

The background of the slide is a close-up photograph of a stone wall. The stones are irregular in shape and size, with a mix of light beige, tan, and reddish-brown hues. The texture is rough and natural, with visible mortar lines between the stones.

Software-Transactional Memory in Haskell

(an overview of the implementation)



Let's start with WHY

A background image of a stone wall with light-colored, irregularly shaped stones in the upper half and darker, more uniform rectangular stones in the lower half.

FACT:

**Many modern applications
have increasingly stringent
concurrency requirements**

A background image of a stone wall with light-colored, irregularly shaped stones in the upper half and darker, more uniform rectangular stones in the lower half.

FACT:
**Commodity multicore
systems are increasingly
affordable and available**

The background of the image is a close-up of a stone wall. The stones are of various sizes and shapes, with colors ranging from light beige to dark brown. The texture is rough and uneven, with visible mortar joints.

FACT:

**The design and
implementation of correct,
efficient, and scalable
concurrent software remains
a daunting task**

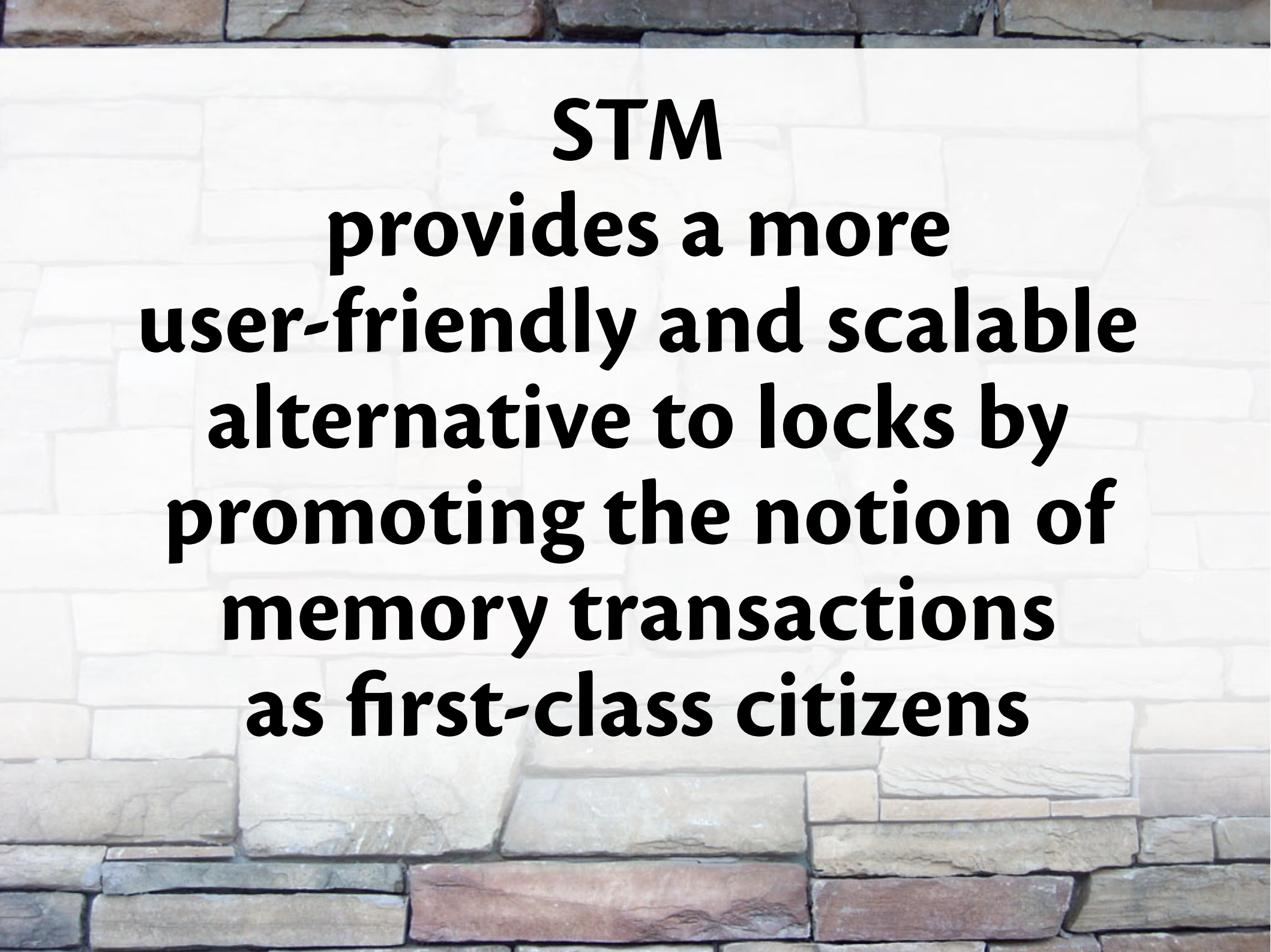
A background image of a stone wall with light-colored rectangular stones in the center and darker, more irregular stones at the top and bottom.

Haskell to the rescue!

Meet STM

The background of the slide is a close-up photograph of a stone wall. The wall is composed of irregularly shaped stones in various shades of beige, tan, and light brown. The stones are laid in a traditional pattern, with some larger flat stones and some smaller, more angular pieces. The lighting is even, highlighting the textures and colors of the stone.


STM
protects shared state in
concurrent programs



STM
provides a more
user-friendly and scalable
alternative to locks by
promoting the notion of
memory transactions
as first-class citizens

The background of the image is a close-up of a stone wall. The stones are of various sizes and colors, including shades of tan, beige, and reddish-brown. They are arranged in a rough, irregular pattern. In the center of the image, there is a large, white rectangular area that serves as a backdrop for the text.

**Transactions, like many of
the best ideas in computer
science, originated in the
data engineering world**

A background image of a stone wall. The top section features a row of dark, irregular stones. Below this is a large area of light-colored, rectangular stones arranged in a regular pattern. The bottom section consists of a row of reddish-brown, irregular stones.

**Transactions
are one of the foundations of
database technology**

**Full-fledged transactions
are defined by the
ACID properties
Memory transactions use
two of them (A+I)**

The background of the slide is a close-up photograph of a stone wall. The wall is composed of irregularly shaped stones in various shades of beige, tan, and light brown. The stones are laid in a roughly horizontal pattern, with some stones protruding more than others, creating a textured surface. The lighting is even, highlighting the natural grain and edges of the stones.

**Transactions provide
atomicity and isolation
guarantees**



**Strong atomicity
means all-or-nothing**



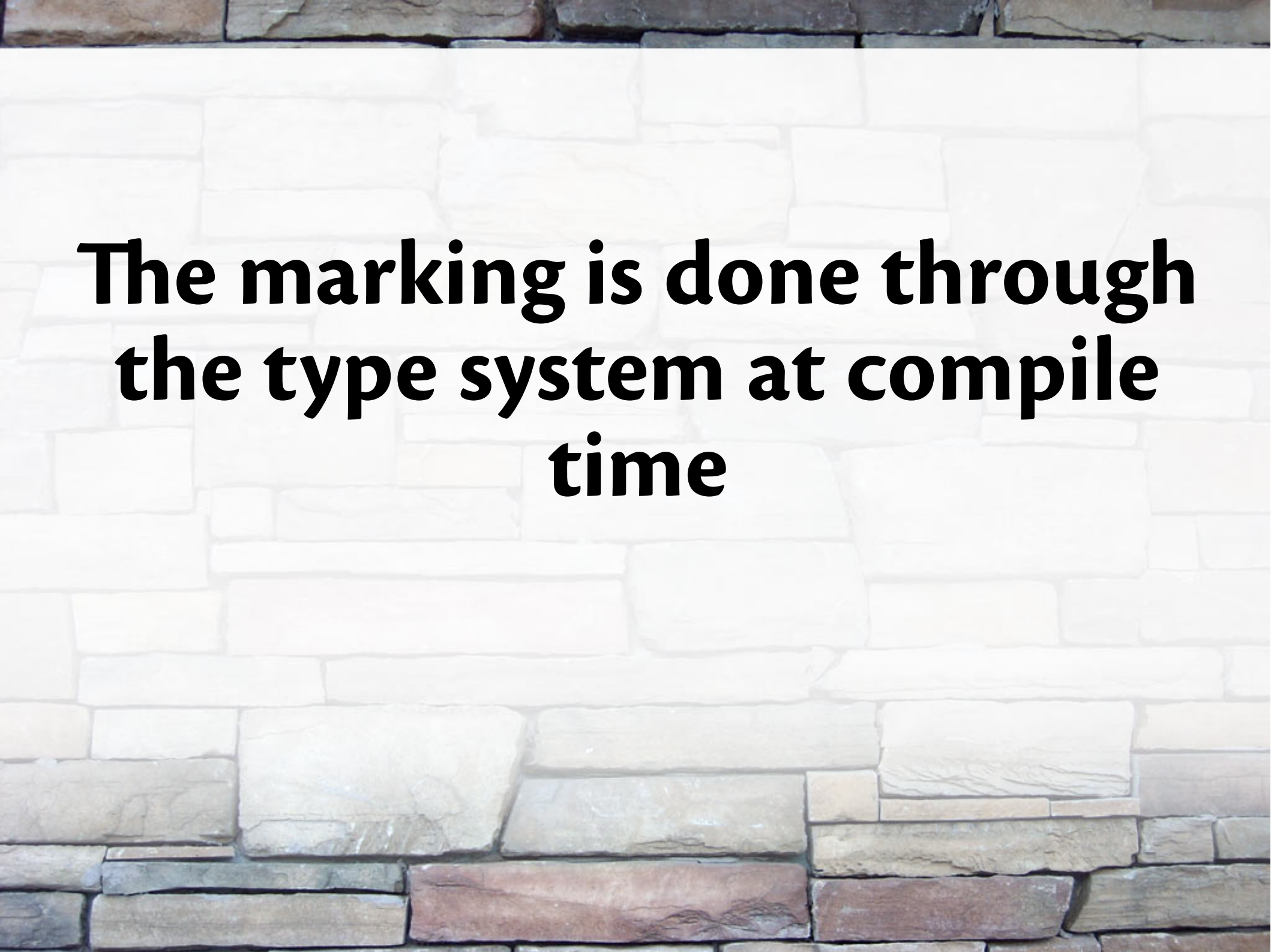
**Strong isolation
means freedom from
interference by other
threads**



**Recall that Haskell is a
strictly-typed, lazy, pure
functional language**



**Pure means that functions
with side-effects must be
marked as such**



**The marking is done through
the type system at compile
time**

STM
is just another kind of
I/O
(with a different marker:
"STM a" instead of "IO a")

The background of the slide is a close-up photograph of a stone wall. The wall is composed of irregularly shaped stones in various shades of beige, tan, and light brown. The stones are laid in a pattern that resembles a running bond or similar masonry style. The lighting is even, highlighting the textures and edges of the stones.

**Transactional memory needs
to be declared explicitly as
TVar**



**The STM library provides an
STM-to-IO converter called
"atomically"**

Transactional memory can only be accessed through dedicated functions like "modifyTVar", "readTVar", "writeTVar" which can only be called inside STM blocks

A background image of a stone wall. The wall is composed of irregularly shaped stones in various shades of beige, tan, and light brown. The stones are arranged in a roughly horizontal pattern, with some larger flat stones and some smaller, more angular ones. The lighting is even, highlighting the textures of the stone surfaces.

Implementation Overview Of GHC's STM

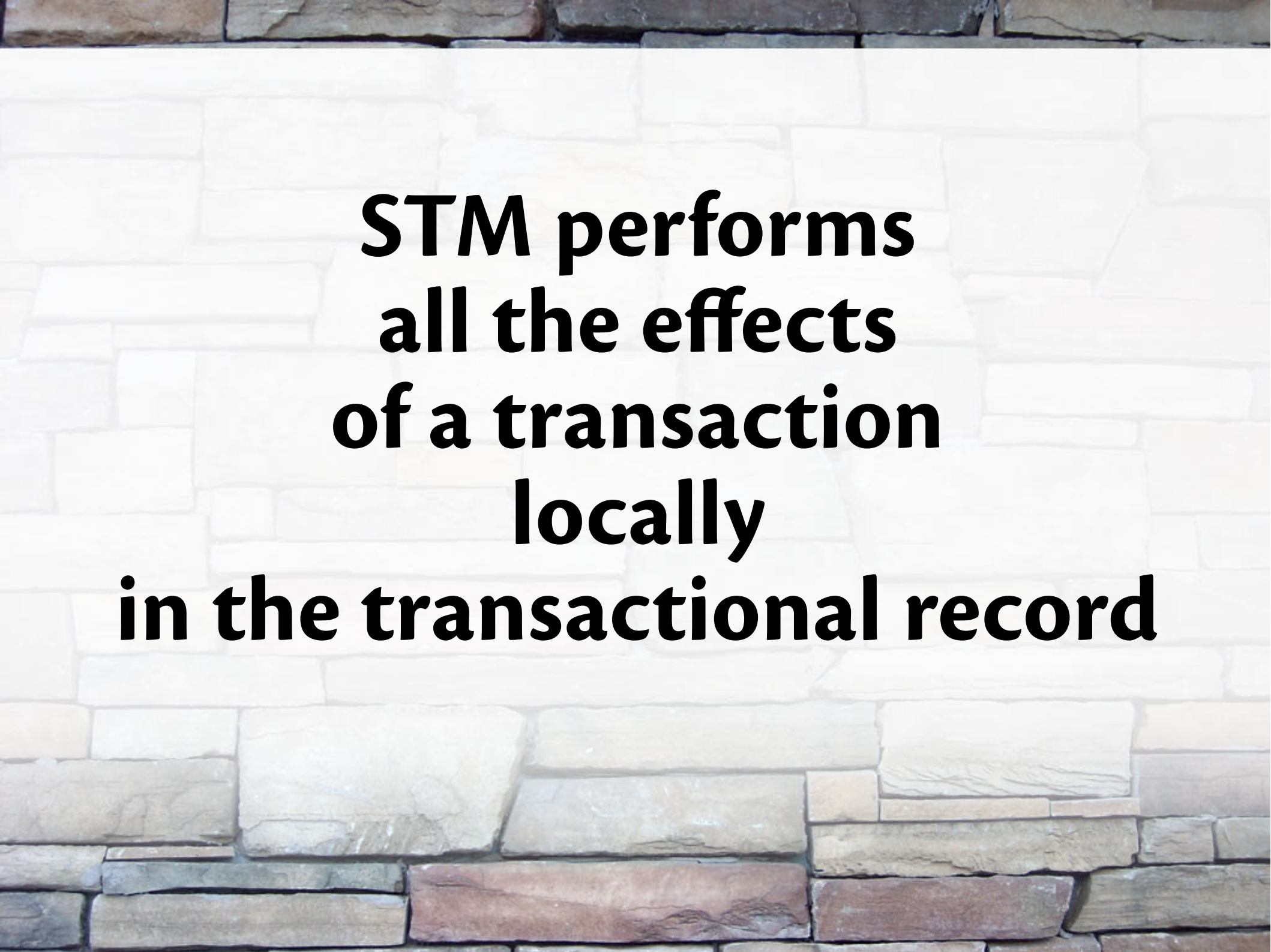
Definition

A transaction memory is a set of tuples in the shape of (Identity, Version, Value)

The version number represents the number of times the value has changed.

The Transactional Record

**Every STM transaction keeps
a record of state changes
(similar to the tx log in the DB world)**

The background of the slide is a close-up photograph of a stone wall. The stones are of various sizes and shapes, with colors ranging from light beige to dark brown. The texture is rough and uneven, with visible mortar joints.

**STM performs
all the effects
of a transaction
locally
in the transactional record**

**Once the transaction
has finished its work locally,
a version-based
consistency check
determines
if the values read
for the entire access set
are consistent**

**This version-based
consistency check
also obtains locks
for the write set
and with those locks
STM updates
the main memory
and then releases the locks**

**Rolling back the effects
of a transaction
means forgetting
the current
transactional record
and starting again**

**Reading: When a readTVar
is attempted
STM first searches
the tr. record
for an existing entry**

**Reading: If the entry is found,
STM will use
that local view
of the TVar**

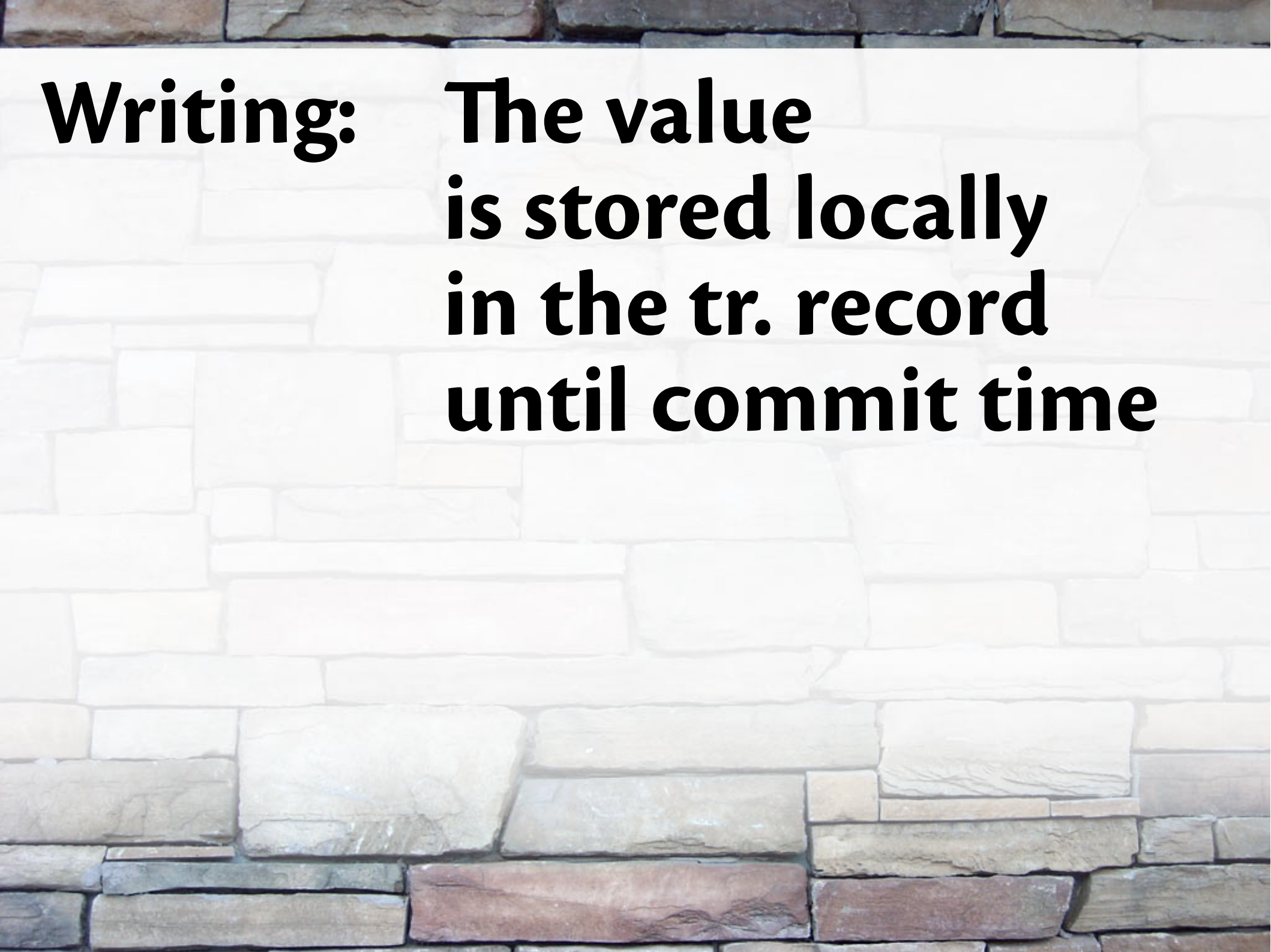
**Reading: On the first
readTVar,
a new entry
is allocated
and the TVar value
is read
and stored locally**

**Reading: The original Tvar
does not need
to be accessed again
for its value
until
validation time**

**Writing: Writing to a Tvar
requires
that the variable
first be
in the tr. record**

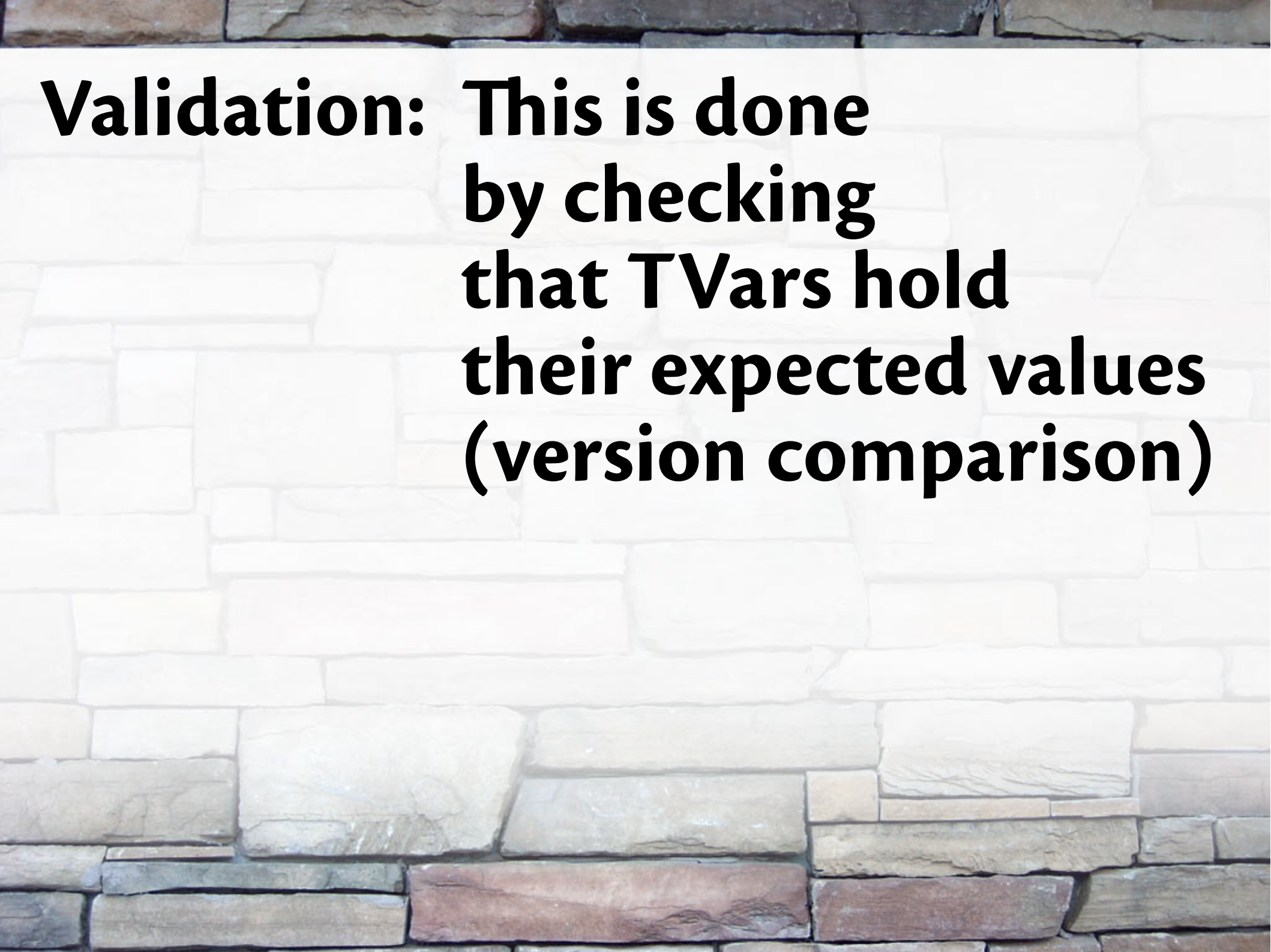
Writing: If it is not currently
in the tr. record,
a readTVar
is performed
and the value
is stored
in a new entry

**Writing: The version
in this entry
will be used
at validation time
to ensure
that no updates
were made
concurrently
to this TVar**

The background of the slide is a close-up photograph of a stone wall. The stones are of various sizes and shapes, with colors ranging from light beige to dark brown and reddish tones. The texture is rough and natural.

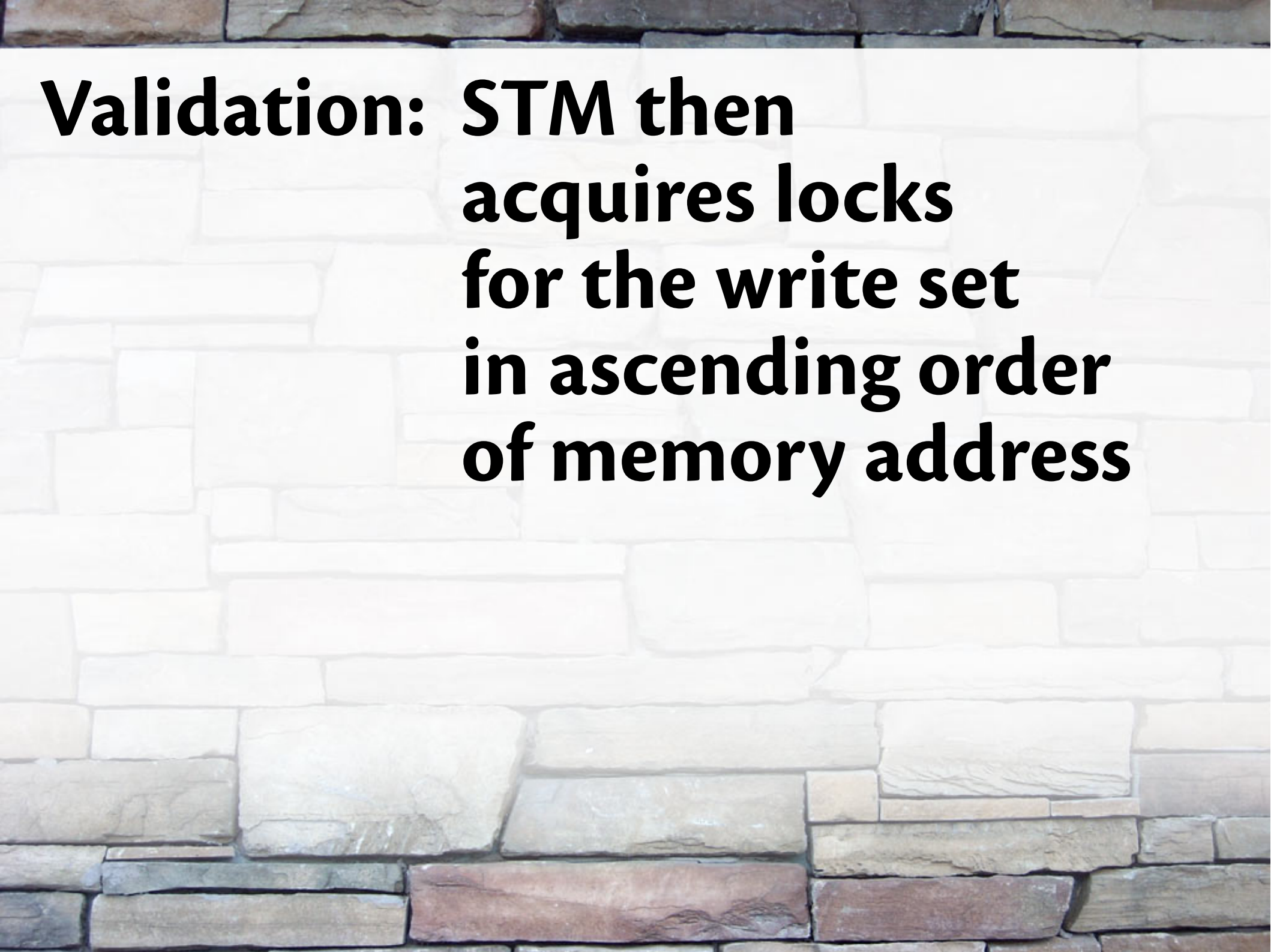
**Writing: The value
is stored locally
in the tr. record
until commit time**

**Validation: Before a transaction
can make
its effects visible
to other threads
it must check
that it has seen
a consistent view
of memory
while it was executing**

The background of the slide is a close-up photograph of a stone wall. The stones are of various sizes and shapes, with colors ranging from light beige to dark brown and reddish tones. The texture is rough and natural.

**Validation: This is done
by checking
that TVars hold
their expected values
(version comparison)**

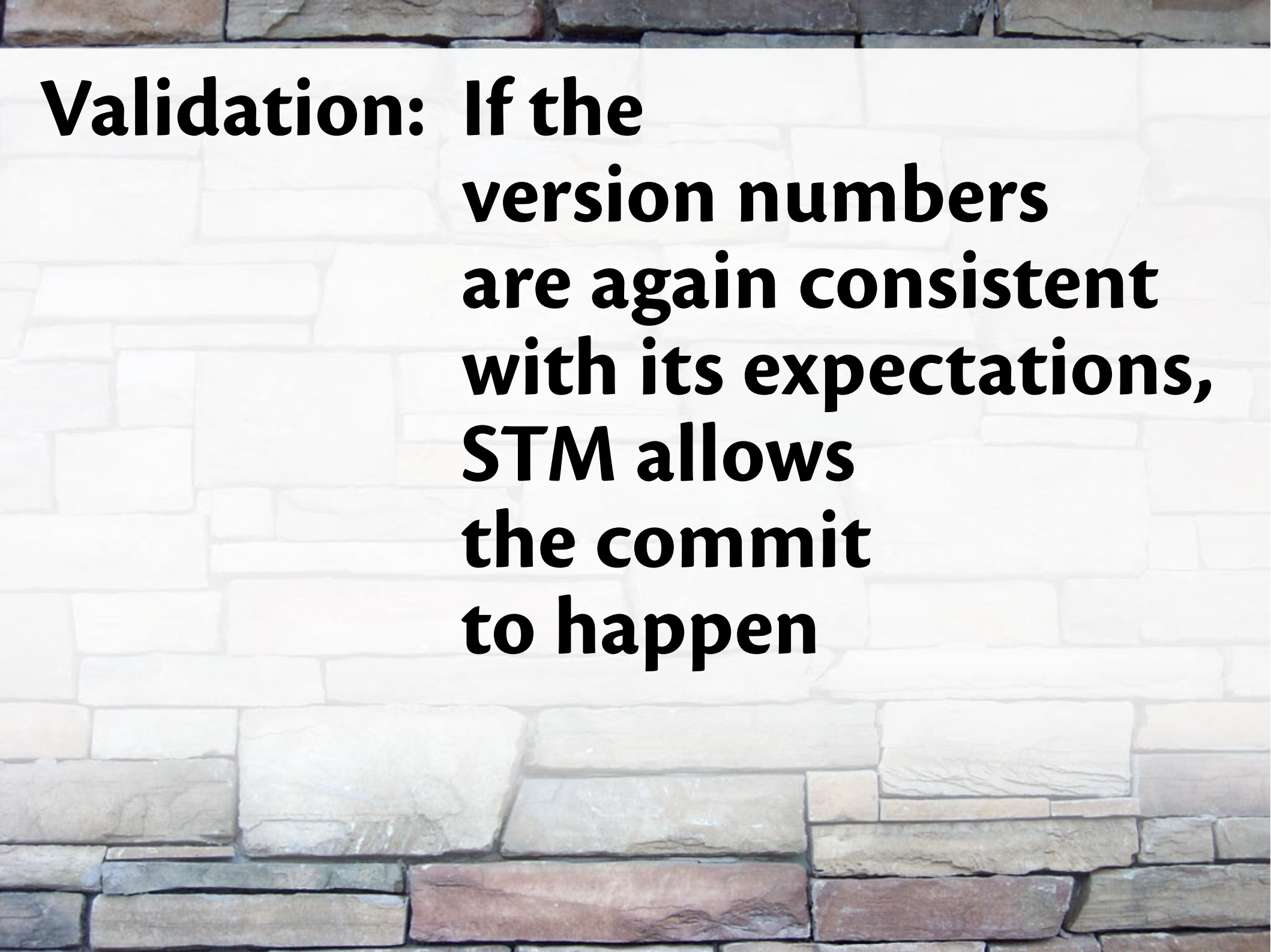
**Validation: During validation,
STM fetches
the version numbers
for all TVars
and checks
that they are
consistent
with its expectations**

The background of the slide is a close-up photograph of a stone wall. The stones are of various sizes and shapes, with colors ranging from light beige to dark brown. The wall is composed of several layers of stones, with some larger, flat stones in the middle and smaller, more irregular stones at the top and bottom.

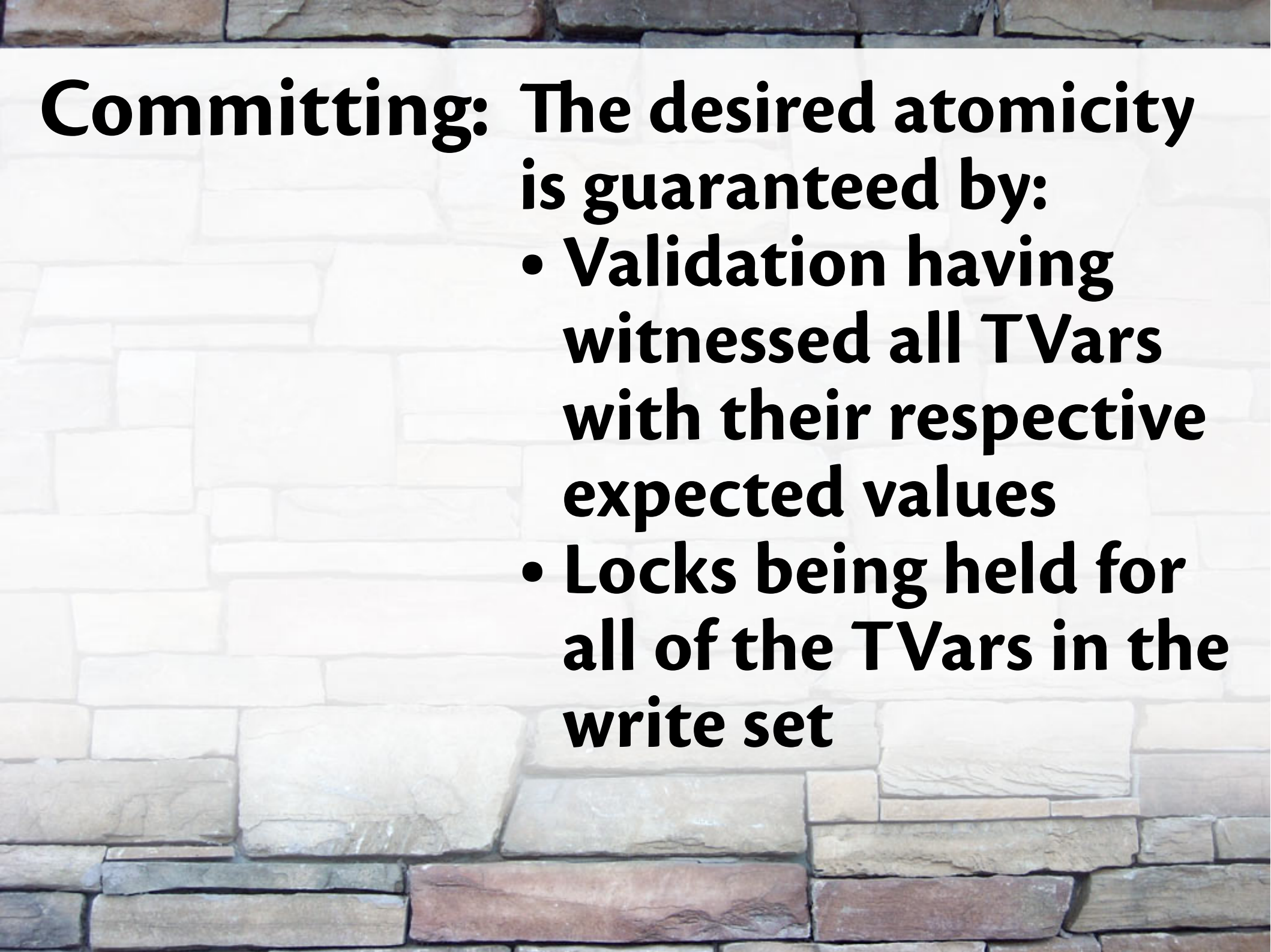
**Validation: STM then
acquires locks
for the write set
in ascending order
of memory address**

A background image of a stone wall with light-colored rectangular stones in the center and darker, more irregular stones at the top and bottom.

**Validation: STM then
reads and checks
all version numbers
again**

The background of the slide is a close-up photograph of a stone wall. The stones are of various sizes and shapes, with colors ranging from light beige to dark brown and reddish tones. The texture is rough and natural.

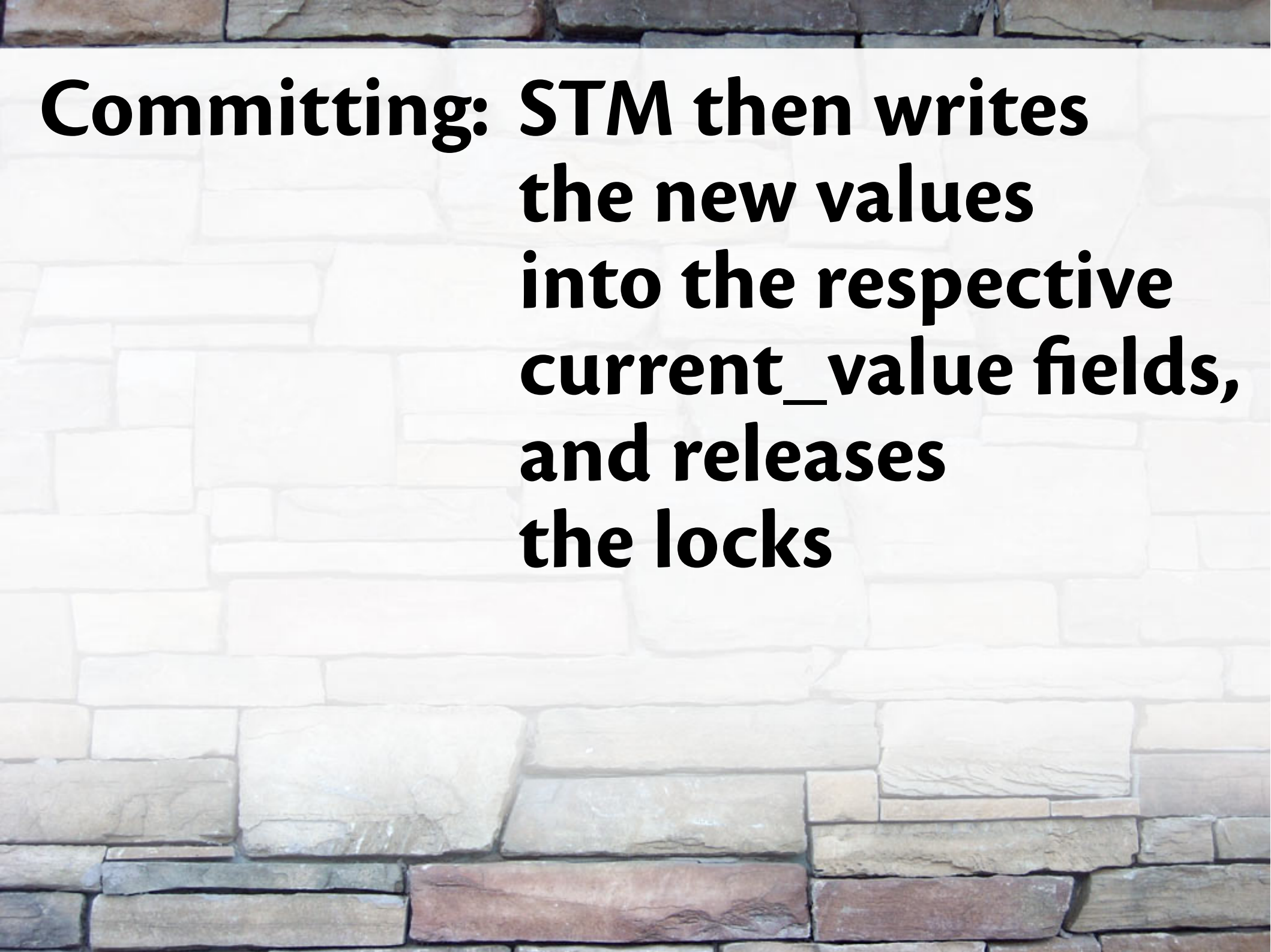
**Validation: If the
version numbers
are again consistent
with its expectations,
STM allows
the commit
to happen**

The background of the slide is a close-up photograph of a stone wall. The stones are of various sizes and shades of brown, tan, and grey, arranged in a traditional pattern. The lighting is even, highlighting the textures of the stones.

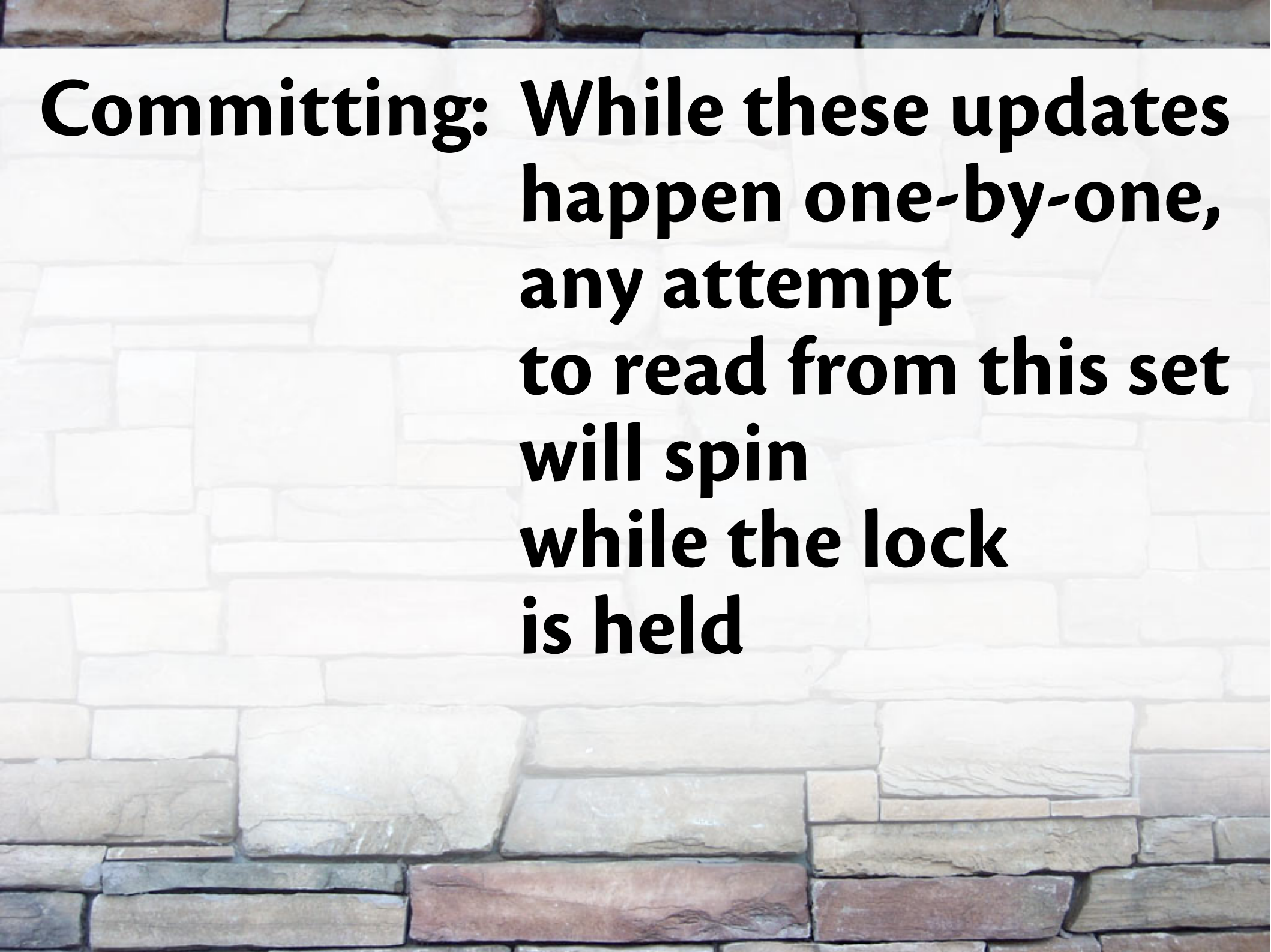
Committing: The desired atomicity is guaranteed by:

- **Validation having witnessed all TVars with their respective expected values**
- **Locks being held for all of the TVars in the write set**

**Committing: STM proceeds
to increment
each locked TVar's
num_updates
(a.k.a. version) field**



**Committing: STM then writes
the new values
into the respective
current_value fields,
and releases
the locks**



Committing: While these updates happen one-by-one, any attempt to read from this set will spin while the lock is held



**Another useful STM
abstraction is the TChan,
an unbounded FIFO channel**

**Once some messages are
transferred into a TChan,
they are ready to be
consumed by other threads
(broadcasting is possible too)**

The background of the slide is a close-up photograph of a stone wall. The stones are of various sizes and shapes, with colors ranging from light beige to dark brown. The texture is rough and uneven, with visible mortar lines.

TChans are useful when threads need to send signals to each other, as opposed to just accessing shared state

A background image of a stone wall with light-colored rectangular stones in the center and darker, irregular stones at the top and bottom.

Compile your STM code with:

```
ghc -threaded program.hs
```

When running the program:

```
./program +RTS -N
```


A background image of a stone wall with light-colored, irregularly shaped stones in the center and darker, reddish-brown stones at the top and bottom.

Follow me on GitHub

github.com/dserban