xBGAS: Toward a RISC-V Extension for Global, Scalable Shared Memory

John Leidel[1], David Donofrio[2], Farzad Fatollahi-Fard[2],
Kurt Keville[3], Xi Wang[4], Frank Conlon[4], Yong Chen[4]
[1]Tactical Computing Labs; [2]Lawrence Berkeley National Lab
[3]MIT; [4]Texas Tech



Overview

- xBGAS Background
- xBGAS Addressing Architecture
- Ongoing Research

xBGAS Background

TEXAS TECH UNIVERSITY

Tactical Computing Labs

Massachusetts Institute of Technology

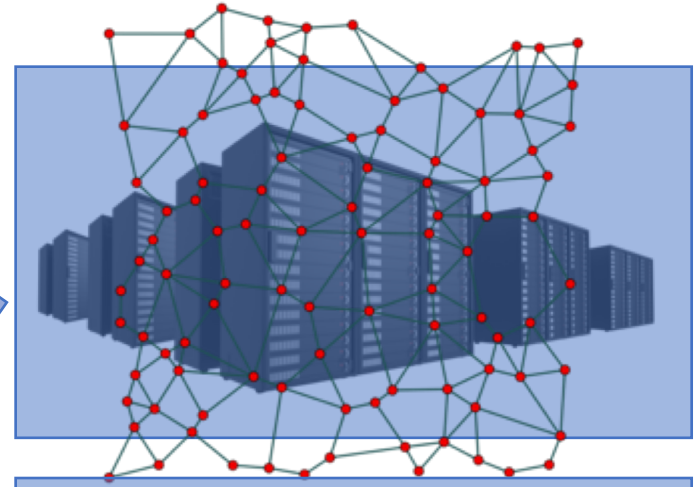
BOSTON UNIVERSITY

Data Center Scale Addressing

- **E**xtended **B**ase **G**lobal **A**ddress **S**pace (xBGAS)
- Goals:
  - Provide extended addressing capabilities without ruining the base ABI
    - EG, RV64 apps will still execute without an issue
  - Extended addressing must be flexible enough to support multiple target application spaces/system architectures
    - Traditional data centers, clouds, HPC, etc..
  - Extended addressing must not specifically rely upon any one virtual memory mechanism
    - EG, provide for object-based memory resolution
- What is xBGAS **NOT**?
  - …a direct replacement for RV128

Application Domains

- HPA-FLAT
  - High performance analytics flat addressing
  - For extremely large datasets that are too difficult/time consuming to shard
- MMAP-IO
  - Map storage tiers into address space
  - Potential for object-based addressing
  - See DDN WOS
- Cloud-BSP
  - Potential for global object visibility for in-memory cloud infrastructures (Spark)
  - Reduce the time/cost to port Java to a full 128-bit addressing model
- Security
  - Fine grained, tagged security extensions to base addressing model
  - Tags are stored/maintained as ACL's for secure memory regions
- HPC-PGAS
  - High Performance Computing: Partitioned Global Address Space



TEXAS TECH UNIVERSITY

Tactical Computing Labs

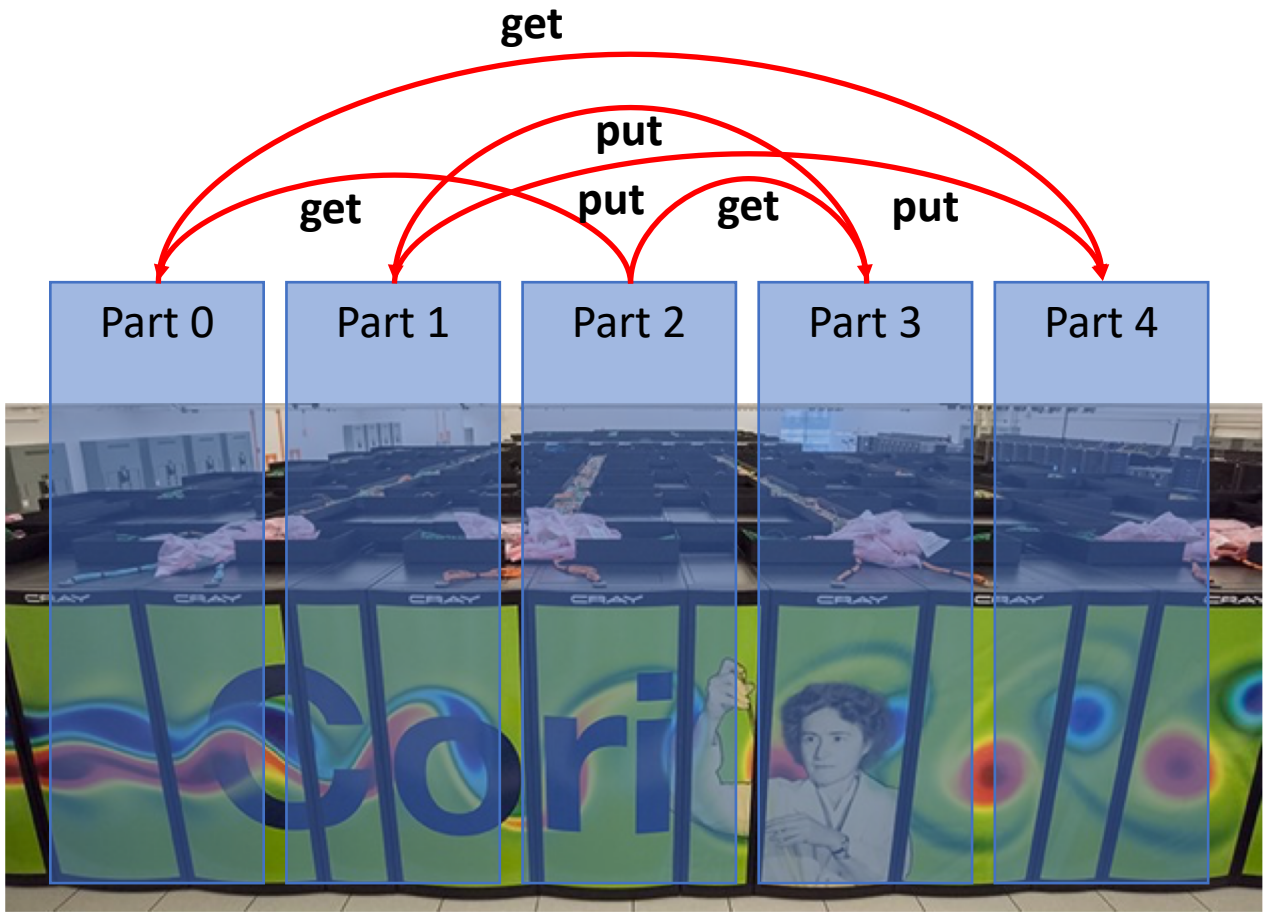
Massachusetts Institute of Technology

BERKELEY LAB

BOSTON UNIVERSITY

HPC-PGAS

- Traditional message passing paradigm has tremendous amount of overhead
  - User library overhead, driver overhead
  - Optimized for large data transfers
  - Management of communication for Exascale-class systems
- We have excellent examples of low-latency PGAS runtimes, but little hardware/uArch support
  - LBNL: GASnet
  - PNNL: Global Arrays/ARMCI
  - Cray: Chapel
  - OpenSHMEM



TEXAS TECH UNIVERSITY

Tactical Computing Labs

Massachusetts Institute of Technology

BERKELEY LAB

BOSTON UNIVERSITY

xBGAS Addressing Architecture

TEXAS TECH UNIVERSITY

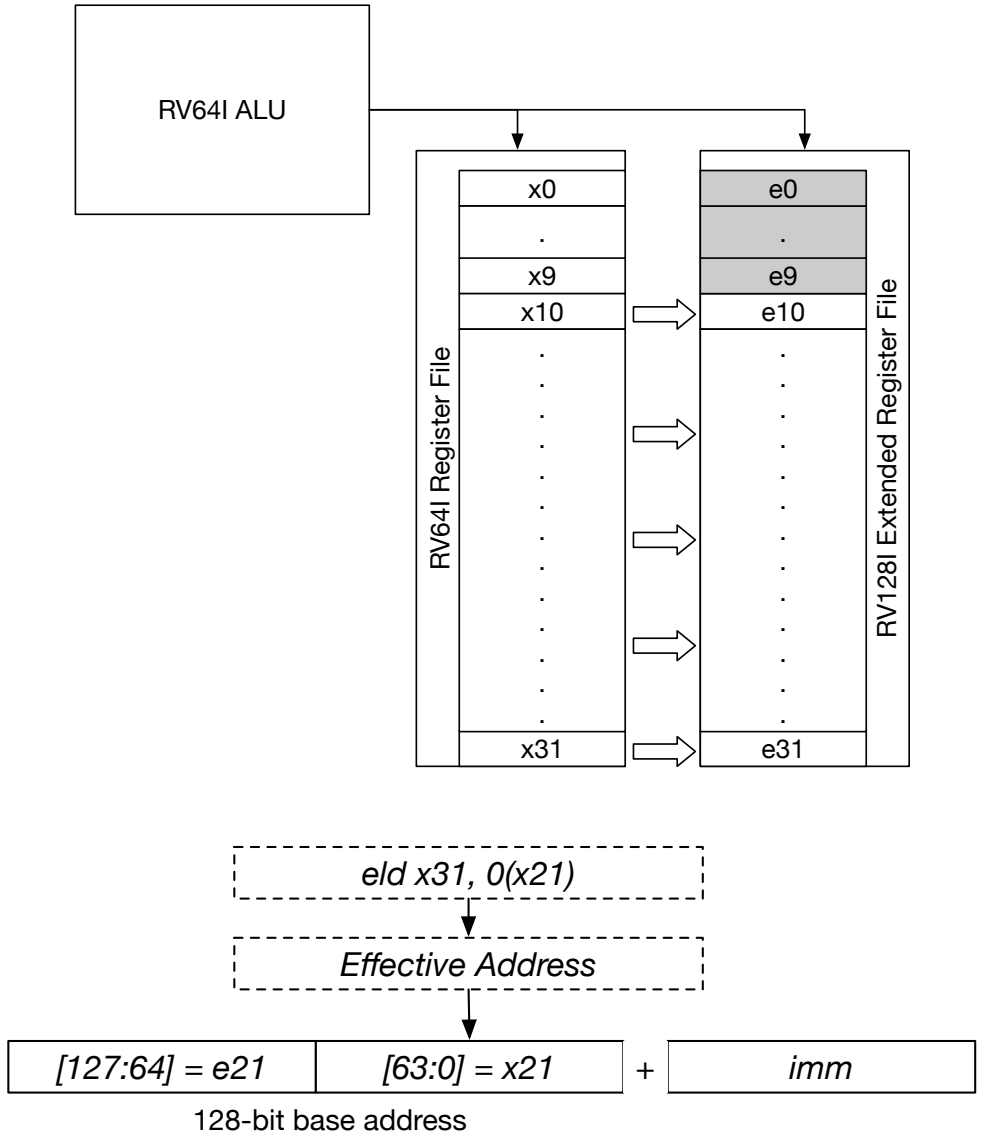
Tactical Computing Labs

Massachusetts Institute of Technology

BOSTON UNIVERSITY

Addressing Architecture

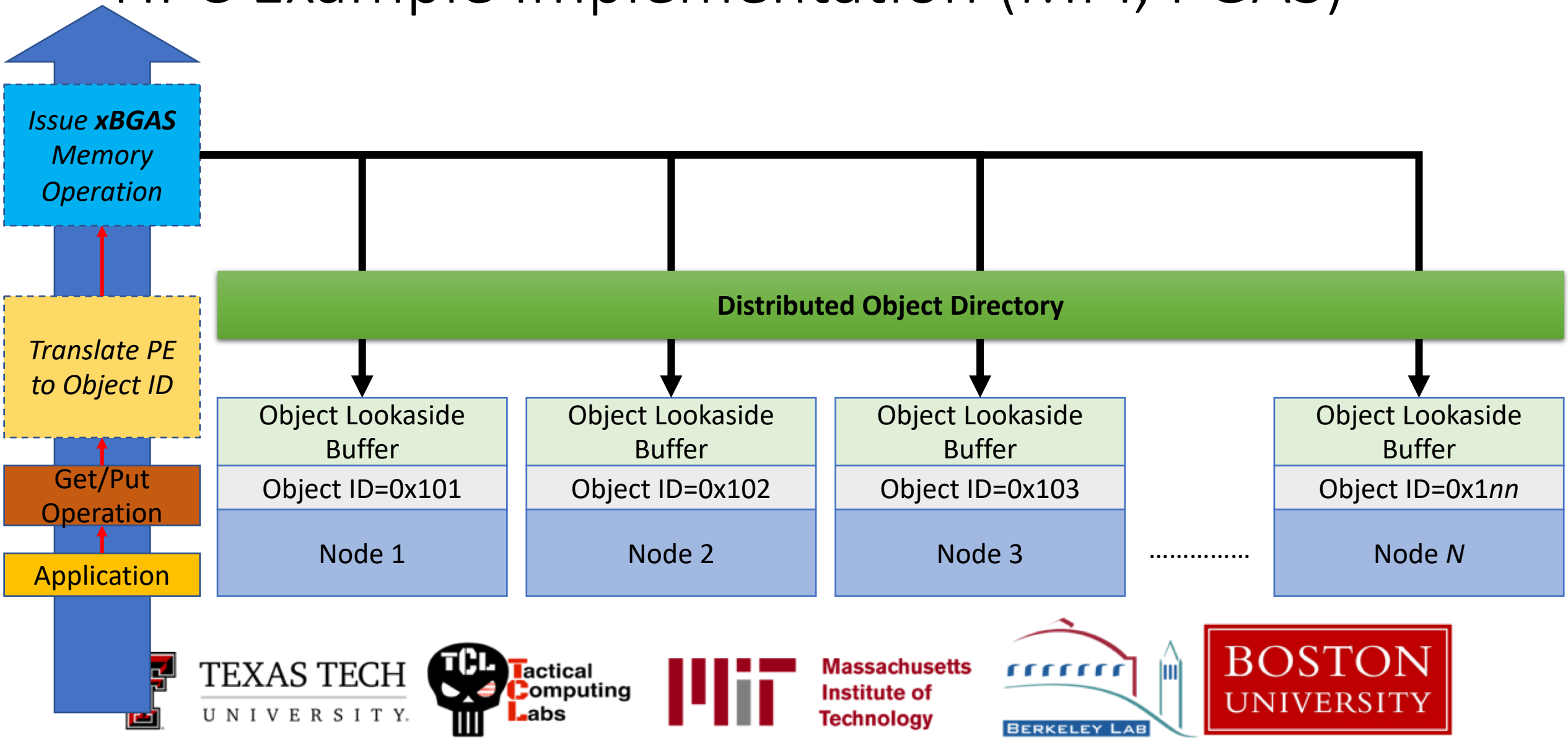
- uArch maps extended addressing into RV64
  - We hope to generalize this for RV32 as well
- CSR bits encoded to appear as standard RV64 uArch
  - XLEN maps to RV64
  - TBD whether we need additional interrupts and exceptions
- Addition of *extended* {eN} registers that map to base general registers
- Extended registers are manually utilized via extended load/store/move instructions



ISA Extension

- Instructions are split into three blocks:
  - Base integer load/store
  - Raw integer load/store
  - Address management
- Base integer load/store (I-type)
  - Permits loading/storing all base RV64I data types using standard mnemonic
  - EX: ***eld rd, imm(rs1)***
  - The extended register mapped to the same index as ’rs1’ is implied
- Raw integer load/store (R-type)
  - Permits loading/storing using explicit extended registers combined with explicit base registers (no imm)
  - ***erld rd, rs1, ext2***
  - LOAD(ext2[127-64], rs1[63-0])
- Address Management
  - Permits explicit manipulation of the extended register contents
  - ***eaddie extd, rs1, imm***
  - extd = rs1+imm

HPC Example Implementation (MPI, PGAS)



TEXAS TECH UNIVERSITY

TCL Tactical Computing Labs

Massachusetts Institute of Technology

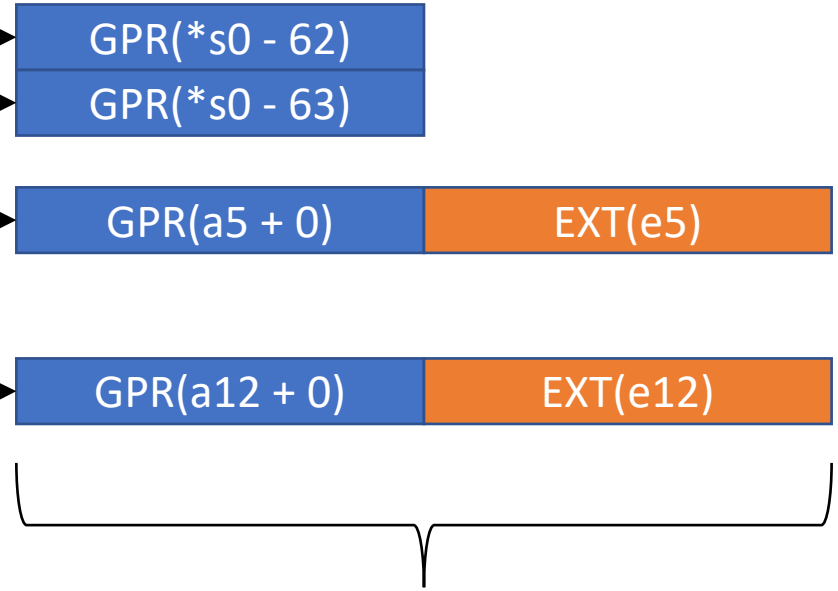
BERKELEY LAB

BOSTON UNIVERSITY

Addressing Example

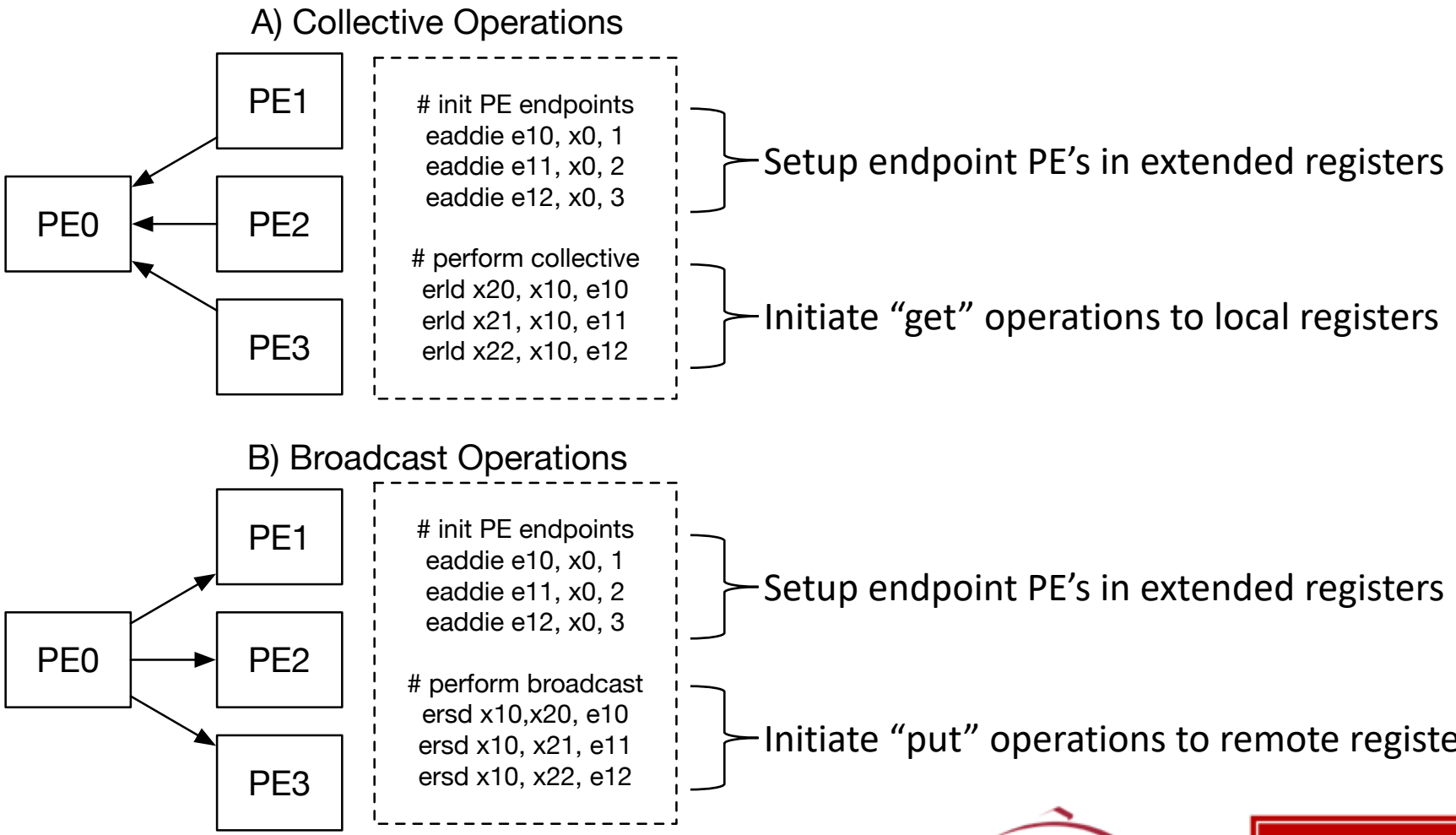
Assembly code from ***xbgas-asm-test***

```
sh zero,-62(s0)
sb zero,-63(s0)
ld a5,-24(s0)
eld a5,0(a5)
sd a5,-56(s0)
ld a5,-32(s0)
elw  a12,0(a12)
sw a5,-60(s0)
ld a5,-40(s0)
elh a5,0(a5)
sh a5,-62(s0)
ld a5,-48(s0)
elb a5,0(a5)
sb a5,-63(s0)
ld a5,-40(s0)
elhu a5,0(a5)
```



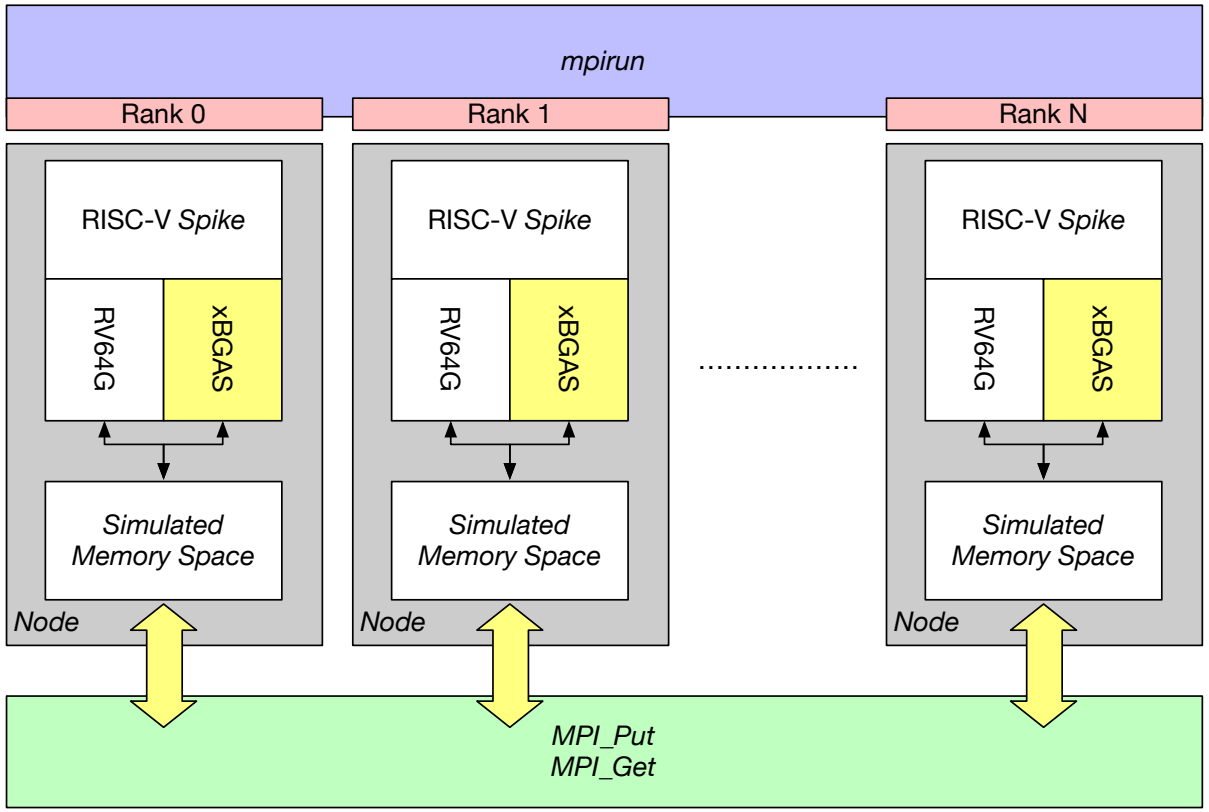
- *Up to 128 bits of address space*
- *Not necessarily contiguous!*
- *Most significant (extended) address can be object ID (as opposed to raw address)*

Collectives and Broadcasts



xBGAS Simulation Infrastructure

- Simulator based upon the RISC-V *Spike* functional simulation infrastructure
- Extended to support all xBGAS machine state/instructions
- Utilizes MPI within the simulator to enable multi-{cpu, node, etc} simulation



TEXAS TECH UNIVERSITY
Tactical Computing Labs
Massachusetts Institute of Technology
BERKELEY LAB
BOSTON UNIVERSITY

xBGAS Runtime

- Machine-level runtime library designed to mimic OpenSHMEM functionality
- Currently supports all get/put interfaces for all OpenSHMEM data types in synchronous and asynchronous modes
- Performance optimization to permit overlapping compute/communication (weak memory ordering)
  - Much of this is written in assembly
- Lacks:
  - Atomics
  - High performance collectives/broadcasts
  - High performance barrier (current implementation is simple)

Ongoing Research

TEXAS TECH UNIVERSITY

Tactical Computing Labs

Massachusetts Institute of Technology

BERKELEY LAB

BOSTON UNIVERSITY

Research & Progress

- Software
  - Data Intensive Scalable Computing Lab at Texas Tech is leading the software research
  - Current xBGAS spec implemented in LLVM & GNU compilers
  - Simulation infrastructure in place with Spike
  - SST simulator coming online
- Hardware
  - TCL/LBNL/MIT leading hardware effort
  - Exploring pipelined and accelerator-based implementations
  - Pipelined implementation has begun in Freechips Rocket
  - Also exploring tightly coupled implementation alongside off-chip interconnects (GenZ)
- Other Topics
  - Operating system (context save info)
  - Debugging
  - Programming Model

Community Support & Interest

- xBGAS spec available on Github
  - https://github.com/tactcomplabs/xbgas-archspec
- RISC-V Tools Branch from Priv-1.10 initial implementation
  - https://github.com/tactcomplabs/xbgas-tools
  - Includes xBGAS GNU and LLVM tool chains
  - Spike implementation ongoing
- ISA Tests
  - https://github.com/tactcomplabs/xbgas-asm-test
- Runtime Library
  - https://github.com/tactcomplabs/xbgas-runtime
- We welcome comments/collaborators!

Acknowledgements

- Bruce Jacob: University of Maryland
- Steve Wallach: Micron

TEXAS TECH UNIVERSITY

Tactical Computing Labs

Massachusetts Institute of Technology

BERKELEY LAB

BOSTON UNIVERSITY

xBGAS

TEXAS TECH UNIVERSITY

Tactical Computing Labs

Massachusetts Institute of Technology

BOSTON UNIVERSITY

ABI (Calling Convention)

- This is where things get tricky…
- The base RV{32,64} ABI defines:
  - Context save/restore space
  - Call/return register utilization
  - Caller/Callee saved state
  - Core data types
- We want to preserve as much as possible while providing extended addressing

- Many outstanding questions
  - How do we link base RV objects with objects containing extended addressing?
  - How do we address the caller/callee saved state with extended registers?
  - Debugging and debugging metadata?

ISA Extension Encodings

Base Integer Load/Store

Mnemonic	base	funct3	dest	opcode
eld rd, imm(rs1)	rs1+ext1	011	rd	1110111
elw rd, imm(rs1)	rs1+ext1	010	rd	1110111
elh rd, imm(rs1)	rs1+ext1	001	rd	1110111
elhu rd, imm(rs1)	rs1+ext1	101	rd	1110111
elb rd, imm(rs1)	rs1+ext1	000	rd	1110111
elbu rd, imm(rs1)	rs1+ext1	100	rd	1110111

Mnemonic	src	base	funct3	opcode
esd rs1, imm(rs2)	rs1	rs2+ext2	011	1111011
esw rs1, imm(rs2)	rs1	rs2+ext2	010	1111011
esh rs1, imm(rs2)	rs1	rs2+ext2	001	1111011
esb rs1, imm(rs2)	rs1	rs2+ext2	000	1111011

Mnemonic	base	funct3	dest	opcode
elq rd, imm(rs1)	rs1+ext1	110	rd	1110111
ele extd, imm(rs1)	rs1+ext1	111	rd	1110111

Mnemonic	src	base	funct3	opcode
esq rs1, imm(rs2)	rs1	rs2+ext2	100	1111011
ese ext1, imm(rs2)	ext1	rs2+ext2	101	1111011

Raw Integer Load/Store

Mnemonic	funct7	rs2	rs1	funct3	rd	opcode
erld rd, rs1, ext2	0000010	ext2	rs1	011	rd	0111111
erlw rd, rs1, ext2	0000010	ext2	rs1	010	rd	0111111
erlh rd, rs1, ext2	0000010	ext2	rs1	001	rd	0111111
erlhu rd, rs1, ext2	0000010	ext2	rs1	101	rd	0111111
erlb rd, rs1, ext2	0000010	ext2	rs1	000	rd	0111111
erlbu rd, rs1, ext2	0000010	ext2	rs1	100	rd	0111111
erle extd, rs1, ext2	0000011	ext2	rs1	100	extd	0111111
ersd rs1, rs2, ext3	0000100	rs2	rs1	011	rs1	0111111
ersw rs1, rs2, ext3	0000100	rs2	rs1	010	rs1	0111111
ersh rs1, rs2, ext3	0000100	rs2	rs1	001	rs1	0111111
ersb rs1, rs2, ext3	0000100	rs2	rs1	000	rs1	0111111
erse ext1, rs2, ext3	0001000	rs2	ext1	011	rs1	0111111

Floating point? Atomics?

ISA Extension Encodings cont.

Address Management

Mnemonic	base	funct3	dest	opcode
eaddi rd, ext1, imm	ext1	110	rd	1111011
eaddie extd, rs1, imm	rs1	111	extd	1111011
eaddix extd, ext1, imm	extd	111	ext1	0000011

Assembly Mnemonics

Mnemonic	Base Instruction
movebe rd, ext1	eaddi rd, ext1, 0
moveeb extd, rs1	eaddie extd, rs1, 0
moveee extd, ext1	eaddix extd, ext1, 0

Moving data between GPR and EXT registers