# User Level Failure Mitigation in MPI

Wesley Bland

MPI Forum Fault Tolerance Working Group

Resilience Workshop

8/28/12

# Motivation

- Failures are common enough that we want to handle them but rare enough that failure free performance is most important right now.

- Want plan for managing failures to be
  - Lightweight
  - Efficient
  - Extensible


"...faults will be continuous and across all parts of the hardware and software layers, which will require new programming paradigms."

# How to Minimize Cost

# How to Minimize Cost

Collective operations return uniformly

```
for (i = 0; i < n; i++) {
  value = compute();
  rc = MPI_Bcast(comm, value);
  if (rc!=MPI_SUCCESS)
    recover();
}
```

# How to Minimize Cost

Collective operations return uniformly

```
for (i = 0; i < n; i++) {

  value = compute();

  rc = MPI_Bcast(comm, value);

  if (rc!=MPI_SUCCESS)

    recover();

}
```

Bcast agrees on return code

$O(n^2)$

# How to Minimize Cost

Collective operations return uniformly

```
for (i = 0; i < n; i++) {
  value = compute();
  rc = MPI_Bcast(comm, value);
  if (rc!=MPI_SUCCESS)
    recover();
}
```

Bcast agrees on return code

$O(n^2)$

Operations notify of any process failure

```
while(morework) {
  rc = MPI_Recv(myrank-1);
  if (rc!=MPI_SUCCESS) recover();
  rc = MPI_Send(myrank+1);
  if (rc!=MPI_SUCCESS) recover();
}
```

# How to Minimize Cost

myrank-2

Collective operations return uniformly

```
for (i = 0; i < n; i++) {
  value = compute();
  rc = MPI_Bcast(comm, value);
  if (rc!=MPI_SUCCESS)
    recover();
}
```

Bcast agrees on return code

$O(n^2)$

Operations notify of any process failure

```
while(morework) {
  rc = MPI_Recv(myrank-1);
  if (rc!=MPI_SUCCESS) recover();
  rc = MPI_Send(myrank+1);
  if (rc!=MPI_SUCCESS) recover();
}
```

Failures must be reported during unrelated operations

# Two Guiding Principles

- A correct MPI program must not deadlock because of process failures.

- An MPI call that does not involve a failed process will complete normally unless we specifically intervene.

# Simple Master/Worker

- FT Method
  - Check Return Codes
- New MPI Constructs
  - MPI_ERR_PROC_FAILED

```
MPI_Irecv(/* all workers */);
while (work_available &&
       active_workers > 0
  rc = MPI_Waitany(&worker);
  if (MPI_ERR_PROC_FAILED == rc)
    active_workers--;
    /* Recover Lost Work */
else
    /* Send new work to worker */
    MPI_Irecv(worker);
```

# Failure Acknowledgement

- MPI_COMM_FAILURE_ACK( *comm* )
- MPI_COMM_FAILURE_GET_ACKED( comm, &group )

- Discover the group of process which are known to have failed in **comm**
- Unmatched MPI_ANY_SOURCE receptions will not return failure for the processes in **group**
- **Group** is updated during the next call to ACK

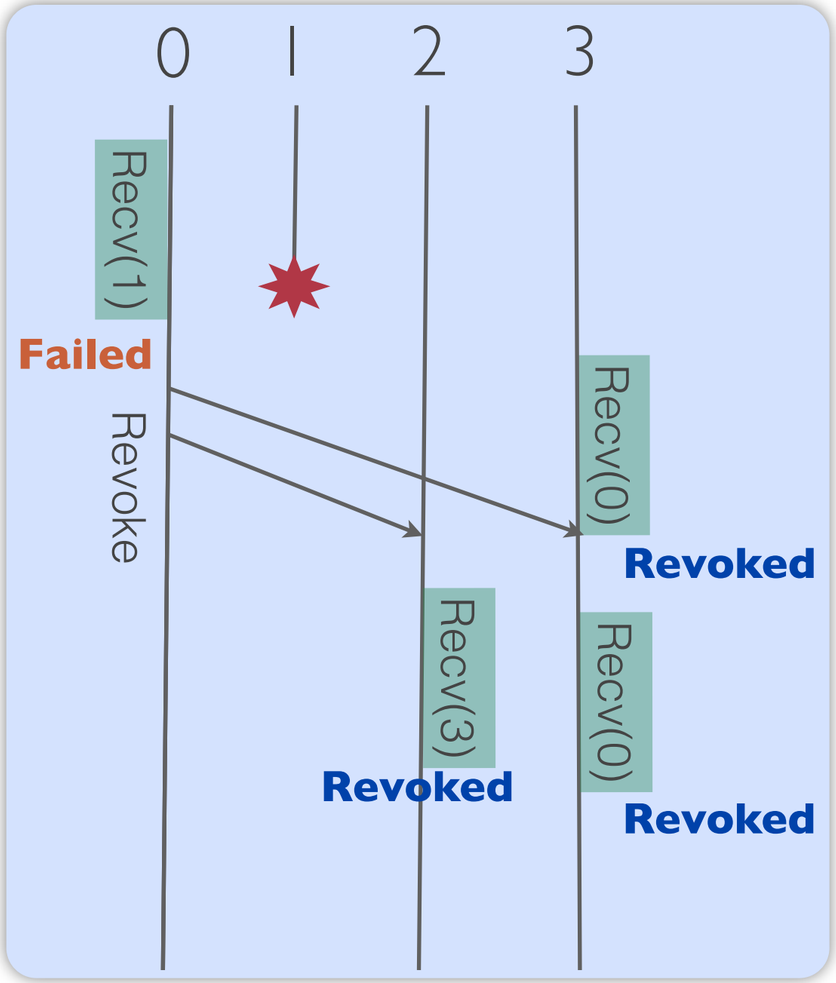
# Slightly harder...

- FT Method
  - Check return codes (MPI_ERR_PENDING)
  - Acknowledge known failures to continue with existing processes
- New MPI Constructs
  - MPI_Comm_failure_ack
  - MPI_Comm_failure_get_acked

```
MPI_Irecv(MPI_ANY_SOURCE);

while (work_available &&

       active_workers > 0)

    rc = MPI_Wait();

if (MPI_ERR_PENDING == rc
    || MPI_ERR_PROC_FAILED == rc)

  MPI_Comm_failure_ack(comm);

  MPI_Comm_failure_get_acked(comm,
  &group);

  MPI_Group_size(group, &gsize);

  active_workers -= /* Size difference */

else

  MPI_Irecv(MPI_ANY_SOURCE);
```

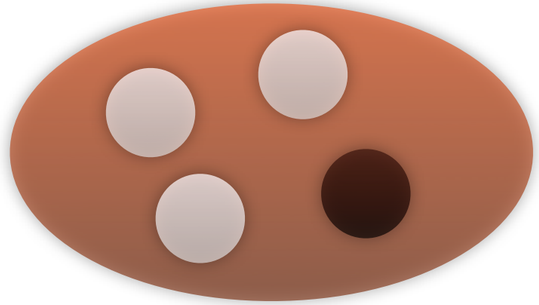
# MPI_COMM_REVOKE(comm)

- Non-collective operation
- All non-local operations complete with error
- New operations will always return an error
- Communicator is revoked when the local process calls REVOKE or returns MPI_ERR_REVOKE because another process called it



# MPI_COMM_SHRINK(comm, comm2)

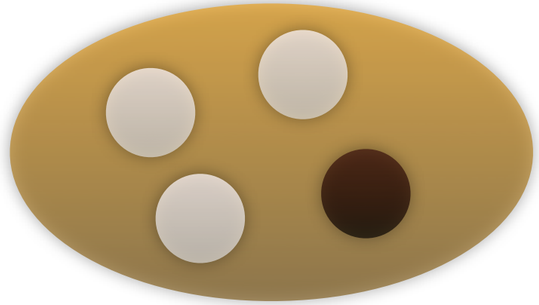
- Creates a new communicator from a revoked one
- Collective over non-failed processes
- Will not return an error due to process failure
- Results in two communicators
  - Dead & Shrunken



# MPI_COMM_SHRINK(comm, comm2)

- Creates a new communicator from a revoked one
- Collective over non-failed processes
- Will not return an error due to process failure
- Results in two communicators
  - Dead & Shrunken

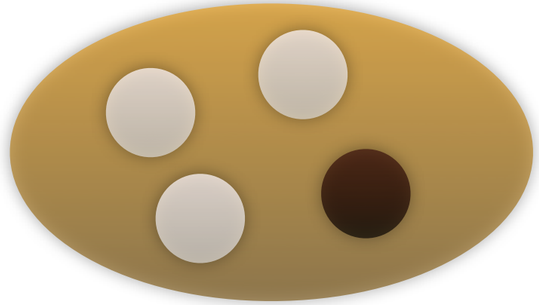
MPI_Revoke



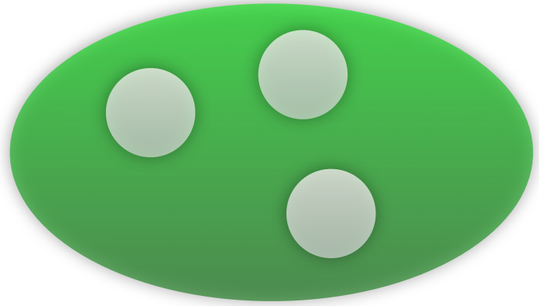
# MPI_COMM_SHRINK(comm, comm2)

- Creates a new communicator from a revoked one
- Collective over non-failed processes
- Will not return an error due to process failure
- Results in two communicators
  - Dead & Shrunken

MPI_Revoke

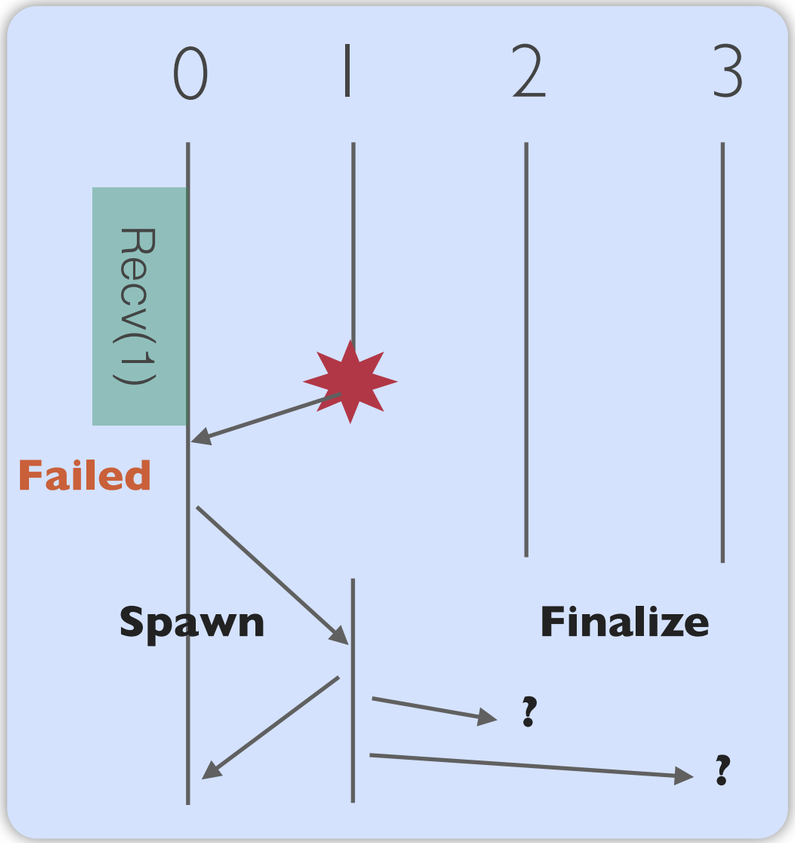


MPI_Shrink



# MPI_COMM_AGREE(comm, flag)

- All living processes perform logical AND over ***flag***
- Failed processes do not contribute
- Will not return error due to process failure
- Revoked communicators uniformly return MPI_ERR_REVOKED



# Error Codes

- MPI_ERR_PROC_FAILED
  - Operation cannot complete because involved process is dead
- MPI_ERR_REVOKED
  - Communication object is revoked and will not work again
- MPI_ERR_PENDING
  - Non-blocking completion **may** not complete because involved process is dead

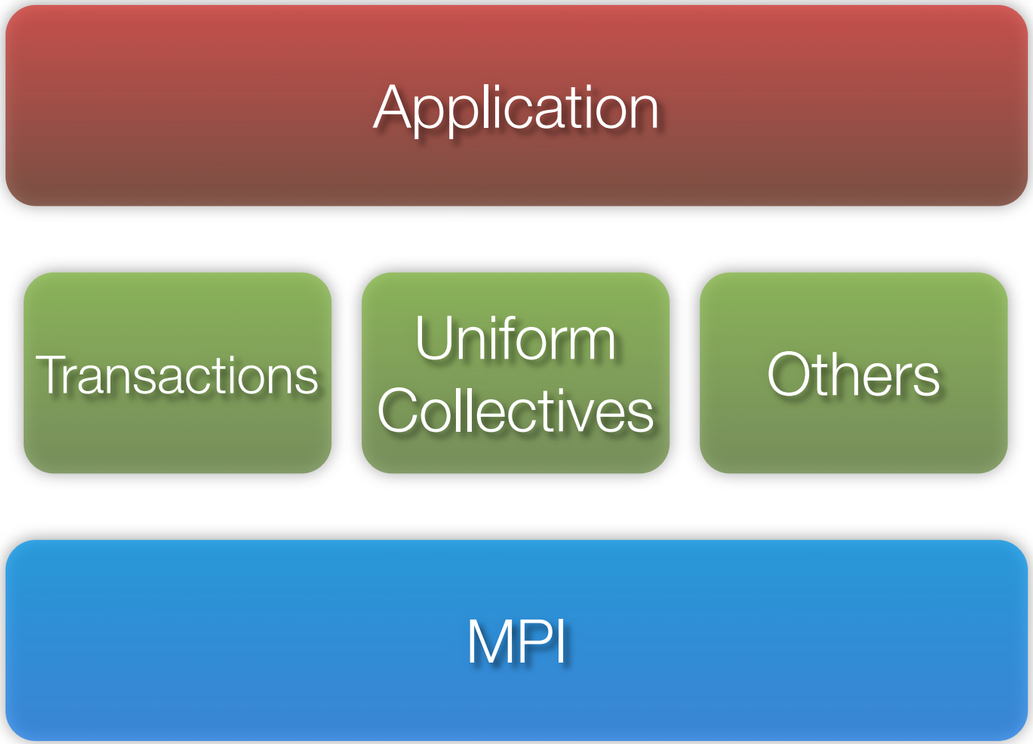
# Iterative Refinement

```
while( gnorm > epsilon ) {
  rc = MPI_Allreduce( &lnorm, &gnorm, 1, MPI_DOUBLE, MPI_MAX,
  comm);
  if( (MPI_ERR_PROC_FAILED == rc) ||
      (MPI_ERR_COMM_REVOKED == rc) ||
      (gnorm <= epsilon) ) {

    allsuceeded = (rc == MPI_SUCCESS);
    MPI_Comm_agree(comm, &allsuceeded);
    if( !allsucceeded ) {
      MPI_Comm_revoke(comm);
      MPI_Comm_shrink(comm, &comm2);
      MPI_Comm_free(comm);
      comm = comm2;
      gnorm = epsilon + 1.0;
```

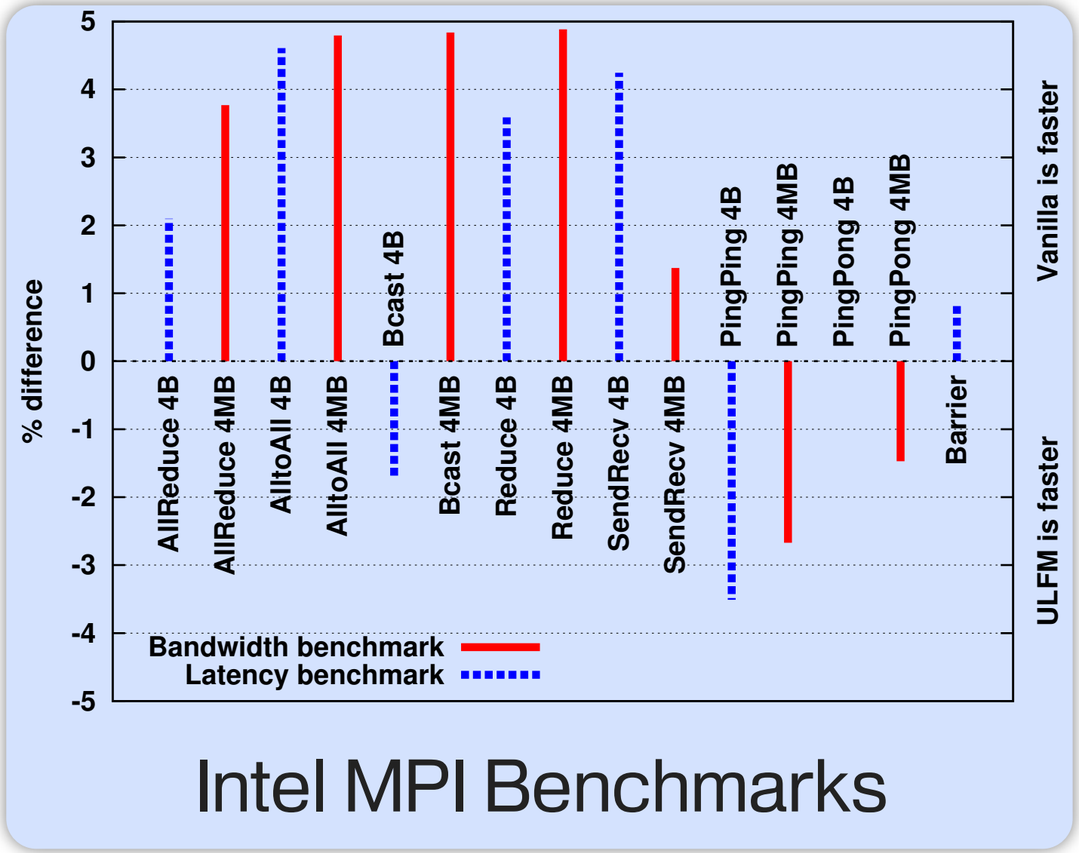
```
  rc = MPI_Allreduce( &lnorm, &gnorm, 1, MPI_DOUBLE, MPI_MAX,
  comm);
  if( (MPI_ERR_PROC_FAILED == rc) ||
      (MPI_ERR_COMM_REVOKED == rc) ||
      (gnorm <= epsilon) ) {

    allsuceeded = (rc == MPI_SUCCESS);
    MPI_Comm_agree(comm, &allsuceeded);
    if( !allsucceeded ) {
      MPI_Comm_revoke(comm);
      MPI_Comm_shrink(comm, &comm2);
      MPI_Comm_free(comm);
      comm = comm2;
      gnorm = epsilon + 1.0;
}}}
```

# This seems weak...

- Stronger consistency models can sit on top of MPI inside a library
- Most flexibility for the user
- No “silver bullet”

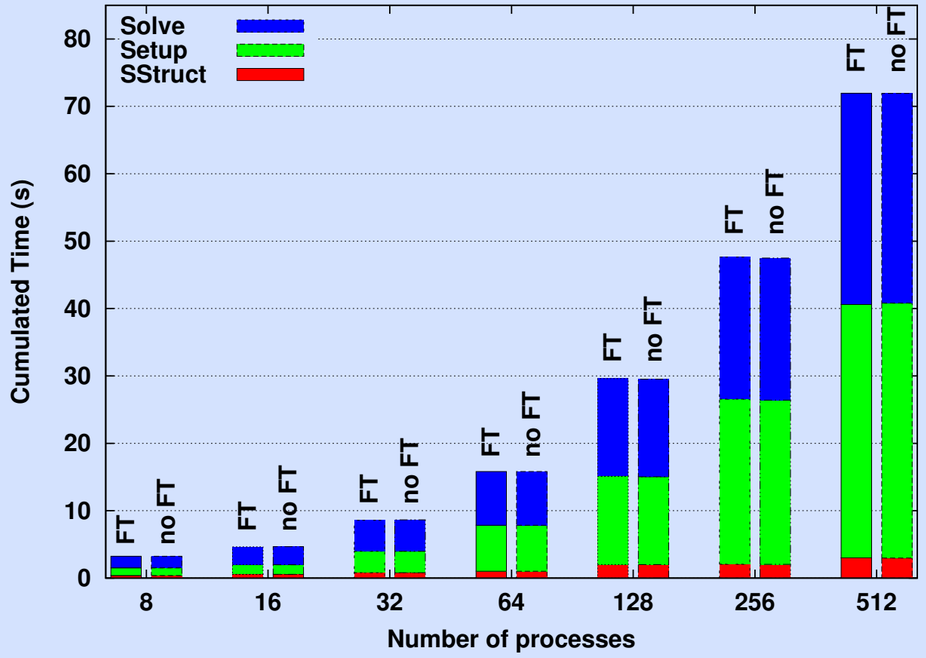


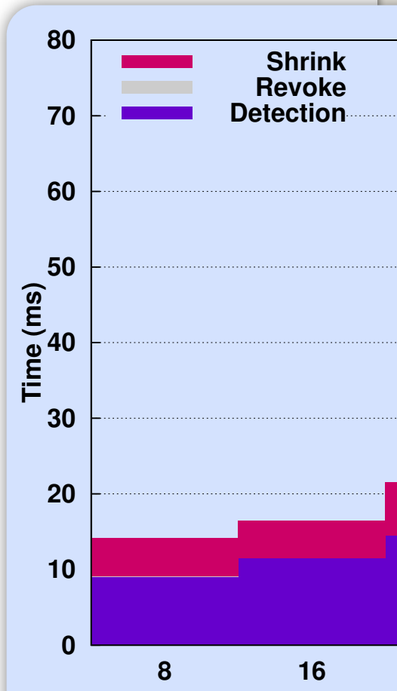
# ULFM Results




Intel MPI Benchmarks

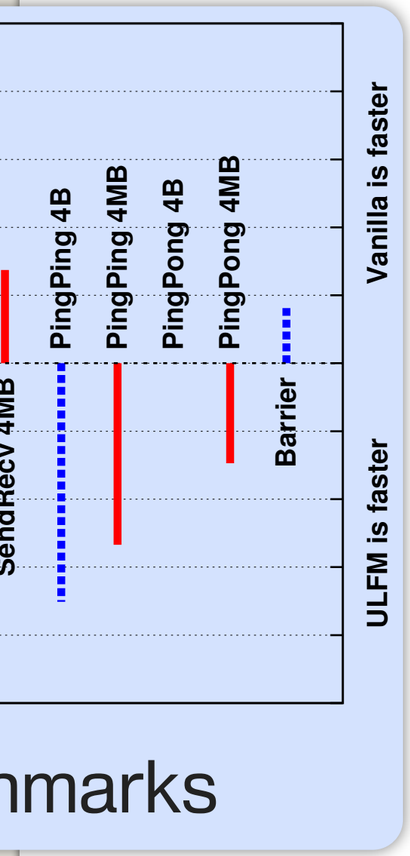
## Sequoia-AMG Benchmarks



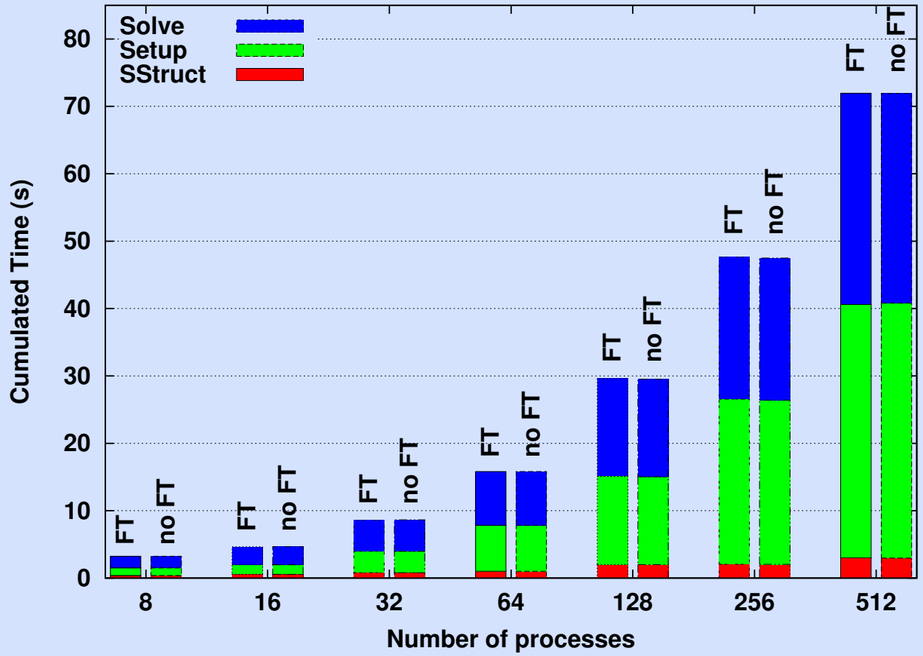

Synthe

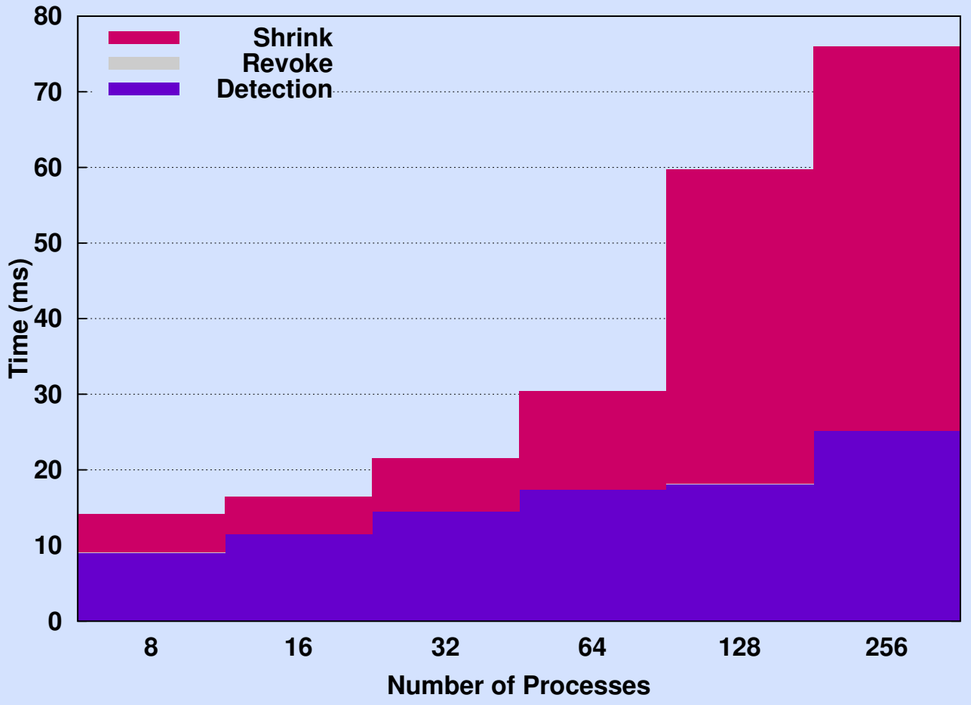
# ULFM Results




hmarks

## Sequoia-AMG Benchmarks




## Synthetic Benchmark

# How do we use this?

- 3 R's
- Library Composition
- Real Applications

# 3 R's of Application Recovery

- **Revoke**
  - Revoke all communicators involved with failed processor(s)
- **Repair**
  - Repair (or replace) the revoked communicators and recover the lost data
- **Resume**
  - Resume execution from the interrupted point (or the most recent recovery data)

# Library Composition

```
MAIN:
main() { }
repair() { }
```

```
MPI_Comm my_comm_world;
```

```
LIBRARY 1:
function() { }
repair() { }
```

```
MPI_Comm lib1_comm;
```

```
LIBRARY 2:
function() { }
repair() { }
```

```
MPI_Comm lib2_comm;
```

# Library Composition

```
MAIN:
main() { }
repair() { }
```

~~MPI_Comm my_comm_world;~~

```
LIBRARY 1:
function() { }
repair() { }
```

~~MPI_Comm lib1_comm;~~

```
LIBRARY 2:
function() { }
repair() { }
```

~~MPI_Comm lib2_comm;~~

# Library Composition

```
MAIN:
main() { }
repair() { }
```

```
MPI_Comm my_comm_world;
```

```
LIBRARY 1:
function() { }
repair() { }
```

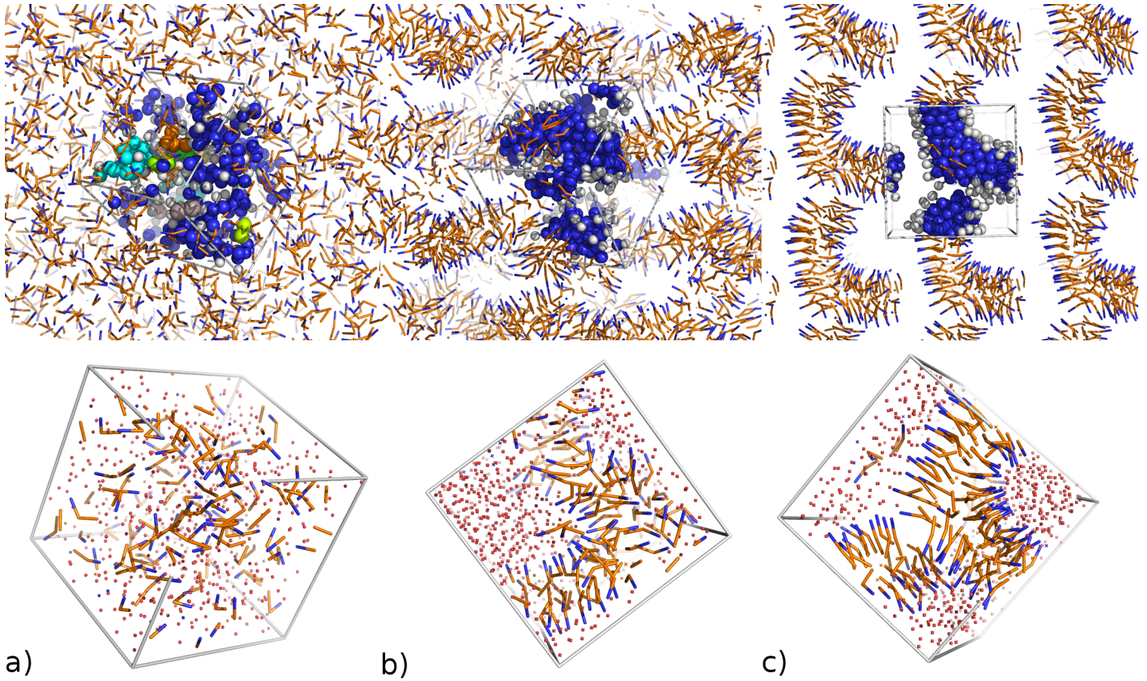
```
MPI_Comm lib1_comm;
```

```
LIBRARY 2:
function() { }
repair() { }
```

```
MPI_Comm lib2_comm;
```

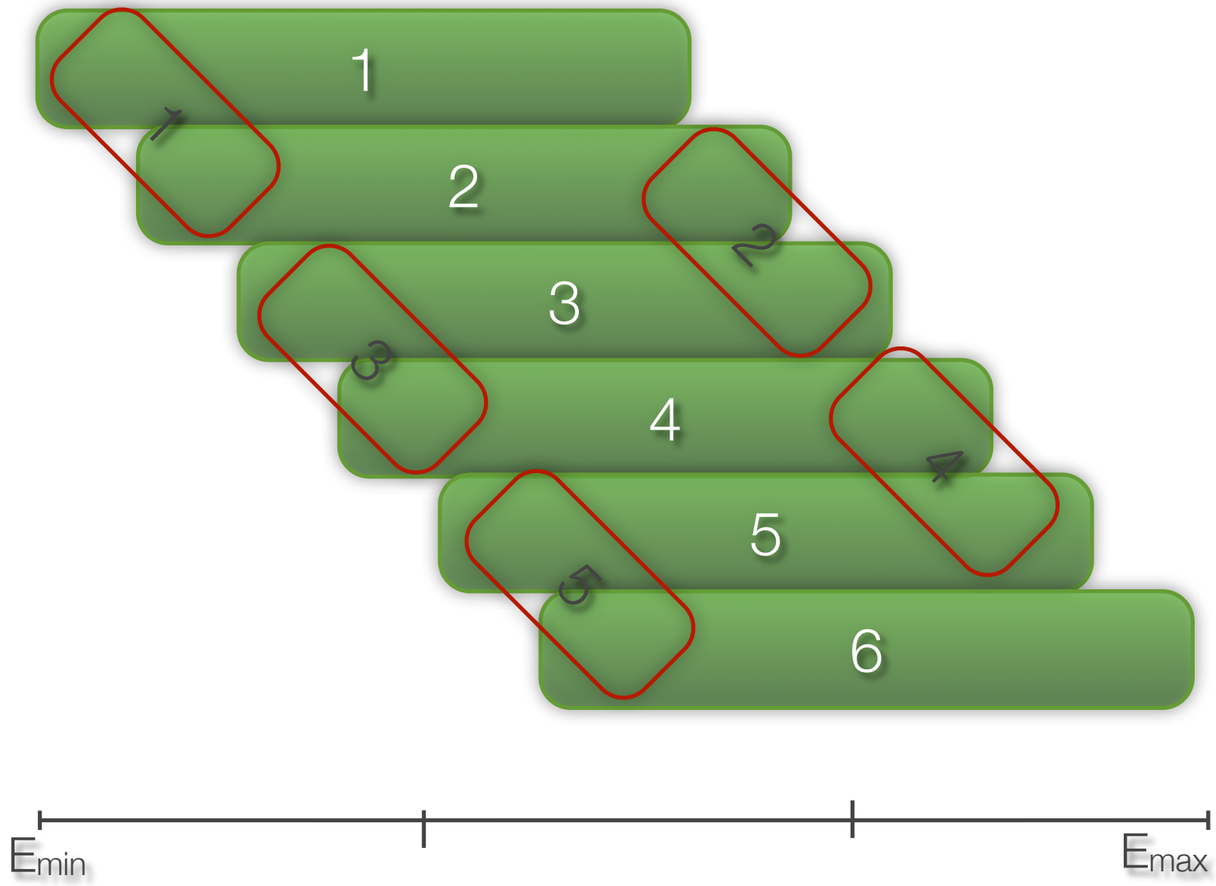
# Computation Chemistry Application

- Developed at the University of Georgia
- Simulating molecular interaction in skin cells
- Uses two types of communicators
  - Communication within an energy window
  - Communication between energy windows



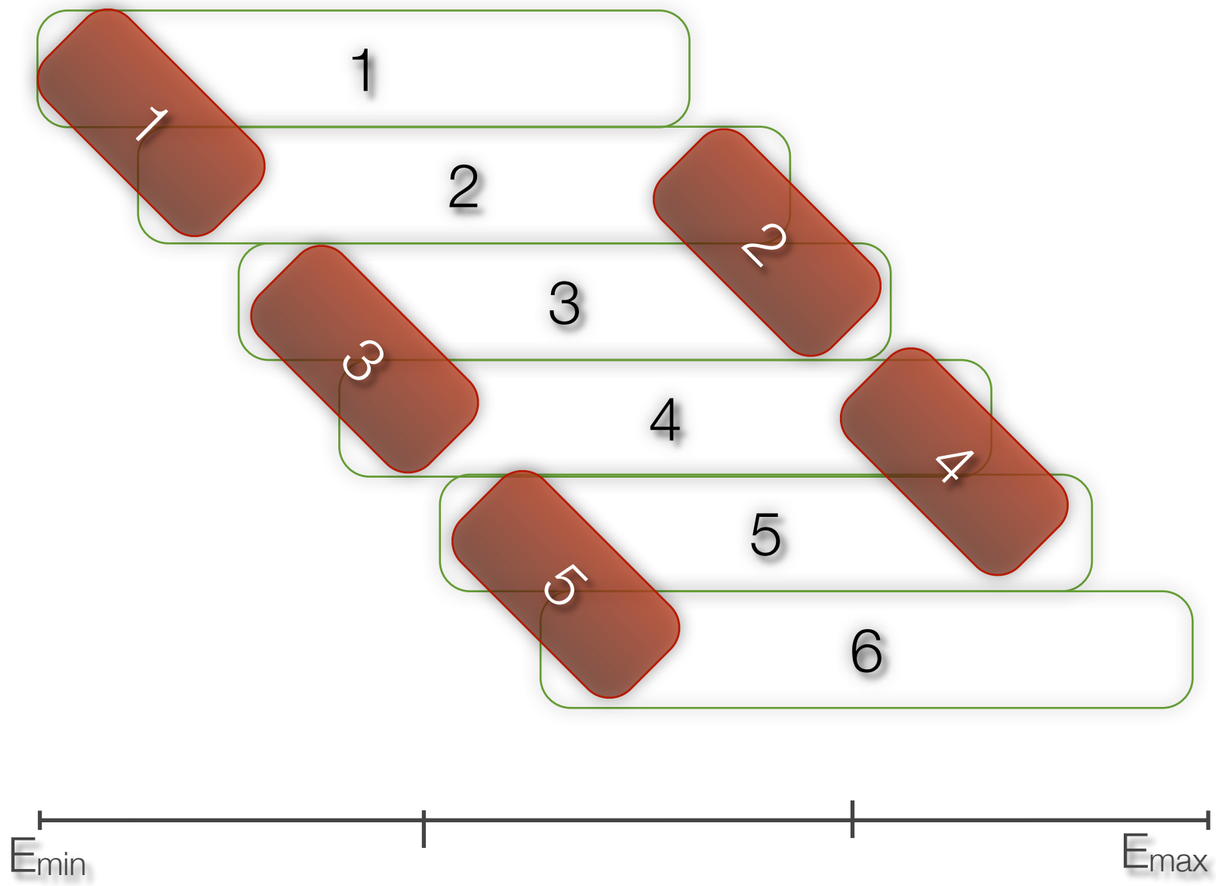
# Application Model

- 'Green' Communicator
  - Includes all processes in own energy window
  - Communication:
    - 2x MPI_Allreduce

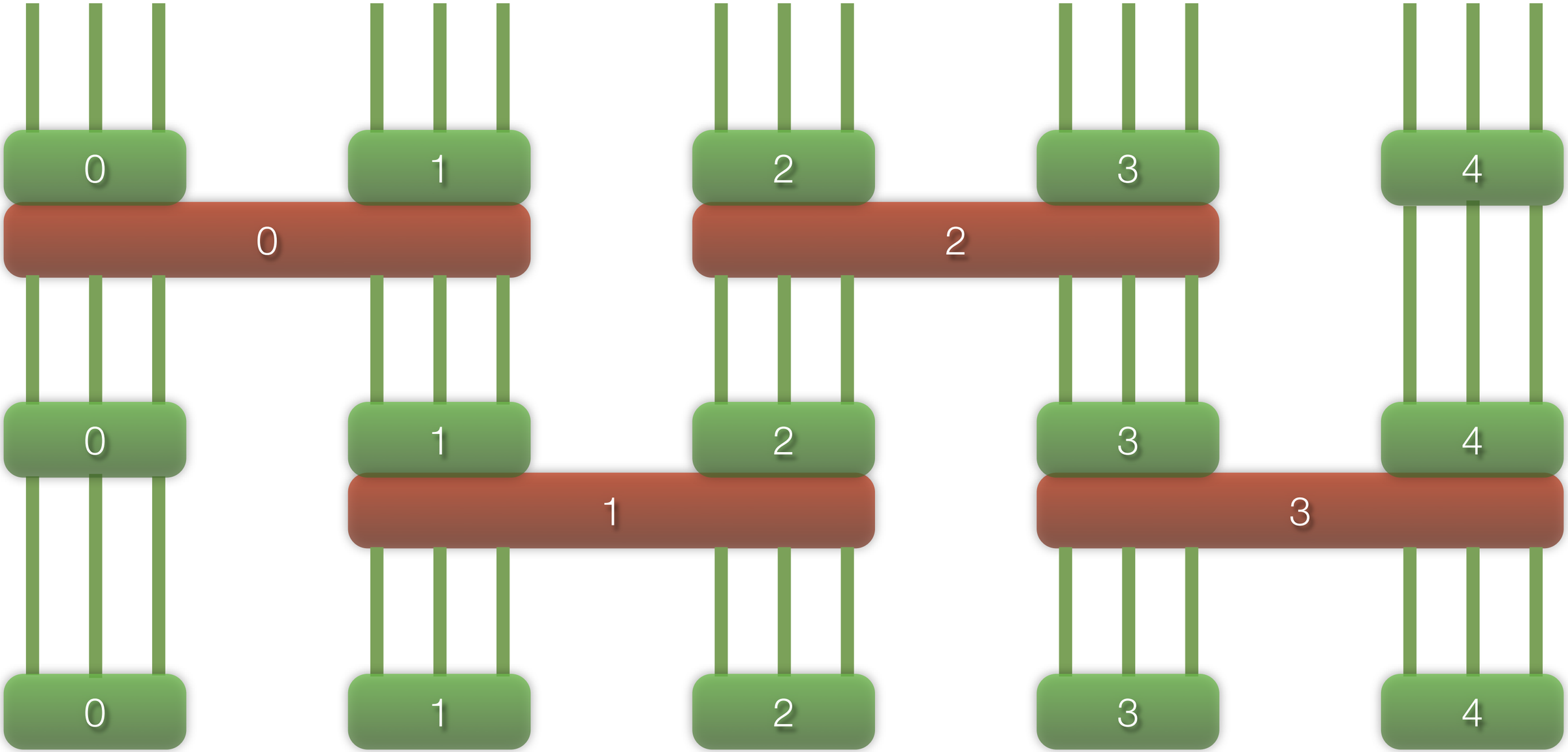


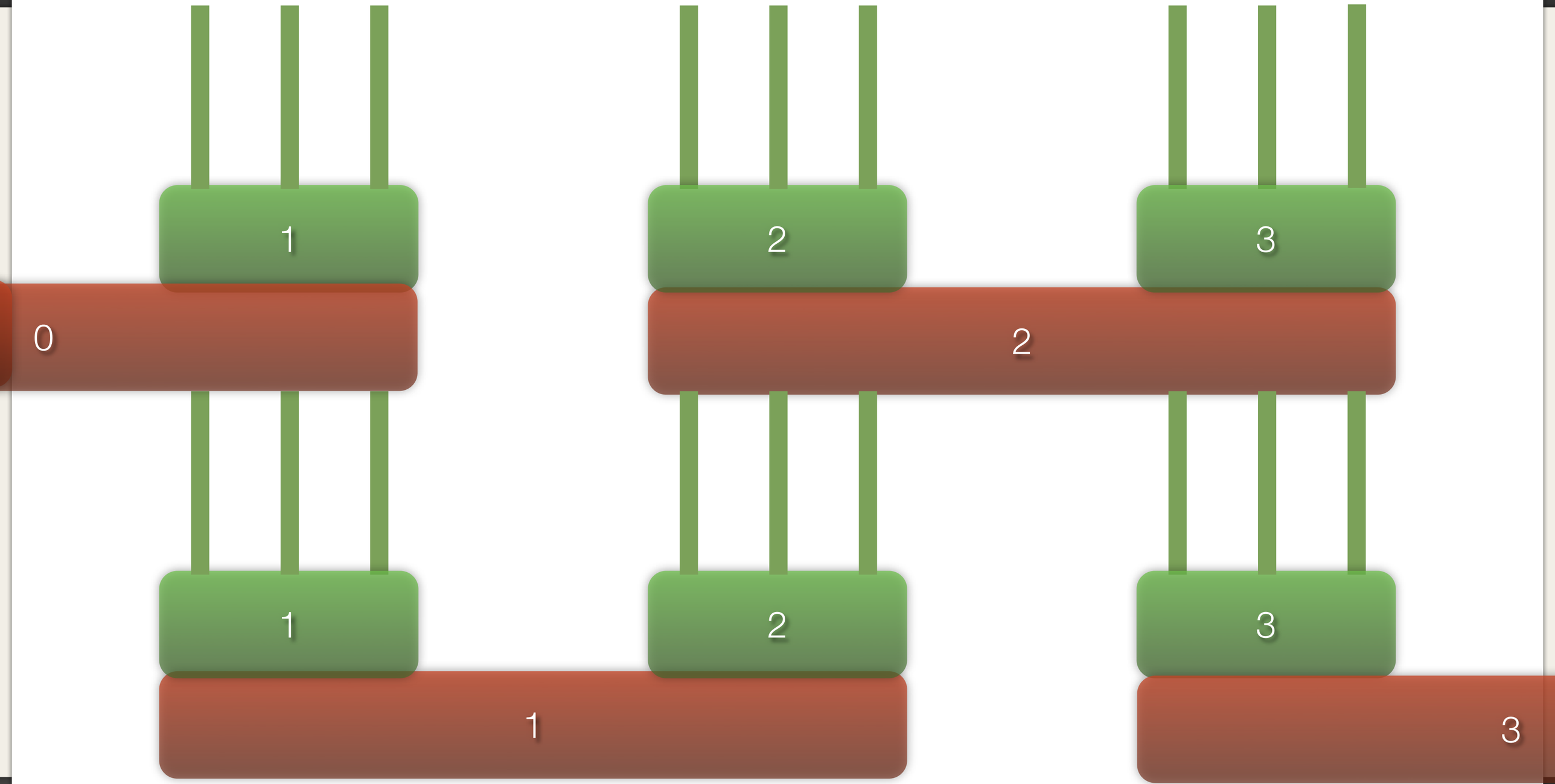
# Application Model

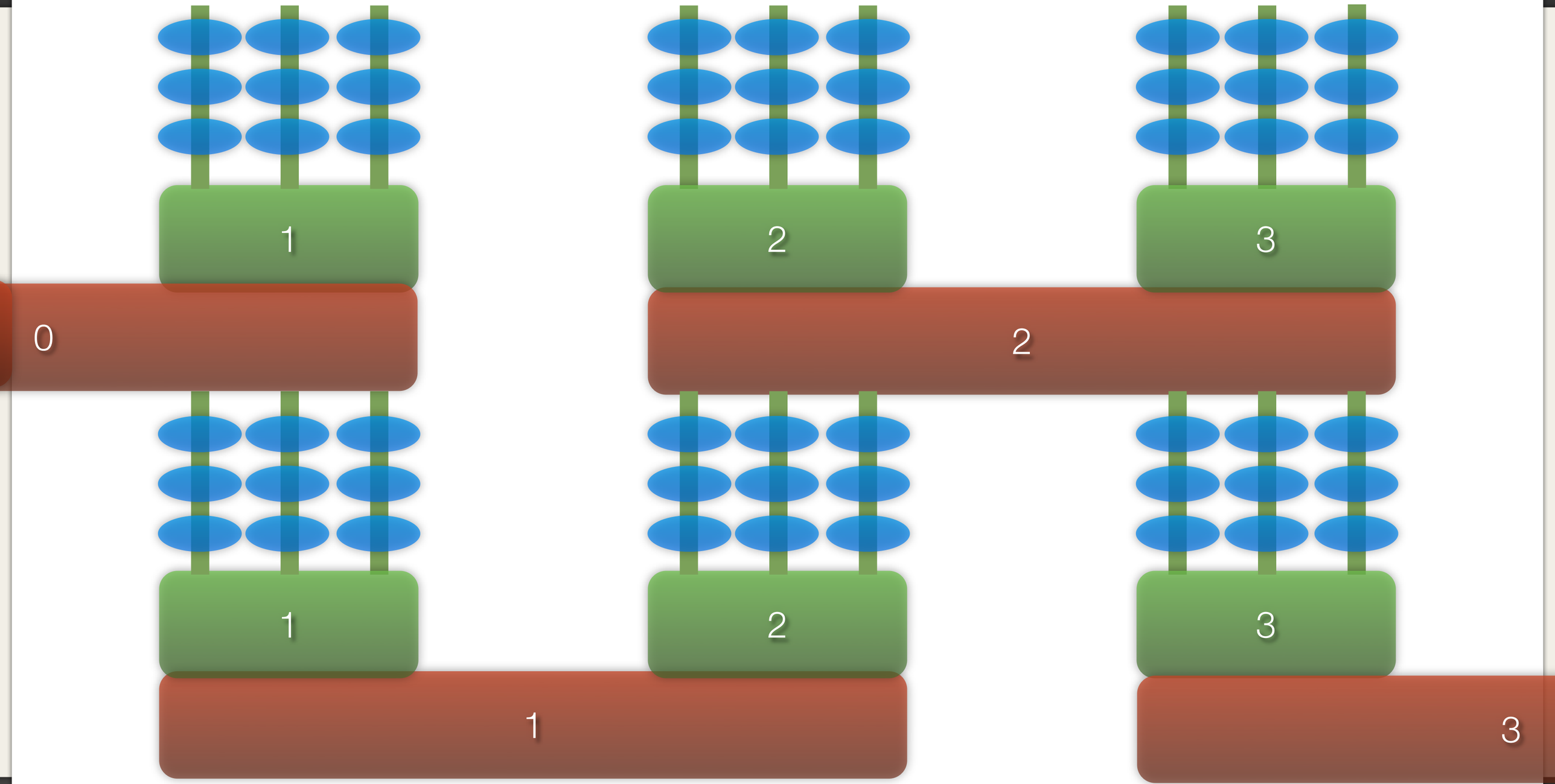
- 'Red' Communicator
  - Includes all processes from own and neighboring energy window
  - Communication:
    - 1x MPI_Scatter
    - 2x (MPI_Send_recv_replace)
    - Many MPI_Send & MPI_Recv

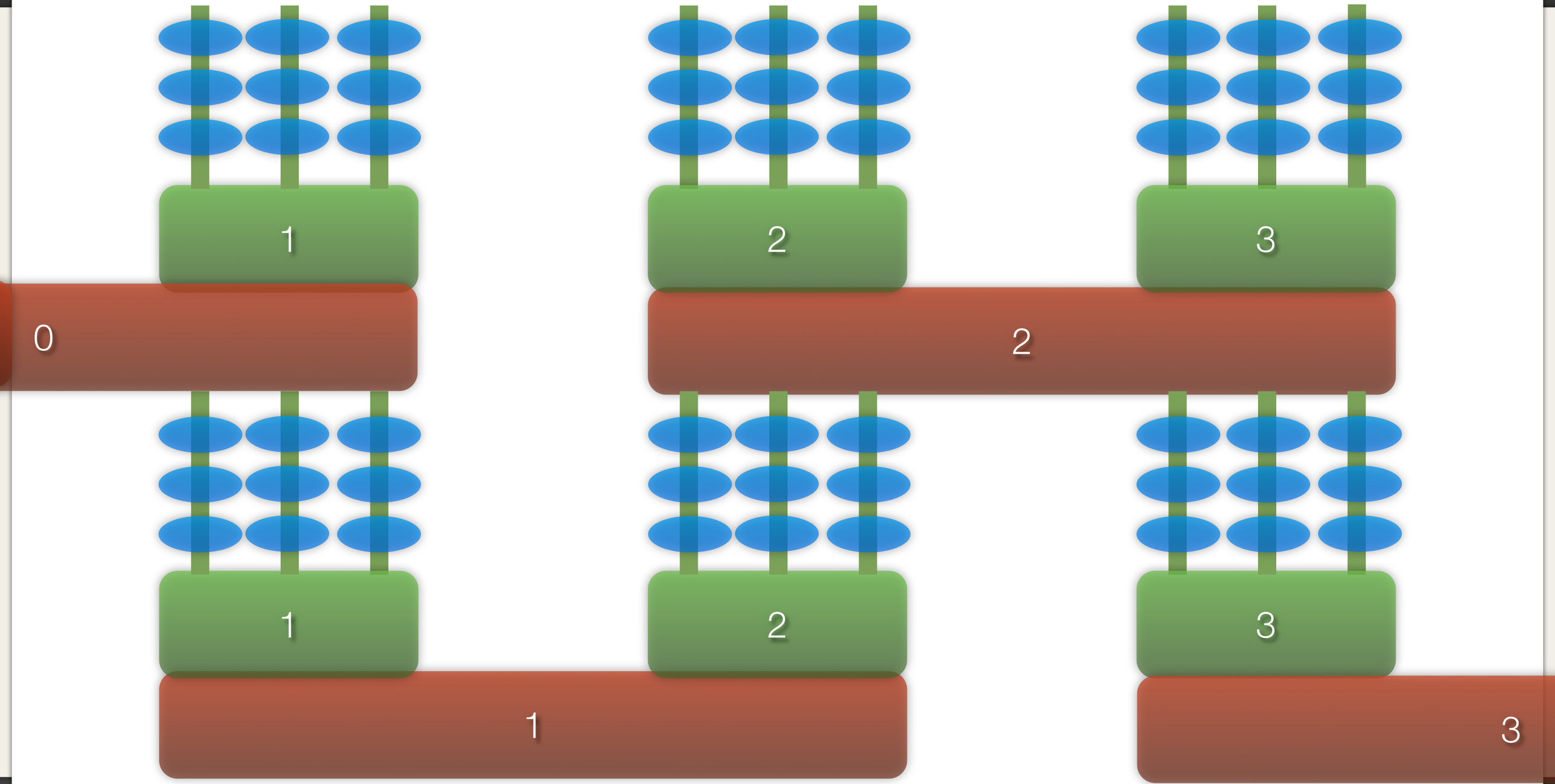


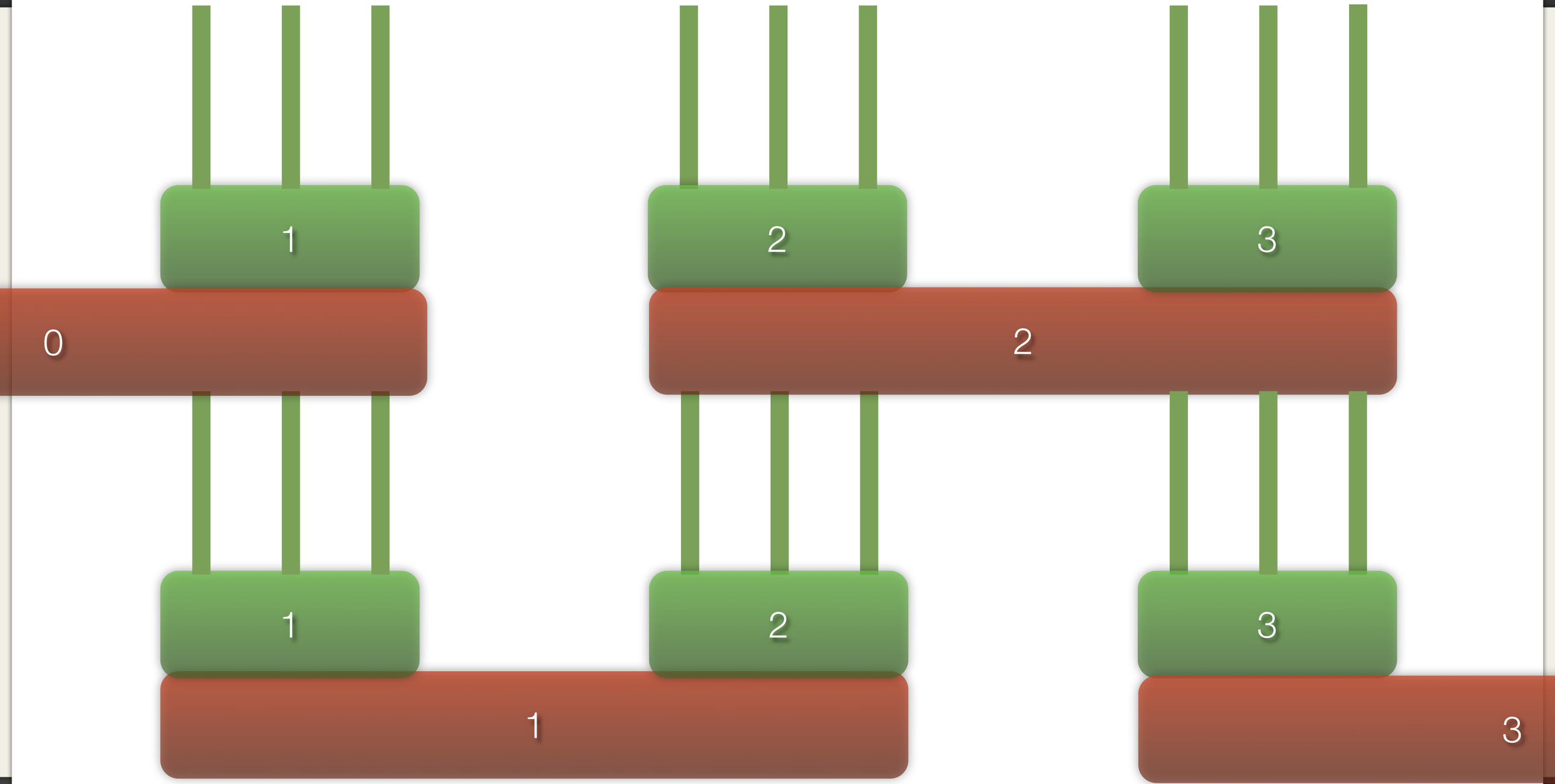
# Recovery Model

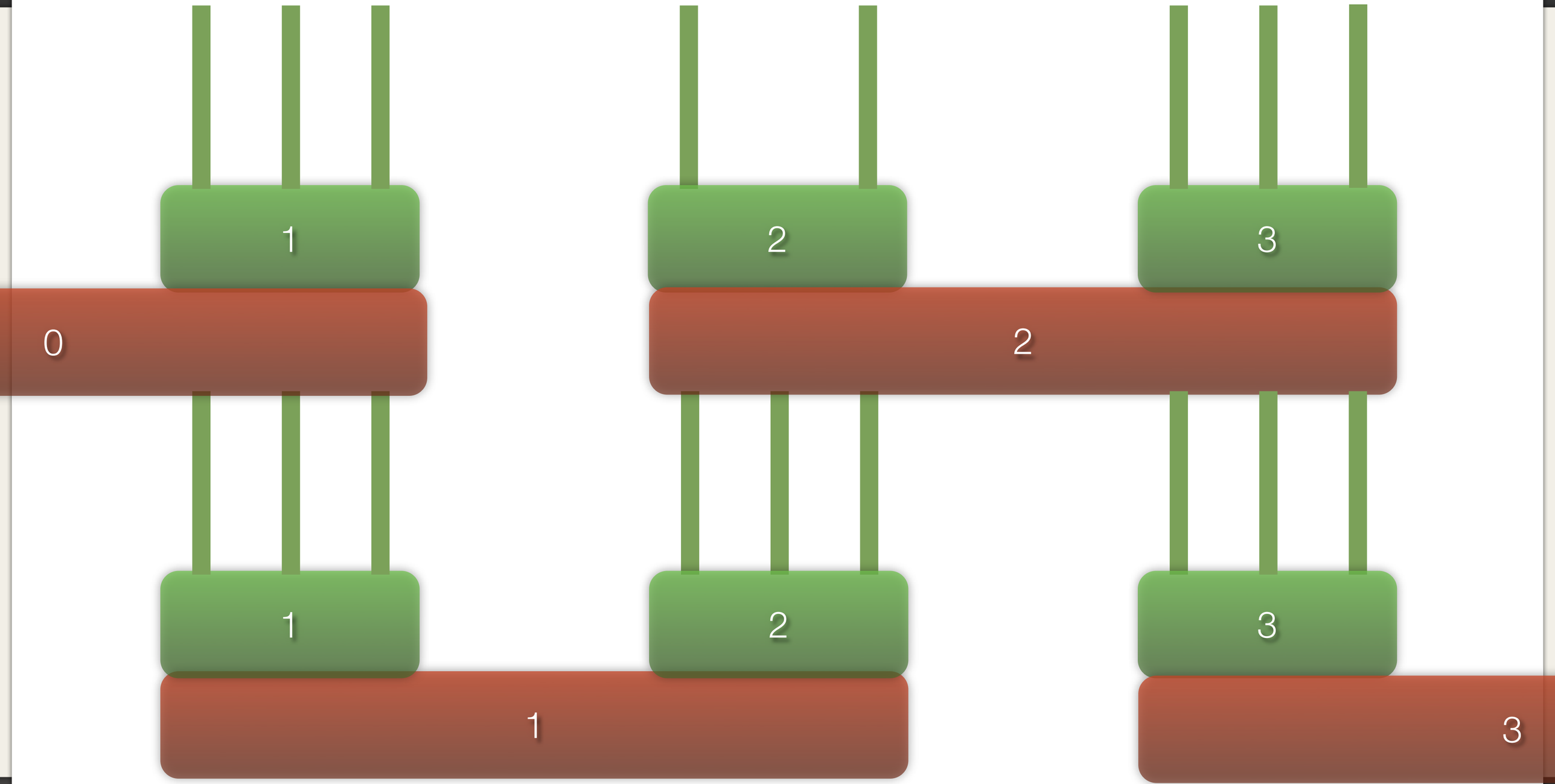


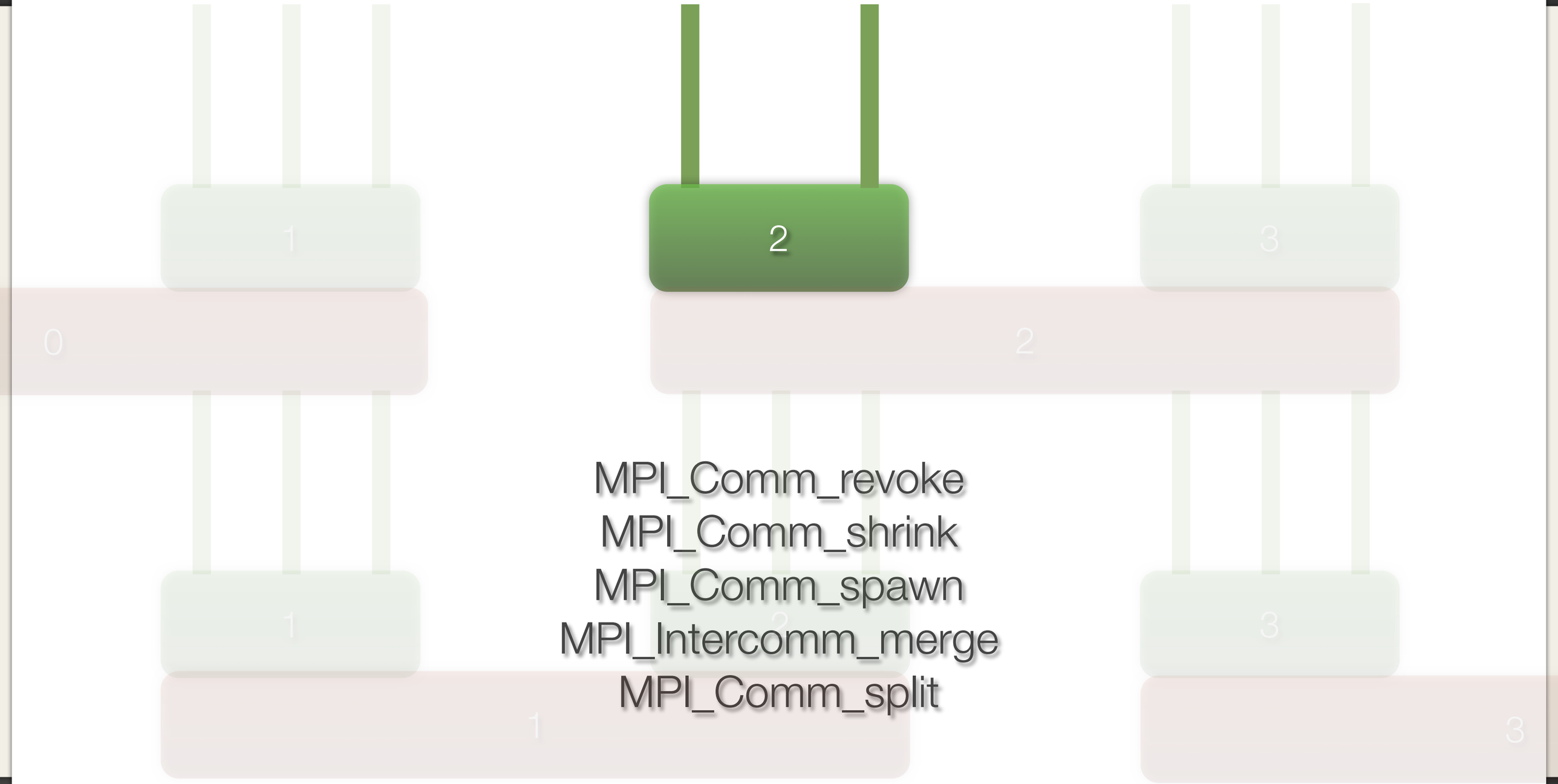

1
2
3
0
2
1
2
3
1
3

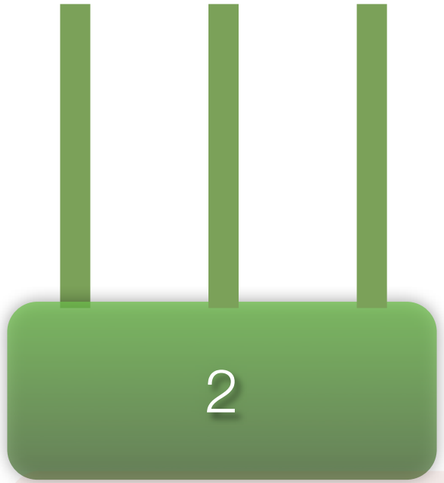

1
2
3
0
2
1
2
3
1
3


1
2
3
0
2
1
2
3
1
3

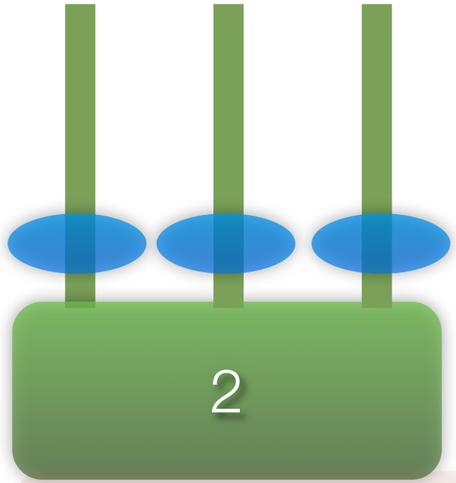

1
2
3
0
2
1
2
3
1
3


1
2
3
0
2
1
2
3
1
3

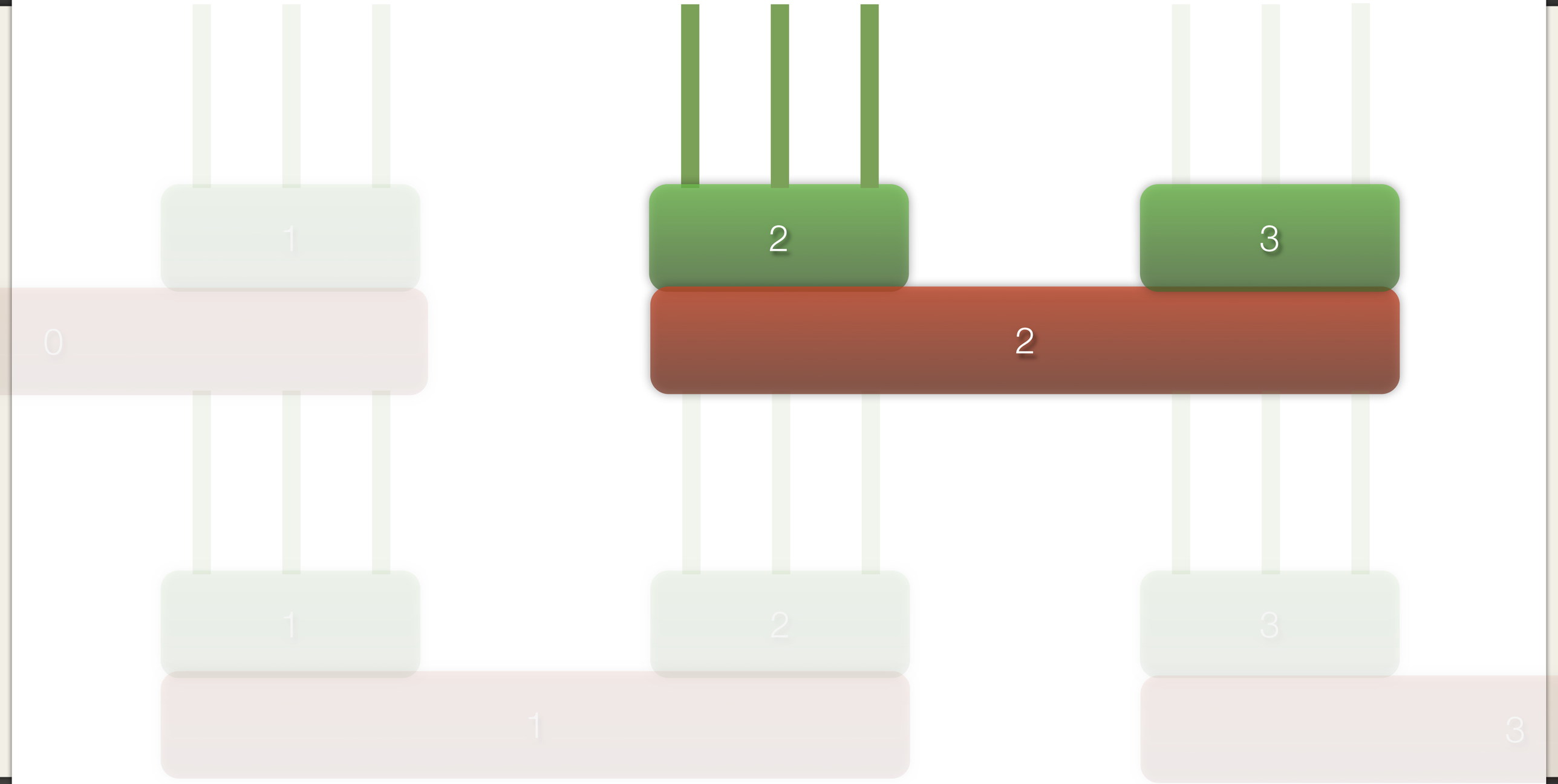

2
MPI_Comm_revoke
MPI_Comm_shrink
MPI_Comm_spawn
MPI_Intercomm_merge
MPI_Comm_split

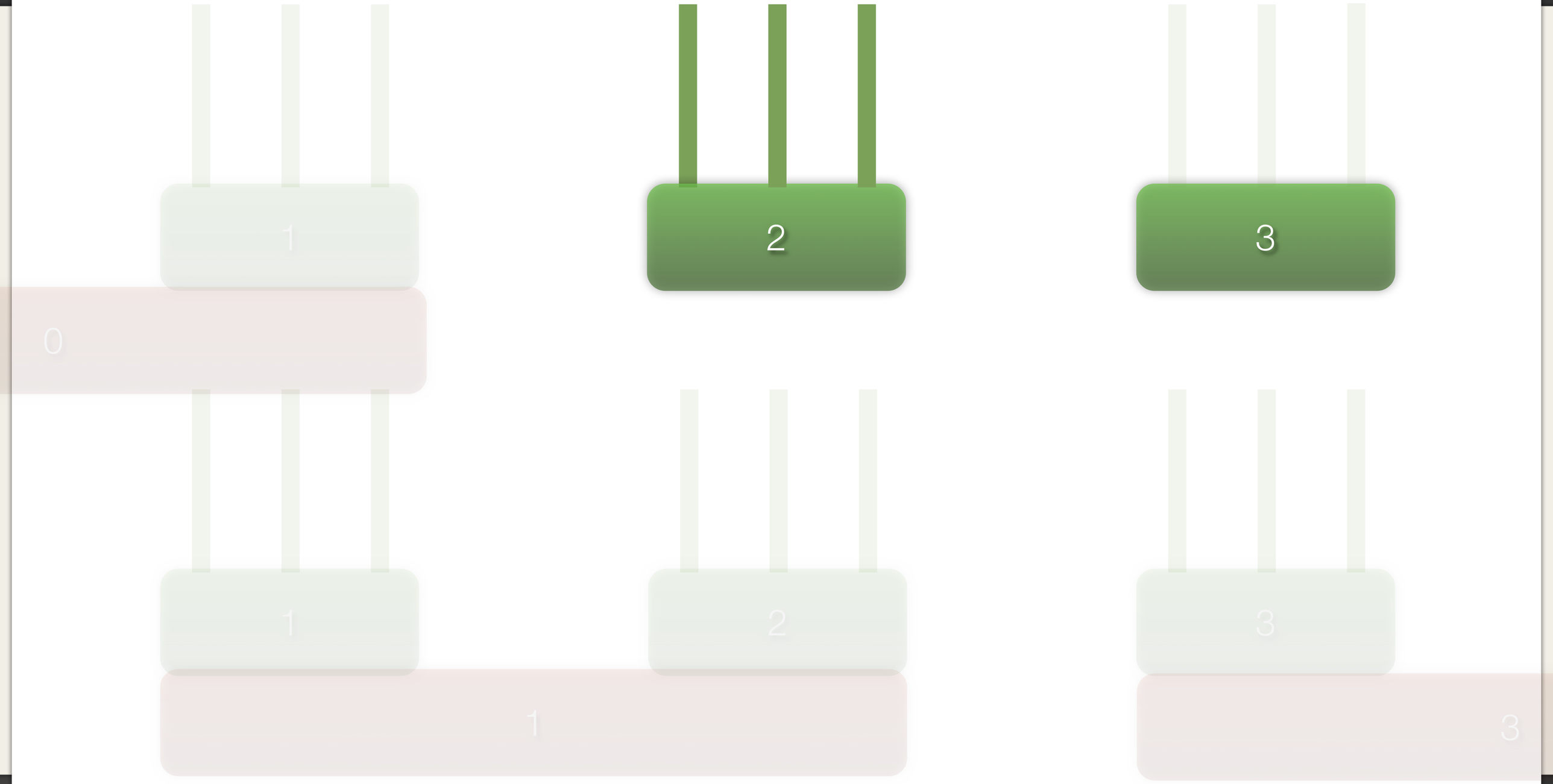


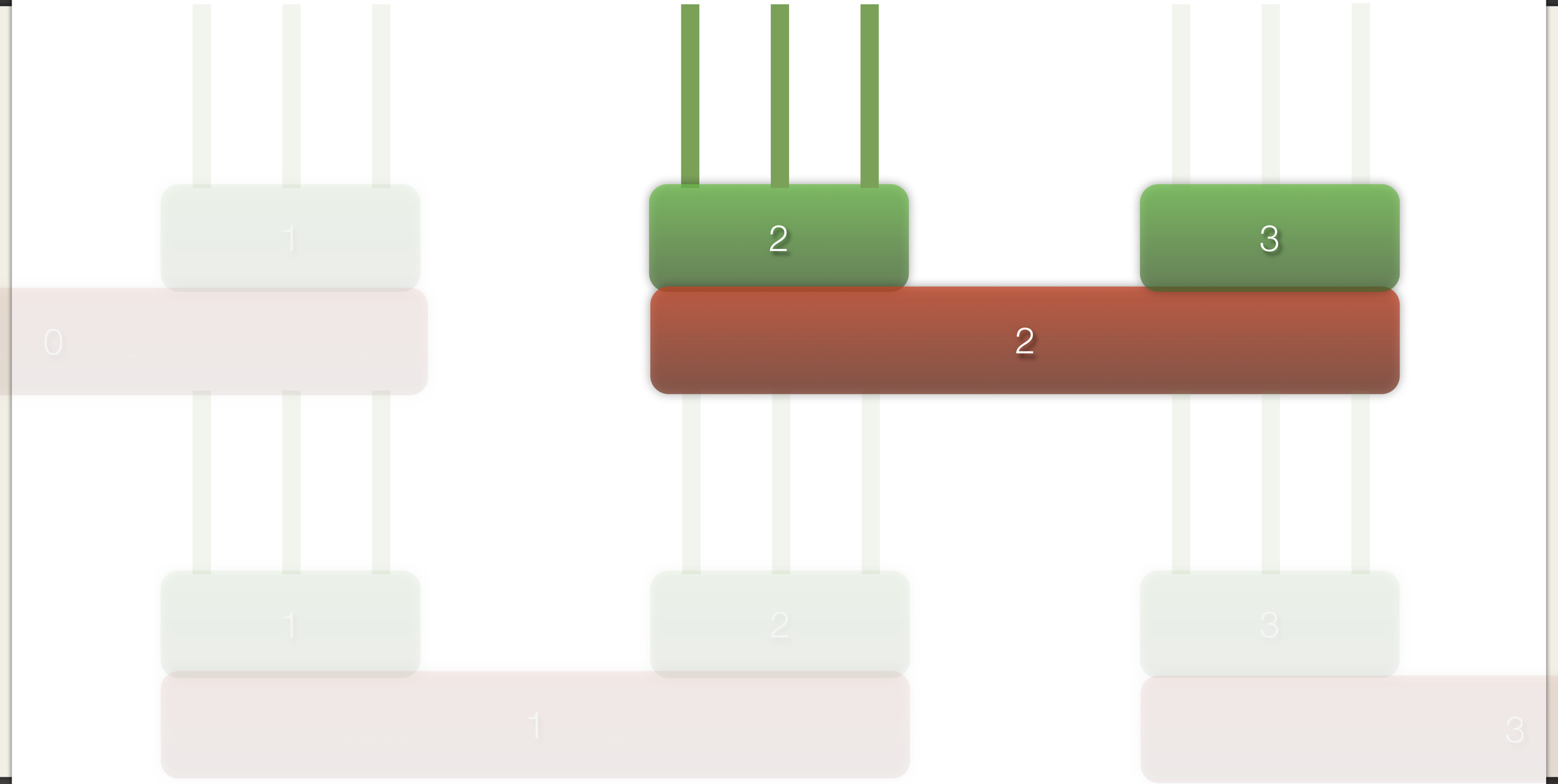

MPI_Comm_revoke
MPI_Comm_shrink
MPI_Comm_spawn
MPI_Intercomm_merge
MPI_Comm_split

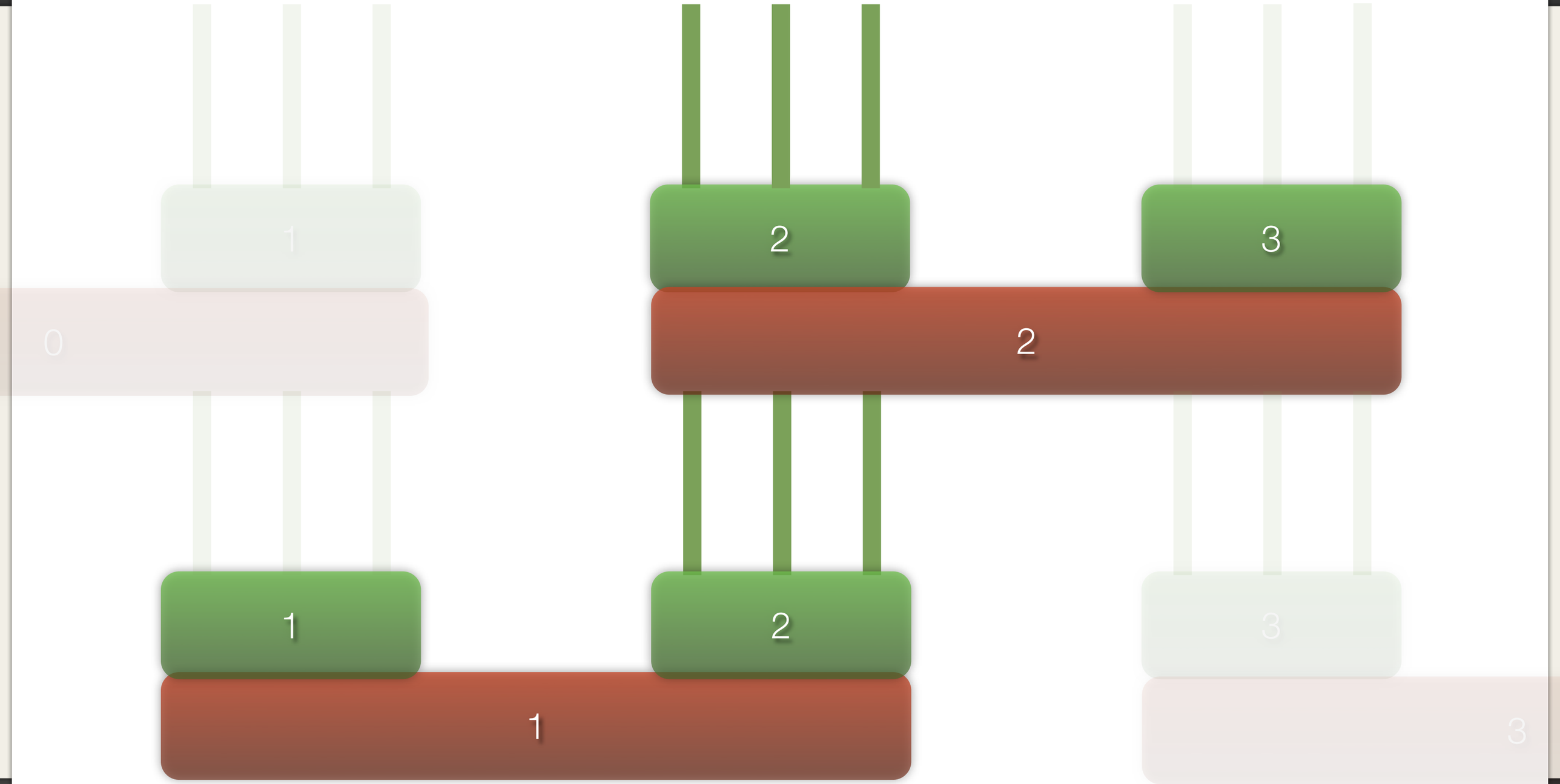


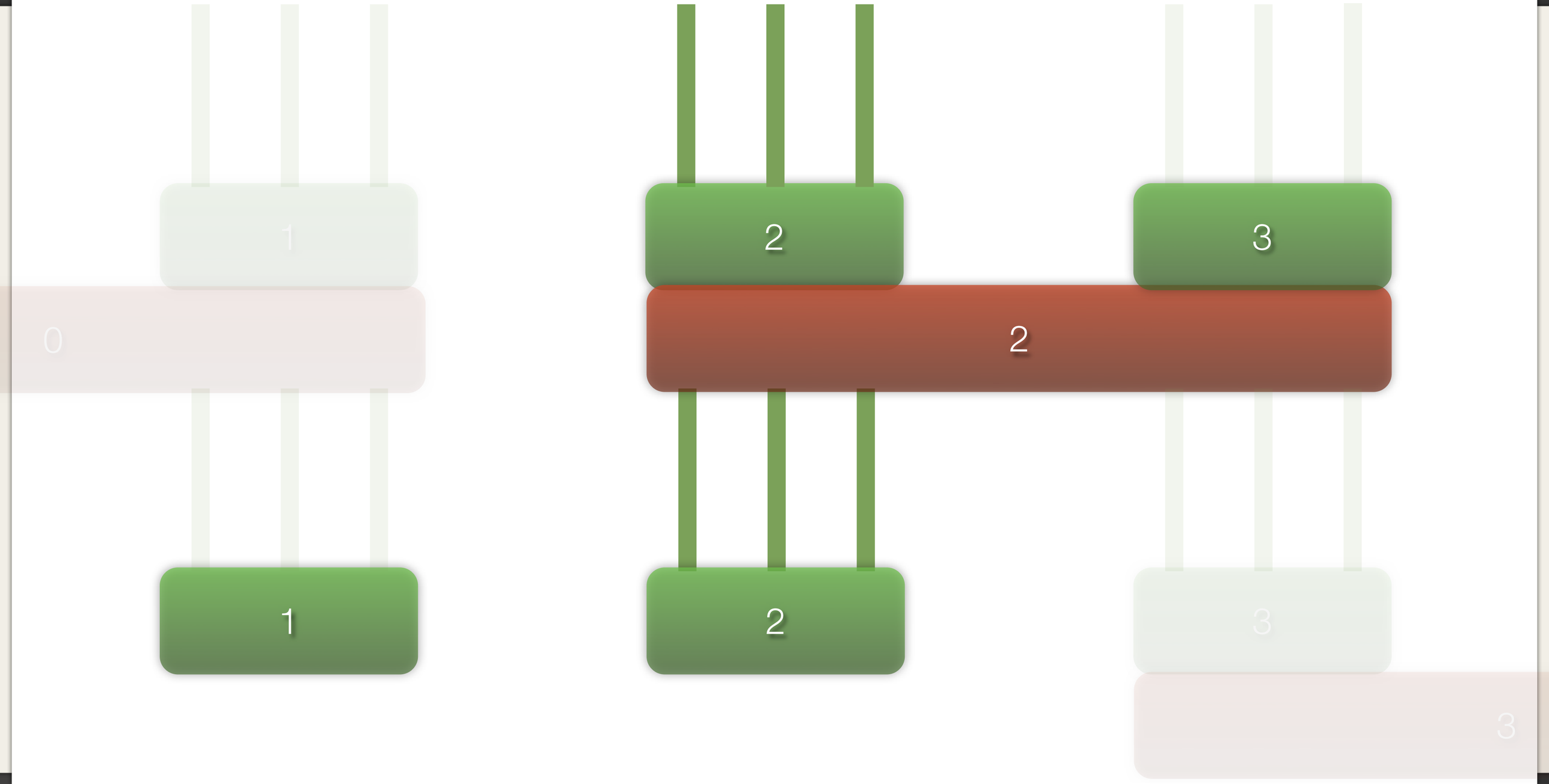

MPI_Comm_revoke
MPI_Comm_shrink
MPI_Comm_spawn
MPI_Intercomm_merge
MPI_Comm_split


1
2
3
0
2
1
2
3
1
3


1
2
3
0
1
2
3
1
3


1
2
3
0
2
1
2
3
1
3


1
2
3
0
2
1
2
3
1
3


1
2
3
0
2
1
2
3
3

# MPI Forum Proposal

- Proposed new chapter for the MPI standard
  - Perceived as incomplete due to short timeframe of submission
  - Proposal will continue in MPI-<next>
- Research is ongoing in multiple implementations


ICL UT
INNOVATIVE
COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE