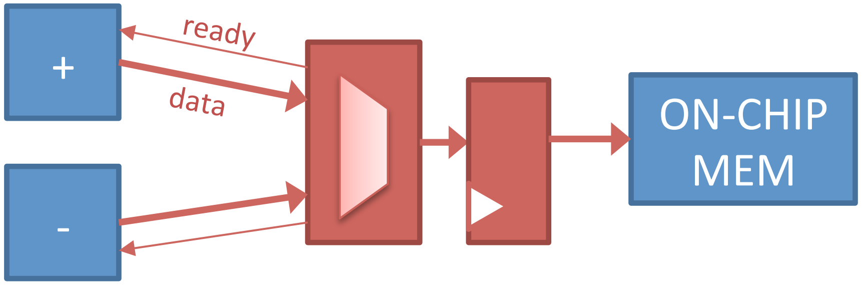
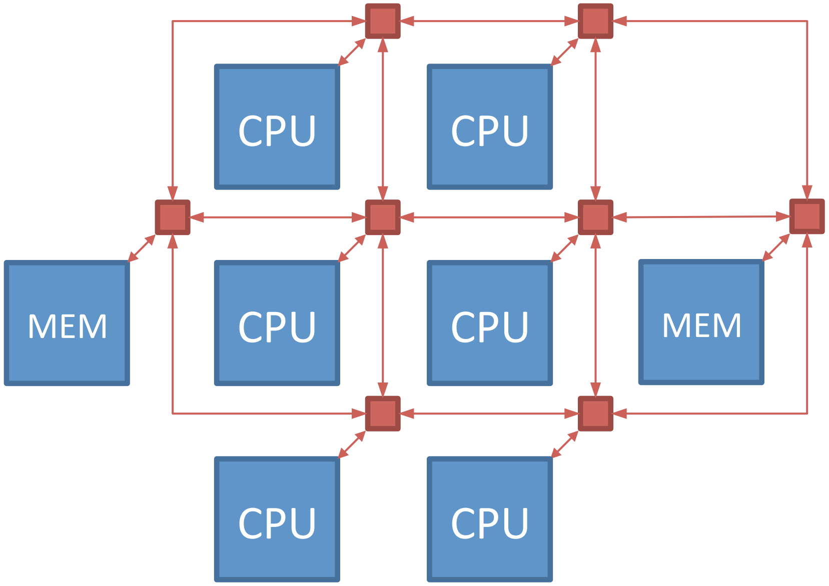
Fine-Grained Interconnect Synthesis

Alex Rodionov, David Biancolin, Jonathan Rose
Department of Electrical & Computer Engineering
University of Toronto

Making Hardware Design Easier

- Interconnect: important part of hardware design
- Allows functional units to communicate



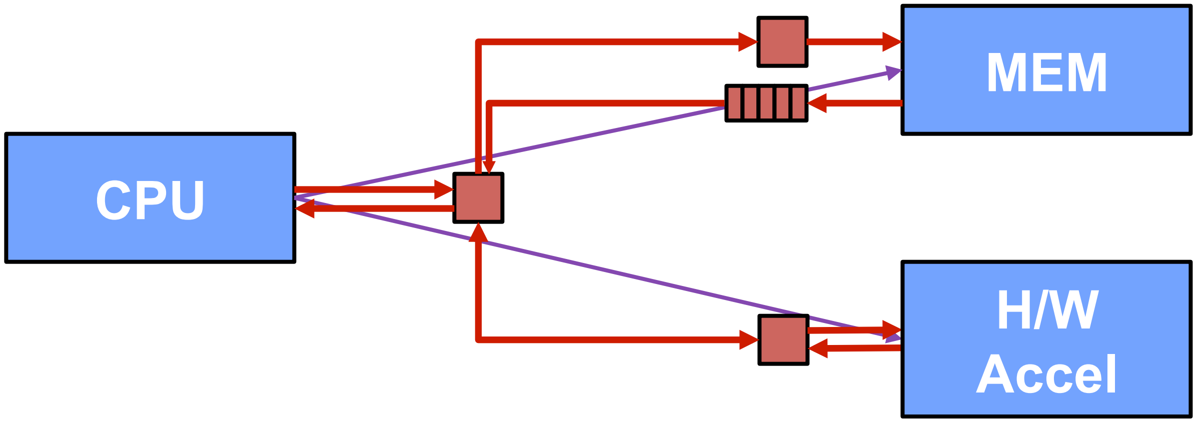


- Observation: it's difficult to design properly

➢ **Our focus: automatic design and synthesis of interconnect**

Existing Tools: Coarse-Grained Design

- Commercial: Altera Qsys, Xilinx IPI
- Academic: Networks-on-Chip (ex: CONNECT)
- Generally connect big things: processors and IP blocks

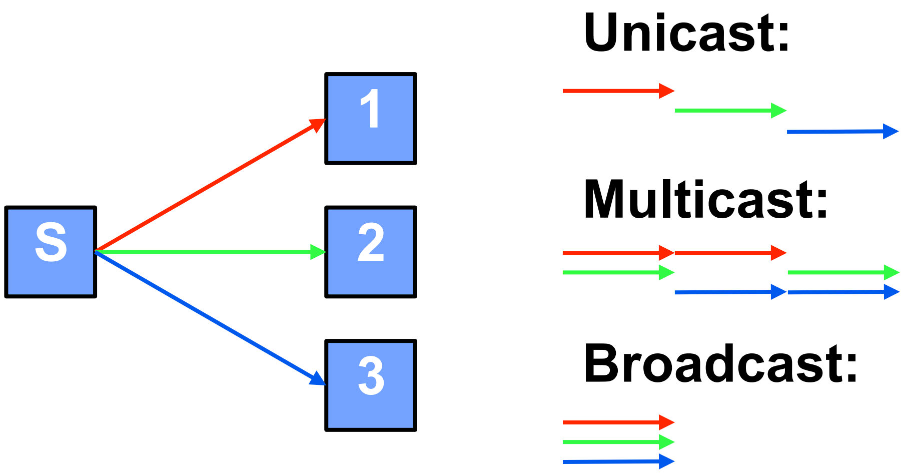


- Interfaces: memory-mapped, streaming
- Variable-latency-tolerant → “plug and play”
- What about *inside* IP blocks?

Fine-Grained Interconnect Design

- smaller functional modules
- coordinate data transfer more closely
  - depend on *specific* interconnect latencies
  - **Area is at a premium** in fine-grained systems
- unicast, but also broadcast/multicast

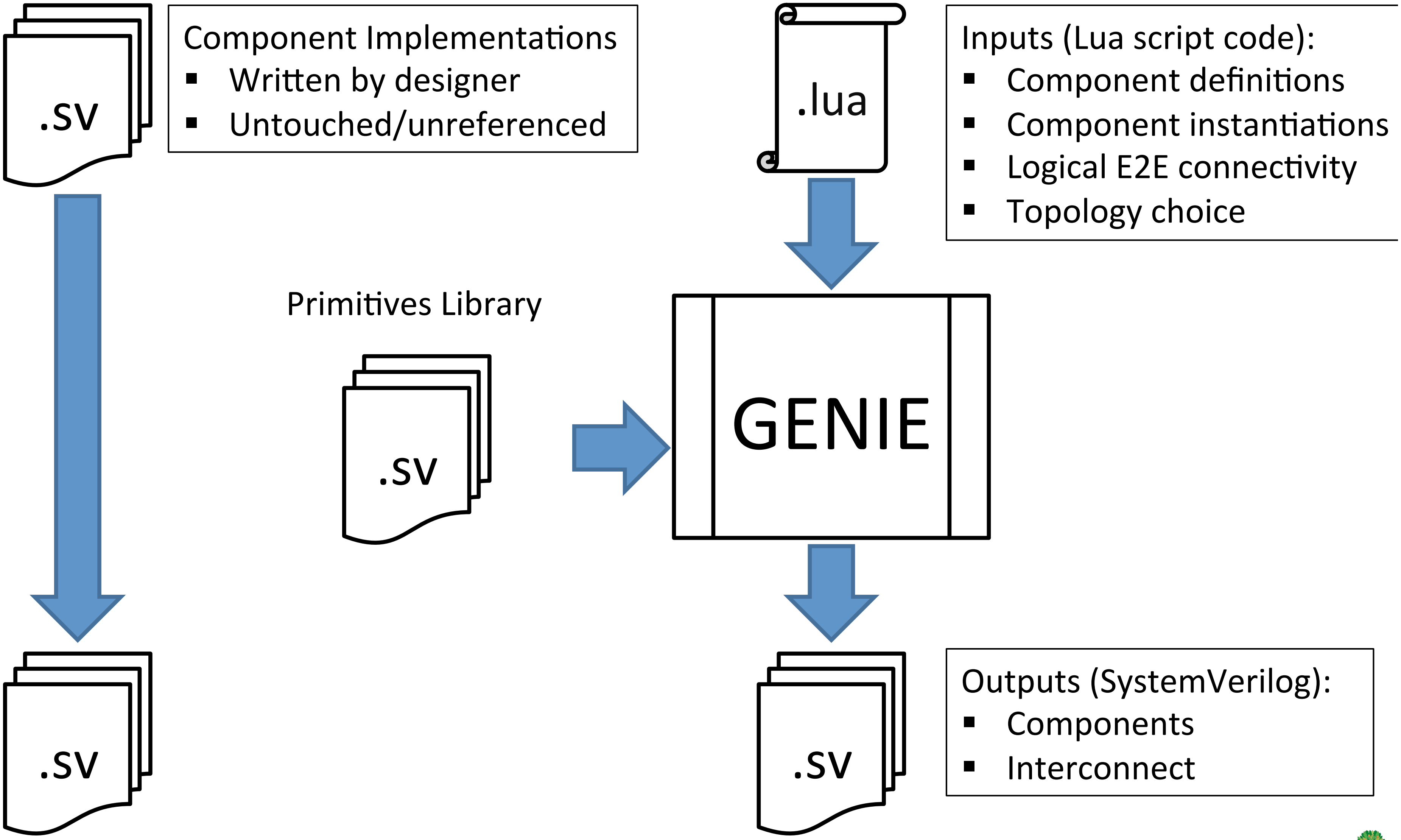
➢ Existing tools aren't good at this



GENIE: Generic Interconnect Engine

- Grand vision wants both: Fine & Coarse Grain
- Input:
  - functional modules
  - logical connectivity
  - **performance specifications**
- Output:
  - instantiated functional modules
  - **Fine or coarse** interconnect **optimized** to meet constraints
- Focus of this work: automatic generation of **fine-grained** interconnect and some optimization

Inputs/Outputs



Component Implementations

- Written by designer
- Untouched/unreferenced

Inputs (Lua script code):

- Component definitions
- Component instantiations
- Logical E2E connectivity
- Topology choice

Primitives Library

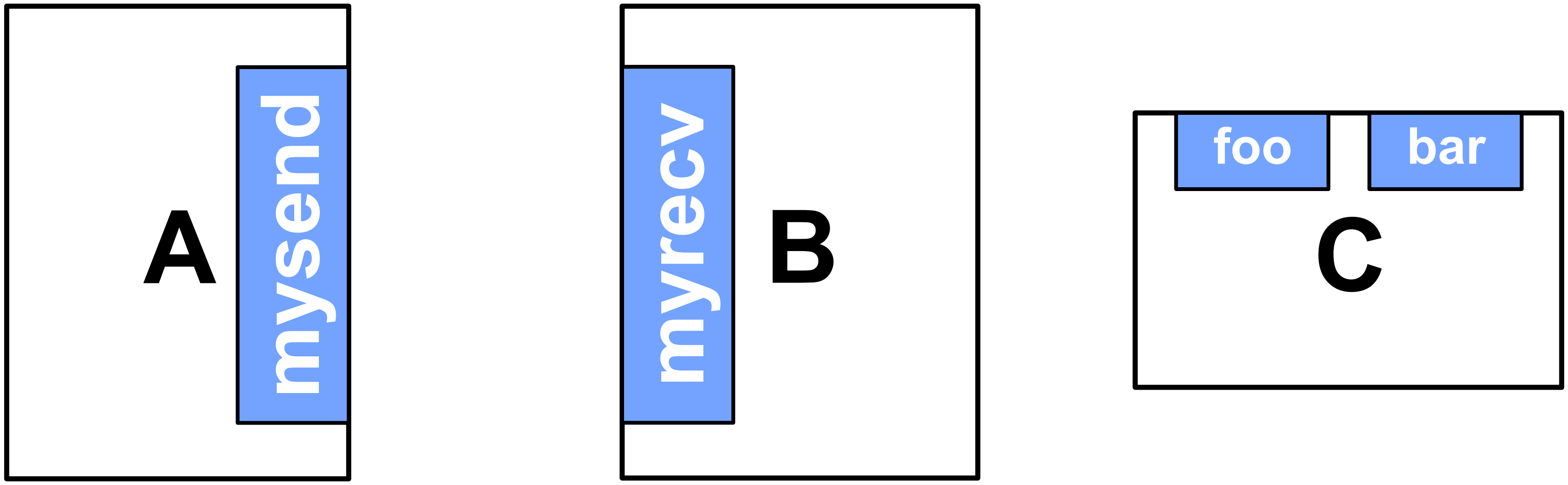
Outputs (SystemVerilog):

- Components
- Interconnect

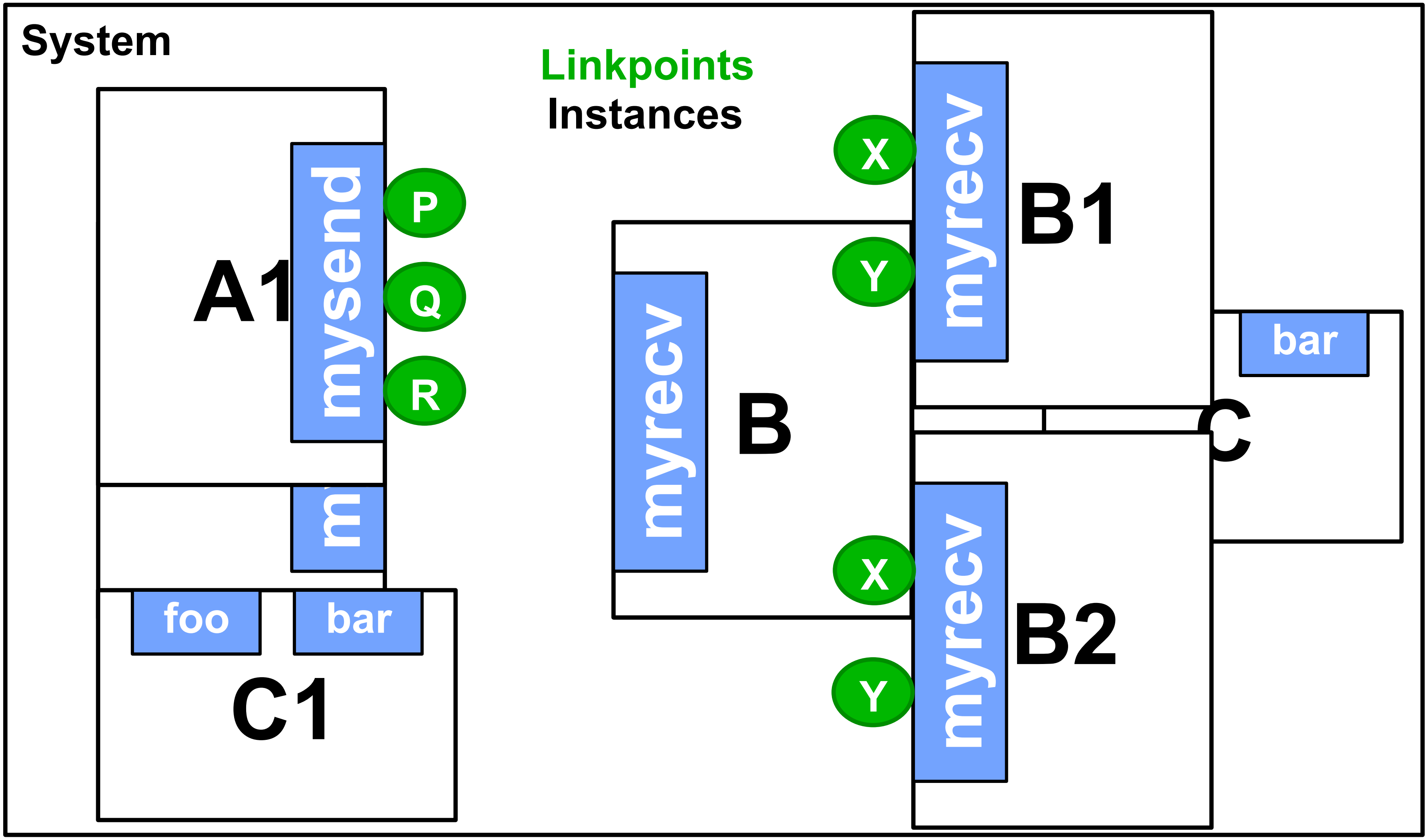
GENIE Flow: Input Specification

Interfaces

Components

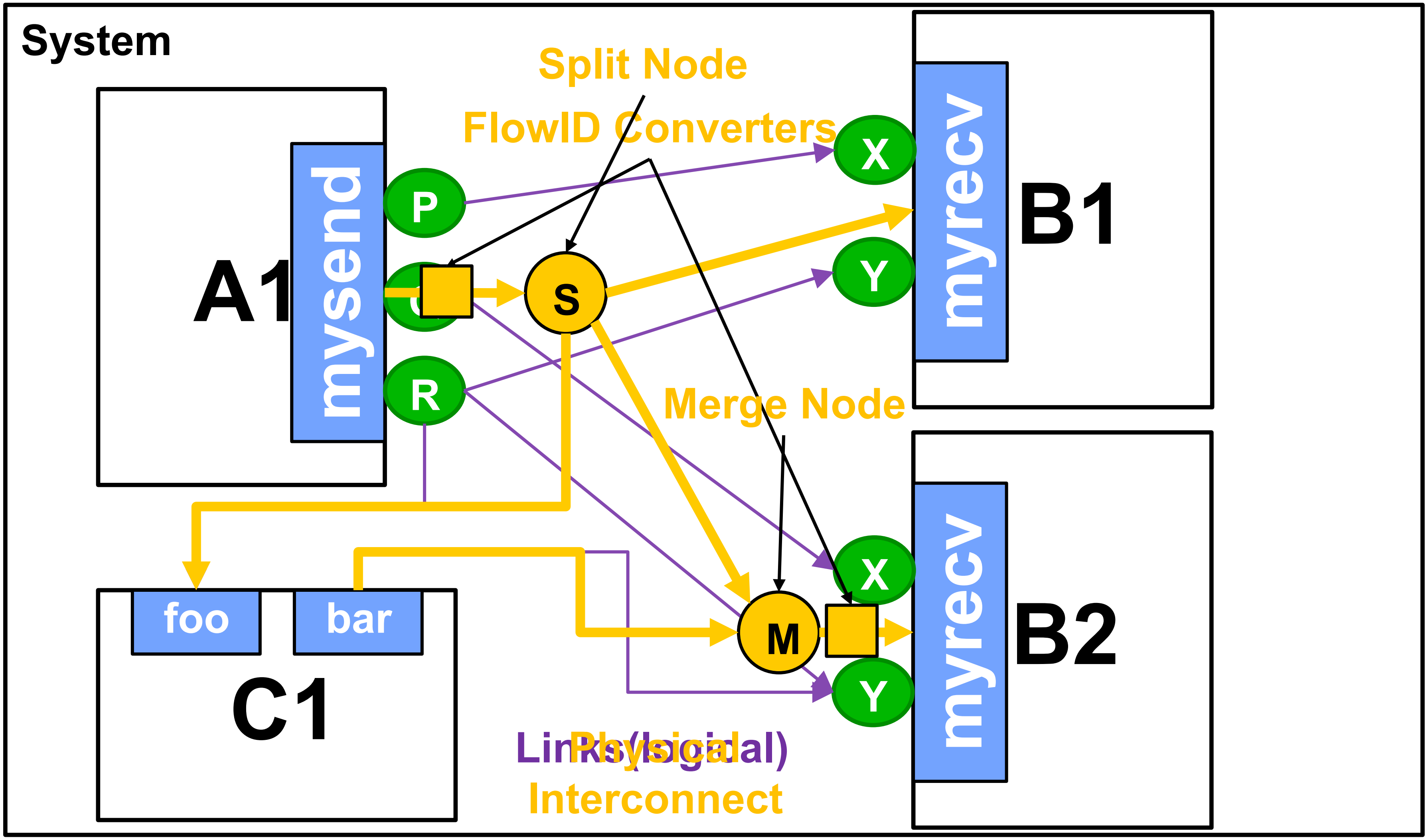


GENIE Flow: Input Specification

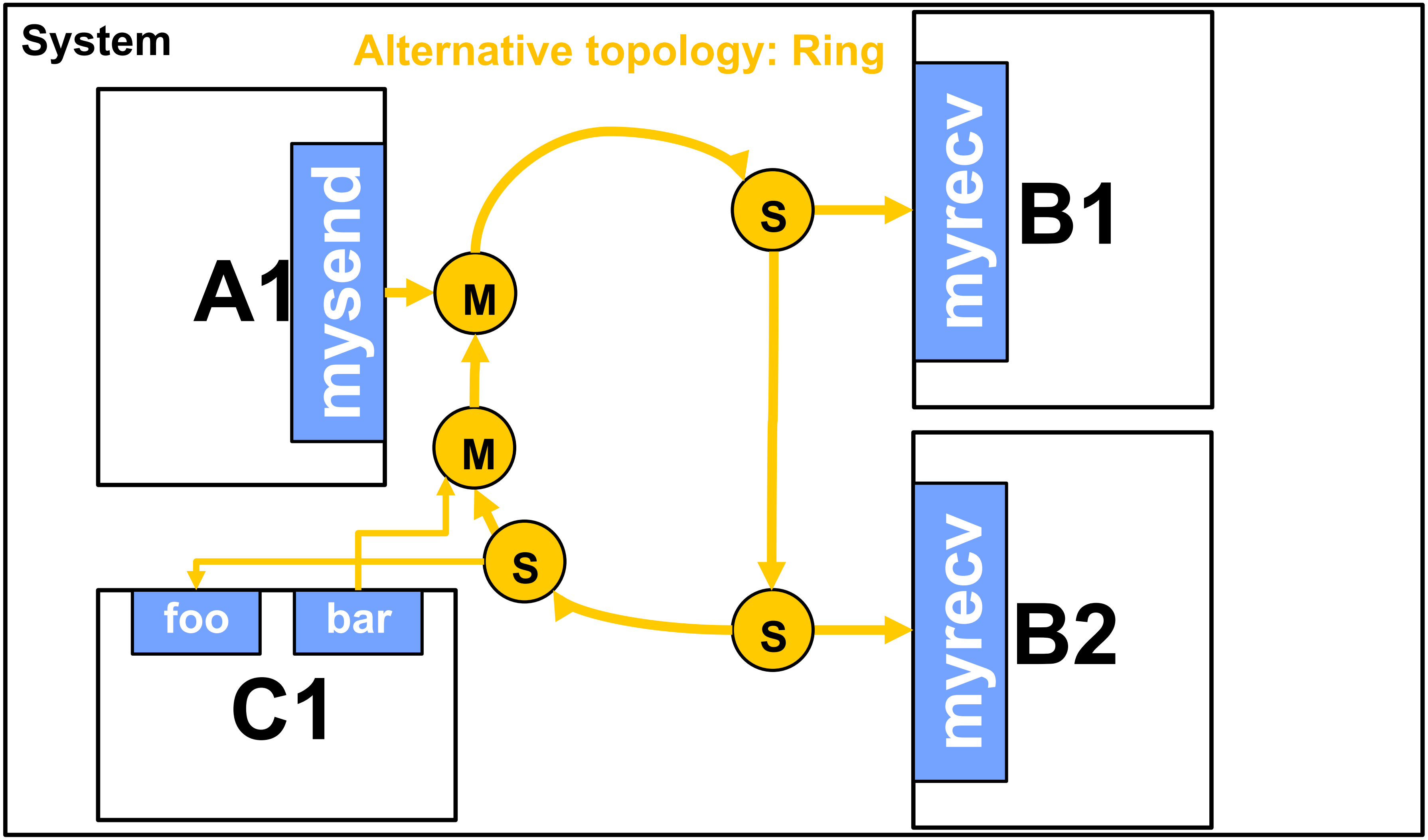


GENIE Flow: Logical Connectivity

GENIE Flow: Synthesizing Interconnect

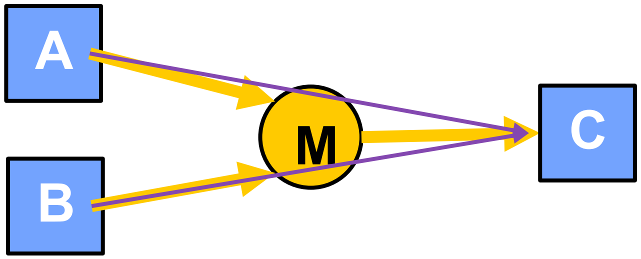


GENIE Flow: Design Space Exploration

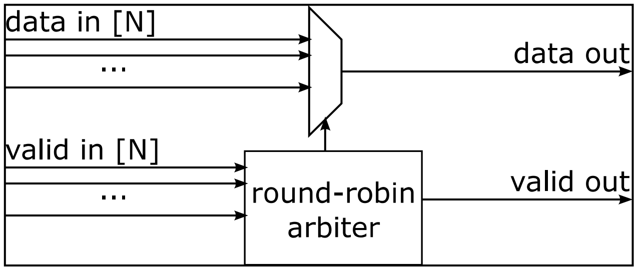


Optimization Features

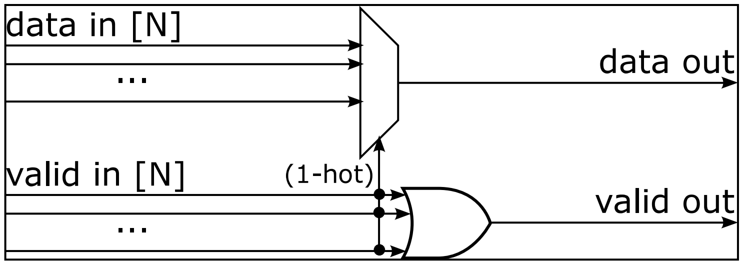
Removing Unnecessary Arbitration



- Designer: “A and B will never try to send to C simultaneously”
- No competition → No arbitration → Simpler circuit



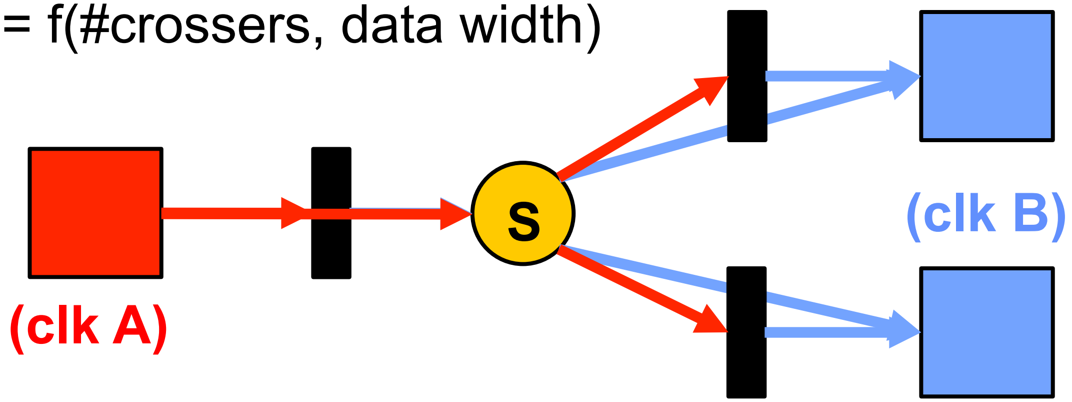
Merge Node



Simplified Merge Node

Smart Clock Domain Crossing

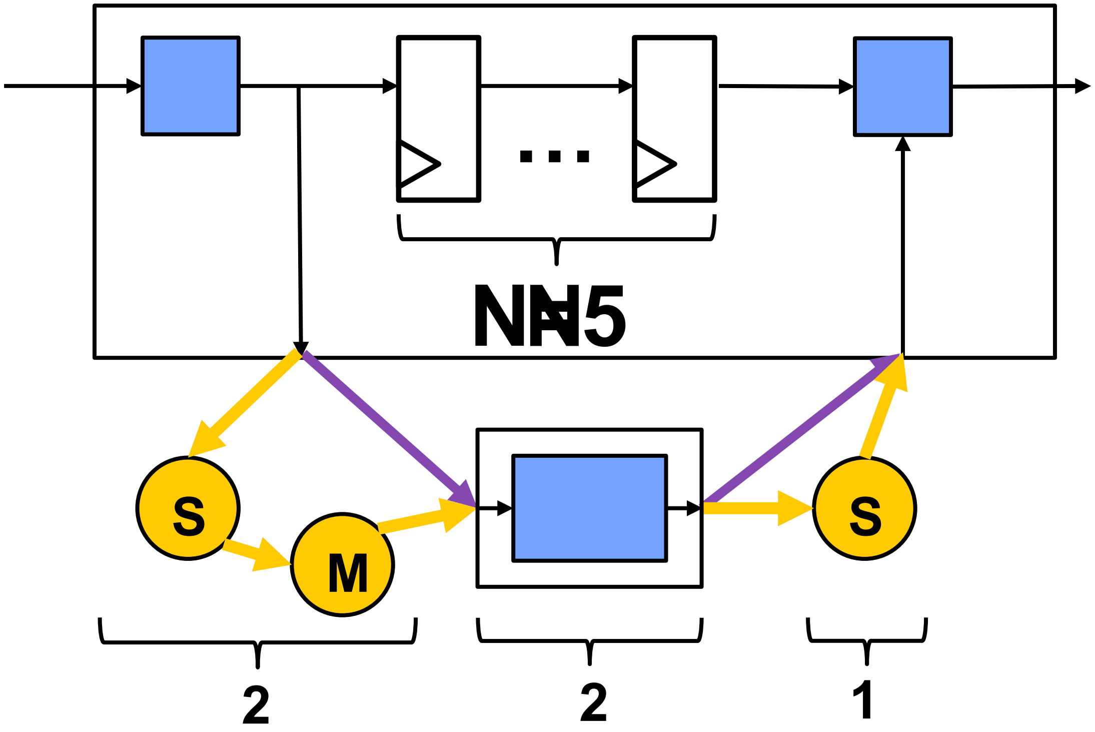
- Applications can use more than one clock domain
  - Need to insert clock crossers where domains meet
  - Typically FIFOs
- There can be many choices of where to put the crossers
  - Some choices more expensive than others
  - Cost = f(#crossers, data width)



➢ Optimization problem: given any topology, where are the crossing points?

Latency Introspection

- Recall: Fine-grained design scenario
- Area/complexity at a premium: don't want overhead of flow control or latency insensitivity
- Component must know *exact* interconnect latency



Results

Measurement and Comparison

- Recall Goals:
  - Easy to design hardware
  - Produce interconnect with good performance, low area

- Will Compare GENIE with:
  1. **Manual** hand-optimized RTL
     - Human engineer
  2. Altera Qsys
     - Commercial Coarse-Grain System Interconnect Tool

Metrics

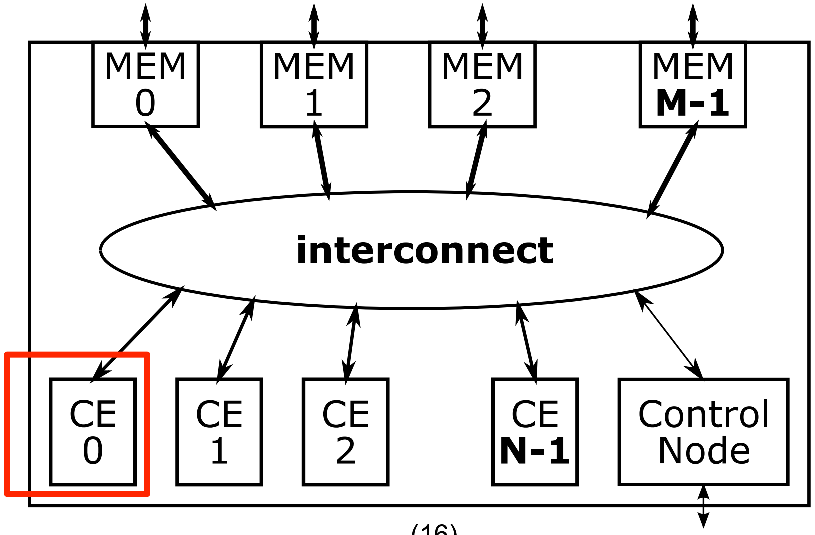
1. **Ease-of-use:** source code line counts
   - Manual: all Verilog
   - Qsys: Verilog for functional modules + TCL for spec
   - GENIE: Verilog for functional modules + Lua for spec

2. **Area**

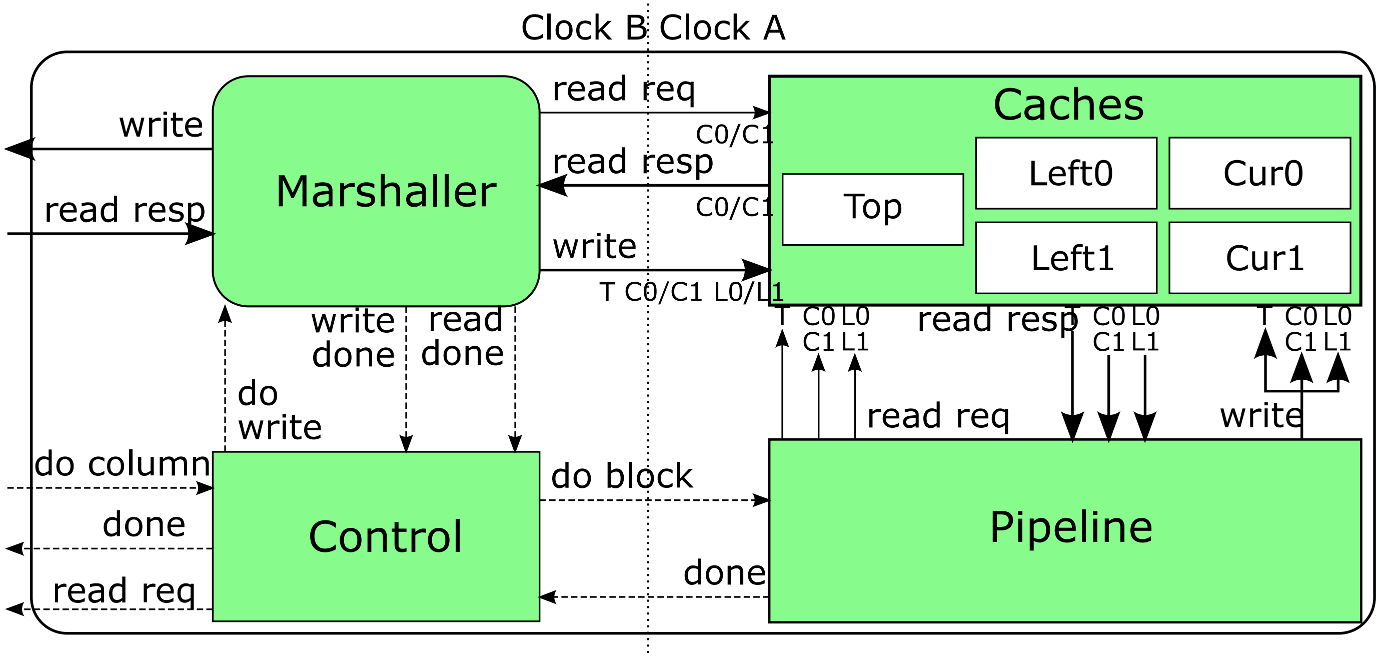
3. **Clock Frequency**

Design Example

- Application: Blocked Matrix LU Decomposition
  - Given: matrix **A**
  - Find: lower/upper-triangular matrices **L**,**U**, s.t. **LU=A**
- Parallelized among N Compute Elements (CE)
- **Fine-grained system: CE interior**



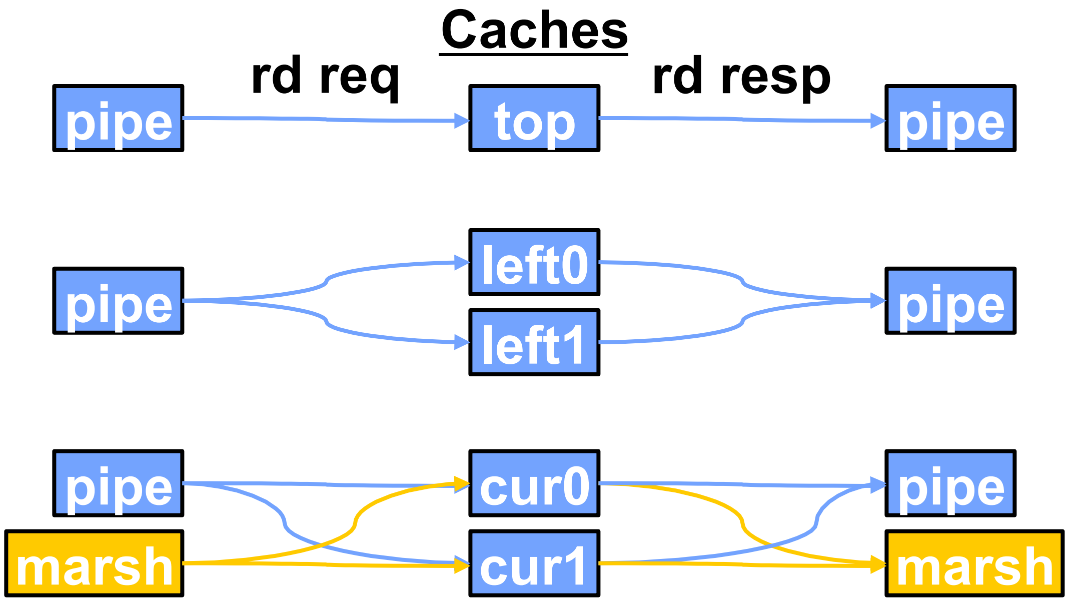
CE Architecture



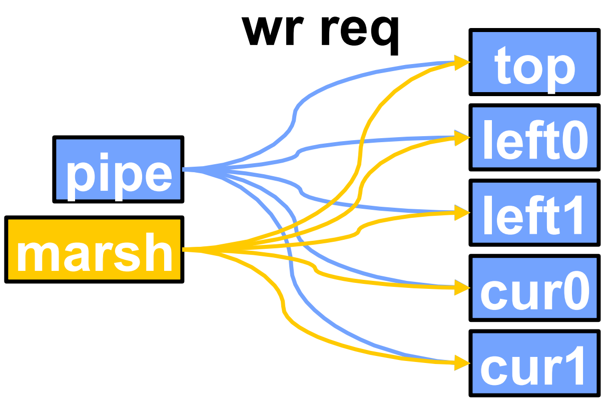
- **5 caches**: 2x(2 double-buffered) + 1
- **Data Marshaller**: fills/empties caches
- **Compute pipeline**: operates on data in caches
- **Control block**

Connections To and From Caches

Read Paths



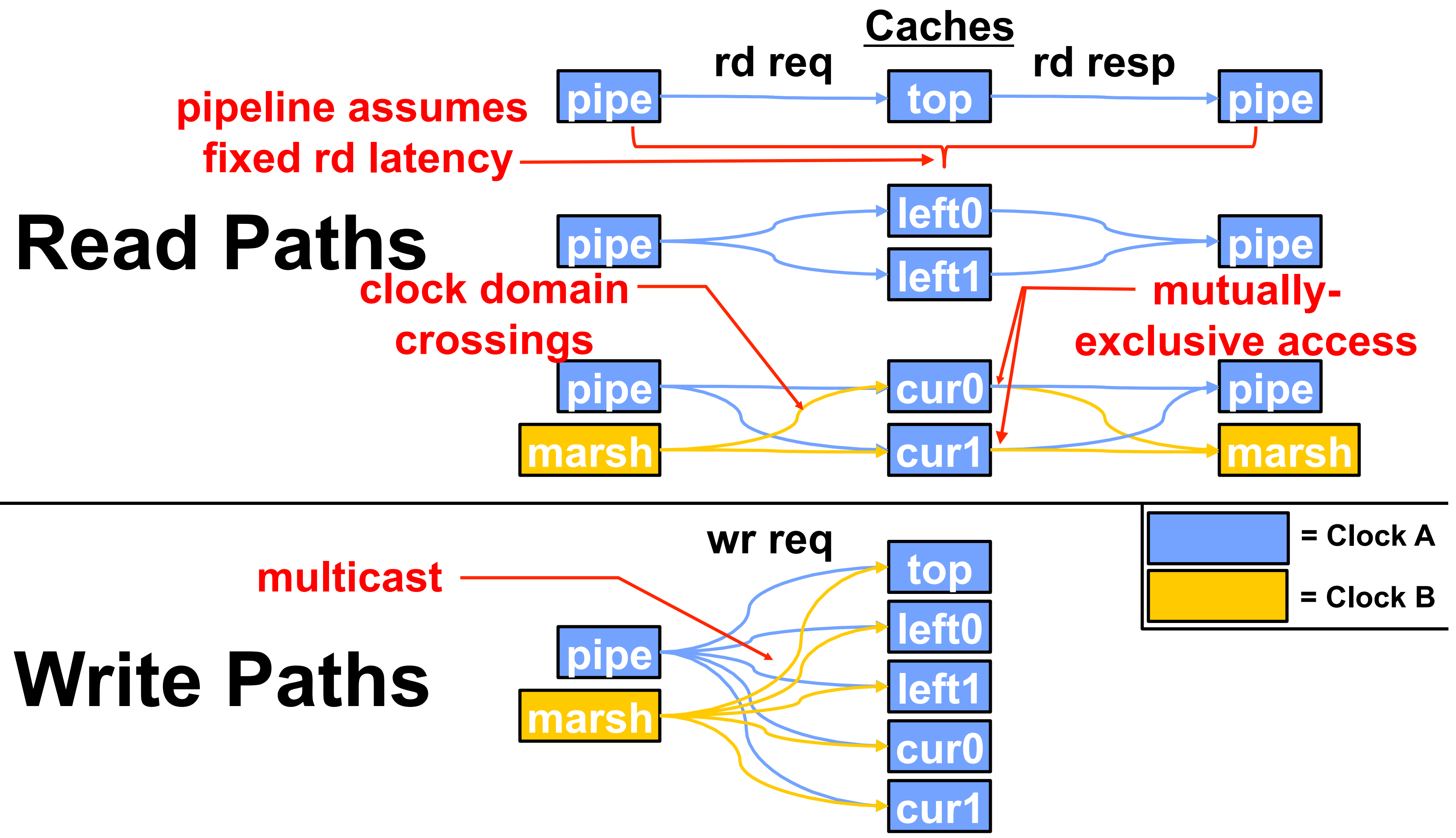
Write Paths



🟦 = Clock A

🟨 = Clock B

Connections To and From Caches



Caches

rd req

rd resp

pipe

top

pipe

pipeline assumes fixed rd latency

Read Paths

pipe

left0

left1

pipe

clock domain crossings

mutually-exclusive access

pipe

marsh

cur0

cur1

pipe

marsh

🟦 = Clock A

🟨 = Clock B

wr req

multicast

Write Paths

pipe

marsh

top

left0

left1

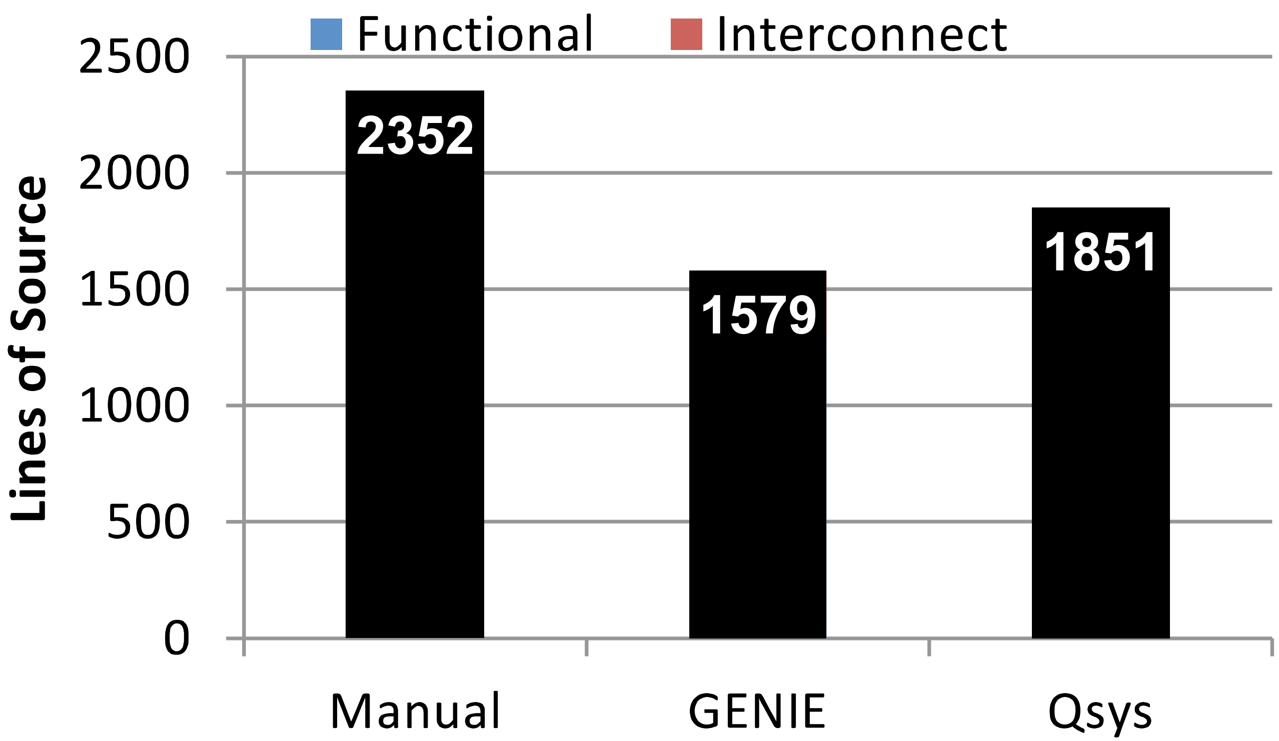
cur0

cur1

Experimental Setup

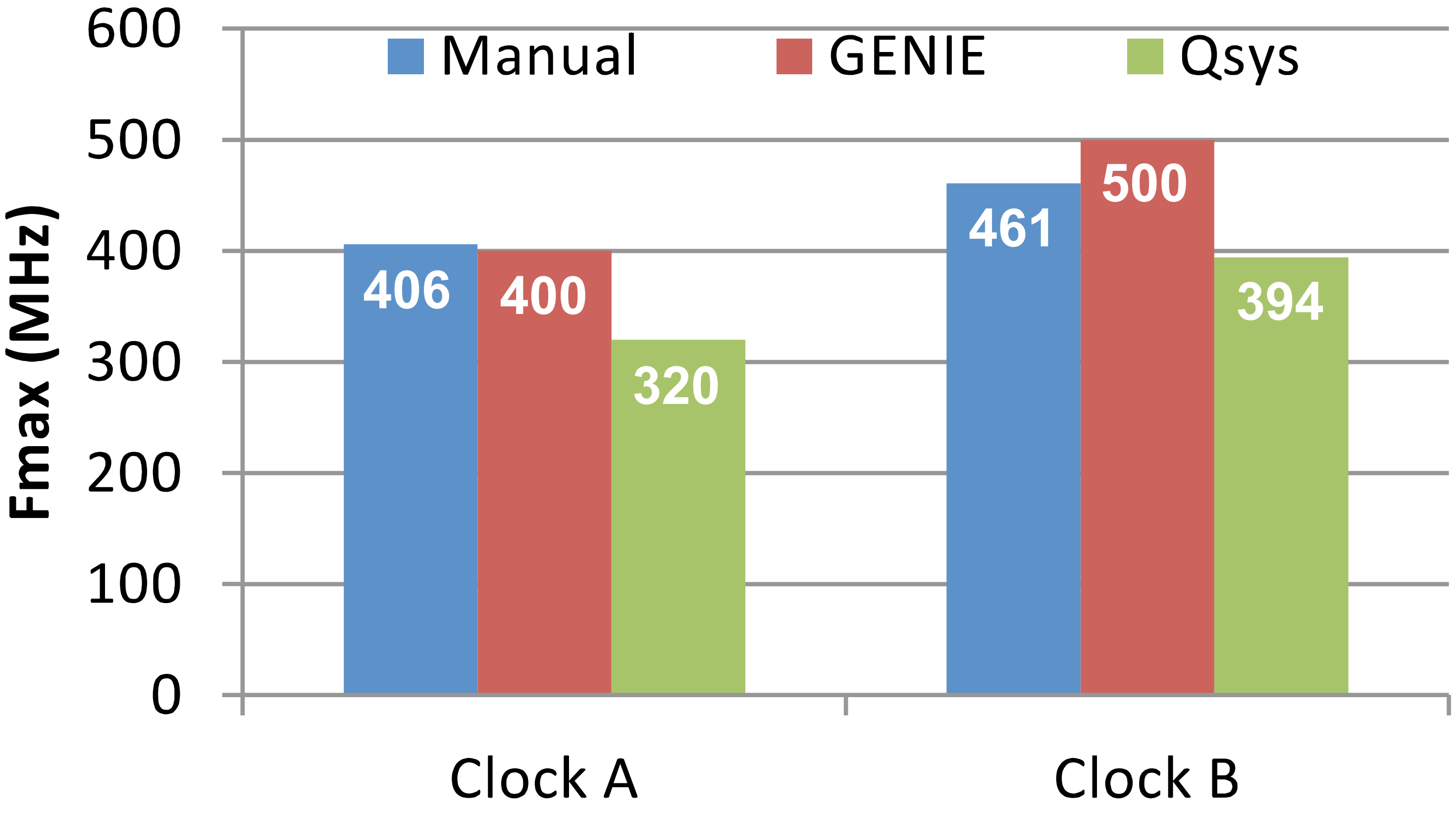
- Create three variants of the CE design
  1. Manual
  2. Altera Qsys
     - Avalon-MM for cache links
     - Avalon-ST for other, misc. point-to-point links
  3. GENIE
- Compile with Quartus 14.0 for Stratix V (6 seeds)
- Measure
  - Source code line counts
  - Area
  - Clock frequency

Results: Lines of Code



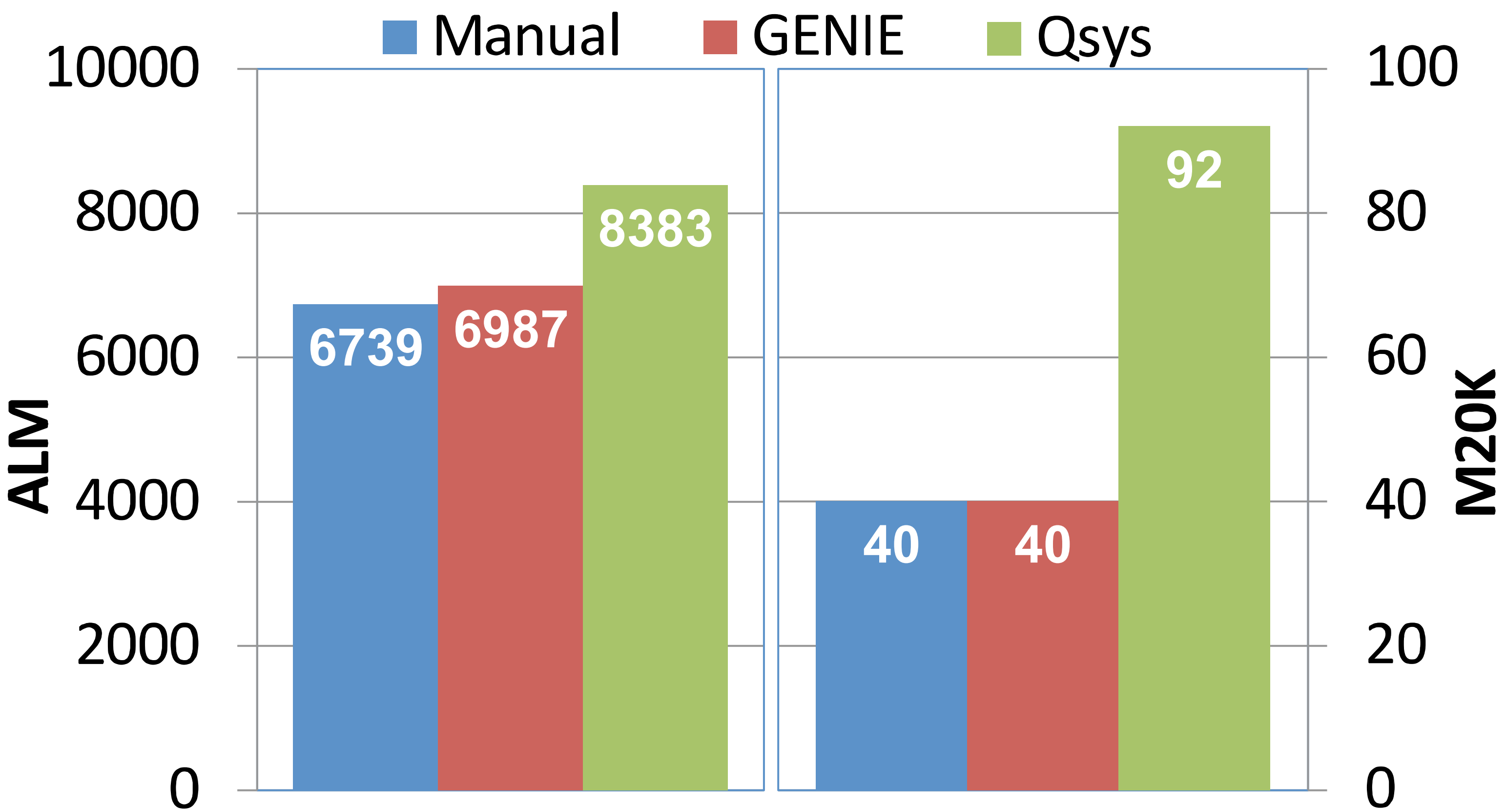
	Total	Functional	Interconnect
GENIE vs. Manual	-33%	-2.6%	-72%
GENIE vs. Qsys	-15%	-8.6%	-34%

Results: Clock Frequencies



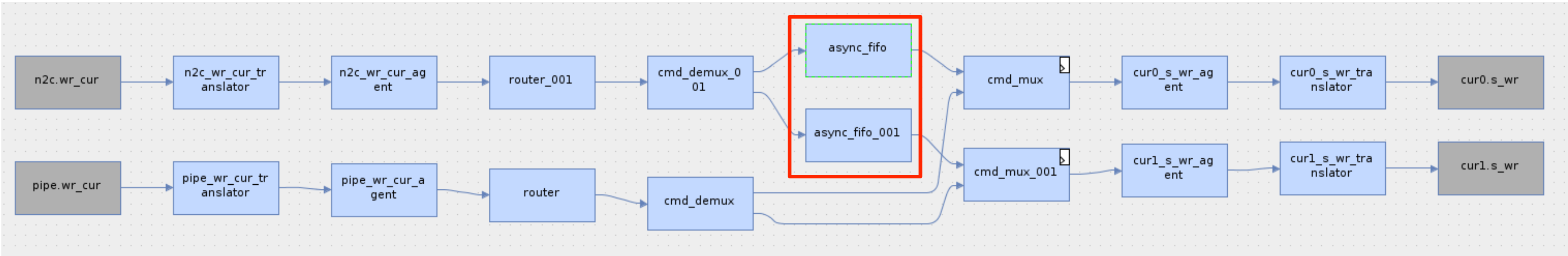
	Clock A	Clock B
GENIE vs. Manual	-1%	+9%
GENIE vs. Qsys	+25%	+27%

Results: Area Usage



	ALM	M20K
GENIE vs. Manual	+3.7%	+0%
GENIE vs. Qsys	-17%	-57%

Results: Observations



- RAM block usage: Clock Crossings
  - Qsys inserts too many crossings → High RAM usage
  - GENIE intelligently inserts crossings to reduce area

Qualitative Ease-of-Use Advantage

Recall: Pipeline’s cache reads require known fixed latency

- With Qsys
  - had to determine interconnect’s contribution by simulation and then modify Verilog source code by hand
- With GENIE
  - Used latency introspection to tell pipeline what the latencies are

Conclusions

1. For our (single, representative) fine-grained example:
   - **Similar** to hand-made: +4% area
     - 72% lines of interconnect code
   - **Better** than Qsys: 25% faster 17% smaller
     - 34% fewer lines of interconnect code

2. Qualitative design flow improvements (fixed latency)

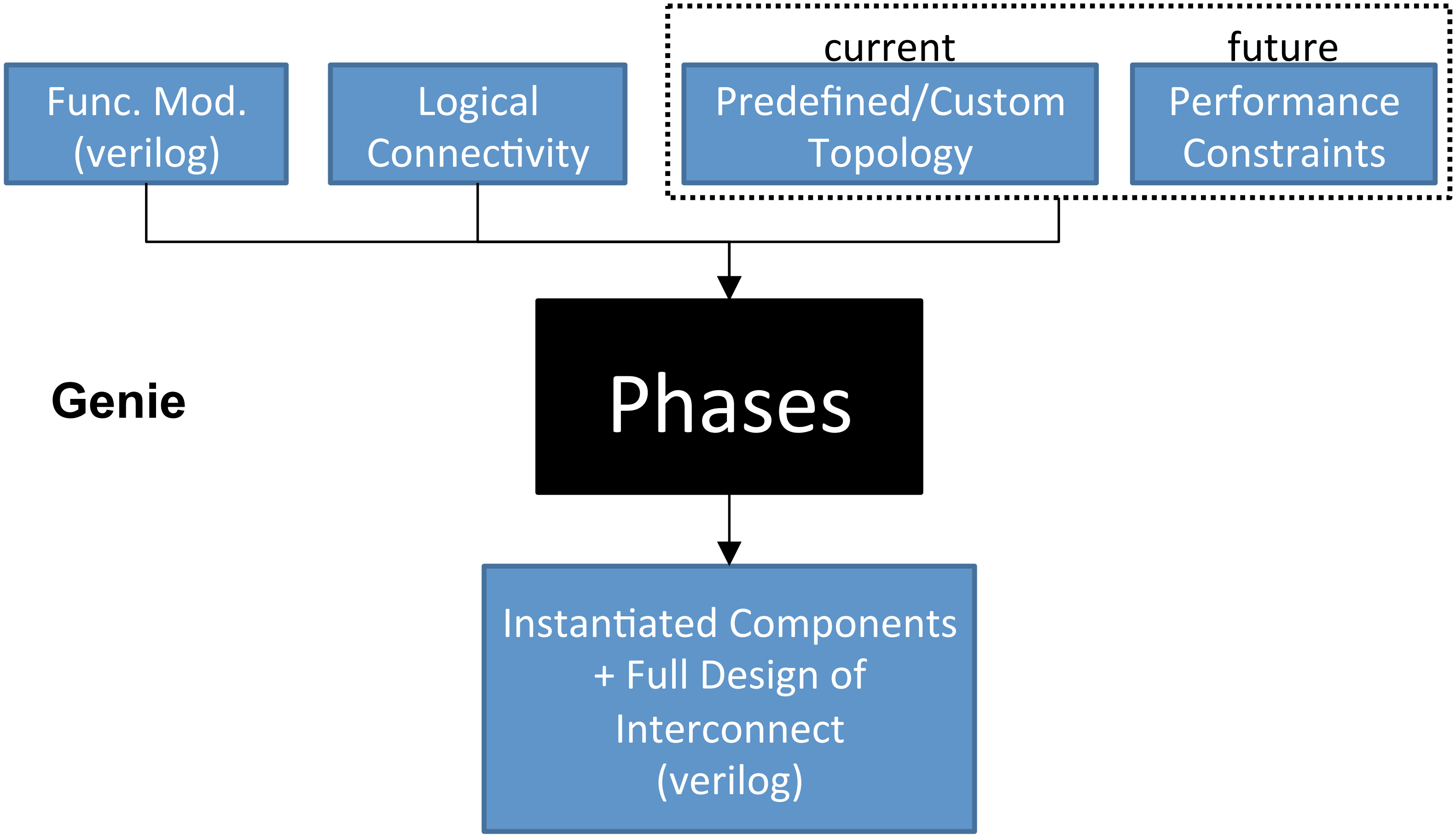
Future Work

1. Move towards Grand Vision
   - Explore/exploit topology communication ability
   - Allow performance spec
   - Automatic topology based on communication patterns
2. Evaluate full LU system, and others (coarse-grained), rapid design-space exploration of different topologies
3. Build higher-level memory-mapped protocols on top of existing GENIE infrastructure

Software Release

- Available at: http://www.eecg.toronto.edu/~jayar/software/GENIE

GenIE Flow

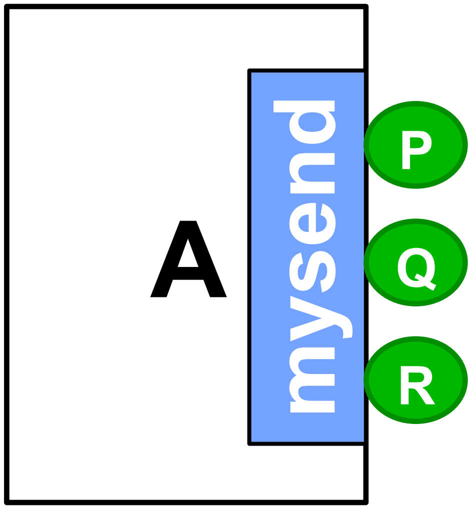

current
future
Func. Mod. (verilog)
Logical Connectivity
Predefined/Custom Topology
Performance Constraints
Genie
Phases
Instantiated Components + Full Design of Interconnect (verilog)

Some Details on the Input

- Data, flow control, and “where to go/where it came from”
- Signals:
  - data: zero or more (tagged), arbitrary width
  - valid
  - ready
  - sop (start of packet)
  - eop (end of packet)
  - lp_id (linkpoint ID, selects/informs linkpoint on interface)
- Not all need to be present

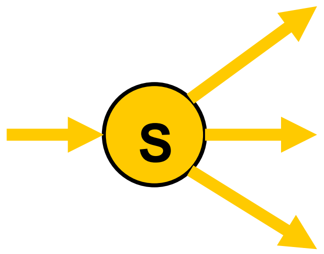
Example Input: Component Spec (Lua)

```
component('A', 'ver_module_name')
  clock_sink('InClk', 'in_clk_sig_name')
  interface('mysend', 'rs', 'out', 'InClk')
    signal('valid', 'valid_sig_name')
    signal('ready', 'ready_sig_name')
    signal('data', 'data_sig_name_x', 8, 'x')
    signal('data', 'data_sig_name_y', 13, 'y')
    signal('lp_id', 'lpid_sig_name', 2)
    linkpoint('P', "2'b00")
    linkpoint('Q', "2'b01")
    linkpoint('R', "2'b10")
```

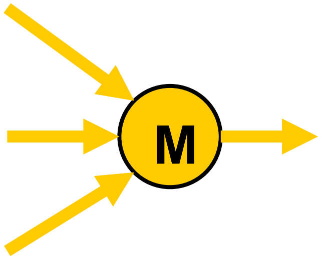


Interconnect Architecture

- Split/Merge (Y. Huan et al., FPT 2012)
- Lightweight, composable switching primitives
- Split: one to many



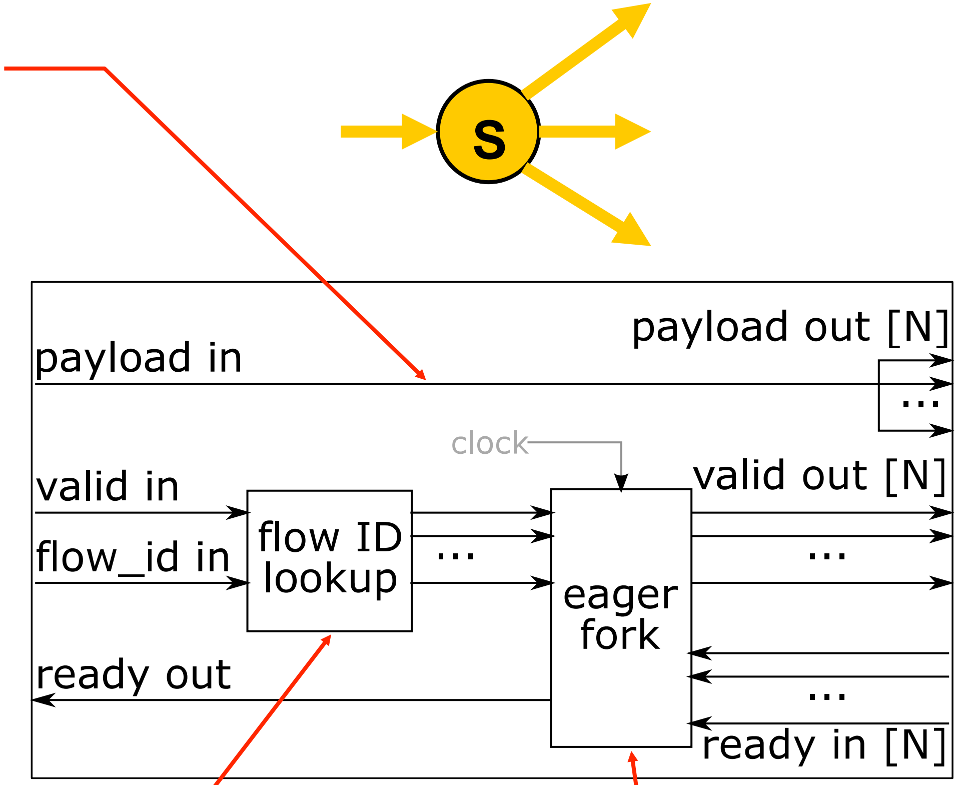
- Merge: many to one



- Misc. conversion/utility blocks

Split Node

payload is broadcast

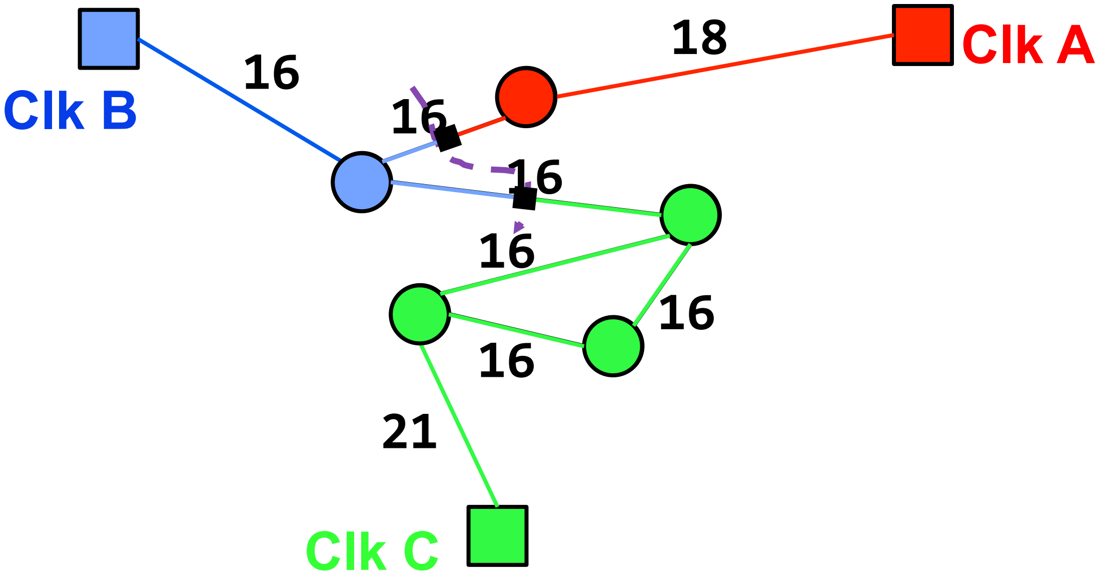


FlowID → which output(s!) to send to

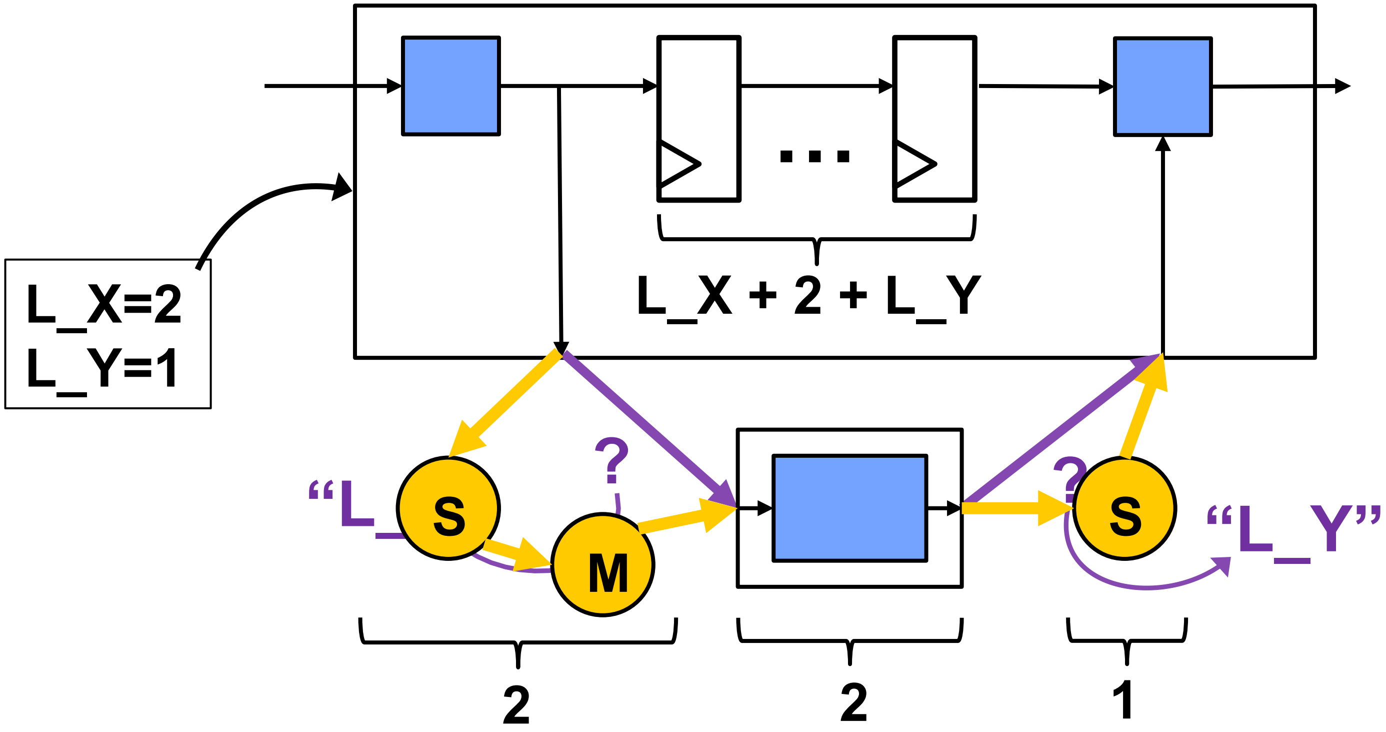
state tracking

Optimization: Smart Clock Domain X-ing

1. Create graph: color = clock domain
2. Edge weight = link width (bits)
3. Find min-weight cut = crossing points
4. Assign domains, insert clock converters



Latency Introspection


L_X=2
L_Y=1
...
L_X + 2 + L_Y
"L_
S
M
?
?
S
"L_Y"
2
2
1