From Locks to STM in N simple steps

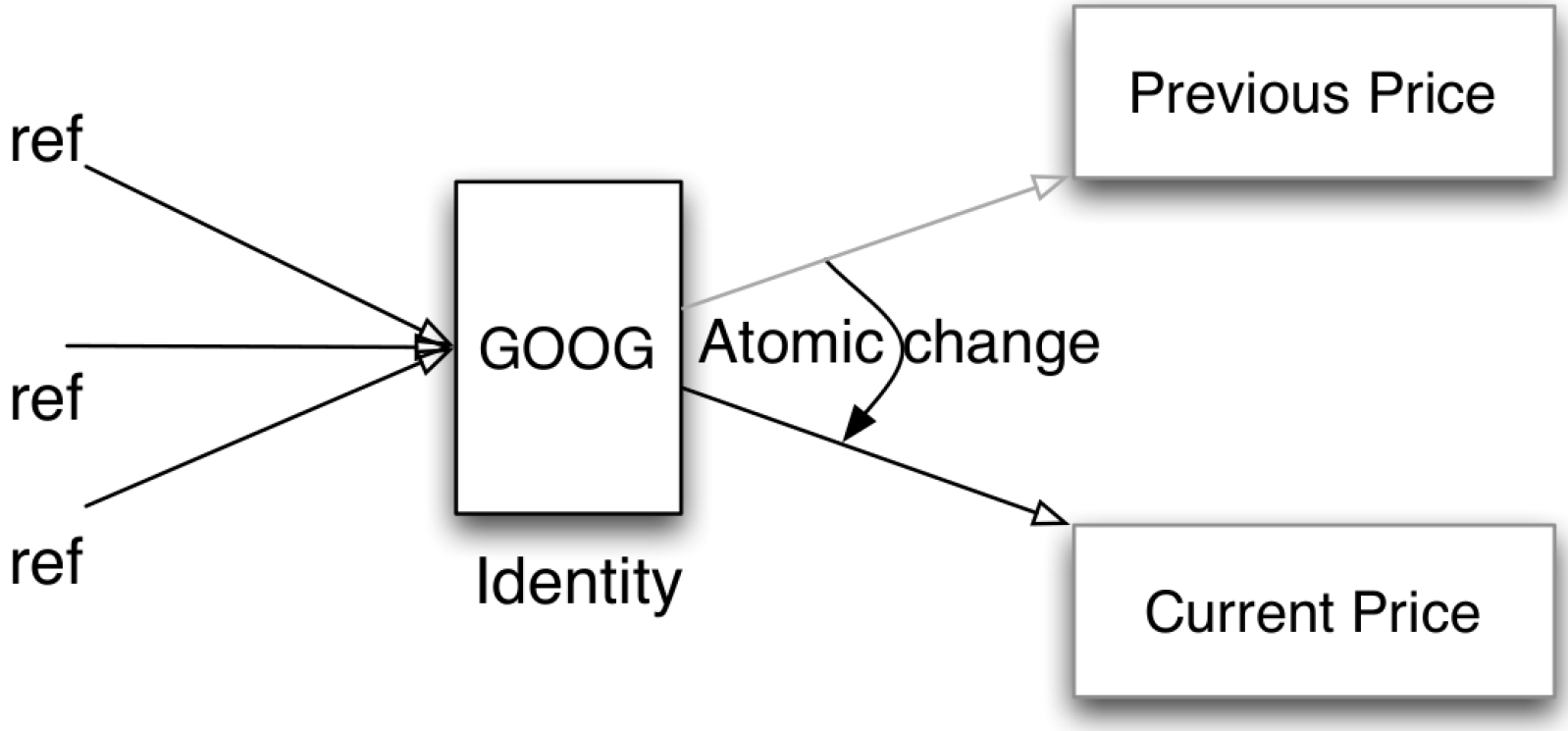
What is STM?

What is STM?

- technically, "atomic blocks"

What is STM?

- technically, "atomic blocks"
- only possible in Haskell (and of course Clojure...)
    - value, identity, state
    - idempotence of atomic blocks


ref
ref
ref
GOOG
Identity
Atomic change
Previous Price
Current Price

What is STM?

- technically, "atomic blocks"
- only possible in Haskell (and of course Clojure...)
    - value, identity, state
    - idempotence of atomic blocks
- AI (no CD)

What is STM?

- technically, "atomic blocks"
- only possible in Haskell (and of course Clojure...)
    - value, identity, state
    - idempotence of atomic blocks
- AI (no CD)
    - Atomicity

What is STM?

- technically, "atomic blocks"
- only possible in Haskell (and of course Clojure...)
    - value, identity, state
    - idempotence of atomic blocks
- AI (no CD)
    - Atomicity
        - all operations are visible, or none of them are

What is STM?

- technically, "atomic blocks"
- only possible in Haskell (and of course Clojure...)
    - value, identity, state
    - idempotence of atomic blocks
- AI (no CD)
    - Atomicity
        - all operations are visible, or none of them are
    - Isolation

What is STM?

- technically, "atomic blocks"
- only possible in Haskell (and of course Clojure...)
    - value, identity, state
    - idempotence of atomic blocks
- AI (no CD)
    - Atomicity
        - all operations visible, or none of them are
    - Isolation
        - snapshots

How does it work?

- optimistic execution

How does it work?

- optimistic execution
- validation

How does it work?

- optimistic execution
- validation
- commit

In Haskell

```haskell
data STM a -- abstract
instance Monad STM -- among other things

atomically :: STM a -> IO a

data TVar a -- abstract
newTVar    :: a -> STM (TVar a)
readTVar   :: TVar a -> STM a
writeTVar  :: TVar a -> a -> STM ()

retry      :: STM a
orElse     :: STM a -> STM a -> STM a

throwSTM   :: Exception e => e -> STM a
catchSTM   :: Exception e => STM a -> (e -> STM a) -> STM a
```

Some notes

- STM
  - note: no MonadIO instance for STM
  - composable
  - executable in IO (`atomically`)

References

- Simon Marlow, Parallel and Concurrent Programming in Haskell
- Simon Peyton Jones, Beautiful Concurrency
- Suhramaniam V, Programming Concurrecy on the JVM
- Andrew S. Tanenbaum, Modern Operating Systems