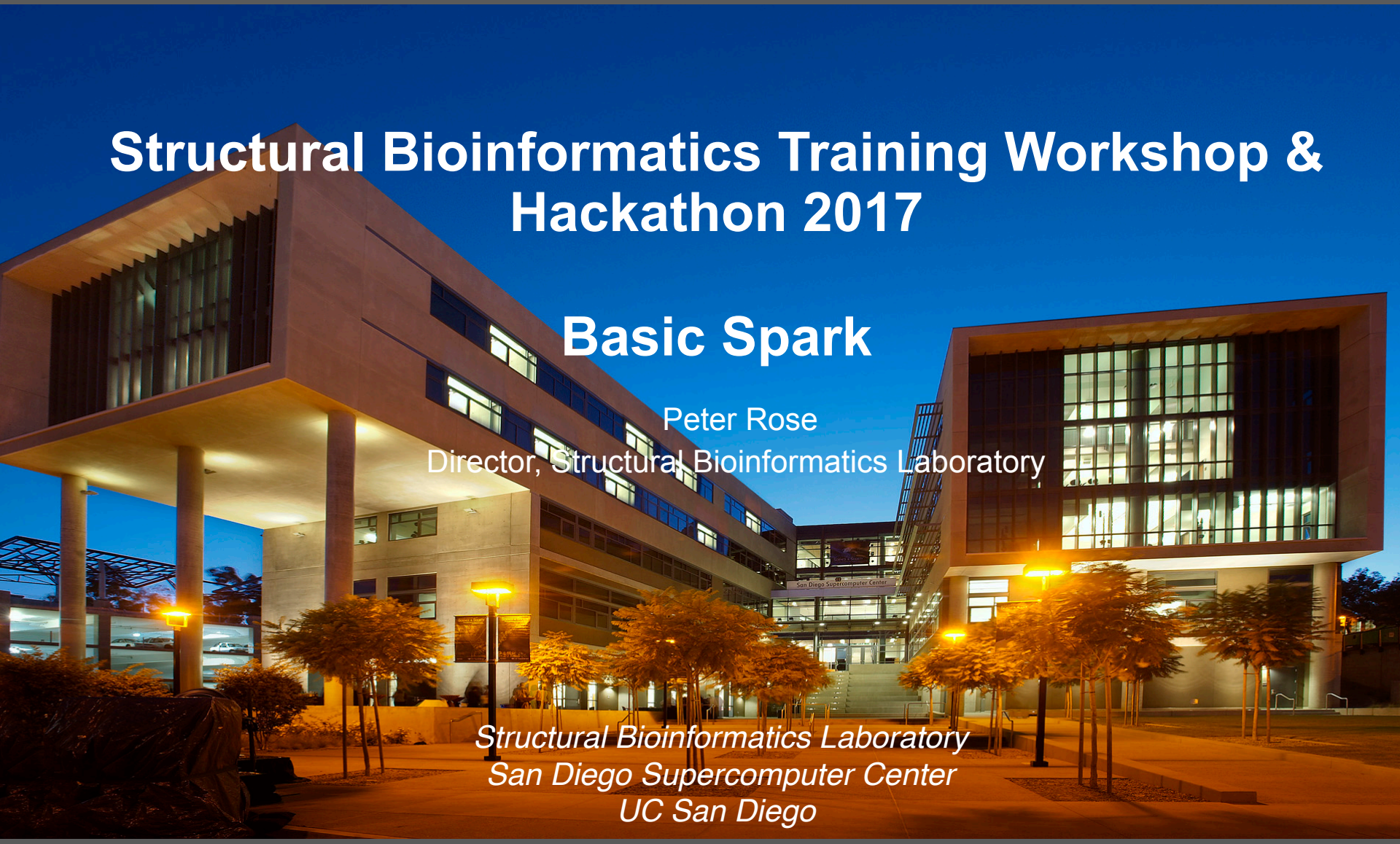
# Structural Bioinformatics Training Workshop & Hackathon 2017



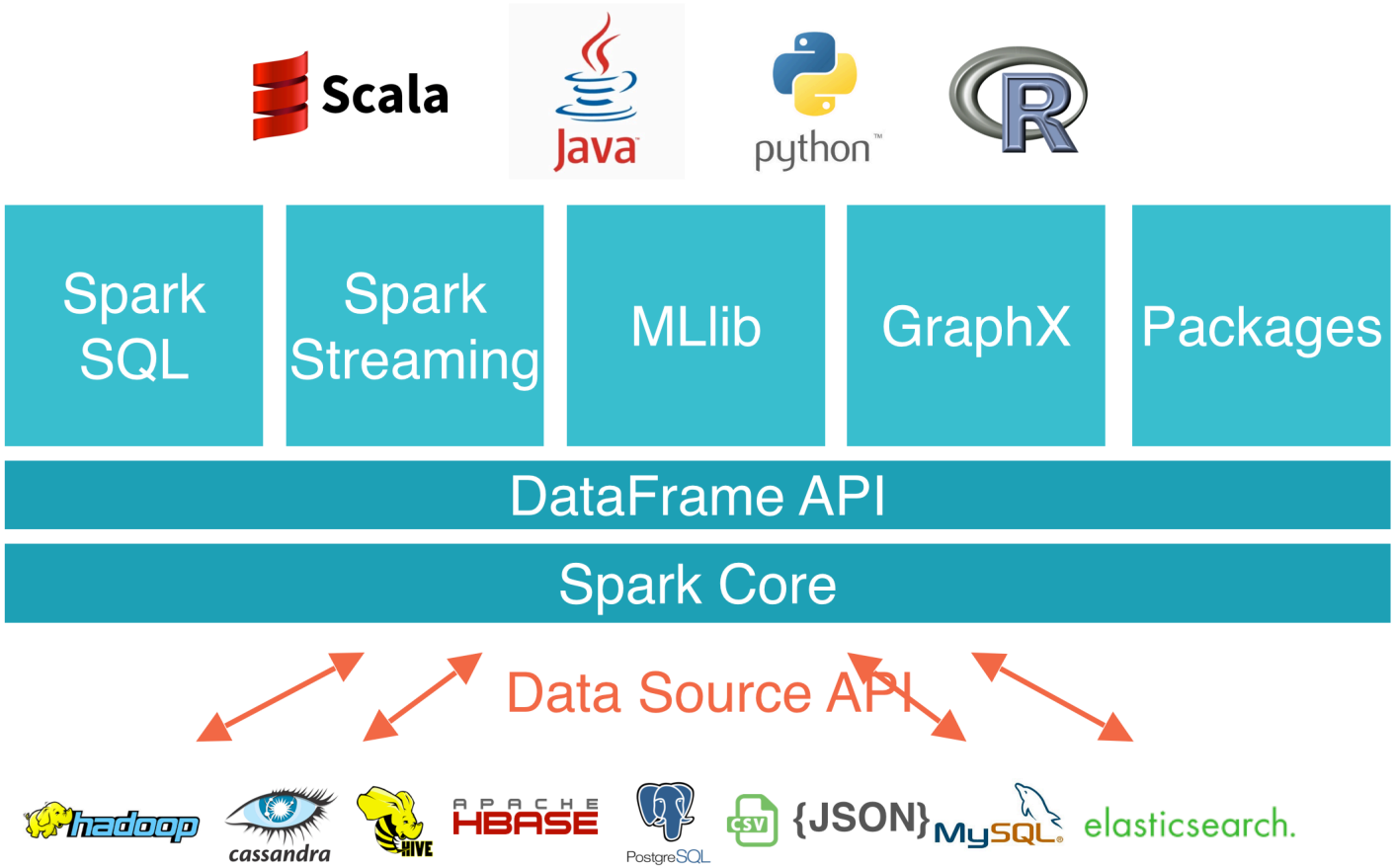
## Basic Spark

Peter Rose
Director, Structural Bioinformatics Laboratory

*Structural Bioinformatics Laboratory*
*San Diego Supercomputer Center*
*UC San Diego*

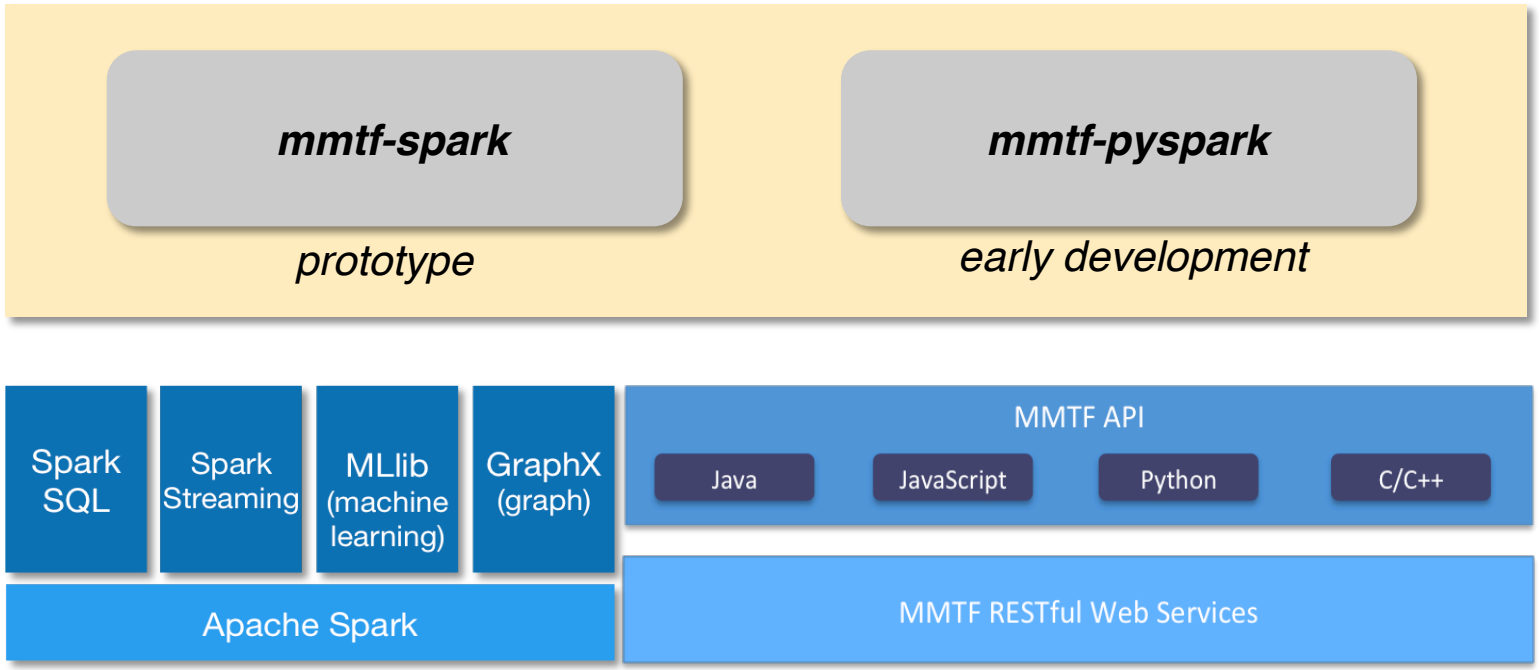
# Spark Ecosystem

**Apache Spark is a fast and general engine for large-scale data processing**



# mmtf-spark

***Framework for parallel distributed analysis and mining of 3D Macromolecular Structures in MMTF file format with Apache Spark***



# MMTF Data Sources

- **Single <pdbId>.mmtf.gz files downloaded using RESTful web services**
  - analyze a few PDB entries
- **Locally downloaded Hadoop Sequence file with MMTF records**
  - analyze many or all PDB entries
- **See: http://mmtf.rcsb.org/download.html**

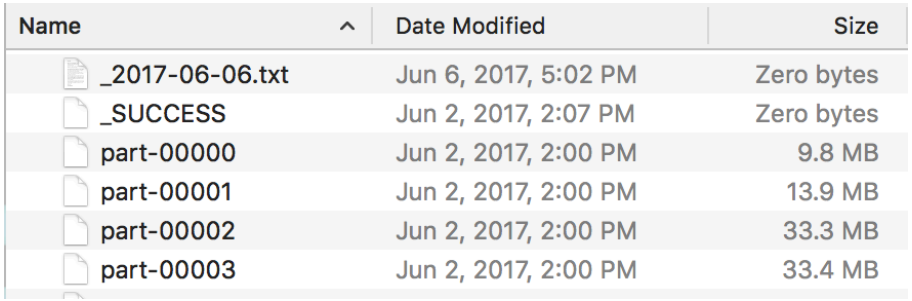
# Hadoop “Sequence” Files

- **A flat file of binary key/value pairs**
- **Used by Big Data Frameworks (Hadoop, Spark)**
  - File systems need few big files for efficient processing
- **Files are splittable**
  - Can be processed in parallel
- **Often consists of a directory of Sequence files**
- **See** https://wiki.apache.org/hadoop/SequenceFile

# MMTF-Hadoop Sequence Files

- **Two representations**
  - **full**
    - all atoms
    - full data precision
  - **reduced**
    - polymers
      - polypeptides: C-alpha
      - polynucleotides: P
      - 1st model only (e.g., NMR)
      - no alternative locations
      - except polysaccharides
        - » all atom
    - non-polymers
      - all atoms
    - water
      - excluded
    - Reduced precision (0.1): coordinates, temperature-factor, occupancy

- **Example: full directory structure**



| Name | Date Modified | Size |
|---|---|---|
| _2017-06-06.txt | Jun 6, 2017, 5:02 PM | Zero bytes |
| _SUCCESS | Jun 2, 2017, 2:07 PM | Zero bytes |
| part-00000 | Jun 2, 2017, 2:00 PM | 9.8 MB |
| part-00001 | Jun 2, 2017, 2:00 PM | 13.9 MB |
| part-00002 | Jun 2, 2017, 2:00 PM | 33.3 MB |
| part-00003 | Jun 2, 2017, 2:00 PM | 33.4 MB |

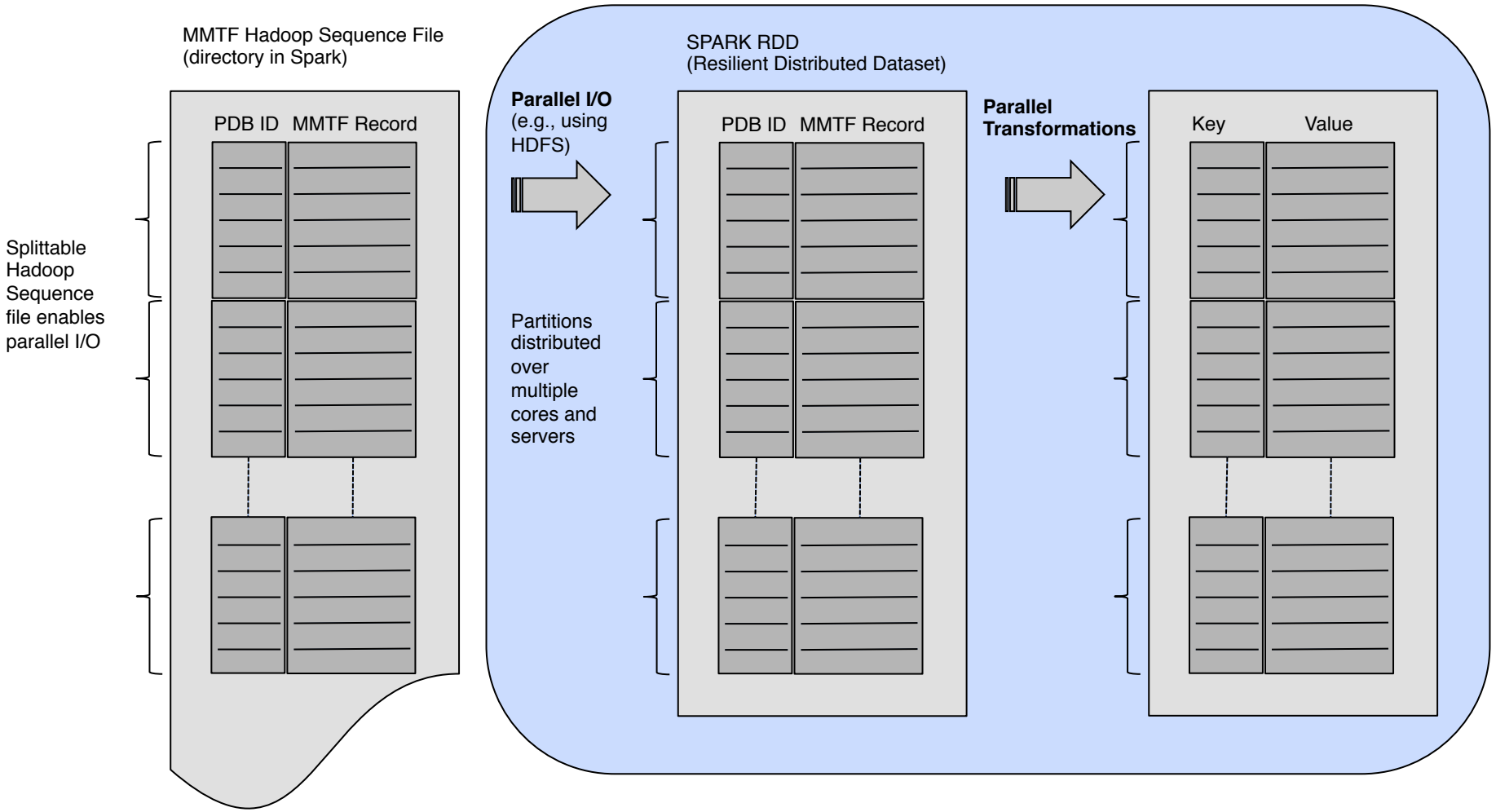
- **Timestamp file (release date)**
  - _yyyy-mm-dd.txt
- **Updated every Wed. ~00:00 UTC**
- **Multiple sequence files**
  - part-00000 …
- **Download**
  - http://mmtf.rcsb.org/download.html

# Sequence Files for Workshop

- **10% random sample of PDB (June 20, 2017)**
  - reduced_10: 13,040 structures, 124 MB
  - full_10: 13,221 structures, 860 MB

- **Complete PDB (June 20, 2017)**
  - reduced: 131,205 structures, 1.25 GB
  - full:131,205 structures, 8.56 GB

# MMTF-Spark Data Pipeline



# Basic MMTF-Spark Operations

- Reading MMTF Files & Hadoop Sequence files
- Filtering PDB structures
- FlatMapping PDB structures
- Filtering with RCSB PDB web services
- Using Map/Reduce with Lambda expressions
- Writing custom Hadoop Sequence files

# Downloading mmtf.gz files

```
 // spark setup
JavaSparkContext sc = …

// download a list of PDB entries from http://mmtf.rcsb.org
List<String> pdbIds = Arrays.asList("1AQ1","1B38","1B39","1BUH");

JavaPairRDD<String, StructureDataInterface> pdb = MmtfReader
                                    .downloadMmtfFiles(pdbIds, sc);
```

JavaPairRDD is a resilient distributed dataset of key/value pairs

key : String – structureId, e.g., pdbId (4HHB)
value: StructureDataInterface - structure representation in uncompressed form

# Reading from a Sequence File

```
 // get path to full Sequence file
String path = System.getProperty("MMTF_FULL");

 // spark setup
JavaSparkContext sc = …

// read a list of PDB entries
List<String> pdbIds = Arrays.asList("1AQ1","1B38","1B39","1BUH");

JavaPairRDD<String, StructureDataInterface> pdb = MmtfReader
                                    .readSequenceFile(path, pdbIds, sc);

// or read all PDB entries
JavaPairRDD<String, StructureDataInterface> pdb = MmtfReader
                                    .readSequenceFile(path, sc);
```

# Problem 0

- **Reading Files**
  - Navigate to project: 3-basic-spark in Eclipse
  - Find and and open Problem00.java (src/main/java)
  - Look at // TODO for the problem description
  - Insert your code after the // TODO and run it

# Filtering by Quality Metrics

```
// read all PDB entries
JavaPairRDD<String, StructureDataInterface> pdb = MmtfReader
                                    .readSequenceFile(path, sc);

// keep PDB entries with a resolution in the inclusive range [0, 2]
pdb = pdb.filter(new Resolution(0.0, 2.0));


// or more concisely with method chaining
JavaPairRDD<String, StructureDataInterface> pdb = MmtfReader
                                    .readSequenceFile(path, sc)
                                    .filter(new Resolution(0.0, 2.0));
```

Related filters: Rfree and Rwork

Note, these filters will eliminate any entries that do not have these metrics, e.g., NMR structures.

See http://pdb101.rcsb.org/learn/guide-to-understanding-pdb-data/introduction

# Filtering by Polymer Chain Types

```
// keep PDB entries that contain at least one L-protein chain
JavaPairRDD<String, StructureDataInterface> pdb = MmtfReader
                                .readSequenceFile(path, sc)
                                .filter(new ContainsLProteinChain());
```

```
// keep PDB entries that contain exclusively L-protein chains
boolean exclusive = true;

JavaPairRDD<String, StructureDataInterface> pdb = MmtfReader
                                .readSequenceFile(path, sc)
                                .filter(new ContainsLProteinChain(exclusive));
```

**Related filters:**
ContainsDProteinChain
ContainsDnaChain
ContainsRnaChain
ContainsDSaccharideChain
ContainsPolymerChainType
PolymerComposition

# Filtering by Heterogeneous Polymer Chain Types

```
// keep PDB that contain DNA, RNA, or both (DNA/RNA hybrid)
JavaPairRDD<String, StructureDataInterface> pdb = MmtfReader
        .readSequenceFile(path, sc)
        .filter(new ContainsPolymerChainType("DNA LINKING","RNA LINKING"));
```

**Monomer types** (most frequent types in bold)
**PEPTIDE_LINKING** (achiral, e.g., GLY)

D_PEPTIDE_LINKING
D_PEPTIDE_COOH_CARBOXY_TERMINUS
D_PEPTIDE_NH3_AMINO_TERMINUS

**L_PEPTIDE_LINKING**
L_PEPTIDE_COOH_CARBOXY_TERMINUS
L_PEPTIDE_NH3_AMINO_TERMINUS

**DNA_LINKING**
DNA_OH_3_PRIME_TERMINUS
DNA_OH_5_PRIME_TERMINUS

NON_POLYMER
SACCHARIDE (achiral)

D_SACCHARIDE
D_SACCHARIDE_14_and_14_LINKING
D_SACCHARIDE_14_and_16_LINKING

L_SACCHARIDE
L_SACCHARIDE_14_AND_14_LINKING
L_SACCHARIDE_14_AND_16_LINKING

**RNA_LINKING**
RNA_OH_3_PRIME_TERMINUS
RNA_OH_5_PRIME_TERMINUS

See http://mmcif.wwpdb.org/dictionaries/mmcif_mdb.dic/Items/_chem_comp.type.html

# Problem 1

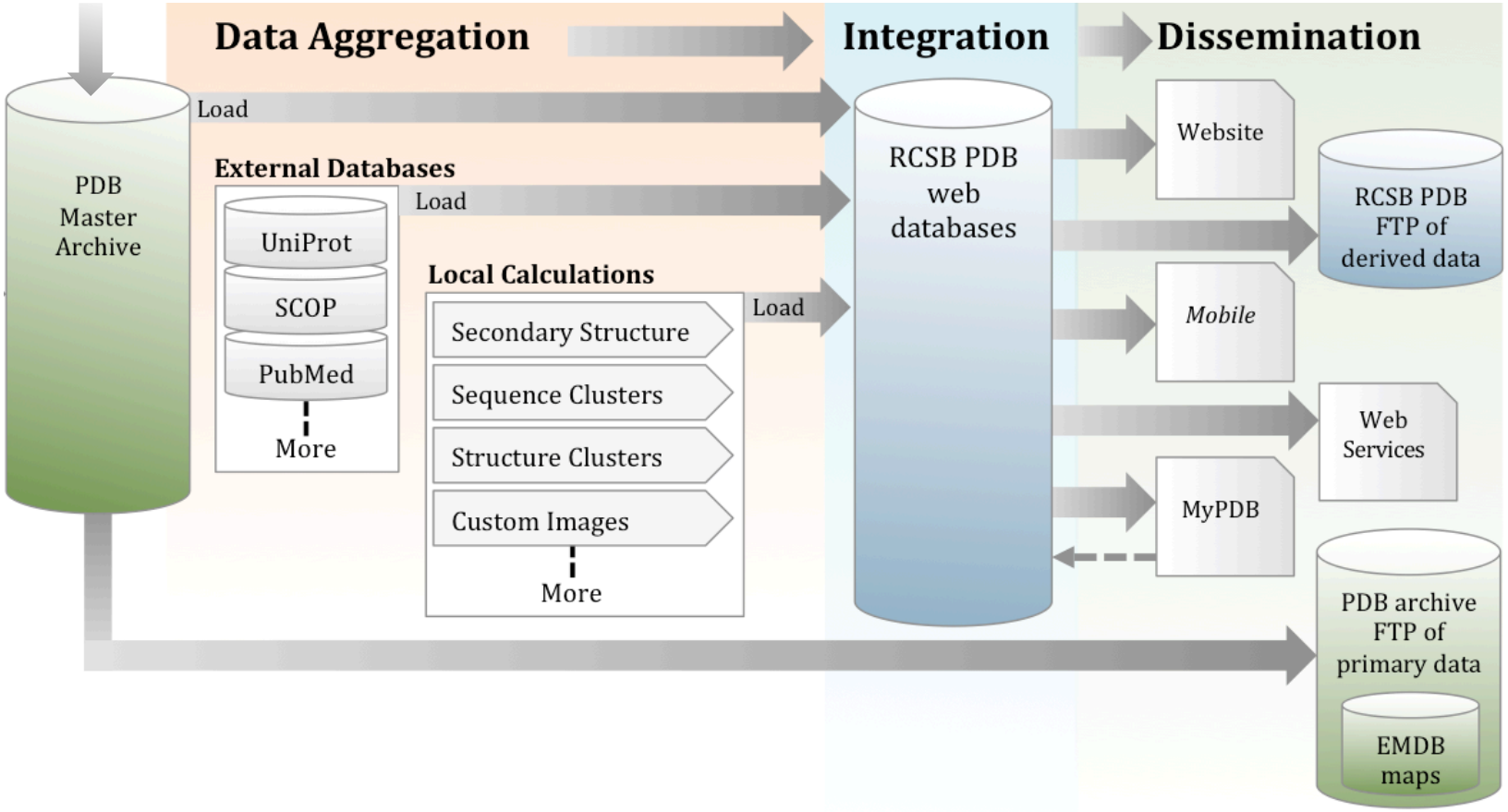
- **Filtering**
  - Navigate to project: 3-basic-spark in Eclipse
  - Find and and open Problem01.java (src/main/java)
  - Look at the // TODO for the problem description
  - Insert your code after the // TODO and run it

# Filtering with RCSB PDB Web Services

- **MMTF file content focused on structural data**
- **Additional data and annotations from RCSB PDB**
  - Ids and keywords, structure annotation, structure features, sequence features, chemical components, biological info, methods info, deposition info, publications
  - http://www.rcsb.org/pdb/staticHelp.do?p=help/advancedsearch/index.html
- **Any advanced query from the RCSB PDB website can be run through RESTful web services**
  - http://www.rcsb.org/pdb/search/advSearch.do?search=new

# RCSB PDB Data Pipeline

***Data In***

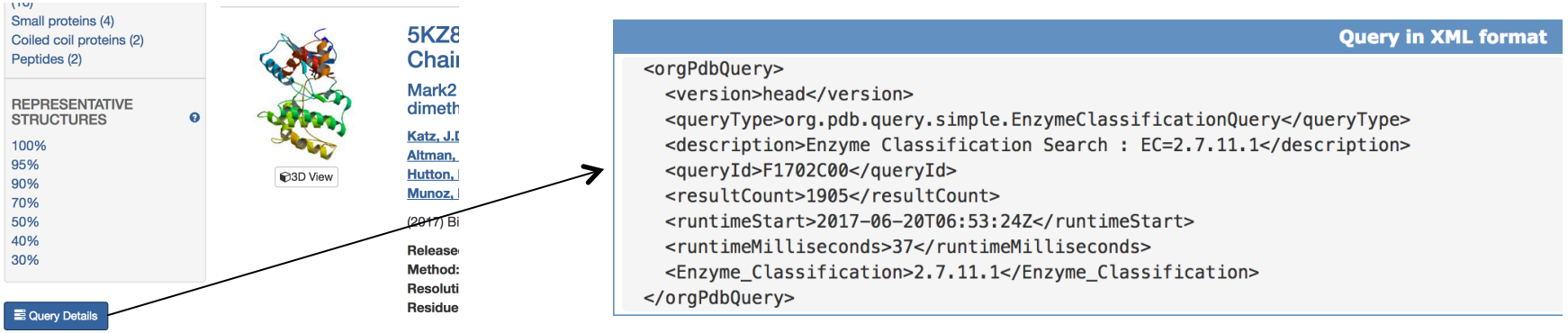


# Demo

## RCSB PDB Advanced Query

# Filtering by Advanced Search XML

- Run any advanced query at http://www.rcsb.org
- Go to results page and click Query Details



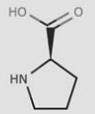
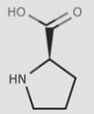
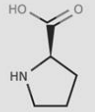
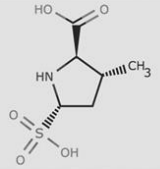
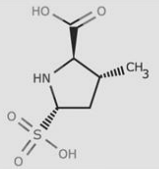
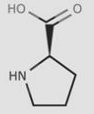
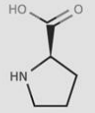
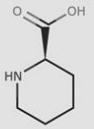

```
String query =
"<orgPdbQuery>" +
    "<queryType>org.pdb.query.simple.EnzymeClassificationQuery</queryType>”+
    "<Enzyme_Classification>2.7.11.1</Enzyme_Classification>" +
"</orgPdbQuery>”;

pdb = pdb.filter(new AdvancedQuery(query));
```

# Filtering by SMILES

```
// keep structures that contain a chemical component
// with this substructure
pdb = pdb.filter(new
    ChemicalStructureQuery("OC(=O)CCCC[C@@H]1SC[C@@H]2N
    C(=O)N[C@H]12”,
    ChemicalStructureQuery.SUBSTRUCTURE, 0));
```

```
// keep structures that contain a chemical component
// that is >= 70% similar to the query structure
int similarity = 70;
pdb = pdb.filter(new
    ChemicalStructureQuery("OC(=O)CCCC[C@@H]1SC[C@@H]2N
    C(=O)N[C@H]12”,
    ChemicalStructureQuery.SIMILAR, similarity));
```

**Query Types:**
EXACT
SUBSTRUCTURE
SUPERSTRUCTURE
SIMILAR

| Search type | Query | Result |
|---|---|---|
| Exact |  |  |
| Substructure |  |  |
| Superstructure |  |  |
| Similar |  |  |

# Problem 3

- **Filtering with RCSB PDB Web Services**
  - Open Problem03.java
  - Insert the code after the // TODO and run your code
  - Pick one of the result PDB Ids and check the http://www.rcsb.org website to see if the result matches the expected result.

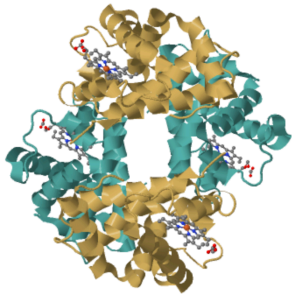
# FlatMapping

- Maps one data record to zero or more data records
- Flatmap a PDB entry (quaternary structure) to its polymer chains (tertiary structure)

```
// Extract polymer chains out of PDB entries
JavaPairRDD<String, StructureDataInterface> pdb = MmtfReader
                                .readSequenceFile(path, sc)
                                .flatMapToPair(new StructureToPolymerChains());
```

# FlatMapping Effect on Keys

```
List<String> pdbId = Arrays.asList("4HHB");
JavaPairRDD<String, StructureDataInterface> pdb = MmtfReader
                                    .downloadMmtfFiles(pdbId, sc);
```



```
pdb.keys().foreach(k -> System.out.println(k));
```

*=> 4HHB (4HHB is a tetramer or dimer of dimers)*

```
pdb.flatMapToPair(new StructureToPolymerChains())
   .keys().foreach(k -> System.out.println(k));
```

*=> 4HHB.A 4HHB.B 4HHB.C 4HHB.D (4HHB contains 4 protein chains)*
*Chain name is appended to PDB Id*

```
// exclude chains with identical sequences
boolean excludeDuplicates = true;
pdb.flatMapToPair(new StructureToPolymerChains(false, excludeDuplicates))
   .keys().foreach(k -> System.out.println(k));
```

*=> 4HHB.A 4HHB.B (4HHB contains two unique protein chains: hemoglobin alpha and beta)*

# Map and Reduce

*Count number of atoms in the PDB*

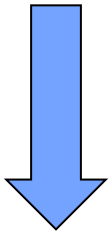
JavaPairRDD<String, StructureDataInterface>

| PDB ID | Structure |
|---|---|
| 1ABC | … |
| 2BCD | … |
| 3CDE | … |
| 4DEF | … |
| 5EFG | … |

*map (getNumAtoms())*

JavaRDD<Integer>

| NumAtoms |
|---|
| 1,000 |
| 2,000 |
| 3,000 |
| 4,000 |
| 5,000 |



*reduce (sum)*

*15,000*

# Map and Reduce Using Lambda Expressions

```
// count total number of atoms in PDB
long numAtoms = pdb
                    .map(t -> t._2.getNumAtoms())
                    .reduce((a,b) -> a+b);
```

# Problem 3

- Open Problem02.java
- Insert the code after the // TODO and run your code
- Look up one of the result PDB Ids on the http://www.rcsb.org website and check if the query result matches your expectation.

# Writing Hadoop Sequence Files

```
// read all PDB entries
JavaPairRDD<String, StructureDataInterface> pdb = MmtfReader
                                    .readSequenceFile(path, sc);

// keep PDB entries with a resolution in the inclusive range [0, 2]
pdb = pdb.filter(new Resolution(0.0, 2.0));

MmtfWriter.writeSequenceFile(path + ”_highResolution", sc, chains);
```

Use MmtfWriter.writeMmtfFiles(path, sc, structure)
to write (a small subset) to individual mmtf.gz files

# Summary

- **mmtf-spark: Framework for parallel distributed mining of the PDB with Apache Spark**
- **Hadoop Sequence file is an efficient container format to process large number of structures**
- **PDB structures represented as key/value pairs**
- **Spark transformations**
  - filter, keys, map, flatMap
- **Spark actions**
  - count, foreach, reduce, collect

# Resources

- **MMTF Website**
  - http://mmtf.rcsb.org
- **GitHub Repository**
  - https://github.com/sbl-sdsc/mmtf-spark
- **MMTF File Format**
  - Bradley AR, et al. (2017) MMTF—An efficient file format for the transmission, visualization, and analysis of macromolecular structures. PLOS Computational Biology 13(6): e1005575. https://doi.org/10.1371/journal.pcbi.1005575
  - Valasatava Y, et al. (2017) Towards an efficient compression of 3D coordinates of macromolecular structures. PLOS ONE 12(3): e0174846. https://doi.org/10.1371/journal.pone.0174846