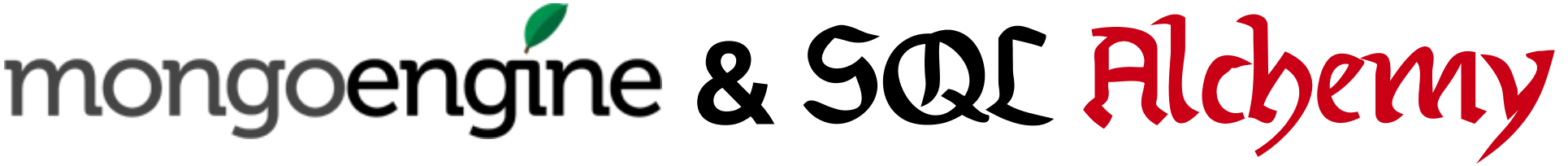
Database madness with



Jaime Buelta

jaime.buelta@gmail.com

wrongsideofmemphis.worpress.com

A little about the project

- Online football management game
- Each user could get a club and control it
- We will import data about leagues, teams, players, etc from a partner
- As we are very optimistic, we are thinking on lots of players!
  - Performance + non-relational data --> NoSQL

Pylons application

- Pylons is a “framework of frameworks”, as allow to custom made your own framework
- Very standard MVC structure
- RESTful URLs
- jinja2 for templates
- jQuery for the front end (JavaScript, AJAX)

Databases used

- MongoDB
- MS SQL Server
  - Legacy data
- MySQL
  - Our static data

MS SQL Server

- Legacy system
  - Clubs
  - Players
  - Leagues

WE CANNOT AVOID USING IT!

MySQL

- Static data
  - Clubs
  - Players
  - Leagues
  - Match Engine definitions
  - Other static parameters...

MongoDB

- User data
  - User league
    - Each user has his own league
  - User team and players
  - User data
    - Balance
    - Mailbox
    - Purchased items

SQL Alchemy

- ORM to work with SQL databases
- Describe MySQL DB as classes (declarative)
- Highly related to SQL
- Can describe complex and custom queries and relationships
- Low level
- Conection with legacy system using Soup to autodiscover schema

Definition of our own MySQL DB

```
Base = declarative_base()
class BaseHelper(object):
    @classmethod
    def all(cls):
        '''Return a Query of all'''
        return meta.Session.query(cls)

    @classmethod
    def one_by(cls, **kwargs):
        return cls.all().filter_by(**kwargs).one()

    def save(self):
        ''' Save method, like in Django ORM'''
        meta.Session.add(self)
        meta.Session.commit()
```

```
class Role(Base, BaseHelper):
    ''' Roles a player can take '''
    __tablename__ = 'roles'
    metadata = meta.metadata

    name = sa.Column(sa.types.String(5),
                     primary_key=True, nullable=False)
    full_name = sa.Column(sa.types.String(25))
    description = sa.Column(sa.types.String(50))

    def __repr__(self):
        return '<{0}>'.format(self.name)
```

Connection to MS SQL Server

- **FreeTDS** to be able to connect from Linux
- **pyodbc** to access the conection on the python code
- Quite painful to make it to work

BE CAREFUL WITH THE ENCODING!

Connect to database

```
import pyodbc
import sqlalchemy as sa
from sqlalchemy.ext.sqlsoup import SqlSoup
def connect():
   return pyodbc.connect('DSN=db_dsn;UID=user;PWD=password')
# force to Unicode. The driver should be configured to UTF-8 mode
engine = sa.create_engine('mssql://', creator=connect,
                                        convert_unicode=True)
connection = engine.connect()
# use Soup
meta = MetaData()
meta.bind = connection
db = SqlSoup(meta)
```

You still have to define the relationships!

```
# tblPerson
db.tblPerson.relate('skin_color', db.tblColour,
                    primaryjoin=db.tblColour.ColourID == db.tblPerson.SkinColourID)
db.tblPerson.relate('hair_color', db.tblColour,
                    primaryjoin=db.tblColour.ColourID == db.tblPerson.HairColourID)
db.tblPerson.relate('nationality_areas', db.tblArea,
                    primaryjoin=db.tblPerson.PersonID == db.tblPersonCountry.PersonID,
                    secondary=db.tblPersonCountry._table,
                    secondaryjoin=db.tblPersonCountry.AreaID == db.tblArea.AreaID)
...
country_codes = [ area.CountryCode for area in player.Person.nationality_areas]
```

Cache

- SQL Alchemy does not have an easy integrated cache system.
  - None I know, at least!
  - There is Beaker, but appears to be quite manual
- To cache static data (no changes), some information is cached the old way.
  - Read the database at the start-up and keep the results on global memory


And now for something completely different

mongoengine

- Document Object Mapper for MongoDB and Python
- Built on top of pymongo
- Very similar to Django ORM

Connection to DB

```
import mongoengine
mongoengine.connect('my_database')
```

BE CAREFUL WITH DB NAMES!
If the DB doesn't exist, it will be created!

Definition of a Collection

```
import mongoengine
class Team(mongoengine.Document):
    name = mongoengine.StringField(required=True)
    players = mongoengine.ListField(Player)
    meta = {
                'indexes':['name'],
               }
team = Team(name='new')
team.save()
```

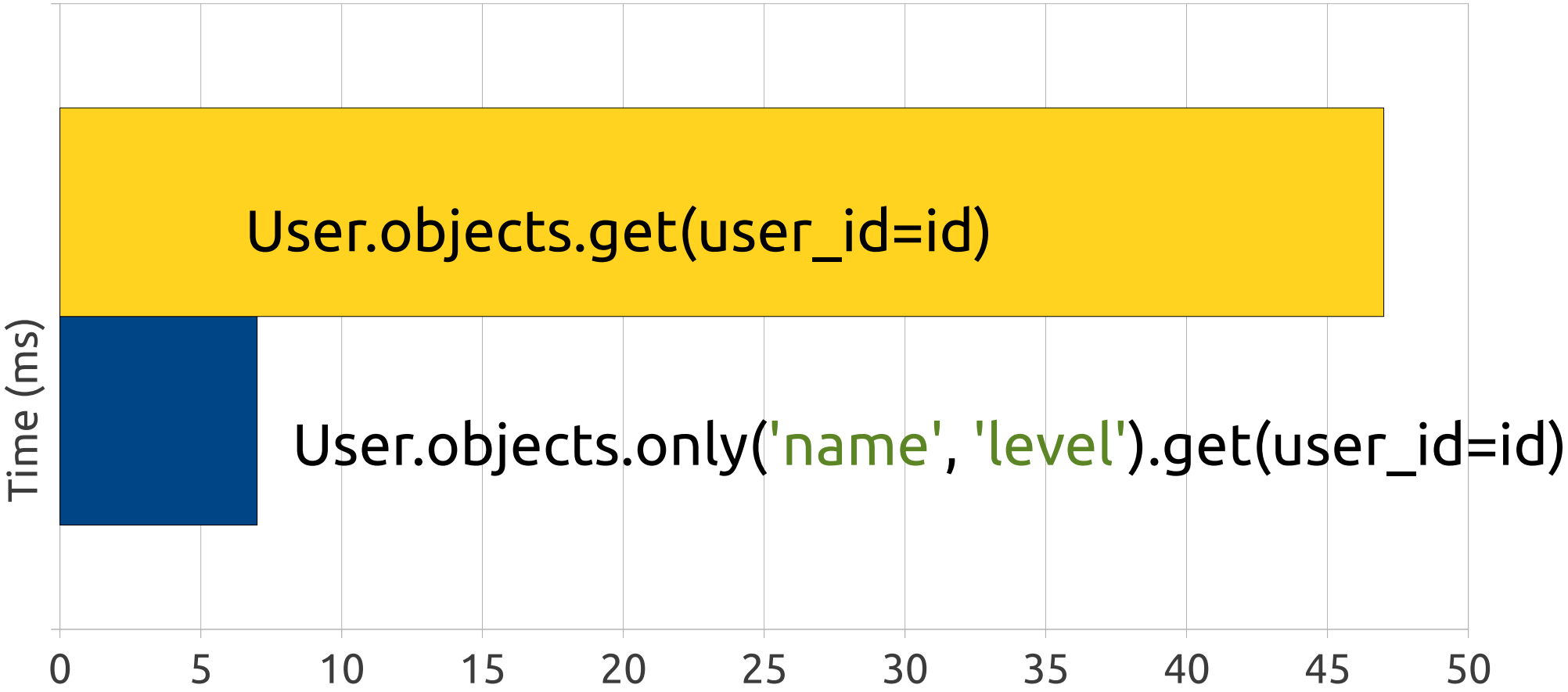
Querying and retrieving from DB

```
Team.objects  # All the objects of the collection
Team.objects.get(name='my_team')
Team.objects.filter(name__startswith='my')
Team.objects.filter(embedded_player__name__startswith='Eric')
```

NO JOINS!
Search on referenced objects
will return an empty list!

```
Team.objects.filter(referenced_player__name__startswith) = []
```

Get partial objects



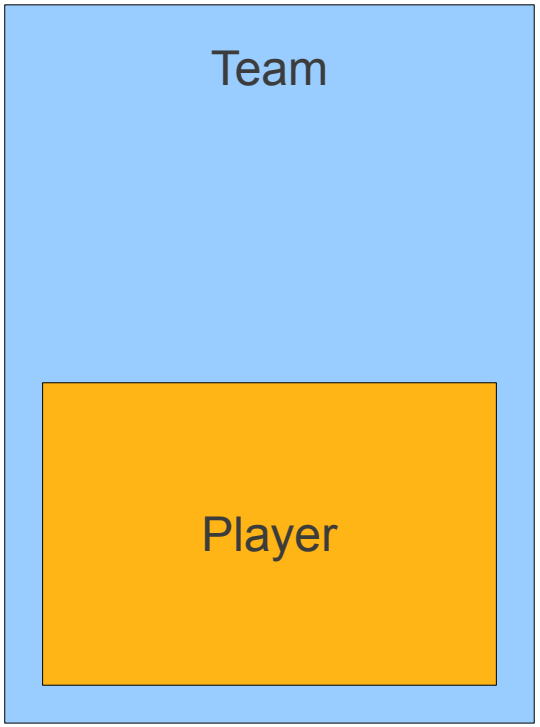

```
# Get all fields
user.reload()
```

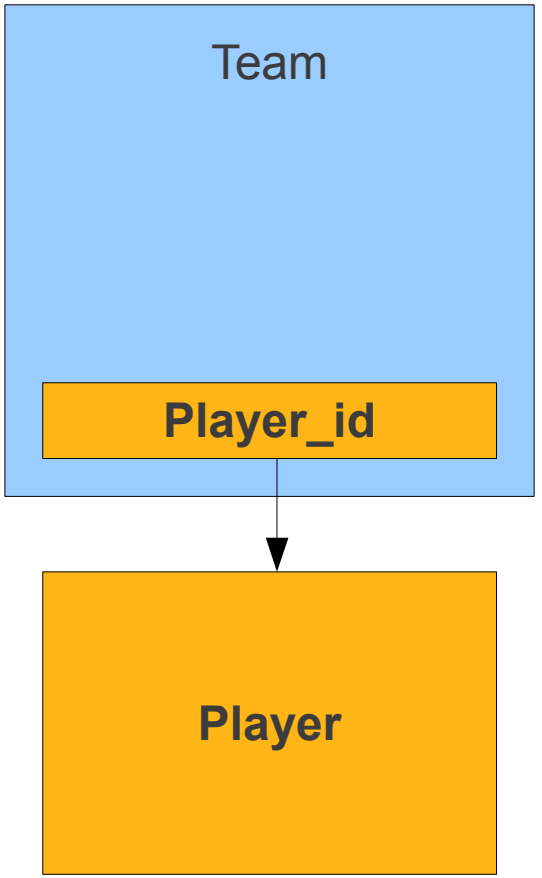
Careful with default attributes!

```
>>> import mongoengine
>>> class Items(mongoengine.Document):
...     items =mongoengine.ListField(mongoengine.StringField(),default=['starting'])
>>> i = Items()
>>> i.items.append('new_one')
>>> d = Items()
>>> print d.items
['starting','new_one']
```

```
# Use instead
items = mongoengine.ListField(mongoengine.StringField(),
                              default=lambda: ['starting'])
```

EmbededDocument vs Document





Embed or reference?

- Our DB is mostly a big User object (not much non-static shared data between users)
- On MongoDB your object can be as big as 4MiB*
- Performance compromise:
  - Dereference objects has an impact (Everything embedded)
  - But retrieve big objects with non-needed data takes time (Reference objects not used everytime)
- You can retrieve only the needed parts on each case, but that adds complexity to the code.

* Our user is about 65Kb

Our case

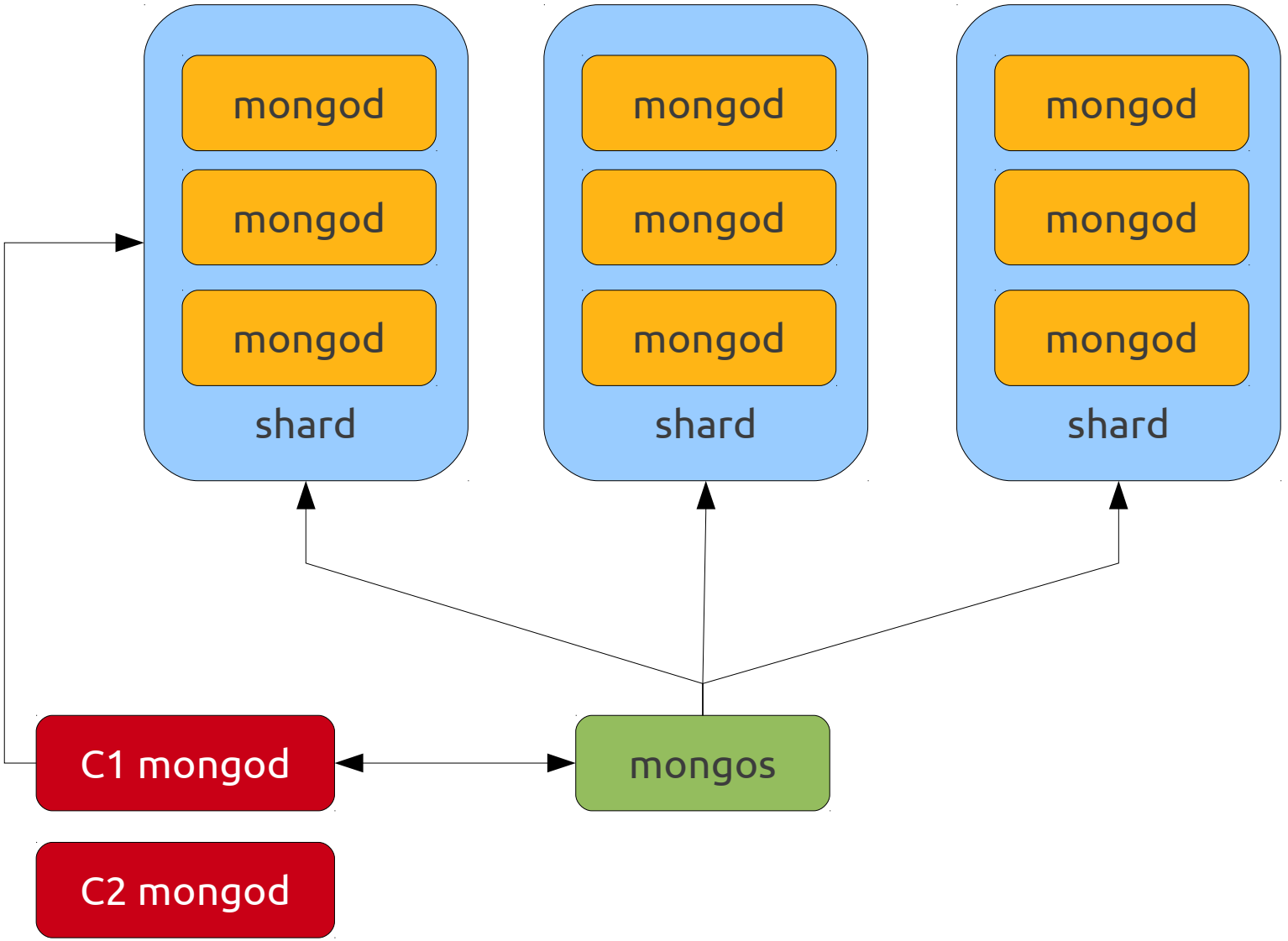
- Practically all is embedded in a big User object.
- Except for the CPU controlled-teams in the same league of the user.
- We'll keep an eye on real data when the game is launched.

The embedded objects have to be
saved from the parent Document.
Sometimes is tricky.
Careful with modifying the data and not saving it

Other details

- As each player has lots of stats, instead of storing each one with a name (DictField), we order them and assign them slots on a list
- Pymongo 1.9 broke compatibility with mongoengine 0.3 Current branch 0.4 working on that, but not official release yet

sharding


mongod
mongod
mongod
shard
mongod
mongod
mongod
shard
mongod
mongod
mongod
shard
C1 mongod
C2 mongod
mongos

Thanks for your attention!

Questions?