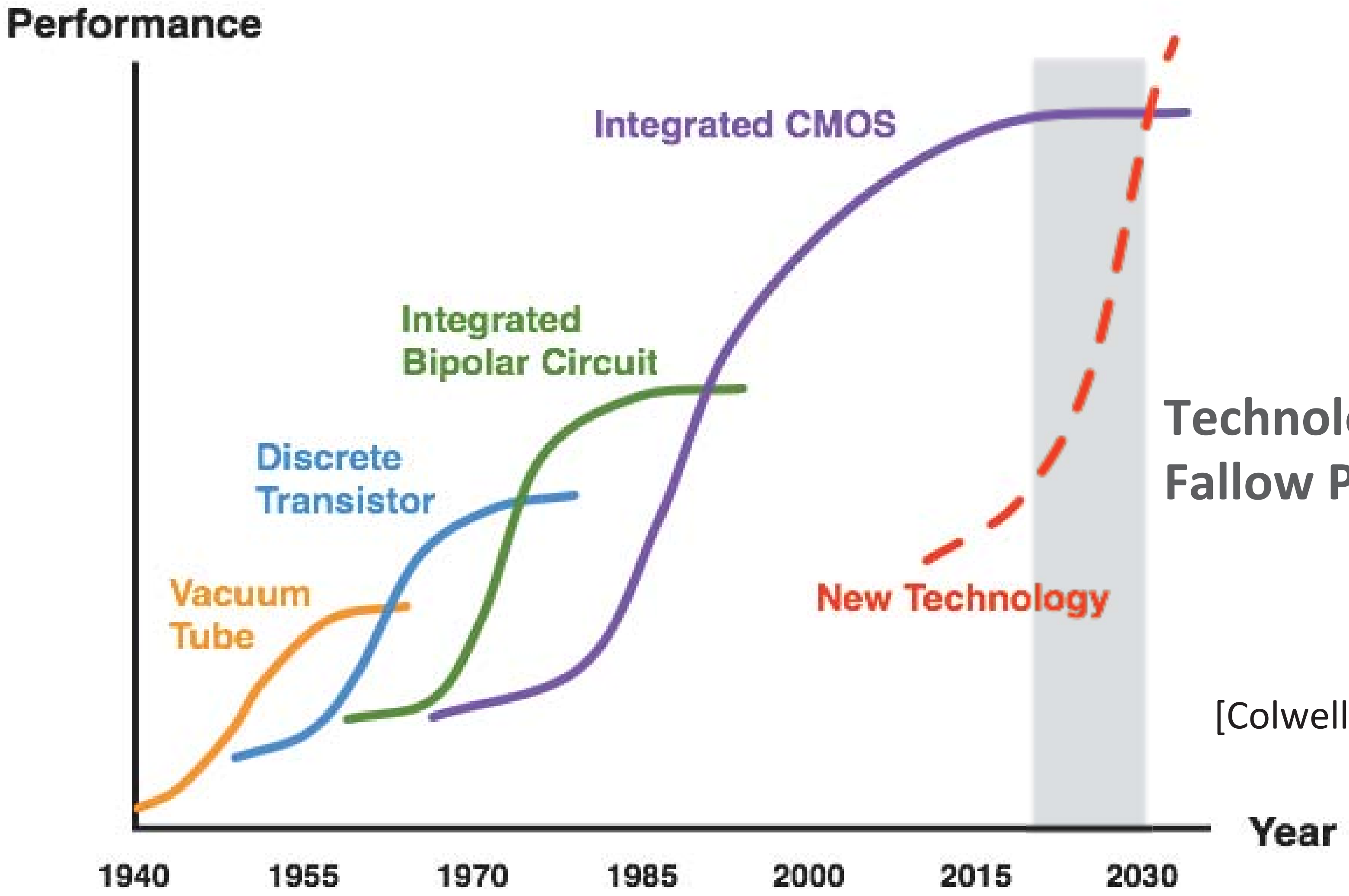
Accelerator Architectures:
High-Level Modeling of Specialization

David Brooks

School of Engineering and Applied Sciences
Harvard University

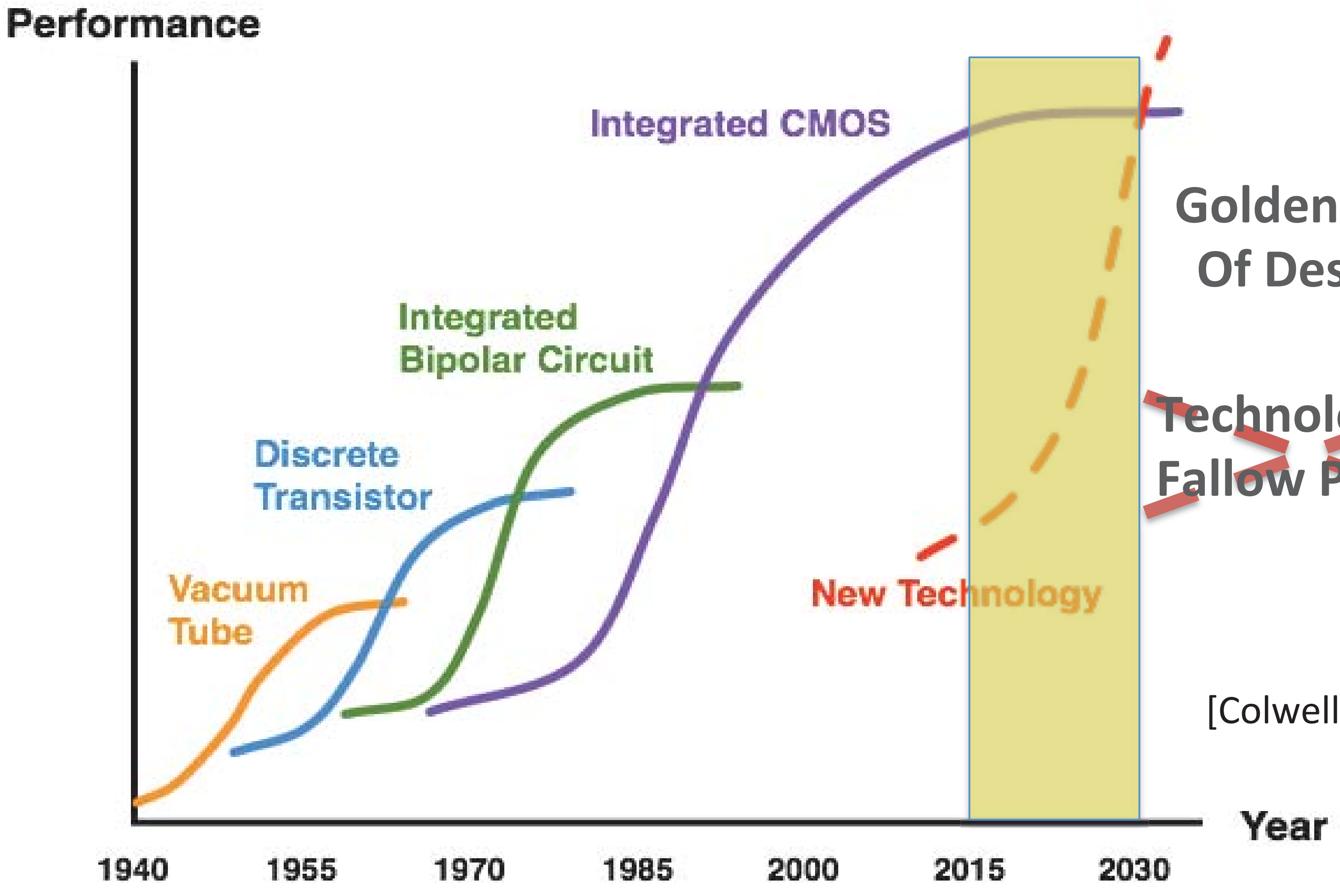
CMOS scaling is running out



Technological Fallow Period

[Colwell 2012]

…and it’s about time.

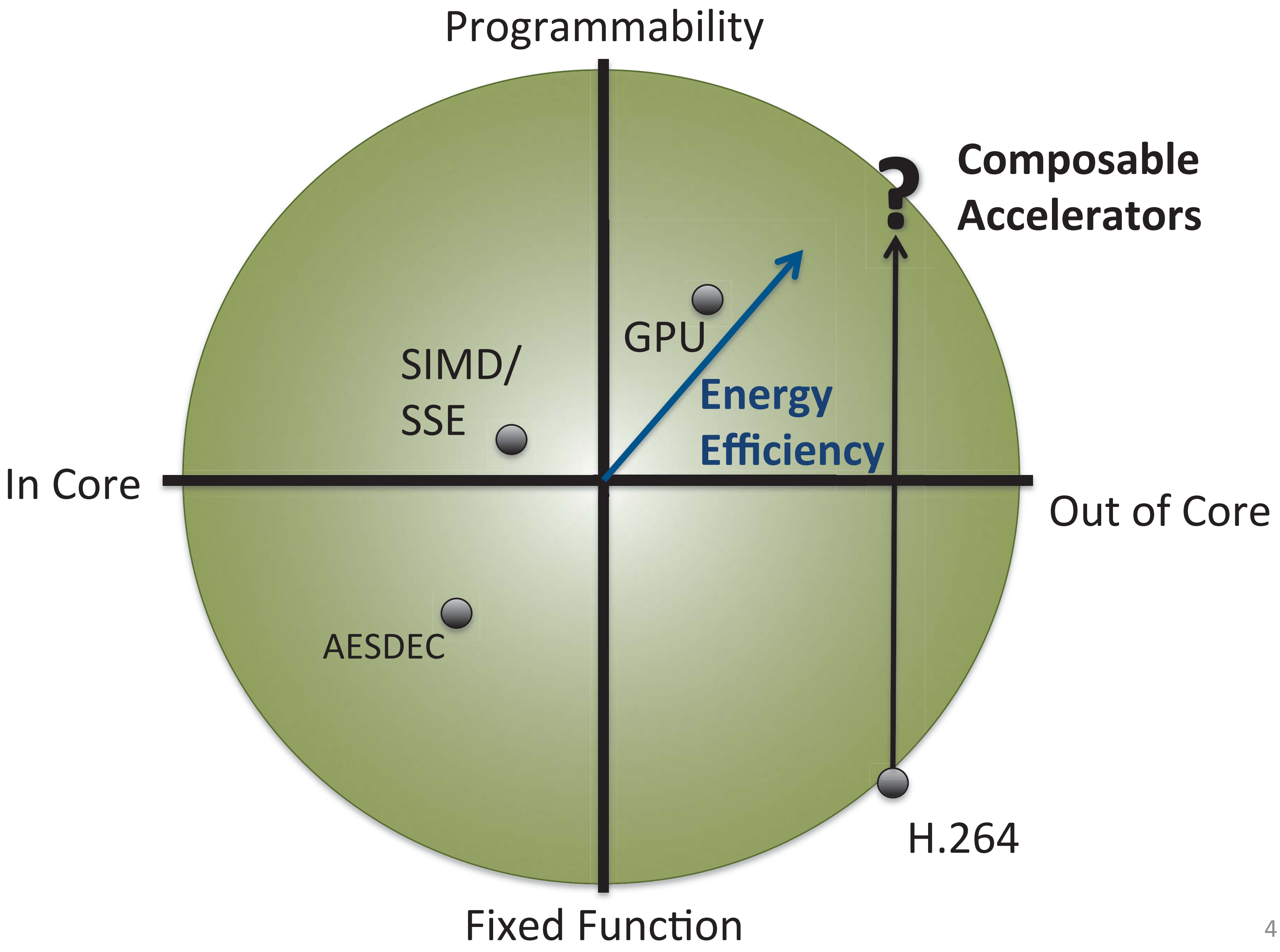


Golden Age Of Design

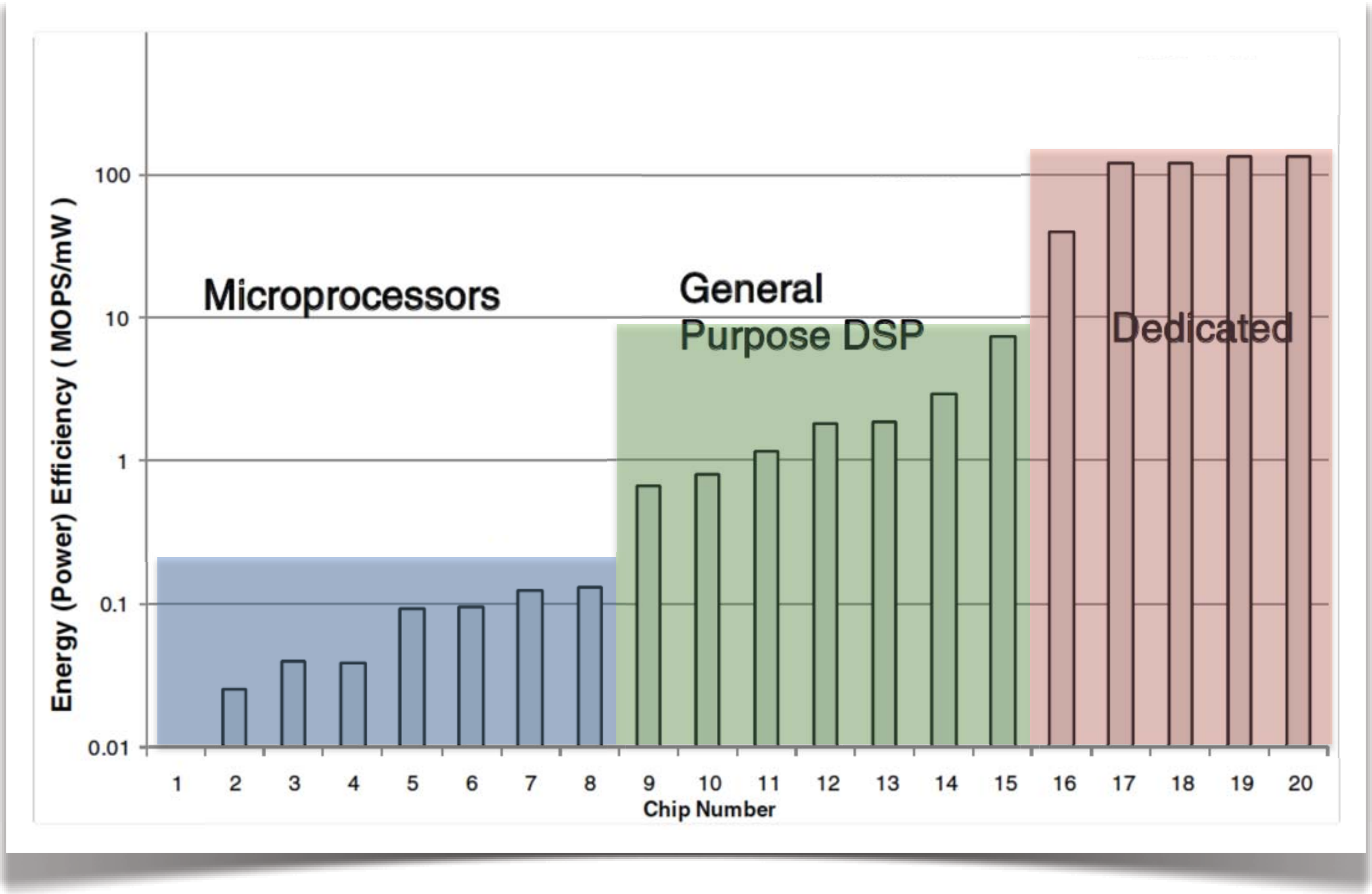
~~Technological Fallow Period~~

[Colwell 2012]

Look beyond homogeneous parallelism



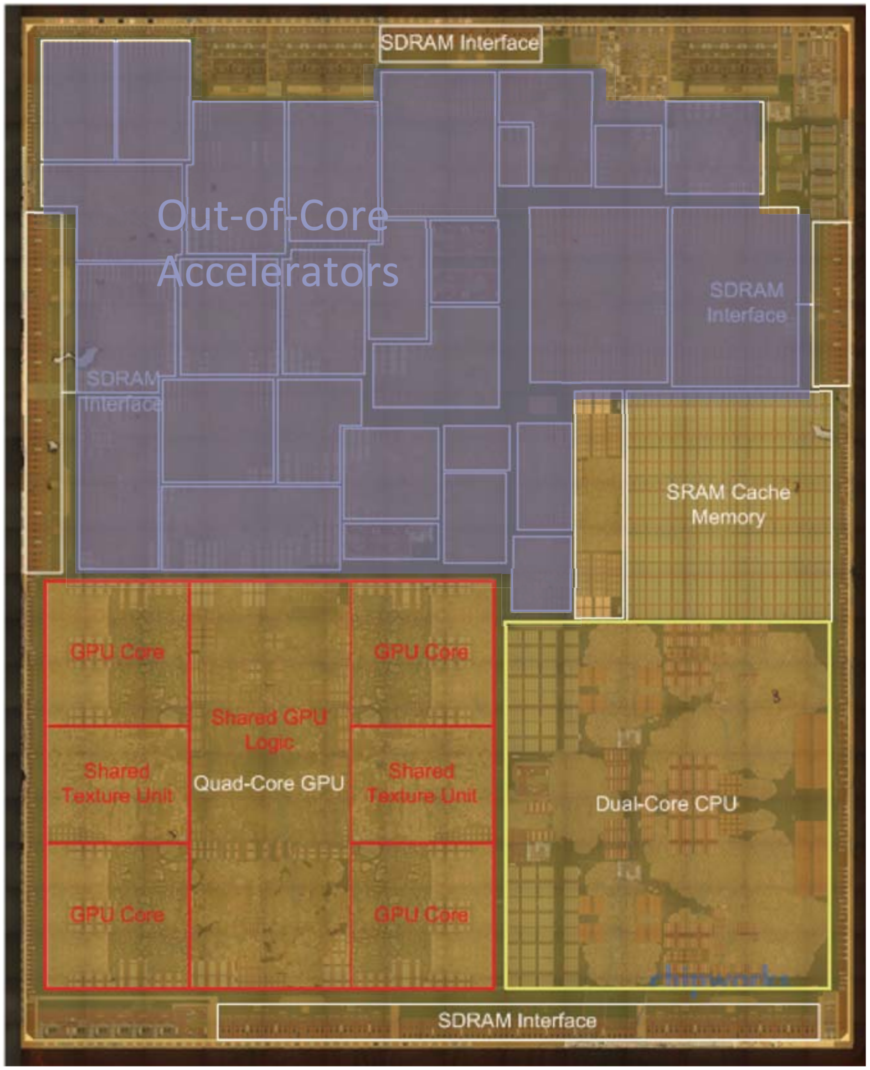
Potential for Specialized Architectures

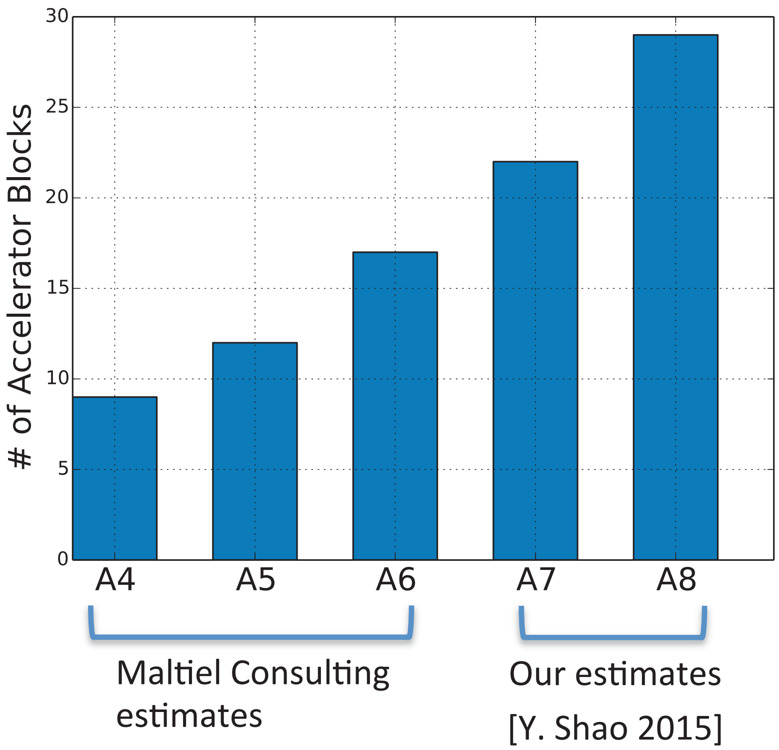


16	Encryption
17	Hearing Aid
18	FIR for disk read
19	MPEG Encoder
20	802.11 Baseband

[Brodersen and Meng, 2002]

Cores, GPUs, and Accelerators: Apple A8 SoC



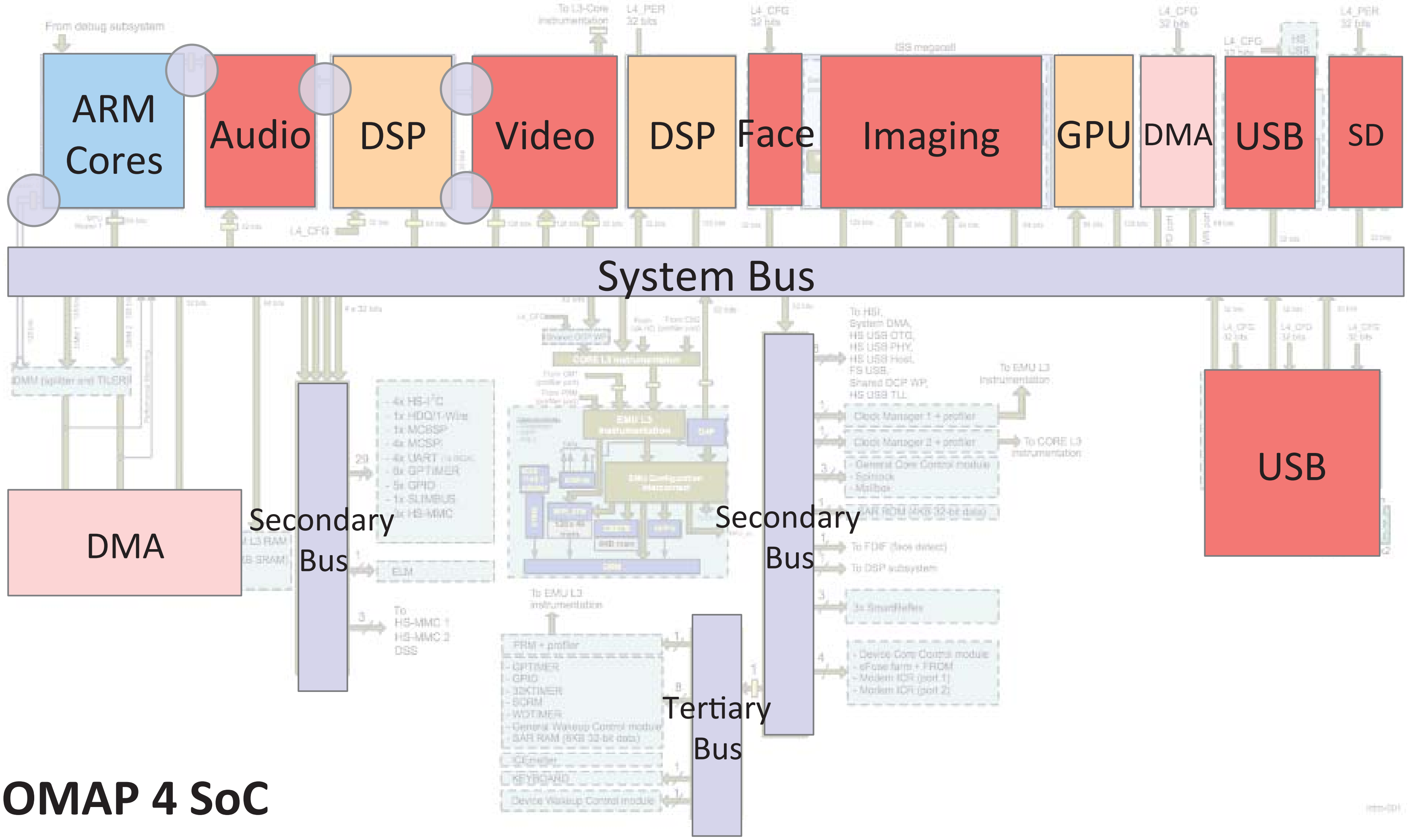


Maltiel Consulting estimates

Our estimates [Y. Shao 2015]

[www.anandtech.com/show/8562/chipworks-a8]

Today's SoC

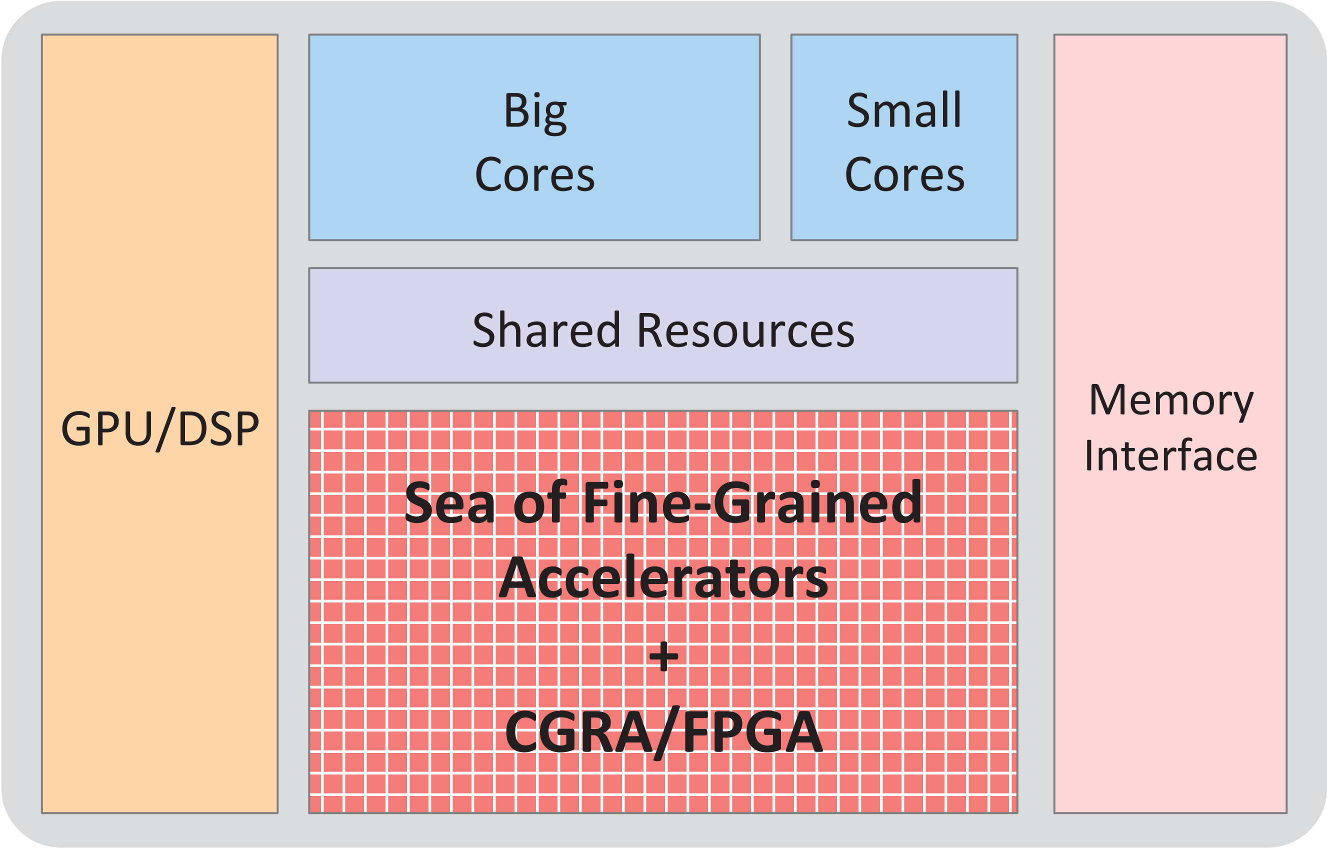


OMAP 4 SoC

Challenges in Accelerators

- Flexibility
  - Fixed-function accelerators are only designed for the target applications.
- Programmability
  - Today's accelerators are explicitly managed by programmers.
- Design Cost
  - Accelerator (and RTL) implementation is inherently tedious and time-consuming.

Future accelerator-centric architectures



GPU/DSP

Big Cores

Small Cores

Shared Resources

Sea of Fine-Grained Accelerators + CGRA/FPGA

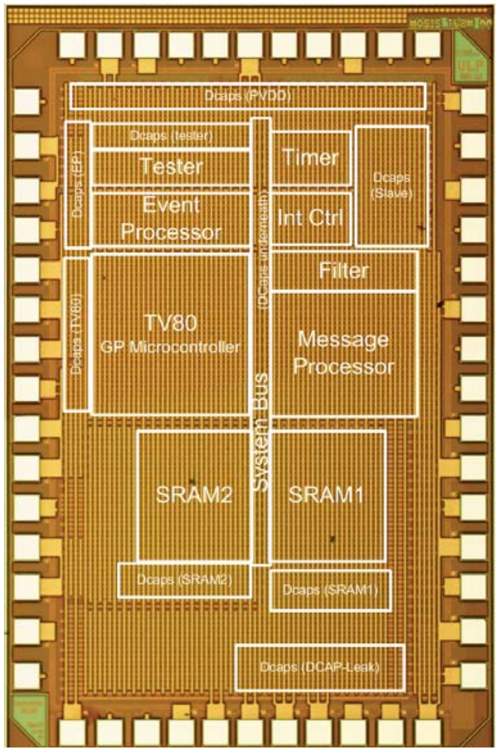
Memory Interface

How to decompose an application to accelerators?
How to rapidly design many accelerators?
How to design and manage the shared resources?

⬆ Flexibility
⬇ Design Cost
⬆ Programmability

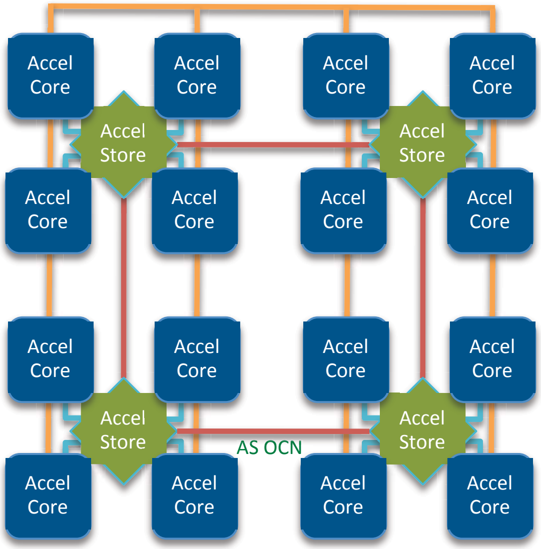
Some highlights (and pain points) of our research in accelerator architectures

Event-Driven Architectures
For Wireless Sensor Nodes



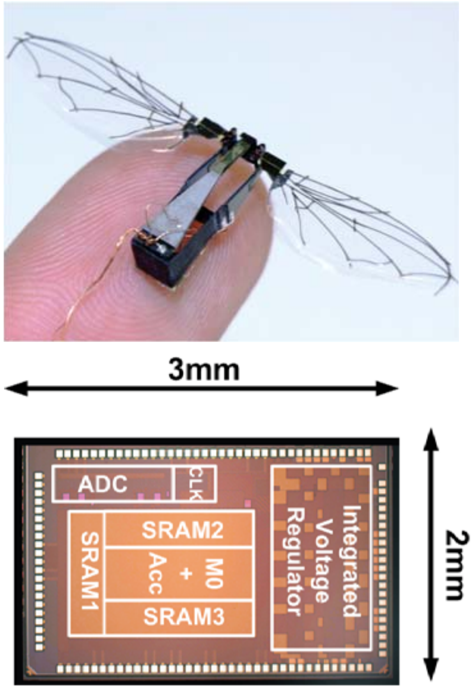
Hempstead, ISCA'05

Accelerator Memory
Systems Design:
"Accelerator Store"



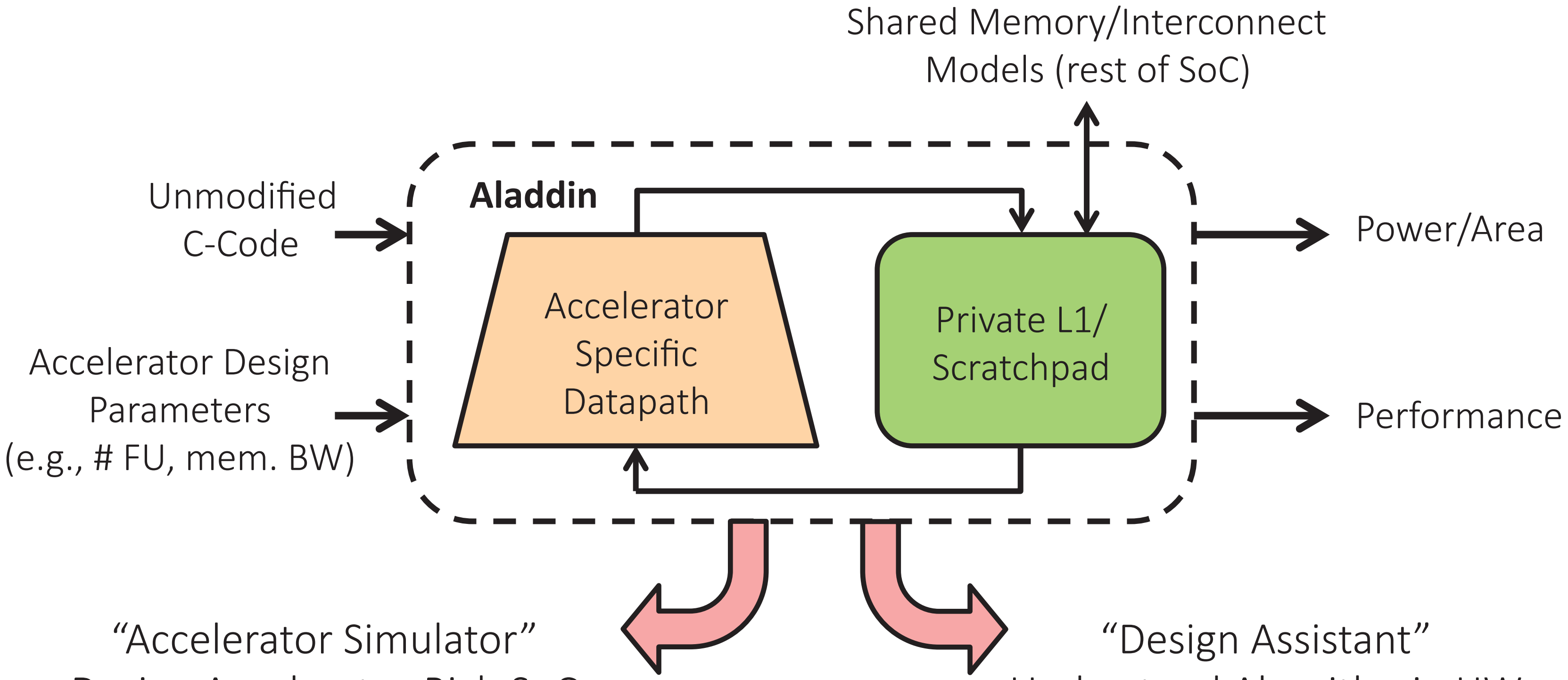
Lyons, CAL'10

Robobee "Brain"
System-on-Chip



Zhang, CICC'13, VLSI'15

Aladdin: A pre-RTL accelerator model



Shared Memory/Interconnect Models (rest of SoC)

Unmodified C-Code

Accelerator Design Parameters (e.g., # FU, mem. BW)

Aladdin

Accelerator Specific Datapath

Private L1/ Scratchpad

Power/Area

Performance

"Accelerator Simulator"
Design Accelerator-Rich SoC
Fabrics and Memory Systems

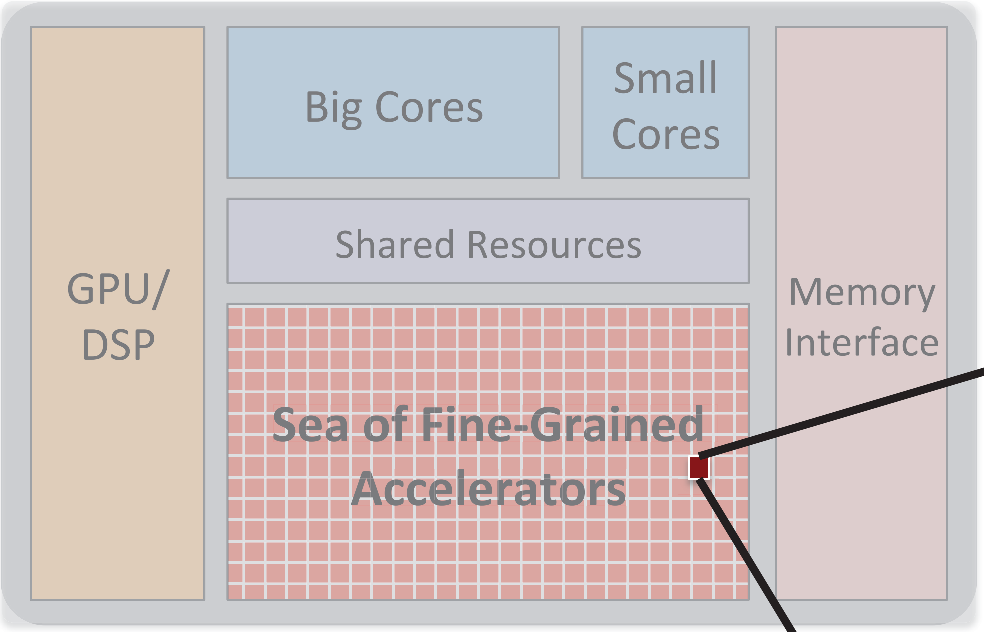
⬆ **Flexibility**
⬆ **Programmability**

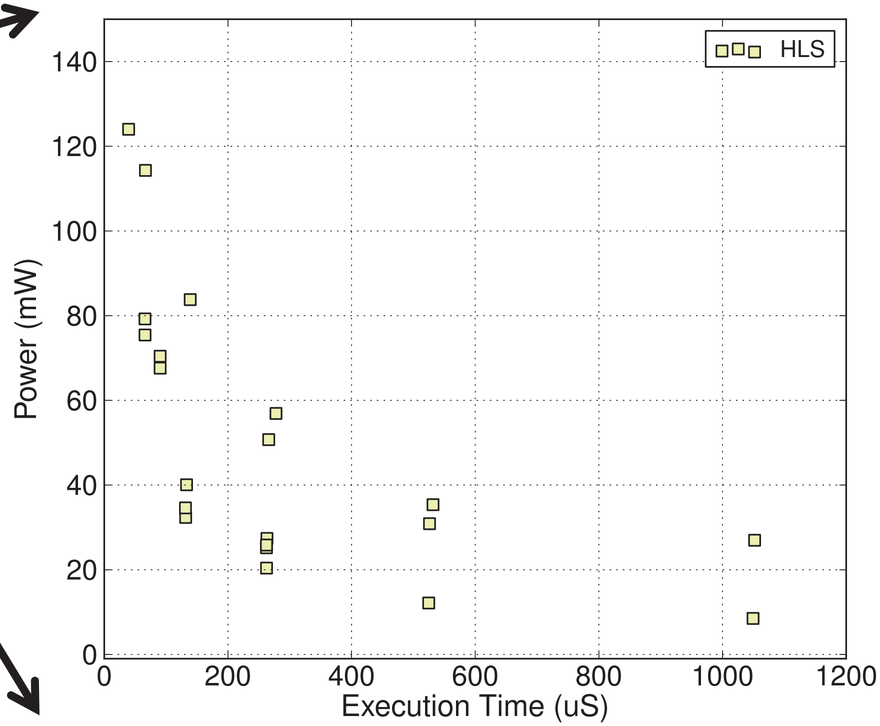
"Design Assistant"
Understand Algorithmic-HW
Design Space before RTL

⬇ **Design Cost**

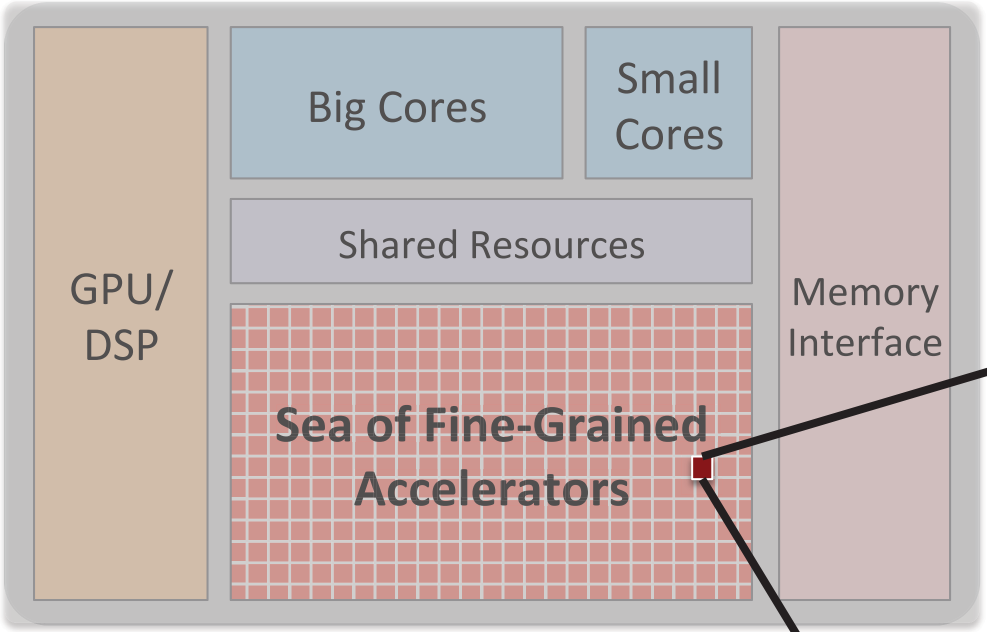
[Y. Shao et al., ISCA 2014]

Accelerator-Centric Architecture





Accelerator-Centric Architecture



GPU/ DSP

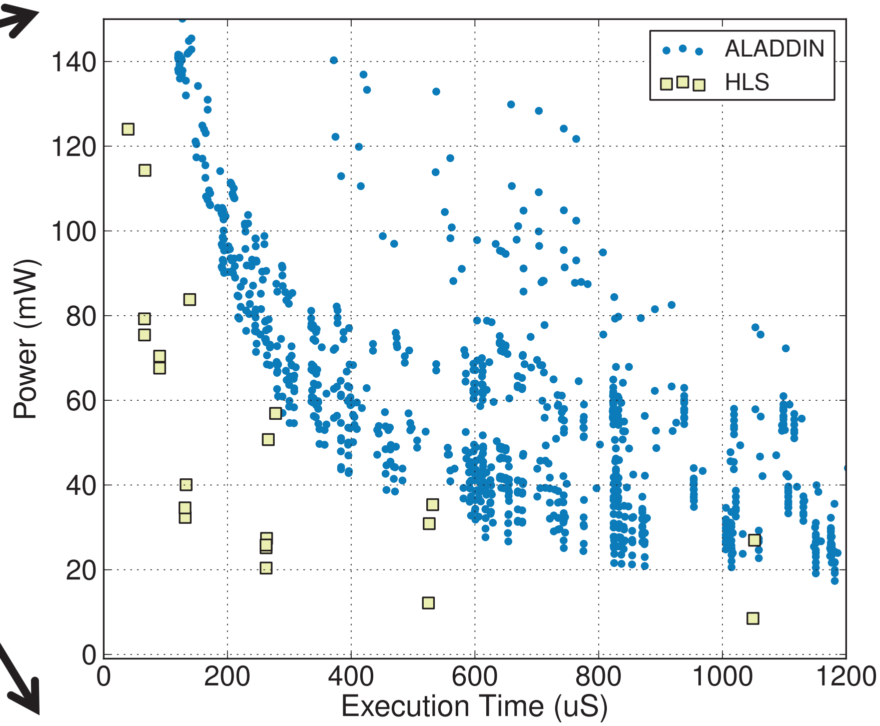
Big Cores

Small Cores

Shared Resources

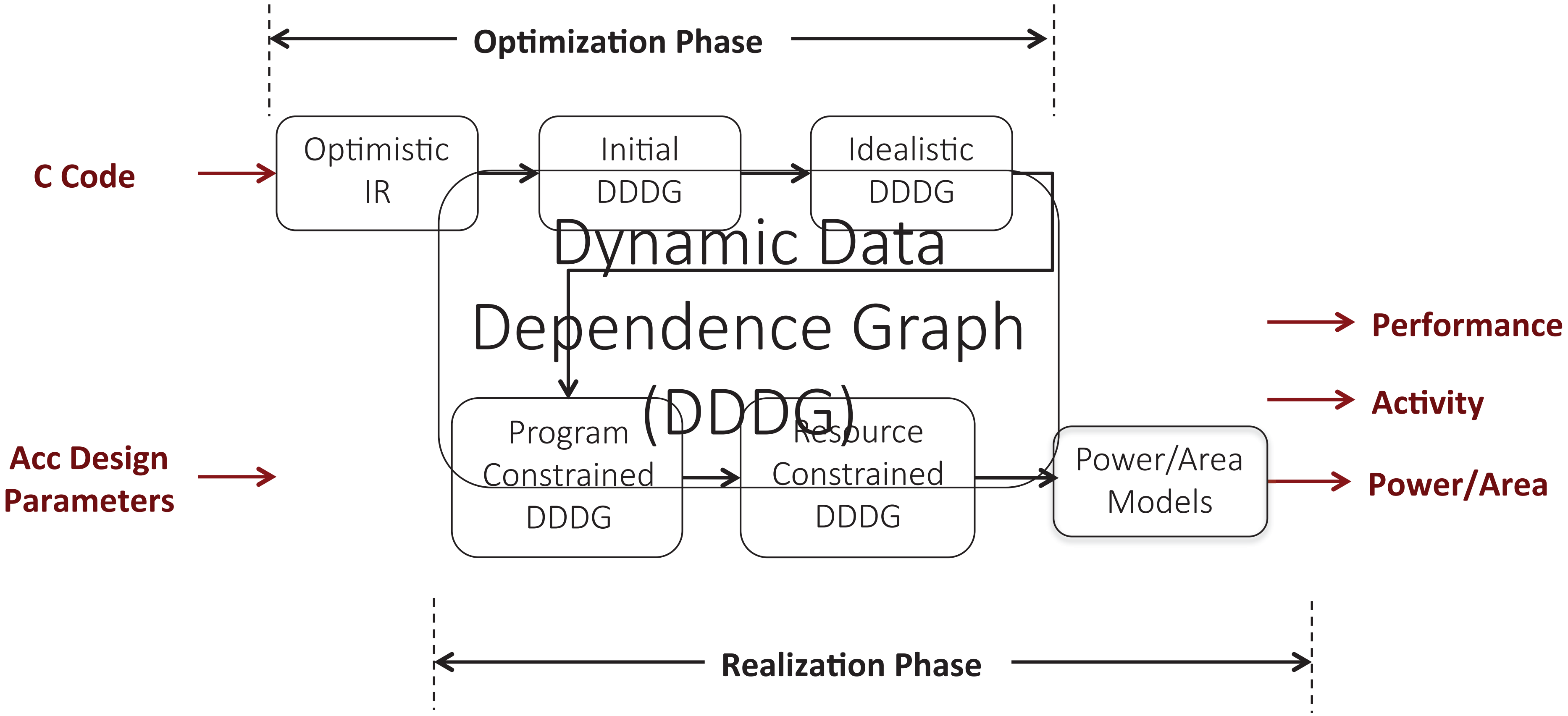
Memory Interface

Sea of Fine-Grained Accelerators

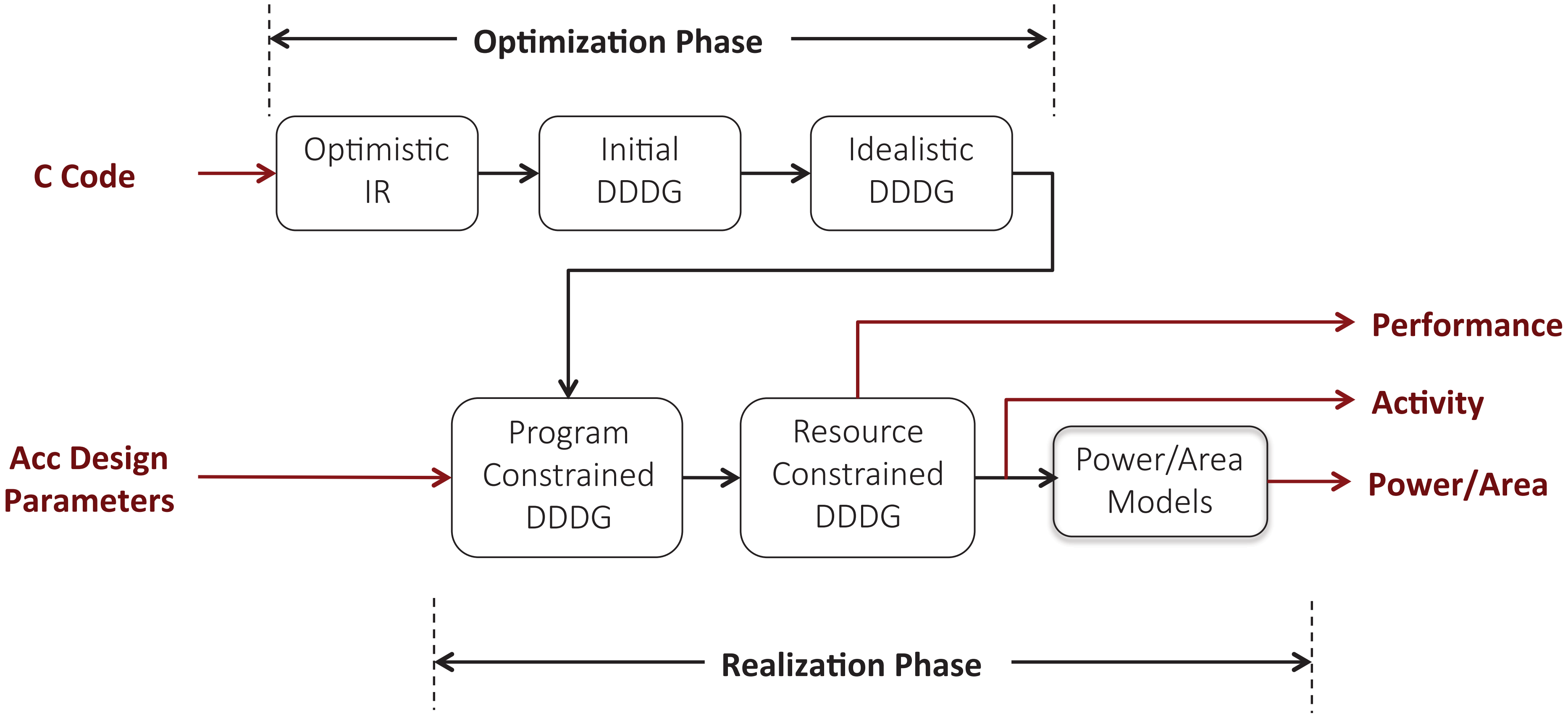


Aladdin can **rapidly** evaluate **large** design space of accelerator-centric architectures.

Aladdin Overview



Aladdin Overview



From C to Design Space

C Code:

```
for(i=0; i<N; ++i)
  c[i] = a[i] + b[i];
```

IR Dynamic Trace

C Code:

```
for(i=0; i<N; ++i)
  c[i] = a[i] + b[i];
```

```
0.  r0=0  //i = 0
1.   r4=load (r0 + r1) //load a[i]
2.   r5=load (r0 + r2) //load b[i]
3.   r6=r4 + r5
4.   store(r0 + r3, r6) //store c[i]
5.   r0=r0 + 1  //++i
6.   r4=load(r0 + r1) //load a[i]
7.   r5=load(r0 + r2) //load b[i]
8.   r6=r4 + r5
9.   store(r0 + r3, r6) //store c[i]
10.  r0 = r0 + 1  //++i
…
```

From C to Design Space

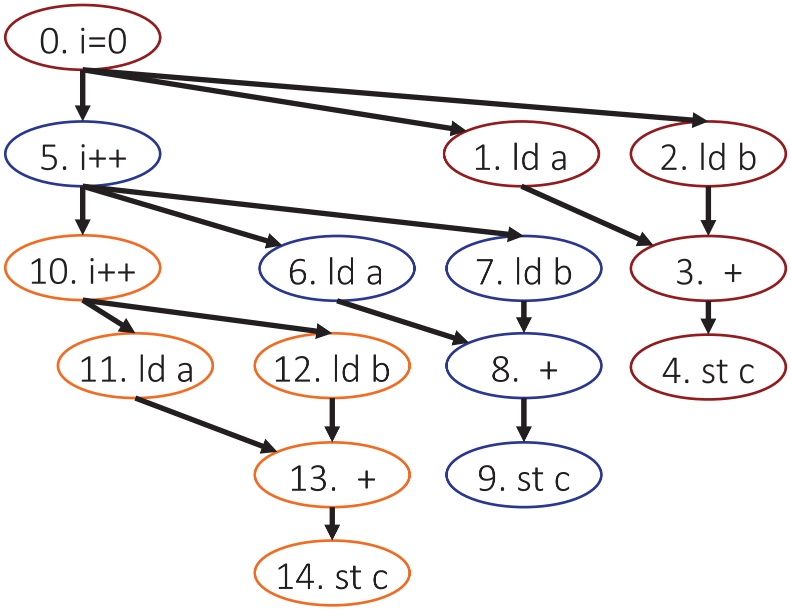
Initial DDDG

C Code:

```
for(i=0; i<N; ++i)
  c[i] = a[i] + b[i];
```

IR Trace:

```
0.  r0=0  //i = 0
1.  r4=load (r0 + r1) //load a[i]
2.  r5=load (r0 + r2) //load b[i]
3.  r6=r4 + r5
4.  store(r0 + r3, r6) //store c[i]
5.  r0=r0 + 1  //++i
6.  r4=load(r0 + r1) //load a[i]
7.  r5=load(r0 + r2) //load b[i]
8.  r6=r4 + r5
9.  store(r0 + r3, r6) //store c[i]
10.r0 = r0 + 1  //++i
…
```

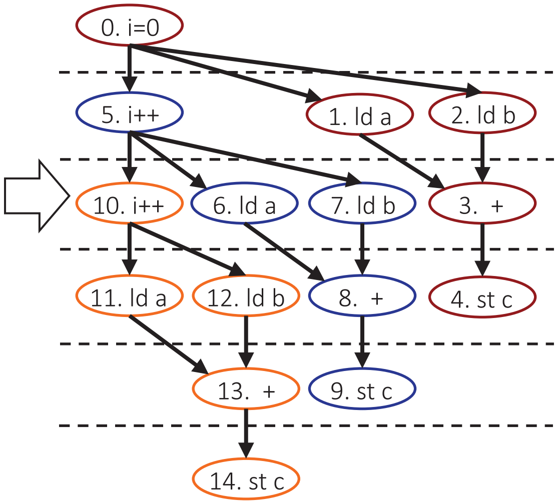


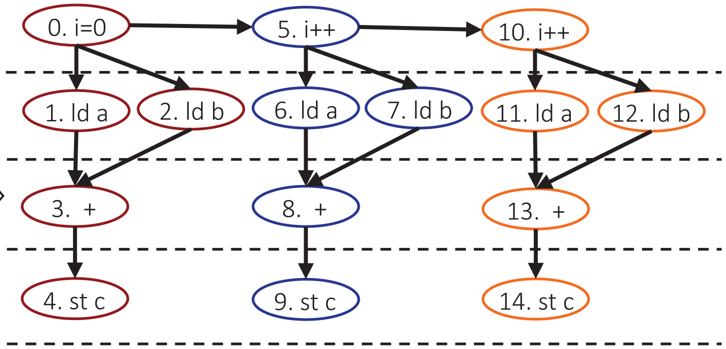
From C to Design Space

Idealistic DDDG

C Code:
```
for(i=0; i<N; ++i)
  c[i] = a[i] + b[i];
```

IR Trace:
```
0.  r0=0  //i = 0
1.  r4=load (r0 + r1) //load a[i]
2.  r5=load (r0 + r2) //load b[i]
3.  r6=r4 + r5
4.  store(r0 + r3, r6) //store c[i]
5.  r0=r0 + 1  //++i
6.  r4=load(r0 + r1) //load a[i]
7.  r5=load(r0 + r2) //load b[i]
8.  r6=r4 + r5
9.  store(r0 + r3, r6) //store c[i]
10.r0 = r0 + 1  //++i
...
```





From C to Design Space

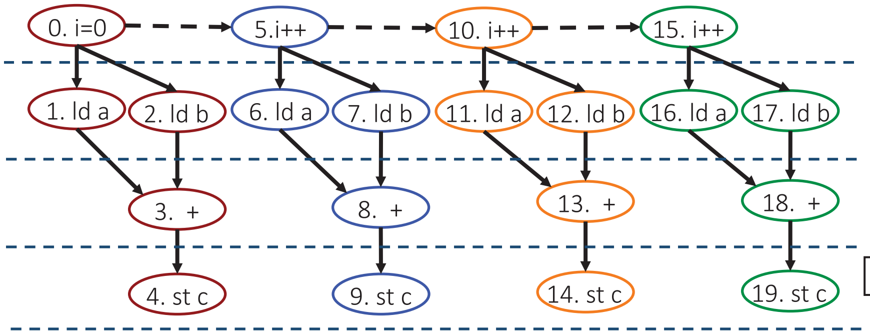
Optimization Phase: C->IR->DDDG

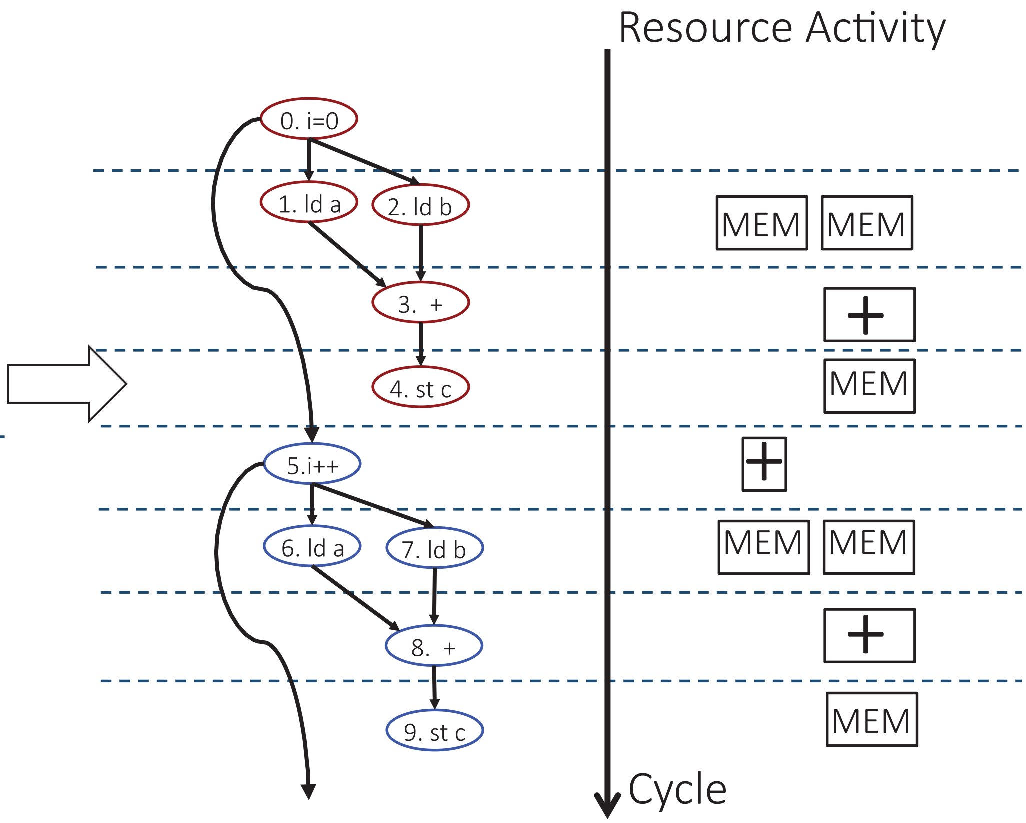
- Include application-specific customization strategies.
- Node-Level:
  - Bit-width Analysis
  - Strength Reduction
  - Tree-height Reduction
- Loop-Level:
  - Remove dependences between loop index variables
- Memory Optimization:
  - Memory-to-Register Conversion
  - Store-Load Forwarding
  - Store Buffer
- Extensible
  - e.g. Model CAM accelerator by matching nodes in DDDG

From C to Design Space

One Design

Idealistic DDDG





Acc Design Parameters:

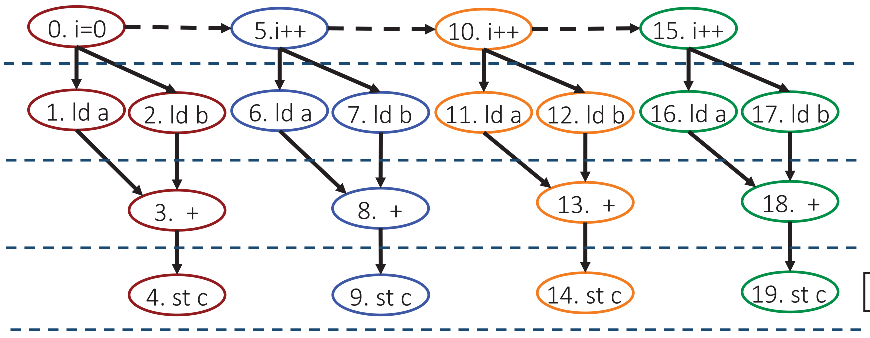
✓ Memory BW <= **2**

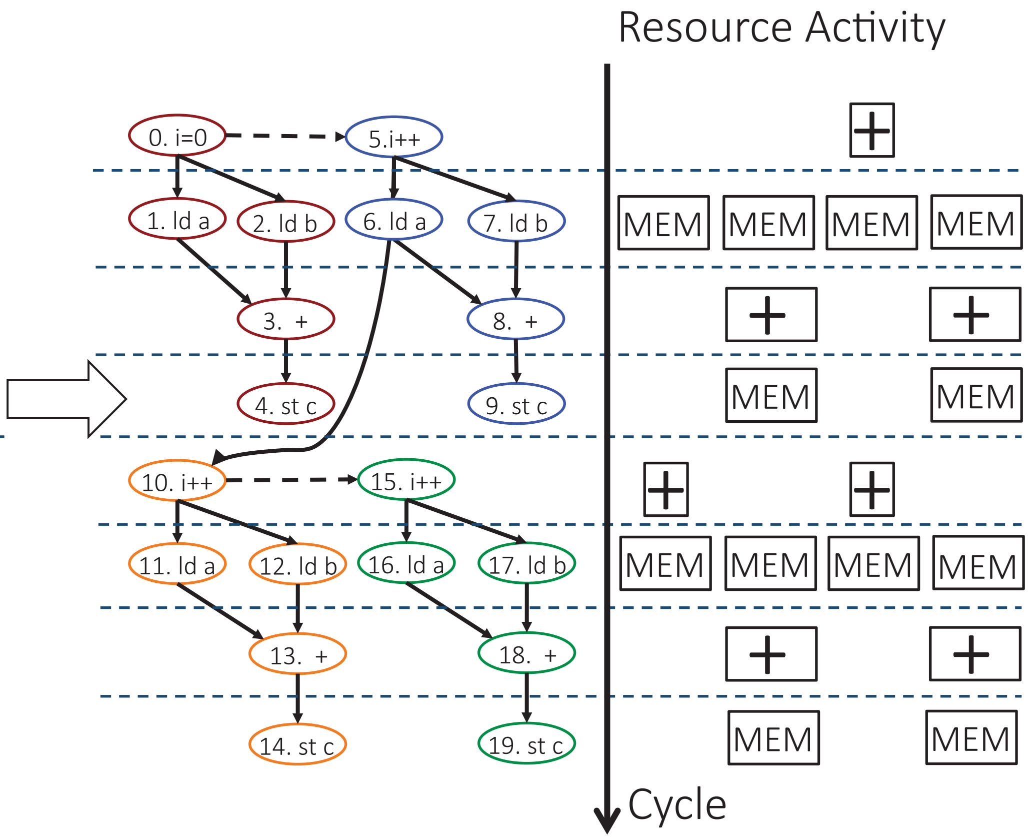
✓ **1** Adder

From C to Design Space

Another Design

Idealistic DDDG





Acc Design Parameters:

- ✓ Memory BW <= **4**
- ✓ **2** Adders

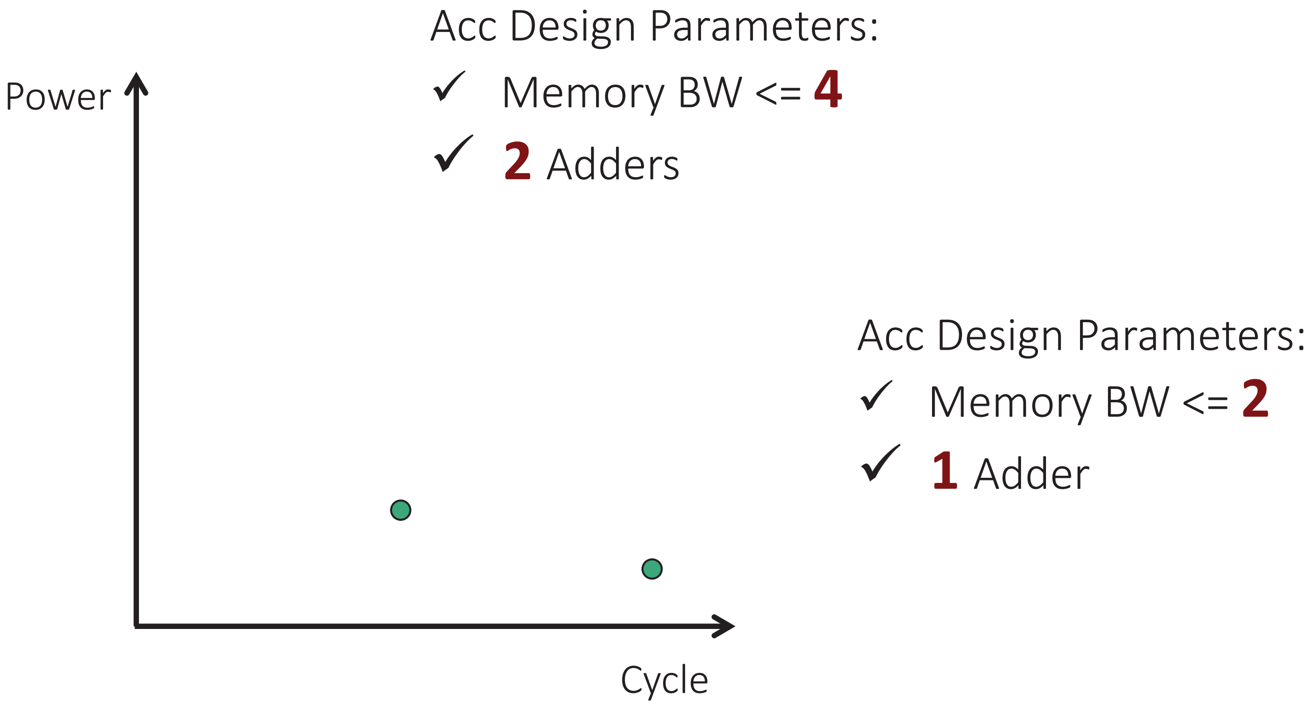
From C to Design Space

Realization Phase: DDDG->Estimates

- Constrain the DDDG with program and user-defined resource constraints
- Program Constraints
  - Control Dependence
  - Memory Ambiguation
- Resource Constraints
  - Loop-level Parallelism
  - Loop Pipelining
  - Memory Ports
  - # of FUs (e.g., adders, multipliers)

From C to Design Space

Power-Performance per Design



Acc Design Parameters:

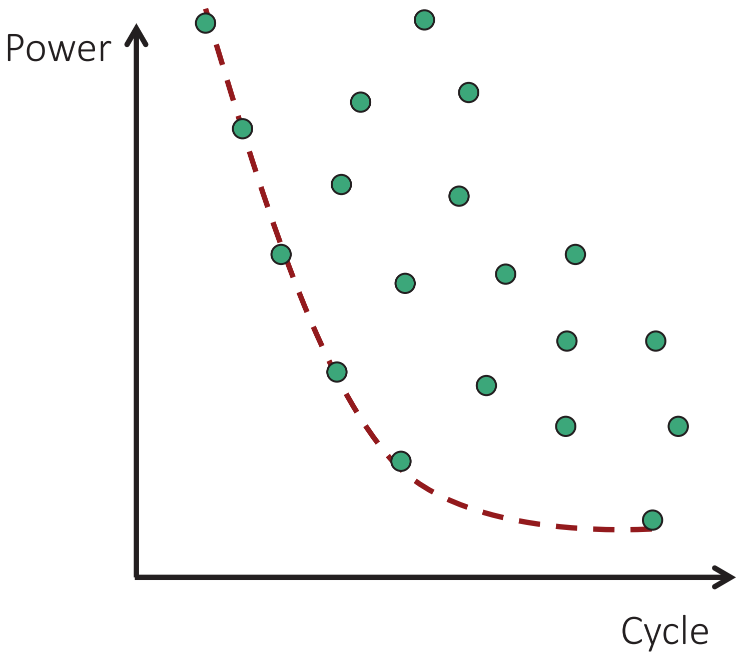
- ✓ Memory BW <= **4**
- ✓ **2** Adders

Acc Design Parameters:

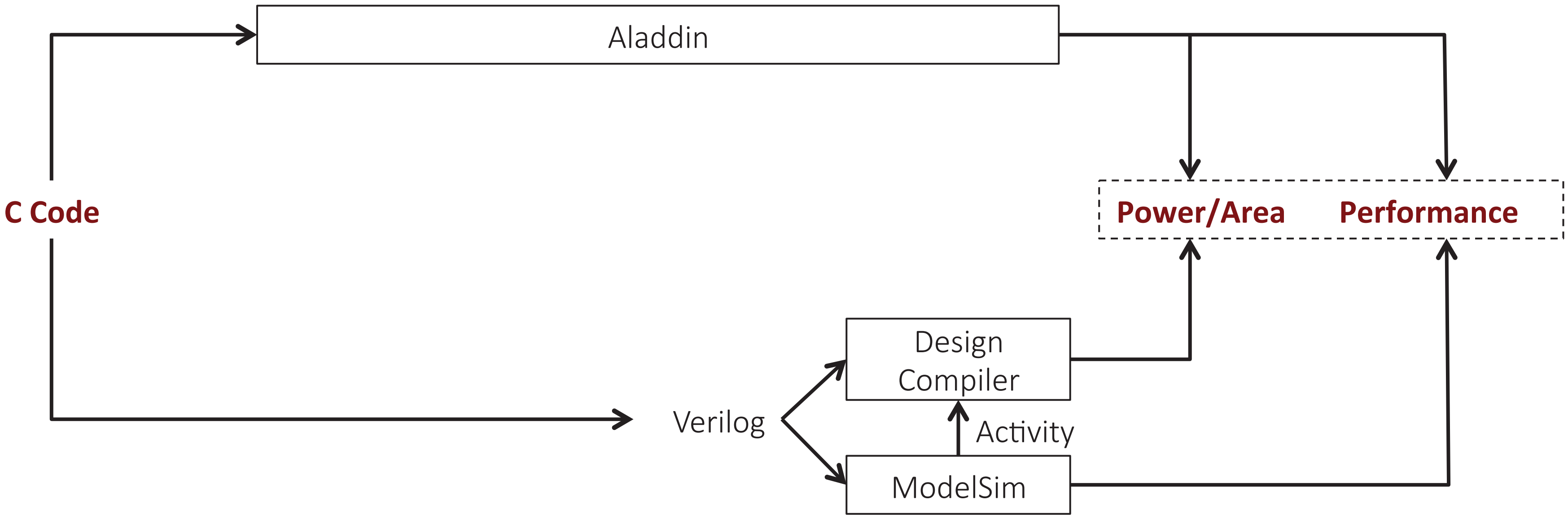
- ✓ Memory BW <= **2**
- ✓ **1** Adder

From C to Design Space

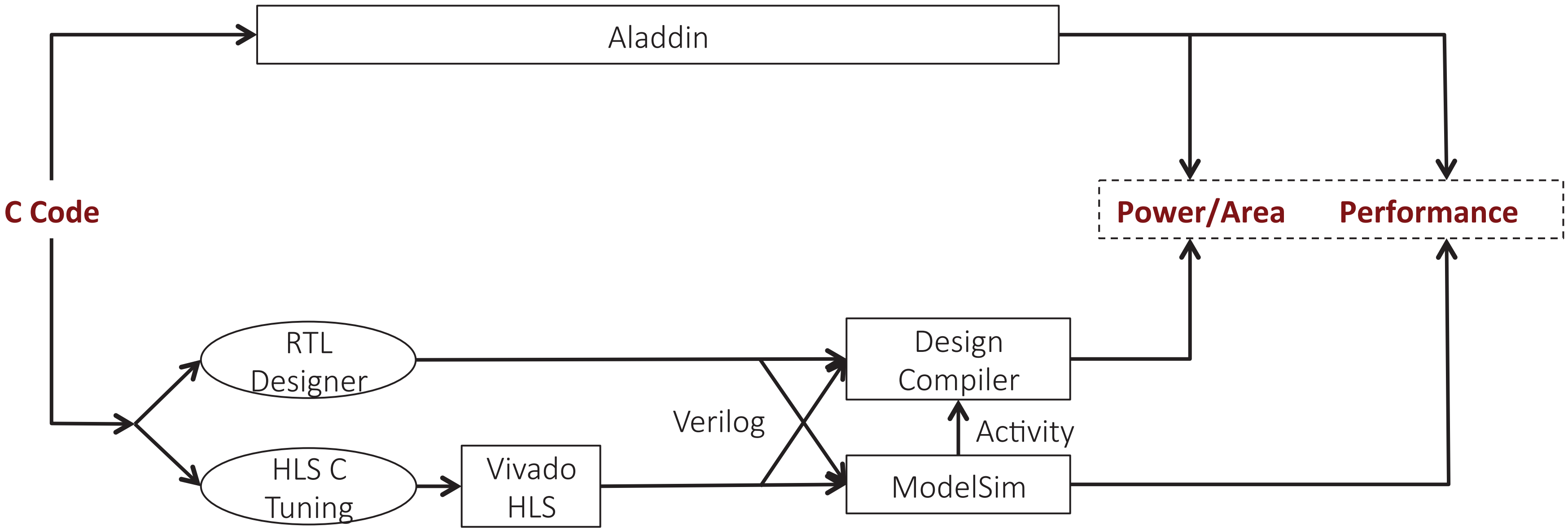
Design Space of an Algorithm



Aladdin Validation



Aladdin Validation



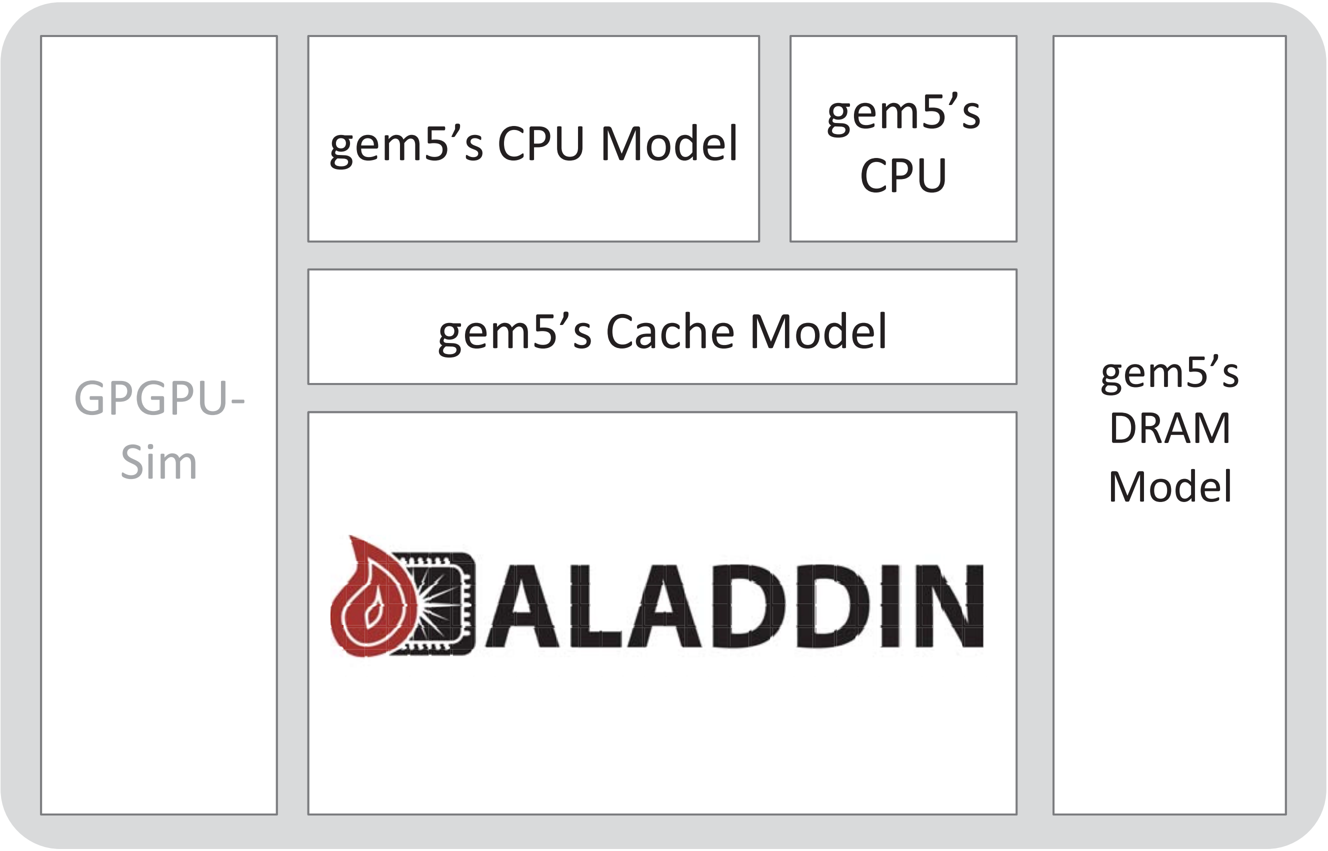
Aladdin Take-Away

- Compared to HLS and hand-written RTL for SHOC benchmarks and custom accelerator designs

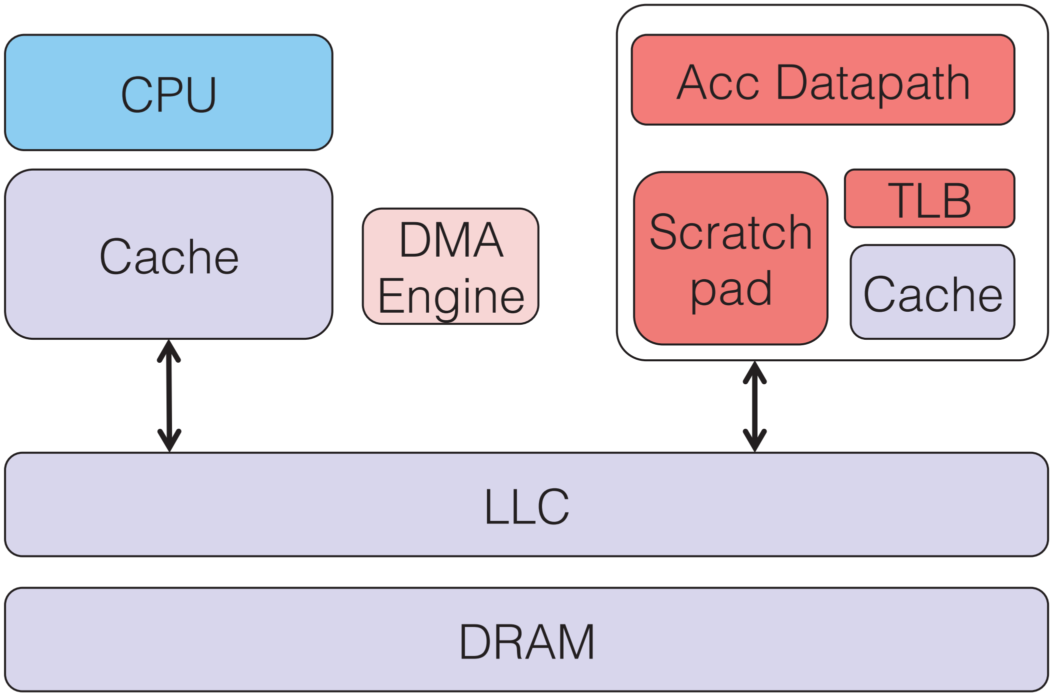
Cycle Counts	**within 2%**
Power	**within 5%**
Area	**within 7%**

- Large design space exploration (DSE) in minutes instead of hours/days with unmodified C/C++ algorithm description
- Limitations
  - Dynamic approach ➔ Aladdin depends on realistic workload inputs
  - Algorithm dependent ➔Aladdin *enables* DSE/algorithm exploration

Aladdin/gem5 integration



gem5-Aladdin Integration

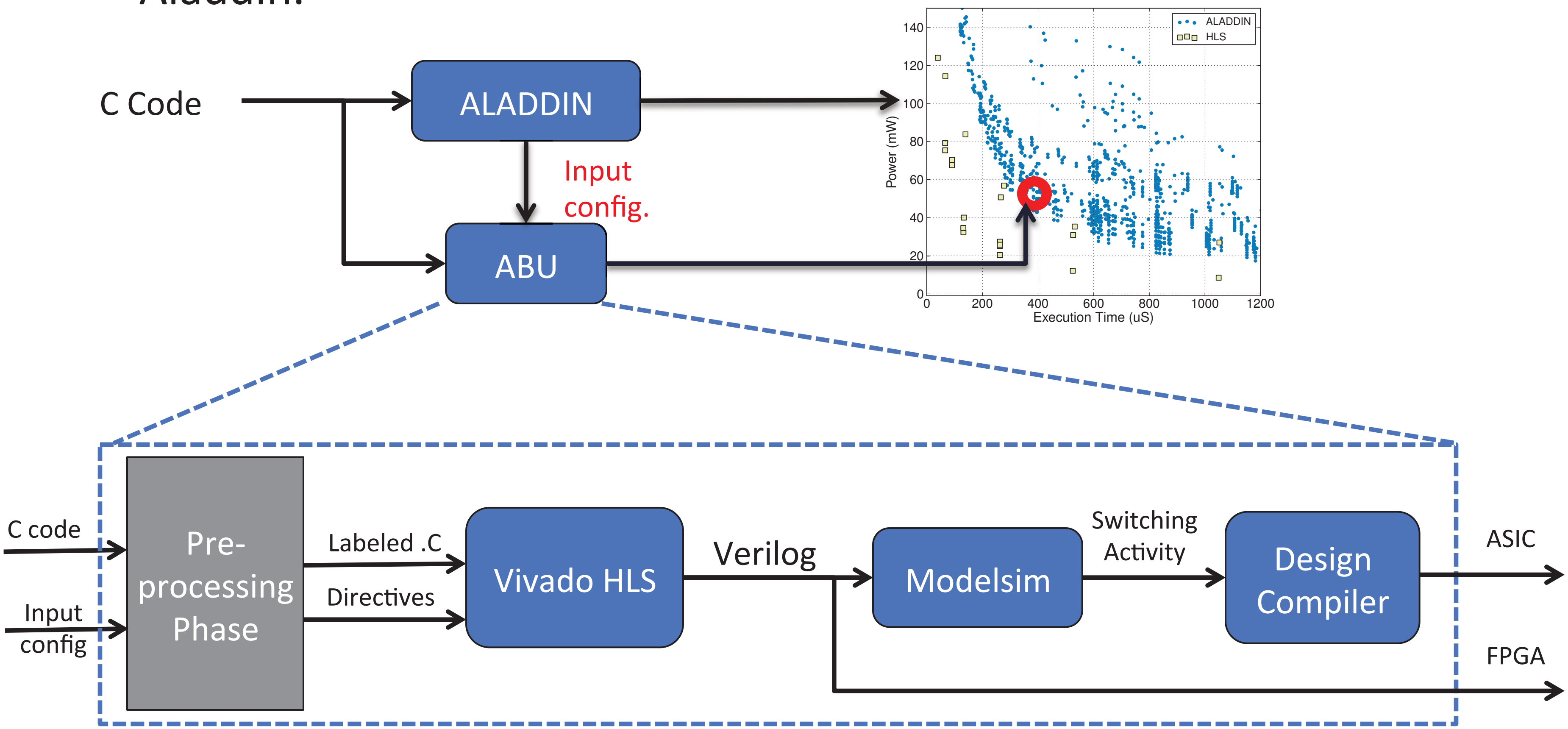


Aladdin/Abu Integration

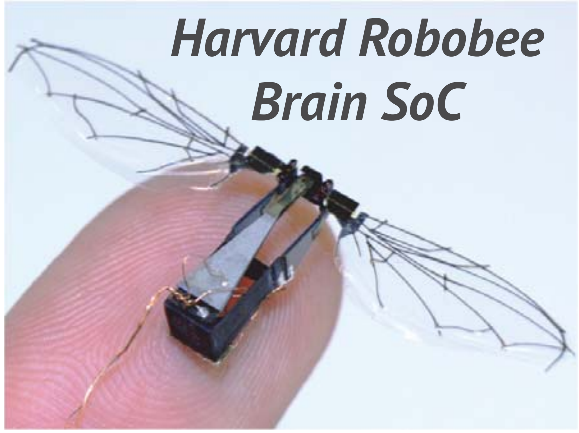
- ***Abu*** automatically generates the accelerator design (i.e., RTL) via backend high-level synthesis (HLS) tools guided by directives from Aladdin.

Aladdin/Abu Integration

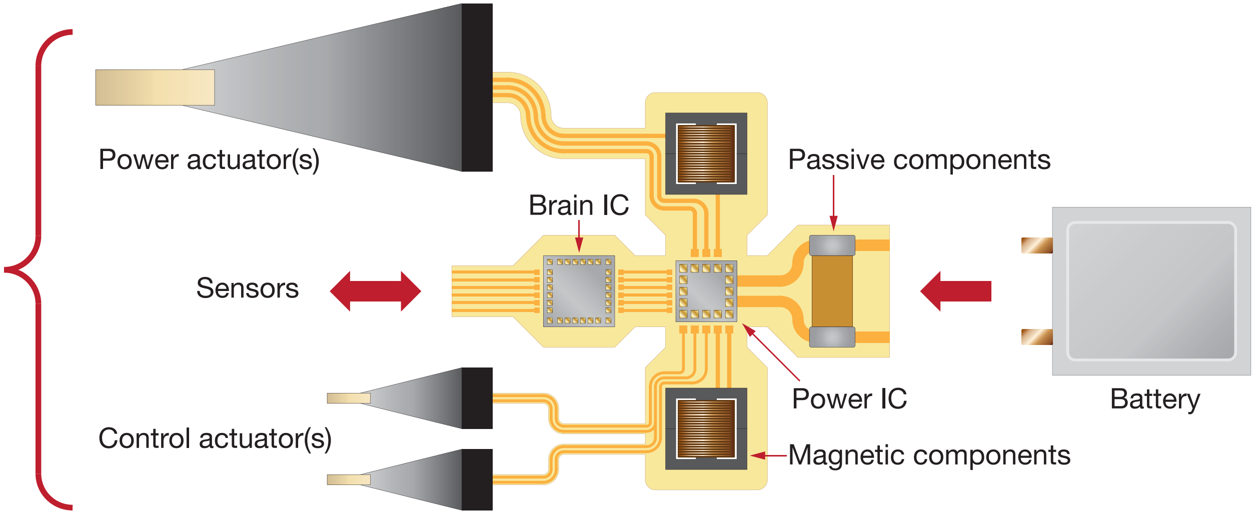
– ***Abu*** automatically generates the accelerator design (i.e., RTL) via backend high-level synthesis (HLS) tools guided by directives from Aladdin.



Where are we applying Aladdin?

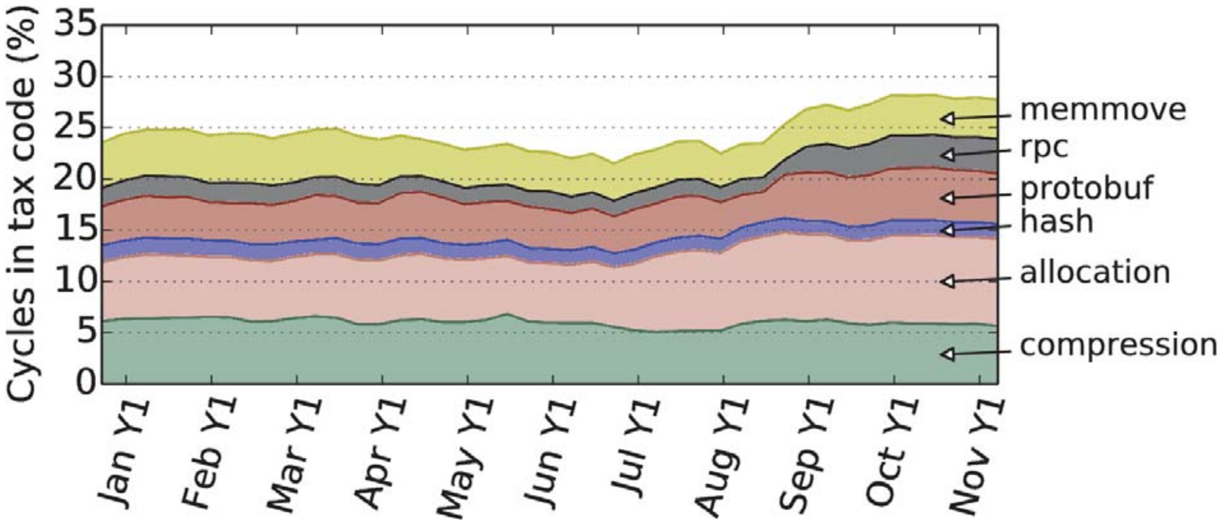


Harvard Robobee Brain SoC



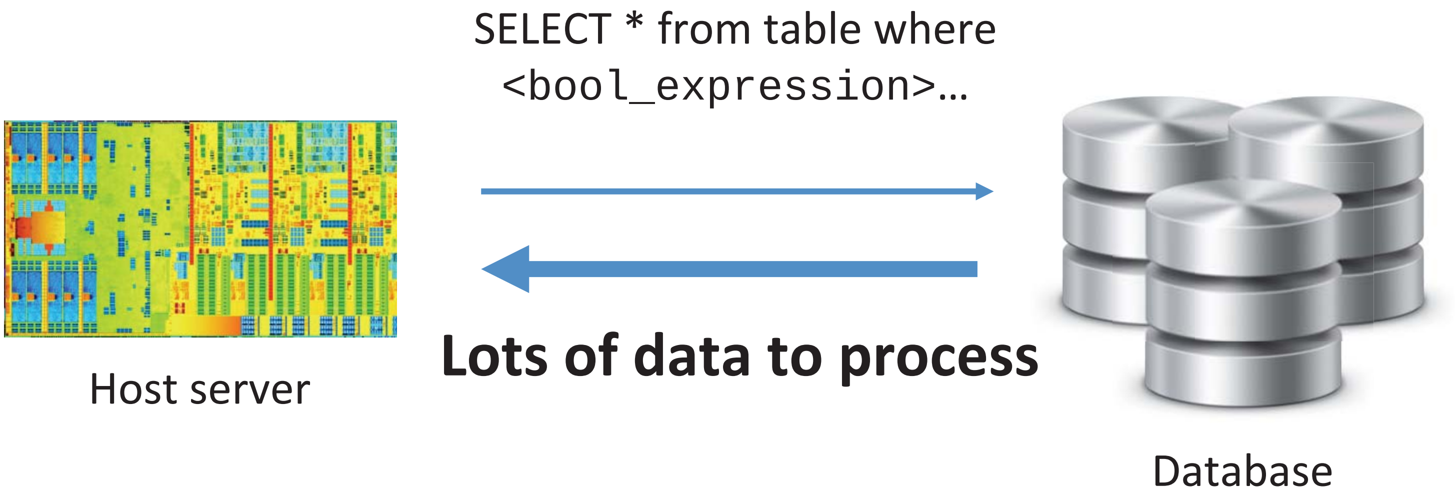
[X. Zhang et al., VLSI2015]

"Datacenter Tax" prime candidates for specialization



[S. Kanev et al., ISCA 2015]

Where else are we applying Aladdin?



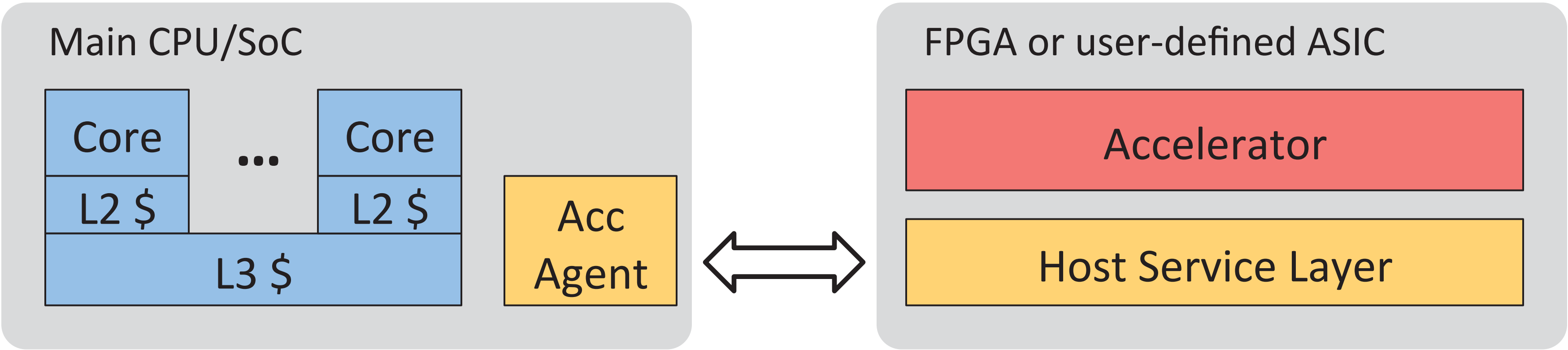
SELECT * from table where `<bool_expression>`...

Lots of data to process

Host server

Database

[S. Xi et al., SIGMOD DaMoN'15]



Main CPU/SoC

Core ... Core

L2 $ L2 $

L3 $

Acc Agent

FPGA or user-defined ASIC

Accelerator

Host Service Layer

[Y. Shao et al., SCAW 2015]

Conclusions

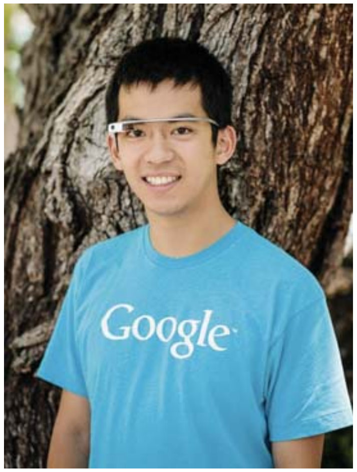
- Embrace the golden age of design
  - 10-100x wins are certainly available
  - Existence proof: Anton, GPUs, mobile SoCs
- Specialization provide one path, more will emerge
  - Models and tools for system designers will need to evolve to address this

Questions & Acknowledgements

Sophia Shao

Brandon Reagen

Svilen Kanev



Sam Xi

CFAR

NSF

DARPA

Google

intel