The Chor programming language

Fabrizio Montesi
<fmontesi@imada.sdu.dk>

Background and Motivations

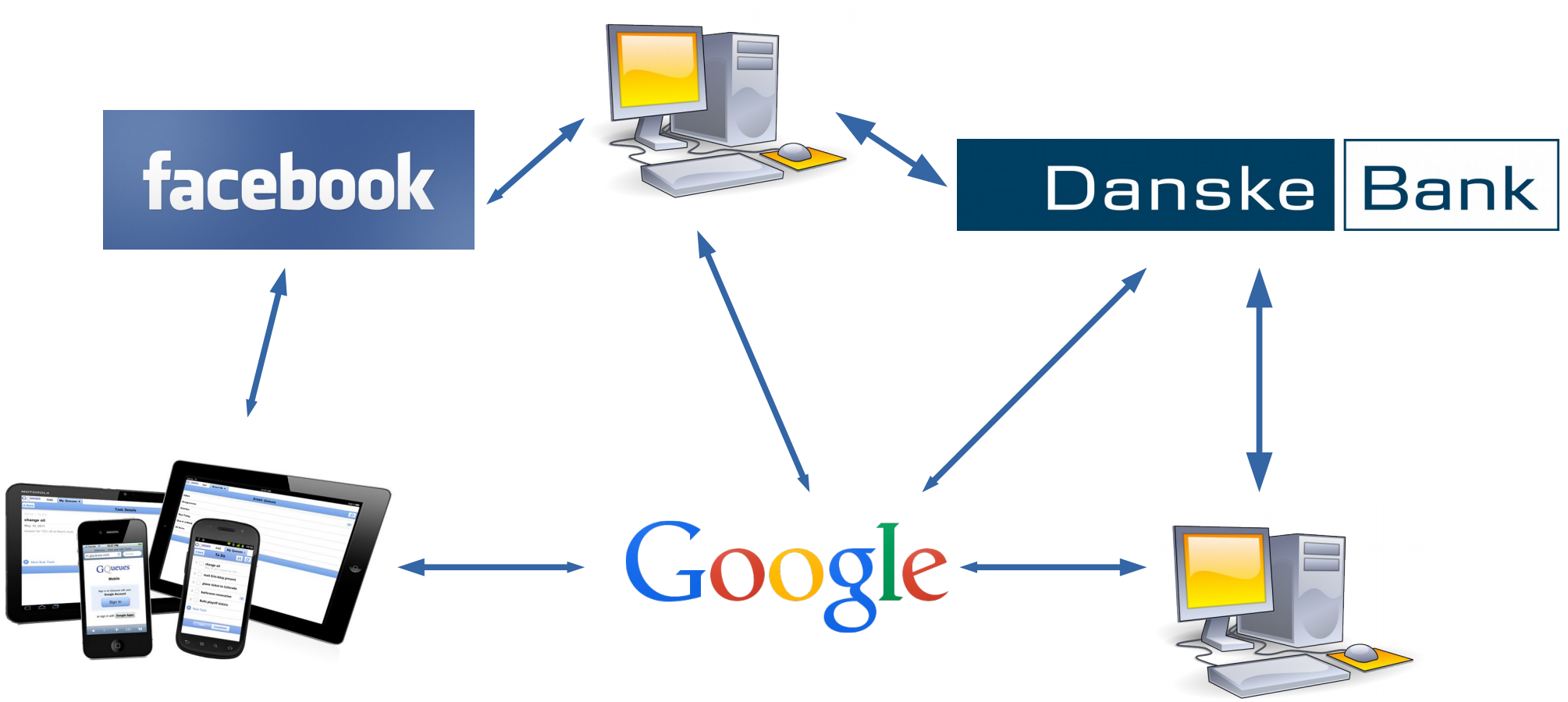
Distributed Systems

- **Distributed system:**
  a network of endpoints that communicate by exchanging messages.

- Widespread! Let's see some examples...

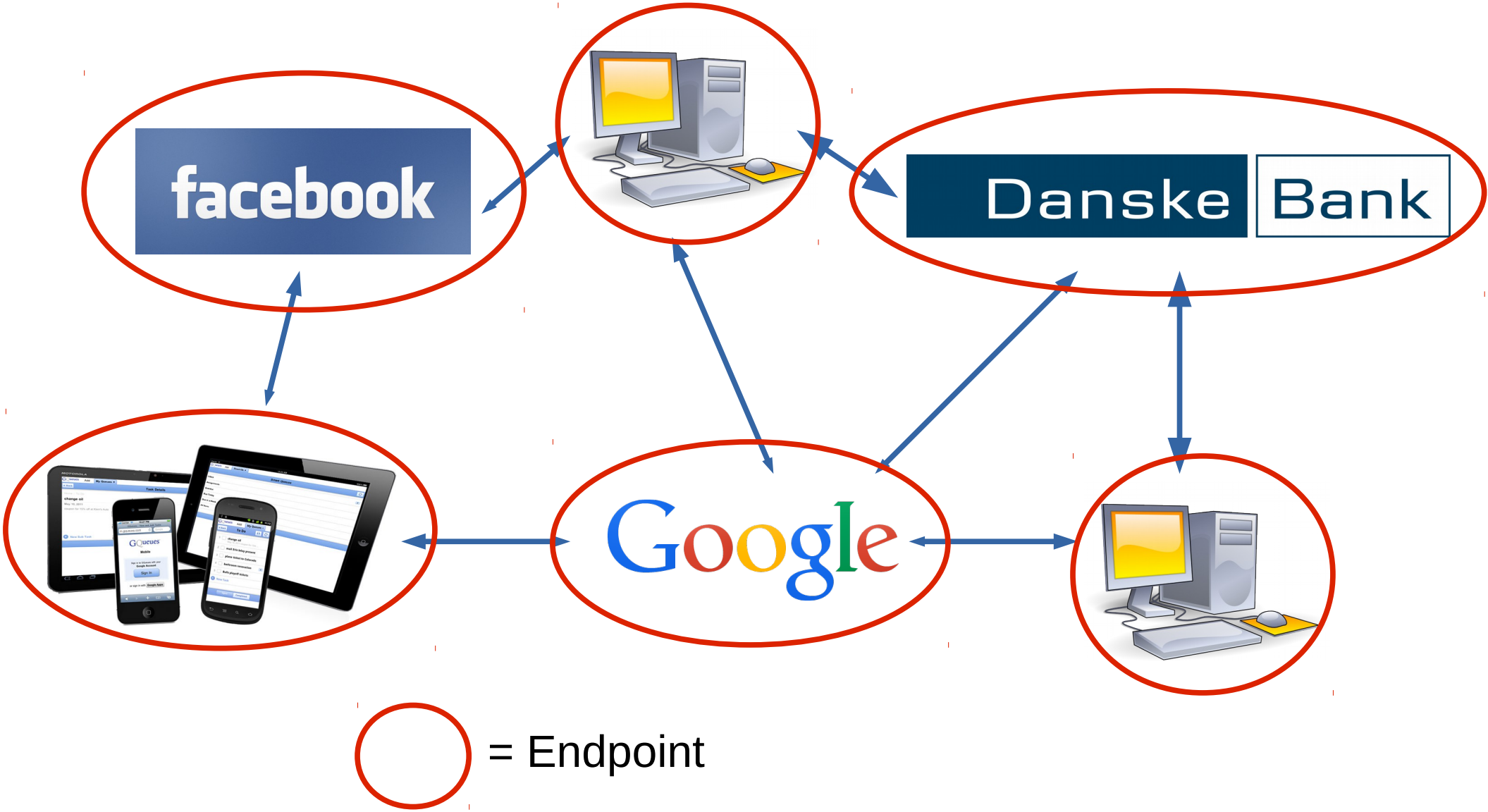
The Internet

- The Internet is a distributed system:



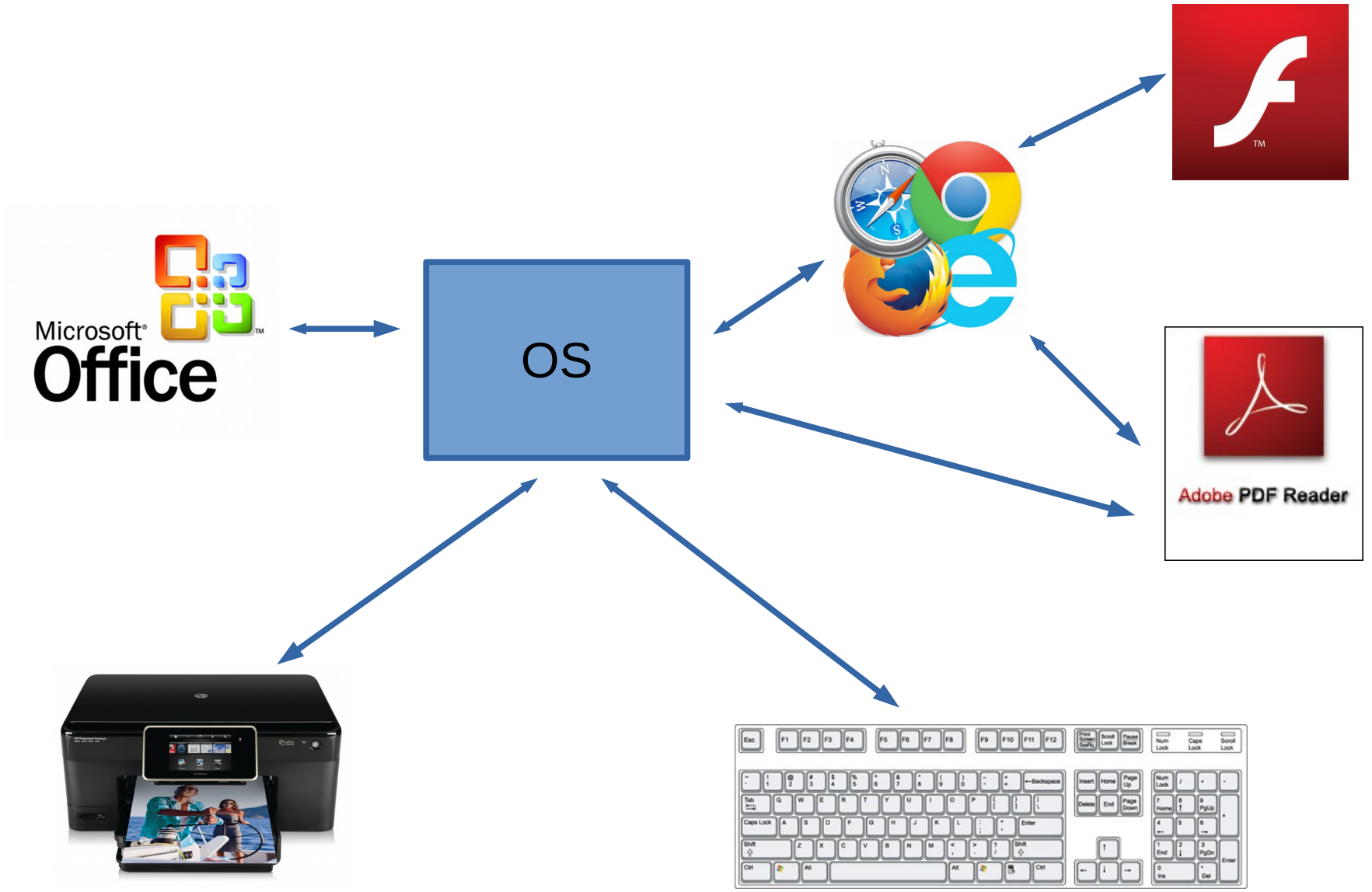
The Internet

- The Internet is a distributed system:



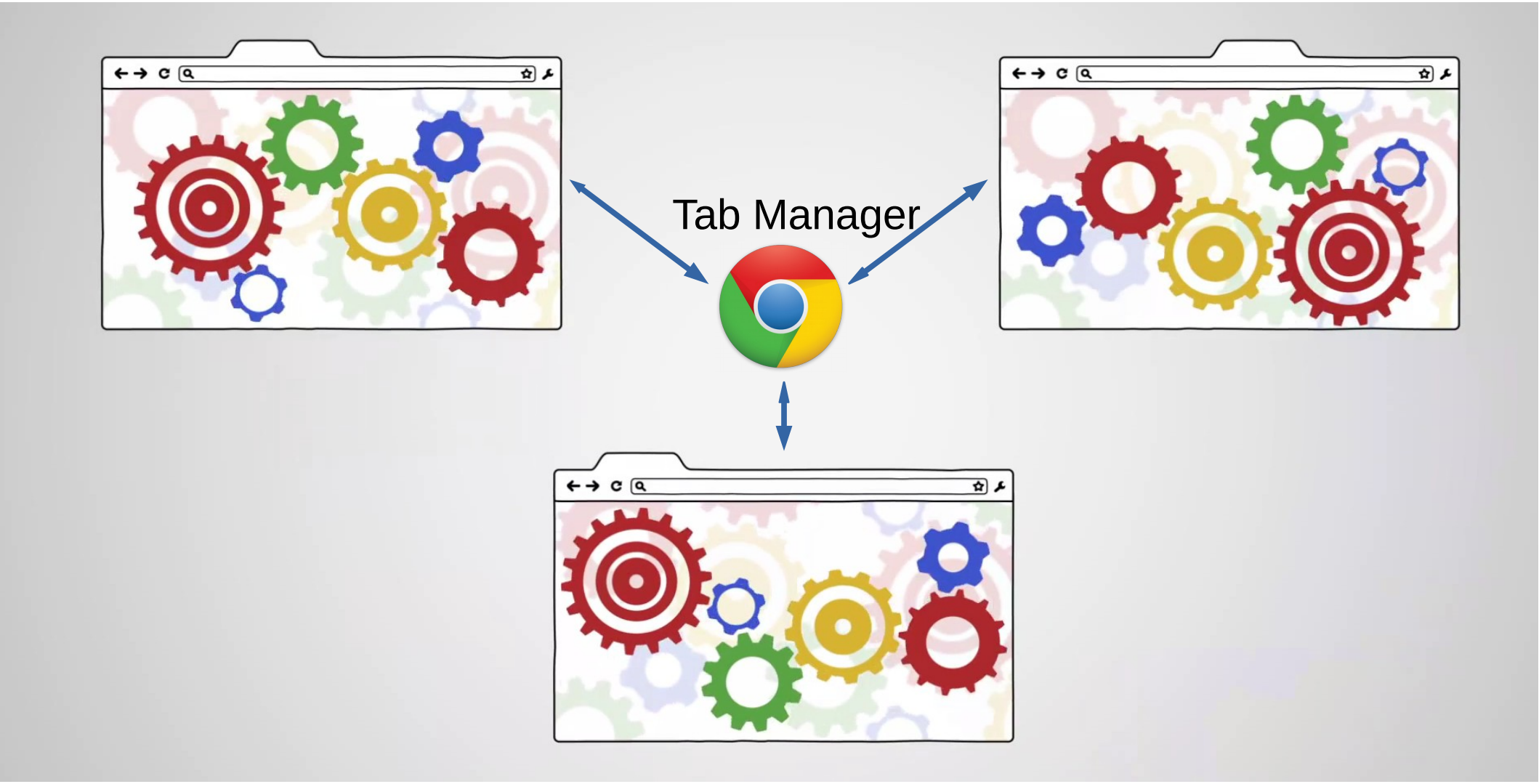
Your Computer

- The OS and apps in your computer (or phone):



Your browser

- Even applications can be distributed systems. Google Chrome:



Distributed systems are big!

System	Number of endpoints
My computer	160
A house	Hundreds
A company	Thousands (or millions)
The Internet	At least 20 billions

Endpoint Programming

- How do we program all these endpoints?
- We write a program for each.
- Programs interact by sending and receiving messages.

Endpoint Programming: example

- Alice wants to know the price of a book from Amazon.

Alice

```
send "rabbits"   to    Amazon

recv price       from  Amazon
```

Amazon

```
recv book              from  Alice

send price(book)       to    Alice
```

Deadlocks

- Endpoint programming is error-prone.

Deadlocks

- Endpoint programming is error-prone.

Alice

```
send "rabbits"   to   Amazon

recv price       from Amazon
```

Amazon

```
recv book          from Alice

send price(book)   to   Alice
```

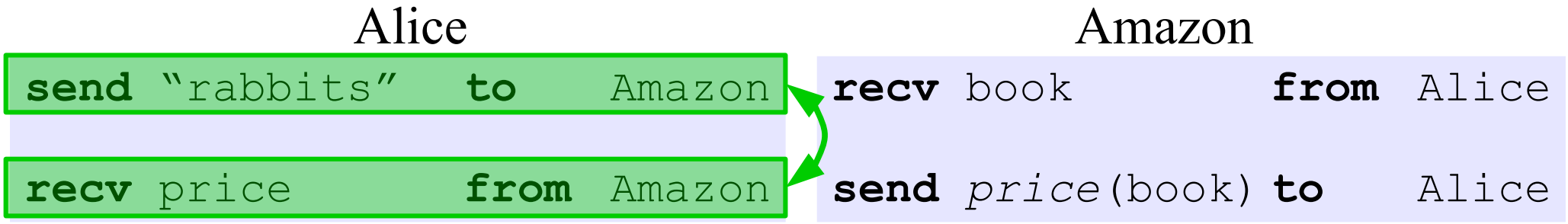
Deadlocks

- Endpoint programming is error-prone.

Alice	Amazon
send "rabbits" **to** Amazon	**recv** book **from** Alice
recv price **from** Amazon	**send** *price*(book) **to** Alice

Deadlocks

- Endpoint programming is error-prone.



Deadlocks

- Endpoint programming is error-prone.

- Mismatching of input/output actions leads to a deadlock.

Alice

```
recv price          from Amazon

send "rabbits"      to   Amazon
```

Amazon

```
recv book           from Alice

send price(book)    to   Alice
```

Deadlocks

- Endpoint programming is error-prone.

- Mismatching of input/output actions leads to a deadlock.

Alice

```
recv price          from  Amazon

send "rabbits"      to    Amazon
```

Amazon

```
recv book           from  Alice

send price(book)    to    Alice
```

- Creating deadlocks is easy.

- Detecting deadlocks is hard [Kobayashi, 06].

Famous bugs

- Therac-25: a machine for radiation therapy in the 80s.
- Unsafe communications caused excessive amounts of radiation (100x).
- At least 6 accidents, 3 deaths.

Famous bugs

- 2003: Blackout in Northeast America.
- Started from a communication bug in a monitoring station.
- Affected 55 million people.
- 11 deaths.
- At least 7 billion USD lost.

Safety

- In general, we would like systems to be **safe**.

- **Safe** = no bugs given by wrong sending/receiving actions.

How does it happen?

- Human error.

How does it happen?

- Human error.

- More quality control?

Checking for bugs

Alice

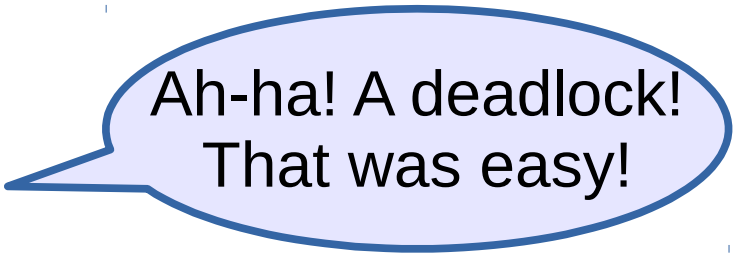
```
recv price          from  Amazon
send "rabbits"      to    Amazon
```

Amazon

```
recv book           from  Alice
send price(book)    to    Alice
```

Checking for bugs

Alice			Amazon		
recv price	**from**	Amazon	**recv** book	**from**	Alice
send “rabbits”	**to**	Amazon	**send** *price*(book)	**to**	Alice



Checking for bugs

Alice

```
send "rabbits"   to     Amazon
recv price       from   Amazon
send price       to     Bob
recv address     from   Charlie
recv text        from   Bob
```

Amazon

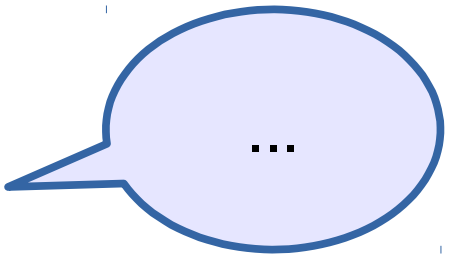
```
recv book          from   Alice
send price(book)   to     Alice
recv price(book)   from   Charlie
send text(book)    to     Charlie
```

Bob

```
recv price       from   Alice
recv text        from   Charlie
send text        to     Alice
```

Charlie

```
recv price             from   Alice
send money(price)      to     Amazon
recv text              from   Amazon
send address           to     Alice
```



Checking for bugs

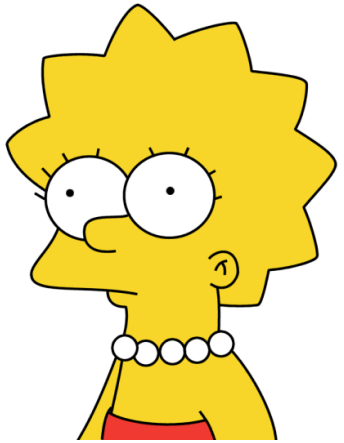
```
send "rabbits"      to   Amaz
recv price          from Amaz
send price          to   Bob
recv address        from Char
recv text           from Bob
send "horses"       to   Amaz
recv price2
send "spiders"
send price3
recv address1
recv price
send money(pri
recv text
send address
recv price
send money(pri
recv text
send address
recv text2
recv price
send price
recv address
recv text
```

```
recv price
recv text
recv price
recv text
send text
send text
recv price
send money(pri
recv price
send mo
recv te
send ad
recv te
send ad
recv pr
recv pr
recv te
send te
recv te
send te
```

```
recv book           from Alice
send price(book)    to   Alice
recv price(book)    from Charlie
recv price          from Alice
send money(price)   to   Amazon
recv text           from Amazon
send address        to   Alice
recv price          from Alice
recv price          from Alice
send money(price)   to   Amazon
recv text           from Amazon
send address        to   Alice
send mone
recv text
send addr
```

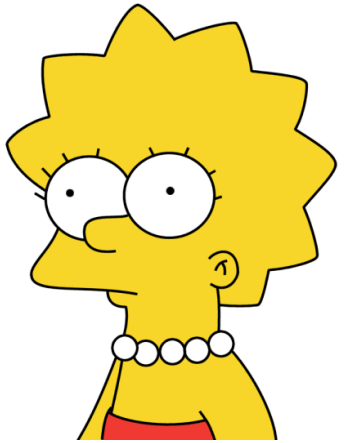
```
recv price                from Alice
send money(price)         to   Amazon
                          from Amazon
                          from Alice
                          from Alice
)                         to   Amazon
                          from Alice
)                         to   Amazon
                          from Amazon
                          to   Alice
                          from Amazon
                          to   Alice
)                         to   Amazon
                          from Amazon
                          to   Alice
                          to   Alice
```

```
send "rabbits"      to   Amazon
recv price          from Amazon
send price          to   Bob
recv address        from Charlie
recv text           from Bob
send "horses"       to   Amazon
recv price2         from Amazon
send "spiders"      to   Amazon
send price3         to   Bob
recv address1       from Charlie
recv price          from Alice
send money(price)   to   Amazon
recv text           from Amazon
send address        to   Alice
recv price          from Alice
send money(price)   to   Amazon
recv text           from Amazon
send address        to   Alice
```



Checking for bugs

```
send "rabbits"      to   Amazon
recv price          from Amazon
send price          to   Bob
recv address        from Charlie
recv text           from Bob
send "horses"       to   Amazon
recv price2         from Amazon
send "spiders"      to   Amazon
send price3         to   Bob
recv address1       from Charlie
recv price          from Alice
send money(price)   to   Amazon
recv text           from Amazon
send address        to   Alice
recv price          from Alice
send money(price)   to   Amazon
recv text           from Amazon
send address        to   Alice
```



Distributed bugs are hard to spot!

- Avoiding distributed bugs, like deadlocks, is hard [Kobayashi, 06].

- We need **tools** to deal with this.

Problem

Developing safe distributed systems by programming endpoints separately is error-prone.

Choreographic Programming: the idea

Choreography

- A **single** program for defining the behaviour of many endpoints [W3C, 05]

- Our previous example as a choreography:

```
alice.”rabbits”     → amazon.book ;
amazon.price(book)  → alice.price
```

Choreography

- A **single** program for defining the behaviour of many endpoints [W3C, 05]

- Our previous example as a choreography:

```
alice.”rabbits”      → amazon.book ;
amazon.price(book)   → alice.price
```

Choreography

- A **single** program for defining the behaviour of many endpoints [W3C, 05]

- Our previous example as a choreography:

```
alice.”rabbits”      → amazon.book ;
amazon.price(book)   → alice.price
```

Choreography

- A **single** program for defining the behaviour of many endpoints [W3C, 05]

- Our previous example as a choreography:

```
alice.”rabbits”      → amazon.book ;
amazon.price(book)   → alice.price
```

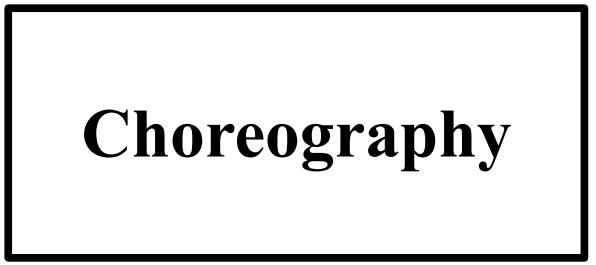
- Defines **what** communications we want to happen, rather than **how** to implement them.

Choreographic Programming

- [Mendling and Hafner, 05] [Qiu et al., 07] [Carbone et al., 07] [Lanese et al., 08] ...

Choreographic Programming

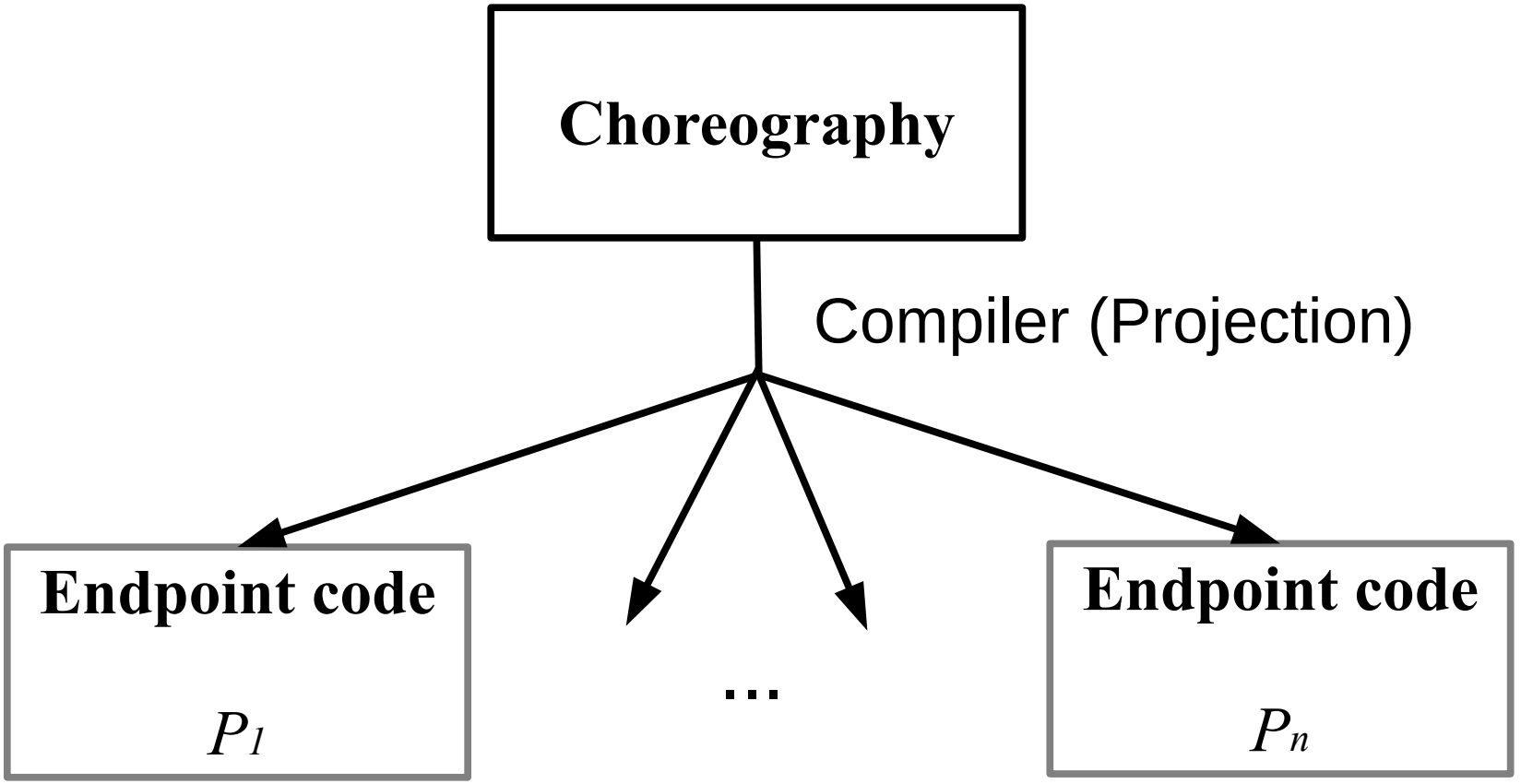
- Write a choreography.



Choreographic Programming

- Write a choreography.

- Compile it to an executable distributed implementation.

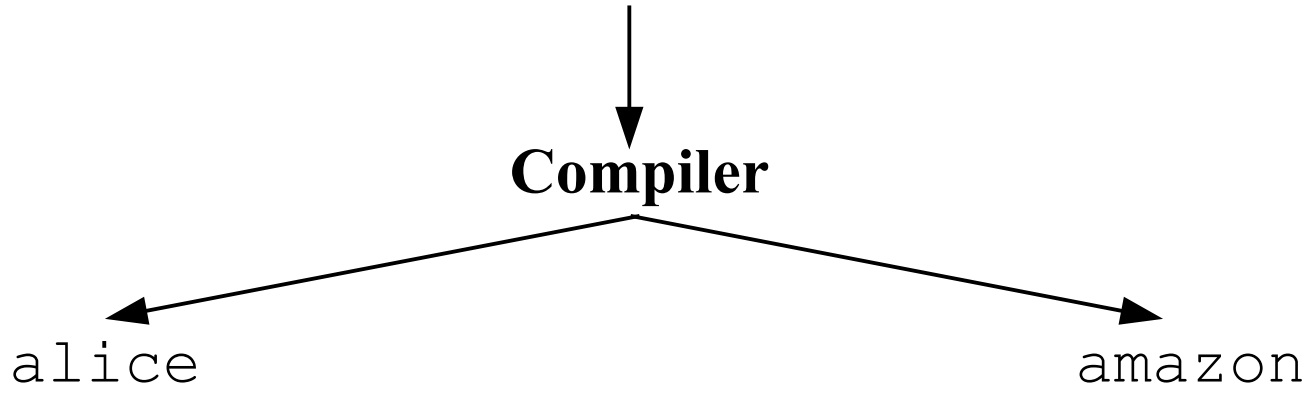


Example of projection

```
alice.”rabbits”     →  amazon.book ;
amazon.price(book)  →  alice.price
```

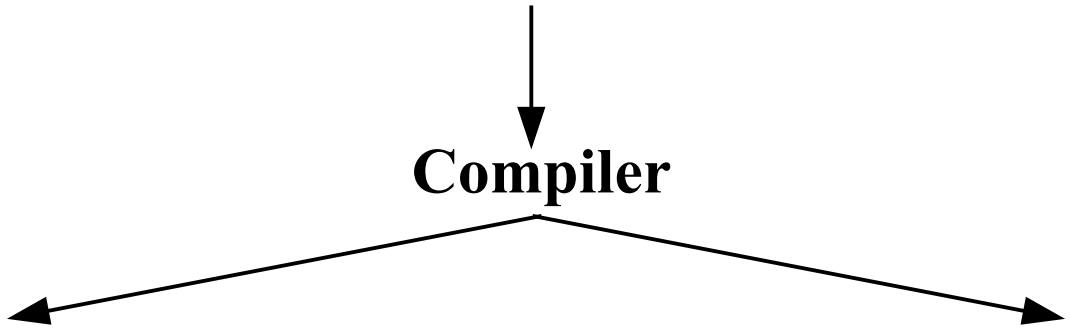
Example of projection

```
alice.”rabbits”     →  amazon.book ;
amazon.price(book)  →  alice.price
```



Example of projection

```
alice.”rabbits”     →  amazon.book ;
amazon.price(book)  →  alice.price
```



alice

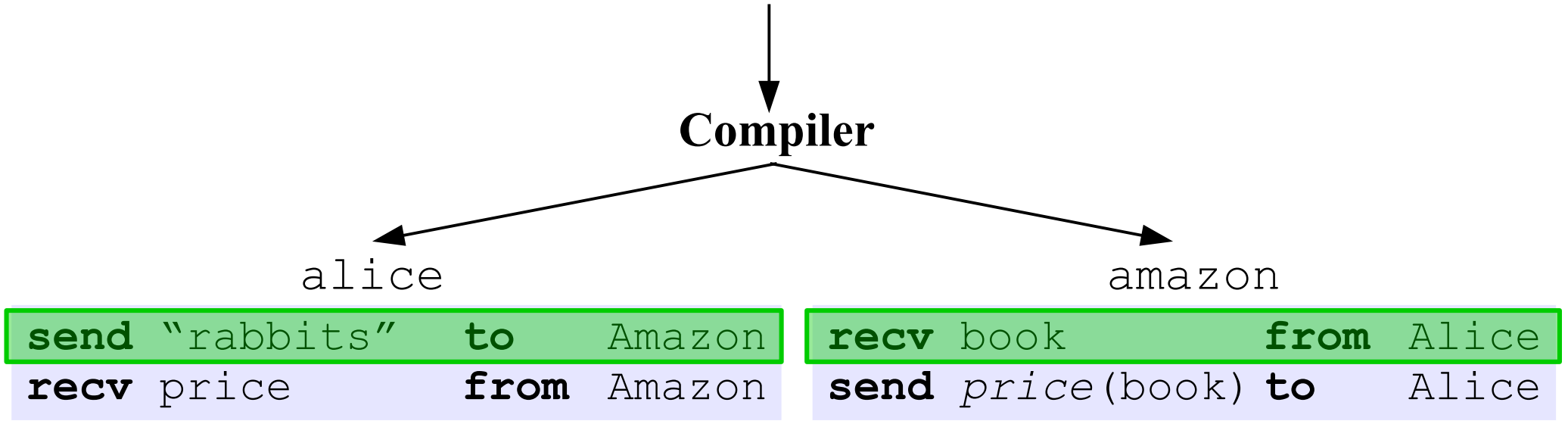
```
send “rabbits”   to    Amazon
recv price       from  Amazon
```

amazon

```
recv book          from  Alice
send price(book)   to    Alice
```

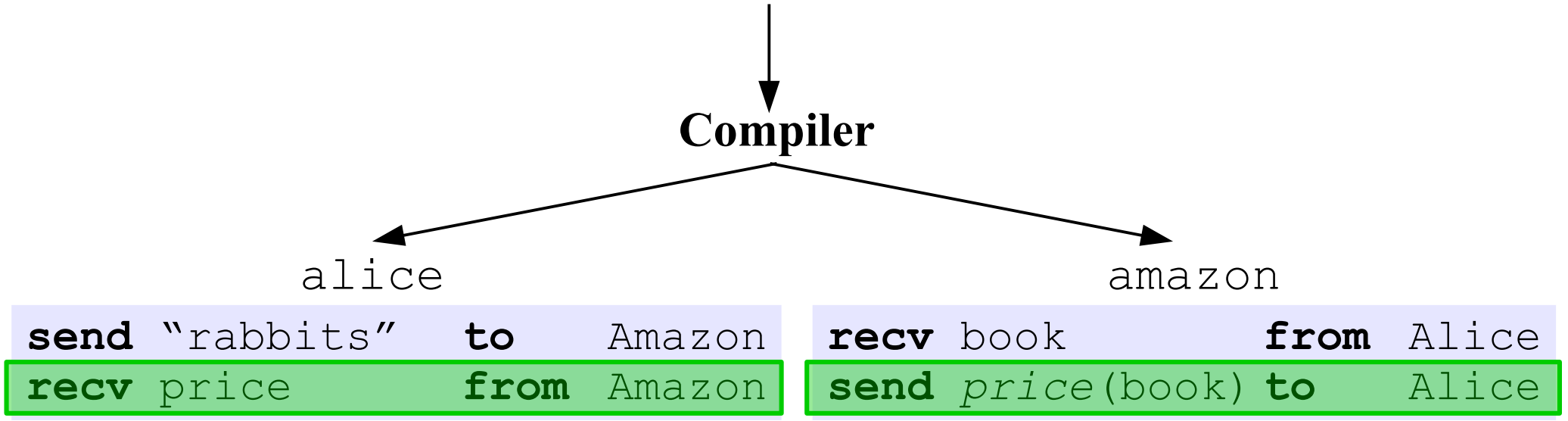
Example of projection

```
alice.”rabbits”      →  amazon.book ;
amazon.price(book)   →  alice.price
```



Example of projection

```
alice."rabbits"      →  amazon.book ;
amazon.price(book)   →  alice.price
```



Correct by construction!

- Correct pairing of I/O actions prevents deadlocks!
- Promising approach.
- Instead of detecting deadlocks after programming, prevent deadlocks from being written.

The momentum of choreographies

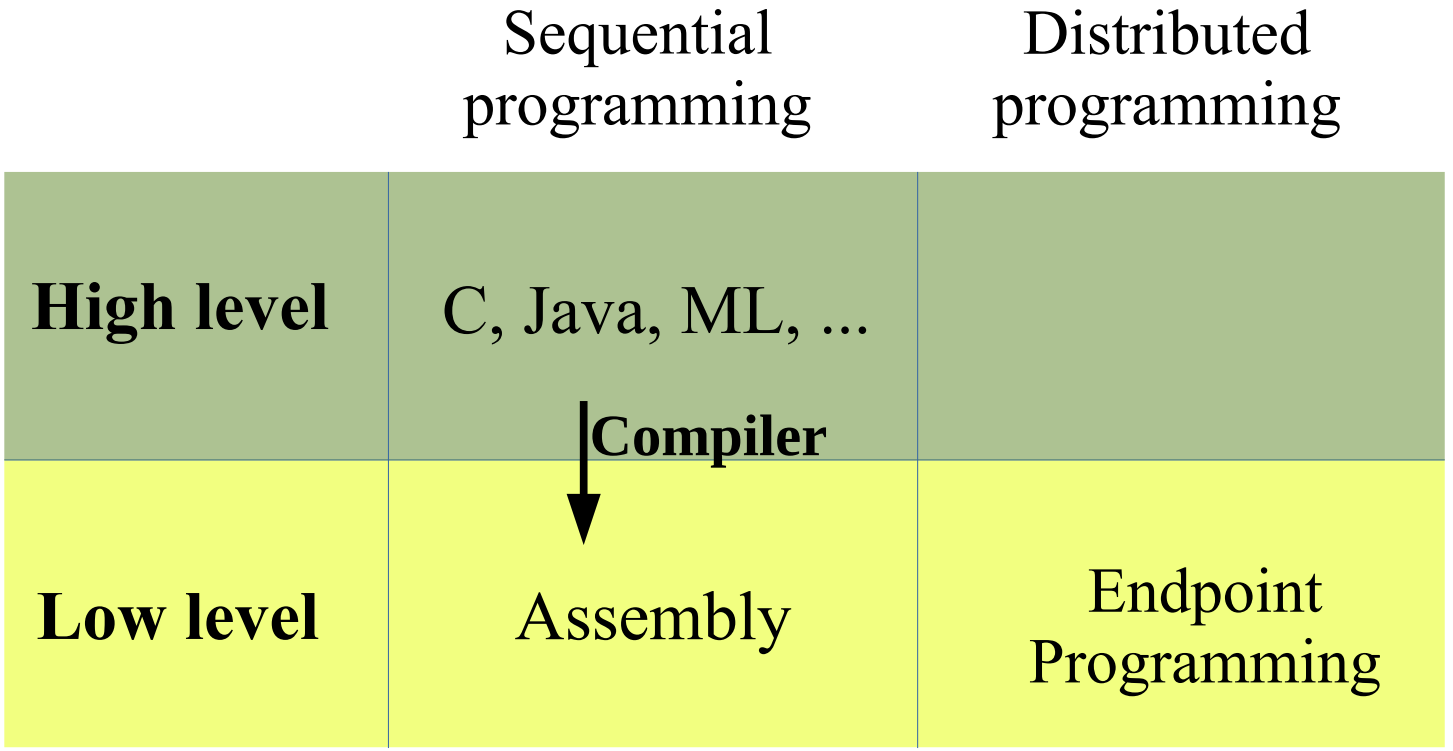
- Great momentum: there are lots of choreography models out there.

  [Busi et al., 05] [Busi et al., 06] [Qiu et al., 07] [Bravetti and Zavattaro, 07] [Carbone et al., 07] [Lanese et al., 08] [Basu et al., 11] [Dalla Preda et al., 14] …

- Bisimulation, session types, web services, adaptability, …

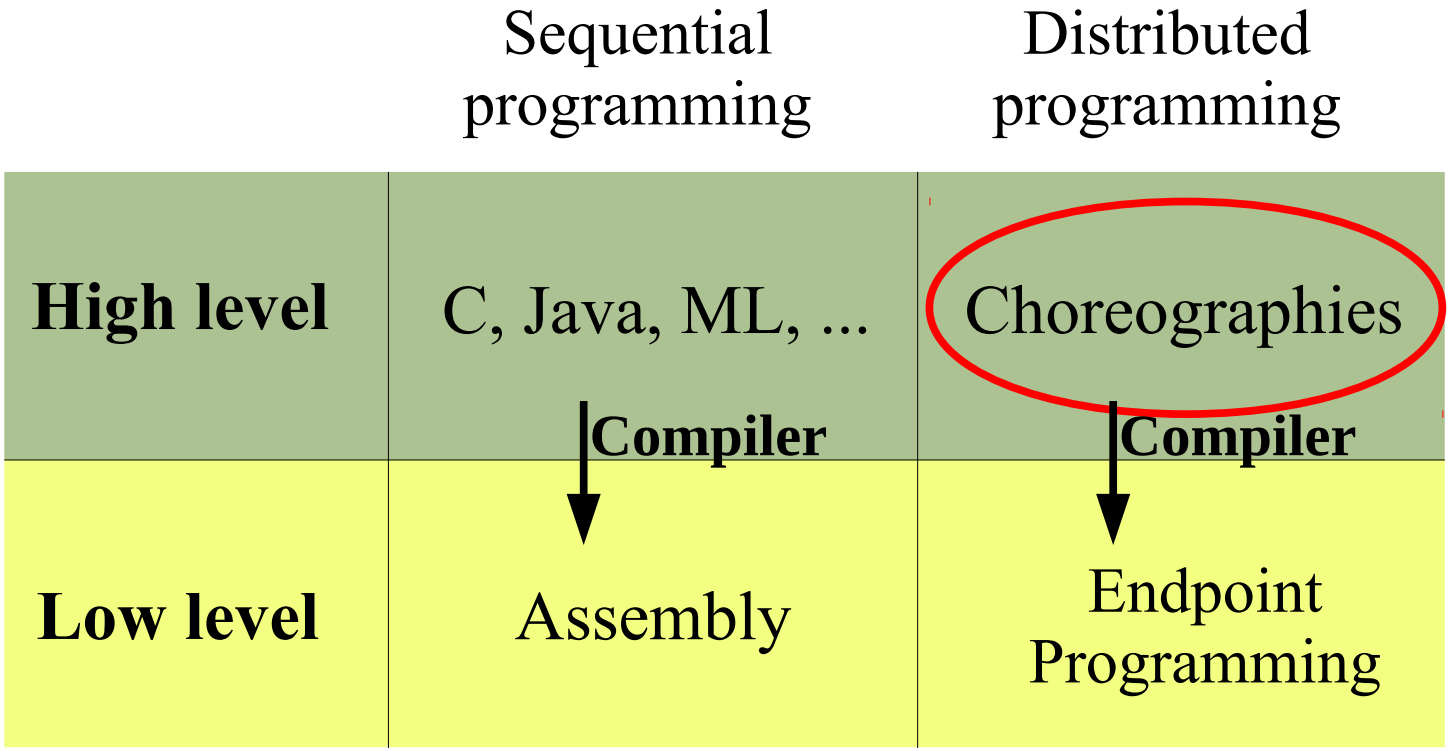
The momentum of choreographies

- Dawn of a new paradigm?



The momentum of choreographies

- Dawn of a new paradigm?



Towards a new paradigm

- Lots of promising formal models.

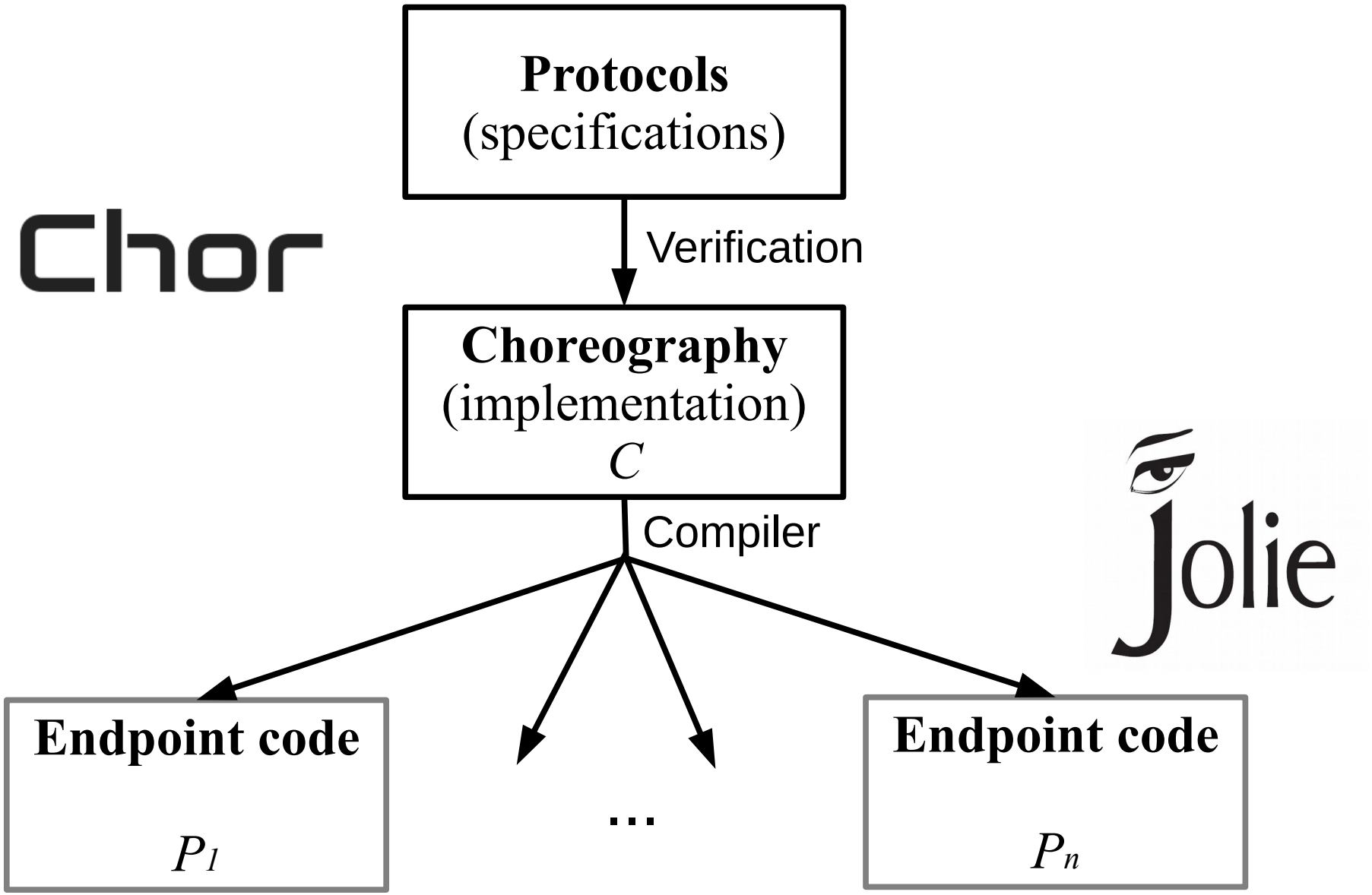
- We need tools to evaluate the paradigm.

Methodology

In this work

- We present the Chor language: Chor
- A programming language based on choreographies.
- Eclipse-based IDE with on-the-fly deadlock verification.
- A compiler for generating executable Jolie code.
- An initial evaluation of the choreographic programming paradigm.

Development methodology



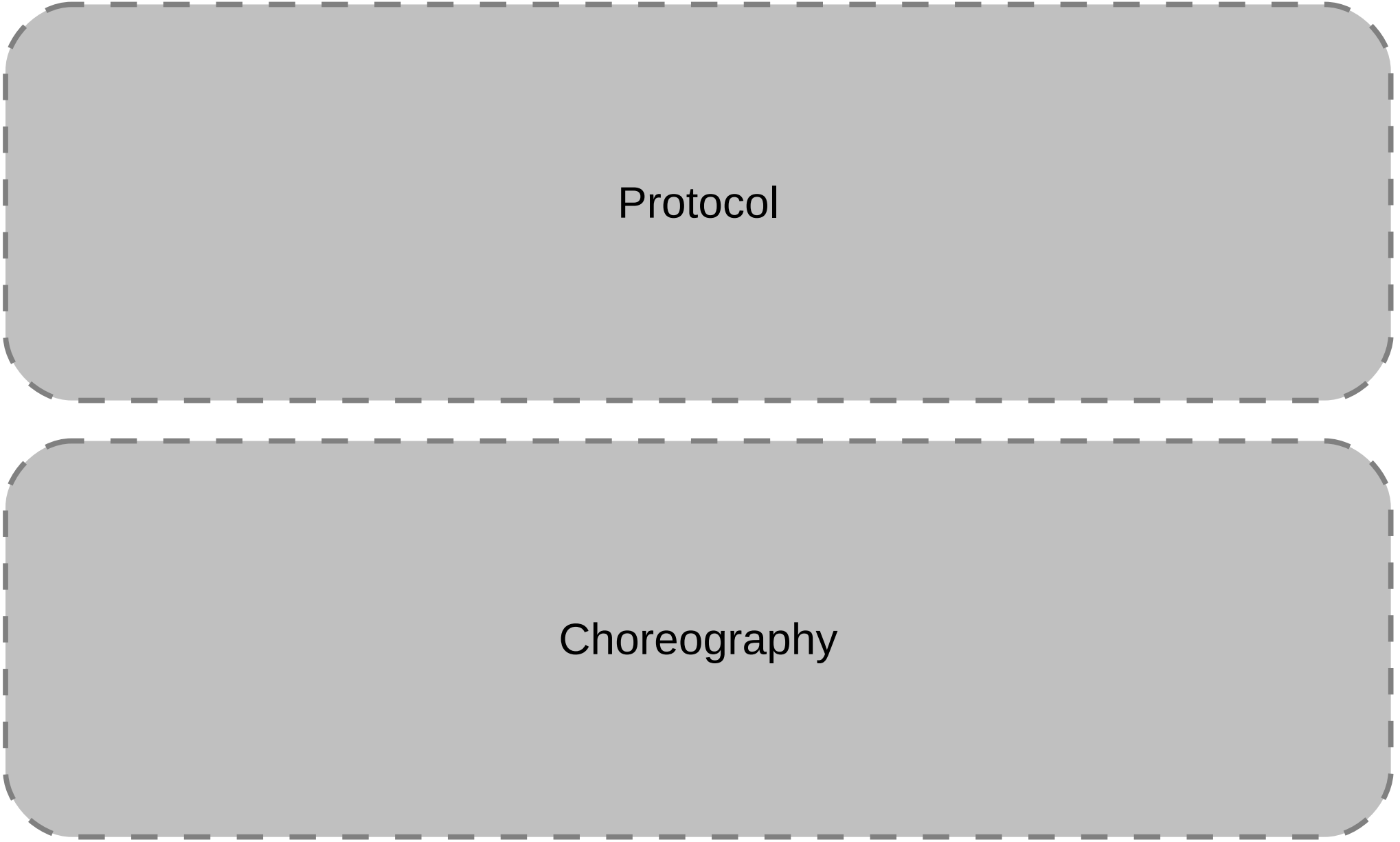
Main design ideas

- We describe the behaviour of **processes** in a system.
- Each process has a local state.
- Processes take part to multiparty conversations, tracked as **sessions**.
- Sessions are **started** through **public channels** (e.g., URLs).
- Both sessions and processes can be dynamically created.

An example

- Alice and Bob buy together a book on Amazon.

An example



Buyer-Helper-Seller protocol

- A protocol for buying a book.

```
Buyer   →  Seller   :  string;      // Ask the price
Seller  →  Buyer    :  int;         // Get the price
Buyer   →  Helper   :  int;         // Contribution
Helper  →  Seller   :  {            // Choice
                       ok:  Seller  → Helper: string;
                            end,
                       ko:  end
                       }
```

Buyer-Helper-Seller protocol

- A protocol for buying a book.

```
Buyer   → Seller  : string;      // Ask the price
Seller  → Buyer   : int;         // Get the price
Buyer   → Helper  : int;         // Contribution
Helper  → Seller  : {            // Choice
                    ok:  Seller  → Helper: string;
                         end,
                    ko:  end
                    }
```

Buyer-Helper-Seller protocol

- A protocol for buying a book.

```
Buyer   → Seller  : string;      // Ask the price
Seller  → Buyer   : int;         // Get the price
Buyer   → Helper  : int;         // Contribution
Helper  → Seller  : {            // Choice
                    ok:  Seller  → Helper: string;
                         end,
                    ko:  end
                    }
```

Buyer-Helper-Seller protocol

- A protocol for buying a book.

```
Buyer   → Seller  : string;     // Ask the price
Seller  → Buyer   : int;        // Get the price
Buyer   → Helper  : int;        // Contribution
Helper  → Seller  : {           // Choice
                    ok:  Seller → Helper: string;
                         end,
                    ko:  end
                    }
```

Buyer-Helper-Seller protocol

- A protocol for buying a book.

```
Buyer   →  Seller   :  string;      // Ask the price
Seller  →  Buyer    :  int;         // Get the price
Buyer   →  Helper   :  int;         // Contribution
Helper  →  Seller   :  {            // Choice
                       ok:  Seller  → Helper: string;
                            end,
                       ko:  end
                       }
```

Buyer-Helper-Seller protocol

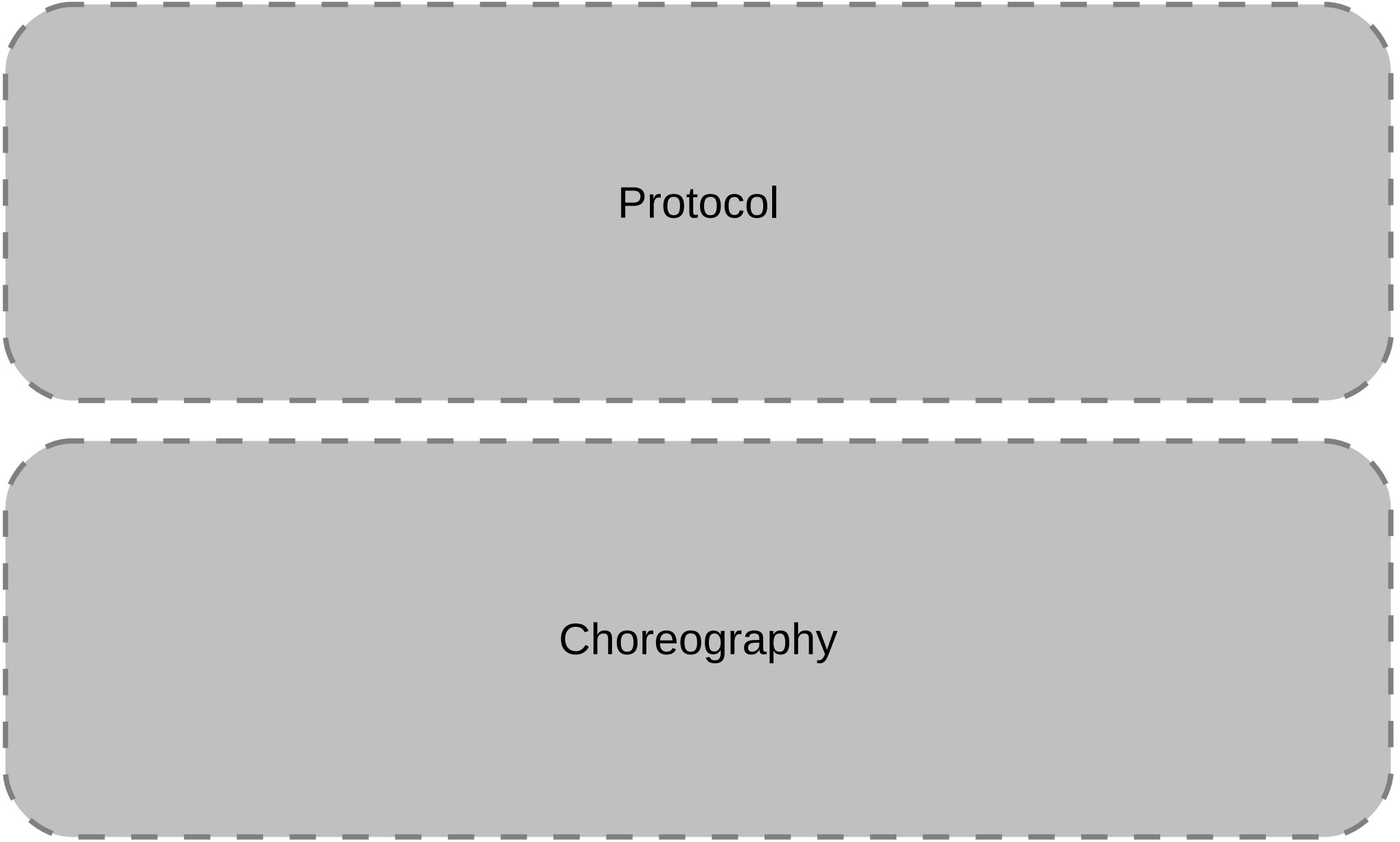
- A protocol for buying a book.

```
Buyer   →  Seller   :  string;      // Ask the price
Seller  →  Buyer    :  int;         // Get the price
Buyer   →  Helper   :  int;         // Contribution
Helper  →  Seller   :  {            // Choice
                       ok:   Seller  → Helper: string;
                             end,
                       ko:   end
                       }
```

Buyer-Helper-Seller protocol

- A protocol for buying a book.

```
Buyer   →  Seller   :  string;      // Ask the price
Seller  →  Buyer    :  int;         // Get the price
Buyer   →  Helper   :  int;         // Contribution
Helper  →  Seller   :  {            // Choice
                       ok:   Seller  → Helper: string;
                             end,
                       ko:   end
                       }
```

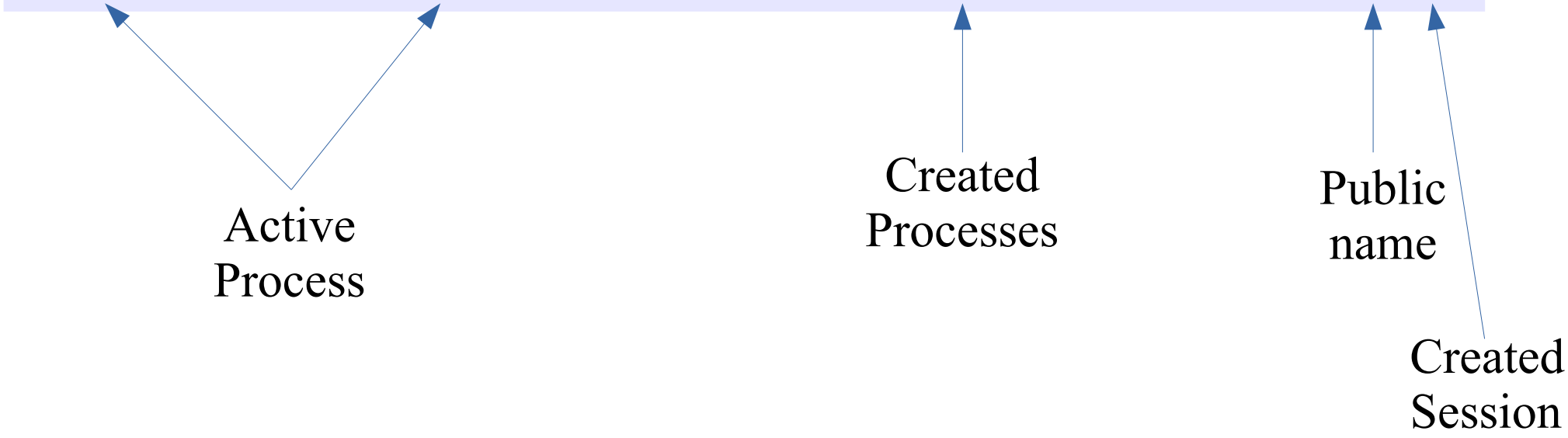

Protocol
Choreography

```
Buyer   → Seller  : string;     // Ask the price
Seller  → Buyer   : int;        // Get the price
Buyer   → Helper  : int;        // Contribution
Helper  → Seller  : {           // Choice
                    ok:  Seller → Helper: string;
                         end,
                    ko:  end
                    }
```

```
Buyer   →  Seller   :  string;      // Ask the price
Seller  →  Buyer    :  int;         // Get the price
Buyer   →  Helper   :  int;         // Contribution
Helper  →  Seller   :  {            // Choice
                       ok:   Seller  → Helper: string;
                             end,
                       ko:   end
                       }
```

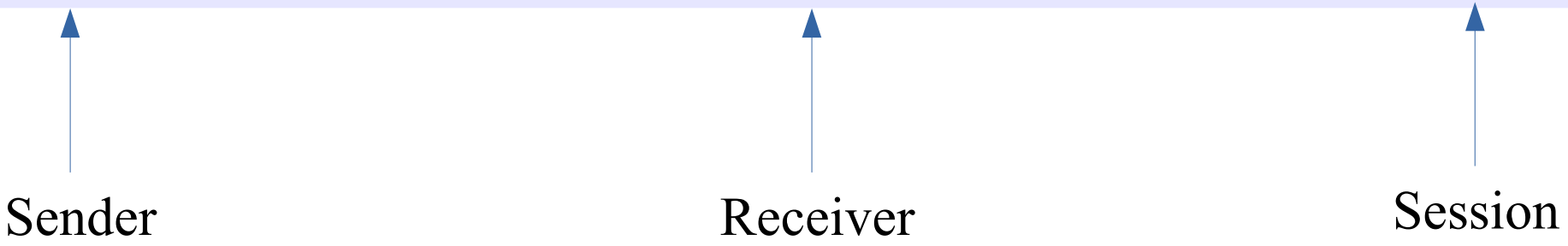
- Let a be a public URL (e.g., www.amazon.com) with the protocol above.

```
alice[Buyer], bob[Helper] start amazon[Seller] :  a(k)
```



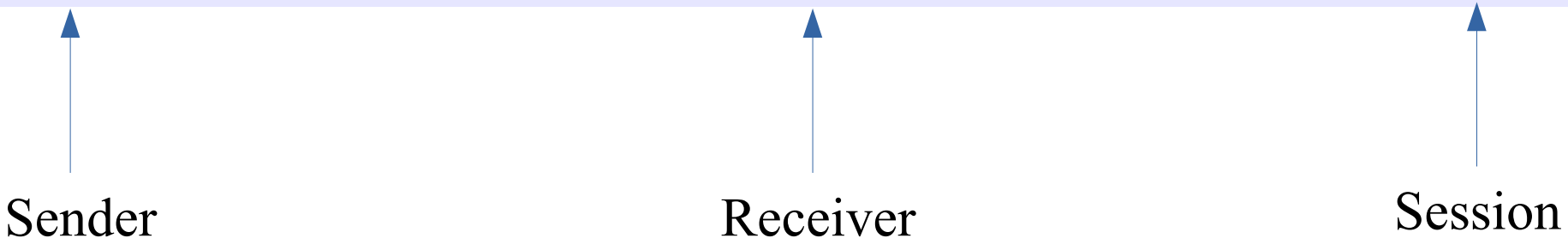
```
Buyer   →  Seller   :  string;      // Ask the price
Seller  →  Buyer    :  int;         // Get the price
Buyer   →  Helper   :  int;         // Contribution
Helper  →  Seller   :  {            // Choice
                       ok:   Seller  → Helper: string;
                             end,
                       ko:   end
                       }
```

```
alice[Buyer], bob[Helper] start amazon[Seller] :  a(k)  ;
alice."rabbits"          →  amazon.book          :  k
```



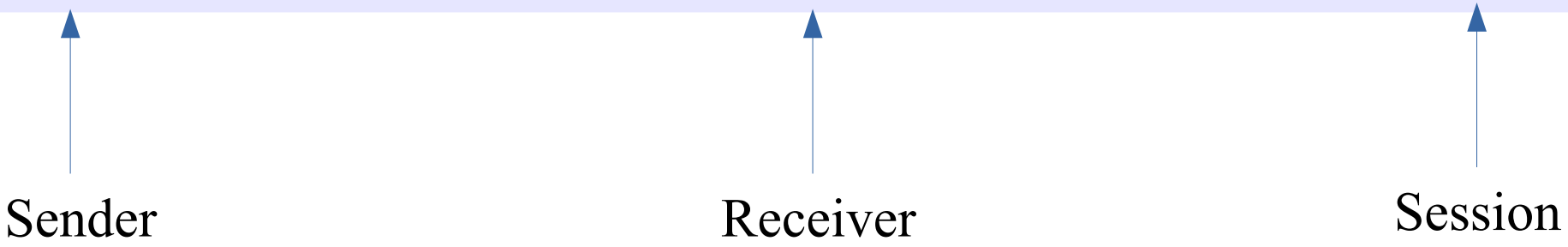
```
Buyer   → Seller  : string;     // Ask the price
Seller  → Buyer   : int;        // Get the price
Buyer   → Helper  : int;        // Contribution
Helper  → Seller  : {           // Choice
                    ok:  Seller → Helper: string;
                         end,
                    ko:  end
                    }
```

```
alice[Buyer], bob[Helper] start amazon[Seller] :  a(k) ;
alice."rabbits"          →  amazon.book        :  k    ;
amazon.price(book)       →  alice.price        :  k
```



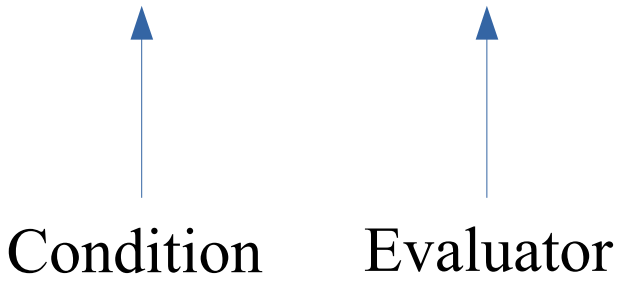
```
Buyer   →  Seller   :  string;      // Ask the price
Seller  →  Buyer    :  int;         // Get the price
Buyer   →  Helper   :  int;         // Contribution
Helper  →  Seller   :  {            // Choice
                       ok:  Seller  → Helper: string;
                            end,
                       ko:  end
                       }
```

```
alice[Buyer], bob[Helper] start amazon[Seller] :  a(k)  ;
alice."rabbits"          →  amazon.book          :  k     ;
amazon.price(book)       →  alice.price          :  k     ;
alice.(price/2)          →  bob.contrib          :  k
```



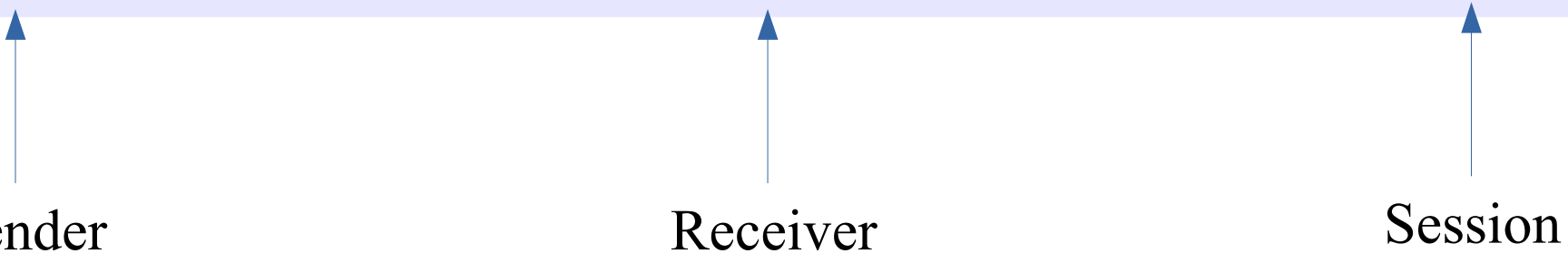
```
Buyer   → Seller  :  string;      // Ask the price
Seller  → Buyer   :  int;         // Get the price
Buyer   → Helper  :  int;         // Contribution
Helper  → Seller  :  {            // Choice
                     ok:  Seller  → Helper: string;
                          end,
                     ko:  end
                     }
```

```
alice[Buyer], bob[Helper] start amazon[Seller] :  a(k)  ;
alice.”rabbits”          →  amazon.book          :  k     ;
amazon.price(book)       →  alice.price          :  k     ;
alice.(price/2)          →  bob.contrib          :  k     ;
if (contrib < 100$)@bob
```



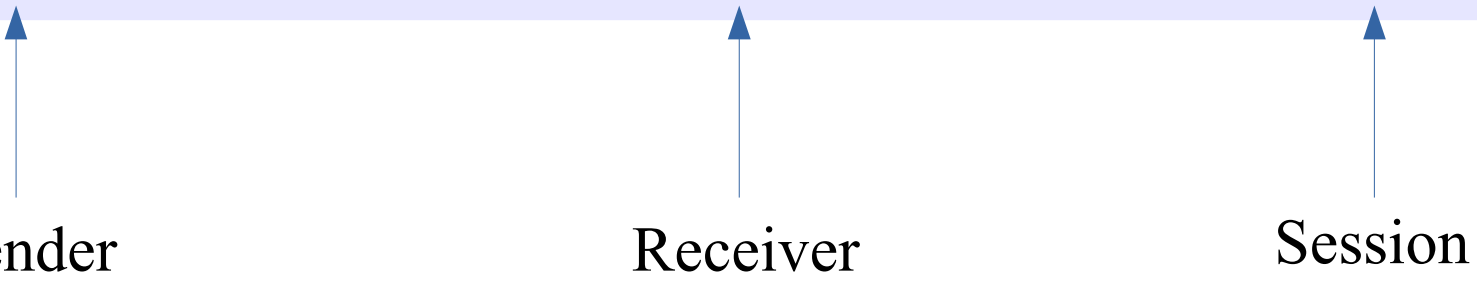
```
Buyer   →  Seller   :  string;      // Ask the price
Seller  →  Buyer    :  int;         // Get the price
Buyer   →  Helper   :  int;         // Contribution
Helper  →  Seller   :  {            // Choice
                       ok:  Seller  → Helper: string;
                            end,
                       ko:  end
                       }
```

```
alice[Buyer], bob[Helper] start amazon[Seller] :  a(k)  ;
alice.”rabbits”          →  amazon.book          :  k     ;
amazon.price(book)       →  alice.price          :  k     ;
alice.(price/2)          →  bob.contrib          :  k     ;
if (contrib < 100$)@bob
   bob                   →  amazon               :  k[ok]
```



```
Buyer   →  Seller   :  string;      // Ask the price
Seller  →  Buyer    :  int;         // Get the price
Buyer   →  Helper   :  int;         // Contribution
Helper  →  Seller   :  {            // Choice
                       ok:  Seller  → Helper: string;
                            end,
                       ko:  end
                       }
```

```
alice[Buyer], bob[Helper] start amazon[Seller] :  a(k)  ;
alice."rabbits"          →  amazon.book      :  k          ;
amazon.price(book)       →  alice.price      :  k          ;
alice.(price/2)          →  bob.contrib      :  k          ;
if (contrib < 100$)@bob
   bob                   →  amazon           :  k[ok]      ;
   amazon.text(book)     →  bob.text         :  k
```



```
Buyer   → Seller  :  string;     // Ask the price
Seller  → Buyer   :  int;        // Get the price
Buyer   → Helper  :  int;        // Contribution
Helper  → Seller  :  {           // Choice
                     ok:  Seller → Helper: string;
                          end,
                     ko:  end
                     }
```

```
alice[Buyer], bob[Helper] start amazon[Seller] :  a(k)  ;
alice.”rabbits”          →  amazon.book      :  k          ;
amazon.price(book)       →  alice.price      :  k          ;
alice.(price/2)          →  bob.contrib      :  k          ;
if (contrib < 100$)@bob
   bob                   →  amazon           :  k[ok]      ;
   amazon.text(book)     →  bob.text         :  k
else
```

```
Buyer   → Seller  : string;     // Ask the price
Seller  → Buyer   : int;        // Get the price
Buyer   → Helper  : int;        // Contribution
Helper  → Seller  : {           // Choice
                    ok:  Seller → Helper: string;
                         end,
                    ko:  end
                    }
```

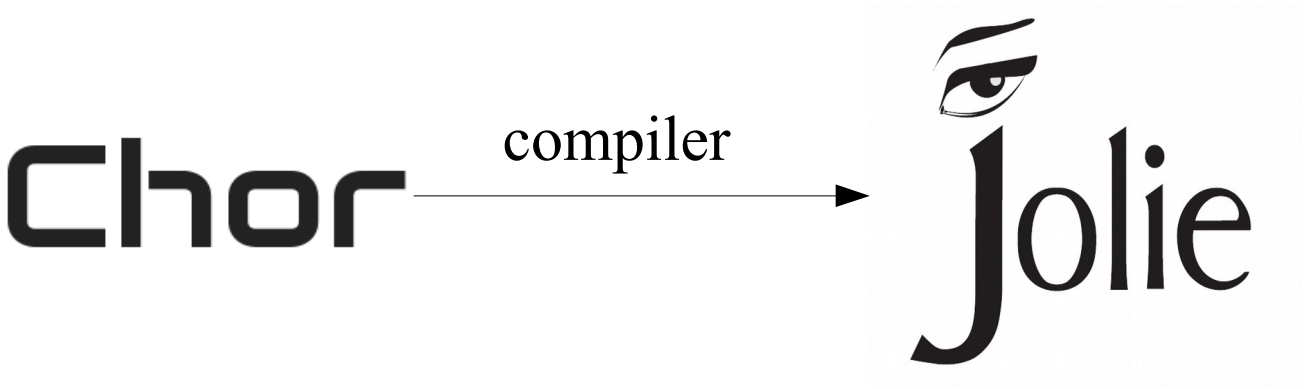
```
alice[Buyer], bob[Helper] start amazon[Seller] :  a(k)  ;
alice."rabbits"          → amazon.book      :  k          ;
amazon.price(book)       → alice.price      :  k          ;
alice.(price/2)          → bob.contrib      :  k          ;
if (contrib < 100$)@bob
   bob                   → amazon           :  k[ok]      ;
   amazon.text(book)     → bob.text         :  k
else
   bob                   → amazon           :  k[ko]
```

Sender

Receiver

Session

Compiler



- We provide a compiler to executable code in Jolie.
- The code is guaranteed to be deadlock-free by construction.
- Jolie allows us to reuse executables in different deployments (HTTP, etc.).
- Demo.

Evaluation

- We used Chor for evaluating our approach with representative use cases:
  - authentication protocols (OpenID);
  - E-Commerce;
  - data streaming;
  - service discovery.

- Industrial collaborators:

Conclusions

Conclusions

- A new language for a new paradigm.
- Guarantees deadlock-freedom: suitable for critical systems.
- Fast prototyping of systems.
- A lot of future work to do!

Limitations

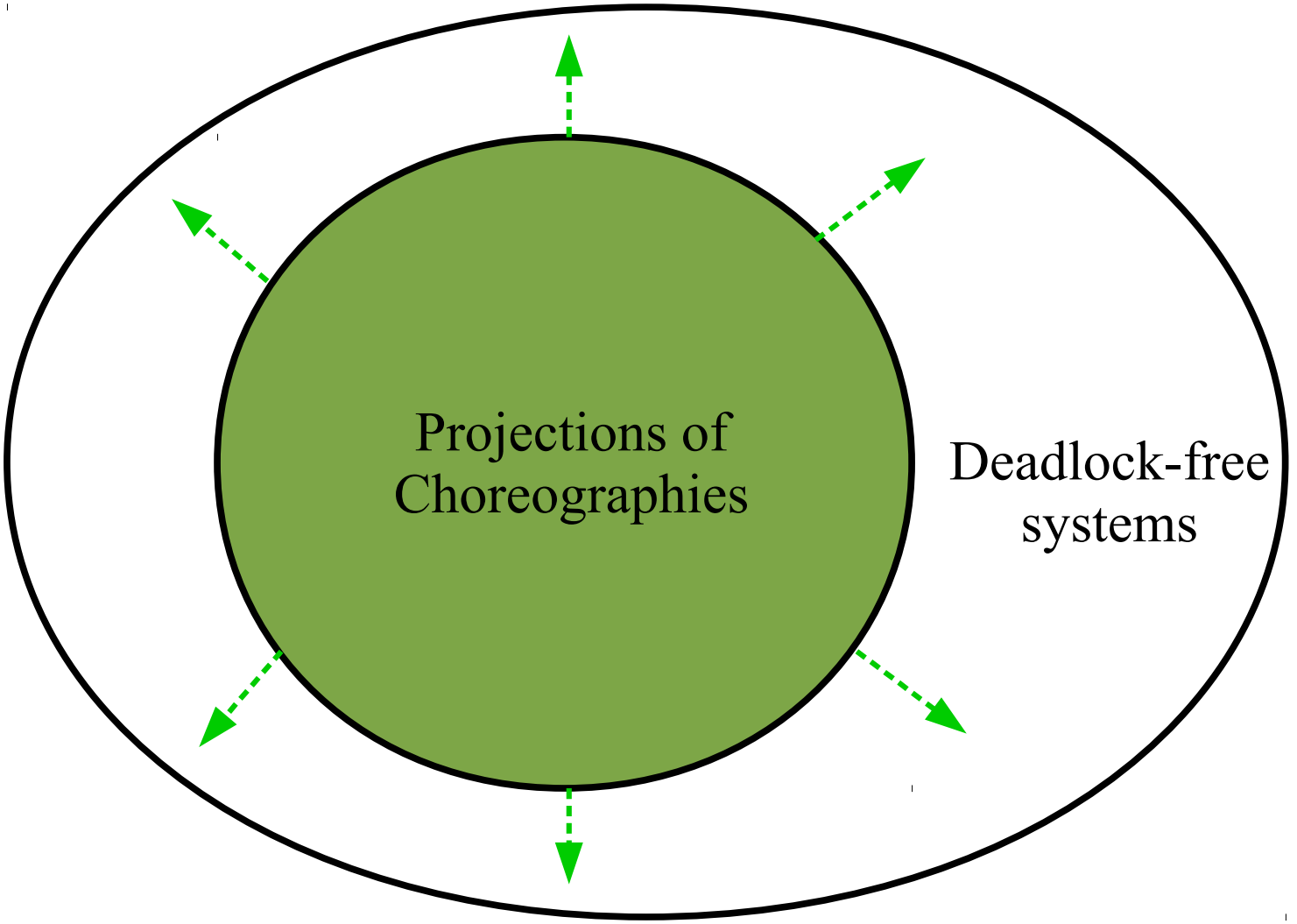
- No support for external services yet (e.g., cannot invoke Google search).

- No support for round-trip programming.

Future Directions

Future directions

- How far can we go? Need for more systematic studies.



Future directions

- Exploiting the global view of choreographies in:
  - Fault handling.
  - Reversible computing.
  - Security.
  - Model checking.

`Fabrizio → audience :` Thank you!

- More information at:
  - Chor Website: `http://www.chor-lang.org/`
  - Jolie Website: `http://www.jolie-lang.org/`
  - My web page: `http://www.fabriziomontesi.com/`

Questions?

Chor

Jolie