CORAL Server Project

Design and Development Proposal for Client, Server and Proxy

Andrea Valassi (CERN IT-DM)

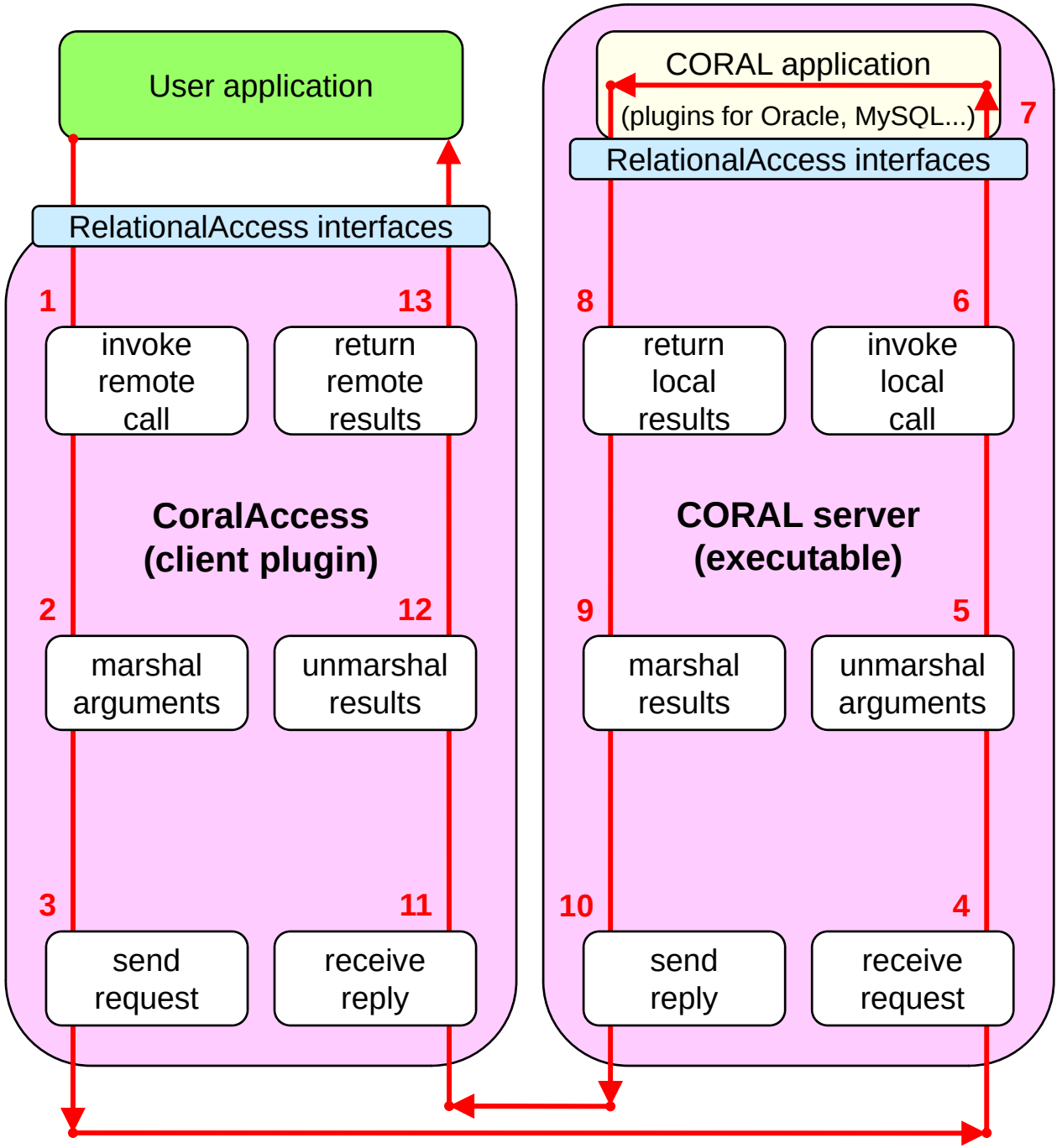
CORAL Server Internal Review, 8th December 2008

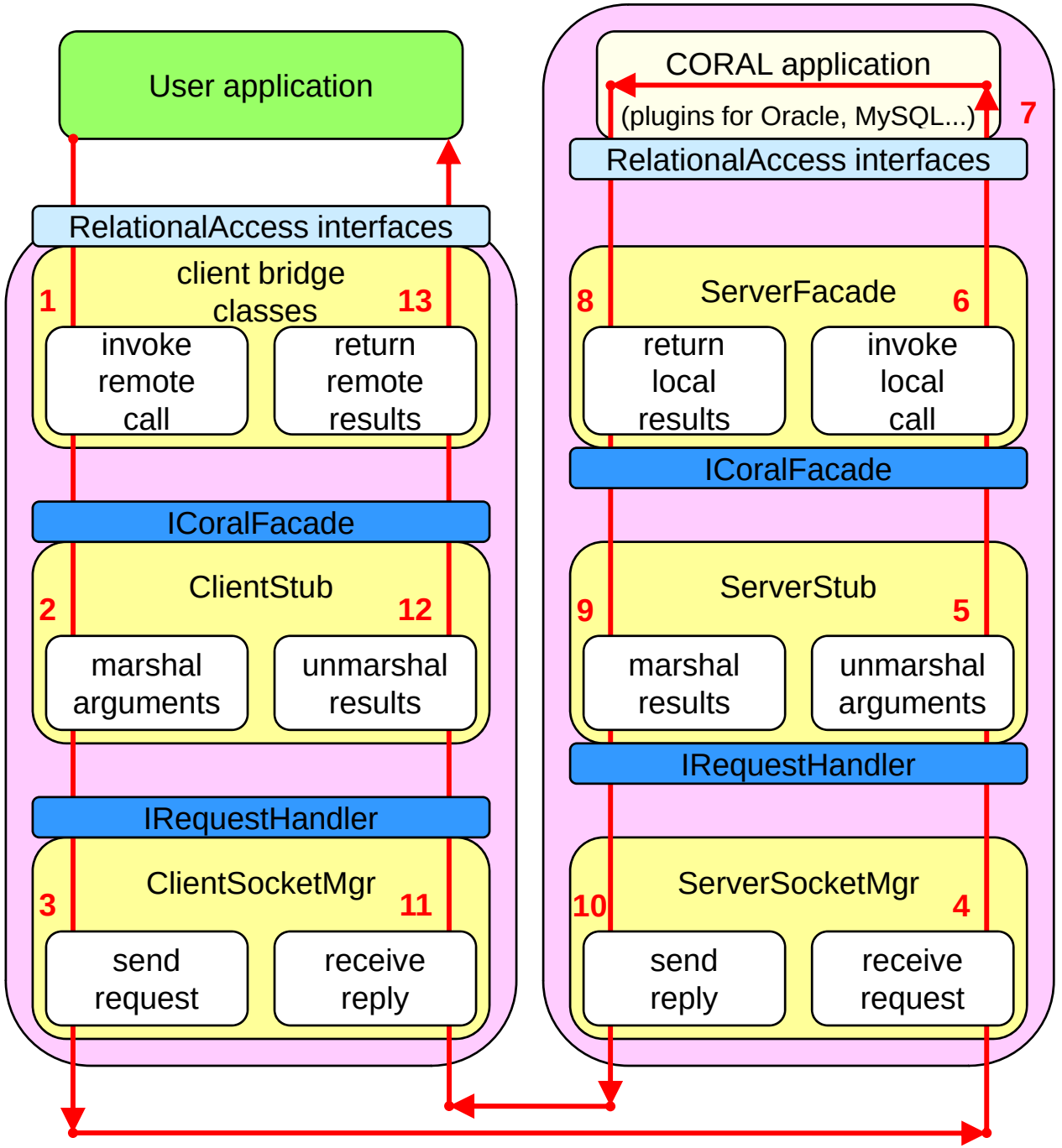
CERN - IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

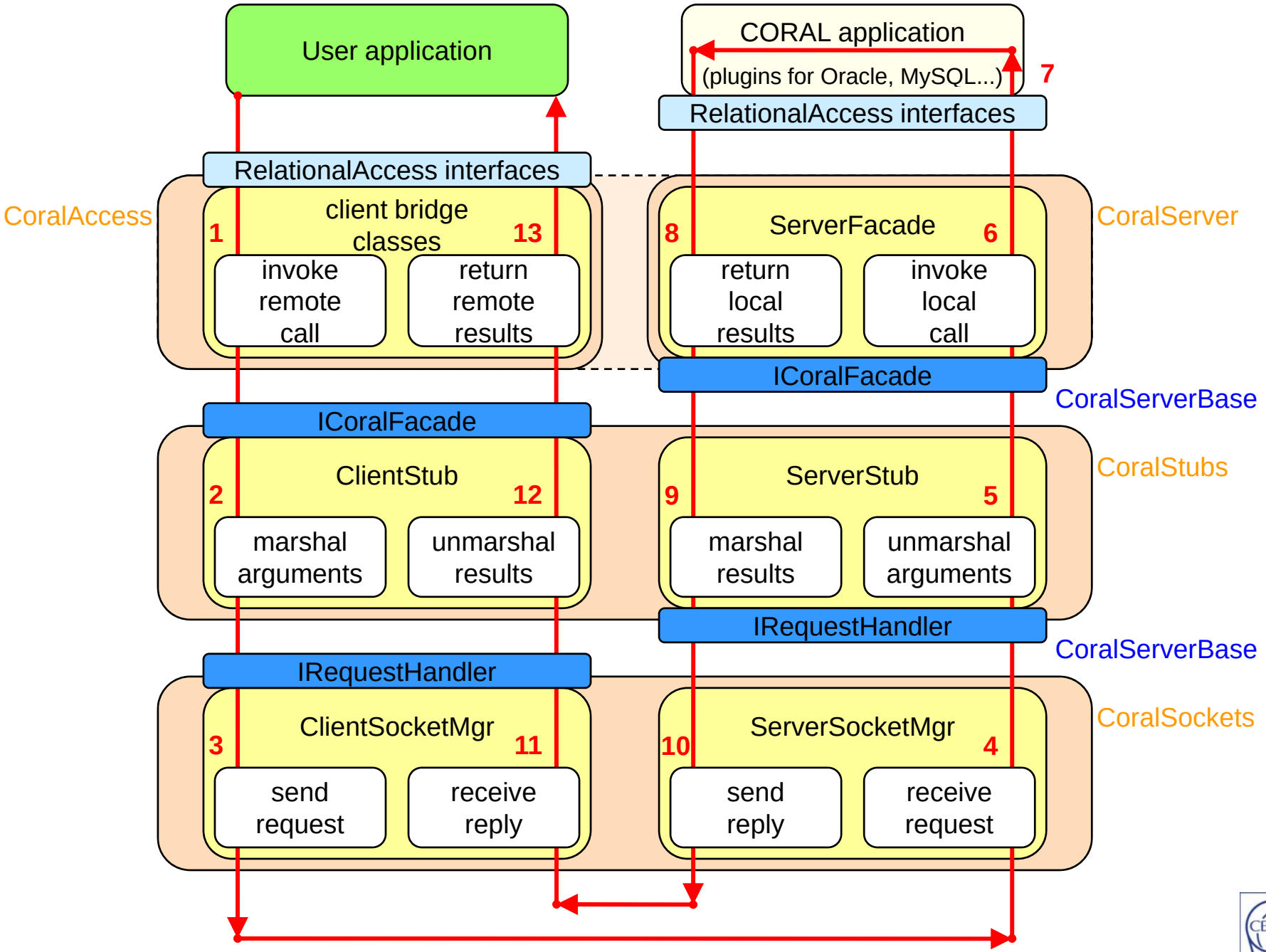
Proposed design guidelines

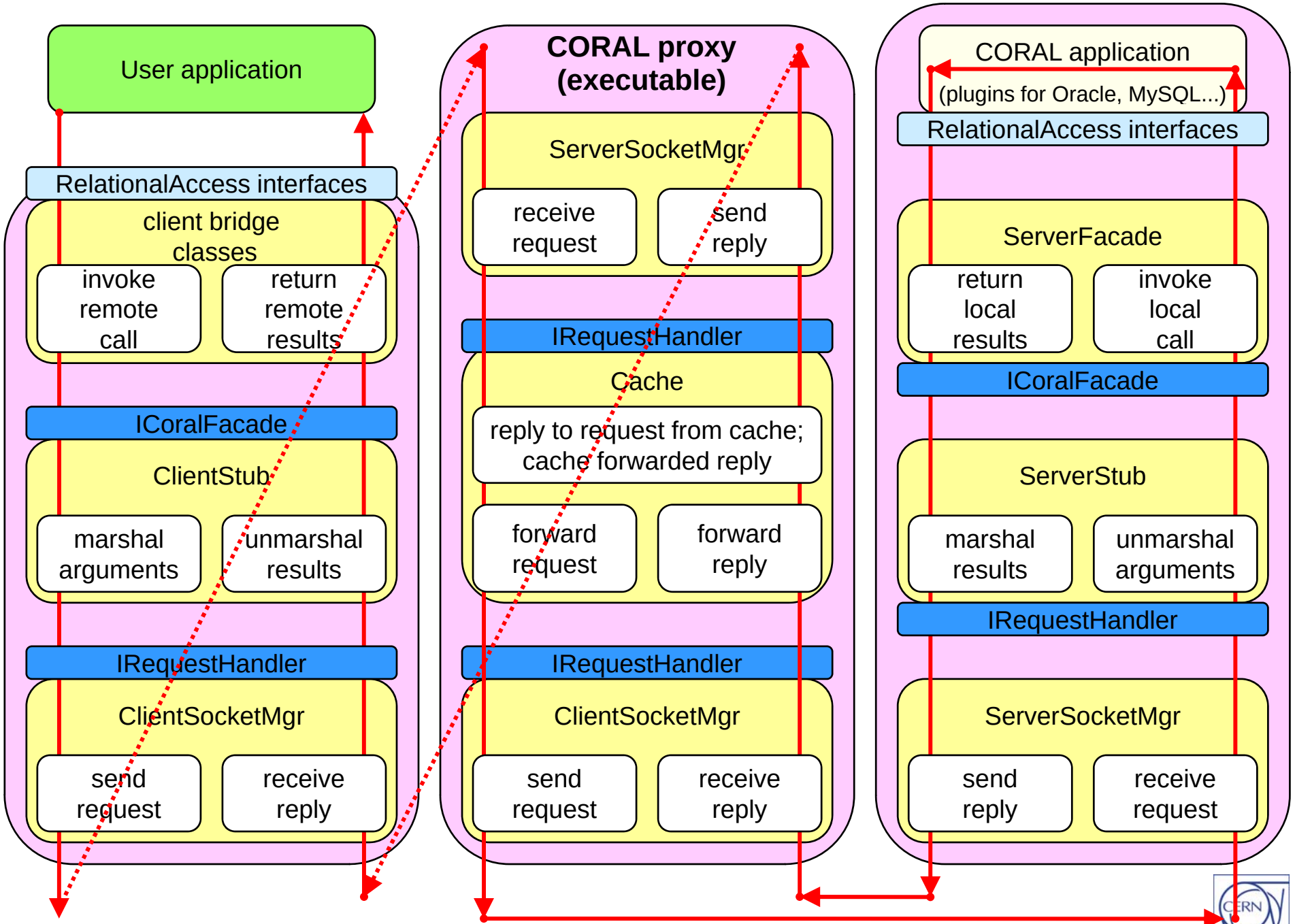
CERN IT Department

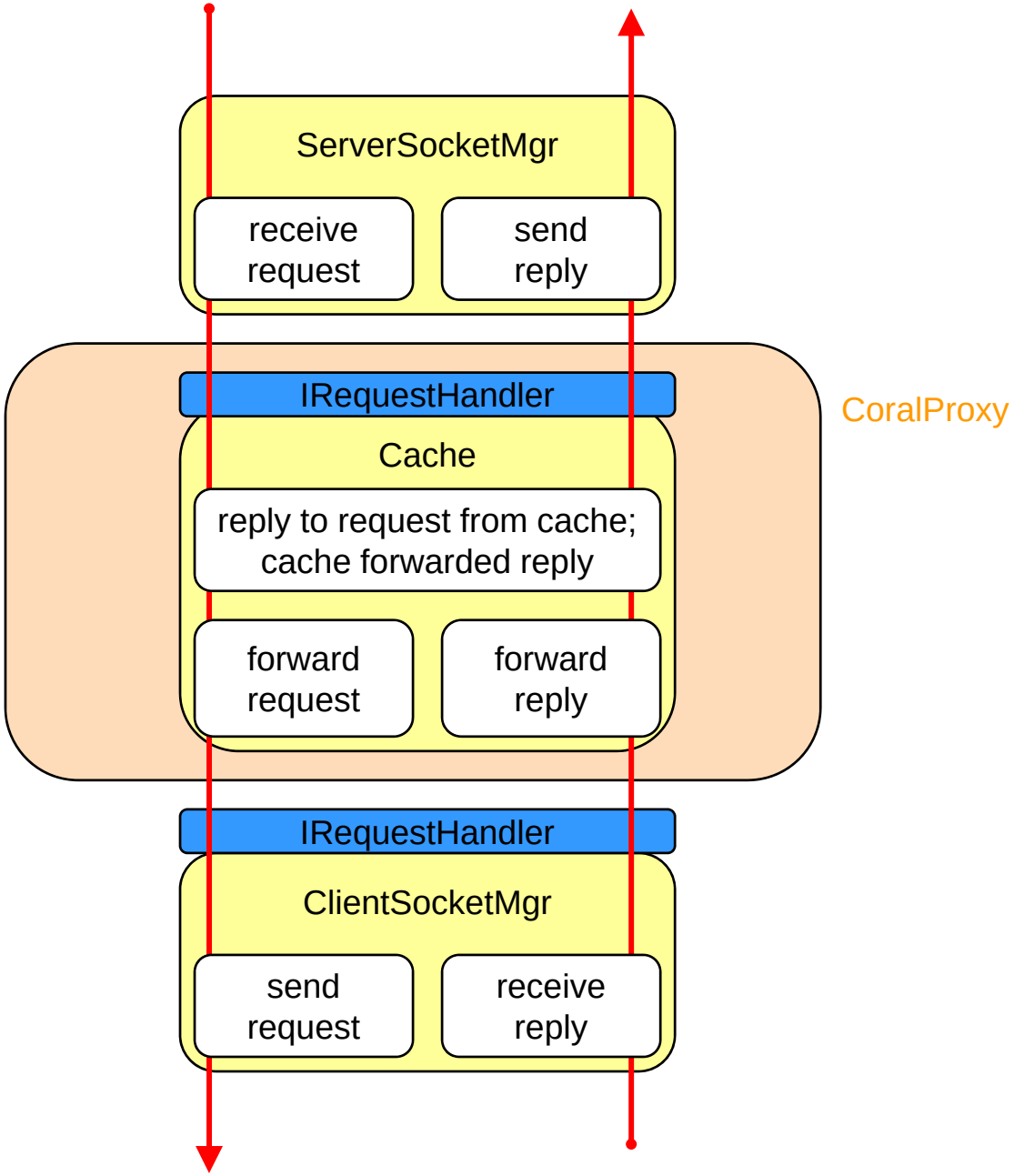
- **Joint design of server/client/proxy components**
  - Software collaboration vs. TCP packet contract
  - Development model: ‘horizontal’ vs. ‘vertical’
- **Allow several people to work independently**
  - Minimize software dependencies and couplings
- **Promote standalone tests of components**
  - Try to intercept bugs before they show up in system tests
- **Modular design favoring iterative development**
  - Allow upgrade of one component without touching the others
- ***Decouple components using abstract interfaces***
  - Minimize size of public headers: thin interfaces, simple constructors
  - Encapsulation: hide implementation details within each component

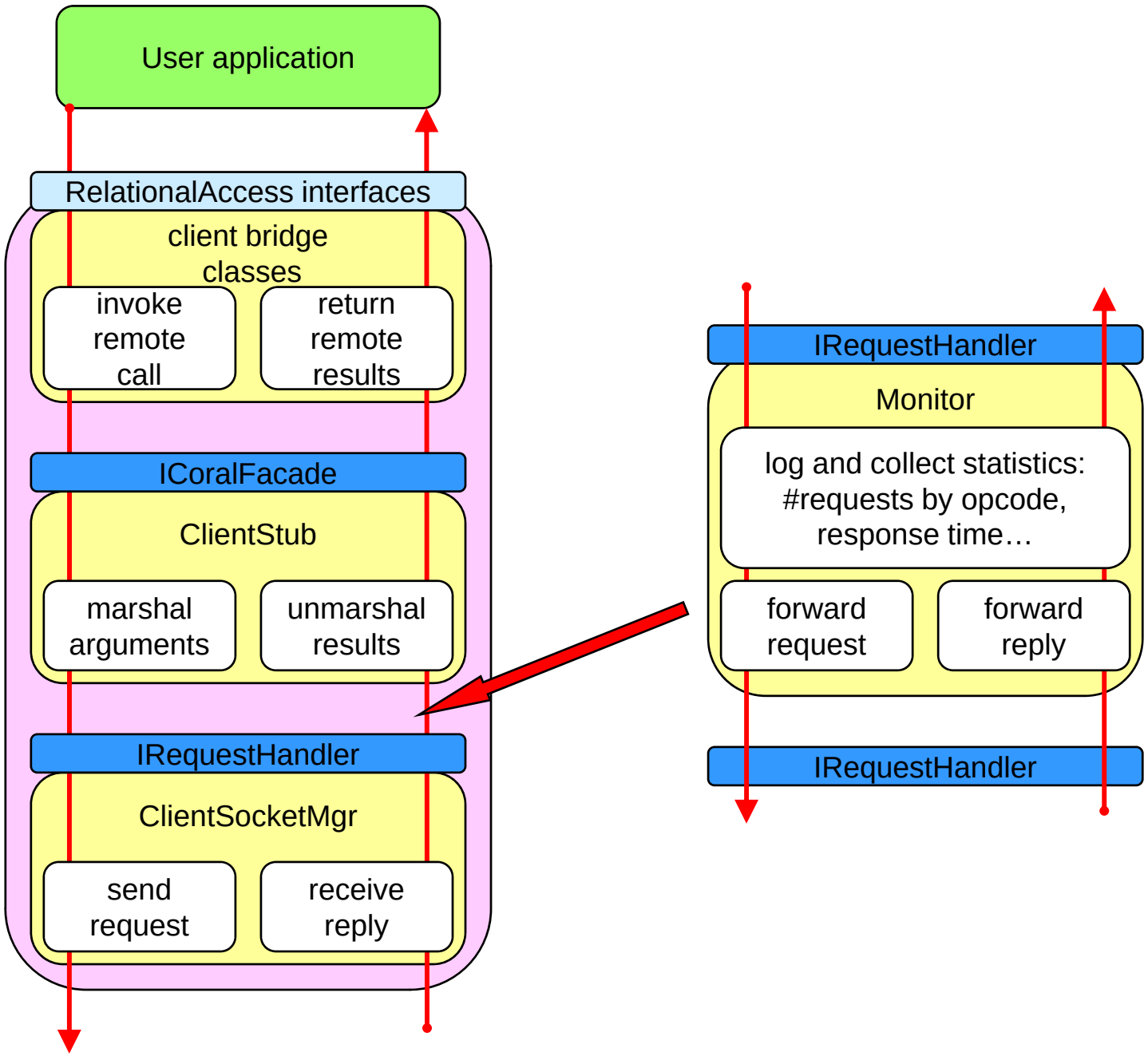

User application
RelationalAccess interfaces
1
invoke remote call
13
return remote results
CoralAccess (client plugin)
2
marshal arguments
12
unmarshal results
3
send request
11
receive reply
CORAL application
(plugins for Oracle, MySQL...)
7
RelationalAccess interfaces
8
return local results
6
invoke local call
CORAL server (executable)
9
marshal results
5
unmarshal arguments
10
send reply
4
receive request

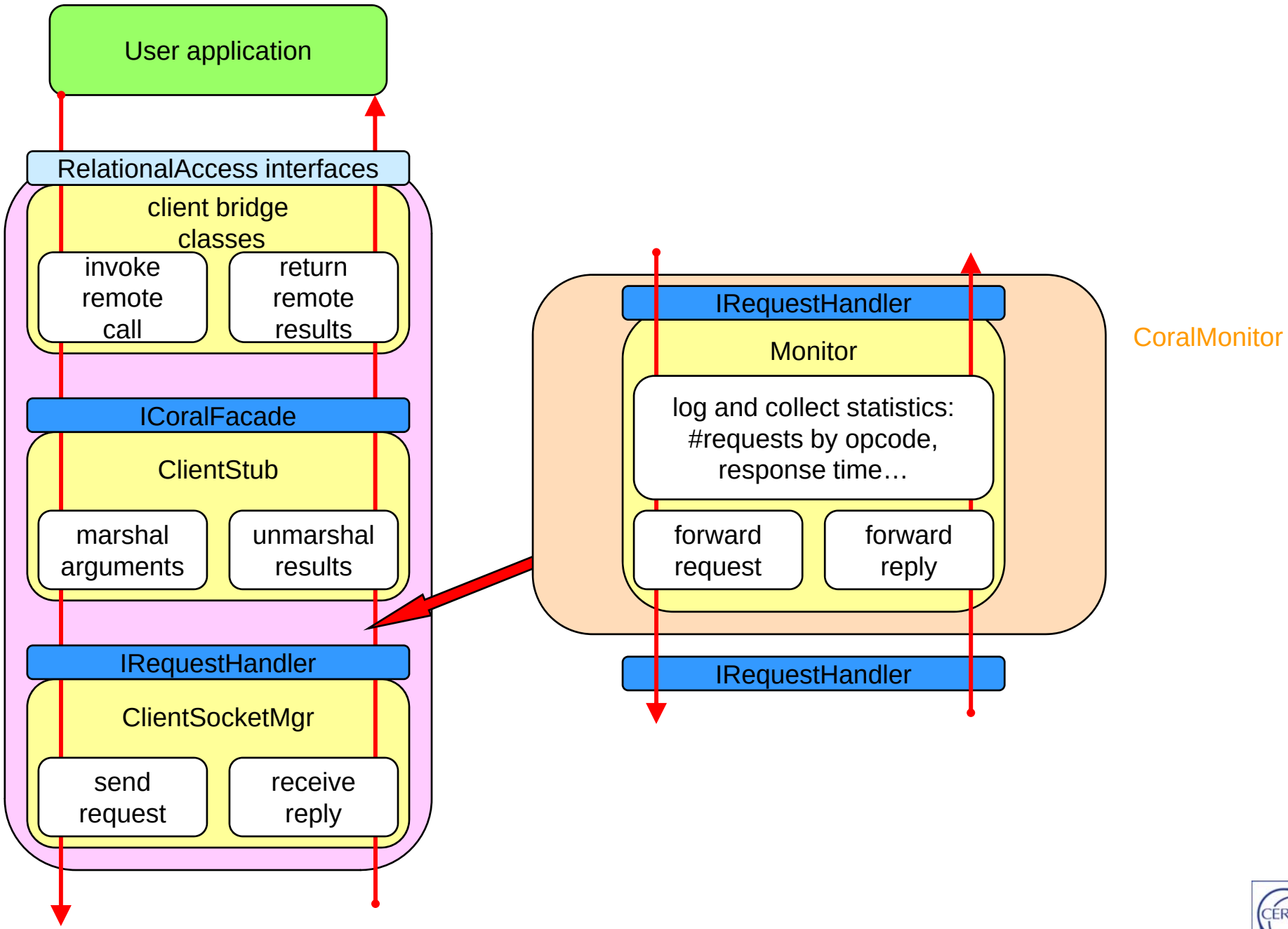

User application
RelationalAccess interfaces
client bridge classes
1
13
invoke remote call
return remote results
ICoralFacade
ClientStub
2
12
marshal arguments
unmarshal results
IRequestHandler
ClientSocketMgr
3
11
send request
receive reply
CORAL application
(plugins for Oracle, MySQL...)
7
RelationalAccess interfaces
ServerFacade
8
6
return local results
invoke local call
ICoralFacade
ServerStub
9
5
marshal results
unmarshal arguments
IRequestHandler
ServerSocketMgr
10
4
send reply
receive request

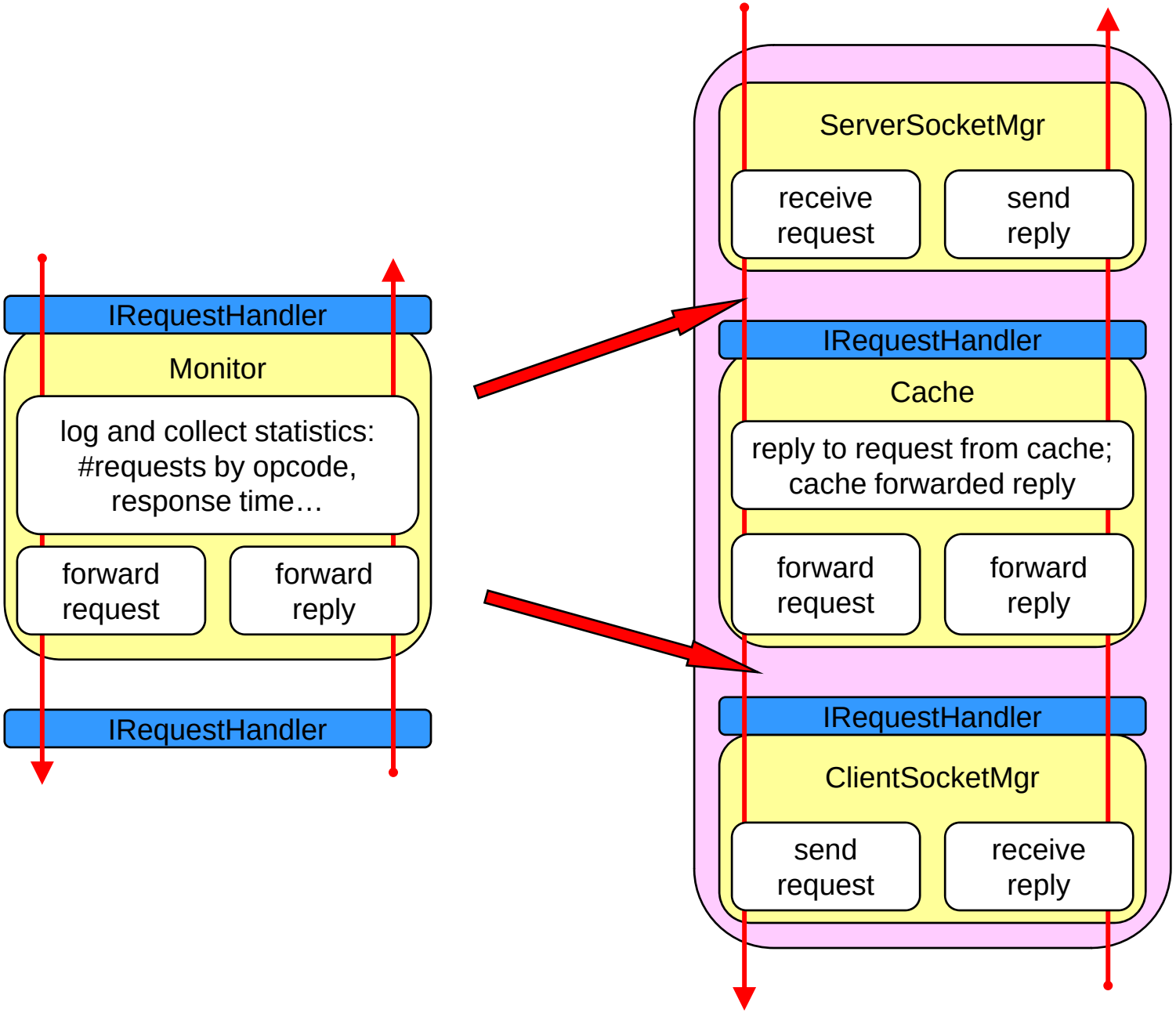

User application
CORAL application
(plugins for Oracle, MySQL...)
7
RelationalAccess interfaces
RelationalAccess interfaces
CoralAccess
client bridge classes
1
13
invoke remote call
return remote results
8
ServerFacade
6
return local results
invoke local call
CoralServer
ICoralFacade
CoralServerBase
ICoralFacade
ClientStub
2
12
marshal arguments
unmarshal results
9
ServerStub
5
marshal results
unmarshal arguments
CoralStubs
IRequestHandler
CoralServerBase
IRequestHandler
ClientSocketMgr
3
11
send request
receive reply
10
ServerSocketMgr
4
send reply
receive request
CoralSockets

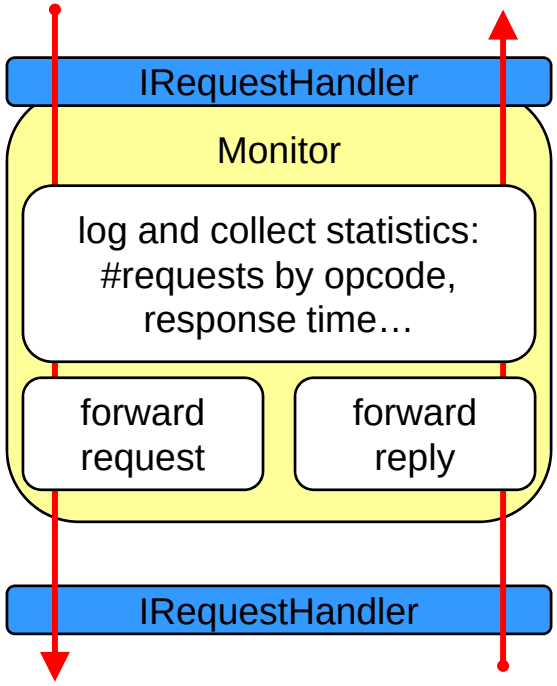

User application
RelationalAccess interfaces
client bridge classes
invoke remote call
return remote results
ICoralFacade
ClientStub
marshal arguments
unmarshal results
IRequestHandler
ClientSocketMgr
send request
receive reply
CORAL proxy (executable)
ServerSocketMgr
receive request
send reply
IRequestHandler
Cache
reply to request from cache; cache forwarded reply
forward request
forward reply
IRequestHandler
ClientSocketMgr
send request
receive reply
CORAL application
(plugins for Oracle, MySQL...)
RelationalAccess interfaces
ServerFacade
return local results
invoke local call
ICoralFacade
ServerStub
marshal results
unmarshal arguments
IRequestHandler
ServerSocketMgr
send reply
receive request

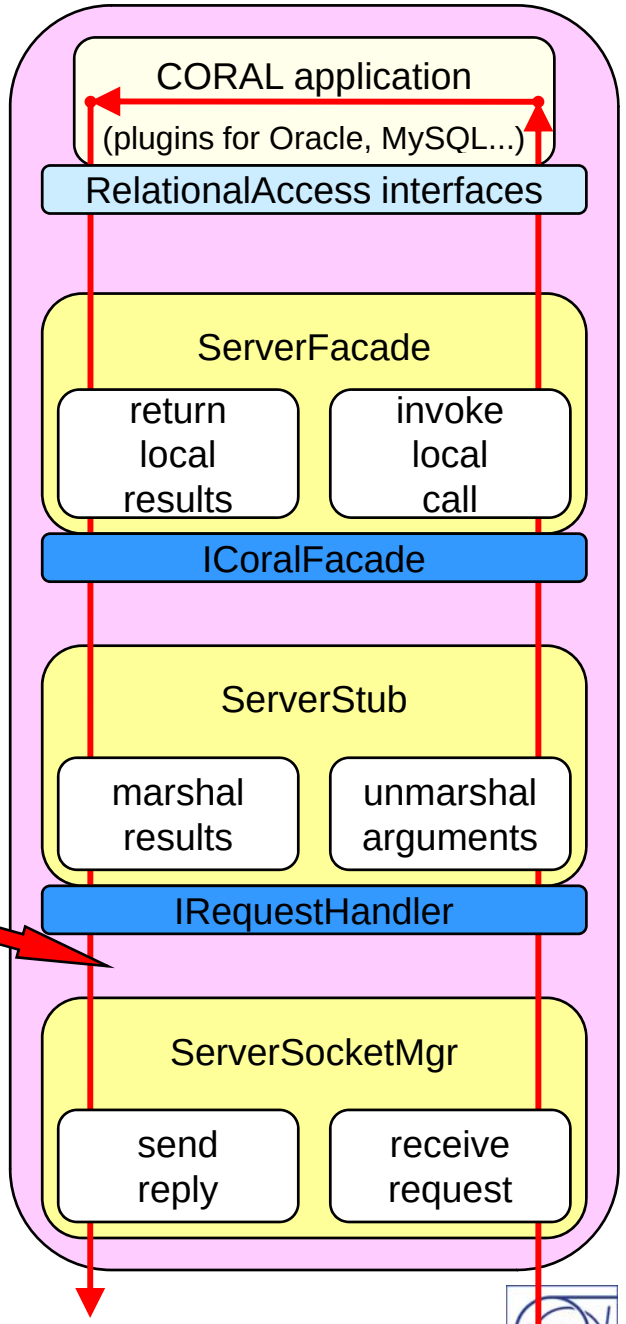

ServerSocketMgr
receive request
send reply
IRequestHandler
Cache
reply to request from cache; cache forwarded reply
forward request
forward reply
CoralProxy
IRequestHandler
ClientSocketMgr
send request
receive reply


User application
RelationalAccess interfaces
client bridge classes
invoke remote call
return remote results
ICoralFacade
ClientStub
marshal arguments
unmarshal results
IRequestHandler
ClientSocketMgr
send request
receive reply
IRequestHandler
Monitor
log and collect statistics: #requests by opcode, response time…
forward request
forward reply
IRequestHandler

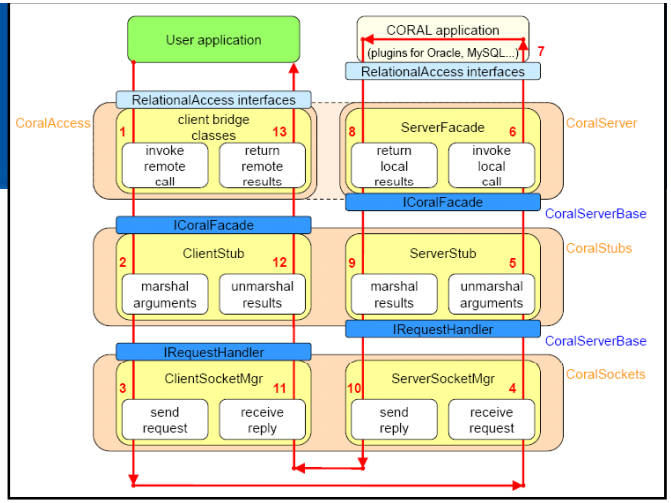

User application
RelationalAccess interfaces
client bridge classes
invoke remote call
return remote results
ICoralFacade
ClientStub
marshal arguments
unmarshal results
IRequestHandler
ClientSocketMgr
send request
receive reply
IRequestHandler
Monitor
log and collect statistics: #requests by opcode, response time…
forward request
forward reply
IRequestHandler
CoralMonitor


IRequestHandler
Monitor
log and collect statistics:
#requests by opcode,
response time…
forward
request
forward
reply
IRequestHandler
ServerSocketMgr
receive
request
send
reply
IRequestHandler
Cache
reply to request from cache;
cache forwarded reply
forward
request
forward
reply
IRequestHandler
ClientSocketMgr
send
request
receive
reply


IRequestHandler
Monitor
log and collect statistics:
#requests by opcode,
response time…
forward
request
forward
reply
IRequestHandler


CORAL application
(plugins for Oracle, MySQL...)
RelationalAccess interfaces
ServerFacade
return
local
results
invoke
local
call
ICoralFacade
ServerStub
marshal
results
unmarshal
arguments
IRequestHandler
ServerSocketMgr
send
reply
receive
request

Package 1 – interfaces



- *In my diagram:* ***CoralServerBase***

- Two main sets of abstract interfaces
  - ICoralFacade (all relational operations that need network roundtrips)
    - Auxiliary interfaces: query definition, …
  - IRequestHandler (get string reply to a string request)
    - Auxiliary interfaces: handler factories, message (byte stream), …

- A few common classes may also be needed
  - Dummy request handler for tests
  - Adapter of a string request to extract relational header

Package 1 – interfaces

CERN IT Department

- ICoralFacade
  - This defines what relational info needs to be shipped on the network (and how/when to do that)
    - Some CORAL API calls do not need network round trips
    - Example: split query definition (local) from query execution (remote)
    - Minimise network round trips by deferring and grouping them
  - Link request type to reply type
    - If you send a 'fetchRows' request message via sockets, the reply message must contain a list of rows (or an exception)

- IRequestHandler
  - Do not assume that exchanged messages will be relational
    - Reorder fields in agreed packet header: first those needed for low-level socket management (endian, size…); then the relational ones (opcode, cacheable...), treated as part of the request/reply string
  - Need optimization to avoid data copy overhead

Package 1 – interfaces

CERN IT Department

```
class IRequestHandler
{
public:
  /// Destructor.
  virtual ~IRequestHandler(){}

  /// Handle a request message and return a reply message.
  /// The ownership of the reply message is delegated to the caller.
  virtual std::auto_ptr<const IMessage> replyToRequest( const IMessage& request ) = 0;
};
```

Or even better, replace
  const std::string&
by
  const coral::Blob*

```
class IMessage
{
public:
  /// Destructor.
  virtual ~IMessage(){}

  /// The message payload for this request/reply.
  virtual const std::string& payload() const = 0;
```

Package 1 – interfaces

CERN IT Department

```
/// ObjectID type used for sessionID, cursorID, bulkOperationID (valid Token's are always > 0).
typedef int Token;

class ICoralFacade
{
public:
  /// Destructor.
  virtual ~ICoralFacade(){}
  [...]

  /// Create a new session and return its sessionID.
  virtual const Token connect( const std::string& dbUrl,
                               const coral::AccessMode mode = coral::Update ) const = 0;

  /// Release a session
  virtual void releaseSession( const Token sessionID ) const = 0;
  [...]

  /// Create a new cursor for the given query and return its cursor ID.
  virtual const Token executeQuery( const Token sessionID,
                                    const IPortableQueryDefinition& qd ) const = 0;

  /// Release a cursor
  virtual void releaseCursor( const Token cursorID ) const = 0;

  /// Fetch a new bulk of rows from the given cursor.
  virtual const std::vector<coral::AttributeList> fetchRows( const Token sessionID,
                                                             const Token cursorID,
                                                             const unsigned int maxSize ) const = 0;

  /// Fetch all rows from the cursor.
  virtual const std::vector<coral::AttributeList> fetchAllRows( const Token sessionID,
                                                                const IPortableQueryDefinition& qd ) const = 0;
  [...]
};
```

Package 1 – interfaces

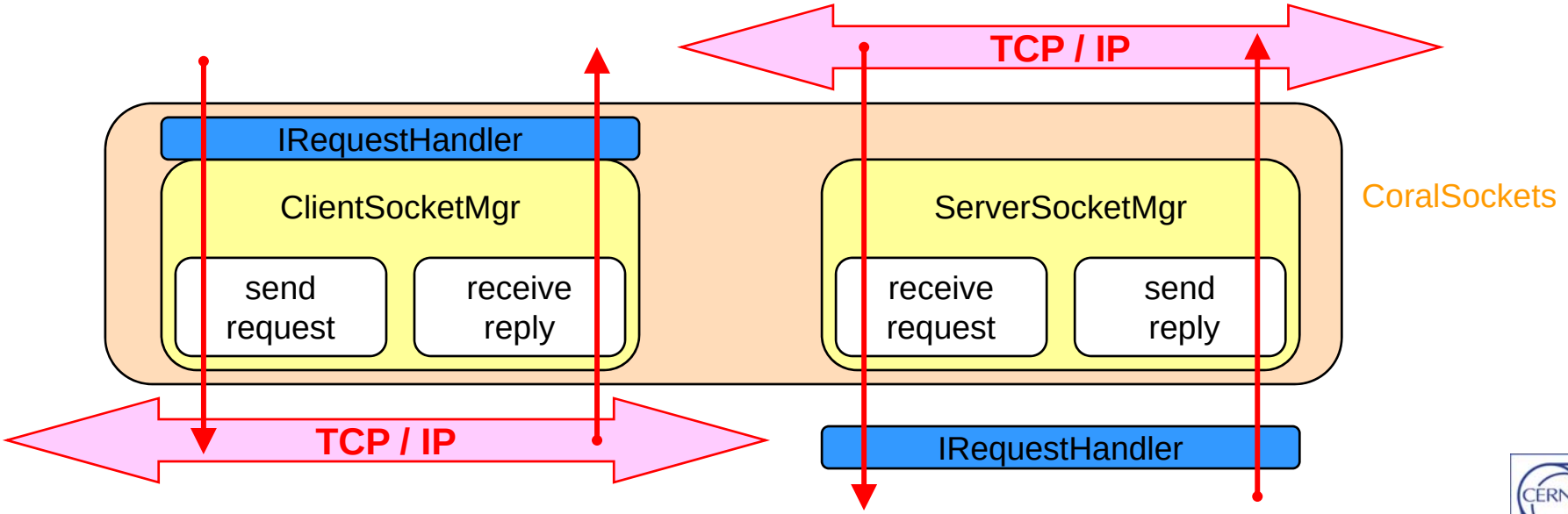
CERN IT Department

- Proposed team: ALL (need stable interfaces)
  - ICoralFacade: mainly Andrea, Alex
  - IRequestHandler: mainly Martin, Andy, Alex
- Standalone tests of the CoralServerBase package?
  - Very few tests needed (mainly an interface definition package)
- Possibilities for reuse of existing code
  - Interfaces from the old prototype following this design
  - Limited use of abstract interfaces in the present code

Package 2 – sockets (+threads)

- *In my diagram:* ***CoralSockets***

- Client-side: ClientSocketMgr
  - Implements IRequestHandler, sends requests over TCP/IP
- Server-side: ServerSocketMgr
  - Receives requests via TCP/IP, delegates to an IRequestHandler



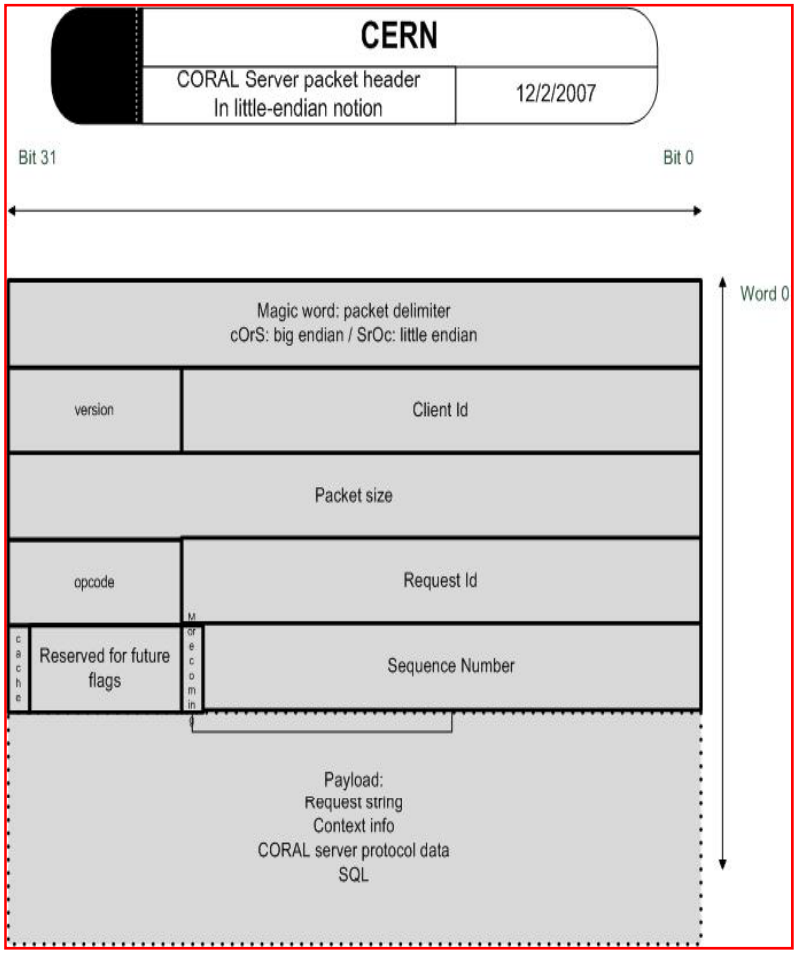
Package 2 – sockets (+threads)

CERN IT Department

- Proposed team: Martin, Andy
  - Can we use the same components for server/client and proxy?!
  - Any special requirements for chaining of proxies?

- Exchange packets without assuming relational content
  - First header for this package only needs low level TCP/IP protocol
    - endian, version, clientID (for proxy), packet size, requestID (, seq#?)
  - A string payload with relational content should start with a second nested header containing the relational metadata, but this package does not need to be aware of this
    - opcode, cacheable flag (, relational version?) (, sessionID?)
    - followed by relational payload
  - Suggest reordering of packet header to simplify IRequestHandler

Package 2 – sockets (+threads)



~CTL ***~CAL***

[SocketHeader [SocketPayload]]]
= [SocketHeader [RelationalHeader [RelationalPayload]]]

SocketHeader
= (MagicWord | Version | ClientID? | Size | RequestID | Seq#?)

RelationalHeader
= (Version2? | Opcode for ICoralFacade method | Flags)

RelationalPayload
= (streaming of ICoralFacade method arguments)

~CAL

Package 2 – sockets (+threads)

CERN IT Department

- User of socket tells the socket:
  - Send this string payload
    - This may encapsulate relational data (relational opcode, cacheable flag, relational payload), but the socket does not need to know that…
- User of socket does not tell the socket:
  - The transport-layer packet metadata assigned by the socket
    - Endianness of socket (? use fixed endianness instead ?)
    - Version of socket protocol header
    - Total size
    - RequestID
    - ClientID (?)
    - Sequence# (not needed?)
  - User of the socket does not need to know the above transport-layer metadata (e.g. for relational processing)

Package 2 – sockets (+threads)

CERN IT Department

- Threads in the client
  - Several client threads can use the same ClientSocketMgr
- Threads in the server
  - One main listener thread in the ServerSocketMgr
  - One socket thread per established client socket
  - Several handler threads per socket thread
    - Asynchronous handling of requests on each client socket
    - Each handler has/is an IRequestHandler
    - ServerSocketMgr instantiated from an IRequestHandlerFactory to create handlers (in CORAL Server, all these handlers use same ICoralFacade)
- Eventually: add SSL authentication via certificates
  - Authenticate with the CORAL Server before sending any message
  - Unrelated to remote DB authentication or relational content

Package 2 – sockets (+threads)

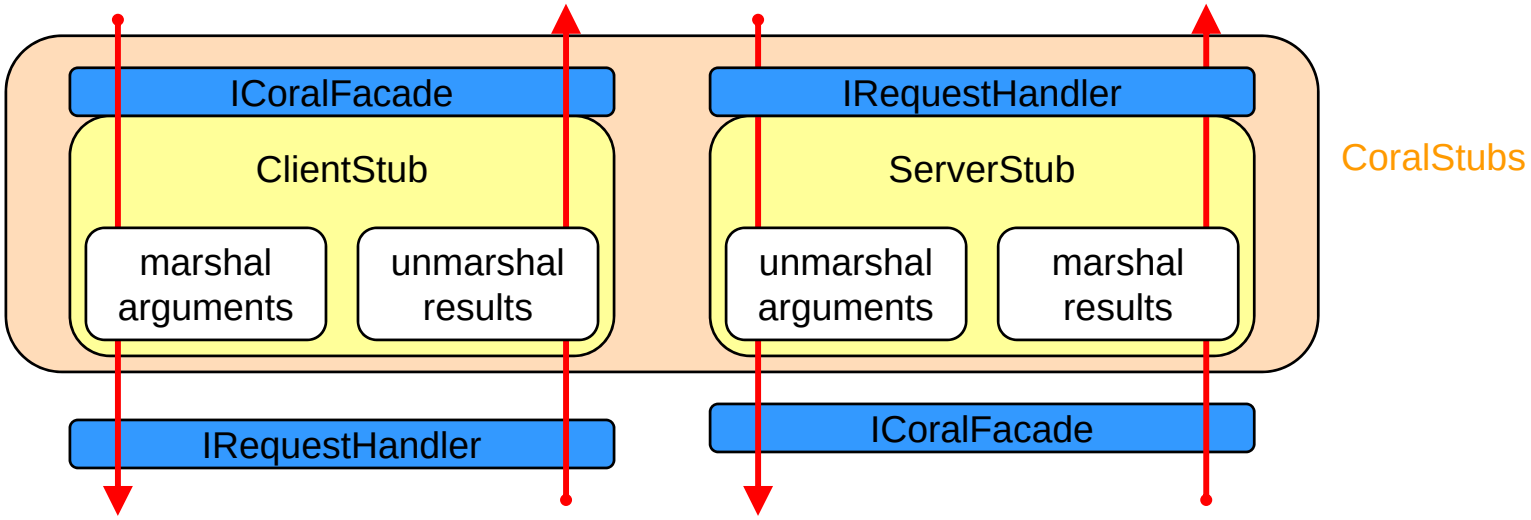
CERN IT Department

- Standalone tests of the CoralSockets package
  - Use a ‘dummy’ test request handler that returns a reply string which is the same as the request string plus some additional characters
  - Check that the client gets the same results through a local dummy handler or a remote dummy handler connected via sockets
  - Several single and multi client examples:
    - Send all 256 possible characters
    - Send very long strings to check segmentation
    - Send numbered asynchronous requests from several client threads, test that the server sends the right reply to the right client
- Possibilities for reuse of existing code
  - Socket-related state machines
  - Reactor/Acceptor pattern

Package 3 – encoding/decoding

- *In my diagram:* ***CoralStubs***

- Client-side: ClientStub
  - Implements ICoralFacade, delegates to an IRequestHandler
- Server-side: ServerStub
  - Implements IRequestHandler, delegates to an ICoralFacade



Package 3 – encoding/decoding

CERN IT Department

- Proposed team: Alex
- Encode as string: relational header plus relational payload
  - Apart from the header, encoding syntax fully encapsulated here
  - System design should avoid need to publish all opcodes
- Exception handling is also part of this package
- Possibilities for reuse of existing code
  - Template-based Boost generator and other encoding mechanisms
  - Exception handling infrastructure

Package 3 – encoding/decoding

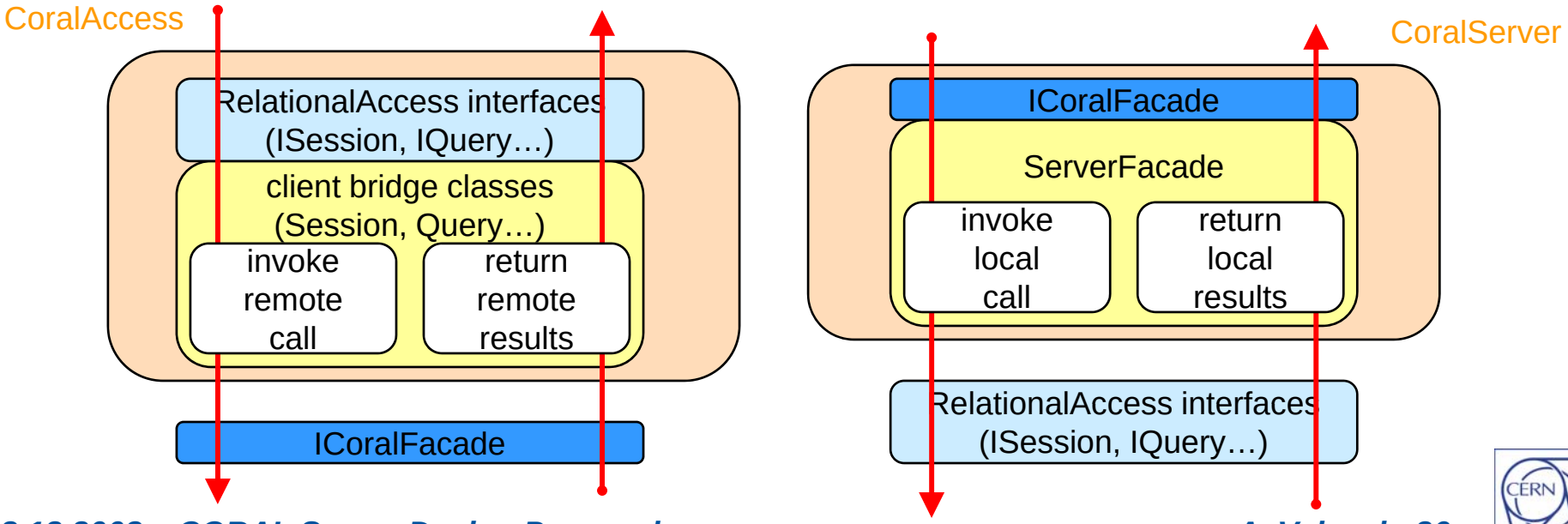
CERN IT Department

- Standalone tests of the CoralStubs package
  - Process test string, decode, encode back, compare to original
  - Process test arguments, encode, decode back, compare to original
  - Cover also exception throw/catch
  - Cover both requests and replies (unit tests of subcomponents)

Packages 4a/4b – relational access

- *In my diagram:* ***CoralAccess, CoralServer***

- Client-side: bridge classes Session, Query…
  - Implement ISession, IQuery… , delegate to an ICoralFacade
- Server-side: ServerFacade
  - Implements ICoralFacade, delegates to ISession, IQuery…



Packages 4a/4b – relational access

CERN IT Department

- Proposed team: Andrea, Alex
- ServerFacade is a CORAL application (ConnectionService)
  - In the general stateful case, an object store registers and looks up sessions, cursors and bulk operations by their token/ID
  - In the stateless case, a single DB session can be opened and no open cursors or bulk operations are allowed
- Remote DB authentication is done by ServerFacade
  - Connection state to remote DB is controlled here

Packages 4a/4b – relational access

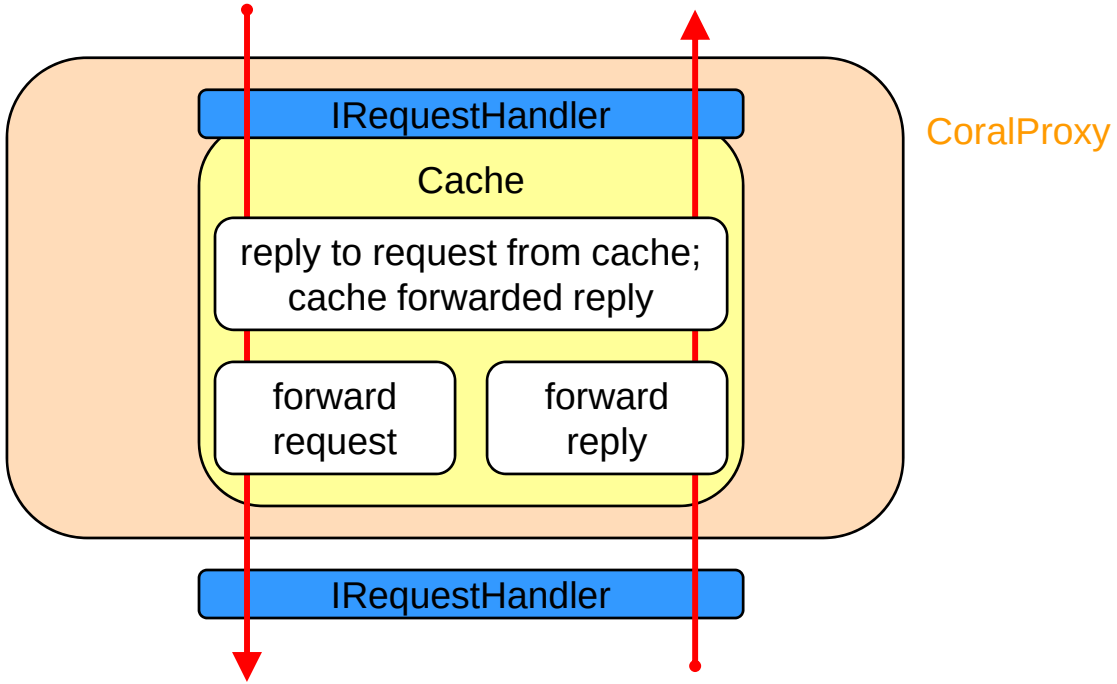
CERN IT Department

- Standalone tests of the CoralAccess/CoralServer packages
  - Special ‘local’ connection option of CoralAccess: bridge classes delegate directly to a local ServerFacade without encoding/decoding and/or without packet exchange over sockets
  - Load ServerFacade as a plugin if possible to avoid direct linking
  - Any read-only (and eventually read-write) CORAL test can be used
- Possibilities for reuse of existing code
  - Present client code derived from old prototype following this design
  - Resurrect server code from old prototype

Package 5 – proxy cache

- *In my diagram:* ***CoralProxy***

- Proxy cache engine: class Cache
  - Implements IRequestHandler, delegates to an IRequestHandler
  - Filters requests/replies: either reply to requests from cache, or forward them and cache replies for next execution of same request



Package 5 – proxy cache

CERN IT Department

- Proposed team: Andy
- Should look at cacheable flag only if possible
  - Which opcodes need a special handling and why?
    - Database connect/disconnect
    - Transaction start/commit/rollback
  - System design should avoid need to publish all opcodes if possible
    - e.g. if startTransaction is a special case of ‘droppable’ packet, add a droppable flag instead, or avoid packet reaching the proxy?
- Any special need for chaining of proxies?

Package 5 – proxy cache

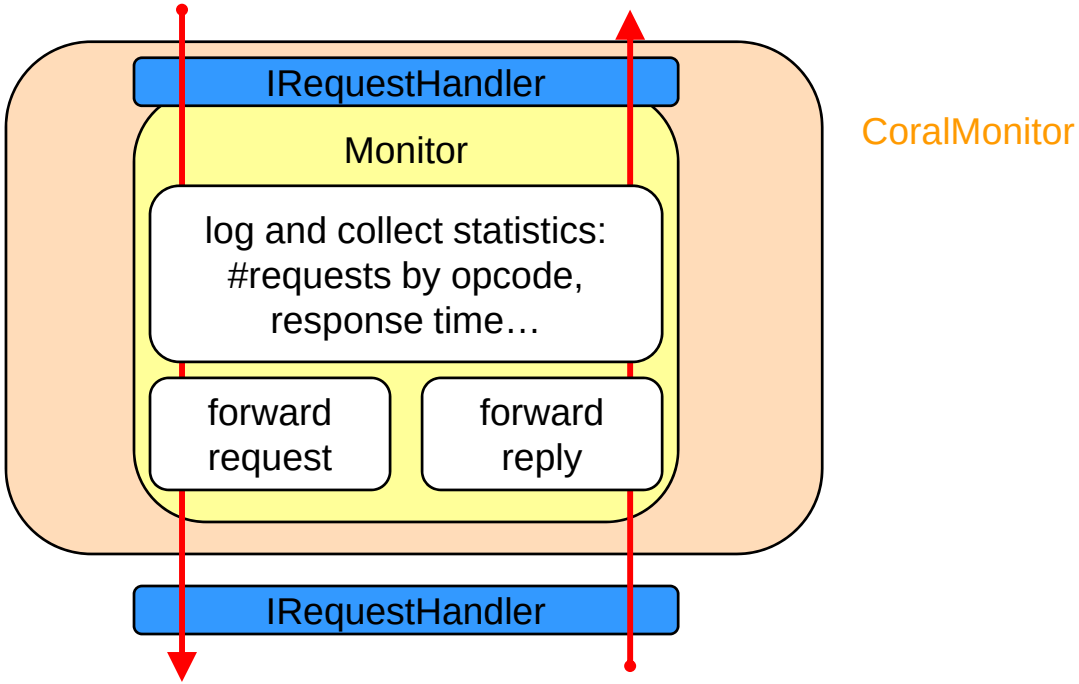
CERN IT Department

- Standalone tests
  - Attach to a test IRequestHandler that always gives different replies to the same request: test that reply from cache does not change after the first time if the cacheable flag is set
- Possibilities for reuse of existing code
  - Internal cache implementation of present proxy

Package 6 – monitoring

- *In my diagram:* ***CoralMonitor***

- Same class for client, server or proxy: Monitor
  - Implements IRequestHandler, delegates to an IRequestHandler
  - Filters requests/replies: logs statistics while forwarding them



Package 6 – monitoring

CERN IT Department

- Proposed team: Andy, Alex
- Possibilities for reuse of existing code
  - Requirements and implementation to be based on present proxy
  - Experience and code from present server monitoring tool
- Standalone tests
  - Attach to a test request handler
  - Check (against references) log files for well defined test loads

Package dependencies

- Most libraries depend on CoralServerBase interfaces only
  - All components implement or use ICoralFacade or IRequestHandler
- CoralAccess needs concrete ICoralFacade implementations
  - ClientStub: link CoralStubs and CoralSockets if necessary
  - ServerFacade for tests: load as a plugin if possible
- Tests and executables need concrete implementations
  - Additional CMT dependencies only for tests and executables
  - Executable code can be an extremely simple chain of components
    - ServerSocketMgr(ServerStub(ServerFacade(ConnectionService())))

Intermediate alternatives

CERN IT Department

- Keep present proxy as standalone executable
  - More difficult to share a monitoring component
  - Duplication of code in the handling of sockets
- Keep present packet header
  - Add some relational metadata to the IRequestHandler interface