ChIP: a Choreographic Integration Process

Saverio Giallorenzo[1], Ivan Lanese[2,3], and Daniel Russo[3]

[1] University of Southern Denmark, Denmark
[2] Focus Team, INRIA
[3] University of Bologna, Italy

Context: System Integration

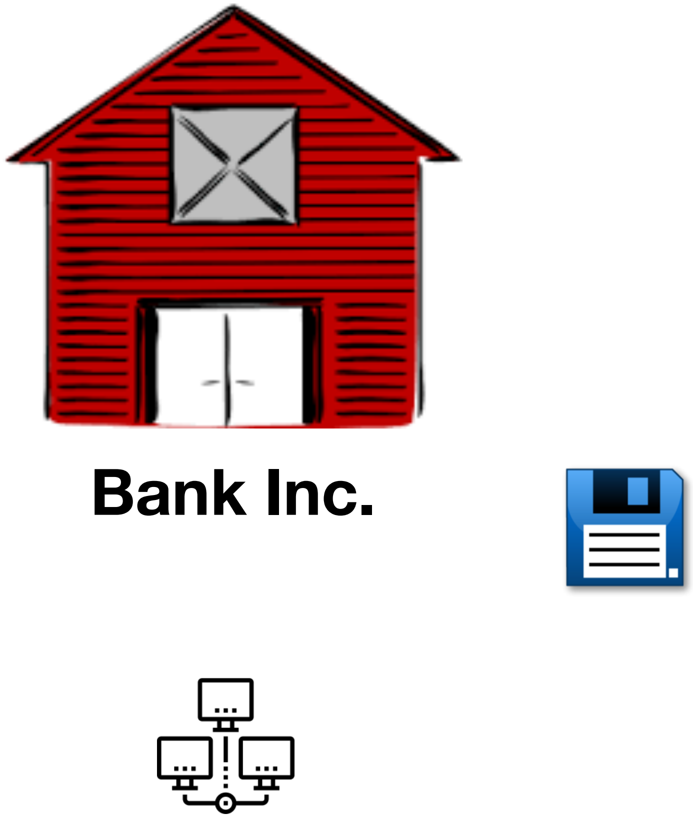
Context: System Integration

Bank Inc.

Context: System Integration

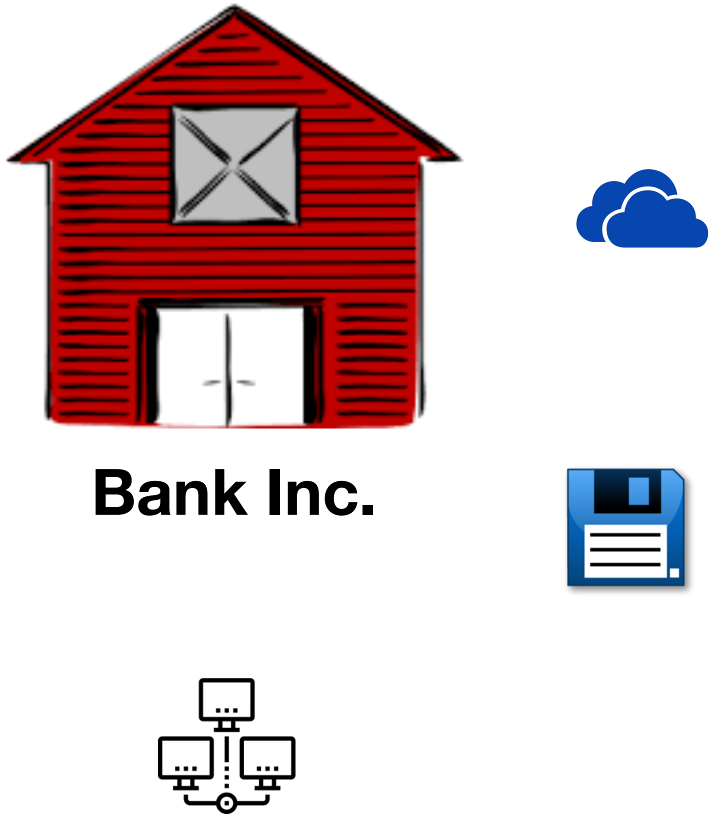
Bank Inc.

Context: System Integration

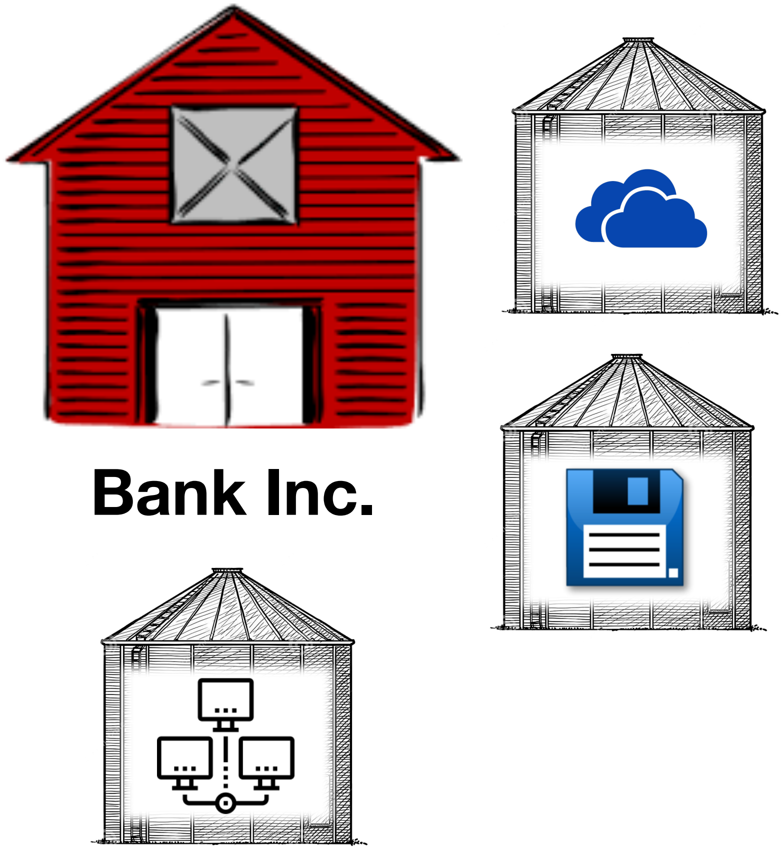


Bank Inc.

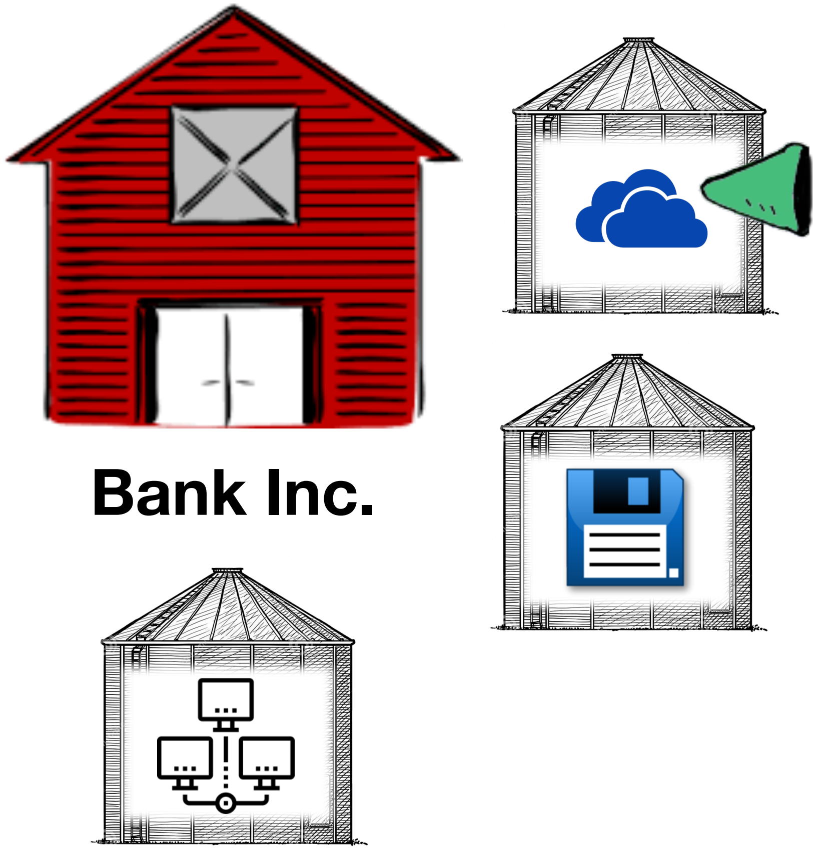
Context: System Integration



Context: System Integration

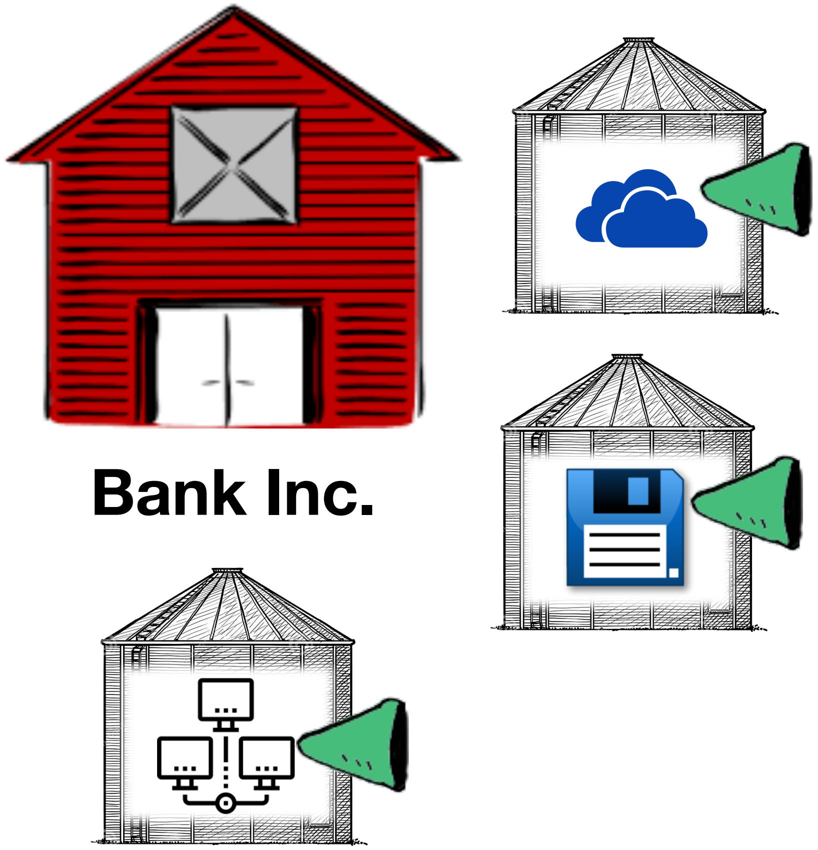


Context: System Integration



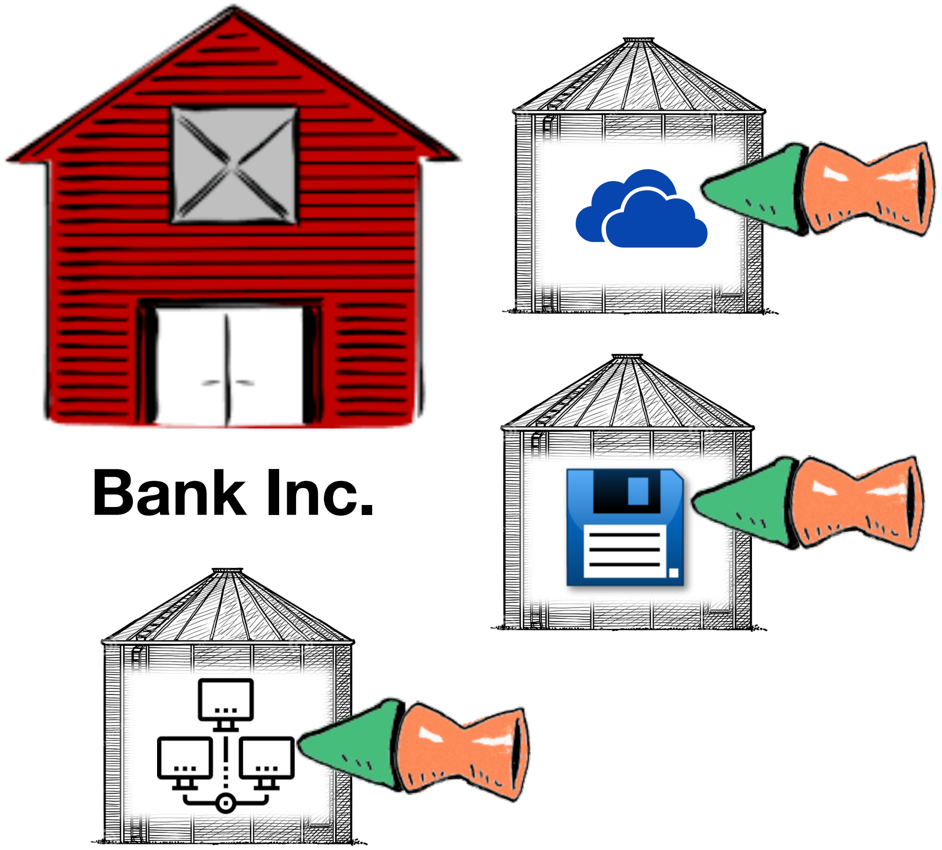
API

Context: System Integration



API

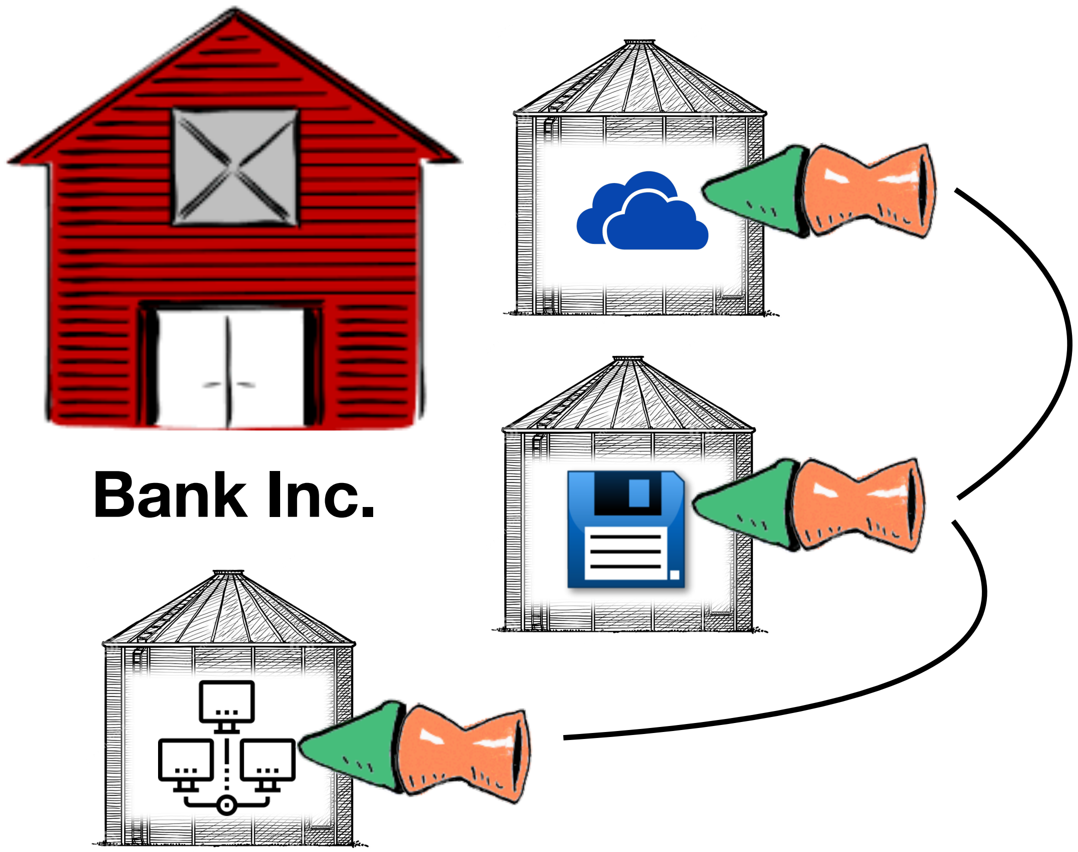
Context: System Integration



Connector

API

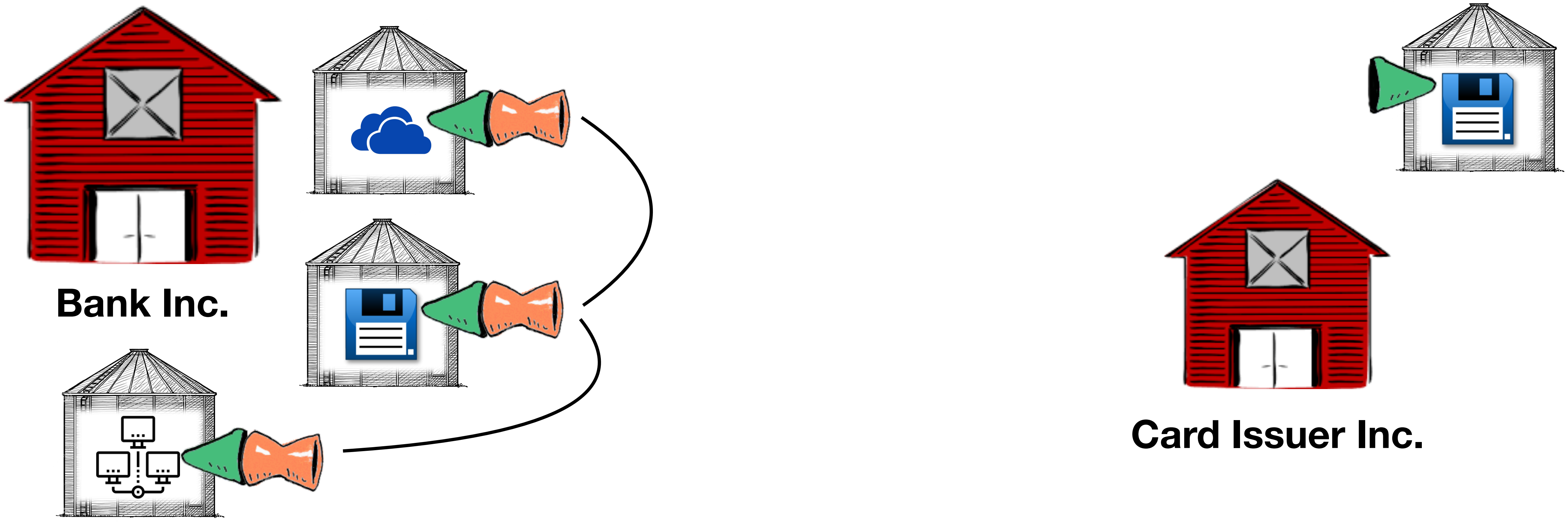
Context: System Integration



Connector

API

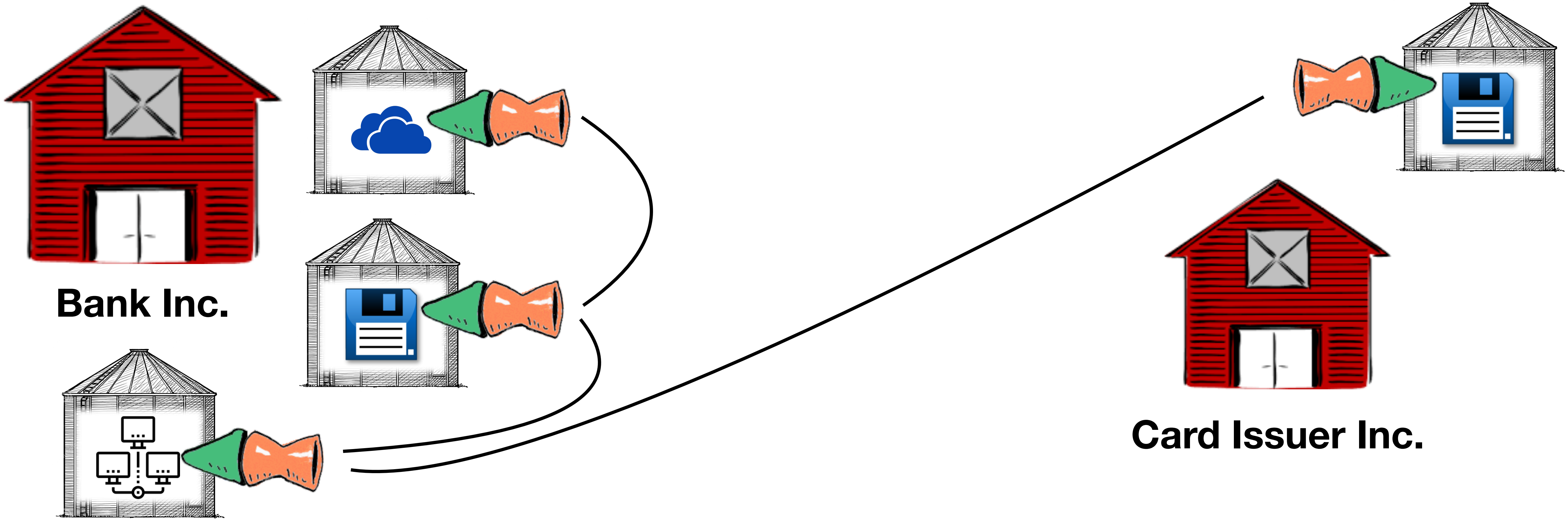
Context: System Integration



Connector

API

Context: System Integration



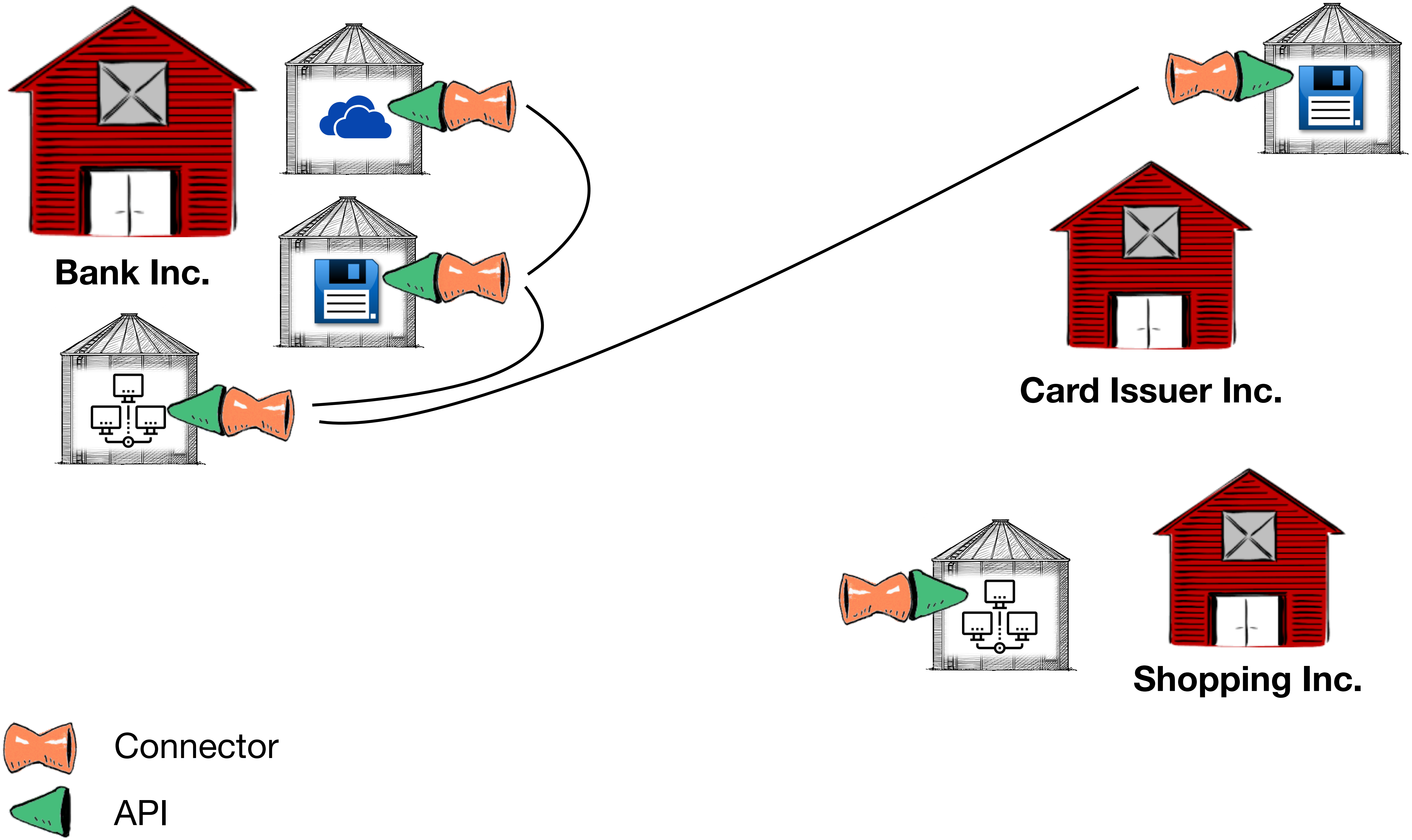
Bank Inc.

Card Issuer Inc.

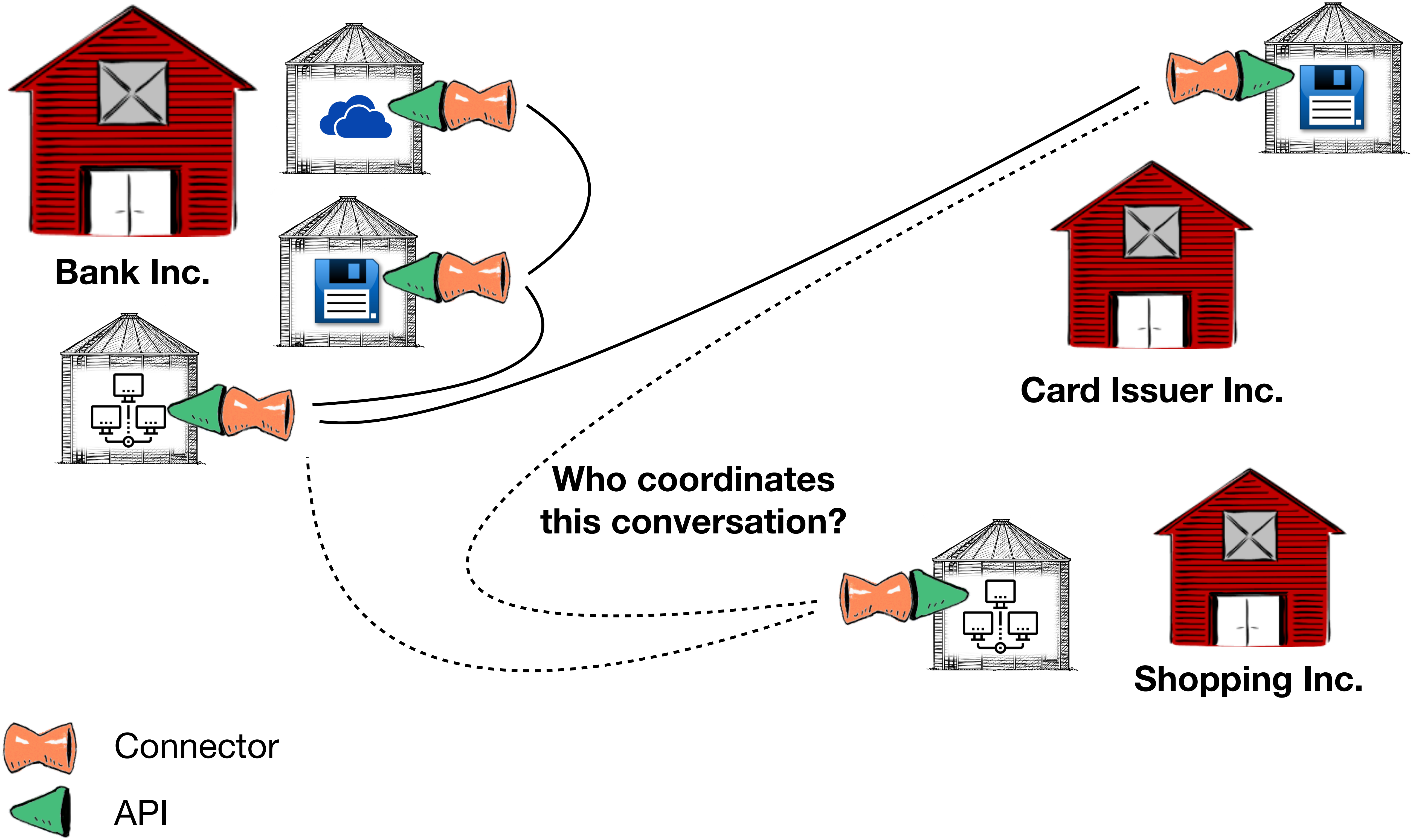
Connector

API

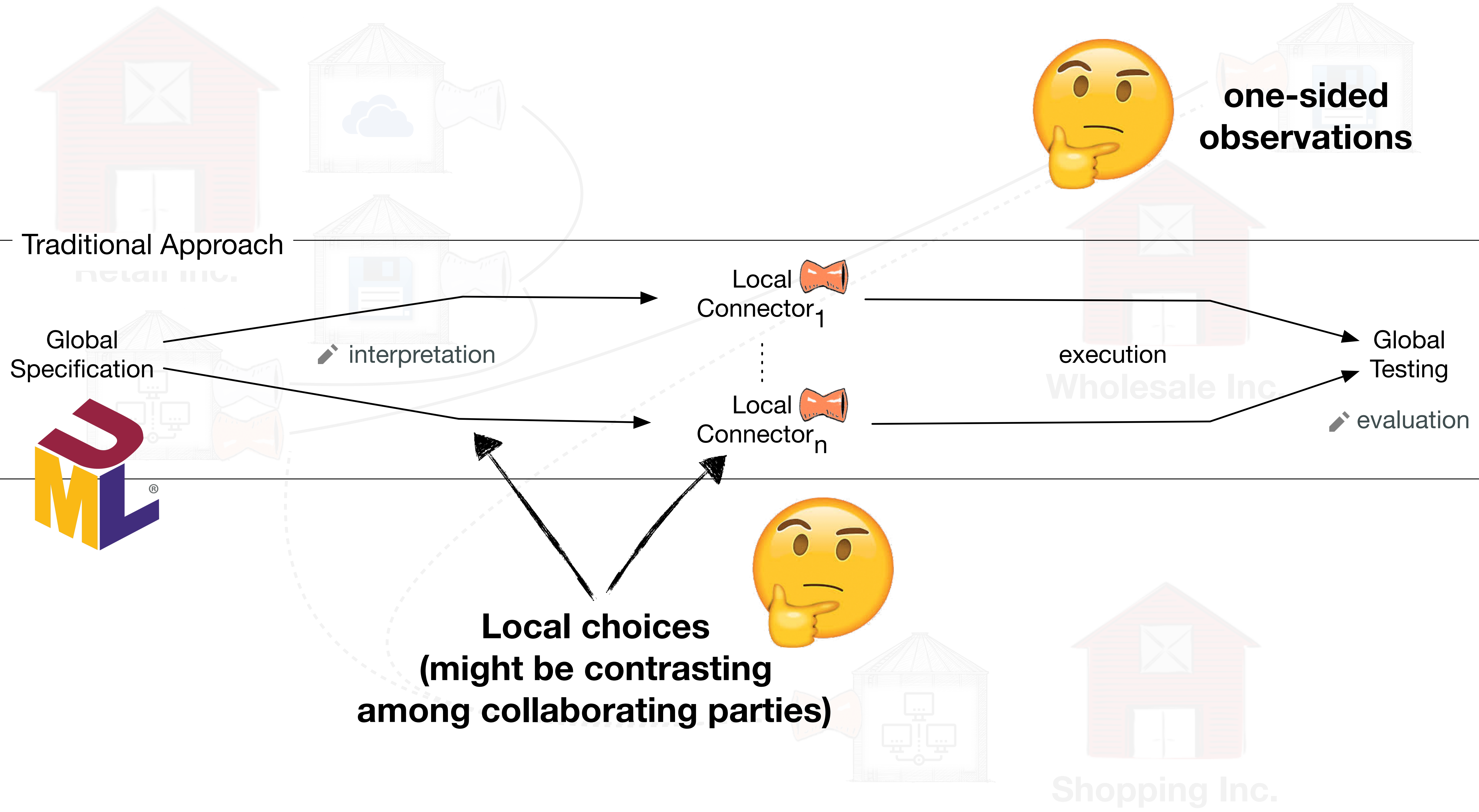
Context: System Integration



Context: System Integration



Context: System Integration



one-sided observations

Local choices (might be contrasting among collaborating parties)

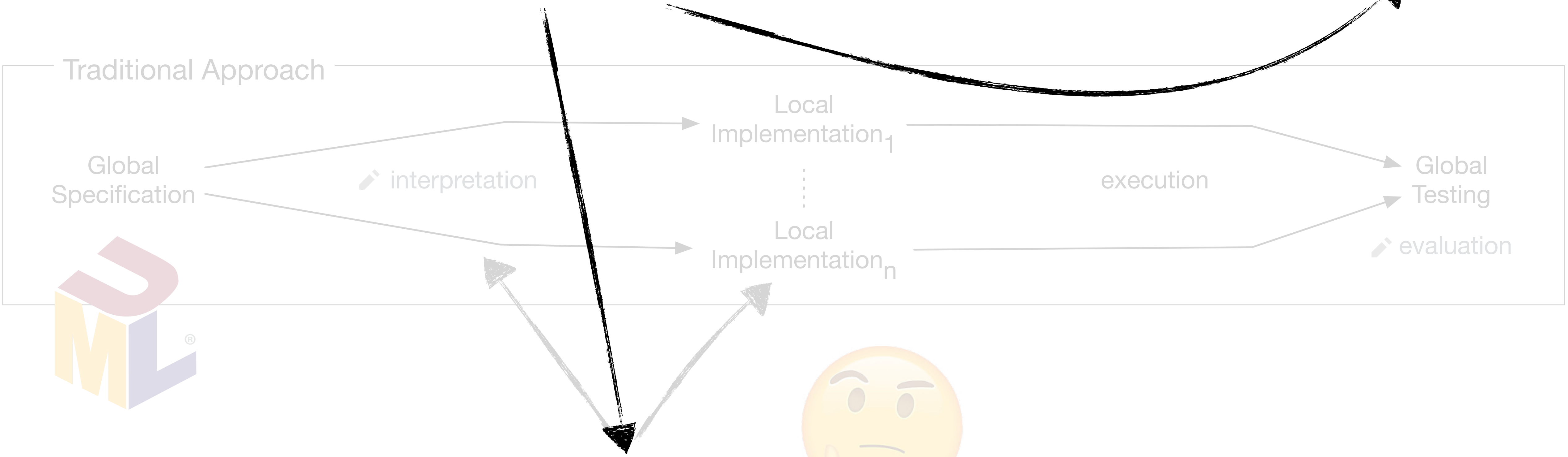
Research Question

Can we fill these information gaps

AND

preserve separation of concerns ?

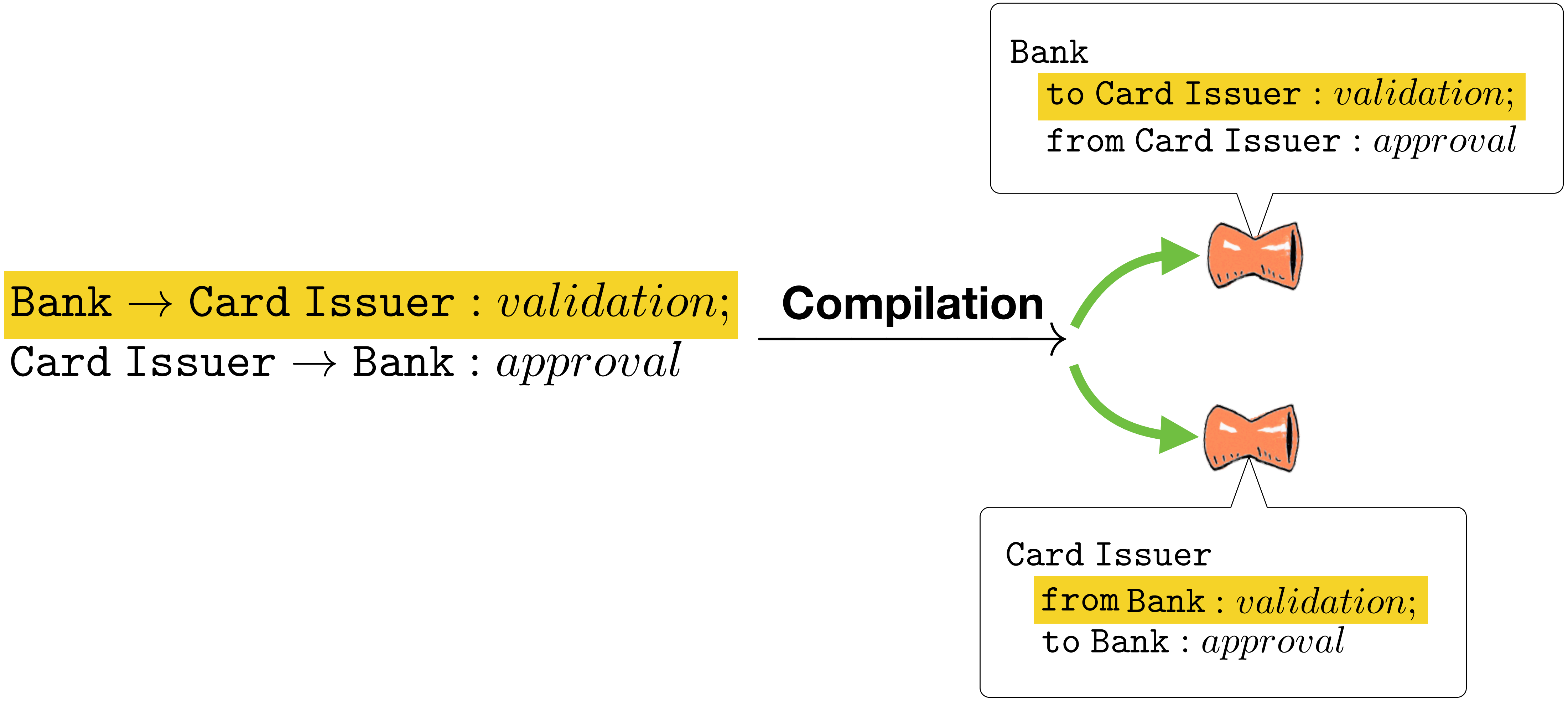
one-sided observations



Local choices (might be contrasting among collaborating parties)

Programming

ChIP: a Choreographic Integration Process



Bank → Card Issuer : *validation*;
Card Issuer → Bank : *approval*

Compilation

```
Bank
  to Card Issuer : validation;
  from Card Issuer : approval
```

```
Card Issuer
  from Bank : validation;
  to Bank : approval
```

Programming

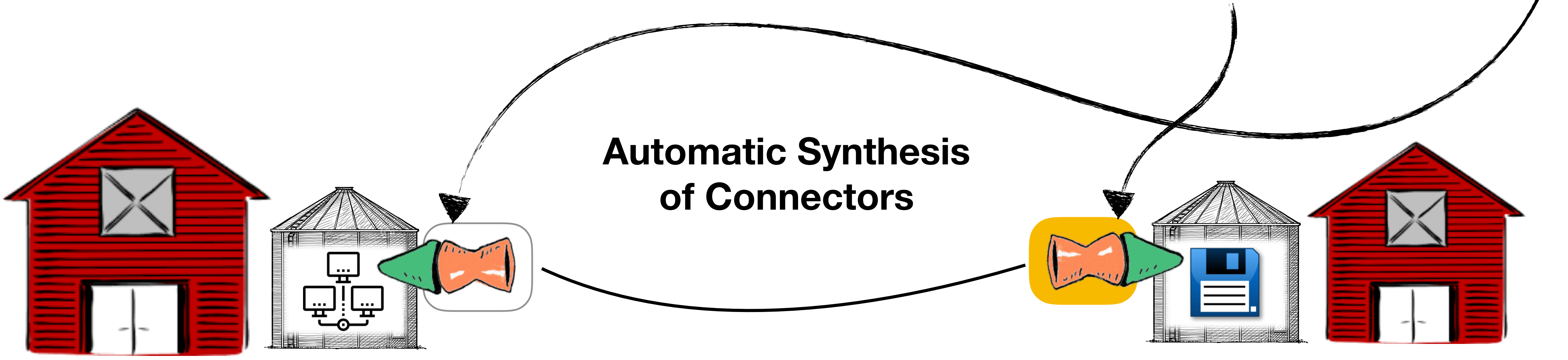
ChIP: a Choreographic Integration Process

Bank → Card Issuer : *validation*;
Card Issuer → Bank : *approval*

Compilation →

Bank
  to Card Issuer : *validation*;
  from Card Issuer : *approval*

Card Issuer
  from Bank : *validation*;
  to Bank : *approval*



Automatic Synthesis of Connectors

Bank Inc.

Card Issuer Inc.

Programming

ChIP: a Choreographic Integration Process

include **verifyRequest** from …

$\mathtt{Bank} \longrightarrow \mathtt{Card\ Issuer} : \mathit{validation};$

approval = **verifyRequest**@Card Issuer(validation);

$\mathtt{Card\ Issuer} \longrightarrow \mathtt{Bank} : \mathit{approval}$

A choreographic programming language is compatible with ChIP if it supports the integration of external functions (provided by the collaborating parties).

Programming

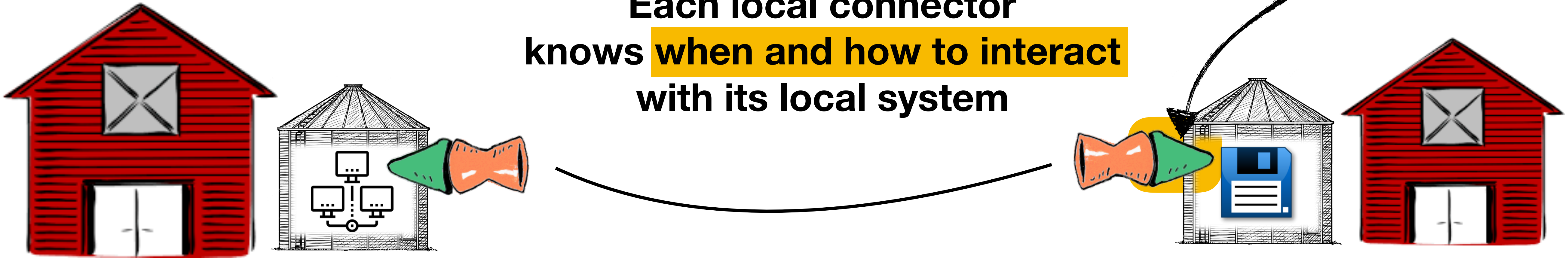
ChIP: a Choreographic Integration Process

include **verifyRequest** from …

$\texttt{Bank} \rightarrow \texttt{Card Issuer} : \textit{validation};$

approval = **verifyRequest**@Card Issuer(validation);

$\texttt{Card Issuer} \rightarrow \texttt{Bank} : \textit{approval}$



Each local connector
knows when and how to interact
with its local system

Bank Inc.

Card Issuer Inc.

Programming

ChIP: a Choreographic Integration Process

include **verifyRequest** from …

The Card Issuer will not provide this information to the other parties, since it is necessary only when it (locally) generates its local connector

$\texttt{Bank} \rightarrow \texttt{Card Issuer} : \mathit{validation};$

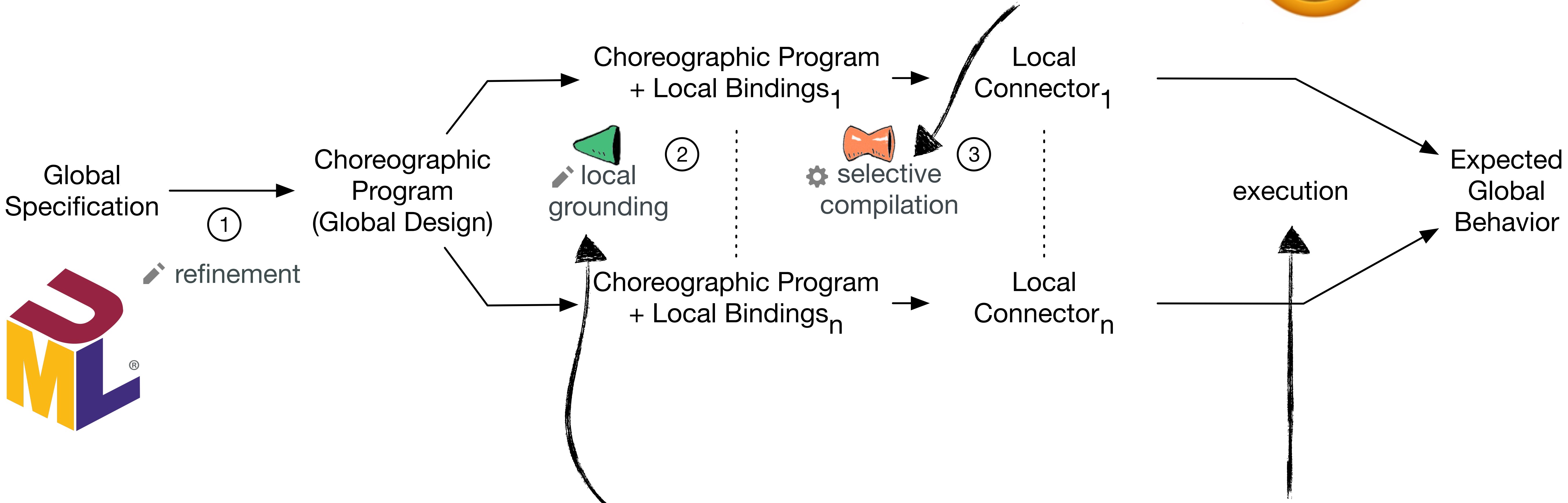
approval = **verifyRequest**@Card Issuer(validation);

$\texttt{Card Issuer} \rightarrow \texttt{Bank} : \mathit{approval}$

Programming

ChIP: a Choreographic Integration Process

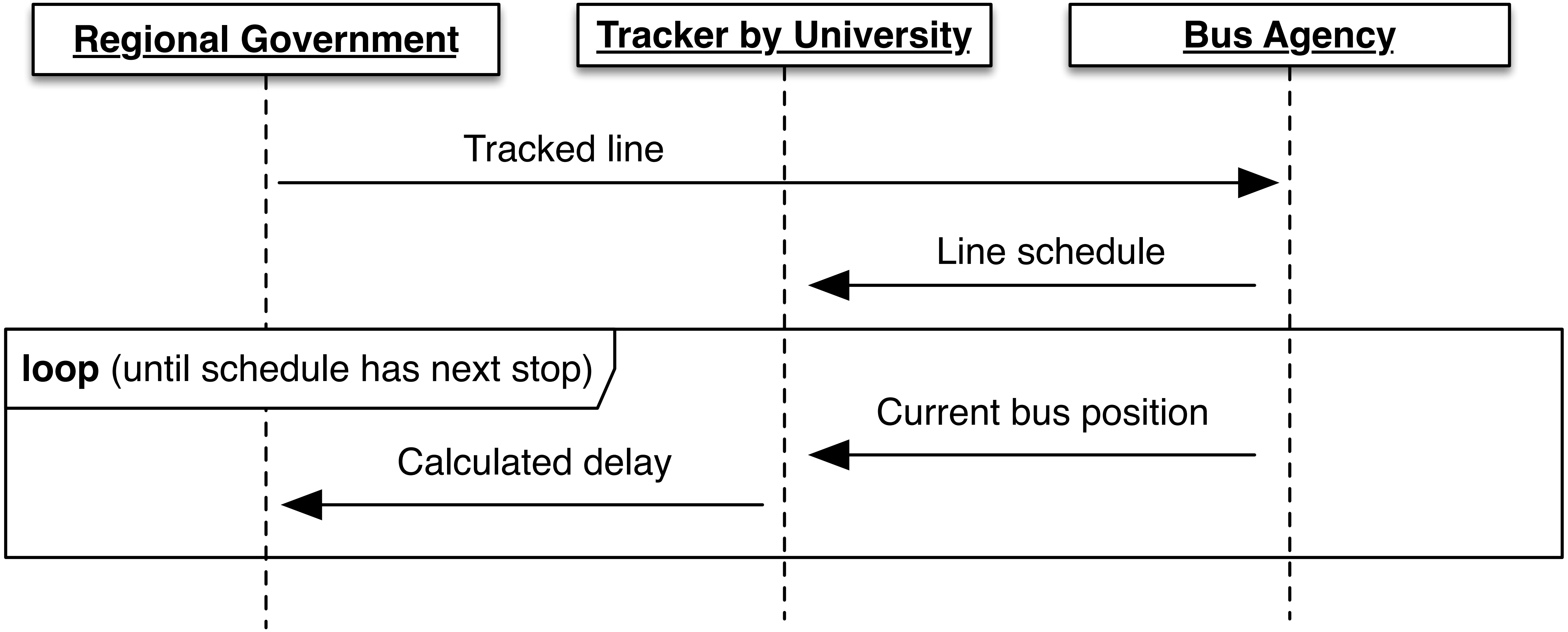
Each party compiles only its own connectors

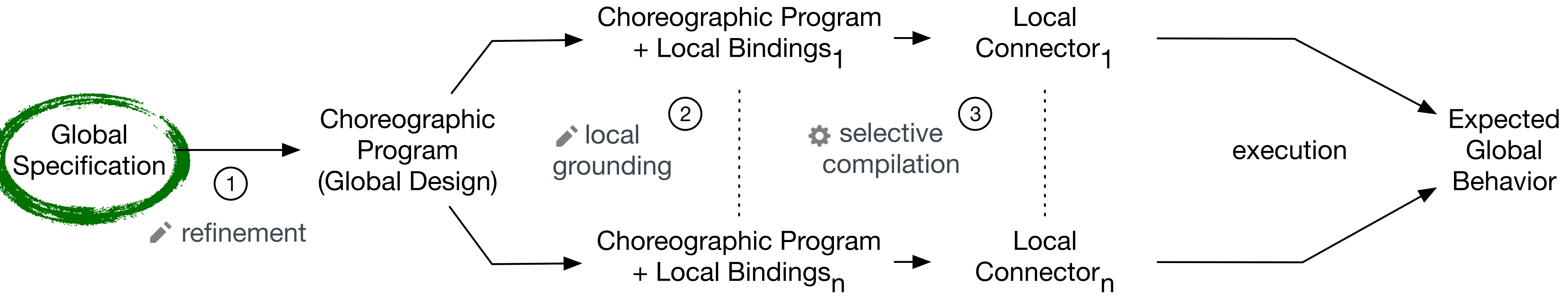


Local choices regard only the deployment information to let connectors interact with local systems

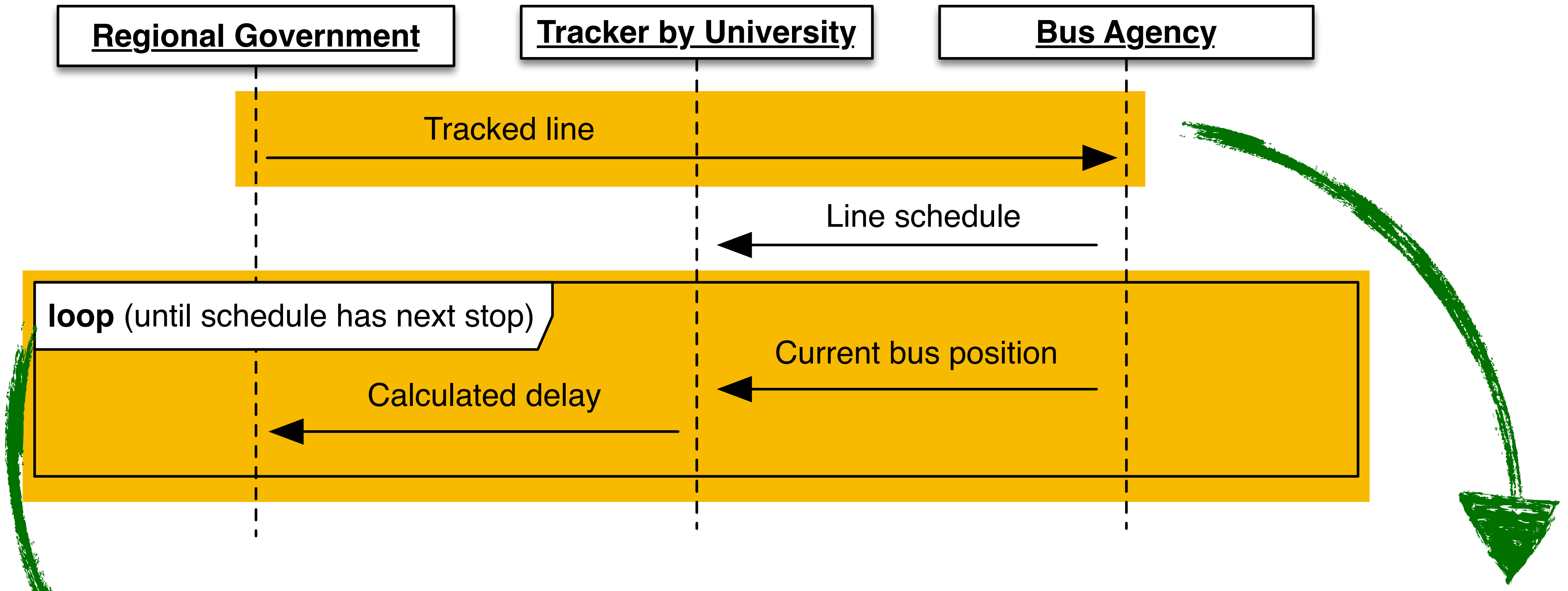
Connectors execute the exact behaviour defined in the choreography.

An Illustrative Example



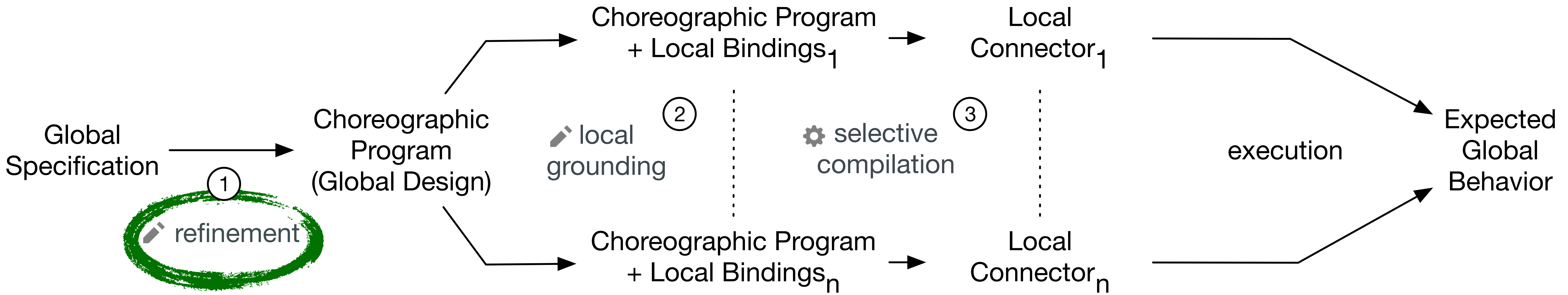


An Illustrative Example



AIOCJ

```
setLine: Government(line) -> BusAgency(line);
passSchedule: BusAgency(shd) -> Tracker(shd);
while(hasNext)@? {
  passPosition: BusAgency(pos) -> Tracker(pos);
  storeDelay: Tracker(delay) -> Government(delay)
}
```



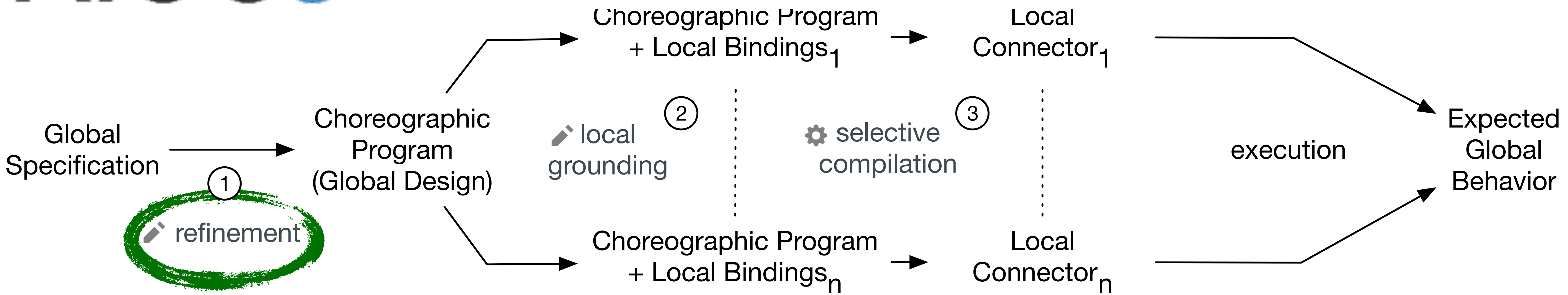
An Illustrative Example

```
locations {
  Admin: "reg-gov.org:80/BusCheckAdmin"
  Tracker: "university.edu:80/Tracker"
  BusAgency: "bus-agency.com:80/BusCheck" }
```

// code

```
shd@BusAgency = getSchedule(line);
passSchedule: BusAgency(shd) -> Tracker(shd);
hasNext@Tracker = hasNextStop(shd);
while(hasNext)@Tracker {
  pos@BusAgency = getPosition(line);
  passPosition: BusAgency(pos) -> Tracker(pos);
```

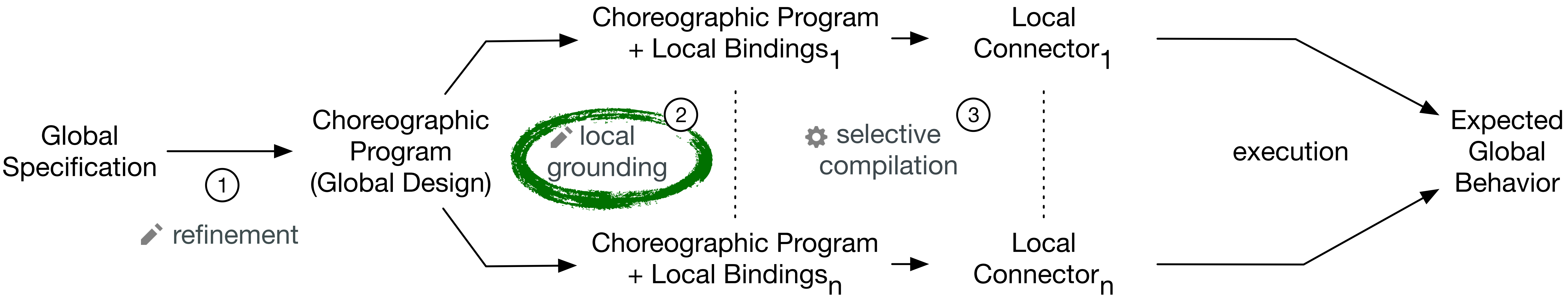
AIOCJ



An Illustrative Example

```
deployment {
 getSchedule from "socket://intranet.schdls:8000" with SOAP
 getPosition from "socket://intranet.GPS:8001" with HTTP }

 locations {
   Admin: "reg-gov.org:80/BusCheckAdmin"

 // code

 shd@BusAgency = getSchedule(line);
 passSchedule: BusAgency(shd) -> Tracker(shd);
 hasNext@Tracker = hasNextStop(shd);
 while(hasNext)@Tracker {
   pos@BusAgency = getPosition(line);
   passPosition: BusAgency(pos) -> Tracker(pos);
```

AIOCJ



Global Specification

① refinement

Choreographic Program (Global Design)

② local grounding

Choreographic Program + Local Bindings$_1$

Choreographic Program + Local Bindings$_n$

③ selective compilation

Local Connector$_1$

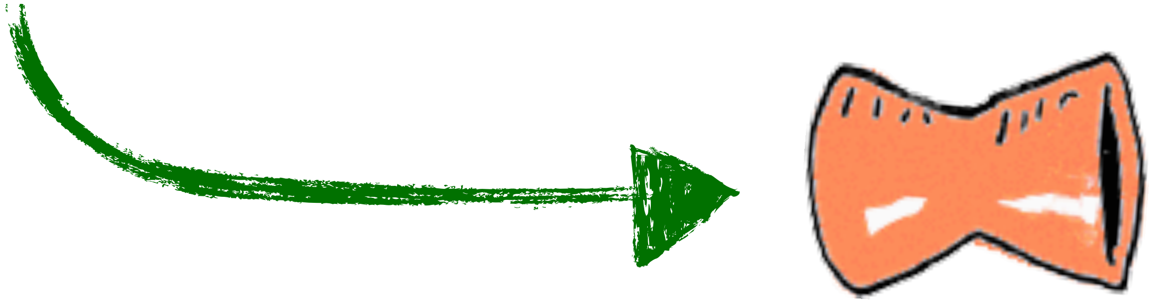
Local Connector$_n$

execution

Expected Global Behavior

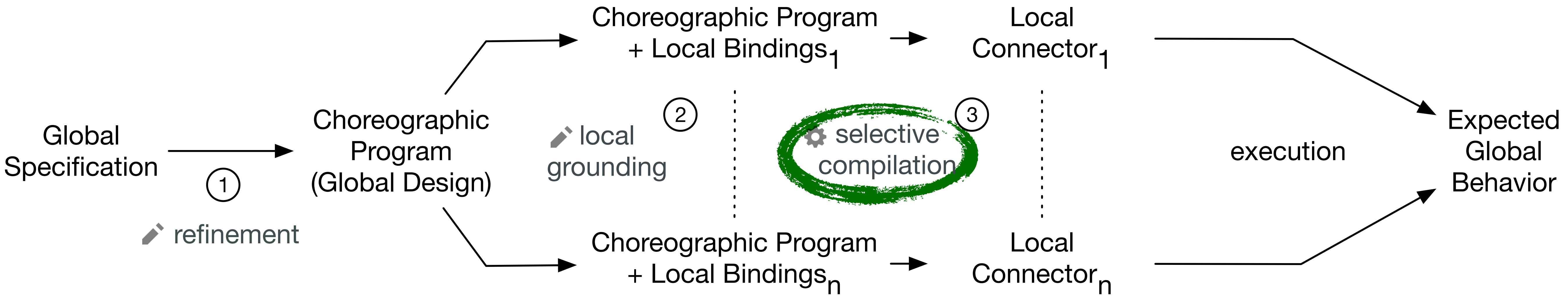
An Illustrative Example

```
deployment {
 getSchedule from "socket://intranet.schdls:8000" with SOAP
 getPosition from "socket://intranet.GPS:8001" with HTTP }

 locations {
   Admin: "reg-gov.org:80/BusCheckAdmin"
   DatabaseConnector: "reg-gov.org:80/BusCheckDB"

   //
 shd@BusAgency = getSchedule(line);
 passSchedule: BusAgency(shd) -> Tracker(shd);
 hasNext@Tracker = hasNextStop(shd);
 while(hasNext)@Tracker {
   pos@BusAgency = getPosition(line);
   passPosition: BusAgency(pos) -> Tracker(pos);
```



Compilation to an executable program

AIOCJ



Global Specification

1 refinement

Choreographic Program (Global Design)

local grounding 2

Choreographic Program + Local Bindings$_1$

Choreographic Program + Local Bindings$_n$

selective compilation 3

Local Connector$_1$

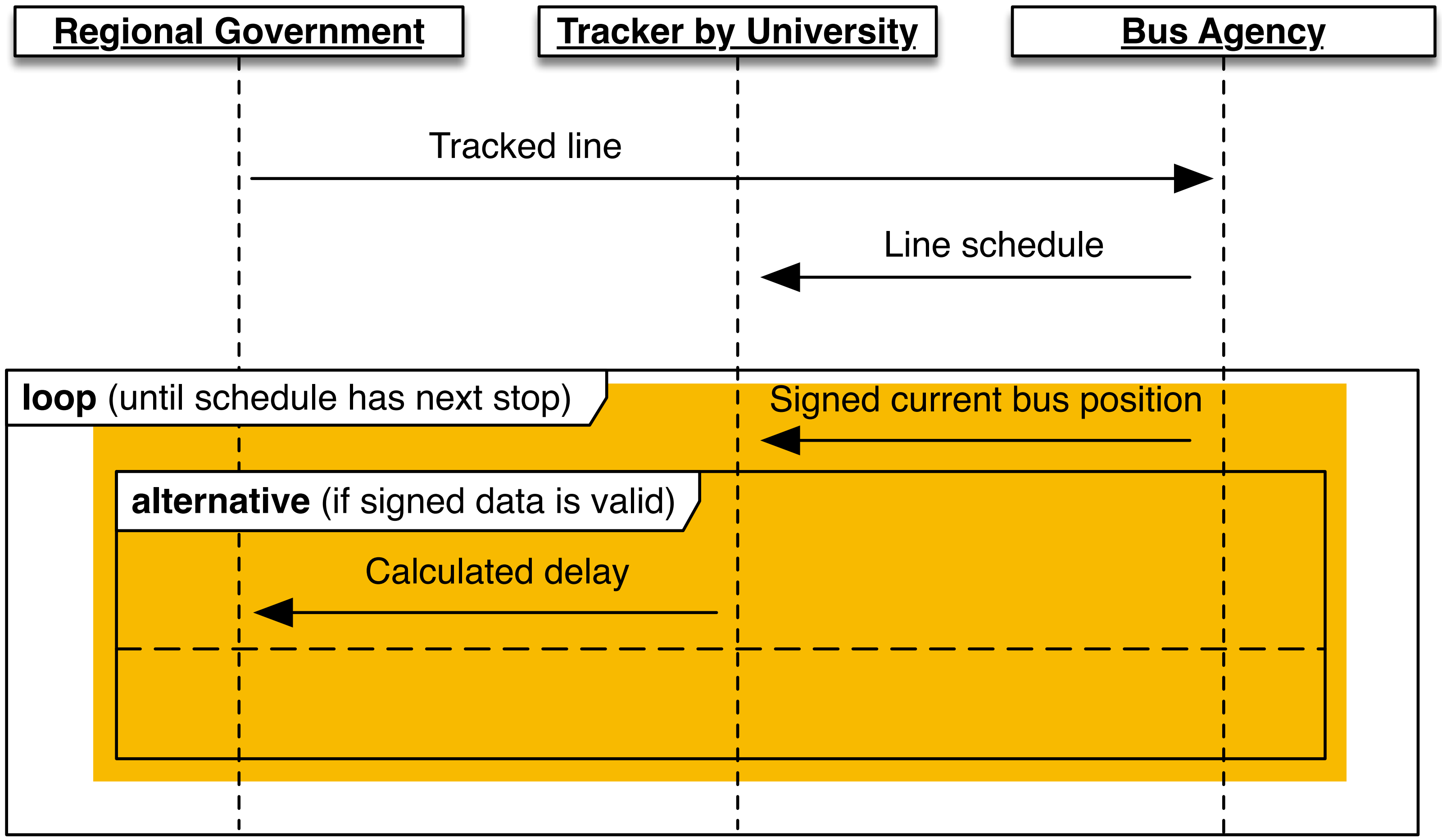
Local Connector$_n$

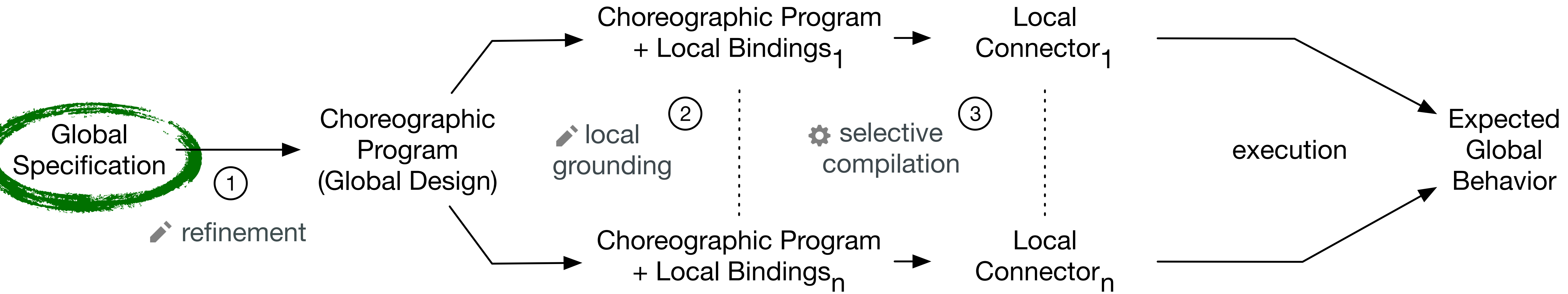
execution

Expected Global Behavior

Evolving ChIP systems

It’s a matter of re-iterating over (some) steps of the process

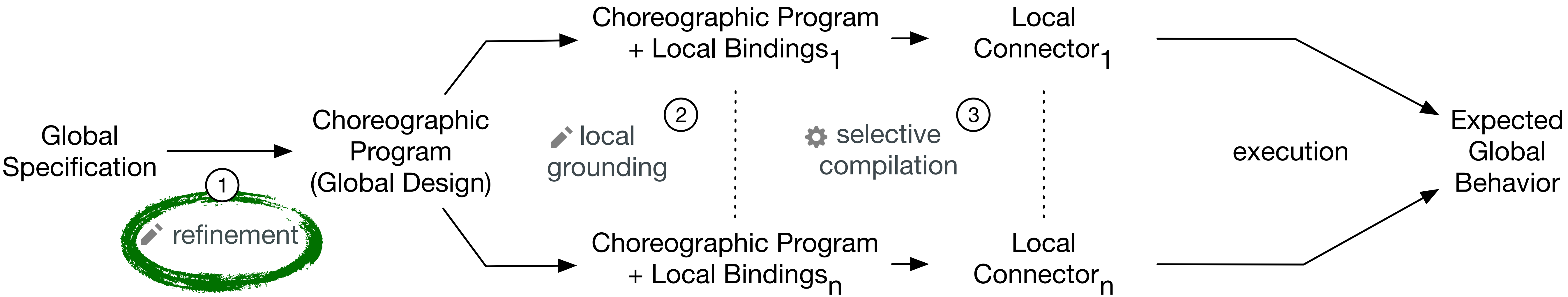

Regional Government
Tracker by University
Bus Agency
Tracked line
Line schedule
loop (until schedule has next stop)
Signed current bus position
alternative (if signed data is valid)
Calculated delay


Global Specification
1
refinement
Choreographic Program (Global Design)
Choreographic Program + Local Bindings$_1$
Choreographic Program + Local Bindings$_n$
local grounding
2
selective compilation
3
Local Connector$_1$
Local Connector$_n$
execution
Expected Global Behavior

Evolving ChIP systems

It’s a matter of re-iterating over (some) steps of the process

```
while(hasNext)@Tracker {
  pos@BusAgency = getPosition(line);
  signed_pos@BusAgency = sign(pos);
  passPosition: BusAgency(signed_pos) -> Tracker(pos);
  valid@Tracker = validate(pos);
  if( valid )@Tracker{
    delay@Tracker = calculateDelay(shd, pos);
    storeDelay: Tracker(delay) -> DatabaseConnector(delay);
    _@DatabaseConnector = insertDelay(line, delay)
  };
  hasNext@Tracker = hasNextStop(shd)
}
```



Global Specification

1 refinement

Choreographic Program (Global Design)

Choreographic Program + Local Bindings$_1$

Choreographic Program + Local Bindings$_n$

local grounding 2

selective compilation 3

Local Connector$_1$

Local Connector$_n$

execution

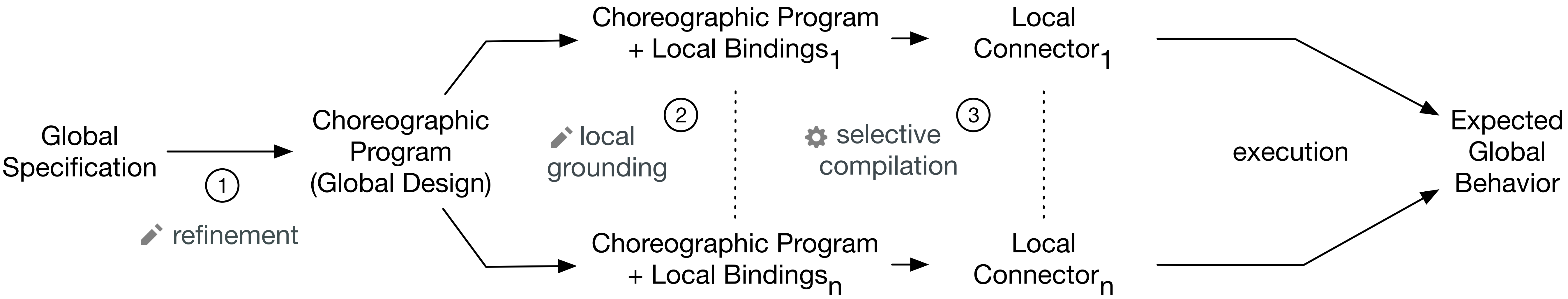
Expected Global Behavior

Evolving ChIP systems

It's a matter of re-iterating over (some) steps of the process

```
while(hasNext)@Tracker {
  pos@BusAgency = getPosition(line);
  signed_pos@BusAgency = sign(pos);
  passPosition: BusAgency(signed_pos) -> Tracker(pos);
  valid@Tracker = validate(pos);
  if( valid )@Tracker{
    delay@Tracker = calculateDelay(shd, pos);
    storeDelay: Tracker(delay) -> DatabaseConnector(delay);
    _@DatabaseConnector = insertDelay(line, delay)
  };
  hasNext@Tracker = hasNextStop(shd)
}
```

Only the BusAgency and the Tracker will update their local groundings and re-compile their connectors

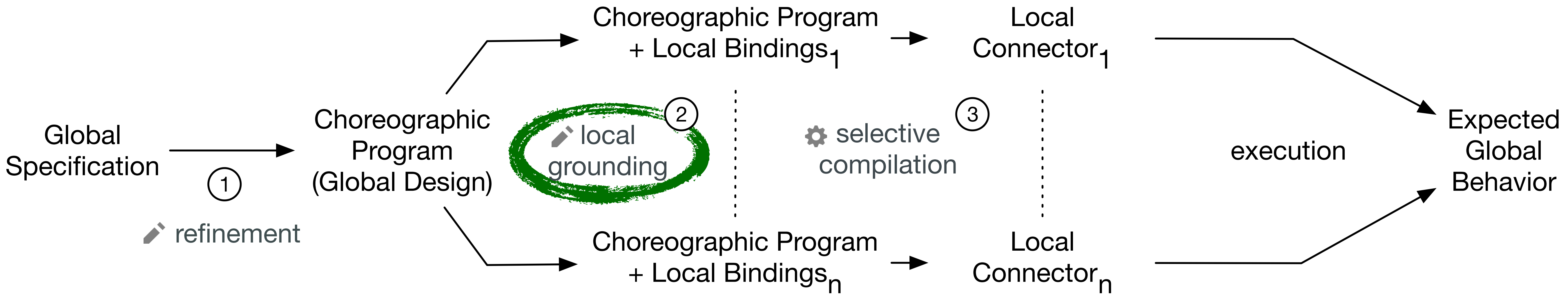


Evolving ChIP systems

It’s a matter of re-iterating over (some) steps of the process

```
while(hasNext)@Tracker {
  pos@BusAgency = getPosition(line);
  signed_pos@BusAgency = sign(pos);
  passPosition: BusAgency(signed_pos) -> Tracker(pos);
  valid@Tracker = validate(pos);
  if( valid )@Tracker{
    delay@Tracker = calculateDelay(shd, pos);
    storeDelay: Tracker(delay) -> DatabaseConnector(delay);
    _@DatabaseConnector = insertDelay(line, delay)
  };
  hasNext@Tracker = hasNextStop(shd)
}
```

Only the BusAgency and the Tracker will update their local groundings and re-compile their connectors

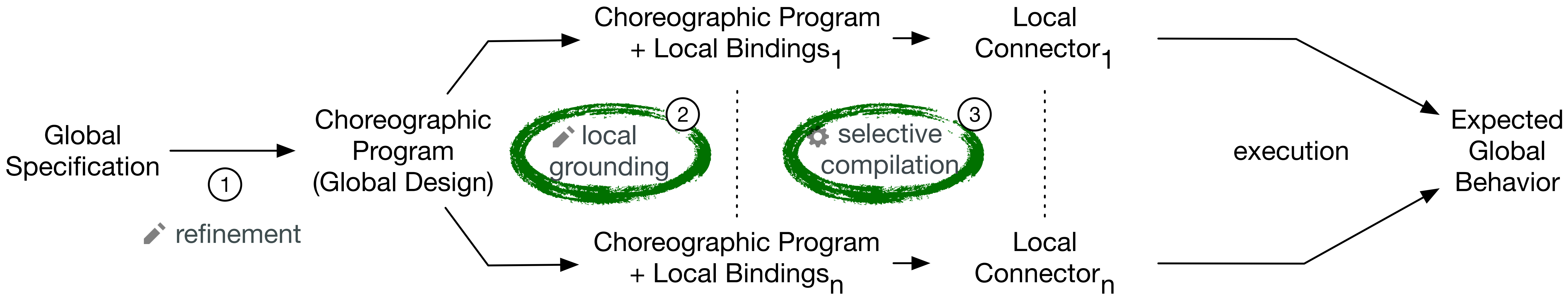


Evolving ChIP systems

It’s a matter of re-iterating over (some) steps of the process

```
while(hasNext)@Tracker {
  pos@BusAgency = getPosition(line);
  signed_pos@BusAgency = sign(pos);
  passPosition: BusAgency(signed_pos) -> Tracker(pos);
  valid@Tracker = validate(pos);
  if( valid )@Tracker{
    delay@Tracker = calculateDelay(shd, pos);
    storeDelay: Tracker(delay) -> DatabaseConnector(delay);
    _@DatabaseConnector = insertDelay(line, delay)
  };
  hasNext@Tracker = hasNextStop(shd)
}
```

Only the BusAgency and the Tracker will update their local groundings and re-compile their connectors



Global Specification

1 refinement

Choreographic Program (Global Design)

Choreographic Program + Local Bindings$_1$

Choreographic Program + Local Bindings$_n$

2 local grounding

Local Connector$_1$

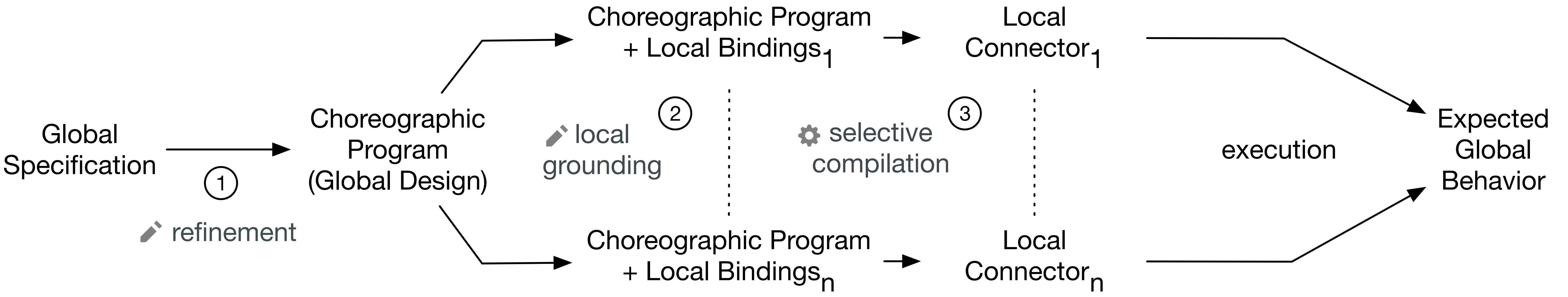
Local Connector$_n$

3 selective compilation

execution

Expected Global Behavior

Conclusion and Future Work



Global Specification

1 refinement

Choreographic Program (Global Design)

2 local grounding

Choreographic Program + Local Bindings$_1$

Choreographic Program + Local Bindings$_n$

3 selective compilation

Local Connector$_1$

Local Connector$_n$

execution

Expected Global Behavior

- Which specification languages are **better suited*** for the **ChIP refinement process**?
- Can we provide **guidelines** for the refinement?
- Can we **automatise** (parts of) the **refinement**?

- Enhance AIOCJ with **richer data structures** and **type system**
- Capture other aspects of the design:
  - **Security**
  - **Exceptional** behaviours
  - **Non-functional properties**
  - …

* ease of transition, expressiveness, automatisation, …