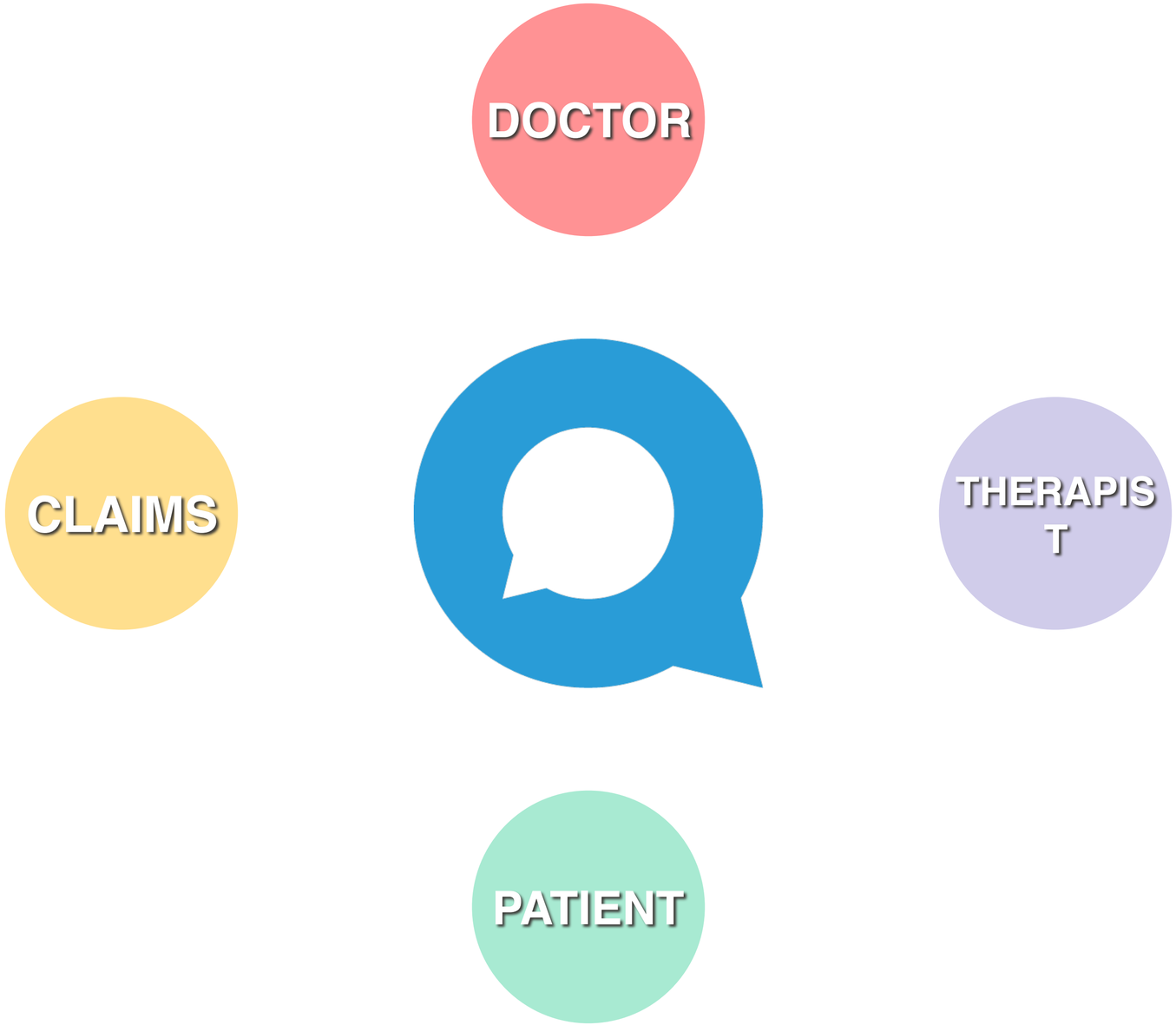
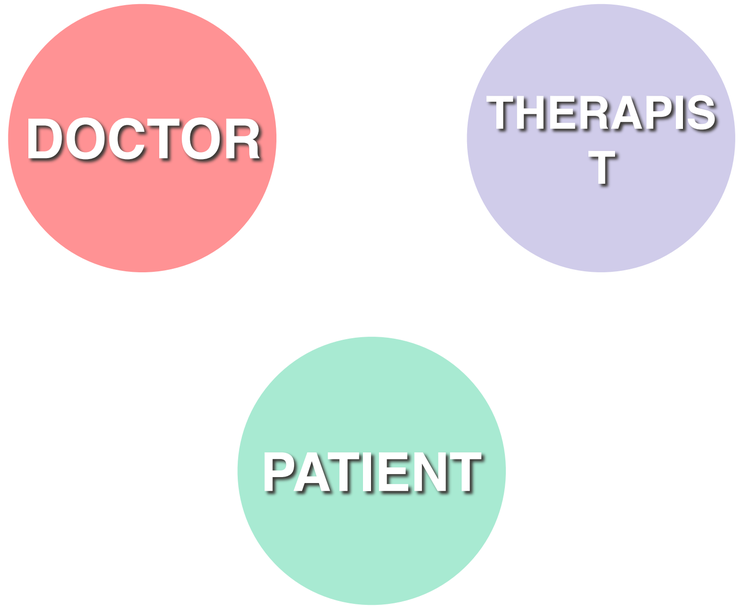
QUARTET PROVIDES A COLLABORATIVE PLATFORM TO IMPROVE QUALITY AND ACCESS TO BEHAVIORAL HEALTH CARE


DOCTOR
CLAIMS
THERAPIS
T
PATIENT

DATA SCIENCE ETL

REAL-TIME APP DATA



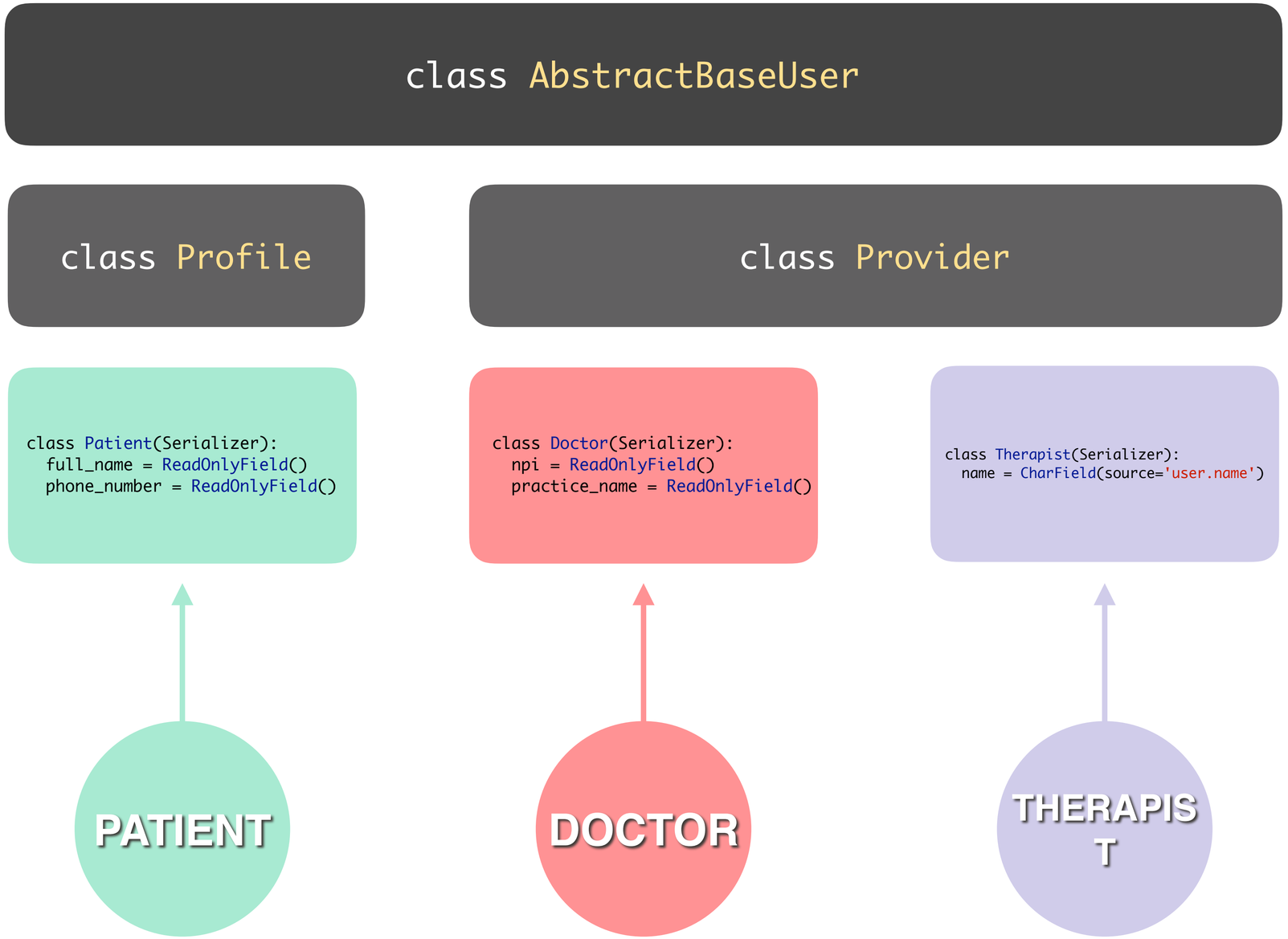
DATA SCIENCE ETL

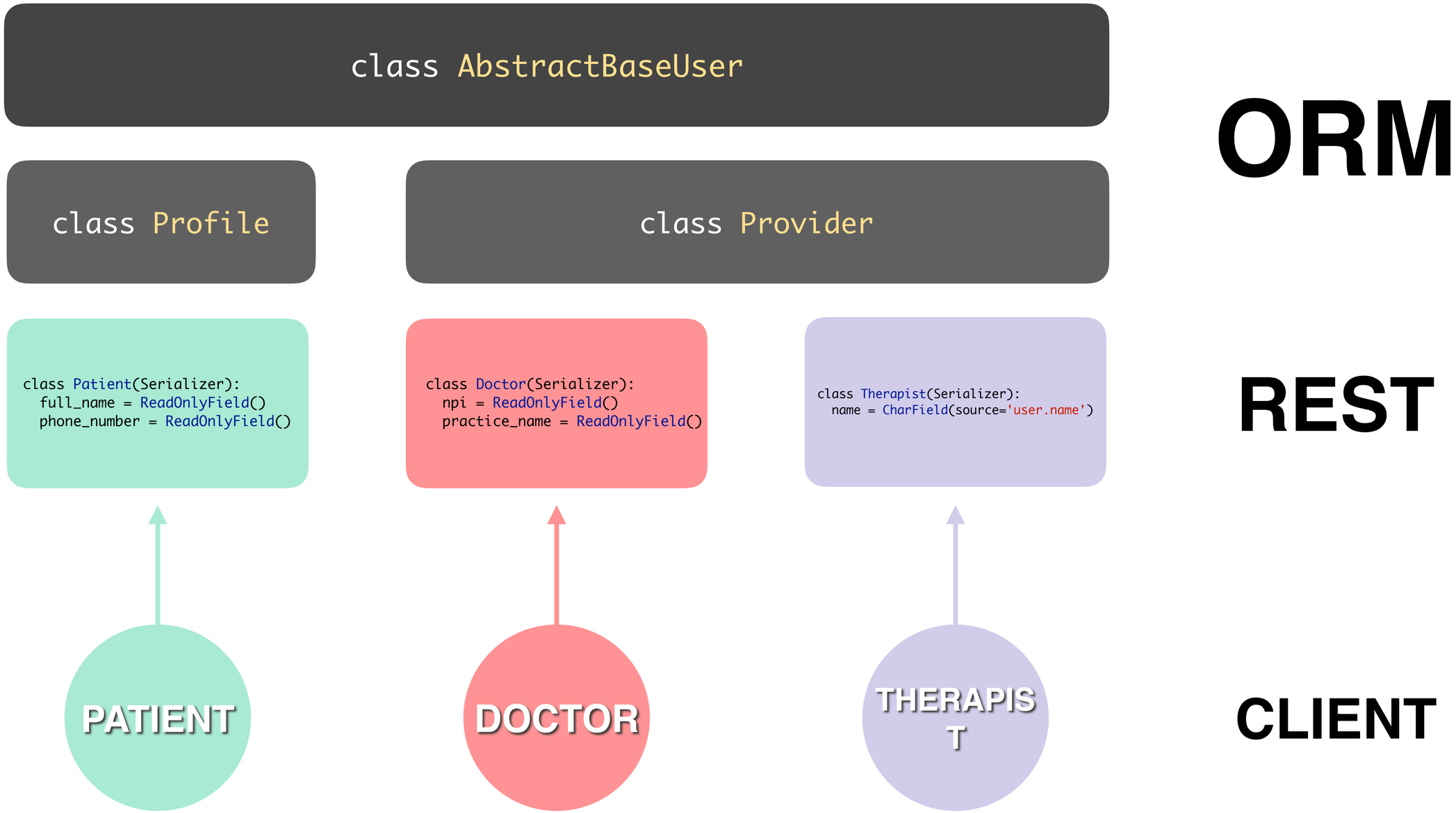
REAL-TIME APP DATA

```python
class Patient(Serializer):
  full_name = ReadOnlyField()
  phone_number = ReadOnlyField()
```

```python
class Doctor(Serializer):
  npi = ReadOnlyField()
  practice_name = ReadOnlyField()
```

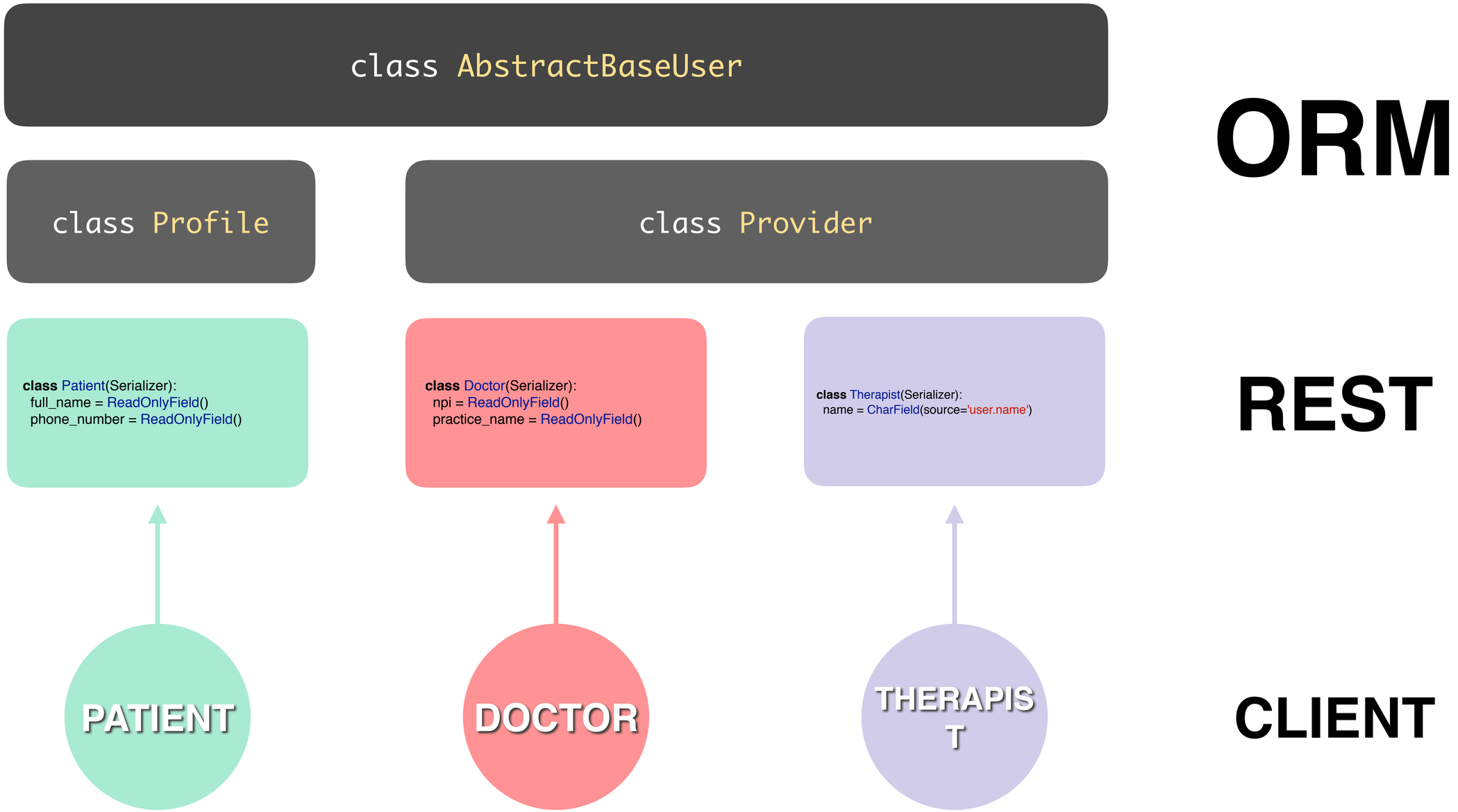
```python
class Therapist(Serializer):
  name = CharField(source='user.name')
```




class AbstractBaseUser
class Profile
class Provider
class Patient(Serializer):
full_name = ReadOnlyField()
phone_number = ReadOnlyField()
class Doctor(Serializer):
npi = ReadOnlyField()
practice_name = ReadOnlyField()
class Therapist(Serializer):
name = CharField(source='user.name')
PATIENT
DOCTOR
THERAPIST
ORM
REST
CLIENT

FUNDAMENTAL THEOREM OF SOFTWARE ENGINEERING

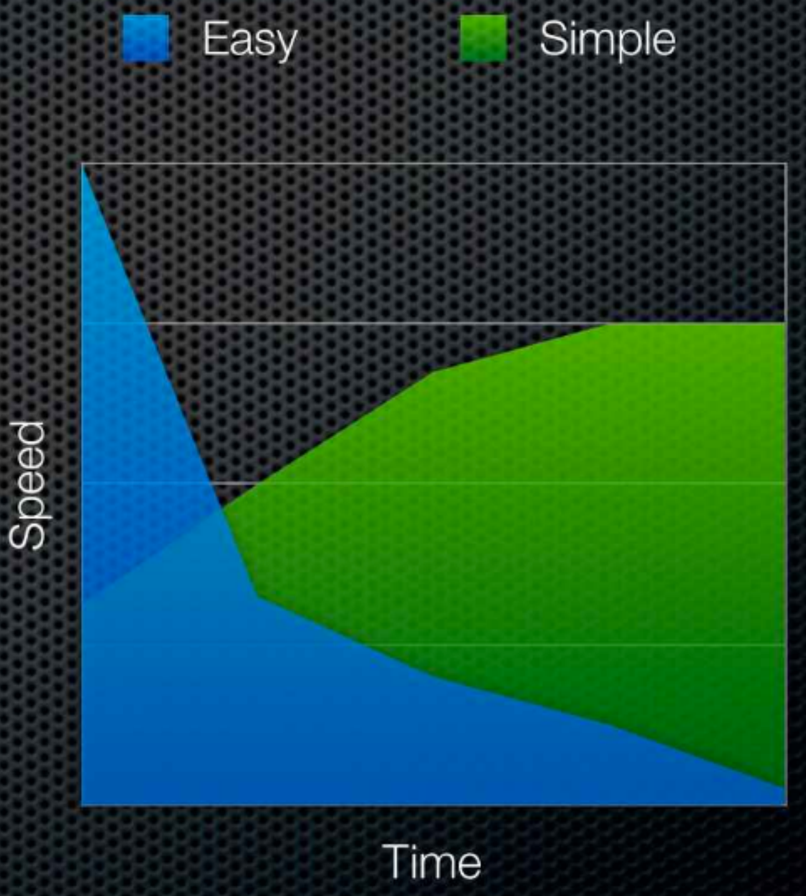
"We can solve any problem by introducing an extra level of indirection."


class AbstractBaseUser
class Profile
class Provider
class Patient(Serializer):
full_name = ReadOnlyField()
phone_number = ReadOnlyField()
class Doctor(Serializer):
npi = ReadOnlyField()
practice_name = ReadOnlyField()
class Therapist(Serializer):
name = CharField(source='user.name')
PATIENT
DOCTOR
THERAPIST
ORM
REST
CLIENT

```
class Therapist(Serializer):
  name = CharField(source='user.name')
```

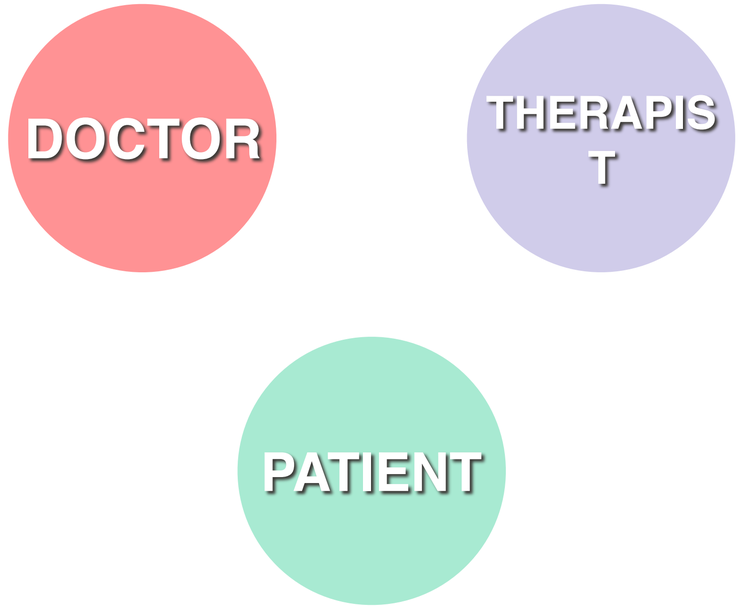
Development Speed

- Emphasizing ease gives early speed
- Ignoring complexity will slow you down over the long haul
- On throwaway or trivial projects, nothing much matters

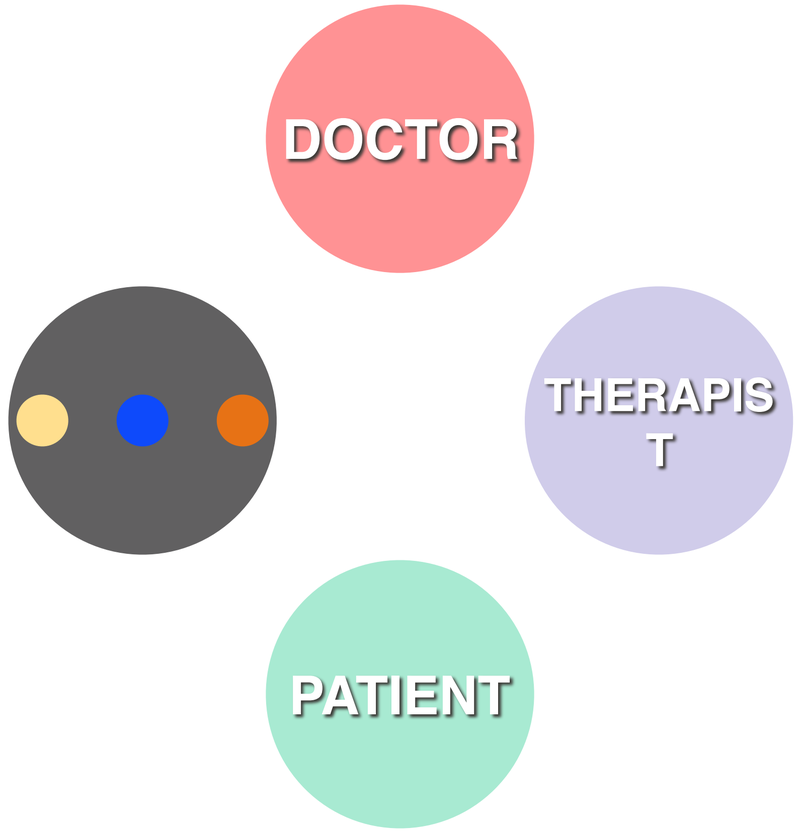


DATA SCIENCE ETL

REAL-TIME APP DATA



NORMATIVE DATA STORE



HIGH-LEVEL PRIORITIES RECAP

- **Flexible, query-driven API**
  - (empower front-end to not depend on back-end serializers)
- **Minimize layers between storage and access**
  - (simple, if any, abstractions)

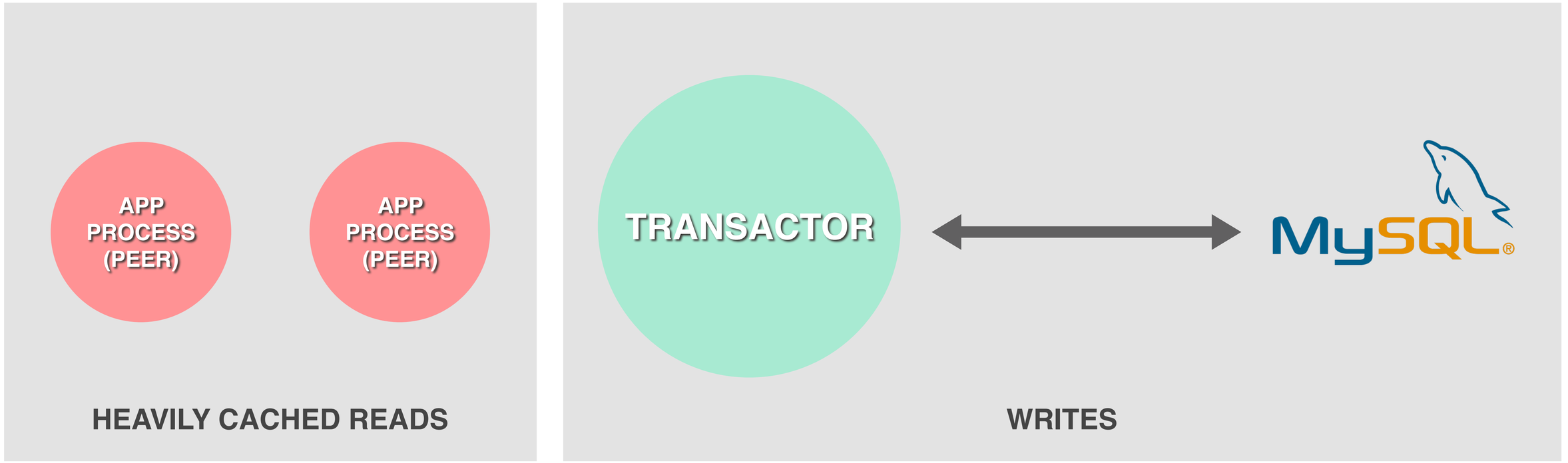
- **Support complex data science queries**
  - (in particular heavy reads)
- **Data must still be accessible by Spark**
  - (stepping stone)
- **Strong relational integrity in a normative data store**
  - (support features like entity resolution and improved data science models)

Datomic is a database system that simplifies the database by:

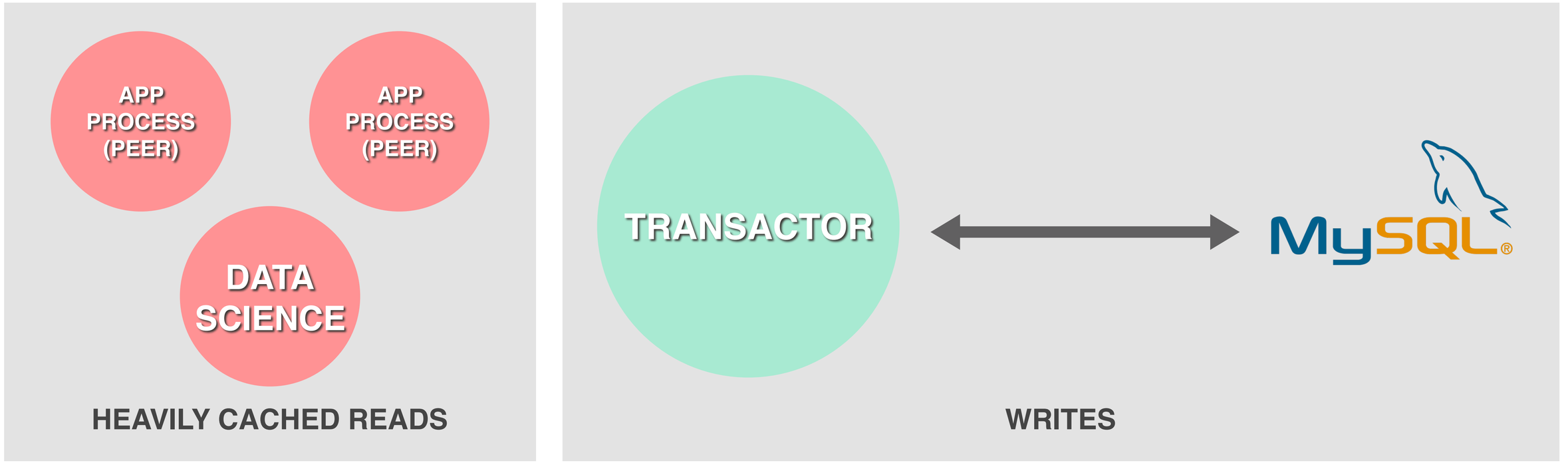
separating transactions process, storage and queries

adopting a simple data model based on facts

separating transactions process, storage and queries



separating transactions process, storage and queries



In Datomic, all data is stored as immutable tuples called datoms or simply facts

Entity Id	**A**ttribute	**V**alue	**T**ransaction Id
175462738	:patient/firstName	“Quentin”	131985334
175462738	:patient/lastLoggedIn	#inst “2016-04-12”	532723742

E A V T Row Access

Entity Id	Attribute	Value	Transaction Id
175462738	:patient/firstName	“Quentin”	131985334
175462738	:patient/ lastLoggedIn	#inst “2016-04-12”	532723742

A E V T Column Access

Attribute	Entity	Value	Transaction Id
:patient/firstName	175462738	“Quentin”	131985334
:patient/firstName	173637253	“Ralph”	532723742

A V E T Explicit Index Lookup

Attribute	Value	Entity Id	Transaction Id
:patient/firstName	“Quentin”	175462738	131985334
:patient/firstName	“Ralph”	173637253	532723742

V A E T Reverse Index

Value	Attribute	Entity Id	Transaction Id
“Quentin”	:patient/firstName	175462738	131985334
“Quentin”	:patient/firstName	173637253	532723742

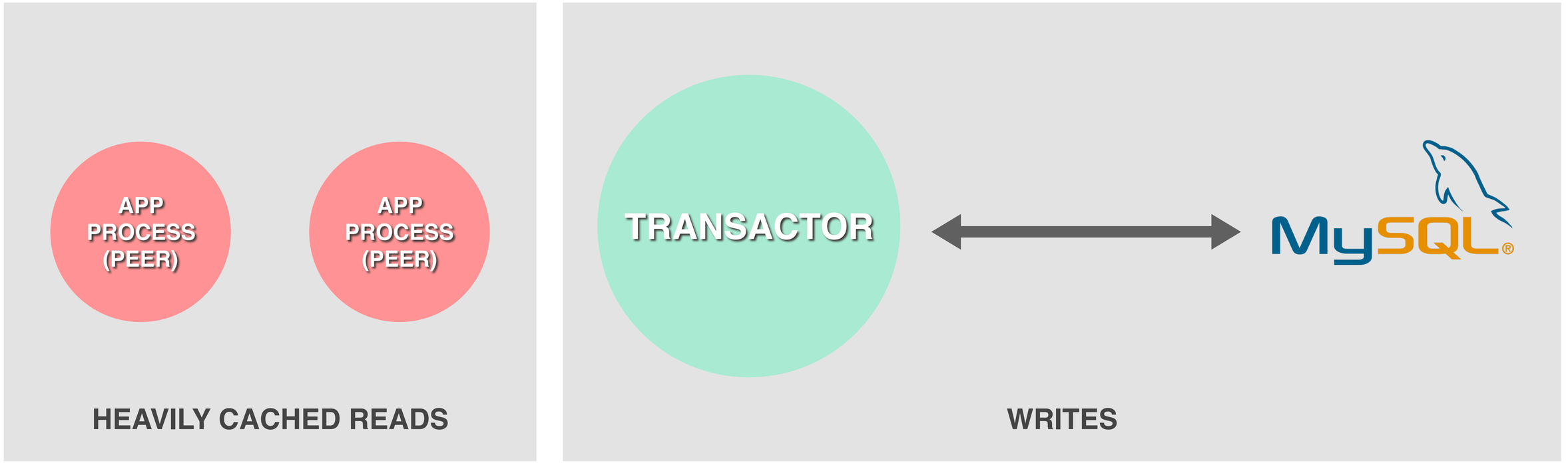
Entity Id	Attribute	Value	Transaction Id
175462738	:patient/firstName	“Quentin”	131985334
175462738	:patient/lastLoggedIn	#inst “2016-04-12”	532723742

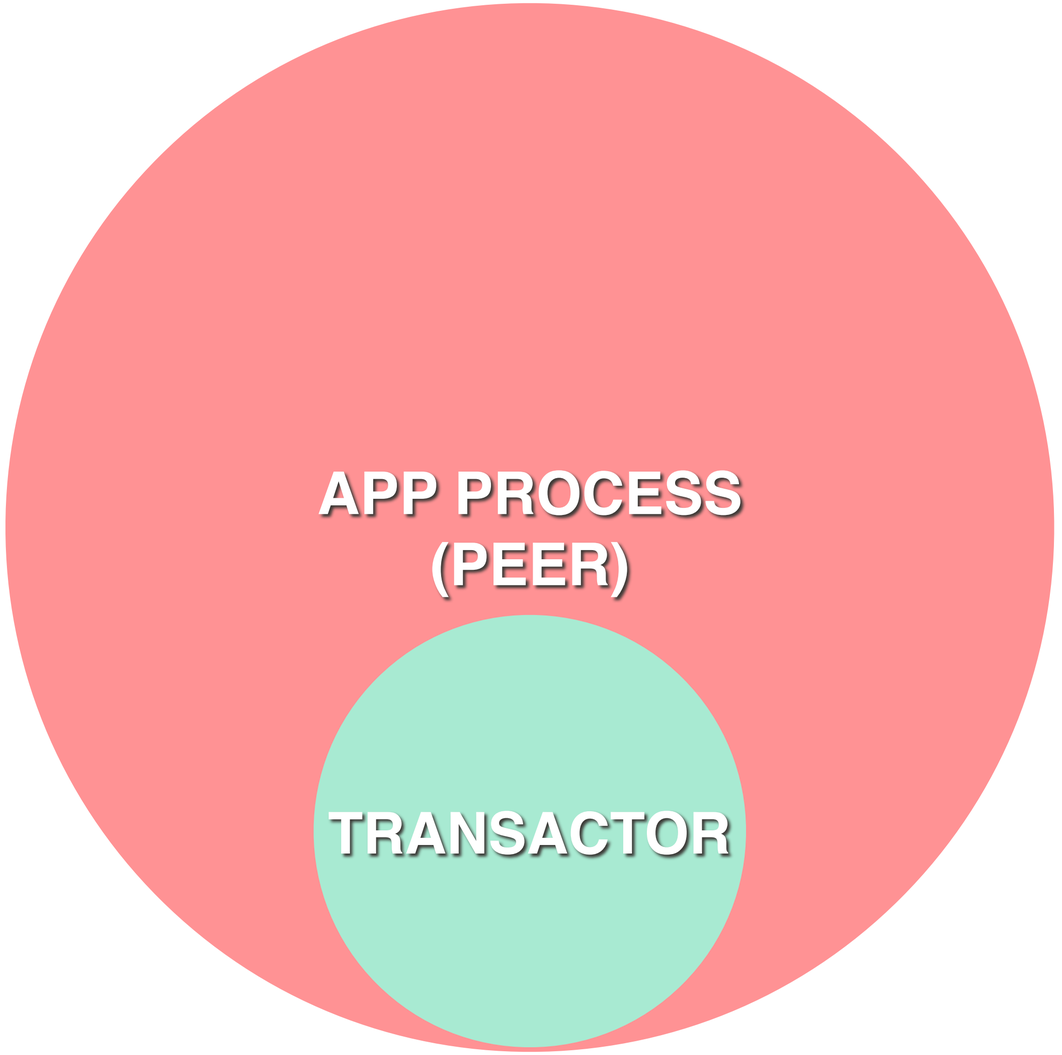
Entity Id	Attribute	Value	Transaction Id	Op
175462738	:patient/firstName	“Quentin”	131985334	added
175462738	:patient/lastLoggedIn	#inst “2016-04-12”	532723742	added
175462738	:patient/receivedCare	TRUE	998346612	retracted

In Datomic, a database is simply an accumulation of immutable facts at any given point in time

Since facts are immutable, we can treat the database as an immutable value as well

This allows for novel use cases where the database is a first-class value in the app


APP PROCESS (PEER)
APP PROCESS (PEER)
HEAVILY CACHED READS
TRANSACTOR
MySQL®
WRITES


APP PROCESS
(PEER)
TRANSACTOR

IN-MEMORY DATABASE

```
(let [uri  "datomic:mem://db"
      conn (d/connect (doto uri d/create-database))
      db   (d/db conn)])
```

TIME TRAVEL

(d/**as-of** db tx-or-date)

QUERYING DATA

178548364 :user/firstName

E **A**

QUERYING DATA

IT’S JUST EDN!

```
(d/q ‘[:find ?name
       :where [ 178548364 :user/firstName ?name]] db)
```

E **A** **V**

WRITING DATA

#db/id[:db.part/user] :user/firstName "Mark"

E **A** **V**

WRITING DATA

IT’S JUST EDN!

```
(d/transact conn
                    [[:db/add        #db/id[:db.part/user] :user/firstName "Mark"                    ]])
```

E **A** **V**

EXPORT FACTS TO SPARK

```
(defn serialize-facts
  [attribute db filename]
  (-> attribute
      (attribute->facts db)
      (facts->avro filename)))
```

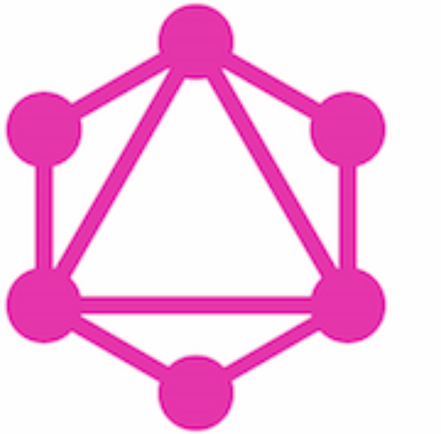
```
(defn attribute->facts
  [attribute db]
  (d/q '[:find ?e ?a ?v ?tx-id ?added
         :in $ ?a
         :where
         [?e ?a ?v ?tx-id ?added]] db attribute))
```

HIGH-LEVEL PRIORITIES RECAP

- **Flexible, query-driven API**
  - (empower front-end to not depend on back-end serializers)
- ~~**Minimize layers between storage and access**~~
  - ~~(simple abstractions for straightforward access patterns)~~

- ~~**Support complex data science queries**~~
  - ~~(in particular heavy reads)~~
- ~~**Data must still be accessible by Spark**~~
  - ~~(stepping stone)~~
- ~~**Strong relational integrity in a normative data store**~~
  - ~~(support features like entity resolution and improved data science models)~~

QUERY-DRIVEN API




FALCOR

PULL API

(d/**pull** db '[:patient/firstName **{:patient/pcp [*]}**] patient)

RING MIDDLEWARE

```
(-> query-api-handler
    (wrap-query-api-response-formatter)
    (wrap-query-api-executor db)
    (wrap-query-api-parser))
```

```
{
    "nora-patients": {
        "model": "patient",
        "filters": [
          ["firstName", "Nora"]
        ],
        "select": [
            "firstName",
            “lastName”
        ]
    }
}
```

```
{
  "nora-patients": [
    {
      "firstName": "Nora",
      "lastName": "Martinez"
    },
    {
      "firstName": "Nora",
      "lastName": "Bell"
    },
    {
      "firstName": "Nora",
      "lastName": "Sutherland"
    },
    {
      "firstName": "Nora",
      "lastName": "Smith"
    },
    ...
  ]
}
```

HIPAA COMPLIANCE

1.9.0-alpha1

```
(s/def :provider/score
  (s/and int?
         #(>= % 1)))
```

```
(s/valid? :provider/score 5
=> true
```

```
(let [score-generator (s/gen :provider/score)]
  (gen/generate score-generator))

=> 2249863
```

```
(s/def :db/id
  (s/with-gen #(instance? datomic.db.DbId %)
    (fn []
      (gen/fmap (fn [id] (d/tempid :db.part/user id))
                (gen/choose -1000000 -1)))))
```

```
(s/def ::db
  (s/with-gen #(instance? datomic.db.Db %)
    (fn []
      (gen/let [provider-entities provider-entitity-generator]
        (-> (empty-db)
            (transact provider-entities))))))
```

```
(defspec attribute->facts
  10
  (prop/for-all [fact (s/gen ::db)]
    (let [[e a v tx-id added] fact]
      (is (int? e))
      (is (= a :provider/score))
      (is (string? v)))))
```

THANK YOU !

QUESTIONS ?