Analyzing Protocol Implementations for Interoperability

Luis Pedrosa Ari Fogel Nupur Kothari
Ramesh Govindan Ratul Mahajan Todd Millstein

http://nsl.cs.usc.edu/Projects/PIC

May 6, 2015 NSDI'15 Oakland, CA







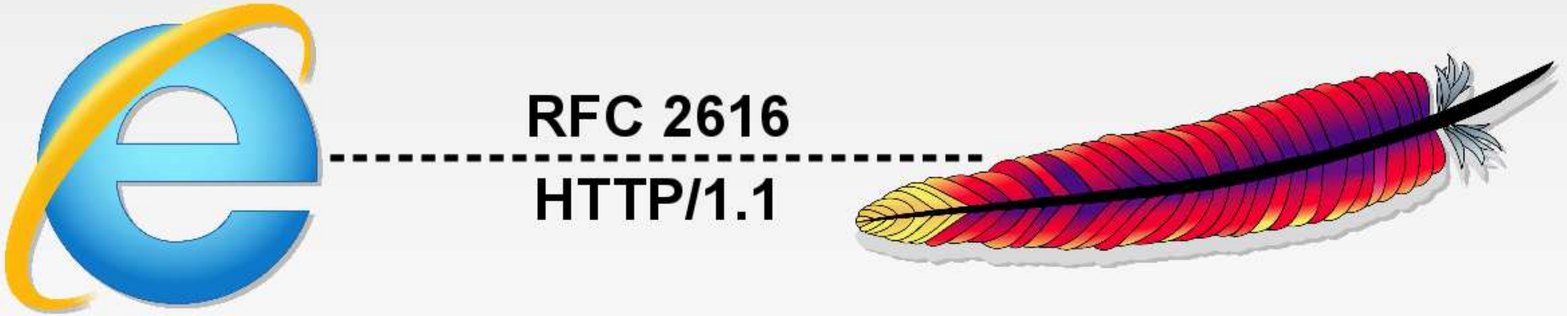


What I Mean by Interoperability

[Protocol] Interoperability

(noun)

The ability of two or more systems to successfully communicate in accordance with an agreed upon protocol.



What I Mean by Interoperability

[Protocol] Interoperability

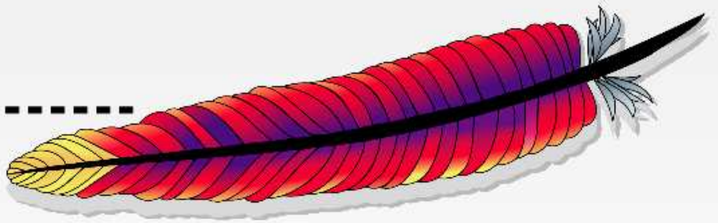
(noun)

Interoperability is crucial for reliability and correctness in networked systems!

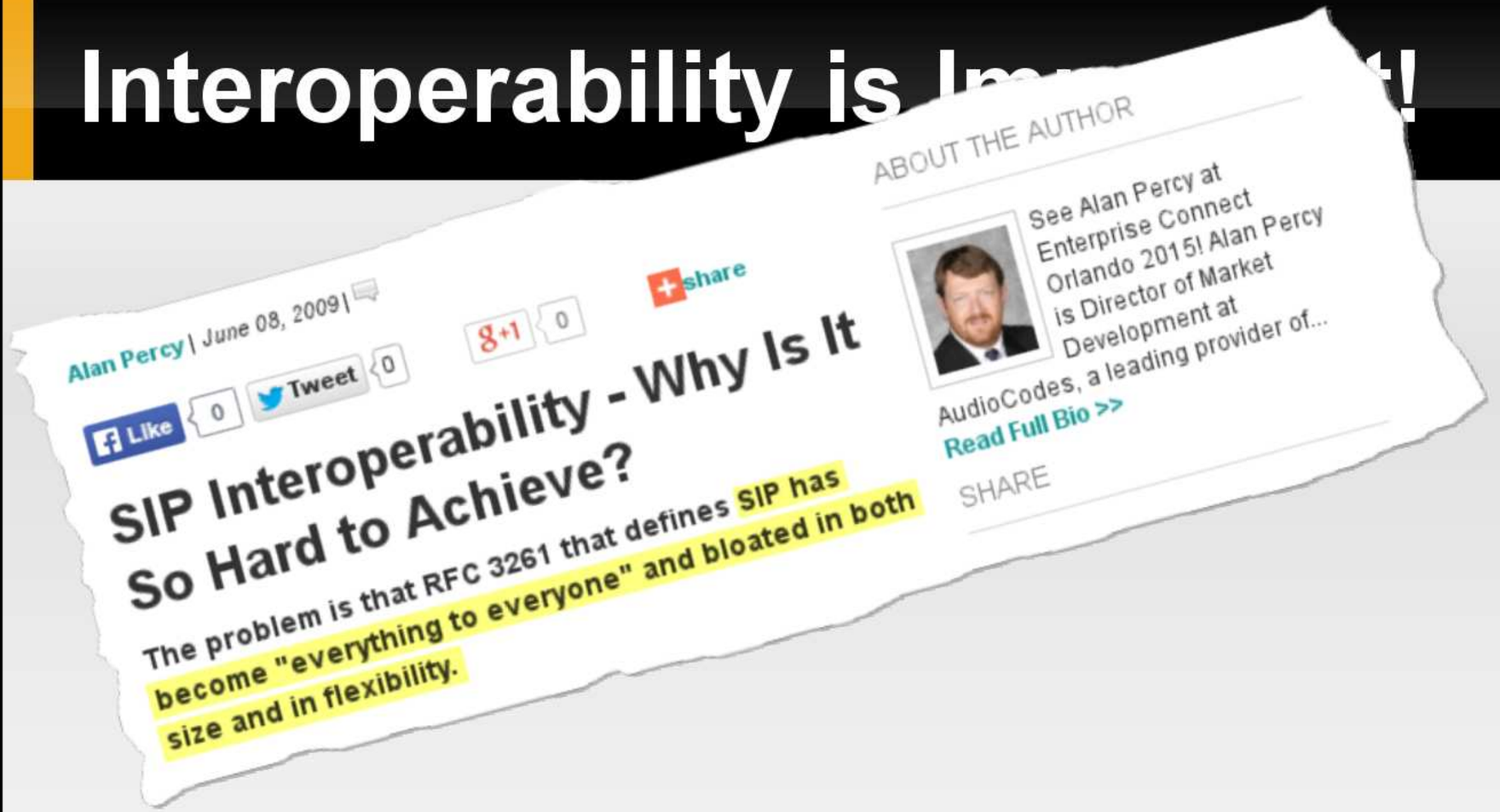


RFC 2616

HTTP/1.1



Interoperability is Im t!



Alan Percy | June 08, 2009 |

Like 0 Tweet 0 +1 0 share

SIP Interoperability - Why Is It So Hard to Achieve?

The problem is that RFC 3261 that defines SIP has become "everything to everyone" and bloated in both size and in flexibility.

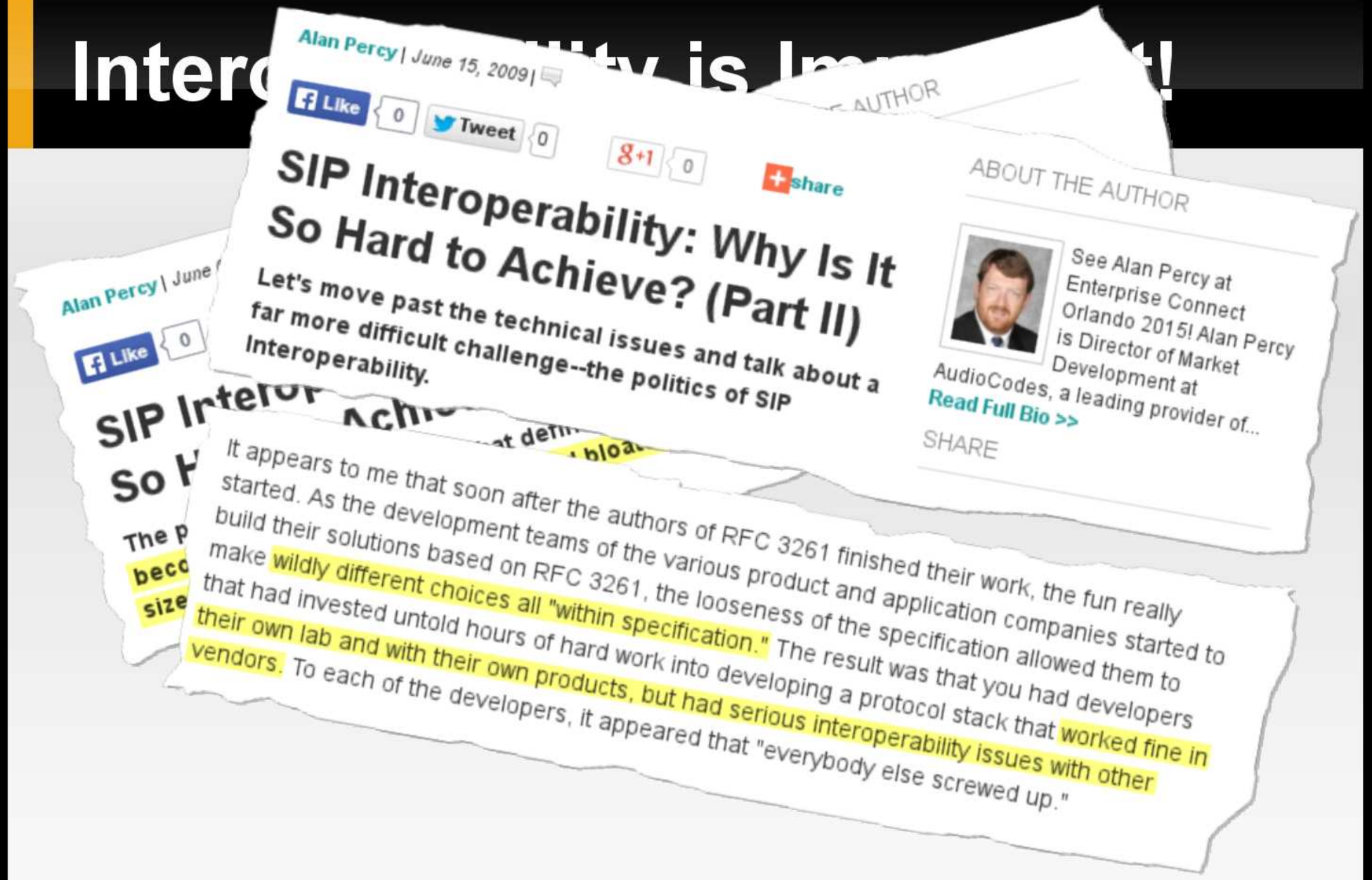
ABOUT THE AUTHOR

See Alan Percy at Enterprise Connect Orlando 2015! Alan Percy is Director of Market Development at AudioCodes, a leading provider of...

Read Full Bio >>

SHARE

Interoperability is Important!



Interoperability is Important!



Interoperability Testing Today



Interoperability Testing Today

Test case:

- Participant roles
- Topology
- Protocol interaction
- Inputs





Interoperability Testing Today

Test case:

- Participant roles
- Topology
- Protocol interaction
- Inputs



PJSIP

Interoperability Testing Today

Test case:

- Participant roles
- Topology
- Protocol interaction
- Inputs



INVITE



Interoperability Testing Today

Test case:

- Participant roles ✔
- Topology ✔
- Protocol interaction ✔
- Inputs



Interoperability Testing Today

Test case:

- Participant roles ✔
- Topology ✔
- Protocol interaction ✔
- Inputs

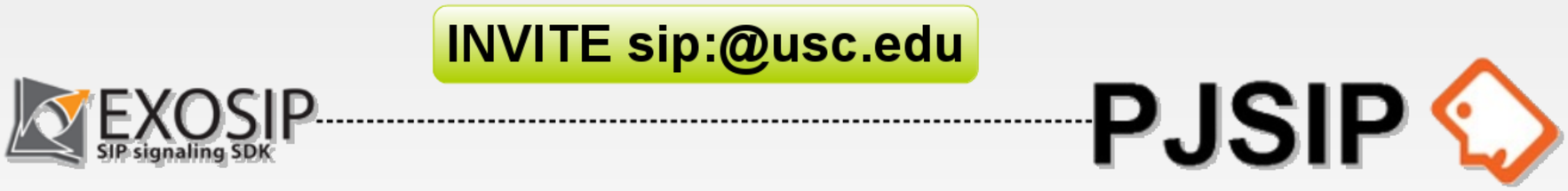


EXOSIP SIP signaling SDK

INVITE

PJSIP

Interoperability Testing Today



Interoperability Testing Today

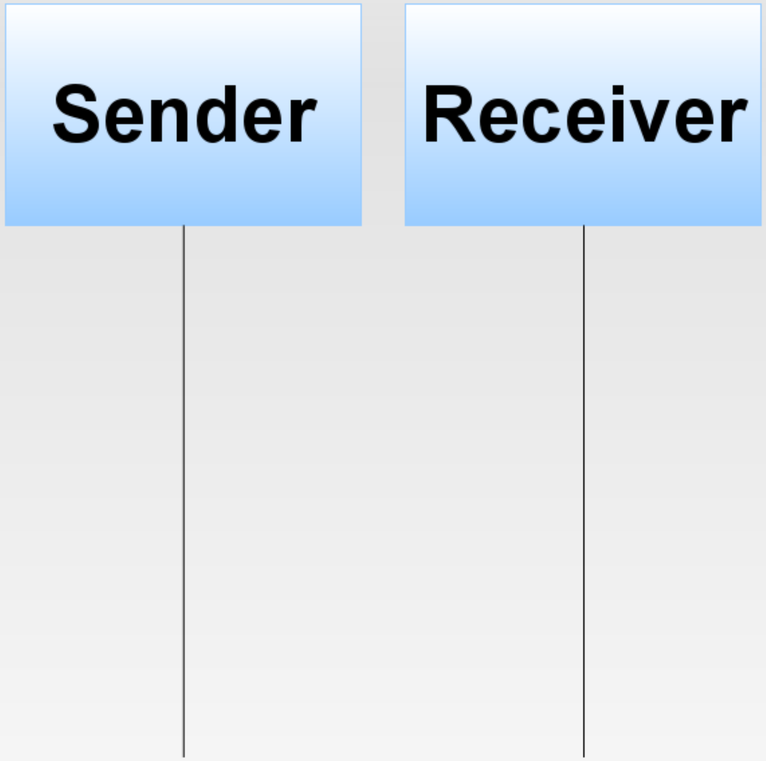
Manual testing is not enough!

- Human intensive
- Not comprehensive

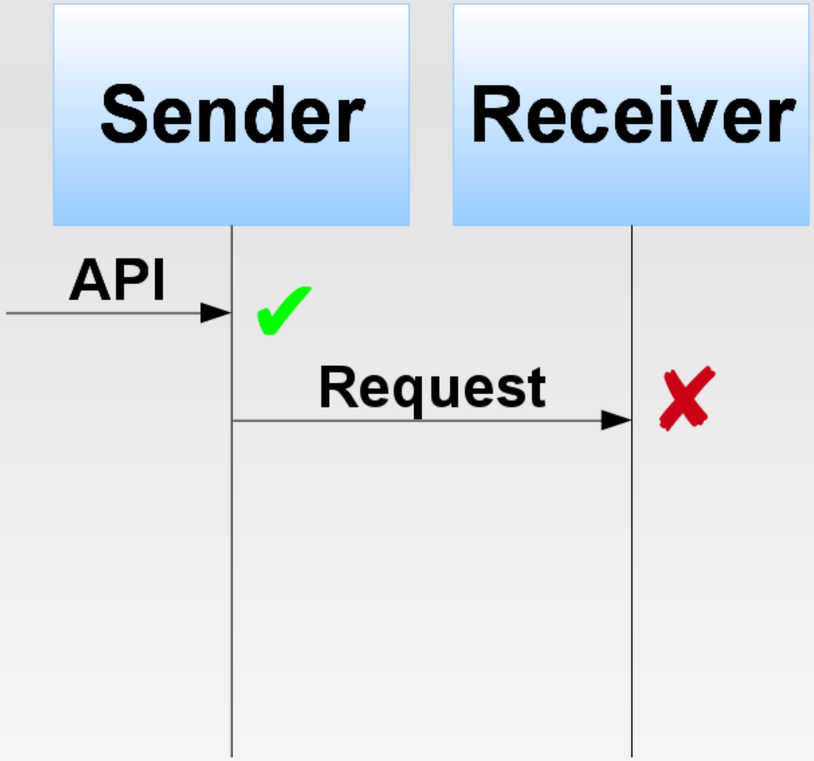


We should automate the search for inputs that will lead to non-interoperabilities

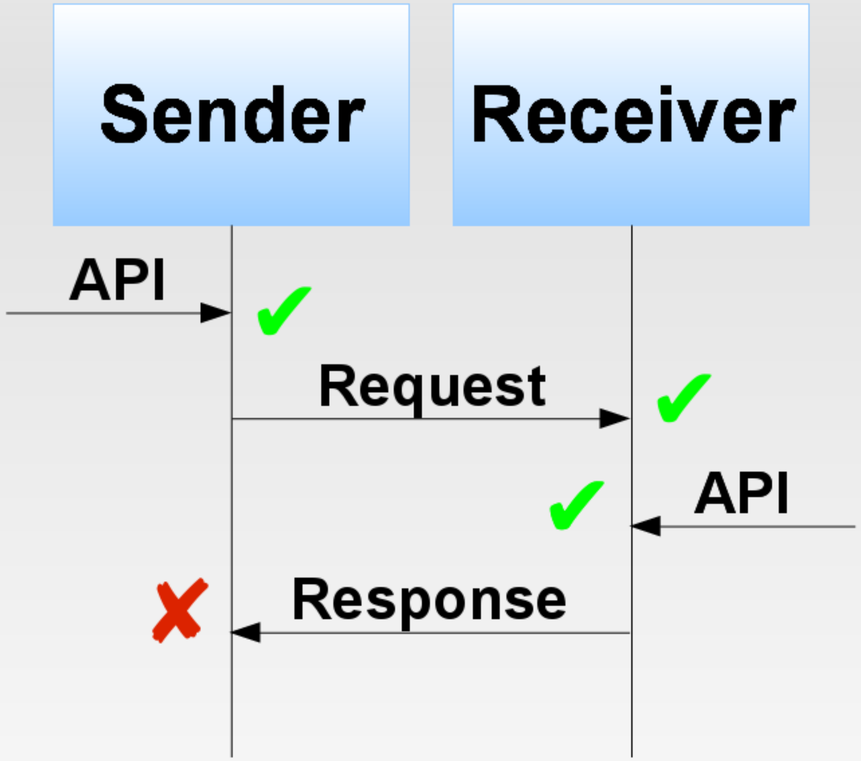
Defining Interoperability



Defining Interoperability



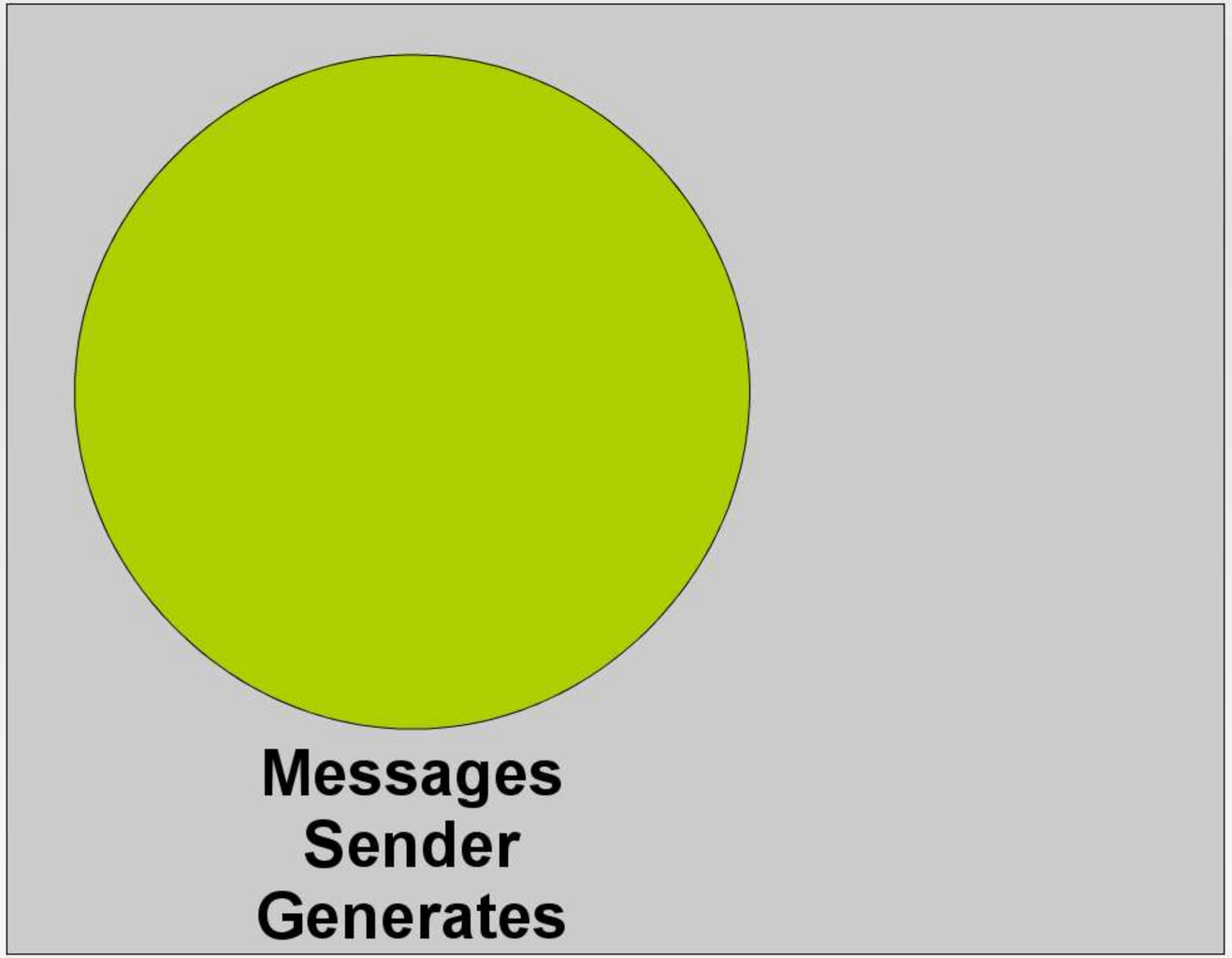
Defining Interoperability



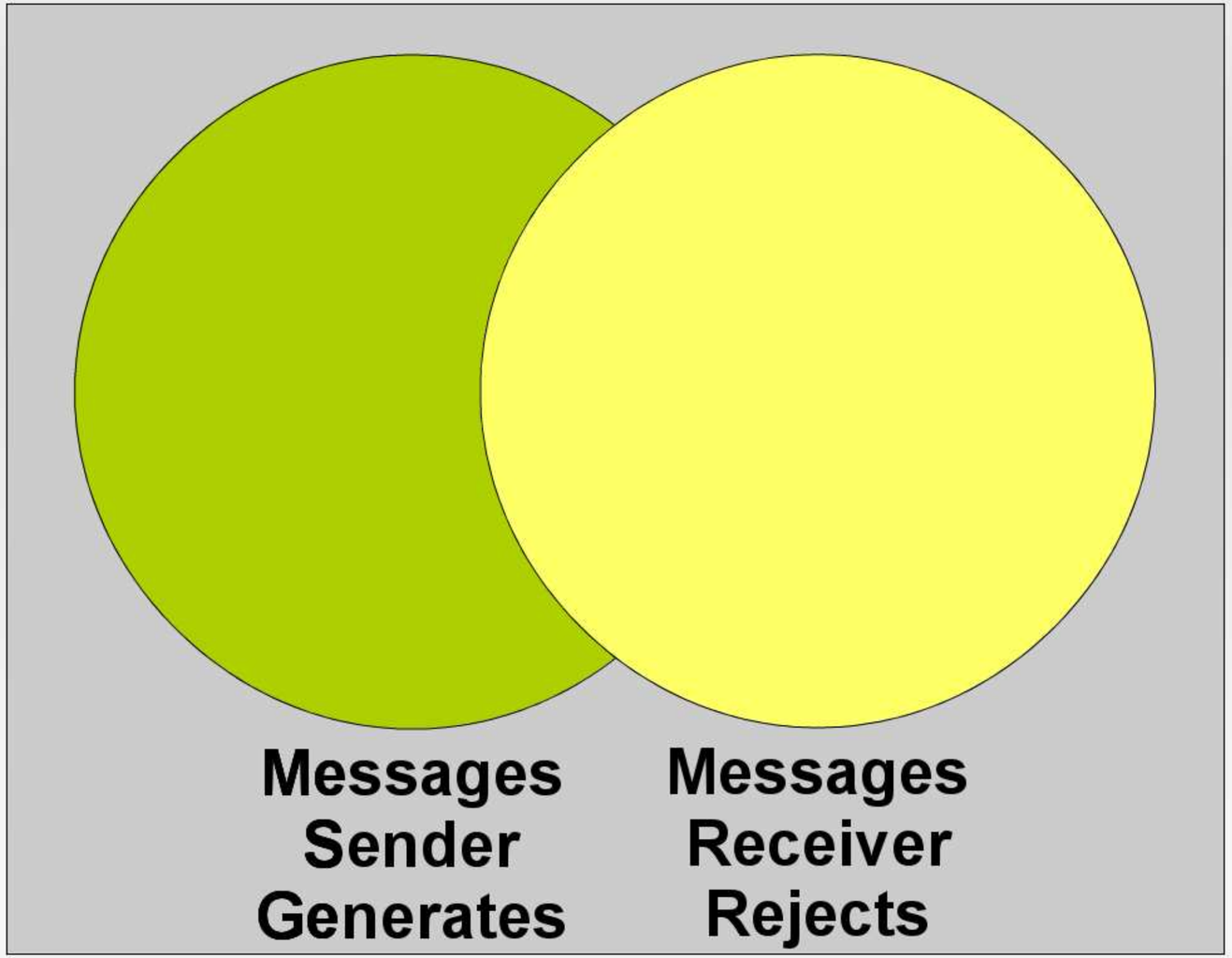
Defining Interoperability



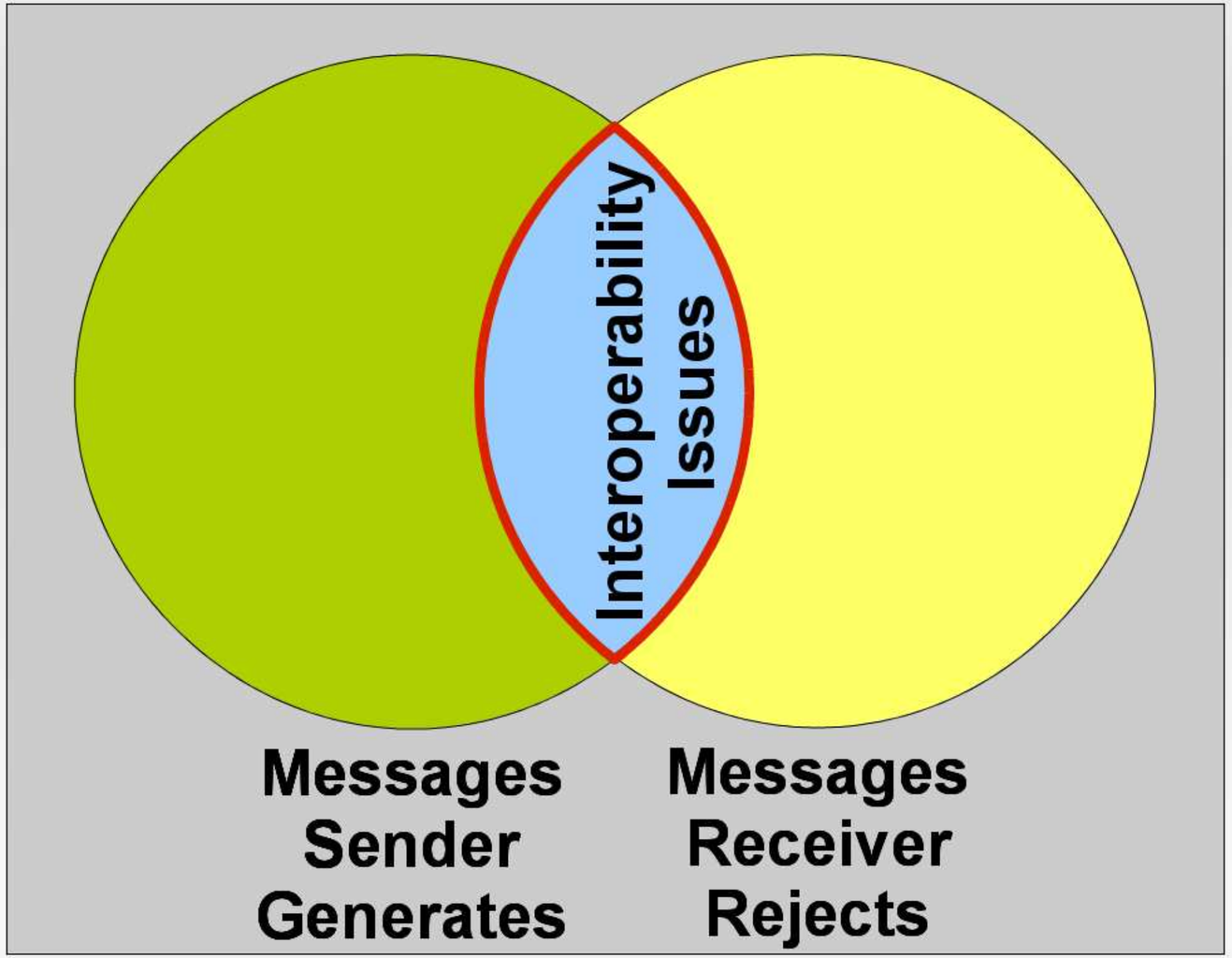
Defining Interoperability



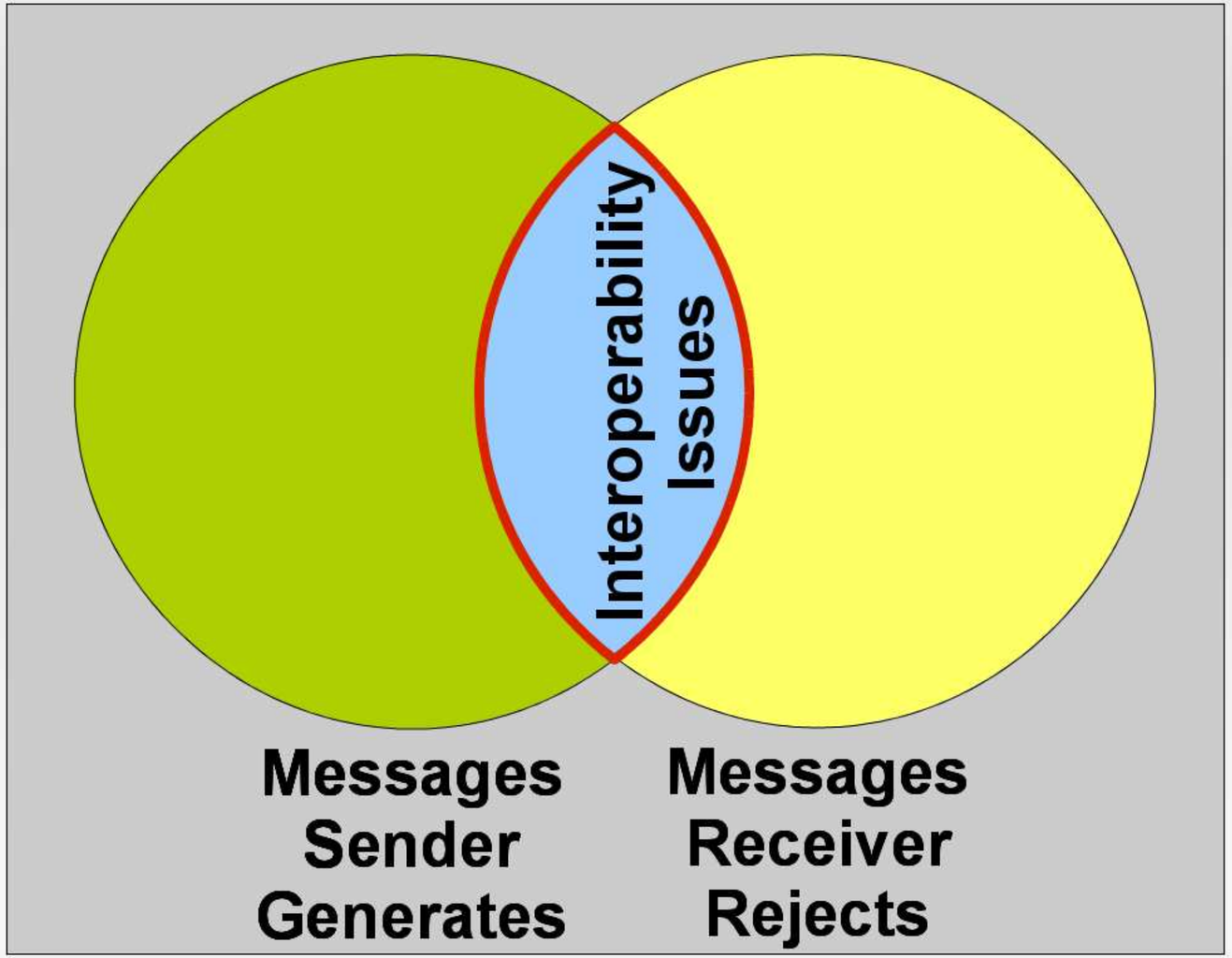
Defining Interoperability



Defining Interoperability



Defining Interoperability



Contributions

- Automate finding non-interoperabilities
  - Use program analysis to map message space

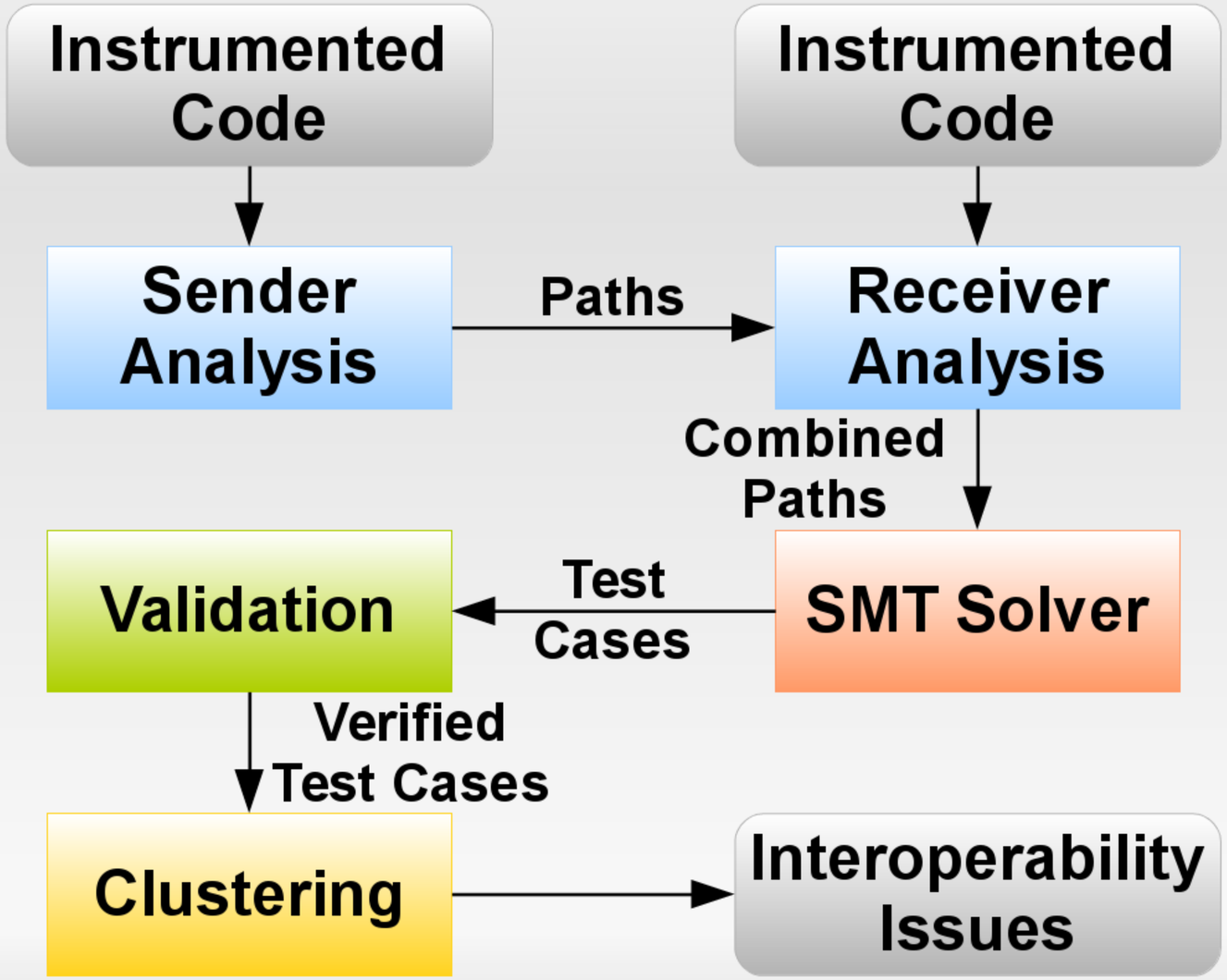
Contributions

- Automate finding non-interoperabilities
  - Use program analysis to map message space
- Scale to real-world protocols
  - Guide and prune the exploration carefully
  - Leverage common constructs in protocol code

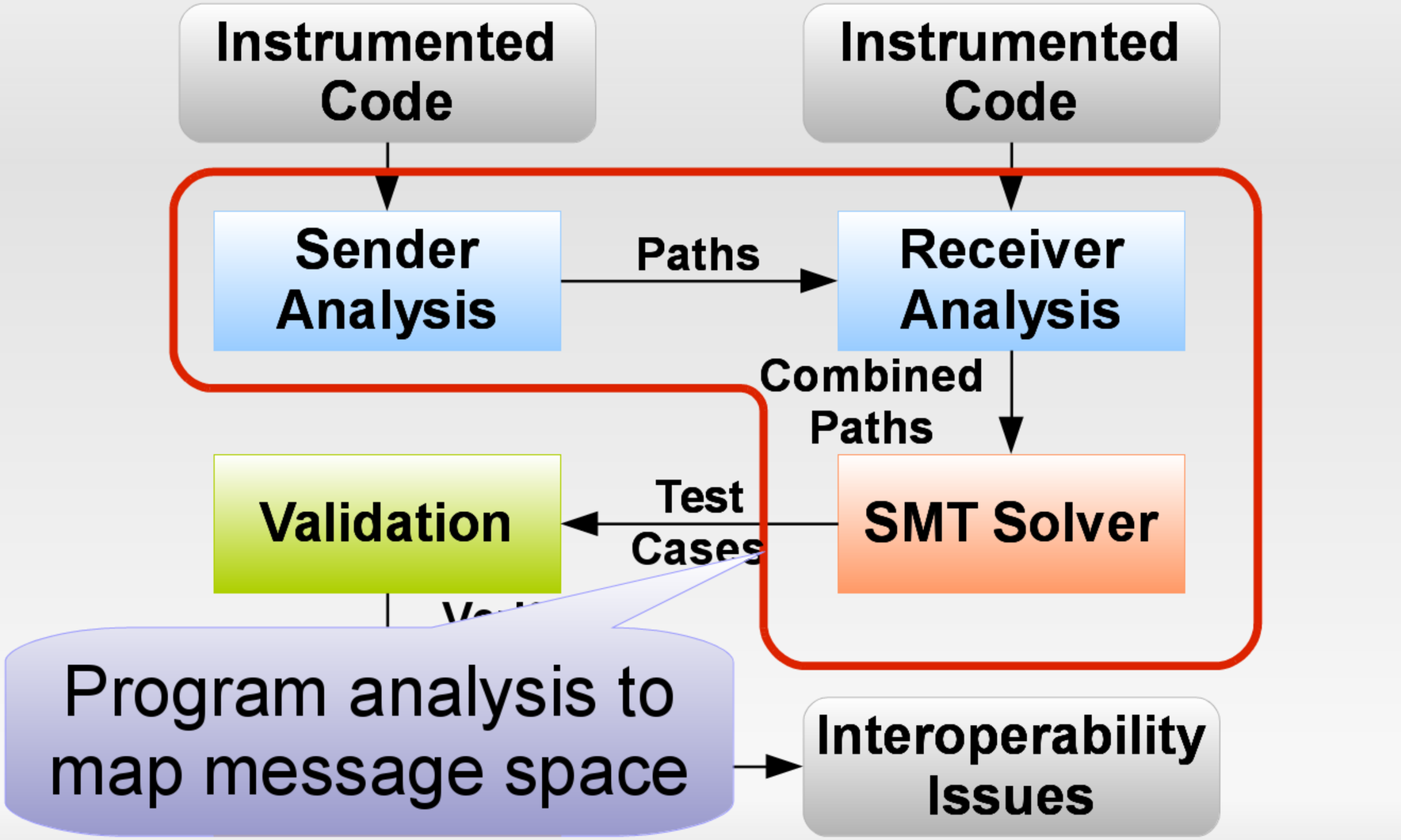
Contributions

- Automate finding non-interoperabilities
  - Use program analysis to map message space
- Scale to real-world protocols
  - Guide and prune the exploration carefully
  - Leverage common constructs in protocol code
- Find real interoperability issues
  - Evaluated on SPDY and SIP implementations
  - Found and reported ~20 issues

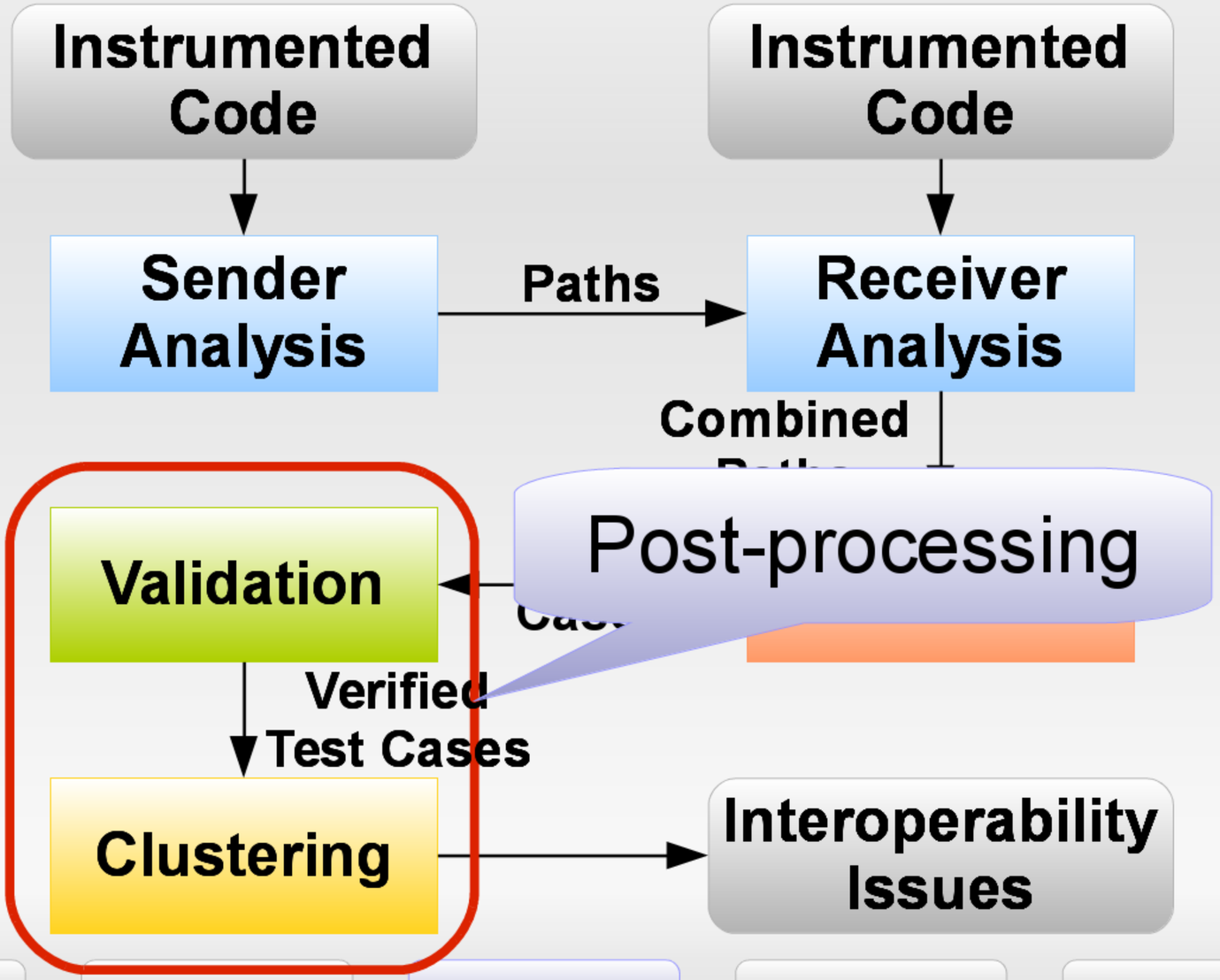
Protocol Interoperability Checker



Protocol Interoperability Checker



Protocol Interoperability Checker



Example: NetCalc

NetCalc Client

```
 1: void compute(char op,
 2:              int arg1,
 3:              int arg2) {
 4:   byte msg[9];
 5:
 6:   if (op == '+')
 7:     msg[0] = 0;
 8:   else if (op == '-')
 9:     msg[0] = 1;
10:   else throw exception;
11:
12:   msg[1..4] = arg1;
13:   msg[5..8] = arg2;
14:
15:   send(msg);
16: }
```

NetCalc Server

```
 1: int handleMsg(byte[] query) {
 2:   int arg1 = query[1..4];
 3:   int arg2 = query[5..8];
 4:   int reply;
 5:
 6:   switch (query[0]) {
 7:     case 0:
 8:       reply = arg1 + arg2;
 9:       break;
10:     default:
11:       throw exception;
12:   }
13:
14:   return reply;
15: }
```

Example: NetCalc

NetCalc Client

```
 1: void compute(char op,
 2:              int arg1,
 3:              int arg2) {
 4:   byte msg[9];
 5:
 6:   if (op == '+')
 7:     msg[0] = 0;
 8:   else if (op == '-')
 9:     msg[0] = 1;
10:   else throw exception;
11:
12:   msg[1..4] = arg1;
13:   msg[5..8] = arg2;
14:
15:   send(msg);
16: }
```

NetCalc Server

```
 1: int handleMsg(byte[] query) {
 2:   int arg1 = query[1..4];
 3:   int arg2 = query[5..8];
 4:   int reply;
 5:
 6:   switch (query[0]) {
 7:     case 0:
 8:       reply = arg1 + arg2;
 9:       break;
10:     default:
11:       throw exception;
12:   }
13:
14:   return reply;
15: }
```

Symbolic Execution Primer

NetCalc Client

```
 1: void compute(char op,
 2:              int arg1,
 3:              int arg2) {
 4:   byte msg[9];
 5:
 6:   if (op == '+')
 7:     msg[0] = 0;
 8:   else if (op == '-')
 9:     msg[0] = 1;
10:   else throw exception;
11:
12:   msg[1..4] = arg1;
13:   msg[5..8] = arg2;
14:
15:   send(msg);
16: }
```

Symbolic Execution Primer



NetCalc Client

```
 1: void compute(char op,
 2:                int arg1,
 3:                int arg2) {
 4:   byte msg[9];
 5:
 6:   if (op == '+')
 7:     msg[0] = 0;
 8:   else if (op == '-')
 9:     msg[0] = 1;
10:   else throw exception;
11:
12:   msg[1..4] = arg1;
13:   msg[5..8] = arg2;
14:
15:   send(msg);
16: }
```

Path Condition:

Symbolic Store:

$op = \sigma_1$

Symbolic Execution Primer

NetCalc Client

```
 1: void compute(char op,
 2:              int arg1,
 3:              int arg2) {
 4:   byte msg[9];
 5:
 6:   if (op == '+')
 7:     msg[0] = 0;
 8:   else if (op == '-')
 9:     msg[0] = 1;
10:   else throw exception;
11:
12:   msg[1..4] = arg1;
13:   msg[5..8] = arg2;
14:
15:   send(msg);
16: }
```

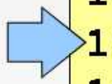
Path Condition:	Symbolic Store:
σ_1 == '+'	op = σ_1 arg1 = σ_2 arg2 = σ_3 msg = Ø

Path Condition:	Symbolic Store:
σ_1 != '+'	op = σ_1 arg1 = σ_2 arg2 = σ_3 msg = Ø

Symbolic Execution Primer

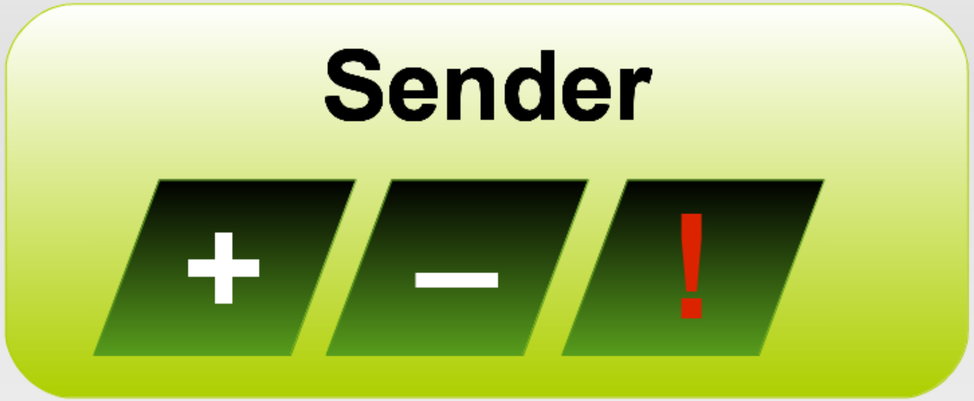
NetCalc Client

```
 1: void compute(char op,
 2:                int arg1,
 3:                int arg2) {
 4:   byte msg[9];
 5:
 6:   if (op == '+')
 7:     msg[0] = 0;
 8:   else if (op == '-')
 9:     msg[0] = 1;
10:   else throw exception;
11:
12:   msg[1..4] = arg1;
13:   msg[5..8] = arg2;
14:
15:   send(msg);
16: }
```

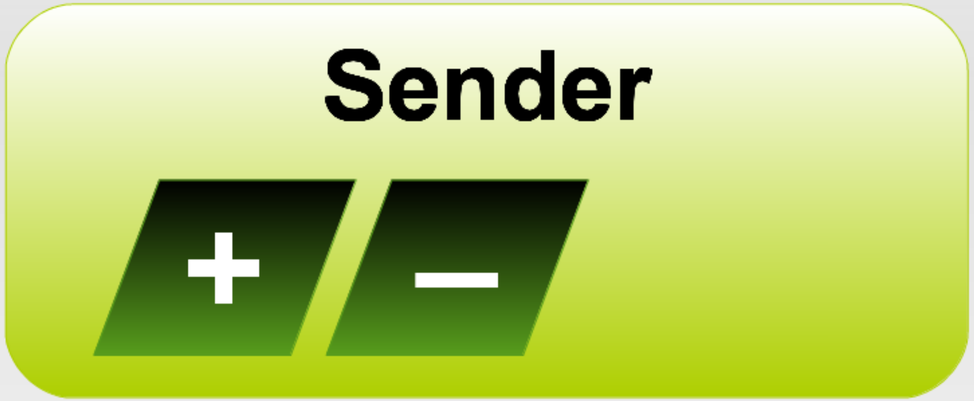


Path Condition:	Symbolic Store:
σ_1 == '+'	op = σ_1
	arg1 = σ_2
	arg2 = σ_3
	msg = 0|σ_2|σ_3

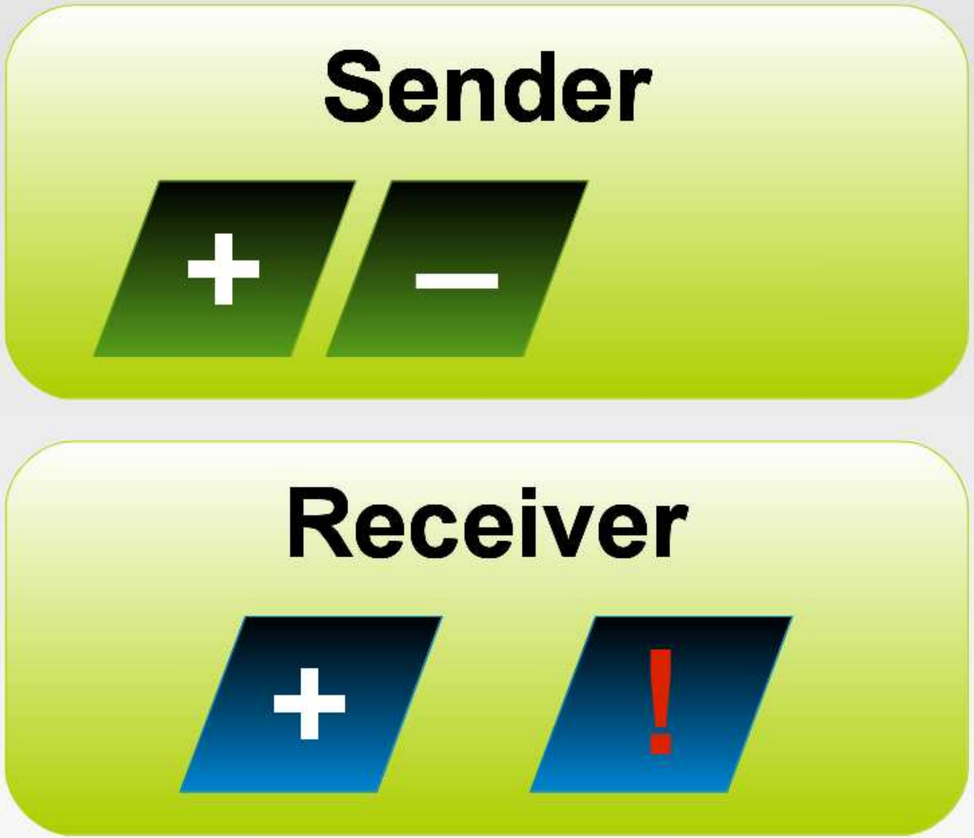
How PIC uses Symbolic Execution



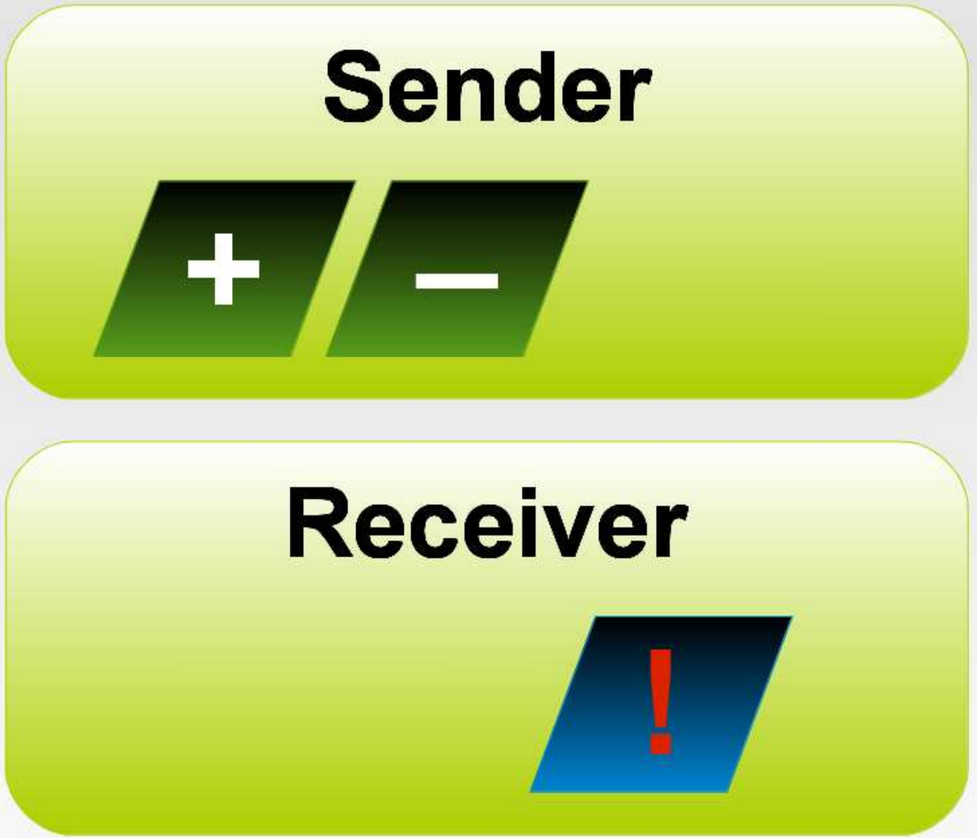
How PIC uses Symbolic Execution



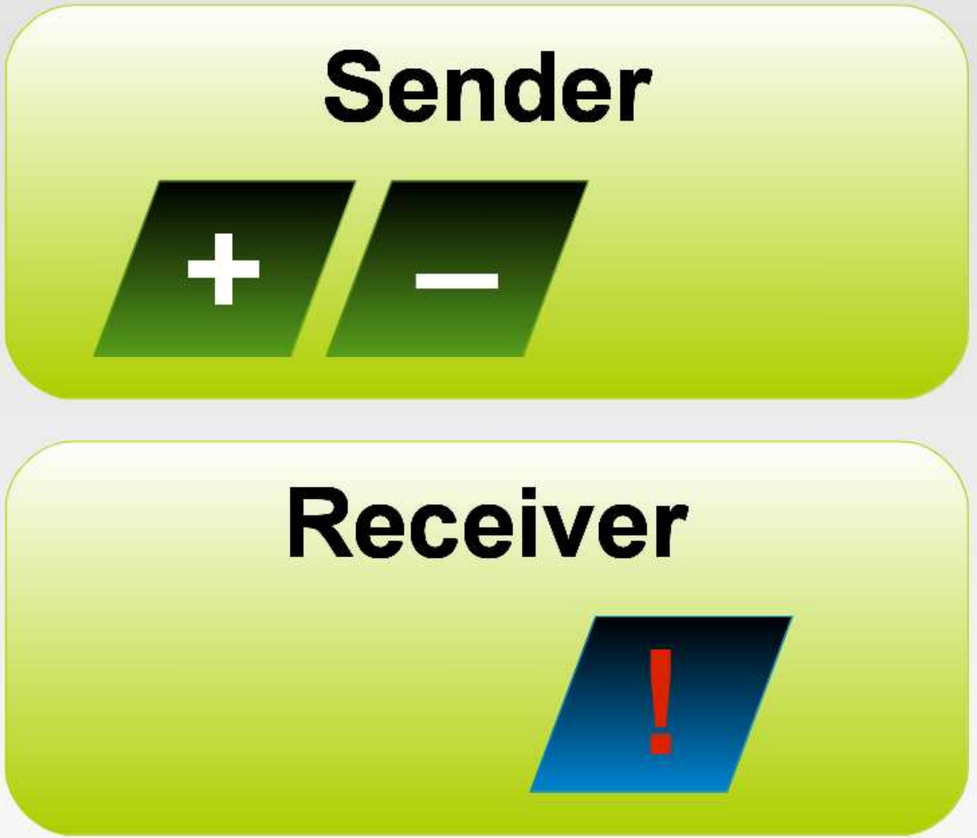
How PIC uses Symbolic Execution



How PIC uses Symbolic Execution



How PIC uses Symbolic Execution



Combining Sender & Receiver



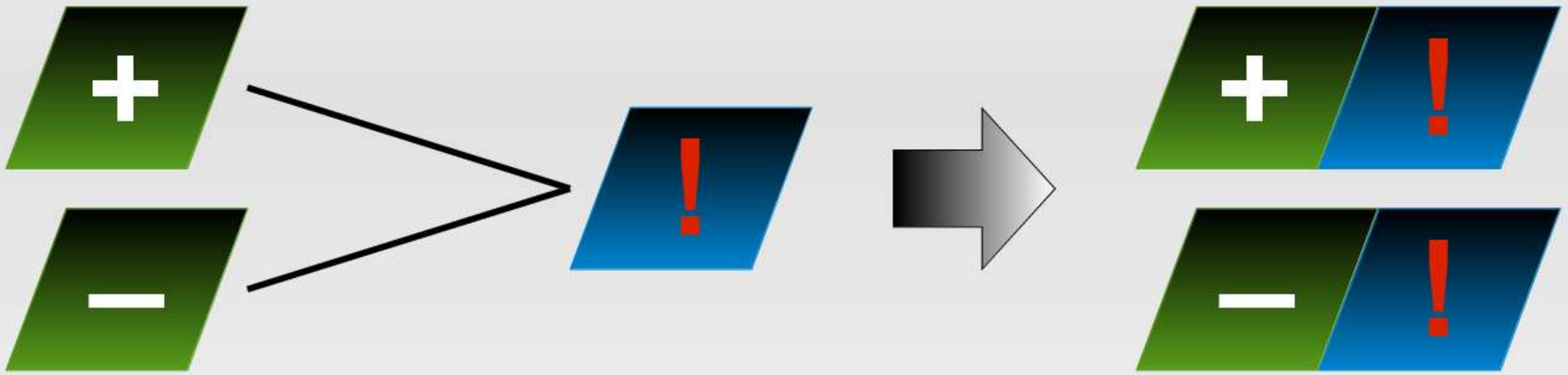




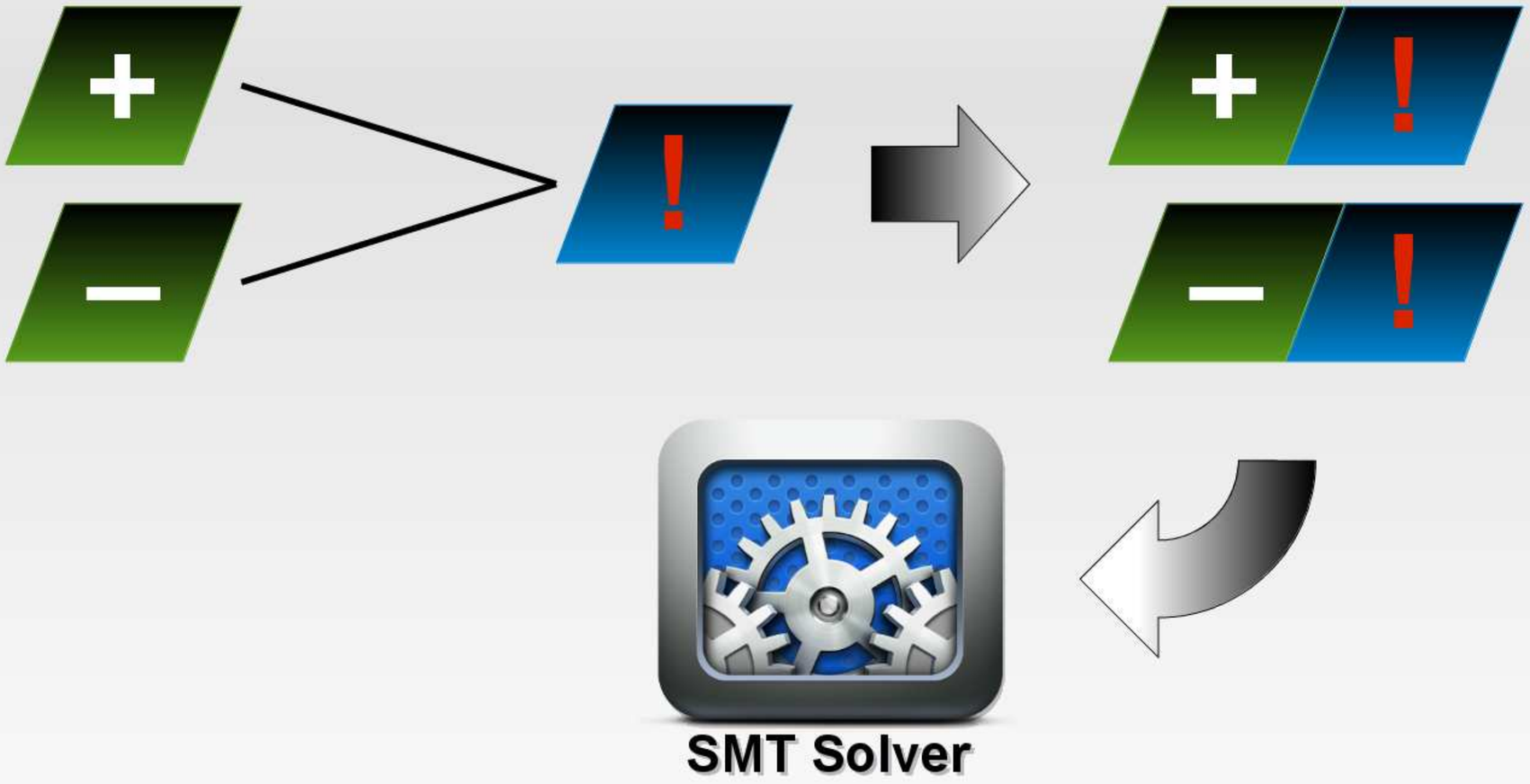
Combining Sender & Receiver



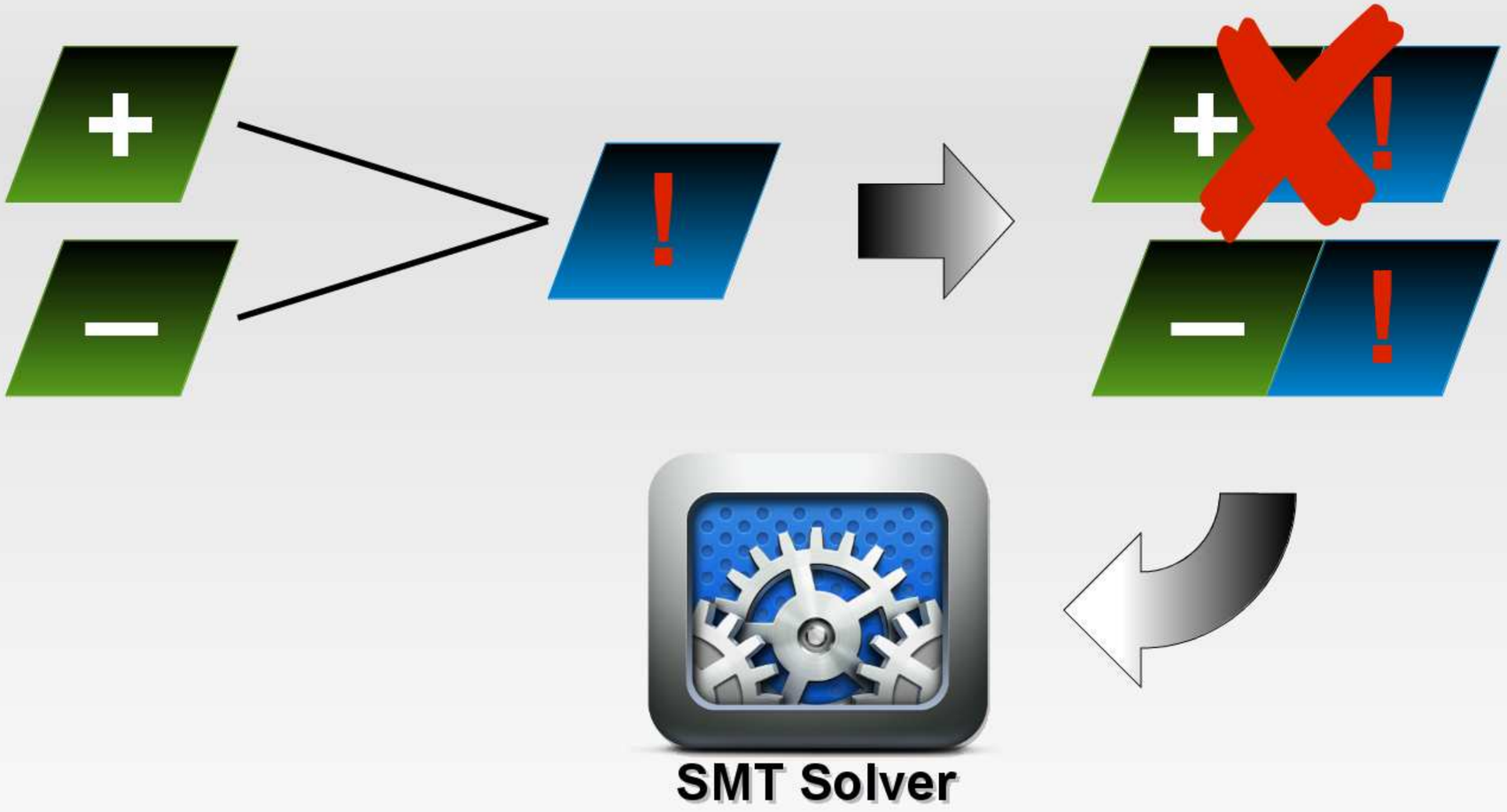
Combining Sender & Receiver



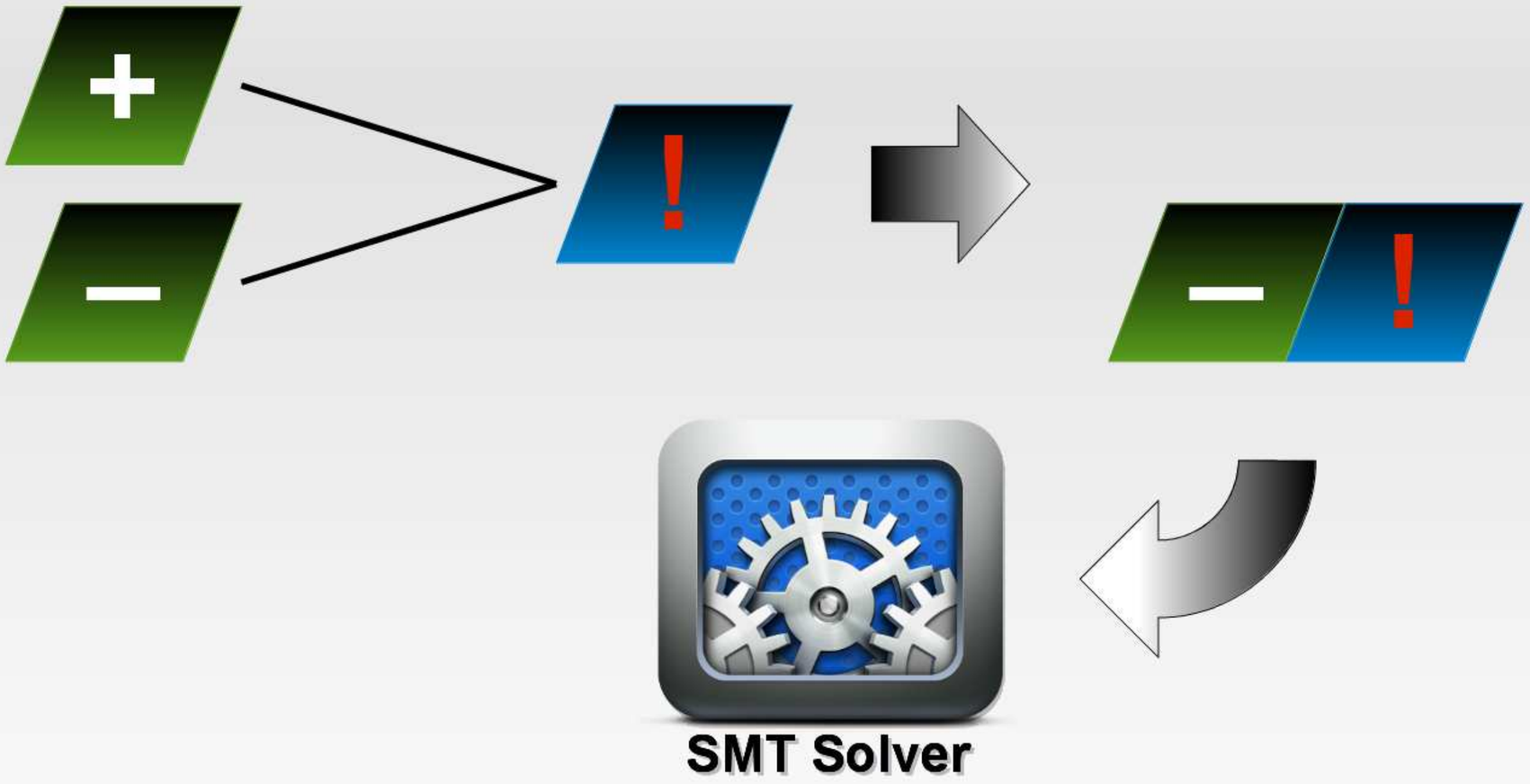
Combining Sender & Receiver



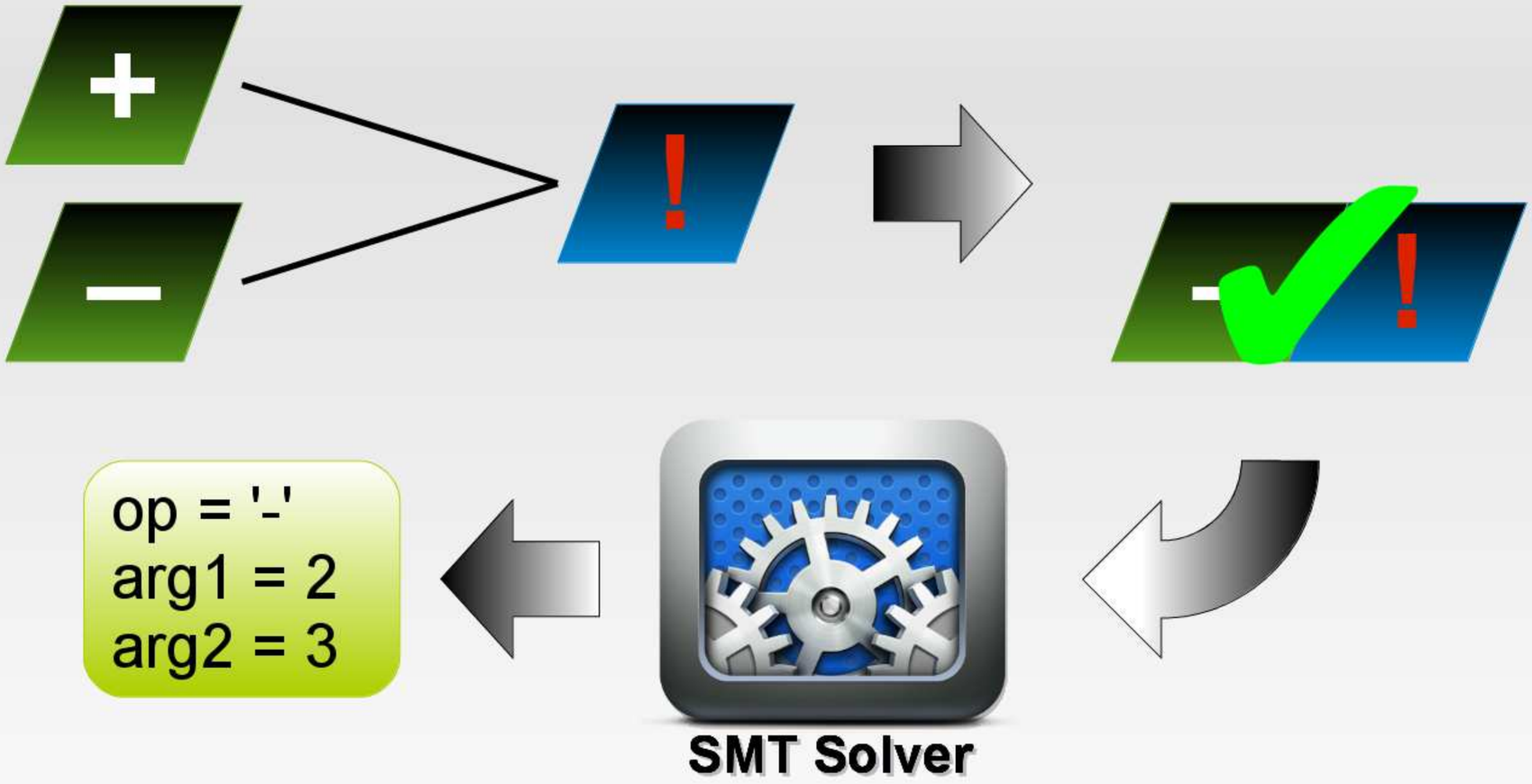
Combining Sender & Receiver



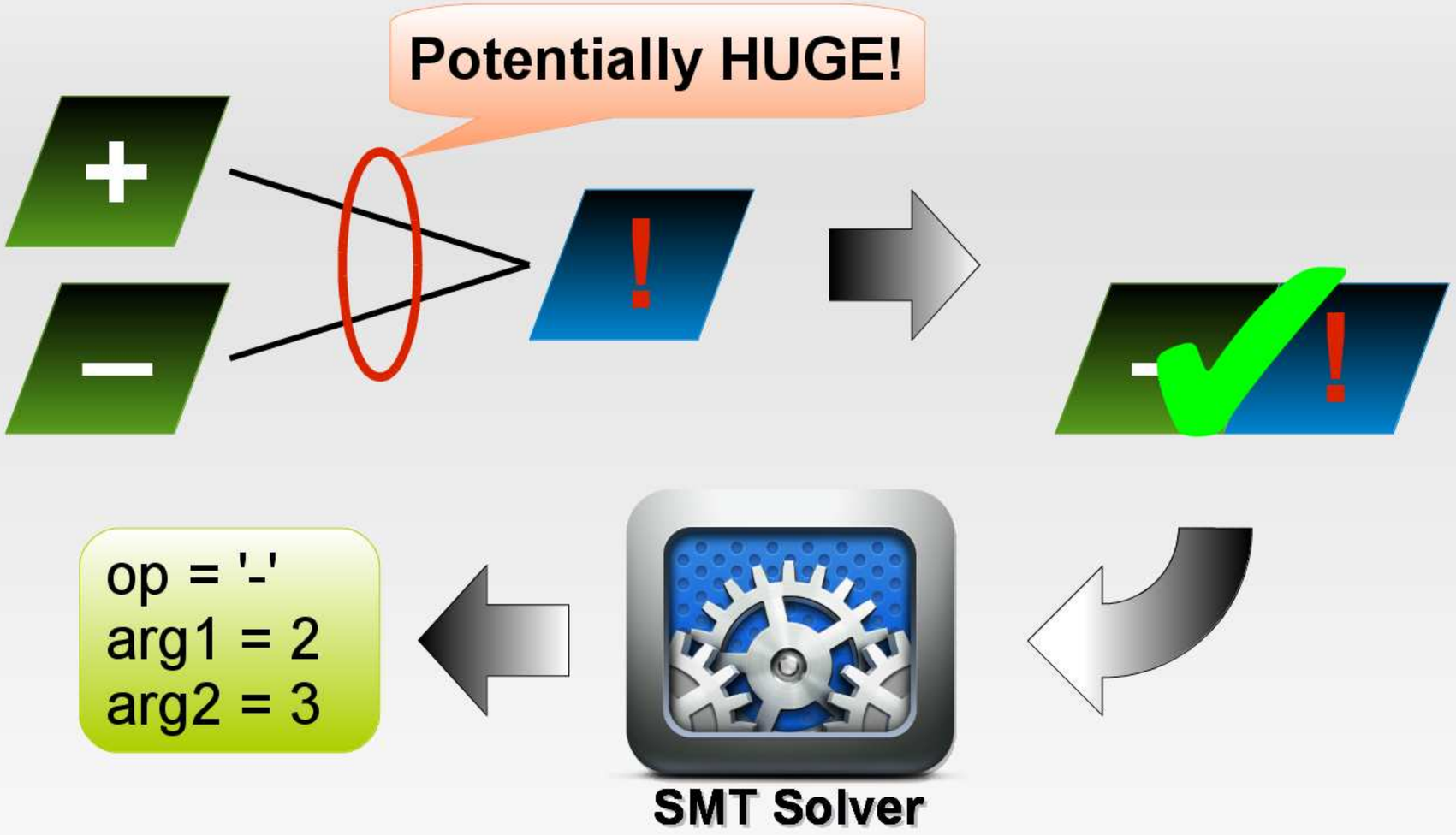
Combining Sender & Receiver



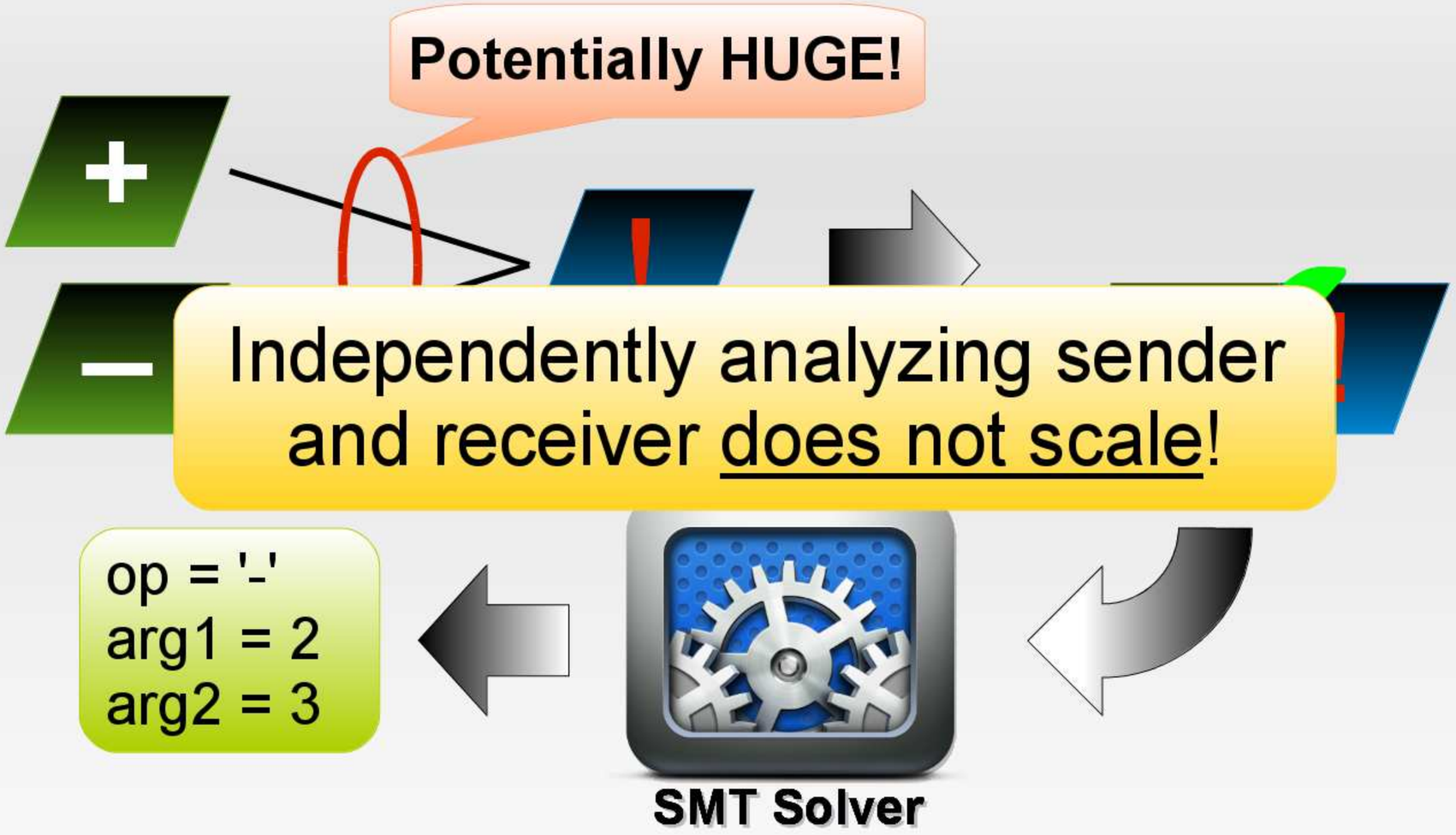
Combining Sender & Receiver



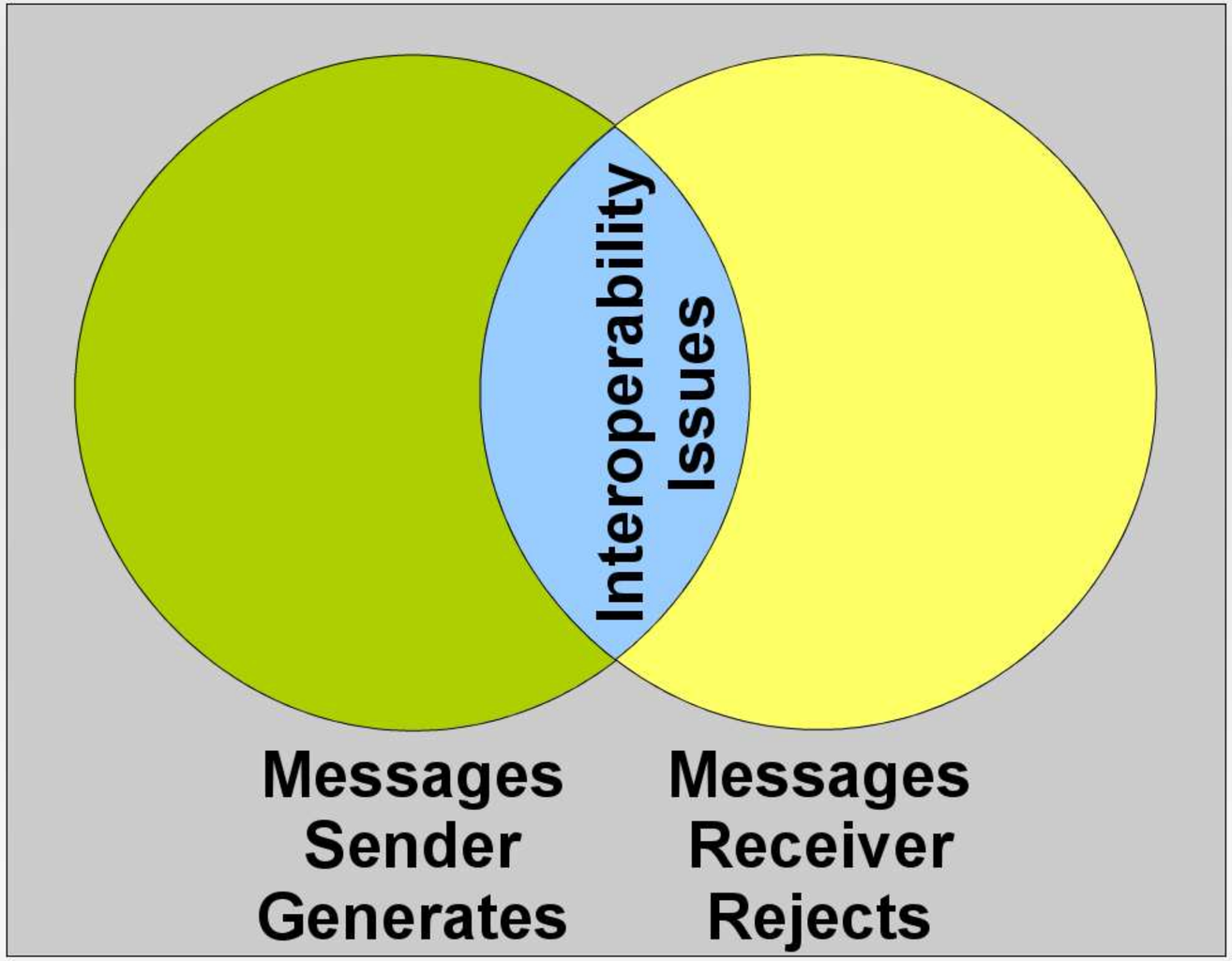
Combining Sender & Receiver



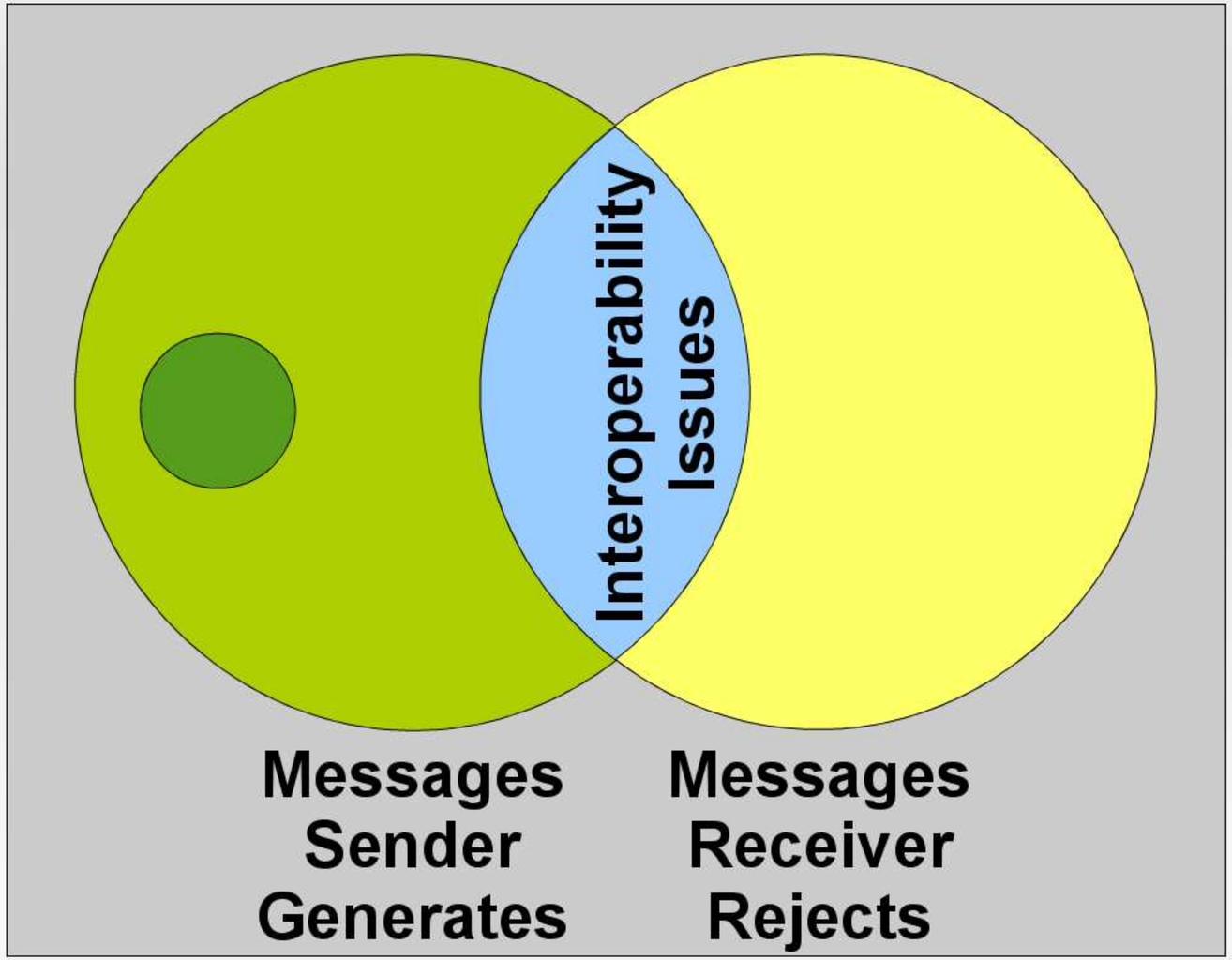
Combining Sender & Receiver



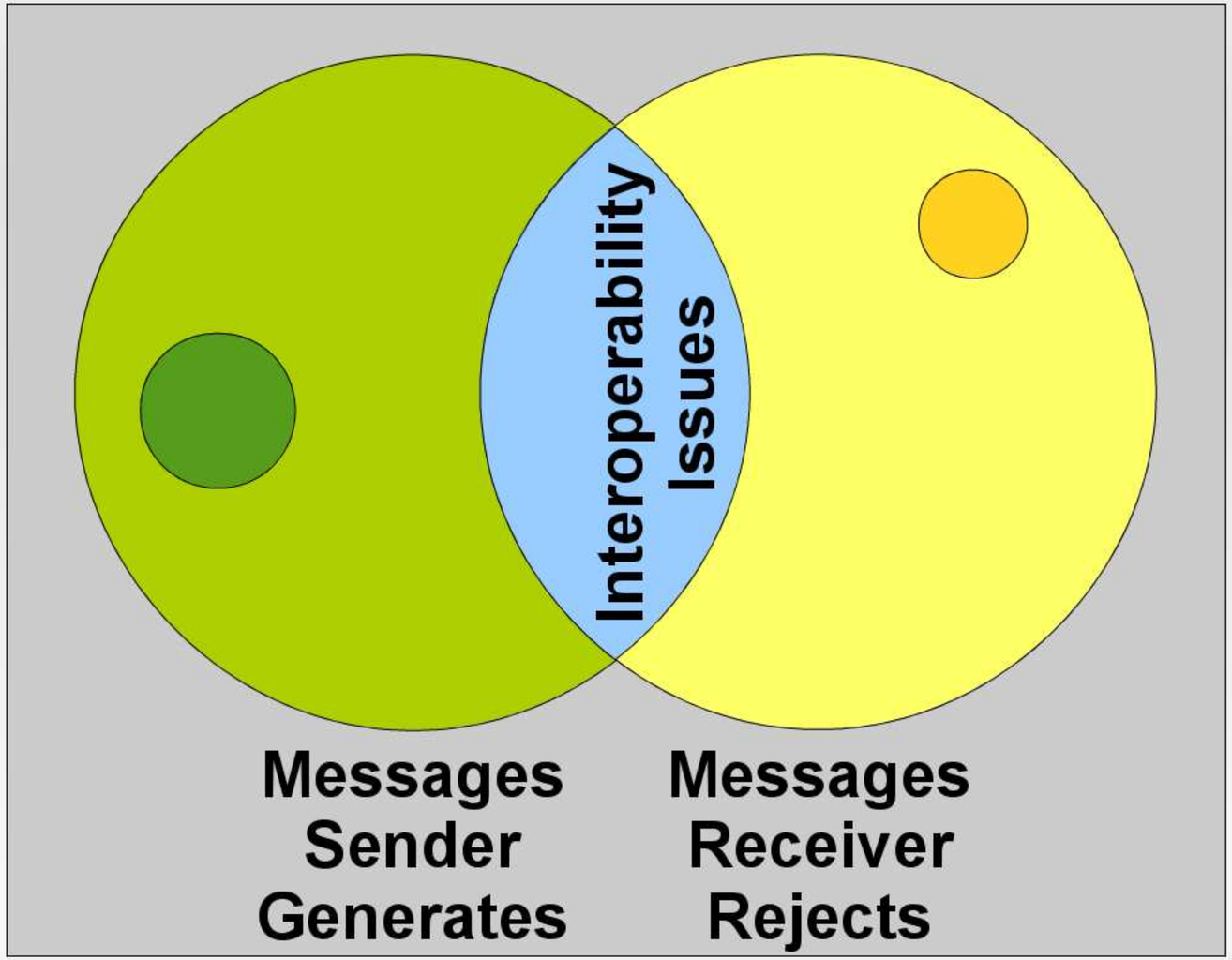
Scaling: Joint Analysis



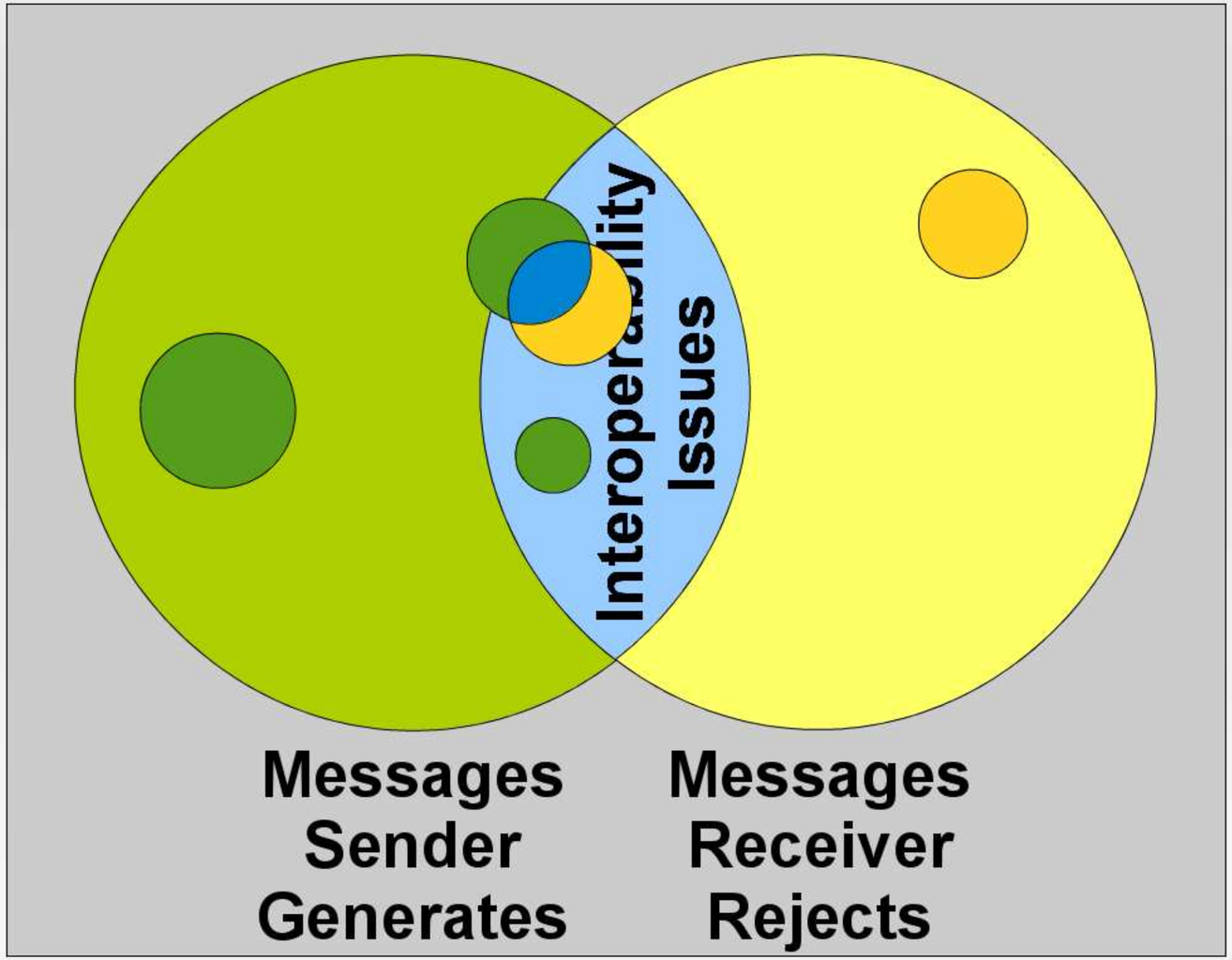
Scaling: Joint Analysis



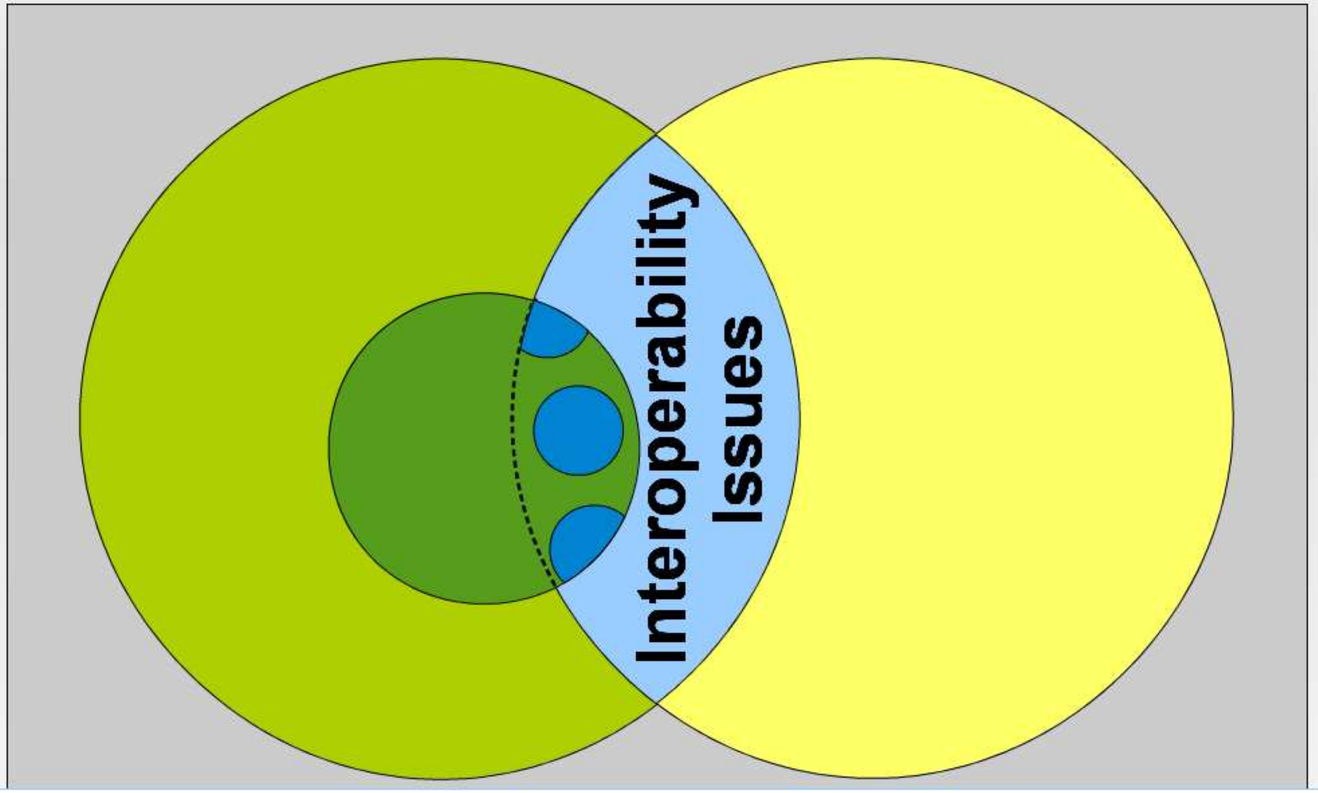
Scaling: Joint Analysis



Scaling: Joint Analysis



Scaling: Joint Analysis



We should scope the receiver-side analysis within the sender context

Joint Symbolic Execution

NetCalc Server

```
 1: int handleMsg(byte[] query) {
 2:   int arg1 = query[1..4];
 3:   int arg2 = query[5..8];
 4:   int reply;
 5:
 6:   switch (query[0]) {
 7:     case 0:
 8:       reply = arg1 + arg2;
 9:       break;
10:     default:
11:       throw exception;
12:   }
13:
14:   return reply;
15: }
```

Joint Symbolic Execution



NetCalc Server

```
 1: int handleMsg(byte[] query) {
 2:   int arg1 = query[1..4];
 3:   int arg2 = query[5..8];
 4:   int reply;
 5:
 6:   switch (query[0]) {
 7:     case 0:
 8:       reply = arg1 + arg2;
 9:       break;
10:     default:
11:       throw exception;
12:   }
13:
14:   return reply;
15: }
```

Joint Symbolic Execution



NetCalc Server

```
 1: int handleMsg(byte[] query) {
 2:   int arg1 = query[1..4];
 3:   int arg2 = query[5..8];
 4:   int reply;
 5:
 6:   switch (query[0]) {
 7:     case 0:
 8:       reply = arg1 + arg2;
 9:       break;
10:     default:
11:       throw exception;
12:   }
13:
14:   return reply;
15: }
```

Joint Symbolic Execution



NetCalc Server

```
 1: int handleMsg(byte[] query) {
 2:   int arg1 = query[1..4];
 3:   int arg2 = query[5..8];
 4:   int reply;
 5:
 6:   switch (query[0]) {
 7:     case 0:
 8:       reply = arg1 + arg2;
 9:       break;
10:     default:
11:       throw exception;
12:   }
13:
14:   return reply;
15: }
```

Joint Symbolic Execution



NetCalc Server

```
 1: int handleMsg(byte[] query) {
 2:   int arg1 = query[1..4];
 3:   int arg2 = query[5..8];
 4:   int reply;
 5:
 6:   switch (query[0]) {
 7:     case 0:
 8:       reply = arg1 + arg2;
 9:       break;
10:     default:
11:       throw exception;
12:   }
13:
14:   return reply;
15: }
```

Joint Symbolic Execution



NetCalc Server

```
 1: int handleMsg(byte[] query) {
 2:   int arg1 = query[1..4];
 3:   int arg2 = query[5..8];
 4:   int reply;
 5:
 6:   switch (query[0]) {
 7:     case 0:
 8:       reply = arg1 + arg2;
 9:       break;
10:     default:
11:       throw exception;
12:   }
13:
14:   return reply;
15: }
```

Joint Symbolic Execution



NetCalc Server

```
1: int handleMsg(byte[] query) {
2:   int arg1 = query[1..4];
3:   int arg2 = query[5..8];
4:   int reply;
10:     default:
11:       throw exception;
12:   }
13:
14:   return reply;
15: }
```

Joint Symbolic Execution enables scaling to large implementations

Other Scaling Challenges

- Challenges with symbolic execution
  - Inherently incomplete; huge search-space
  - Typically tries to explore all paths
  - PIC only cares about a few

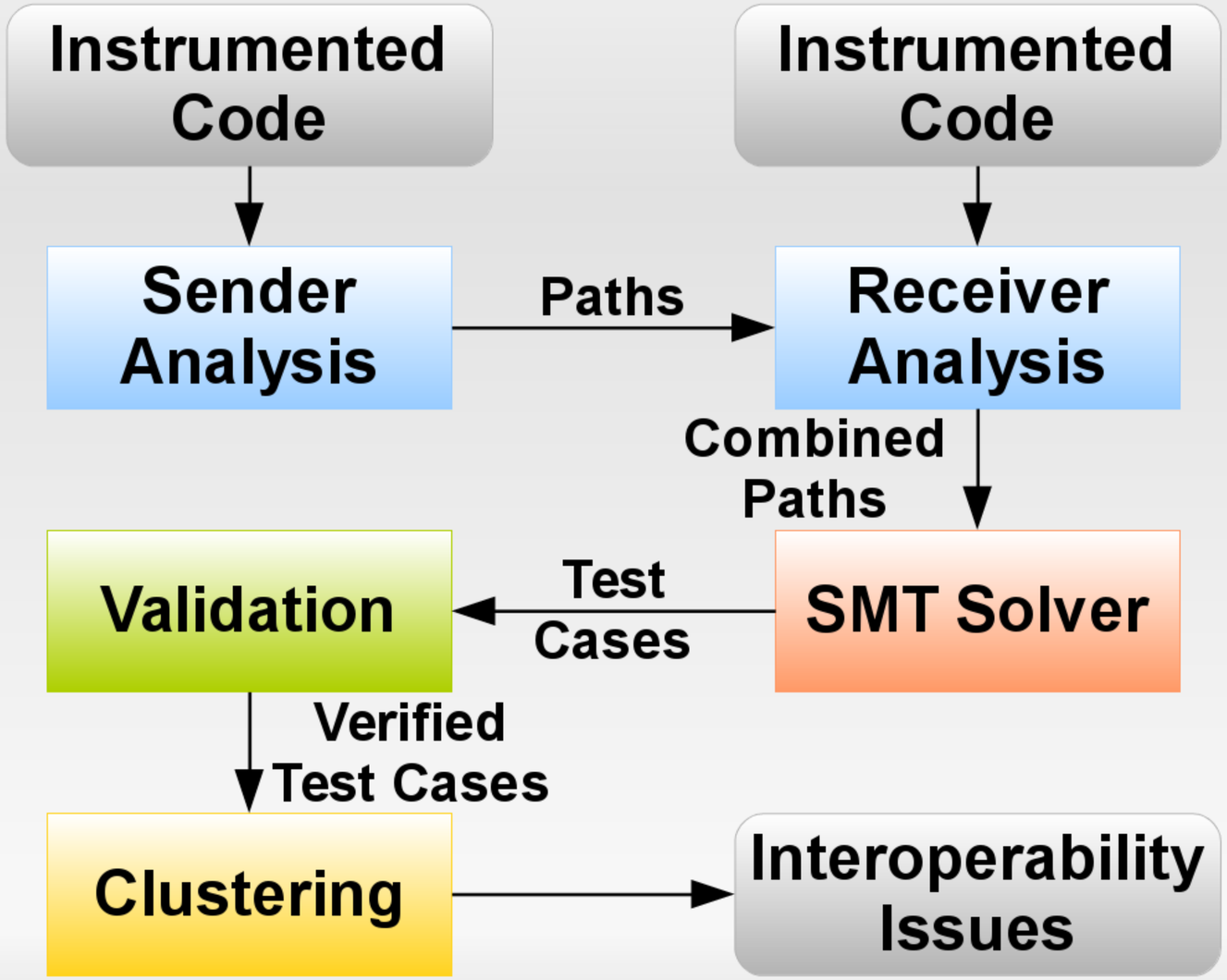
Other Scaling Challenges

- Challenges with symbolic execution
  - Inherently incomplete; huge search-space
  - Typically tries to explore all paths
  - PIC only cares about a few

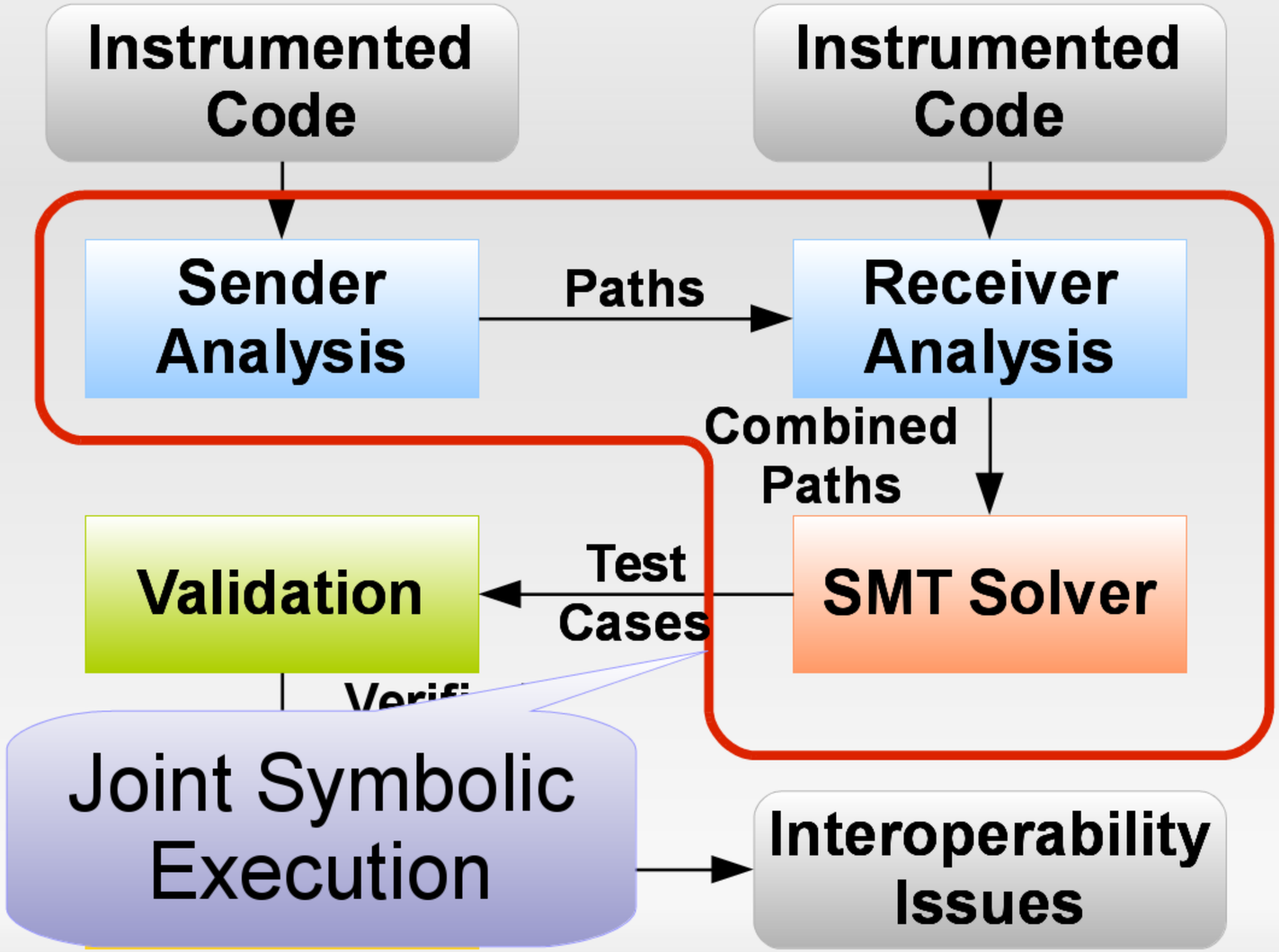
- Solution: Guide and prune search-space
  - Prior art: Directed Symbolic Execution[1]
  - Dispersive Exploration: used in receiver analysis
  - Return Normalization: mitigate local minima

[1] K.-K. Ma, K. Y. Phang, J. S. Foster, M. Hicks. Directed symbolic execution. SAS'11

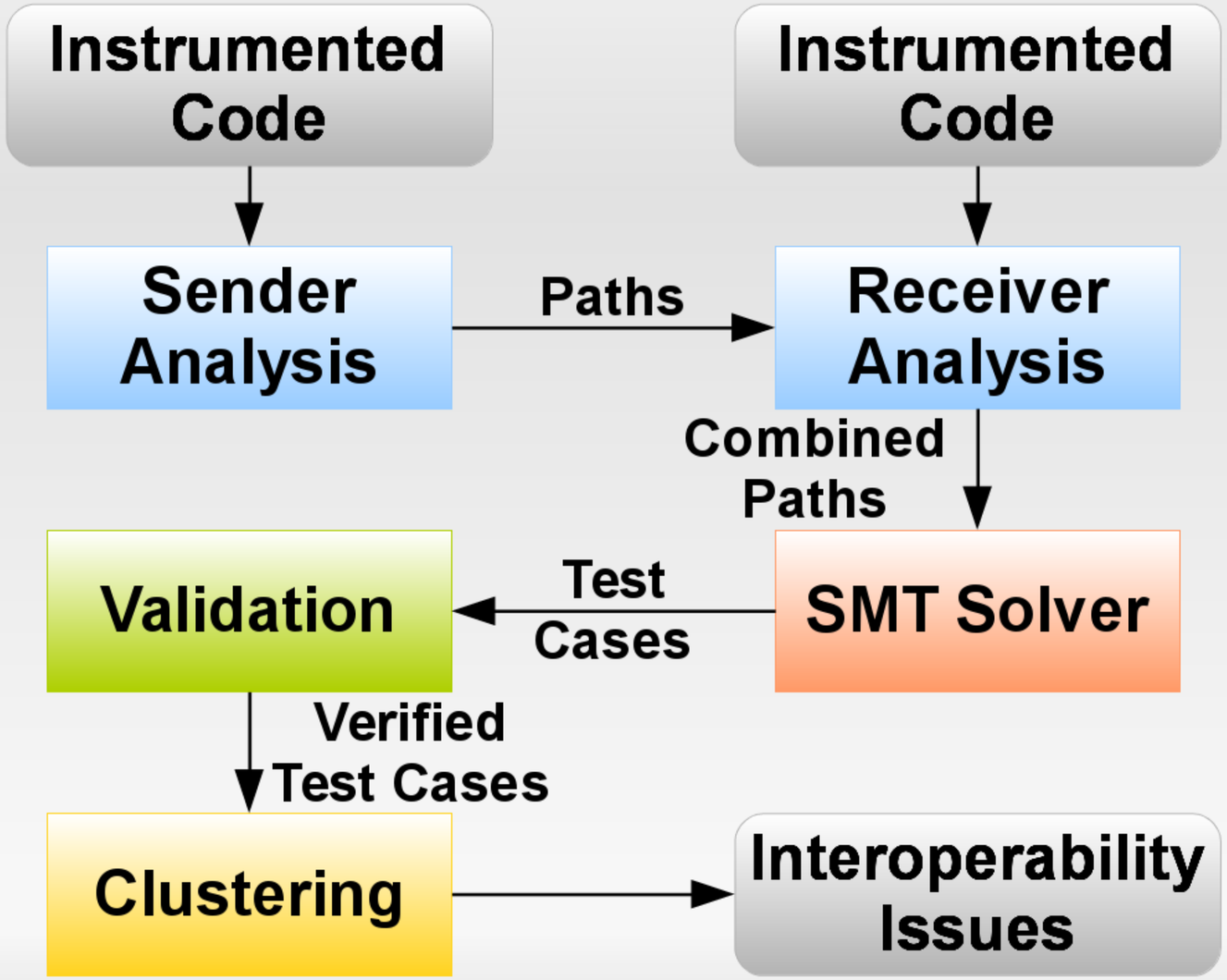
Protocol Interoperability Checker



Protocol Interoperability Checker



Protocol Interoperability Checker



Evaluation

SPDY		
Implementation	**Library**	**Annotations**
spdylay-0.3.7	22739	23
spdylay-1.3.1	25495	23
nginx-1.5.5	99446	4
nginx-1.7.4	104364	4

SIP – Session Initiation Protocol		
Implementation	**Library**	**Annotations**
eXoSIP-3.6.0	43990	38
PJSIP-1.12	83916	21

Evaluation

Implementation	Library	Annotations
eXoSIP-3.6.0	43990	38
PJSIP-1.12	83916	21

PIC can analyze reasonably large code-bases of ~100 kSLOC

The amount of developer input is small with ~10s SLOC

Results: Summary

SPDY	
Cause	**Issues Found**
Liberal Sender	4
Conservative Receiver	4
Bug	1
Optional Feature	2
Unsupported Version	2

SIP – Session Initiation Protocol	
Cause	**Issues Found**
Liberal Sender	8
Optional Feature	1

Example: Liberal Sender



Example: Liberal Sender

URI standard is ambiguous about empty usernames

INVITE sip:@usc.edu



PJSIP

Example: Liberal Sender



Example: Liberal Sender

eXoSIP doesn't check inputs; mangles message

SUBSCRIBE event="\r\n"





Example: Optional Feature



Example: Optional Feature

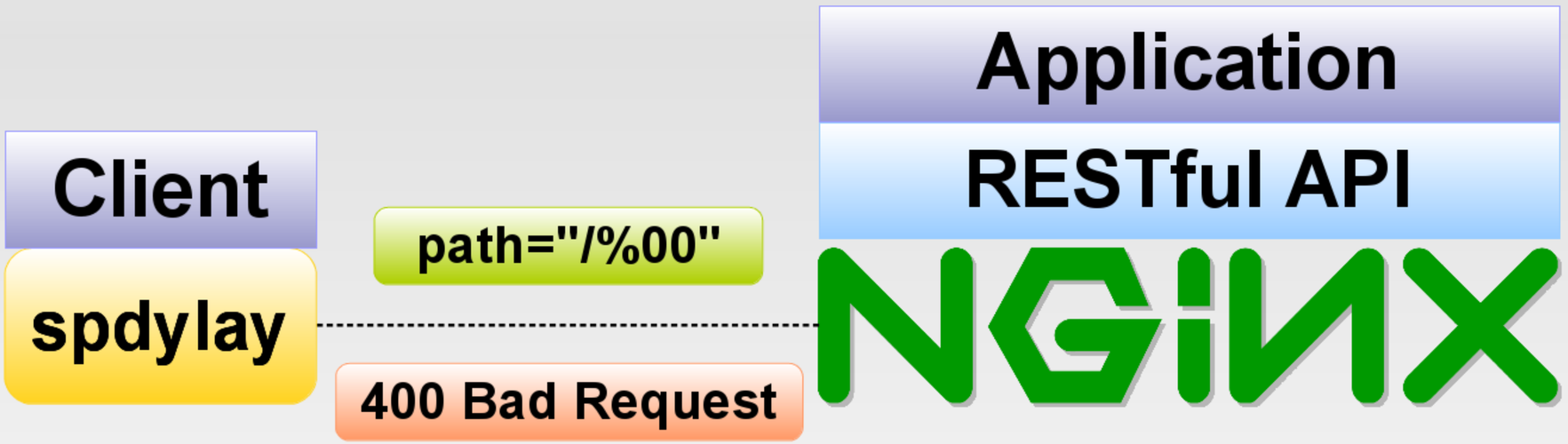
Fixing and reanalyzing reveals deeper issues!



Example: Optional Feature



Example: Optional Feature



Conclusions

- Automated non-interoperability finding with PIC
  - Joint symbolic execution
  - Dispersive exploration
  - Return normalization
  - Bugs found and fixed!

- Future Work
  - Generalized analysis framework: poster + demo!