

# *Integratie* mogelijkheden!

## Web Services en **Structs**

Berry Kuijer Saat  
Solution Specialist  
27 November 2014

**UNIFACE**



# Application integration (EAI)

Example:

- ▲ Accounting
- ▲ Asset management
- ▲ Employee management
- ▲ Service desk
- ▲ CRM
- ▲ Sales

**Per connection: at least one call-out and one call-in.**

*How many connection to be integrated?*

**UNIFACE**



# “Point to point” connections

▲ 1 → 0

▲ 2 → 1

▲ 3 → 3

▲ 4 → 6

▲ 5 → 10

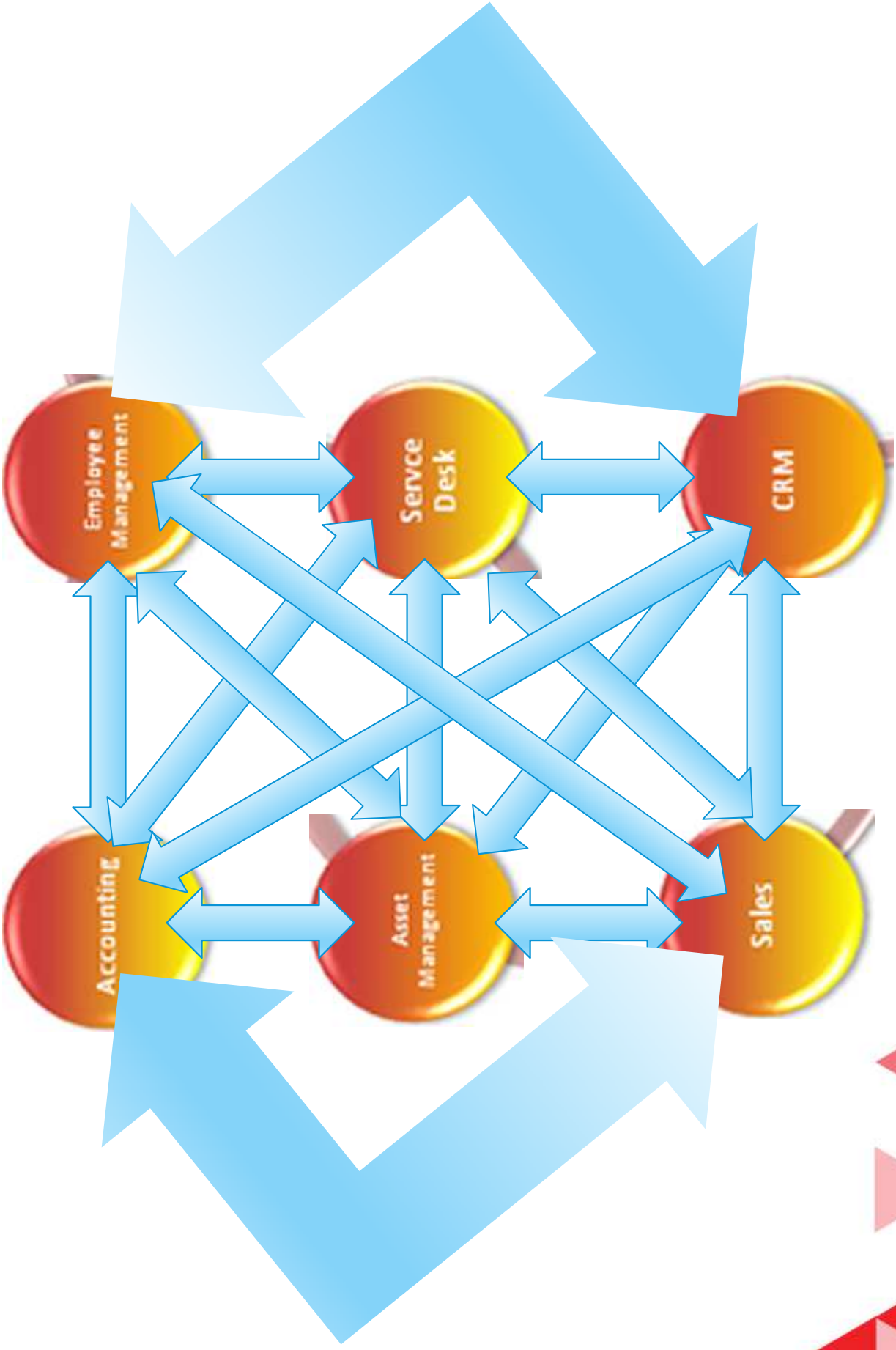
▲ 6 → 15

▲ ...  $\frac{n(n-1)}{2}$

▲  $N \rightarrow \frac{n(n-1)}{2}$

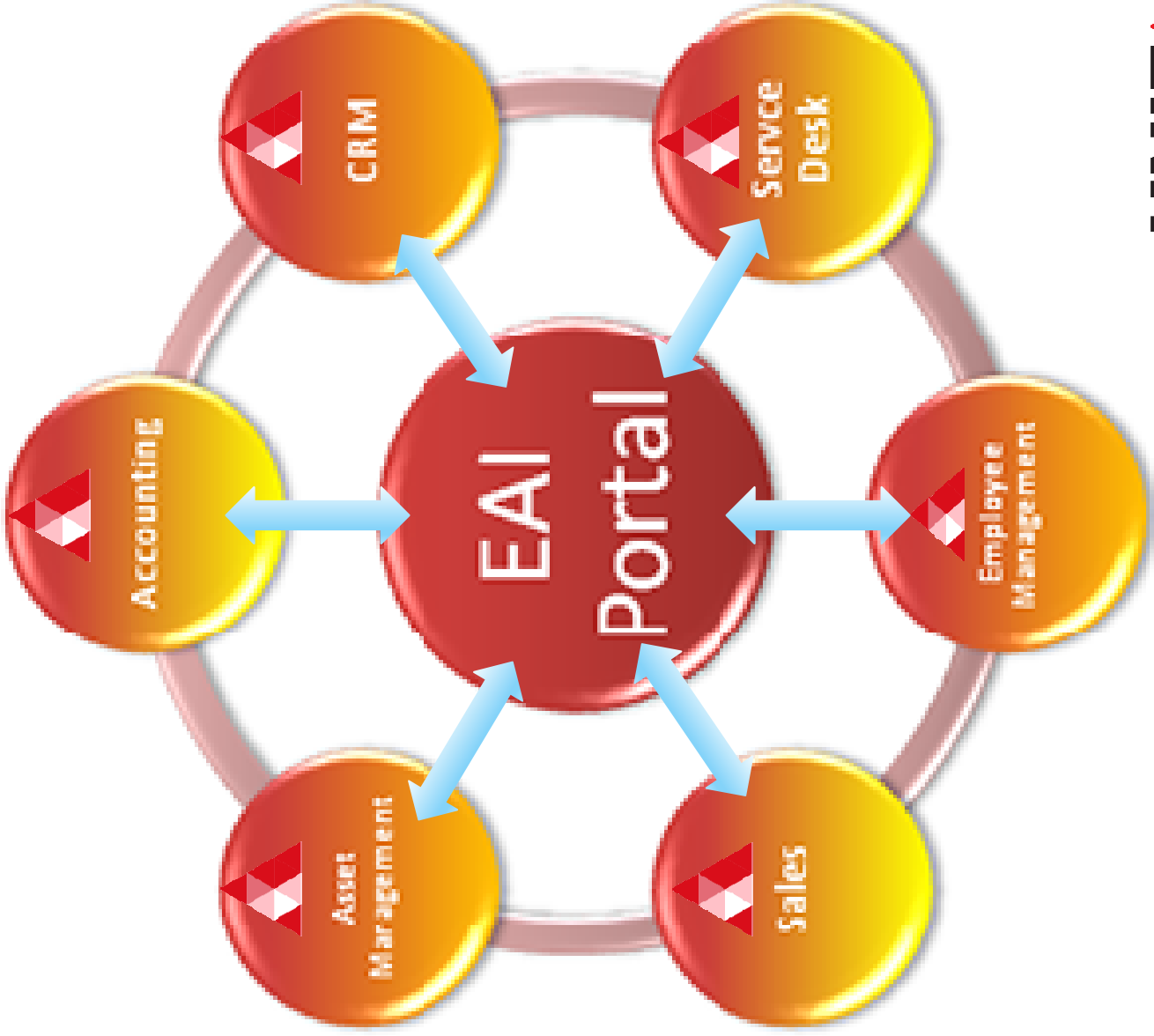
**additional 2 applications  
additional 13 connections**





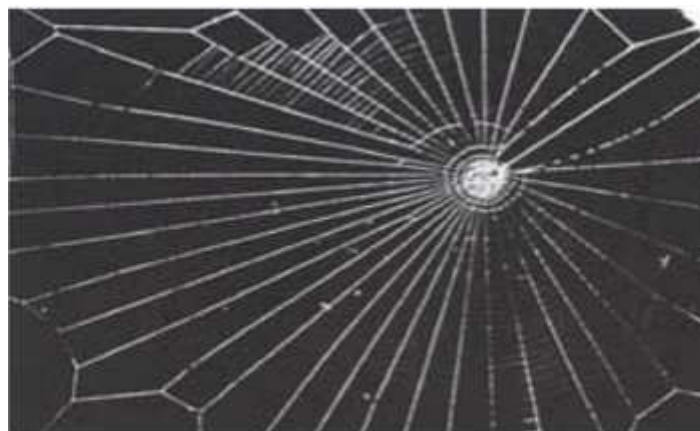
**UNIFACE**





**UNIFACE**





web of a spider on mescaline



spider on hashish



web of a spider on caffeine

Web Services  
The Model

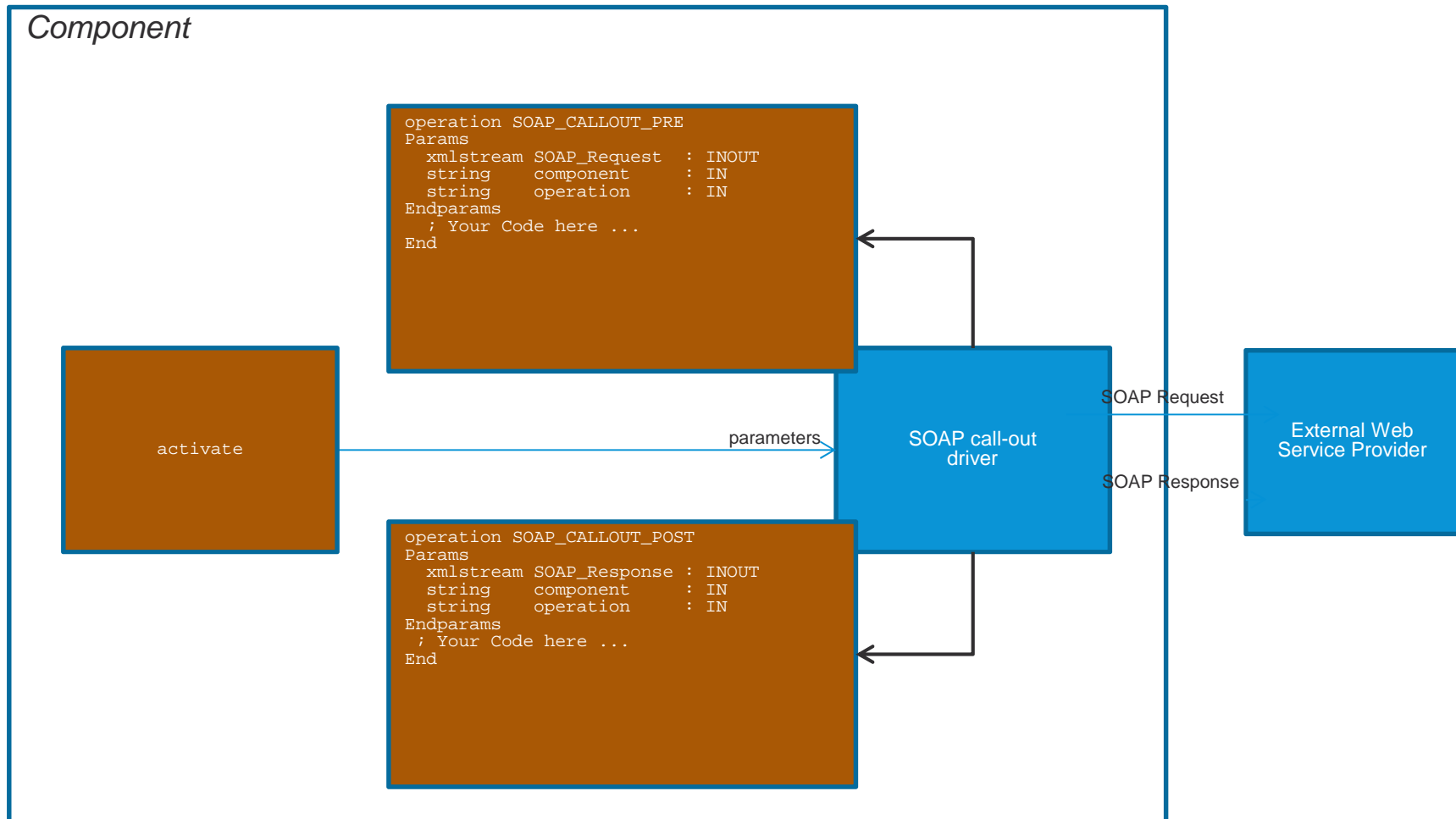
**UNIFACE**



Web Services  
A little recap



# SOAP call-out call-back operations





# SOAP call-out call-back operations

Call-back operation execution sequence:

[DRIVER\_SETTINGS]

USYS\$SOP\_PARAMS = callback=svc1,svc2,svc3

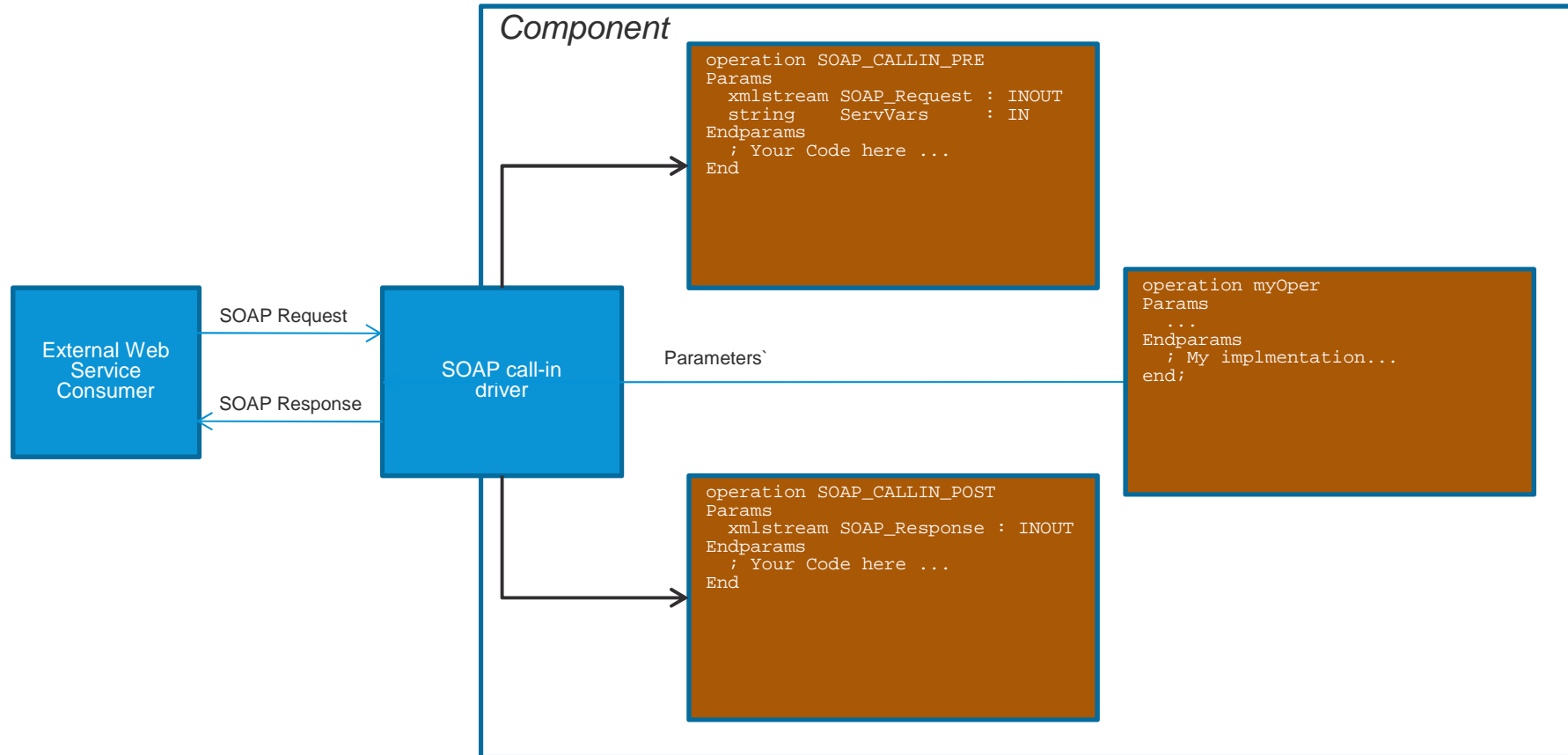
; overlaid with:

[SERVICES\_EXEC]

MYSOAPCPT = \$SOP:COMP1 callback=svc1,svc2



# SOAP call-in call-back operations



# SOAP call-in call-back operations

Call-back operation execution sequence:

[SETTINGS]

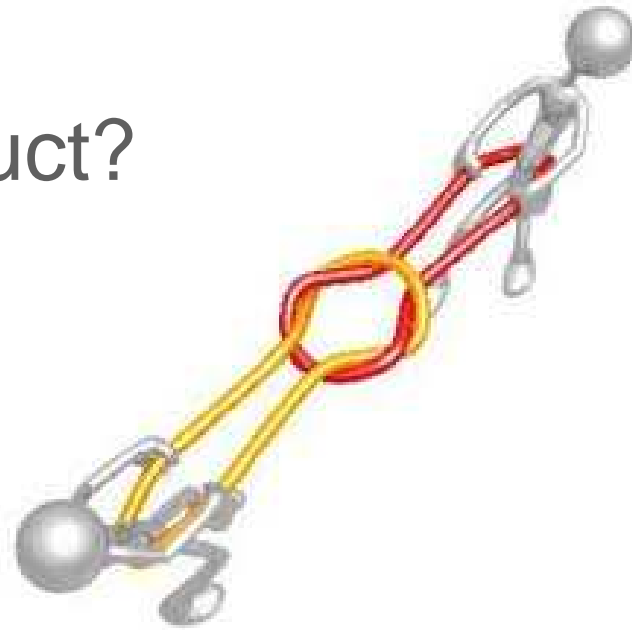
`$SOAP_CALLIN_CB = svc1, svc2, self, svc3`

Where 'self' refers to the current activated instance

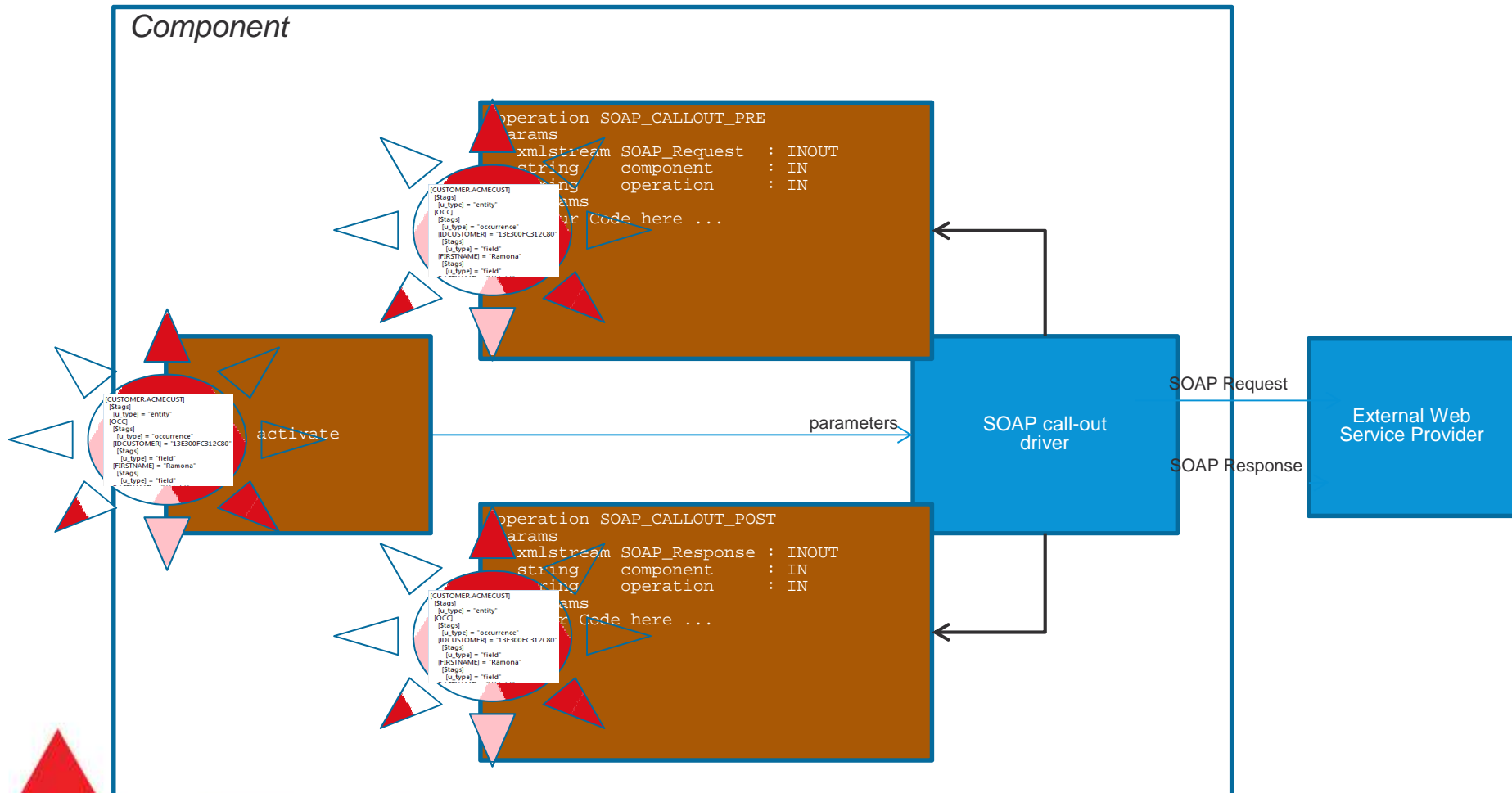


# Application

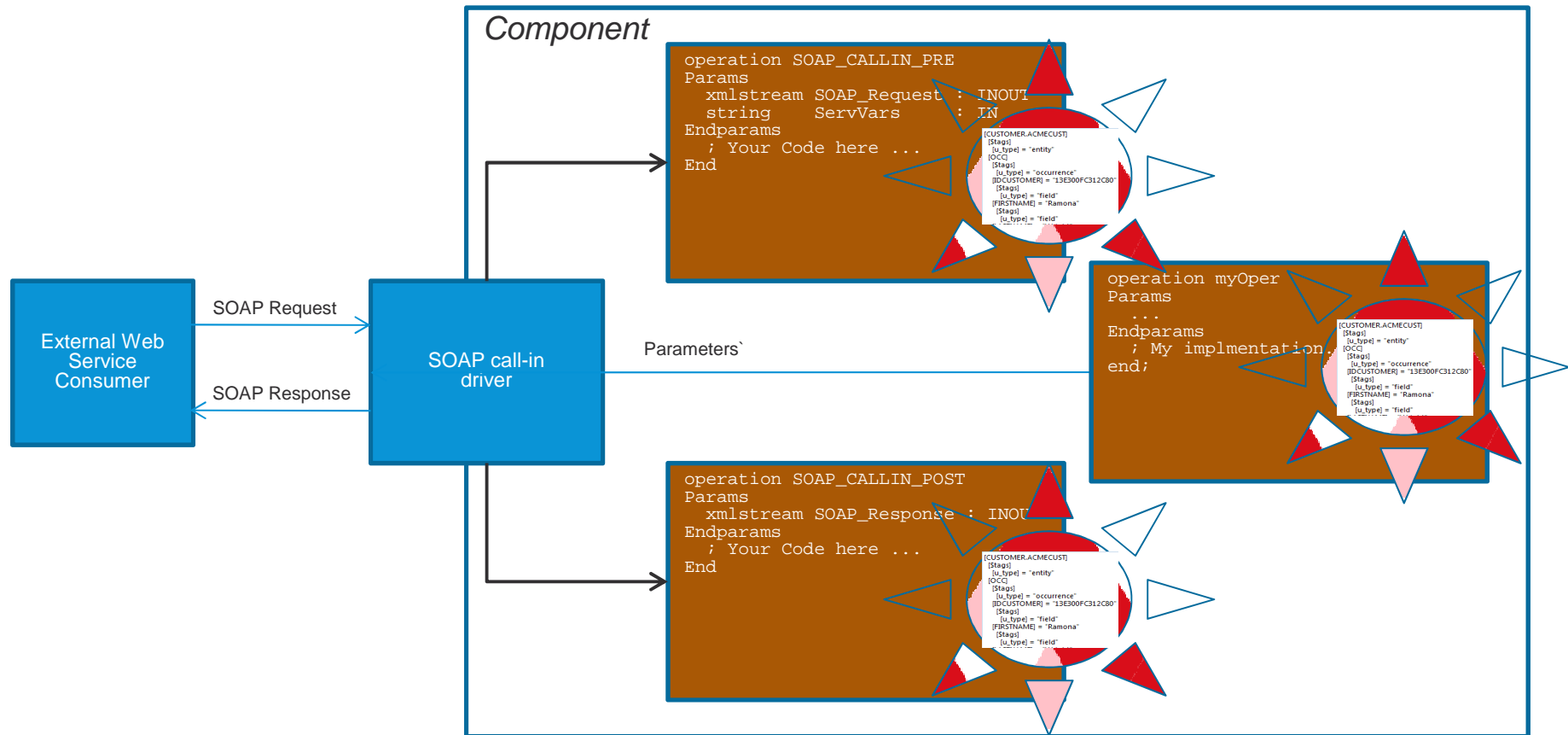
Where can we use Struct?



# SOAP call-out call-back operations

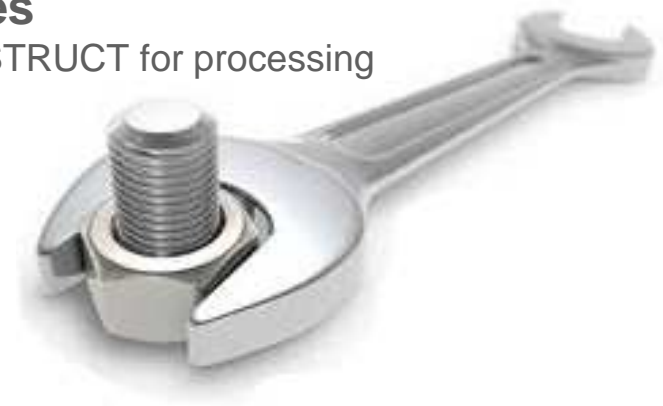


# SOAP call-in call-back operations



# Application examples

- **Complex parameter support of Web Services**  
complex parameters are created using Structs and then converted into XML to be passed as parameter and vice versa. (Struct supports XML schemas and all XML data types)  
Or are passed as Struct parameter.
- **Transformation of SOAP Headers and Messages**  
SOAP headers are made available as XML and converted into STRUCT for processing and/or encryption
- **Splitting and merging entities**



# Application examples

- **Complex data exchange between component instances**  
the Struct is directly used as a parameter, both for component instances running in the same and in different processes. Serialization is done automatically and on demand.
- **3-Tier communication**  
the developer already has a 3-tier application and wants to gradually replace xmlSave and xmlLoad statements (including all the DTDs and other overhead) with Struct constructions, where the Struct basically takes over the DTD and mapping administration
- **Exchange of JavaScript objects between browser and server**  
the serialized format of these JavaScript objects is JSON which can be converted into a Struct (on the server) for further processing. JavaScript objects are typically used in DSPs to exchange field properties and valreps, parameters and return values of custom JavaScript functions, parameters and return values of JavaScript functions of third party technology with a JS API (e.g. Google Maps)
- **Replacement of expensive list processing** –  
The Uniface list is a String and therefore inefficient for any type of manipulation; the Struct is an ordered collection of references to individual data members in memory and therefore very efficient for any type manipulation
- **Complex data exchange between functions/entries/operations**  
the developer already has entries/operations that exchange complex data using lists and he wants to interact with those. The lists can be converted to Struct for further processing.



Structs?



**UNIFACE**



# Structs

What it is:

- ▲ Data type
- ▲ Complex
- ▲ Tree-like, hierarchical

Where it is:

- ▲ In memory
- ▲ As Variable
- ▲ As Parameter



# Structs

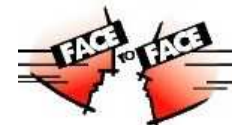
## Why

- ▲ Complex data manipulation
- ▲ Data transformation
- ▲ Good performance
- ▲ Standardization
- ▲ .....

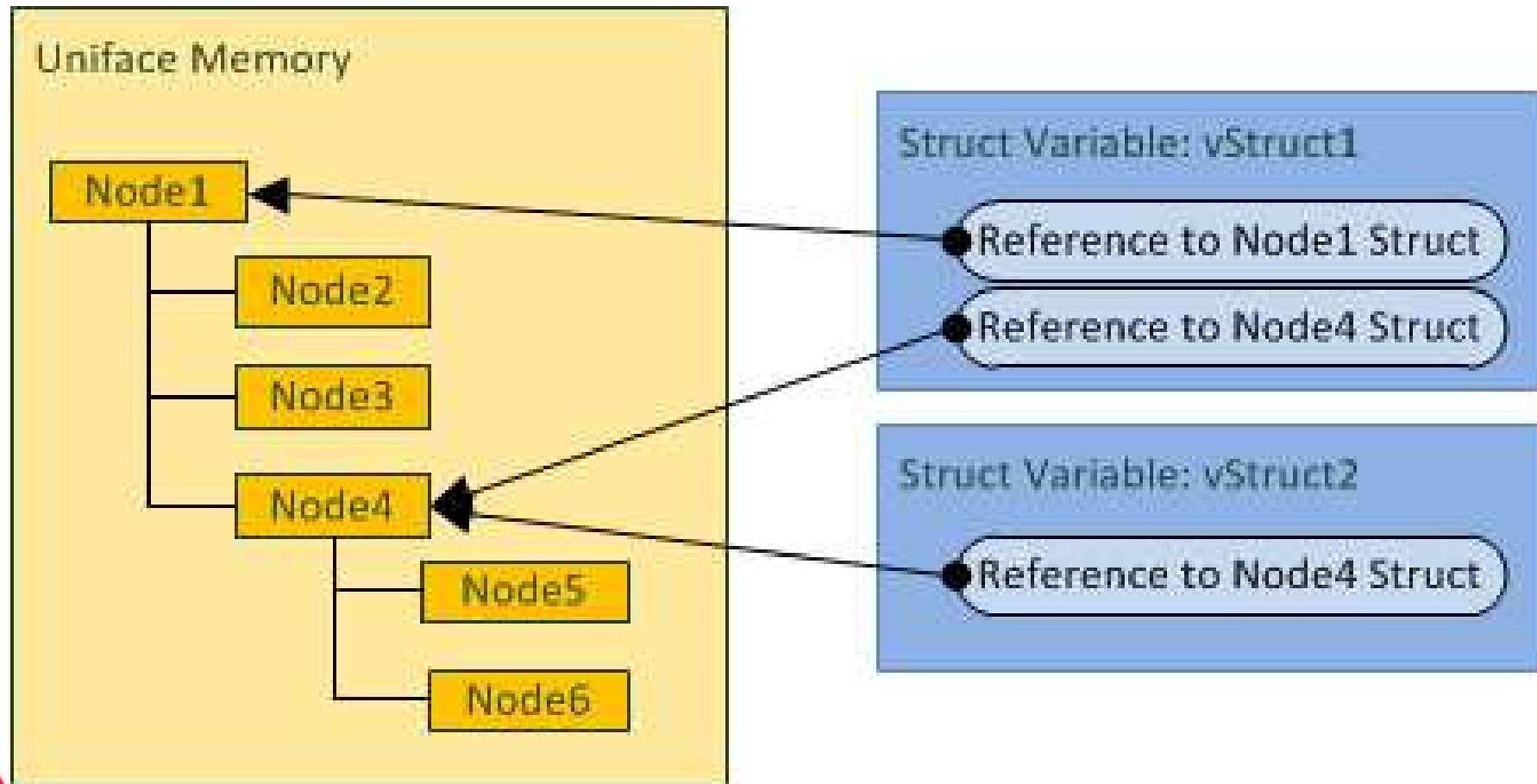
## How

- ▲ Access operators
- ▲ Proc functions

**UNIFACE**



# Structs, a recap



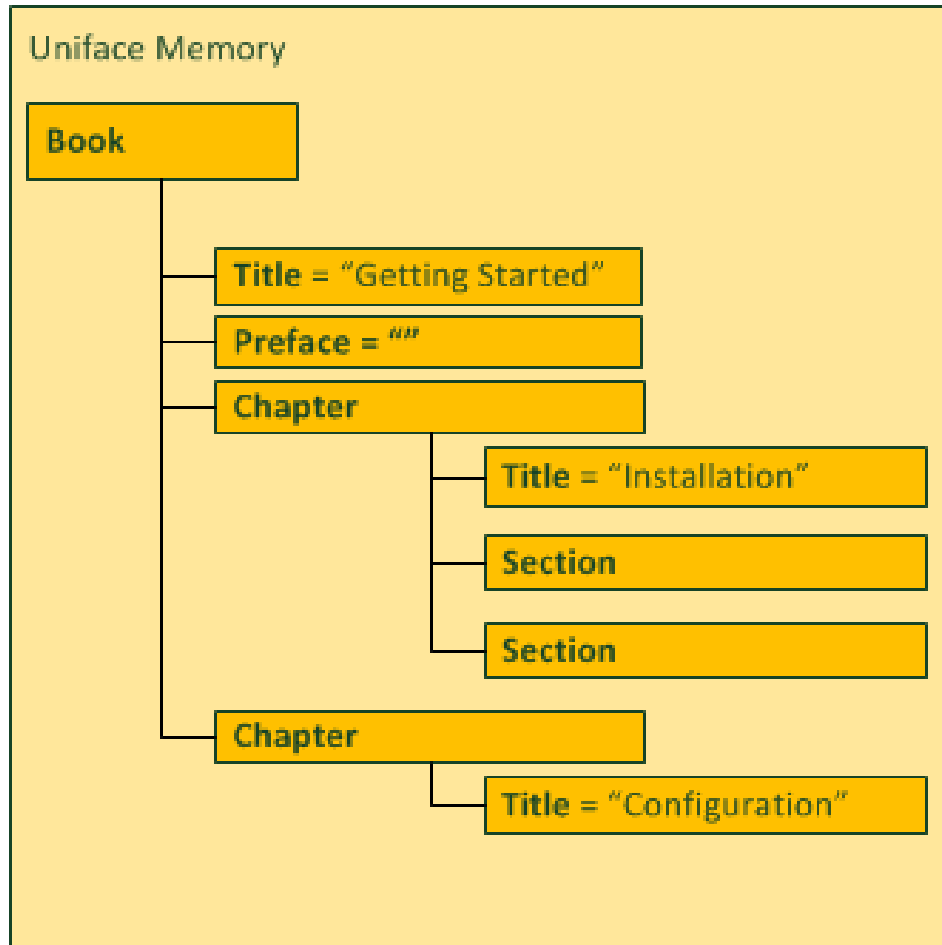
# Structs members

- ▲ Top (root) node
- ▲ Zero or more members = subnodes
  - ▲ Struct (nested structs)
  - ▲ Scalar (number, string, date)
  - ▲ Can have a name (doesn't need to)
  - ▲ Are sequentially ordered (have an index number)
- ▲ Can be:
  - ▲ Added
  - ▲ Removed
  - ▲ Copied
  - ▲ Moved

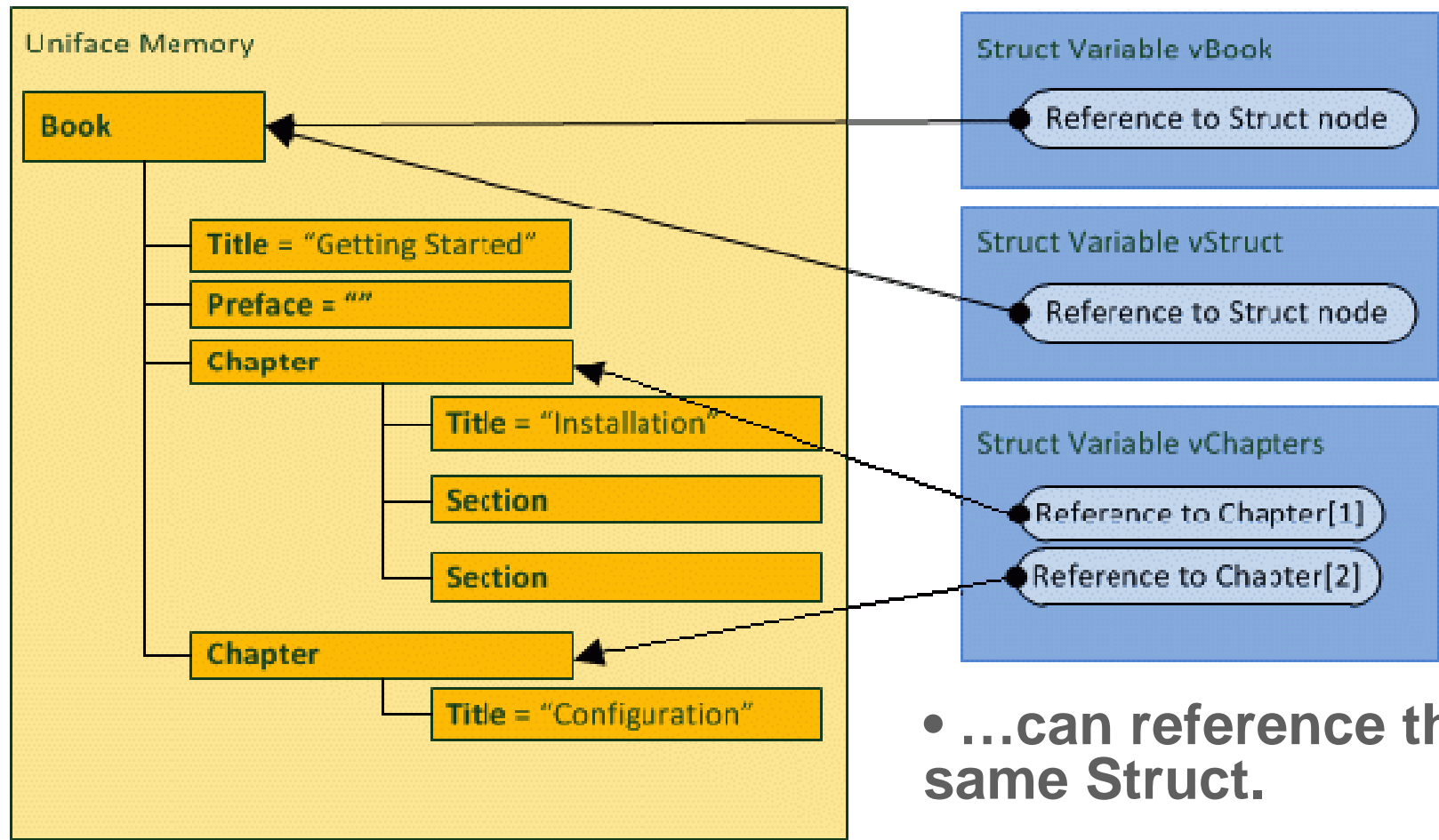
**UNIFACE**



# Structs and Members



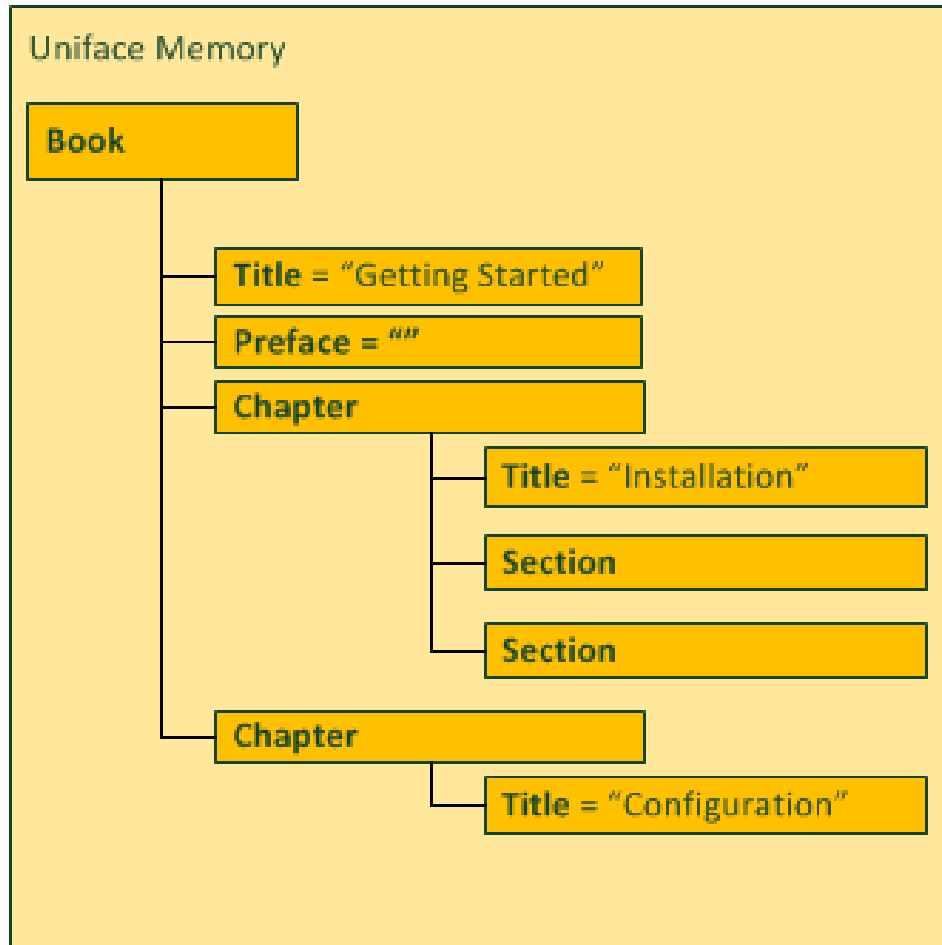
# Struct Variables



- ...can reference the same Struct.
- ...can reference multiple Structs.

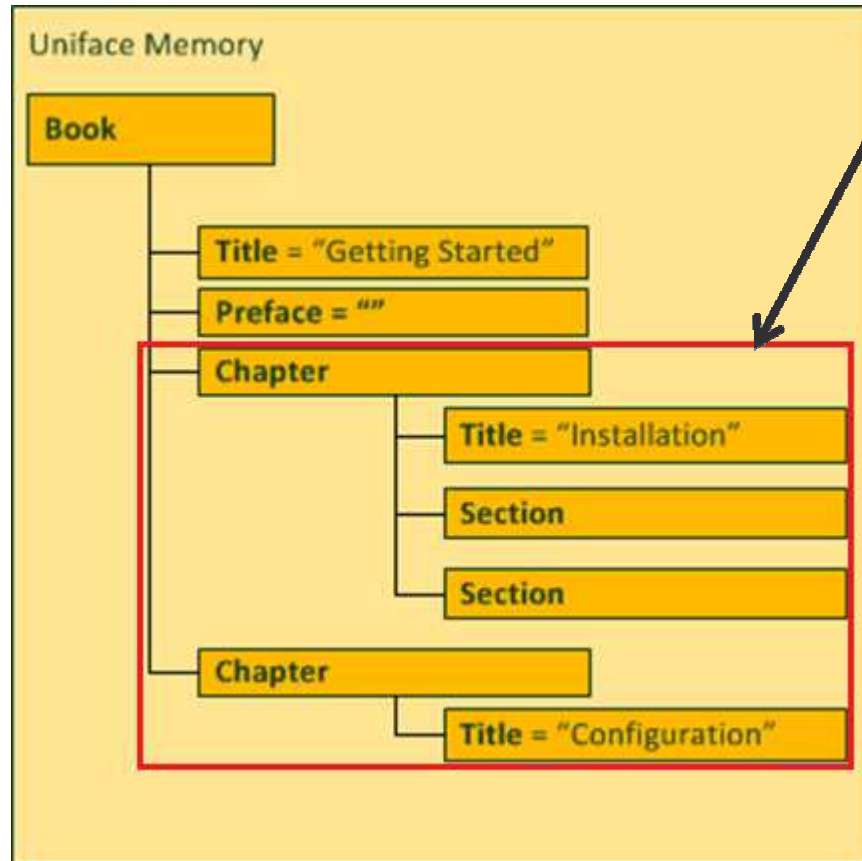


# Structs and Members





# Struct Variables



vBookStruct->Chapter

...can reference a Struct member via access operators.



# Struct Variables

- A change made via one struct variable is reflected in other struct variables referencing the same Struct.
- A struct variable can reference 0 Structs (\$collsize is 0).
- A NULL struct variable is a struct variable with \$collsize=0.
- Proc variables/parameters of type 'any' can reference Structs.



# Struct Access Operators

- – dereference operator
  - ***Variable->Name*** – returns a collection of references to all Structs named *Name*
  - ***Variable->\**** - returns a collection of references to all members of a Struct
- {N} – Struct index operator
  - ***Variable->Name{N}*** – returns a reference to a single Struct member from a collection of references, based on its index position in the collection

vStruct = Book->Preface->Acknowledgements{2}

...you can extract the value of the second occurrence of the member with name *phone\_number*, where multiple members have the same name:

vMobile = vStruct->phone\_number{2}



# Struct Access Operators

The \* wildcard is not the same as using a \* in a SQL statement or a retrieve profile. It can only be used directly after the de-reference operator. It cannot be used to limit the selection by preceding it with other characters.

Variable->ab\* ----- *incorrect*



# Struct Annotations (Tags)

Descriptive data elements to identify Struct members.

`<div class="note">Text can be <b>bold</b></div>`

converted to Struct:

```
[]
  [$tags]
    [xmlClass] = document
  [div]
    [$tags]
      [xmlClass] = element
      [class] = "note"
      [$tags]
        [xmlClass] = attribute
        "Text can be"
        [b] = "bold"
      [$tags]
        [xmlClass] = element
```

**UNIFACE**



# Some Struct Functions

<b>\$name</b>	<i>vStruct-&gt;\$name="New Name"</i>
<b>\$parent</b>	<i>vStruct1-&gt;\$parent=vStruct2</i>
<b>\$scalar</b>	<i>vStruct1=vStruct2-&gt;\$scalar</i>
<b>\$collSize</b>	<i>vNum=vStruct-&gt;\$collSize</i>
<b>\$index</b>	<i>vStruct-&gt;\$index=-1</i>
<b>\$memberCount</b>	<i>vNum=vStruct-&gt;\$memberCount</i>



# Struct String Representation

**\$dbgString** (*include tags*)

e.g. `putmess vStruct->$dbgString`

**\$dbgStringPlain** (*exclude tags*)

e.g. `StringFld=vStruct->$dbgStringPlain`





# Proc for Manipulating Structs

**struct** – data type declaration for variables and parameters

**\$newstruct** – create a new empty Struct (*vStruct=\$newstruct*)

**\$equalStructRefs** – check if struct variables reference the same Struct  
( *\$equalStructRefs(strVar1, strVar2)* )

**componentToStruct** – write Uniface occurrence data from a component instance to a Struct

**structToComponent** – convert data from a Struct into respective entities and occurrences in a component instance



# Proc for Manipulating Structs

**xmlToStruct** – convert any well formed XML document to a Struct

**structToXml** – convert a Struct to an XML document

**jsonToStruct** – convert JSON text to a Struct

**structToJson** – convert a Struct to JSON text



# Struct Tags for XML Conversion

- **xmlClass**
- **xmlNamespaceURL**
- **xmlVersion**
- **xmlEncoding**
- **xmlStandAlone**
- **xmlNamespaceAlias**

For a full list of all tags, see the Uniface documentation or online help.



# Component to Struct

AUFG\_STRUCT\_03L

Arbeiten mit Struct: XML-Datei mit geänderter Struktur

build struct    structToXML

Filename:     XML -> File    Exit

ID: 13E300FC312C80

First Name: Ramona

Last Name: Wright

Email address: MissWright@dreamland.com

Hint Text:

Country: United States of America

Telephone Code: 001

House: 1

Street: Sawgrass Corporate Parkway

Town: Sunrise

County: Florida

Post Code: 33324

Telephone: 955.331

Order#	Status	OrdDate	ShpDate	OrdTotal
60	Delivered	06-JUN-2009	16-JUN-2009	674.87

Articlename	Qty	Total
NCKia N80 - Sim Free Smart Phon...	3	235.17
Nckia 6630	1	53.88
New Apple iPod touch 8GB (4th ...	1	185.22

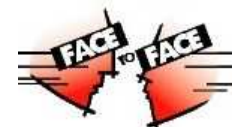


```
[CUSTOMER.ACMECUST]
  [$tags]
    [u_type] = "entity"
  [OCC]
    [$tags]
      [u_type] = "occurrence"
    [IDCUSTOMER] = "13E300FC312C80"
    [$tags]
      [u_type] = "field"
    [FIRSTNAME] = "Ramona"
    [$tags]
      [u_type] = "field"
```

*component*

*Struct*

**UNIFACE**



# Component to Struct to XML

```
[CUSTOMER.ACMECUST]
[$tags]
[u_type] = "entity"
[OCC]
[$tags]
[u_type] = "occurrence"
[IDCUSTOMER] = "13E300FC312C80"
[$tags]
[u_type] = "field"
[FIRSTNAME] = "Ramona"
[$tags]
[u_type] = "field"
```



```
<CUSTOMER.ACMECUST>
<OCC>
<IDCUSTOMER>13E300FC312C80</IDCUSTOMER>
<FIRSTNAME>Ramona</FIRSTNAME>
<LASTNAME>Wright</LASTNAME>
<EMAIL>MissWright@dreamland.com</EMAIL>
<HINTTEXT/>
<ADDRESS.ACMECUST>
<OCC>
<COUNTRY.ACMECUST>
<OCC>
<NM>United States of America</NM>
<TELCODE>001</TELCODE>
```


**Struct**

**XML document**

**UNIFACE**



# Component to Struct Uniface tags

- *u\_type="component"*
- *u\_type="entity"*
- *u\_type="occurrence"*
- *u\_type="field"* 



# Struct as parameter

- **Passed by reference** – by default in private operations, a copy of the reference is made
- **Passed by value** – by default in public operations, a copy of the Struct is made
- Defaults can be changed by using the *byRef* and *byVal* qualifiers when declaring the Struct parameters



# If time allows: Demo time

### Some Struct Basics

run of STRUCT\_BASICS

Struct referred to by vStruct:

```
[group]
[person]
[firstname] = "John"
[email] = "john@home.com"
```

Struct referred to by vPerson:

```
[person]
[firstname] = "John"
[email] = "john@home.com"
```

Struct referred to by vStruct:

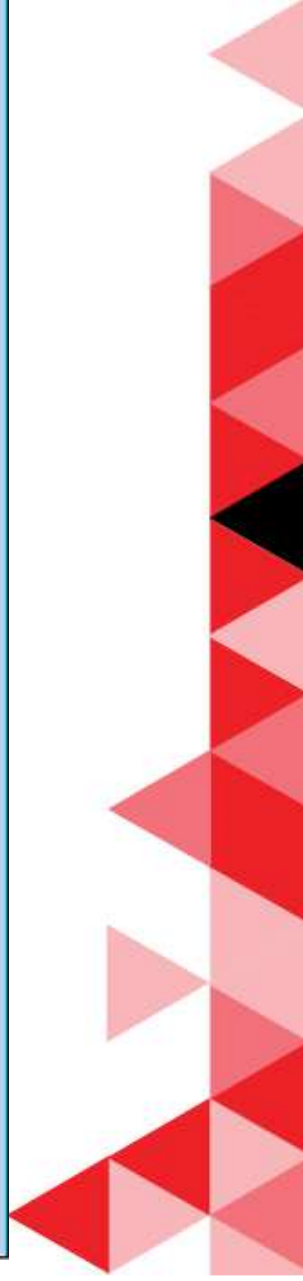
```
[group]
[person]
[firstname] = "John"
[email] = "john.smith@home.com"
```

Note: 'email' changed with vPerson is also changed in vStruct:  
The parent of vPerson is: [group]  
After assigning a new value to vStruct, vStruct points to empty Struct:  
[] = ""

vPerson is still valid: [John]  
but the parent of vPerson is now invalid. []

out

- struct\_basics
- member\_name\_charset
- copy\_struct
- name\_inheritance
- tags\_inheritance
- move\_struct
- remove\_member
- struct\_as\_params
- identical\_member\_names
- member\_references
- struct\_collections
- remove\_struct\_level
- insert\_struct\_level





; Create Struct members:

```
vStruct->group = $newstruct
```

```
vStruct->group->person = $newstruct
```

```
vStruct->group->person->firstname = "John"
```

```
vStruct->group->person->email = "john@home.com"
```

;Update the Struct using a different variable

```
vPerson = vStruct->group->person
```

```
vPerson->email = "john.smith@home.com"
```



; Copy by reference

```
vStruct1->a = "AAA"
```

```
vStruct1->b = "BBB"
```

*;Copy vStruct1 to vStruct2 (by reference)*

```
vStruct2 = vStruct1
```

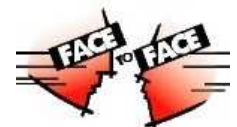
*;Update the Struct using vStruct2*

```
vStruct2->b = "BBB-updated"
```

*"Although vStruct2 changed the Struct, vStruct1->b  
returns the change:"*

```
putmess "%%(vStruct1->b)%%"
```

**UNIFACE**



;Copy by value

```
vStruct1= $newstruct
```

```
vStruct2= $newstruct
```

```
vStruct1->a = "AAA"
```

```
vStruct1->b = "BBB"
```

*;Copy vStruct1 to vStruct2 (by value)*

*;the left side of the assignment is a Struct member*

```
vStruct2->subnode = vStruct1
```

*;Update the copied struct*

```
vStruct2->subnode->b = "BBB-updated"
```



;Tags inheritance when copying a Struct:

```
vStruct1->member1 = "value A"
```

```
vStruct1->member1->$tags->someTags = "tag value A"
```

```
vStruct1->member2 = "value B"
```

```
vStruct1->member2->$tags->someTags = "tag value B"
```

*;Overwrite member2 with a copy of member1;*

```
vStruct1->member2 = vStruct1->member1
```

*;Tags inheritance when assigning a scalar value*

```
vStruct1->member2 = "updated value"
```



## Struct as parameter:

```
vStructAsIn = $newstruct
```

```
call STRUCT_PARAMS_IN_DO(vStructAsIn)
```

```
entry STRUCT_PARAMS_IN_DO
params
  struct pStruct: IN
endparams

  pStruct->aValue = "1111"

end ;
```



**UNIFACE**



Struct as parameter:

Entry creates a Struct passes it back

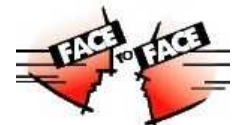
call STRUCT\_PARAMS\_OUT\_DO(vStructAsOut)

```
entry STRUCT_PARAMS_OUT_DO
params
  struct pStruct: OUT
endparams

pStruct = $newstruct
pStruct->aValue = "2222"

end
```

**UNIFACE**



## Struct as parameter:

Struct manipulations done by the called module are not visible if the Struct is (1)not created in the calling module, or is (2)not returned by the called module.

**UNIFACE**



## Struct collections:

vStruct1->\$name = "Struct1"

vStruct2->\$name = "Struct2"

vStruct1->a = "A1"

vStruct1->a{2} = "A2" ; Add a member at position 2

vStruct1->a{-1} = "A3" ; Append new member at end

vStruct1->b = "B"

vStruct1->a->\$collsize

vStruct1->\*





## Struct collections:

```
i = 1
```

```
while (i <= vStruct1->a->$collSize)
```

```
  putmess "vStruct1->a{%%i%%}% has value  
    %% (vStruct1->a{i}) %% %"
```

```
    i = i + 1
```

```
endwhile
```

```
vStruct1->a{1} = "A1 - updated"
```



A collection of multiple Structs always returns an empty value

```
vStruct1->a =  
However, if the collection contains a single Struct,  
it is treated as a single Struct:  
= A1 - collection reassigned  
vStruct1->a  
vStruct1->a{1} = A1 - collection reassigned
```

Using Struct functions on collections:

```
All Structs vStruct1->a share the same name: [a]  
: []  
but not all vStruct1->*  
Proc error: -1151: Structs do not have a common name or parent  
: [Struct1]  
All Structs vStruct1->* share their parent : []  
but not all Structs in any collection  
Proc error: -1151: Structs do not have a common name or parent
```



**UNIFACE**

## Struct collections:

*;\$parent*

vStruct1->a->\$parent = vStruct2

*;\$name*

vStruct2->a->\$name = "AAA"



Struct collections:

*;assign a subnode to each 'a' member*

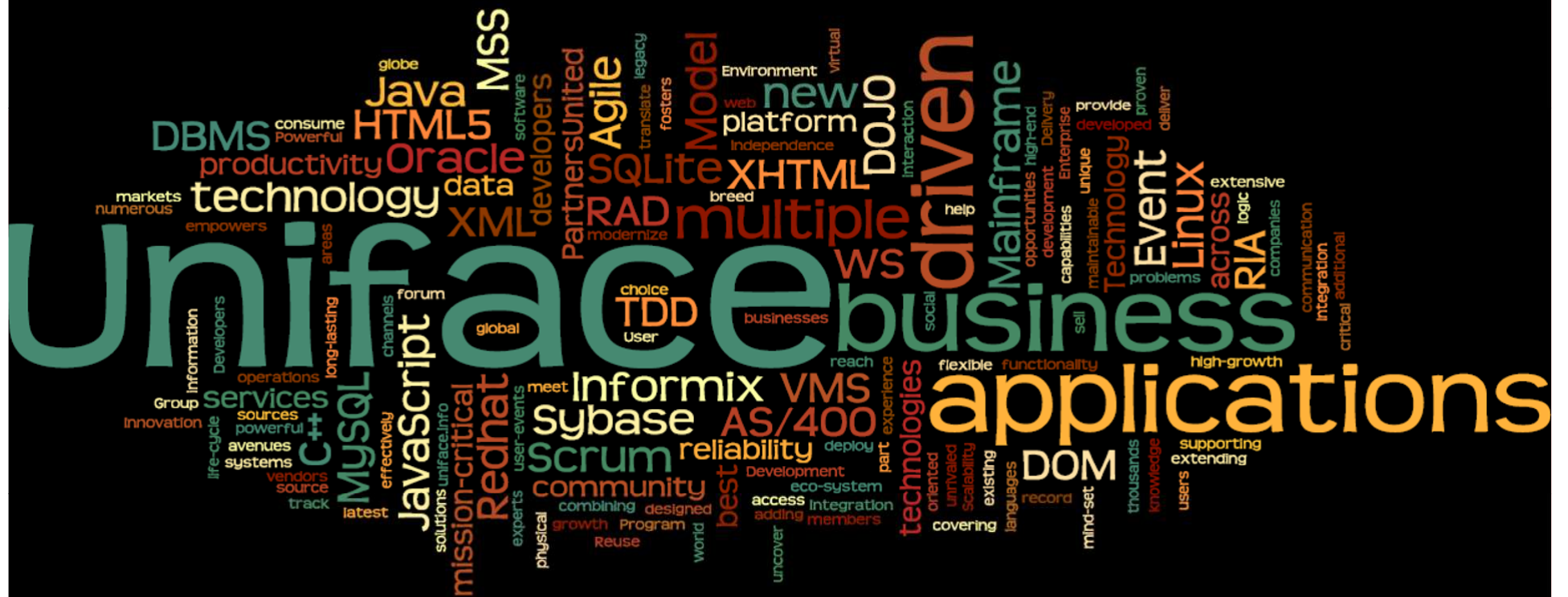
vStruct2->\*->x = "xyz"





**Binnenkort in dit  
theater**

**Uniface Struct Workshop**



Thanks