ISA-Independent Workload Characterization and Implications for Specialized Architectures

Yakun Sophia Shao and David Brooks
Harvard University
{shao,dbrooks}@eecs.harvard.edu

Specialized architectures are decoupled from legacy ISAs.

Spectrum of Specialization: *General-Purpose CPU* --------> *GPU* --------> *Fixed-Function ASIC*

Low Efficiency ⟷ High Efficiency

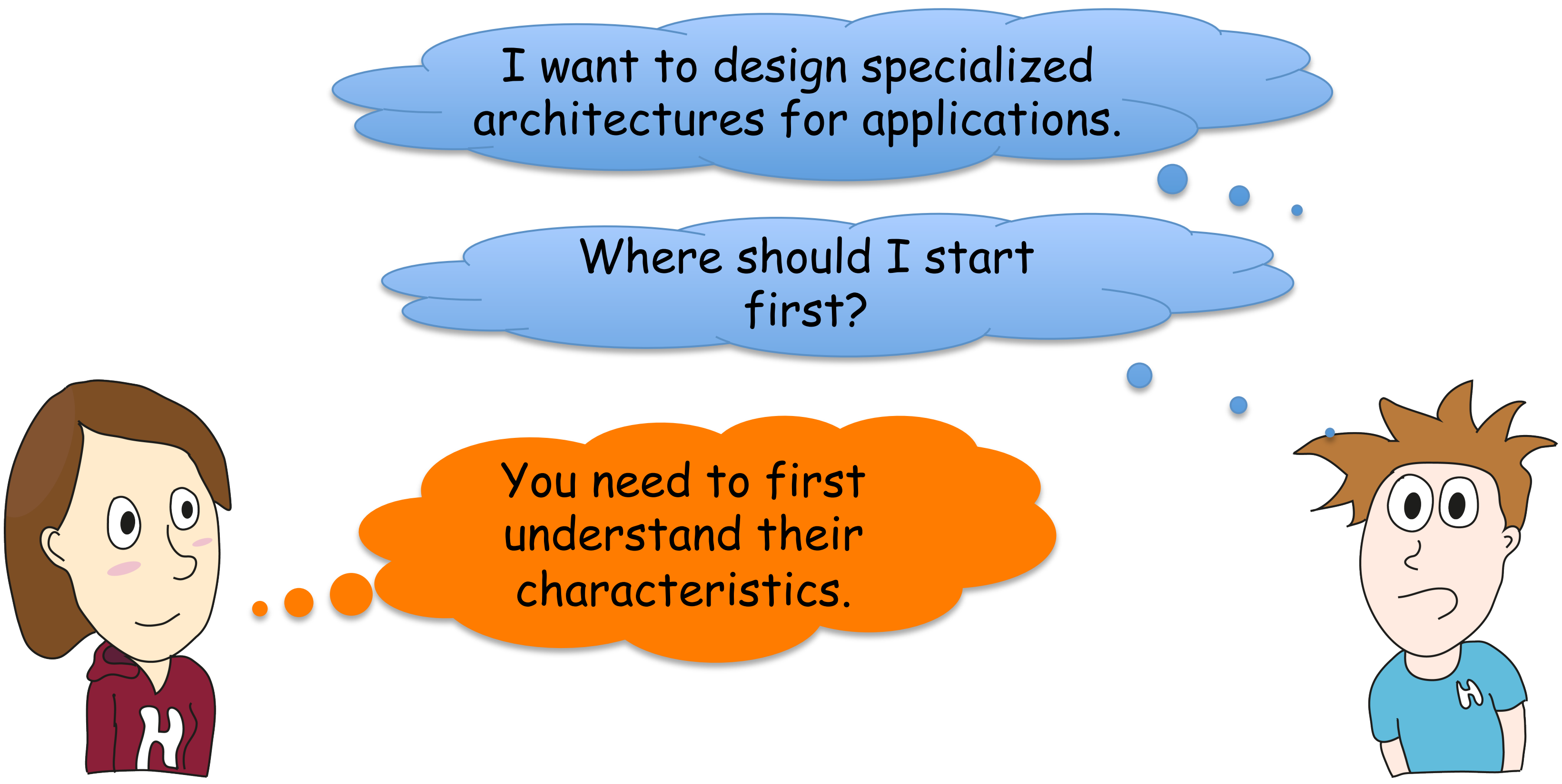
High Programmability ⟷ Low Programmability

Tied to a Specific ISA ⟷ No ISA

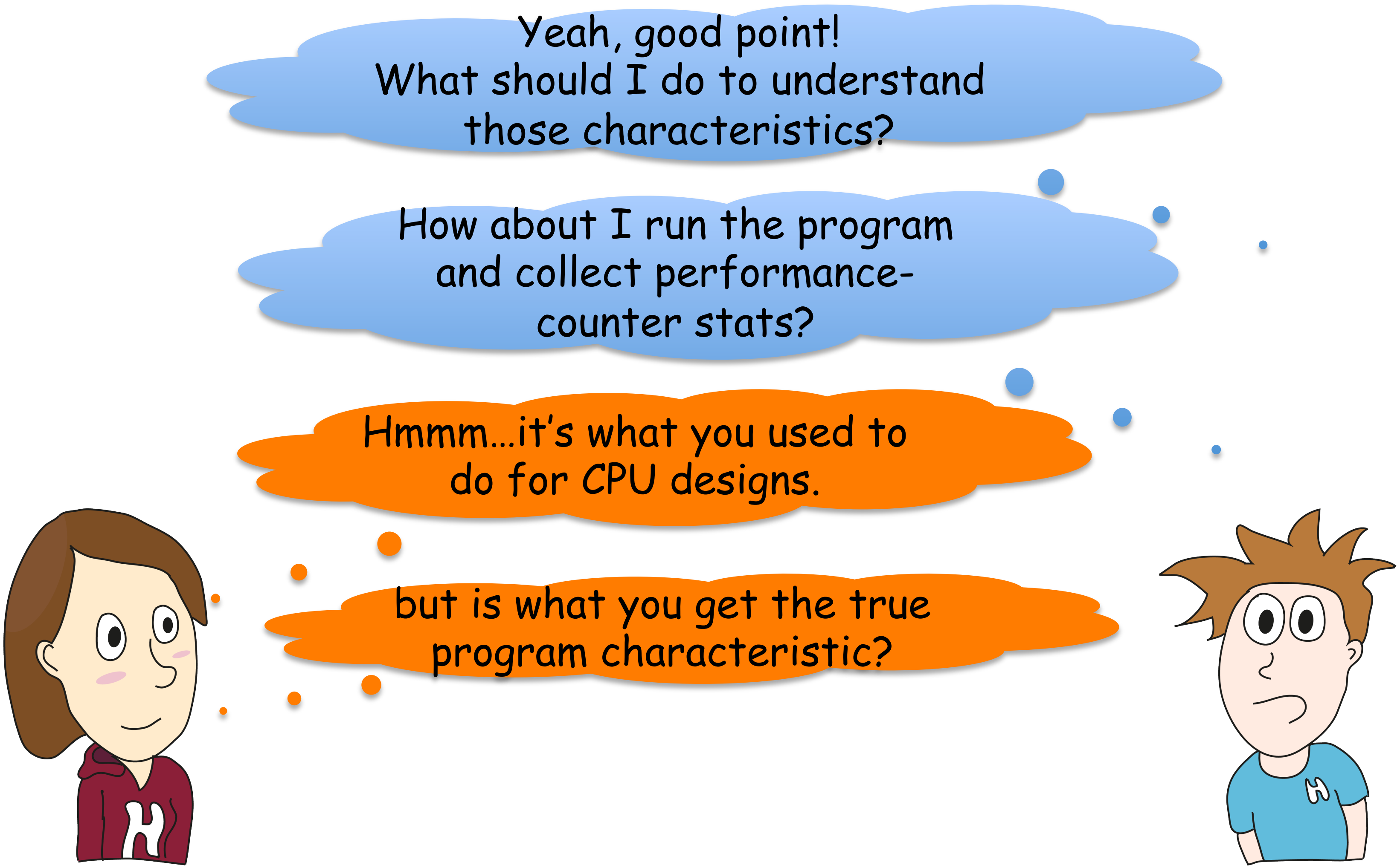
Specialization requires workload intrinsic characteristics.

Specialized architecture is tailored to applications.

- e.g. special data path, memory access patterns.



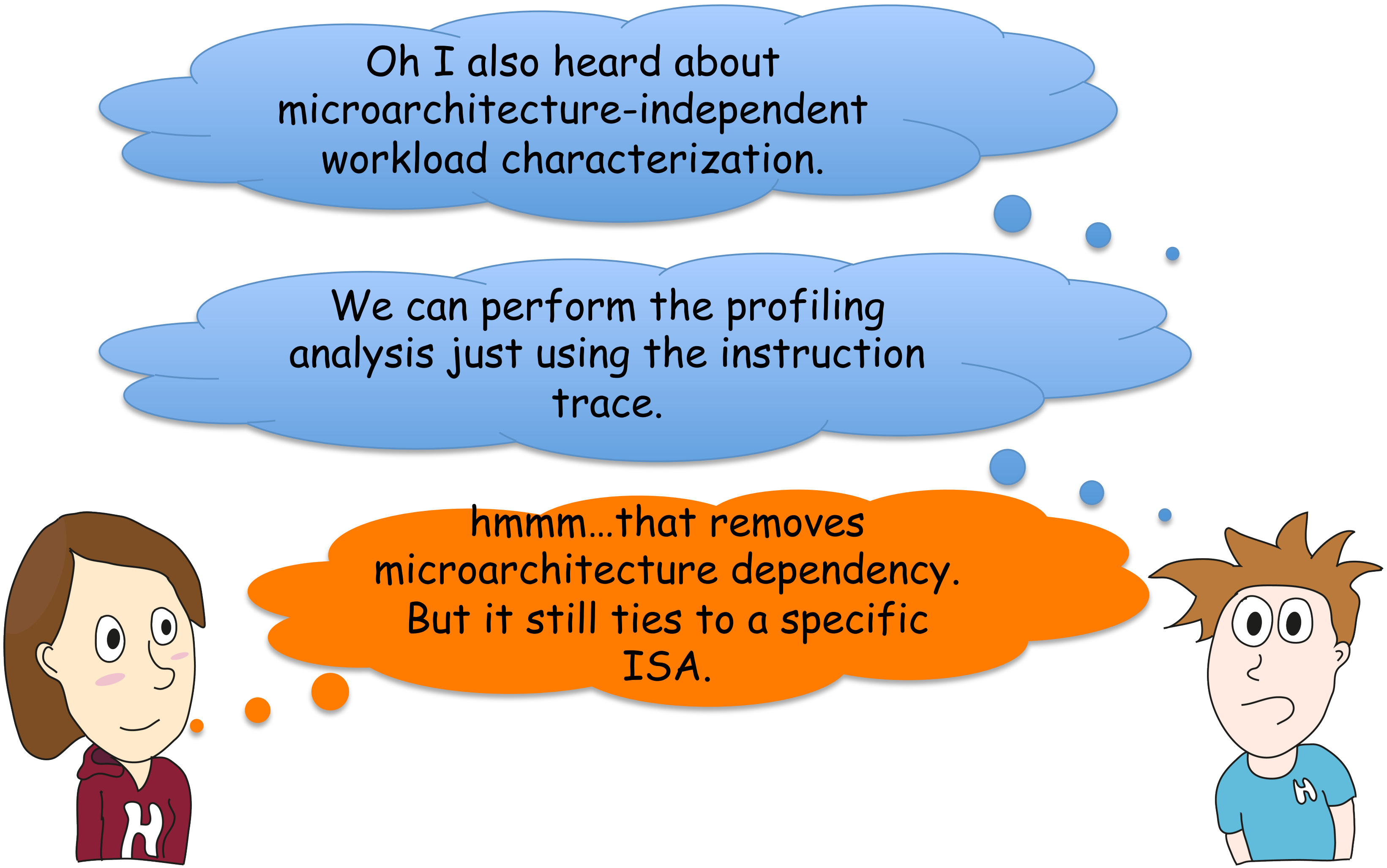
Specialization requires workload intrinsic characteristics.



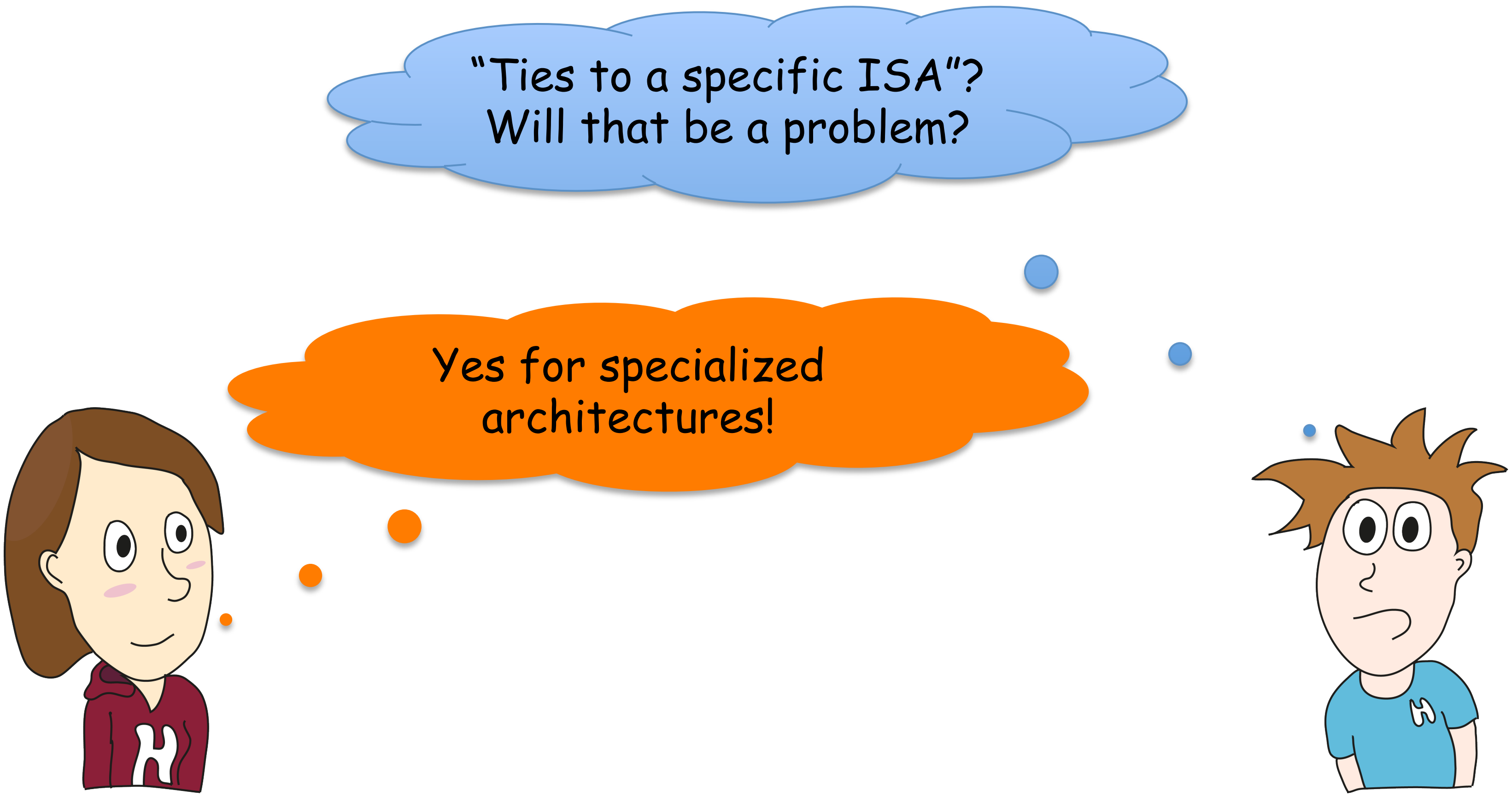
Performance-Counter Based Workload Characterization

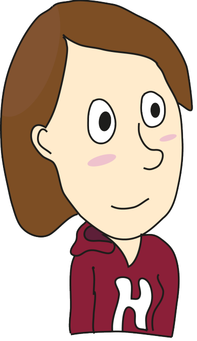
- Metrics
  - IPC
  - Cache miss rates
  - Branch mis-prediction rates
  - …
- Microarchitecture-dependent
  - What if there is a bigger cache/a better branch predictor?
  - Not program intrinsic characteristics

Specialization requires workload intrinsic characteristics.



Specialization requires workload intrinsic characteristics.





ISA impacts program behaviors.

Stack Overhead

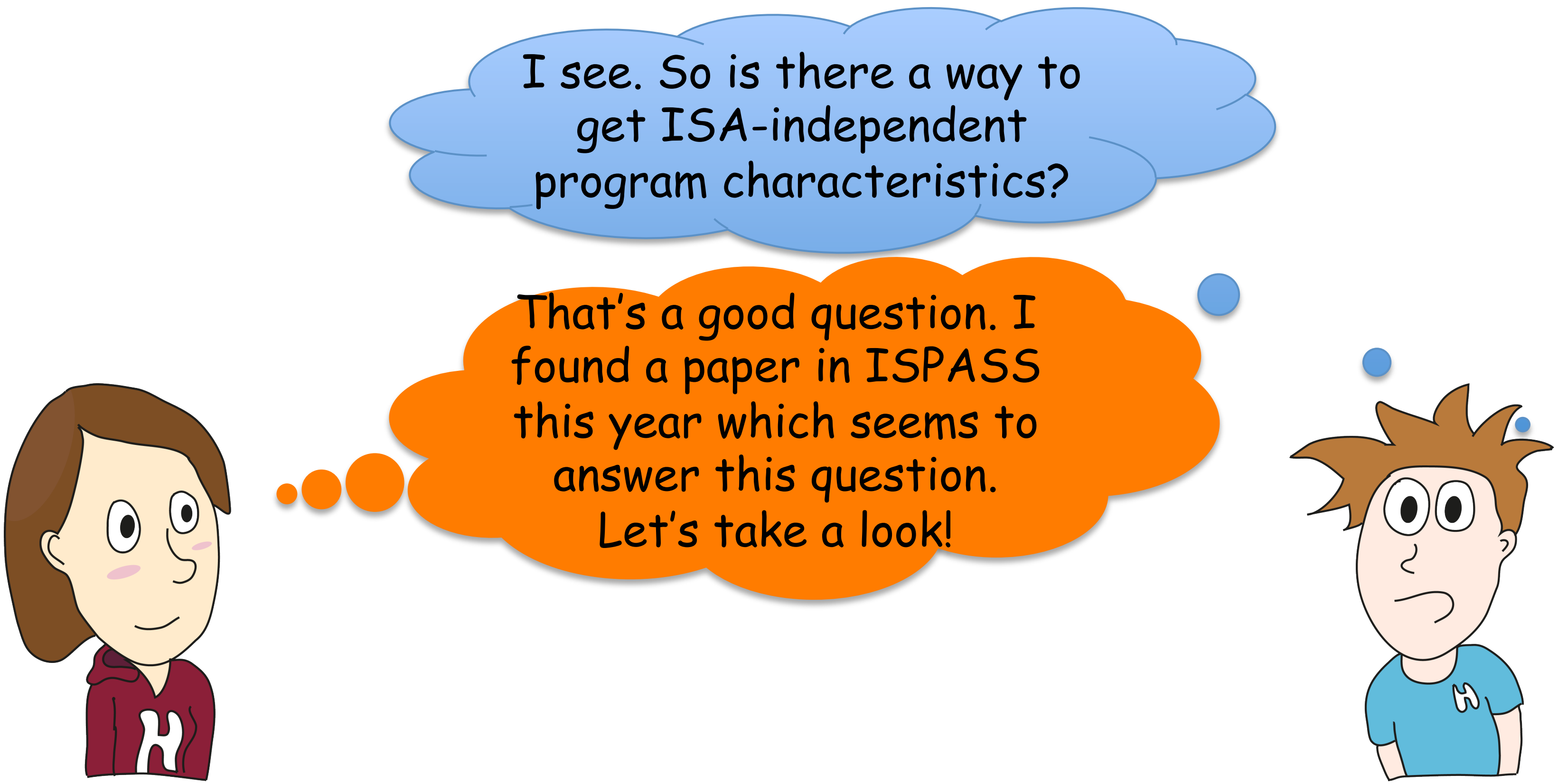
- Limited Registers
- Additional Load/Store

Complex Operations

- Memory Operands
- Vector Operations

Calling Conventions

Specialization requires workload intrinsic characteristics.



Paper Summary

Goal:

- An analysis tool to characterize workloads ISA-Independent characteristics for specialized architectures

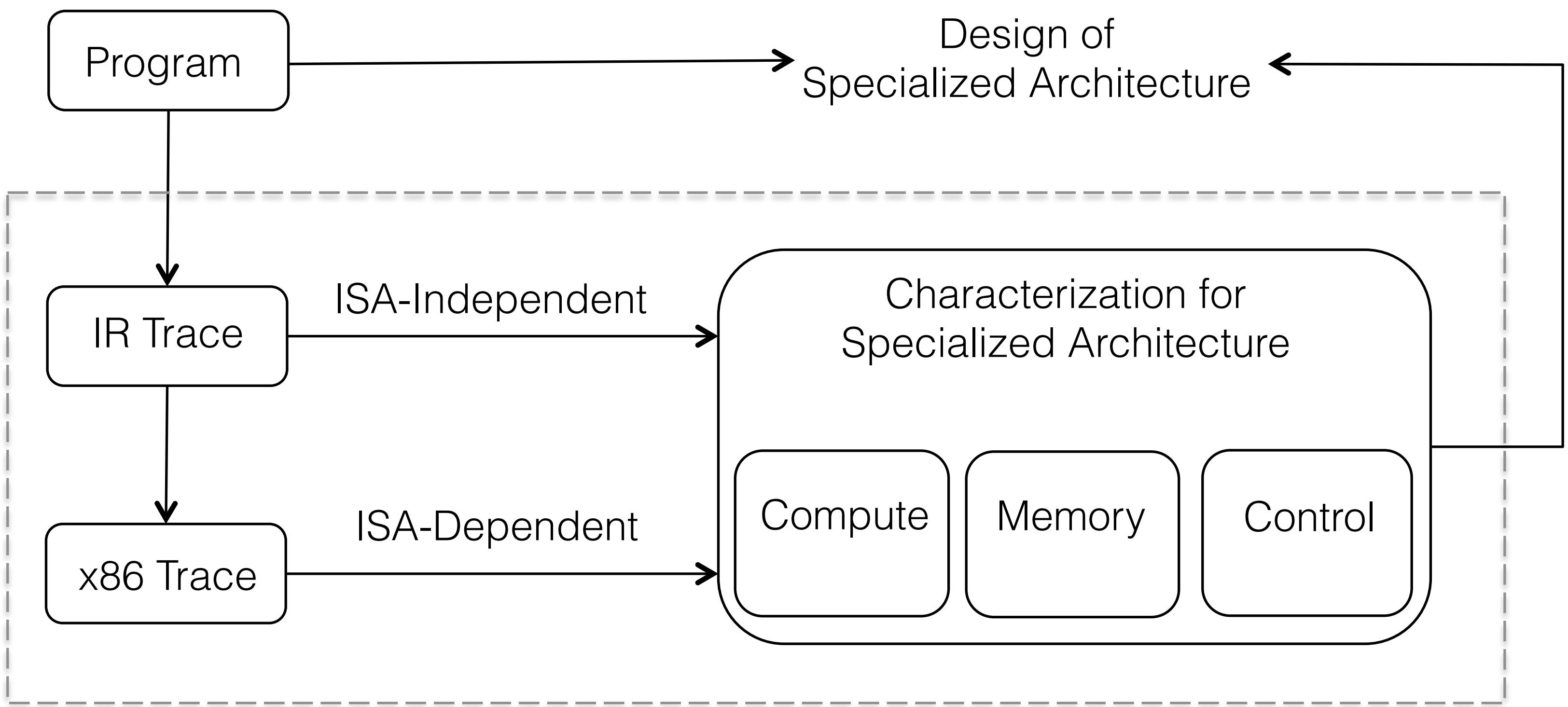
Methods:

- Leverage compiler's intermediate representation (IR)
- Categorize characteristics into compute, memory, and control

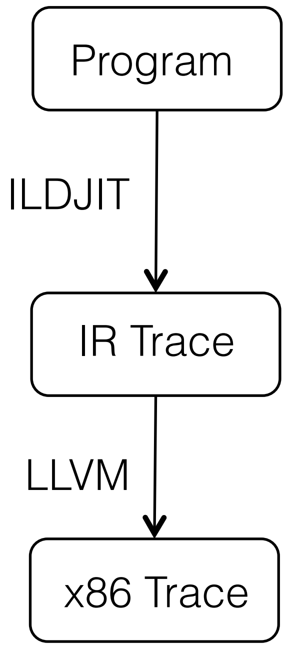
Takeaways:

- ISA-dependent characterization is misleading for specialization.
- ISA-independent characterization allows designers to quickly identify opportunities for specialization.

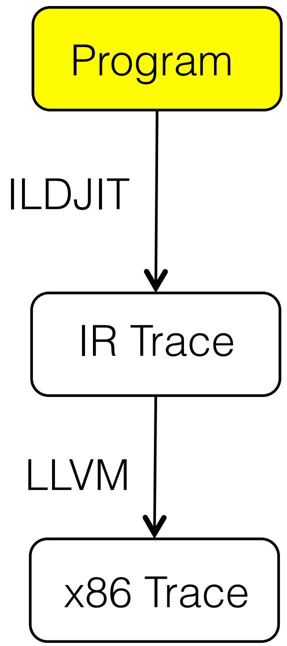
Tool Overview



Program Representations

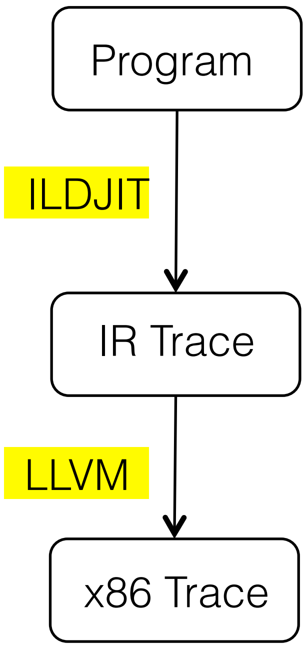


Program Representations



- SPEC CPU2000

Program Representations

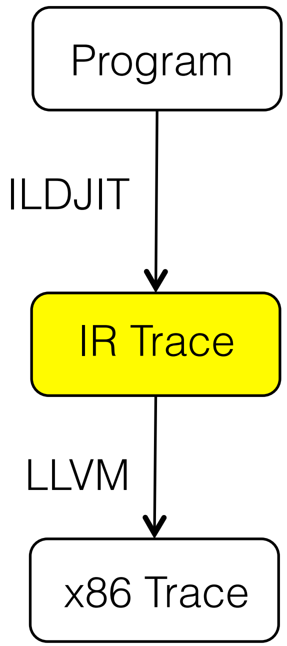


ILDJIT

- A modular compilation framework
- Performs machine-independent classical optimizations at the IR level
- Uses LLVM's back end to
  - Do machine-dependent optimizations
  - Generate machine code

Campanoni, et al., A Highly Flexible, Parallel Virtual Machine: Design and Experience of ILDJIT, Software Practice Experience, 2010

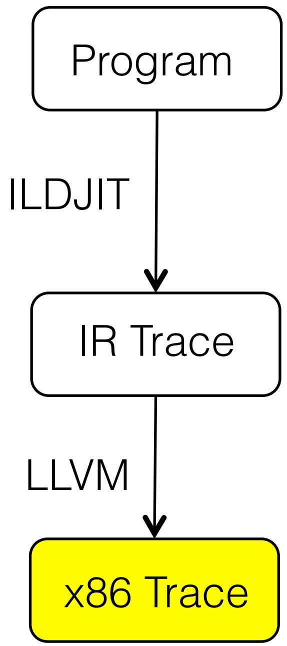
Program Representations



ILDJIT IR

- High-level IR
- Machine-, ISA-, and system-library-independent
- Features:
  - 80 instructions
  - Unlimited registers
  - Only loads/stores access memory
  - No vector operations
  - Parameters are passed by variables

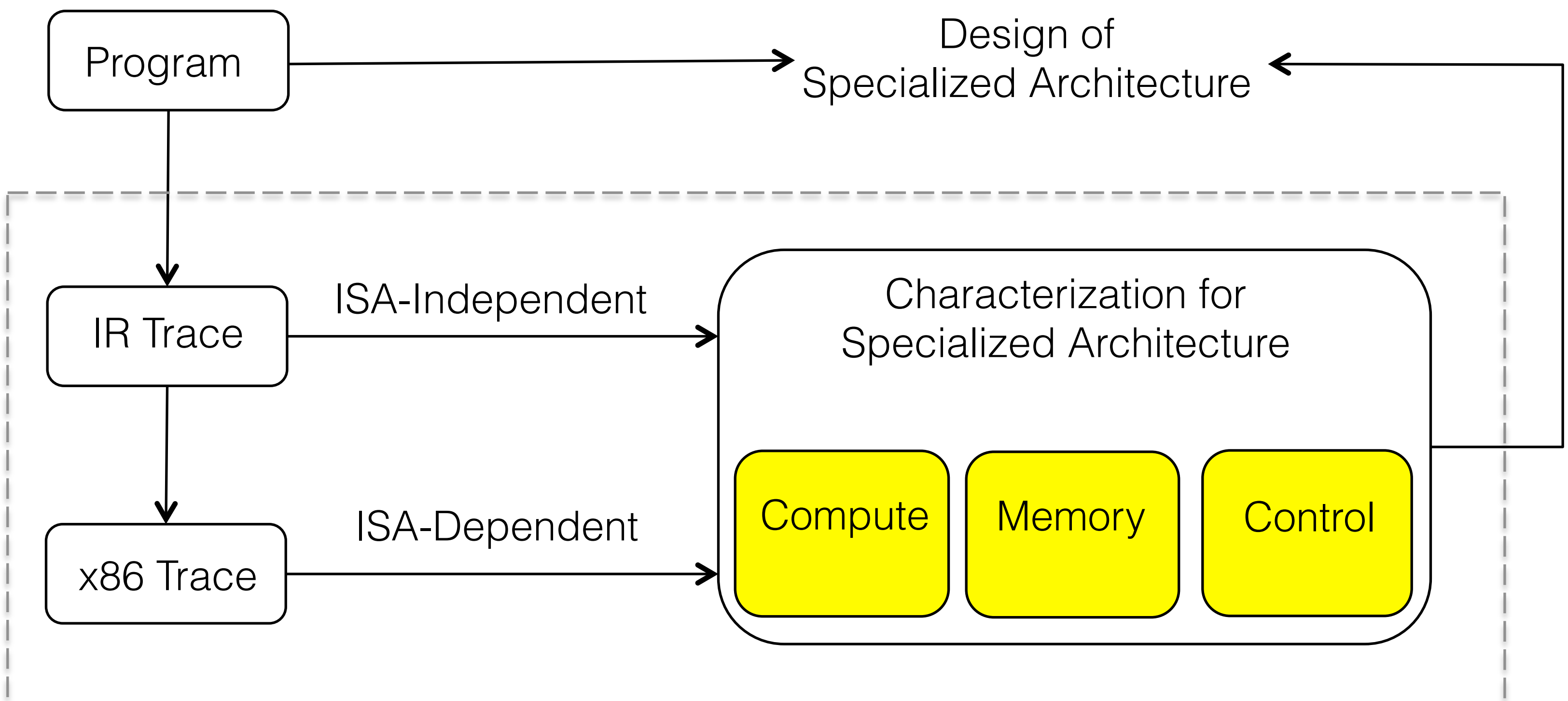
Program Representations



x86 Trace

- Used for ISA-dependent analysis
- Semantically equivalent to the IR code
- Collected with Pin instrumentation

Tool Overview



ISA-Independent Workload Characteristics

Compute

- Opcode Diversity
- **Static Instructions (I-MEM)**

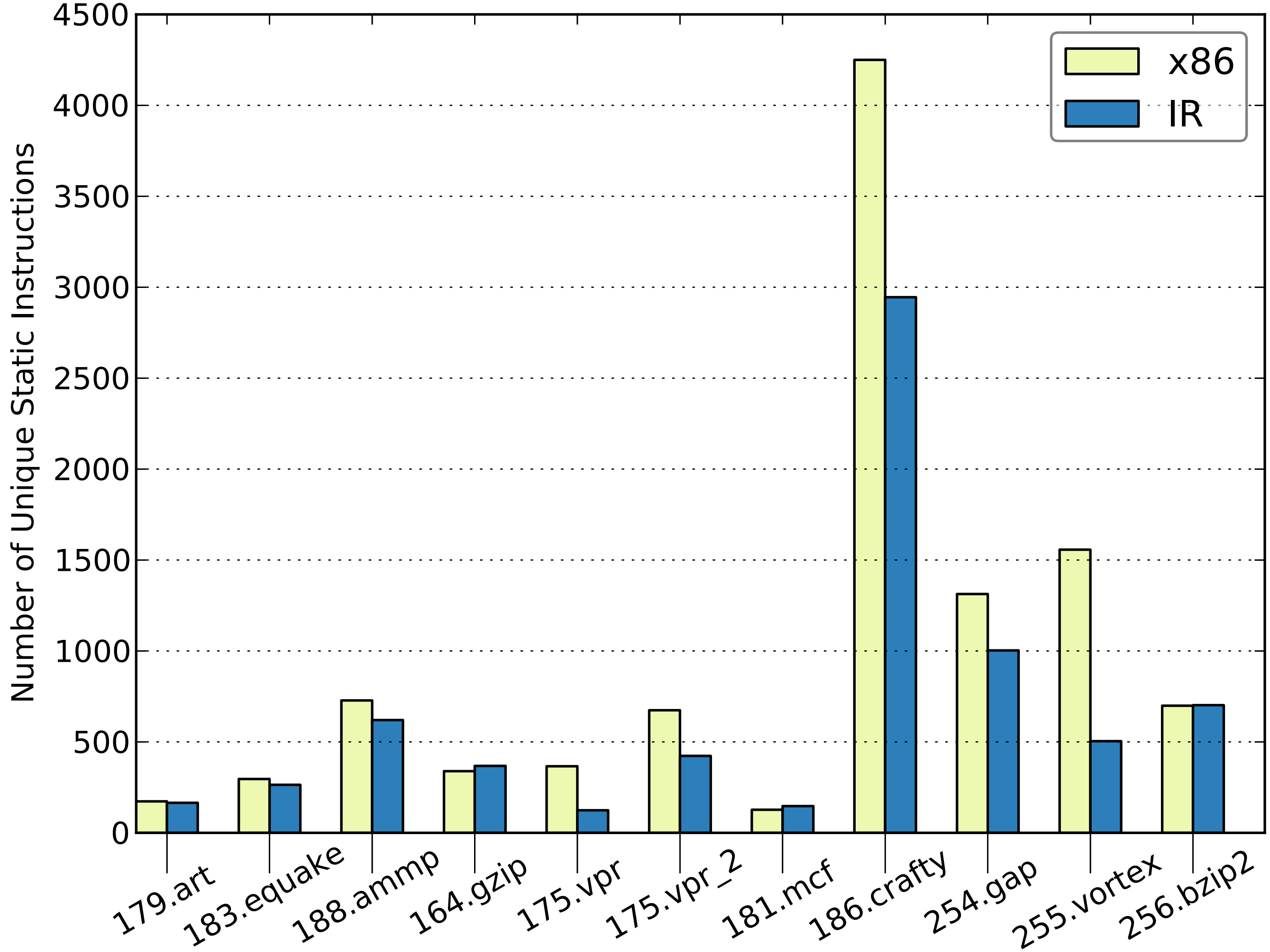
Memory

- Memory Footprint (D-MEM)
- Global Address Entropy
- Local Address Entropy

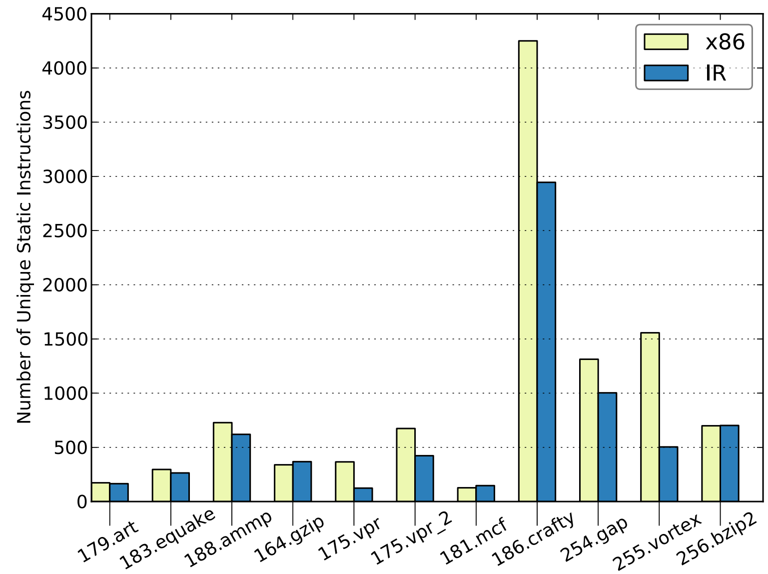
Control

- Branch Instruction Counts
- Branch Entropy

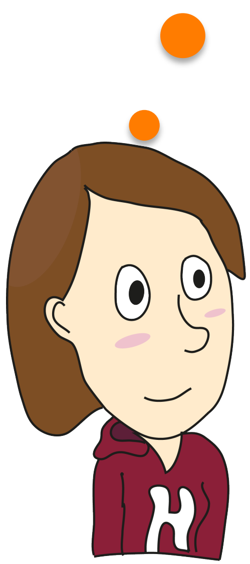
Compute::Static Instructions



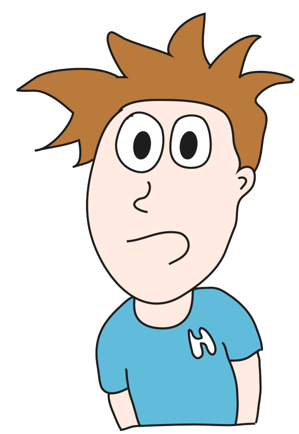
Compute::Static Instructions



So if you use x86 trace instead of IR trace…



I will think those stack operations are part of the "hot code".



ISA-Independent Workload Characteristics

Compute

- Opcode Diversity
- Static Instructions (I-MEM)

Memory

- Memory Footprint (D-MEM)
- **Global Address Entropy**
- **Local Address Entropy**

Control

- Branch Instruction Counts
- Branch Entropy

Memory::Entropy

Entropy: a measure of the randomness

$$Entropy = -\sum_{i=1}^{N} p(x_i) * \log_2 p(x_i)$$

Case 1:

X is always a constant.

$$p(X) = 1$$

$$\log_2 p(X) = 0$$

$$Entropy = 0$$

Case 2:

N possible outcomes of X occur equally.

$$p(X) = \frac{1}{N}$$

$$\log_2 p(X) = \log_2 N^{-1}$$

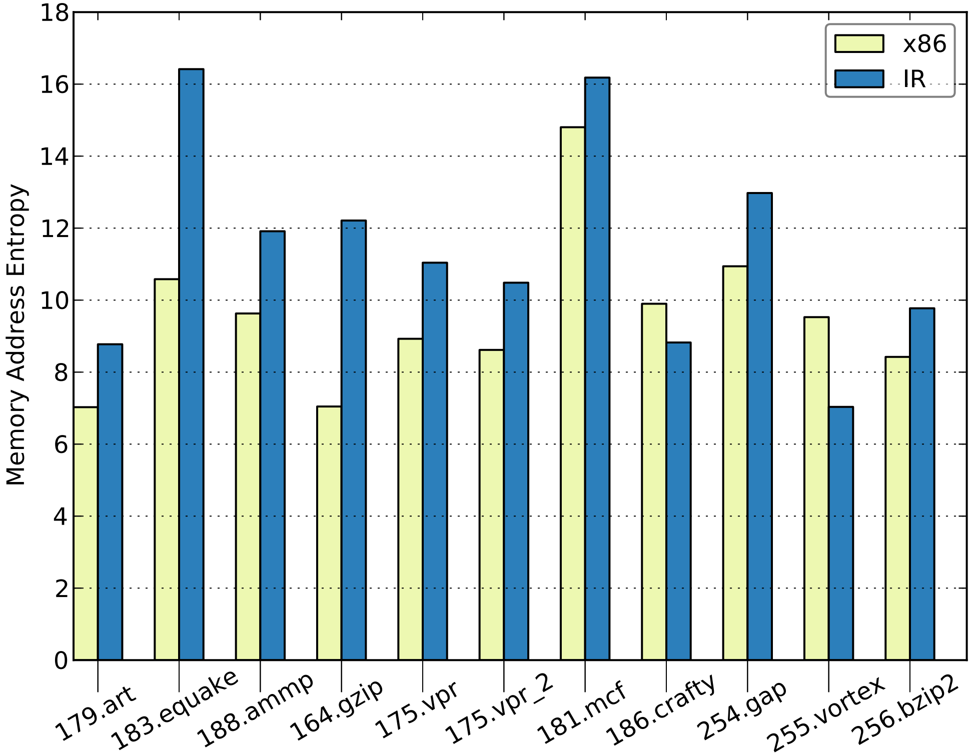
$$Entropy = -N * \frac{1}{N} * \log_2 N^{-1}$$

$$Entropy = \log_2 N$$

Memory::Global Address Entropy

Temporal Locality

Address Stream A (less temporal locality)	Address Stream B (more temporal locality)
0 0 0 0	0 0 1 1
0 0 0 1	0 0 1 1
0 0 1 0	0 0 1 1
0 0 1 1	0 0 1 1
Entropy = 2	Entropy = 0

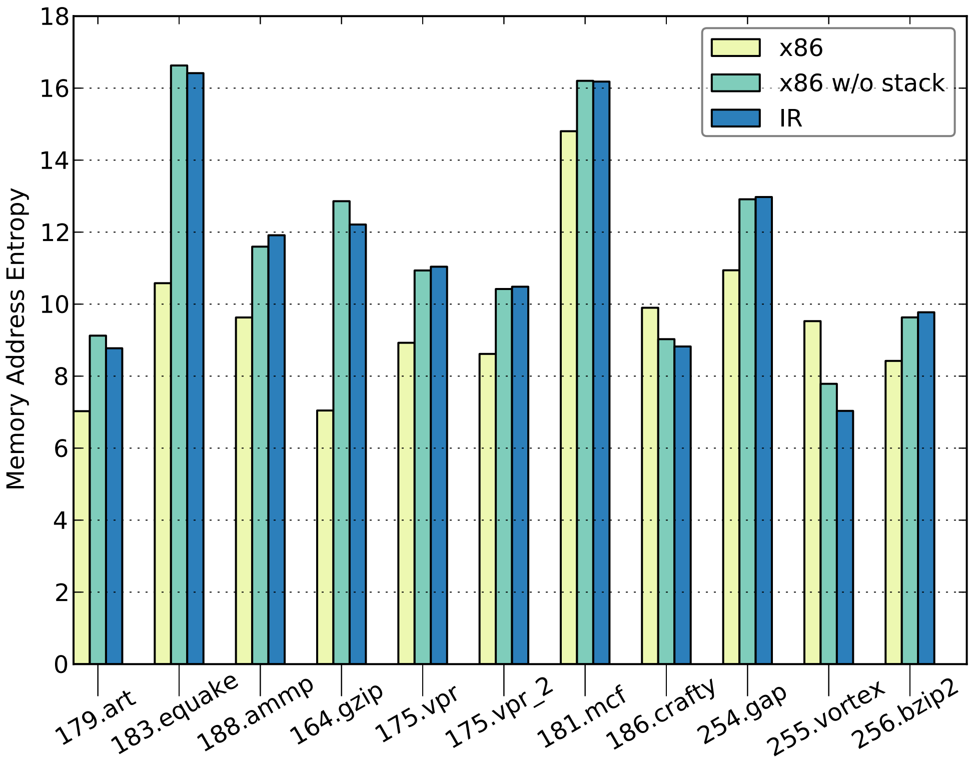


Yen, Draper, and Hill. Notary: Hardware Techniques to Enhance Signatures. MICRO 08

Memory::Global Address Entropy

Temporal Locality

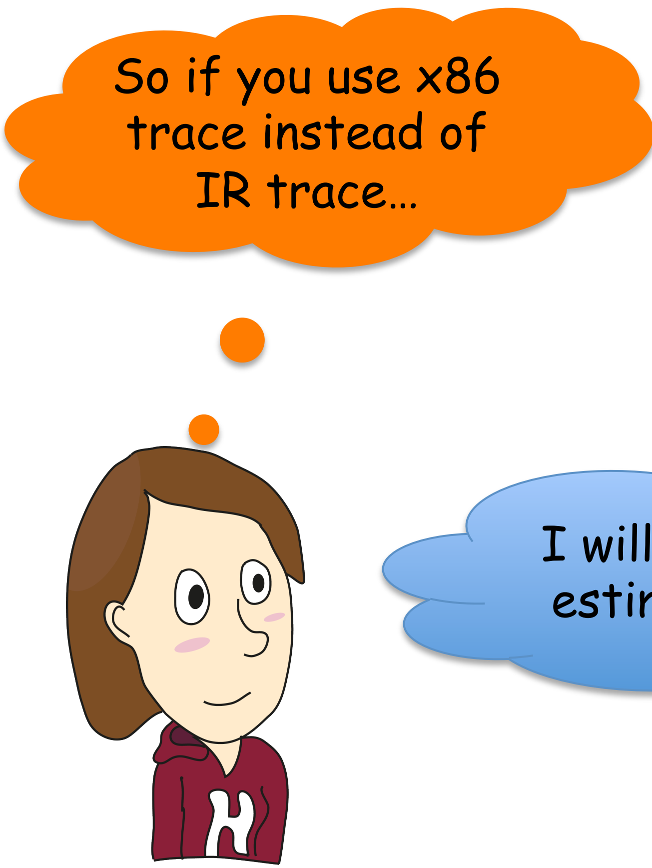
Address Stream A (less temporal locality)	Address Stream B (more temporal locality)
0 0 0 0	0 0 1 1
0 0 0 1	0 0 1 1
0 0 1 0	0 0 1 1
0 0 1 1	0 0 1 1
Entropy = **2**	Entropy = 0

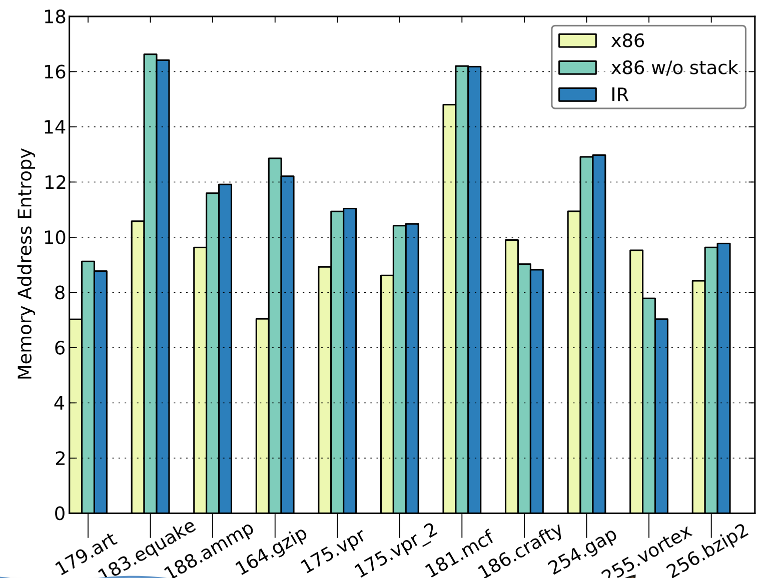


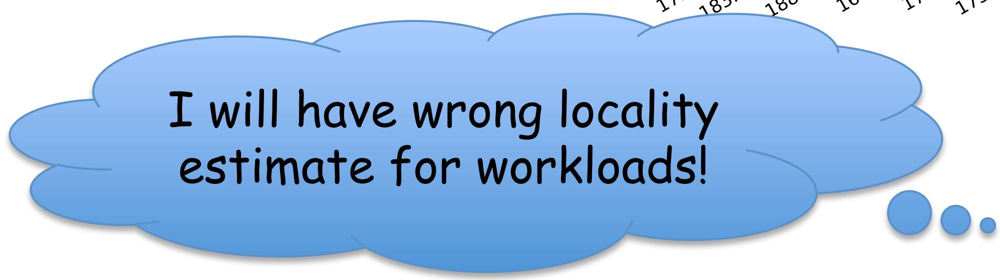
Yen, Draper, and Hill. Notary: Hardware Techniques to Enhance Signatures. MICRO 08

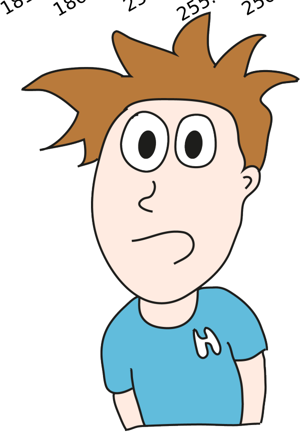
Memory::Global Address Entropy

Temporal Locality





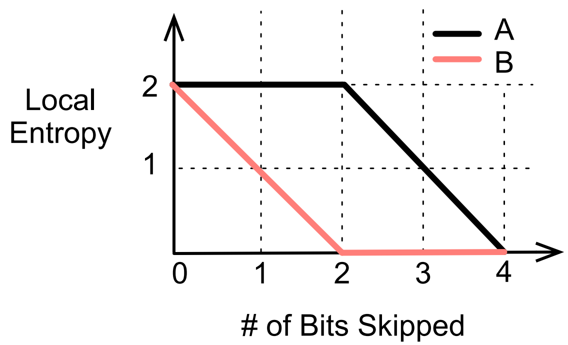


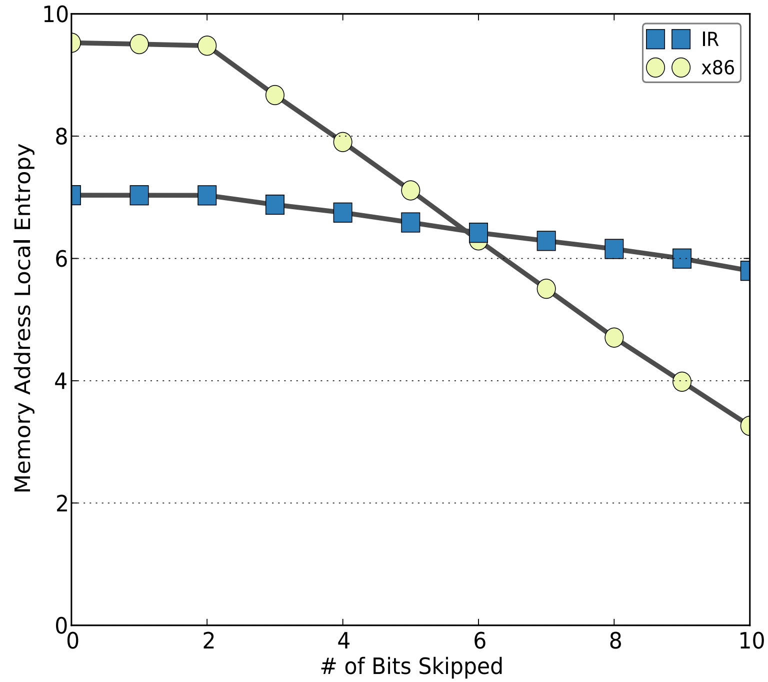


Memory::Local Address Entropy

Spatial Locality

Address Stream A (less spatial locality)	Address Stream B (more spatial locality)
0 0 0 0	0 0 0 0
0 1 0 0	0 0 0 1
1 0 0 0	0 0 1 0
1 1 0 0	0 0 1 1

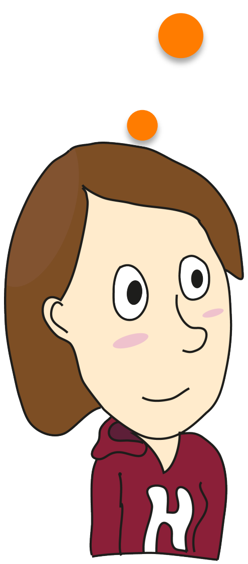


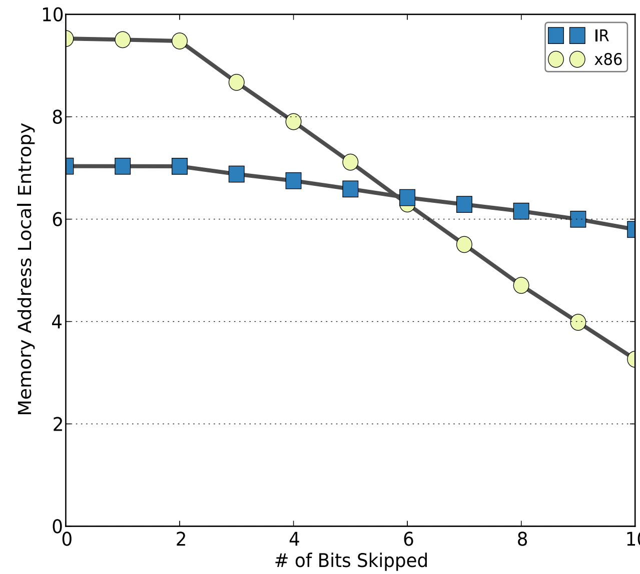


Memory::Local Address Entropy

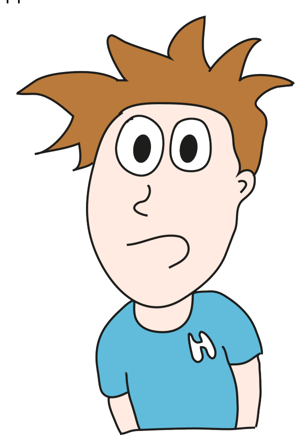
Spatial Locality

So if you use x86 trace instead of IR trace...





I will think program has more spatial locality than it really has.



ISA-Independent Workload Characteristics

Compute

- Opcode Diversity
- Static Instructions (I-MEM)

Memory

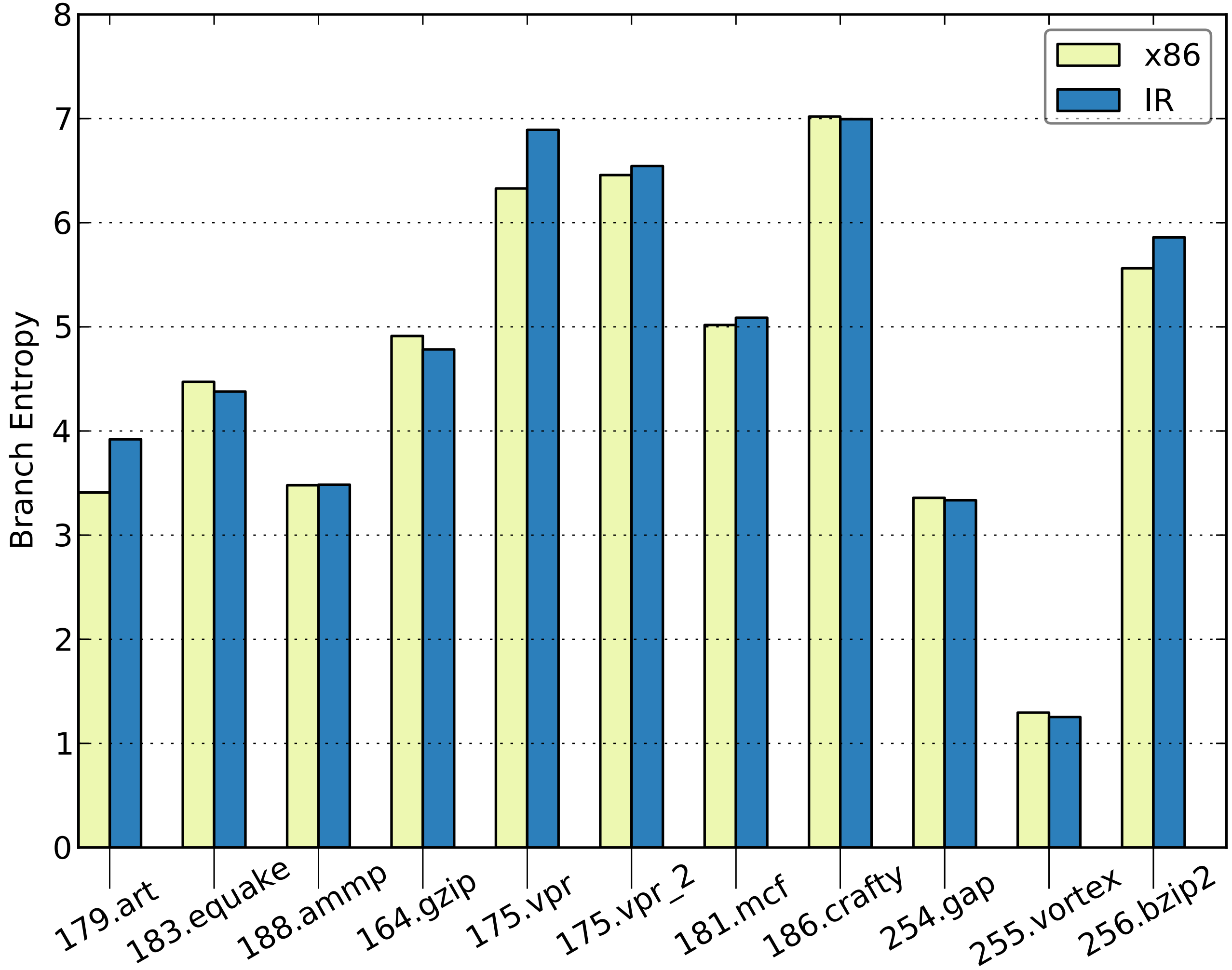
- Memory Footprint (D-MEM)
- Global Address Entropy
- Local Address Entropy

Control

- Branch Instruction Counts
- **Branch Entropy**

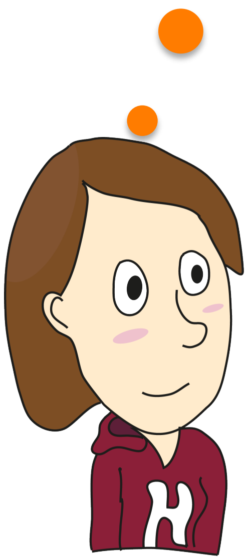
Yokota, et all, Introducing Entropies for Representing Program Behavior and Branch Predictor Performance, 07

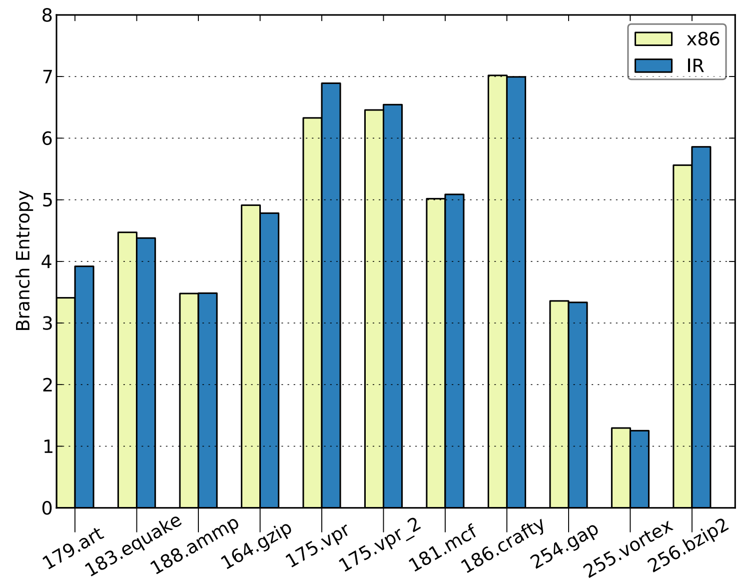
Control::Branch Entropy



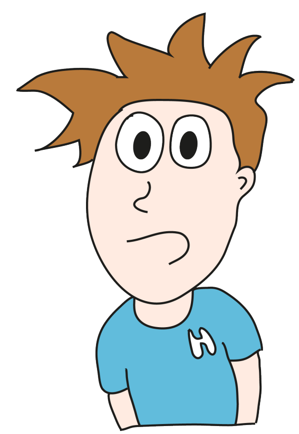
Control::Branch Entropy

So if you use x86 trace instead of IR trace...

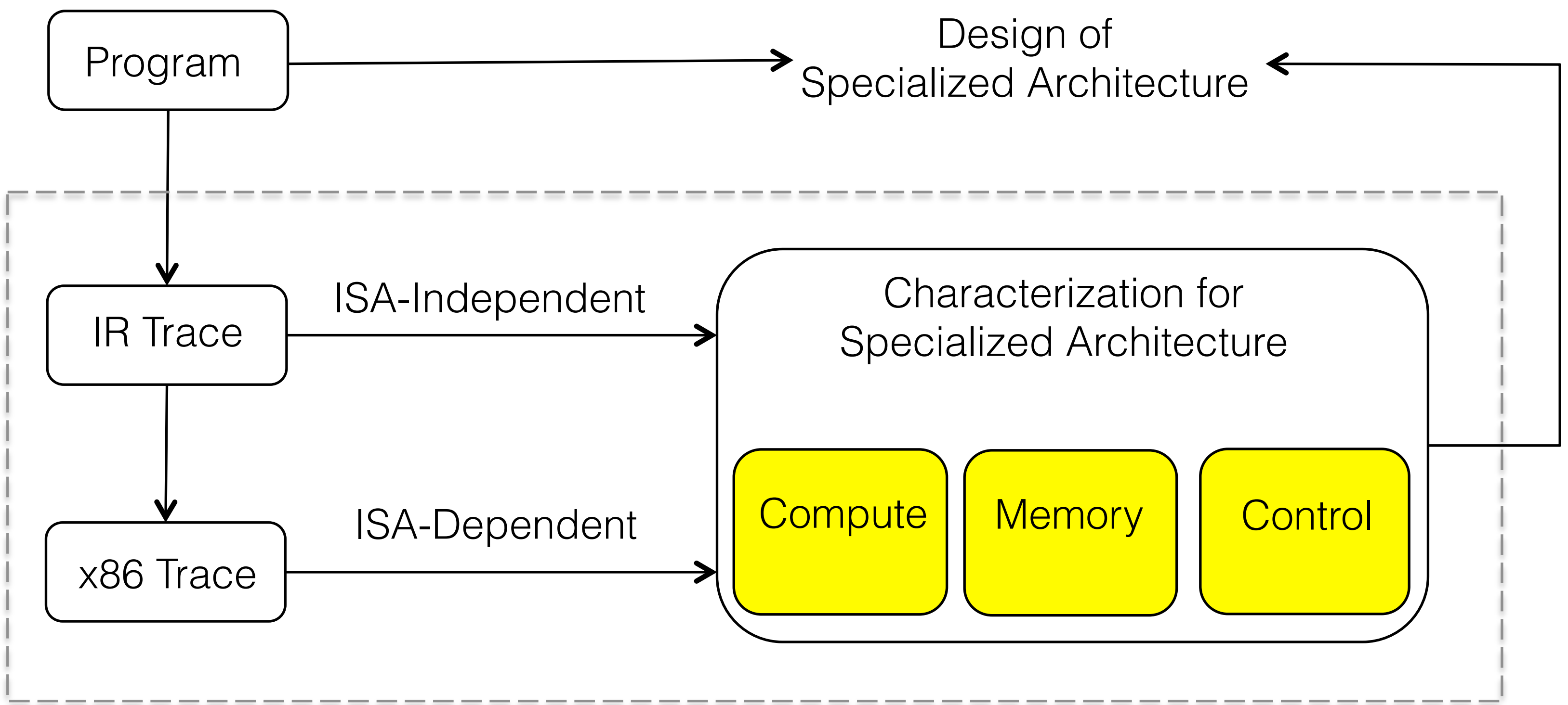




I won't get much wrong for control.



Tool Overview



ISA-Independent Workload Characteristics

Compute

- Opcode Diversity
- Static Instructions (I-MEM)

Memory

- Memory Footprint (D-MEM)
- Global Address Entropy
- Local Address Entropy

Control

- Branch Instruction Counts
- Branch Entropy

ISA-Independent Workload Characteristics

Compute

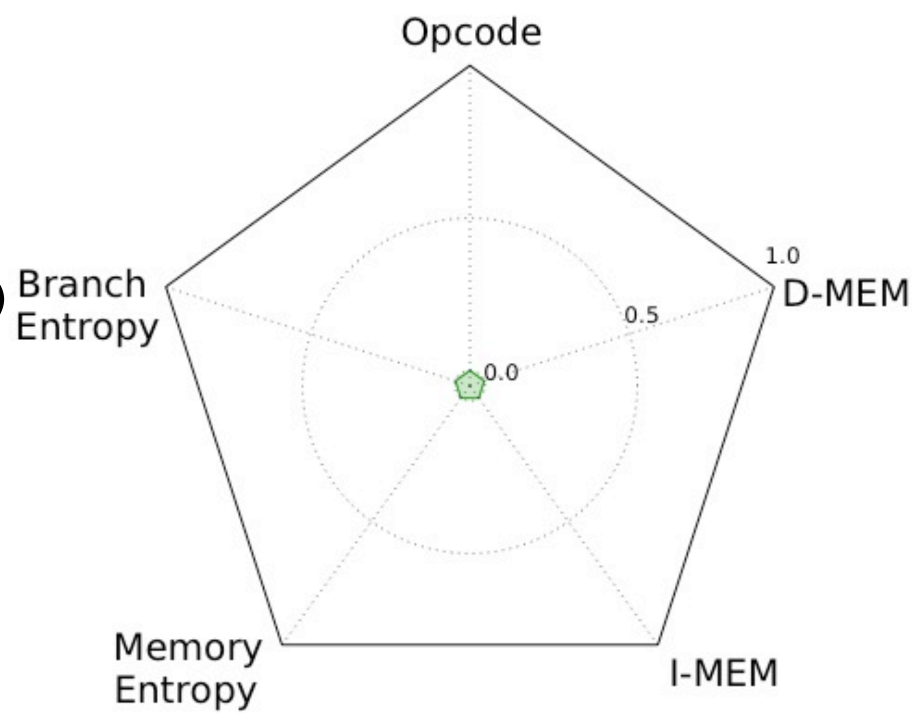
- **Opcode Diversity**
- **Static Instructions (I-MEM)**

Memory

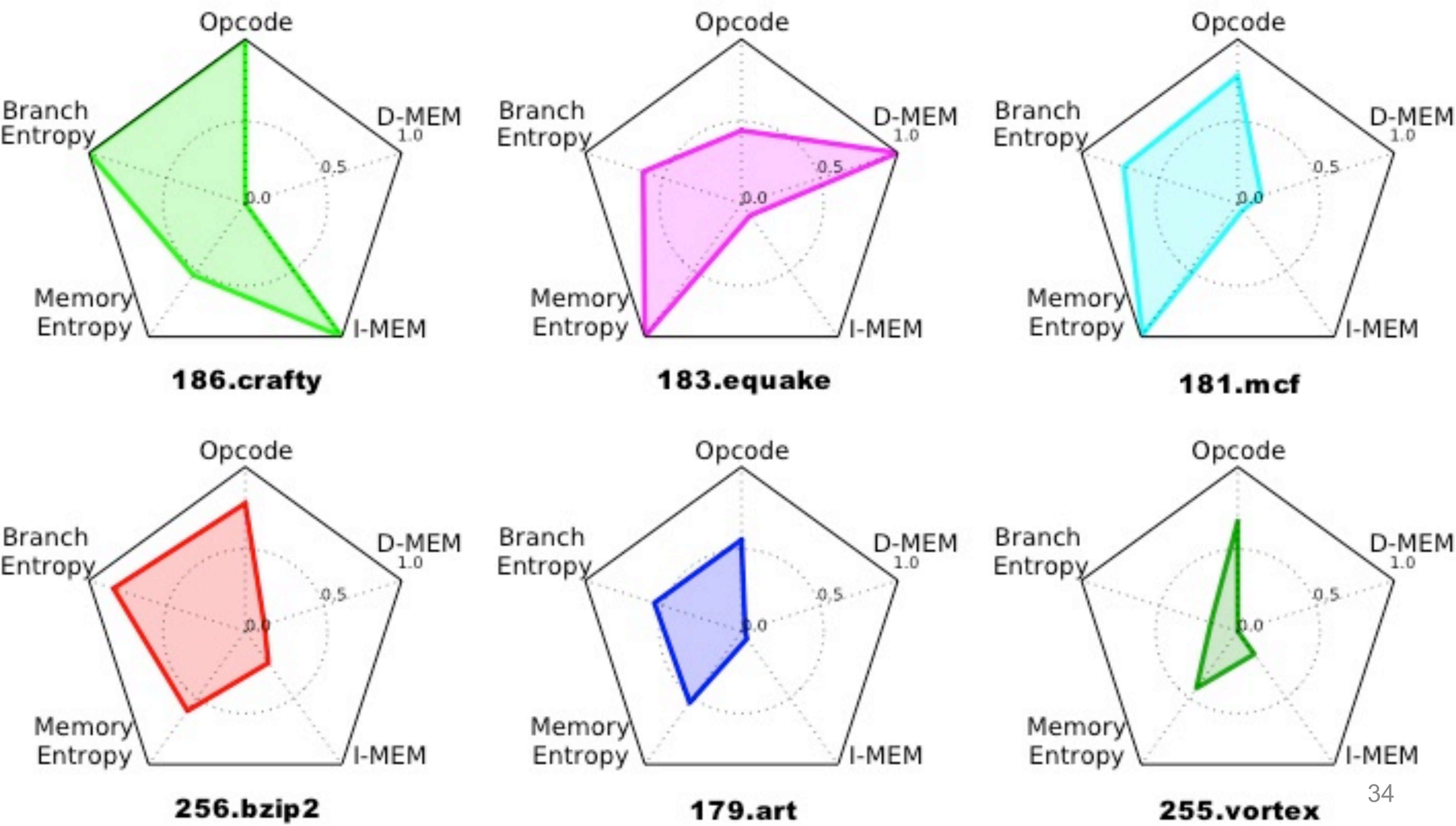
- **Memory Footprint (D-MEM)**
- **Global Address Entropy**
- Local Address Entropy

Control

- Branch Instruction Counts
- **Branch Entropy**



Workload Characterization



Conclusions

- We demonstrate that ISA-dependent analysis can be misleading for specialized architectures.

- We present an analysis tool to characterize ISA-independent characteristics for specialization.

- We show that our tool provides opportunities for designers to compare workloads' characteristics.