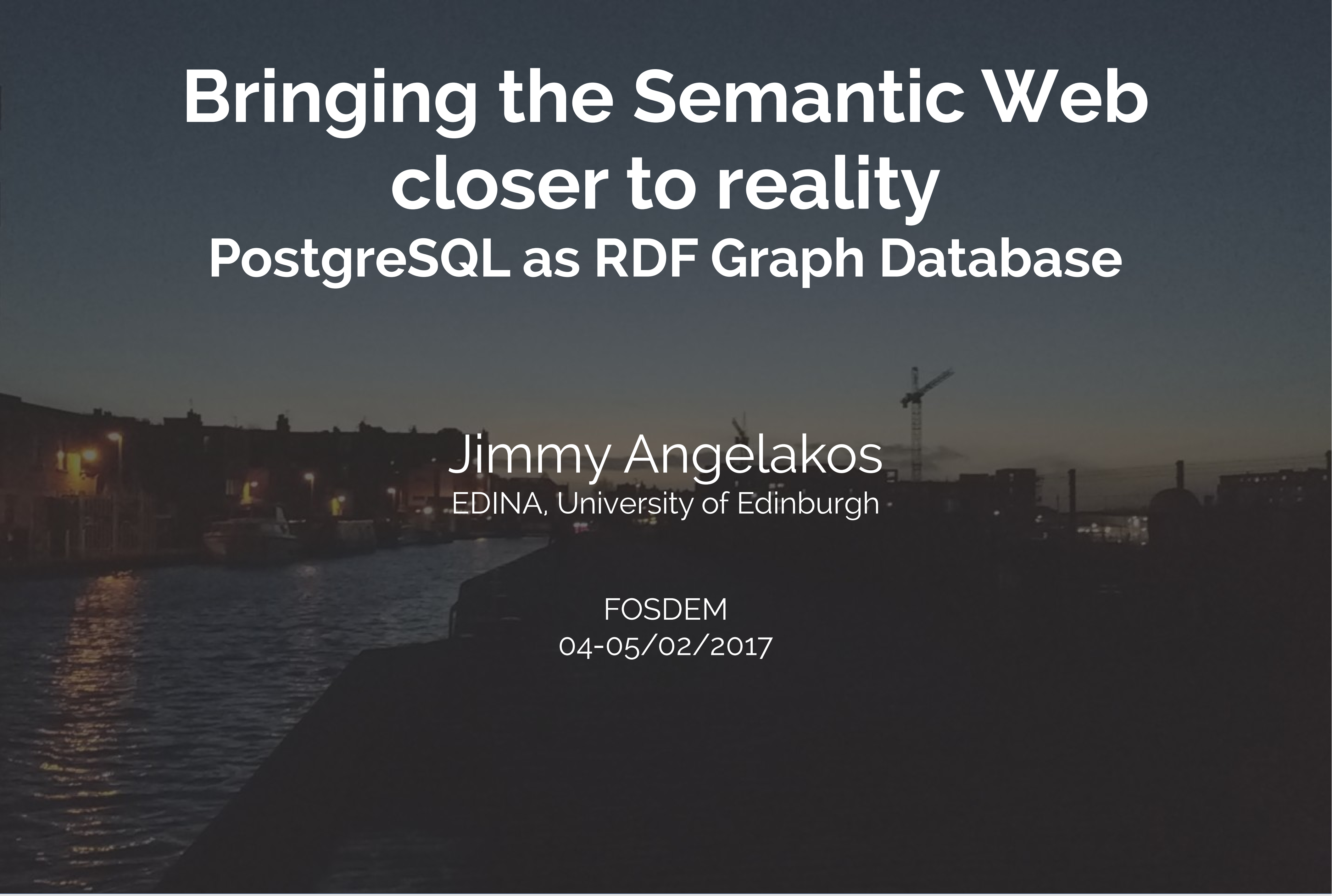


Bringing the Semantic Web closer to reality

PostgreSQL as RDF Graph Database

Jimmy Angelakos

EDINA, University of Edinburgh

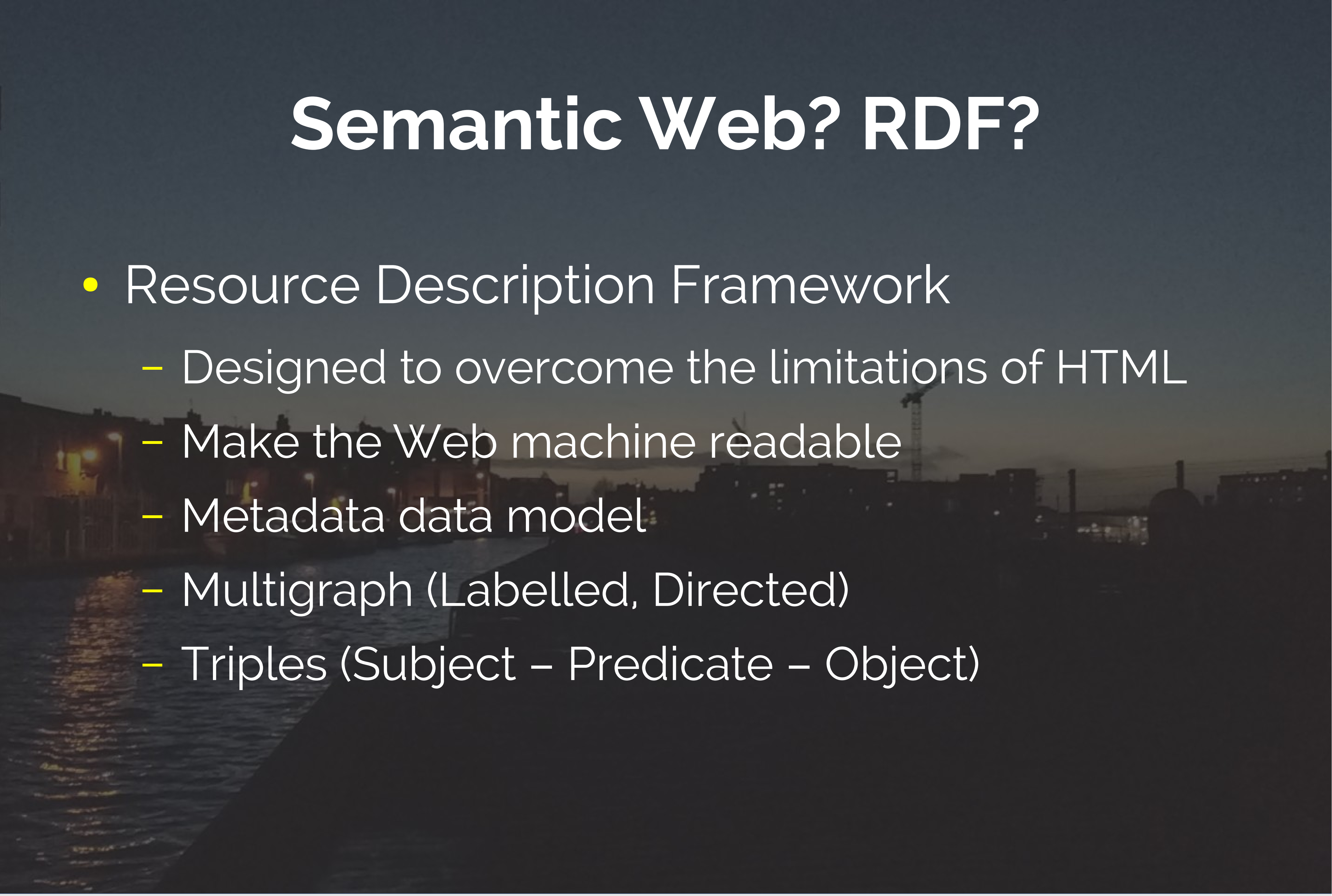
FOSDEM

04-05/02/2017

or how to export your data to someone who's expecting RDF

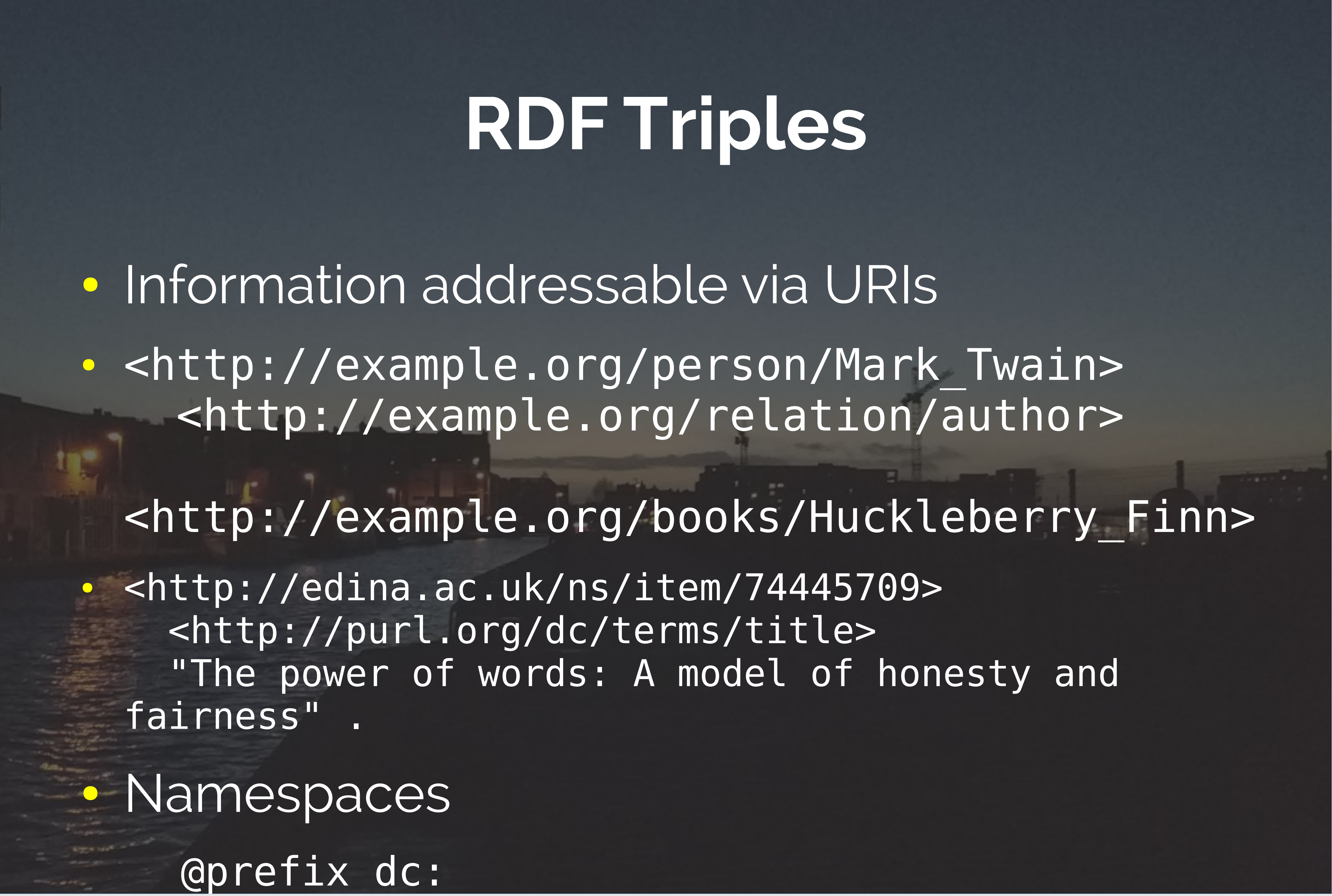
Jimmy Angelakos
EDINA, University of Edinburgh

FOSDEM
04-05/02/2017



Semantic Web? RDF?

- Resource Description Framework
  - Designed to overcome the limitations of HTML
  - Make the Web machine readable
  - Metadata data model
  - Multigraph (Labelled, Directed)
  - Triples (Subject – Predicate – Object)



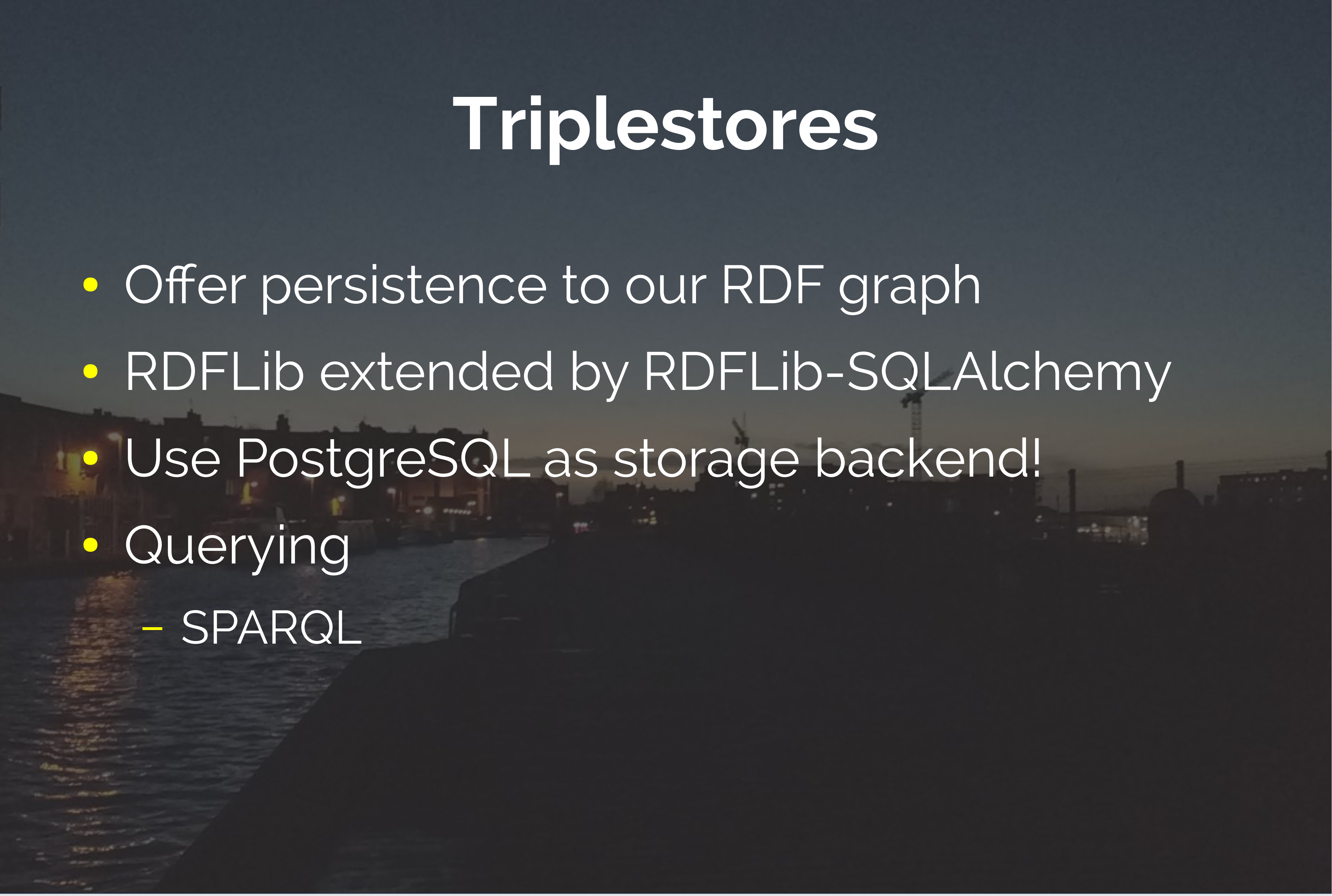
RDF Triples

- Information addressable via URIs

-
  ```
  <http://example.org/person/Mark_Twain>
    <http://example.org/relation/author>

  <http://example.org/books/Huckleberry_Finn>
  ```

-
  ```
  <http://edina.ac.uk/ns/item/74445709>
    <http://purl.org/dc/terms/title>
    "The power of words: A model of honesty and
  fairness" .
  ```
- Namespaces

  ```
  @prefix dc:
  <http://purl.org/dc/elements/1.1/> .

  dc:title "RDF/XML Syntax Specification
  ```



Triplestores

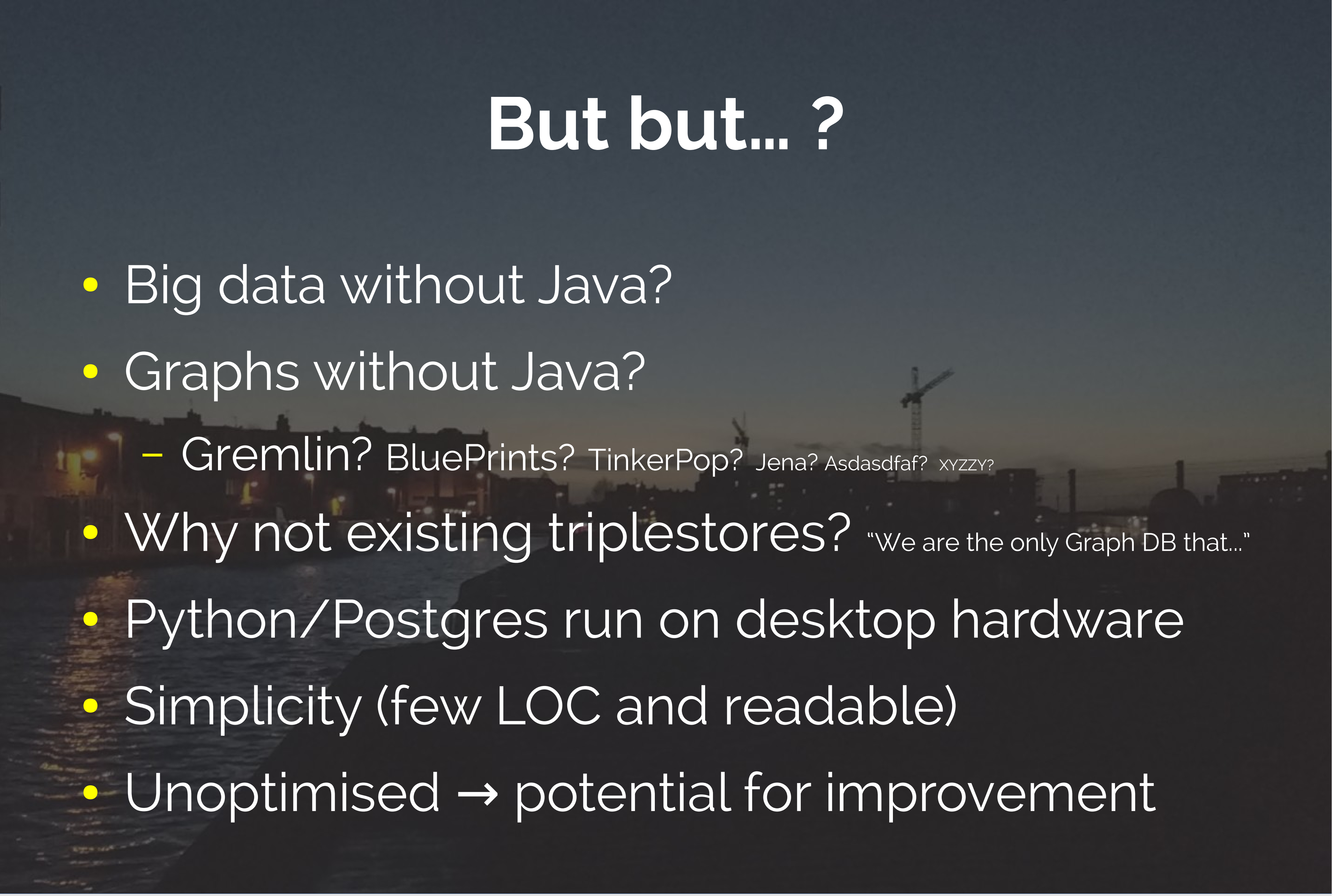
- Offer persistence to our RDF graph
- RDFLib extended by RDFLib-SQLAlchemy
- Use PostgreSQL as storage backend!
- Querying
  - SPARQL

```
{                                                        +
    "DOI": "10.1007/11757344_1",                         +
    "URL": "http://dx.doi.org/10.1007/11757344_1",       +
    "type": "book-chapter",                              +
    "score": 1.0,                                        +
    "title": [                                           +
        "CrossRef Listing of Deleted DOIs"               +
    ],                                                   +
    "member": "http://id.crossref.org/member/297",       +
    "prefix": "http://id.crossref.org/prefix/10.1007",   +
    "source": "CrossRef",                                +
    "created": {                                         +
        "date-time": "2006-10-19T13:32:01Z",             +
        "timestamp": 1161264721000,                      +
        "date-parts": [                                  +
            [                                            +
                2006,                                    +
                10,                                      +
                19                                       +
            ]                                            +
        ]                                                +
    },                                                   +
    "indexed": {                                         +
        "date-time": "2015-12-24T00:59:48Z",             +
        "timestamp": 1450918788746,                      +
        "date-parts": [                                  +
import psycopg2
from rdflib import plugin, Graph, Literal, URIRef
from rdflib.namespace import Namespace
from rdflib.store import Store
import rdflib_sqlalchemy

EDINA  = Namespace('http://edina.ac.uk/ns/')
PRISM  =
Namespace('http://prismstandard.org/namespaces/basic/2.1/')

rdflib_sqlalchemy.registerplugins()
dburi =
URIRef('postgresql+psycopg2://myuser:mypass@localhost/rdfgraph'
)
ident = EDINA.rdfgraph
store = plugin.get('SQLAlchemy', Store)(identifier=ident)
gdb = Graph(store, ident)
gdb.open(dburi, create=True)
gdb.bind('edina',  EDINA)
gdb.bind('prism',  PRISM)

item = EDINA['item/' + str(1)]
triples = []
triples += (item, RDF.type, EDINA.Item),
triples += (item, PRISM.doi, Literal('10.1002/crossmark_policy'),
gdb.addN(t + (gdb,) for t in triples)
gdb.serialize(format='turtle')
```

BIG DATA

Super size me!

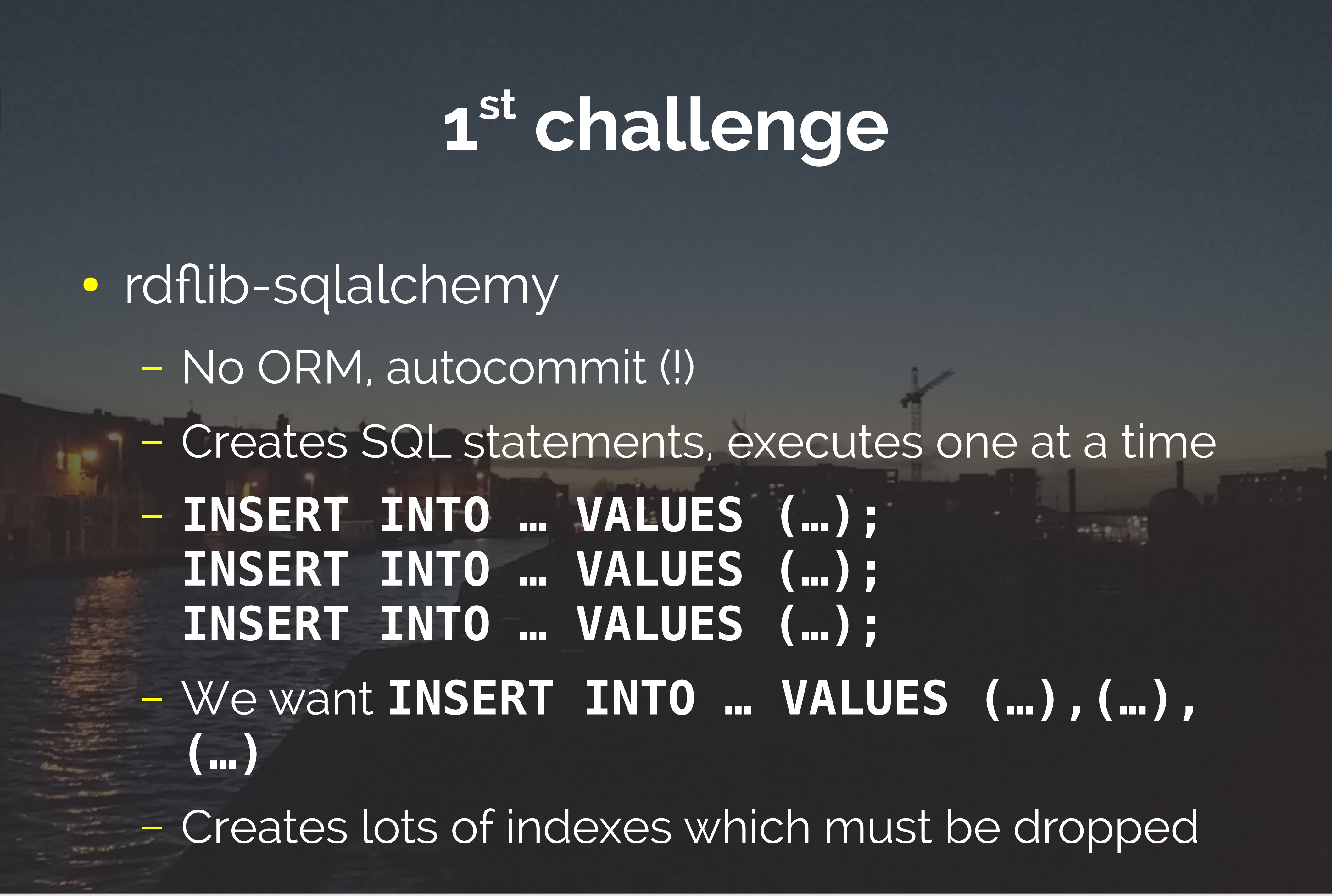
- Loop over original database contents
- Create triples
- Add them to graph efficiently
- Serialise graph efficiently



But but... ?

- Big data without Java?
- Graphs without Java?
  - Gremlin? BluePrints? TinkerPop? Jena? Asdasdfaf? XYZZY?
- Why not existing triplestores? "We are the only Graph DB that..."
- Python/Postgres run on desktop hardware
- Simplicity (few LOC and readable)
- Unoptimised → potential for improvement

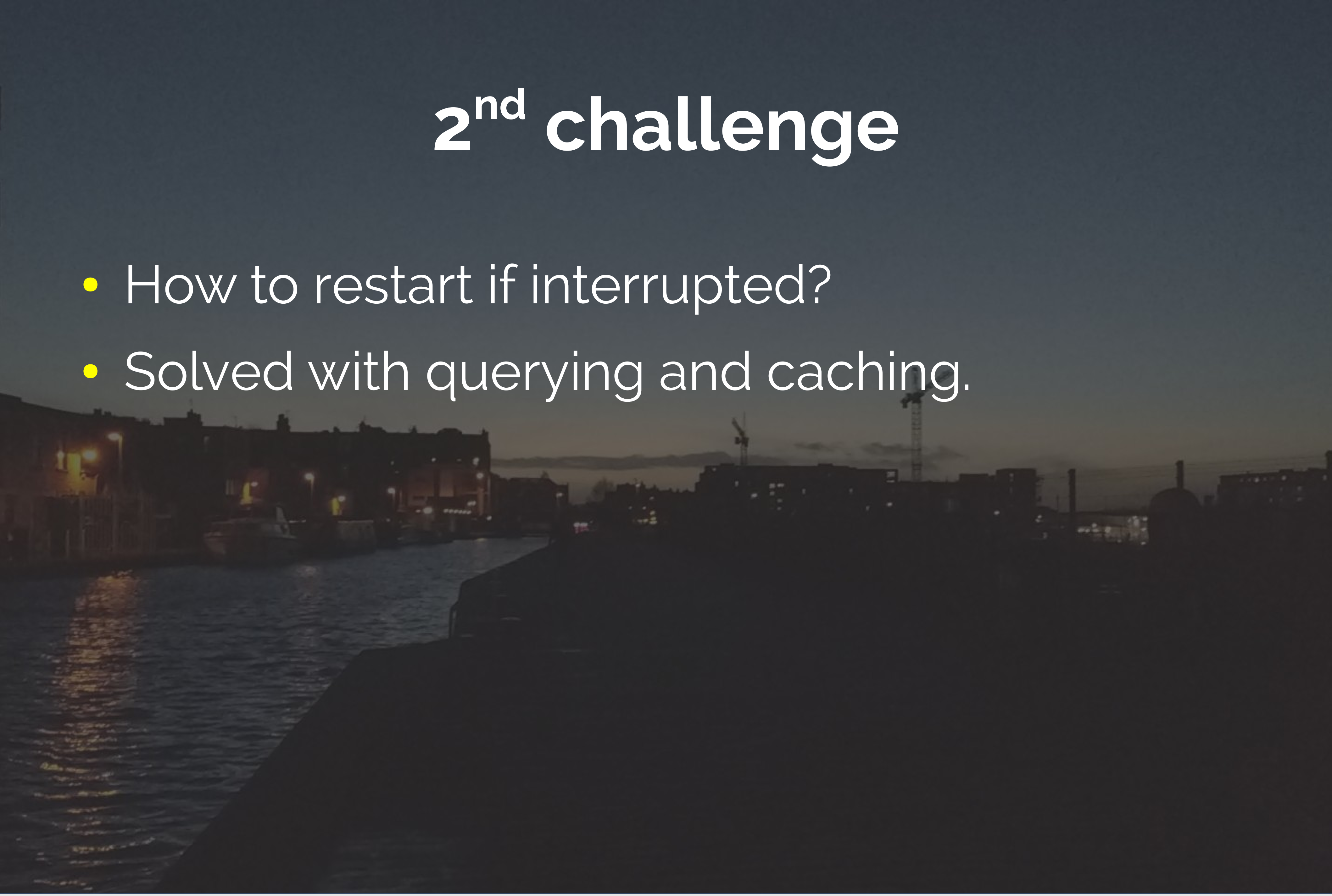
```
conn = psycopg2.connect(database='mydb', user='myuser')
cur = conn.cursor(cursor_factory=psycopg2.extras.DictCursor)
seqcur = conn.cursor()
seqcur.execute("""
                  CREATE SEQUENCE IF NOT EXISTS item;
               """)
conn.commit()
cur = conn.cursor('serverside_cur',
                  cursor_factory=psycopg2.extras.DictCursor)
cur.itersize = 50000
cur.arraysize = 10000
cur.execute("""
               SELECT data
               FROM mytable
            """)
while True:
    recs = cur.fetchmany(10000)
    if not recs:
        break
    for r in recs:
        if 'DOI' in r['data'].keys():
            seqcur.execute("SELECT nextval('item')")
            item = EDINA['item/' + str(seqcur.fetchone()[0])]
            triples += (item, RDF.type, EDINA.Item),
            triples += (item, PRISM.doi, Literal(r['data']['DOI'])),
```



1st challenge

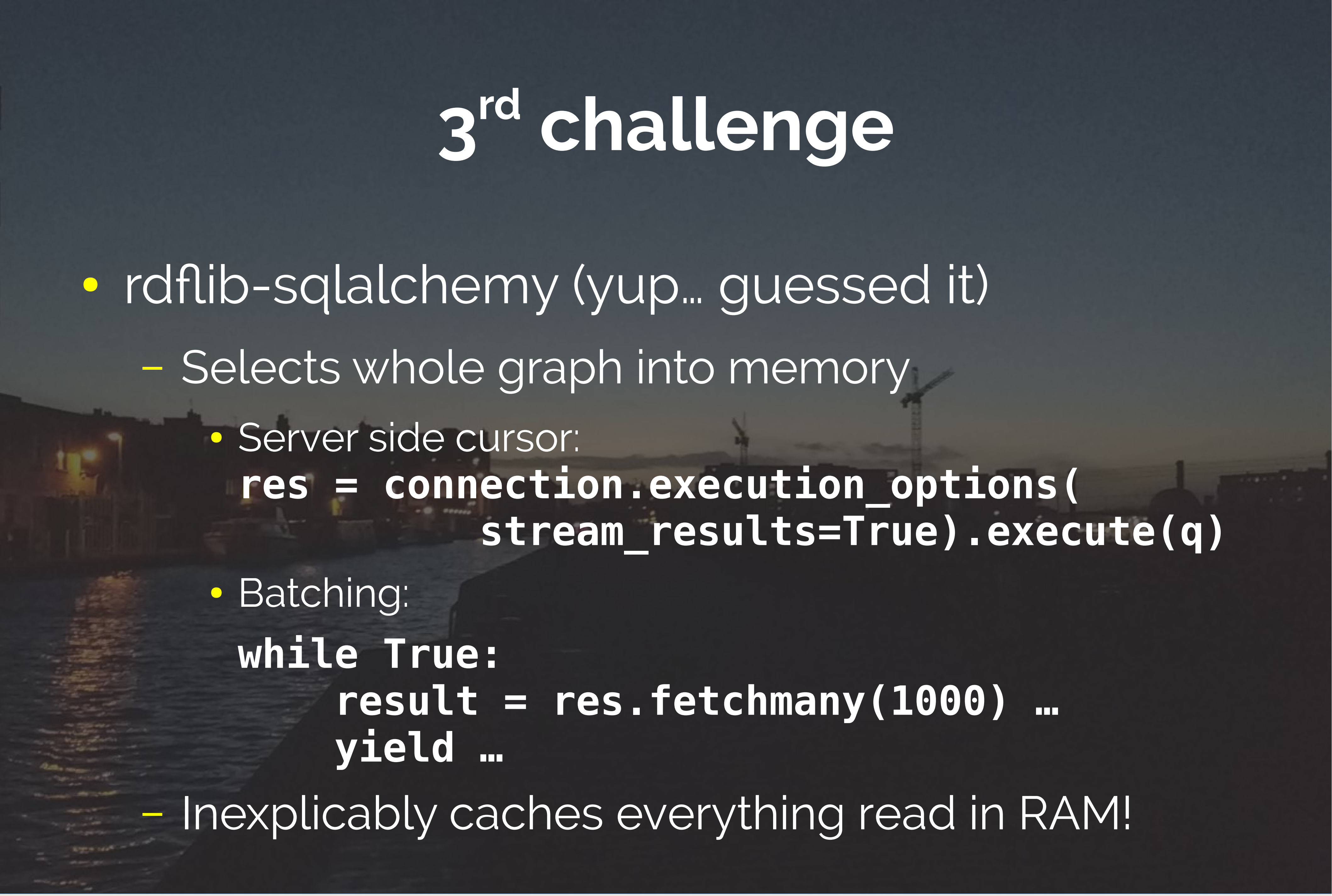
- rdflib-sqlalchemy
  - No ORM, autocommit (!)
  - Creates SQL statements, executes one at a time
  - **INSERT INTO … VALUES (…);**
INSERT INTO … VALUES (…);
INSERT INTO … VALUES (…);
  - We want **INSERT INTO … VALUES (…),(…),(…)**
  - Creates lots of indexes which must be dropped

2nd challenge



- How to restart if interrupted?
- Solved with querying and caching.

```
from rdflib.plugins.sparql import prepareQuery
orgQ = prepareQuery("""
                       SELECT ?org ?pub
                       WHERE { ?org a foaf:Organization .
                               ?org rdfs:label ?pub . }
                    """, initNs = { 'foaf': FOAF, 'rdfs': RDFS })
orgCache = {}
for o in gdb.query(orgQ):
    orgCache[o[1].toPython()] = URIRef(o[0].toPython())

        if 'publisher' in r['data'].keys():
            publisherFound = False
            if r['data']['publisher'] in orgCache.keys():
                publisherFound = True
                triples += (item, DCTERMS.publisher, orgCache[r['data']
                                                        ['publisher']]),
            if not publisherFound:
                seqcur.execute("SELECT nextval('org')")
                org = EDINA['org/' + str(seqcur.fetchone()[0])]
                orgCache[r['data']['publisher']] = org
                triples += (org, RDF.type, FOAF.Organization),
                triples += (org, RDFS.label, Literal(r['data']
                                                      ['publisher'])),
```



3rd challenge

- rdflib-sqlalchemy (yup... guessed it)
  - Selects whole graph into memory
    - Server side cursor:

```
res = connection.execution_options(
          stream_results=True).execute(q)
```

    - Batching:

```
while True:
    result = res.fetchmany(1000) …
    yield …
```

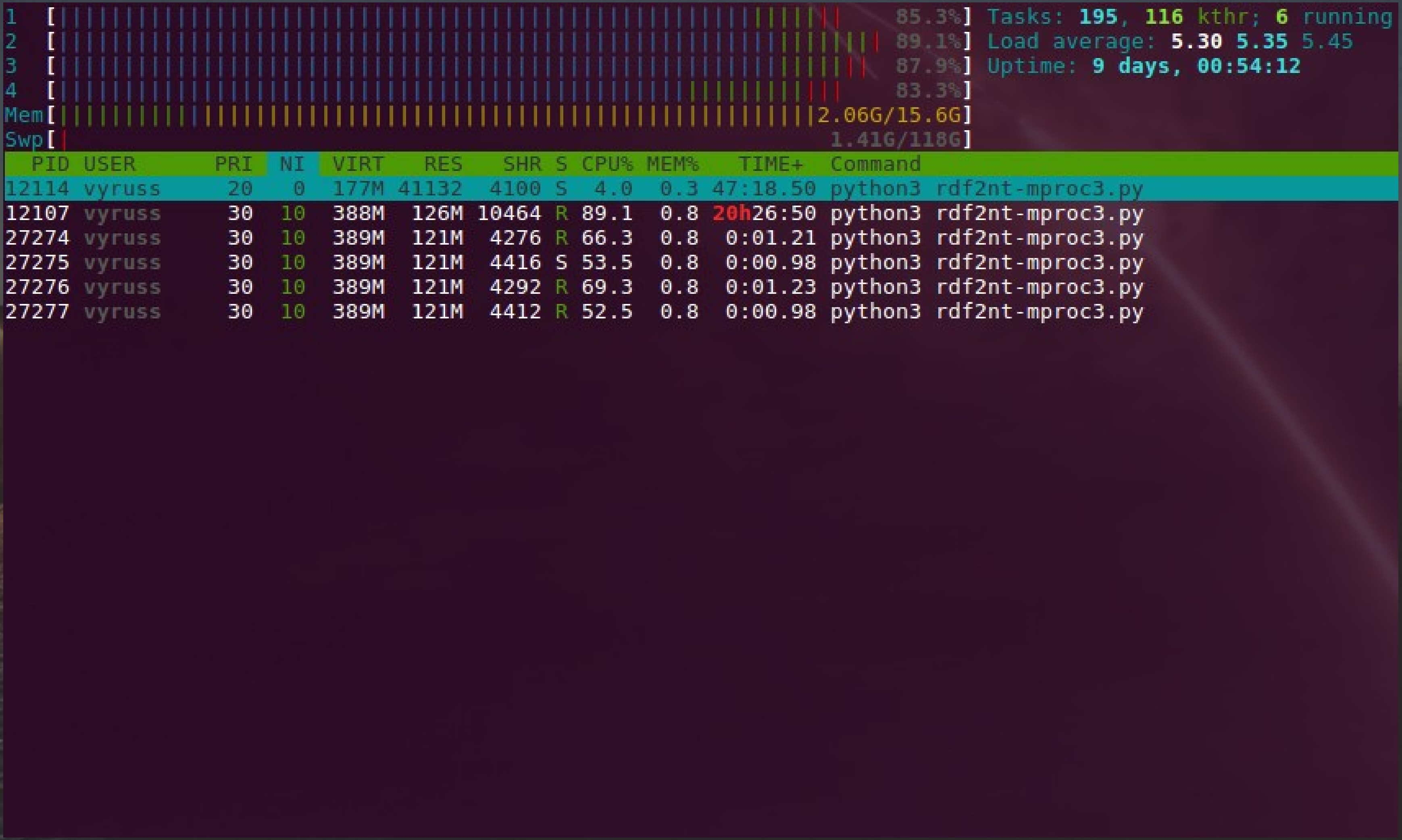
  - Inexplicably caches everything read in RAM!

4th challenge

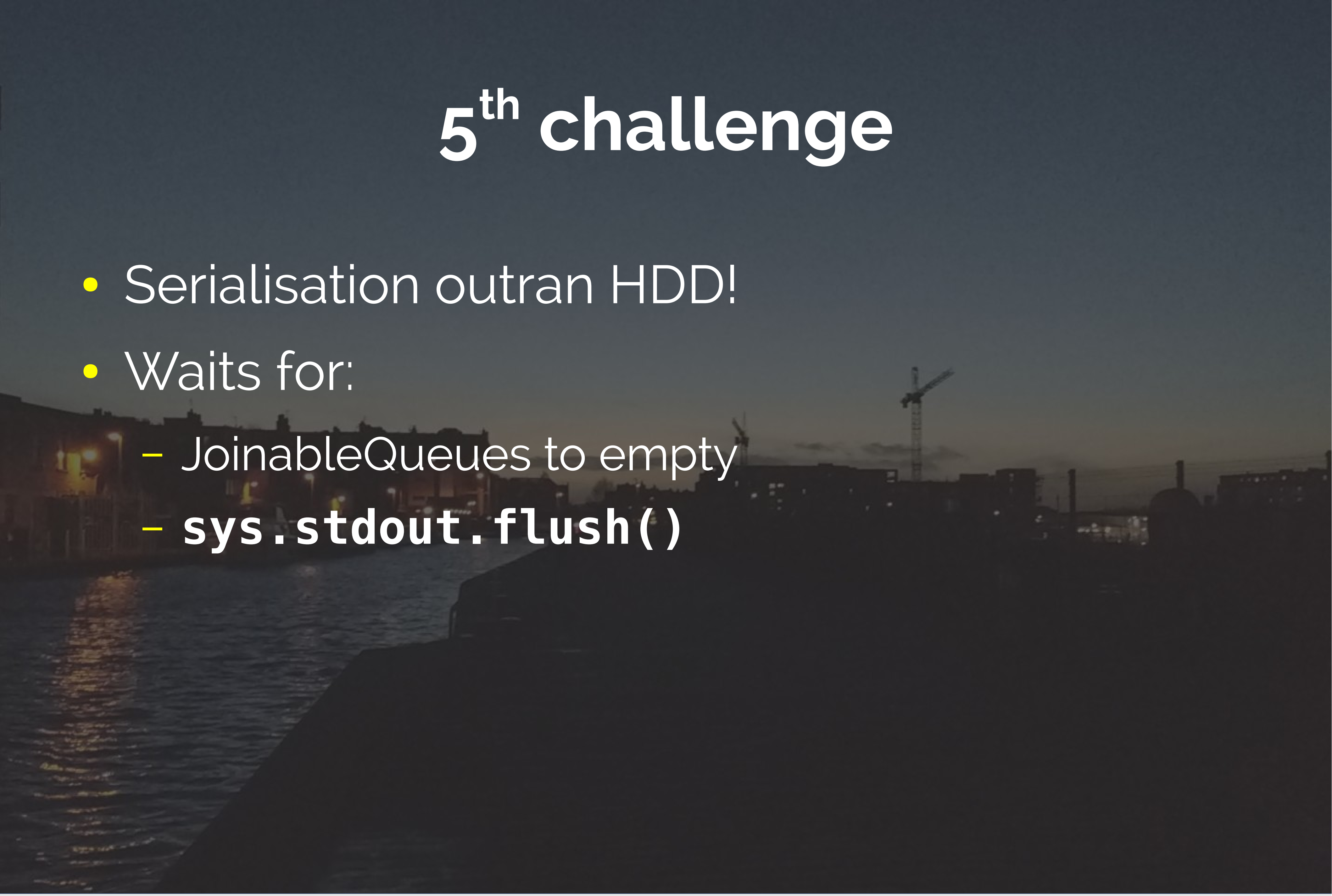
- Serialise efficiently! → Multiprocessing
  - Processes, JoinableQueues
- Turtle: Unsuitable → N-Triples
  - UNIX magic

```
python3 rdf2nt.py |
split -a4 -d -C4G
--additional-suffix=.nt
--filter='gzip > $FILE.gz' -
exported/rdfgraph_
```

Desktop hardware...



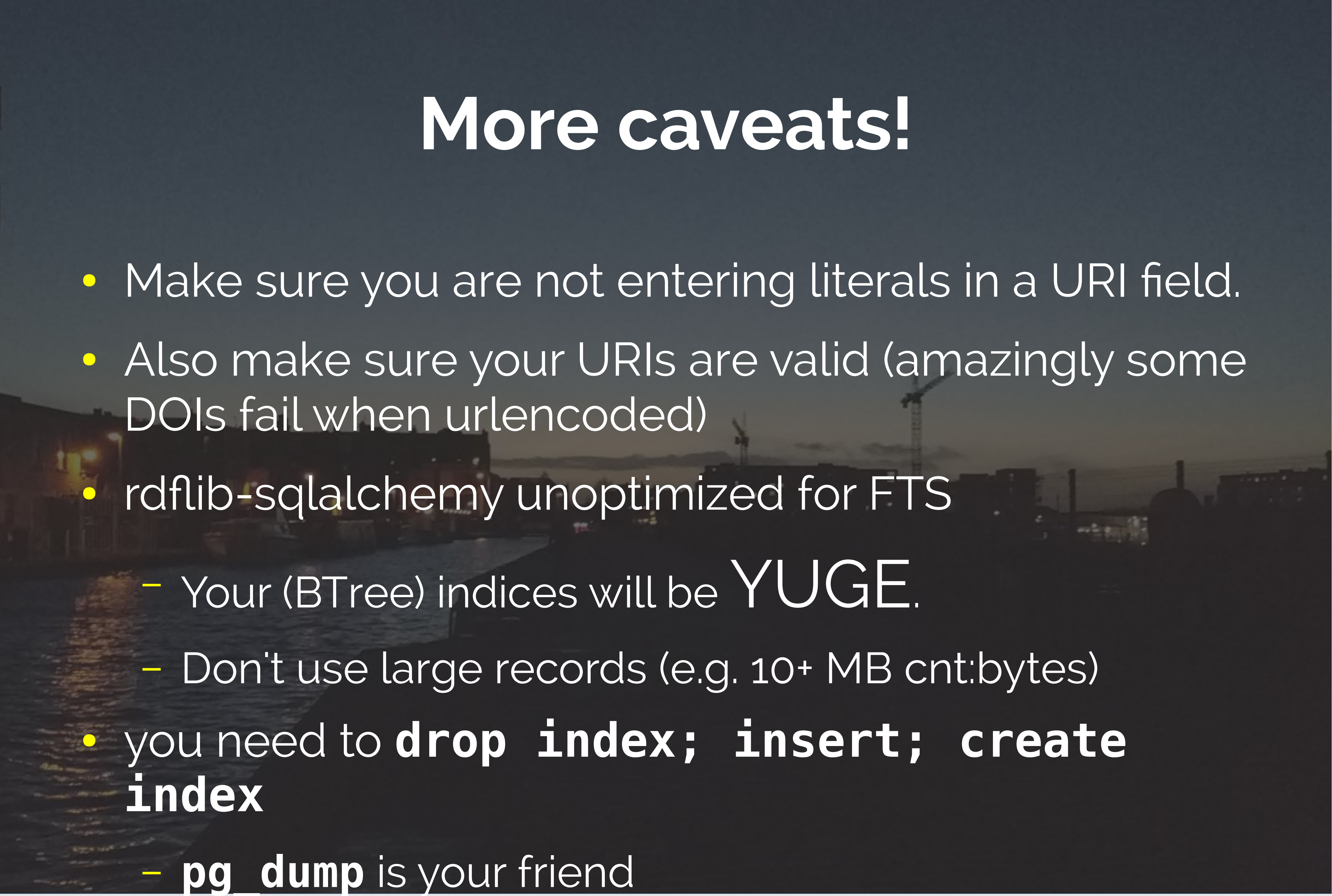
```
1  [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||    85.3%] Tasks: 195, 116 kthr; 6 running
2  [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 89.1%] Load average: 5.30 5.35 5.45
3  [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||  87.9%] Uptime: 9 days, 00:54:12
4  [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||    83.3%]
Mem[||||||||||||||||||||||||||||||||||||||||||||||||||||||||||2.06G/15.6G]
Swp[|                                                          1.41G/118G]
  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
12114 vyruss     20   0  177M 41132  4100 S  4.0  0.3 47:18.50 python3 rdf2nt-mproc3.py
12107 vyruss     30  10  388M  126M 10464 R 89.1  0.8 20h26:50 python3 rdf2nt-mproc3.py
27274 vyruss     30  10  389M  121M  4276 R 66.3  0.8  0:01.21 python3 rdf2nt-mproc3.py
27275 vyruss     30  10  389M  121M  4416 S 53.5  0.8  0:00.98 python3 rdf2nt-mproc3.py
27276 vyruss     30  10  389M  121M  4292 R 69.3  0.8  0:01.23 python3 rdf2nt-mproc3.py
27277 vyruss     30  10  389M  121M  4412 R 52.5  0.8  0:00.98 python3 rdf2nt-mproc3.py
```



5th challenge

- Serialisation outran HDD!
- Waits for:
  - JoinableQueues to empty
  - `sys.stdout.flush()`

```
rdfgraph=> \dt
                       List of relations
 Schema |               Name                | Type  |  Owner
--------+-----------------------------------+-------+----------
 public | kb_a8f93b2ff6_asserted_statements | table | myuser
 public | kb_a8f93b2ff6_literal_statements  | table | myuser
 public | kb_a8f93b2ff6_namespace_binds     | table | myuser
 public | kb_a8f93b2ff6_quoted_statements   | table | myuser
 public | kb_a8f93b2ff6_type_statements     | table | myuser
(5 rows)

rdfgraph=> \x
Expanded display is on.
rdfgraph=> select * from kb_a8f93b2ff6_asserted_statements limit
1;
-[ RECORD 1 ]--------------------------------------------
id        | 62516955
subject   | http://edina.ac.uk/ns/creation/12993043
predicate | http://www.w3.org/ns/prov#wasAssociatedWith
object    | http://edina.ac.uk/ns/agent/12887967
context   | http://edina.ac.uk/ns/rdfgraph
termcomb  | 0

Time: 0.531 ms
rdfgraph=>
```



More caveats!

- Make sure you are not entering literals in a URI field.
- Also make sure your URIs are valid (amazingly some DOIs fail when urlencoded)
- rdflib-sqlalchemy unoptimized for FTS
  - Your (BTree) indices will be YUGE.
  - Don't use large records (e.g. 10+ MB cnt:bytes)
- you need to **`drop index; insert; create index`**
  - **`pg_dump`** is your friend
  - Massive maintenance work memory (Postgres)

```
DROP INDEX public.kb_a8f93b2ff6_uri_index;
DROP INDEX public.kb_a8f93b2ff6_type_mkc_key;
DROP INDEX public.kb_a8f93b2ff6_quoted_spoc_key;
DROP INDEX public.kb_a8f93b2ff6_member_index;
DROP INDEX public.kb_a8f93b2ff6_literal_spoc_key;
DROP INDEX public.kb_a8f93b2ff6_klass_index;
DROP INDEX public.kb_a8f93b2ff6_c_index;
DROP INDEX public.kb_a8f93b2ff6_asserted_spoc_key;
DROP INDEX public."kb_a8f93b2ff6_T_termComb_index";
DROP INDEX public."kb_a8f93b2ff6_Q_termComb_index";
DROP INDEX public."kb_a8f93b2ff6_Q_s_index";
DROP INDEX public."kb_a8f93b2ff6_Q_p_index";
DROP INDEX public."kb_a8f93b2ff6_Q_o_index";
DROP INDEX public."kb_a8f93b2ff6_Q_c_index";
DROP INDEX public."kb_a8f93b2ff6_L_termComb_index";
DROP INDEX public."kb_a8f93b2ff6_L_s_index";
DROP INDEX public."kb_a8f93b2ff6_L_p_index";
DROP INDEX public."kb_a8f93b2ff6_L_c_index";
DROP INDEX public."kb_a8f93b2ff6_A_termComb_index";
DROP INDEX public."kb_a8f93b2ff6_A_s_index";
DROP INDEX public."kb_a8f93b2ff6_A_p_index";
DROP INDEX public."kb_a8f93b2ff6_A_o_index";
DROP INDEX public."kb_a8f93b2ff6_A_c_index";
ALTER TABLE ONLY public.kb_a8f93b2ff6_type_statements
DROP CONSTRAINT kb_a8f93b2ff6_type_statements_pkey;
ALTER TABLE ONLY public.kb_a8f93b2ff6_quoted_statements
DROP CONSTRAINT kb_a8f93b2ff6_quoted_statements_pkey;
ALTER TABLE ONLY public.kb_a8f93b2ff6_namespace_binds
DROP CONSTRAINT kb_a8f93b2ff6_namespace_binds_pkey;
ALTER TABLE ONLY public.kb_a8f93b2ff6_literal_statements
DROP CONSTRAINT kb_a8f93b2ff6_literal_statements_pkey;
ALTER TABLE ONLY public.kb_a8f93b2ff6_asserted_statements
DROP CONSTRAINT kb_a8f93b2ff6_asserted_statements_pkey;
```

Thank you =)

Twitter: @vyruss

EDINA Labs blog – http://labs.edina.ac.uk
Hack – http://github.com/vyruss/rdflib-sqlalchemy
Dataset – Stay tuned