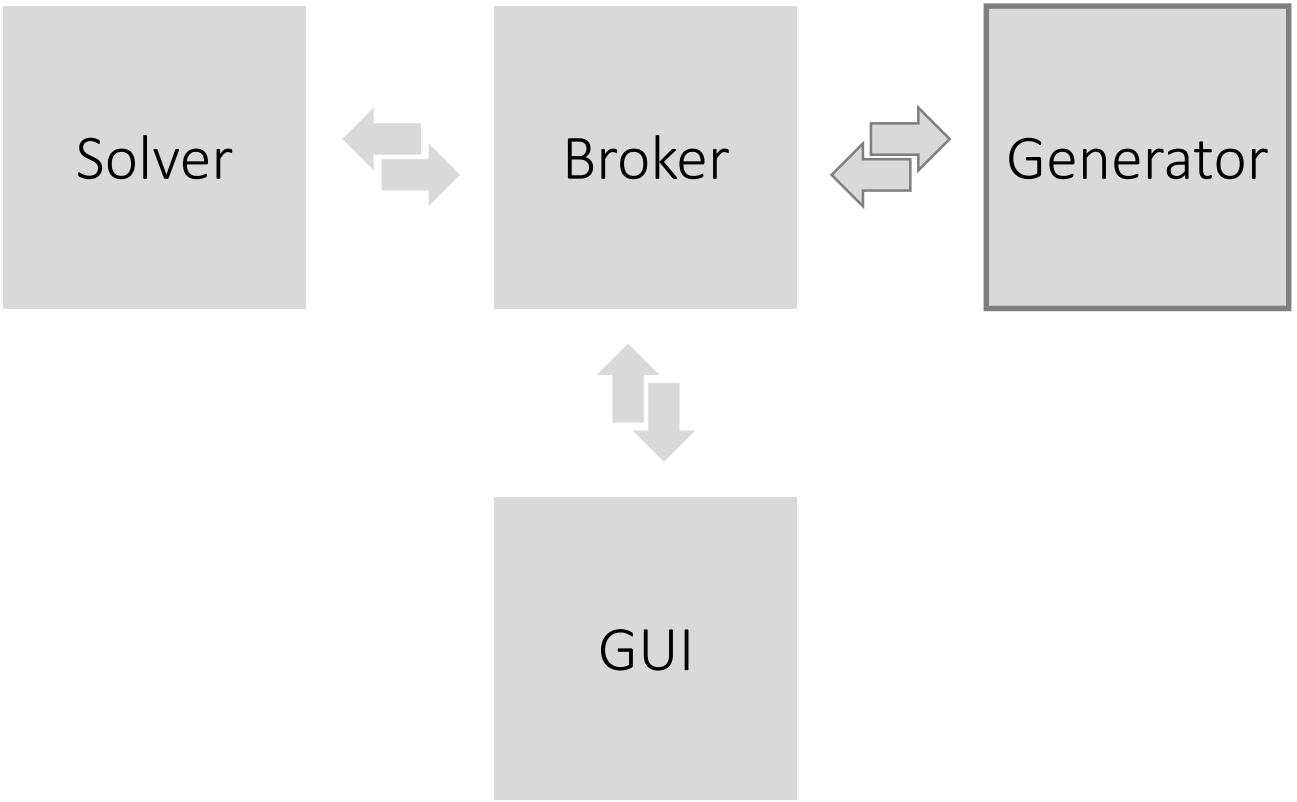
Generator

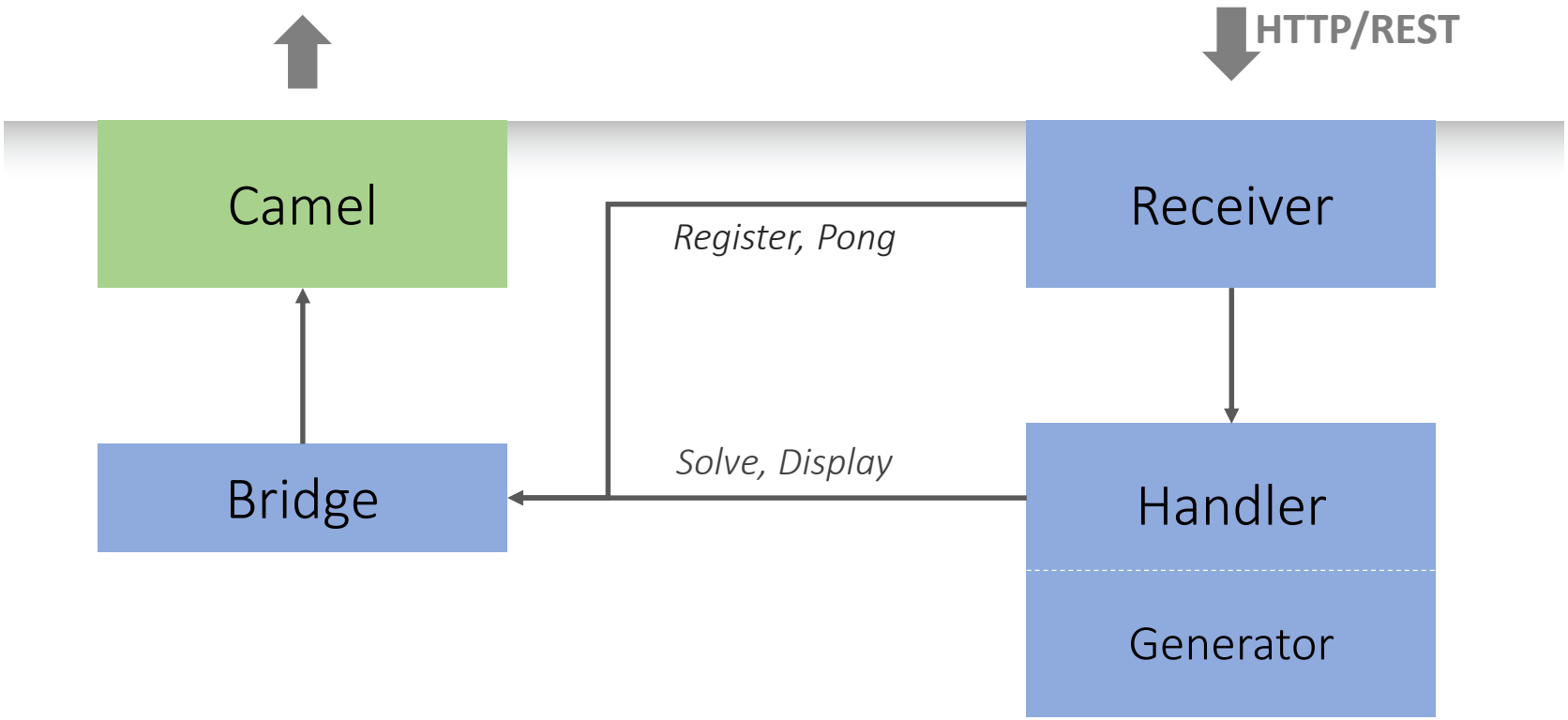
Verteiltes Sudoku

Apache Camel, ZeroMQ, REST, Python

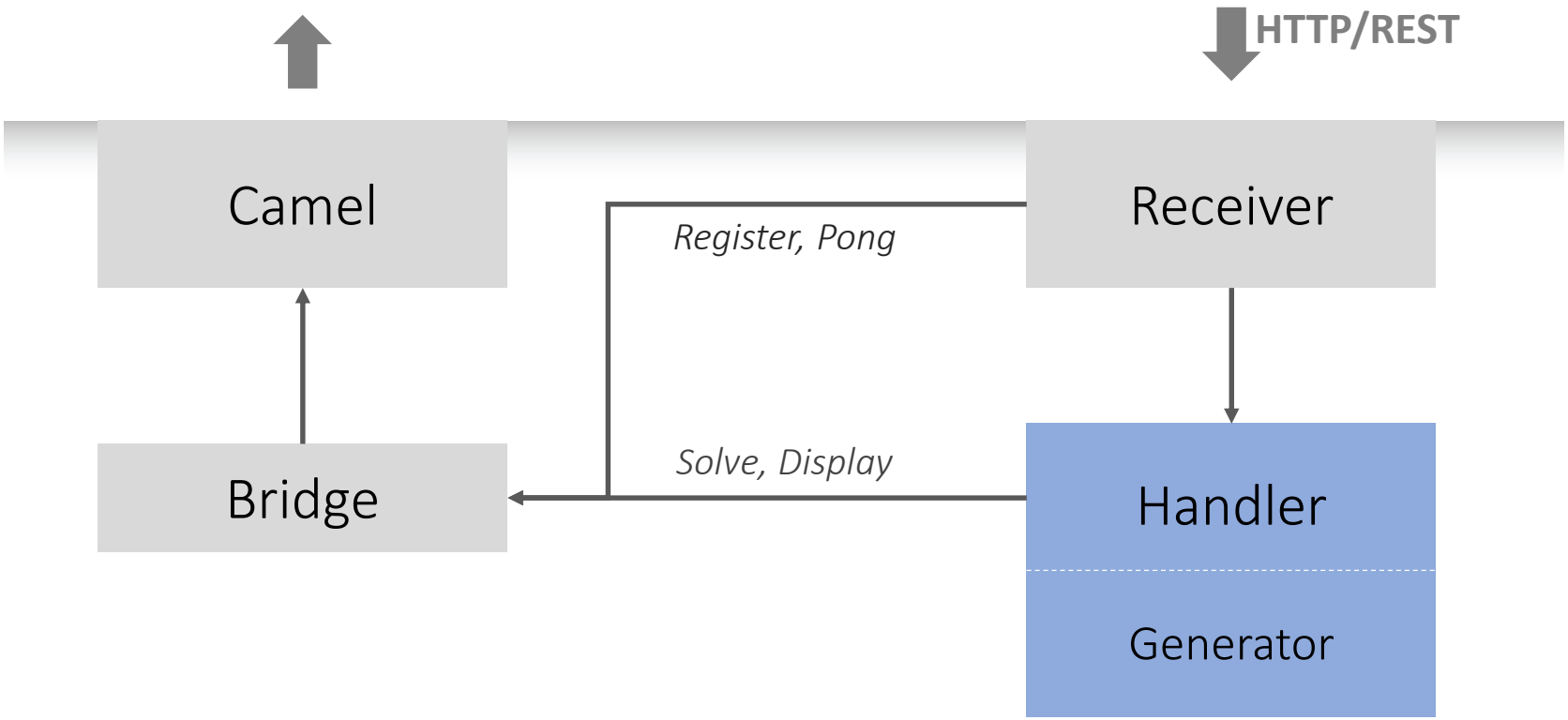
Kontext



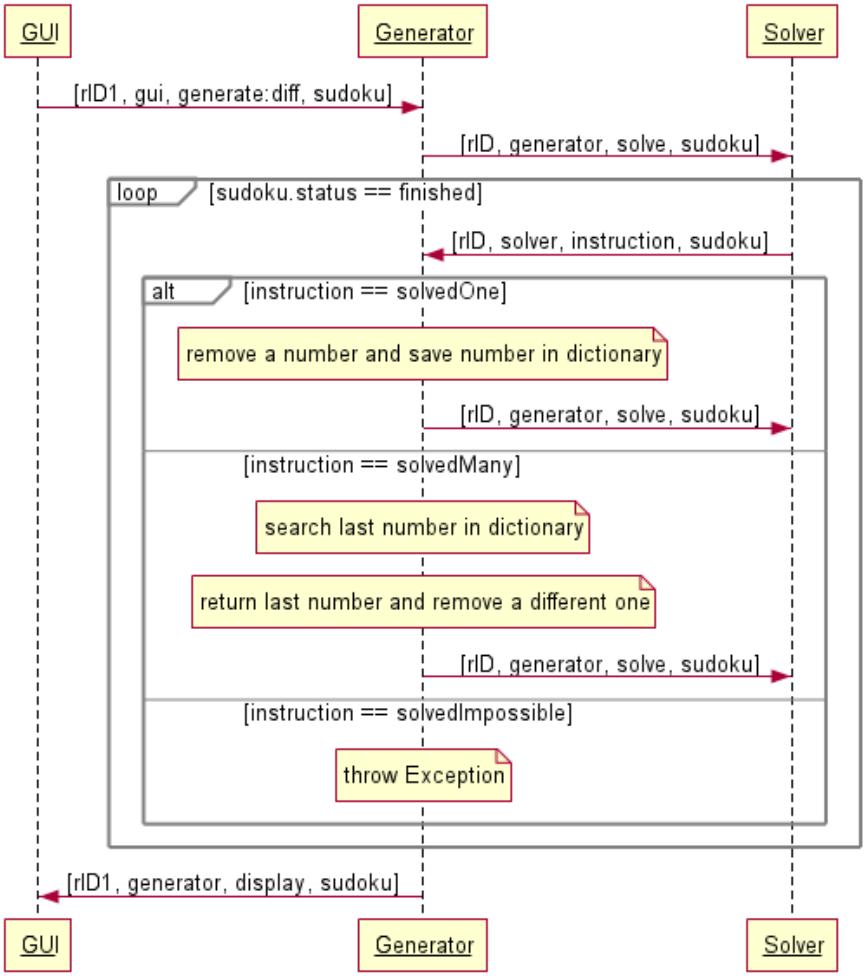
Architektur



Bussineslogik



Sequenzdiagramm



Generate

- Gültigkeit
  ((math.sqrt(len(sudoku))) % 1) == 0

- Senken der Netzwerklast
  len(msg.sudoku) == 1

Initial Sudoku

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	3	4	5	6	7	8	9	1
5	6	7	8	9	1	2	3	4
8	9	1	2	3	4	5	6	7
3	4	5	6	7	8	9	1	2
6	7	8	9	1	2	3	4	5
9	1	2	3	4	5	6	7	8

Swap Numbers

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	3	4	5	6	7	8	9	1
5	6	7	8	9	1	2	3	4
8	9	1	2	3	4	5	6	7
3	4	5	6	7	8	9	1	2
6	7	8	9	1	2	3	4	5
9	1	2	3	4	5	6	7	8

1	5	3	4	2	6	7	8	9
4	2	6	7	8	9	1	5	3
7	8	9	1	5	3	4	2	6
5	3	4	2	6	7	8	9	1
2	6	7	8	9	1	5	3	4
8	9	1	5	3	4	2	6	7
3	4	2	6	7	8	9	1	5
6	7	8	9	1	5	3	4	2
9	1	5	3	4	2	6	7	8

Swap Rows

1	5	3	4	2	6	7	8	9
4	2	6	7	8	9	1	5	3
7	8	9	1	5	3	4	2	6
5	3	4	2	6	7	8	9	1
2	6	7	8	9	1	5	3	4
8	9	1	5	3	4	2	6	7
3	4	2	6	7	8	9	1	5
6	7	8	9	1	5	3	4	2
9	1	5	3	4	2	6	7	8

1	5	3	4	2	6	7	8	9
4	2	6	7	8	9	1	5	3
7	8	9	1	5	3	4	2	6
5	3	4	2	6	7	8	9	1
2	6	7	8	9	1	5	3	4
8	9	1	5	3	4	2	6	7
9	1	5	3	4	2	6	7	8
6	7	8	9	1	5	3	4	2
3	4	2	6	7	8	9	1	5

Swap Blocks

1	5	3	4	2	6	7	8	9
4	2	6	7	8	9	1	5	3
7	8	9	1	5	3	4	2	6
5	3	4	2	6	7	8	9	1
2	6	7	8	9	1	5	3	4
8	9	1	5	3	4	2	6	7
9	1	5	3	4	2	6	7	8
6	7	8	9	1	5	3	4	2
3	4	2	6	7	8	9	1	5

5	3	4	2	6	7	8	9	1
2	6	7	8	9	1	5	3	4
8	9	1	5	3	4	2	6	7
1	5	3	4	2	6	7	8	9
4	2	6	7	8	9	1	5	3
7	8	9	1	5	3	4	2	6
9	1	5	3	4	2	6	7	8
6	7	8	9	1	5	3	4	2
3	4	2	6	7	8	9	1	5

Rotate Sudoku

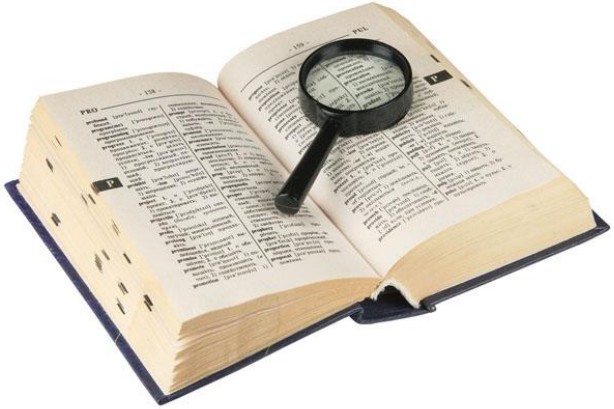
5	3	4	2	6	7	8	9	1
2	6	7	8	9	1	5	3	4
8	9	1	5	3	4	2	6	7
1	5	3	4	2	6	7	8	9
4	2	6	7	8	9	1	5	3
7	8	9	1	5	3	4	2	6
9	1	5	3	4	2	6	7	8
6	7	8	9	1	5	3	4	2
3	4	2	6	7	8	9	1	5

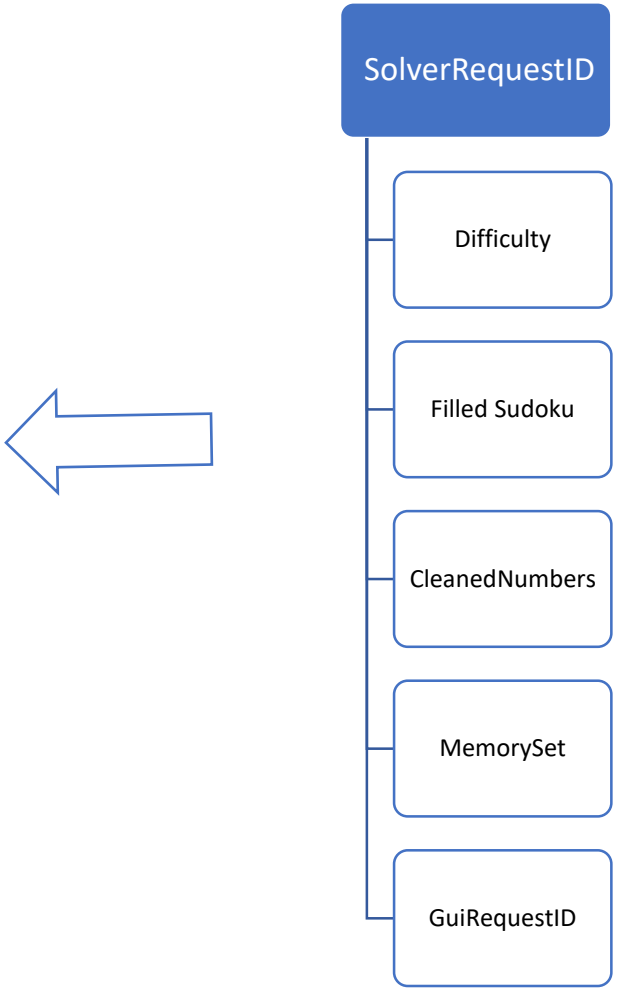
1	4	7	9	3	6	8	2	5
9	3	6	8	5	2	7	4	1
8	5	2	7	1	4	6	3	9
7	1	4	6	9	3	2	5	8
6	9	3	2	8	5	4	1	7
2	8	5	4	7	1	3	9	6
4	7	1	3	6	9	5	8	2
3	6	9	5	2	8	1	7	4
5	2	8	1	4	7	9	6	3

Randomisierung

```
117 def generateFilledSudoku(k=3):
118     """
119     Generiert ein Random-Sudoku der Groesse k*k und gibt es zurueck.
120     """
121     sudoku = generateInitialSudoku(k=k)
122
123     for i in range(1,4):
124         for j in range(0,random.randint(k,k*2)):
125             if(i == 1):
126                 swapNumbers(sudoku)
127             elif(i == 2):
128                 swapRows(sudoku)
129             elif(i == 3):
130                 swapBlocks(sudoku)
131             else:
132                 rotateSudoku(sudoku)
133
134     return sudoku
```

Generate




SolverRequestID
Difficulty
Filled Sudoku
CleanedNumbers
MemorySet
GuiRequestID

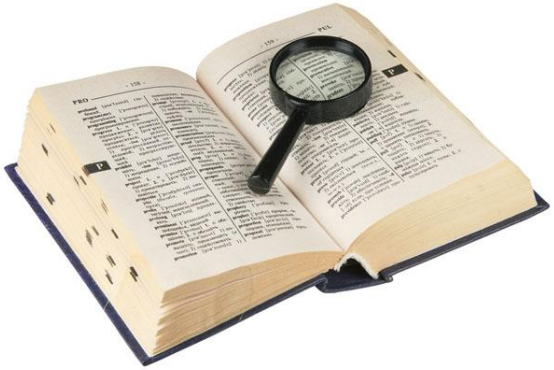
Generate

1		3	4	5	6	7	8	9
4	5	6	7	8	9	1		3
7	8	9	1	2		4	5	6
2	3	4	5	6	7	8	9	1
5		7	8		1	2	3	4
8	9	1	2	3	4		6	7
3	4	5	6	7	8	9	1	2
6	7		9		2	3	4	5
9	1	2	3	4	5	6	7	8

Solver

SolvedOne

- Gültigkeit der Antwort



ID ?

SolvedOne

1		3	4	5	6	7	8	9
4	5	6	7	8	9	1		3
7	8	9	1	2		4	5	6
2	3	4	5	6	7	8	9	1
5		7	8		1	2	3	4
8	9	1	2	3	4		6	7
3	4	5	6	7	8	9	1	2
6	7		9		2	3		5
9	1	2	3	4	5	6	7	8

SolvedOne

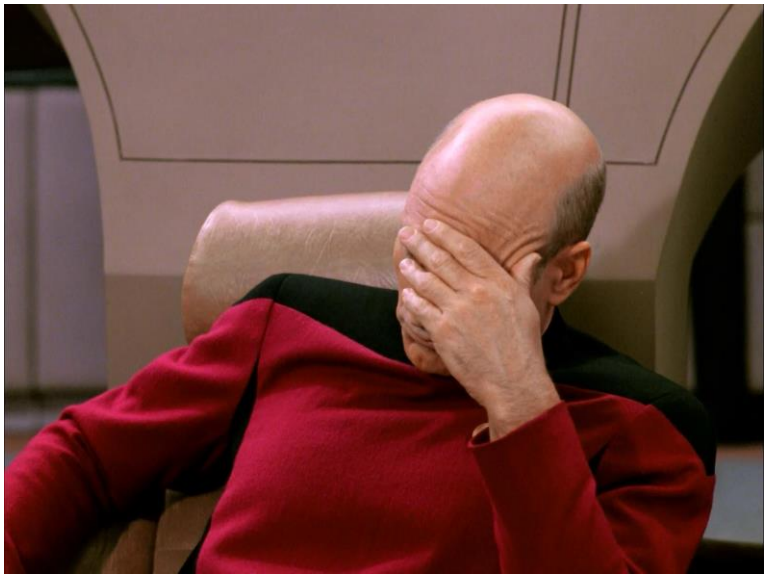
1		3	4	5	6	7	8	9
4	5	6	7	8	9	1		3
7	8	9	1	2		4	5	6
2	3	4	5	6		8	9	1
5		7	[illegible]		1	2	3	4
8	9	1	2		4		6	7
3	4	5	6	7	8	9	1	2
6	7		9		2	3		5
9	1	2	3	4	5	6	7	8

SolvedOne

1		3	4	5	6	7	8	9
4	5	6	7	8	9	1		3
7	~~8~~	9	1	2		4	5	6
2	3	4	5	6	7	8	9	1
5		7	8		1	2	3	4
8	9	1	2	3	4		6	7
3	4	5	6	7	8	9	1	2
6	7		9		2	3		5
9	1	2	3	4	5	6	7	8

SolvedMany

- Gültigkeit der Antwort 
- Eine "falsche Zahl" entfernt



SolvedMany

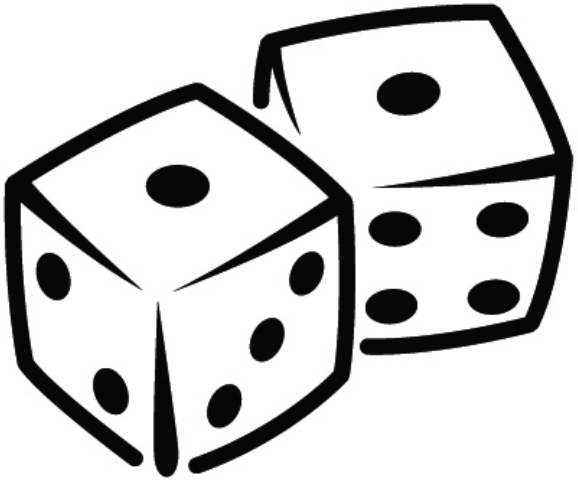
1		3	4	5	6	7	8	9
4	5	6	7	8	9	1		3
7	8							
2	3	4	5	6	7	8	9	1
5		7	8		1	2	3	4
8	9	1	2	3	4		6	7
3	4	5	6	7	8	9	1	2
6	7		9		2	3		5
9	1	2	3	4	5	6	7	8

SolvedMany

1		3	4	5	6	7	8	9
4	5	6	7	8	9	1		3
7	8	9						
2	3	4	5	6	7	8	9	1
5		7	8		1	2	3	4
8	9	1	2	3	4		6	7
3	4	5	6	7	8	9	1	2
6	7		9		2	3		5
9	1	2	3	4	5	6	7	8

SolvedMany

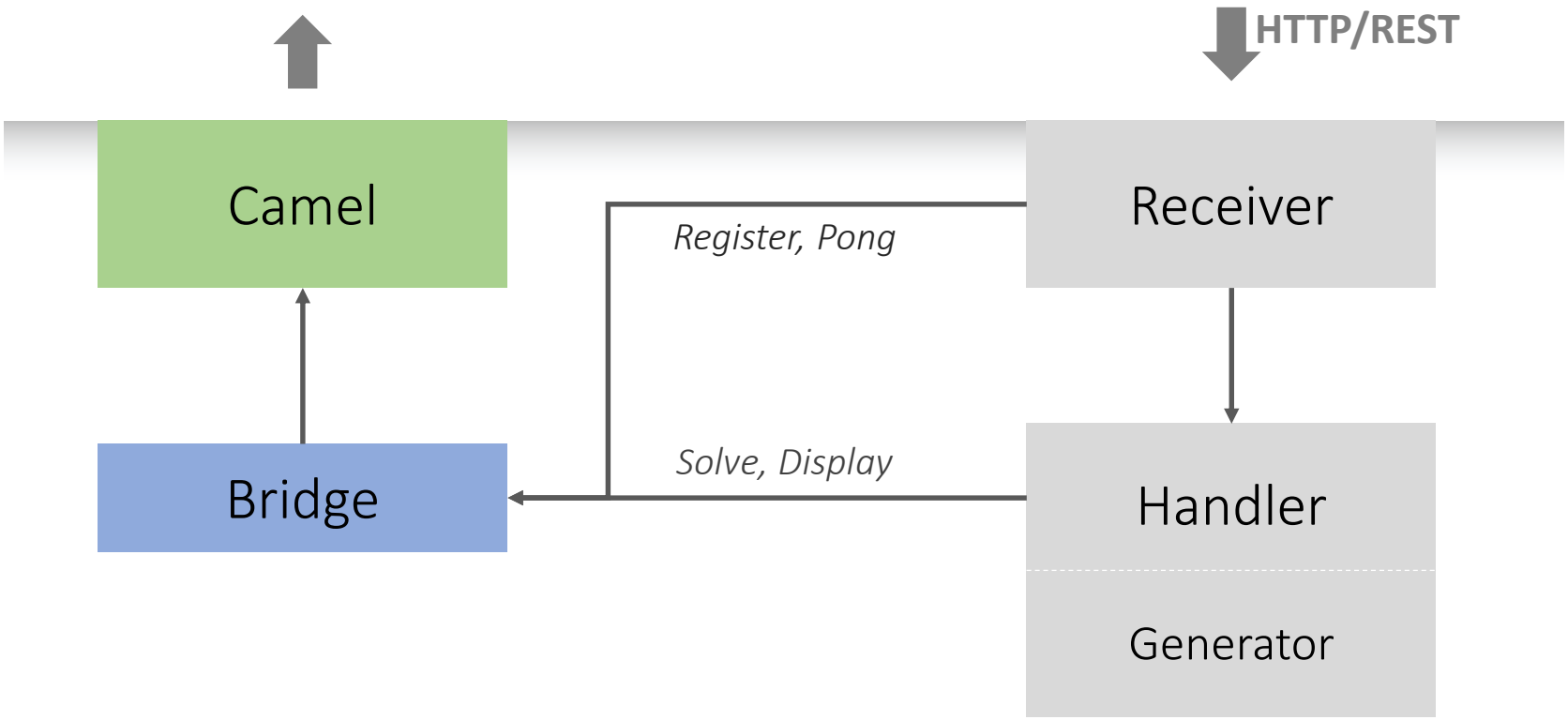
~~1~~		3	4	5	6	7	8	9
4	5	6	7	8	9	1		3
7	8	9	1	2		4	5	6
2	3	4	5	6	7	8	9	1
5		7	8		1	2	3	4
8	9	1	2	3	4		6	7
3	4	5	6	7	8	9	1	2
6	7		9		2	3		5
9	1	2	3	4	5	6	7	8



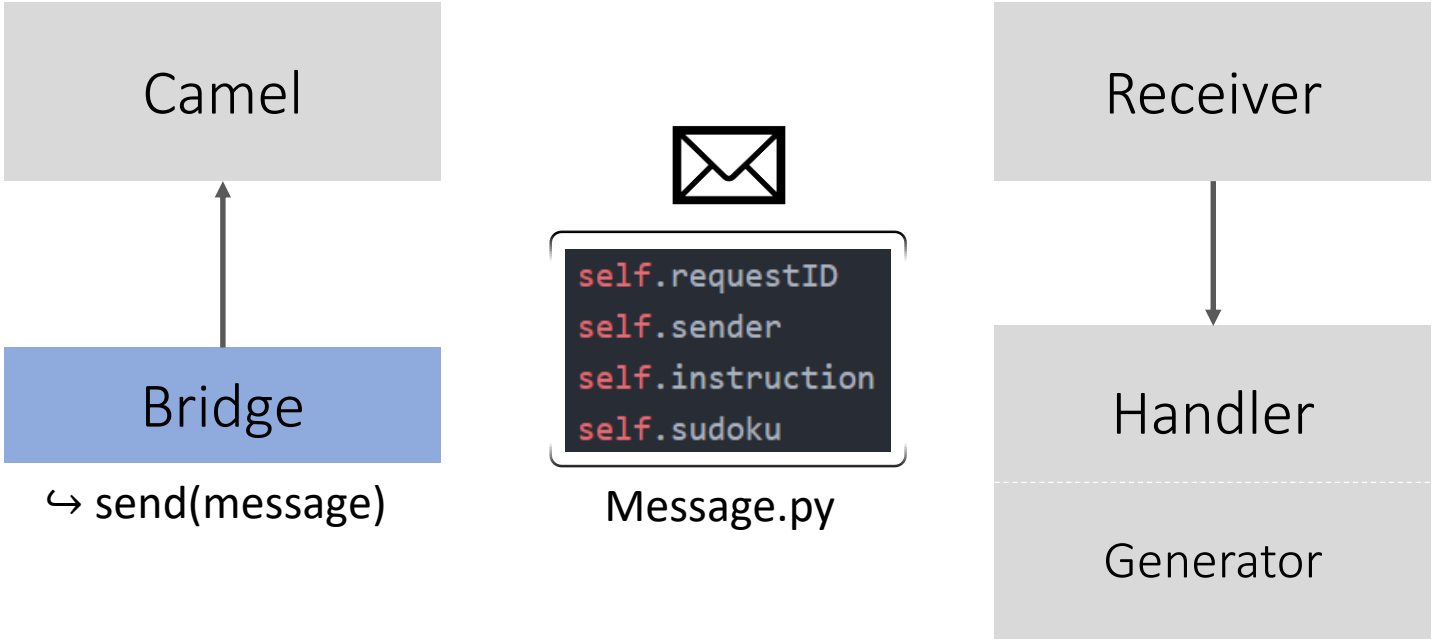
SolvedMany

		3	4	5	6	7	8	9
4	5	6	7	8	9	1		3
7	8	9	1	2		4	5	6
2	3	4	5	6	7	8	9	1
5		7	8		1	2	3	4
8	9	1	2	3	4		6	7
3	4	5	6	7	8	9	1	2
6	7		9		2	3		5
9	1	2	3	4	5	6	7	8

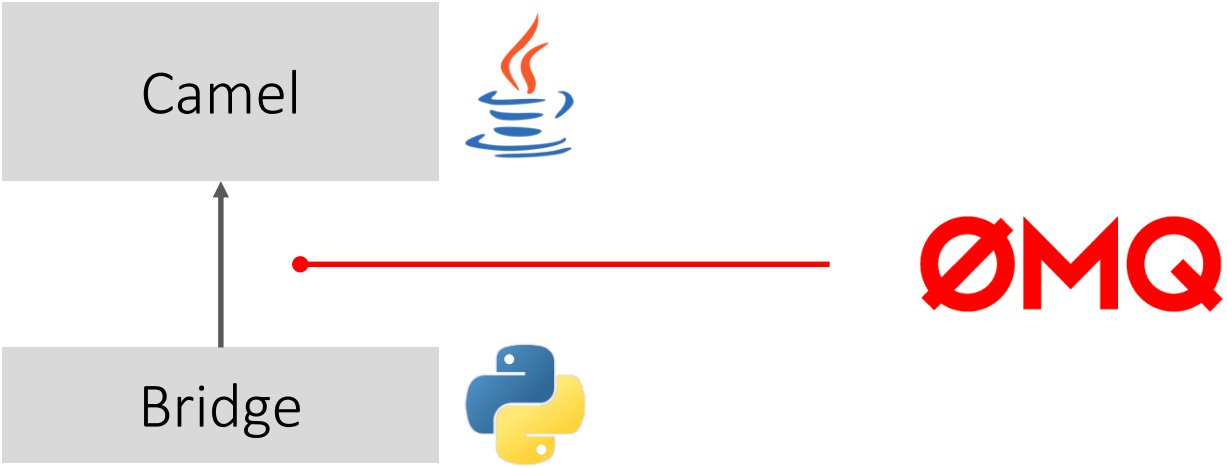
Senden


HTTP/REST
Camel
Receiver
Register, Pong
Bridge
Solve, Display
Handler
Generator

Messages


Camel
Bridge
↪ send(message)
self.requestID
self.sender
self.instruction
self.sudoku
Message.py
Receiver
Handler
Generator

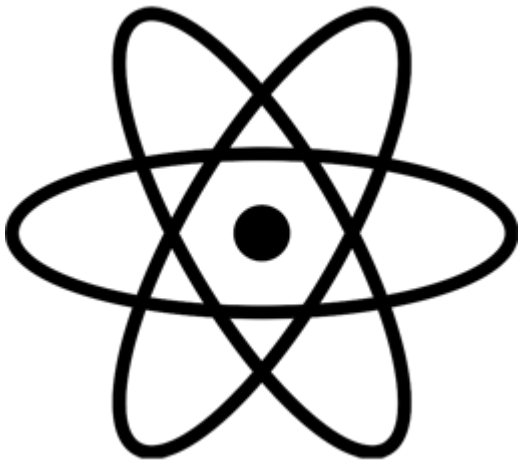
Python & Camel



ZMQ Messages

```
2inR2aoDR4WXMQ12e5BX
2Qw1W1ptcPUbDjcQ5Mge
DYpyMph8SpzbpW5K7hS8
gCvCdBqYeWLTAKSvJWeS
FiA1PorymmKhaKns7zXr
h5vBmhX0EVtxErwL6HkB
fRwLksMvvWWBwDreKM03
j8Wl9QBd6P2yX45zPOH8
8migVgPTHLfwWyfoJ7qY
lXm1LYDypvvV6vFnp3tS
Afbtwdg3j9yX6H99RlzX
```

String



Atomic

ZMQ verwenden

```python
meinSocket = context.socket(zmq.PUSH)
meinSocket.bind("tcp://*:5555")
meinSocket.send_string("Hello World")
```

```java
ZMQ.Socket  socket  = context.socket(ZMQ.PULL);
socket.connect ("tcp://*:5555");
byte[] reply = socket.recv(0);
```

ZMQ verwenden

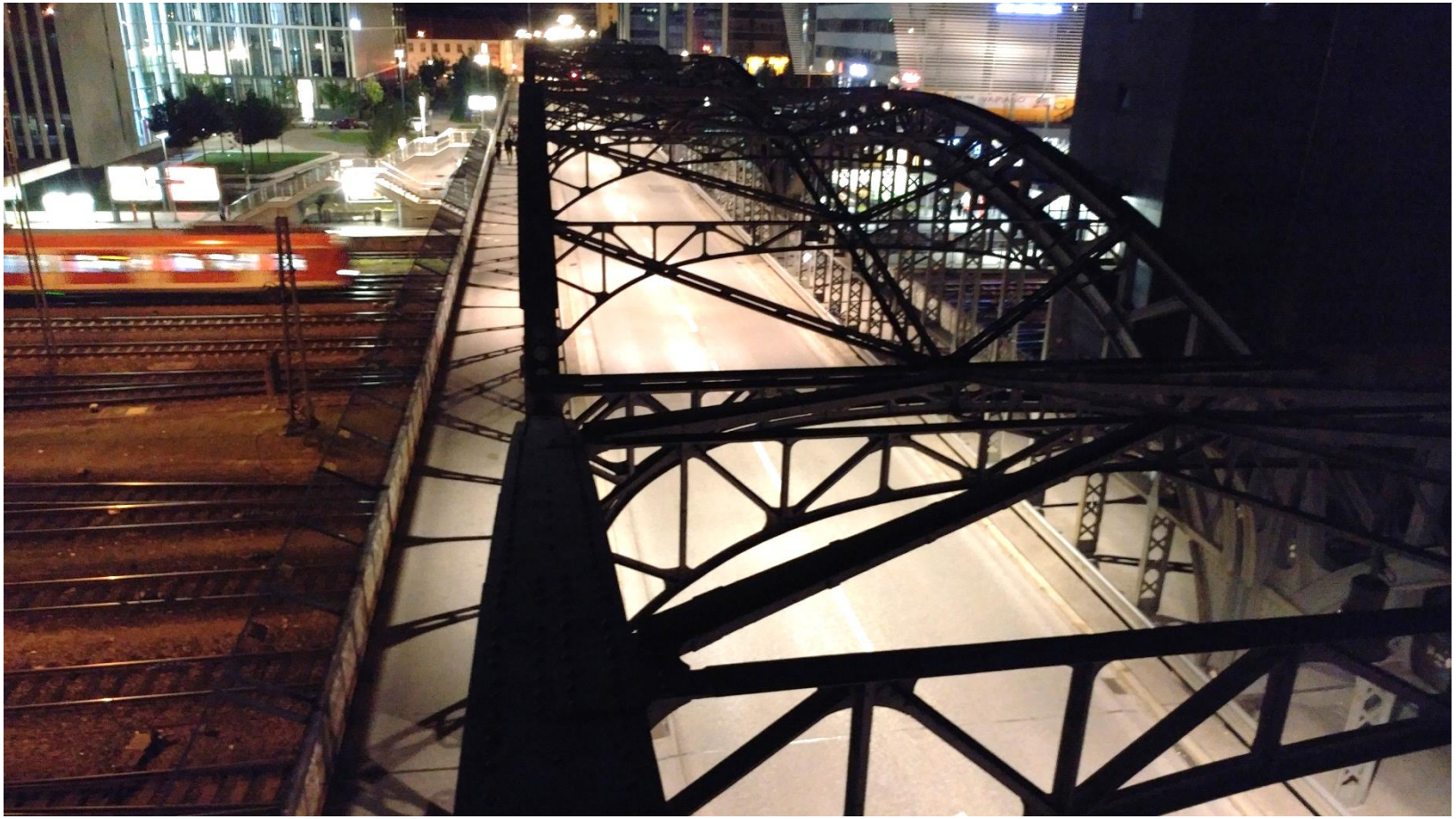
```python
meinSocket = context.socket(zmq.PUSH)
meinSocket.bind("tcp://*:5555")
meinSocket.send_string("Hello World")
```

python

```
from("zeromq:tcp://127.0.0.1:5555?socketType=PULL")
```

Apache Camel

Bridge.py



Camel Route


```
from(String)
    .process(Processor)
    .to(String)
```

Camel Processor

Input

```
936DA01F[…];

restlet:http://[…];

register:generator;

[[1, 2, 3], [4, 5, 6]]
```

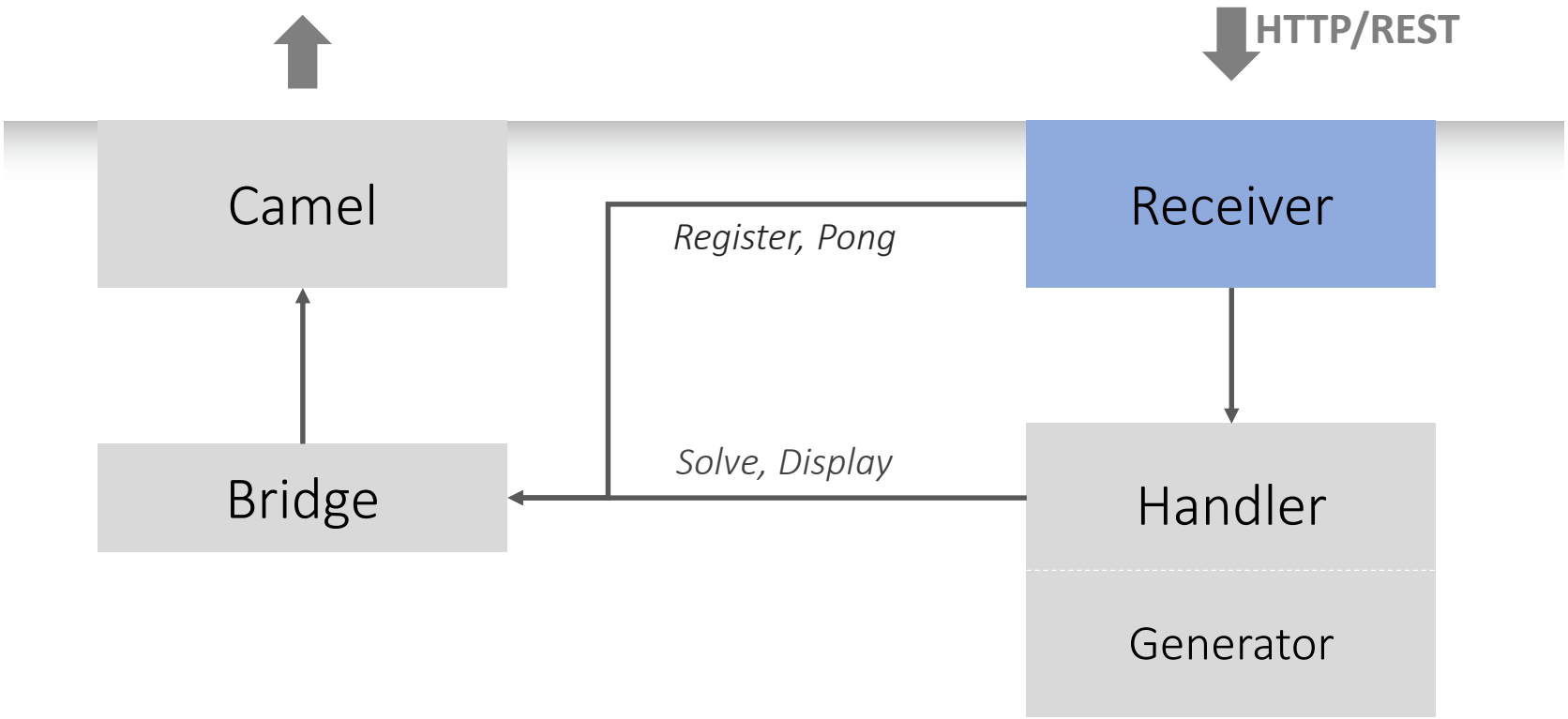
Output

```
{
"request_id":
   "936DA01F[…]",
"sender":
   "restlet:http://[…]",
"instruction":
   "register:generator",
"sudoku":
   [1,2,3,4,5,6]
}
```

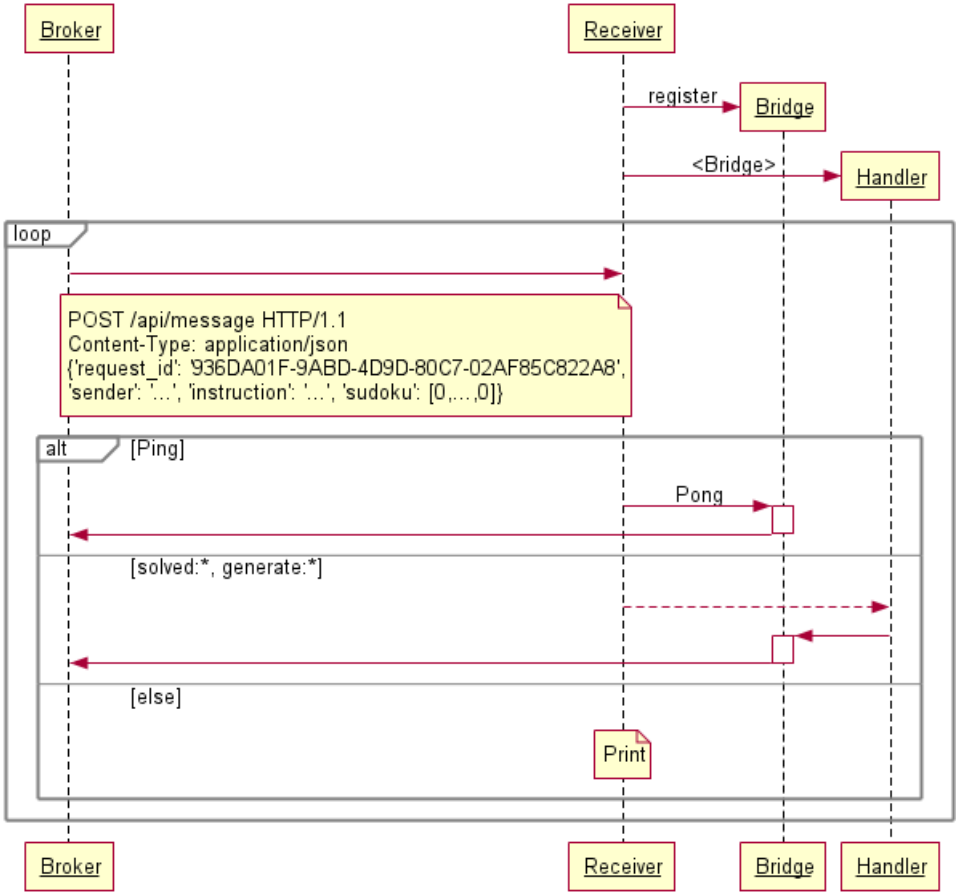
Camel Processor

```
1
2   String [] msgArray = zmqString.split(";");
3
4   String json = Json.createObjectBuilder()
5       .add("request_id", msgArray[0])
6       .add("sender", msgArray[1])
7       .add("instruction", msgArray[2])
8       .add("sudoku", sudokuConverter(msgArray[3]))
9       .build()
10      .toString();
11
```

Empfangen


HTTP/REST
Camel
Receiver
Register, Pong
Bridge
Solve, Display
Handler
Generator

Empfangen


Broker
Receiver
register
Bridge
<Bridge>
Handler
loop
POST /api/message HTTP/1.1
Content-Type: application/json
{'request_id': '936DA01F-9ABD-4D9D-80C7-02AF85C822A8',
'sender': '...', 'instruction': '...', 'sudoku': [0,...,0]}
alt
[Ping]
Pong
[solved:*, generate:*]
[else]
Print
Broker
Receiver
Bridge
Handler

Empfangen

```
1   from Message import *
2   from flask import Flask, jsonify, request, json
3
4   app = Flask(__name__)
5
6   @app.route('/api/message', methods=['POST'])
7   def receive():
8       m = Message(request.json['request_id'], request.json['instruction'],
9                   request.json['sudoku'], request.json['sender'])
10      return m.json(), 201
11
12  if __name__ == '__main__':
13      app.run(debug=False,port=80,host='0.0.0.0')
```

Empfangen

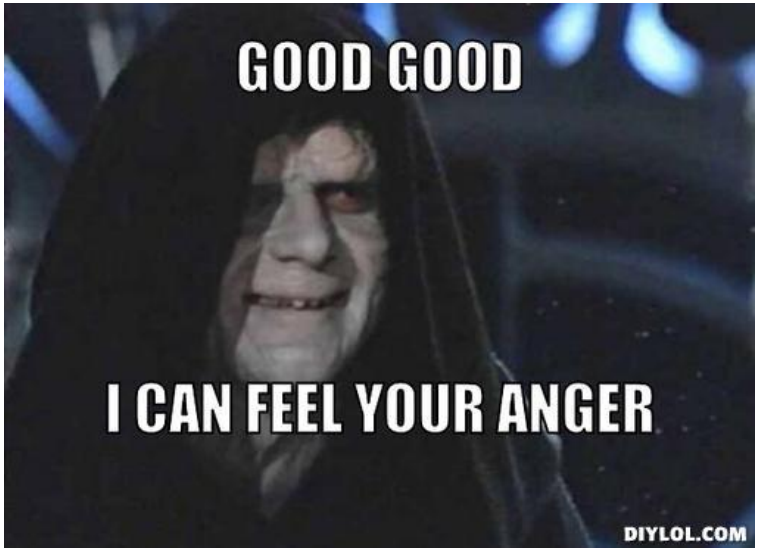
- Camel Rest Endpoint
  - nur als From
- Restlet Endpoint
  - restlet:http://<IP>/api/message?restletMethod=post

Empfangen

```
36  @app.route('/api/message_urlencoded', methods=['POST'])
37  def receive():
38      recvRaw = request.stream.read().decode("utf-8")
39      recvDecoded = urllib.parse.unquote(recvRaw)
40      jsonString = recvDecoded.split('}')[0] + "}"
41      recvJson = json.loads(jsonString)
42      recvMsg = Message(recvJson["request_id"], recvJson["instruction"],
43                        recvJson["sender"], recvJson["sudoku"])
44      return recvMsg.json(), 200
```

Anmerkungen

- Clojure



Clojure

Clojure

```clojure
(ns generator-clojure.generate_init
  (:gen-class))

;; Imports
(import GenInit)

(defn print_sudoku
  "Print a sudoku in correct form"
  [sudoku]
  (doseq [item sudoku]
    (println item)))

(defn generate_row
  [size offset]
  (loop [j 0, l (list)]
    (if (>= j size)
      (into [] (reverse l))
      (recur (+ j 1) (list* (+ (mod (+ j offset) size) 1) l)))))

(defn generate_init_sudoku
  "Generate sudoku"
  [size]
  (let [k (int(Math/sqrt size))]
    (loop [i 0, l (list)]
    (if (>= i size)
      (into [] (reverse l))
      (recur (+ i 1) (list* (generate_row size (+ (* k (mod i k)) (int(/ i k)))) l))))))

(defn swap [v i1 i2]
   (assoc v i2 (v i1) i1 (v i2)))

(defn inner_swap
  [sudoku, size, fst, snd]
  (loop [i 0, s sudoku]
          (if (>= i size)
            s
            (recur (+ i 1) (assoc s i (swap (nth s i) (.indexOf (nth s i) fst)
                                      (.indexOf (nth s i) snd)))))))
```

Clojure

```clojure
(defn swap_numbers
  "swap all numbers in sudoku"
  [sudoku]
  (let [size  (count sudoku), fst (+ (rand-int (- size 1)) 1), snd (+ (rand-int (- size 1)) 1)]
    (loop [done 0, fst (+ (rand-int (- size 1)) 1), snd (+ (rand-int (- size 1)) 1)]
      (if (= fst snd)
        (recur (+ done 0) (fst (+ (rand-int (- size 1)) 1)) (snd (+ (rand-int (- size 1)) 1)))
        (inner_swap sudoku size fst snd)))))

(defn swap_rows
  "swap some rows in sudoku"
  [sudoku]
  (let [k (int(Math/sqrt (count sudoku)))]
    (let [block (rand-int (- k 1))]
      (let [cc1 (* block k), cc2 (+ cc1 (+ (rand-int (- k 2)) 1))]
        (swap sudoku cc1 cc2)))))

(defn inner_block_swap
  [sudoku k b1 b2]
  (loop [i 0, s sudoku]
    (if (>= i k)
      s
      (recur (+ i 1) (swap s (+ (* b1 k) i) (+ (* b2 k) i))))))

(defn swap_blocks
  "swap blocks in a given sudoku"
  [sudoku]
  (let [k (int (Math/sqrt (count sudoku)))]
    (loop [b1 (rand-int (- k 1)), b2 (rand-int (- k 1))]
      (if (= b1 b2)
        (recur b1 (rand-int (- k 1)))
        (inner_block_swap sudoku k b1 b2)))))

(defn rotate_sudoku [m]
  (apply mapv vector m))
```

Architektur