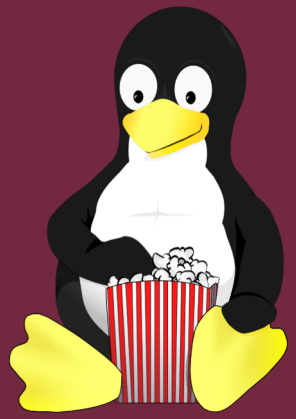


Popcorn Linux:
System Software for Heterogeneous Hardware

Sang-Hoon Kim

Postdoctoral Associate

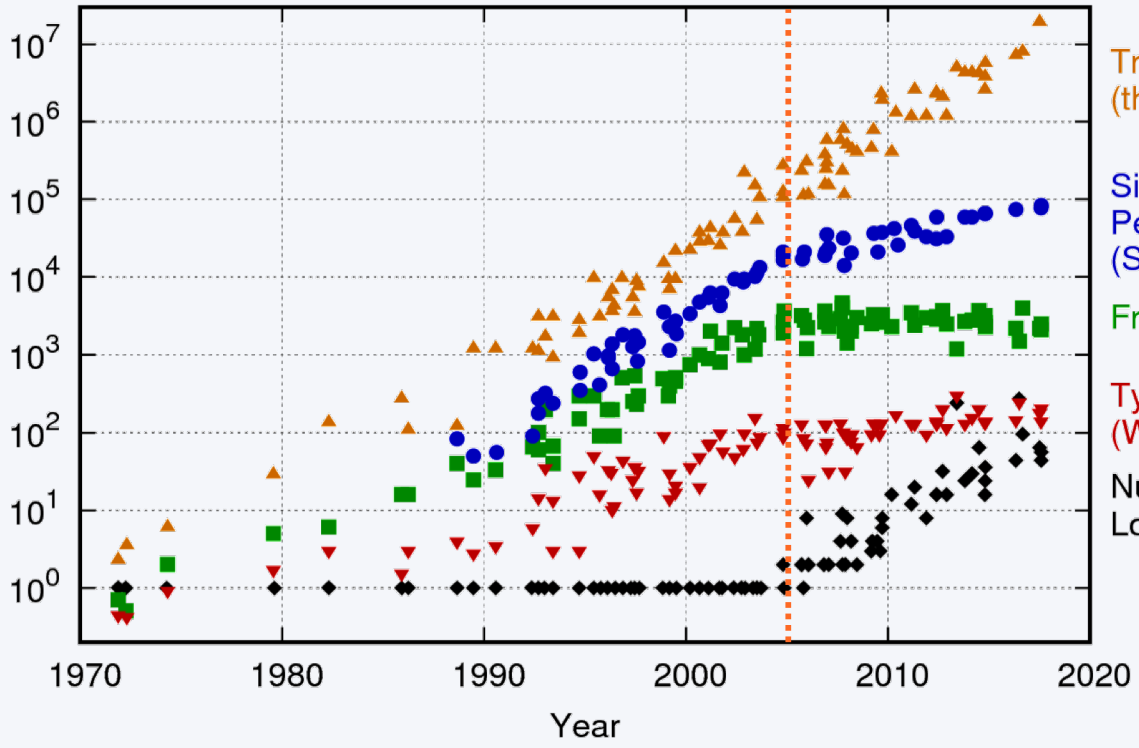
Systems Software Research Group

VIRGINIA TECH™

May 25, 2018

Trend towards heterogeneous systems

- Clear that microprocessor trends have shifted since 2005



Transistors (thousands)

Single-Thread Performance (SpecINT x 10^3)

Frequency (MHz)

Typical Power (Watts)

Number of Logical Cores

Limited single thread performance
- Thermal and power budget
- Dark silicon effect

Increase core counts

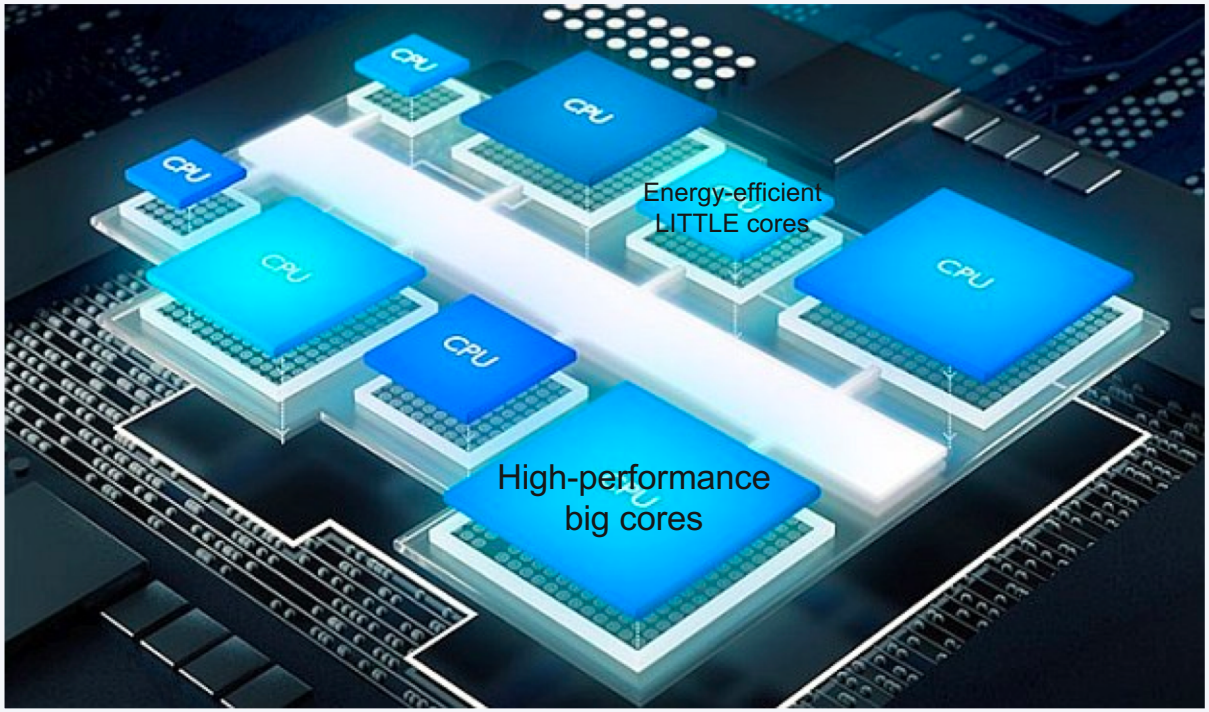
Specialize cores

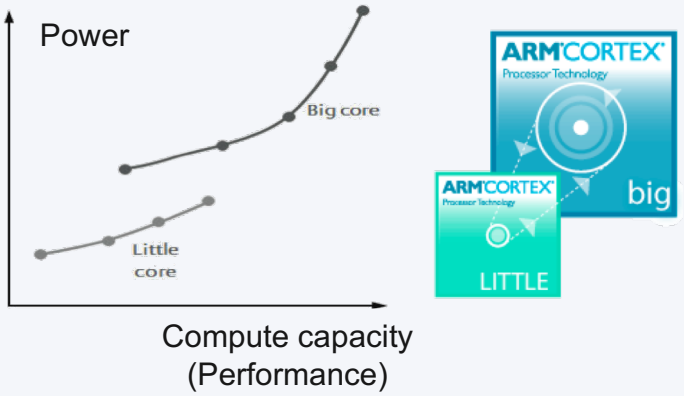
Exploit heterogeneity

[https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data]

Micro-architectural heterogeneity is already here

ARM DynamIQ / big.LITTLE

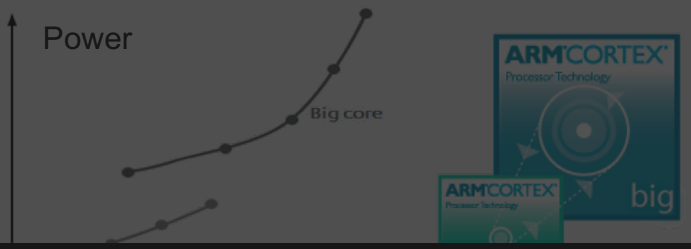


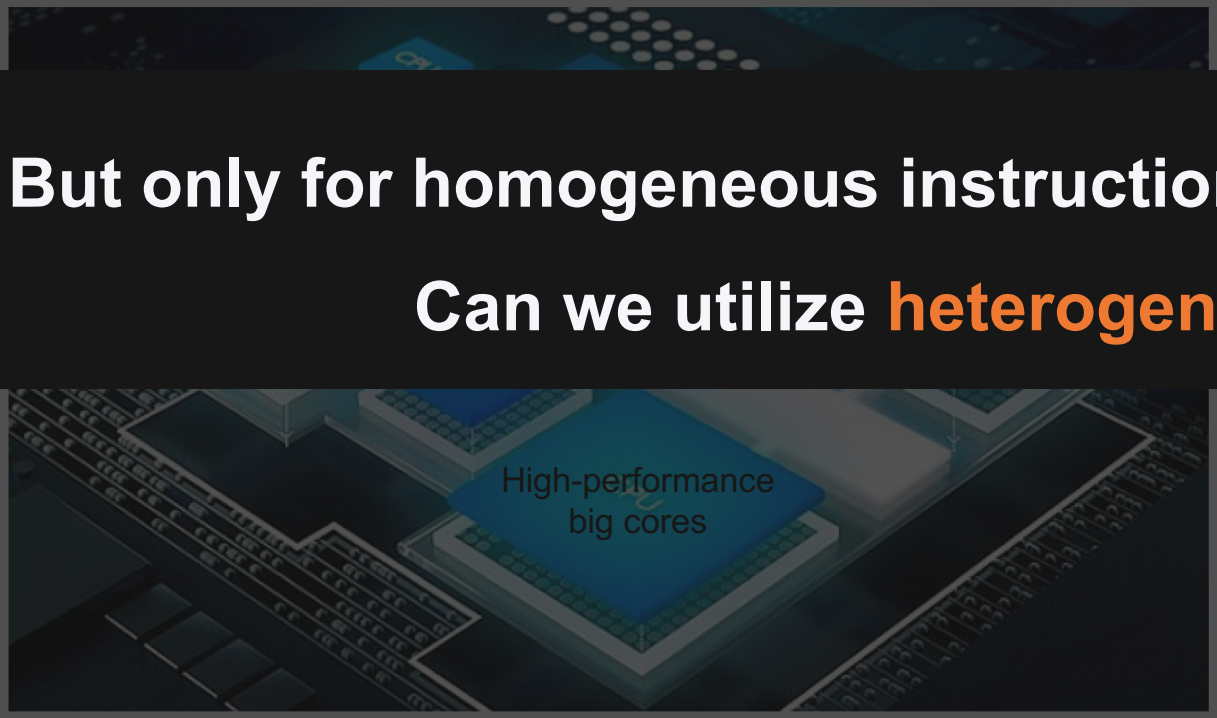


iPhone X Galaxy S8

Micro-architectural heterogeneity is already here

ARM big.LITTLE / DynamIQ





But only for homogeneous instruction set architecture (ISA)

Can we utilize heterogeneous-ISA?

iPhone X Galaxy S8

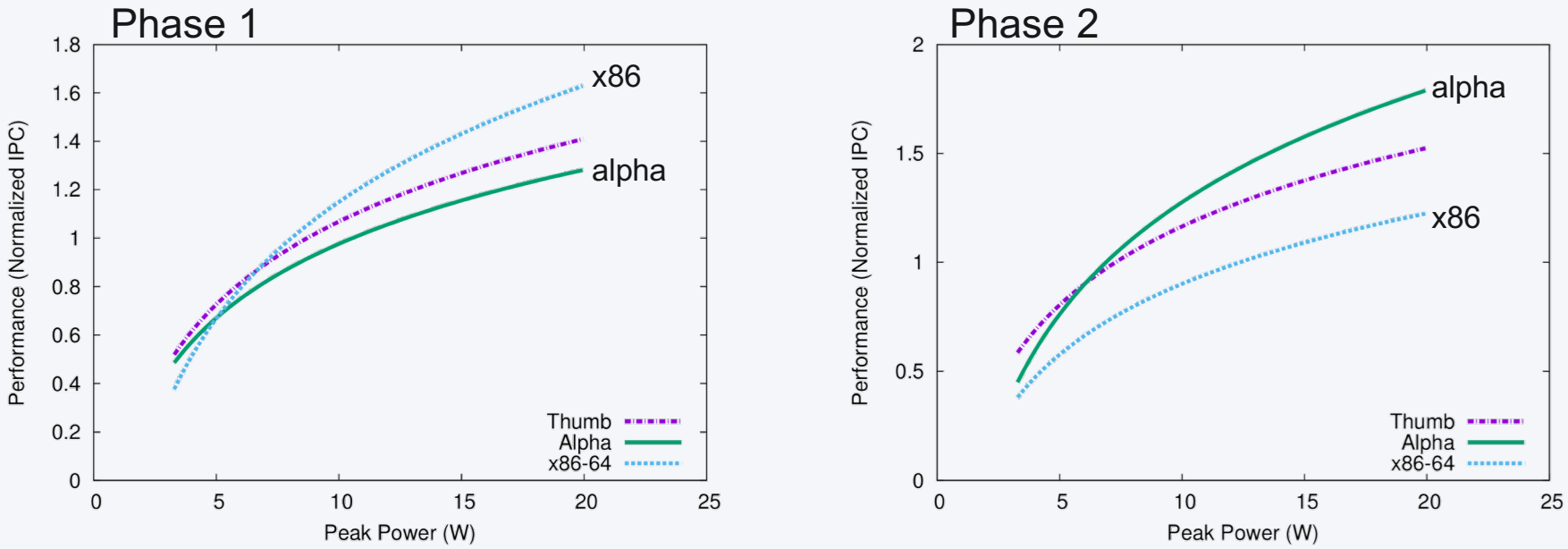
Different ISA, different execution profile

- "Harnessing ISA Diversity: Design of a Heterogeneous-ISA Chip Multiprocessor," Venkat and Tullsen (UCSD), ISCA'14

  - RISC vs CISC
  - Register memory architecture vs load/store architecture
  - Vector instruction support (e.g., SIMD)
  - Power efficiency per instruction
  - Pipeline depth
  - Degree of parallelism

Different ISA, different execution profile

- "Harnessing ISA Diversity: Design of a Heterogeneous-ISA Chip Multiprocessor," Venkat and Tullsen (UCSD), ISCA'14



Performance of bzip2 for different peak power budgets

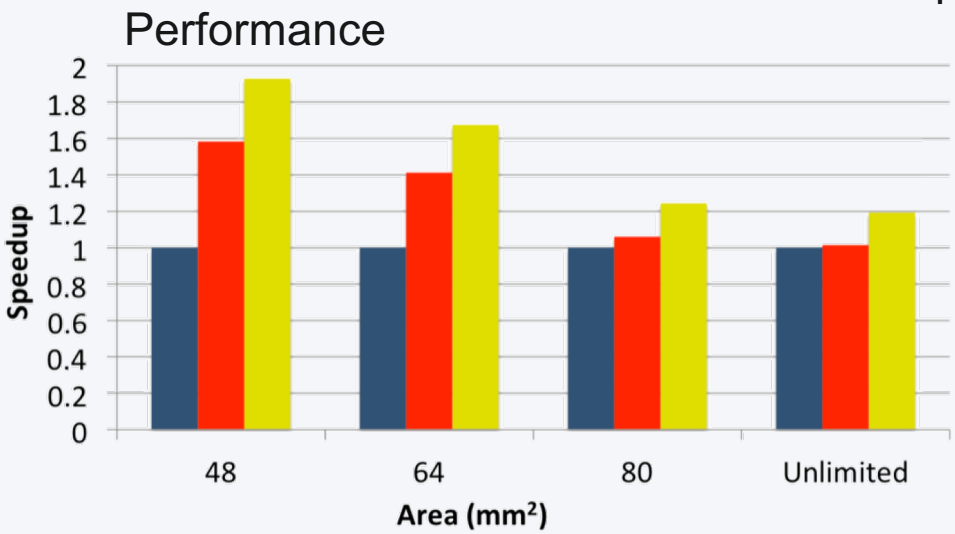
ISA affinity opens up opportunities

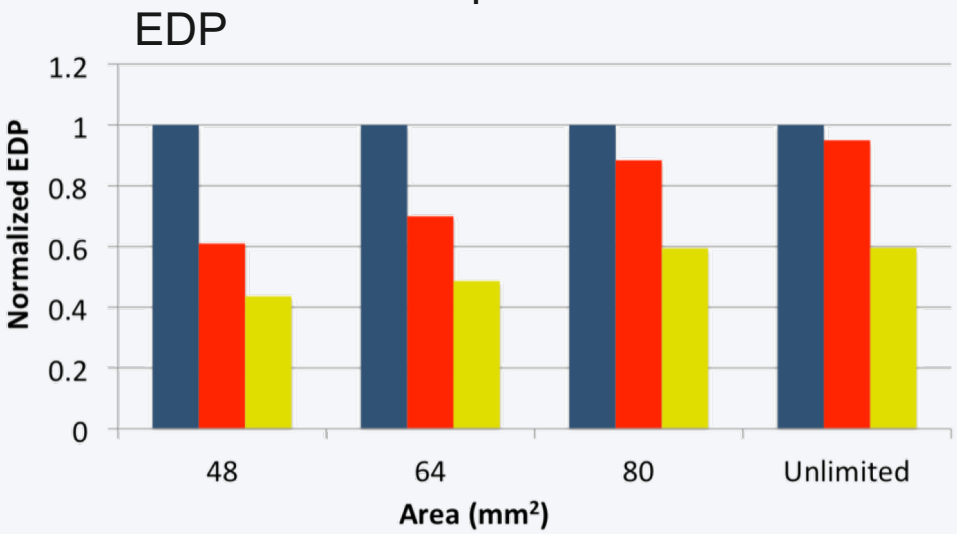
- Can improve performance and energy consumption by migrating work to an optimal-ISA node

Homogeneous
- Alpha

Single-ISA
- big Alpha
- medium Alpha
- little Alpha

Heterogeneous-ISA
- ARM's thumb
- x86_64
- Alpha





Challenges in exploiting the ISA affinity

- Relocate execution across machine boundaries
  - Single-chip/board heterogeneous-ISA architecture is not available
  - Not obvious even between homogeneous-ISA machines
- Deal with discrepancies between ISAs
  - Let assume ISAs have the same endian and primitive data type size
  - However, register set, stack layout, executable layout, …
- Want to run applications as-is
  - Cost(developer/software) >>>> cost(hardware)
  - Can enable future-proofing – important for legacy!

Popcorn Linux considers programmability

Serial

```
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];
...
}
```

MPI

```
void full_verify(void)
{
  MPI_Status  status;
  MPI_Request request;
  INT_TYPE    i, j;
  INT_TYPE    k, last_local_key;

  for( i=0; i<total_local_keys; i++ )
    key_array[--key_buff_ptr_global[key_buff2[i]]- total_lesser_keys]
        = key_buff2[i];
  last_local_key = (total_local_keys<1)? 0 : (total_local_keys-1);

  if( my_rank > 0 )
    MPI_Irecv( &k, 1, MP_KEY_TYPE, my_rank-1, 1000, MPI_COMM_WORLD,
        &request );
  if( my_rank < comm_size-1 )
    MPI_Send( &key_array[last_local_key], 1, MP_KEY_TYPE, my_rank+1,
        1000, MPI_COMM_WORLD );
  if( my_rank > 0 )
    MPI_Wait( &request, &status );
...
}
```

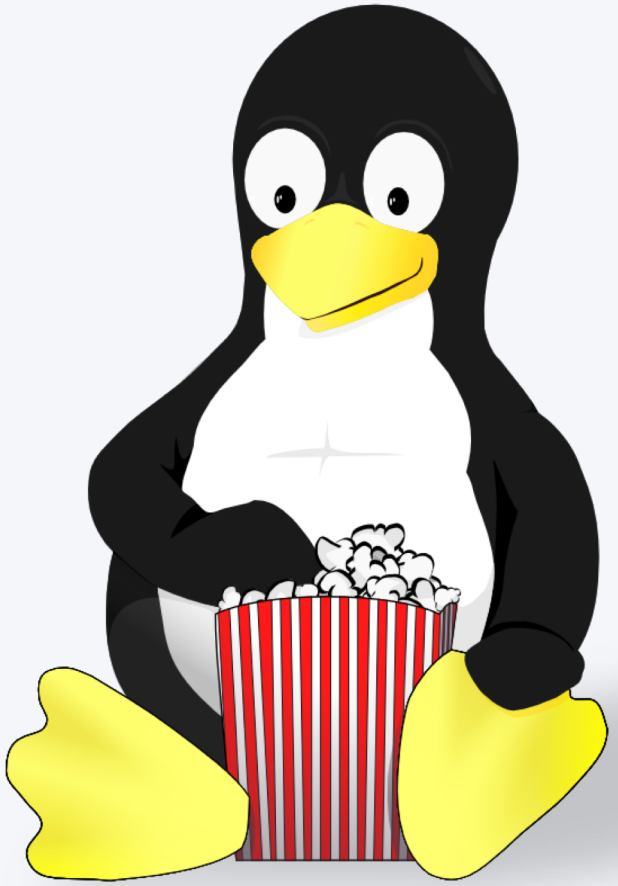
OpenCL

```
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

Popcorn Linux

Software framework
to run applications ***"as-is"***
on heterogeneous-ISA hardware



http://popcornlinux.org

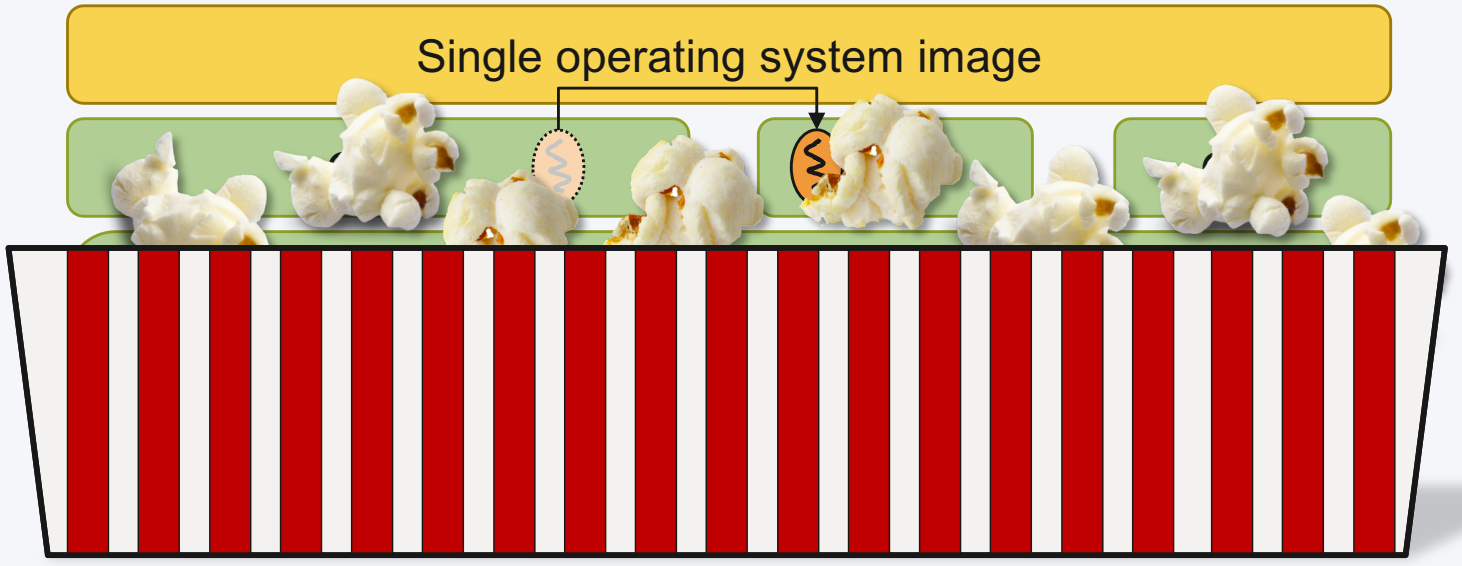
Outline

- What for heterogeneous-ISA systems?
- Introduction to Popcorn Linux
- Our approaches in Popcorn Linux
  - Compiler
  - Runtime
  - Operating System
- Ongoing work

Previously:

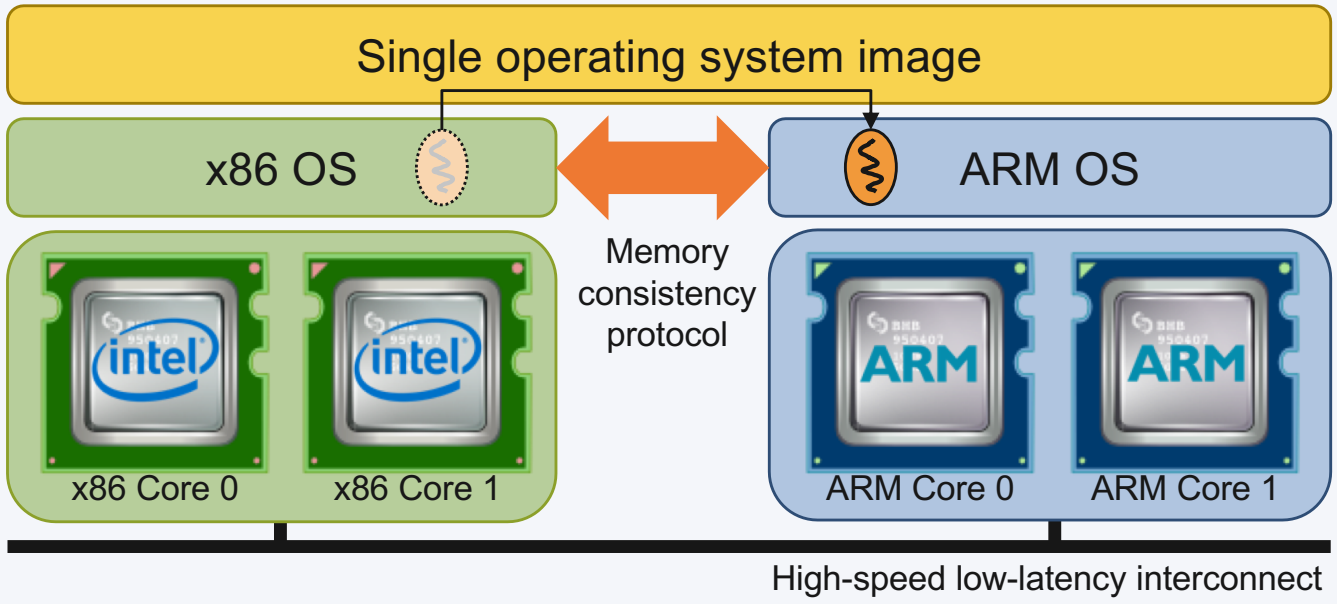
Popcorn Linux for replicated kernels

- Run multiple kernels on a single system
  - Run a kernel on a subset of processors in a system
  - Primarily for OS scalability
- Provide a single system image over the multiple kernels
- Migrate processes across the kernel boundary



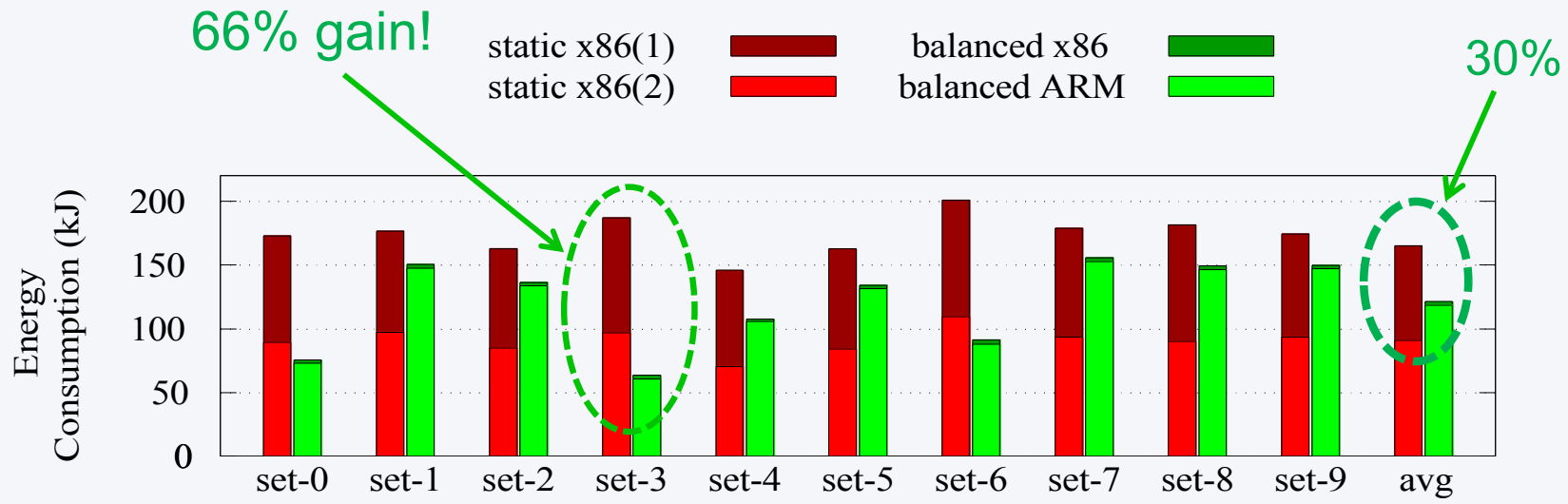
Popcorn Linux for heterogeneous ISAs

- Extend the replicated kernel concept over multiple **nodes**
  - Exploit the execution migration feature
- Allow threads in a process to be split over multiple nodes
- Support execution migration across ISA-different nodes

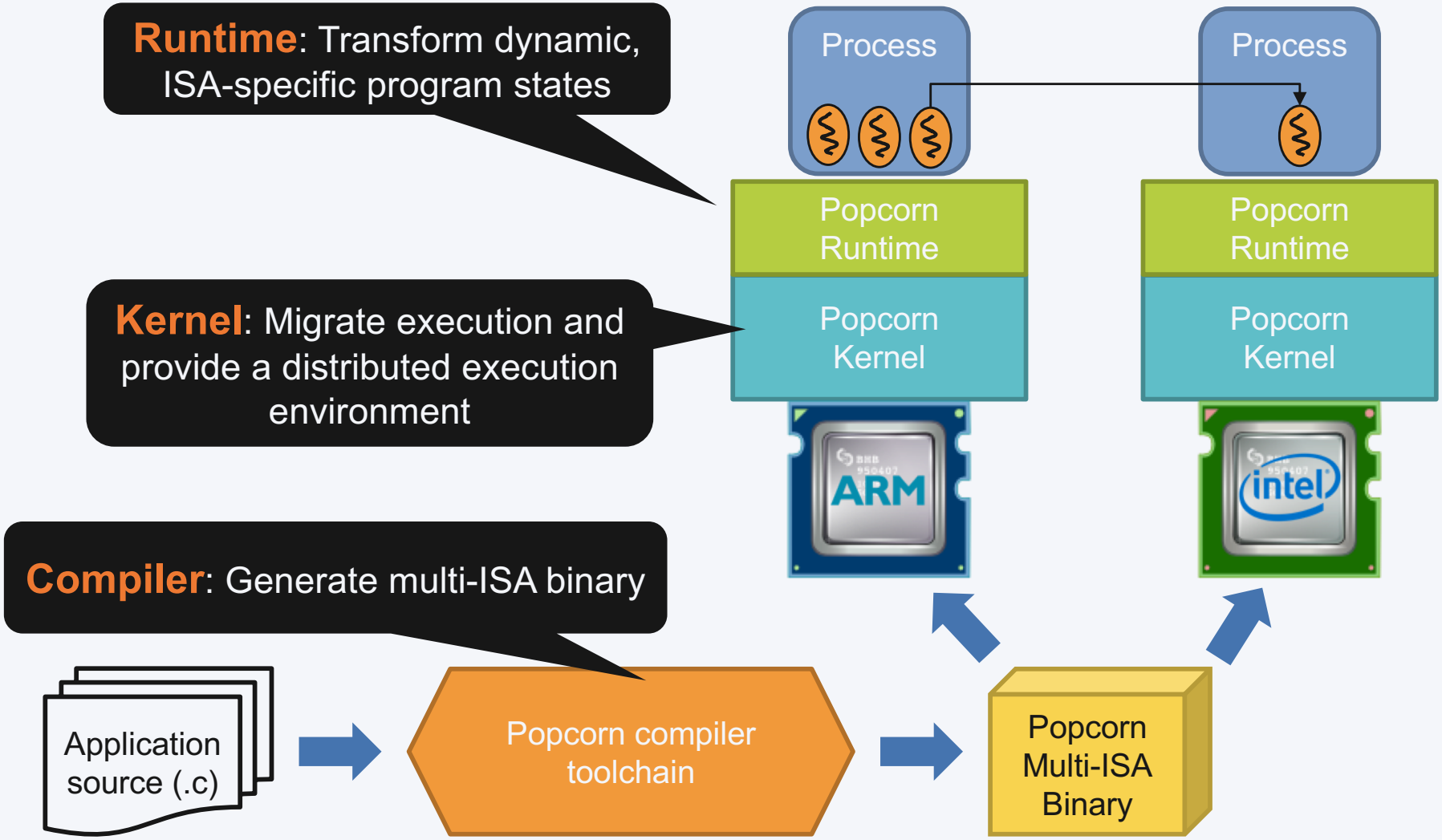


Popcorn Linux yields performance and energy gains over homogenous-ISA

- "Breaking the boundaries in heterogeneous-ISA datacenters," Barbalace et al., ASPLOS'17
  - Workload sets drawn from HPC benchmark suite (NPB)
  - Yields 30% energy savings on average (max is 66% for set-3)



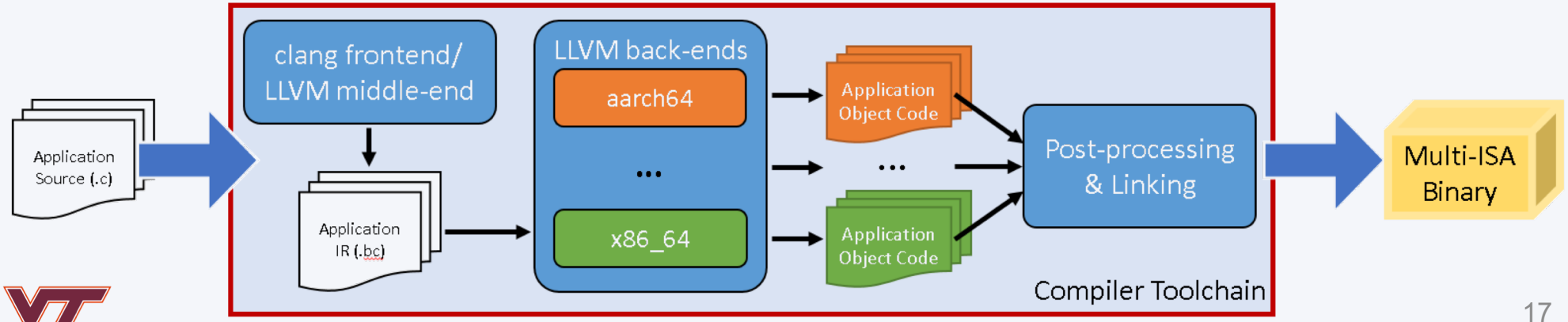
How Popcorn Linux work?



Popcorn Compiler

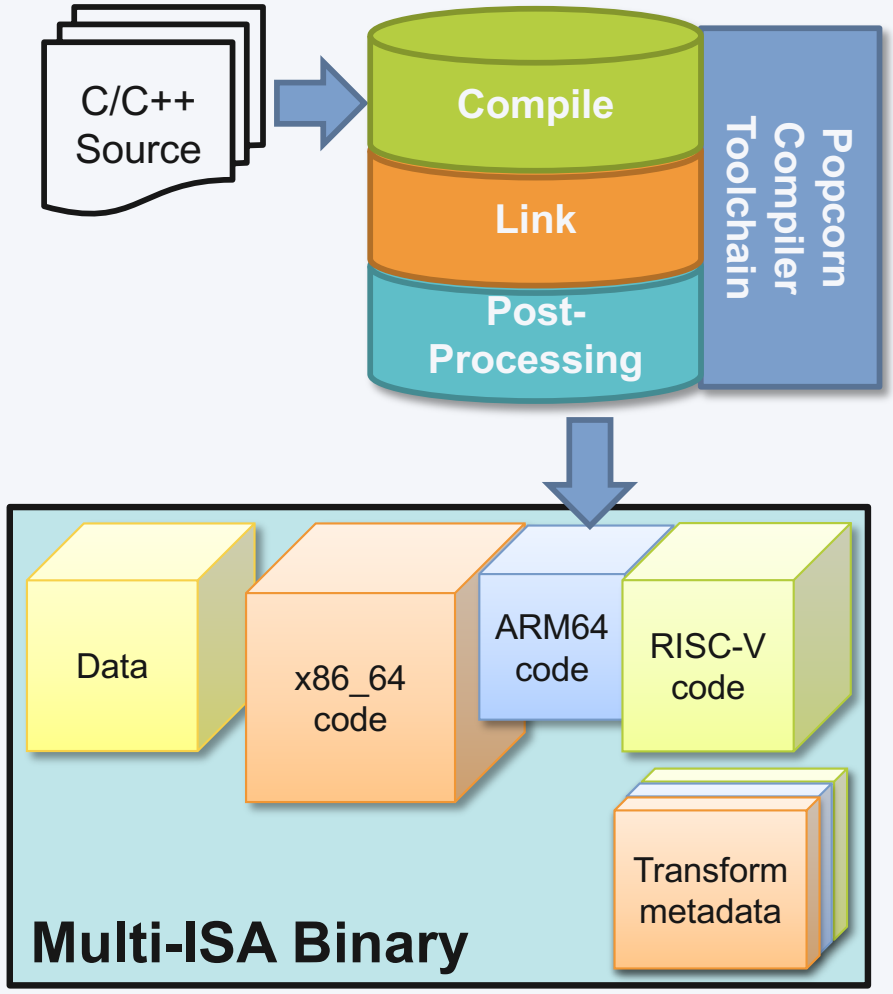
Compilation

- Built on top of clang/LLVM
- Application source lowered into LLVM IR
  - Insert migration points
    - Migration only at “equivalence points”; e.g., function entry/exit
  - Analyze liveness of variables
- IR passed through each ISA backend for generating code
  - Instrumentation to generate metadata (e.g., live locations)
- A post-process aligns code and data in uniform layout



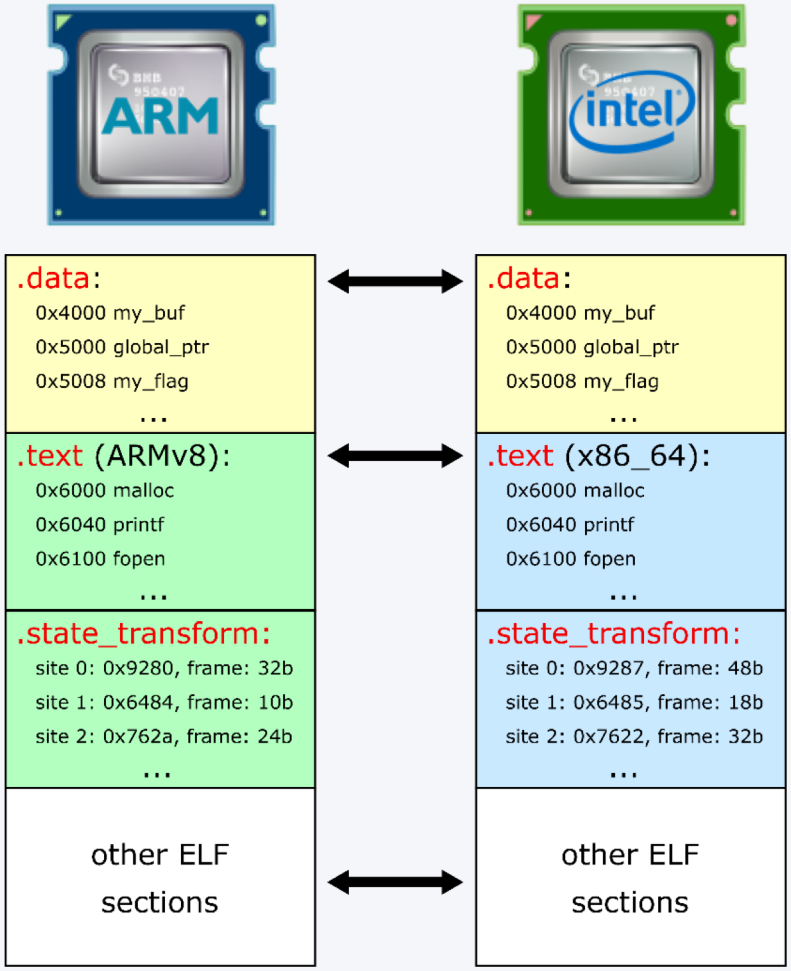
Multi-ISA binary

- Migratable across ISAs
  - Single .data section, multiple .text sections (one per-ISA)
  - Global data (.data), code (.text) and TLS aligned across all compilations
    - Pointers are valid across all ISAs
  - State transformation metadata
    - Added to binary for translating registers/stack between ISA-specific formats



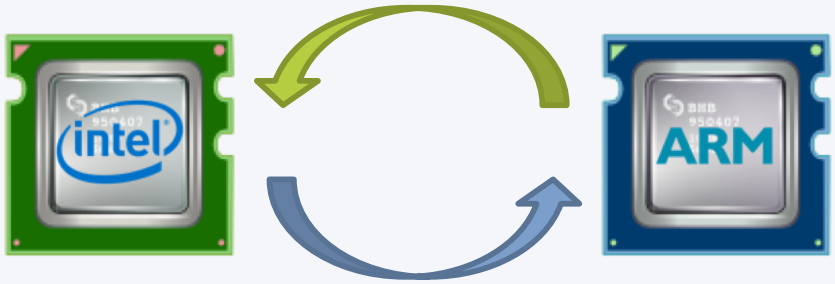
Multi-ISA binary

- Migratable across ISAs
  - Single .data section, multiple .text sections (one per-ISA)
  - Global data (.data), code (.text) and TLS aligned across all compilations
    - Pointers are valid across all ISAs
  - State transformation metadata
    - Added to binary for translating registers/stack between ISA-specific formats

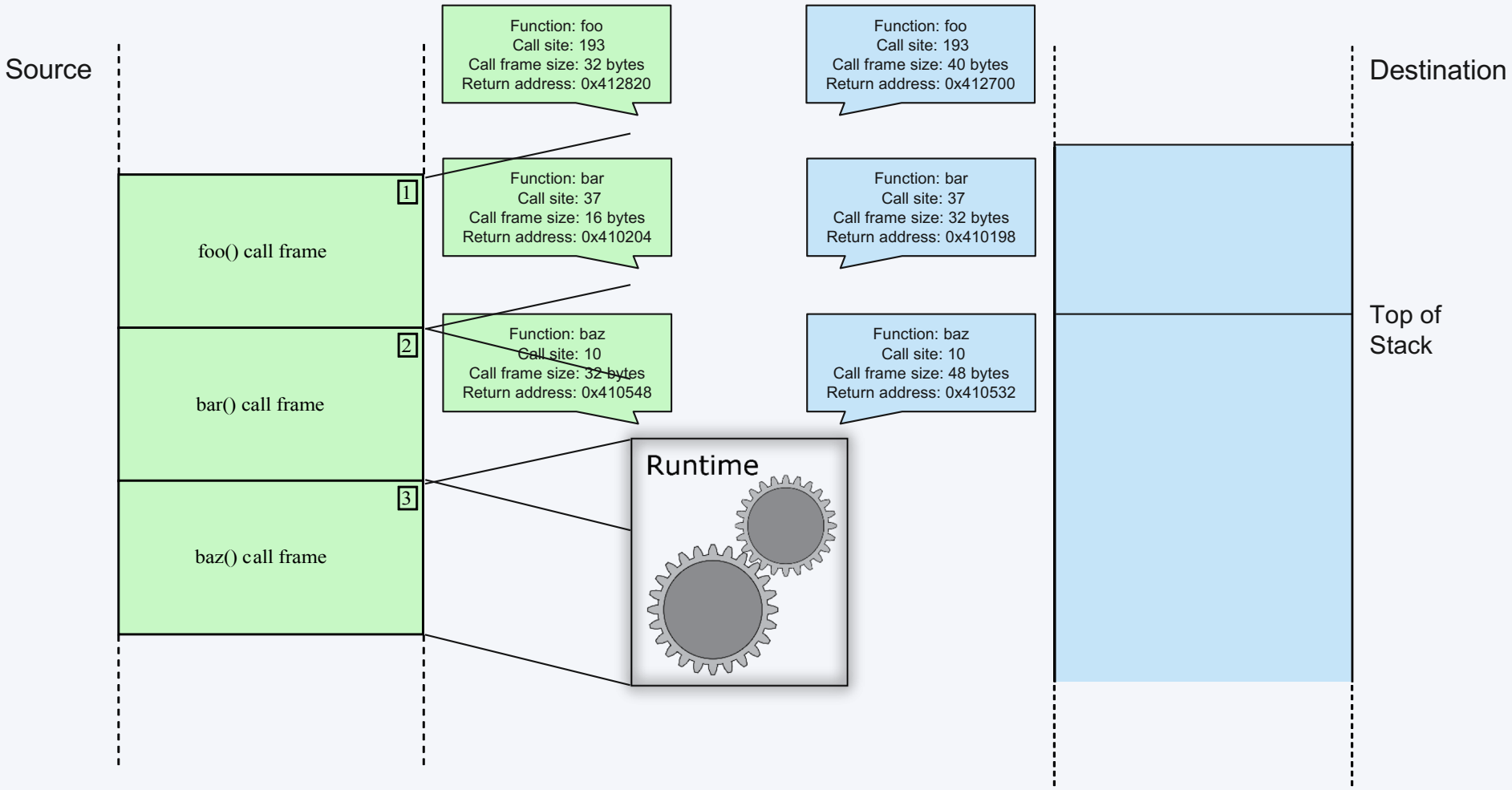


Popcorn Runtime

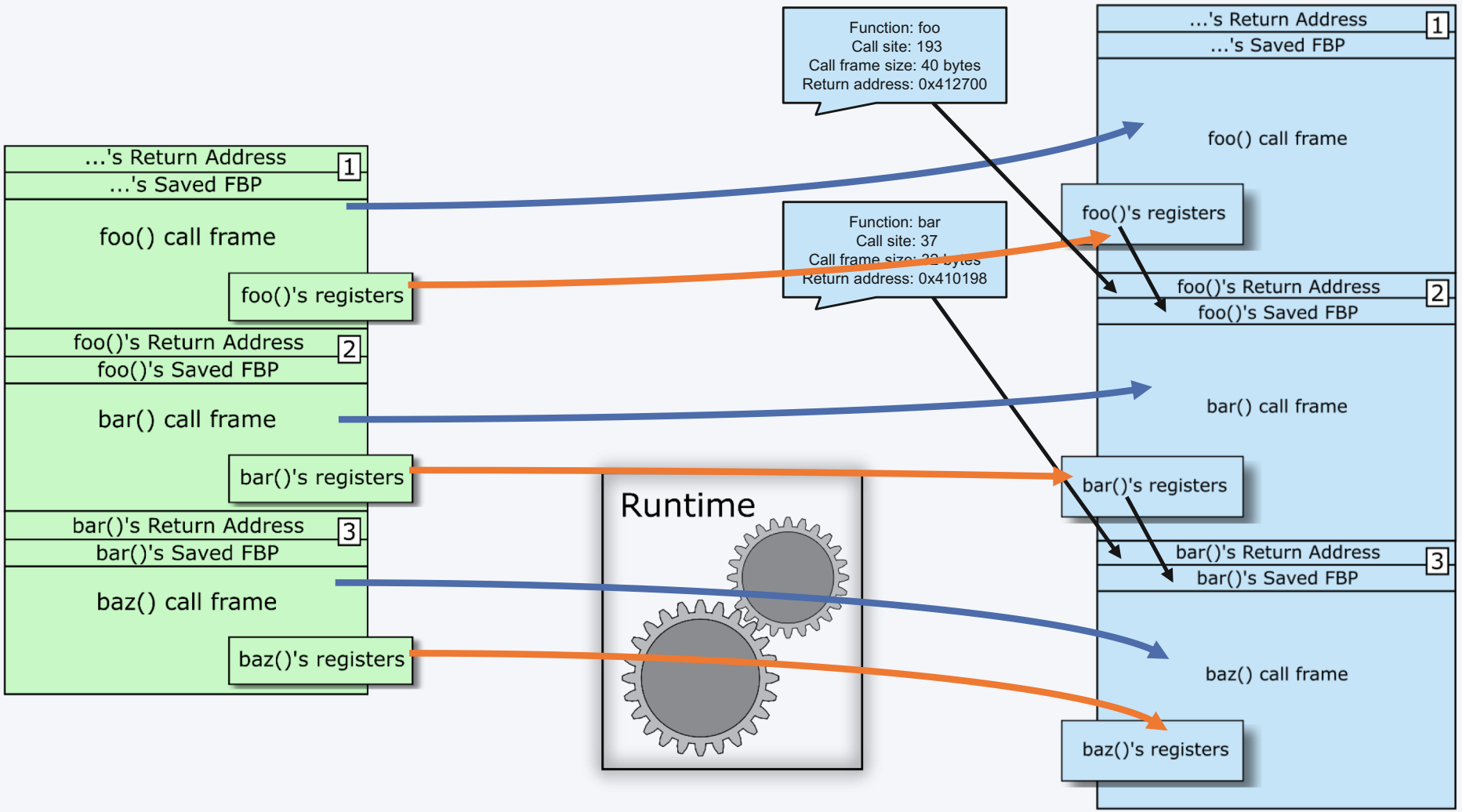
- Transform registers and stack between ISA-specific formats
  - Refer to the transformation metadata in the binary
- Two-phase process
  - Read compiler metadata describing function activation layouts
  - Rewrite stack in its entirety from source to destination ISA format
- After transformation, runtime invokes migration
  - Pass destination ISA's register state and stack to OS



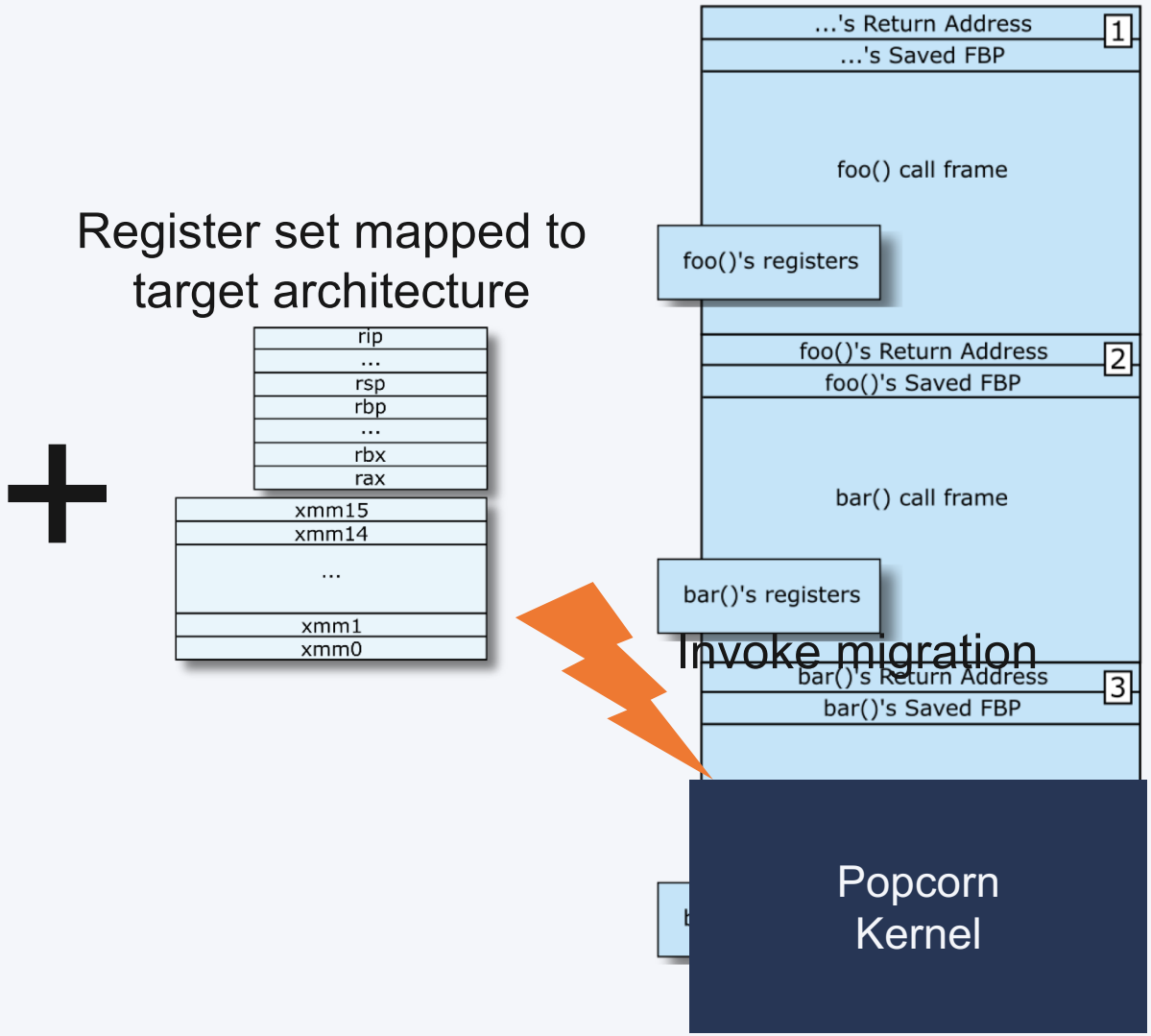
Stack transformation



Stack transformation



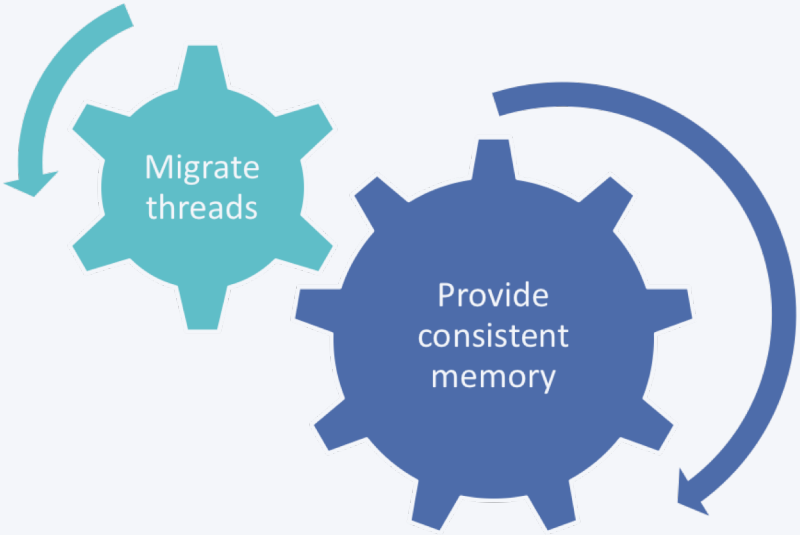
Stack transformation



Popcorn Kernel

- Based on Linux kernel v4.4.55
  - Working on x86-64 and aarch64
- Tried to be ***architecture-agnostic***
  - Except for register and PTE manipulation

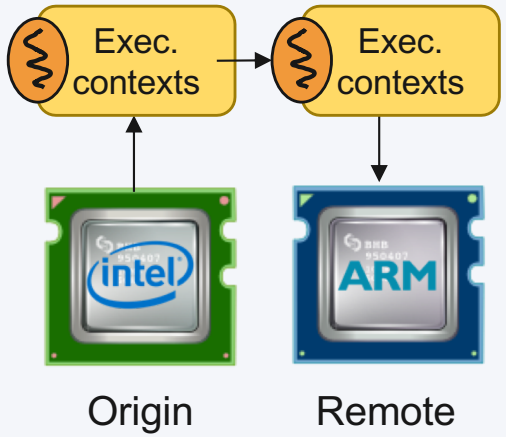
- Relocate/distribute threads over multiple nodes



- Migrating entire memory is infeasible
- Should provide sequential consistency

Migrating execution

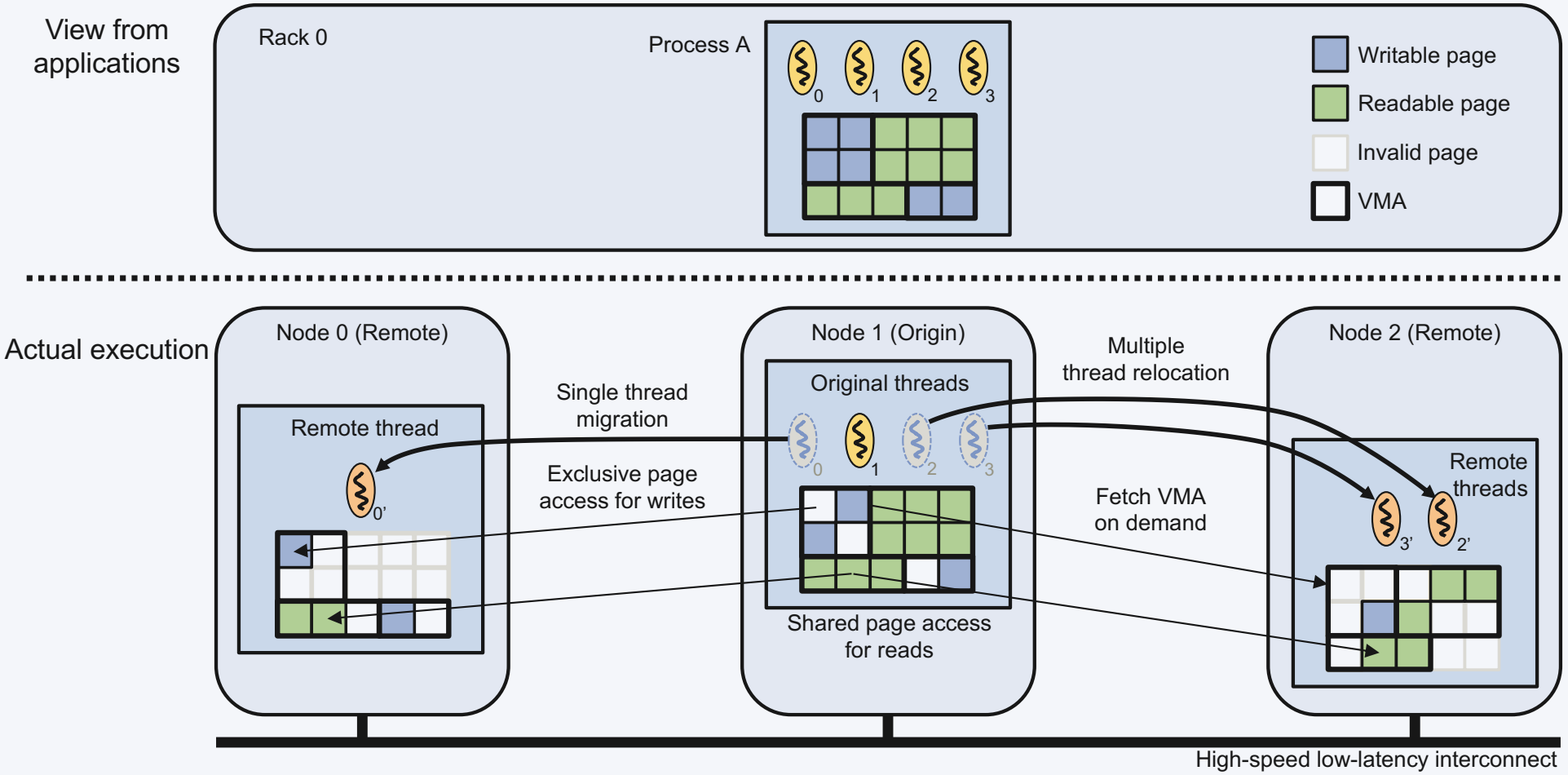
- Equivalent to context switching across machines



- At ***origin***: Save the execution context
  - The runtime provides the register set
  - `thread_struct + mm_struct`
- At ***remote***: Restore the context on a thread
  - Fork a kernel thread, and downgrade it to a user thread
  - Construct `mm_struct` and associate it with the thread
  - Setup register set and `thread_struct`
  - Return from the kernel space

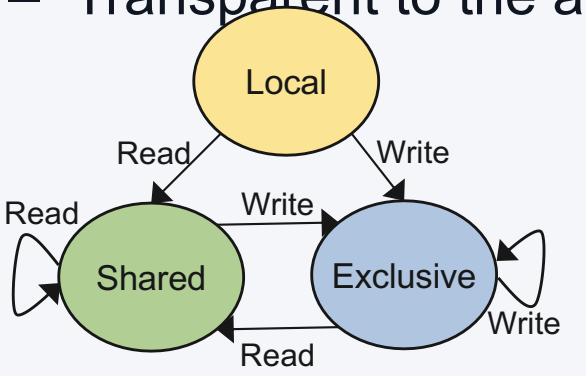
  ➔ Resume execution as if returned from system call

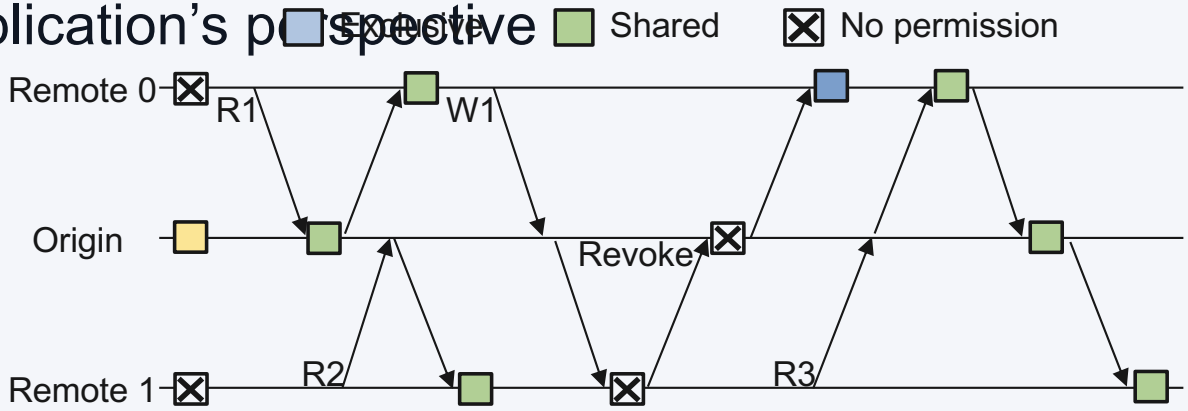
Distributed thread execution in action



Providing a consistent memory view to distributed threads

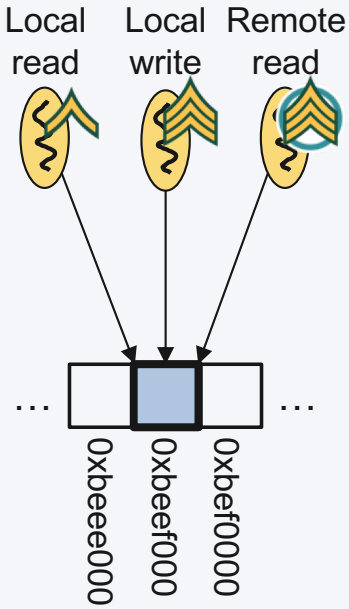
- The origin controls the ownership and data
  - Origin owns all pages in the beginning
  - Contact origin to get an ownership and data for pages
- Read-replicate, write-invalidate protocol at page granularity
  - To exploit the common cases in memory-intensive workloads
- Implemented in the virtual memory system in operating system
  - Transparent to the application's perspective





Taming concurrent page faults with leader/follower model

- Coalesce multiple faults and handle with a single operation
- Leader
  - The first thread that starts a page fault operation for a page at a moment
  - Execute the fault handling operation for the page
    - E.g., bring the page from remotes, fix up page table, flush TLB, …
- Followers
  - Threads that can utilize the leader's outcome
  - Wait for the completion of the leader's fault handling
- Otherwise
  - Wait or retry



Reducing false page sharing

- Inherent in page-level consistency protocol
  - A page can bounce between nodes if they access different data object in the same page
- Behavior analysis tool helps to identify false page sharing
  - Analyze page fault events collected in profiling mode
  - Pinpoint to the location in code
    - # of faults, type of faults, type of program objects

Program Object	Number of Accesses	R	W	I
stack/mmap	28068523	12589437	4053812	11425274
heap	2198224	1265049	464711	468464
_dlfcn_hooks	13184	2261	5646	5277

Location	Number of faults	R	W	I
numa-PageRank.C:81	14010634	10664750	0	3345884
numa-PageRank.C:145	11661205	0	3850227	7810978
numa-PageRank.C:212	2140582	1450619	0	689963
graph.h:52	1574677	1572864	0	1813
polymer.h:2395	536313	0	507966	28347
utils.h:301	124023	117731	0	6292
utils.h:256	123871	0	117731	6140
graph.h:46	36732	36622	0	110

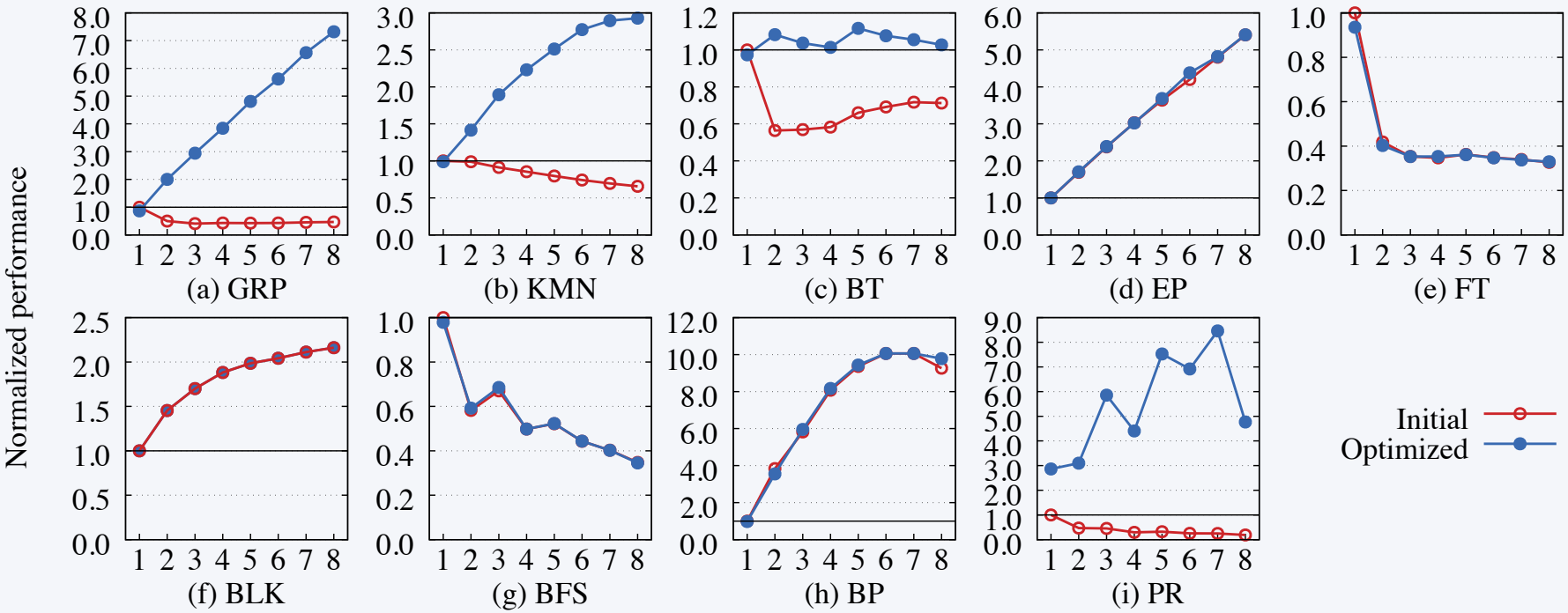
Reducing false page sharing

- Inherent in page-level consistency protocol
  - A page can bounce between nodes if they access different data object in the same page
- Behavior analysis tool helps to identify false page sharing
  - Analyze page fault events collected in profiling mode
  - Pinpoint to the location in code
    - # of faults, type of faults, type of program objects

Application		Mod. LoC		Application		Mod. LoC	
Simple	Grep	+21	-12	**PARSEC**	Blackscholes	-	-
	Kmeans	+6	-3				
NPB	Common	+1	-1	**Polymer**	Common	+86	-67
	BT	+5	-2		BFS	+10	-4
	EP	+2	-1		BP	+13	-10
	FT	+1	-1		PageRank	+32	-30

Took 4 days for a Ph.D. student to reduce false page sharing from 9 applications

The memory consistency protocol allows applications to scale their performance



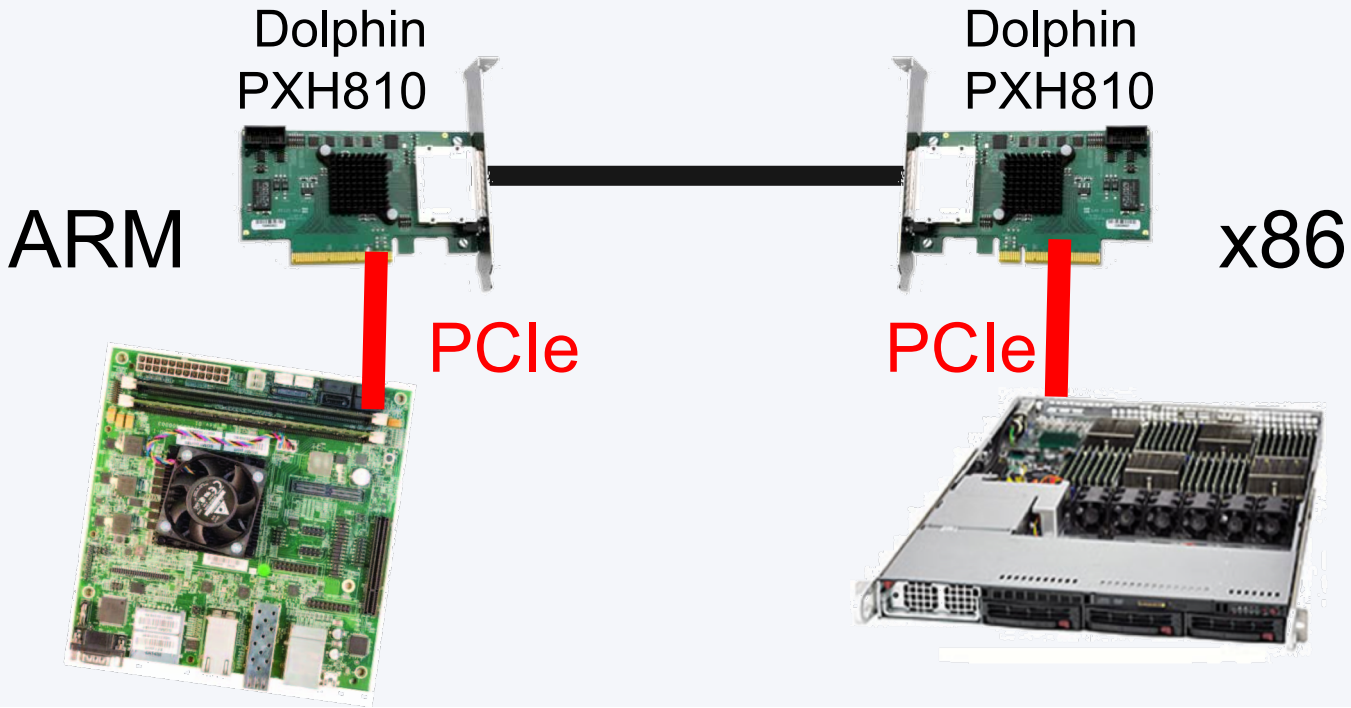
Results from 8 homogeneous x86 nodes

Summary: How Popcorn Linux work?

- **Compiler** generates “multi-ISA” binaries
  - `.text` for every ISA (symbol-aligned)
  - One `.data` for the entire machine (symbol-aligned)
- **Runtime** transforms dynamic ISA-specific program state
  - Stack, registers, etc, re-written on the fly
- **Operating system** migrates execution and provides a consistent execution environment across machines
  - Guarantee sequential data consistency to distributed threads

Ongoing work

- Towards a heterogeneous rack-scale system
  - Previously: ARM + x86 prototype platform
  - aarch64
    - APM883208-X1
      - 8 cores @2.4GHz
    - 16GB RAM, PCIe 8x
  - x86_64
    - Intel Xeon E5-1650v2
      - 6 cores 2HT @3.50 GHz
    - 16GB RAM, PCIe 8x



Rack-scale prototype platform

Currently on-line

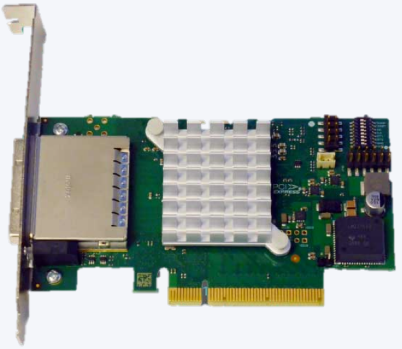
- 8 Intel Xeon (x86_64)
- 8 Cavium ThunderX (ARM64)

Working on

- APM X-Gene2
- IBM Power8
- RISC-V

Interconnects

- Dolphin interconnect
  - Dolphin PXH810 over PCIe up to 56Gb/s
  - Between tightly coupled nodes



- InfiniBand
  - Mellanox ConnectX-4/3 NICs and SX6036 switch up to 56Gb/s
  - Utilize Remote DMA (RDMA) feature
  - For global communication
- Ethernet
  - Based on the standard TCP/IP and sockets
  - As a standard, versatile interconnect

Ongoing work

- Towards a heterogeneous rack-scale system
- Incorporate more heterogeneity
  - IBM Power8
  - RISC-V
- Task scheduling in a heterogeneous-ISA rack
- Popcorn as a security infrastructure
  - E.g., Increase entropy to prevent ROP attacks
- Cross-ISA execution in a virtualization setting

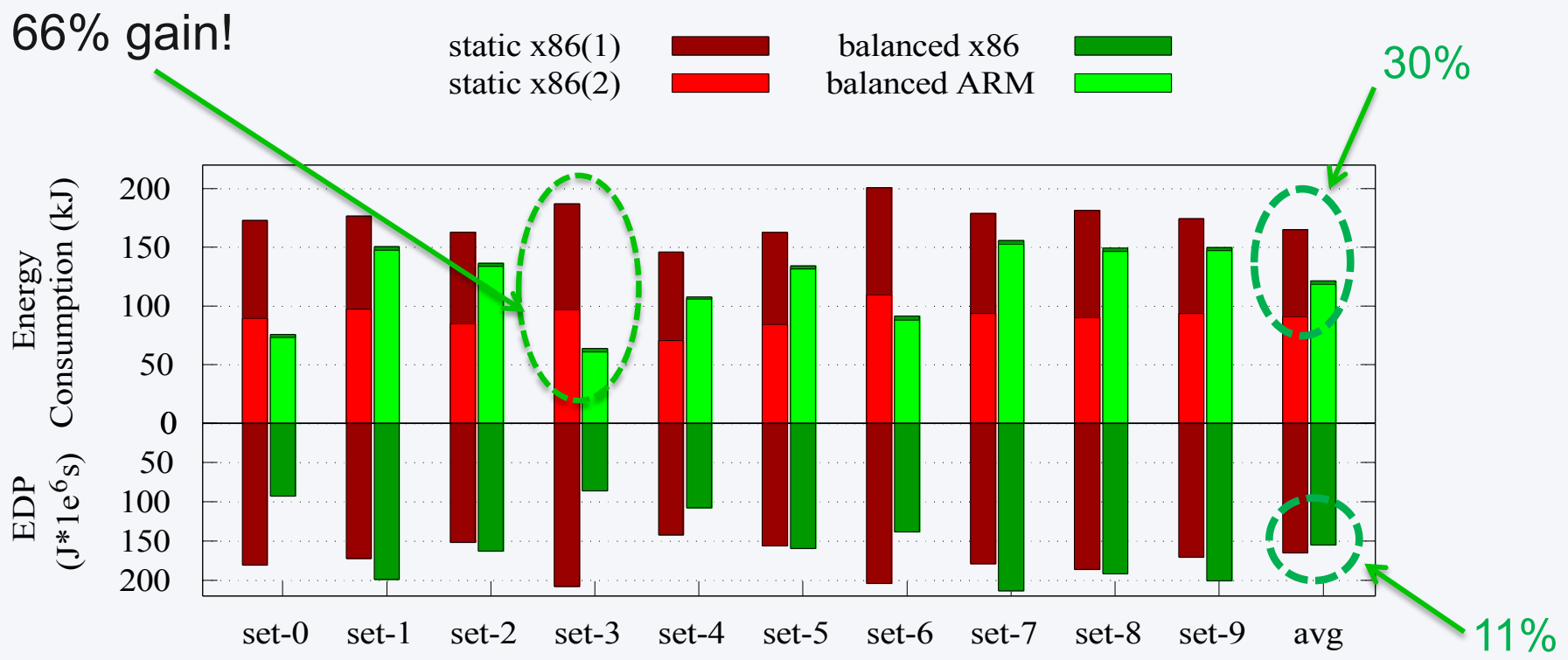
Thank you!

*** Popcorn Linux Team ***

Supervising: Changwoo Min, Binoy Ravindran
Compiler, runtime: Anthony Carno, Mohamed Karaoui, Robert Lyerly
Kernel: Horen Chuang, *Sang-Hoon Kim*

Backup slides

Performance and energy gains over homogenous-ISA



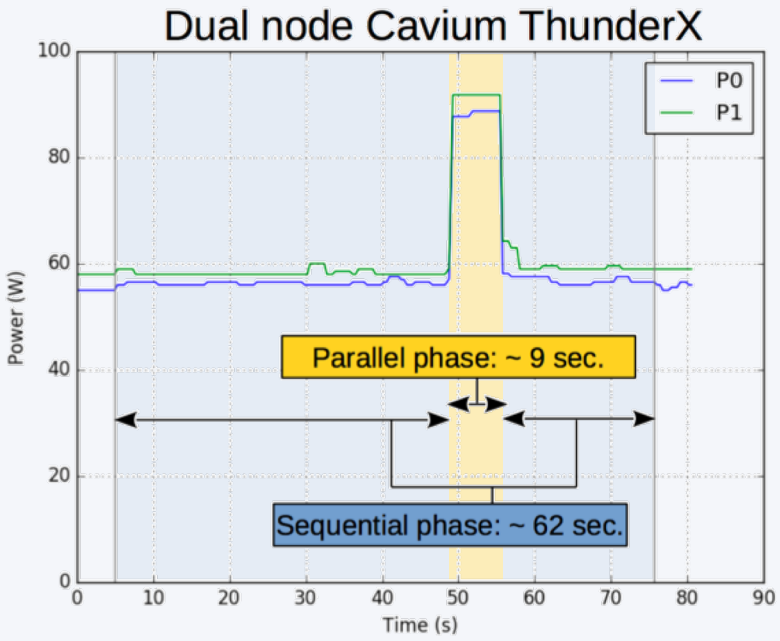
- Workload sets drawn from HPC benchmark suite (NPB)
- Smaller the energy-delay product, the better

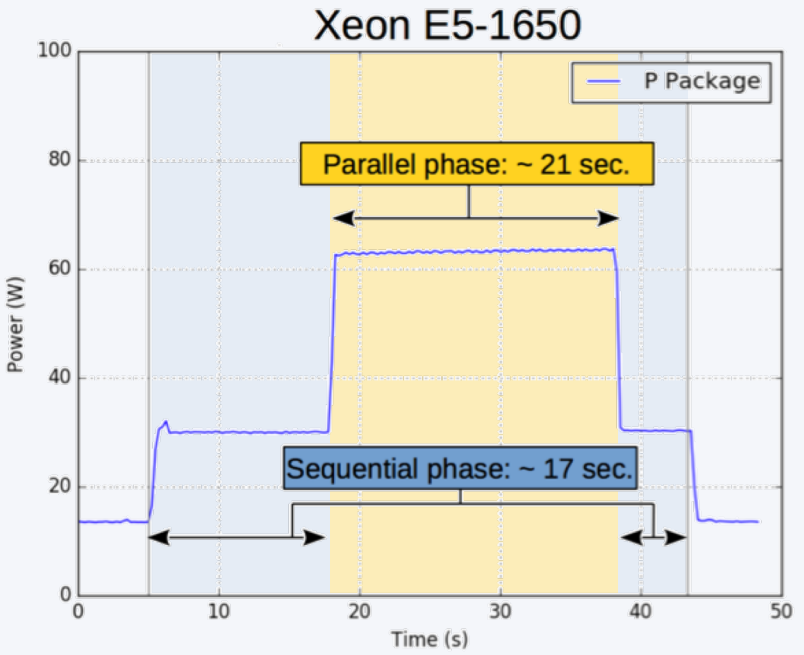
✓ Popcorn yields 30% energy savings on average (max is 66% for set-3)
✓ Popcorn yields 11% reduction in EDP

["Breaking the boundaries in heterogeneous-ISA datacenters," Barbalace et al., ASPLOS'17]

ISA affinity opens up opportunities

- "The Impact of ISAs on Performance," Akram and Sawalha, WDDD/ISCA'17
- "OS Support for Thread Migration and Distribution in the Fully Heterogeneous Datacenter," Olivier et al., HotOS'17





PARSEC blackscholes

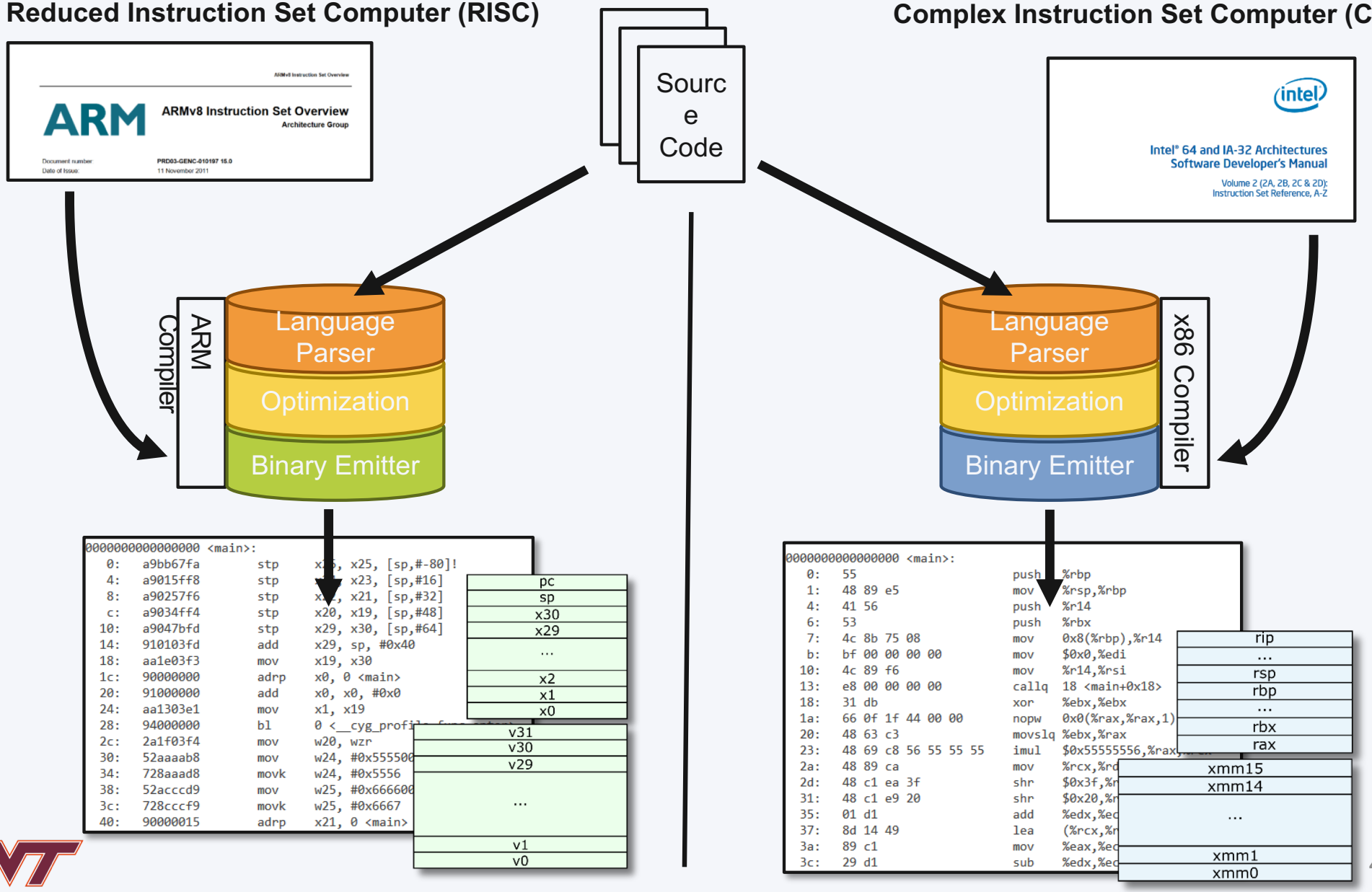
Motivation

- Proliferation of heterogeneous-ISA platforms
  - Discrete
    - Xeon Phi, GPUs
  - Integrated On-Die/SoC
    - CPU + GPU (AMD A-series)
    - CPU + Accelerator Slices (Tilera TILEncore Gx-series)
    - CPU + GPU + DSP + … (Qualcomm Snapdragon)
- Mix of OS & non-OS capable

Instruction Set Architecture (ISA)

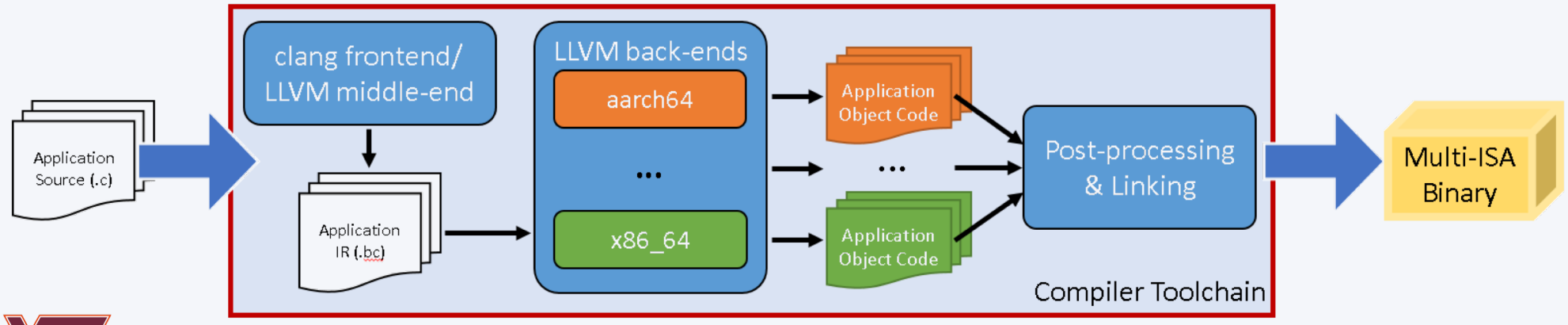
Reduced Instruction Set Computer (RISC)

Complex Instruction Set Computer (CISC)



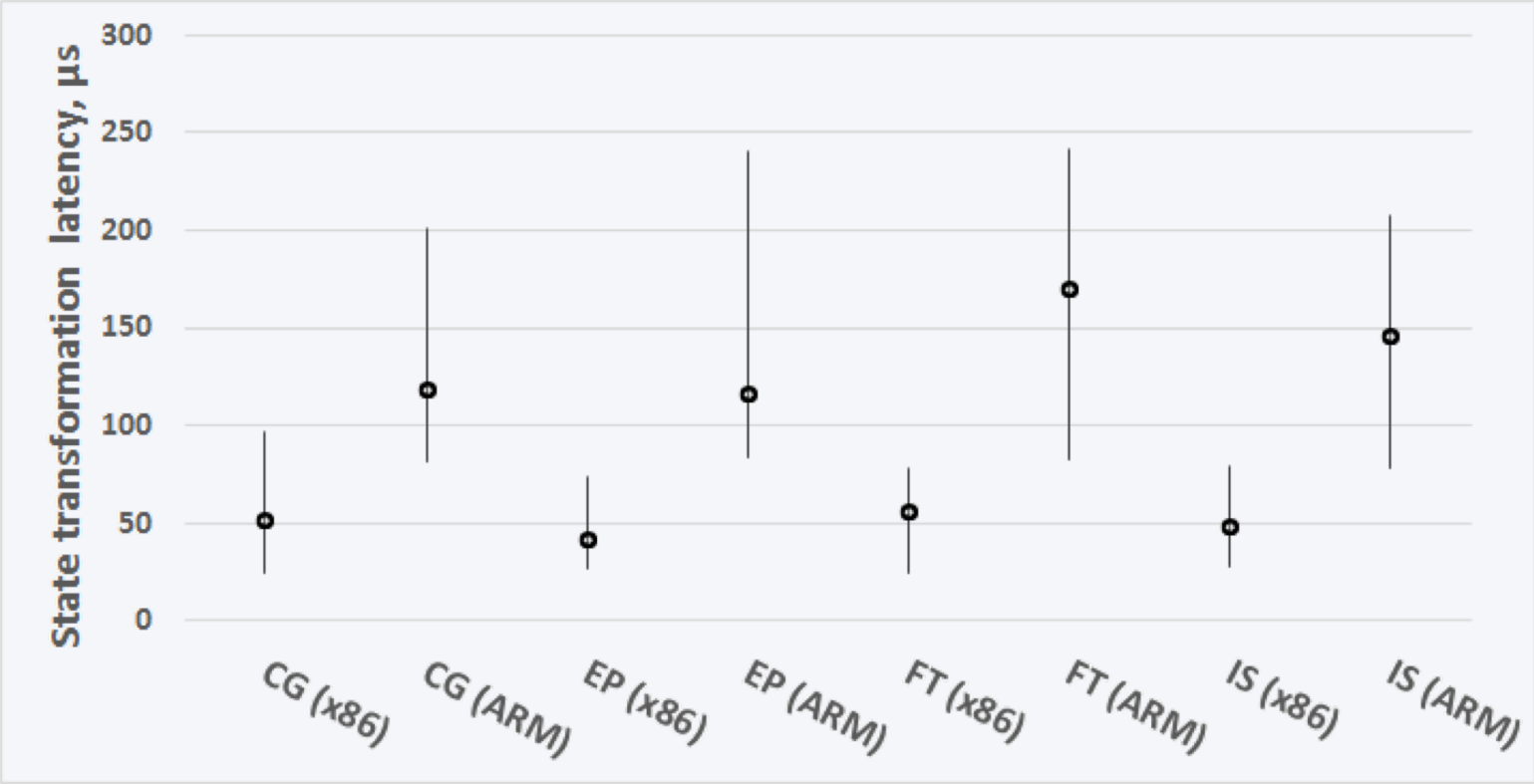
Popcorn compilation

- Built on top of clang/LLVM
  - clang/LLVM 3.7.1, GNU gold 2.27 (~12.4k LoC)
  - Address space alignment (~700 LoC), post-processing (~1.7k LoC) tools
  - State transformation/migration libraries (~5.9k LoC)
  - Minor updates to musl-libc 1.1.18, libelf, and GNU OpenMP runtime



State Transformation

- How fast is state transformation?
  - Reference: typically scheduling is done at every 10 milliseconds



Migration Points

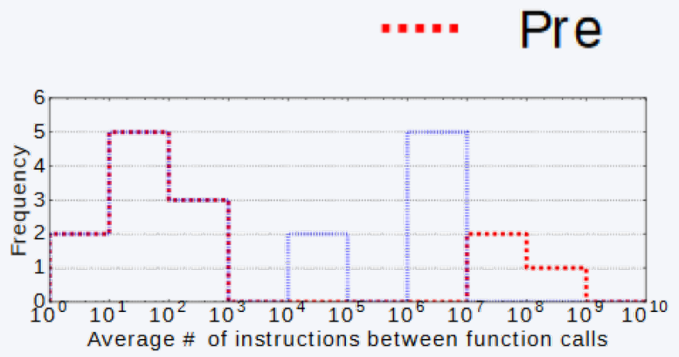


Figure 3. NPB CG number of instructions between migration points.

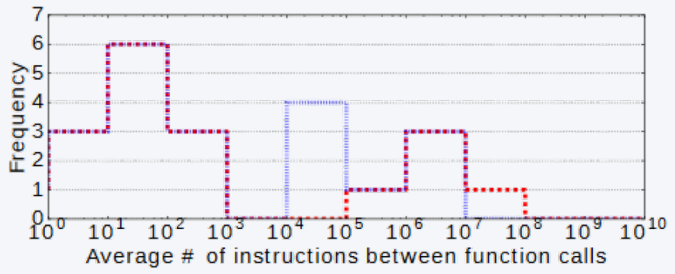


Figure 4. NPB IS number of instructions between migration points.

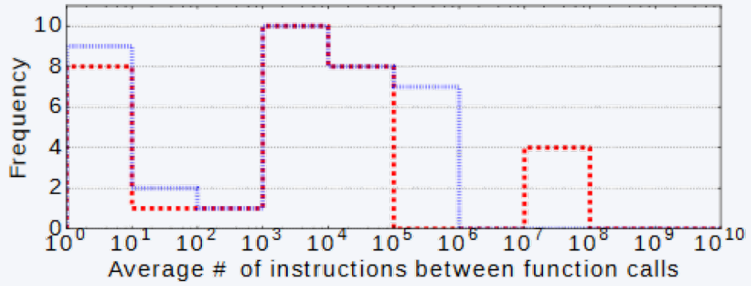


Figure 5. NPB FT number of instructions between migration points.

Significant rewriting cost: NPB example

- From ***shared memory/OpenMP*** to MPI

Benchmark	CG	EP	FT	IS	MG
OpenMP LOC	1150	297	1106	1108	1481
MPI modified	98%	44%	98%	46%	97%

OpenMP and MPI version of NASA NPB

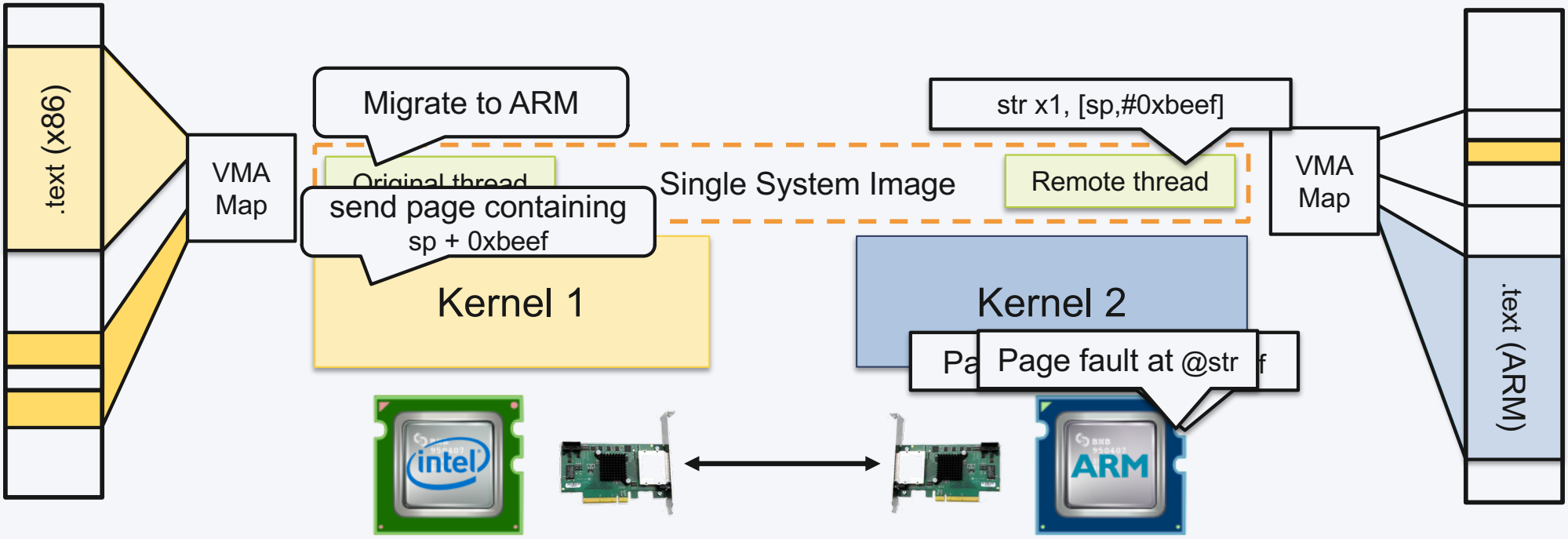
- From ***serial code*** to OpenCL

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

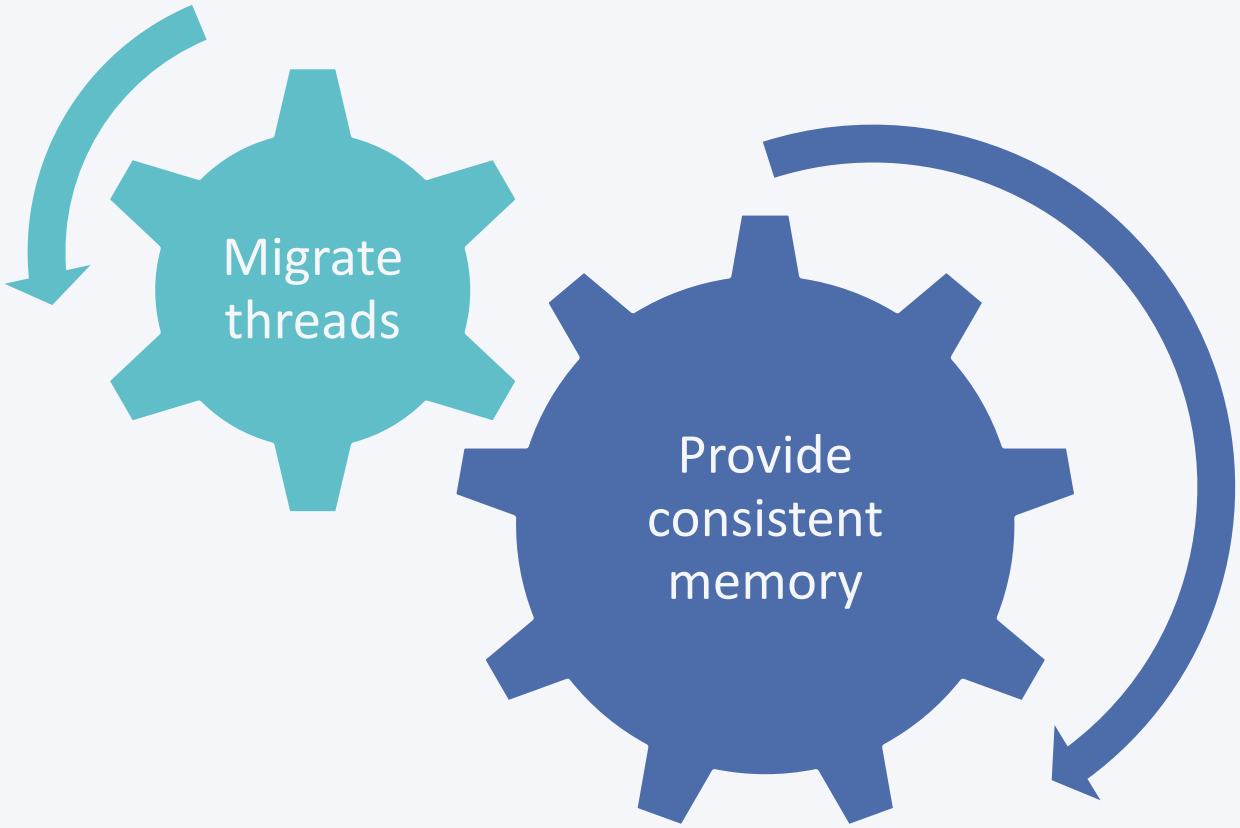
OpenCL and serial version of SNU NPB

"Popcorn: bridging the programmability gap in heterogeneous-ISA platforms," A. Barbalace et al., EuroSys, 2015.

Thread migration in action

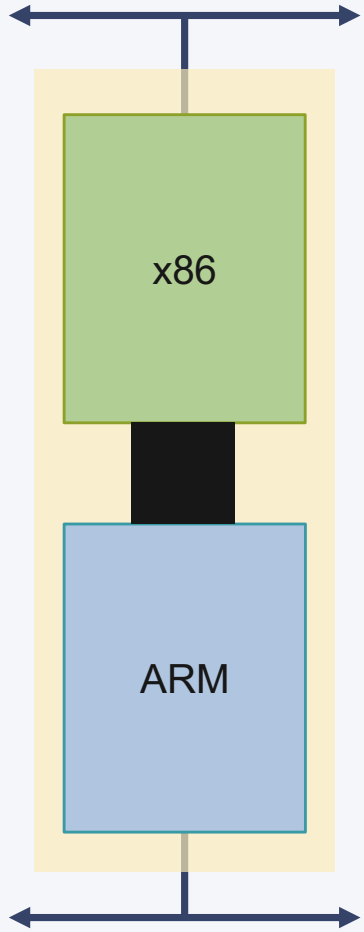


Popcorn Kernel

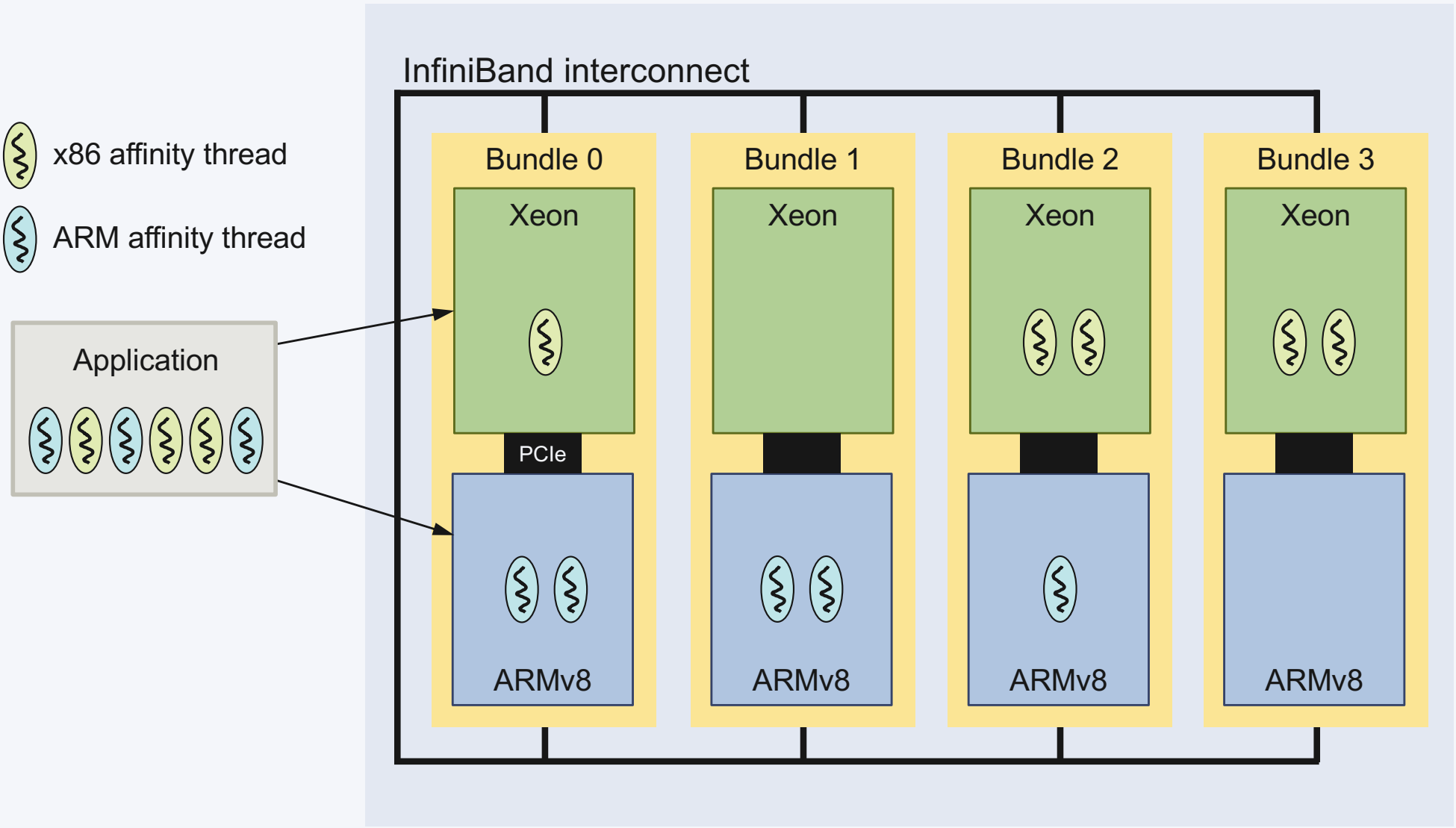


The Rack

- Bundle
  - The building block for “The Rack”
  - A set of nodes that are tightly-coupled each other
    - To control the latency of memory consistency protocol
- Bundles are connected via a high-speed switching interconnect



The Rack


x86 affinity thread
ARM affinity thread
Application
InfiniBand interconnect
Bundle 0
Bundle 1
Bundle 2
Bundle 3
Xeon
Xeon
Xeon
Xeon
PCIe
ARMv8
ARMv8
ARMv8
ARMv8