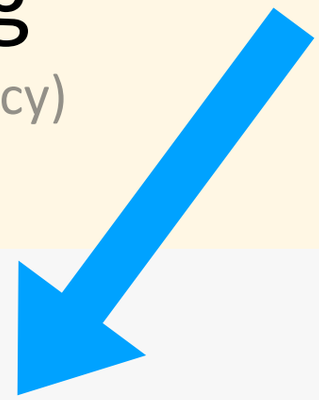
Row Locking

(pessimistic concurrency)



```
with transaction.atomic():
    to_update = SomeModel.objects.select_for_update().filter(id=thing.id)
    ...
    to_update.update(val=new)
```

.select_for_update() allows you to lock rows

Locking prevents other threads from changing the row until the end of the current transaction, when the lock is released.

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!