Multiparty Languages

The Choreographic and Multitier Cases

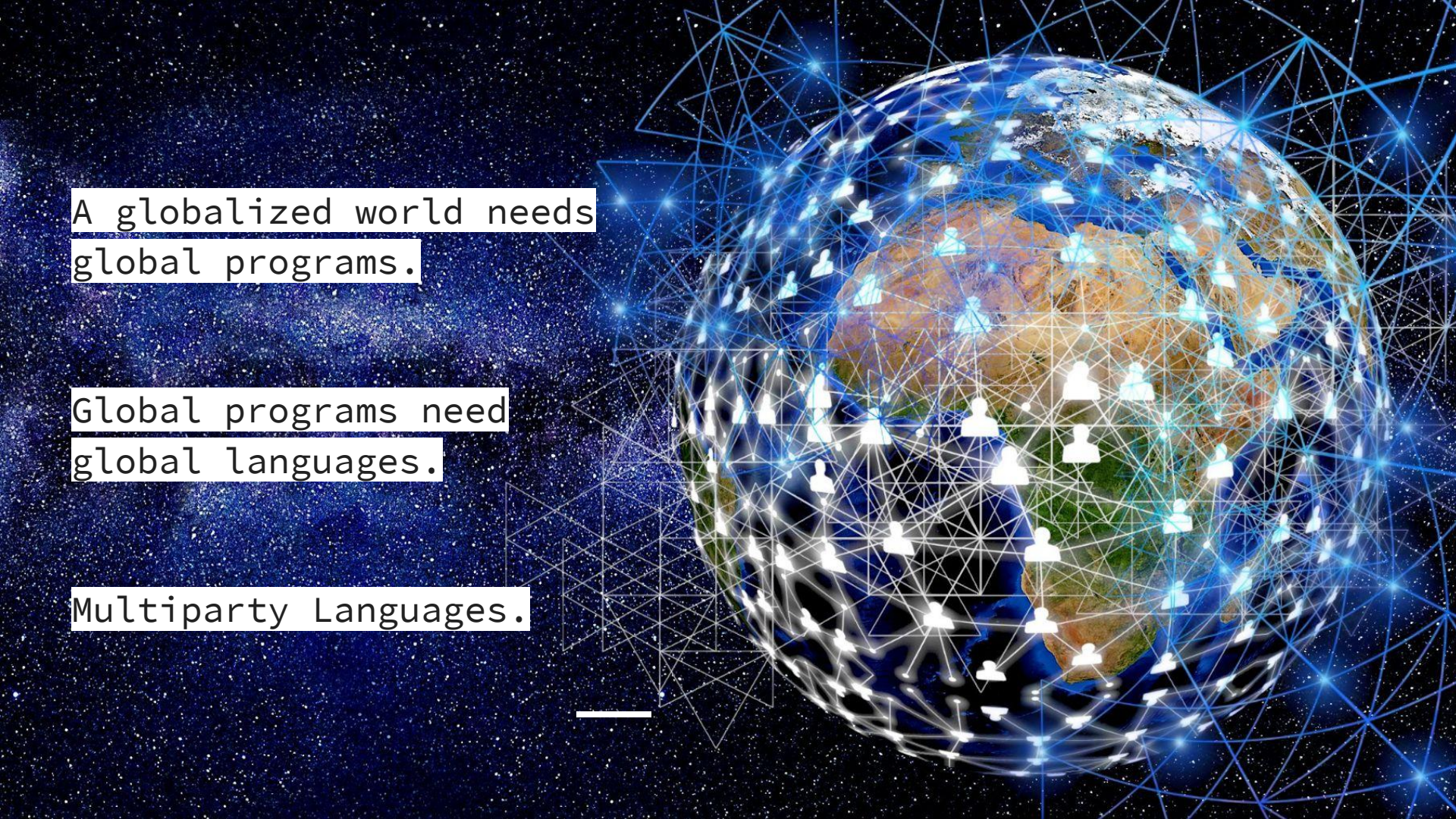
Multi Party People

Saverio Giallorenzo, Fabrizio Montesi, Marco Peressotti,
David Richter, Guido Salvaneschi, Pascal Weisenburger

A globalized world needs
global programs.

Global programs need
global languages.

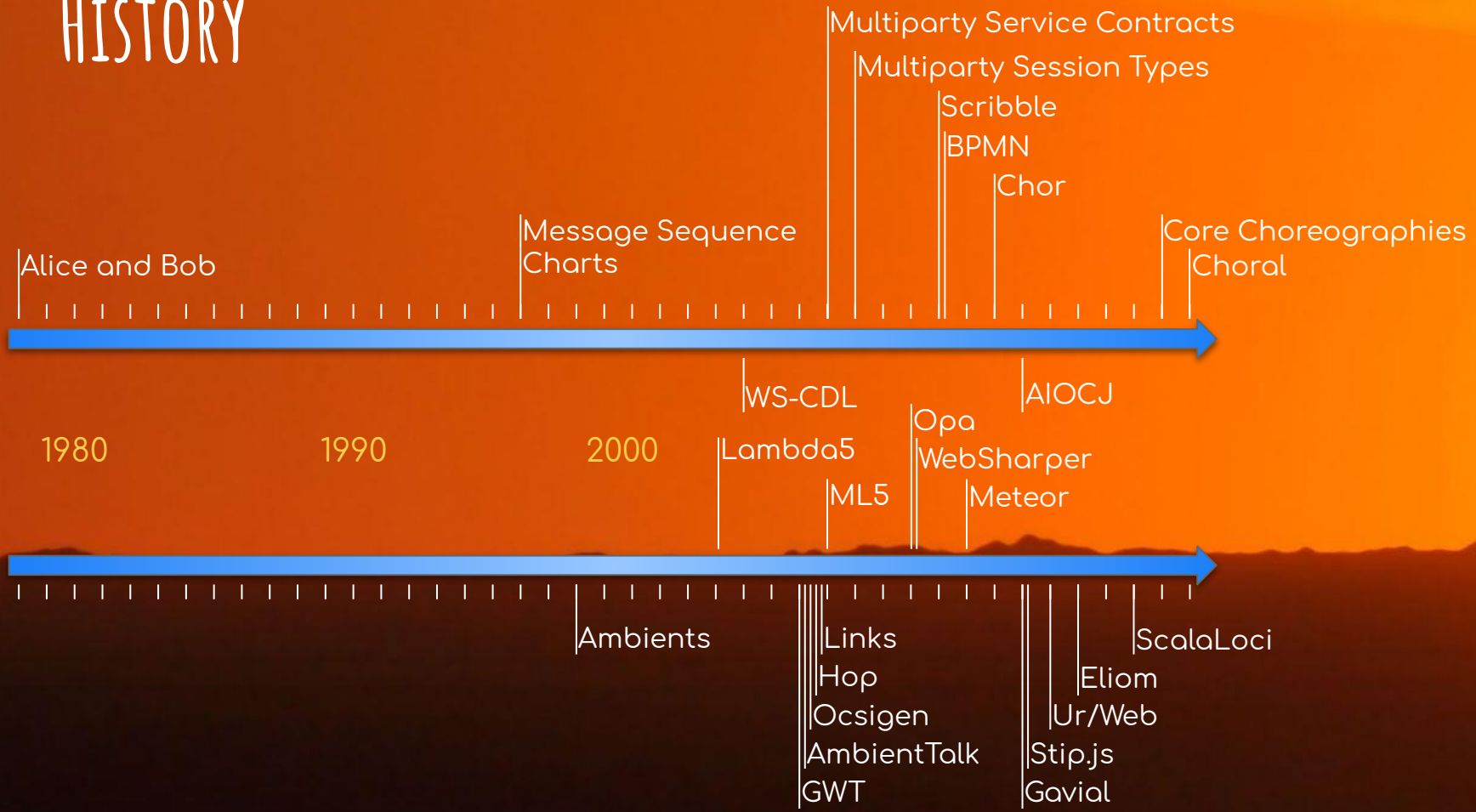
Multiparty Languages.



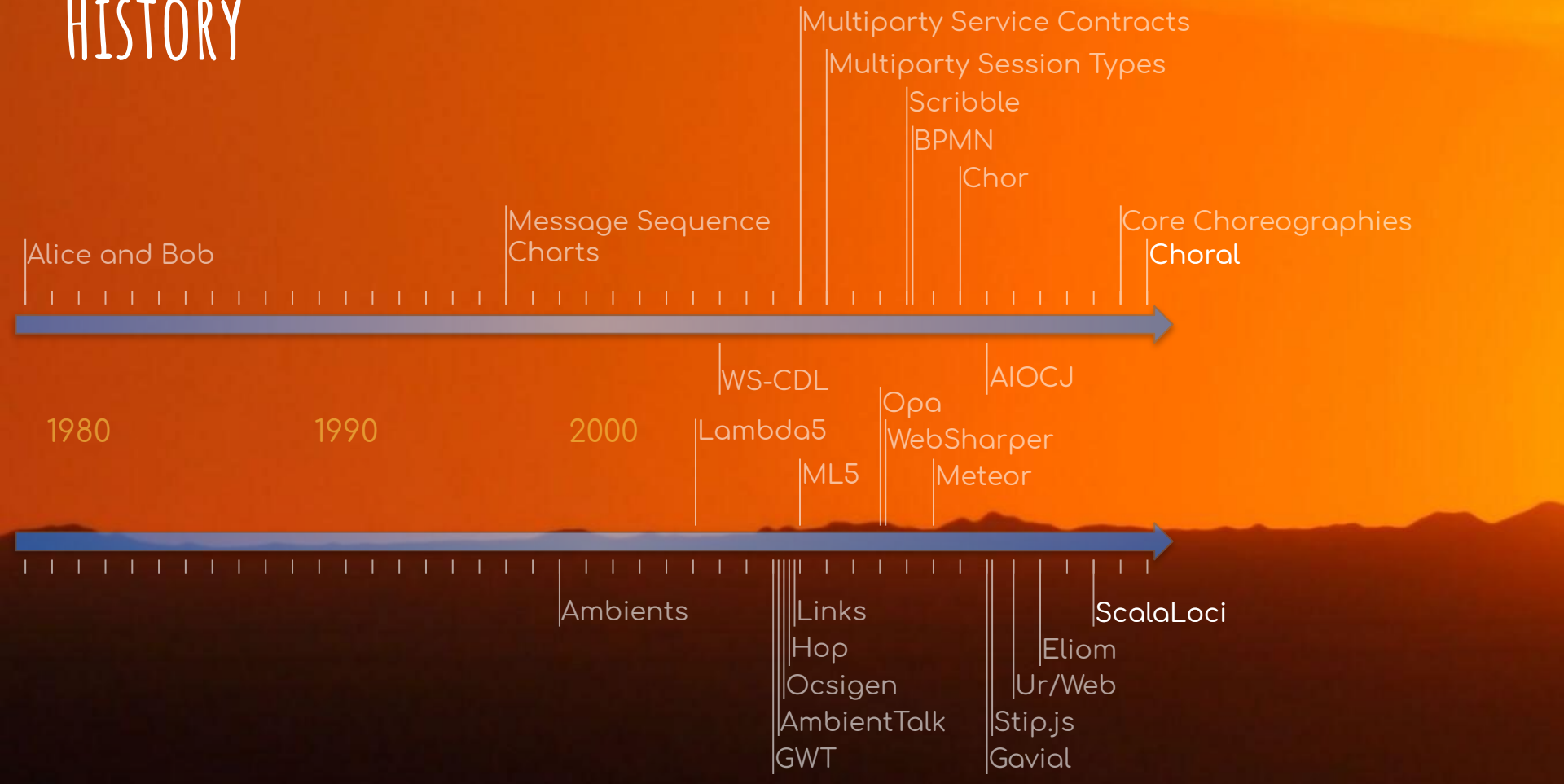
Fractured Design Space

Fractured Design Space

History

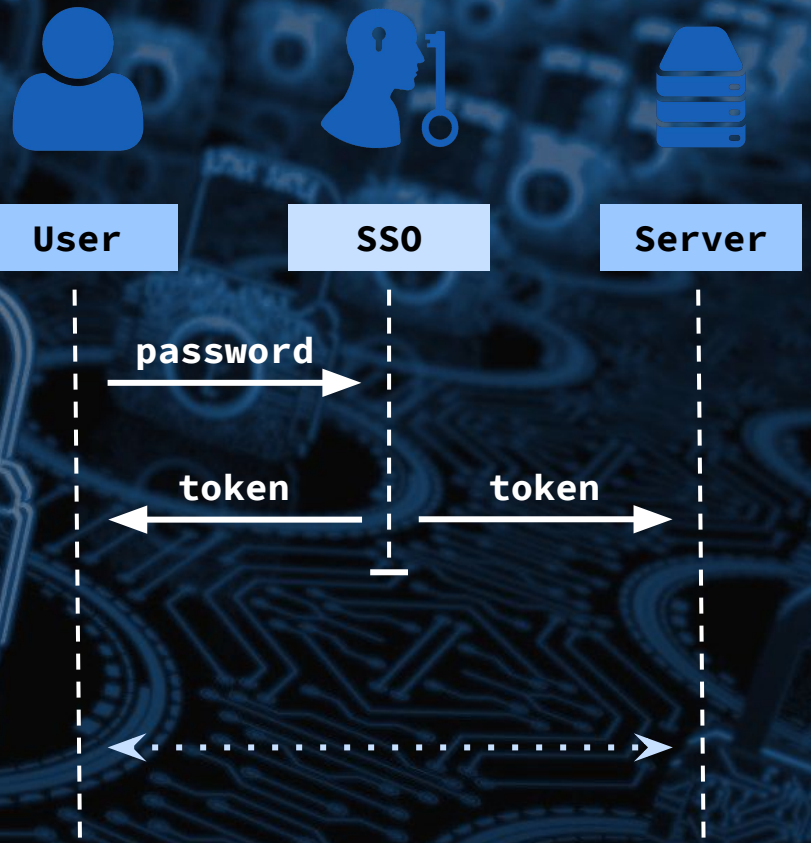

Multiparty Service Contracts
Multiparty Session Types
Scribble
BPMN
Chor
Message Sequence Charts
Alice and Bob
Core Choreographies
Choral
WS-CDL
AIOCJ
Opa
1980
1990
2000
Lambda5
WebSharper
ML5
Meteor
Ambients
Links
Hop
Ocsigen
AmbientTalk
GWT
ScalaLoci
Eliom
Ur/Web
Stip.js
Gavial

History


Multiparty Service Contracts
Multiparty Session Types
Scribble
BPMN
Chor
Message Sequence Charts
Alice and Bob
Core Choreographies
Choral
WS-CDL
AIOCJ
Opa
1980
1990
2000
Lambda5
WebSharper
ML5
Meteor
Ambients
Links
Hop
Ocsigen
AmbientTalk
GWT
ScalaLoci
Eliom
Ur/Web
Stip.js
Gavial

Exemplary Authentication

Single Sign-On Token



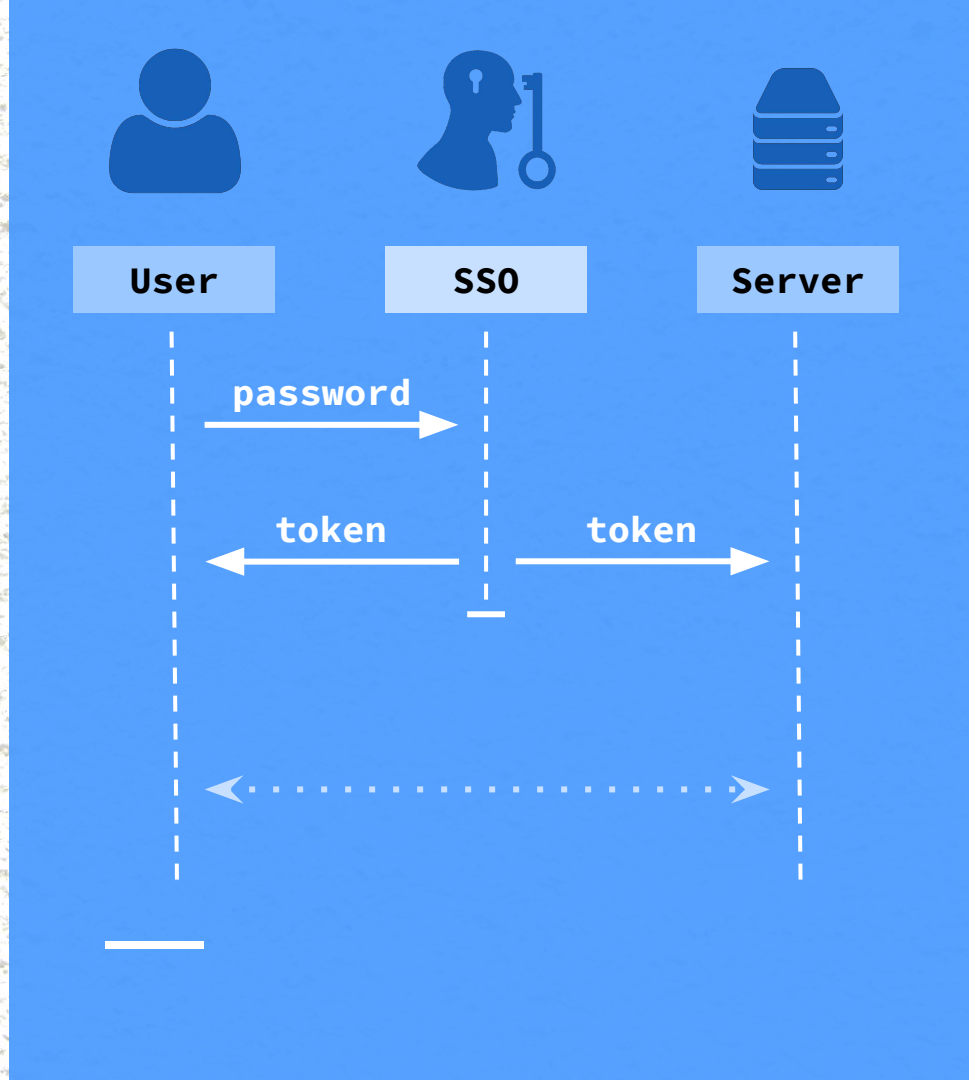
User sends pw to **SSO**

SSO : token = check(pw)

SSO sends token to **Server**

Server : store(token)

SSO sends token to **User**





Choreographies:
Objective view

User sends pw to **SSO**
SSO : token = check(pw)
SSO sends token to **Server**
Server : store(token)
SSO sends token to **User**



Choreographies: Objective view

User sends pw to **SSO**

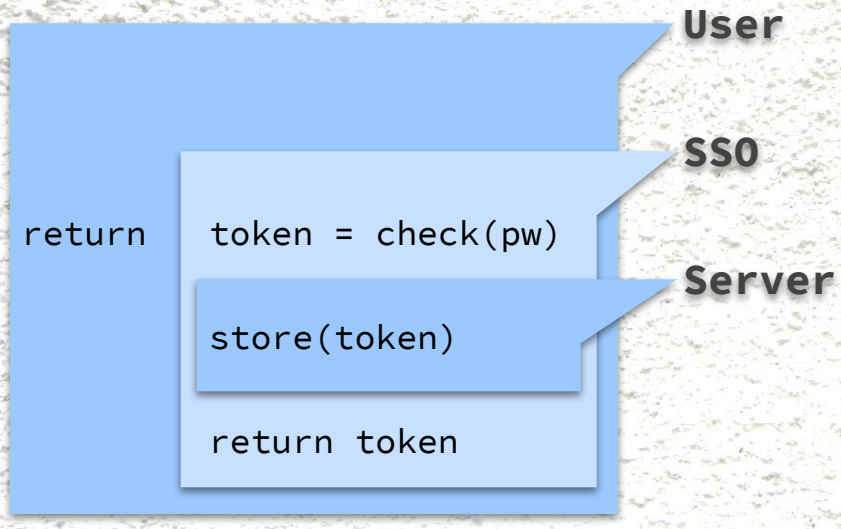
SSO : token = check(pw)

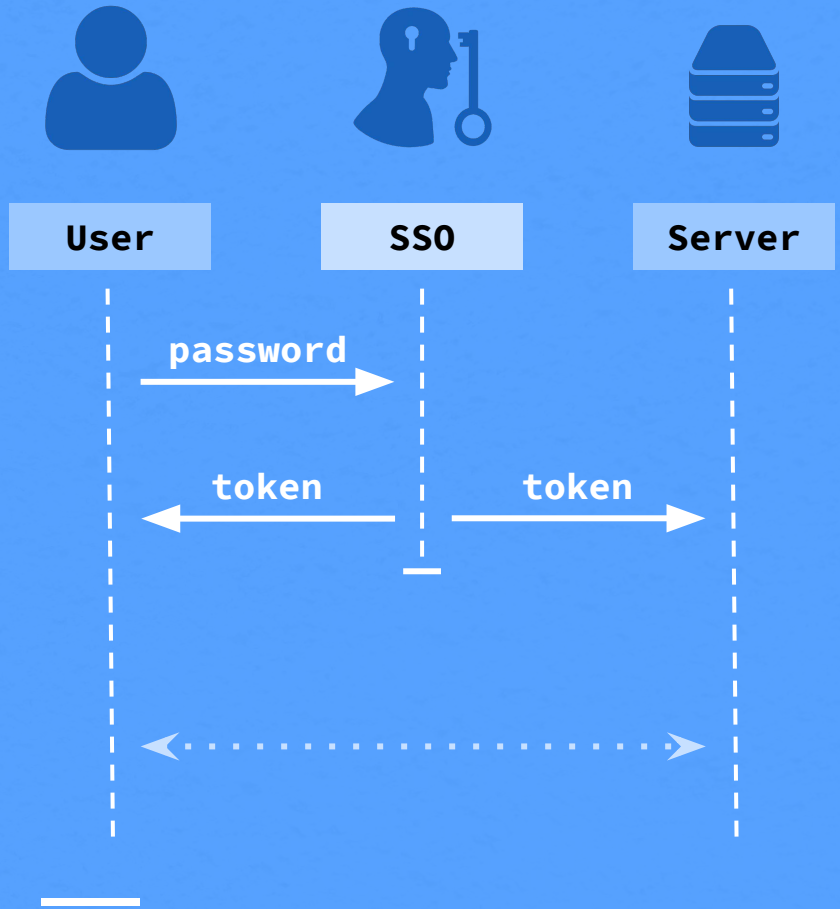
SSO sends token to **Server**

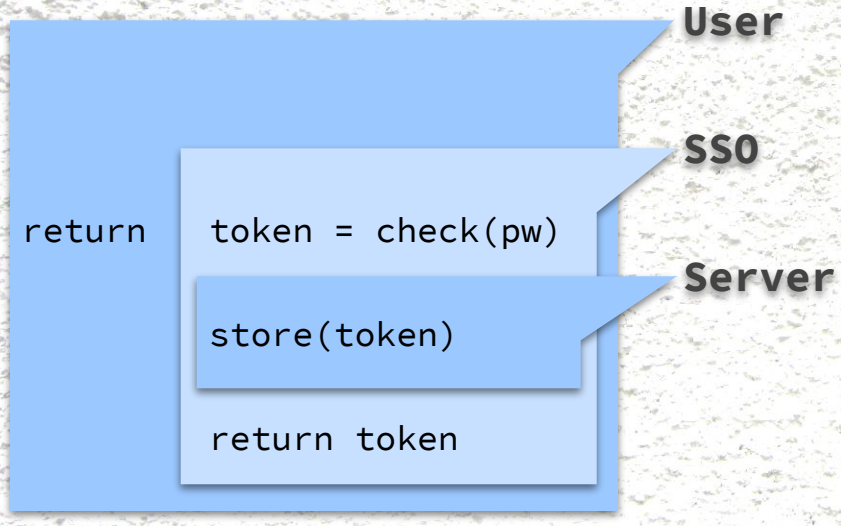
Server : store(token)

SSO sends token to **User**

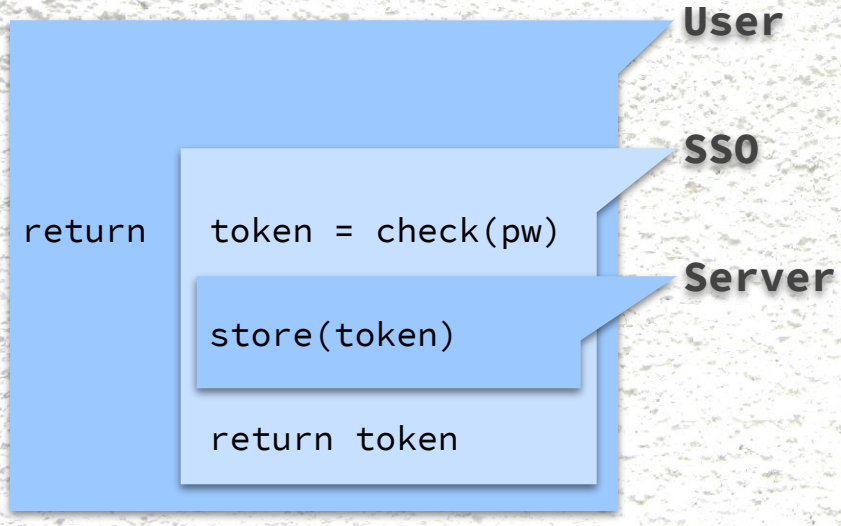
```
class Auth@(User, Server, SSO) {
  SymChannel@(User, SSO) ch1;
  SymChannel@(SSO, Server) ch2;

  Token@User authenticate(String@User pw) {
    Token@SSO token = check(ch1.com(pw));
    store(ch2.com(token));
    return ch1.com(token);
  }
}
```


User
SSO
Server
return
token = check(pw)
store(token)
return token


User
SSO
Server
password
token
token

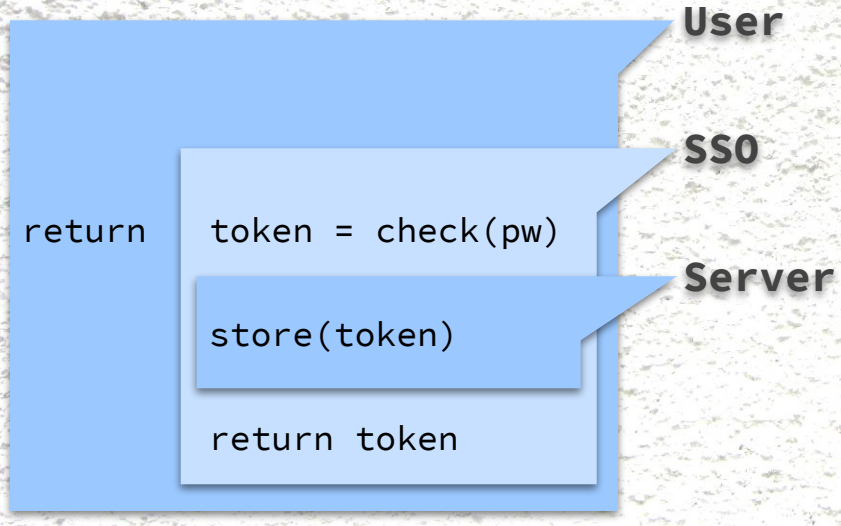


Multitier:
Subjective view



Multitier:
Subjective view

```
on[User] {
 on[SSO].run.capture(pw) {
  val token = check(pw)
  on[Server].run.capture(token) { store(token) }
  token
 }.asLocal
}
```





```
@multitier object Auth {
 @peer type User <: {
  type Tie <: Single[SSO] with Single[Server] }
 @peer type Server <: {
  type Tie <: Single[User] with Single[User] }
 @peer type SSO <: {
  type Tie <: Single[User] with Single[Server] }

 def authenticate(pw: String): Token on User =
  on[User] {
   on[SSO].run.capture(pw) {
    val token = check(pw)
    on[Server].run.capture(token) { store(token) }
    token
   }.asLocal
  }
}
```

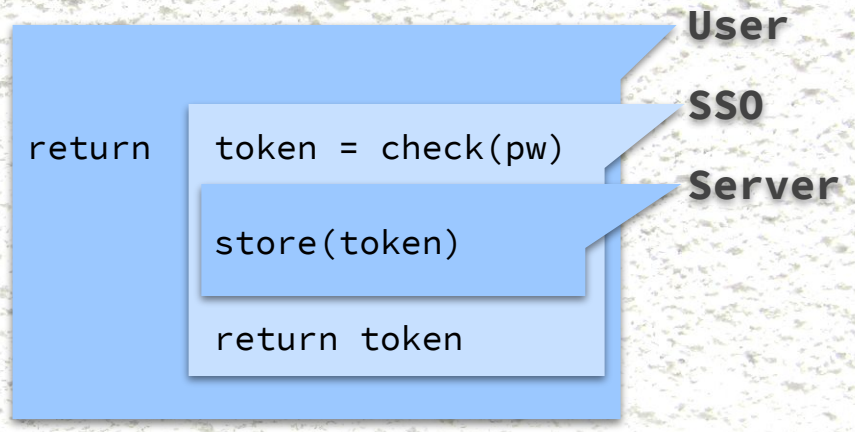
User sends pw to **SSO**

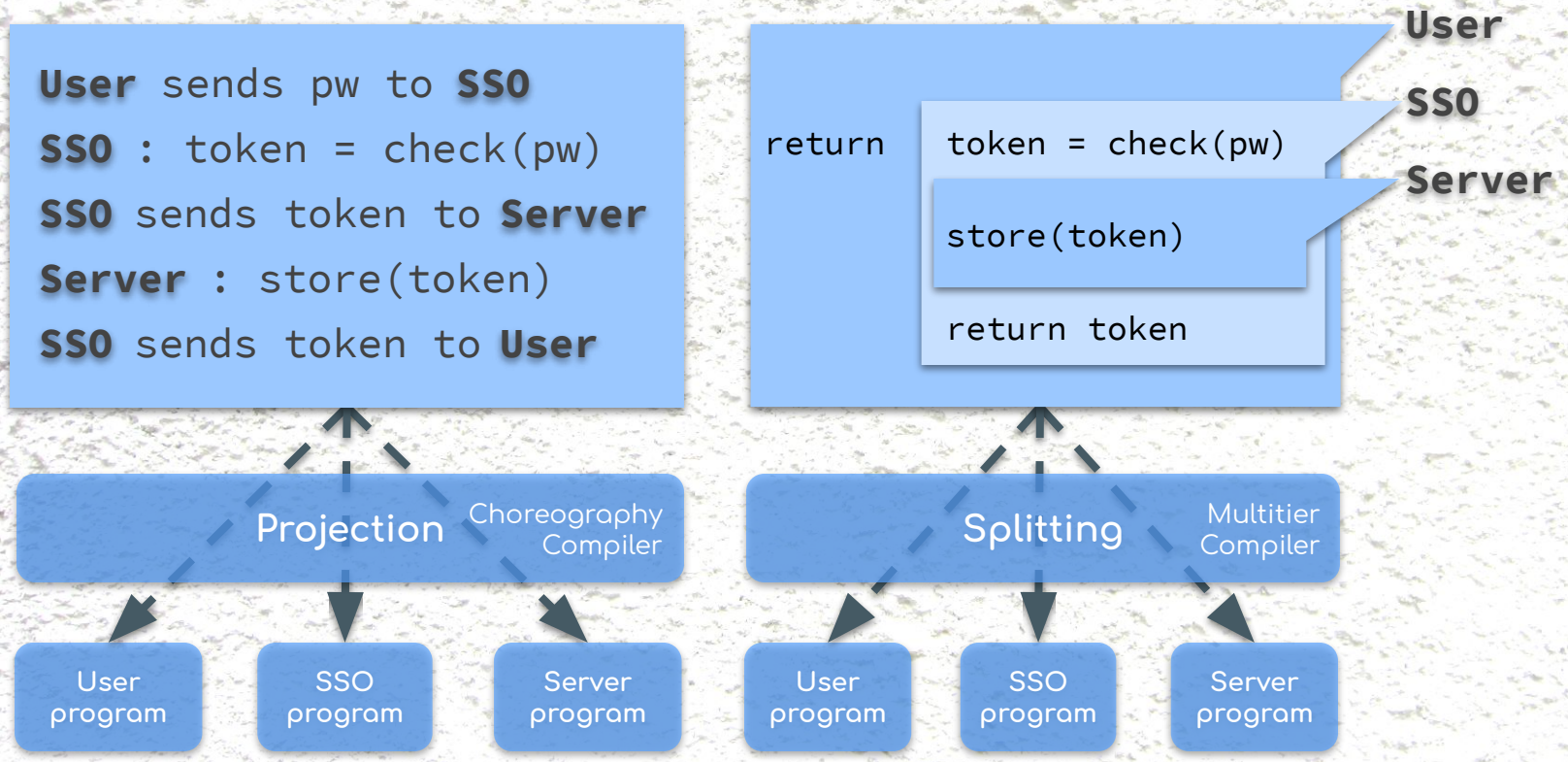
SSO : token = check(pw)

SSO sends token to **Server**

Server : store(token)

SSO sends token to **User**




User sends pw to SSO
SSO : token = check(pw)
SSO sends token to Server
Server : store(token)
SSO sends token to User
Projection
Choreography Compiler
User program
SSO program
Server program
return
token = check(pw)
store(token)
return token
User
SSO
Server
Splitting
Multitier Compiler
User program
SSO program
Server program

... YET BOTH MAP TO THE SAME EXECUTION MODEL

```
User sends pw to SSO
SSO : token = check(pw)
SSO sends token to Server
Server : store(token)
SSO sends token to User
```

```
return   token = check(pw)      User
         store(token)           SSO
         return token           Server
```

Projection — Choreography Compiler

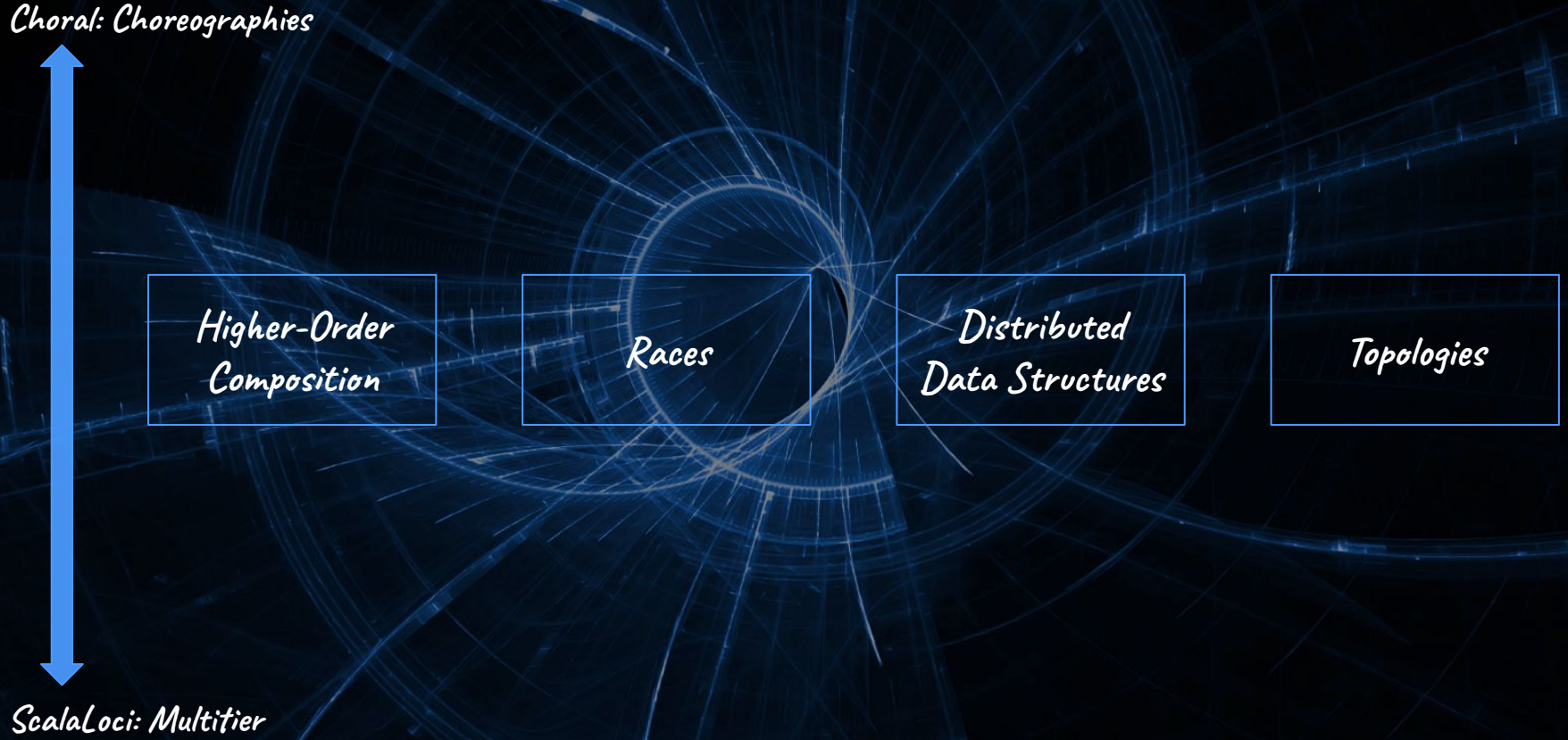
Splitting — Multitier Compiler

User program | SSO program | Server program

User program | SSO program | Server program

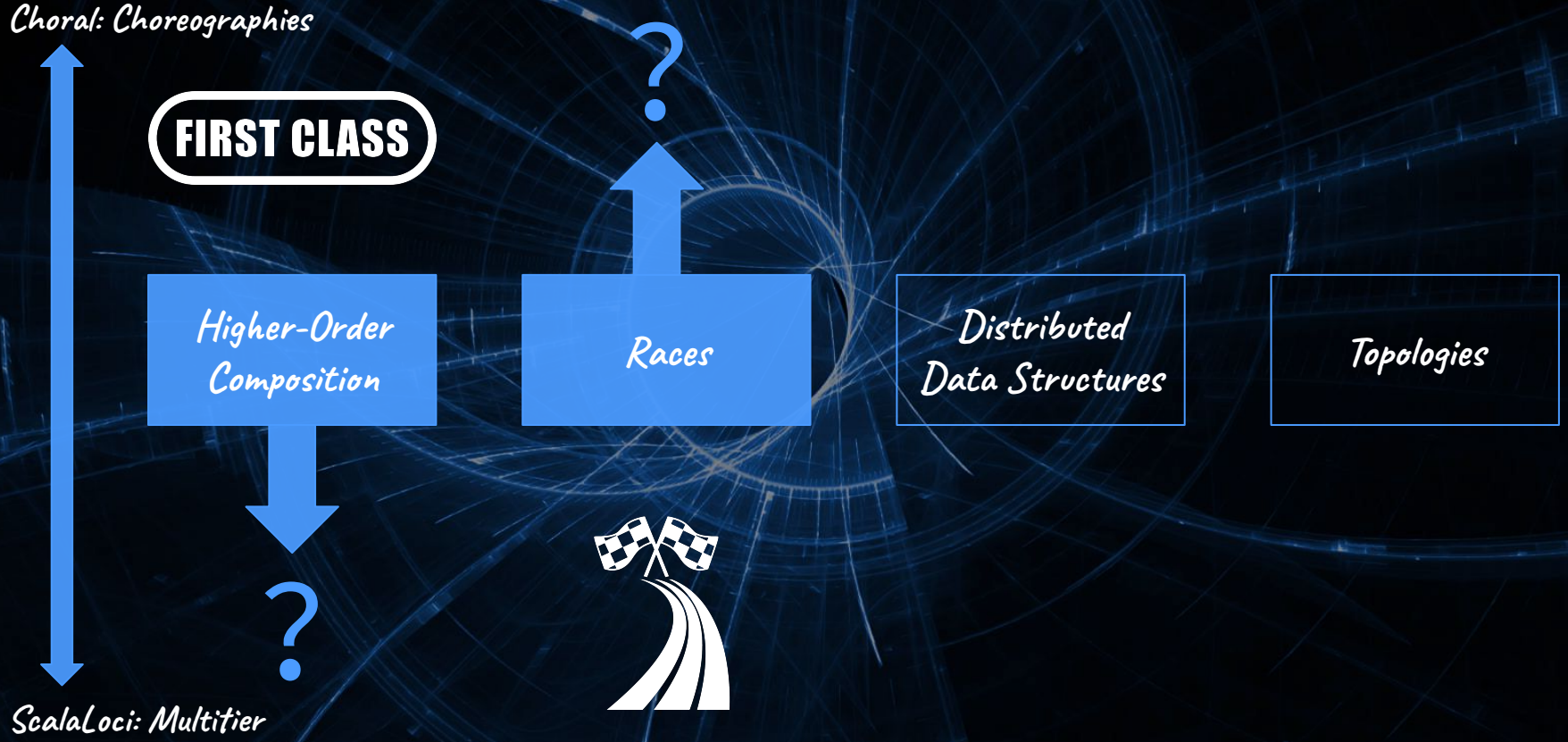
A Unified Perspective: Transferring Features

A Unified Perspective: Transferring Features



A Unified Perspective: Transferring Features

A Unified Perspective: Transferring Features



A Unified Perspective: Transferring Features

Choral: Choreographies

FIRST CLASS

Higher-Order Composition

?

Races

Distributed Data Structures

Topologies

?

?

ScalaLoci: Multitier

A Unified Perspective: Transferring Features

Choral: Choreographies

FIRST CLASS

Higher-Order Composition

?

Races

Distributed Data Structures

?

Topologies

ScalaLoci: Multitier

A Unified Perspective: Transferring Features

Choral: Choreographies

FIRST CLASS

?

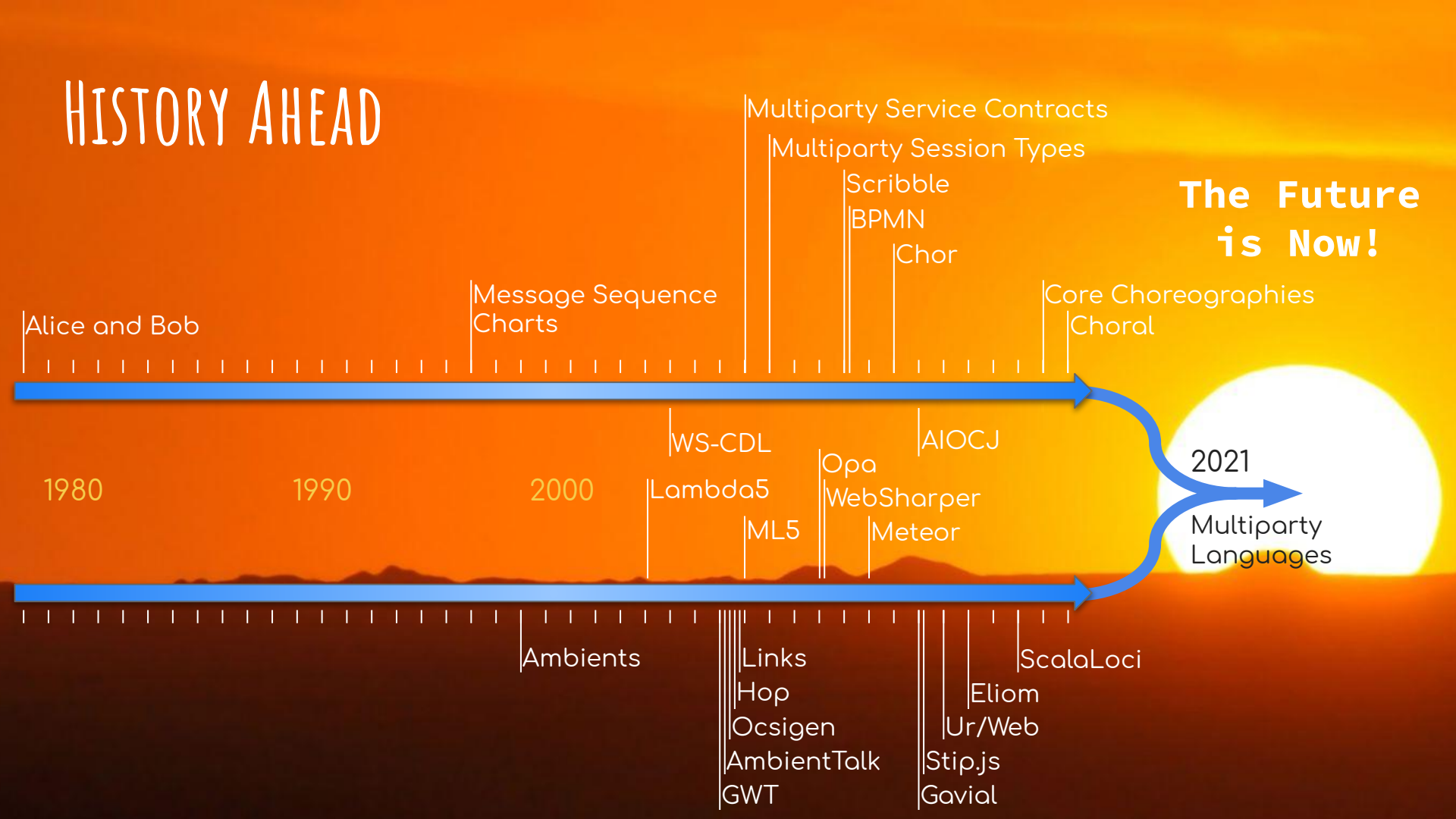
?

Thanks to similar underlying execution models,
features may be ported among multiparty languages

?

?

ScalaLoci: Multitier


History Ahead
Multiparty Service Contracts
Multiparty Session Types
Scribble
BPMN
Chor
The Future is Now!
Message Sequence Charts
Alice and Bob
Core Choreographies
Choral
WS-CDL
AIOCJ
Opa
2021
1980
1990
2000
Lambda5
WebSharper
ML5
Meteor
Multiparty Languages
Ambients
Links
Hop
Ocsigen
AmbientTalk
GWT
ScalaLoci
Eliom
Ur/Web
Stip.js
Gavial

More in the paper:

- Definition of MiniChoral
- Definition of MiniLoci
- Similarities & Translation
- Differences & Transfer

Multiparty Languages: The Choreographic and Multitier Cases

Saverio Giallorenzo
Università di Bologna, Italy
INRIA, Sophia Antipolis, France
Fabrizio Montesi
University of Southern Denmark, Odense, Denmark
Marco Peressotti
University of Southern Denmark, Odense, Denmark
David Richter
Technical University of Darmstadt, Germany
Guido Salvaneschi
University of St. Gallen, Switzerland
Pascal Weisenburger
University of St. Gallen, Switzerland

Abstract

Choreographic languages aim to express multiparty communication protocols, by providing primitives that make interaction manifest. Multitier languages enable programming computation that spans across several tiers of a distributed system, by supporting primitives that allow computation to change the location of execution. Rooted into different theoretical underpinnings—respectively process calculi and lambda calculus—the two paradigms have been investigated independently by different research communities with little or no contact. As a result, the link between the two paradigms has remained hidden for long.

In this paper, we show that choreographic languages and multitier languages are surprisingly similar. We substantiate our claim by isolating the core abstractions that differentiate the two approaches and by providing algorithms that translate one into the other in a straightforward way. We believe that this work paves the way for joint research and cross-fertilisation among the two communities.

2012 ACM Subject Classification Computing methodologies → Distributed programming languages; Theory of computation → Distributed computing models; Software and its engineering → Multi-paradigm languages; Software and its engineering → Concurrent programming languages; Software and its engineering → Distributed programming languages

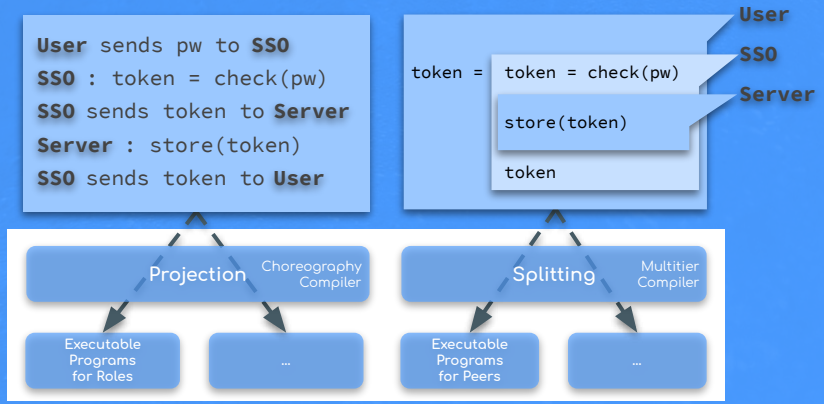
Keywords and phrases Distributed Programming, Choreography Programming, Multitier Programming

Digital Object Identifier 10.4230/LIPIcs.ECOOP.2021.22

Category Pearl

Funding *Fabrizio Montesi*: Villum Fonden, grant no. 29518, and Independent Research Fund Denmark, grant no. 0135-00219.
Marco Peressotti: Villum Fonden, grant no. 29518, and Independent Research Fund Denmark, grant no. 0135-00219.
David Richter: German Federal Ministry of Education and Research (BMBF) iBlockchain Project, grant no. 16KIS0902
Guido Salvaneschi: German Research Foundation (DFG) project no. 383964710, LOEWE initiative (Hesse, Germany) within the Software-Factory 4.0 project, and Swiss National Science Foundation (SNSF) project "Multitier Programming above the Clouds"
Pascal Weisenburger: University of St. Gallen, IPF project no. 1031569

Thanks for your attention



choral-lang.org

scala-loci.github.io

tinyurl.com/multiparty-languages