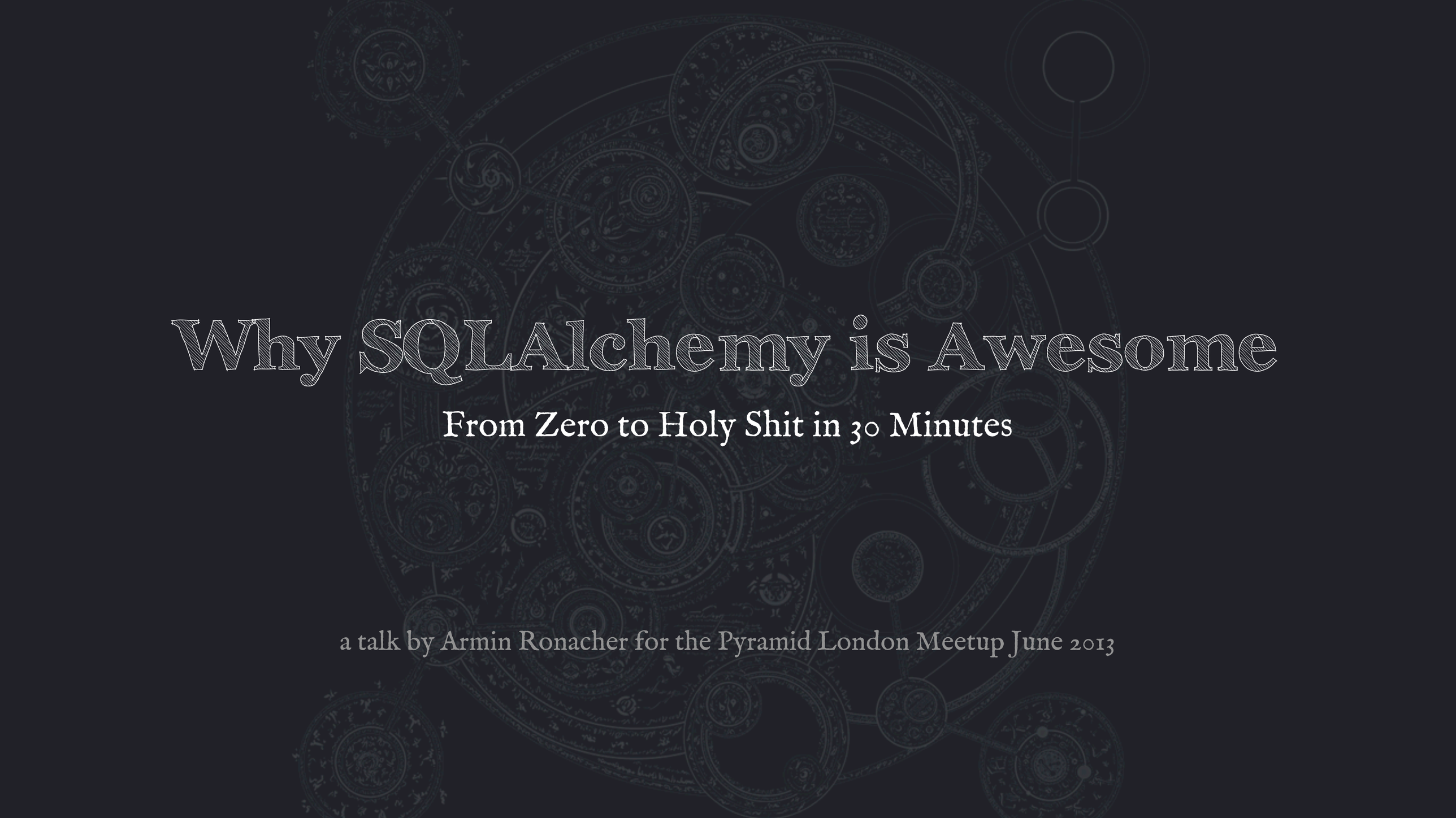
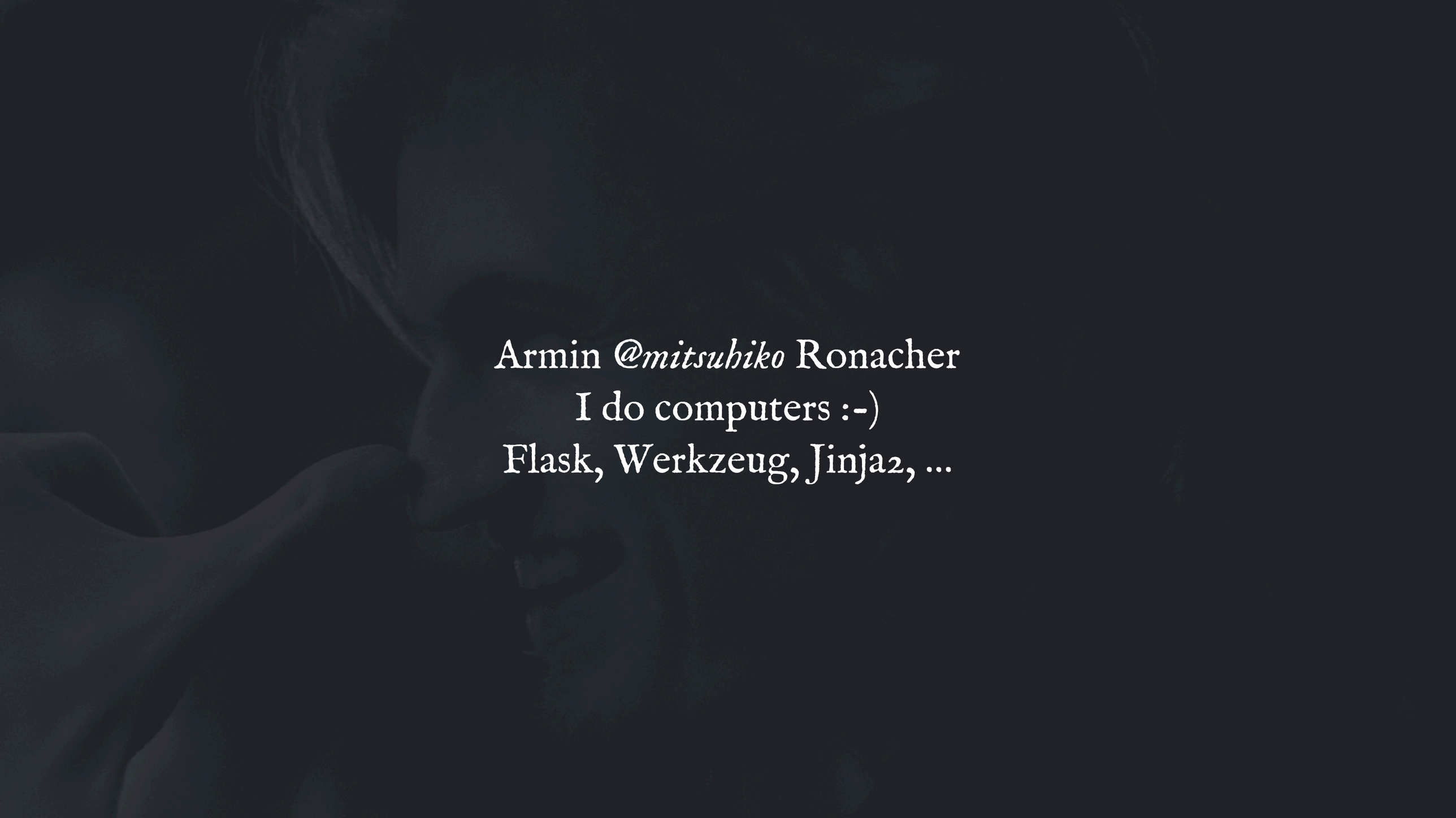


Why SQLAlchemy is Awesome

From Zero to Holy Shit in 30 Minutes

a talk by Armin Ronacher for the Pyramid London Meetup June 2013


Armin @mitsuhiko Ronacher
I do computers :-)
Flask, Werkzeug, Jinja2, ...

SQL Databases are Awesome

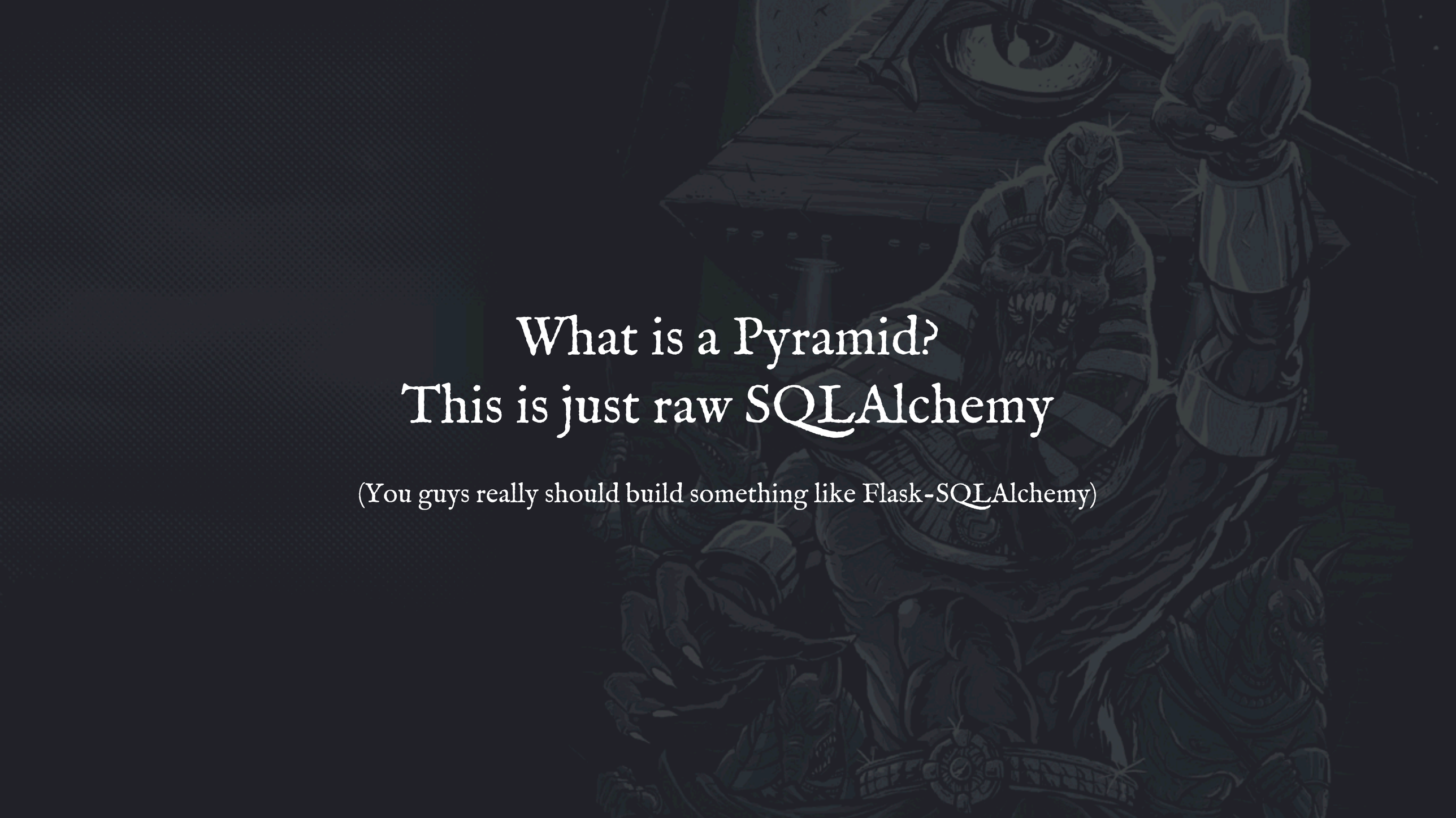
SQLAlchemy is not *just* an ORM

- Connection Management
- Driver Abstraction
- SQL Expression Language
- ORM + Unit-of-Work

Way too much for one talk

might be hostile to start with

but it's worth it.



What is a Pyramid?
This is just raw SQLAlchemy

(You guys really should build something like Flask-SQLAlchemy)

N SQLAlchemy Components

Compiler

- SQL Compiler
- DDL Compiler

Engine

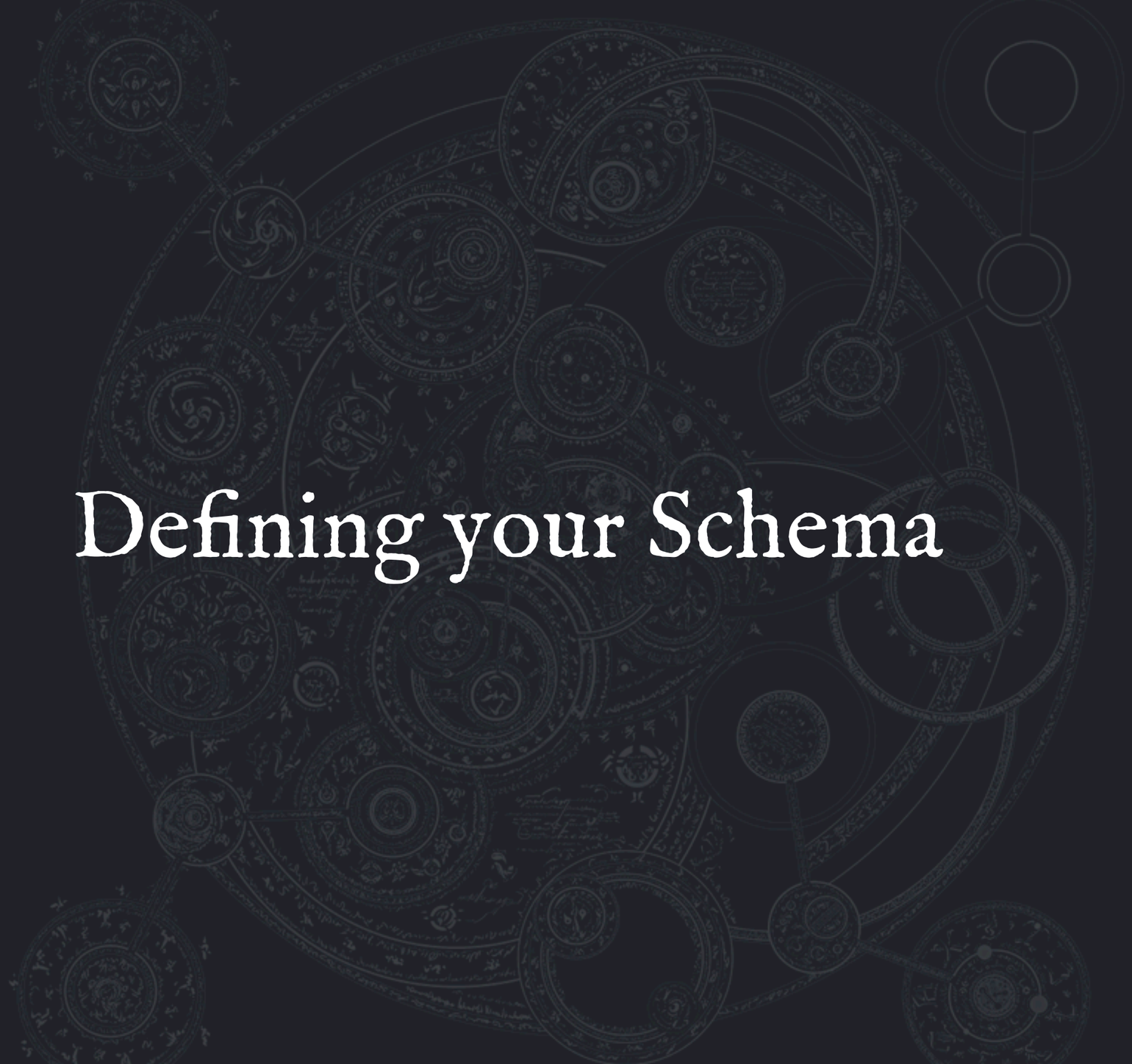
- DB Driver
- Connection Pool

Meta

- Table Definitions
- Mapper

Session

- Transaction State
- Connection
- Unit of Work
- Identity Map



I Defining your Schema

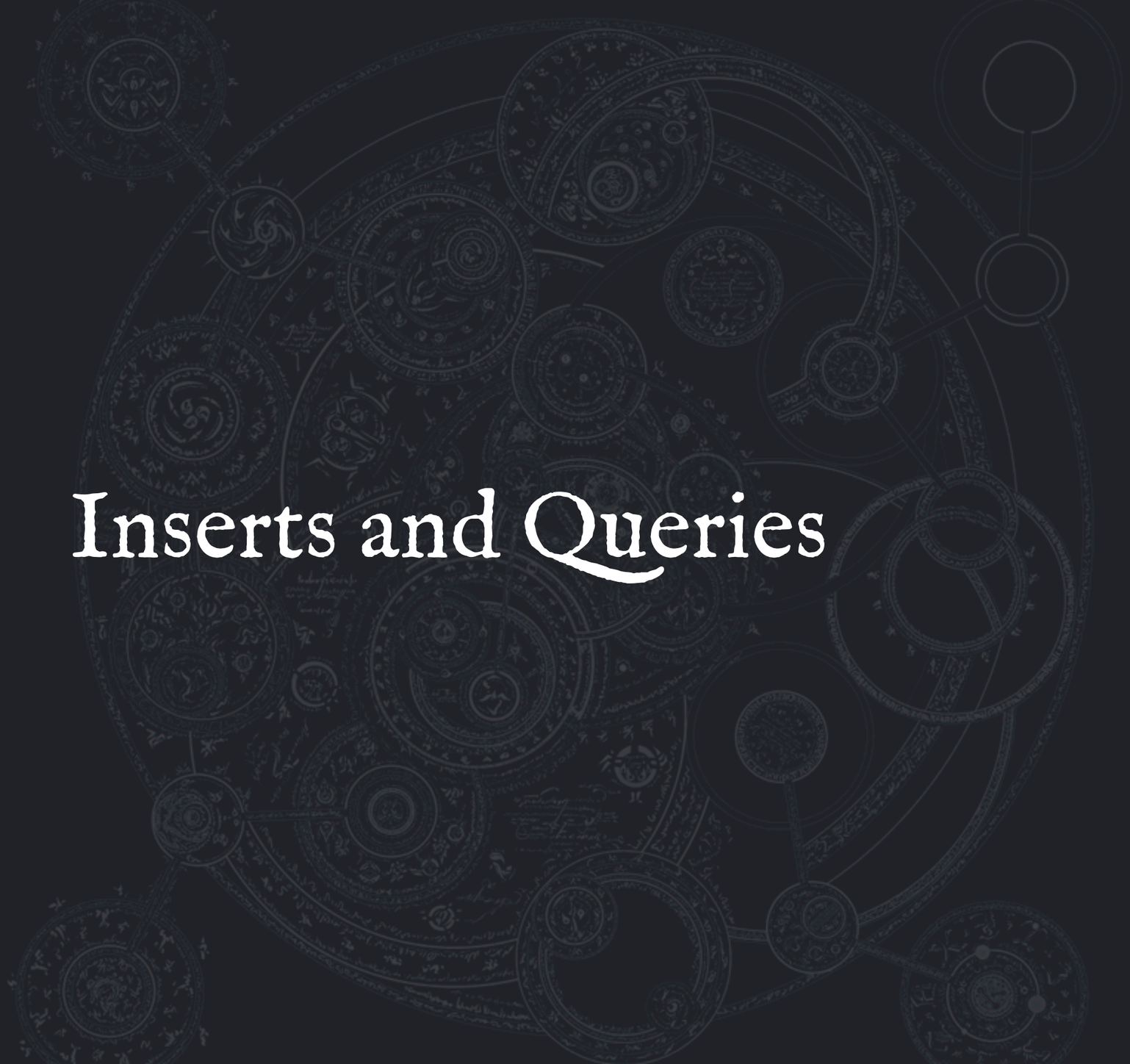
i. Reflection from Database Schema
ii. Table Definitions
iii. Class Definitions (Declarative Base)

```
import sqlalchemy as db

metadata = db.MetaData()

users = db.Table('users', metadata,
    db.Column('user_id', db.Integer(), primary_key=True),
    db.Column('username', db.String()),
    db.Column('email_address', db.String())
)
```


engine = db.create_engine('postgres:///yourdb')
metadata.create_all(engine)



II Inserts and Queries

```
>>> q = users.insert({'username': 'john',
...                   'email_address': 'john@example.com'})
...
>>> engine.execute(q)


>>> engine.execute(users.select()
...     .where(users.c.username == 'john')
... ).fetchone()
...
(1, u'john', u'john@example.com')
```

```
posts = db.Table('posts', metadata,
    db.Column('post_id', db.Integer(), primary_key=True),
    db.Column('author_id', db.Integer(),
              db.ForeignKey('users.user_id')),
    db.Column('title', db.String()),
    db.Column('body', db.String())
)
```

```
>>> engine.execute(posts.insert({
...         'author_id': 1,
...         'title': 'Hello World',
...         'body': 'Oh, so interesting'
... }))
```

```
>>> dict(engine.execute(posts.join(users).select()).fetchone())
{u'body': u'Oh, so interesting',
 u'username': u'john',
 u'user_id': 1,
 u'title': u'Hello World',
 u'post_id': 1,
 u'author_id': 1,
 u'email_address': u'john@example.com'}
```

III Sessions

```
>>> from sqlalchemy.orm import Session
>>> s = Session(engine)
>>> s.execute(users.select()).fetchone()
(1, u'john', u'john@example.com')
```

Sessions wrap Transactions

```
>>> s.execute('invalid sql')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
sqlalchemy.exc.ProgrammingError: (ProgrammingError)
    syntax error at or near "invalid"
LINE 1: invalid sql
        ^


>>> s.execute(users.select()).fetchone()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
sqlalchemy.exc.InternalError: (InternalError)
    current transaction is aborted, commands ignored
    until end of transaction block
```

IV The Mapper

Mapper: maps Tables to Classes

```
>>> from sqlalchemy.orm import mapper
>>>
>>> class User(object):
...  pass
...
>>> mapper(User, users)
<Mapper at 0x1066c6750; User>

>>> s.query(User).filter(User.username == 'john').first()
<__main__.User object at 0x106b19810>
>>> _.email_address
u'john@example.com'
```

V Declarative Base

Nicer API for Class + Table Creation

```
import sqlalchemy as db
from sqlalchemy.ext.declarative import declarative_base

metadata = db.MetaData()
Base = declarative_base(metadata=metadata)


class User(Base):
    __tablename__ = 'users'
    id = db.Column('user_id', db.Integer(), primary_key=True)
    username = db.Column(db.String())
    email_address = db.Column(db.String())
```

```
import sqlalchemy as db
from sqlalchemy.ext.declarative import declarative_base

metadata = db.MetaData()
Base = declarative_base(metadata=metadata)
engine = db.create_engine('postgresql:///yourdb')
```

```
class User(Base):
    __tablename__ = 'users'
    id = db.Column('user_id', db.Integer(), primary_key=True)
    username = db.Column(db.String())
    email_address = db.Column(db.String())
    posts = db.orm.relationship('Post', backref='author',
                                lazy='dynamic')


class Post(Base):
    __tablename__ = 'posts'
    id = db.Column('post_id', db.Integer(), primary_key=True)
    author_id = db.Column(db.Integer(),
                          db.ForeignKey('users.user_id'))
    title = db.Column(db.String())
    body = db.Column(db.String())
```

```
>>> s = db.orm.Session(engine)
>>> john = s.query(User).filter(
...     User.username == 'john').first()
...
>>> john.email_address
u'john@example.com'
>>> john.posts.count()
1L
>>> john.posts.first().author is john
True
```

notice how john is in memory only once!

```
>>> me = User()
>>> me.username = u'mitsuhiko'
>>> me.email_address = u'armin.ronacher@active-4.com'
>>> s.add(me)
>>> s.commit()

>>> me.email_address = u'armin.ronacher+spam@active-4.com'
>>> s.commit()
```

notice how it automatically knows what to update

VI Session Scoping

Sessions can be context bound

```
import sys
from sqlalchemy.orm import scoped_session, Session

session = scoped_session(lambda: Session(engine))


def handle_request(request):
    try:
        return actual_request_handling(request)
    finally:
        if sys.exc_info()[2] is None:
            session.commit()
        session.remove()
```

```
Base.query = session.query_property()

>>> session.query(User).first()
<__main__.User object at 0x106b19810>

>>> User.query.first()
<__main__.User object at 0x106b19810>
```

that's where context binding comes in useful

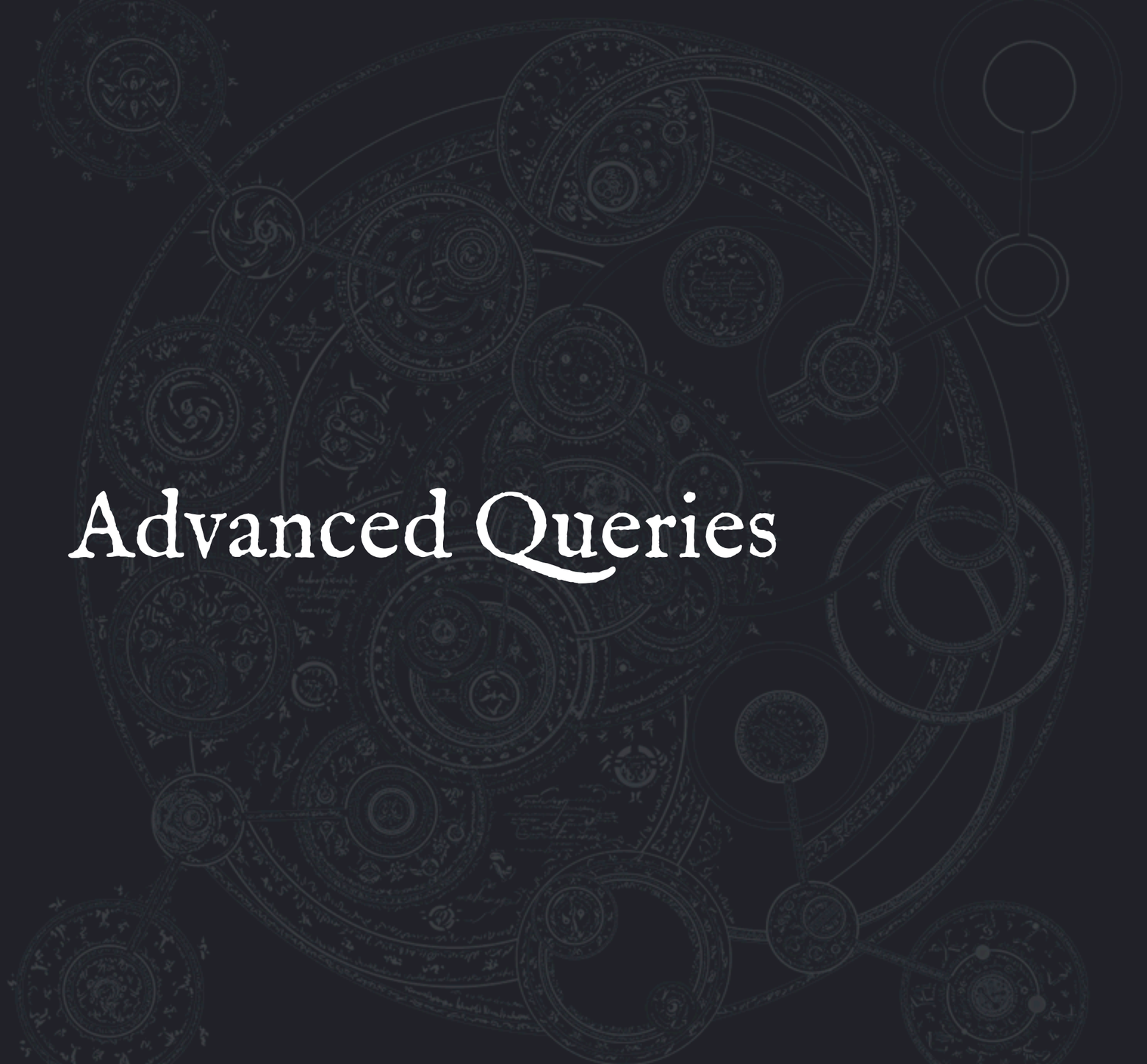
```
from sqlalchemy.orm import class_mapper, Query
from sqlalchemy.exc import UnmappedClassError

class _QueryProperty(object):

    def __get__(self, obj, type):
        try:
            mapper = class_mapper(type)
            if mapper:
                return type.query_class(mapper,
                                        session=session())
        except UnmappedClassError:
            raise AttributeError('Model not mapped')

Base = declarative_base()
Base.query = _QueryProperty()
Base.query_class = Query
```

personal favorite



VII Advanced Queries

```
entries_a_month = s.query(Entry).filter(
    (db.extract(Entry.pub_date, 'year') == 2013) &
    (db.extract(Entry.pub_date, 'month') == 1)
).all()
```

```
q = session.query(
    User.age,
    db.func.count(User.id)
).group_by(User.age)

for age, count in q.all():
    print 'Users aged %d: %d' % (age, count)
```

```
session.query(Thread) \
    .filter(Thread.id == thread.id) \
    .update({Thread.view_count: Thread.view_count + 1})
```

```
thread = session.query(Thread).get(42)
thread.views = Thread.views + 1
session.commit()
```

the attribute temporarily will be an expression –
commit early!

```
session.query(PlayerStats) \
    .filter(PlayerStats.player_id == player.id) \
    .update({
        PlayerStats.score: db.func.greatest(
            PlayerStats.score,
            new_score
        )
    })
```

VIII The Sky is the Limit

```
from sqlalchemy.dialects.postgresql import UUID, ARRAY


class Post(Base):
    __tablename__ = 'posts'
    id = db.Column('post_id', UUID(as_uuid=True),
                   primary_key=True)
    tags = db.Column(ARRAY(db.String()))
```

```
from sqlalchemy.sql import expression as expr
from sqlalchemy.ext.compiler import compiles


class Explain(expr.Executable, expr.ClauseElement):

    def __init__(self, stmt, analyze=False):
        self.stmt = stmt
        self.analyze = analyze


@compiles(Explain)
def visit_explain(element, compiler, **kw):
    return 'EXPLAIN (FORMAT JSON, ANALYZE %s) %s' % (
        element.analyze and 'true' or 'false',
        compiler.process(element.stmt),
    )
```

```
from sqlalchemy.orm import Query
from yourframework import NotFound


class WebQuery(Query):

    def first_or_404(self):
        rv = self.first()
        if rv is None:
            raise NotFound()
        return rv
```

- Hybrid Properties
- Association Proxies
- Custom SQL Generation
- Database Extensions
- Update + Readback

- Automatic Partitioning
- Automatic Sharding
- Master+Slave Setup

Now build stuff!

That's it.
Now ask questions.

And add me on twitter: @mitsuhiko
Slides at lucumr.pocoo.org/talks

Like the talk: gittip.com/mitsuhiko
Also send something Mike's way: gittip.com/zzzeek