FRACTAL

AN EXECUTION MODEL FOR FINE-GRAIN NESTED SPECULATIVE PARALLELISM

SUVINAY SUBRAMANIAN, MARK C. JEFFREY,
MALEEN ABEYDEERA, HYUN RYONG LEE, VICTOR A. YING,
JOEL EMER, DANIEL SANCHEZ

ISCA 2017

Current speculative systems scale poorly

Speculative parallelization, e.g. TM, simplifies parallel programming

Performs poorly on real world applications…

…because applications comprise large atomic tasks

Large atomic tasks limit performance

Database Transaction

```
query X
…
…
update Z
…
…
query U
…
…
update V
```

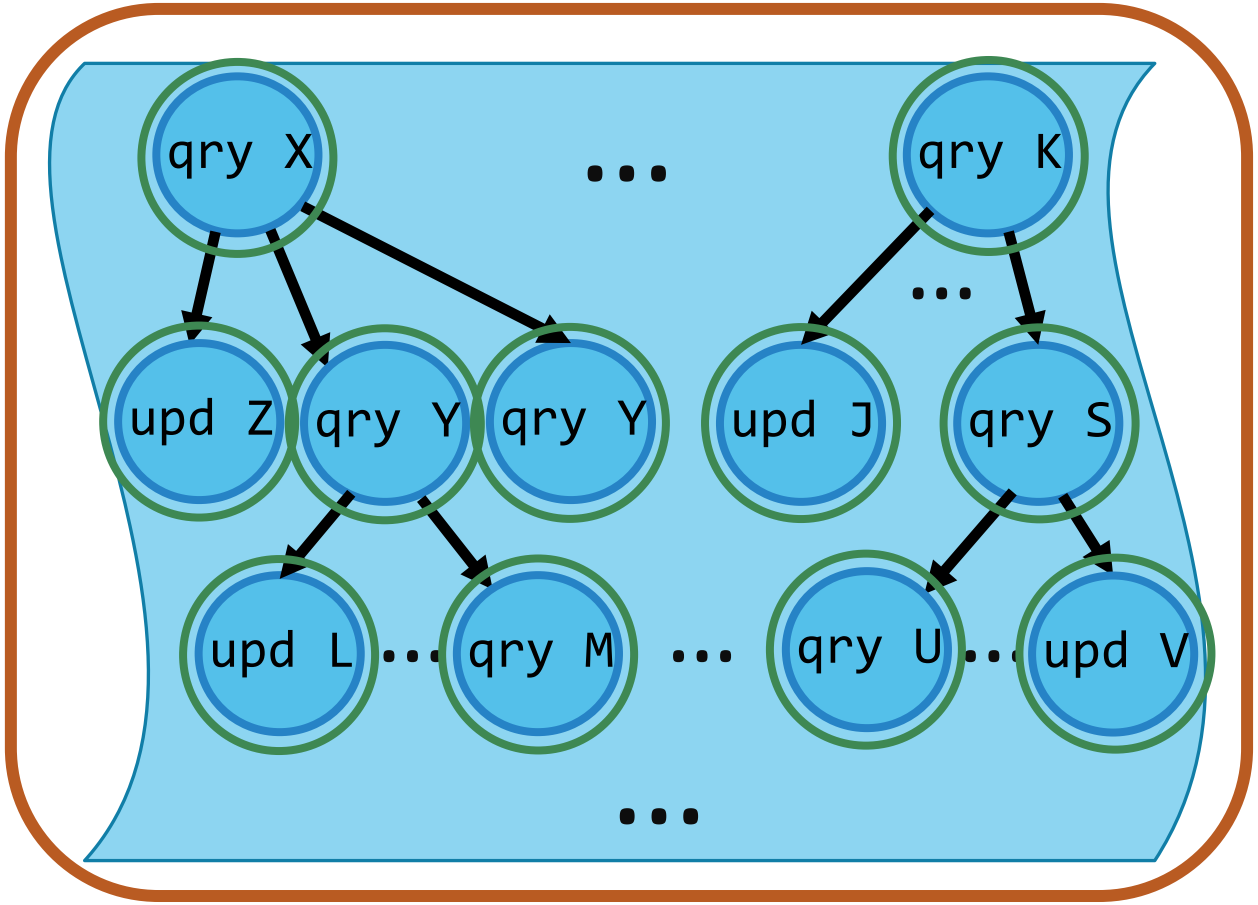
Millions of cycles

Prone to aborts

Challenging to track

Serial (misses parallelism)

Large atomic tasks have abundant nested parallelism!

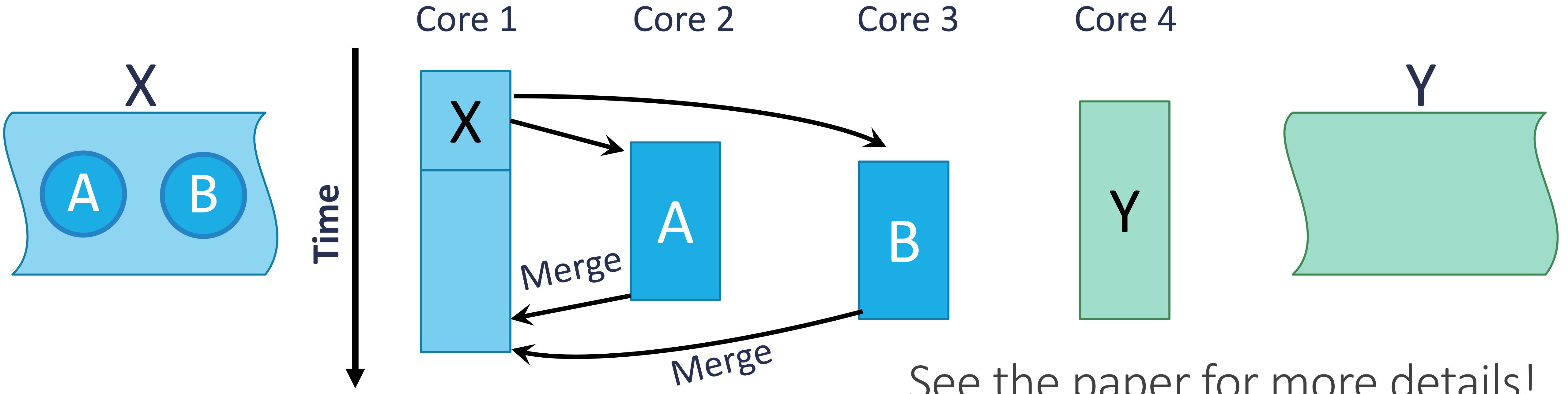


How to

- extract parallelism?
- maintain atomicity?
- achieve high performance?

Prior TMs fail to exploit nested parallelism

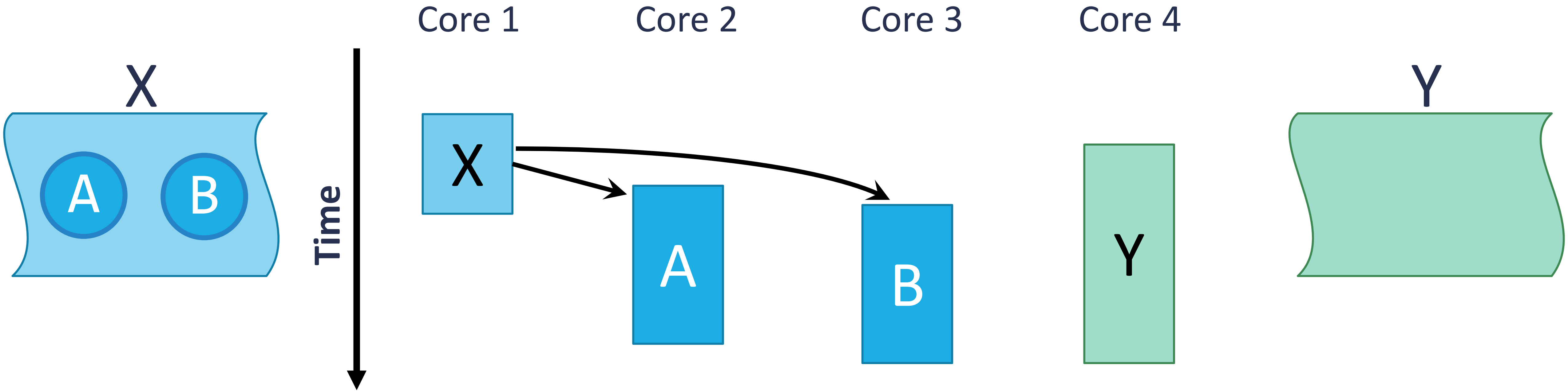
1. Merging of "nested" speculative state with parent
   Large speculative state, prone to aborts
2. Cyclic dependence between parent and nested children
   Deadlock and livelock issues



See the paper for more details!

Ordering tasks to guarantee atomicity

$$\underset{1}{X} \prec \underset{2}{Y} \qquad\qquad \underset{1}{X} \prec \underset{1.1}{A} \prec \underset{1.2}{B} \prec \underset{2}{Y}$$



Fractal decouples atomicity from parallelism

1. Decouples unit of atomicity from unit of parallelism
   - *Domain*: All tasks belonging to a domain appear to execute atomically

2. Implementation guarantees atomicity by ordering tasks
   - No merging speculative state

Benefits of **Fractal** $\left\{ \begin{array}{l} \text{Tiny tasks} \\ \text{Easy to track} \\ \text{Composable speculative parallelism} \end{array} \right.$

Fractal Execution Model

DECOUPLING ATOMICITY FROM PARALLELISM

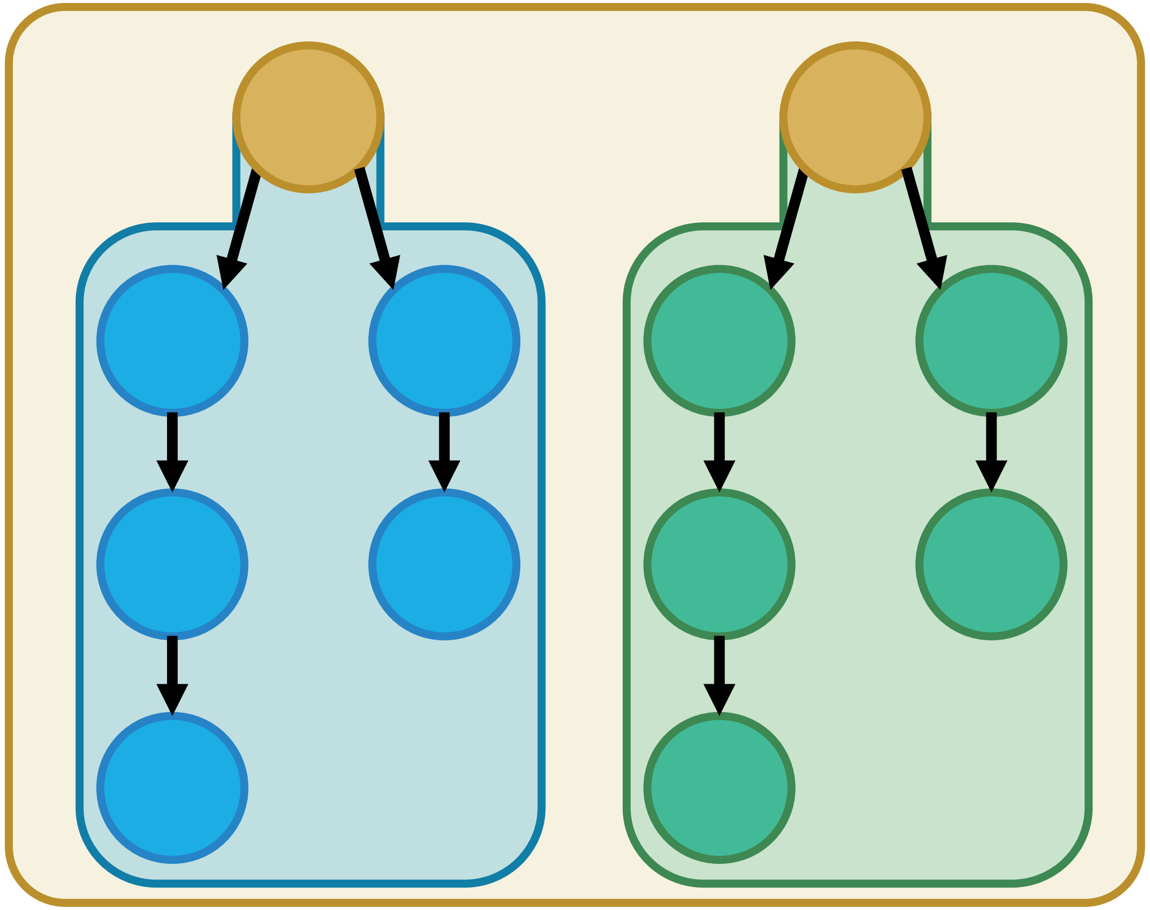
Domains to group tasks into atomic units

Fractal programs consist of atomic tasks

Tasks may access arbitrary data

Tasks may create child tasks

Tasks belong to a hierarchy of nested domains



Semantics across domains

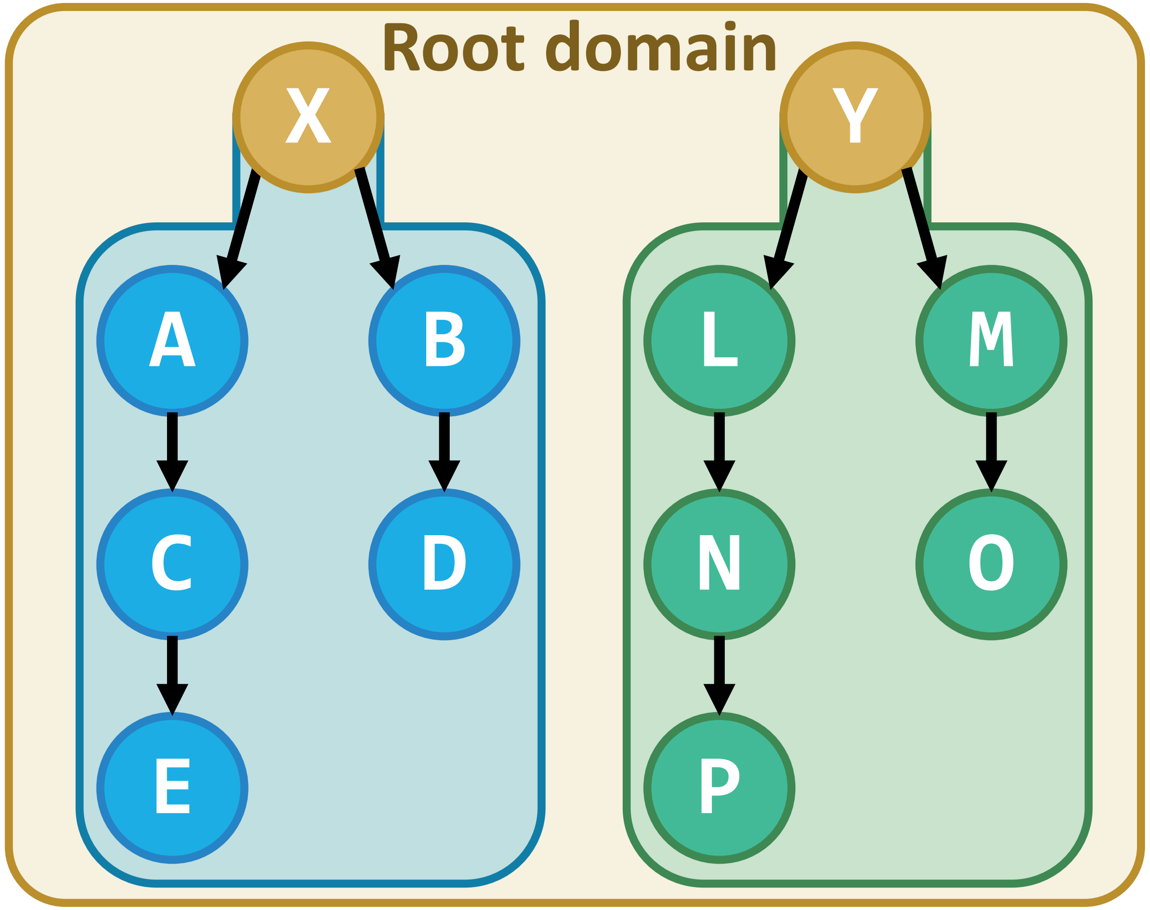
Each task:

- can create a single *subdomain*
- can enqueue child tasks to *subdomain* or *current domain*

(All tasks in domain + creator of domain)

↓

Appear to execute as single atomic unit



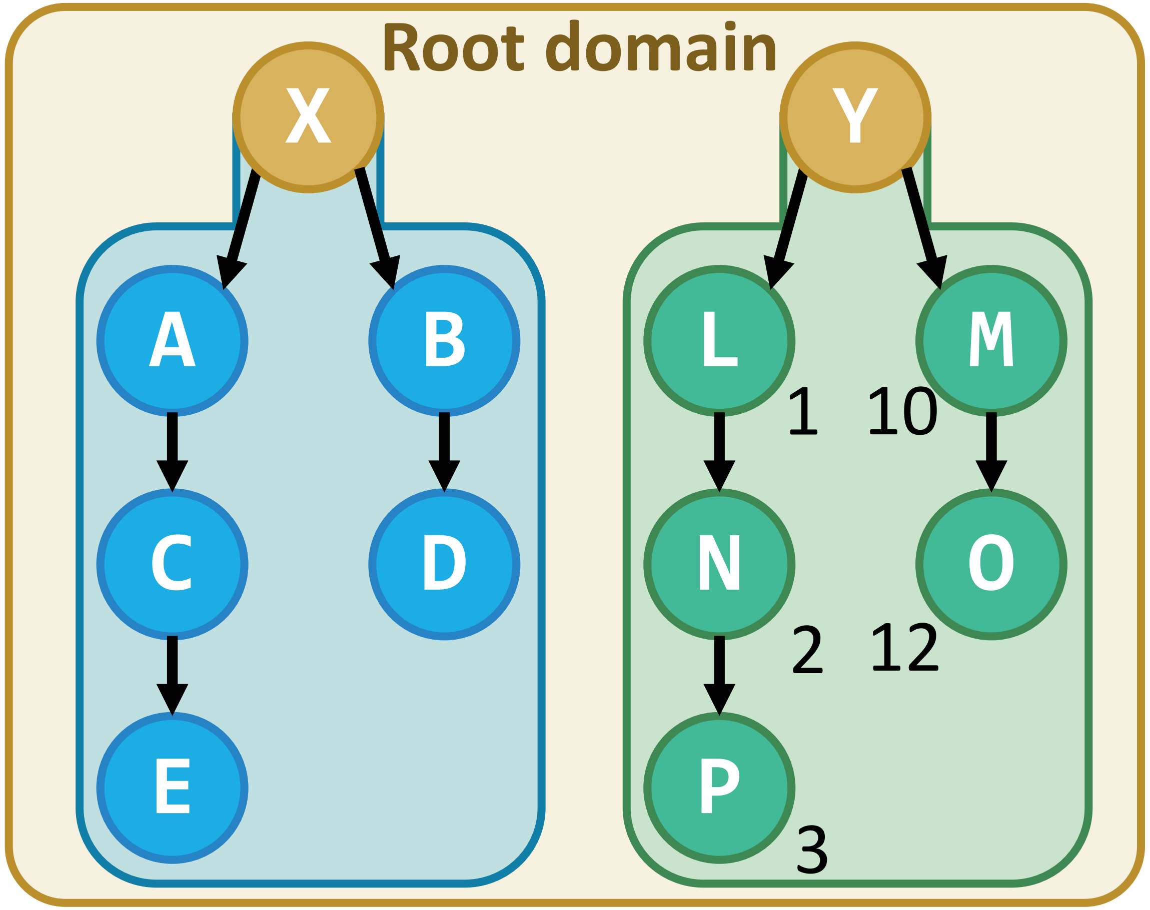
Semantics within a domain

Unordered

- Arbitrary order while respecting parent-child dependences

Timestamp-ordered

- Tasks appear to execute in increasing timestamp order
- Children appear to execute after parent



Fractal software API

Creating and enqueuing tasks

```
fractal::enqueue(function_pointer,
                 timestamp,
                 arguments...);
```

Creating sub-domains

```
fractal::create_subdomain(<domain_type>);
```

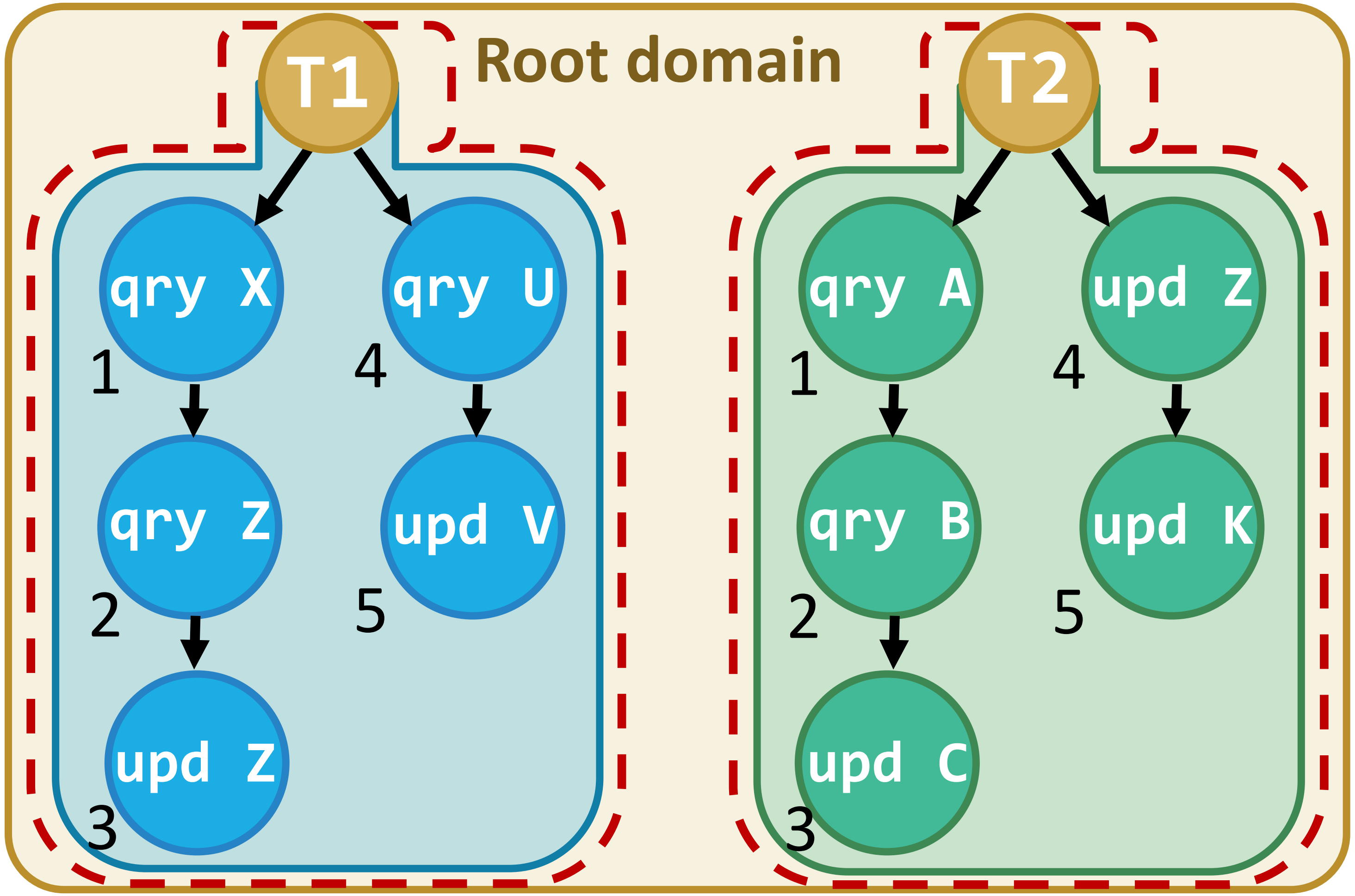
High-level programming interface, e.g.

```
forall(), callcc(), parallel_reduce()
```

Example: Database transactions in **Fractal**

TXN 1

```
query X
query Z
update Z
query U
update V
```



TXN 2

```
query A
query B
update C
update Z
update K
```

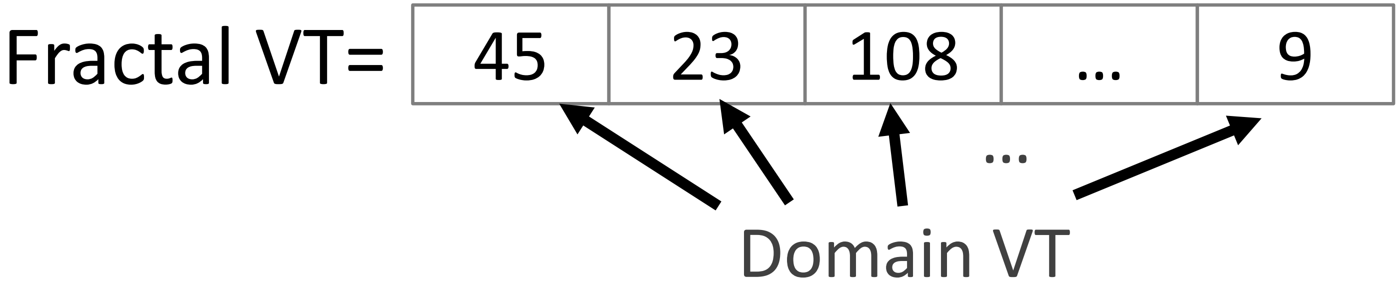
Fractal Implementation

ATOMICITY THROUGH ORDERING

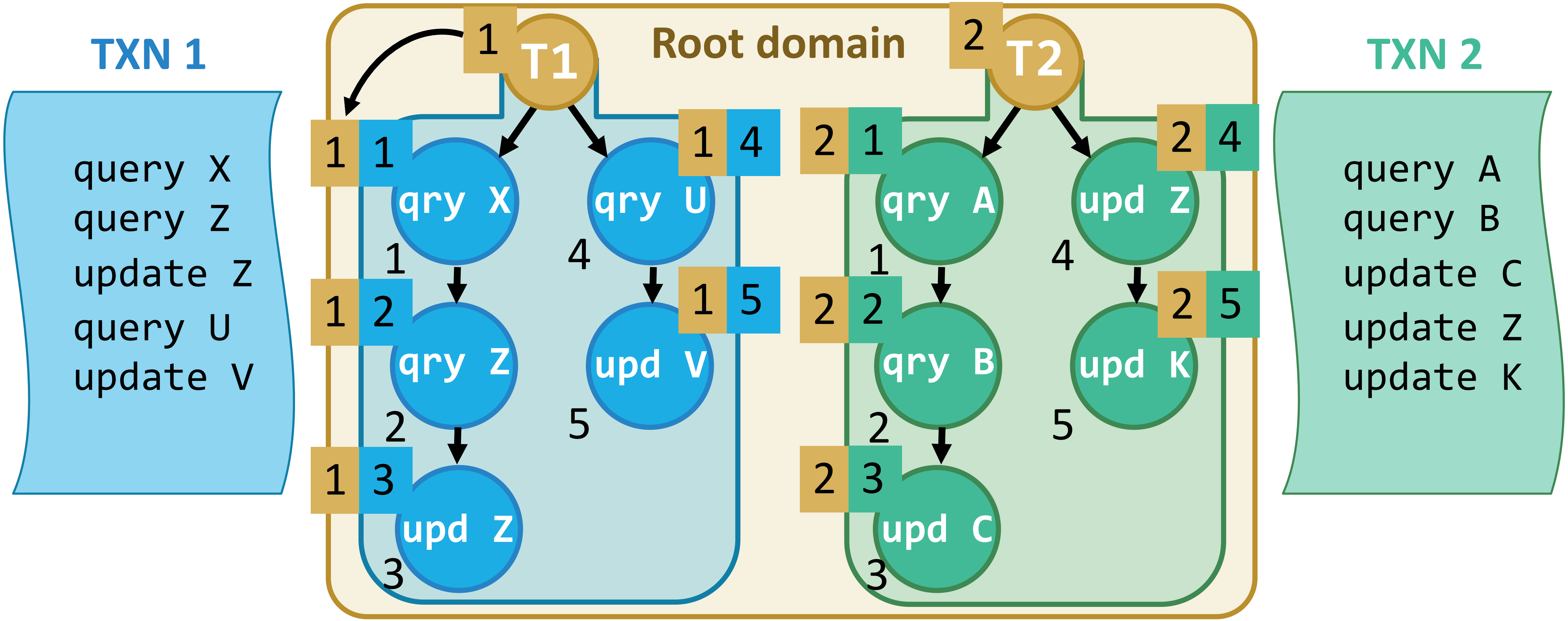
Fractal Virtual Time (VT)

Fractal assigns a *fractal virtual time (VT)* to each task

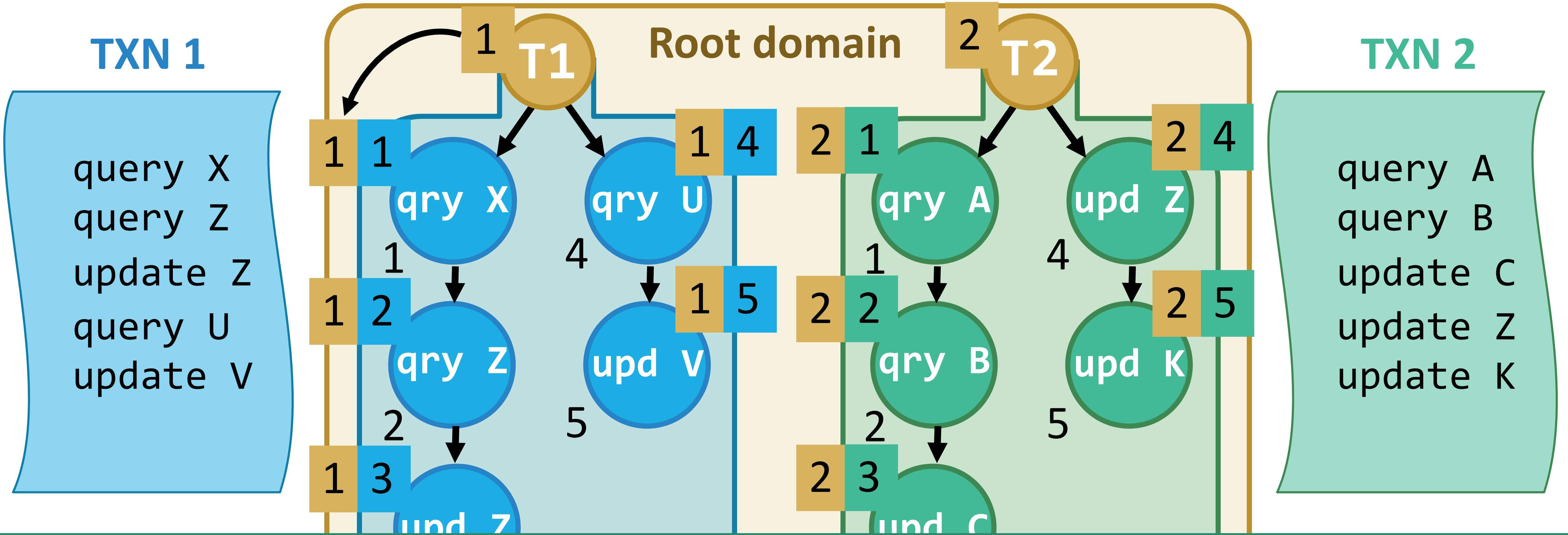
Captures the ordering of tasks across domains, within a domain



Example: Database transactions in **Fractal**



Example: Database transactions in **Fractal**



Fractal VT captures all ordering information

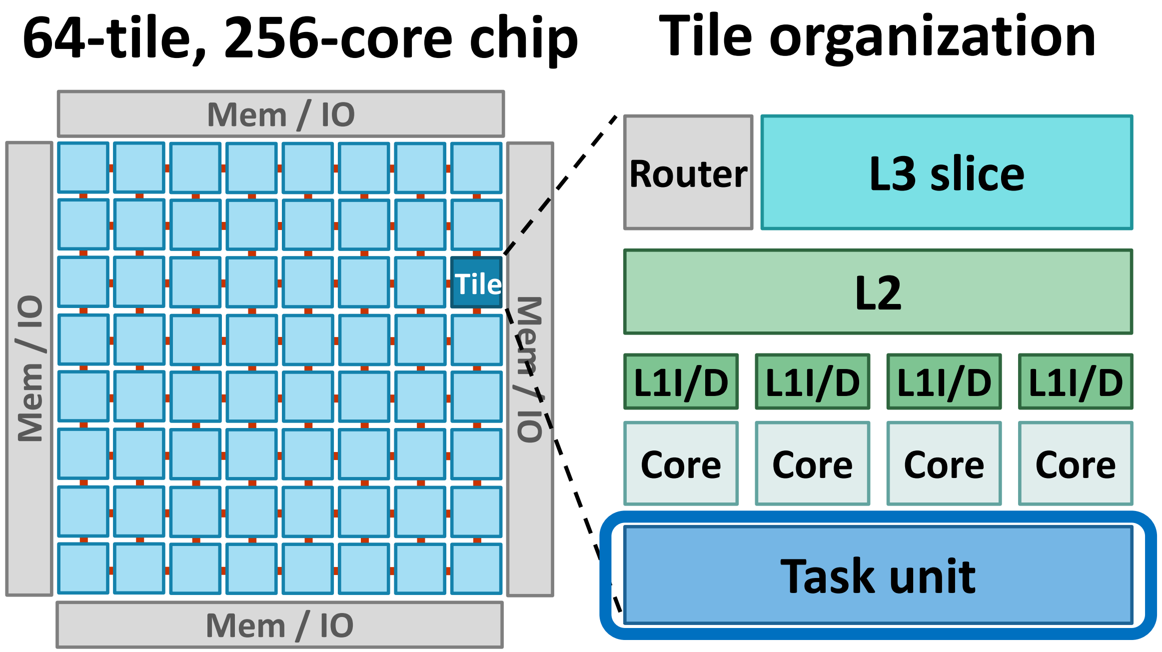
Swarm[MICRO'15] : An efficient substrate for ordered speculation

Swarm executes tasks speculatively and out of order

Large hardware task queues

Scalable ordered commits

Scalable ordered speculation



Efficiently supports tiny speculative tasks

Fractal features

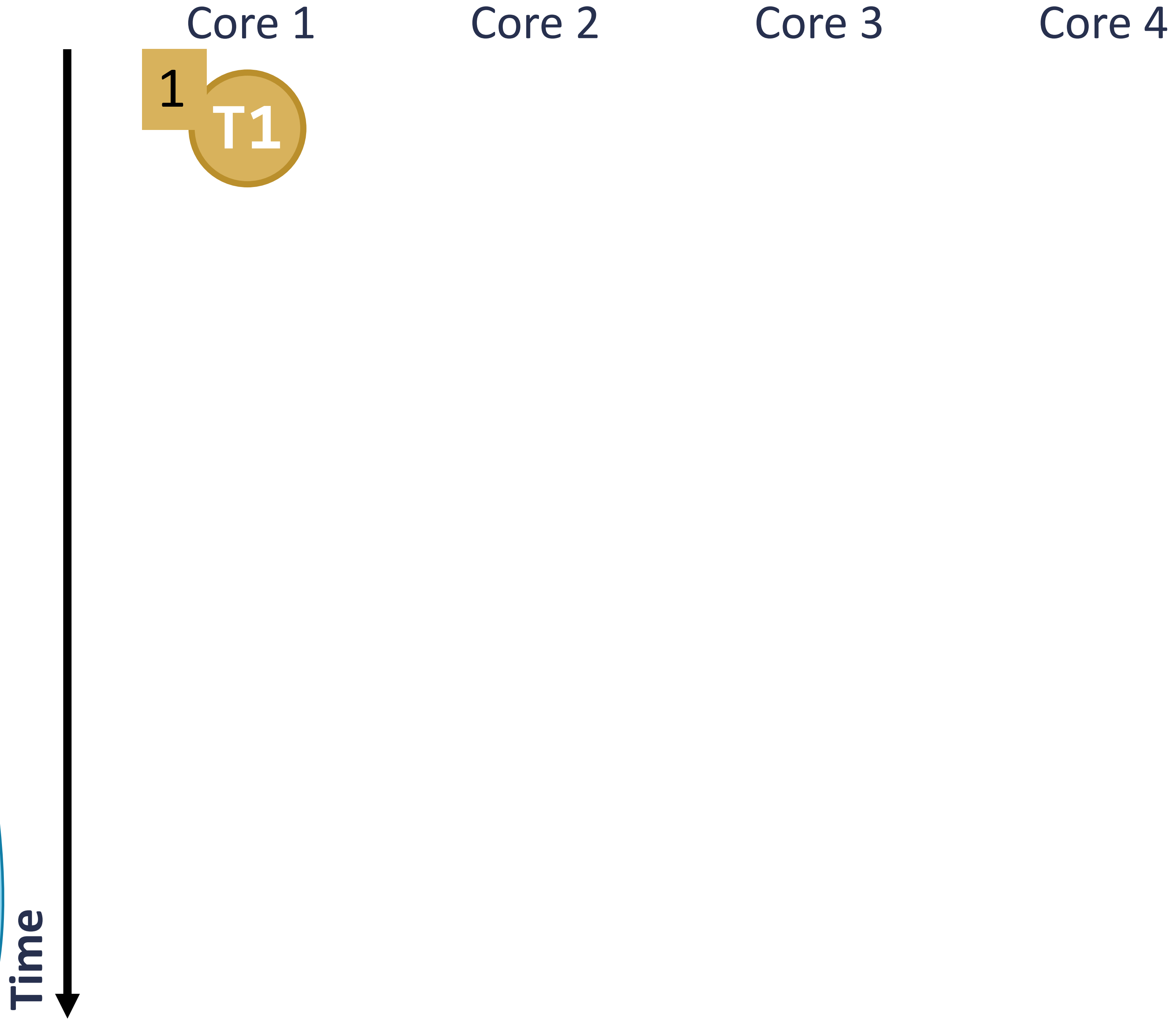
Fractal VT construction requires no centralized structures

Fractal VT assigns order dynamically

Hardware supports a few number of *concurrent depths*

- "Zooming" operations allow for unbounded nesting
- Spill tasks from shallower domains to memory
- Parallelism compounds quickly with depth

See the paper for more details!

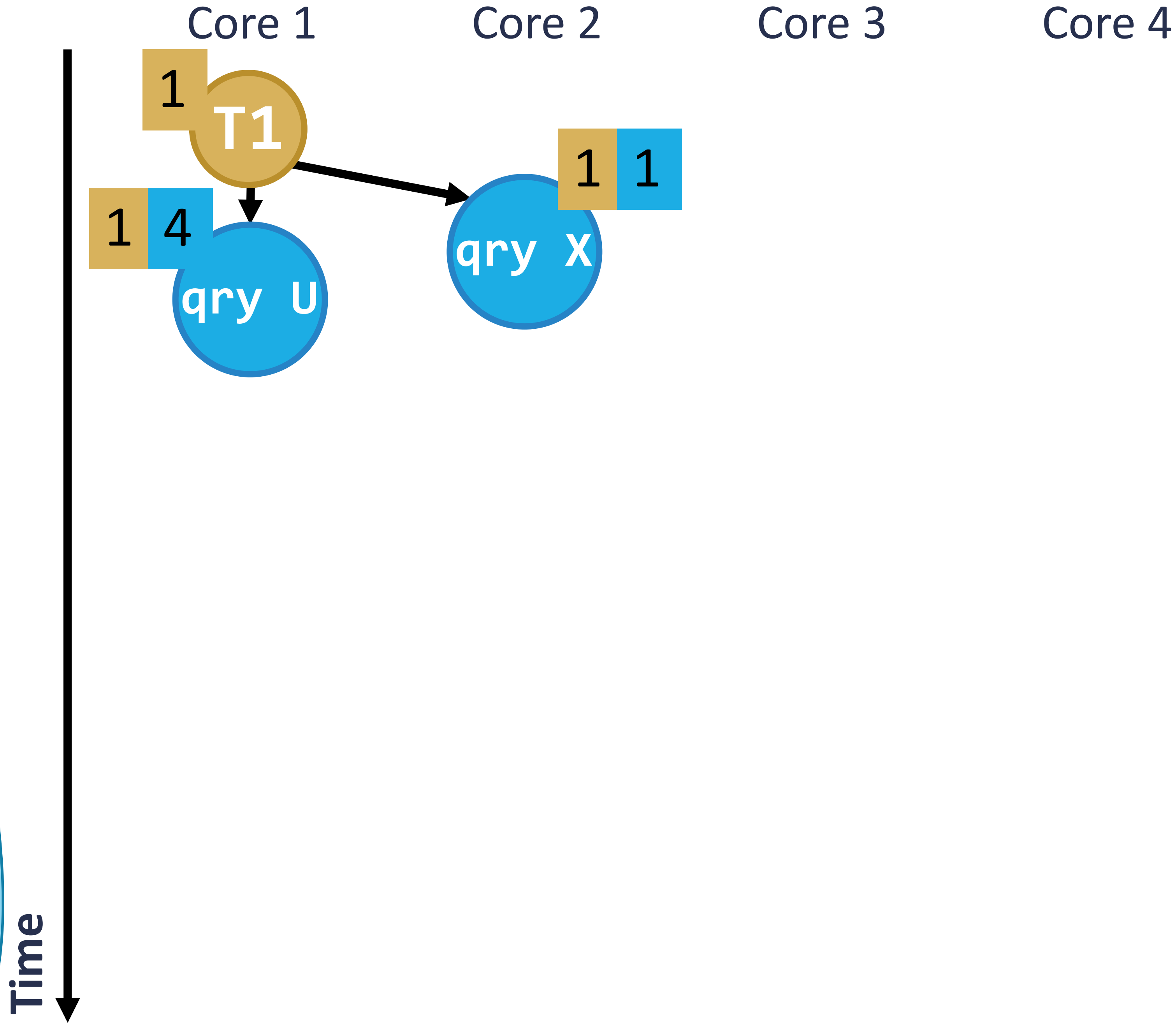


TXN 1

```
query X
query Z
update Z
query U
update V
```

TXN 2

```
query A
query B
update C
update Z
update K
```

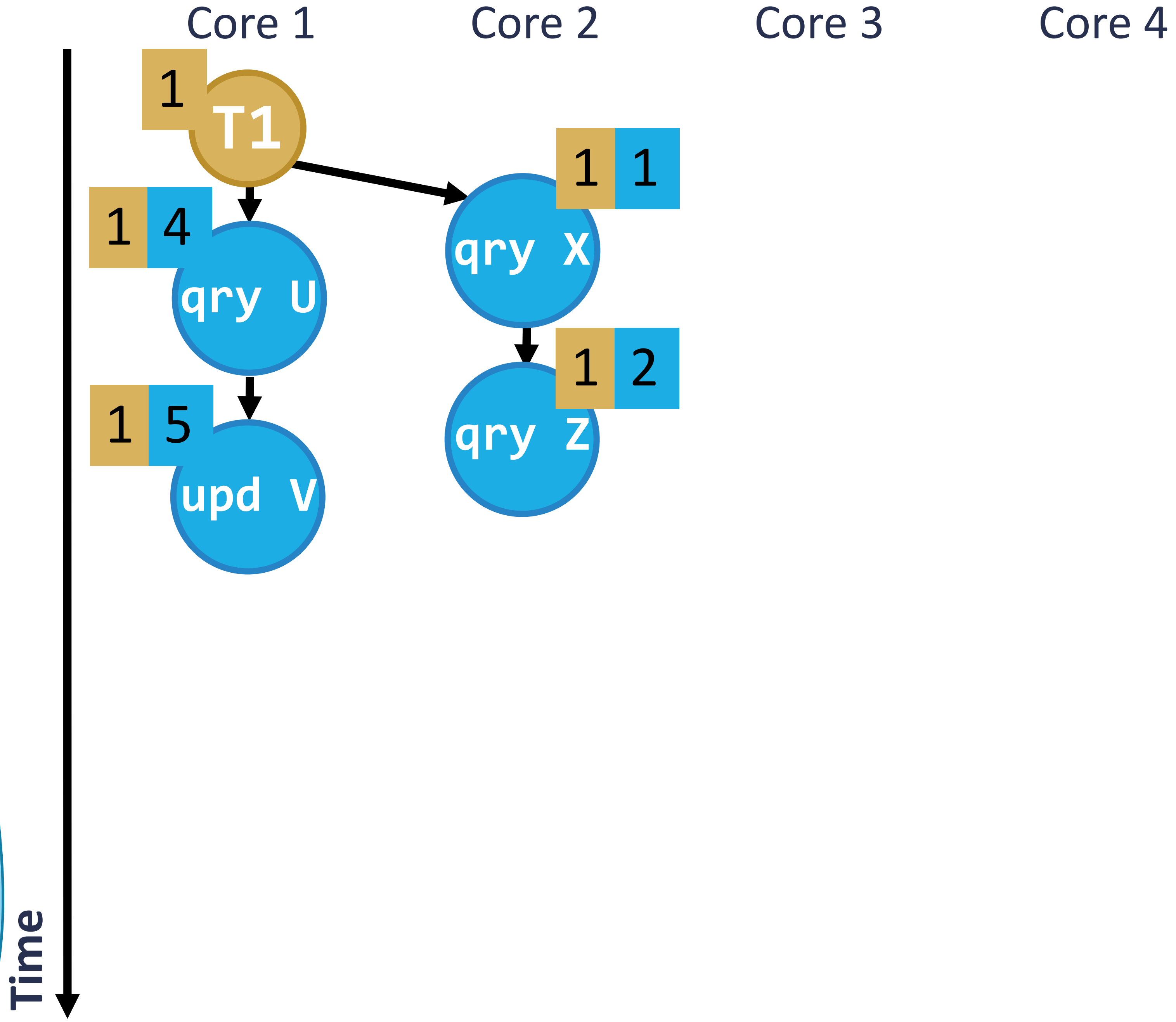


TXN 1

```
query X
query Z
update Z
query U
update V
```

TXN 2

```
query A
query B
update C
update Z
update K
```

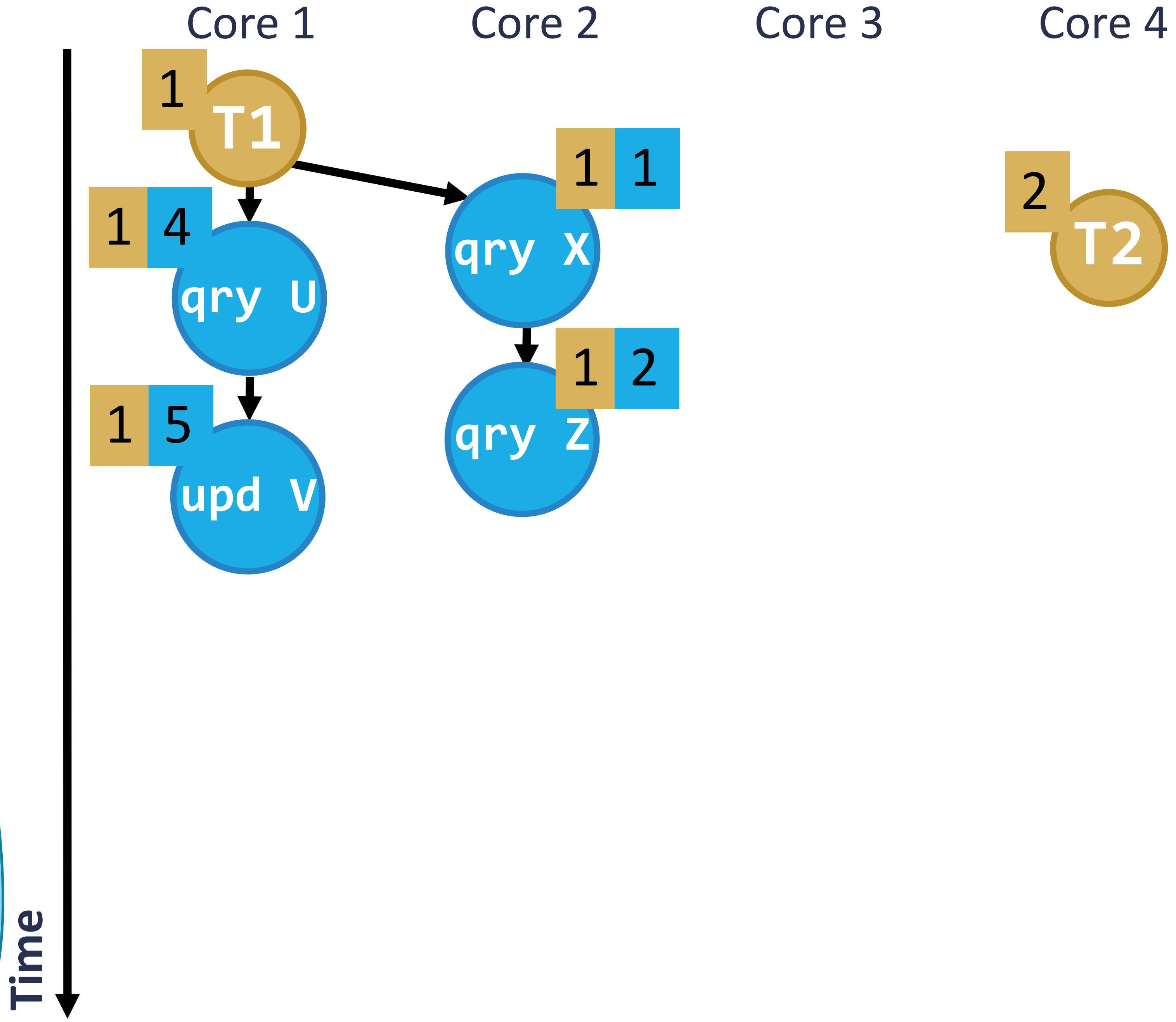


TXN 1

```
query X
query Z
update Z
query U
update V
```

TXN 2

```
query A
query B
update C
update Z
update K
```

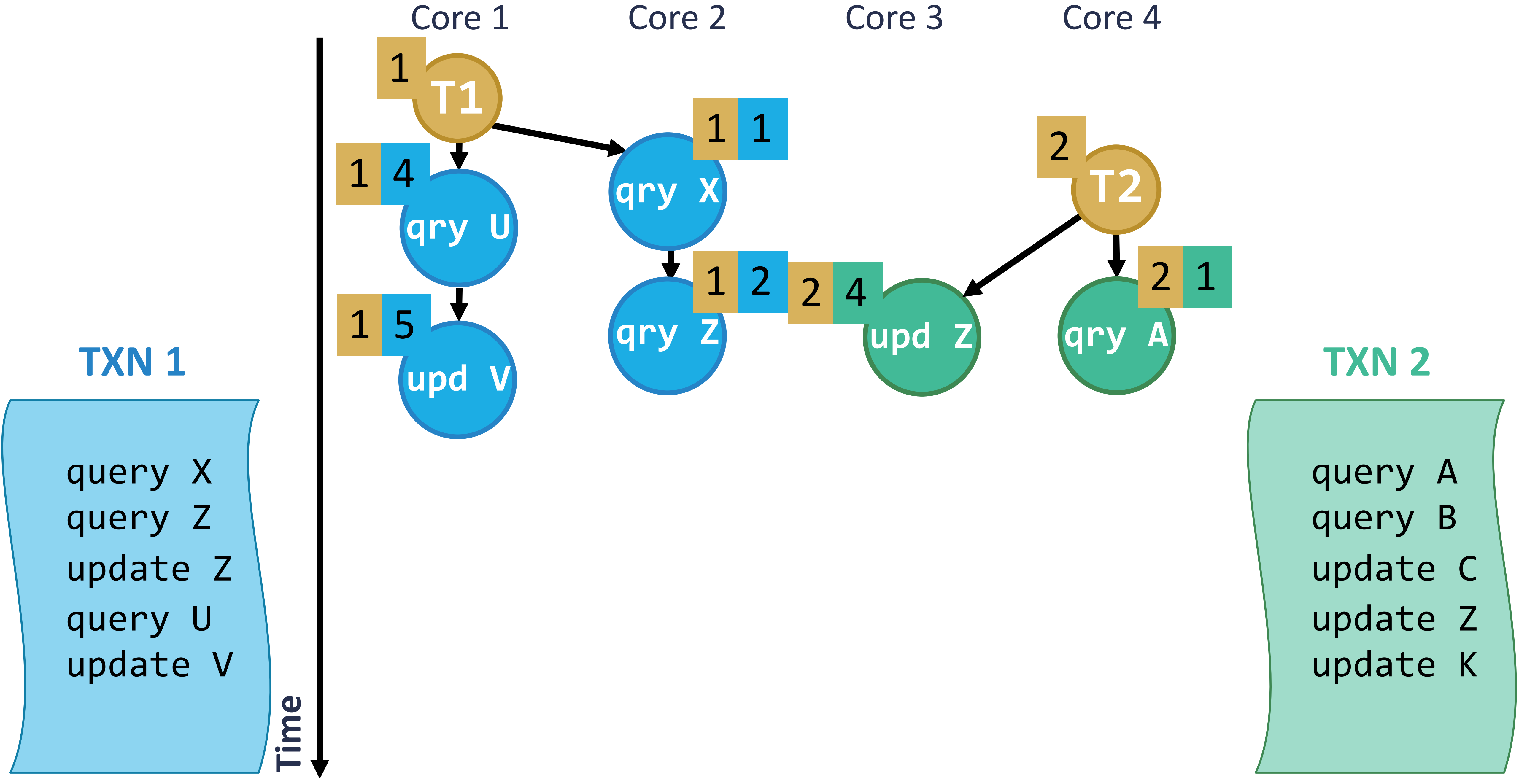


TXN 1

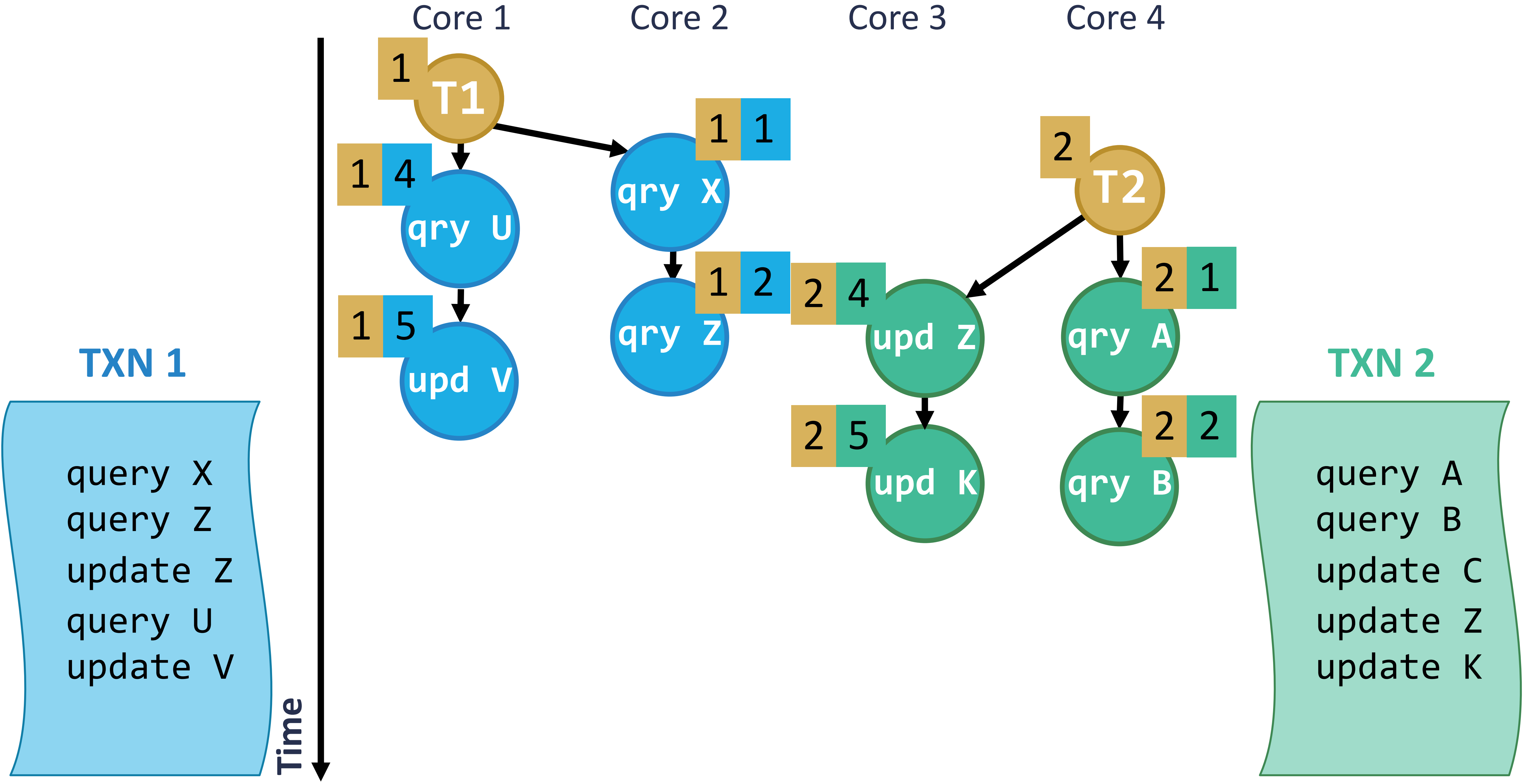
```
query X
query Z
update Z
query U
update V
```

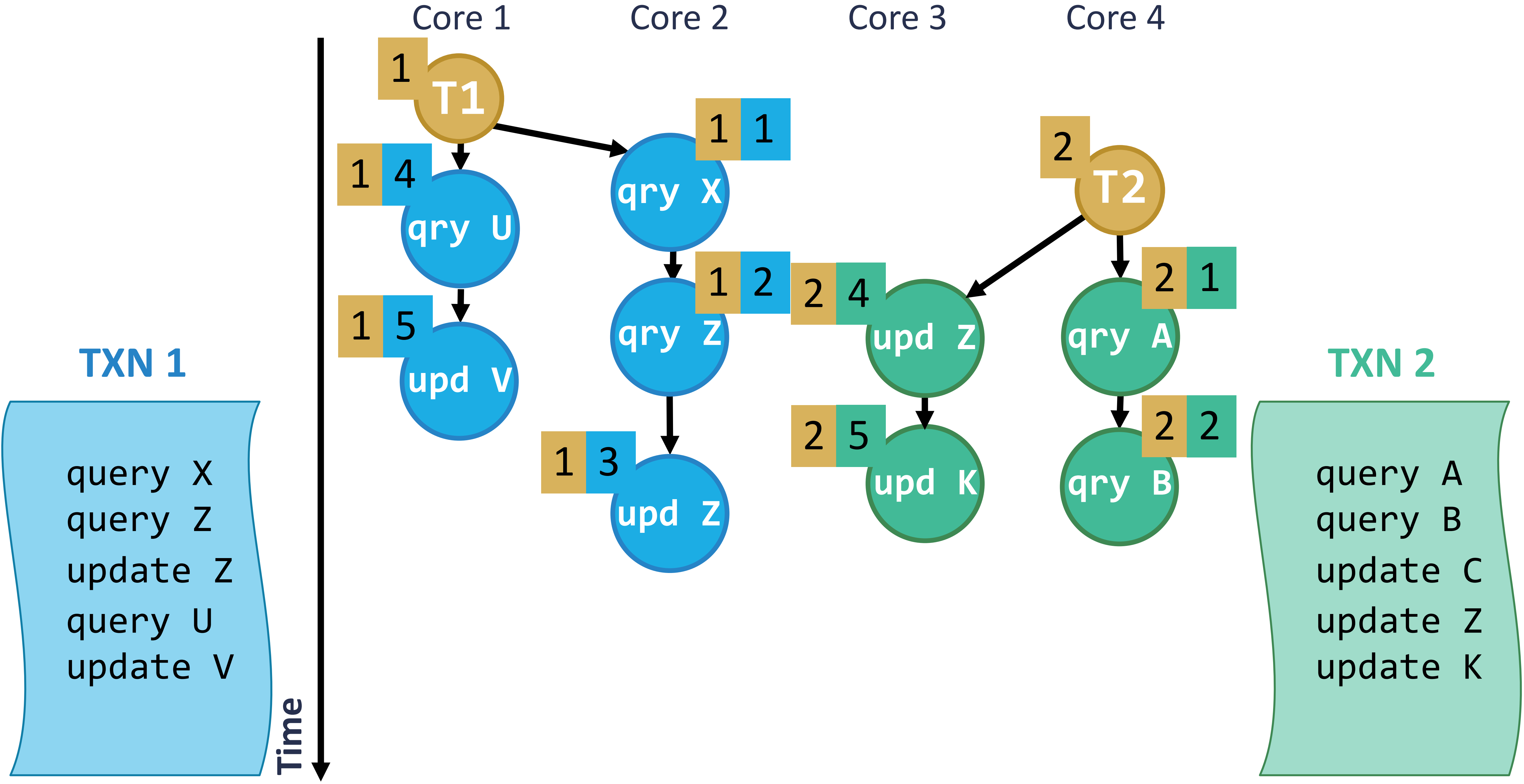
TXN 2

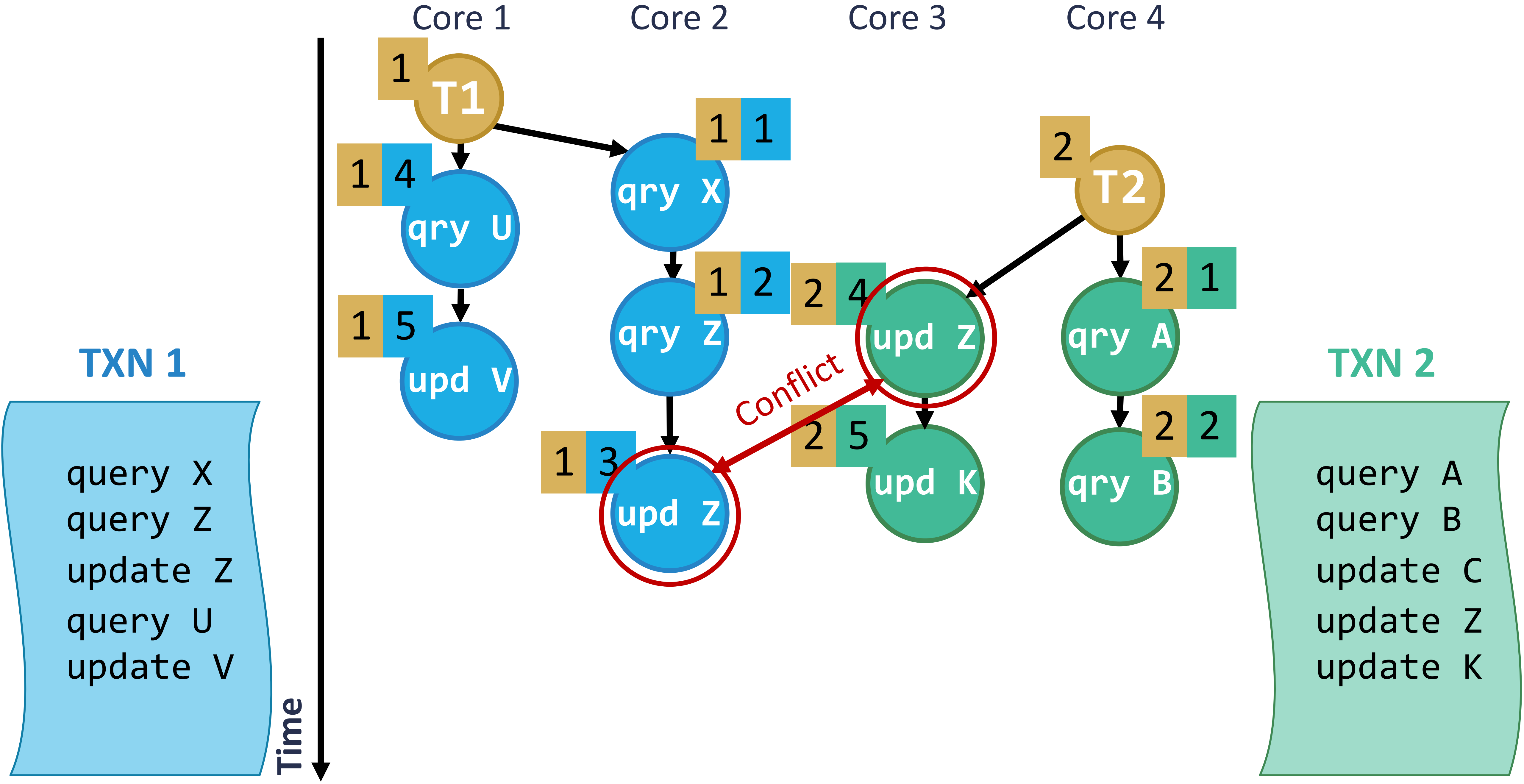
```
query A
query B
update C
update Z
update K
```



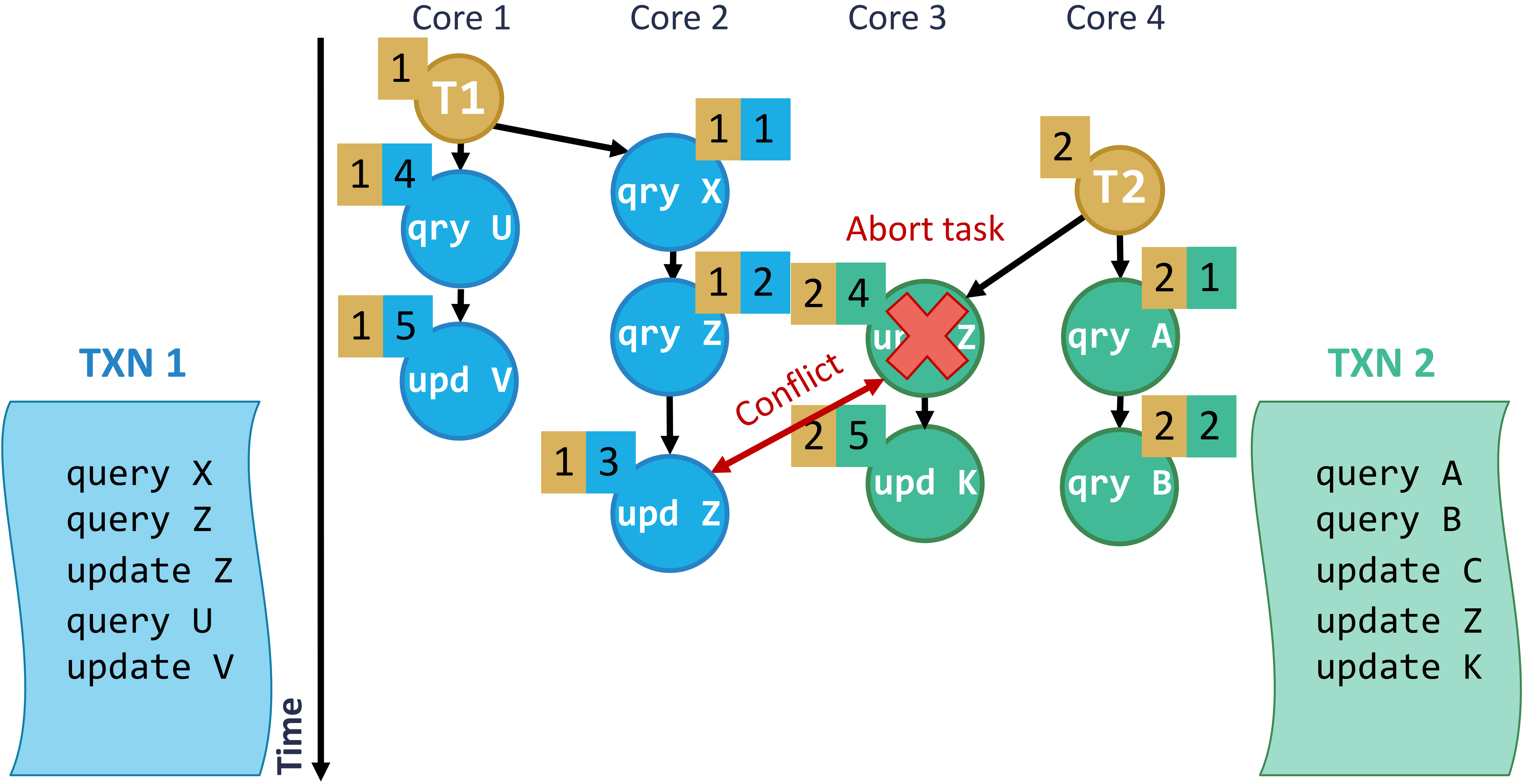
Core 1
Core 2
Core 3
Core 4
1
T1
1 1
qry X
1 4
qry U
1 5
upd V
1 2
qry Z
2
T2
2 4
upd Z
2 1
qry A
Time
TXN 1
query X
query Z
update Z
query U
update V
TXN 2
query A
query B
update C
update Z
update K

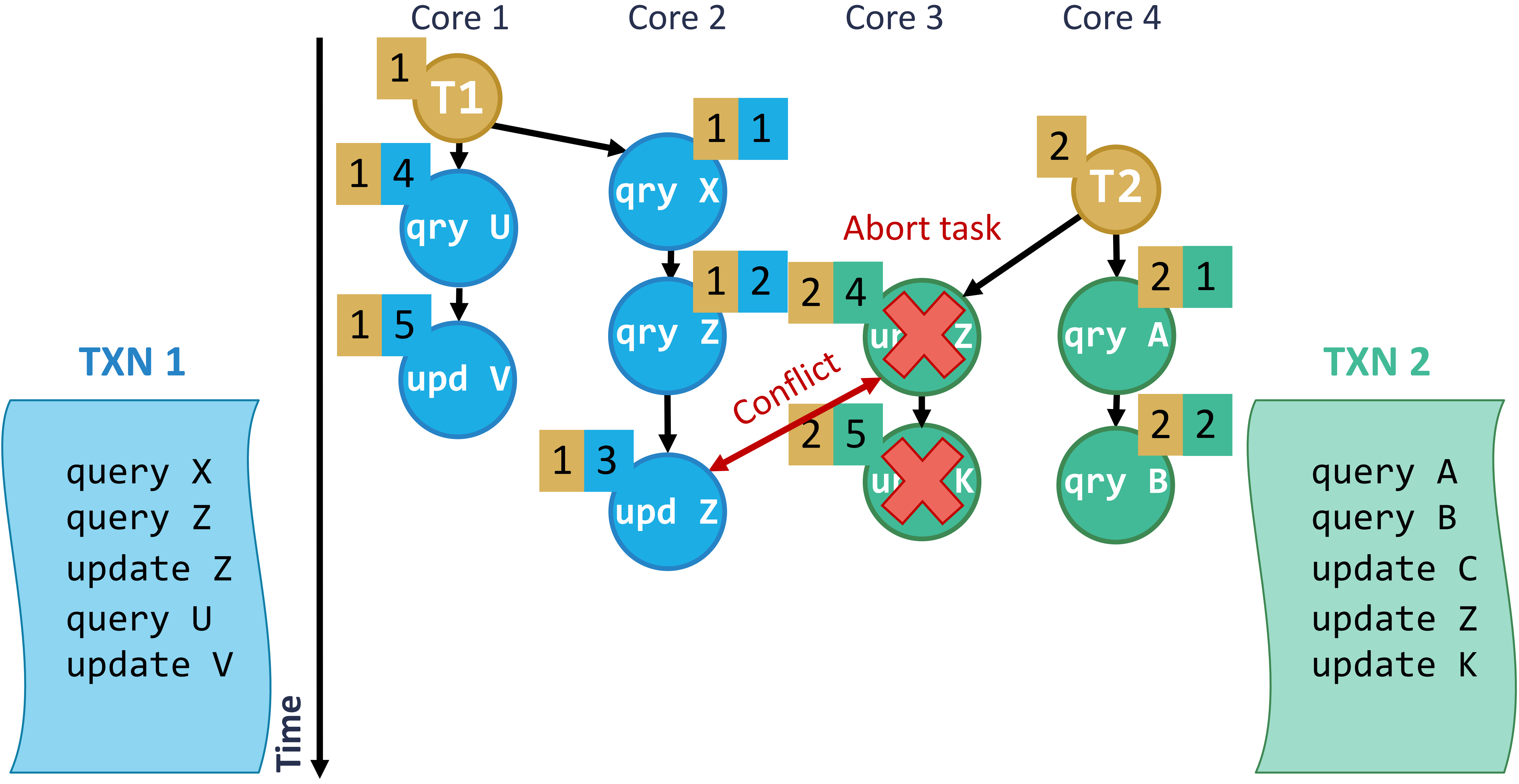

Core 1
Core 2
Core 3
Core 4
T1
T2
1
1 4
qry U
1 5
upd V
1 1
qry X
1 2
qry Z
2 4
upd Z
2 5
upd K
2
2 1
qry A
2 2
qry B
Time
TXN 1
query X
query Z
update Z
query U
update V
TXN 2
query A
query B
update C
update Z
update K

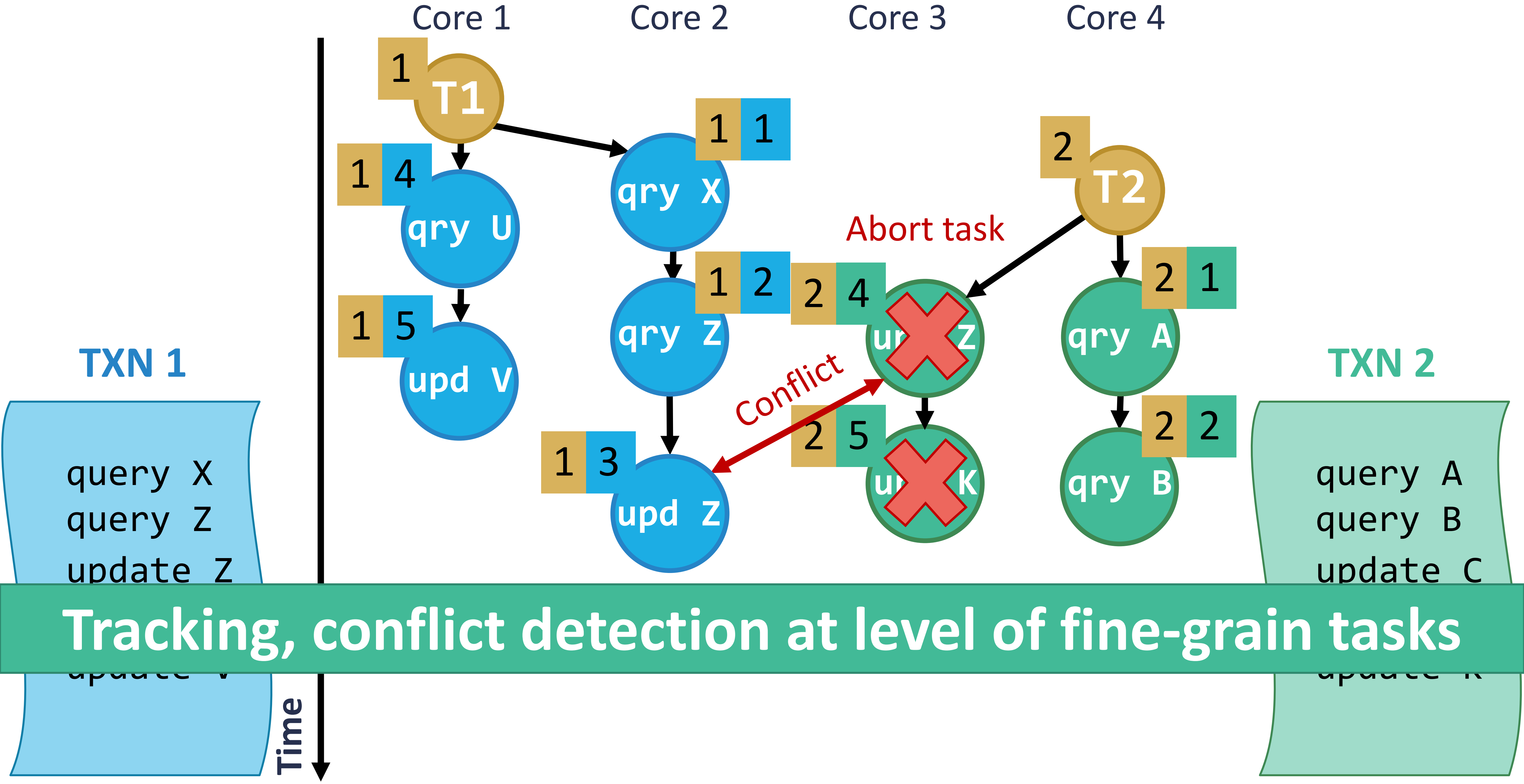

Core 1
Core 2
Core 3
Core 4
T1
1
qry U
1 4
upd V
1 5
qry X
1 1
qry Z
1 2
upd Z
1 3
T2
2
upd Z
2 4
upd K
2 5
qry A
2 1
qry B
2 2
TXN 1
query X
query Z
update Z
query U
update V
TXN 2
query A
query B
update C
update Z
update K
Time



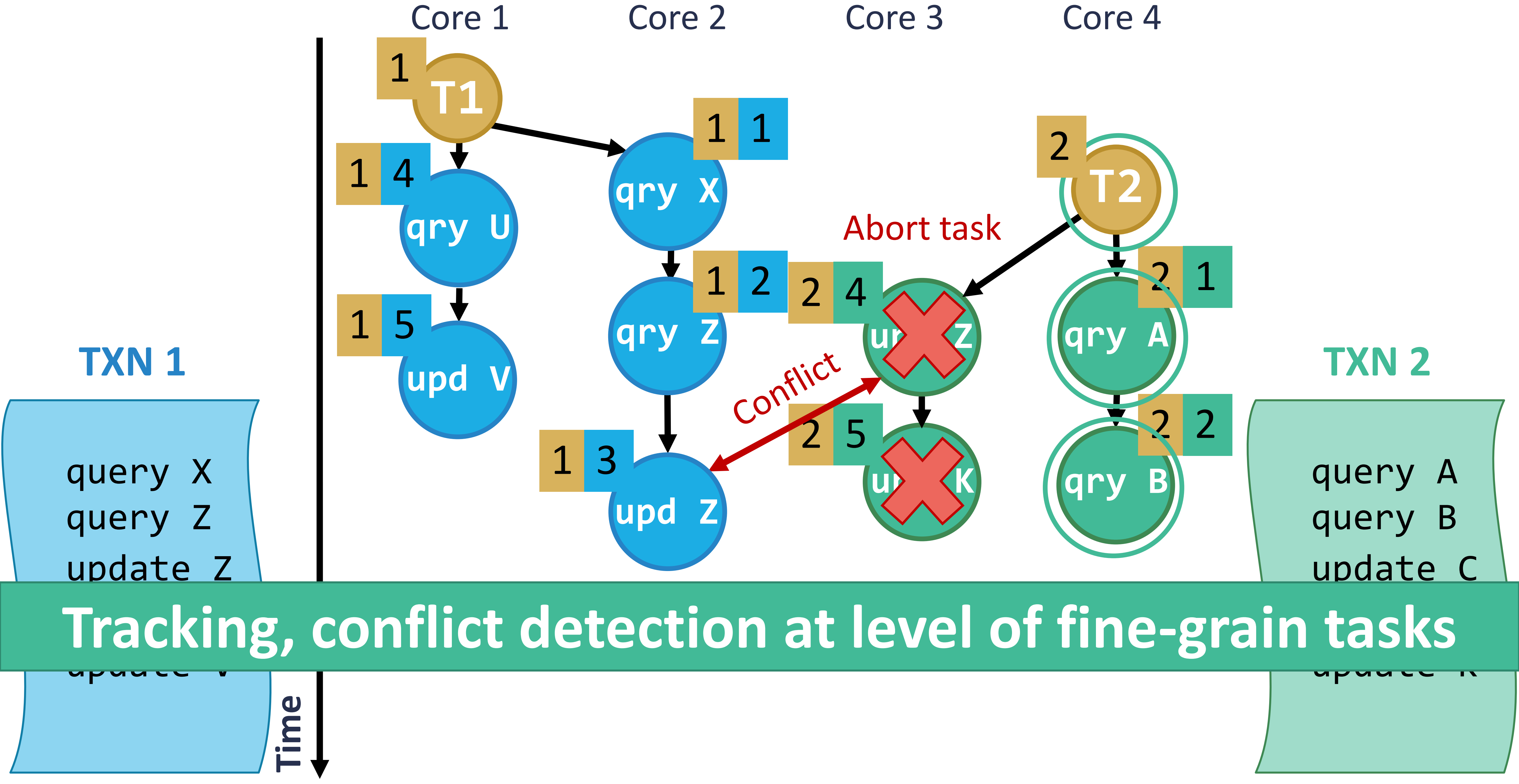
Core 1
Core 2
Core 3
Core 4
T1
T2
1
1 4
qry U
1 5
upd V
1 1
qry X
1 2
qry Z
1 3
upd Z
2
2 4
upd Z
2 5
upd K
2 1
qry A
2 2
qry B
Conflict
Time
TXN 1
query X
query Z
update Z
query U
update V
TXN 2
query A
query B
update C
update Z
update K


Core 1
Core 2
Core 3
Core 4
1
T1
1 1
qry X
1 4
qry U
1 2
qry Z
1 5
upd V
1 3
upd Z
2
T2
Abort task
2 4
upd Z
Conflict
2 5
upd K
2 1
qry A
2 2
qry B
TXN 1
query X
query Z
update Z
query U
update V
TXN 2
query A
query B
update C
update Z
update K
Time

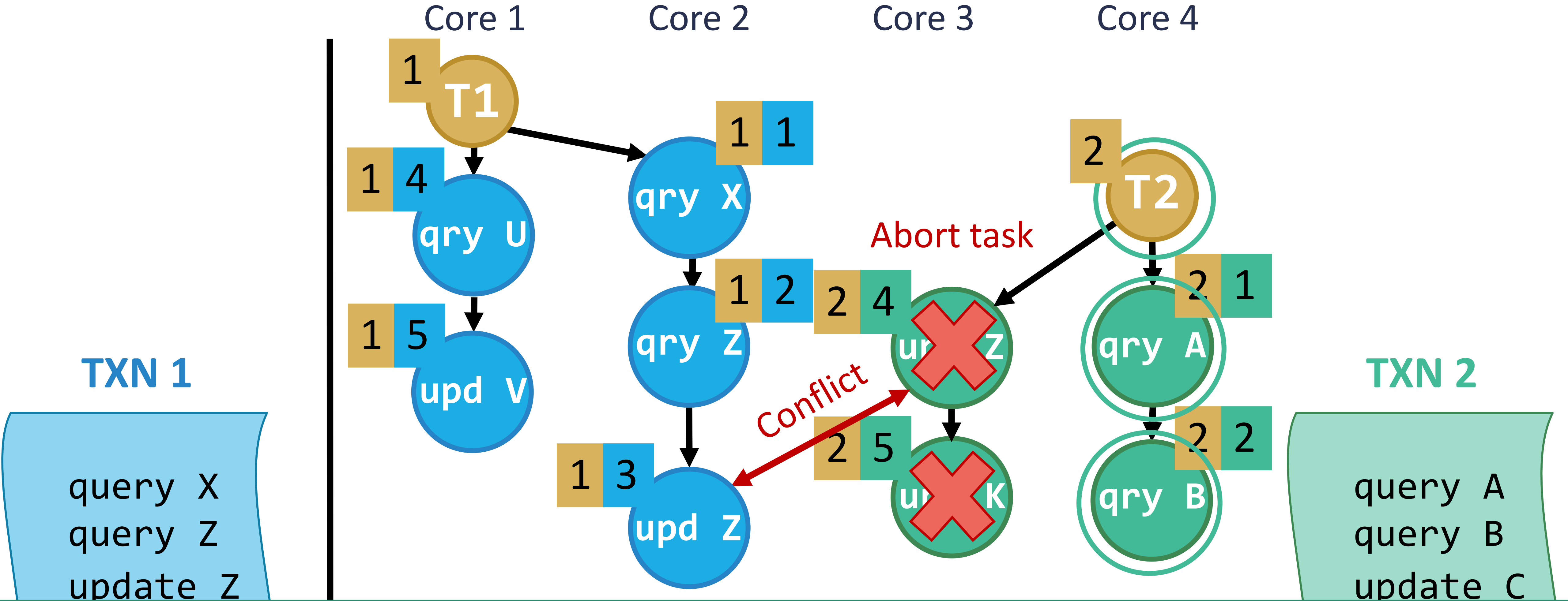

Core 1
Core 2
Core 3
Core 4
1
T1
1 1
qry X
1 4
qry U
1 5
upd V
1 2
qry Z
1 3
upd Z
2
T2
Abort task
2 4
2 5
Conflict
2 1
qry A
2 2
qry B
TXN 1
query X
query Z
update Z
query U
update V
Time
TXN 2
query A
query B
update C
update Z
update K



Tracking, conflict detection at level of fine-grain tasks

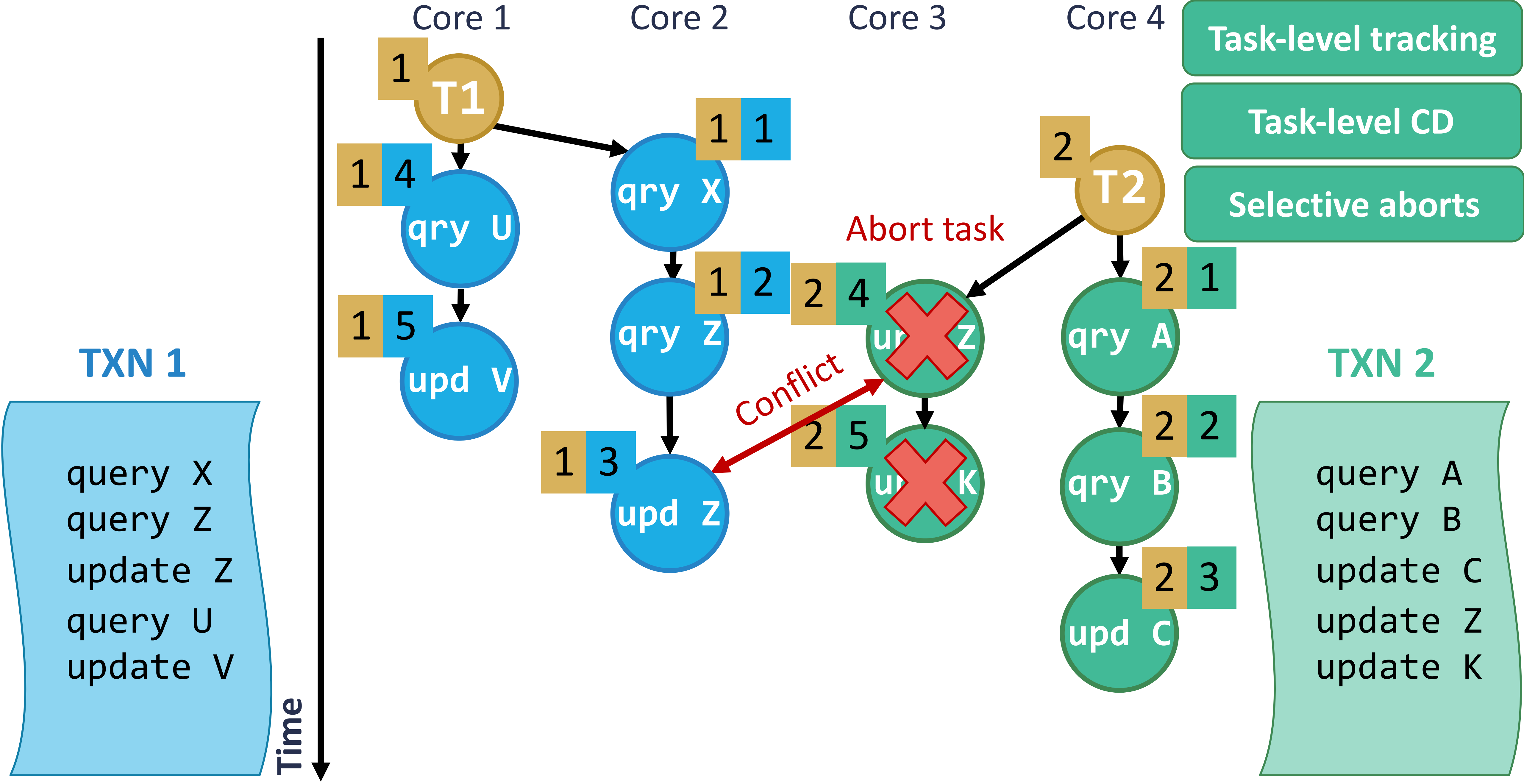


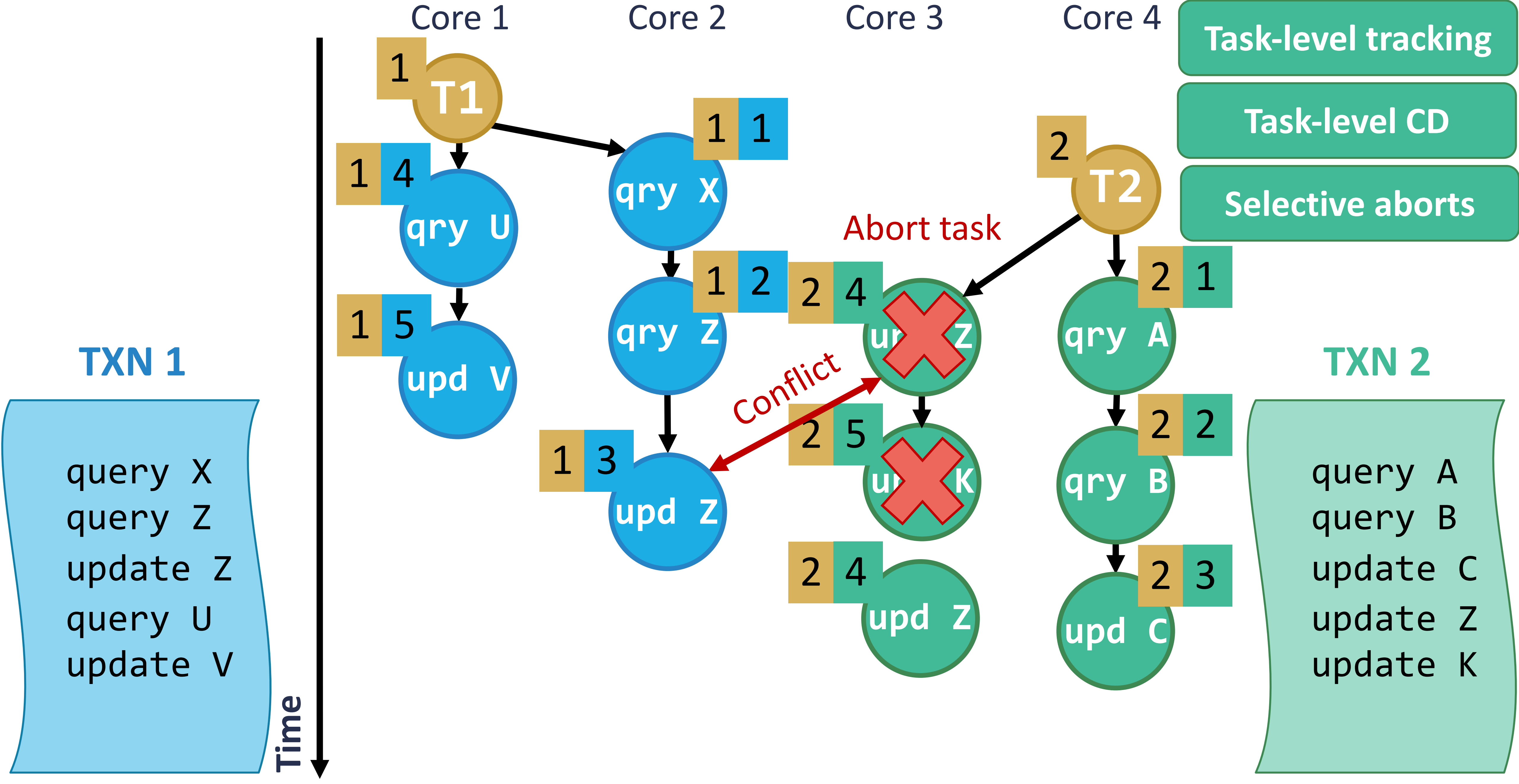
Tracking, conflict detection at level of fine-grain tasks



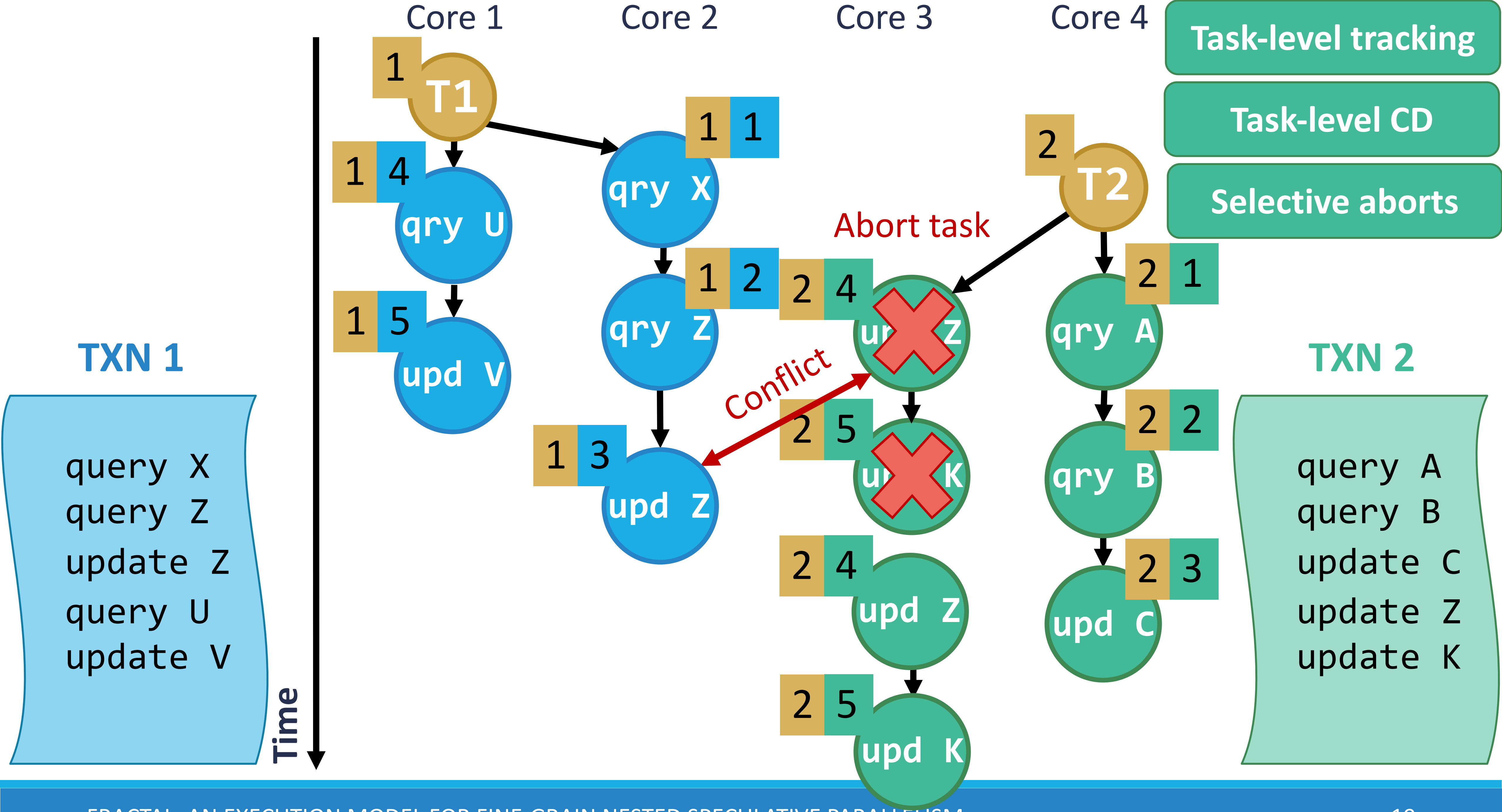
Tracking, conflict detection at level of fine-grain tasks

Selective aborts waste less work


Core 1
Core 2
Core 3
Core 4
Task-level tracking
Task-level CD
Selective aborts
T1
T2
qry U
upd V
qry X
qry Z
upd Z
qry A
qry B
upd C
Abort task
Conflict
Time
TXN 1
query X
query Z
update Z
query U
update V
TXN 2
query A
query B
update C
update Z
update K

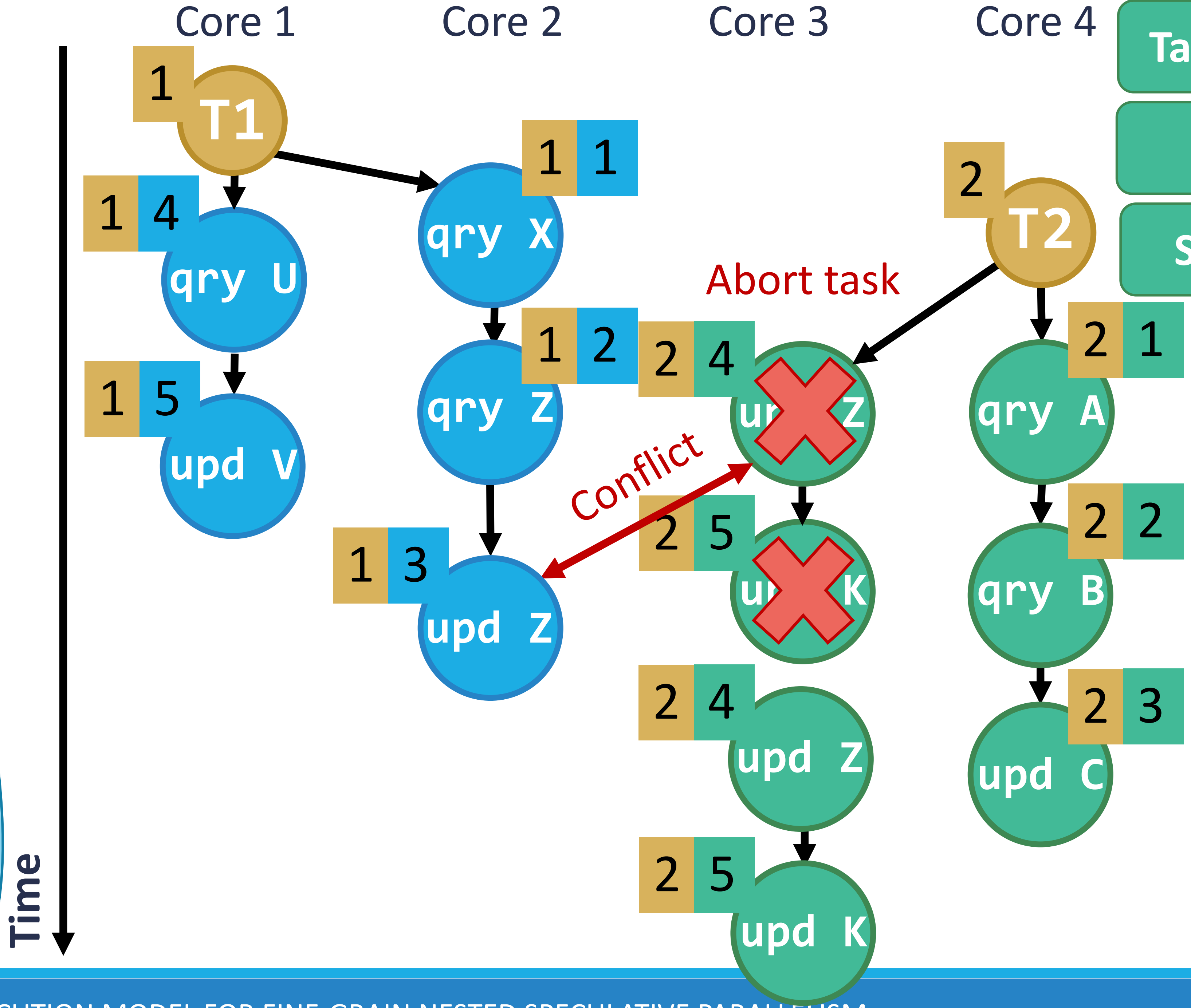


Core 1
Core 2
Core 3
Core 4
Task-level tracking
Task-level CD
Selective aborts
T1
1
1 4
qry U
1 5
upd V
1 1
qry X
1 2
qry Z
1 3
upd Z
Conflict
Abort task
2 4
2 5
2 4
upd Z
T2
2
2 1
qry A
2 2
qry B
2 3
upd C
TXN 1
query X
query Z
update Z
query U
update V
Time
TXN 2
query A
query B
update C
update Z
update K


Core 1
Core 2
Core 3
Core 4
Task-level tracking
Task-level CD
Selective aborts
T1
T2
qry U
upd V
qry X
qry Z
upd Z
upd Z
upd K
qry A
qry B
upd C
Abort task
Conflict
Time
TXN 1
query X
query Z
update Z
query U
update V
TXN 2
query A
query B
update C
update Z
update K

TXN 1

```
query X
query Z
update Z
query U
update V
```



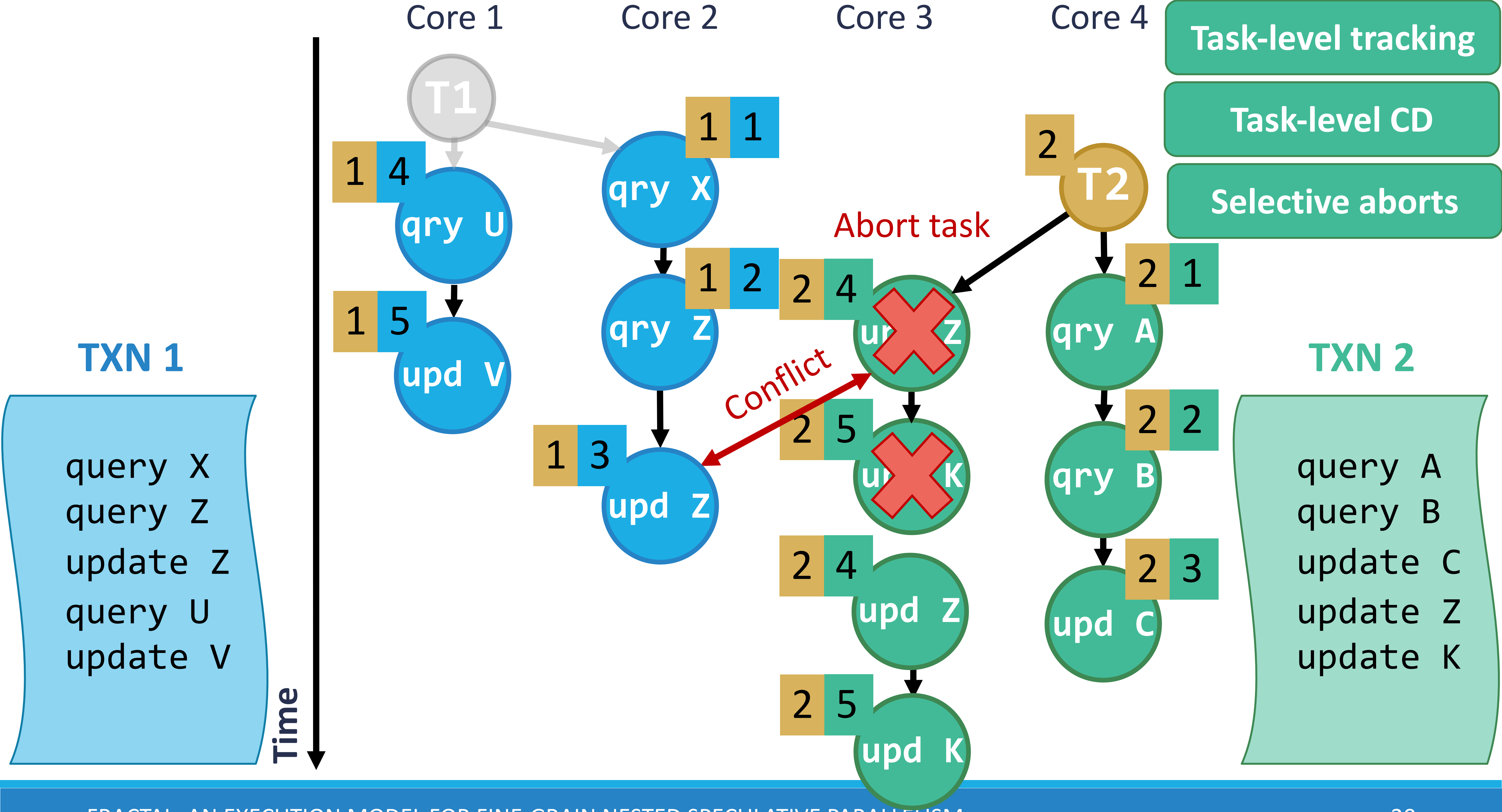
Task-level tracking

Task-level CD

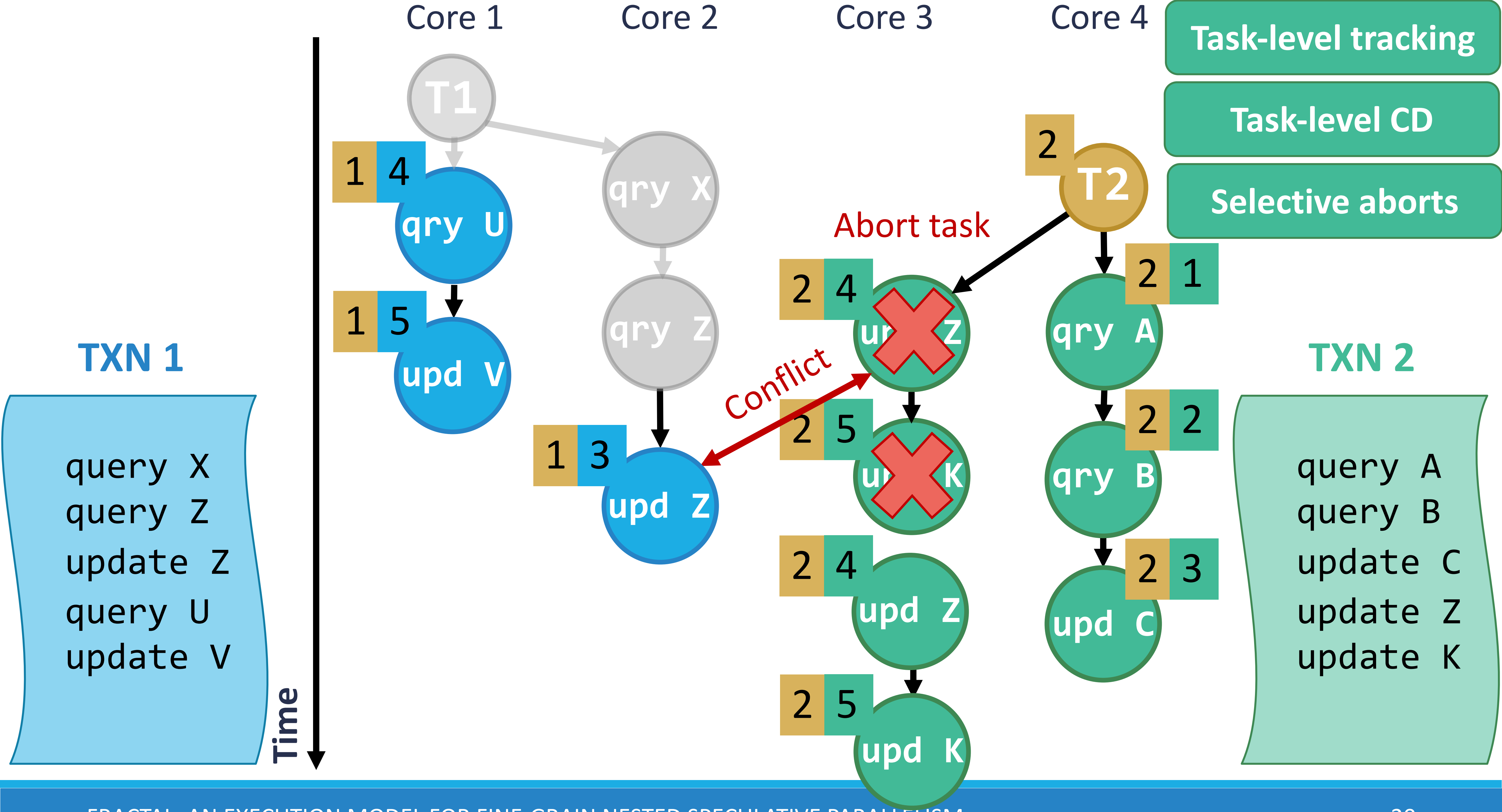
Selective aborts

TXN 2

```
query A
query B
update C
update Z
update K
```



Core 1
Core 2
Core 3
Core 4
Task-level tracking
Task-level CD
Selective aborts
T1
T2
1 4
qry U
1 5
upd V
1 1
qry X
1 2
qry Z
1 3
upd Z
2 4
upd Z
2 5
upd K
2
2 1
qry A
2 2
qry B
2 3
upd C
Abort task
Conflict
TXN 1
query X
query Z
update Z
query U
update V
Time
TXN 2
query A
query B
update C
update Z
update K

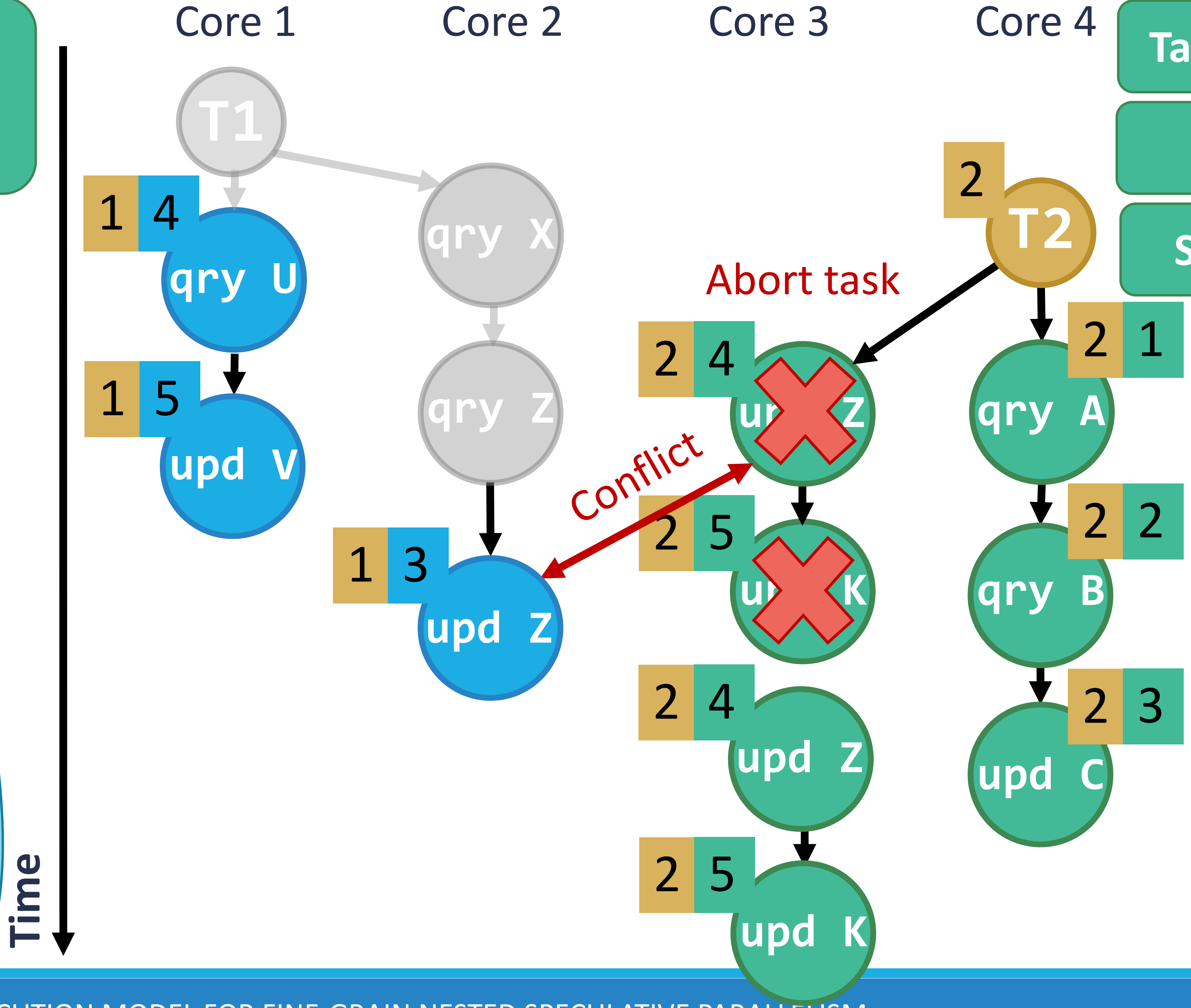

Core 1
Core 2
Core 3
Core 4
Task-level tracking
Task-level CD
Selective aborts
T1
1 4
qry U
1 5
upd V
qry X
qry Z
1 3
upd Z
Conflict
Abort task
2 4
upd Z
2 5
upd K
2 4
upd Z
2 5
upd K
2
T2
2 1
qry A
2 2
qry B
2 3
upd C
Time
TXN 1
query X
query Z
update Z
query U
update V
TXN 2
query A
query B
update C
update Z
update K

Commit parent before child completes

Task-level tracking

Task-level CD

Selective aborts



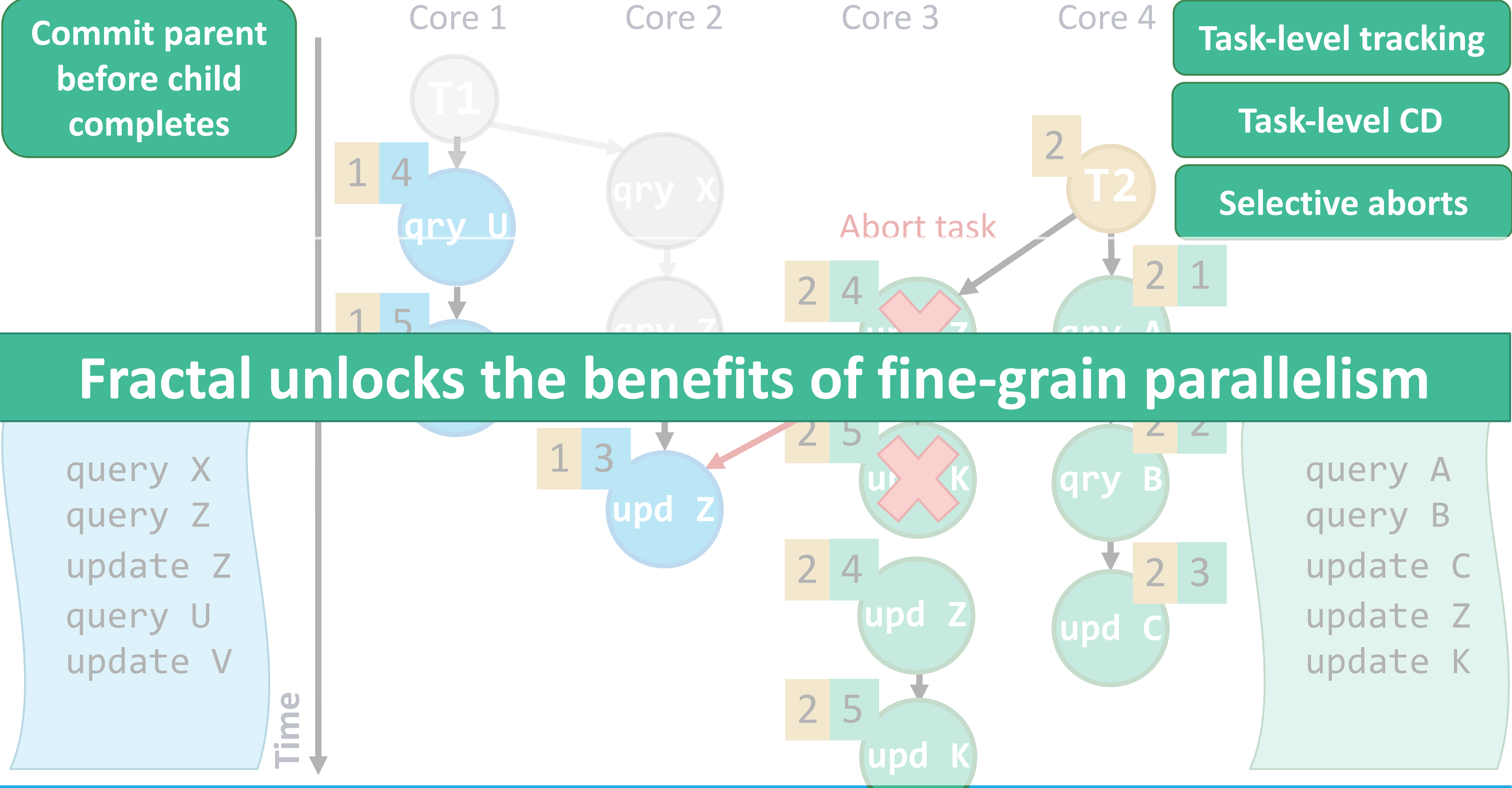
TXN 1

```
query X
query Z
update Z
query U
update V
```

TXN 2

```
query A
query B
update C
update Z
update K
```

Fractal unlocks the benefits of fine-grain parallelism



Evaluation

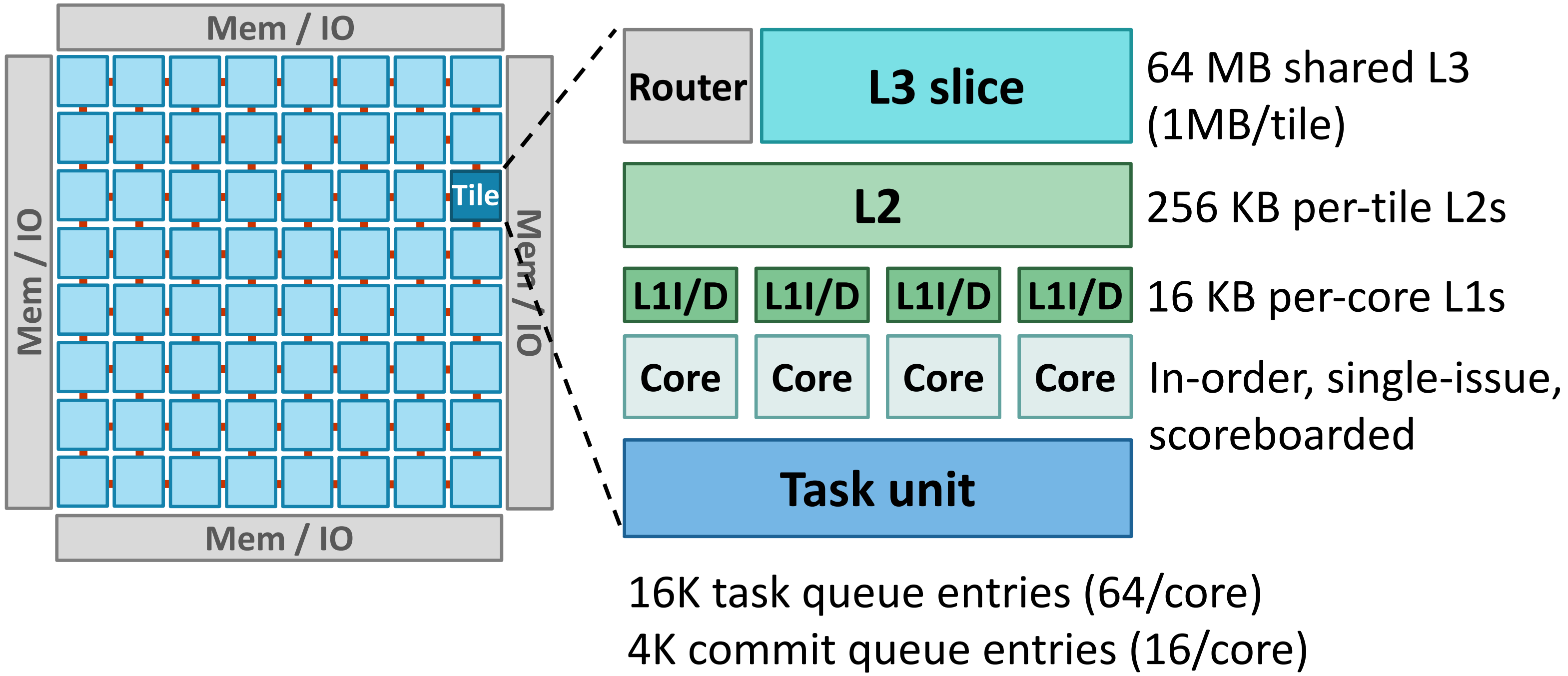
Methodology

Event-driven, Pin-based simulator

Target system: 256-core, 64-tile chip

Scalability experiments from 1–256 cores
- Scaled-down systems have fewer tiles

Applications
- Unordered (STAMP): labyrinth, bayes
- Ordered: color, msf, silo, maxflow, mis



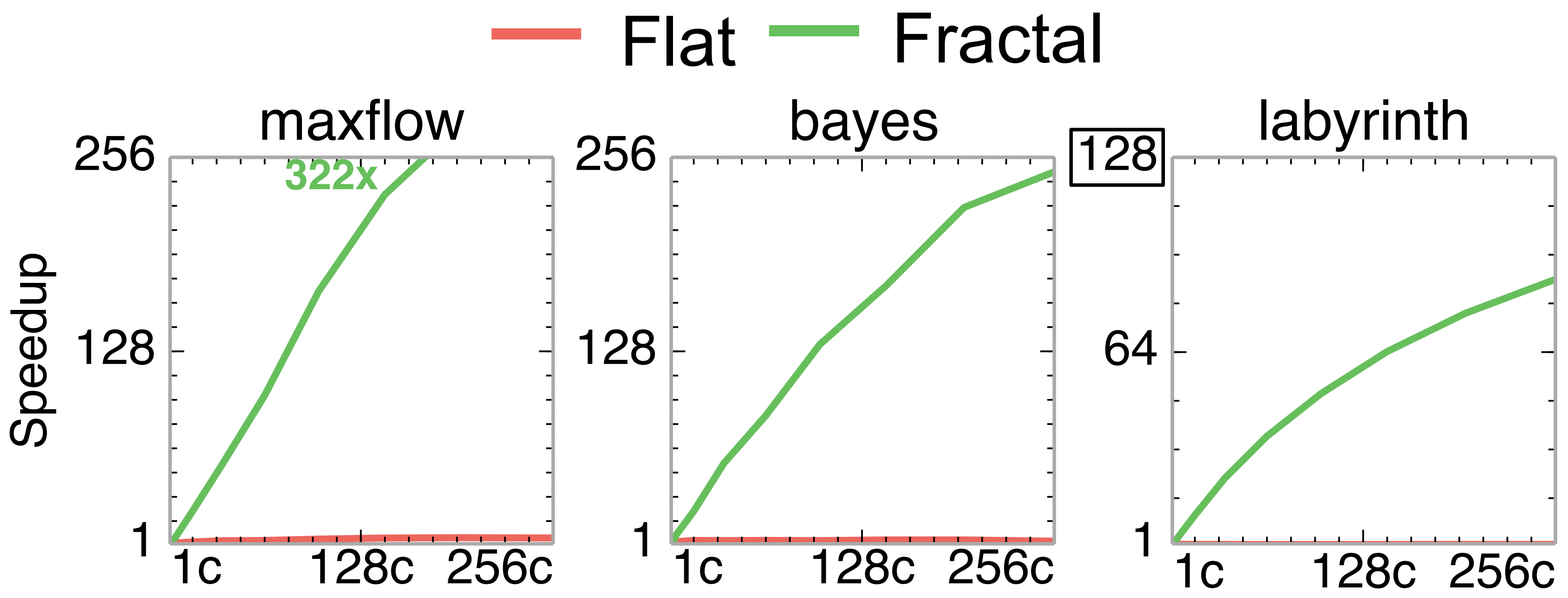
Fractal uncovers abundant nested parallelism

Flat Fractal

Large atomic tasks

Nested parallelism exposed through fine-grained tasks

Fractal uncovers abundant nested parallelism



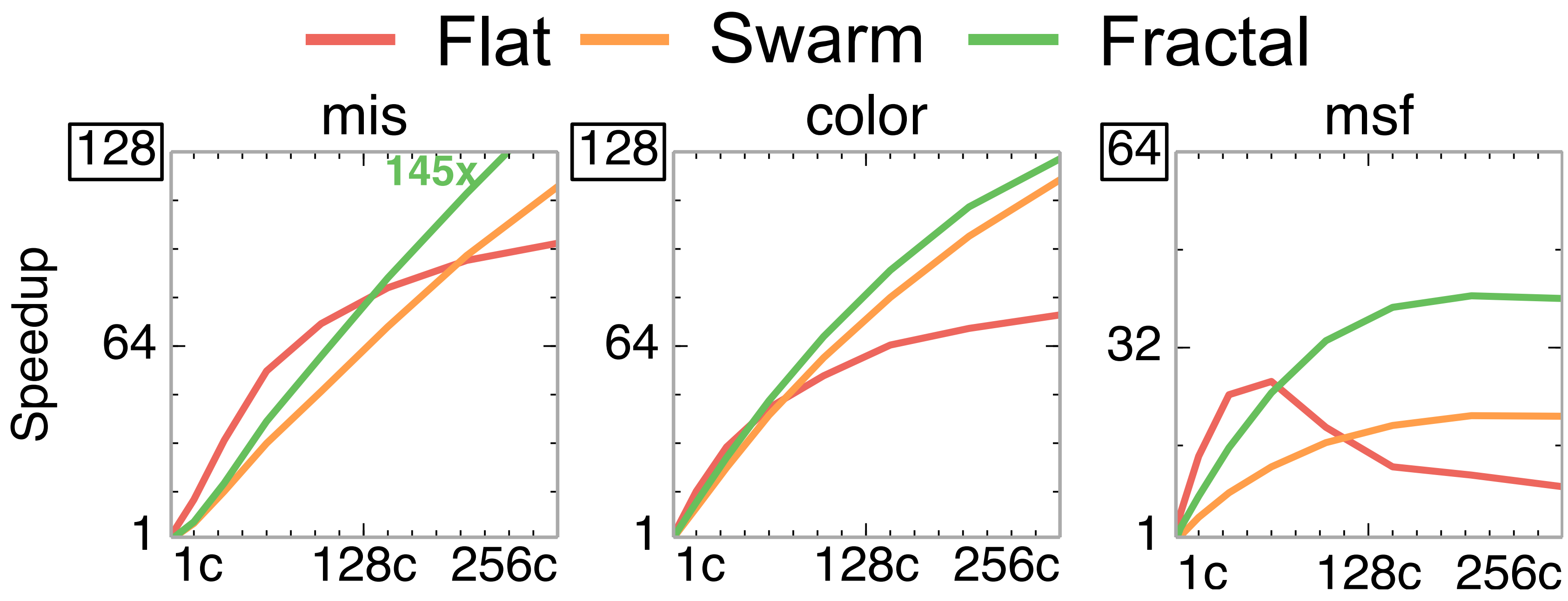
Flat
1x—4.9x

Fractal
88x—322x

Flat	3260	1.8 M	16 M
Fractal	373	3590	220

} Average task length (cycles)

Fractal avoids over-serialization



Flat
26x—98x
Swarm
21x—119x
Fractal
40x—145x

	mis	color	msf
Flat	162	633	113
Fractal	115	96	49

Average task length (cycles)

Conclusion

Speculative systems must extract nested parallelism in order to scale large, complex, real-world applications

Fractal: An execution model for fine-grain nested speculative parallelism
- Decouple atomicity from parallelism
- Guarantee atomicity by ordering tasks

Fractal unlocks the benefits of fine-grain speculative parallelism
- Parallelizes many challenging workloads
- Enables composition of speculative parallel algorithms

Thank You! Questions?

Speculative systems must extract nested parallelism in order to scale large, complex, real-world applications

Fractal: An execution model for fine-grain nested speculative parallelism
- Decouple atomicity from parallelism
- Guarantee atomicity by ordering tasks

Fractal unlocks the benefits of fine-grain speculative parallelism
- Parallelizes many challenging workloads
- Enables composition of speculative parallel algorithms