Software Transactional Memory for Large Scale Clusters

Robert L. Bocchino Jr. and Vikram S. Adve
University of Illinois at Urbana-Champaign

Bradford L. Chamberlain
Cray Inc.

Transactional Memory (TM)

Can simplify parallel programming

Well studied for small-scale, cache-coherent platforms

No prior work on TM for *large scale platforms*

- Potentially thousands of processors
- Distributed memory, no cache coherence
- Slow communication between nodes

Why STM On Clusters?

TM is a natural fit for *PGAS Languages*

- UPC, CAF, Titanium, Chapel, Fortress, X10, ...
- Address space is *global* (unlike message passing)
- But *data distribution is explicit* (unlike cc-NUMA, DSM)

Commodity clusters are in widespread use

Software transactional memory (STM) is natural choice

- Communication done in software anyway
- Could leverage hardware TM support if it exists

What's New About STM on Clusters?

	Classic STM	Cluster STM

What's New About STM on Clusters?

	Classic STM	Cluster STM
Primary overhead	Extra scalar ops	Extra remote ops

What's New About STM on Clusters?

	Classic STM	Cluster STM
Primary overhead	Extra scalar ops	Extra remote ops
Read and write	Words of data	Blocks of data

What's New About STM on Clusters?

	Classic STM	Cluster STM
Primary overhead	Extra scalar ops	Extra remote ops
Read and write	Words of data	Blocks of data
Heap address space	Uniform	Partitioned

What's New About STM on Clusters?

	Classic STM	Cluster STM
Primary overhead	Extra scalar ops	Extra remote ops
Read and write	Words of data	Blocks of data
Heap address space	Uniform	Partitioned
STM metadata	Uniform	Distributed

What's New About STM on Clusters?

	Classic STM	Cluster STM
Primary overhead	Extra scalar ops	Extra remote ops
Read and write	Words of data	Blocks of data
Heap address space	Uniform	Partitioned
STM metadata	Uniform	Distributed
Distributing computation for data locality	N/A	*on p { ... }*

Research Contributions

First STM designed for high performance on large clusters

- Block data movement
- Computation migration
- Distributed metadata

Experimental evaluation of prototype

- Performance vs. locks
- New design tradeoffs

Decomposition of STM design space into eight axes

Outline

Interface Design

Algorithm Design

Evaluation

- Cluster STM vs. Manual Locking
- Read Locks vs. Read Validation

Conclusion

Interface Design

Compiler target, not primarily for programmers

Correct use guarantees *serializability of transactions*

Interface Design

Compiler target, not primarily for programmers

Correct use guarantees *serializability of transactions*

Chapel

```
atomic {
   // Array assignment
   cache = A;
   compute(cache);
   A = cache;
}
```

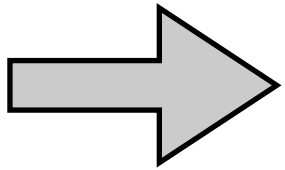
Interface Design

Compiler target, not primarily for programmers

Correct use guarantees *serializability of transactions*

Chapel

```
atomic {
   // Array assignment
   cache = A;
   compute(cache);
   A = cache;
}
```



compiler

Cluster STM API

```
stm_start(...)
stm_get(&cache, &A,...)
compute(&cache);
stm_put(&A, &cache, ...);
stm_commit(...);
```

Interface Summary

Transaction start and commit

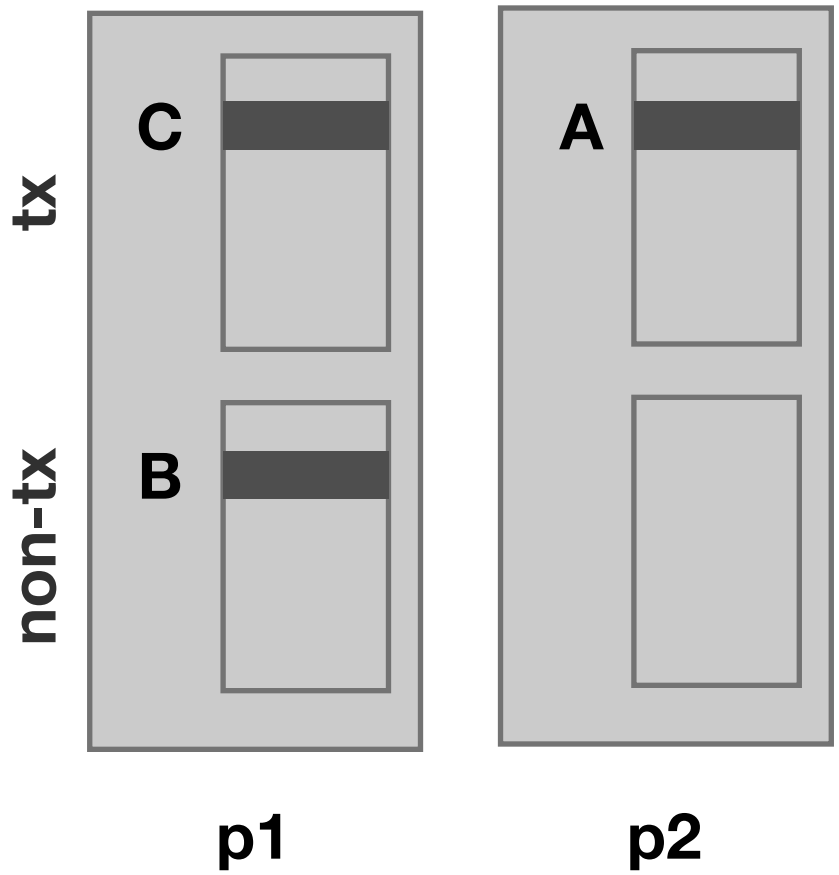
Transactional memory allocation

Block data movement to and from transactional store

Remote execution of transactional computations

Block Data Movement

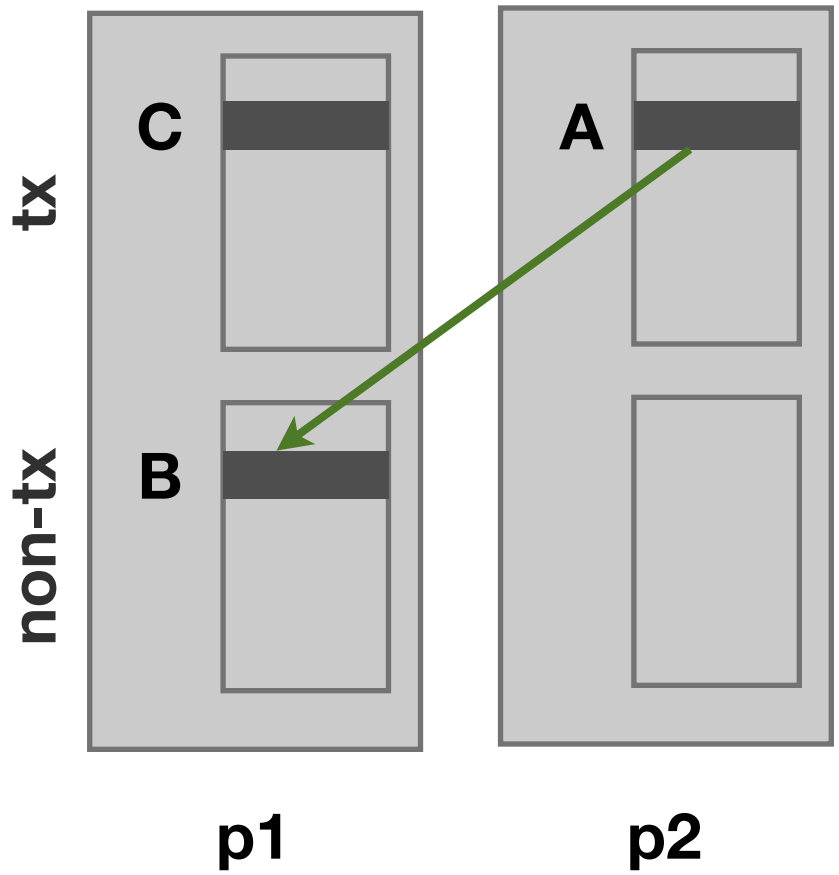
All ops occur on p1



Block Data Movement

All ops occur on p1

stm_get(work_proc=p2, src=A, dest=B, size=n, ...)

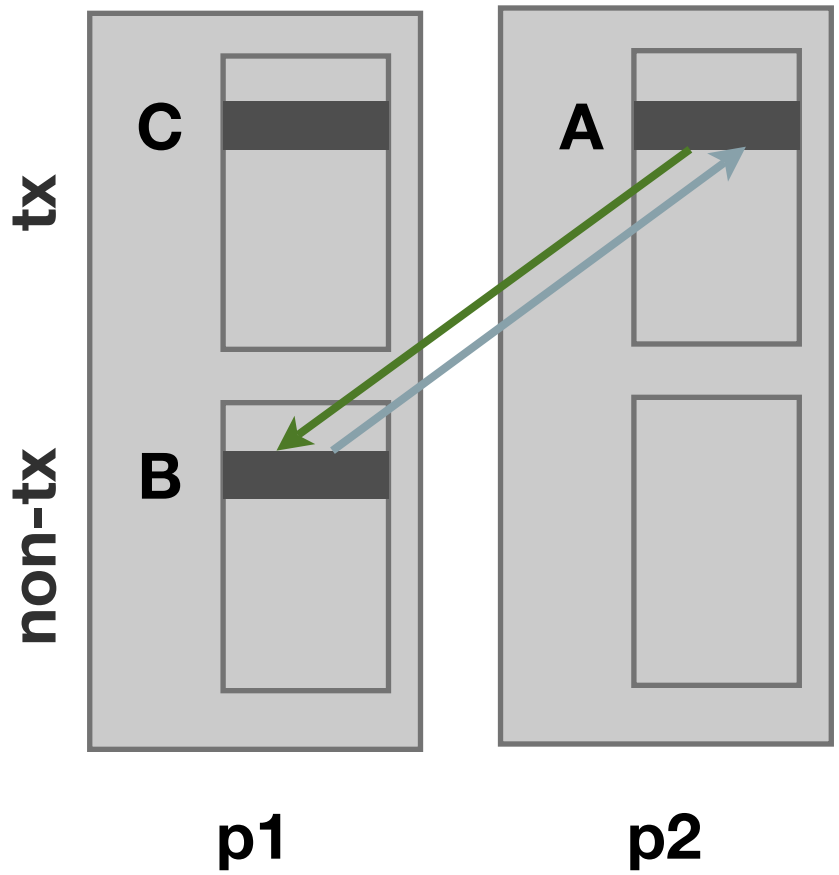


Block Data Movement

All ops occur on p1

stm_get(work_proc=p2, src=A, dest=B, size=n, ...)

stm_put(work_proc=p2, src=B, dest=A, size=n, ...)



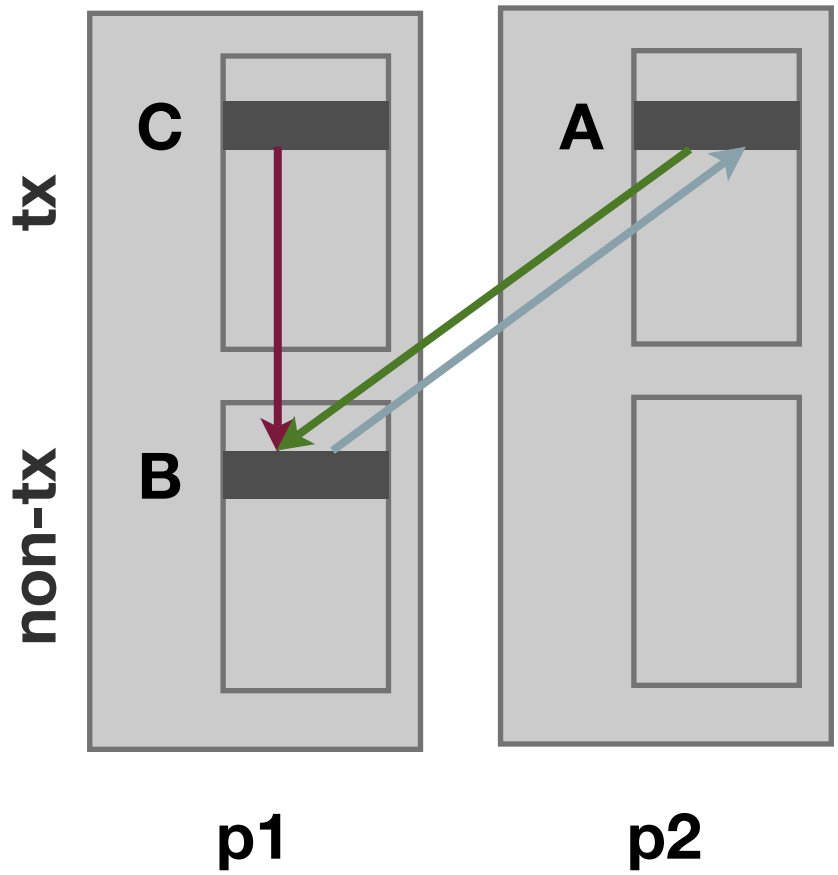
Block Data Movement

All ops occur on p1

stm_get(work_proc=p2, src=A, dest=B, size=n, ...)

stm_put(work_proc=p2, src=B, dest=A, size=n, ...)

stm_read(src=C, dest=B, size=n, ...)



Block Data Movement

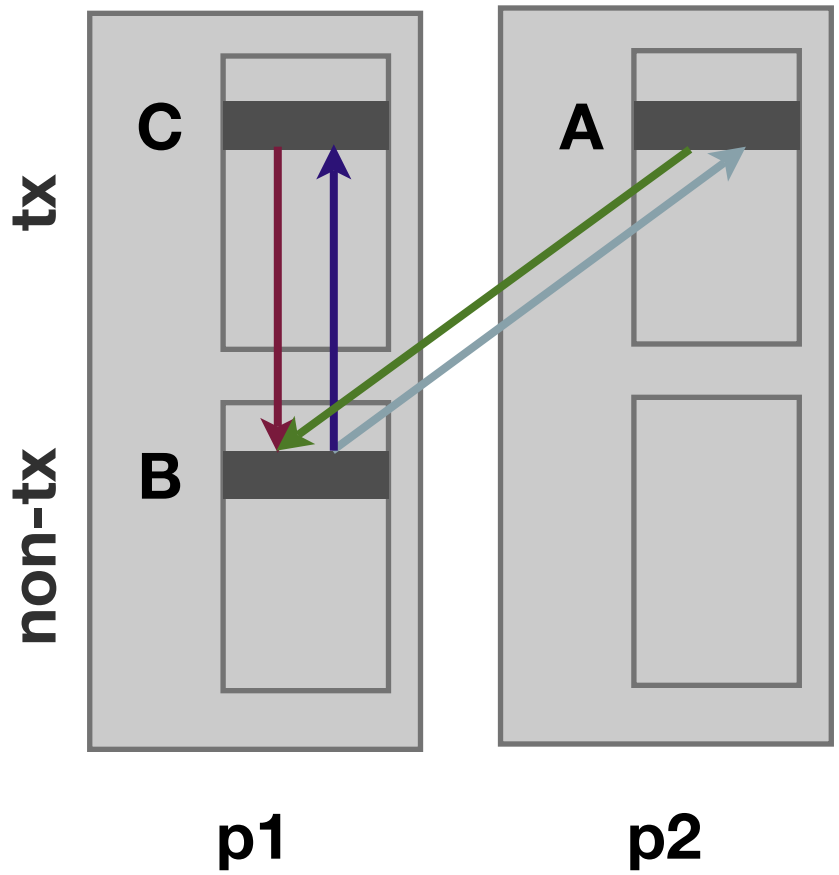
All ops occur on p1

stm_get(work_proc=p2, src=A, dest=B, size=n, ...)

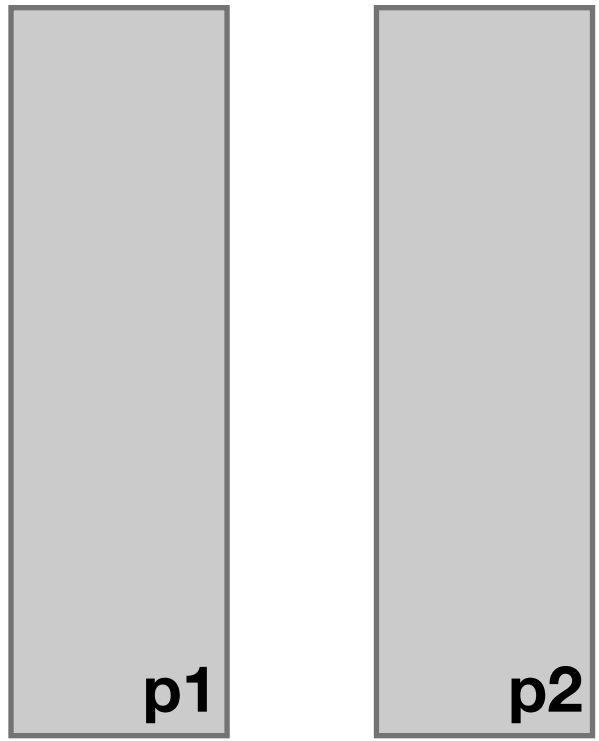
stm_put(work_proc=p2, src=B, dest=A, size=n, ...)

stm_read(src=C, dest=B, size=n, ...)

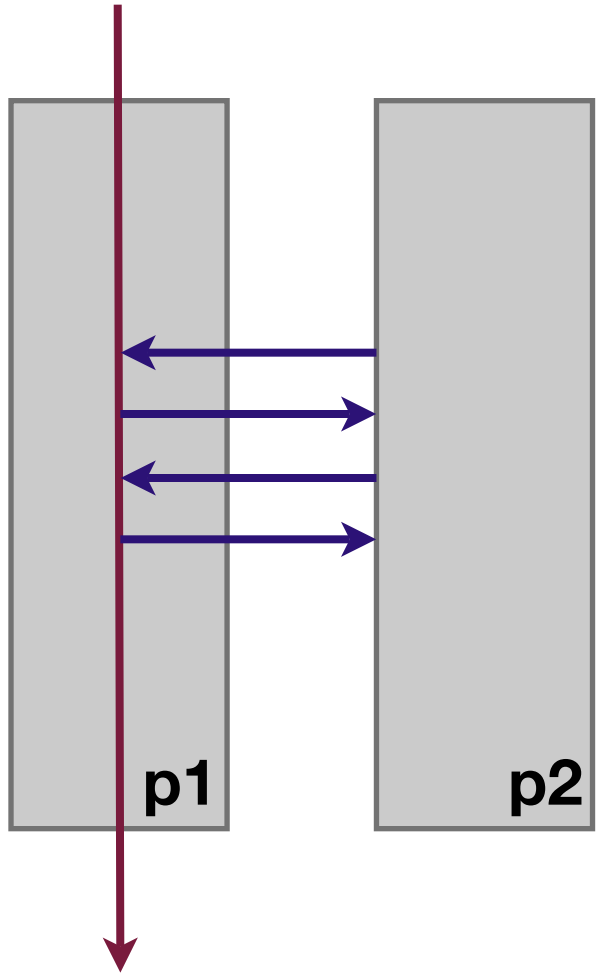
stm_write(src=B, dest=C, size=n, ...)



Remote Work for Exploiting Locality

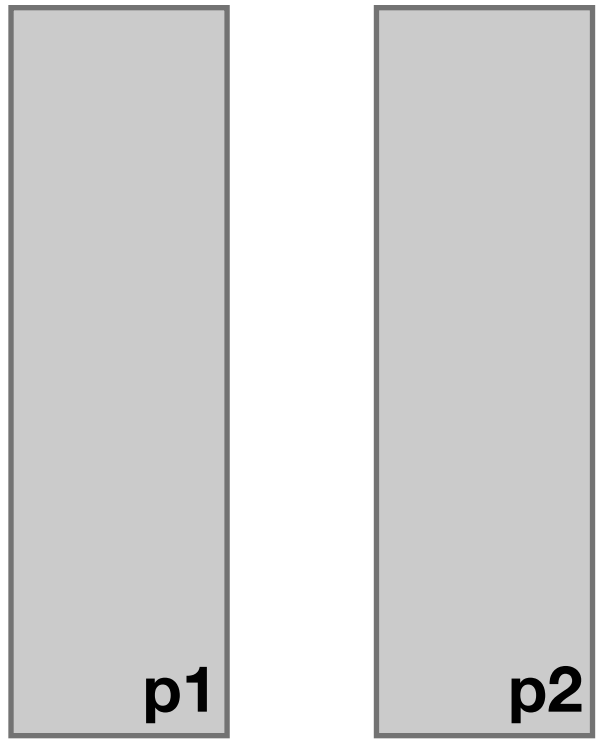


Remote Work for Exploiting Locality

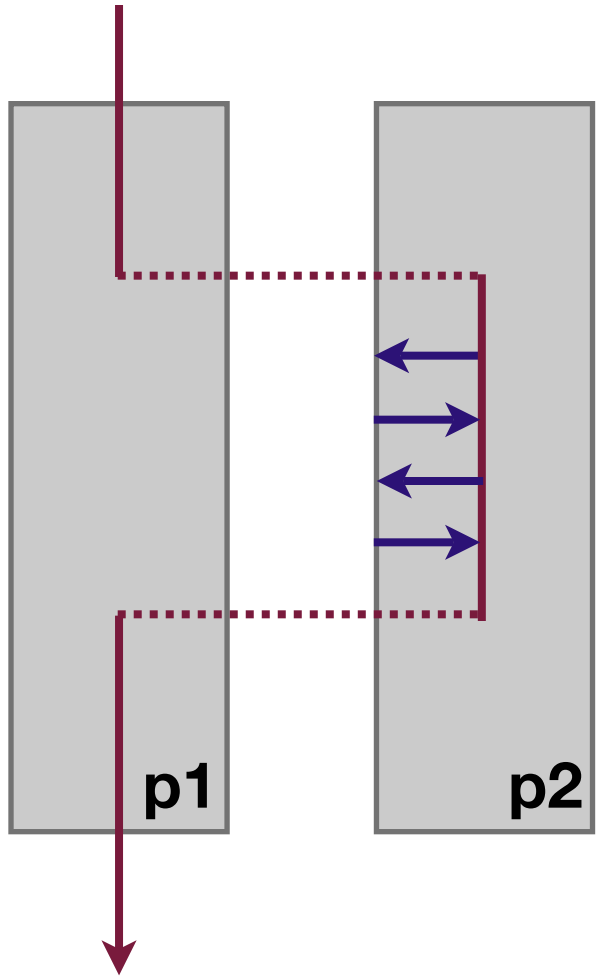


remote puts & gets

Remote Work for Exploiting Locality



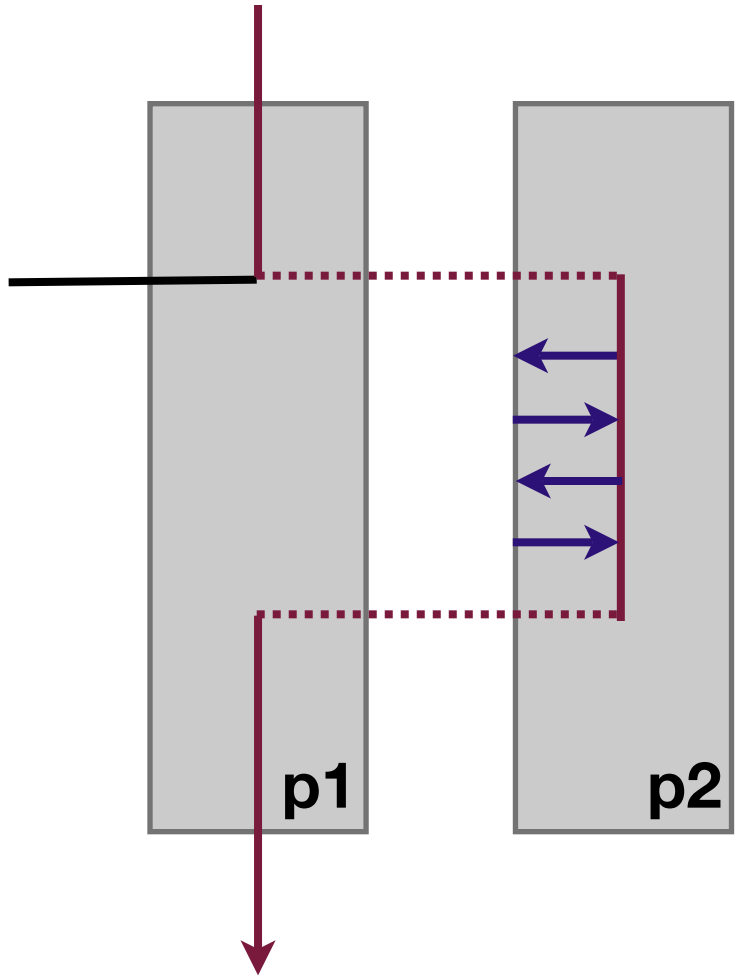
Remote Work for Exploiting Locality



local reads & writes

Remote Work for Exploiting Locality

stm_on(work_proc=p2,
function=f, ...)

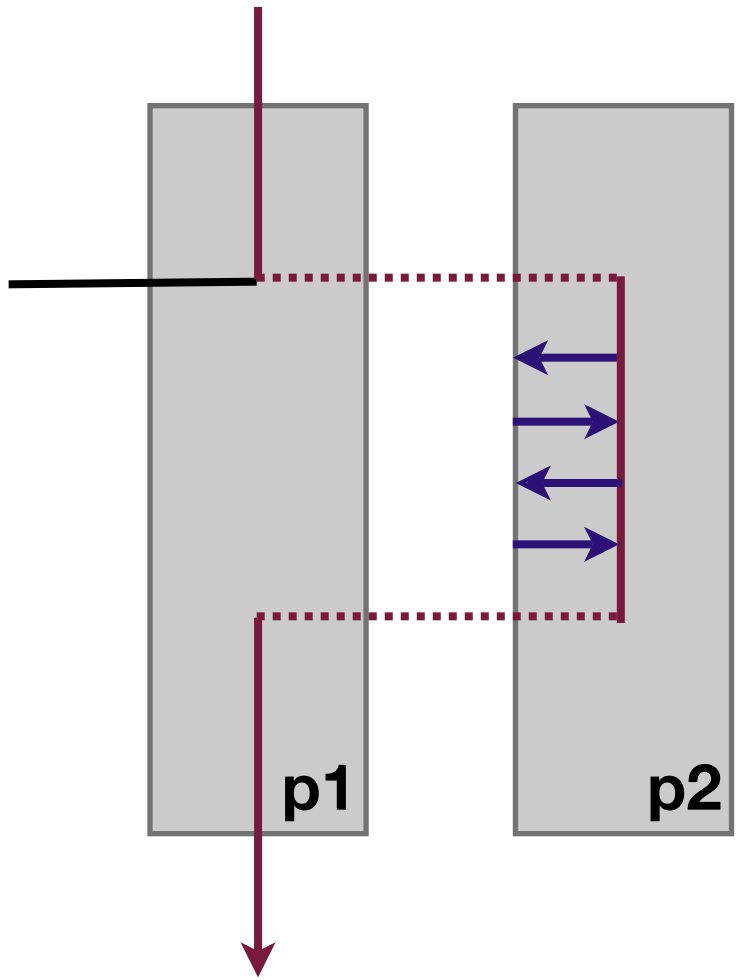


local reads & writes

Remote Work for Exploiting Locality

stm_on(work_proc=p2, function=f, ...)

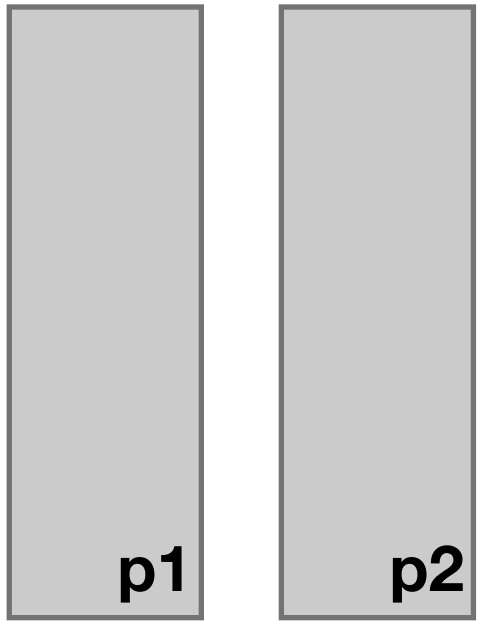
Chapel: on p2 { f(...); }



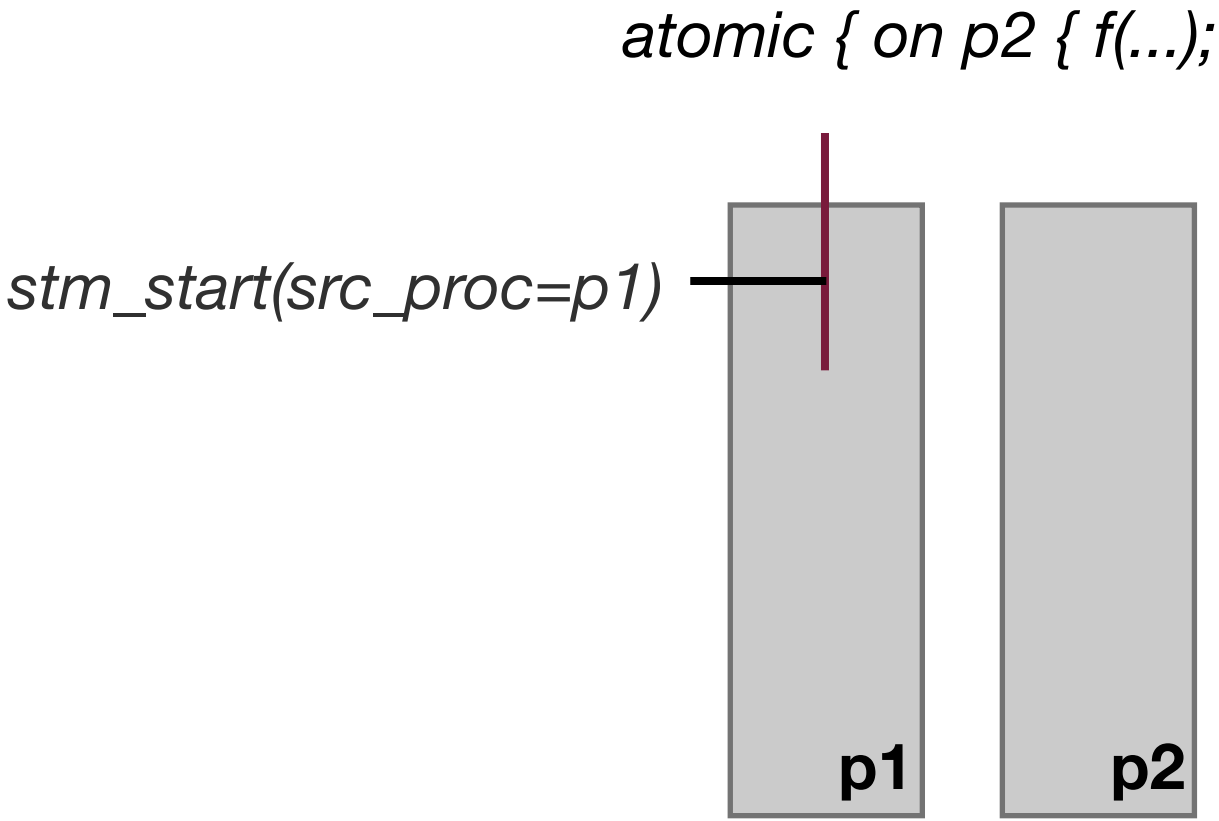
local reads & writes

Remote Work Inside Transaction

atomic { on p2 { f(...); } }



Remote Work Inside Transaction

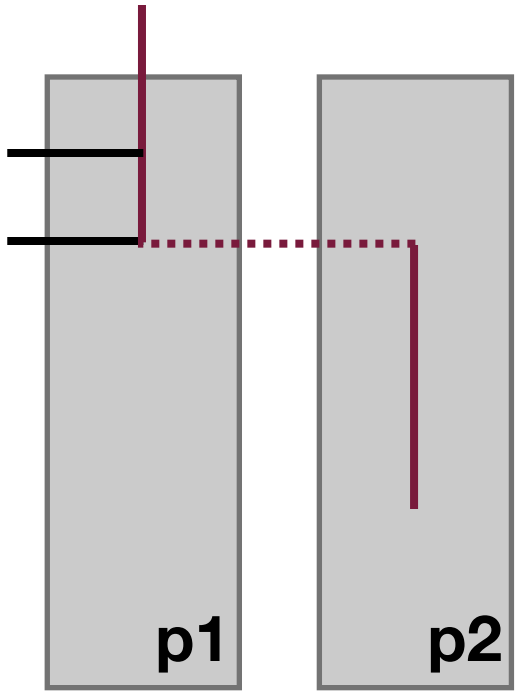


Remote Work Inside Transaction

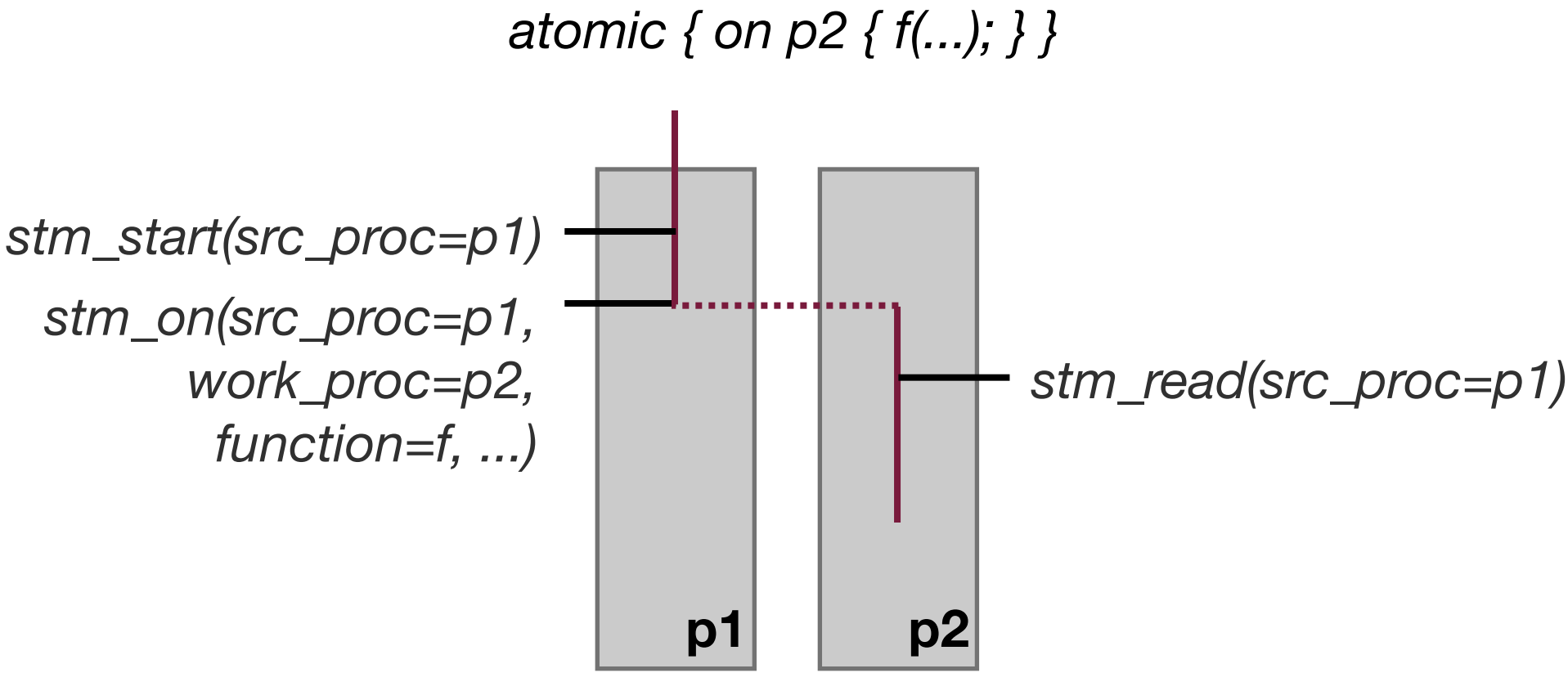
atomic { on p2 { f(...); } }

stm_start(src_proc=p1)

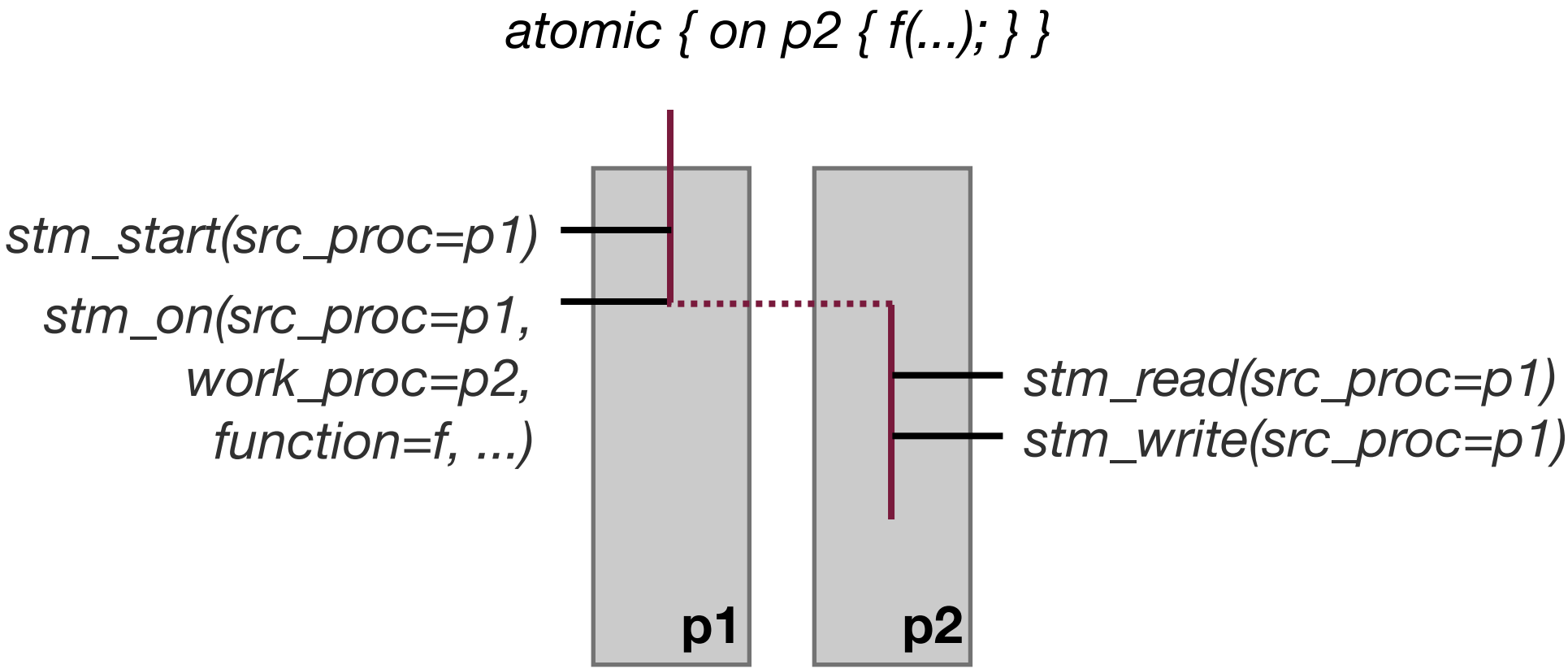
stm_on(src_proc=p1, work_proc=p2, function=f, ...)



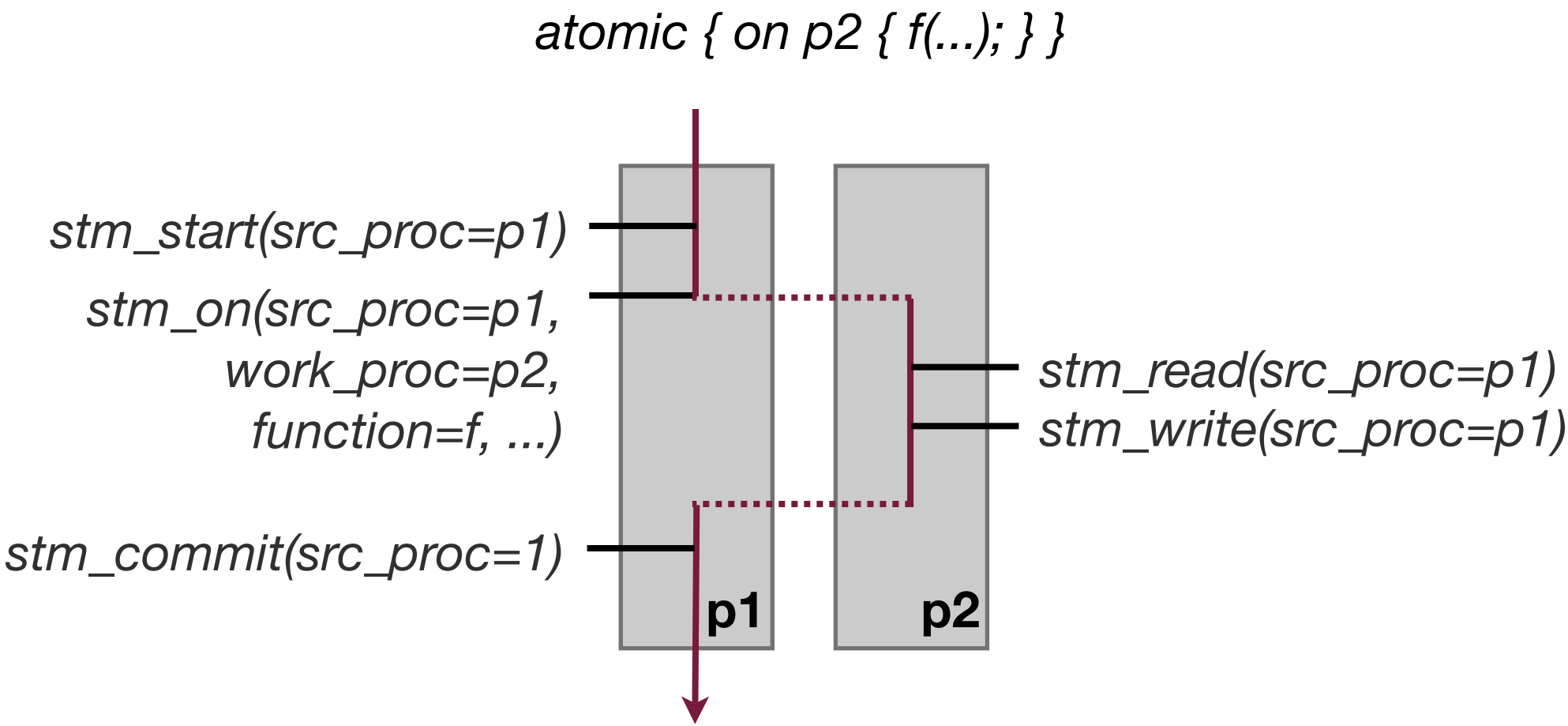
Remote Work Inside Transaction



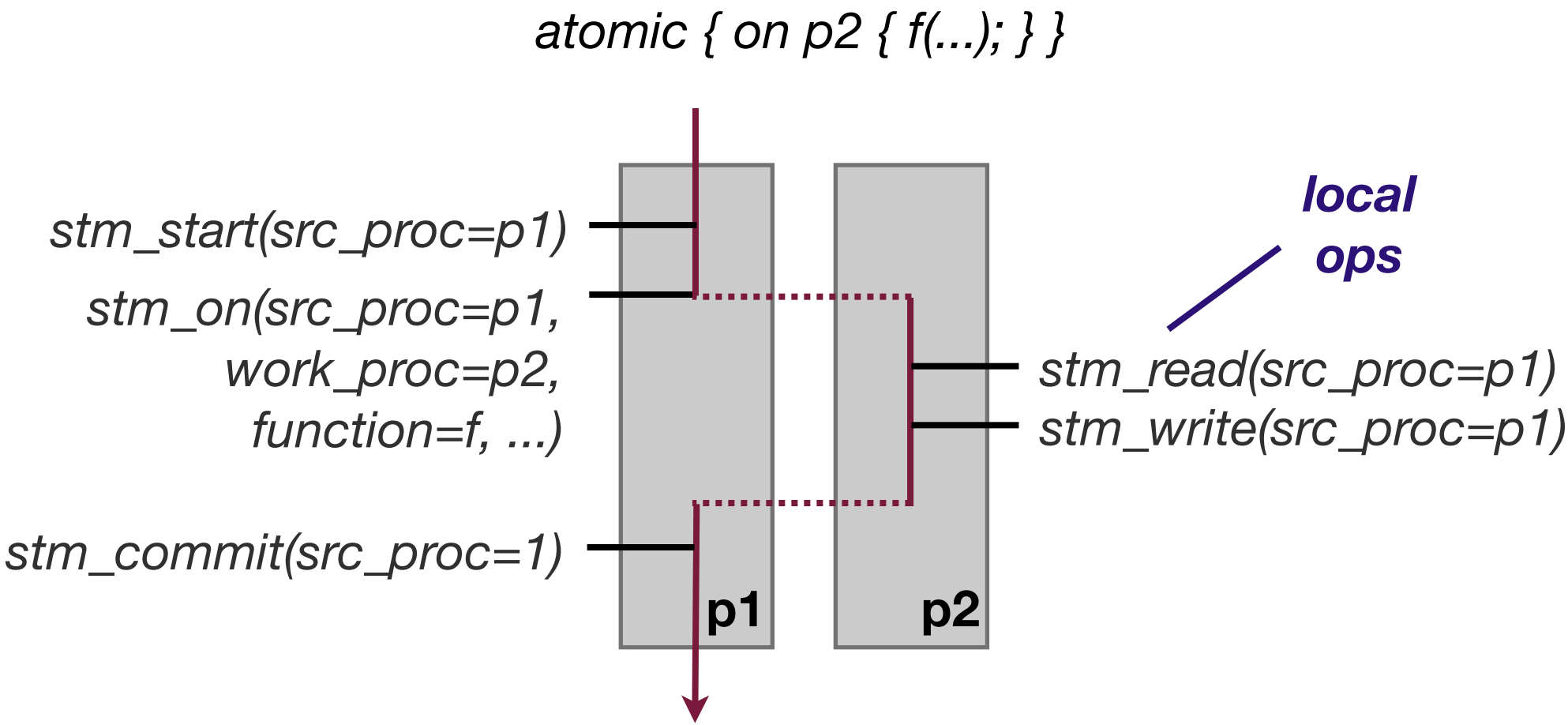
Remote Work Inside Transaction



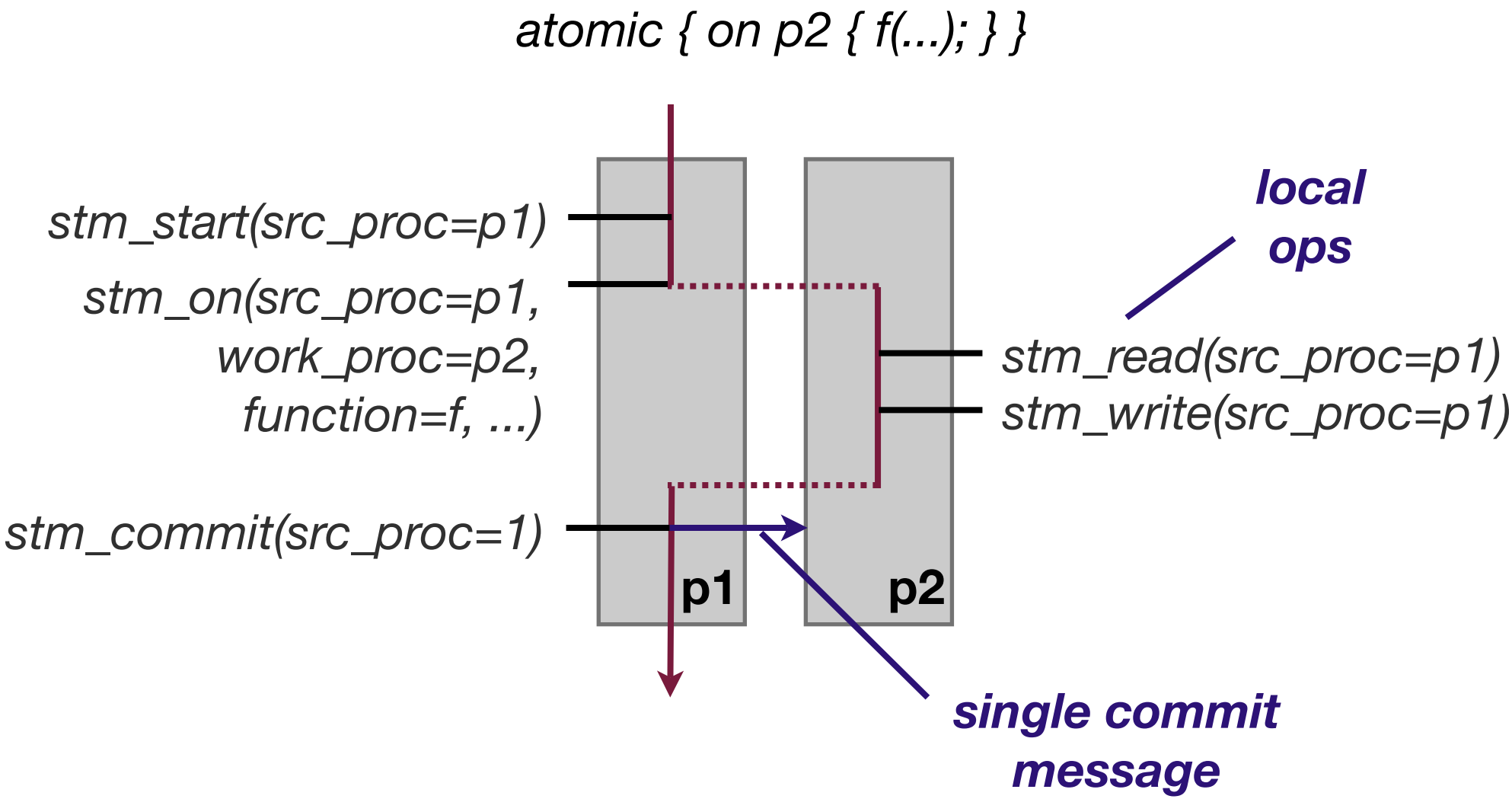
Remote Work Inside Transaction



Remote Work Inside Transaction

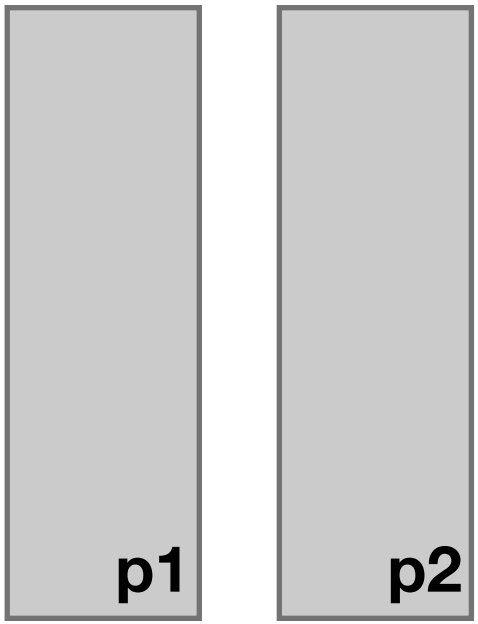


Remote Work Inside Transaction



Transaction Inside Remote Work

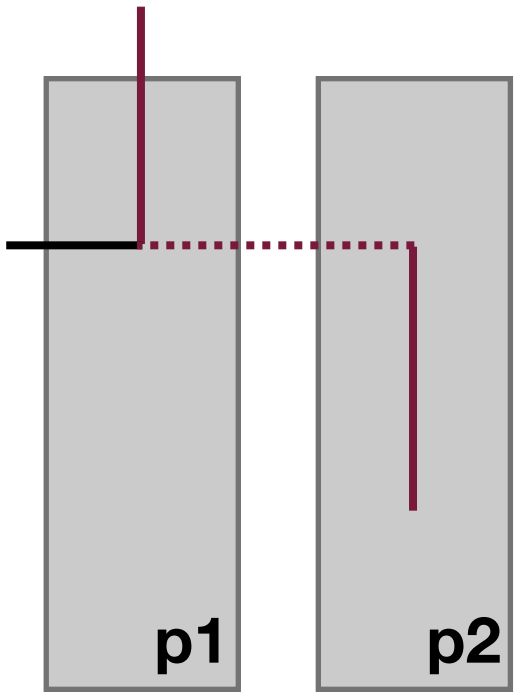
on p2 { atomic { f(...); } }



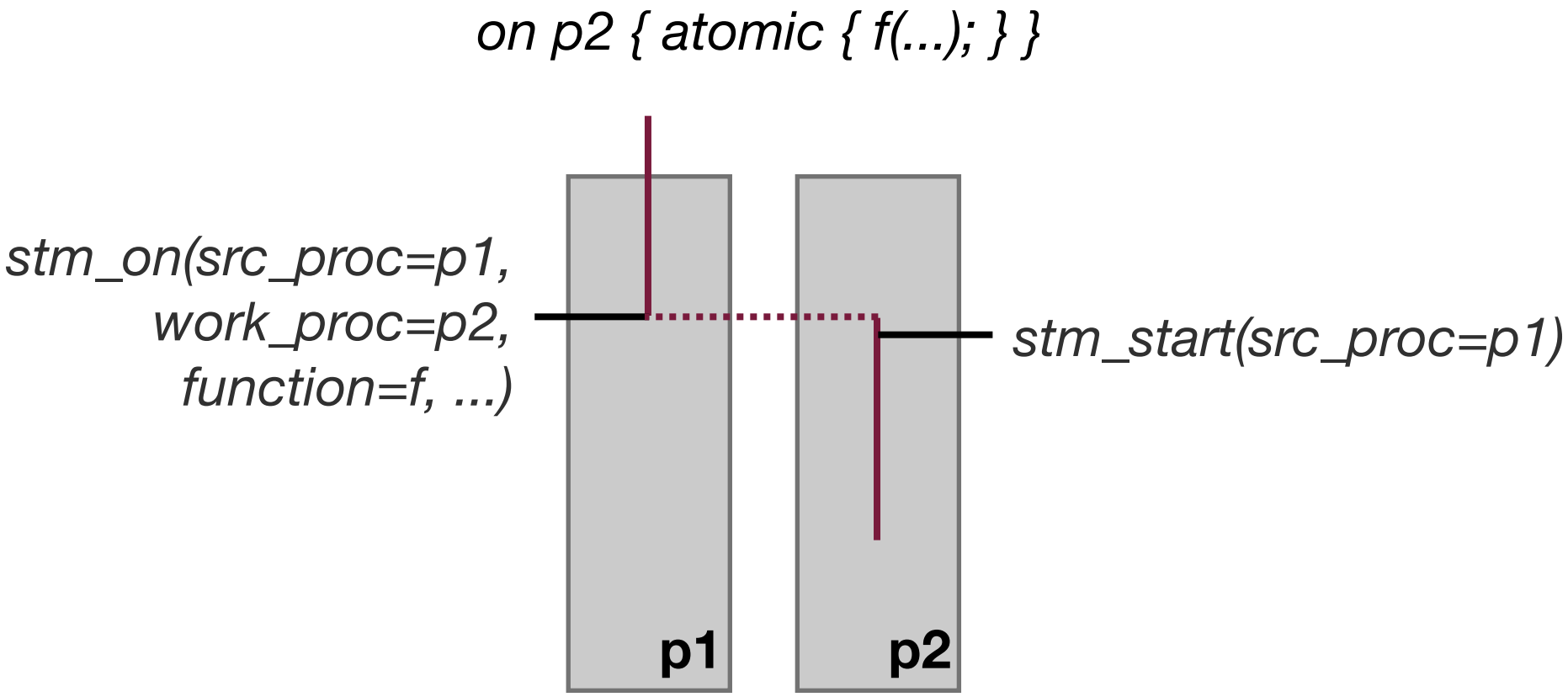
Transaction Inside Remote Work

on p2 { atomic { f(...); } }

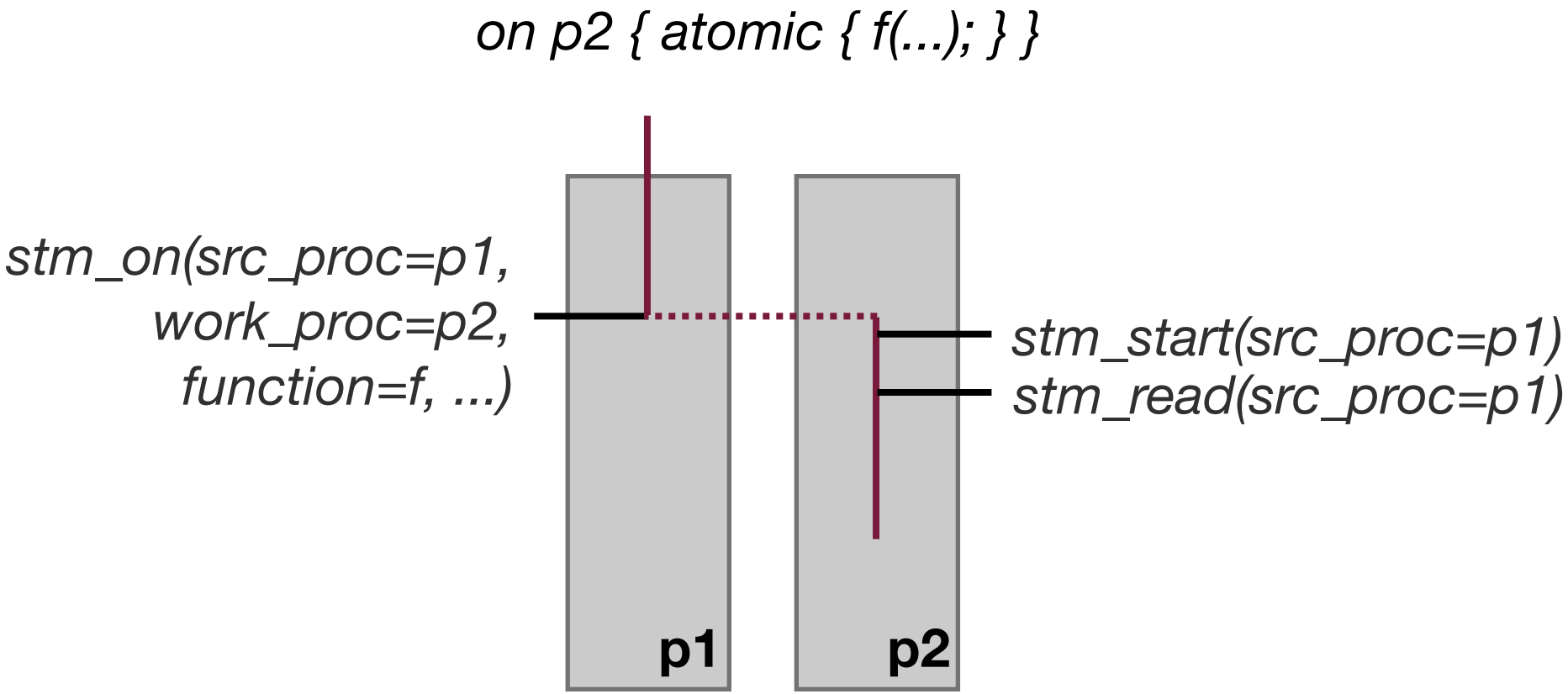
stm_on(src_proc=p1,
work_proc=p2,
function=f, ...)



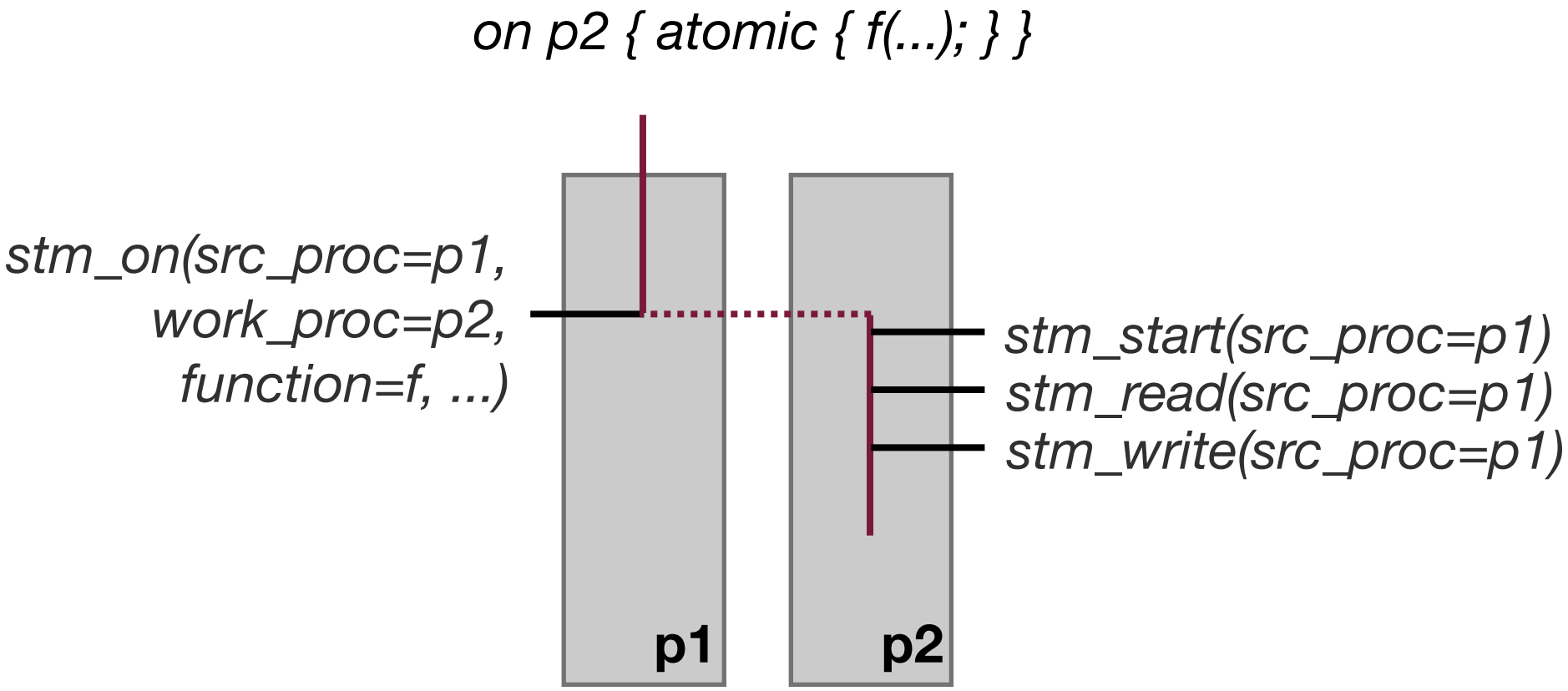
Transaction Inside Remote Work



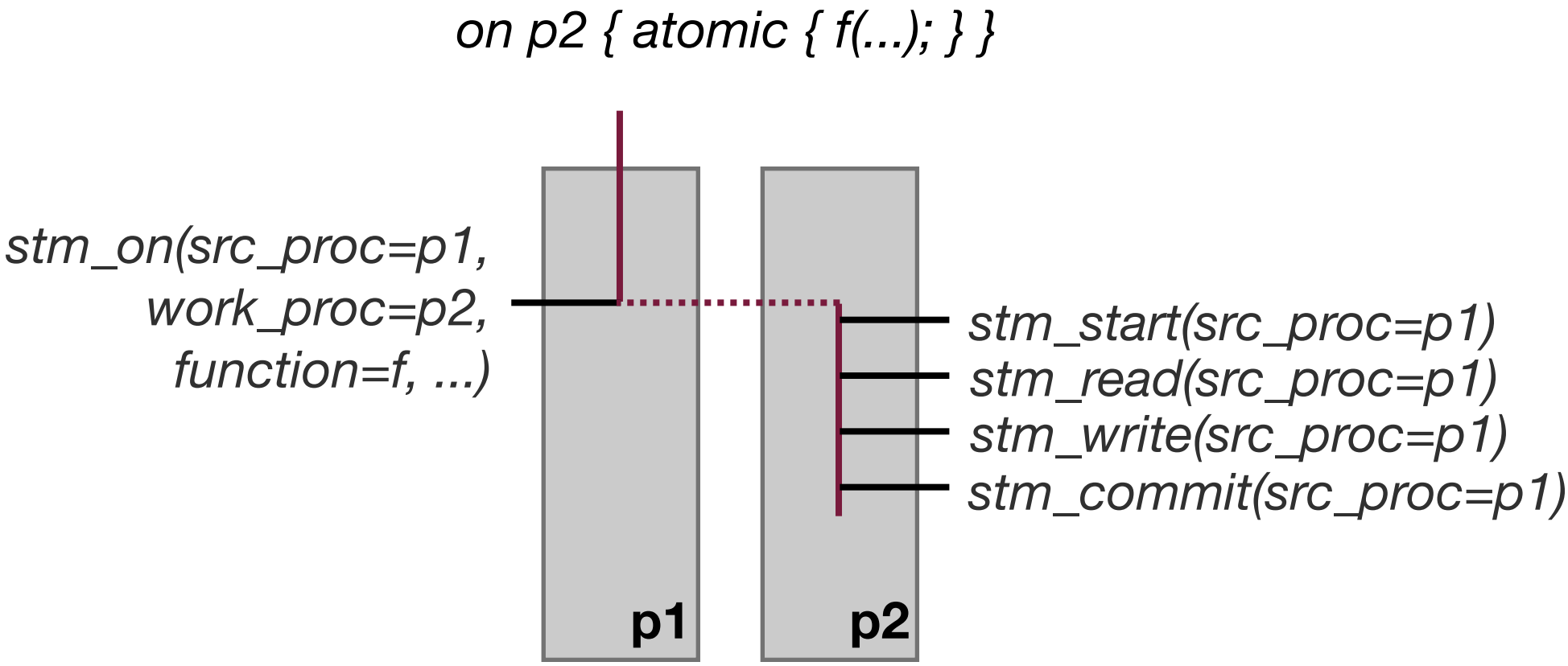
Transaction Inside Remote Work



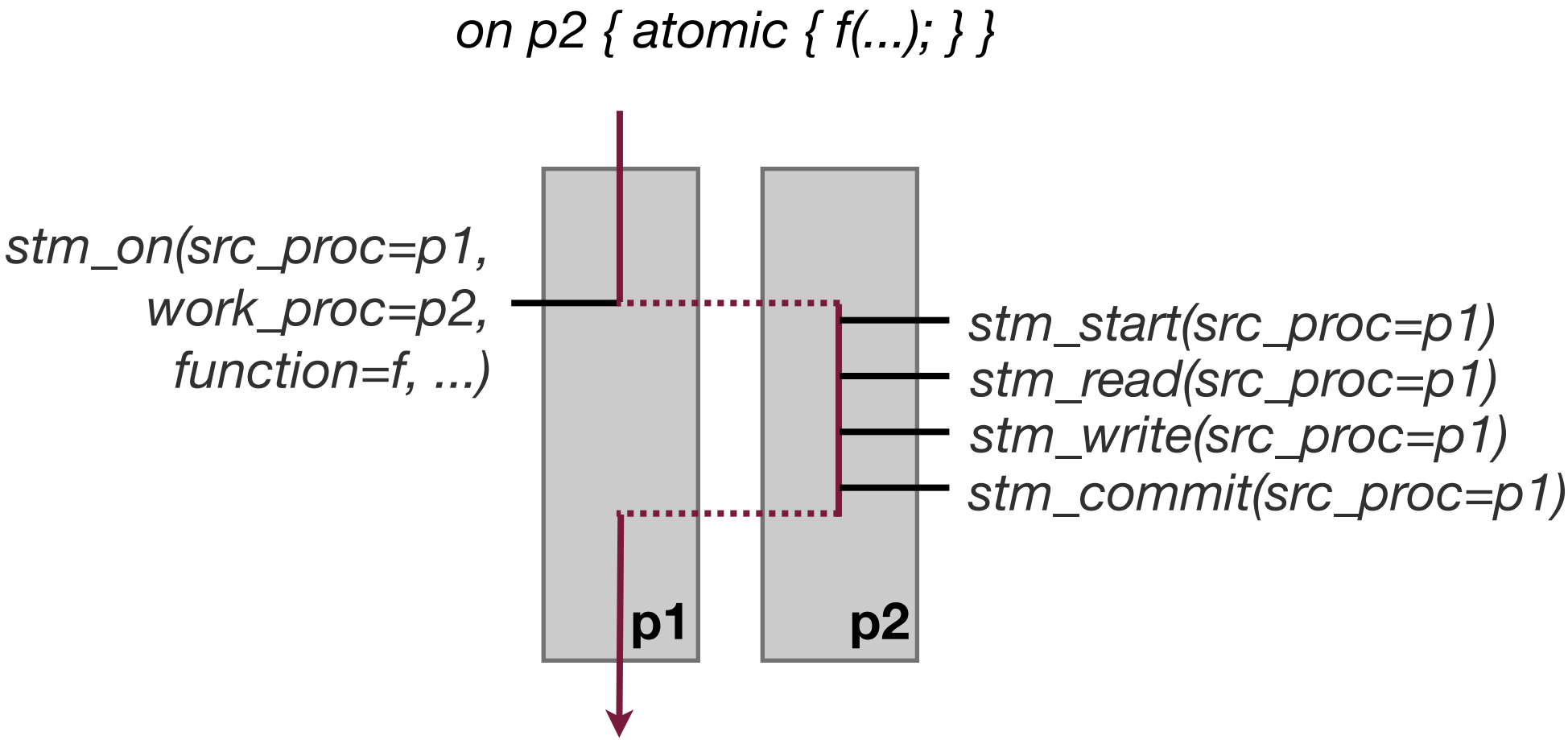
Transaction Inside Remote Work



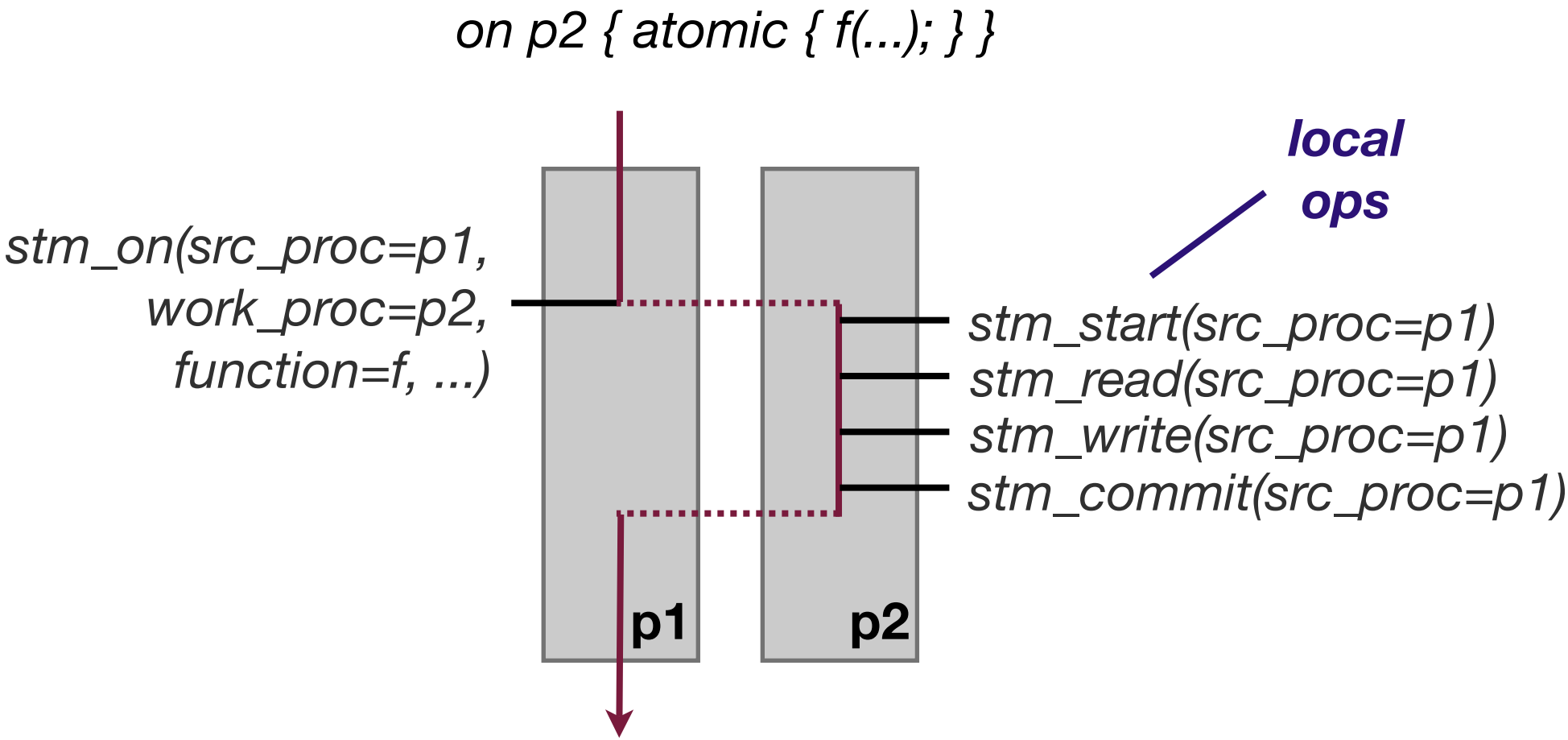
Transaction Inside Remote Work



Transaction Inside Remote Work



Transaction Inside Remote Work



Outline

Interface Design

Algorithm Design

Evaluation

- Cluster STM vs. Manual Locking
- Read Locks vs. Read Validation

Conclusion

Algorithm Design

Shared metadata

Transaction-local metadata

STM design choices

Shared Metadata

Some metadata must be visible to all transactions

- Read and write locks
- Validation ID

Memory overhead compromise

- Each metadata word guards one *conflict detection unit (CDU)*
- CDU size is $s \geq 1$ words
- $s > 1$ may introduce false sharing

Shared Metadata

Some metadata must be visible to all transactions

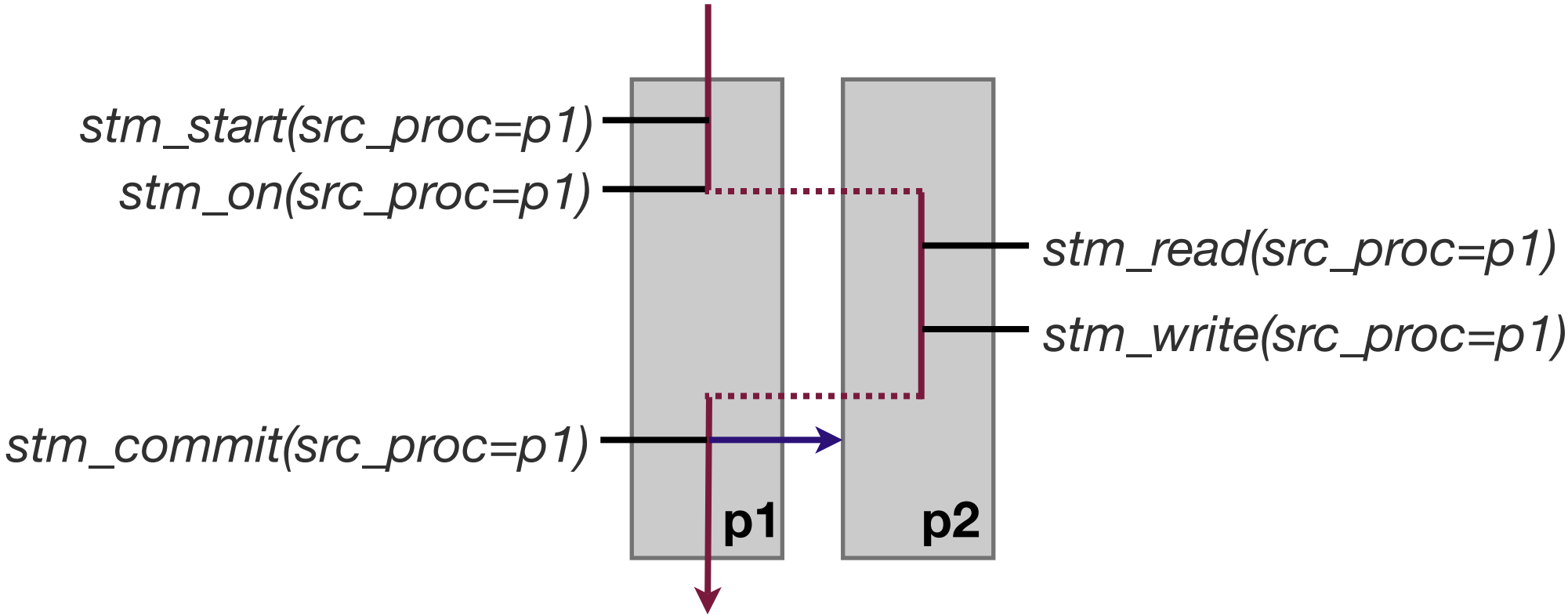
- Read and write locks
- Validation ID

Memory overhead compromise

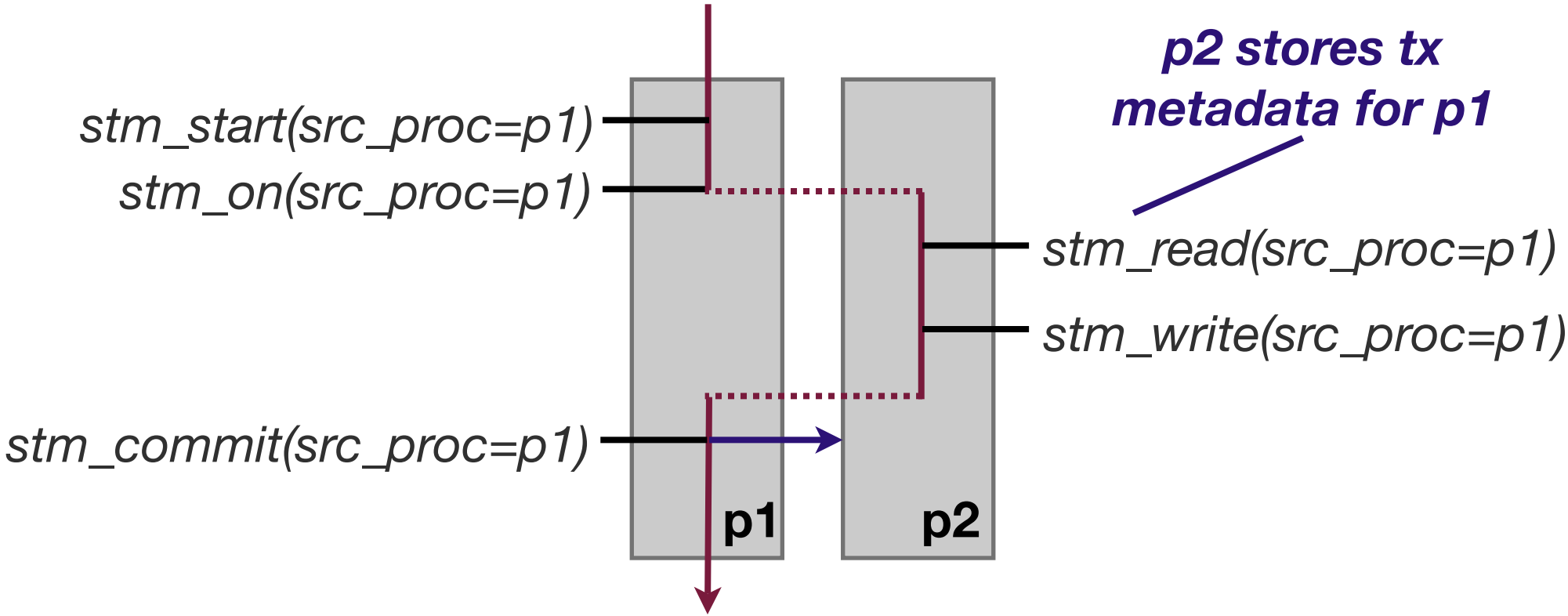
- Each metadata word guards one *conflict detection unit (CDU)*
- CDU size is $s \geq 1$ words
- $s > 1$ may introduce false sharing

Keep metadata on same processor as corresponding CDU

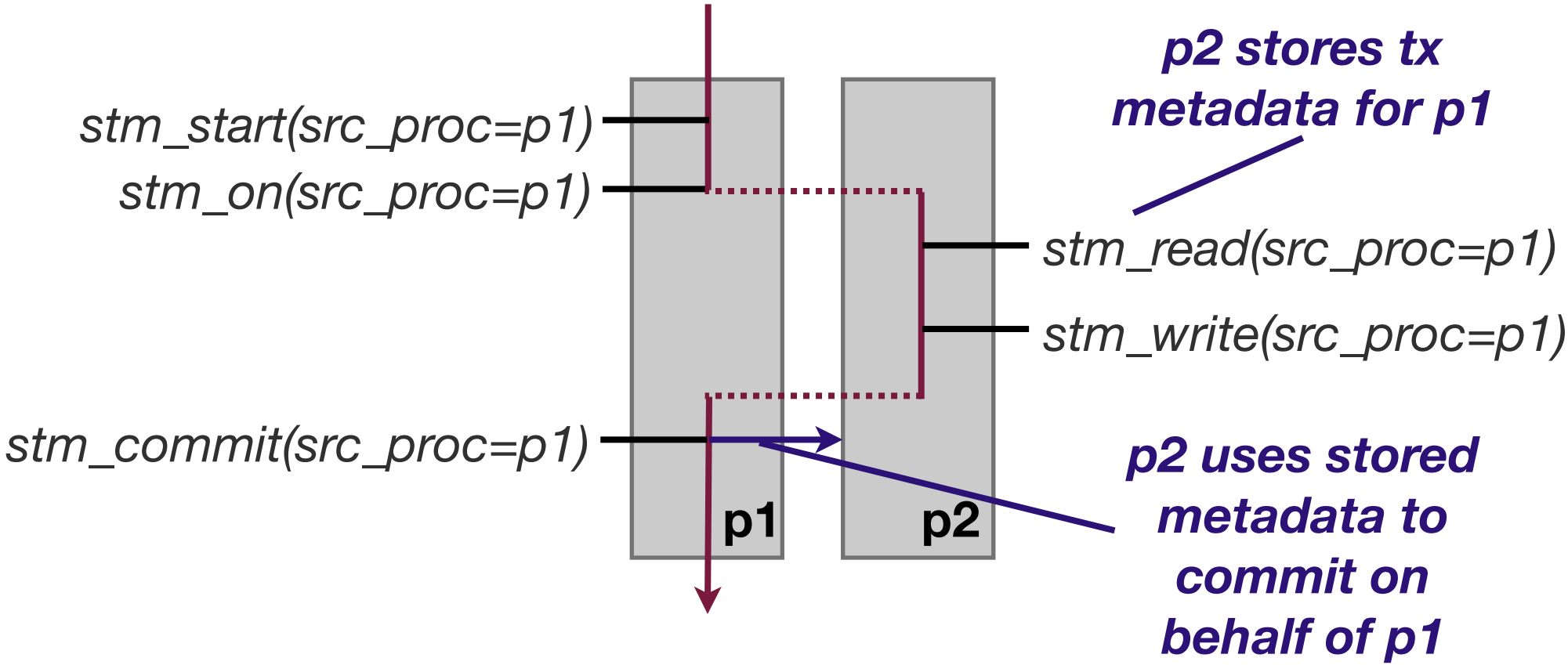
Transaction-Local Metadata



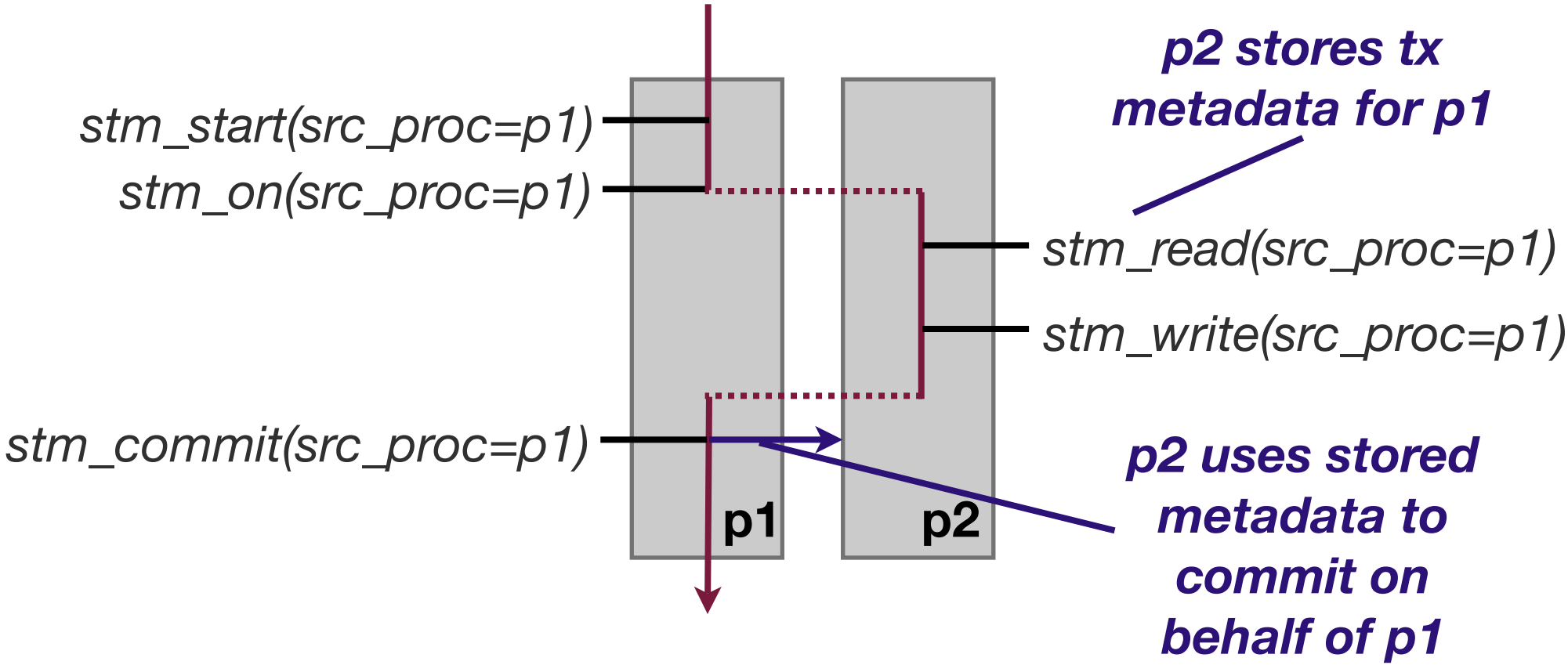
Transaction-Local Metadata



Transaction-Local Metadata



Transaction-Local Metadata



Keep metadata local to processor where access occurred

STM Design Choices

Eight-axis design space (see paper)

Choose four axes to explore

- CDU size
- Read locks (RL) vs. read validation (RV)
- Undo log (UL) vs. write buffer (WB)
- Early acquire (EA) vs. late acquire (LA)

STM Design Choices

Eight-axis design space (see paper)

Choose four axes to explore

- CDU size
- Read locks (RL) vs. read validation (RV)
- Undo log (UL) vs. write buffer (WB)
- Early acquire (EA) vs. late acquire (LA)

Some tradeoffs come out differently on clusters

STM Design Choices

Eight-axis design space (see paper)

Choose four axes to explore

- CDU size
- Read locks (RL) vs. read validation (RV)
- Undo log (UL) vs. write buffer (WB)
- Early acquire (EA) vs. late acquire (LA)

Some tradeoffs come out differently on clusters

Discuss RL vs. RV here

STM Design Choices

Eight-axis design space (see paper)

Choose four axes to explore

- CDU size
- Read locks (RL) vs. read validation (RV)
- Undo log (UL) vs. write buffer (WB)
- Early acquire (EA) vs. late acquire (LA)

Some tradeoffs come out differently on clusters

Discuss RL vs. RV here

See paper for additional results

Outline

Interface Design

Algorithm Design

Evaluation

- Cluster STM vs. Manual Locking
- Read Locks vs. Read Validation

Conclusion

Evaluation

Benchmarks

- Micro Benchmarks: Intset, Hashmap Swap
- Graph clustering: SSCA 2 Kernel 4

Machine

- Intel Xeon cluster
- Two cores per node
- Myrinet

Outline

Interface Design

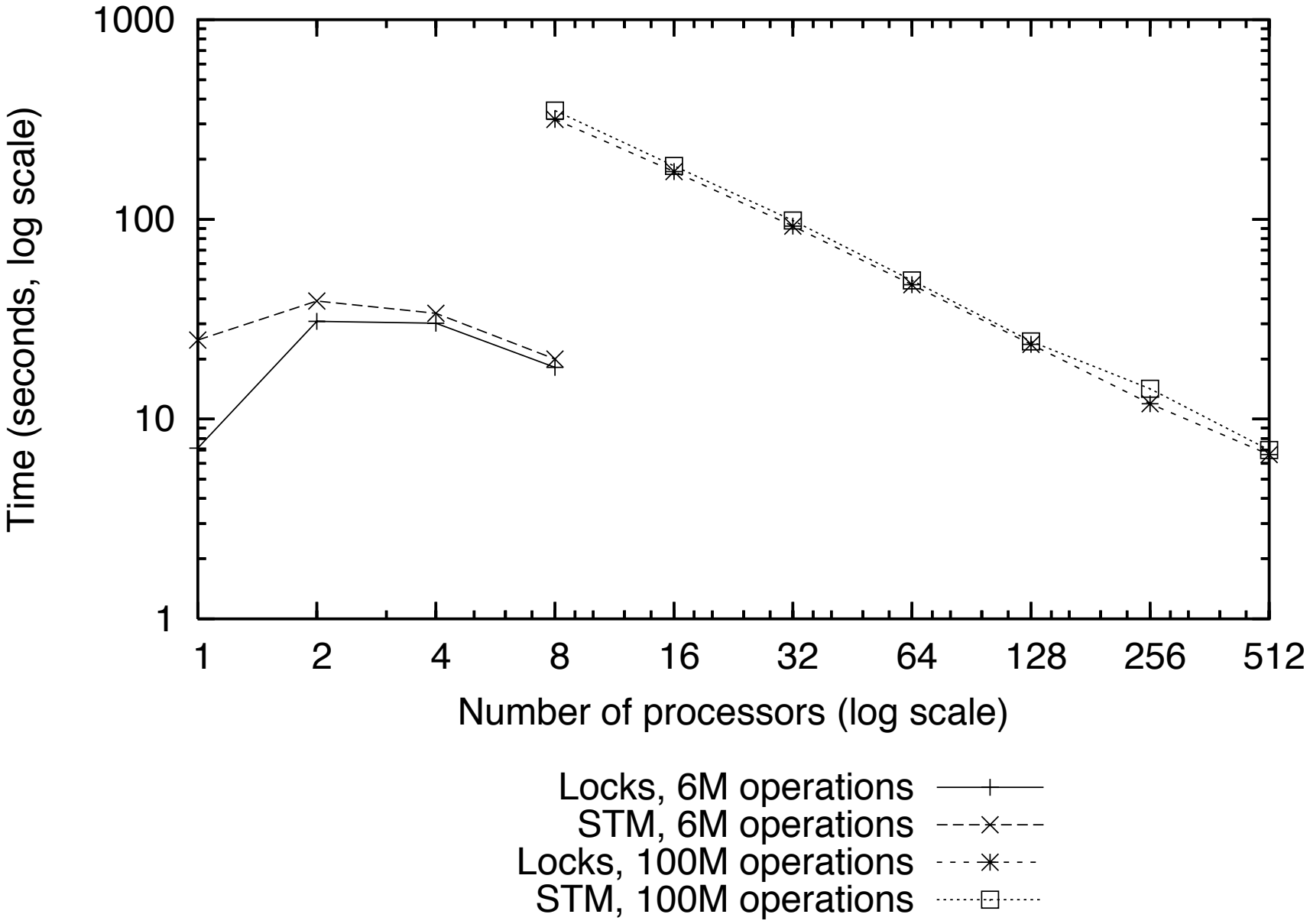
Algorithm Design

Evaluation

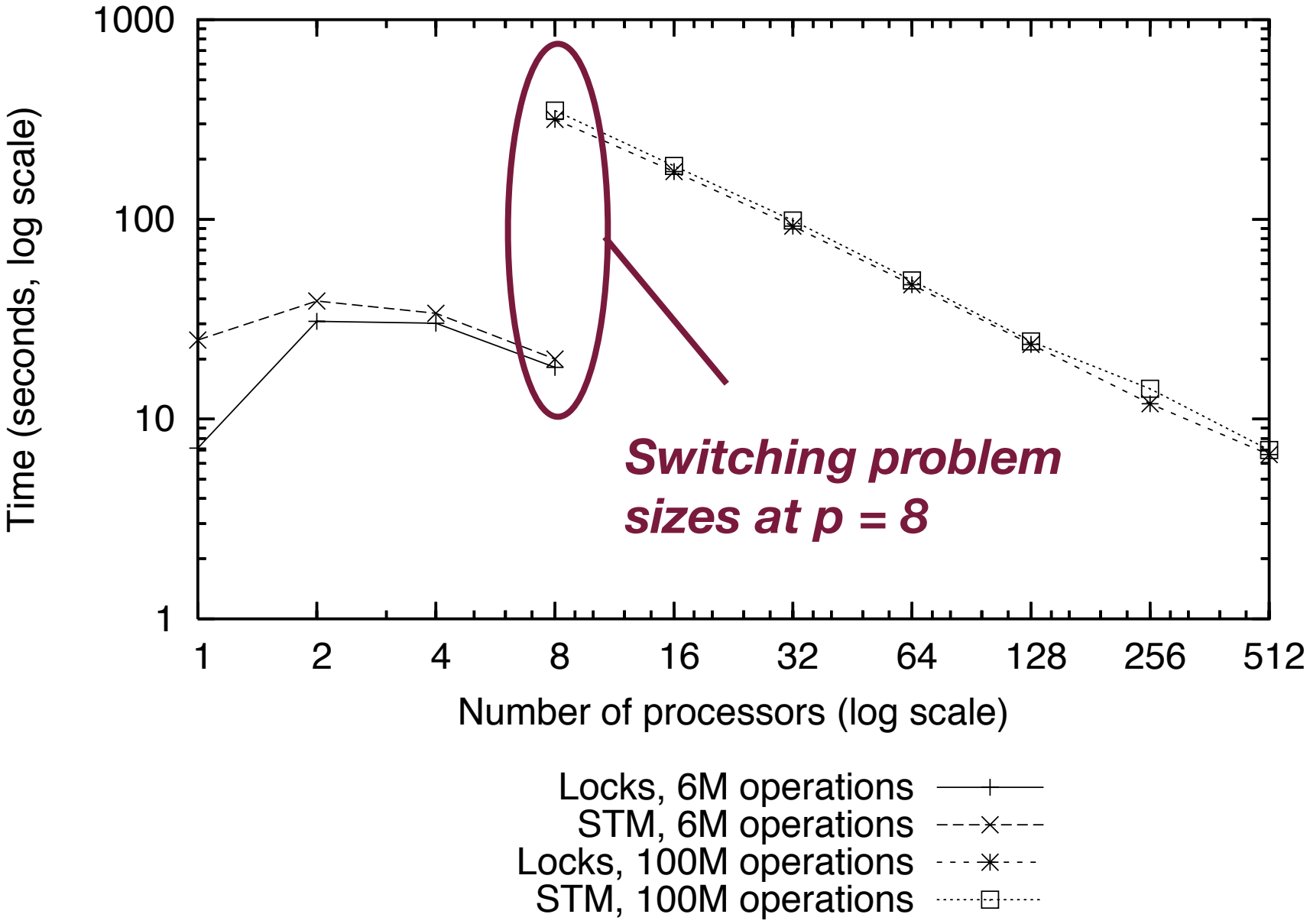
- **Cluster STM vs. Manual Locking**
- Read Locks vs. Read Validation

Conclusion

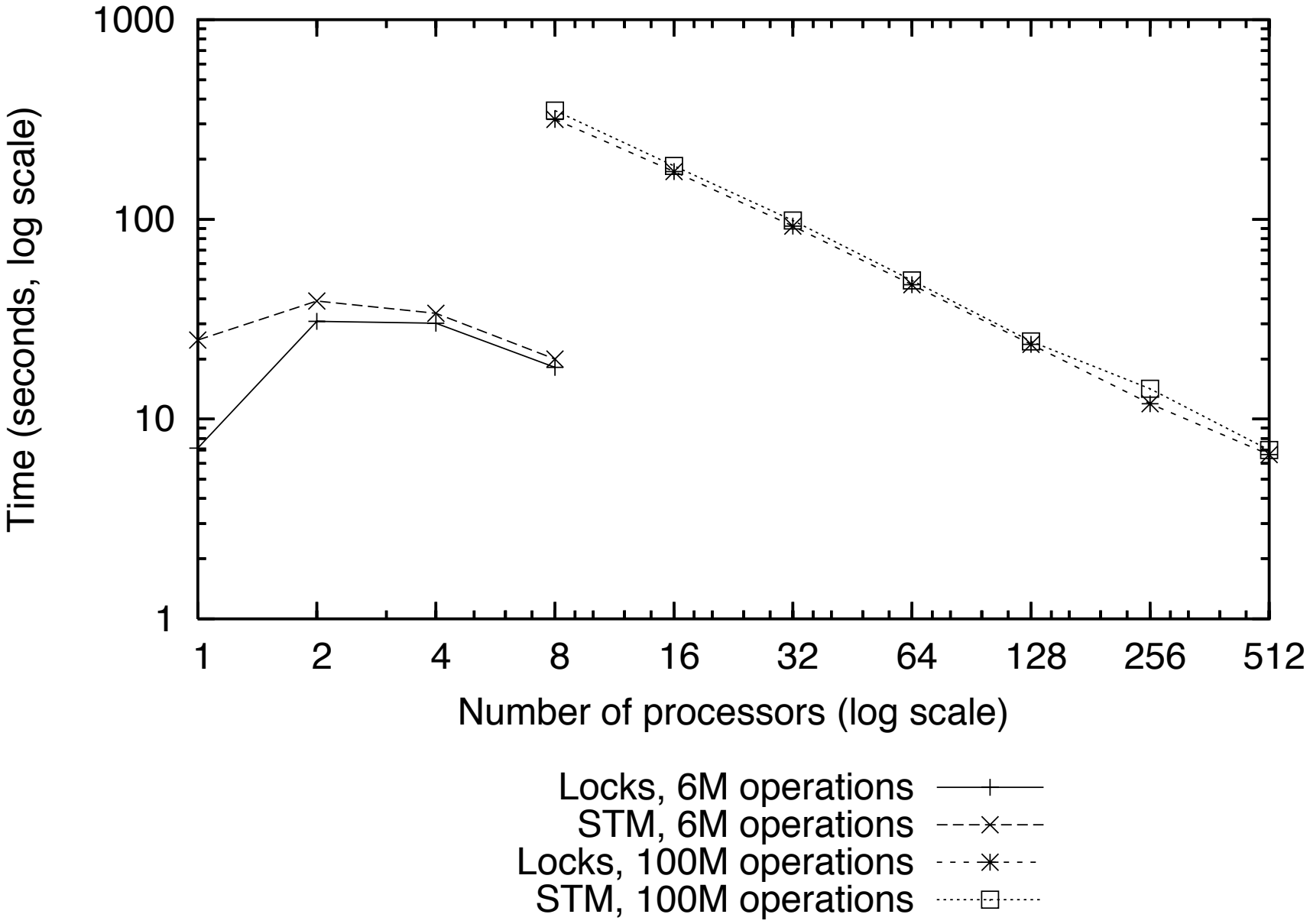
Intset Results



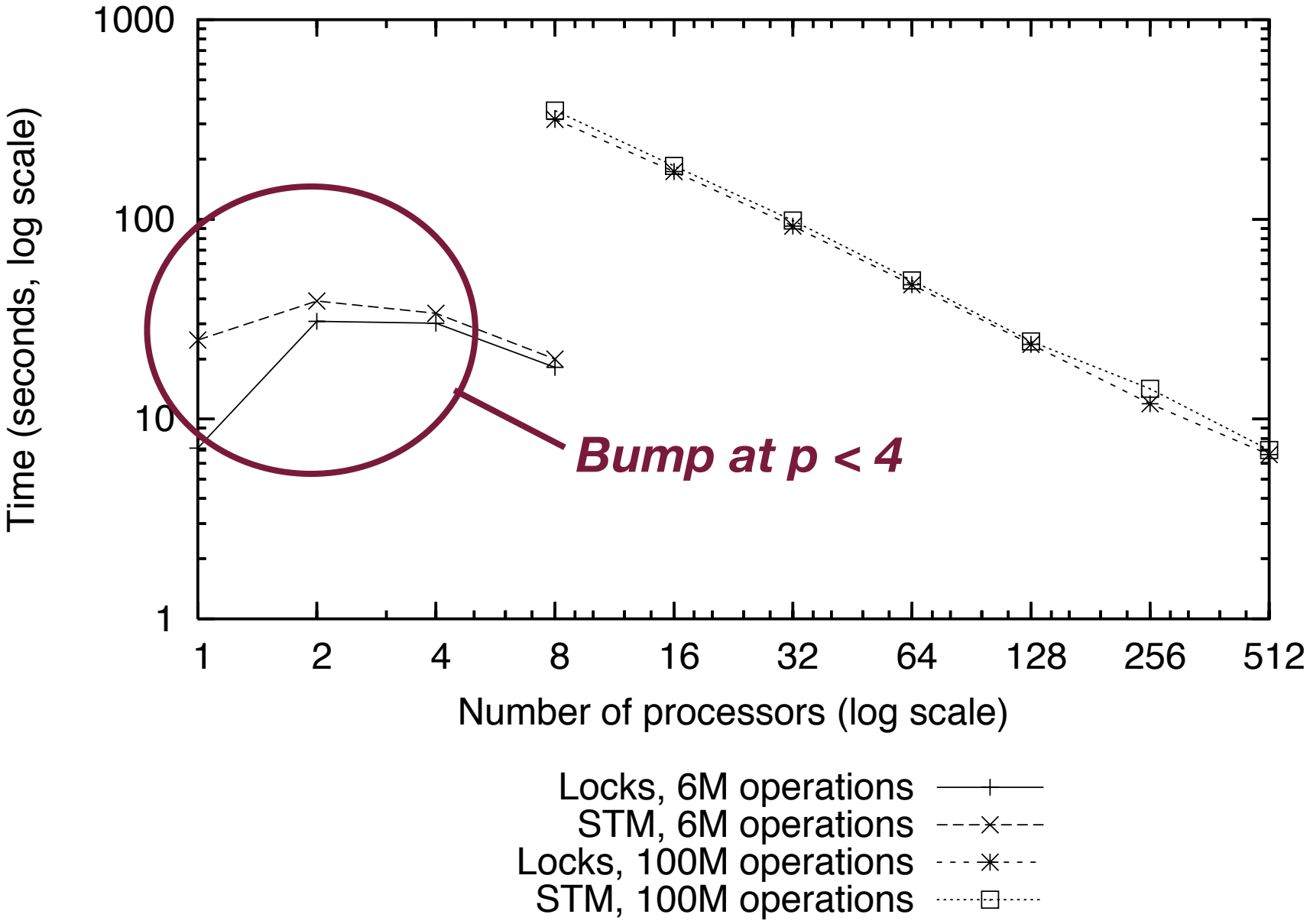
Intset Results



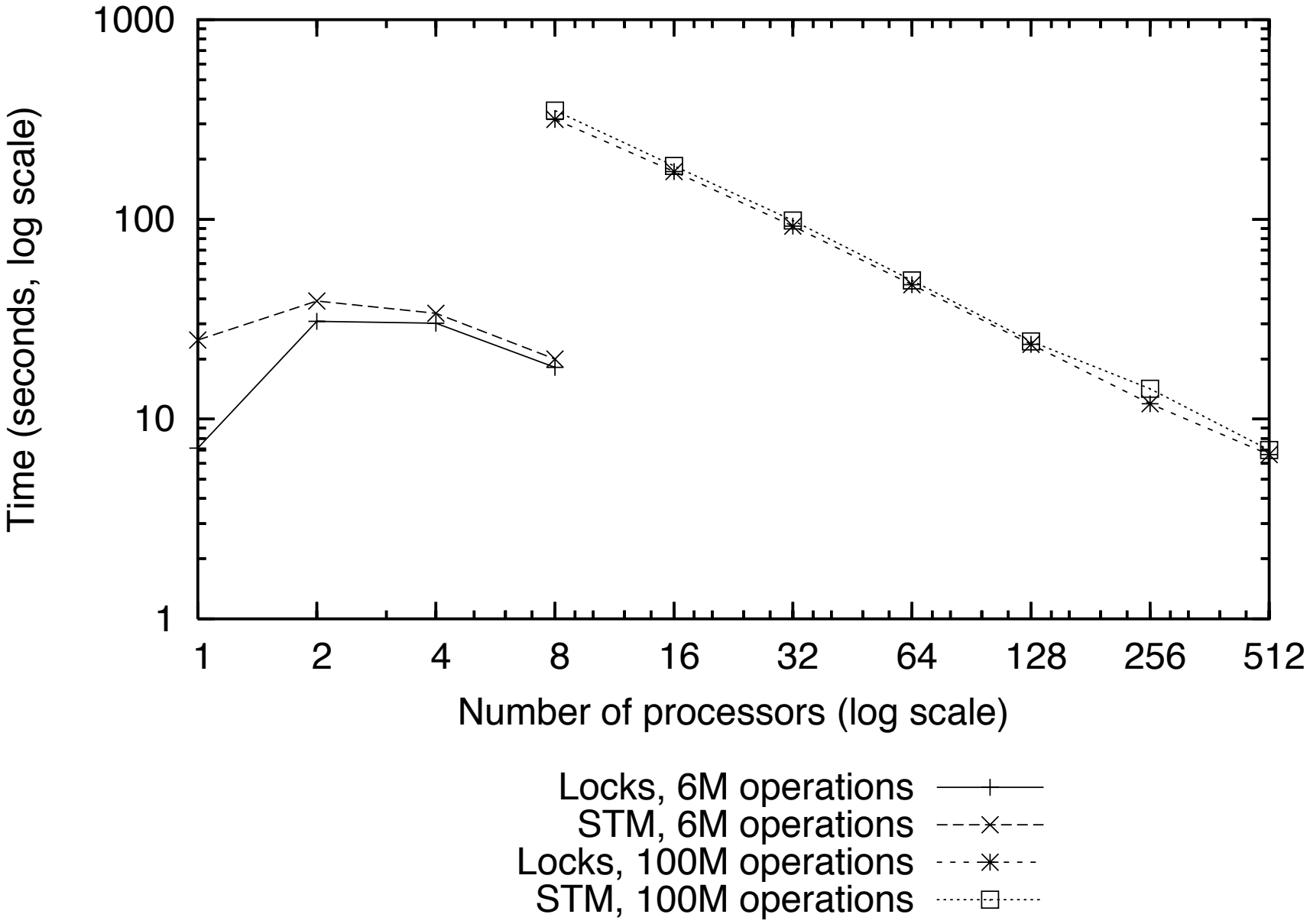
Intset Results



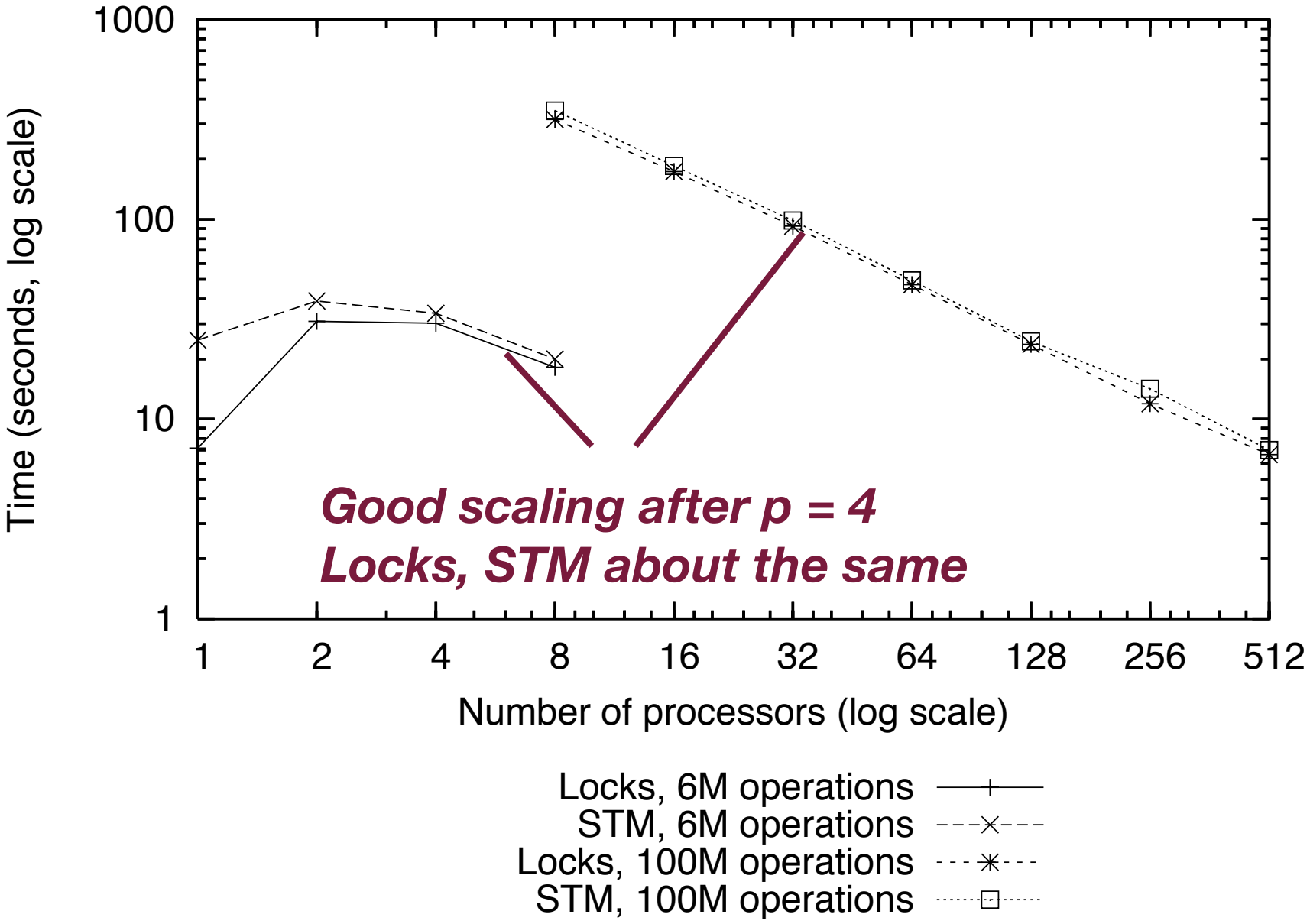
Intset Results



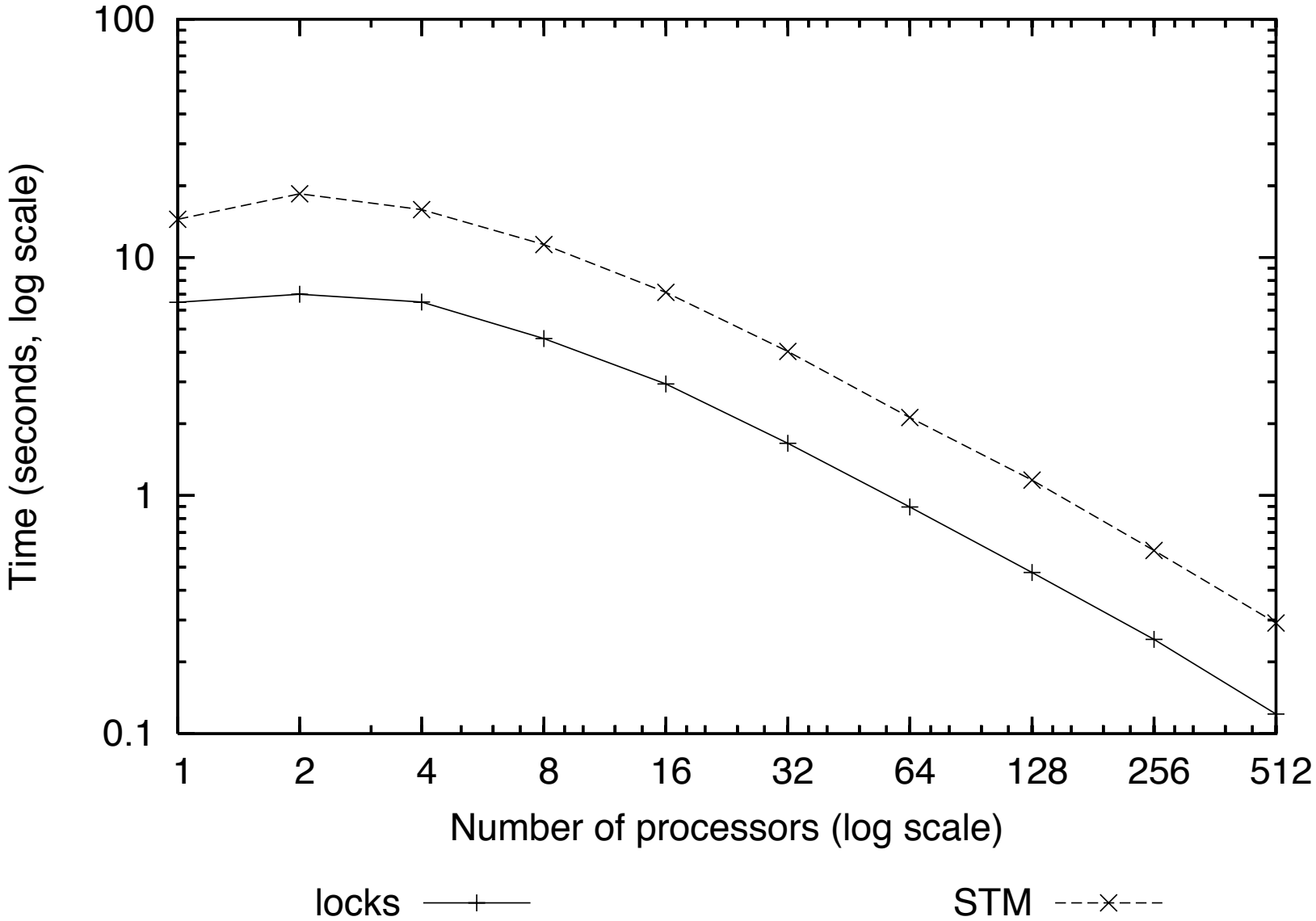
Intset Results



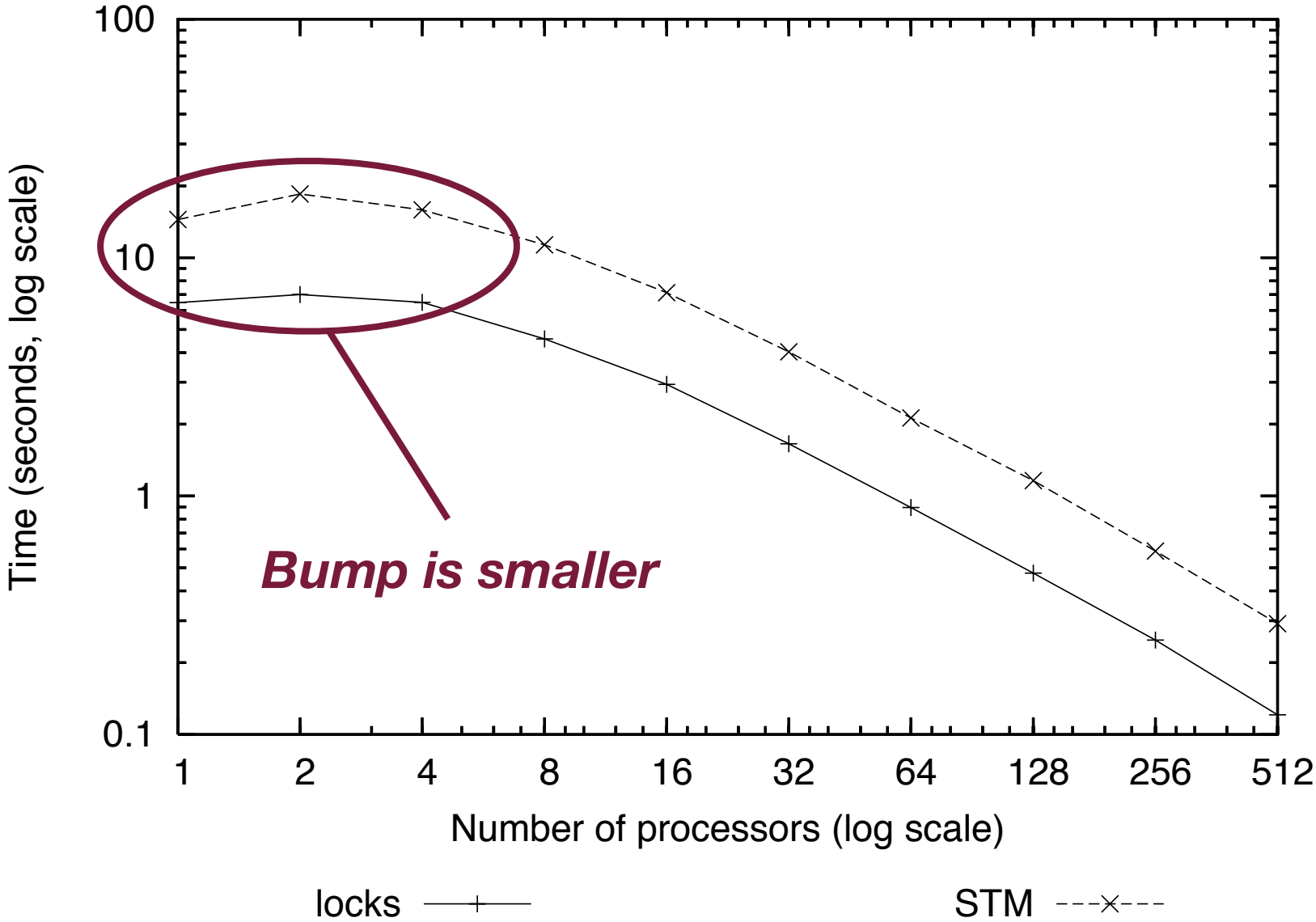
Intset Results



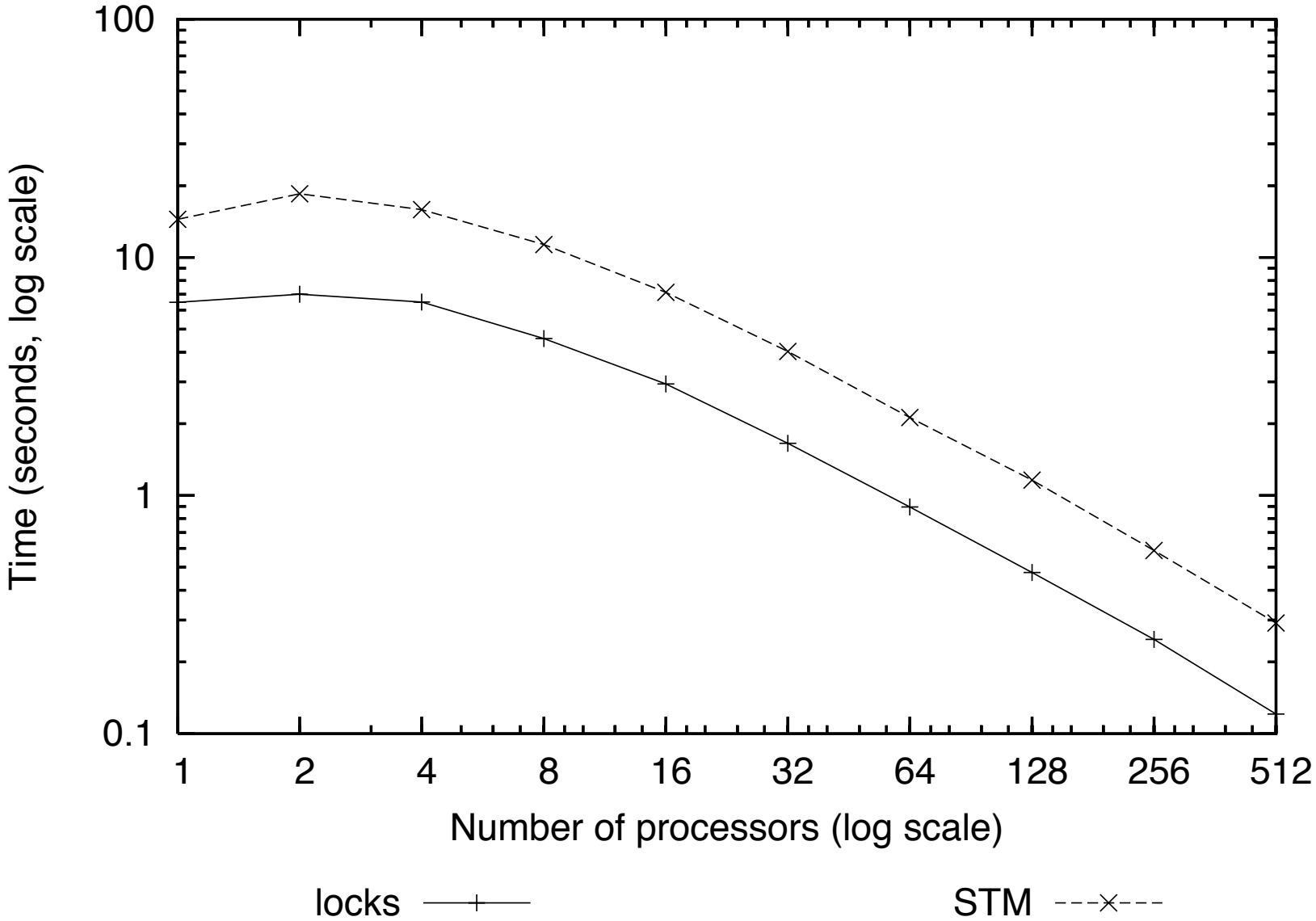
SSCA 2 Results



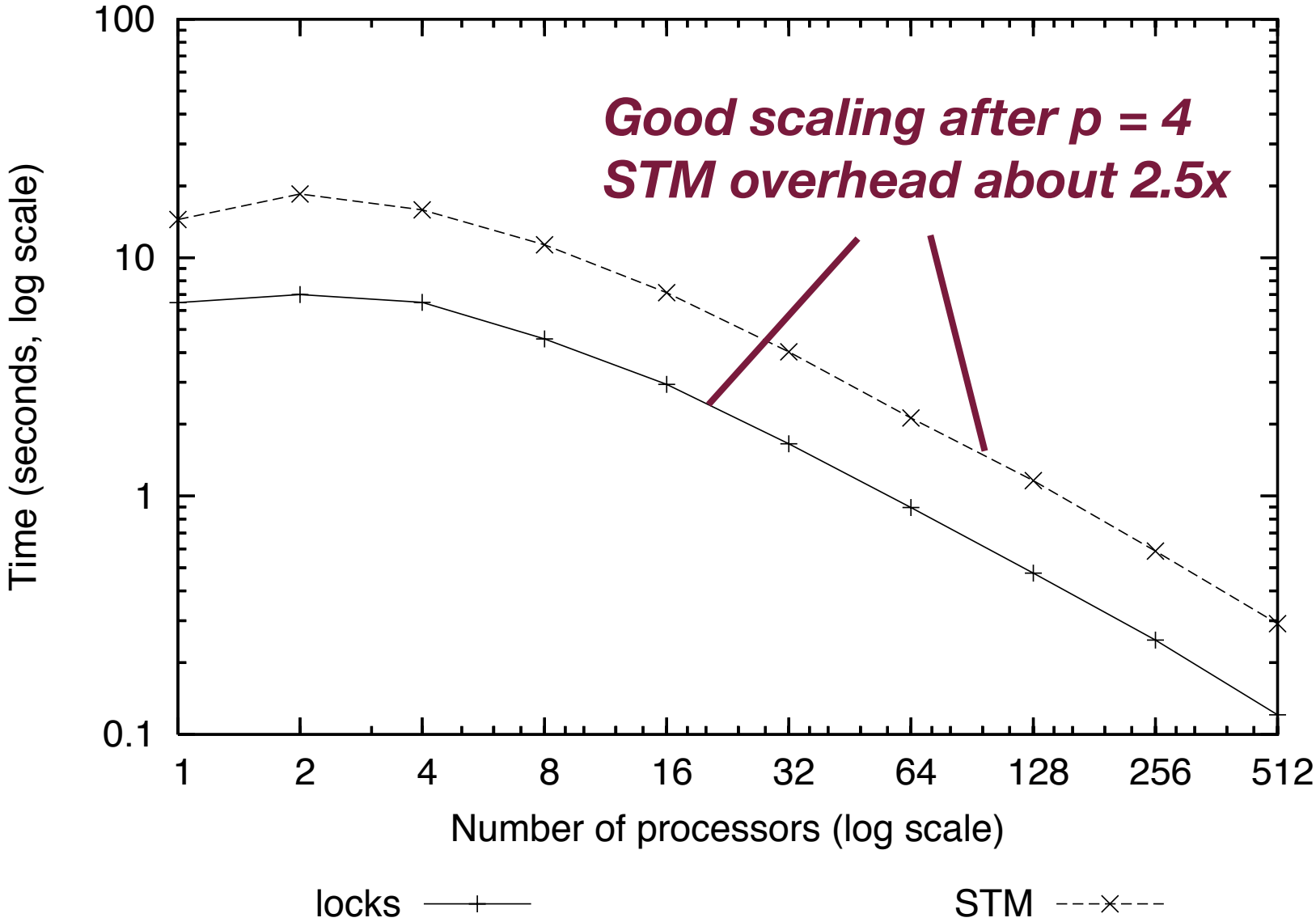
SSCA 2 Results



SSCA 2 Results



SSCA 2 Results



Outline

Interface Design

Algorithm Design

Evaluation

- Cluster STM vs. Manual Locking
- **Read Locks vs. Read Validation**

Conclusion

Implementation

Read locks (**RL**)

- Metadata word holds *write bit* and *reader count*
- Abort on attempt to read or write conflicting CDU

Read validation (**RV**)

- Metadata word holds *write bit* and *validation ID*
- Increment ID with each write
- Abort at commit if ID changes after CDU is read

Implementation

Read locks (**RL**)

- Metadata word holds *write bit* and *reader count*
- Abort on attempt to read or write conflicting CDU

Read validation (**RV**)

- Metadata word holds *write bit* and *validation ID*
- Increment ID with each write
- Abort at commit if ID changes after CDU is read

Validation requires extra remote op at commit

Implementation

Read locks (**RL**)

- Metadata word holds *write bit* and *reader count*
- Abort on attempt to read or write conflicting CDU

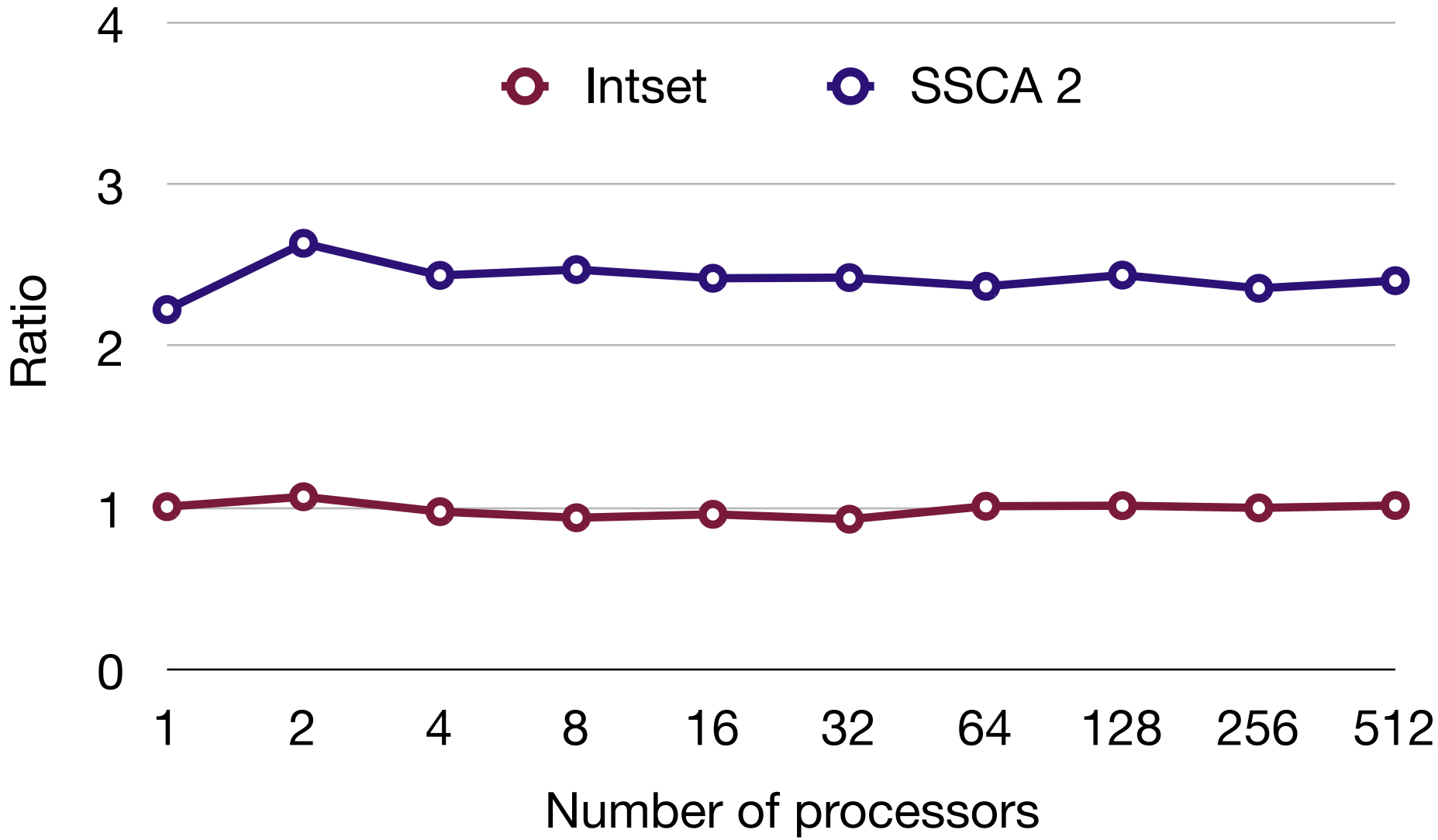
Read validation (**RV**)

- Metadata word holds *write bit* and *validation ID*
- Increment ID with each write
- Abort at commit if ID changes after CDU is read

Validation requires extra remote op at commit

No global cache ⇒ no helpful cache effects

Ratio of RV to RL Runtimes



Results Summary

Good scalability to $p = 512$

Good performance

- Nearly identical to locks on micro benchmarks
- Some STM overhead for SSCA2

Different design tradeoffs

- RL outperforms RV
- EA outperforms LA (see paper)
- Minimal penalty for WB (see paper)

Outline

Interface Design

Algorithm Design

Evaluation

Conclusion

Conclusion

Presented design and evaluation of Cluster STM

- First STM for high performance on large scale clusters
- Good performance, scaling to $p = 512$
- New evaluation of design tradeoffs

Future work

- Exploiting shared memory within a node
- Nested parallelism in a transaction
- Dynamic spawning of threads

Thank You