# A Coherent and Managed Runtime for ML on the SCC

**KC Sivaramakrishnan**
*Purdue University*

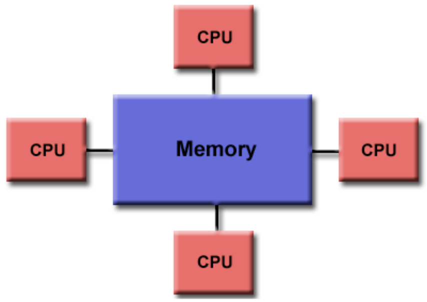
Lukasz Ziarek
*SUNY Buffalo*

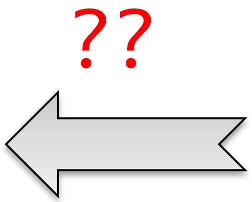
Suresh Jagannathan
*Purdue University*

PURDUE
UNIVERSITY

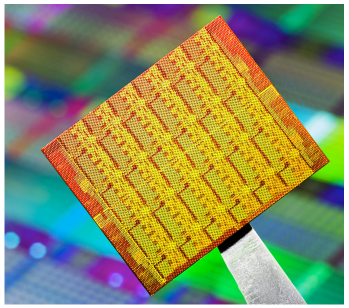
# Big Picture

**Cache Coherent**





**Intel SCC**



**Cluster of Machines**

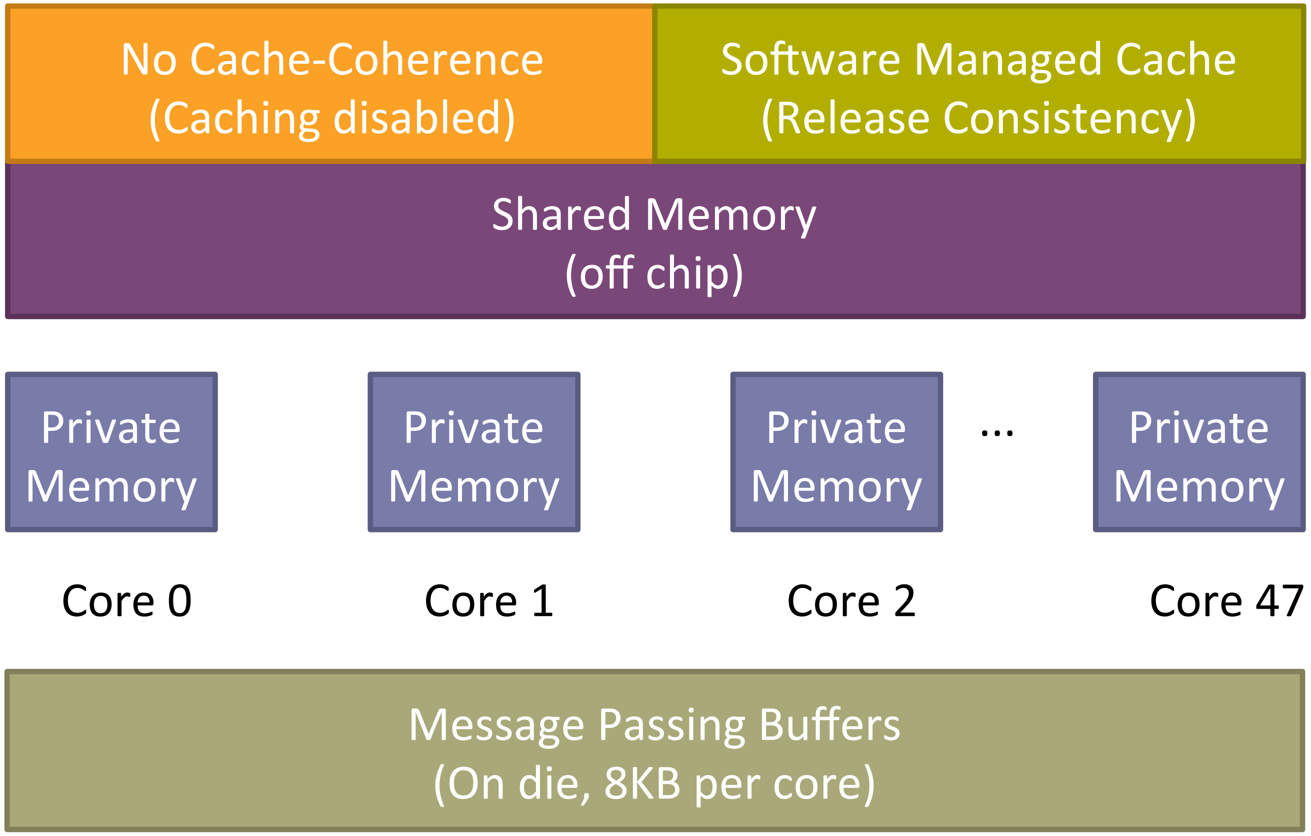
✓ No change to programming model
✓ Automatic memory management

- No cache coherence
- Message passing buffers
- *Shared memory*
- *Software Managed Cache-Coherence (SMC)*

- Distributed programming
- RCCE, MPI, TCP/IP

Can we program SCC as a cache coherent machine?

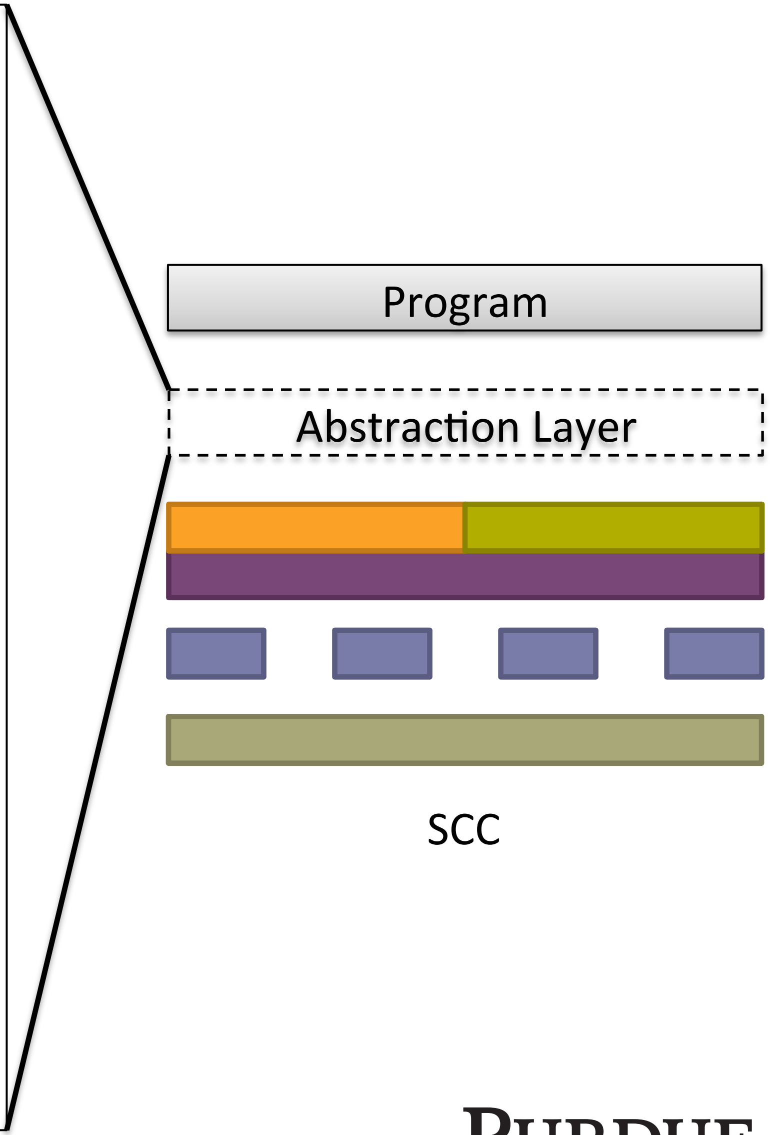
# Intel SCC Architecture



How to provide an *efficient* cache coherence layer?

# SMP Programming Model for SCC

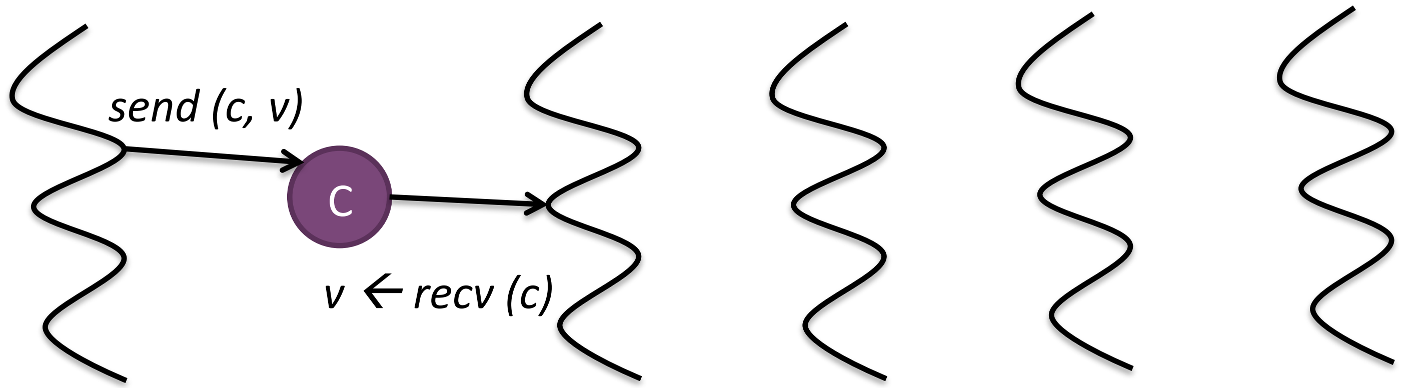
- Desirable properties
  - Single address space
  - Cache coherence
  - Sequential consistency
  - Automatic memory management
  - Utilize MPB for inter-core communication
- Abstraction Layer – MultiMLton VM
  1. A new GC to provide coherent and managed global address space
  2. Mapping first-class channel communication on to the MPB



PURDUE UNIVERSITY

# Programming Model

- MultiMLton
  - Safety, scalability, ready for future manycore processors
  - Parallel extension of MLton – a whole-program, optimizing Standard ML compiler
  - Immutability is default, mutations are explicit
- ACML – first-class message passing language



- Automatic memory management

## Coherent and Managed Address Space

## Coherent and Managed Address Space

- Requirements
  1. Single global address space
  2. Memory consistency
  3. Independent core-local GC

*Thread-local GC!*

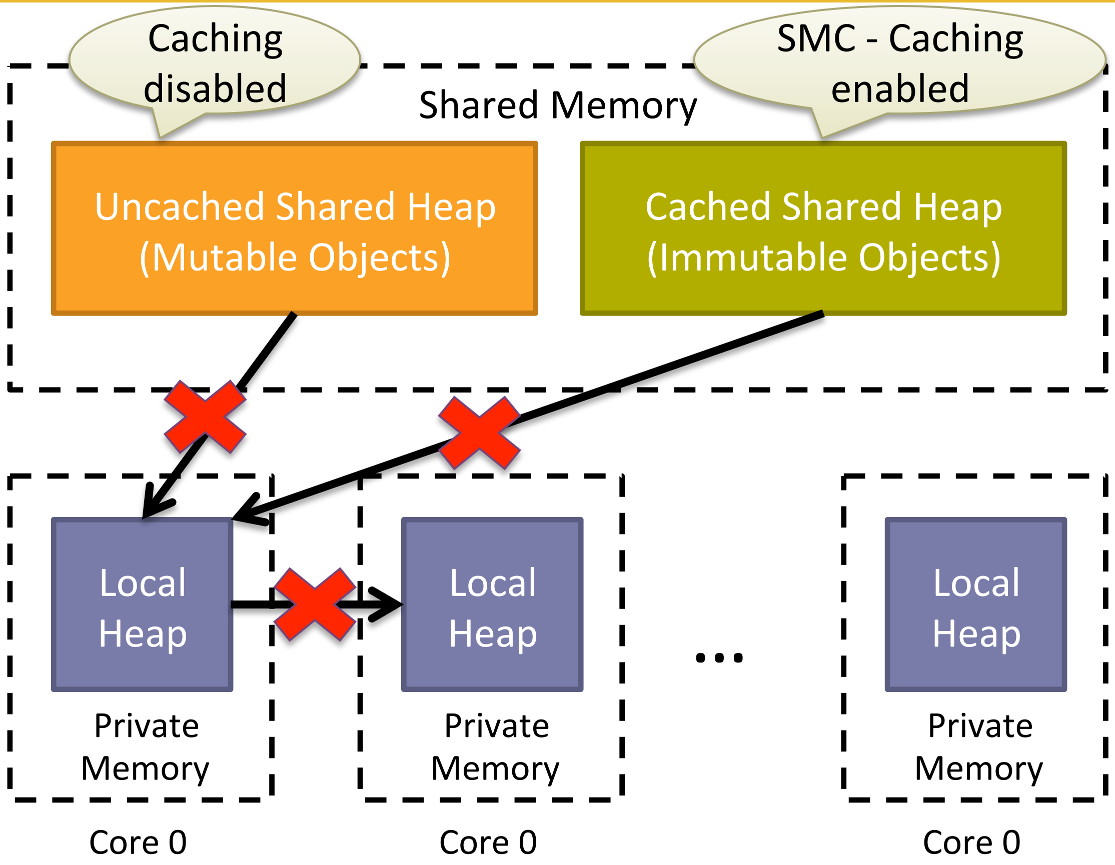
Private-nursery GC

Local heap GC

On-the-fly GC

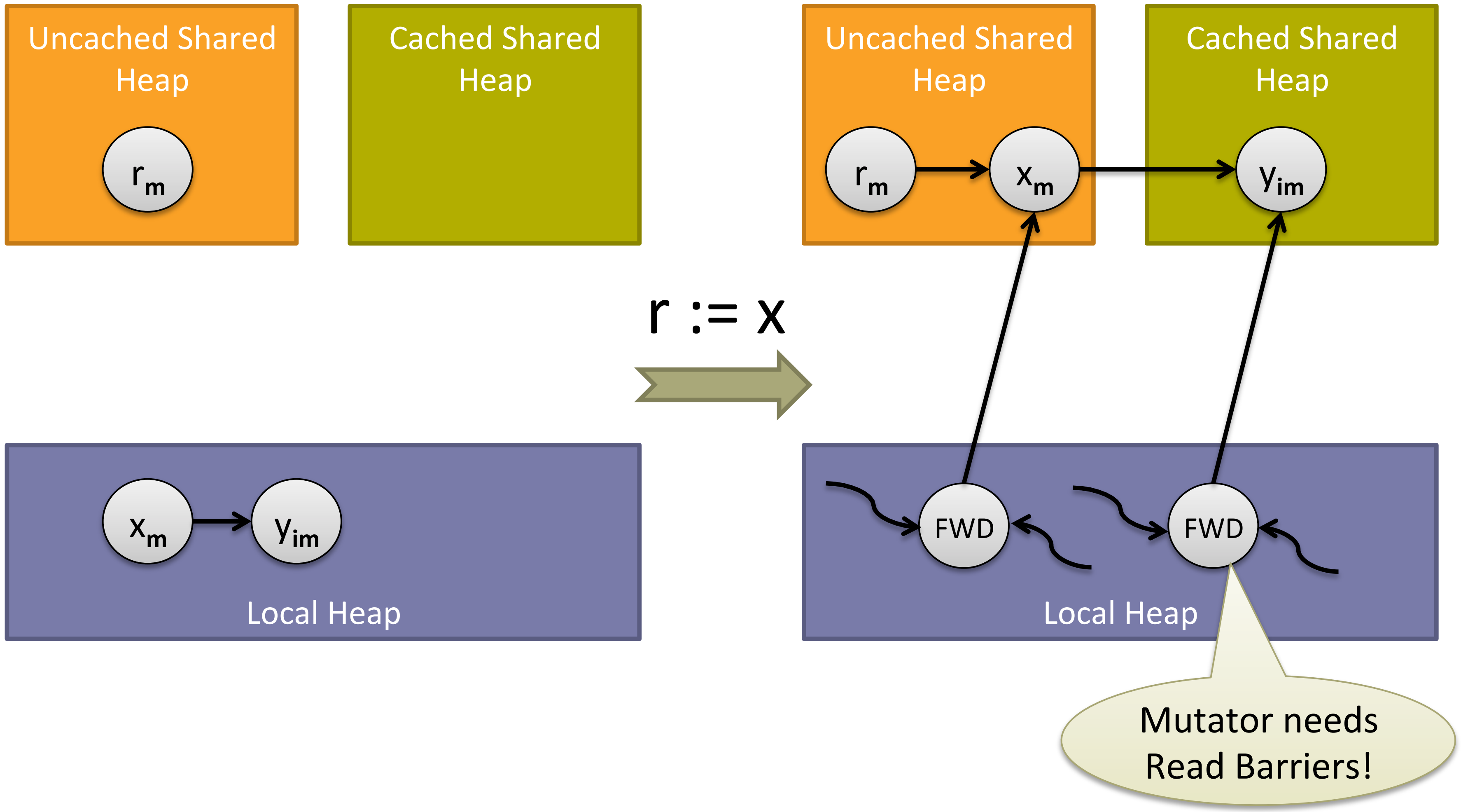
Thread-specific heap GC

# Thread-local GC for SCC



- Consistency preservation
  - No inter-coherence-domain pointers!
- Independent collection of local heaps

# Heap Invariant Preservation

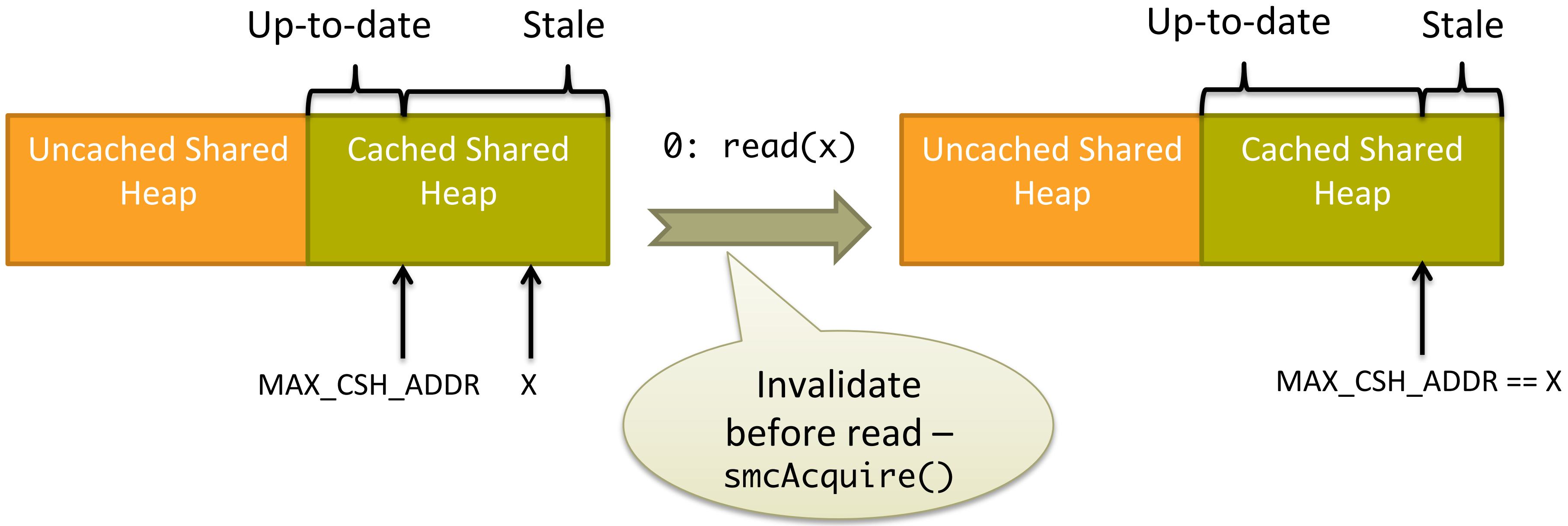


# Maintaining Consistency

- Local heap objects are not shared by definition
- Uncached shared heap is consistent by construction
- Cached shared heap (CSH) uses SMC
  - *Invalidation* and *flush* has to be managed by the runtime
  - Unwise to *invalidate before every CSH read* and *flush after every CSH write*
- Solution
  - Key observation: CSH only stores immutable objects!

# Ensuring Consistency (Reads)

- Maintain MAX_CSH_ADDR at each core
- Assume values at ADDR < MAX_CSH_ADDR are up-to-date



# Ensuring Consistency (Reads)

- No need to invalidate before `read (y)` where

  `y < MAX_CSH_ADDR`

- Why?
  1. Bump pointer allocation
  2. All objects in CSH are immutable

`y < MAX_CSH_ADDR` → *Cache invalidated after y was created*

## Ensuring Consistency (Writes)

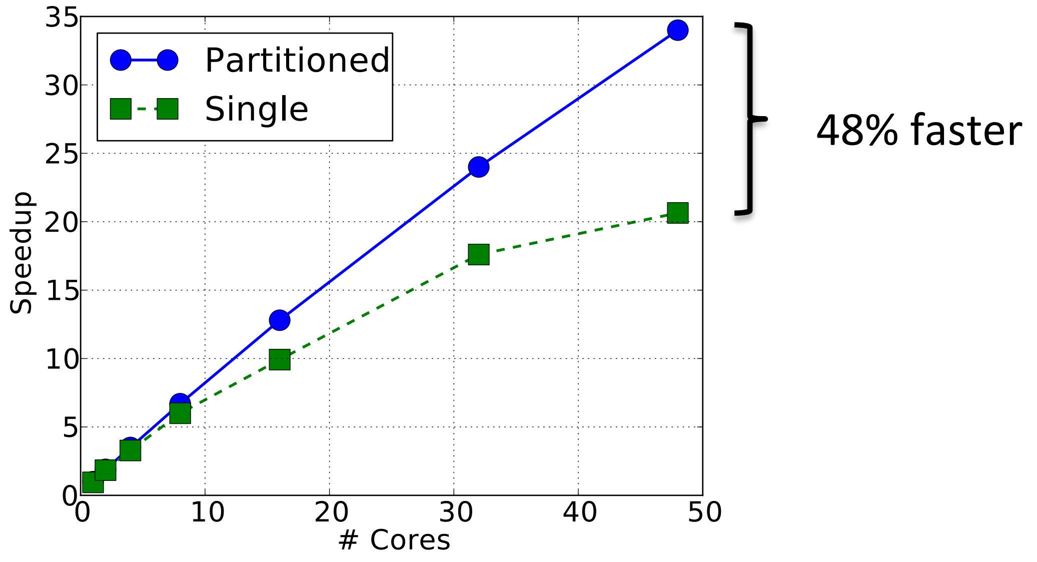
- Writes to shared heap occurs only during globalization
- Flush cache after globalization
  - `smcRelease()`

# Garbage Collection

- Local heaps are collected independently!
- Shared heaps are collected after stopping all of the cores
  - Proceeds in SPMD fashion
  - Each core prepares the shared heap reachable set independently
  - One core collects the shared heap
- Sansom's dual mode GC
  - A good fit for SCC!

# GC Evaluation

- 8 MultiMLton benchmarks



- Memory Access profile
  - 89% local heap, 10% cached shared heap, 1% uncached shared heap
  - *Almost all accesses are cacheable!*

## ACML Channels on MPB

# ACML Channels on MPB

- Challenges
  - First-class objects
  - Multiple senders/receivers can share the same channel
  - Unbounded
  - Synchronous and asynchronous
- Channel Implementation

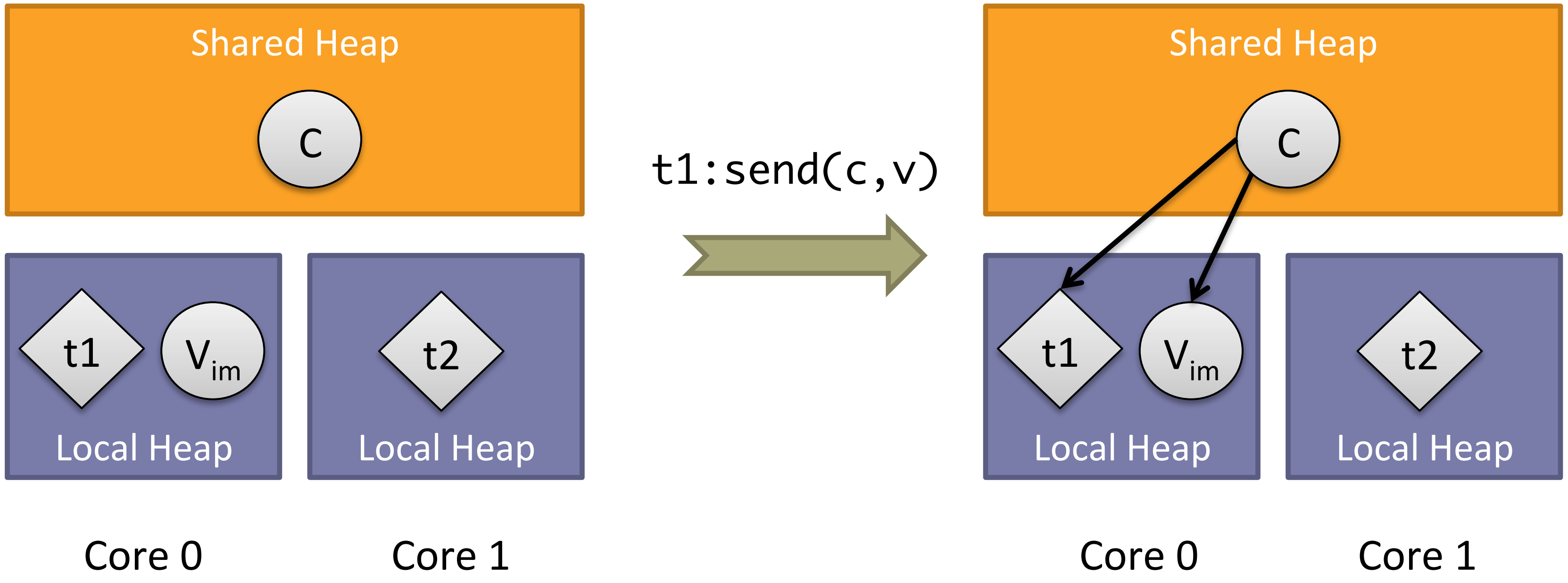
```
datatype ‘a chan = {sendQ : (‘a * unit thread) Q.t,
                    recvQ : (‘a thread) Q.t}
```

## Specializing Channel Communication

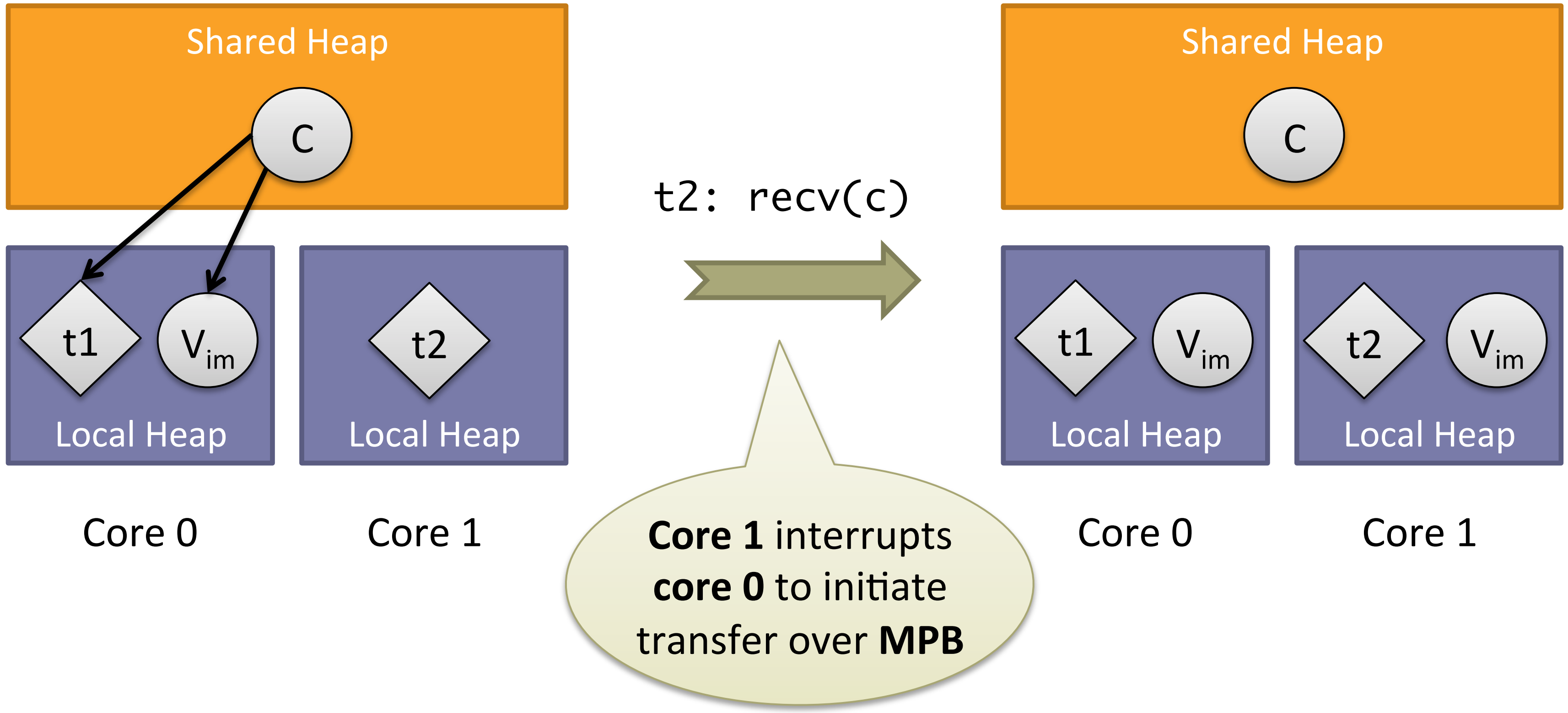
- Mutable messages must be globalized
  - Must maintain consistency
- Immutable messages can utilize MPB

# Sender Blocks

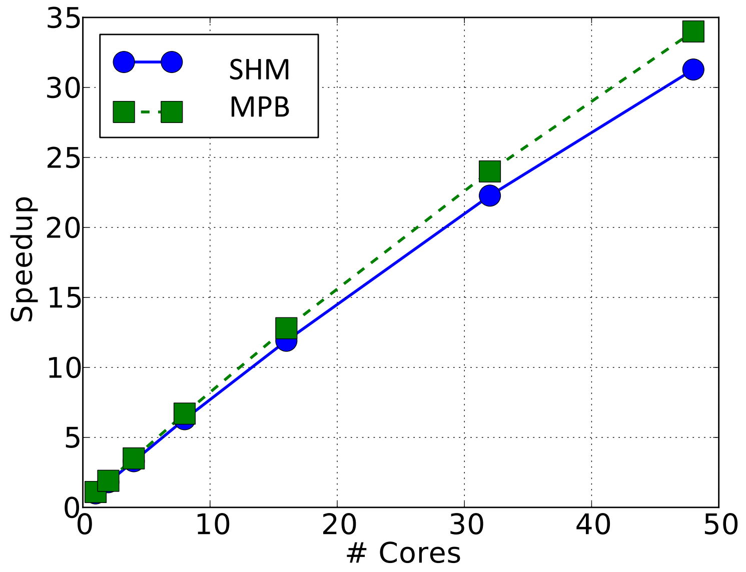
- Channel in shared heap, message is immutable and in local heap



# Receiver Interrupts



# Message Passing Evaluation



- On 48-cores, MPB only 9% faster
- Inter-core interrupt are expensive
  - Context switches + idling cores
  - Polling is not an option due to user-level threading

## Conclusion

- **Cache coherent runtime for ML on SCC**
  - Thread-local GC
    - Single address space, Cache coherence, Concurrent collections
    - Most memory accesses are cacheable
  - Channel communication over MPB
    - Inter-core interrupts are expensive

## Questions?

MultiMLton

http://multimlton.cs.purdue.edu

PURDUE UNIVERSITY

# Read Barrier

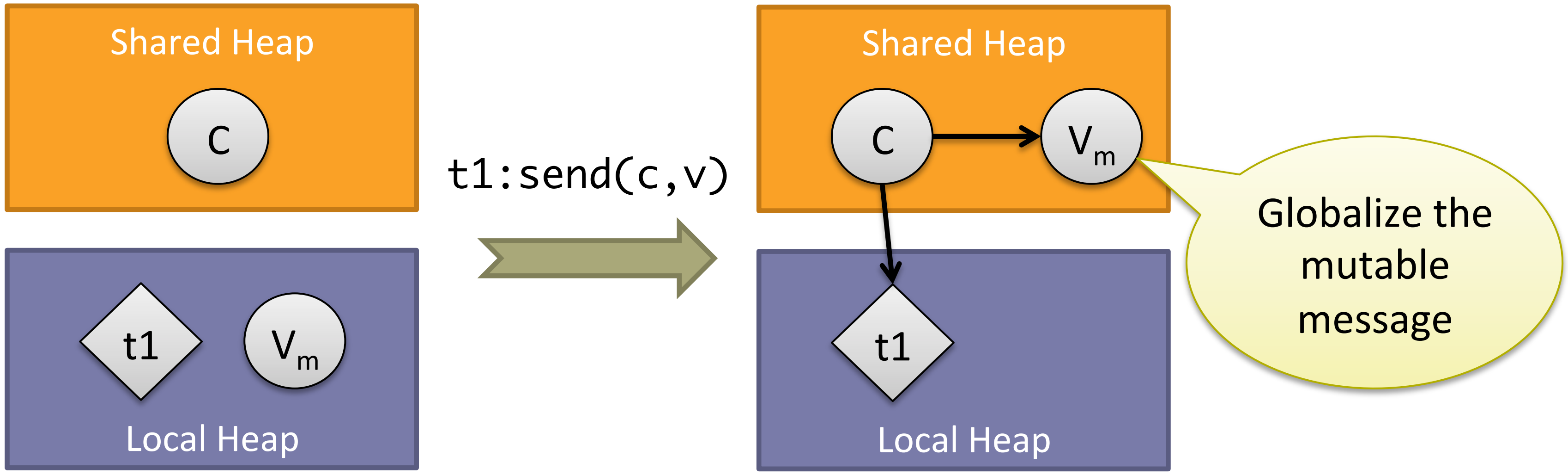
```
pointer readBarrier (pointer p) {
  if (getHeader (p) == FORWARDED) {
    //A globalized object
    p = *(pointer*)p;
    if (p > MAX_CSH_ADDR) {
      smcAcquire ();
      MAX_CSH_ADDR = p;
    }
  }
  return p;
}
```

# Write Barrier

```
val writeBarrier (Ref r, Val v) {
  if (isObjptr (v) && isInSharedHeap (r) &&
      isInLocalHeap (v)) {
      v = globalize (v);
      smcRelease ();
  }
  return v;
}
```

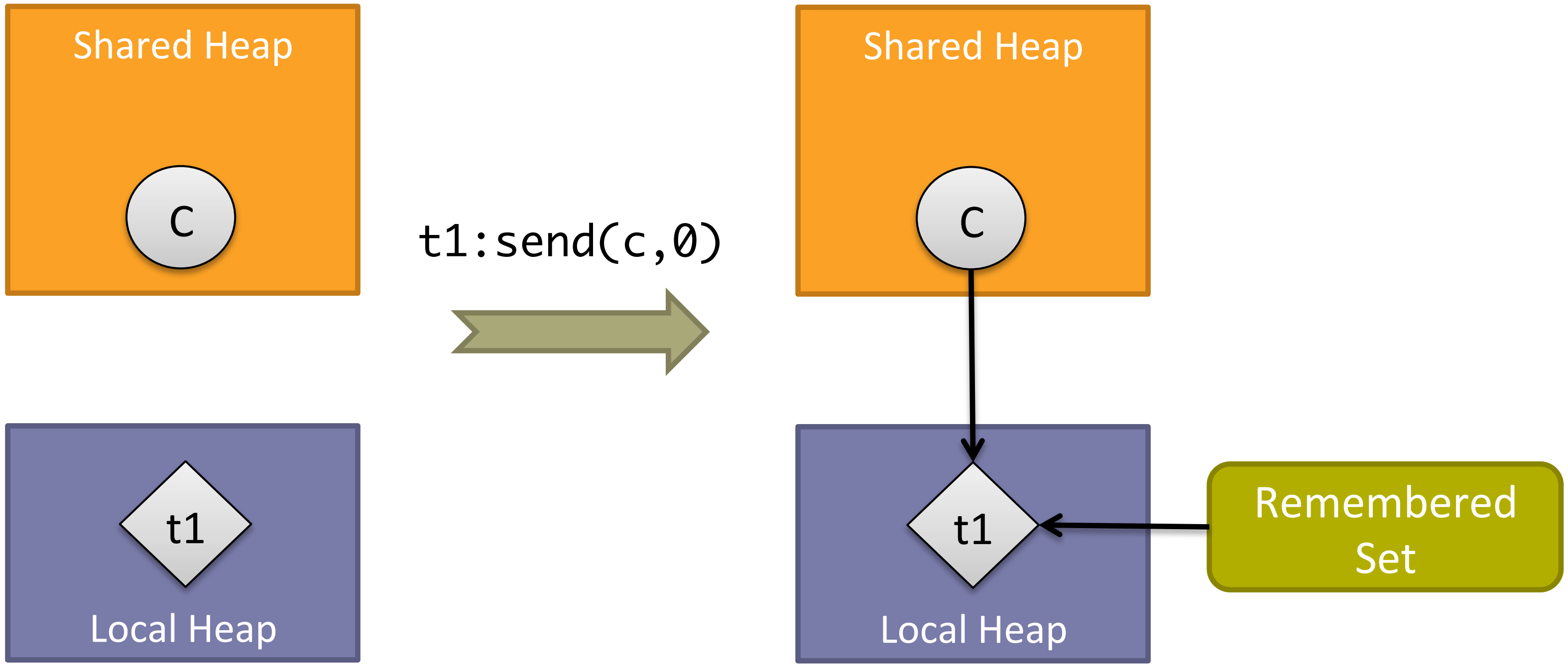
## Case 4

- Channel in shared heap, message is mutable and in local heap



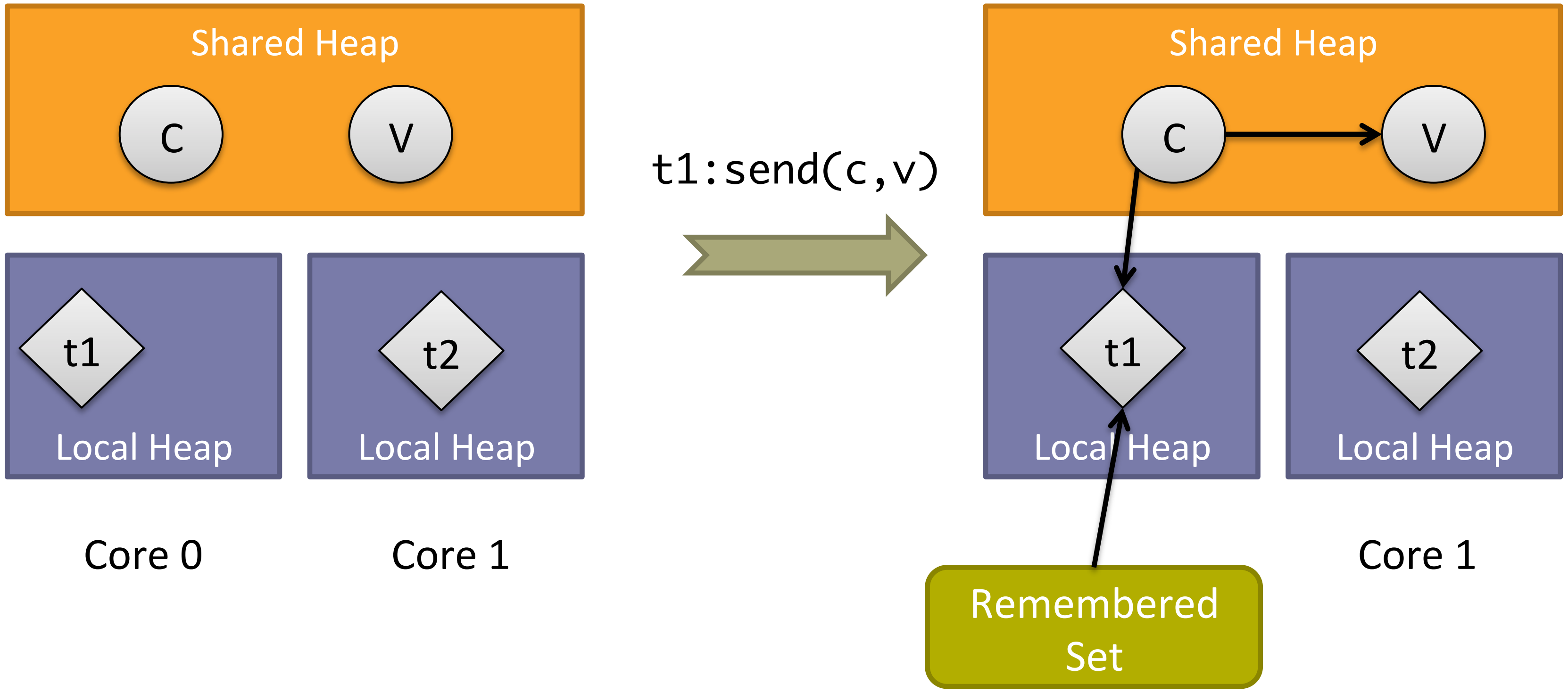
# Case 2

- Channel in Shared heap, Primitive-valued message

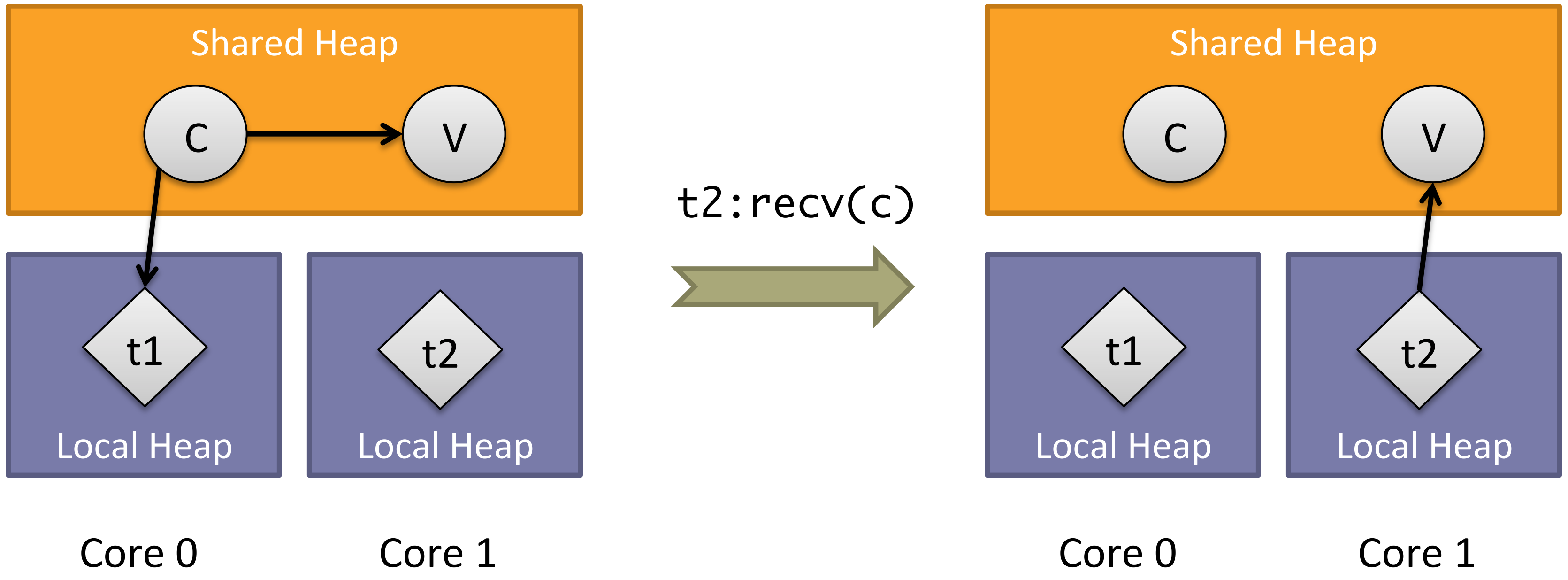


# Case 3 – Sender Blocks

- Channel in shared heap, message in shared heap



# Case 3 – Receiver Unblocks


Shared Heap
c
v
t1
Local Heap
t2
Local Heap
Core 0
Core 1
t2:recv(c)
Shared Heap
c
v
t1
Local Heap
t2
Local Heap
Core 0
Core 1