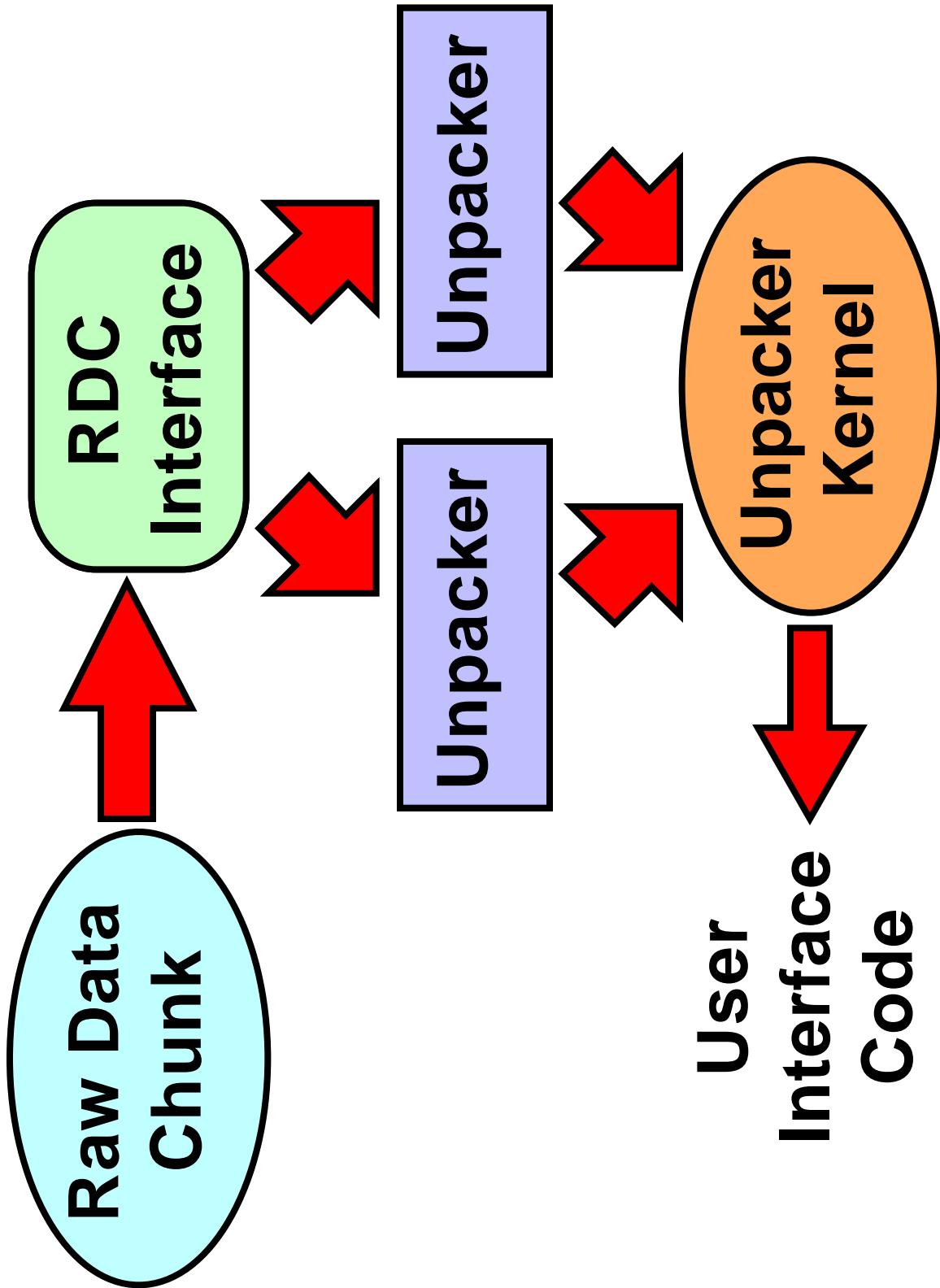


L1/L2 Unpacker

- Package to perform the reverse of the l1l2collector
- Provides several features
 - Simplified access to the Raw Data Chunk
 - A very simple Unpacker class interface
 - Easy to use user interface to get at the unpacked data
- Goal is to provide access to Raw Data Chunk for L1/L2 analyse packages
- General enough for use outside L1/L2 if desired

L1/L2 Unpacker Overview



Raw Data Chunk Interface

- Singleton class "RDCInterface" provides simple access to Raw Data Chunk
 - So far only one accessor provided
 - DataBroadcast getL3Data(crate,module)
 - Returns a data broadcast object which contains the raw data
 - Same object you sent to the Raw Data Chunk in the simulation
 - Just use the retrieve method to fill an I/Ogen object

RDCInterface Example

```
unsigned int crate=3; // Choose the crate number
unsigned int mod=2; // Choose the module number

// Get a pointer to the singleton RDCInterface
RDCInterface *rdc=RDCInterface::instance();

// Call the getL3Data method to get the data
DataBroadcast bcast=rdc->getL3Data(crate,mod);

// Now unpack the data into your own I/Ogen class
bcast.retrieve(myIOgenClass);
```

- New interfaces easy to add...
 - `getL3Data(l2base::BlockType)`
 - `vector<DataBoardcast> getL3Data(crate)`
 - etc...

Unpacker Class

- All unpackers inherit from the Unpacker base class
- Requires two methods + constructor
 - Constructor
 - Used to setup class with, for example, a create and module number
 - void types(vector<fwk::Action::ID>&)
 - Add class types to given vector to declare to the kernel which classes you create
- void unpack(void)
 - Called ONLY ONCE per RDC to unpack data

Unpacker Example

```
class MyUnpacker : public Unpacker {
    ...method prototypes...
private:
    MyIOgenData _data;
    unsigned int _crate; // Create containing the data
    int _mod;           // Module containing the data
};

// Constructor
MyUnpacker::MyUnpacker(unsigned int crate,int mod) {
    _crate=crate; _mod=mod;
}

// Method to declare class types
void MyUnpacker::types(std::vector<fwk::Action::Id> &ids) {
    ids.push_back(CLASS_ID(_data));
}

// Method to do the unpacking
void MyUnpacker::unpack(void) {
    RDCCInterface *rdc=RDCCInterface::instance();
    DataBroadcast data=rdc->getL3Data(_crate,_mod);
    data.retrieve(_data); // Unpack the raw data
    addObject(&_data); // Declare object to kernel
}
```

Unpacker Kernel

- Central management class
- Calls all unpackers needed to provide requested user data
- Will (soon!) have a separate framework package
- Sets up RDCTInterface to use the current Raw Data Chunk
- Increments internal event counter to ensure that the unpackers are called again
- Unpackers are declared using:
`declareUnpacker(new MyUnpacker([args]))`

User Interface

- User interface designed to be as simple as possible:
 - All you need to know is the type of object you want returned!
 - `const vector<const T*> *extract< T>()`
 - Returns a vector of all objects of type T that the unpacker knows how to unpack from the Raw Data Chunk
 - For example:
`extract<CTTPTrackData>()`
will return all `CTTPTrackData` objects

Generic Module Unpackers

- Template class to unpack a module into a single I/Ogen class already exists
 - ModulePacker<IOgenClassData>
 - Constructor takes crate and module as arguments
- Generic L2 unpacker will also be written (not done yet)
- Will allow direct access to MBT channels as well as the objects they contain e.g. CTT tracks, Cal Trig Towers etc.
- Avoids need to use L3Output -> MBT path