なんて美しい国でしょう

Extending SPARQL for data analytic tasks

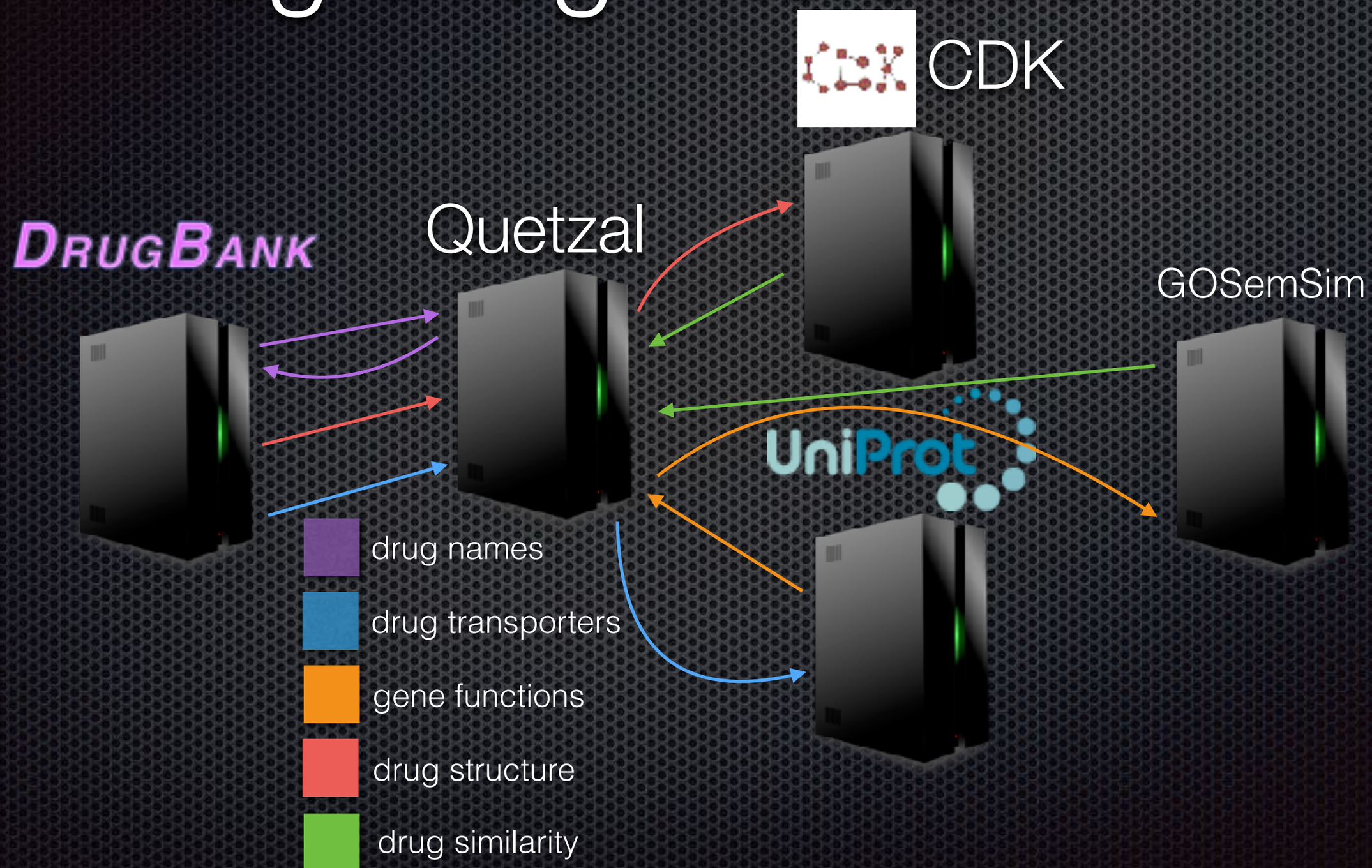
Julian Dolby
Achille Fokoue
Mariano Rodriguez Muro
Kavitha Srinivas
Wen Sun
IBM Research (Watson and China)

ISWC - Kobe (神戸) - October 2017

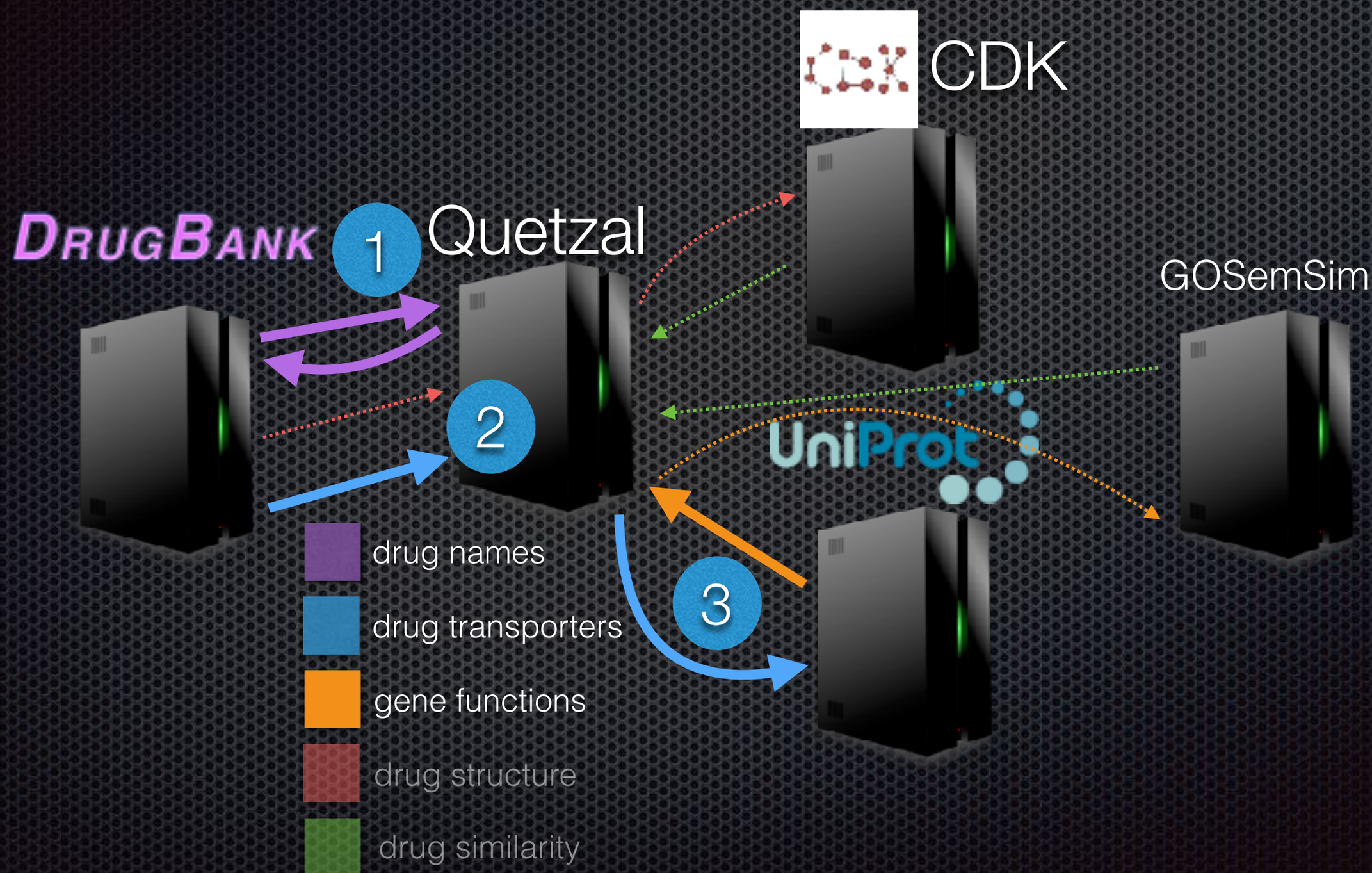
Data Everywhere

- Numerous structured data sources available
  - medical, e.g. Drugbank, Uniprot; general, e.g. DBpedia
  - much data in RDF, integrated and queried with SPARQL
- But data increasingly diverse
  - RDF, XML, JSON, CSV formats
  - accessible as dumps, query endpoints and APIs
- Powerful if integrated and queried effectively
  - reuse and extend existing declarative SPARQL

Data Integration: Drug-Drug Interaction



Focus of Talk



DrugBank UniProt

- Compendia of detailed medical information
  - DrugBank lists thousands of drugs
  - UniProt includes detailed data about genes
- Both regularly updated
- DrugBank available as XML; UniProt has a service API
- Free for non-commercial use
- http://www.drugbank.ca; http://www.uniprot.org

1

Obtain drug ids

- collect drugs to study
  - e.g. filter by status
- directly read drugbank XML
  - read latest data
  - avoid ETL overhead

```
<drugbank …>
  <drug …>
    <drugbank-id …>
    DB00158
    </drugbank-id>
…
<groups>
  <group>
  approved
  </group>
</groups>
…
```

1

Obtain drug ids

```
prefix db:http://www.drugbank.ca/system/downloads

function db:getDrugNames
service db:4.3/drugbank.xml.zip [] ->
“/drugbank/drug[./groups/group = ‘approved’]“ :
“./name”

select ?drug where {
 bind(db:getDrugNames() as (?drug))
}
```

service accessed by URI

XPath to define each solution binding

each element of binding defined by XPath

service called using bind syntax

2

Obtain drug transporters

- study transporters
  - delivering protein
  - multiple per drug
- batch drugbank read
  - not call per drug
  - reduce overhead

```
<drugbank ...>
 <drug ...>
  <drugbank-id...>DB00158...
  ...
  <transporters>
   <transporter>
    <external-identifiers>
     <external-identifier>
      <resource>UniProtKB...
      <identifier>Q9UNQ0...
    ...
```

2

Obtain drug transporters

```
prefix db:http://www.drugbank.ca/system/downloads
```

service needs one bound variable

```
function db:getDrugTransporters
table db:4.3/drugbank.xml.zip
[?drug -> ?drug ?transporter] -> “/drugbank/drug” :
"./name" “./transporters/…/identifier“
```

service binds two variables

single call for all bindings

```
select ?d ?t where {
 bind(db:getDrugNames() as (?d))
 bind(db:getDrugTransporters(?d) as (?d ?t))
}
```

?d is joined, linking ?d, ?t pairs

3

Obtain gene functions

- get transporter functions
  - use GO annotations
  - multiple per transporter
- UniProt API calls get XML
  - API is transporter id
  - computed URL

```
<uniprot …>
  <entry …>
    <accession>Q9UNQ0
    </accession>
    <dbReference
      type="GO"
      id="GO:0016324">
```

3

Obtain gene functions

```
prefix up:http://www.uniprot.org/uniprot
```

computed URI with SPARQL expression

```
function up:getGeneFunctions (?t -> ?gf ?t) service
CONCAT(“http://www.uniprot.org/uniprot/“, ?t, ".xml")
[ ] -> "/up:uniprot/up:entry/up:dbReference" ::
"./@id" "../accession[1]"
```

.. refers to parent in XML

```
select ?d ?t ?gf where {
 ...
 bind(up:getGeneFunctions(?t) as (?gf ?t))
}
```

?t is joined, linking ?gf, ?t pairs

3

Obtain gene functions

```
function drug:getGeneGroup ( ?ids -> ?gene ?id)
service CONCAT("http://www.ebi.ac.uk/...&id=", ?ids)
[ ] xs-> "/uniprot/entry/dbReference" ::
"./@id" "../accession[1]"
```

exploit “group by” in SPARQL

```
select (group_concat(?t; separator='+') AS ?ids) where {
 bind(db:getDrugTransporters(?d) as (?d ?t))
} group by (xsd:int(rowNumber() / 30))

BIND( drug:getGeneGroup( ?ids ) AS ( ?gf ?t2 ) )
```

Evaluation

- Demonstrated data integration on real data
  - 541 drugs in DrugBank, 113 transporters in UniProt
  - 941 drugs-transporter pairs
- Similarity computations expensive
  - simple to integrate as services
  - further extensions for parallelism desirable

Related Work

- Map relational data into RDF
  - R2RML, Direct Mapping translate SPARQL
  - We expand to more data, and improve federation
- Create RDF data
  - XSPARQL, CSV2RDF create RDF triples
  - Often looks like programming, e.g. XSPARQL

Conclusion

- Integrating diverse data key
  - diverse data formats proliferating
  - access as data, endpoints, APIs
- Simple extensions generalize SPARQL
  - maintain declarative query language
  - few constructs for practical integration, modularity
- Ideally standardize constructs like for SQL/XML