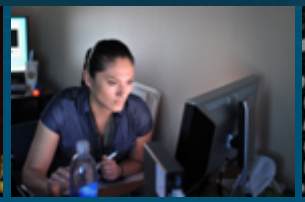
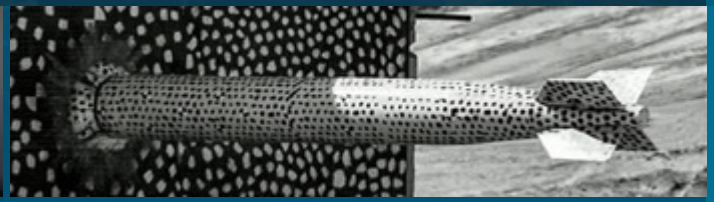
SAND2021-3357PE

Sandia National Laboratories

SST – llyr Development Update





Presented By:

Clay Hughes, Sandia National Laboratories

U.S. DEPARTMENT OF ENERGY | NNSA National Nuclear Security Administration

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Dataflow Accelerators

Compilers build internal representations of applications that represent the behavior as a series of graphs -- abstracting the control and data flow

Traditional processors execute instruction sequentially, destroying an application's inherent ILP

- Superscalar OoO processors go to great lengths to reconstruct the ILP
  - Multiple queues and complex logic allows instructions to issue when operands are available rather than in program order
  - Results are placed in additional queues and made visible to the system in program order

Dataflow architectures are able to execute these graphs directly, without the need to flatten the graph and artificially recover the parallelism

- Dedicated or shared PEs
- Statically or dynamically scheduled

Simulating Dataflow Architectures

There are examples of each of the different types of accelerators in the literature

- Dedicated/Static
  - Softbrain, MAERI(?)
- Dedicated/Dynamic
  - Plasticine, SPU
- Shared/Static
  - CGRA
- Shared/Dynamic
  - TRIPS, SGMF

The SST dataflow component, llyr, is being developed to model dedicated/static designs

- Flexible interface to add custom PEs
- Arbitrary connectivity (can even be used to model ReRAM-like crossbars)
- Mappers are dynamically loaded allowing them to be swapped at runtime
- Leverages SST component interface to enable scalable simulations

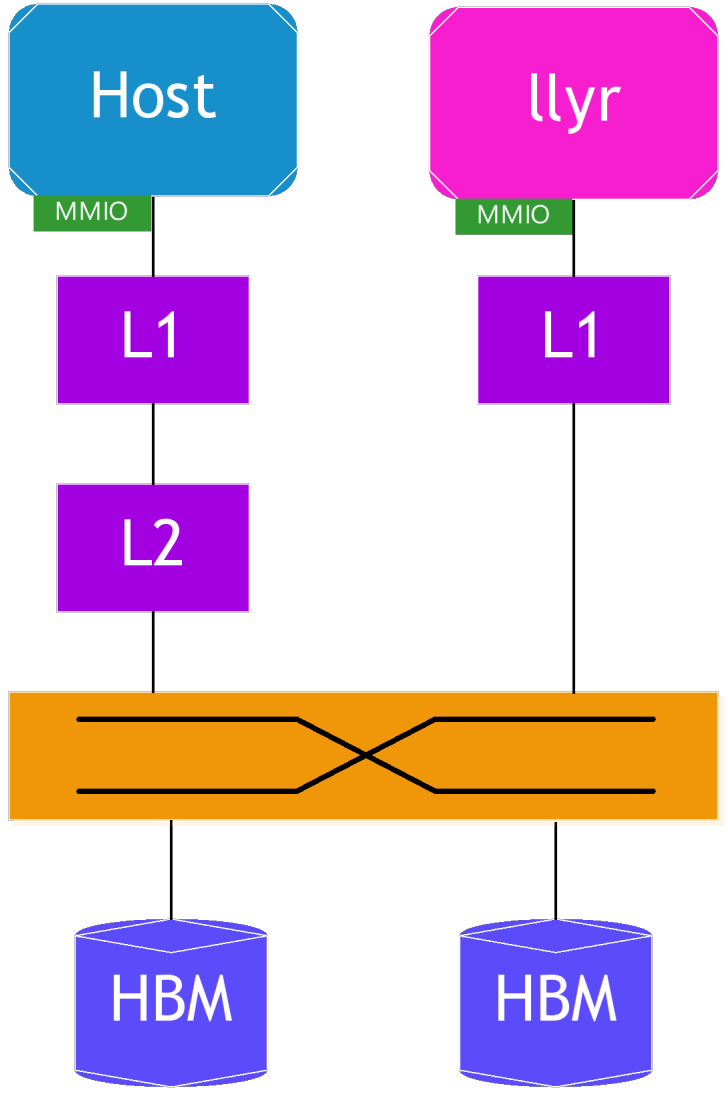
SST Ilyr Component

Instantiated like any other SST component

- Allows for…
  - Multiple instances, possibly with different configurations
  - Arbitrary memory hierarchy
  - Some limitations on node configurations
    - Unable to launch from device

Utilizes new MMIO interface

- Address range set aside for control
  - Doorbell and kernel location
- Can send and receive data in global address space
  - Eventually will be capable of self-hosted memory with TLB



*Currently being tested as a standalone compute device

SST Ilyr Component

PEs have a compute unit, input buffers, and output buffers

- Number of buffers bounded by connectivity
- Buffer depth is configurable but is uniform for all PEs

Current PE list

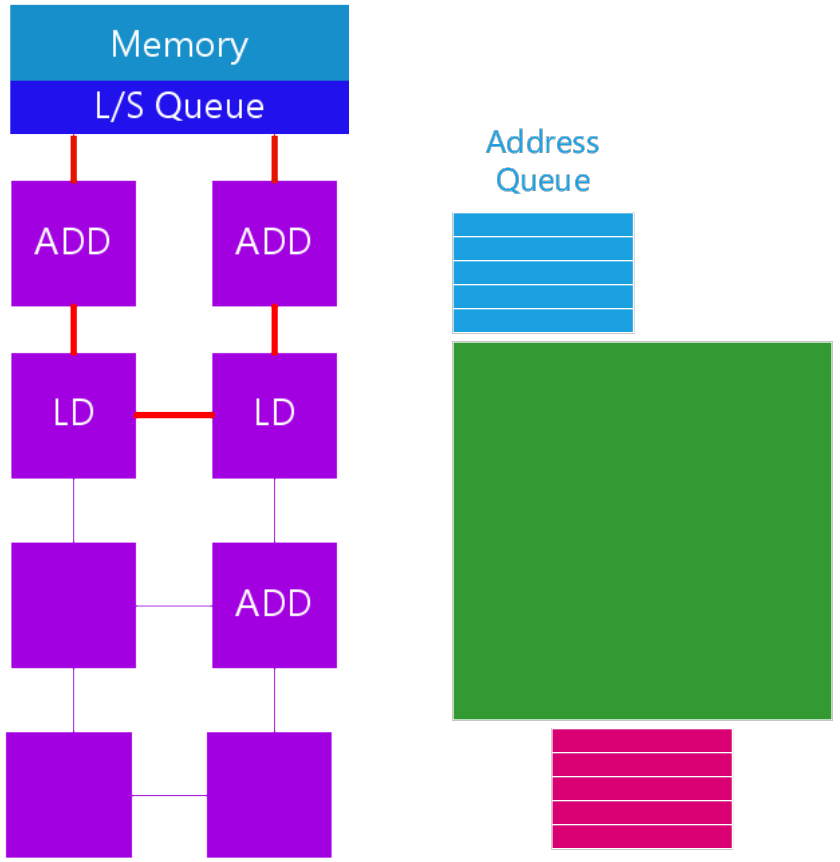
- LD, ST, LD_ST
- SLL, SLR, ROL, ROR
- ADD, SUB, MUL, DIV
- FPADD, FPSUB, FPMUL, FPDIV, FPMATMUL
- BUFFER
- ANY, ANYMEM, ANYLOGIC, ANYINT, ANYARITH, ANYFP

SST Ilyr Component

Address calculation performed by preceding PEs

Memory operations forced to return in program order via common L/S queue

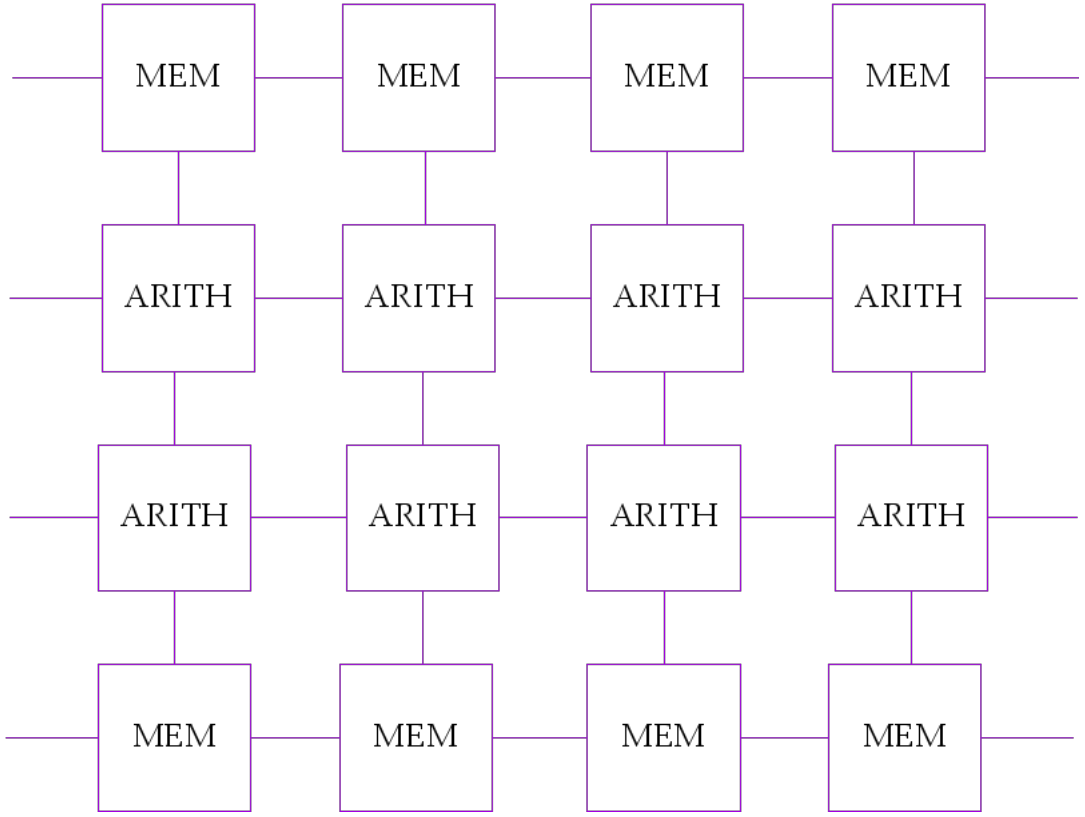
Reponses are forwarded directly to output queue of memory PE



SST Ilyr Component

Constructed using a hardware description file

- Describes connectivity between PEs
- Describes allowable operations per-PE




```
0 [pe_type=ANYMEM]          0 -- 1
1 [pe_type=ANYMEM]          0 -- 4
2 [pe_type=ANYMEM]          1 -- 0
3 [pe_type=ANYMEM]          1 -- 2
4 [pe_type=ANYARITH]        1 -- 5
5 [pe_type=ANYARITH]        2 -- 1
6 [pe_type=ANYARITH]        2 -- 3
7 [pe_type=ANYARITH]        2 -- 6
8 [pe_type=ANYARITH]        3 -- 2
9 [pe_type=ANYARITH]        3 -- 7
10 [pe_type=ANYARITH]       4 -- 1
11 [pe_type=ANYARITH]       4 -- 5
12 [pe_type=ANYMEM]         4 -- 8
…                           …
```

SST Ilyr Component

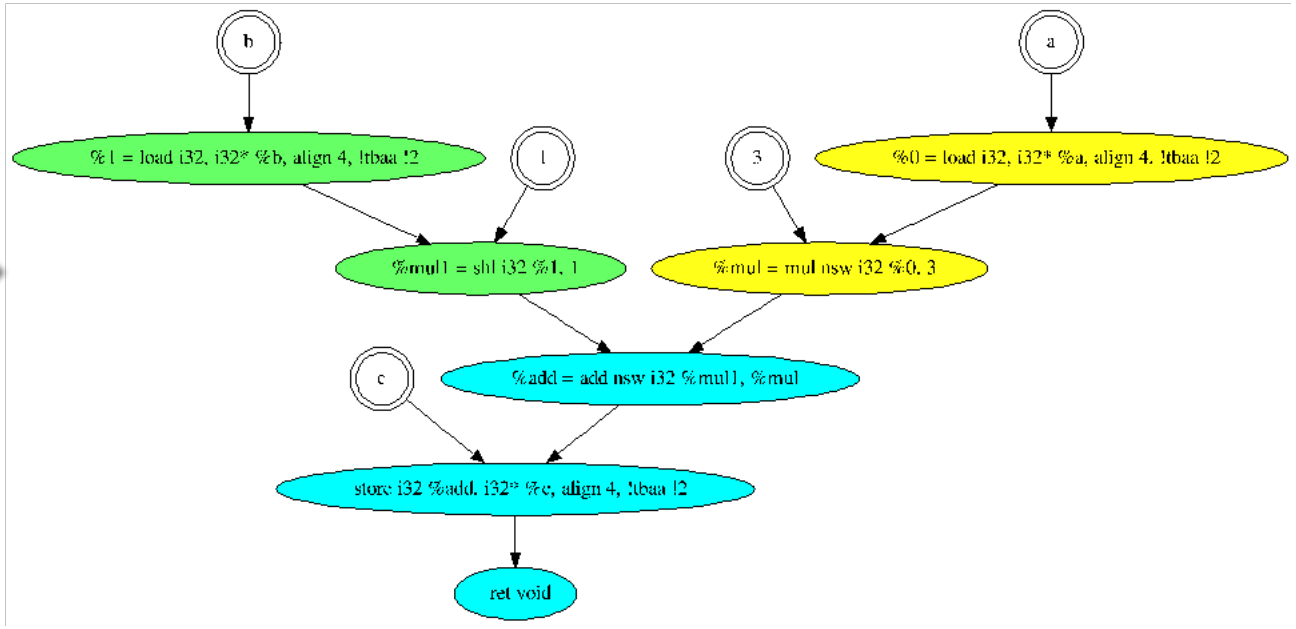
The hardware configuration controls the *potential* computation, but the application graph controls the *actual* computation

LLVM IR will be extracted from the ELF executable and passed to the Ilyr parser

Parser will construct a dataflow graph with control information (backward edges) from the LLVM IR

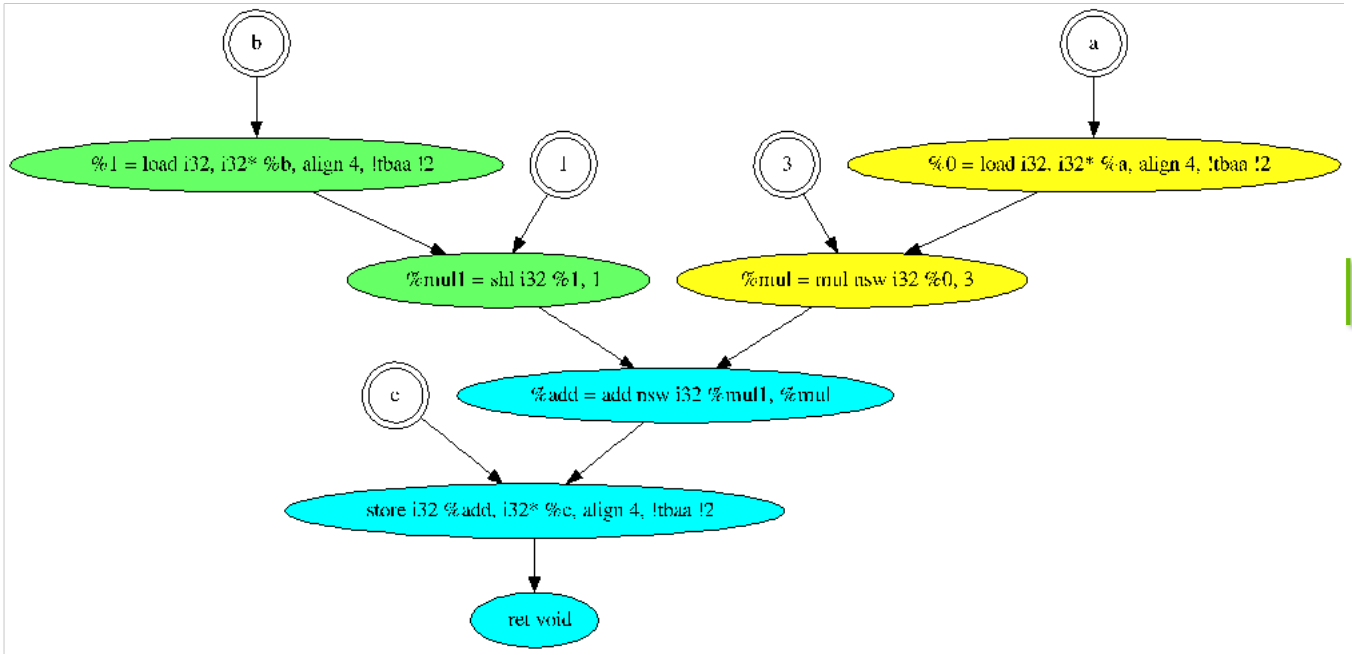
```
void  multiply_mod(int* a, int *b, int* const c)
{
  int d = *a;
  int e = *b;

  int f = 3 * d;
  int g = 2 * e;

  *c = f + g;
}
```

Parser

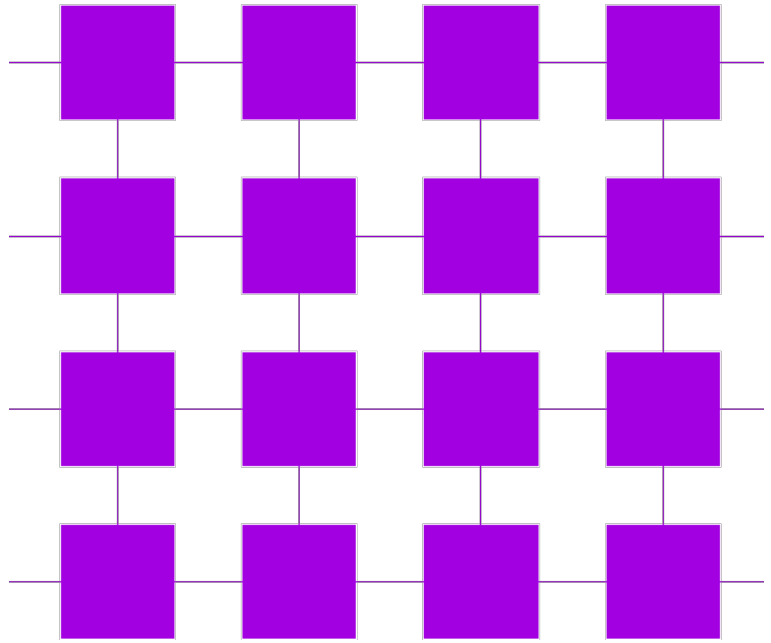


SST Ilyr Component

Mapper modules will handle the embedding of the application graph

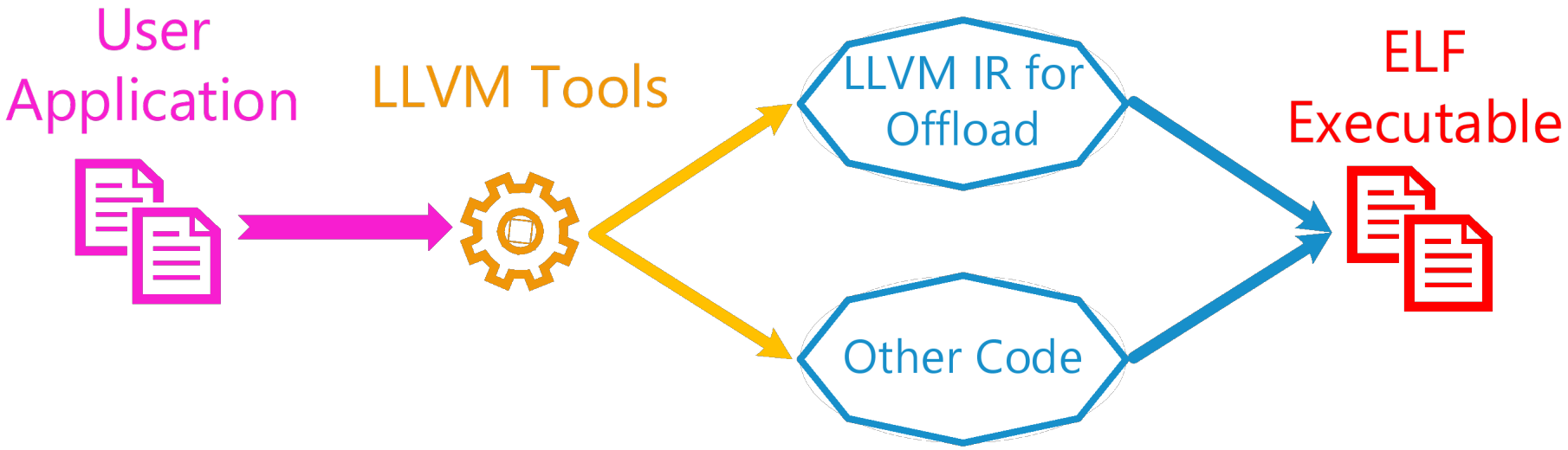


Mapper



Can be NP-complete (factorial complexity) [Conte, 2004]

LLVM and llyr Parsing



Work in progress…

Offload targets will be marked in the user application – currently studying how

- offload_myFunction()
- __attribute__ ((offload (device, 0))) myFunction()
- #pragma secret offload directive device(0)
  myFunction()

These functions/loops will be compiled with the user code but the LLVM IR will be embedded with them in the final executable

SST llyr Component Sample Configuration

```
llyr = sst.Component("dataflow0", "llyr.llyr")
llyr.addParams({
        "verbose": "1",
        "clock"  : "1GHz",
        "config" : "maeri_layout.cfg",
        "fp_lat" : "4",
        "int_lat": "1",
        "div_lat": "3",
        "mul_lat": "2",
        )}
```

Parameters

- clock: Operating frequency for entire device
- config: Input hardware layout
- xxx_lat: Number of cycles to complete the operation
- queue_depth: number of buffer entries
- ls_entries: number of L/S entries to process each cycle
- mapper: app_graph → hw_graph embedding

Additional parameters for standalone/testing

- application: application in LLVM IR
- mem_init: memory initialization file

Embedding Problem

Dataflow hardware consists of a set of PEs that may have fixed or programmable connectivity (MAERI, Softbrain, Plasticine, *etc.*)

A user application consists of a set of operations that map to one or more PE types (add, sub, mul, *etc.*)

Goal: Map user application operations onto set of hardware PEs

Graph matching

- Exact, Inexact
- Exponential complexity

Subgraph iso/monomorphism

- All possible subgraphs from reference graph
- NP-complete (factorial complexity) [Conte, 2004]

Integer Optimization

- Treat app_graph → hw_graph as optimization problem

Current Approach

Running standalone VF3 implementation from the University of Salerno

Have tested VF3 on multiple graphs (up to 32 nodes)

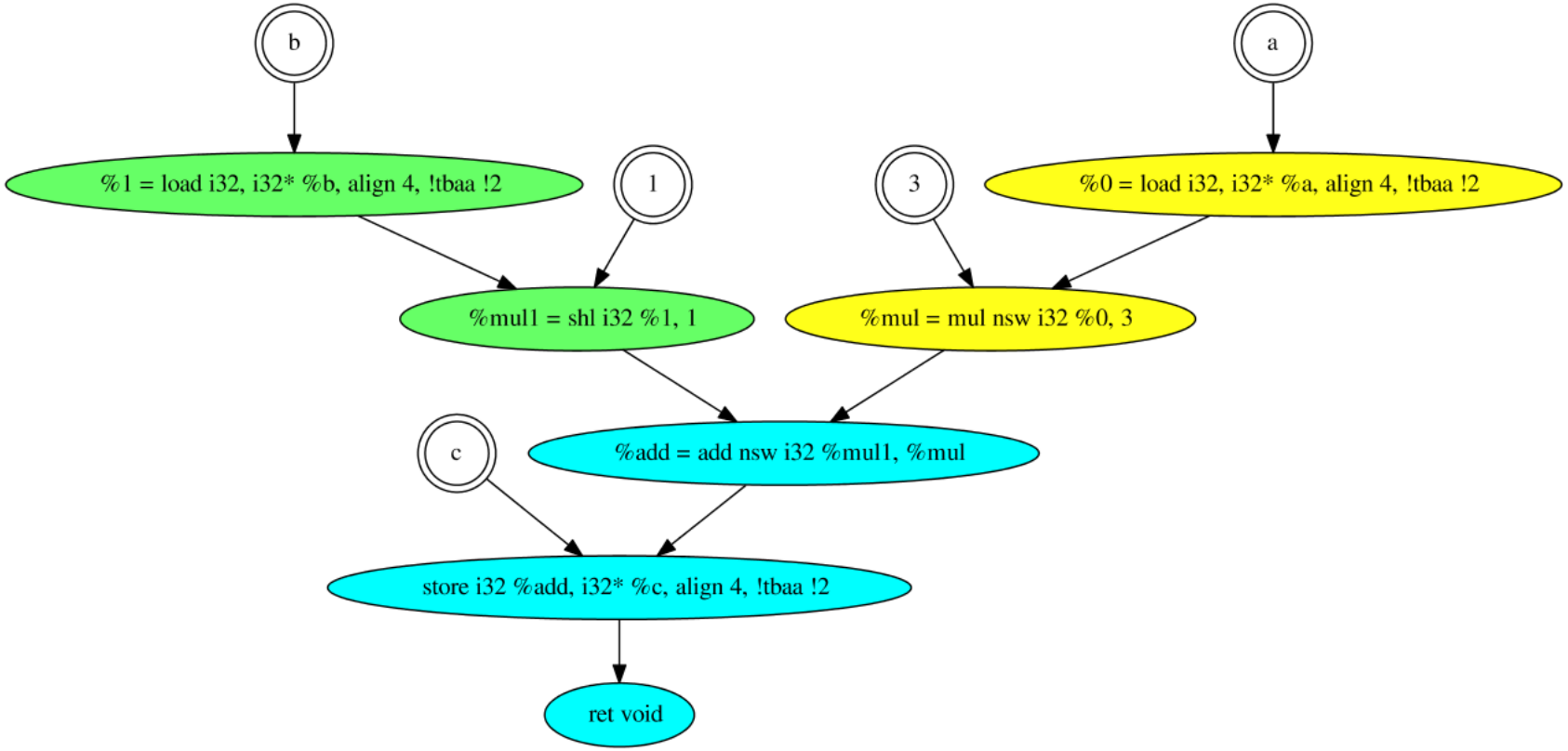
Current problem is that constraints can result in no valid mapping

- Need a way to bypass a node
- Will require some tweaking to be able to perform lookahead in the algorithm

MIP-Mapping discussions in progress

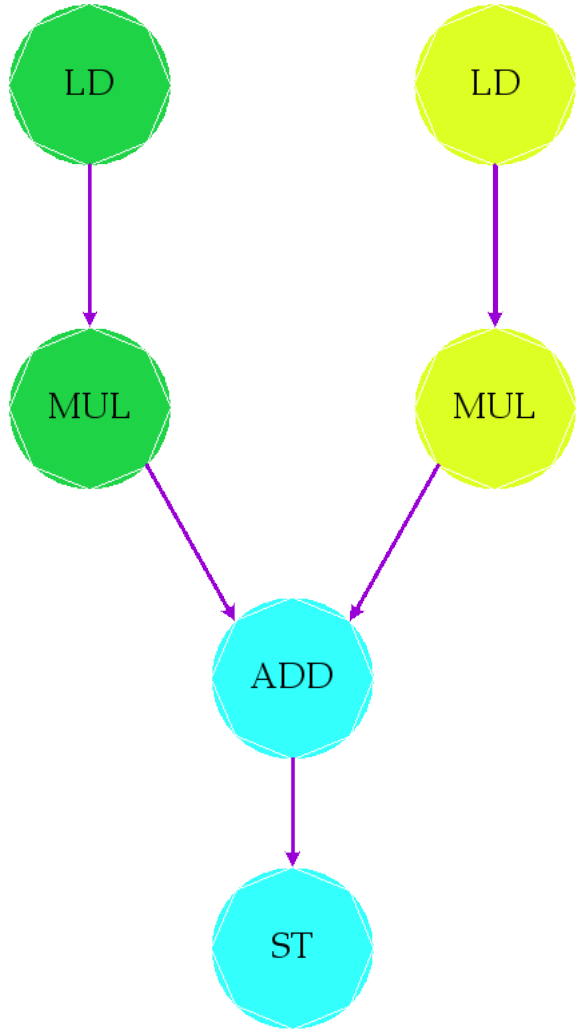
- Still working on expressing constraints and understanding design space

LLVM IR to AppGraph

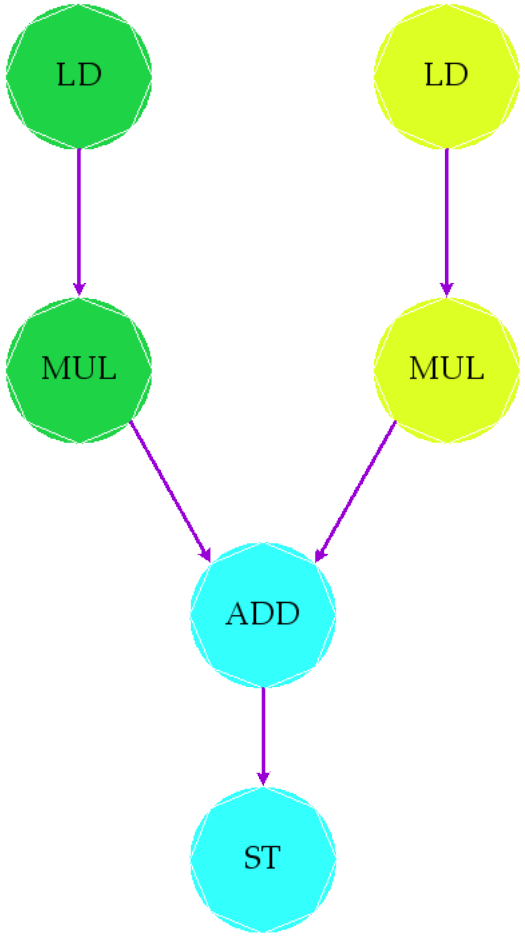


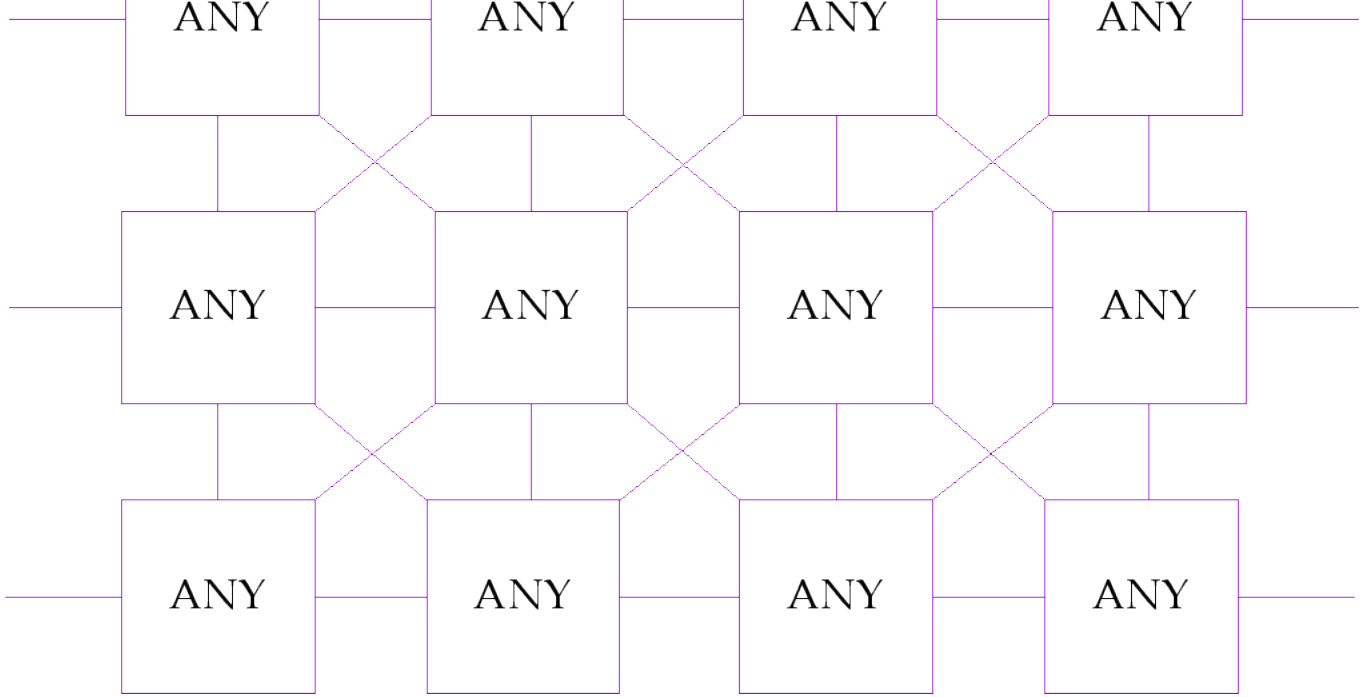
LLVM IR

Application Graph

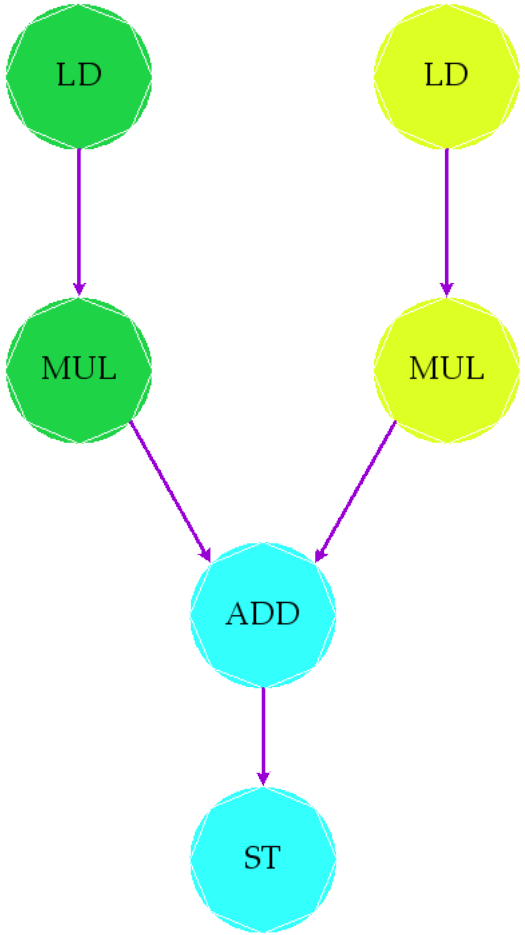


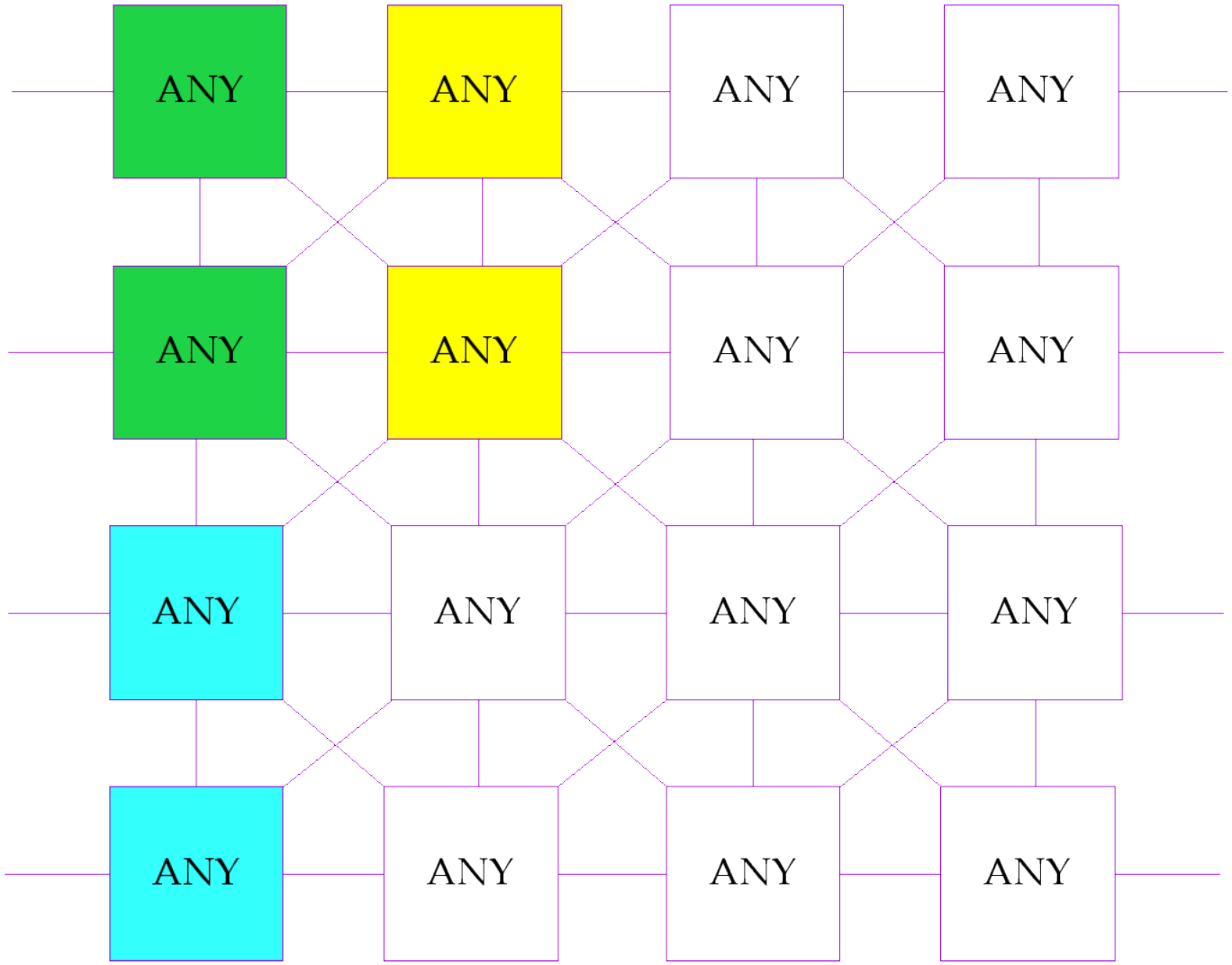
Node Attributes Don’t Matter





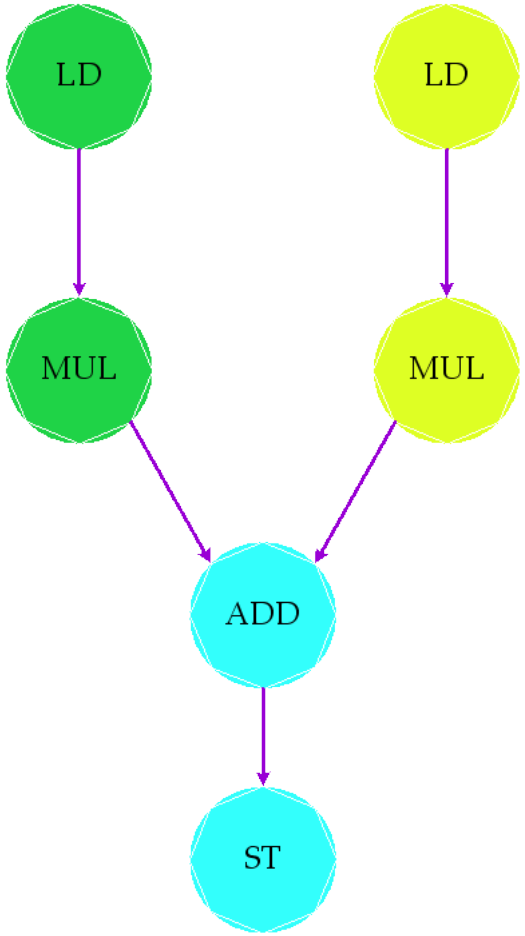
Node Attributes Don’t Matter

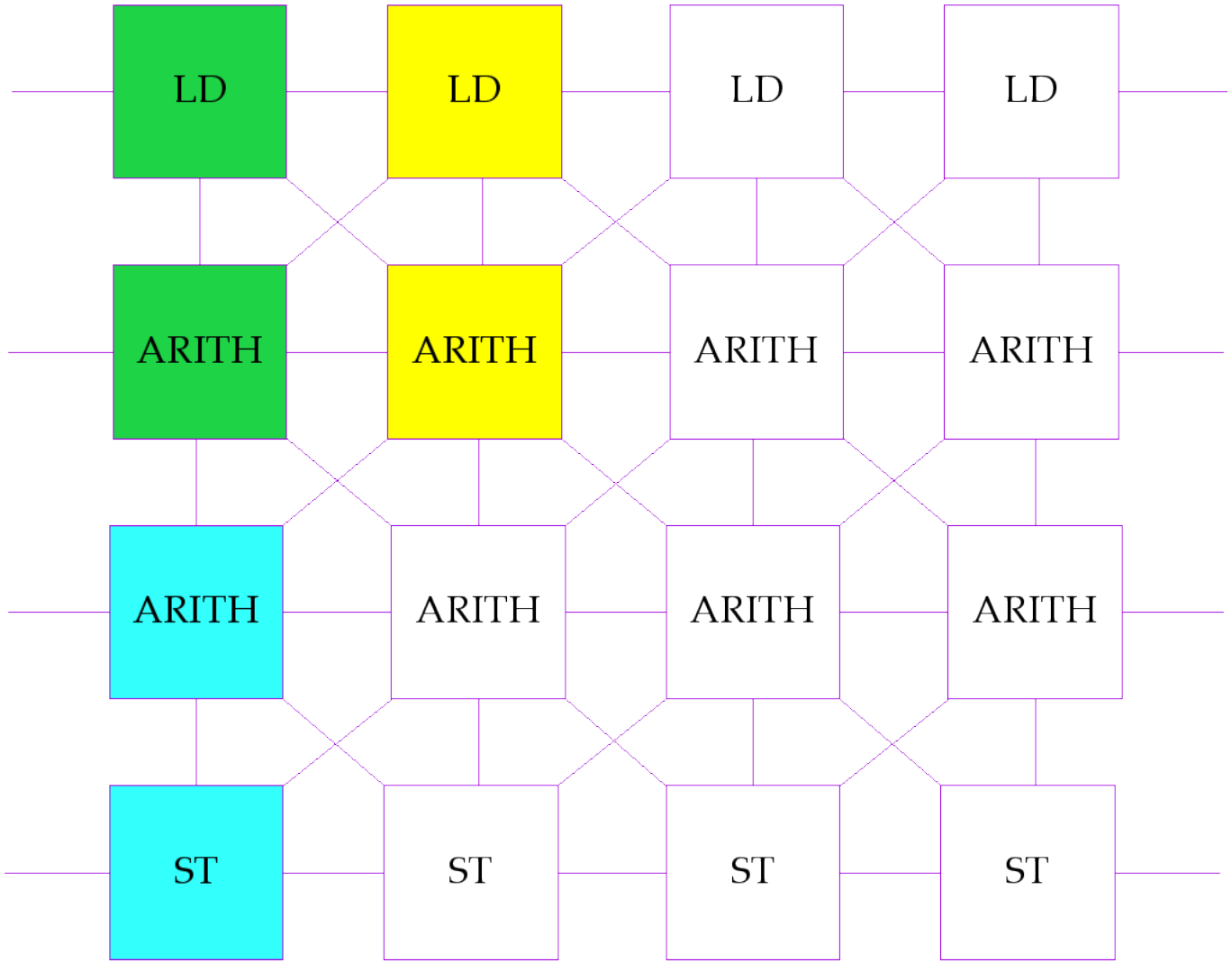




13720 possible mappings

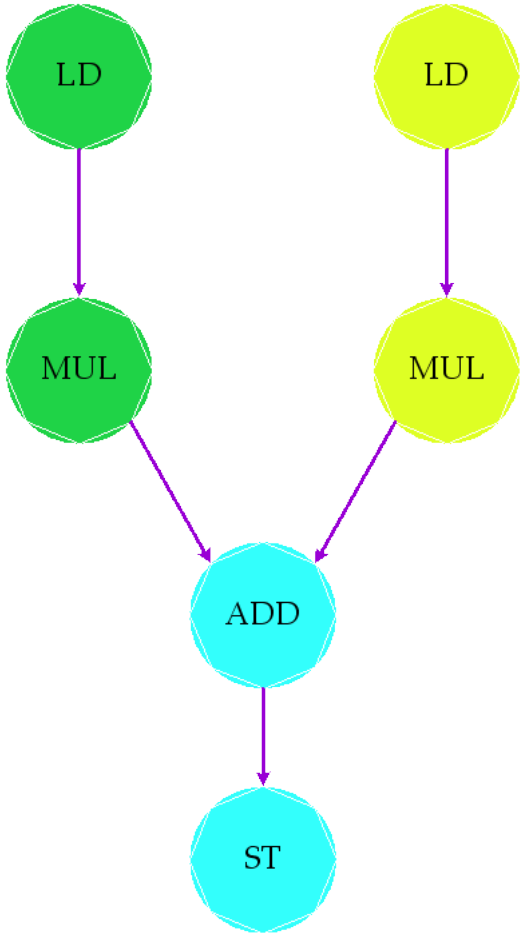
Node Attributes Matter (Mostly…)

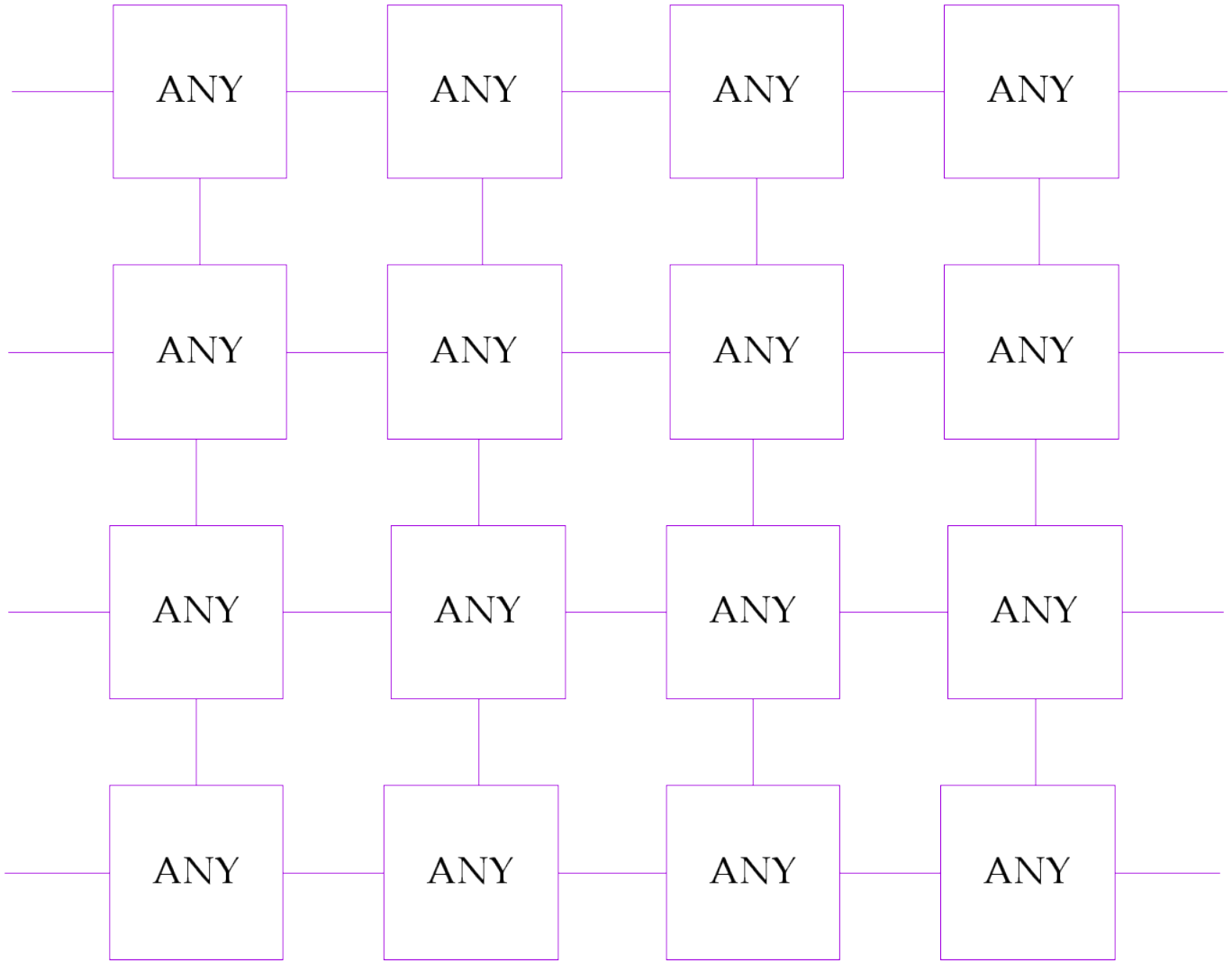




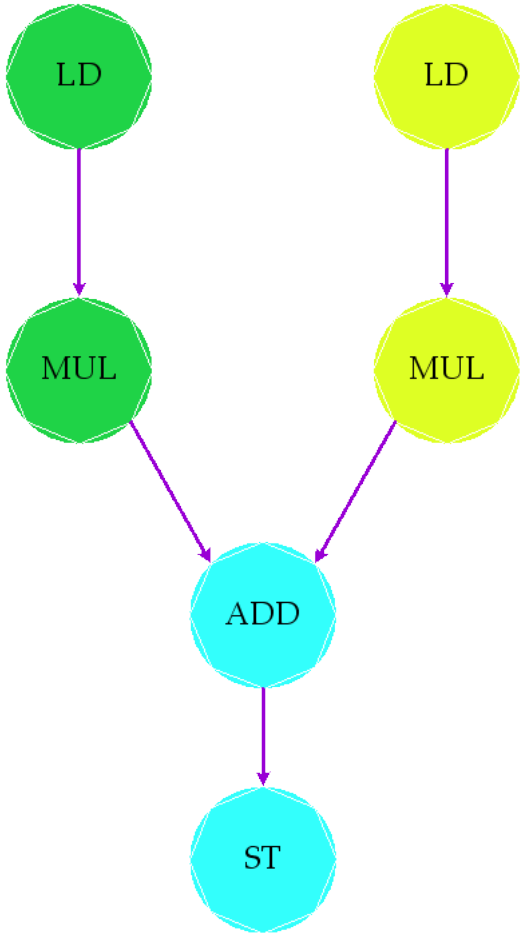
184 possible mappings

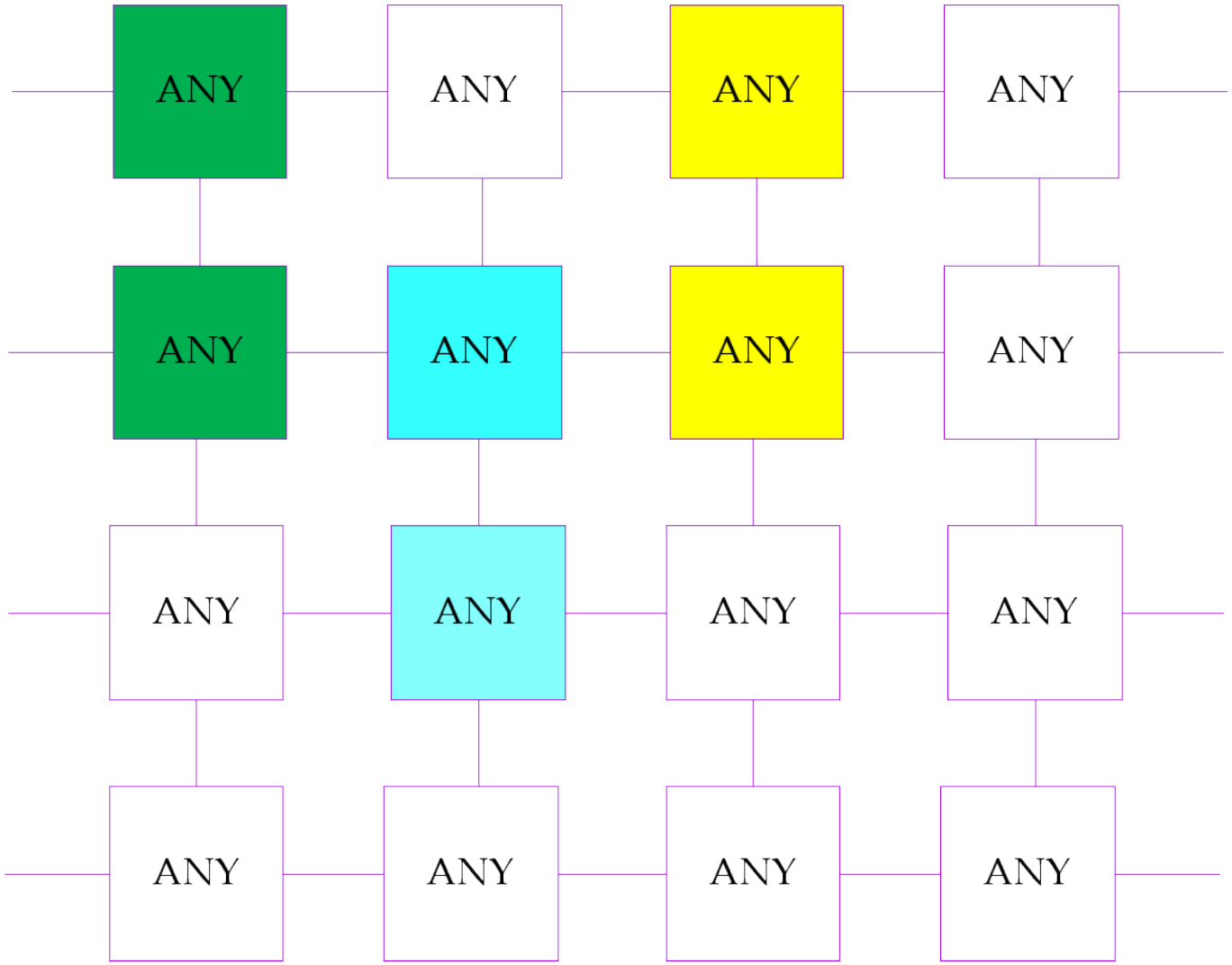
Node Attributes Don’t Matter





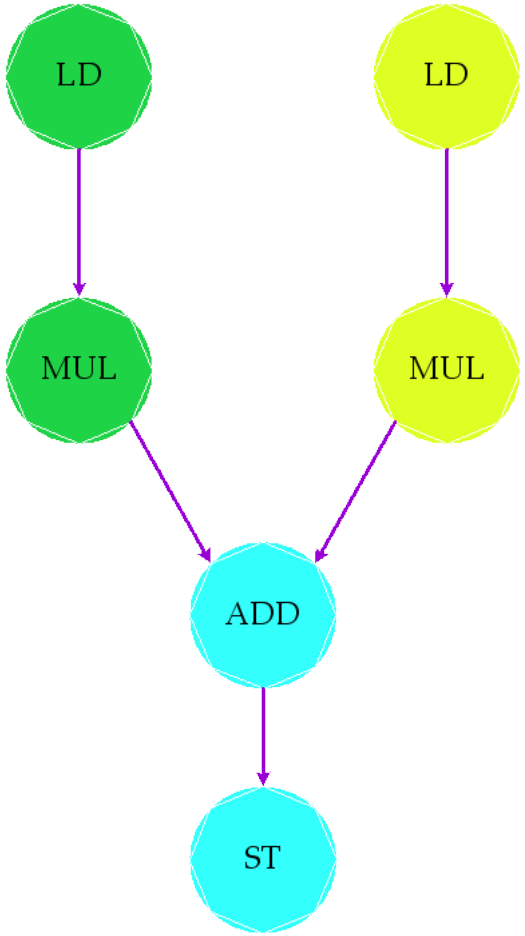
Node Attributes Don’t Matter

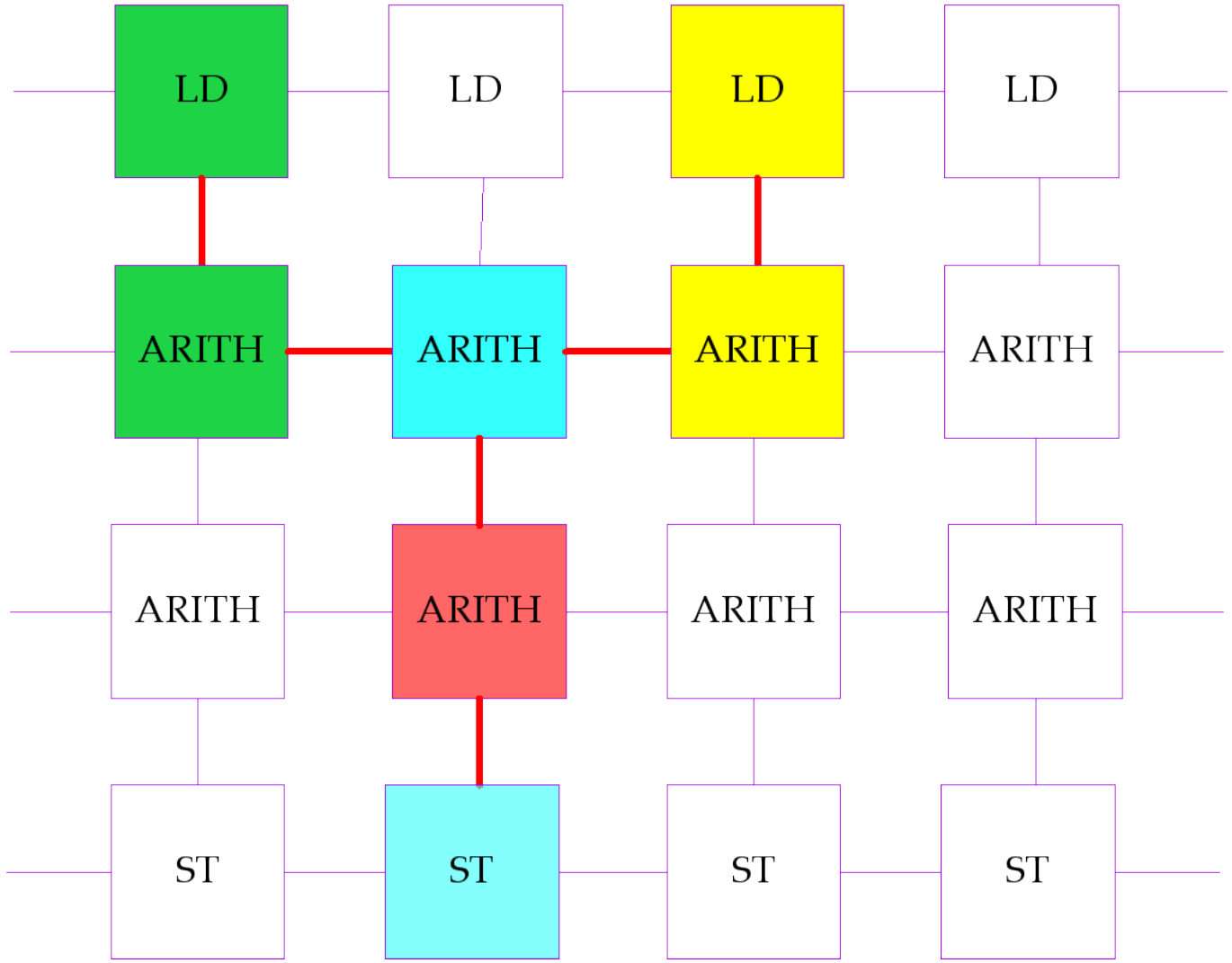




656 possible mappings

Node Attributes Matter (Mostly…)

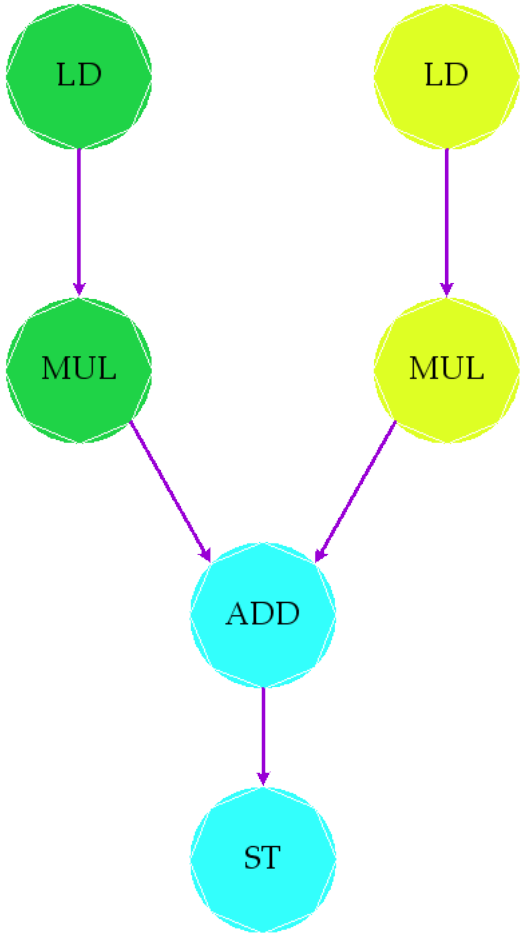


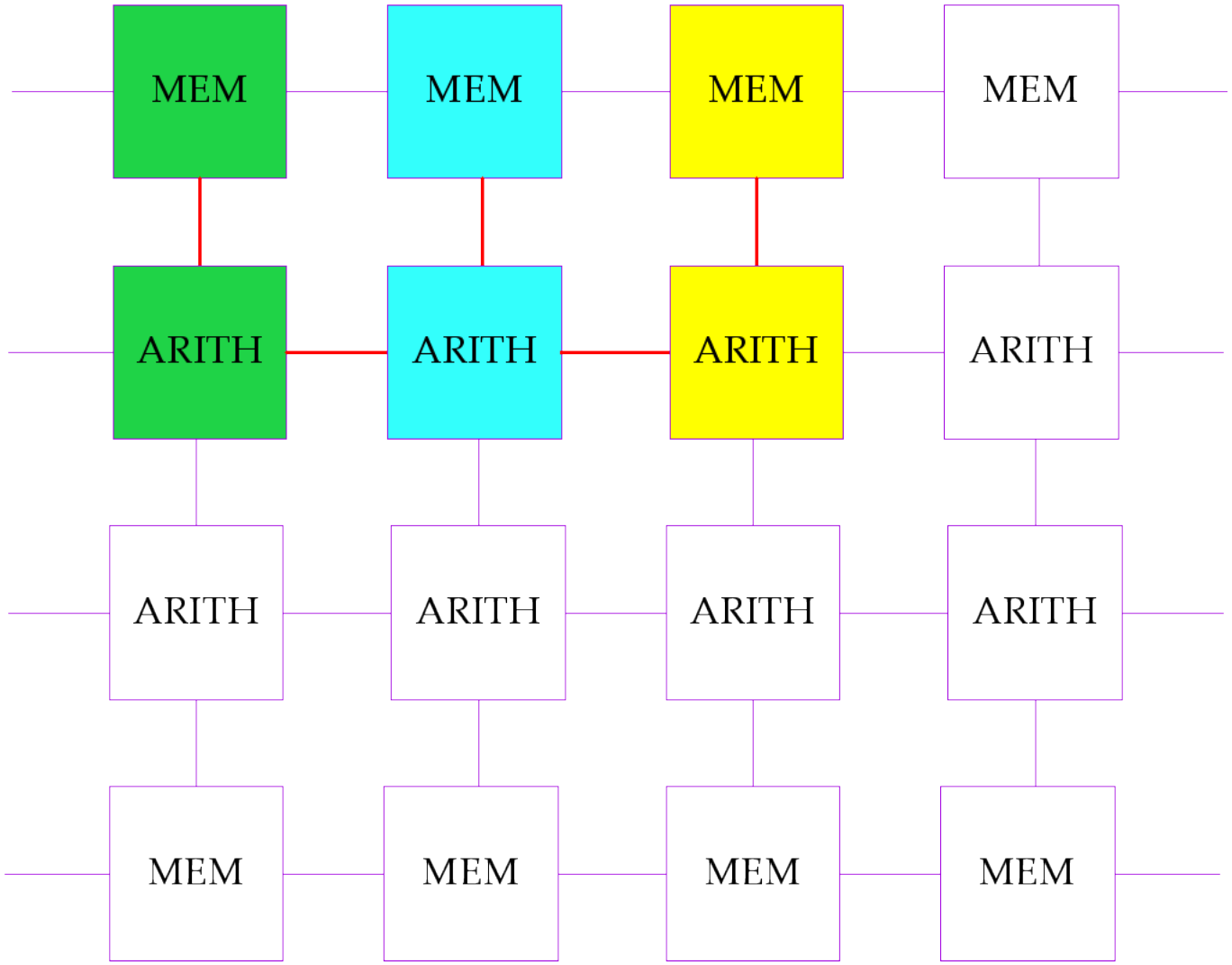


0 possible mappings

- No path from ADD → ST

Node Attributes Matter (Kind'a Sort'a)





32 possible mappings

Summary

SST dataflow component, llyr, is being tested as a standalone compute unit

- Successfully runs torus and grid hardware graphs with GEMM application graph
- Successfully runs RRAM crossbar model with GEMM

No automatic mapping of application to hardware yet

- VF3 shown to work using sample application and hardware graphs -- currently working on integration with SST as a ‘mapper’ subcomponent

Currently testing with integer types

- Data passed between PEs as raw bitstream so double/float requires some conversion


QUESTIONS

BACKUP SLIDES