Status update on data model developments

Colin Bernet Benedikt Hegner

FCC SW Meeting
25.9.2014

Developments of last week

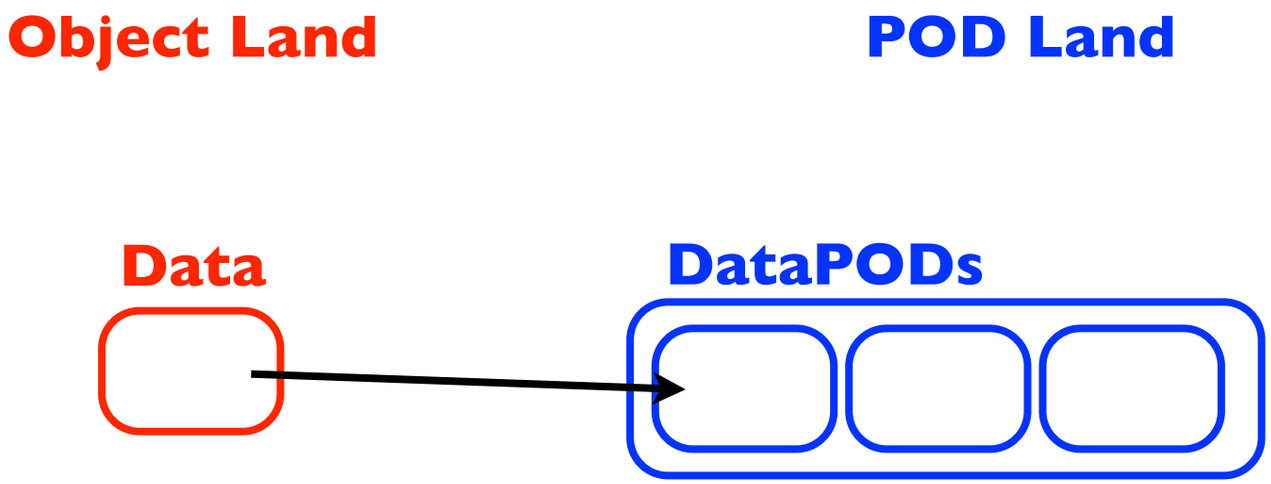
- **Colin made himself familiar with the concepts and the code**
  - Doubles the manpower! :-)
- **Extension of the simple I/O prototype for event 'looping'**
- **New iteration on data model definition syntax**
  - Now supporting annotations and comments
- **Enabled compilation on MacOS X**
- **Somehow more realistic data model examples put in**
- **Code cleanup**

Reminder of the main concepts

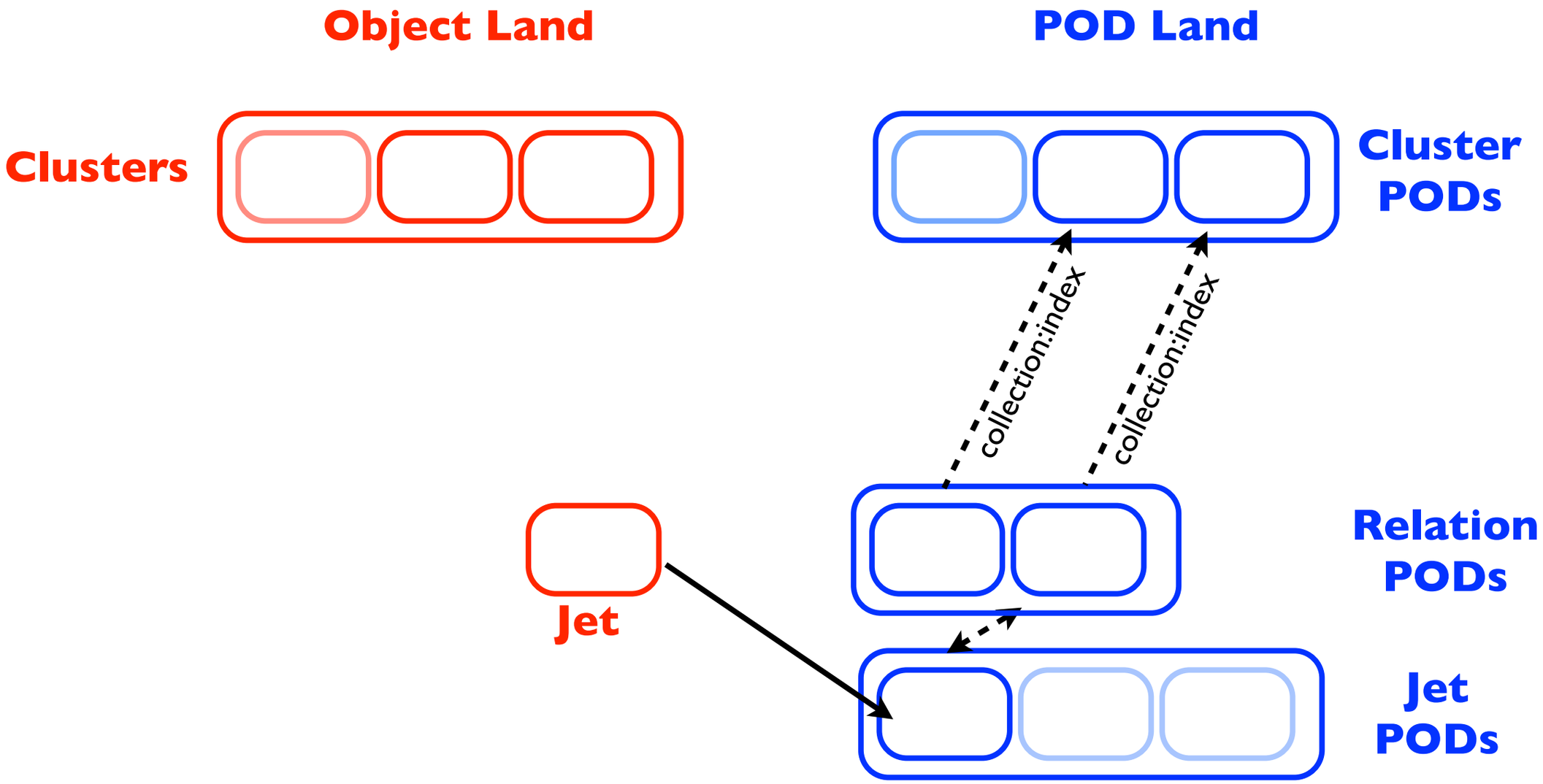
- **ROOT as first choice for I/O**
  - Keep transient to persistent layer as thin as possible
- **Simple memory model**
  - Employ **simple structs (PODs)** instead of fat objects
  - Allow for *Structs of Arrays (vectorization friendly!)*
- **Simple class hierarchies**
  - Wherever possible use **concrete types**
  - Favor composition over inheritance
- **Employ code generation**
  - Quick turn-around for improvement on the back-end
  - Easy creation of new types for the user

Separation of Concerns

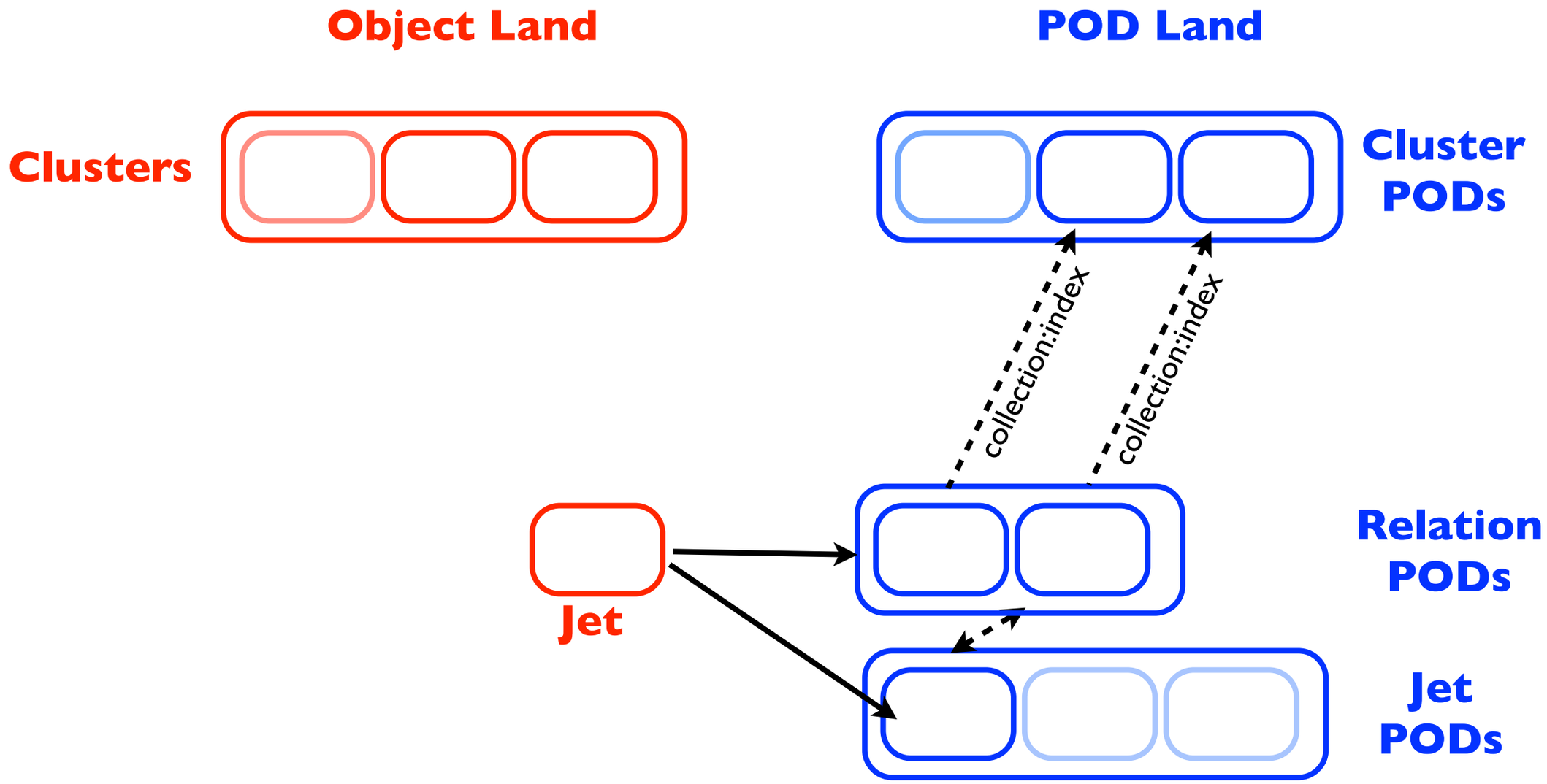
- **Use dumb objects (PODs) for every performance critical part**
- **Provide smart layer on top of the PODs**
  - Dealing with ownership
  - Allow referencing between objects
  - Deal with non-trivial I/O operations



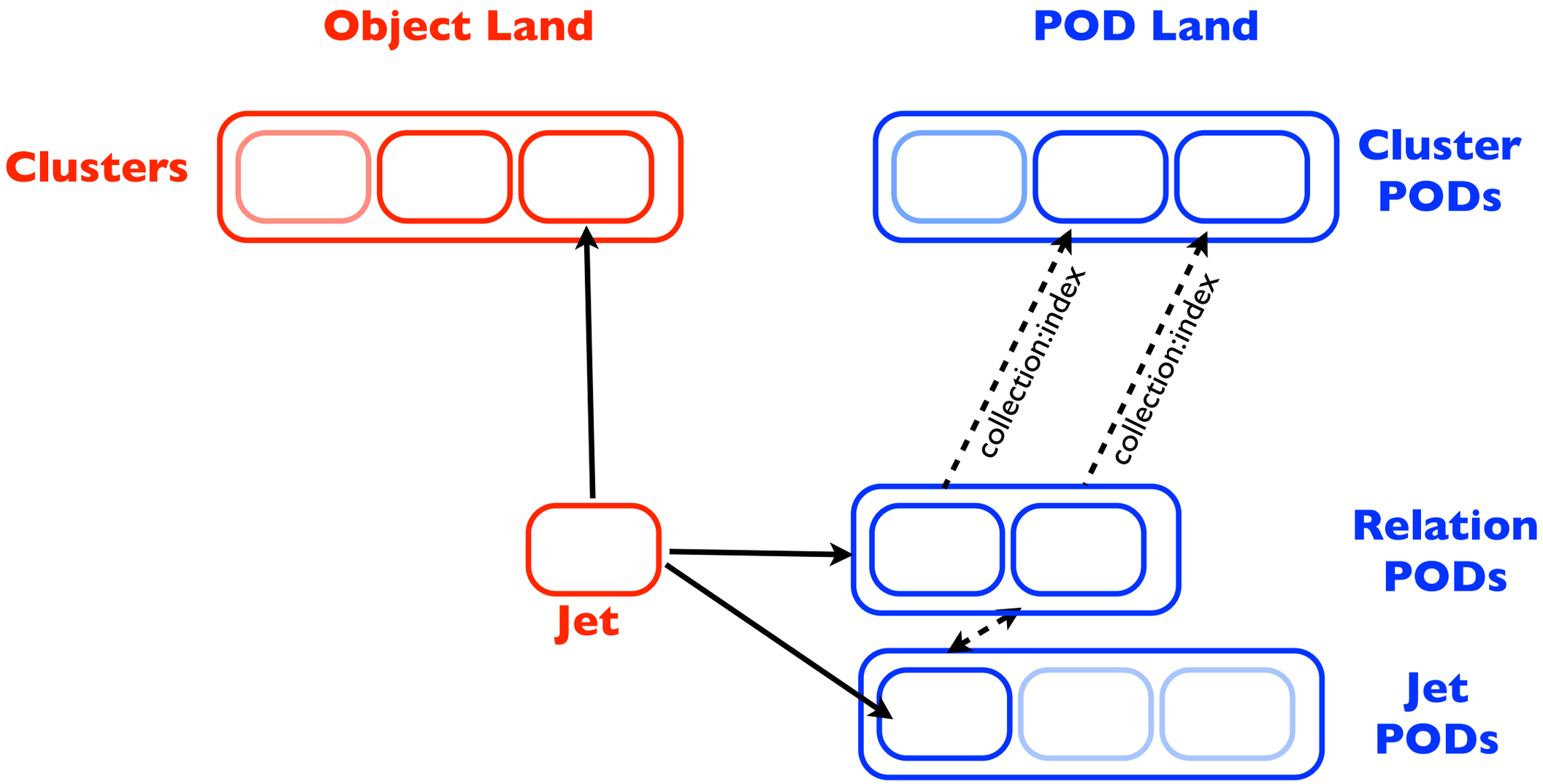
Storing I-M Relations



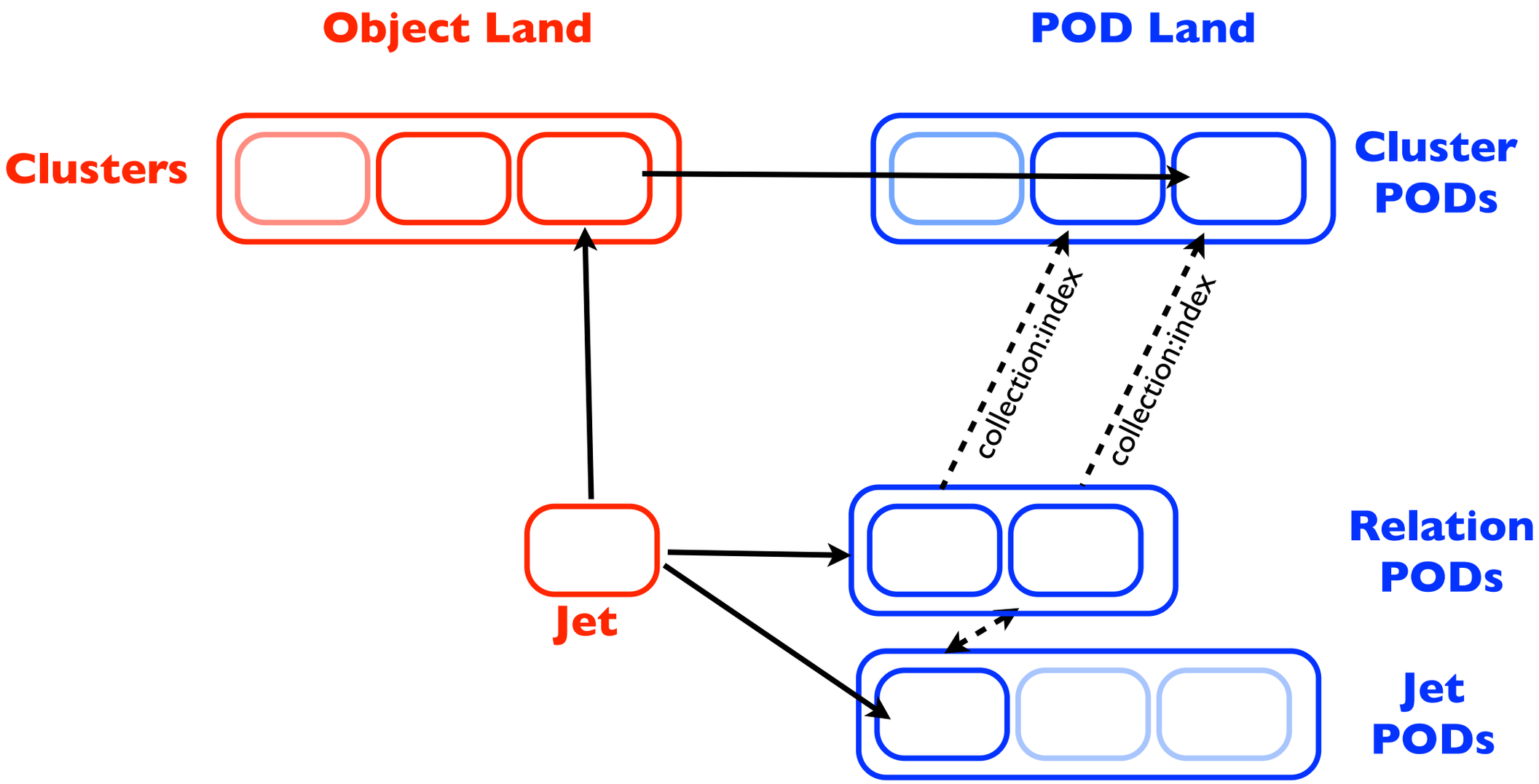
Storing N-M Relations



Storing N-M Relations



Storing N-M Relations



Data model definition

- **Data model classes are defined in text files, not in code**
- **Two categories of types**
  - Components
  - Real data types
- **Types can contain**
  - simple data types (double, int, ...)
  - fixed-size arrays of simple data types
  - Components
  - (vector of) references to other data types

not the final word!

```
components :
  ...

datatypes :
  MyClass :
    description : "..."
    members :
      - name : x
        type : double
        description : ...
      - name : ...
```

- **These definitions are then passed to a code generator...**
  ... which then produces the PODs and the smart layer for all "datatypes"

Code generation

Data definition

```
DummyData:
  ...
```

Data Object

```
class DummyData {
  friend DummyDataCollection;
public:
  DummyData(){};
  const int& Number() const;
  void setNumber(int value);
  bool isAvailable() const;
  void prepareForWrite(const albers::Registry*);
  void prepareAfterRead(albers::Registry*);
private:
  DummyData(int index, int containerID,
            std::vector<DummyDataPOD>* container);
  int m_index;
  int m_containerID;
  mutable std::vector<DummyDataPOD>* m_container;
  albers::Registry* m_registry;
};
```

Data POD

```
class DummyDataCollection;

class DummyDataPOD {
  friend DummyDataCollection;

public:
  const int& Number() const { return m_Number;};
  void setNumber(int& value){ m_Number = value;};

private:

  int m_Number;

};
```

All preliminary code examples

Object Collection

```
class DummyDataCollection : public albers::CollectionBase {
..
  DummyDataCollection();
...
  DummyData& create();
  DummyData& get(int index) const;
  void prepareForWrite(const albers::Registry* registry);
  void prepareAfterRead(albers::Registry* registry);
  void setPODsAddress(const void* address);
  std::vector<DummyDataPOD>* _getBuffer(){ return m_data;};
...
private:
  int m_collectionID;
  std::vector<DummyDataPOD>* m_data;
...
};
```