# Software Helping Hardware Innovation

Michael O’Boyle
University of Edinburgh
EPSRC Senior Research Fellow

What are the key challenges in your area?

Why should ARM care?

What are the opportunities for ARM?

How could ARM help?

How does your work compare against rest of world?

What are the key challenges in your area?

- exploiting potential of accelerators
- enabling all code to utilise any accelerator

Why should ARM care?

What are the opportunities for ARM?

How could ARM help?

How does your work compare against rest of world?

What are the key challenges in your area?

- exploiting potential of accelerators
- enabling all code to utilise any accelerator

Why should ARM care?

- enabling  general acceleration opens up hardware innovation
- sell IP to customers knowing their customers’  code  will work

What are the opportunities for ARM?

How could ARM help?

How does your work compare against rest of world?

What are the key challenges in your area?
- exploiting potential of accelerators
- enabling all code to utilise any accelerator

Why should ARM care?
- enabling general acceleration opens up hardware innovation
- sell IP to customers knowing their customers' code will work

What are the opportunities for ARM?
- innovate, design and develop accelerator IP others will miss

How could ARM help?

How does your work compare against rest of world?

What are the key challenges in your area?
- exploiting potential of accelerators
- enabling all code to utilise any accelerator

Why should ARM care?
- enabling general acceleration opens up hardware innovation
- sell IP to customers knowing their customers' code will work

What are the opportunities for ARM?
- innovate, design and develop accelerator IP others will miss

How could ARM help?
- engage with research; ARM researchers + funded PhDs
- explore impact on accelerator roadmap

How does your work compare against rest of world?

What are the key challenges in your area?
- exploiting potential of accelerators
- enabling all code to utilise any accelerator

Why should ARM care?
- enabling general acceleration opens up hardware innovation
- sell IP to customers knowing their customers' code will work

What are the opportunities for ARM?
- innovate, design and develop accelerator IP others will miss

How could ARM help?
- engage with research; ARM researchers + funded PhDs
- explore impact on accelerator roadmap

How does your work compare against rest of world?
- Top conferences: PLDI, ASPLOS, HPCA, Micro, CGO, NeurIPS
- 3 Best paper awards: ACM GPCE20, HPCA21, ASPLOS21
- "*highest ranked software"* DARPA ERI SDH program
- World-leading compiler group at Edinburgh

Matching Hardware to Software - Hardware Defined Software (HDS)

Other

- Neural Architecture Search as Program Transformation Exploration

- Software Defined  Hardware (SDH)

Beyond Simple Acceleration

**Matching Hardware to Software - hardware defined software (HDS)**

Other

- Neural Architecture Search as Program Transformation Exploration

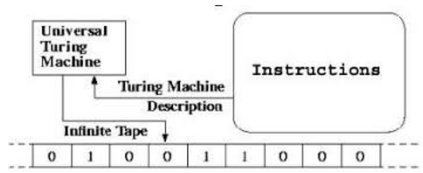
- Software Defined  Hardware (SDH)

Beyond Simple Acceleration

# Hardware/software contract breaking down

Technology trends means
- Hardware specialised or heterogenous

Great
- up to 100,000x performance/energy gains




No free lunch
- Software cannot fit on new hardware

Heterogeneous crisis
- hardware stalls as software cannot fit

# Hardware/software contract breaking down

Technology trends means
- Hardware specialised or heterogenous

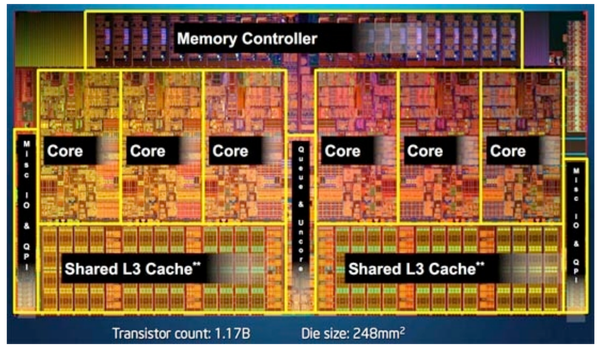
Great
- up to 100,000x performance/energy gains

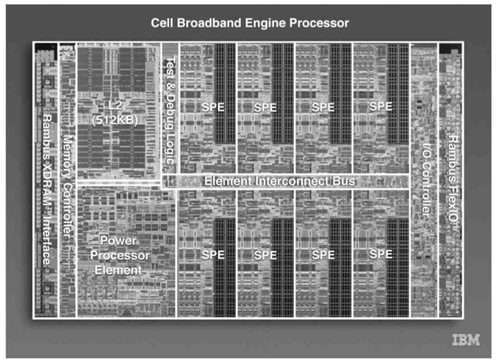

# Rethink the contract

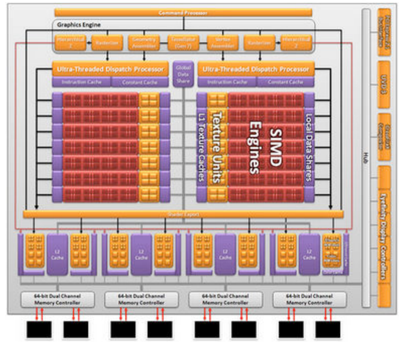
Heterogeneous crisis
- hardware stalls as software cannot fit

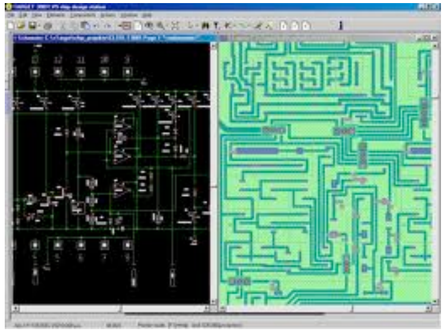
# Taming the  Hardware Zoo?

New Application/Legacy Code



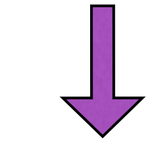



# Language Approach

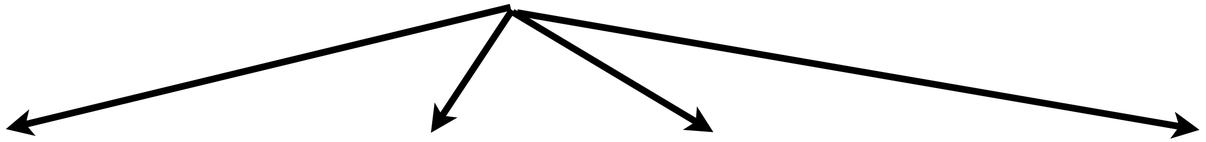
New Application/Legacy Code

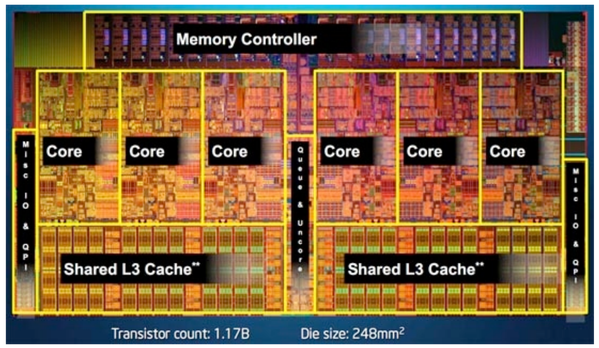


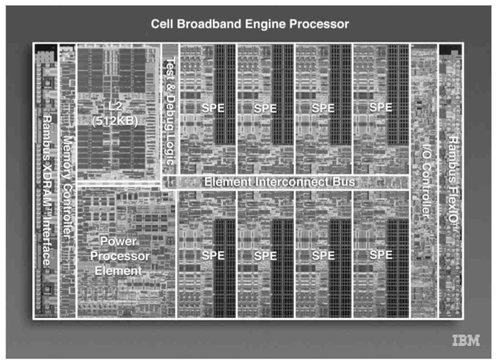
Parallel Language

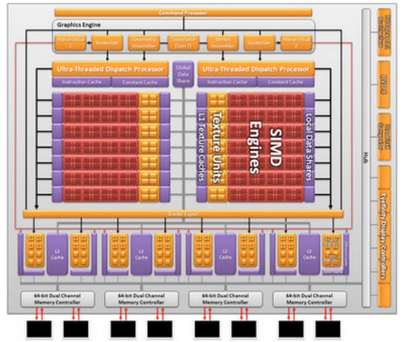
User rewrites

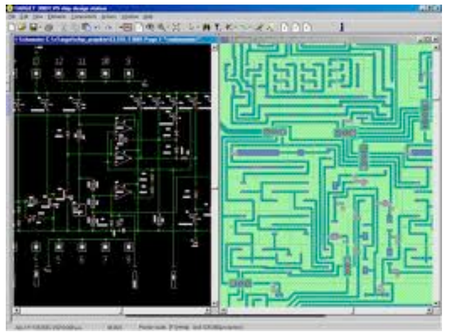
Write new compiler





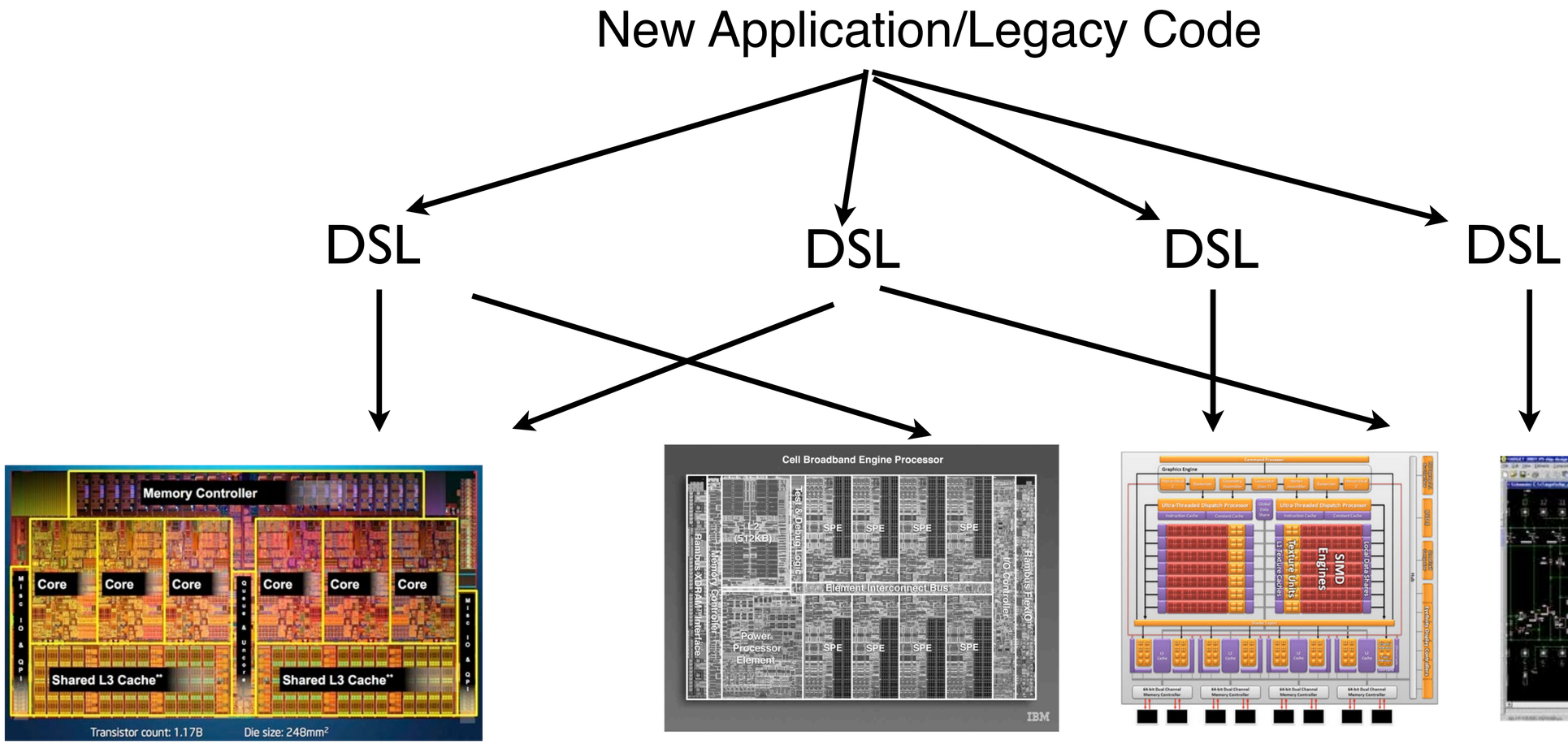



A universal parallel language + opt compiler per ISA/platform
+ smart runtime/glue?

# DSL approach




Many specialised languages. Rewrite and hope it works on your (next) machine?

Good performance is hard to get even with
well defined parallel language CUDA/OpenCL

# GPU-Accelerated Libraries

**GPU-Accelerated libraries provide highly-optimized algorithms and functions you can incorporate into your applications, with minimal changes to your existing code. Many support drop-in compatibility to replace industry standard CPU-only libraries such as MKL, IPP, FFTW and widely-used libraries. Some also feature automatic multi-GPU performance scaling.**

### AmgX

A simple path to accelerated core solvers, providing up to 10x acceleration in the computationally intense linear solver portion of simulations, and is very well suited for implicit unstructured methods.


### cuDNN

NVIDIA cuDNN is a GPU-accelerated library of primitives for deep neural networks. It is designed to be integrated into higher-level machine learning frameworks.

### cuFFT

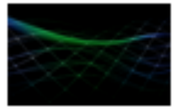
NVIDIA CUDA Fast Fourier Transform Library (cuFFT) provides a simple interface for computing FFTs up to 10x faster, without having to develop your own custom GPU FFT implementation.

### IndeX Framework

NVIDIA IndeX Framework is a real-time scalable visualization plug-in for ParaView.

### cuRAND

The CUDA Random Number Generation library performs high quality GPU-accelerated random number generation (RNG) over 8x faster than typical CPU only code.



### CUDA Math Library

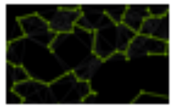
An industry proven, highly accurate collection of standard mathematical functions, providing high performance on NVIDIA GPUs.


### Thrust

A powerful, open source library of parallel algorithms and data structures. Perform GPU-accelerated sort, scan, transform, and reductions with just a few lines of code.

### NVBIO

A GPU-accelerated C++ framework for High-Throughput Sequence Analysis for both short and long read alignment.



### nvGRAPH

nvGRAPH Analytics Library is a GPU-accelerated graph analytics library.

### GIE

NVIDIA GPU Inference Engine is a high performance neural network inference library for deep learning applications

### NPP

NVIDIA Performance Primitives is a GPU accelerated library with a very large collection of 1000's of image processing primitives and signal processing primitives.


### FFmpeg

FFmpeg is a popular open-sour multi-media framework with a library of plugins that can be applied to various parts of the audio and video processing pipelines.

### NVIDIA VIDEO CODEC SDK

Accelerate video compression with the NVIDIA Video Codec SDK. This SDK includes documentation and code samples that illustrate how to use NVIDIA's NVENC and NVDEC hardware in GPUs to accelerate encode, decode, and transcode of H.264 and HEVC video formats.


### HiPLAR

HiPLAR (High Performance Linear Algebra in R) delivers high performance linear algebra (LA) routines for the R platform for statistical computing using the latest software libraries for heterogeneous architectures.


### OpenCV

OpenCV is the leading open source library for computer vision, image processing and machine learning, and now features GPU acceleration for real-time operation.


### Geometry Performance Primitives(GPP)

GPP is a computational geometry engine that is optimized for GPU acceleration, and can be used in advanced Graphical Information Systems (GIS), Electronic Design Automation (EDA), computer vision, and motion planning solutions.


### CHOLMOD

GPU-accelerated CHOLMOD is part of the SuiteSparse linear algebra package by Prof. Tim Davis. SuiteSparse is used extensively throughout industry and academia.


### CULA Tools

GPU-accelerated linear algebra library by EM Photonics, that utilizes CUDA to dramatically improve the computation speed of sophisticated mathematics.


### MAGMA

A collection of next gen linear algebra routines. Designed for heterogeneous GPU-based architectures. Supports current LAPACK and BLAS standards.


### IMSL Fortran Numerical Library

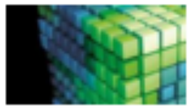
Developed by RogueWave, a comprehensive set of mathematical and statistical functions that offloads work to GPUs.


### aralution

library for sparse iterative ethods with special focus on ulti-core and accelerator chnology such as GPUs.


### Triton Ocean SDK

Triton provides real-time visual simulation of the ocean and bodies of water for games, simulation, and training applications.

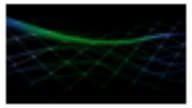


### cuBLAS

NVIDIA CUDA BLAS Library (cuBLAS) is a GPU-accelerated version of the complete standard BLAS library that delivers 6x to 17x faster performance than the latest MKL BLAS.


### ArrayFire

Comprehensive, open source GPU function library. Includes functions for math, signal and image processing, statistics, and many more. Interfaces for C, C++, Java, R and Fortran.



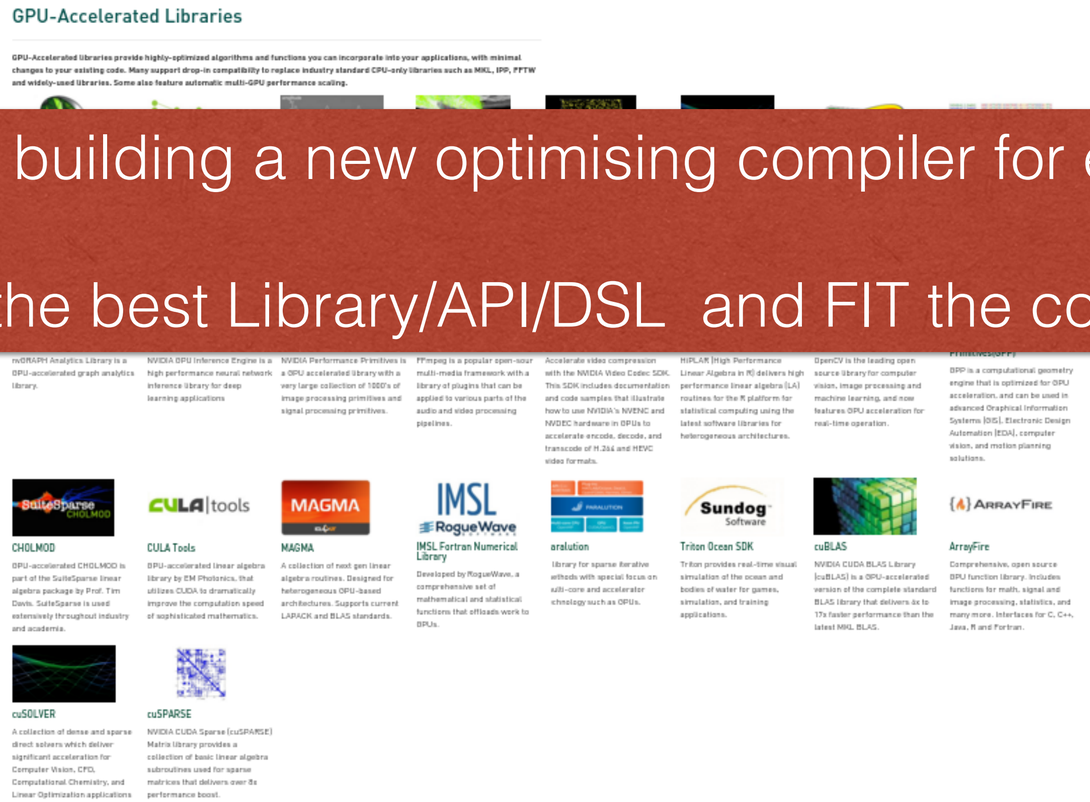
### cuSOLVER

A collection of dense and sparse direct solvers which deliver significant acceleration for Computer Vision, CFD, Computational Chemistry, and Linear Optimization applications

### cuSPARSE

NVIDIA CUDA Sparse (cuSPARSE) Matrix library provides a collection of basic linear algebra subroutines used for sparse matrices that delivers over 8x performance boost.

Good performance is hard to get even with well defined parallel language CUDA/OpenCL



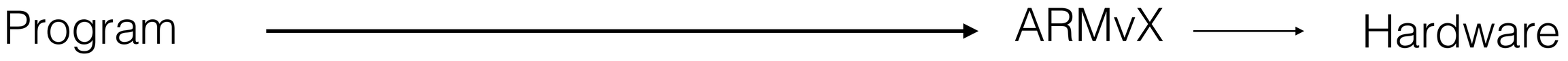

Rather than building a new optimising compiler for each platform

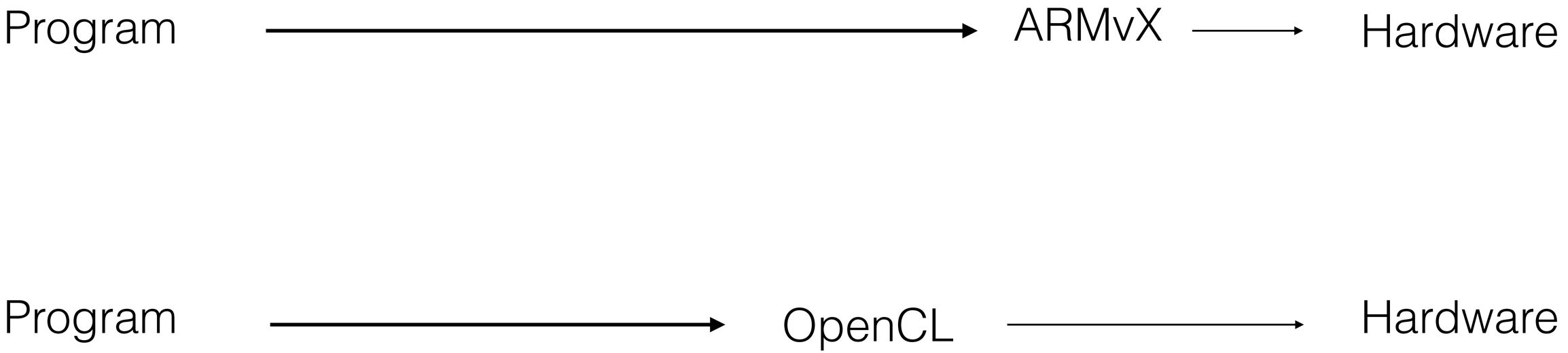
Pick the best Library/API/DSL  and FIT the code to it

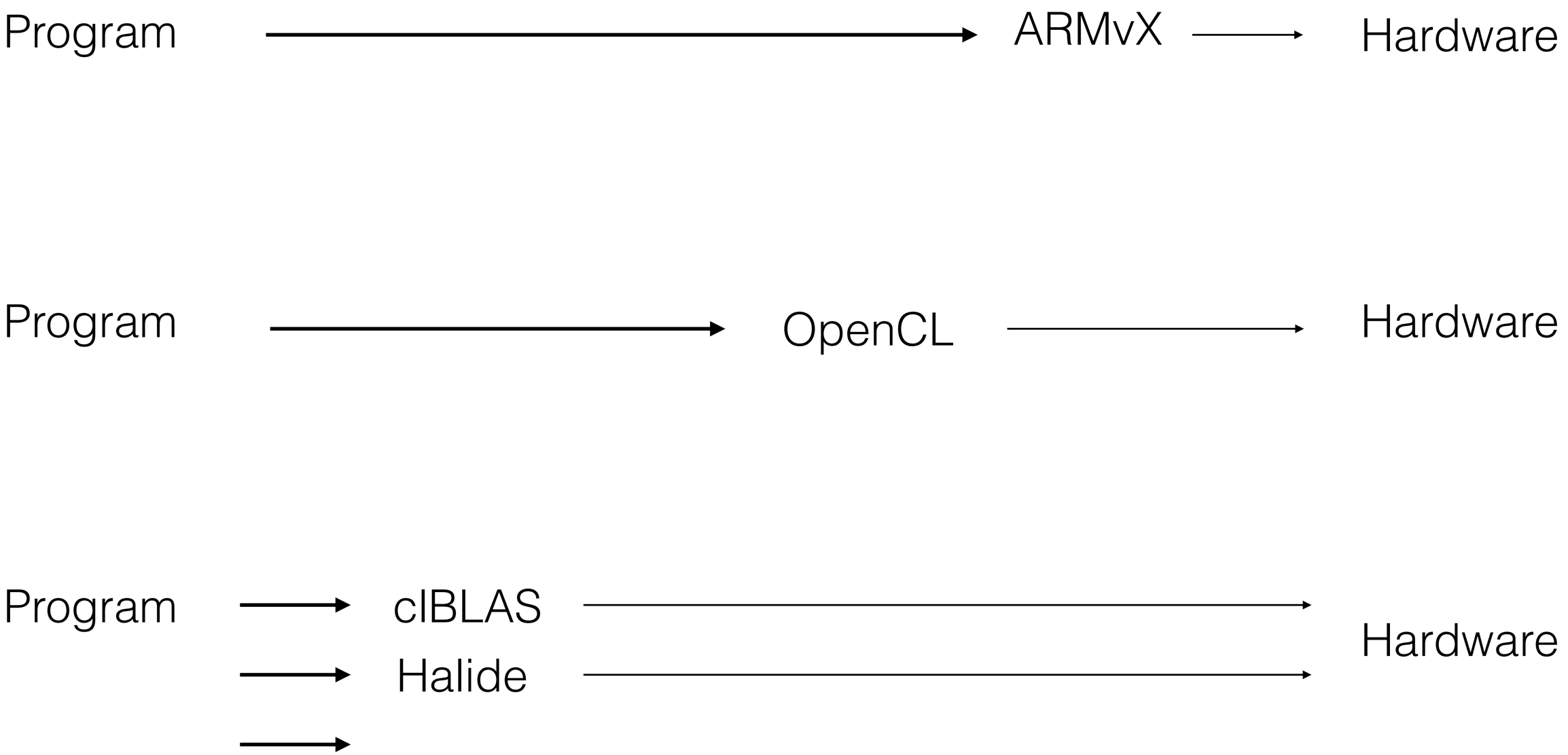

Rather than building a new optimising compiler for each platform

Pick the best Library/API/DSL and FIT the code to it

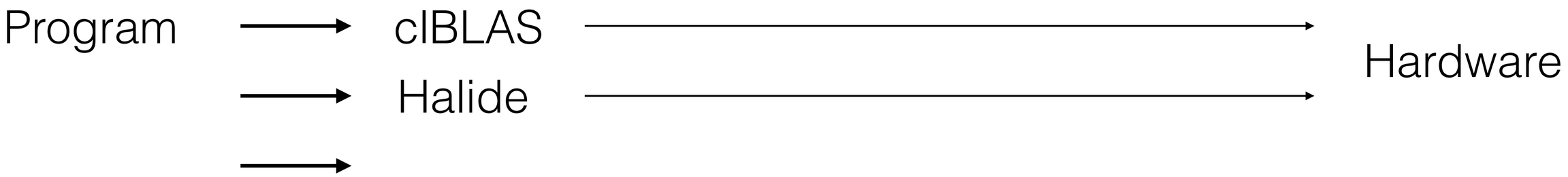
# Libraries/DSLs are the new ISA


Program
ARMvX
Hardware

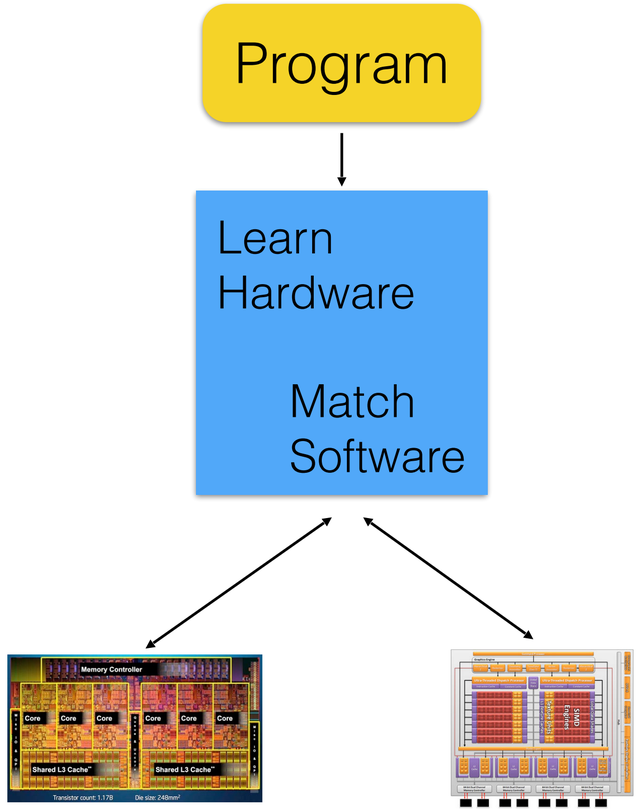

Program
ARMvX
Hardware
Program
OpenCL
Hardware


Program
ARMvX
Hardware
Program
OpenCL
Hardware
Program
clBLAS
Halide
Hardware

+ Interface nearer to algorithm
- Interface complex and changeable

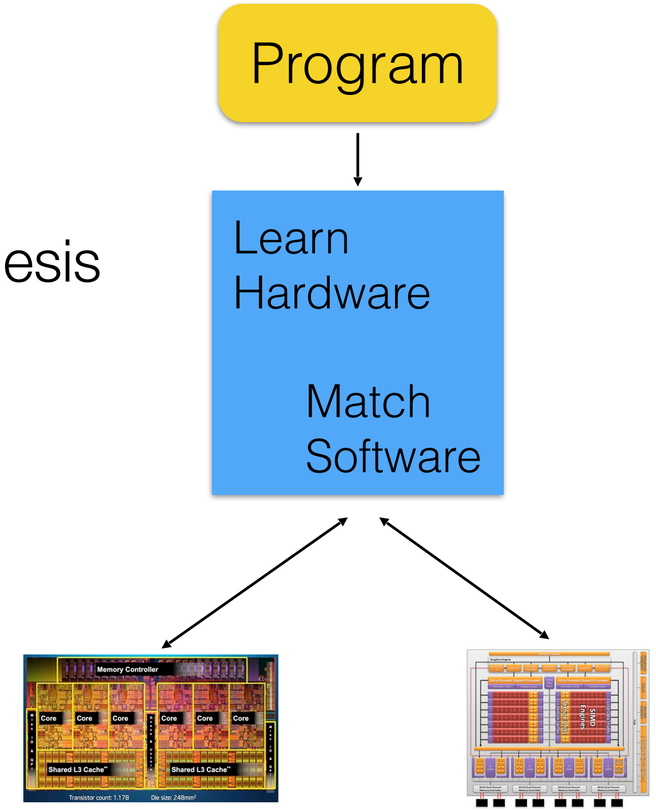


# Detect code structures that match interface



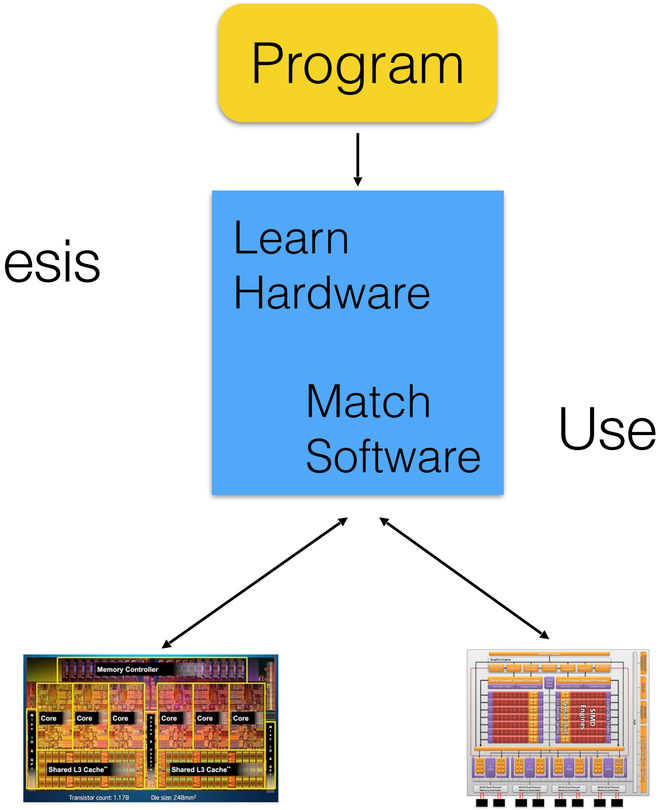
# Detect code structures that match interface

Use IO grey-box program synthesis



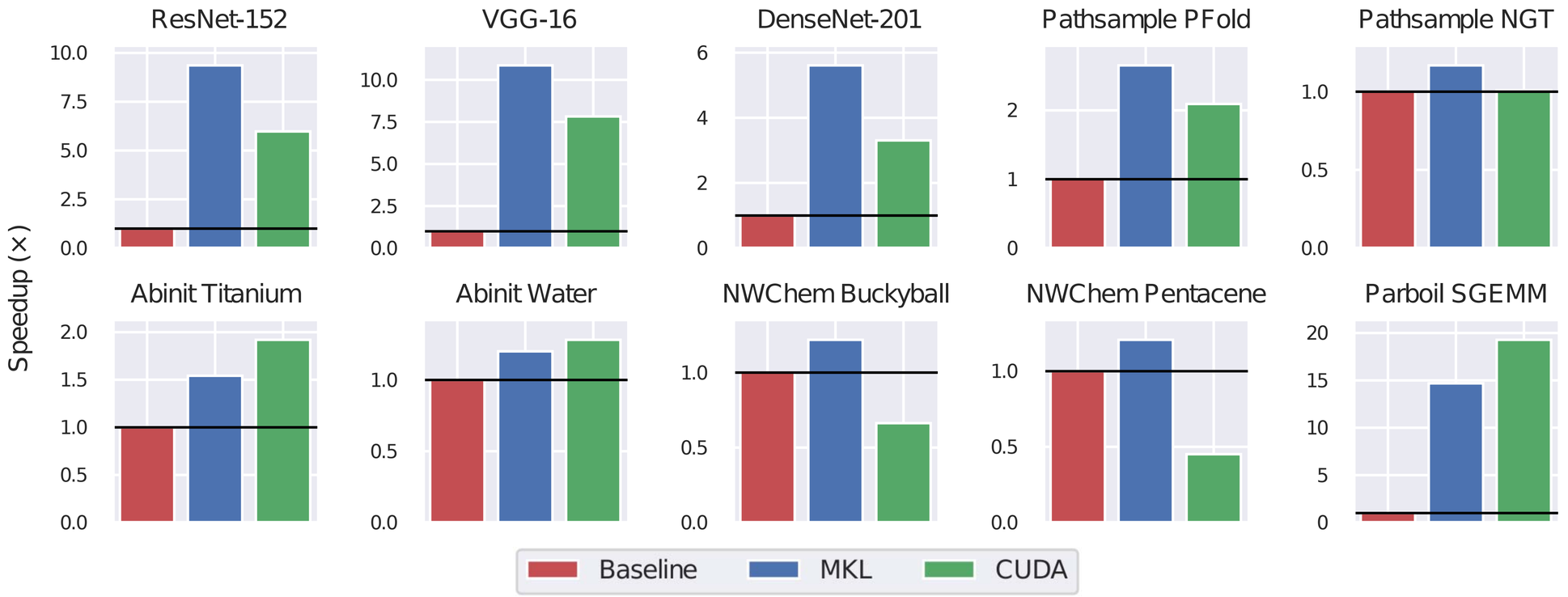
# Detect code structures that match interface

Use IO grey-box program synthesis



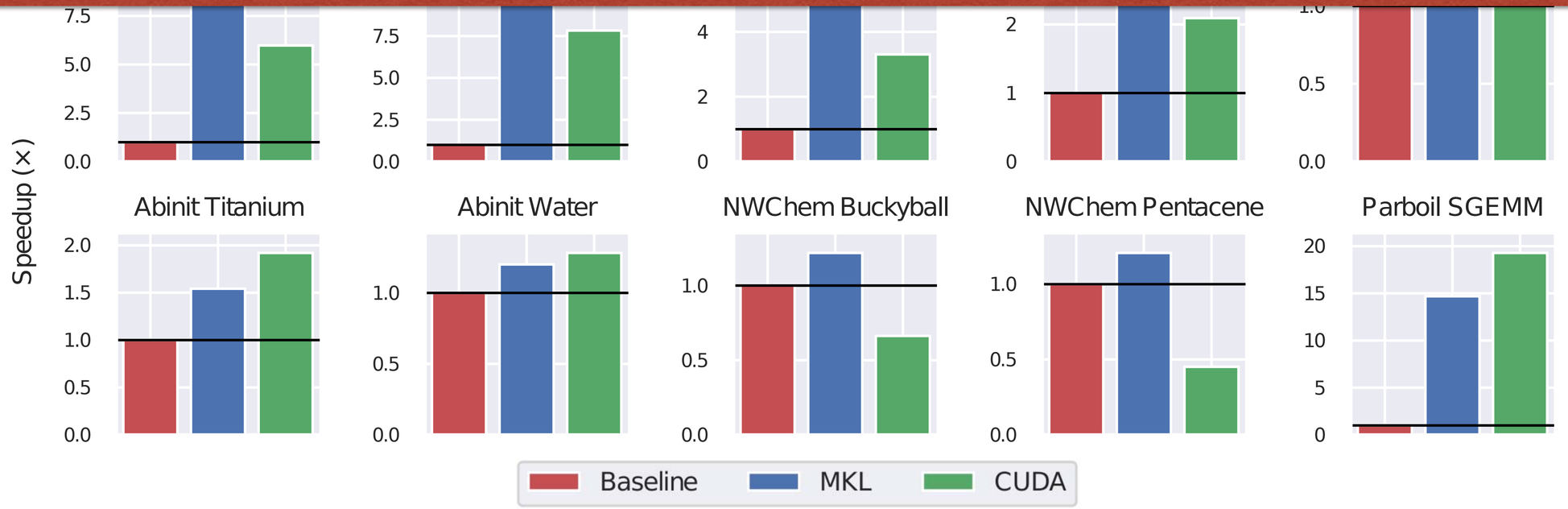

Use SMT solver/graph matching/synthesis

# It works



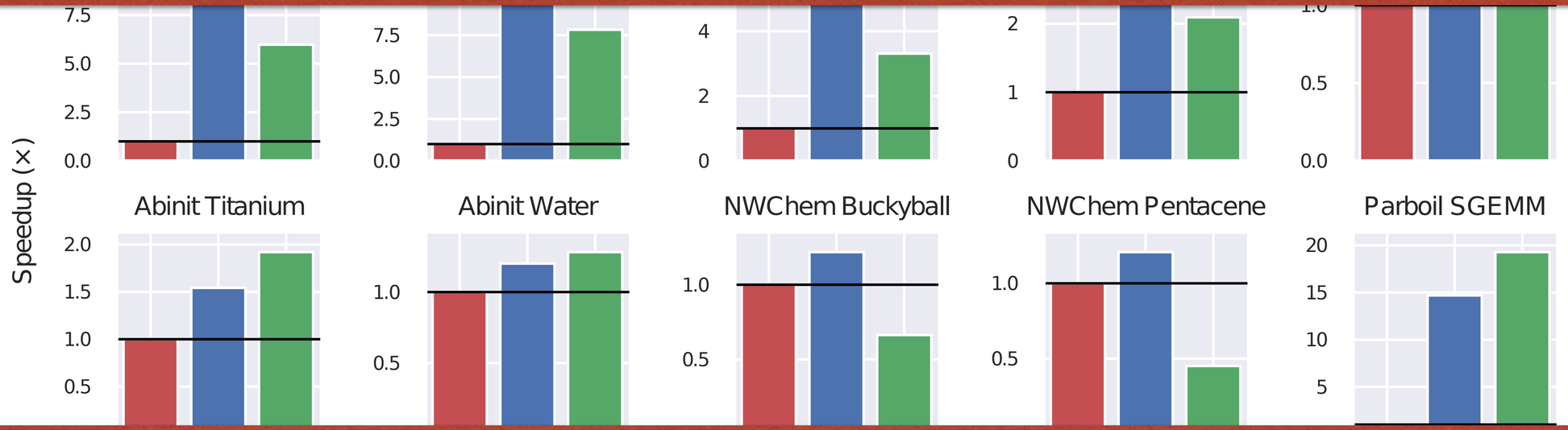

[ASPLOS18] [PACT19] [GPCE20] [PACT21]

# Automatically matches accelerator libraries to legacy code




[ASPLOS18] [PACT19] [GPCE20] [PACT21]

# Automatically matches accelerator libraries to legacy code



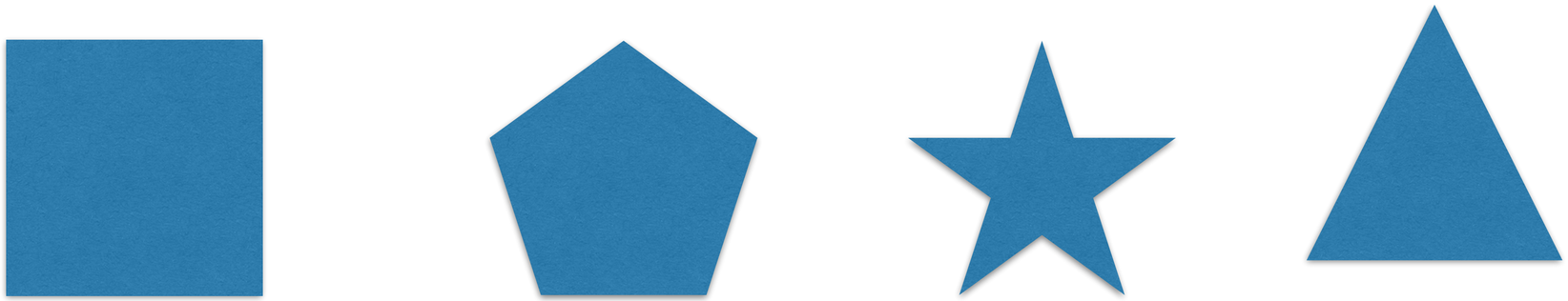

# No programmer in the loop

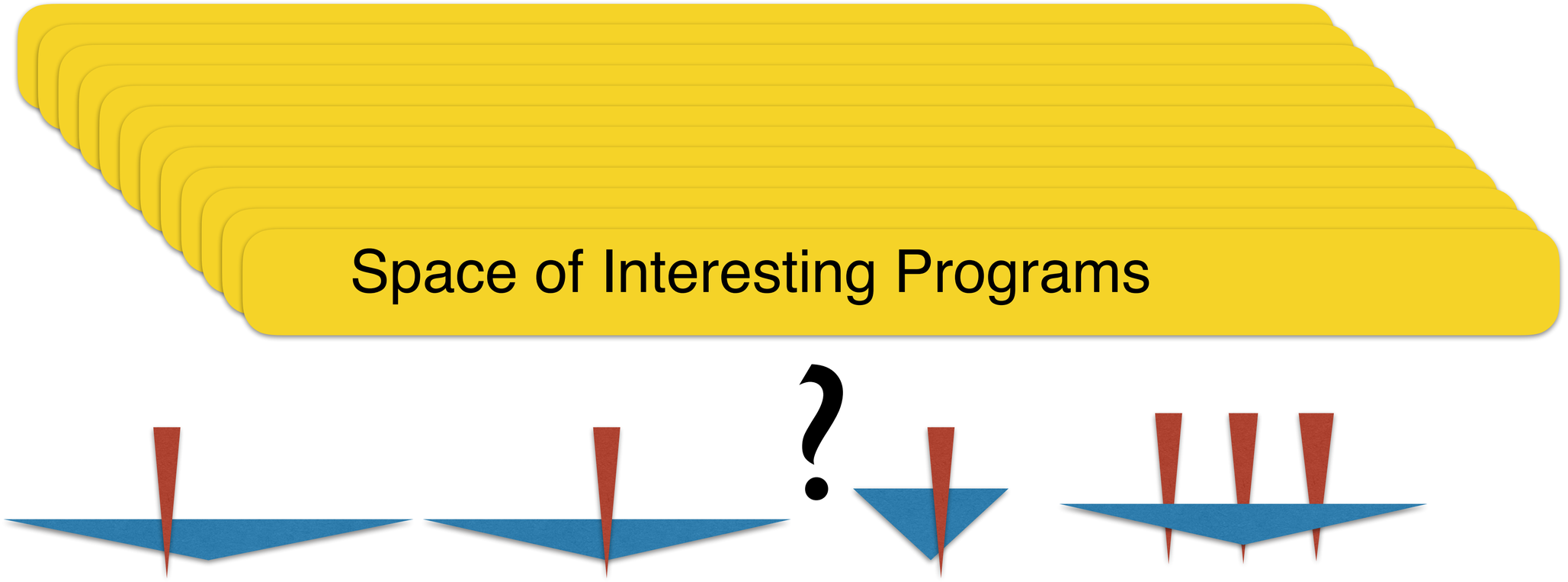
[ASPLOS18] [PACT19] [GPCE20] [PACT21]

Automatically matching APIs frees up hardware creativity

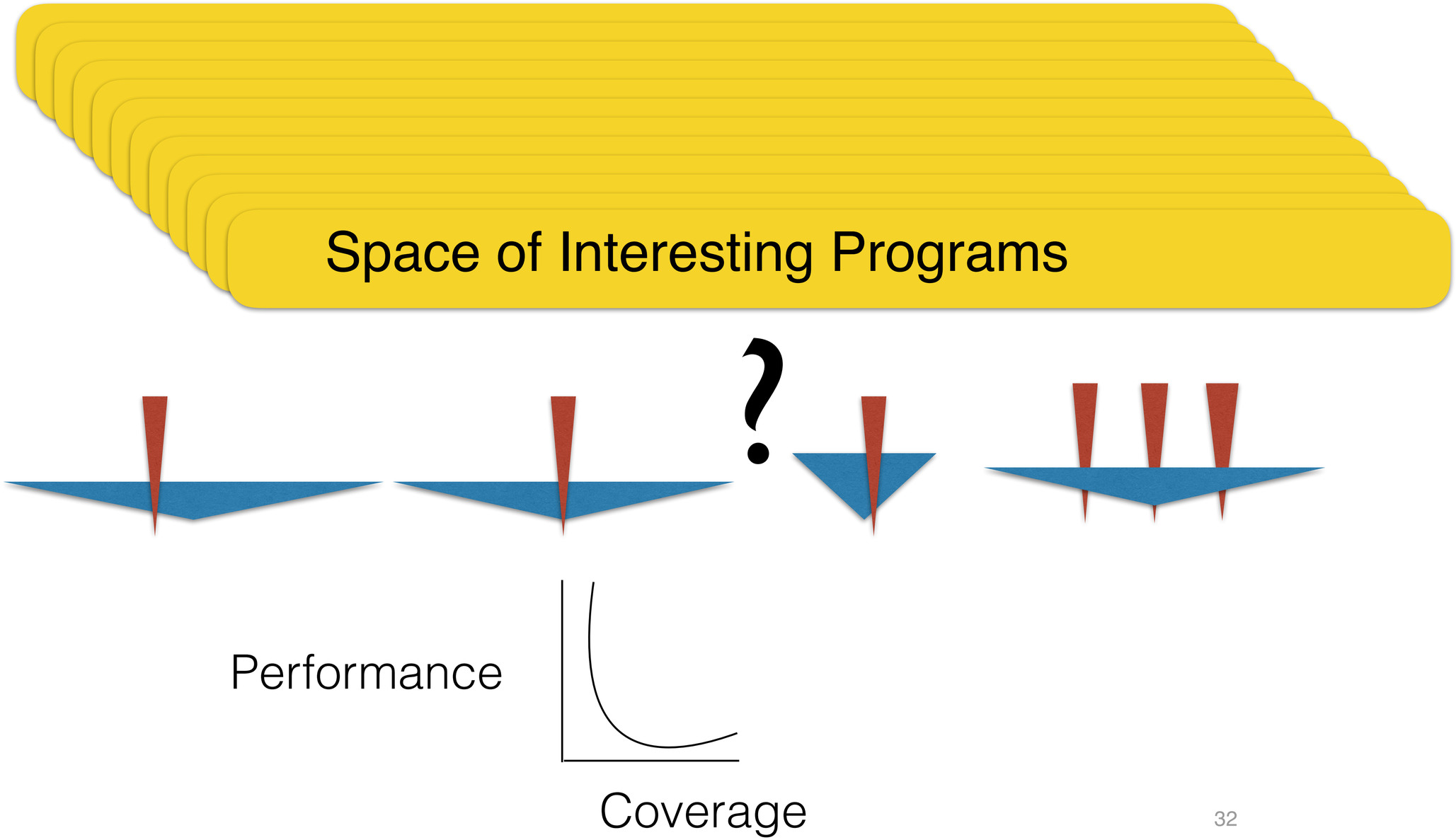
Space of Interesting Programs

?

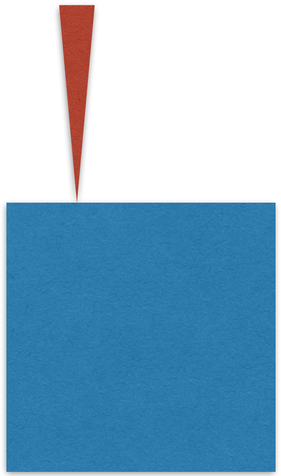


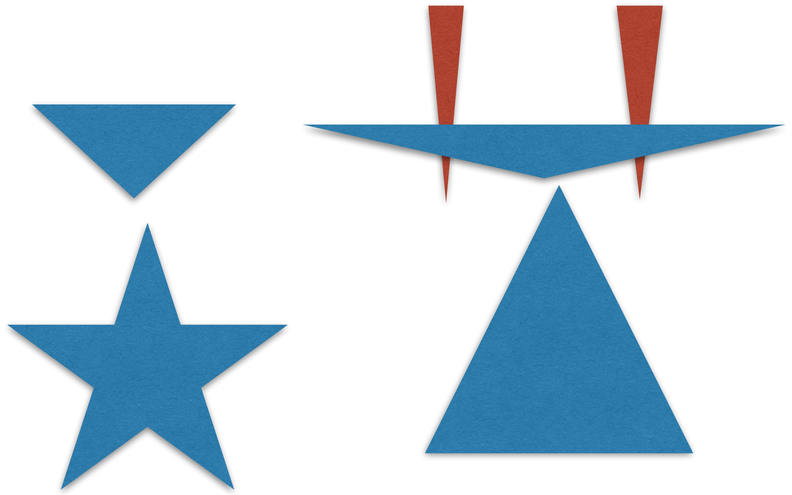



+Performance/Cost Estimate


Space of Interesting Programs
?
Performance
Coverage

# Space of Interesting Programs





Matching Hardware to Software - hardware defined software (HDS)

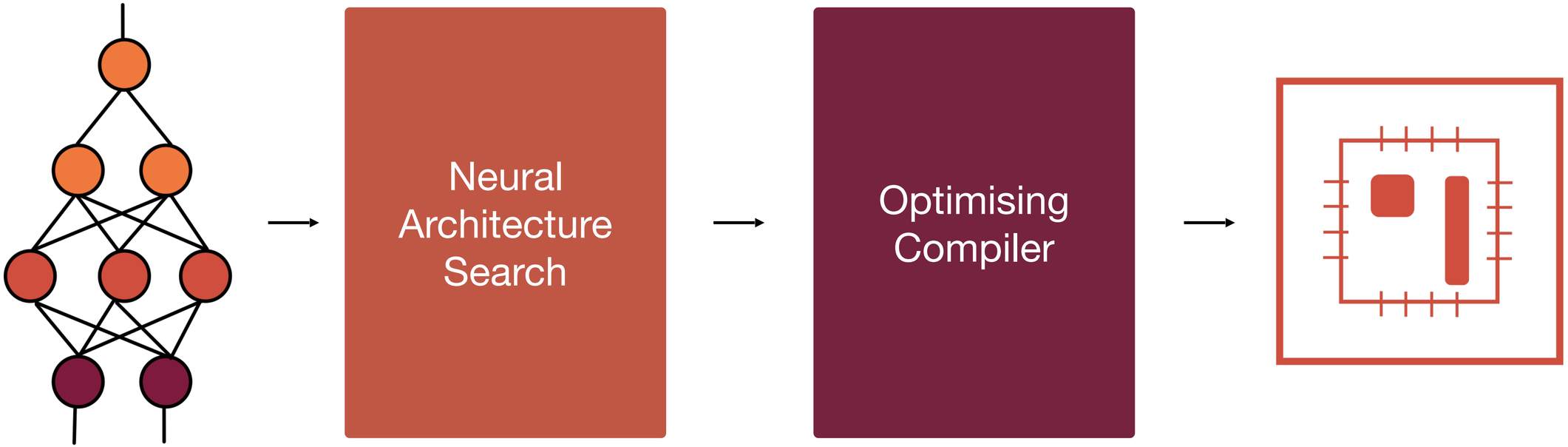
**Other**

**- Neural Architecture Search as Program Transformation Exploration**

- Software Defined  Hardware (SDH)

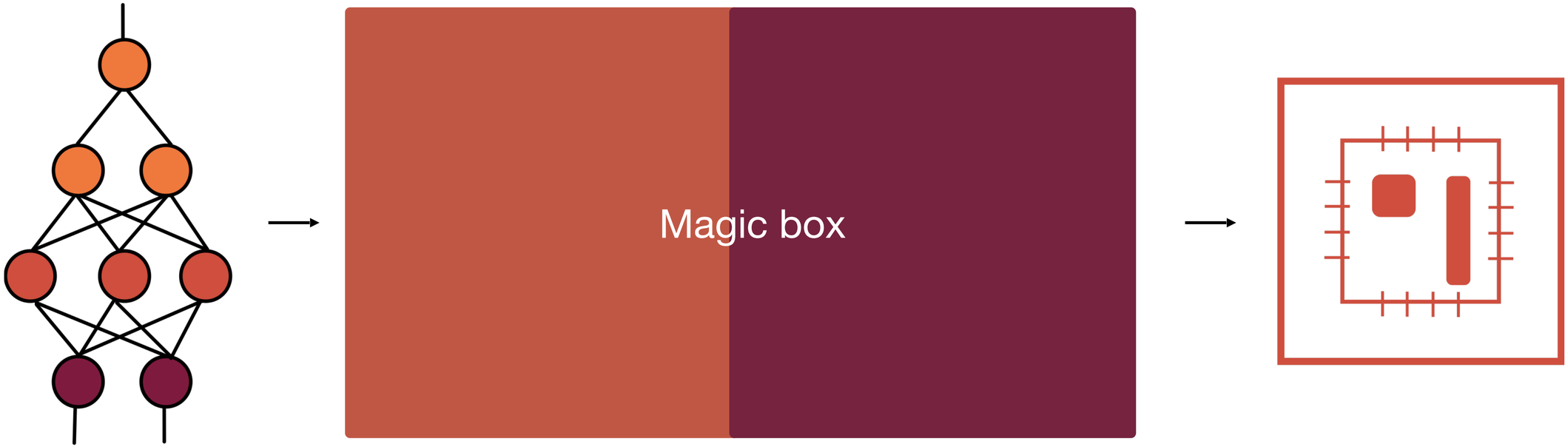
Beyond Simple Acceleration

# How we deploy neural networks




[ASPLOS 2021 Distinguished paper]

# Unifying the optimisation steps



```python
for i in range(10):
  for j in range(20):
    A[i][j] = i*j
```

Optimising Compiler

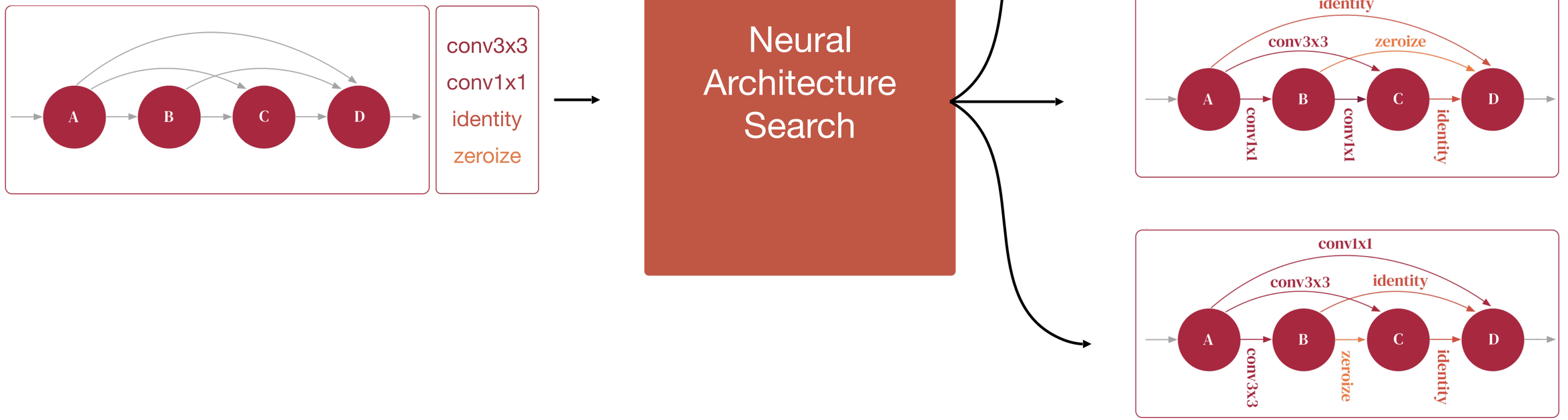


```
120:                                              ; preds = %116
  %121 = load i32, i32* %7, align 4
  %122 = sext i32 %121 to i64
  %123 = getelementptr inbounds [3 x [3 x i32]], [3 x [3 x i32]]* %10, i64 0, i64 %122
  %124 = load i32, i32* %9, align 4
  %125 = sext i32 %124 to i64
  %126 = getelementptr inbounds [3 x i32], [3 x i32]* %123, i64 0, i64 %125
  %127 = load i32, i32* %126, align 4
  %128 = load i32, i32* %9, align 4
  %129 = sext i32 %128 to i64
  %130 = getelementptr inbounds [3 x [3 x i32]], [3 x [3 x i32]]* %11, i64 0, i64 %129
  %131 = load i32, i32* %8, align 4
  %132 = sext i32 %131 to i64
  %133 = getelementptr inbounds [3 x i32], [3 x i32]* %130, i64 0, i64 %132
  %134 = load i32, i32* %133, align 4
  %135 = mul nsw i32 %127, %134
  %136 = load i32, i32* %7, align 4
  %137 = sext i32 %136 to i64
  %138 = getelementptr inbounds [10 x [10 x i32]], [10 x [10 x i32]]* %2, i64 0, i64 %137
  %139 = load i32, i32* %8, align 4
  %140 = sext i32 %139 to i64
  %141 = getelementptr inbounds [10 x i32], [10 x i32]* %138, i64 0, i64 %140
  %142 = load i32, i32* %141, align 4
  %143 = add nsw i32 %142, %135
  store i32 %143, i32* %141, align 4
  br label %144
```

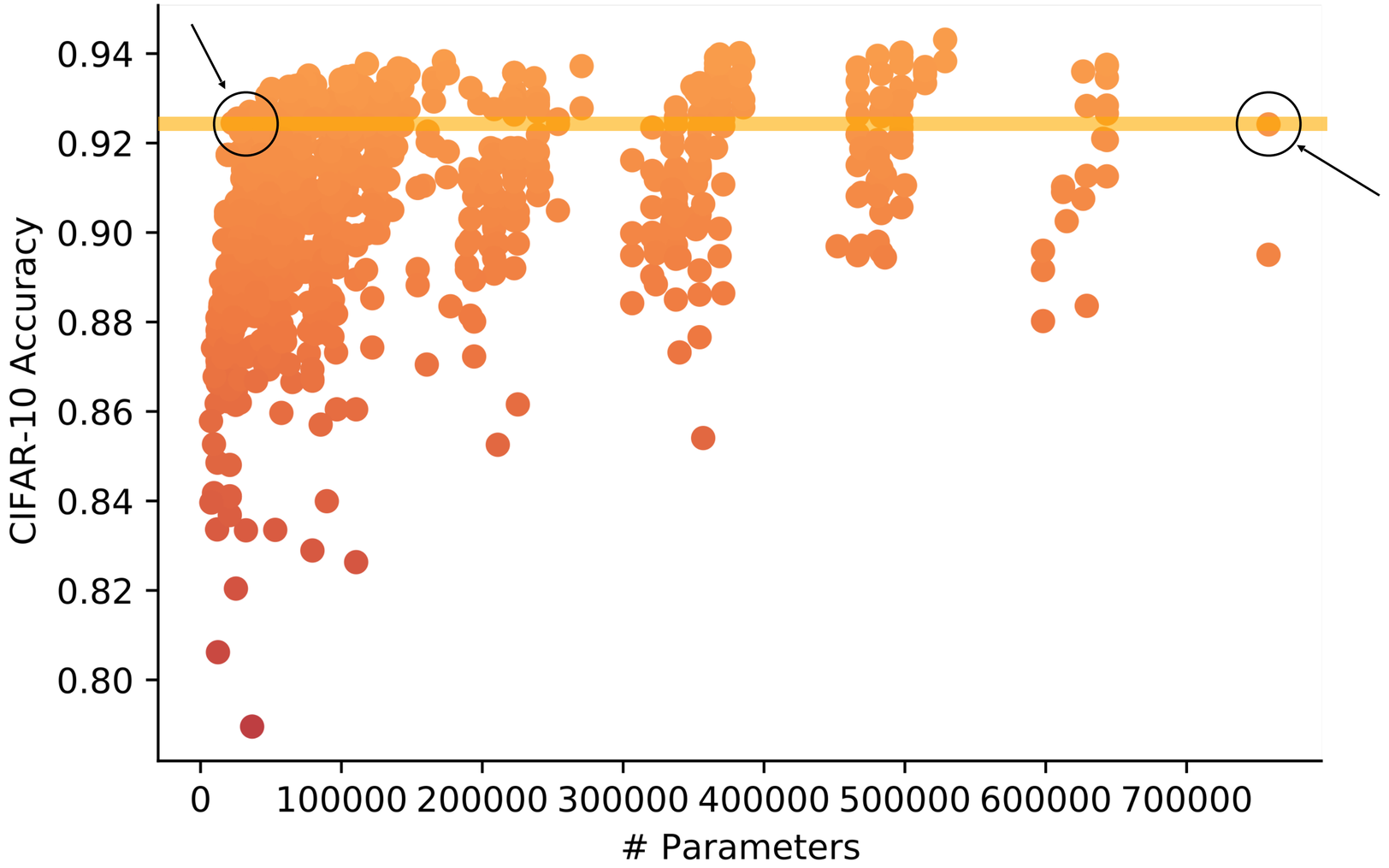


```
51:
  %52 = phi i64 [ 0, %47 ], [ %60, %51 ]
  %53 = phi i32 [ %50, %47 ], [ %59, %51 ]
  %54 = getelementptr inbounds [3 x [3 x i32]]
  %55 = load i32, i32* %54, align 4, !tbaa !3
  %56 = getelementptr inbounds [3 x [3 x i32]]
  %57 = load i32, i32* %56, align 4, !tbaa !3
  %58 = mul nsw i32 %57, %55
  %59 = add nsw i32 %53, %58
  %60 = add nuw nsw i64 %52, 1
  %61 = icmp eq i64 %60, 3
  br i1 %61, label %62, label %51, !llvm.loop !13
```

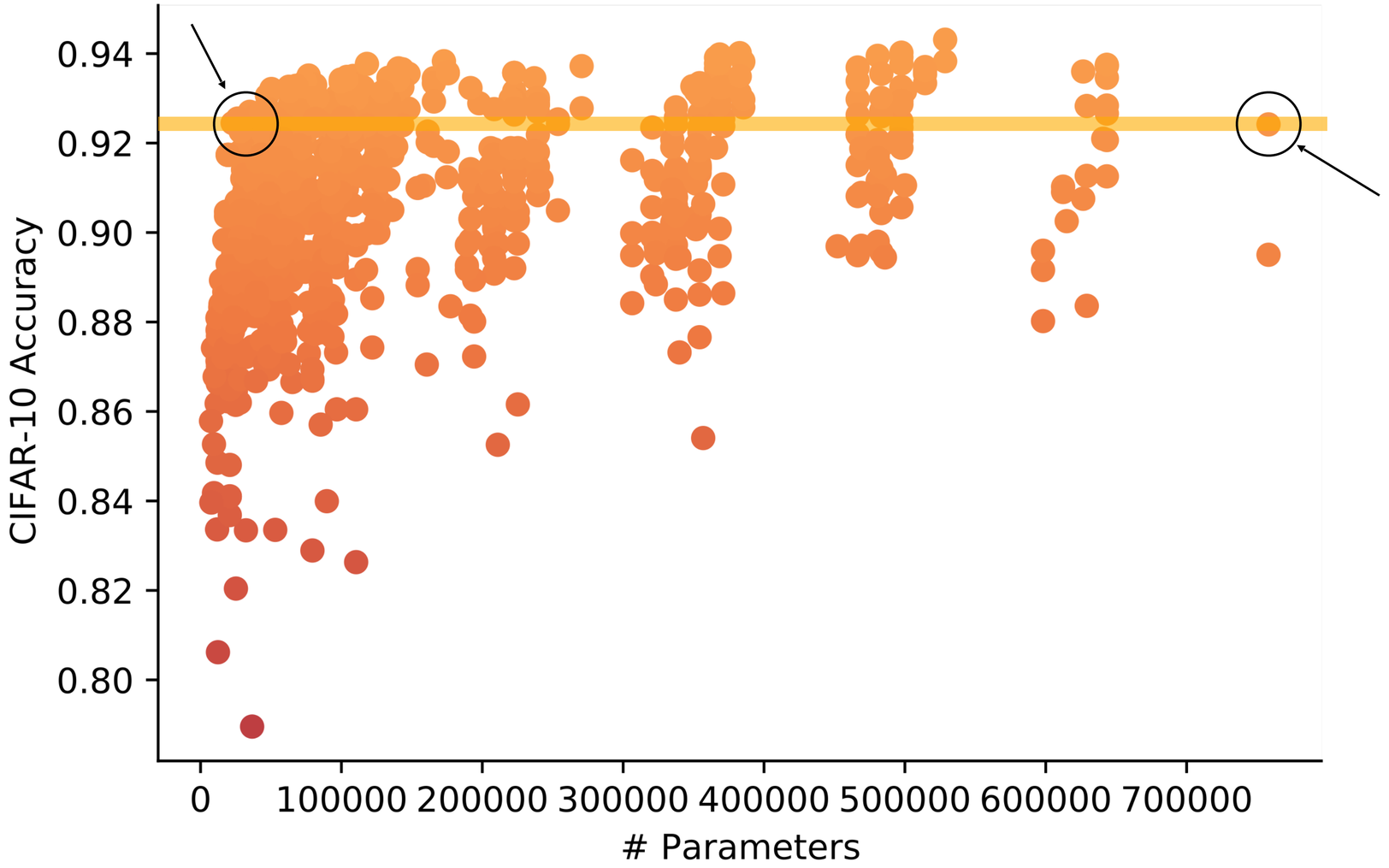
```
34:                                               ; preds = %31
  %35 = add nuw i64 %18, 16
  %36 = and i64 %35, -16
  %37 = tail call noalias nonnull i8* @_Znwm(i64 %36) #11
  %38 = getelementptr inbounds %"class.std::__1::basic_string", %
  store i8* %37, i8** %38, align 8, !tbaa !21
  %39 = or i64 %36, -9223372036854775808
  %40 = getelementptr inbounds %"class.std::__1::basic_string", %
  store i64 %39, i64* %40, align 8, !tbaa !21
  %41 = getelementptr inbounds %"class.std::__1::basic_string", %
  store i64 %18, i64* %41, align 8, !tbaa !21
  %42 = bitcast %"class.std::__1::basic_string"* %7 to %"struct.s
  br label %47
```

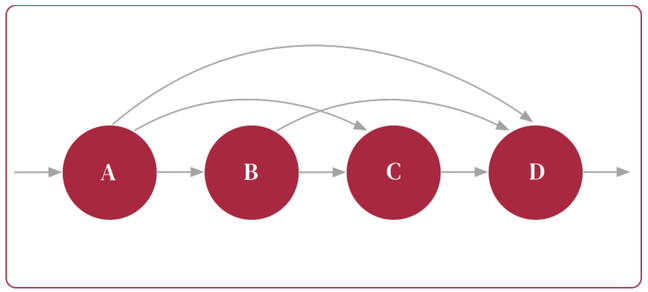

A
B
C
D
conv3x3
conv1x1
identity
zeroize
Neural Architecture Search
zeroize
identity
identity
A
B
C
D
conv3x3
conv1x1
conv3x3
identity
identity
zeroize
A
B
C
D
conv1x1
conv1x1
identity
conv1x1
conv3x3
identity
A
B
C
D
conv3x3
zeroize
identity

# Very different networks loosely “equivalent”



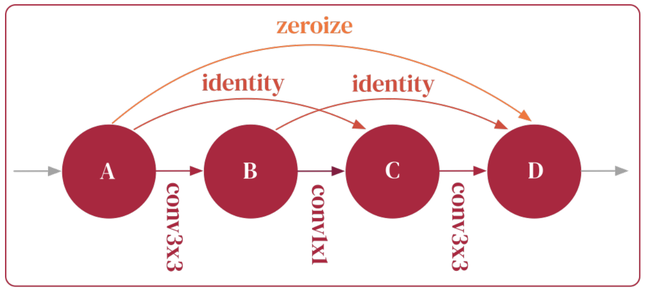
# Can we find  better equivalent networks?

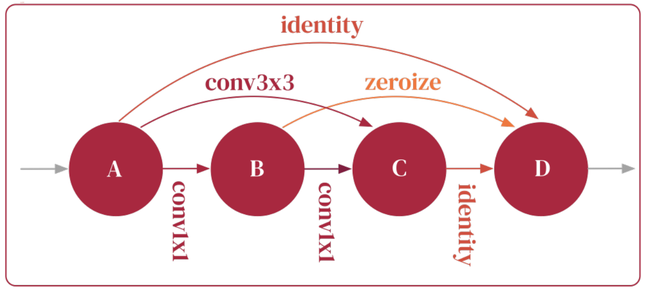


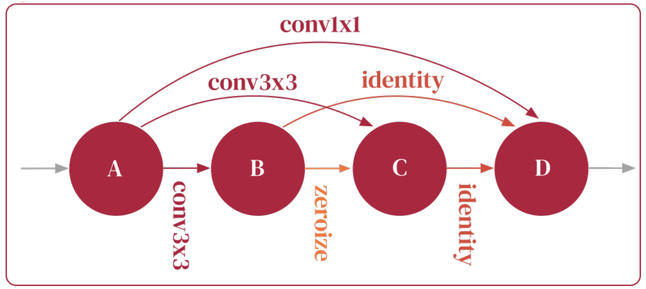



conv3x3
conv1x1
identity
zeroize

Neural Architecture Search



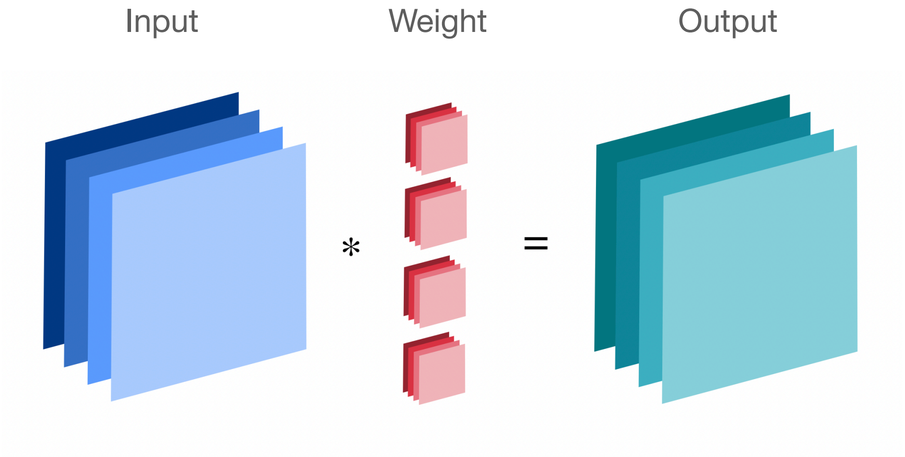




Neural Architectures are just programs
Can we characterise options as transformations?
Can we characterise loosely equivalent?
Then mix with compiler transformations

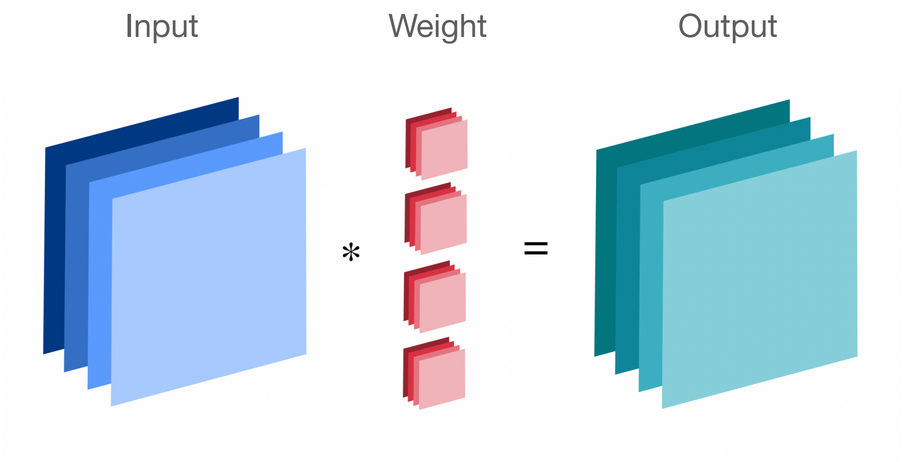
# Example

# Example optimisation target: convolution




```
for ci in [0,CI-1]:
 for co in [0,CO-1]:
  for oh in [0,OH-1]:
   for ow in [0,OW-1]:
    for kh in [0,KH-1]:
     for kw in [0,KW-1]:
      O[ci][co][oh][ow] +=
       W[ci][co][kh][kw]*
       I[ci][oh+kh][ow+kw]
```

# Example optimisation target: convolution

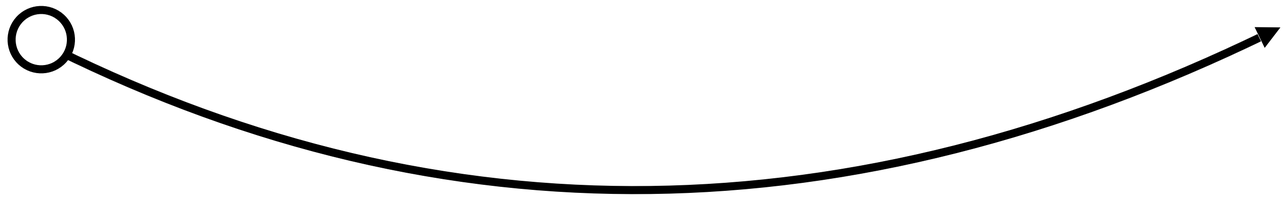



```
for ci in range(4):
    for co in range(4):
        spatial_conv(O, W, I, co, ci)
```

Reorder changes data access pattern

```
for ci in range(4):
    for co in range(4):
        spatial_conv(O, W, I, co, ci)
```

```
for co in range(4):
    for ci in range(4):
        spatial_conv(O, W, I, co, ci)
```

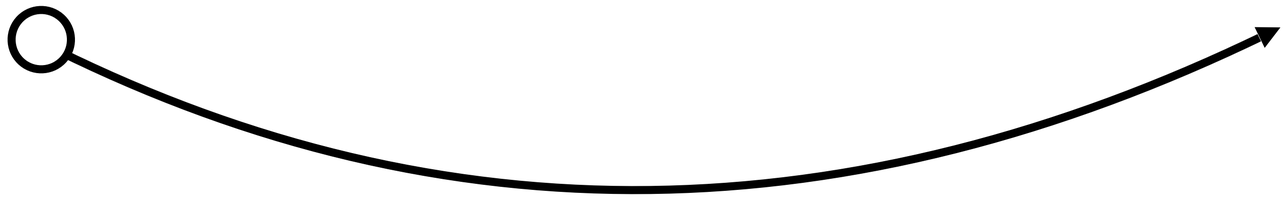


interchange (ci,co)

Reduce iteration domain
by factor B

```
for co in range(4):
    for ci in range(4):
        spatial_conv(O, W, I, co, ci)
```

```
for co in range(4/B):
    for ci in range(4):
        spatial_conv(O, W, I, co, ci)
```



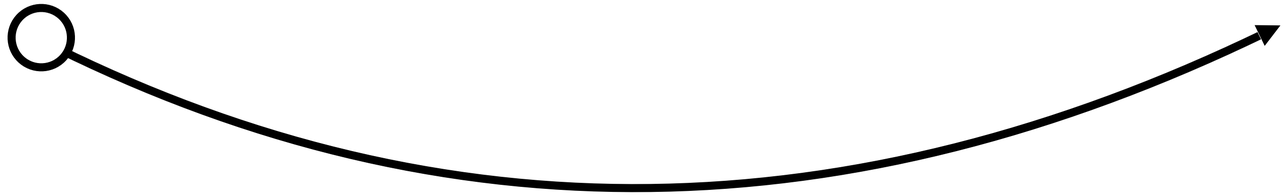
bottleneck

When is this OK?

Reduce iteration domain
by factor B

```
for co in range(4):
    for ci in range(4):
        spatial_conv(O, W, I, co, ci)
```

```
for co in range(4/B):
    for ci in range(4):
        spatial_conv(O, W, I, co, ci)
```



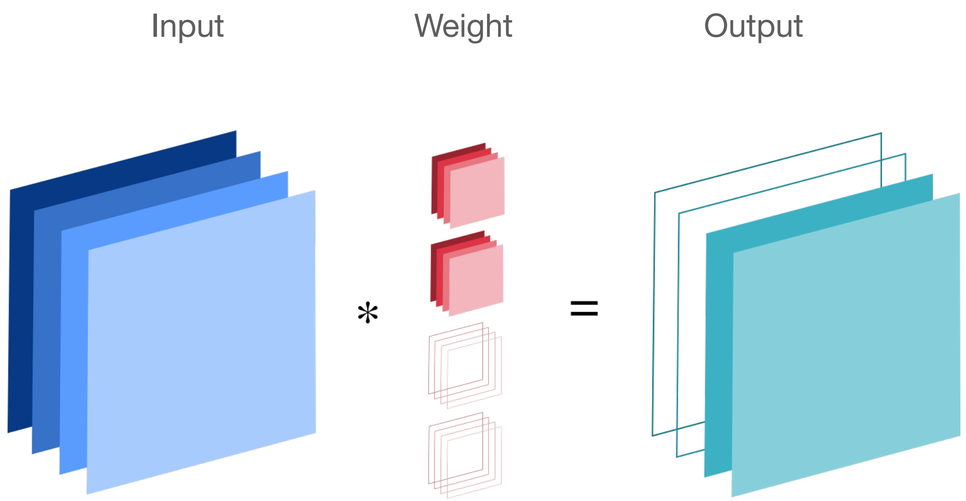
bottleneck

When is this OK?

**Fisher**
**See Amos Storkey talk**

# Transformations

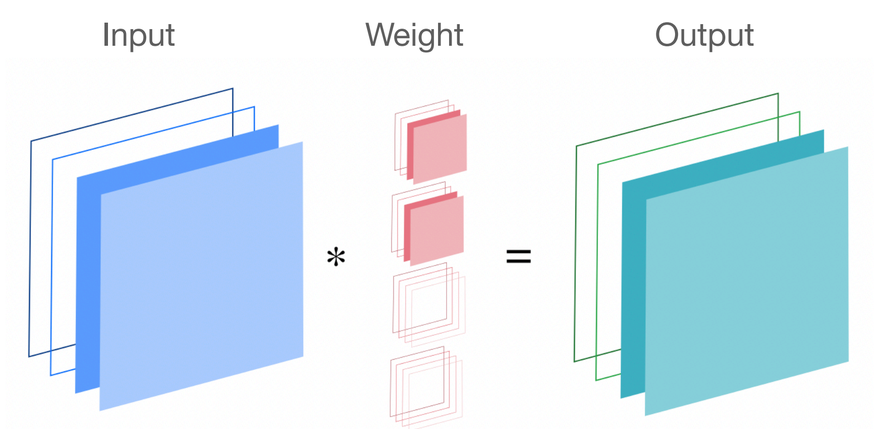
# Bottlenecking

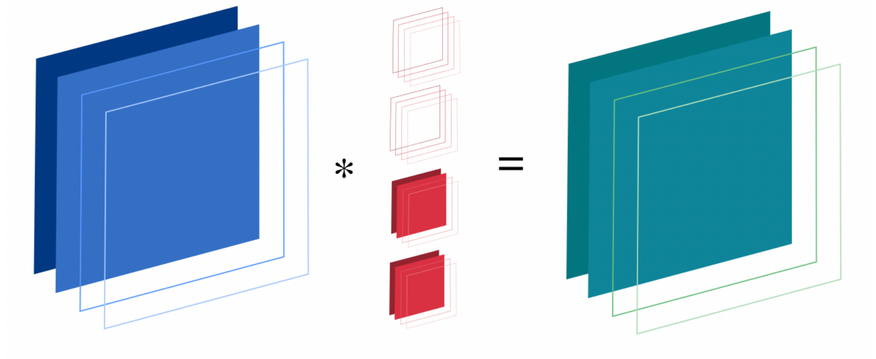



```
1 for co in range(4//B):
2     for ci in range(4):
3         spatial_conv(O, W, I, co, ci)
```

$$T(co, J') = (co', J') \mid co' < Co/B$$

# Grouping




```
for ci in range(0,2):
  for co in range(0,2):
    spatial_conv(O, W, I, co, ci)
for ci in range(2,4):
  for co in range(2,4):
    spatial_conv(O, W, I, co, ci)
```

$$T(co, ci, J'') = (g, co/G, ci/G, J')$$

# Transformation space

Compiler optimisations:

interchange

tile

unroll

prefetch

split

fuse

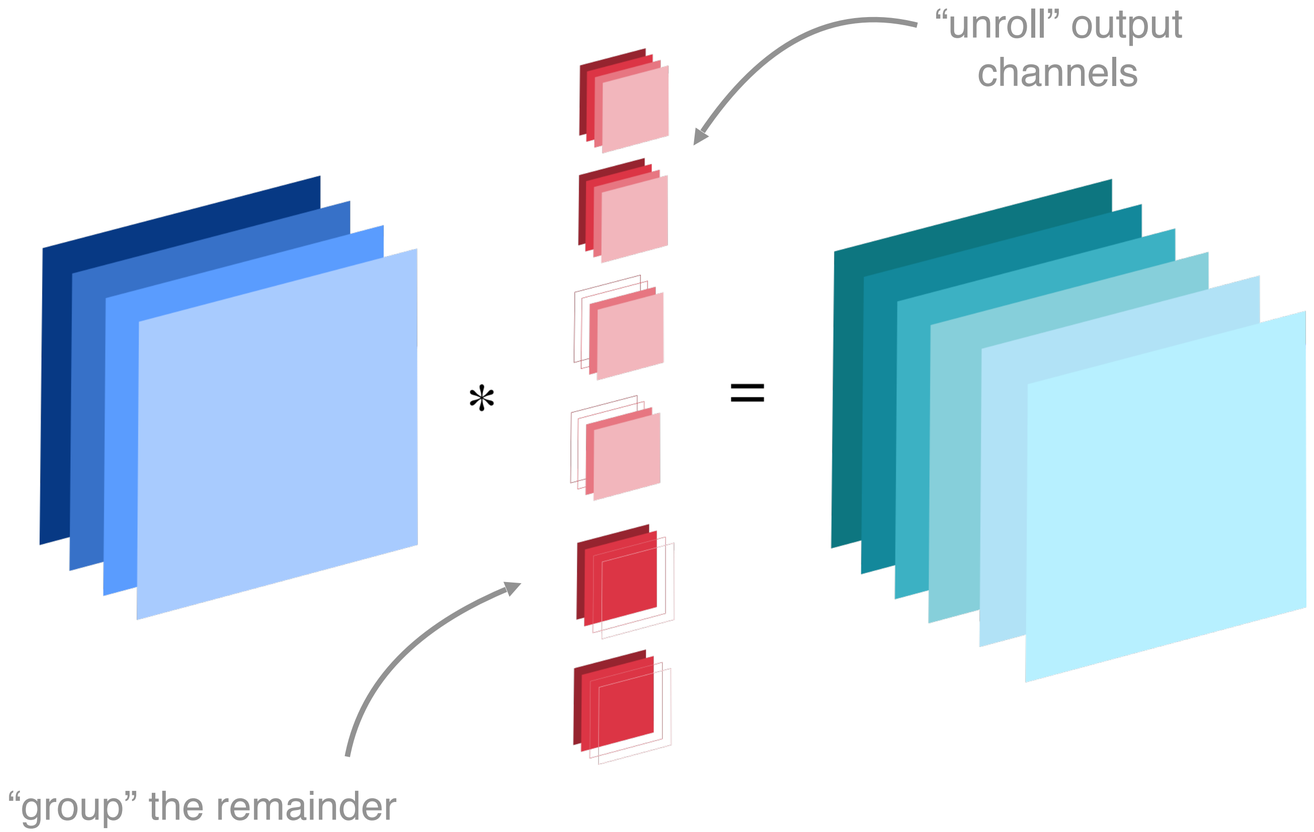
Network optimisations:

bottleneck

group

Can mix and match to give new convolutions

# Example: unrolled group convolution



T: [Co, Ci, H, W, Kh, Kw] ->

[H, W, Co, Ci, Kh, Kw] ->

[H(b), W, Co, Ci, Kh, Kw] ->

[W, H(b), Co, Ci, Kh, Kw] ->

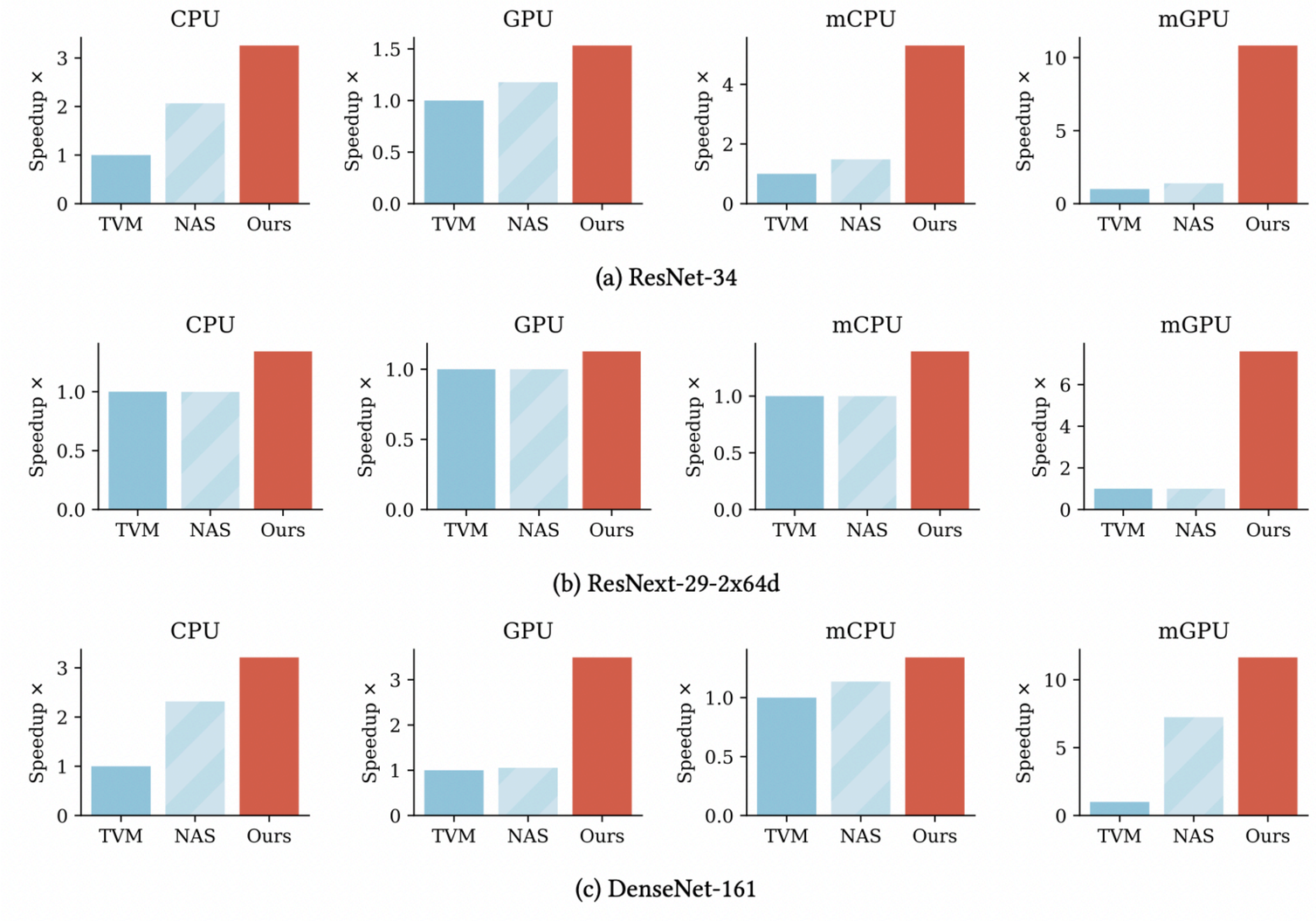
[W(b), H(b), Co, Ci, Kh, Kw] ->

[Co, Ci, H(b), W(b), Kh, Kw]

Spatial bottlenecking is bottlenecking plus interchange

# Results

# CIFAR-10: average 4x speedup over best

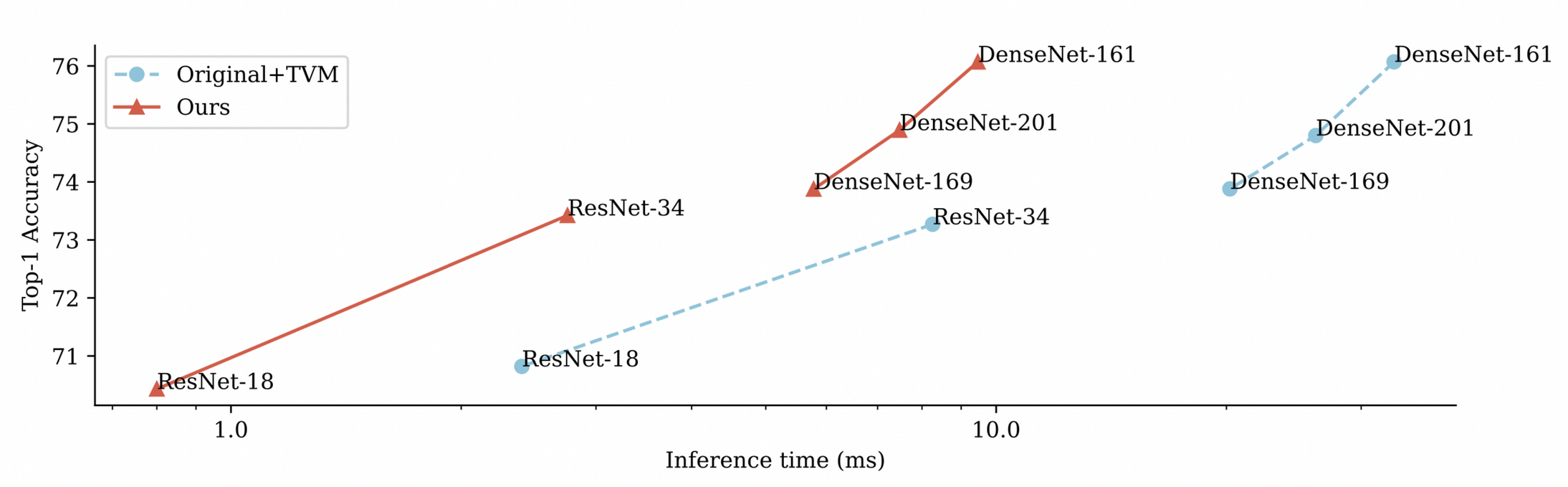



Compiling the original network with TVM

NAS-compression, compiled by TVM

TVM with our additional transformations as options.

# ImageNet: order magnitude improvement




Ported to Transmuter - see next topic
Darpa workloads:
- reduce exec time by 80.6%
- reduce energy by 79.4%

Matching Hardware to Software - hardware defined software (HDS)

**Other**

- Neural Architecture Search as Program Transformation Exploration

- **Software Defined  Hardware (SDH)**

Beyond Simple Acceleration

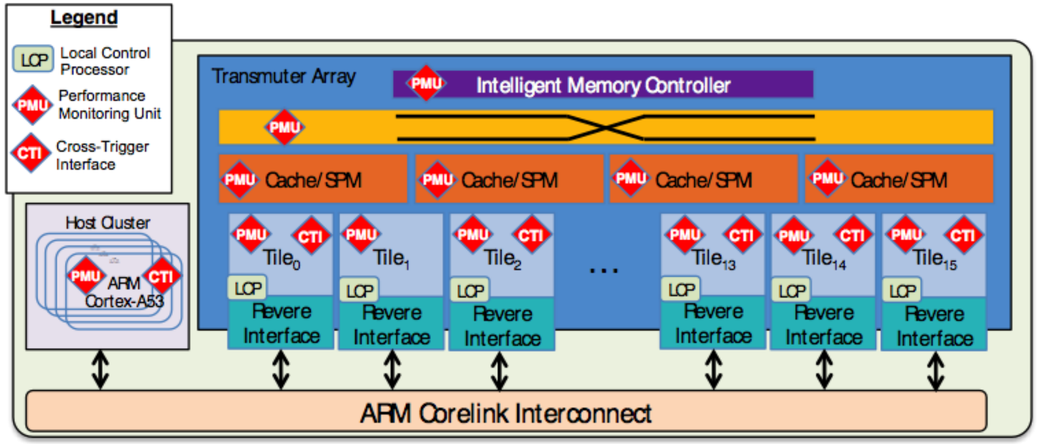
# Transmuter: Software Defined Hardware

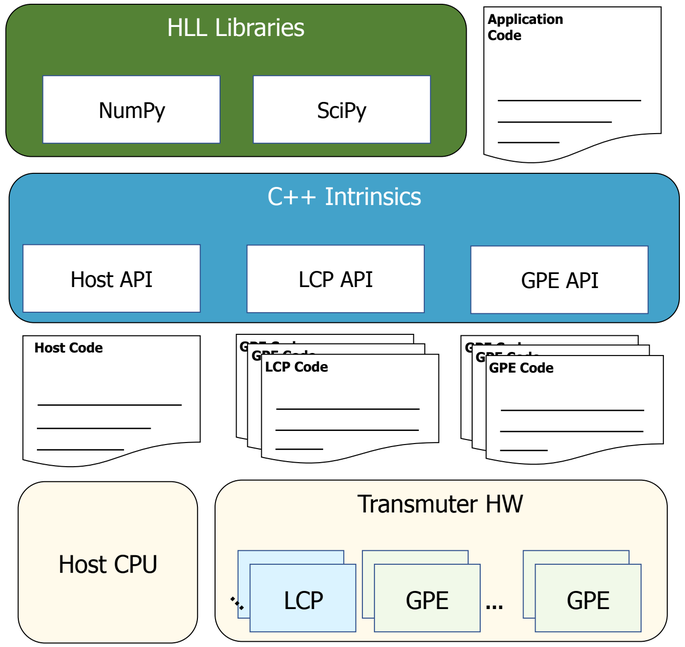
DARPA
- ARM, U Michigan, ASU, Edinburgh
- Python Stack, NumPy/SciPy acceleration
- Software monitors hardware and reconfigures

Coarse Grain Reconfigurable Multi-core
- ARM M-Class cores
- Fast reconfiguration
- Reconfigure cache/scratchpad, interconnect

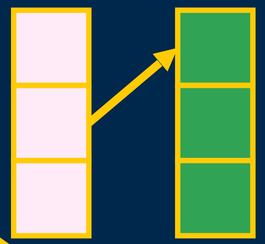
**Prodigy**
- Software assisted prefetcher(HPCA21 Best Paper)

**SparseAdapt**
- Runtime reconfiguration (Micro21)



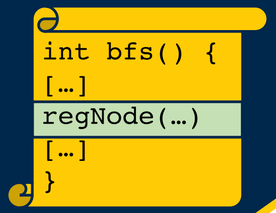



Irregular Memory Accesses



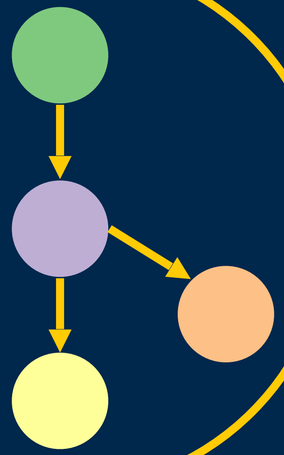
Compiler Analysis


Program Annotated

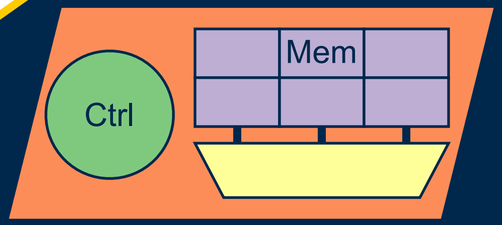



Software

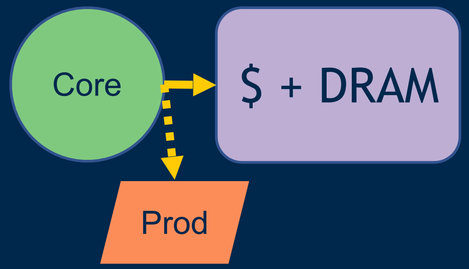
DIG



Hardware



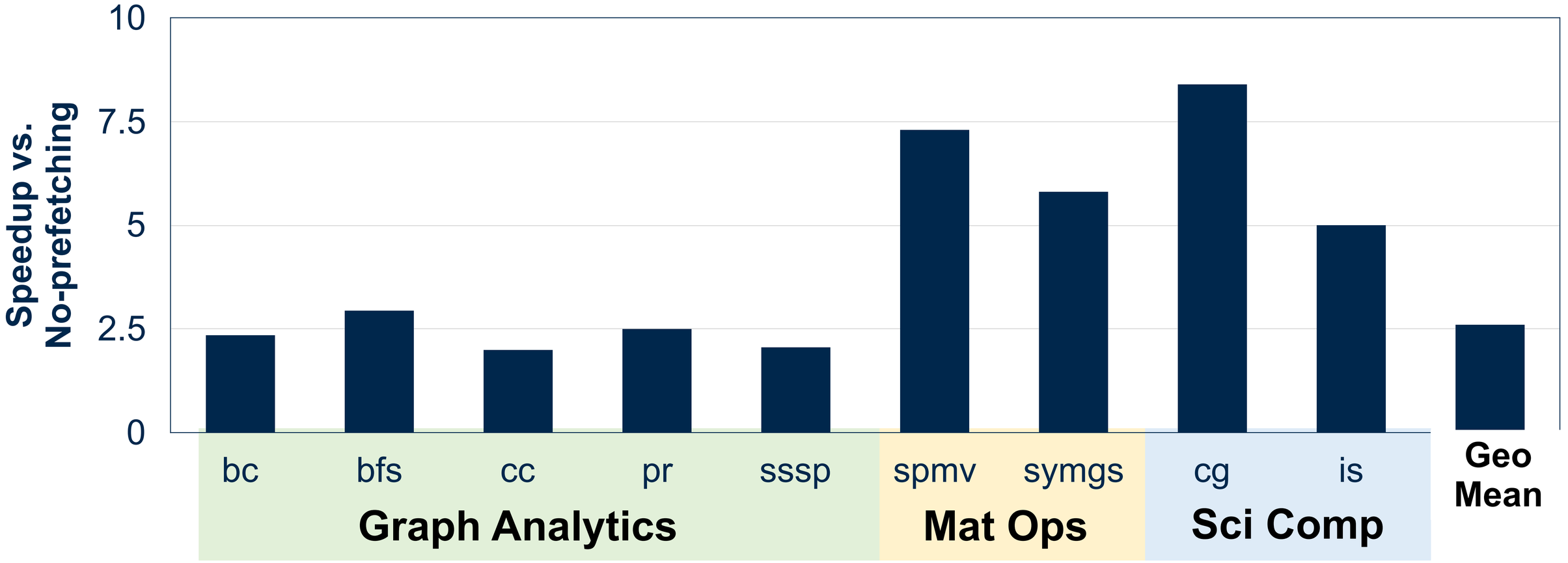

Programmable Prefetching Hardware




Prodigy Operation

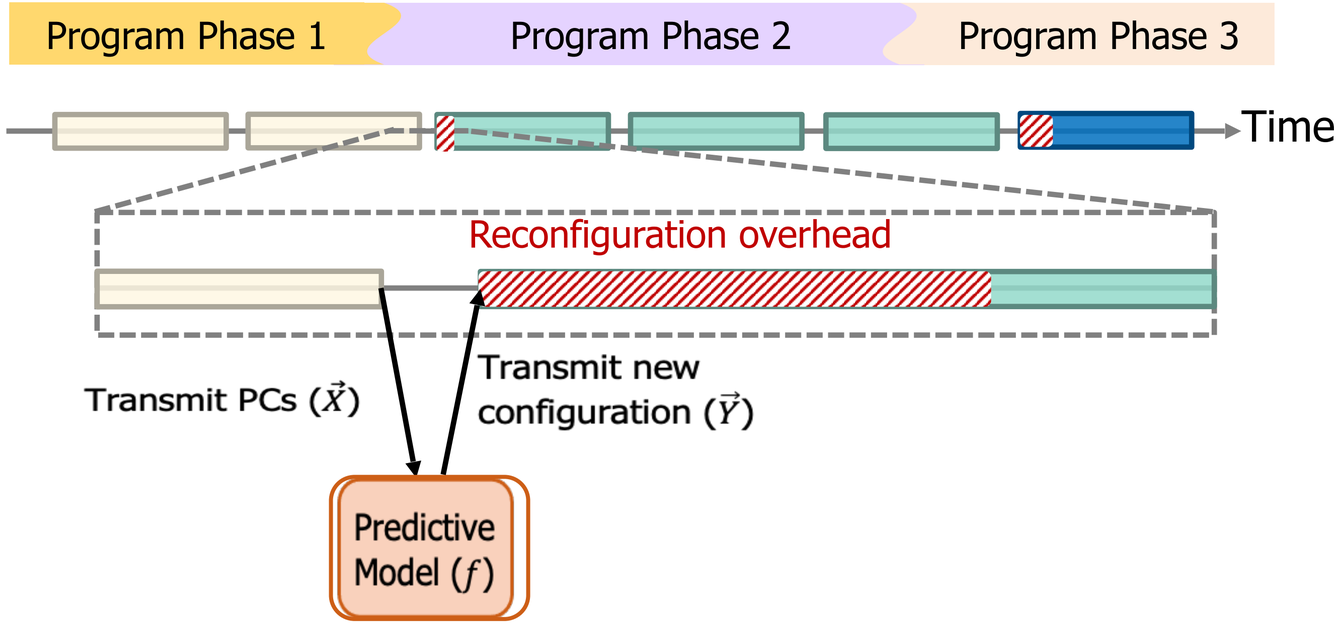
# Effect on Performance

## Speedup vs. No-Prefetching




**On average, 2.6x speedup compared to no prefetching**
**Reduction in DRAM-stalls by 80.3% and branch-stalls by 65.3%**

# Sparse Adapt: Hardware Reconfiguration

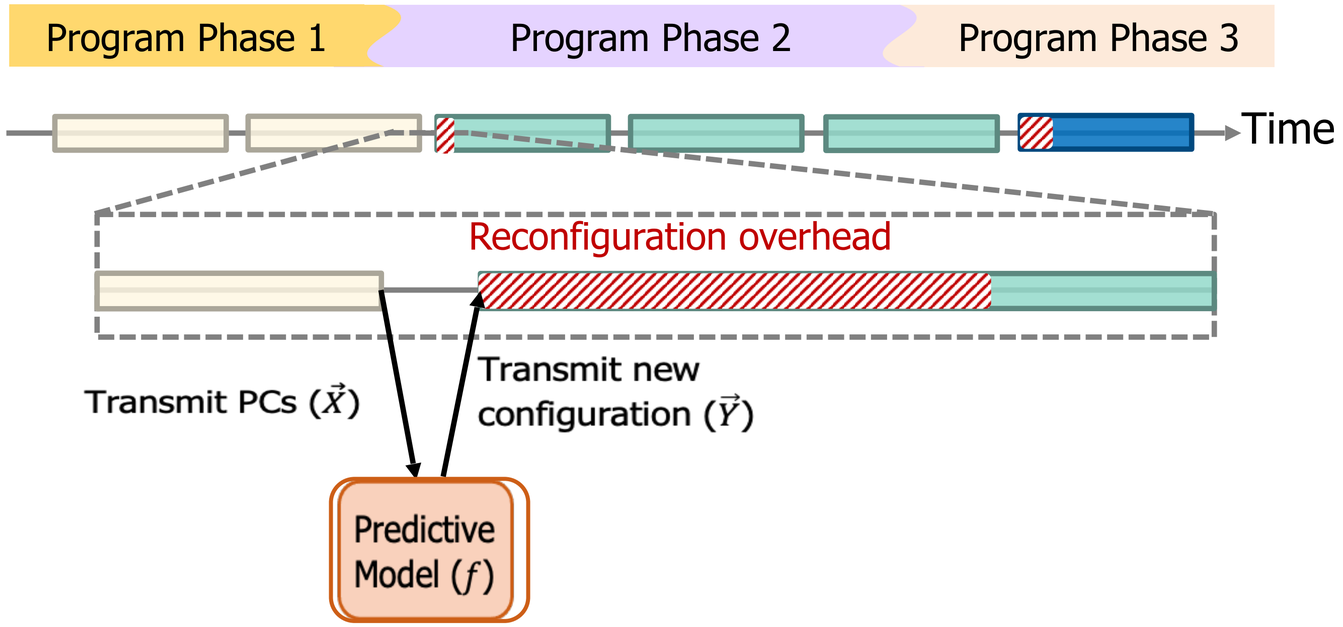



Model learns mapping f : X → Y

X: the space of performance counters

Y: best micro-architectural configurations

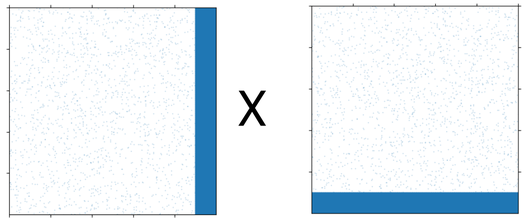
# Sparse Adapt: Hardware Reconfiguration

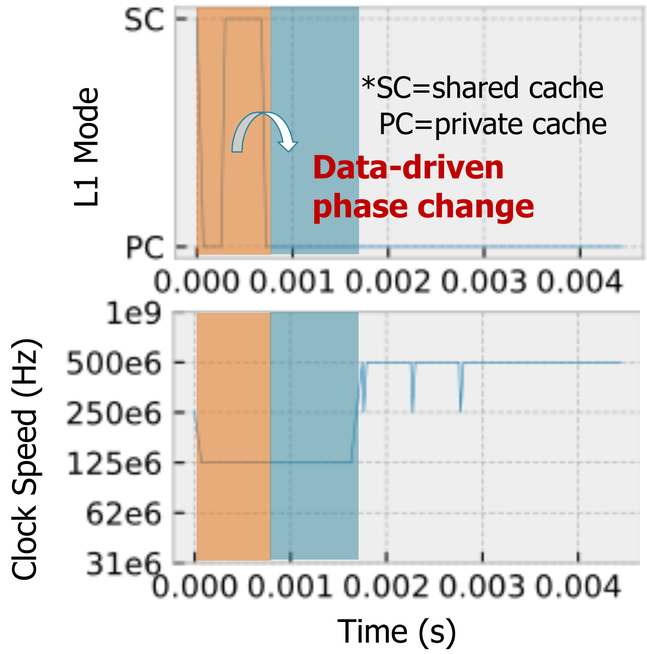



Model learns mapping f : X → Y

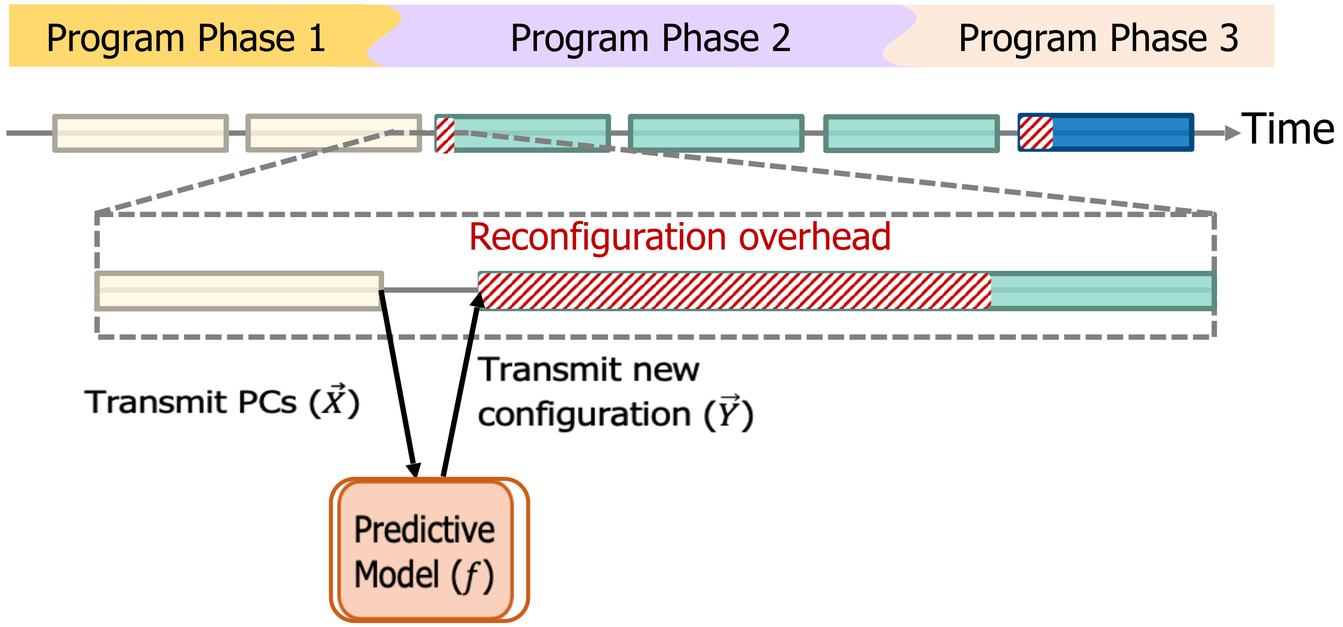
X: the space of performance counters

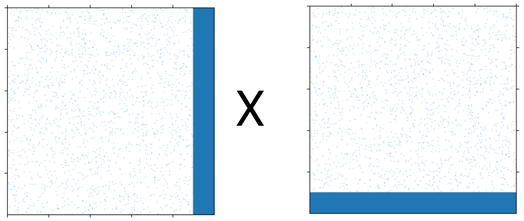
Y: best micro-architectural configurations

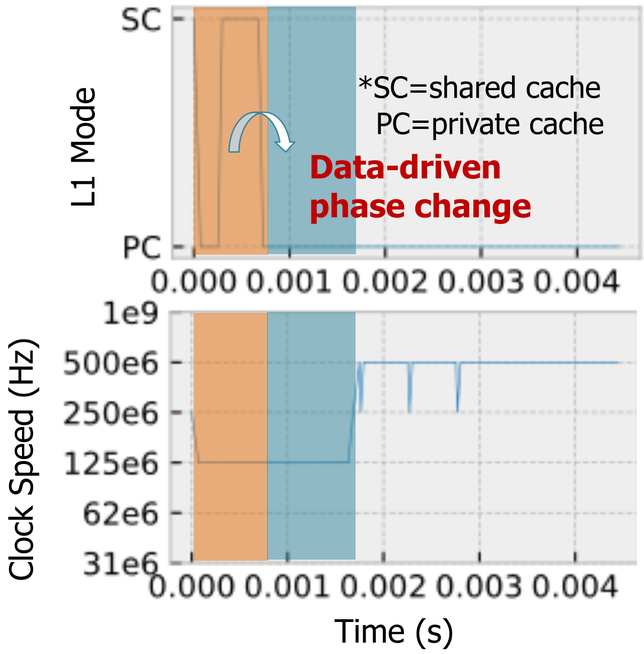


# Sparse Adapt: Hardware Reconfiguration




Model learns mapping f : X → Y

X: the space of performance counters

Y: best micro-architectural configurations

***2.9x energy-efficiency***

Matching Hardware to Software - hardware defined software (HDS)

Other

- Neural Architecture Search as Program Transformation Exploration

- Software Defined  Hardware (SDH)

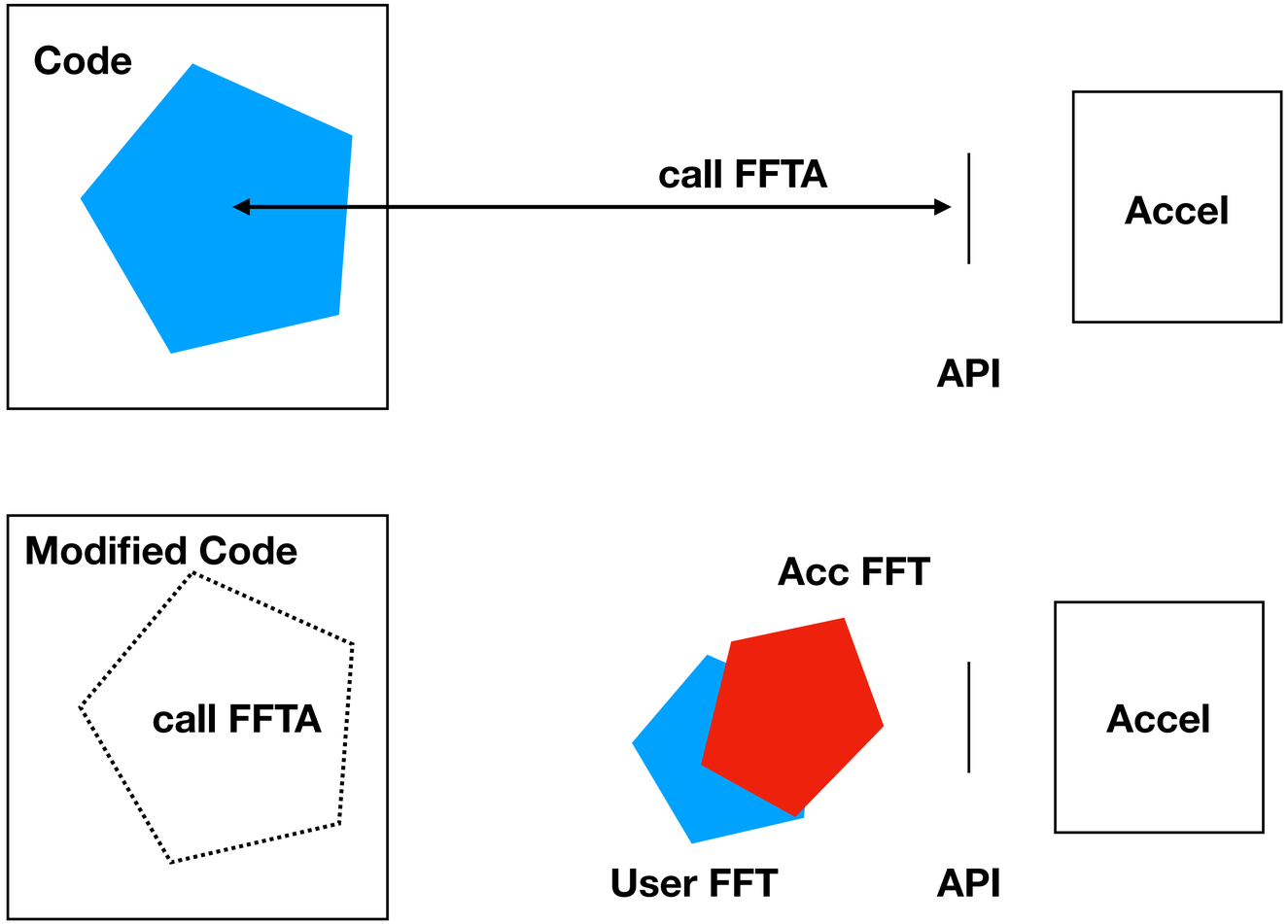
**Beyond Simple Acceleration**

# Big-step Acceleration: FFT

Matching complex accelerators is challenging

- Behaviour unlikely to match user code

- FFT  acceleration a good example



# Big-step Acceleration: FFT

Matching complex accelerators is challenging

- Behaviour unlikely to match user code

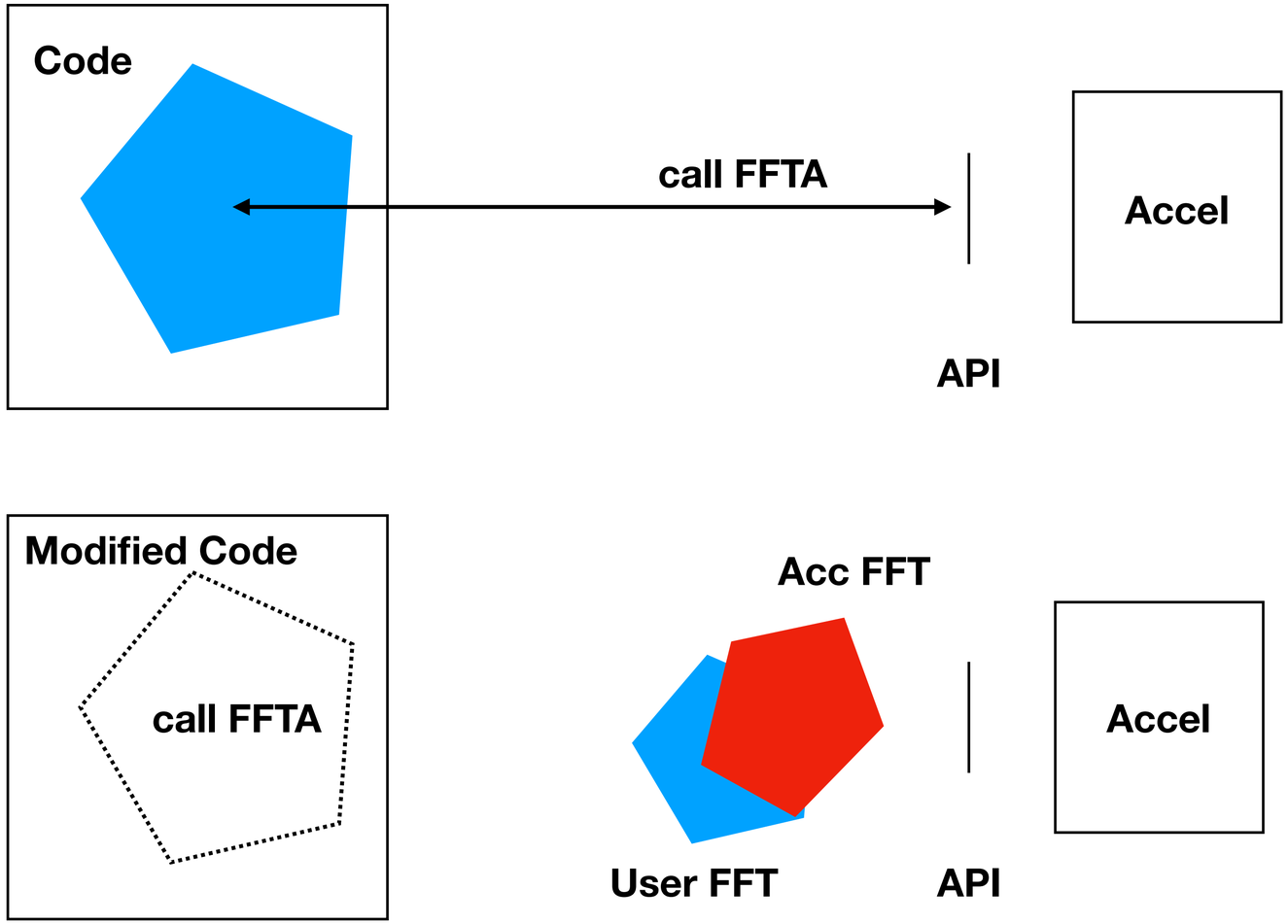
- FFT  acceleration a good example

Need to bridge gap

- Applied to GitHub code

- Significant speedups

See Jackson Woodruff presentation



# Beyond fixed function: Neural Compilation

Significant accelerators will be programmable

- Likely to have specialised prog lang

Can we learn how to translate existing code into new lang?

# Beyond fixed function: Neural Compilation

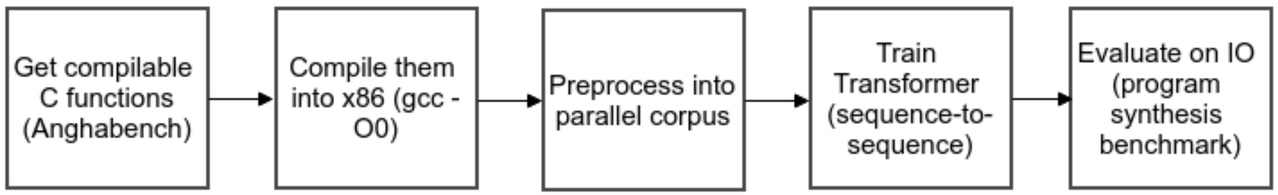
Significant accelerators will be programmable

- Likely to have specialised prog lang

Can we learn how to translate existing code into new lang?

Proof of concept: Learn C->x86

- Used Transformer model



# Beyond fixed function: Neural Compilation

Significant accelerators will be programmable

- Likely to have specialised prog lang

Can we learn how to translate existing code into new lang?

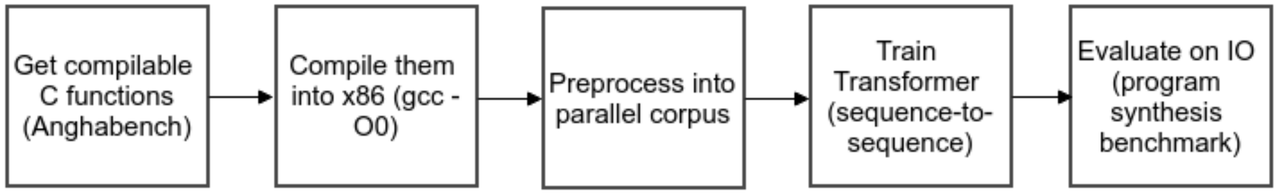
Proof of concept: Learn C->x86

- Used Transformer model

Surprising results!

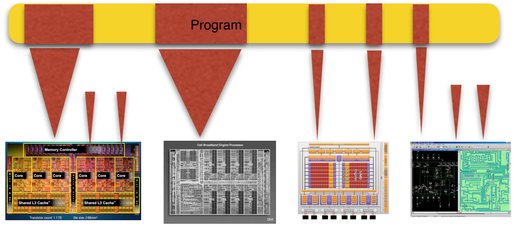
```
int triangle_sum(int n) {
  int r = 0;
  for (int i = 1; i < n;
       ++i) {
    for (int m = 1; m < i;
         ++m) {
      r += m;
    }
  }
  return r;
}
```

```
triangle_sum:
 .LFB0:
 .cfi_startproc
 pushq %rbp
 .cfi_def_cfa_offset 16
 .cfi_offset 6, -16
 movq %rsp, %rbp
 .cfi_def_cfa_register 6
 movl %edi, -20(%rbp)
 movl $0, -12(%rbp)
 movl $1, -8(%rbp)
 jmp .L2
 .L5:
 movl $1, -4(%rbp)
 jmp .L3
 .L4:
 movl -4(%rbp), %eax
 addl %eax, -12(%rbp)
 addl $1, -4(%rbp)
 .L3:
 movl -4(%rbp), %eax
 cmpl -8(%rbp), %eax
 jl .L4
 addl $1, -8(%rbp)
 .L2:
 movl -8(%rbp), %eax
 cmpl -20(%rbp), %eax
 jl .L5
 movl -12(%rbp), %eax
 popq %rbp
 .cfi_def_cfa 7, 8
 ret
 .cfi_endproc
```

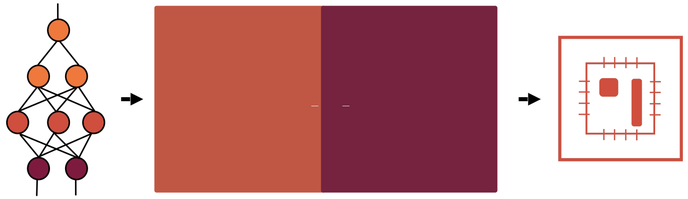


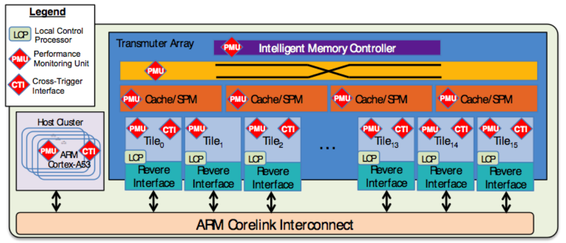
# Conclusion

Matching Hardware to Software
- enables hardware innovation




Expressing  NAS as program transformation
- generates new designs






Software can help hardware improve performance
- prefetching and reconfiguration

Going beyond simple acceleration requires new approaches