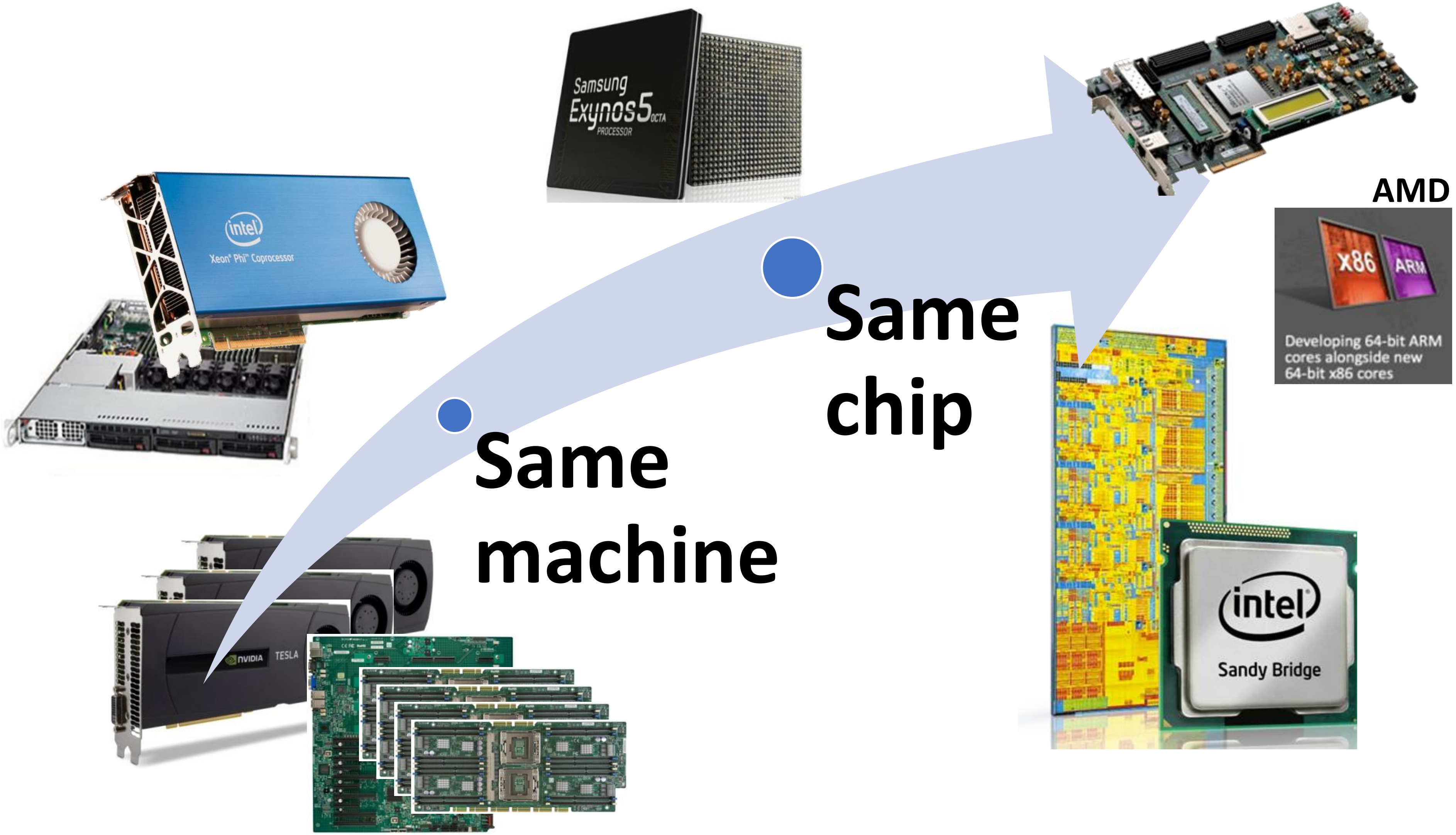
Popcorn Linux OS and Compiler Framework: lessons from 7 years of research, development, and deployments

Antonio Barbalace, Pierre Olivier, Binoy Ravindran

From my old slide sets (2013) …

Heterogeneity Trends: **Integration**

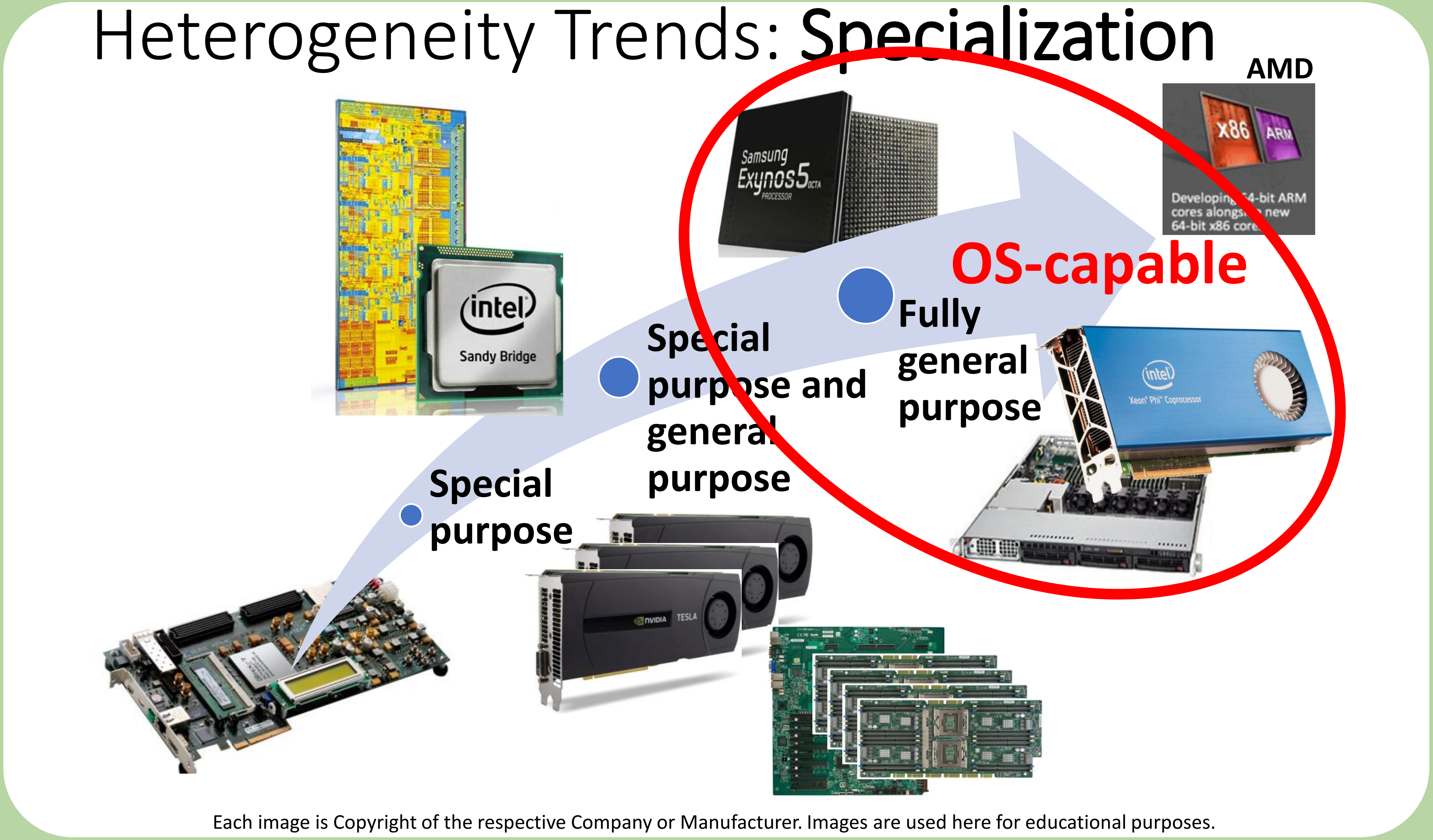


Same machine

Same chip

AMD

From my old slide sets (2013) …



Heterogeneity Trends: **Specialization**

Special purpose

Special purpose and general purpose

Fully general purpose

OS-capable

AMD

Sandy Bridge
intel
Samsung Exynos5 OCTA PROCESSOR
x86
ARM
Developing 64-bit ARM cores alongside new 64-bit x86 cores
intel Xeon Phi Coprocessor
NVIDIA TESLA

Popcorn Linux and Compiler Framework Project

- Started at Virginia Tech, Blacksburg, VA, **mid-2012**
  - Binoy Ravindran, Antonio Barbalace

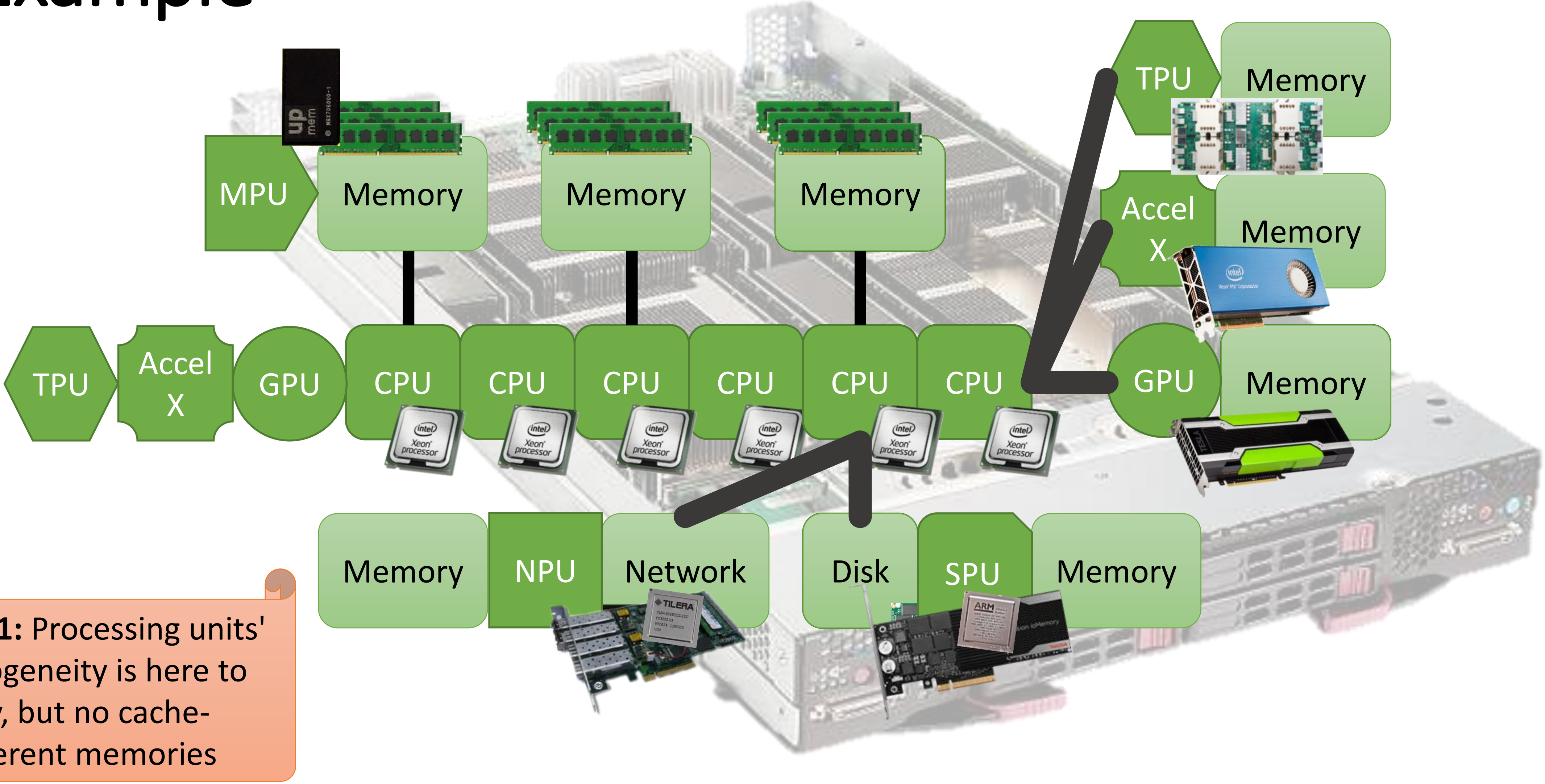
- Targets platforms with multiple groups of **general-purpose** processing units
  - **Non-cache-coherent**
  - **Microarchitectural or ISA heterogenous**

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

- OS and compiler provide **SMP functionalities** on **non-SMP platforms**

Was that worth? How to do that?

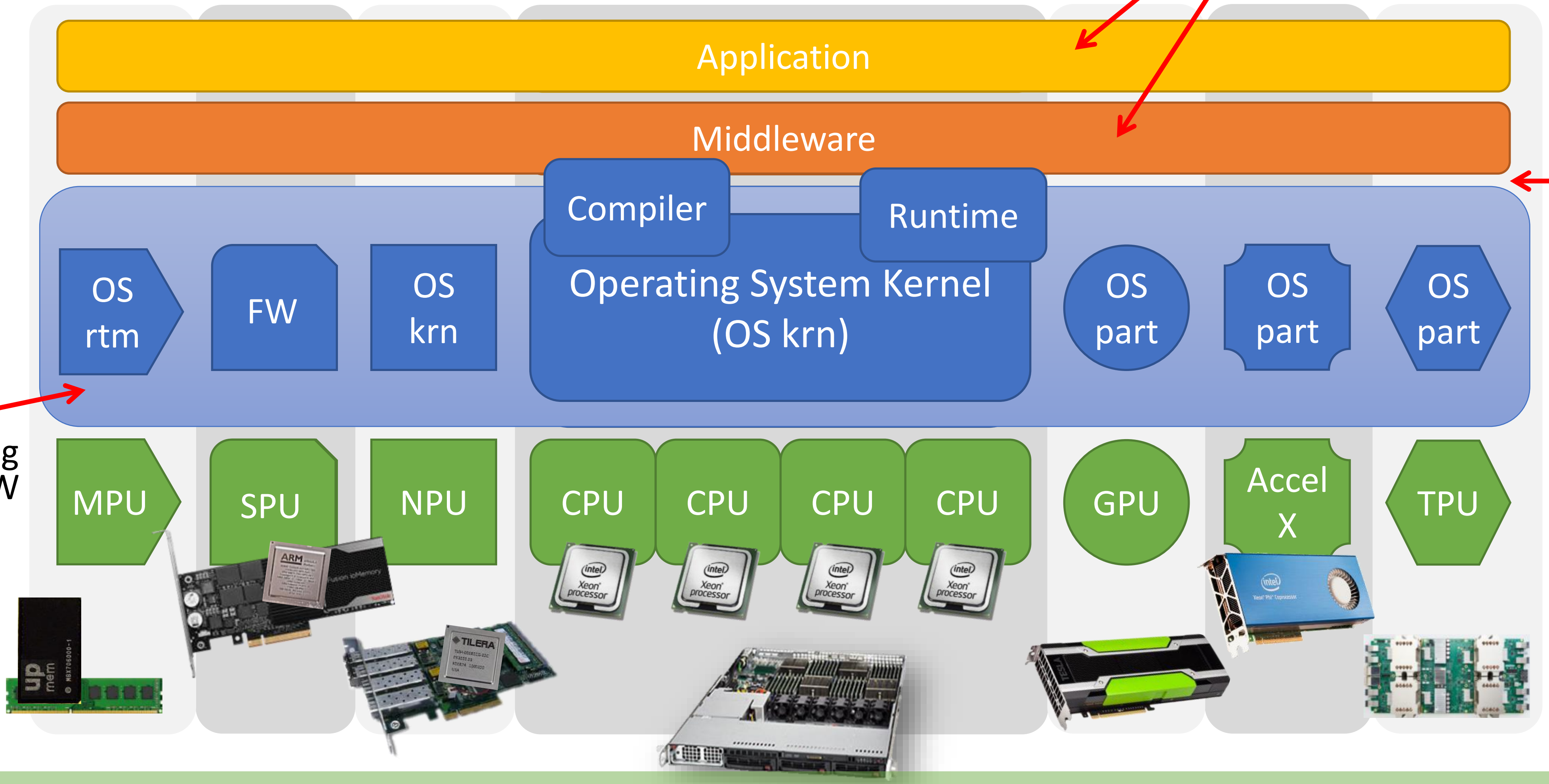
Today’s Wildly Heterogenous Hardware **Example**



Lesson 1: Processing units' heterogeneity is here to stay, but no cache-coherent memories

Popcorn Design

Program like SMP
don’t care about heterogeneity



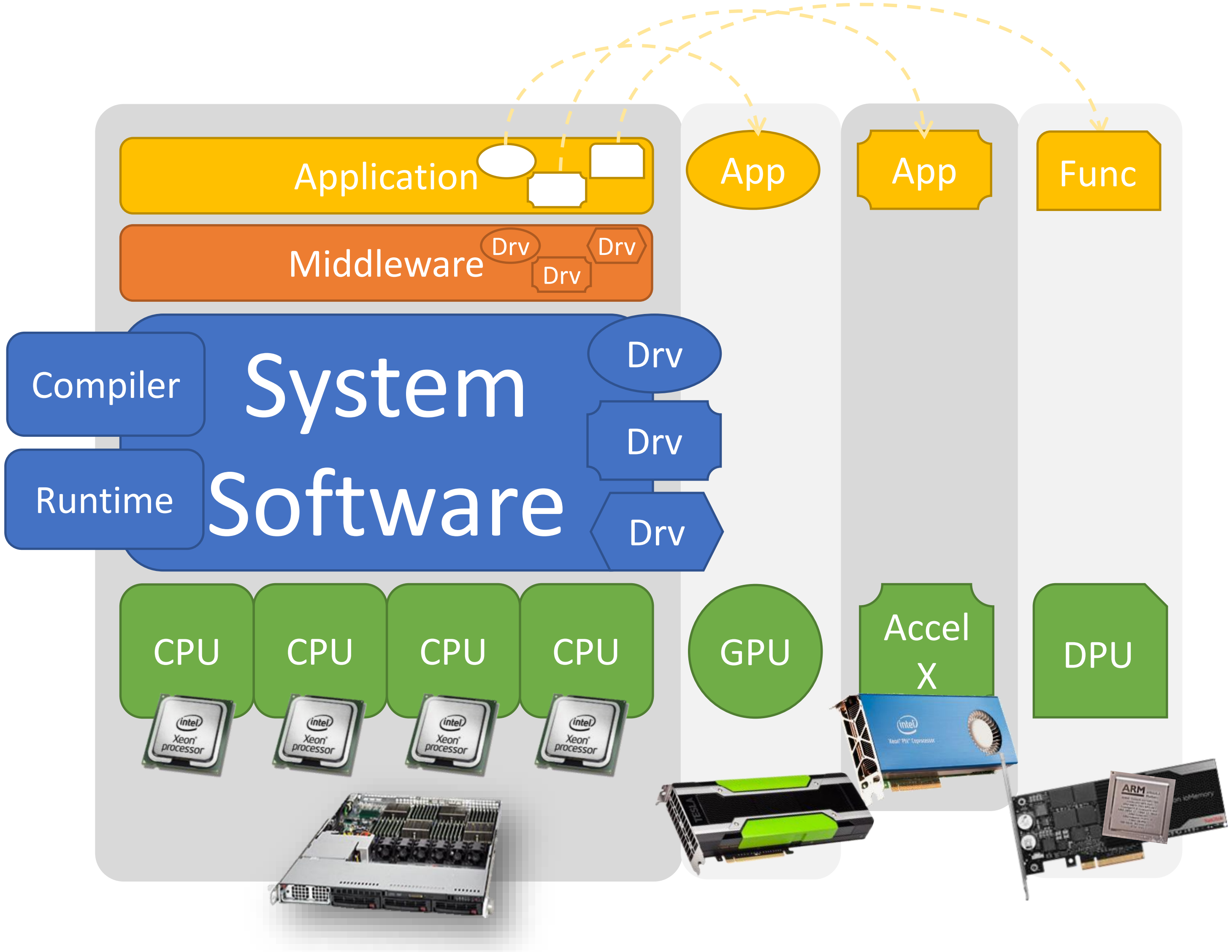
Same
interface

Multiple
communicating
OS krn/rtm/FW

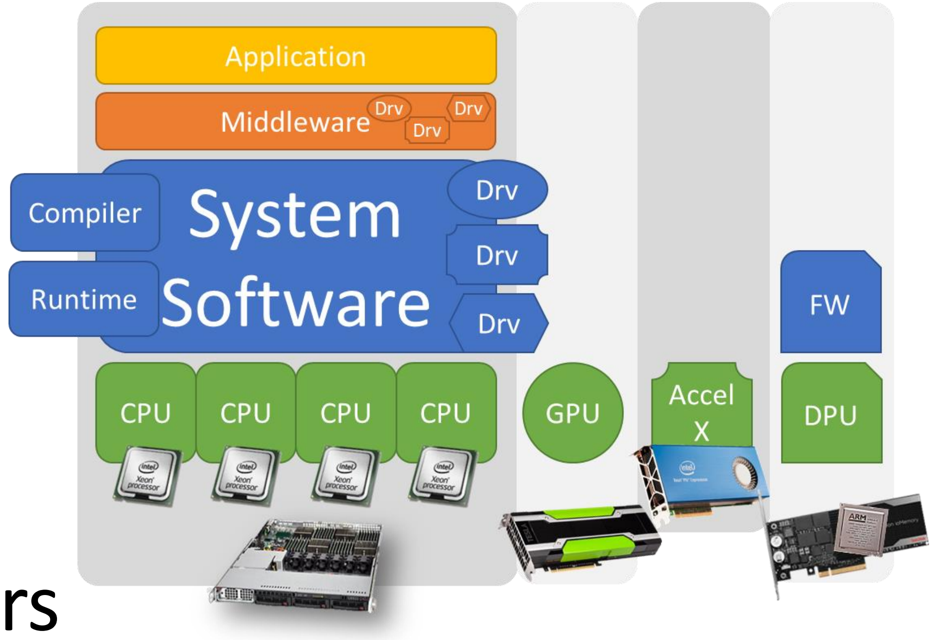
Why and how?

Classic Software for Heterogeneous Hardware

- Software runs on CPUs
- Other processing units **cannot run** the same software as the CPUs
- **Programmer (strictly) partitions** the application
- Each partition **runs only** on a predefined processing unit
- **Supporting** drivers, runtime, compilers



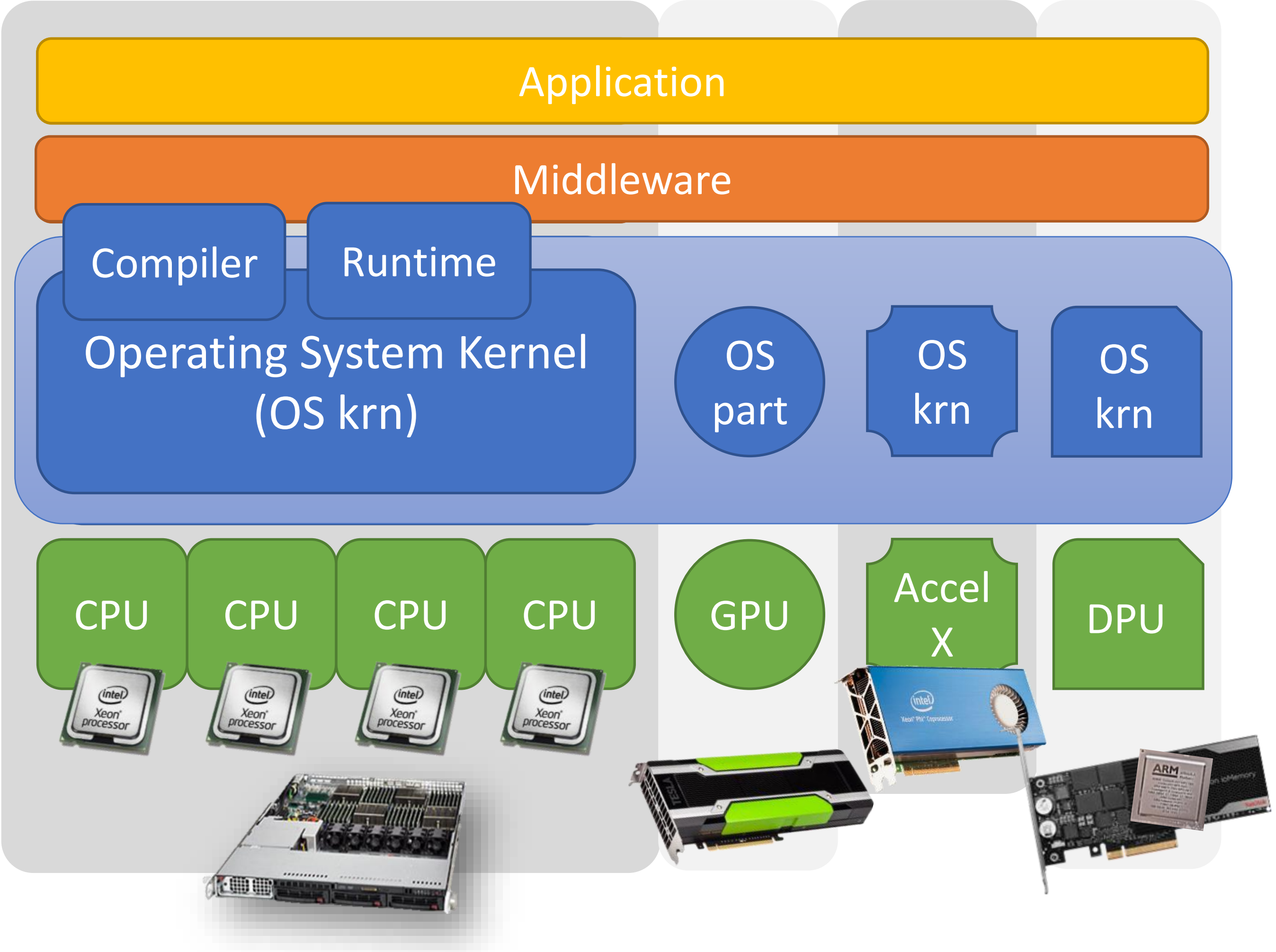
What Are the Problems?



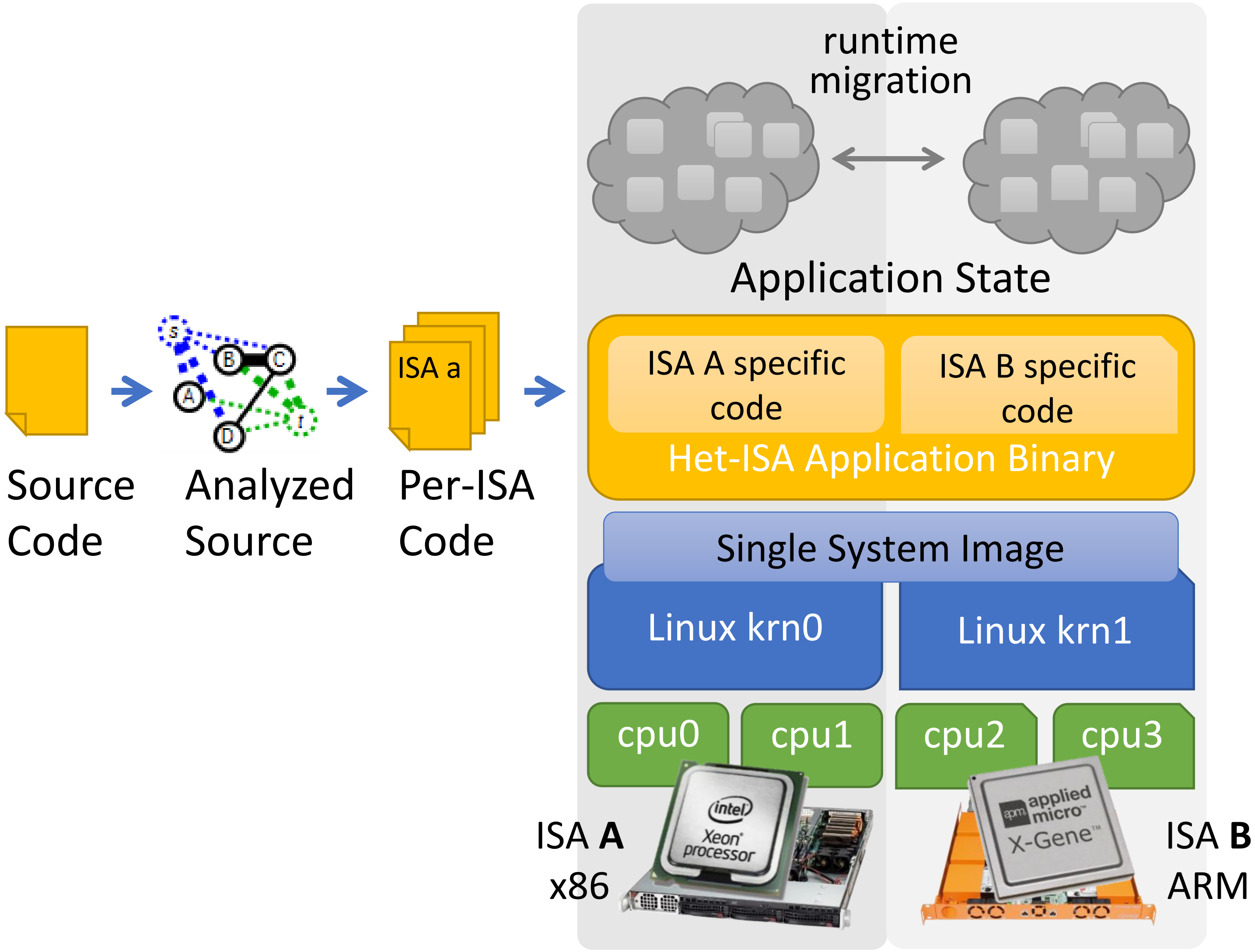
- **For each** hardware component
  - Modify **all software layers**
- **Nightmare** for application’s programmers
  - Hard to program
  - Difficult to port to a new platform
  - Poor resource utilization (performance, energy efficiency, determinism)
    - One programmer focuses on one application
    - Many applications run at the same time

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**



Popcorn Linux



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

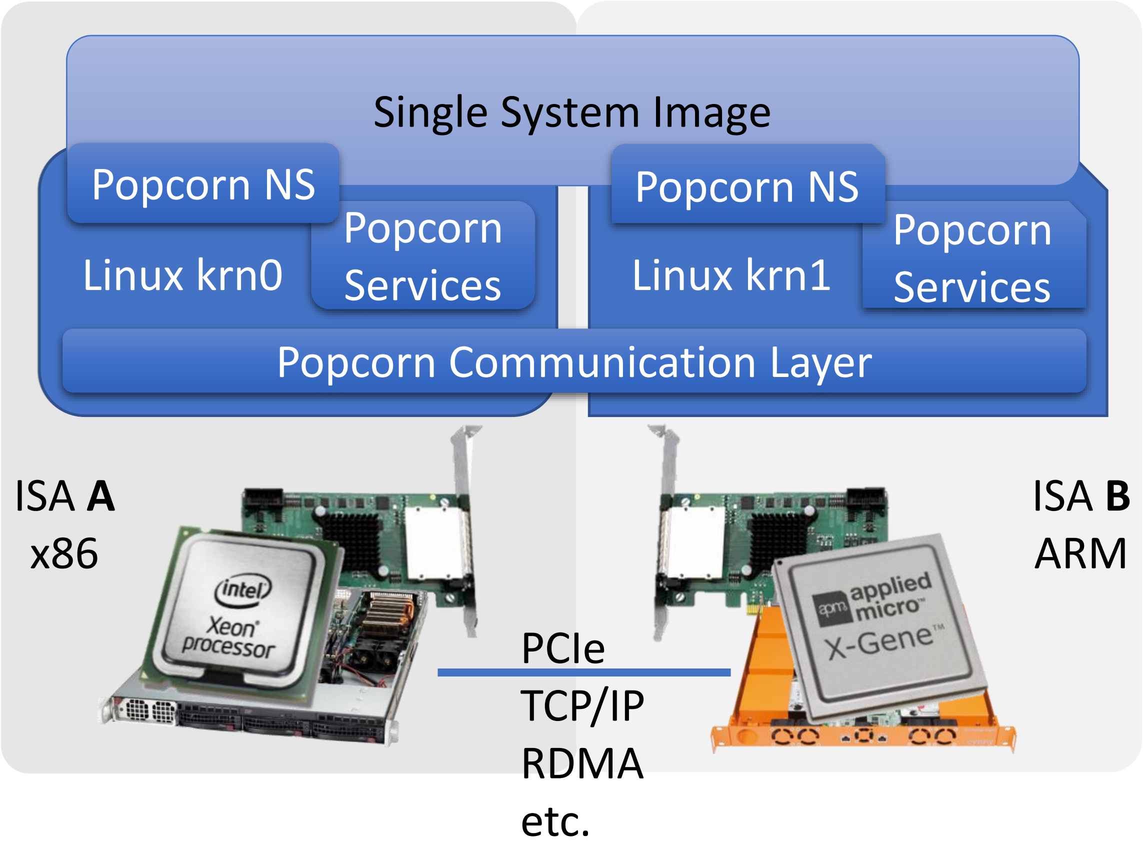
cpu0 cpu1 cpu2 cpu3

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

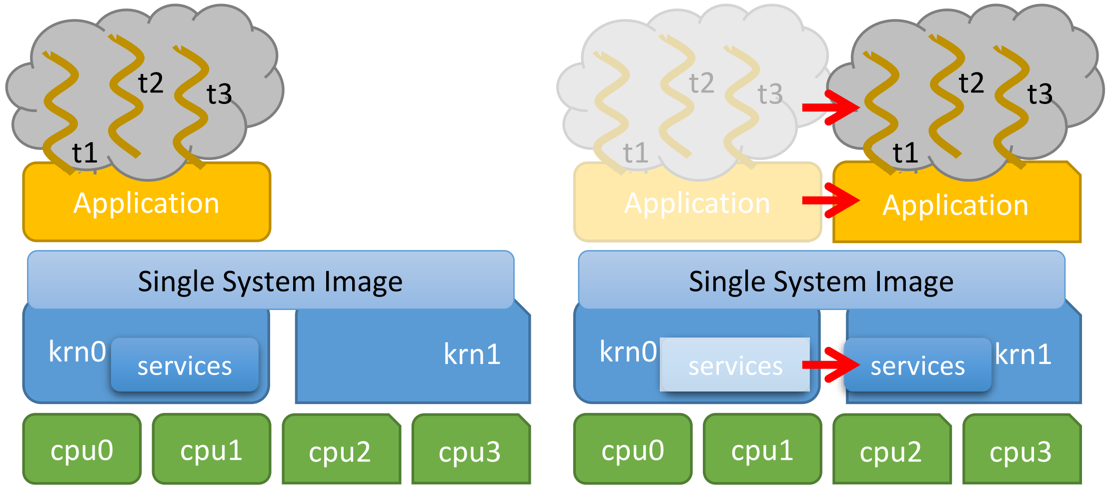


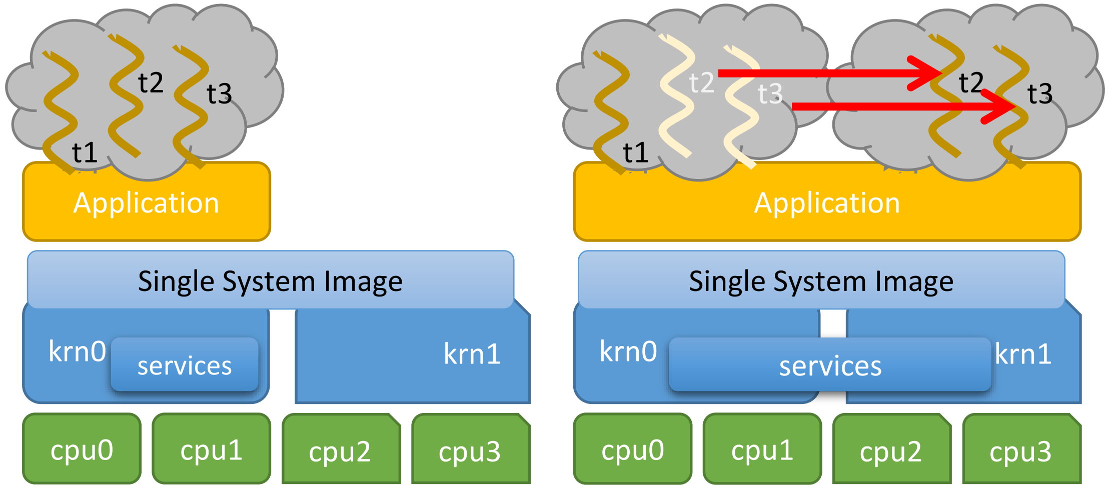
- **Single System Image**
  - Based on Popcorn namespaces (NS)
  - Creates a single operating environment
    - Migrating app sees the same OS
  - Extends Linux namespaces
- **Distributed OS Services**
  - Task (thread and process) migration
    - Native code migration
  - Distributed memory management (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

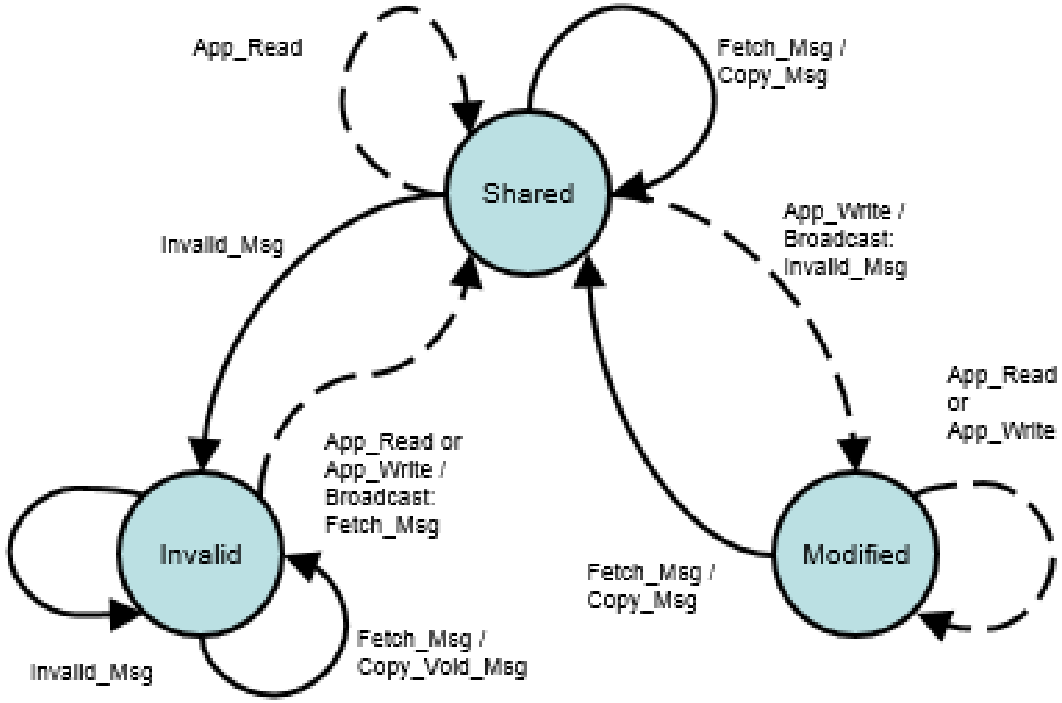
- **Thread** Migration
- Selected threads are transferred
  - Threads' state is transferred
- Kernels coordinate to maintain application state consistent



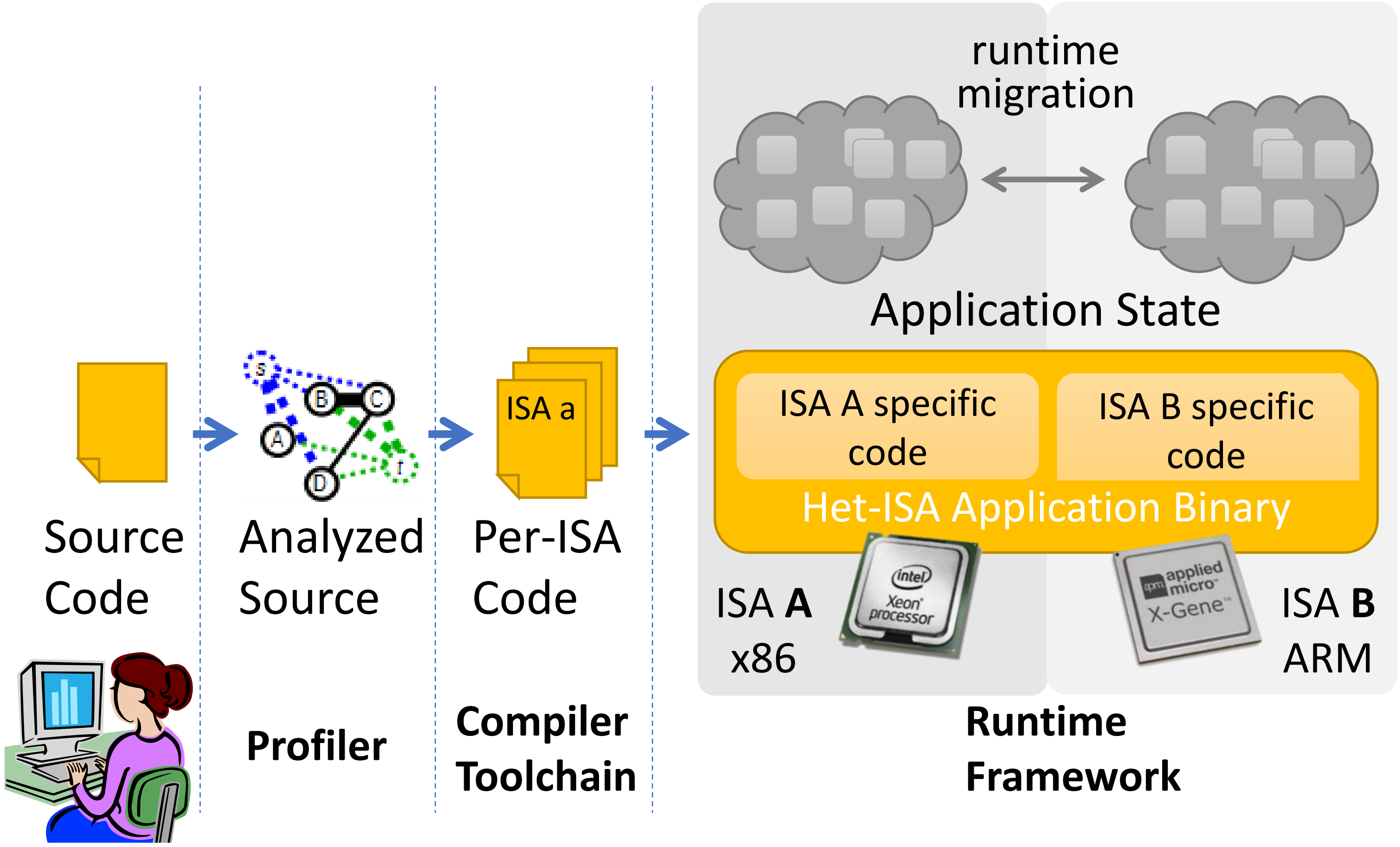


Popcorn Linux – Thread Migration’s **DSM**

- 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



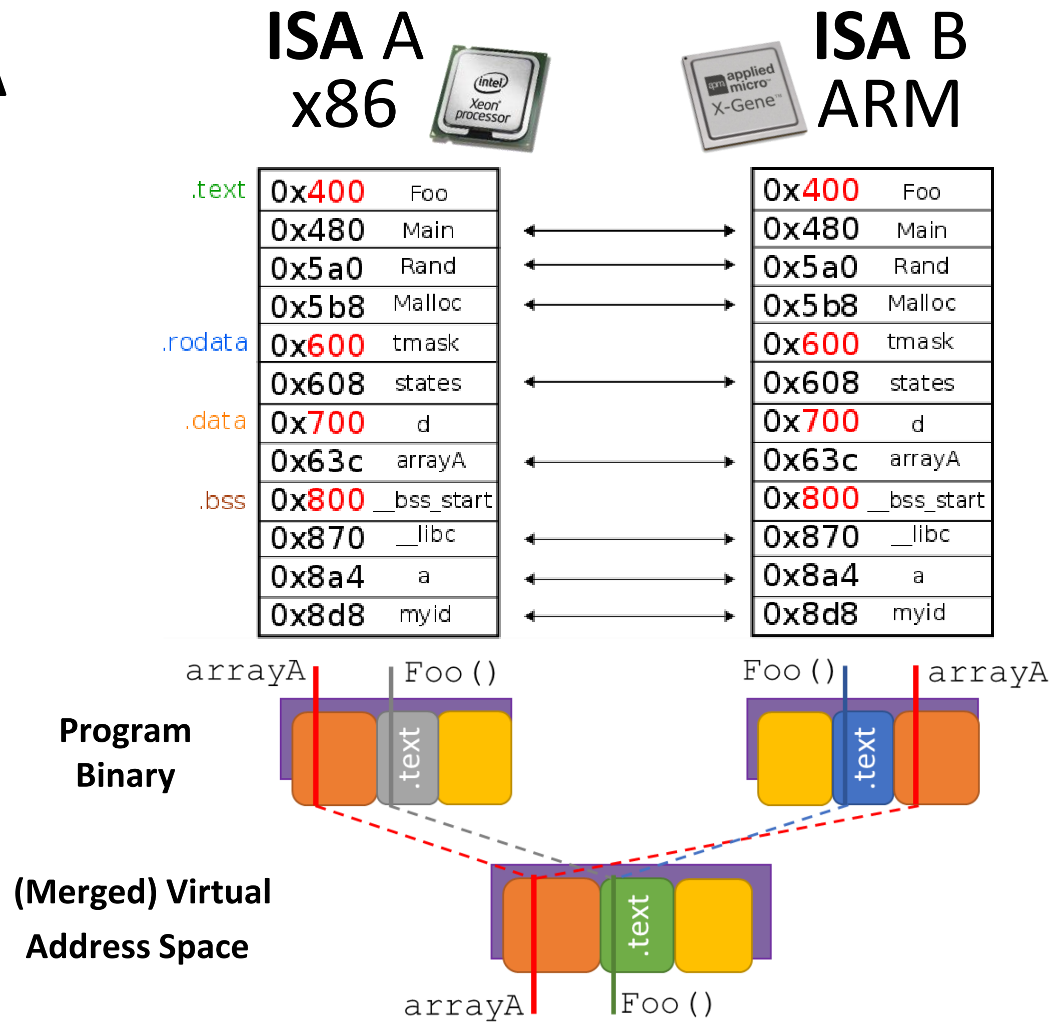
Popcorn Linux – Compiler/Runtime



- **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)
    - Insert migration points (fun 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
    - 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



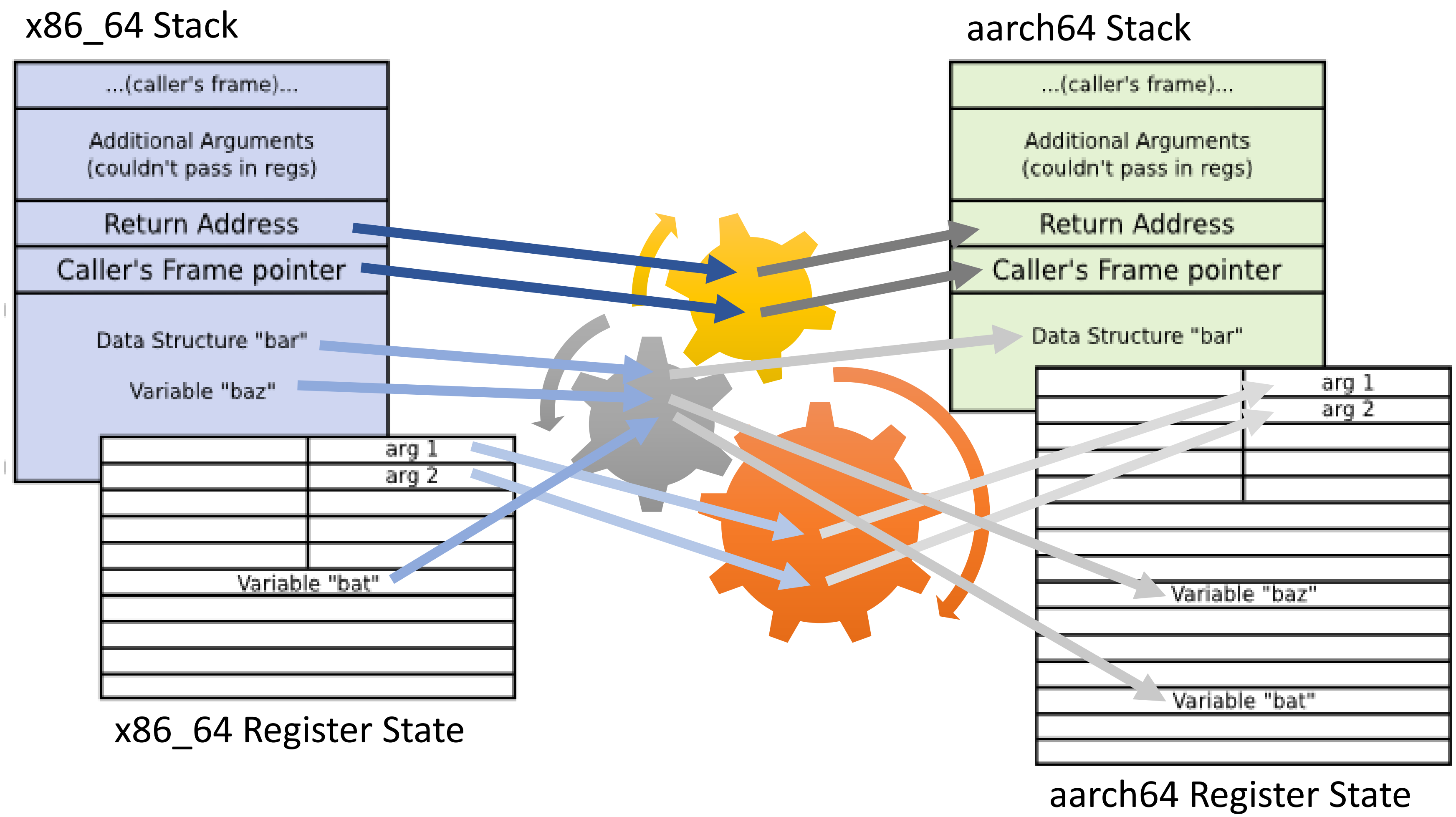
ISA A x86

ISA B ARM

	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



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"

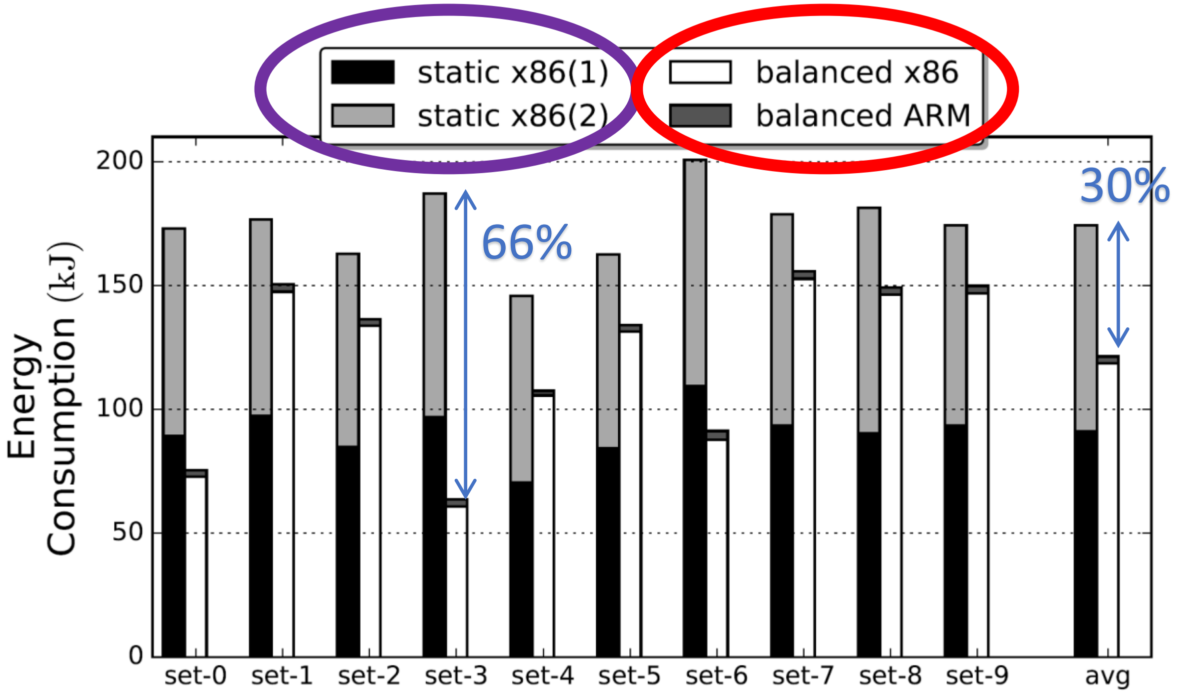
aarch64 Register State

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

First 5 years of the project in Summary

- Gigantic Engineering Effort
- Operating Systems
  - **Multiple kernels Linux**
  - Repurpose monolithic Linux kernel as a message-passing kernel
  - Convert Linux’s subsystems from SHM to SHM+message-passing
- Compiler/Linker
  - **Common address space layout, per-ABI stack layout**
  - Compile into different ISA binaries with LLVM/gold
  - Insert equivalence points at which stacks can be converted (stackmaps)
- Runtime Library
  - **Extended standard library (based on muslc)**
  - Provide “builtin” functions to convert and migrate at eq points

Lesson 2: very complex to build and debug because development affects several software layers

Lesson 3: instead of Linux, Darwin or DragonFly BSD may have reduced development time

Lesson 4: LLVM as a cross-compiler saved a lot of time, and muslc supports a large amount of apps

Feedback from Industry and Academia #1

- **Constraining** dependencies
  - Need application **source-code**
    - Eventual code modifications
    - and compiler script rewriting
  - Must use Popcorn Linux **Compiler Framework**
    - Specific version of LLVM
    - Specific version of musl C library
  - Must use Popcorn **Linux kernel**
    - Few kernel versions and CPU architectures supported
    - Limited POSIX support
      - Not all Linux subsystems supported

Lesson 5: for production apps, that use hacks for performance, transparency is hard to provide

Lesson 6: impossible to keep up with upstream developments – fix one version

Lesson 7: adding a new CPU architecture may be incompatible with previous assumptions (32bit?)

Lesson 8: cannot support all Linux subsytems, need automatic way to convert subsystems into SHM+MSG

Feedback from Industry and Academia #2

- **Limiting** factors
  - **Not well integrated** in the Linux kernel nor in LLVM
    - Requires Linux kernel patching
    - Requires LLVM patching
  - Doesn’t support **dynamically compiled code**
    - Including JIT, self-modifying, etc.
    - E.g., Java, .NET
  - Restricted **library support**
    - Doesn’t support dynamic libraries
    - Cannot migrate in library-code (if not recompiled)
  - Supports **application/container** migration
    - Doesn’t generalize to VMs

Lesson 9: Implement functionalities in modules or plugins to minimize patching

Lesson 10: for dynamically compiled code, need to control the way code is generated

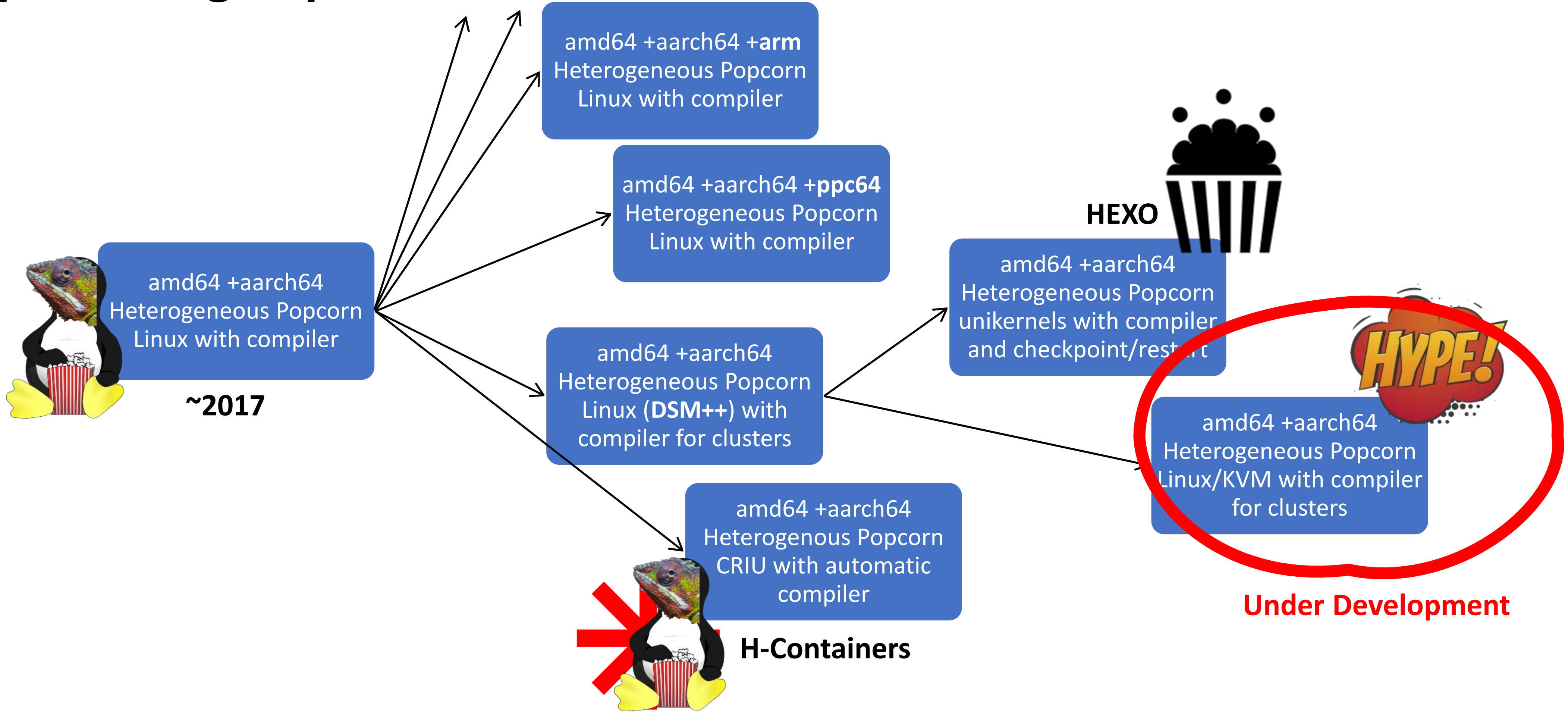
Lesson 12: containers/namespaces nice abstraction for migration

Lesson 11: a more generic techniques is needed to runtime migration among VMs (Popcorn relies on the syscall abstraction)

List continues …

The latest 2+ years …

- **Keep evolving Popcorn**



amd64 +aarch64
Heterogeneous Popcorn
Linux with compiler

~2017

amd64 +aarch64 +**arm**
Heterogeneous Popcorn
Linux with compiler

amd64 +aarch64 +**ppc64**
Heterogeneous Popcorn
Linux with compiler

amd64 +aarch64
Heterogeneous Popcorn
Linux (**DSM++**) with
compiler for clusters

amd64 +aarch64
Heterogenous Popcorn
CRIU with automatic
compiler

H-Containers

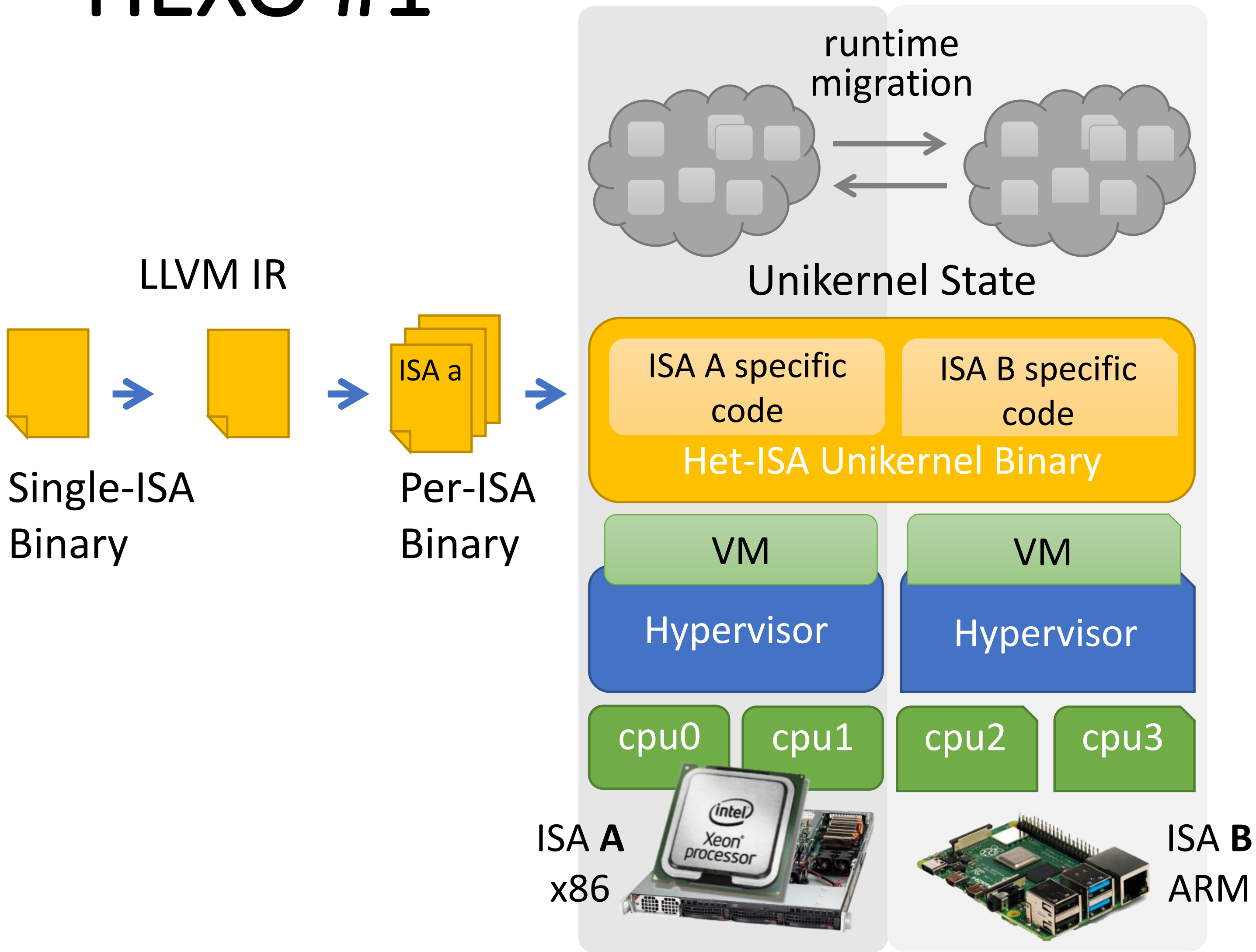
HEXO

amd64 +aarch64
Heterogeneous Popcorn
unikernels with compiler
and checkpoint/restart

amd64 +aarch64
Heterogeneous Popcorn
Linux/KVM with compiler
for clusters

Under Development

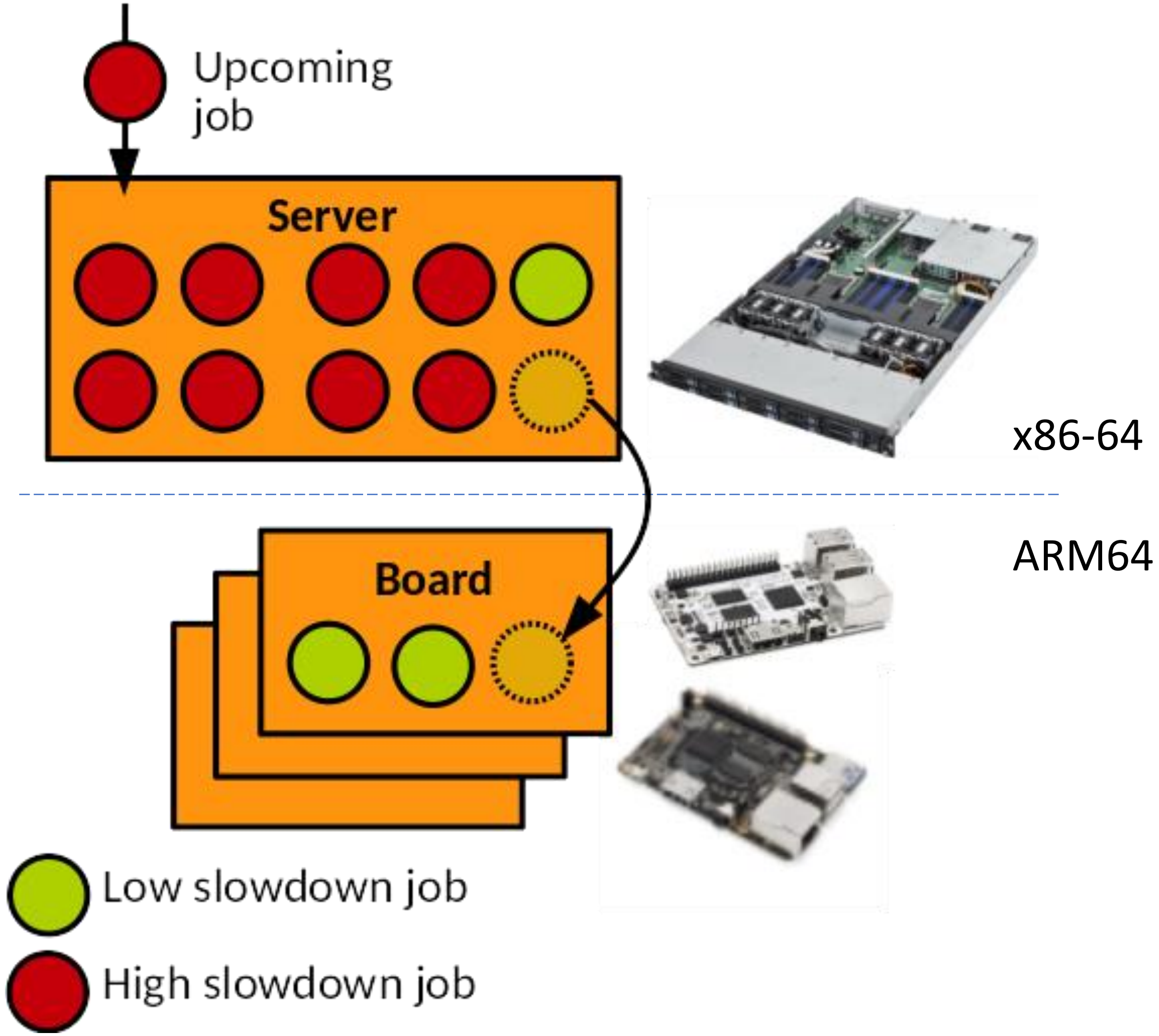
HEterogeneous eXecution Offloading
HEXO #1



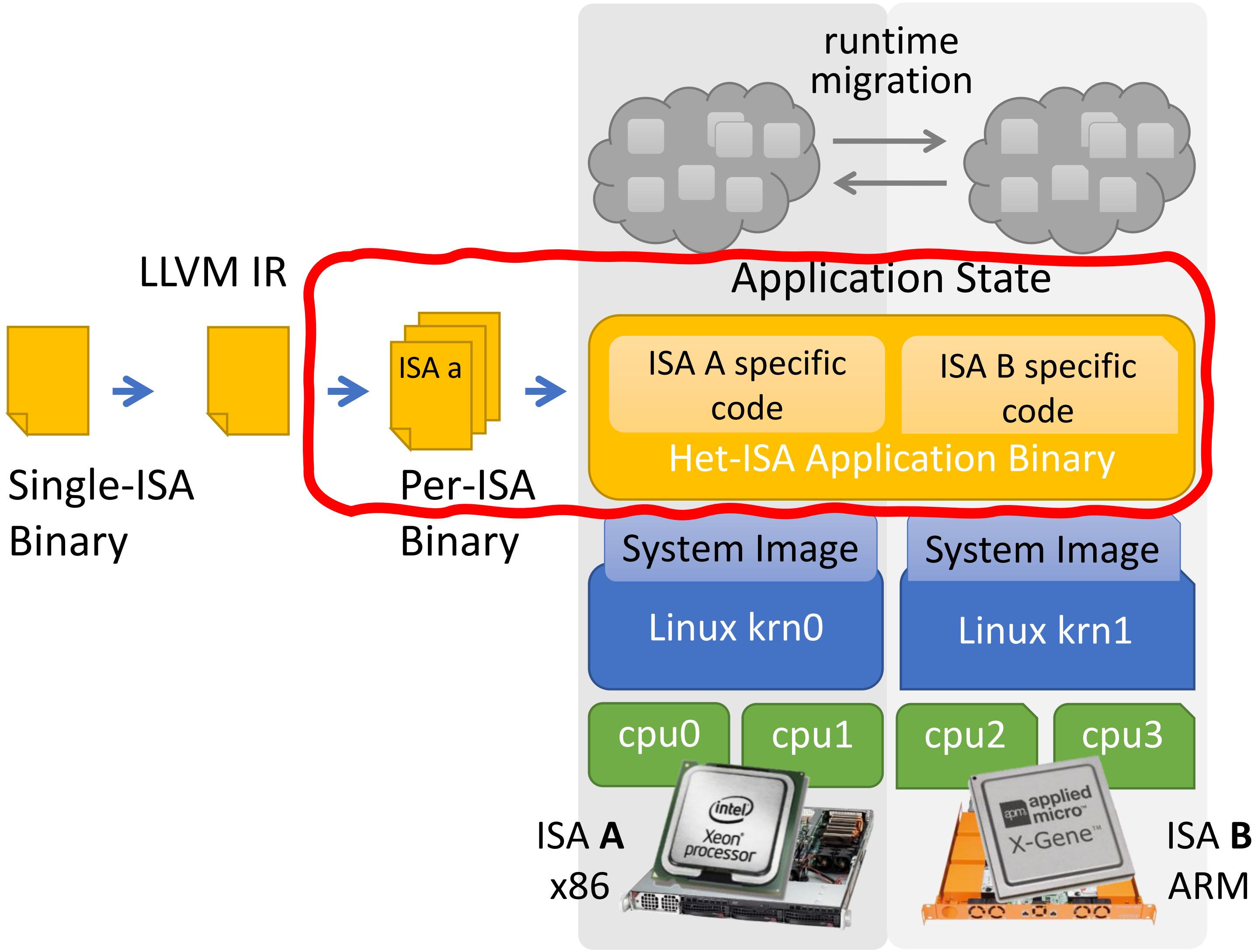
- **Runtime**
  - Unikernel-level checkpoint
  - libOS code is per-ISA
    - Substituted at runtime
- **Compiler** Framework
  - One binary per ISA
    - Including libOS
  - Based on **gcc/LLVM**
- Migration-aware **Hypervisor**
  - One hypervisor per ISA
  - Migration service
    - Aware of the migrating unikernel
  - Based on **Linux/KVM**

HEterogeneous eXecution Offloading
HEXO #2

- **HEXO migrates at runtime** compute-intensive background jobs
- **From fast & expensive** x86-64 servers **to slow and cheap** ARM64 embedded boards
  - Uses Popcorn state transformation
  - Lightweight VMs (**unikernels**) as unit of execution
- **Slowdown from running on the board is highly variable**
  - Profiles jobs at runtime on the server
  - Offloads the ones with the smallest **estimated** slowdown



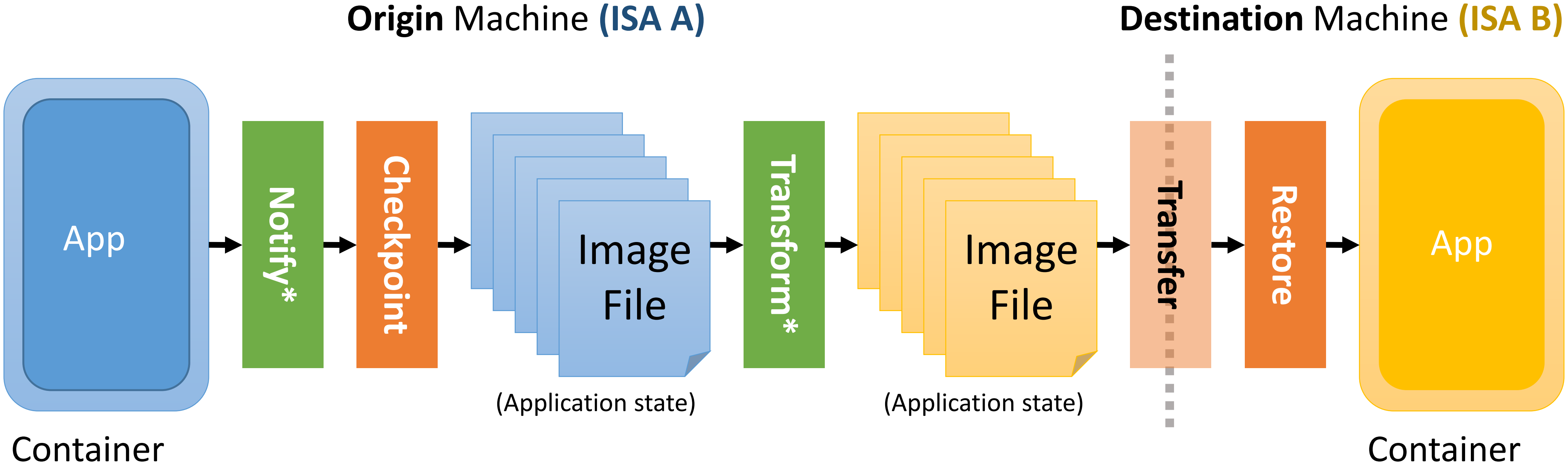
H-Containers



- **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

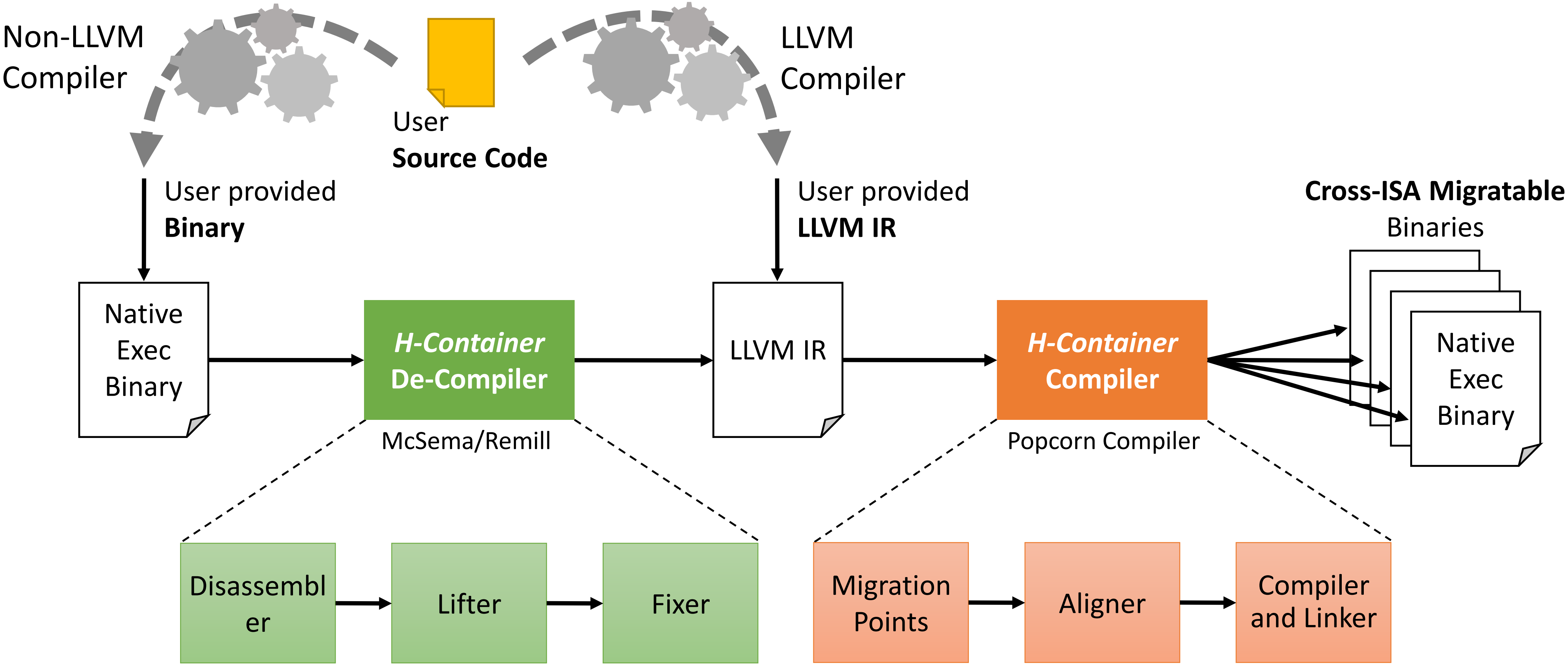
H-Container – Runtime

Checkpoint/Restart Migration



**New Components*

H-Containers – Transpiler



Summary

Thanks! Questions?

- Computing platforms with multiple groups of processing units are **here to stay**
  - **Non cache-coherent**
  - **Microarchitectural or ISA heterogeneous**

- Can be programmed as (homogenous) **SMP platforms – hence, easily!**
  - By means of **new systems software (Popcorn Linux and Co)**
    - Common OS interface and transferrable OS state
    - Common address space layout and format/type/padding
  - **Transforming how we are building software today**
  - Tested on open-source **real-world system software**
    - **Several lessons learned** in the process
      - We are not in the early days of computing – gigantic amount of work to modifying all SW layers
      - Hard to keep up with upstream developments
      - etc.

abarbala@ed.ac.uk