Transactional IPC in L4/Fiasco.OC

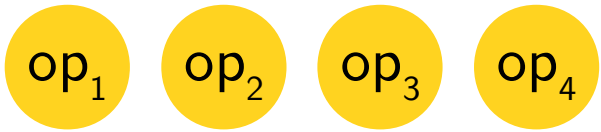
Can we get the multicore case verified for free?

Till Smejkal, Adam Lackorzynski, Benjamin Engel and Marcus Völp

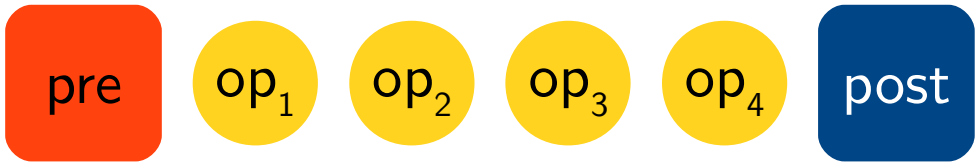
Operating Systems Group
Technische Universität Dresden
Germany

July 7, 2015

Simplifying the Multicore Verification

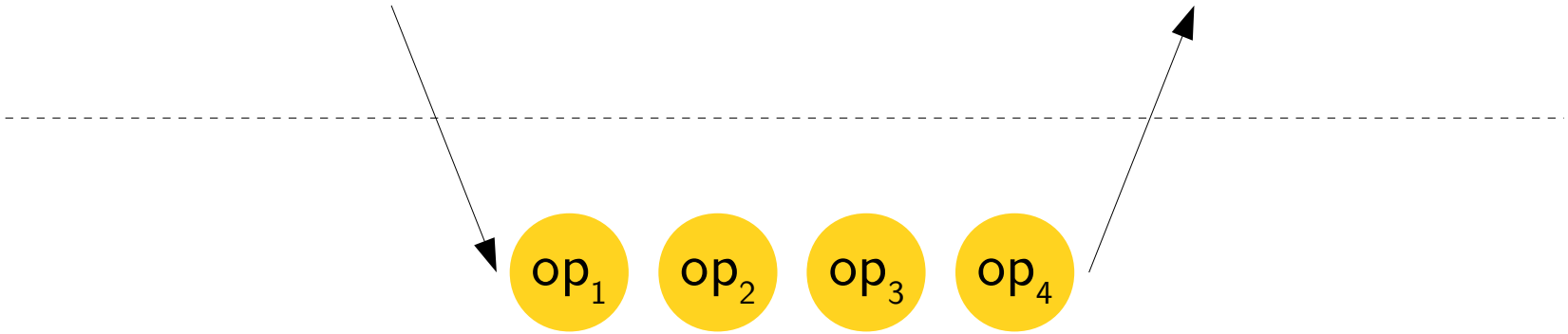


Simplifying the Multicore Verification



Simplifying the Multicore Verification

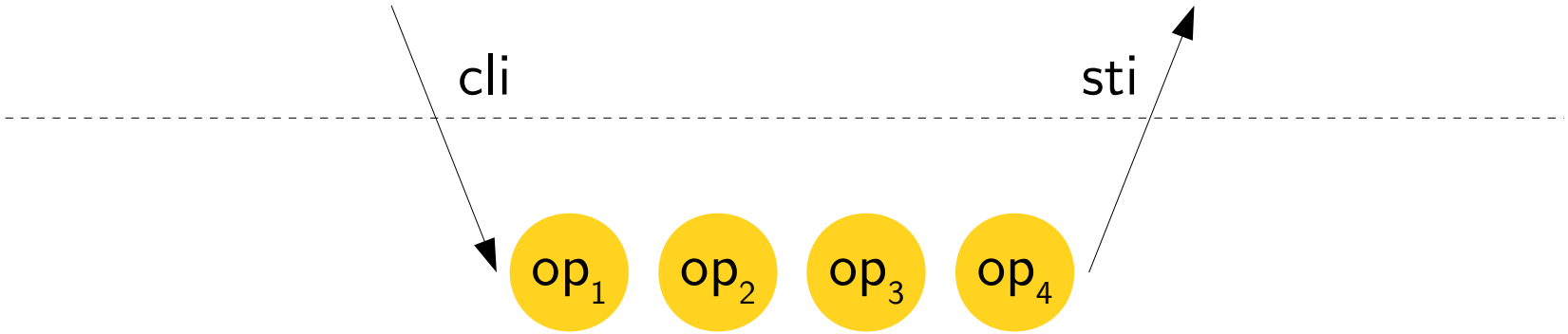
user



kernel

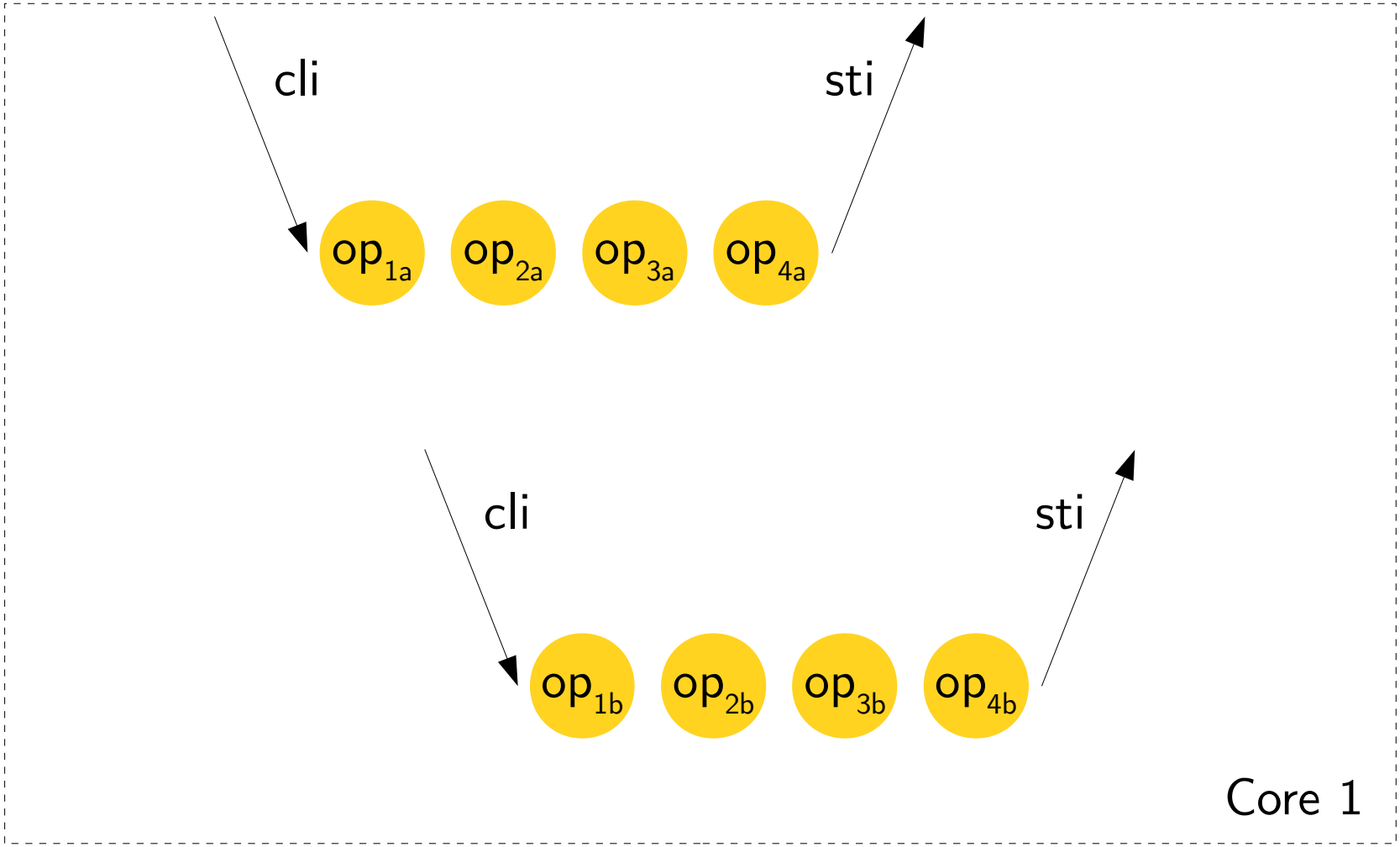
Simplifying the Multicore Verification

user

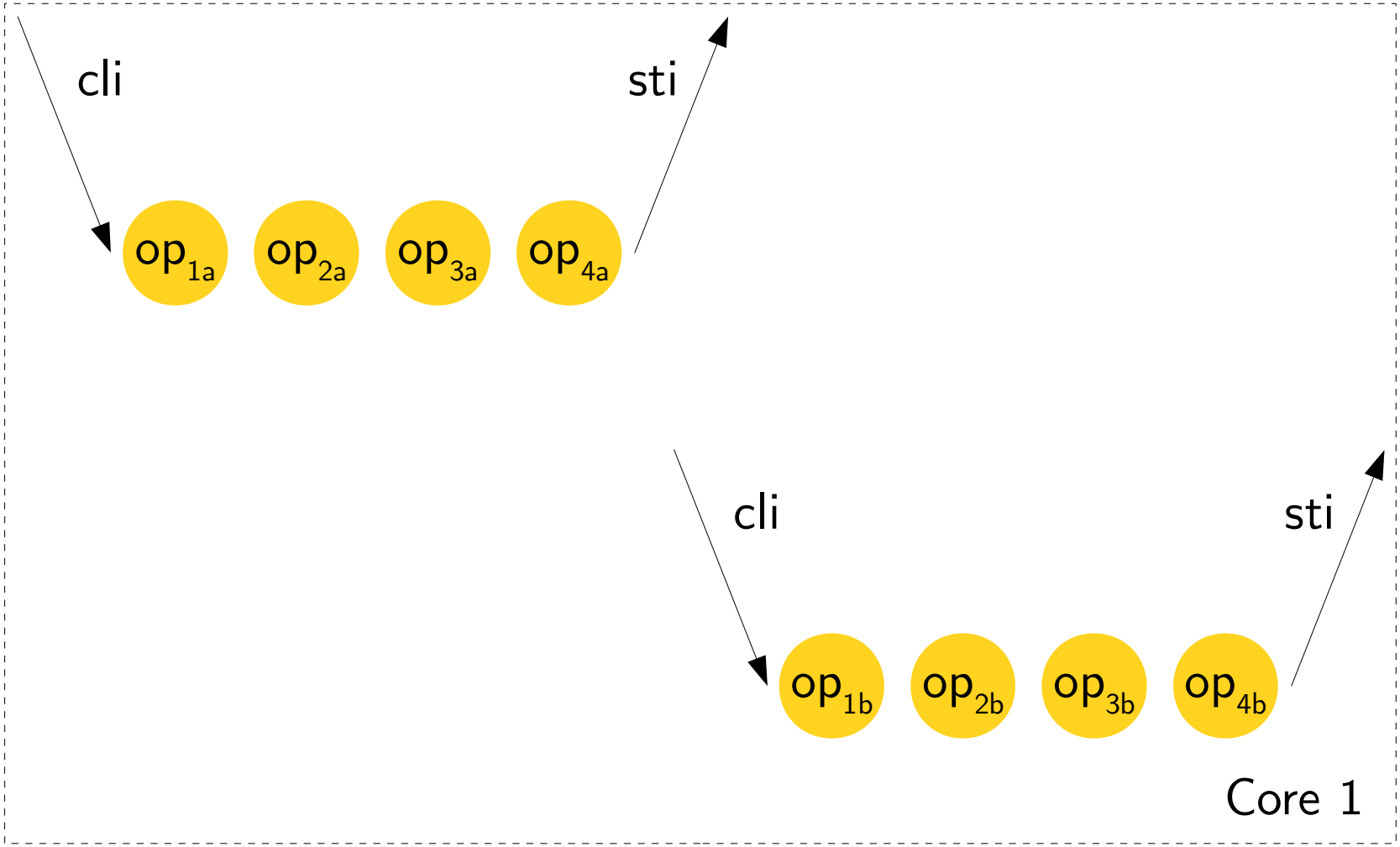


kernel

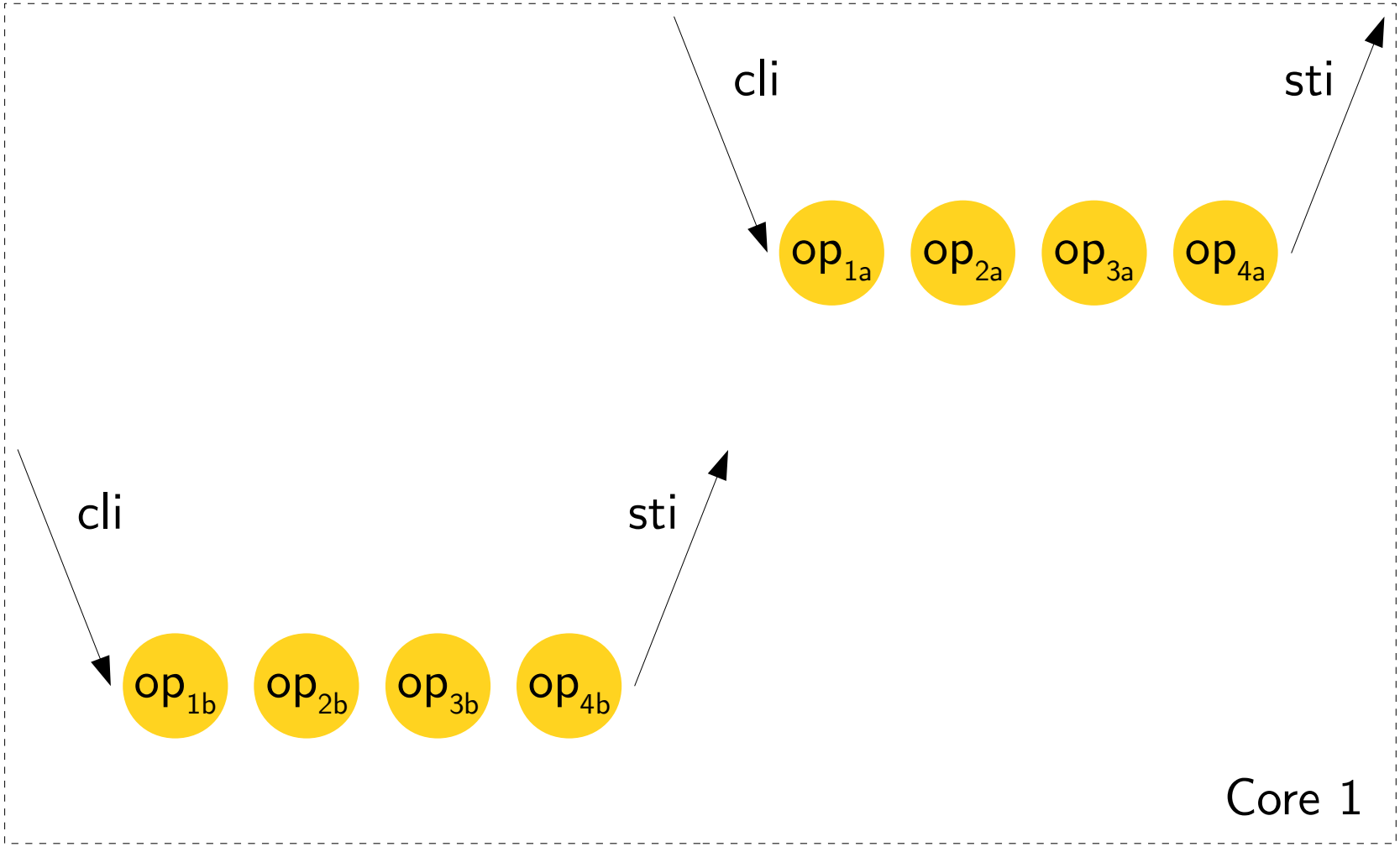
Simplifying the Multicore Verification



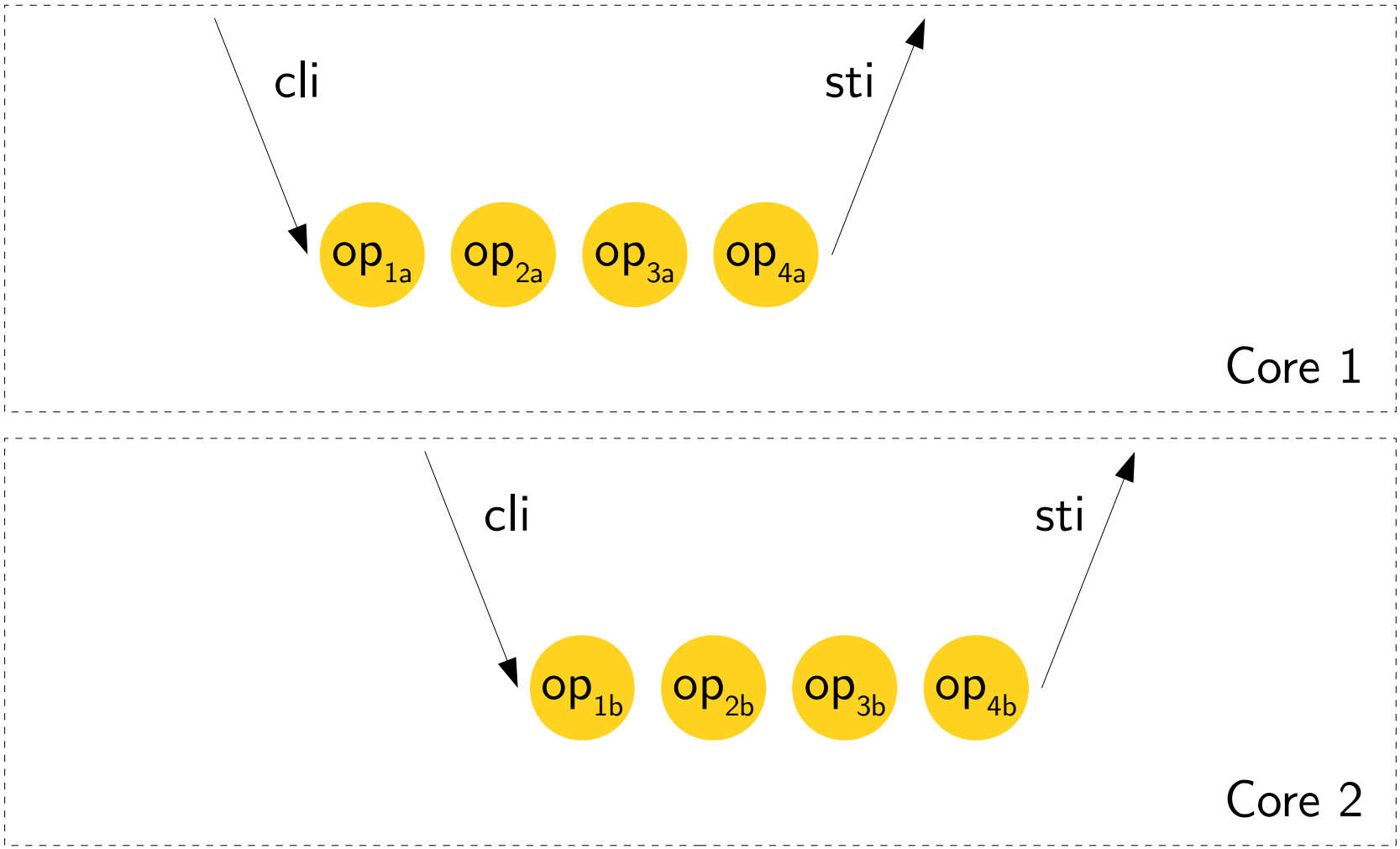
Simplifying the Multicore Verification



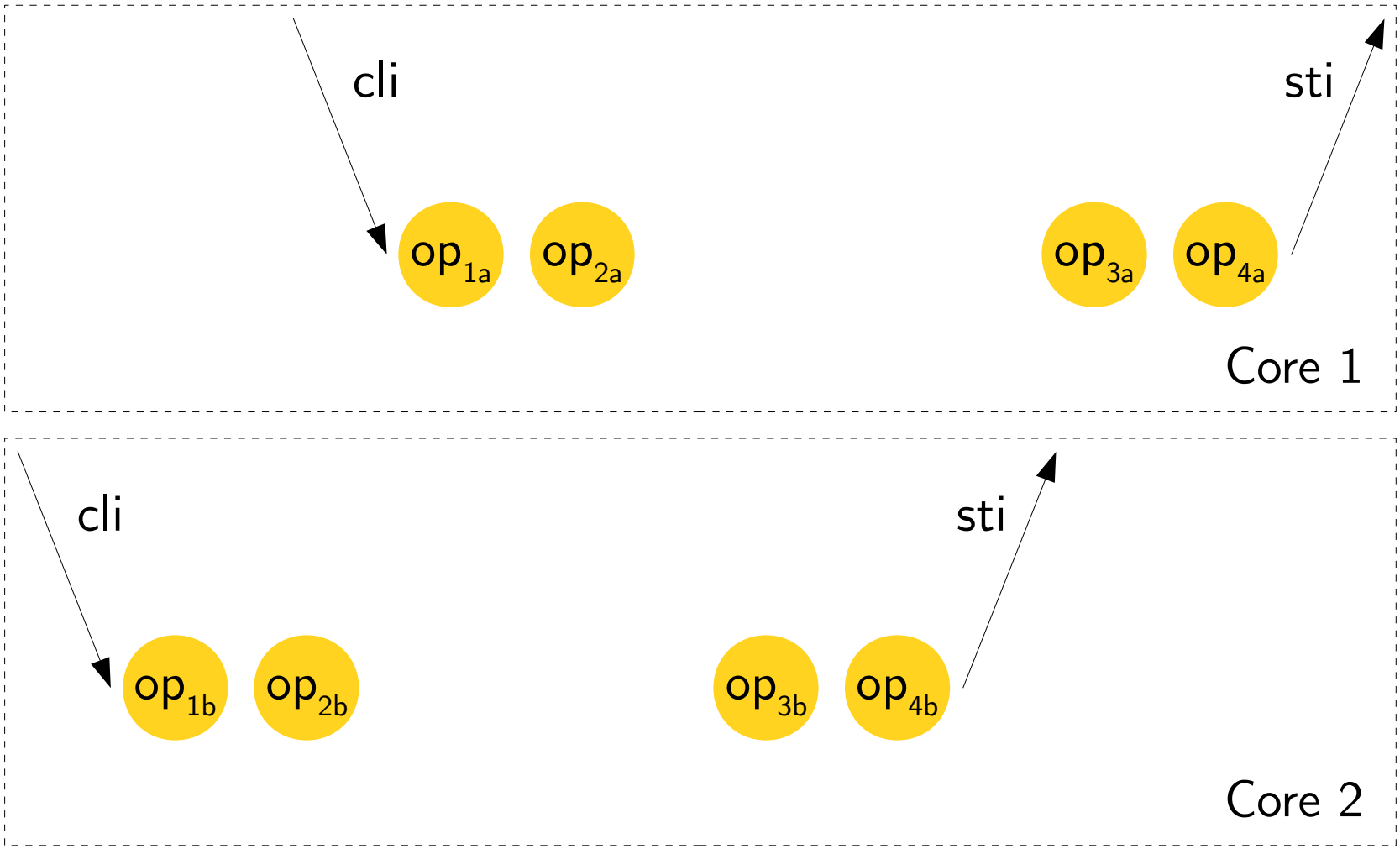
Simplifying the Multicore Verification



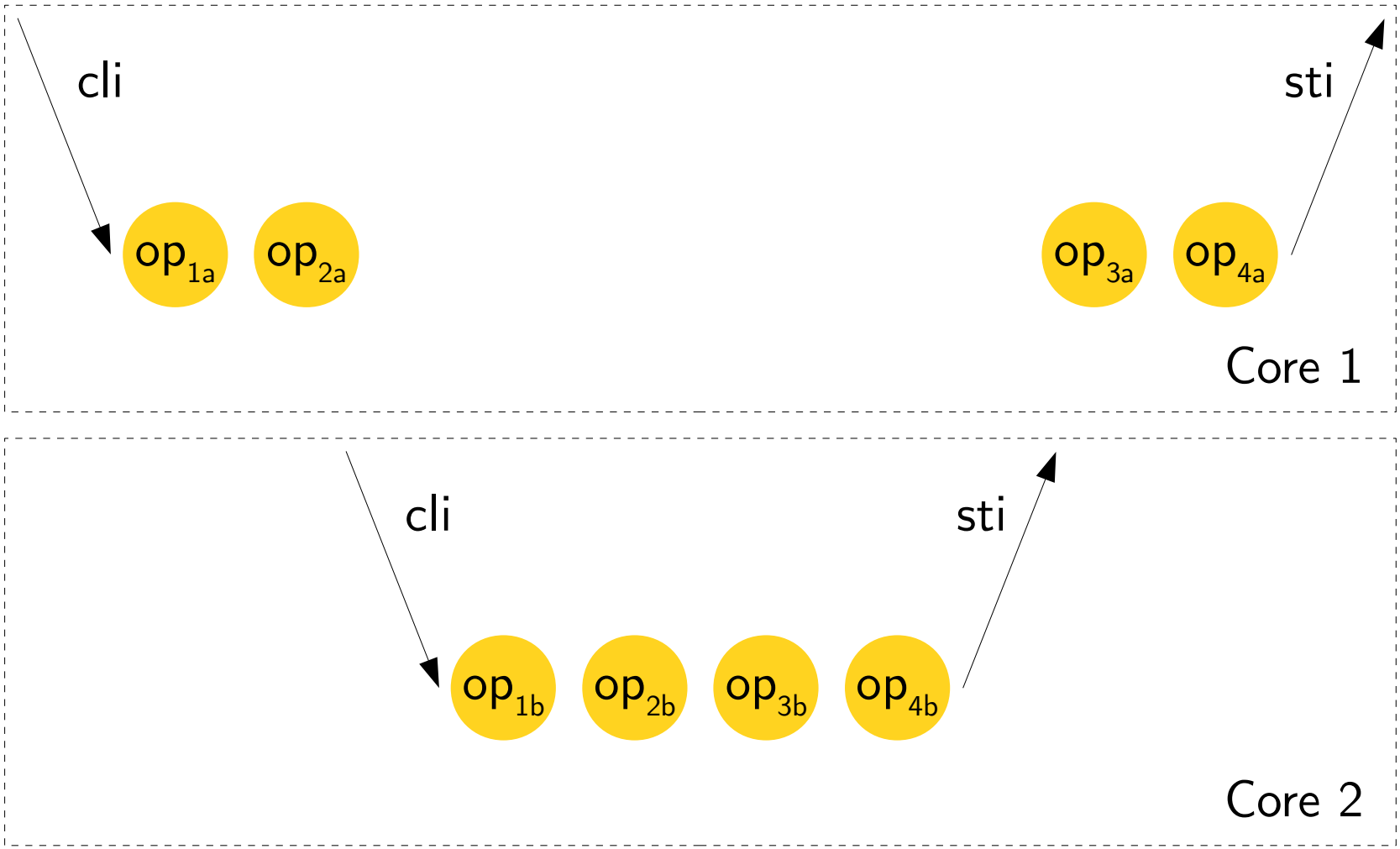
Simplifying the Multicore Verification



Simplifying the Multicore Verification



Simplifying the Multicore Verification



A short introduction into *transactional memory*

Introduction into Transactional Memory

operation$_1$

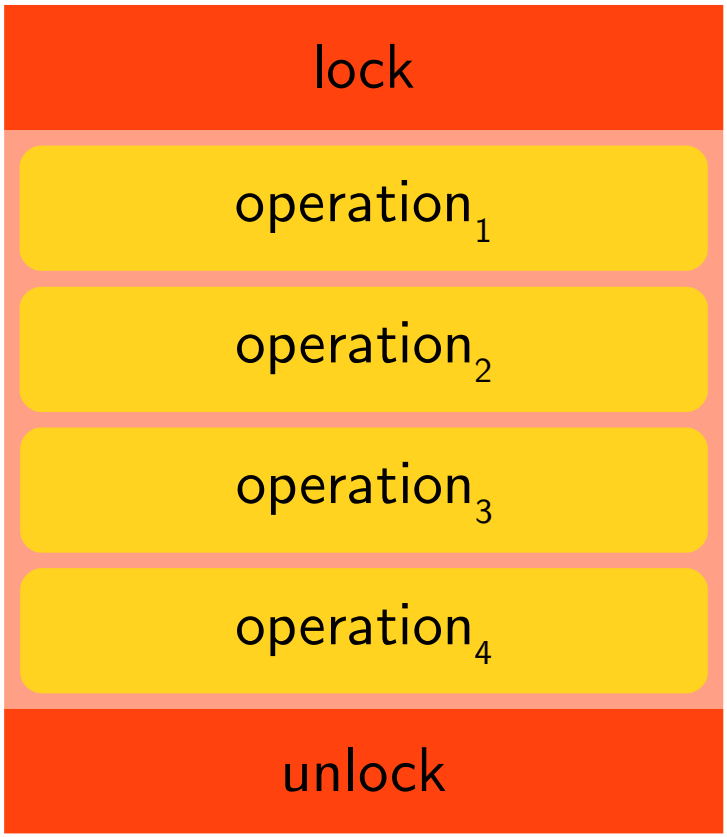
operation$_2$

operation$_3$

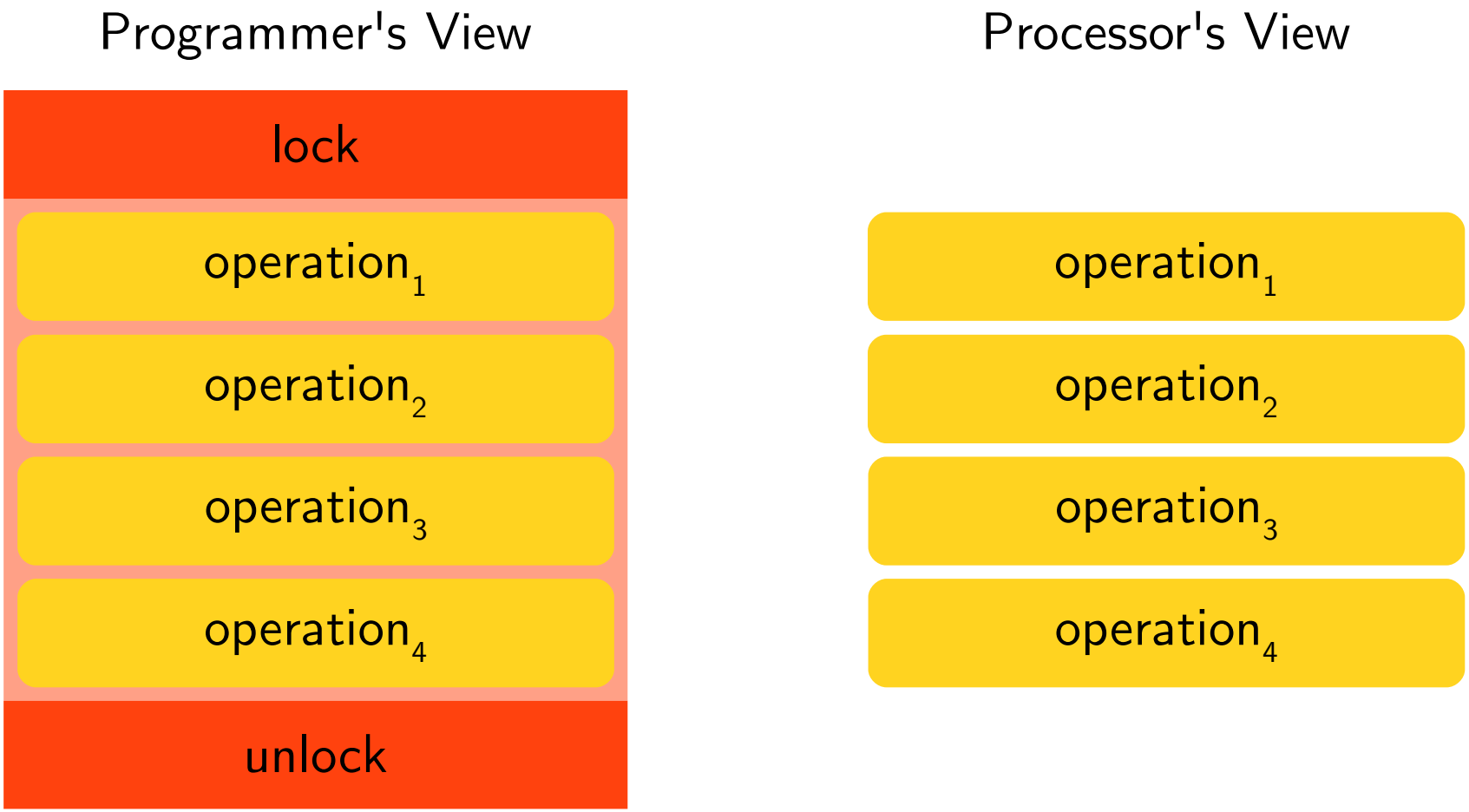
operation$_4$

Introduction into Transactional Memory

Programmer's View

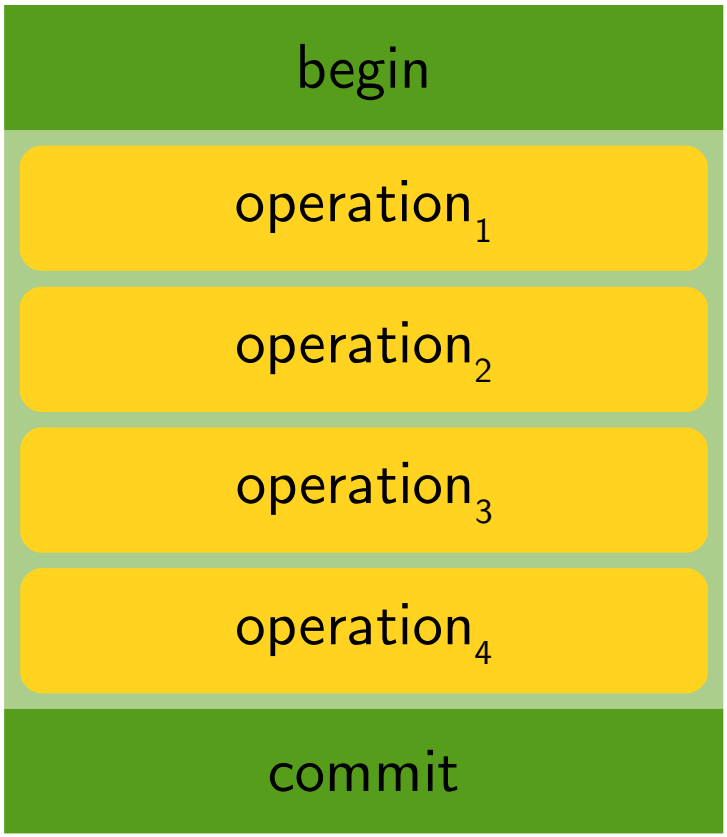


Introduction into Transactional Memory



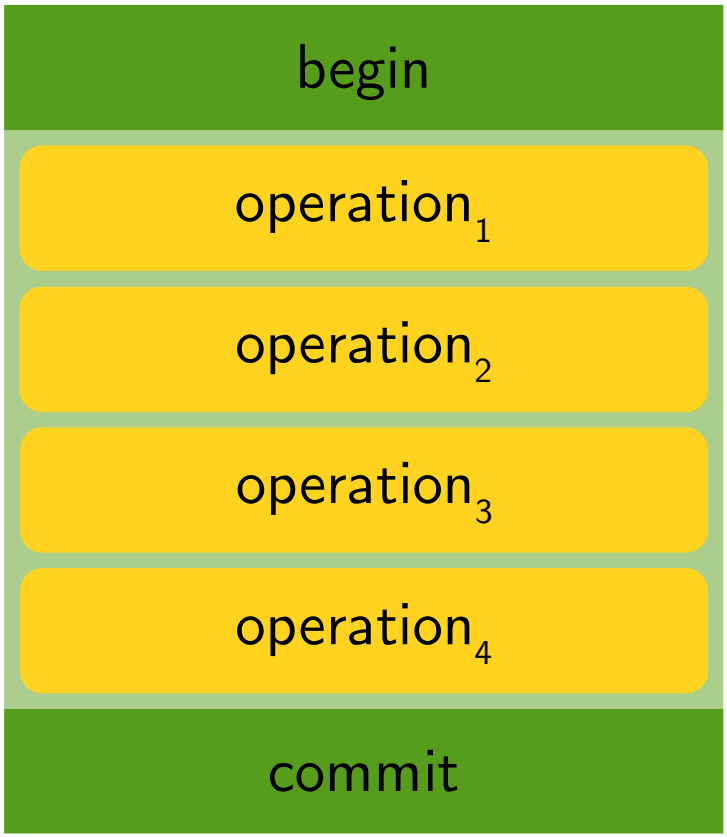
Introduction into Transactional Memory

Programmer's View



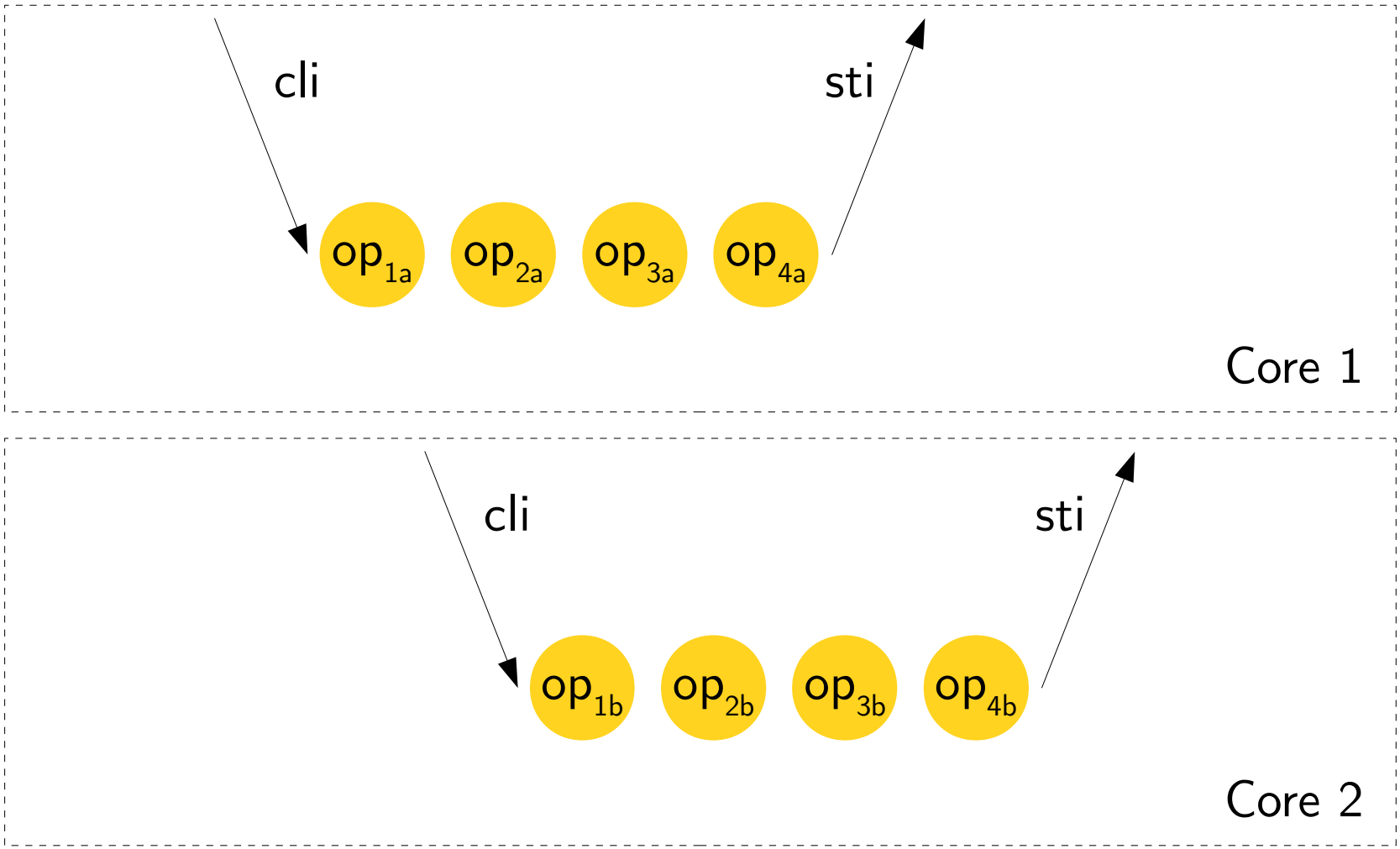
Introduction into Transactional Memory

Programmer's View

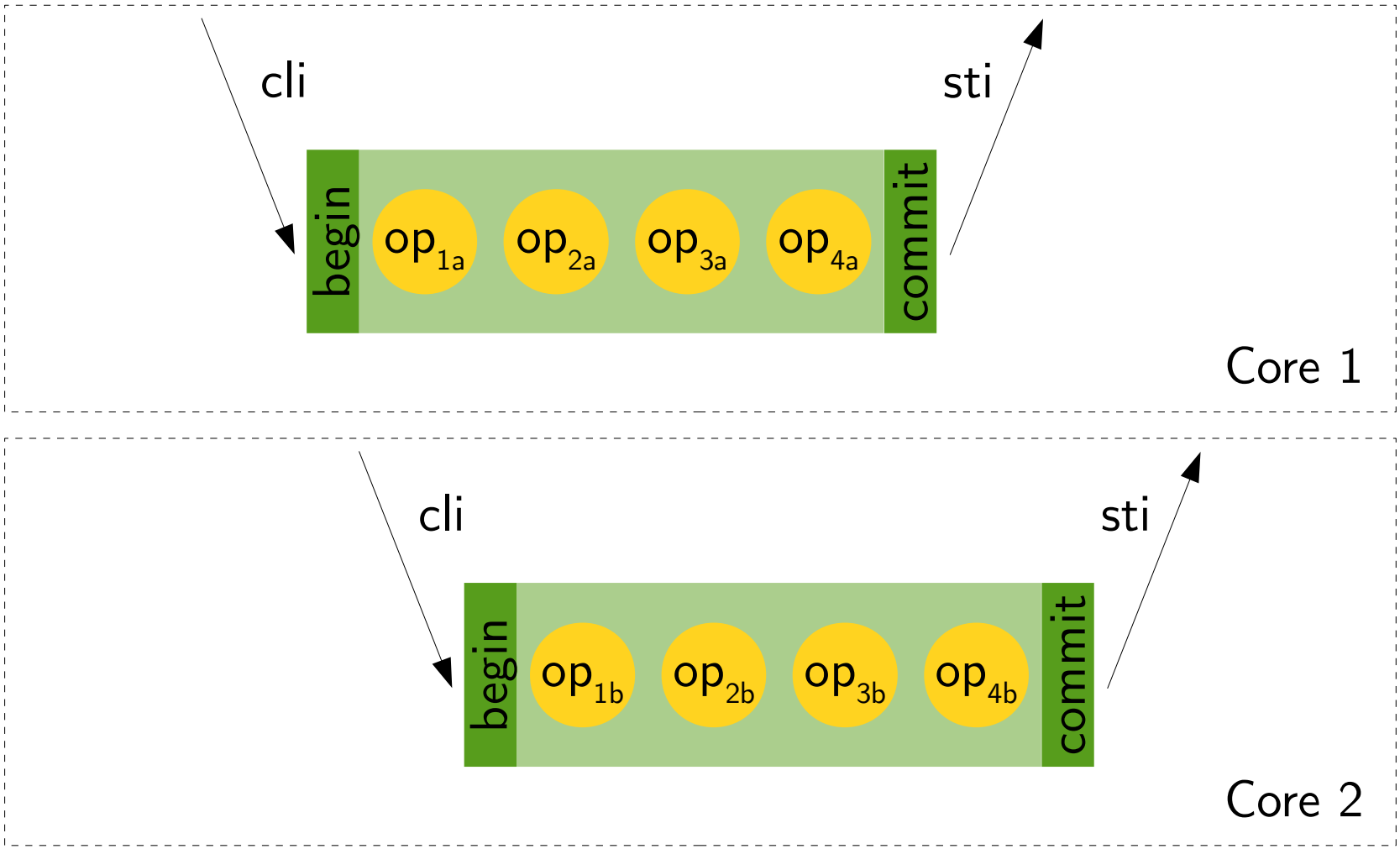


Processor's View

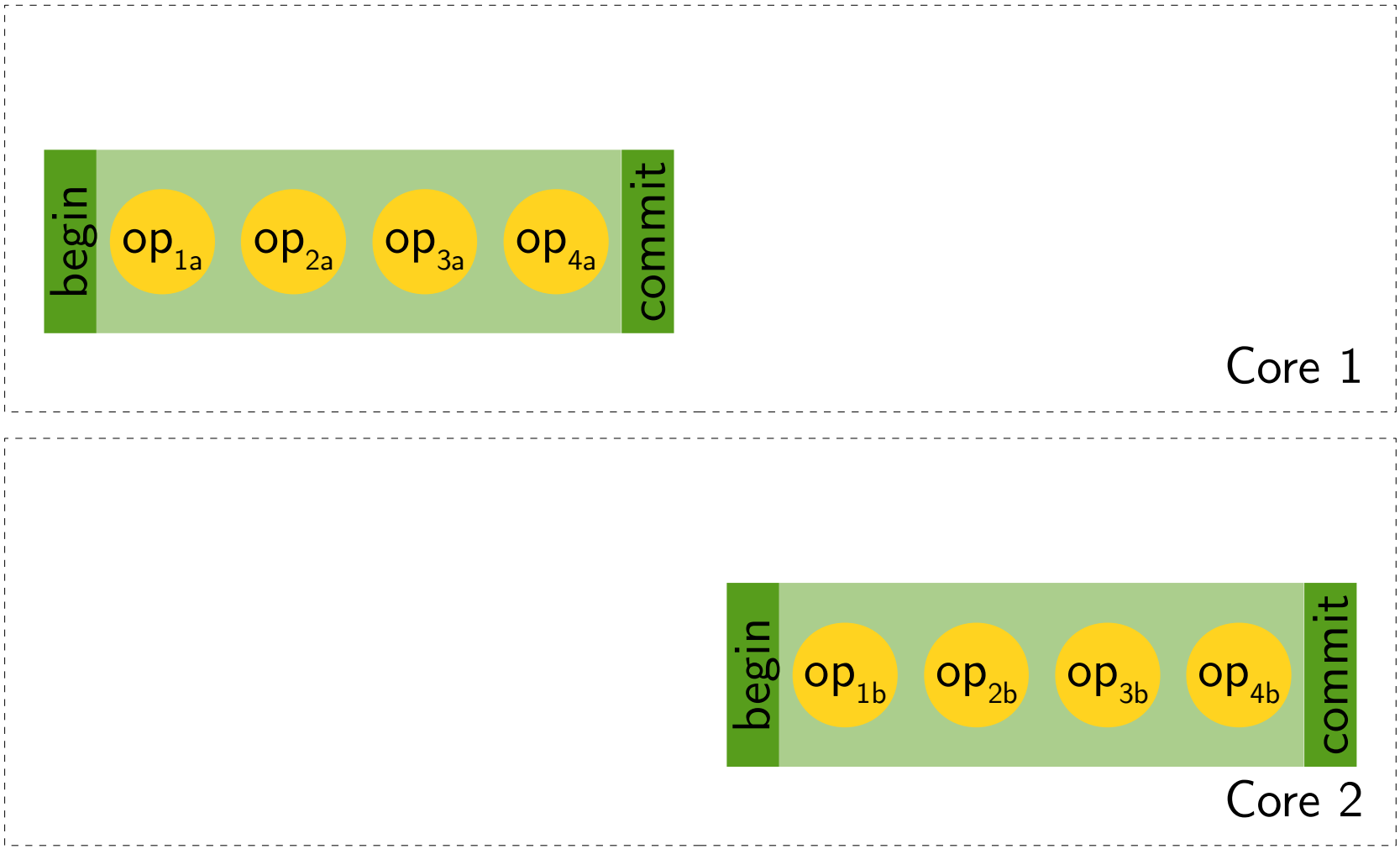
Simplifying the Multicore Verification



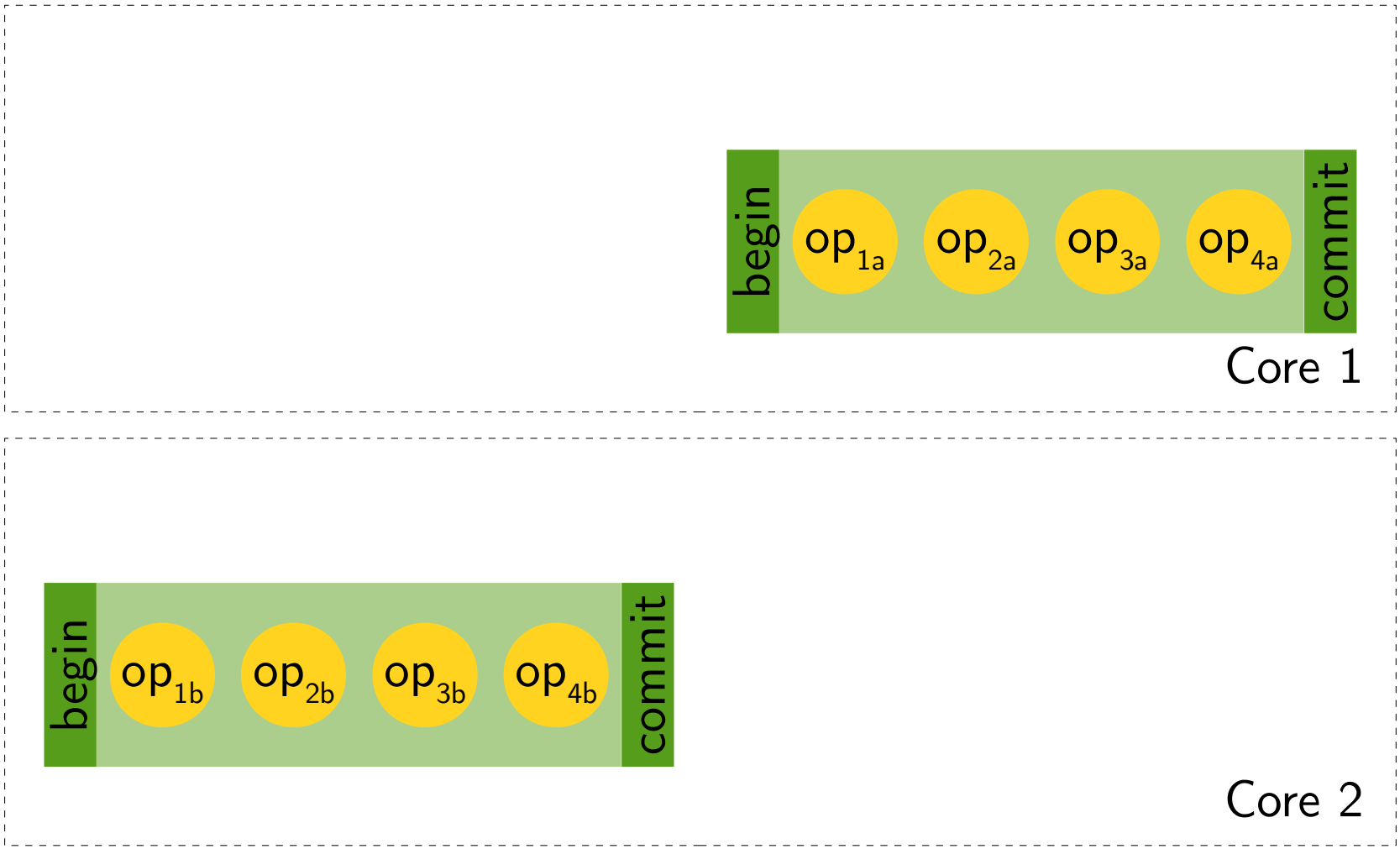
Simplifying the Multicore Verification



Simplifying the Multicore Verification

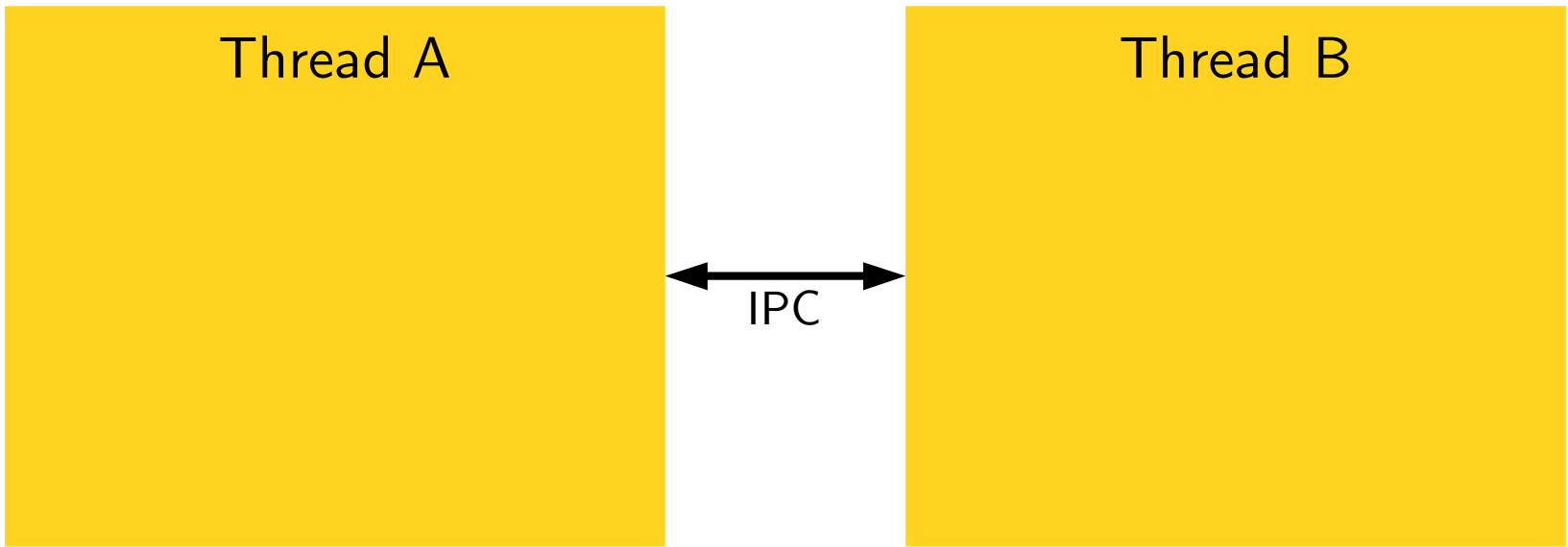


Simplifying the Multicore Verification

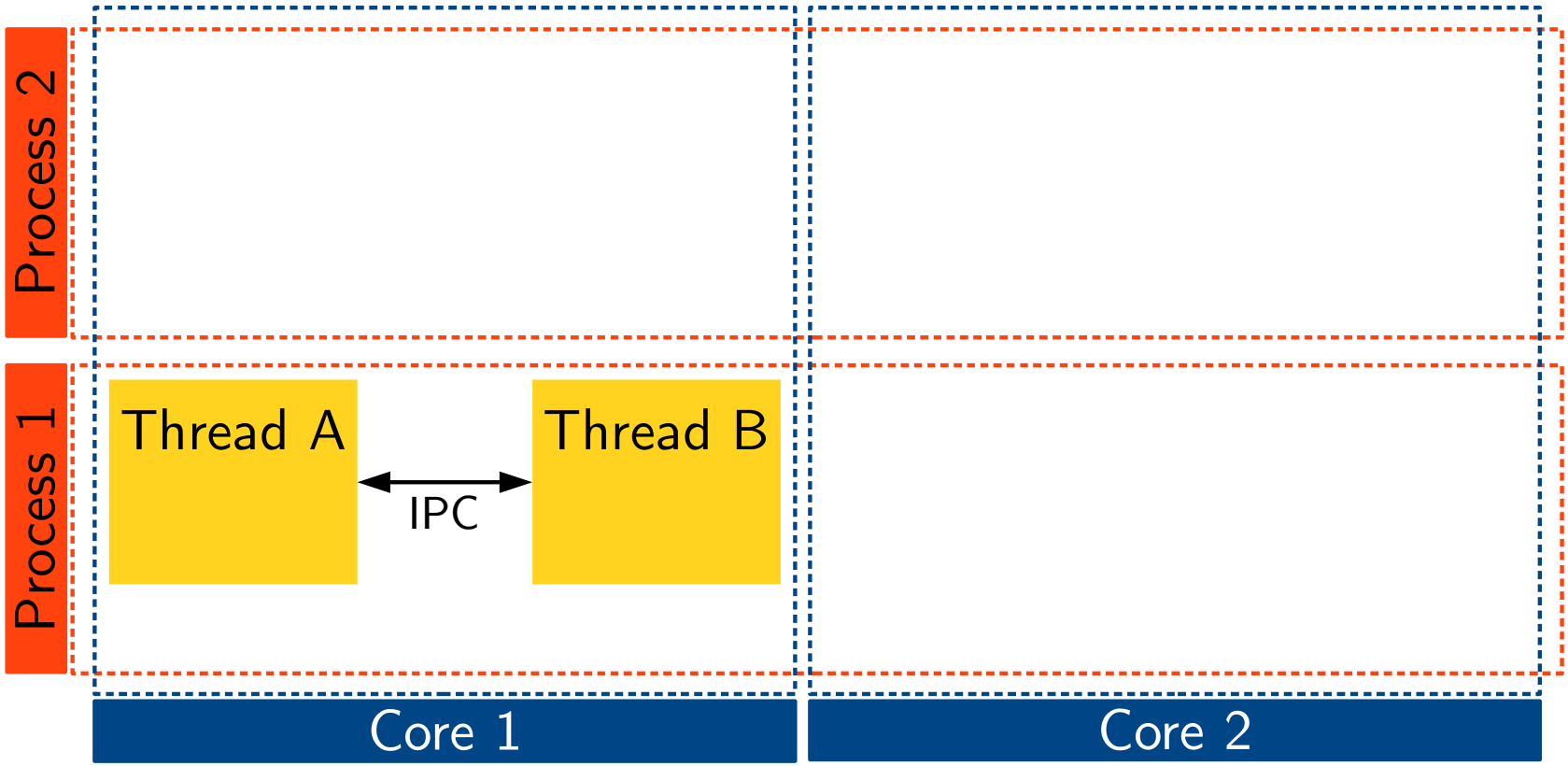


Transactional L4/Fiasco.OC

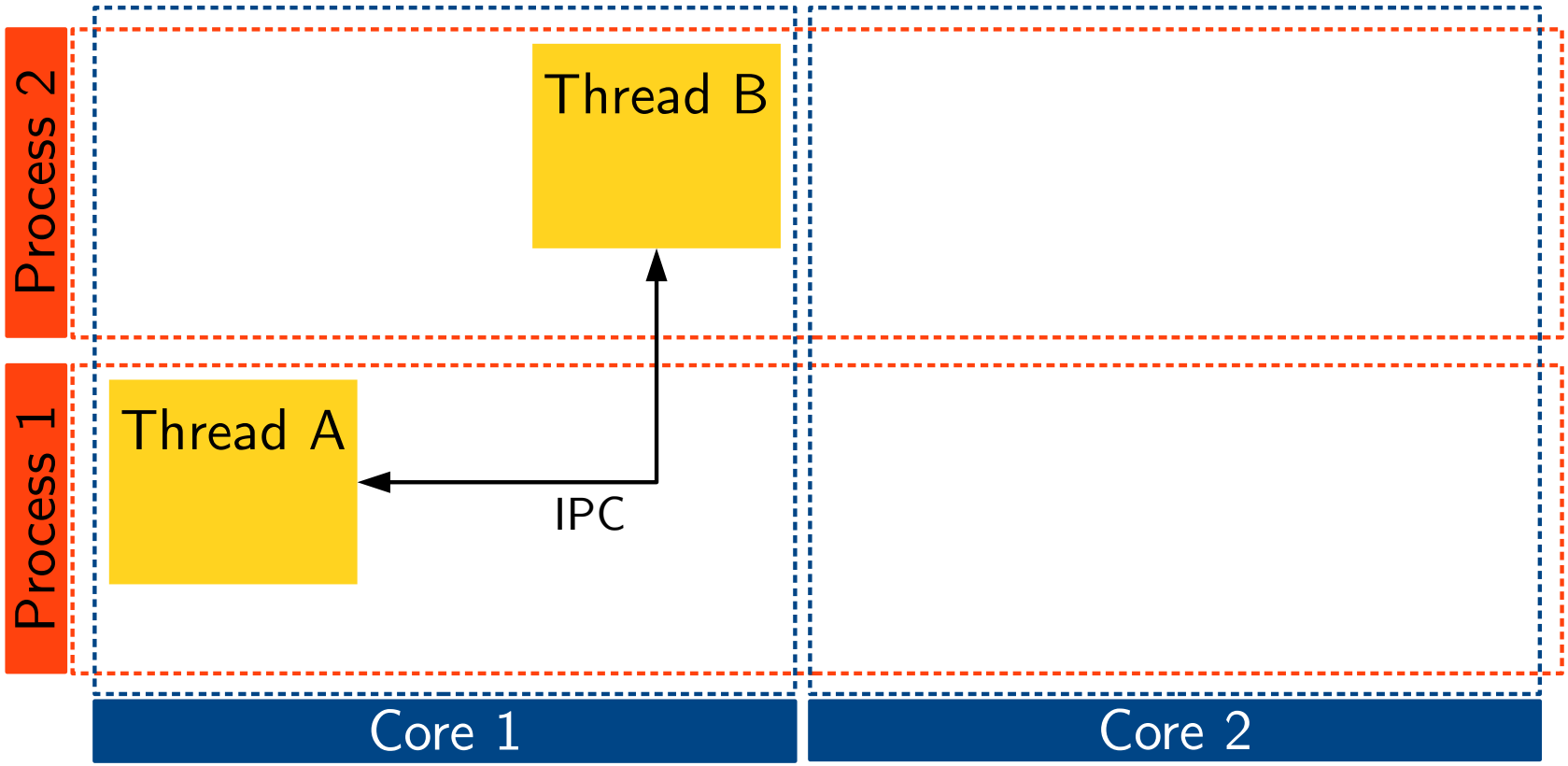
Inter-Process Communication is the main building block of the L4/Fiasco.OC microkernel.



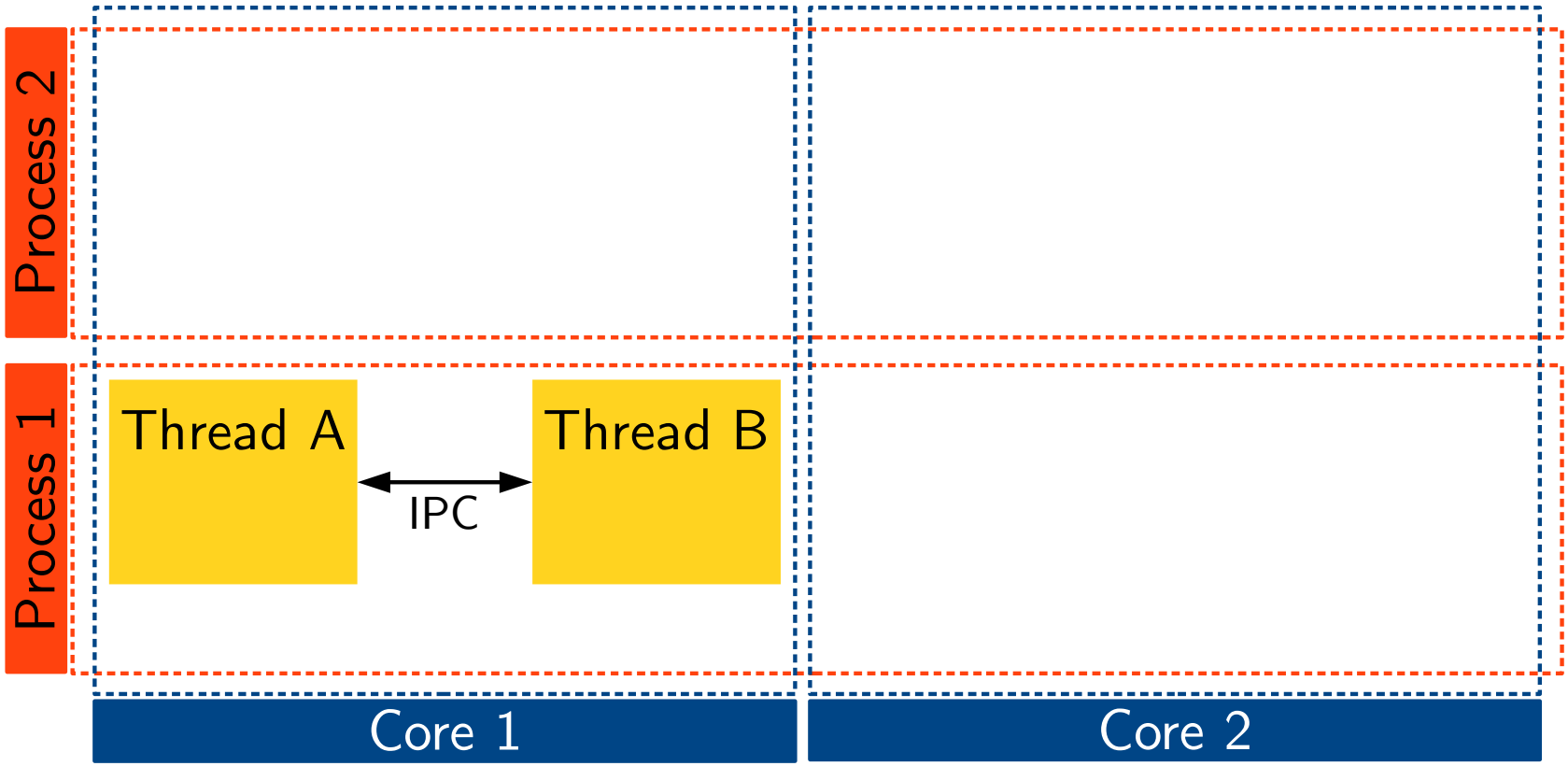
Intra Process IPC



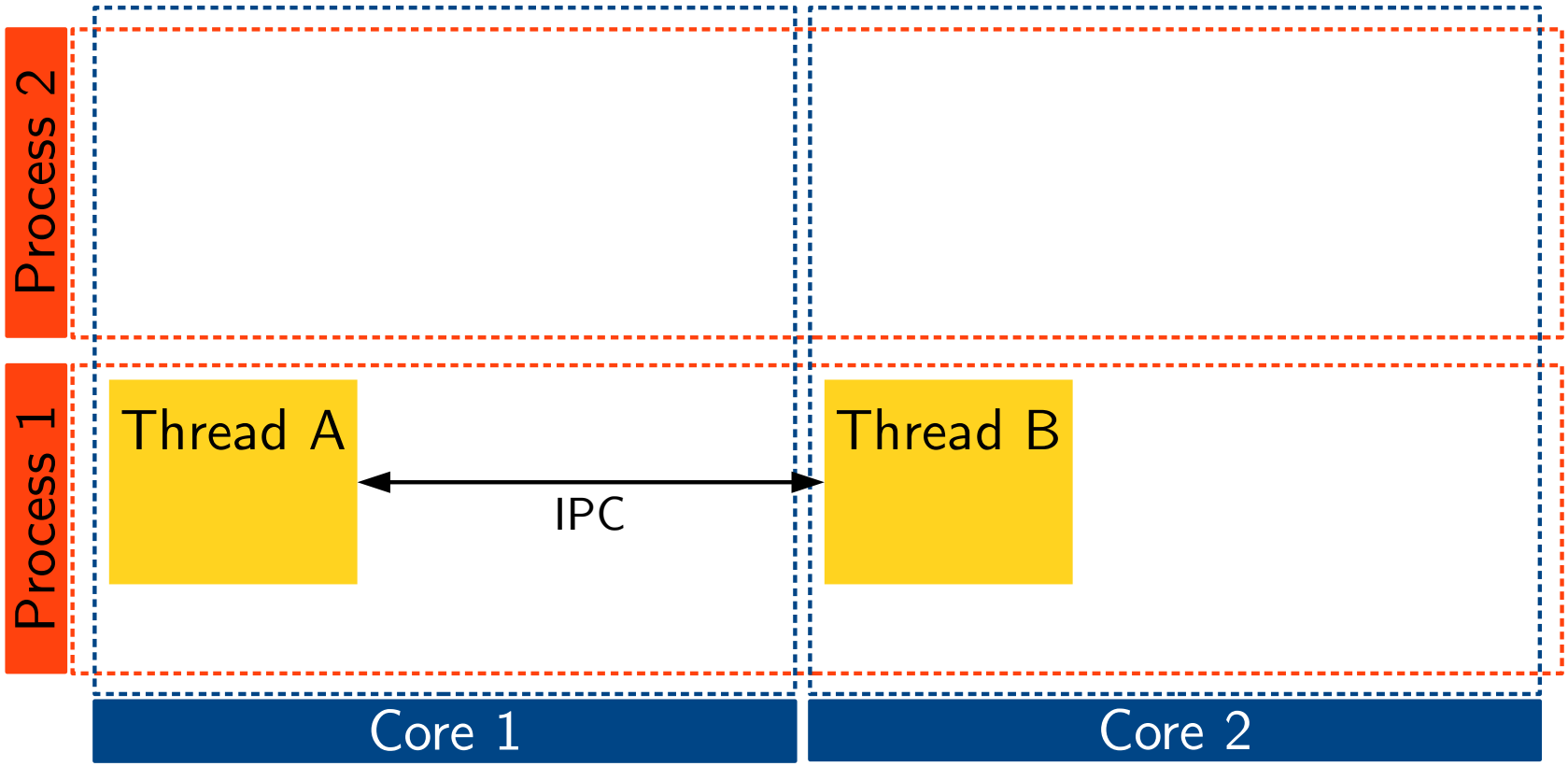
Inter Process IPC



Core-Local-IPC

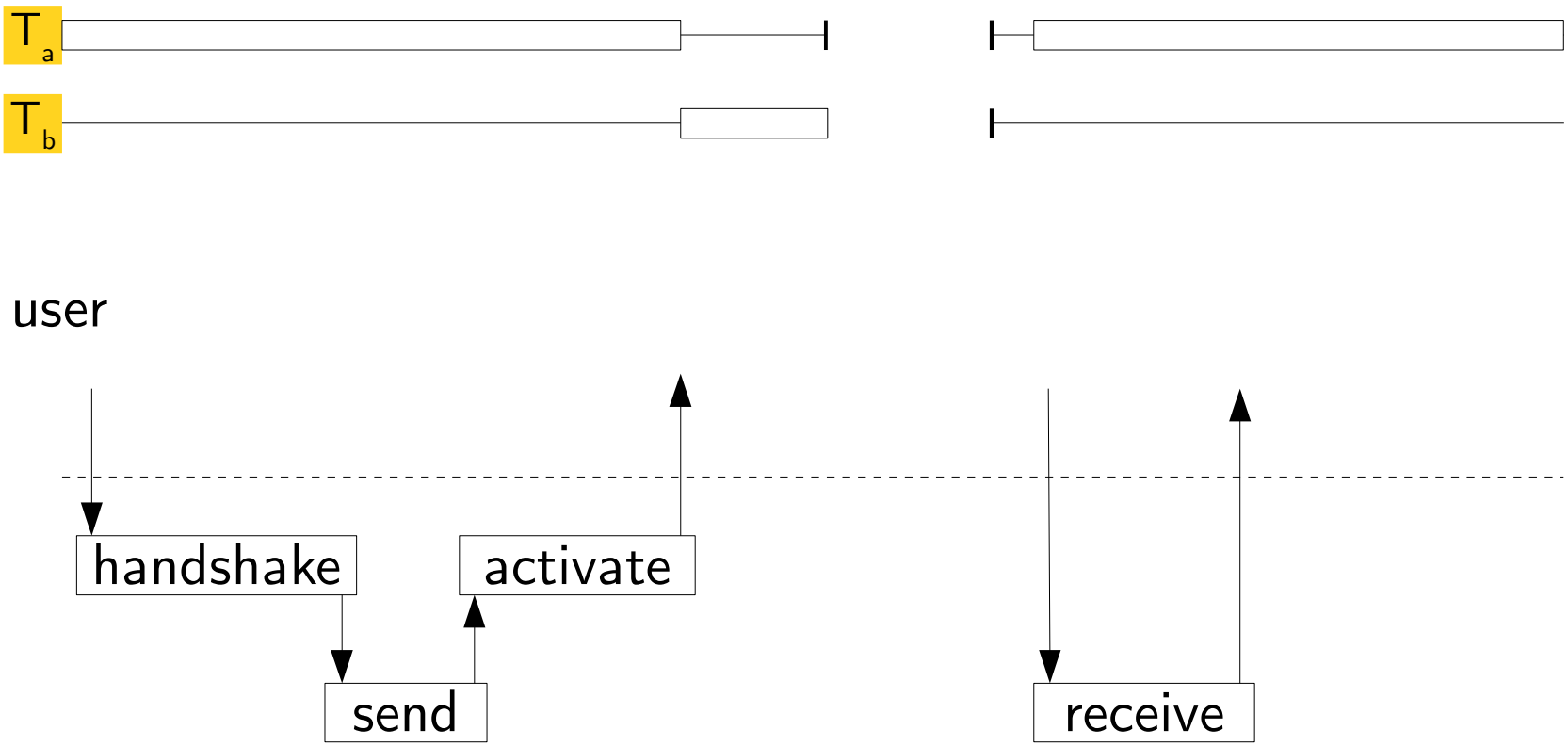


X-Core-IPC



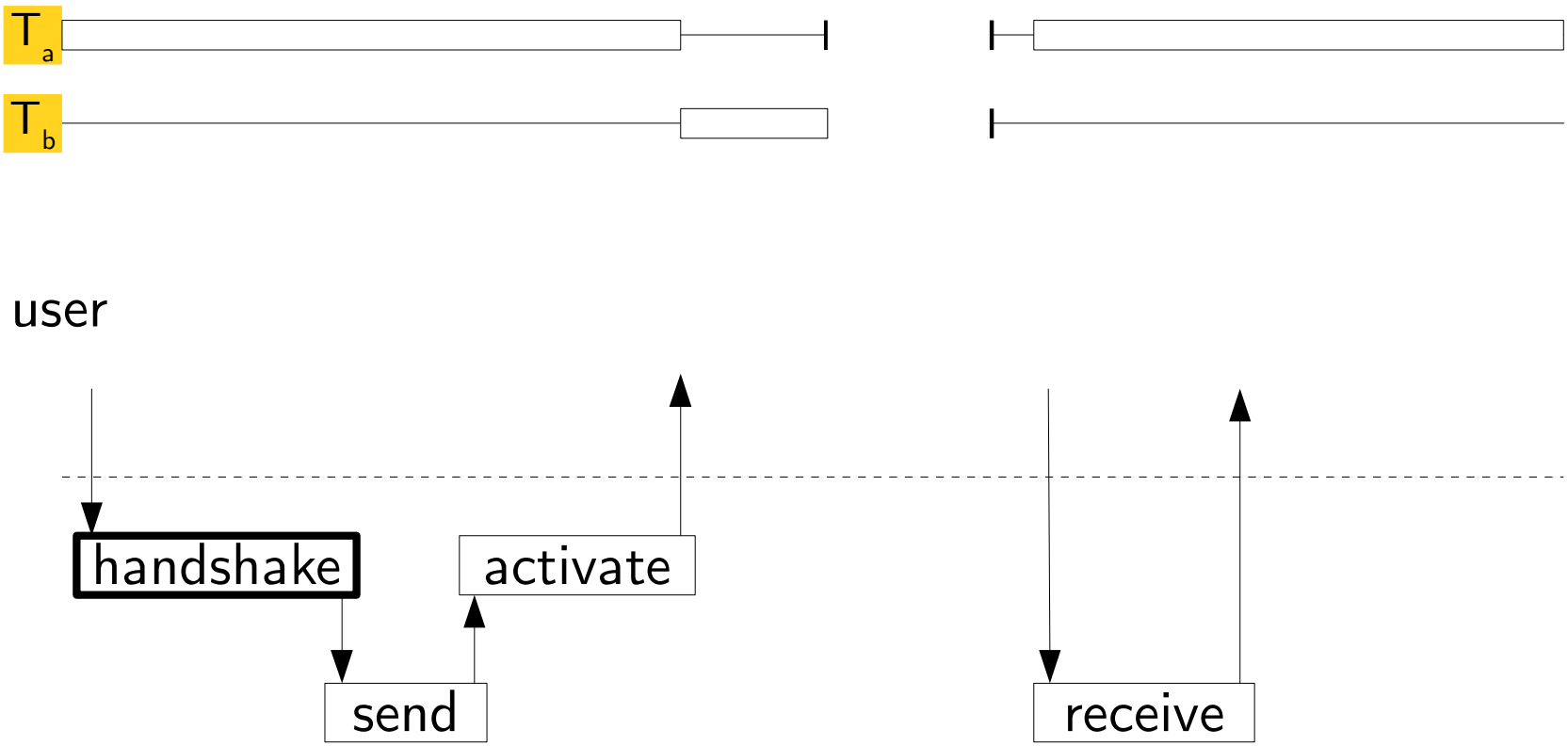
Transactional IPC

Making the System Call Transactional



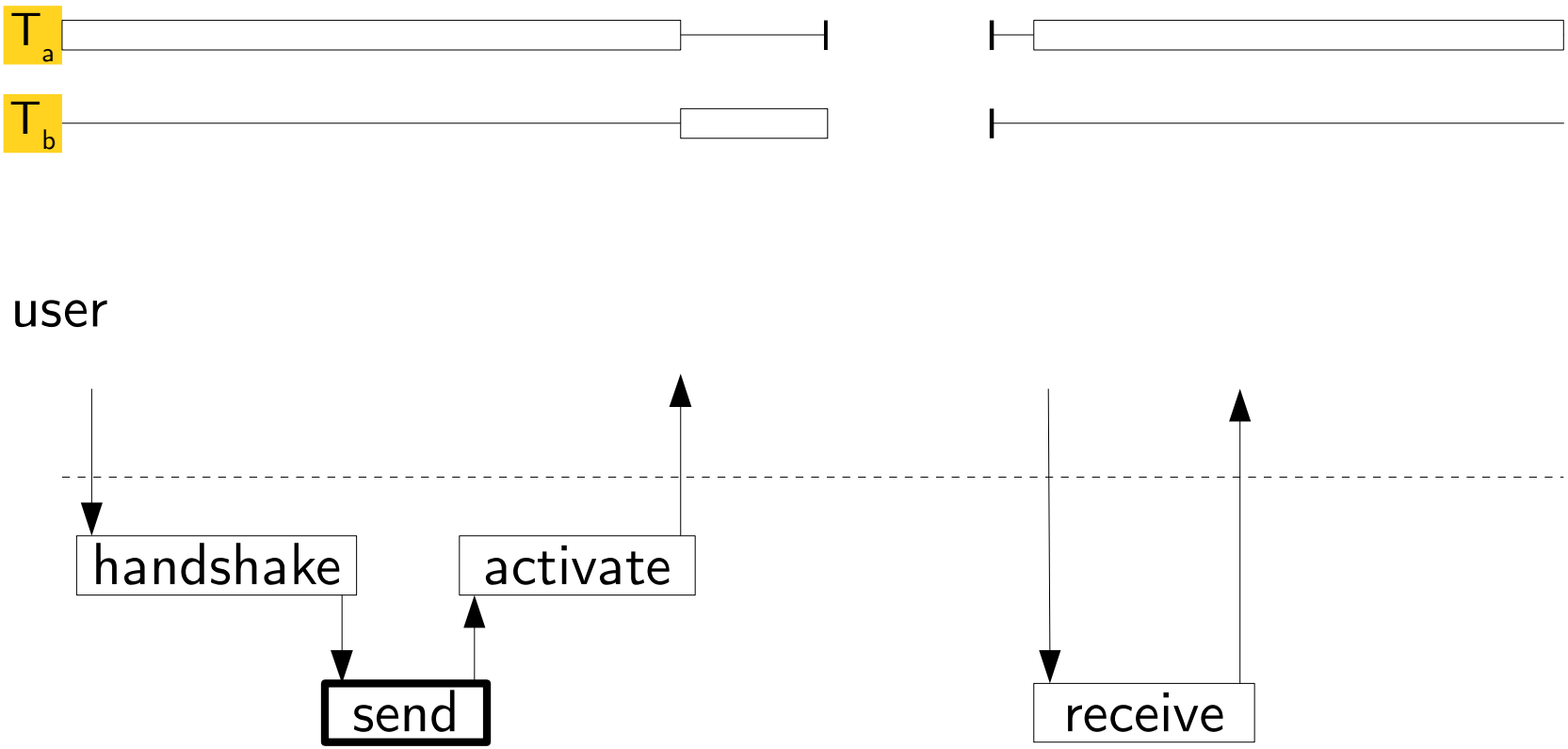
Transactional IPC

Making the System Call Transactional



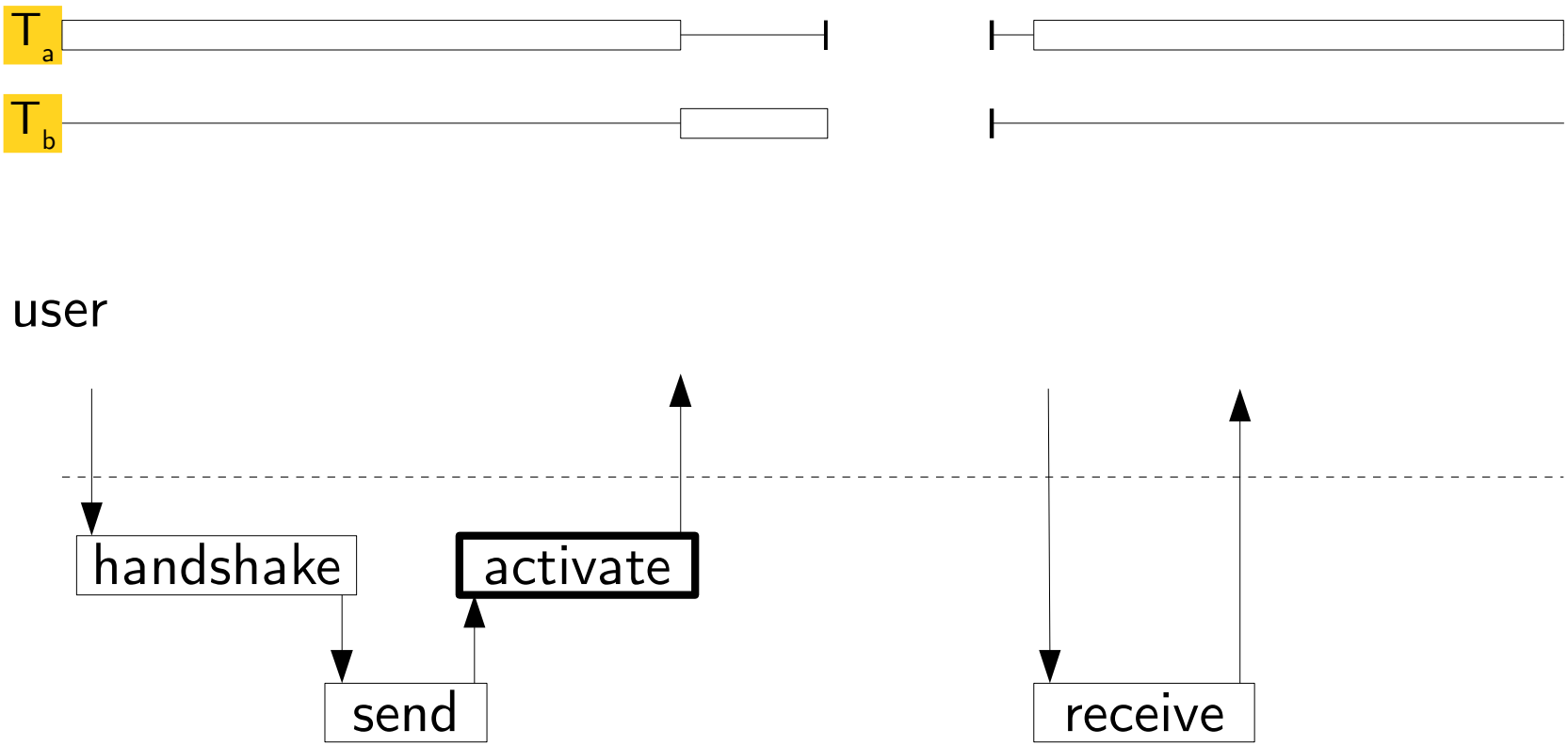
Transactional IPC

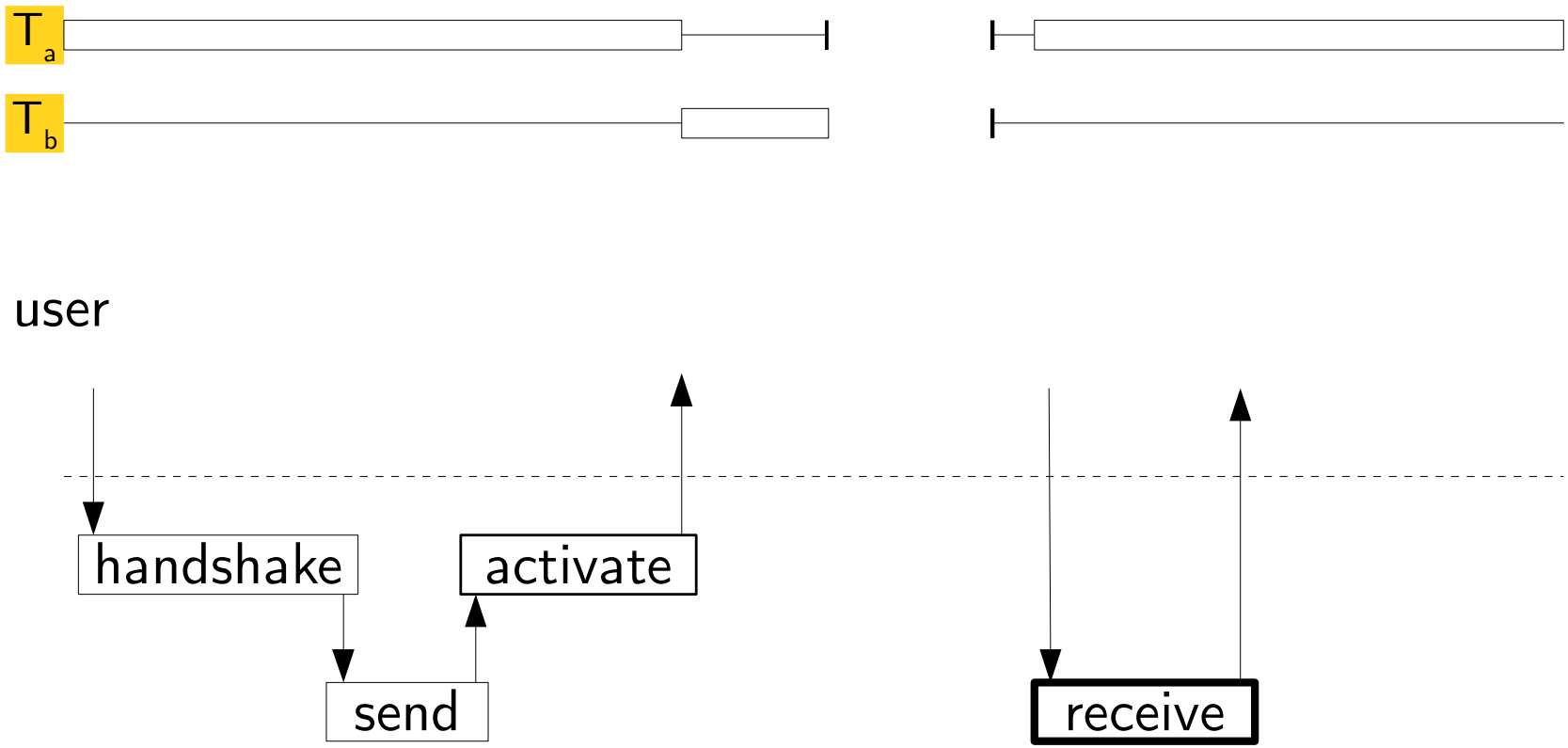
Making the System Call Transactional



Transactional IPC

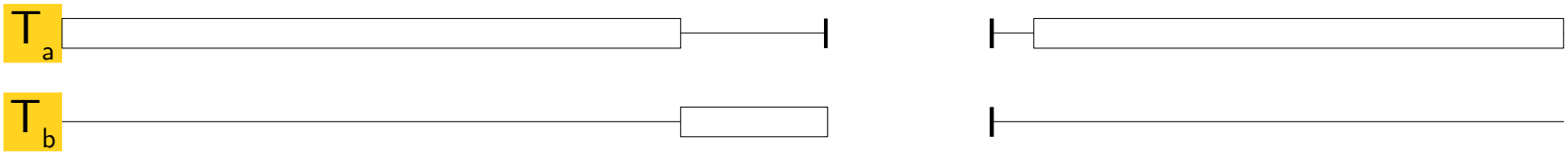
Making the System Call Transactional



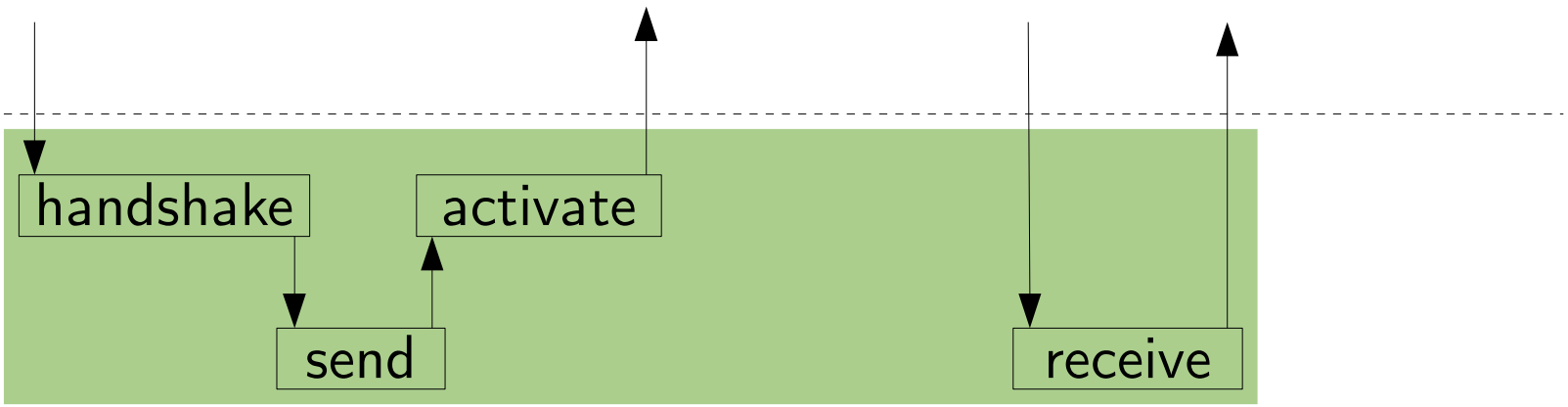

T a
T b
user
handshake
activate
send
receive
kernel

Transactional IPC

Making the System Call Transactional



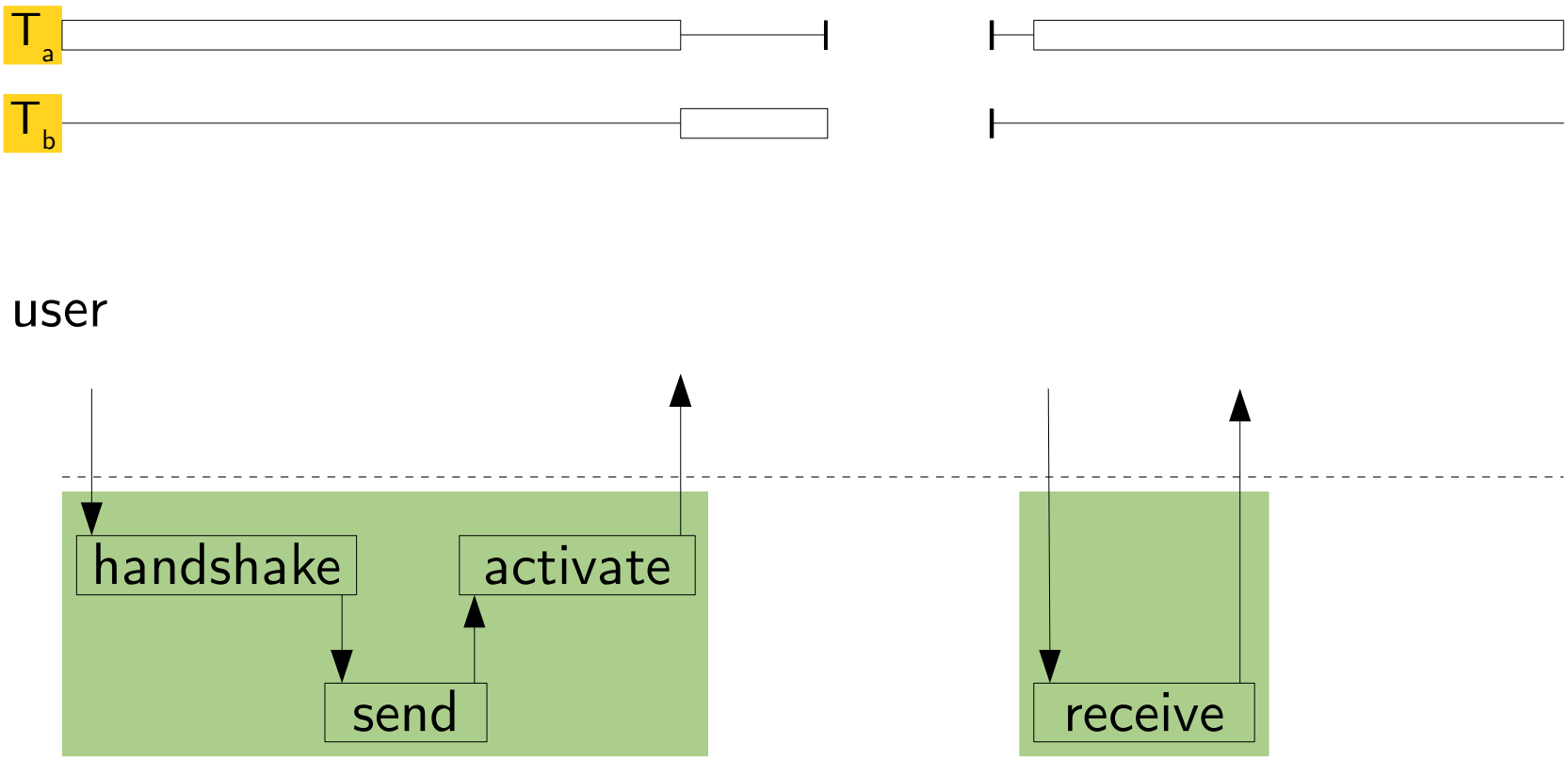
user



kernel

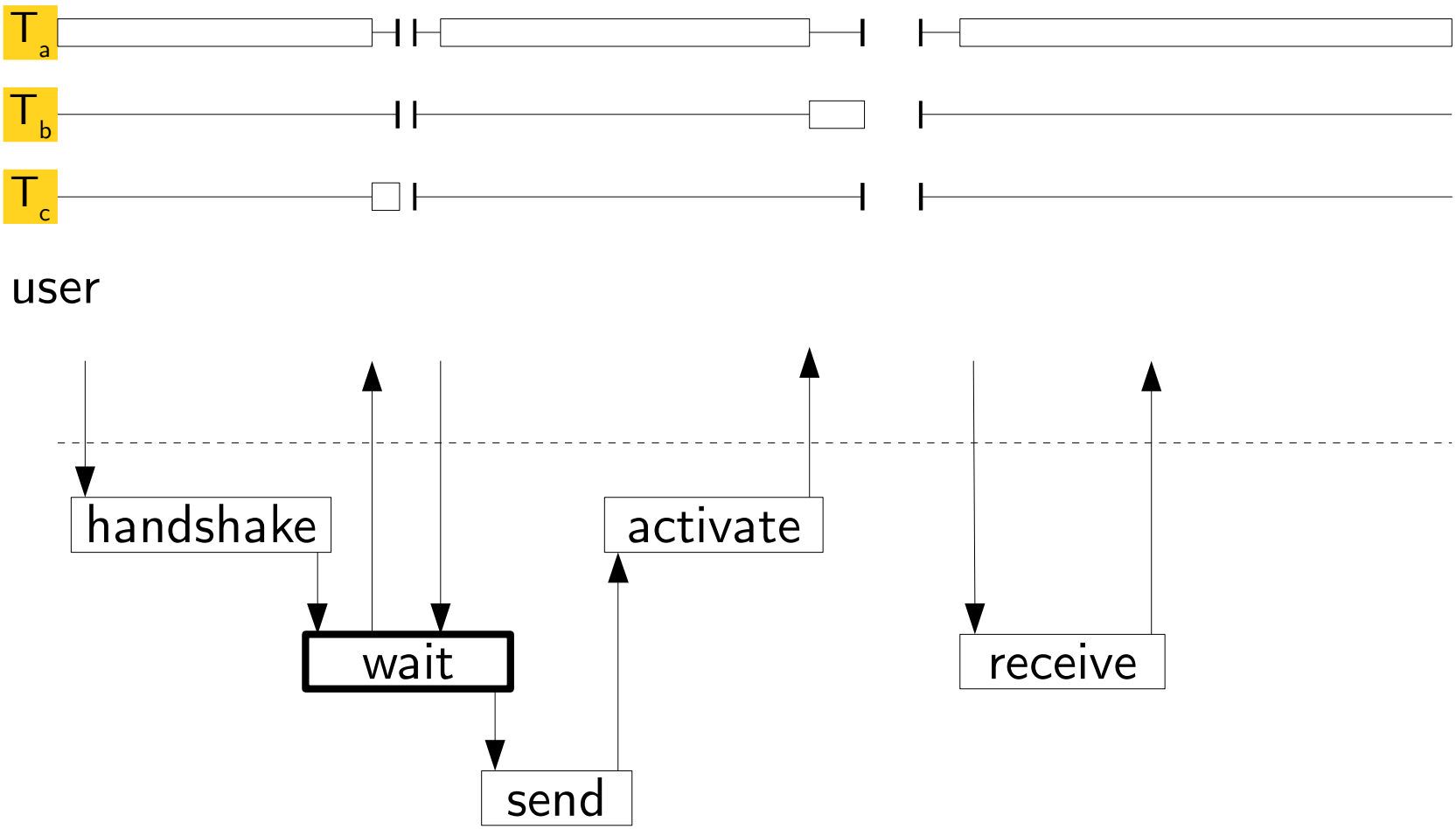
Transactional IPC

Making the System Call Transactional



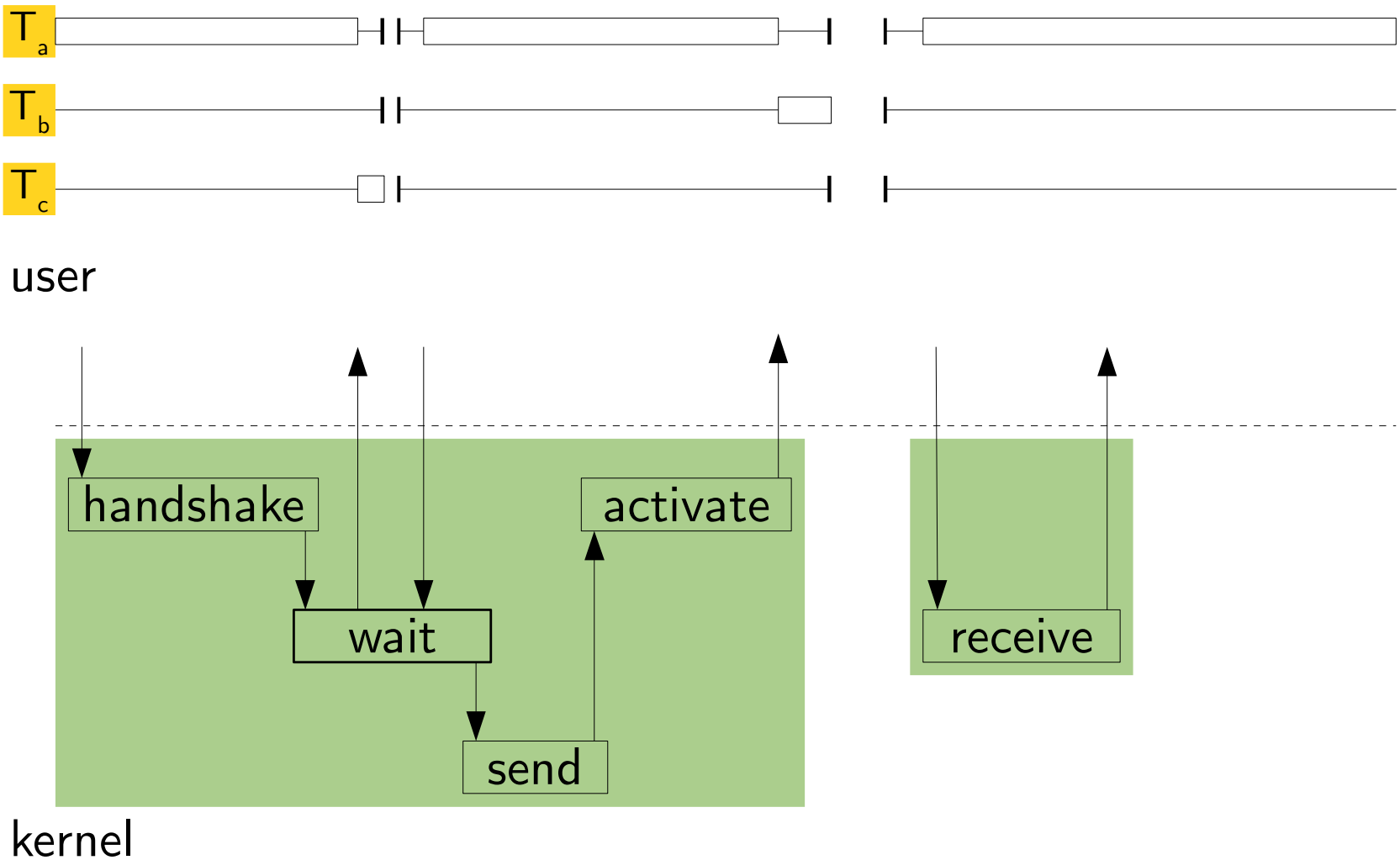
Transactional IPC

Making the System Call Transactional



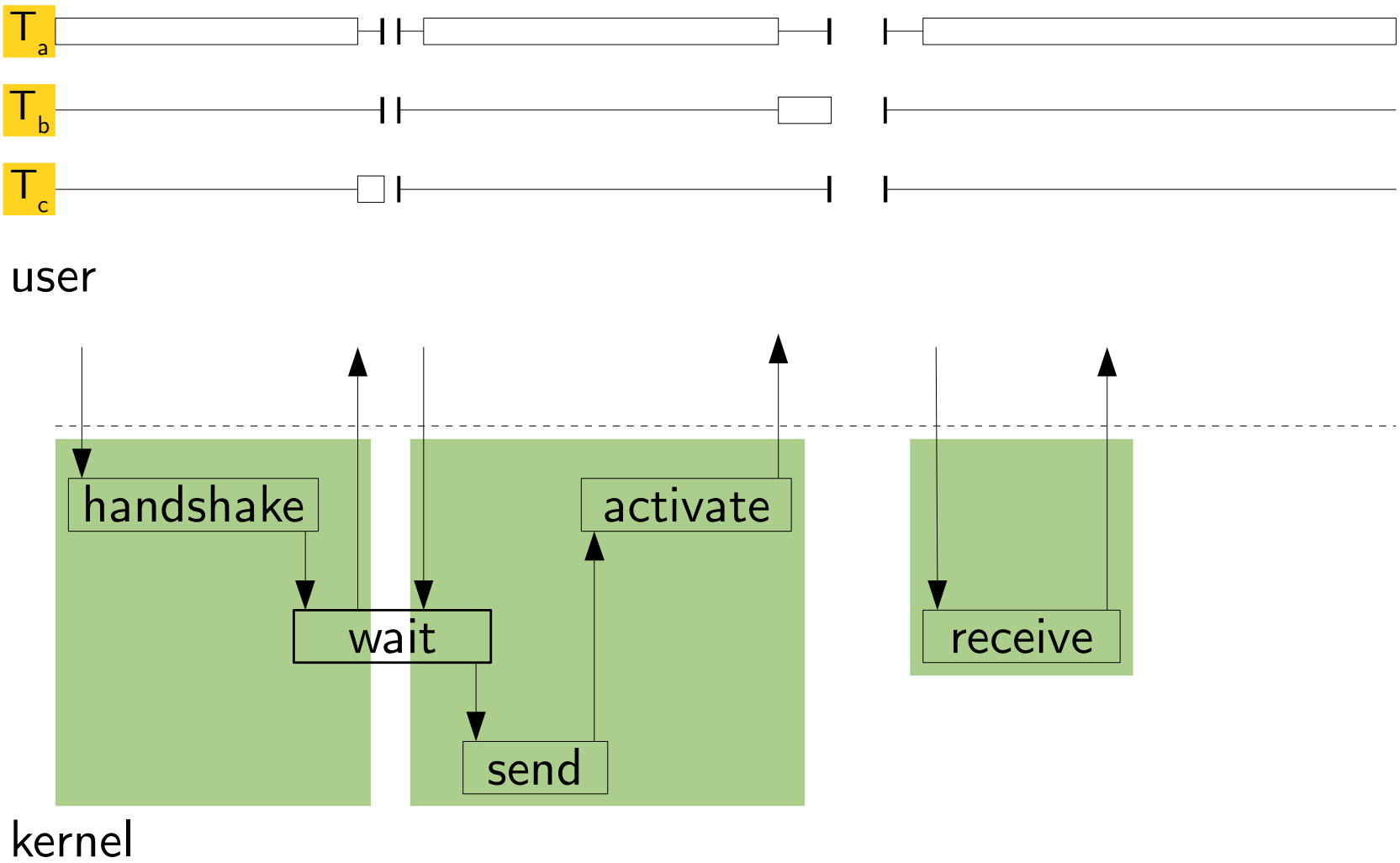
Transactional IPC

Making the System Call Transactional



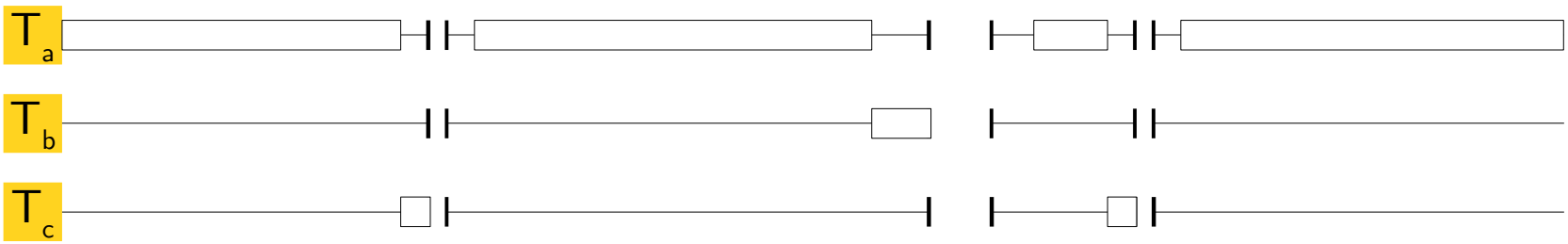
Transactional IPC

Making the System Call Transactional

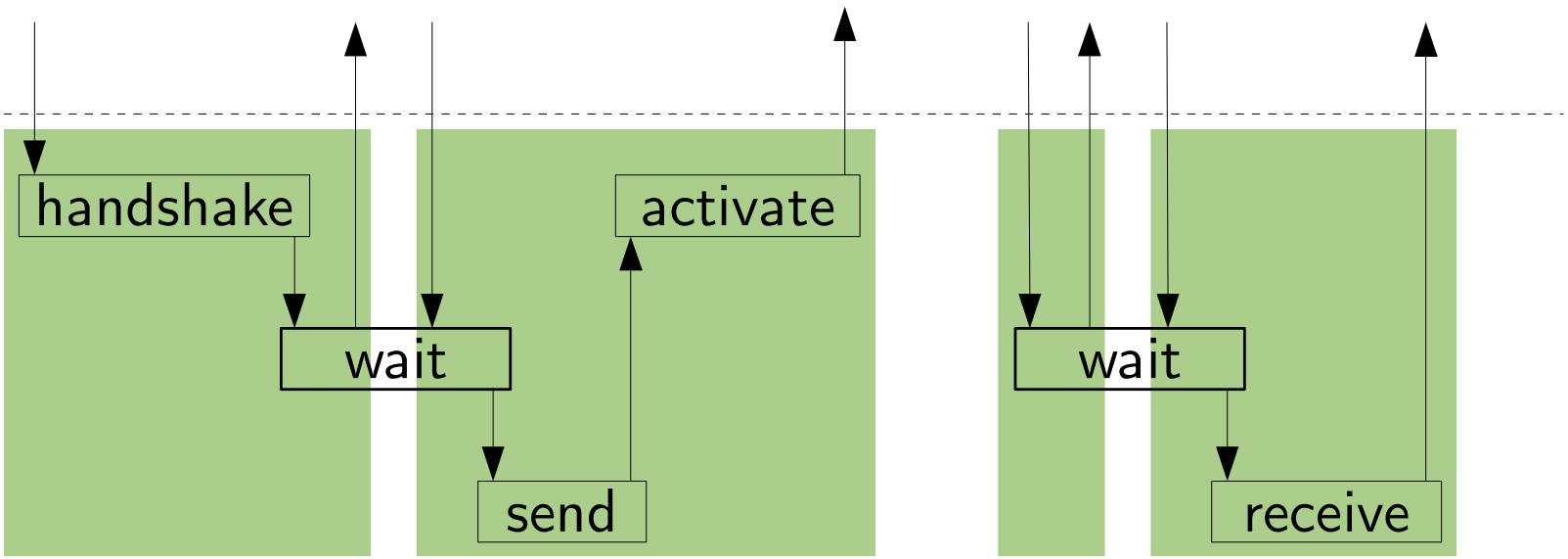


Transactional IPC

Making the System Call Transactional



user



kernel

Evaluation

For Fiasco.OC it is necessary to distinguish in the evaluation between *Core-Local-IPC* and *X-Core-IPC*.

Evaluation

For Fiasco.OC it is necessary to distinguish in the evaluation between *Core-Local-IPC* and *X-Core-IPC*.

Core-Local-IPC uses an optimized path with no internal synchronization.

- only core local state is altered
- protection is achieved via disabling interrupts

Evaluation

For Fiasco.OC it is necessary to distinguish in the evaluation between *Core-Local-IPC* and *X-Core-IPC*.

X-Core-IPC requires additional synchronization as non-local state is changed.

- need to execute the IPC on one of the two partner's cores
- meanwhile the other's core is interrupted
- expensive IPIs are used for this negotiation

Evaluation

Complexity

The introduction of transactions into the IPC path adds internal synchronization to it.

Evaluation

Complexity

The introduction of transactions into the IPC path adds internal synchronization to it.

Core-Local-IPC path still uses the same optimized path.

- local data accesses are additionally protected by transactions

No complexity increase or decrease in the *Core-Local-IPC* operation.

Evaluation

Complexity

The introduction of transactions into the IPC path adds internal synchronization to it.

X-Core-IPC can use optimized path, too.

- non-local data accesses are protected by transactions
- no need to use expensive IPIs for synchronization

Complexity of the *X-Core-IPC* operation could be reduced to the much simpler *Core-Local-IPC* one.

Evaluation

Complexity

All complexity analyses assume that transactions itself do not increase the complexity.

Evaluation

Complexity

All complexity analyses assume that transactions itself do not increase the complexity.

- only valid if transactions provide *progress guarantees*

Evaluation

Complexity

All complexity analyses assume that transactions itself do not increase the complexity.

- only valid if transactions provide *progress guarantees*
- TSX does not provide such guarantee

Evaluation

Complexity

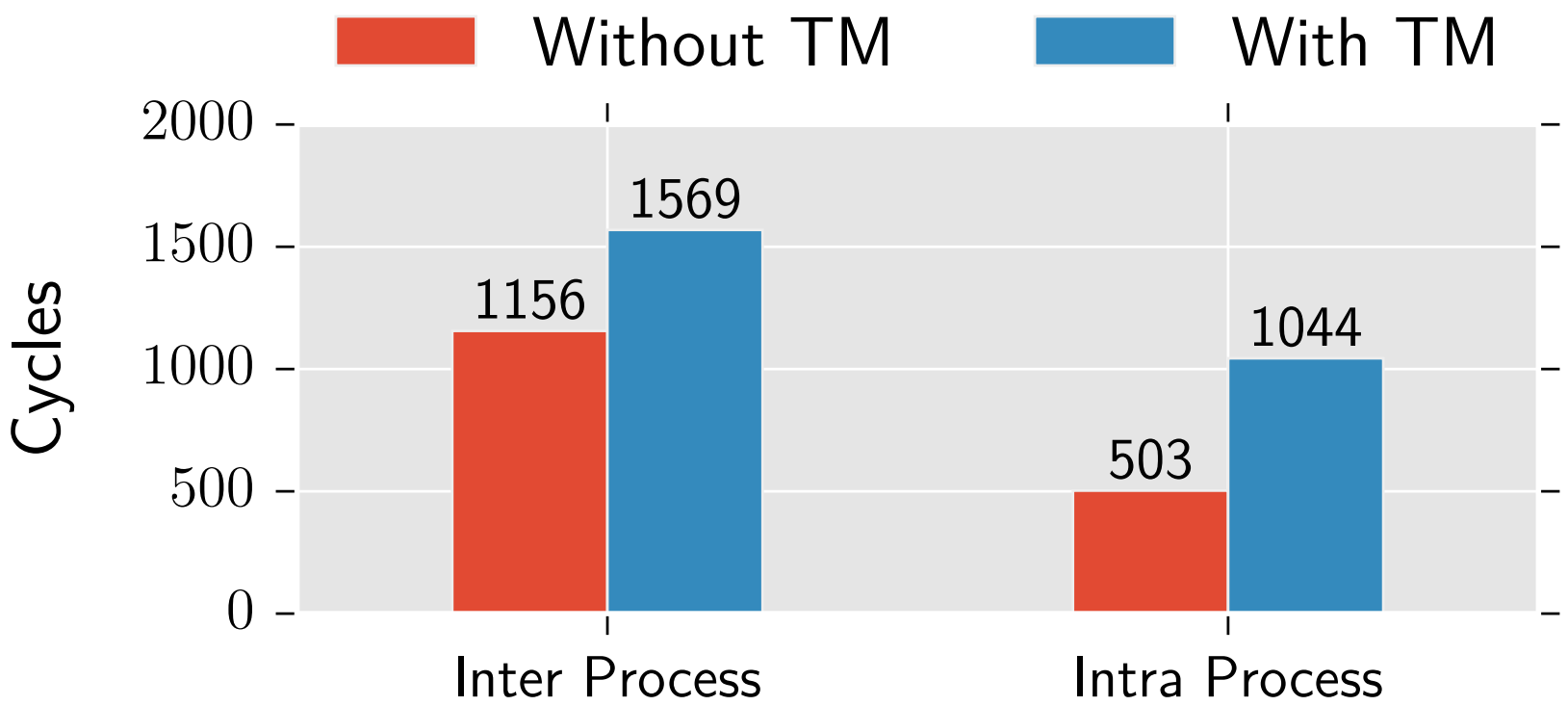
All complexity analyses assume that transactions itself do not increase the complexity.

- only valid if transactions provide *progress guarantees*
- TSX does not provide such guarantee

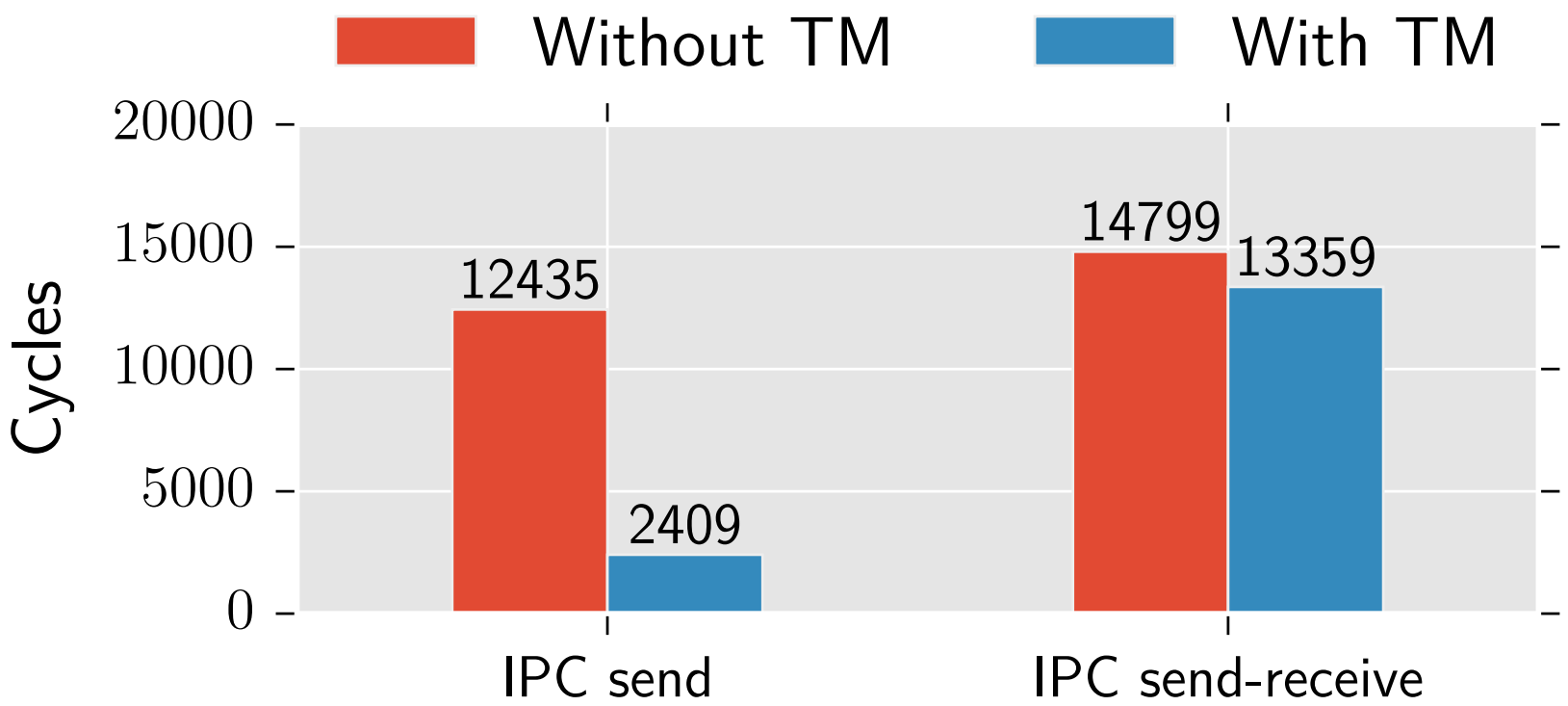
To overcome this limitation, we retry transactions if they are aborted.

direct	1 retry	2 retries	> 2 retries
99.9999%	$8.6 * 10^{-5}\%$	$1.05 * 10^{-7}\%$	0%

Raw kernel performance for *Core-Local-IPC* on an Intel Haswell i4770.



Raw kernel performance for *X-Core-IPC* on an Intel Haswell i4770.



Evaluation

Performance

Using transactions during the IPC operations influences the performance of the system call.

Evaluation

Performance

Using transactions during the IPC operations influences the performance of the system call.

The *Core-Local-IPC* operation performance suffers.

- housekeeping for the transactions introduces overhead
- overall slowdown is significant

Evaluation

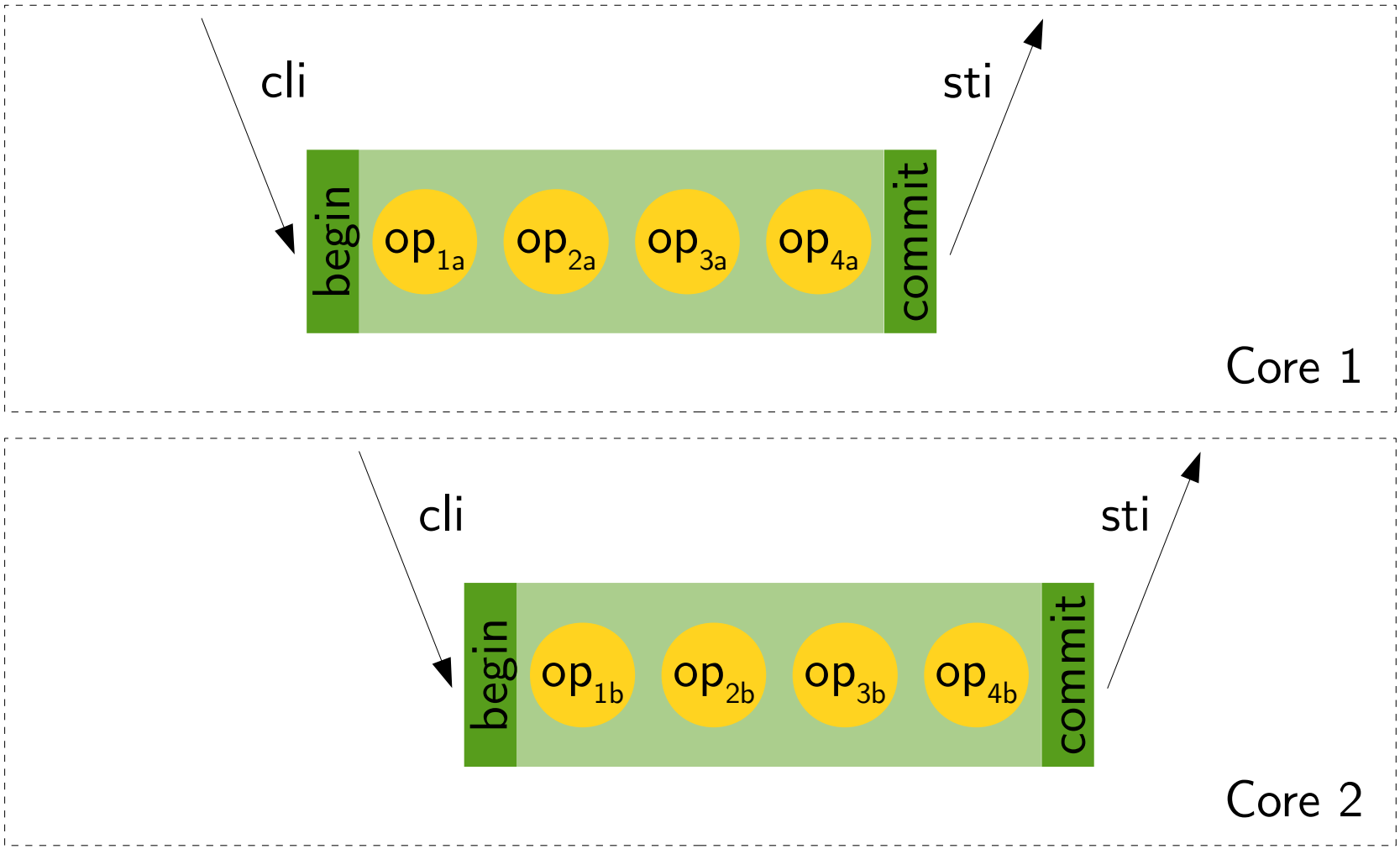
Performance

Using transactions during the IPC operations influences the performance of the system call.

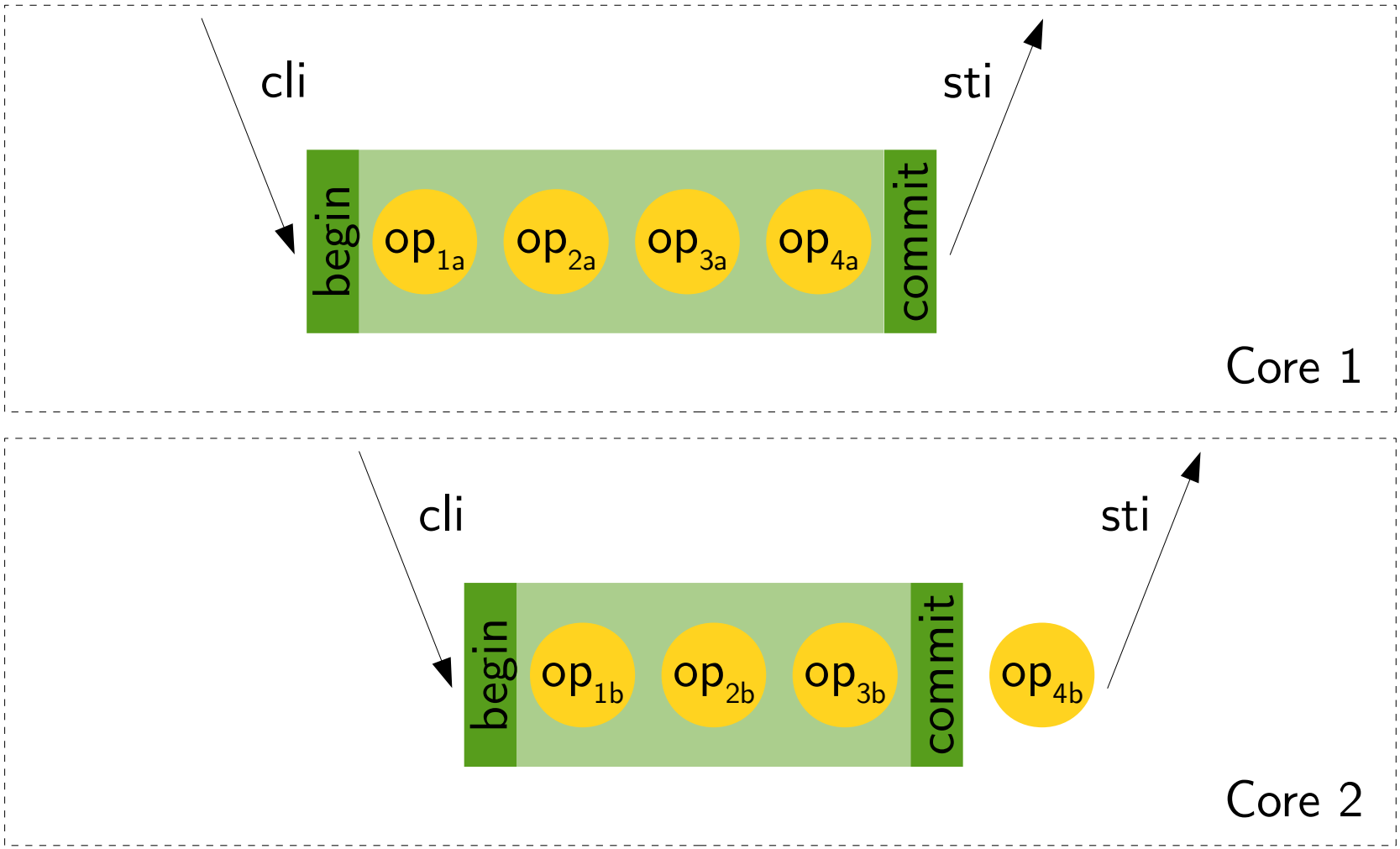
For the *X-Core-IPC* operation the performance increases.

- removal of expensive IPI synchronization speeds up the operation
- detailed evaluation not possible because we triggered the famous bug in the TSX implementation

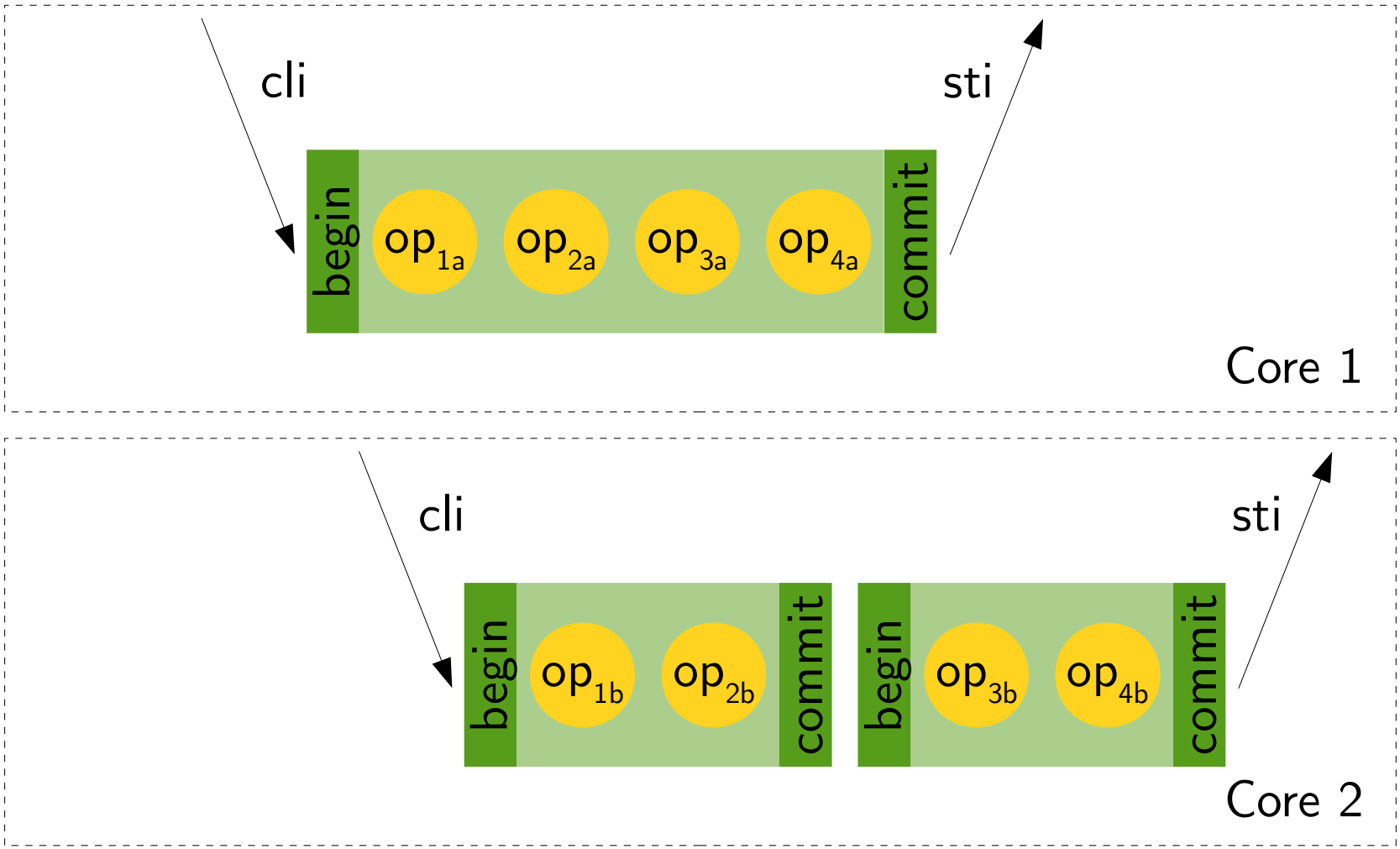
Consequences for the Verification



Consequences for the Verification



Consequences for the Verification



Conclusion

- transactional memory can simplify verification of programs
- possible to build kernels using transactional memory

Conclusion

- transactional memory can simplify verification of programs
- possible to build kernels using transactional memory

First steps towards verifiable transactional multicore kernels are done.