CW5 - the end is nigh

- Julia floating-point – sorry!
  - I forget that architecture 1 was a long time ago
  - Without the reference it is more difficult to get bit-accurate
    - though some did!

- Try to avoid last-minute performance hacks
  - Unlikely to be that much faster
  - Quite likely to have a failure mode
- Do your push, then pull+build+test in a *different* directory
  - Useful to find uncommitted files
  - Can highlight where stale files
- Decide it is working then... walk away!
  - Performance is engineered, not rushed

Parallel Design Patterns

What are Design Patterns?

- Design patterns are templates for “solving” a problem
  - Solving = {structuring, executing, writing, analysing, ...}
  - Range from application level down to scheduling instructions
  - Emphasis on ***composability*** : able to safely combine patterns
- Structured programming introduced the first design patterns
  - Ordering execution of statements: *sequences, branches, loops*
  - Grouping data into a unit: *structures*
  - Grouping statements into a unit: *procedures and functions*
- Object-oriented programming standardised common patterns
  - `struct X; F(X*,int);  ->  class X{  F(int);  }`
  - Polymorphism (virtual functions) increase composability
  - Don’t need to know what an object is, just what it does

Describing Design Patterns

- Design patterns try to formally capture intuition or experience
  - Reduce the need for “rock-star” programmers: *engineering not art*
  - Increase productivity: *don’t re-invent the wheel*
  - Increase reliability: *record both the patterns **and** when they apply*

- Elements of a design pattern
  - ***Name*** : we need a common name so people can talk about them
  - ***Problem*** : simple description of what problem the pattern solves
  - ***Context*** : where does the problem occur, and any background info
  - ***Forces*** : any intrinsic tradeoffs that are being addressed
  - ***Solution*** : how the pattern is applied
  - ***Examples*** : application of the pattern to a real-world example

Application of Design Patterns

- Concreteness of the design pattern varies hugely

1. Both solution and conditions are described programmatically
   - Can be turned into compiler optimisations
   - Rare and usually local in scope: checking conditions is difficult
   - *Why does this course even exist – can't the compiler do it?*
2. Solution can be captured in code, but not the conditions
   - Compiler/library will apply the pattern for us: `spawn/parallel_for`
   - We need to check whether the pattern can and should be applied
3. Both solution and conditions cannot be formally captured
   - Programmer has to interpret the conditions and the solution
   - Requires human skill and (maybe) some thought

Umm...

- This all sounds great in theory, but the practise is less good
  - People have different names for the same thing
  - People have the same name for different things
  - Currently a lack of standardisation in parallel pattern names
  - GPUs etc. are muddying the waters further
- I will broadly use the terms from these two sources
  - TBB patterns (part of TBB documentation): http://software.intel.com/sites/products/documentation/doclib/tbb_sa/help/tbb_userguide/Design_Patterns/Design_Patterns.htm
  - Berkely parlab: http://parlab.eecs.berkeley.edu/wiki/patterns/patterns

Map-Reduce

- ***Problem*** : a function must be applied to many pieces of data, followed by an associative reduction

- ***Context*** : some container contains $x_1..x_n$, and we wish to calculate $a(f(x_1), a(f(x_2), ... f(x_n)))$. Both *f*(.) and *a*(.) must be side-effect free, and *a*(.) must be associative

- ***Solution*** : apply f(.) to all data-items as independent parallel tasks (*map*), then use a recursive tree of parallel tasks to collect the results (*reduce*)

Example : largest magnitude complex number

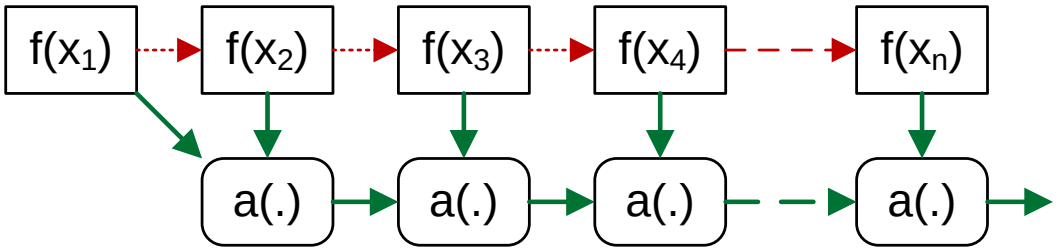
```
double max_magnitude(int n, const complex_t *data)
{
   double best = abs(data[0]);

   for(int i=1;i<n;i++ ){
       double curr = abs(data[i]);
       if(curr > best)
           best=curr;
   }
   return best;
}
```

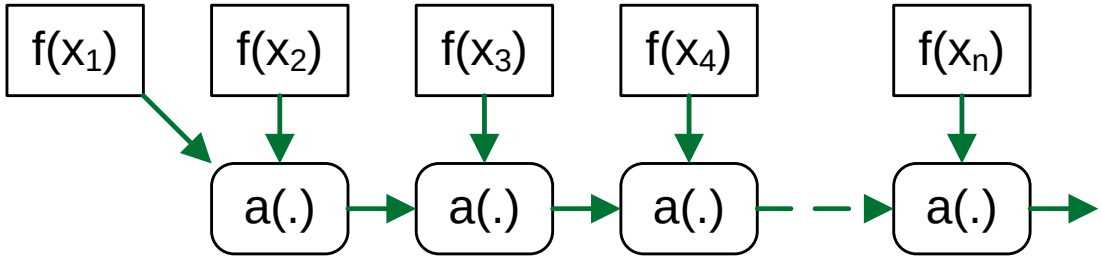
- f(.) = abs(.) - Complex magnitude (modulus)
- a(.) = max(.) – Maximum of two numbers

- Do we meet requirements of pattern?
  - a(.) and f(.) are side-effect free
  - a(.) is associative

```
double max_magnitude(int n, const complex_t *data)
{
    double best = 0.0;
    for(int i=0;i<n;i++ ){
        double curr = abs(data[i]);
        if(curr > best)
            best=curr;
    }
    return best;
}
```

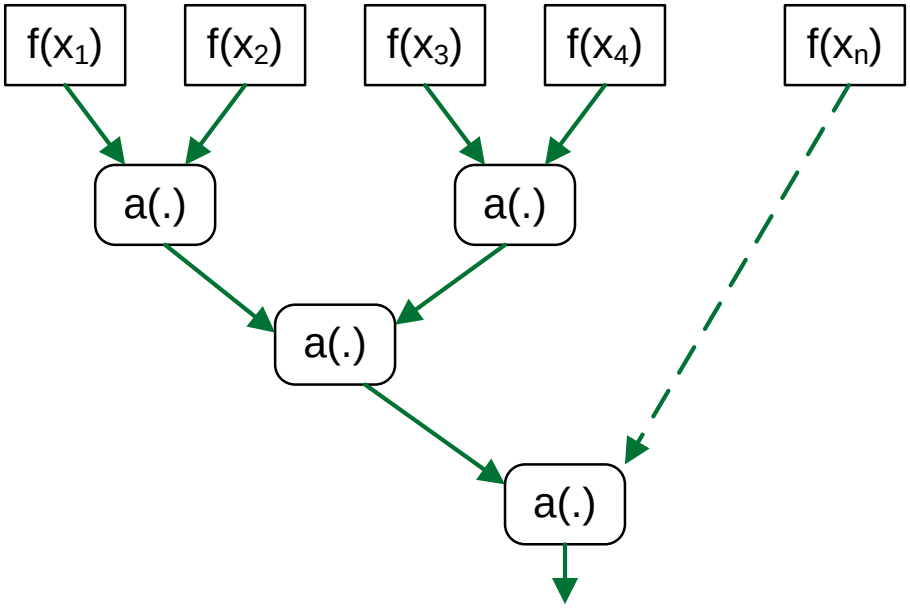


```
double max_magnitude(int n, const complex_t *data)
{
    std::vector<double> temp(n);
    tbb::parallel_for(0, n, [&](int i){
       temp[i] = abs(data[i]);
    });
    for(int i=0;i<n;i++ ){
        if(temp[i] > best)
            best=temp[i];
    }
    return best;
}
```



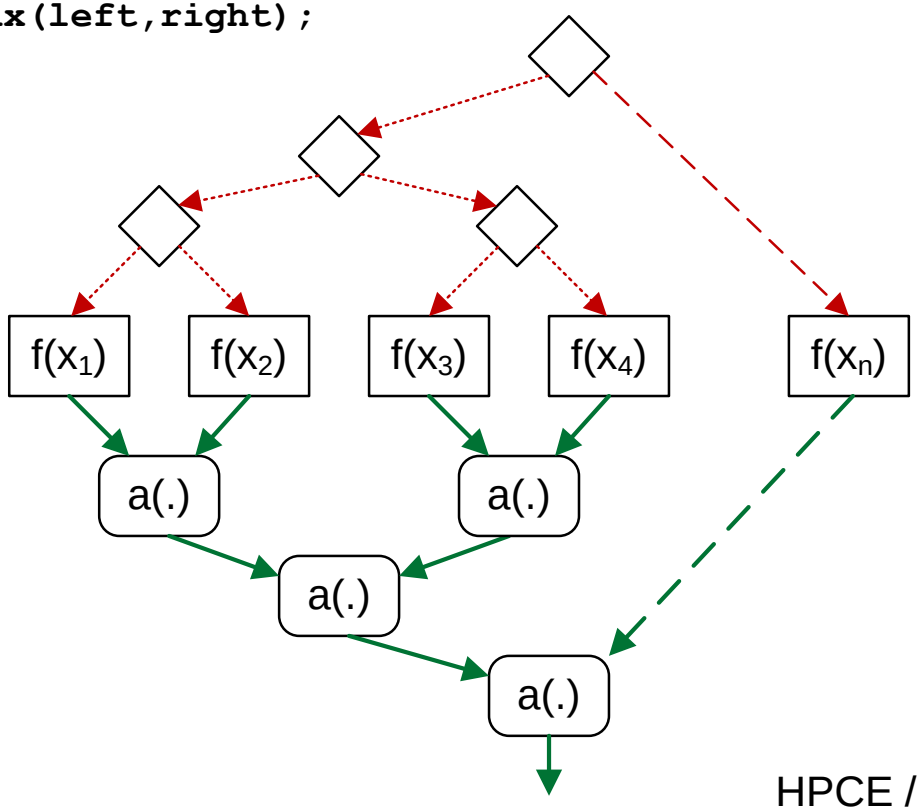
```
double max_magnitude(int n, const complex_t *data)
{
   if(n==1)  return abs(data[0]);

   double left, right;
   tbb::parallel_invoke(
       [&](){ left  = max_magnitude(n/2,   data);      }),
       [&](){ right = max_magnitude(n-n/2, data+n/2); })
   );
   return max(left,right);
}
```



```
double max_magnitude(int n, const complex_t *data)
{
    if(n==1)  return abs(data[0]);

    double left, right;
    tbb::parallel_invoke(
        [&](){ left  = max_magnitude(n/2,   data);      }),
        [&](){ right = max_magnitude(n-n/2, data+n/2); })
    );
    return max(left,right);
}
```



Map-Reduce : Forces

- Balancing parallelism vs overhead
  - We want to maximise available tasks to increase parallelism
  - But the reduction graph contains lots of synchronisation

- Determinism vs speed
  - Want to have as much scheduling flexibility as possible
  - Prefer deterministic results with pseudo-associative operations
    - e.g. floating-point addition

(*I find the "Forces" section often ends up a bit vague*)

Implementation of Map-Reduce

- Map-Reduce occurs everywhere
  - Sequential version is available in many languages and libraries
  - Often the map and reduce functions must be combined
    - mr(x,y) = a(x, m(y))
  - C++ : std::accumulate; Python: reduce; Haskell: foldl/foldr
- Parallel versions of Map-Reduce are very common
  - Very easy to understand for users
  - Applicable in large numbers of real-world scenaries
- Google use a tool called MapReduce with thousands of CPUs
  - Excellent paper on real-world scaling in distributed systems:
    - http://research.google.com/archive/mapreduce.html
    - *But they didn't invent the idea, been around for decades*
  - Many similar approaches: Hadoop etc.

tbb::parallel_reduce

```
class Func
{
   Func(Func &src, tbb::split);

   template<class T>
   void operator()( const T &r );

   void join(Func &rhs);
};


template<class TR,class TF>
void tbb::parallel_reduce(
   const TR &range,
   TF &body
);
```

- Must pass object with specific members
- Splitting constructor
  - Used by TBB when the range is split
- `operator()(r)`
  - Similar to `tbb::parallel_for`
  - Except – ***non-const***!
  - Object must change during reduction
- `join` function
  - Merge results from two ranges
  - Accumulator of rhs is merged into lhs
- `parallel_reduce` returns void
  - Results have to come out of body

```
struct Worker
{
 double m_max;
 const complex_t *m_data;

 Worker(const complex_t *data)
 {   m_data=data;    m_max=0.0;  }

 Worker(Worker &src, tbb::split)
 {   m_data=src.m_data;  m_max=0.0;  }

 template<class T>
 void operator()( const T &r ) {
   double acc=0.0;
   for(int i=r.begin();i<r.end();i++)
     acc=std::max(acc,abs(m_data[i]));
   m_max = std::max(m_max, acc);
 }

 void join(Worker &rhs)
 { m_max = std::max(m_max,rhs.m_max); }
};
```

- Two member variables
  - `m_data` : What we are working on
  - `m_max` : The maximum so far
- Standard constructor
  - Creates the “root” object
- Splitting constructor
  - Called when TBB splits the range
  - Need to store where the data is
- Function operator
  - Do local maximum over range
  - Add to maximum seen so far
- join: merge two sub-ranges
  - rhs will then disappear

```
double max_magnitude(
    int n,
    const complex_t *data
){
    return tbb::parallel_reduce(
        // range
        tbb::blocked_range<int>(0,n),
        // identity
        DBL_MAX,
        // reduction over a range
        [](tbb::blocked_range<int> &r) -> double
        {
          double acc=0.0;
          for(int i=r.begin(); i<r.end(); i++)
            acc=std::max(acc,abs(data[i]));
          return acc;
        },
        // reduction of two values
        [](double left, double right) -> double
        {
          return std::max(left, right);
        }
    );
}
```

- There is also a lambda form
- Needs an identity element
  - f(x,identity) = x

Element-wise / Data-Parallel

- ***Problem*** : apply a function to all items in a container
- ***Context*** : an identical function is to be applied to all items in a container, either transforming the items themselves, or performing some other side-effect
- ***Forces*** : standard (no. of tasks vs task overhead)
- ***Solution*** : create work by sub-dividing the container
- ***Examples*** : `tbb::parallel_for`, CUDA, OpenCL

Agglomeration

- ***Problem*** : an application has too much fine-grain parallelism
- ***Context*** : many algorithms can be easily decomposed to the level where each task executes a single instruction, but this results in an extremely high cost of work
- ***Forces*** : we need to balance the need for average parallelism versus the cost of work
- ***Solution*** : halt production of new tasks when the remaining work drops below a certain level, and switch to sequential execution

Agglomeration Examples

- Recursive FFT
  - Switching to sequential when the matrix gets too small
  - Potential problem: a fixed threshold may give sub-optimal results
    - What is optimal on one platform may not be on another
    - Compiler optimisations may increase the agglomeration required
- `tbb::parallel_for` and `tbb::parallel_reduce`
  - Function object is given a ***range*** of indices, not a single index
  - Size of range is optimised to try to balance execution time
  - Potential problem: auto-optimisation may guess wrong
    - Splits a large but extremely fast range into individual tasks
    - Doesn't split a small loop where each iteration is very slow
- Usually auto-tuning is better, unless you perform experiments
  - Humans are pretty bad at guessing how fast things run
  - 8-CPUs used to be rare, now 32-CPUs is fairly common

Divide and Conquer / Recursive

- ***Problem*** : a sequential program with recursive functions or data-structures must be parallelised

- ***Context*** : the existing program repeatedly splits the data into independent sub-sections, which can then be further decomposed. The splitting may be over a 1D or 2D range, or follow some graph-like data-structure

- ***Forces***: standard (creating parallelism vs task size)

- ***Solution*** : when the program splits the problem, spawn new tasks for each sub-problem, then sync for all child tasks

```
// Split the range in two, returning
// the pivot element. Takes
// O(end-begin) steps
double *partition(
    double *begin,
    double *end
);

void QSort(
    double *begin,
    double *end
){
 if(end-begin<1000){
    // Agglomeration - drop to serial
    std::sort(begin, end);
 }else{
    double *mid=partition(begin, end);
    tbb::parallel_invoke(
      [&](){ QSort(begin, mid-1); }),
      [&](){ QSort(mid, end);     })
    );
  }
}
```

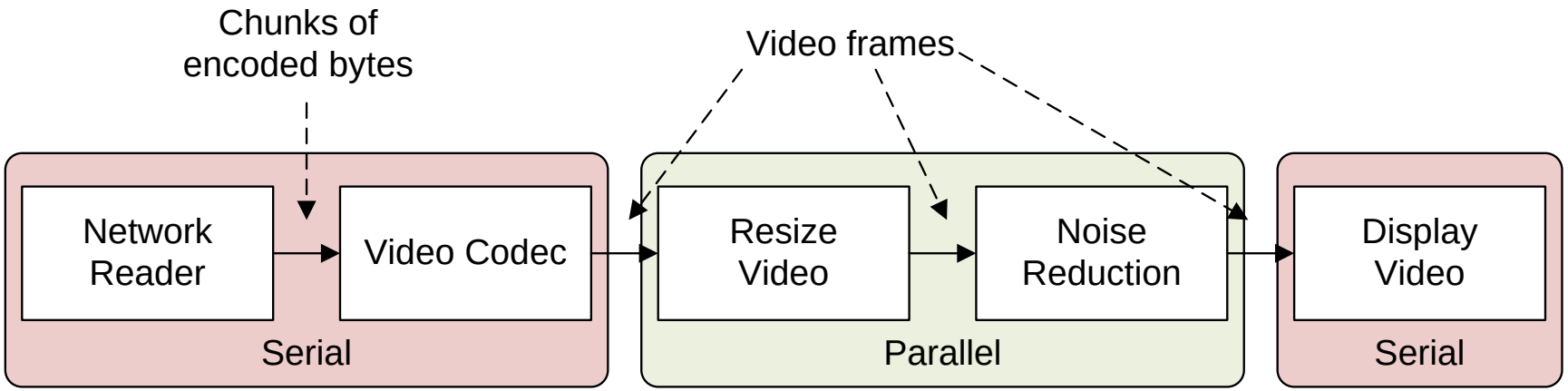
- Sorting: fundamental operation
  - Quick-Sort repeatedly splits problem into smaller problems
  - What is the big-O complexity?
- Sorting is so common it is supported directly in TBB
  - `tbb::parallel_sort`
  - Same interface as std::sort
  - Guaranteed ***deterministic***
    - Always gives the same result
  - Not guaranteed ***stable***
    - Different objects with the same key may be re-ordered

Pipeline

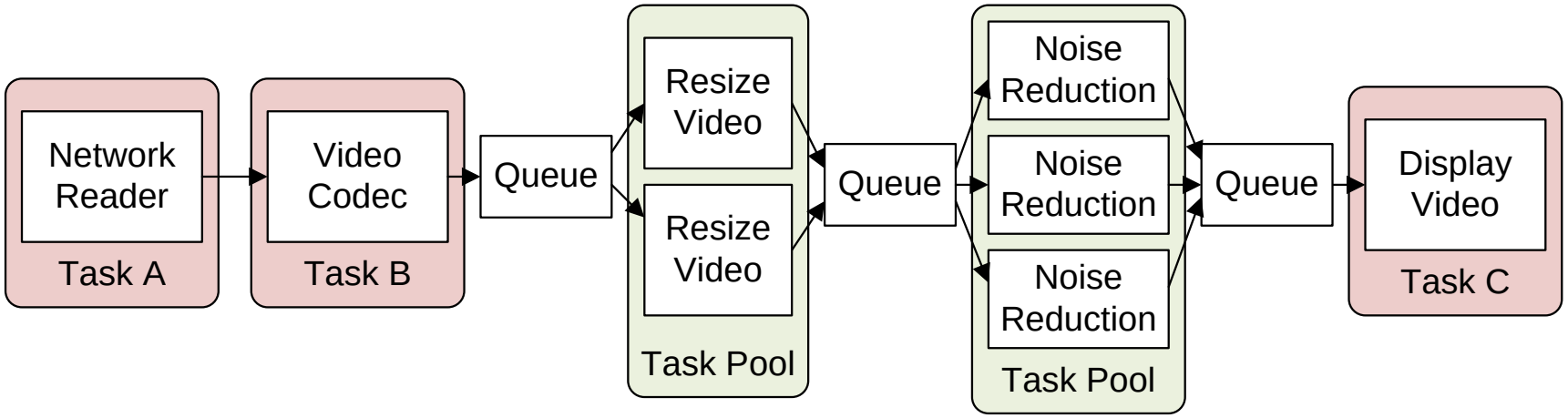
- ***Problem*** : a stream of data needs to be processed in stages
- ***Context*** : some source function produces a stream of individual chunks of data, which must then be transformed using a set of operations, then sent to some sink function
- ***Forces*** : allowing many parallel data items increases available parallelism, but can greatly increase memory requirements
- ***Solution*** : create a pipeline of filter objects which can be applied as parallel tasks, then manage lifecycle of a limited number of tokens through the pipeline

Example : Video Processing

- A pre-recorded video stream is being decoded and displayed
  - Fetch and decode: get encoded bytes and turn into frames
  - Resize and de-noise: transform individual frames before display
  - Display : present the frames to the user
- Consider the metrics : optimise throughput, not execution time
- Video decoding and display have an intrinsic order
- Individual frames can be transformed in parallel in any order



- Use queues as a buffer between different tasks
- Ordered/serial processes are scheduled as a single task
  - Can only be executed on one CPU at a time
- Unordered/parallel processes can expand to multiple tasks
  - If there is lots of work in the queue create new task for each item
  - Task set expands to soak up CPUs
- Problem: process creates data faster than consuming process
  - Solution: limited size queues. e.g. only 16 frames in flight at once
- Implemented in TBB as `tbb::pipeline`



Compare and Swap Loop

- ***Problem*** : a read-write variable is shared between tasks
- ***Context*** : multiple threads are communicating via some shared variable, such as the best or worst solution seen so far
- ***Forces*** : the update needs to be thread-safe, but also needs to be fast (so task dependencies cannot be used)
- ***Solution*** : use an atomic variable combined with a compare and swap loop
- ***Examples*** : see earlier lecture

Further design patterns

- There are many more parallel design patterns
  - Some are very specialised for particular applications
  - Some seem extremely obvious
  - People who are into design patterns go a bit over the top
- Best design patterns are those that can be turned into a library
  - Require programmer's analysis to decide whether it applies
  - Provide a way to quickly apply solution ***if*** it is appropriate
- TBB has a number of design patterns as algorithms
  - Section 4 of ref. manual : `tbb::parallel_do`,`tbb::parallel_scan`
- Further useful patterns in the TBB Design Patterns Guide
  - *Fenced Data Transfer, Local Serializer*, etc.

CW6

“Real” problems

- You’re at a research-intensive university
  - Theory is that good researchers make good teachers...
- We’re supposed to combine teaching and research
- Giving you problems I already solved is not real
- So....

POETS

Andrew Brown (Southampton)
Simon Moore (Cambridge)
Andrey Mokhov (Newcastle)
David Thomas (Imperial)

POETS consortium

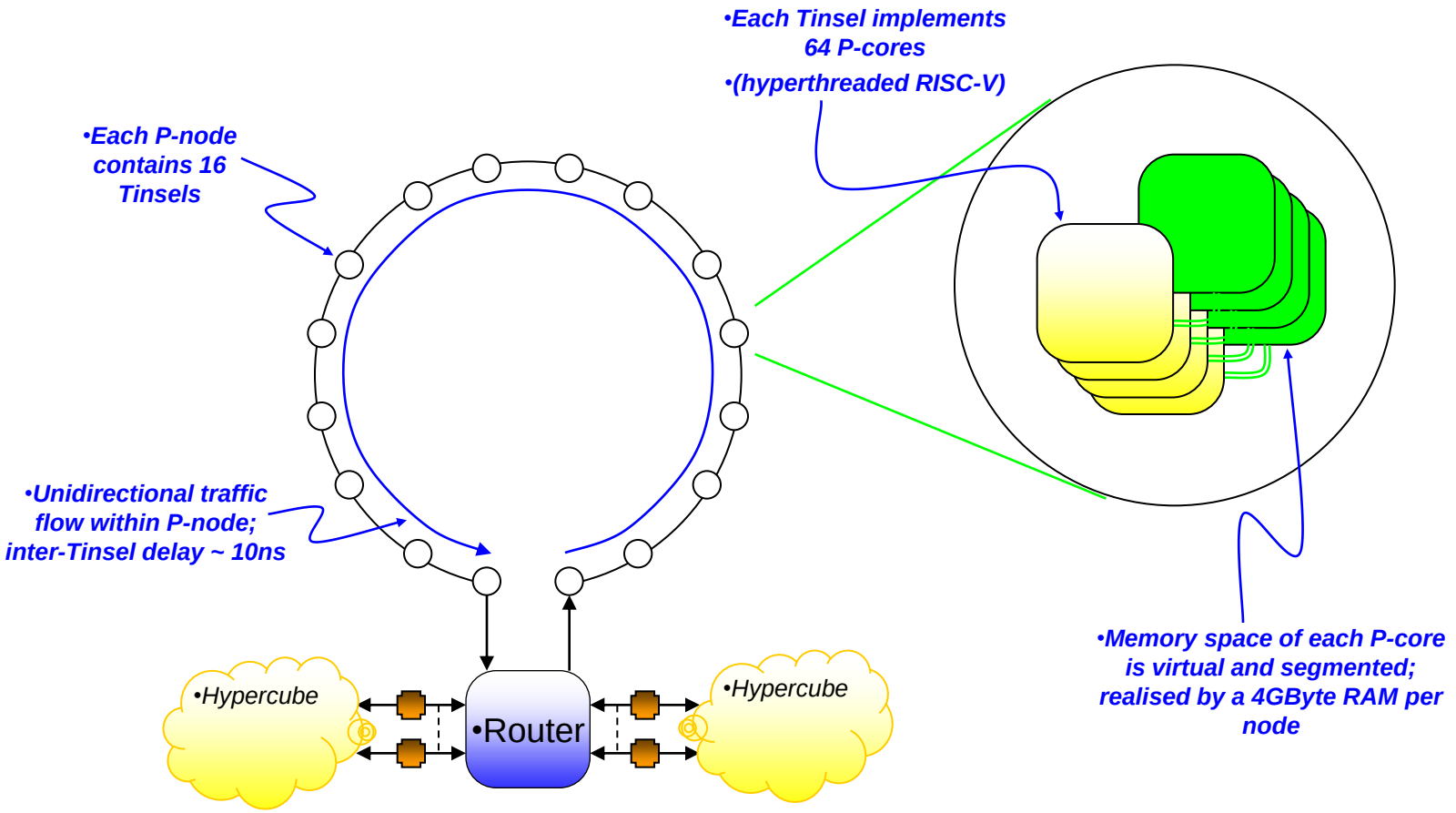
POETS : Partially Ordered Event Triggered Systems

- A key barrier to high performance is synchronisation
  - Core-to-core : cache coherency
  - Workitem-to-workitem : OpenCL barrier
  - Thread-to-thread : mutexes
  - CPU to GPU : OpenCL events
  - Machine-to-Machine: MPI and networking

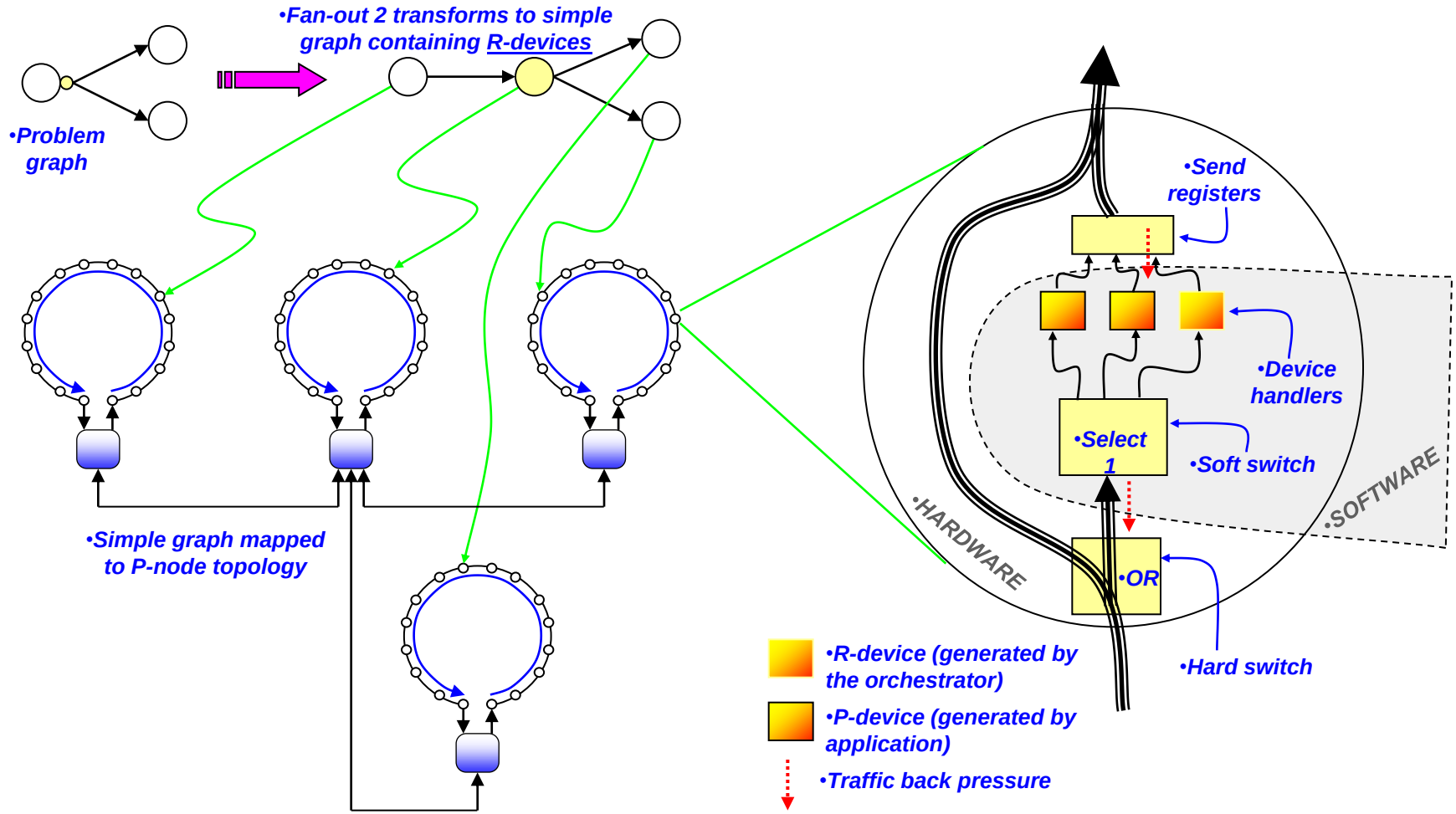
- What if we could avoid synchronisation?

Imagine... : 10,000 cores shouting at each other in parallel

The P-node

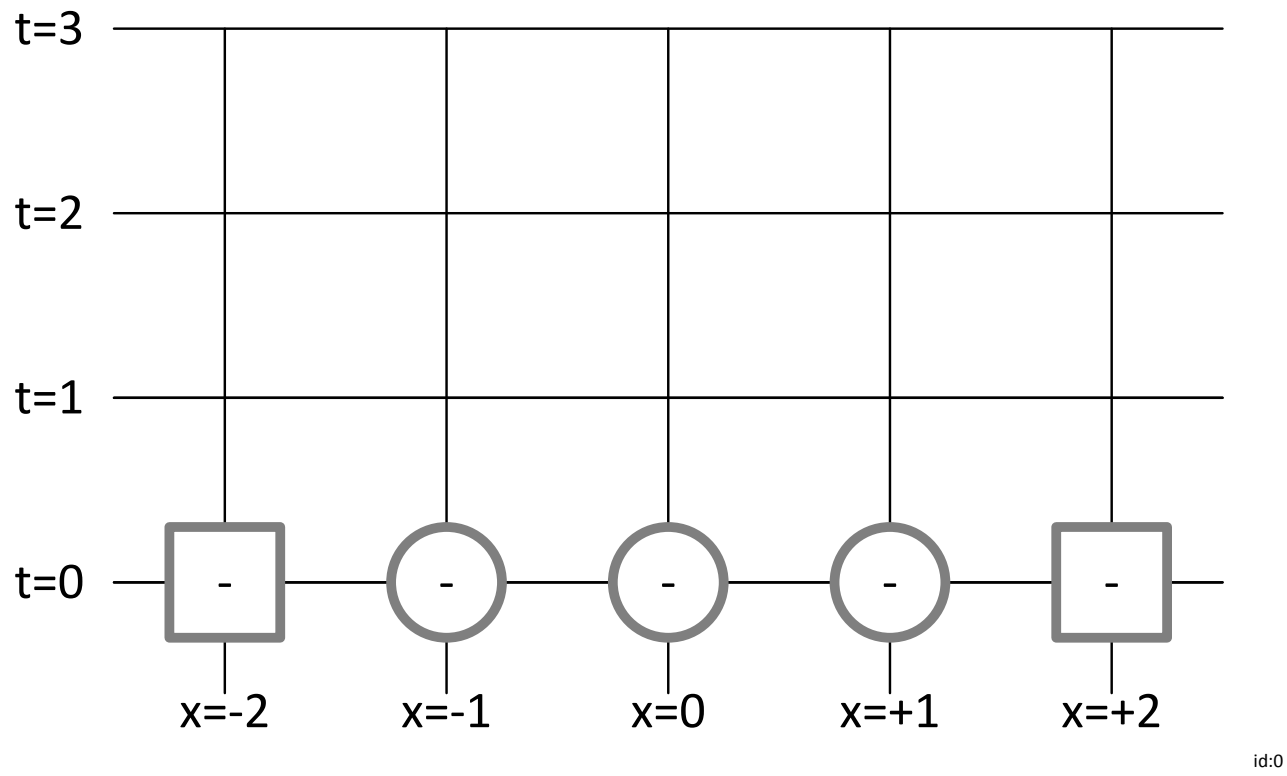

•Each Tinsel implements 64 P-cores
•(hyperthreaded RISC-V)
•Each P-node contains 16 Tinsels
•Unidirectional traffic flow within P-node; inter-Tinsel delay ~ 10ns
•Memory space of each P-core is virtual and segmented; realised by a 4GByte RAM per node
•Hypercube
•Router
•Hypercube

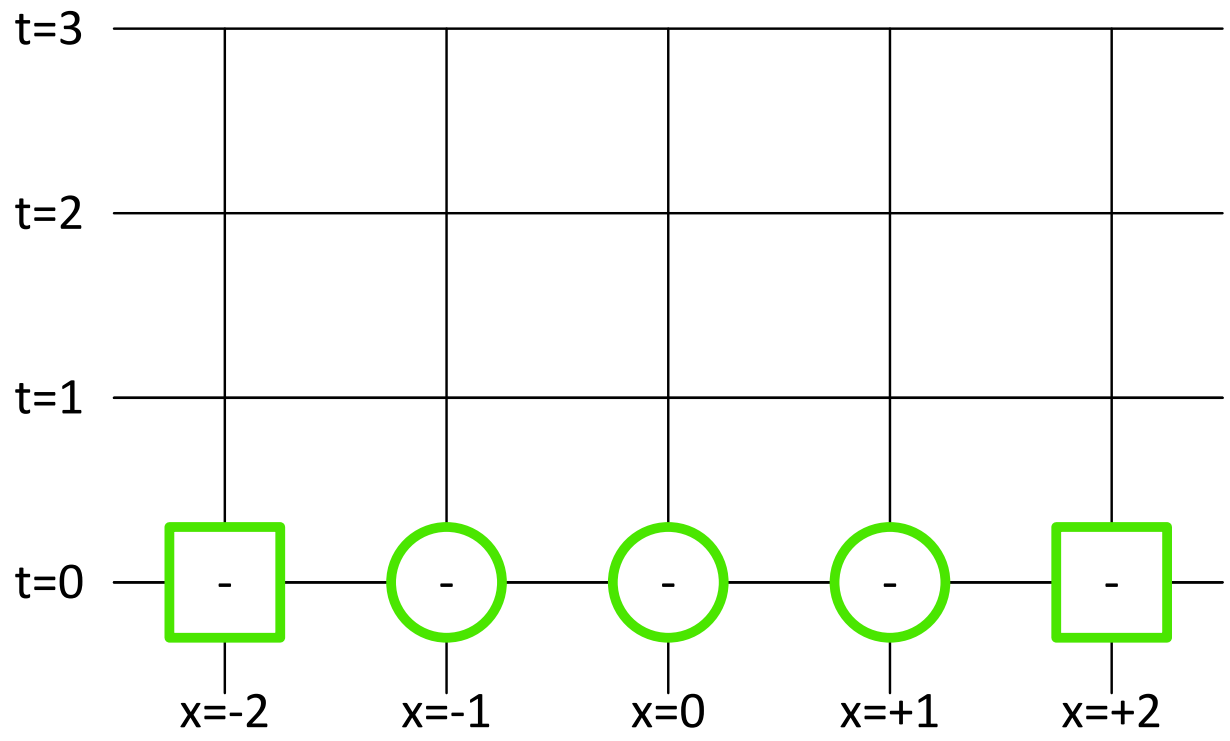
The P-core


•Fan-out 2 transforms to simple graph containing R-devices
•Problem graph
•Simple graph mapped to P-node topology
•Send registers
•Device handlers
•Select 1
•Soft switch
•OR
•Hard switch
•HARDWARE
•SOFTWARE
•R-device (generated by the orchestrator)
•P-device (generated by application)
•Traffic back pressure

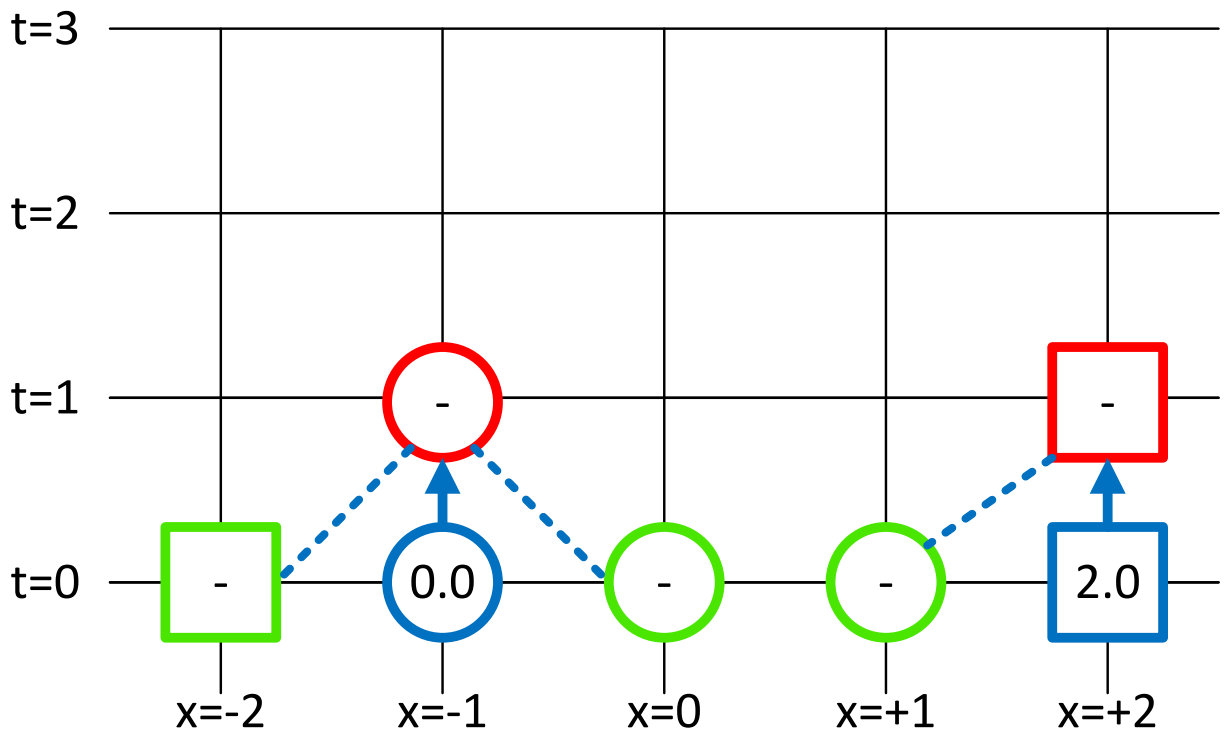
POETS applications are asynchronous

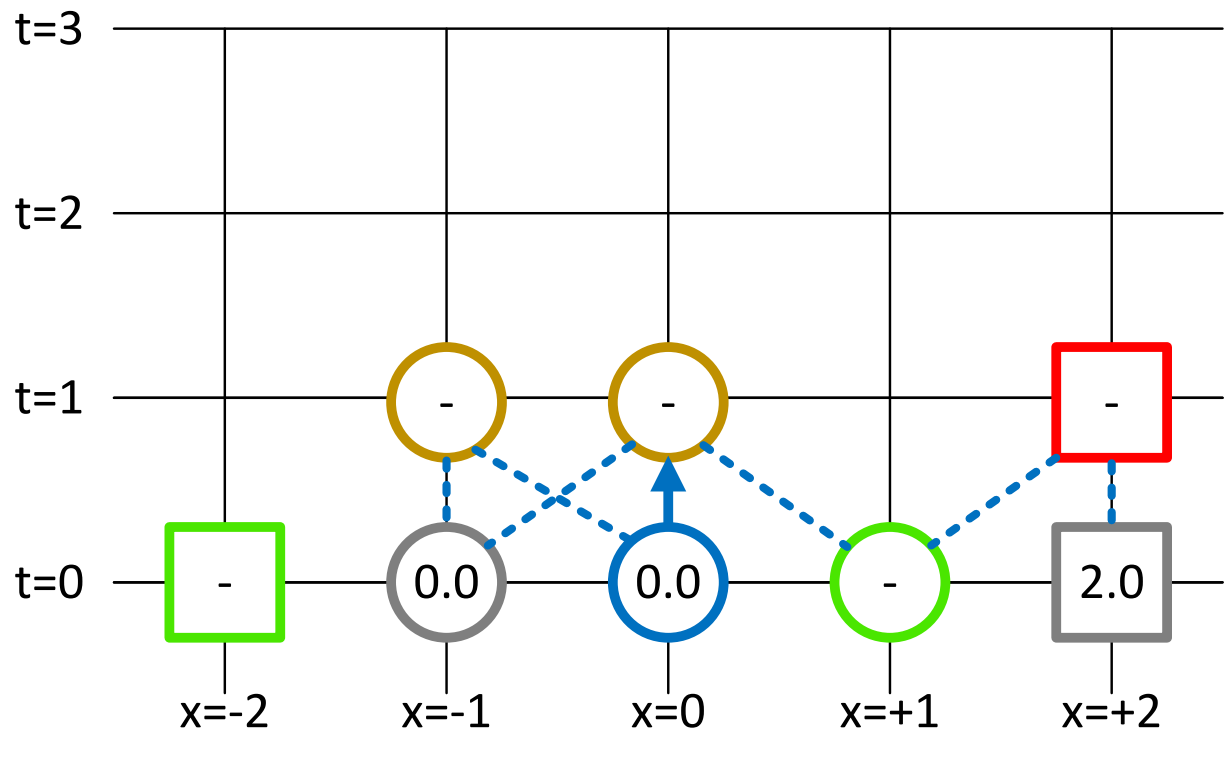
- Applications are split into thousands of “devices”
  - A device represents a point in space
  - Recall coursework 3 : world discretised into heat-cells
- Devices send messages to each other
  - For example in a heat equation:
    - Device A -> Device B : “My temperature is 0.6”
    - Device B : Ok, I’ll update to temperature 0.7
    - DeviceB -> DeviceA : “My temperature is now 0.7”
  - Messages are very efficient
    - No synchronisation
- But... messages are sent in any order
  - How do you maintain a notion of time?

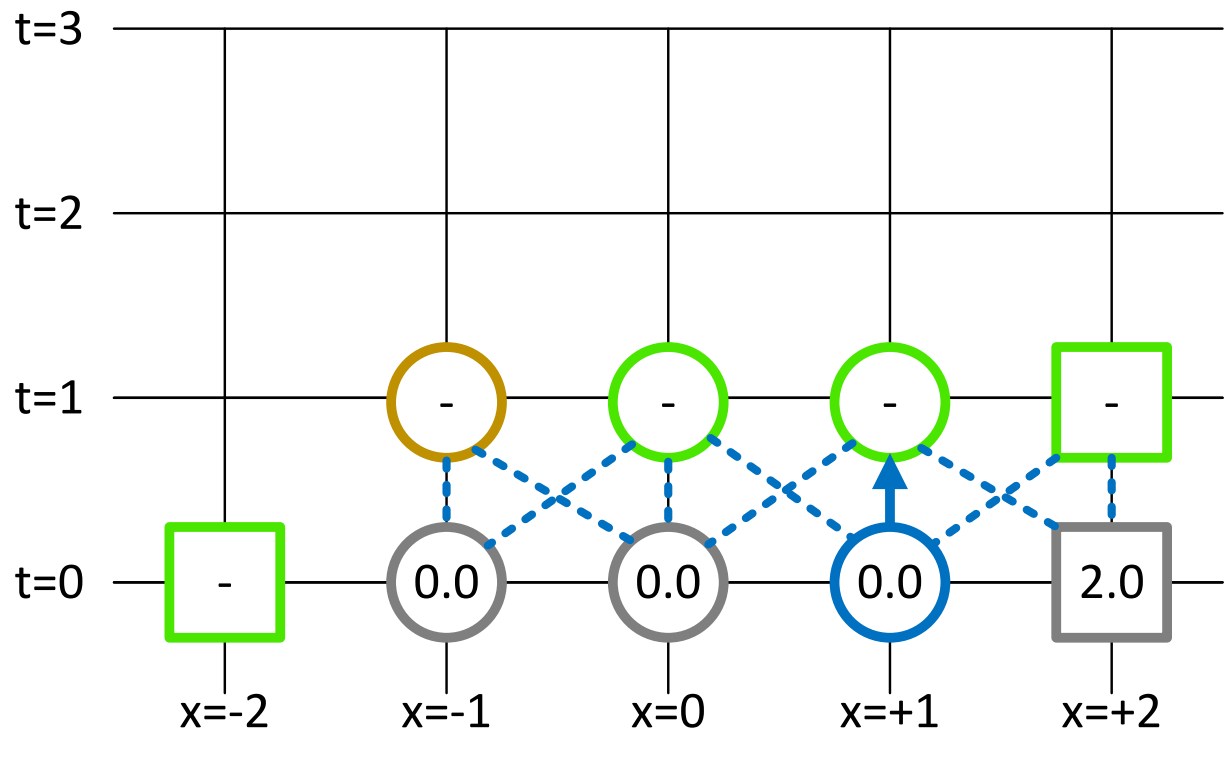

t=3
t=2
t=1
t=0
-
-
-
-
-
x=-2
x=-1
x=0
x=+1
x=+2
id:0

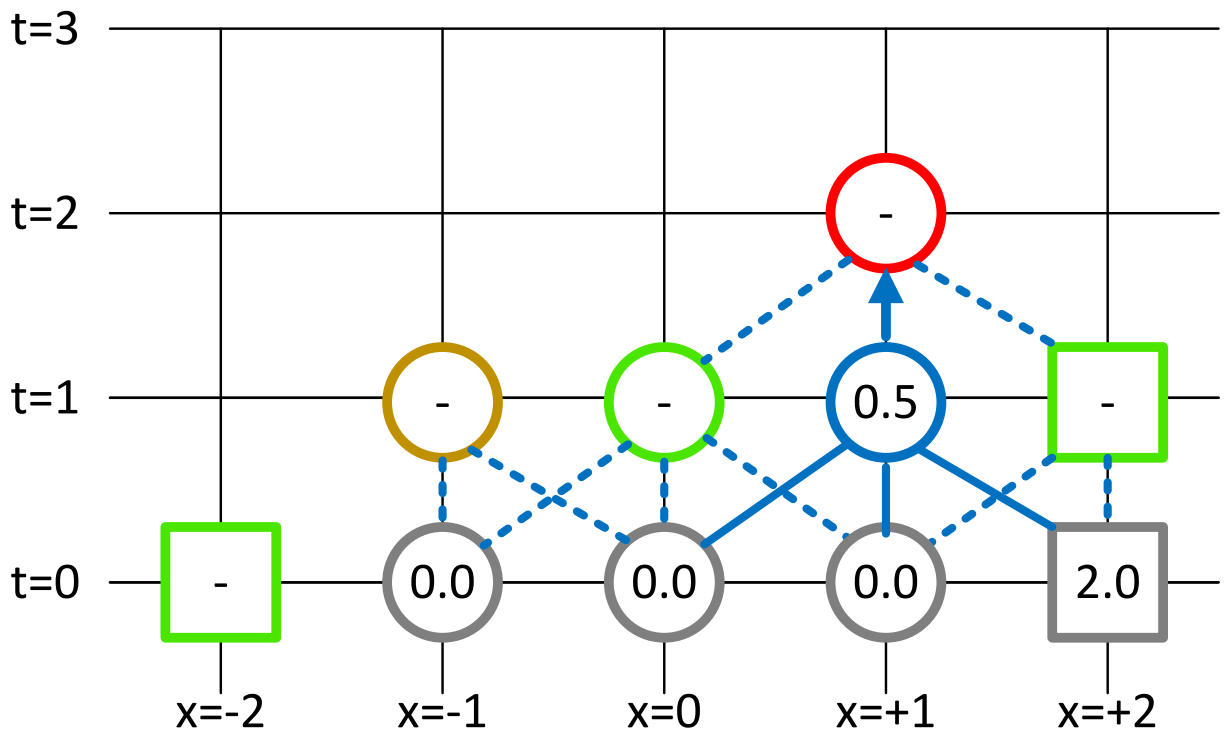


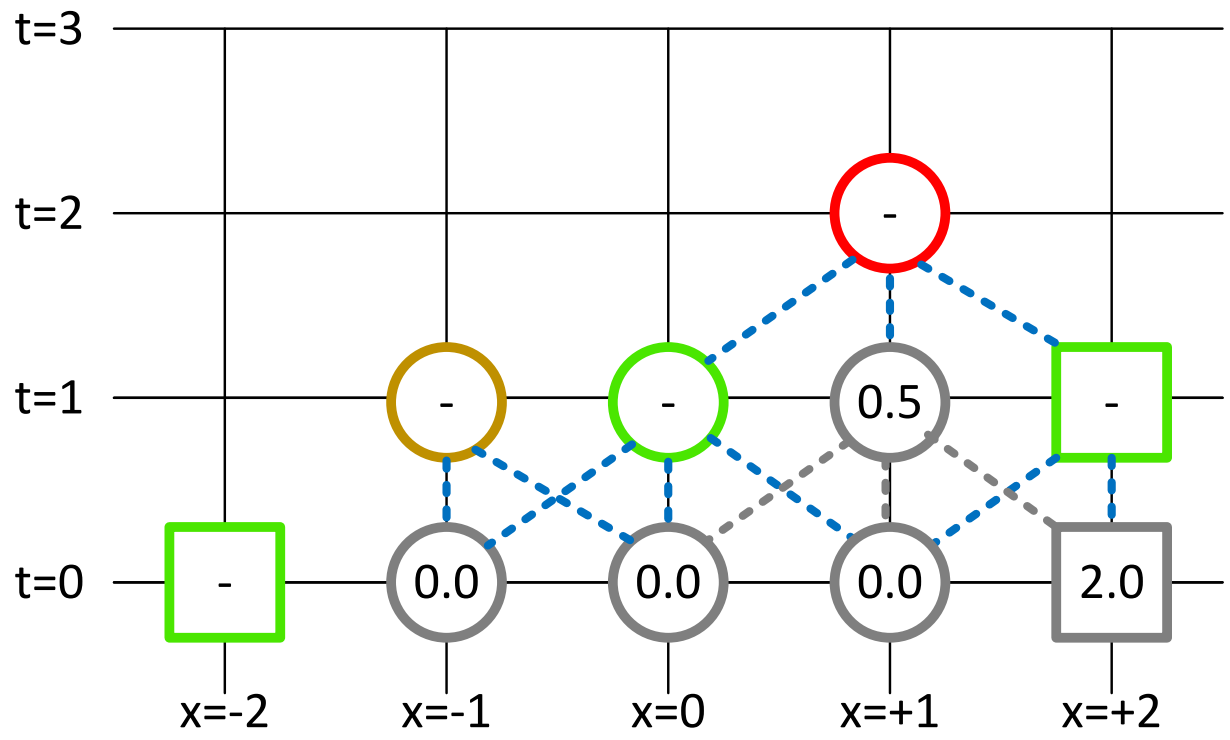
id:1

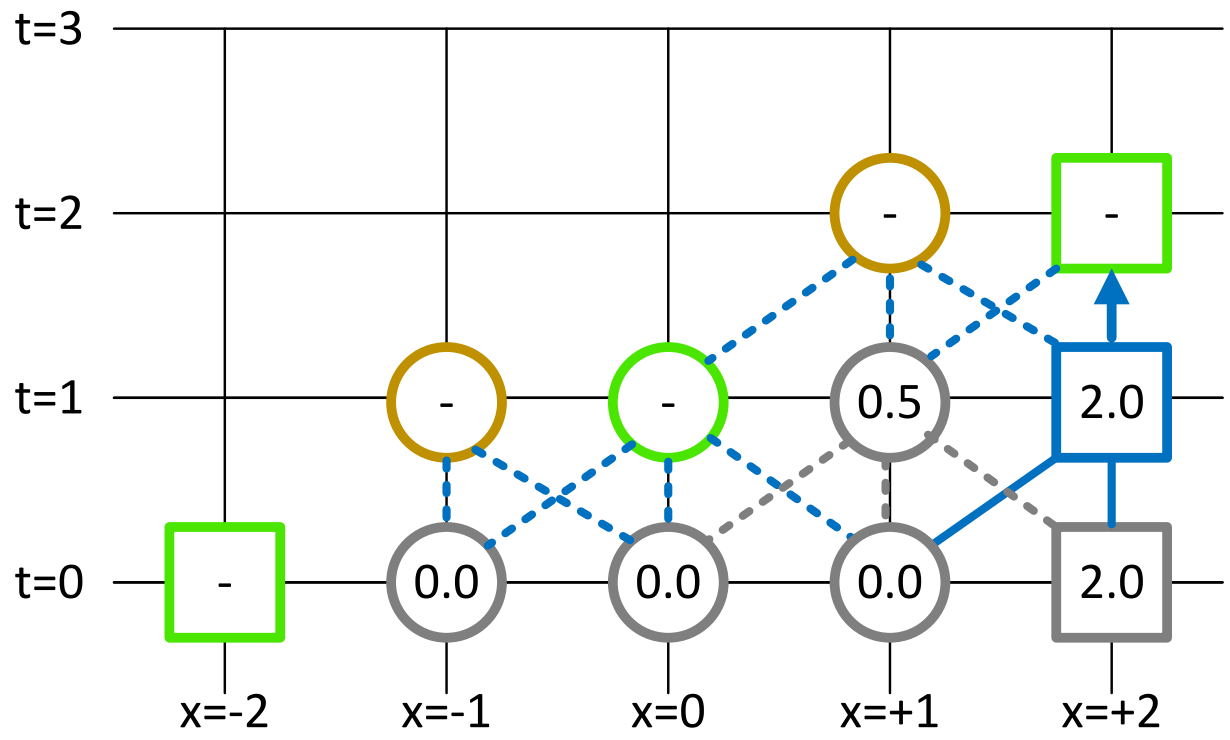

t=3
t=2
t=1
t=0
-
-
-
0.0
-
-
2.0
x=-2
x=-1
x=0
x=+1
x=+2
id:2

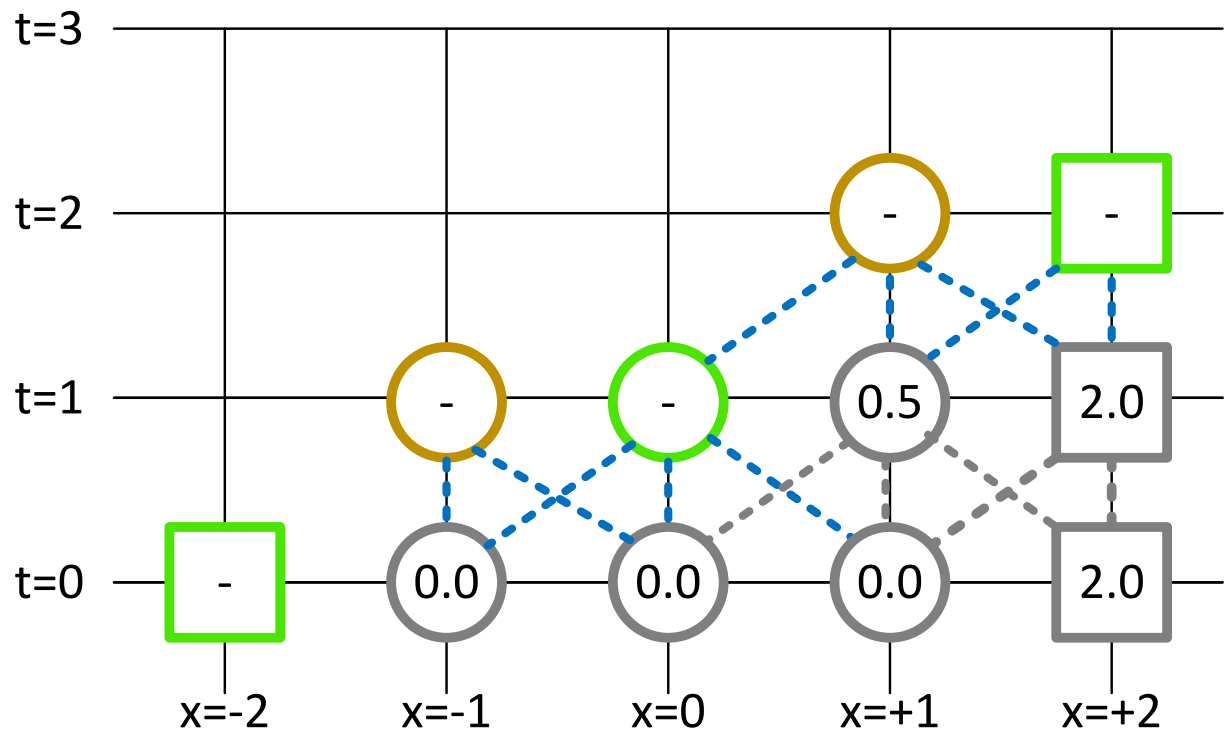

t=3
t=2
t=1
t=0
-
-
-
-
0.0
0.0
-
2.0
x=-2
x=-1
x=0
x=+1
x=+2
id:3

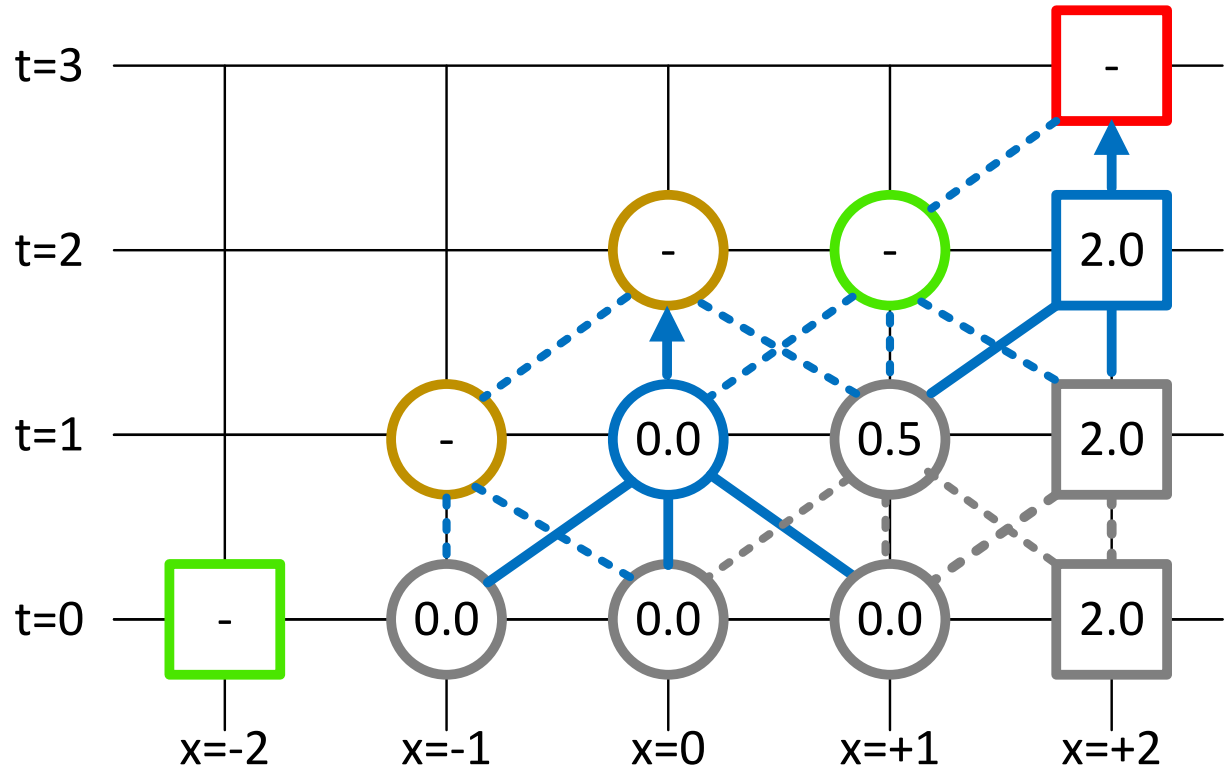

t=3
t=2
t=1
t=0
-
-
-
-
-
0.0
0.0
0.0
2.0
x=-2
x=-1
x=0
x=+1
x=+2
id:4

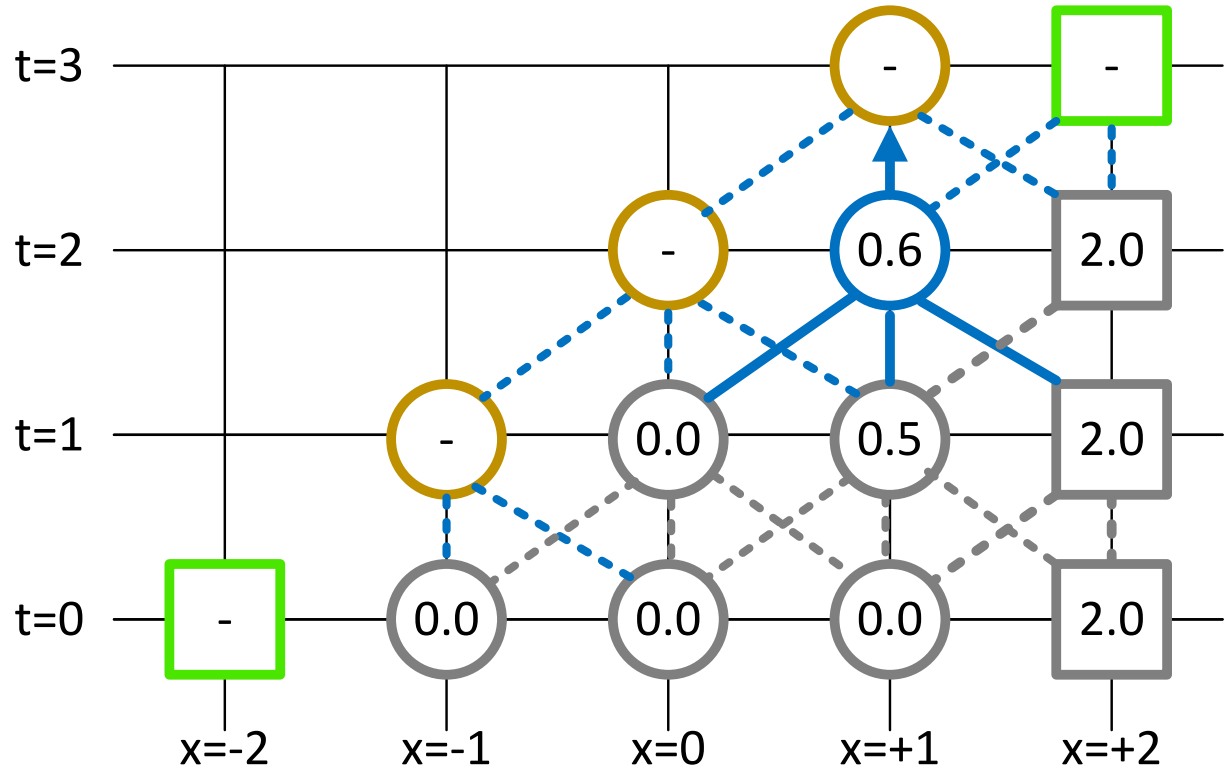

t=3
t=2
t=1
t=0
-
-
-
0.5
-
-
0.0
0.0
0.0
2.0
x=-2
x=-1
x=0
x=+1
x=+2
id:5

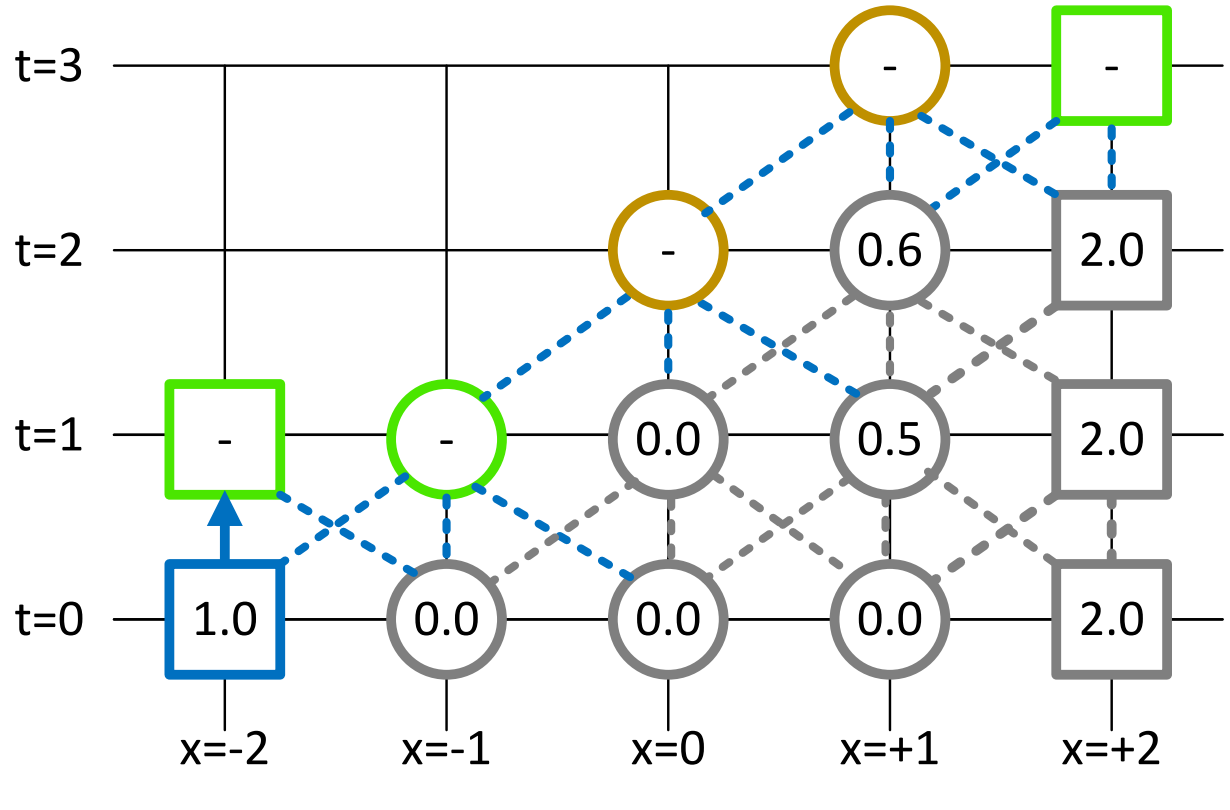

t=3
t=2
t=1
t=0
-
-
-
0.5
-
-
0.0
0.0
0.0
2.0
x=-2
x=-1
x=0
x=+1
x=+2
id:6

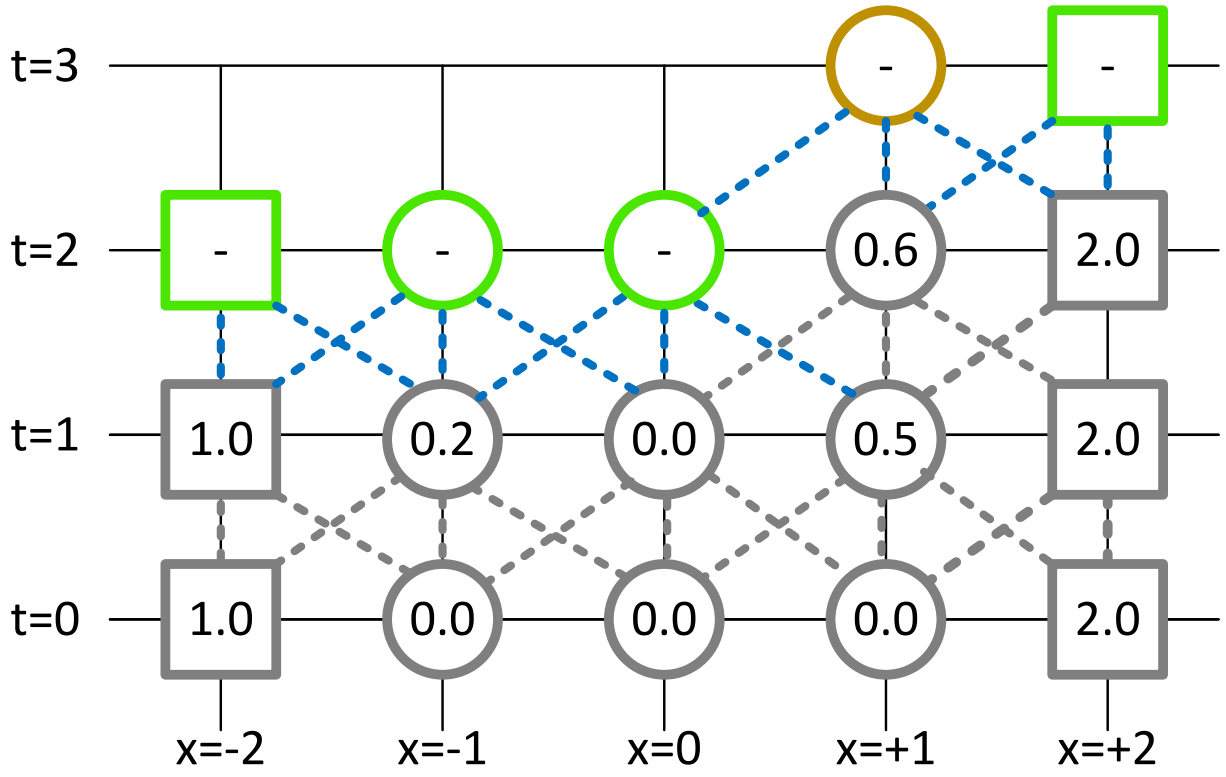

t=3
t=2
t=1
t=0
-
-
-
-
0.5
2.0
-
0.0
0.0
0.0
2.0
x=-2
x=-1
x=0
x=+1
x=+2
id:7

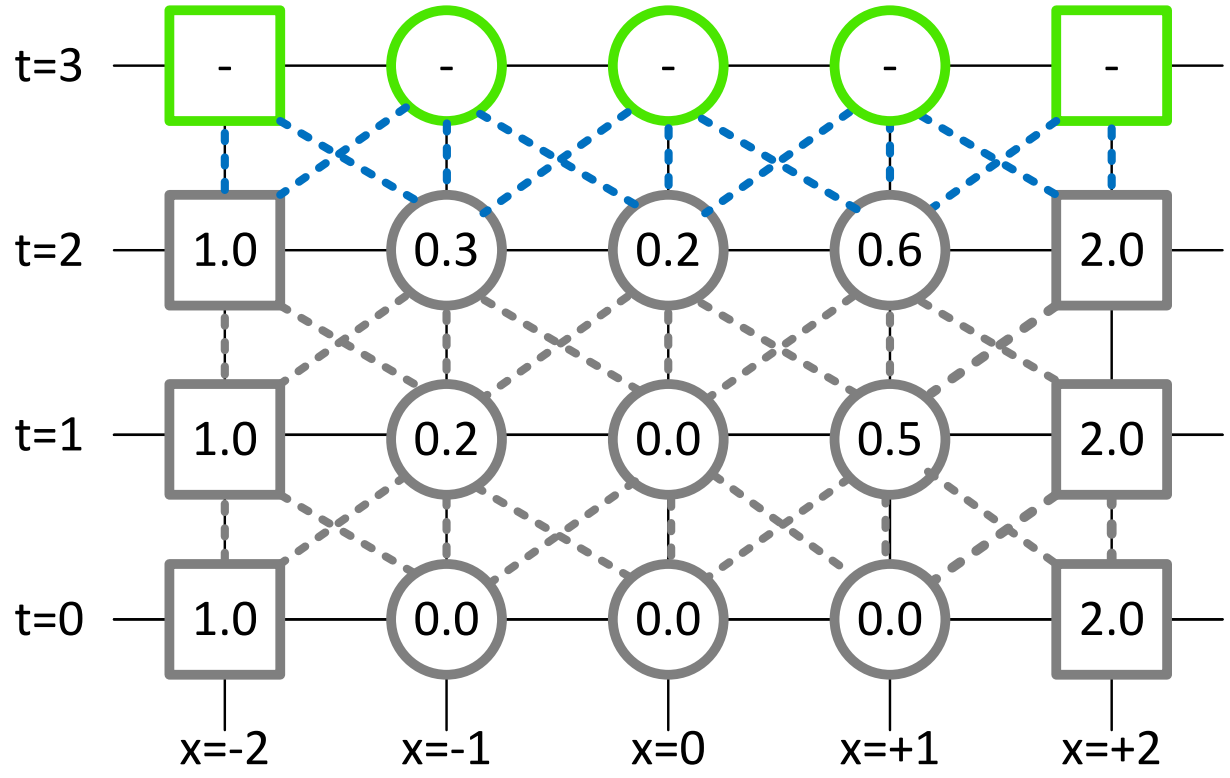

t=3
t=2
t=1
t=0
-
-
-
-
0.5
2.0
-
0.0
0.0
0.0
2.0
x=-2
x=-1
x=0
x=+1
x=+2
id:8


t=3
t=2
t=1
t=0
-
-
-
-
2.0
-
0.0
0.5
2.0
-
0.0
0.0
0.0
2.0
x=-2
x=-1
x=0
x=+1
x=+2
id:9


t=3
t=2
t=1
t=0
-
-
-
-
0.6
2.0
-
0.0
0.5
2.0
-
0.0
0.0
0.0
2.0
x=-2
x=-1
x=0
x=+1
x=+2
id:10


t=3
t=2
t=1
t=0
-
-
-
-
0.6
2.0
-
-
0.0
0.5
2.0
1.0
0.0
0.0
0.0
2.0
x=-2
x=-1
x=0
x=+1
x=+2
id:11

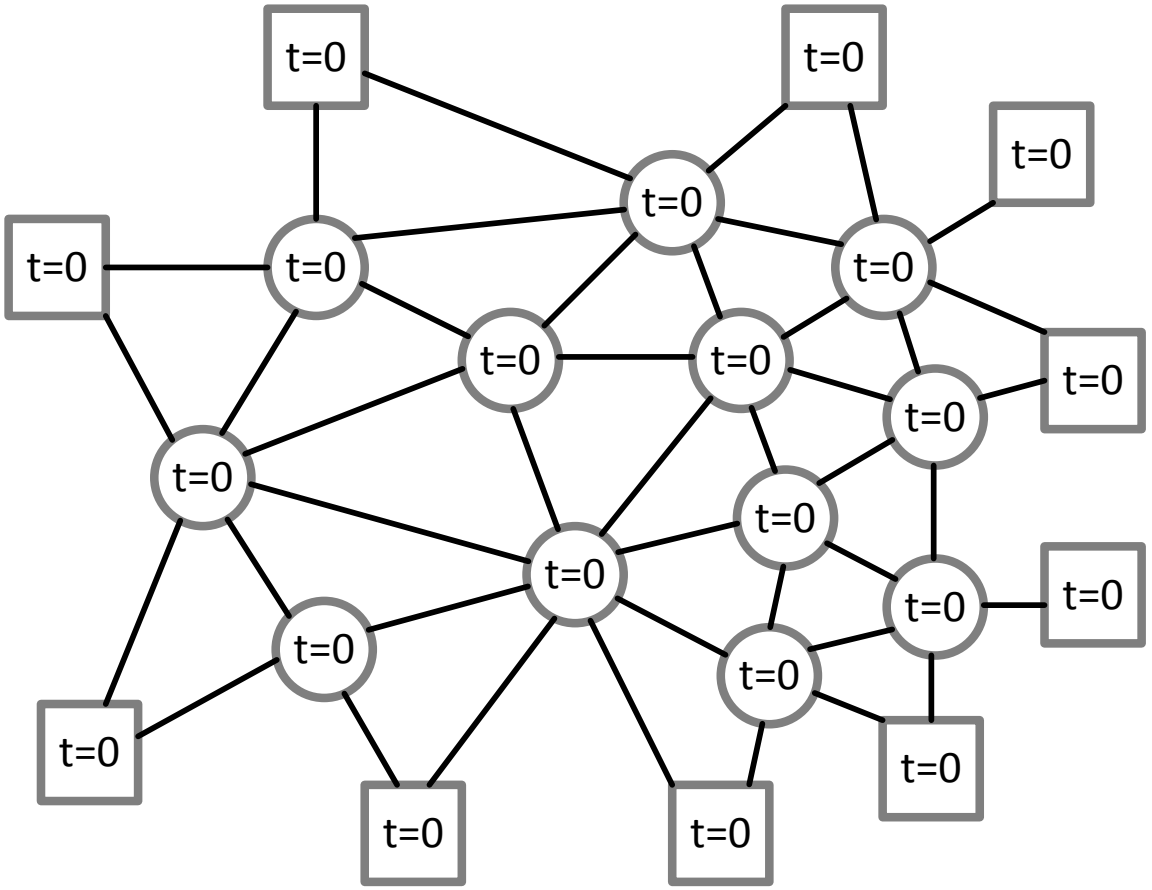

t=3
t=2
t=1
t=0
-
-
-
-
-
0.6
2.0
1.0
0.2
0.0
0.5
2.0
1.0
0.0
0.0
0.0
2.0
x=-2
x=-1
x=0
x=+1
x=+2
id:12

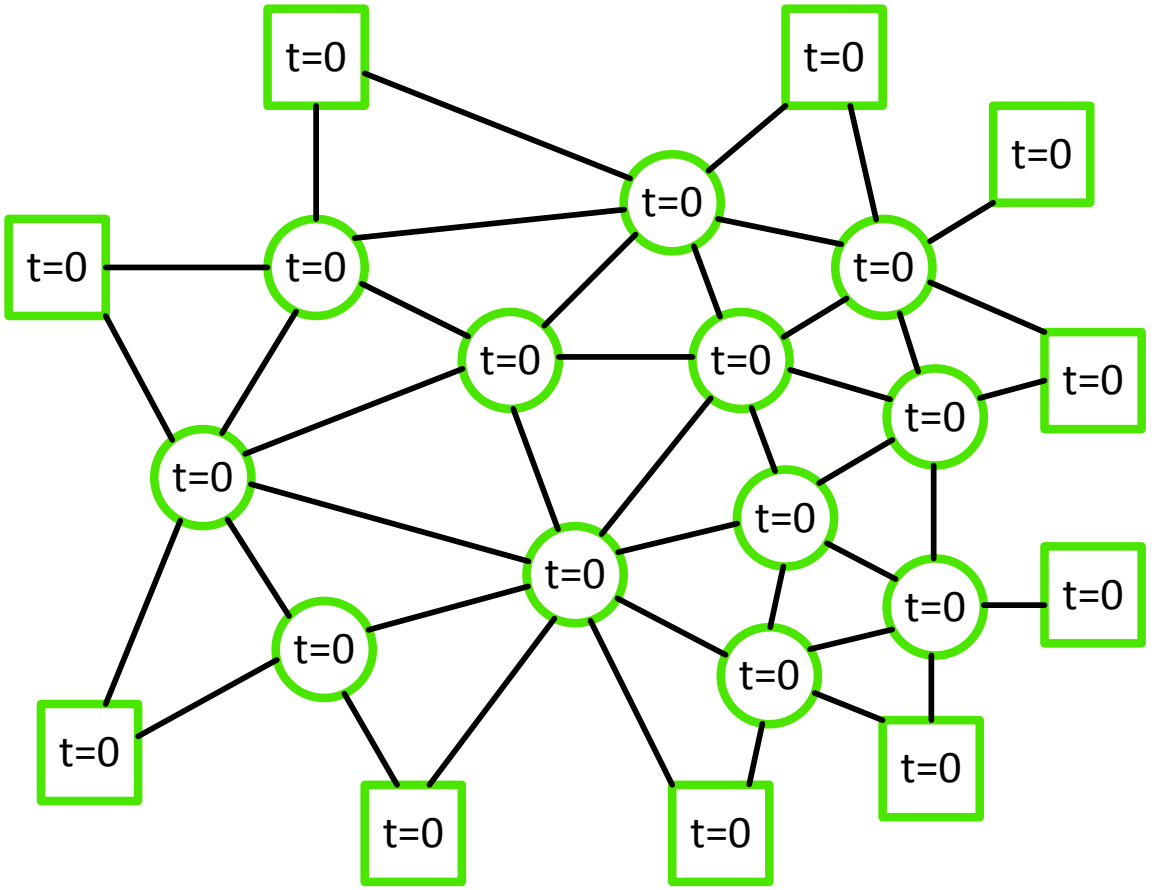


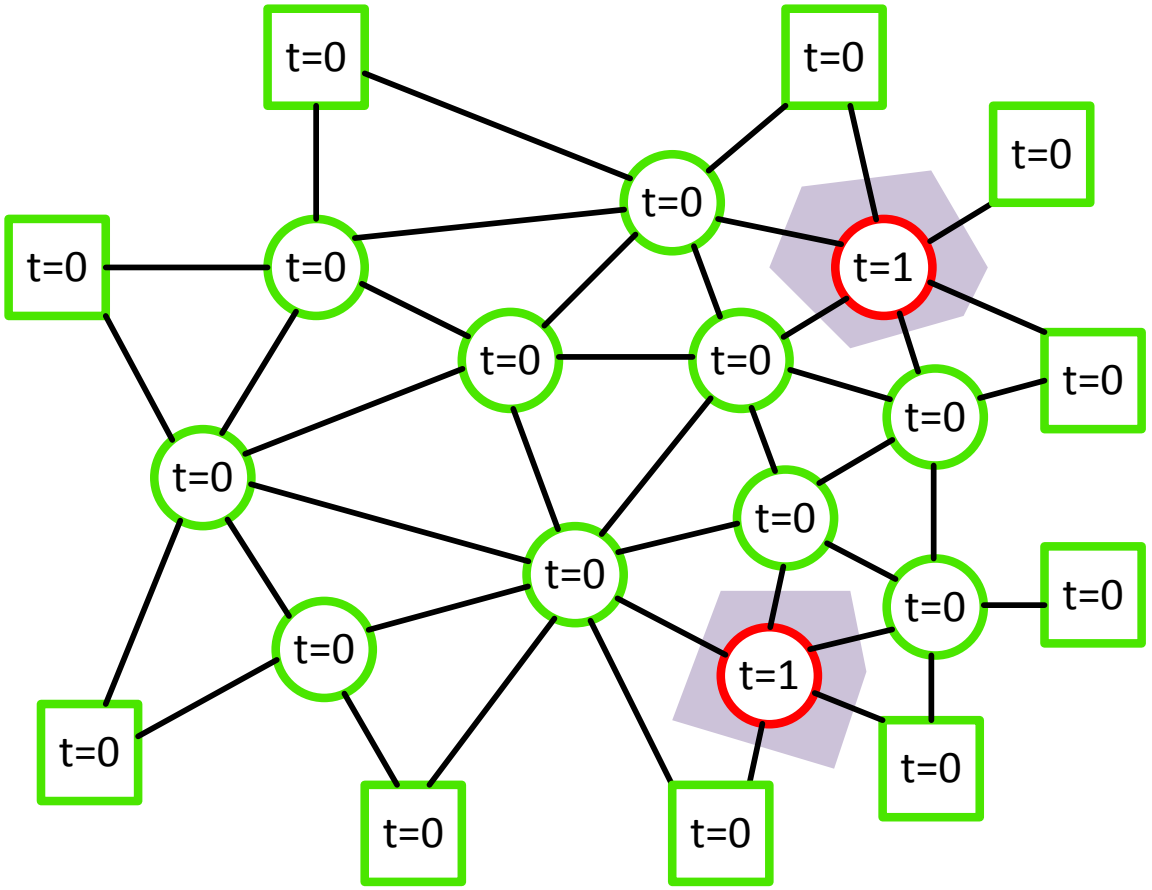
id:13

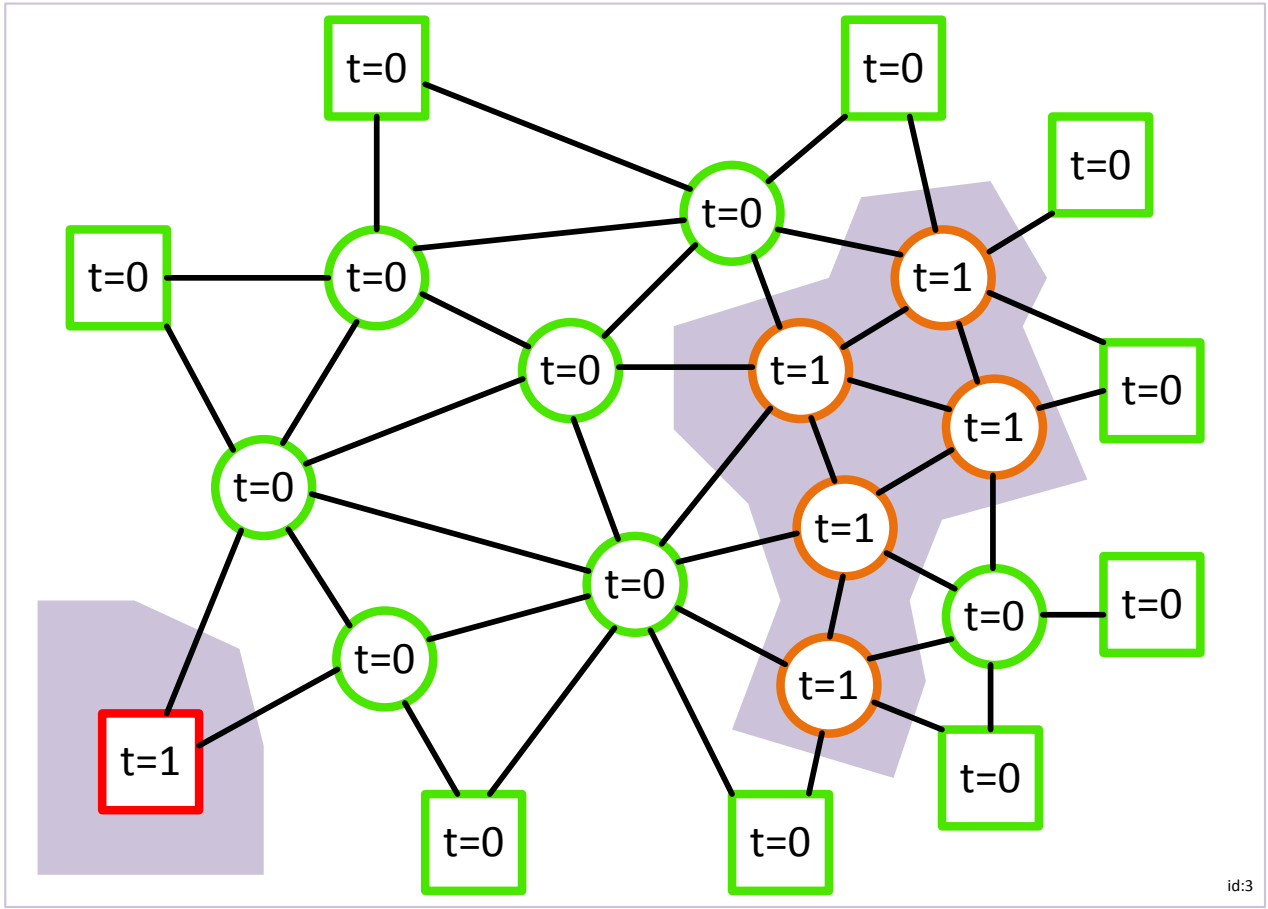
Over to you

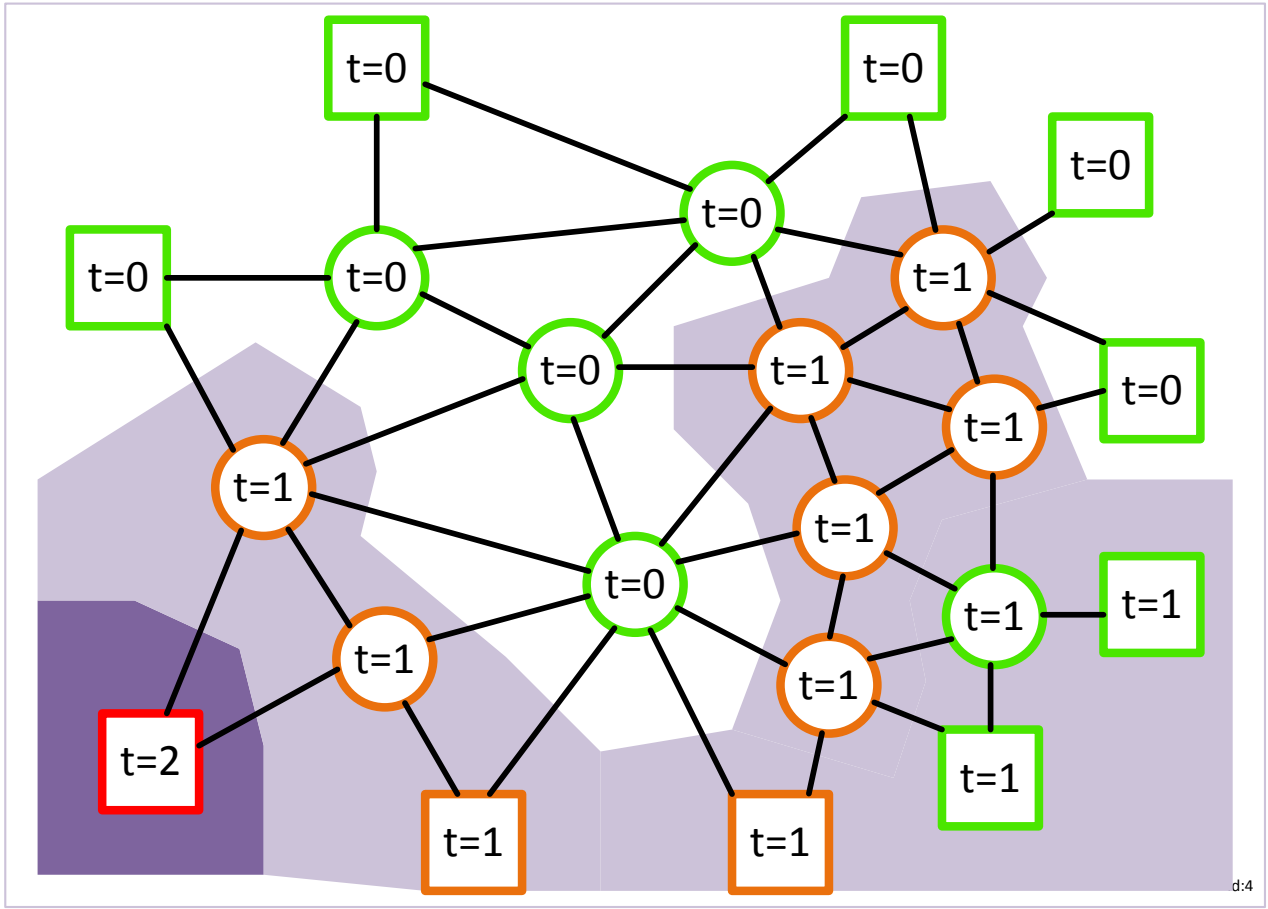
- There are diminishing returns of giving more and more detail
- Best approach is to think about how to solve:
  - Simple problems very well: CW5
  - Complex problems fairly well: CW6
- There will be the oral assessment in early summer term
  - Think about what you are doing as you do it
  - You will not have time to do everything
    - Remember what you could have done
    - Remember what you should have done (given time, hindsight, ...)

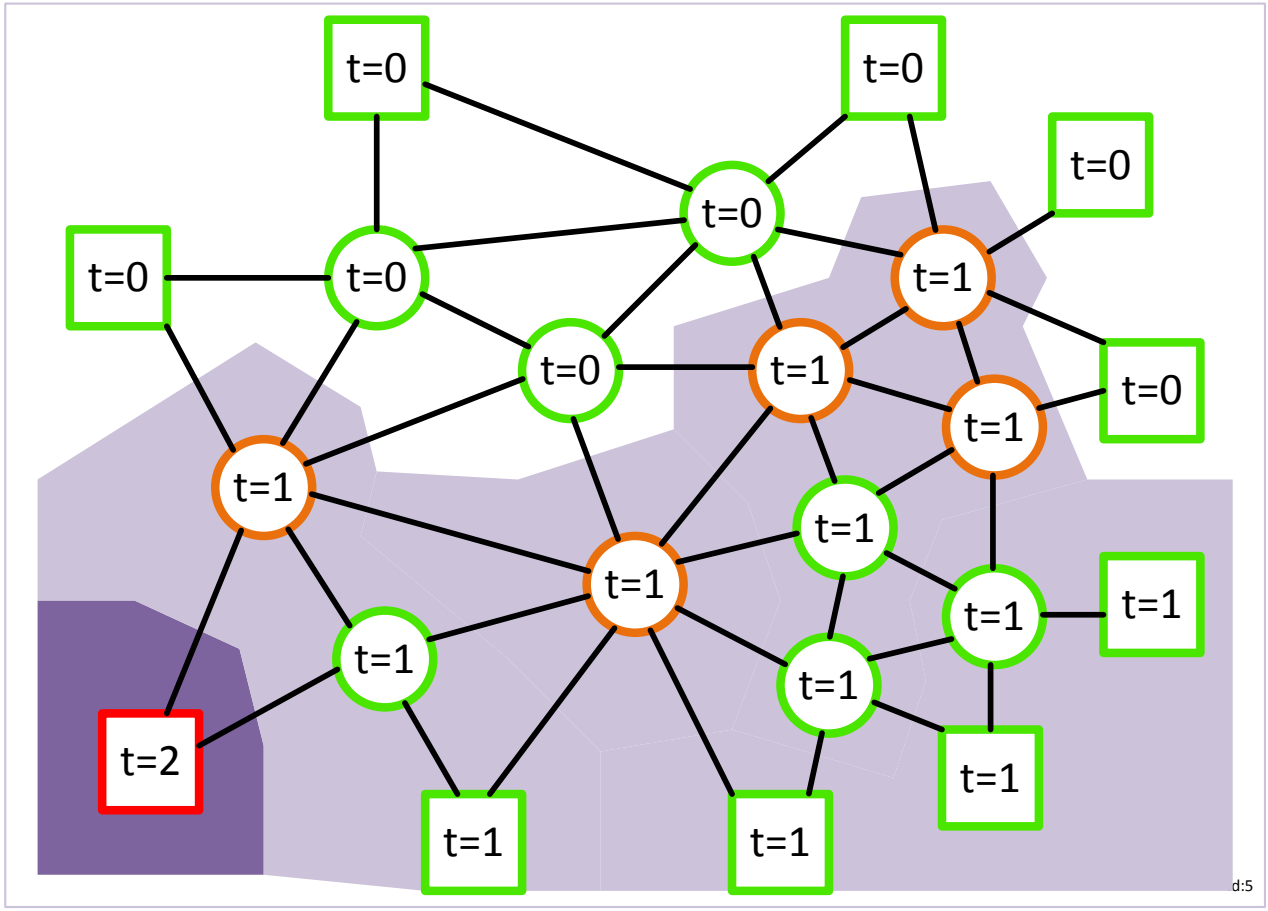

t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
id:0

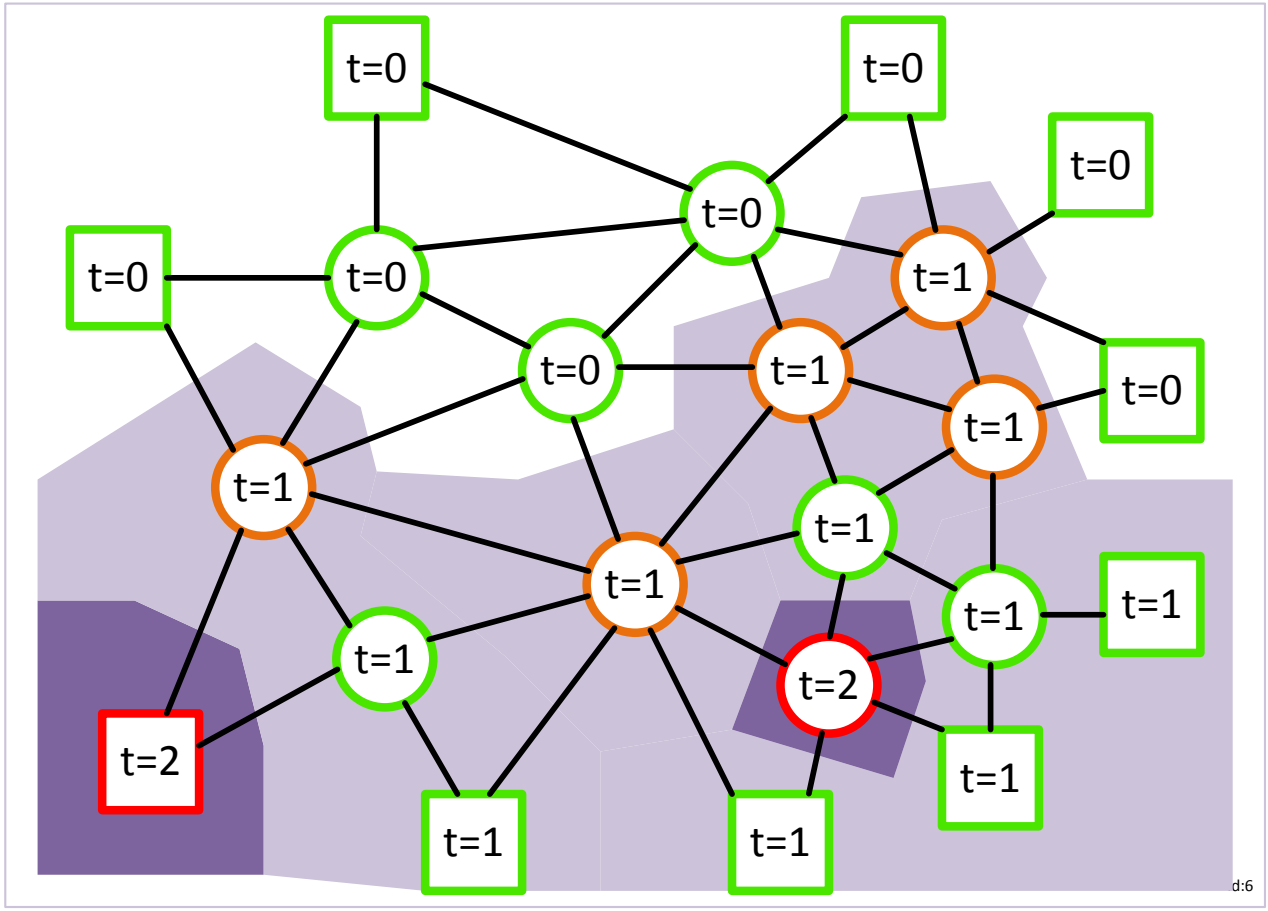

t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
id:1

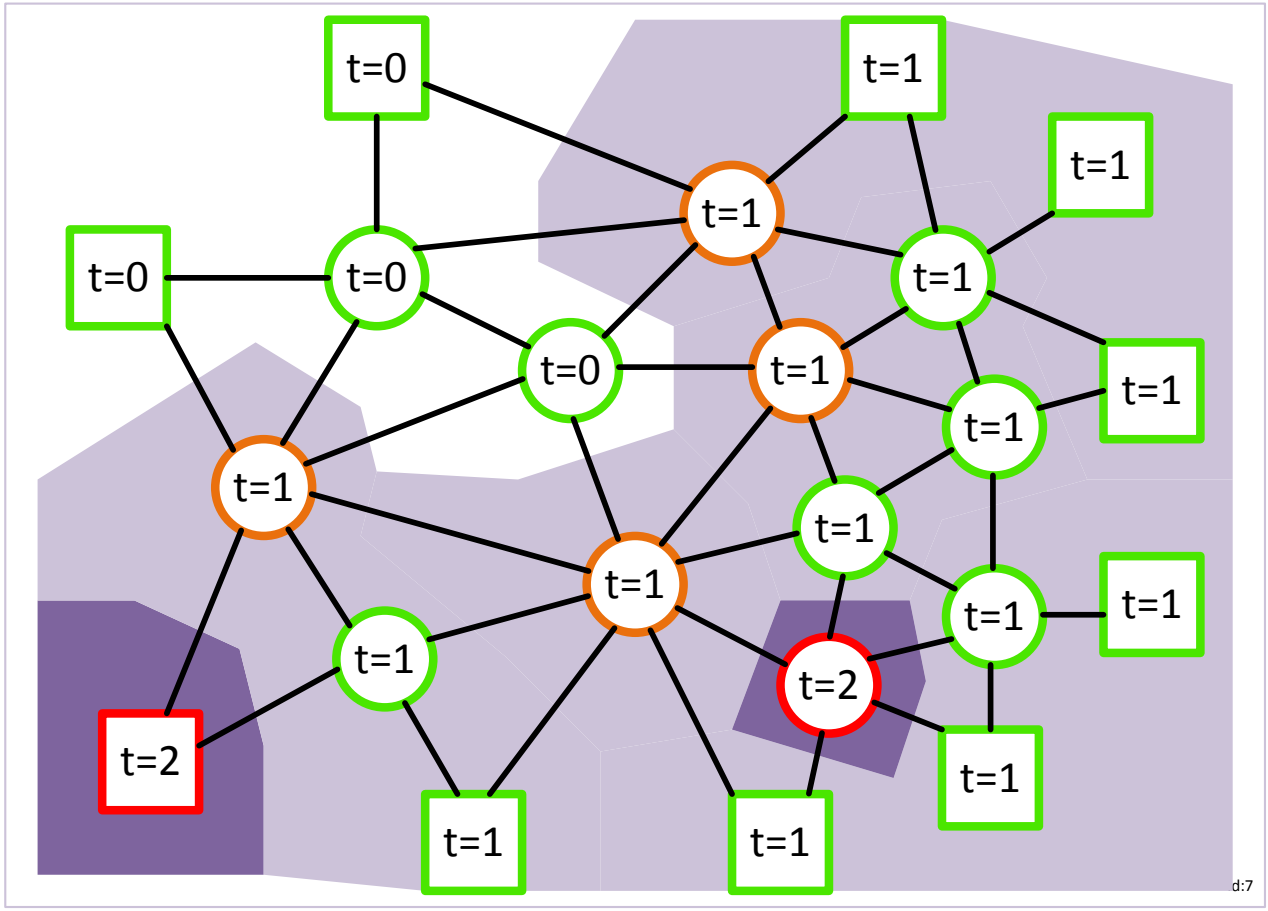

t=0
t=0
t=0
t=0
t=1
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=0
t=1
t=0
t=0
t=0
t=0
id:2

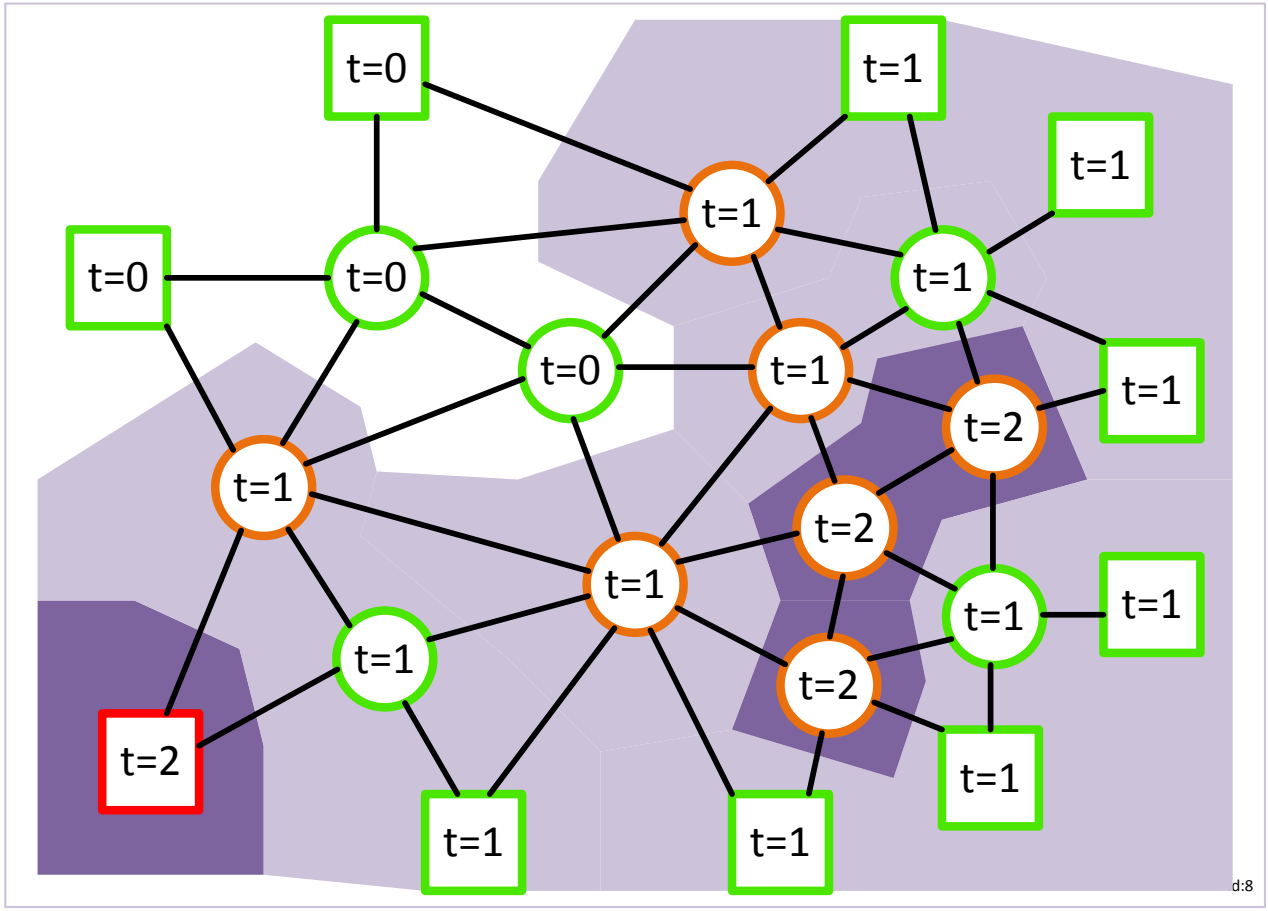

t=0
t=0
t=0
t=0
t=0
t=1
t=0
t=0
t=1
t=0
t=1
t=0
t=0
t=1
t=0
t=0
t=0
t=1
t=0
t=1
t=0
t=0
t=0
id:3

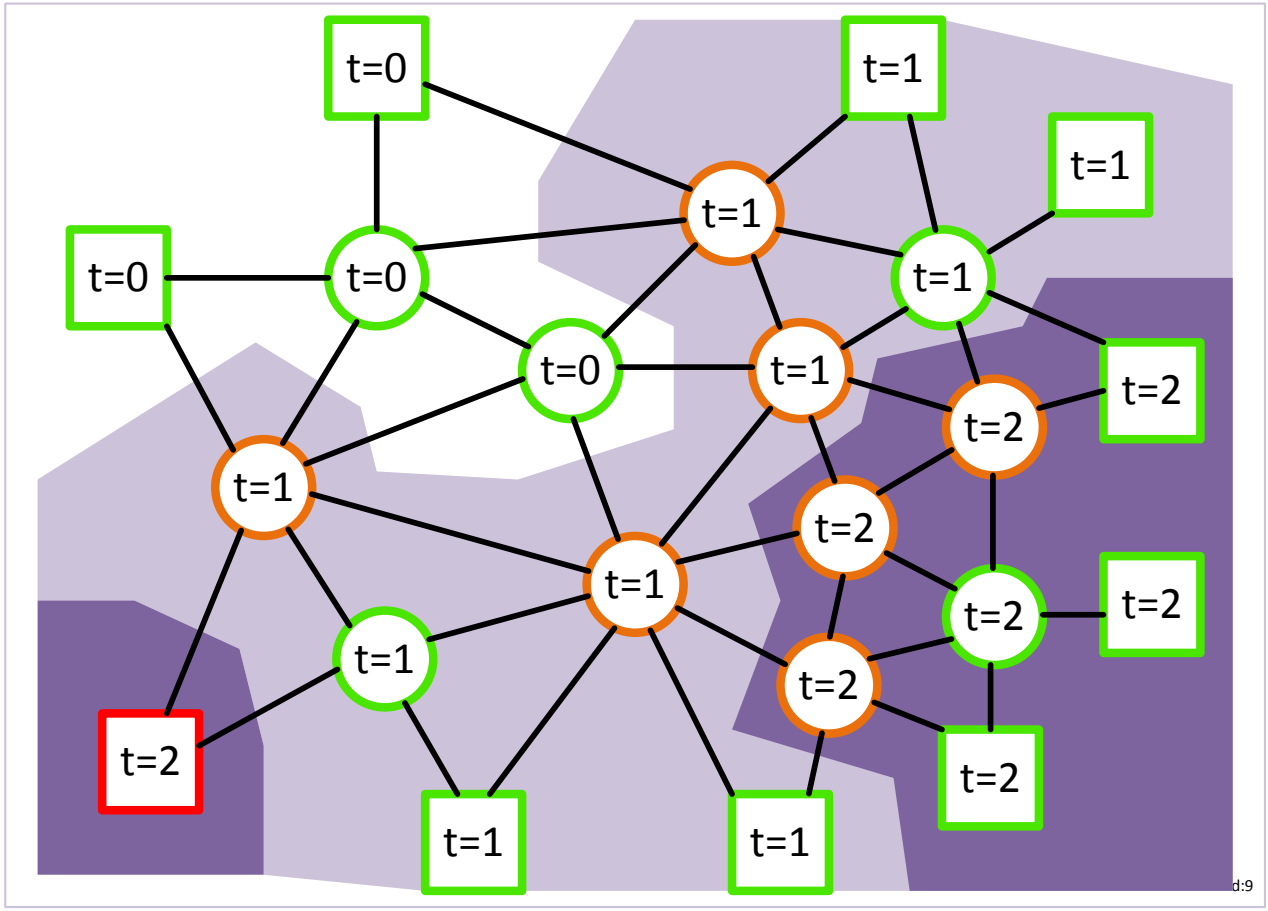

t=0
t=0
t=0
t=0
t=0
t=1
t=0
t=0
t=1
t=0
t=1
t=1
t=1
t=0
t=1
t=1
t=0
t=1
t=1
t=2
t=1
t=1
t=1
t=1
d:4

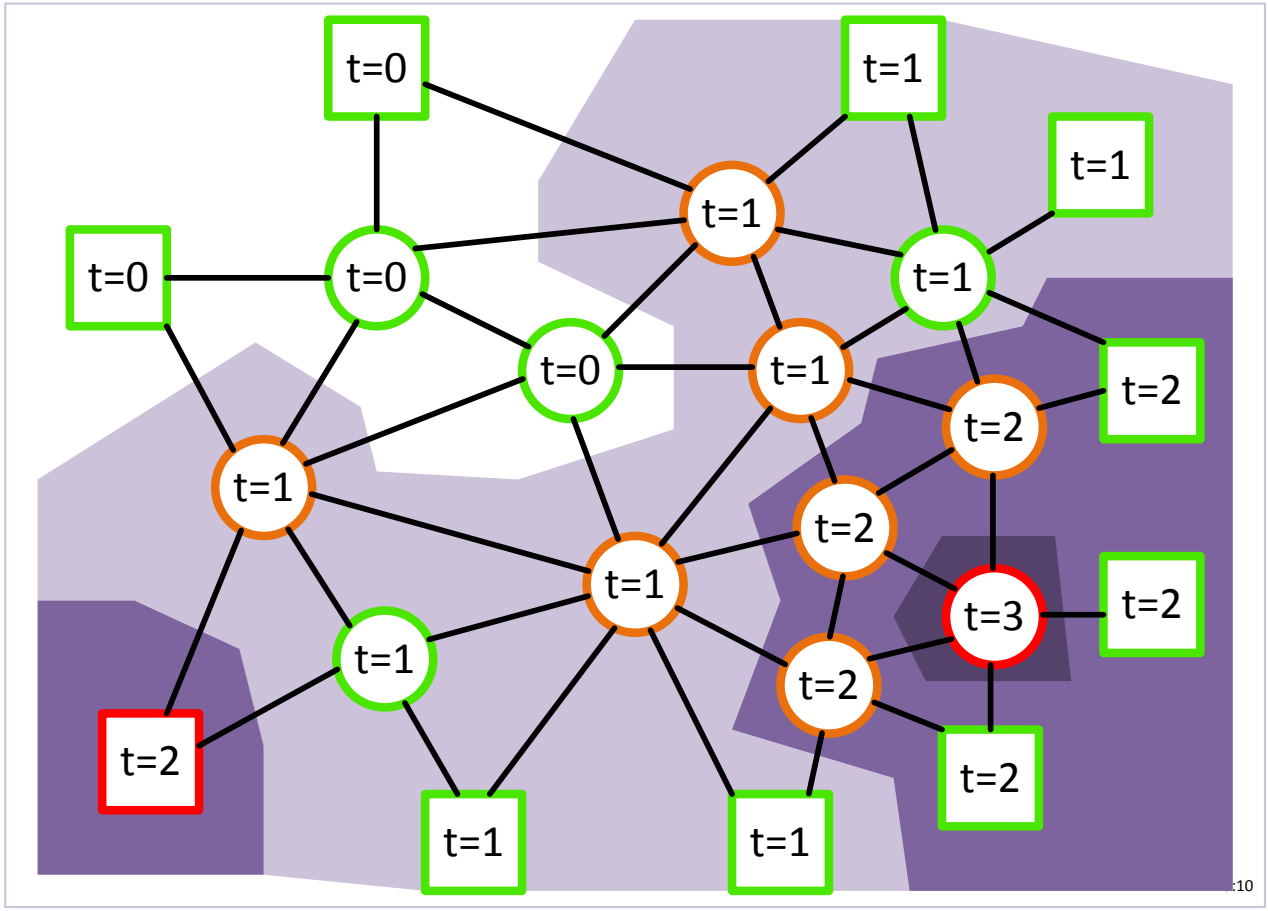

t=0
t=0
t=0
t=0
t=0
t=1
t=0
t=0
t=1
t=0
t=1
t=1
t=1
t=1
t=1
t=1
t=1
t=1
t=2
t=1
t=1
t=1
t=1
d:5


t=0
t=0
t=0
t=0
t=0
t=1
t=0
t=0
t=1
t=0
t=1
t=1
t=1
t=1
t=1
t=1
t=1
t=2
t=2
t=1
t=1
t=1
t=1
d:6

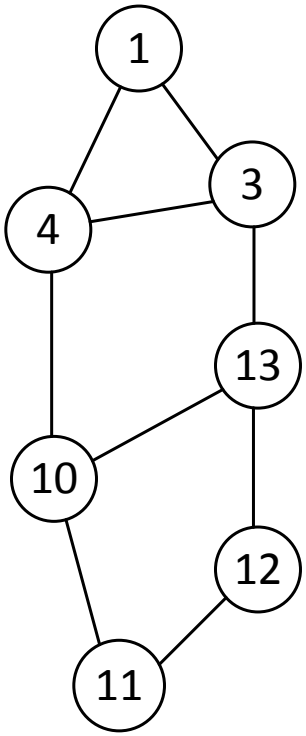

t=0
t=1
t=1
t=1
t=0
t=0
t=1
t=0
t=1
t=1
t=1
t=1
t=1
t=1
t=1
t=1
t=1
t=2
t=2
t=1
t=1
t=1
d:7

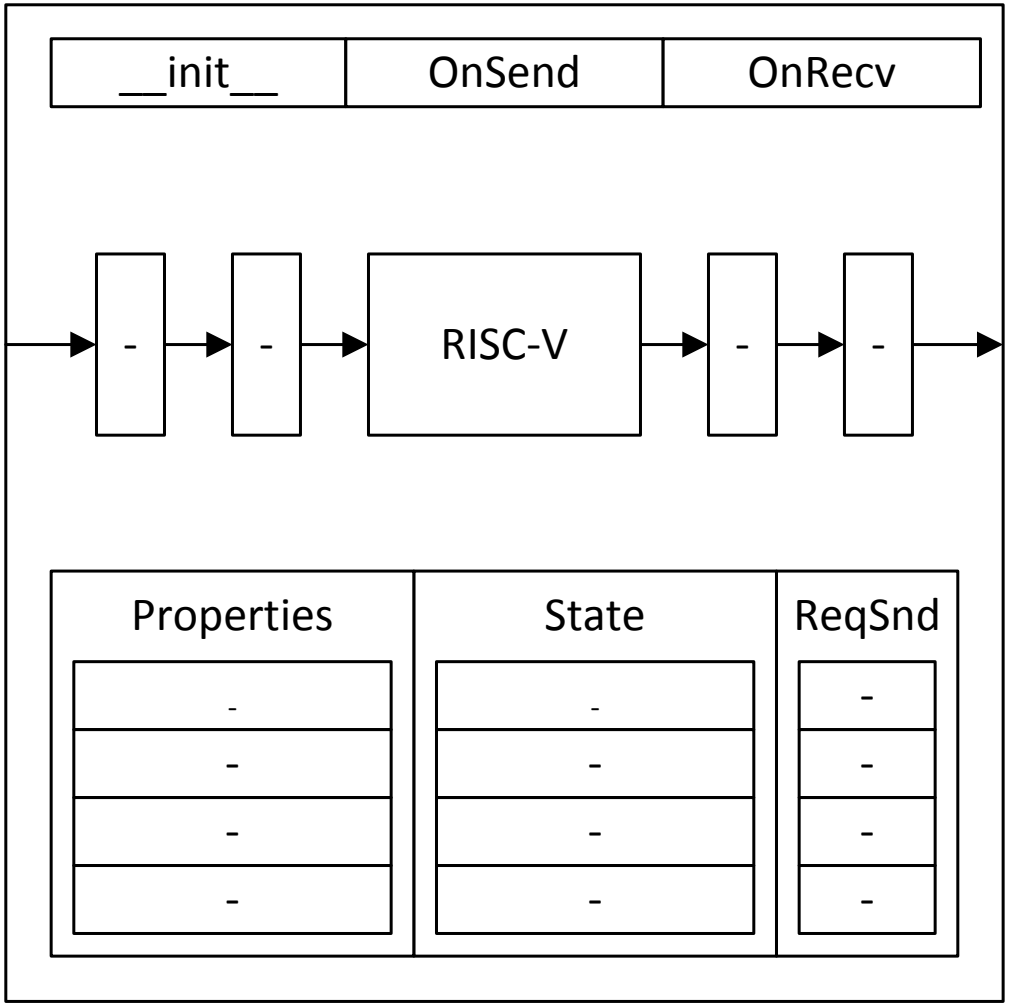

t=0
t=1
t=1
t=1
t=0
t=0
t=1
t=0
t=1
t=1
t=2
t=1
t=2
t=1
t=1
t=1
t=1
t=2
t=2
t=1
t=1
t=1
d:8

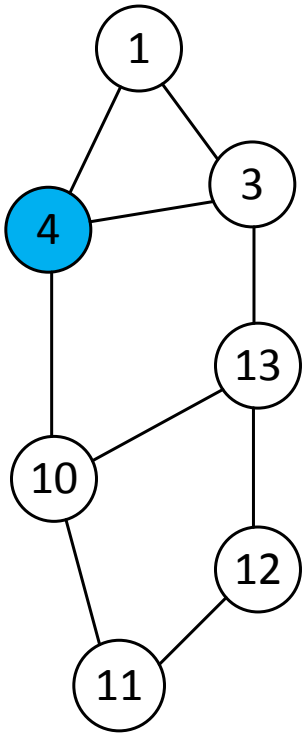

t=0
t=1
t=1
t=1
t=0
t=0
t=1
t=0
t=1
t=2
t=2
t=1
t=2
t=1
t=2
t=2
t=1
t=2
t=2
t=2
t=1
t=1
t=2

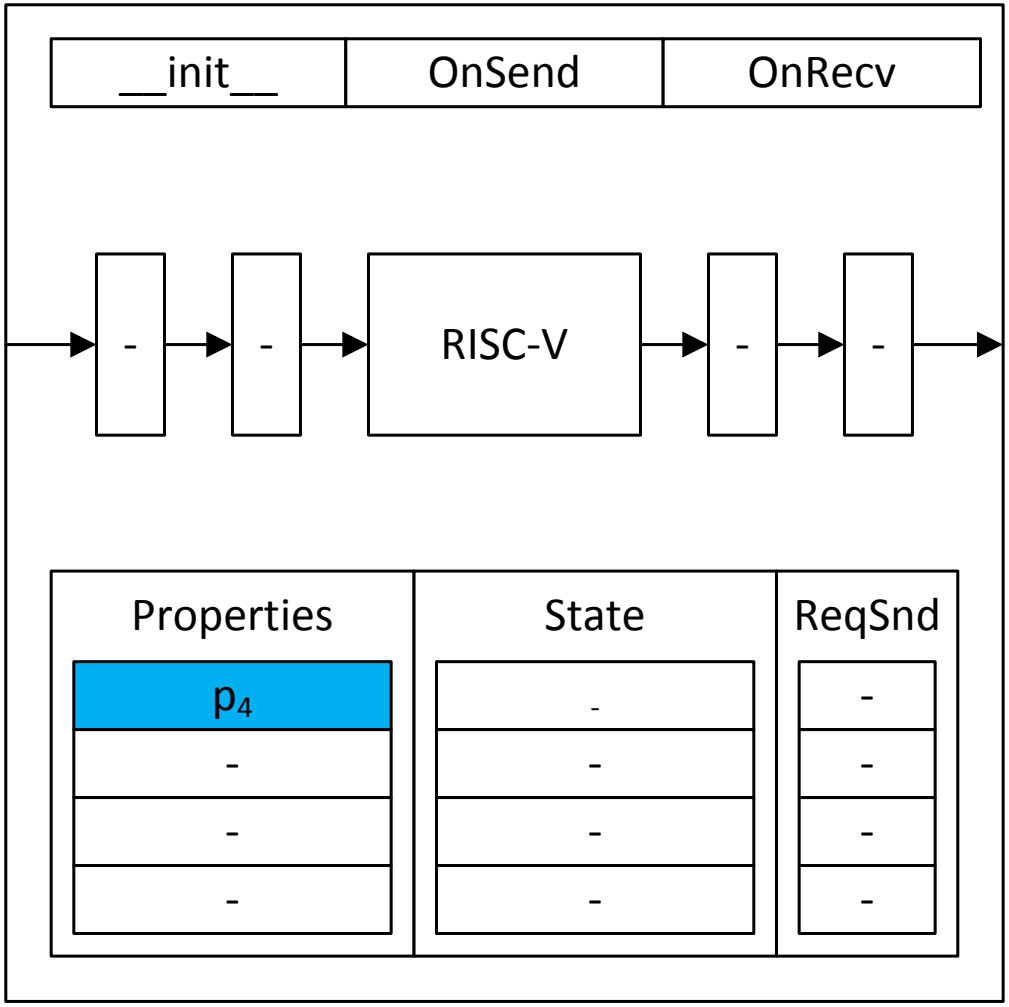

t=0
t=1
t=1
t=1
t=0
t=0
t=1
t=0
t=1
t=2
t=2
t=1
t=2
t=1
t=3
t=2
t=1
t=2
t=2
t=1
t=1

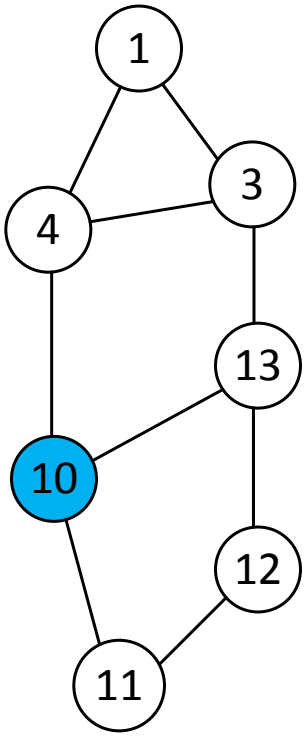
device to core mapping

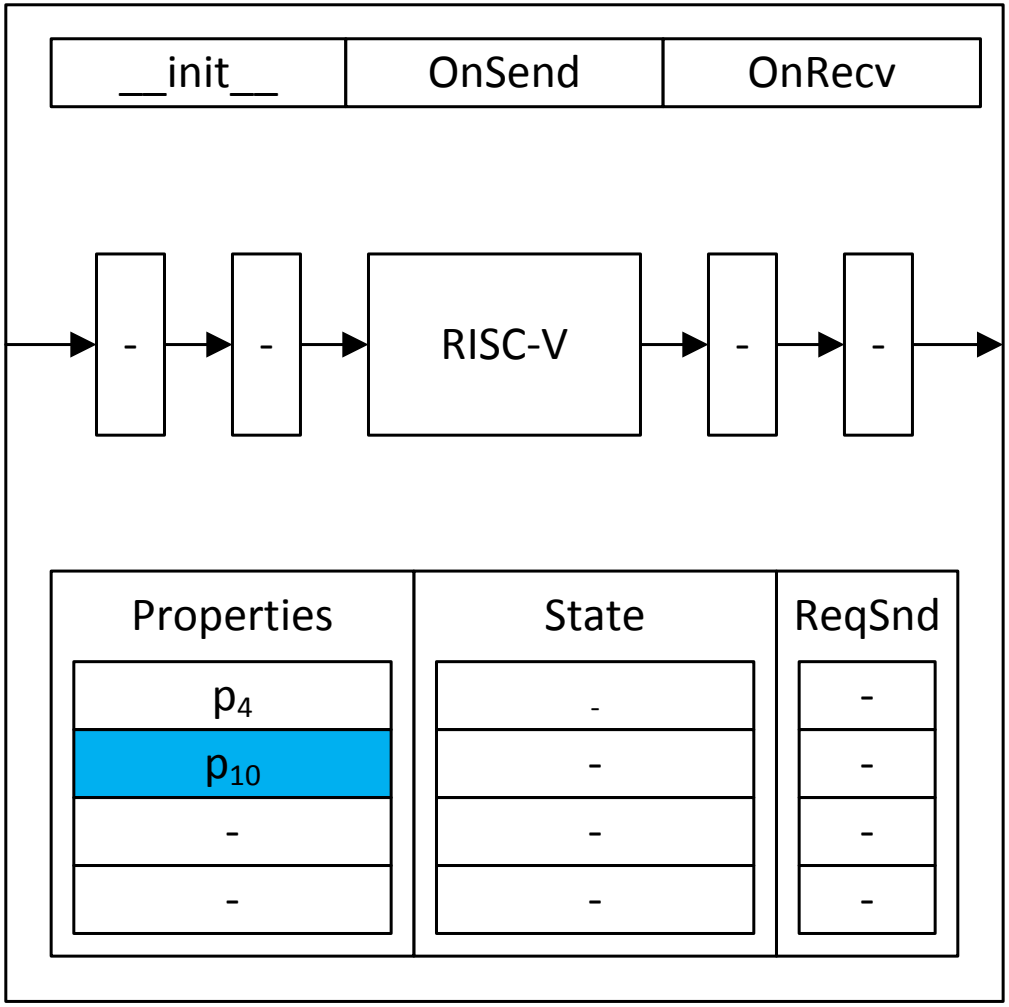

1
3
4
13
10
12
11

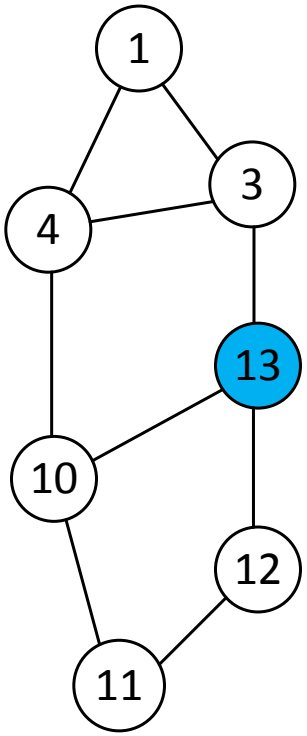

__init__
OnSend
OnRecv
-
-
RISC-V
-
-
Properties
State
ReqSnd
-
-
-
-
-
-
-
-
-
-
-
-

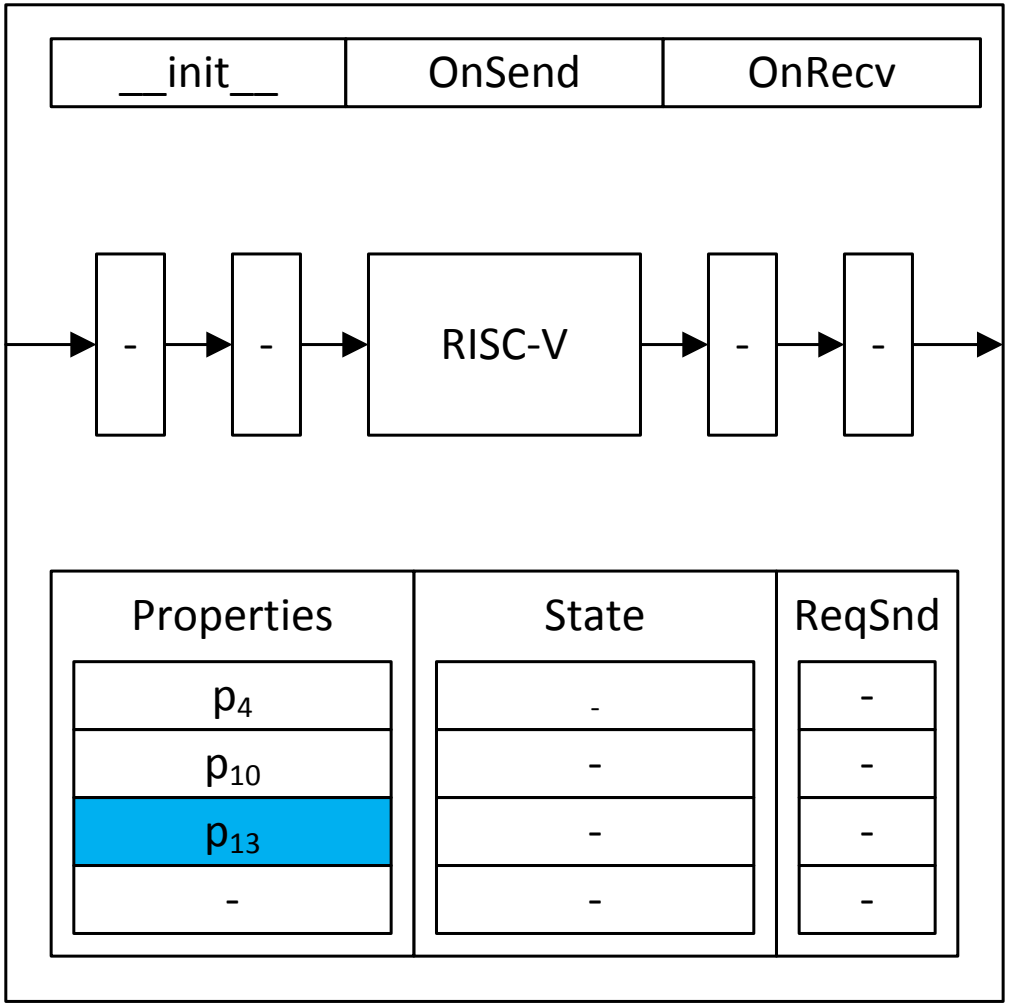

1
3
4
13
10
12
11

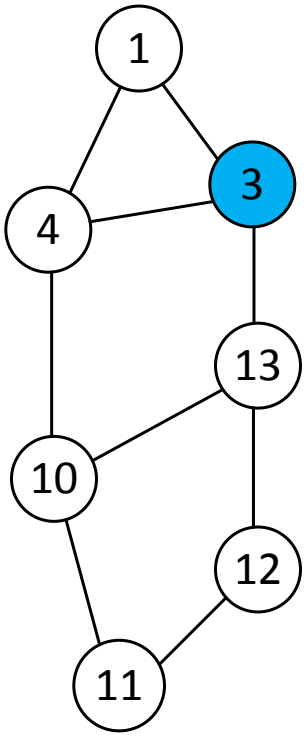

__init__
OnSend
OnRecv
-
-
RISC-V
-
-
Properties
State
ReqSnd
p4
-
-
-
-
-
-
-
-
-
-
-
-

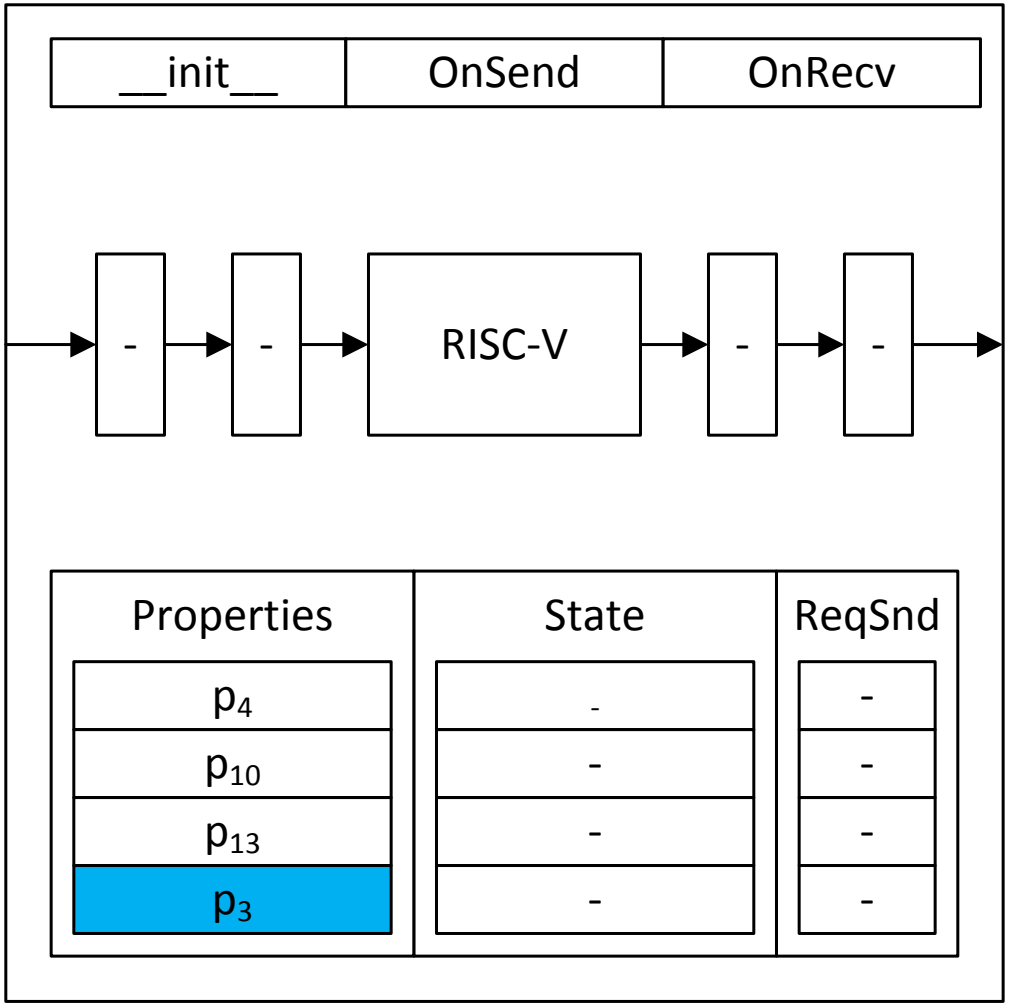

1
3
4
13
10
12
11

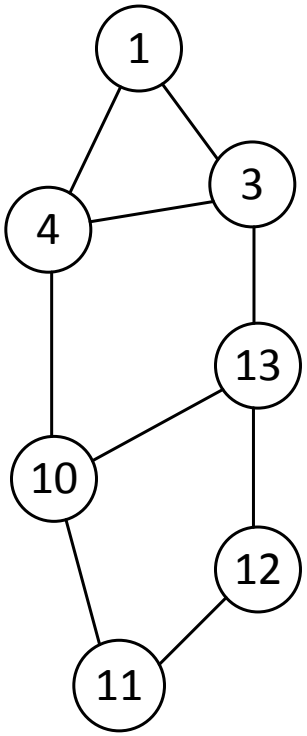

__init__
OnSend
OnRecv
-
-
RISC-V
-
-
Properties
State
ReqSnd
p4
p10
-
-
-
-
-
-
-
-
-

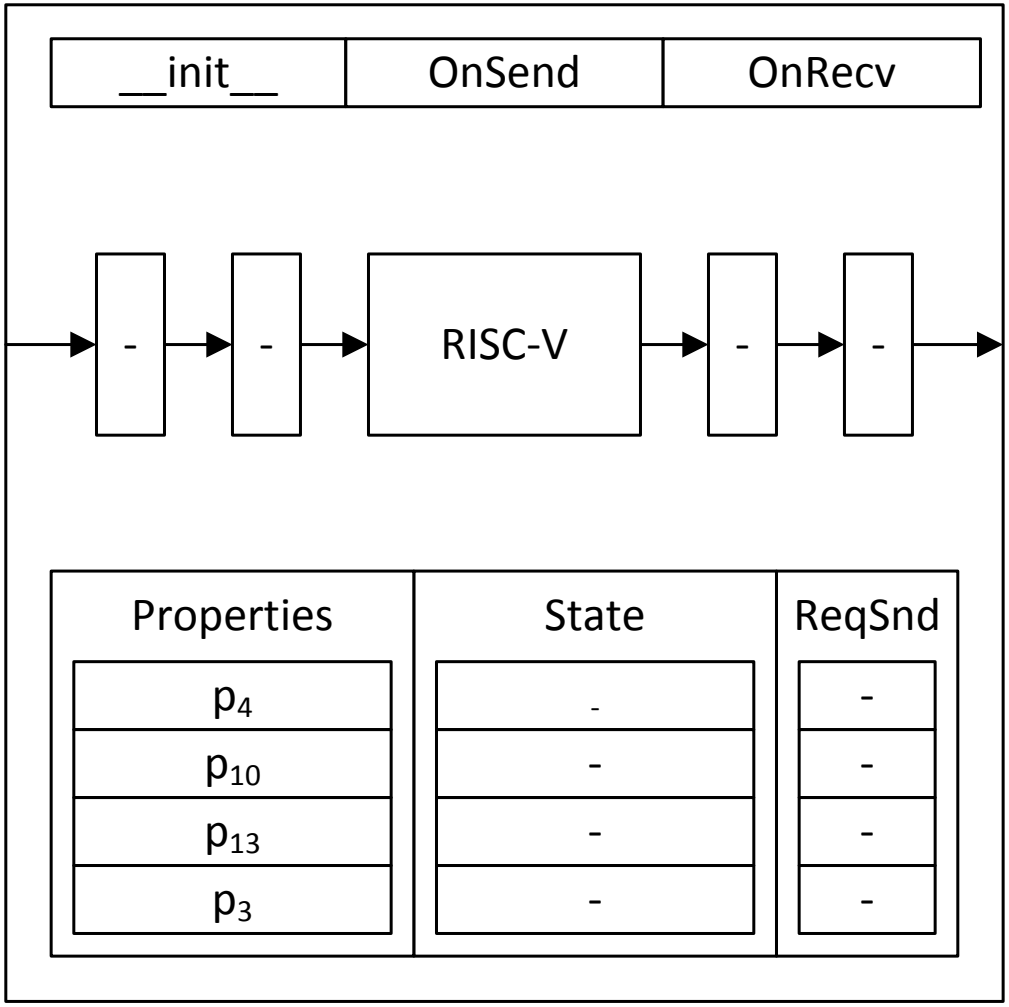

1
3
4
13
10
12
11

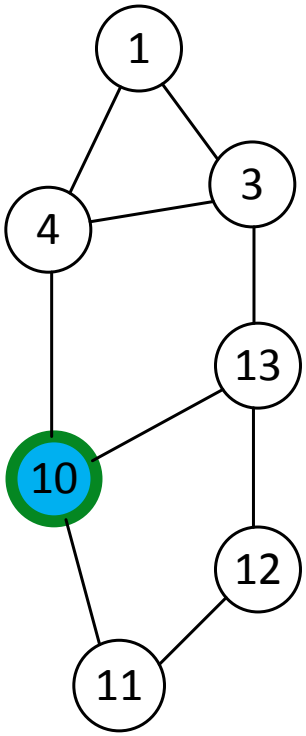

__init__
OnSend
OnRecv
-
-
RISC-V
-
-
Properties
State
ReqSnd
p4
p10
p13
-
-
-
-
-
-
-
-

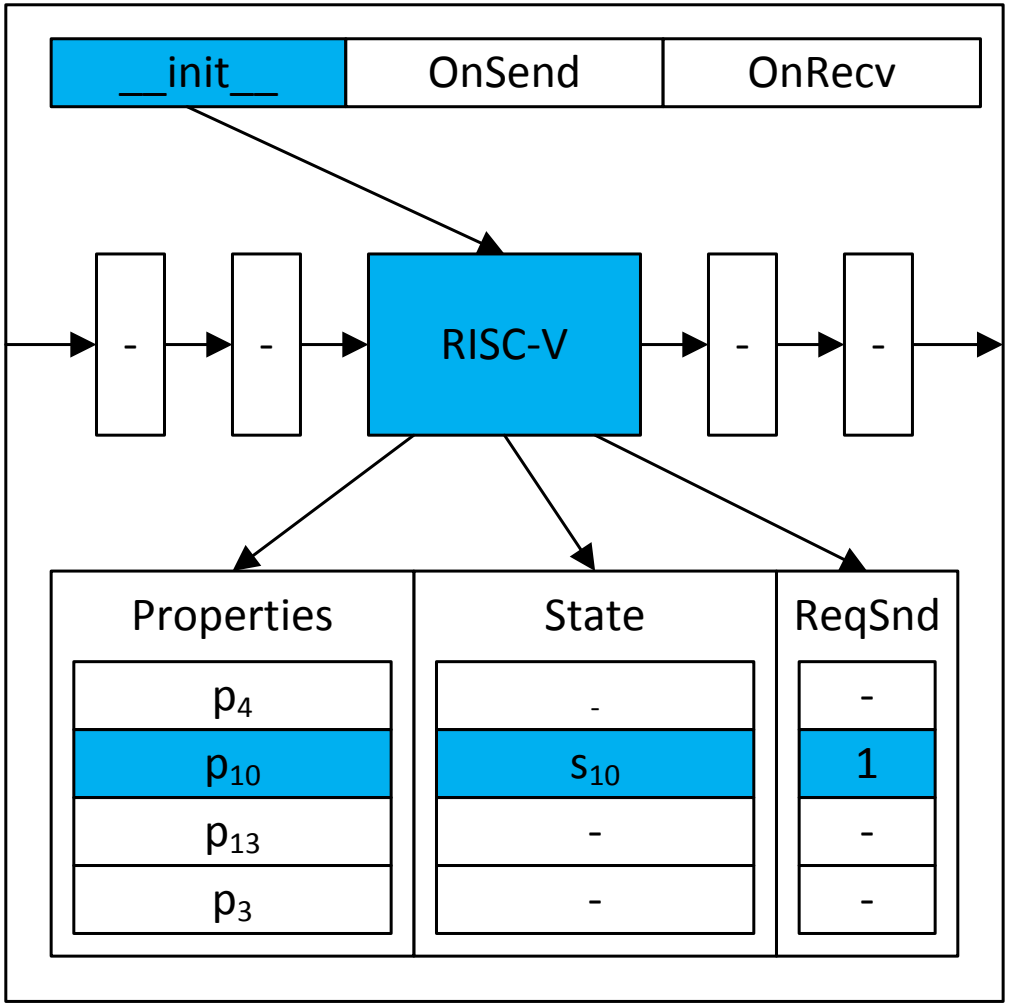

1
3
4
13
10
12
11

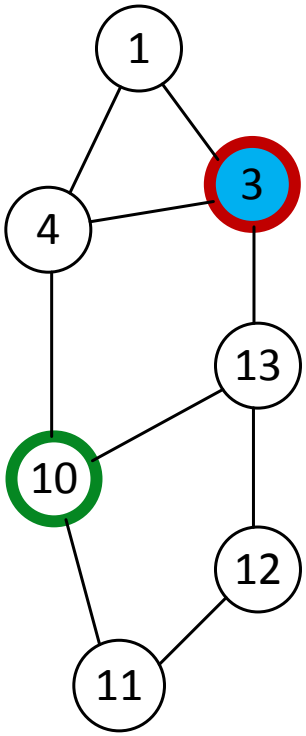

__init__
OnSend
OnRecv
-
-
RISC-V
-
-
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
-
-
-
-
-
-
-
-

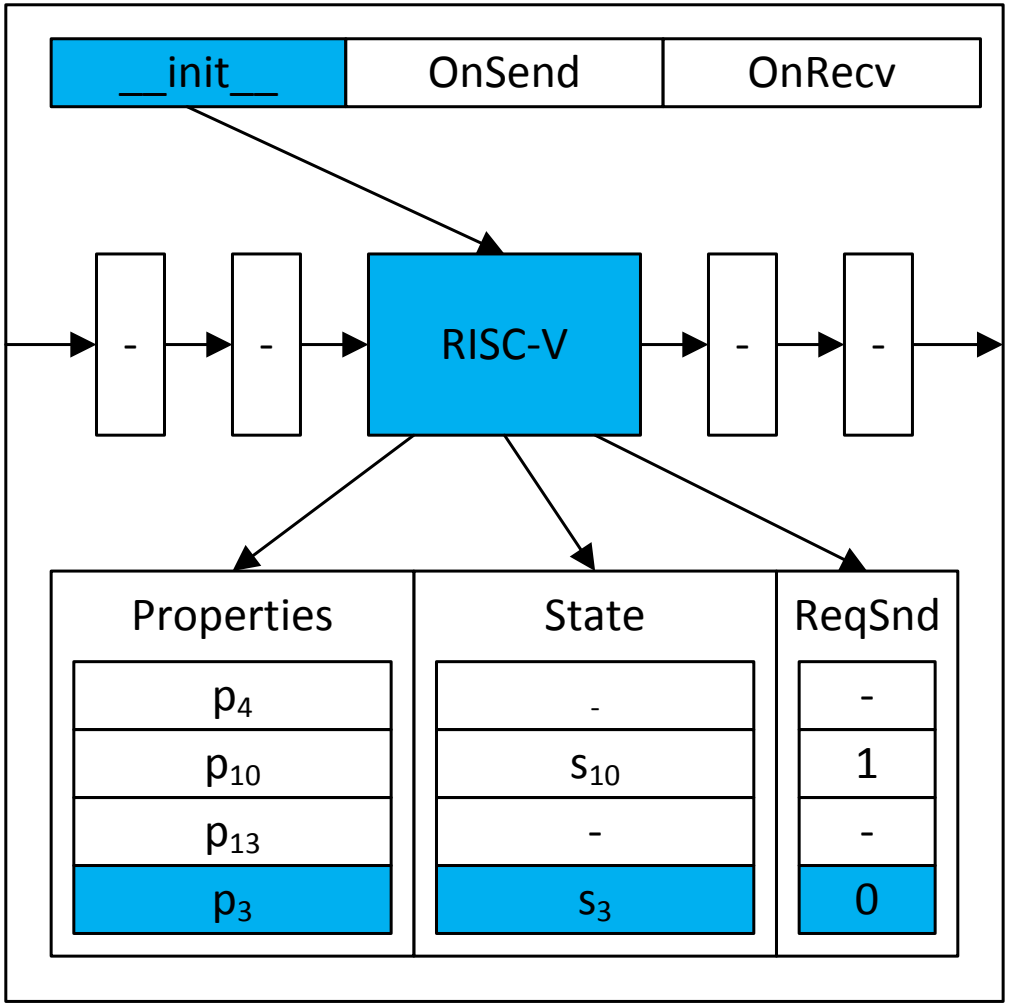

1
3
4
13
10
12
11

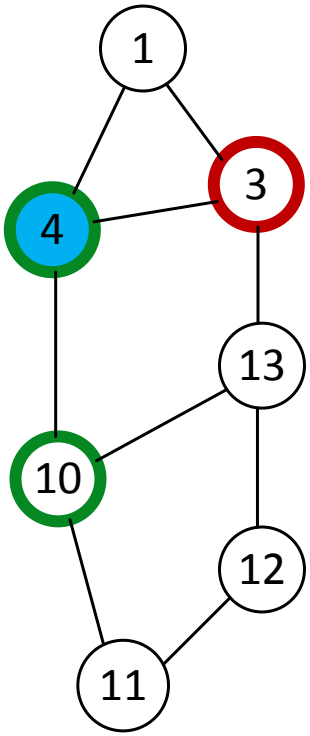

__init__
OnSend
OnRecv
-
-
RISC-V
-
-
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
-
-
-
-
-
-
-
-

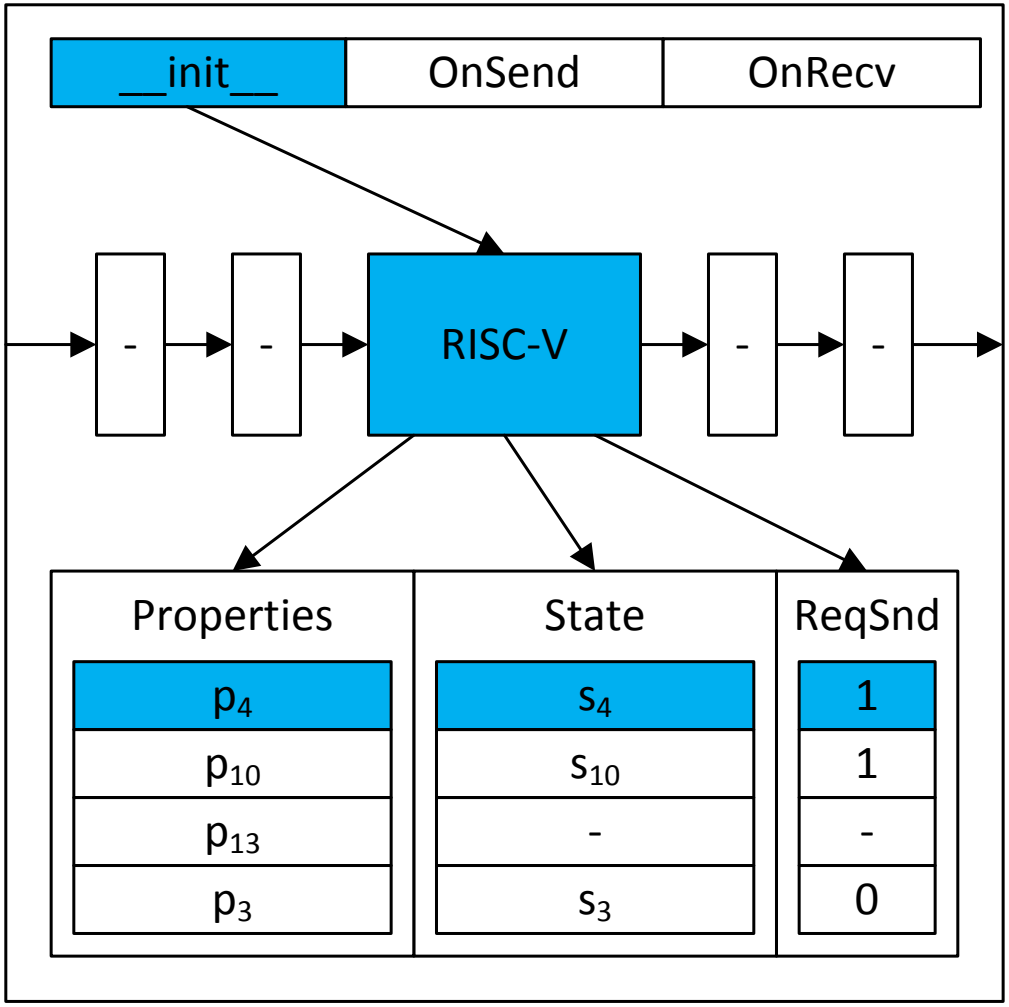

1
3
4
13
10
12
11

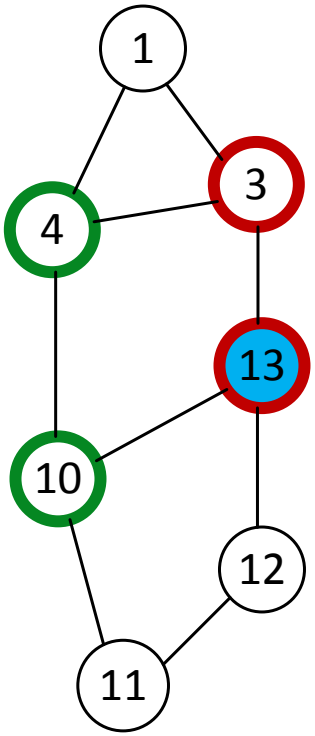

__init__
OnSend
OnRecv
-
-
RISC-V
-
-
Properties
State
ReqSnd
p4
p10
p13
p3
-
s10
-
-
-
1
-
-

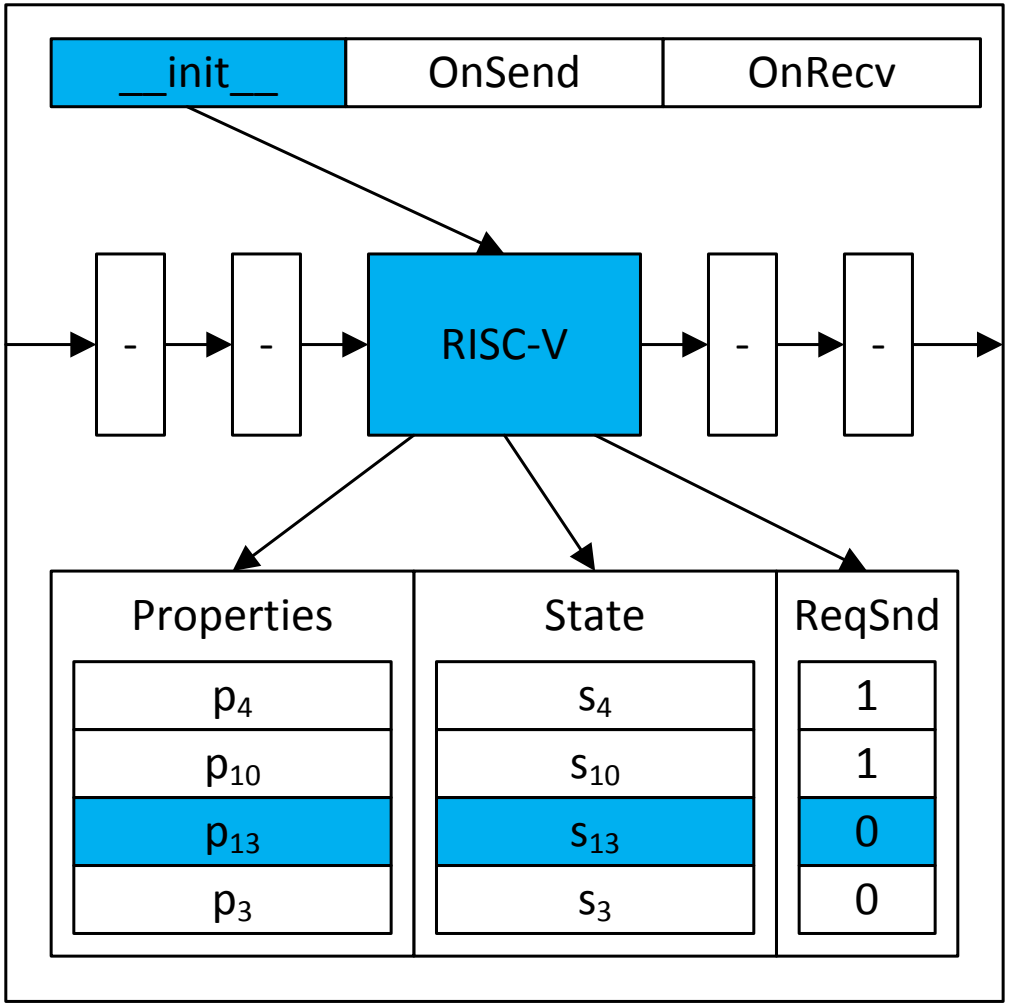

1
3
4
13
10
12
11

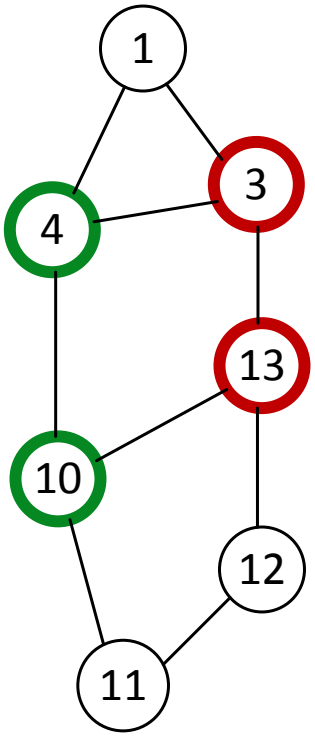

__init__
OnSend
OnRecv
-
-
RISC-V
-
-
Properties
State
ReqSnd
p4
p10
p13
p3
-
s10
-
s3
-
1
-
0

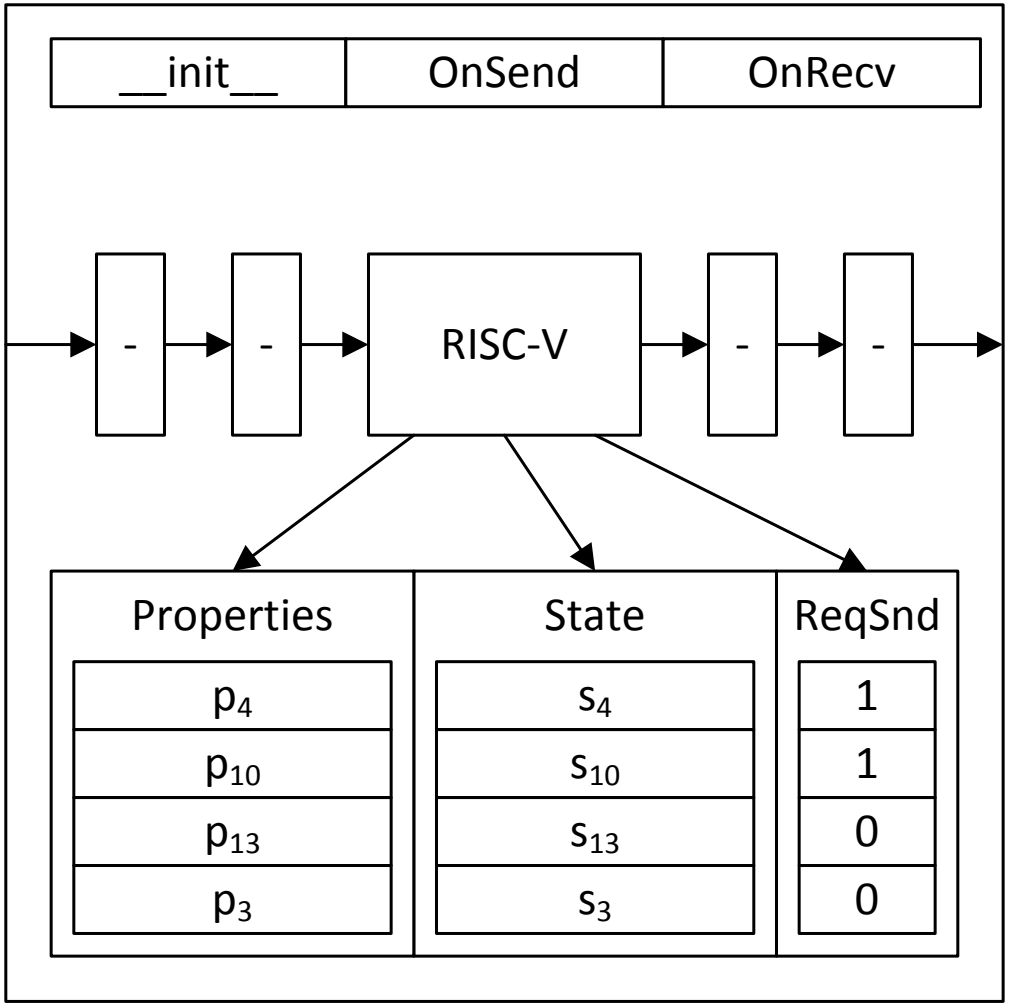

1
3
4
13
10
12
11

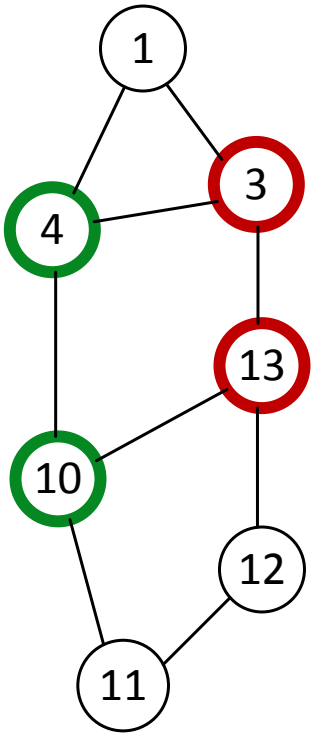

__init__
OnSend
OnRecv
-
-
RISC-V
-
-
Properties
State
ReqSnd
p4
p10
p13
p3
s4
s10
-
s3
1
1
-
0

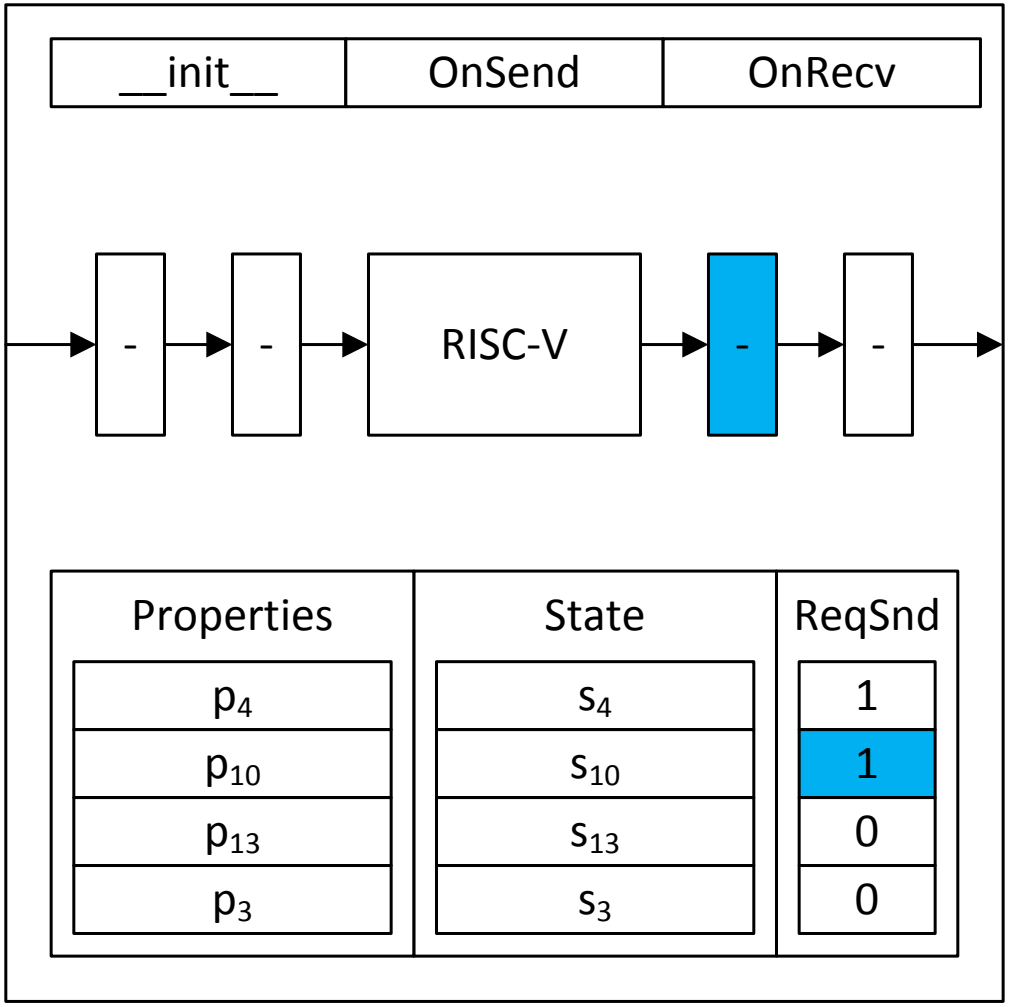

1
3
4
13
10
12
11

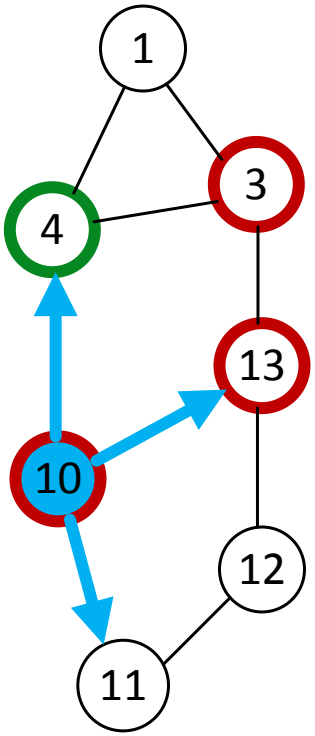

__init__
OnSend
OnRecv
-
-
RISC-V
-
-
Properties
State
ReqSnd
p4
p10
p13
p3
s4
s10
s13
s3
1
1
0
0

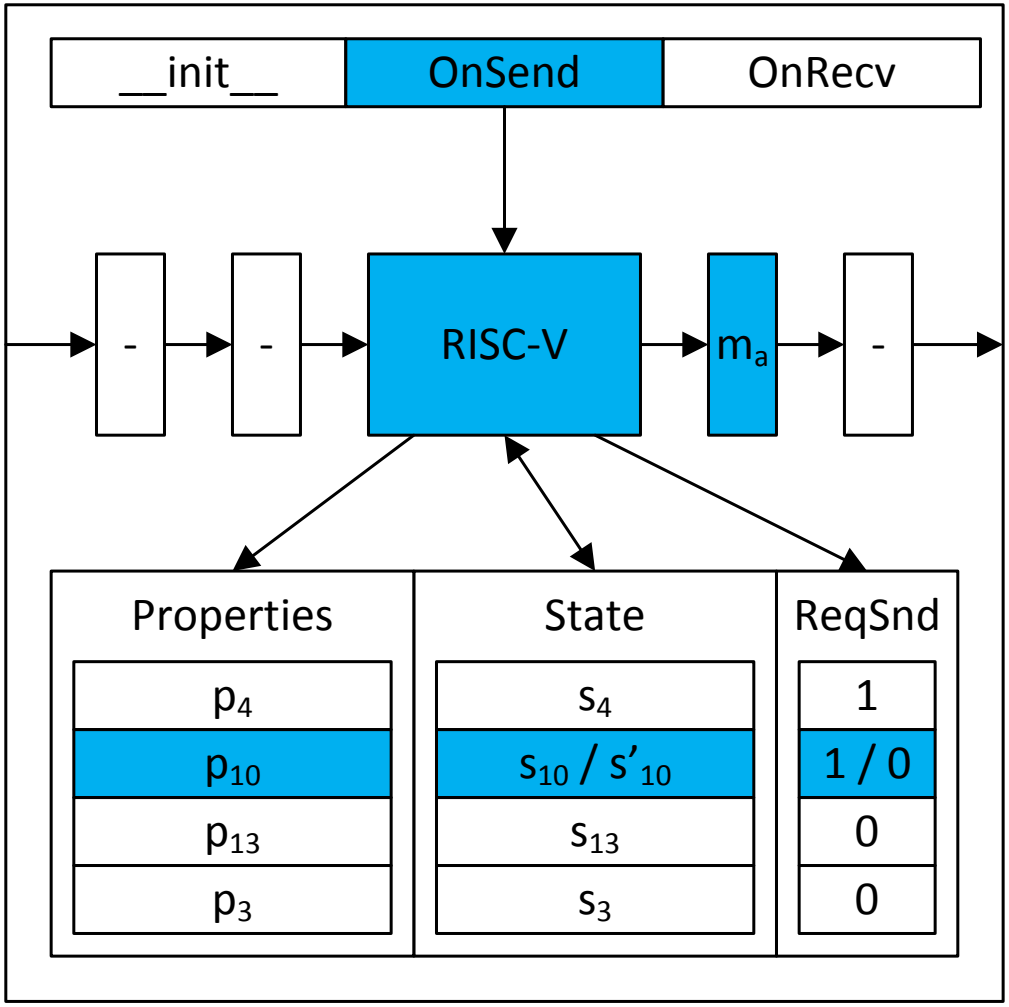

1
3
4
13
10
12
11

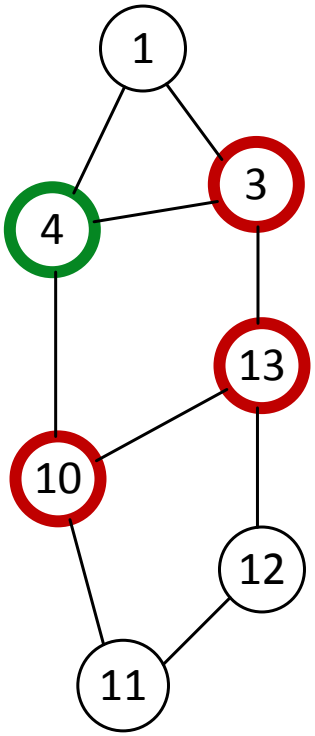

__init__
OnSend
OnRecv
-
-
RISC-V
-
-
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s_4
s_{10}
s_{13}
s_3
1
1
0
0

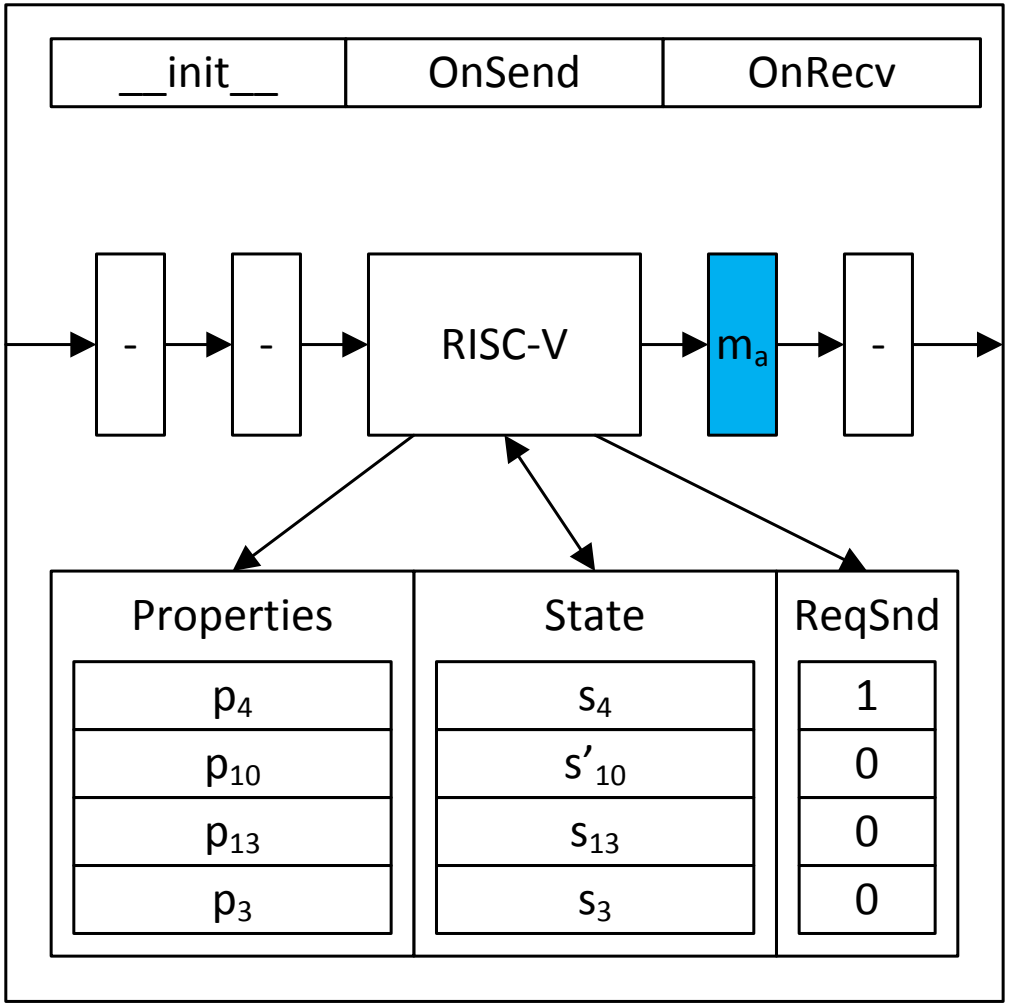

1
3
4
13
10
12
11

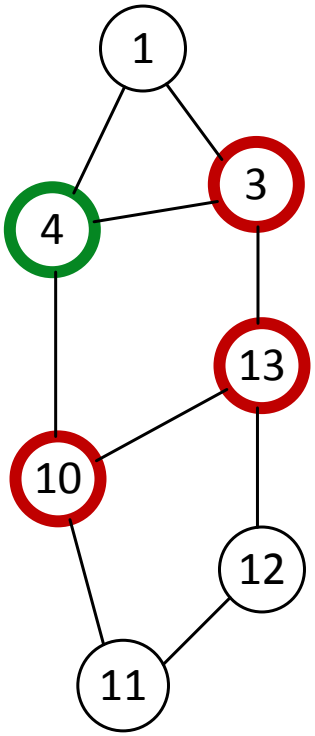

__init__
OnSend
OnRecv
-
-
RISC-V
-
-
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s_4
s_{10}
s_{13}
s_3
1
1
0
0

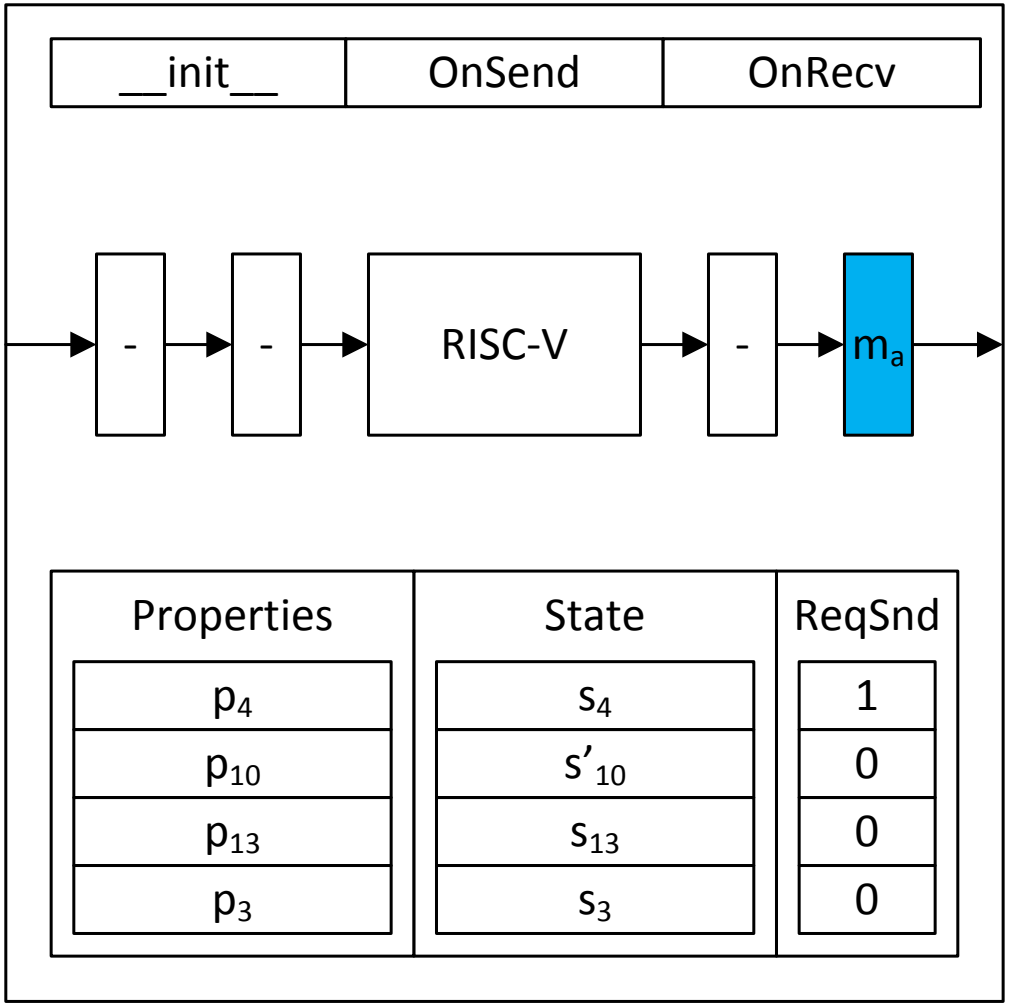

1
3
4
13
10
12
11

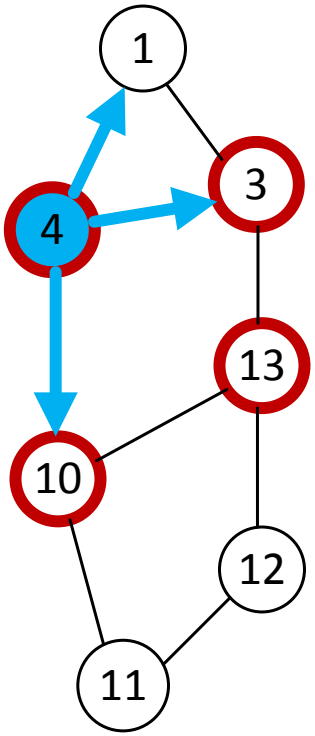

__init__
OnSend
OnRecv
-
-
RISC-V
m_a
-
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s_4
s_{10} / s'_{10}
s_{13}
s_3
1
1 / 0
0
0

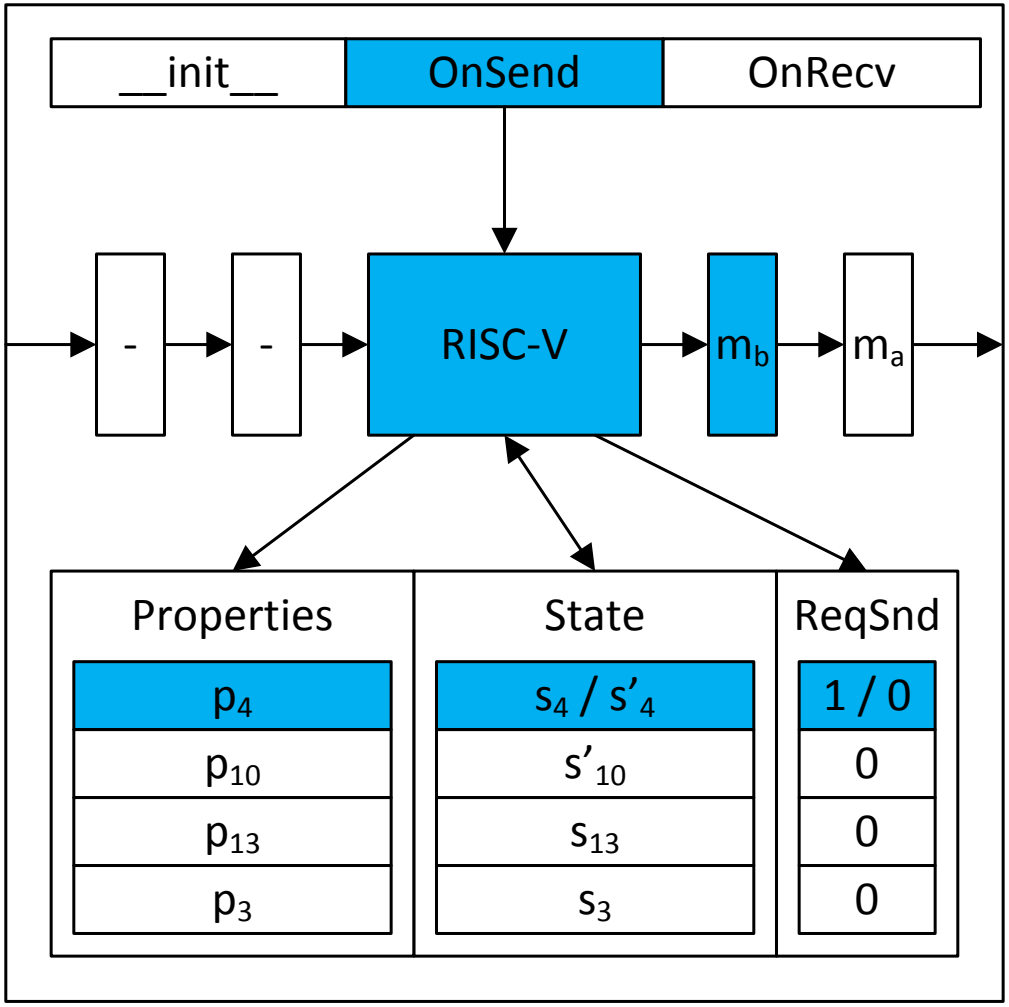

1
3
4
13
10
12
11

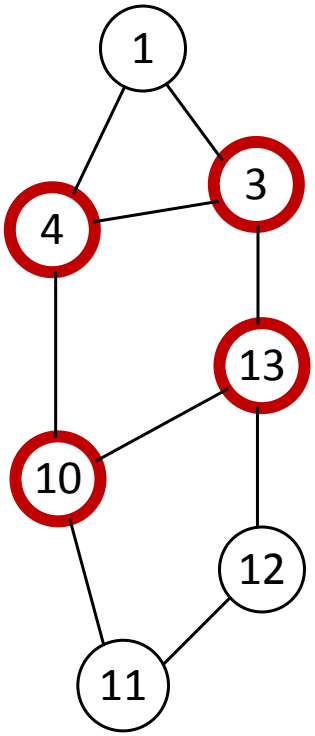

__init__
OnSend
OnRecv
-
-
RISC-V
m_a
-
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s_4
s'_{10}
s_{13}
s_3
1
0
0
0

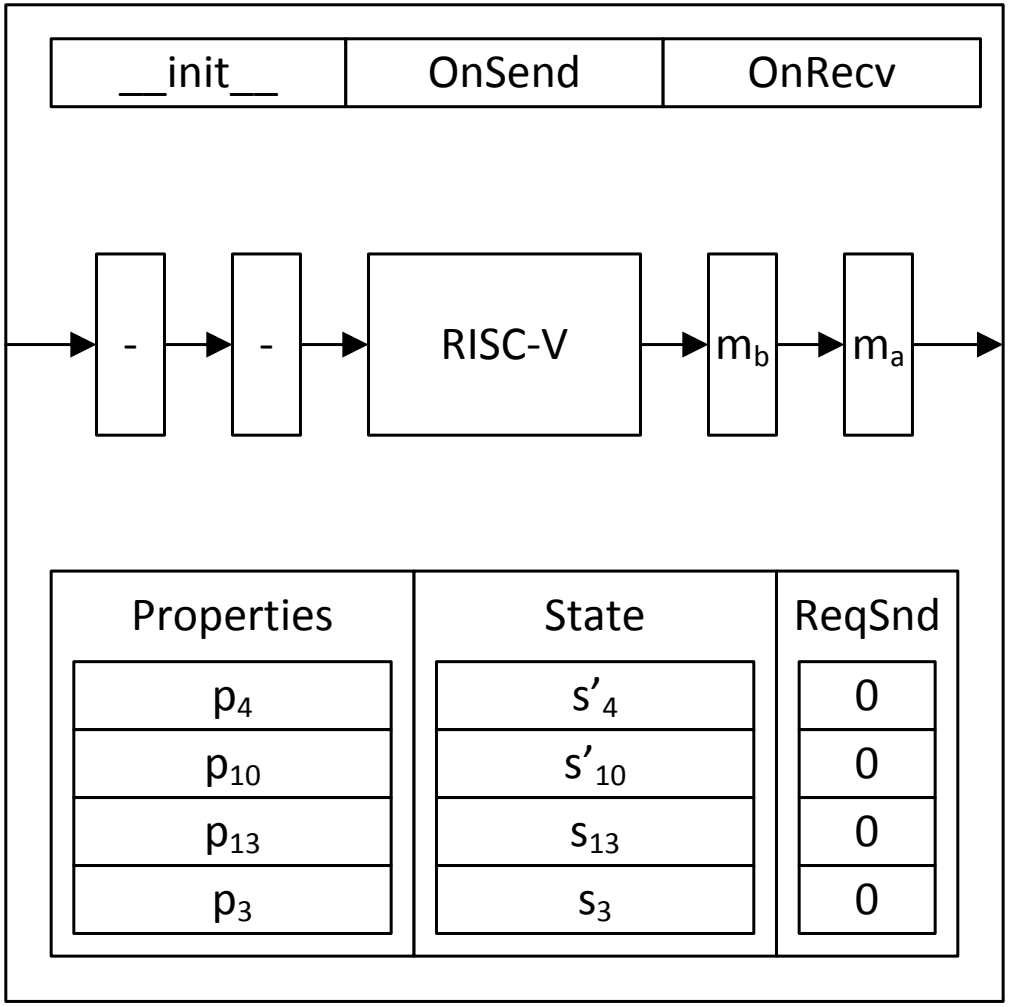

1
3
4
13
10
12
11

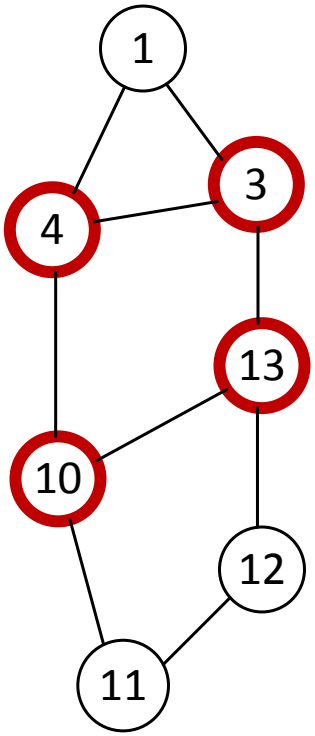

__init__
OnSend
OnRecv
-
-
RISC-V
-
m_a
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s_4
s'_{10}
s_{13}
s_3
1
0
0
0

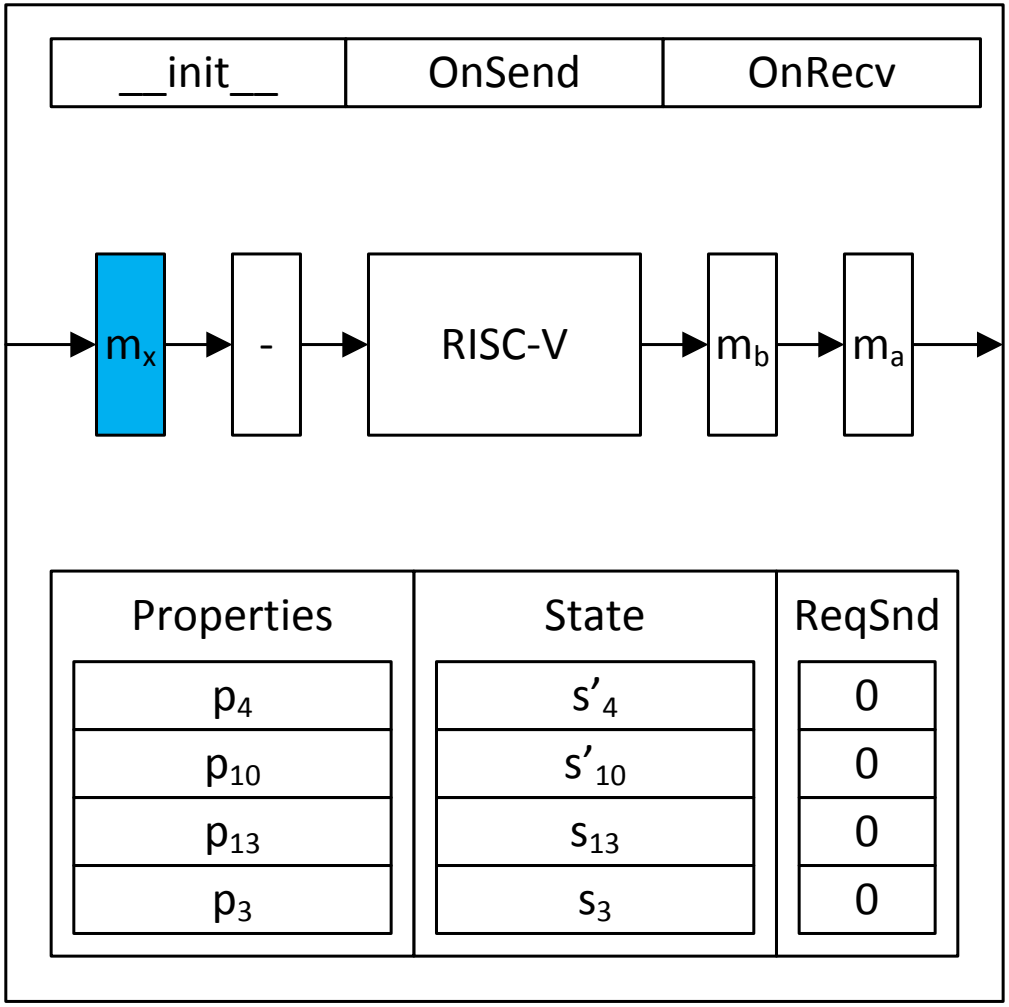

1
3
4
13
10
12
11

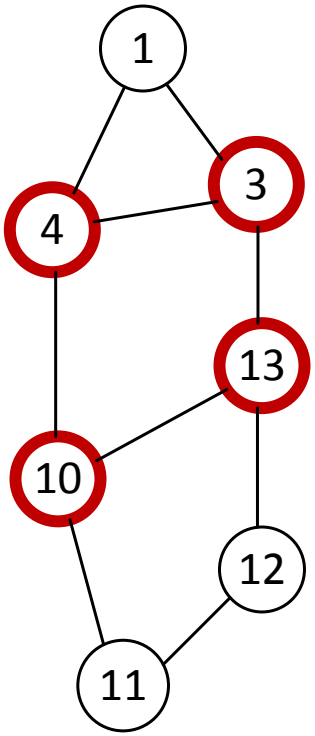

__init__
OnSend
OnRecv
-
-
RISC-V
m_b
m_a
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s_4 / s'_4
s'_{10}
s_{13}
s_3
1 / 0
0
0
0

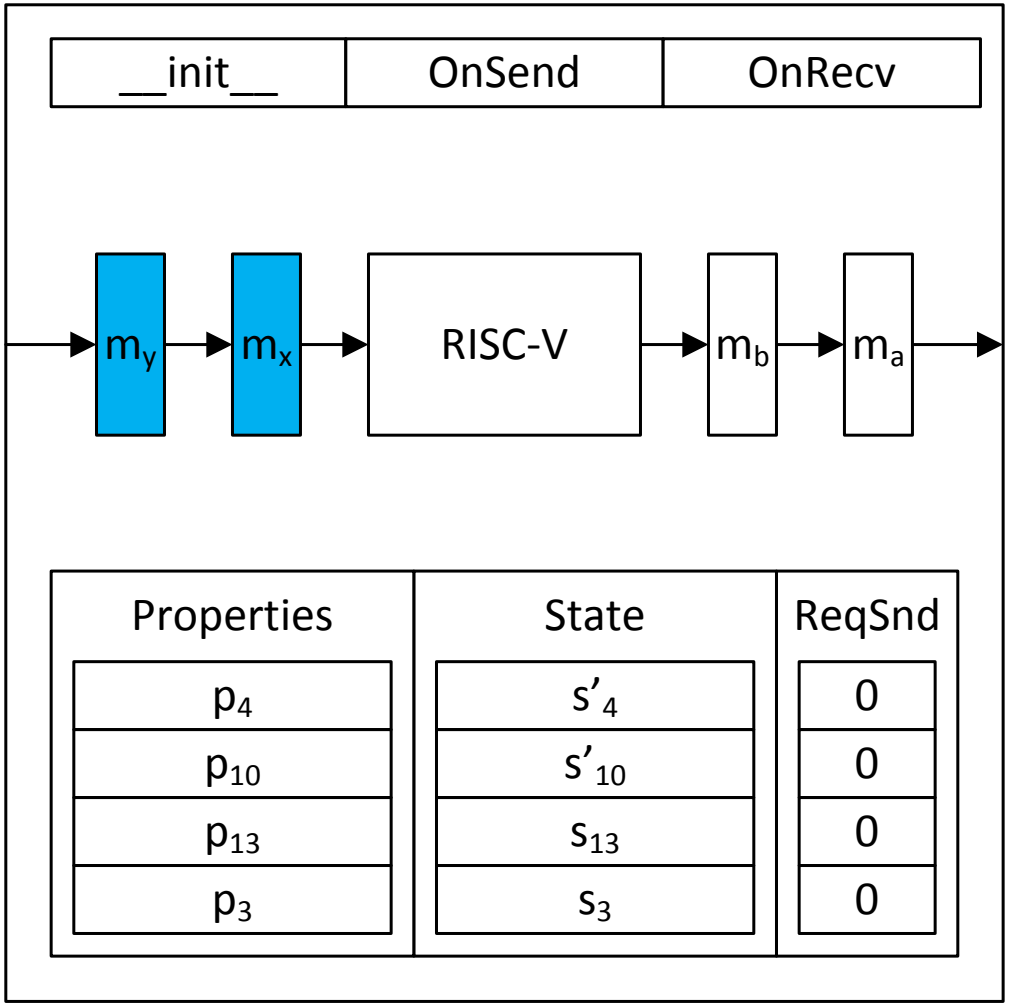

1
3
4
13
10
12
11

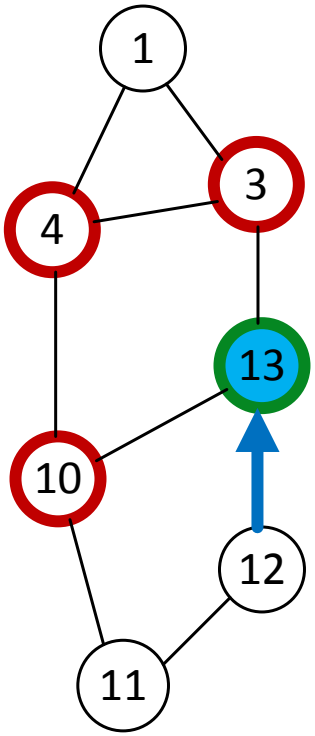

__init__
OnSend
OnRecv
-
-
RISC-V
m_b
m_a
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s'_4
s'_{10}
s_{13}
s_3
0
0
0
0

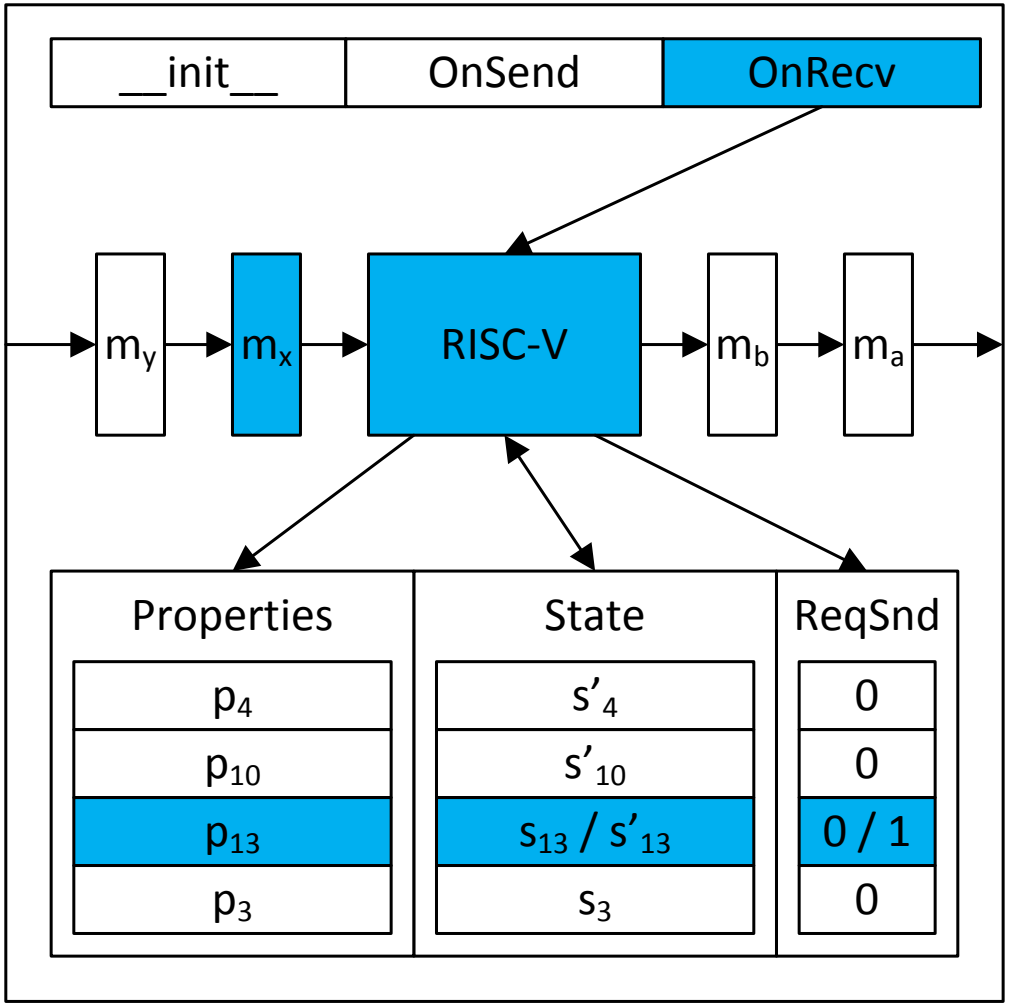

1
3
4
13
10
12
11

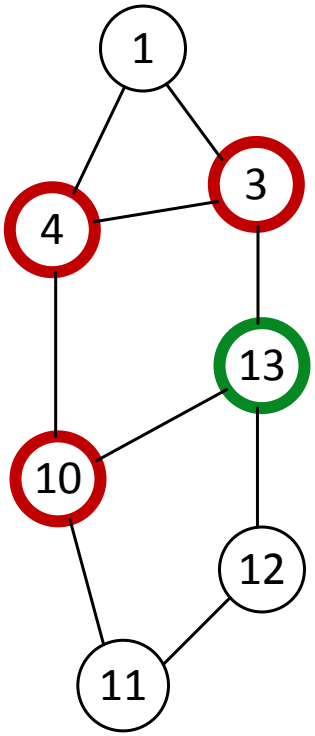

__init__
OnSend
OnRecv
m_x
-
RISC-V
m_b
m_a
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s'_4
s'_{10}
s_{13}
s_3
0
0
0
0

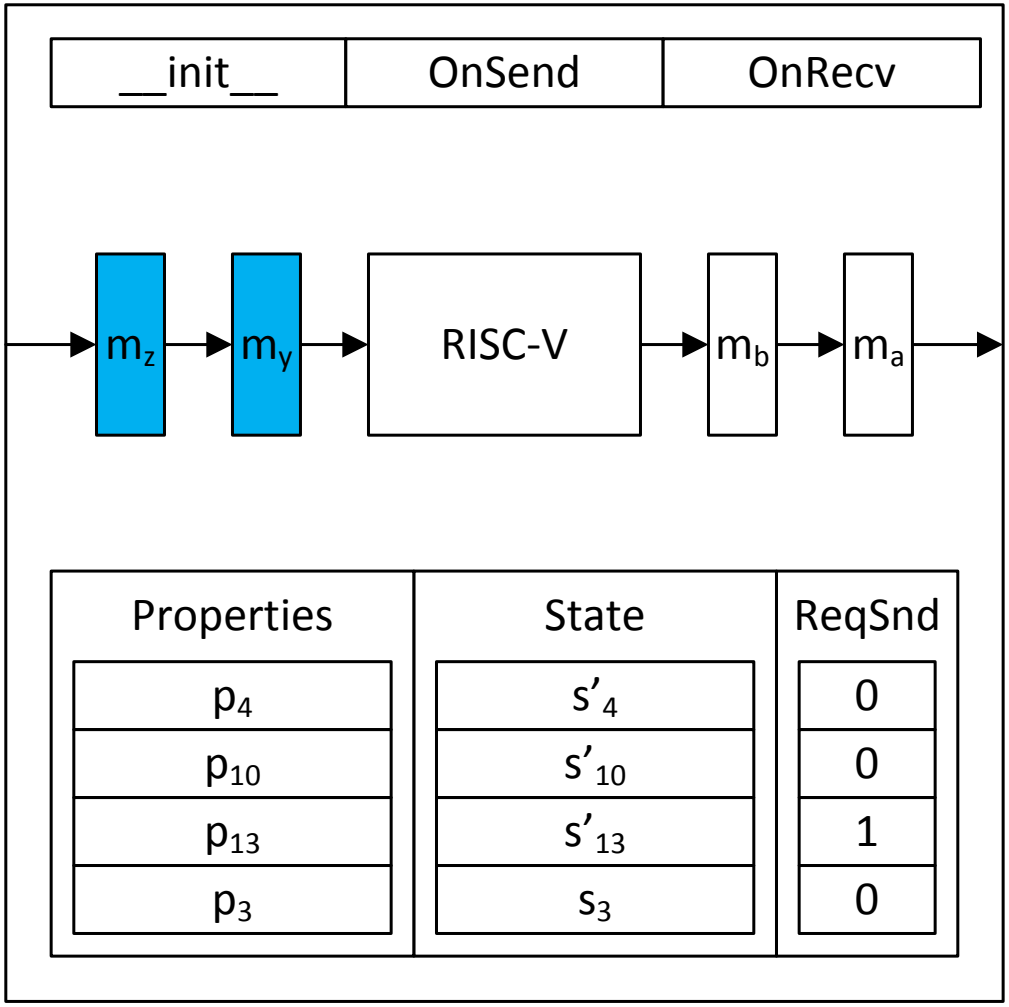

1
3
4
13
10
12
11

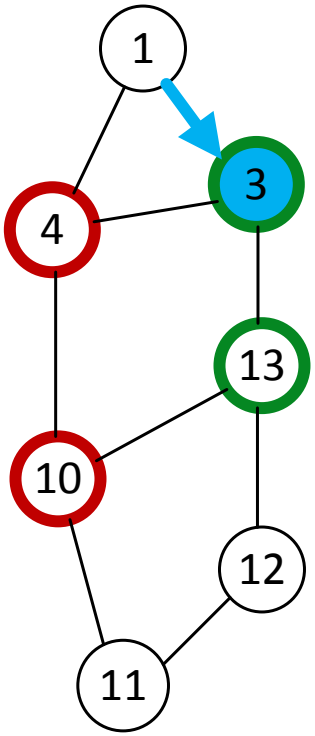

__init__
OnSend
OnRecv
m_y
m_x
RISC-V
m_b
m_a
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s'_4
s'_{10}
s_{13}
s_3
0
0
0
0

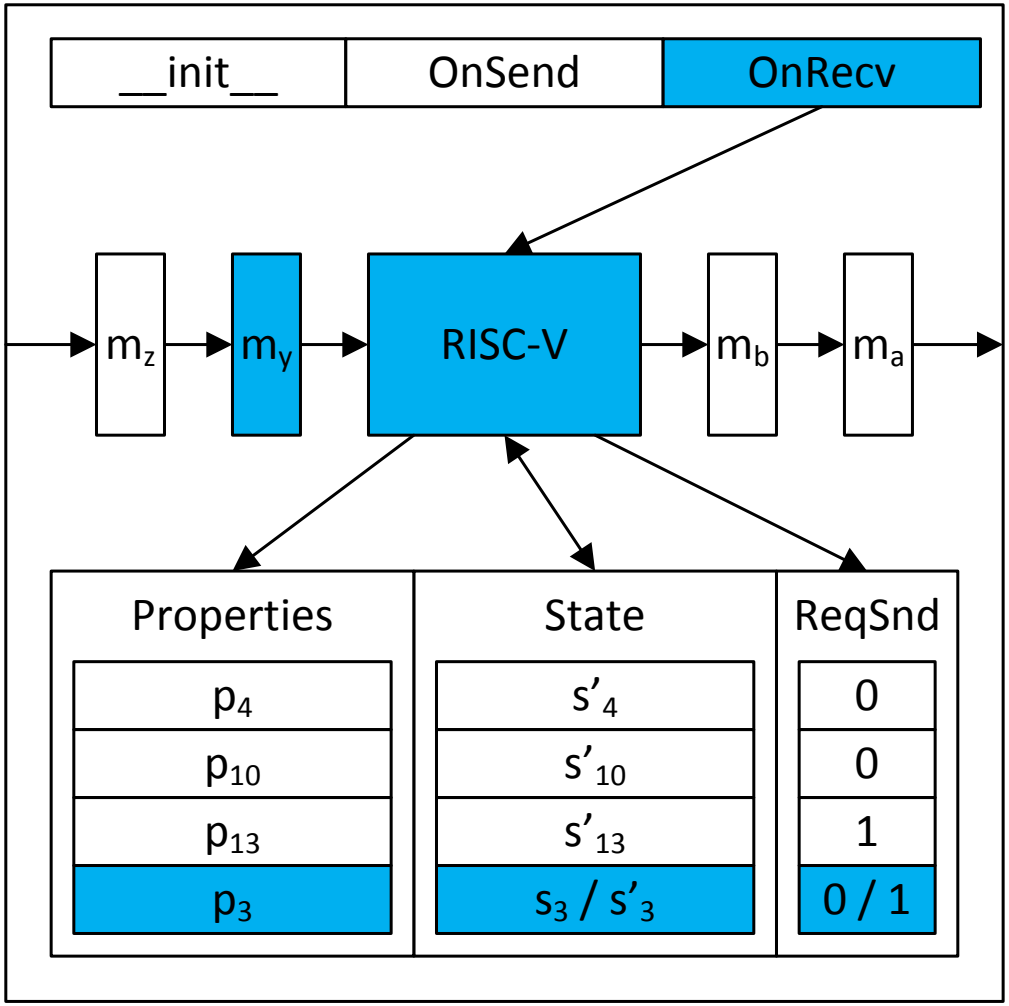

1
3
4
13
10
12
11

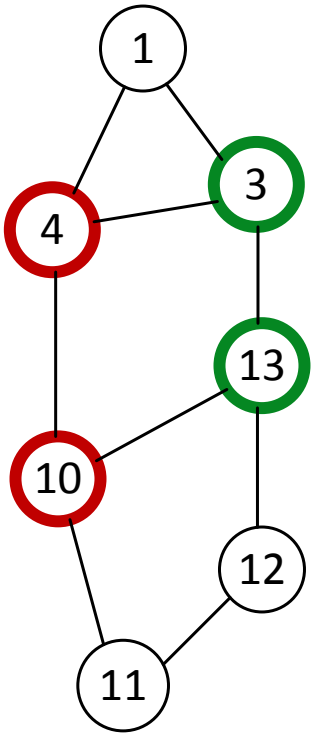

__init__
OnSend
OnRecv
m_y
m_x
RISC-V
m_b
m_a
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s'_4
s'_{10}
s_{13} / s'_{13}
s_3
0
0
0 / 1
0

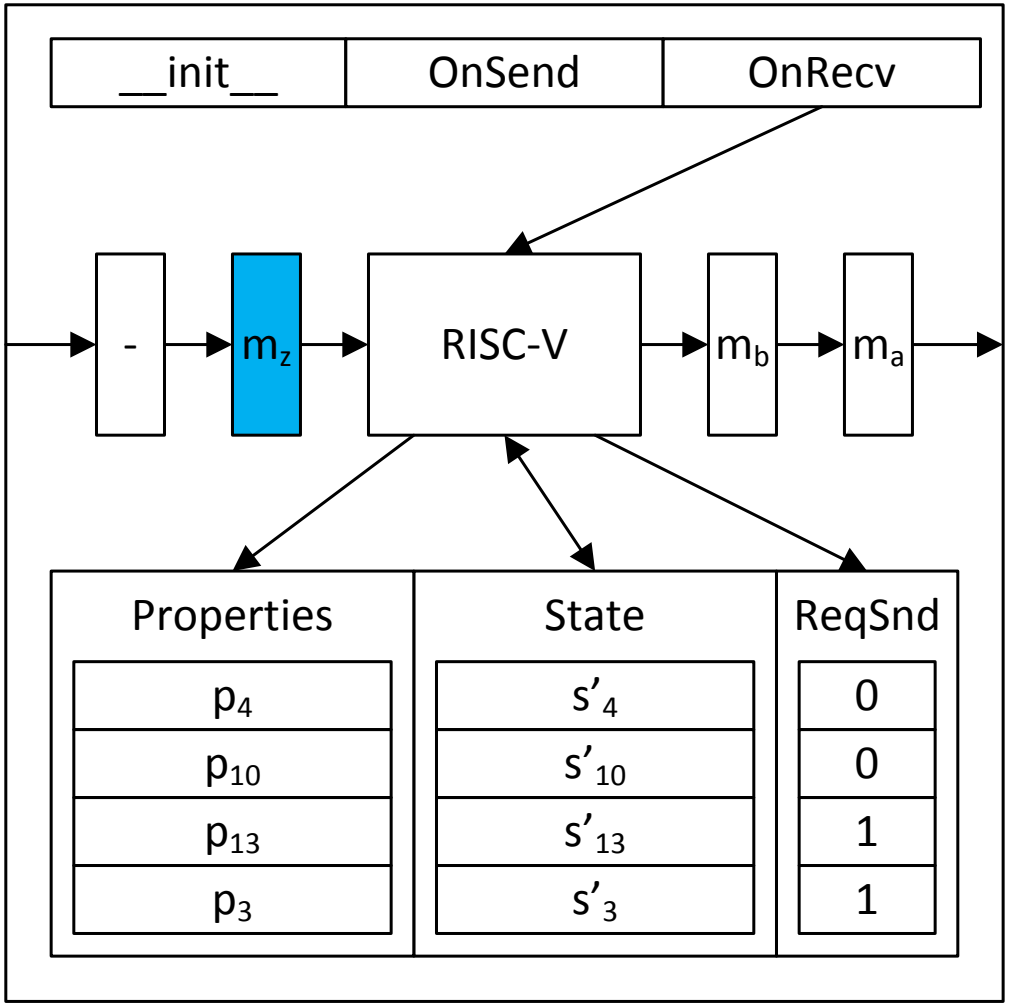

1
3
4
13
10
12
11

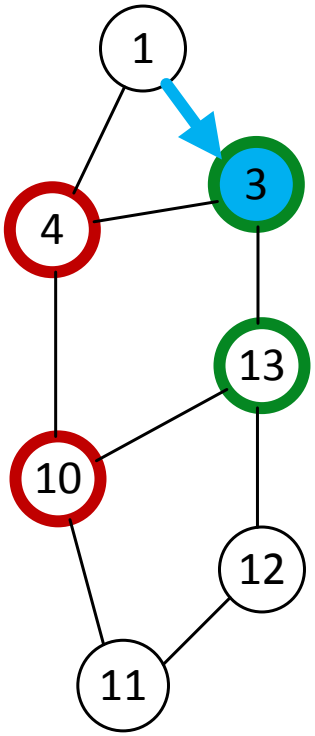

__init__
OnSend
OnRecv
m_z
m_y
RISC-V
m_b
m_a
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s'_4
s'_{10}
s'_{13}
s_3
0
0
1
0

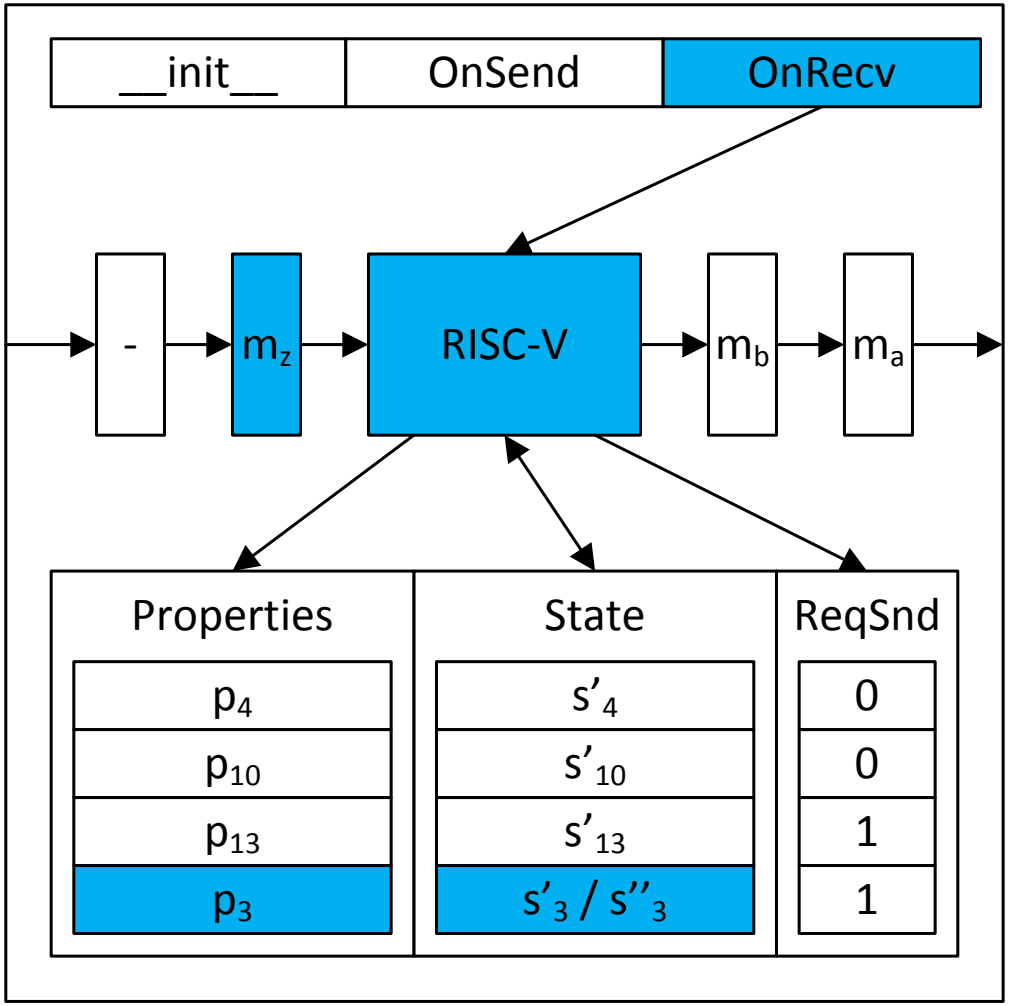

1
3
4
13
10
12
11

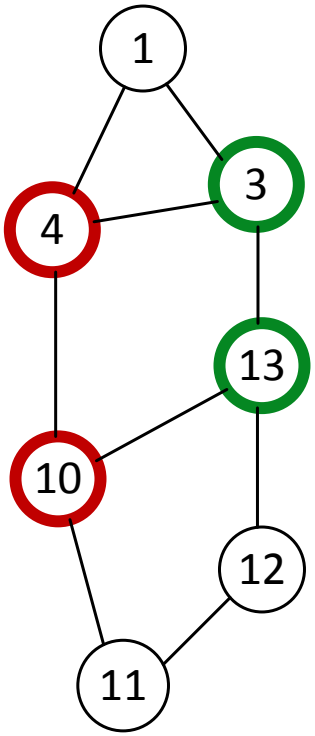

__init__
OnSend
OnRecv
m_z
m_y
RISC-V
m_b
m_a
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s'_4
s'_{10}
s'_{13}
s_3 / s'_3
0
0
1
0 / 1

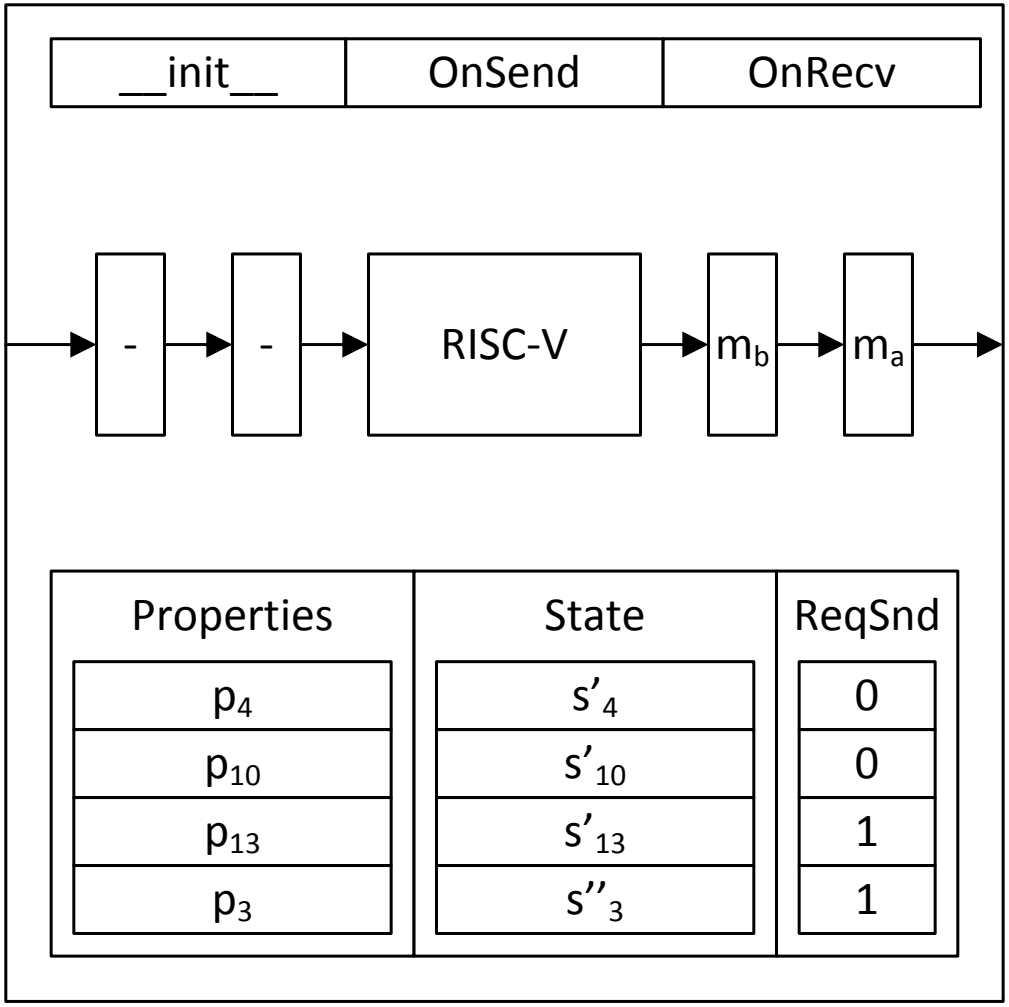

1
3
4
13
10
12
11

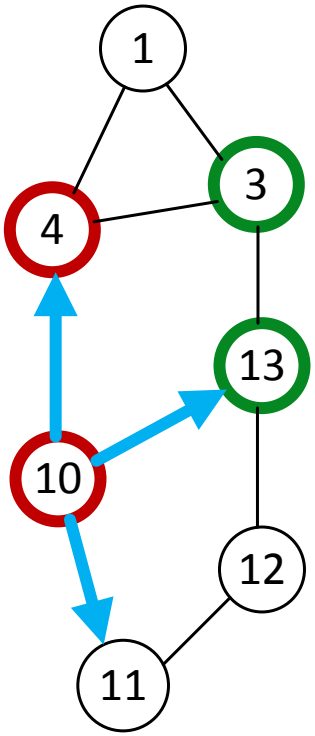

__init__
OnSend
OnRecv
-
m_z
RISC-V
m_b
m_a
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s'_4
s'_{10}
s'_{13}
s'_3
0
0
1
1

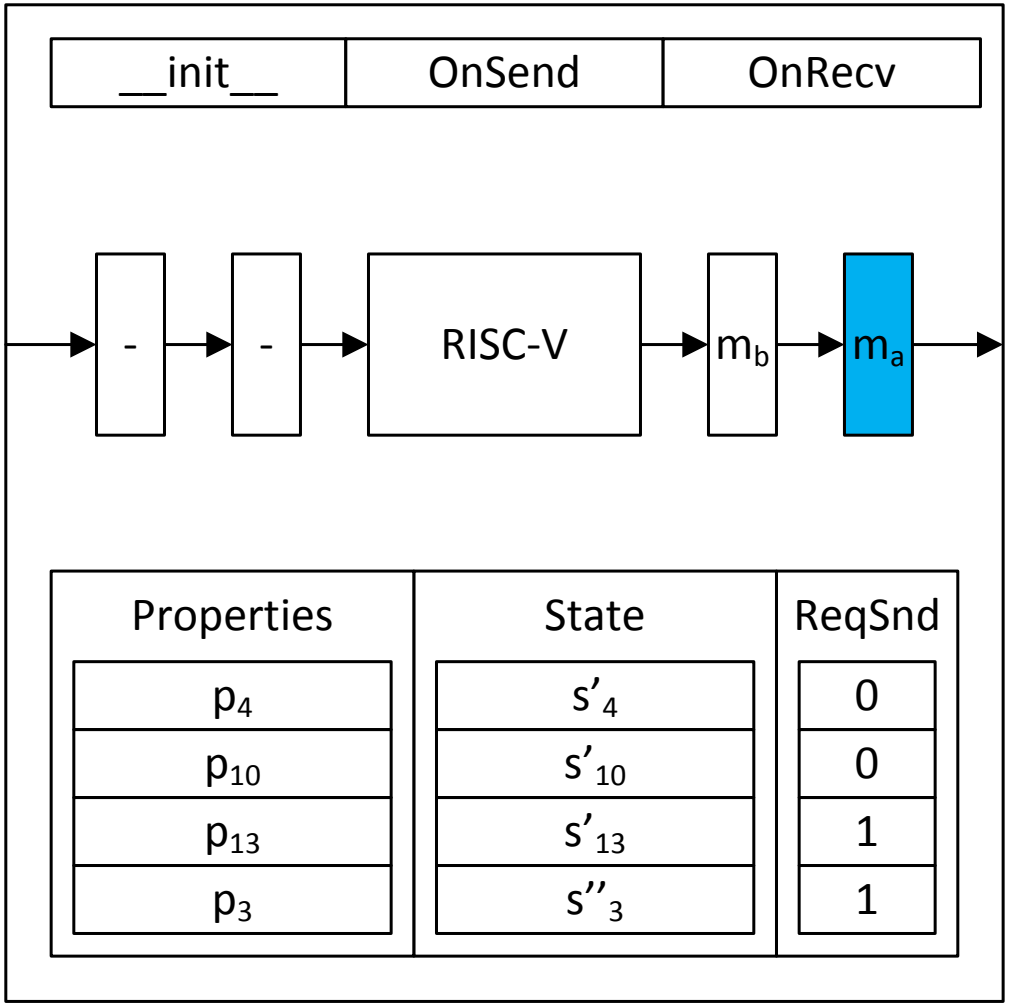

1
3
4
13
10
12
11

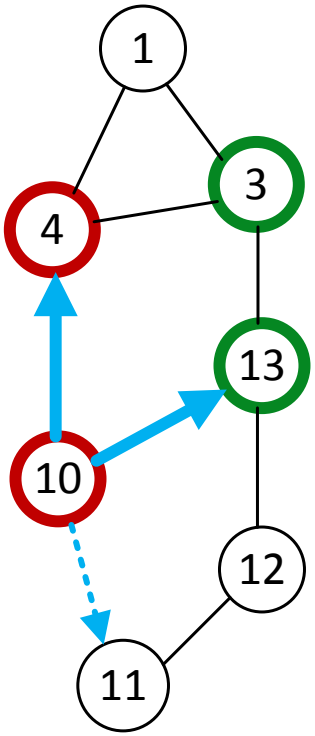

__init__
OnSend
OnRecv
-
m_z
RISC-V
m_b
m_a
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s'_4
s'_{10}
s'_{13}
s'_3 / s''_3
0
0
1
1

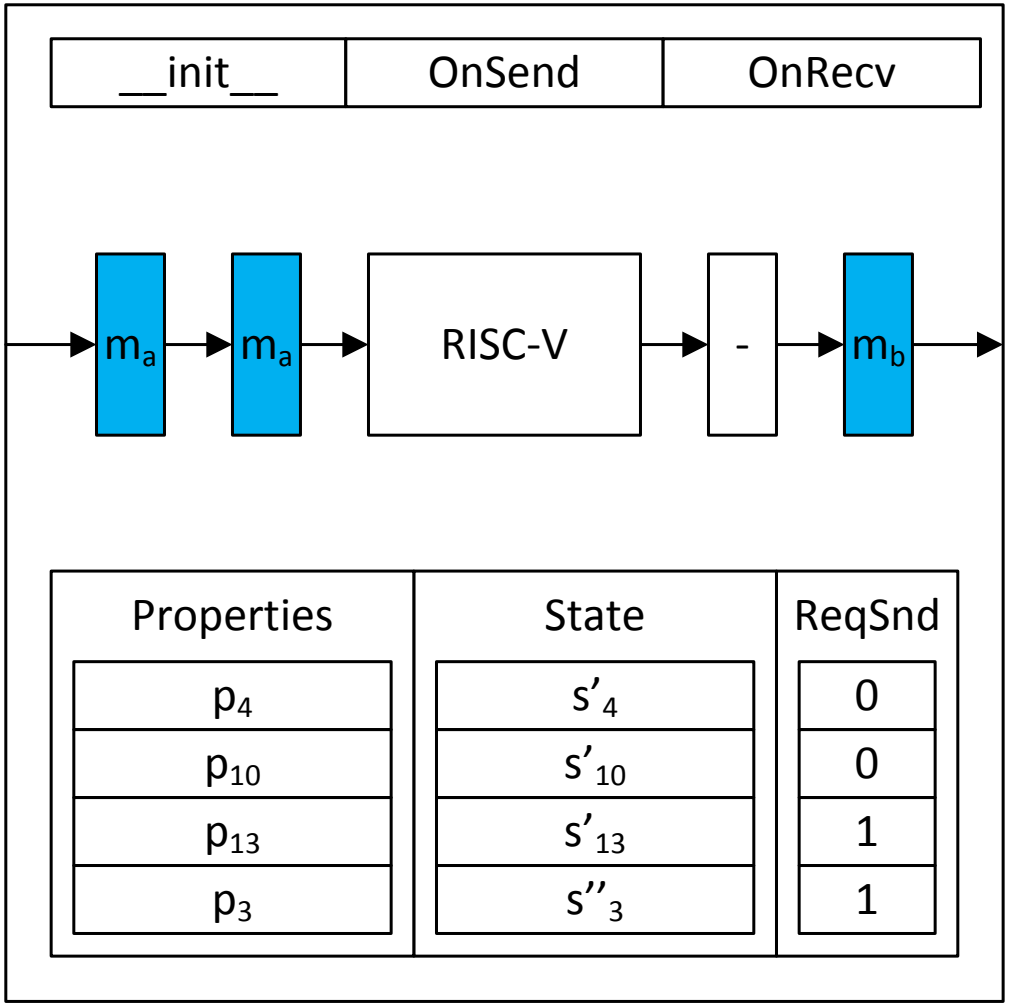

1
3
4
13
10
12
11

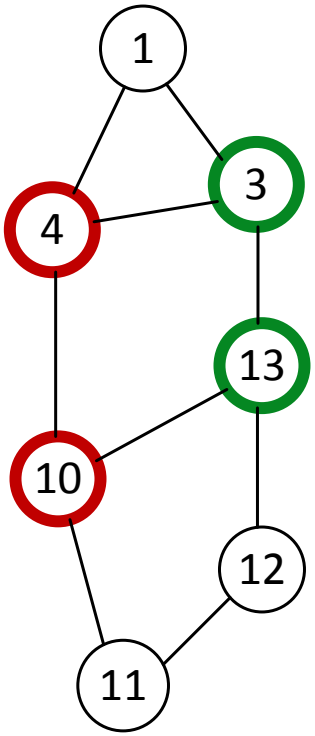

__init__
OnSend
OnRecv
-
-
RISC-V
m_b
m_a
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s'_4
s'_{10}
s'_{13}
s''_3
0
0
1
1

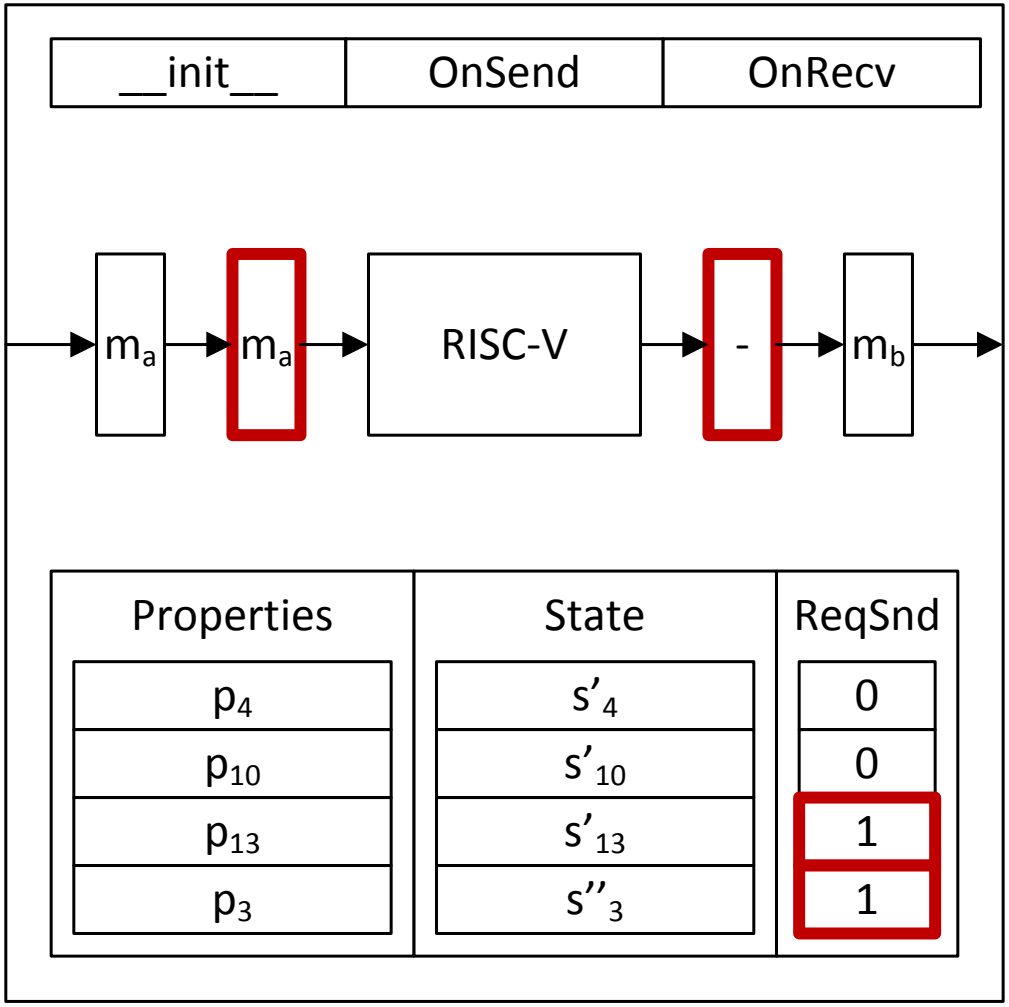

1
3
4
13
10
12
11

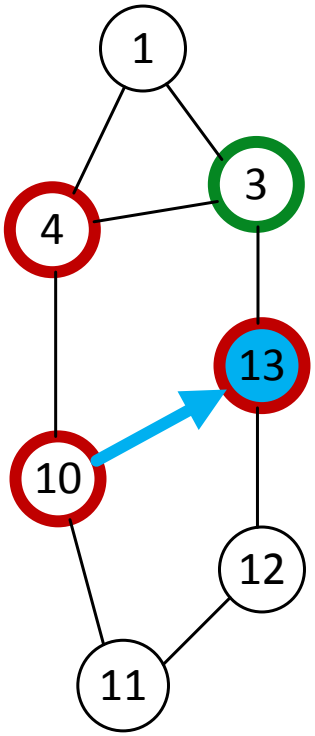

__init__
OnSend
OnRecv
-
-
RISC-V
m_b
m_a
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s'_4
s'_{10}
s'_{13}
s''_3
0
0
1
1

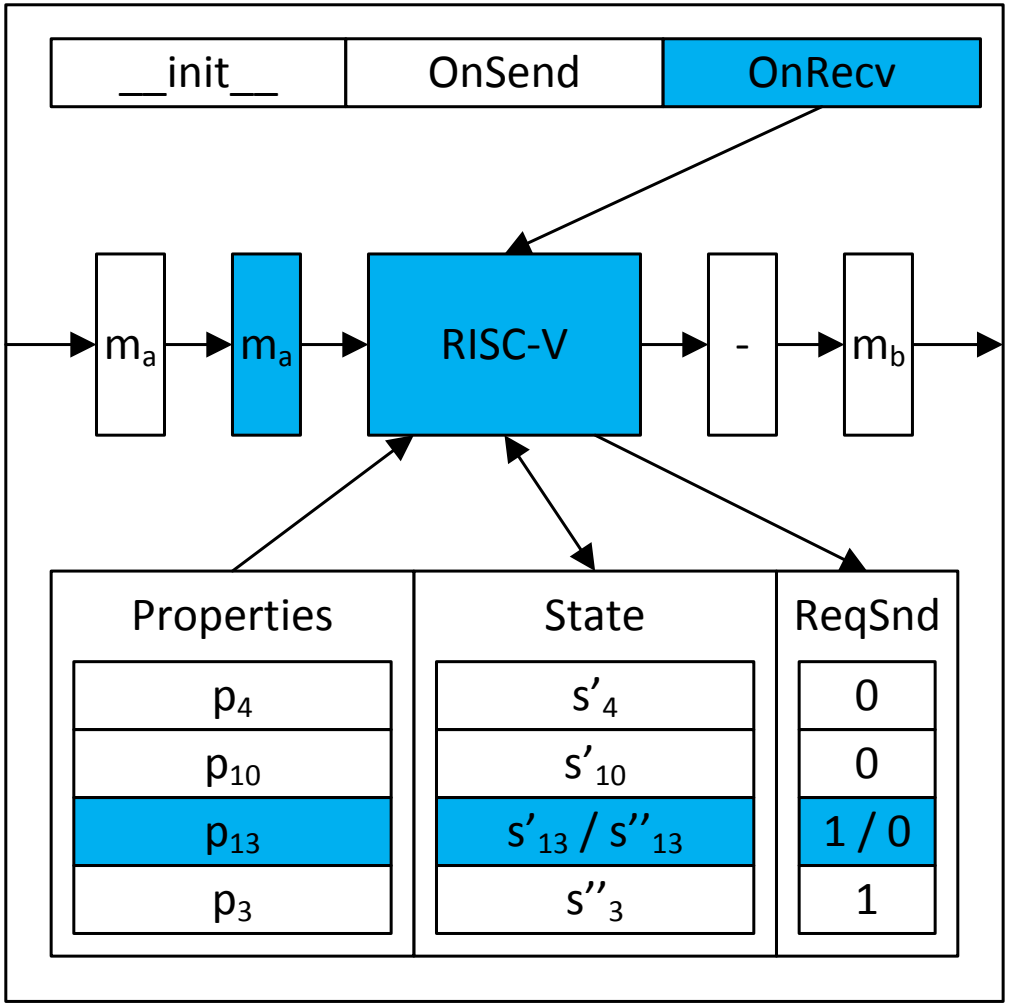

1
3
4
13
10
12
11

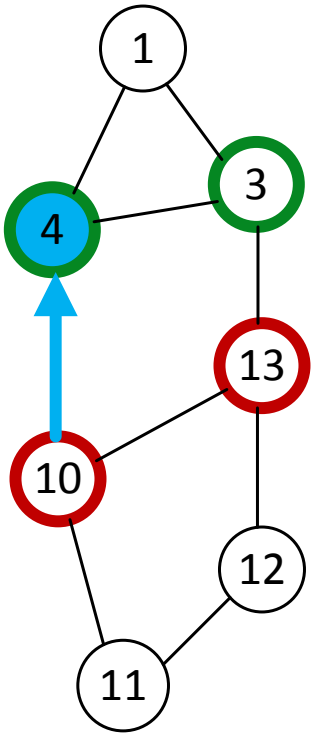

__init__
OnSend
OnRecv
m_a
m_a
RISC-V
-
m_b
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s'_4
s'_{10}
s'_{13}
s''_3
0
0
1
1

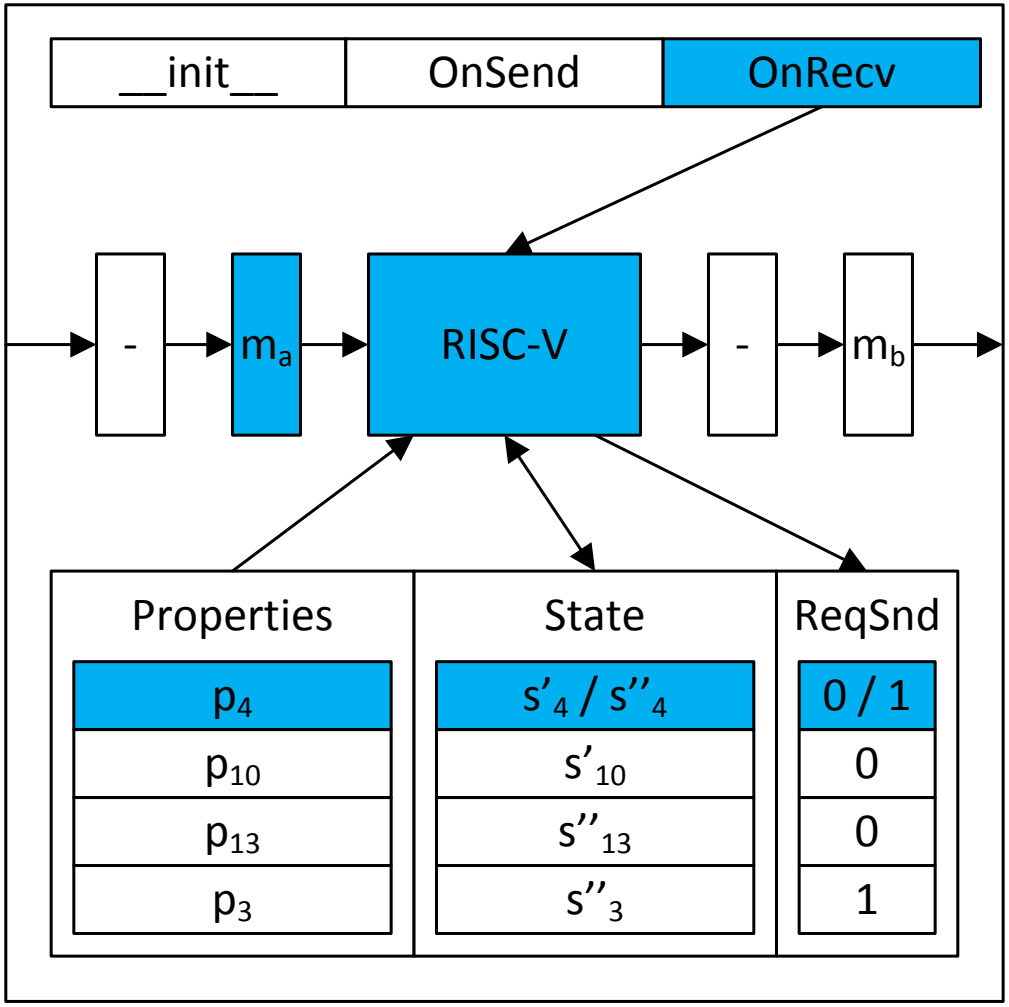

1
3
4
13
10
12
11

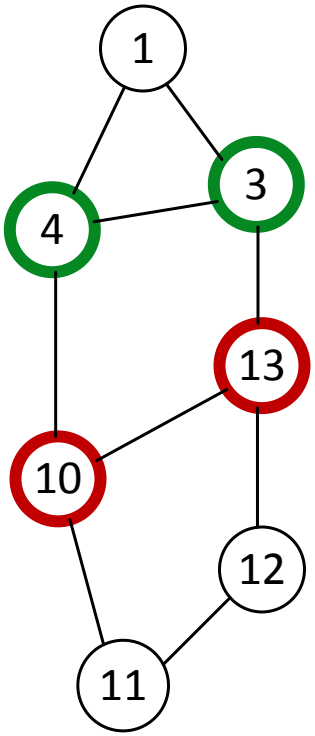

__init__
OnSend
OnRecv
m_a
m_a
RISC-V
-
m_b
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s'_4
s'_{10}
s'_{13}
s''_3
0
0
1
1

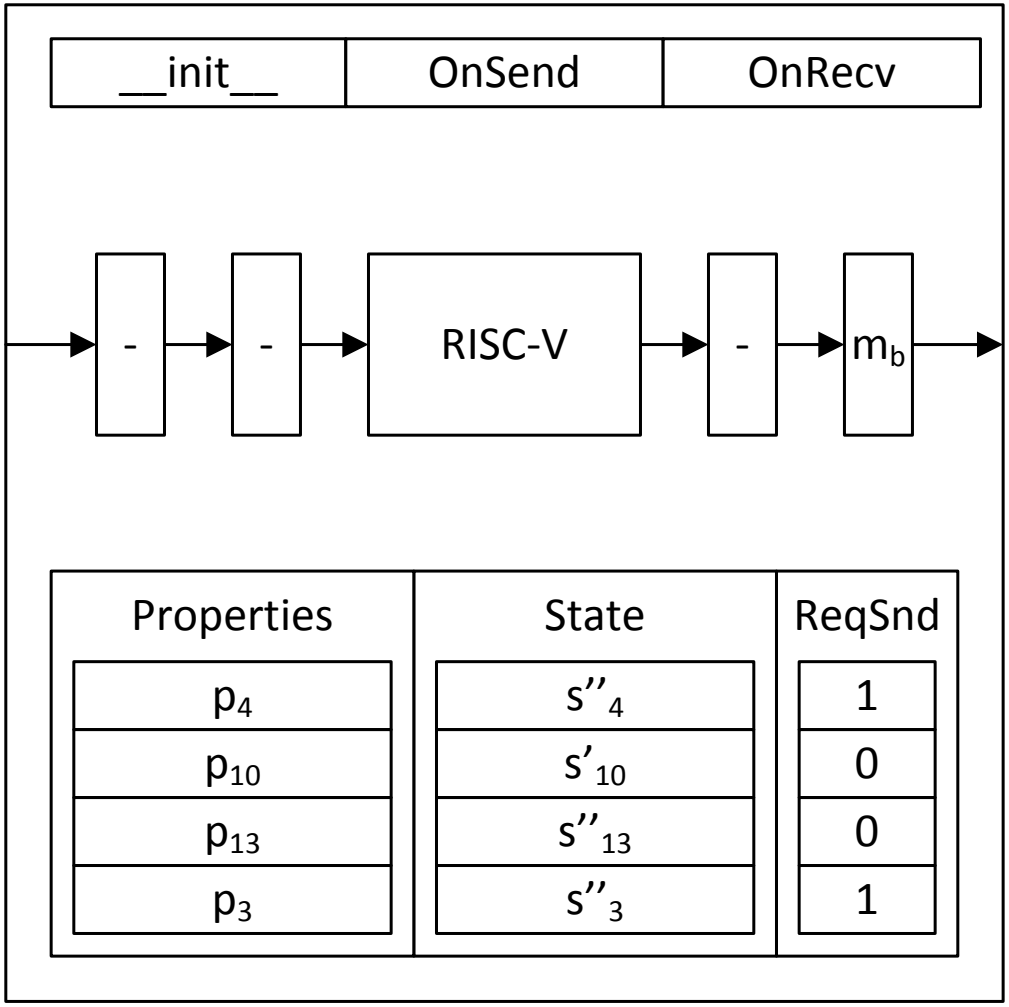

1
3
4
13
10
12
11

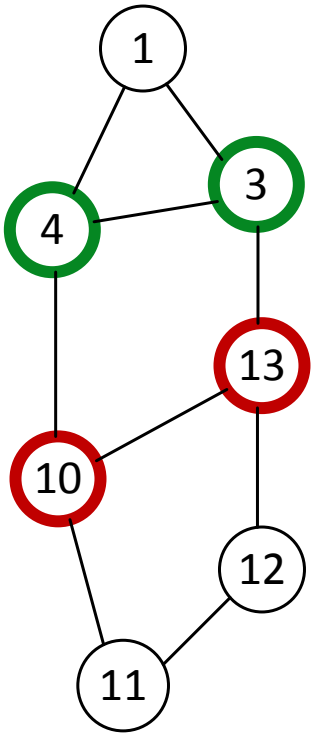

__init__
OnSend
OnRecv
m_a
m_a
RISC-V
-
m_b
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s'_4
s'_{10}
s'_{13} / s''_{13}
s''_3
0
0
1 / 0
1

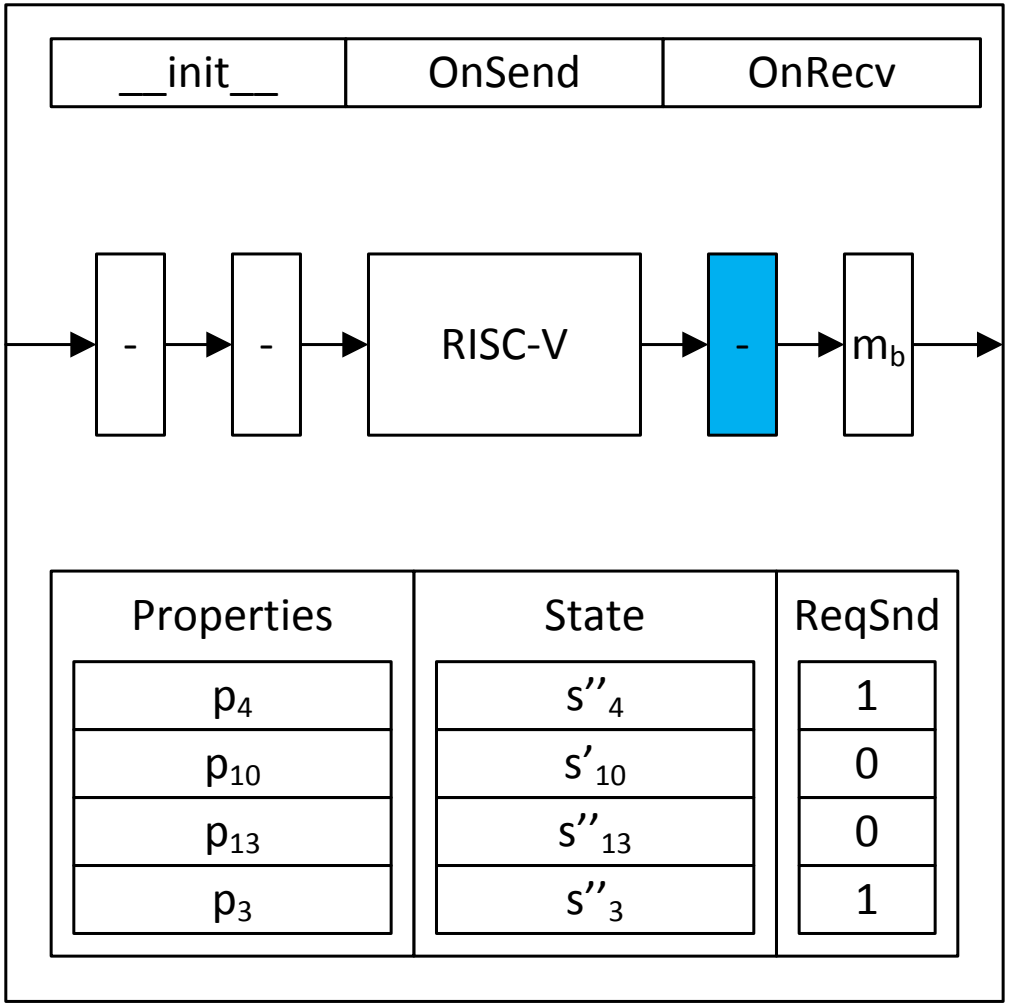

1
3
4
13
10
12
11

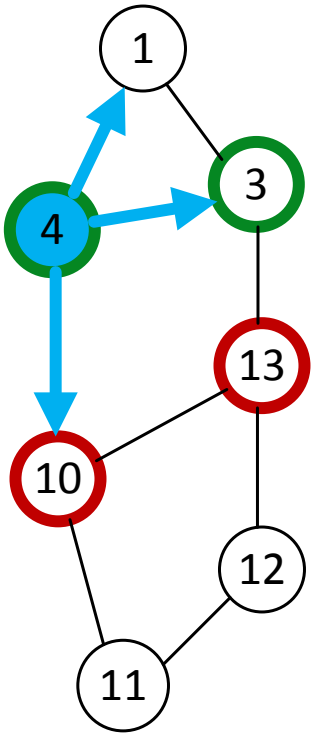

__init__
OnSend
OnRecv
-
m_a
RISC-V
-
m_b
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s'_4 / s''_4
s'_{10}
s''_{13}
s''_3
0 / 1
0
0
1

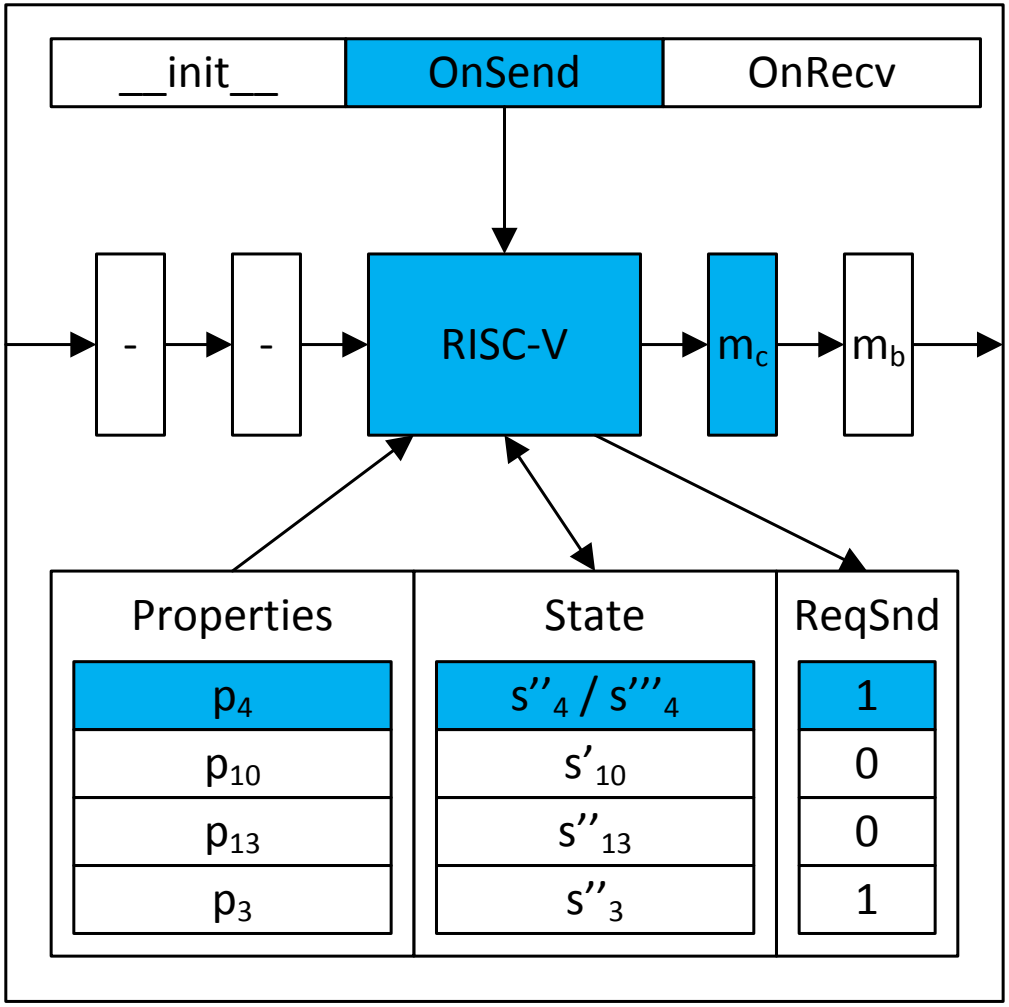

1
3
4
13
10
12
11

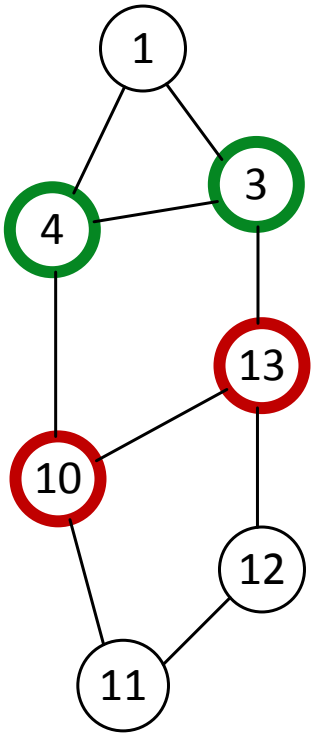

__init__
OnSend
OnRecv
-
-
RISC-V
-
m_b
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s''_4
s'_{10}
s''_{13}
s''_3
1
0
0
1

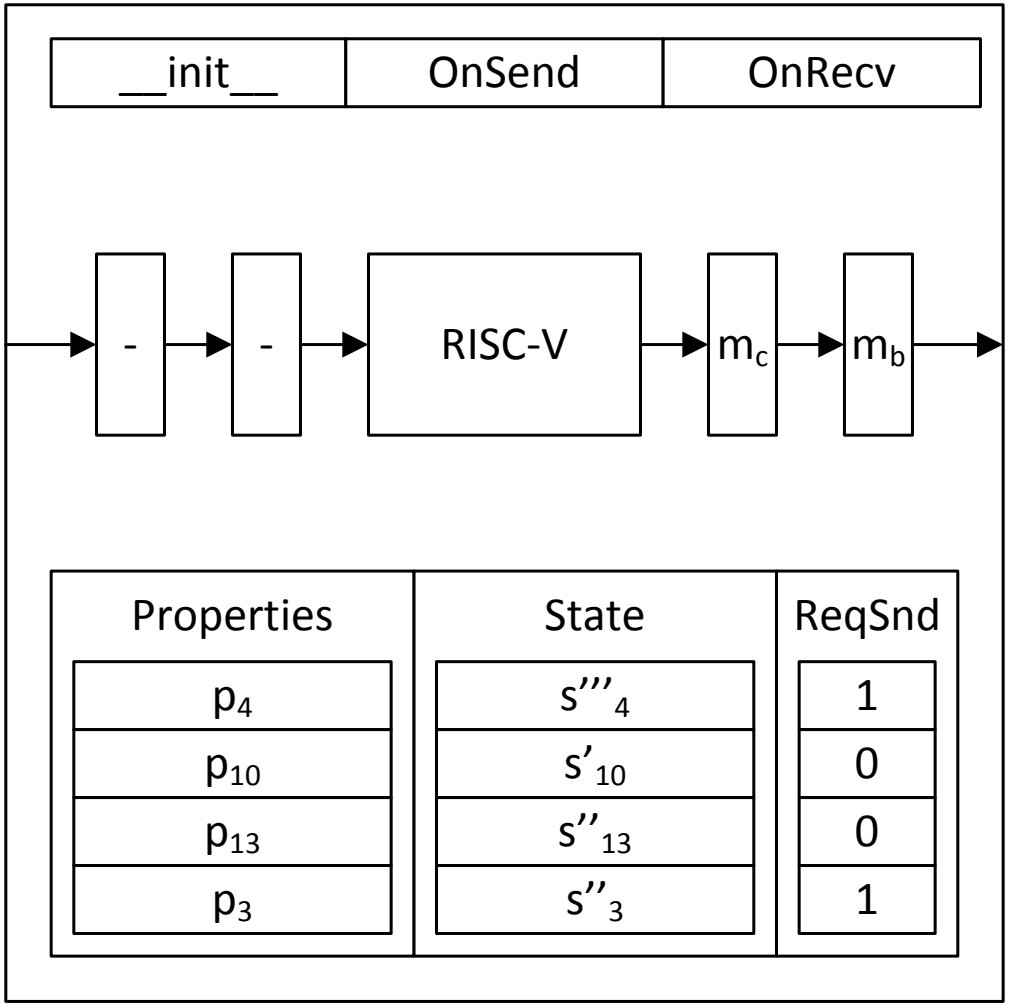

1
3
4
13
10
12
11

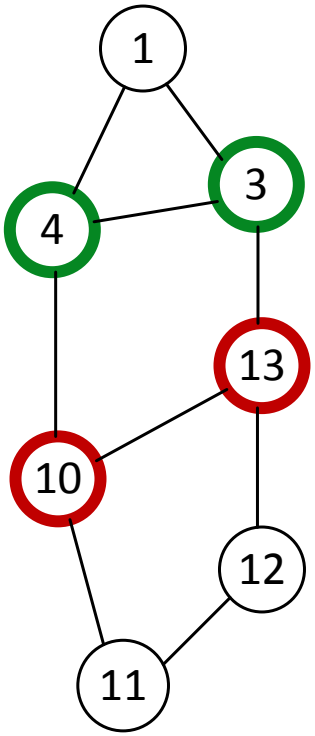

__init__
OnSend
OnRecv
-
-
RISC-V
-
m_b
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s''_4
s'_{10}
s''_{13}
s''_3
1
0
0
1

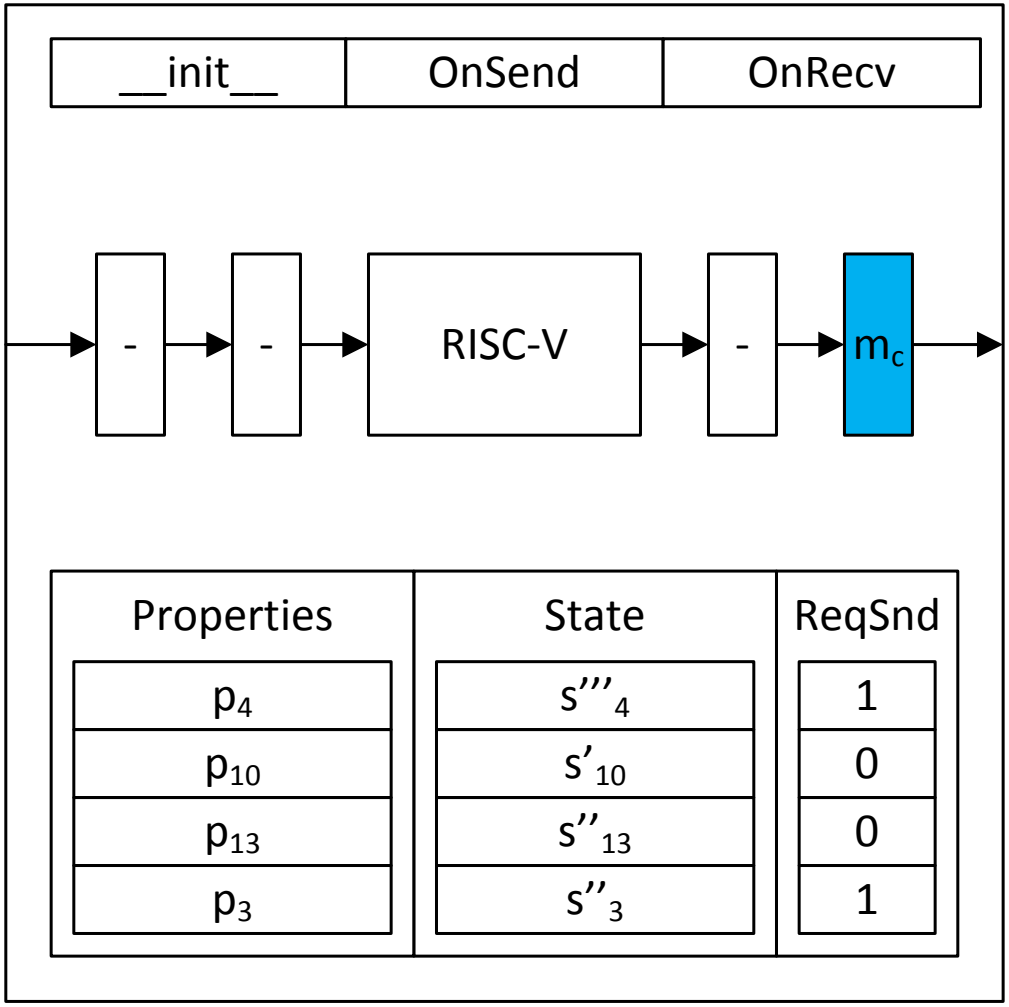

1
3
4
13
10
12
11

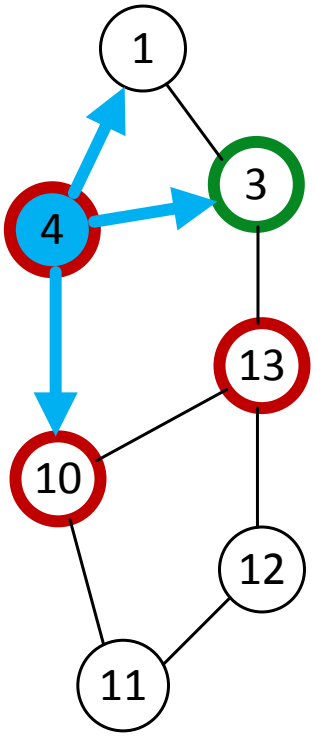

__init__
OnSend
OnRecv
-
-
RISC-V
m_c
m_b
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s''_4 / s'''_4
s'_{10}
s''_{13}
s''_3
1
0
0
1

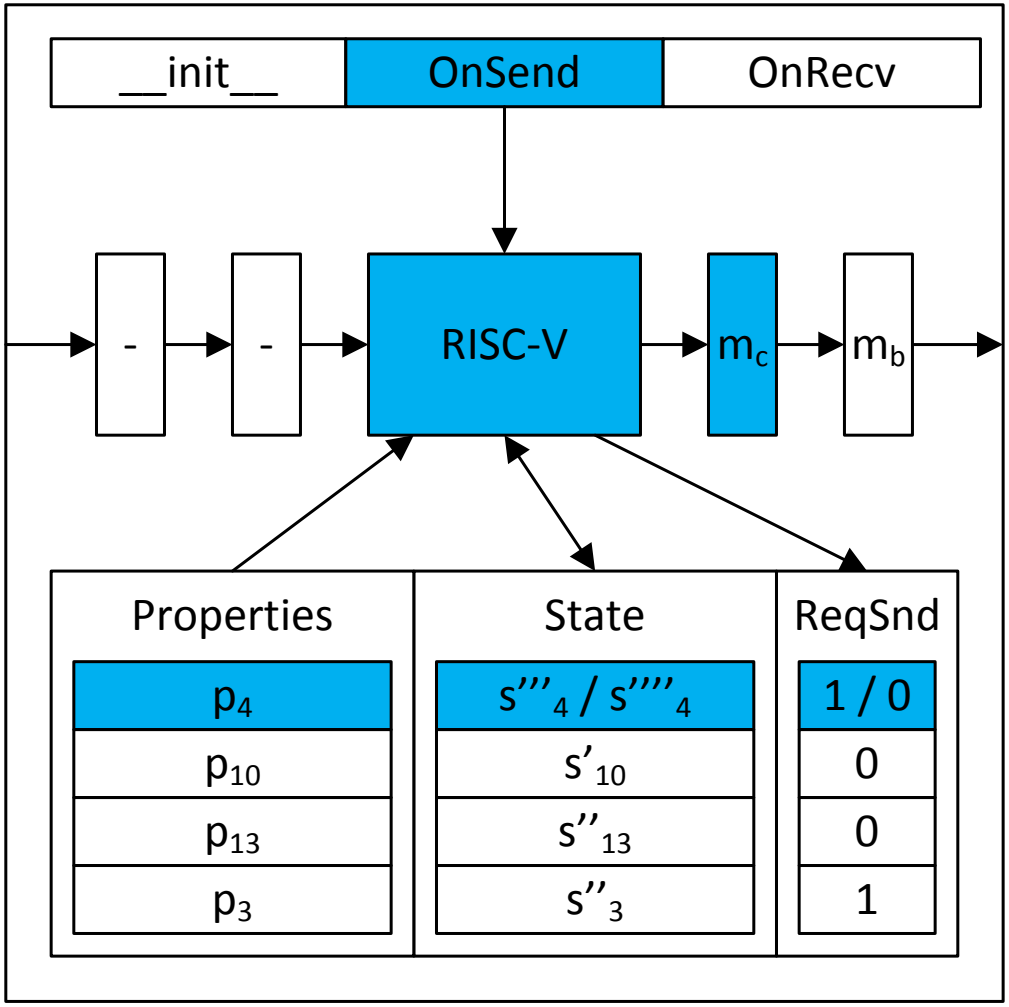

1
3
4
13
10
12
11

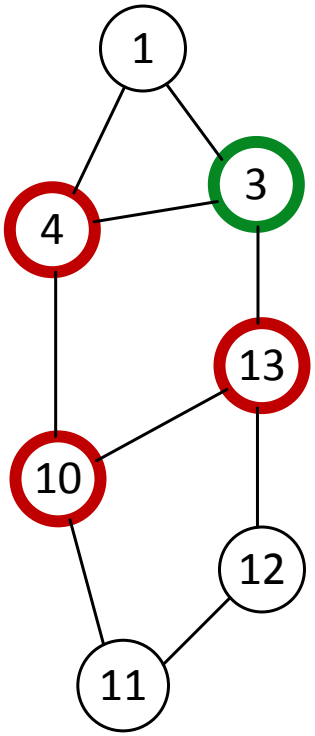

__init__
OnSend
OnRecv
-
-
RISC-V
m_c
m_b
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s'''_4
s'_{10}
s''_{13}
s''_3
1
0
0
1

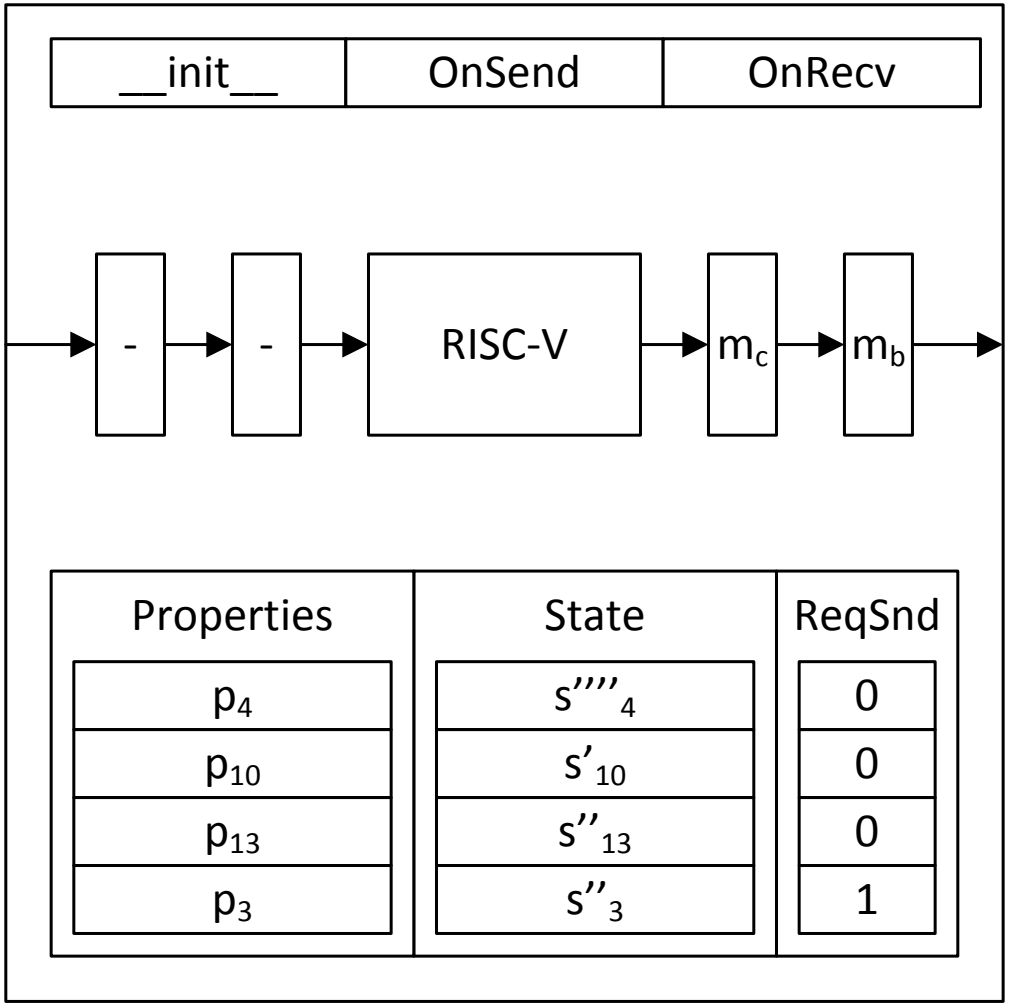

1
3
4
13
10
12
11

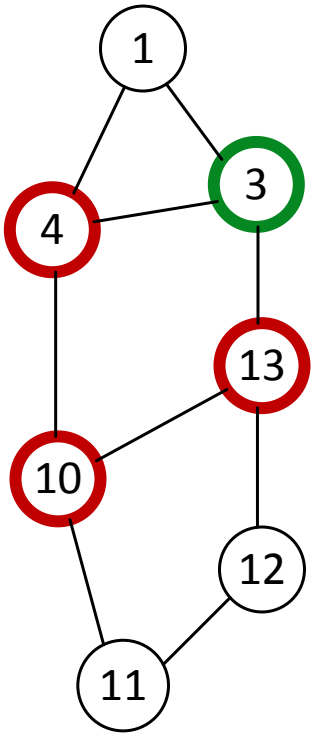

__init__
OnSend
OnRecv
-
-
RISC-V
-
m_c
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s'''_4
s'_{10}
s''_{13}
s''_3
1
0
0
1

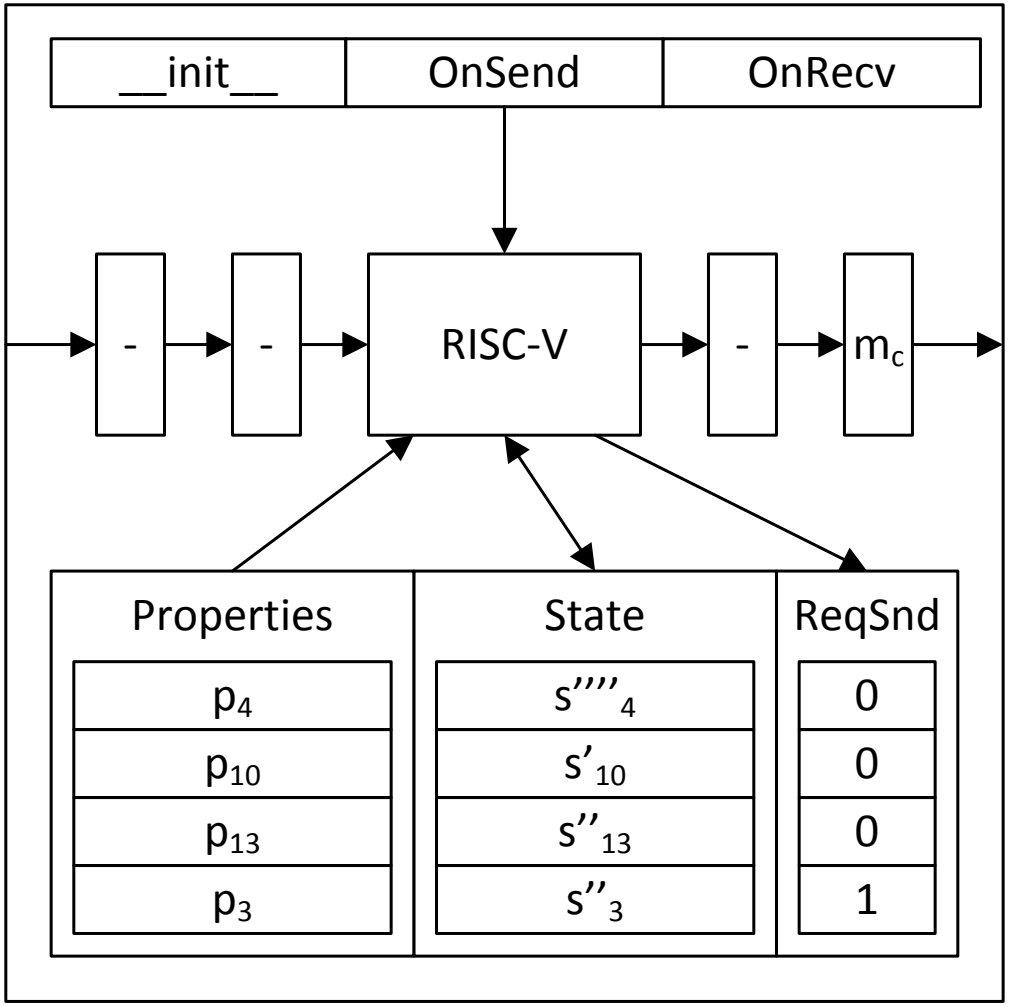

1
3
4
13
10
12
11

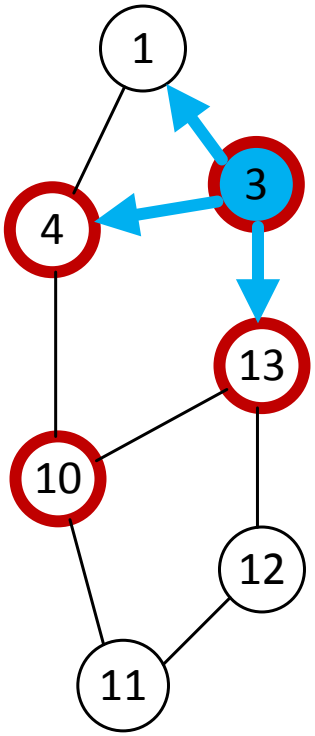

__init__
OnSend
OnRecv
-
-
RISC-V
m_c
m_b
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s'''_4 / s''''_4
s'_{10}
s''_{13}
s''_3
1 / 0
0
0
1

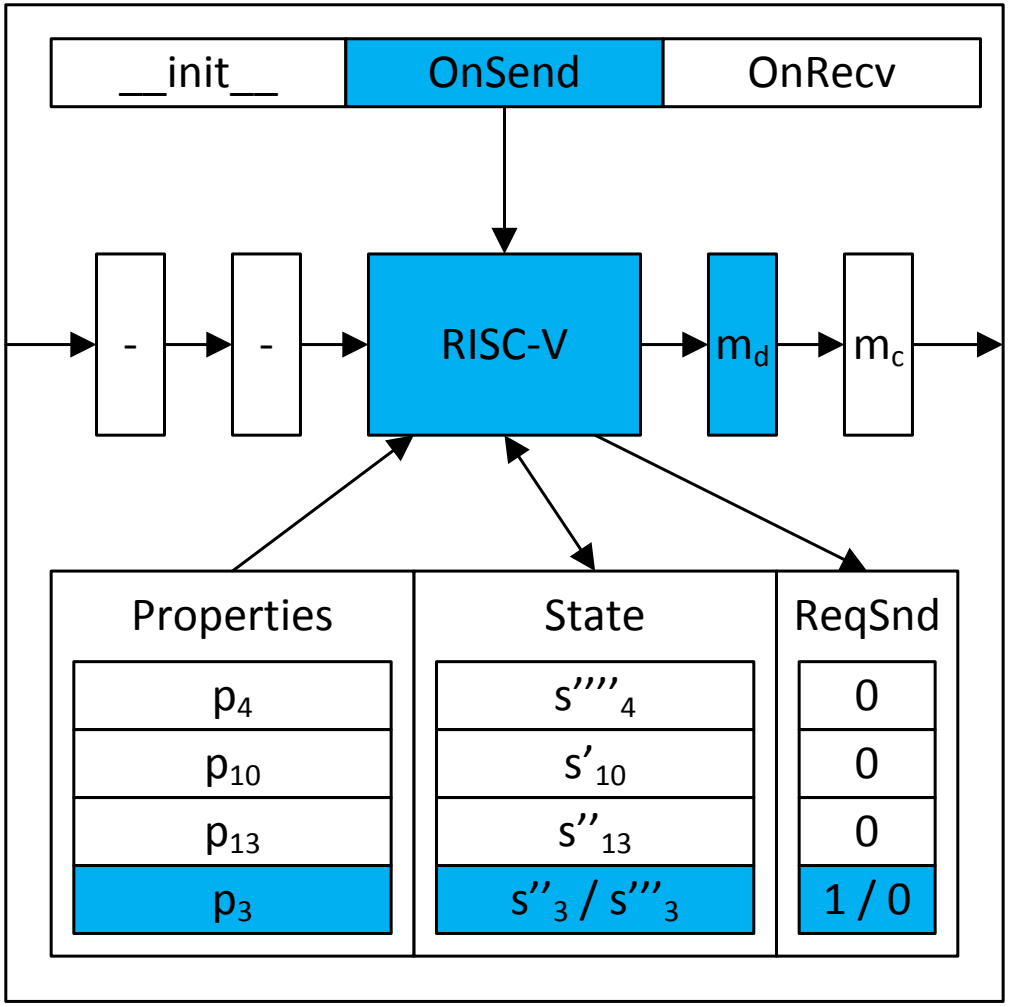

1
3
4
13
10
12
11

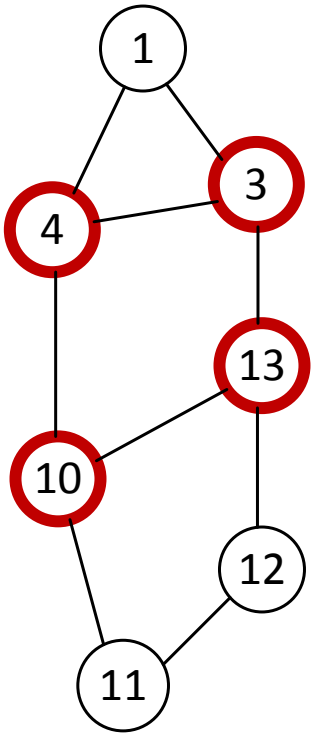

__init__
OnSend
OnRecv
-
-
RISC-V
m_c
m_b
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s''''_4
s'_{10}
s''_{13}
s''_3
0
0
0
1

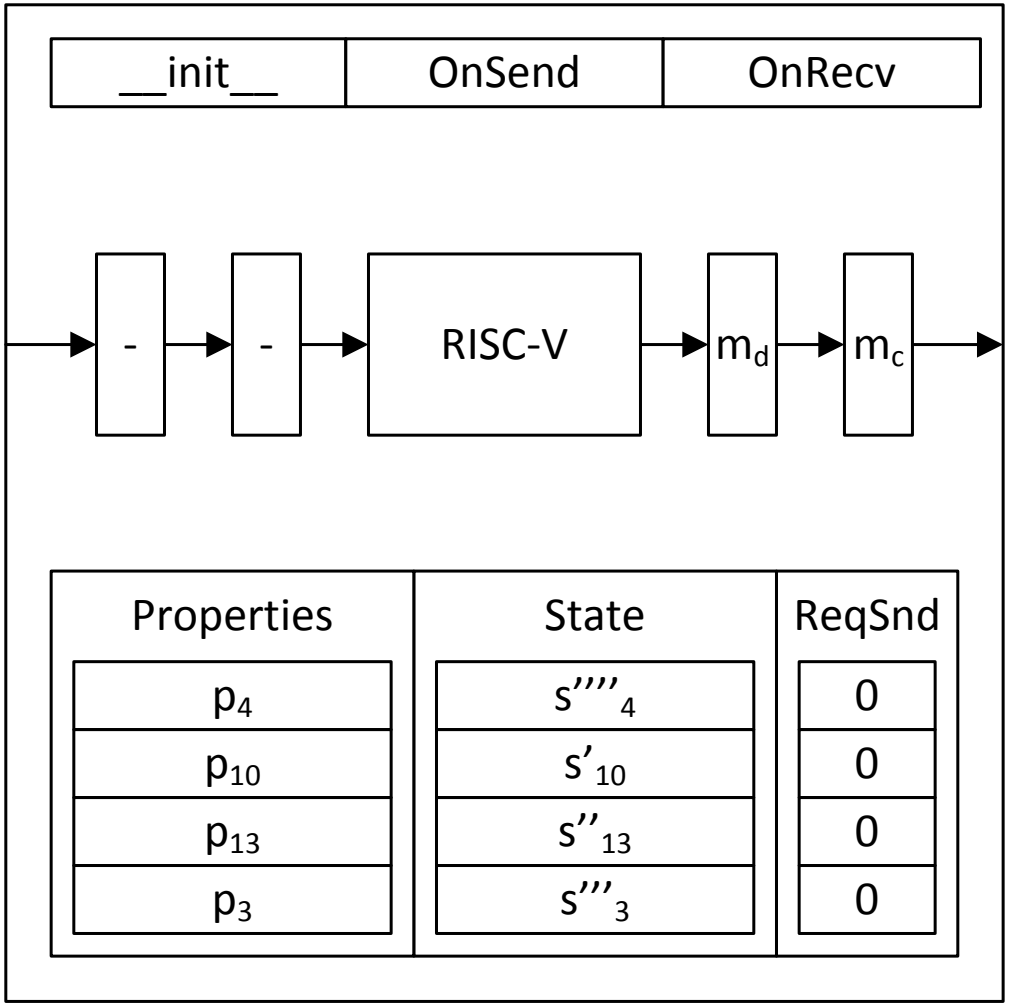

1
3
4
13
10
12
11


__init__
OnSend
OnRecv
-
-
RISC-V
-
m_c
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s''''_4
s'_{10}
s''_{13}
s''_3
0
0
0
1

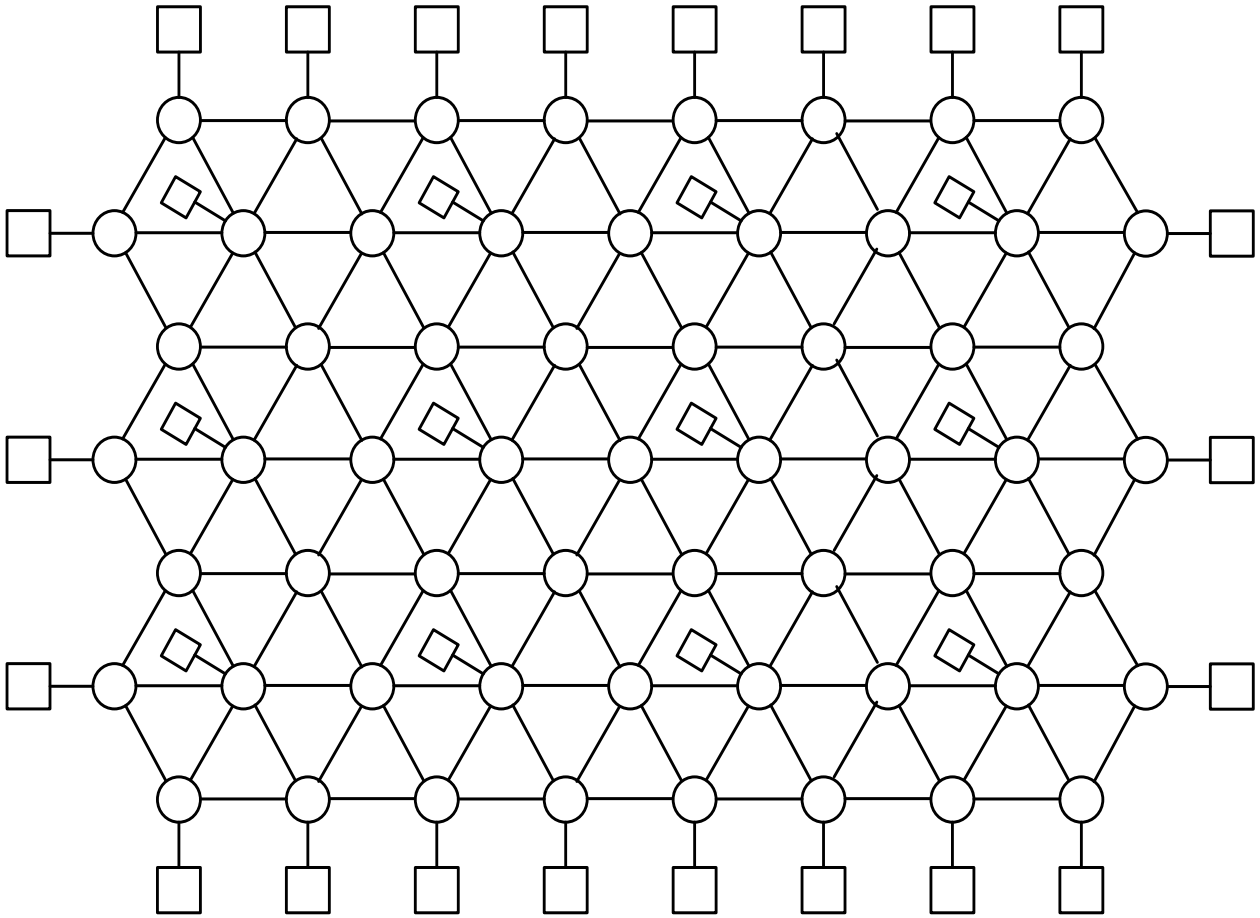

1
3
4
13
10
12
11

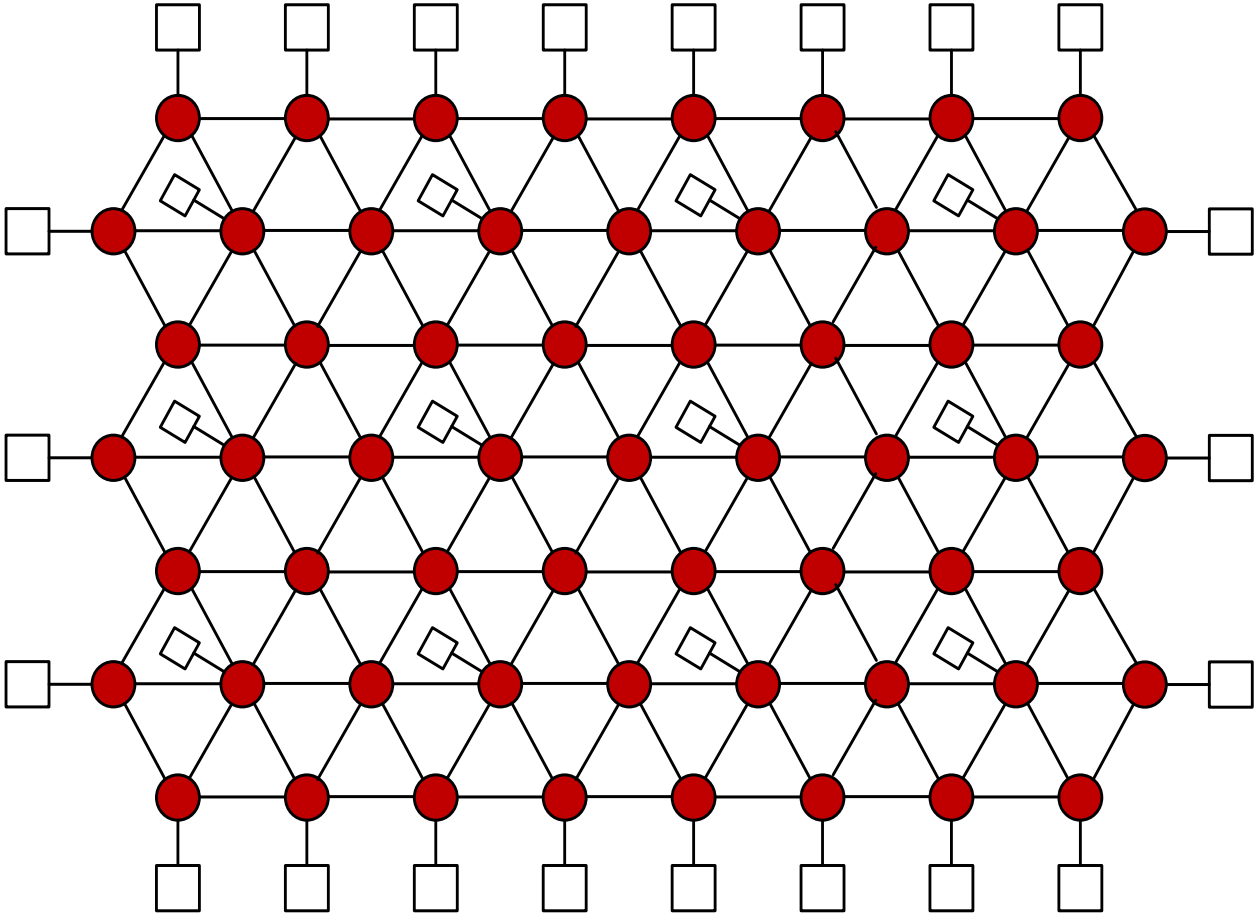

__init__
OnSend
OnRecv
-
-
RISC-V
m_d
m_c
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s''''_4
s'_{10}
s''_{13}
s''_3 / s'''_3
0
0
0
1 / 0

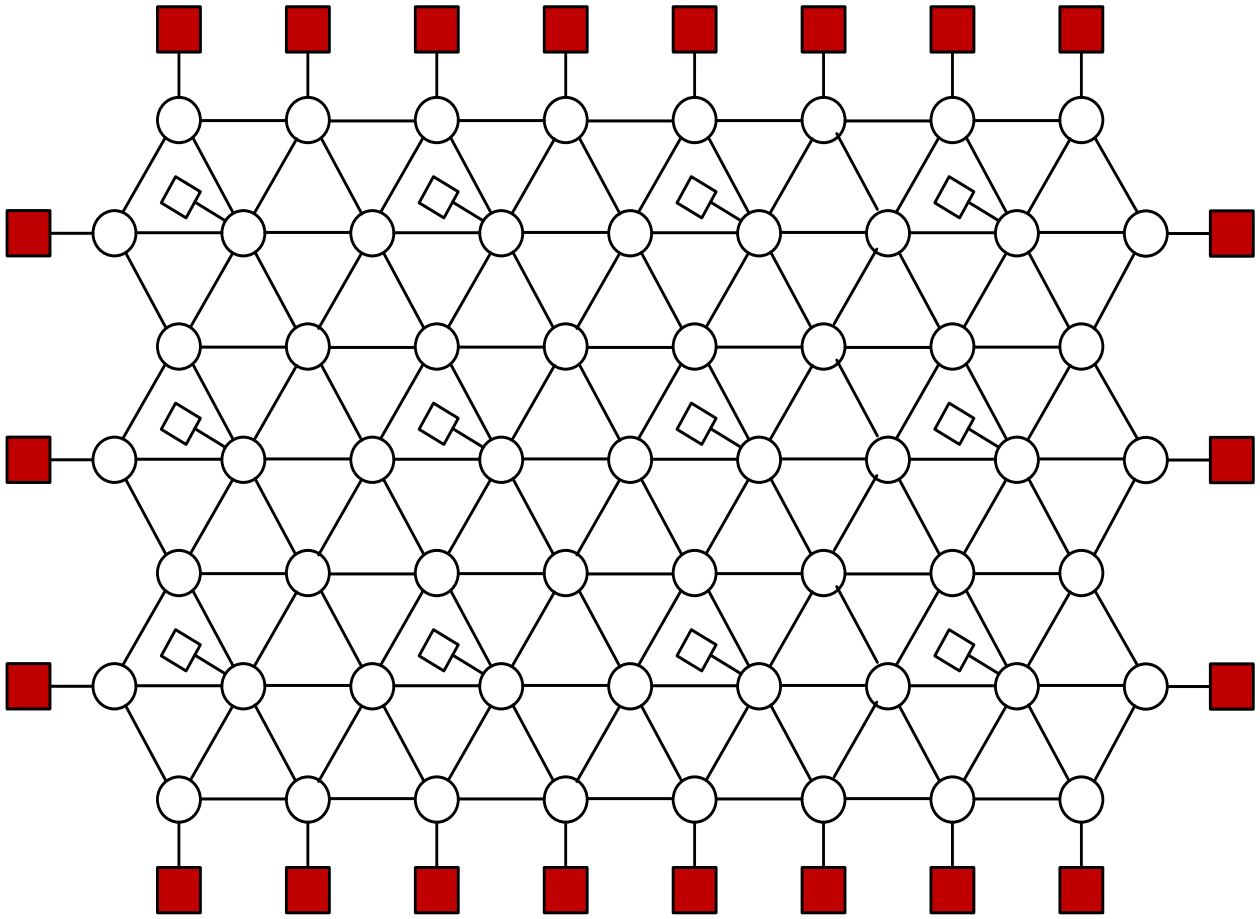

1
3
4
13
10
12
11

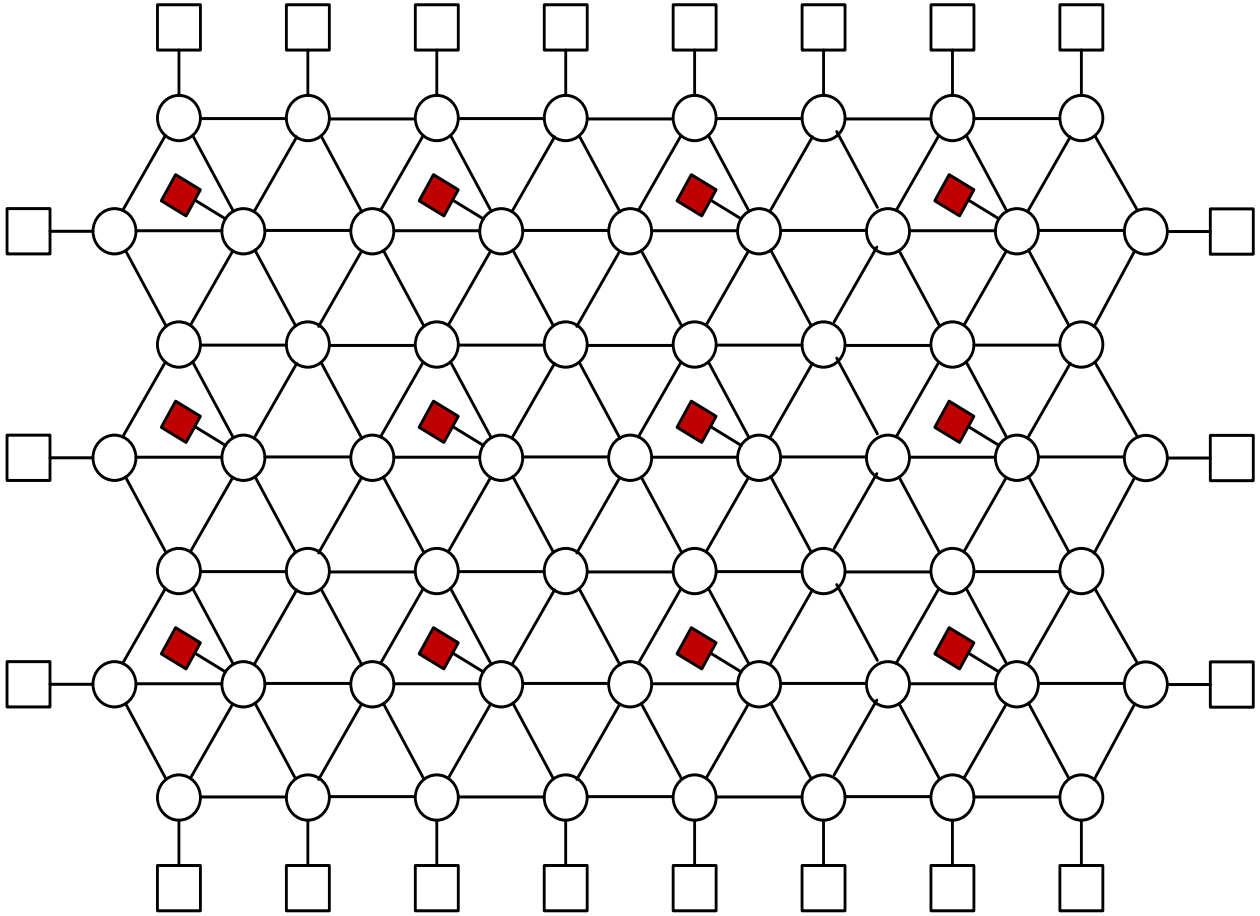

__init__
OnSend
OnRecv
-
-
RISC-V
m_d
m_c
Properties
State
ReqSnd
p_4
p_{10}
p_{13}
p_3
s''''_4
s'_{10}
s''_{13}
s'''_3
0
0
0
0

Exfiltration (output)

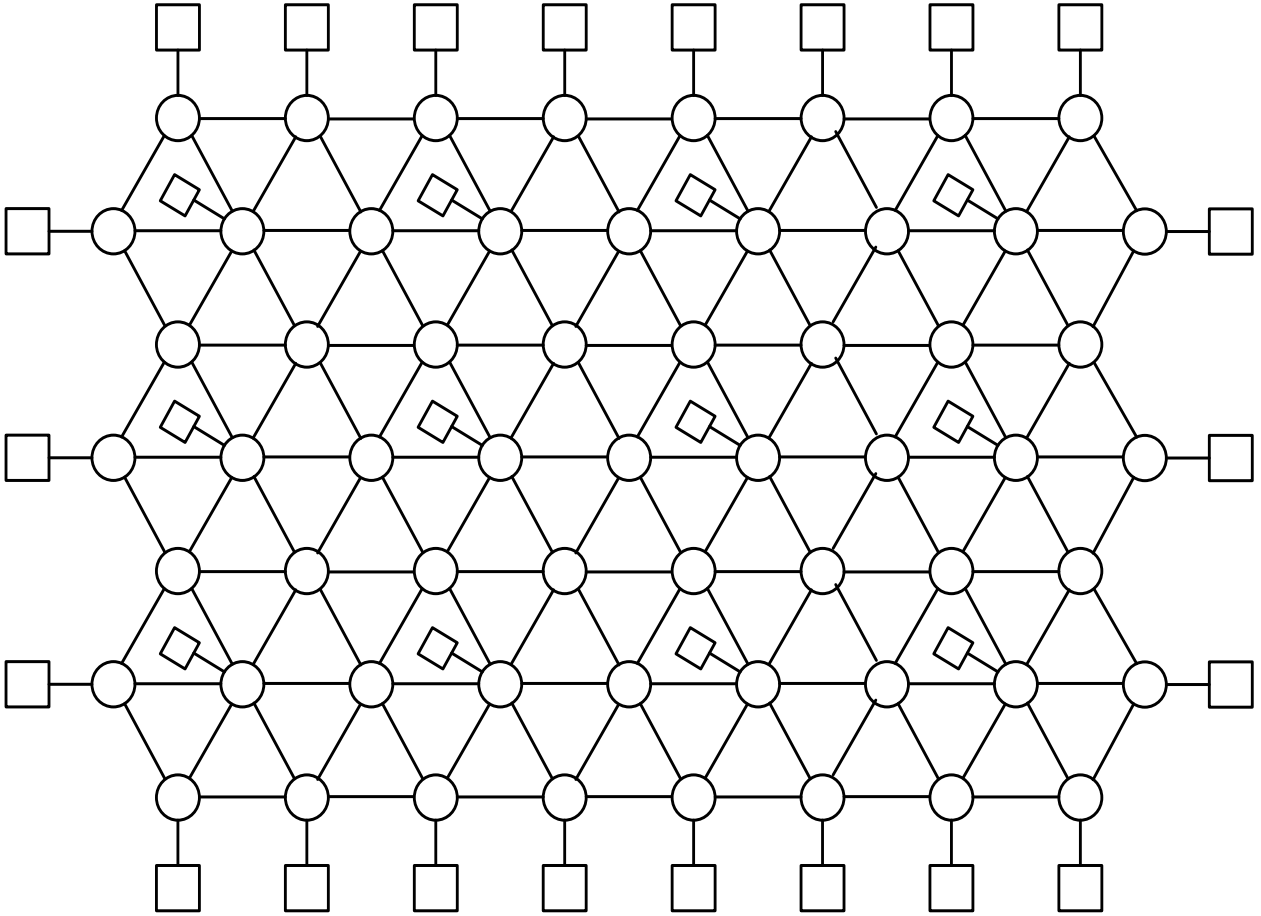




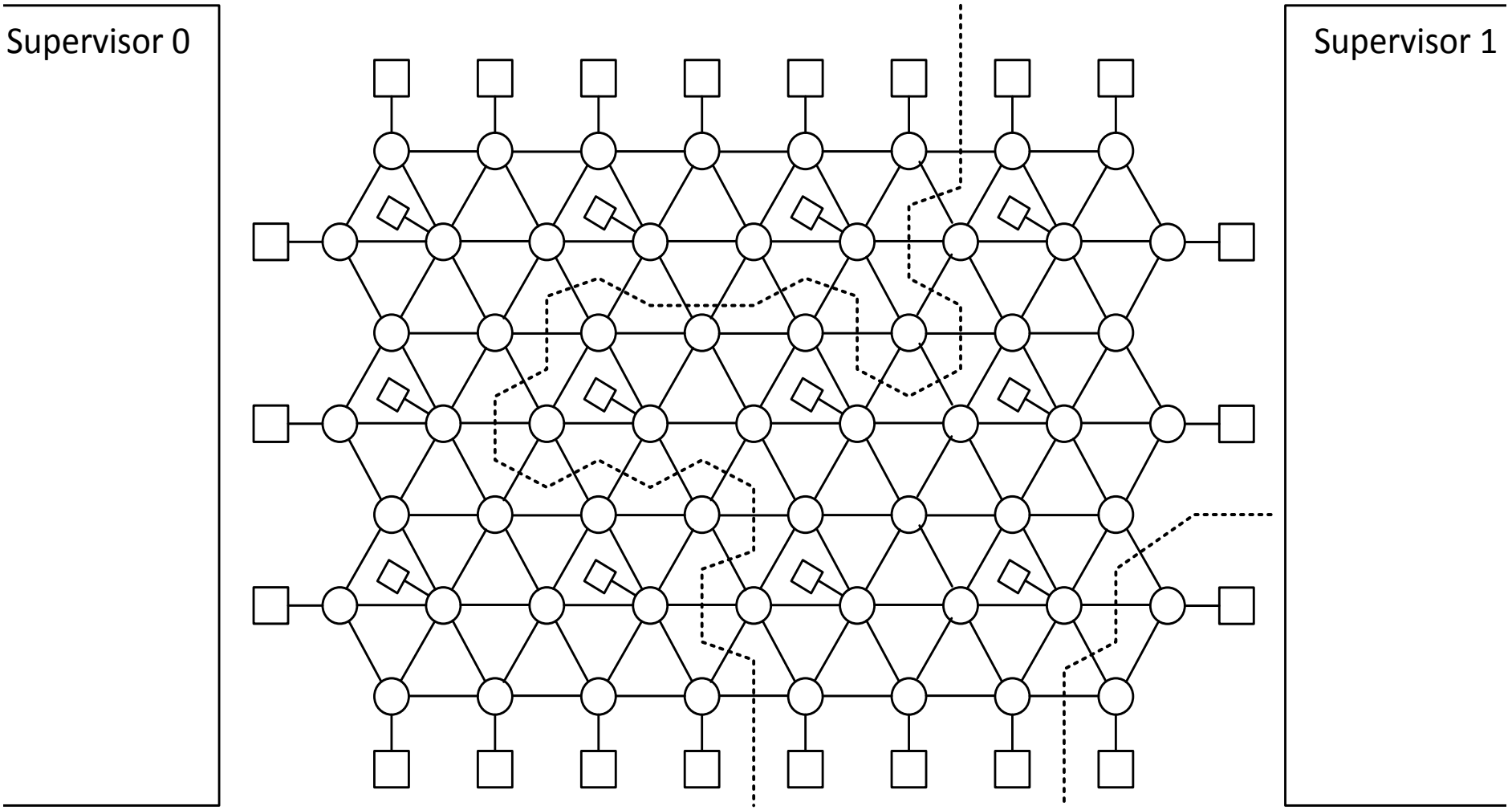




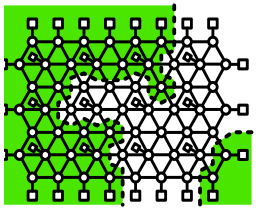
Supervisor 0

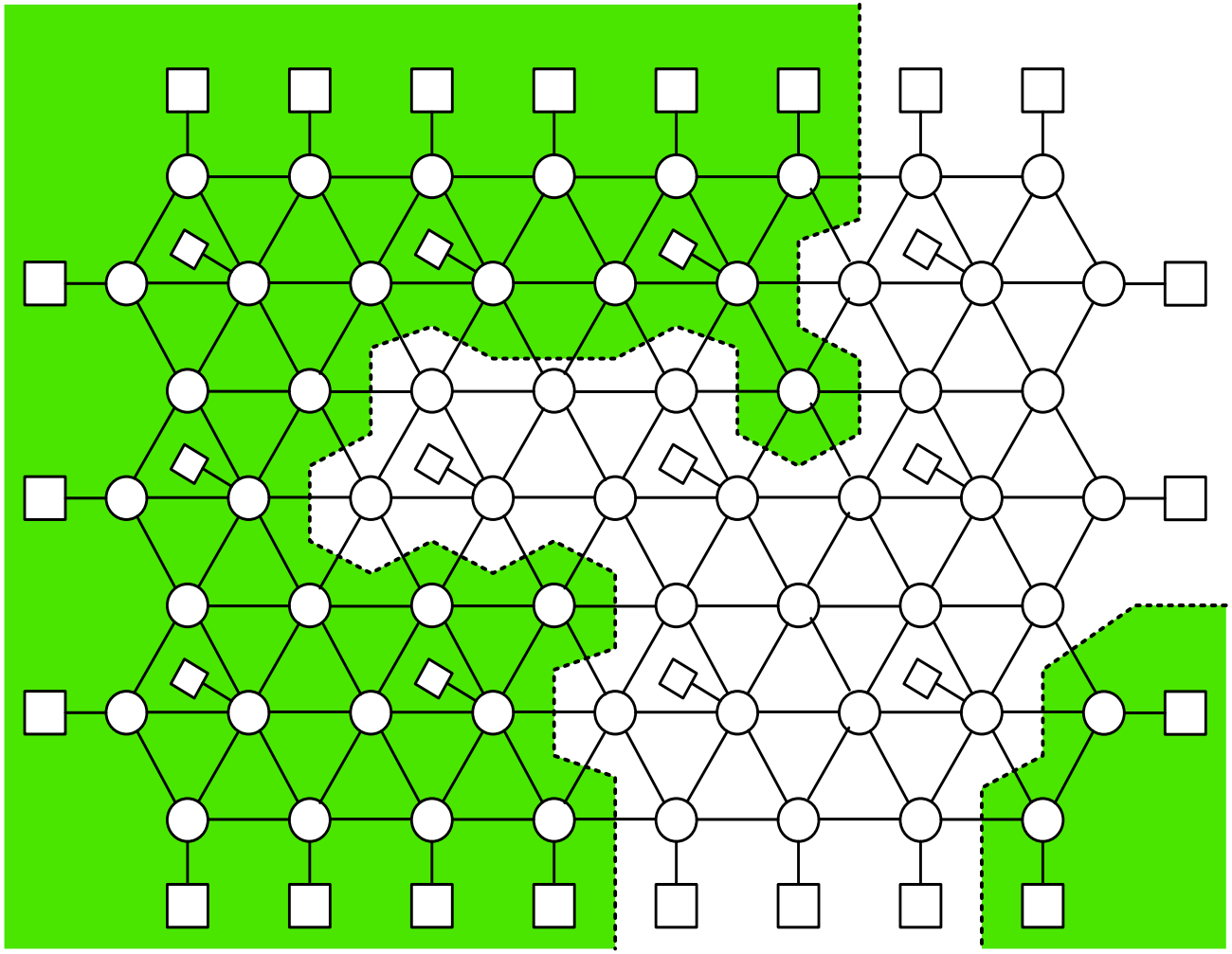


Supervisor 1

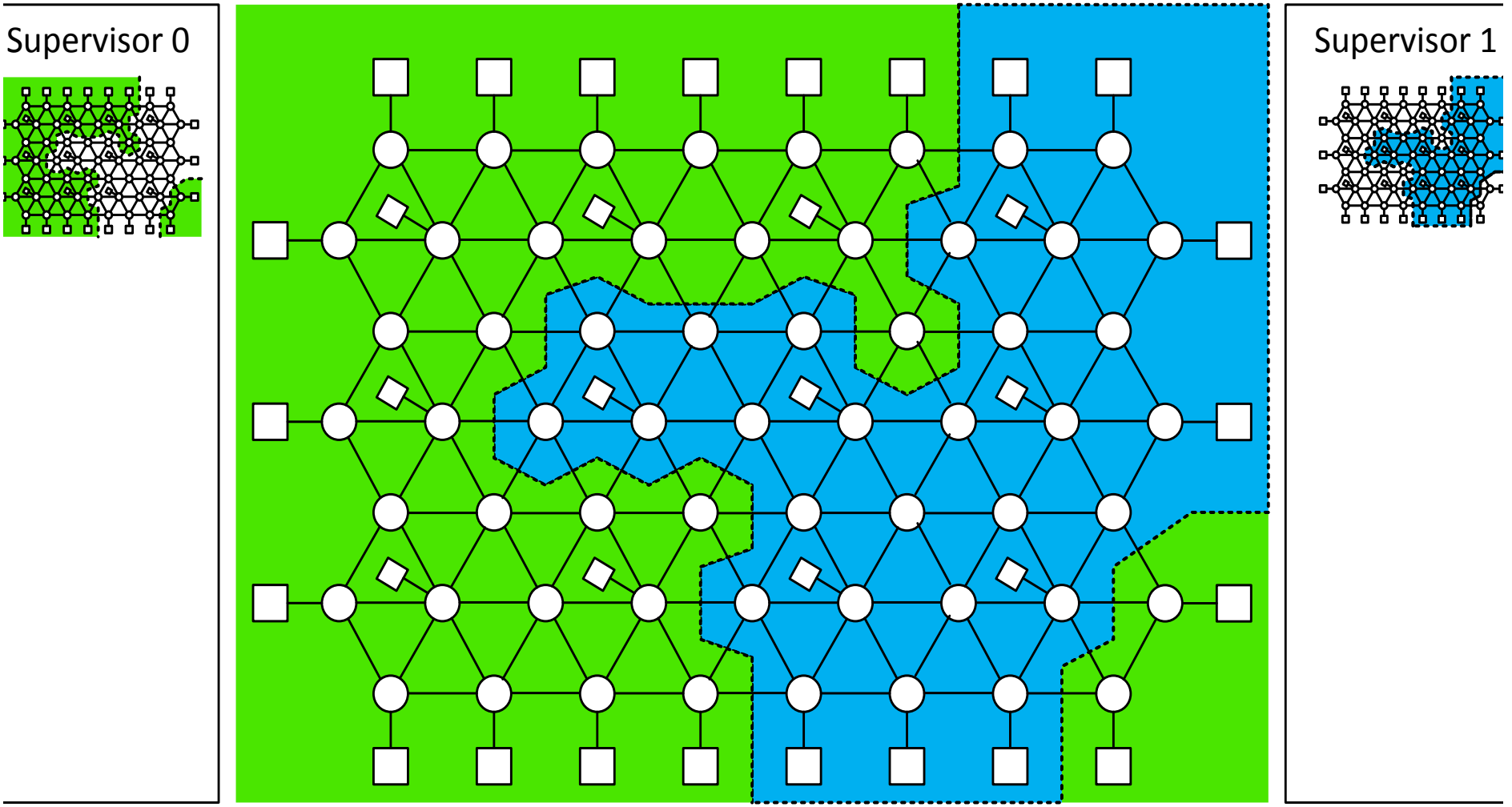

Supervisor 0
Supervisor 1

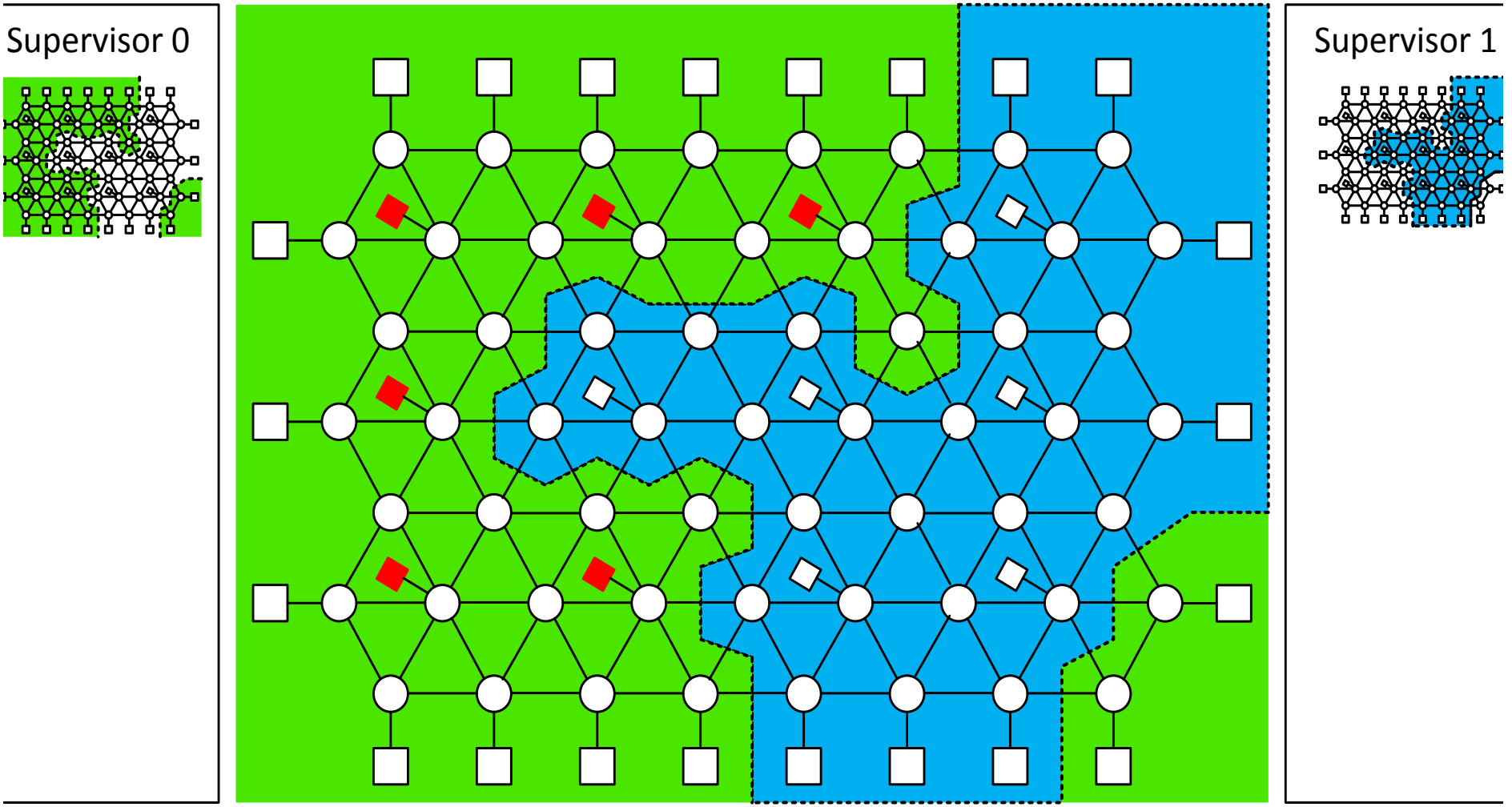
Supervisor 0

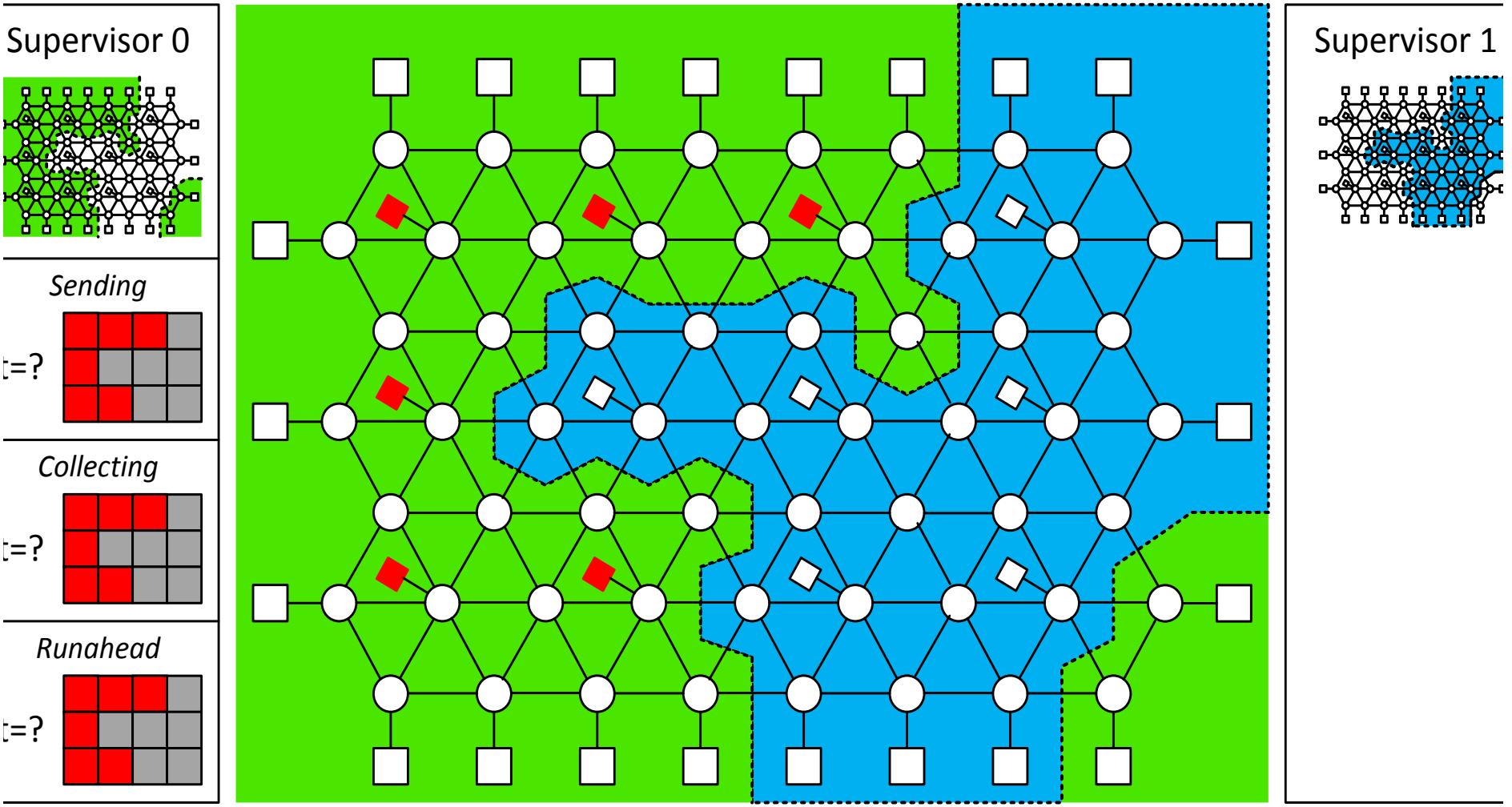


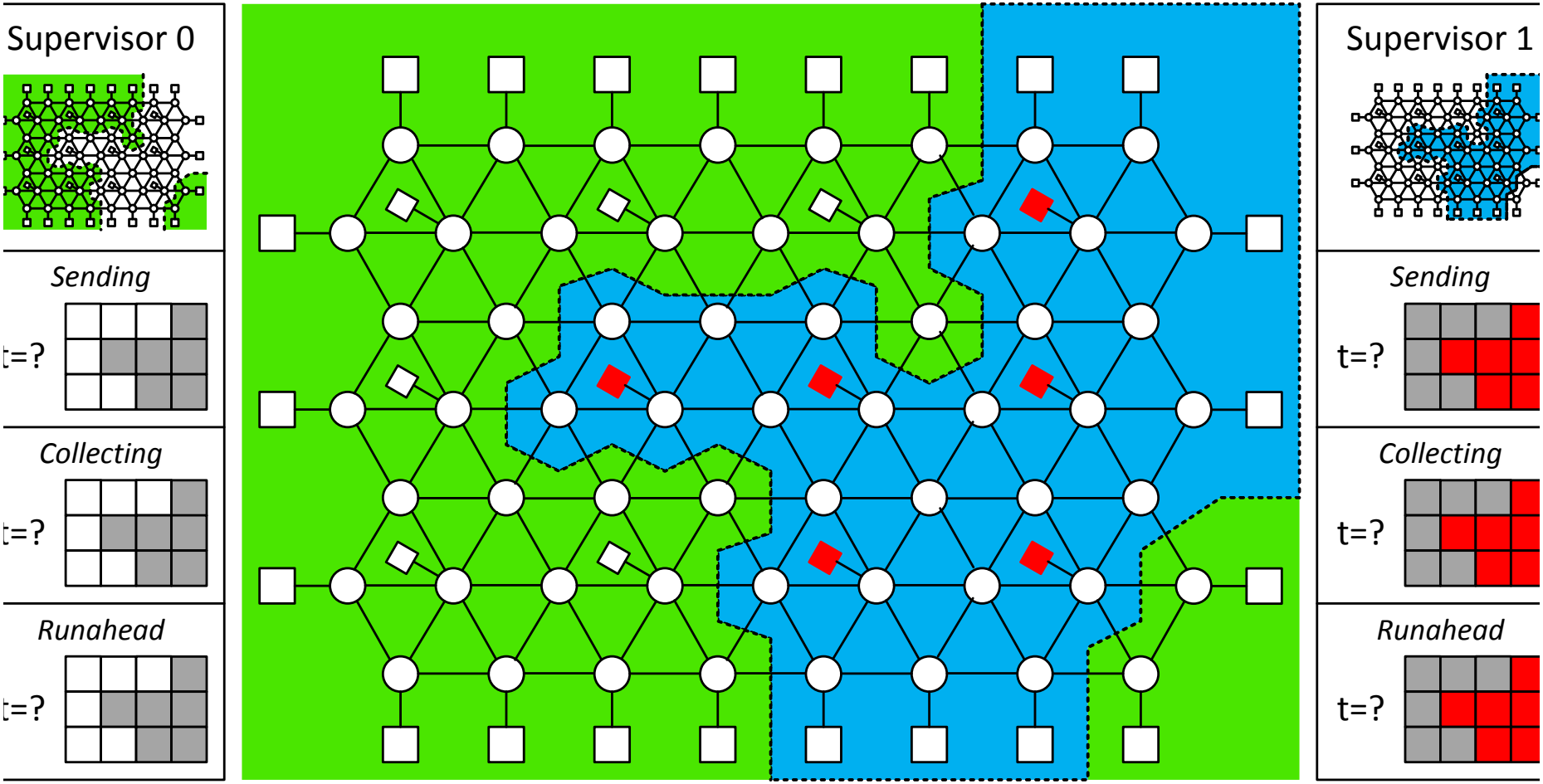


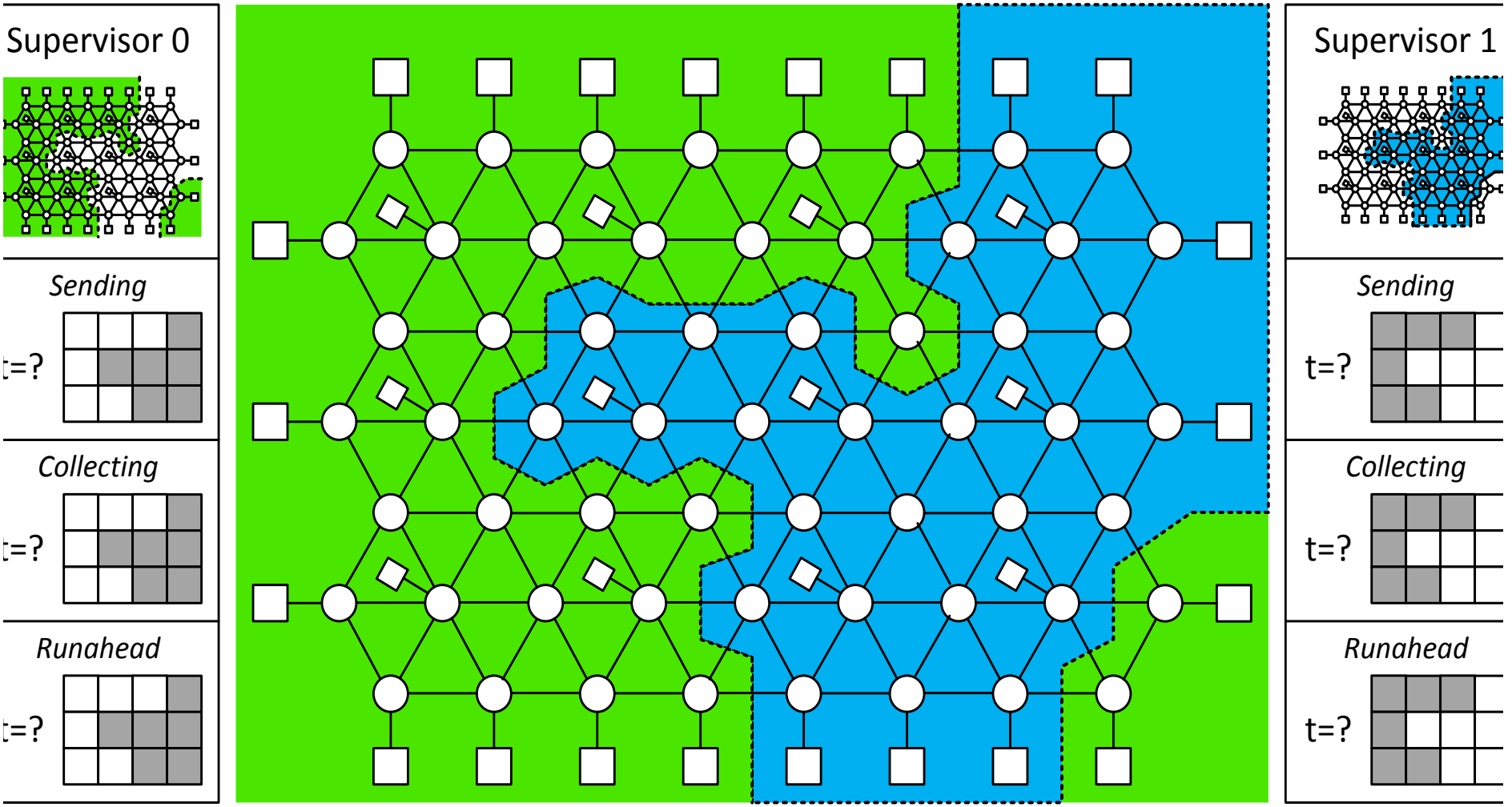
Supervisor 1

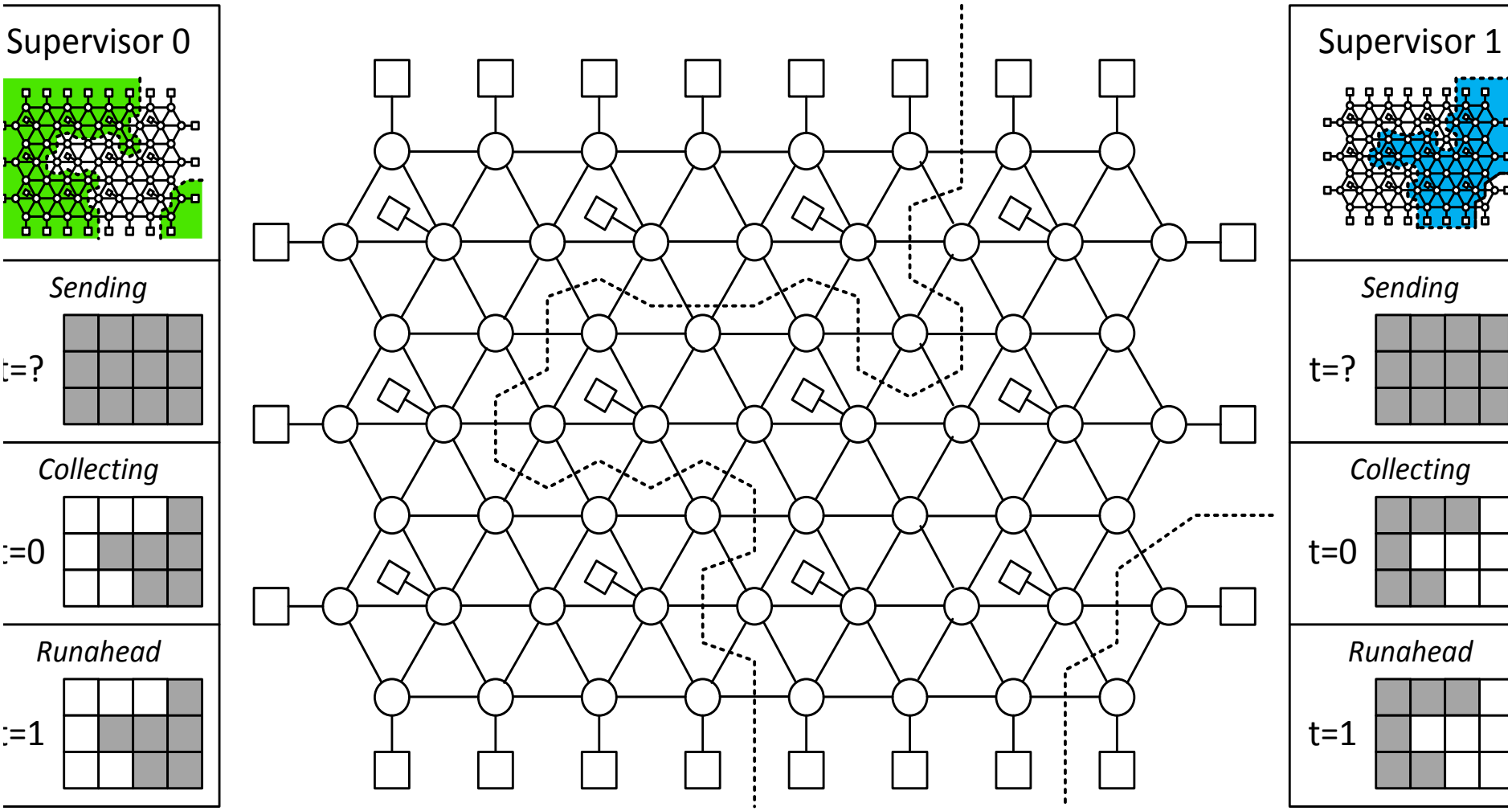

Supervisor 0
Supervisor 1

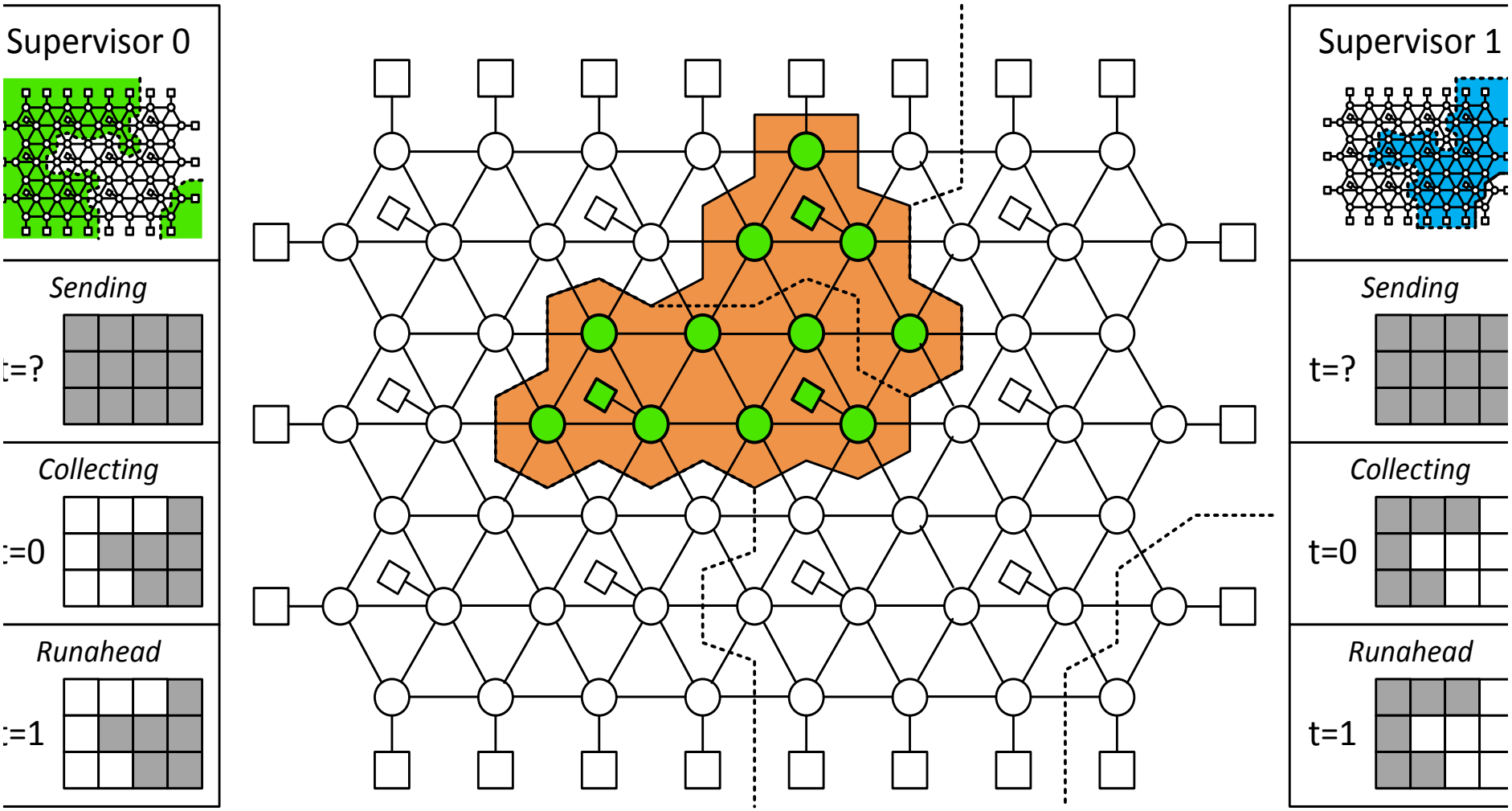

Supervisor 0
Supervisor 1

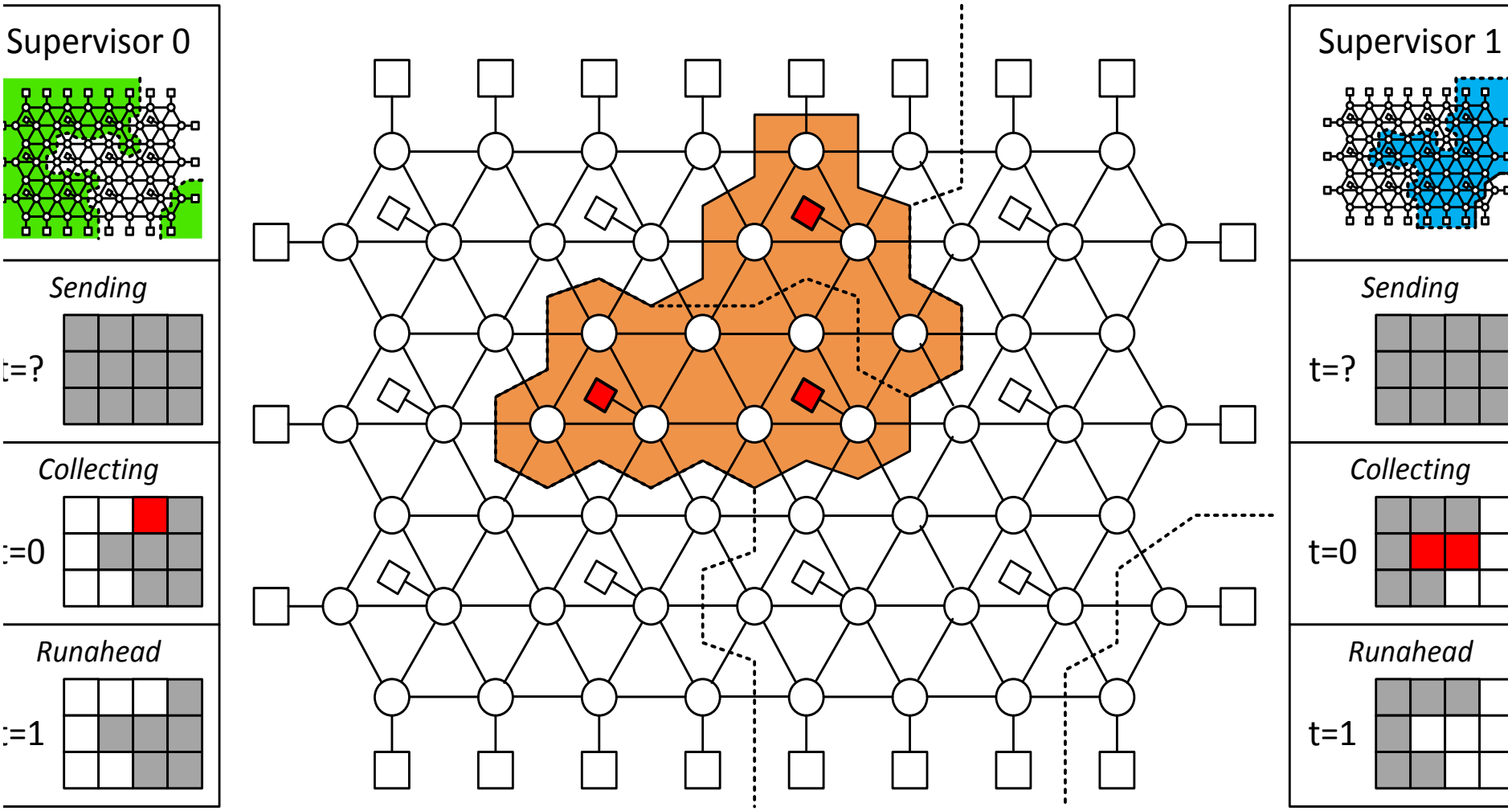

Supervisor 0
Sending
t=?
Collecting
t=?
Runahead
t=?
Supervisor 1

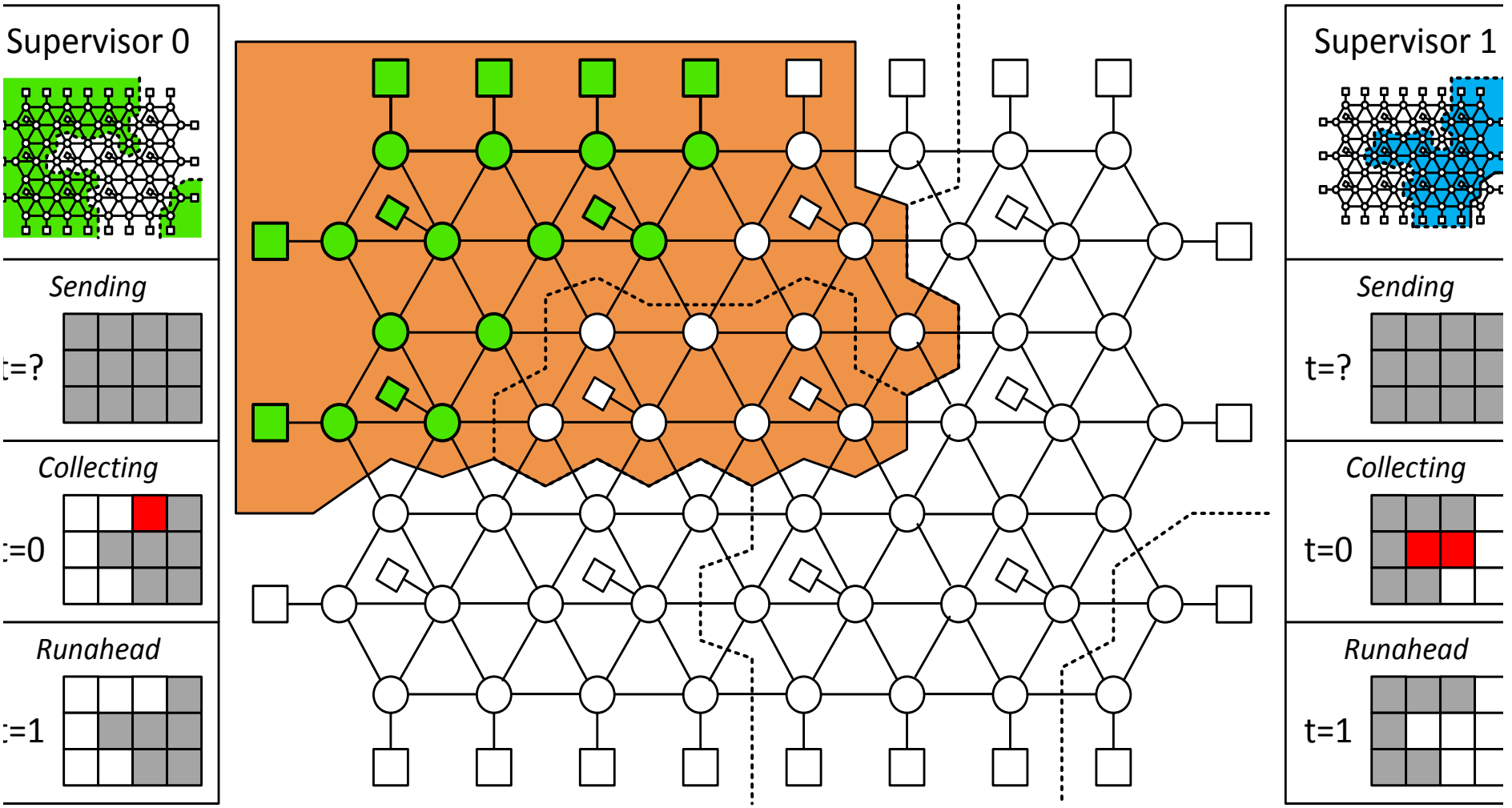

Supervisor 0
Sending
t=?
Collecting
t=?
Runahead
t=?
Supervisor 1
Sending
t=?
Collecting
t=?
Runahead
t=?

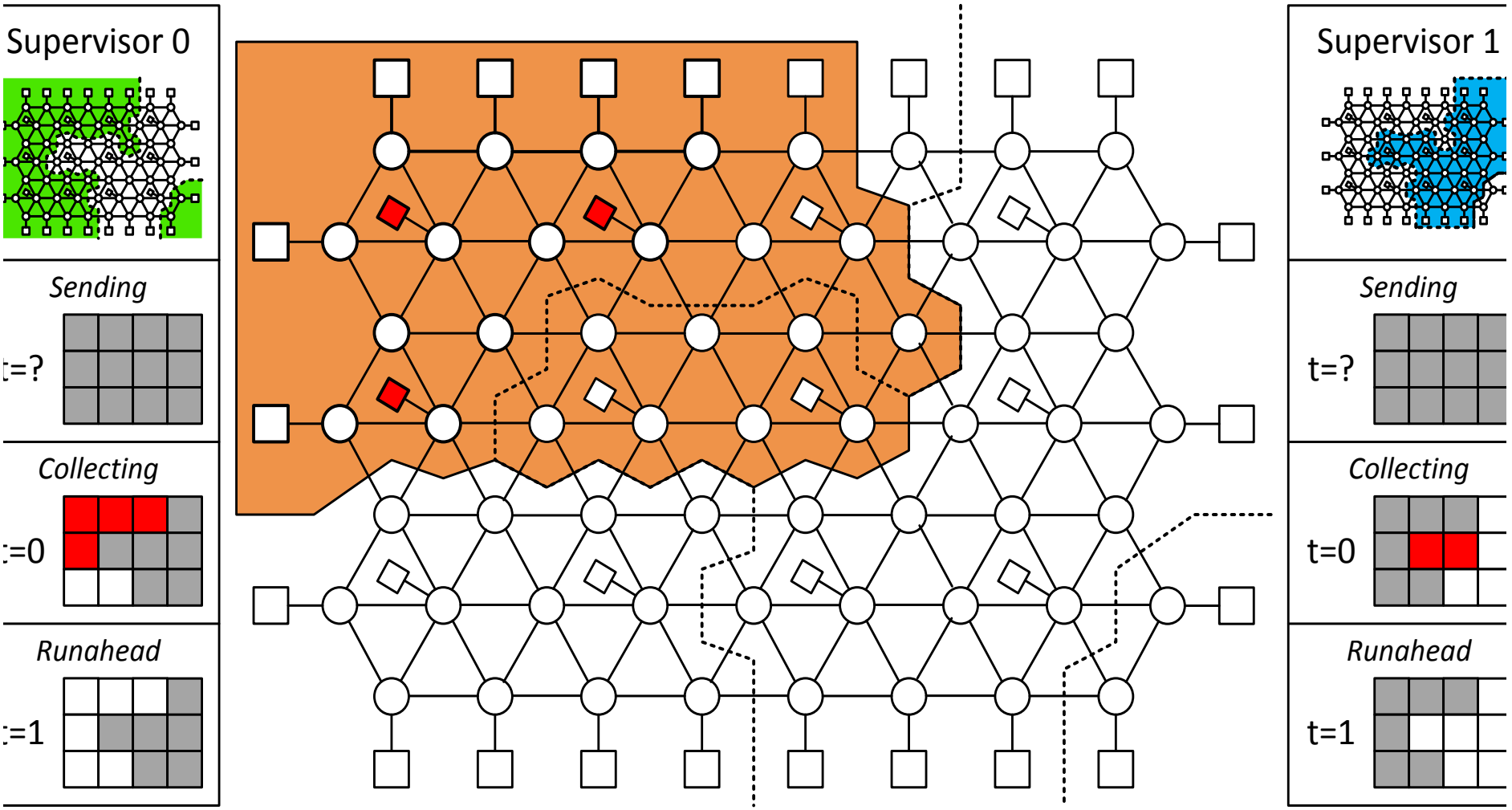

Supervisor 0
Sending
t=?
Collecting
t=?
Runahead
t=?
Supervisor 1
Sending
t=?
Collecting
t=?
Runahead
t=?

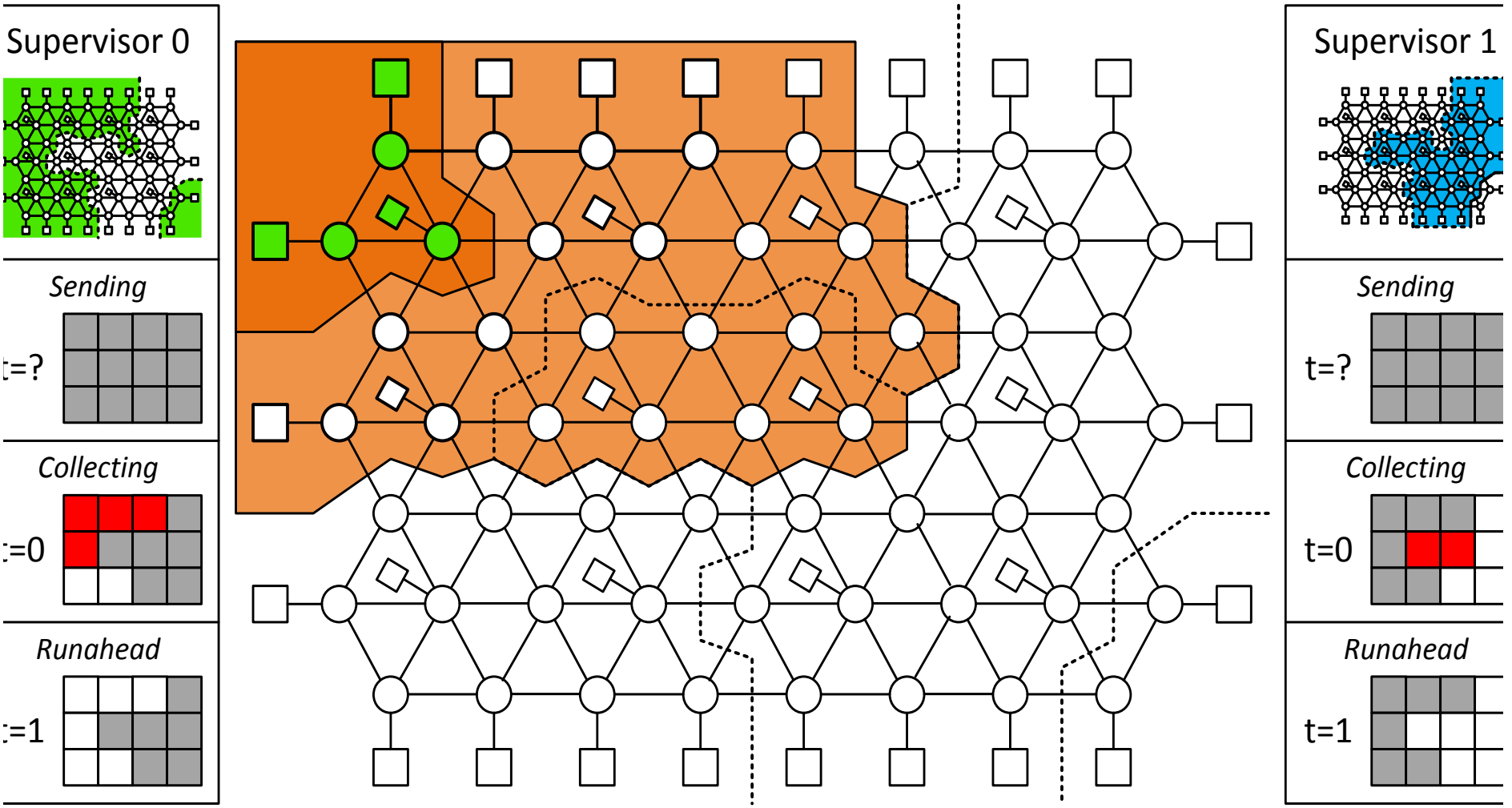

Supervisor 0
Sending
t=?
Collecting
t=0
Runahead
t=1
Supervisor 1
Sending
t=?
Collecting
t=0
Runahead
t=1

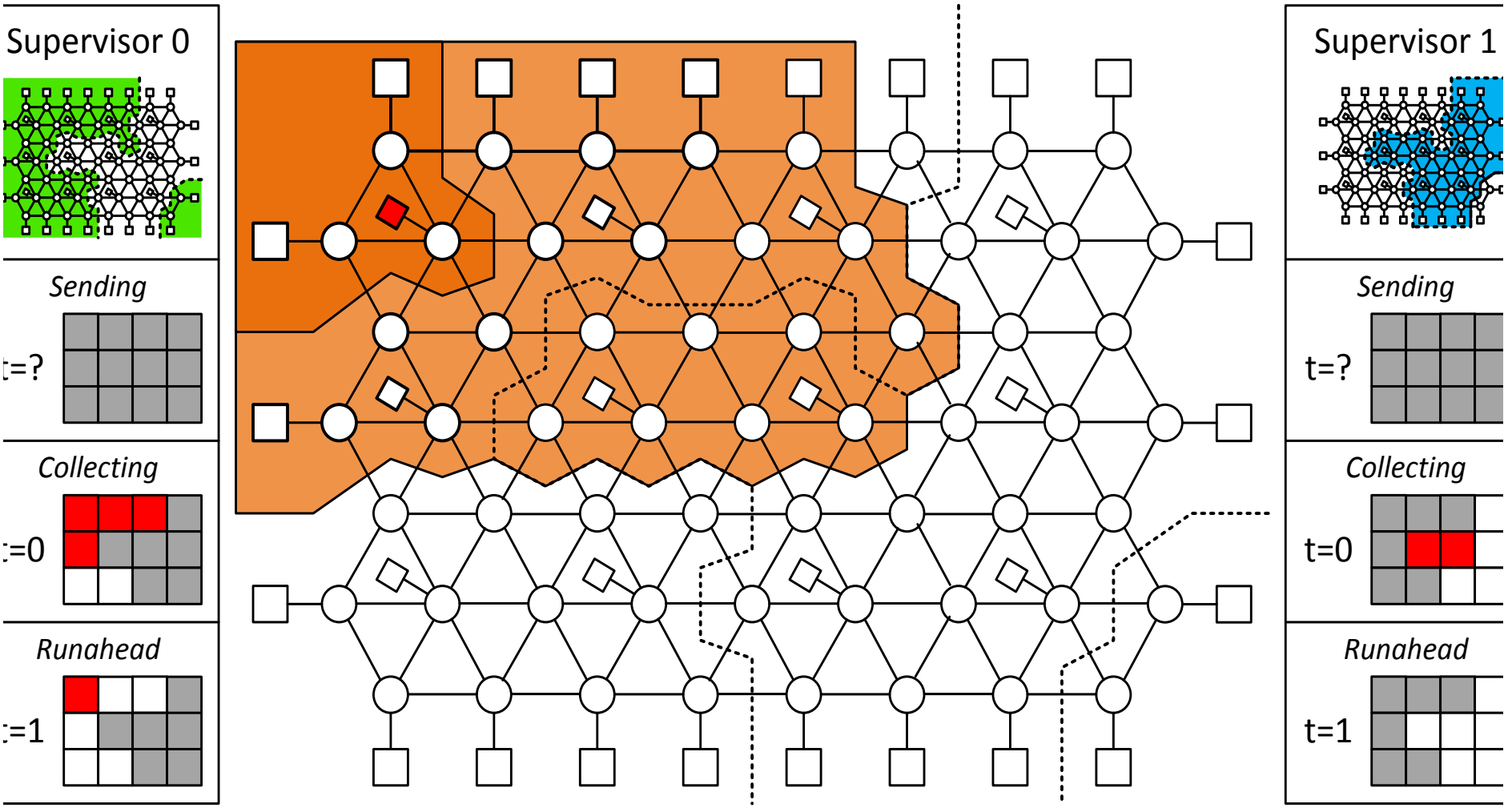

Supervisor 0
Sending
t=?
Collecting
t=0
Runahead
t=1
Supervisor 1
Sending
t=?
Collecting
t=0
Runahead
t=1

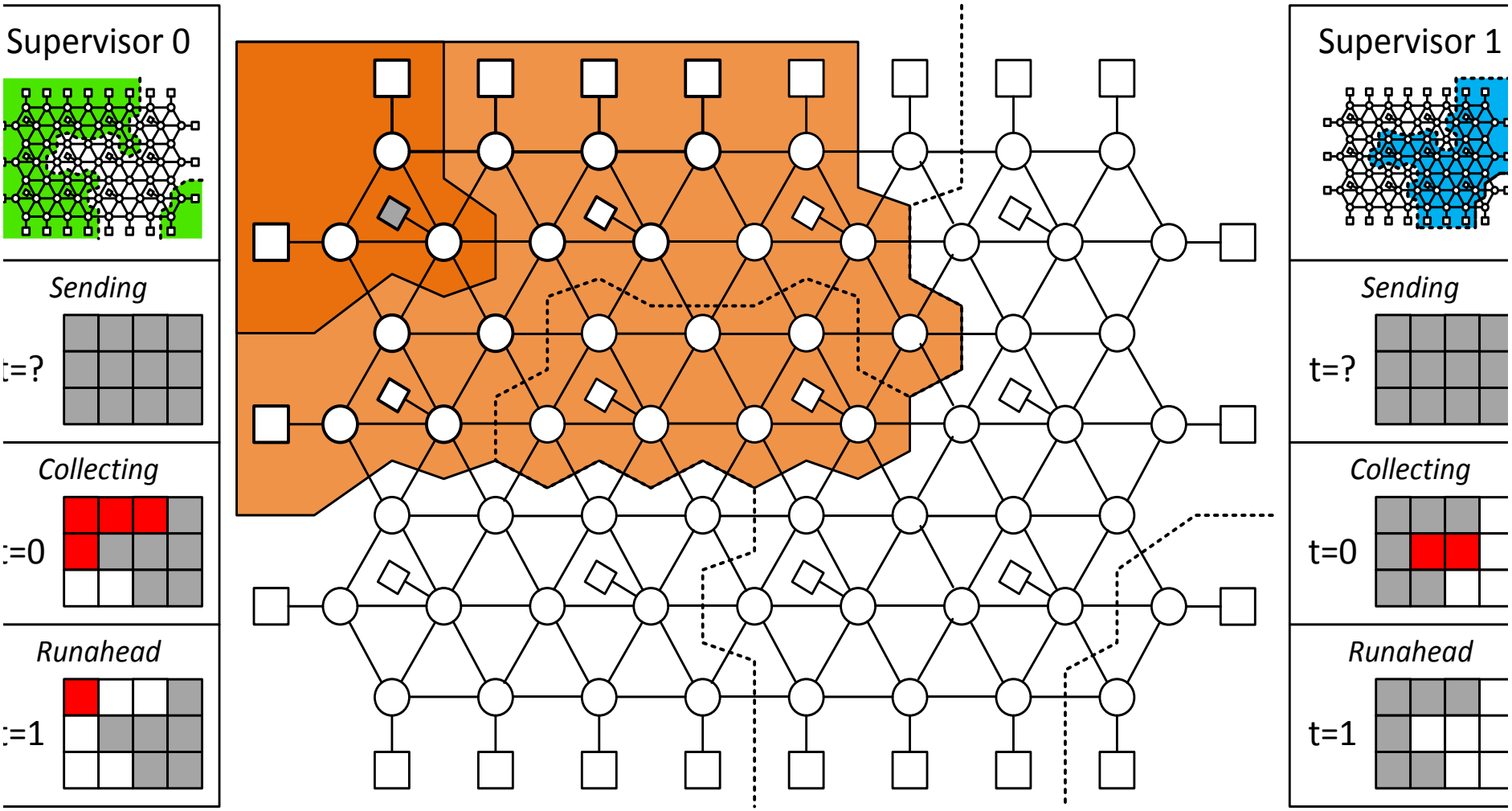

Supervisor 0
Sending
t=?
Collecting
t=0
Runahead
t=1
Supervisor 1
Sending
t=?
Collecting
t=0
Runahead
t=1

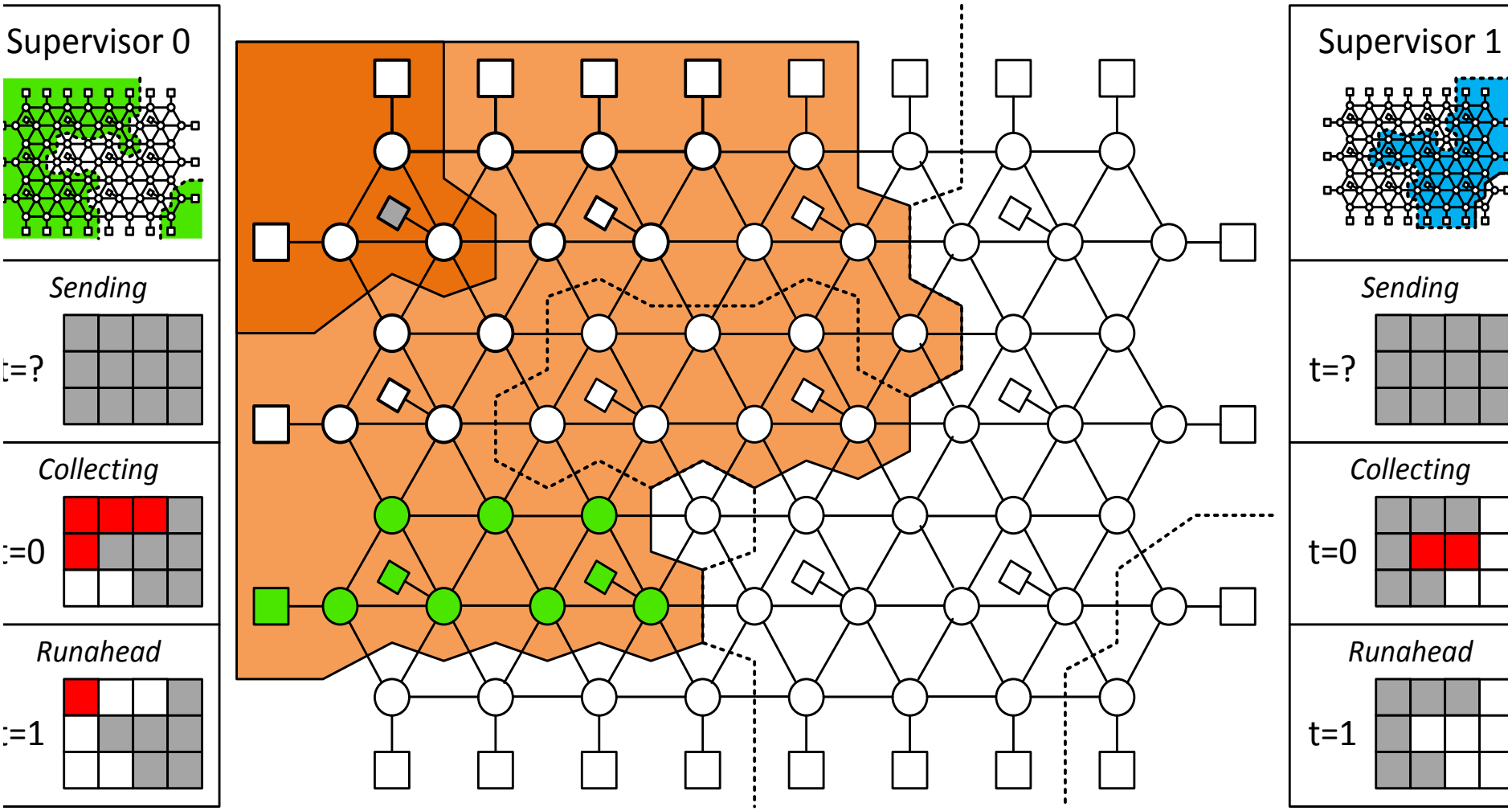

Supervisor 0
Sending
t=?
Collecting
t=0
Runahead
t=1
Supervisor 1
Sending
t=?
Collecting
t=0
Runahead
t=1

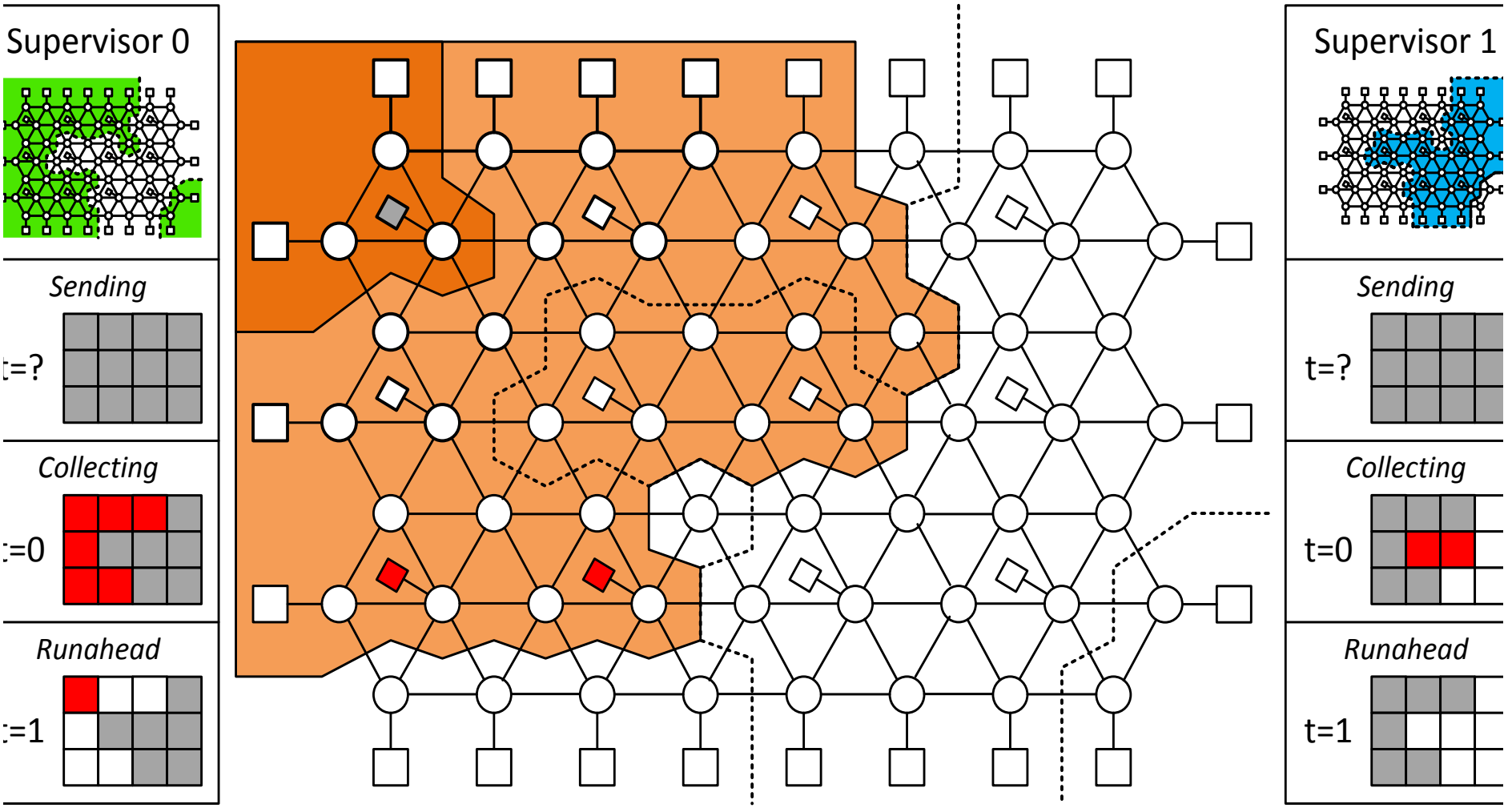

Supervisor 0
Sending
t=?
Collecting
t=0
Runahead
t=1
Supervisor 1
Sending
t=?
Collecting
t=0
Runahead
t=1

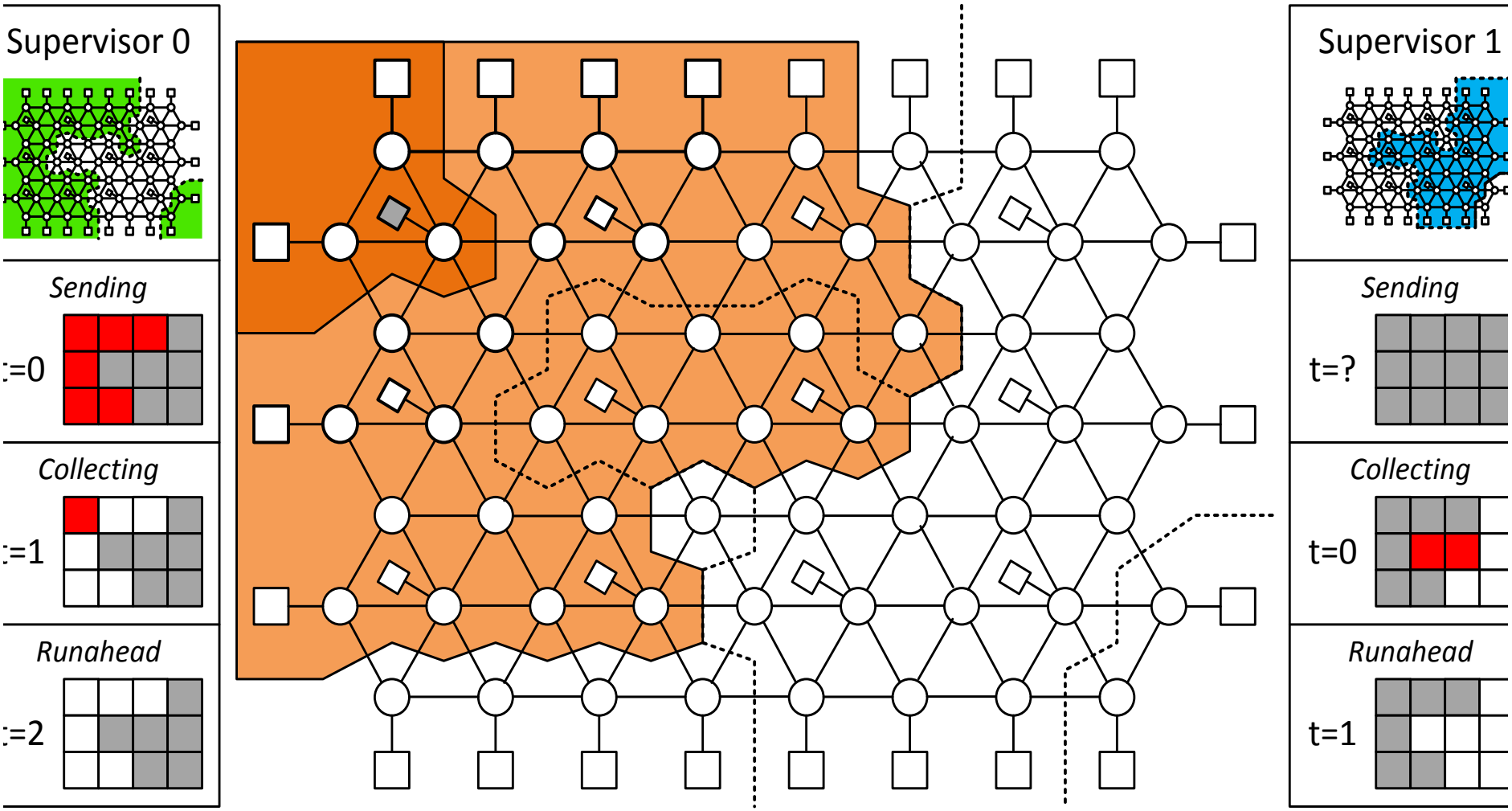

Supervisor 0
Sending
t=?
Collecting
t=0
Runahead
t=1
Supervisor 1
Sending
t=?
Collecting
t=0
Runahead
t=1

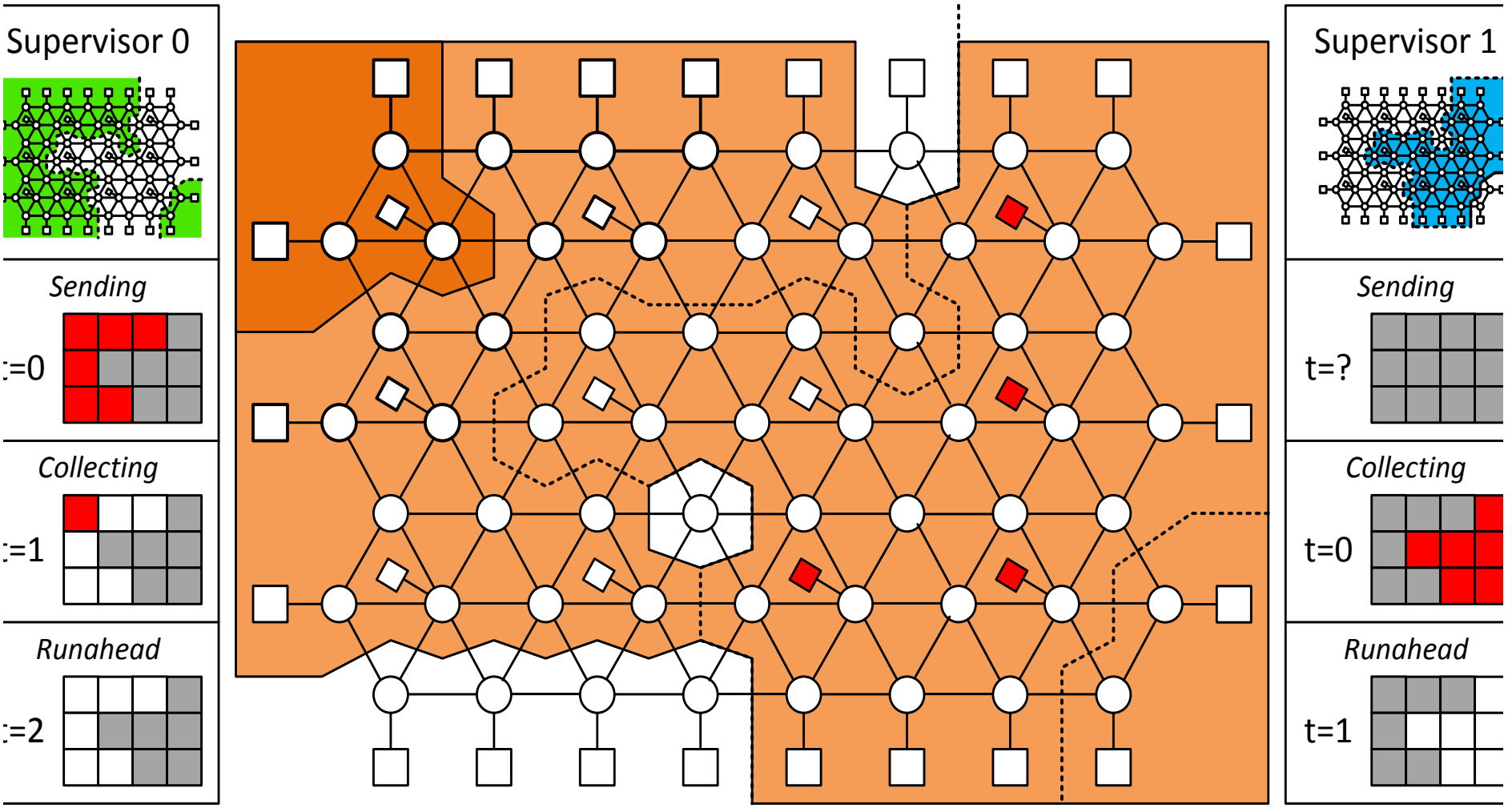

Supervisor 0
Sending
t=?
Collecting
t=0
Runahead
t=1
Supervisor 1
Sending
t=?
Collecting
t=0
Runahead
t=1

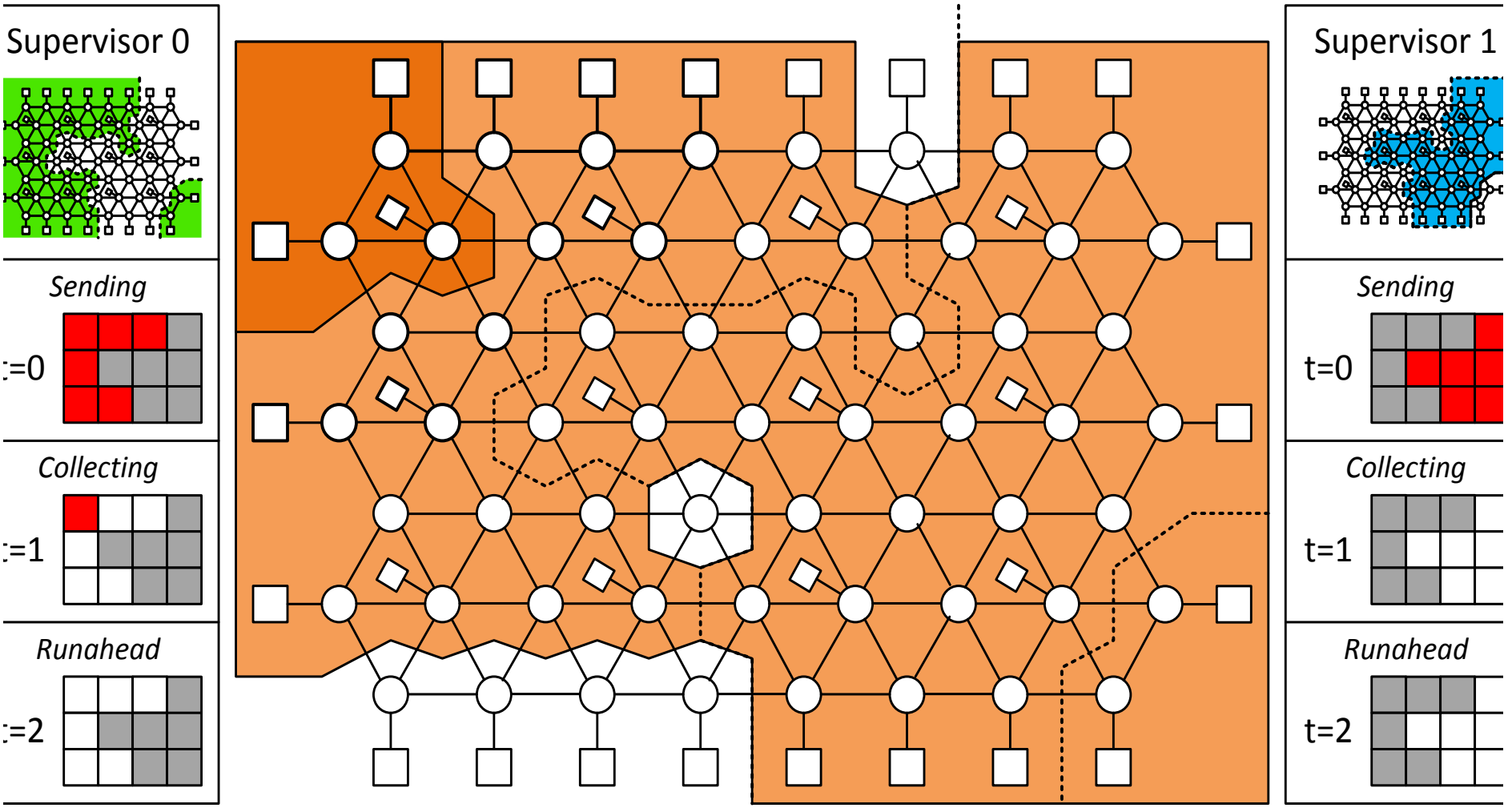

Supervisor 0
Sending
t=?
Collecting
t=0
Runahead
t=1
Supervisor 1
Sending
t=?
Collecting
t=0
Runahead
t=1

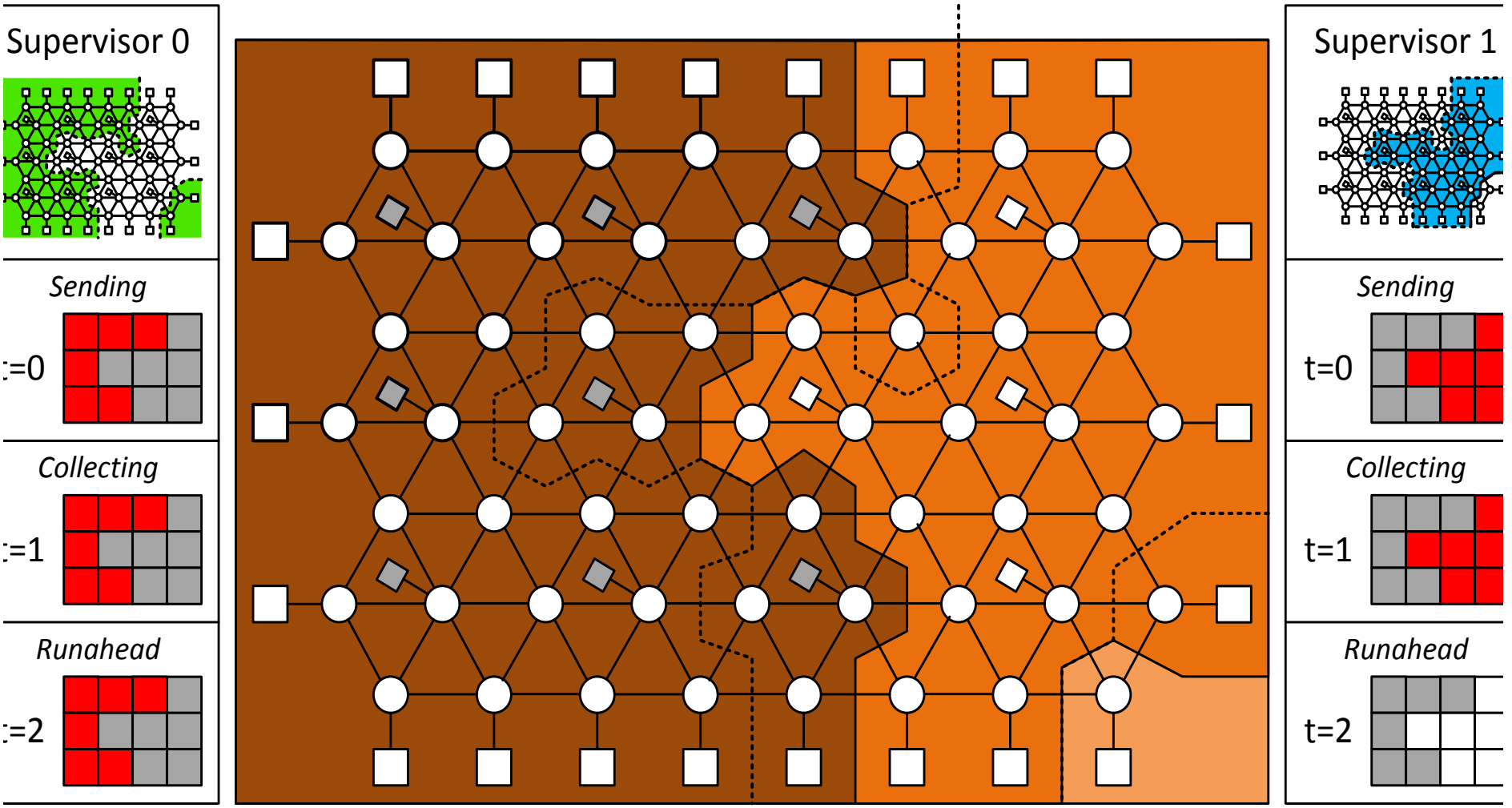

Supervisor 0
Sending
t=?
Collecting
t=0
Runahead
t=1
Supervisor 1
Sending
t=?
Collecting
t=0
Runahead
t=1

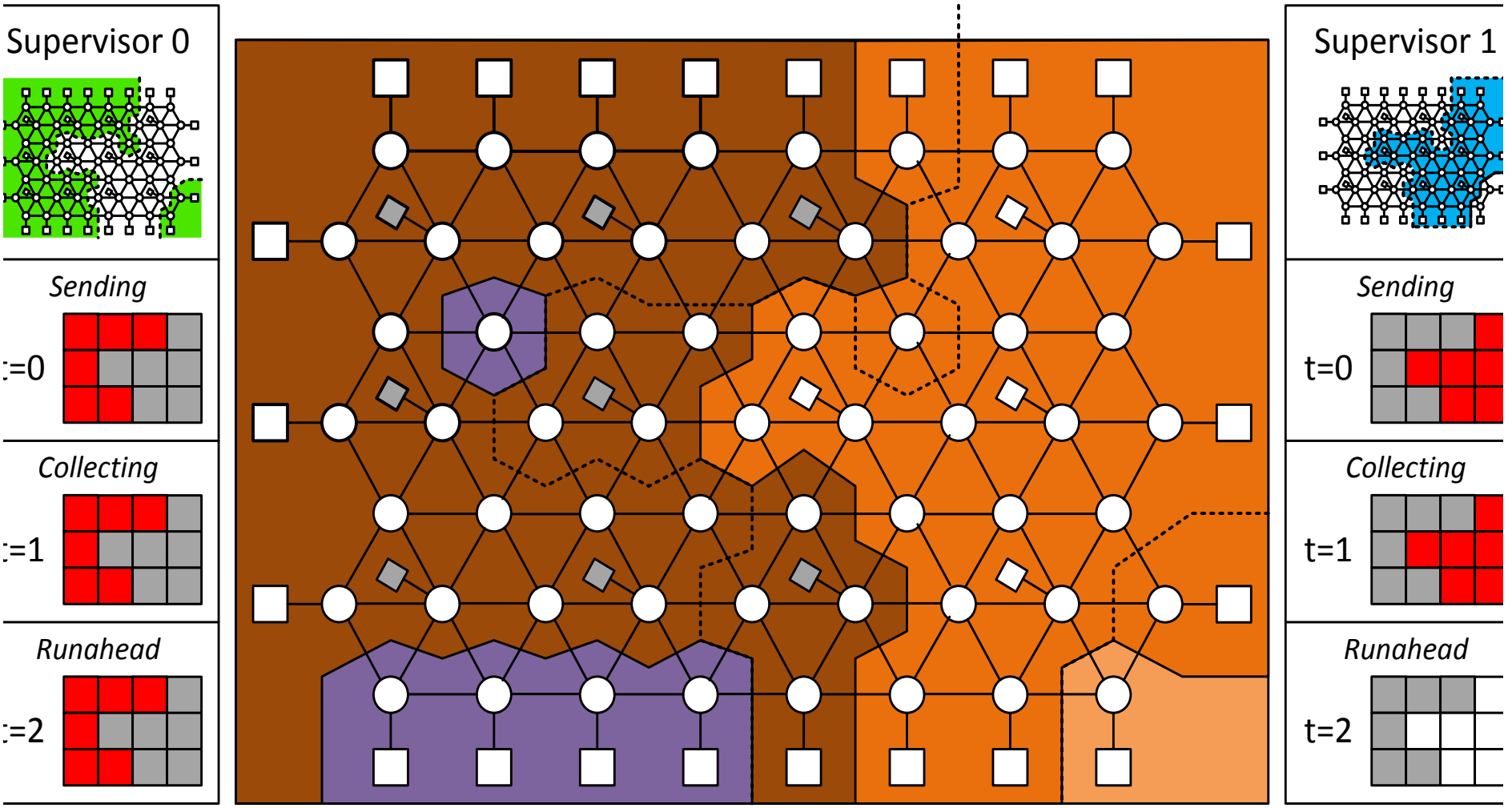

Supervisor 0
Sending
t=?
Collecting
t=0
Runahead
t=1
Supervisor 1
Sending
t=?
Collecting
t=0
Runahead
t=1


Supervisor 0
Sending
t=0
Collecting
t=1
Runahead
t=2
Supervisor 1
Sending
t=?
Collecting
t=0
Runahead
t=1


Supervisor 0
Sending
:=0
Collecting
:=1
Runahead
:=2
Supervisor 1
Sending
t=?
Collecting
t=0
Runahead
t=1


Supervisor 0
Sending
t=0
Collecting
t=1
Runahead
t=2
Supervisor 1
Sending
t=0
Collecting
t=1
Runahead
t=2


Supervisor 0
Sending
t=0
Collecting
t=1
Runahead
t=2
Supervisor 1
Sending
t=0
Collecting
t=1
Runahead
t=2


Supervisor 0
Sending
t=0
Collecting
t=1
Runahead
t=2
Supervisor 1
Sending
t=0
Collecting
t=1
Runahead
t=2

CW6

CW6 – Speed up the simulator

- I have multiple applications written for POETS
  - Heat equation
  - Ising Spin
  - Helmholtz (EM field)
  - Navier Stokes (fluid mechanics)
- They all work, but...
  - Take a very long to simulate
  - https://poetsii.github.io/graph_schema/heat/test.html
  - It's difficult to estimate performance on real hardware
  - Bottlenecks are hard to find
- I'll give you a simulator for POETS
- You make it as fast as possible

And that’s it for lectures

- Available for consultation next Friday in the same slot (?)
- CW6 spec released tomorrow (Sat) morning
  - Have a good nights sleep...

- Feedback to keep coming
  - CW5 + CW6 feedback mostly given in orals
  - Orals scheduled on one-to-one basis in Spring term