CS250 VLSI Systems Design Lecture 8: Introduction to Hardware Design Patterns

John Wawrzynek
Chris Yarp (GSI)

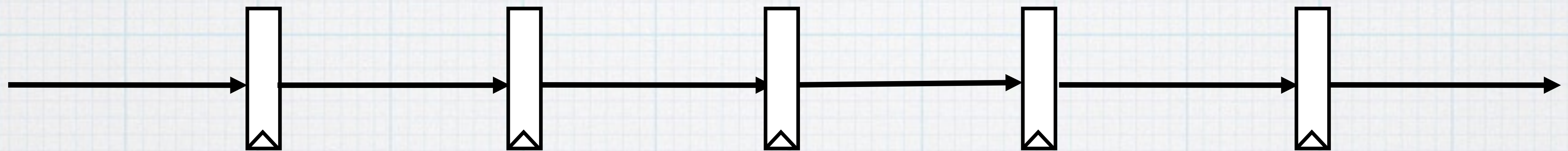
UC Berkeley

Spring 2016

Slides from Krste Asanovic

A Difficult Design Problem?

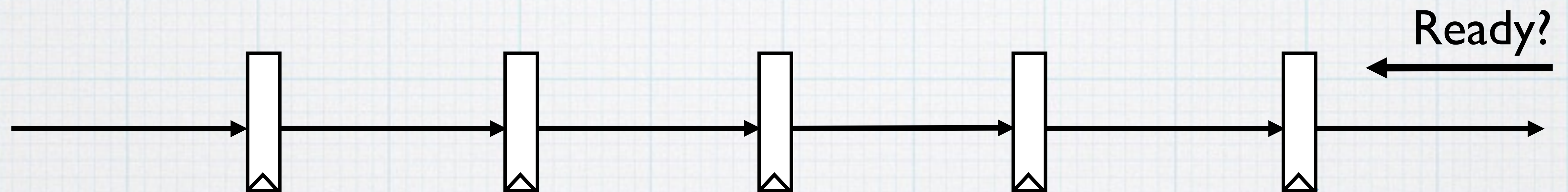
A humble shift register



(For today's lecture, we'll assume clock distribution is not an issue)

First Complication: Output Stall

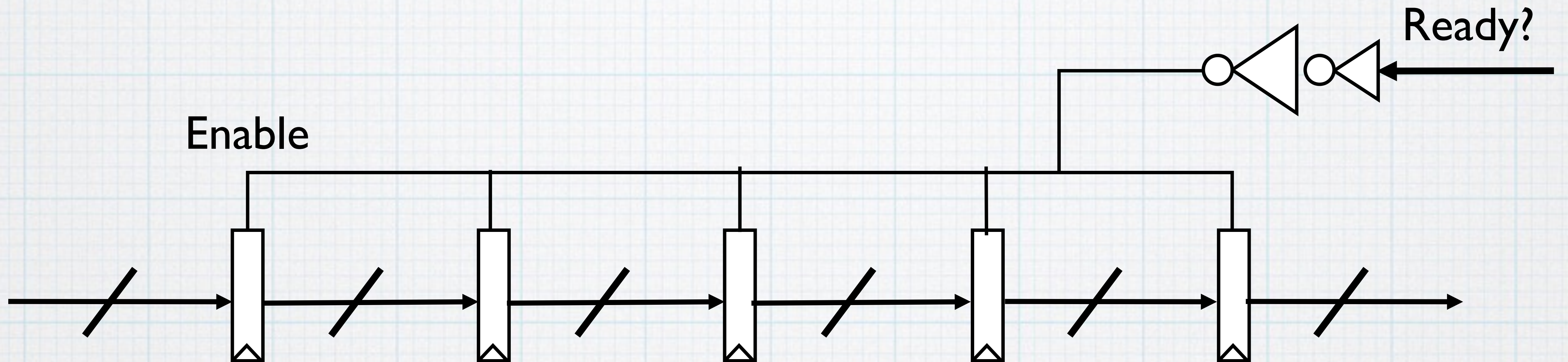
Shift register should only move data to right if output ready to accept next item



What complication does this introduce?

Need to fan out to enable signal on each flop

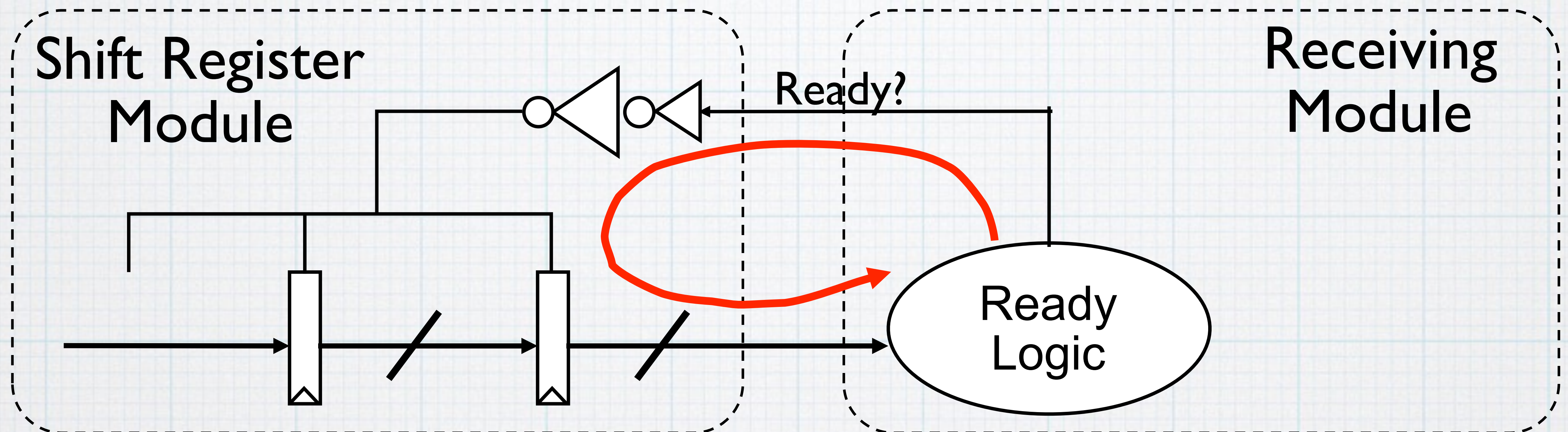
Stall Fan-Out Example



- 200 bits per shift register stage, 16 stages
- 3200 flip-flops
- How many fanout-of-four gate delays to buffer up ready signal?
  - *$Log_4(3200) = 5.82$, ~ 6 FO4 delays!*

This doesn't include any penalty for driving enable signal wires!

Loops Prevent Arbitrary Resizing



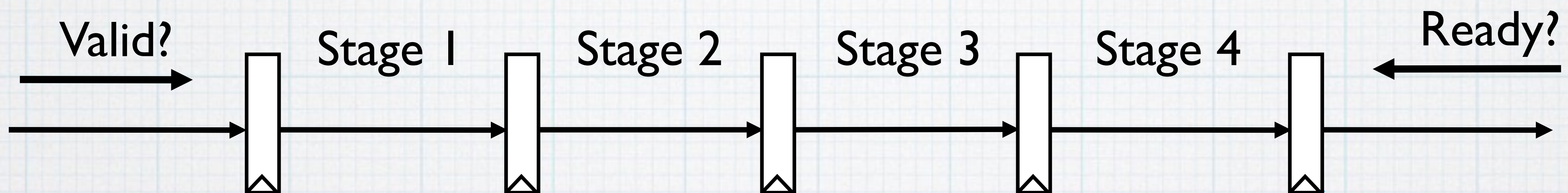
We could increase size of gates in ready logic block to reduce fan out required to drive ready signal to flop enables…

But this increases load on flops, so they have to get bigger

--- a vicious cycle!

Second Complication: Input Bubbles

Sender doesn't have valid data every clock cycle, so empty "bubbles" inserted into pipeline



Pipeline Diagram

!Valid

Want to "squeeze" bubble out of pipeline

!Valid

Stage 1

Stage 2

Stage 3

Stage 4

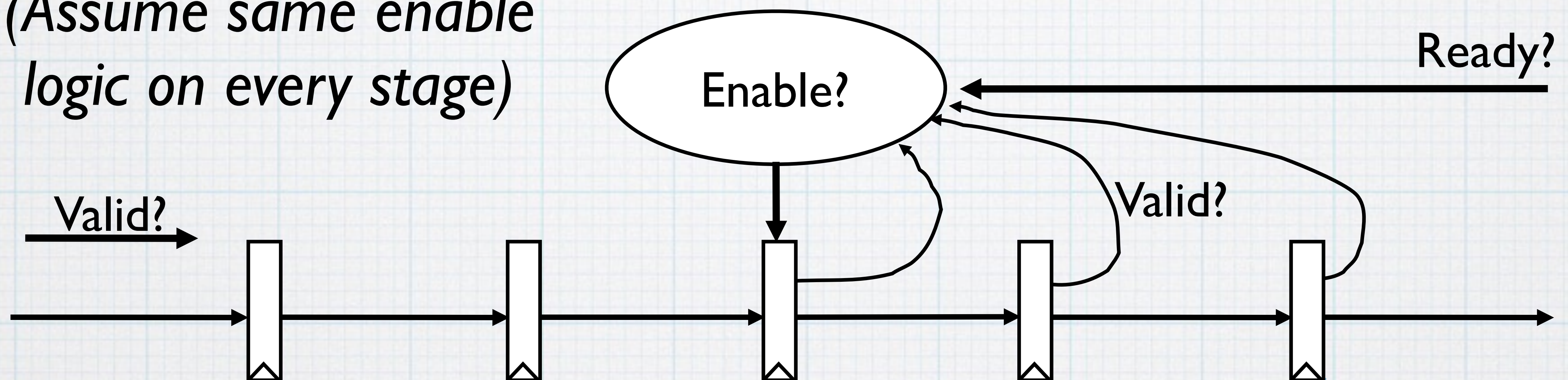
!Ready

Time

!Ready

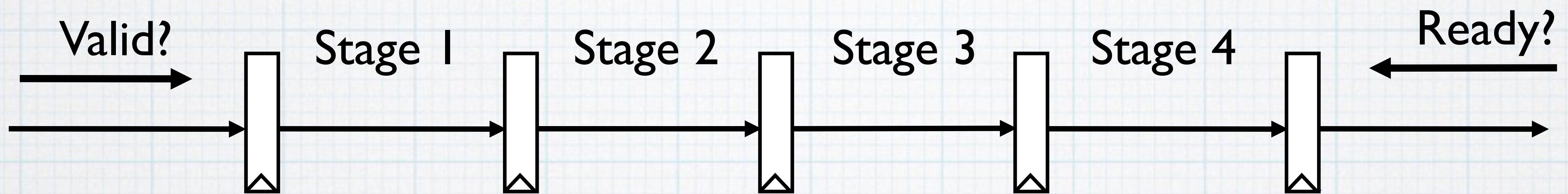
Logic to Squeeze Bubbles

Can move one stage to right if Ready asserted, or if there are any bubbles in stages to right of current stage



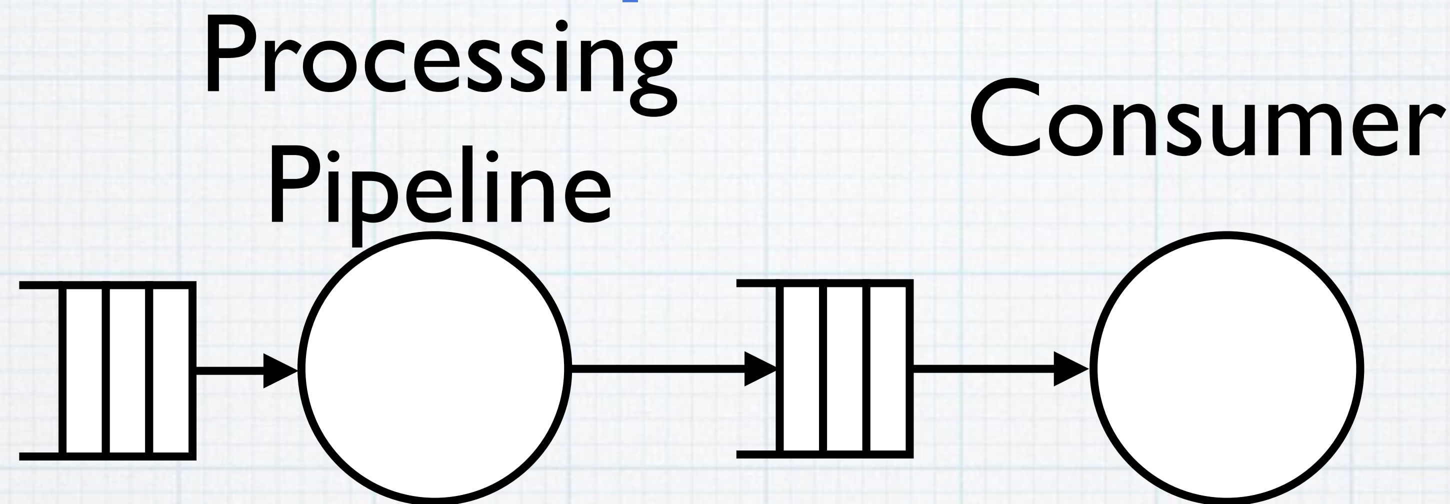
- Fan-in of number of valid signals grows with number of stages
- Fan-out of each stage's valid signal grows with number of stages
- Longer combinational paths as number of pipeline stages grows

A Common Design Problem



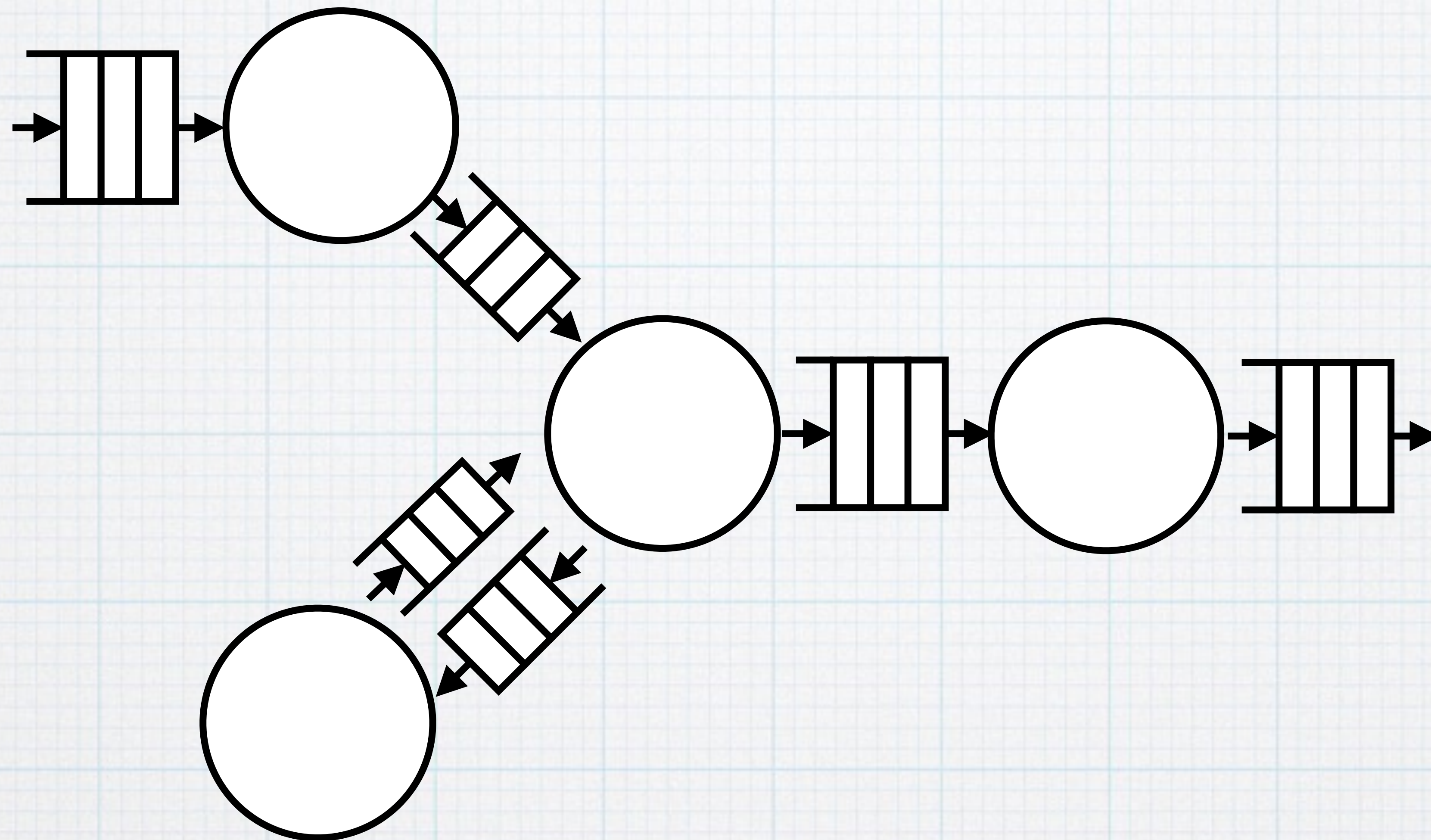
- The shift register is an abstraction of any synchronous pipelined block of logic that accepts input data and produces output data, where input and output might not be ready every clock cycle
- How to manage growth in control logic complexity?

Solution: Decouple Units with FIFOs



- Pipeline only cares whether space in FIFO, not about whether consumer can take next value
  - Breaks combinational path between pipeline control logic and consumer control logic
- For full throughput with decoupling, need at least two elements in FIFO
  - With only one element, have to ping-pong between pipeline enqueue and consumer dequeue
  - Allowing both enqueue and dequeue in same cycle to single-element FIFO destroys decoupling (back to a synchronous connection)

Decoupled Design Discipline



- Many large digital designs are divided into local synchronous pipelines, or units, connected via decoupling FIFOs
  - Approx. 10K-100K gates per unit
- Decoupled units may have different clocks (GALS)
  - In which case, need asynchronous FIFOs

Hardware Design Patterns

- Decoupled units are an example of a *design pattern*
- Pattern: *Solution to a commonly recurring design problem*
- Idea of patterns and a "pattern language" first proposed for building architecture (Christopher Alexander)
  - "Pattern language" is an interlocking set of design patterns
    - Probably better named a "pattern hierarchy"
  - Alexander proposed single pattern language covering architecture from design of cities to design of roof caps
- Patterns popular in software engineering ("Gang of Four")*
- This semester continues an experiment to see if we can teach hardware design using patterns

* In 1994, four authors Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides published a book titled *Design Patterns - Elements of Reusable Object-Oriented Software*.

Berkeley Hardware Pattern Language (BHPL) Goals

- BHPL captures problem-solution pairs for creating hardware designs (machines) to execute applications

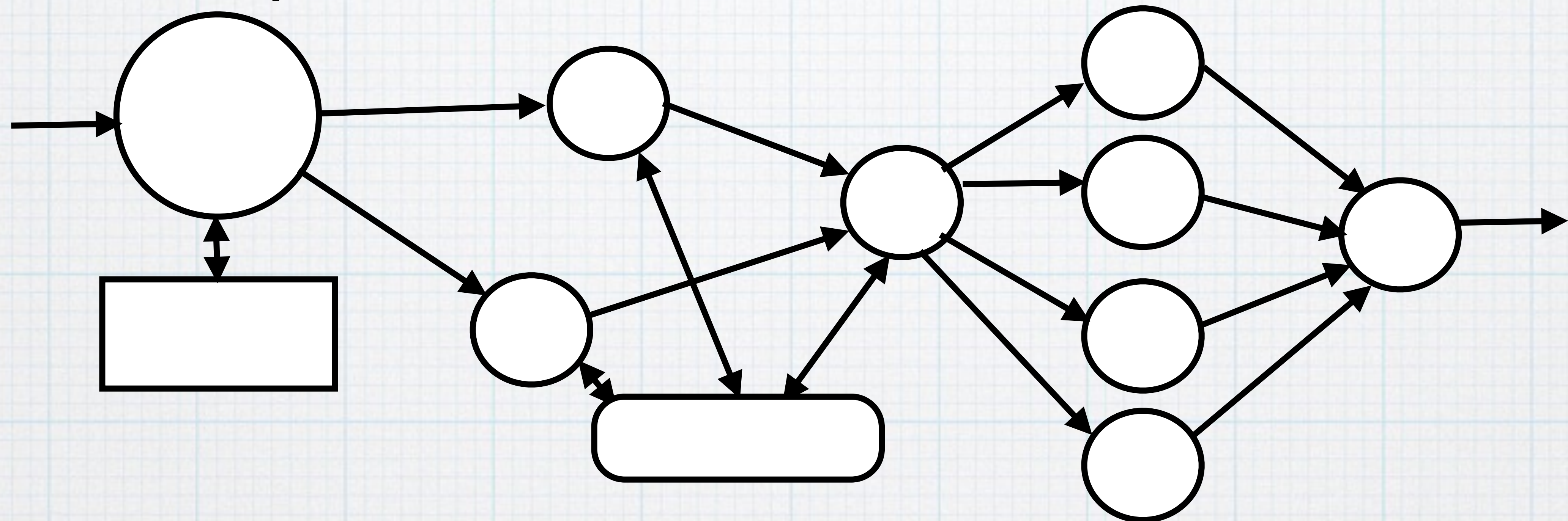
- BHPL Non-Goals
  - Doesn't describe applications themselves, only machines that execute applications and strategies for mapping applications onto machines

Pattern Vocabulary: Why a Vocabulary?

- Need a standard graphical and textual language to describe the problems and solutions in our pattern language
  - Really just a consistent way of drawing and talking about block diagrams

Machine Vocabulary

- Machines described using a hierarchical structural decomposition



Units (generalized processing engines)

Memories

Networks (connect multiple entities)

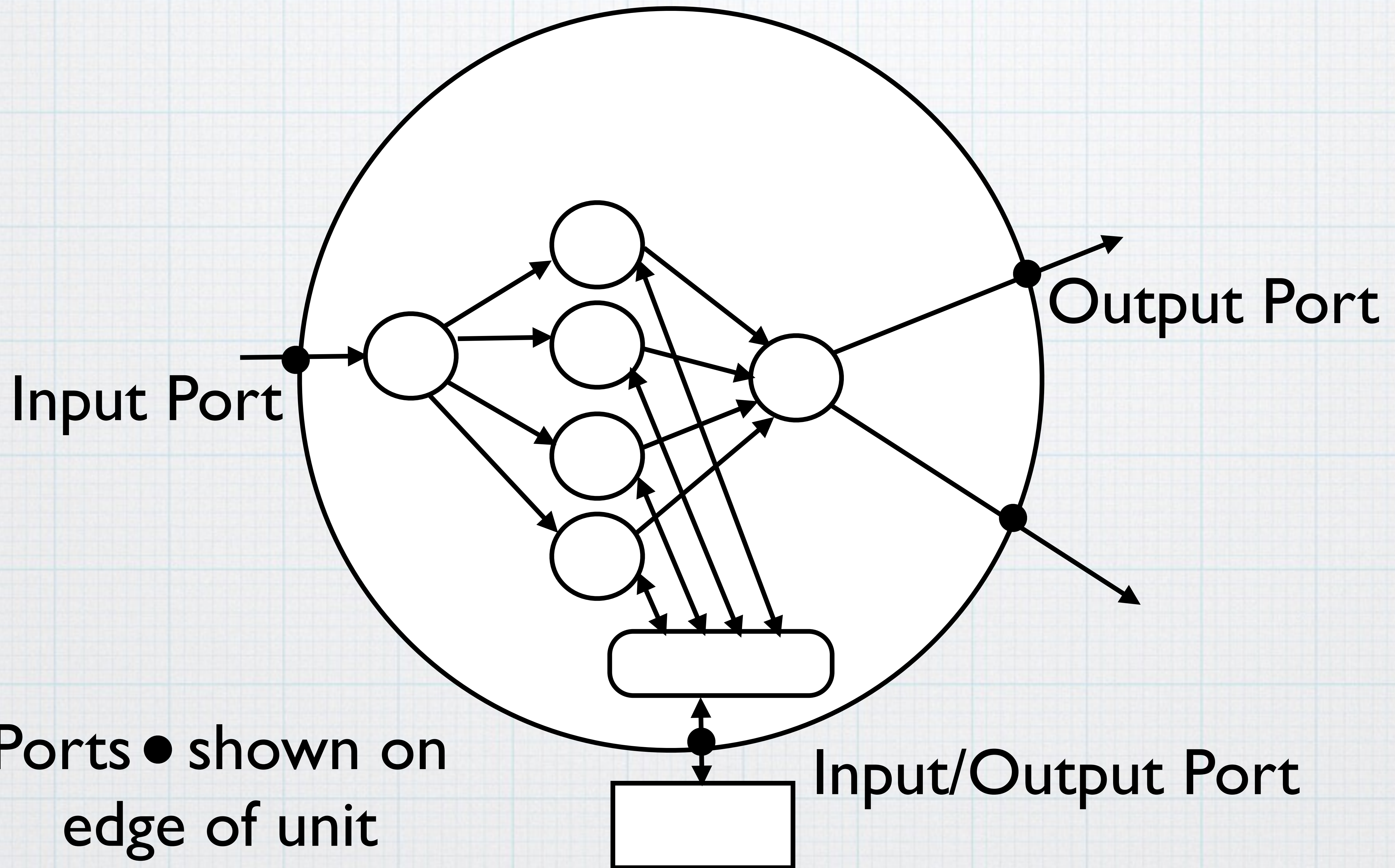
Channels (point-to-point connections)

(Memories, Networks, Channels are specialized Units)

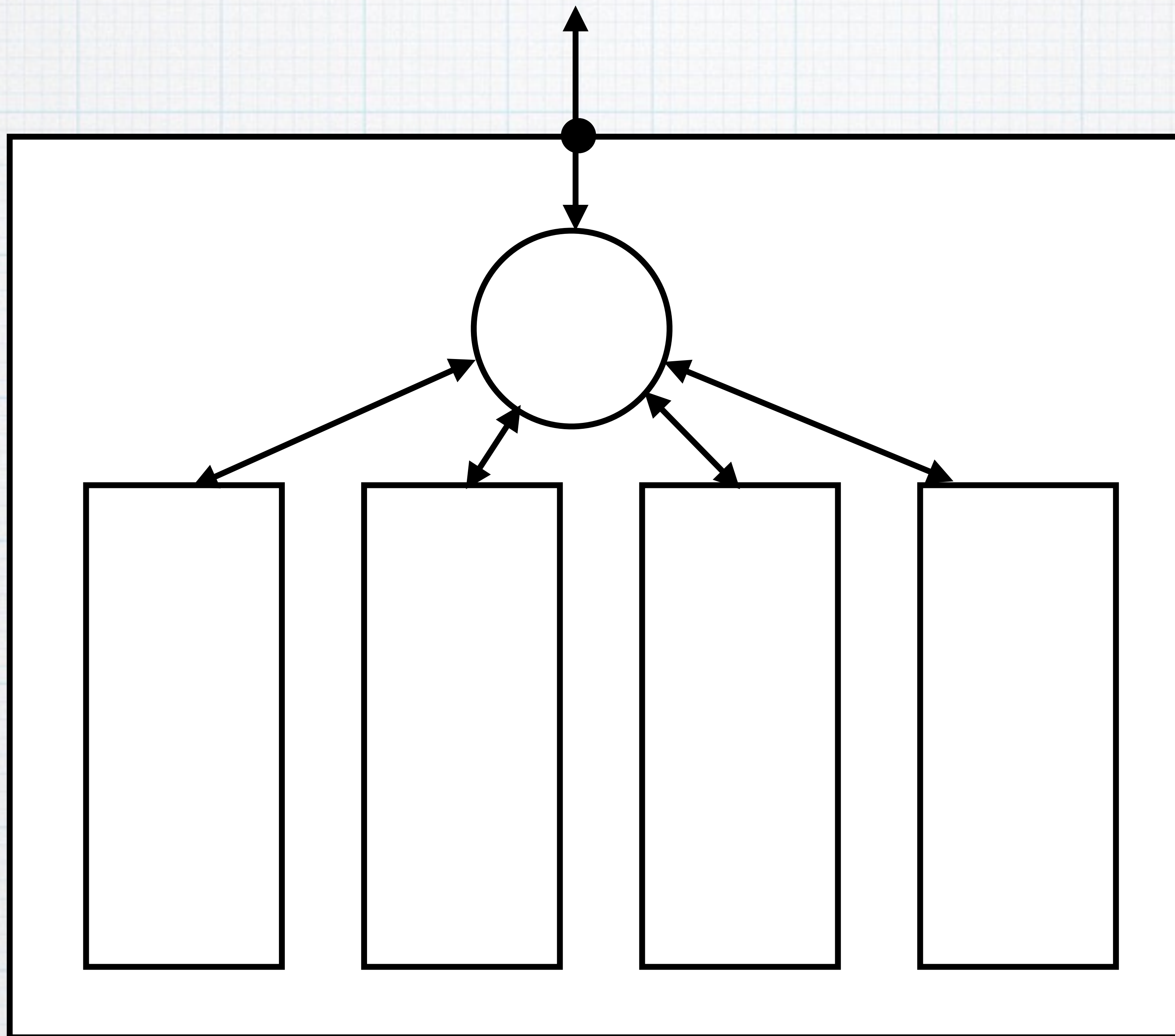
Unit types

- Memories, Networks, and Channels are also units, just with specialized symbol to convey their main intended purpose.
  - Memories store data.
  - Networks connect multiple entities
  - Channels are point-point communication paths
- High-level channels show primary direction of information flow
  - Might have wires in other direction to handle flow-control etc.

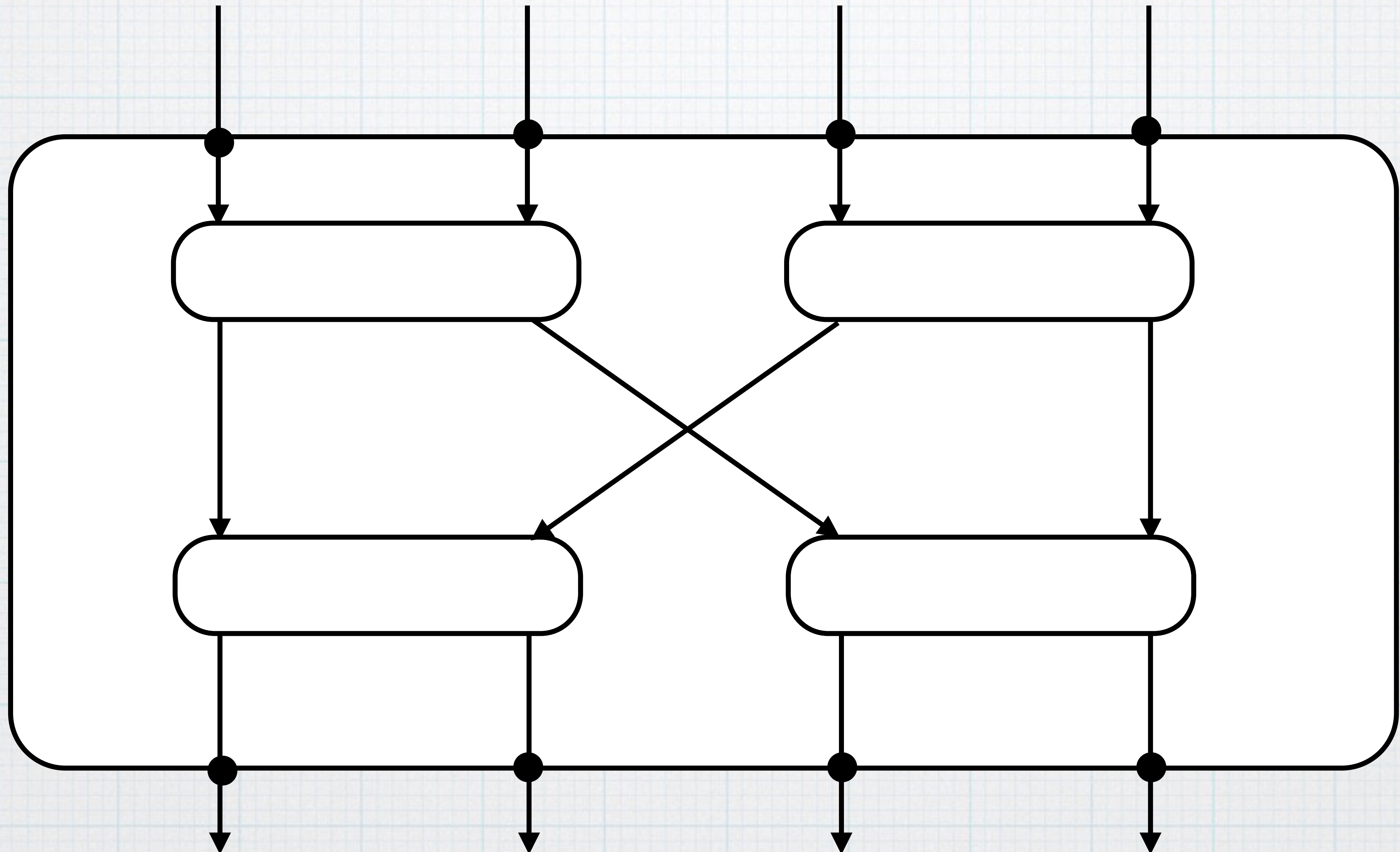
Hierarchy within Unit



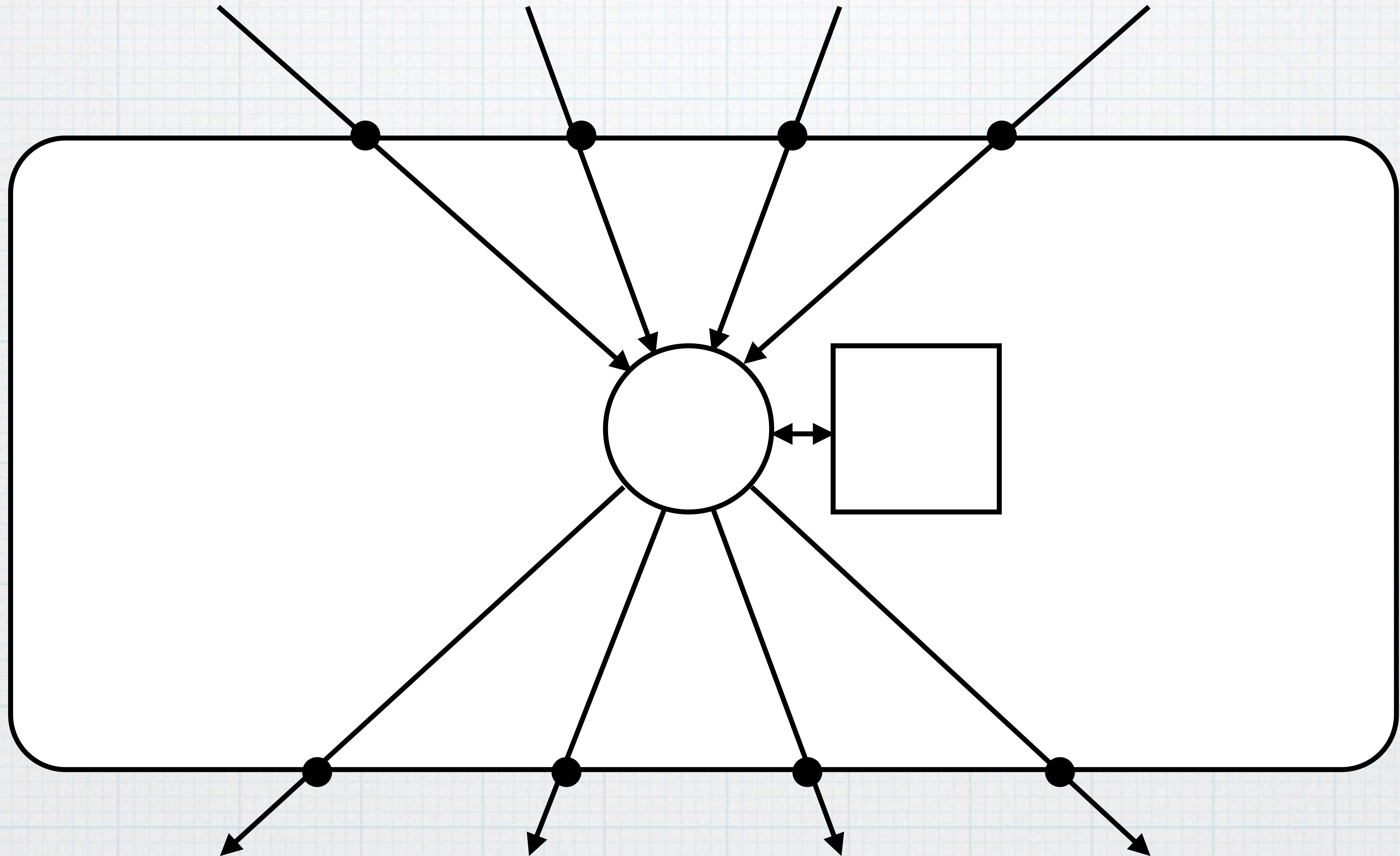
Hierarchy within Memory



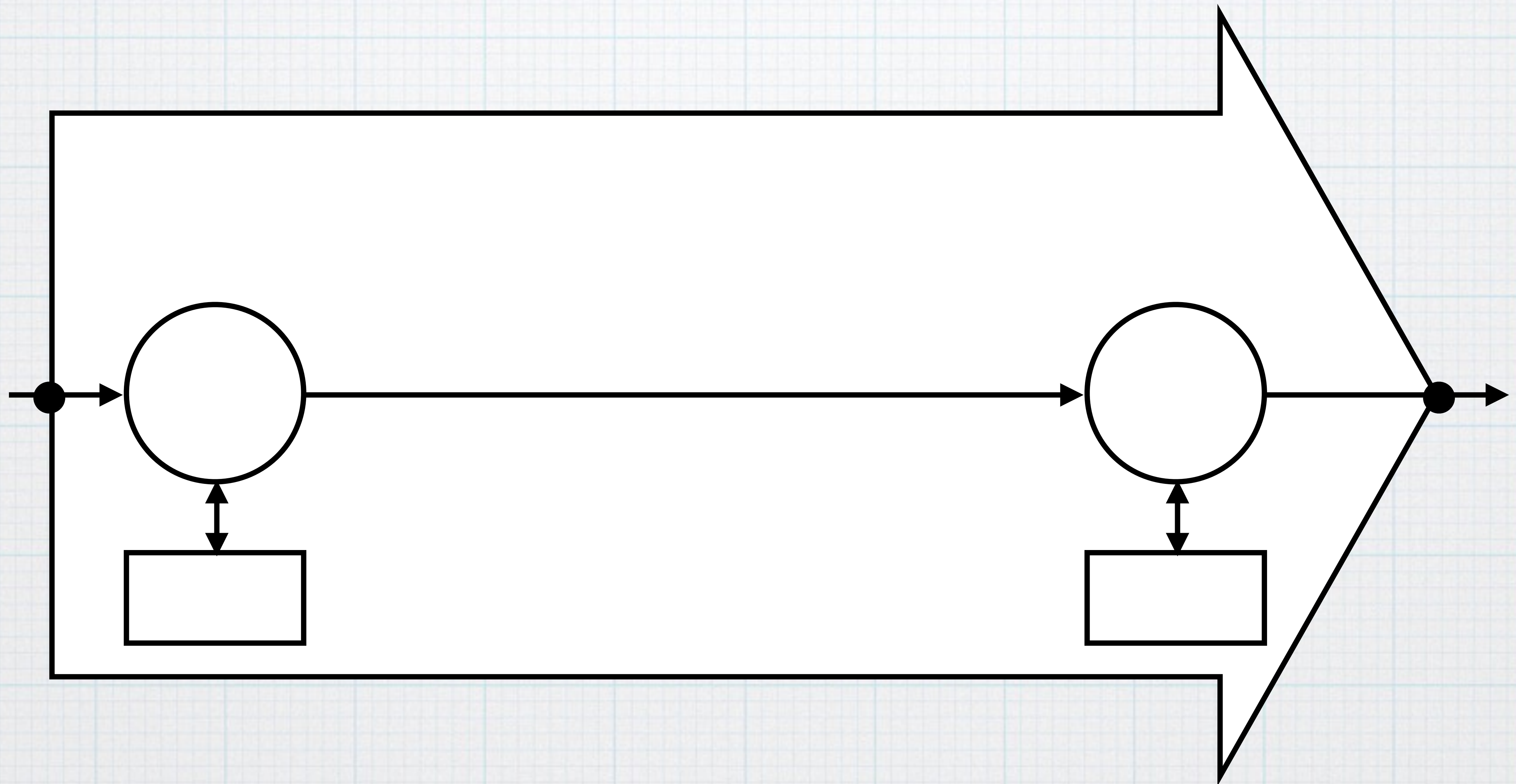
Hierarchy within Network



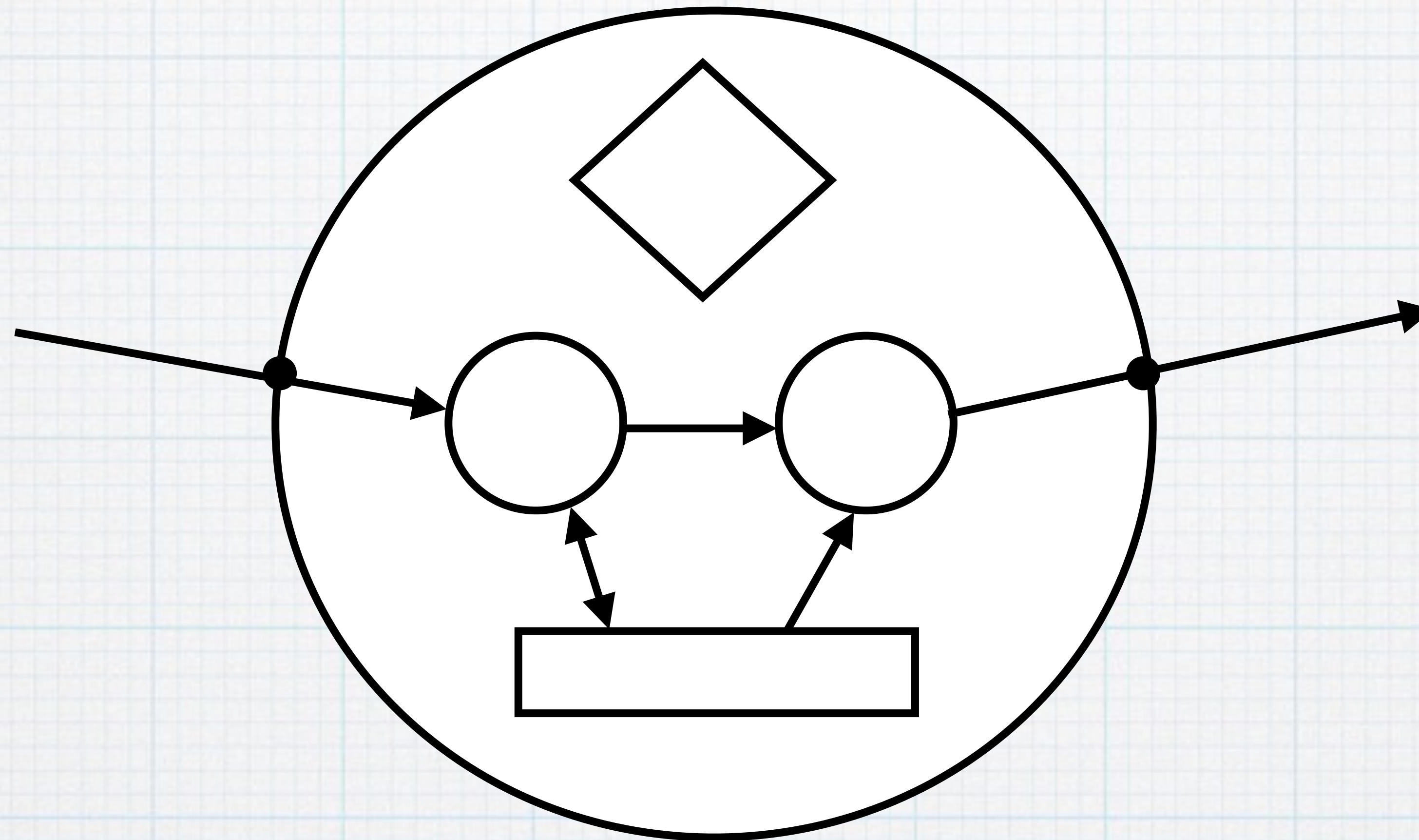
Hierarchy within Network (2)



Hierarchy within Channel

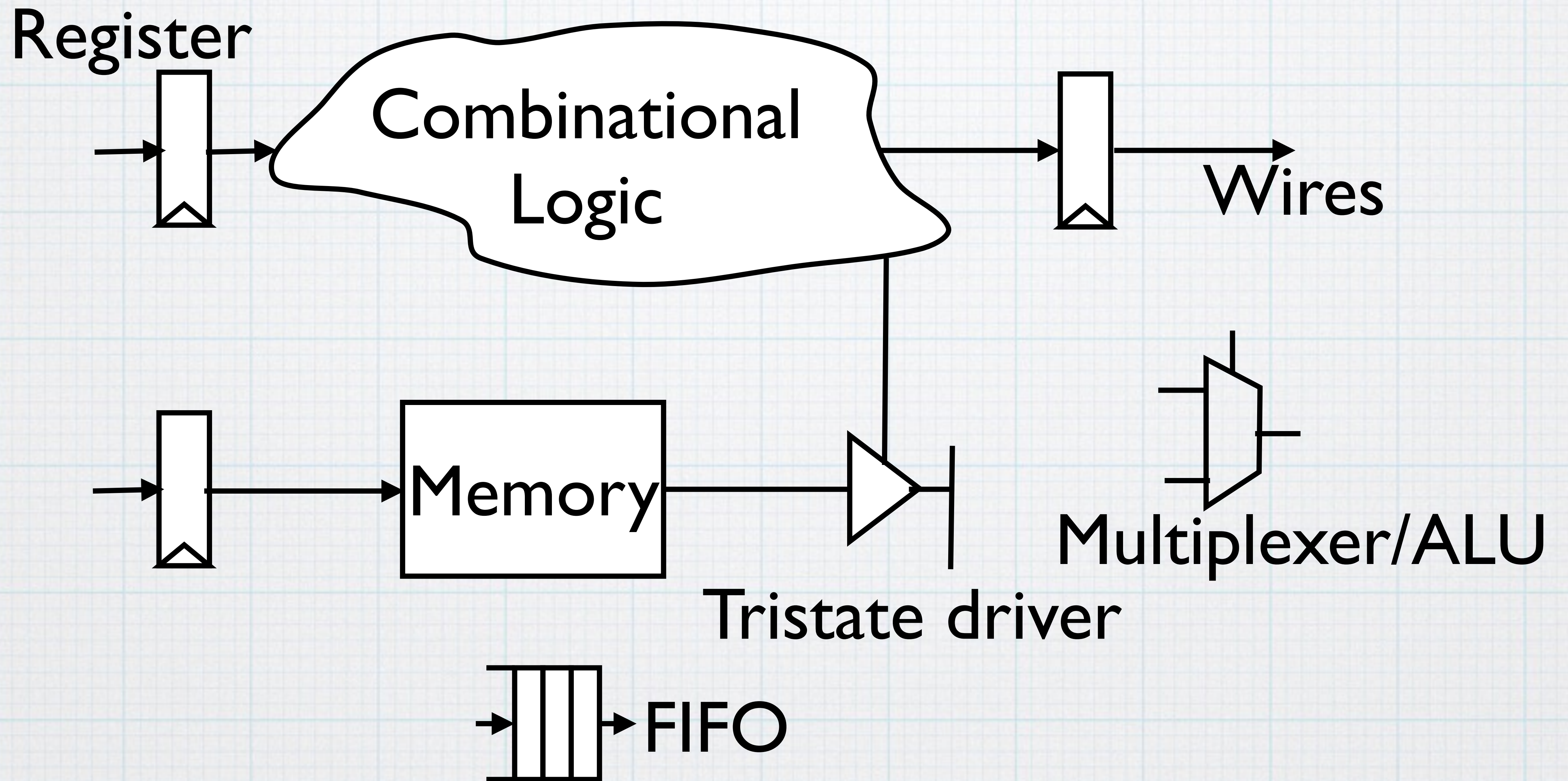


Unit with Synchronous Control



- When there is only a single state machine controlling operation of a unit
- Diamond represents unit-level controller, implied control connections to all other components (datapaths + memories) in unit

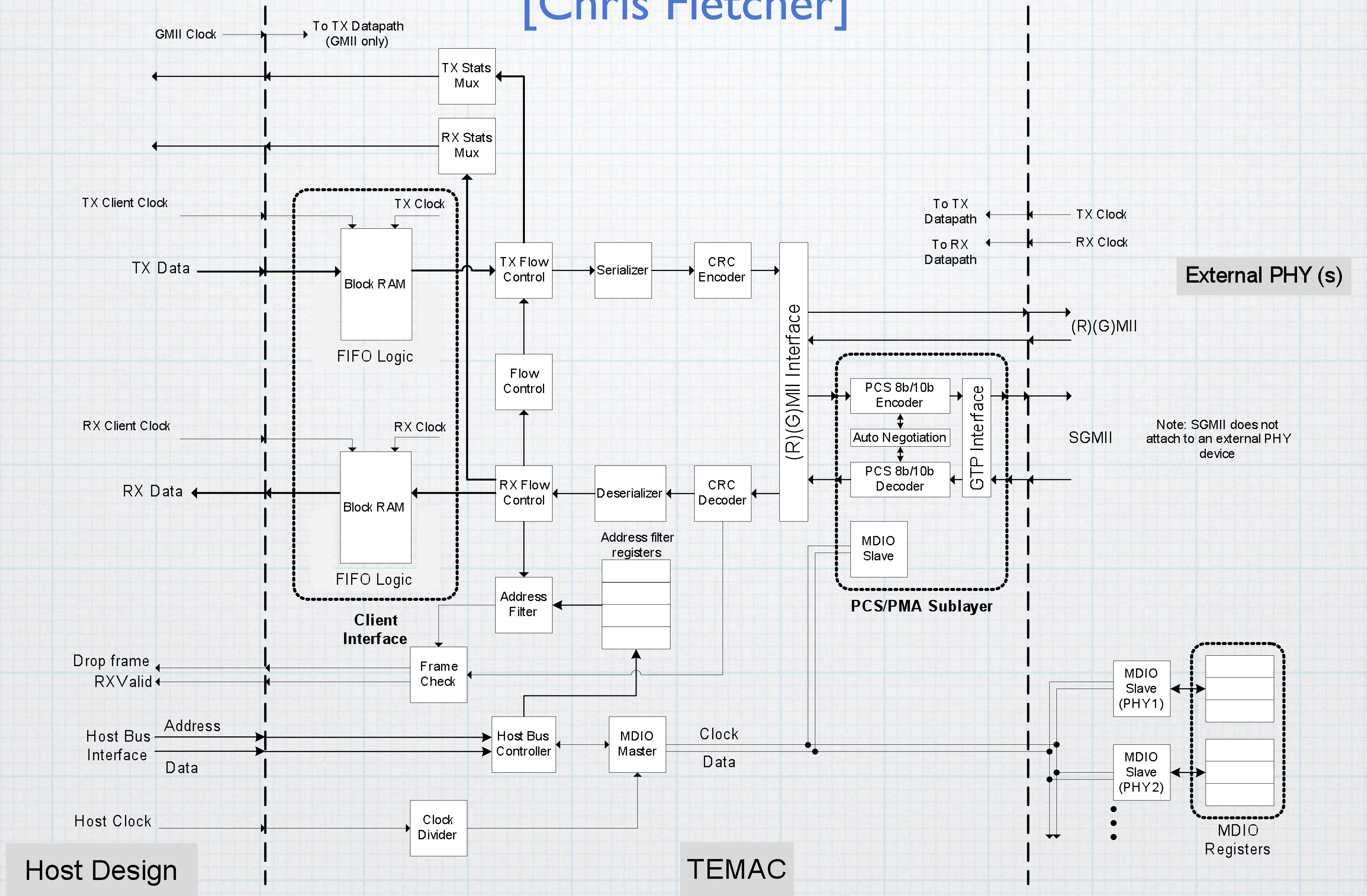
Leaf-Level Hardware



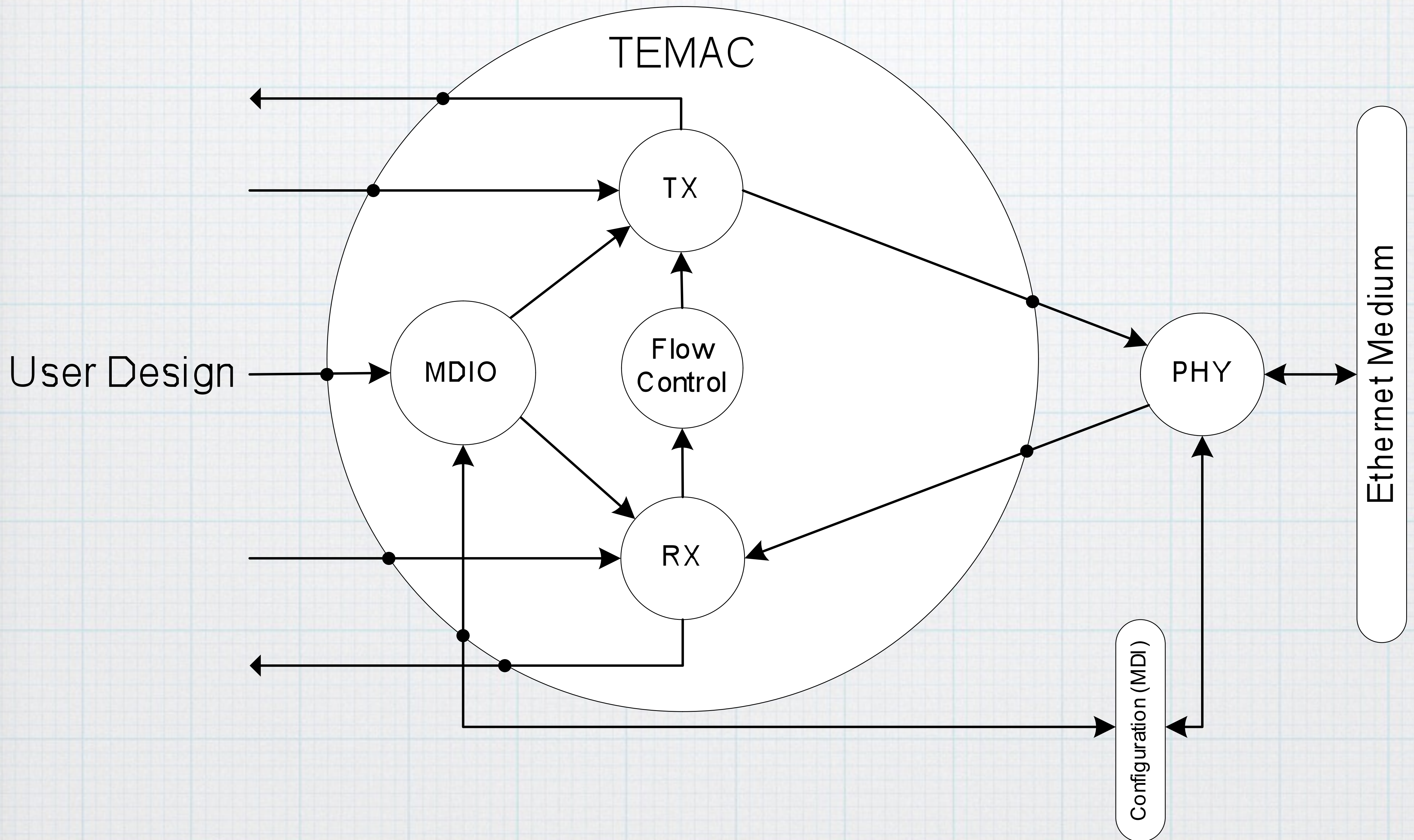
- Conventional schematic notation

Example: Ethernet MAC for Xilinx

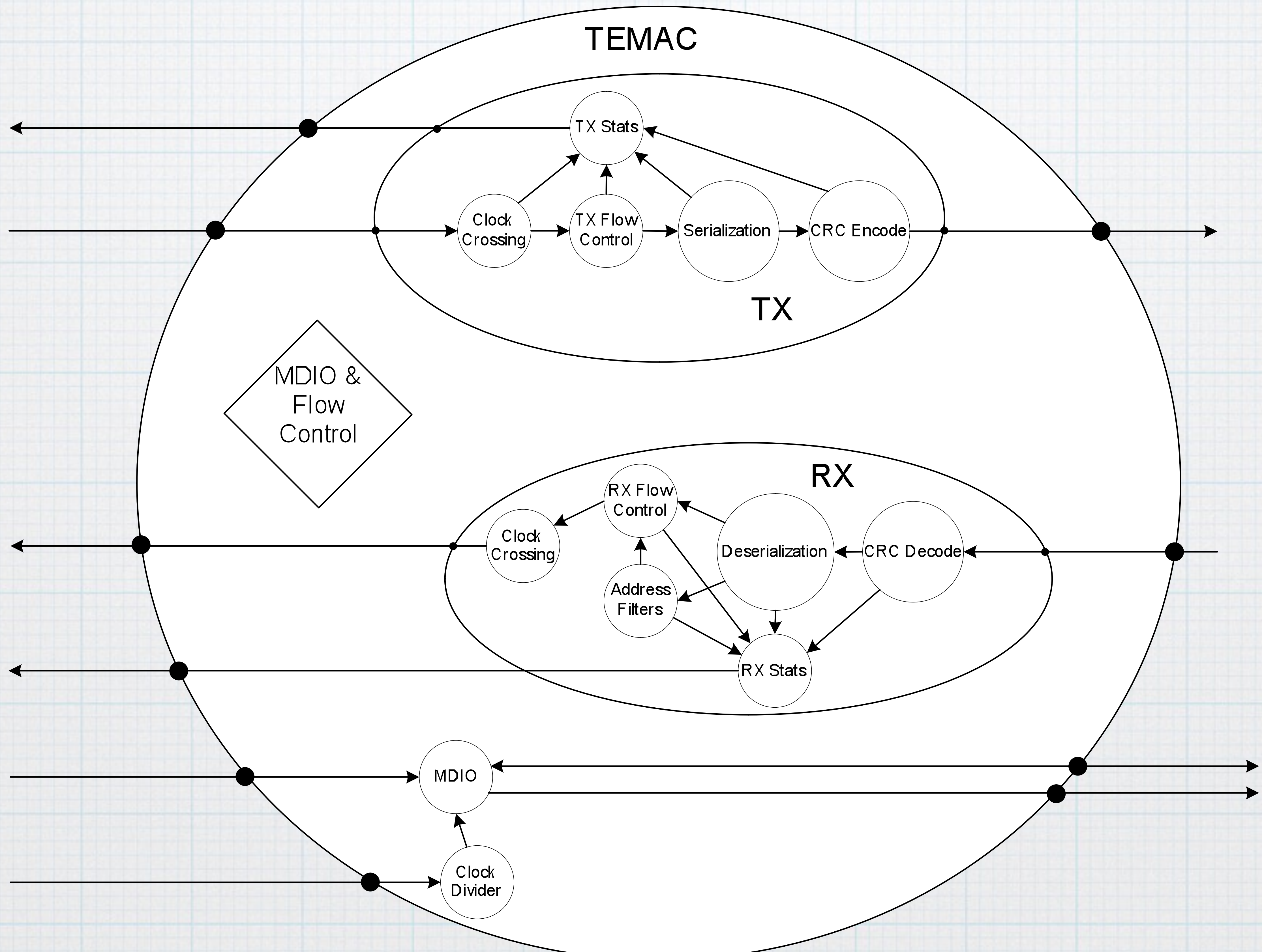
[Chris Fletcher]



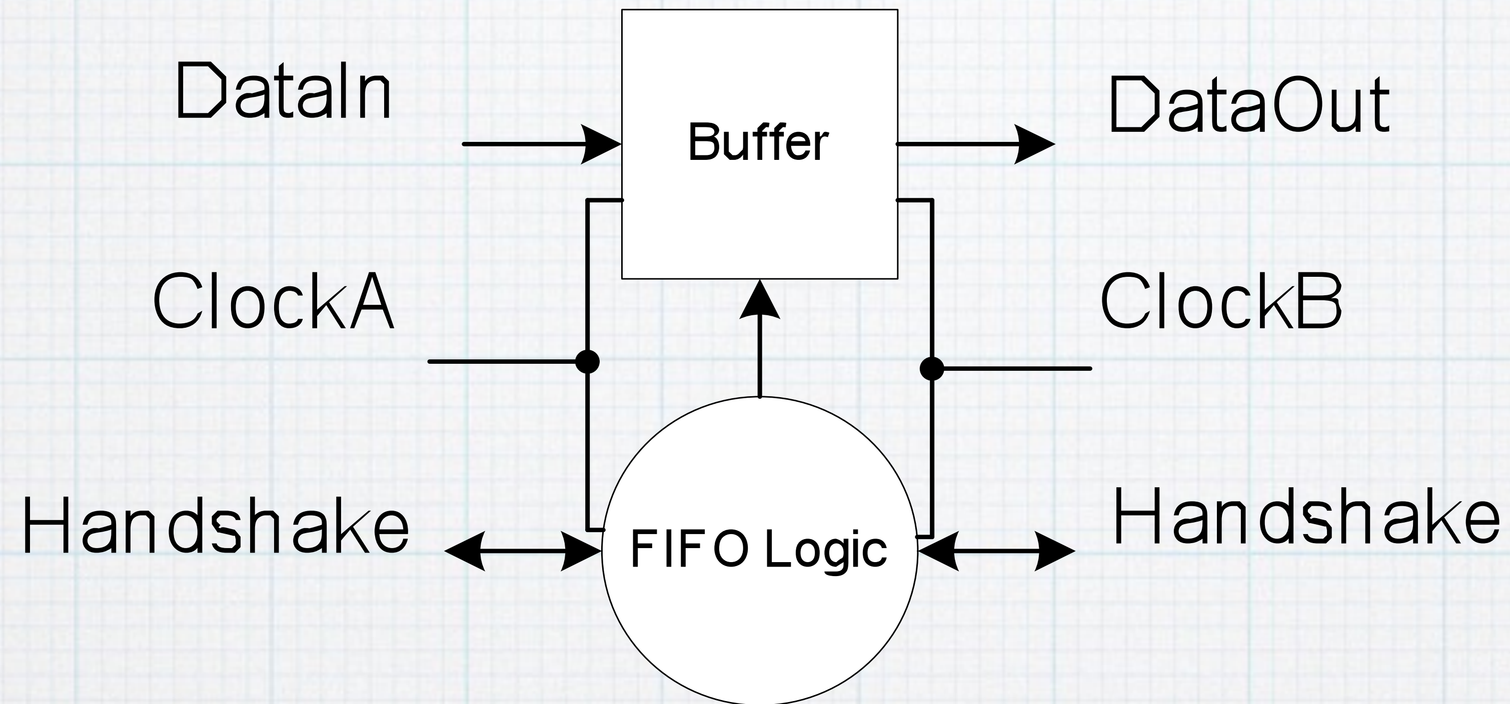
Top-Level

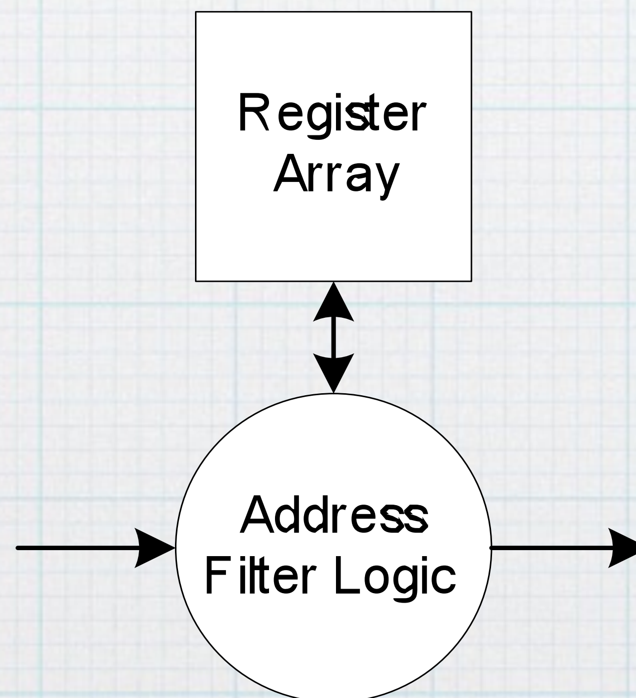


One Level Down



Low-Level Building Blocks





Berkeley Hardware Pattern Language

Pattern Template (Name here)

Problem: Describe the particular problem the pattern is meant to solve. Should include some context (small, high throughput), and also the layer of the pattern hierarchy where it fits.

Solution: Describe the solution, which should be some hardware structure with a figure. Solution is usually the pattern name. Should not provide a family of widely varying solutions - these should be separate patterns, possibly grouped under a single more abstract parent pattern.

Applicability: Longer discussion of where this particular solution would normally be used, or where it would not be used.

Consequences: Issues that arise when using this pattern, but only for cases where it is appropriate to use (use **Applicability** to delineate cases where it is not appropriate to use). These might point at sub-problems for which there are sub-patterns. There might also be limitations on resulting functionality, or implications in design complexity, or CAD tool use etc.

Decoupled Units

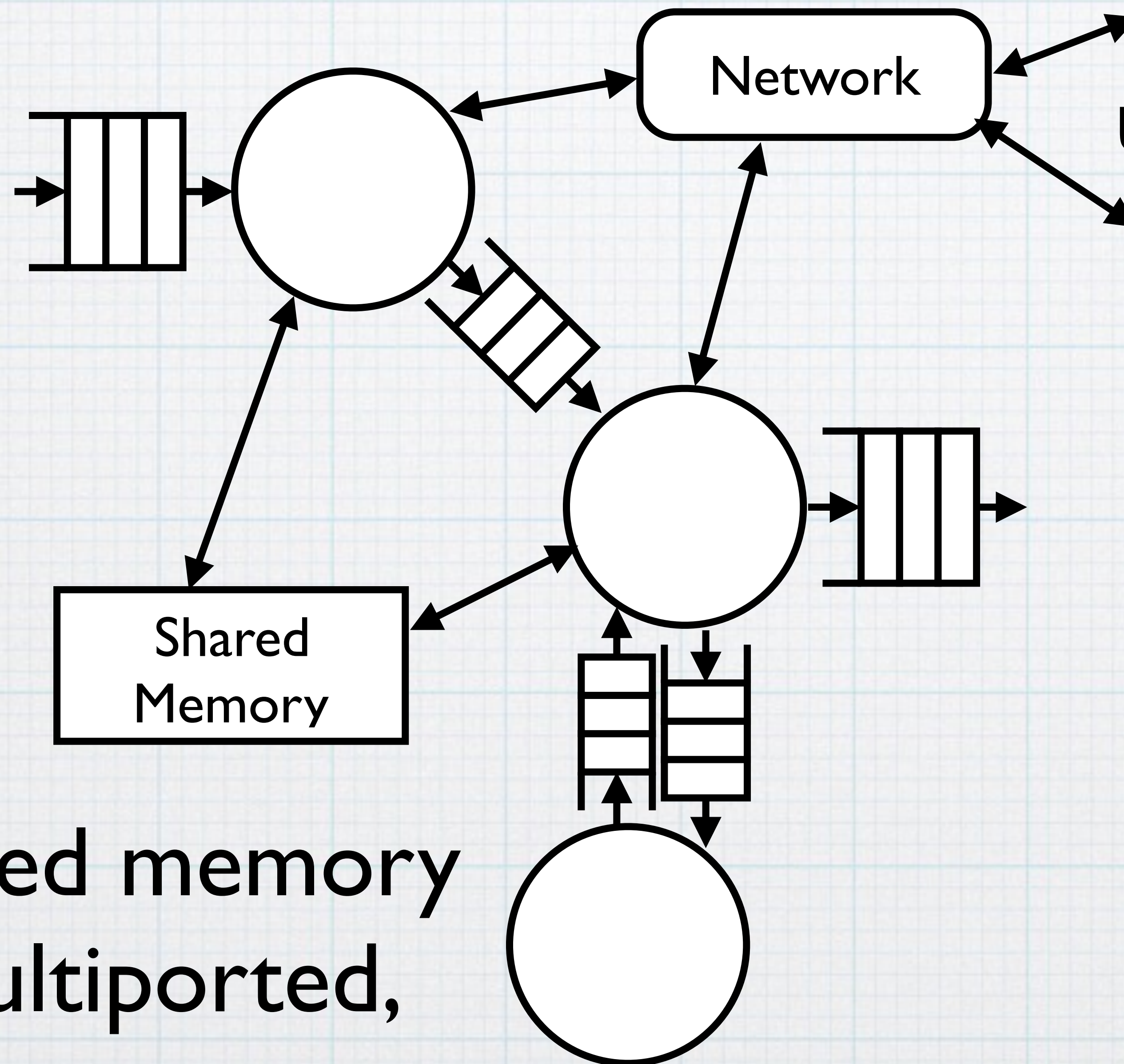
Problem: Difficult to design a large unit with a single controller, especially when components have variable processing rates. Large controllers have long combinational paths.

Solution: Break large unit into smaller sub-units where each sub-unit has a separate controller and all channels between sub-units have some form of decoupling (i.e., no assumption about timing of sub-units).

Applicability: Large unit where area and performance overhead of decoupling is small compared to benefits of simpler design and shorter controller critical paths.

Consequences: Decoupled channels generally have greater communication latency and area/power cost. Sub-unit controllers must cope with unknown arrival time of inputs and unknown time of availability of space on outputs. Sub-units must be synchronized explicitly.

Decoupled Units



Channels to network are usually decoupled in any case

Unless shared memory is truly multiported, channels to memory must be decoupled

Related Patterns

- Often multiple solutions to same problem, but which to pick depends on situation
- **Problem** statement and **Applicability** text should help select the correct pattern to use
- Example: Pipelined versus multi-cycle operators when delay through operator exceeds desired cycle time

Pipelined Operator

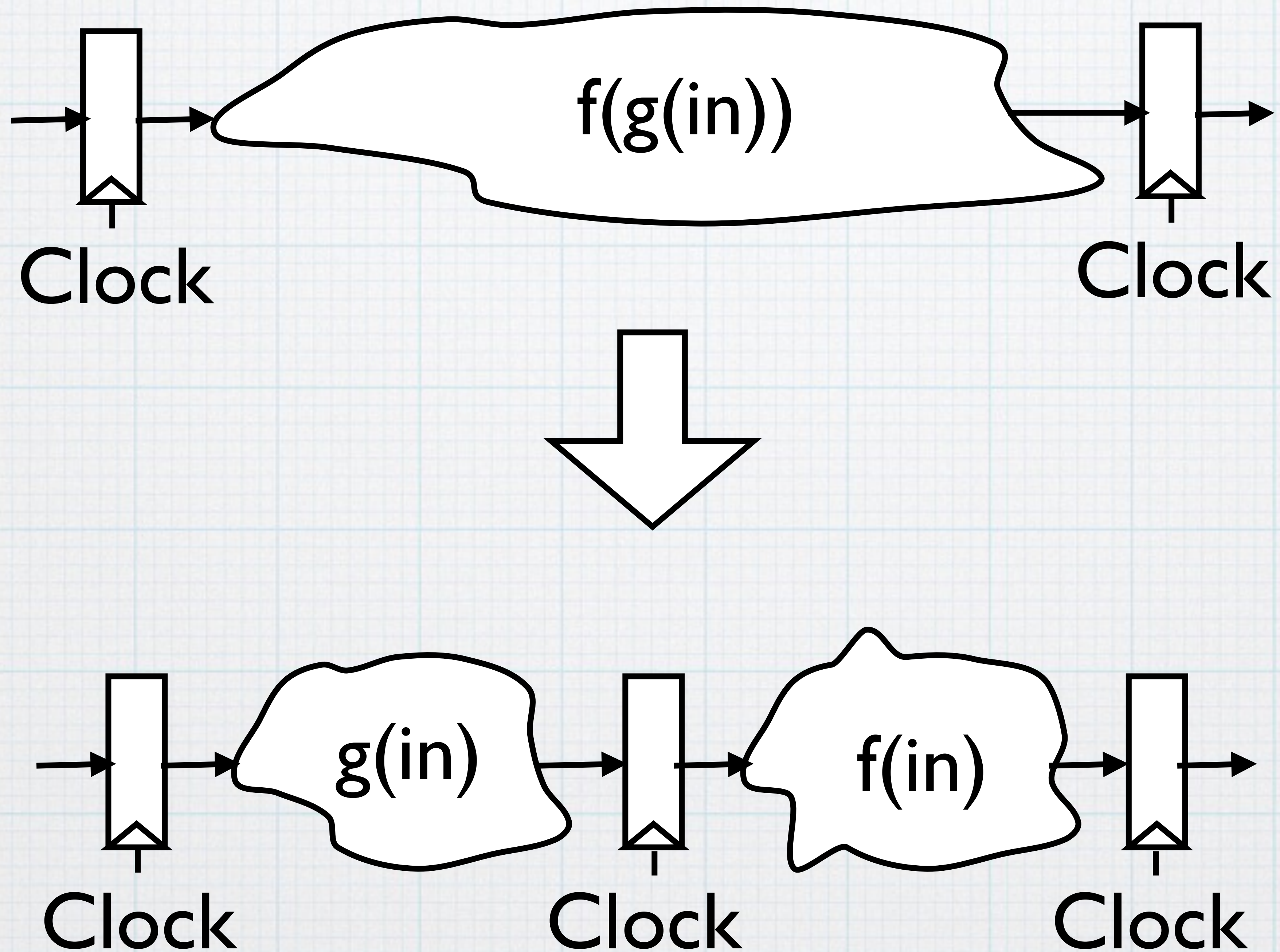
Problem: Combinational function of operator has long critical path that would reduce system clock frequency. High throughput of this function is required.

Solution: Divide combinational function using pipeline registers such that logic in each stage has critical path below desired cycle time. Improve throughput by initiating new operation every clock cycle overlapped with propagation of earlier operations down pipeline.

Applicability: Operators that require high throughput but where latency is not critical.

Consequences: Latency of function increases due to propagation through pipeline registers, adds energy/op. Any associated controller might have to track execution of operation across multiple cycles.

Pipelined Operator



Multicycle Operator

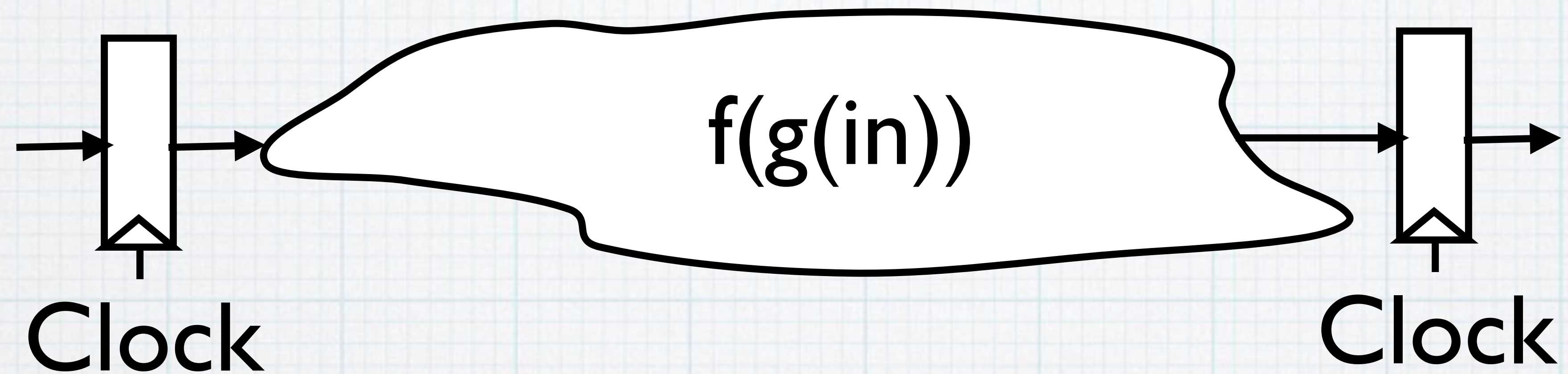
Problem: Combinational function of operator has long critical path that would reduce system clock frequency. High throughput of this function is not required.

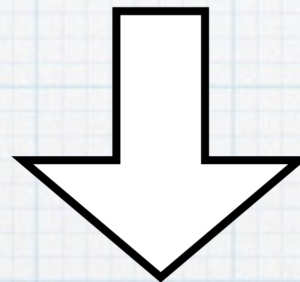
Solution: Hold input registers stable for multiple clock cycles of main system, and capture output after combinational function has settled.

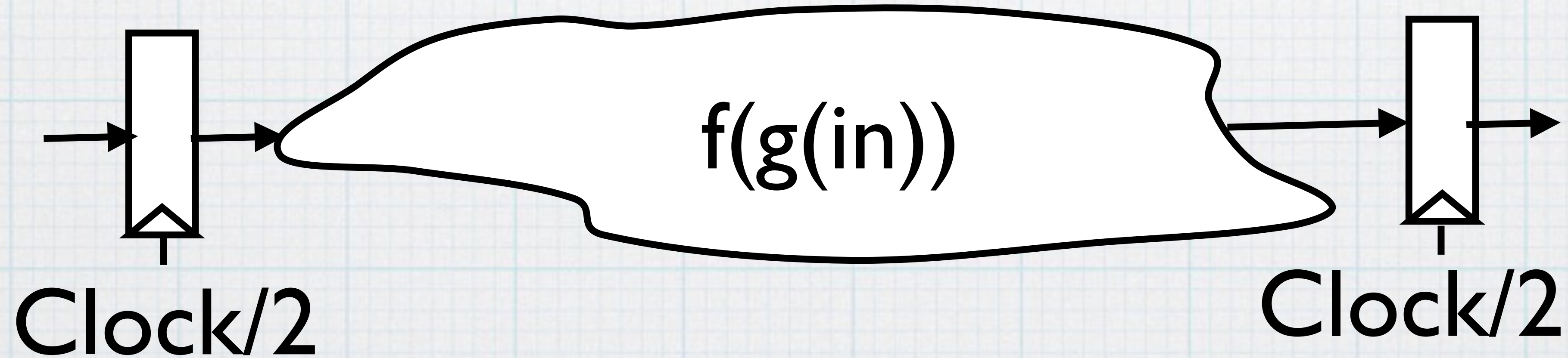
Applicability: Operators where high throughput is not required, or if latency is critical (in which case, replicate to increase throughput).

Consequences: Associated controller has to track execution of operation across multiple cycles. CAD tools might detect false critical path in block.

Multicycle Operator







Some Families of Patterns

Multiport Memory Patterns

Provides multiple access ports to a common memory

- True Multiport Memory
- Banked Multiport Memory
  - Interleave lesser-ported banks to provide higher bandwidth
- Cached Multiport Memory
  - Use large single-port main memory, but add cache to service requests for each access port

Controller Patterns

Synchronous control of local datapath

- State Machine Controller
  - control lines generated by state machine
- Microcoded Controller
  - single-cycle datapath, control lines in ROM/RAM
- In-Order Pipeline Controller
  - pipelined control, dynamic interaction between stages
- Out-of-Order Pipeline Controller
  - operations within a control stream might be reordered internally
- Threaded Pipeline Controller
  - multiple control streams one execution pipeline
  - can be either in-order (PPU) or out-of-order

Network Patterns

Connects multiple units using shared resources

- Bus
  - Low-cost, ordered
- Crossbar
  - High-performance
- Multi-stage network
  - Trade cost/performance