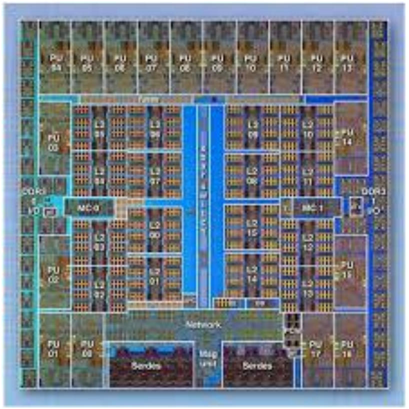
Low-Overhead Software Transactional Memory with Progress Guarantees and Strong Semantics

Minjia Zhang,

Jipeng Huang, Man Cao, Michael D. Bond

Do We Need Efficient STM?

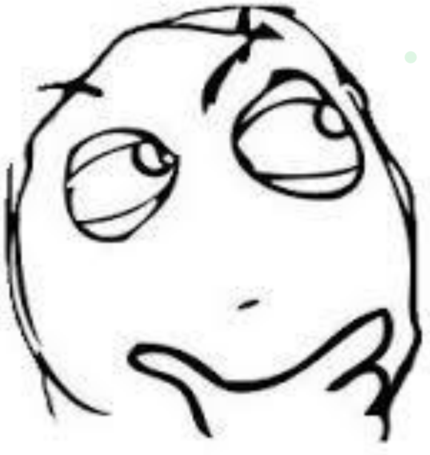
Problem Solved!



Blue Gene/Q

Problem Solved?

HTM is limited…



Problem Solved?

Best-effort HTM: no completion guarantee[1]
Performance penalty: short transactions[2]
Language-level support for atomic blocks: STM fallback

```
atomic {
    from.balance -= amount;
    to.balance += amount;
}
```

transaction

[1] I. Calciu et al. Invyswell: A Hybrid Transactional Memory for Haswell's Restricted Transactional Memory. In PACT, 2014.

[2] R. M. Yoo et al. Performance Evaluation of Intel Transactional Synchronization Extensions for High-Performance Computing. In SC, 2013.

Software Transactional Memory Is Slow

Existing STMs add high overhead [1,2,3]

[1] C. Cascaval et al. Software Transactional Memory: Why Is It Only a Research Toy? In CACM, 2008
[2] A. Dragojevi´c, et al. Why STM Can Be More than a Research Toy. In CACM, 2011
[3] R. M. Yoo et al. Kicking the Tires of Software Transactional Memory: Why the Going Gets Tough. In SPAA, 2008.

Software Transactional Memory Is Slow

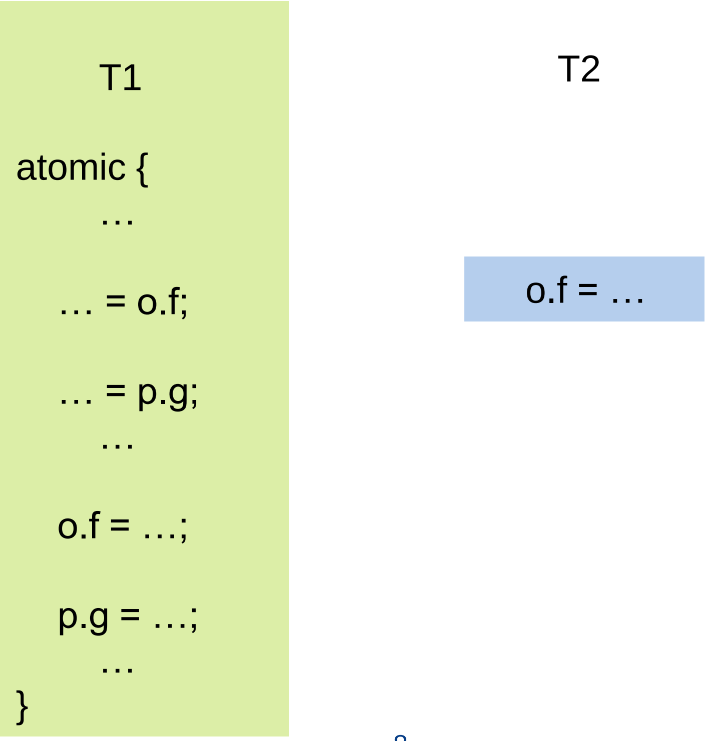
Existing STMs add high overhead [1,2,3]

Related challenges: scalability, progress guarantees, strong semantics

[1] C. Cascaval et al. Software Transactional Memory: Why Is It Only a Research Toy? In CACM, 2008
[2] A. Dragojevi´c, et al. Why STM Can Be More than a Research Toy. In CACM, 2011
[3] R. M. Yoo et al. Kicking the Tires of Software Transactional Memory: Why the Going Gets Tough. In SPAA, 2008.

Challenge

Expensive to detect conflicts



T1

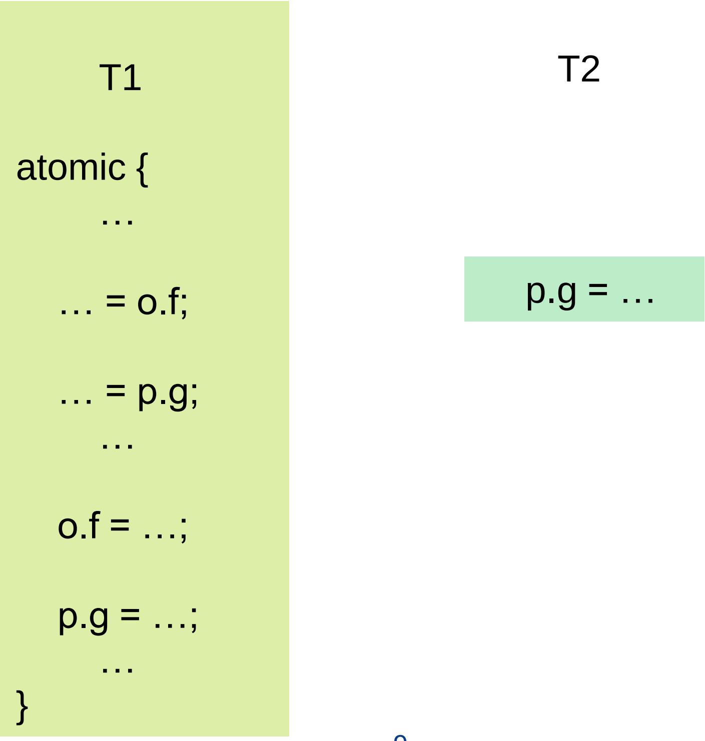
```
atomic {
   …

  … = o.f;

  … = p.g;
   …

  o.f = …;

  p.g = …;
   …
}
```

T2

```
o.f = …
```

Challenge

Expensive to detect conflicts



T1

```
atomic {
    …

  … = o.f;

  … = p.g;
    …

  o.f = …;

  p.g = …;
    …
}
```

T2

```
p.g = …
```

Challenge

Expensive to detect conflicts

T1

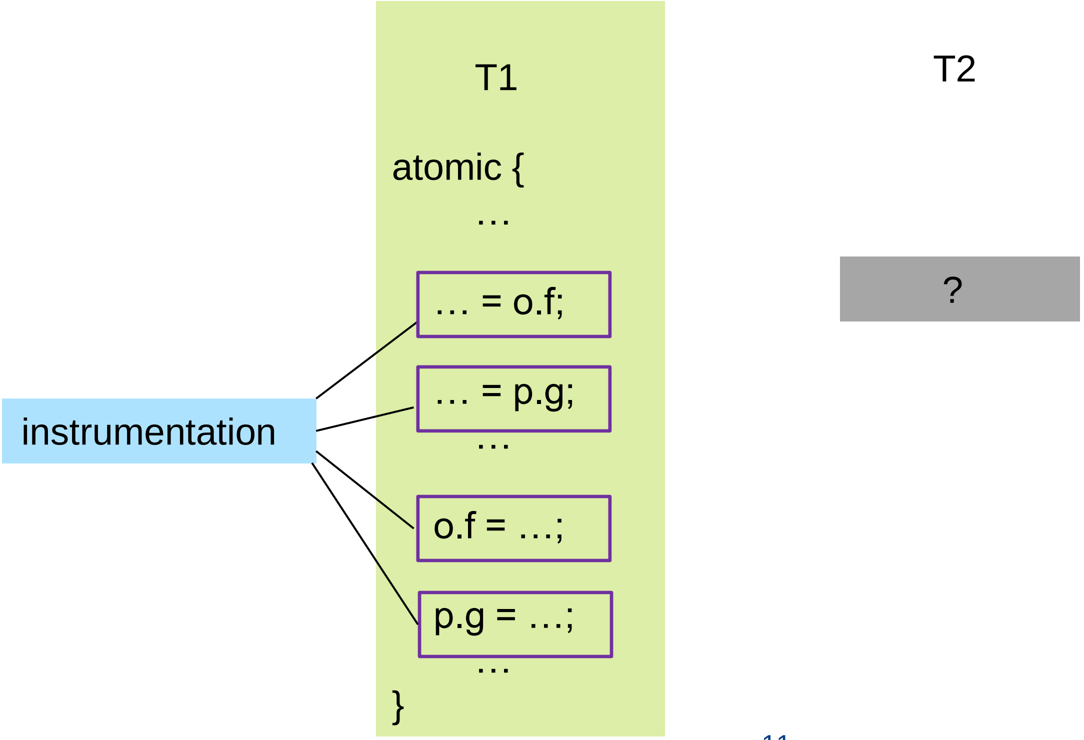
```
atomic {
      …

   … = o.f;

   … = p.g;
      …

   o.f = …;

   p.g = …;
      …
}
```

T2

```
t.k = …
```

Challenge

Expensive to detect conflicts



T1

```
atomic {
      …
   … = o.f;
   … = p.g;
      …
   o.f = …;
   p.g = …;
      …
}
```

instrumentation

T2

?

LarkTM

LarkTM Contributions

- ❑ Adds very low overhead
- ❑ Achieves good scalability by using a hybrid approach
- ❑ Provides strong progress guarantees
- ❑ Provides strong atomicity

Key Insight

Avoid high instrumentation costs by minimizing instrumentation costs for non-conflicting accesses

LarkTM Design

Per-object biased reader-writer locks[1,2]

+

Eager concurrency control

+

Piggybacking conflict detection and conflict resolution on lock transfers

[1] M. D. Bond et al. Octet: Capturing and Controlling Cross-Thread Dependences Efficiently. In OOSPLA, 2013.
[2] B. Hindman and D. Grossman. Atomicity via Source-to-Source Translation. In MSPC, 2006.

LarkTM Design

Per-object biased reader-writer locks[1,2]

+

Eager concurrency control

+

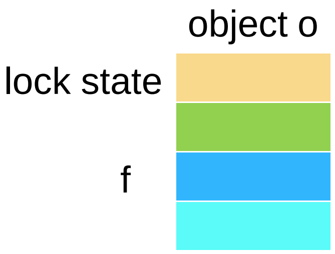
Piggybacking conflict detection and conflict resolution on lock transfers

- **Minimal instrumentation and synchronization** for both transactional and non-transactional non-conflicting accesses
- Does **not release locks** even if transactions commit

[1] M. D. Bond et al. Octet: Capturing and Controlling Cross-Thread Dependences Efficiently. In OOSPLA, 2013.

[2] B. Hindman and D. Grossman. Atomicity via Source-to-Source Translation. In MSPC, 2006.

Biased Locks

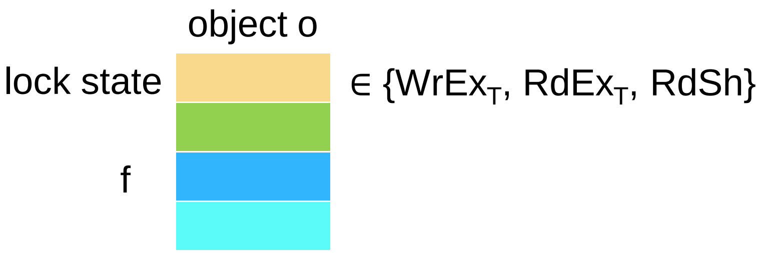


object o

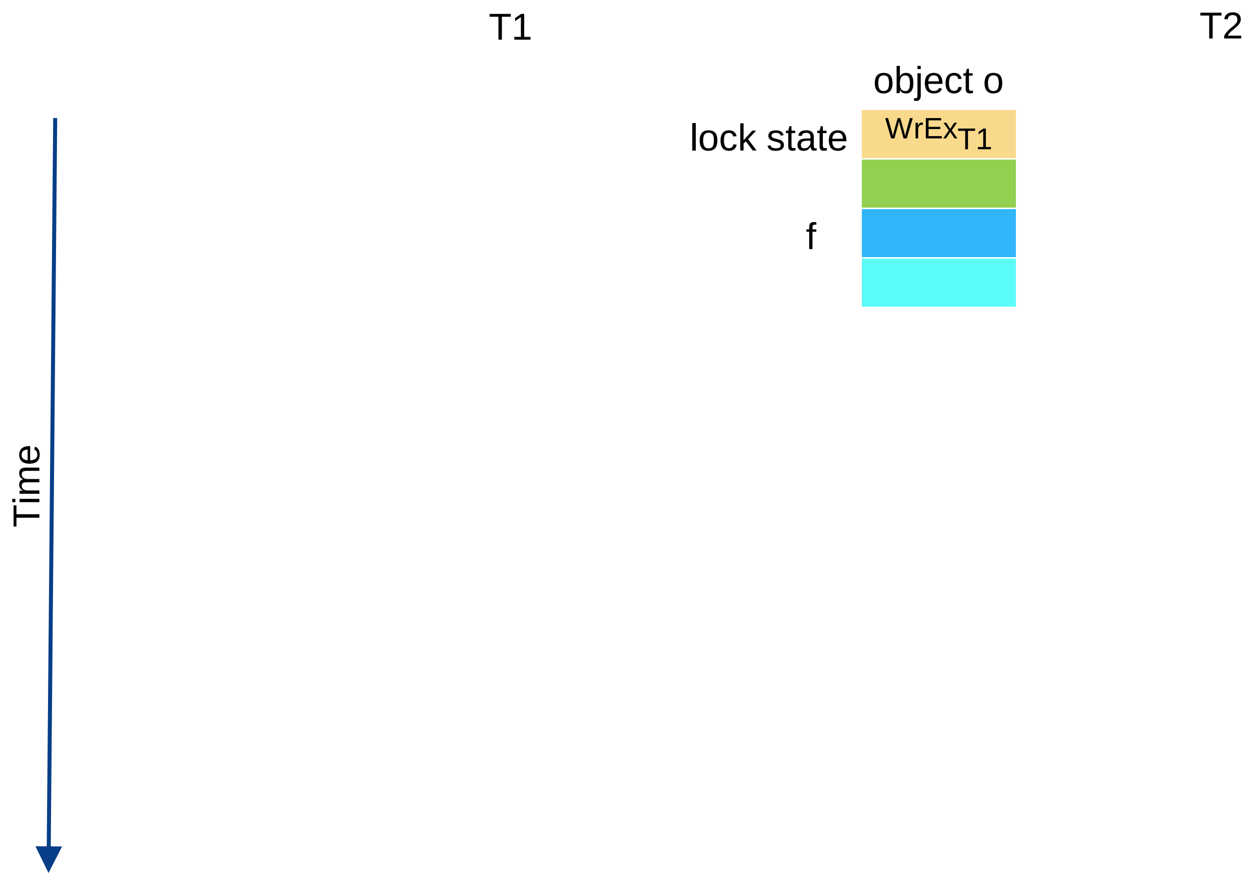
lock state

f

Biased Locks



Multi-thread Execution



T1

T2

object o

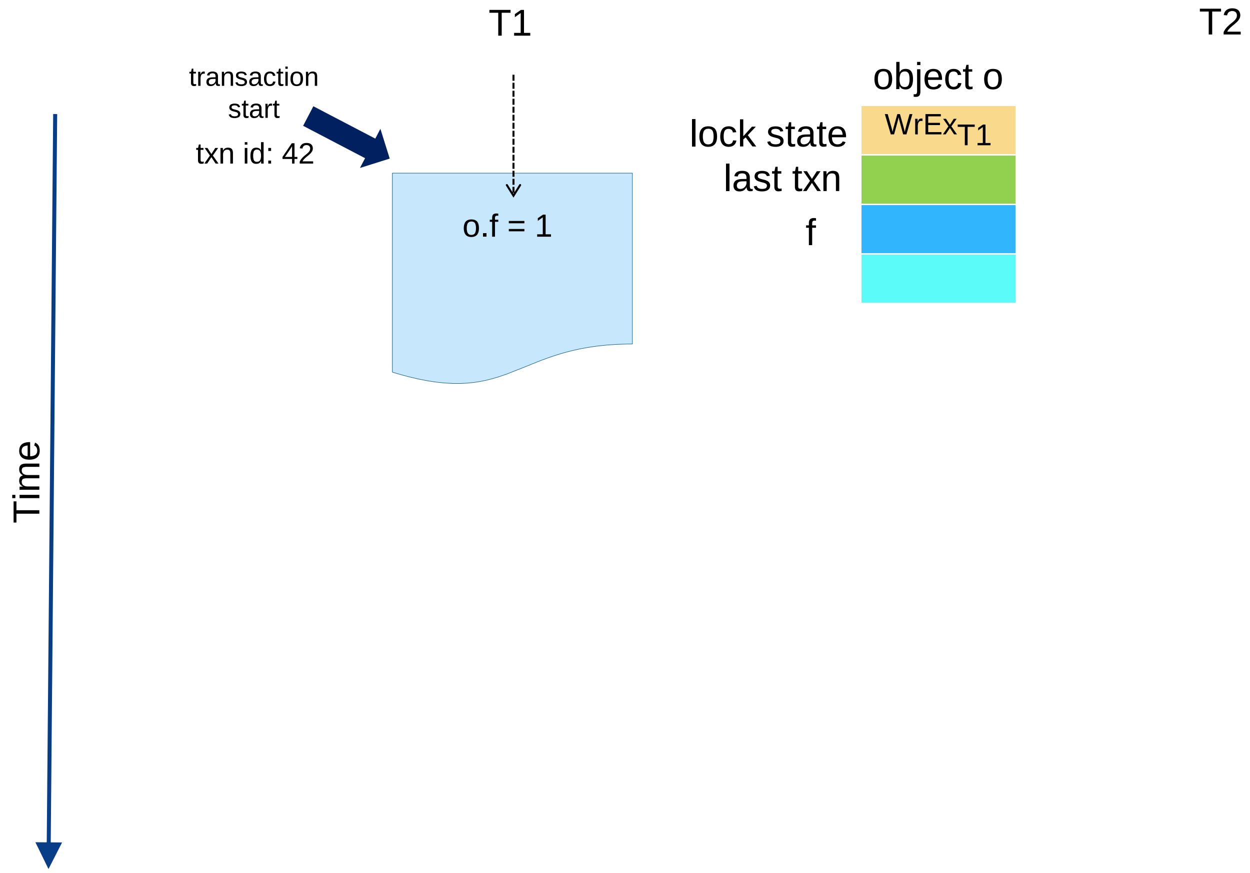
lock state

WrEx_{T1}

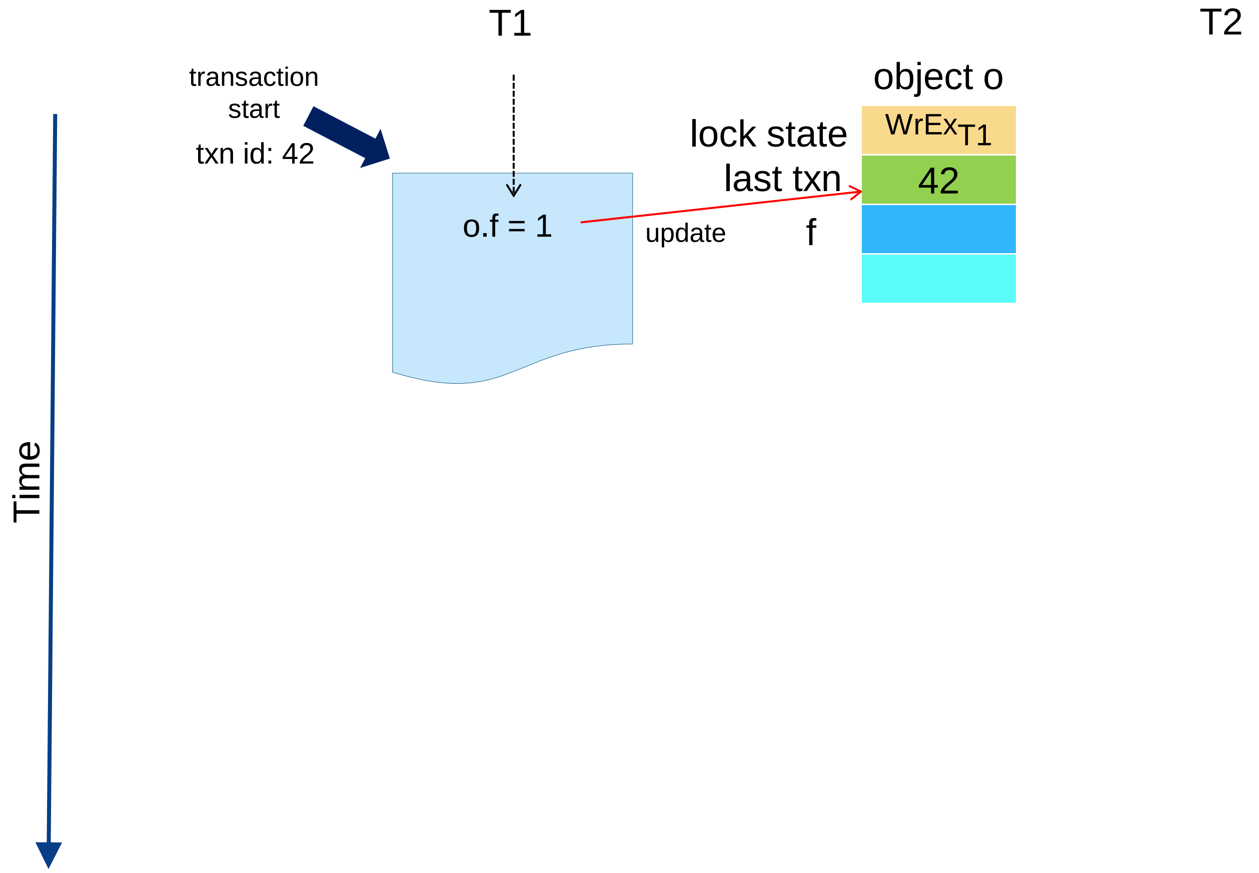
f

Time

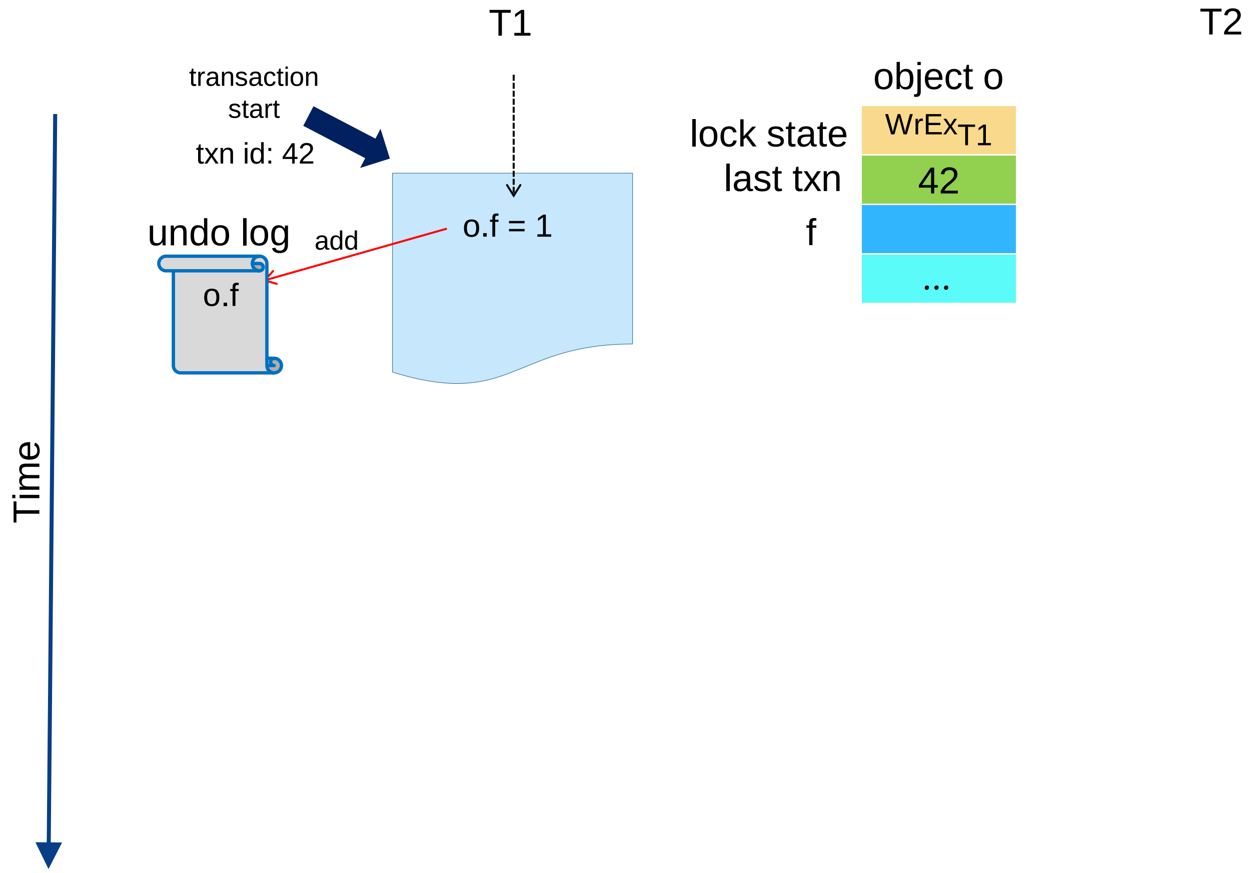
Multi-thread Execution



Multi-thread Execution



Multi-thread Execution



T1

T2

transaction start

txn id: 42

undo log

add

o.f = 1

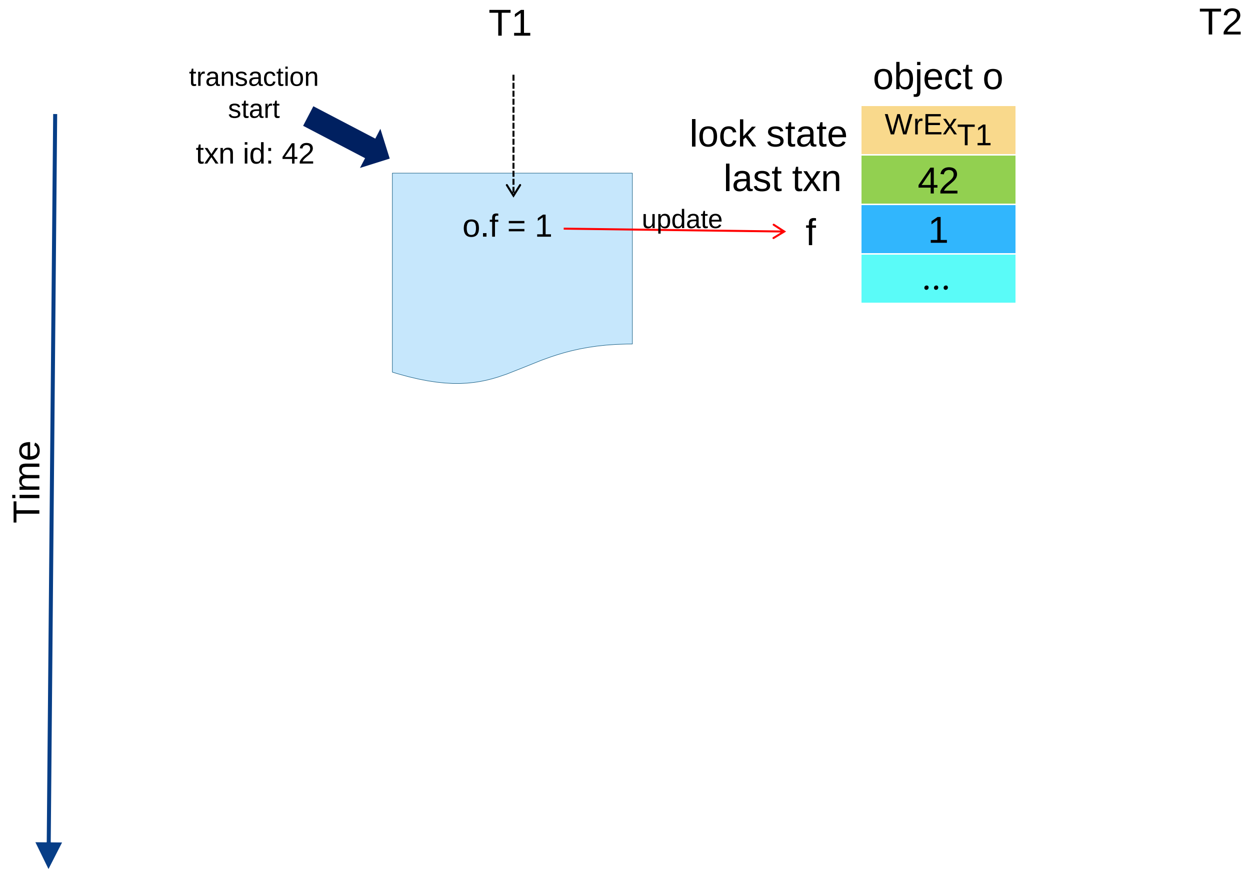
o.f

Time

object o

lock state	WrEx$_{T1}$
last txn	42
f	
	…

Multi-thread Execution



T1

T2

transaction start

txn id: 42

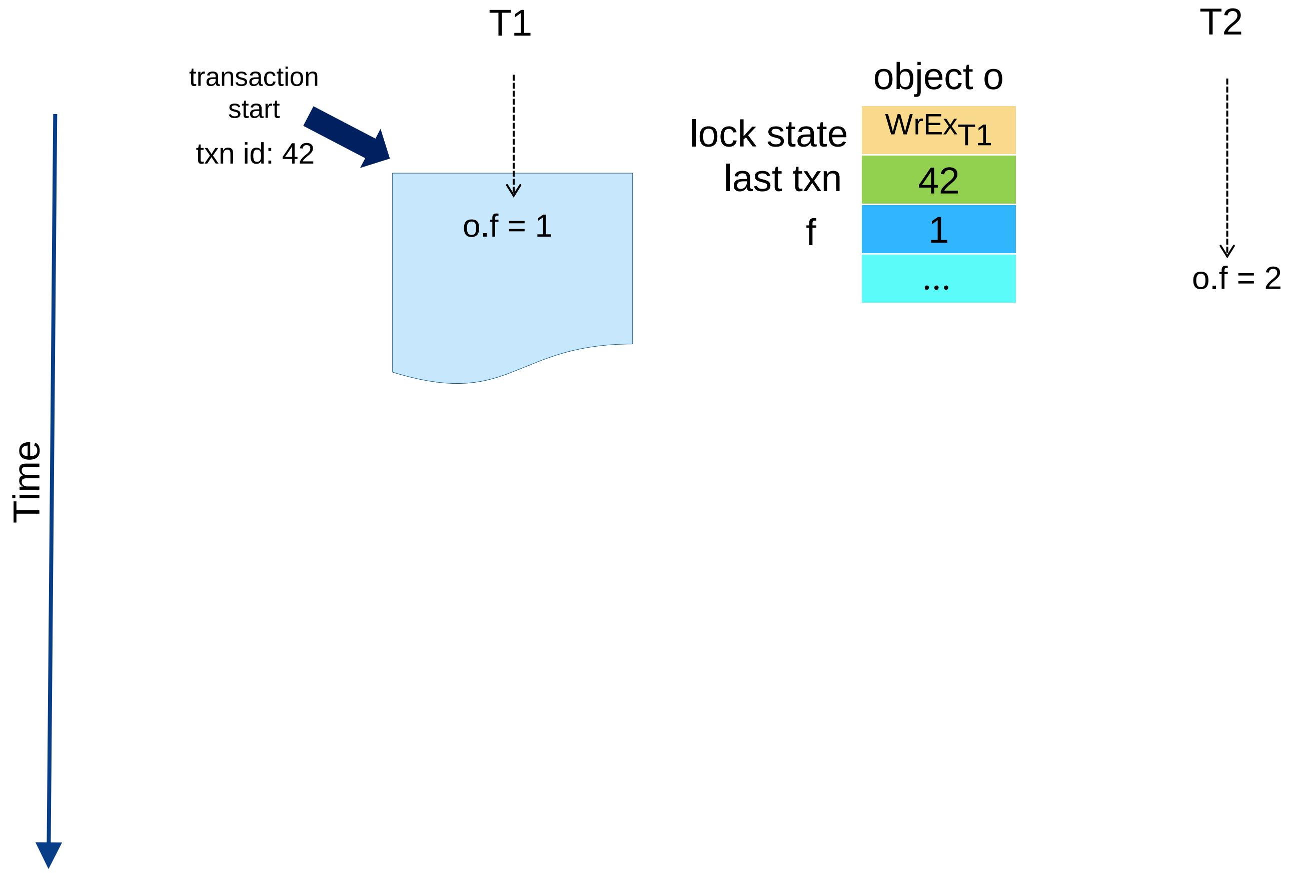
o.f = 1

update

object o

lock state	$WrEx_{T1}$
last txn	42
f	1
	…

Multi-thread Execution



T1

T2

transaction start

txn id: 42

o.f = 1

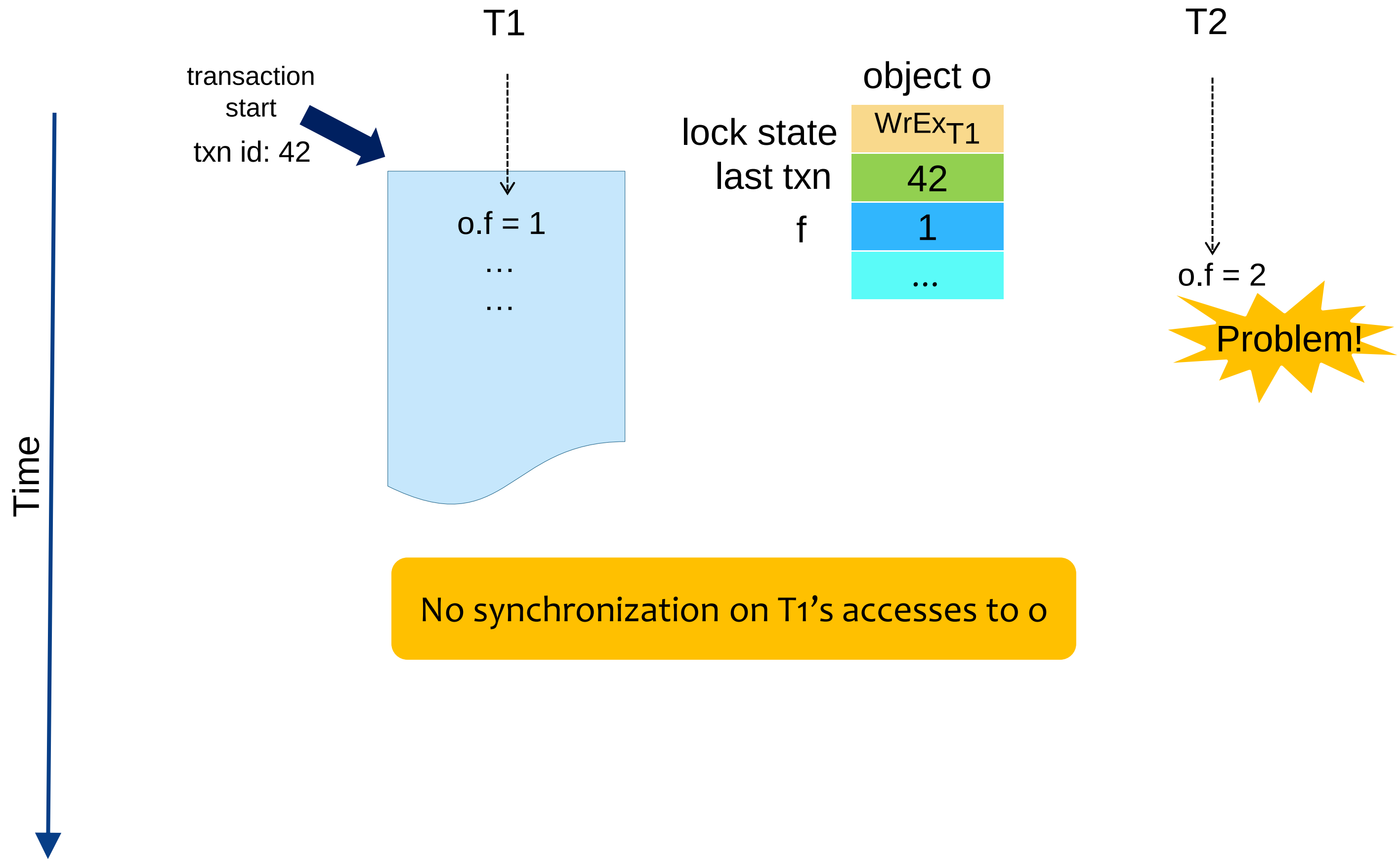
o.f = 2

Time

object o

lock state	WrEx_{T1}
last txn	42
f	1
	...

Multi-thread Execution



T1

T2

transaction start

txn id: 42

o.f = 1

…

…

object o

lock state WrEx_{T1}

last txn 42

f 1

...

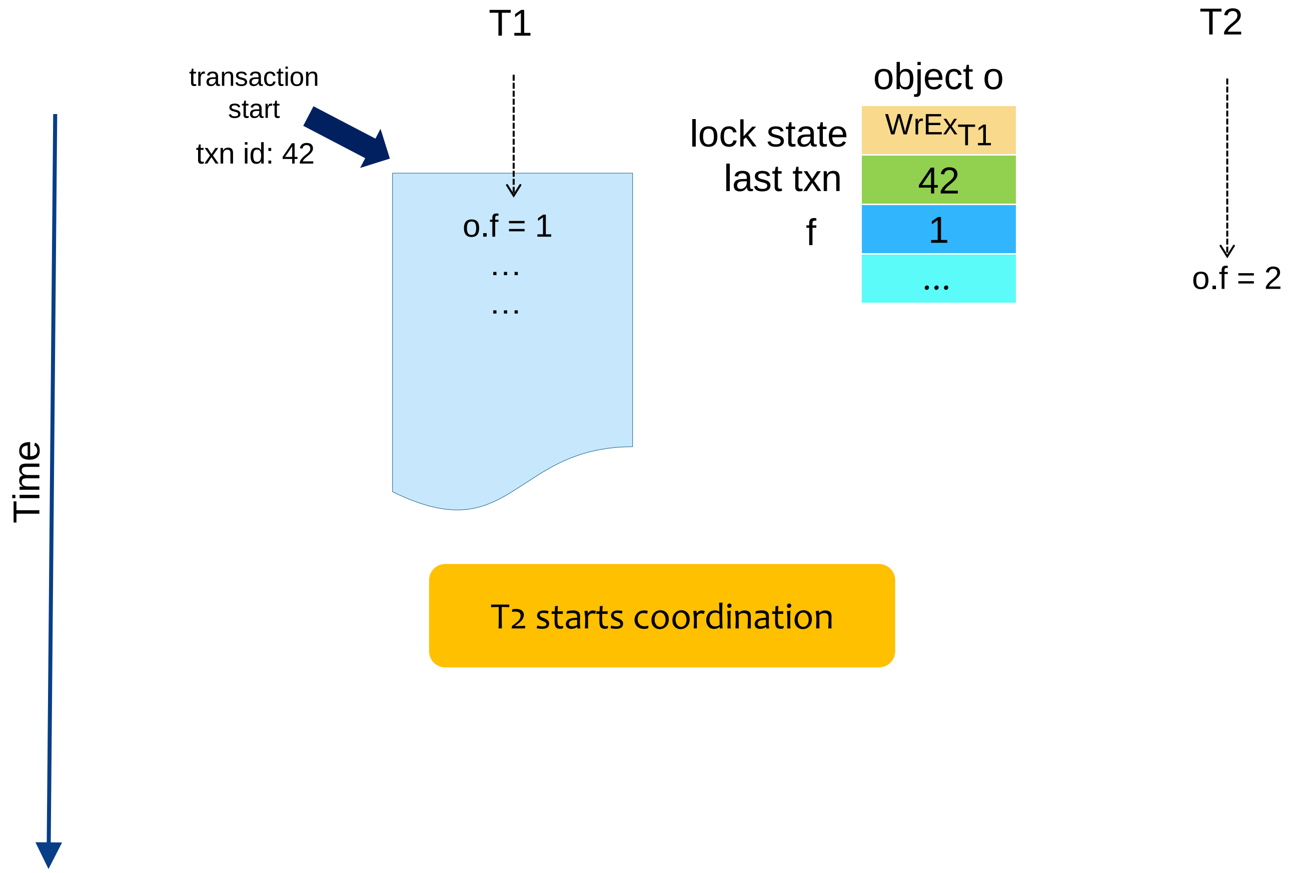
o.f = 2

Problem!

Time

No synchronization on T1's accesses to o

Multi-thread Execution



T1

T2

transaction start

txn id: 42

o.f = 1

…

…

object o

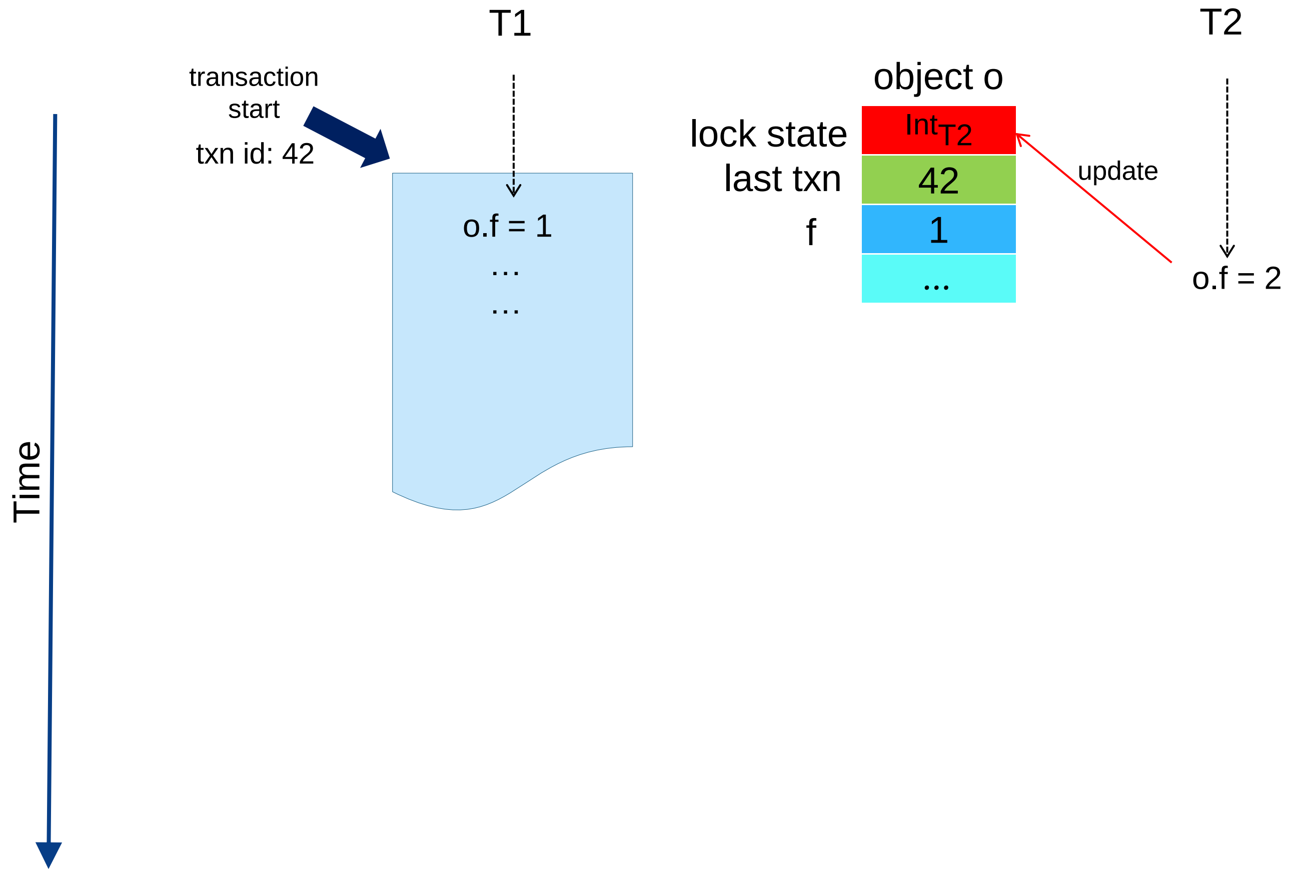
lock state	WrEx_{T1}
last txn	42
f	1
	...

o.f = 2

Time

T2 starts coordination

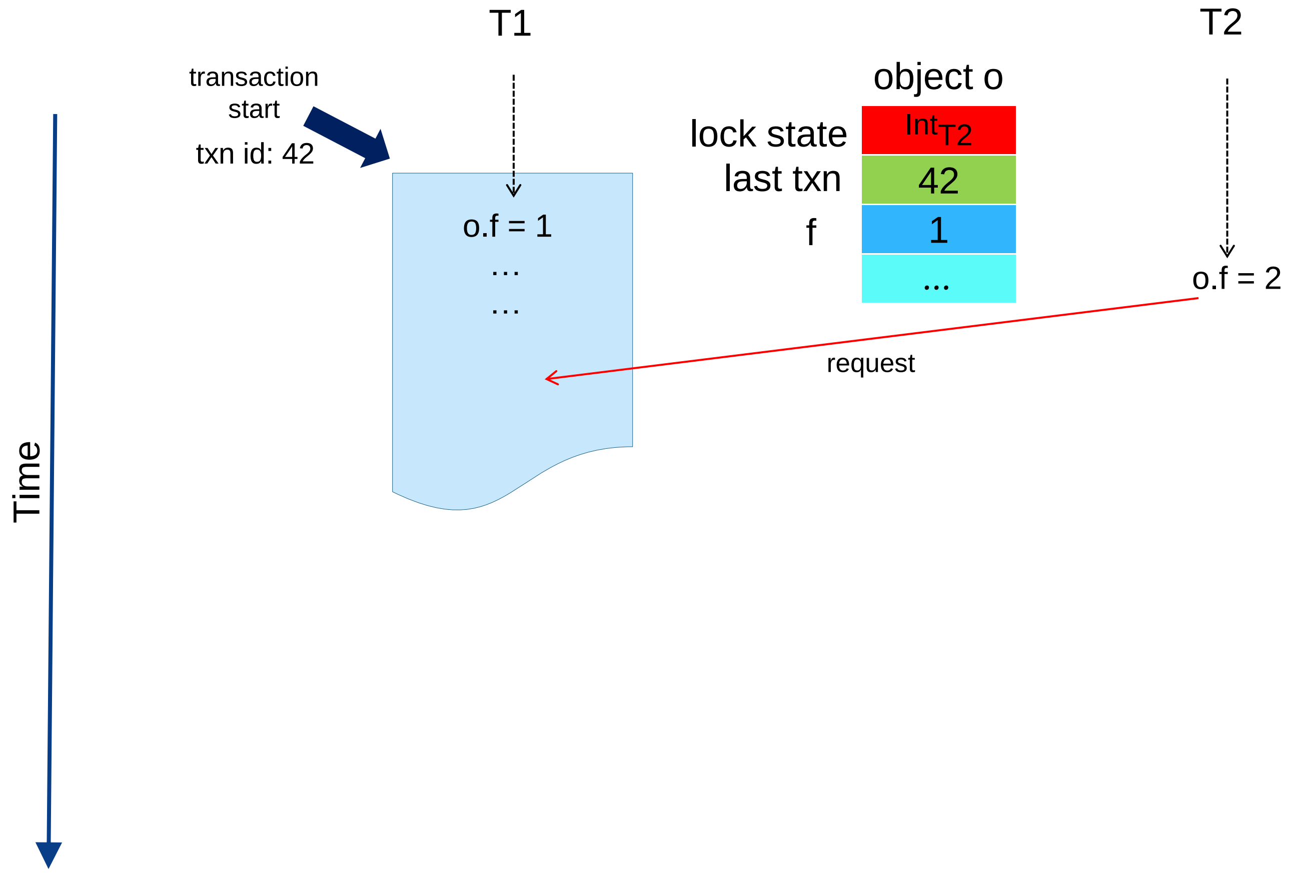
Coordination



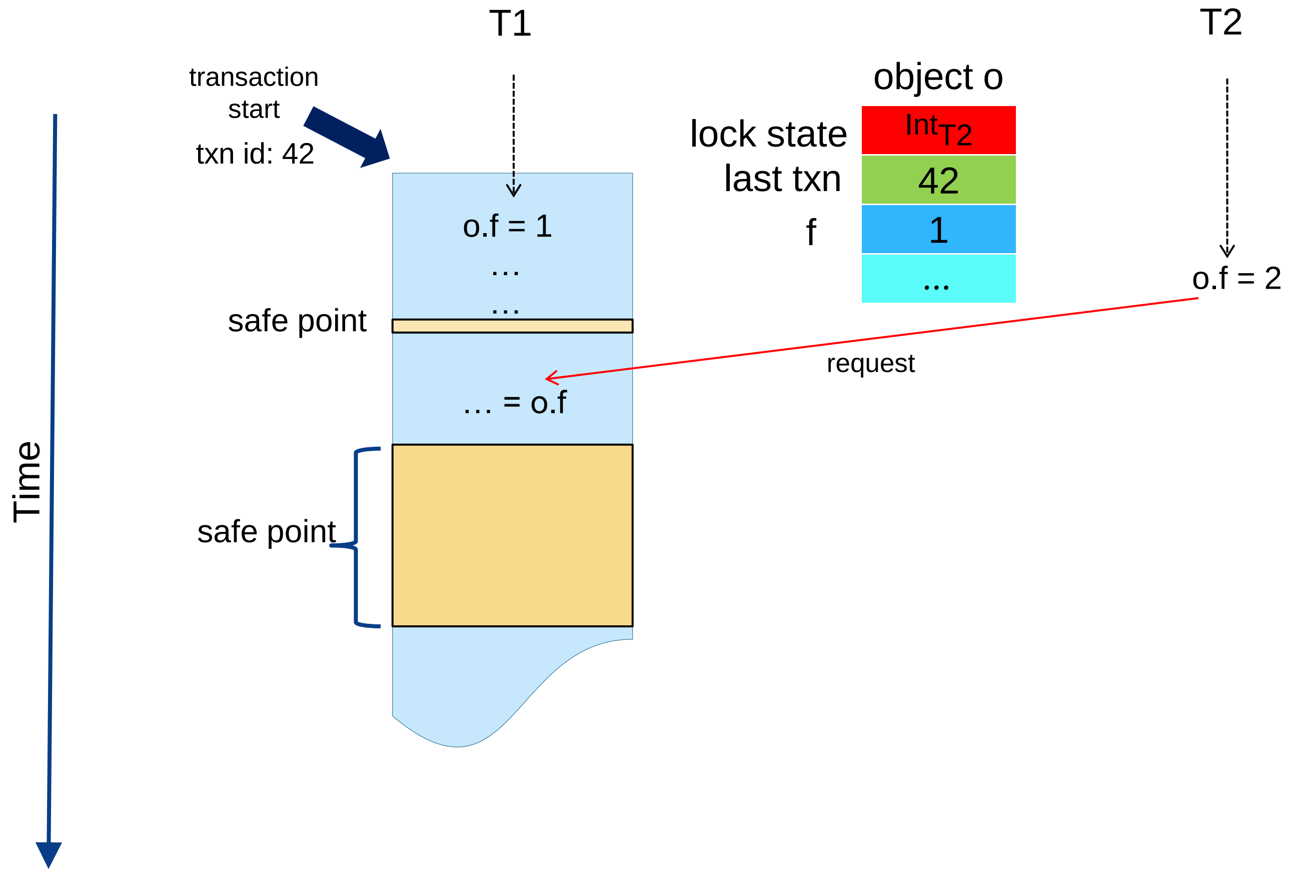
object o

lock state	Int_{T2}
last txn	42
f	1
	...

Coordination



Coordination



T1

T2

transaction start

txn id: 42

object o

lock state	Int_{T2}
last txn	42
f	1
	…

o.f = 1

…

…

safe point

o.f = 2

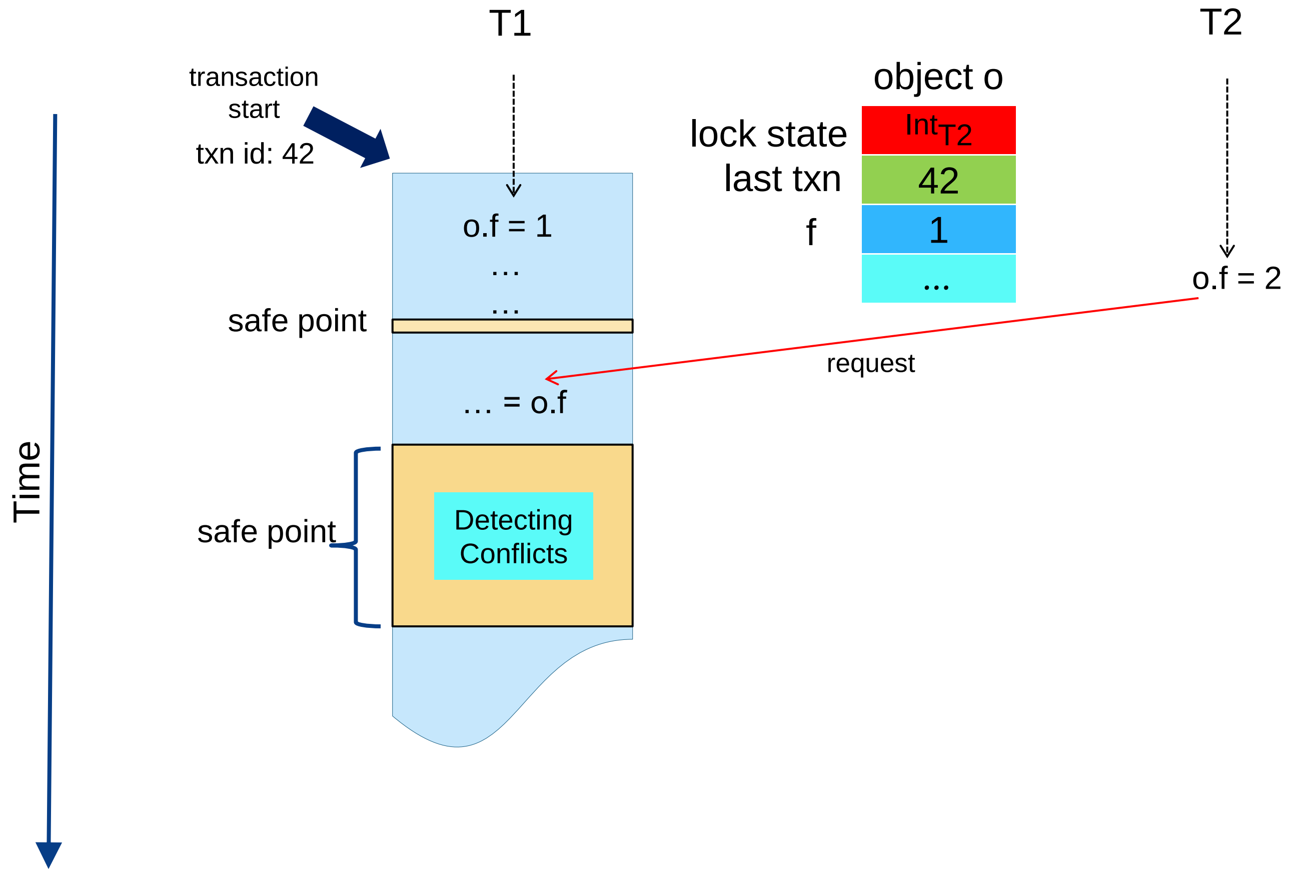
request

… = o.f

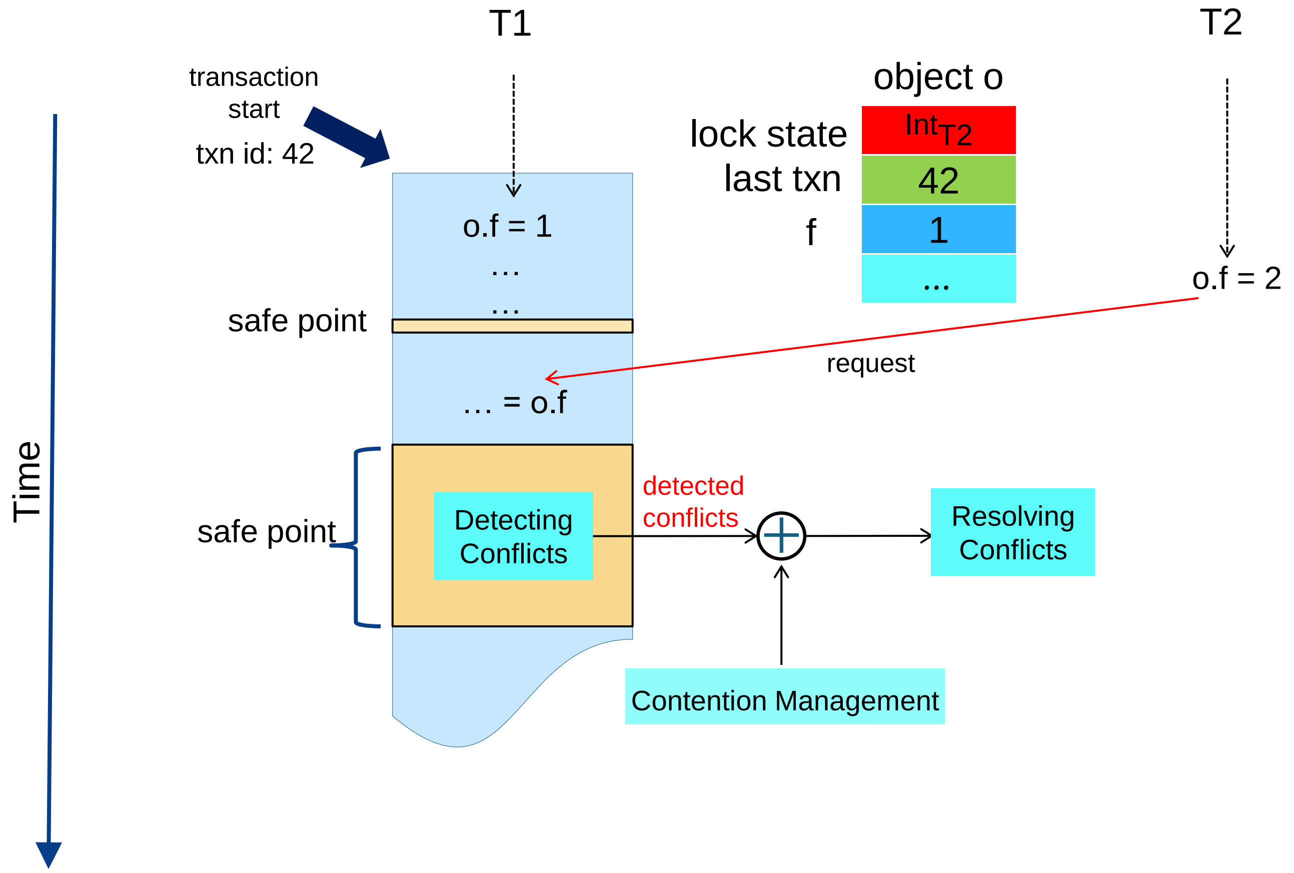
safe point

Time

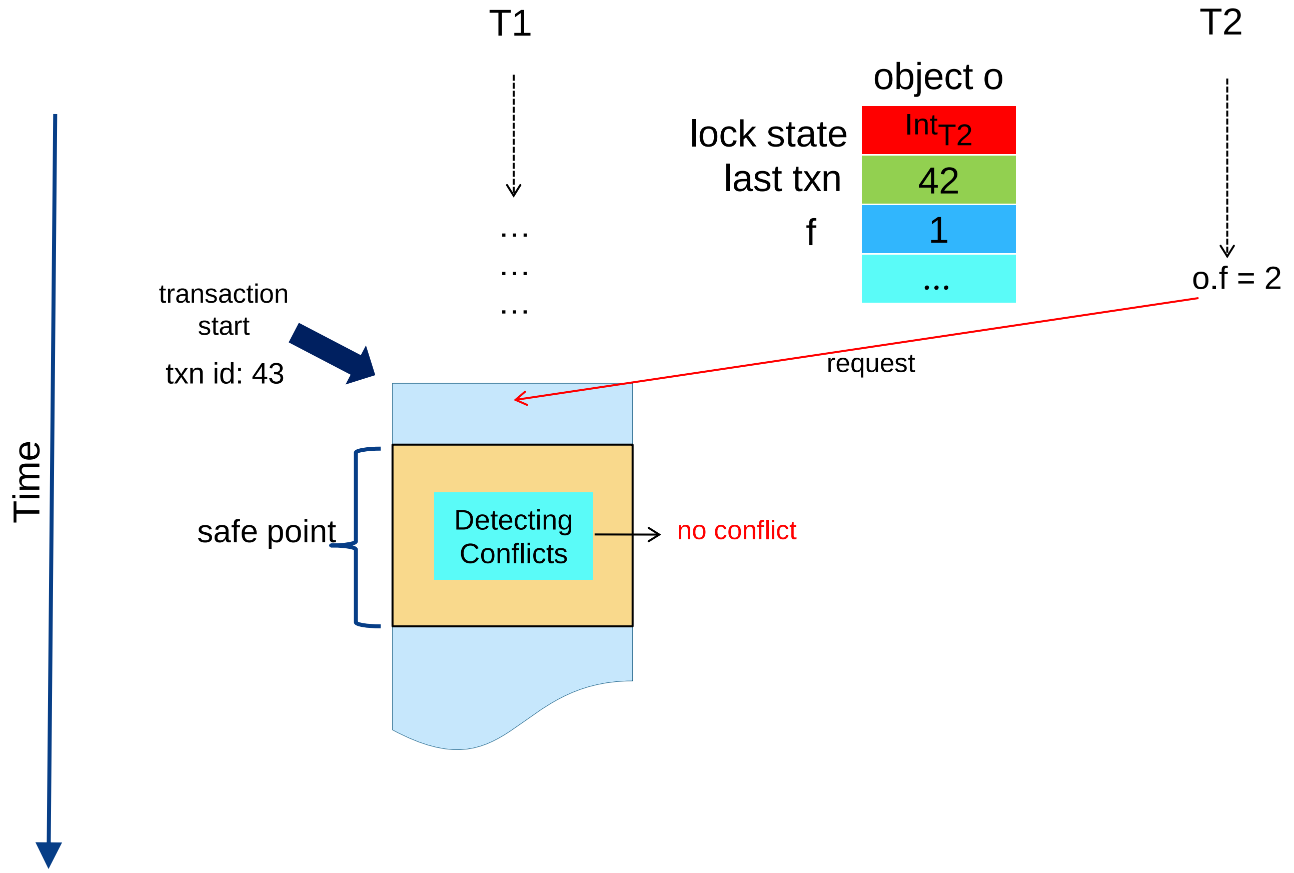
Coordination



A Transactional Conflict



Not A Transactional Conflict



T1

T2

object o

lock state	Int_{T2}
last txn	42
f	1
	...

…
…
…

o.f = 2

request

transaction start

txn id: 43

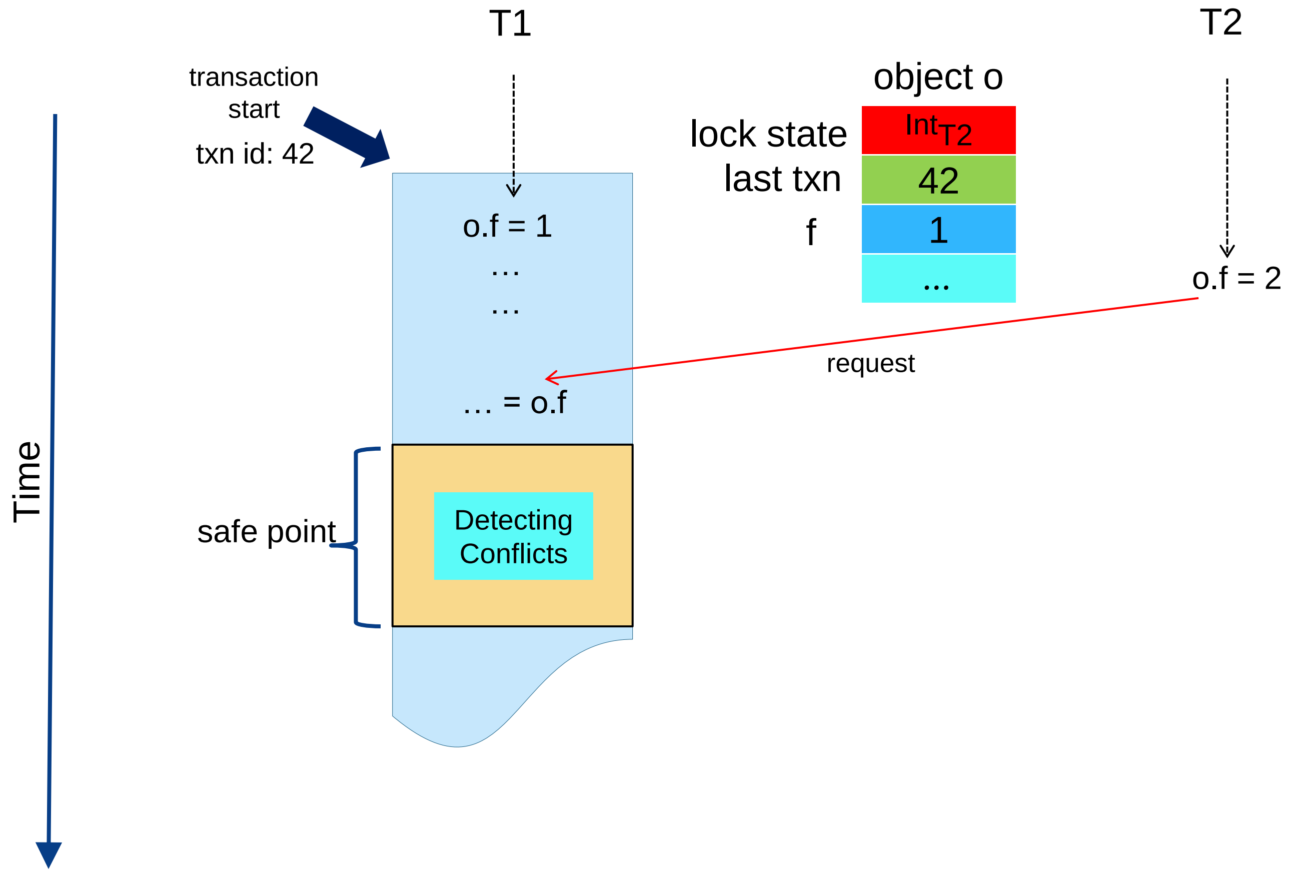
safe point

Detecting Conflicts

no conflict

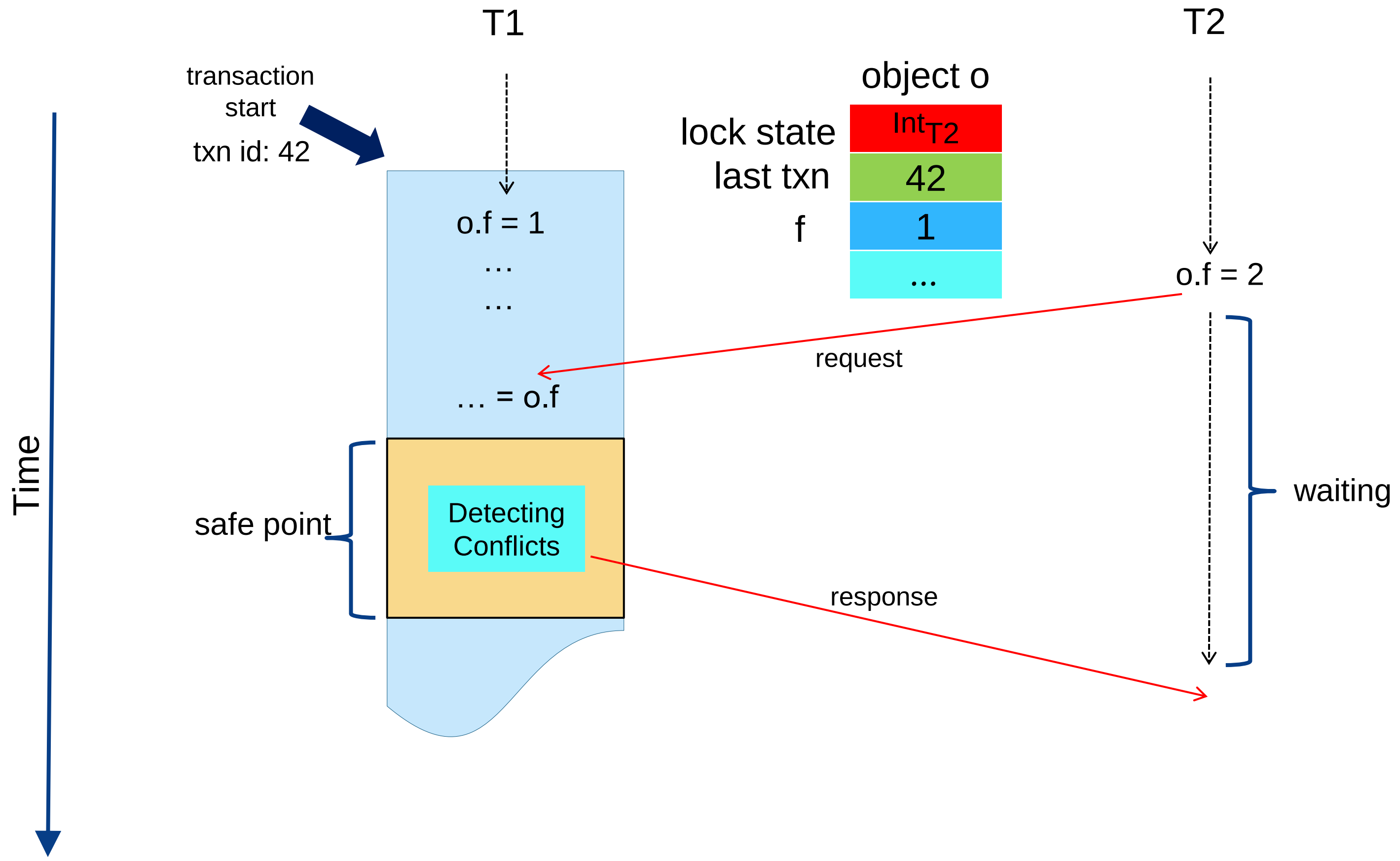
Time

Coordination



object o	
lock state	Int_{T2}
last txn	42
f	1
	...

Coordination



T1

T2

transaction start

txn id: 42

object o

lock state	Int_{T2}
last txn	42
f	1
	...

o.f = 1

…

…

… = o.f

o.f = 2

request

safe point

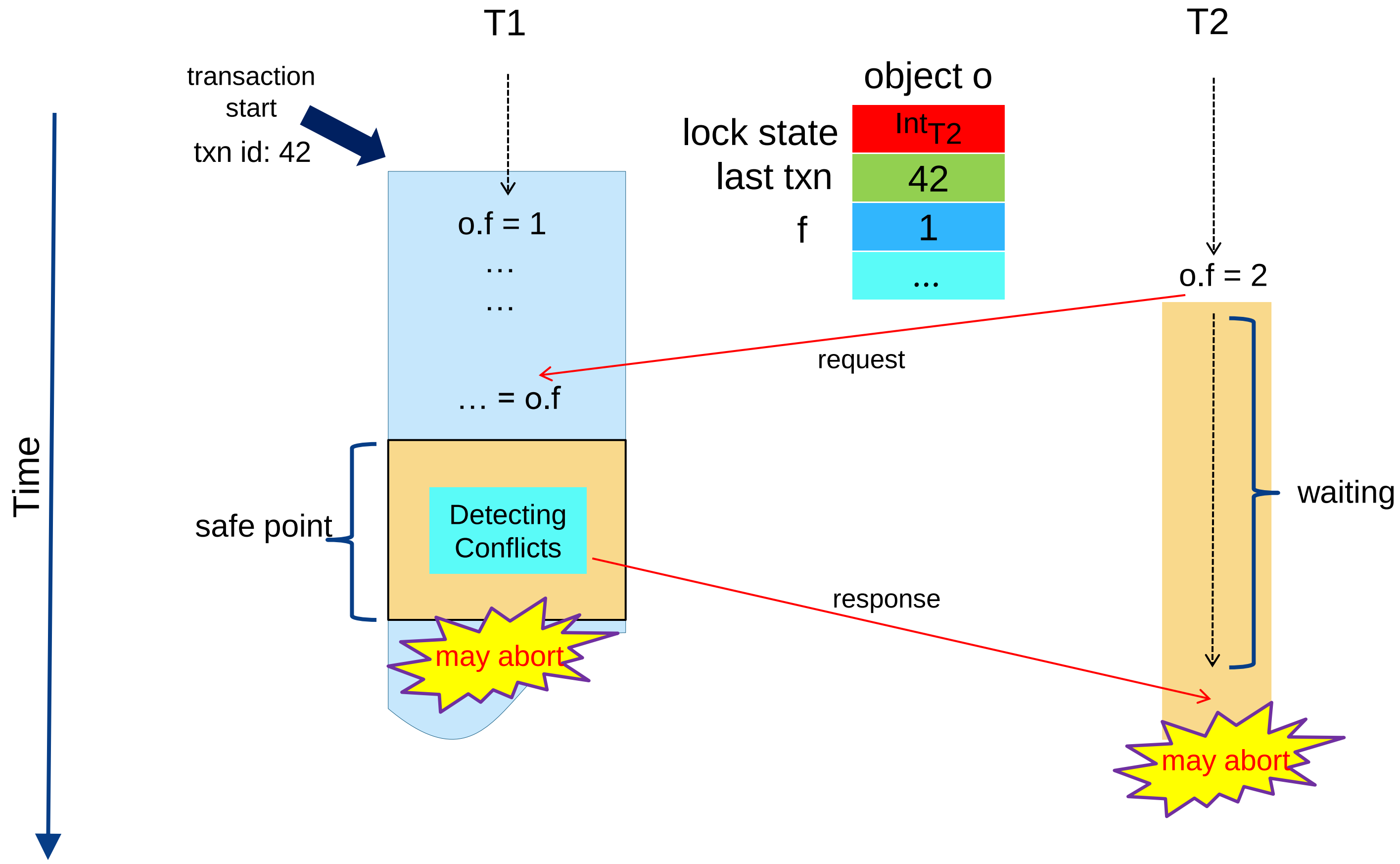
Detecting Conflicts

response

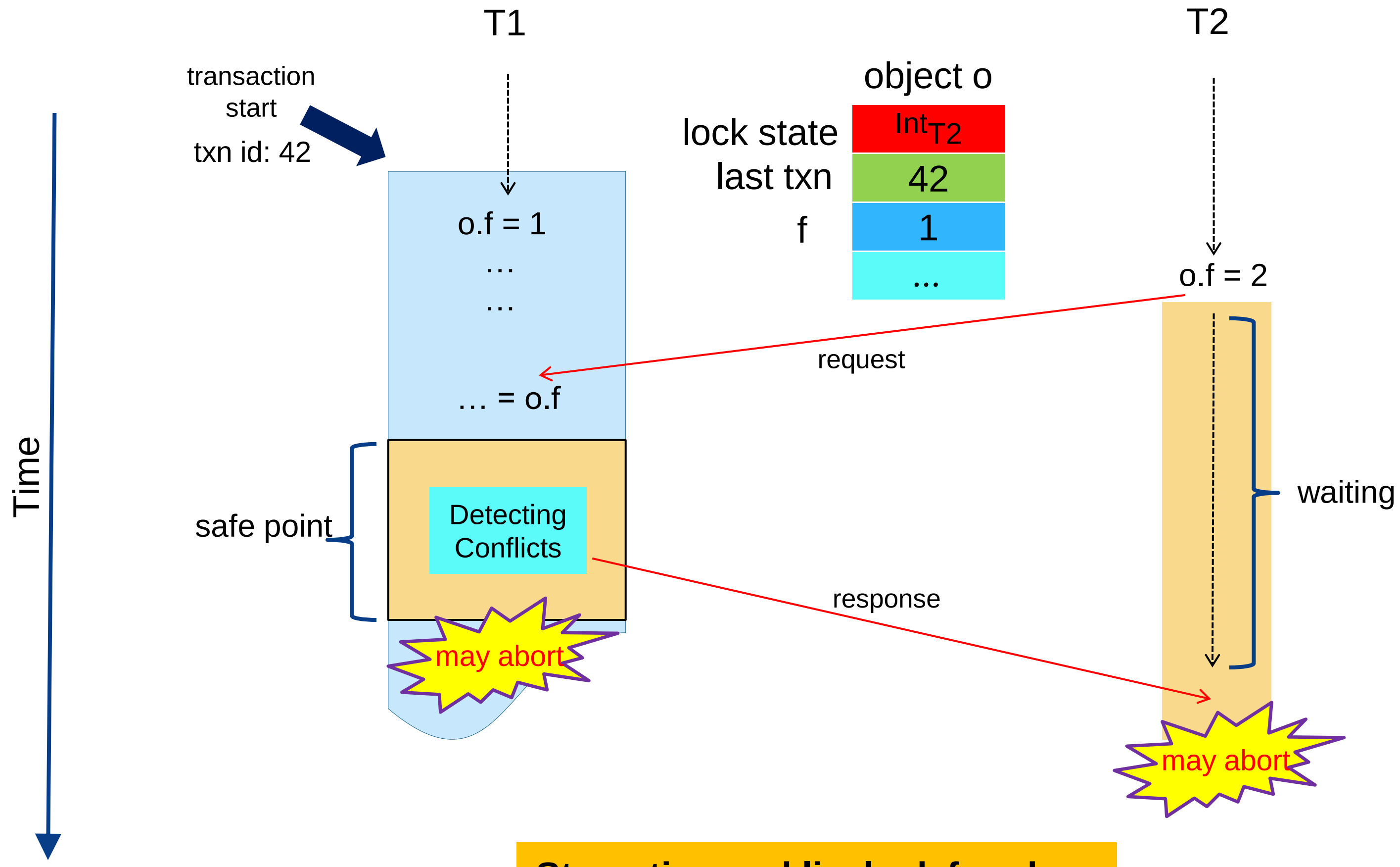
waiting

Time

Strong Progress Guarantees

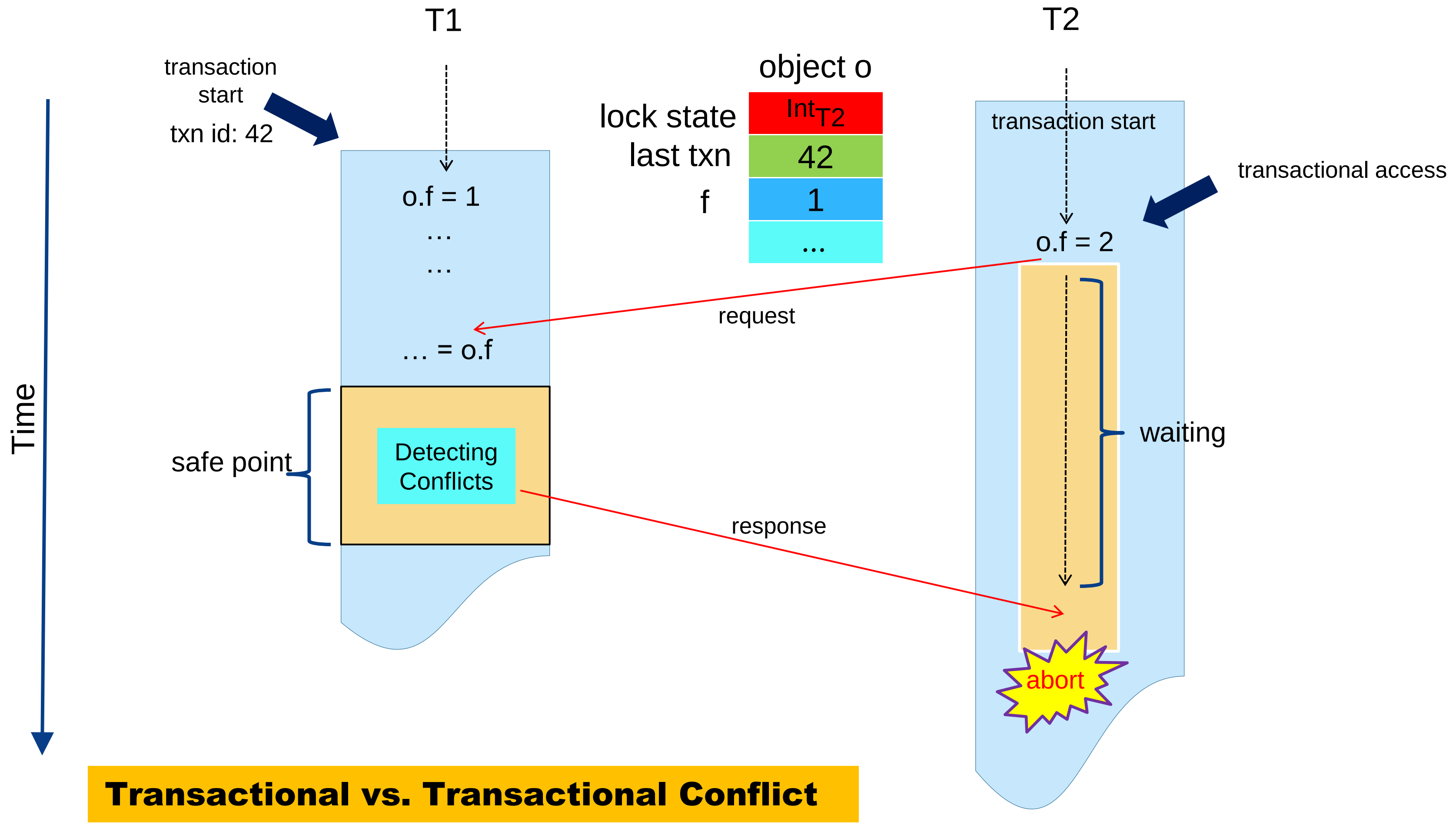


Strong Progress Guarantees



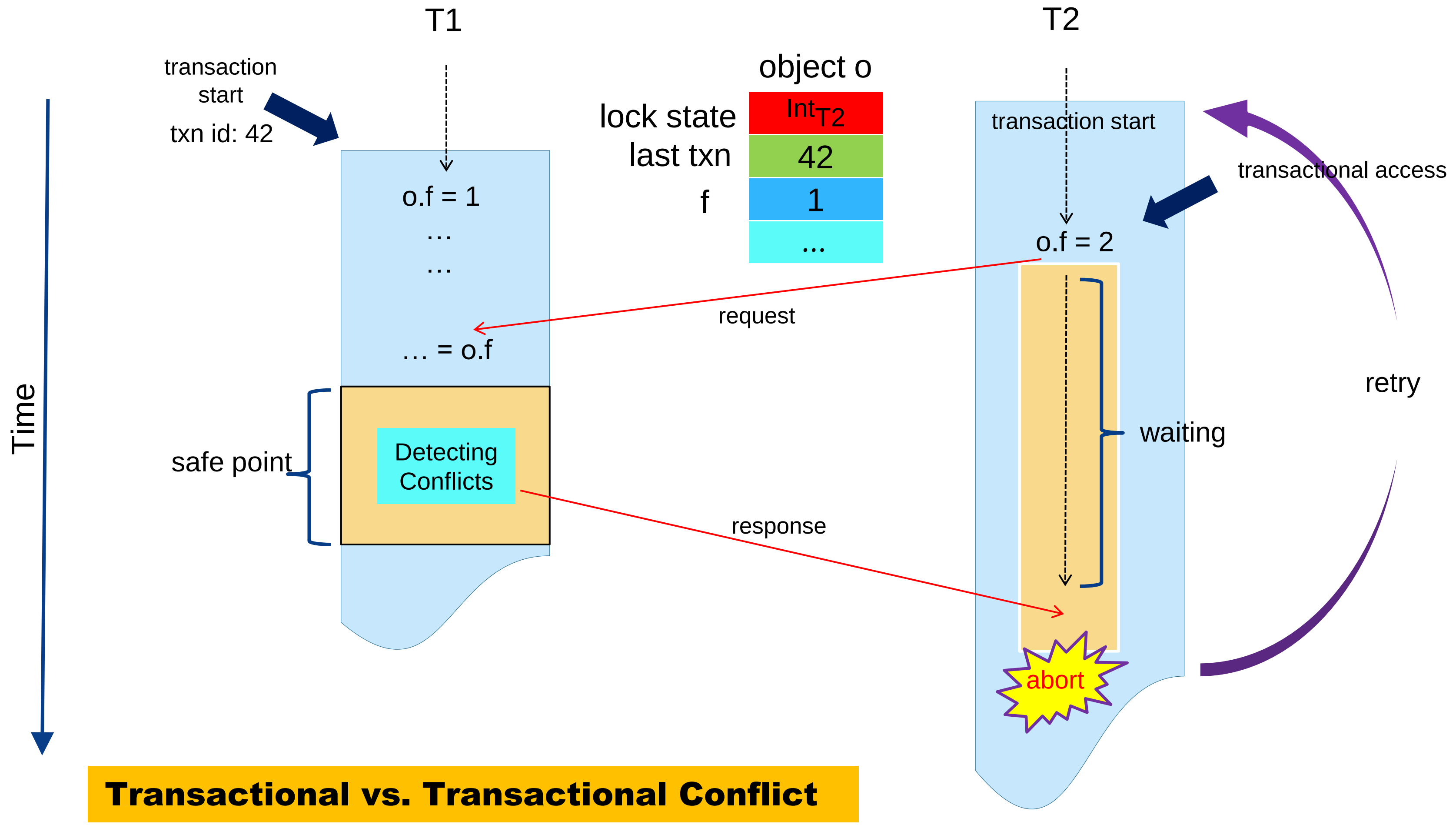
Starvation and livelock freedom

Strong Atomicity Semantics



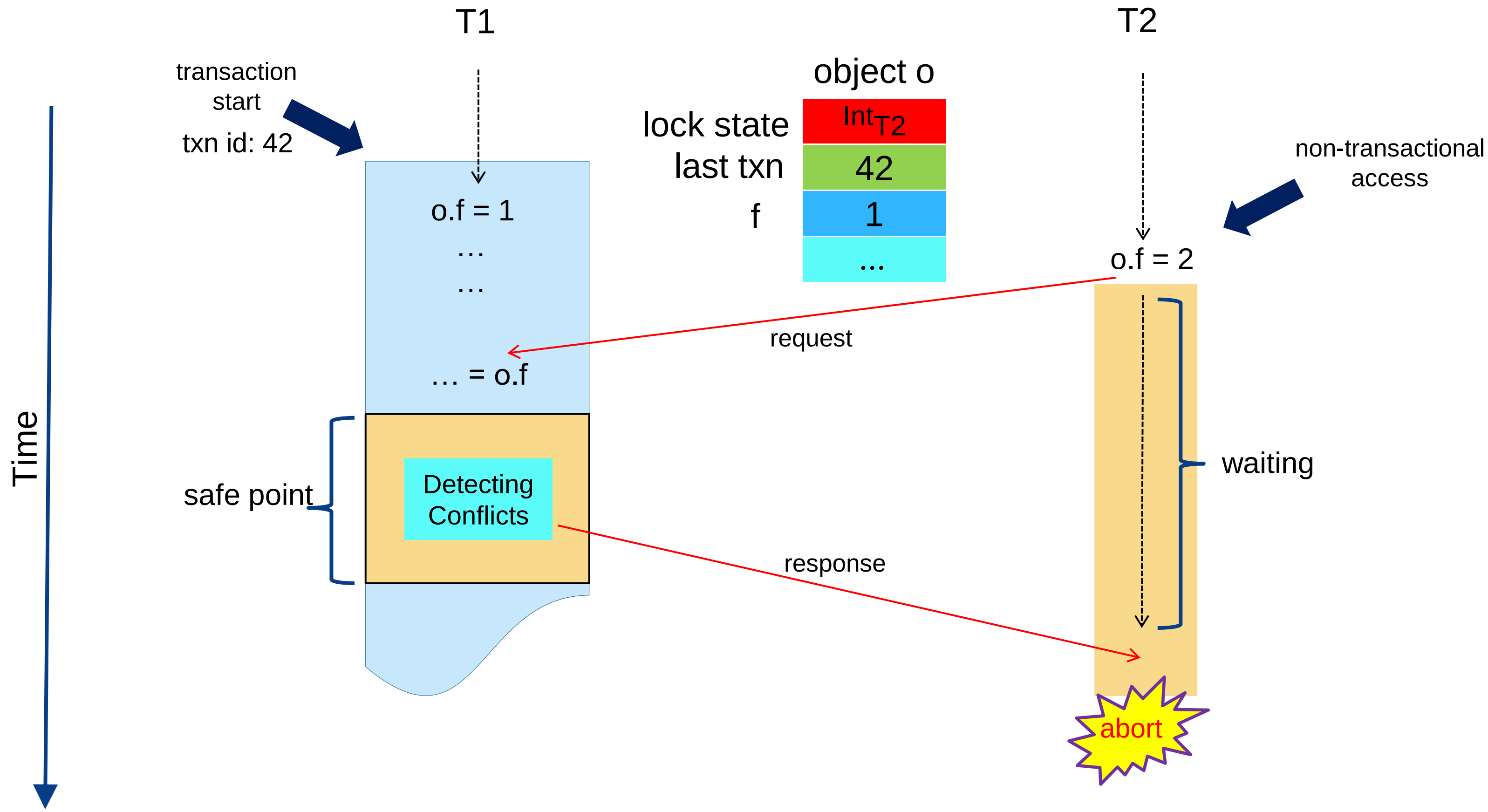
Transactional vs. Transactional Conflict

Strong Atomicity Semantics



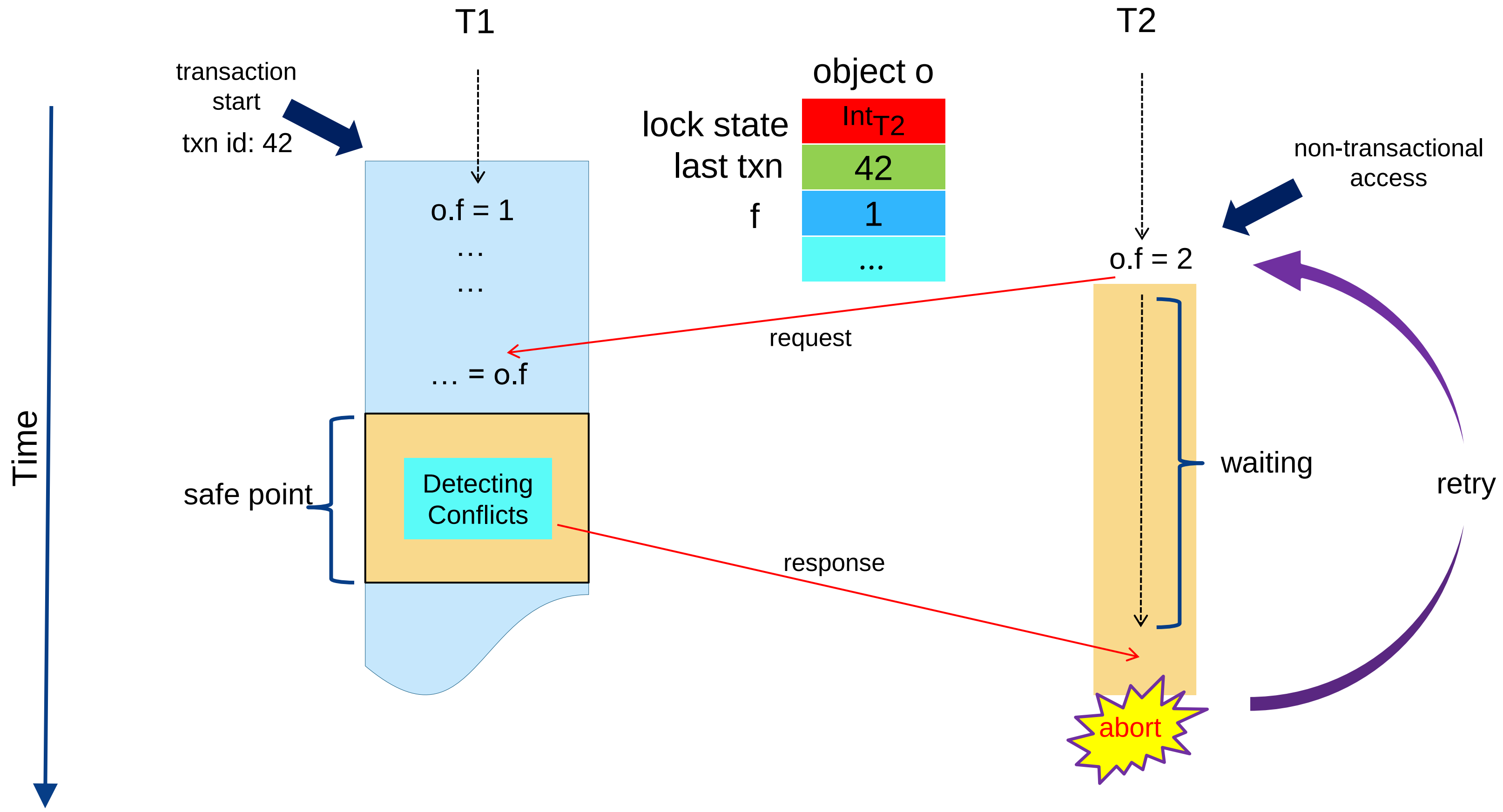
Transactional vs. Transactional Conflict

Strong Atomicity Semantics



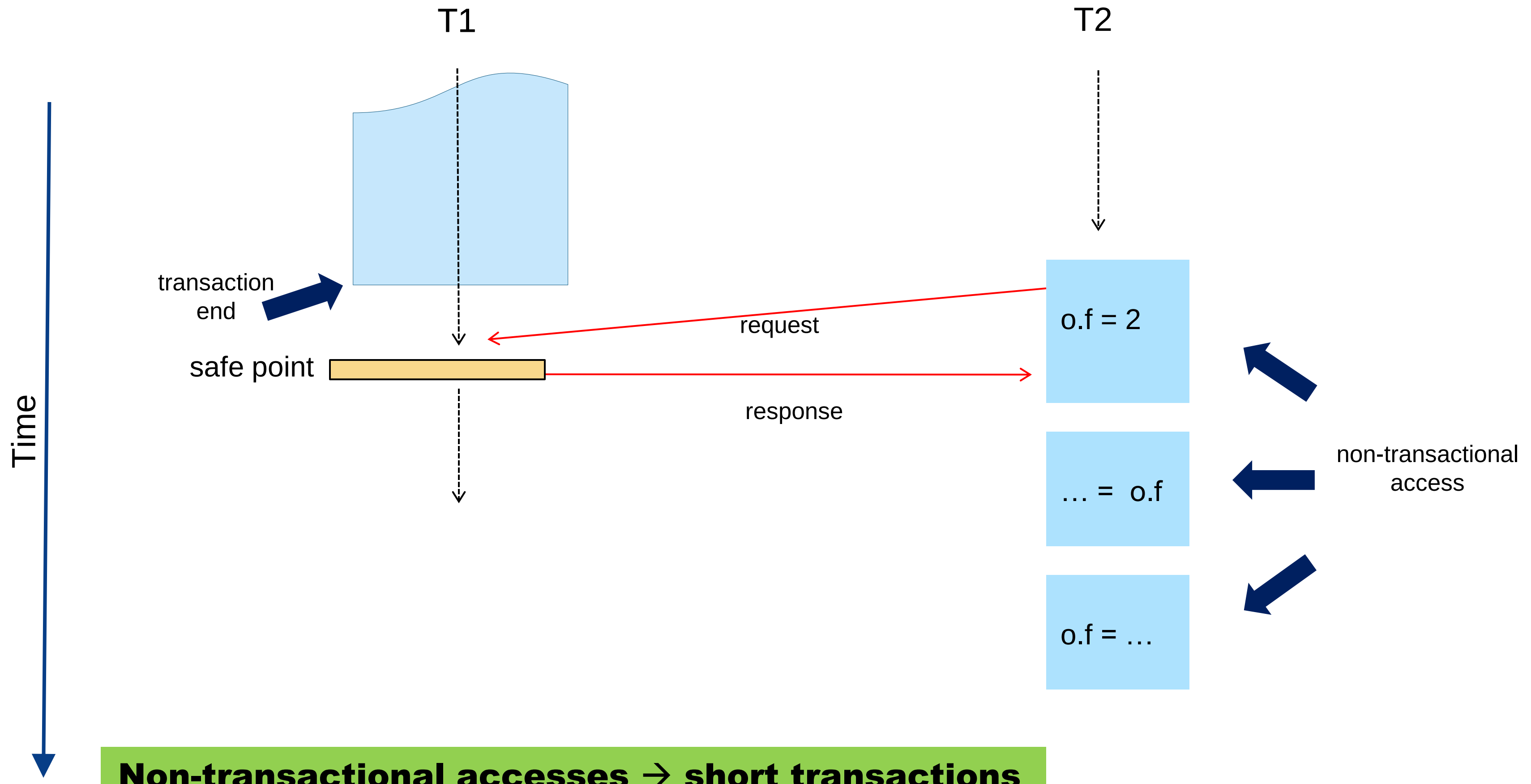
Transactional vs. Non-transactional Conflict

Strong Atomicity Semantics



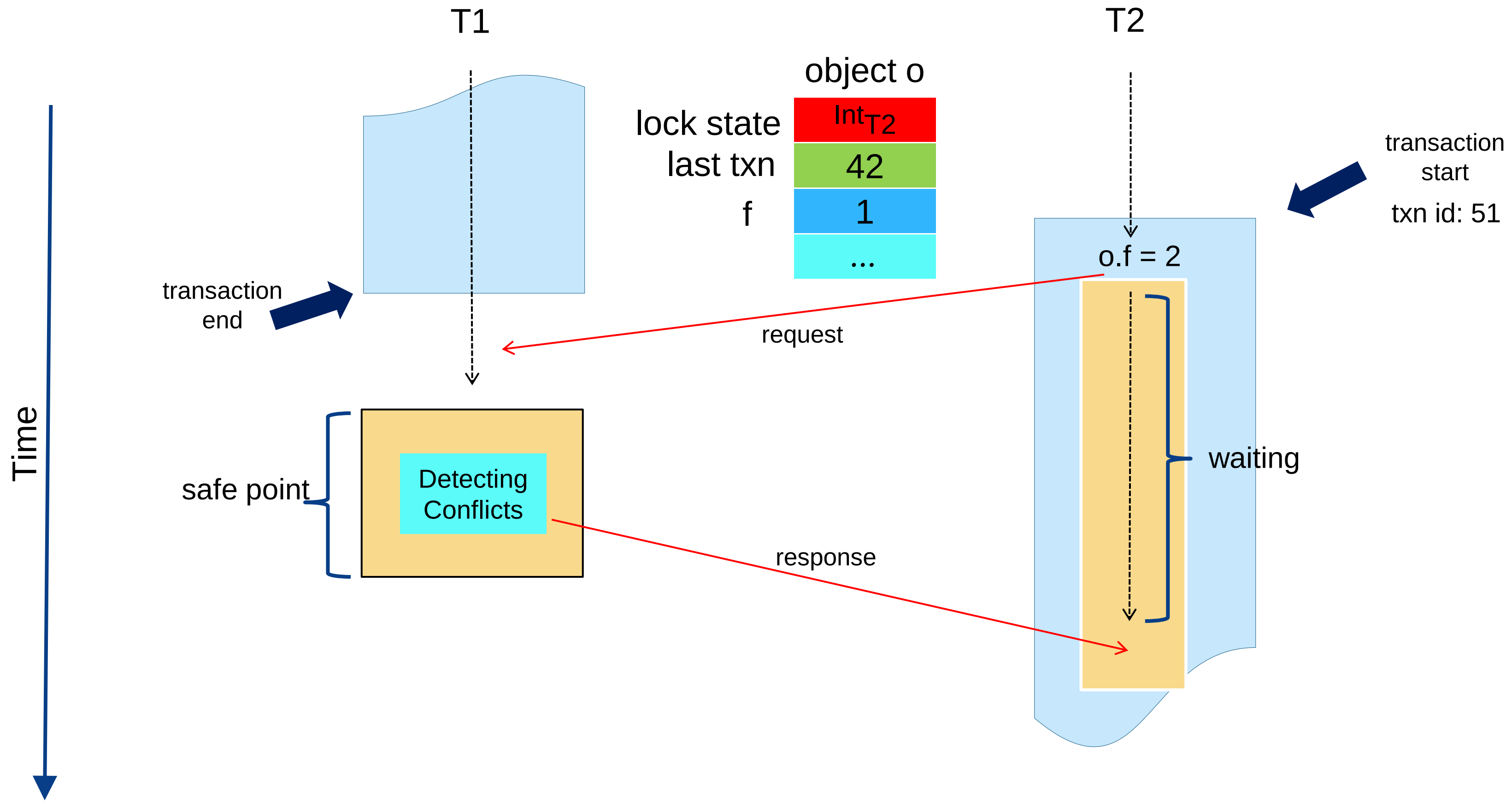
Transactional vs. Non-transactional Conflict

Strong Atomicity Semantics



Non-transactional accesses → short transactions
no setting up/tearing down cost

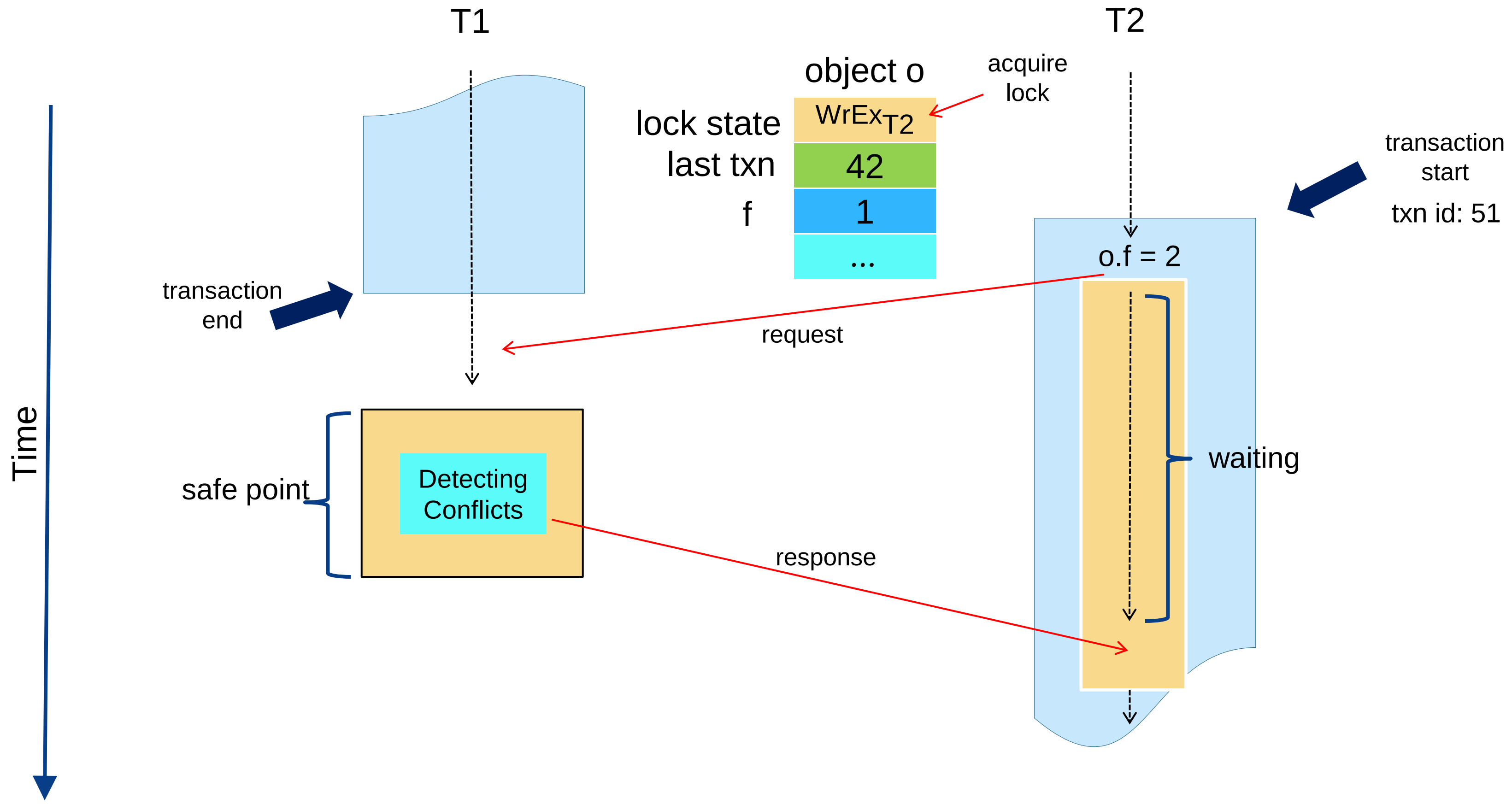
No Transactional Conflict



object o

lock state	Int_{T2}
last txn	42
f	1
	...

No Transactional Conflict



T1

T2

object o

lock state	WrEx_{T2}
last txn	42
f	1
	...

acquire lock

transaction start

txn id: 51

o.f = 2

transaction end

request

waiting

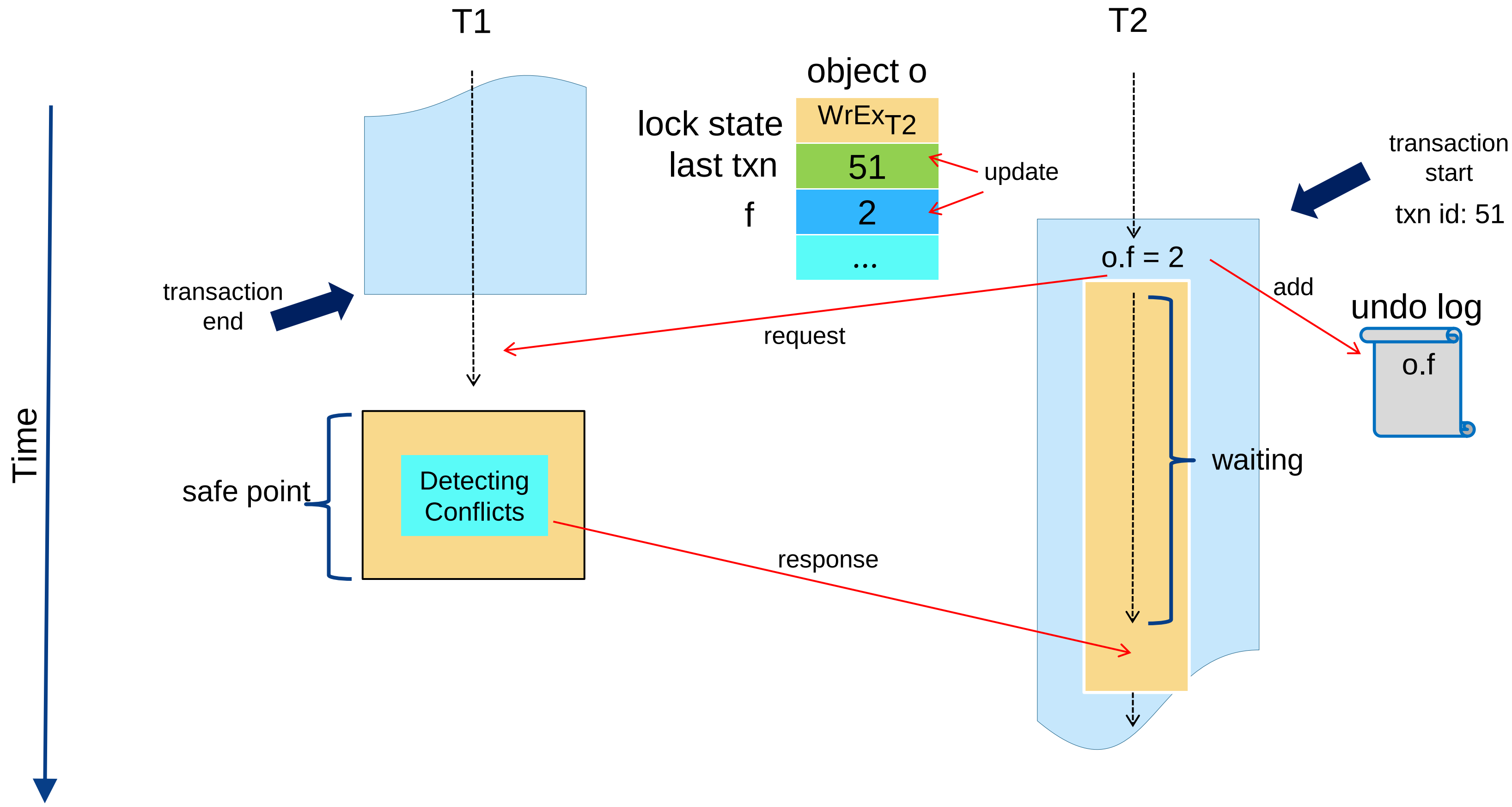
Time

safe point

Detecting Conflicts

response

No Transactional Conflict



T1

T2

object o

lock state	WrEx_{T2}
last txn	51
f	2
	...

update

transaction start

txn id: 51

o.f = 2

add

undo log

transaction end

request

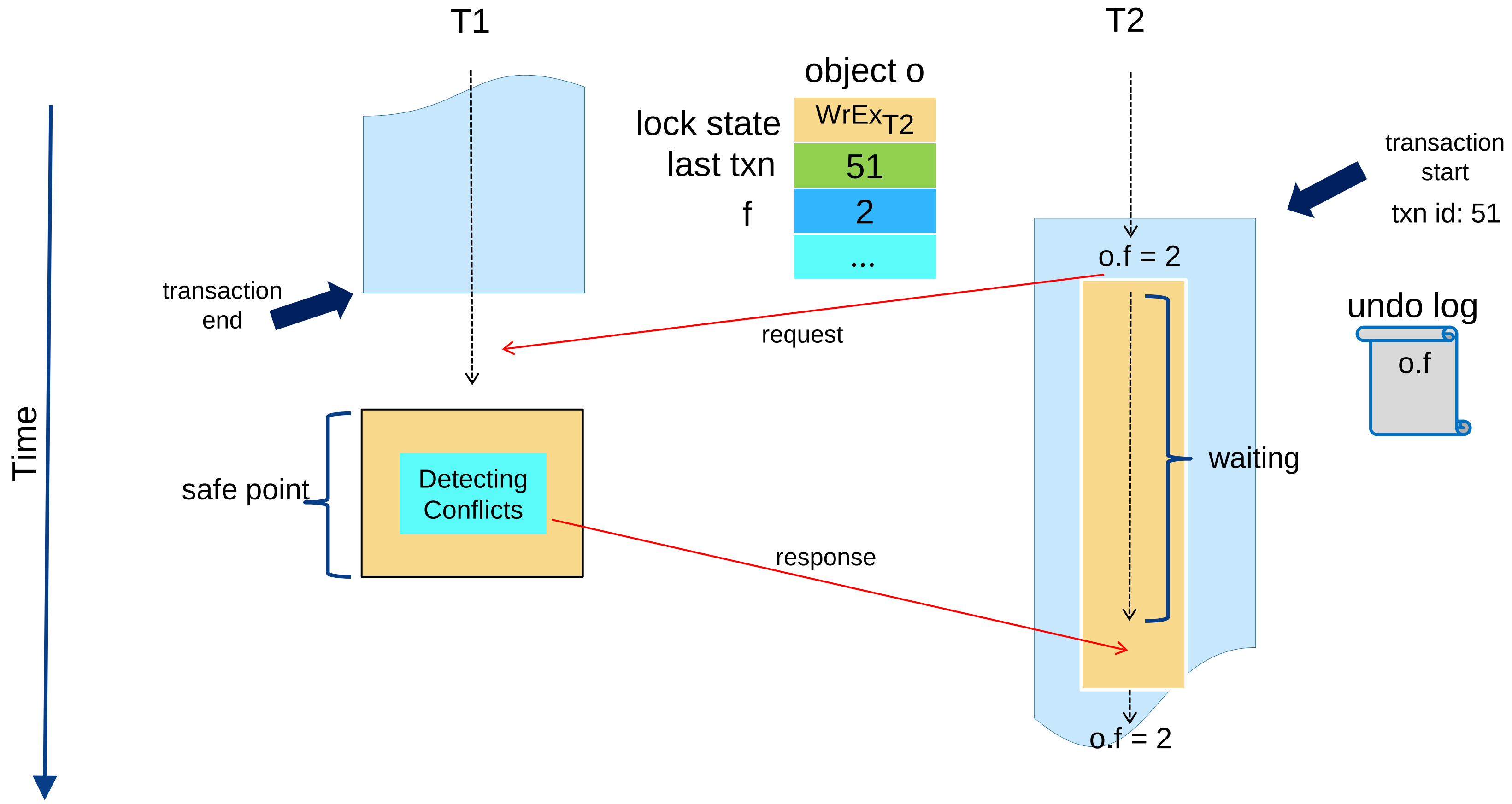
waiting

safe point

Detecting Conflicts

response

No Transactional Conflict

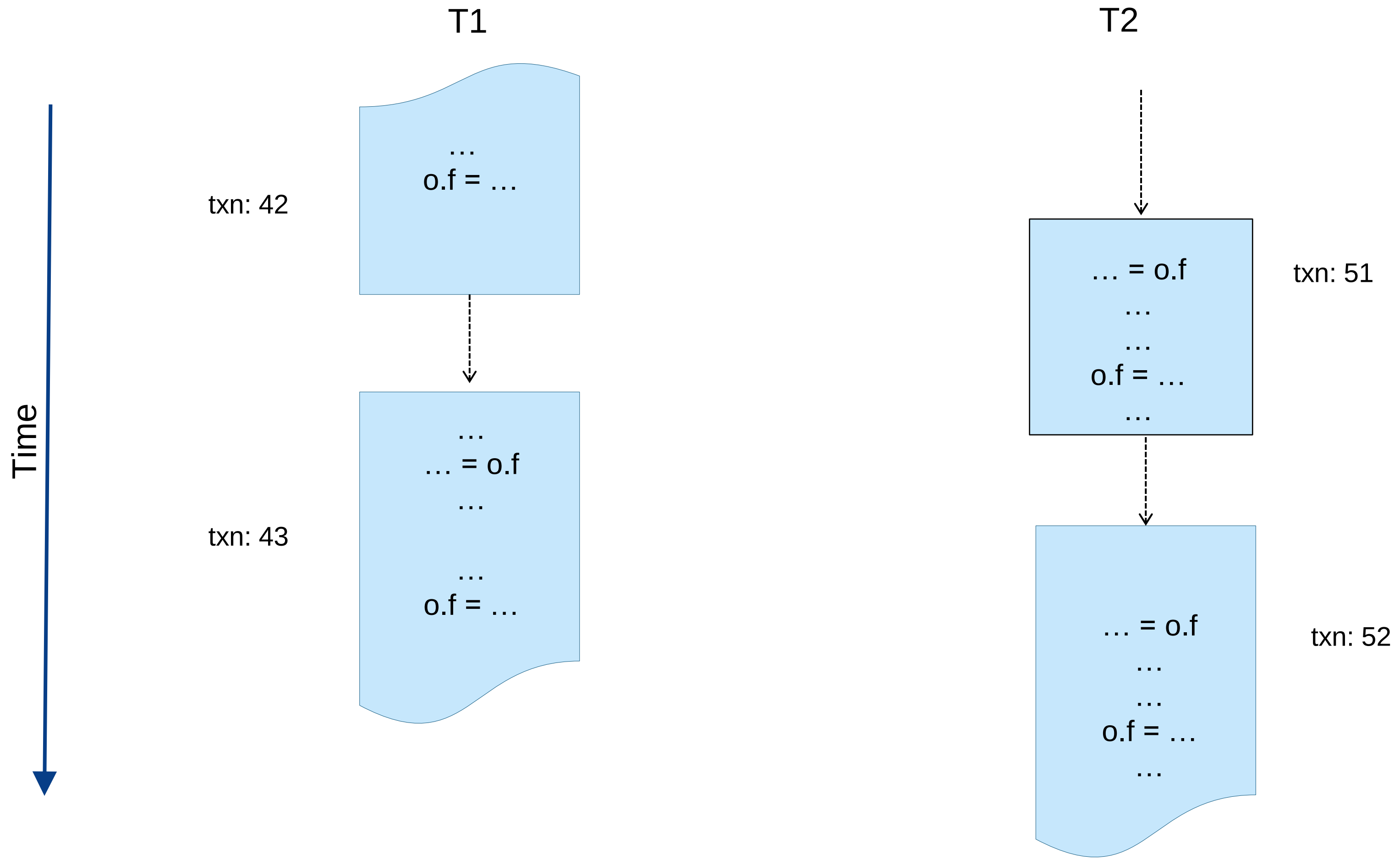


Two versions of coordination protocol

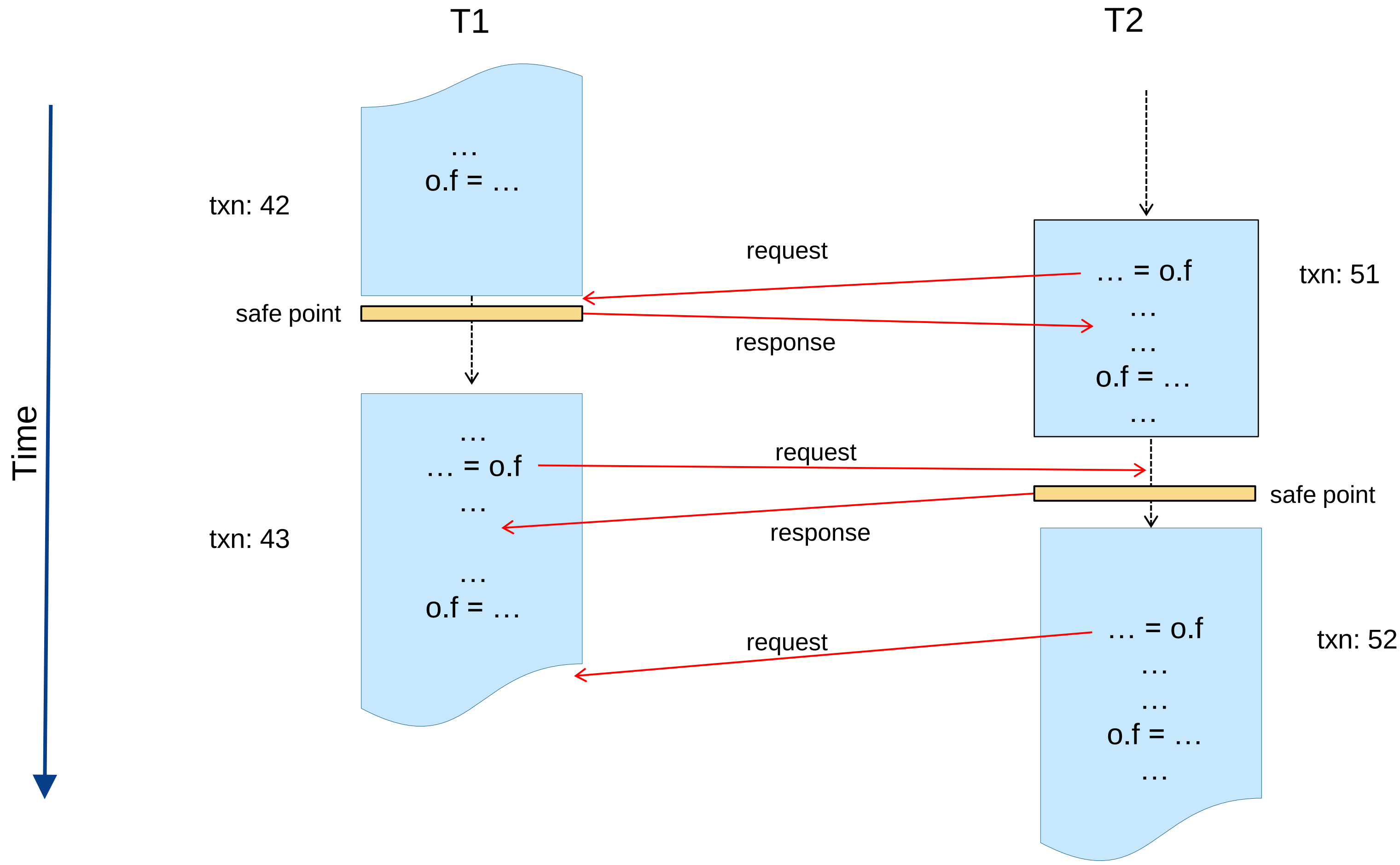
LarkTM-O

Adds very low overhead and scales well for low-contention cases

High-Contention Applications



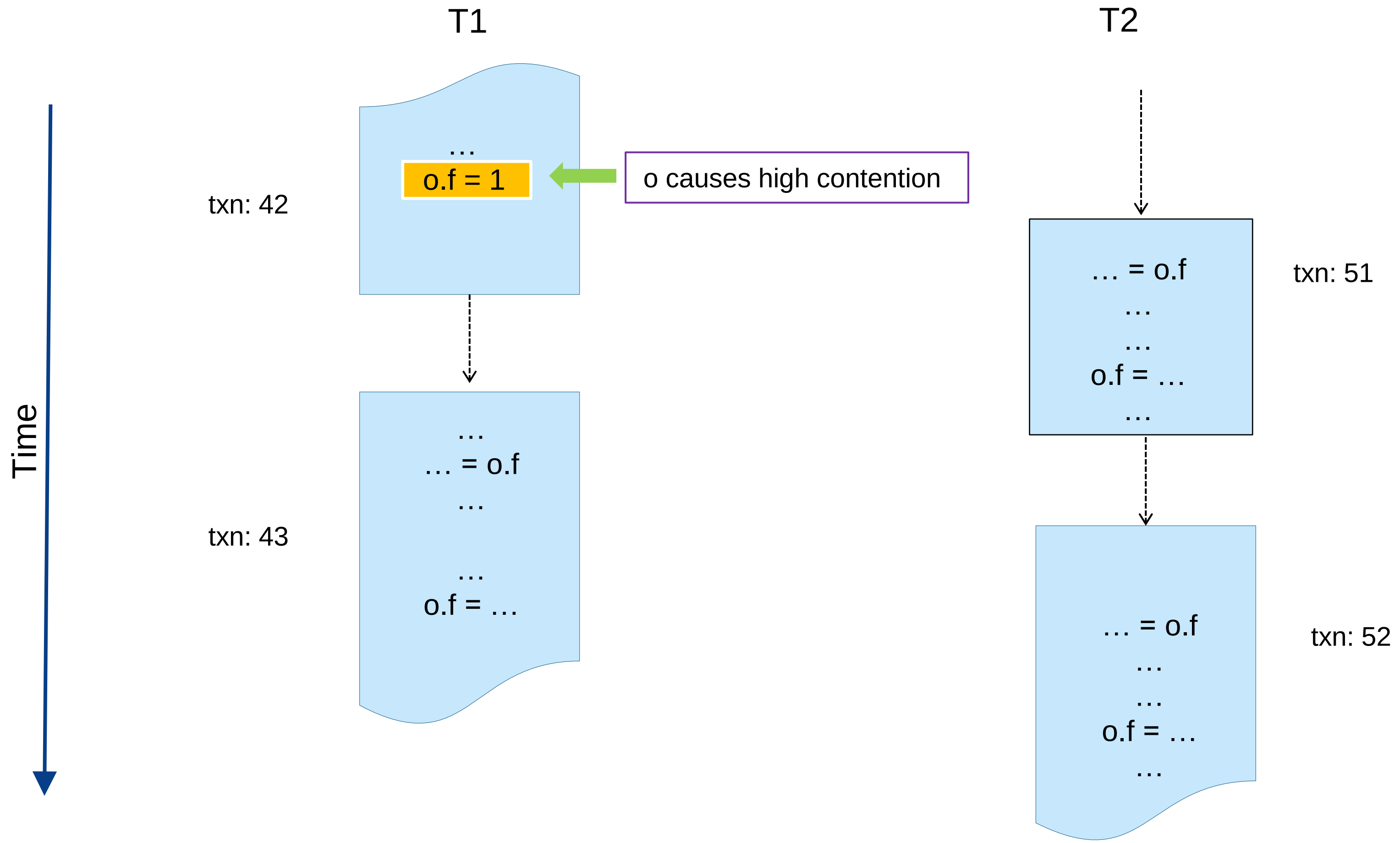
High-Contention Applications



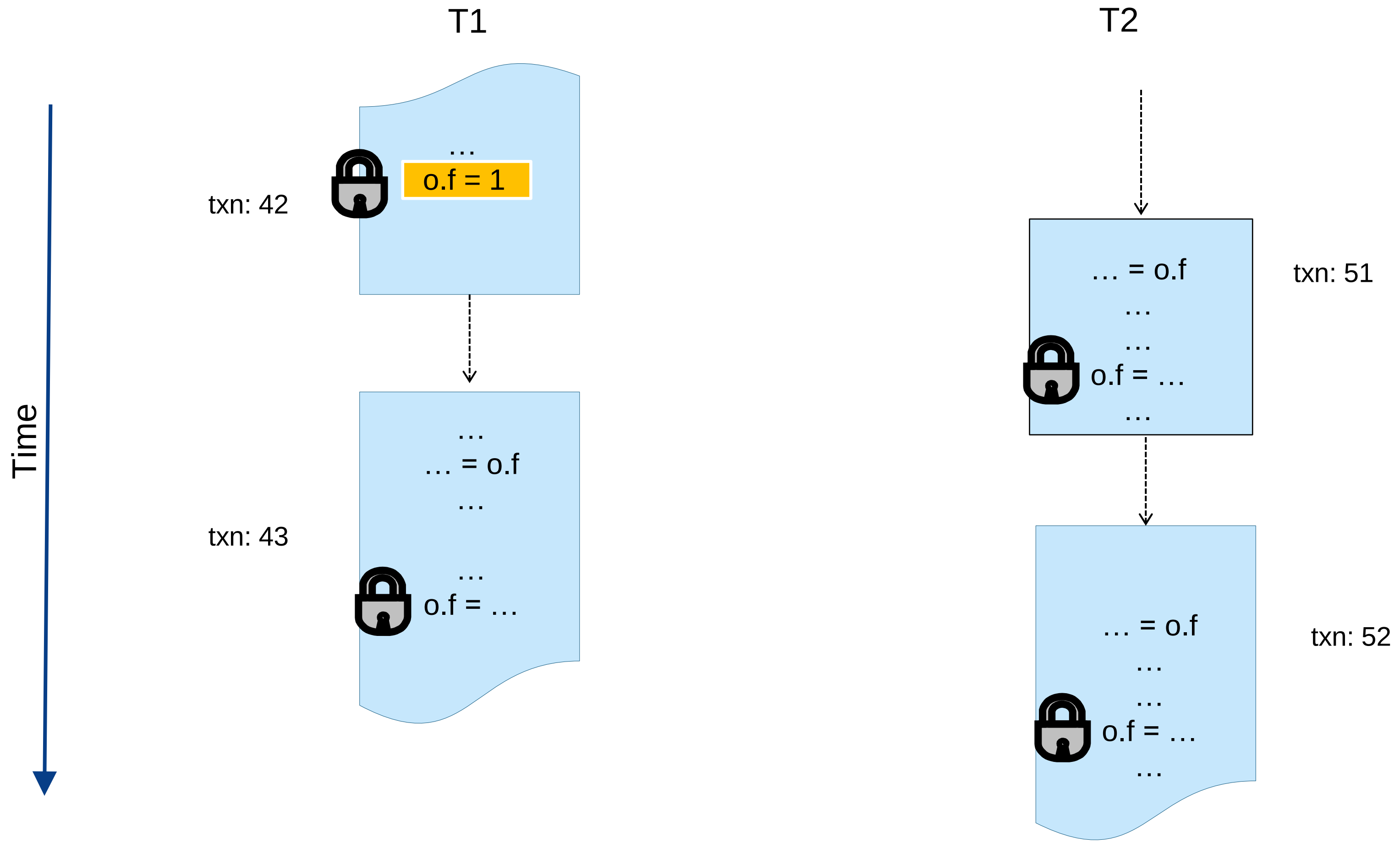
LarkTM-S

Handling High Contention

LarkTM-S: Hybrid with Traditional Locking



LarkTM-S: Hybrid with Traditional Locking



Comparison Of Concurrency Control

	Write concurrency control	Read concurrency control
LarkTM-O	Eager per-object biased reader–writer lock	Eager per-object biased reader–writer lock
LarkTM-S	IntelSTM–LarkTM-O hybrid	IntelSTM–LarkTM-O hybrid
IntelSTM[1,2]	Eager per-object lock	Lazy version validation
NOrec[3]	Lazy global seqlock	Lazy value validation

[1] B. Saha et al. McRT-STM: A High Performance Software Transactional Memory System for a Multi-Core Runtime. In PPoPP, 2006.
[2] T. Shpeisman et al. Enforcing Isolation and Ordering in STM. In PLDI, 2007.
[3] L. Dalessandro et al. NOrec: Streamlining STM by Abolishing Ownership Records. In PPoPP, 2010.

Comparison Of Instrumentation

	Instrumented accesses
LarkTM-O	All accesses
LarkTM-S	All accesses
IntelSTM	All accesses
NOrec	All transactional accesses

except redundant accesses

Comparison Of Progress Guarantees

	Progress Guarantee
LarkTM-O	Livelock and starvation free
LarkTM-S	Livelock and starvation free
IntelSTM	None
NOrec	Livelock free

Comparison Of Semantics

	Semantics
LarkTM-O	Strong Atomicity
LarkTM-S	Strong Atomicity
IntelSTM	Strong Atomicity
NOrec	Single Global Lock Atomicity (SLA)

Implementation

- LarkTM-O, LarkTM-S, IntelSTM (McRT), and NOrec
  - Developed in Jikes RVM 3.1.3
  - All STMs share features as much as possible (e.g., inlining decisions, redundant barrier analysis, name-mangling)
  - Source code publicly available on the Jikes RVM Research Archive

Artifact

Consistent * Complete * Well Documented * Easy to Reuse *

Evaluated

* PPoPP * AEC *

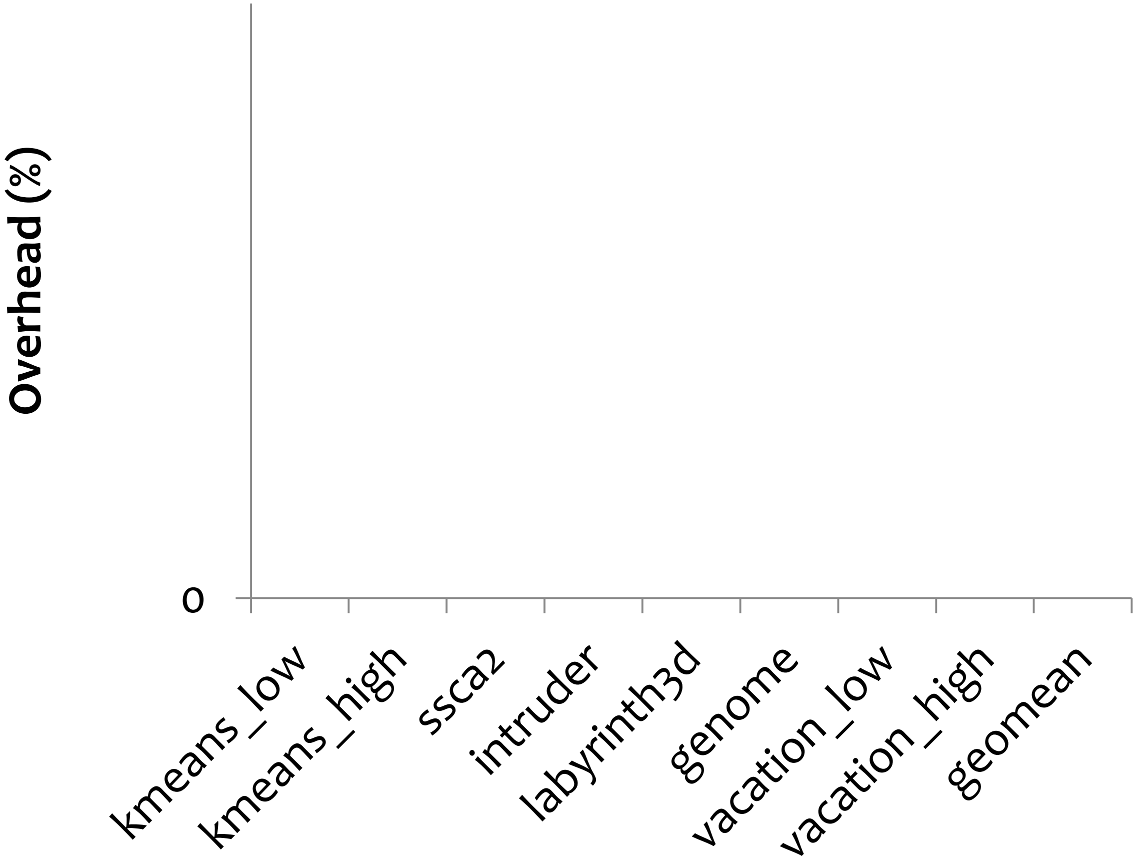
Evaluation Methodology

- **TM programs**
  - STAMP benchmarks

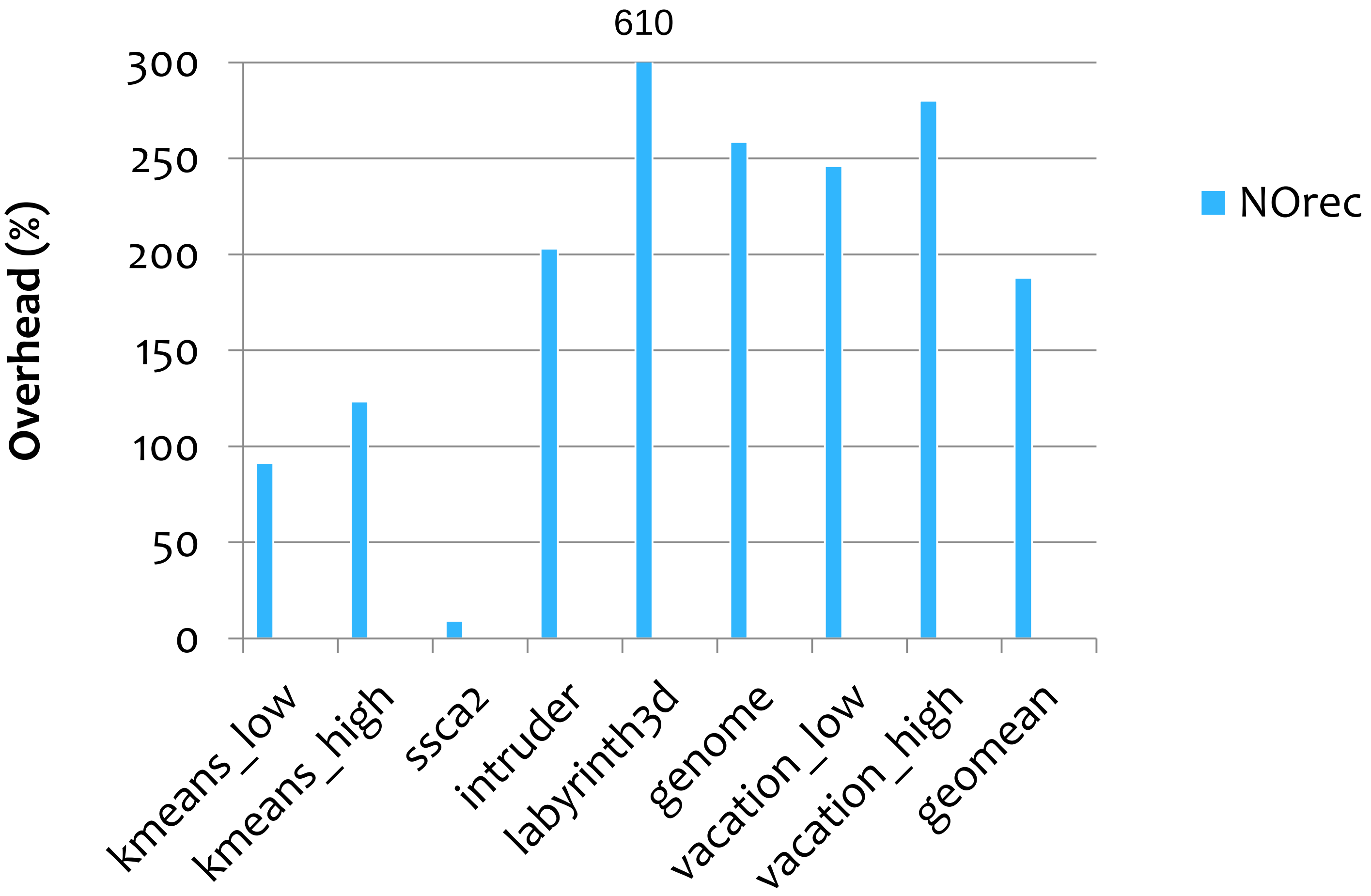
- **STM comparison**
  - Norec
  - IntelSTM
  - LarkTM-O
  - LarkTM-S

- **Platform**
  - **Eight 8-core processors (AMD Opteron 6272)**
  - Four 8-core processors (Intel Xeon E5-4620)

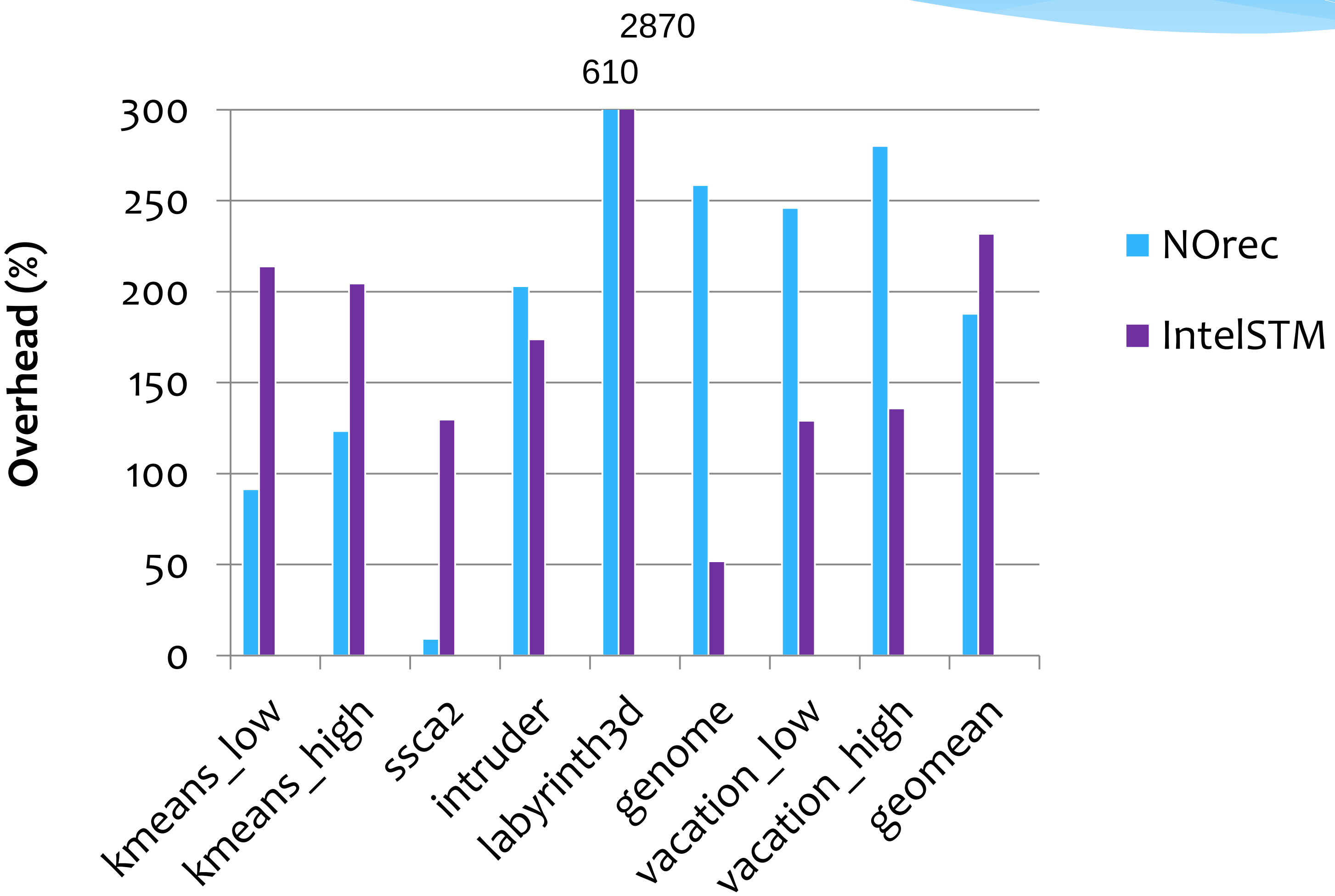
Single-Thread Performance



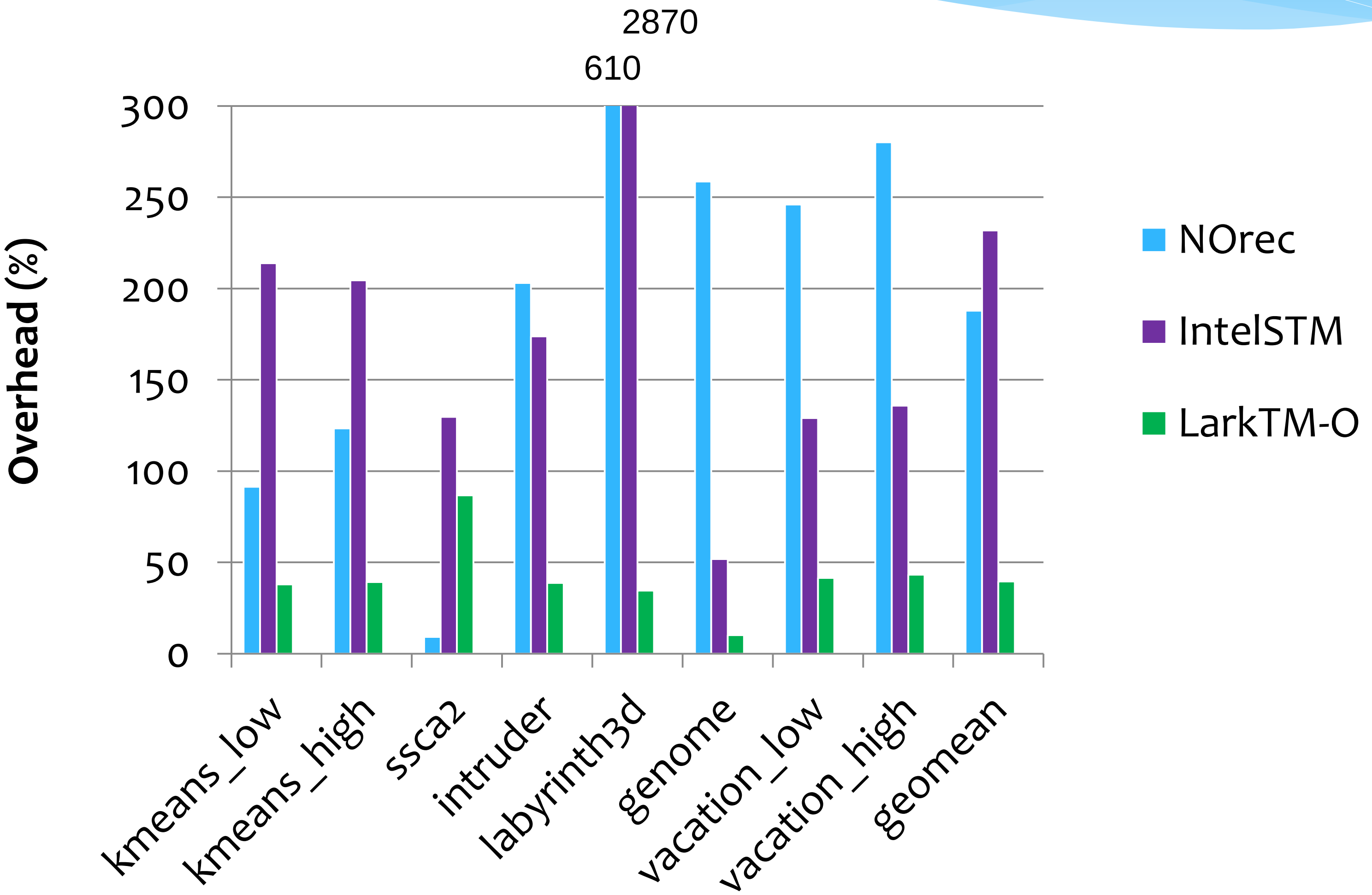
Single-Thread Performance



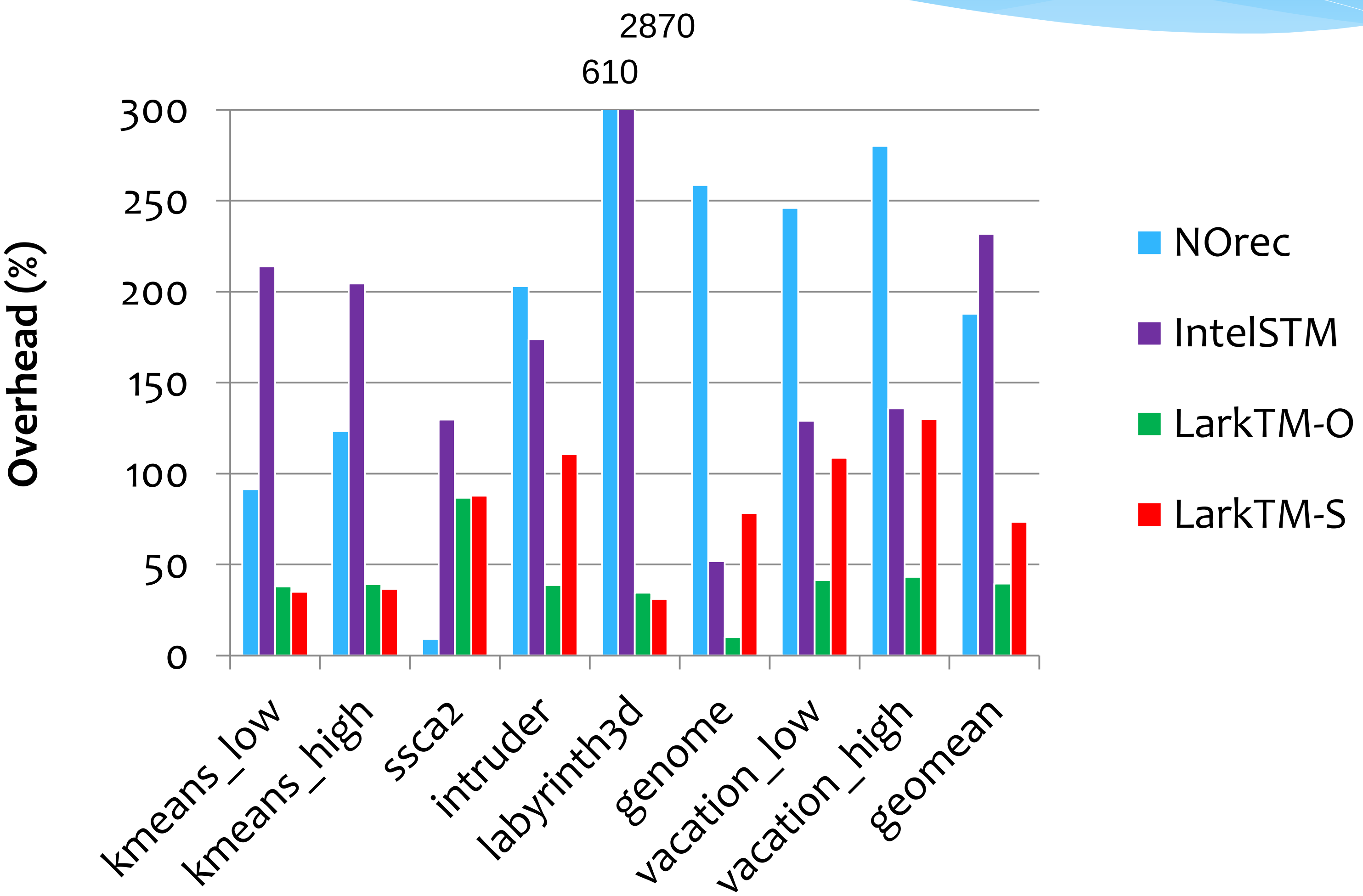
Single-Thread Performance



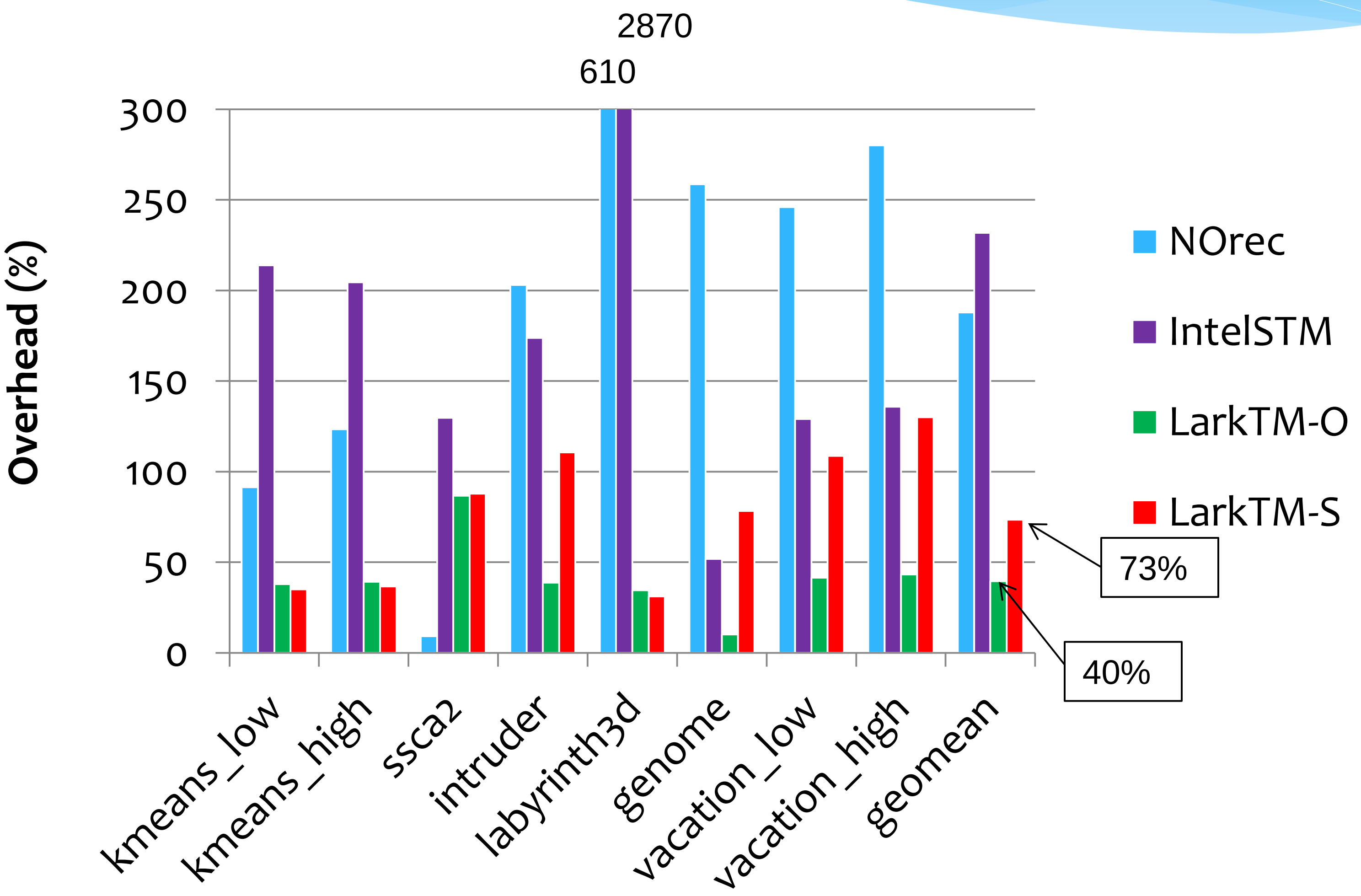
Single-Thread Performance



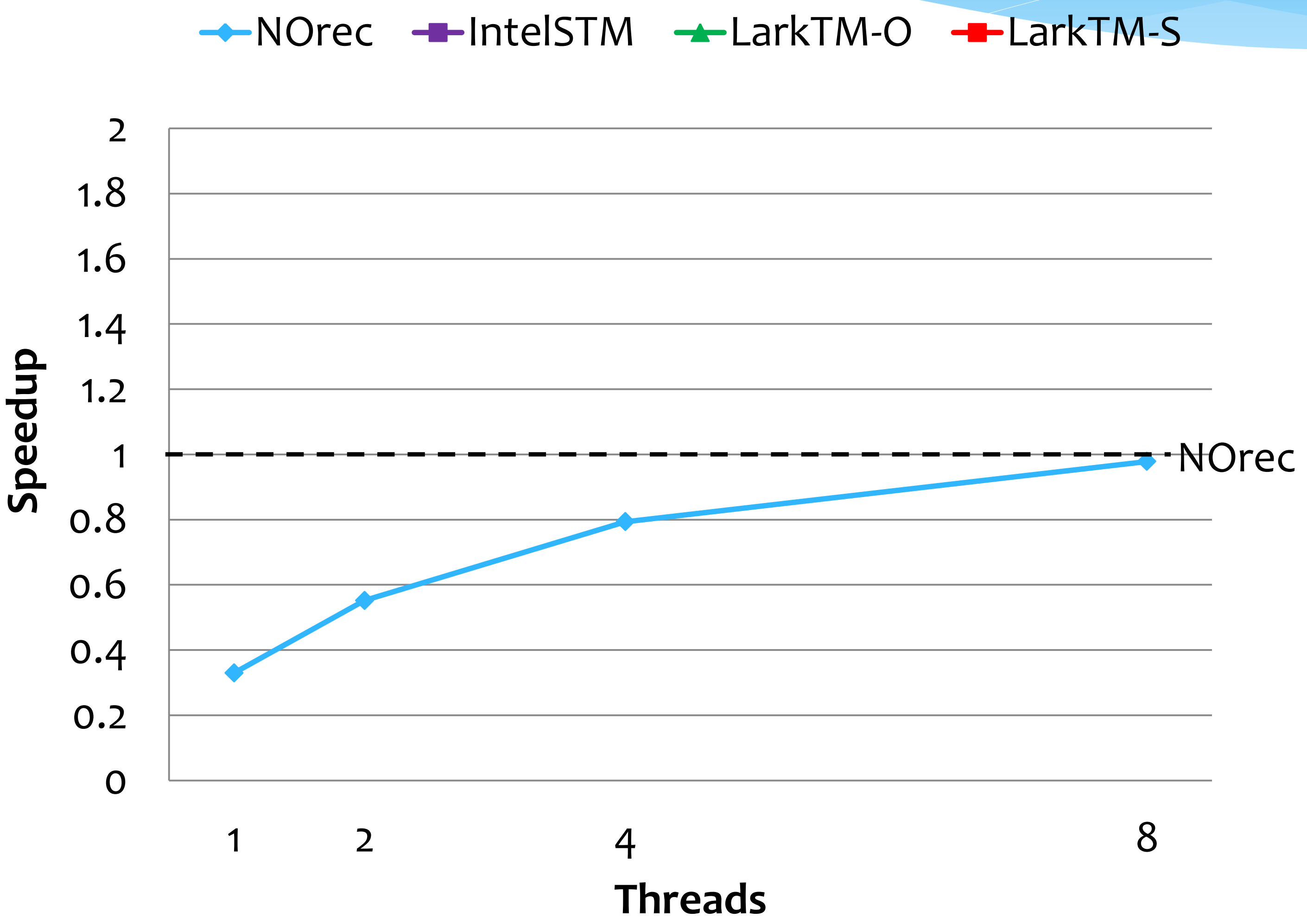
Single-Thread Performance



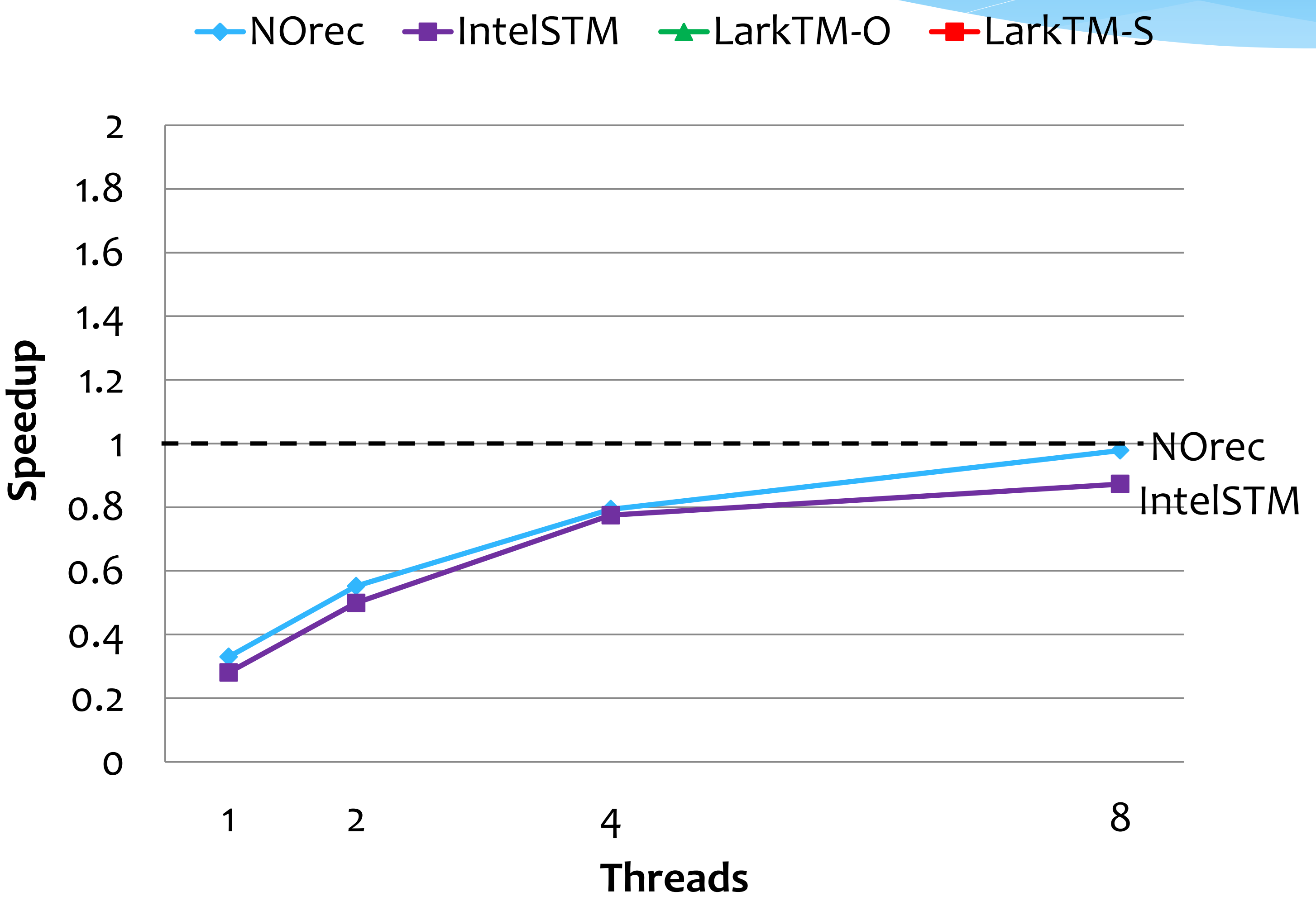
Single-Thread Performance



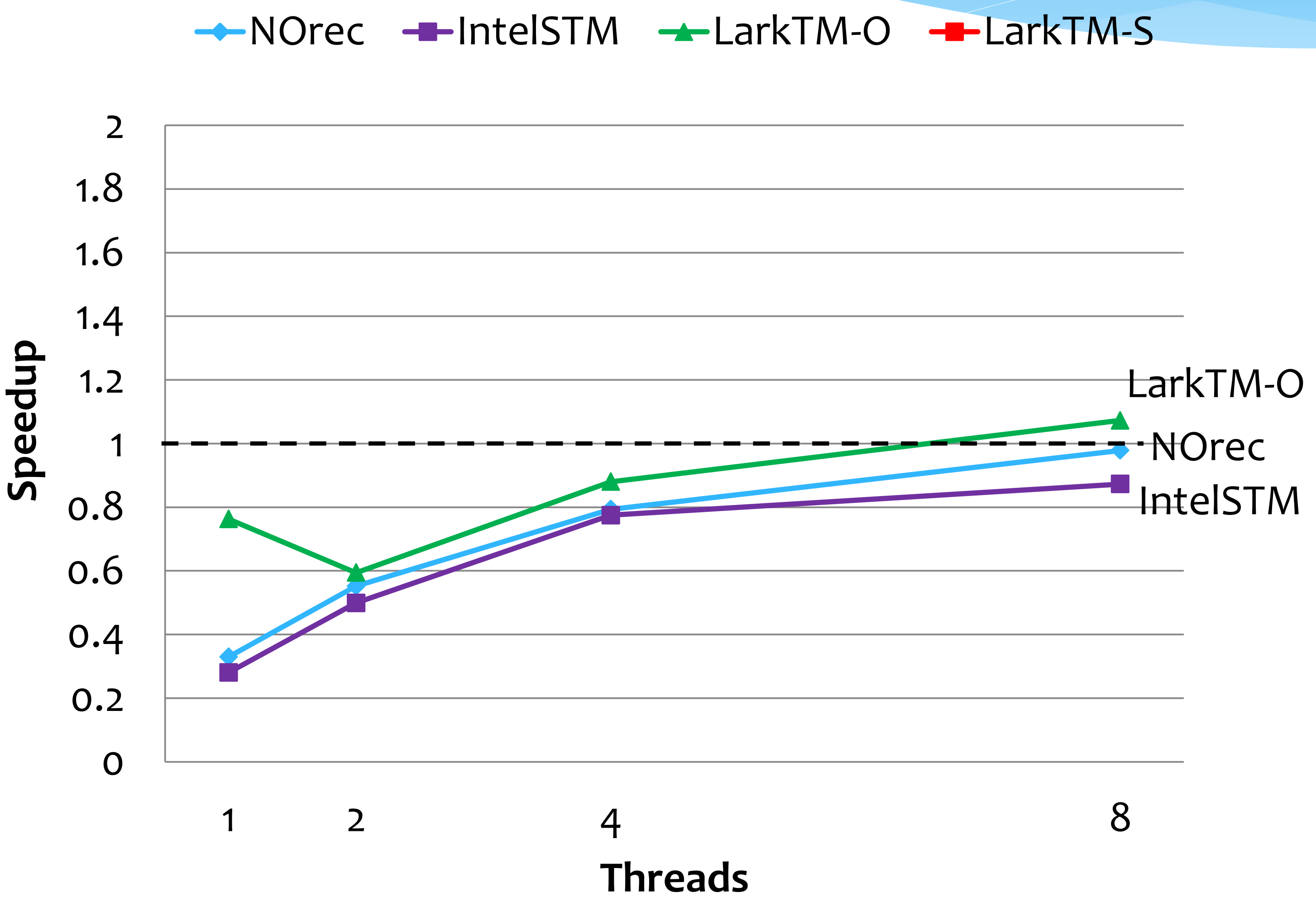
Speedup Geomean



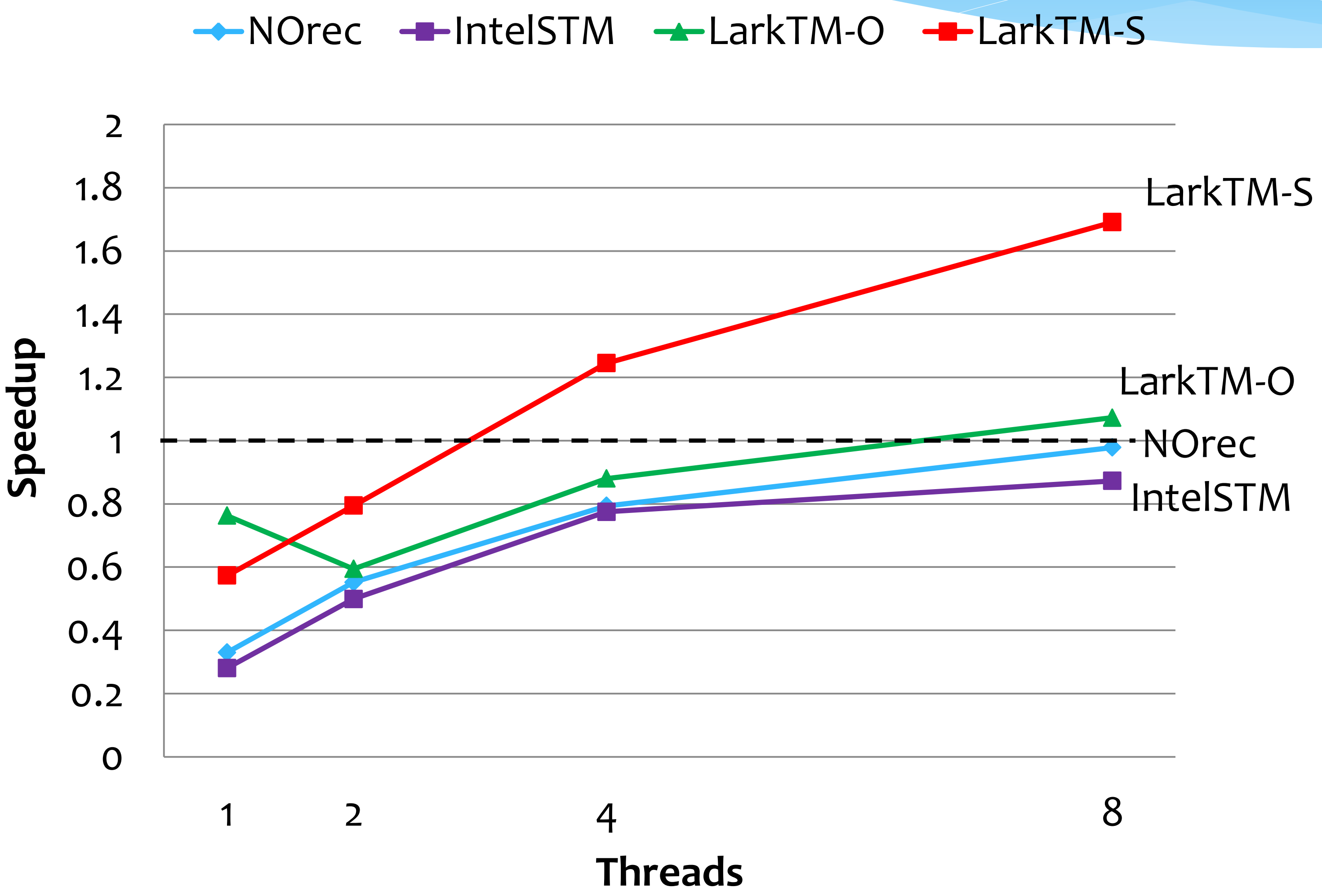
Speedup Geomean



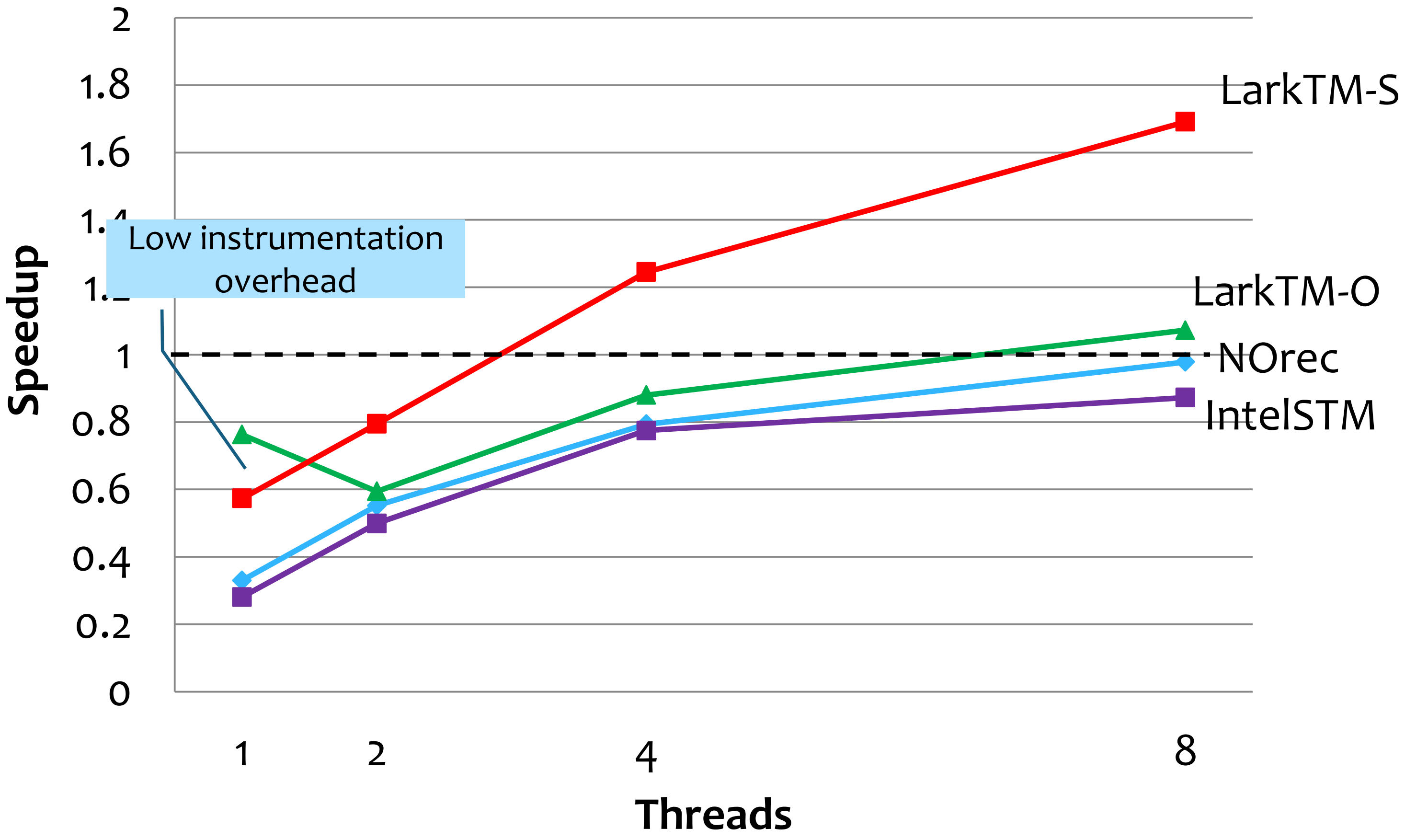
Speedup Geomean



Speedup Geomean

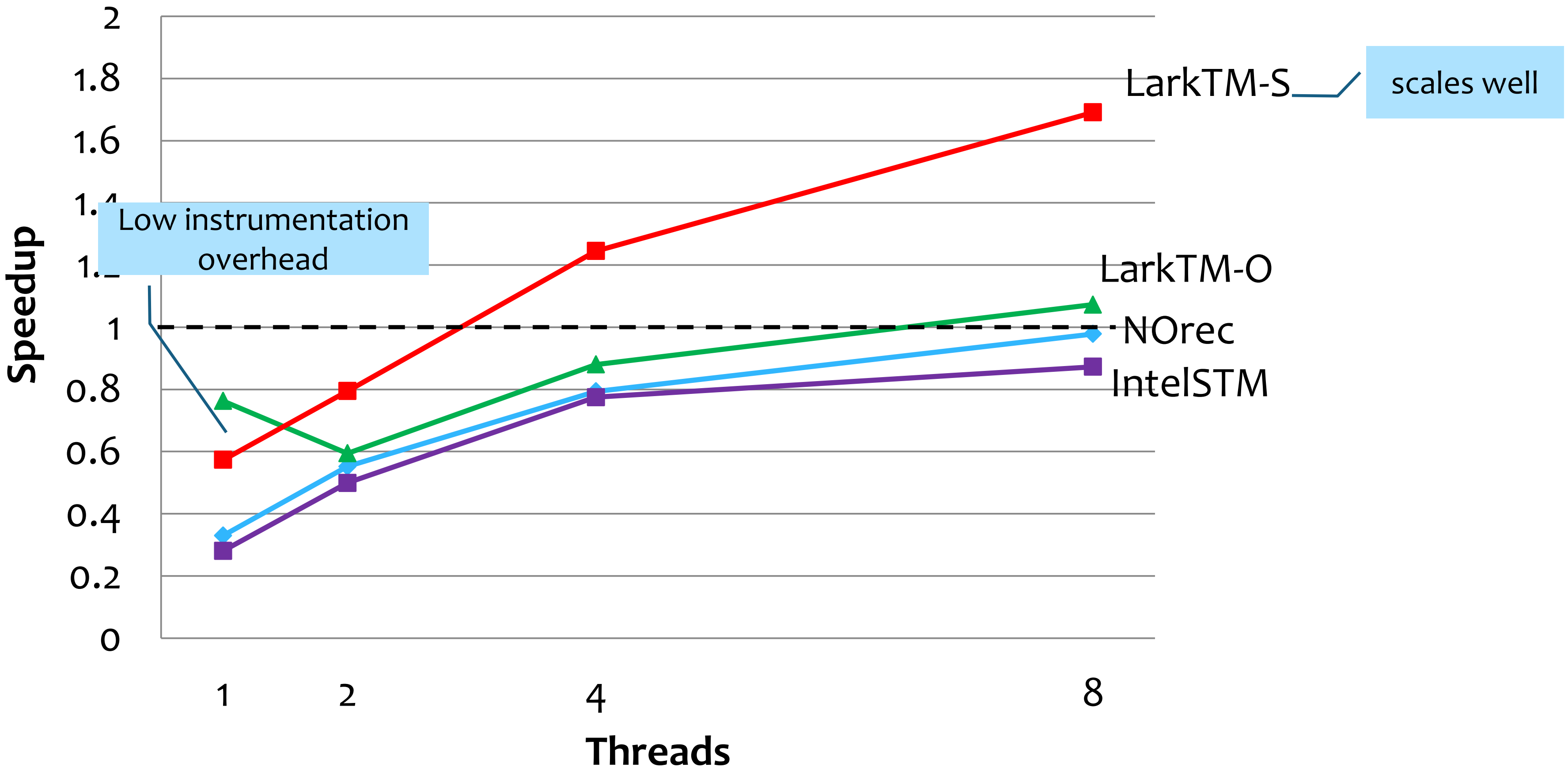


Toward Practical STM



Low instrumentation overhead

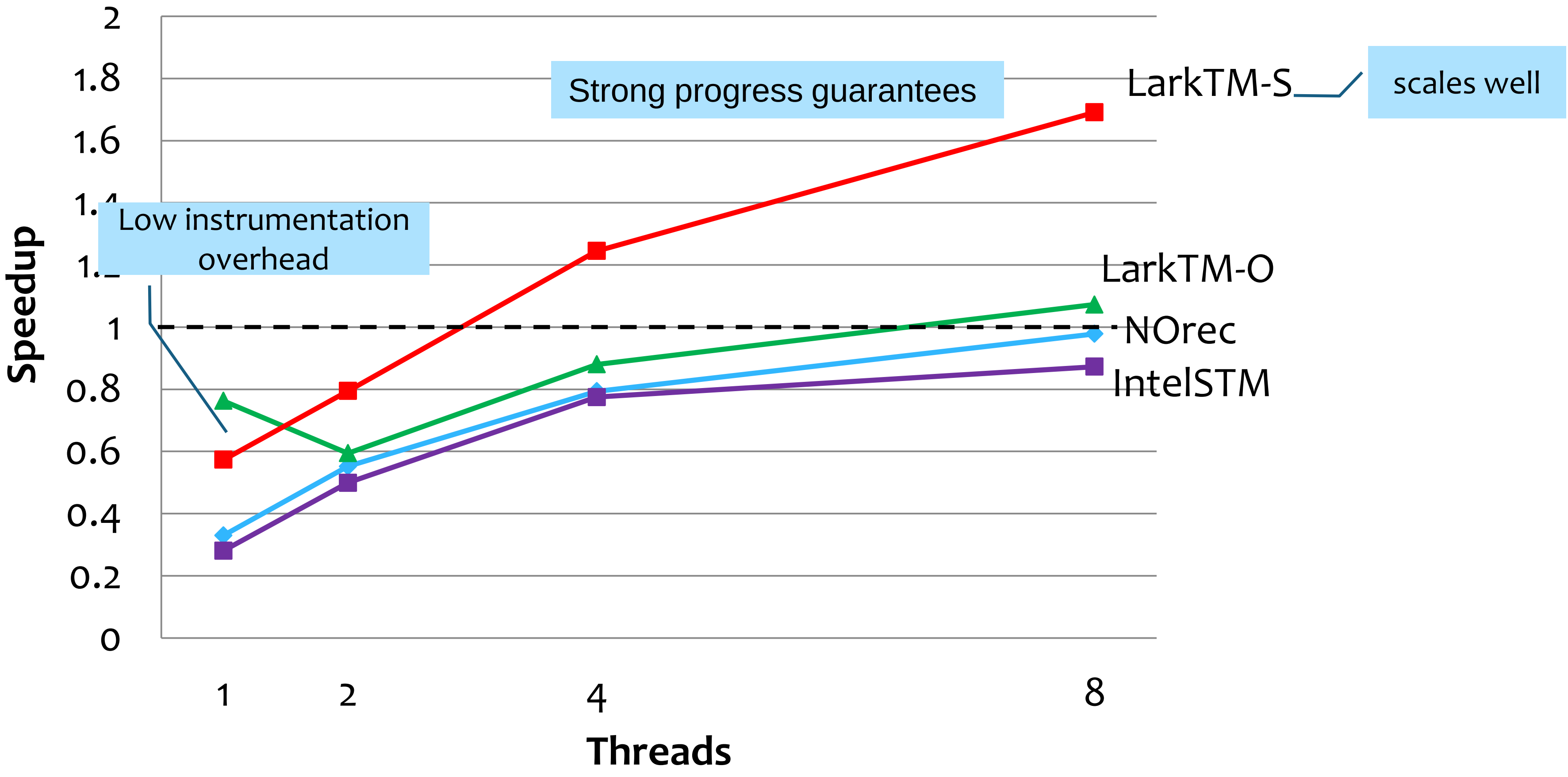
Toward Practical STM



Low instrumentation overhead

scales well

Toward Practical STM

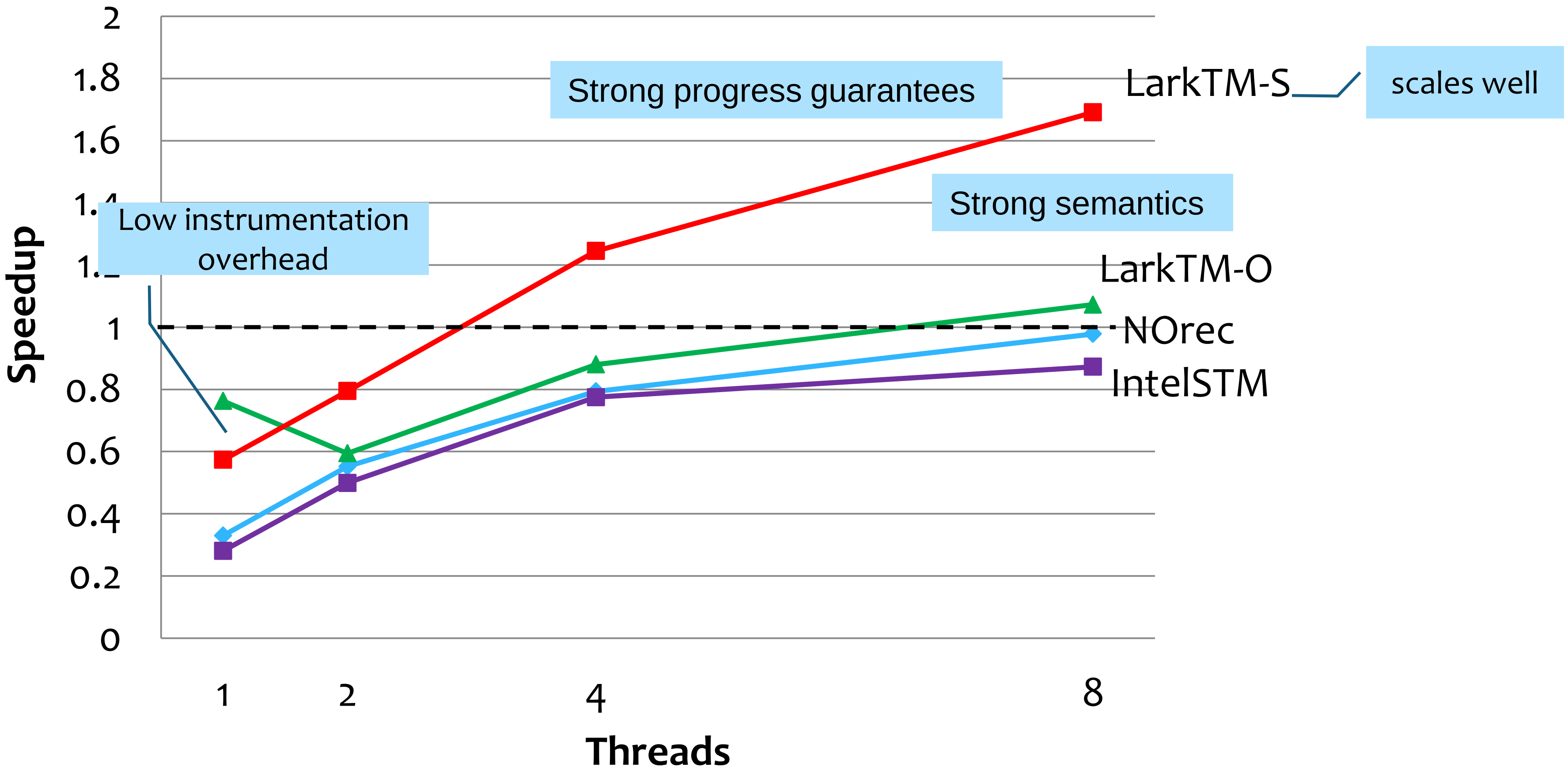


Strong progress guarantees

scales well

Low instrumentation overhead

Toward Practical STM



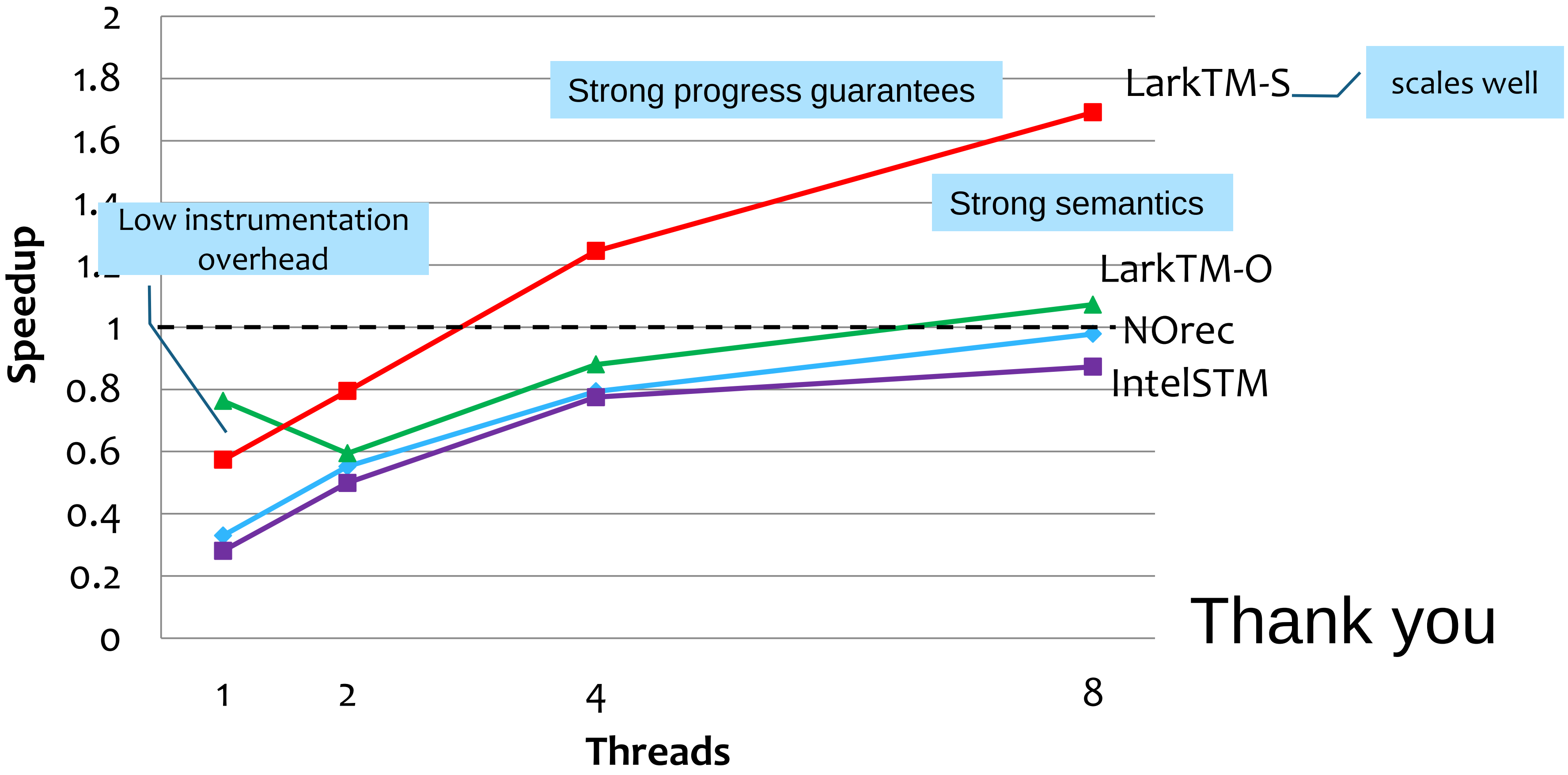
Strong progress guarantees

scales well

Strong semantics

Low instrumentation overhead

Toward Practical STM



Strong progress guarantees

scales well

Strong semantics

Low instrumentation overhead

Thank you