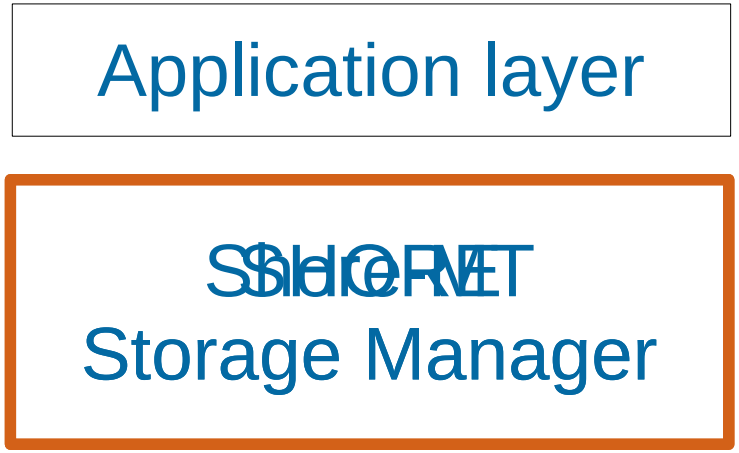
Metadata Front-end for Shore-MT Storage Manager

"*A dynamic relational application layer for Shore-MT with metadata management capability*"

- EPFL I&C Semester Project, Sept 2012 – Jan 2013
- *Student*: Bao Duy TRAN (210215)

- *Supervisor*: Prof Anastasia Ailamaki
- *Advisers*: Pınar Tözün & Danica Porobic

History



Easing Shore-MT's usage

- Shore-MT: low-level API
- No restrictions:
  - Interaction patterns
  - Data models

  → Flexibility

- For relational model:
  - Hard-coding
  - Repetition
- Prototyping by scripting?

Shore-MT metadata **front-end** with scripting support

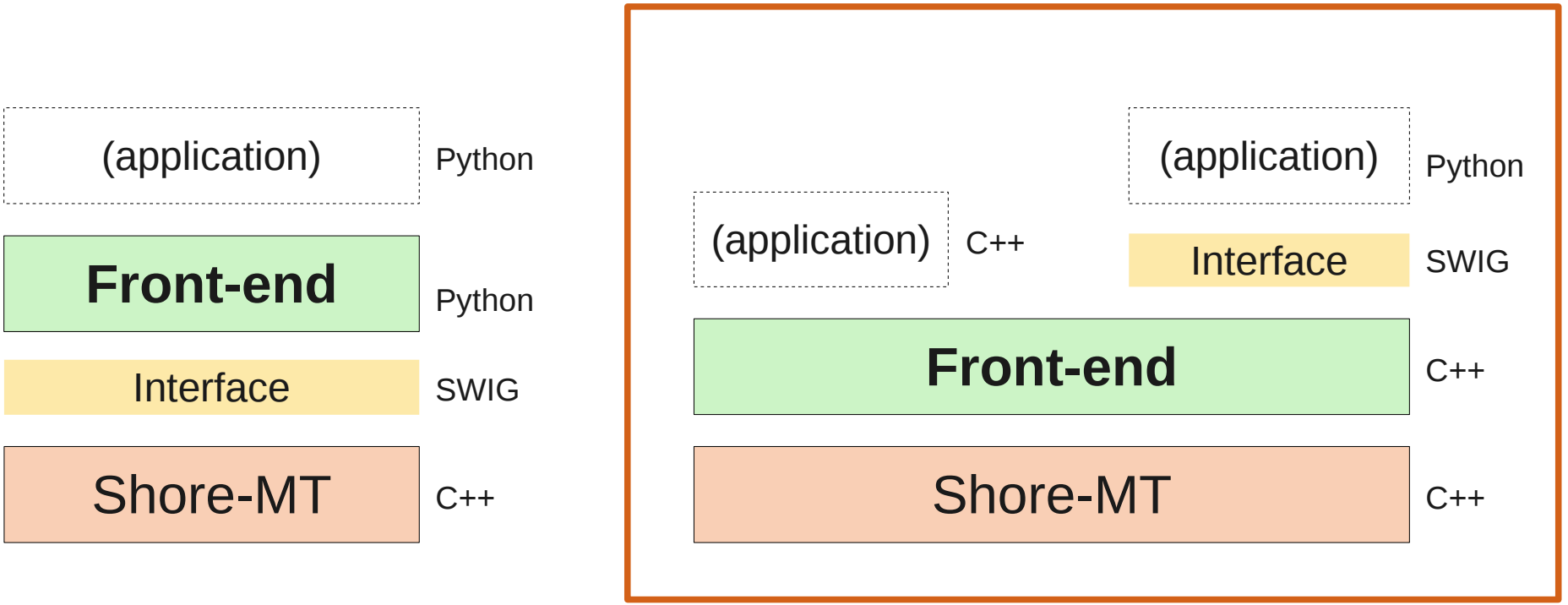
Project objectives

- Familiarise with Shore-MT
- Investigate SWIG (a cross-language interfacing library)
- **Design & develop Shore-MT front-end**
- Implement interactive console applications (demos)
- Ensure transferability, maintainability and reusability (Document all features & design decisions)

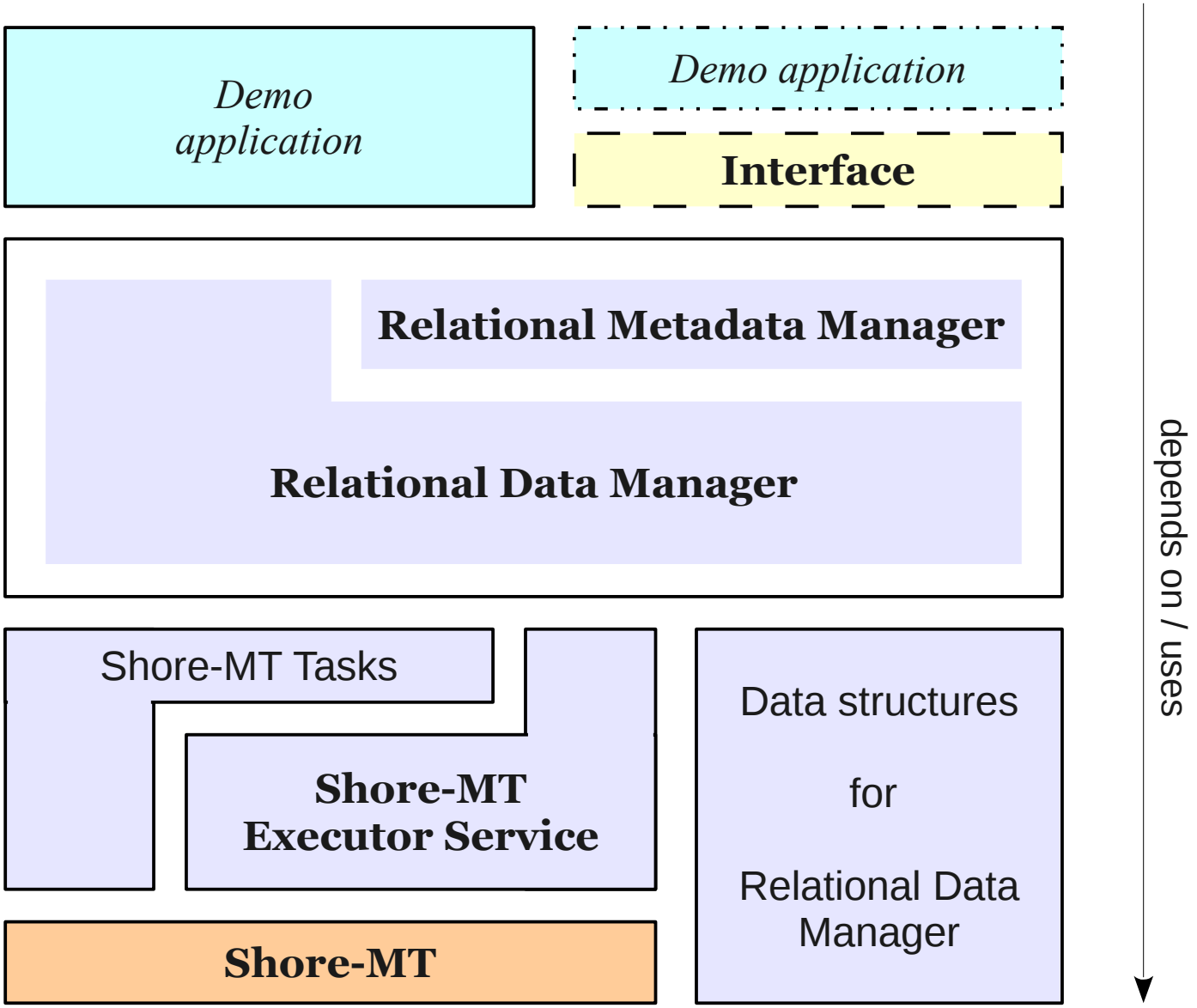
Project scope

- **Python** as target scripting environment
- Limited set of **operations**
- No **parsers**!
- Sole focus: **Functionality**

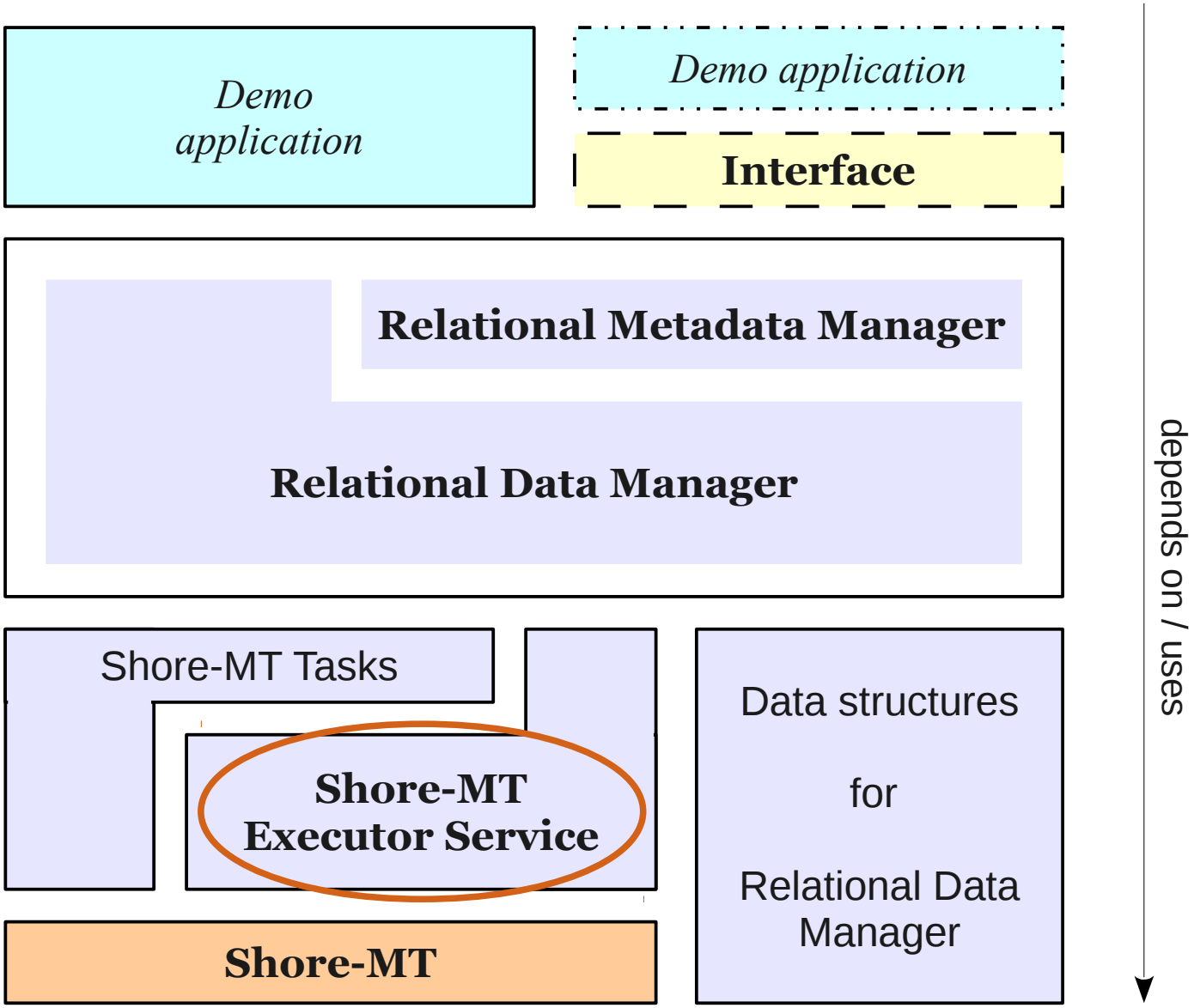
Preliminary



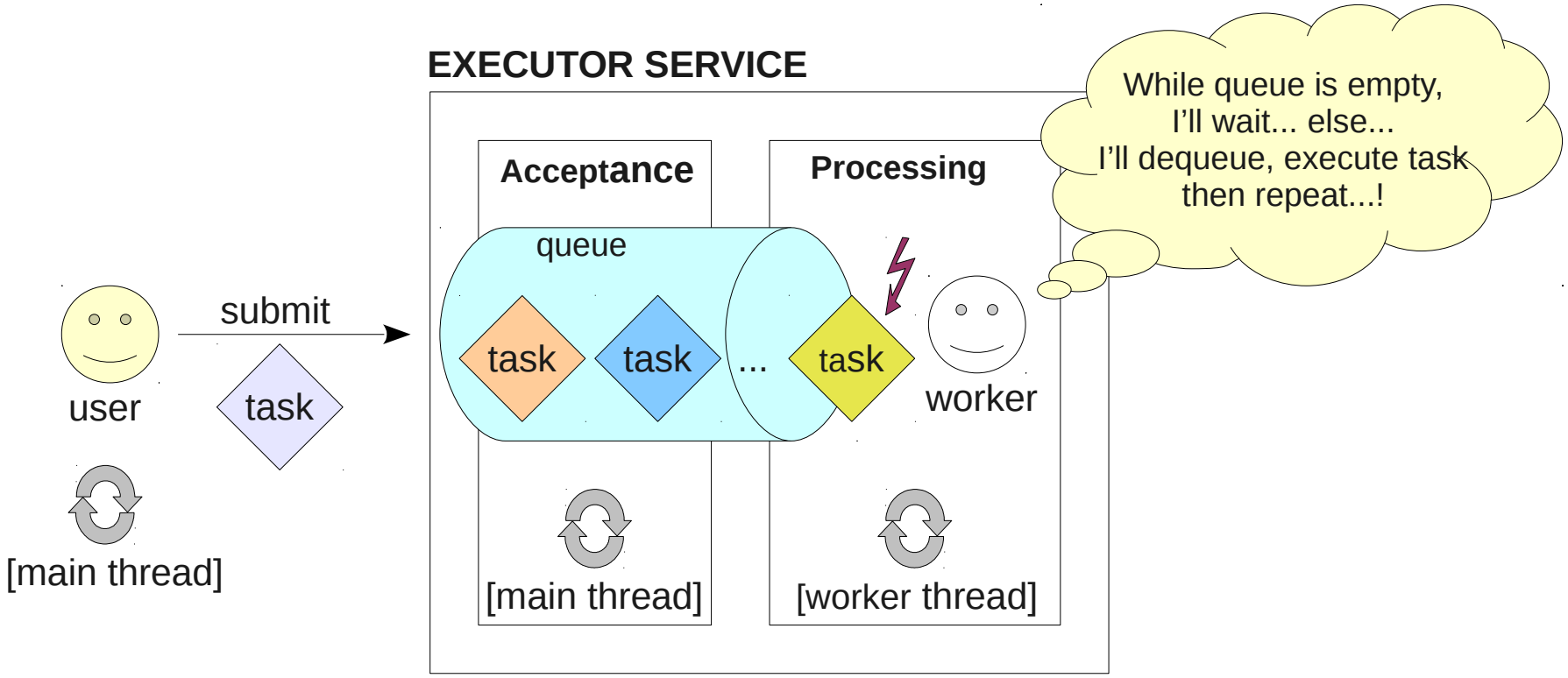
Architecture



Executor Service: A generic utility

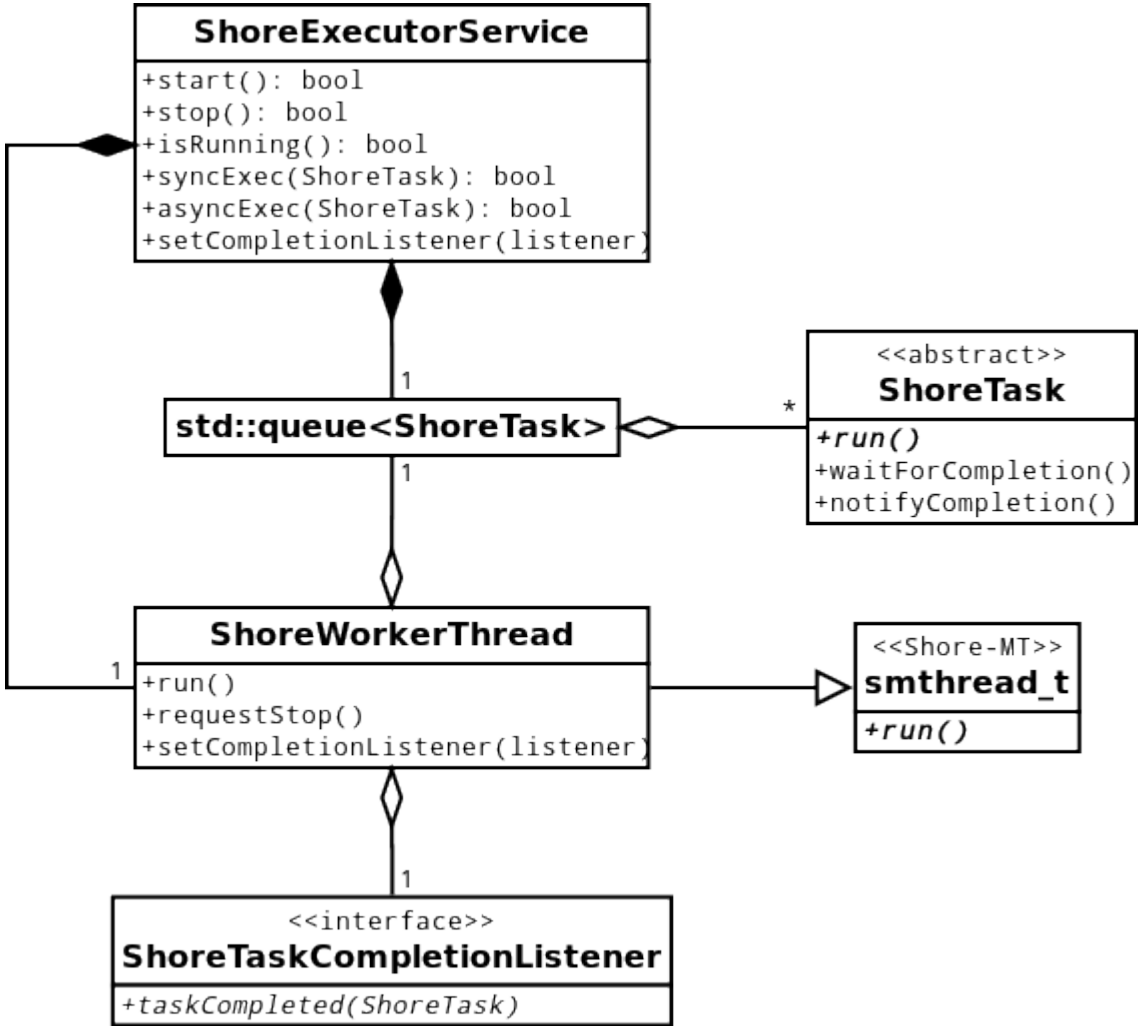


Functional design



- Synchronous / Asynchronous execution modes
- Proper signalling & synchronisation schemes

Object design



Usage example

```
void myShoreMtOperation() {
    ss_m::doSomething();
    ss_m::doSomethingElse();
}
```

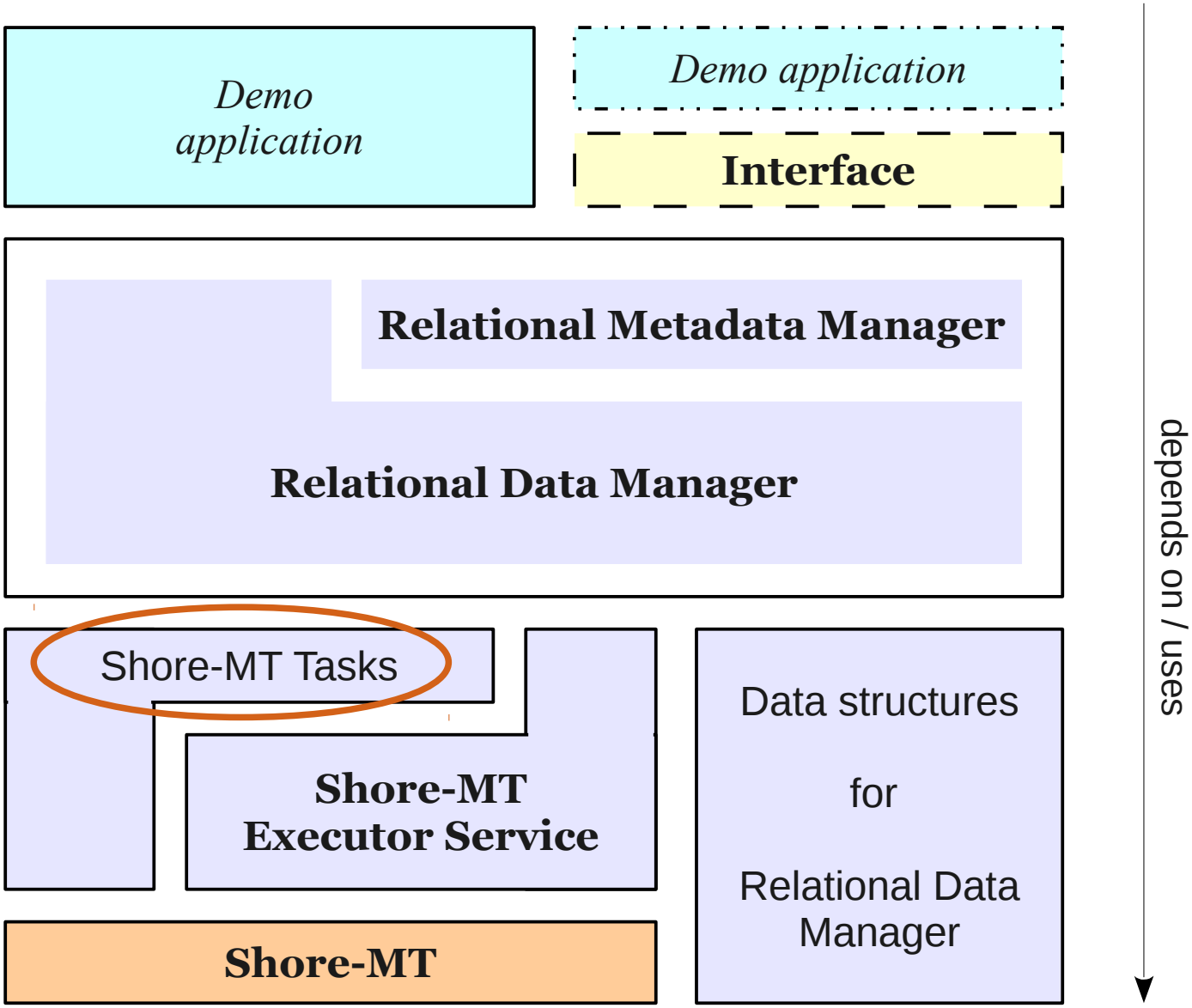


```
void myShoreMtOperation(ShoreExecutorService* myExecSvc) {
    DoTwoThingsTask myTask;
    myExecSvc->syncExec(&myTask);
}
```

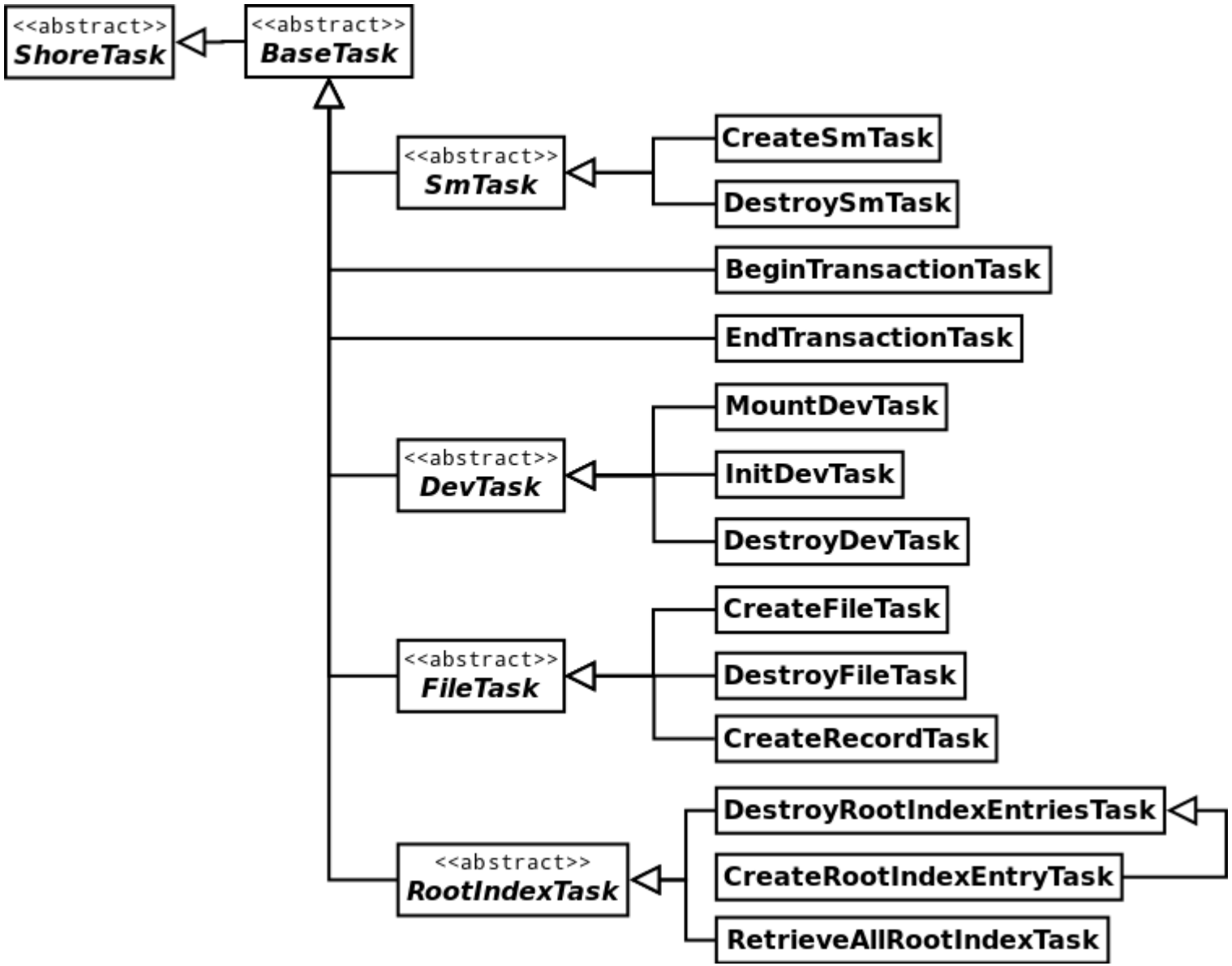
```
class DoTwoThingsTask : public ShoreTask { … }
```

```
void DoTwoThingsTask::run() {
    ss_m::doSomething();
    ss_m::doSomethingElse();
}
```

Front-end tasks



Front-end tasks



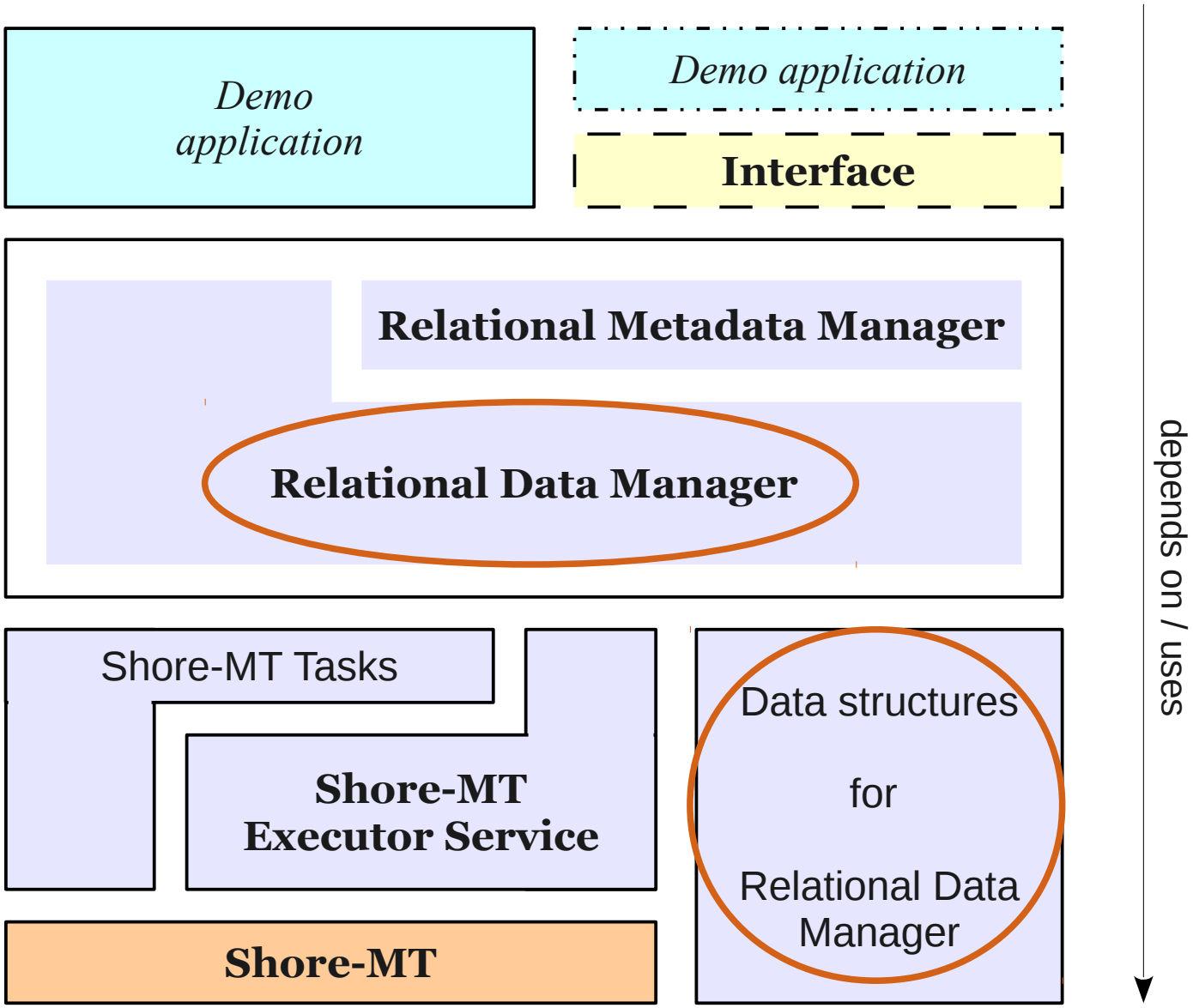
Threading strategy

- Single Executor Service
  - Instantiated upon top-level initialisation
  - Passed to subordinates as necessary

  → Single dedicated thread (extensible)

- Synchronous execution mode

Relational Data Manager



Functional design

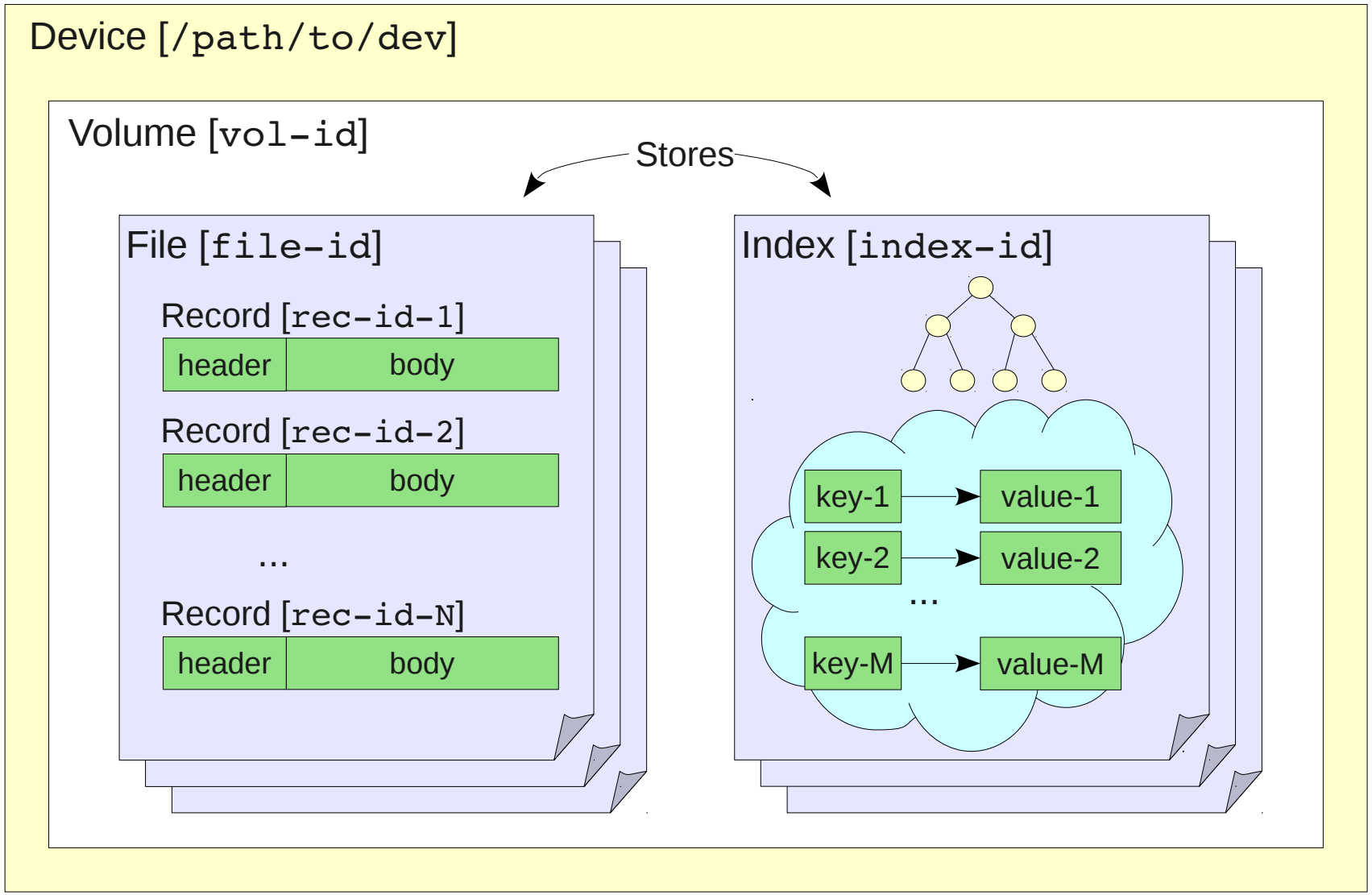
- Databases → tables → tuples → fields
- Schema → field descriptions (name, type, size, nullability)

- **Operations**:
  - Initialisation, shut-down
  - DB creation/deletion/selection
  - Table creation/deletion
  - Tuple insertion/retrieval

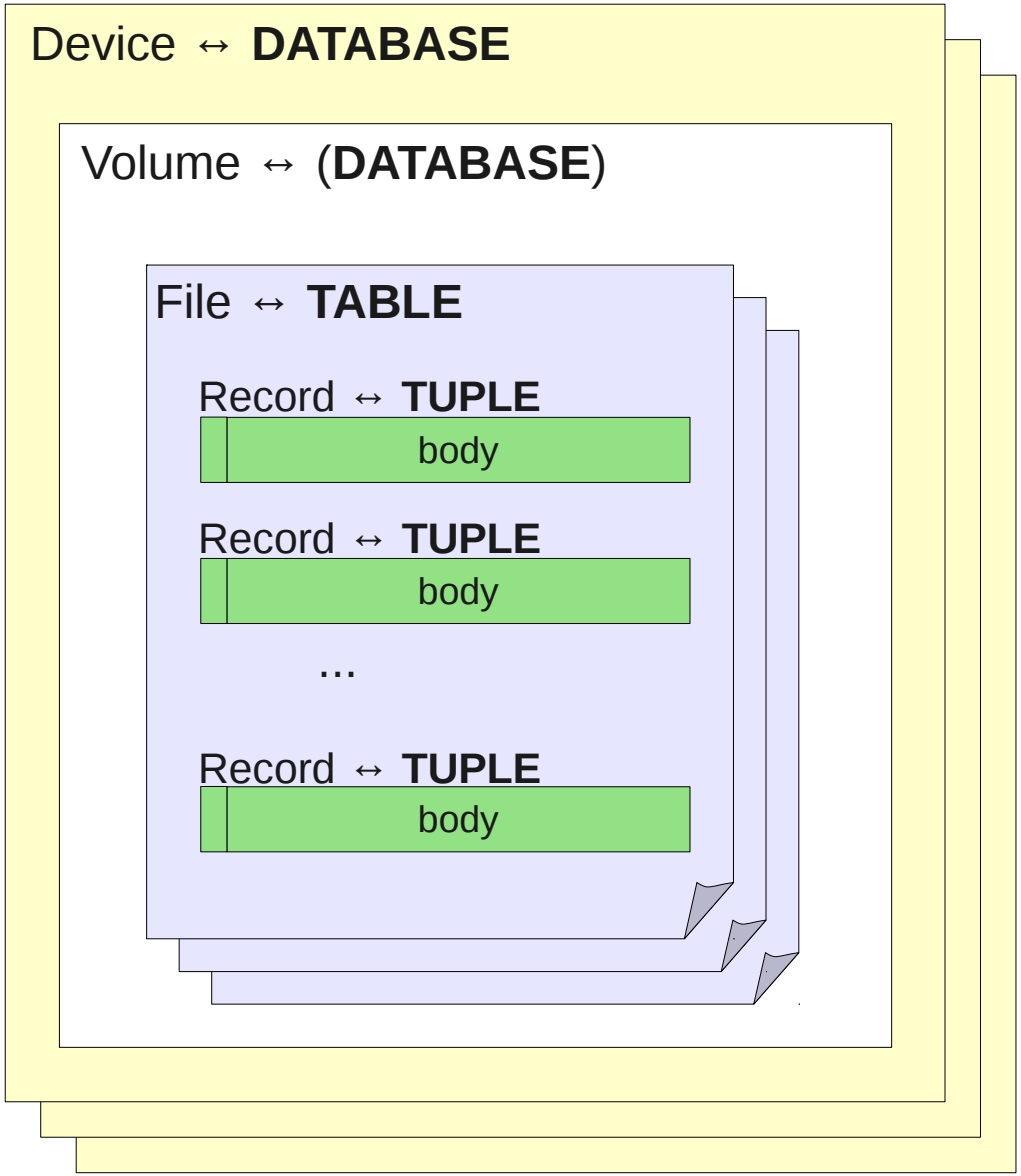
- At most one database is selected (in use)

- No persistence of relational metadata

RECAP: Shore-MT storage structures



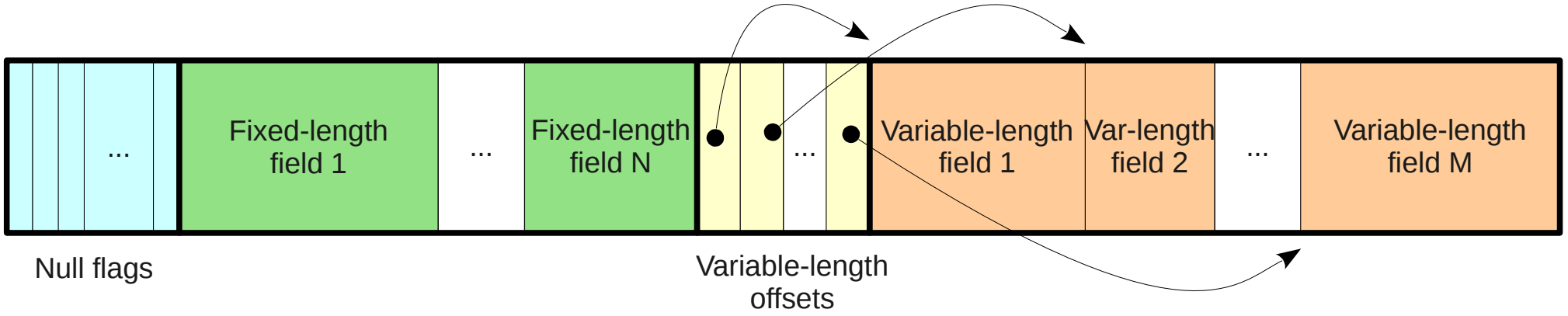
Relational model → Storage structures



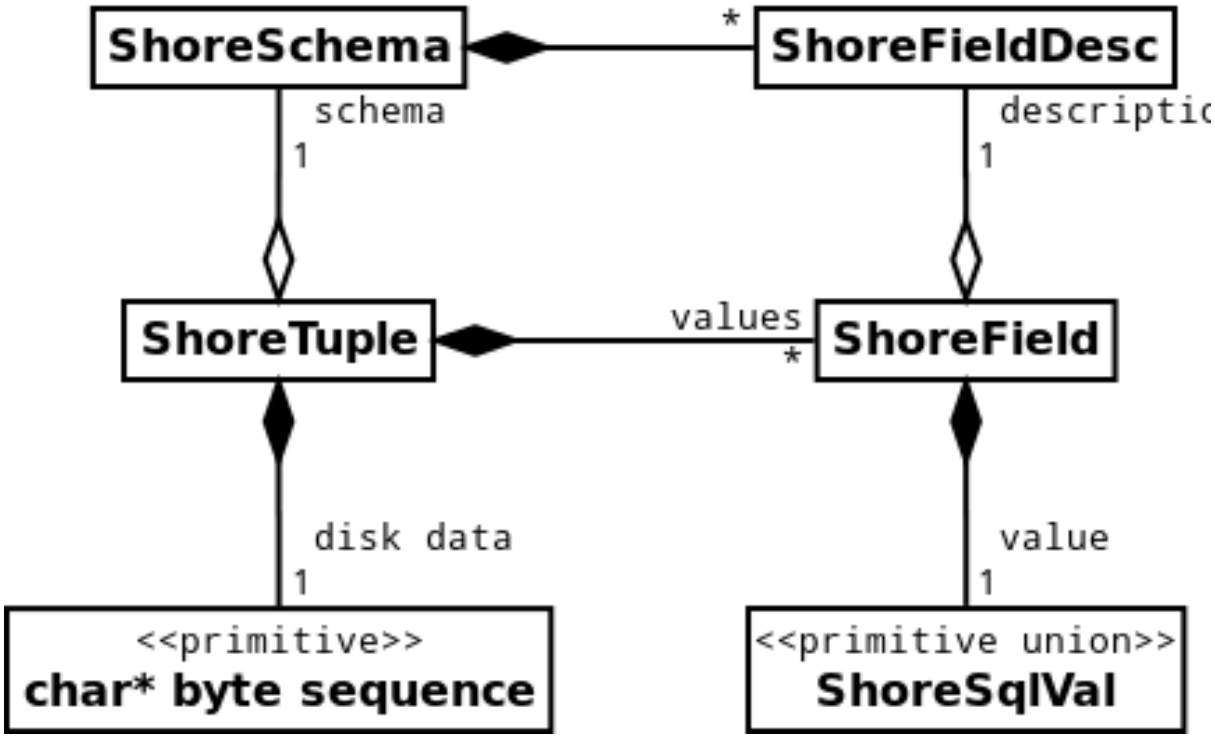
SQL → system data types

SQL type	C++ type	Description	Single unit?	Fixed length?
BIT	bool	Boolean (bit)	√	√
SMALLINT	short	Short integer	√	√
INT	int	Integer	√	√
LONG	long	Long integer	√	√
FLOAT	float	Floating-point	√	√
DOUBLE	double	Double-precision floating-point	√	√
CHAR	char[]	Fixed-length character string		√
VARCHAR	char[]	Variable-length character string		

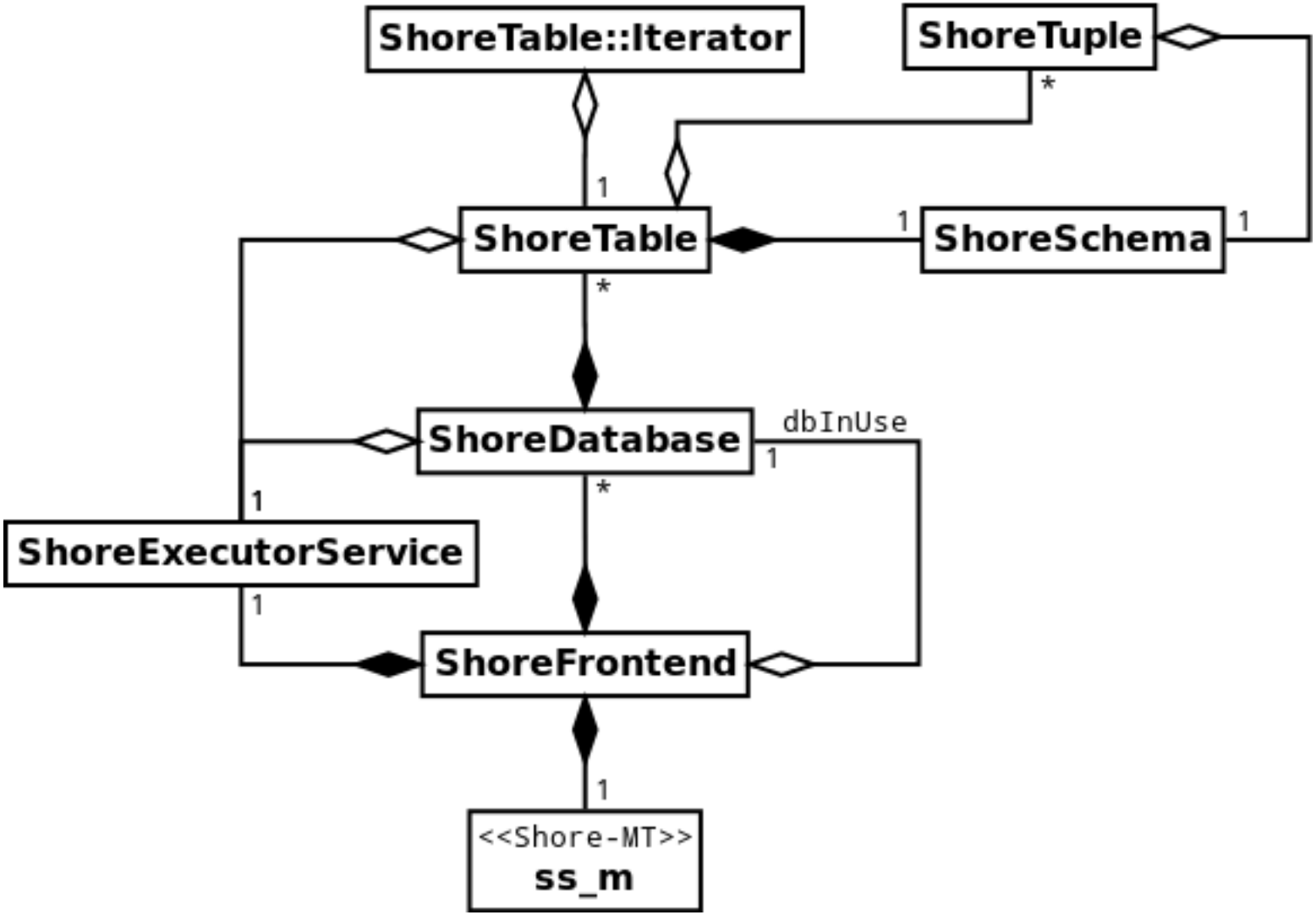
Relational tuples → Shore-MT records



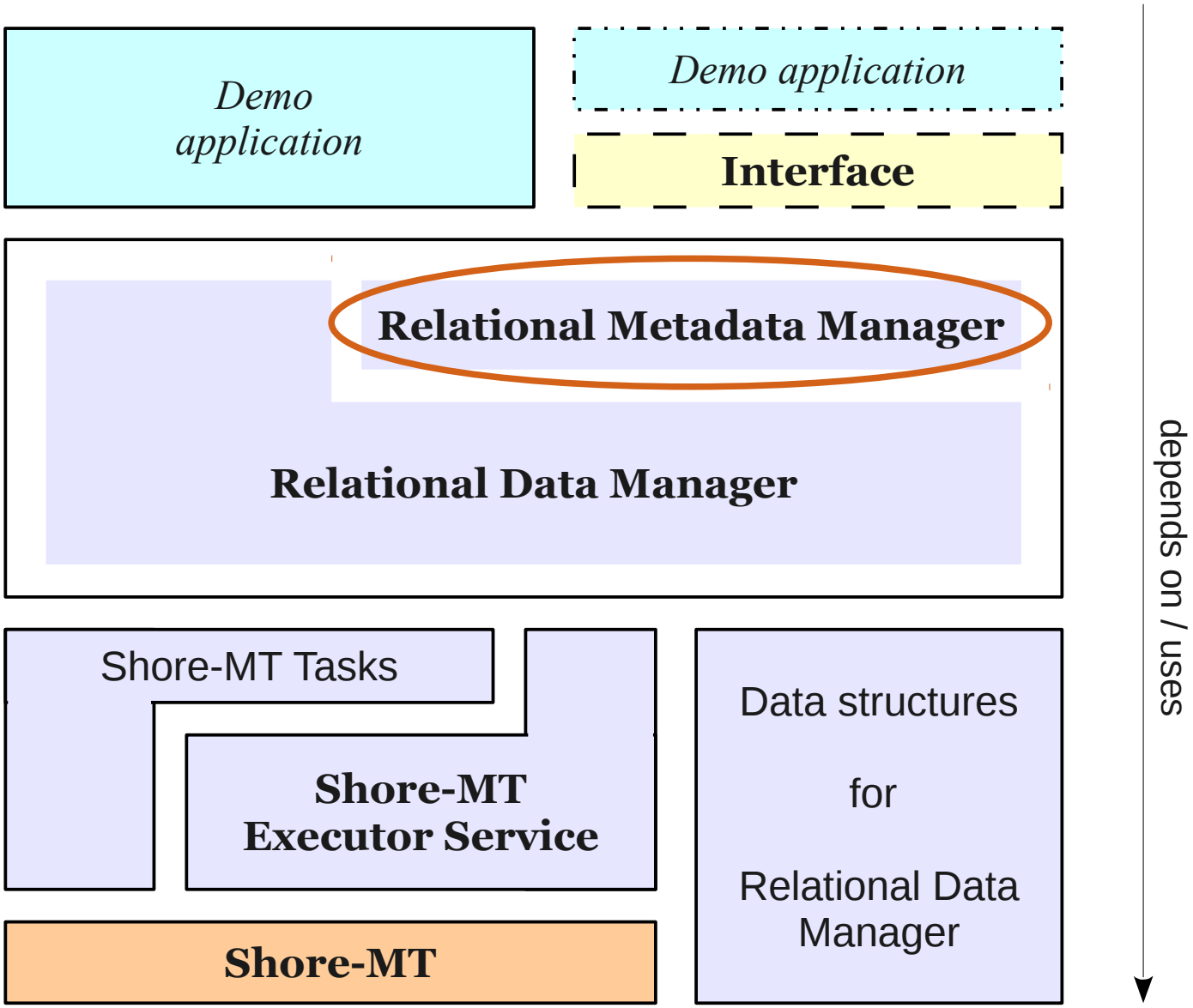
Data structures



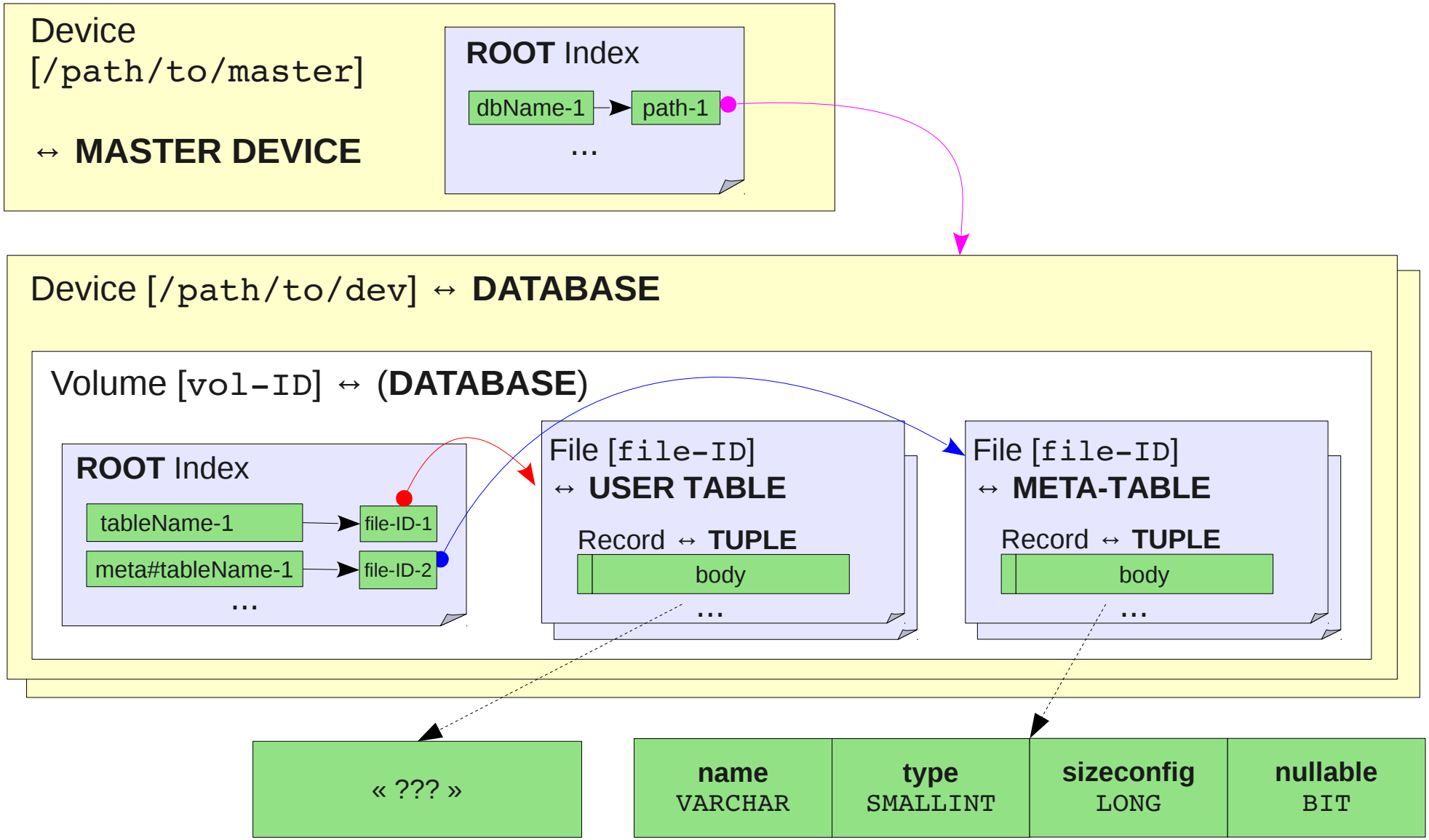
Core object design



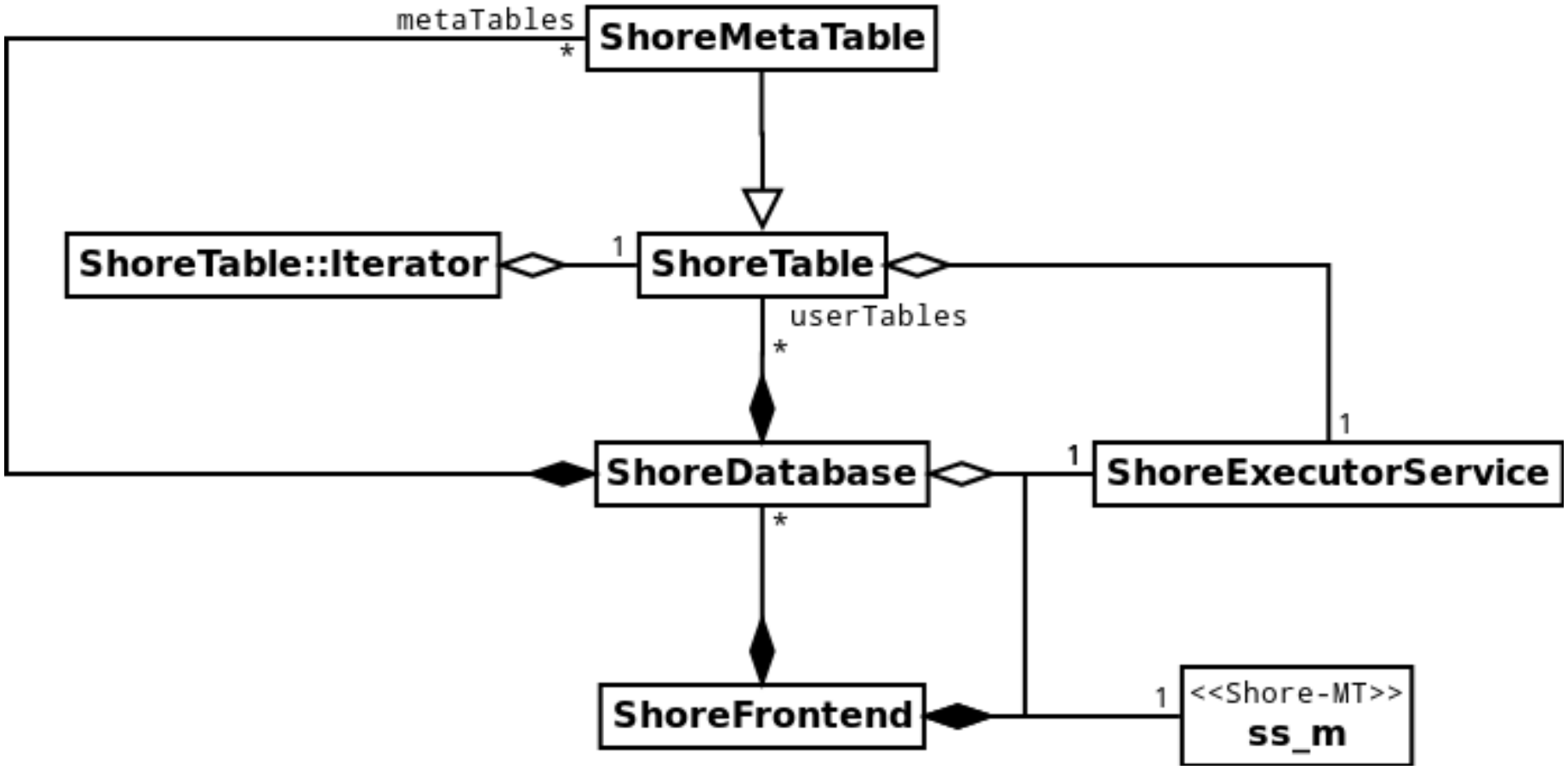
Relational Metadata Manager



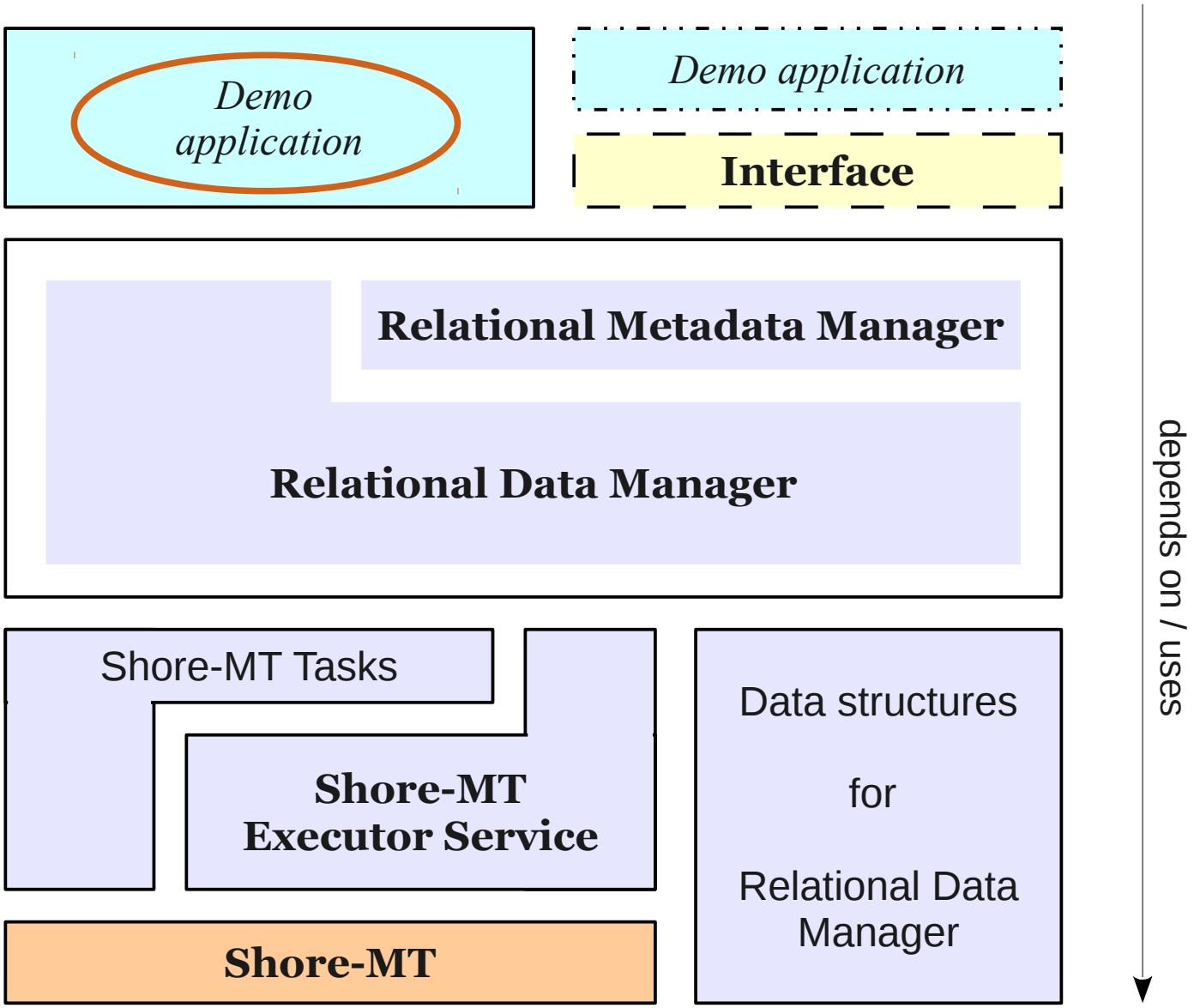
Persisting relational metadata



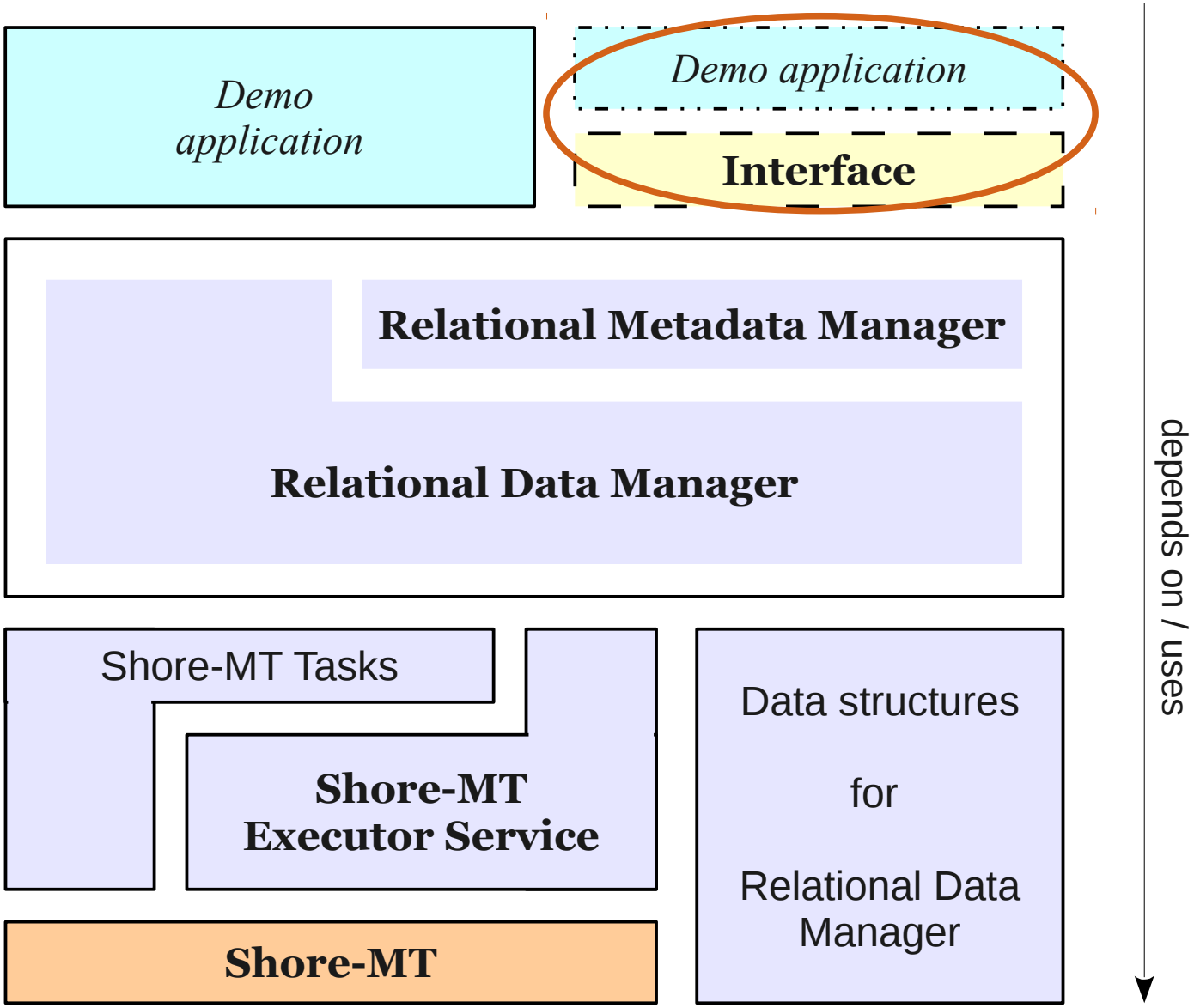
Finalised top-level API



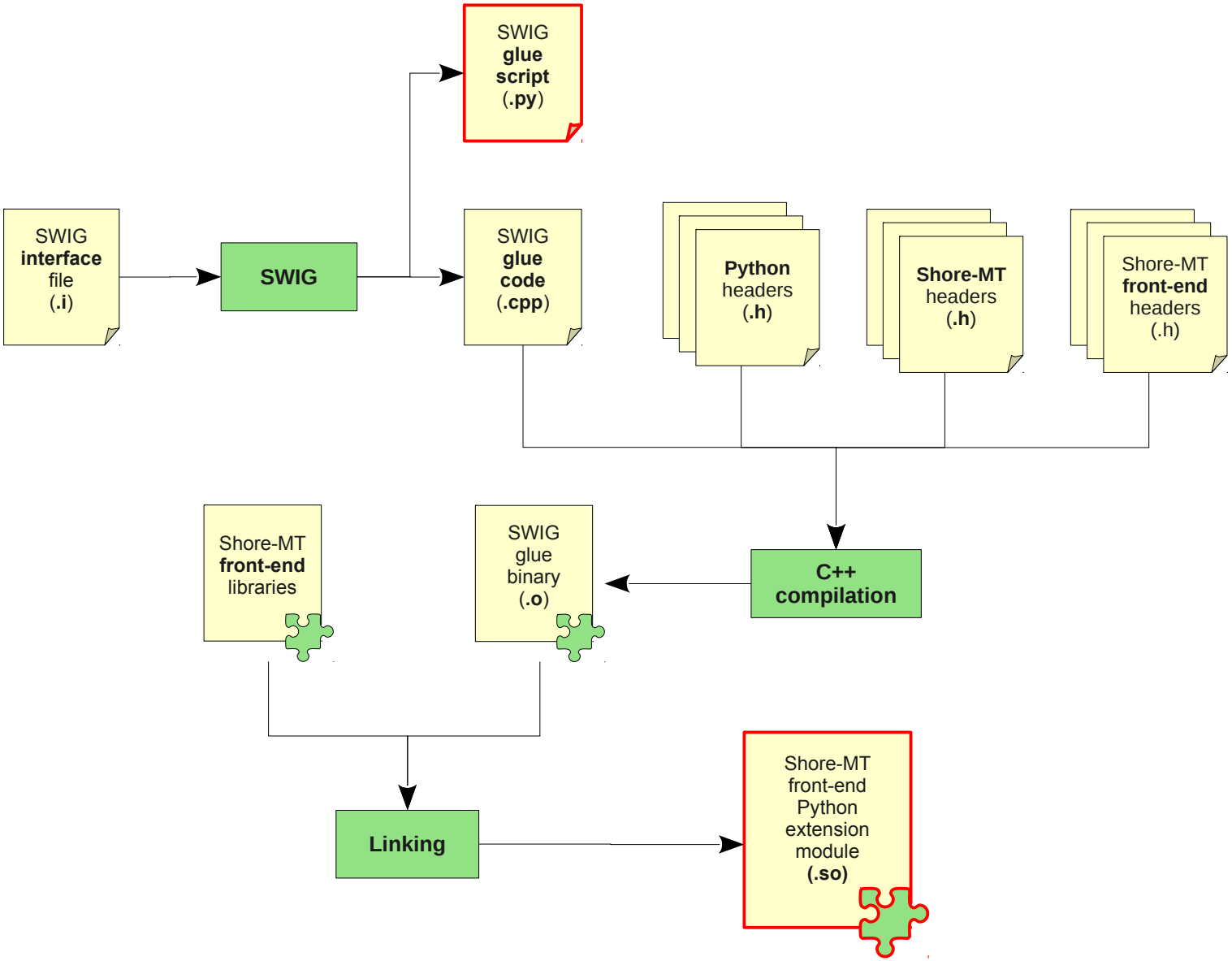
Demo application



SWIG-ing into the scripting world



Interfacing process



Conclusions

- **Achievements**
  - Fully functional deliverables
  - Well-documented design & source codes

- **Lessons learnt**
  - Interfacing very efficient if designed with target scripting in mind
  - Modular design with object-oriented approach boosts development

Possible future work

- Separation & enhancement of Executor Service
- Index supports for user data
- SQL operators for prototyping of transaction processing
- SWIG interface for other target languages
- SQL parser
- Multi-threaded transactions
- etc.

Thank you for your attention!