Towards Fast Remote Atomic Object Reads for In-Memory Rack-Scale Computing

Alexandros Daglis,† Dmitrii Ustiugov,† Stanko Novakovic,† Babak Falsafi,† Boris Grot‡

†EcoCloud, EPFL ‡University of Edinburgh

Distributed In-Memory Processing Systems

Large-scale online services
- Vast datasets distributed across hundreds of servers
- Data kept memory-resident to meet tight latency goals
- Data organized in distributed object stores (e.g., Key-Value stores)

Frequent fine-grain communication
- Conventional networking: remote memory latency ~1000x of local
- Shrinks the benefit of keeping data in memory

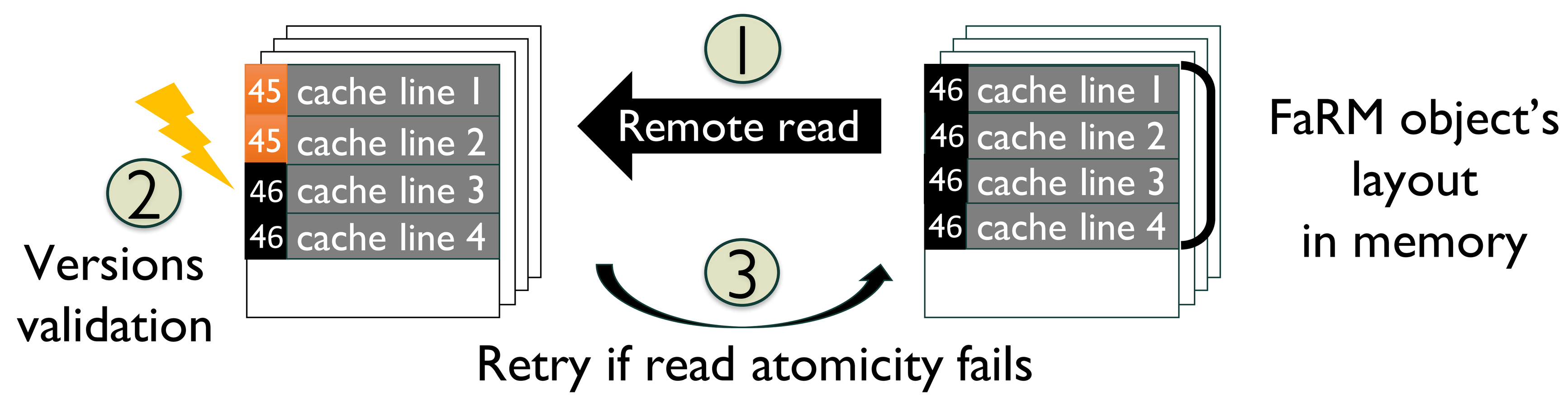
RDMA one-sided operations for fast remote memory access
- Remote memory access within 10-20x of local
- But limited semantics: no atomicity beyond a single cache line

Need to rely on software mechanisms for atomic object reads

Software-Based Atomic Remote Object Reads

Current approach: embedded metadata in every object
- **FaRM**: Per-cache-line object versions
  - ☹ Need to extract application's useful data
- **Pilaf**: Per-object CRC codes
  - ☹ High CPU overhead (~10 cycles per byte)

***Example*: Remote atomic object read in FaRM**

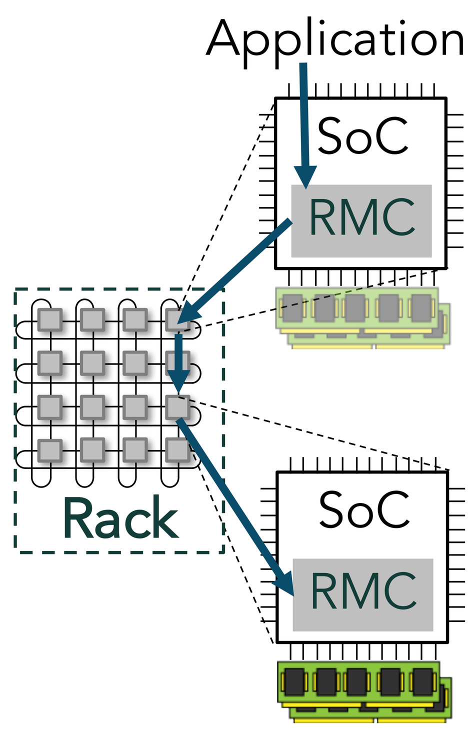


Software checks only add minimal overhead to RDMA remote reads

Rack-Scale Systems for Fast Remote Memory

Emerging rack-scale systems
- Lean user-level protocols, tight integration, high-performance fabrics
- Bring remote memory access latency down to a bare minimum
- E.g., HP's Moonshot & The Machine, AMD SeaMicro, Oracle Exadata

Case study: Scale-Out NUMA



Scale-Out NUMA in a nutshell
- Lean user-level communication protocol
- Low-latency, high-bandwidth memory fabric
- Intra- but not inter-SoC coherence
- RDMA-like one-sided operations
- Remote Memory Controller (RMC)
  - Integrated in SoC's coherence domain

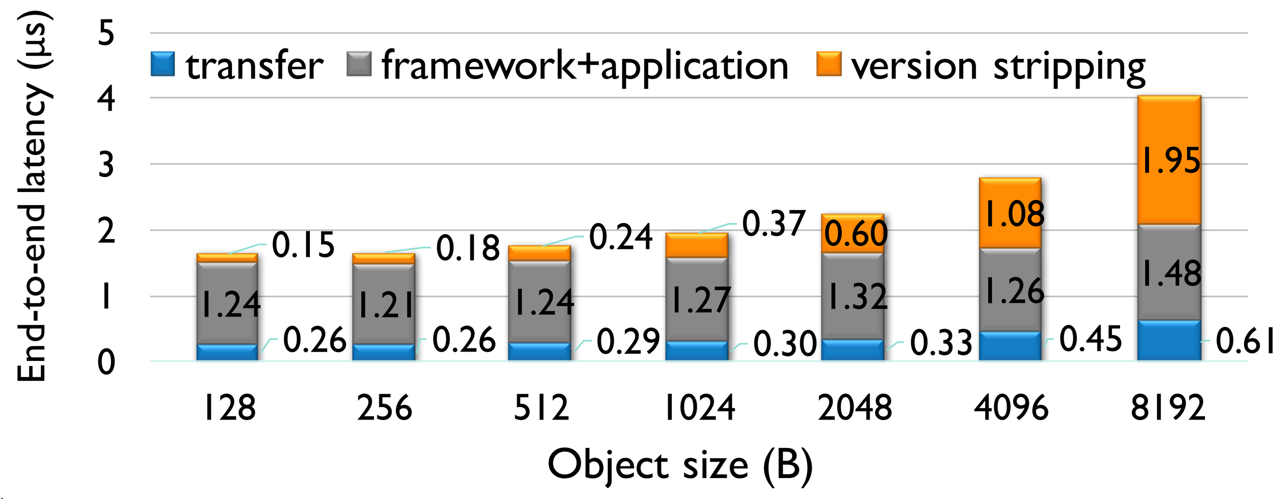
Remote memory access ≈ 4x local

Software overhead starts to perceivably affect end-to-end latency

Evaluation of Software Overhead

Methodology
- Flexus full-system, cycle-accurate simulation
- Two directly attached 16core soNUMA nodes

FaRM benchmark: synchronous remote object reads
1. Remote object reads over soNUMA
2. Software-based object atomicity validation (per-cache-line versions)



Results
- Software atomicity check significant fraction of end-to-end latency
- Hardware support can reduce end-to-end latency by up to **50%**

Hardware support for atomic object reads necessary for low latency

Atomic Object Reads in Hardware: Design Space

Where to provide object atomicity?

At the Source	**At the destination**
Intrusive data layout modifications	Can keep native data layout
Limited to post-transfer checks	Can leverage local coherence integration
Late conflict detection	Early conflict detection

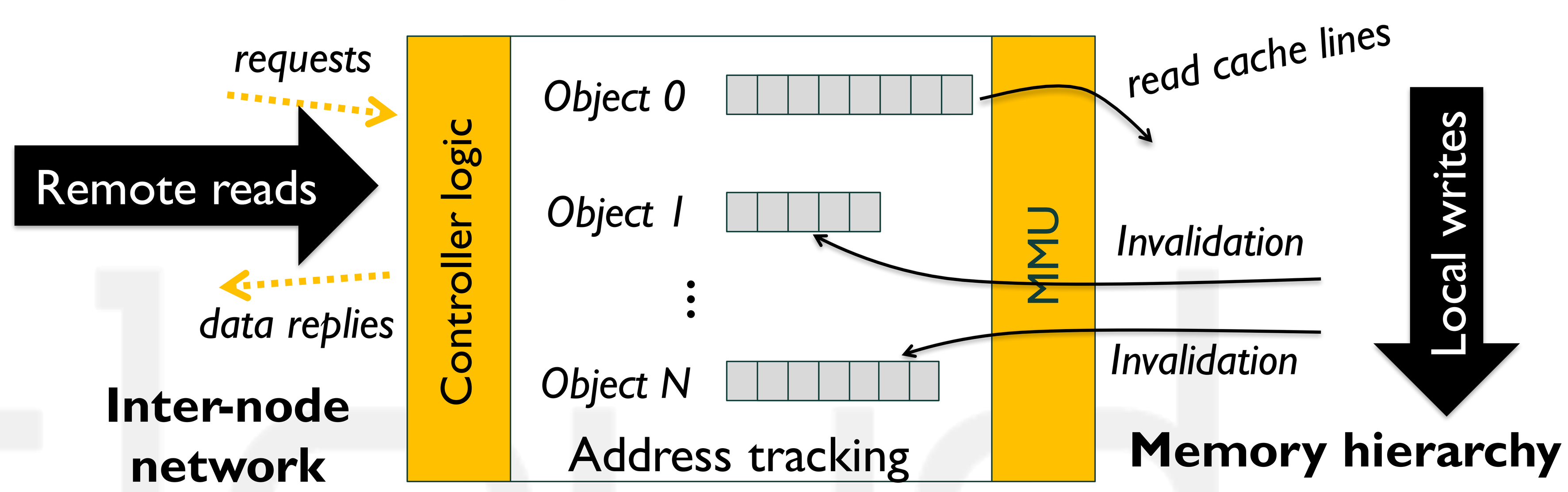
Destination-based designs are inherently superior

Destination-based hardware for atomicity checks: Design Goals
- Maximum concurrency (across multiple object reads)
- Minimum latency (for a single object read)
- Minimum hardware complexity/cost (keep hardware simple)

Towards Efficient Hardware Support

Insight: leverage hardware/software contract to simplify hardware
- Objects are well-defined software structures
  - Object header with a lock or a version
- Object spans range of consecutive physical addresses

One-sided ops controller at destination



Our goal: Simple hardware for zero-overhead atomic object reads

Microsoft Research nano-tera.ch DCSL PARSA PARALLEL SYSTEMS ARCHITECTURE LAB EPFL ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE