Automatic Construction of Program-Optimized FPGA Memory Networks

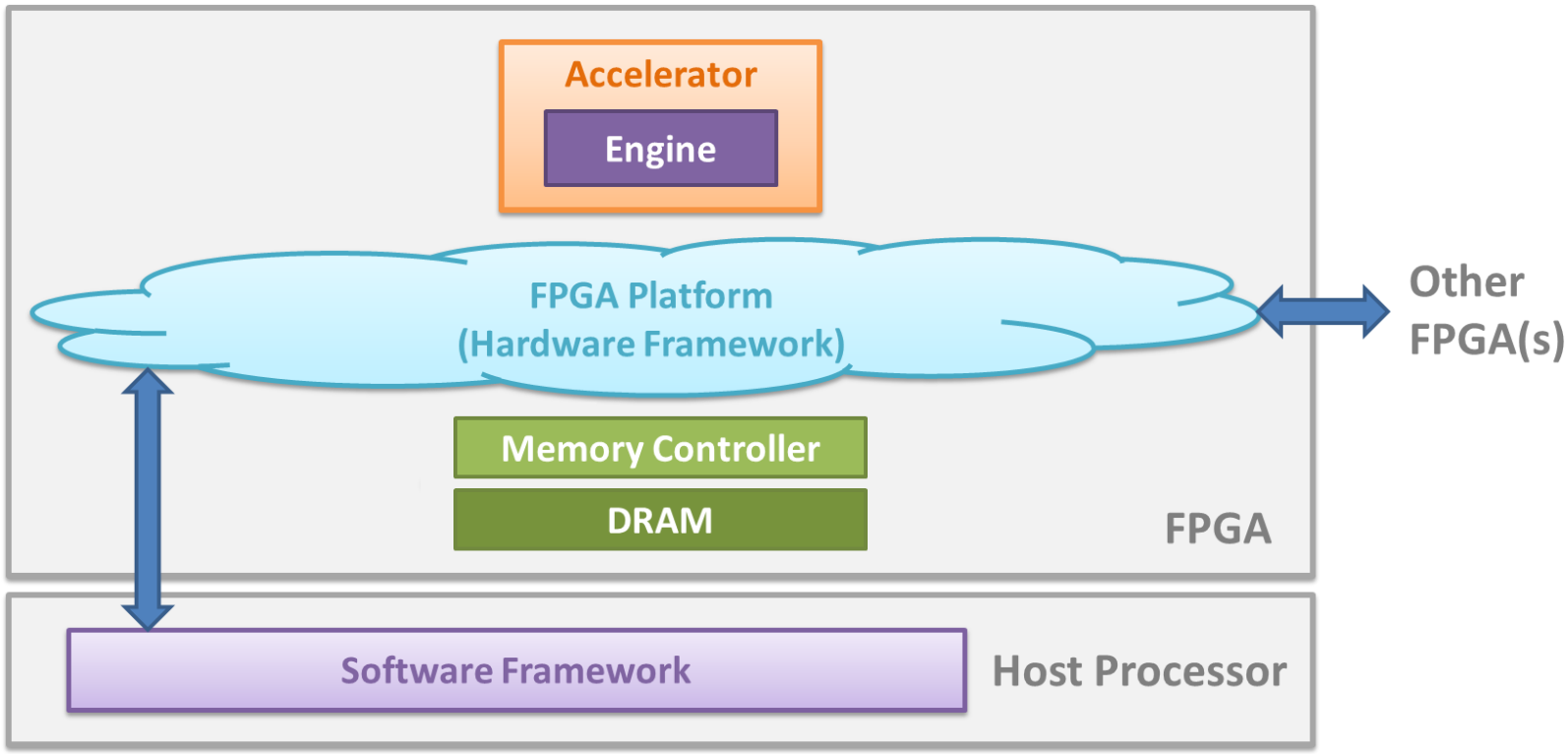
Hsin-Jung Yang[†], **Kermin E. Fleming**[‡], **Felix Winterstein**[§], **Annie I. Chen**[†], **Michael Adler**[‡], **and Joel Emer**[†]

[†] Massachusetts Institute of Technology,
[‡] Intel Corporation, [§] Imperial College London

February 23rd, FPGA 2017

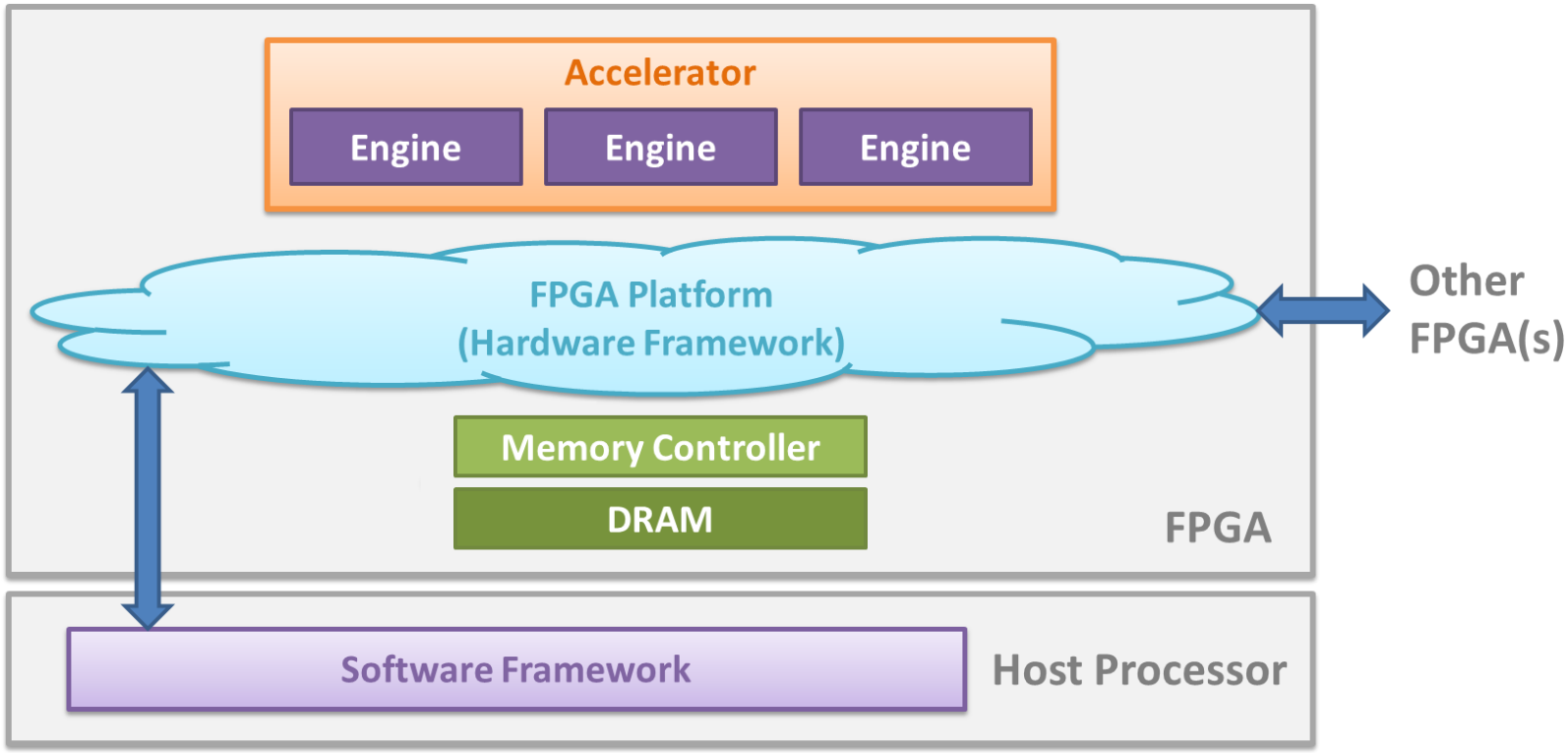
Motivation

- **FPGA applications are getting more complicated**



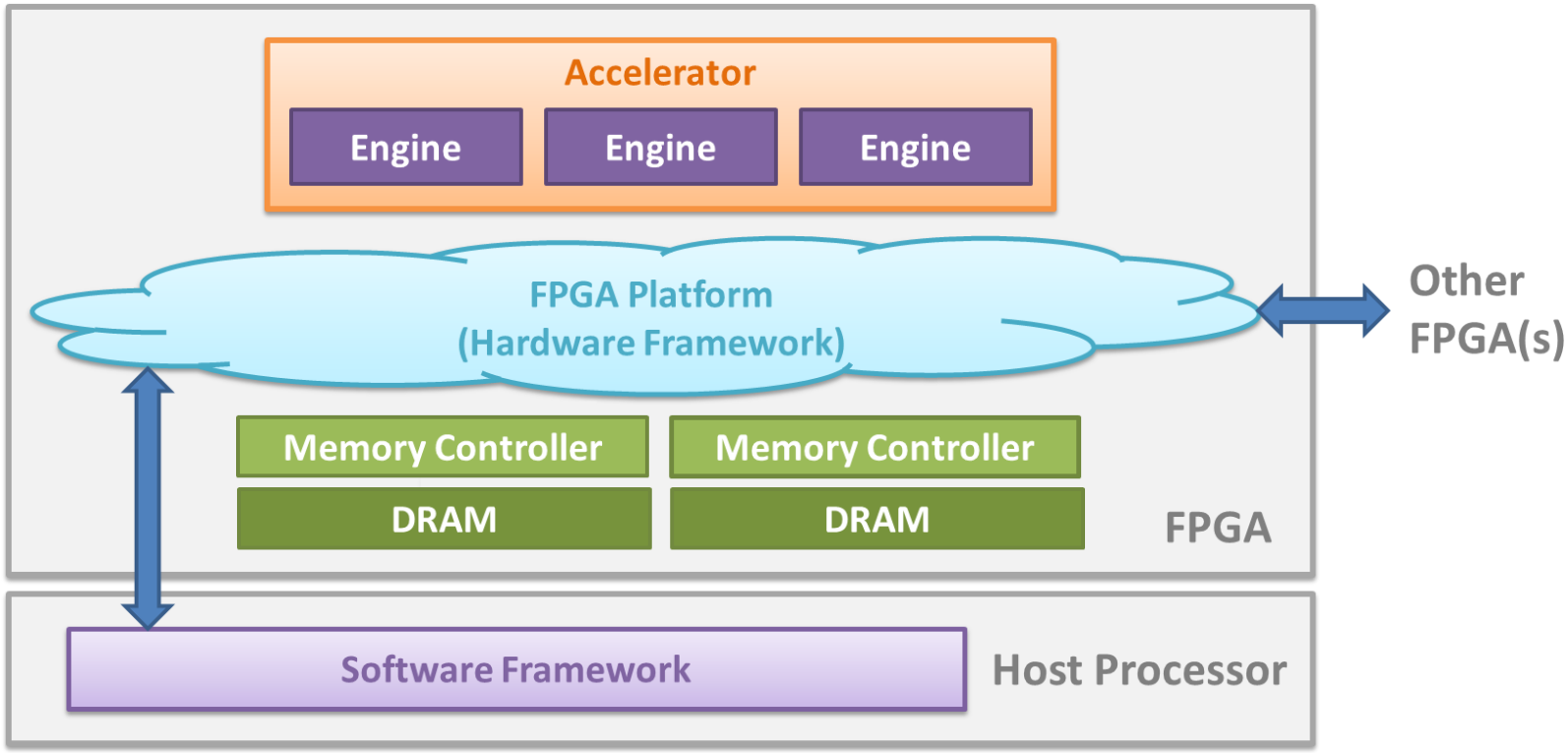
Motivation

- **FPGA applications are getting more complicated**
  - More transistors
  - More engines



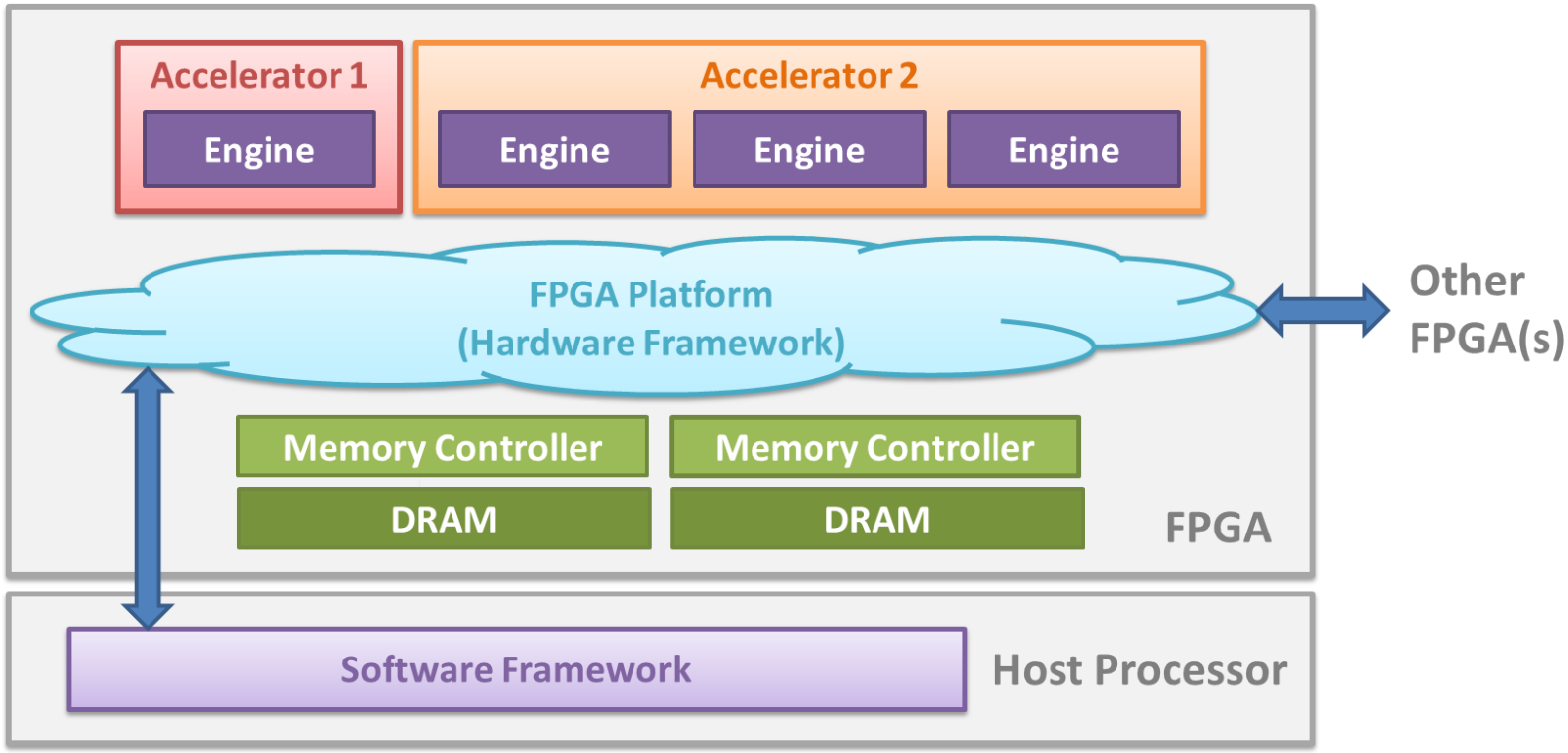
Motivation

- **FPGA applications are getting more complicated**
  - More transistors
  - More engines
  - Multiple memory controllers



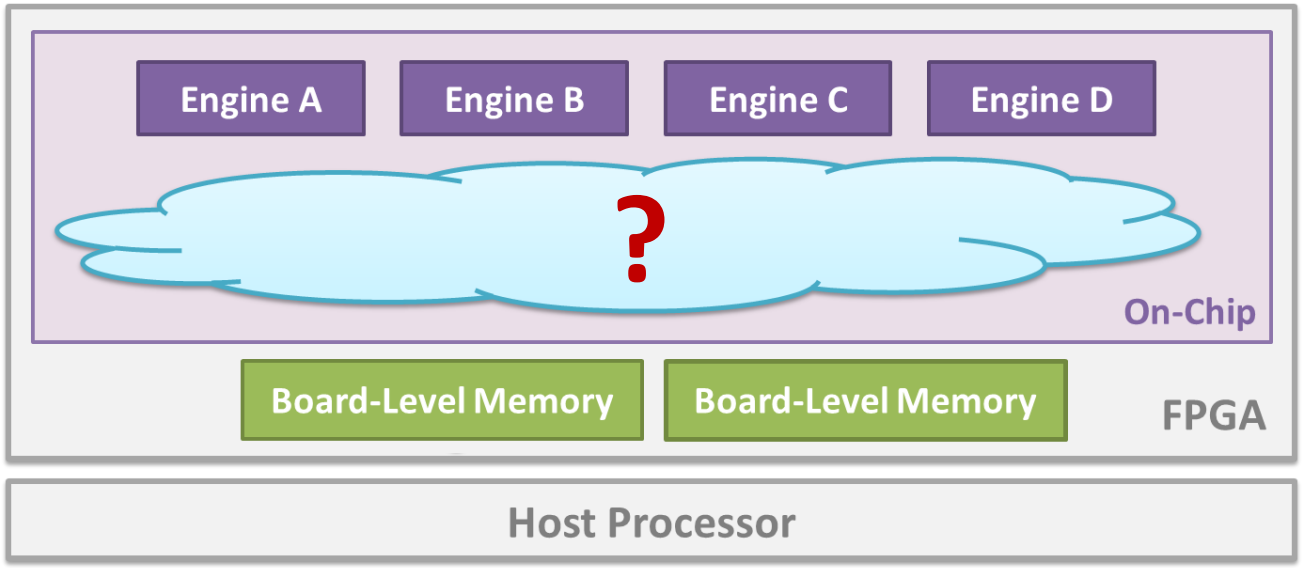
Motivation

- **FPGA applications are getting more complicated**
  - More transistors
  - More engines
  - Multiple memory controllers
  - Multiple programs



Customizing FPGA Platform

- How to connect computational engines to board-level memories in order to maximize program performance?

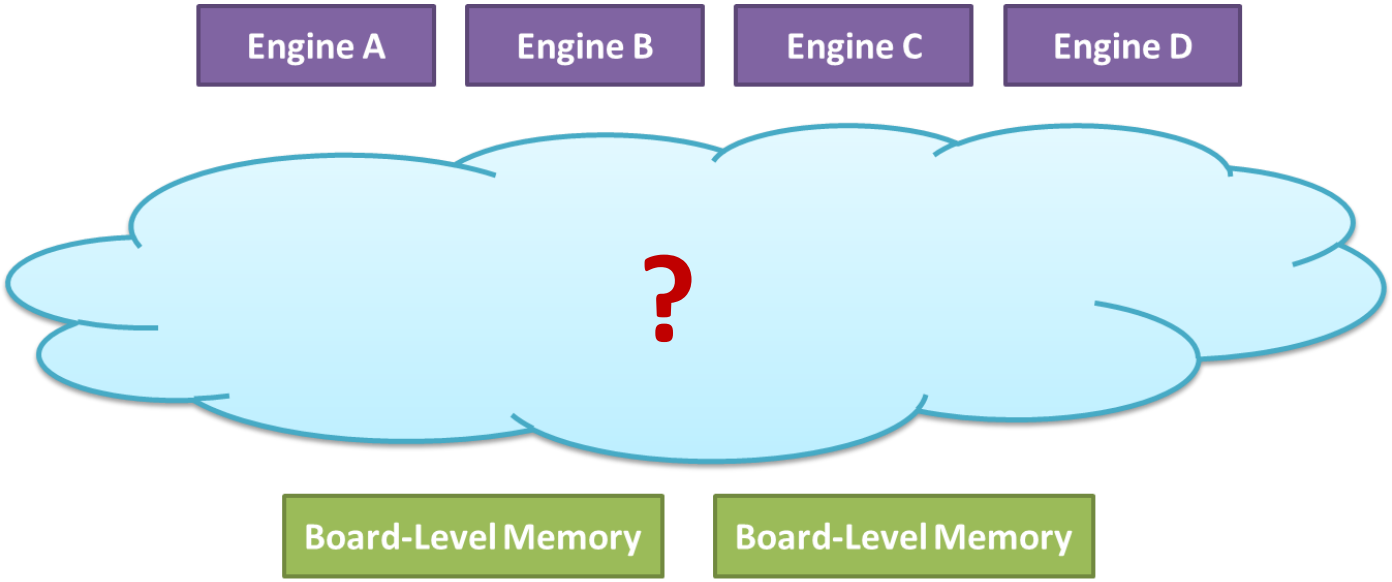


  - On-chip caching
  - Network topology: latency, bandwidth

High design complexity!

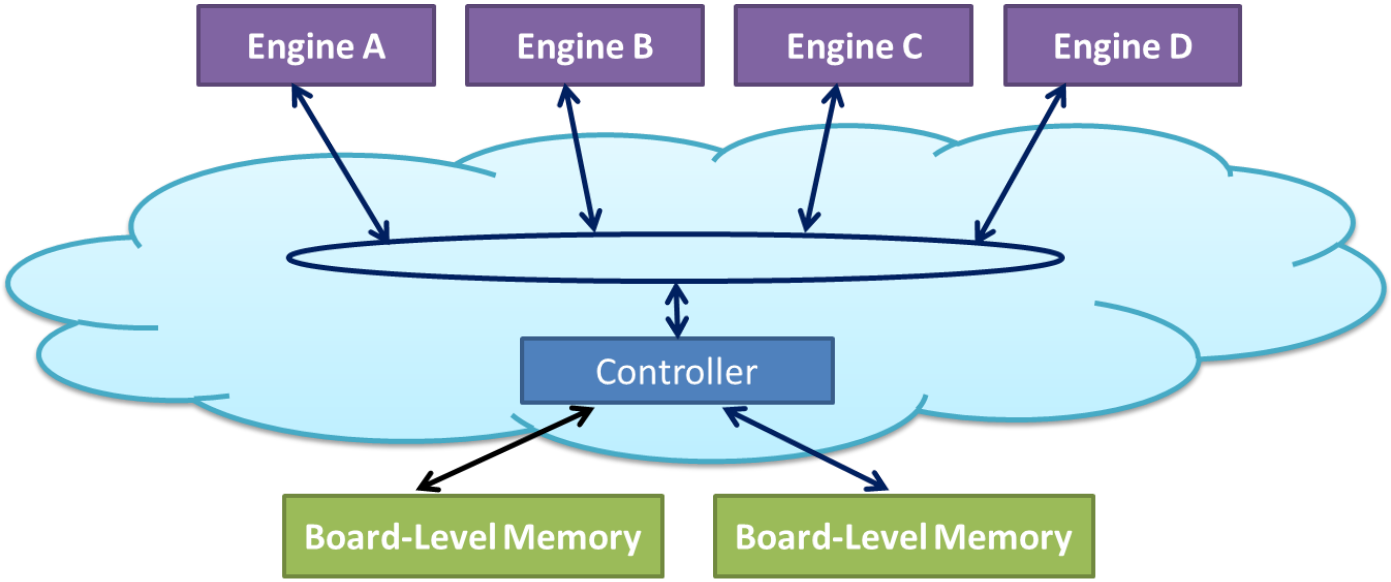
Customizing FPGA Platform

- How to connect computational engines to board-level memories in order to maximize program performance?
  - High design complexity: caching, network,…



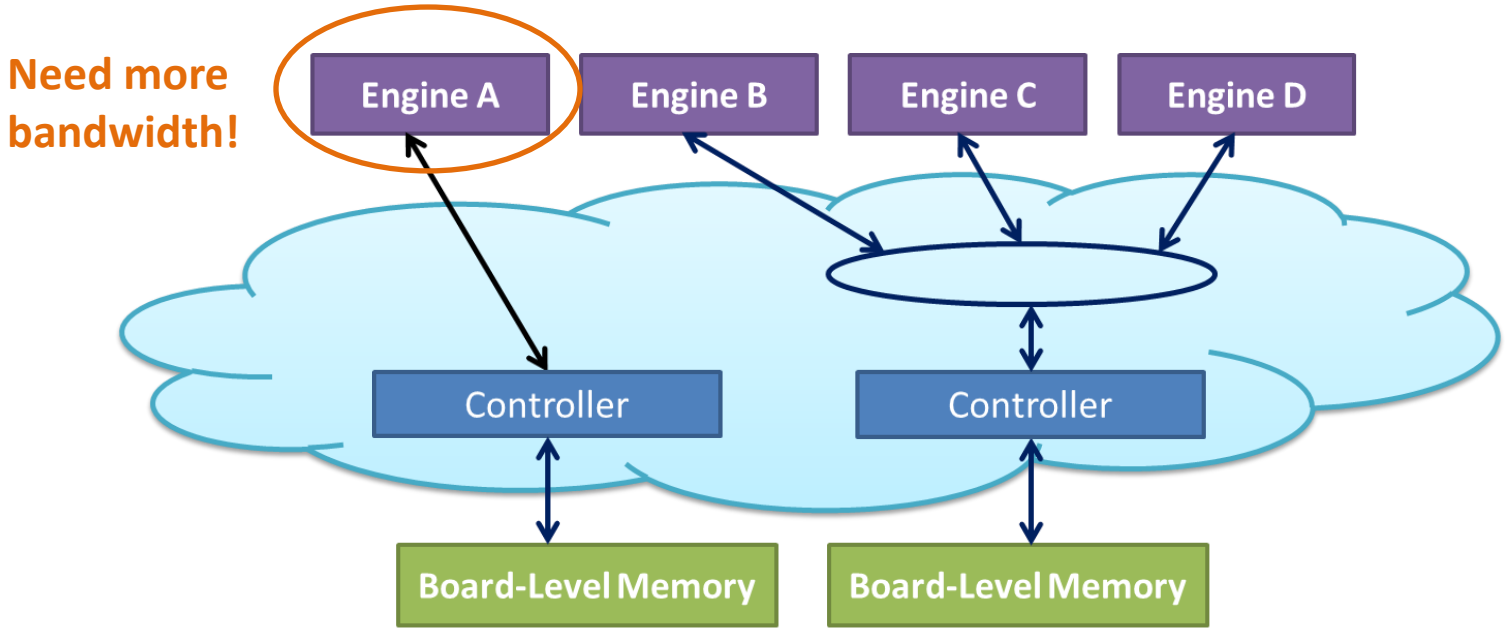
Customizing FPGA Platform

- How to connect computational engines to board-level memories in order to maximize program performance?
  - High design complexity: caching, network,…



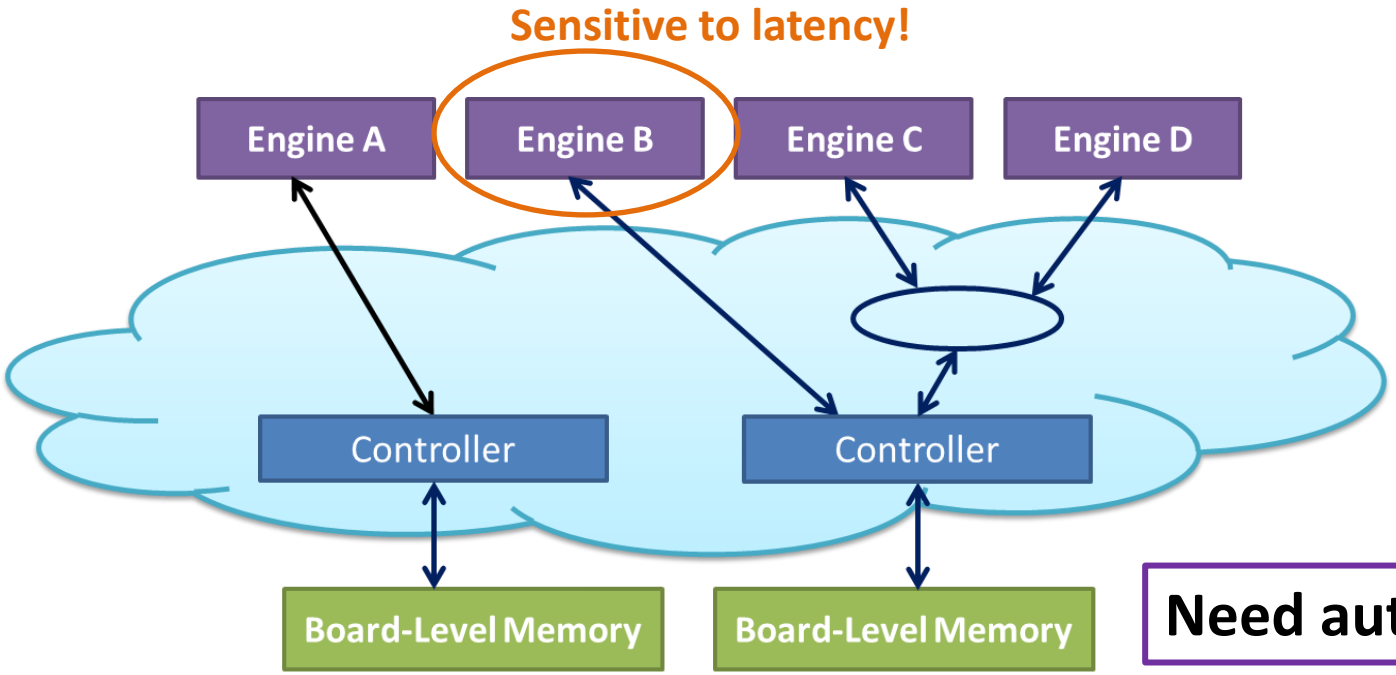
Customizing FPGA Platform

- How to connect computational engines to board-level memories in order to maximize program performance?
  - High design complexity: caching, network,…
- Applications have different memory behavior



Customizing FPGA Platform

- How to connect computational engines to board-level memories in order to maximize program performance?
  - High design complexity: caching, network,…
- Applications have different memory behavior

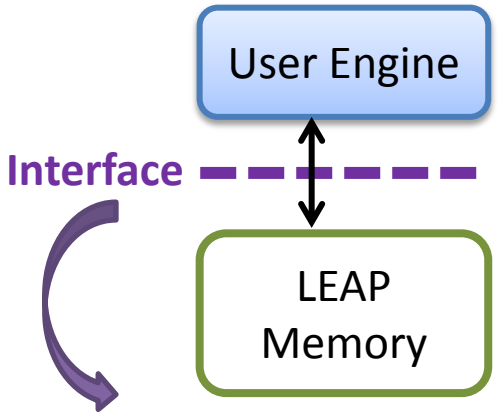


Need automation!

Automatic Construction of Program-Optimized Memories

- **A clearly-defined, generic memory abstraction**
  - Separate the user program from the memory system implementation
- **Program introspection**
  - Understand the program's memory behavior
- **A resource-aware, feedback-driven memory compiler**
  - Use introspection results as feedback to automatically construct the "best" memory system for the target program and platform

LEAP Memory Abstraction

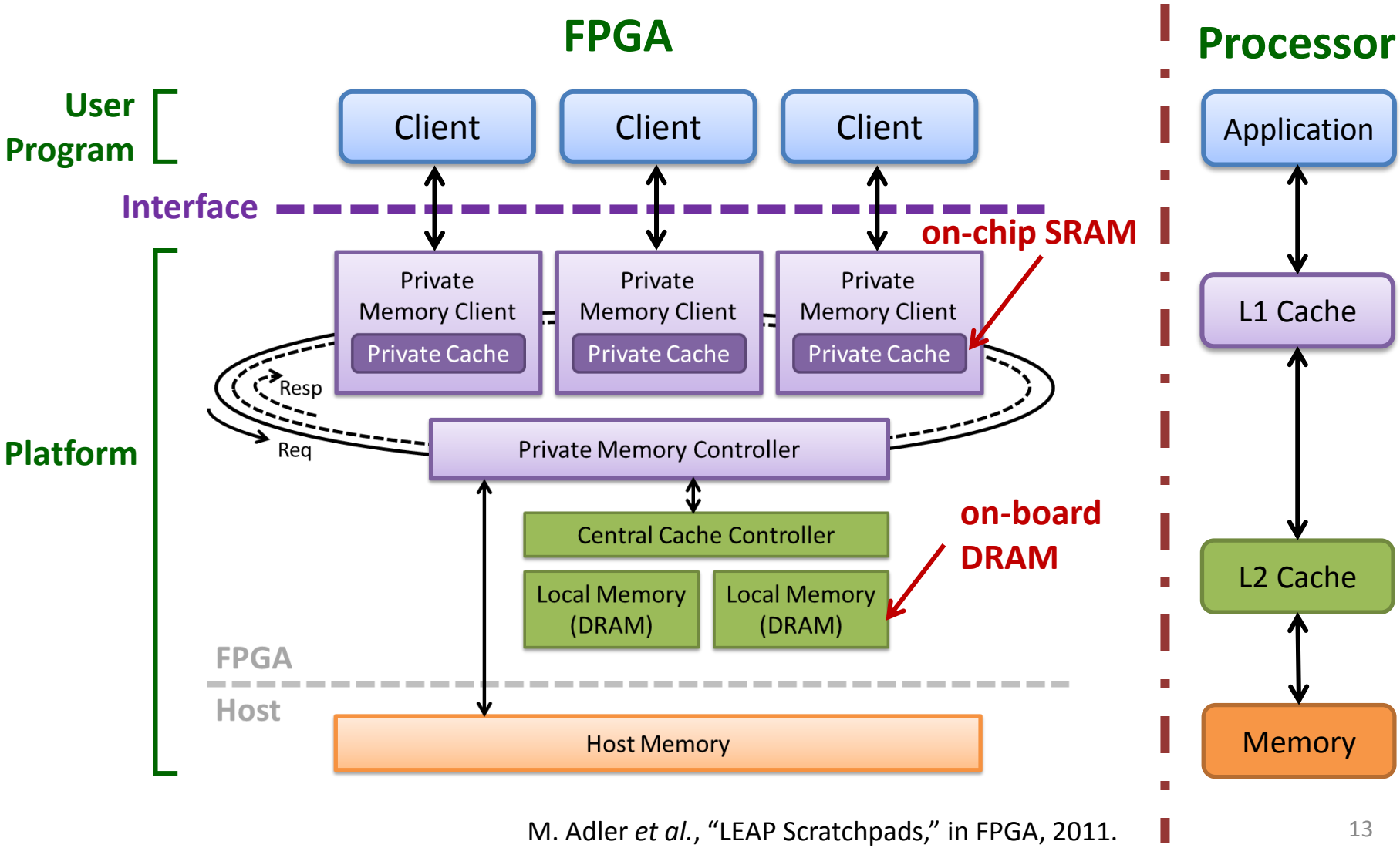


LEAP memory block

- Simple memory interface
- Arbitrary data size
- Private address space
- **"Unlimited" storage**
- **Automatic caching**

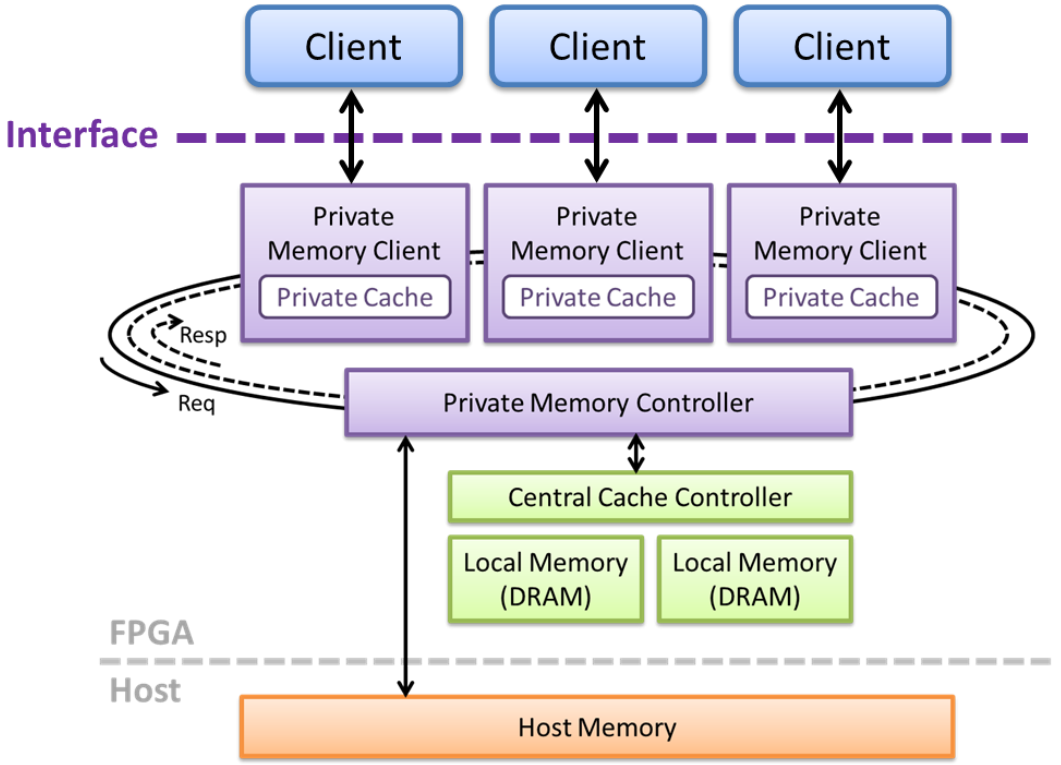
```
interface MEM_IFC#(type t_ADDR, type t_DATA)
    method void readReq(t_ADDR addr);
    method void write(t_ADDR addr, t_DATA din);
    method t_DATA readResp();
endinterface
```

Baseline LEAP Private Memory



M. Adler *et al.*, "LEAP Scratchpads," in FPGA, 2011.

Baseline LEAP Private Memory



- **Simplicity**
- **Cache capacity scales with the increasing number of DRAMs**

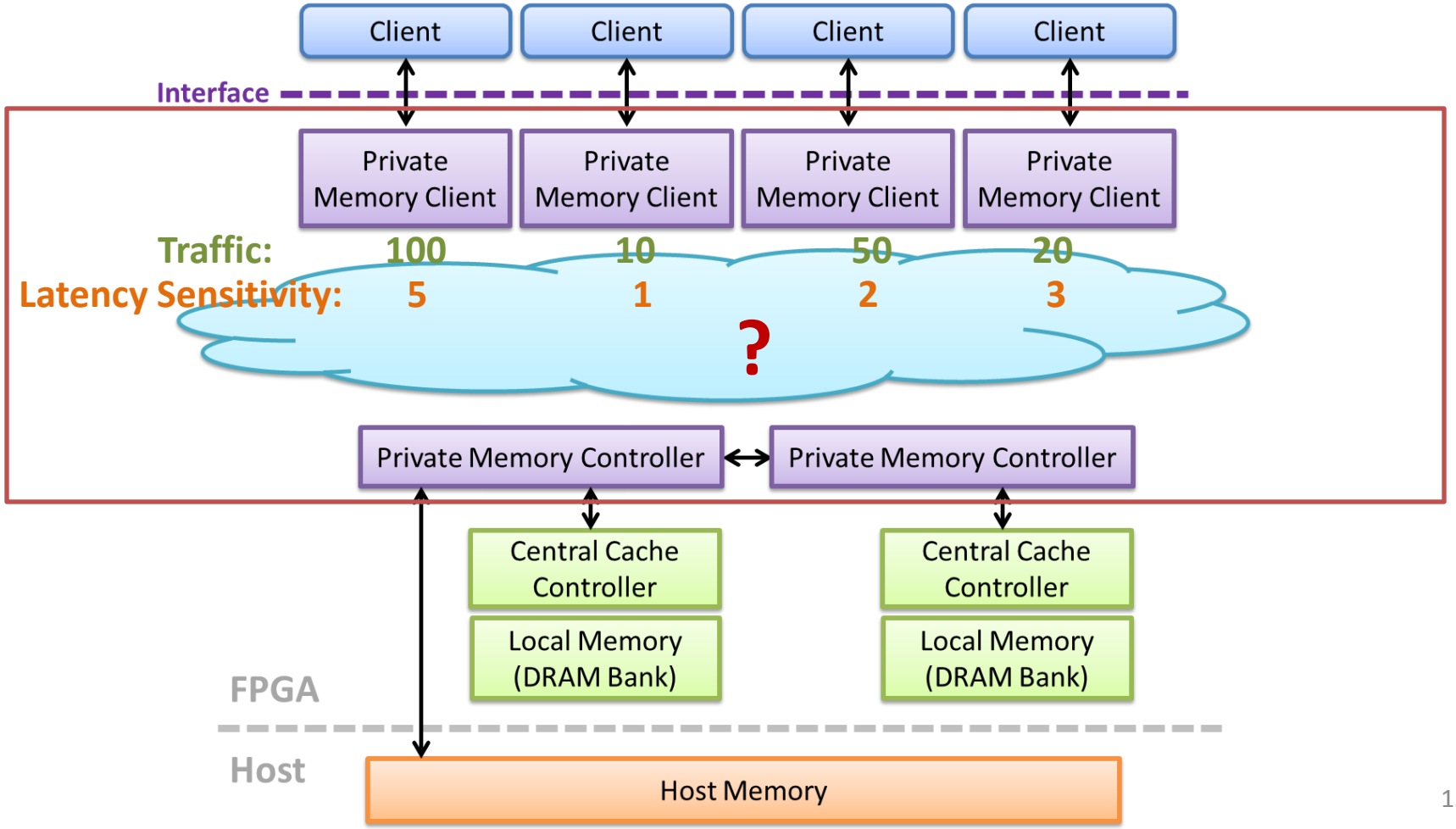
Difficulty: Performance is limited

- **Limited bandwidth**
- **Long latency for large rings**

Can we do better?

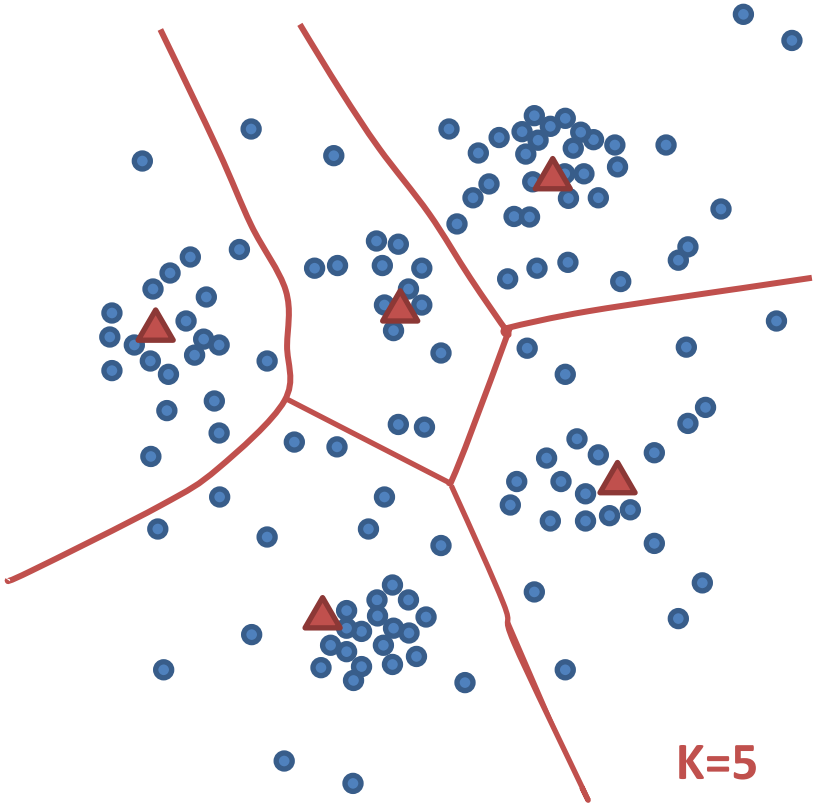
Customizing LEAP Memory Network

- **Distributed memory controllers**



Motivating Example #1

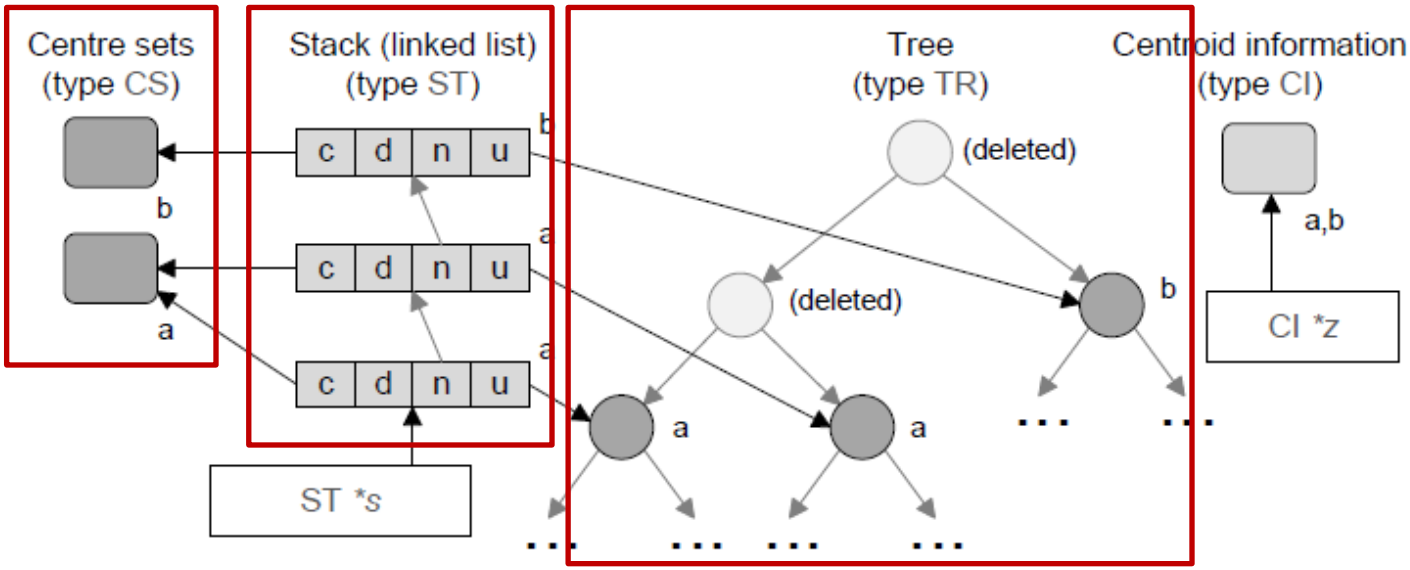
- Filtering algorithm for K-means clustering



Motivating Example #1

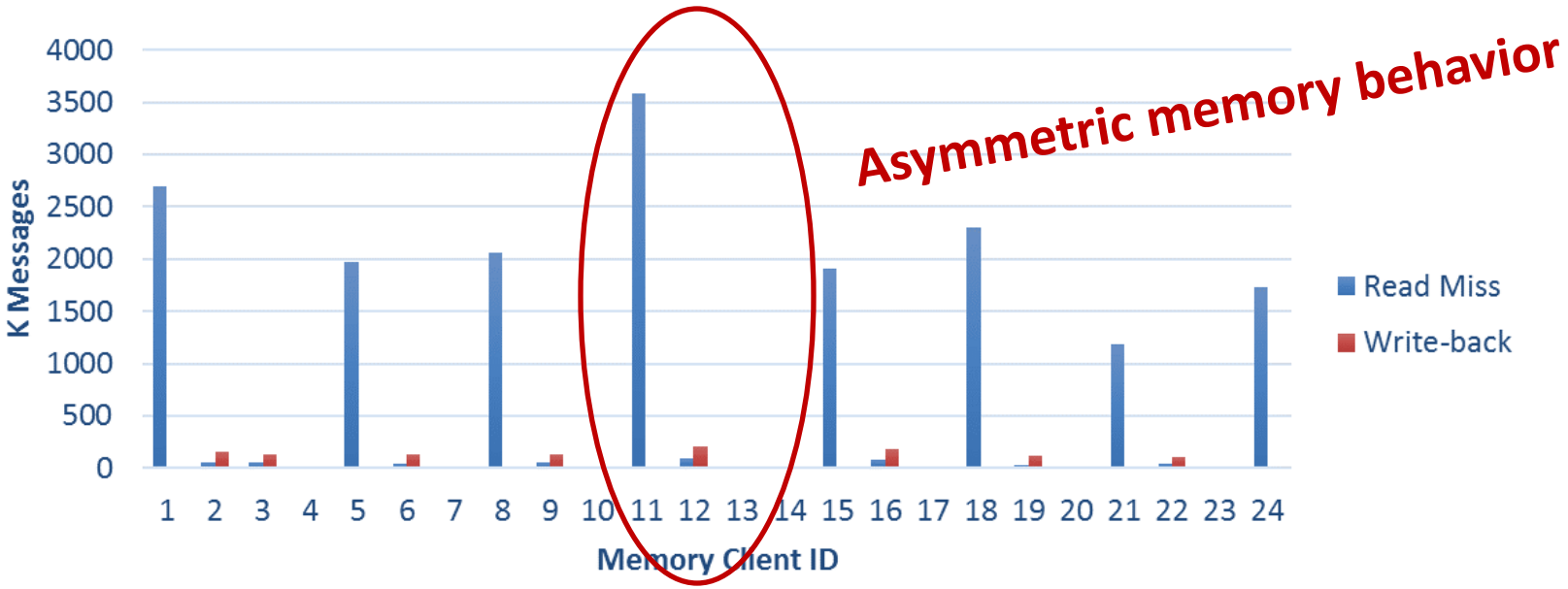
- **Filtering algorithm for K-means clustering (HLS kernel)**
  - 3 different data structures
  - 8 parallel partitions,
    24 LEAP memory clients in total

Three data structures:
(1) Tree nodes (low locality)
(2) Center sets (high locality)
(3) Stack (very high locality)



Motivating Example #1

- **Filtering algorithm for K-means clustering**
  - Program introspection: number of network messages



Three types of memory clients:

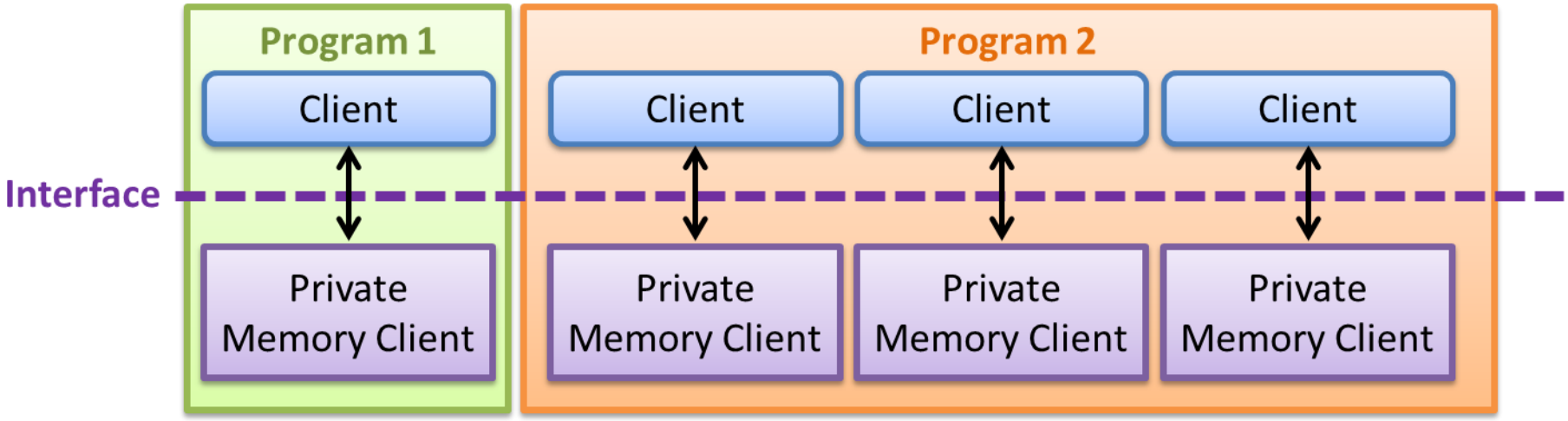
(1) Lots of read misses, few write-backs (tree nodes)

(2) More write-backs than read misses (center sets)

(3) No messages at all (stacks)

Motivating Example #2

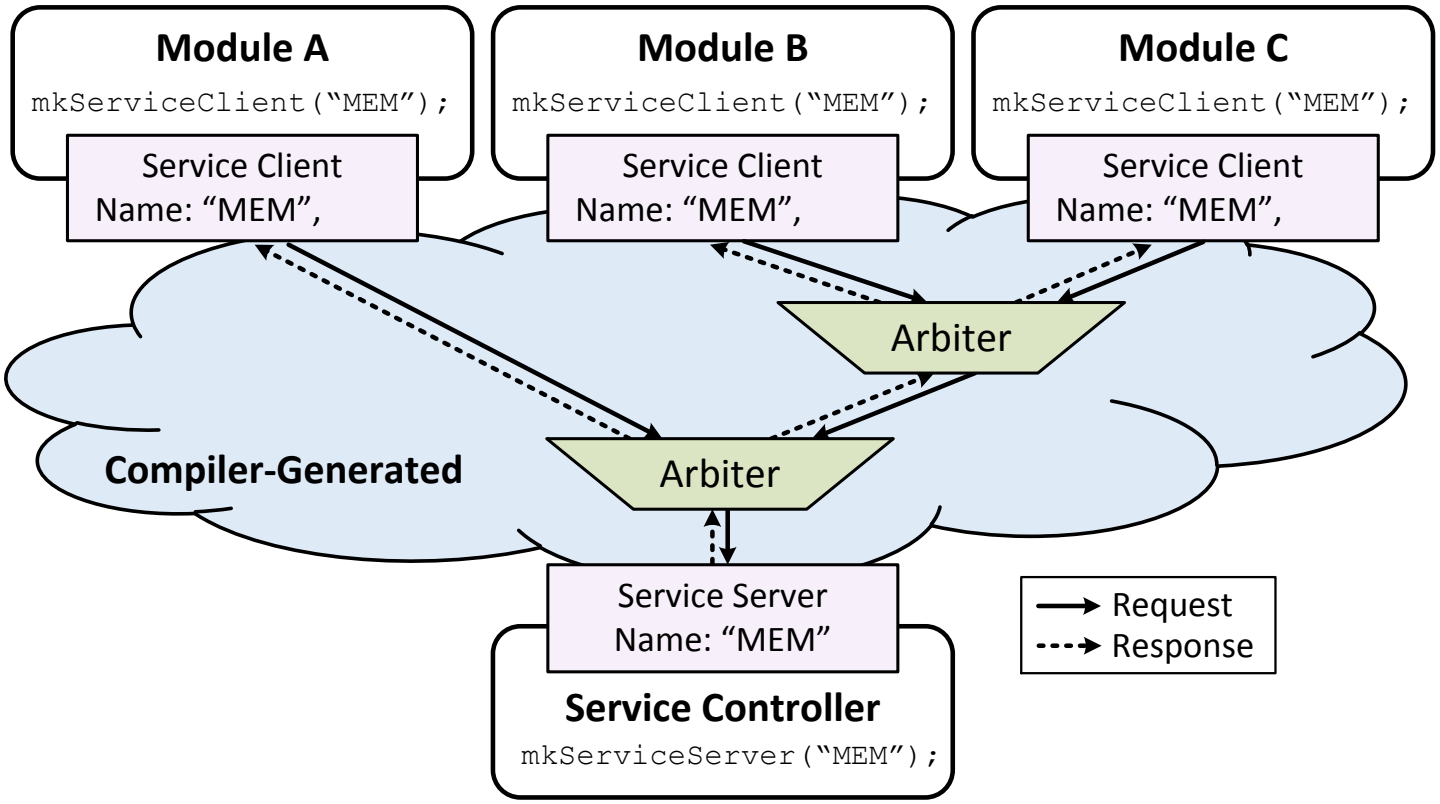
- **FPGA virtualization: mapping multiple programs on FPGA**



  - Different programs are likely to have different behavior
  - May need some quality-of-service (QoS) control

Communication Abstraction

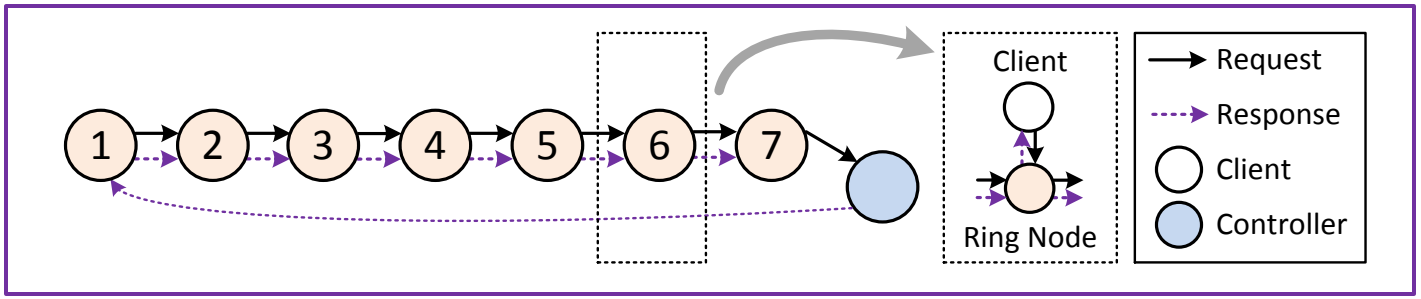
- **Service connection**
  - A new communication abstraction for centralized services
    - Enabling compilers to freely pick interconnect topology



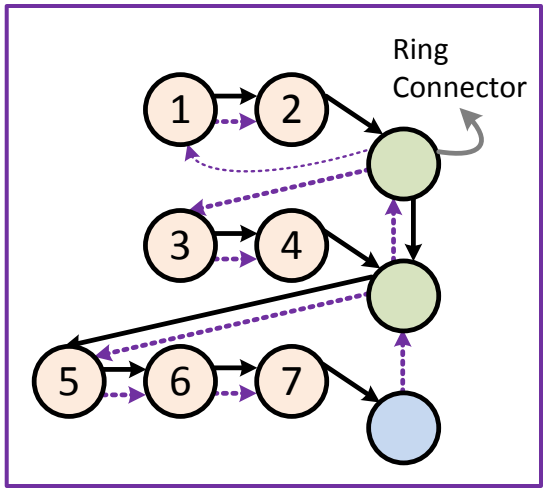
Compiler-Generated Network Topologies

Single Ring

Low complexity, long latency

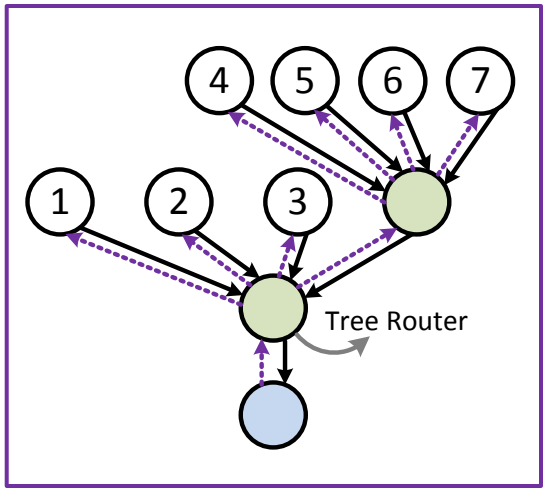


Hierarchical Ring



Shorter latency, larger area

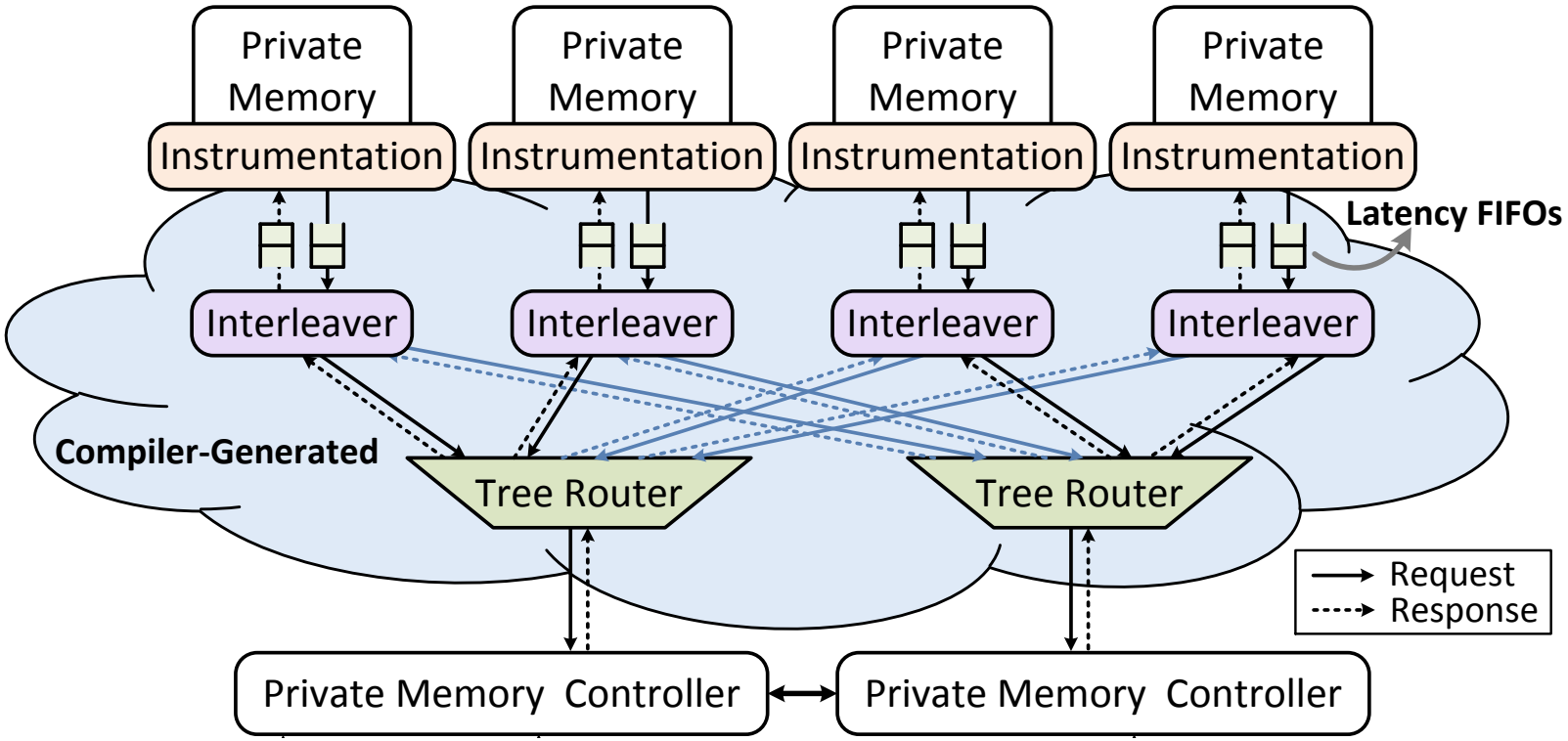
Tree



Highest complexity, shortest latency

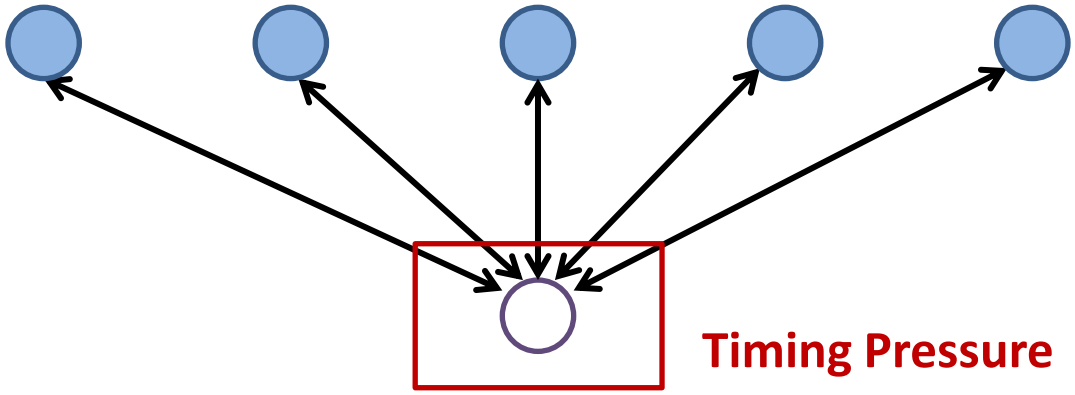
Network Profiler

- **Goal:** to emulate different networks in a single compilation
  - Network partitioning, latency and bandwidth are all dynamically configurable



Tree-Based Network

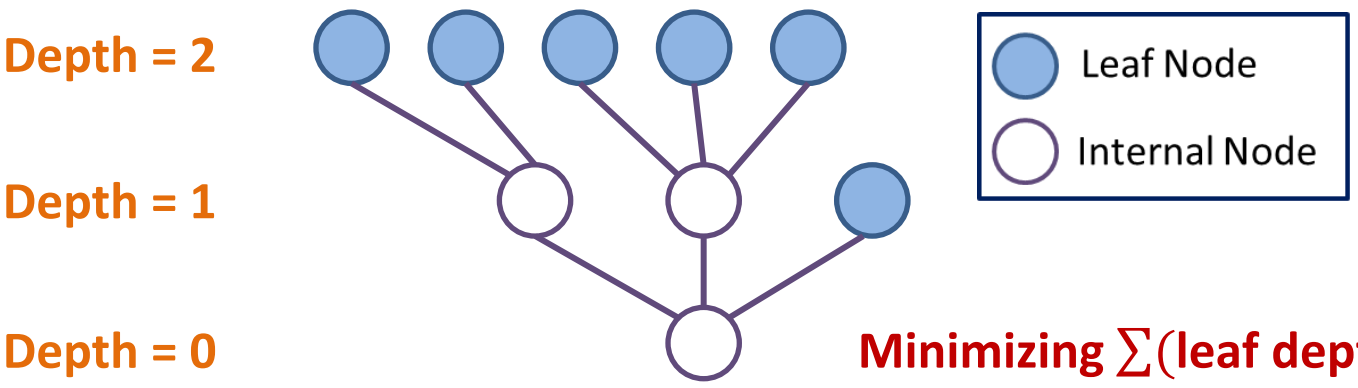
- **Construct a tree network that maximizes performance**
  - Ideal case:



  - More children per node, larger timing pressure on routers
  - Fix K = max(#children per node) given a target frequency

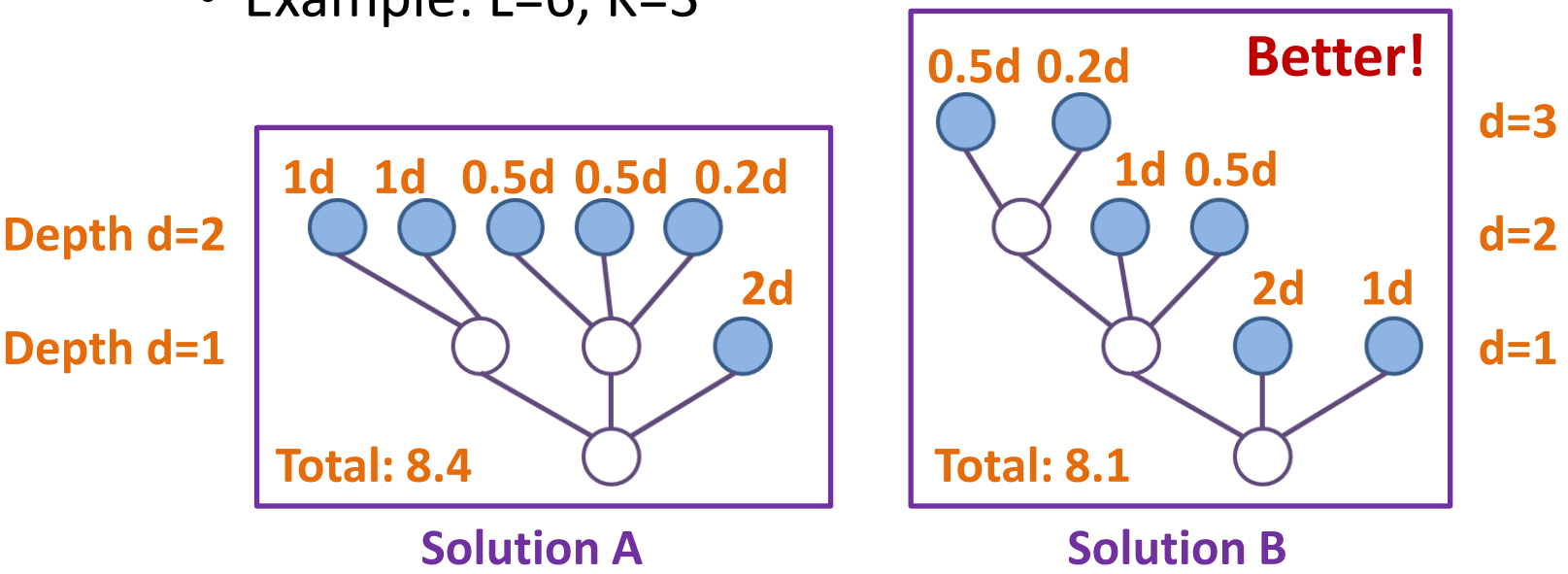
Tree-Based Network

- **Construct a K-ary tree that maximizes performance**
  - Given L: number of leaves (clients)
    K: max number of children per node
  - **Case 1: clients with homogeneous behavior**
    - **Solution:** build a balanced tree with the minimum number internal nodes
    - Example: L=6, K=3



Tree-Based Network

- **Construct a K-ary tree that maximizes performance**
  - **Case 2: clients with heterogeneous behavior**
    - Some clients are more sensitive to latency
    - Place latency-sensitive clients closer to root
    - A balanced tree may not be optimal
    - Example: L=6, K=3

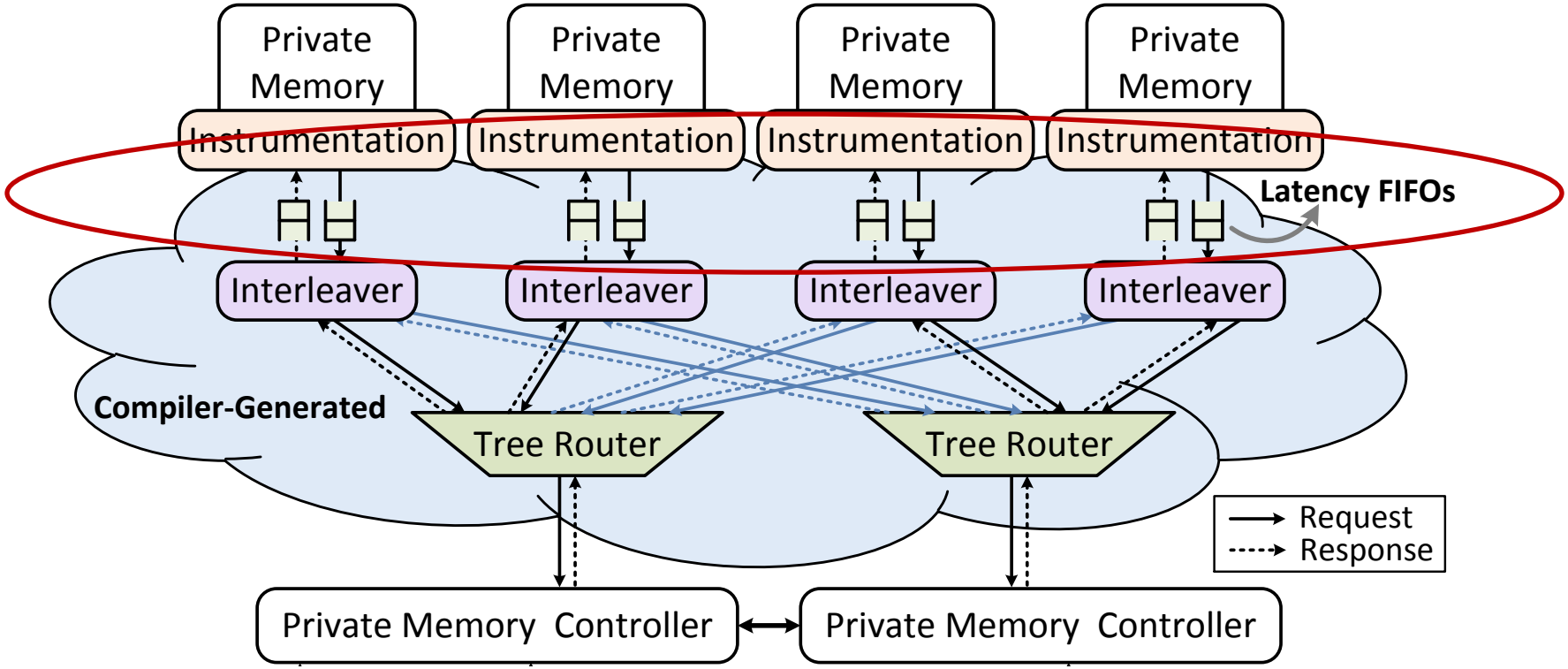


Automatic Construction of Program-Optimized Memories

- **A clearly-defined, generic memory abstraction**
  - Separate the user program from the memory system implementation
- **Program introspection**
  - Understand the program's memory behavior
- **A resource-aware, feedback-driven memory compiler**
  - Use introspection results as feedback to automatically construct the "best" memory system for the target program and platform

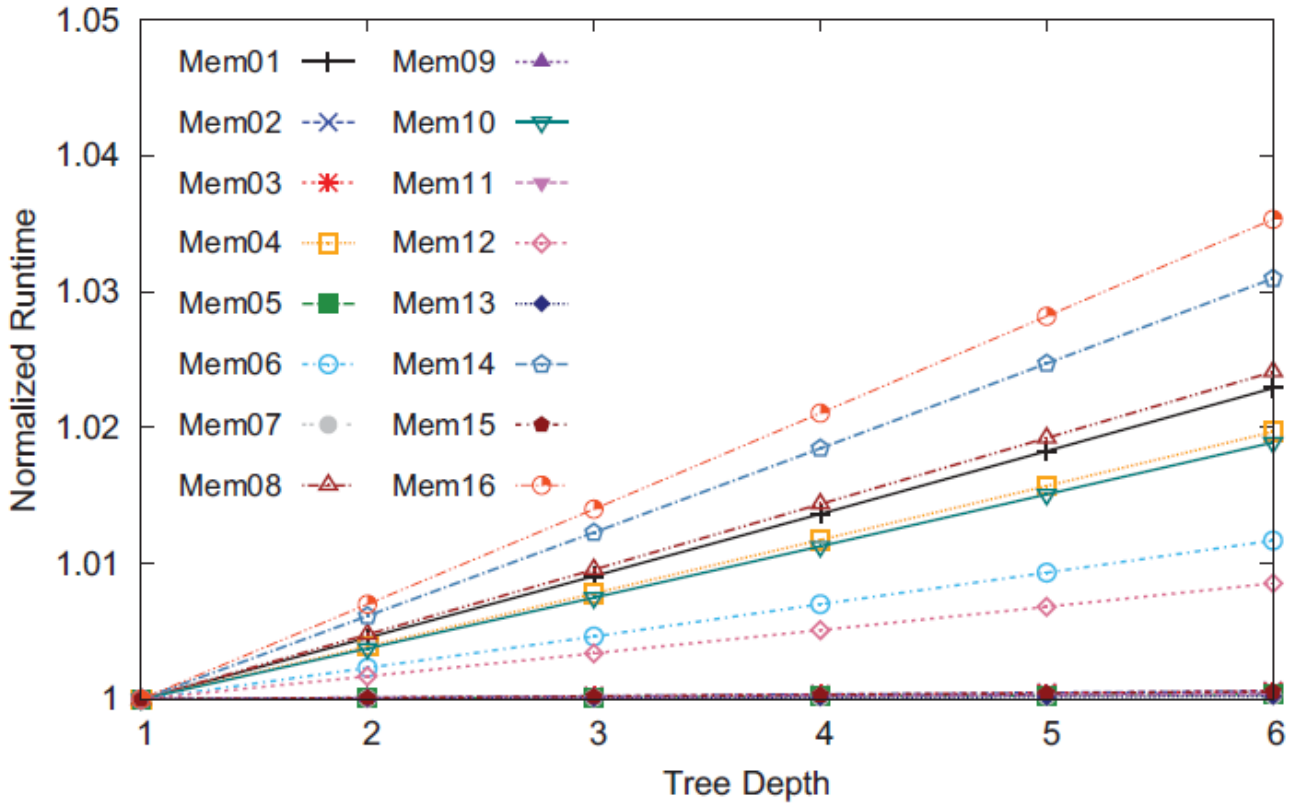
Program Introspection with Network Profiler

- Network profiler measures latency sensitivity per memory client



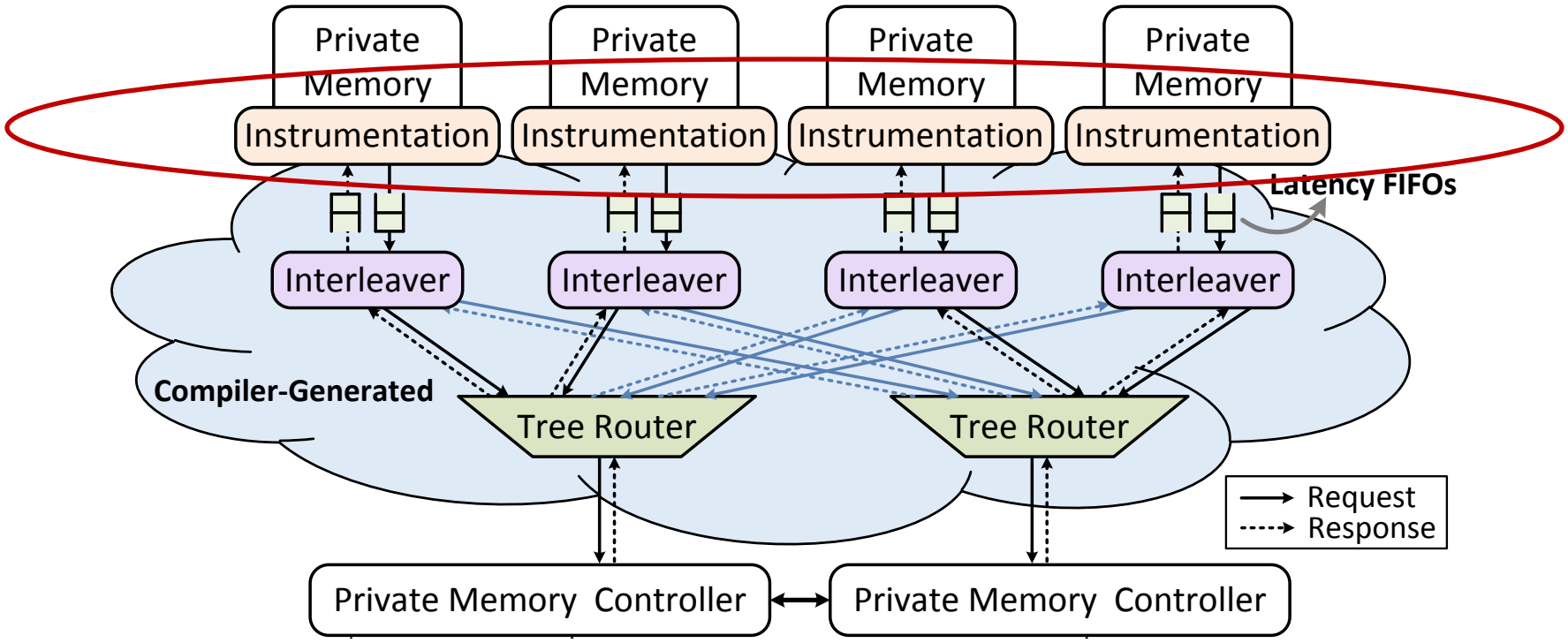
Program Introspection with Network Profiler

- Network profiler measures latency sensitivity per memory client



Program Introspection with Network Profiler

- Instrumentation logic monitors total number of requests, request rates, queueing delays…



Construction of Optimized Cache Networks

- **Profiling compilation**
  - Measure clients' latency sensitivity, bandwidth demands

- **Main compilation: three-stage network construction**
  - **Network partitioning:** (FPGA'16)
    to balance the total traffic among controller networks
  - **Topology selection with client placement:** to minimize the network latency impact on program performance

H.-J. Yang *et al.*, "LMC: Automatic Resource-Aware Program-Optimized Memory Partitioning," in FPGA, 2016.

Optimized Tree Construction

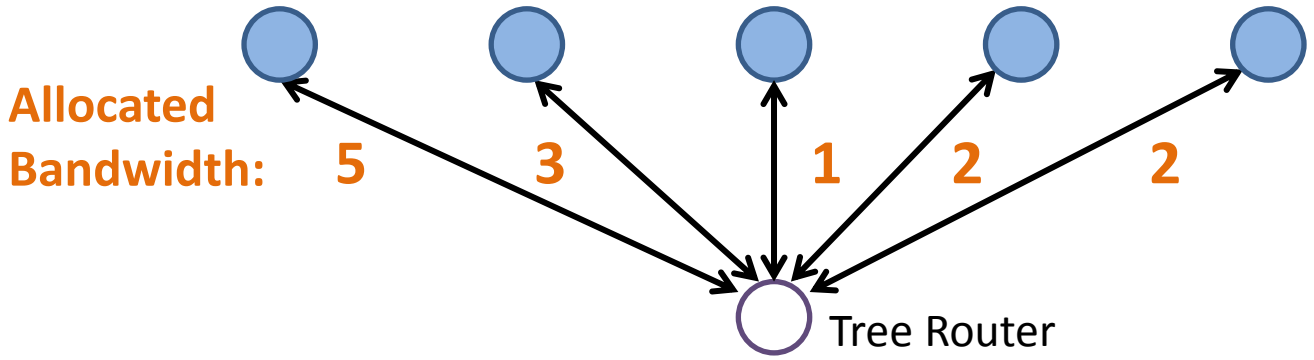
- **Construct a K-ary tree minimizing the total tree weights**
  - **Given** N: # leaves, K: max # children, D: max tree depth,
    w_{nd}: weight of leaf n at depth d
  - **Variables:** $\lambda_{nd} \in \{0,1\}$: whether leaf n is at depth d
    $x_d \in \mathbb{Z}_{\geq 0}$: # leaves at depth d
    $y_d \in \mathbb{Z}_{\geq 0}$: # internal nodes at depth d
  - **Integer linear programming (ILP) Problem:**

$$\min_{\lambda,x,y} \sum_{n=1}^{N} \sum_{d=1}^{D} \lambda_{nd} w_{nd}$$

$$\text{s.t.} \sum_d \lambda_{nd} = 1, \forall n$$
$$x_d = \sum_n \lambda_{nd}, \forall d$$
$$y_d + x_d = K \cdot y_{d-1}, \forall d$$
$$y_0 = 1 \text{ (root)}$$

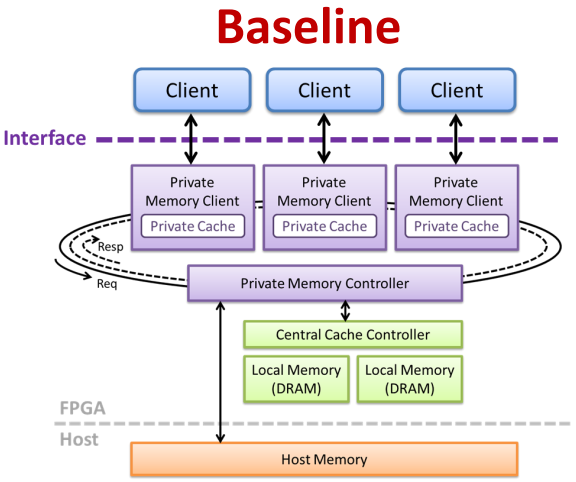
Construction of Optimized Cache Networks

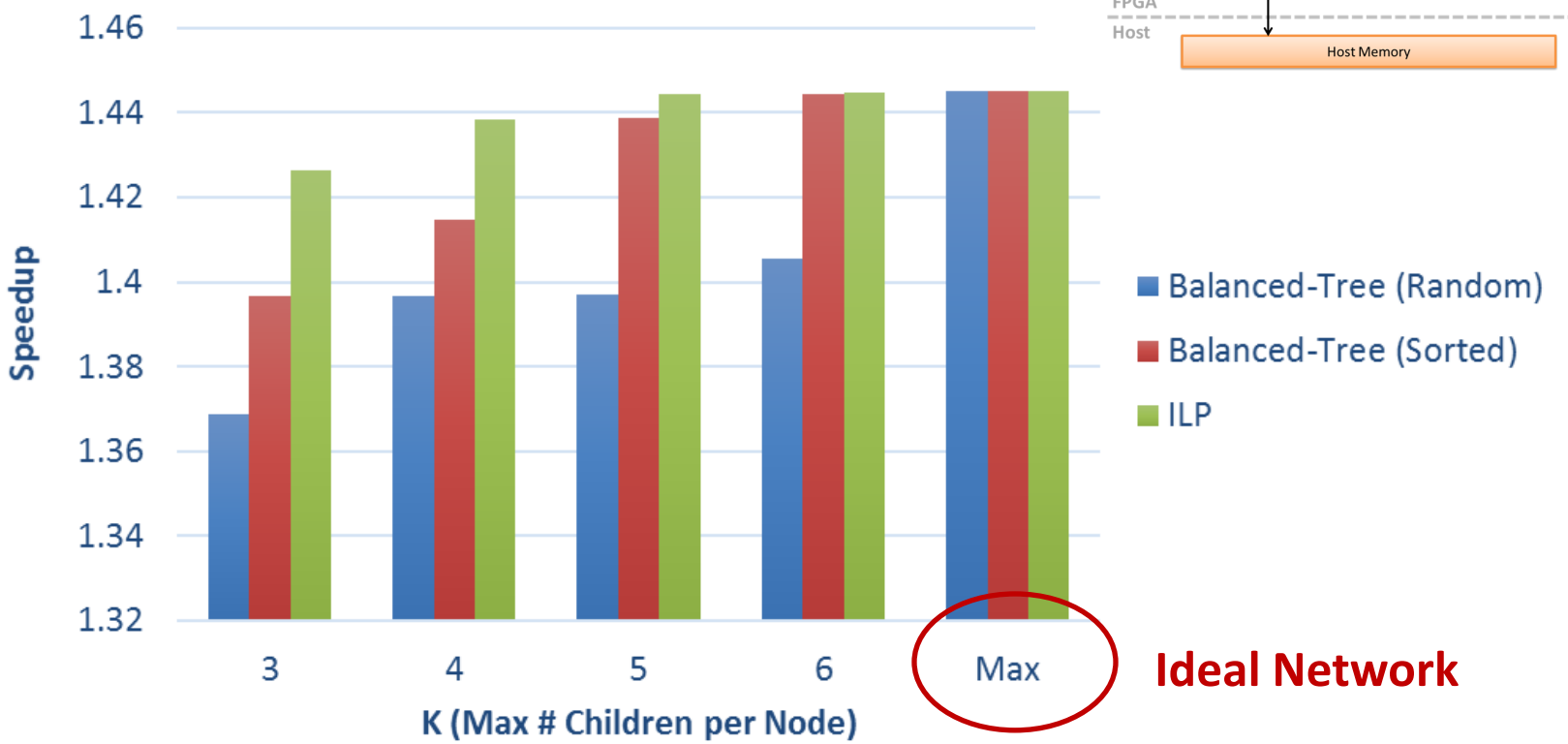
- **Profiling compilation**
- **Main compilation: three-stage network construction**
  - **Network partitioning** (FPGA'16)
  - **Topology selection with client placement**
  - **Bandwidth allocation:** (for multi-program applications) to control the fairness among multiple programs



Evaluation

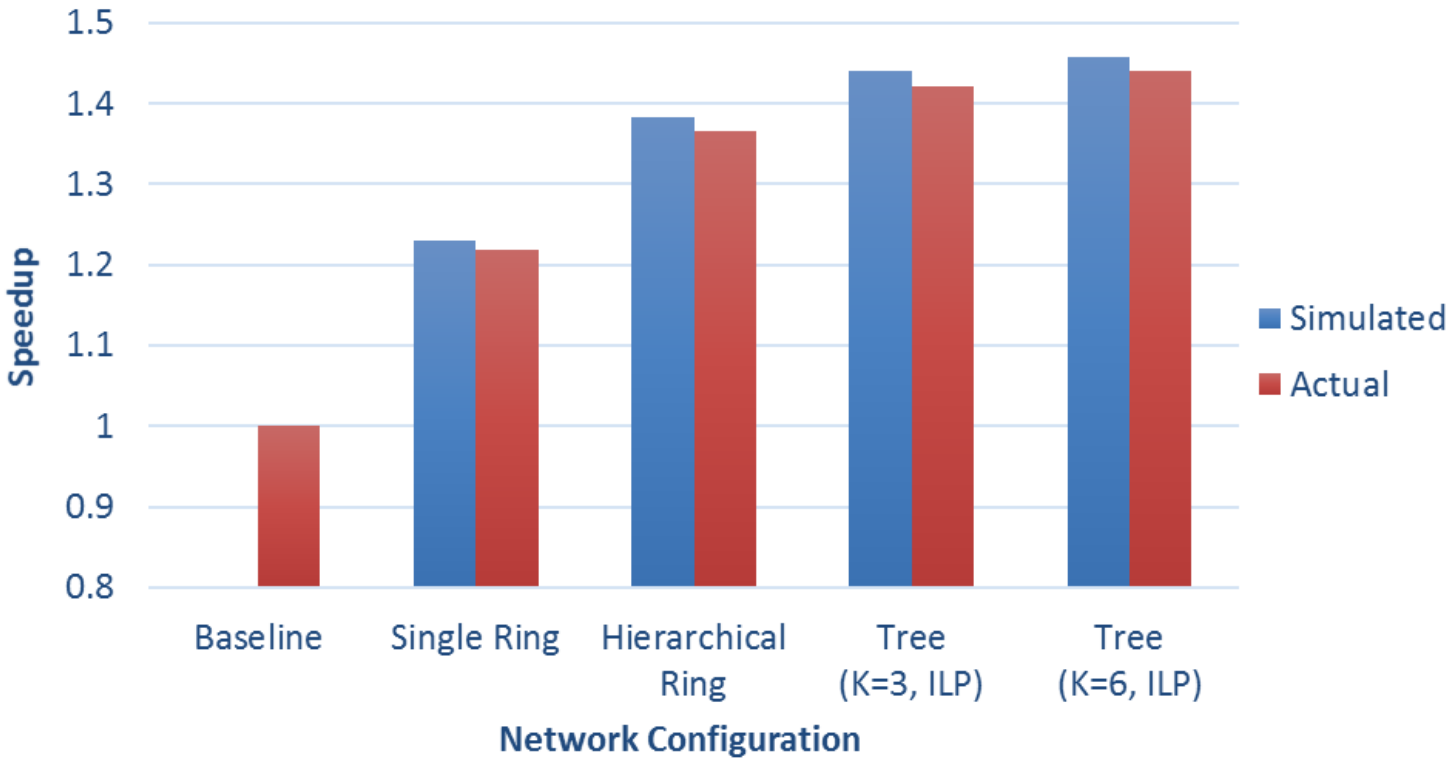
- **Filtering algorithm on VC709**
  - Varying K





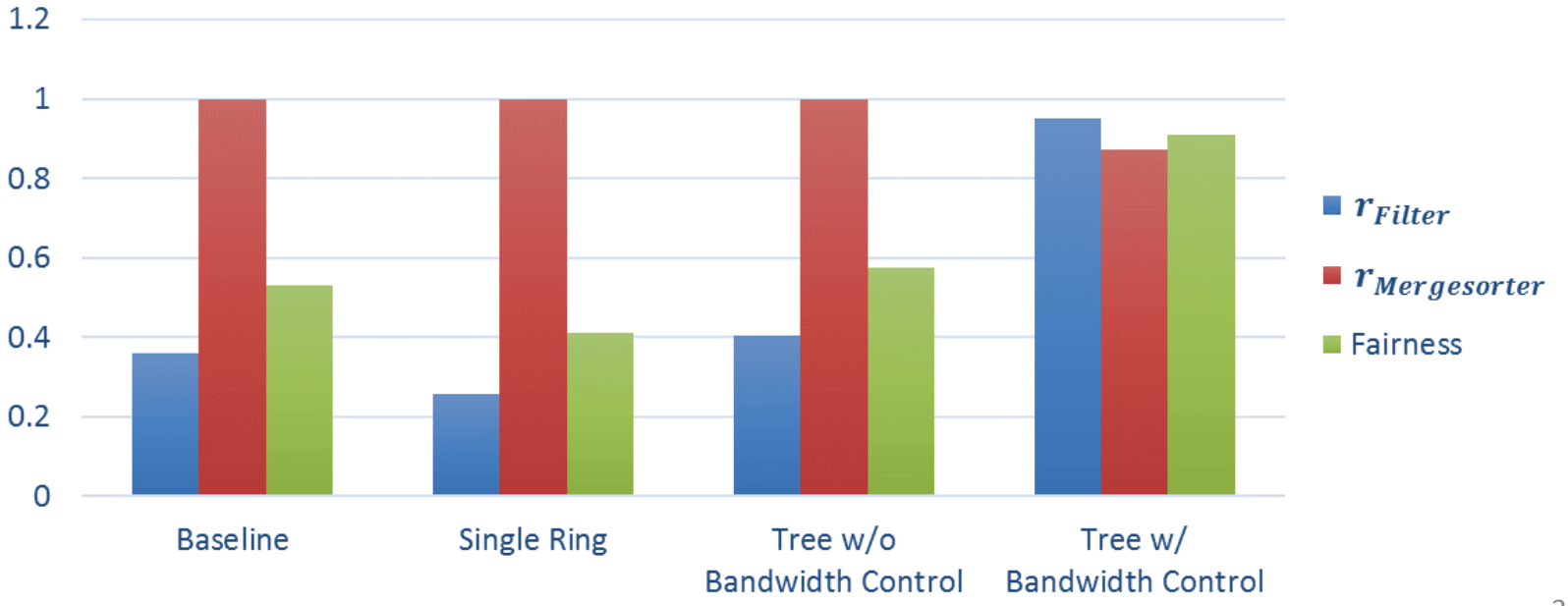
Evaluation

- **Filtering algorithm on VC709**
  - Different network configurations



Virtualizing FPGA

- **Case study: Mergesorter + Filtering algorithm**
  - Mergesorter: 4 LEAP memories (Filter: 24 memories)
  - Performance ratio $r = \frac{Performance_{MP}}{Performance_{SP}}$
  - Fairness $= n/(\sum_{i=1}^{n} {}^{1}/_{r_i})$



Conclusion

- We introduce a feedback-driven compiler that automatically constructs memory networks optimized for the target application.
  - A communication abstraction for centralized services
  - A dynamically configurable network profiler
  - Tree topology selection algorithms
- Future work:
  - Resource-aware memory network optimizations for asymmetric memory controllers