = 3, t0 = 2] -> Input[w = 4]] }
{ [Timestamp[t1 = 4, t0 = 0] -> Input[w = 5]] }
{ [Timestamp[t1 = 4, t0 = 1] -> Input[w = 4]] }
{ [Timestamp[t1 = 4, t0 = 2] -> Input[w = 5]] }
{ [Timestamp[t1 = 5, t0 = 0] -> Input[w = 6]] }
```

Previous

```
{ [Timestamp[t1 = 0, t0 = 0] -> Input[w = 0]] }
{ [Timestamp[t1 = 0, t0 = 1] -> Input[w = 1]] }
{ [Timestamp[t1 = 0, t0 = 2] -> Input[w = 2]] }
{ [Timestamp[t1 = 1, t0 = 0] -> Input[w = 1]] }
{ [Timestamp[t1 = 1, t0 = 1] -> Input[w = 2]] }
{ [Timestamp[t1 = 1, t0 = 2] -> Input[w = 3]] }
{ [Timestamp[t1 = 2, t0 = 0] -> Input[w = 2]] }
{ [Timestamp[t1 = 2, t0 = 1] -> Input[w = 3]] }
{ [Timestamp[t1 = 2, t0 = 2] -> Input[w = 4]] }
{ [Timestamp[t1 = 3, t0 = 0] -> Input[w = 3]] }
{ [Timestamp[t1 = 3, t0 = 1] -> Input[w = 4]] }
{ [Timestamp[t1 = 3, t0 = 2] -> Input[w = 5]] }
{ [Timestamp[t1 = 4, t0 = 0] -> Input[w = 4]] }
{ [Timestamp[t1 = 4, t0 = 1] -> Input[w = 5]] }
{ [Timestamp[t1 = 4, t0 = 2] -> Input[w = 6]] }
```

Current

# Timesteps Using a New Input

```
df_ts2input_delta = df_ts2input_current.subtract(df_ts2input_previous)
```

```
{ [Timestamp[t1 = 0, t0 = 0] -> Input[w = 0]] }
{ [Timestamp[t1 = 0, t0 = 1] -> Input[w = 1]] }
{ [Timestamp[t1 = 0, t0 = 2] -> Input[w = 2]] }
{ [Timestamp[t1 = 1, t0 = 0] -> Input[w = 1]] }
{ [Timestamp[t1 = 1, t0 = 1] -> Input[w = 2]] }
{ [Timestamp[t1 = 1, t0 = 2] -> Input[w = 3]] }
{ [Timestamp[t1 = 2, t0 = 0] -> Input[w = 2]] }
{ [Timestamp[t1 = 2, t0 = 1] -> Input[w = 3]] }
{ [Timestamp[t1 = 2, t0 = 2] -> Input[w = 4]] }
{ [Timestamp[t1 = 3, t0 = 0] -> Input[w = 3]] }
{ [Timestamp[t1 = 3, t0 = 1] -> Input[w = 4]] }
{ [Timestamp[t1 = 3, t0 = 2] -> Input[w = 5]] }
{ [Timestamp[t1 = 4, t0 = 0] -> Input[w = 4]] }
{ [Timestamp[t1 = 4, t0 = 1] -> Input[w = 5]] }
{ [Timestamp[t1 = 4, t0 = 2] -> Input[w = 6]] }
```

# Output Use in Previous Timestamp

```
df_ts2output_previous = timestamp_previous.apply_range(df_ts2output_current)
```

```
{ [Timestamp[t1 = 0, t0 = 1] -> Output[0]] }
{ [Timestamp[t1 = 0, t0 = 2] -> Output[0]] }
{ [Timestamp[t1 = 1, t0 = 0] -> Output[0]] }
{ [Timestamp[t1 = 1, t0 = 1] -> Output[1]] }
{ [Timestamp[t1 = 1, t0 = 2] -> Output[1]] }
{ [Timestamp[t1 = 2, t0 = 0] -> Output[1]] }
{ [Timestamp[t1 = 2, t0 = 1] -> Output[2]] }
{ [Timestamp[t1 = 2, t0 = 2] -> Output[2]] }
{ [Timestamp[t1 = 3, t0 = 0] -> Output[2]] }
{ [Timestamp[t1 = 3, t0 = 1] -> Output[3]] }
{ [Timestamp[t1 = 3, t0 = 2] -> Output[3]] }
{ [Timestamp[t1 = 4, t0 = 0] -> Output[3]] }
{ [Timestamp[t1 = 4, t0 = 1] -> Output[4]] }
{ [Timestamp[t1 = 4, t0 = 2] -> Output[4]] }
```

Previous

```
{ [Timestamp[t1 = 0, t0 = 0] -> Output[0]] }
{ [Timestamp[t1 = 0, t0 = 1] -> Output[0]] }
{ [Timestamp[t1 = 0, t0 = 2] -> Output[0]] }
{ [Timestamp[t1 = 1, t0 = 0] -> Output[1]] }
{ [Timestamp[t1 = 1, t0 = 1] -> Output[1]] }
{ [Timestamp[t1 = 1, t0 = 2] -> Output[1]] }
{ [Timestamp[t1 = 2, t0 = 0] -> Output[2]] }
{ [Timestamp[t1 = 2, t0 = 1] -> Output[2]] }
{ [Timestamp[t1 = 2, t0 = 2] -> Output[2]] }
{ [Timestamp[t1 = 3, t0 = 0] -> Output[3]] }
{ [Timestamp[t1 = 3, t0 = 1] -> Output[3]] }
{ [Timestamp[t1 = 3, t0 = 2] -> Output[3]] }
{ [Timestamp[t1 = 4, t0 = 0] -> Output[4]] }
{ [Timestamp[t1 = 4, t0 = 1] -> Output[4]] }
{ [Timestamp[t1 = 4, t0 = 2] -> Output[4]] }
```

Current

# Timesteps Using a New Output

```
df_ts2output_delta = df_ts2output_current.subtract(df_ts2output_previous)
```

```
{ [Timestamp[t1 = 0, t0 = 0] -> Output[0]] }
{ [Timestamp[t1 = 1, t0 = 0] -> Output[1]] }
{ [Timestamp[t1 = 2, t0 = 0] -> Output[2]] }
{ [Timestamp[t1 = 3, t0 = 0] -> Output[3]] }
{ [Timestamp[t1 = 4, t0 = 0] -> Output[4]] }
```

# L1 Buffering Recipe

- Group sets of timestamp points into tiles
- Create map of tiles to corresponding set of timestamp points
- Create map of timesteps of previous tiles
- Calculate deltas

# Tiling Timestamps

```
timestamp2L1timestamp = isl.Map('{ Timestamp[t1,t0] -> L1Timestamp[t1] }')
```

```
{ [Timestamp[t1 = 0, t0 = 0] -> L1Timestamp[0]] }
{ [Timestamp[t1 = 0, t0 = 1] -> L1Timestamp[0]] }
{ [Timestamp[t1 = 0, t0 = 2] -> L1Timestamp[0]] }
{ [Timestamp[t1 = 1, t0 = 0] -> L1Timestamp[1]] }
{ [Timestamp[t1 = 1, t0 = 1] -> L1Timestamp[1]] }
{ [Timestamp[t1 = 1, t0 = 2] -> L1Timestamp[1]] }
{ [Timestamp[t1 = 2, t0 = 0] -> L1Timestamp[2]] }
{ [Timestamp[t1 = 2, t0 = 1] -> L1Timestamp[2]] }
{ [Timestamp[t1 = 2, t0 = 2] -> L1Timestamp[2]] }
{ [Timestamp[t1 = 3, t0 = 0] -> L1Timestamp[3]] }
{ [Timestamp[t1 = 3, t0 = 1] -> L1Timestamp[3]] }
{ [Timestamp[t1 = 3, t0 = 2] -> L1Timestamp[3]] }
{ [Timestamp[t1 = 4, t0 = 0] -> L1Timestamp[4]] }
{ [Timestamp[t1 = 4, t0 = 1] -> L1Timestamp[4]] }
{ [Timestamp[t1 = 4, t0 = 2] -> L1Timestamp[4]] }
```

L1Timestamp = 0

L1Timestamp = 1

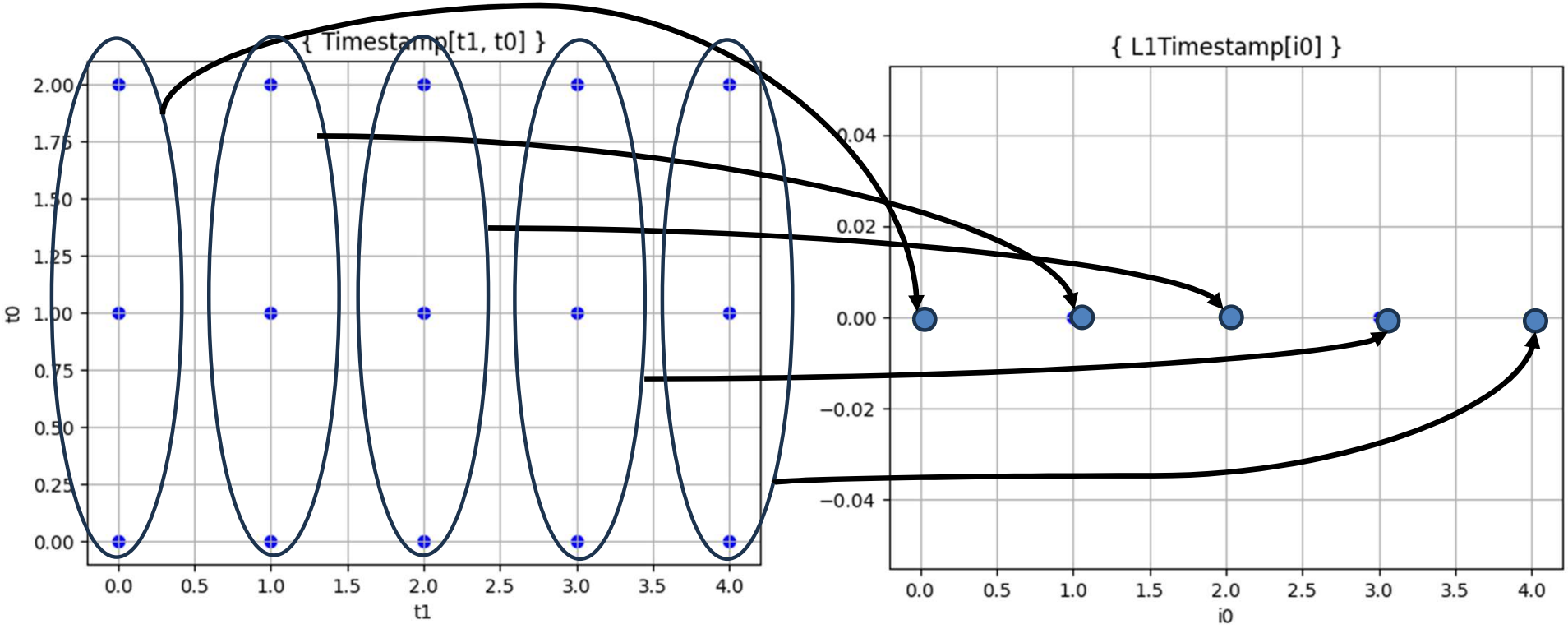
L1Timestamp = 2

L1Timestamp = 3

L1Timestamp = 4

# Tiling Timestamps

```
timestamp2L1timestamp = isl.Map('{ Timestamp[t1,t0] -> L1Timestamp[t1] }')
```



# Tiling Timestamps

L1timestamp2timestamp = timestamp2L1timestamp.reverse()

```
{ [L1Timestamp[0] -> Timestamp[t1 = 0, t0 = 0]] }
{ [L1Timestamp[0] -> Timestamp[t1 = 0, t0 = 1]] }
{ [L1Timestamp[0] -> Timestamp[t1 = 0, t0 = 2]] }
{ [L1Timestamp[1] -> Timestamp[t1 = 1, t0 = 0]] }
{ [L1Timestamp[1] -> Timestamp[t1 = 1, t0 = 1]] }
{ [L1Timestamp[1] -> Timestamp[t1 = 1, t0 = 2]] }
{ [L1Timestamp[2] -> Timestamp[t1 = 2, t0 = 0]] }
{ [L1Timestamp[2] -> Timestamp[t1 = 2, t0 = 1]] }
{ [L1Timestamp[2] -> Timestamp[t1 = 2, t0 = 2]] }
{ [L1Timestamp[3] -> Timestamp[t1 = 3, t0 = 0]] }
{ [L1Timestamp[3] -> Timestamp[t1 = 3, t0 = 1]] }
{ [L1Timestamp[3] -> Timestamp[t1 = 3, t0 = 2]] }
{ [L1Timestamp[4] -> Timestamp[t1 = 4, t0 = 0]] }
{ [L1Timestamp[4] -> Timestamp[t1 = 4, t0 = 1]] }
{ [L1Timestamp[4] -> Timestamp[t1 = 4, t0 = 2]] }
```

L1Timestamp = 0

L1Timestamp = 1

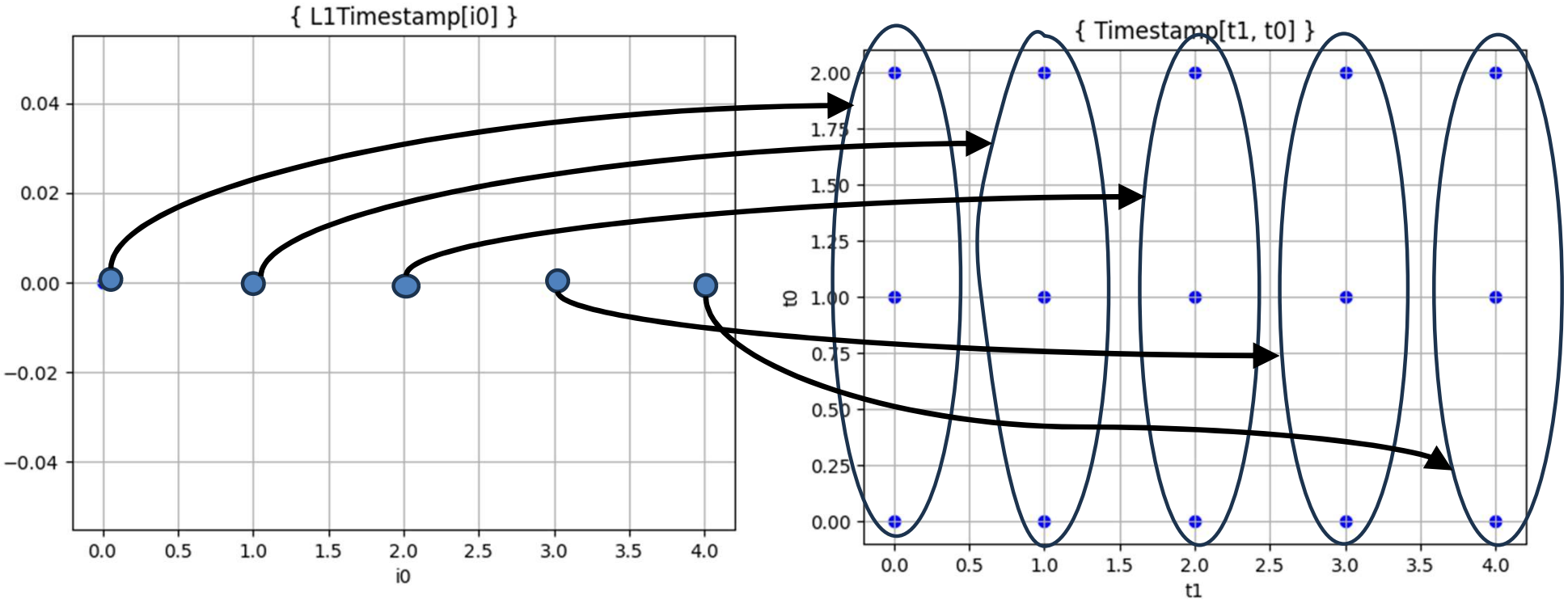
L1Timestamp = 2

L1Timestamp = 3

L1Timestamp = 4

# Tiling Timestamps

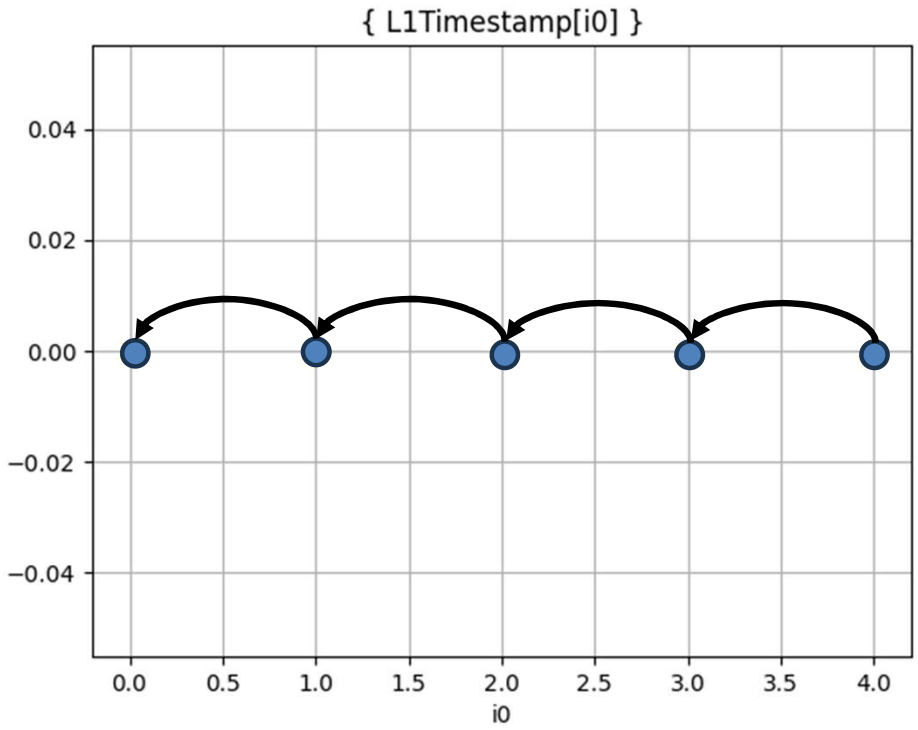
L1timestamp2timestamp = timestamp2L1timestamp.reverse()



# Calculating Tile Data Movement

```
L1timestamp_previous = isl.Map(f"{{ L1Timestamp[t'] -> L1Timestamp[t] : t'=t+1 }}")
```

```
{ [L1Timestamp[1] -> L1Timestamp[0]] }
{ [L1Timestamp[2] -> L1Timestamp[1]] }
{ [L1Timestamp[3] -> L1Timestamp[2]] }
{ [L1Timestamp[4] -> L1Timestamp[3]] }
```



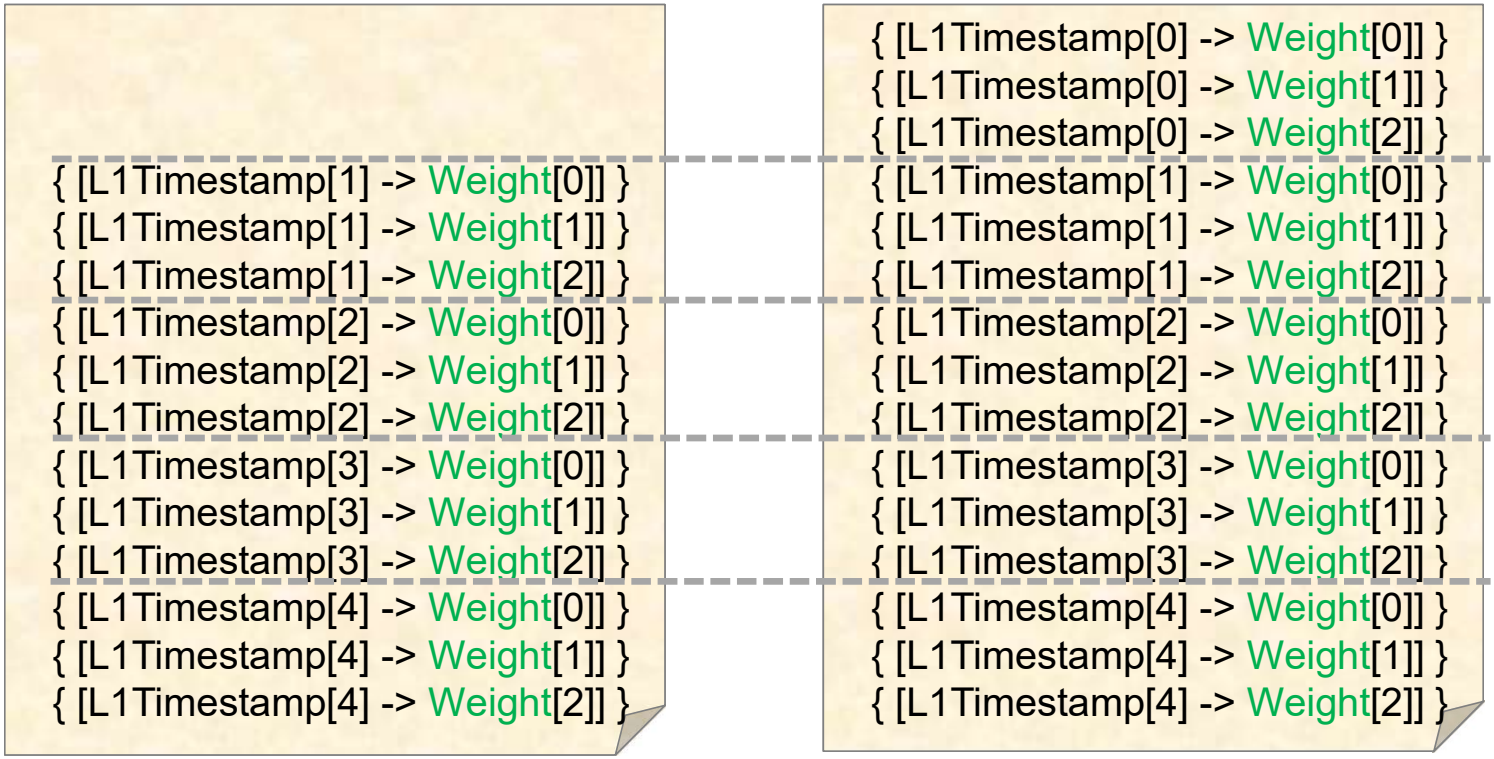
# L1 Weight Accesses

```
df_L1ts2weight_current = L1timestamp2timestamp.apply_range(df_ts2weight_current)
```

```
{ [L1Timestamp[0] -> Weight[0]] }
{ [L1Timestamp[0] -> Weight[1]] }
{ [L1Timestamp[0] -> Weight[2]] }
{ [L1Timestamp[1] -> Weight[0]] }
{ [L1Timestamp[1] -> Weight[1]] }
{ [L1Timestamp[1] -> Weight[2]] }
{ [L1Timestamp[2] -> Weight[0]] }
{ [L1Timestamp[2] -> Weight[1]] }
{ [L1Timestamp[2] -> Weight[2]] }
{ [L1Timestamp[3] -> Weight[0]] }
{ [L1Timestamp[3] -> Weight[1]] }
{ [L1Timestamp[3] -> Weight[2]] }
{ [L1Timestamp[4] -> Weight[0]] }
{ [L1Timestamp[4] -> Weight[1]] }
{ [L1Timestamp[4] -> Weight[2]] }
```

# L1 Weight Accesses

```
df_L1ts2weight_current  = L1timestamp2timestamp.apply_range(df_ts2weight_current)
df_L1ts2weight_previous = L1timestamp_previous.apply_range(df_L1ts2weight_current)
```



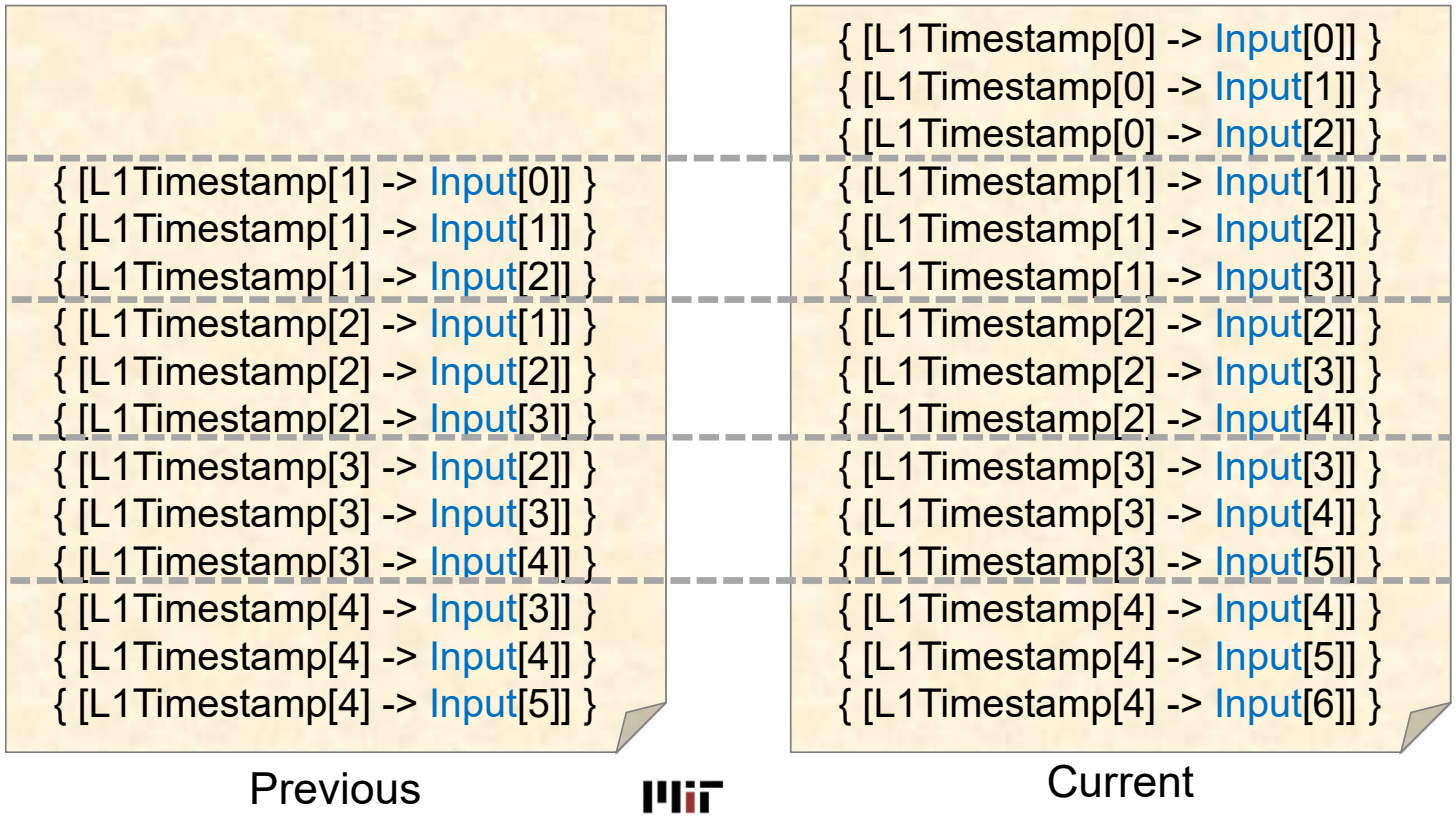
# L1 Weight Deltas

```
df_L1ts2weight_delta = df_L1ts2weight_current.subtract(df_L1ts2weight_previous)
```

```
{ [L1Timestamp[0] -> Weight[0]] }
{ [L1Timestamp[0] -> Weight[1]] }
{ [L1Timestamp[0] -> Weight[2]] }
```

# L1 Input Accesses

```
df_L1ts2input_current = L1timestamp2timestamp.apply_range(df_ts2input_current)
df_L1ts2input_previous = L1timestamp_previous.apply_range(df_L1ts2input_current)
```



# L1 Input Deltas

```
df_L1ts2input_delta = df_L1ts2input_current.subtract(df_L1ts2input_previous)
```

```
{ [L1Timestamp[0] -> Input[0]] }
{ [L1Timestamp[0] -> Input[1]] }
{ [L1Timestamp[0] -> Input[2]] }
-----------------------------------
{ [L1Timestamp[1] -> Input[3]] }
-----------------------------------
{ [L1Timestamp[2] -> Input[4]] }
-----------------------------------
{ [L1Timestamp[3] -> Input[5]] }
-----------------------------------
{ [L1Timestamp[4] -> Input[6]] }
```

It's a sliding window!

# L1 Output Accesses

```
df_L1ts2output_current = L1timestamp2timestamp.apply_range(df_ts2weight_current)
df_L1ts2output_previous = L1timestamp_previous.apply_range(df_L1ts2output_current)
```

| Previous | Current |
|---|---|
| | { [L1Timestamp[0] -> Output[0]] } |
| { [L1Timestamp[1] -> Output[0]] } | { [L1Timestamp[1] -> Output[1]] } |
| { [L1Timestamp[2] -> Output[1]] } | { [L1Timestamp[2] -> Output[2]] } |
| { [L1Timestamp[3] -> Output[2]] } | { [L1Timestamp[3] -> Output[3]] } |
| { [L1Timestamp[4] -> Output[3]] } | { [L1Timestamp[4] -> Output[4]] } |

# L1 Weight Deltas

```
df_L1ts2output_delta = df_L1ts2output_current.subtract(df_L1ts2output_previous)
```

```
{ [L1Timestamp[0] -> Output[0]] }
{ [L1Timestamp[1] -> Output[1]] }
{ [L1Timestamp[2] -> Output[2]] }
{ [L1Timestamp[3] -> Output[3]] }
{ [L1Timestamp[4] -> Output[4]] }
```

# Shrink Recipe

- Calculate data needed in next cycle
- Take difference from data needed in previous cycle

# Tiling Timestamps

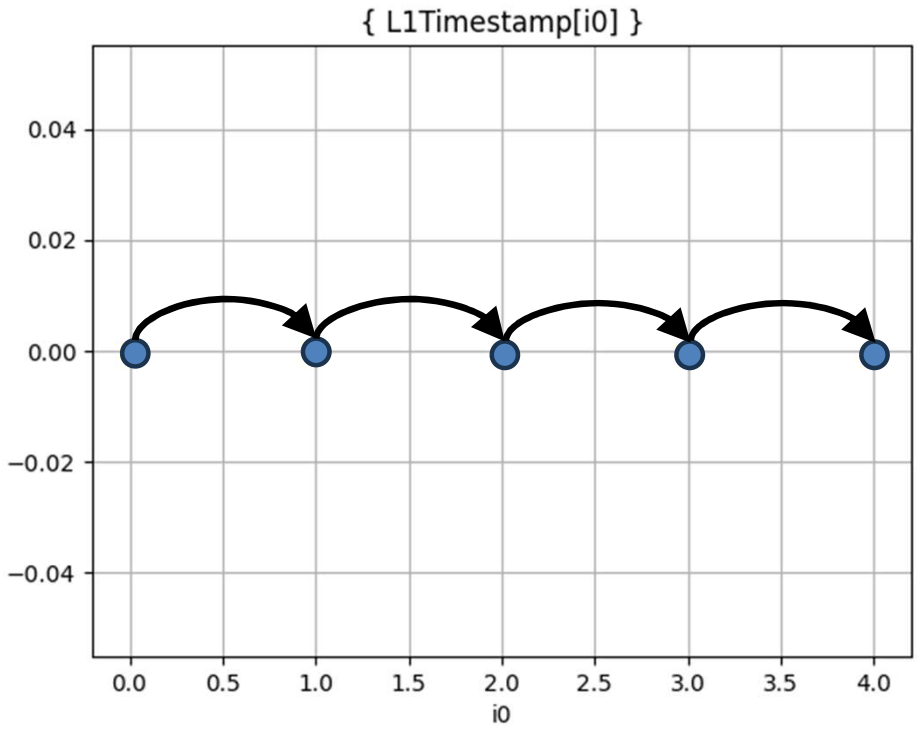
L1timestamp_next = L1timestamp_previous.reverse()

{ [L1Timestamp[0] -> L1Timestamp[1]] }

{ [L1Timestamp[1] -> L1Timestamp[2]] }

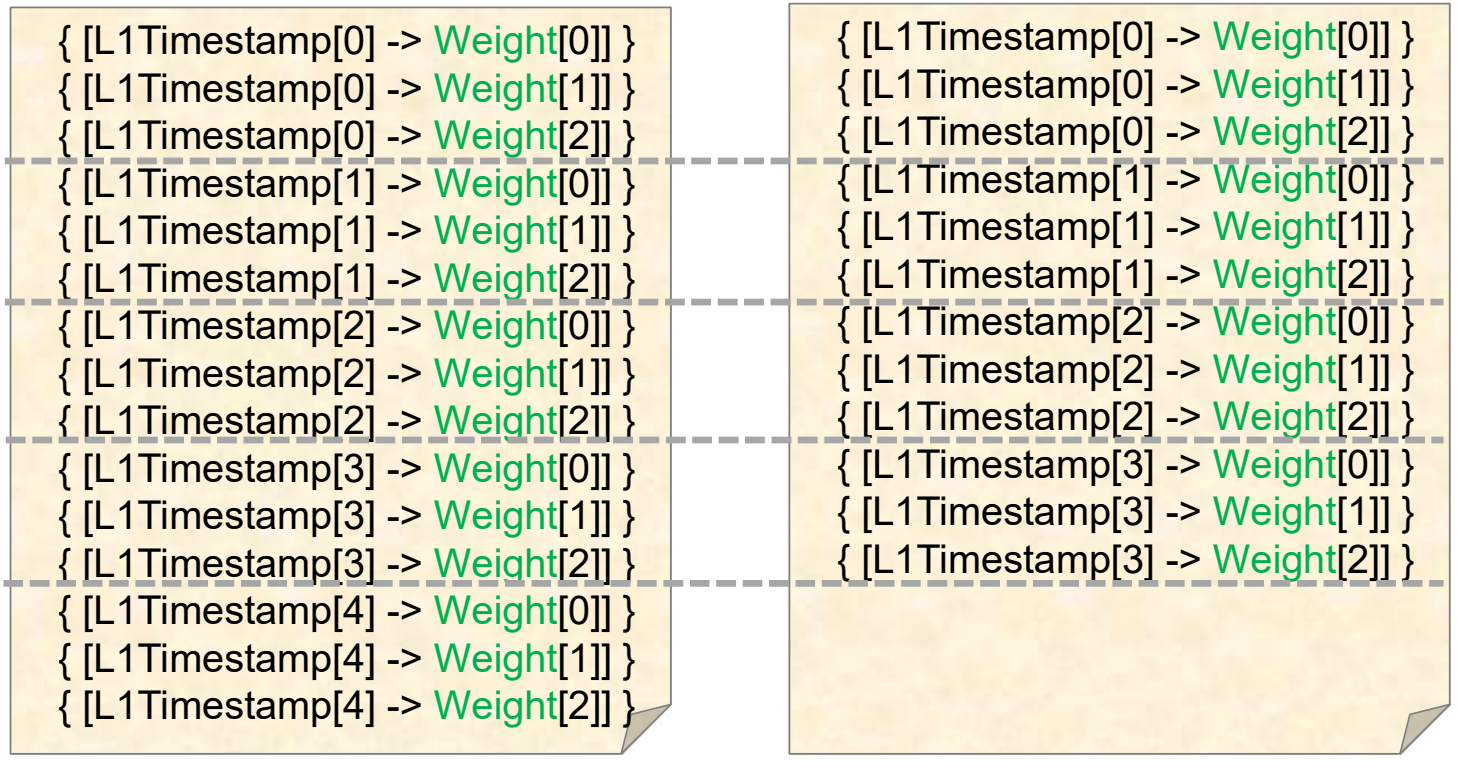
{ [L1Timestamp[2] -> L1Timestamp[3]] }

{ [L1Timestamp[3] -> L1Timestamp[4]] }



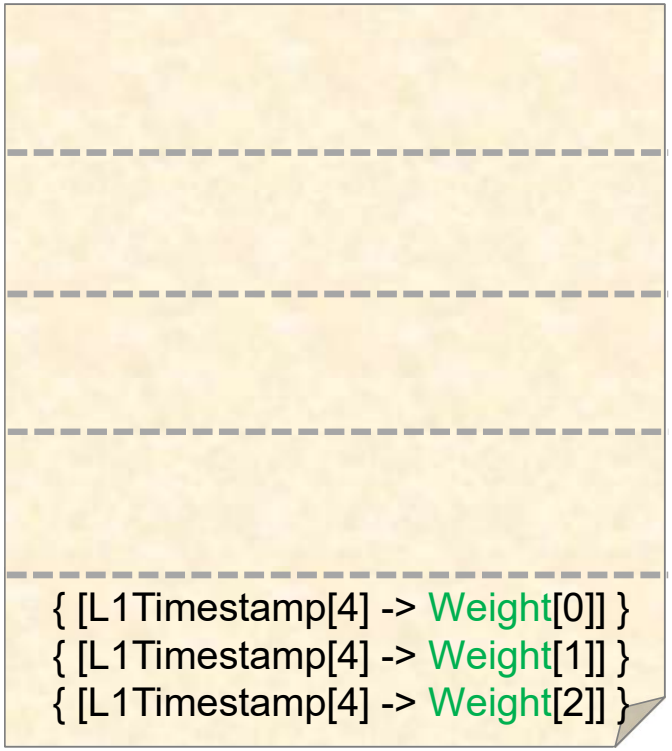
# L1 Weight Accesses (next)

```
df_L1ts2weight_current = L1timestamp2timestamp.apply_range(df_ts2weight_current)
df_L1ts2weight_next = L1timestamp_next.apply_range(df_L1ts2weight_current)
```



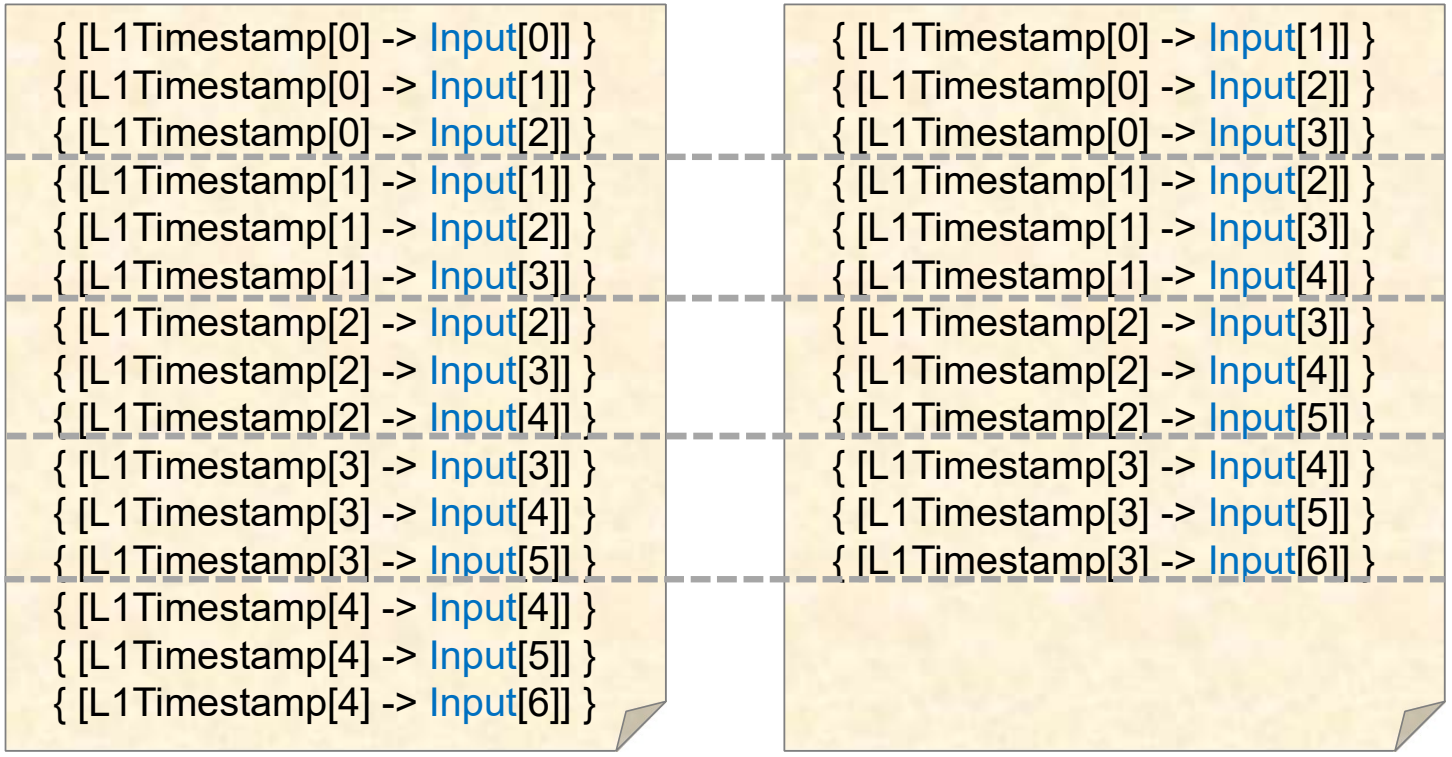
# L1 Weight Shrinks

```
df_L1ts2weight_shrink = df_L1ts2weight_current.subtract(df_L1ts2weight_next)
```



# L1 Input Accesses (Next)

```
df_L1ts2input_current = L1timestamp2timestamp.apply_range(df_ts2input_current)
df_L1ts2input_next = L1timestamp_next.apply_range(df_L1ts2input_current)
```




Current

Next

# L1 Input Shrink

df_L1ts2input_shrink = df_L1ts2input_current.subtract(df_L1ts2input_next)

{ [L1Timestamp[0] -> Input[0]] }

{ [L1Timestamp[1] -> Input[1]] }

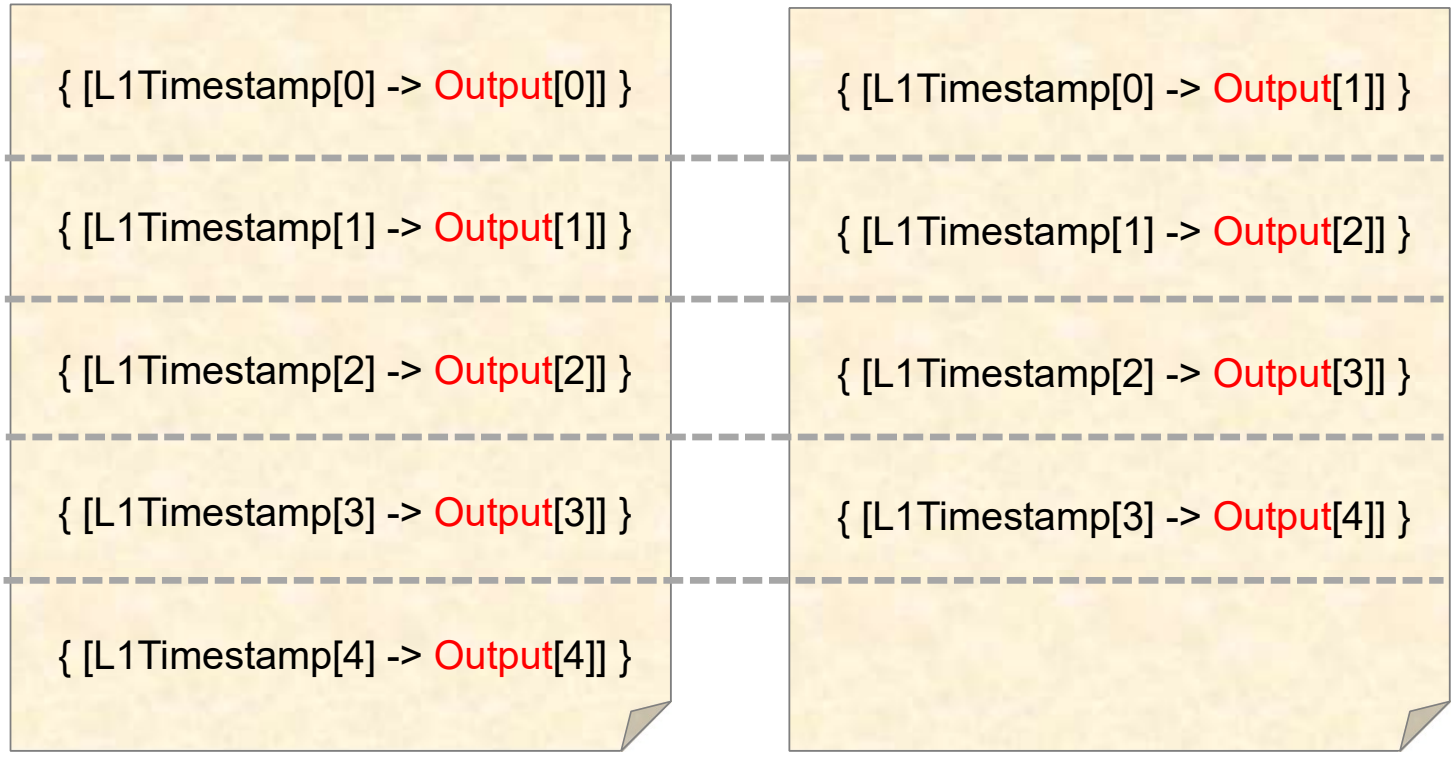
{ [L1Timestamp[2] -> Input[2]] }

{ [L1Timestamp[3] -> Input[3]] }

{ [L1Timestamp[4] -> Input[4]] }
{ [L1Timestamp[4] -> Input[5]] }
{ [L1Timestamp[4] -> Input[6]] }

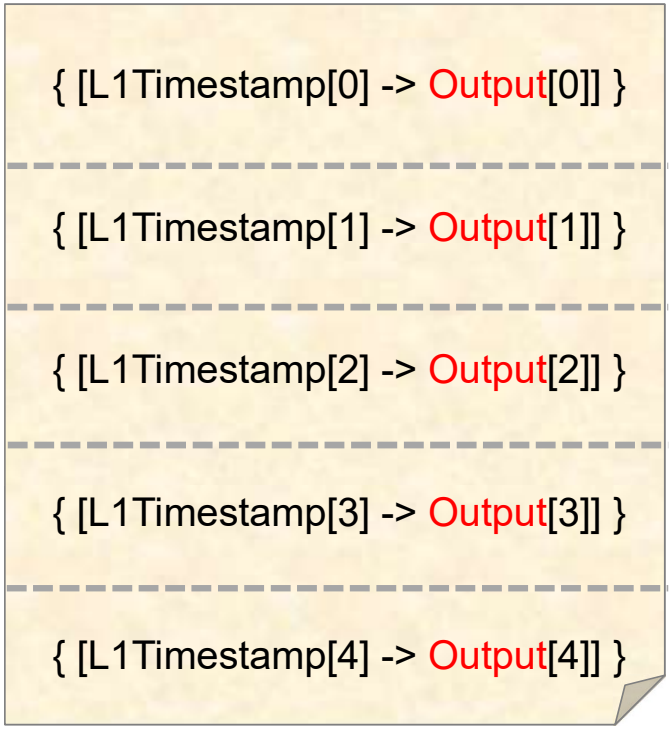
# L1 Output Accesses (Next)

```
df_L1ts2output_current = L1timestamp2timestamp.apply_range(df_ts2weight_current)
df_L1ts2output_next = L1timestamp_next.apply_range(df_L1ts2output_current)
```



# L1 Output Shrink

```
df_L1ts2output_shrink = df_L1ts2output_current.subtract(df_L1ts2output_next)
```



# OS -> WS

```
os_schedule = isl.Map(f'{{ IterationSpace[q,s] -> Timestamp[t1,t0] : t1=q and t0=s }}')
```

```
ws_schedule = isl.Map(f'{{ IterationSpace[q,s] -> Timestamp[t1,t0] : t1=s and t0=q }}')
```

Thank you!