ESOP 2016, Eindhoven, Netherlands; 20160405

Coordinated Concurrent Programming in SYNDICATE

Tony Garnock-Jones ✉
tonyg@ccs.neu.edu
Northeastern University

Matthias Felleisen
matthias@ccs.neu.edu
Northeastern University

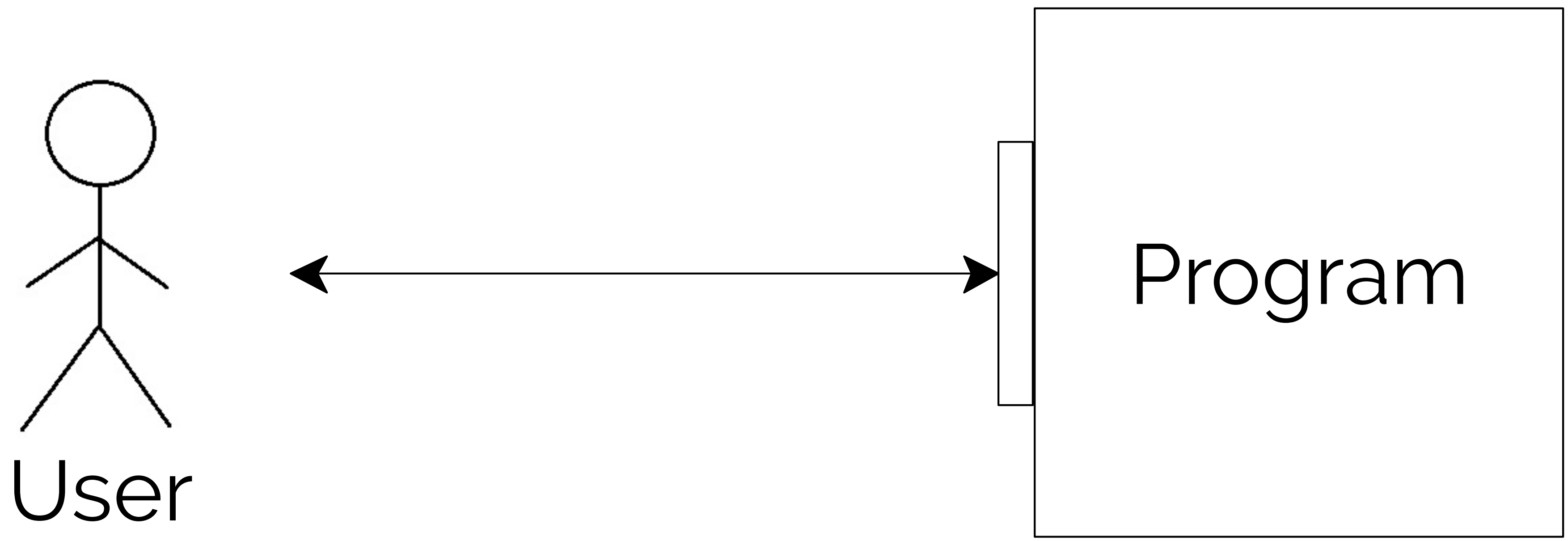
How can we organise our interactive programs?

How can we organise our interactive programs?

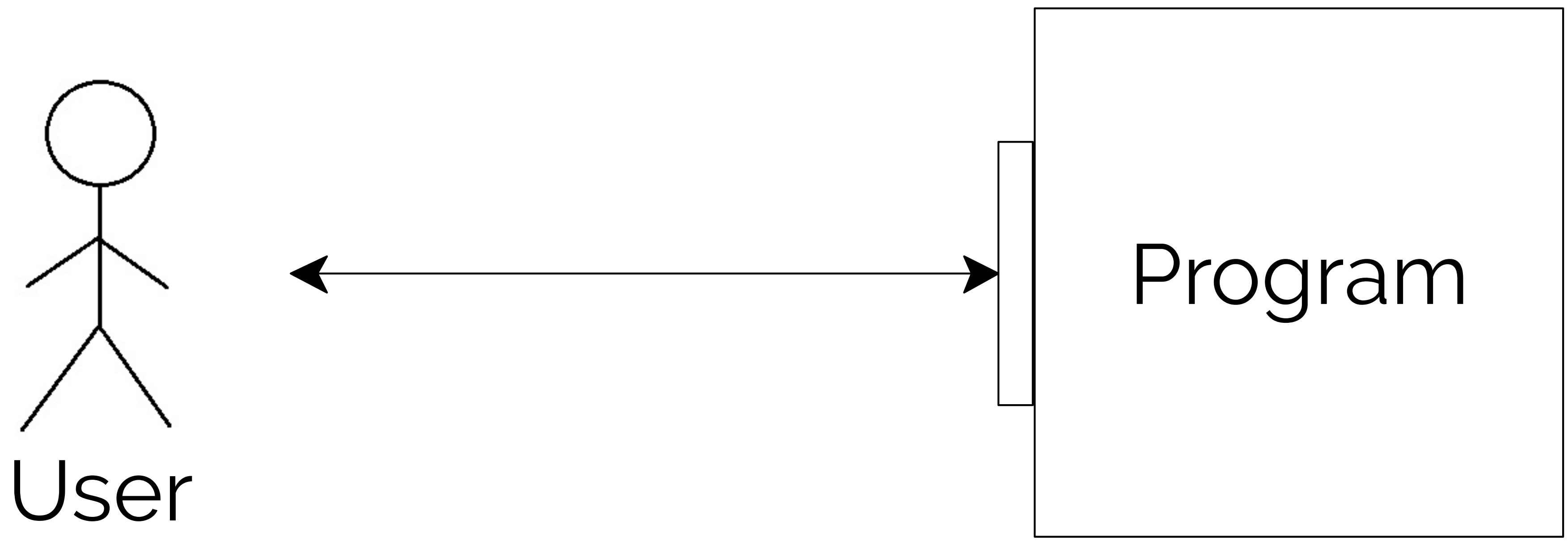
With a programming language!

How can we organise our interactive programs?

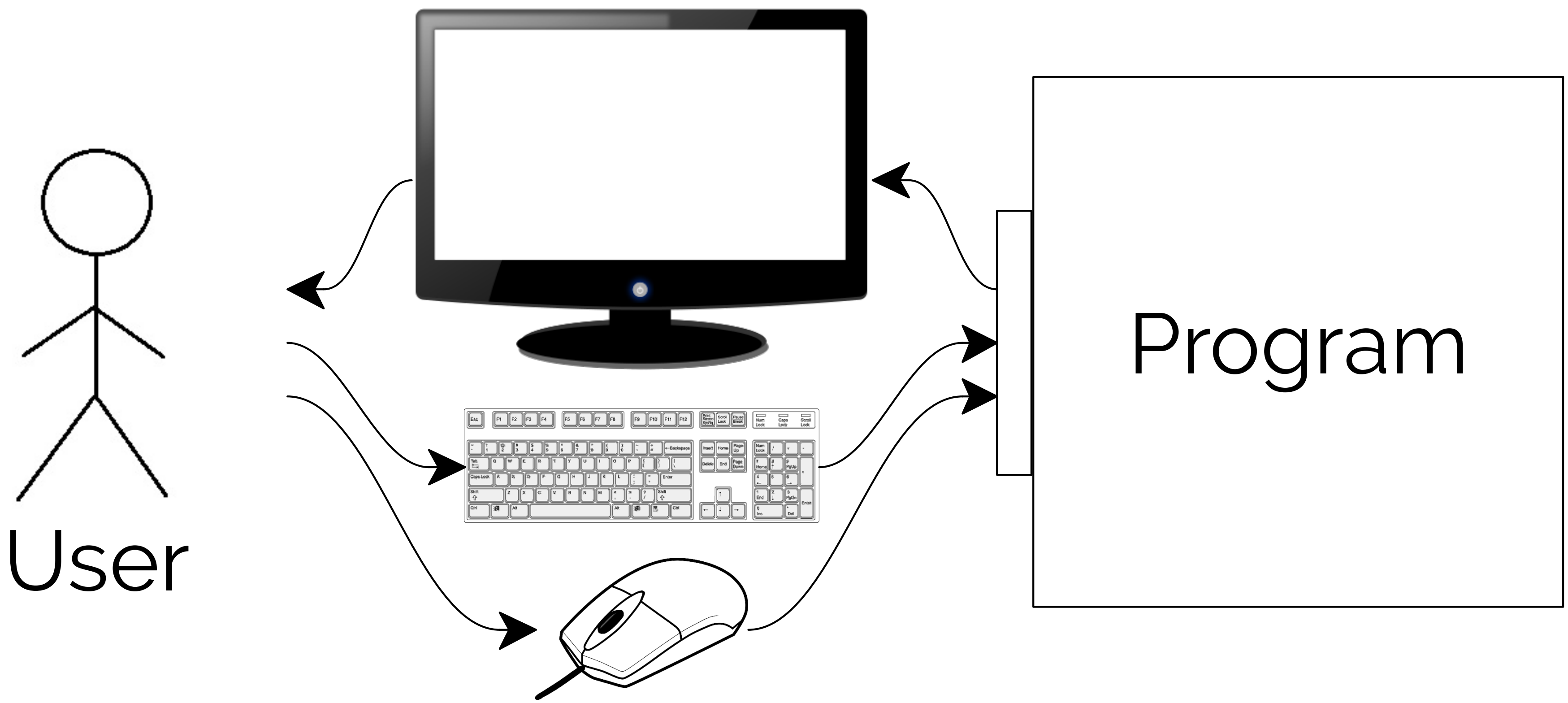
SYNDICATE



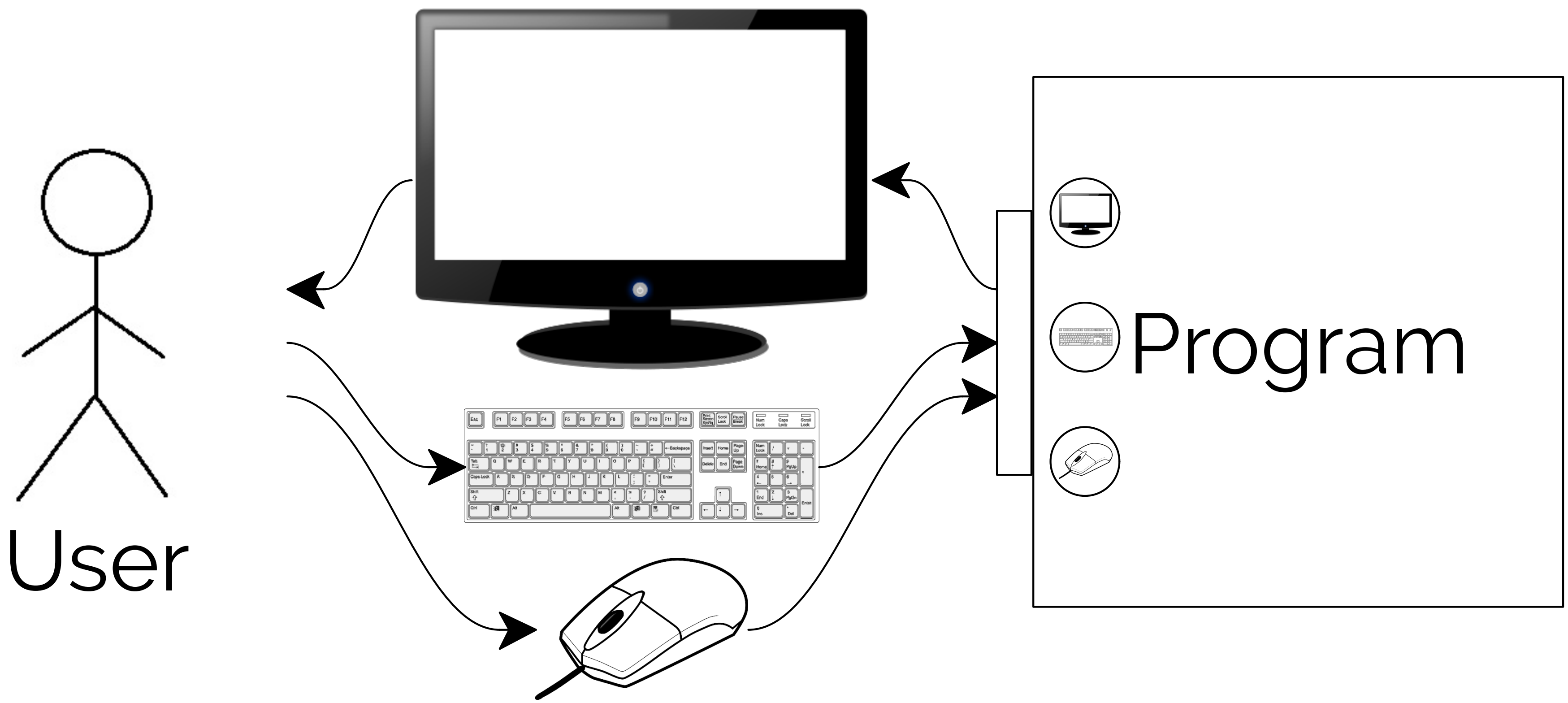
Interactive System



External Concurrency

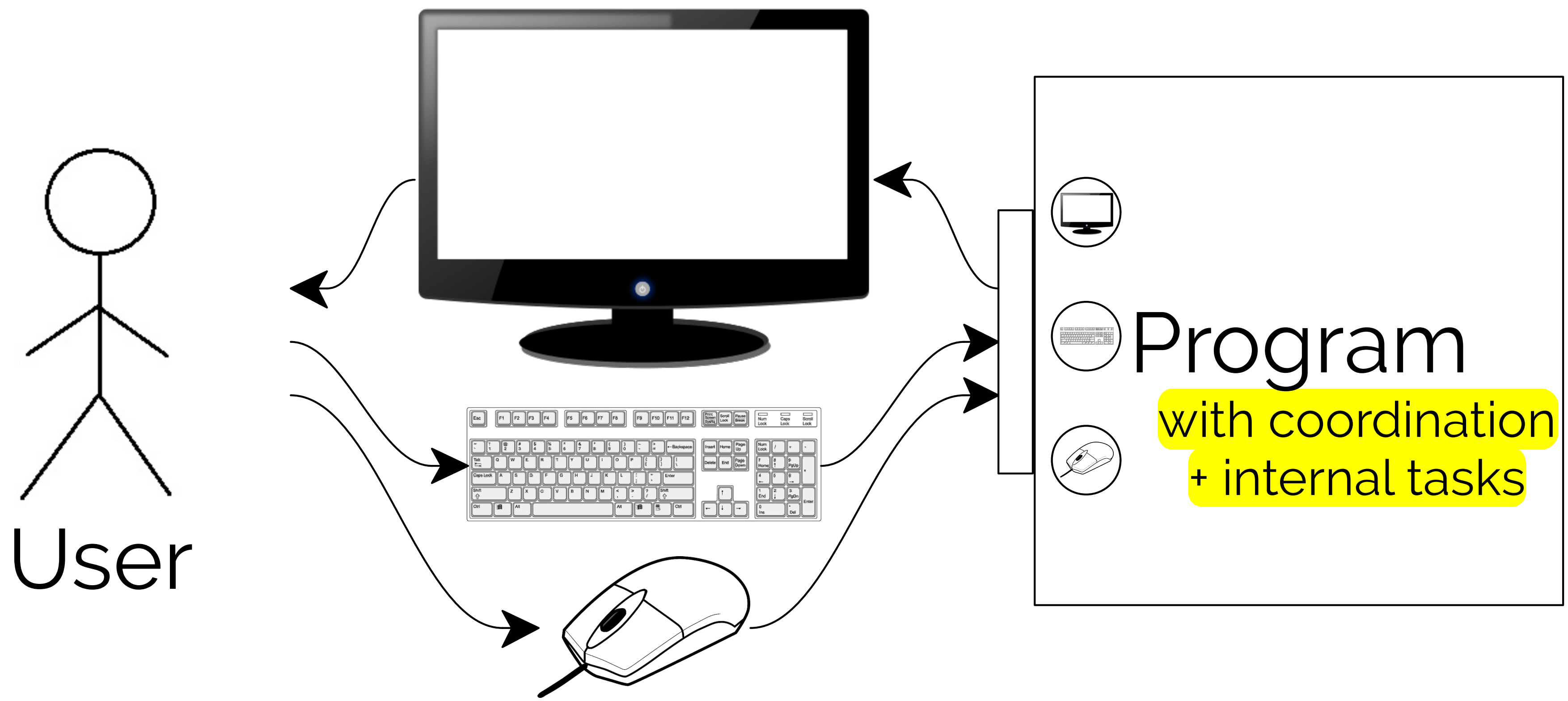


Lots of External Concurrency



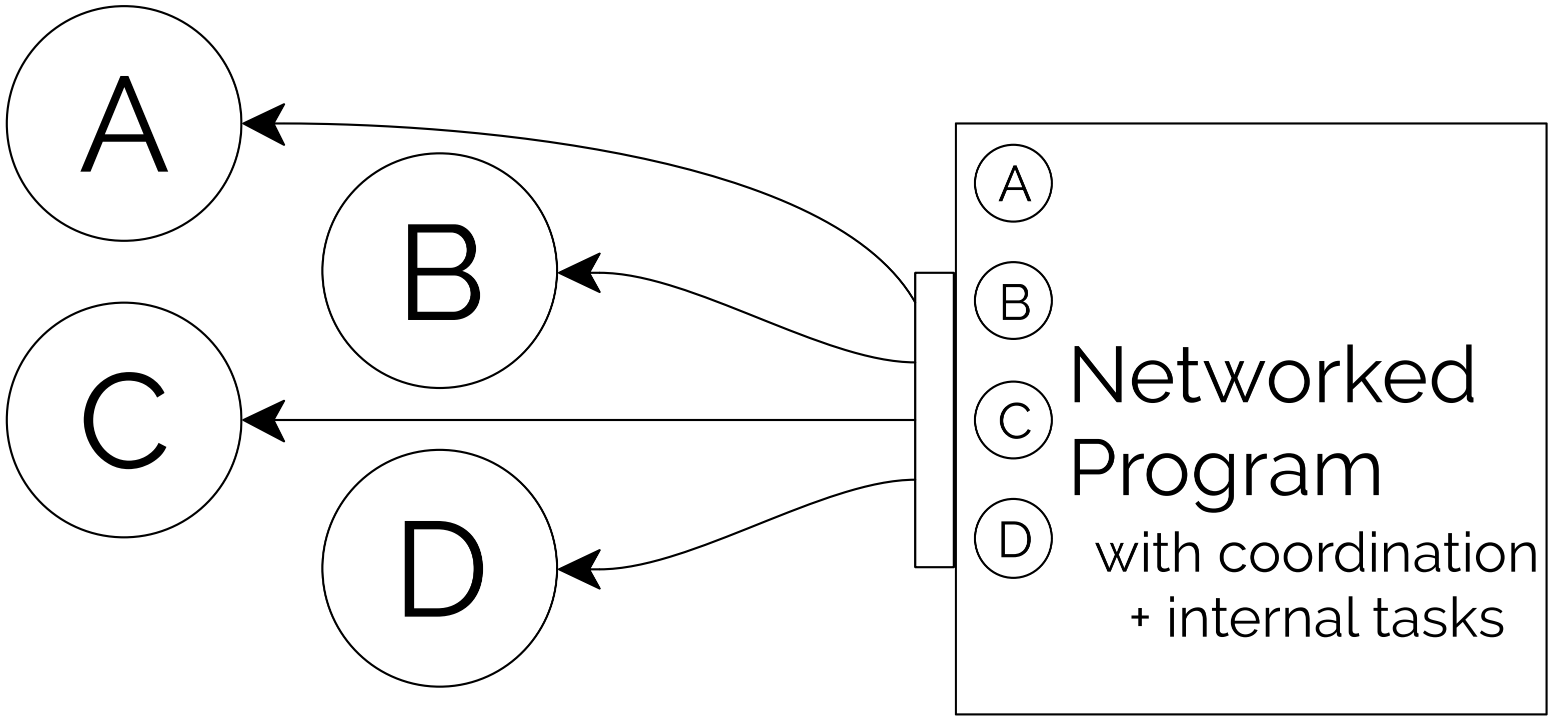
Lots of External Concurrency

Internal Organisation Reflects External Concurrency



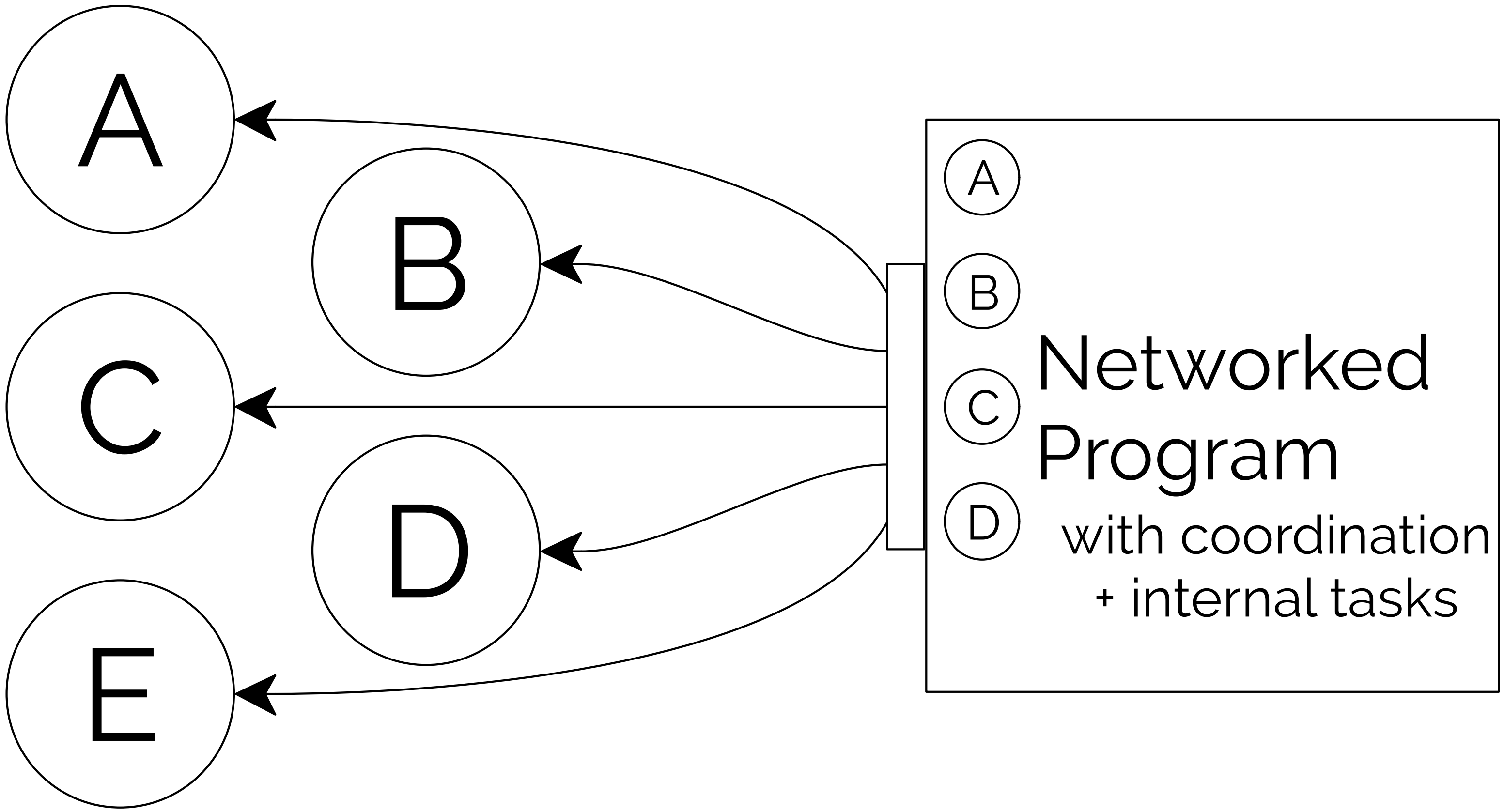
Lots of External Concurrency

Internal Organisation Reflects External Concurrency



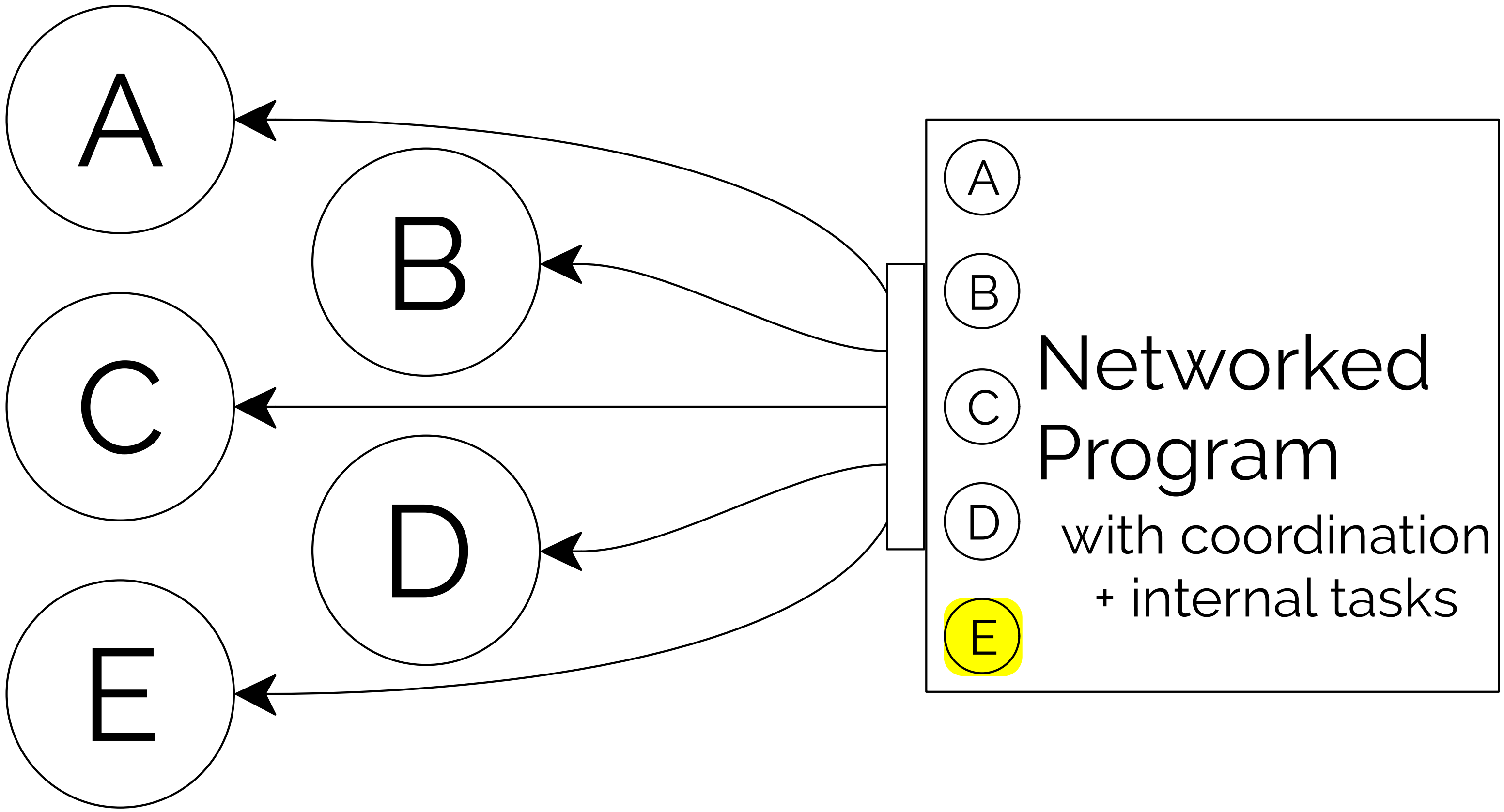
Lots of External Concurrency

Internal Organisation Reflects External Concurrency



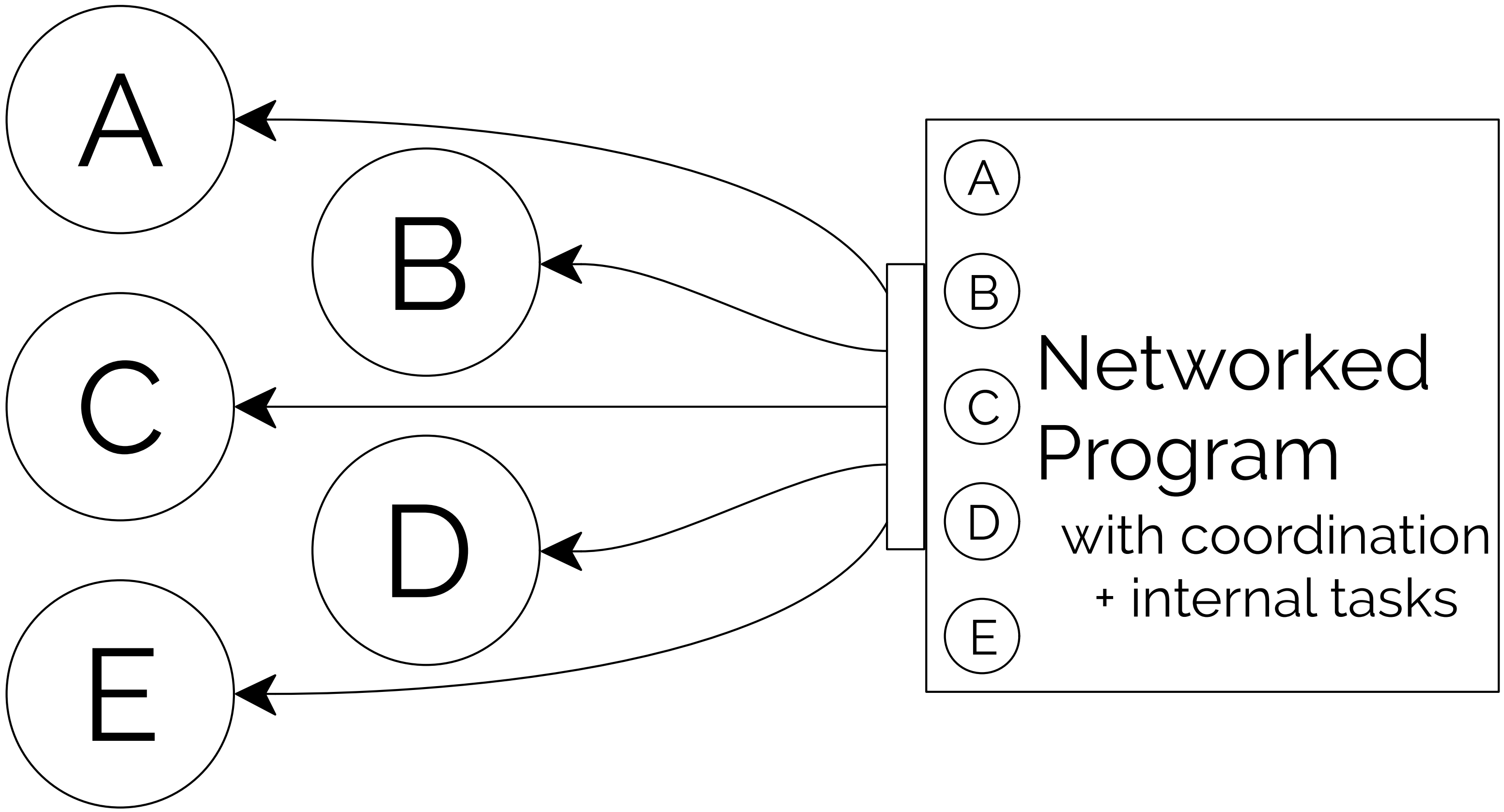
Lots of Dynamic, External Concurrency

Component startup → interaction → shutdown/failure



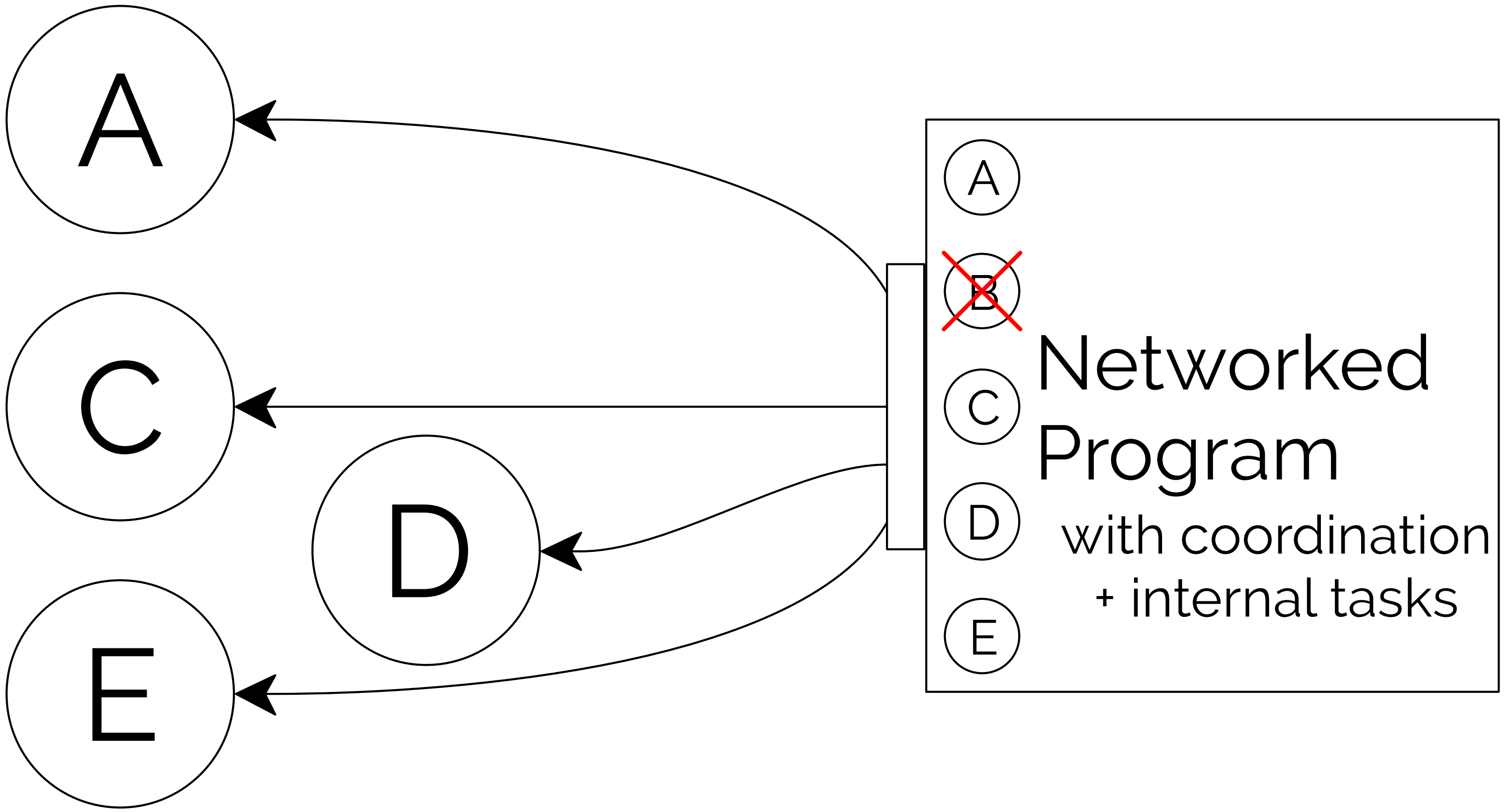
Lots of Dynamic, External Concurrency

Component startup → interaction → shutdown/failure



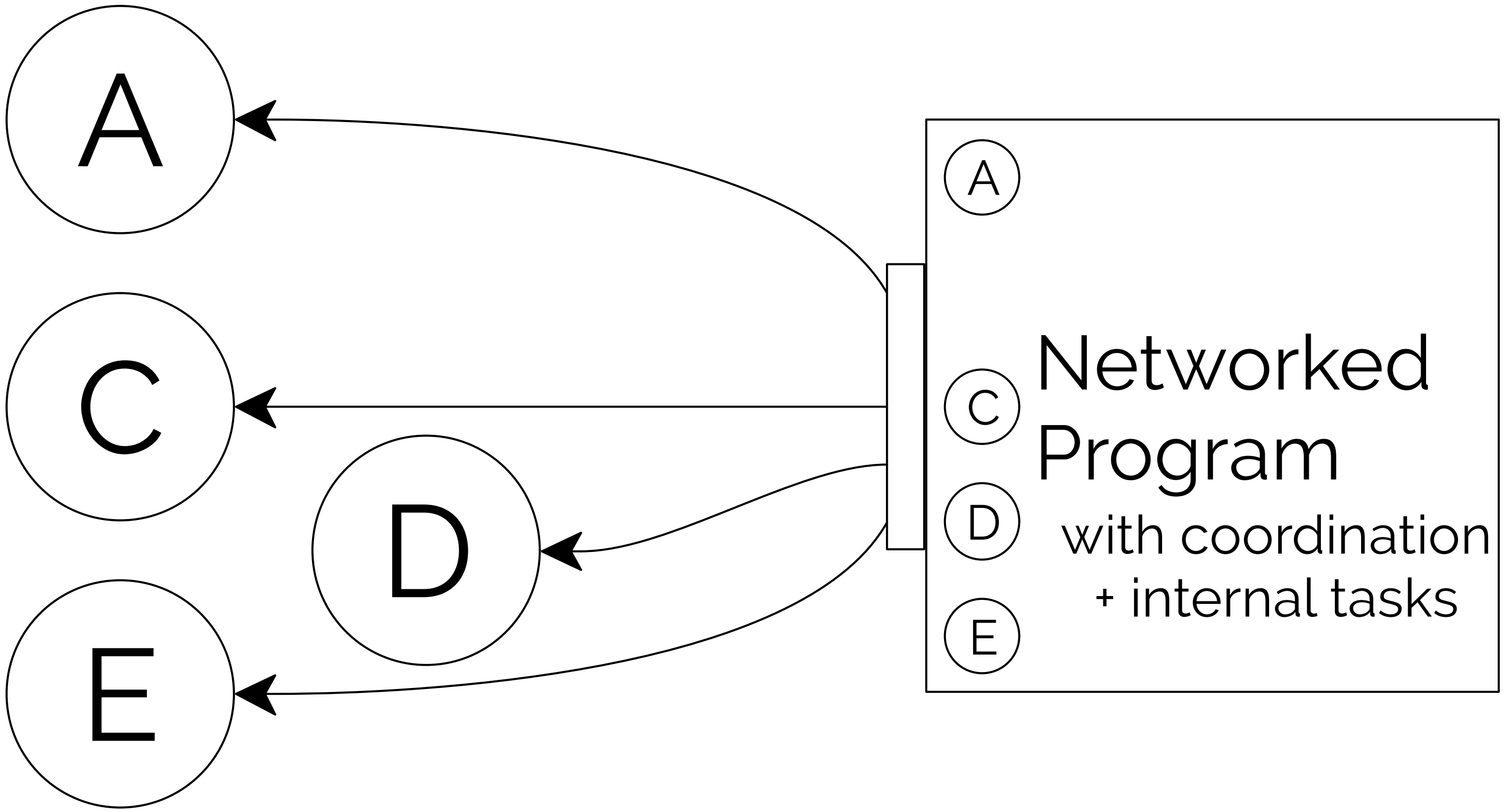
Lots of Dynamic, External Concurrency

Component startup → interaction → shutdown/failure



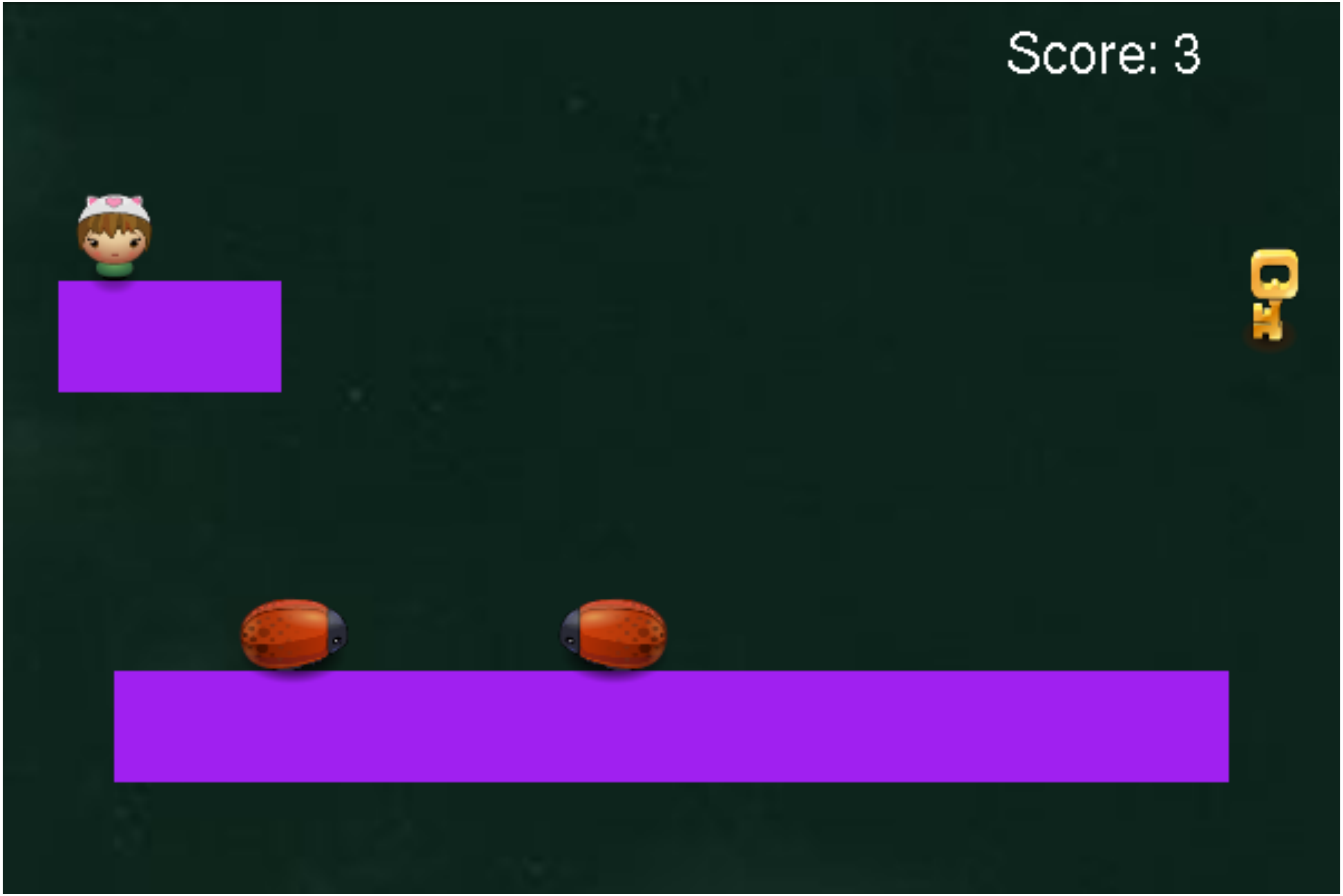
Lots of Dynamic, External Concurrency

Component startup → interaction → shutdown/failure



Lots of Dynamic, External Concurrency

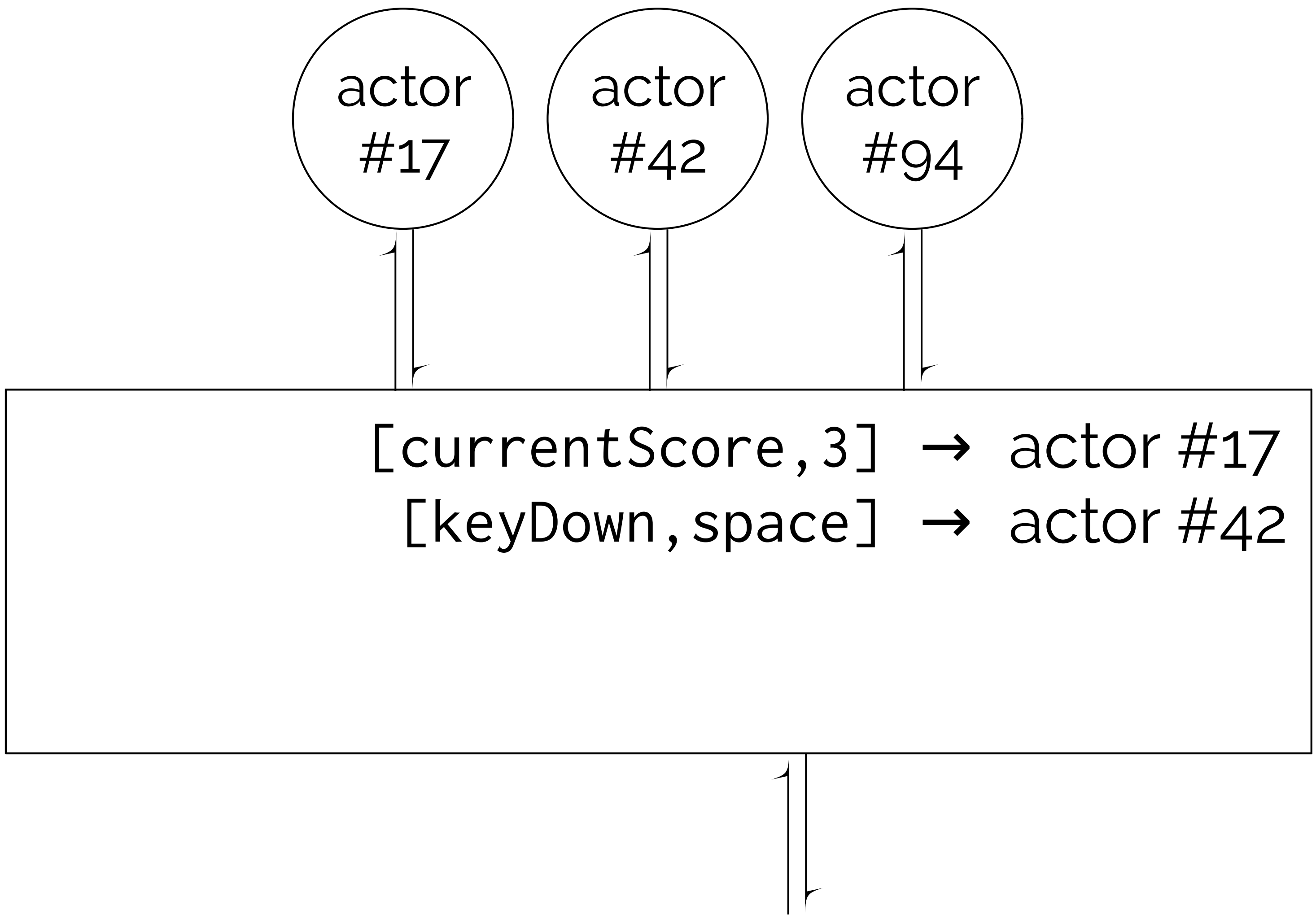
Component startup → interaction → shutdown/failure


Score: 3

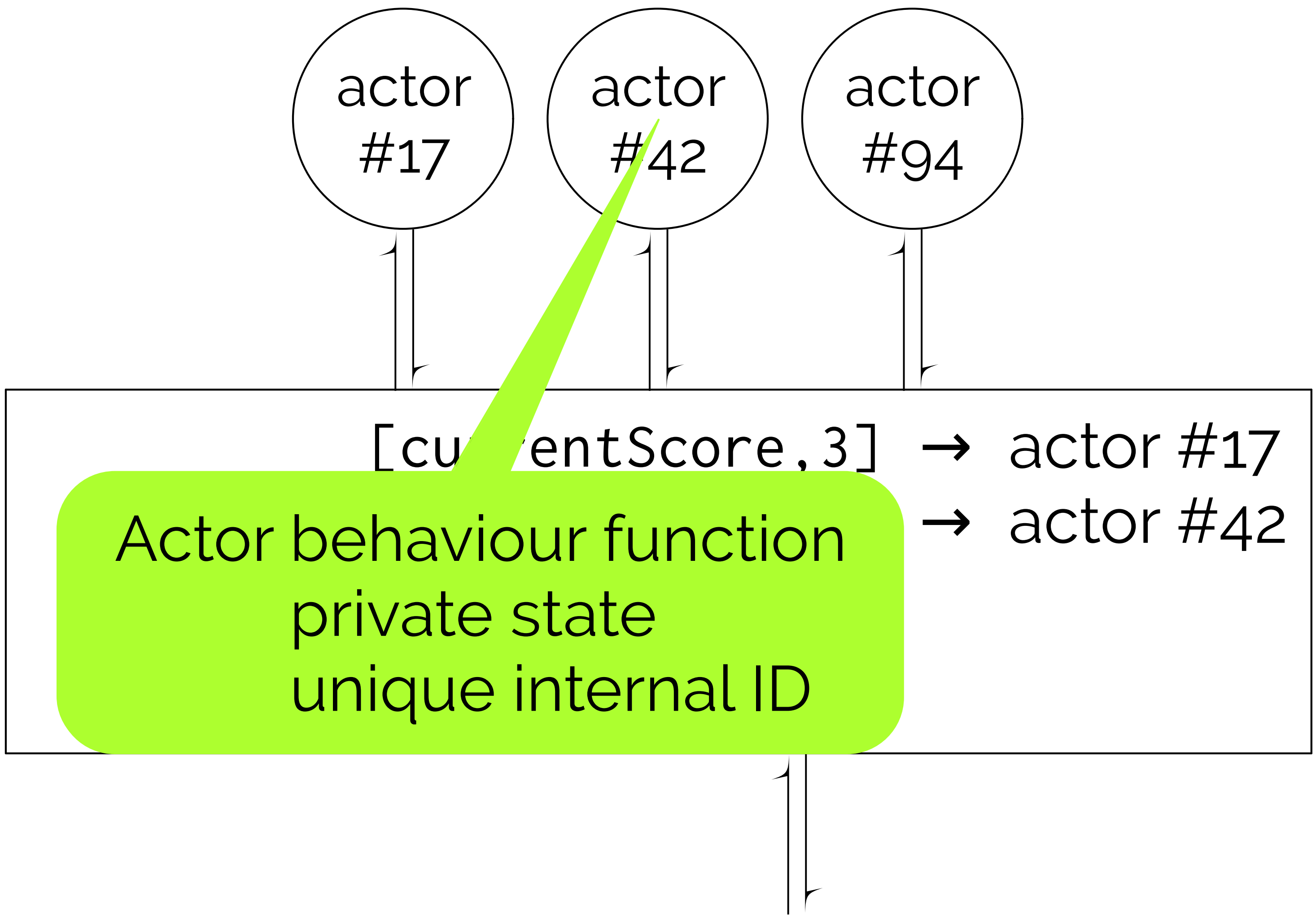
Syndicate

event × state → [action] × state

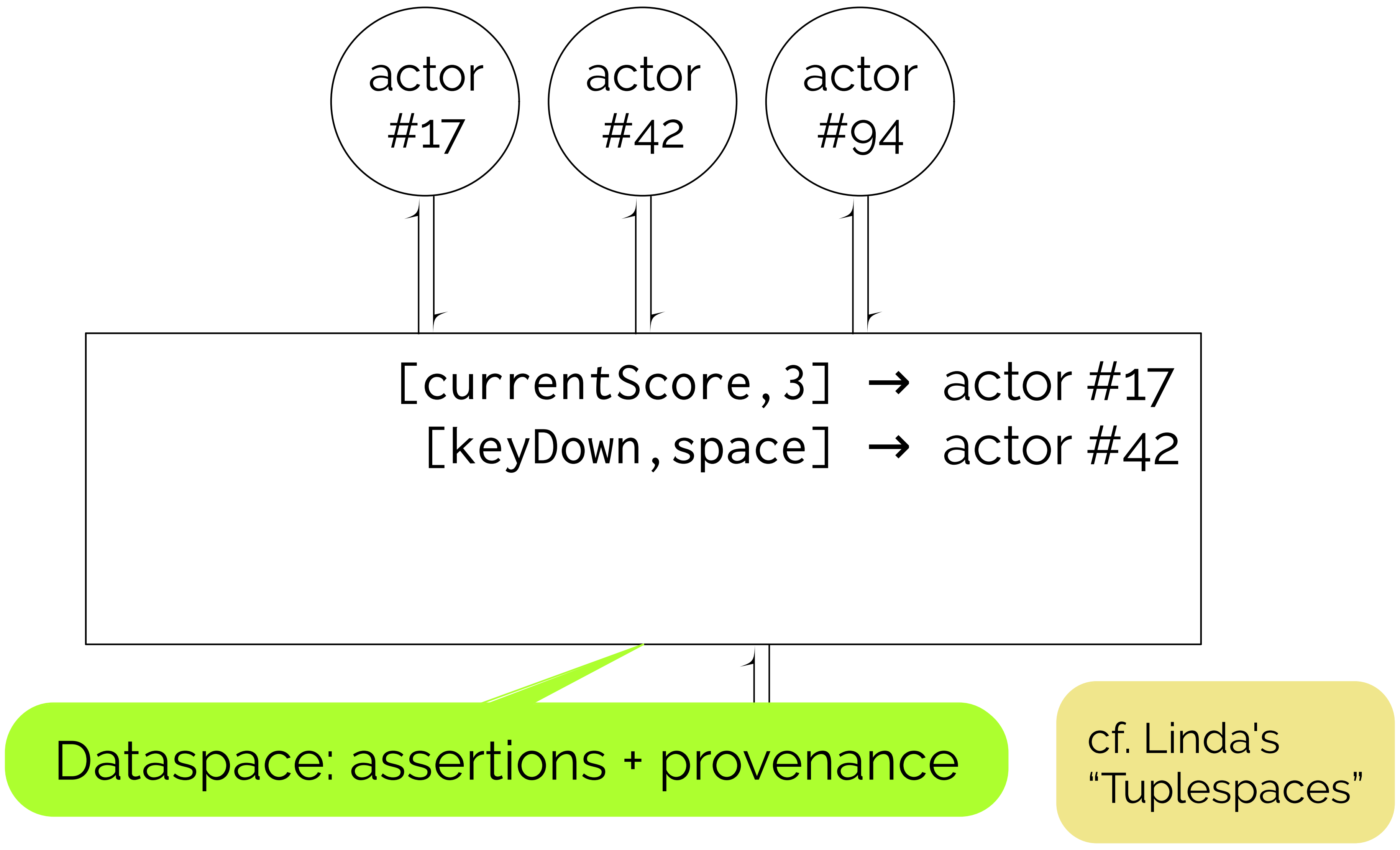
event × state → [action] × state



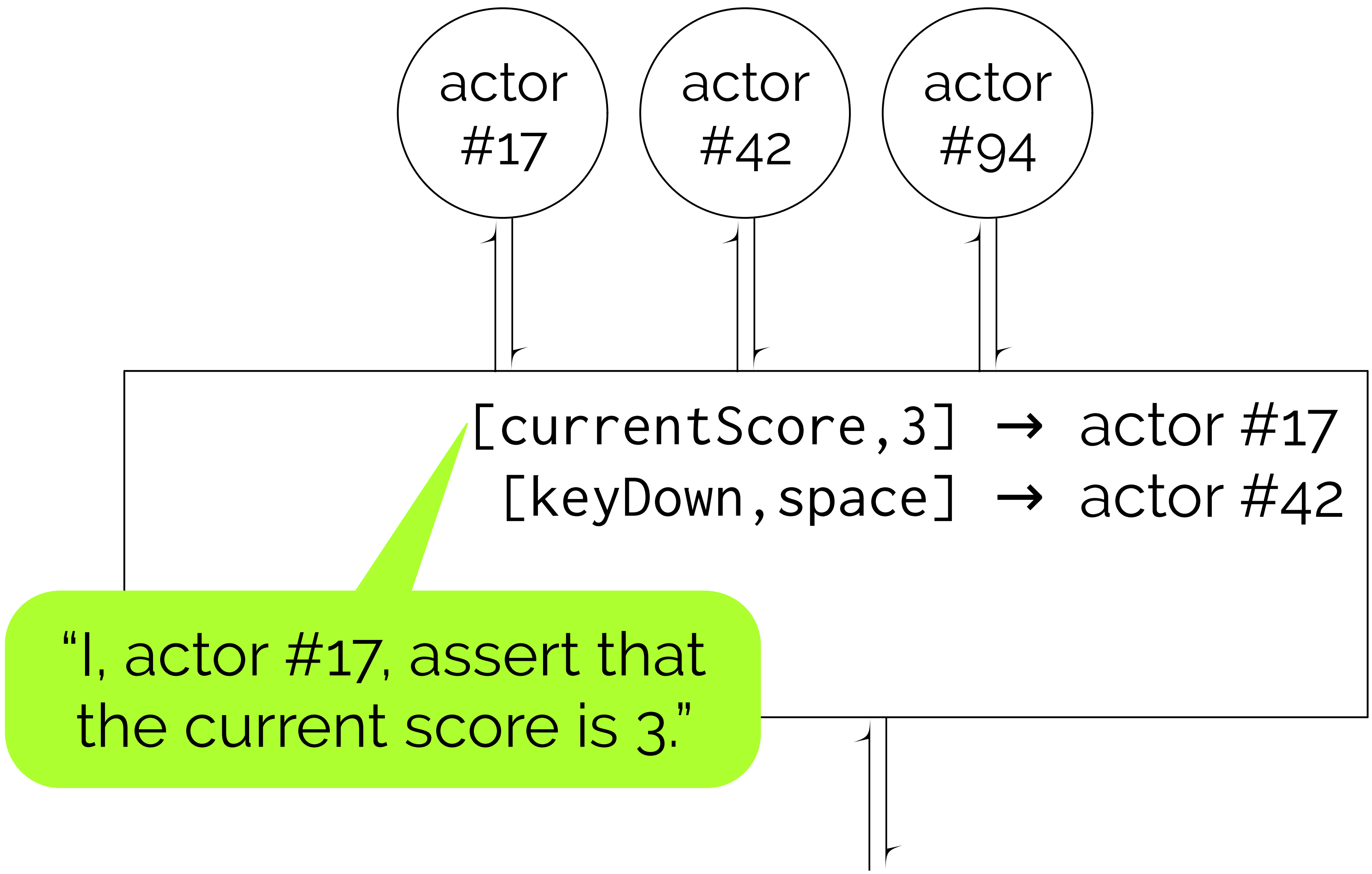
event × state → [action] × state



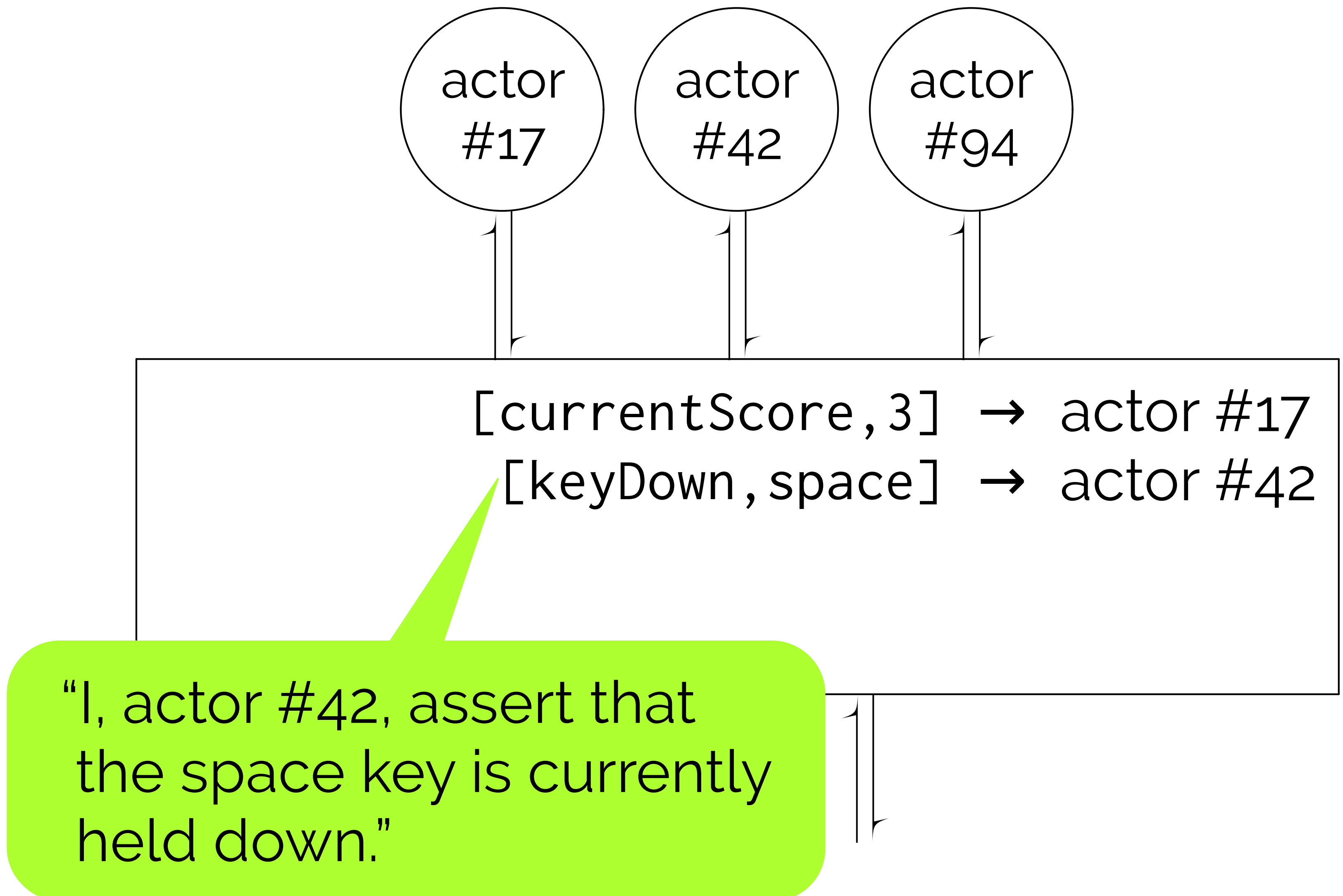
event × state → [action] × state



event × state → [action] × state

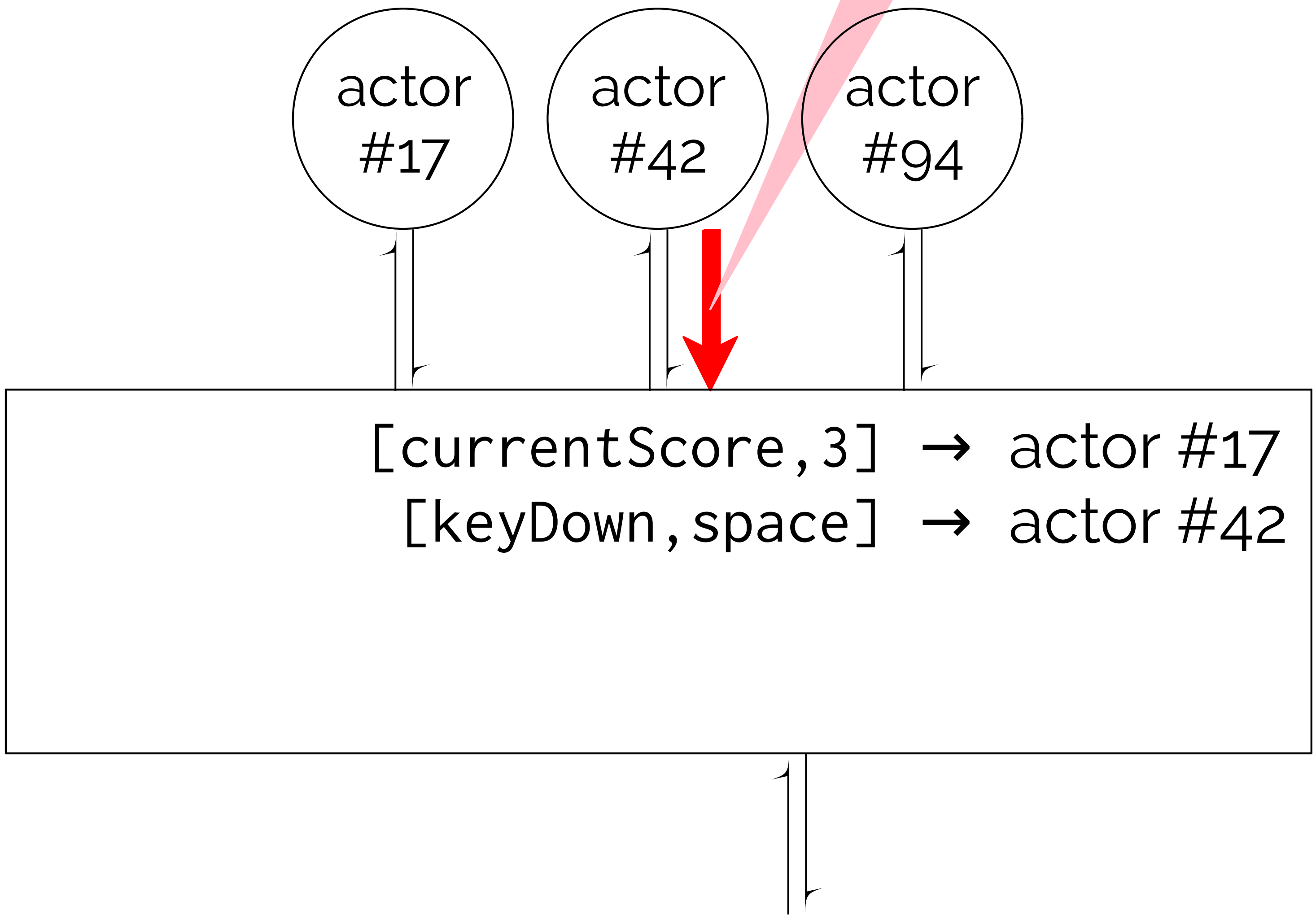


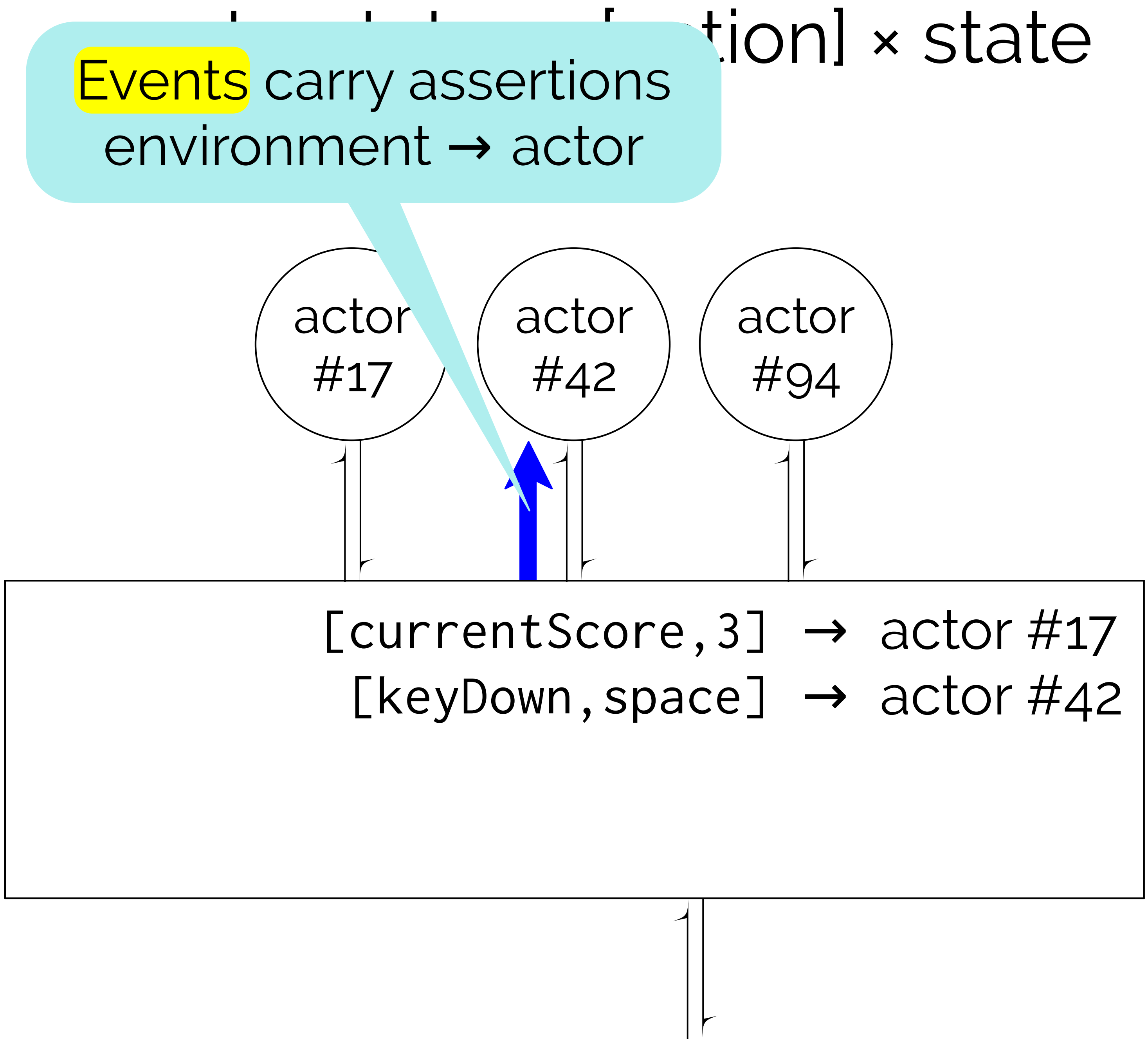
event × state → [action] × state

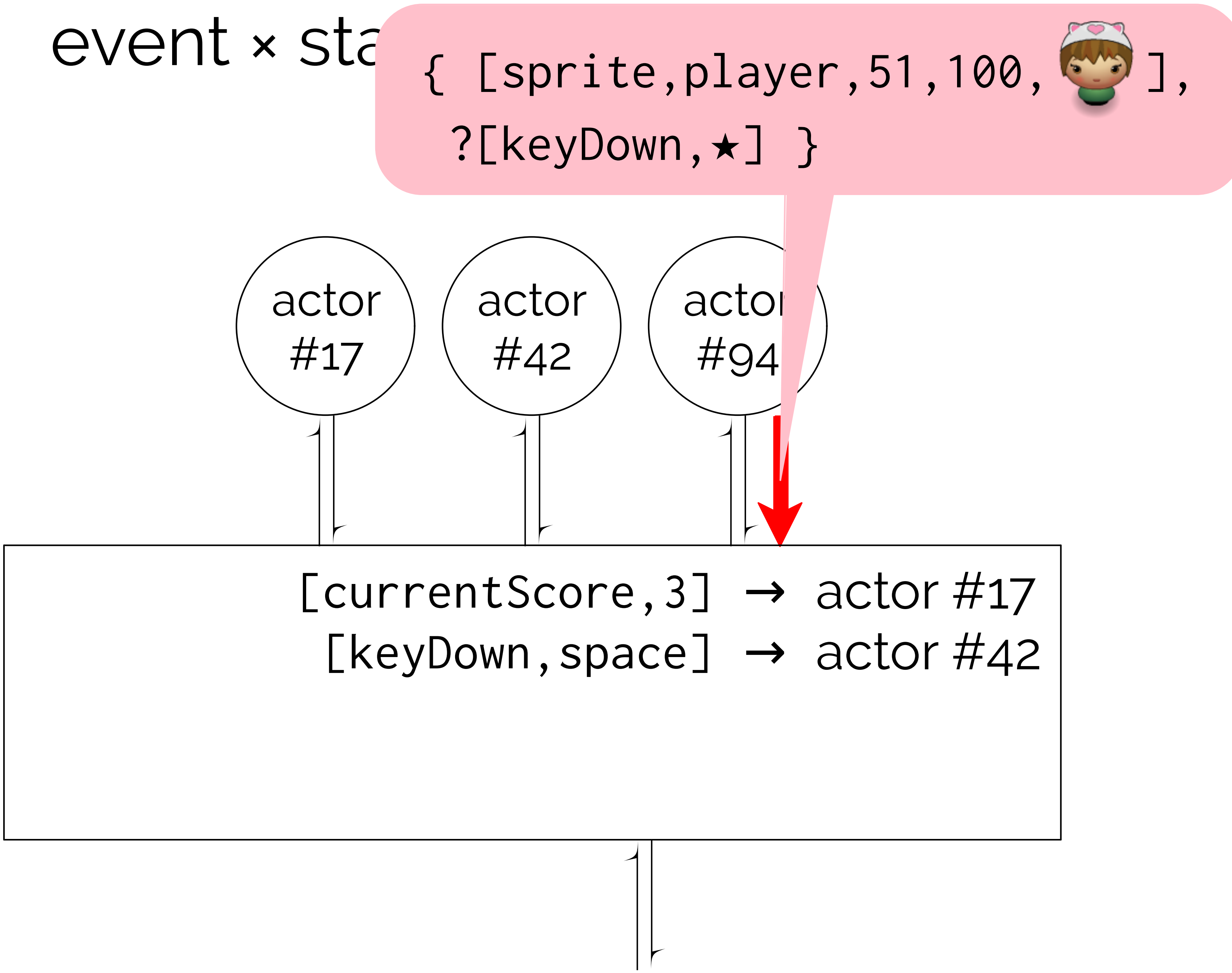


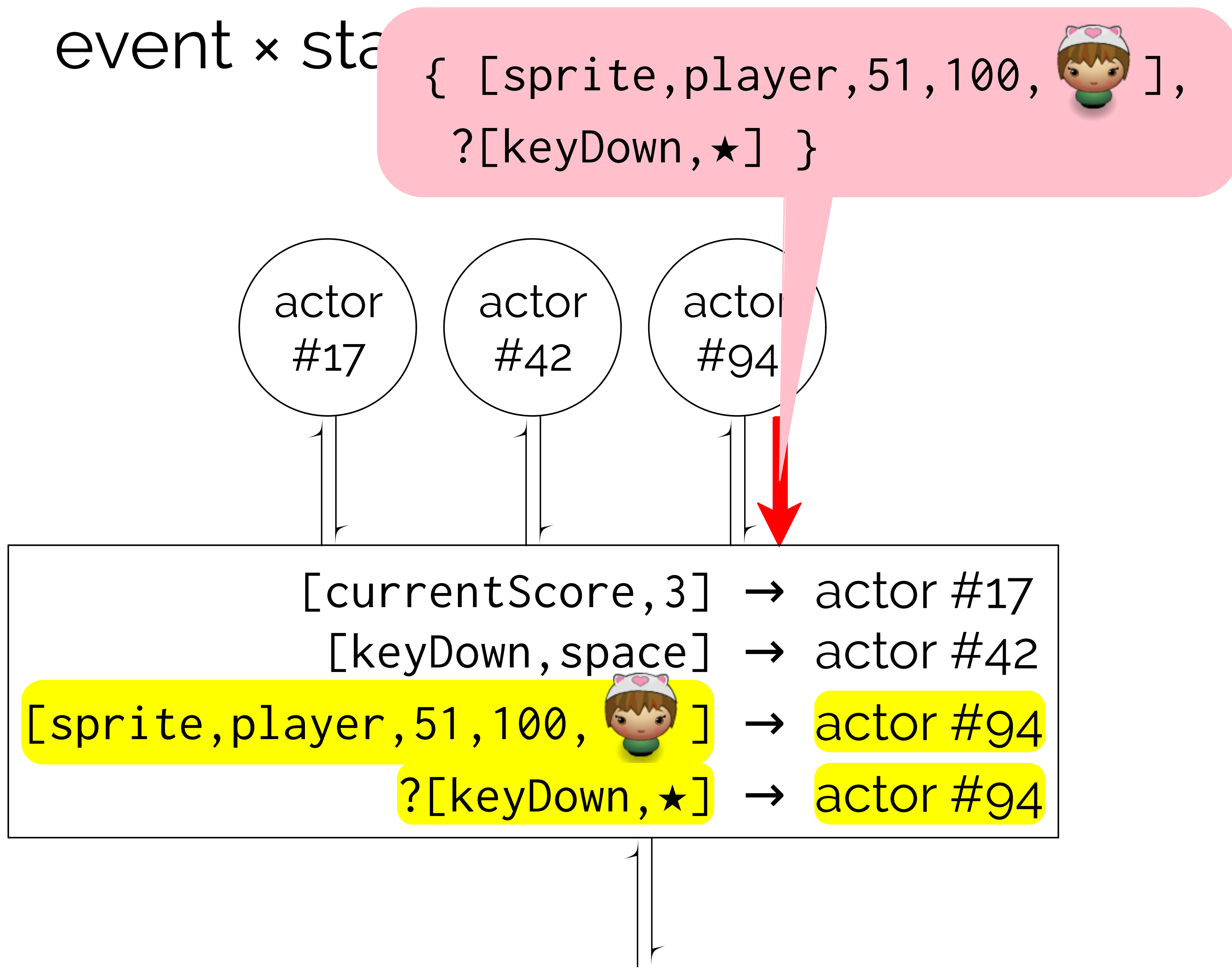
event × stat

Actions carry assertions
actor → environment



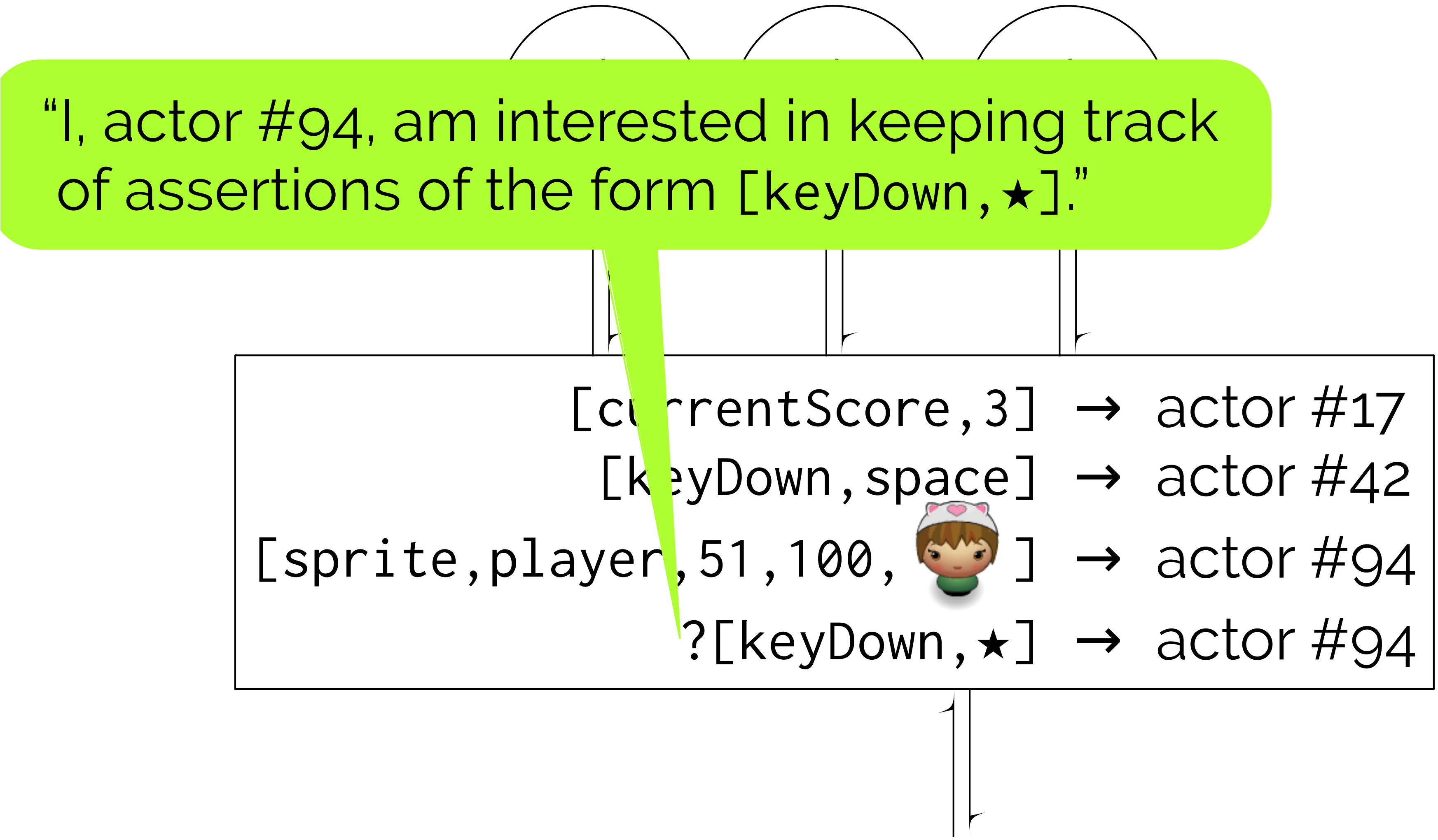

tion] × state
Events carry assertions
environment → actor
actor #17
actor #42
actor #94
[currentScore,3] → actor #17
[keyDown,space] → actor #42


event × sta
{ [sprite,player,51,100,],
?[keyDown,★] }
actor #17
actor #42
actor #94
[currentScore,3] → actor #17
[keyDown,space] → actor #42

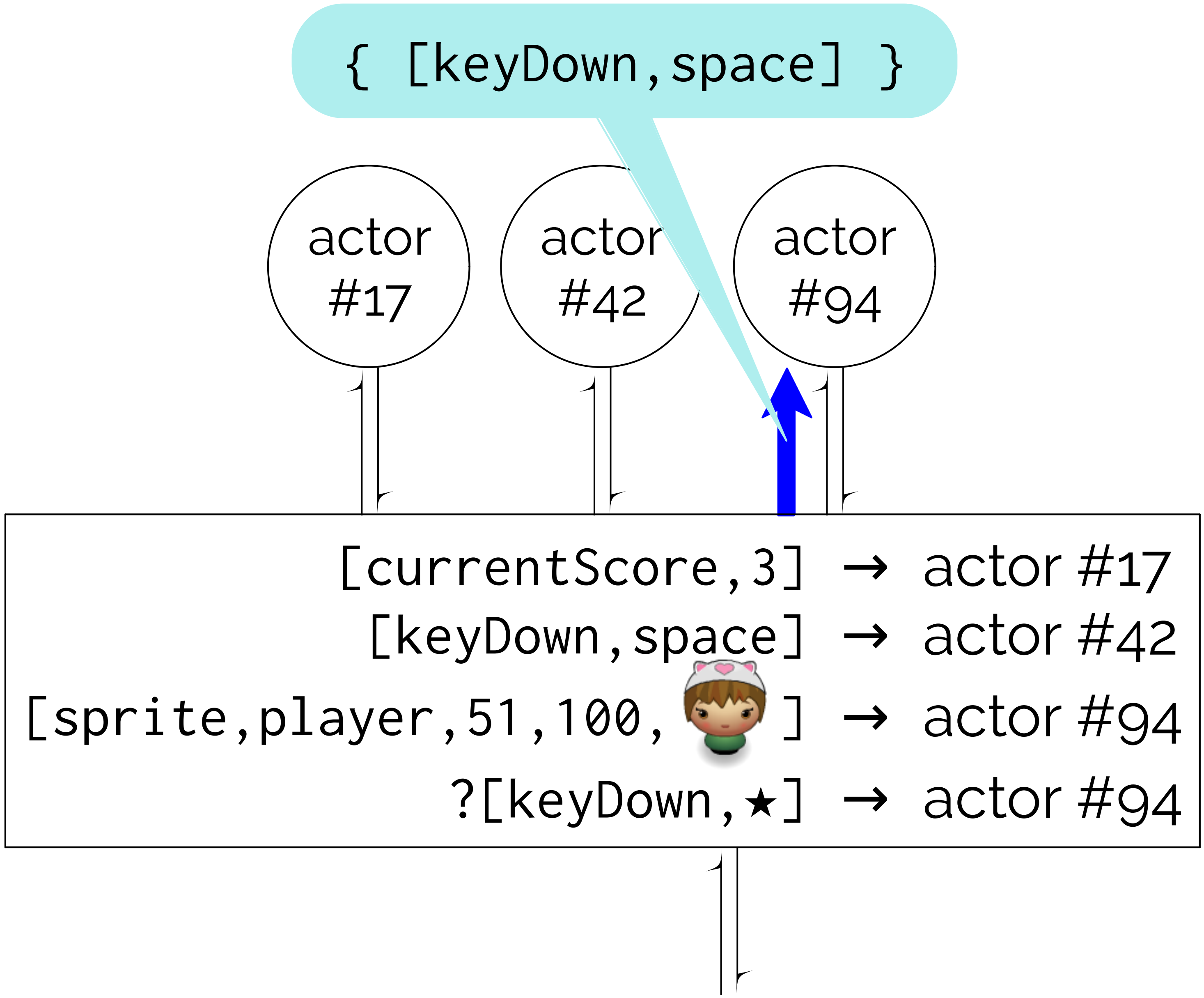


event × sta
{ [sprite,player,51,100,],
?[keyDown,★] }
actor #17
actor #42
actor #94
[currentScore,3] → actor #17
[keyDown,space] → actor #42
[sprite,player,51,100,] → actor #94
?[keyDown,★] → actor #94

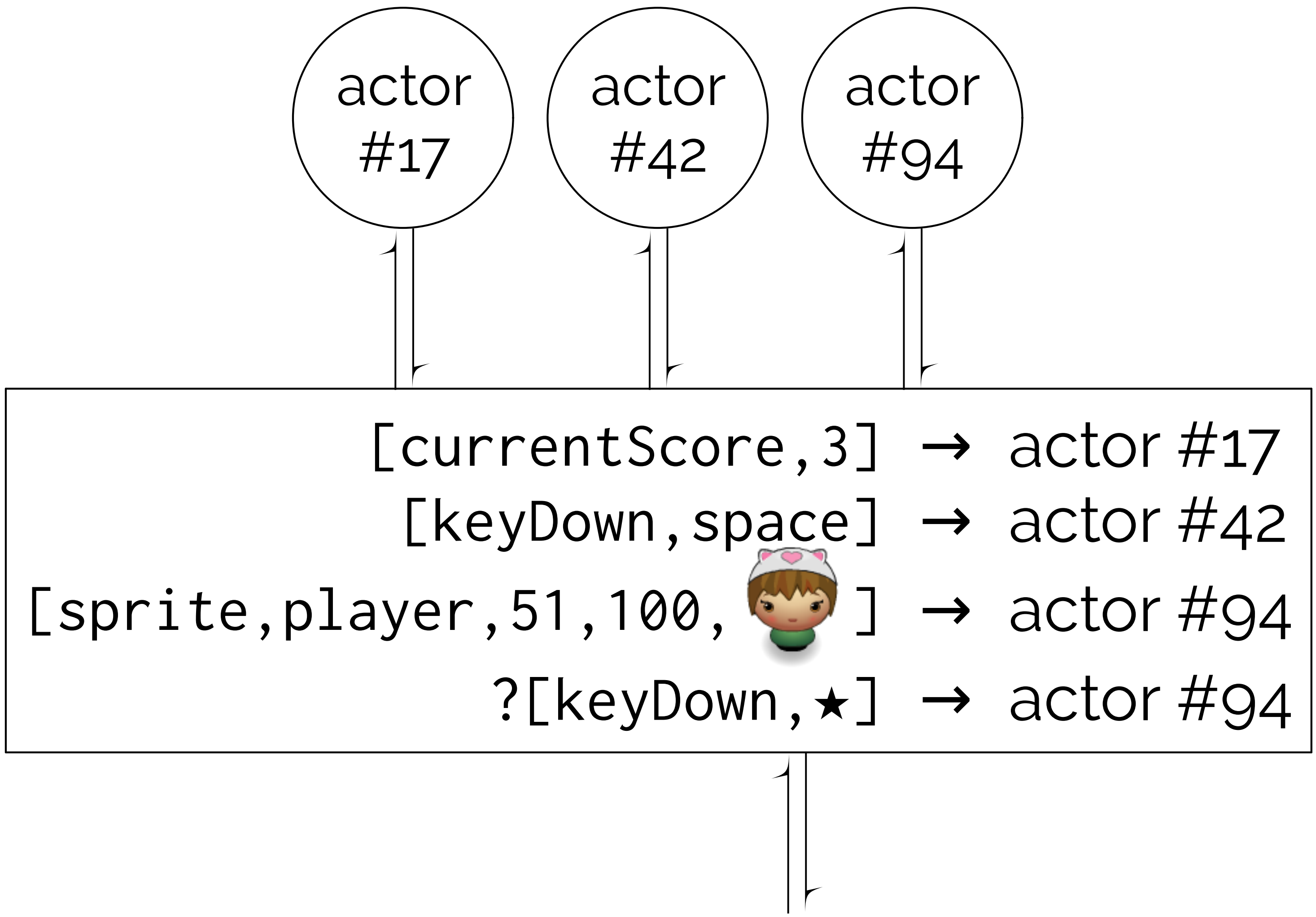
event × state → [action] × state

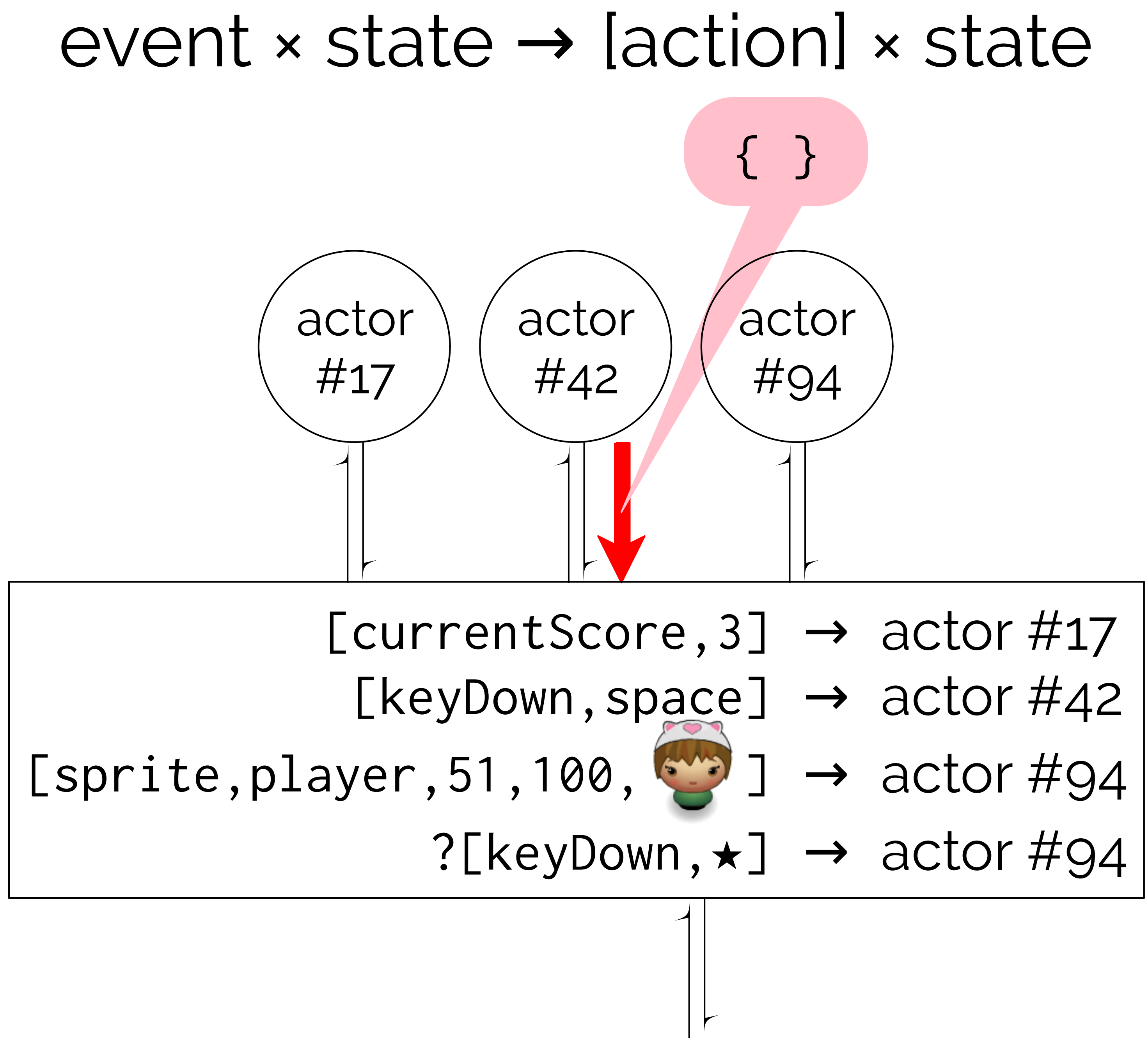


event × state → [action] × state



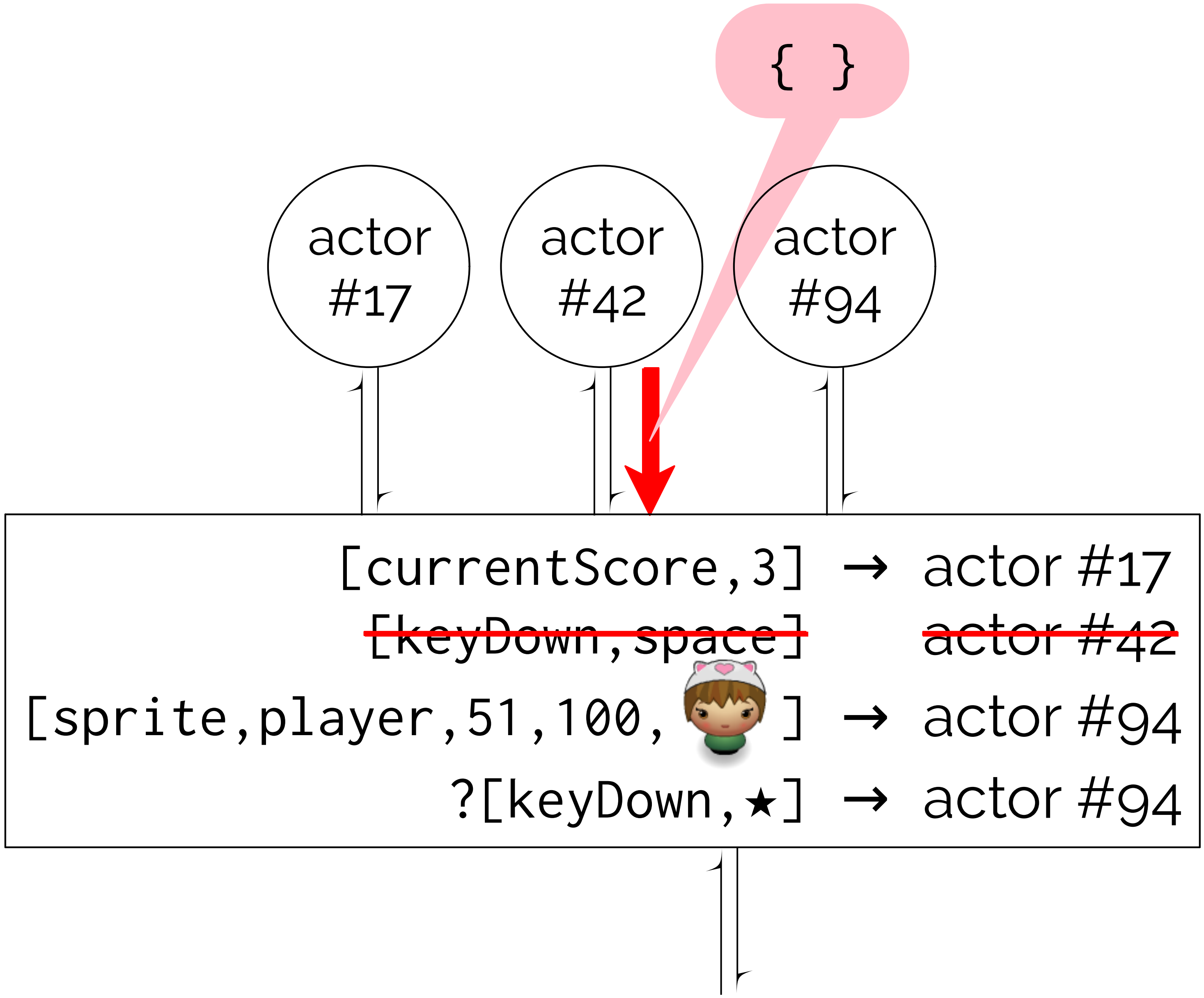
event × state → [action] × state



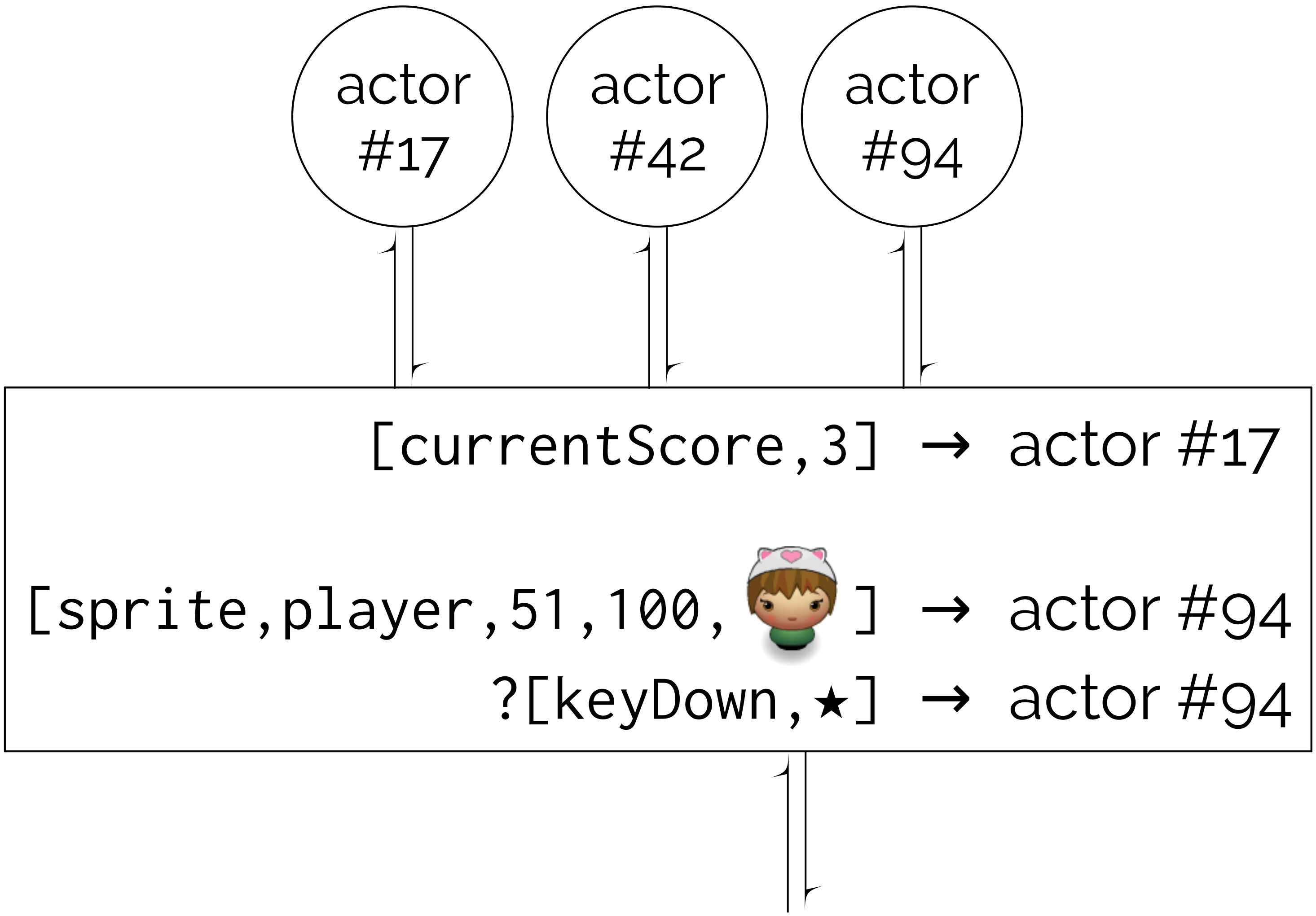


event × state → [action] × state
{ }
actor #17
actor #42
actor #94
[currentScore,3] → actor #17
[keyDown,space] → actor #42
[sprite,player,51,100,] → actor #94
?[keyDown,★] → actor #94

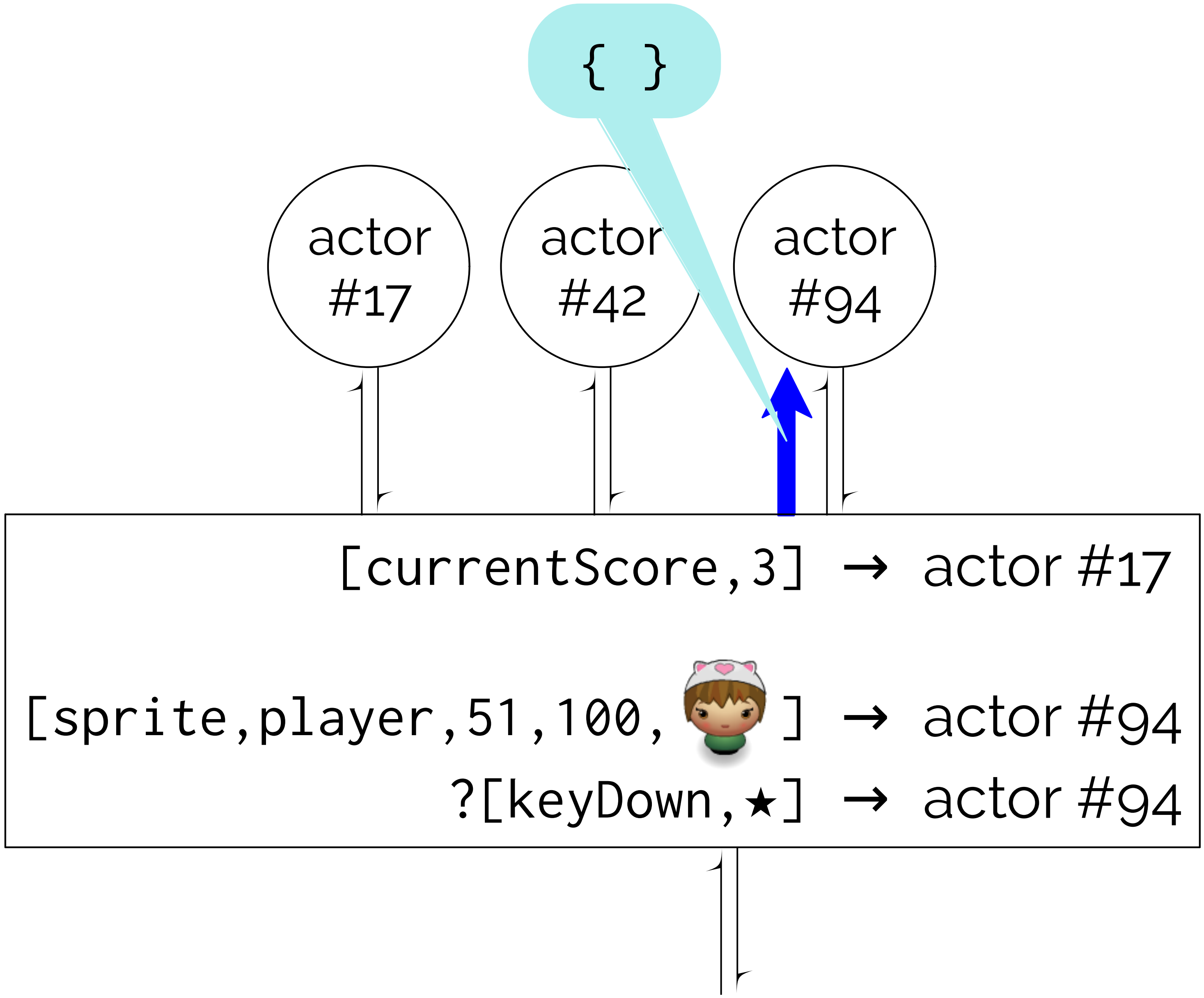
event × state → [action] × state



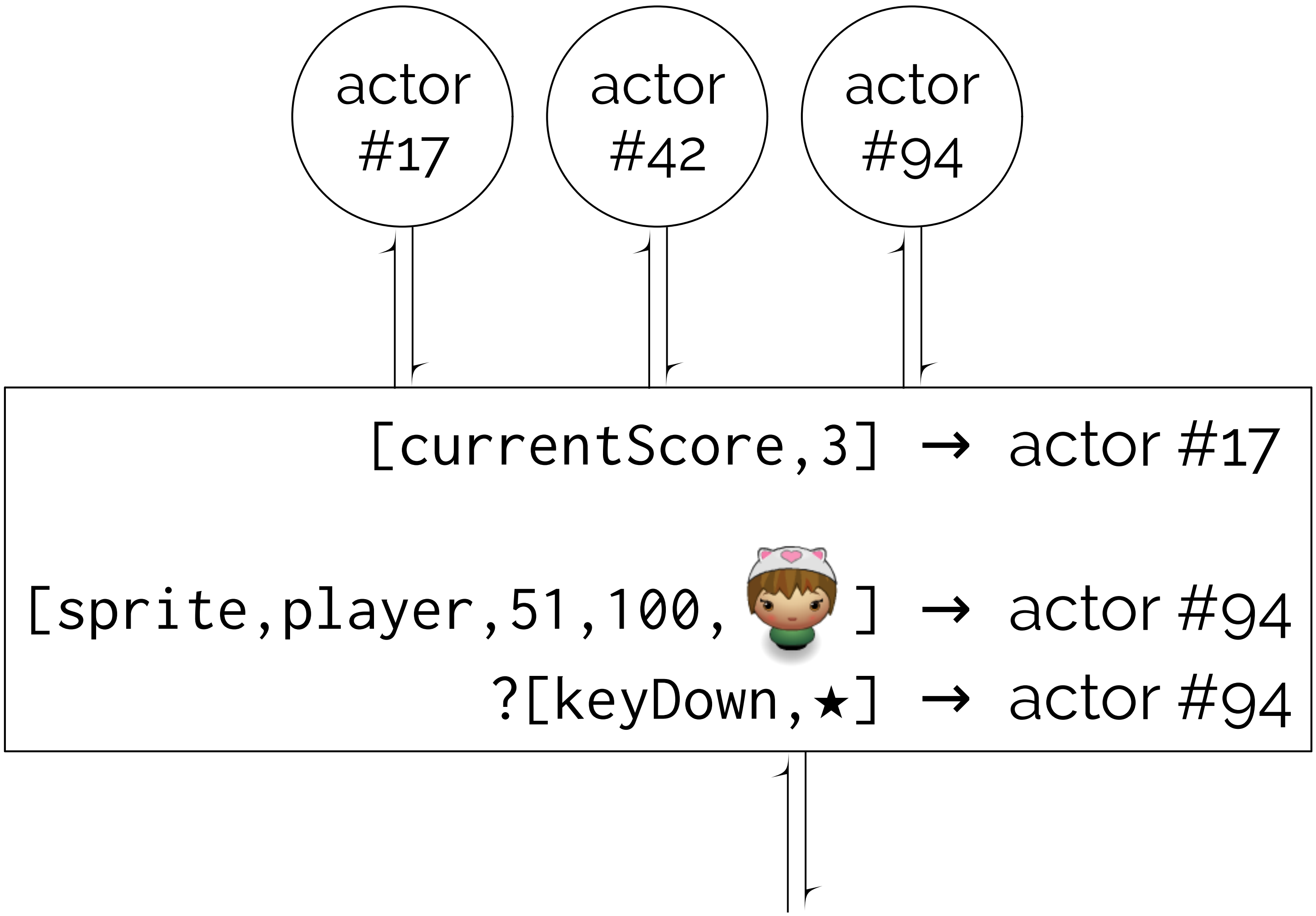
event × state → [action] × state

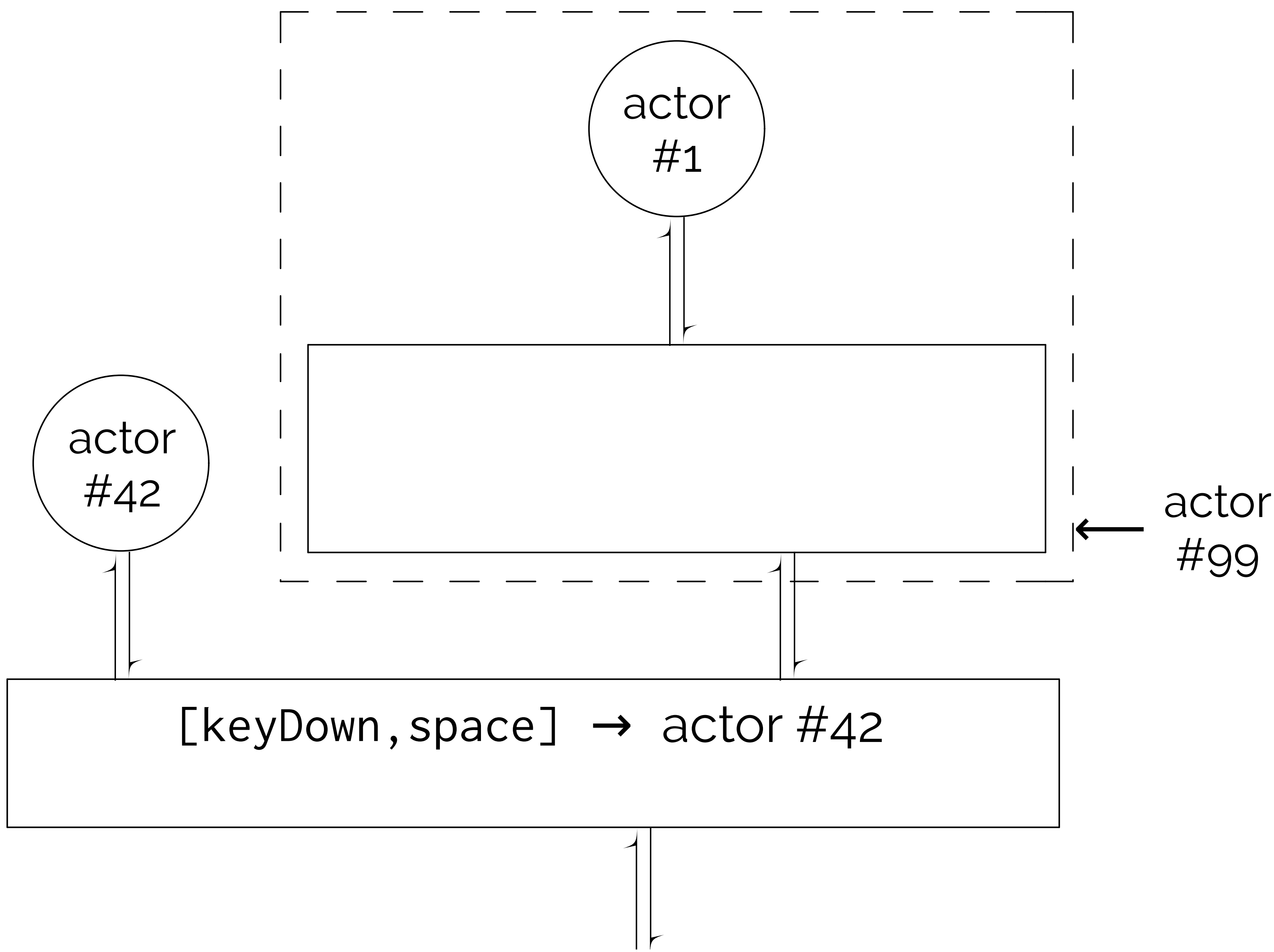


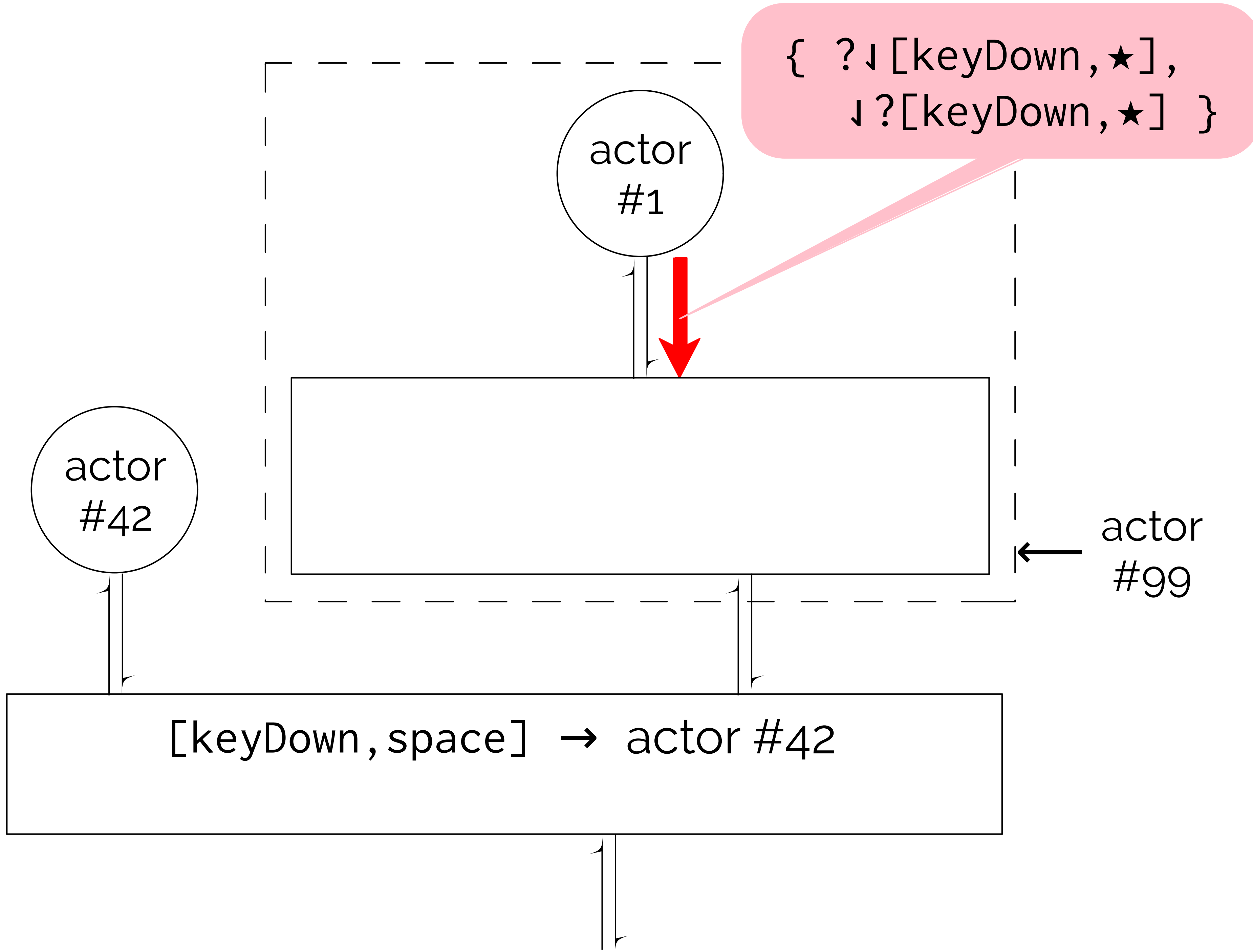
event × state → [action] × state

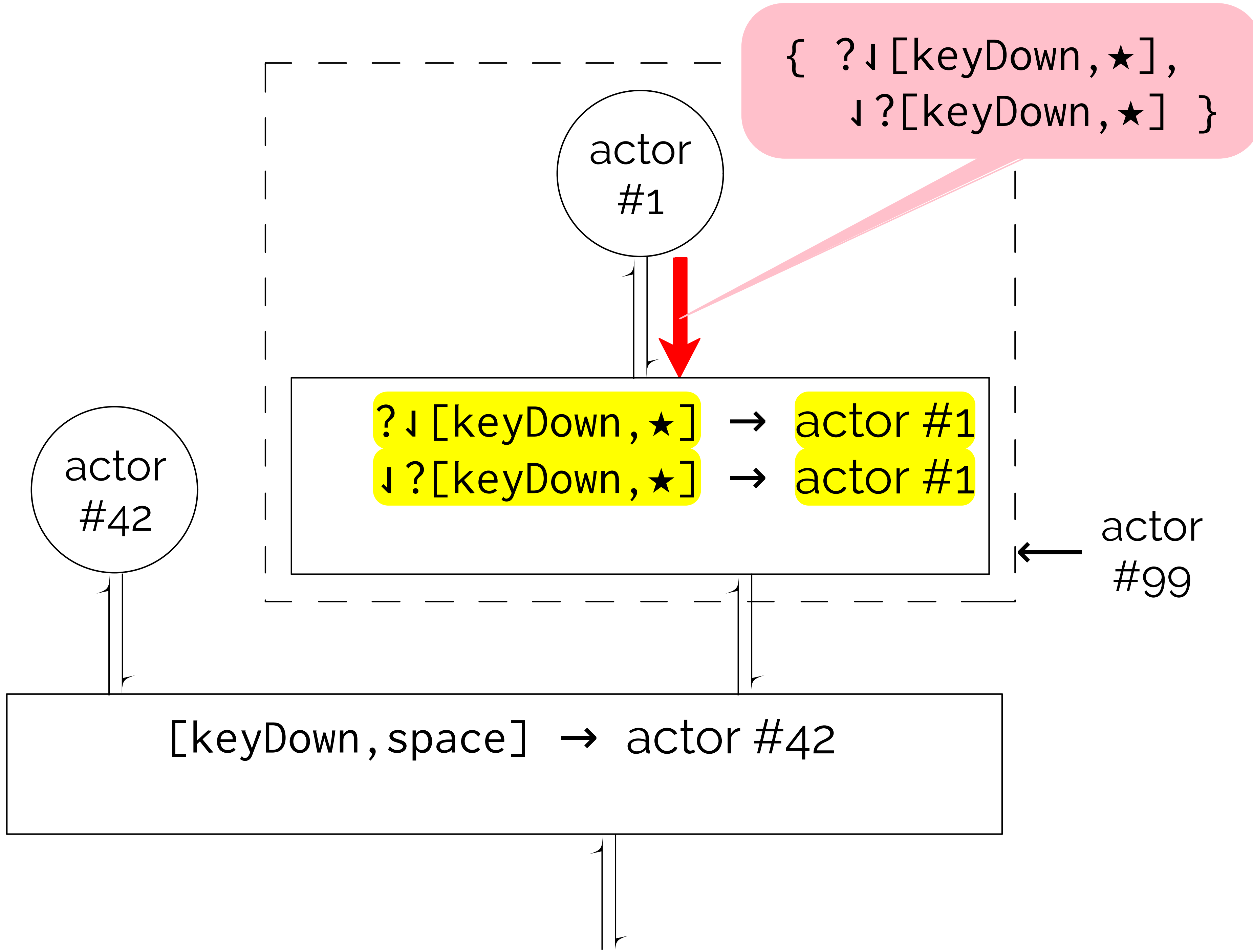


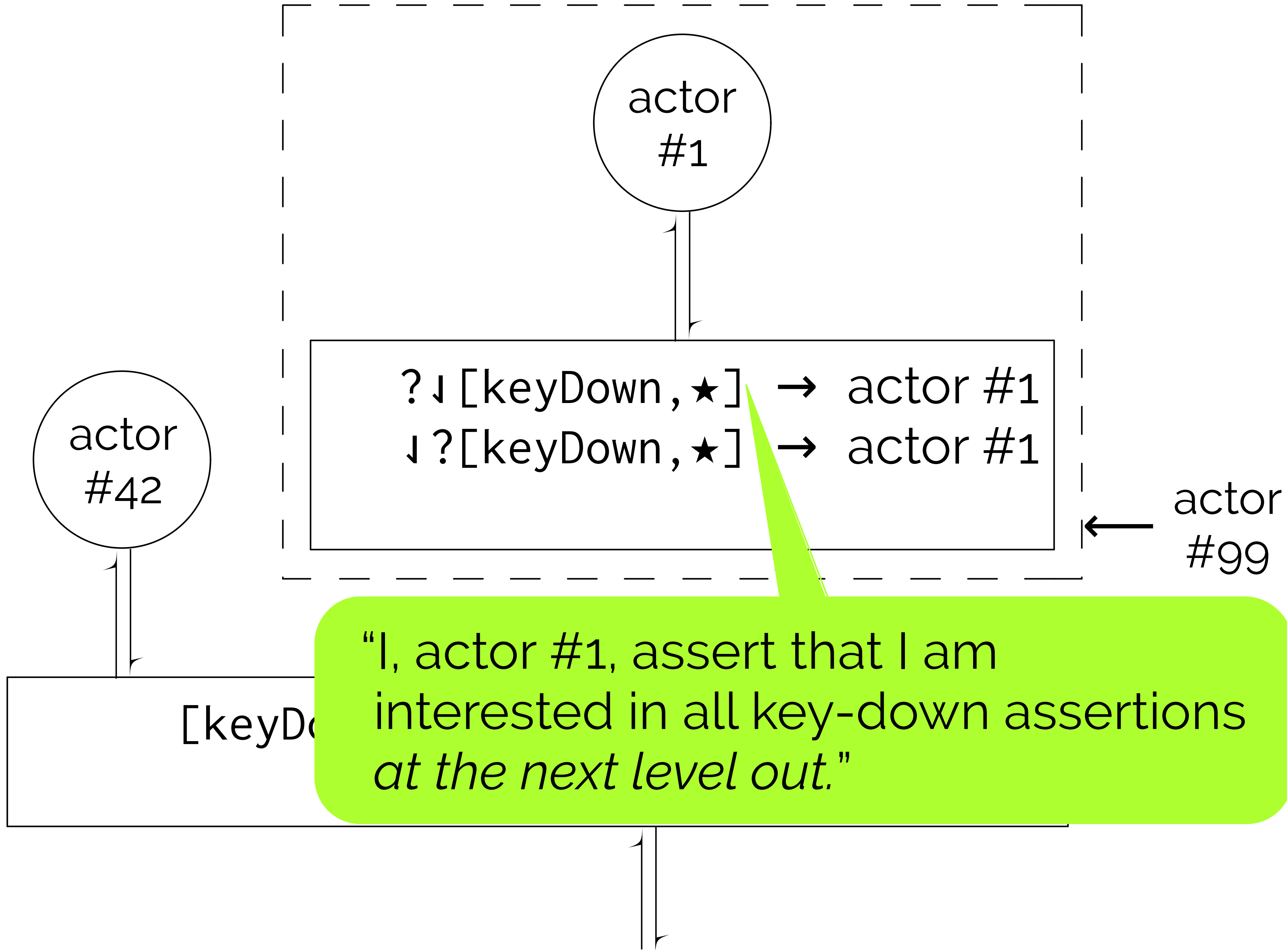
event × state → [action] × state



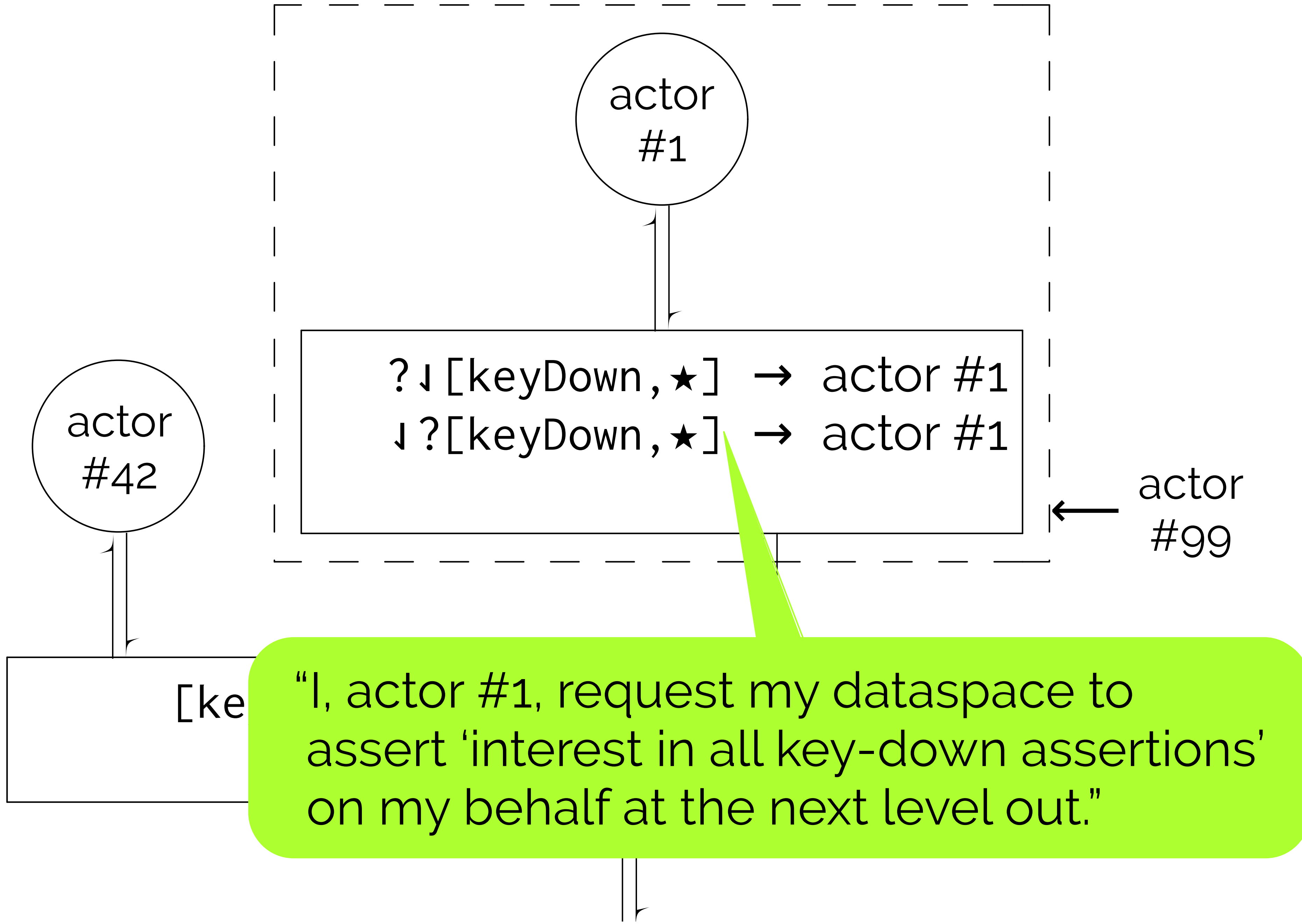

actor
#1
actor
#42
actor
#99
[keyDown,space] → actor #42

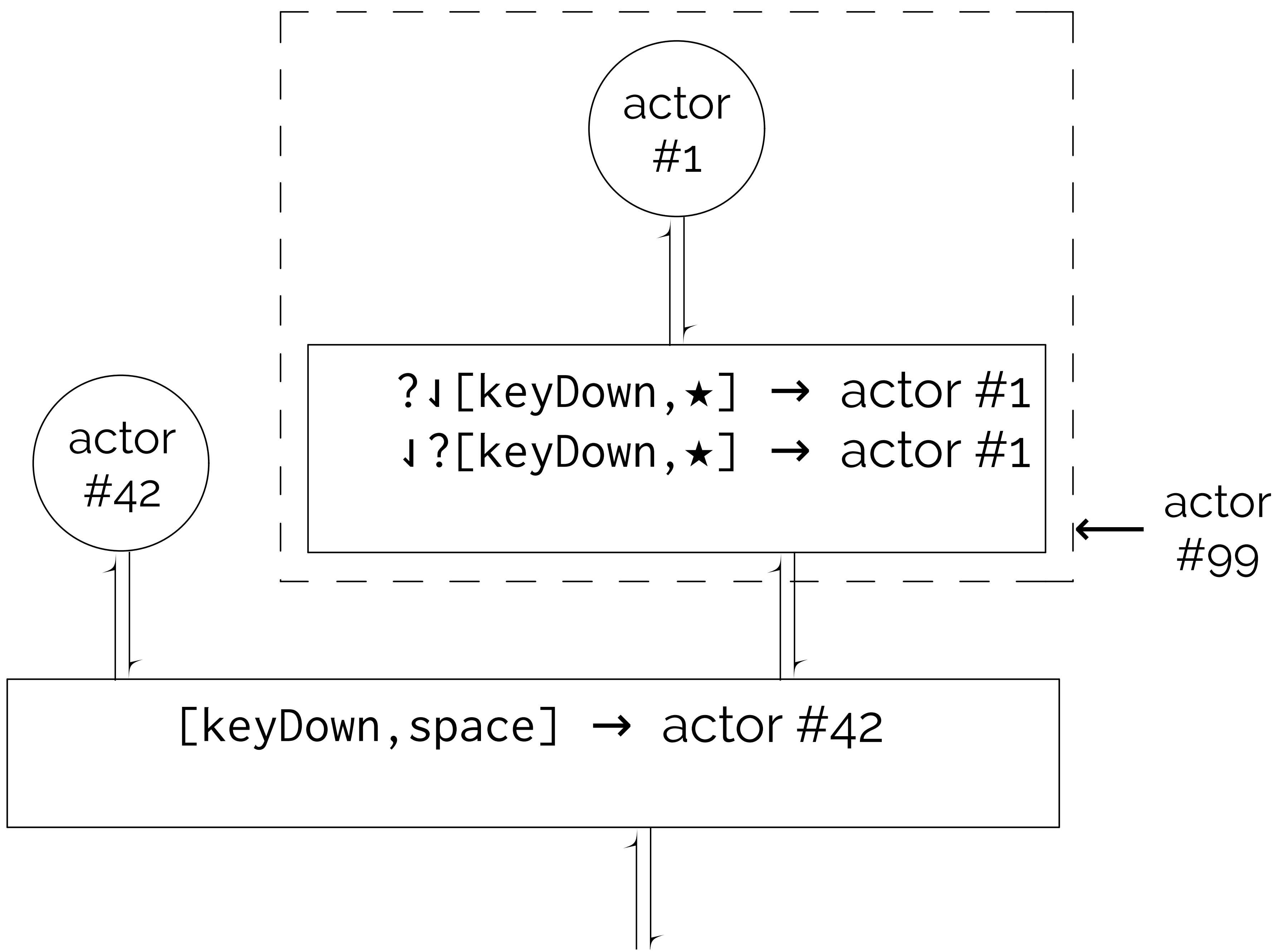

{ ?↓[keyDown,★],
↓?[keyDown,★] }
actor
#1
actor
#42
actor
#99
[keyDown,space] → actor #42

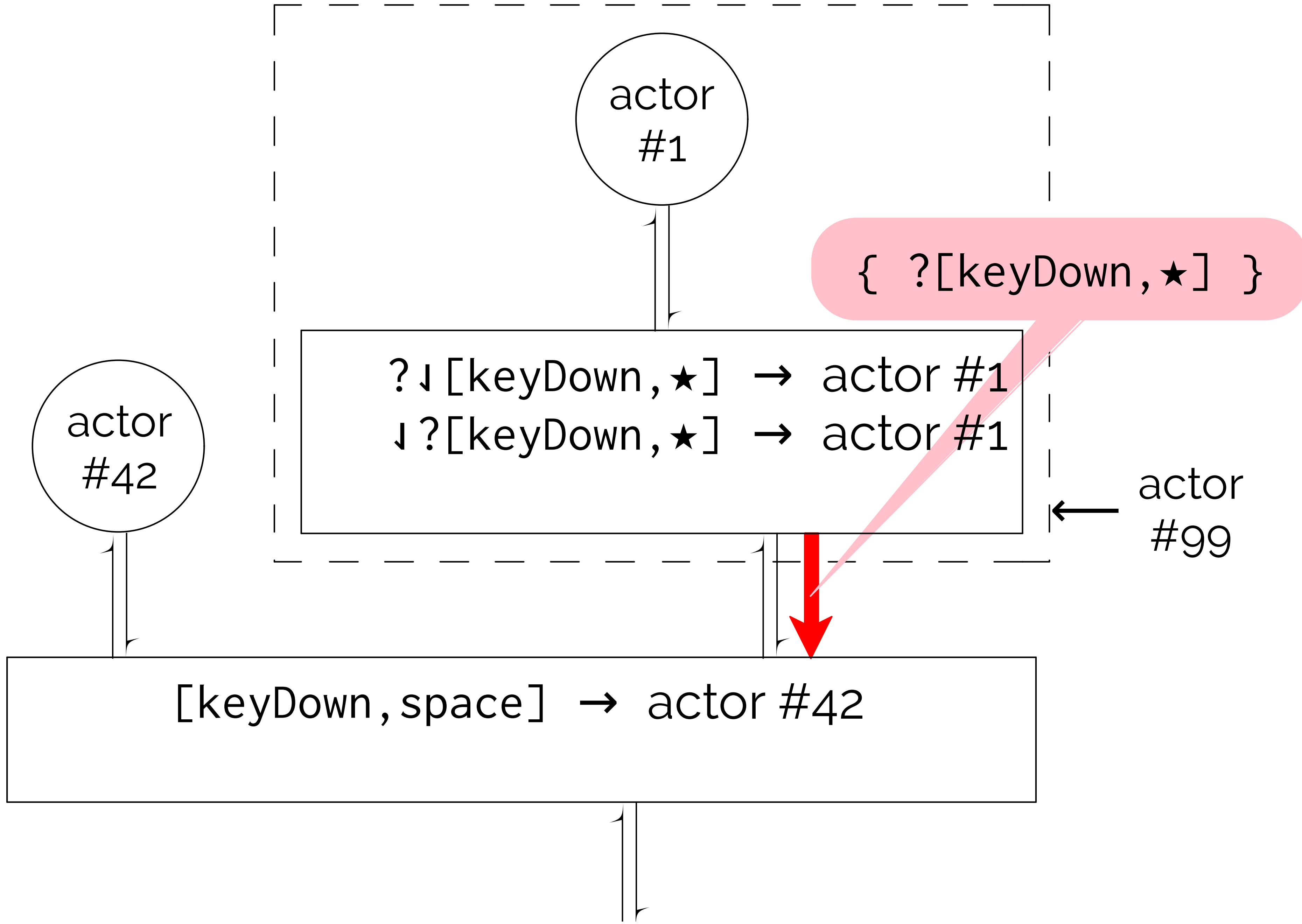

{ ?⇃[keyDown,★],
⇃?[keyDown,★] }
actor
#1
?⇃[keyDown,★] → actor #1
⇃?[keyDown,★] → actor #1
actor
#42
actor
#99
[keyDown,space] → actor #42



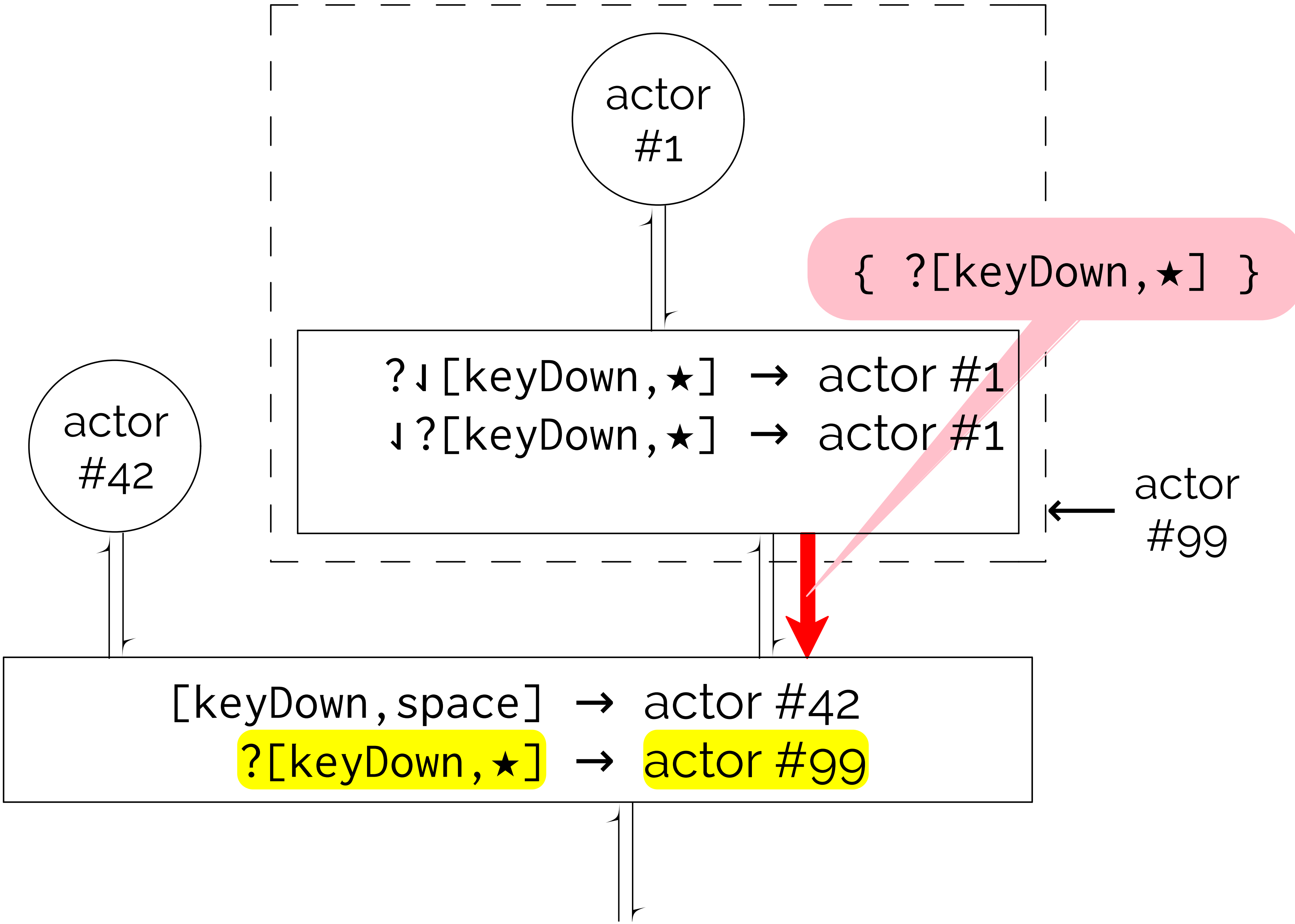
actor #1
?↓[keyDown,★] → actor #1
↓?[keyDown,★] → actor #1
actor #99
actor #42
[keyD
“I, actor #1, assert that I am interested in all key-down assertions *at the next level out.*”

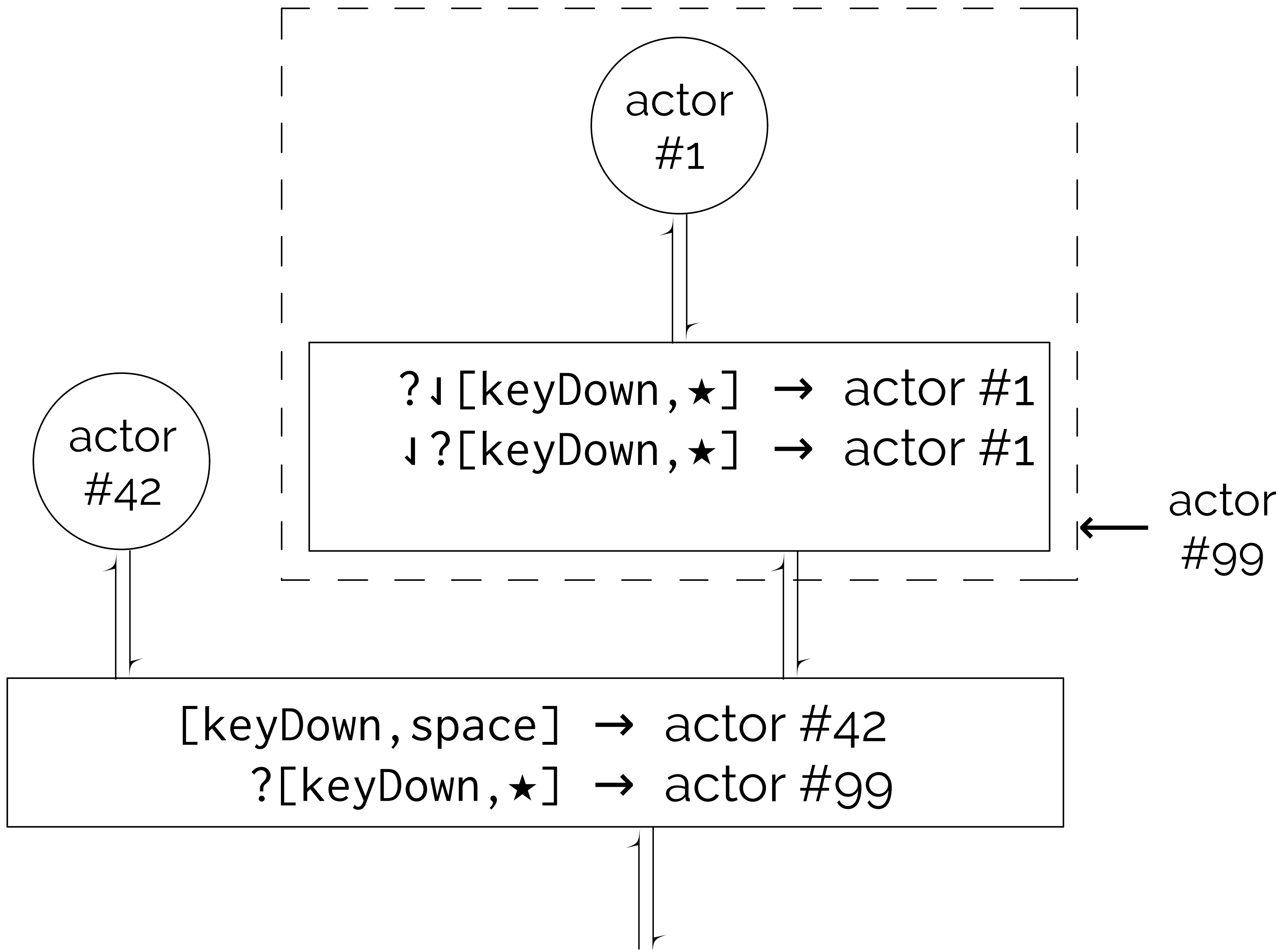

actor #1
?↓[keyDown,★] → actor #1
↓?[keyDown,★] → actor #1
actor #42
actor #99
“I, actor #1, request my dataspace to assert ‘interest in all key-down assertions’ on my behalf at the next level out.”

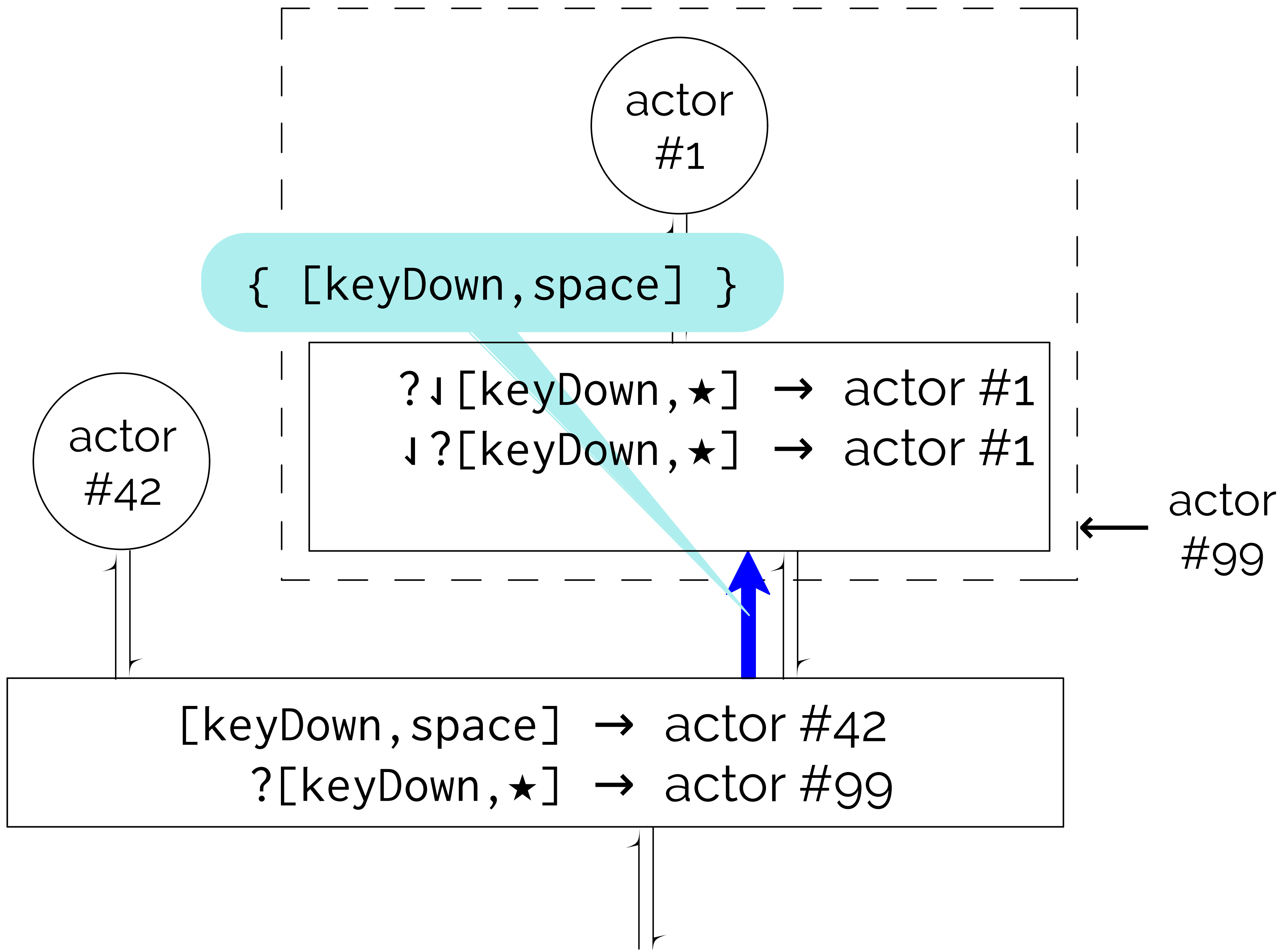

actor #1
?⇃[keyDown,★] → actor #1
⇃?[keyDown,★] → actor #1
actor #42
actor #99
[keyDown,space] → actor #42

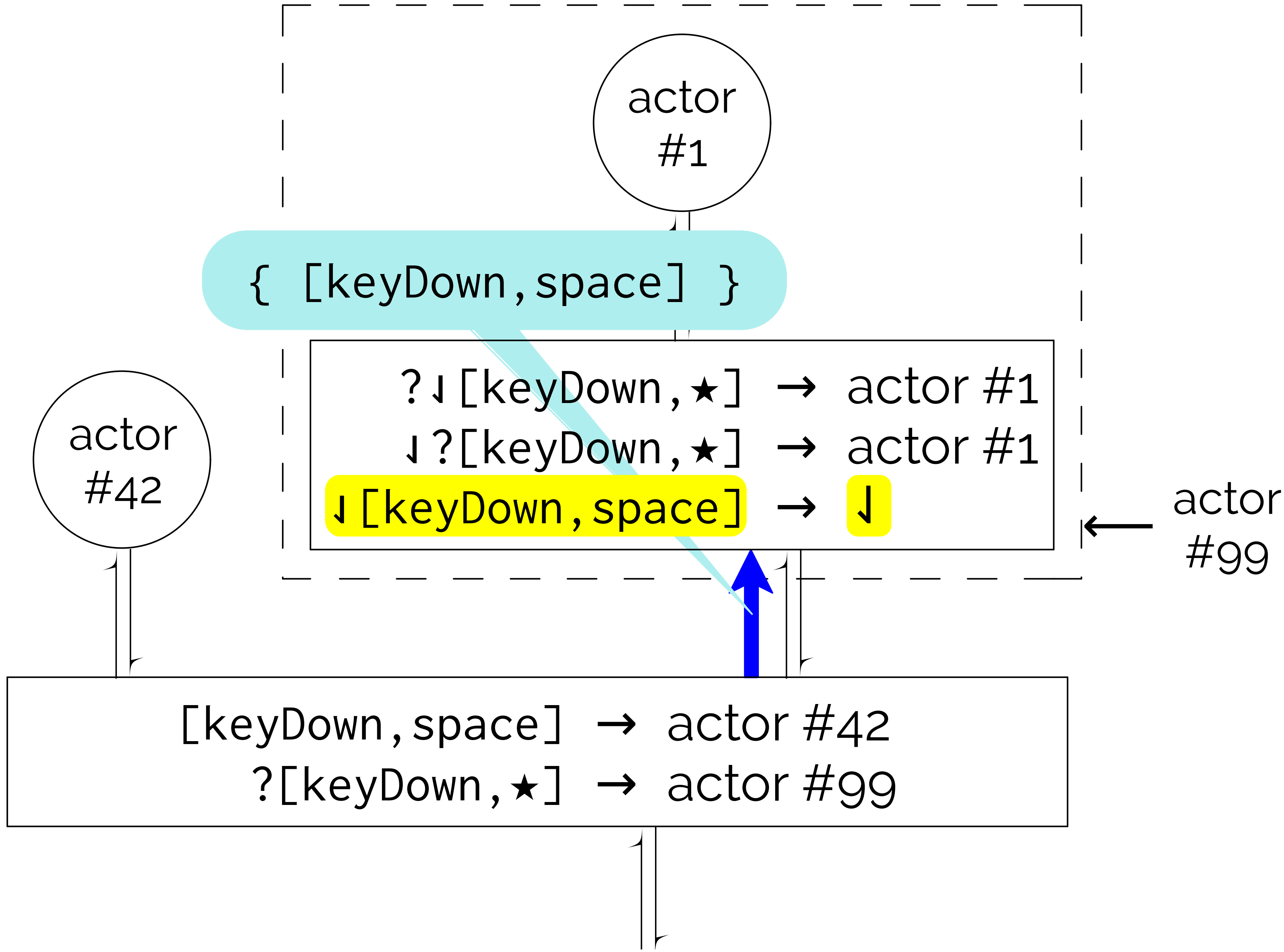


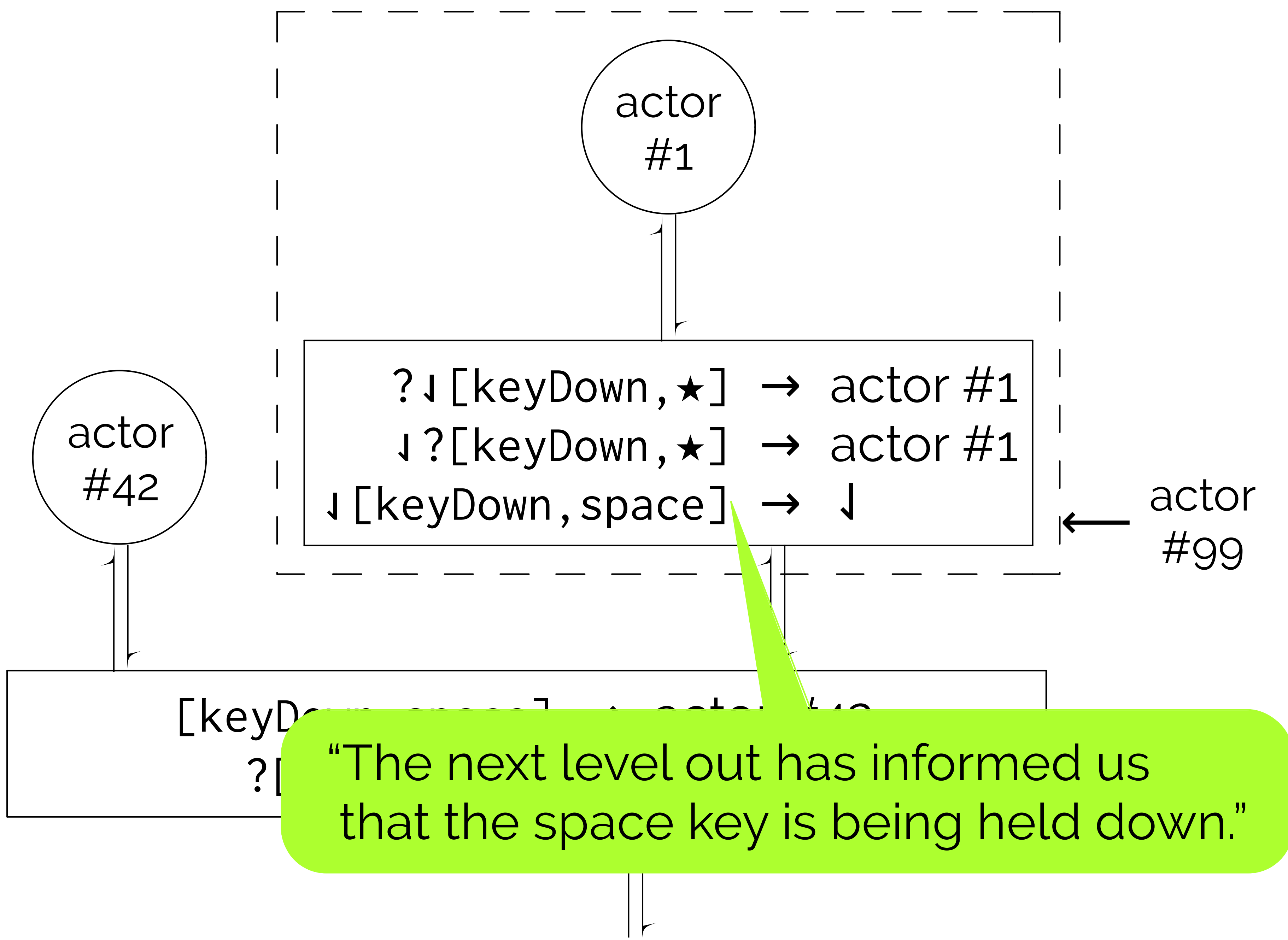
actor #1
{ ?[keyDown,★] }
?↓[keyDown,★] → actor #1
↓?[keyDown,★] → actor #1
actor #42
actor #99
[keyDown,space] → actor #42

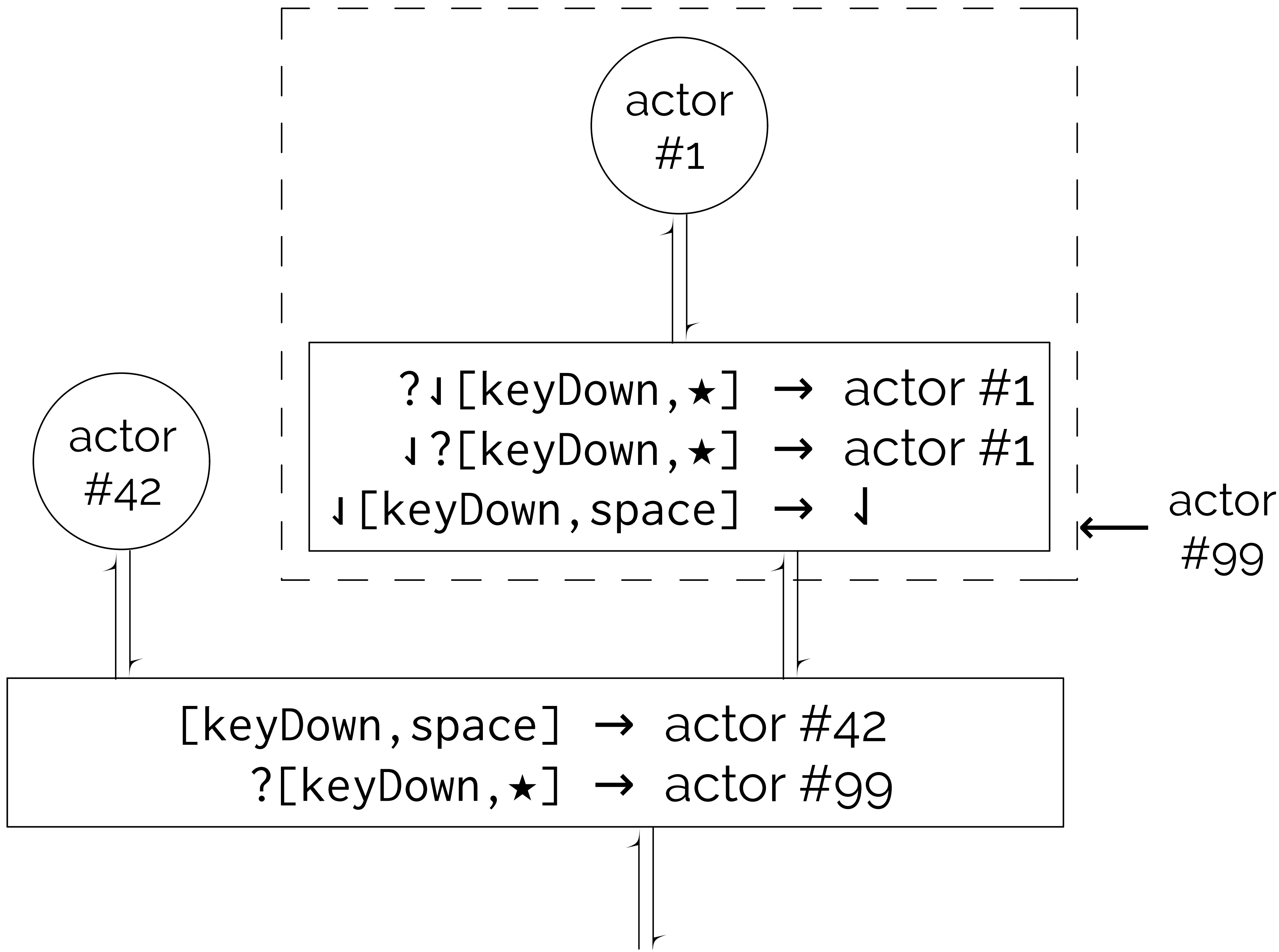

actor #1
{ ?[keyDown,★] }
?↓[keyDown,★] → actor #1
↓?[keyDown,★] → actor #1
actor #42
actor #99
[keyDown,space] → actor #42
?[keyDown,★] → actor #99

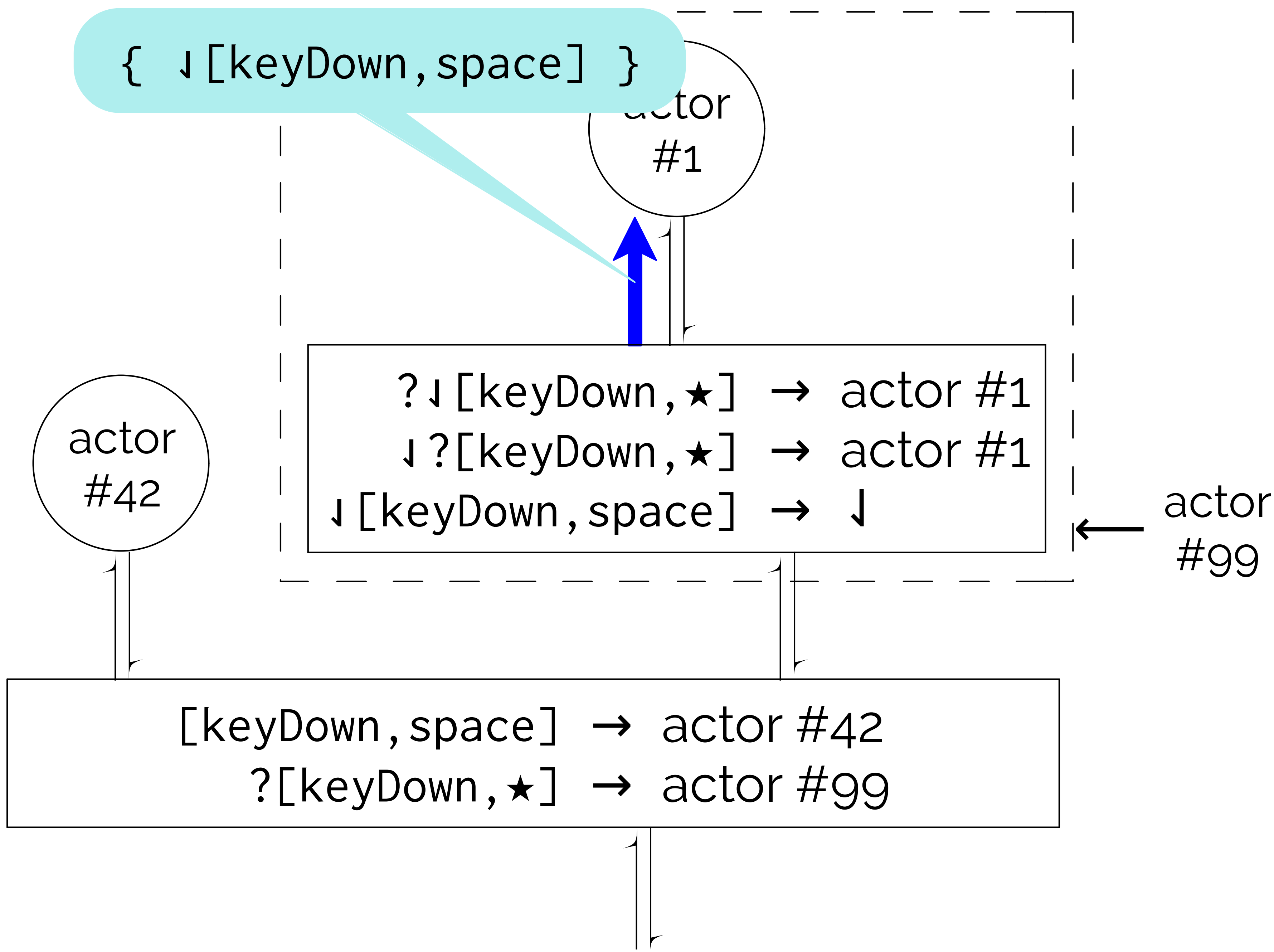

actor #1
?↓[keyDown,★] → actor #1
↓?[keyDown,★] → actor #1
actor #42
actor #99
[keyDown,space] → actor #42
?[keyDown,★] → actor #99

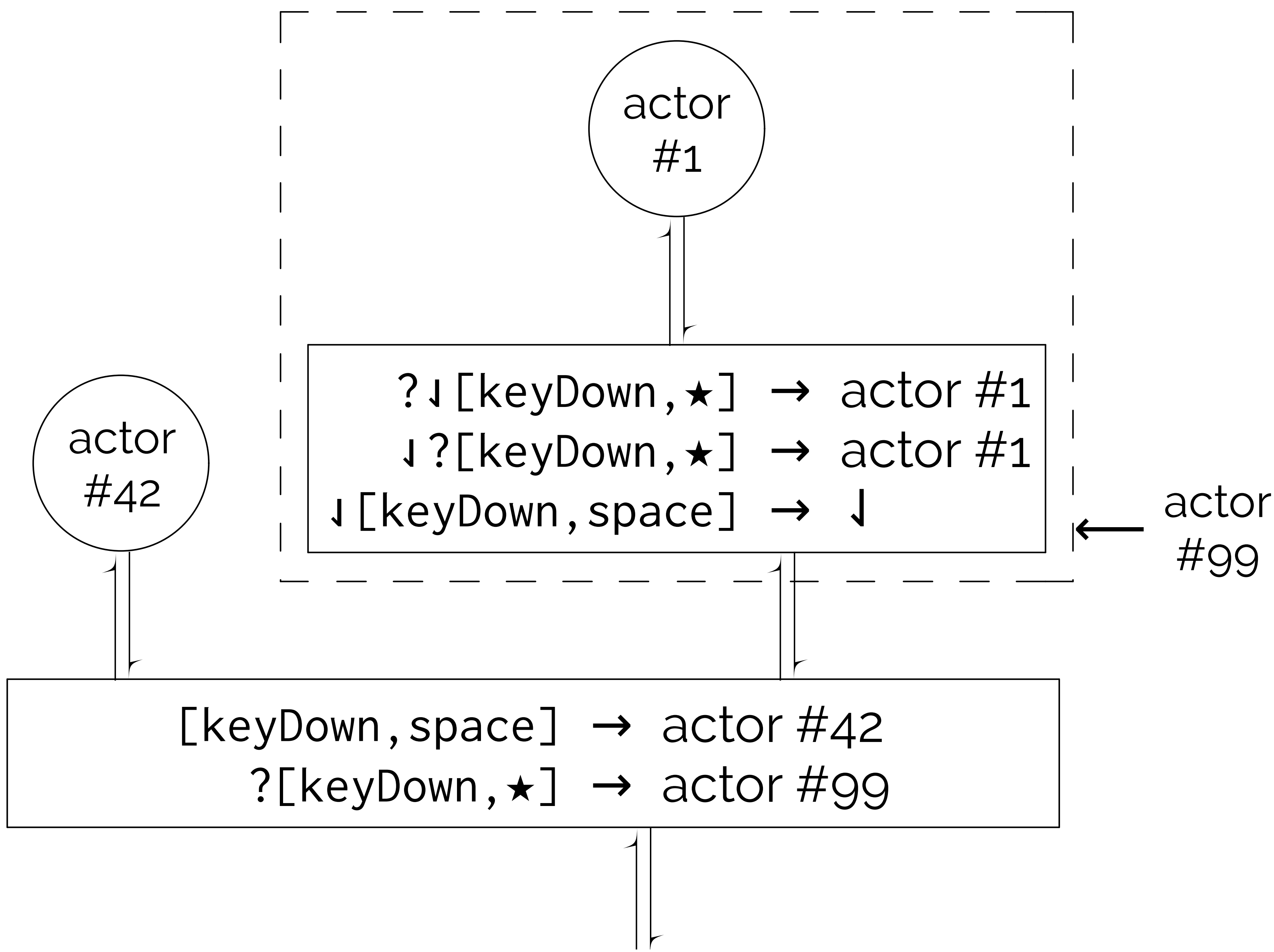

actor #1
{ [keyDown,space] }
?↓[keyDown,★] → actor #1
↓?[keyDown,★] → actor #1
actor #42
actor #99
[keyDown,space] → actor #42
?[keyDown,★] → actor #99


actor #1
{ [keyDown,space] }
?⇃[keyDown,★] → actor #1
⇃?[keyDown,★] → actor #1
⇃[keyDown,space] → ⇃
actor #42
actor #99
[keyDown,space] → actor #42
?[keyDown,★] → actor #99


actor #1
?↓[keyDown,★] → actor #1
↓?[keyDown,★] → actor #1
↓[keyDown,space] → ↓
actor #99
actor #42
[keyD
?[
"The next level out has informed us that the space key is being held down."


actor #1
?⇃[keyDown,★] → actor #1
⇃?[keyDown,★] → actor #1
⇃[keyDown,space] → ⇃
actor #42
actor #99
[keyDown,space] → actor #42
?[keyDown,★] → actor #99


{ ⇃[keyDown,space] }
actor #1
?⇃[keyDown,★] → actor #1
⇃?[keyDown,★] → actor #1
⇃[keyDown,space] → ⇃
actor #99
actor #42
[keyDown,space] → actor #42
?[keyDown,★] → actor #99


actor #1
?⇃[keyDown,★] → actor #1
⇃?[keyDown,★] → actor #1
⇃[keyDown,space] → ⇃
actor #42
actor #99
[keyDown,space] → actor #42
?[keyDown,★] → actor #99

Messages are transient assertions

`< [incrementScoreBy,3] >`

~

`{ [incrementScoreBy,3] }`

followed by

`{ }`

General challenges of interactivity

- Mapping events to components
- Building a shared understanding
- Partial failure
- Scoped conversational state

General challenges of interactivity

- Mapping events to components
- Building a shared understanding
- Partial failure
- Scoped conversational state

Mapping events to components: OO


```java
public class ControllerListener {
    private Level currentLevel;
    private Player player;
    ... /* repeat for each target! */

    public void handleControllerEvent(ControllerEvent e) {
        switch (e.getCode()) {
          case ControllerEvent.VK_START:
              currentLevel.abandon(); return;
          case ControllerEvent.VK_LEFT:
              player.moveLeft(); return;
          ... /* repeat for each key! */
        }
    }

    public void changeLevel(Level newLevel) {
        this.currentLevel = newLevel;
    }
}
```

Mapping events to components: Actors

```erlang
-module(eventmapping).
-behaviour(gen_server).

-record(state, {current_level, player}).

init([PlayerPid]) ->
    ok = controller:subscribe(self()),
    {ok, #state{current_level = undefined,
                player = PlayerPid}}.

handle_cast({controller_event, start}, State) ->
    gen_server:cast(State#state.current_level, abandon),
    {noreply, State};

handle_cast({controller_event, left}, State) ->
    gen_server:cast(State#state.player, move_left),
    {noreply, State}.

handle_call({change_level, LevelPid}, _From, State) ->
    {reply, ok, State#state{current_level = LevelPid}}.
```

Mapping events to components: Syndicate

```
;; Level actor:
(actor
 (until (message (controller-event 'start))
   ;; ... event handlers ...
   ))

;; Player actor:
(actor
 (until (message 'kill-player)
        #:collect [(state (initial-player-state))]
   (on (message (controller-event 'left))
       (update-position state -1 0))
   ;; ... other event handlers ...
   ))
```

General challenges of interactivity

- ✓ Mapping events to components
- Building a shared understanding
- Partial failure
- Scoped conversational state

Building a shared understanding: OO

```
public class GamePieces {
    private Set<GamePiece> pieces = new HashSet<>();


    public void addGamePiece(GamePiece p) {
        pieces.add(p);


    }

    public void removeGamePiece(GamePiece p) {
        pieces.remove(p);


    }






}
```

Building a shared understanding: OO

```java
public class GamePieces {
    private Set<GamePiece> pieces = new HashSet<>();
    private Set<GamePieceListener> subscribers = new HashSet<>();

    public void addGamePiece(GamePiece p) {
        pieces.add(p);
        for (GamePieceListener l : subscribers)
            l.gamePieceAdded(p);
    }

    public void removeGamePiece(GamePiece p) {
        pieces.remove(p);
        for (GamePieceListener l : subscribers)
            l.gamePieceRemoved(p);
    }

    public void subscribe(GamePieceListener l) {
        subscribers.add(l);


    }
}

public interface GamePieceListener {
    void gamePieceAdded(GamePiece p);
    void gamePieceRemoved(GamePiece p);
}
```

Building a shared understanding: OO

```java
public class GamePieces {
    private Set<GamePiece> pieces = new HashSet<>();
    private Set<GamePieceListener> subscribers = new HashSet<>();

    public void addGamePiece(GamePiece p) {
        pieces.add(p);
        for (GamePieceListener l : subscribers)
            l.gamePieceAdded(p);
    }

    public void removeGamePiece(GamePiece p) {
        pieces.remove(p);
        for (GamePieceListener l : subscribers)
            l.gamePieceRemoved(p);
    }

    public void subscribe(GamePieceListener l) {
        subscribers.add(l);
        for (GamePiece p : pieces)
            l.gamePieceAdded(p);
    }
}

public interface GamePieceListener {
    void gamePieceAdded(GamePiece p);
    void gamePieceRemoved(GamePiece p);
}
```

Building a shared understanding: OO

```java
public class GamePieces {
    private Set<GamePiece> pieces = new HashSet<>();
    private Set<GamePieceListener> subscribers = new HashSet<>();

    public void addGamePiece(GamePiece p) {
        pieces.add(p);
        for (GamePieceListener l : new HashSet<GamePieceListener>(subscribers))
            l.gamePieceAdded(p);
    }

    public void removeGamePiece(GamePiece p) {
        pieces.remove(p);
        for (GamePieceListener l : new HashSet<GamePieceListener>(subscribers))
            l.gamePieceRemoved(p);
    }

    public void subscribe(GamePieceListener l) {
        subscribers.add(l);
        for (GamePiece p : new HashSet<GamePiece>(pieces))
            l.gamePieceAdded(p);
    }
}

public interface GamePieceListener {
    void gamePieceAdded(GamePiece p);
    void gamePieceRemoved(GamePiece p);
}
```

Building a shared understanding: Actors

```erlang
-record(state, [pieces, subscribers]).

handle_call({add_piece, P}, _From, State) ->
    Subscribers = sets:to_list(State#state.subscribers),
    [ gen_server:cast(S, {add_piece, P}) || S <- Subscribers],
    NewState = State#state{pieces = sets:add_element(P, State#state.pieces)},
    {reply, ok, NewState};

handle_call({del_piece, P}, _From, State) ->
    Subscribers = sets:to_list(State#state.subscribers),
    [ gen_server:cast(S, {del_piece, P}) || S <- Subscribers],
    NewState = State#state{pieces = sets:del_element(P, State#state.pieces)},
    {reply, ok, NewState};

handle_call({add_sub, S}, _From, State) ->
    Pieces = sets:to_list(State#state.pieces),
    [ gen_server:cast(S, {add_piece, P}) || P <- Pieces],
    NewState = State#state{subscribers =
                                   sets:add_element(S, State#state.subscribers)},
    {reply, ok, NewState}.
```

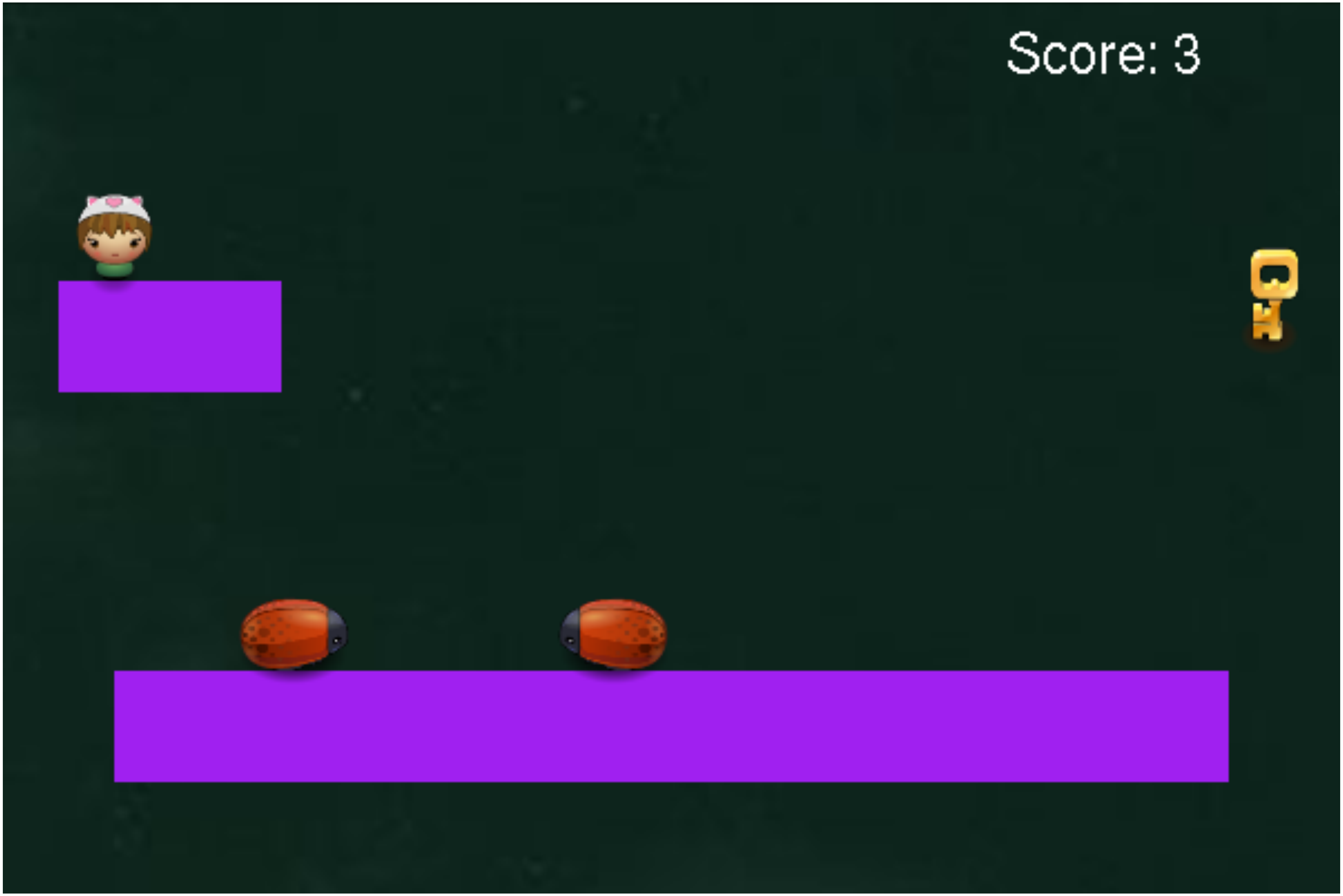
Building a shared understanding: Syndicate

```
;; Each game piece:
(actor
 (forever #:collect [(state (initial-game-piece-state))]
  (assert (game-piece-state state))
  ;; ... other event handlers change `state`,
  ;;      and `assert` automatically re-publishes it
  ))

;; Each subscribing party:
(actor
 (forever
  (on (retracted (game-piece-state $old-state))
      ;; ... remove old state from records ...
      )
  (on (asserted (game-piece-state $new-state))
      ;; ... add new state to records ...
      )))
```

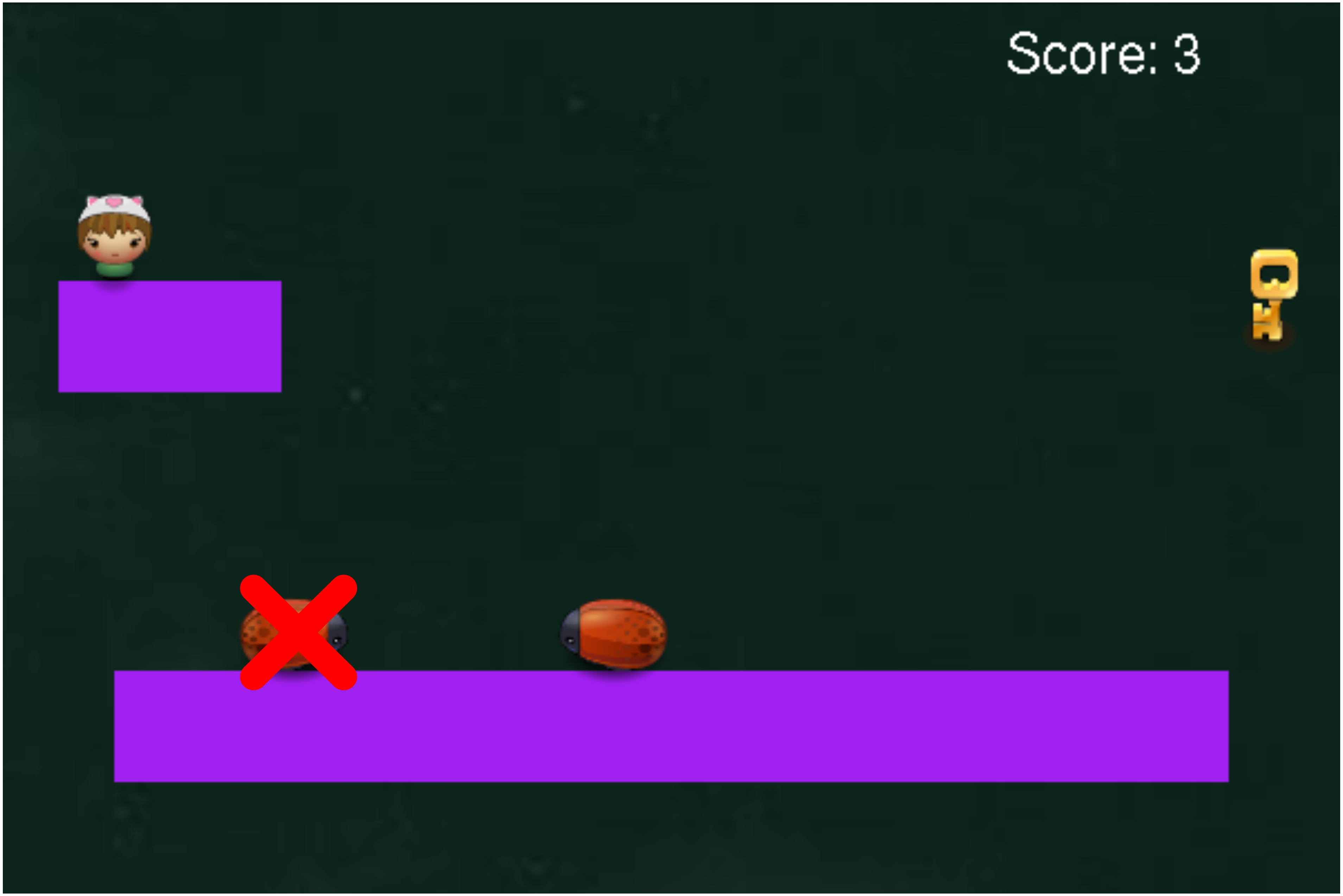
General challenges of interactivity

- ✓ Mapping events to components
- ✓ Building a shared understanding
- Partial failure
- Scoped conversational state

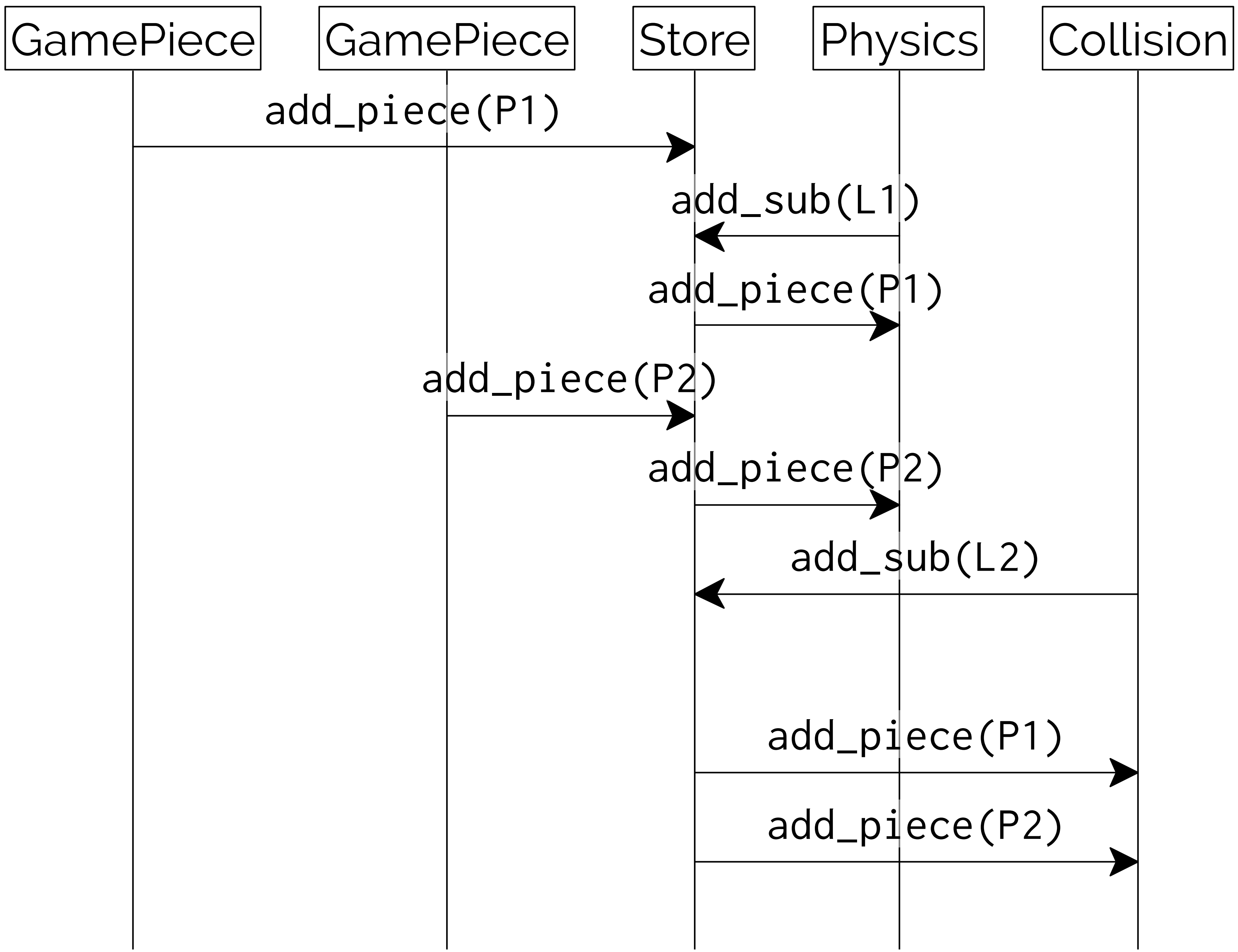
Partial Failure: OO & Actors



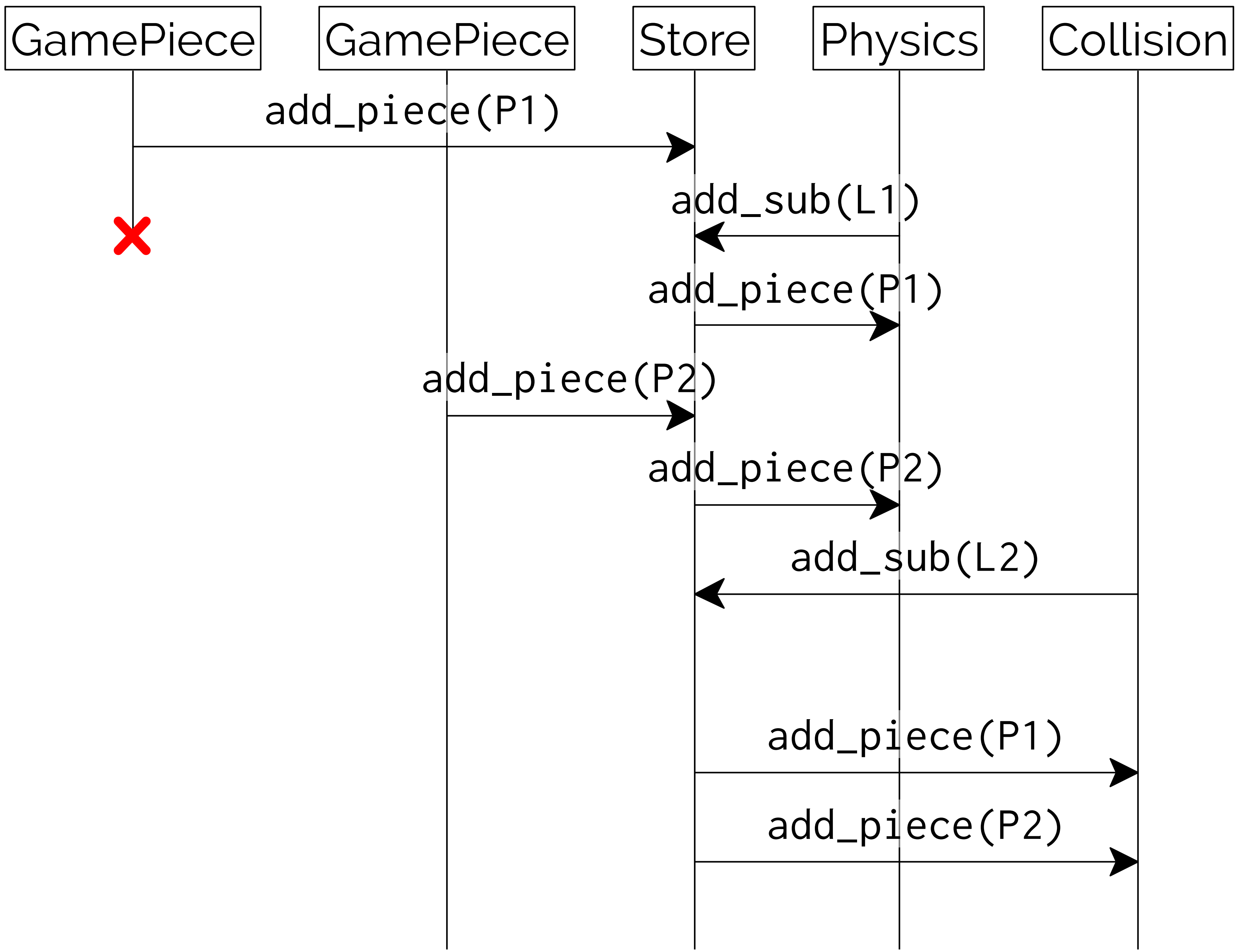
Partial Failure: OO & Actors



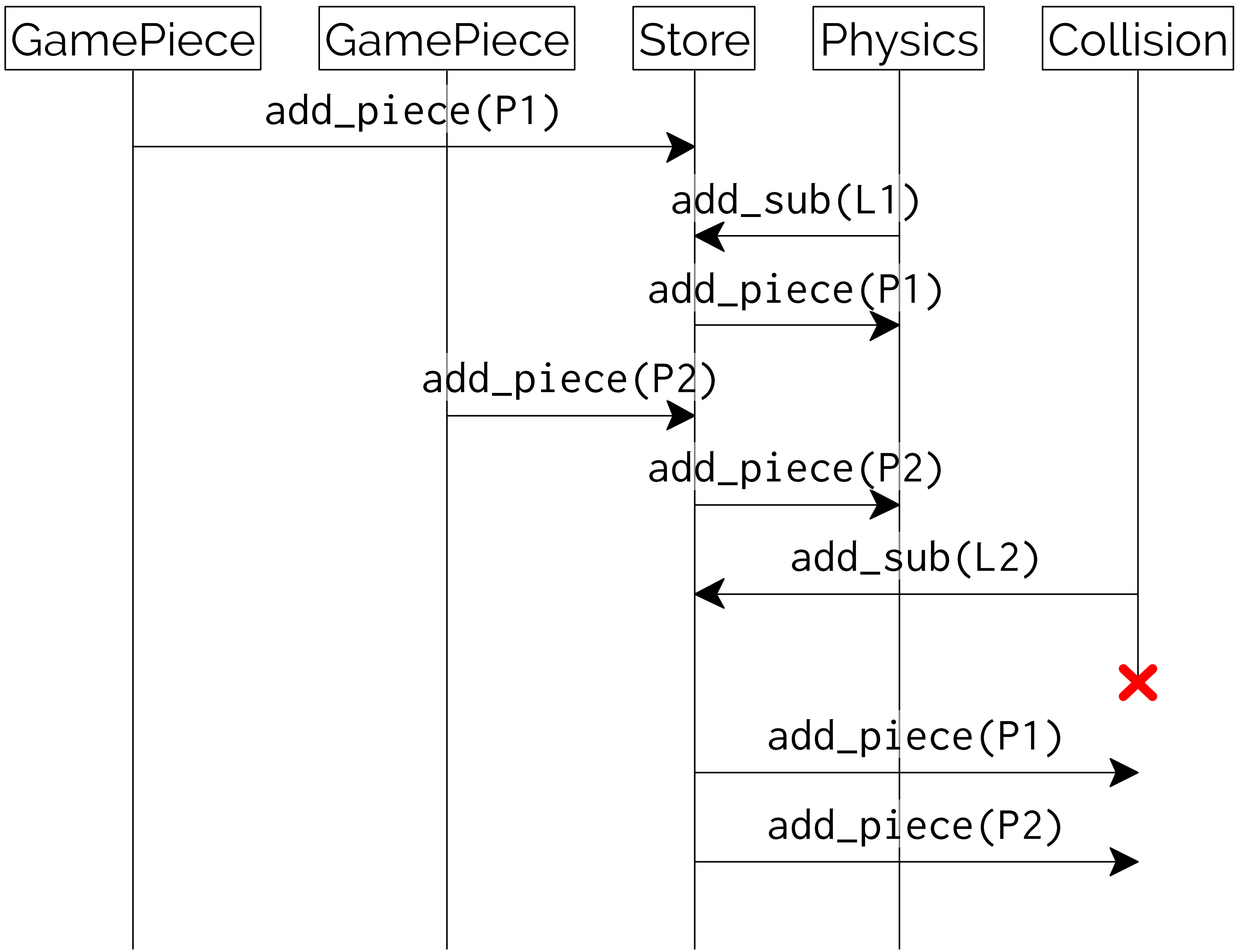
Partial Failure: OO & Actors



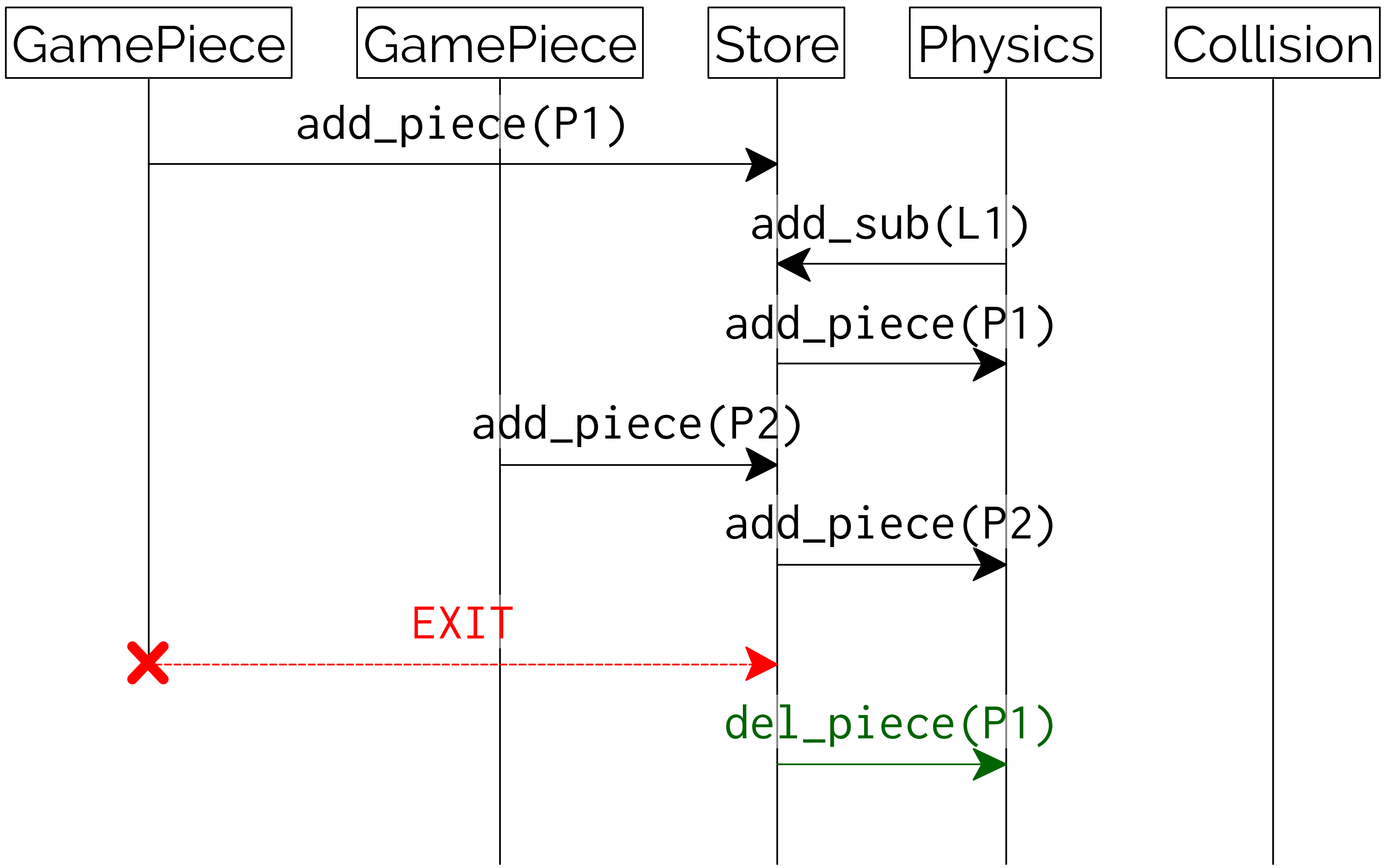
Partial Failure: OO & Actors



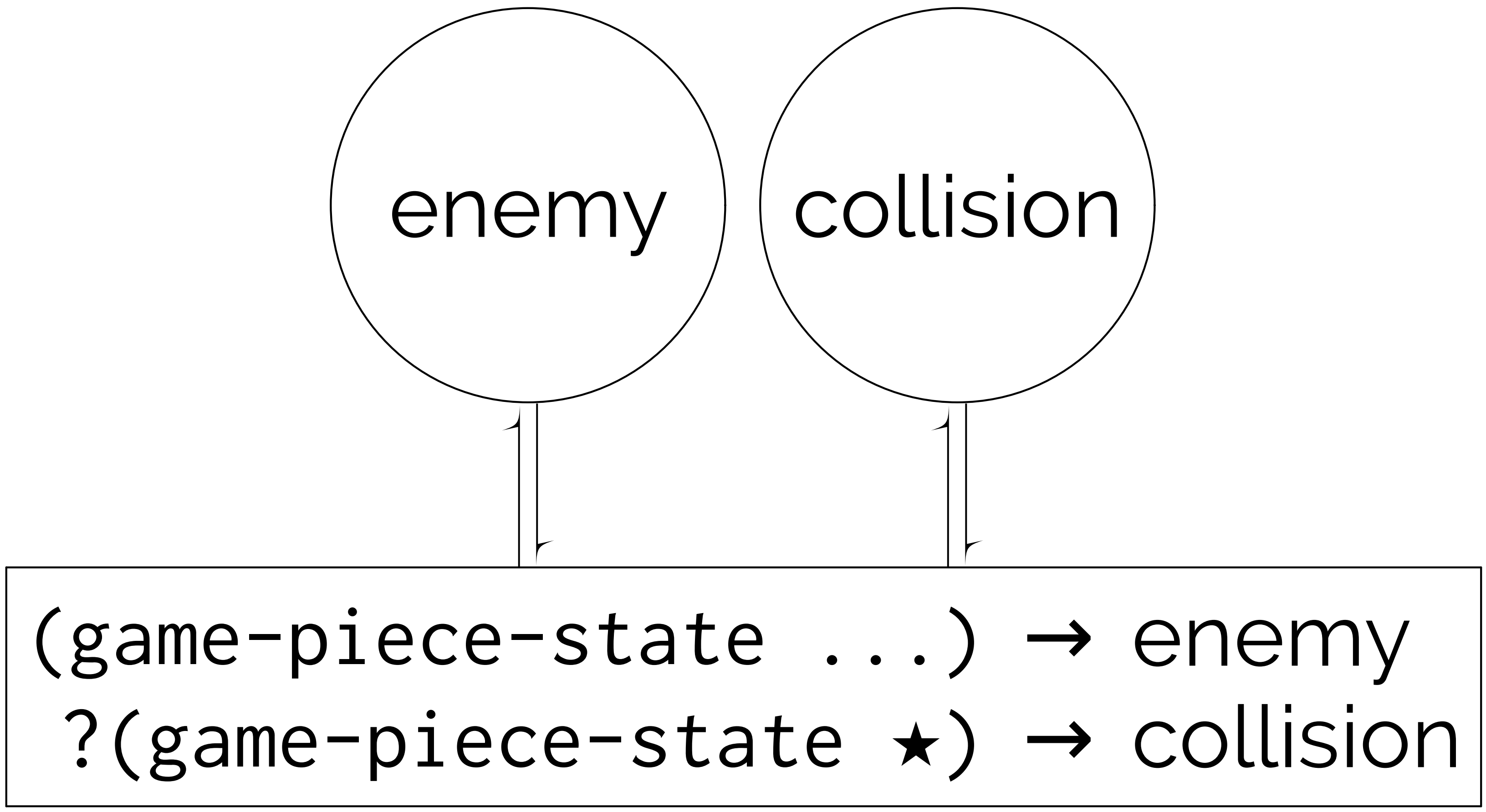
Partial Failure: OO & Actors



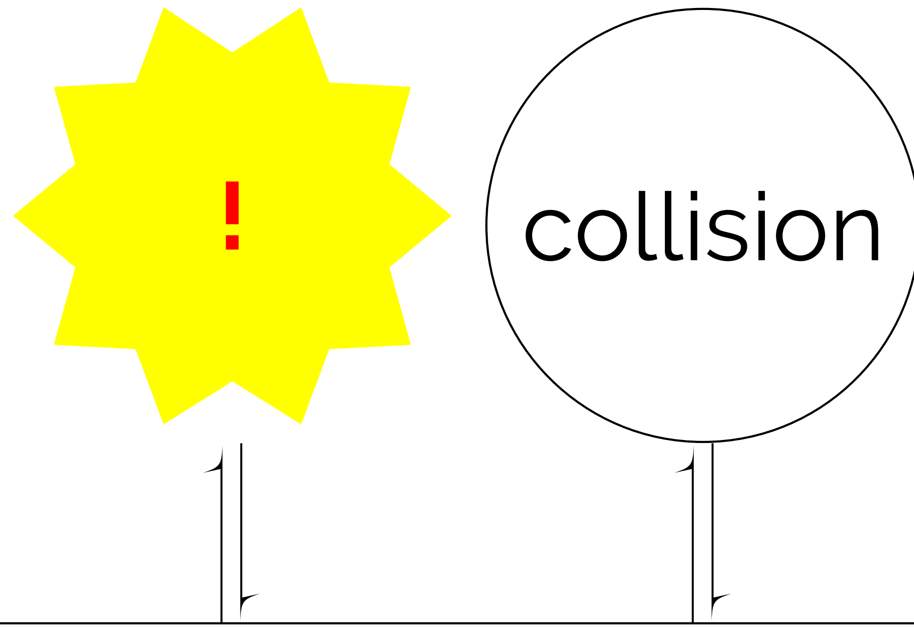
Partial Failure: Actors (Erlang)



Partial Failure: Syndicate

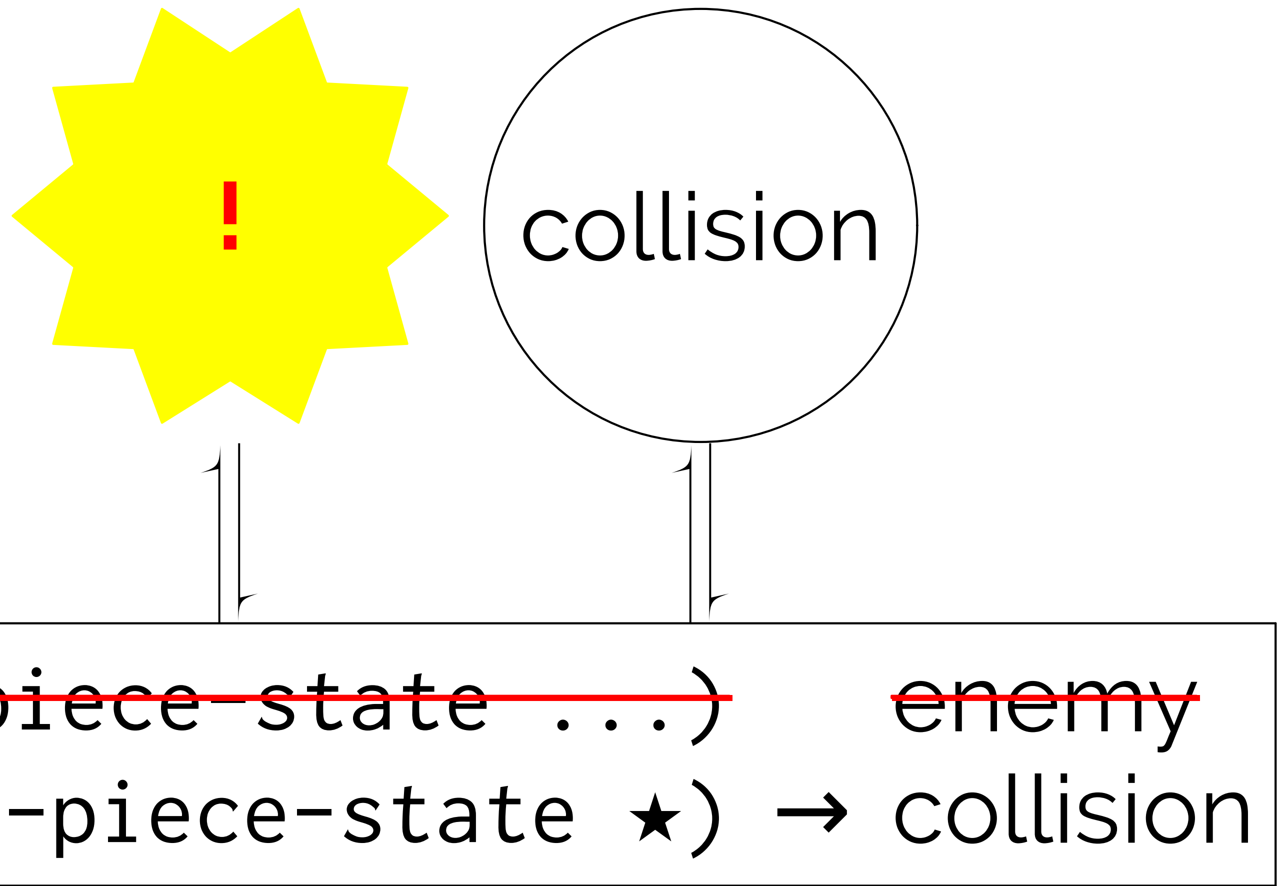


Partial Failure: Syndicate

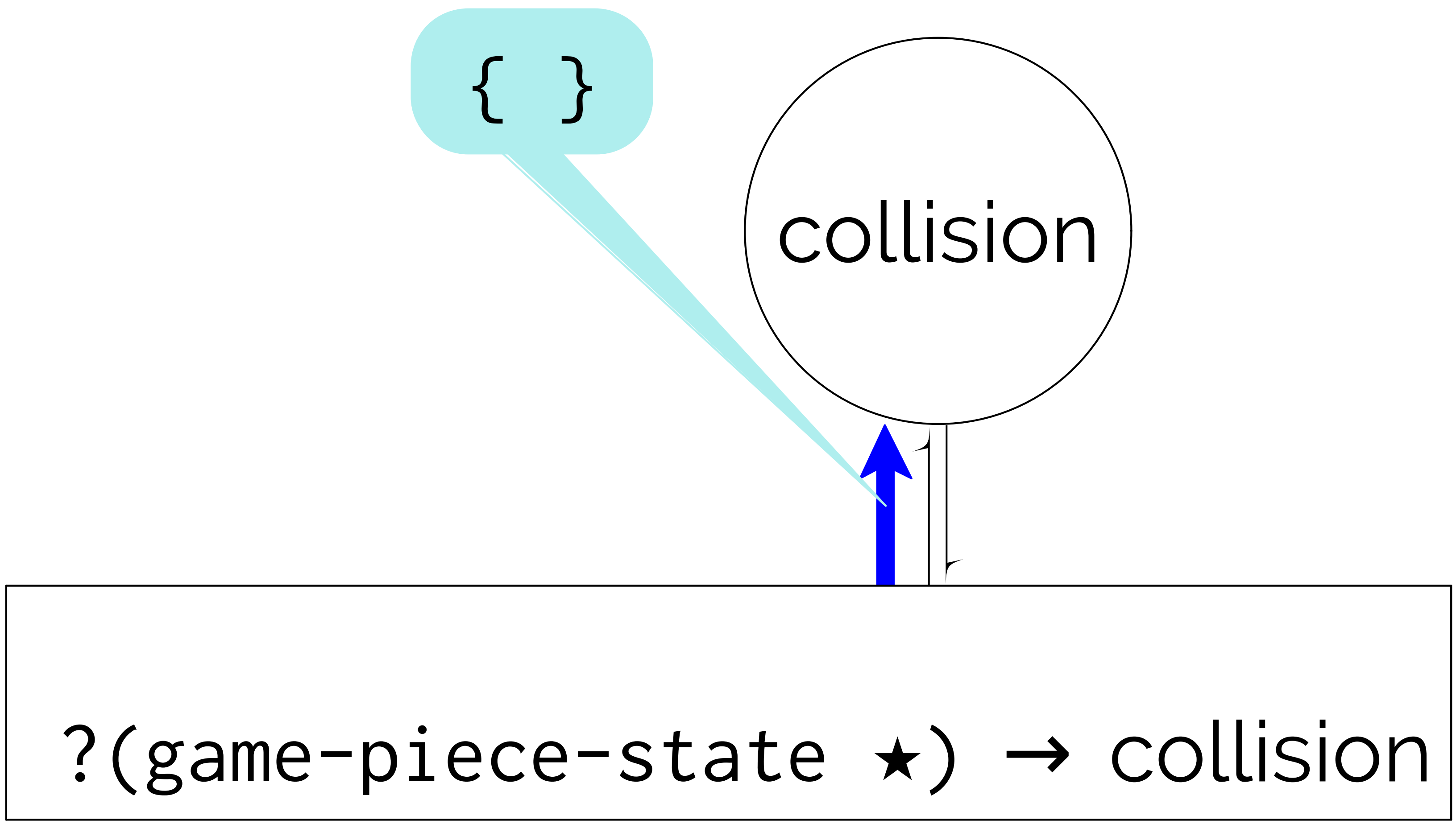



```
(game-piece-state ...) → enemy
 ?(game-piece-state ★) → collision
```

Partial Failure: Syndicate



Partial Failure: Syndicate



Partial Failure: Syndicate

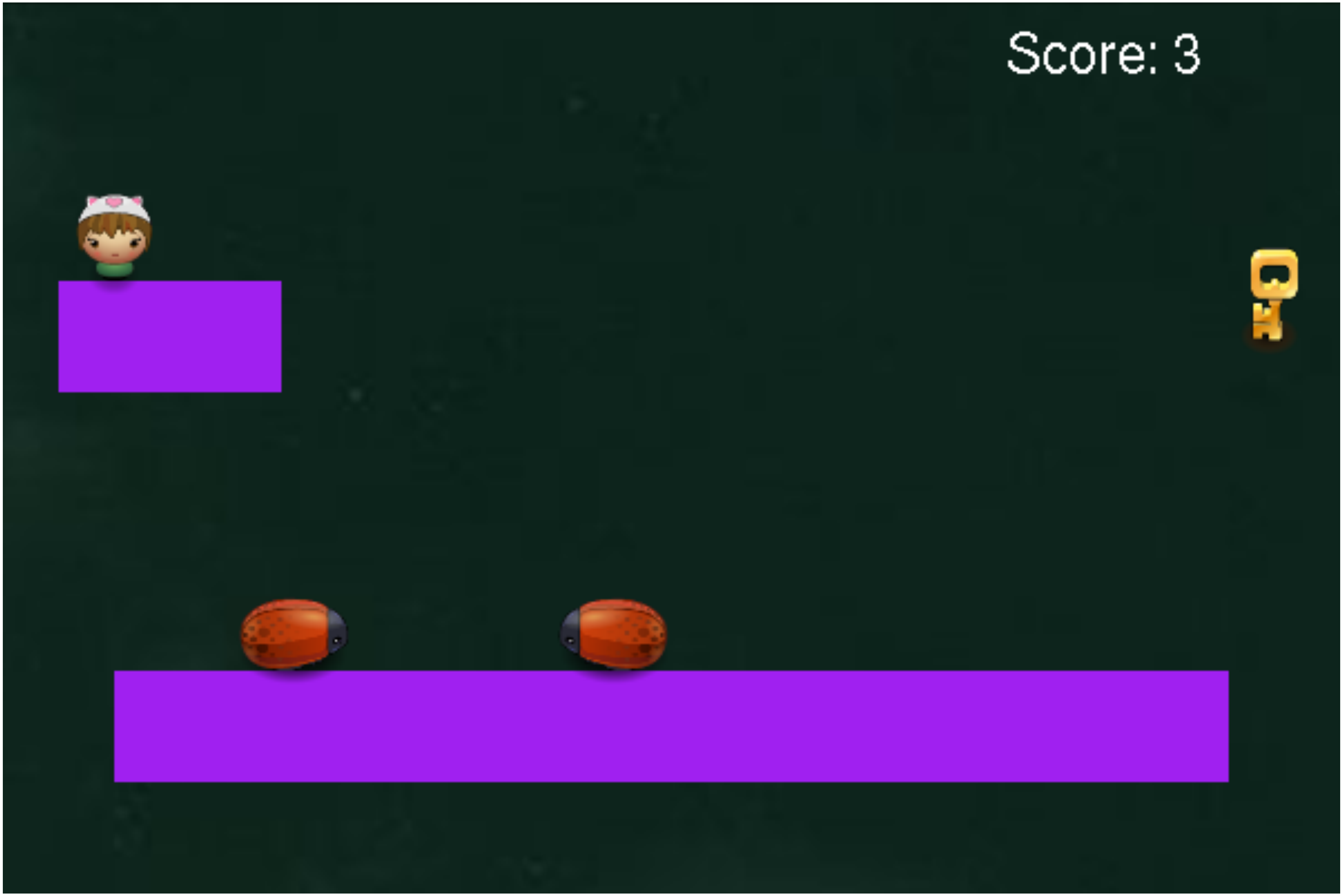
collision

`?(game-piece-state ★) → collision`

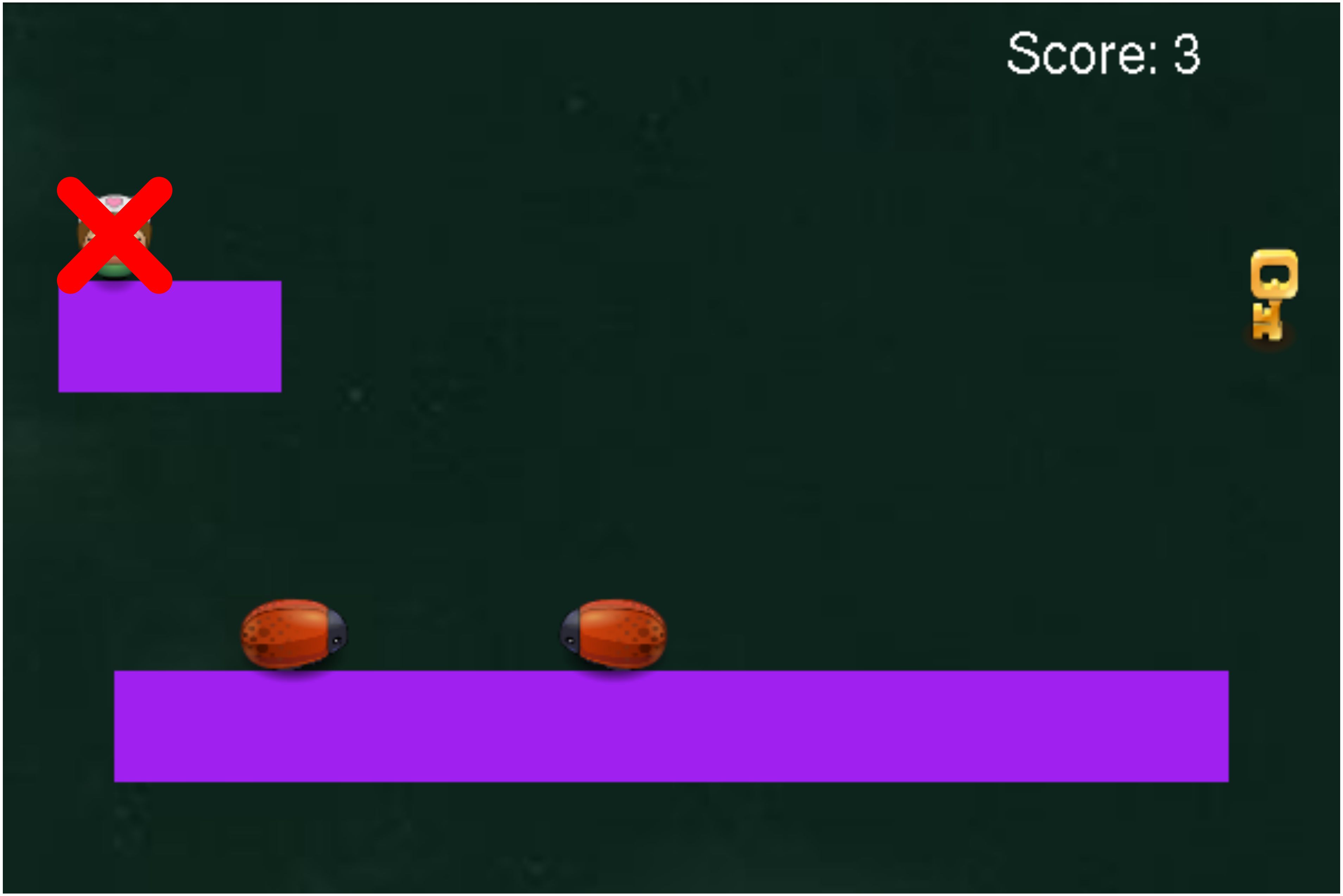
General challenges of interactivity

✓ Mapping events to components
✓ Building a shared understanding
✓ Partial failure
- Scoped conversational state

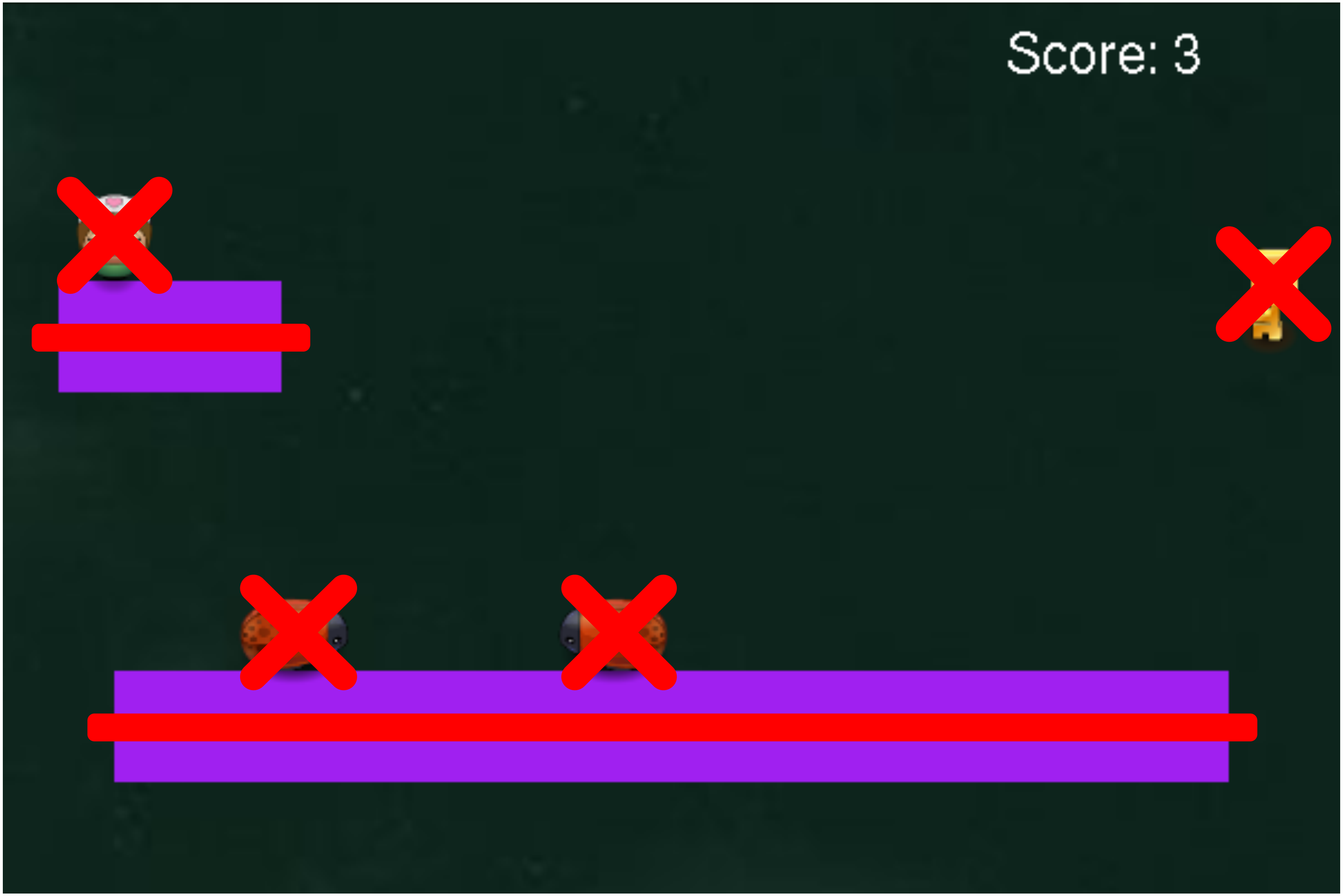
Scoped Conversational State: OO & Actors



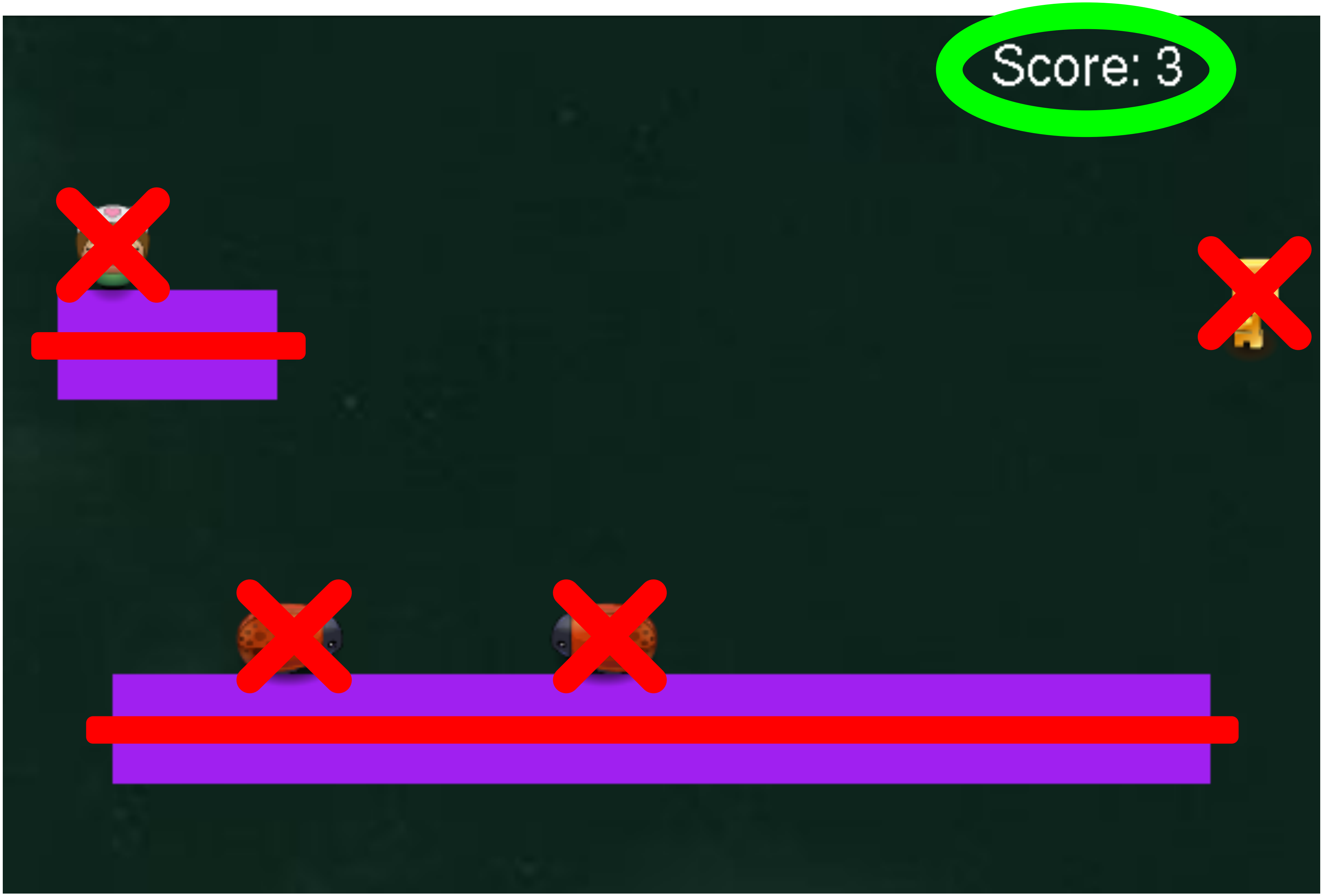
Scoped Conversational State: OO & Actors



Scoped Conversational State: OO & Actors



Scoped Conversational State: OO & Actors

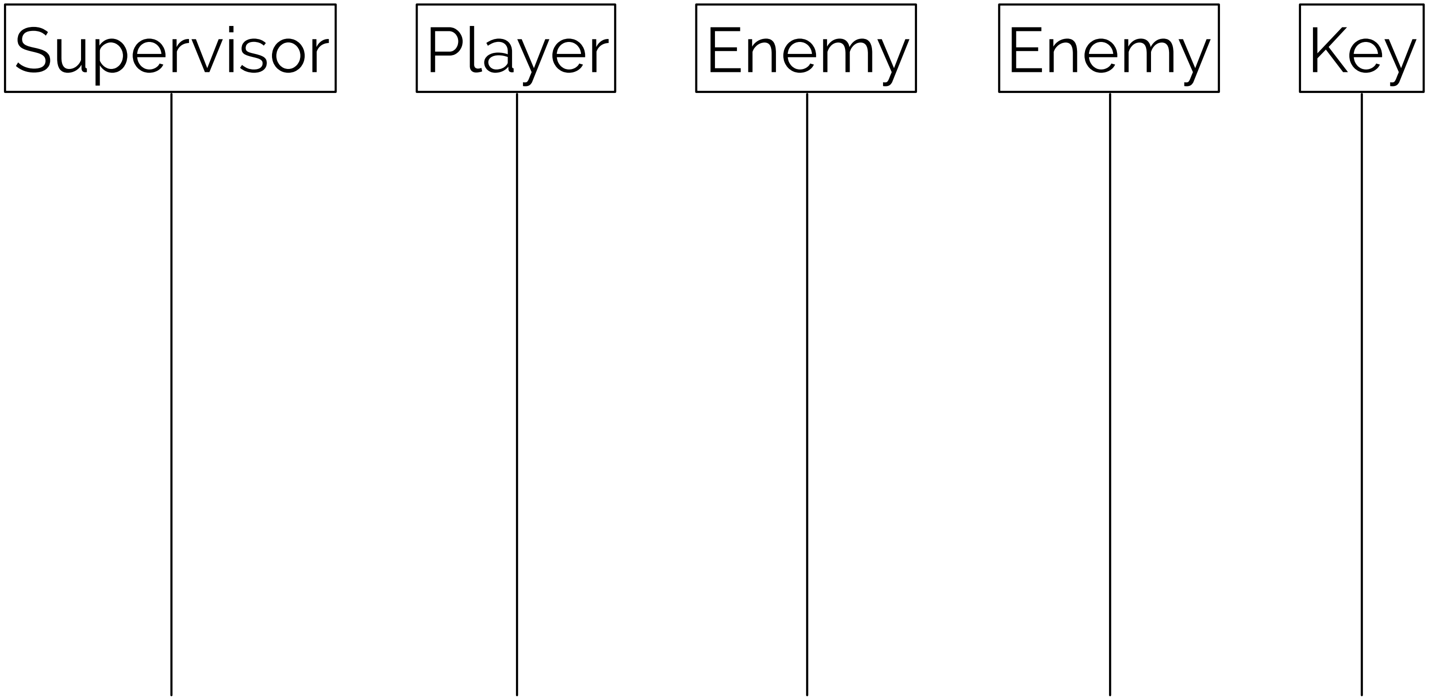


Scoped Conversational State: OO

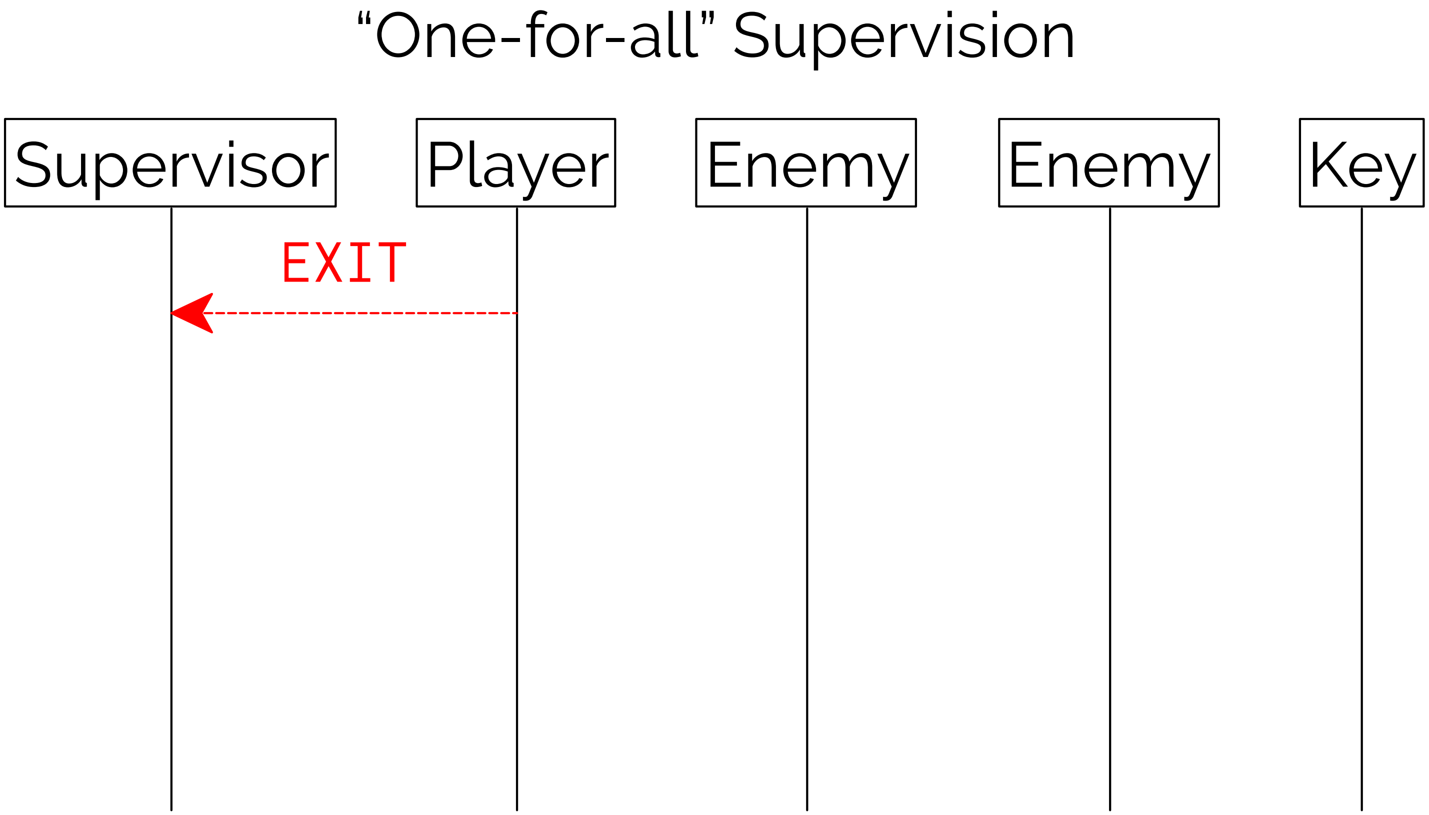
```java
public class LevelInstance {
    private PlayerAvatar avatar;
    private Set<EnemyPiece> enemies;
    private GoldenKey key;
    ...

    private ScoreKeeper scoreKeeper; // needed to be able to add points

    public void dispose() {
        avatar.dispose();
        for (EnemyPiece e : enemies) e.dispose();
        key.dispose();
        // don't accidentally dispose scoreKeeper here!
    }
}
```

Scoped Conversational State: Actors (Erlang/OTP)

"One-for-all" Supervision

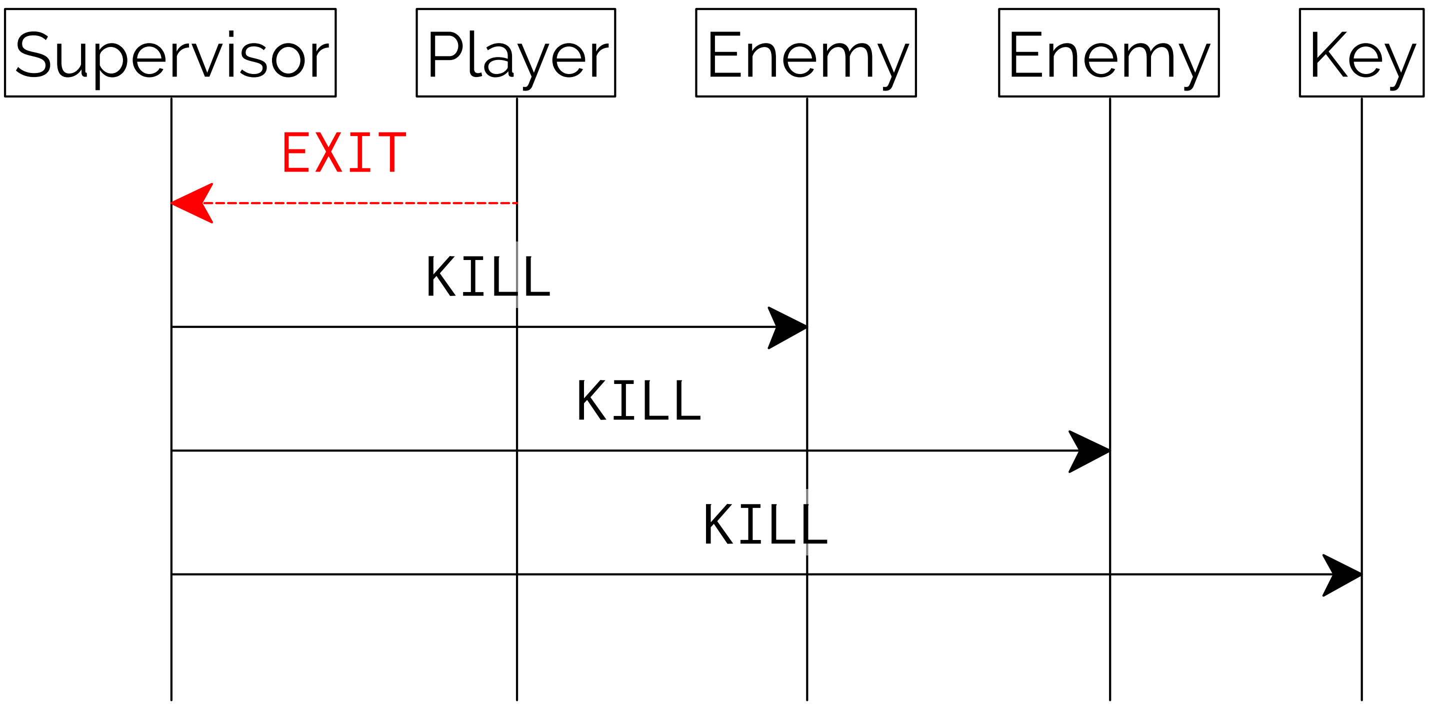


Scoped Conversational State: Actors (Erlang/OTP)



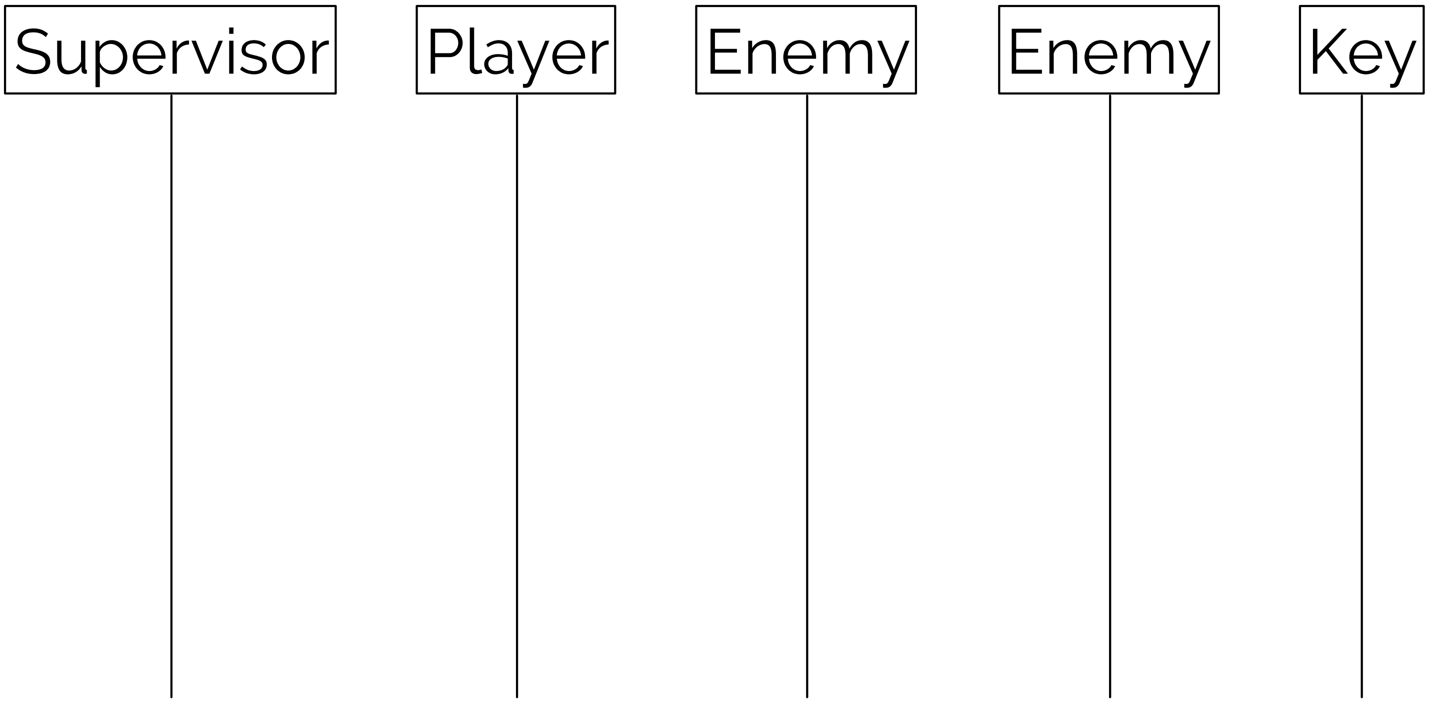
Scoped Conversational State: Actors (Erlang/OTP)

“One-for-all” Supervision



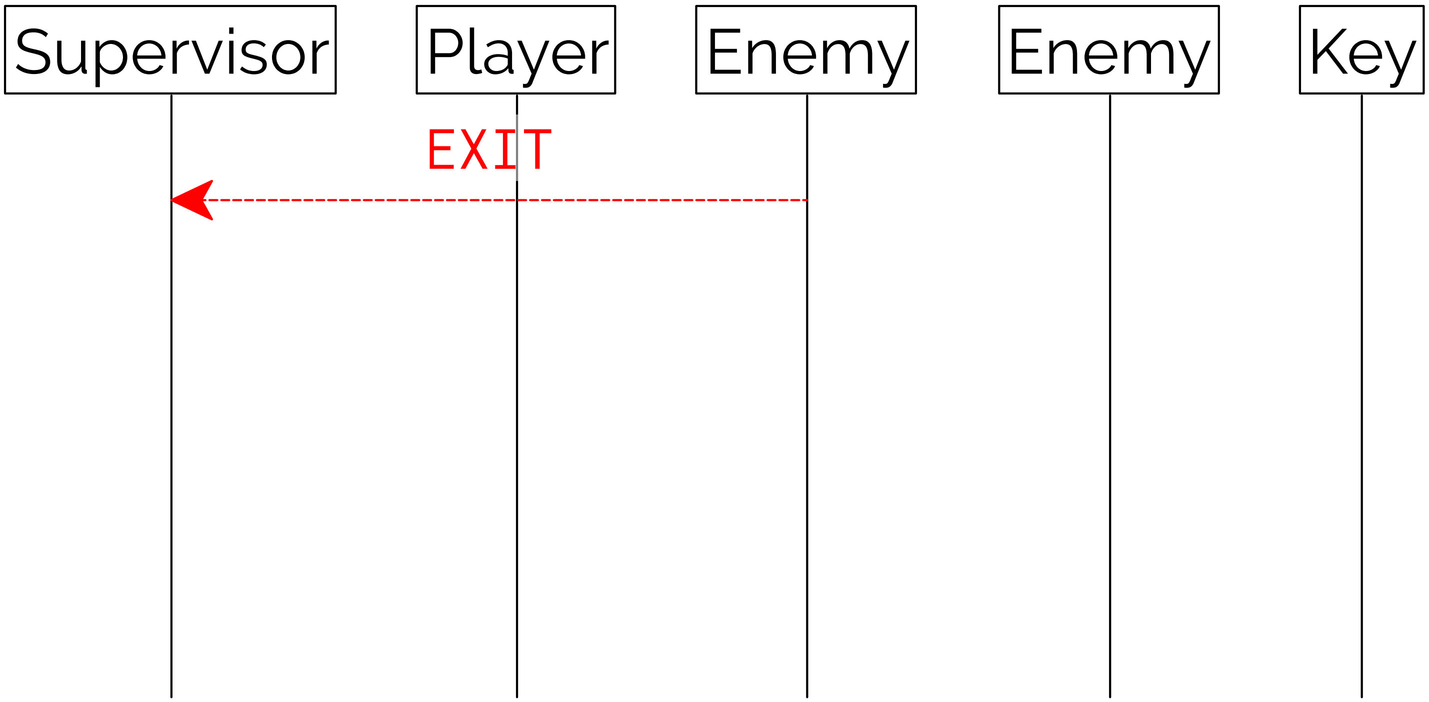
Scoped Conversational State: Actors (Erlang/OTP)

“One-for-all” Supervision

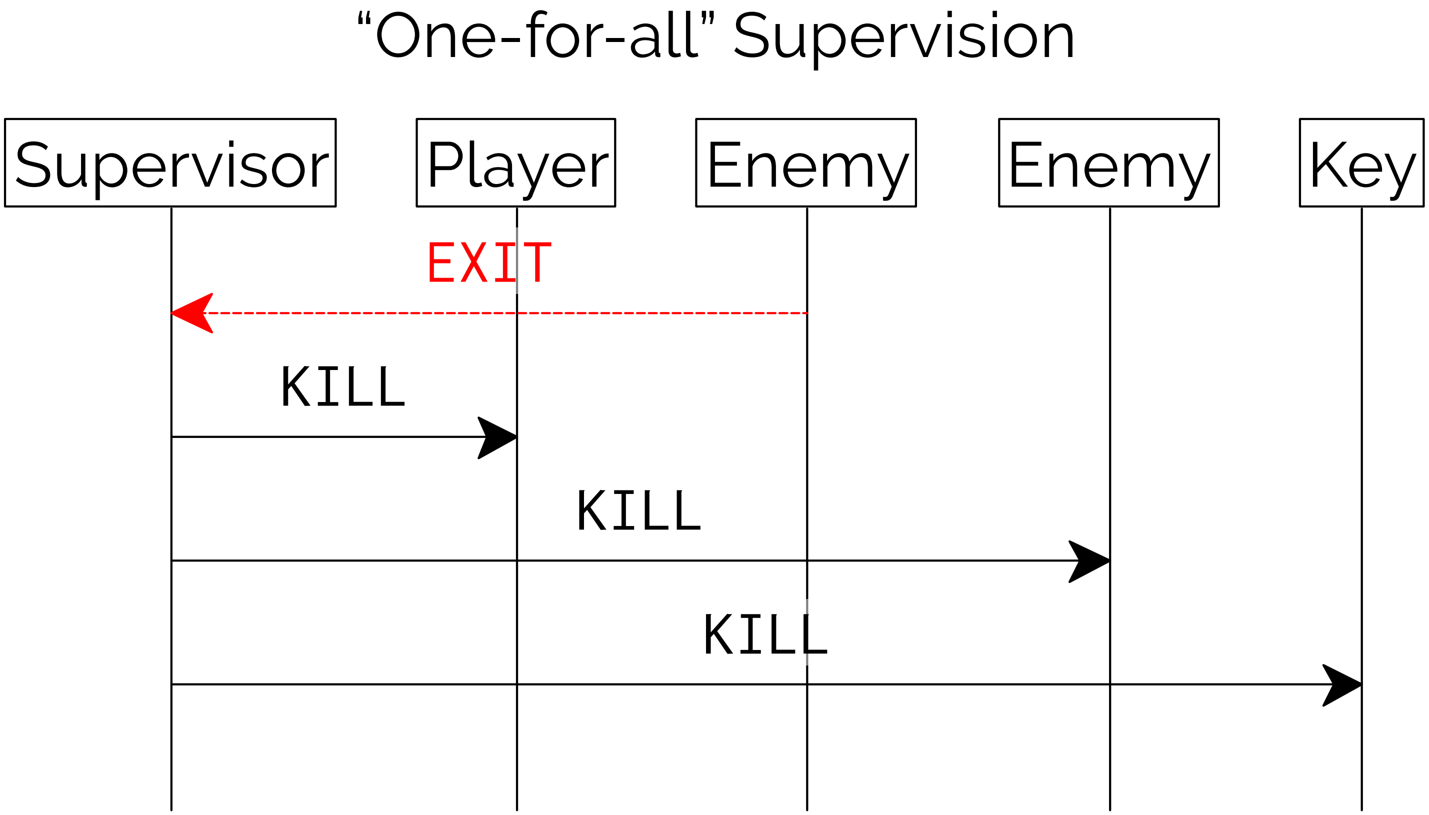


Scoped Conversational State: Actors (Erlang/OTP)

“One-for-all” Supervision



Scoped Conversational State: Actors (Erlang/OTP)



Scoped Conversational State: Syndicate

```
(dataspace
    (spawn-score-keeper)

    ;; The level:
    (dataspace
        (spawn-enemy1)
        (spawn-enemy2)
        (spawn-golden-key)
        (spawn-fixed-blocks)
        (spawn-player) ;; asserts 'player-alive

        (until (retracted 'player-alive)))

    (until (asserted 'game-over)))
```

Grade table

	OO/Callbacks/Threads	Actors	SYNDICATE
Mapping events to components	ZERO	ZERO	OK
Building a shared understanding	ZERO	OK–	OK
Partial failure	ZERO	OK–	OK
Scoped conversational state	ZERO	OK–	OK

ZERO → OK– → OK → OK+

Grade table

Pattern-matching via assertions of interest

	OO/…llbacks/Threads	Actors	SYNDICATE
Mapping events to components	ZERO	ZERO	OK
Building a shared understanding	ZERO	OK–	OK
Partial failure	ZERO	OK–	OK
Scoped conversational state	ZERO	OK–	OK

ZERO → OK– → OK → OK+

Grade table

The dataspace **is** the shared understanding!

	/Callbacks/Threads	Actors	SYNDICATE
Mapping events to components	ZERO	ZERO	OK
Building a shared understanding	ZERO	OK–	OK
Partial failure	ZERO	OK–	OK
Scoped conversational state	ZERO	OK–	OK

ZERO → OK– → OK → OK+

Grade table

Automatic retraction of assertions

	OO/Callbacks/Threads	Actors	SYNDICATE
Mapping			OK
Building a shared understanding	ZERO	OK–	OK
Partial failure	ZERO	OK–	OK
Scoped conversational state	ZERO	OK–	OK

ZERO → OK– → OK → OK+

Grade table

	OO/Callbacks/Threads	Actors	SYNDICATE
M	RO	ZERO	OK
Bu	RO	OK–	OK
Partial failure	ZERO	OK–	OK
Scoped conversational state	ZERO	OK–	OK

Nested dataspaces + controlled assertion flow between them

ZERO → OK– → OK → OK+

Grade table

	OO/Callbacks/Threads	Actors	SYNDICATE
Mapping events to components	ZERO	ZERO	OK
Building a shared understanding	ZERO	OK–	OK
Partial failure	ZERO	OK–	OK
Scoped conversational state	ZERO	OK–	OK

ZERO → OK– → OK → OK+

syn·di·cate

a language for interactive programs

Actors + Dataspaces + Assertions + Nesting

Paper:
- Formal semantics & basic properties
- Incremental SCN protocol & equivalence thm
- Tries for efficient dataspace implementation
- Performance model & measurements
- Case studies: TCP/IP stack, GUI widget

`http://syndicate-lang.org/`