In [1]:
```
# Begin - startup boilerplate code

import pkgutil

if 'fibertree_bootstrap' not in [pkg.name for pkg in pkgutil.iter_modules()]
  !python3 -m pip  install git+https://github.com/Fibertree-project/fibertre

# End - startup boilerplate code


from fibertree_bootstrap import *
fibertree_bootstrap(style="uncompressed", animation="spacetime")
```

```
Running bootstrap
The fibertree module is already installed and available to import
interactive(children=(Dropdown(description='style', options=('tree', 'uncompr
essed', 'tree+uncompressed'), val…
Button(description='Run all cells below', style=ButtonStyle())
```

An Unfused Mapping of a Matrix Vector Multiplication

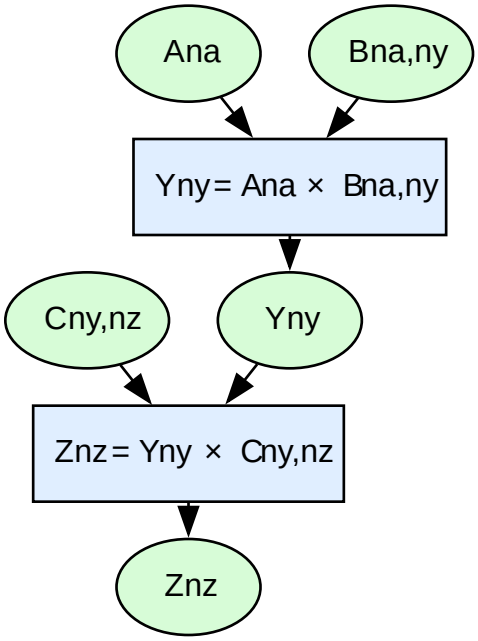
First, we import our specification.

In [2]:
```
import accelforge as af

spec = af.Spec.from_yaml(
    af.examples.arches.simple,
    af.examples.workloads.matvecs,
    af.examples.mappings.unfused_matvecs_to_simple,
)
```

Our workload is a matrix vector multiplication:

In [3]:
```
spec.workload
```

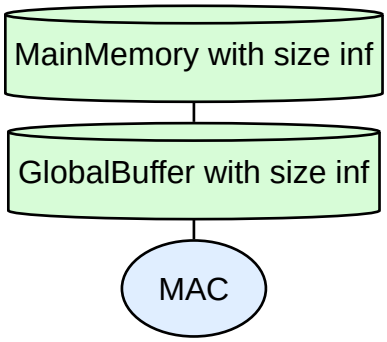
Out[3]:



And our architecture has a MainMemory and a GlobalBuffer:

In [4]:
```
spec.arch
```

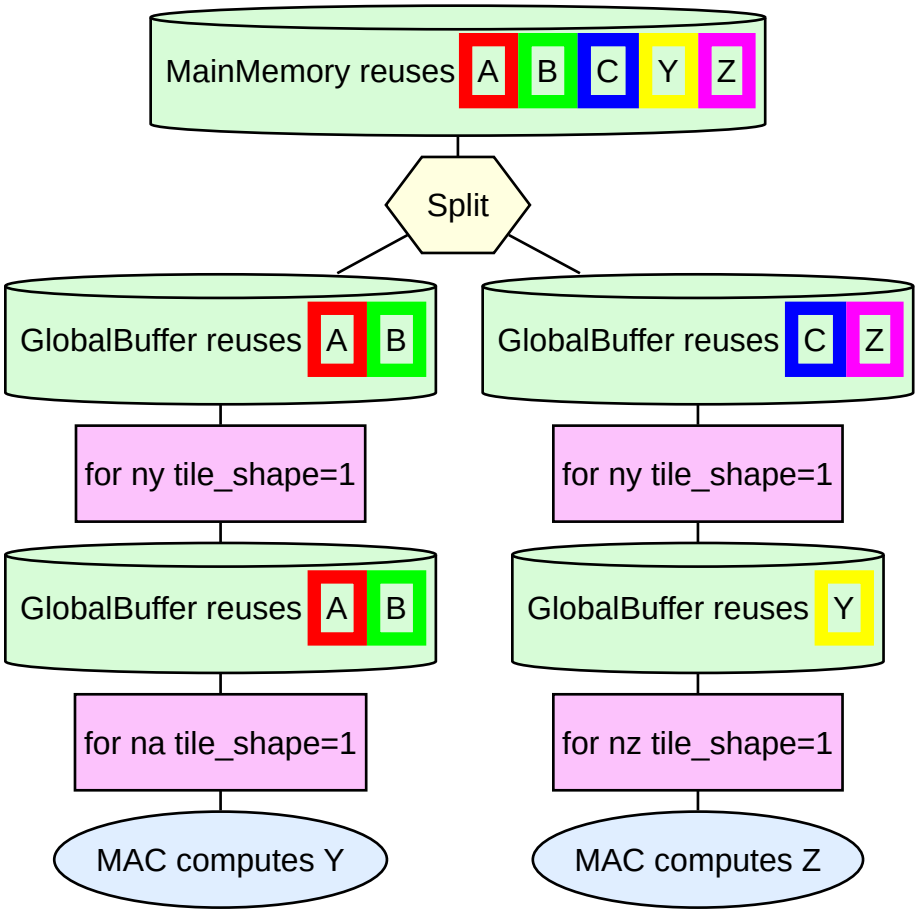
Out[4]:



Our mapping processes EinsumY and then EinsumZ, keeping tensor Y in MainMemory.

In [5]:
```
spec.mapping
```

Out[5]:



In [6]:

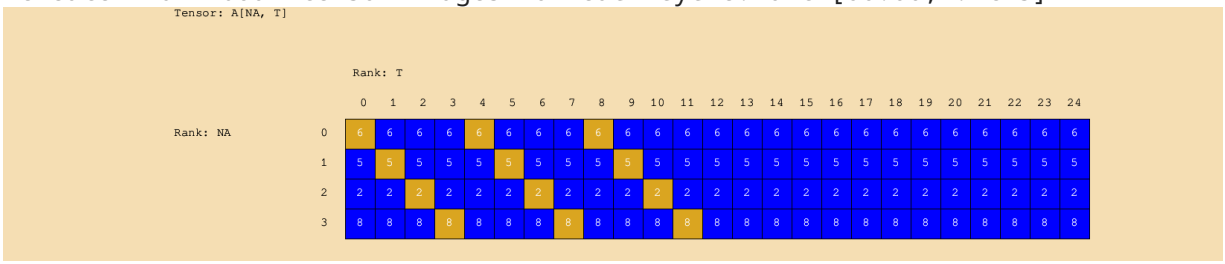
```
def make_tensors_from_spec(spec):
    tensor_name_to_fibertree_tensor = {}
    workload = spec.workload
    for tensor_name in workload.tensor_names:
        is_mutable = any(a.output for a in workload.accesses_for_tensor(tens
        is_output = all(a.output for a in workload.accesses_for_tensor(tenso
        tensor_shape_dict = workload.get_tensor_shape(tensor_name)
        rank_names = tuple(tensor_shape_dict.keys())
        shape = tuple(tensor_shape_dict.values())
        if is_mutable:
            raw_tensor = np.zeros(shape).tolist()
        else:
            raw_tensor = np.random.randint(1, 9, shape).tolist()
        tensor = Tensor.fromUncompressed(rank_names, raw_tensor, default=Non
        tensor.setName(tensor_name)
        tensor.setMutable(is_mutable)
        if is_mutable and not is_output:
            tensor.setColor("purple")
        elif is_mutable and is_output:
            tensor.setColor("red")
        else:
            tensor.setColor("blue")
        tensor_name_to_fibertree_tensor[tensor_name] = tensor
    return tensor_name_to_fibertree_tensor
```

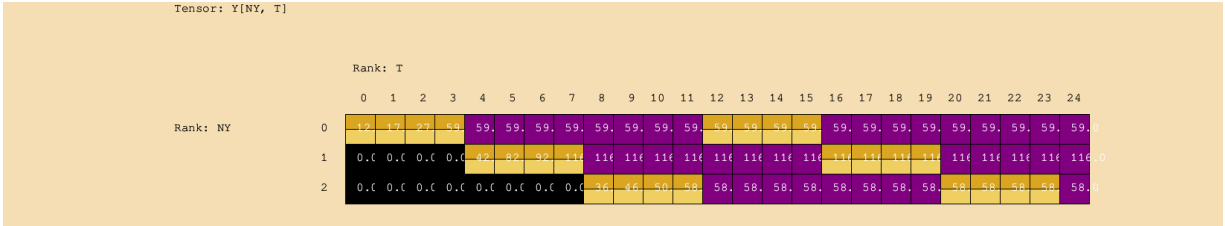
The animation is shown below. The darker highlight shows the tensor elements currently used in a computation. The lighter highlight shows what is resident in the

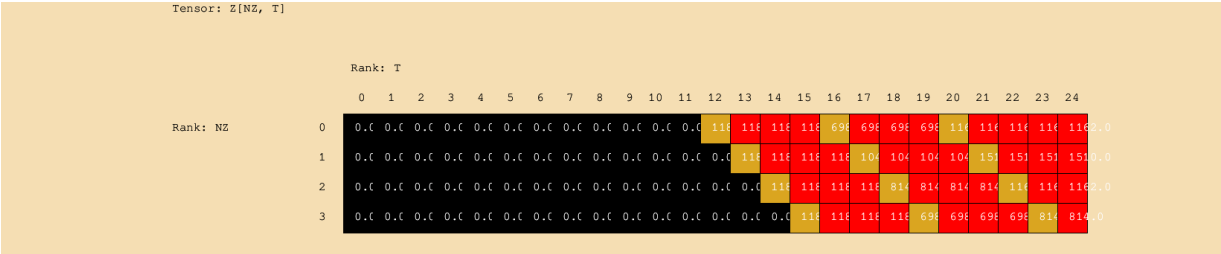
GlobalBuffer.

In [7]:
```
tensor_name_to_fibertree_tensor = make_tensors_from_spec(spec)

A = tensor_name_to_fibertree_tensor["A"]
Y = tensor_name_to_fibertree_tensor["Y"]
Z = tensor_name_to_fibertree_tensor["Z"]
B = tensor_name_to_fibertree_tensor["B"]
B.setColor("green")
C = tensor_name_to_fibertree_tensor["C"]
C.setColor("green")

a = A.getRoot()
y = Y.getRoot()
z = Z.getRoot()
b = B.getRoot()
c = C.getRoot()


canvas = createCanvas(A, Y, Z)
for ny in range(3):
    for na in range(4):
        y[ny] += a[na]*b[na][ny]
        canvas.addActivity([], [(ny,)], [], worker="Y in GlobalBuffer")
        canvas.addFrame([(na,)], [(ny,)], [])
for ny in range(3):
    for nz in range(4):
        z[nz] += y[ny]*c[ny][nz]
        canvas.addActivity([], [(ny,)], [], worker="Y in GlobalBuffer")
        canvas.addFrame([], [(ny,)], [(nz,)])

displayCanvas(canvas)
```

```
Starting simulation
Finished simulation
Spacetime
Create individual tensor images for each cycle: 0it [00:00, ?it/s]
```







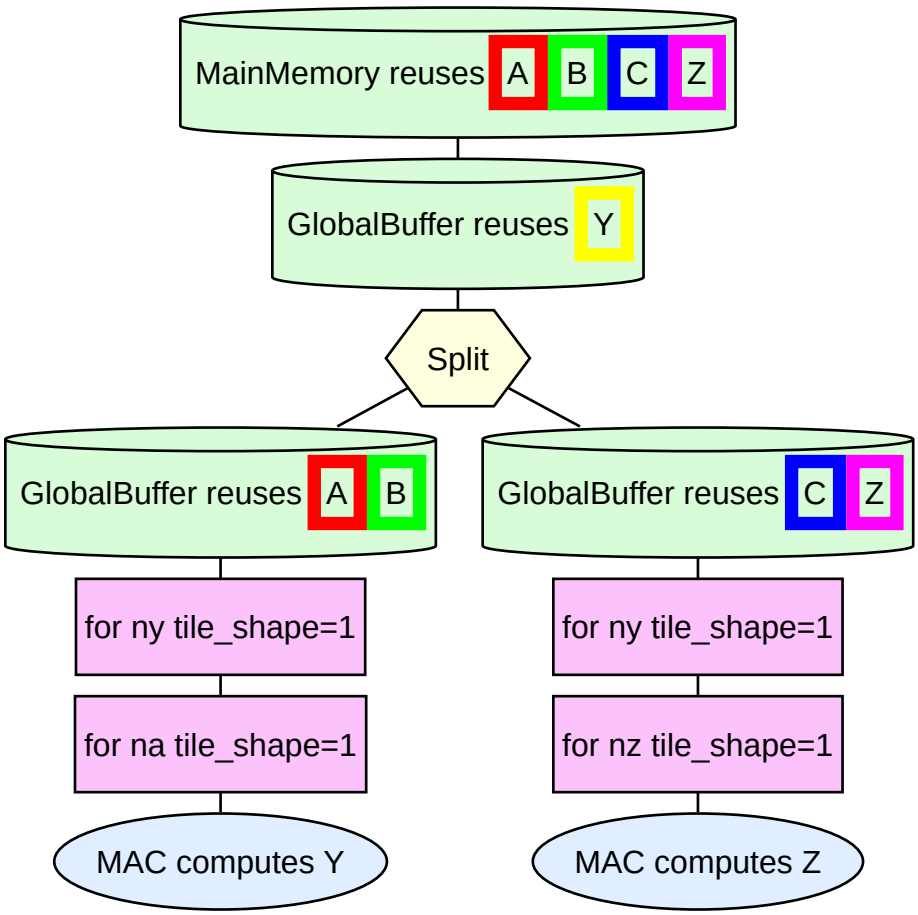
A Fused Mapping without Tiling

The term *fusion* refers to keeping and reusing tensors *between different Einsums*. We will look at a mapping with fusion below.

First, we load specs for the same architecture and workload, but with a mapping that fuses Einsums Y and Z.

In [8]:
```
spec = af.Spec.from_yaml(
    af.examples.arches.simple,
    af.examples.workloads.matvecs,
    af.examples.mappings.fused_matvecs_to_simple_untiled,
)
spec.mapping
```

Out[8]:



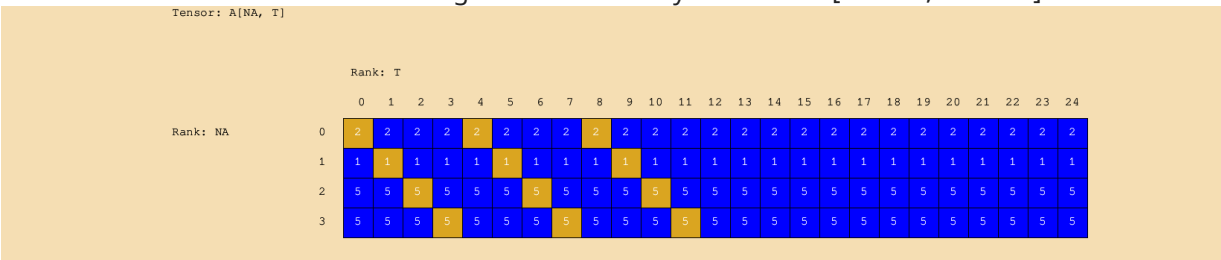
This mapping fuses fuses Einsums Y and Z by keeping the intermediate tensor Y in the

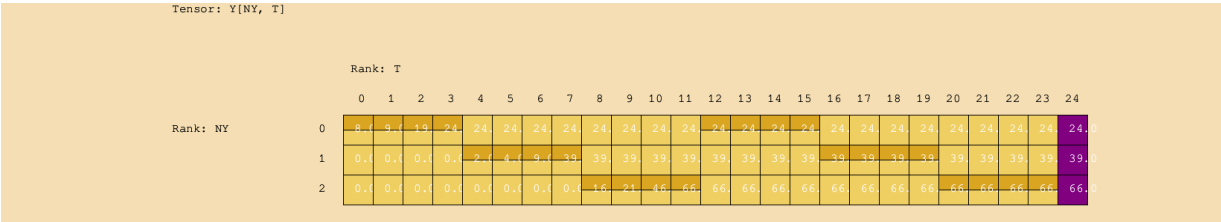
GlobalBuffer between the processing of Einsum Y and Z.

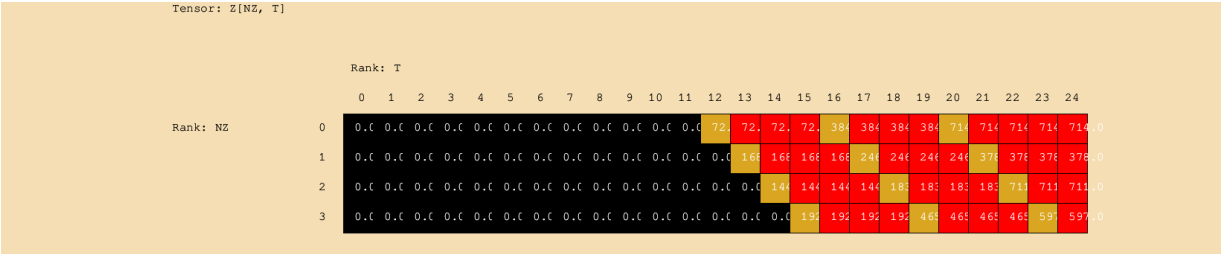
This mapping is animated below.

In [9]:
```
tensor_name_to_fibertree_tensor = make_tensors_from_spec(spec)

A = tensor_name_to_fibertree_tensor["A"]
Y = tensor_name_to_fibertree_tensor["Y"]
Z = tensor_name_to_fibertree_tensor["Z"]
B = tensor_name_to_fibertree_tensor["B"]
B.setColor("green")
C = tensor_name_to_fibertree_tensor["C"]
C.setColor("green")

a = A.getRoot()
y = Y.getRoot()
z = Z.getRoot()
b = B.getRoot()
c = C.getRoot()


canvas = createCanvas(A, Y, Z)
for ny in range(3):
    for na in range(4):
        y[ny] += a[na]*b[na][ny]
        canvas.addActivity([], [(nyt,) for nyt in range(3)], [], worker="Y i
        canvas.addFrame([(na,)], [(ny,)], [])
for ny in range(3):
    for nz in range(4):
        z[nz] += y[ny]*c[ny][nz]
        canvas.addActivity([], [(nyt,) for nyt in range(3)], [], worker="Y i
        canvas.addFrame([], [(ny,)], [(nz,)])

displayCanvas(canvas)
```

```
Starting simulation
Finished simulation
Spacetime
Create individual tensor images for each cycle: 0it [00:00, ?it/s]
```







Although this mapping reduces accesses to MainMemory, the amount of GlobalBuffer capacity required scales with the size of tensor Y, which could be very large.

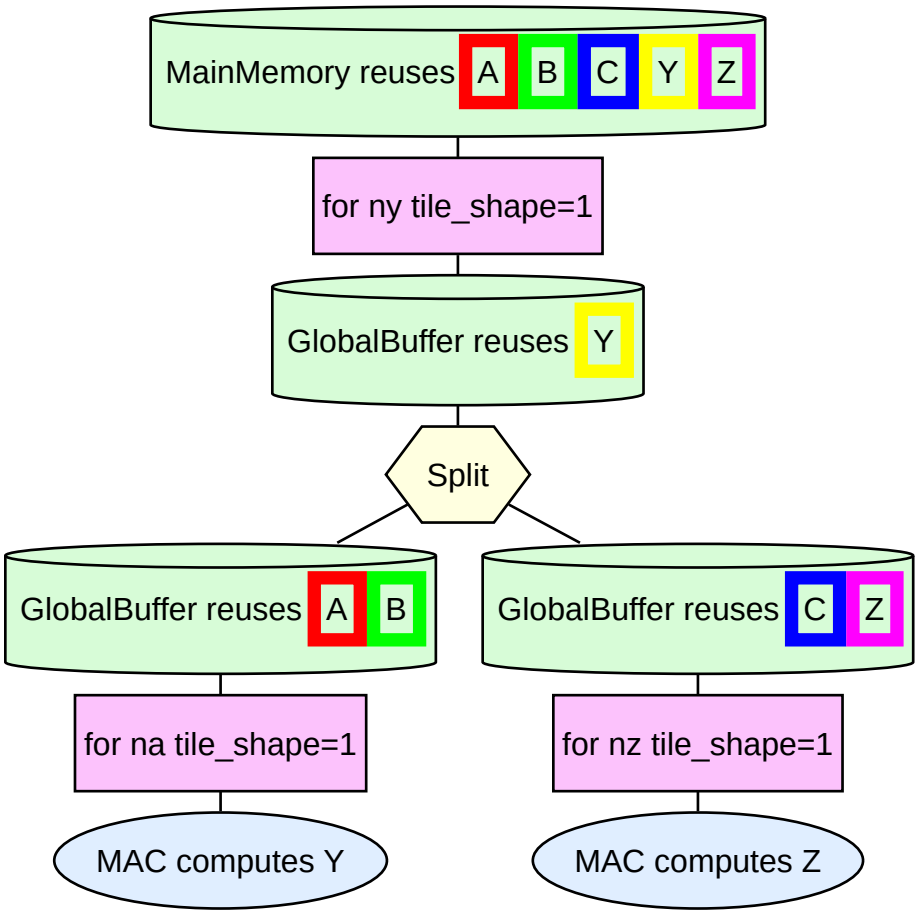
A Mapping with Fusion and Tiling

Often, we can fuse while keeping only a tile of the shared tensor (tensor Y in our case).

We load a specification for such a mapping below.

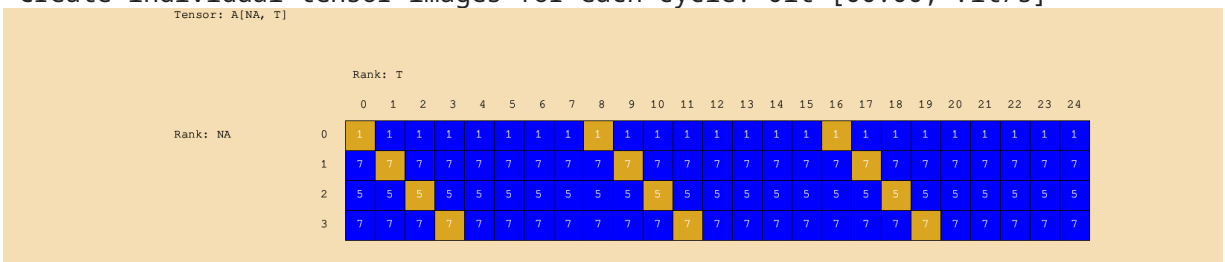
In [10]:
```
spec = af.Spec.from_yaml(
    af.examples.arches.simple,
    af.examples.workloads.matvecs,
    af.examples.mappings.fused_matvecs_to_simple_tiled,
)
spec.mapping
```

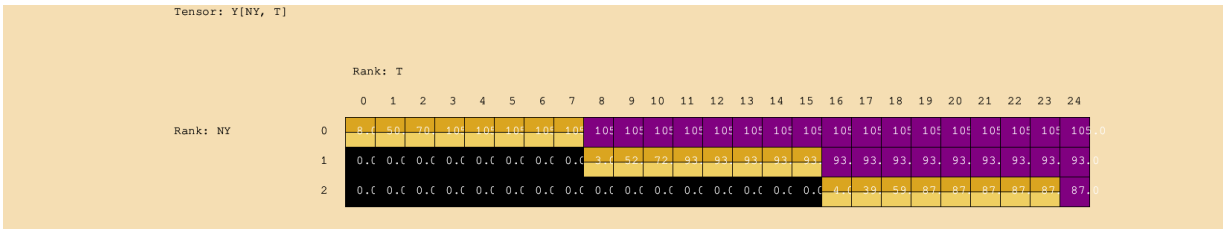
Out[10]:

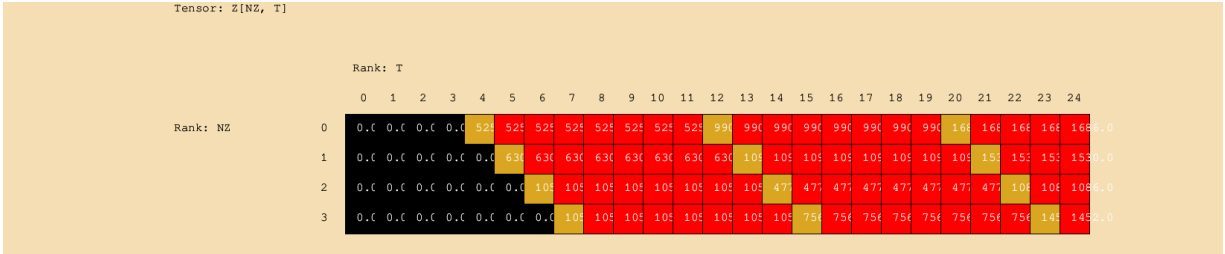


In [11]:
```
tensor_name_to_fibertree_tensor = make_tensors_from_spec(spec)

A = tensor_name_to_fibertree_tensor["A"]
Y = tensor_name_to_fibertree_tensor["Y"]
Z = tensor_name_to_fibertree_tensor["Z"]
B = tensor_name_to_fibertree_tensor["B"]
B.setColor("green")
C = tensor_name_to_fibertree_tensor["C"]
C.setColor("green")

a = A.getRoot()
y = Y.getRoot()
z = Z.getRoot()
b = B.getRoot()
c = C.getRoot()


canvas = createCanvas(A, Y, Z)
for ny in range(3):
    for na in range(4):
        y[ny] += a[na]*b[na][ny]
        canvas.addActivity([], [(ny,)], [], worker="Y in GlobalBuffer")
        canvas.addFrame([(na,)], [(ny,)], [])
    for nz in range(4):
        z[nz] += y[ny]*c[ny][nz]
        canvas.addActivity([], [(ny,)], [], worker="Y in GlobalBuffer")
        canvas.addFrame([], [(ny,)], [(nz,)])

displayCanvas(canvas)
```

```
Starting simulation
Finished simulation
Spacetime
Create individual tensor images for each cycle: 0it [00:00, ?it/s]
```







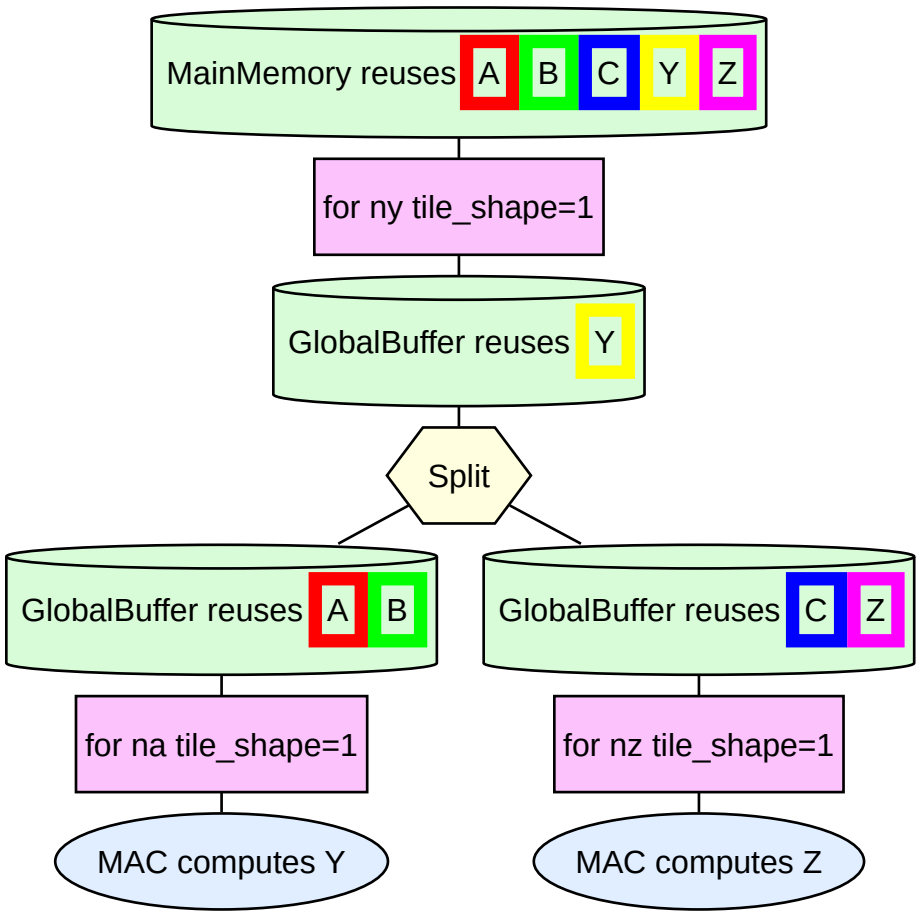
A Note on Tensor Lifetimes

With a mapping of a cascade of Einsums, it is not uncommon to have tiles from different sets of tensors in a memory level at different times.

For example, in the following mapping, the tensors A and Z only live in the GlobalBuffer during their respective Einsums Y and Z, but tensor Y lives throughout.

In [12]:
```
spec = af.Spec.from_yaml(
    af.examples.arches.simple,
    af.examples.workloads.matvecs,
    af.examples.mappings.fused_matvecs_to_simple_tiled,
)
spec.mapping
```

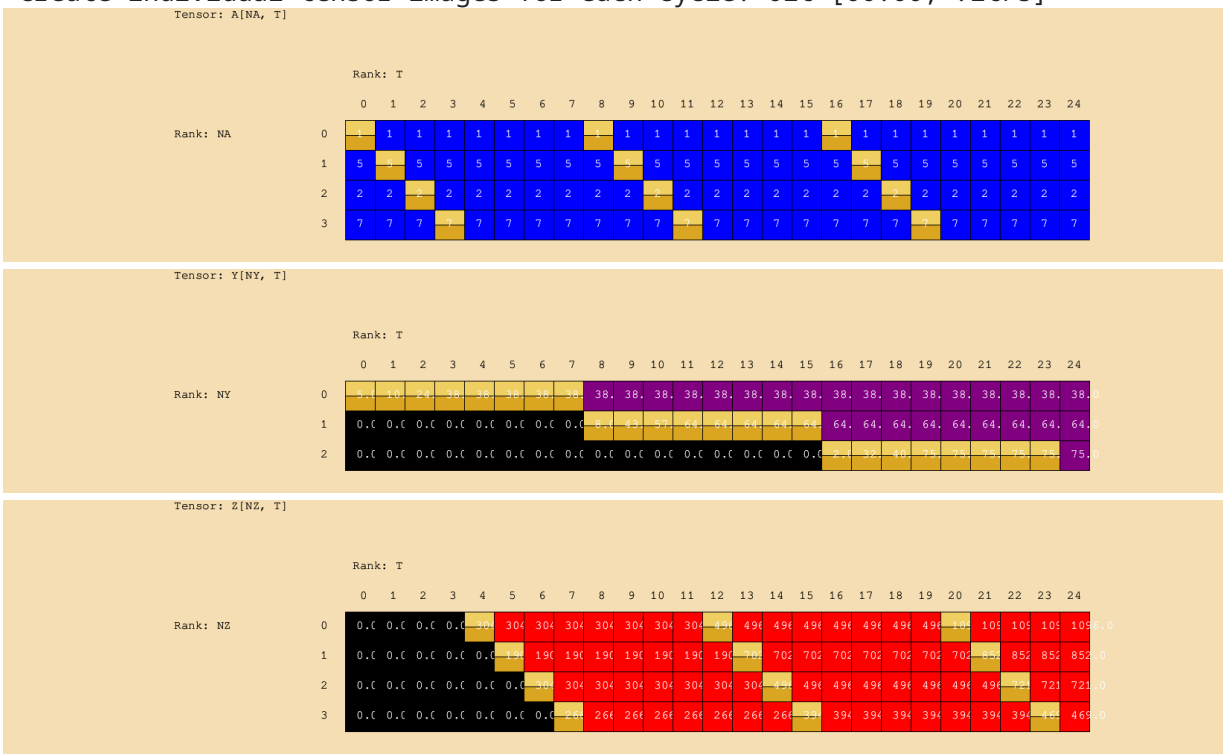
Out[12]:



In [13]:
```
tensor_name_to_fibertree_tensor = make_tensors_from_spec(spec)

A = tensor_name_to_fibertree_tensor["A"]
Y = tensor_name_to_fibertree_tensor["Y"]
Z = tensor_name_to_fibertree_tensor["Z"]
B = tensor_name_to_fibertree_tensor["B"]
B.setColor("green")
C = tensor_name_to_fibertree_tensor["C"]
C.setColor("green")
```

```
a = A.getRoot()
y = Y.getRoot()
z = Z.getRoot()
b = B.getRoot()
c = C.getRoot()


canvas = createCanvas(A, Y, Z)
for ny in range(3):
    for na in range(4):
        y[ny] += a[na]*b[na][ny]
        canvas.addActivity([(na,)], [(ny,)], [], worker="In GlobalBuffer")
        canvas.addFrame([(na,)], [(ny,)], [])
    for nz in range(4):
        z[nz] += y[ny]*c[ny][nz]
        canvas.addActivity([], [(ny,)], [(nz,)], worker="In GlobalBuffer")
        canvas.addFrame([], [(ny,)], [(nz,)])

displayCanvas(canvas)
```

```
Starting simulation
Finished simulation
Spacetime
Create individual tensor images for each cycle: 0it [00:00, ?it/s]
```

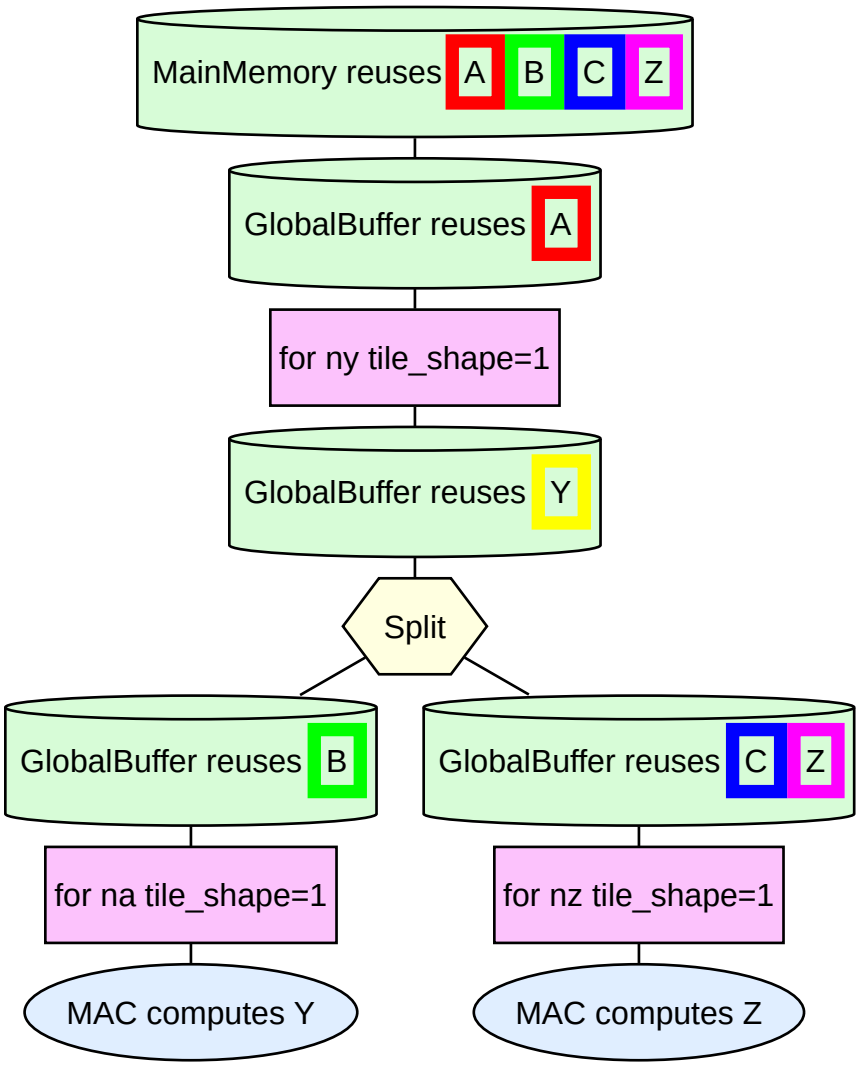


But lifetime decision is a trade-off impacted by dataplacement choices. For example, we may decide to keep A across the two Einsums to achieve more reuse in GlobalBuffer.

In [14]:
```
spec = af.Spec.from_yaml(
    af.examples.arches.simple,
    af.examples.workloads.matvecs,
```

```
    af.examples.mappings.fused_matvecs_to_simple_tiled_reuse_A,
)
spec.mapping
```

Out[14]:



You can see the animation below.

In [15]:

```
tensor_name_to_fibertree_tensor = make_tensors_from_spec(spec)

A = tensor_name_to_fibertree_tensor["A"]
Y = tensor_name_to_fibertree_tensor["Y"]
Z = tensor_name_to_fibertree_tensor["Z"]
B = tensor_name_to_fibertree_tensor["B"]
B.setColor("green")
C = tensor_name_to_fibertree_tensor["C"]
C.setColor("green")

a = A.getRoot()
y = Y.getRoot()
z = Z.getRoot()
b = B.getRoot()
c = C.getRoot()
```

```
canvas = createCanvas(A, Y, Z)
for ny in range(3):
    for na in range(4):
        y[ny] += a[na]*b[na][ny]
        canvas.addActivity([(nat,) for nat in range(4)], [(ny,)], [], worker
        canvas.addFrame([(na,)], [(ny,)], [])
    for nz in range(4):
        z[nz] += y[ny]*c[ny][nz]
        canvas.addActivity([(nat,) for nat in range(4)], [(ny,)], [(nz,)], w
        canvas.addFrame([], [(ny,)], [(nz,)])

displayCanvas(canvas)
```

```
Starting simulation
Finished simulation
Spacetime
Create individual tensor images for each cycle: 0it [00:00, ?it/s]
```