

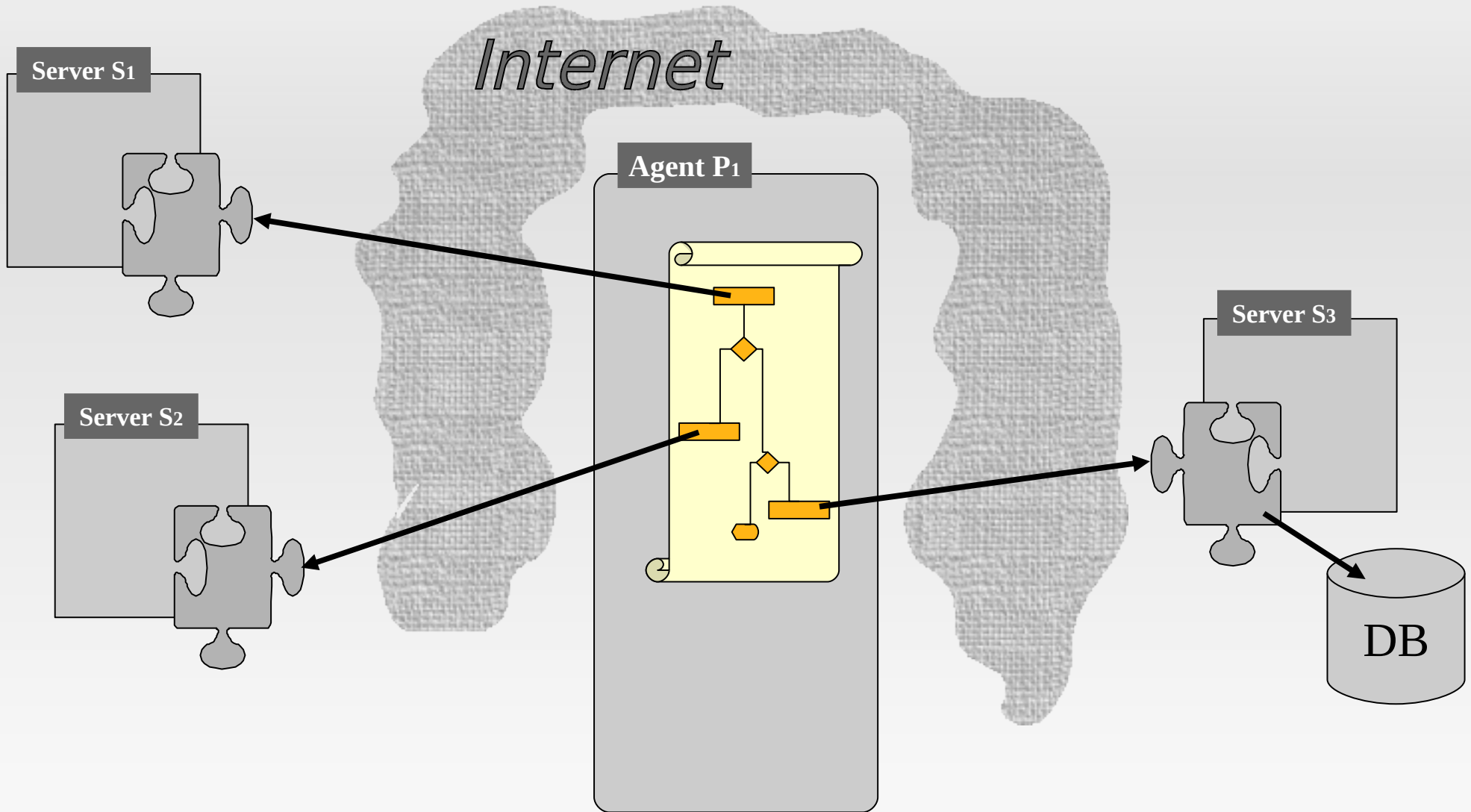
An Executable Calculus for Service Choreography

Paolo Besana, *University of Edinburgh*
Adam Barker, *University of Melbourne*

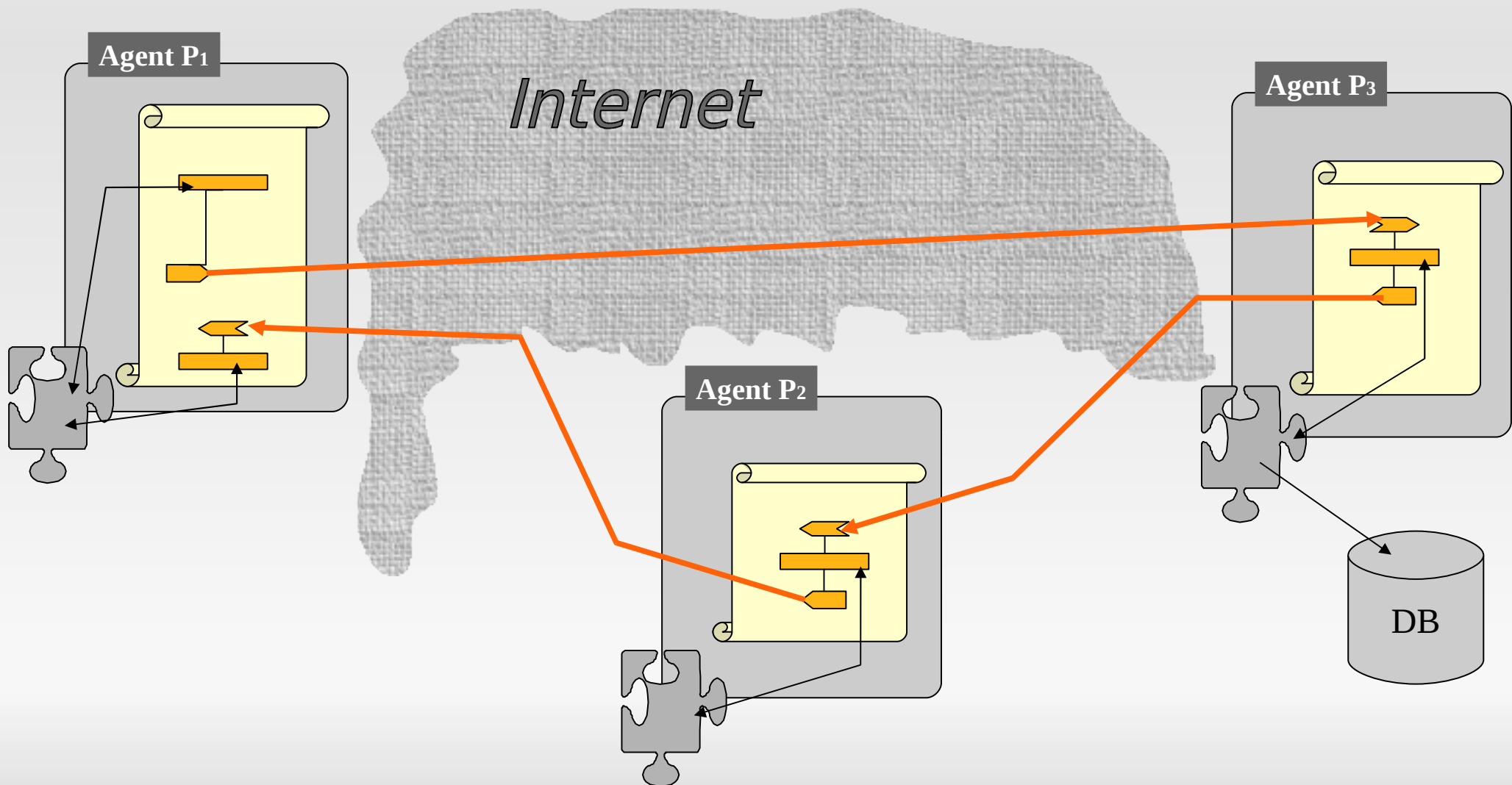
Presentation structure

- Orchestration vs Choreography architectures
- Lightweight Coordination Calculus (LCC) as choreography language
- Limitations and extensions
 - Abstraction mechanism
 - Parallel operator

Orchestration architecture



Choreography architecture



LCC

- LCC is an executable specification language for choreographies
- Based on process calculus:
 - facilitates model checking
- It uses **roles** for agents and **constraints** on message sending to enforce norms
- Similar to Actor model

- The basic behaviours are:
 - $\text{Msg} \Rightarrow a(\text{Role}, \text{AgID})$ for sending a message
 - $\text{Msg} \Leftarrow a(\text{Role}, \text{AgID})$ for receiving a message
 - $\dots \leftarrow \text{constraint}(X, Y \dots)$ for constraints
 - $a(\text{Role}, \text{AgID})$ to change role (also recursively)
- These elementary behaviours can be composed using connectives:
 - then to create sequences
 - or to create choices

LCC example

a (buyer, B) ::

ask (Prdct) \Rightarrow a (seller, S) \leftarrow want (Prdct)

then

price (Prce) \leftarrow a (seller, S)

a (seller, S) ::

ask (Prdct) \leftarrow a (buyer, B)

then price (Prce) \Rightarrow a (buyer, S)

\leftarrow check (Prdct, Prce)

Applications

- Used in EU and UK-funded projects
- Tested in the domains of:
 - Emergency response
 - Bioinformatics
 - Semantic web queries
 - Health informatics
- But mainly for demonstration purposes:
 - No real deployment yet

Limitations

- Extensive testing highlighted some limitations
- Two categories:
 - *Design-time*: dependent on the LCC language
 - *Execution-time*: dependent on the implemented platform
- We will discuss only design-time limitations:
 - Lack of mechanism for separating different abstraction levels
 - Lack of parallel operator

Lack of Abstraction mechanism

- An interaction model contains activities and messages at all levels of abstractions
- Work-around:
 - Start new interaction from a constraint
- BUT:
 1. Interaction cannot be specified at design time
 2. Cannot specify the participants

Abstraction with scenes

New operator:

```
scene(scenename, rolename)
```

- Specifies that the peer needs to participate in a new interaction, defined by scenename, taking the role rolename
- All peers that encounter the operator participate in the same sub-interaction

Lack of parallel operator

- Work-around:
 - Have a role subscribed by many peers and send messages (non-blocking activity) to all of them
- BUT:
 1. Cannot be defined at design-time
 2. Max number of possible parallel processes cannot be decided after start of interaction

Introducing parallelism

- Normal change of role operator ($a(\text{Role}, R)$) is blocking
- We introduced a new role change, non blocking
 - Creates a new process, with own queues, run inside the peer that executed the operation
 - Compatible with agent model and with PI-calculus
 - If spawner process terminates, spawned processes terminates as well

Evaluation

- We used Service Interaction Patterns by Barros
- We focused on representing interaction patterns with time-frames
- Complex and awkward with classic LCC
 - Compact and intuitive with extensions
 - Timers created by spawned roles

Thanks for your attention

Any questions?