Service-Oriented Design in a Large Plex Project

Customer Case

Contents

- Speaker and Project
- SOA Interpretation and Focus
- Separate Development Models Accessed through Interface Model
- Stable Interfaces by Restriction of Parameters
- Stable Interfaces by Versions
- Transactions and Server-Side Validation

SPEAKER AND PROJECT

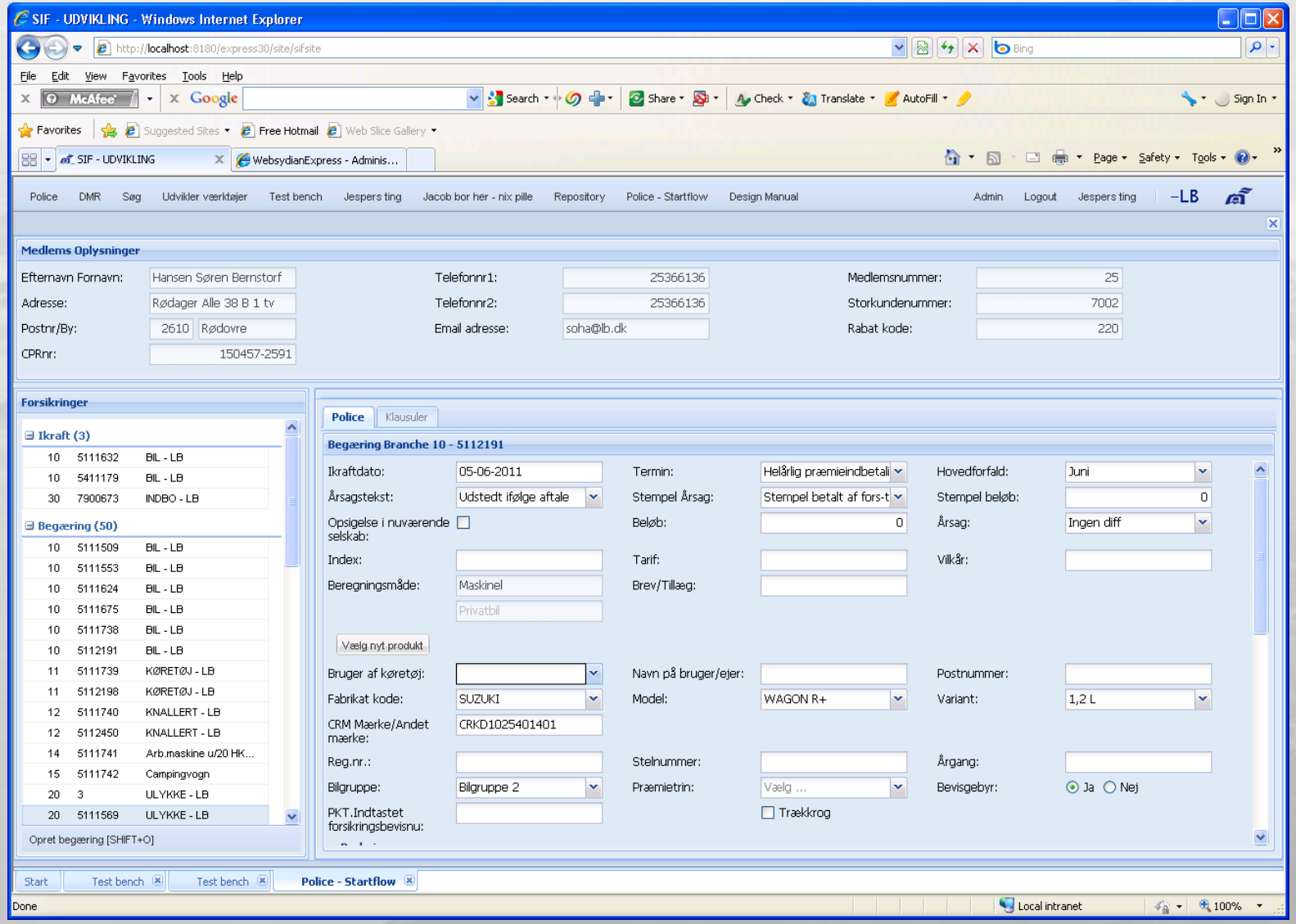
About the Speaker

- Morten Knudsen
- Danish Post IT Department
- M.Sc. Computer Science
- Zurich Insurance
- Soft Design (Websydian Development)
- KODA – Head of IT development
- Various…
- Soft Design (Project leader, Consultant)

About the SIF Project

- Insurance application built from scratch using Plex
- 7 Plex development models
- 20 Plex developers
- SOA approach – focus on server functionality
- Online synchronization with existing system
- Soft Design is a sub-contractor (5-7 consultants)

SIF: Websydian, Ext-js, Ajax, Java



Soft Design

casmart

SOA INTERPRETATION AND FOCUS

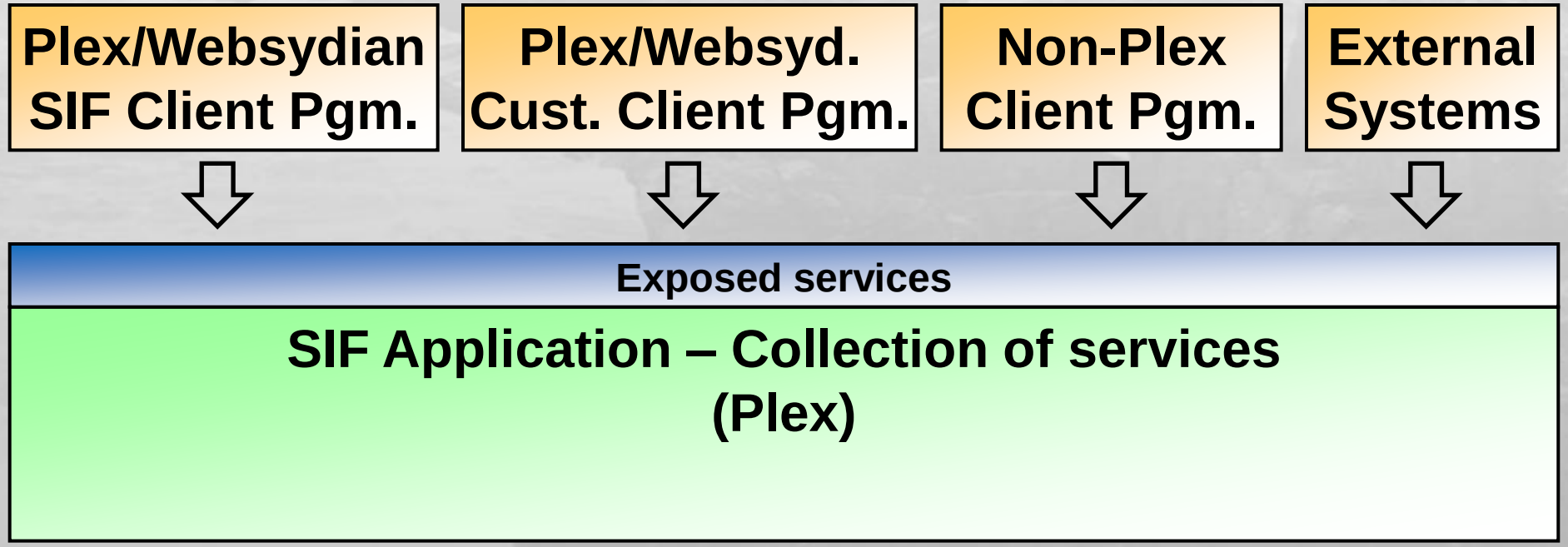
SOA Definition

- *"SOA establishes an architectural model that aims to enhance the efficiency, agility, and productivity of an enterprise* ***by positioning services as the primary means through which solution logic is represented*** *in support of the realization of the strategic goals associated with service-oriented computing."*

  Thomas Erl (http://www.whatissoa.com)

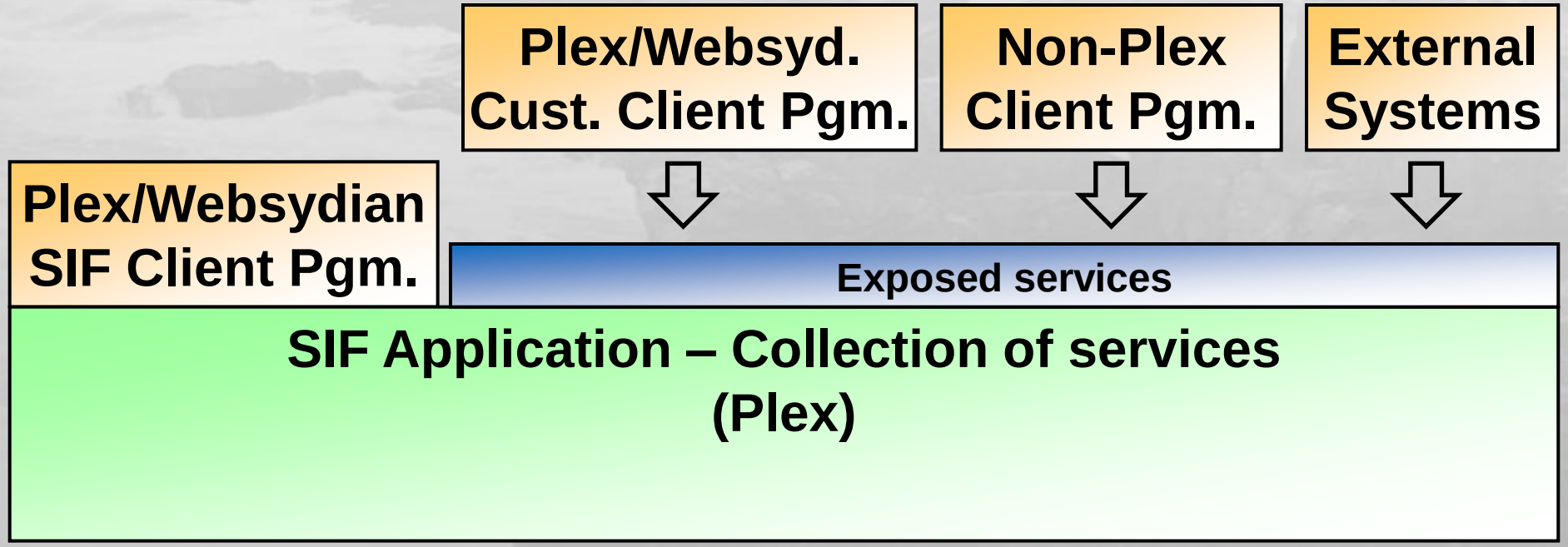
SIF – Collection of Services

- Collection of services accessed from client programs and external systems



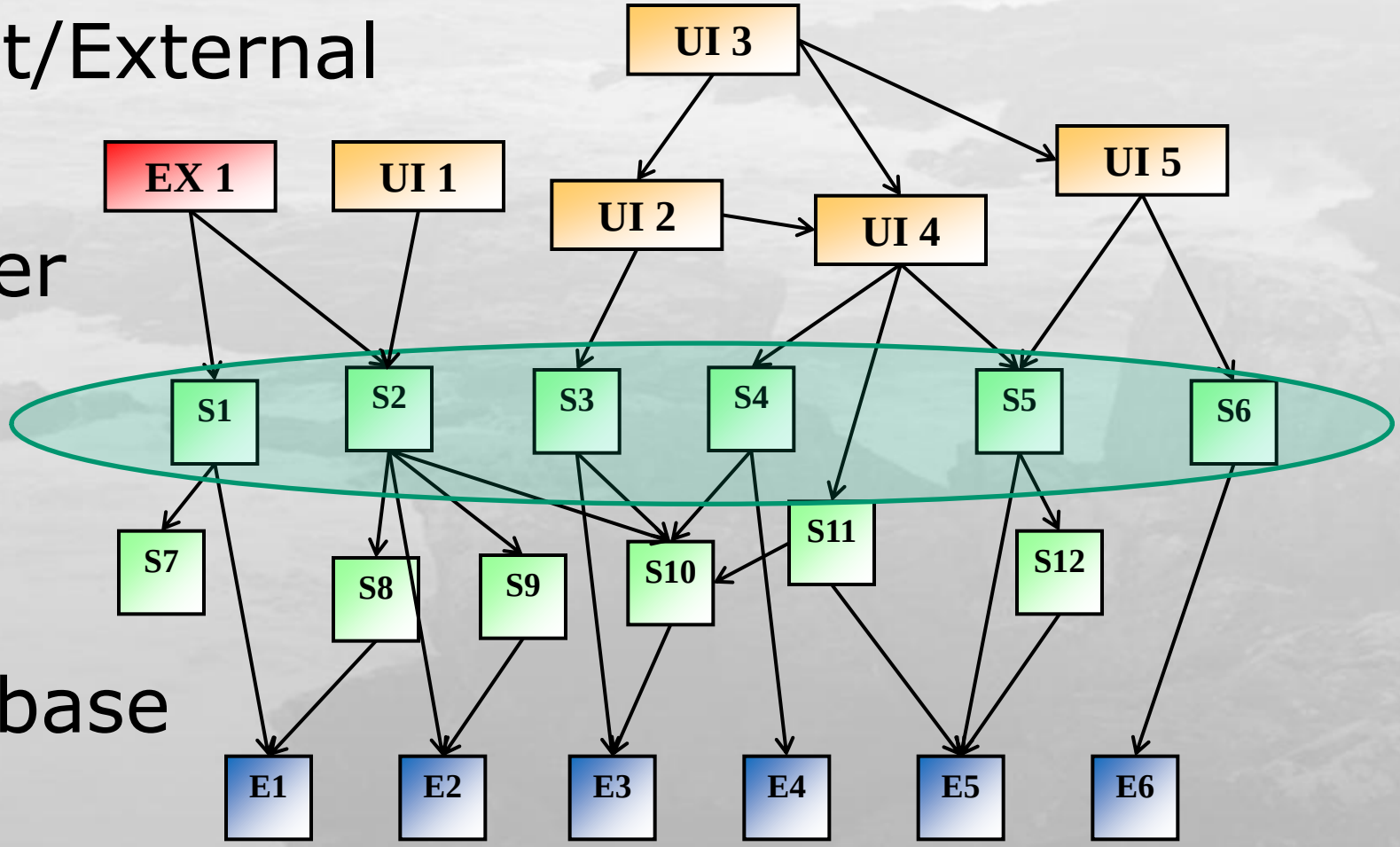
SIF – Collection of Services

- Pragmatic approach taken in implemented system



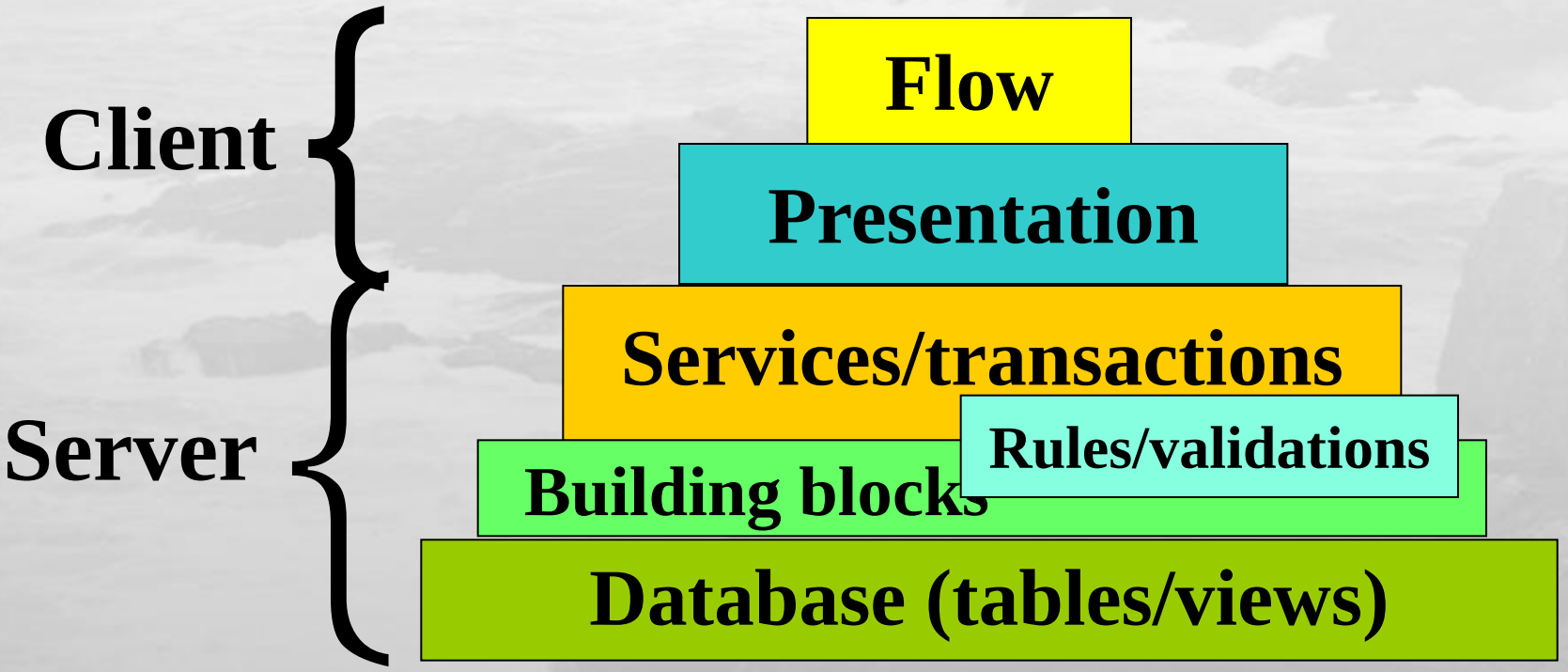
Services in SOA Architecture

- Client/External
- Server
- Database

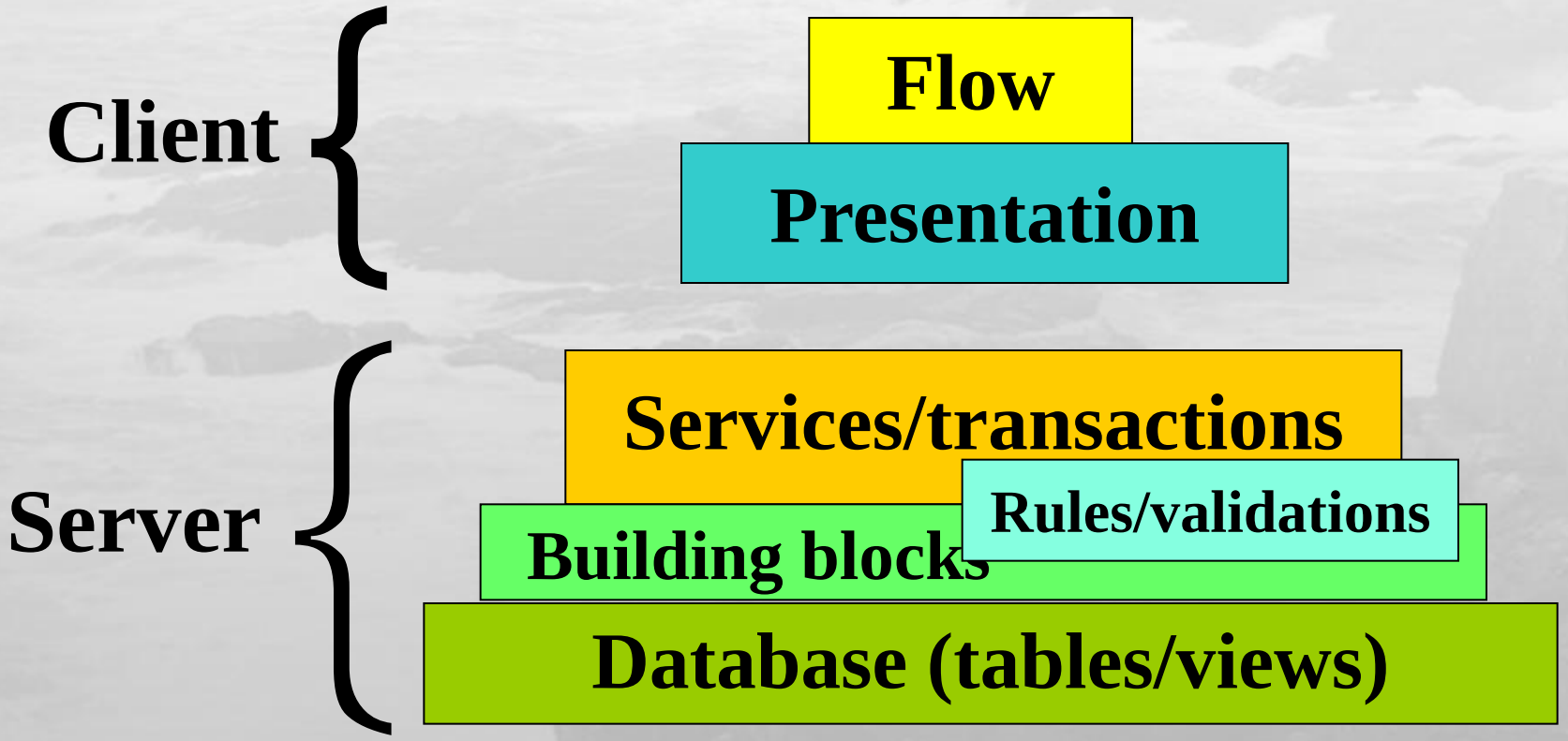


SIF application regarded as set of services

SIF Application Architecture



Decoupling the Role of the Client and the Server



SOA Focus in SIF Project

- From Plex we get:
  - Modularization
    - Single-view access for each function
    - Alternative to portions of inline code
  - Focus on interfaces, not implementation
  - Expose selected programs as web services (TransacXML)

Plex is also a great tool for organization, reuse and documentation of code (but this is not particularly SOA)

SOA Focus in SIF Project

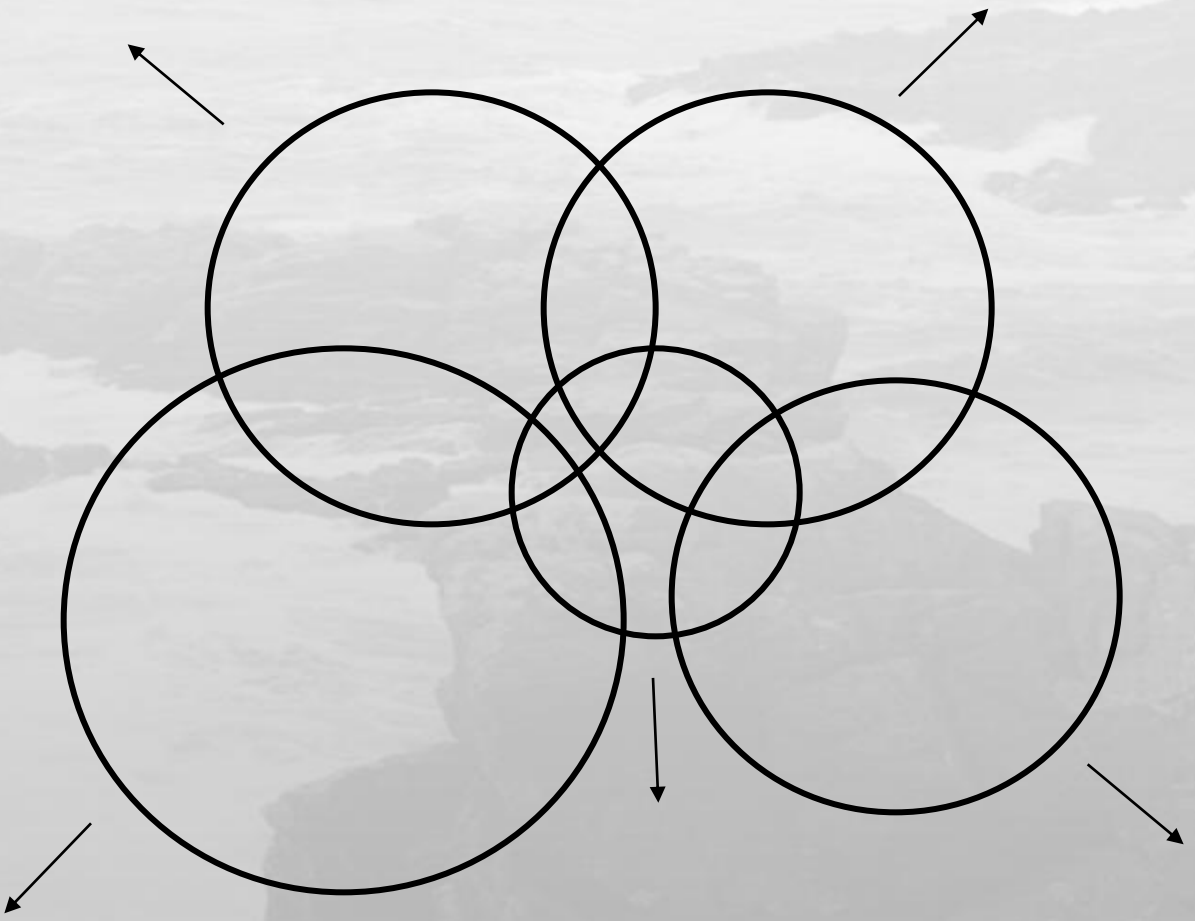
- Additional focus on stable interfaces
  - Further modularization
  - Business focus
  - Test bench applied for services…
  - Further stabilisation of function interfaces by restriction of parameters
  - Versions of services
  - Responsibility and ownership
  - Statefull versus stateless…

4 SOA-Related Issues in This Presentation

- 1) Interfaces
  - Separate development models accessed through **interface** model
- 2) Interfaces
  - Stable **interfaces** by versions
- 3) Interfaces
  - Stable **interfaces** by restriction of parameters
- 4) Interfaces
  - Server-side validation providing a single **interface** to main transactions comprising validation rules

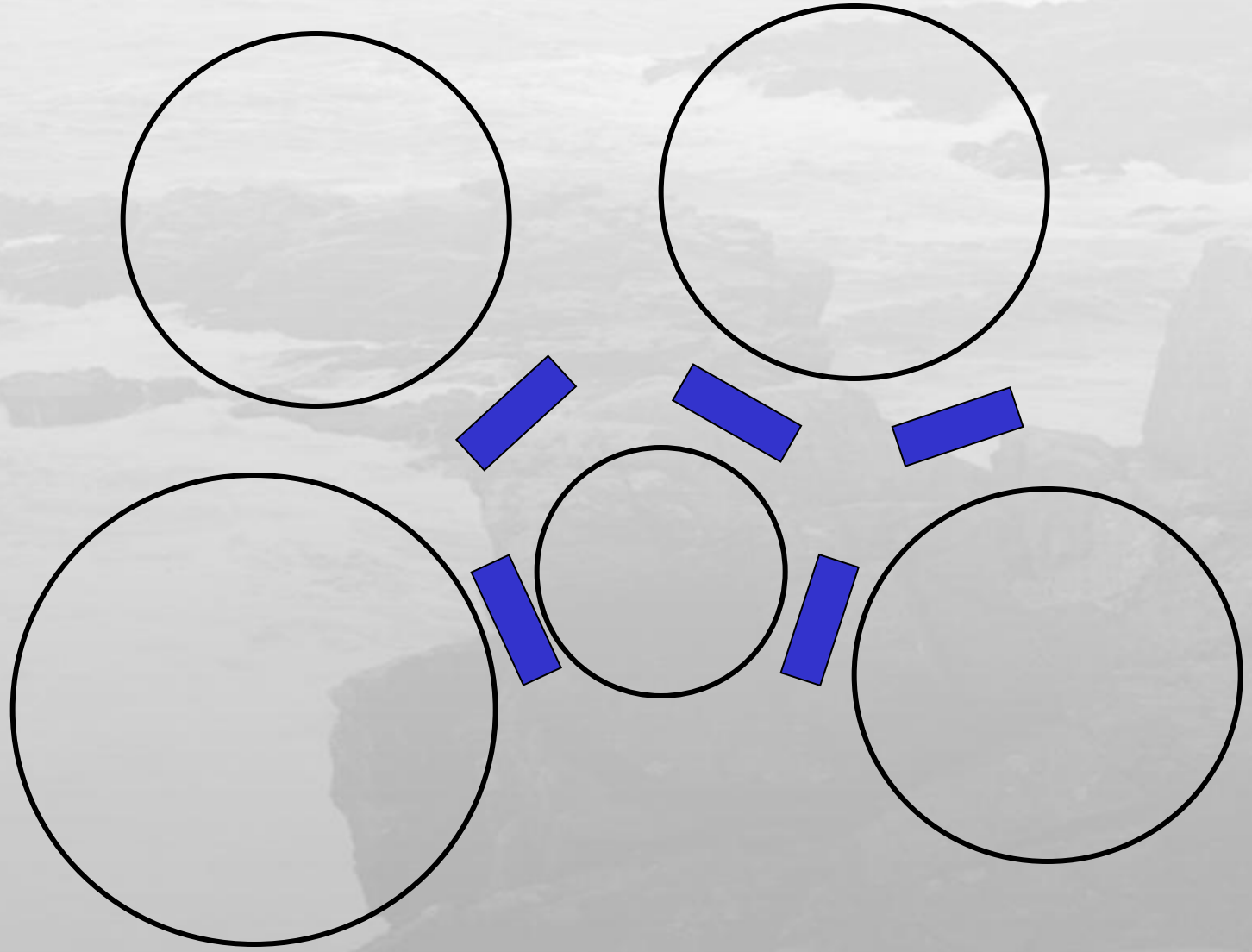
1) SEPARATE DEVELOPMENT MODELS ACCESSED THROUGH INTERFACE MODEL

SOA – Separate Systems

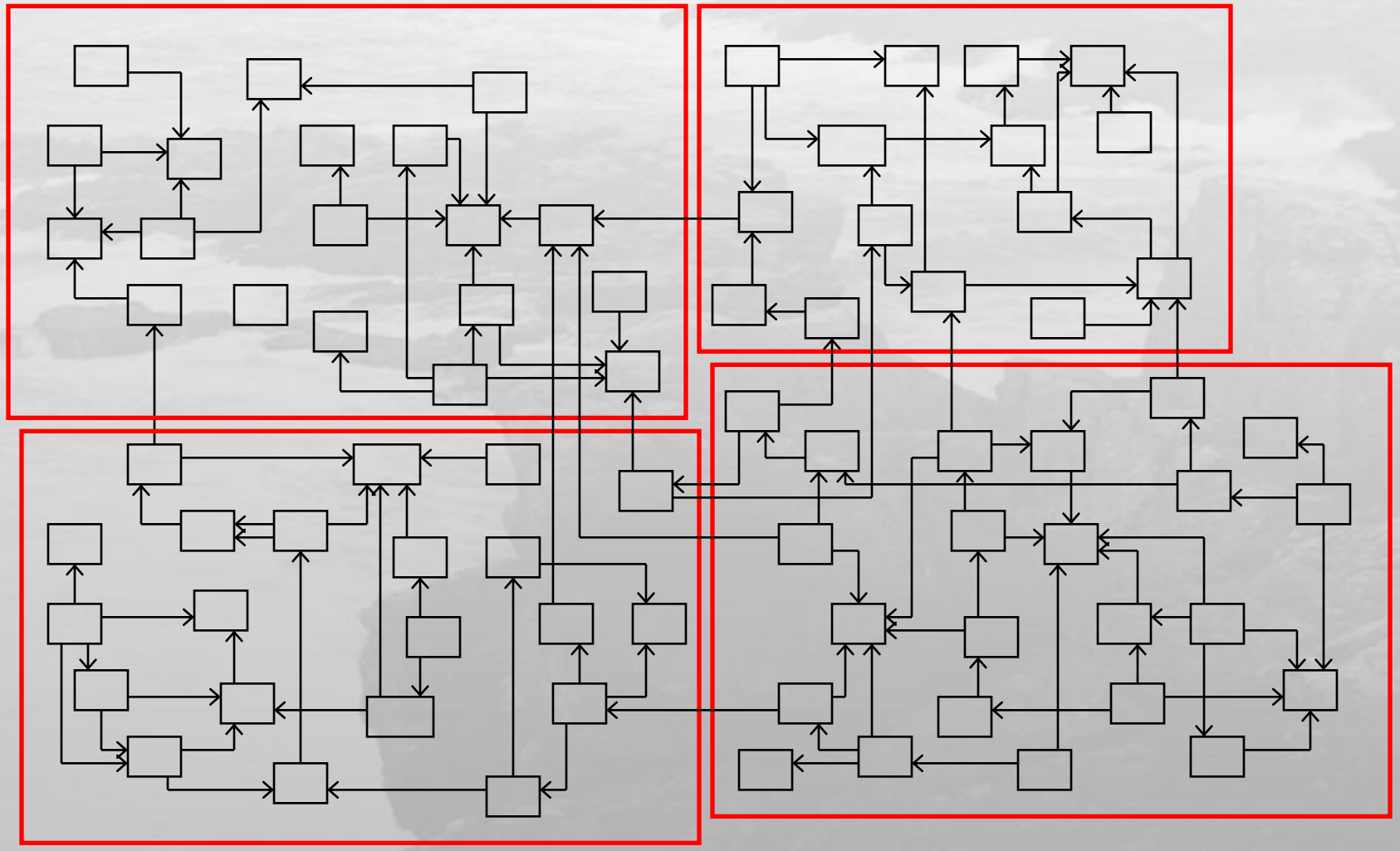


Principle applies for large-scale applications as well as programs and components

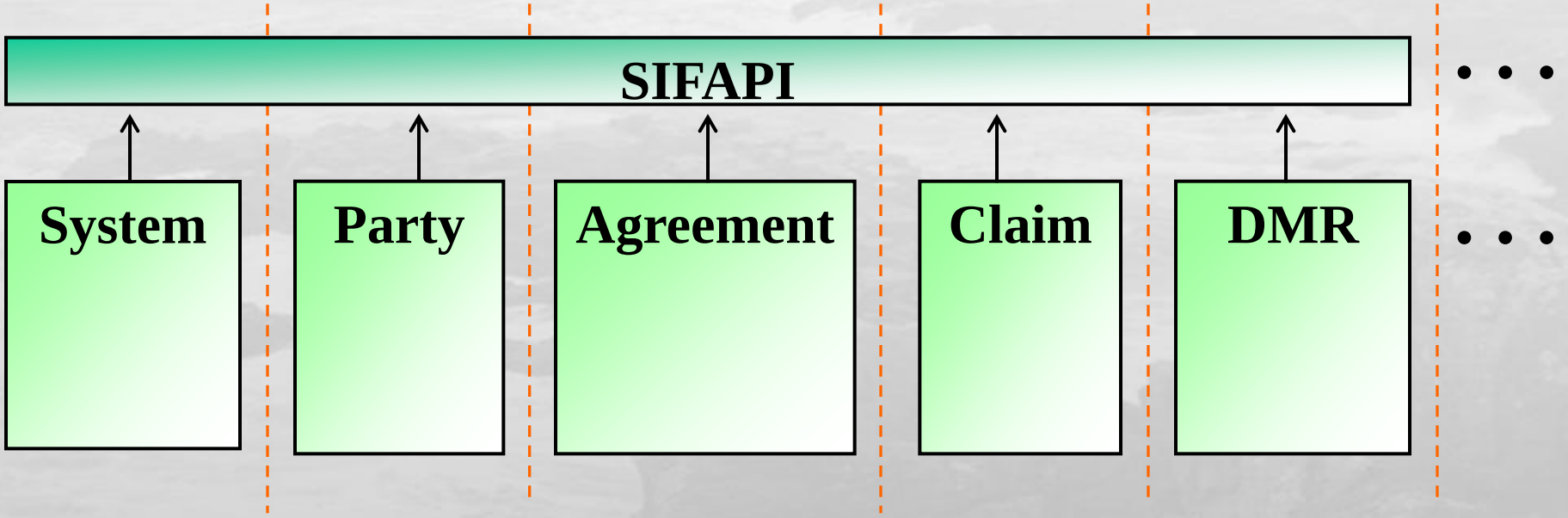
SOA – Establish Interfaces

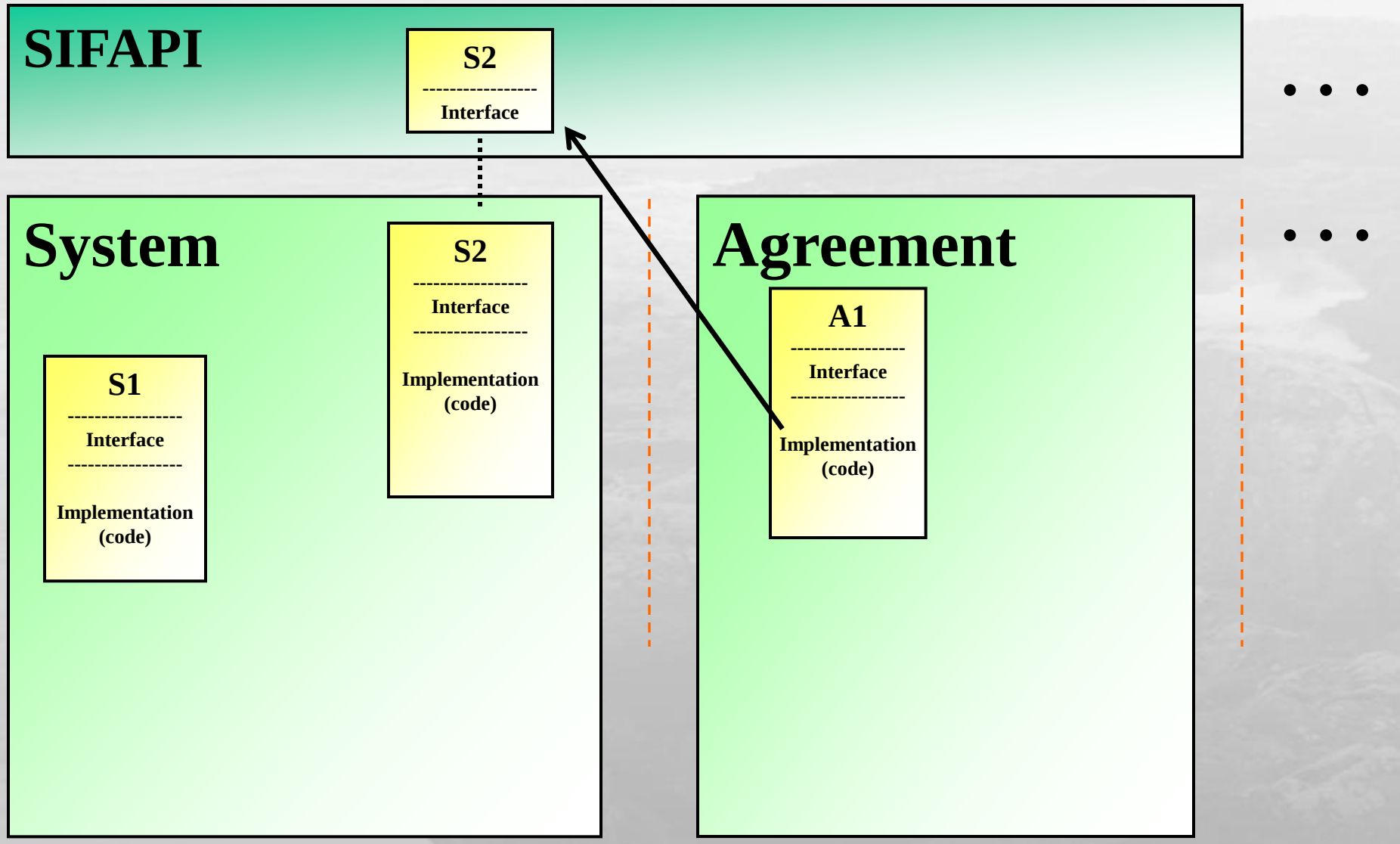


Split Application Database into Separate Models



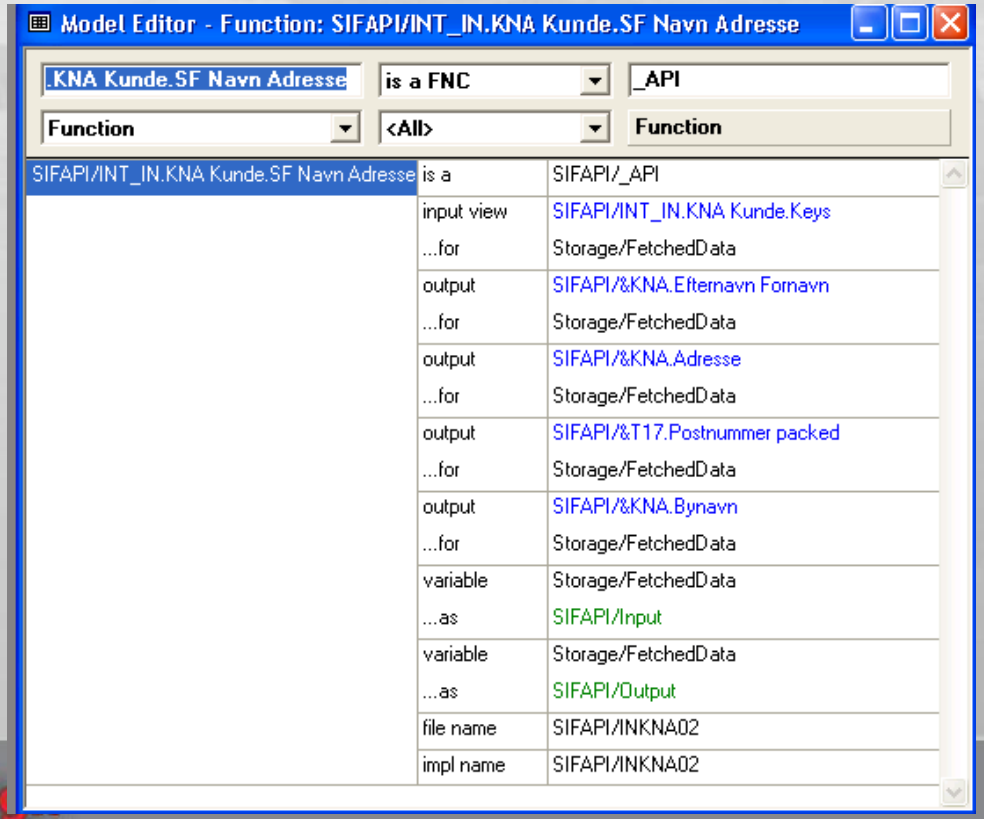
Plex Models

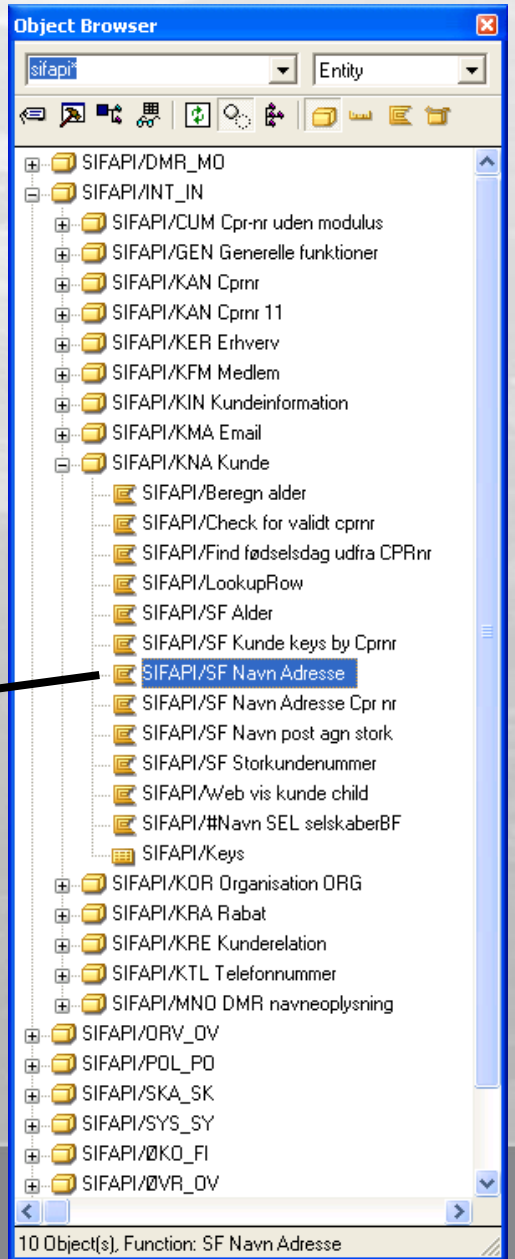


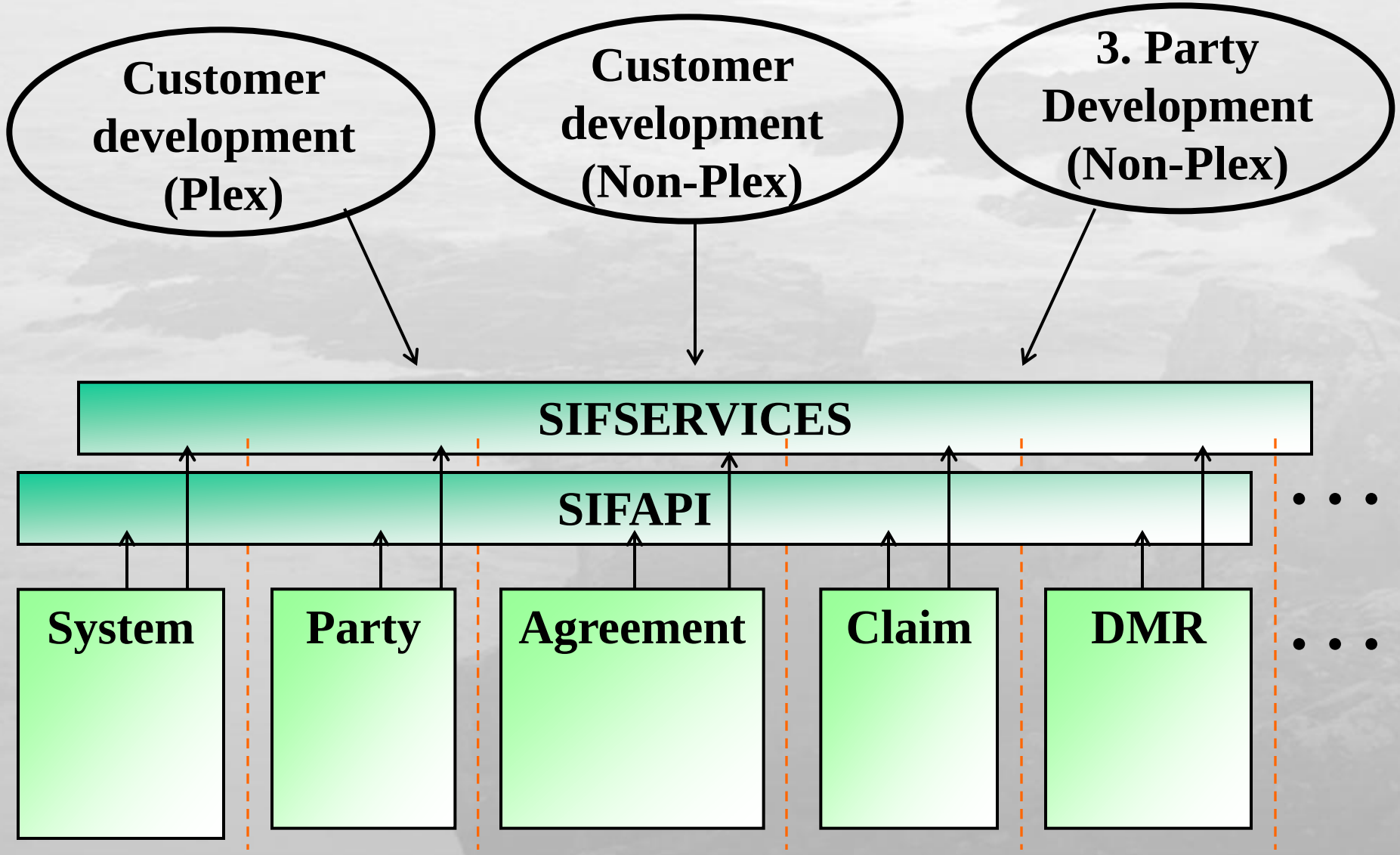

SIFAPI
S2
Interface
System
S2
Interface
Implementation
(code)
S1
Interface
Implementation
(code)
Agreement
A1
Interface
Implementation
(code)

API Model as Service Catalogue

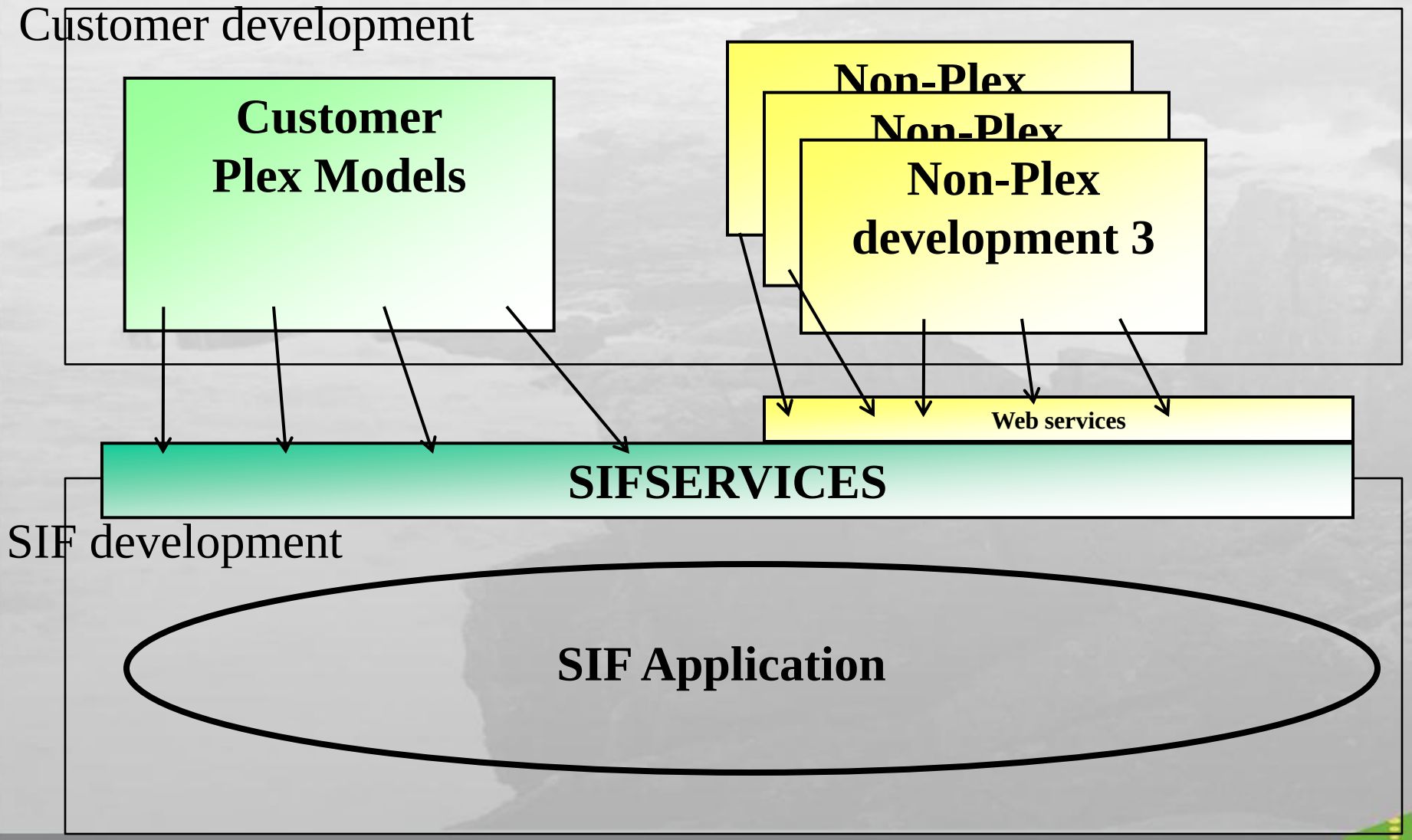
- *SIFAPI* model contains only interface specification






Customer development (Plex)
Customer development (Non-Plex)
3. Party Development (Non-Plex)
SIFSERVICES
SIFAPI
System
Party
Agreement
Claim
DMR

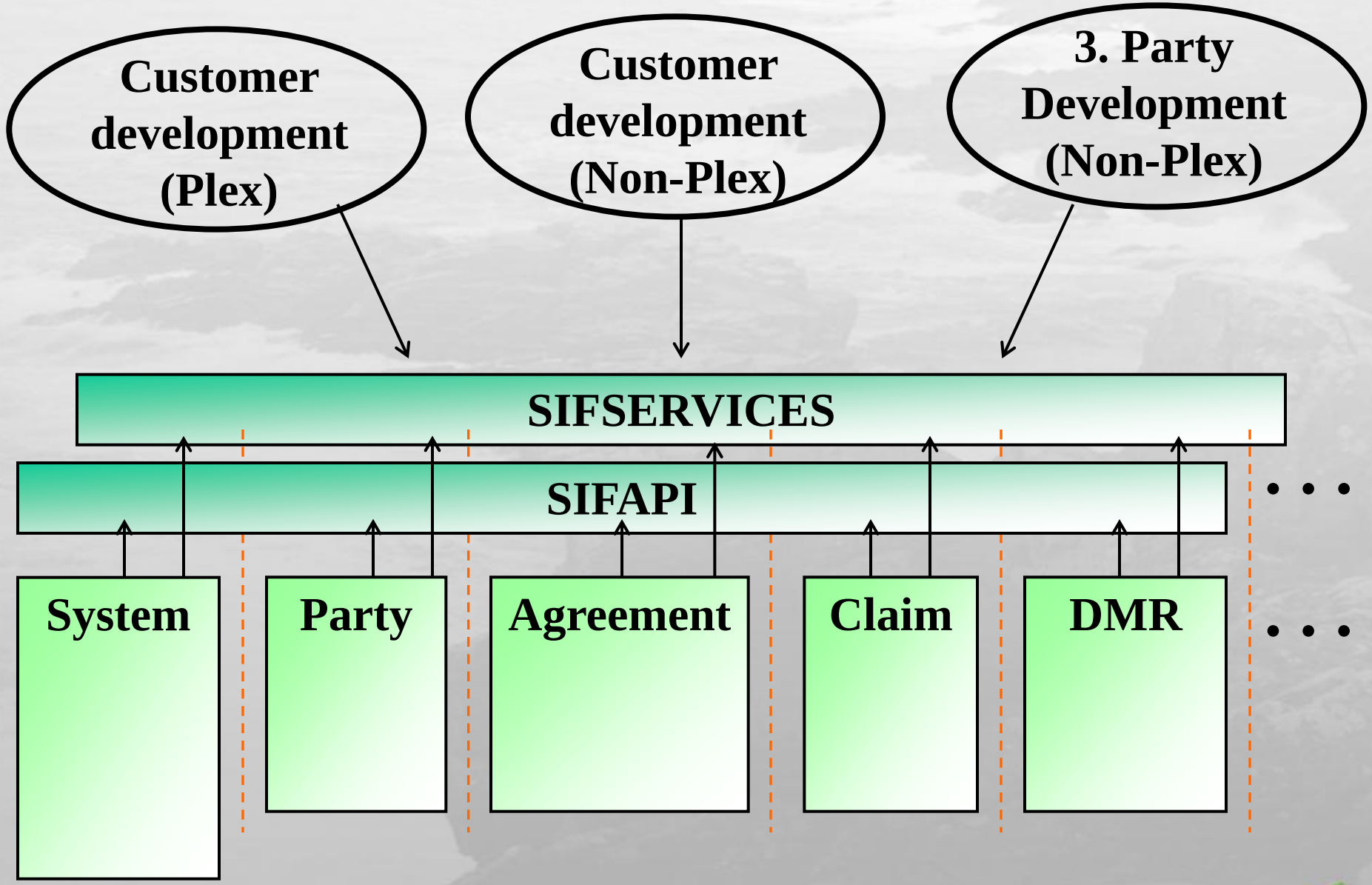
Customer Access to SIF Services



Purpose of Splitting

- SOA-principles applied to internal application structure
- Ownership and responsibility
- Possibility to replace model/subsystem
- Simple API Plex model handed over to Customer

2) STABLE INTERFACES BY VERSIONS

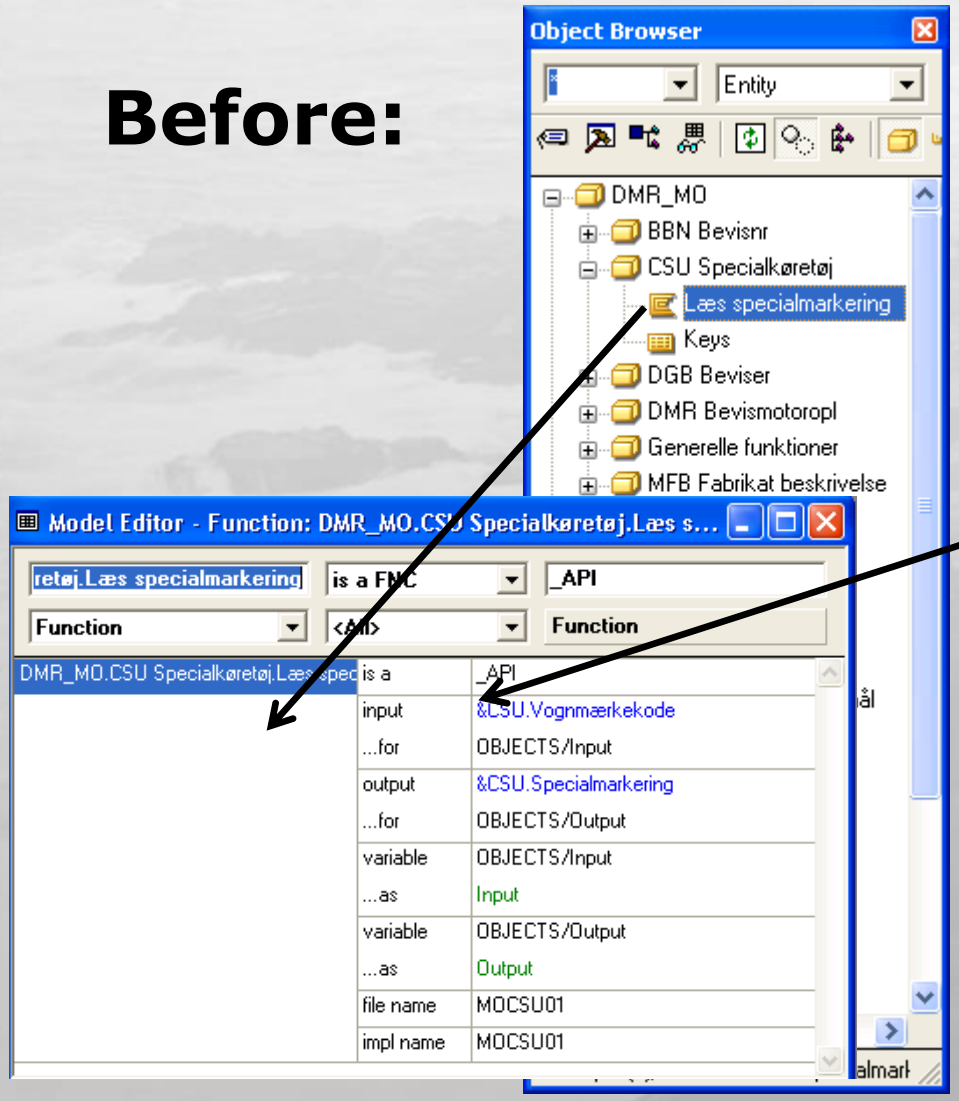

Customer development (Plex)
Customer development (Non-Plex)
3. Party Development (Non-Plex)
SIFSERVICES
SIFAPI
System
Party
Agreement
Claim
DMR

Do Not Change Service Interfaces

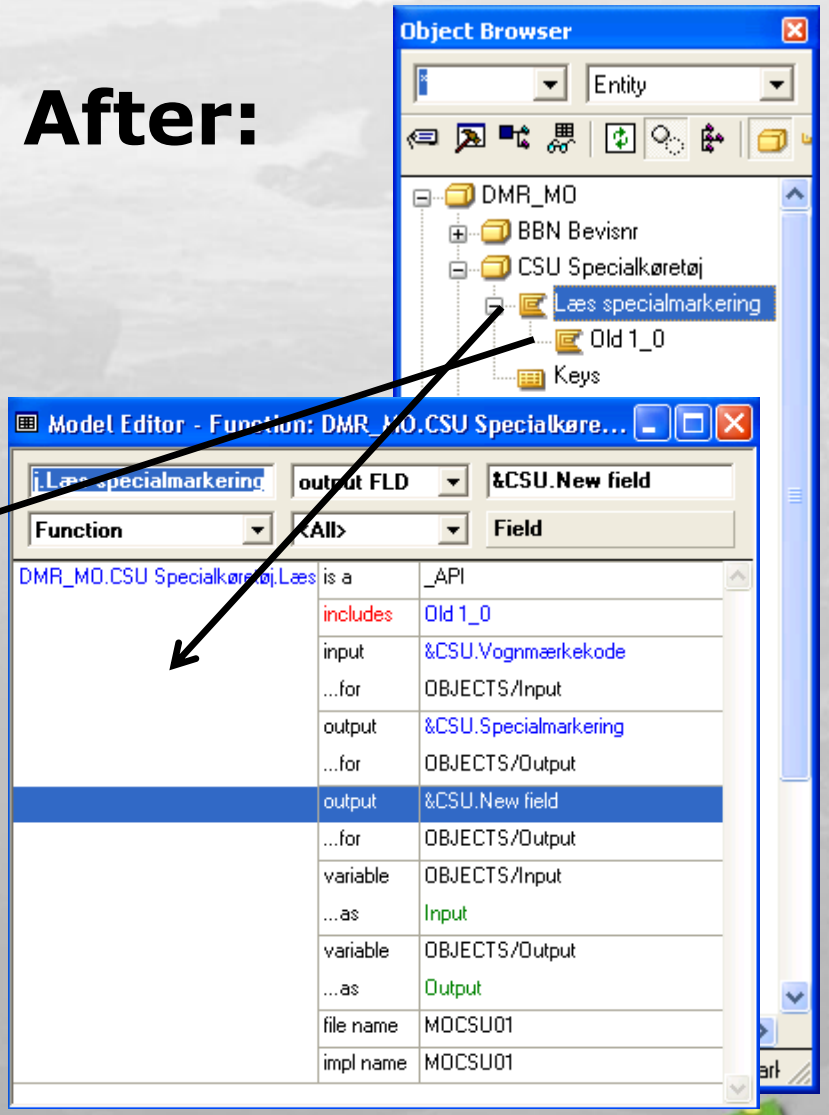
- First published, Parameter interfaces of services in catalogue (*SIFERVICES*) must be stable
  - Internal logic may be modified/corrected
- Define new version of service
  - Create and 'publish' new service (function)
  - Calling functions may shift to new version
  - Existing service remain stable

New Version of a Service

Before:



After:



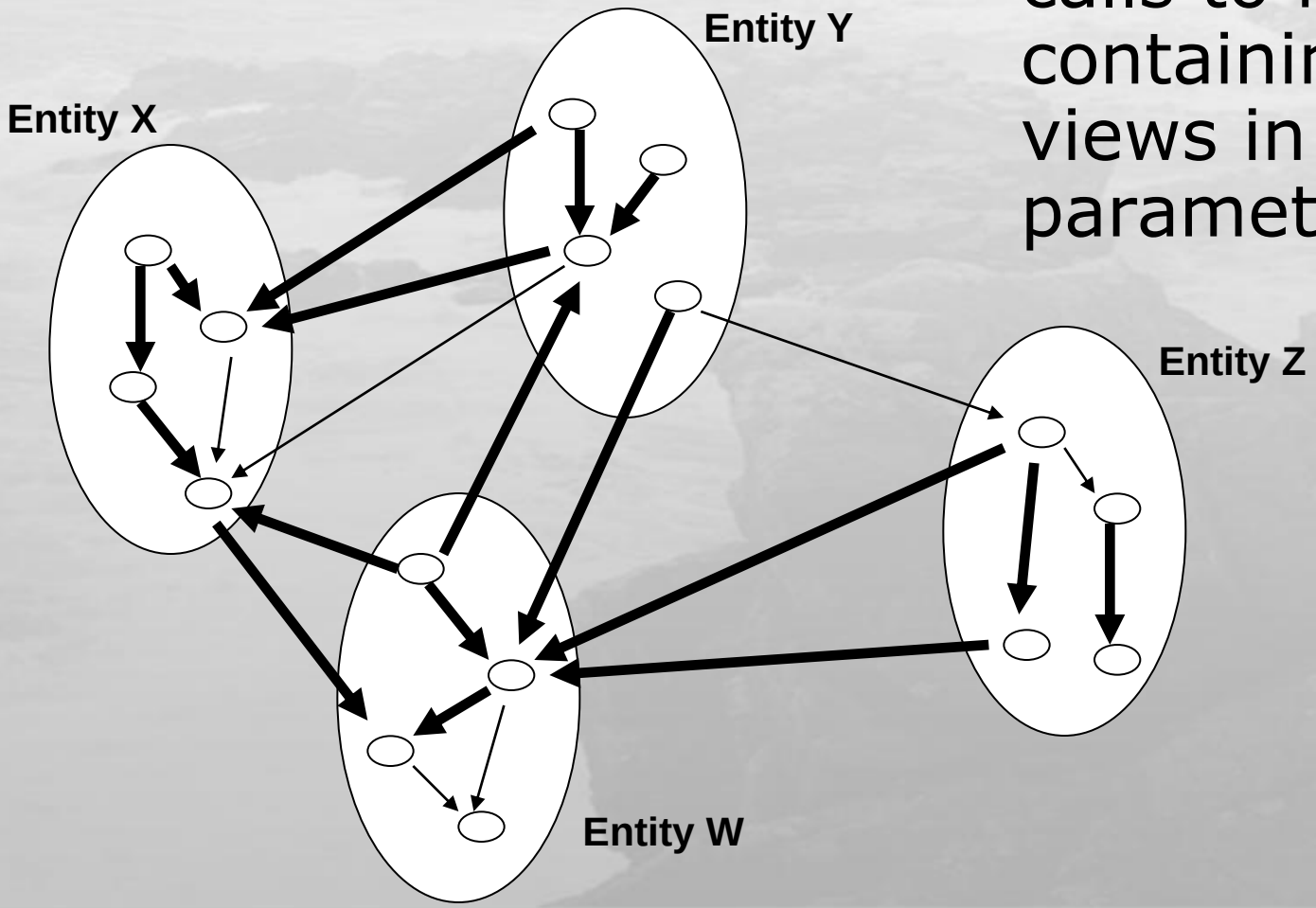
3) STABLE INTERFACES BY RESTRICTION OF PARAMETERS

Use of Views in Parameter Lists of Abstract Functions

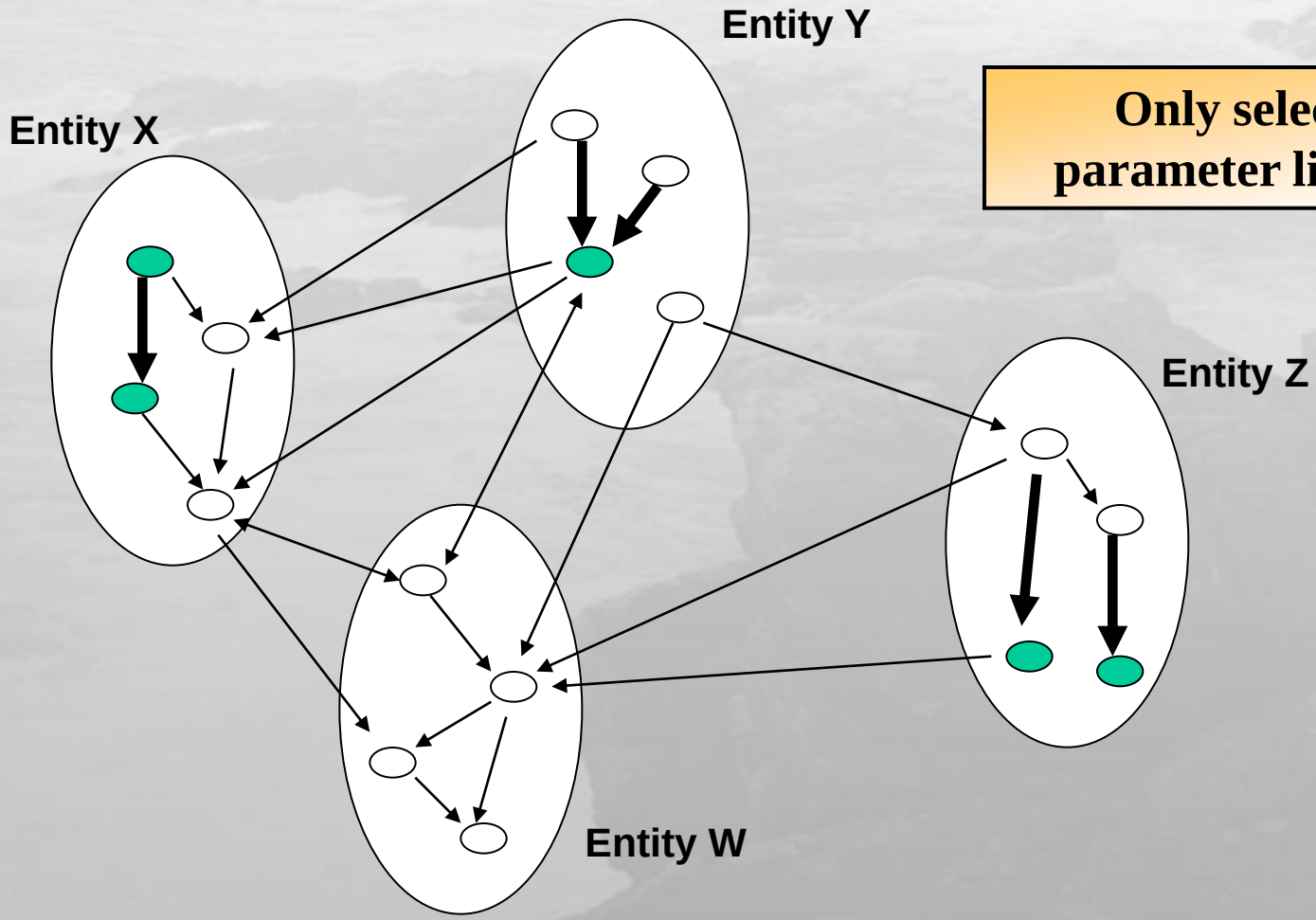
- *Fetch.SingleFetch*
  - *Fetch* view as output in *Output/FetchedData*
- *Fetch.BlockFetch*
  - *Fetch* view (64) as output in *Output/FetchedData*
- *Update.InsertRow*
  - *Update* view as dual input in *Input/InsertData*
- *Update.UpdateRow*
  - *Update* view (non-key) as dual input in *Input/InsertData*

Use of Views in Parameter Lists

- Bold arrows denotes calls to functions containing full-entity views in their parameter lists.



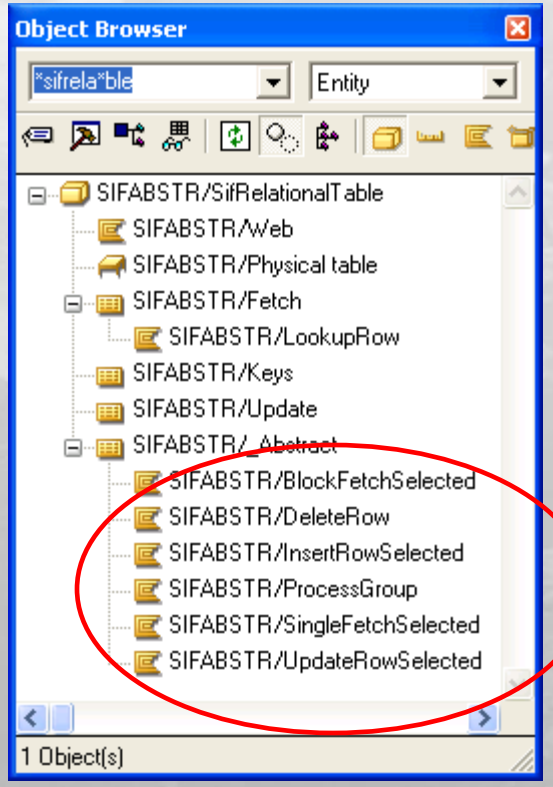
Restricted Use of Fields in Parameter Lists



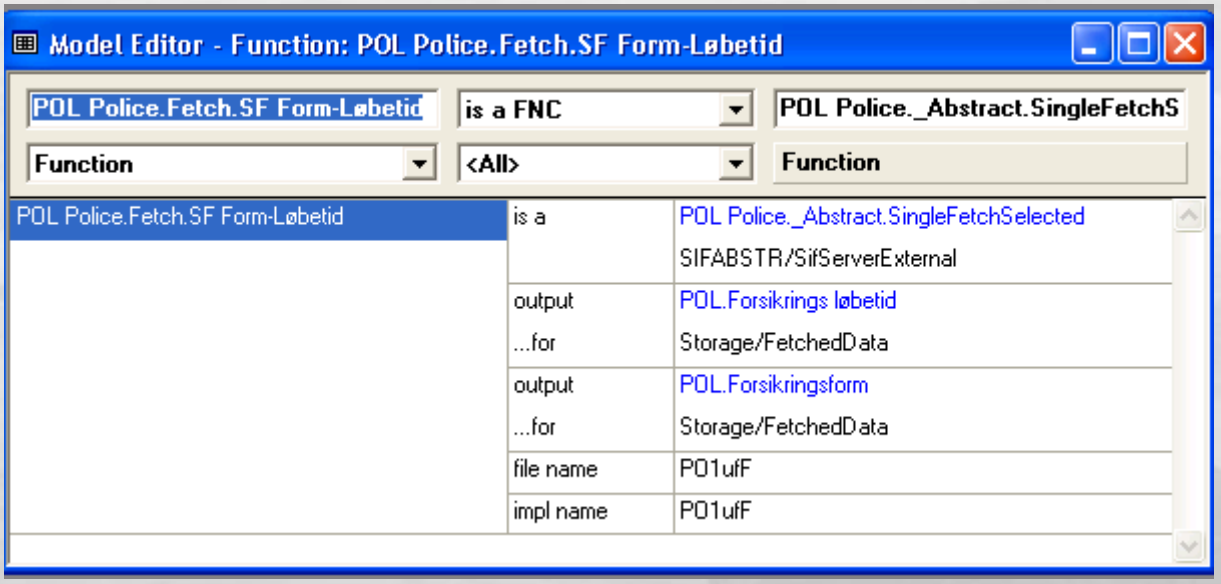
Only selected fields in parameter lists of functions

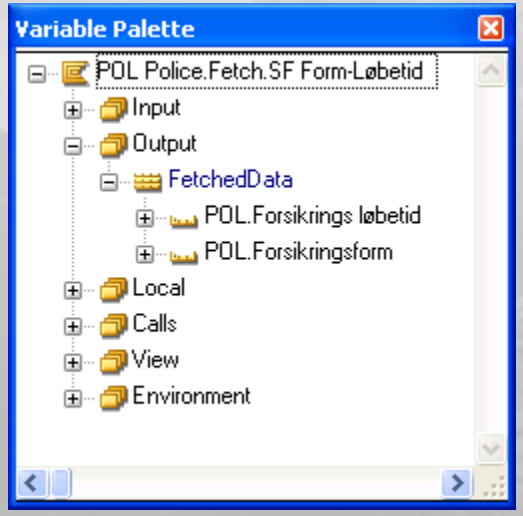
Abstract *RelationalTableSelected* entity

- Traditional naming of *Physical table* and *Update* and *Fetch* view
- *Keys* views…
- *LookupRow* as only implemented function
- Functions scoped under *_Abstract* view…



Example of SingleFetch Function



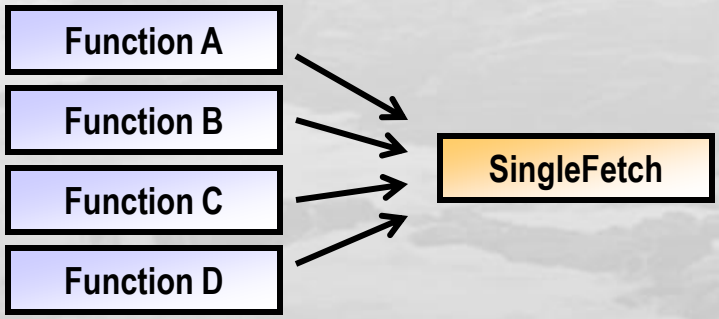


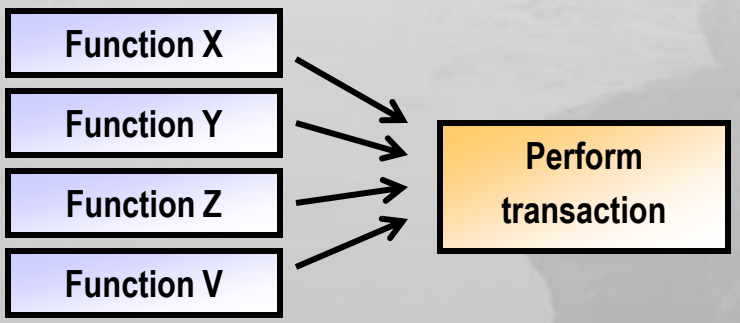
- Only selected fields in *FetchedData* variable

How Does Parameter Restriction Relate to Stable Interfaces?

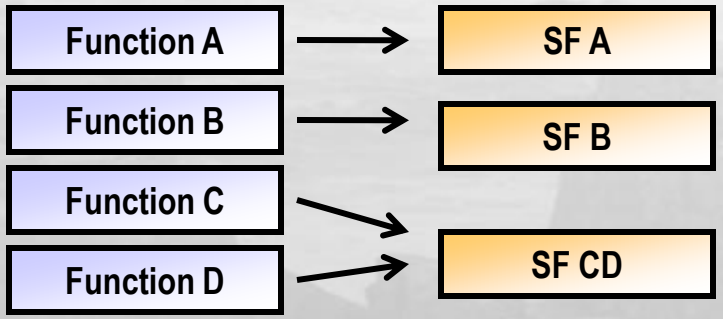
General or Granular?

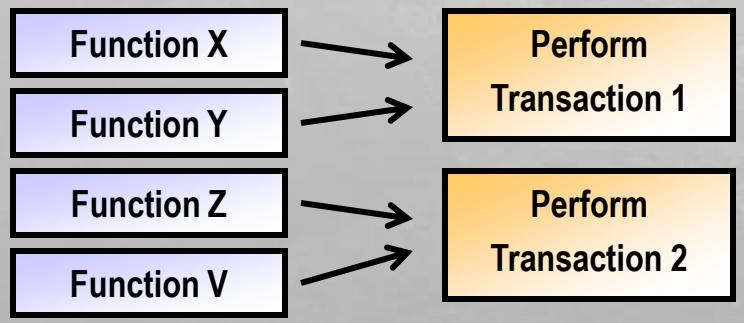
- General





- Granular





Granular Design?

Pros

- Robustness towards addition of fields and relation to entities
- Reduction of amount of objects/functions to be generated when changing data model
  - Less interference in change management
- Simple design and easy-to-understand functions
- Use of individual fields can be tracked

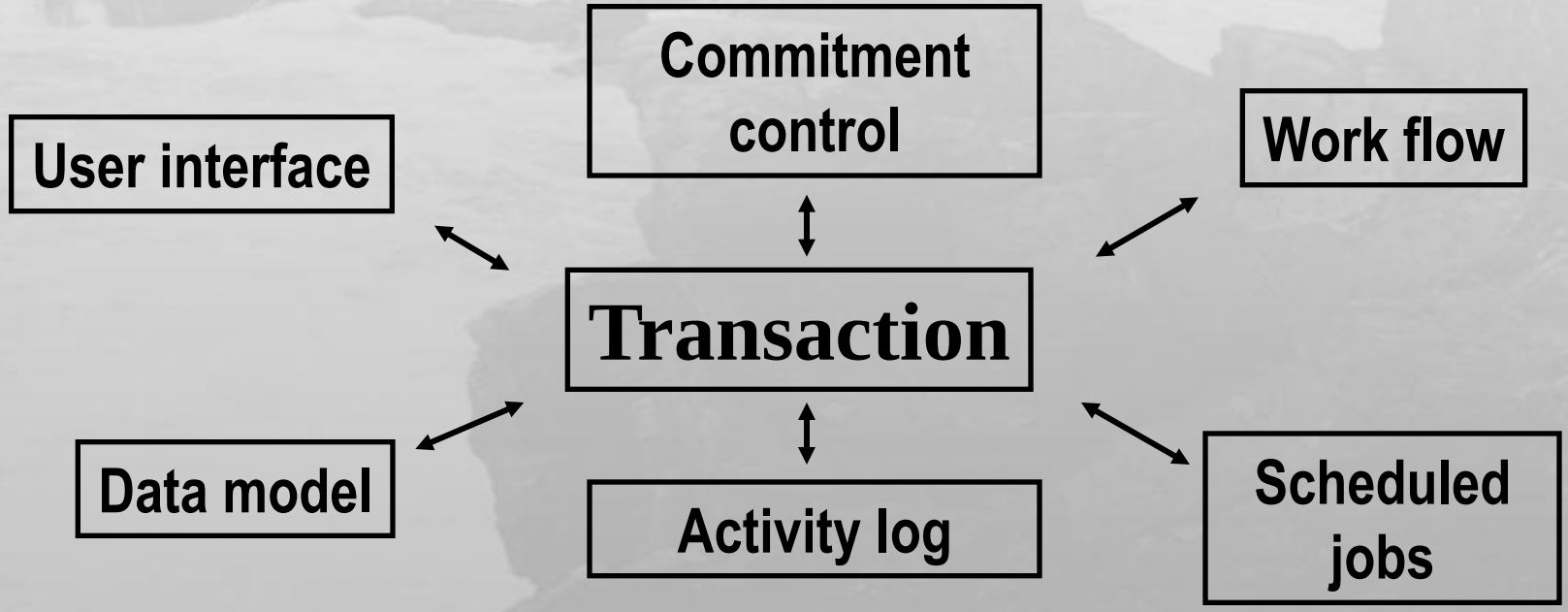
Cons

- Many implemented objects

4) TRANSACTIONS AND SERVER-SIDE VALIDATION

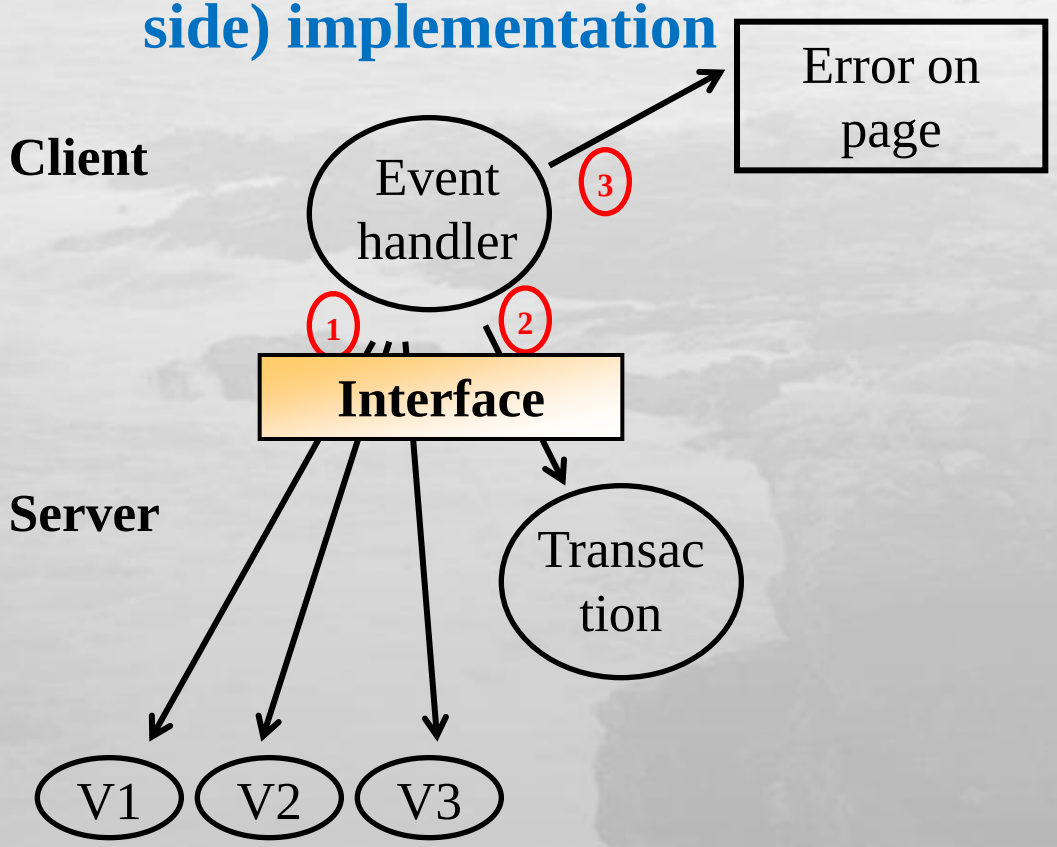
Transactions as Key Concept

- Services should have well-defined input, output and behaviour
- Focus on business and requirements

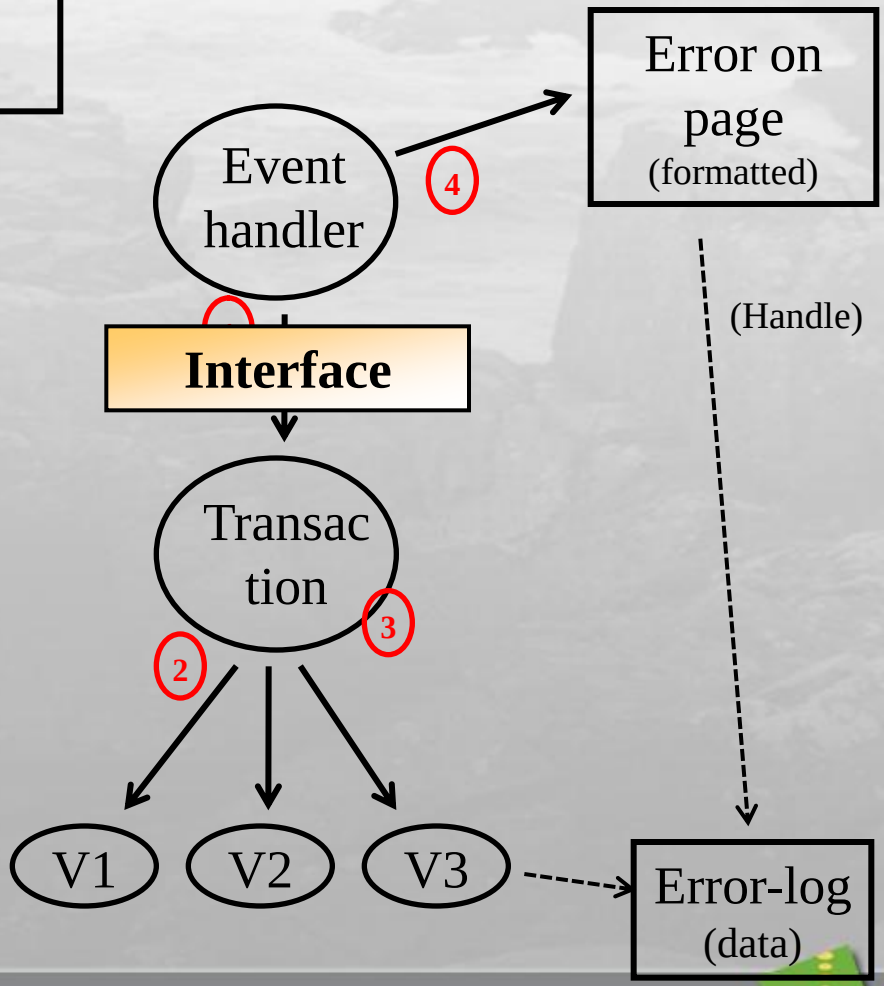


Server-Side Validation

Traditional (client-side) implementation

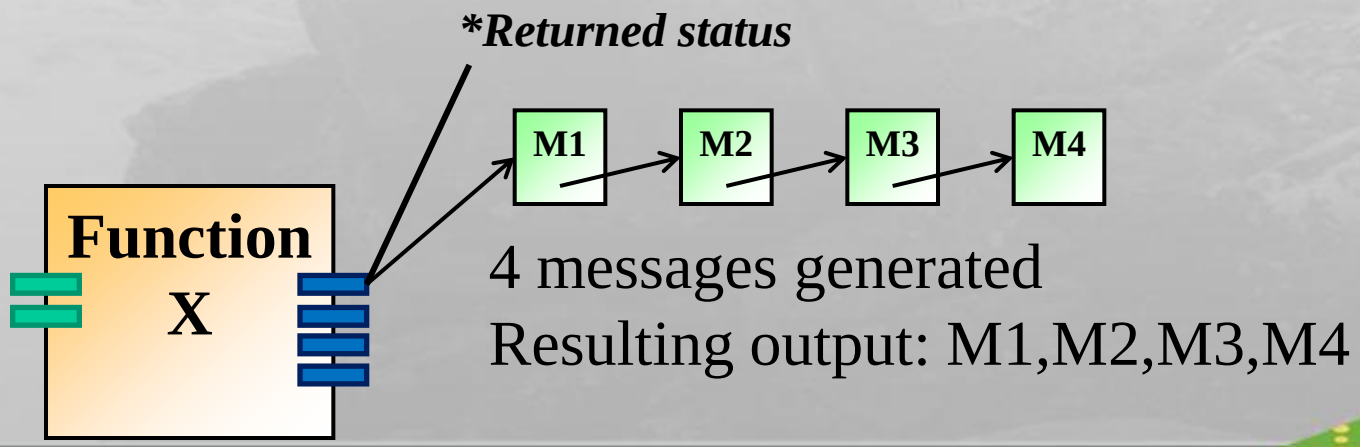


New (server-side) implementation

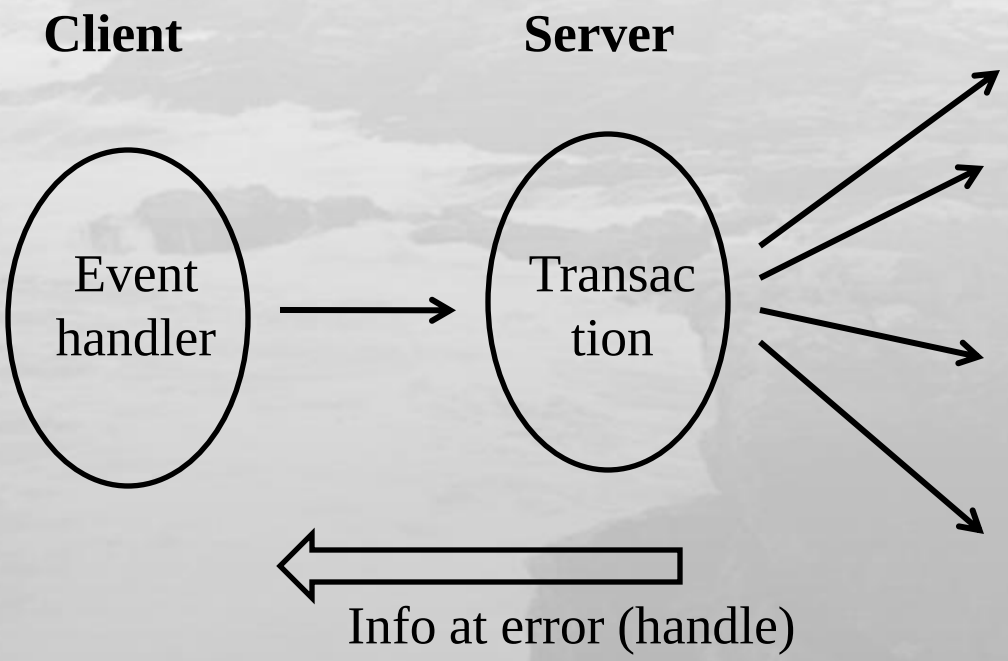


**Returned status* Used as Error Message Pointer

- Server-side validation
  - Error state passed back in **Returned status* as a pointer to list of messages
- Expected output as contents of list
  - Compare with actual list returned by transaction
- Error message list facilitated by Websydian Express…



Validation rules associated to data and transactions



1. Data validation
2. Function validation
3. Validation in transaction
4. Perform transaction

Server-Side Validation and Message Generation

1) Call transaction for Event handler
2) Call associated validations from transaction
3) Perform validation functions
4) Call message function (on error)
5) Create record for error message (on error)
6) Perform transaction
7) Call next page or call error on page
8) Retrieve error message(s) associated to *MsgID*
9) Display and mark errors in page

Meta code applying validation rules moved from client layer to transactions