arm Research

ManyCore Summer School 2018

Architecture, Security and Specialization

Matt Horsnell (matt.horsnell@arm.com)
Architecture Research Lead, Arm Research

Agenda

About Me

Scene Setting

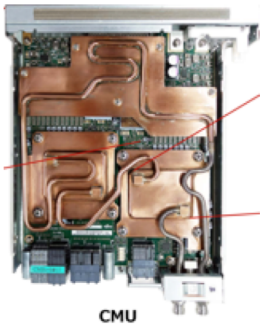
Session 1 - Security

Session 2 - Specialization

arm Research

About me

1999-2003	MEng. University of Bristol
2003-2007	PhD. University of Manchester “A chip multi-cluster architecture with locality based work distribution”
2007-2008	Post-doc “Object-Based Transactional Memory”
2008-2010	Azuro (now part of Cadence) Clock Tree Synthesis and Clock Concurrent Optimization
2010-	Arm Research Armv8 Crypto Extensions, Scalable Vector Extensions, Statistical Profiling Extensions



arm Research

Arm Research

Mission

- Partner to accelerate innovation and transfer research knowledge across Arm and the Arm Ecosystem

Objectives

- Build a pipeline to create and bring future technology into the Arm Ecosystem
- Create and maintain the emerging technology landscape
- Enable innovative research through collaboration and partnership

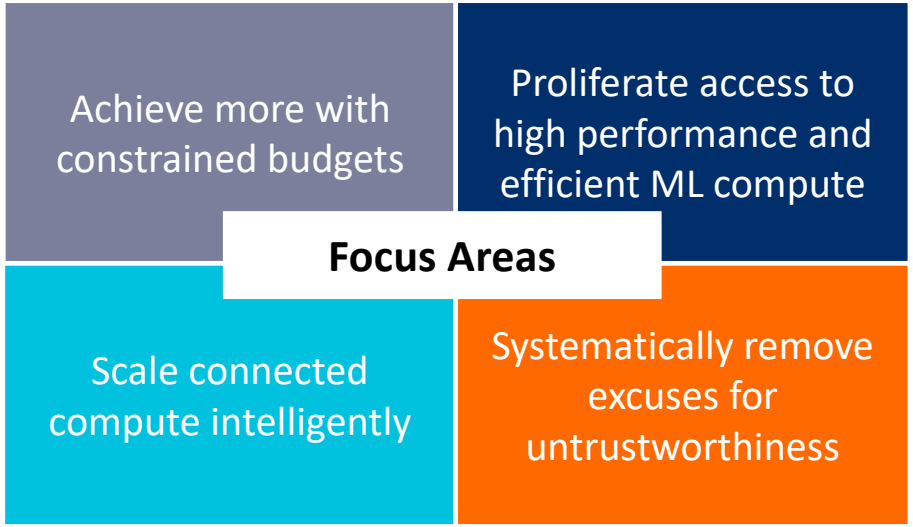
150+ researchers worldwide (~4% of Arm)
Cambridge UK, Austin TX, San Jose CA, Boston MA, Seattle WA

arm Research

Research Group Charter

Our research focus is defined by these significant challenges

- Demand for increased performance and functionality continue in spite of limited future process scaling
- High growth of connected devices will not be possible without intelligent approaches to scaling connected compute
- Machine learning workloads must migrate to constrained devices at the edge
- Growth of diverse connected systems pose significant security challenges



arm Research

Archeology

Future Directions in Computer Architecture

System reliability must be translated into what the end user actually sees—system availability, integrity, and security.

There has to be an explicit communications interface between loosely coupled processors.

Recent developments in technology are making special-purpose solutions to information processing problems more attractive.

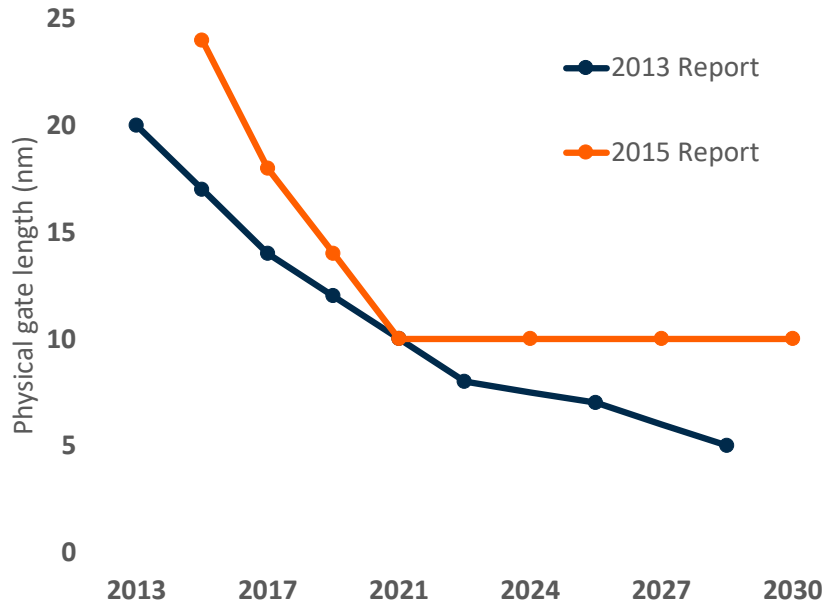
Of course, users have wanted these attributes for a long time. The problem is, how much are they willing to pay for them?

"Intelligent memory," the technique for merging some processing into memory, needs more work.

March 1978

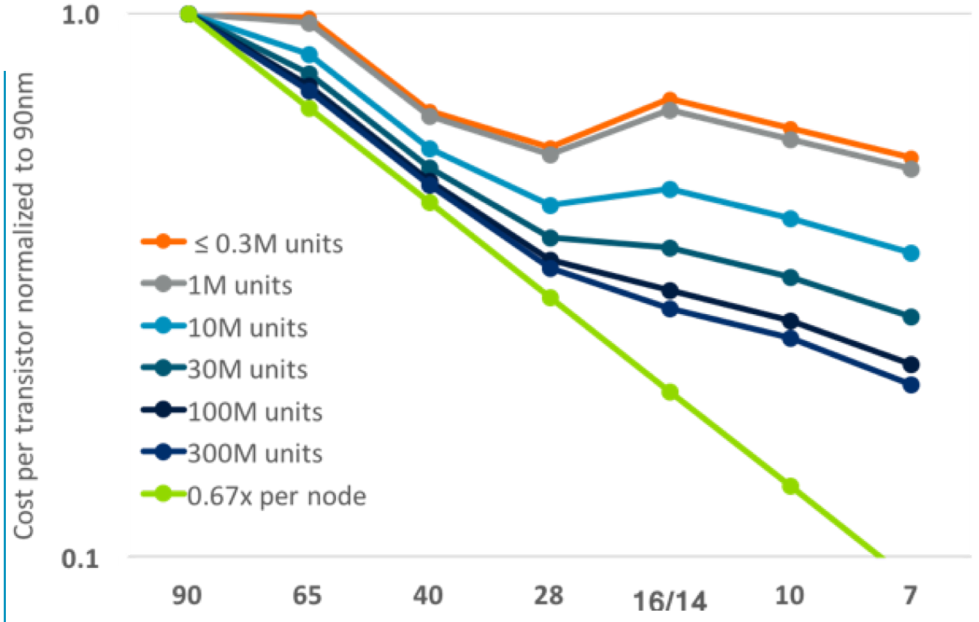
arm Research

Transistors aren’t scaling the way they used to…



ITRS previously predicted shrinkage until at least 2028, but latest report shows feature size going flat. ITRS chair: “Some further scaling may be possible after transistors go vertical”.

source: spectrum.ieee.org/semiconductors/devices/transistors-could-stop-shrinking-in-2021

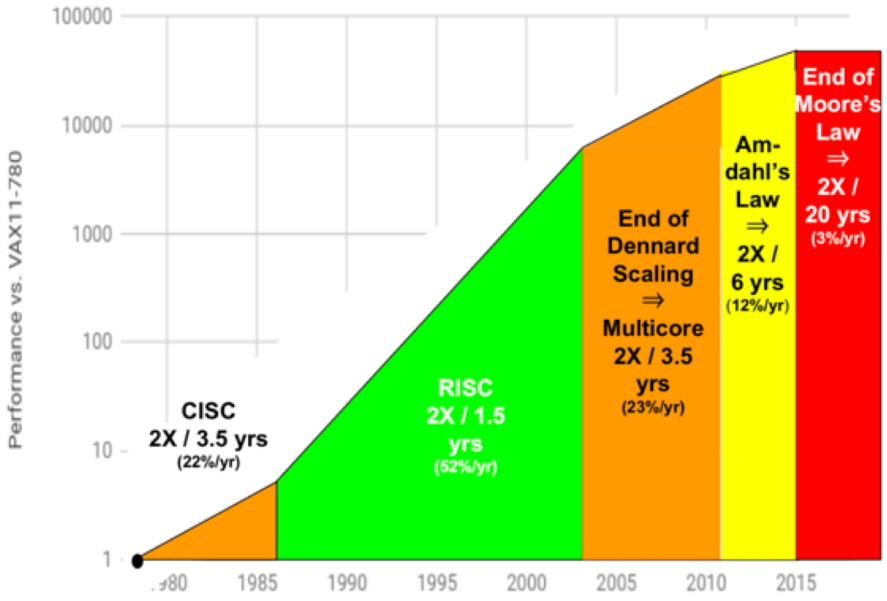


Unless your design run is sufficiently large, the cost per transistor stopped scaling at 28nm: certainly no longer following

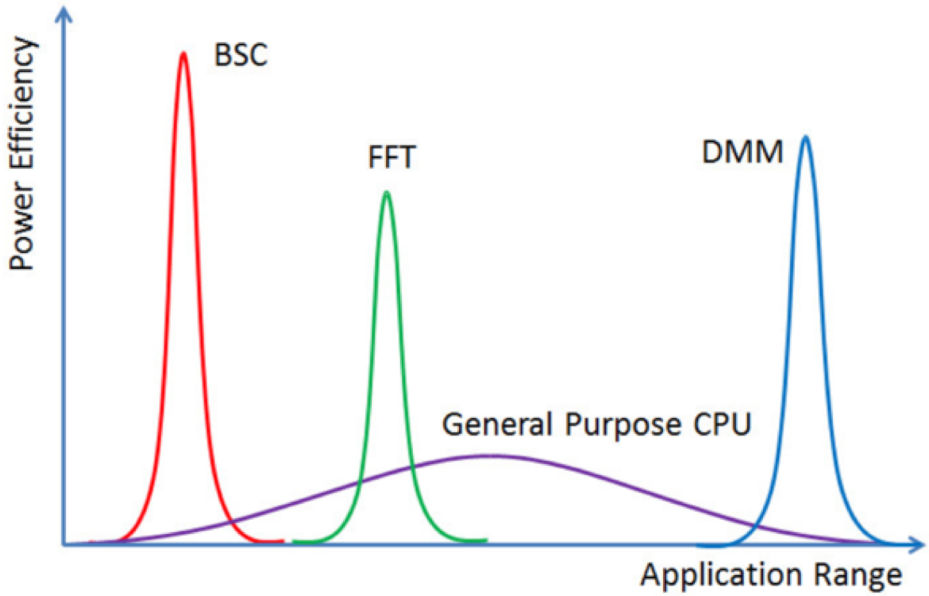
source: G. Yeric, IEDM 2015 Keynote

arm Research

… general purpose performance is stalling …



Based on SPECintCPU. Source: John Hennessy and David Patterson, Computer Architecture: A Quantitative Approach, 6/e. 2018



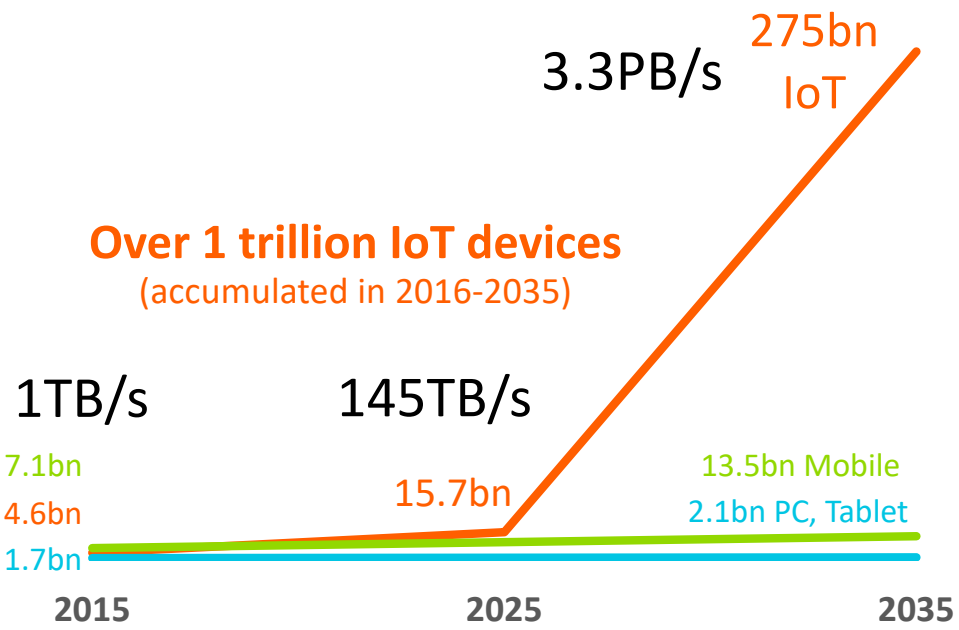
MultiAmdahl: Optimal Resource Allocation in Heterogeneous Architectures

Yavits et. al – IEEE Comp. Arch. Letters. 13 (1) pg. 37-40.

arm Research

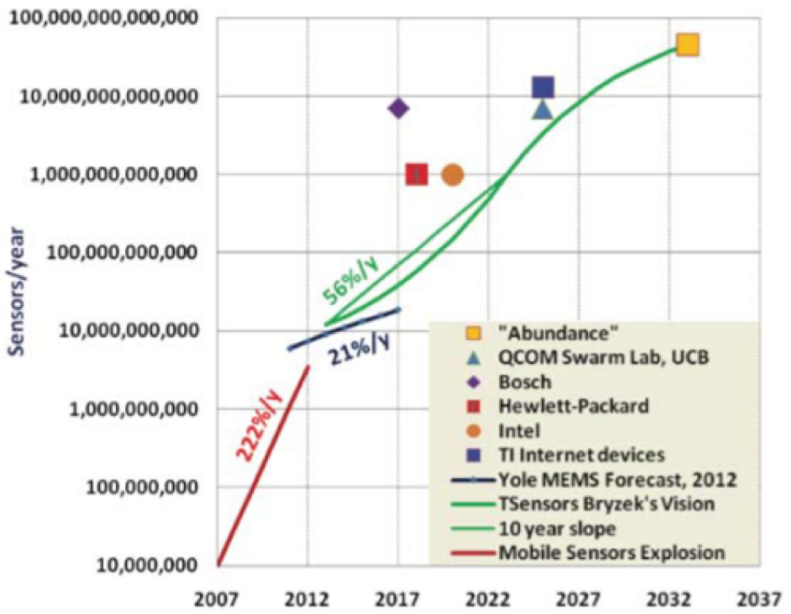
... in a new connected device era

Connected Device Forecast



Source: softbank, based on data by Ericsson
Bandwidth source: cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html

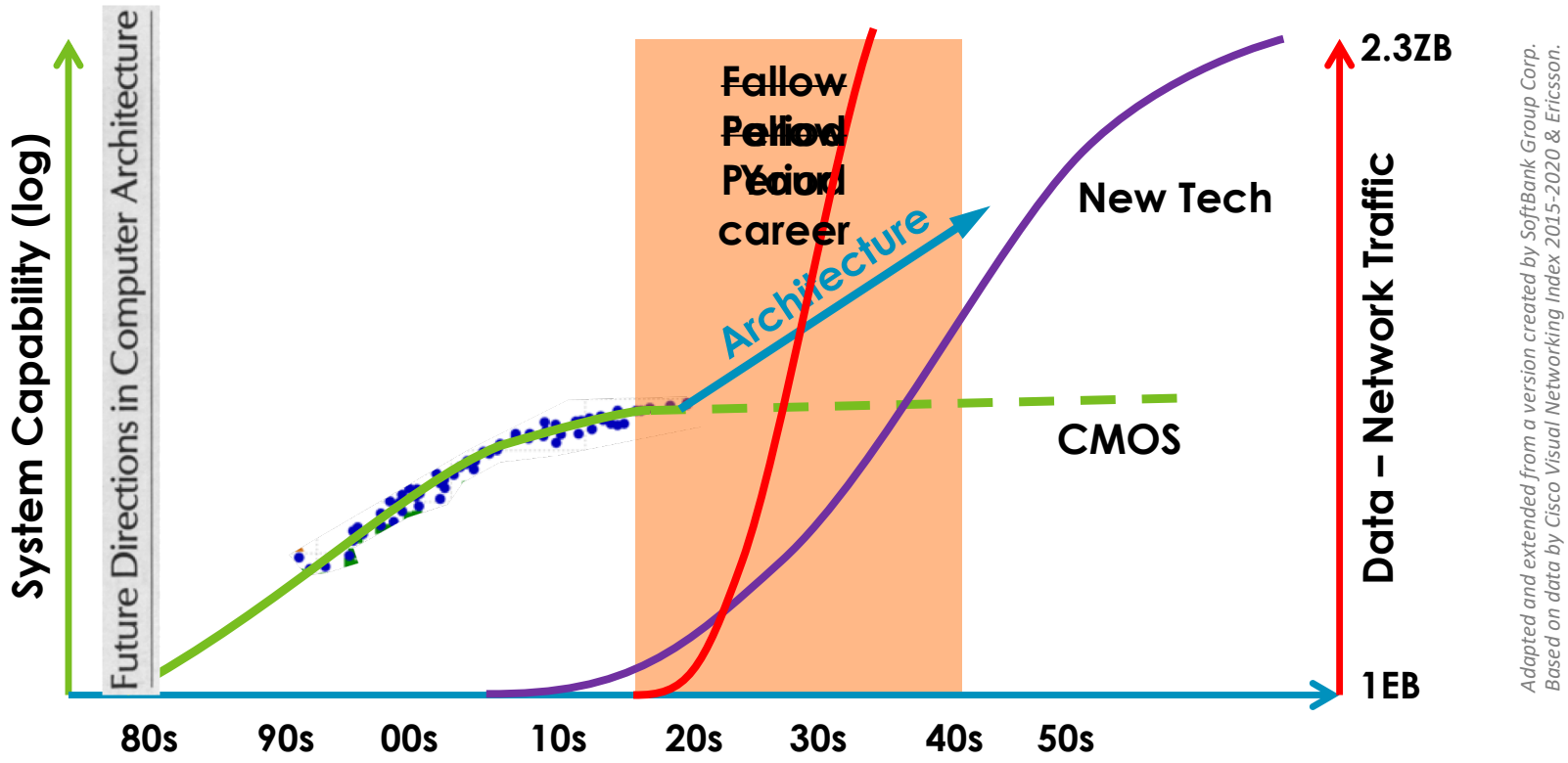
Sensors will populate the world of the IoE



Source: itrs

arm Research

Or to paint this another way…



Adapted and extended from "Advancing Computer Systems without Technology Process"
Christos Kozyrakis, Keynote, ISPASS 2013.

arm Research

Two challenges moving forward

I think this is an excellent time to be studying Computer Architecture – two grand challenges and plenty of competing ideas for solving them...

The data explosion is made up of **our** data

- our movements, payments, photos, connections, details, *thoughts*...
- data is worth money, which makes it worth stealing...
- **Protecting data and privacy is paramount**

Processing this data requires **immense** compute

- if all Google's customers used 3 minutes of voice recognition per day they would need to double their datacenters

Architecture, Security and Specialization.

arm Research

ManyCore Summer School 2018

Plucking Lemons

Can Architecture remove the low hanging Security fruit?

Matt Horsnell

Architecture Research Lead, Arm Research

Acknowledgements

This is a collage of material from a number of people:

- Paul Kocher and Google Project Zero.
- Mark Hill, Ruby Lee, Simha Sethumadhavan, Timothy Sherwood
- Arm colleagues from both the Architecture and the Central technology groups.

Any mistakes or inaccuracies are mine...

arm Research

Agenda

I’m going to attempt to talk about security from the perspective of a computer architect.

Let me make it clear up front, that I’m not a security “expert”.

Definition of “Architecture” (in a computer science sense)

The Architecture is the interface *contract* between:

Hardware and the Software

Different layers of Software

Different components of Hardware

For CPU architecture :

Software obeying the architected behaviours is portable across different hardware

Architecture defines what the hardware implementation must do

Micro-architecture defines how the hardware does it

An architecture doesn’t define everything

It only defines the timing independent functional behavior*

- which allows implementations the flexibility to design a micro-architecture that suits the required performance, power and area constraints of the target application.

Notably doesn’t define the timing of each instruction

- floating point multiply takes 3 cycles on cpu A, 7 on cpu B.
- floating point multiply of value 3.5 * 2.5 may take 3 cycles, 3.75 * 2.5 may take 10.
- the time taken for an address A to be loaded from memory can take 2 cycles if cached, it might take 10-200 cycles if it isn’t.

Performance as a driver

Performance goal

- Lowest time to reach the result same as running the program **in-order**

Single-thread speed gains require getting **more done per clock cycle**

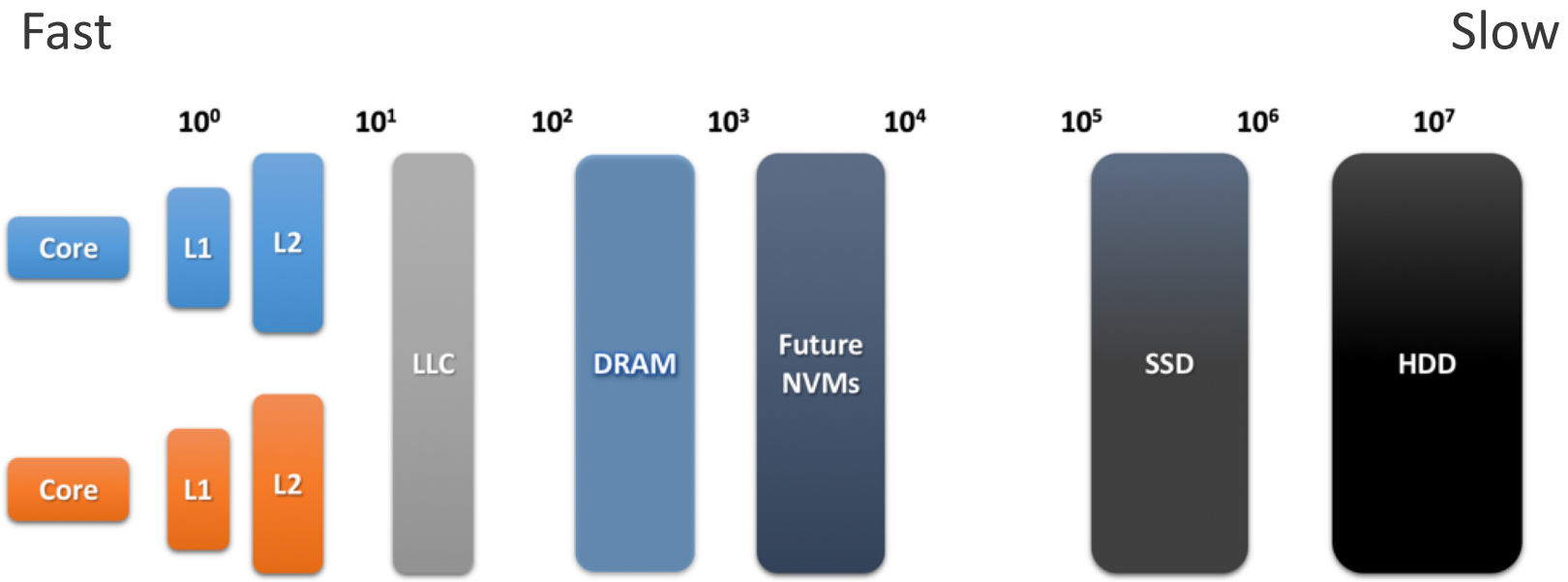
- Memory latency is slow and not improving much
- Clock rates are maxed out: Pentium 4 reached 3.8 GHz in **2004**

How to do more per clock?

- Reducing memory delays → Caches
- Working during delays → Speculative execution

Reducing memory delays → Caches…

- Architecture advances only help performance when memory can “keep up” w/cores
- Hierarchy of caches added to exploit data’s ***temporal*** and ***spatial*** locality

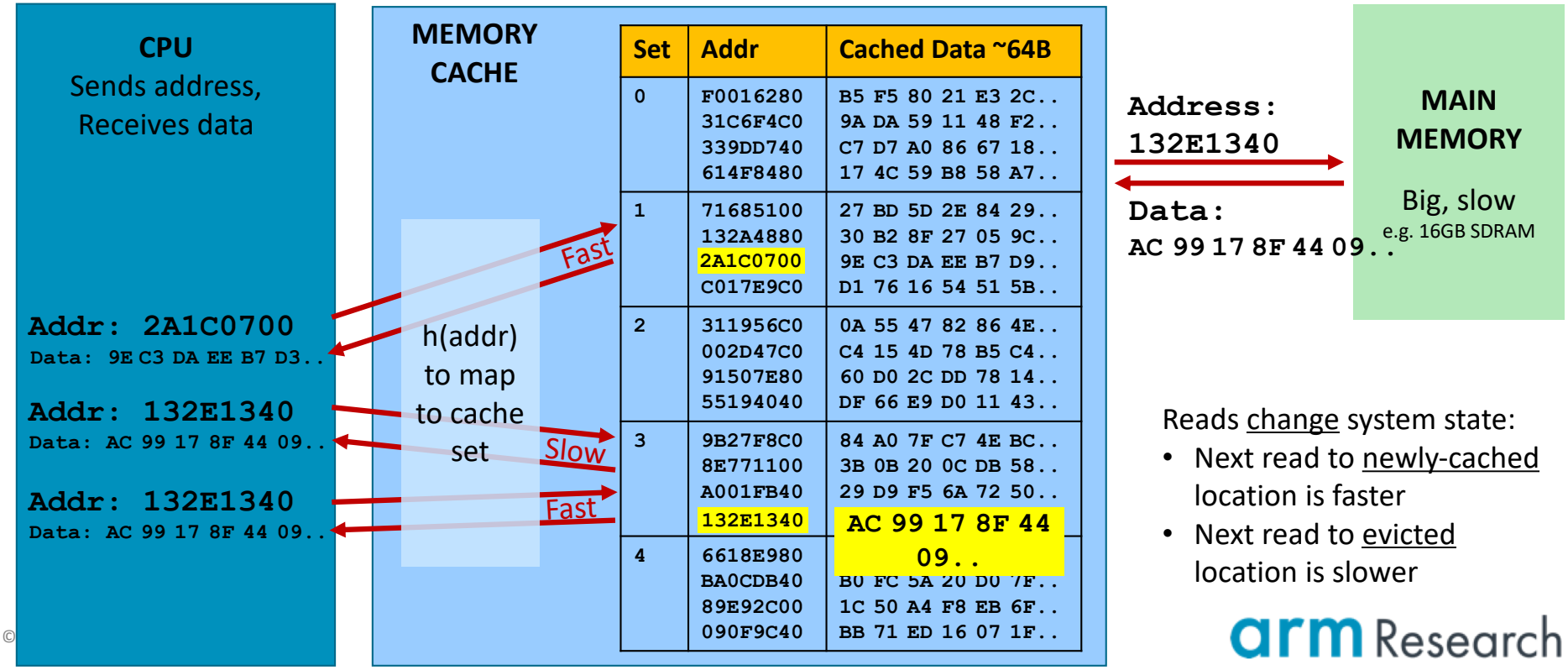


arm Research

**Animations from Paul Kocher

Caches for dummies...

Caches hold local (fast) copy of recently-accessed 64-byte chunks of memory



Reads change system state:

- Next read to newly-cached location is faster
- Next read to evicted location is slower

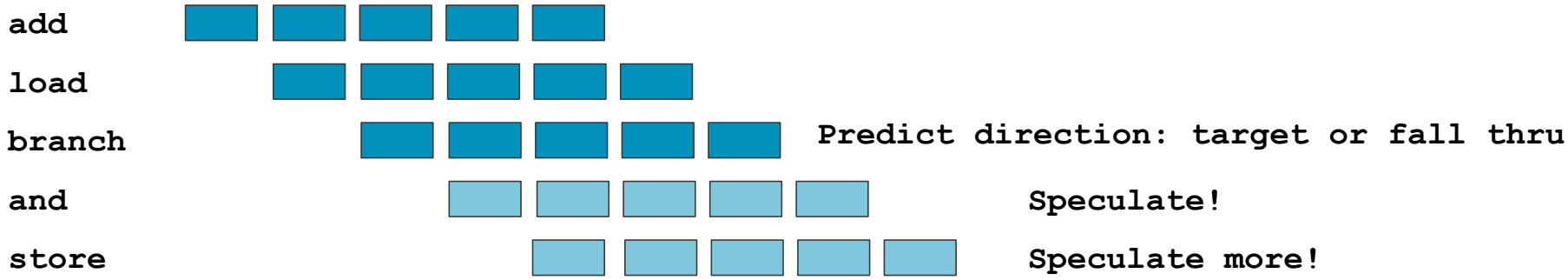
arm Research

**Animations from Mark Hill

Working during delays → Speculation…

Many steps (cycles) to execute one instruction; time flows left to right →

Go Faster: Pipelining, branch prediction, & instruction speculation



Speculation correct: Commit architectural changes of **`and`** (register) & **`store`** (memory) go fast!

Mis-speculate: Abort architectural changes (registers, memory); go in other branch direction

arm Research

Speculative execution

Instead of idling, CPUs can *guess* likely program path and do speculative execution

- Example:

```
if (uncached_value_usually_1 == 1)
    foo()
```

- Branch predictor: if() will probably be 'true' (based on prior history)
- CPU starts foo() speculatively -- but doesn't commit changes
- When value arrives from memory, if() can be evaluated definitively -- check if guess was correct:
  - Correct: Commit speculative work – performance gain
  - Incorrect: Discard speculative work

But what happens if you put these two pieces of knowledge together in devious ways?

1. Cache state is observable through timing....
2. Speculative loads might perturb the cache state...

Google Project Zero - Spectre/Meltdown

Google's Project Zero team raised a major significant security threat

- Affects all major CPU architectures, including x86, Power and Arm

Typical timing attack

- Use controlled parts of higher privilege-only data to form an address
- Fetch from this address into the cache
- By timing analysis (data access time), determine the value of the privilege data

Demonstrated by Google as real implementations – dumping higher privilege data at kb/s

Variant 1 (CVE-2017-5753): bypassing software checking of untrusted values

```
if (x < array1_size)
   y = array2[array1[x]*512];
```

Assume code in kernel API, where unsigned int `x` comes from untrusted caller

Execution without speculation is safe

- CPU will not evaluate `array2[array1[x]*512]` unless `x < array1_size`

What about with speculative execution?

arm Research

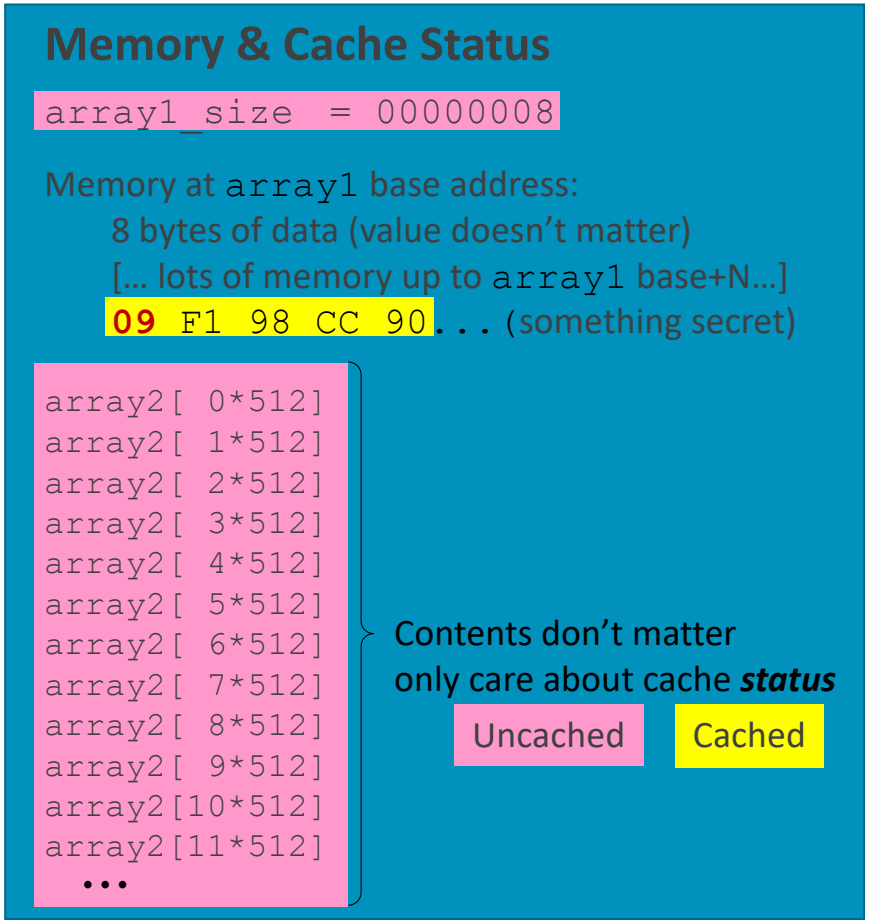
**Animations from Paul Kocher

Conditional branch (Variant 1) attack

```
if (x < array1_size)
    y = array2[array1[x]*512];
```

Before attack:

- Train branch predictor to expect if() is true (e.g. call with `x` < `array1_size`)
- Evict `array1_size` and `array2[]` from cache



Conditional branch (Variant 1) attack

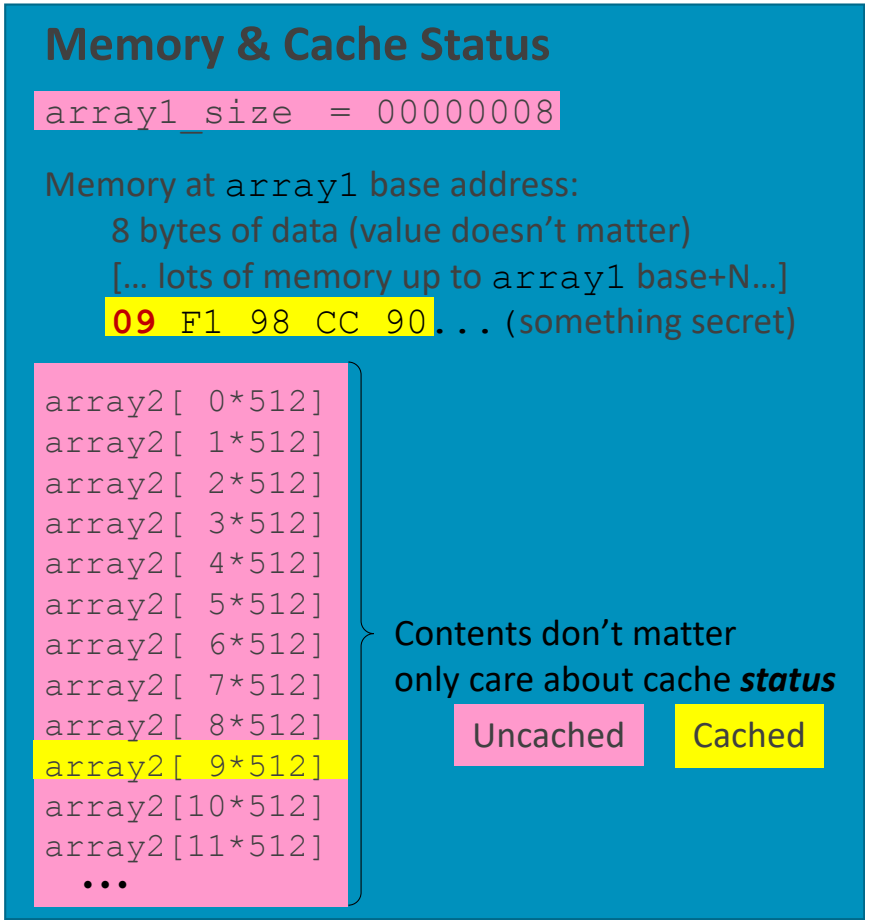
```
if (x < array1_size)
    y = array2[array1[x]*512];
```

Attacker calls victim with `x`=N (where N > 8)

- Speculative exec while waiting for `array1_size`
  - Predict that if() is true
  - Read address (`array1` base + `x`) w/ out-of-bounds `x`
  - Read returns secret byte = **09** (fast – in cache)
  - Request memory at (`array2` base + **09***512)
  - Brings `array2[09*512]` into the cache
  - Realize if() is false: discard speculative work
- Finish operation & return to caller

Attacker measures read time for `array2[i*512]`

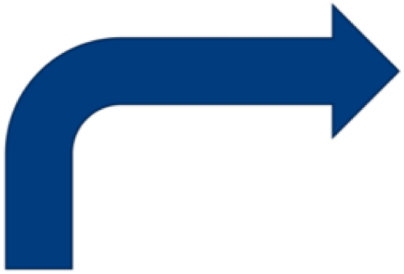
- Read for `i`=**09** is fast (cached), revealing secret byte
- Repeat with many `x` (eg ~10KB/s)



Spectre visualized...

- Jason Lowepower (Professor @ UC Davis) recreated spectre in a gem5 out-of-order cpu, and visualized it bit.ly/gem5-spectre

**Diagrams from Jason Lowepower



```
void victim_function(size_t x) {
  if (x < array1_size) {
    temp &= array2[array1[x] * 512];
  }
}
```

```
000000000040105e <victim_function>:
  40105e:    push   %rbp
  40105f:    mov    %rsp,%rbp
  401062:    mov    %rdi,-0x8(%rbp)
  401066:    mov    0x2bf014(%rip),%eax
  40106c:    mov    %eax,%eax
  40106e:    cmp    -0x8(%rbp),%rax
  401072:    jbe    40109f <victim_function+0x41>
  401074:    mov    -0x8(%rbp),%rax
  401078:    add    $0x6c00a0,%rax
  40107e:    movzbl (%rax),%eax
  401081:    movzbl %al,%eax
  401084:    shl    $0x9,%eax
  401087:    cltq
  401089:    movzbl 0x6c1d80(%rax),%edx
  401090:    movzbl 0x2e0ce9(%rip),%eax
  401097:    and    %edx,%eax
  401099:    mov    %al,0x2e0ce1(%rip)
  40109f:    pop    %rbp
  4010a0:    retq
```

https://gist.github.com/ErikAugust/724d4a969fb2c6ae1bbd7b2a9e3d4bb6

```
000000000040105e <victim_function>:
  40105e:     push   %rbp
  40105f:     mov    %rsp,%rbp
  401062:     mov    %rdi,-0x8(%rbp)
  401066:     mov    0x2bf014(%rip),%eax
  40106c:     mov    %eax,%eax
  40106e:     cmp    -0x8(%rbp),%rax
  401072:     jbe    40109f <victim_function+0x41>
  401074:     mov    -0x8(%rbp),%rax
  401078:     add    $0x6c00a0,%rax
  40107e:     movzbl (%rax),%eax
  401081:     movzbl %al,%eax
  401084:     shl    $0x9,%eax
  401087:     cltq
  401089:     movzbl 0x6c1d80(%rax),%edx
  401090:     movzbl 0x2e0ce9(%rip),%eax
  401097:     and    %edx,%eax
  401099:     mov    %al,0x2e0ce1(%rip)
  40109f:     pop    %rbp
  4010a0:     retq
```

```
void victim_function(size_t x) {
  if (x < array1_size) {
    temp &= array2[array1[x] * 512];
  }
}
```

load array1_size

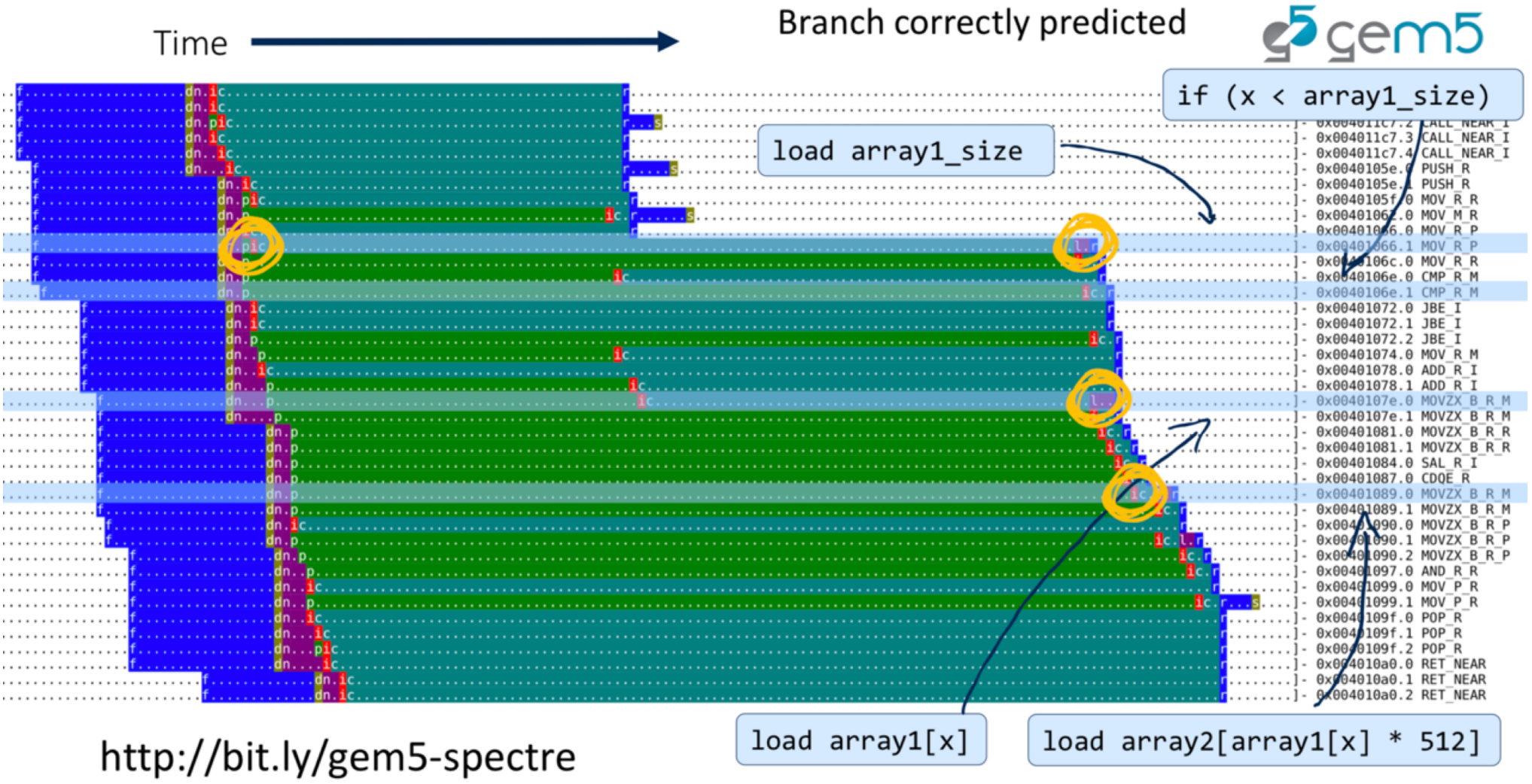
if (x < array1_size)

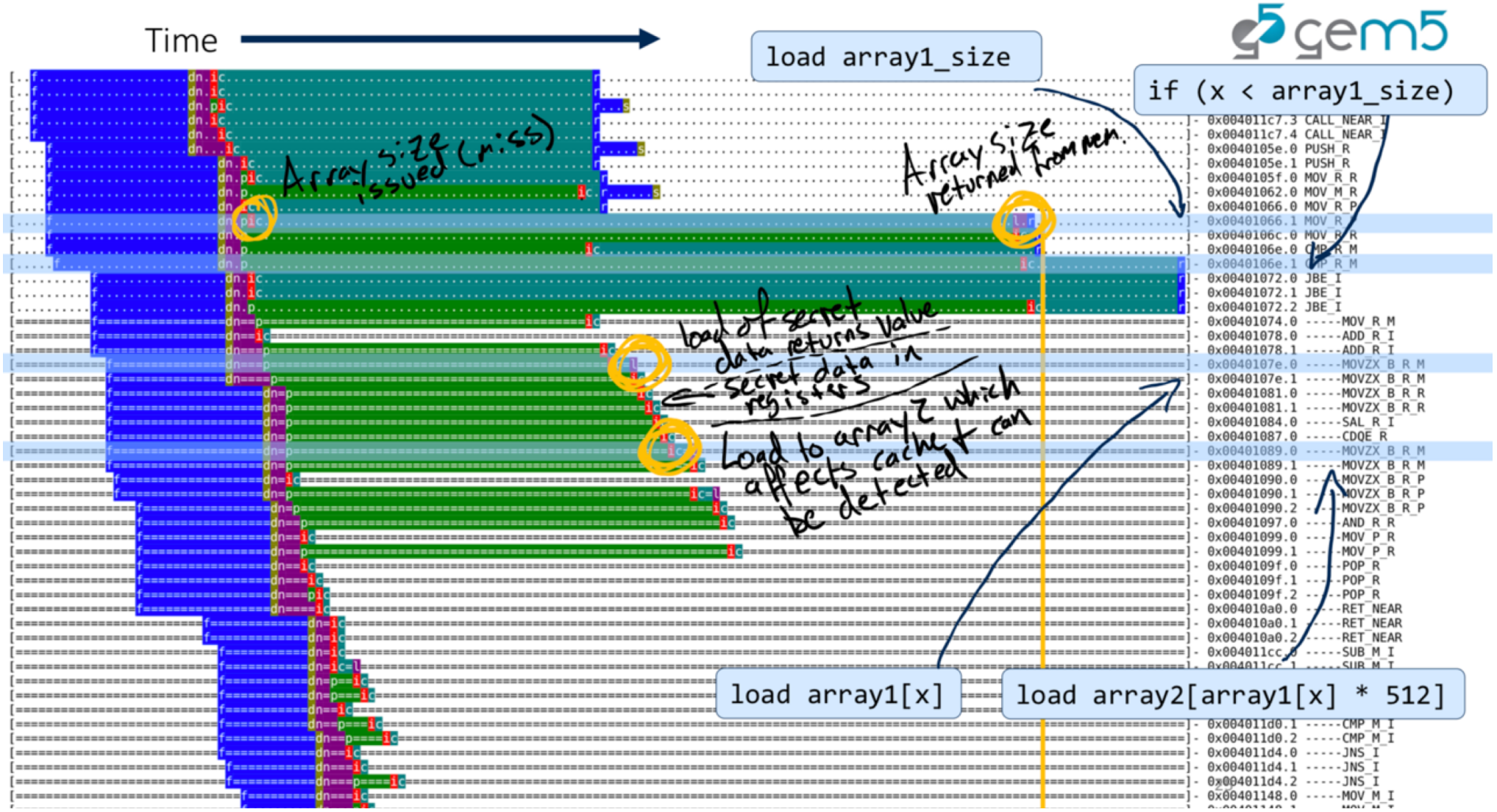
load array1[x]

load array2[array1[x] * 512]

Modifies addresses present in L1 cache

UCDAVIS


Time
Branch correctly predicted
gem5
if (x < array1_size)
load array1_size
load array1[x]
load array2[array1[x] * 512]
http://bit.ly/gem5-spectre


Time
gem5
load array1_size
if (x < array1_size)
Array size issued (r:ss)
Array size returned from mem.
load of secret data returns value
<- secret data in registers
Load to array2 which affects cache & can be detected
load array1[x]
load array2[array1[x] * 512]

Variant 2 (CVE-2017-5715): forcing privileged speculation by training branch predictors

```
0x1000  ;EL2 code at 0x1000 loads a private key into x0
        LDR x0, [address of a private key] ; On some uArch, a branch instruction might
                                           ; be required @0x1004 or after for this attack
        ;Somewhere else, EL2 code at address 0x2000, no relation to code at 0x1000
0x2000  MOV x0, x0, lsl#x2
        LDR x3, [x0]
```

Attack

EL1 trains Branch Prediction to branch from 0x1004 to 0x2000 (and preloads L1 cache with known data)

EL2 starts execution and goes through code at 0x1000

Processor speculatively branches from 0x1004 to 0x2000

One cache line, which address is formed out of the “Key” is *speculatively* loaded into the cache – evicting data loaded by EL1

Same timing attack as Variant 1 is used.

A fix is to tag branch predictors entries with context, you then can’t train in Elx to make Elx+n jump to arbitrary locations.

arm Research

Variant 3 (CVE-2017-5754): using speculative reads of inaccessible data

```
;; Code executed at EL0
LDR    x0, [some privileged location EL0 doesn't have access to]
LDR    x1, [x0],lsl #offset
```

Attack

EL0 tries to access a "key" in a privileged location – will get denied but *gets speculatively executed*

Second Load address formed out of the "key" – *speculatively executed*

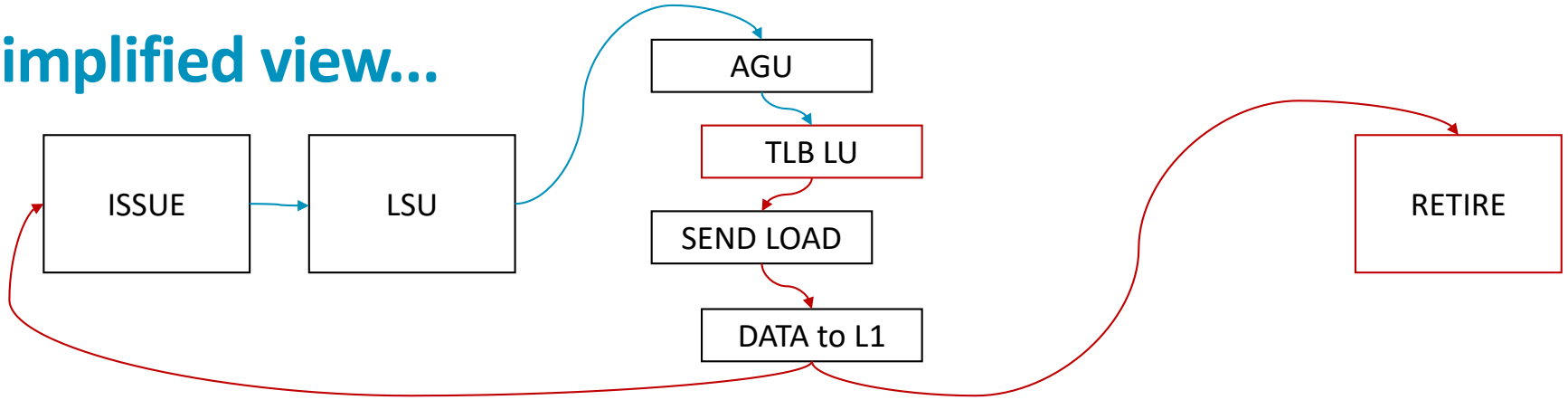
cache line from the second *speculative* load allocated into the L1D, evicting data previously pre-loaded by EL0

Same timing attack concept as before

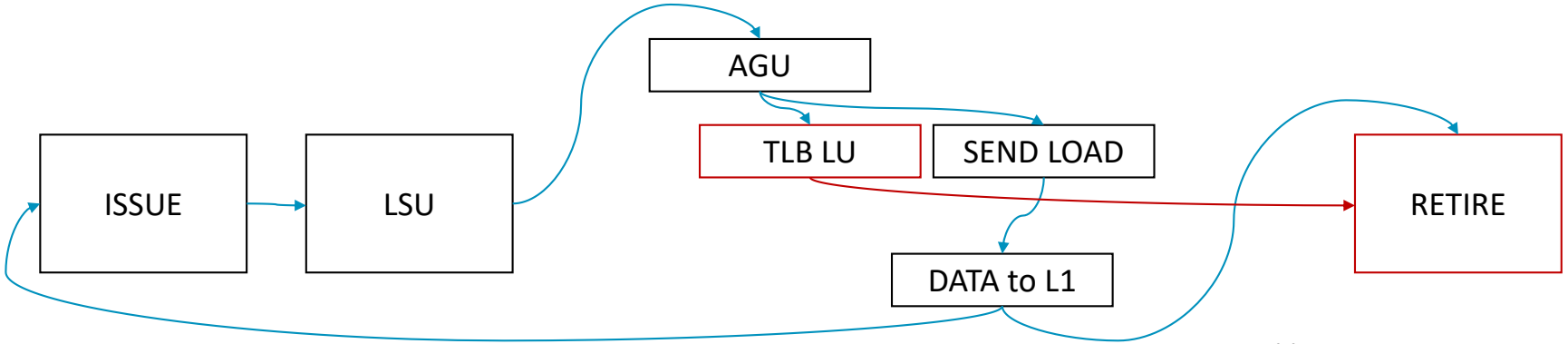
A CPU is immune to variant 3 if it cannot speculatively execute a load with address dependency on a previous load aborted

A software mitigation is by removing kernel mappings while at user level (small perf impact).

simplified view...




```
LDR     x0, [priv-addr]
LDR     x1, [x0],lsl #offset
```



** Hypothetical abstract machine

arm Research

Variant 3a (CVE-2017-5754): using speculative reads of inaccessible data

```
LDR X1, [X2] ; arranged to miss in the cache
CBZ X1, over ; This will be taken
MRS X3, TTBR0_EL1;
LSL X3, X3, #imm
AND X3, X3, #0xFC0
LDR X5, [X6,X3] ; X6 is an EL0 base address
over:
```

Very similar attack to variant 3 – disclosed by Arm.

Speculative read to an inaccessible control register e.g. TTBR0_EL1 which shouldn't be accessible from EL0.

Using the same offset and perturb cache technique as other variants, can read value of higher privilege system register.

arm Research

Variant 4 (CVE-2018-3639): Speculative bypassing of stores by younger loads despite the presence of a dependency

In many modern high-performance processors

- a performance optimization is made whereby a load to an address will speculatively bypass an earlier store whose target address is not yet known by the hardware, but is actually the same as the address of the load.
- When this happens, the load will speculatively read an earlier value of the data at that address than the value written by the store.
- That speculatively loaded value can then be used for subsequent speculative memory accesses that will cause allocations into the cache, and the timing of those allocations can be used as an observation side-channel for the data values selected as an address.

```
STR      x1, [x2]      ; x2 is slow resolving...
...
LDR      x3, [x4]      ; x4 contains the same address as x2
<arbitrary data processing of x3>
LDR      x5, [x6, x3]
```


arm Research

What can we do about these attacks?

- Mitigate with Architecture
  - Target and prevent the speculation in code
  - Fairly simple to prevent with new barriers
  - ... but hard to ensure all code and all scenarios are covered
- Mitigate by switching off certain speculation (e.g. speculative store bypassing)
  - heavy handed, a blunt instrument
- Mitigate with Micro-architecture
  - Track, Taint and disallow state changes for certain speculative changes
  - Requires intrusive micro-architecture
  - ... but potentially over-pessimistic and a performance hit

Architecture mitigation – speculation barriers

Arm announced 3 new speculation barriers to Armv8.4-A

CSDB Consumption of Speculative Data Barrier
is a memory barrier that controls speculative execution and data value prediction

SSBB Speculative Store Bypassing Barrier
is a memory barrier which prevents speculative loads from bypassing earlier stores to the same virtual address under certain conditions

PSSBB Physical Speculative Store Bypass Barrier
is a memory barrier which prevents speculative loads from bypassing earlier stores to the same physical address.

… but there's more

arm Research

Attack scenarios

- user process attacks user
- intra-process sandbox escape (JavaScript, safe languages…)
- user process attacks special modes (SGX/TrustZone/SMM…)
- hypervisor guests attack each other/host
- remoteuser process attacks kernel/driver
- timing attack
- …

Covert channels

- Changes in cache state
  [many variations]
- Cache state on entry affects timing of later operations that use shared resources
- Other digital (FPU, buffers…)
- Analog (power, RF…)
- …

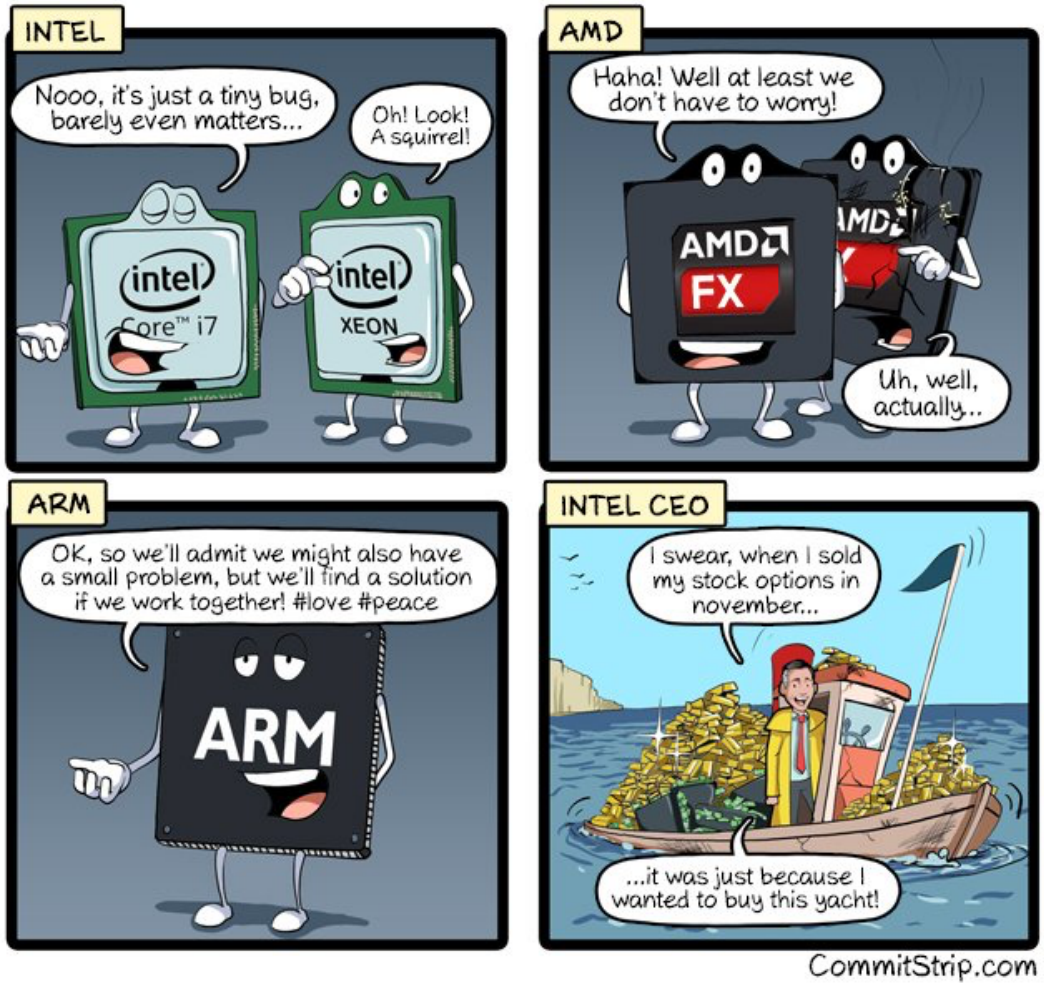
Speculation scenarios

- conditional branch
- indirect jump
- return instruction
  (e.g. Skylake w/ return stack buffer empty)
- speculative store bypass
- data speculation
- …

Target CPUs

- Arm
- Intel
- AMD
- Power
- Sparc

arm Research



arm Research

Vulnerabilities vs Exploits

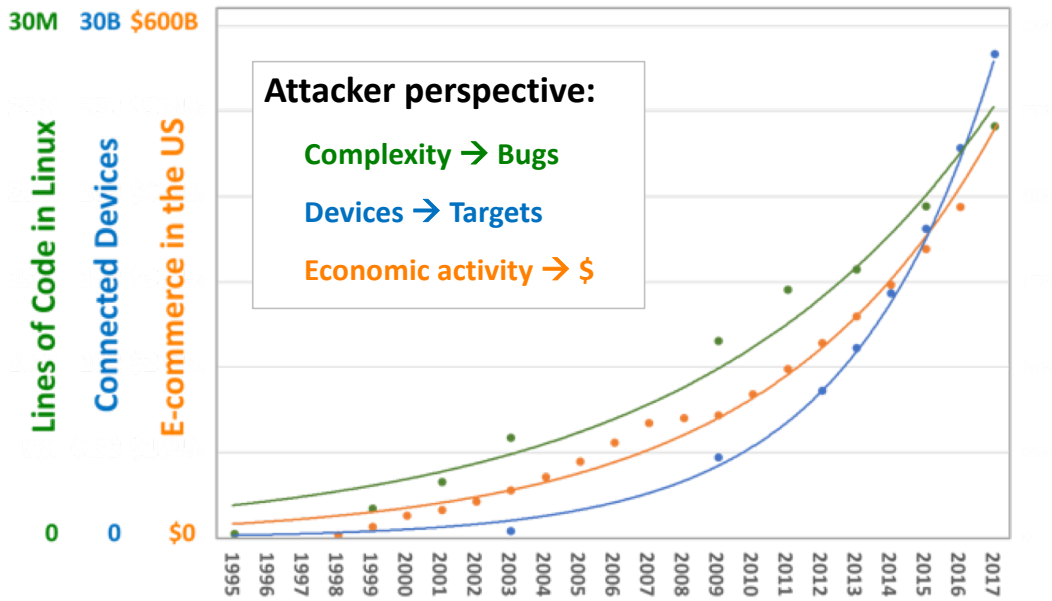
The **vulnerability** is the ‘open door’ allowing the attacker in…

- There are many and they cannot all be found in lab conditions

An **exploit** occurs when the attacker takes advantage of the vulnerability…

- For profit, nuisance or warfare

You can't "solve" security - but you can raise the bar



Past:

Performance dominated the economics of technology

Today:

Cost of insecurity[1] (10^{12} - 10^{13})

>

Value of performance gains[2] (10^{11})

Technical challenges:

Architecting and implementing robust and resilient systems

[1] "Estimating the Global Cost of Cyber Risk" (https://www.rand.org/pubs/research_reports/RR2299.html): $275B-$6.6T direct costs depending on assumptions, and $799B-$22.5T incl. indirect costs.
[2] My estimate. For reference, Intel's entire 2017 revenues were $62.8B, ARM <$2B.

arm Research

arm Research

ManyCore Summer School 2018

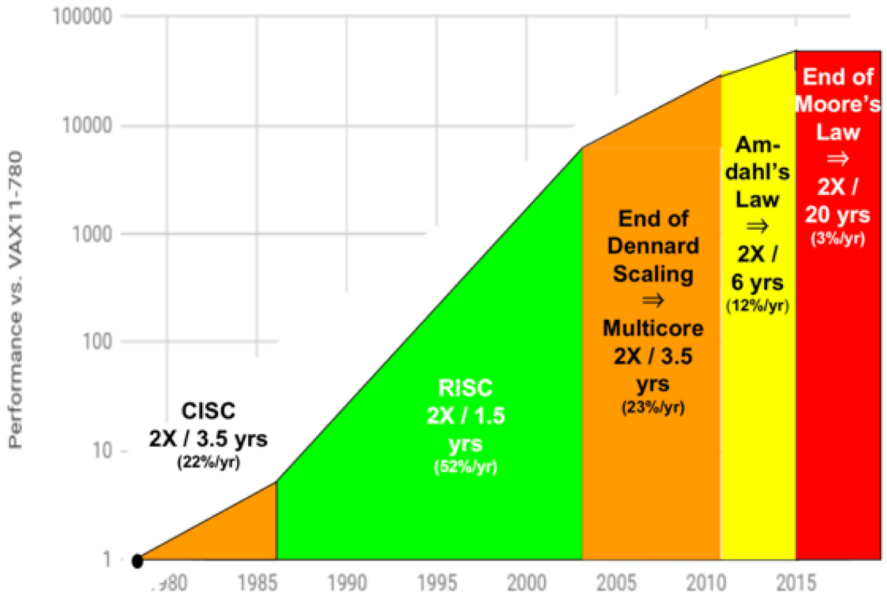
Apples and Oranges

Supporting Domain Specific Compute and Acceleration.

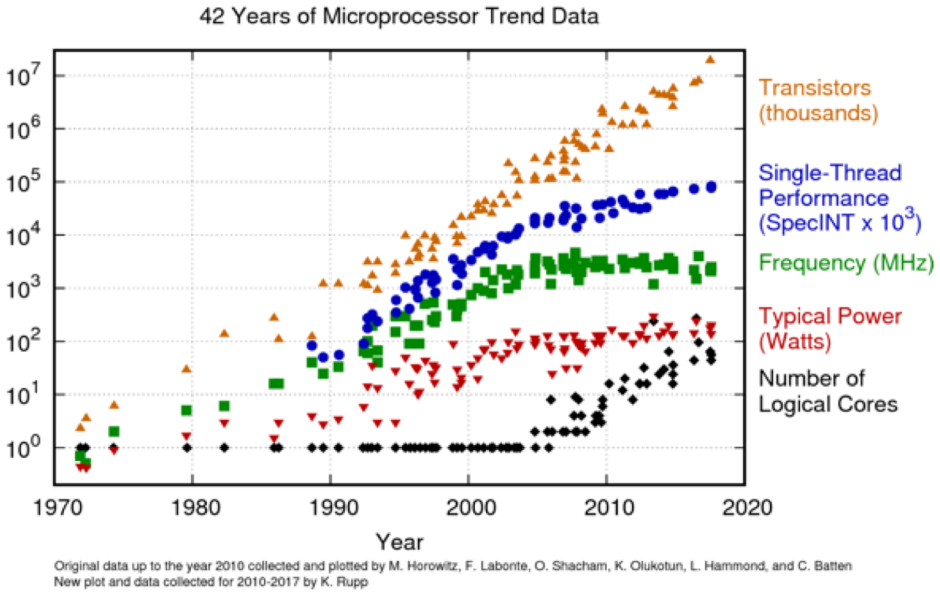
Matt Horsnell

Architecture Research Lead, Arm Research

Recap – general purpose performance is stalling ...

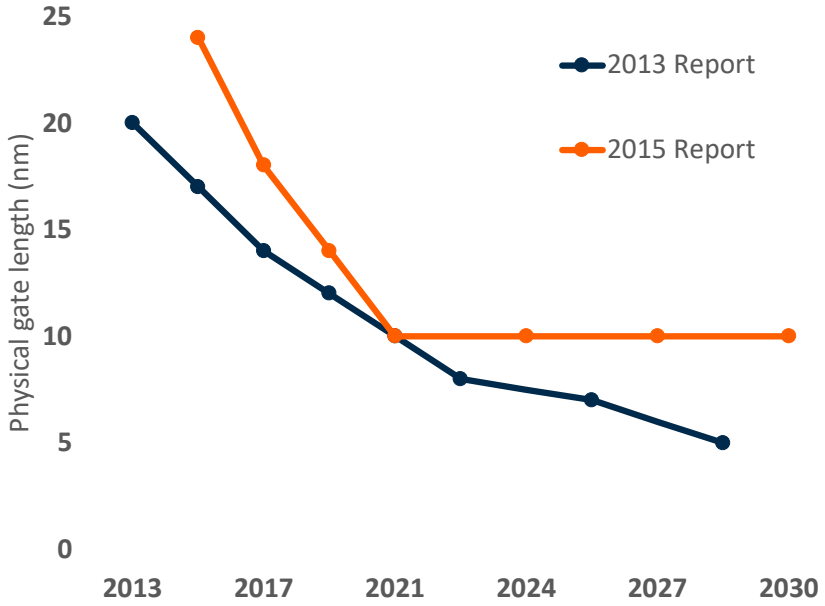


Based on SPECintCPU. Source: John Hennessy and David Patterson, Computer Architecture: A Quantitative Approach, 6/e. 2018



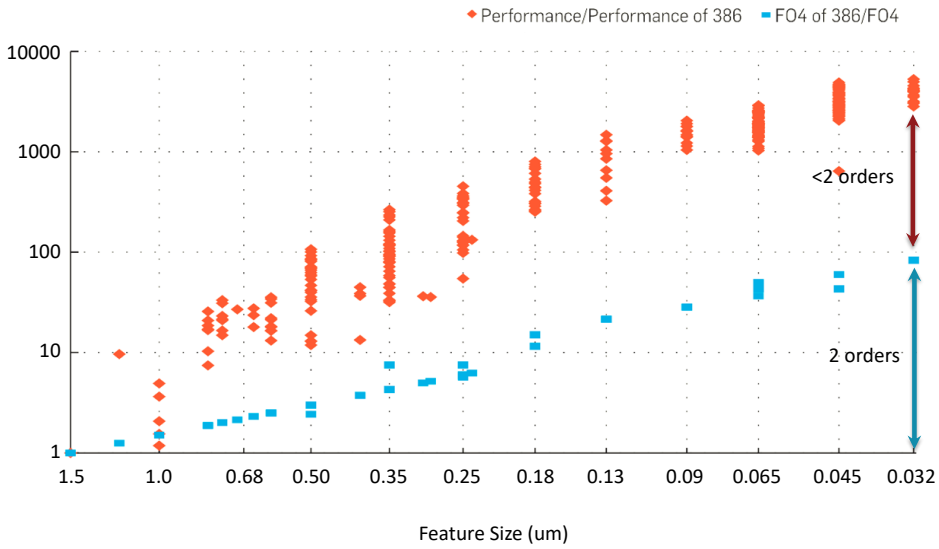
arm Research

Recap - process and speculative techniques plateaued



ITRS previously predicted shrinkage until at least 2028, but latest report shows feature size going flat. ITRS chair: “Some further scaling may be possible after transistors go vertical”.

source: spectrum.ieee.org/semiconductors/devices/transistors-could-stop-shrinking-in-2021

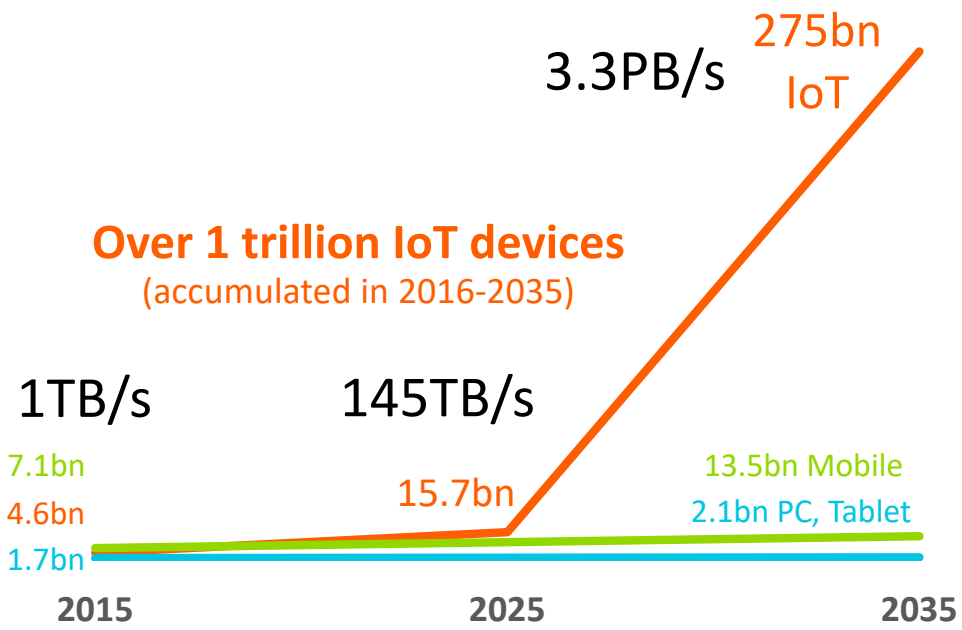


*data sourced from http://cpudb.stanford.edu/

arm Research

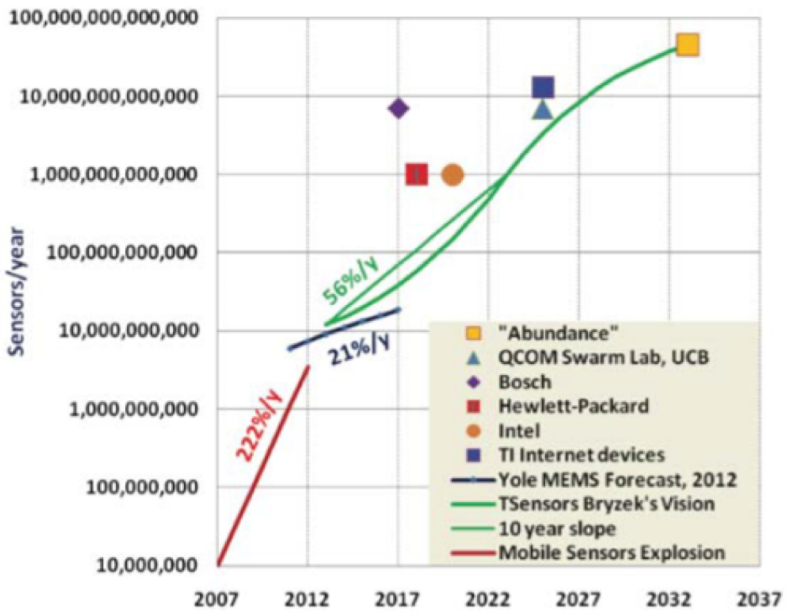
Recap – data is exploding and generating new applications

Connected Device Forecast



Source: softbank, based on data by Ericsson
Bandwidth source: cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html

Sensors will populate the world of the IoE

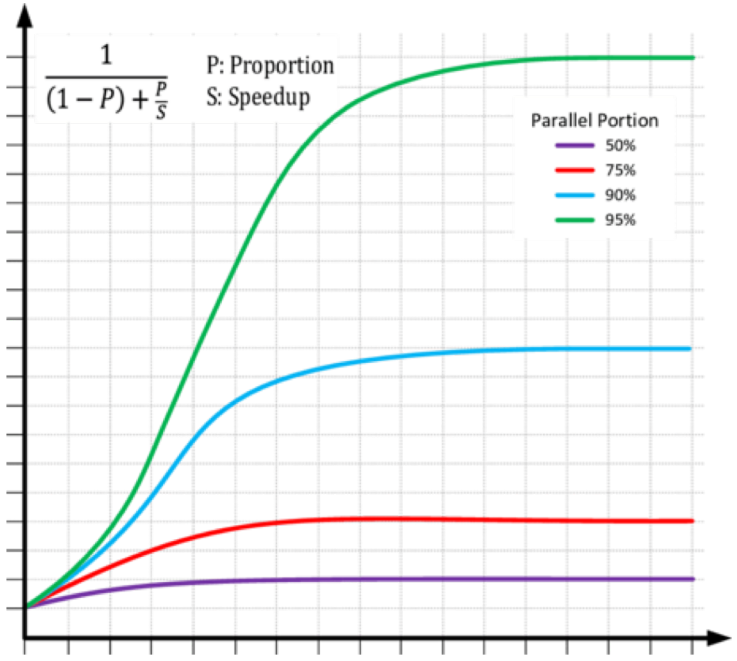


Source: itrs

arm Research

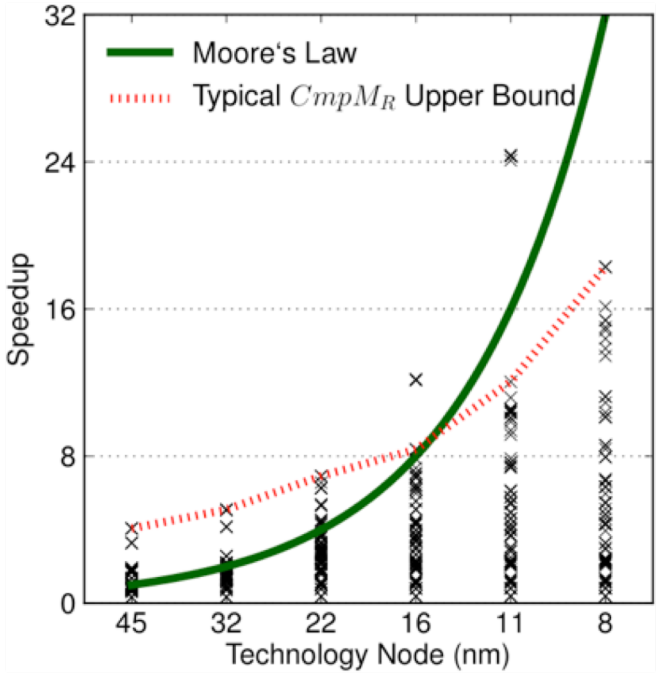
We thought the future was multi- and many-core

Amdahl's Law can't be ignored...



* De Witt et el, Technical Report KU Leuven, July 2013.

... embarassingly parallel code is still rare

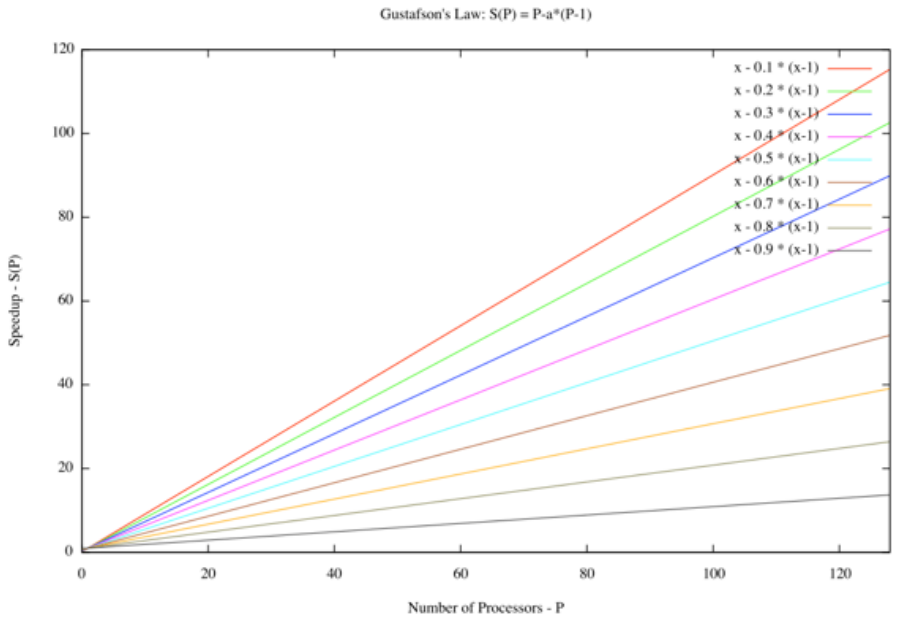


* Eseilzadeh et el, ISCA'11

arm Research

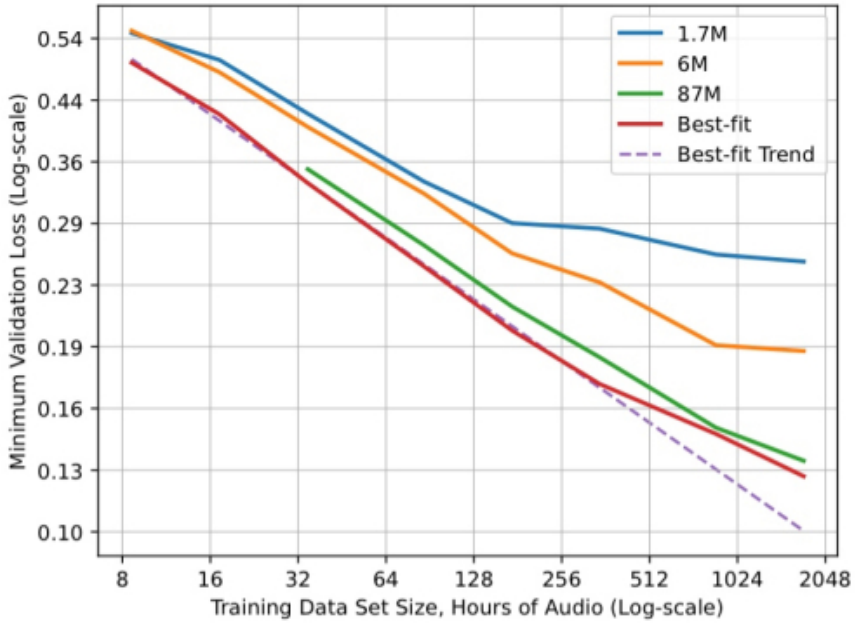
... and data is on our side, but how to process it?

Don’t forget Gustafson’s law



CC3.0 Peahihawaii

... and data hungry models

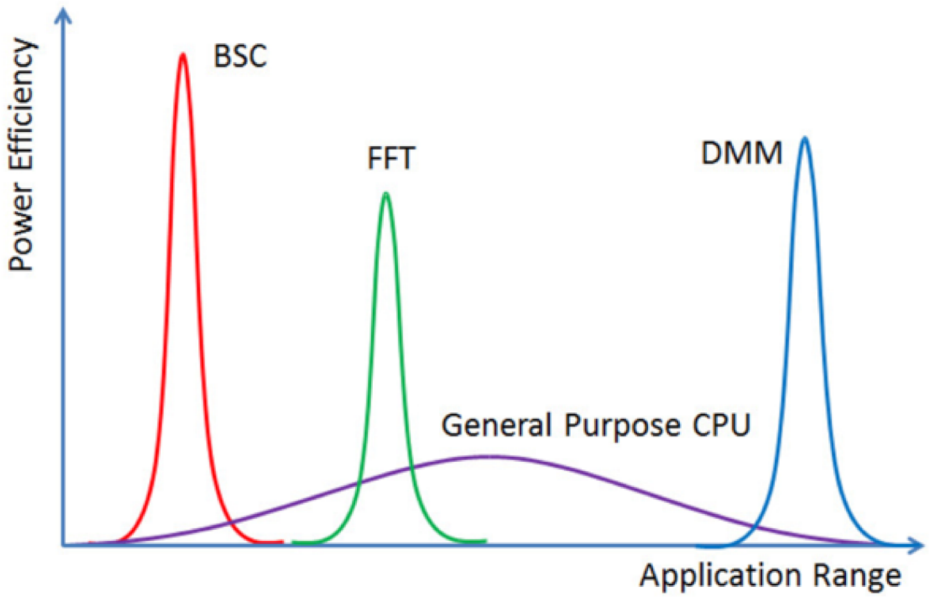


*Deep learning scaling is predictable, empirically Hestness et al., *arXiv, Dec.2017*

arm Research

... specialization is back in fashion

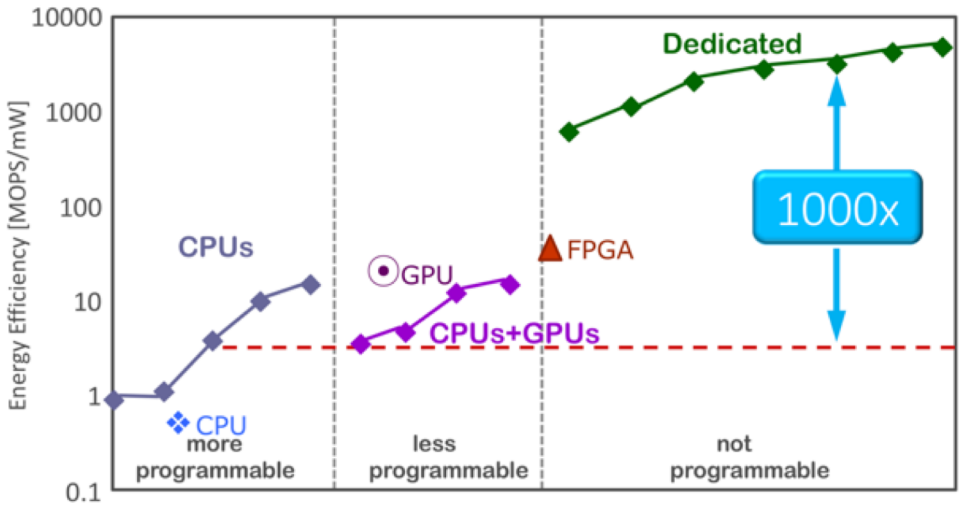
But not all the boats rise with the tide...



MultiAmdahl: Optimal Resource Allocation in Heterogeneous Architectures

Yavits et. al – IEEE Comp. Arch. Letters. 13 (1) pg. 37-40.

... and gains come with loss of flexibility



Extracted from: Scaling Machine Learning Performance with Moore's Law.

Olukotun, Cadence Embedded NN Conference, Feb 2017.

arm Research

Specialization in multiple forms

- **Domain** specialization is compute bound to a given problem domain
  e.g. Neural Networks, Image Signal Processing...

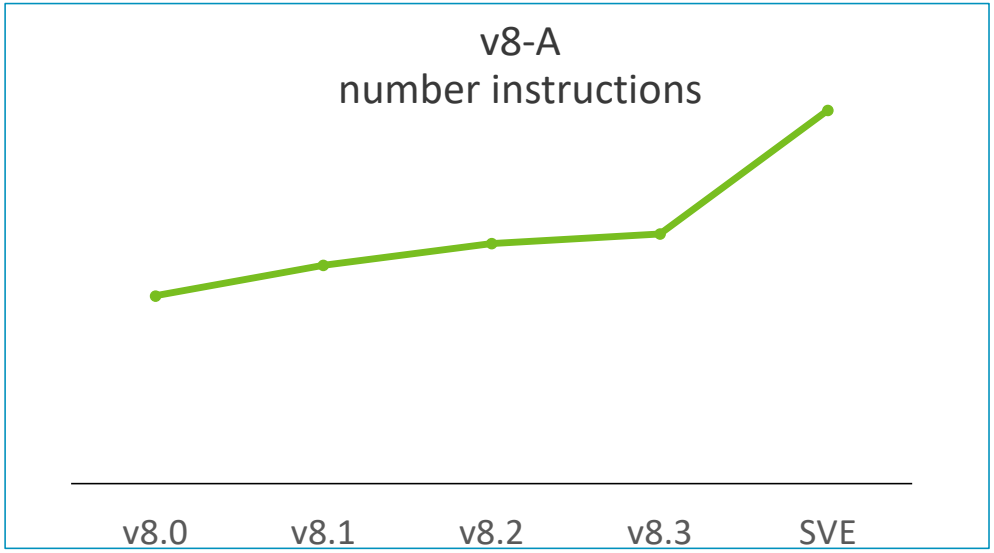
- **Structural** specialization is compute bound to common structures of work (see image)
  e.g. Loop accelerators, Pointer chasing...

- Interfacing, integrating and connecting specialized compute is key to realizing its benefits.

Specialization is a spectrum

Revisit ideas outpaced by Technology Improvements?

Approximate Computing by proxy



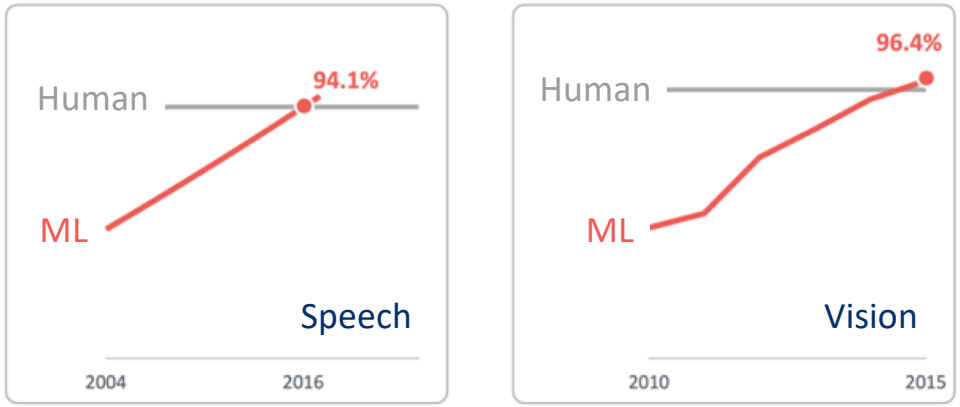
Data types	ISA	TC accelerators	Heterogeneity	LC offload engines	systems
FP16 int8 Bfloat HPA	AES/SHA MPA Atomics (S)DOT	**CCA** **Co-processor**	Homogeneous ISA Fractal ISA	GPU IPU NPU VPU	Embedded Mobile HPC

arm Research

Aside on Machine Learning

Machine Learning is a type of artificial intelligence that provides computers with the ability to learn without being explicitly programmed.

Useful when we don’t have algorithms, but do have lots of data



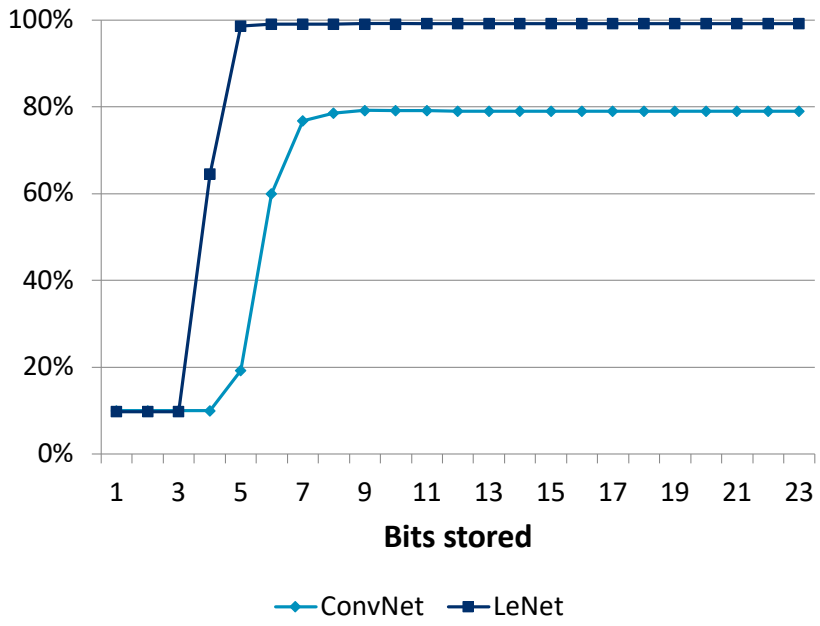
ML is already exceeding human capability

Note: Created by SoftBank Group Corp. based on data by ImageNet, Microsoft and IBM.

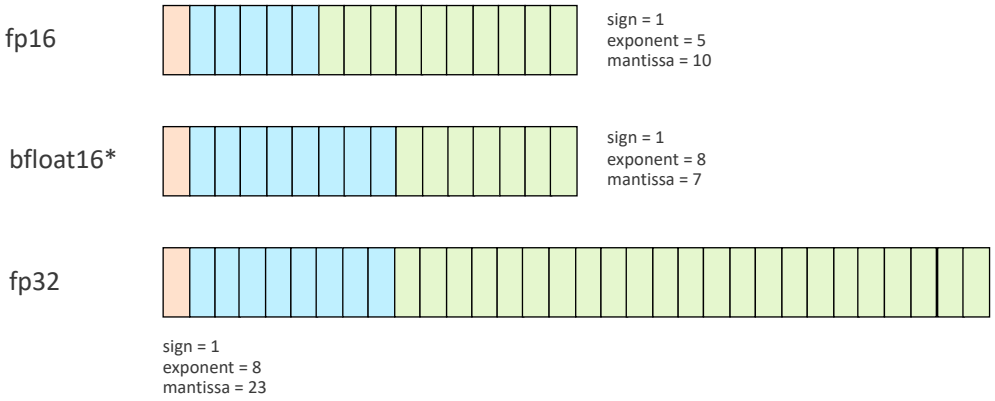
arm Research

Specialization – Data types

In ML precision and range important...



... goldrush of new non-IEEE types

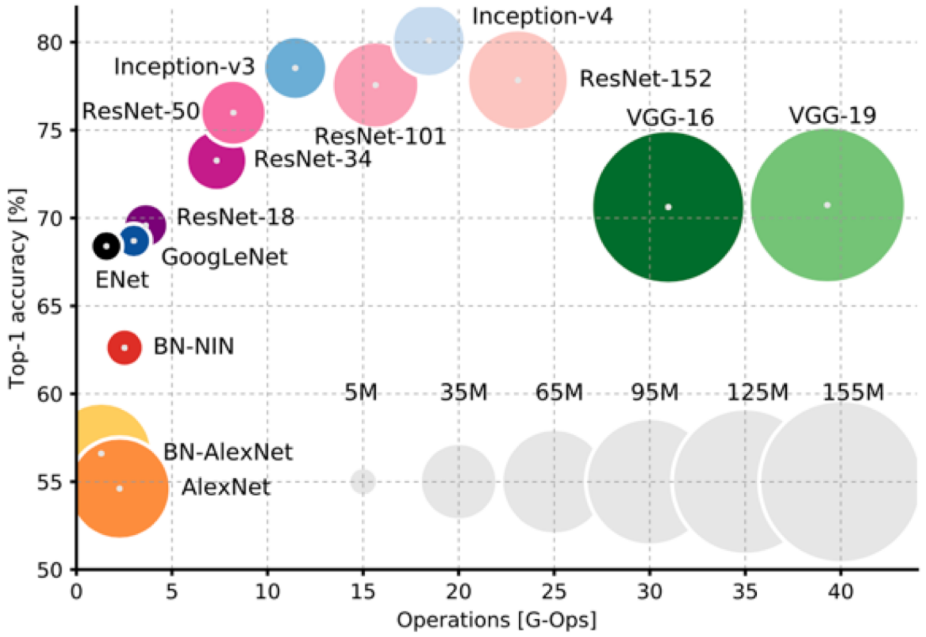


* Support present in some accelerators, no support for this in Arm ISA

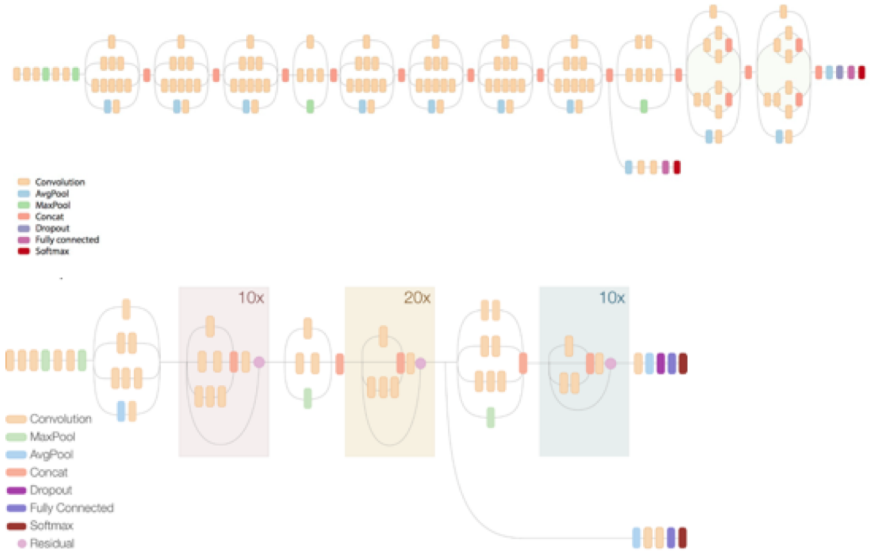
arm Research

Specialization

Complexity trade-offs



2x compute, 2x memory, 1.2% gain..



arm Research

Complexity vs. Accuracy

The difference between a Siberian Husky and an Alaskan Malamute

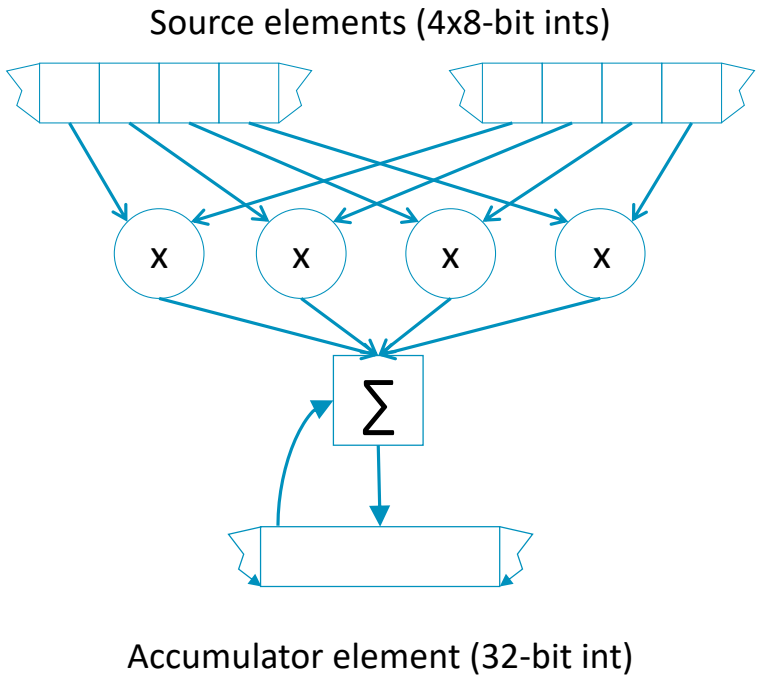
Images from Wikipedia.

Philosophically what is accurate enough?

- Humans are fallible

arm Research

Specialization – Dense widening: dot product



GEMMlowp operates on 8-bit integers

- Widely used in mobile inference

Widening within same sized element

- Multiply smaller components and accumulate into 4x wider lane (double-widened)
- Performance of int8 rather than int32
  - 32 ops/cycle per 128-bit segment

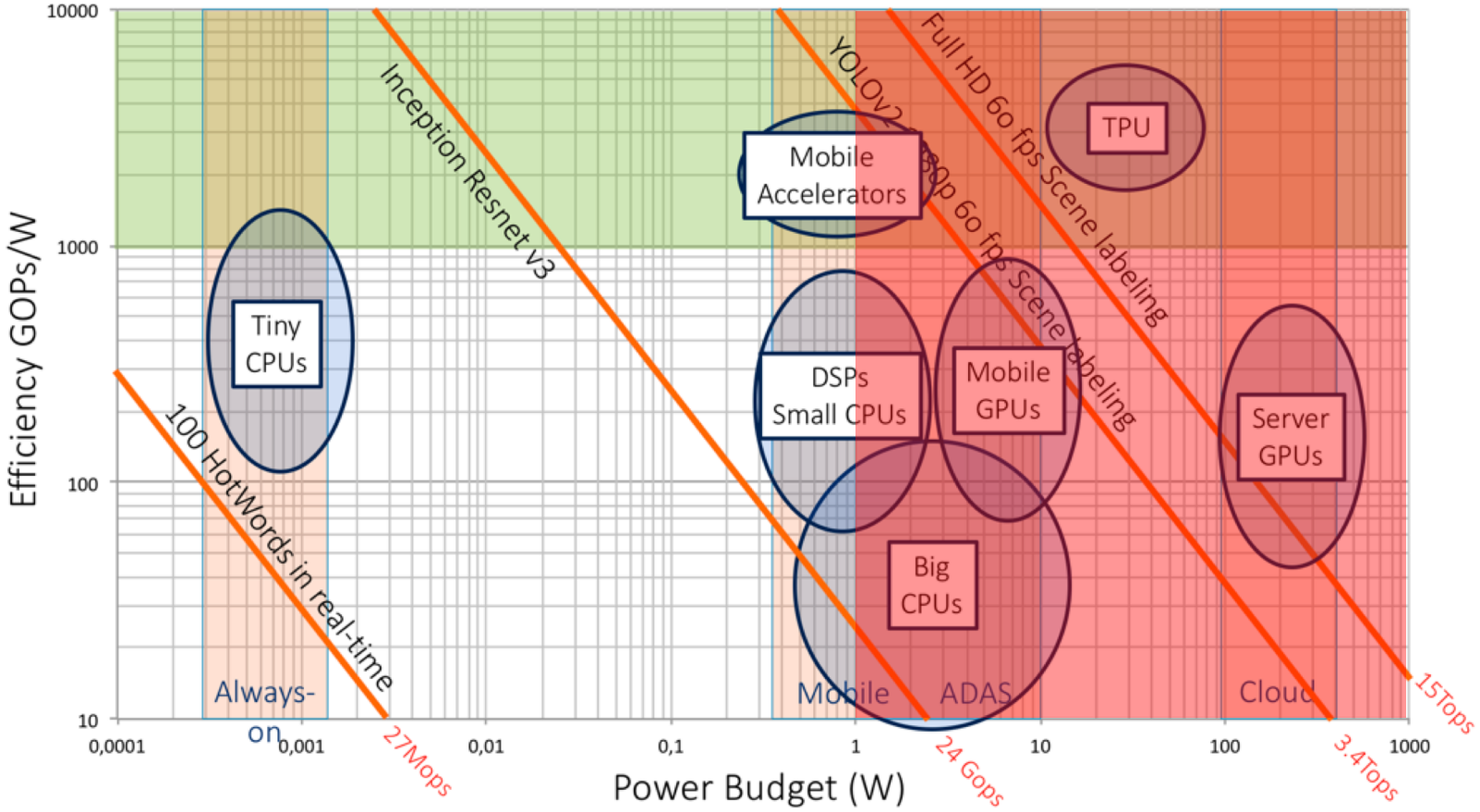
Two variants are helpful

- vector x vector
- vector x indexed-element

UDOT/SDOT introduced in Armv8.2, alongside support for half-precision FP.

arm Research

Specialization - Accelerators



arm Research

Embracing Specialization

- Arm interested in domain specific assist **in the abstract** as well
  - Not just targeting deep algo/app specific research
  - More interested in meta questions it raises such as:
  - What does it take to make it work? (full system of hw/sw)
  - What can be generalized?
  - What are the non-differentiating technical challenges?
  - How do you make deployment easy?

- Will briefly look at two technologies
  - DynamiQ – an Arm cluster with accelerator attach support.
  - ACAI – a collaboration by Arm Research, University of Michigan and Xilinx.

arm Research

ACAI: Arm Coherent Accelerator Interface

Framework for Integrating Hardware Accelerators

Tutu Ajayi ajayi@umich.edu
Balaji Venu balaji.venu@arm.com
Paul Hartke paul.hartke@xilinx.com

DAC 2018 | San Francisco, CA | June 24-28

ACAI: Motivation

- Massive accelerator adoption to keep up with performance demands
- Accelerator benchmark speedups are misleading without proper system integration
- Major challenges with accelerator integration:
  - High engineering cost
    - Hardware design and verification cost
    - Increased complexity in programming models and developing custom software (drivers, libs, etc.)
  - High job dispatch overhead
  - Security, isolation and protection concerns
  - Job scheduling from multiple processes to multiple accelerators

ACAI Framework addresses these challenges

ACAI: Main Idea

- Hardware and software framework to enable easy adoption of accelerators on SoC platforms
- Simpler programming model
  - User application written in C/C++ runs on CPU - easier to debug & modify
  - ACAI software libraries and drivers assist with job creation, scheduling and dispatch
  - Support for accelerator virtualization (sharing across different processes)
- Easier hardware accelerator integration
  - ACAI provides accelerator with cache coherency and virtual addressing capabilities
  - Accelerator interfaces to ACAI using standard AXI protocol
  - Compatible with Xilinx Vivado HLS (High-Level Synthesis)
- User mode job dispatch
  - Maximize performance gains and enable fine-grained task acceleration

Disclaimer: ACAI is a research project in collaboration with Arm. Made available to researchers/partners to understand market needs and collect feedback

arm Research

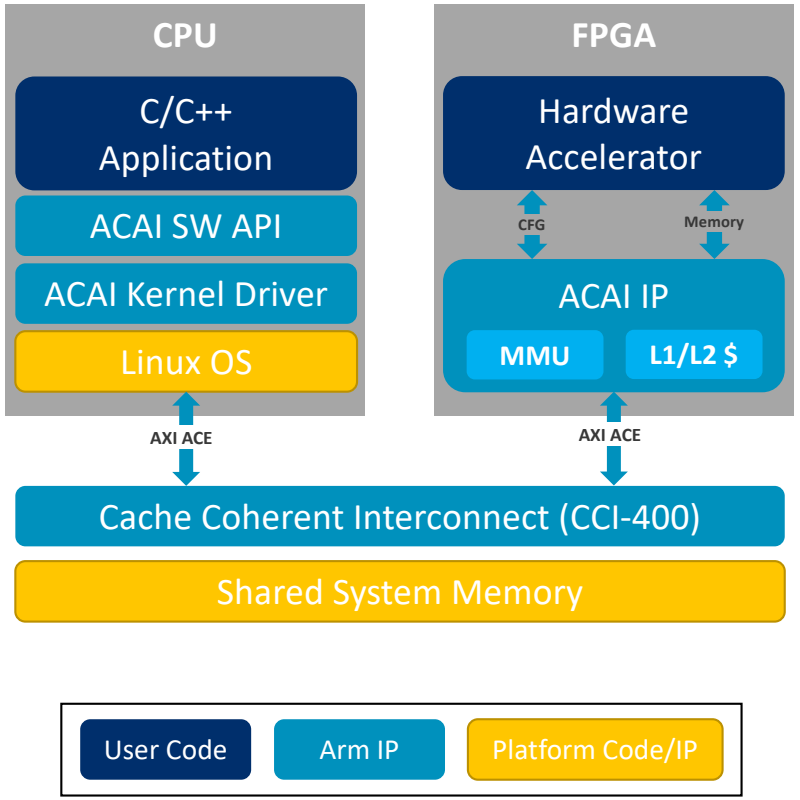
ACAI Framework - Logical Overview

CPU and accelerator are fully coherent and run in the same virtual addressing space

- User application written in C/C++ runs on CPU
- ACAI software libraries assist with job creation, scheduling and dispatch
- ACAI kernel driver sets up page tables and configures ACAI IP
- ACAI IP configures the accelerator, provides a cache and coherent memory interface
- User accelerator executes on user described job

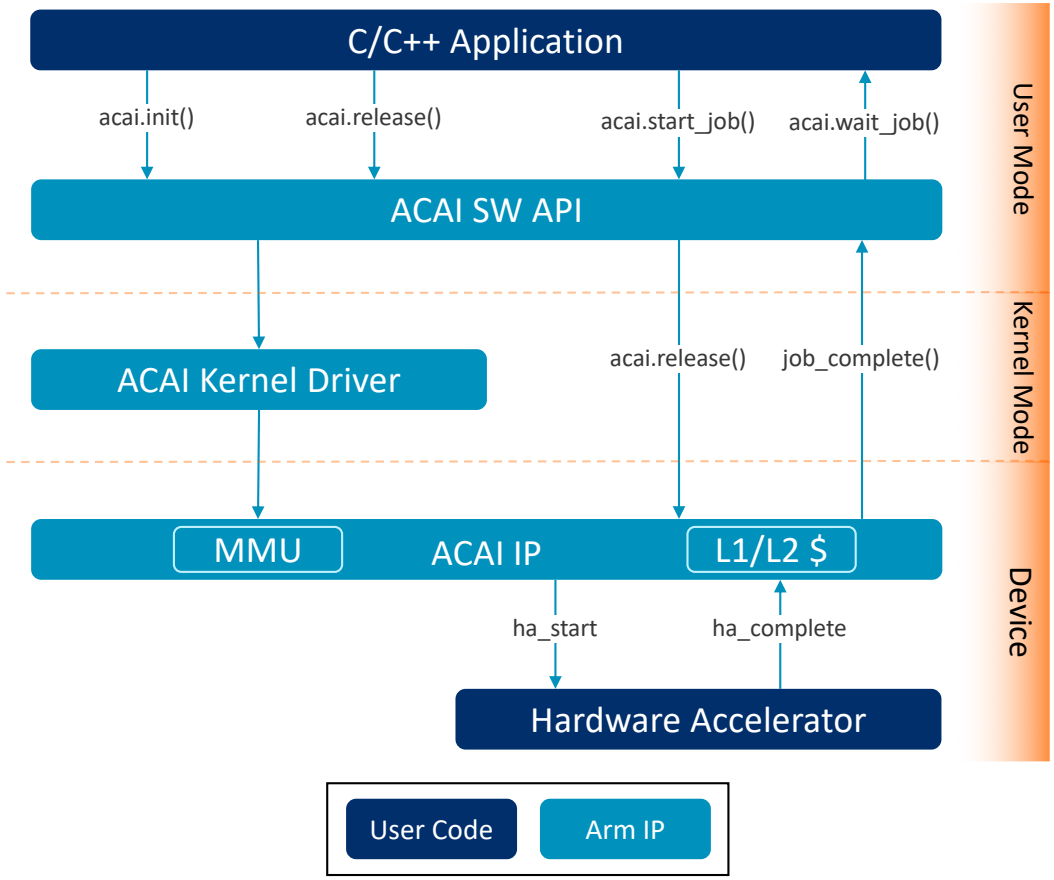
Reduced accelerator dispatch time

- **No explicit data copies**
- **No CPU flush/invalidate required**
- **Supports pointer dereferencing**



arm Research

ACAI Component Interaction



arm Research

ACAI User Code Example

Setup and dispatch an FFT job on ACAI framework

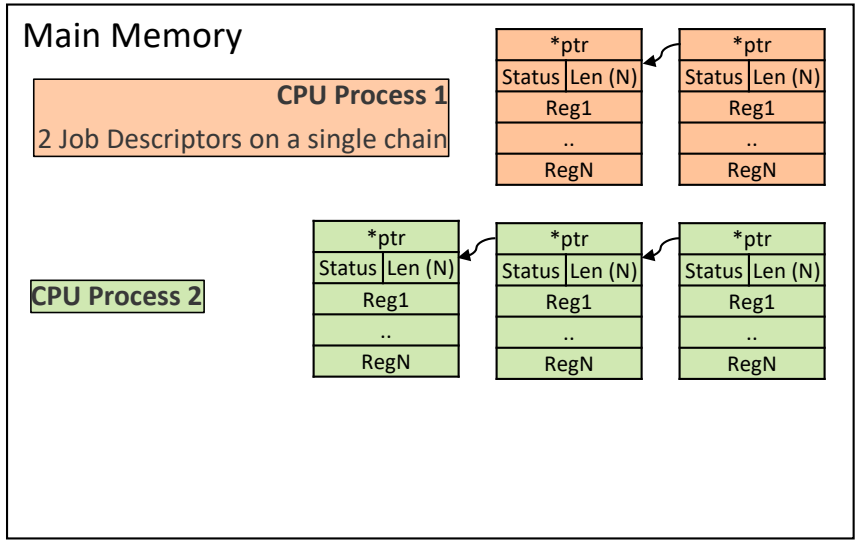
```
void acai_hw_fft() {
  // initialize acai
  acai *p_acai = new acai();
  p_acai->init();

  // setup job chain with a single job
  vector<acaijd> job_chain;
  job_chain.reserve(1);

  // setup job descriptor to write 3 registers
  job_chain.push_back(acaijd(3, 0));
  job_chain[0][0] = (uint32_t)length;
  job_chain[0][1] = (uint32_t)src_data;
  job_chain[0][2] = (uint32_t)result_data;

  // start and wait on the job to complete
  p_acai->start_job(job_chain[0]);
  p_acai->wait_job(job_chain[0]);

  // cpu reads results
  p_acai->release();
};
```

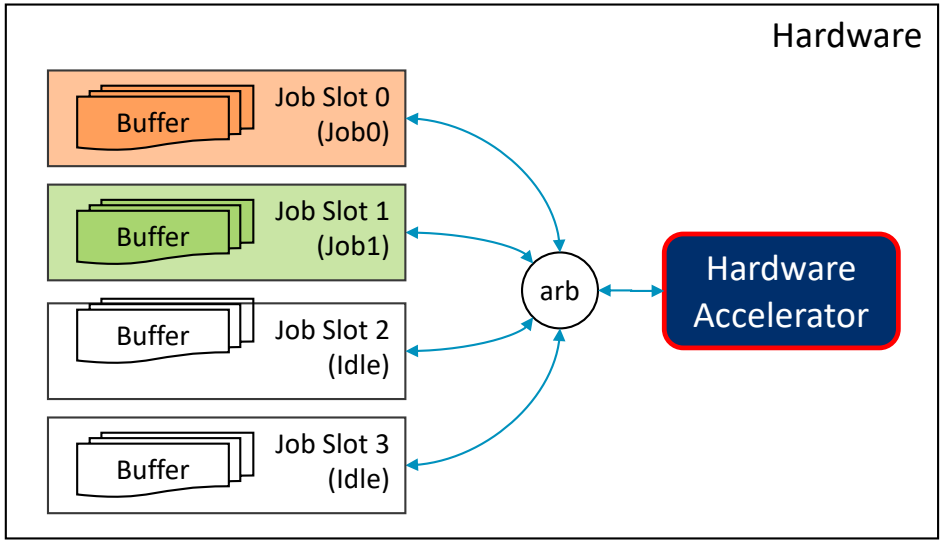
Hardware Accelerator (Verilog)

```
module ha_fft (
  input  wire clk,
  input  wire reset_n,

  // Memory access interface (AXI4 Master)
  // ..

  // Register Configuration interface (AXI4-Lite Slave)
  // ..

  input  wire ai_job_start_i,
  output wire ai_job_complete_o
);
  reg [31:0] length;
  reg [31:0] src_addr;
  reg [31:0] result_addr;

  // more code

endmodule // ha_fft
```


arm Research

ACAI Job Scheduling



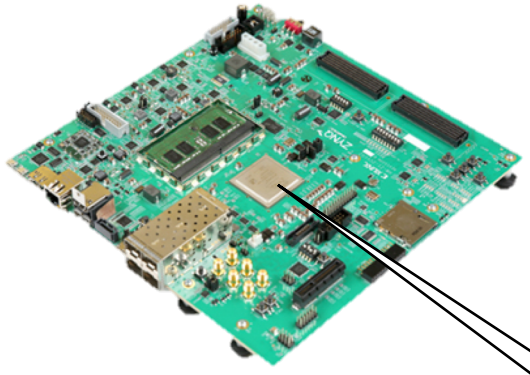


- Framework Schedules/Arbitrates requests from several threads to the same accelerator

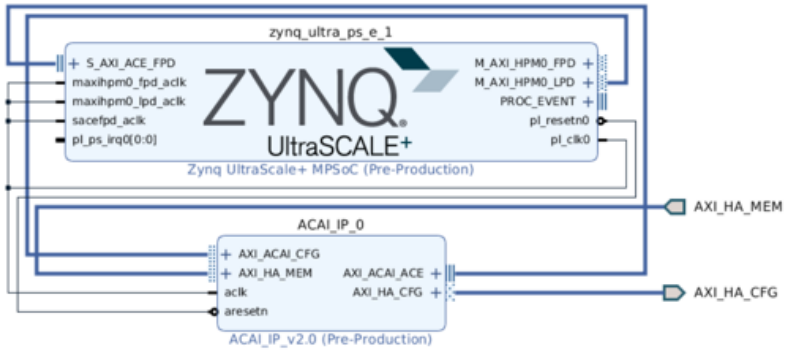
arm Research

Xilinx Prototype Platform

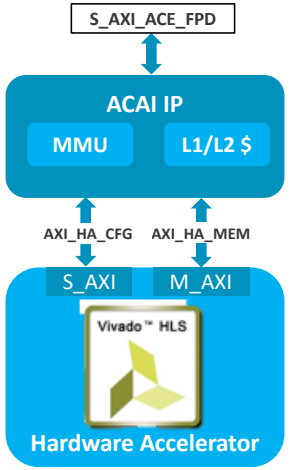
Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit

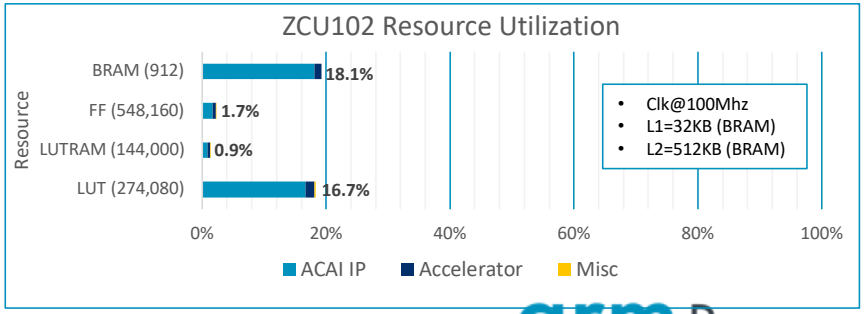


Vivado Block Diagram - IP Usage



Supports Vivado HLS (High Level Synthesis)



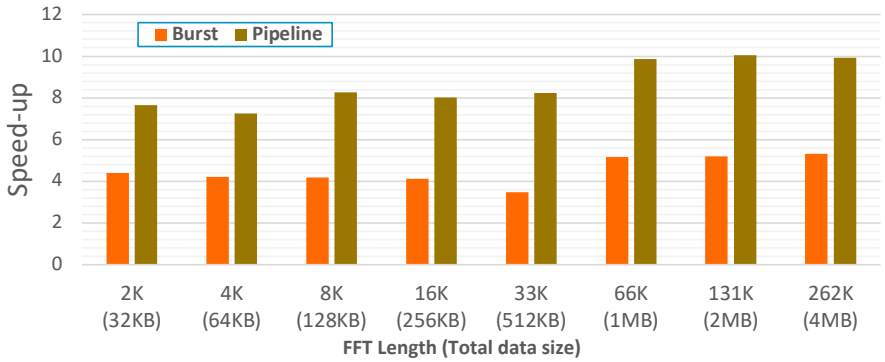


arm Research

Benchmark Results

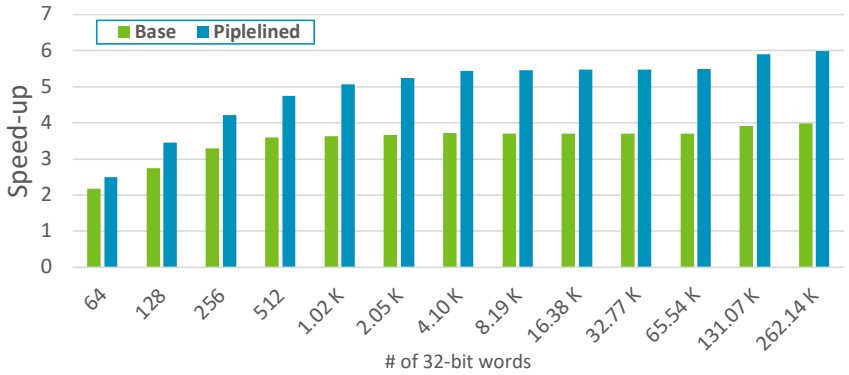
Hardware accelerator speedup over software implementations on CPU (A53 – single-core)

FFT Accelerator



- CPU implementation uses FFTW software libraries and Floating Point Unit (FPU)
- FFT (1D-FFT single precision complex numbers) hardware accelerator implemented using Xilinx LogiCORE IP

RGB->YUV Accelerator

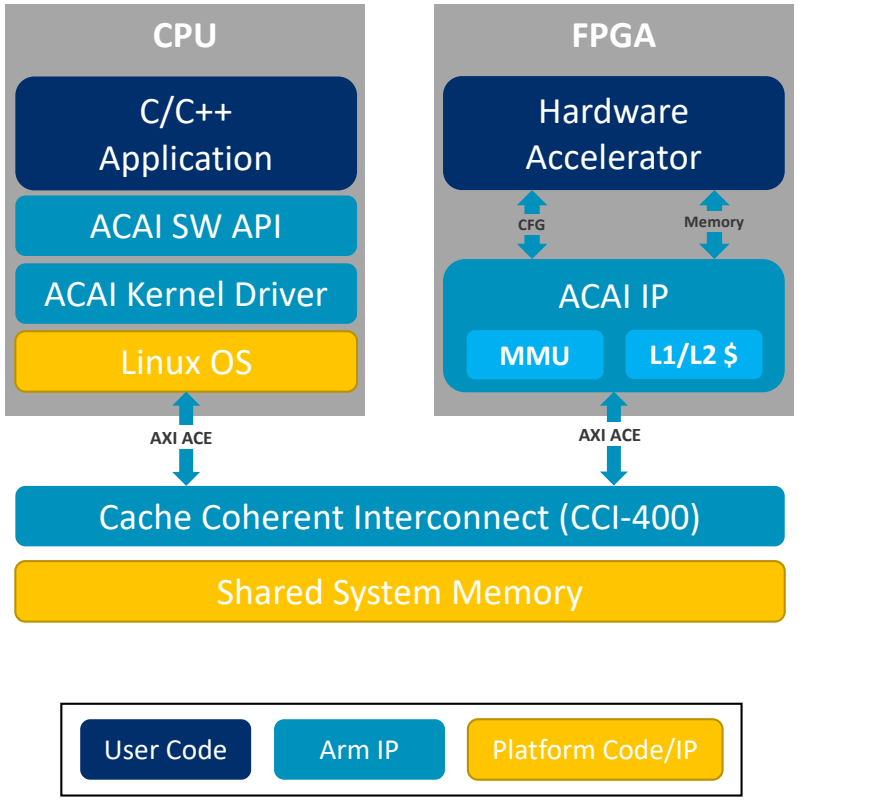


- Base version
  - Hardware accelerator developed internally
  - Executes in three stages: load, compute and store
- Pipelined version
  - Developed using Vivado HLS
  - Contains no explicit load/store stages
  - Reduced overhead for chained or multiple jobs

arm Research

ACAI: Framework

- ACAI prototyped and benchmarked on Xilinx Zynq UltraScale+ MPSoC (zcu102 evaluation board)
- Integrated FFT and RGB2->YUV accelerator behind ACAI
  - FFT accelerator (4x – 10x)
  - RGB->YUV accelerator (2x – 6x)
- Design Effort:
  - ACAI IP effort - 1.8 man years
  - Integrate an existing accelerator - 2 weeks
  - Design and integrate HLS accelerator - 2 days
- More accelerators are being ported to the framework using Vivado HLS

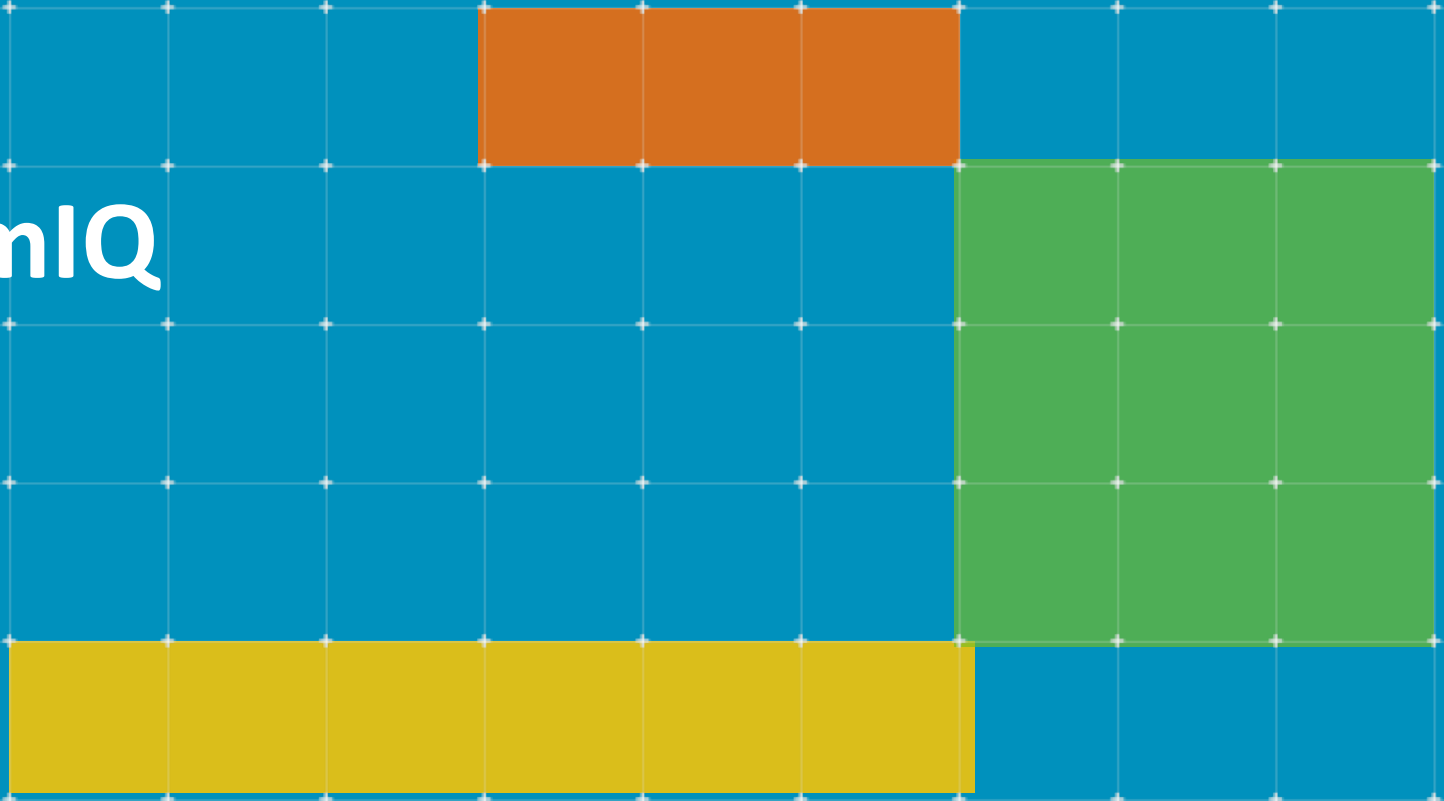


arm Research

Summary

- ACAI presents an easier and more natural programming model
- Faster accelerator integration enables design space exploration and faster time to market
  - Integrate accelerator in days/weeks
  - ACAI in combination with Vivado HLS enables software/application developers to offload jobs onto accelerators easily
- ACAI can be used to model full system performance, implement SoC designs or integrate accelerators on FPGA platforms
- For more information, see us at the poster session or contact acai@arm.com

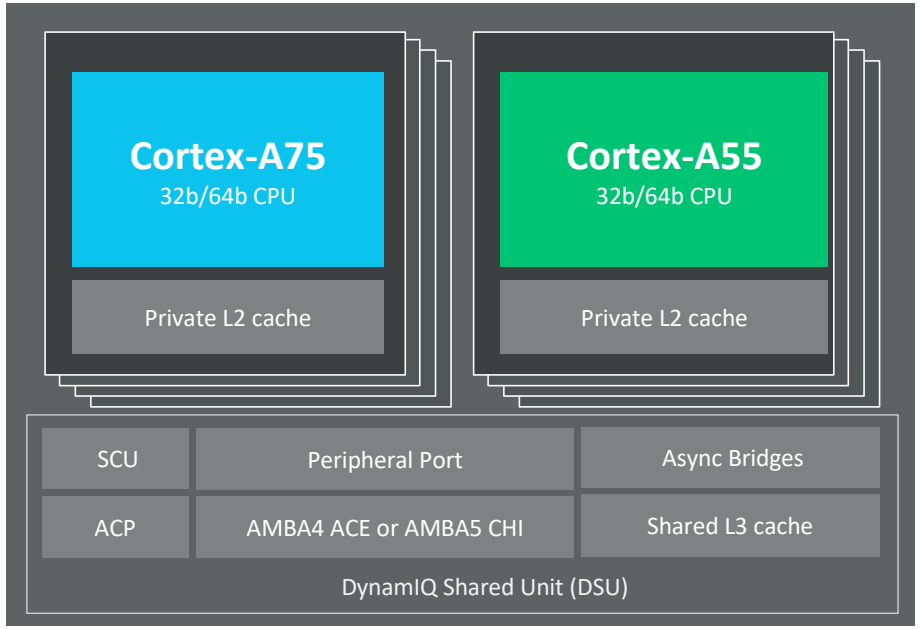
DynamIQ



arm Research

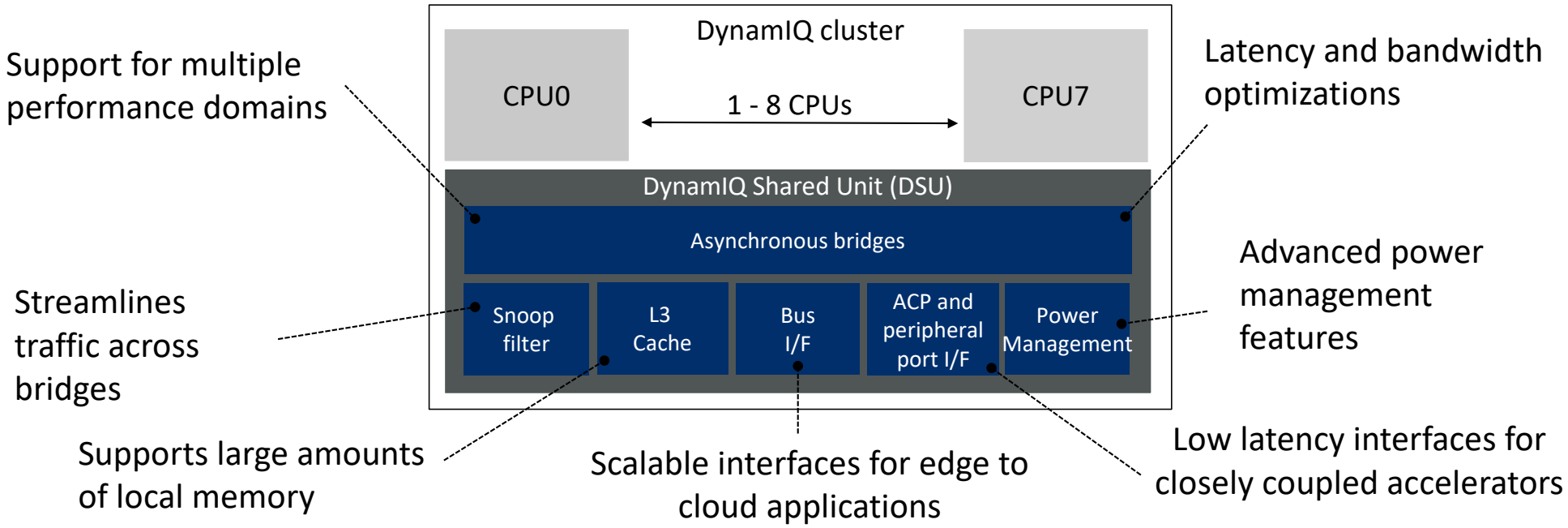
DynamIQ – New Single Cluster Design

- New application processors
  - Cortex-A75 and Cortex-A55 CPUs
  - Built for DynamIQ technology
- Private L2 and shared L3 caches
  - Local cache close to processors
  - L3 cache shared between all cores
- DynamIQ shared unit
  - Contains L3, Snoop Control Unit (SCU), and all cluster interfaces



arm Research

DynamIQ Shared Unit (DSU)

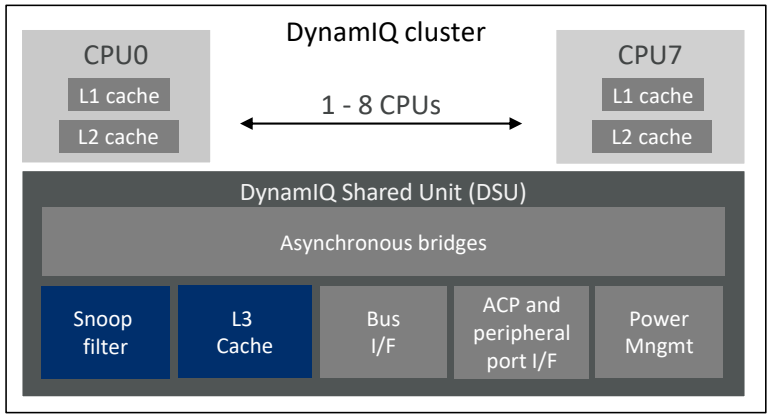


arm Research

Level 3 cache memory system

New memory system for Cortex-A clusters

- 16-way set associative
- Configurable from 0KB to 4MB in size
- Up to 3.5x higher outstanding transactions



- L3 memory system can be partitioned
  - Reduces the affects of cache thrashing in certain types of applications
  - Important for markets such as infrastructure and automotive
- Integrated snoop filter to improve efficiency
  - Capable of back invalidates and is sized automatically

arm Research

Innovations to increase throughput

Cache stashing enables reads/writes into the shared L3 cache or per-core L2 cache

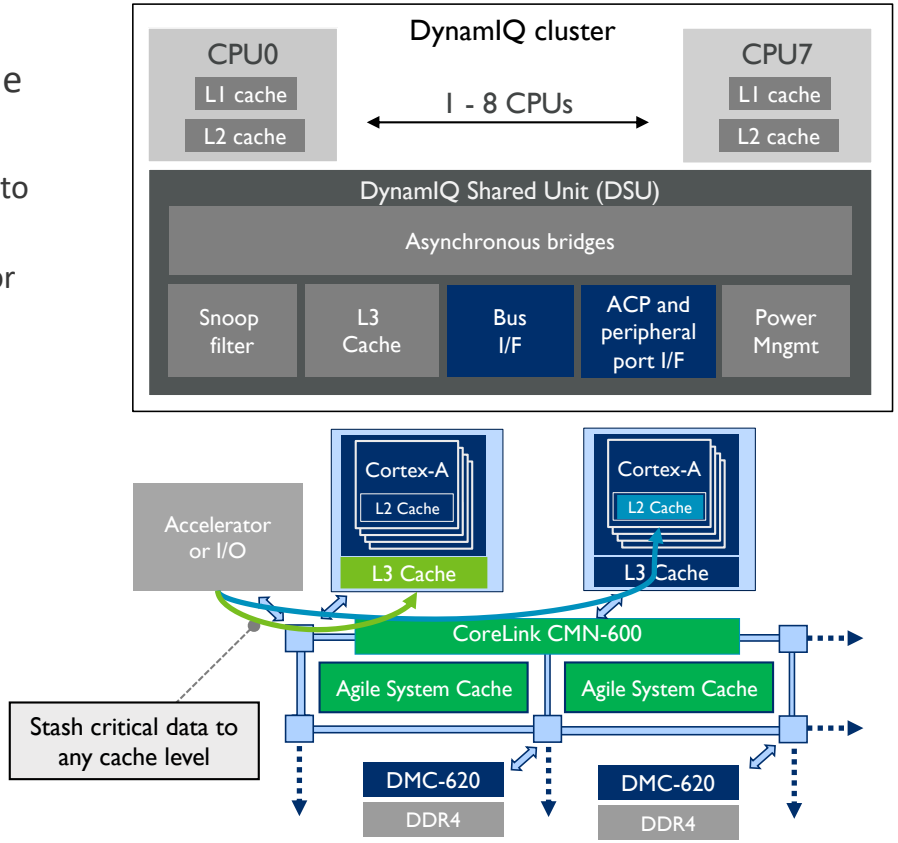
- Allows closely coupled accelerators and I/O agents to gain access to CPU memory
- AMBA 5 CHI and Accelerator Coherency Port (ACP) can be used for cache stashing

More throughput with cache stashing

- In combination with the Peripheral Port (PP)
- Used for acceleration, network, storage use-cases

Critical data stays on-chip

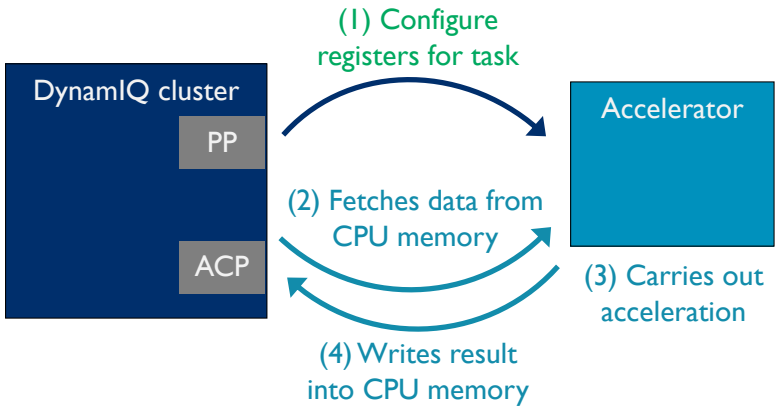
- Low latency access by a CPU or group of CPUs
- Accelerator or I/O selects data placement



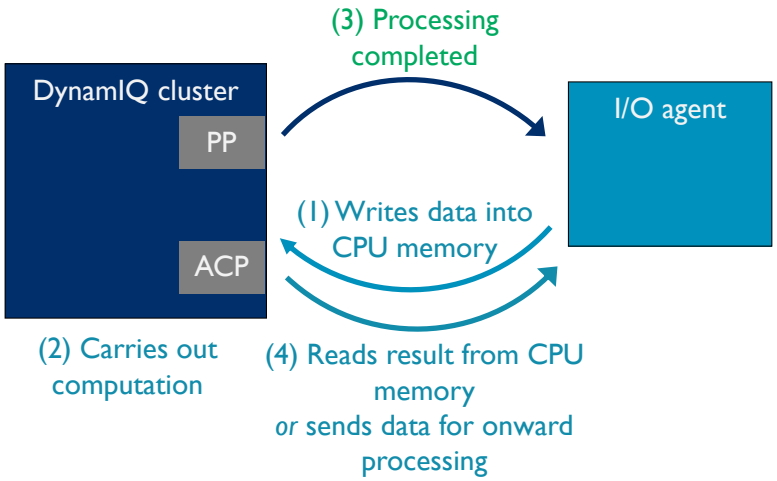
arm Research

Offload acceleration and I/O processing

Offload acceleration

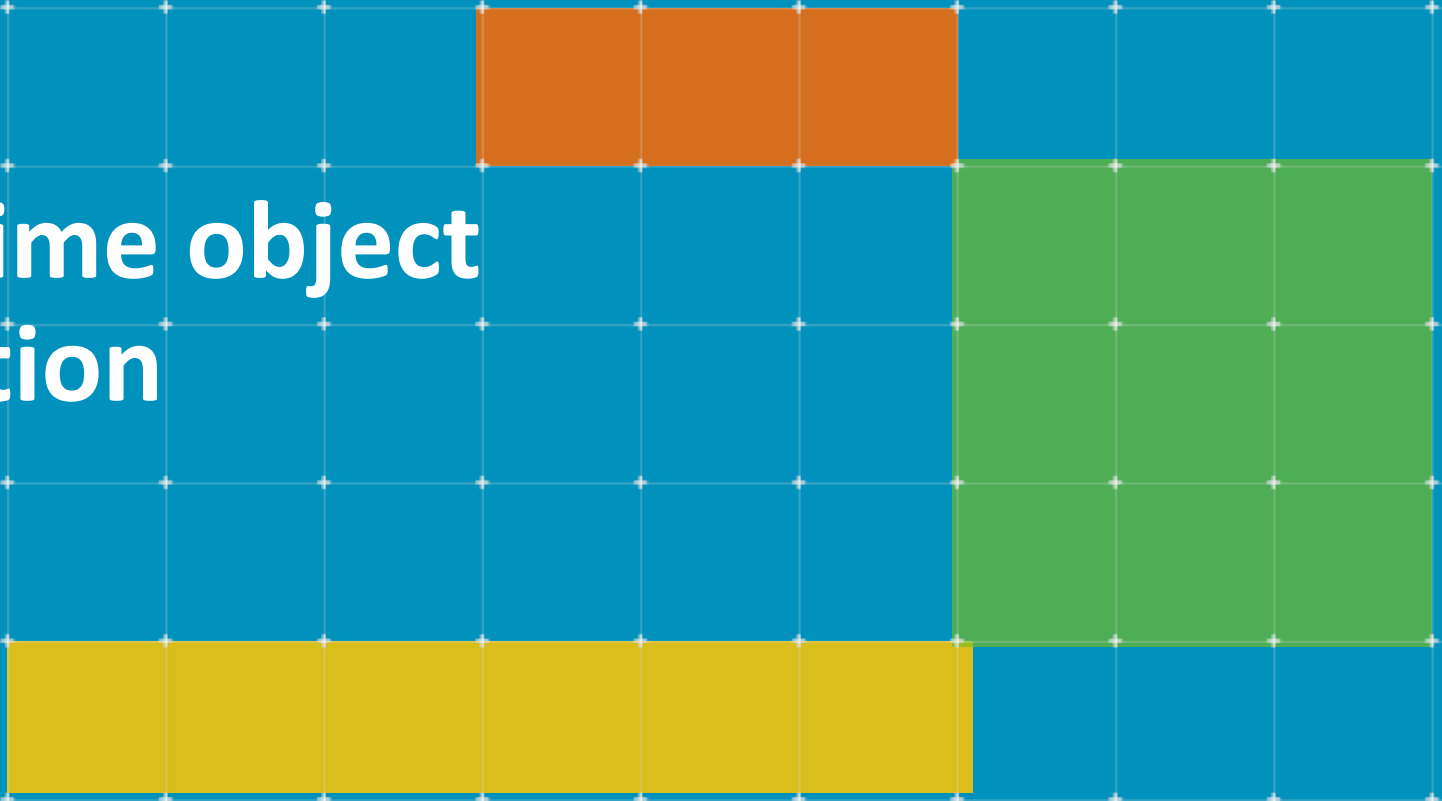


I/O processing



arm Research

Real time object detection



arm Research

Real time object detection and tracking

Best known approach is currently Covolutional Neural Networks (CNNs).

”YOLO” CNN architecture – can perform object detection in images in real-time.

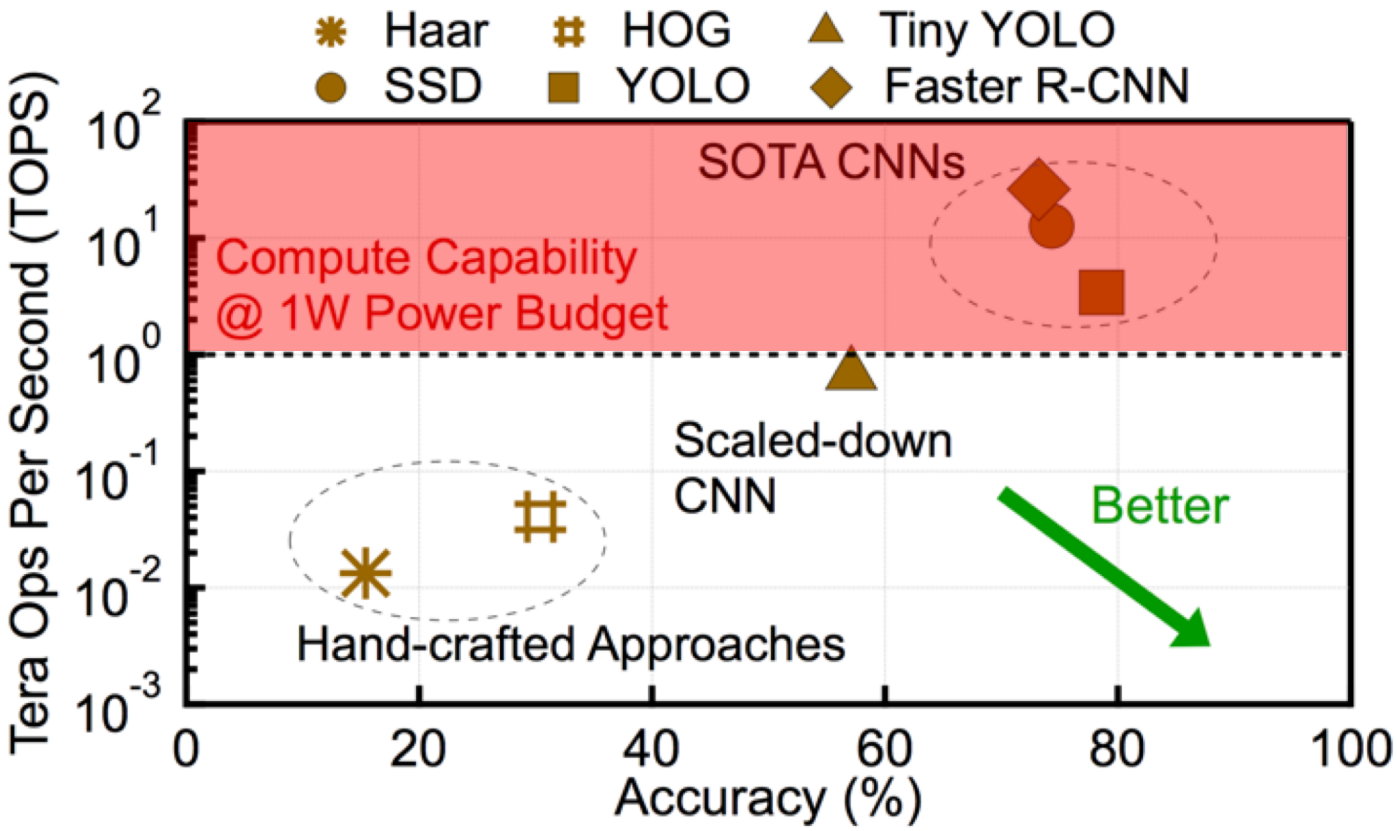
- CNNs require large amounts of compute and memory.
- Real-time object detection difficult on mobile/IoT devices with contrained power and memory budgets
- Bandwidth, Power, Cost, Latency, Reliability and Privacy all motivating RT OD on mobile devices.

Euphrates: Algorithm-SoC Co-Design for Low-Power Mobile Continuous Vision

Yuhao Zhu[1], Anand Samajdar[2], Matthew Mattina[3], and Paul Whatmough[4]
[1]University of Rochester, [2]Georgia Institute of Technology, [3]Arm Research

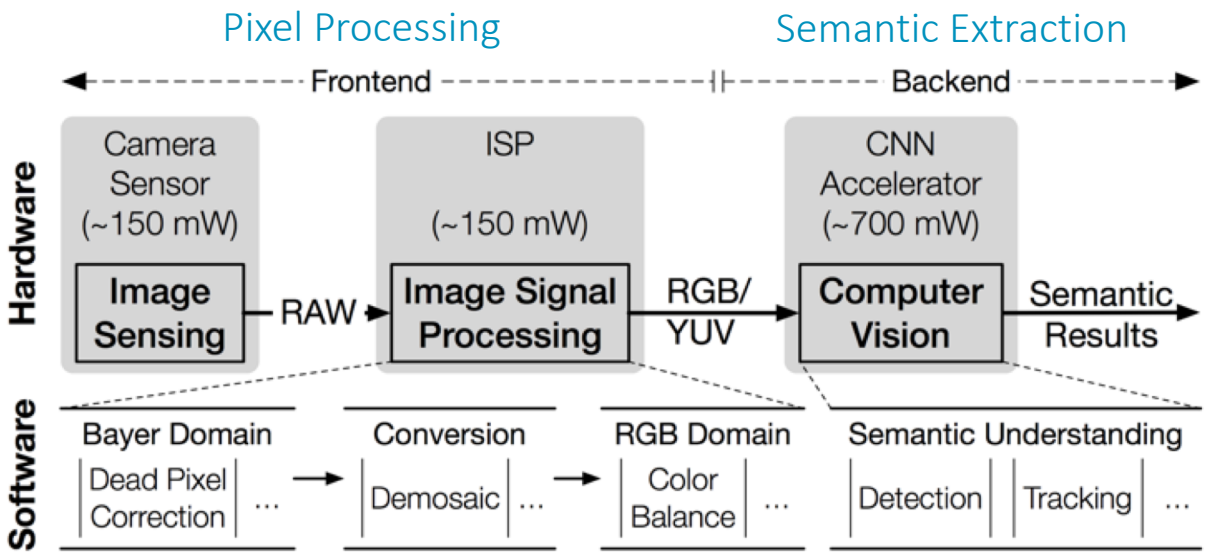
arm Research

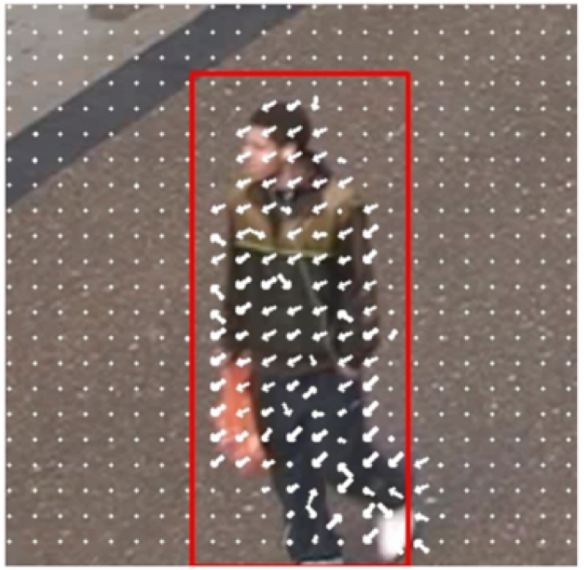
Specialization – Algorithms



arm Research

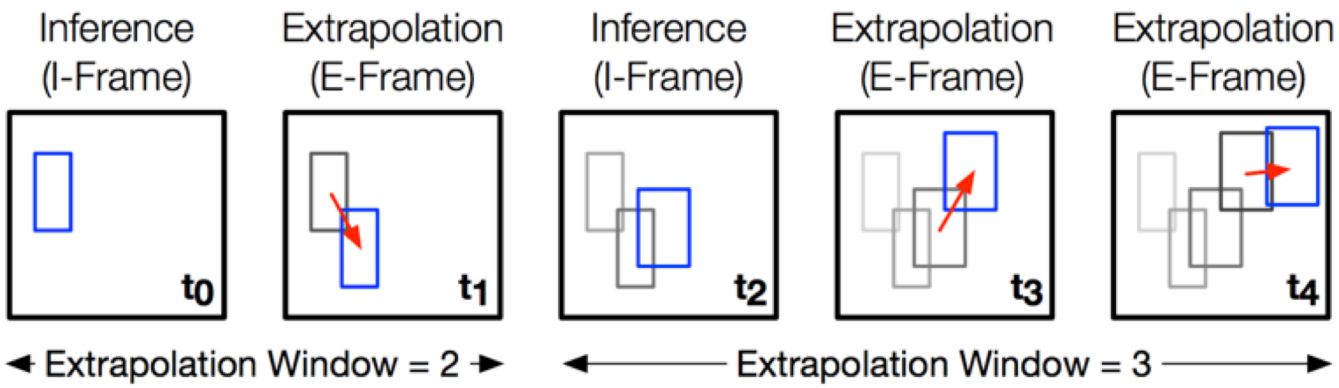
End-to-End Continuous CV Pipeline

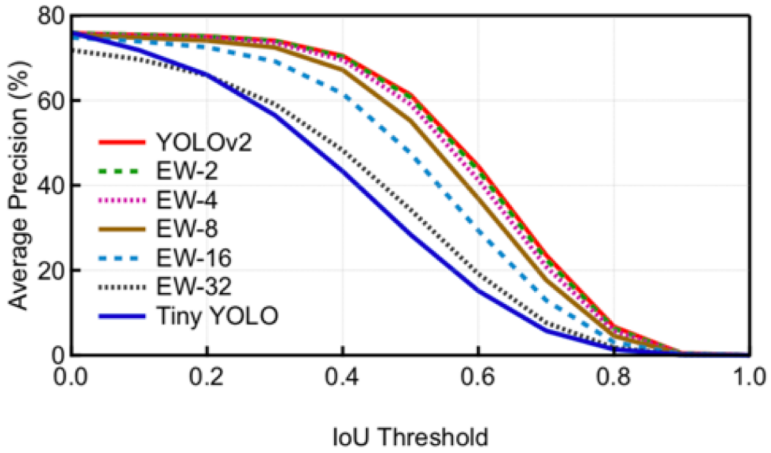


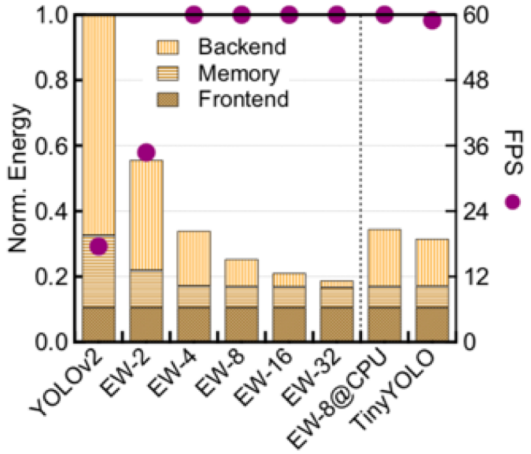


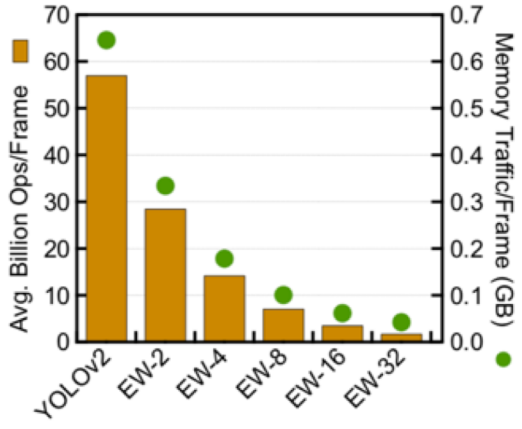
arm Research

Exploiting Motion to Reduce OD and Tracking Power



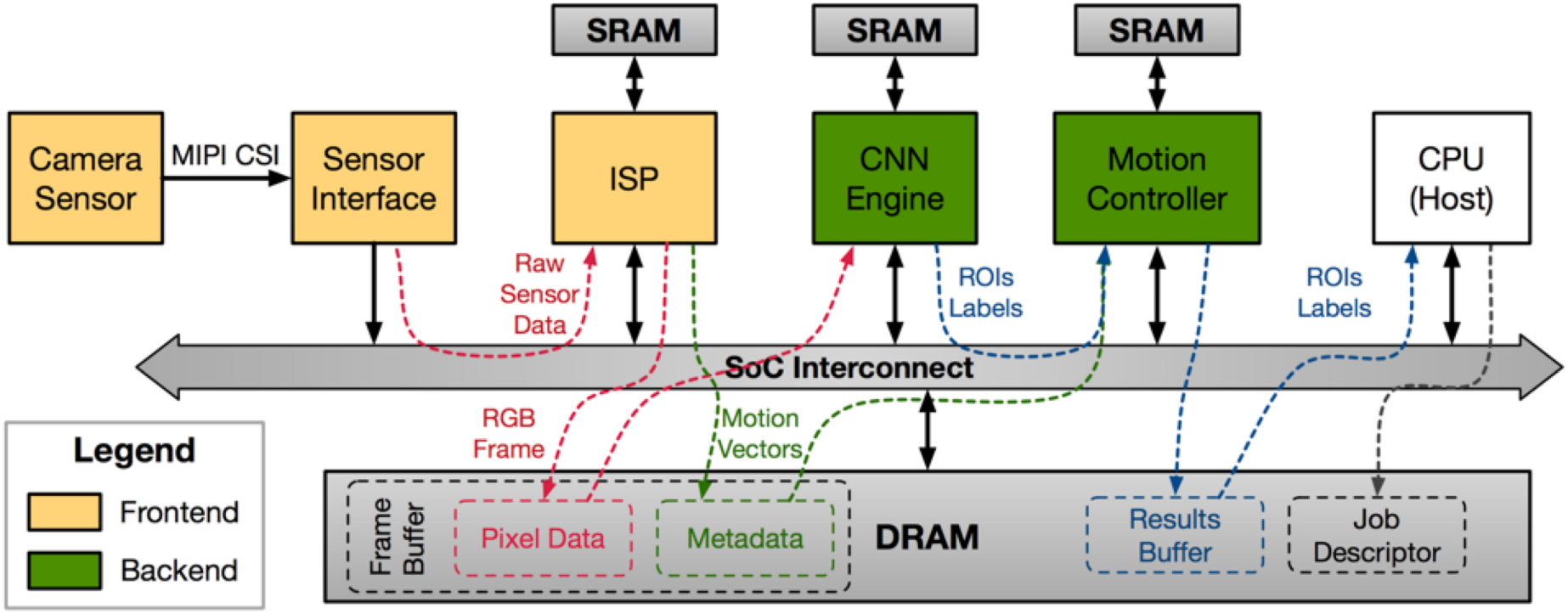






arm Research

Algorithm-SoC Co-Design



arm Research

SCALE-Sim – NN Accelerator Simulator

NN accelerator research requires accurate modelling

- Power consumption, Latency, Circuit Area, Memory Bandwidth

SCALE-Sim was a collaboration with PhD student Ananda Samajdar (Georgia Tech)

- Allows anyone to easily generate metrics for any CNN model in Tensorflow
- Open-sourced, on github, MIT License.
- https://github.com/ARM-software/SCALE-Sim

Some take-aways from this work

- Overcoming the energy-efficiency barrier required expansion of the research horizon from individual accelerator optimizations to a holistic co-design of multiple SoC components.
- Looking forward, as SoCs incorporate more specialized domain-specific IPs, exploiting the synergies across them becomes ever more important.

Thank You!
Danke!
Merci!
谢谢!
ありがとう!
Gracias!
Kiitos!

matt.horsnell@arm.com