Rethinking Systems Software for **Emerging** Data Center **Hardware**

Antonio Barbalace

THE UNIVERSITY OF EDINBURGH

1 June 2023

Antonio Barbalace - Bio

- **The University of Edinburgh, Scotland (9/2019 – present)**
  - **Senior Lecturer at the School of Informatics**
- Stevens Institute of Technology, NJ (2018/8 – 2019/12)
  - Assistant Professor in Computer Science
- *Huawei German Research Center, Germany (2016/9 – 2018/7)*
  - *Principal Research Scientist and Manager*
- Virginia Tech, VA (2011/11 – 2016/8)
  - Postdoc in Computer Engineering
  - Research Assistant Professor in Computer Engineering
- University of Padova, Italy (2002/9 – 2011/10)
  - BS/MS in Computer Engineering
  - PhD in Industrial Engineering (Nuclear Fusion)
  - Research Staff Member (CNR)

icsa Institute for Computing Systems Architecture

https://barbalace.it/antonio/

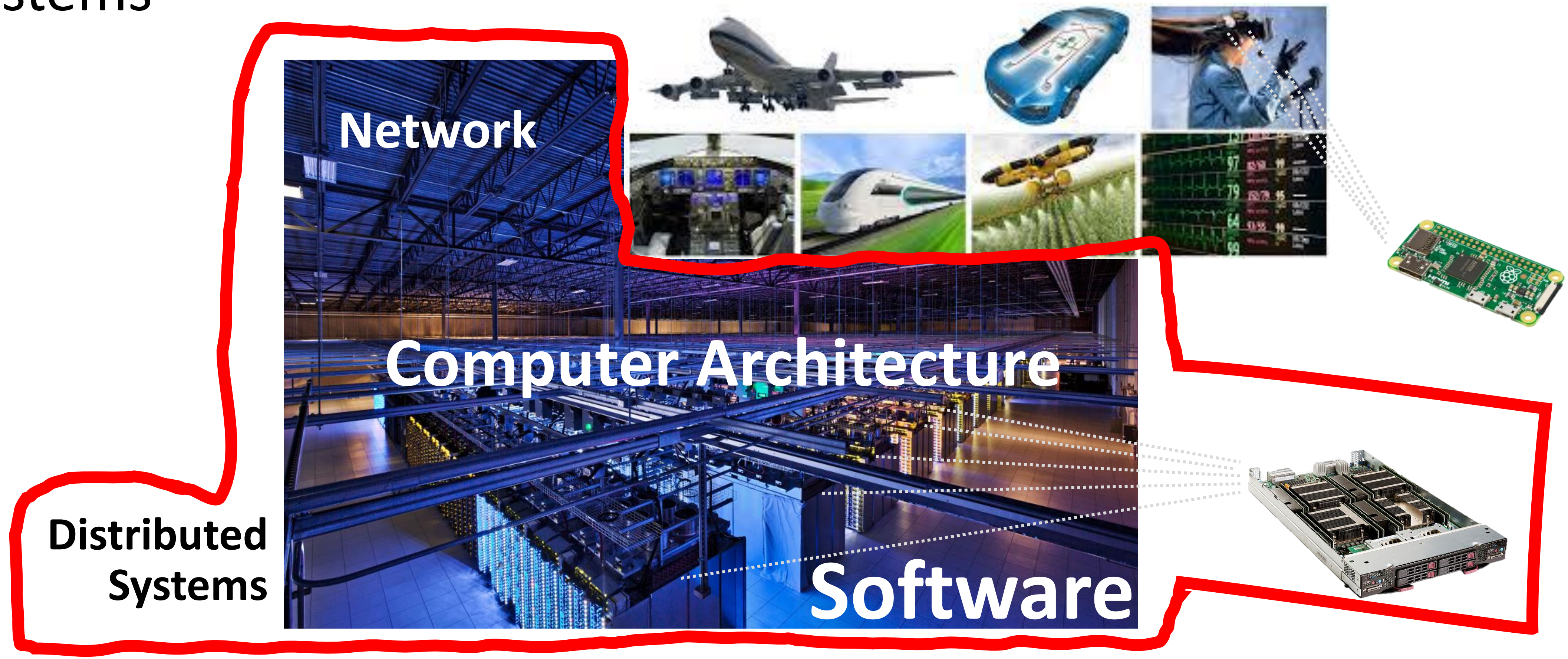
abarbala@ed.ac.uk

THE UNIVERSITY OF EDINBURGH

Antonio Barbalace – Research Interests

- Systems

Real-time Systems



Network

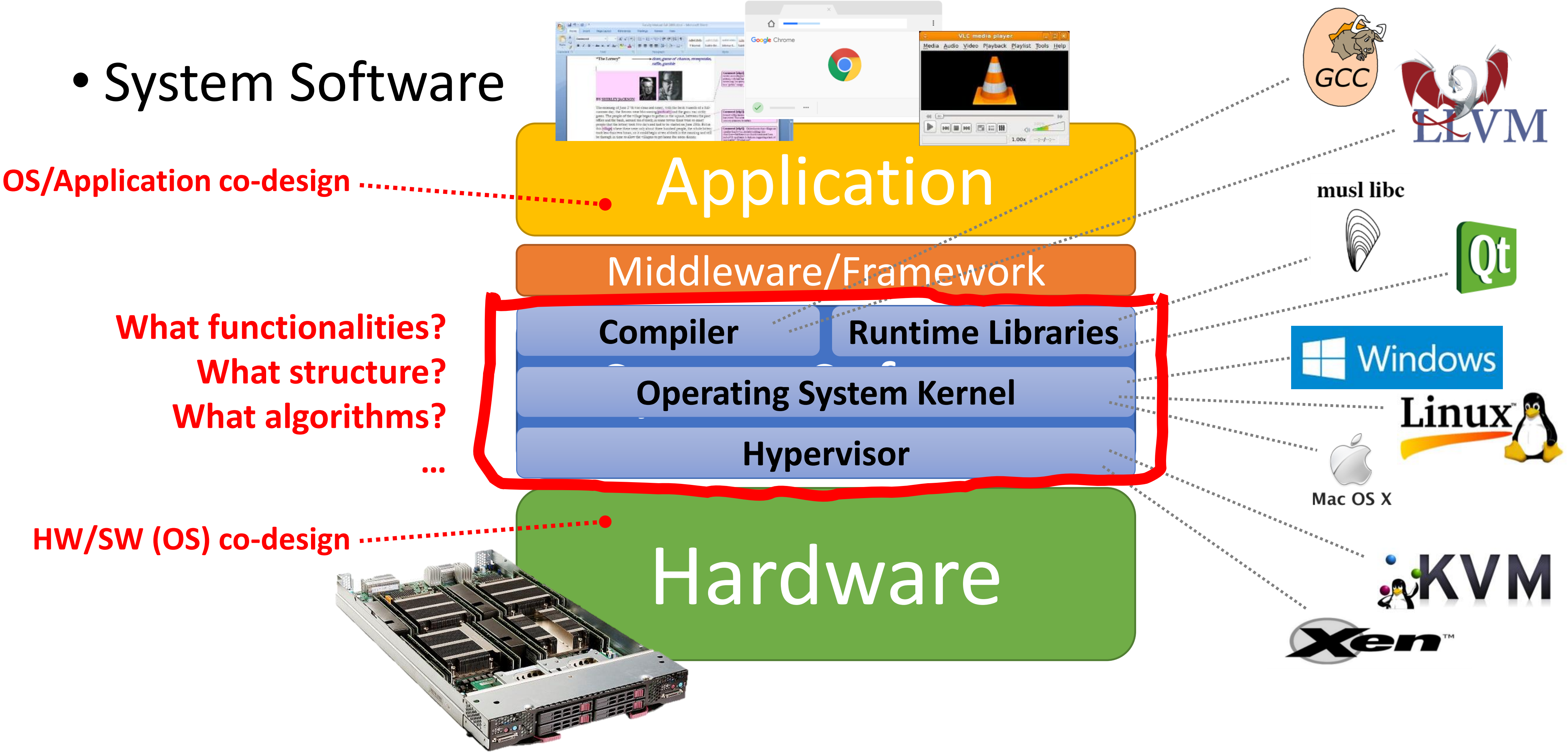
Computer Architecture

Software

Distributed Systems

Antonio Barbalace – Research Focus

- System Software



OS/Application co-design

Application

Middleware/Framework

What functionalities?

What structure?

What algorithms?

...

Compiler

Runtime Libraries

Operating System Kernel

Hypervisor

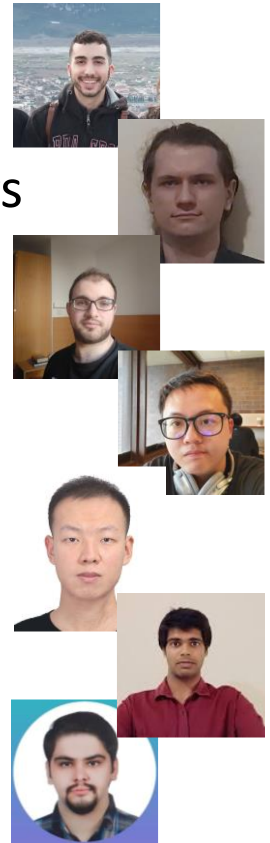
HW/SW (OS) co-design

Hardware

Antonio Barbalace – Research Team

- Postdoc/RA
  - Maxime France-Pillois – FPGA, Runtime, Compiler

- PhD
  - Nikos – Compilers
  - Raven** – Runtime, Compilers
  - Karim – OS
  - Tong – OS, Virtualization
  - Pei – Compilers, Databases
  - Alan – Data Center
  - Amir – OS
  - Xiangyu – Quantum OS



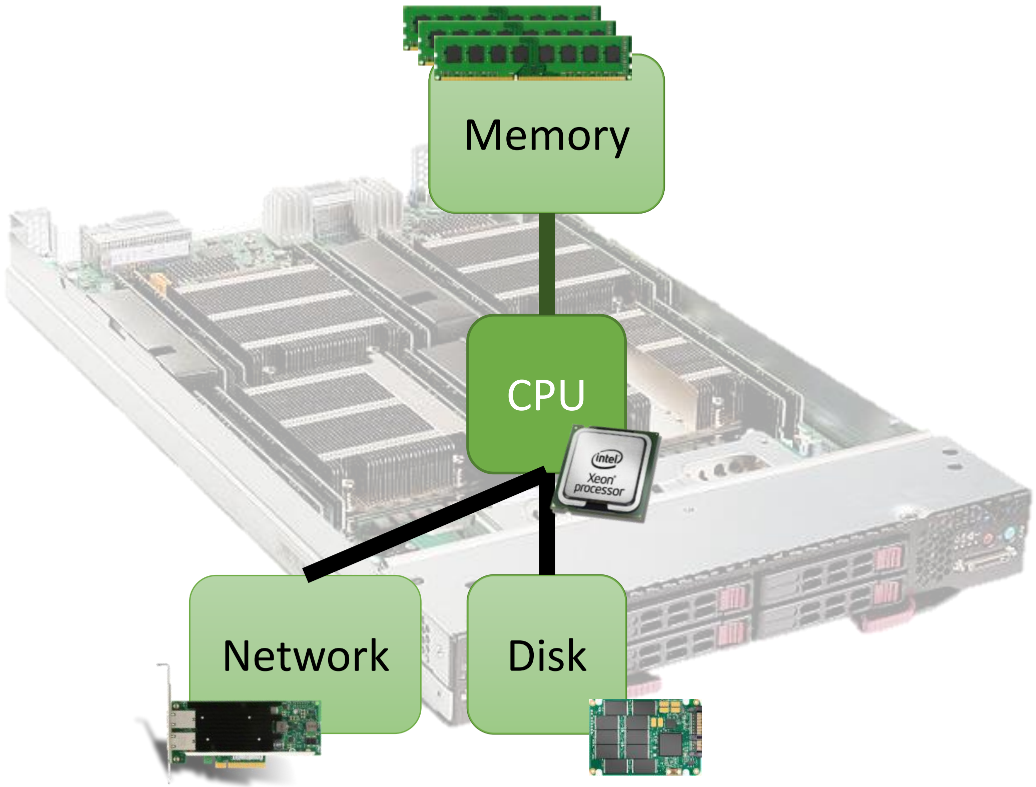
- Minf/Honors UG
  - Utsav Agarwal (Minf)
  - Karoly Lovasz (Minf)
  - Dale Huang (Minf)
  - Stephen Huang (UG)
  - Vladimir Hanin (UG)
  - Sergio Dominguez (UG)
- Others UG
  - Yang – FPGA
  - Cong – Virtualization
  - Nicholas – Applications

Systems Nuts Research Group

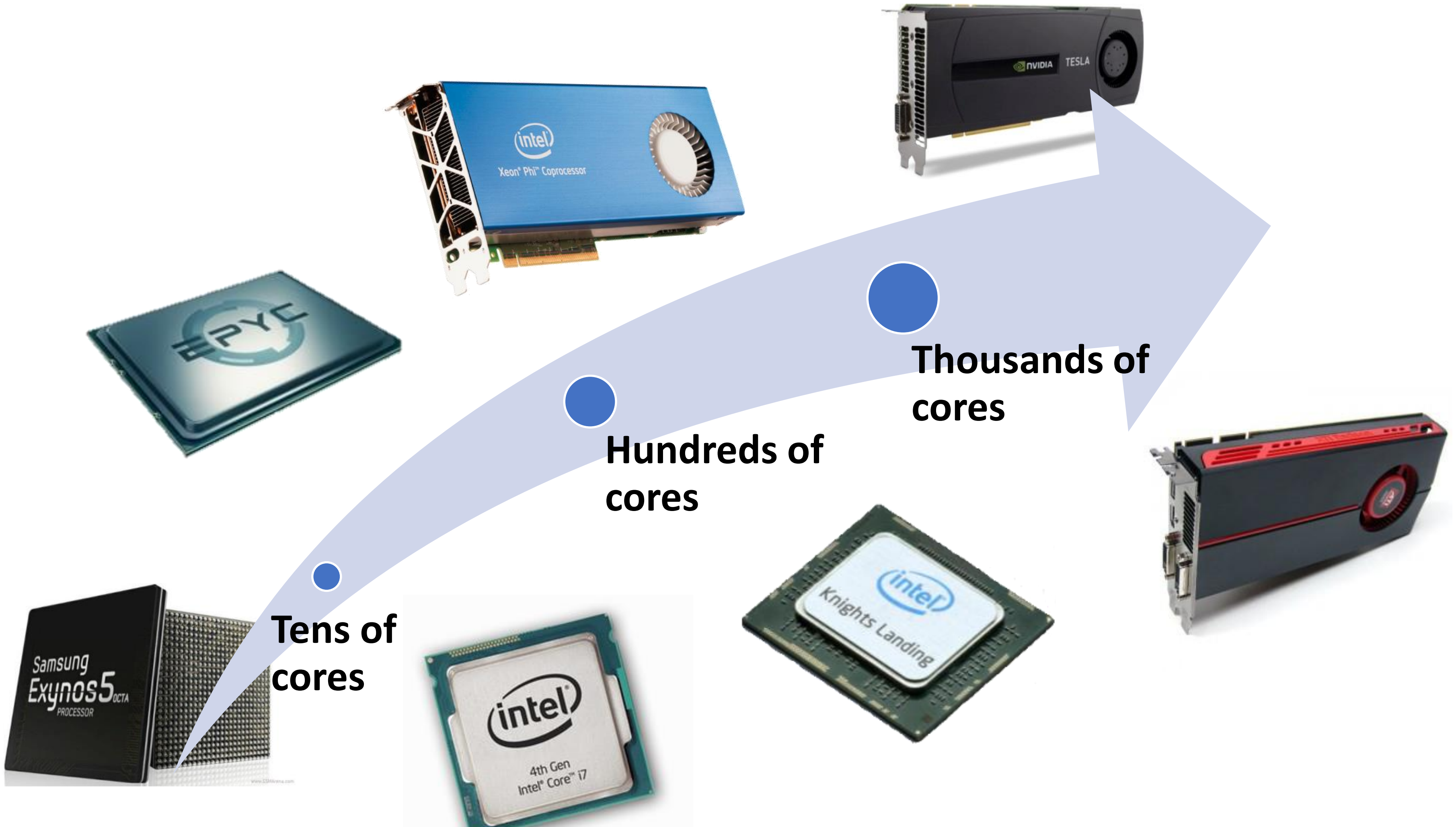
https://github.com/systems-nuts

Traditional Computer Hardware

- CPU, memory, disk, network
  - Central processor
  - Hierarchy of memory
  - Slow interconnects and IOs
- **Cannot satisfy** applications' demands
- Hardware is (radically) **changing**
  - Circumventing limitations
    - Moore law
    - Dennard scaling
    - Physical limits
    - etc.



Hardware Trend: **Parallelism**



Tens of cores

Hundreds of cores

Thousands of cores

Each image is Copyright of the respective Company or Manufacturer. Images are used here for educational purposes.

Hardware Trend: **Heterogeneity**

AMD

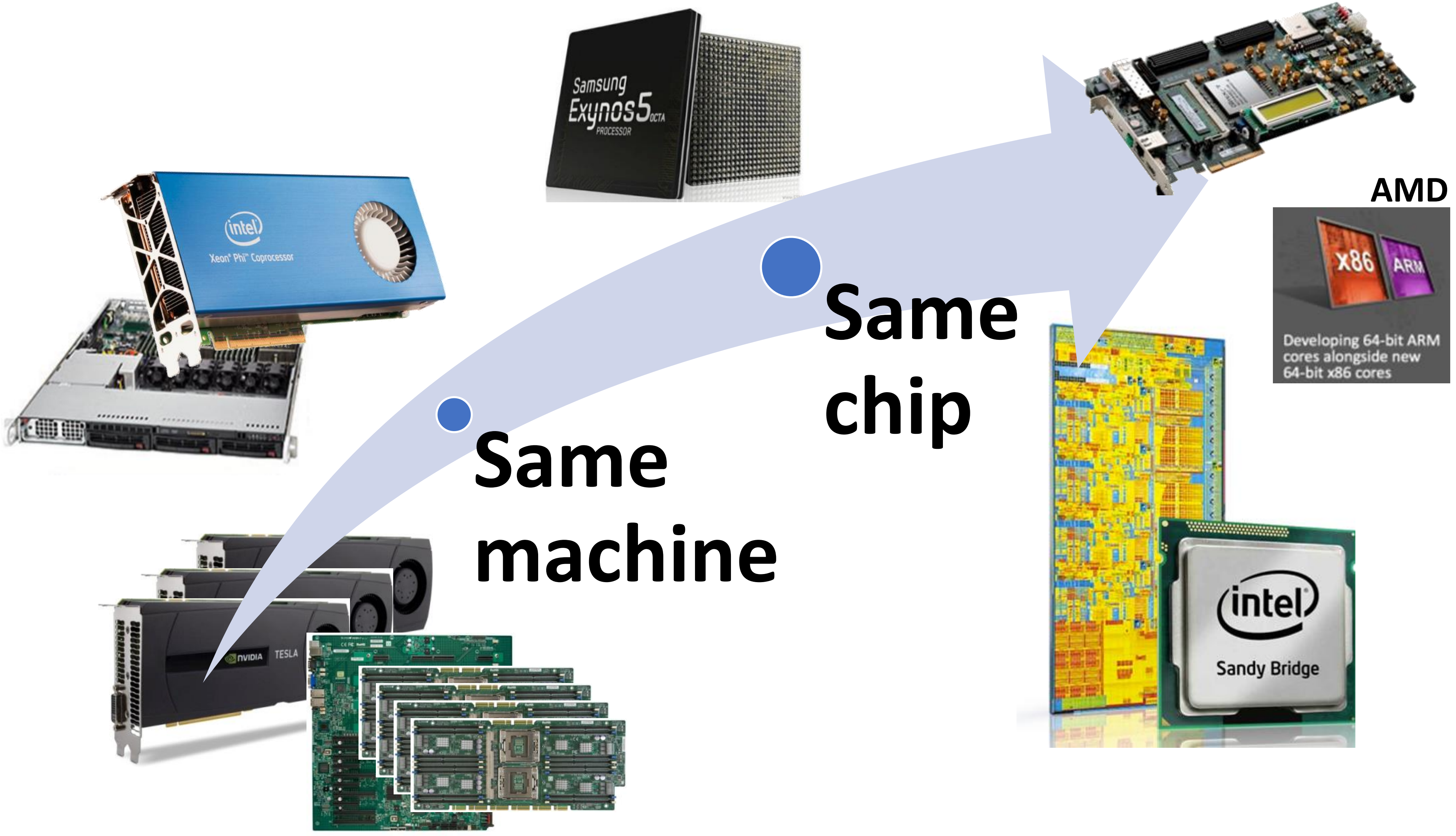
OS-capable

Reconfigurable

Special purpose

Fully general purpose

Hardware Trend: Integration



Same machine

Same chip

Hardware Trend: **Coherency**

AMBA 5 CHI

Ncore

Intel® QuickPath Interconnect

FABRIC

Die/Package Inter-connect

OpenPiton

CCIX™

Device Inter-connect

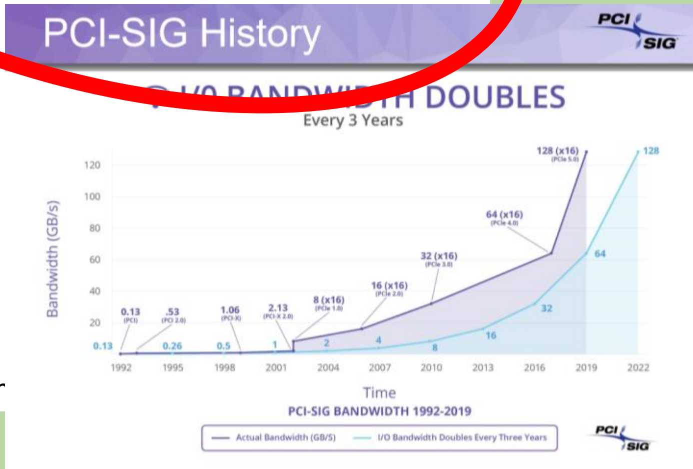
CXL Compute Express Link

Commercial Prototypes

OpenCAPI™

GEN Z

Each image is Copyright of the respective Company or Manufacturer. Images are used here for educatior



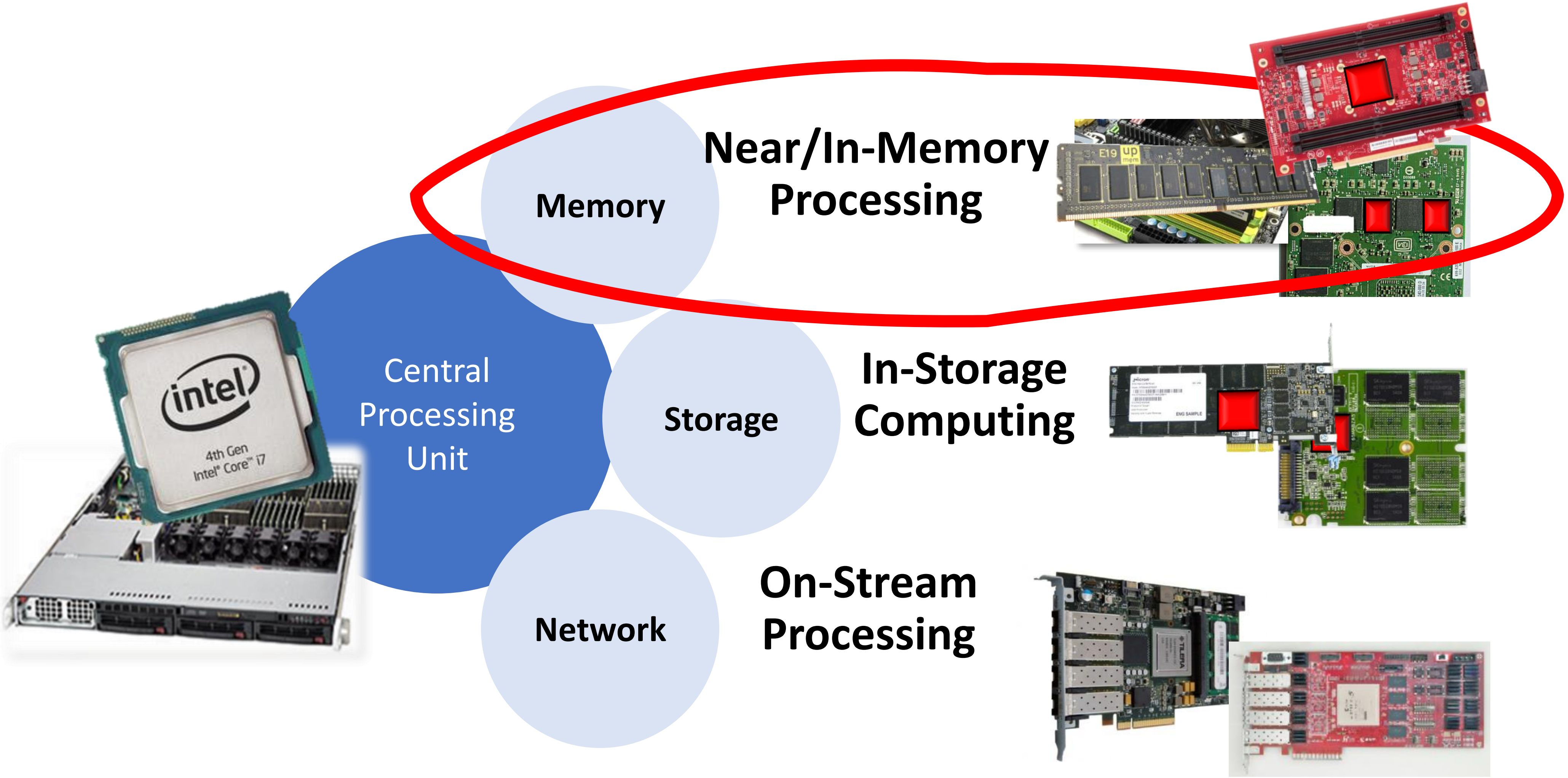
PCI-SIG History

I/O BANDWIDTH DOUBLES

Every 3 Years

PCI-SIG BANDWIDTH 1992-2019

Hardware Trend: **Near Data Processing**



Central Processing Unit

Memory

Near/In-Memory Processing

Storage

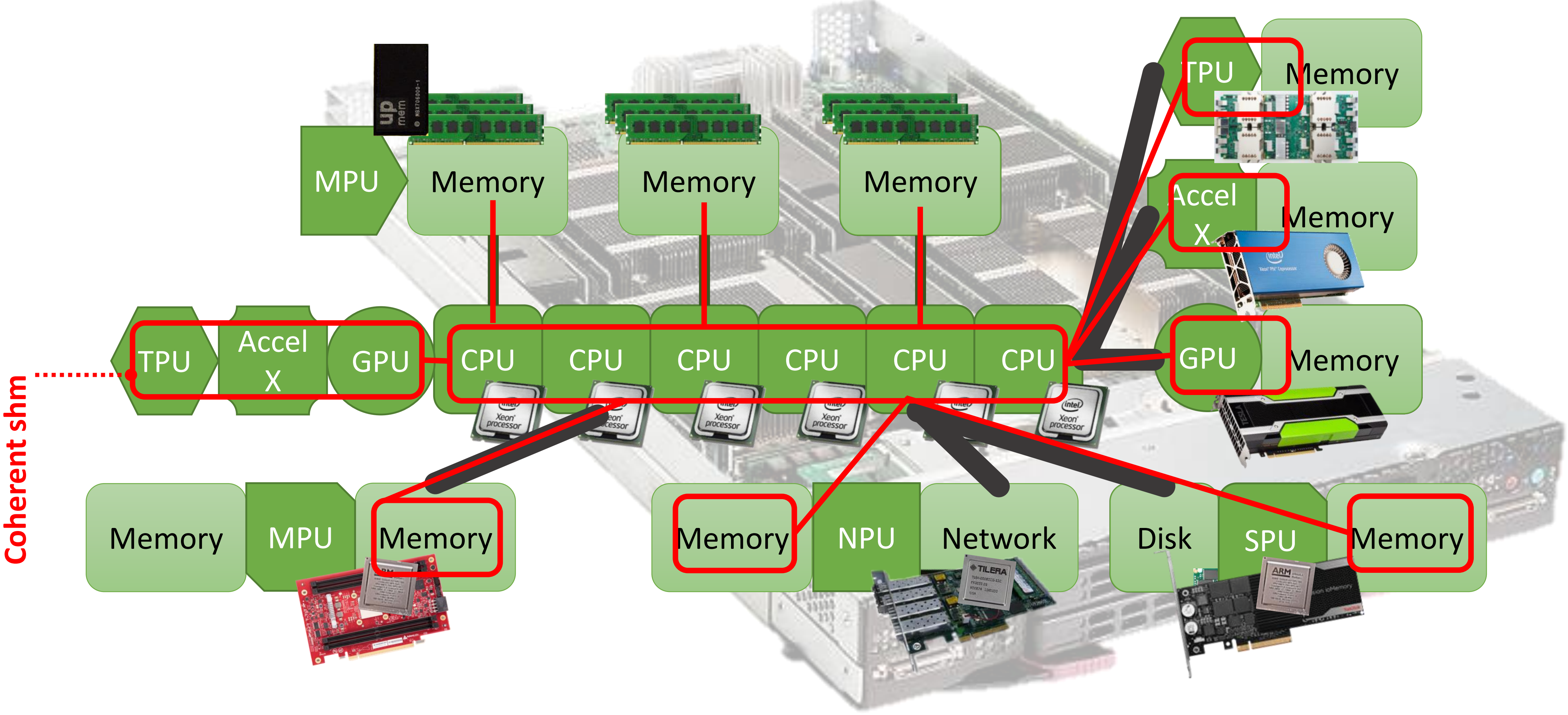
In-Storage Computing

Network

On-Stream Processing

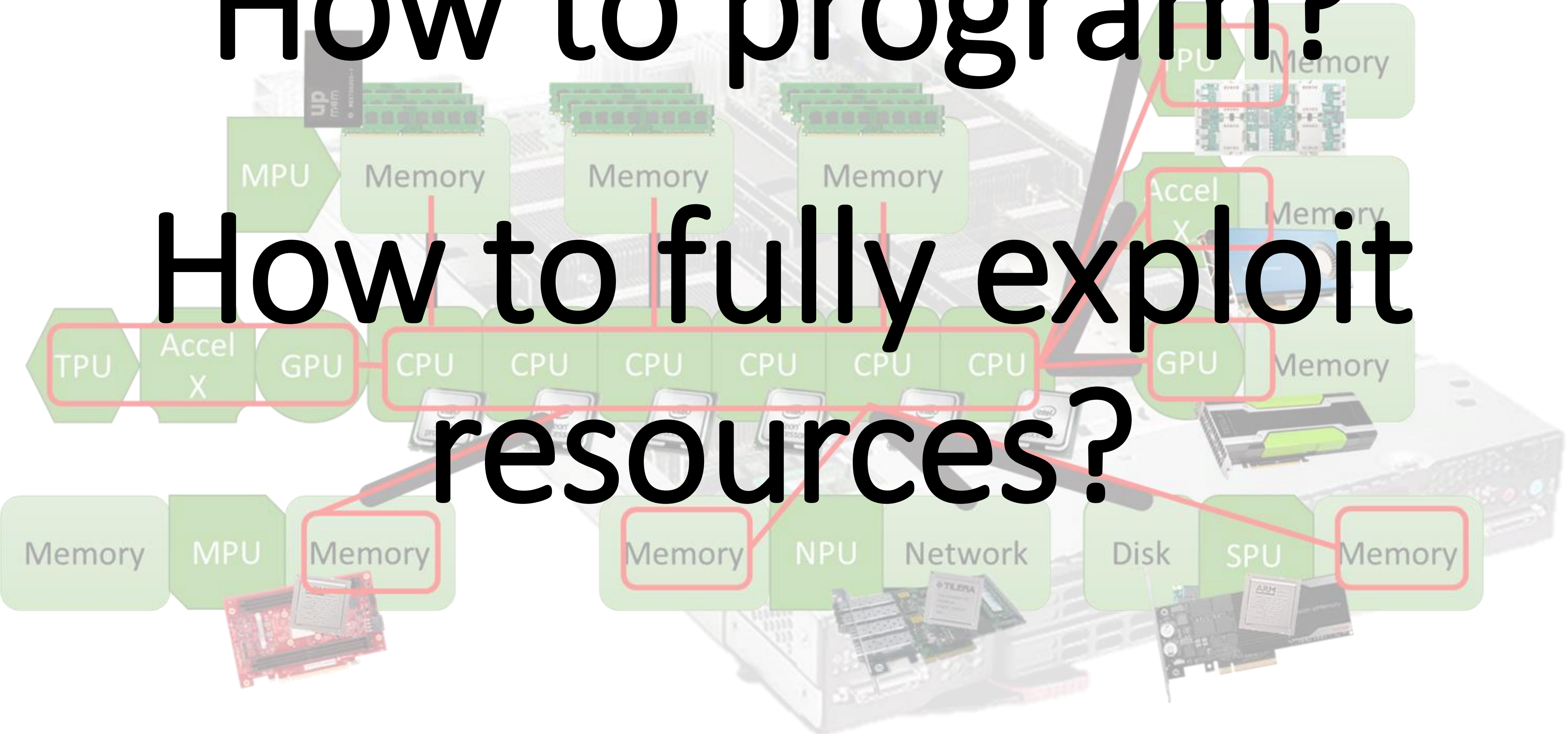
Each image is Copyright of the respective Company or Manufacturer. Images are used here for educational purposes.

Today's New Computer Hardware

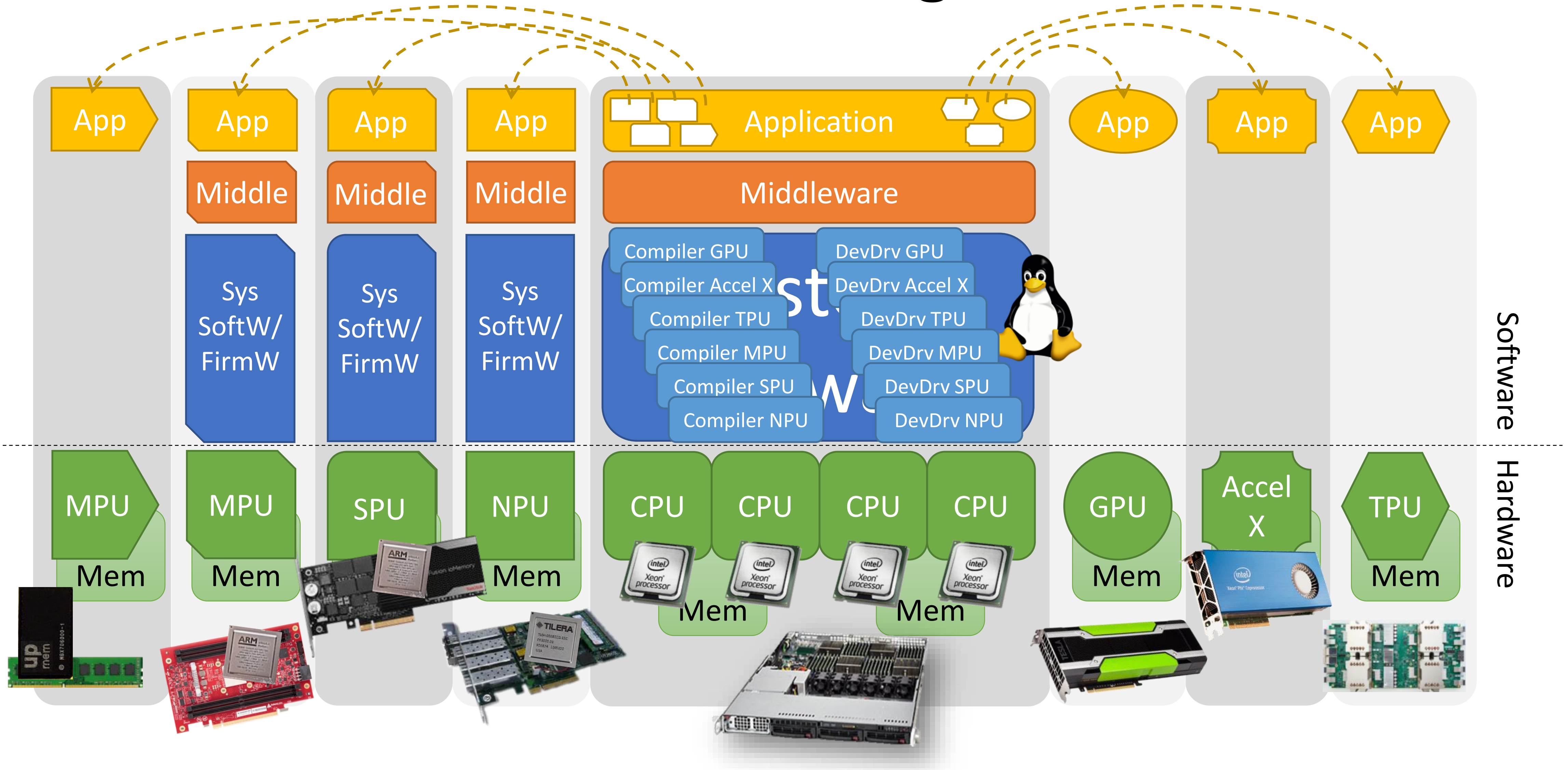


How to program?

How to fully exploit resources?

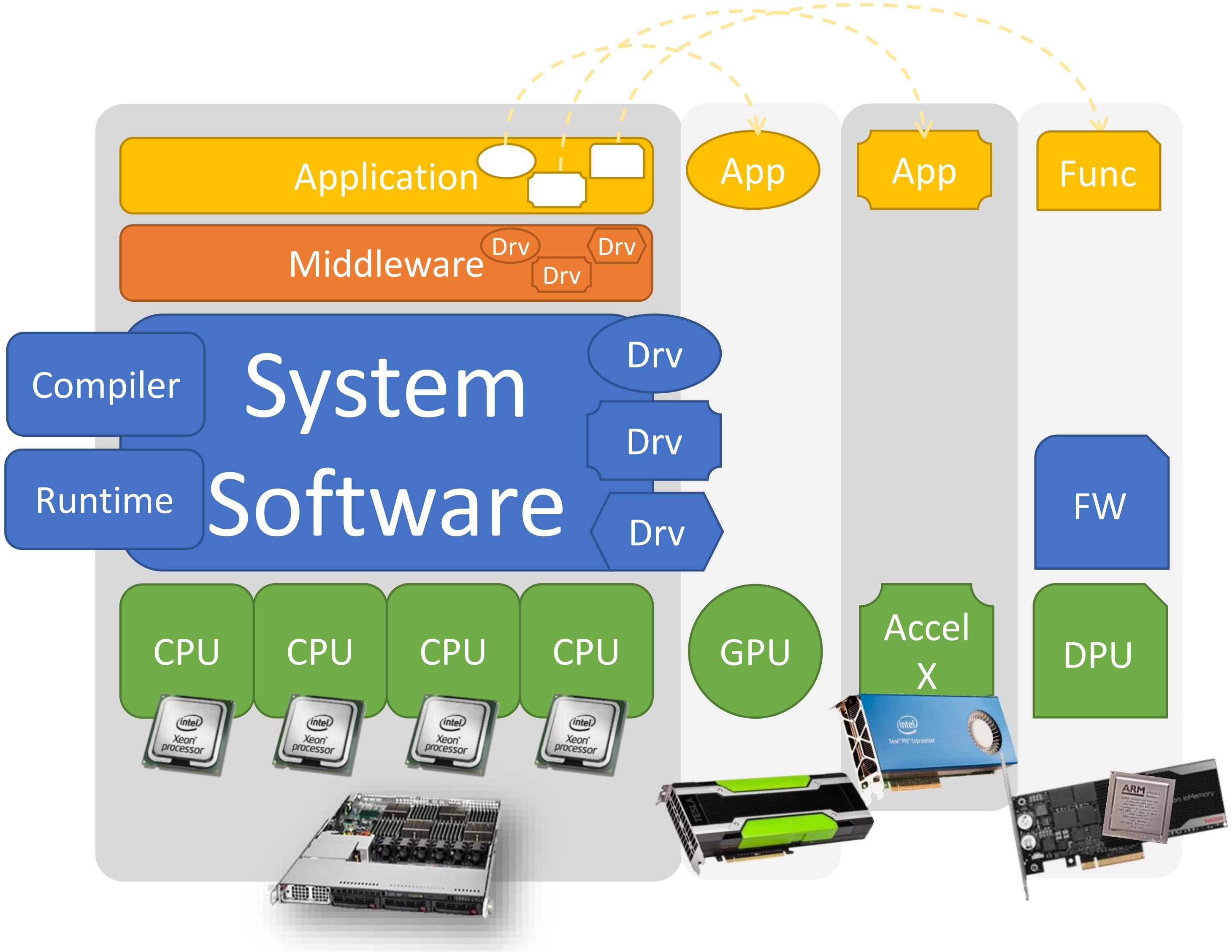


Current Software for Heterogeneous Hardware



Current Software for Heterogeneous Hardware

- App Software runs on CPUs
- Other processing units **cannot run** the same software as the CPUs
  - **Different ISA**
  - **No** shared memory
- **Programmer (strictly) partitions** the application
- Each partition **runs only** on a predefined processing unit
- **Supporting** drivers, runtime, compilers



Current Software for Heterogeneous Hardware

```
void full_verify( void )
{
  INT_TYPE    i, j;

  for( i=0; i<NUM_KEYS; i++ )
     key_buff2[i] = key_array[i];

  for( i=0; i<NUM_KEYS; i++ )
    key_array[--
    key_buff_ptr_global[key_buff2[i]]]
      = key_buff2[i];
...
}
```

NPB IS **serial snippet**

Benchmark	CG	EP	FT	IS	MG
Serial LOC	506	163	606	454	852
OpenCL added	**303%**	**164%**	**143%**	**177%**	**189%**

serial and OpenCL version of SNU NPB

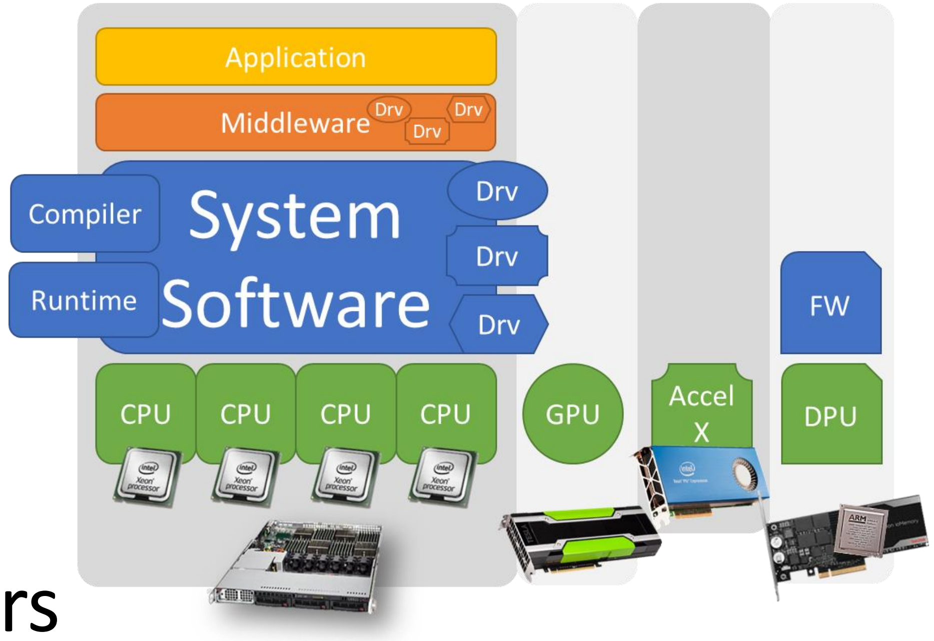
[1] “Popcorn: bridging the programmability gap in heterogeneous-ISA platforms” A. Barbalace et al., EuroSys ‘15

```
void full_verify( void )

{

  cl_kernel k_fv0, k_fv1;

  cl_mem    m_j; cl_int ecode;

  INT_TYPE *g_j;

  INT_TYPE j = 0, i;

  size_t j_size; size_t fv0_lws[1], fv0_gws[1]; size_t fv1_lws[1], fv1_gws[1];


  j_size = sizeof(INT_TYPE) * (FV2_GLOBAL_SIZE / FV2_GROUP_SIZE);

  m_j = clCreateBuffer(context, CL_MEM_READ_WRITE, j_size, NULL, &ecode);


  k_fv1 = clCreateKernel(program, "full_verify1", &ecode);

  k_fv0 = clCreateKernel(program, "full_verify0", &ecode);


  ecode  = clSetKernelArg(k_fv0, 0, sizeof(cl_mem), (void*)&m_key_array);

  ecode |= clSetKernelArg(k_fv0, 1, sizeof(cl_mem), (void*)&m_key_buff2);

  fv0_lws[0] = work_item_sizes[0];

  fv0_gws[0] = NUM_KEYS;

  ecode = clEnqueueNDRangeKernel(cmd_queue, k_fv0, 1, NULL, fv0_gws, fv0_lws,
0, NULL, NULL);


  ecode  = clSetKernelArg(k_fv1, 0, sizeof(cl_mem), (void*)&m_key_buff2);

  ecode |= clSetKernelArg(k_fv1, 1, sizeof(cl_mem), (void*)&m_key_buff1);

  fv1_lws[0] = work_item_sizes[0];

  fv1_gws[0] = NUM_KEYS;

  ecode = clEnqueueNDRangeKernel(cmd_queue, k_fv1, 1, NULL, fv1_gws, fv1_lws,
0, NULL, NULL);
```

NPB IS **OpenCL snippet**

What Are the Problems?

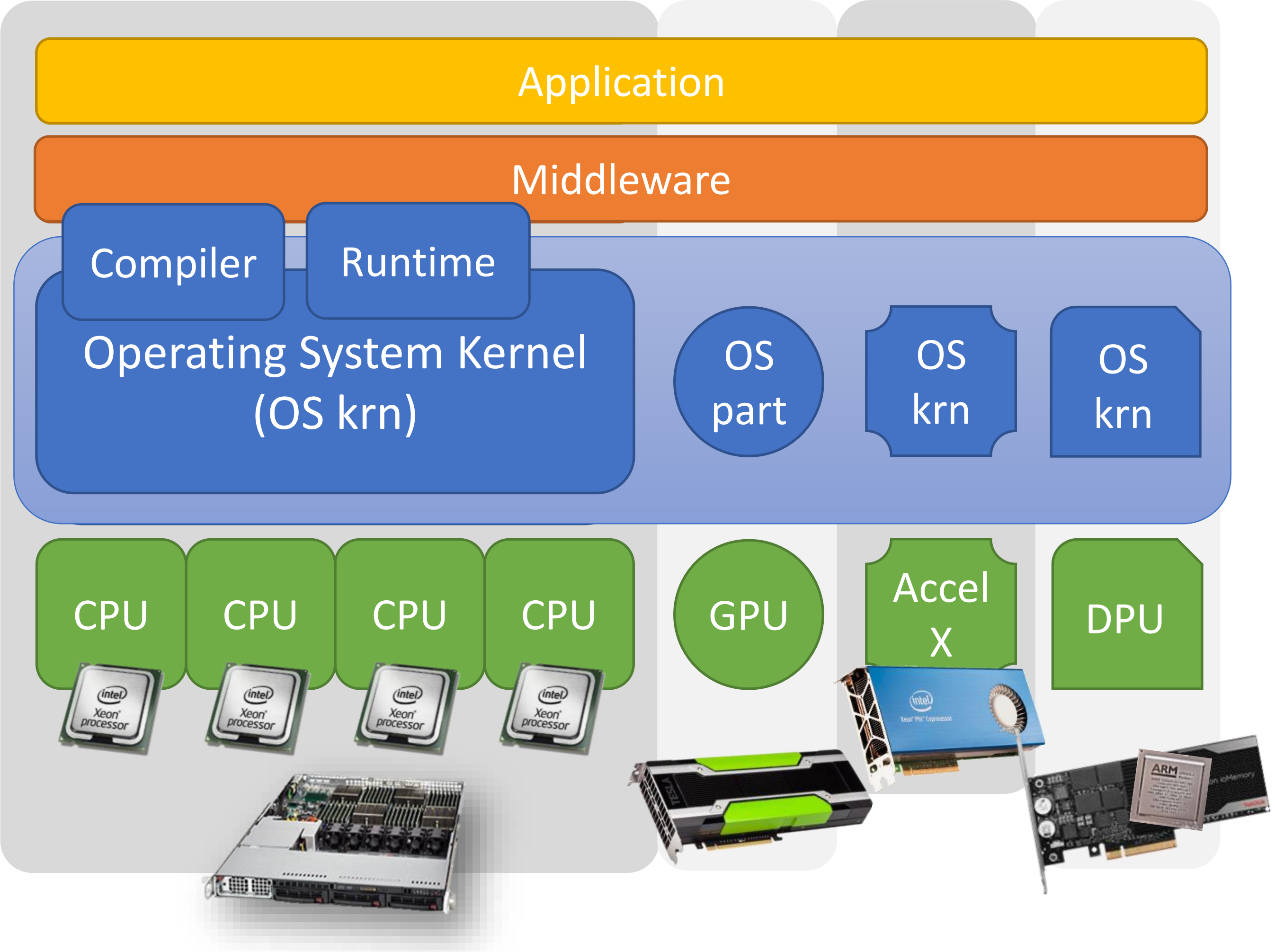
- **For each new** hardware component
  - A new **support software**
- **Nightmare** for application’s programmers
  - Hard to program (lot of boilerplate code)
    - Several APIs/programming models exist
  - Difficult to port to a new platform
  - Rigid, **poor resource utilization**
    - Performance, energy efficiency, determinism
    - One programmer focuses on one application
      - Many applications run at the same time



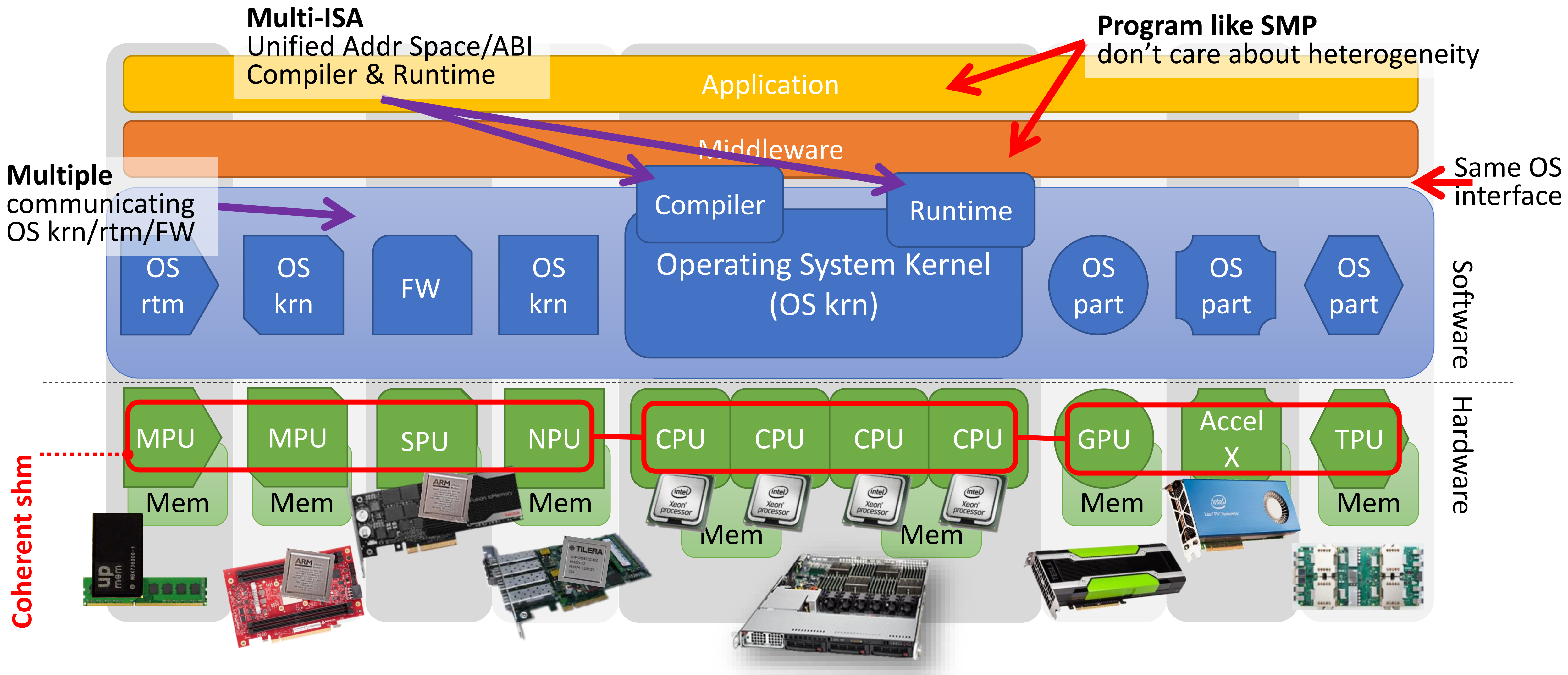
Solved by middleware?
But software lock-in …

New Software for Heterogeneous Hardware

- The **OS** extends among all processing units
- The **compiler** builds applications software to run among all processing units
- The **runtime** supports all processing units
- **Programmers** don't have to partition the application, which may run everywhere, **transparently**



Key Idea:
Forget Offloading, Program like **SMP**



Multi-ISA
Unified Addr Space/ABI
Compiler & Runtime

Program like SMP
don't care about heterogeneity

Multiple
communicating
OS krn/rtm/FW

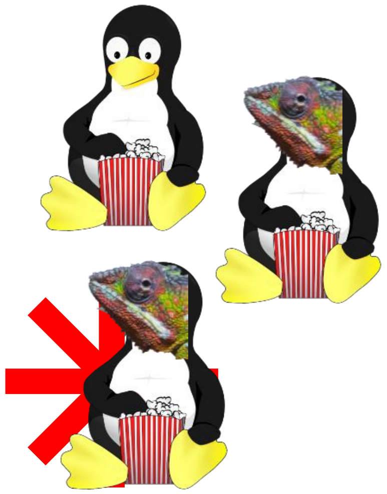
Same OS
interface

Coherent shm

Taming pervasive **CPU Heterogeneity** (earlier work)

Introducing pervasive
Coherent Shared Memory
(current work)

Popcorn Linux and Compiler Framework Family of Projects



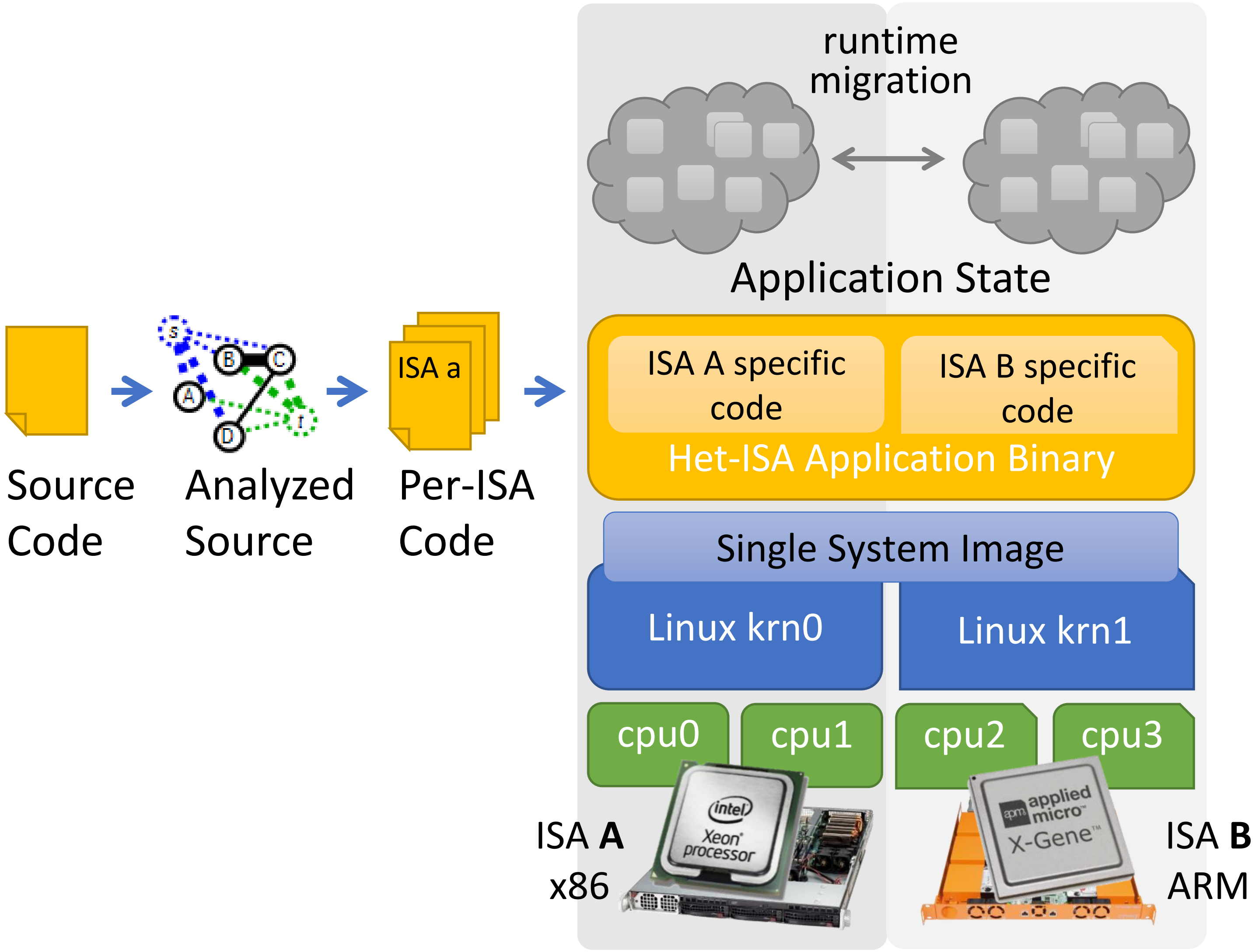
- Started at Virginia Tech, Blacksburg, VA, **mid-2012**
  - Several Linux kernel/LLVM versions exists – upstream attempted

- Targets platforms with multiple groups of **general-purpose** PUs
  - **Non-cache-coherent** groups
  - **Microarchitectural** or **ISA heterogenous**

- Initial goal(s)
  - Extend the **multiple kernel OS design** (Barrelfish, DragonFly BSD) to Linux
  - Provide the **same OS and programming environment** among (diverse) processing units

- **OS + compiler** transparently provide **SMP environment** on **non-SMP platforms**

Popcorn Linux and Compiler Framework



Source Code

Analyzed Source

Per-ISA Code

ISA a

runtime migration

Application State

ISA A specific code

ISA B specific code

Het-ISA Application Binary

Single System Image

Linux krn0

Linux krn1

cpu0

cpu1

cpu2

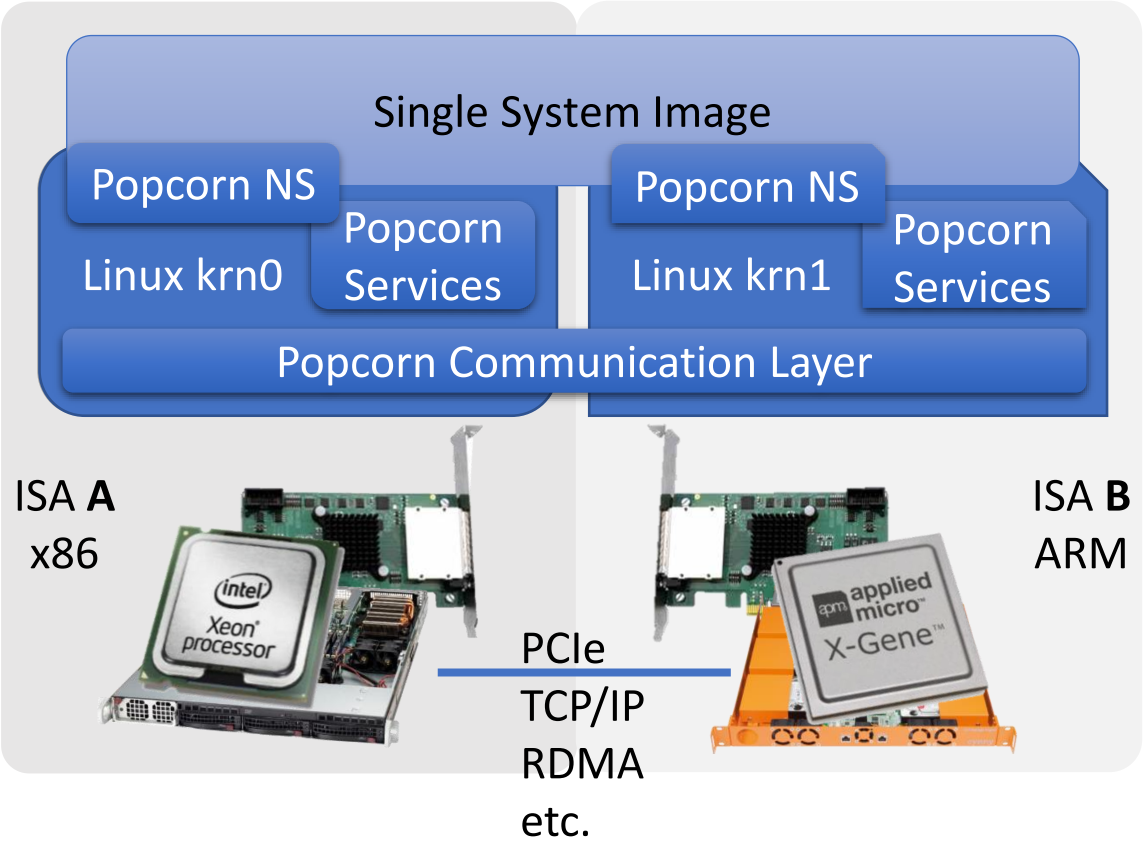
cpu3

ISA **A**
x86

ISA **B**
ARM

- **Runtime**
  - Runtime ISA execution migration
    - State transformation
  - Based on **musl C library**
- **Compiler** Framework
  - Offline analysis
    - Model-based code optimization
  - One binary per ISA
  - Based on **gcc/LLVM**
- Replicated-kernel **Operating System**
  - One kernel per ISA
  - Distributed systems services
    - Single system Image
  - Based on **Linux**

Popcorn Linux – Operating System

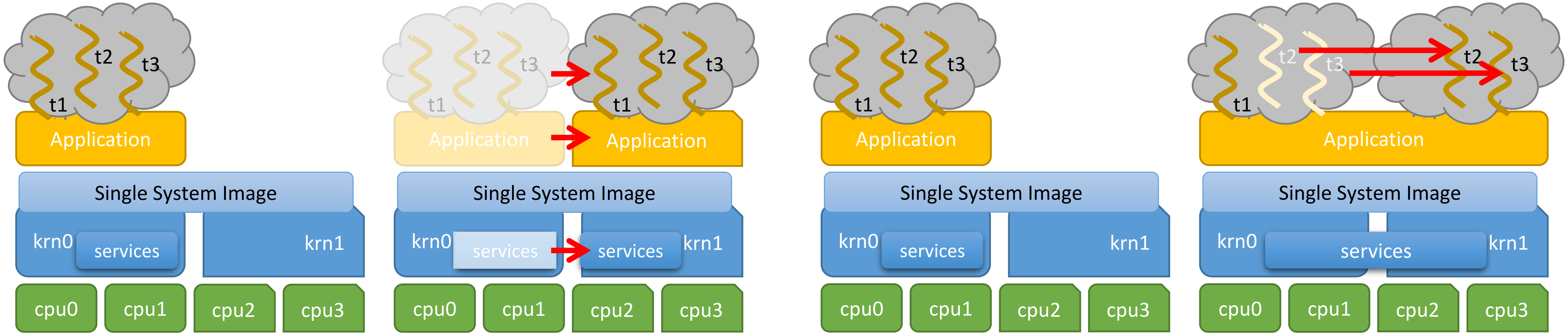


- **Single System Image**
  - Based on Popcorn namespaces (NS)
    - Extends Linux namespaces
  - Creates a single operating environment
    - Migrating app sees the same OS
- **Distributed OS Services**
  - Task (thread and process) migration
    - Native code migration
  - Distributed memory management (DSM)
  - Distributed file system
- **Inter-kernel Communication Layer**
  - Performance critical component
    - low-latency and high-throughput
  - Exclusively kernel-space
  - Single format among ISAs

Popcorn Linux – Task Migration

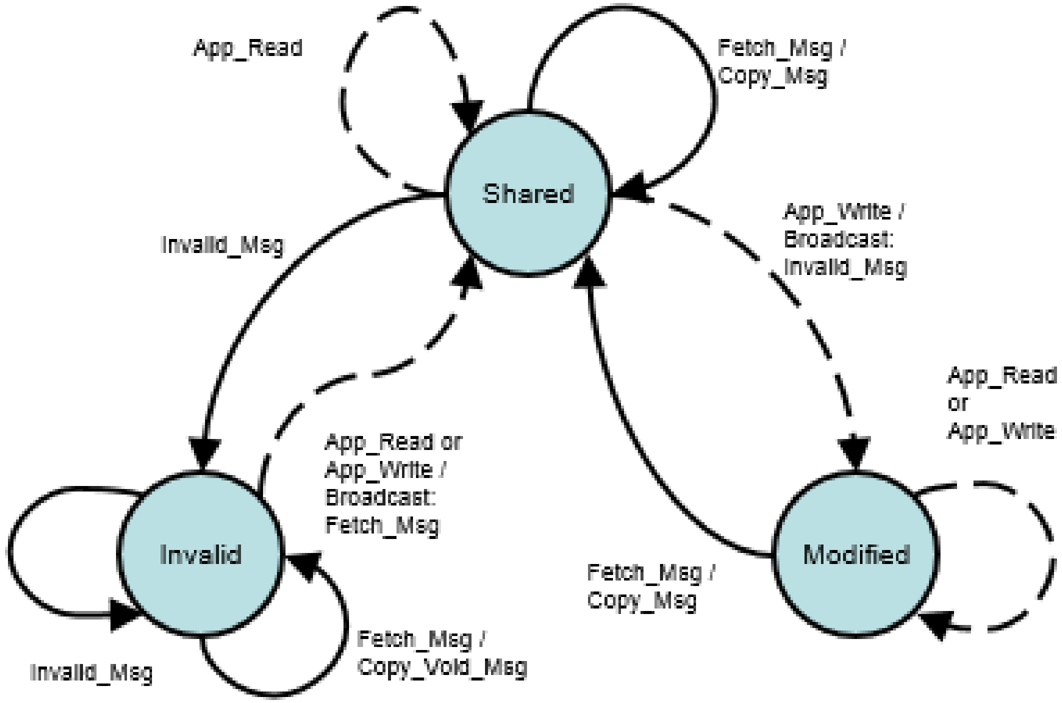
- **Process** Migration
- Whole application is transferred
  - All threads, user- & kernel-state
- No dependecies are left on the origin kernel

- **Thread** Migration
- Selected threads are transferred
  - Threads' state is transferred
- Kernels coordinate to maintain application state consistent

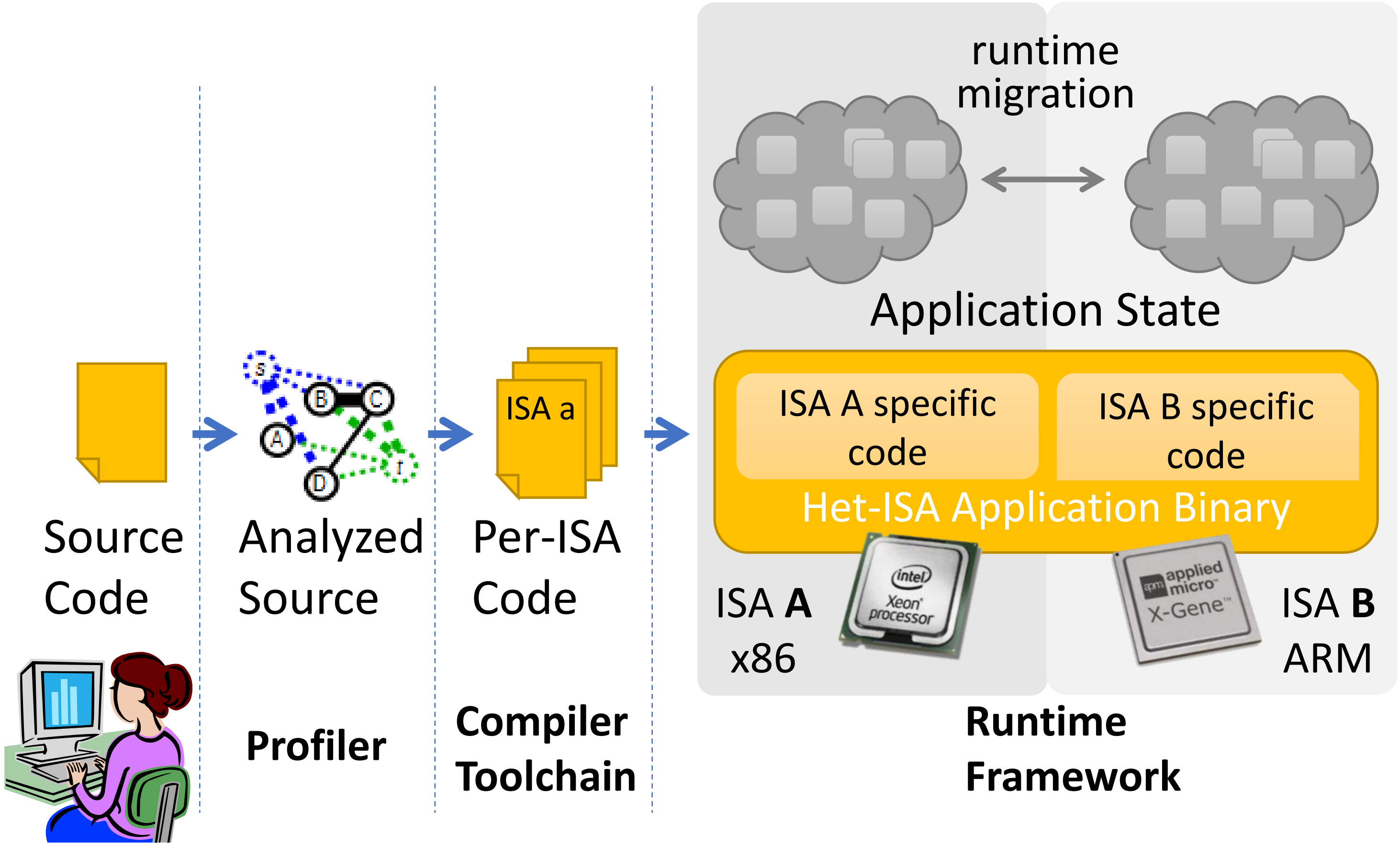


Popcorn Linux – Thread Migration's **DSM**

- Replicated virtual address space
- Kept consistent among kernels
- Page coherency protocol
  - Based on **Modified-Shared-Invalid (MSI)** cache coherency protocol
  - Memory page granularity instead of cache line granularity
  - Additional states to improve performance
  - Scaled from two kernels to multiple kernels



Popcorn Linux – Compiler/Runtime



Source Code

Analyzed Source

Per-ISA Code

ISA a

runtime migration

Application State

ISA A specific code

ISA B specific code

Het-ISA Application Binary

ISA **A** x86

ISA **B** ARM

Profiler

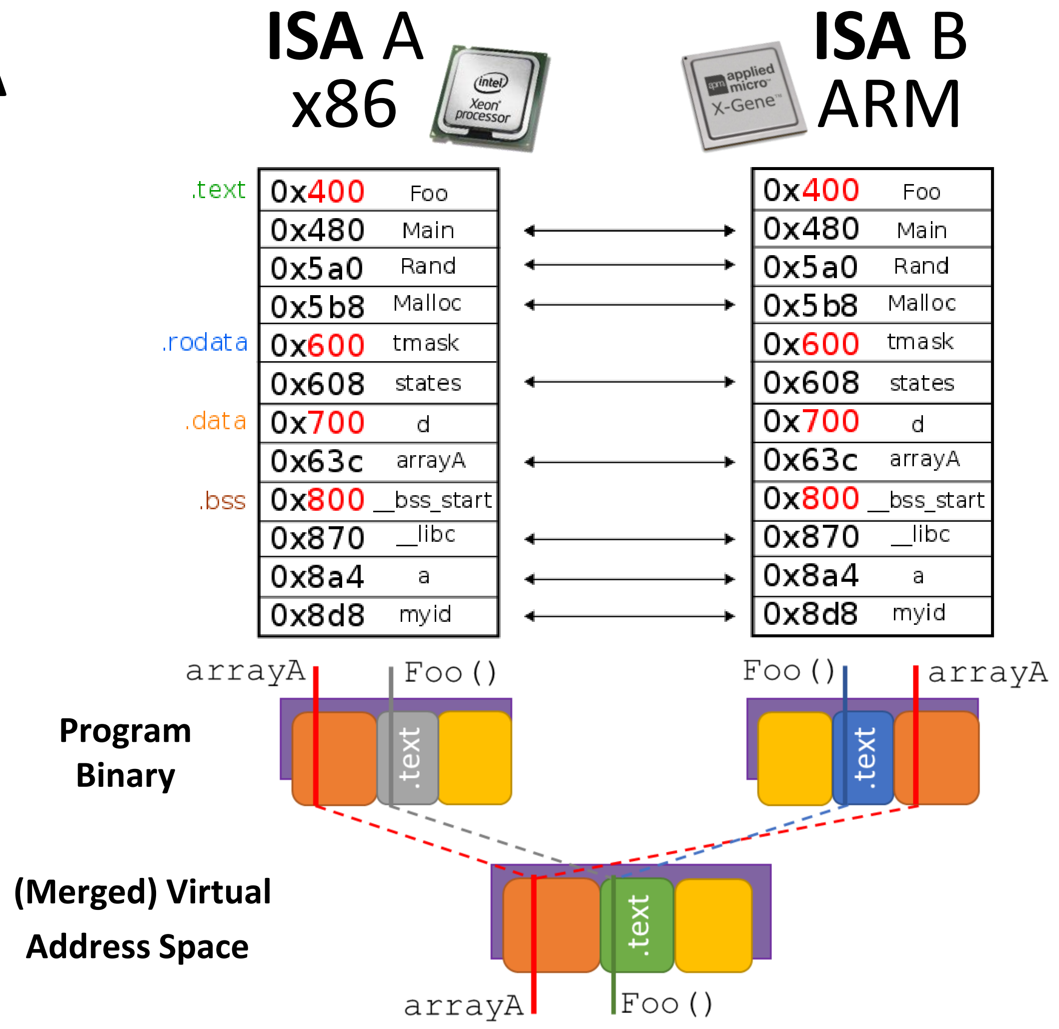
Compiler Toolchain

Runtime Framework

- **Profiler**
  - Performance and power profiles
  - Function and sub-function granularity
  - Output performance and power code indicators
    - Affinity estimations with cost model
- **Compiler Toolchain**
  - Output heterogenous-ISA binary (native)
    - Common address space (including TLS)
    - Add migration points (func boundaries)
    - Add state transformation metadata
- **Runtime Framework**
  - Support task migration
  - Implements state transformation
    - Stack-transformation (rewriting)
    - Register-transformation

Popcorn Linux – Compiler

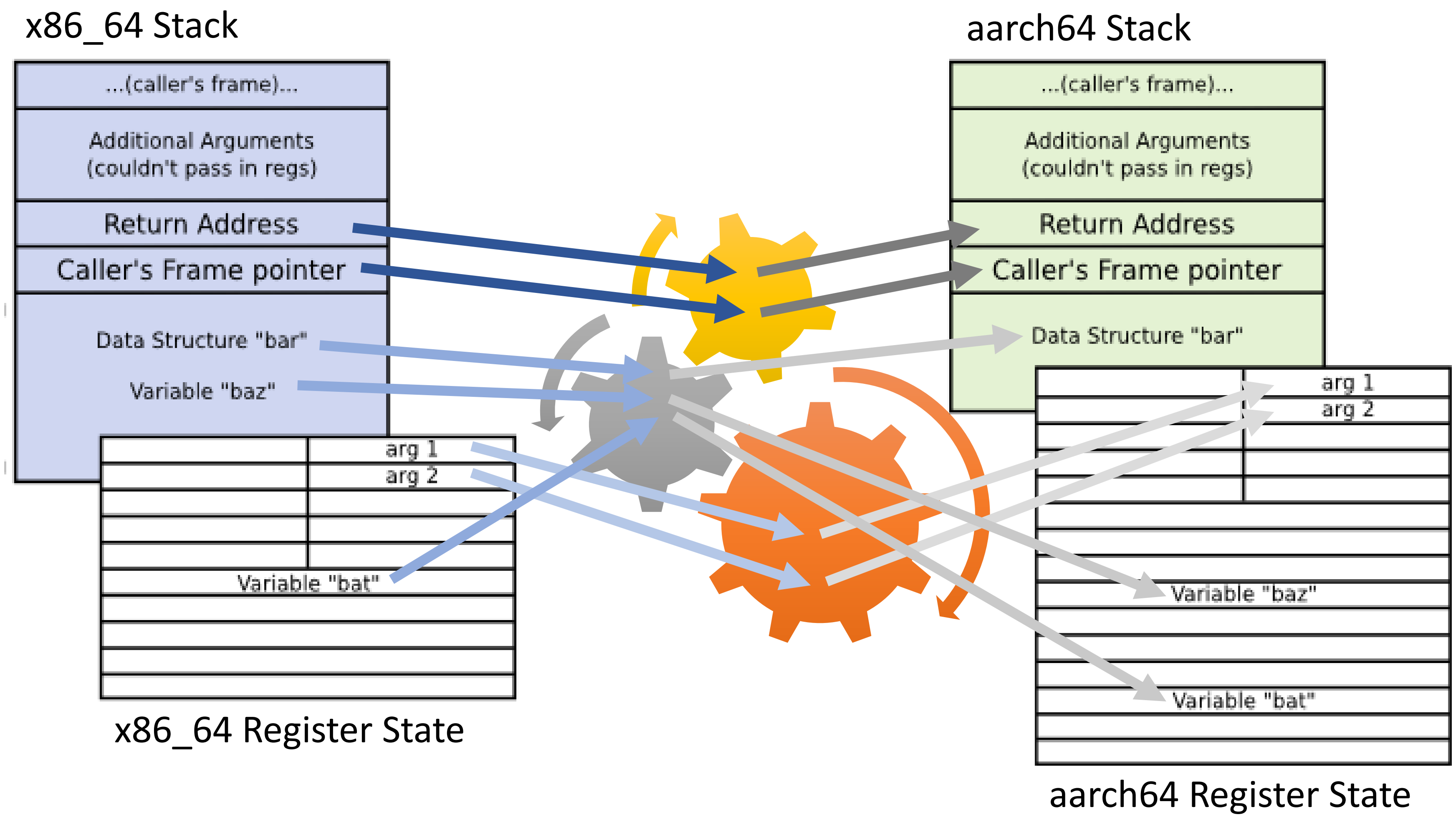
- Produces **program binaries for each ISA**
  - **Common address space**
    - Common type system (and alignments)
    - Each symbol at same **virtual address** on any ISA
    - *No address space conversion!*
  - **Common thread-local storage** (TLS) layout
    - x86_64 layout forced
    - *No TLS conversion!*
  - **Migration points**
    - Cannot migrate at any instruction
  - **State-transformation meta-data** in binaries
    - E.g., var properties, stack frame offsets



	ISA A x86		ISA B ARM	
.text	0x400	Foo	0x400	Foo
	0x480	Main	0x480	Main
	0x5a0	Rand	0x5a0	Rand
	0x5b8	Malloc	0x5b8	Malloc
.rodata	0x600	tmask	0x600	tmask
	0x608	states	0x608	states
.data	0x700	d	0x700	d
	0x63c	arrayA	0x63c	arrayA
.bss	0x800	__bss_start	0x800	__bss_start
	0x870	__libc	0x870	__libc
	0x8a4	a	0x8a4	a
	0x8d8	myid	0x8d8	myid

Popcorn Linux – Runtime

Stack Transformation

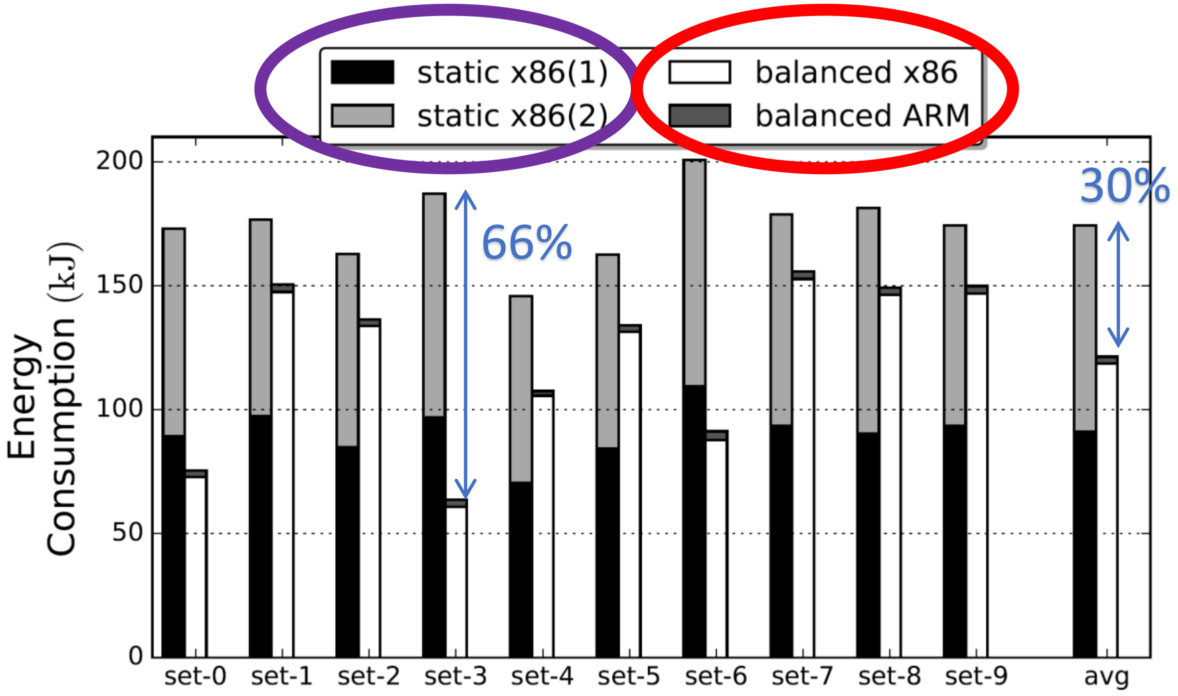


Popcorn Linux Results

- Ease programmability
- Enable portability (and legacy support)
- Improve resource utilization
    - **Runtime decisions** (vs static)
        - On heterogeneous-ISA **[1]**
            - Up to 3.5x more performant than other heterogeneous frameworks
        - On **fully** heterogeneous-ISA **[2]**
            - Up to 66% better energy consumption for bursty arrivals

Homogeneous System

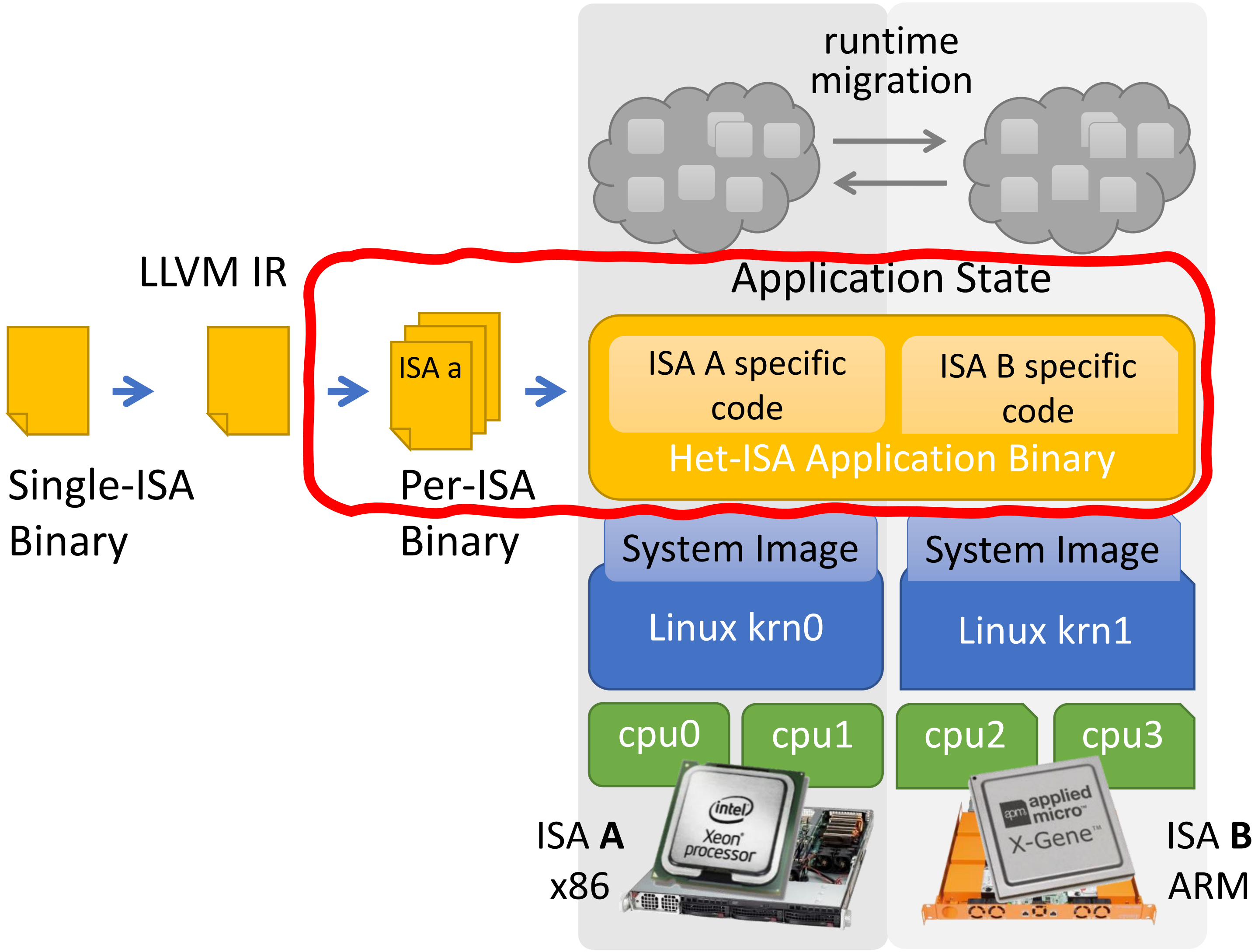
Heterogeneous System



[1] “Bridging the Programmability Gap in Heterogeneous-ISA Platforms” A. Barbalace et al., EuroSys '15

[2] “Breaking the Boundaries in Heterogeneous-ISA Datacenters” A. Barbalace et al., ASPLOS '17

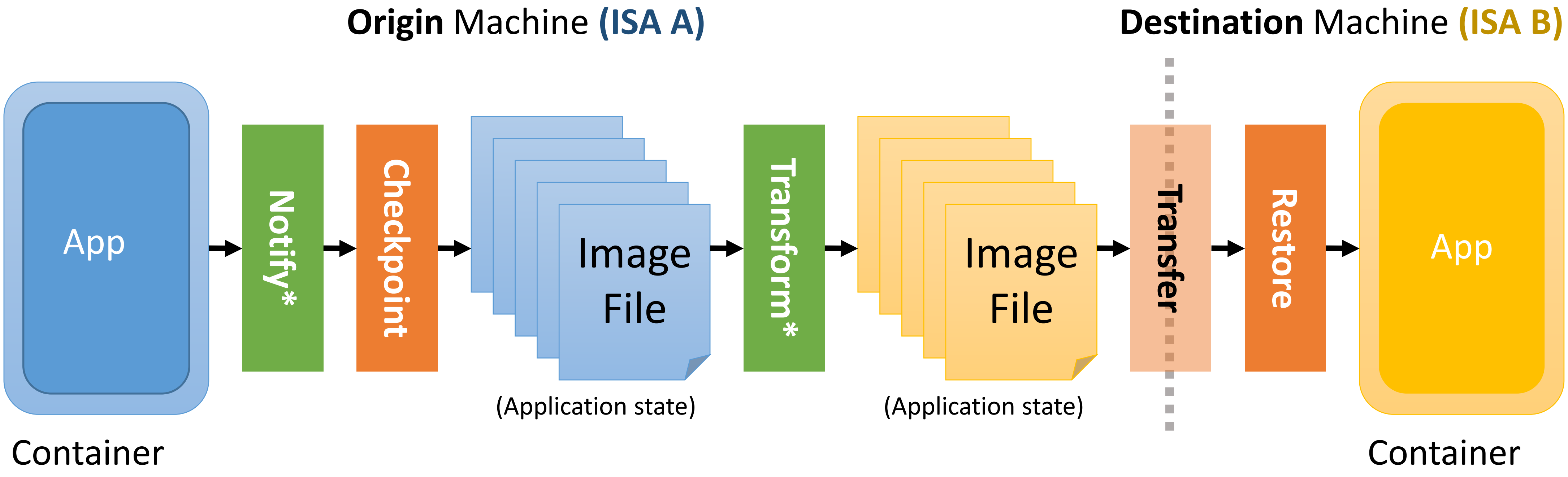
H-Containers



- **Runtime**
  - OS Process-level Checkpoint/Restart
  - Based on **CRIU** and **Popcorn Runtime** (muslc-based)
- **Transpiler** Framework
  - Binary decompiled to LLVM IR
  - LLVM IR to per-ISA Binary
  - Based on **McSema/Remill** and **Popcorn Compiler** (LLVM)
- Vanilla ~~**Operating System**~~
  - ~~Based on Linux,~~ Linux containers
    - Namespaces, cgroups

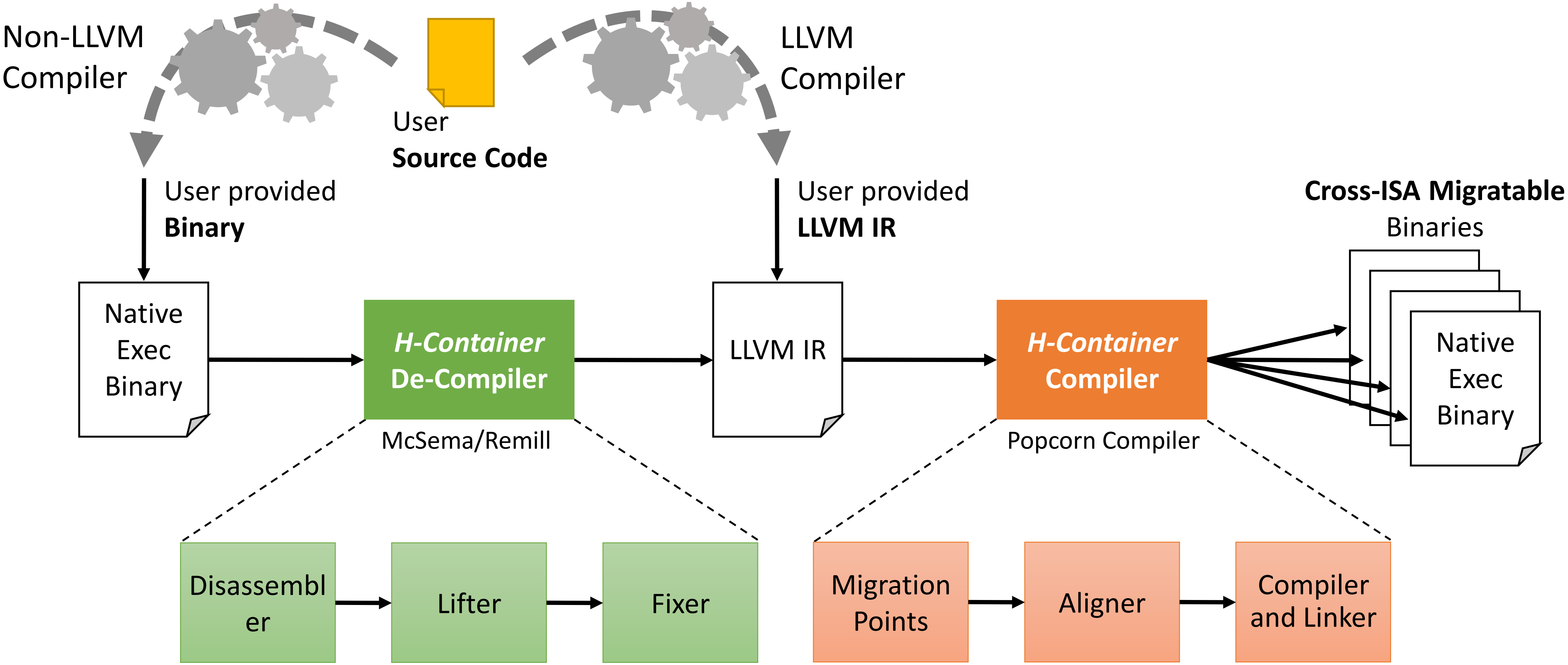
H-Container – Runtime

Checkpoint/Restart Migration



**New Components*

H-Containers – Transpiler



Non-LLVM Compiler

User **Source Code**

LLVM Compiler

User provided **Binary**

User provided **LLVM IR**

Cross-ISA Migratable Binaries

Native Exec Binary

***H-Container* De-Compiler**

McSema/Remill

LLVM IR

***H-Container* Compiler**

Popcorn Compiler

Native Exec Binary

Disassembl er

Lifter

Fixer

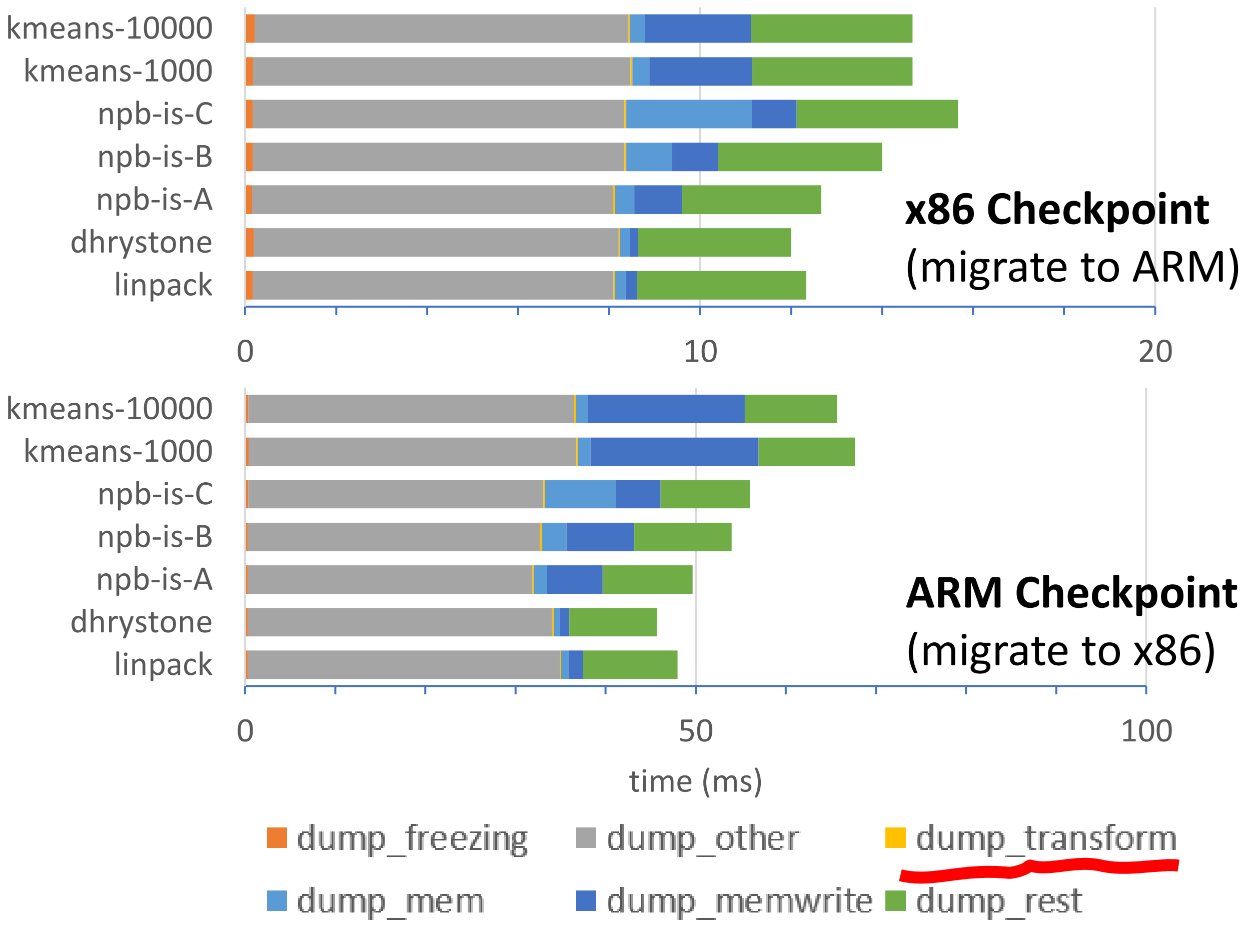
Migration Points

Aligner

Compiler and Linker

H-Containers Results

- Fully-working implementation
  - Docker support
- True dependency-free
  - No need of source code
  - Works on any Linux kernel
- **Minimal overheads** [1]
  - Multiple benchmarks
  - Application run time is the same or lower
  - State transform contribute to less than 1% to migration



[1] "Edge Computing: the Case for Heterogeneous-ISA Container Migration" A. Barbalace et al., VEE '20

Taming pervasive
CPU Heterogeneity
(earlier work)

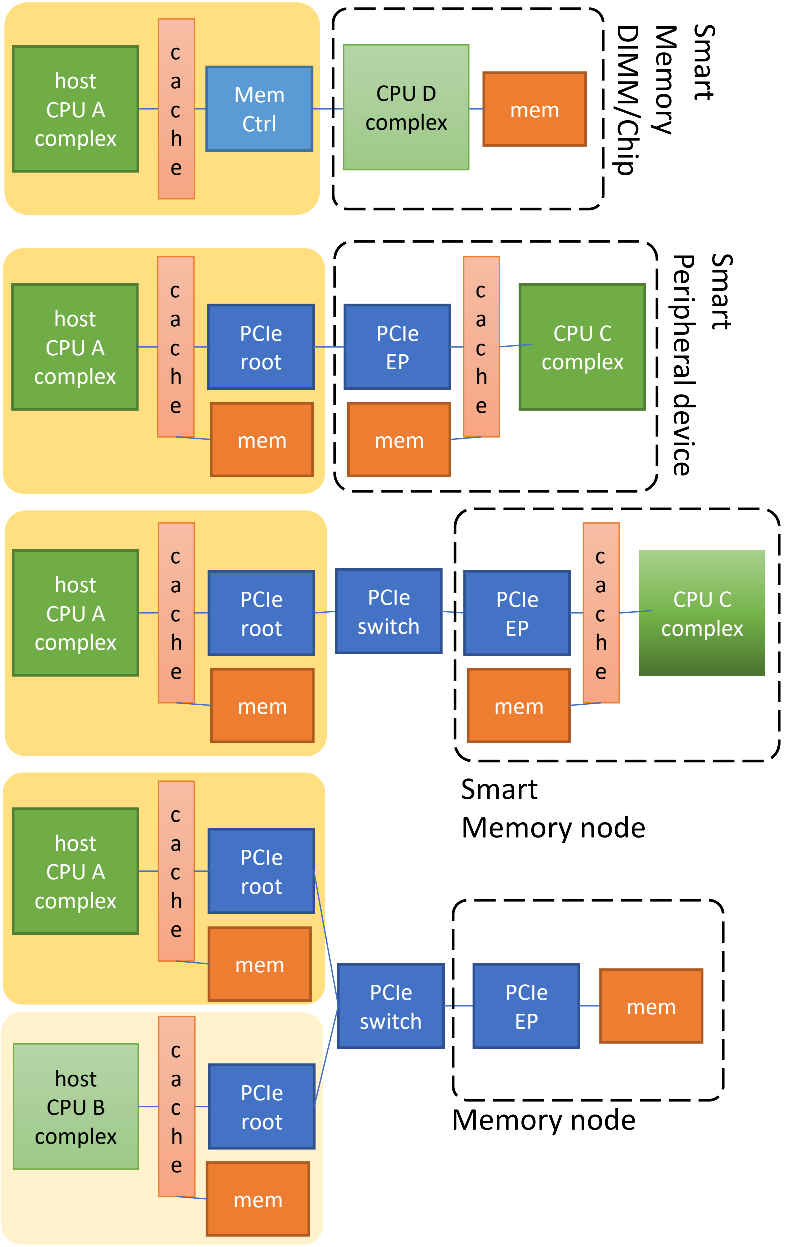
Introducing pervasive
Coherent Shared Memory
(current work)

Heterogeneous-ISA CPUs with **Coherent Shared Memory** Projects

- Started at Huawei, Germany, **mid-2017** [1]
  - CCIX SmartNICs, SmartSSD, Smart Memory nodes, etc.

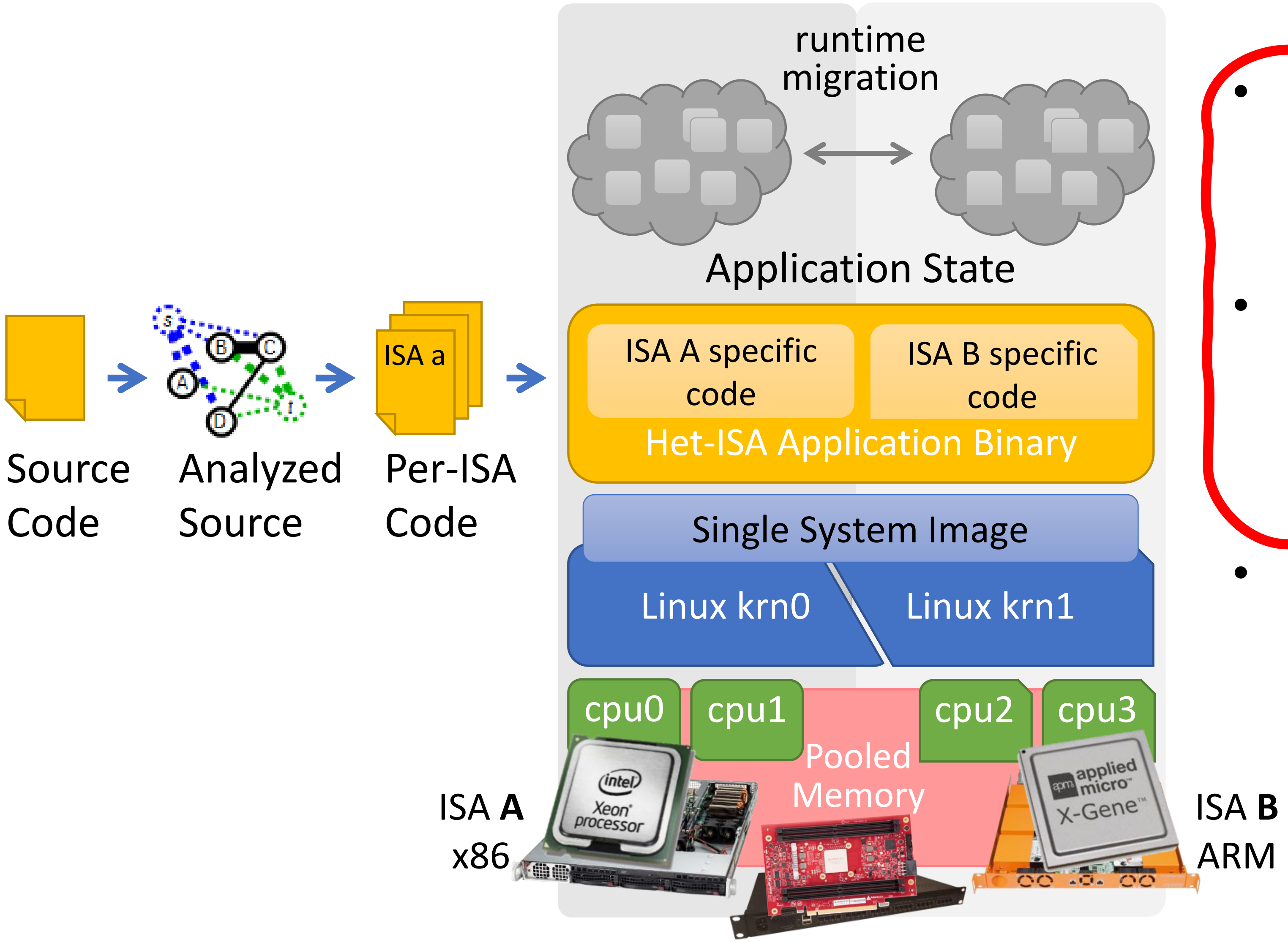
- Targets platforms with multiple groups of **general-purpose** processing units
  - **Coherent Shared Memory, or a mix of coherencies**
  - **Microarchitectural or ISA heterogeneous**

- Initial goal(s)
  - Exploring **alternatives to offloading**
    - Driven by huge-memory applications use cases
    - **SMP environment** on het-ISA coherent shared memory
  - Extend OS/Compiler to support emerging hardware



[1] "It's Time to Think About an Operating System for Near Data Processing Architectures" A. Barbalace et al., HotOS '17

Amash Linux and Compiler Framework



Source Code

Analyzed Source

Per-ISA Code

ISA a

runtime migration

Application State

ISA A specific code

ISA B specific code

Het-ISA Application Binary

Single System Image

Linux krn0

Linux krn1

cpu0

cpu1

cpu2

cpu3

Pooled Memory

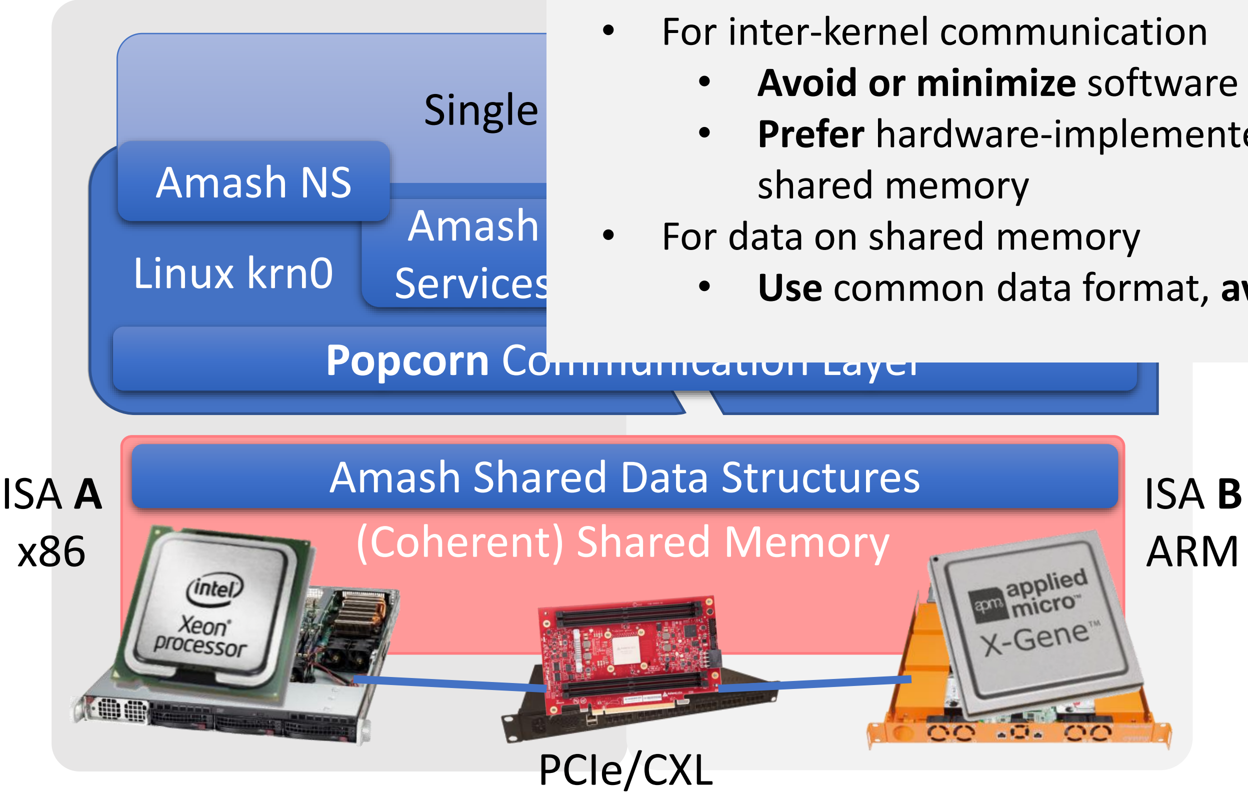
ISA **A**
x86

ISA **B**
ARM

From Popcorn

- **Runtime**
  - Runtime ISA execution migration
    - State transformation
  - Based on **musl C library**
- **Compiler** Framework
  - Offline analysis
    - Model-based code optimization
  - One binary per ISA
  - Based on **gcc/LLVM**
- Fused-kernel **Operating System**
  - One kernel per ISA
  - Distributed/Shm systems services
    - Single system Image
  - Based on **Linux**

Amash Linux – Operating System

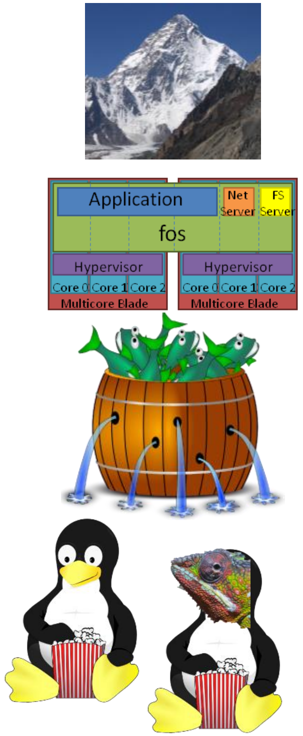


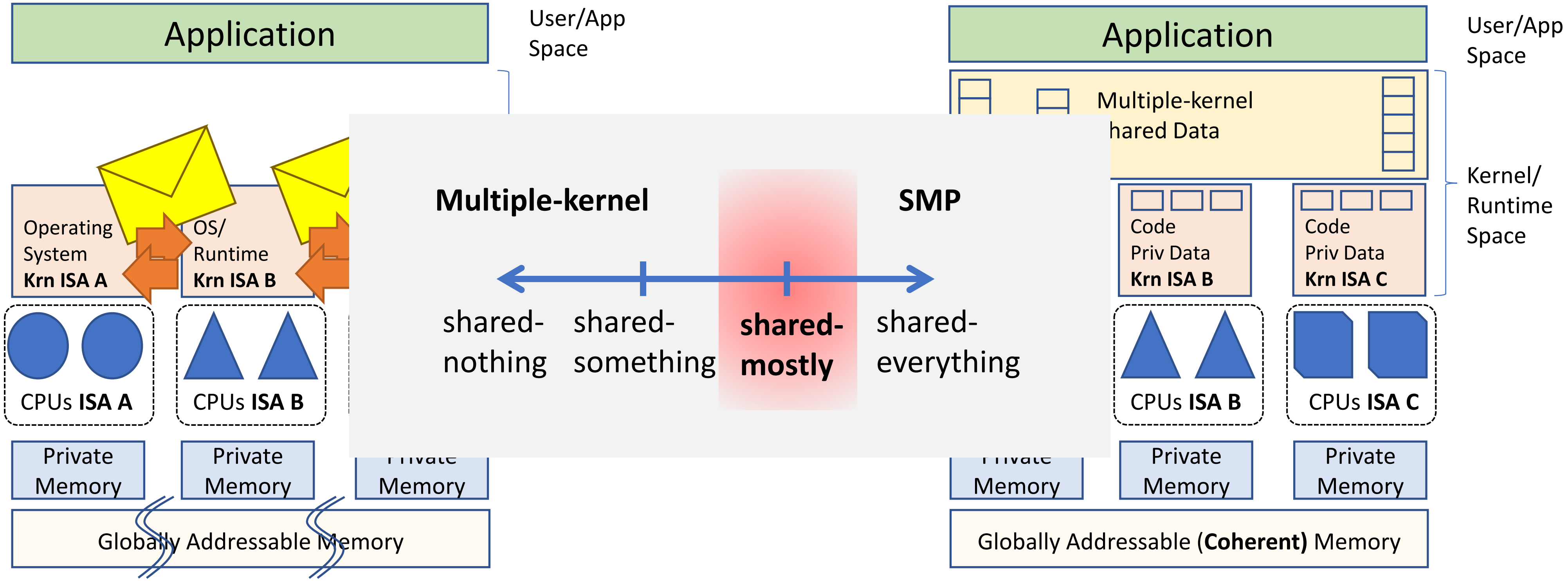
Design Principles

- For inter-kernel communication
  - **Avoid or minimize** software message passing
  - **Prefer** hardware-implemented coherent shared memory
- For data on shared memory
  - **Use** common data format, **avoid** data conversions

- **Single System Image**
  - d on Popcorn namespaces (NS)
    - xtends Linux namespaces
  - es a single operating environment
    - igrating app sees the same OS
- **ted/shm OS Services**
  - buted OS services (msg passing)
  - l OS services (shared memory)
  - ask (thread and process) migration
    - Native code migration
- **Inter-kernel Communication**
  - Performance critical component
    - low-latency and high-throughput
  - Exclusively kernel-space
  - Message passing & Shared memory
  - Single format among ISAs

Amash Linux – Multiple-kernel vs Fused-kernel





Application

User/App Space

Operating System **Krn ISA A**

OS/ Runtime **Krn ISA B**

CPUs **ISA A**

CPUs **ISA B**

Private Memory

Private Memory

Private Memory

Globally Addressable Memory

Multiple-kernel

SMP

shared-nothing

shared-something

shared-mostly

shared-everything

Application

User/App Space

Multiple-kernel hared Data

Kernel/ Runtime Space

Code Priv Data **Krn ISA B**

Code Priv Data **Krn ISA C**

CPUs **ISA B**

CPUs **ISA C**

Private Memory

Private Memory

Private Memory

Globally Addressable **(Coherent)** Memory

State-of-the-art: Multiple Kernels OS

Shared-nothing

Example distributed protocol: DSM

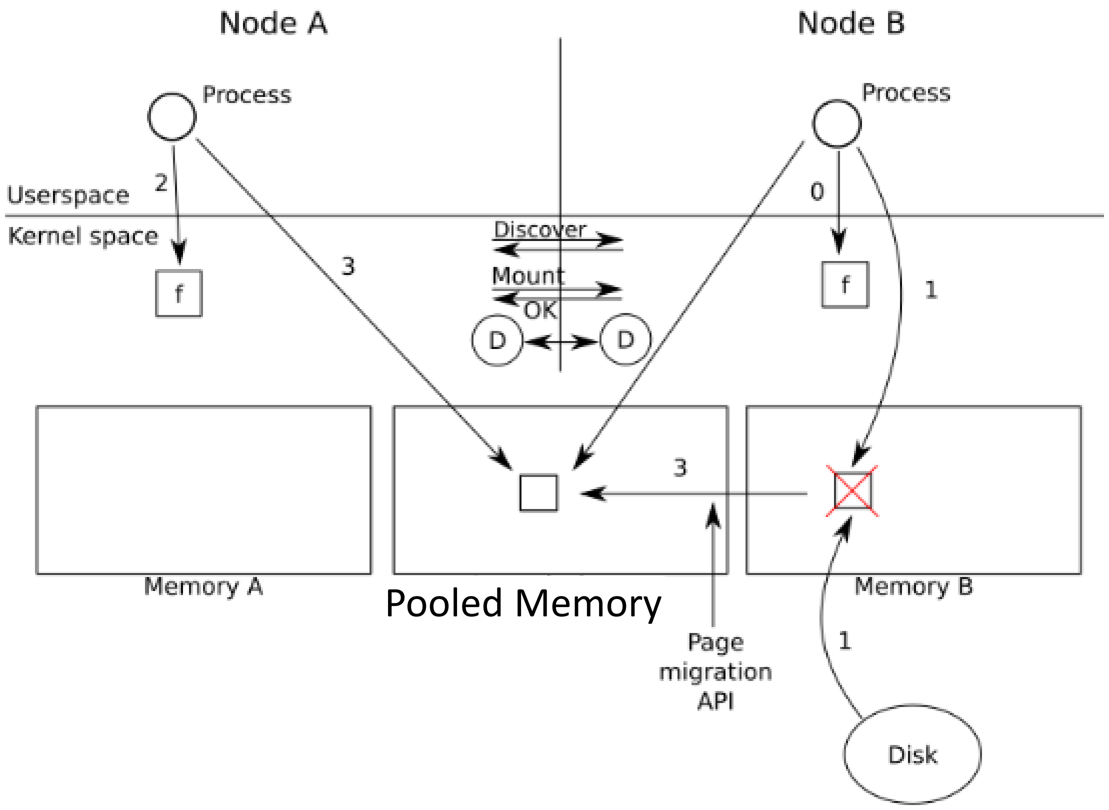
Amash Linux: Fused-kernel OS

Shared-mostly

Example shared data: VFS/Page Cache (prototype built)

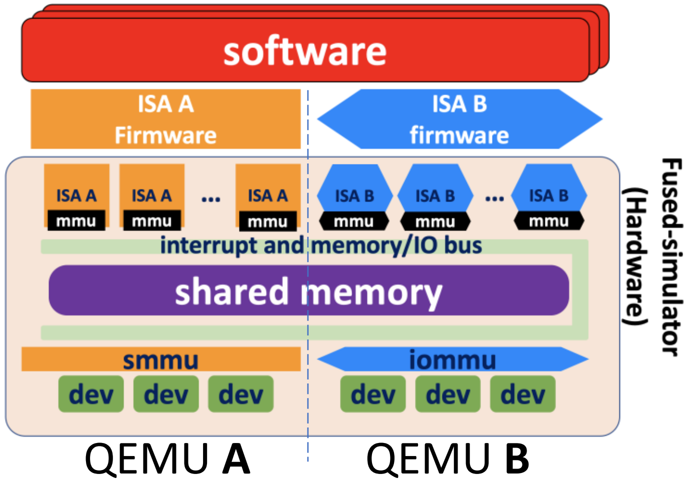
Amash Linux – VFS/Page Cache

- **Distributed VFS**
  - Based on Popcorn Linux
  - Seek pointers on (CC) SHM
- **Global page cache** on (CC) SHM
  - Uses Global Memory Allocator
  - Data pages migrated to pooled memory **on demand**
    - Pages can be accessed by any kernel
  - Data pages flushed to storage **on demand**



Amash Linux – Platform Simulator and Future Work

- **Simulator** based on QEMU (with UCSD)
  - Interconnects two QEMUs
  - Supports diverse memory latencies and consistencies
    - Same SoC, NUMA, CXL, etc.
    - Based on Multicachesim

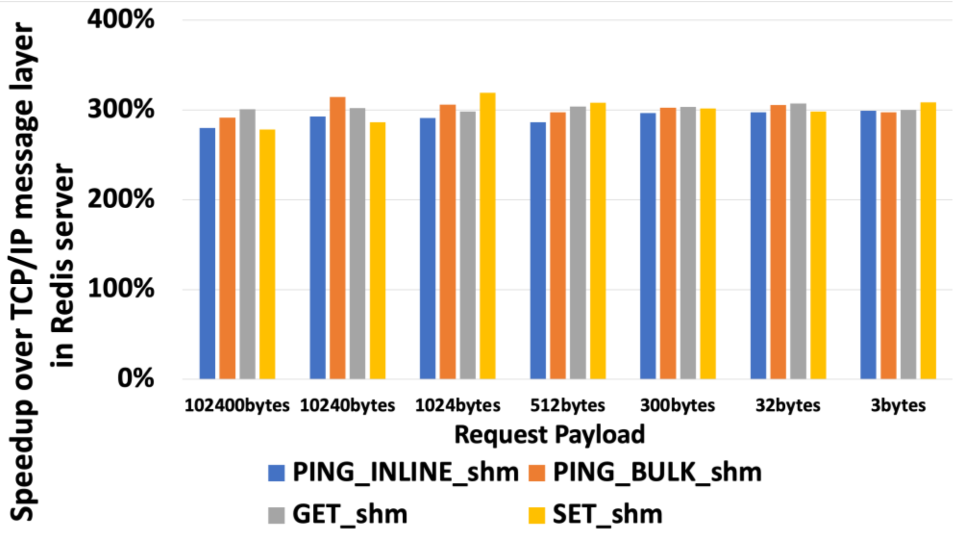


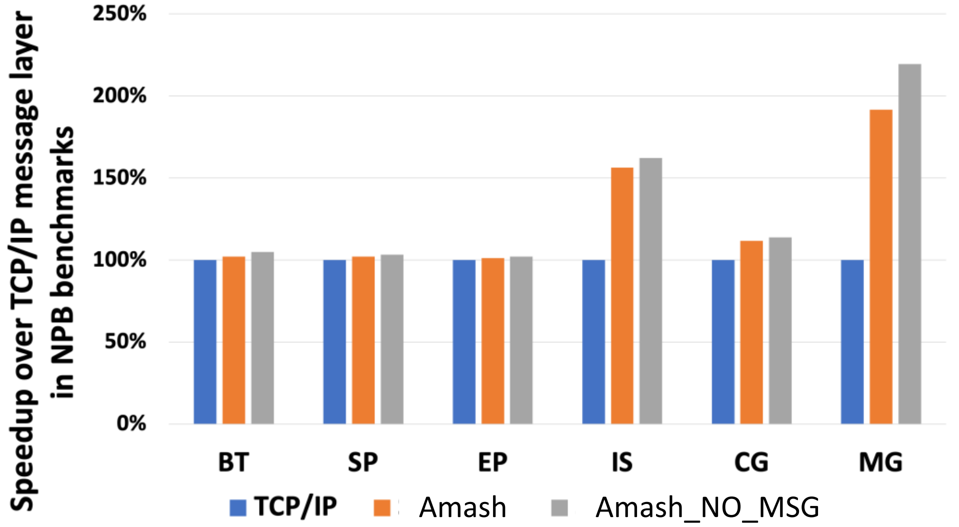
- Future Work (sketched in [1])
  - Transparent data sharing between kernel instances
    - **Typed-shared** memory support in
      - Compiler
      - Operating system

[1] "Rethinking Communication in Multiple-kernel OSes for New Shared Memory Interconnects" A. Barbalace et al., PLOS '19

Amash Linux Results

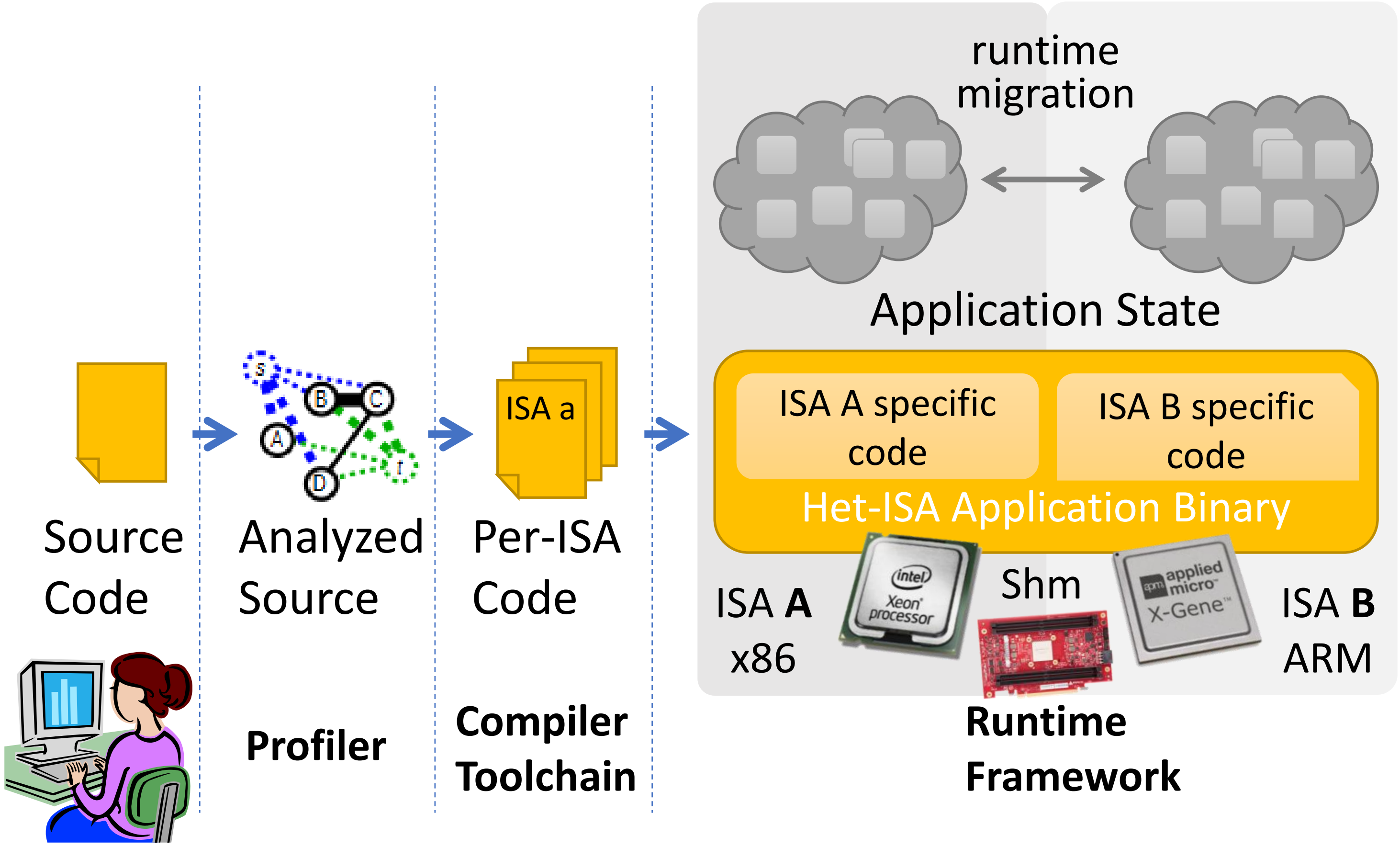
- On Cache-coherent het-ISA SoC **[1]**
- Improve performance
  - Amash **faster** than Popcorn
    - Shared memory **faster** than msg passing
    - Removes distributed protocols overheads (where possible)
- Proof of **feasibility**
  - Shared memory data structures between heterogeneous-ISA CPUs
    - Like in (homogeneous-ISA) **SMP**





[1] “Amash: Exploring Coherent Shared Memory Heterogeneous-ISA Platforms” T. Xing et al., TO BE SUBMITTED

Unificum Compiler Framework



Source Code

Analyzed Source

Per-ISA Code

runtime migration

Application State

ISA A specific code

ISA B specific code

Het-ISA Application Binary

ISA **A** x86

Shm

ISA **B** ARM

Profiler

Compiler Toolchain

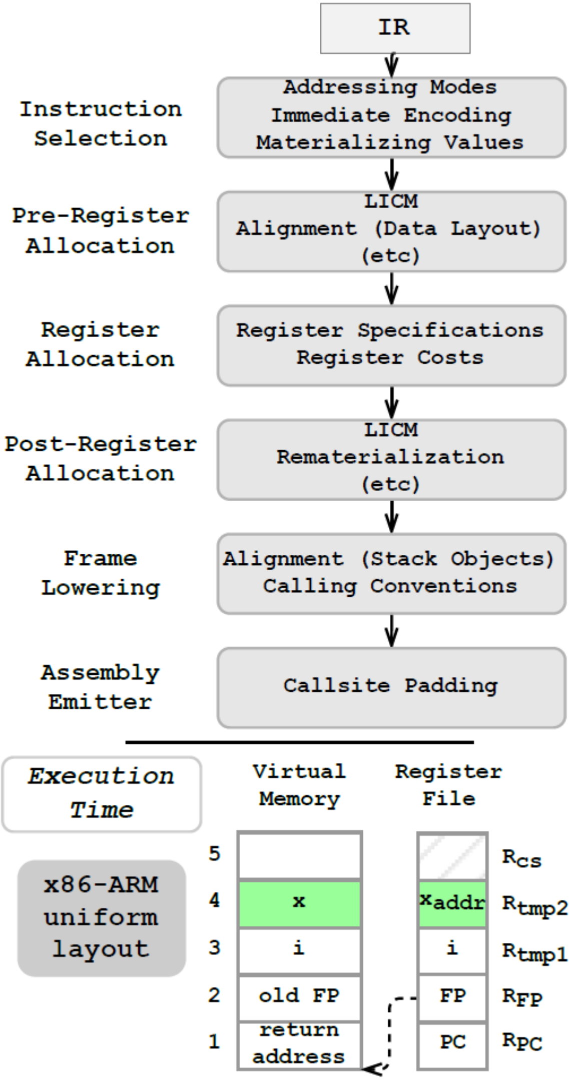
Runtime Framework

From Popcorn

- **Profiler**
  - Performance and power profiles
  - Function and sub-function granularity
  - Output performance and power code indicators
    - Affinity estimations with cost model
- **Compiler Toolchain**
  - Output heterogenous-ISA binary (native)
    - Unified address space
      - **Including TLS, heap, and stack**
    - Add migration points (func boundaries)
- **Runtime Framework**
  - Support for task migration
  - Minimal state transformation
    - **Register remapping**

Unificum Compiler

- Produces **program binaries for each ISA**
  - **Common address space**
    - Common type system (and alignments)
    - Each symbol at the same **virtual address** on any ISA
    - *No address space conversion!*
  - **Common thread-local storage** (TLS) layout
    - x86_64 layout forced
    - *No TLS conversion!*
  - **Common stack layout**
    - Based on commonalities among ISAs
    - *No stack transformation, no additional metadata!*
  - **Migration points**
    - Cannot migrate at any instruction
- Extension to the **LLVM compiler backend**



IR

Instruction Selection: Addressing Modes, Immediate Encoding, Materializing Values

Pre-Register Allocation: LICM, Alignment (Data Layout), (etc)

Register Allocation: Register Specifications, Register Costs

Post-Register Allocation: LICM, Rematerialization, (etc)

Frame Lowering: Alignment (Stack Objects), Calling Conventions

Assembly Emitter: Callsite Padding

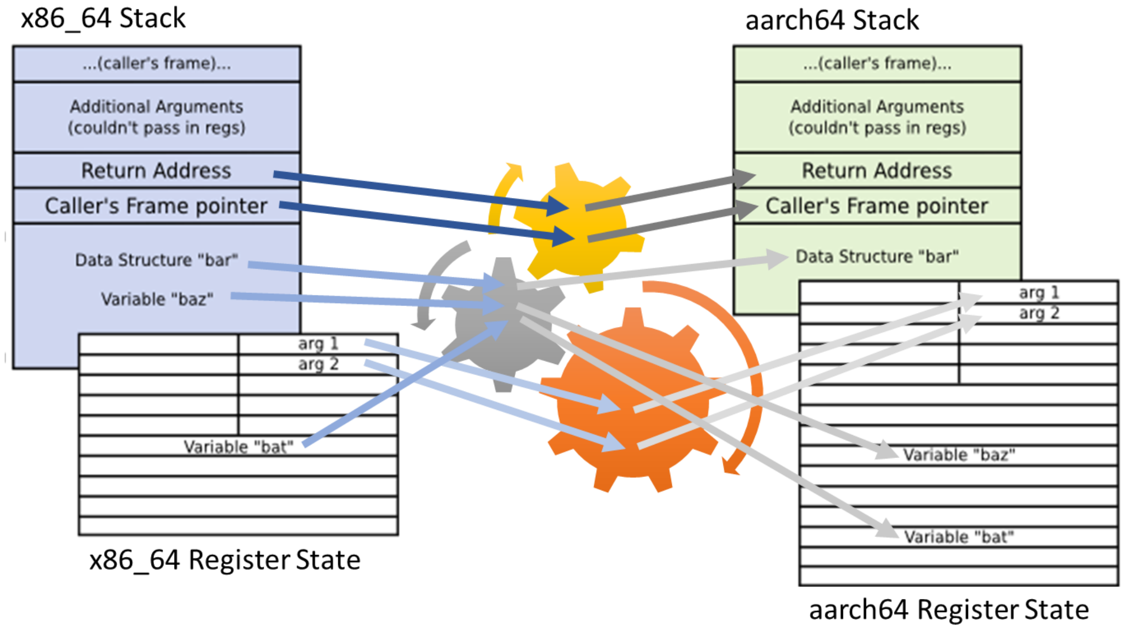
Execution Time

x86-ARM uniform layout

	Virtual Memory	Register File	
5			R_{CS}
4	x	xaddr	R_{tmp2}
3	i	i	R_{tmp1}
2	old FP	FP	R_{FP}
1	return address	PC	R_{PC}

Unificum versus Popcorn Compiler Frmwrk

- Popcorn Compiler Framework
  - **“One ABI per ISA”**
- Unificum
  - **“Single ABI across ISAs”**



x86_64 Stack

...(caller's frame)...

Additional Arguments (couldn't pass in regs)

Return Address

Caller's Frame pointer

Data Structure "bar"

Variable "baz"

arg 1

arg 2

Variable "bat"

x86_64 Register State

aarch64 Stack

...(caller's frame)...

Additional Arguments (couldn't pass in regs)

Return Address

Caller's Frame pointer

Data Structure "bar"

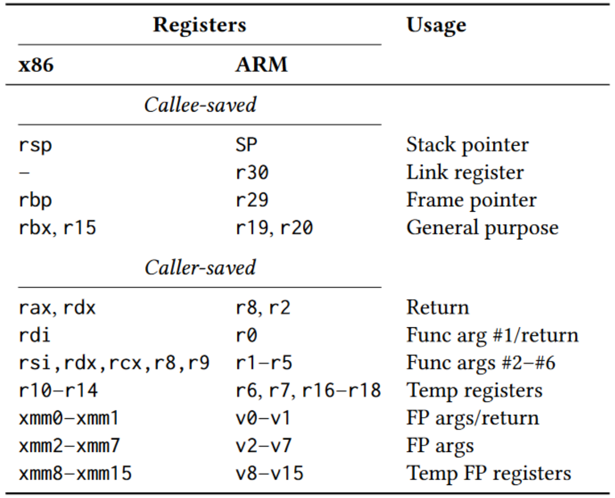
arg 1

arg 2

Variable "baz"

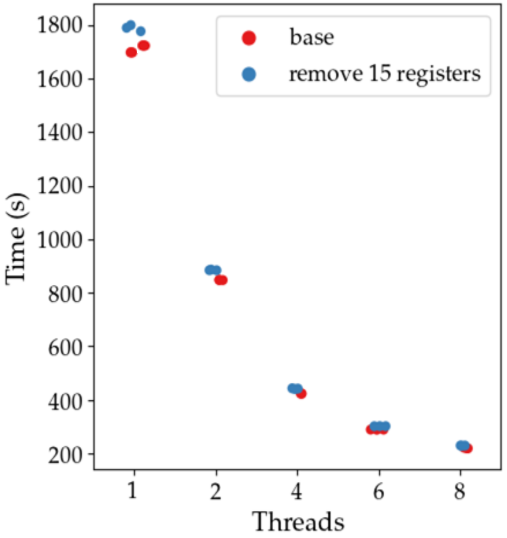
Variable "bat"

aarch64 Register State



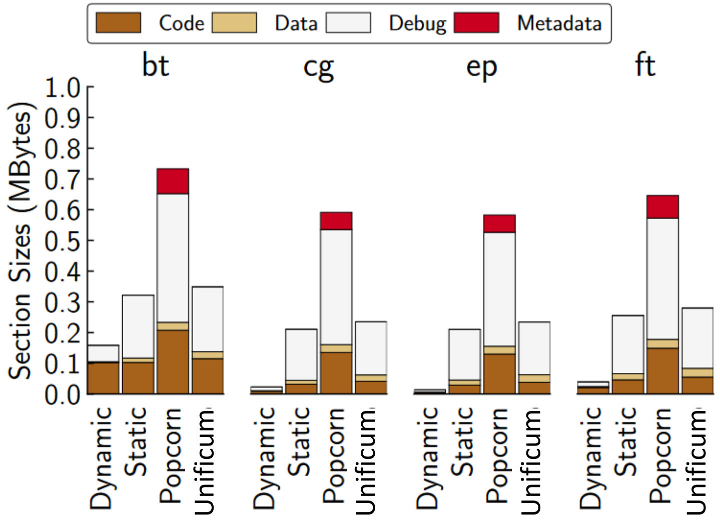
Registers		Usage
x86	**ARM**	
Callee-saved		
rsp	SP	Stack pointer
–	r30	Link register
rbp	r29	Frame pointer
rbx, r15	r19, r20	General purpose
Caller-saved		
rax, rdx	r8, r2	Return
rdi	r0	Func arg #1/return
rsi,rdx,rcx,r8,r9	r1–r5	Func args #2–#6
r10–r14	r6, r7, r16–r18	Temp registers
xmm0–xmm1	v0–v1	FP args/return
xmm2–xmm7	v2–v7	FP args
xmm8–xmm15	v8–v15	Temp FP registers

NPB BT exec time

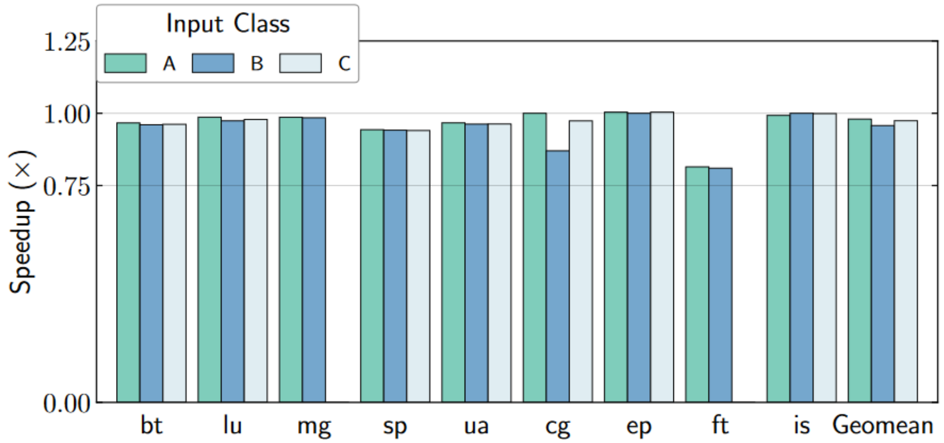


Unificum Results

- Fully-working implementation
  - Migration uses H-Containers
- No metadata in binaries
  - Fixed ISA-Register mapping
  - No other transformation
  - Minimal size increase for paddings
- **Minimal overheads** [1]
  - Multiple benchmarks
  - Single ABI restrictions introduce <2% overheads on avg
  - FT case needs more work



Binary size (x86_64)



Exec time NPB (x86_64)

[1] "The Unificum Approach to Multiple ISA Shared Memory Compilation" N. Mavrogeorgis et al., UNDER SUBMISSION

More on Heterogeneous-ISA CPUs with Coherent Shared Memory

- Work for new memory interconnects
  - **Operating systems** (current)
  - Redesign **memory subsystem** in traditional OSes
    - Automatic memory tiering
    - Provide memory caching/coherency where not supported
    - Security/partition/control
  - New **OS abstractions**
    - To connect/open a block of memory
    - To address a block of memory
  - **Compiler** (future)
  - A la Twizzler/my_plos_paper pointers
  - Rethinking Compilation/Linking

Conclusion

Thanks! Questions?

- Today’s new (heterogeneous) hardware
  - **Requires New** systems software for “better”
    - Programmability
    - Exploitability
    - *… but also accessibility, security, fault-tolerance, etc.*
  - **Key Idea:** SMP programming among heterogeneous-ISA processing units
    - With or without coherent shared memory
    - **Is possible**, and **enables flexible resource exploitation**
  - New **OS designs and abstractions**
    - Multiple-kernel and fused-kernel proved to extend traditional OSes to *het-ISA CPUs*
  - New **Compilers backends**
    - Multi-ISA, “single ABI”, compilation for zero-cost task migration among *het-ISA CPUs*

abarbala@ed.ac.uk