

# Atomic compare-and-swaps

(optimistic concurrency)

```
last_changed = obj.modified
```

```
...
```

```
SomeModel.objects.filter(id=obj.id, modified=last_changed).update(val=new_val)
```

Only updates if the db row is unchanged by other threads.

- > any modified obj in db will differ from our stale in-memory obj ts
- > filter() wont match any rows, update() fails
- > overwriting newer row in db with stale data is prevented

This is very hard to get right, locking is better for 90% of use cases!

# Hybrid Solution

(optimistic concurrency + pessimistic or Multiversion Concurrency Control)

```
last_changed = obj.modified
```

```
... read phase
```

```
SomeModel.objects.select_for_update().filter(id=obj.id, modified=last_changed)
```

```
... write phase
```

## Best of both worlds

- > locking is limited to write-phase only
- > no need for complex multi-model compare-and-swaps

MVCC is used internally by PostgreSQL