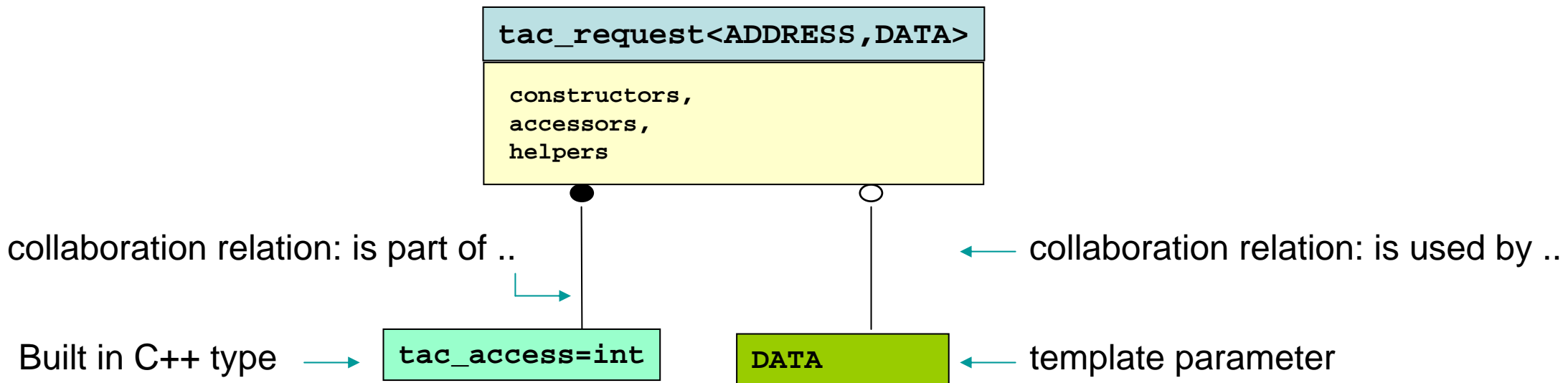
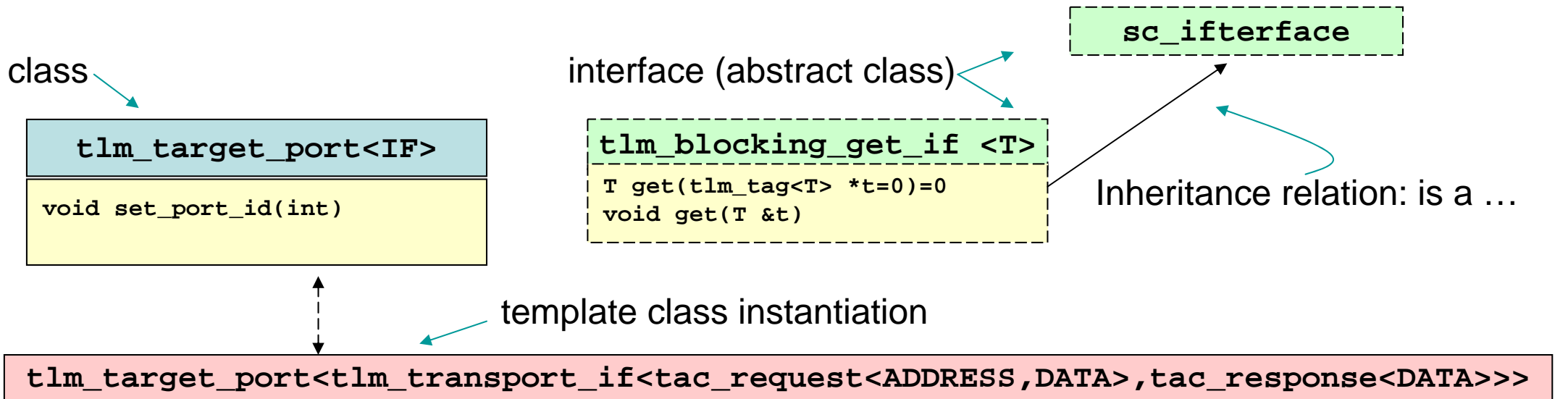


# Notation



# tlm\_[slave,initiator]\_if<REQ,RSP>

1

2

```
tlm_blocking_put_if <T>
void put(const T &t)=0
```

```
tlm_blocking_get_if <T>
T get(tlm_tag<T> *t=0)=0
void get(T &t)
```

```
tlm_blocking_peek_if <T>
T peek(tlm_tag<T> *t=0)=0
void peek(T &t)
```

```
sc_ifterface
```

```
tlm_nonblocking_put_if <T>
bool nb_put(const T &t)=0
bool nb_can_put(tlm_tag<T> *t=0)=0
sc_event & ok_to_put(tlm_tag<T> *t=0)=0
```

```
tlm_nonblocking_get_if <T>
bool nb_get(T &t)=0
bool nb_can_get(tlm_tag<T> *t=0)=0
sc_event & ok_to_get(tlm_tag<T> *t=0)=0
```

```
tlm_nonblocking_peek_if <T>
bool nb_peek(T &t)=0
bool nb_can_peek(tlm_tag<T> *t=0)=0
sc_event & ok_to_peek(tlm_tag<T> *t=0)=0
```

1

2

```
tlm_blocking_get_peek_if <T>
```

```
tlm_nonblocking_get_peek_if <T>
```

1

2

```
tlm_get_if <T>
```

```
tlm_peek_if <T>
```

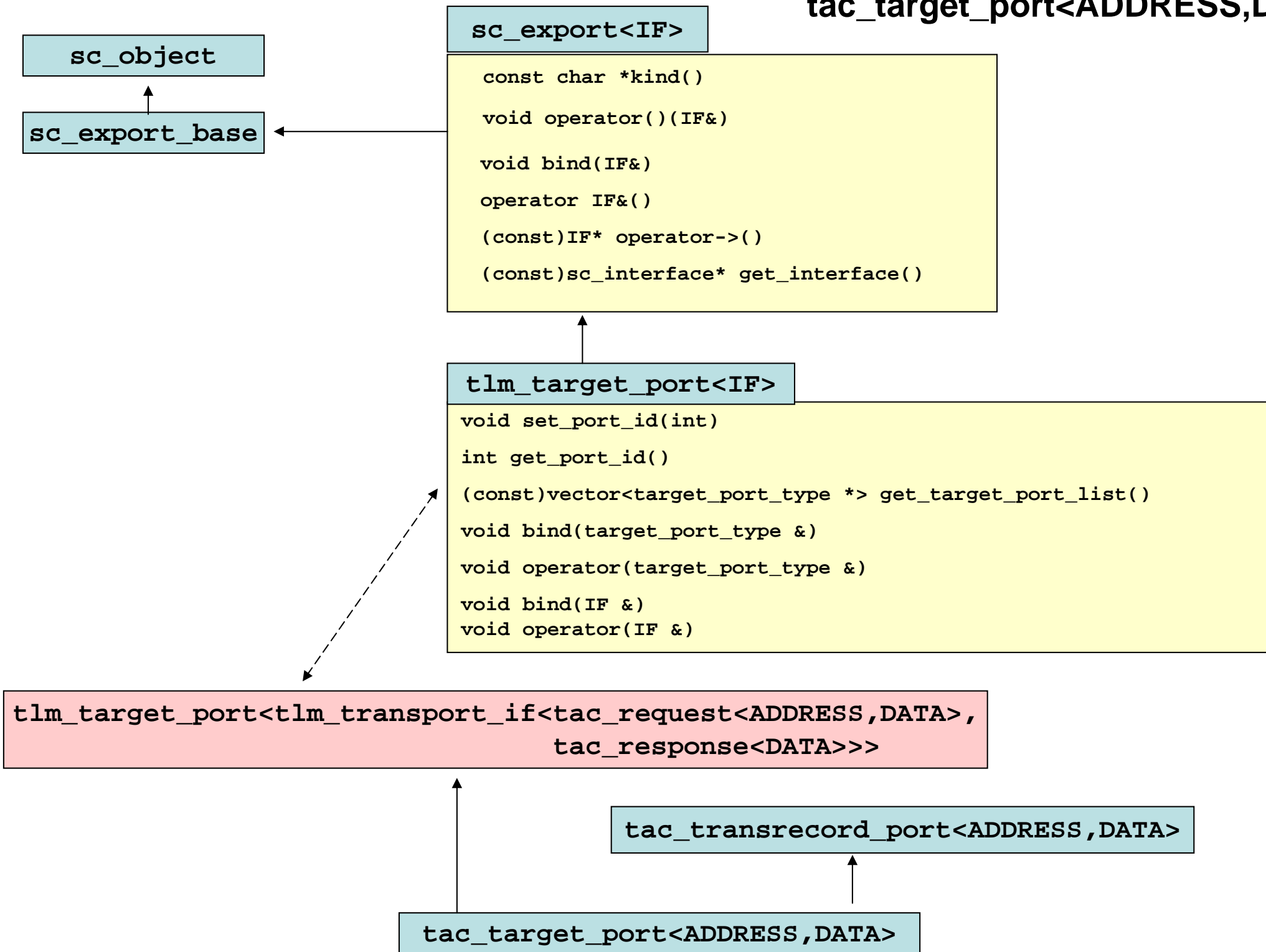
```
tlm_put_if <REQ>
```

```
tlm_get_peek_if <RSP>
```

```
tlm_slave_if<REQ,RSP>
```

```
tlm_initiator_if<REQ,RSP>
```

# tac\_target\_port<ADDRESS,DATA>



# tac\_initiator\_port<ADDRESS,DATA,N>

sc\_object

sc\_port\_base

sc\_port<IF, N=1, sc\_port\_policy=SC\_ONE\_OR\_MORE\_BOUND>

```
const char *kind()
void operator()(IF&)
void operator()(sc_port<IF,N>&)
void bind(IF&)
void bind(sc_port<IF,N>&)
int size()
(const)IF* operator->()
(const)IF* operator[](int)
(const)sc_interface* get_interface()
```

tlm\_initiator\_port<IF,N>

```
void set_target_port_list(vector<target_port_type *>& )
void set_initiator_port_list(vector<initiator_port_type *>& )
(const)vector<initiator_port_type *> get_initiator_port_list()
(const)vector<target_port_type *> get_target_port_list()
void bind(target_port_type &)
void operator(target_port_type &)
void bind(sc_port_b<IF> &)
void operator(sc_port_b<IF> &)
void bind(IF &)
void operator(IF &)
```

tlm\_initiator\_port<tlm\_transport\_if<tac\_request<ADDRESS,DATA>,tac\_response<DATA>>,N>

tac\_if

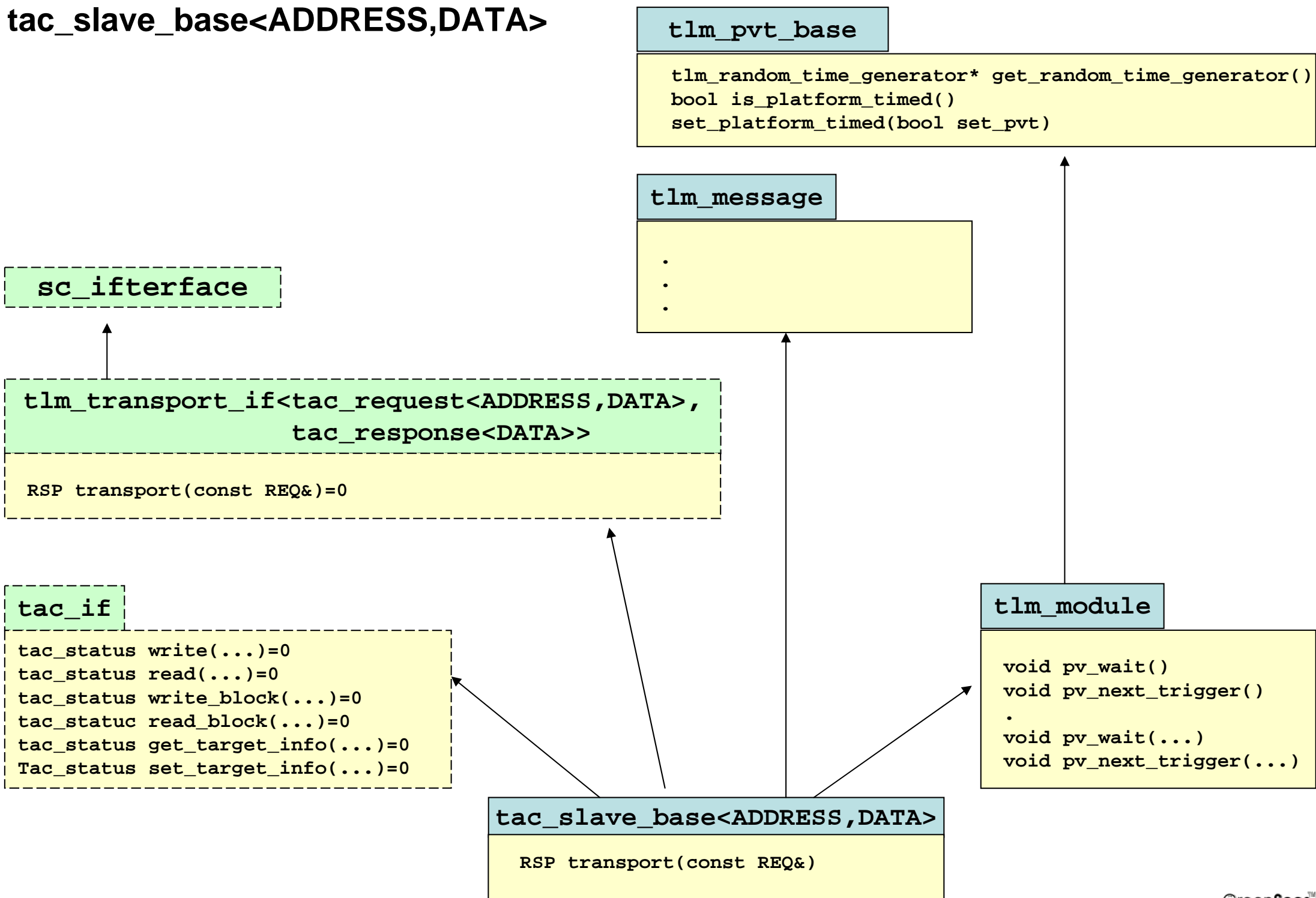
```
tac_status write(...)=0
tac_status read(...)=0
tac_status write_block(...)=0
tac_status read_block(...)=0
tac_status get_target_info(...)=0
Tac_status set_target_info(...)=0
```

tac\_transrecord\_port<ADDRESS,DATA>

tac\_initiator\_port<ADDRESS,DATA,N>

```
do_transport(REQ,RSP)
implementation of tac_if
```

# tac\_slave\_base<ADDRESS,DATA>



**sc\_event**

```
void notify()
void notify(const sc_time &)
void notify(double, sc_time_unit)
void cancel()
void notify_delayed();
void notify_delayed( const sc_time& );
void notify_delayed( double, sc_time_unit );
sc_event_or_list& operator | ( const sc_event& ) const;
sc_event_and_list& operator & ( const sc_event& ) const;
```

**tlm\_event**

```
void notify()
void notify(const sc_time &)
void notify(double, sc_time_unit)
void notify_delayed(const sc_time &)
void notify_delayed(double, sc_time_unit)
```



**tac\_metadata** **tac\_request<ADDRESS,DATA>**, **tac\_responce<DATA>**, **tac\_metadata**, **tac\_status**

```
void set_metadata(const char *)  
void set_metadata(const string)  
string get_metadata() const  
static const char * get_key_string(const info_key key)  
const char * get_info_string(const char *) const;  
const char * get_info_string(const info_key) const  
const char * get_target_name() const  
const char * get_slave_base_ptr()  
const char * get_target_port_id()  
const char * get_endianness()  
const char * get_memory_ptr()  
const char * get_memory_size()  
const char * get_memory_mau()  
const char * get_memory_typeid()  
const char * get_synchro_address()  
const char * get_add_synchro_address()  
char * get_remove_synchro_address()
```

**tac\_error\_reason**

```
void set_reason(const string &  
string& get_reason()
```

**tac\_request<ADDRESS,DATA>**

constructors,  
accessors,  
helpers

**tac\_status**

```
[set,reset,is]_ok/error/no_responce()
```

**tac\_responce<DATA>**

constructors,  
accessors,  
helpers

tac\_access=int

tac\_mode=int

ADDRESS=templ.param

DATA=templ.param

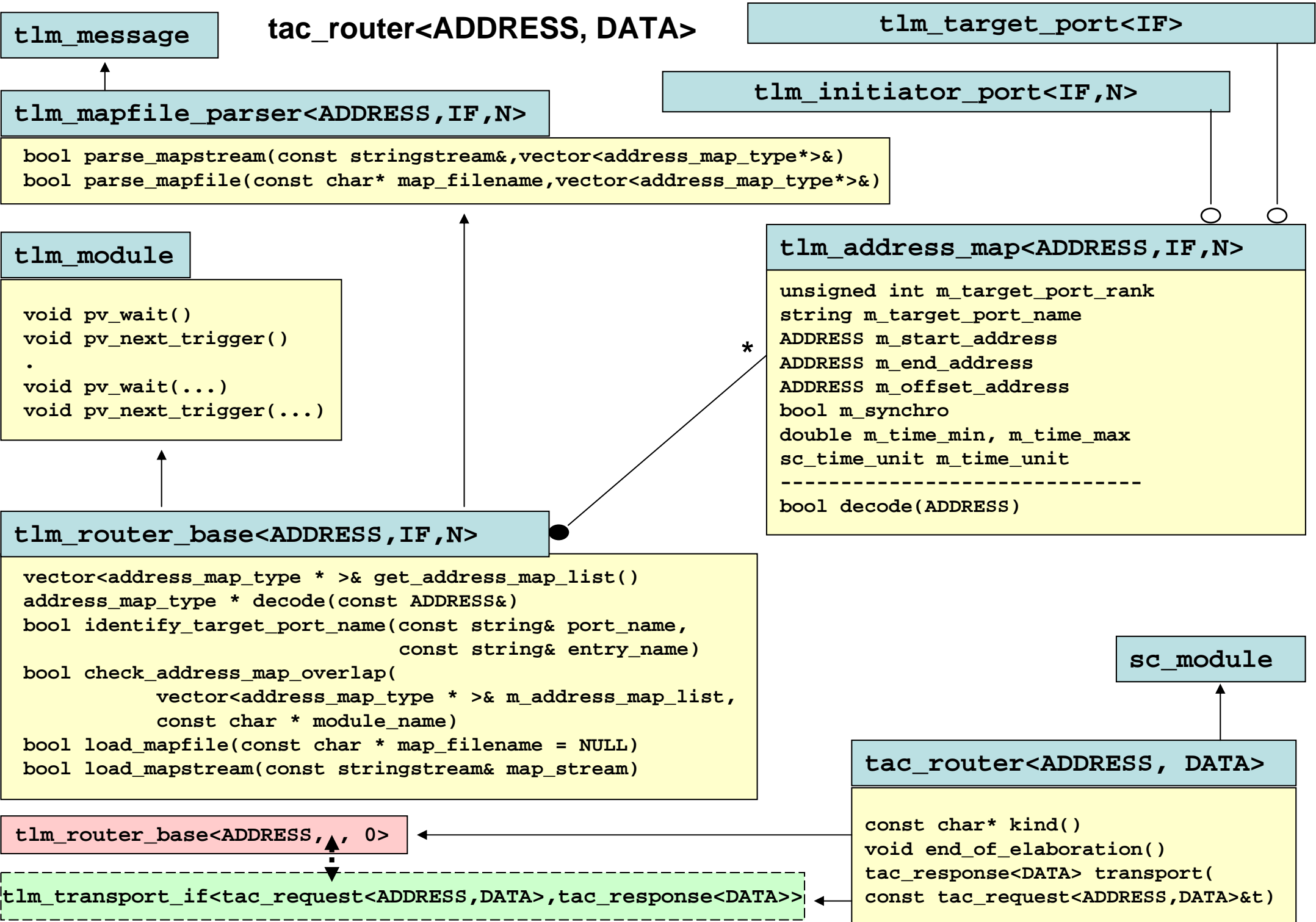
number=int

byte\_enable=int

port\_id=int

bl\_byte\_en=int

bl\_byte\_en\_period=int



**tlm\_message**

**tlm\_router<ADDRESS, DATA>**

**tlm\_target\_port<IF>**

**tlm\_mapfile\_parser<ADDRESS, IF, N>**

**tlm\_initiator\_port<IF, N>**

```

bool parse_mapstream(const stringstream&, vector<address_map_type*>&)
bool parse_mapfile(const char* map_filename, vector<address_map_type*>&)
  
```

**tlm\_module**

```

void pv_wait()
void pv_next_trigger()
.
void pv_wait(...)
void pv_next_trigger(...)
  
```

**tlm\_address\_map<ADDRESS, IF, N>**

```

unsigned int m_target_port_rank
string m_target_port_name
ADDRESS m_start_address
ADDRESS m_end_address
ADDRESS m_offset_address
bool m_synchro
double m_time_min, m_time_max
sc_time_unit m_time_unit
-----
bool decode(ADDRESS)
  
```

**tlm\_router\_base<ADDRESS, IF, N>**

```

vector<address_map_type * >& get_address_map_list()
address_map_type * decode(const ADDRESS&)
bool identify_target_port_name(const string& port_name,
                               const string& entry_name)
bool check_address_map_overlap(
    vector<address_map_type * >& m_address_map_list,
    const char * module_name)
bool load_mapfile(const char * map_filename = NULL)
bool load_mapstream(const stringstream& map_stream)
  
```

**sc\_module**

**tlm\_router\_base<ADDRESS, , 0>**

**tlm\_router\_base<ADDRESS, , 0>**

**tlm\_transport\_if<tac\_request<ADDRESS, DATA>, tac\_response<DATA>>**

```

const char* kind()
void end_of_elaboration()
tac_response<DATA> transport(
    const tac_request<ADDRESS, DATA>&t)
  
```