# Detecting Script-to-Script Interactions in Call Processing Language

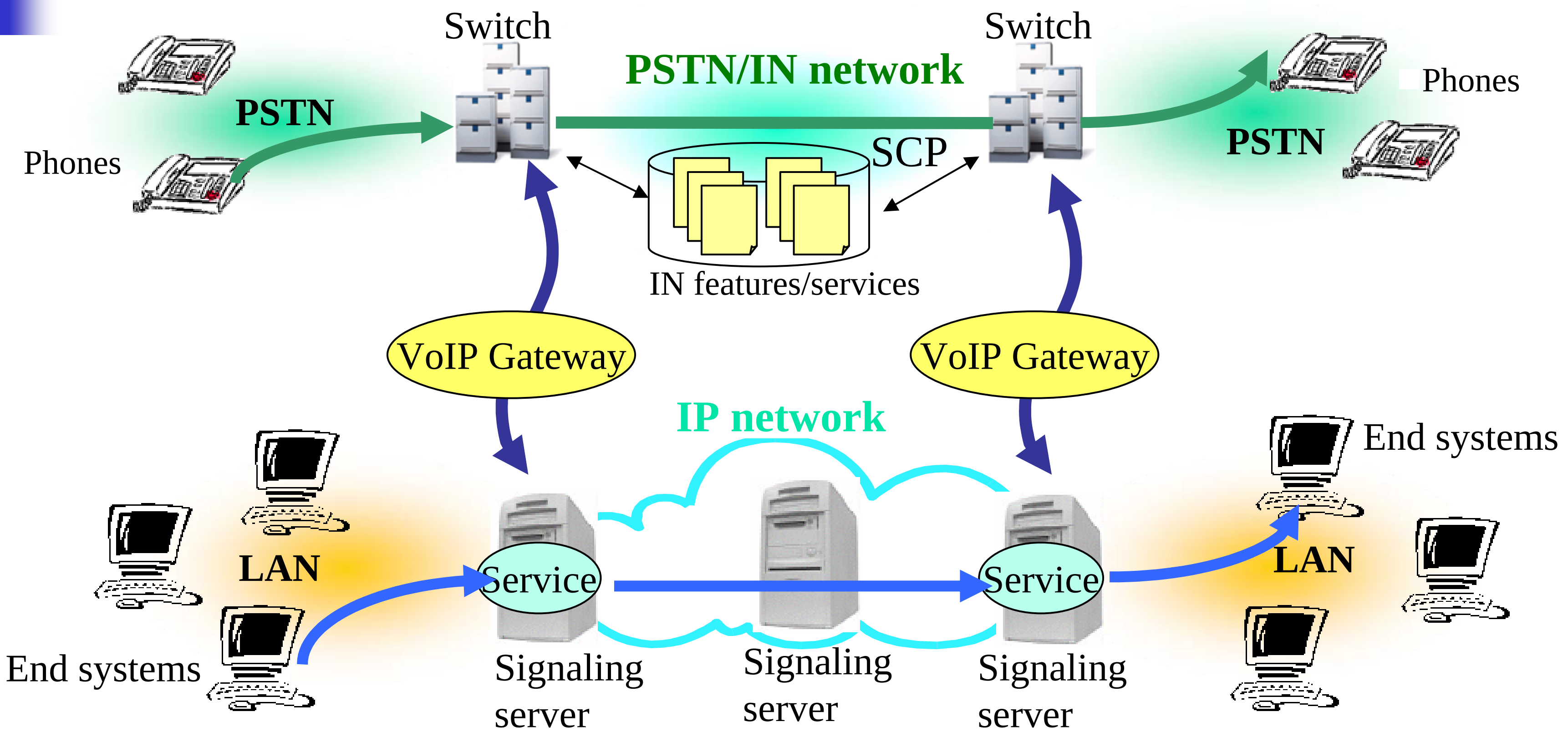
Masahide Nakamura, Ken-ichi Matsumoto,

*Grad. School of Information Science, Nara Institute of Science and Technology*

Ken-ichi Matsumoto, Tohru Kikuno

*Graduate School of Information Science and Technology, Osaka University*

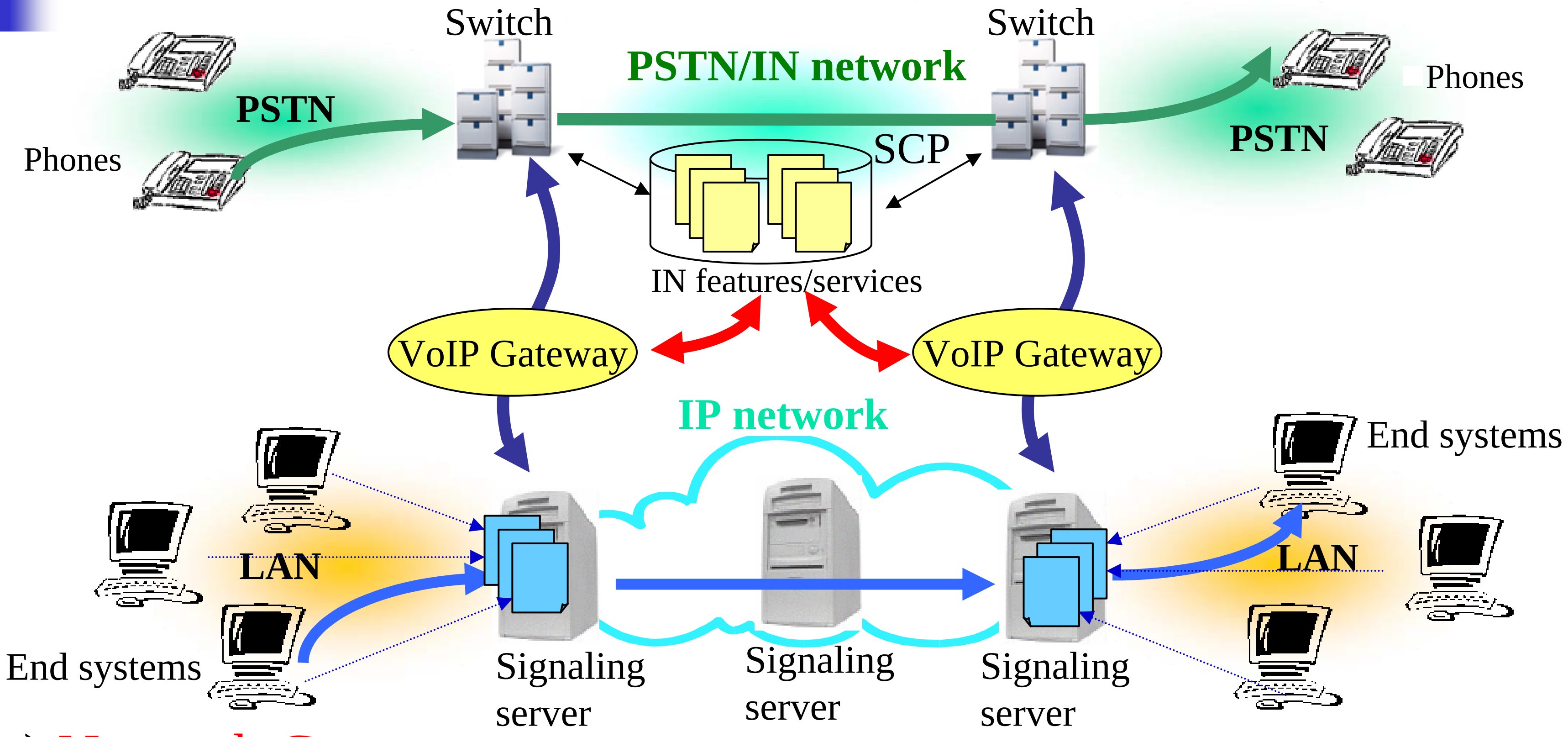
## Internet Telephony



- Widely studied at protocol level (SIP, H323)
- Advanced telecom services integrated with data services
- Decentralized service/feature management

⇨ Concerns are shifting to service level.

## Two Approaches for Service Provision



(a) Network Convergence

- Activate IN features/services through API (e.g., JAIN).

(b) Programmable Services

- End-users define and deploy own features/services.

## Call Processing Language (CPL)

An XML-based language for programmable service in the Internet Telephony.

- RFC 2824 of IETF (proposed standard )
- DTD-based syntax definition (also, XML-schemas)
- Mainly for switching / network services (for SIP, H.323)
- Some security considerations
  - Prohibits loops, recursive calls, activations of external programs.
- Commercial and open-source implementations (e.g., VOCAL)

Each user describes own customized service in a *CPL script*.
Then, install the script in the local signaling server.

⇨ Powerful and flexible service creation.

## Drawbacks of Programmable Service

(a) Service description by naive users

- The DTD-based syntax definition cannot guarantee the semantic correctness of a CPL script.

⇩

- There are many ways to make CPL scripts semantically wrong
- Cause ambiguity, redundancy, inconsistency

(b) Services in the Signaling servers distributed on the Internet can be added, deleted or modified at anytime

- It is impossible to enumerate all possible services

⇩

- FI detection and resolution by off-line analysis cannot be performed

## Goal of research

(a) Establish a guideline to guarantee semantic correctness for *each single* CPL script

Characterize *semantic warnings* in CPL script

(b) Propose algorithm to detect FIs among all scripts involved in a call at run time

Characterize FIs as the *semantic warnings* over *multiple CPL scripts*

## CPL Script

**Switches** represent conditional branches

- `<address switch>`, `<string switch>`, `<time switch>`, and `<priority switch>`

**Location Modifiers** add/remove locations

- `<explicit location>`, `<location lookup>`, `<location removal>`

**Signaling operations** cause signaling events

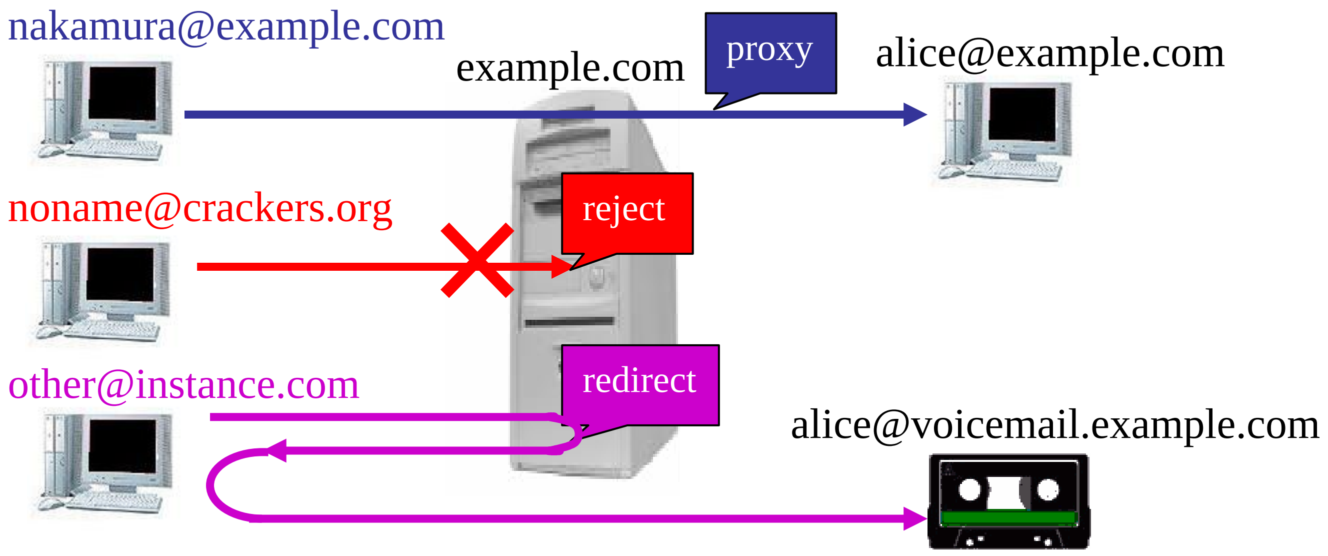
- `<proxy>`, `<redirect>` and `<reject>`

Full specification is found in RFC2824
http://www.ietf.org/rfc/rfc2824.txt
http://www.ietf.org/internet-drafts/draft-ietf-iptel-cpl-06.txt

## Describing Services with CPL(1)

Example requirement

- Alice *alice@example.com* wants to receive incoming calls only from domain *example.com*.
- Alice wants to reject all calls from *crackers.org*.
- Alice wants to redirect any other calls to her voice mail *alice@voicemail.example.com*.



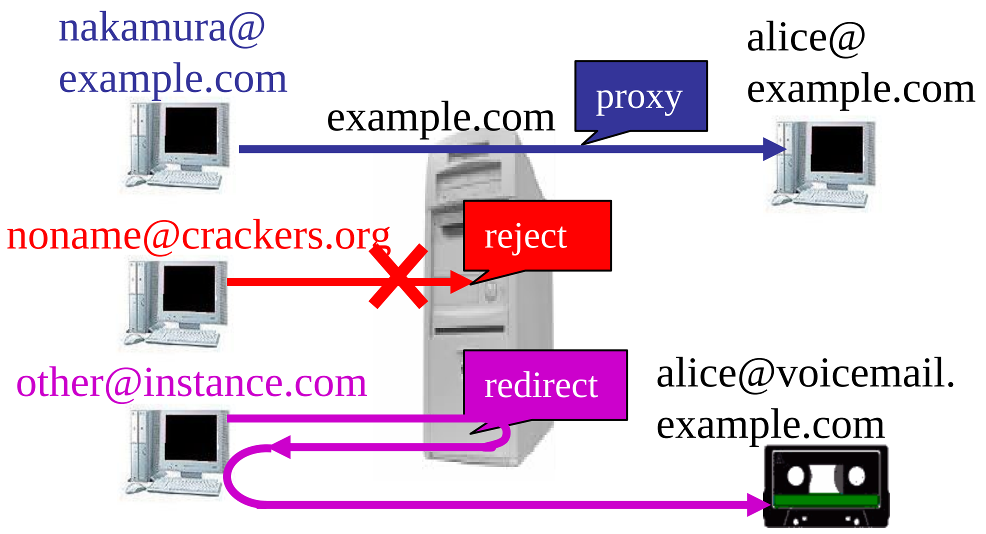
# Describing Services with CPL(2)

```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL
1.0//EN"  "cpl.dtd">

<cpl>
 <subaction id="voicemail">
   <location url=
         "sip:alice@voicemail.example.com">
     <redirect />
   </location>
 </subaction>

 <incoming>
   <address-switch field="origin" subfield="host">
     <address subdomain-of="example.com">
       <location url="sip:alice@example.com">
         <proxy />
       </location>
     </address>
     <address subdomain-of="crackers.org">
       <reject status="reject" />
     </address>
     <otherwise>
       <sub ref="voicemail" />
     </otherwise>
   </address-switch>
 </incoming>
</cpl>
```

- DTD = (Data Type Definition)
- Begins with <tag>, ends with <tag/>
- Subaction = Subroutine

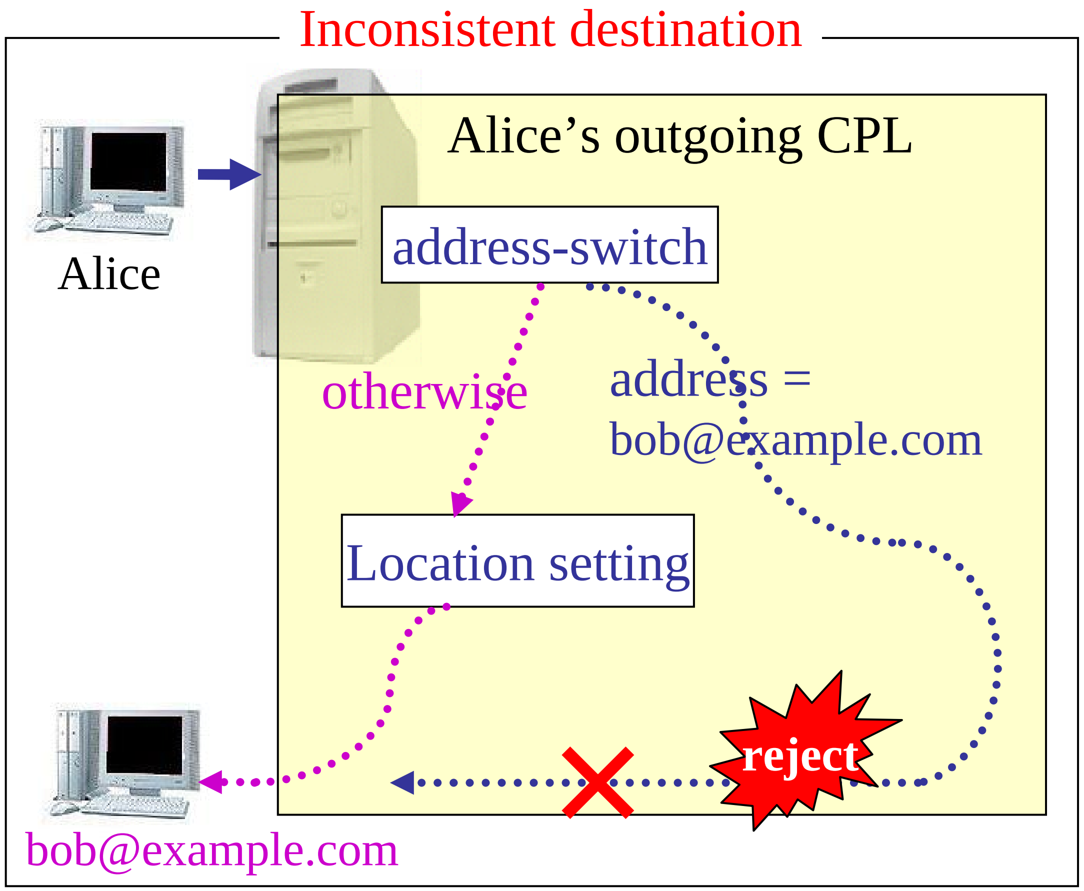


## Semantic warnings

1. Multiple forwarding addresses
2. Unused subactions
3. Call rejection in all paths
4. Address set after address switch
5. Overlapped conditions in single switch
6. Identical switches with the same parameters
7. Overlapped conditions in nested switches
8. Incompatible conditions in nested switches

## Address set after address switch (ASAS)

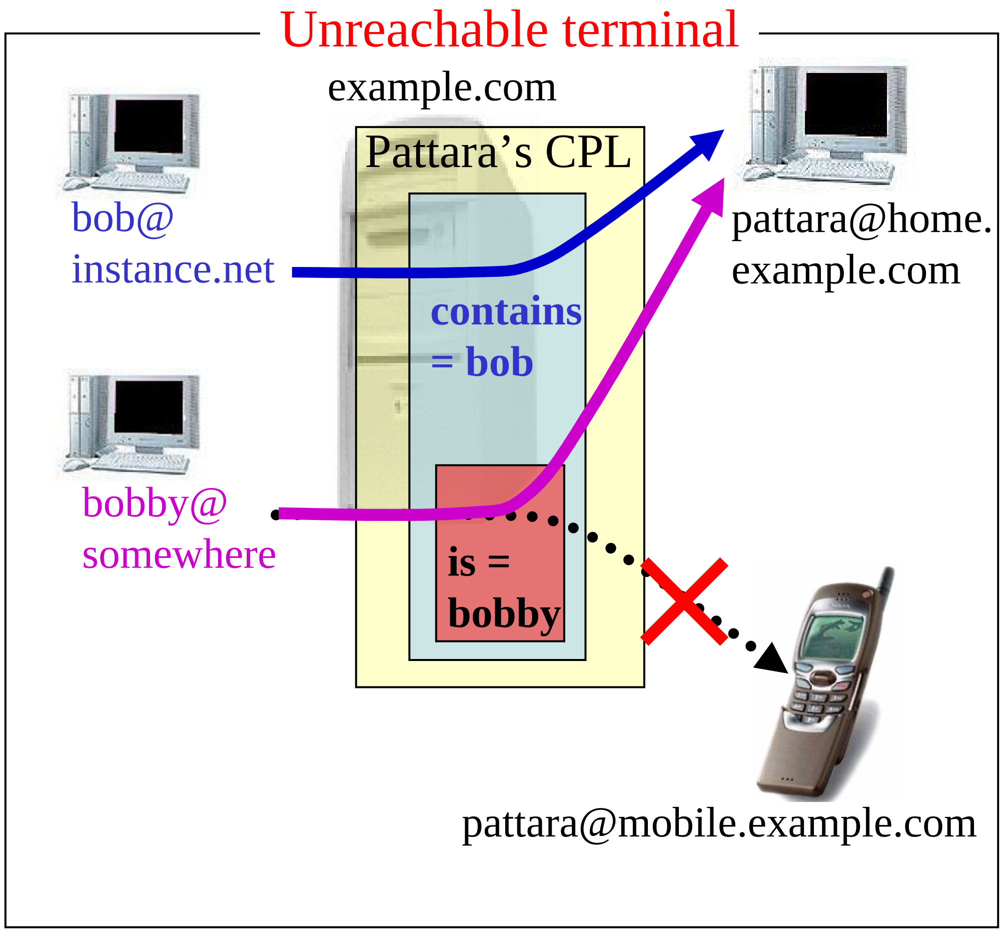
***Definition:*** When `<address>` and `<otherwise>` tags are specified as outputs of `<address-switch>`, the same address evaluated in the `<address>` is set in the `<otherwise>` block.



```
<cpl>
  <outgoing>
    <address-switch field="destination">
      <address is="sip:bob@example.com">
        <reject status="reject"
            reason="I don't call Bob" />
      </address>
      <otherwise>
        <location url="sip:bob@example.com">
            <proxy/>
        </location>
      </otherwise>
    </address-switch>
  </outgoing >
</cpl>
```

## Overlapped Conditions in Single Switches (OCSS)

***Definition:*** The condition is overlapped among the multiple output tags of a switch.



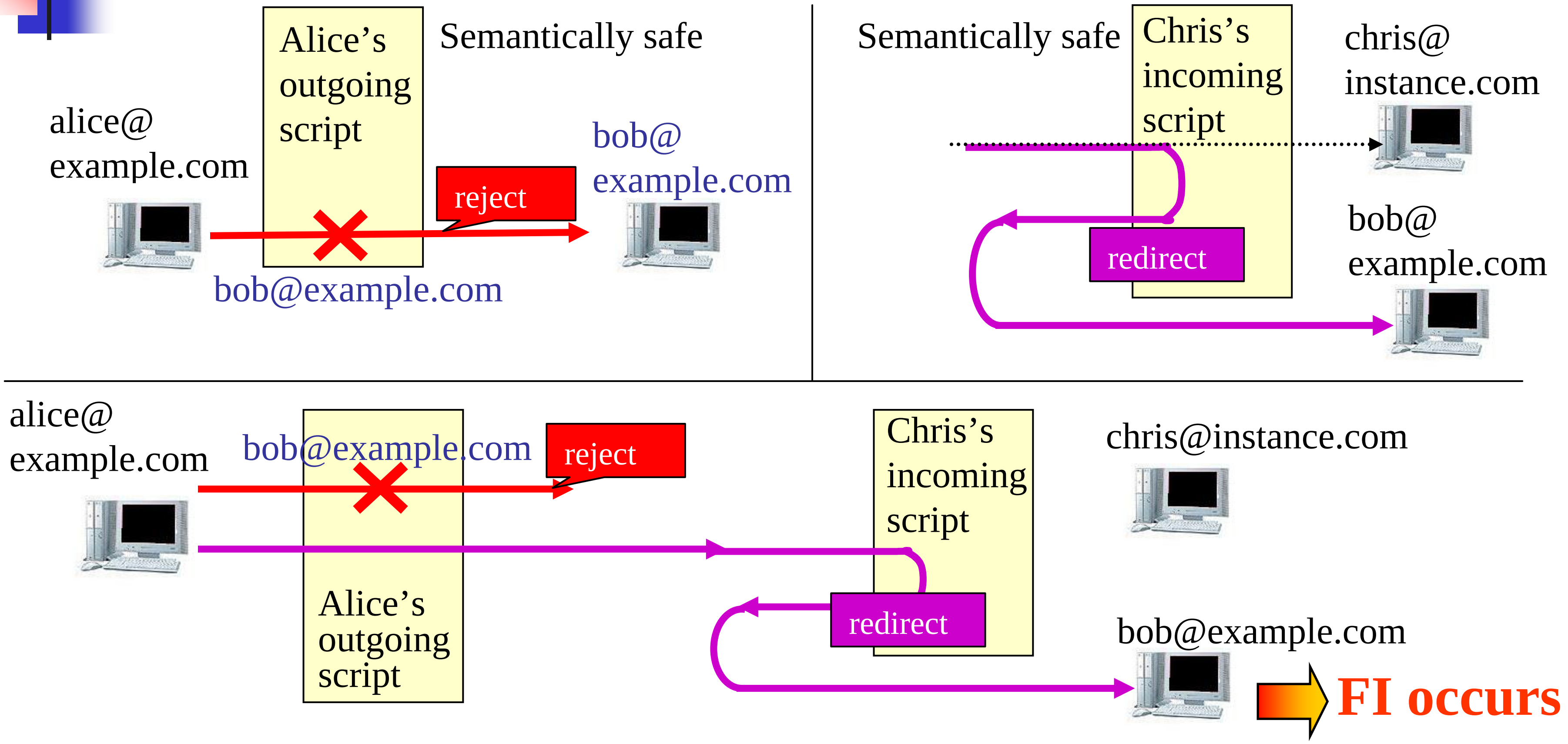
```
<cpl>
  <incoming>
    <address-switch field="originator" >
      <address contains="bob">
        <location url=
             "sip:pattara@home.example.com">
          <proxy />
        </location>
      </address>
      <address is="bobby">
        <location url=
           "sip:pattara@mobile.example.com">
          <proxy />
        </location>
      </address>
    </address-switch>
  </incoming>
</cpl>
```

## Feature Interaction in CPL script

- Even if each individual script is free from *semantic warnings* (*semantically safe*), FIs can occur when multiple scripts are executed simultaneously at run time.

- SU-type interactions (e.g., CW&TWC) do not occur.
  - Each user can have a single CPL script at a time.
- Interactions occur between different scripts owned by different users.

## Example of FI in multiple CPL scripts



Address Set after Address Switch (ASAS)

Define the FIs as
*semantic warnings over multiple scripts*

## FI detection Problem

- FI definition:

CPL script *s* and *t* interact with respect to *a call scenario c*
$\Longleftrightarrow$ *s* and *t* are *semantically safe*, but $s \triangleright_c t$ is *NOT semantically safe*
( $\triangleright_c$ is *combine operator*)

- FI detection Problem:
  - Detect FIs among multiple CPL scripts involved in a call with a call scenario *c*.

Detect FIs as the
*semantic warnings* over *multiple CPL scripts*

- Input and Output:
  - *Input:* CPL script *s* of the call originator, and *a call scenario c*
  - *Output*: FI occurs or not

## Combine Operator

- To get a combined behavior of two (successively proxied) scripts, we present the *combine operator* $\triangleright_c$

$$\boxed{\text{Combined script } r = s \triangleright_c t}$$

- *Definition*: Substituting the <proxy> nodes in *s* that is executed in *the call scenario c*, with *incoming actions* of *t*

script *r*

```
<cpl>
 <outgoing>
   <location url="sip:u@exam.com">
     </redirect>
   </location>
 </outgoing >
</cpl>
```

## FI Involved in More than 2 Scripts

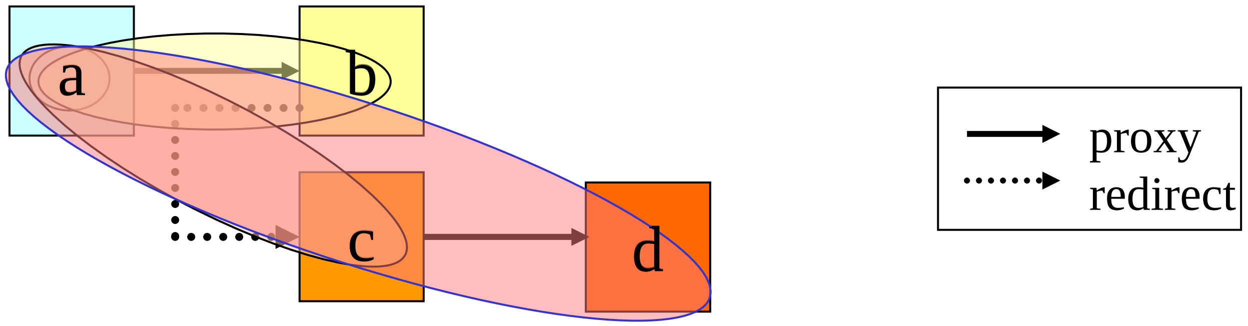
- A call could involved more than two scripts.

**Generalized FI Definition**

A feature interaction occurs w.r.t. $s_0$ and c $\Leftrightarrow$

There exists some $k$ s.t. $s_0 \rhd_c s_1 \rhd_c \ldots \rhd_c s_k$ is not safe.

Proposed Algorithm $Succ(s_0, c)$



(1) a (2) $a \rhd_c b$ (3) $a \rhd_c c$ (4) $a \rhd_c c \rhd_c d$

We check semantic warnings for these four combination

# Example of FI Detection

Alice's Script (*S1*)

```
<cpl>
  <outgoing>
    <address-switch field="destination">
      <address is="sip:bob@example.com">
        <reject status="reject"/>
      </address>
      <otherwise>
        </proxy>
      </otherwise>
    </address-switch>
  </outgoing >
</cpl>
```

**semantically safe**

Chris's Script (*S2*)

```
<cpl>
  <incoming>
    <location url="sip: bob@example.com ">
      <redirect />
    </location>
  </incoming>
</cpl>
```

*Input* { Originator: *Alice*
Call Scenario: *Alice calls Chris*

$$\begin{tikzcd} S1 \arrow[r] & S2 \end{tikzcd}$$

(1) *S1*   (2) $S1 \rhd_c S2$

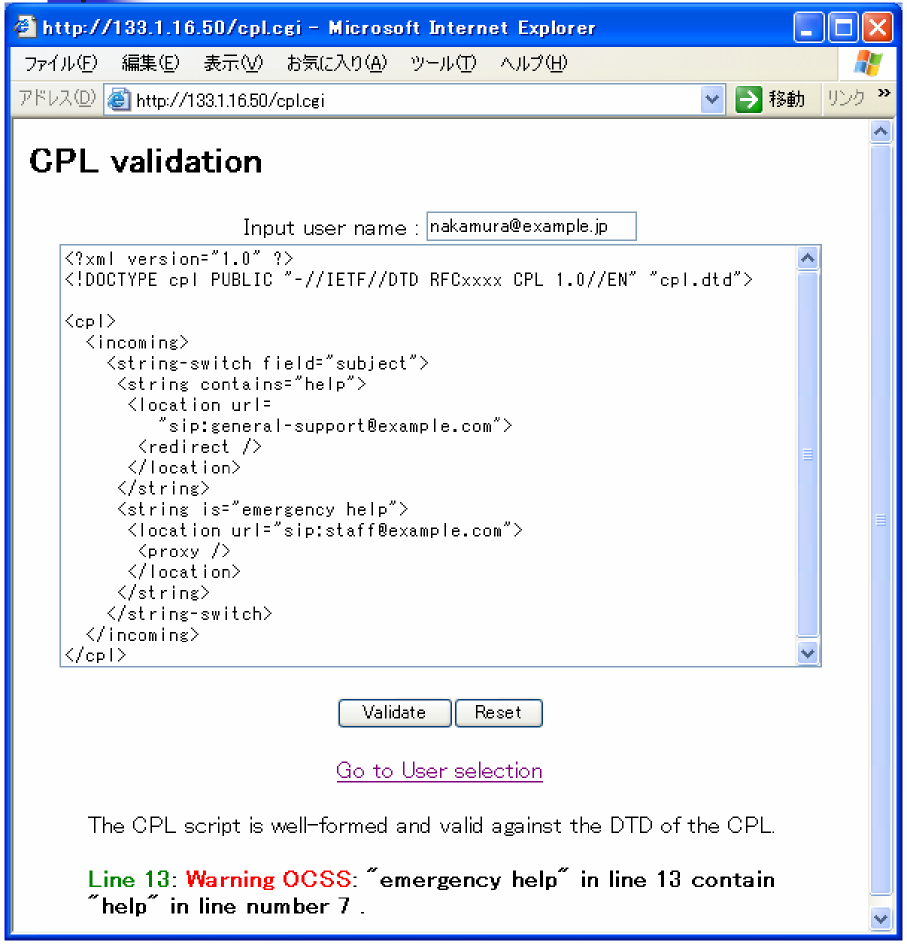
$S1 \rhd_c S2$

```
<cpl>
  <outgoing>
    <address-switch field="destination">
      <address is="sip:bob@example.com">
        <reject status="reject"/>
      </address>
      <otherwise>
        <location url="sip:bob@example.com ">
          <redirect />
        </location>
      </otherwise>
    </address-switch>
  </outgoing >
</cpl>
```

**ASAS**

**FI occurs**

## Tool Support


http://133.1.16.50/cpl.cgi - Microsoft Internet Explorer
ファイル(F) 編集(E) 表示(V) お気に入り(A) ツール(T) ヘルプ(H)
アドレス(D) http://133.1.16.50/cpl.cgi
移動 リンク

CPL validation

Input user name : nakamura@example.jp

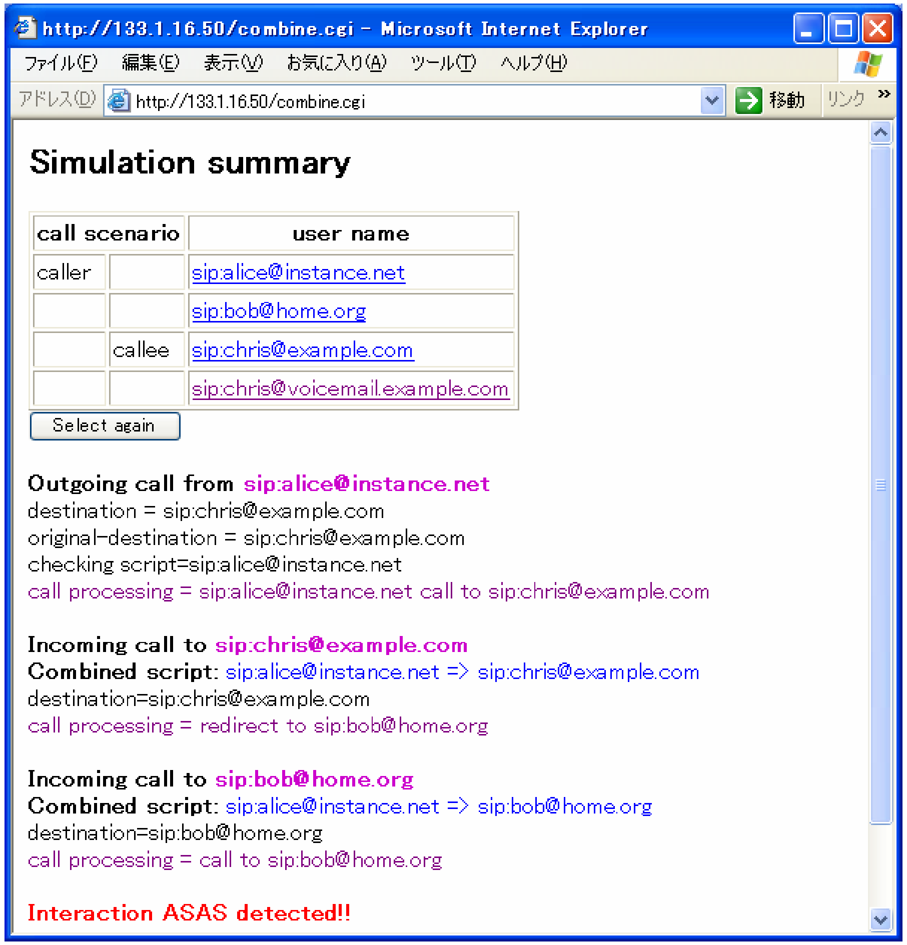
```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd">

<cpl>
  <incoming>
    <string-switch field="subject">
     <string contains="help">
      <location url=
         "sip:general-support@example.com">
       <redirect />
      </location>
     </string>
     <string is="emergency help">
      <location url="sip:staff@example.com">
       <proxy />
      </location>
     </string>
    </string-switch>
  </incoming>
</cpl>
```

Go to User selection

The CPL script is well-formed and valid against the DTD of the CPL.

Line 13: Warning OCSS: "emergency help" in line 13 contain "help" in line number 7 .


http://133.1.16.50/combine.cgi - Microsoft Internet Explorer
ファイル(F) 編集(E) 表示(V) お気に入り(A) ツール(T) ヘルプ(H)
アドレス(D) http://133.1.16.50/combine.cgi
移動 リンク

Simulation summary

| call scenario | | user name |
|---|---|---|
| caller | | sip:alice@instance.net |
| | | sip:bob@home.org |
| | callee | sip:chris@example.com |
| | | sip:chris@voicemail.example.com |

**Outgoing call from sip:alice@instance.net**
destination = sip:chris@example.com
original-destination = sip:chris@example.com
checking script=sip:alice@instance.net
call processing = sip:alice@instance.net call to sip:chris@example.com

**Incoming call to sip:chris@example.com**
**Combined script:** sip:alice@instance.net => sip:chris@example.com
destination=sip:chris@example.com
call processing = redirect to sip:bob@home.org

**Incoming call to sip:bob@home.org**
**Combined script:** sip:alice@instance.net => sip:bob@home.org
destination=sip:bob@home.org
call processing = call to sip:bob@home.org

**Interaction ASAS detected!!**

(a) CPL Checker (b) FI Simulator

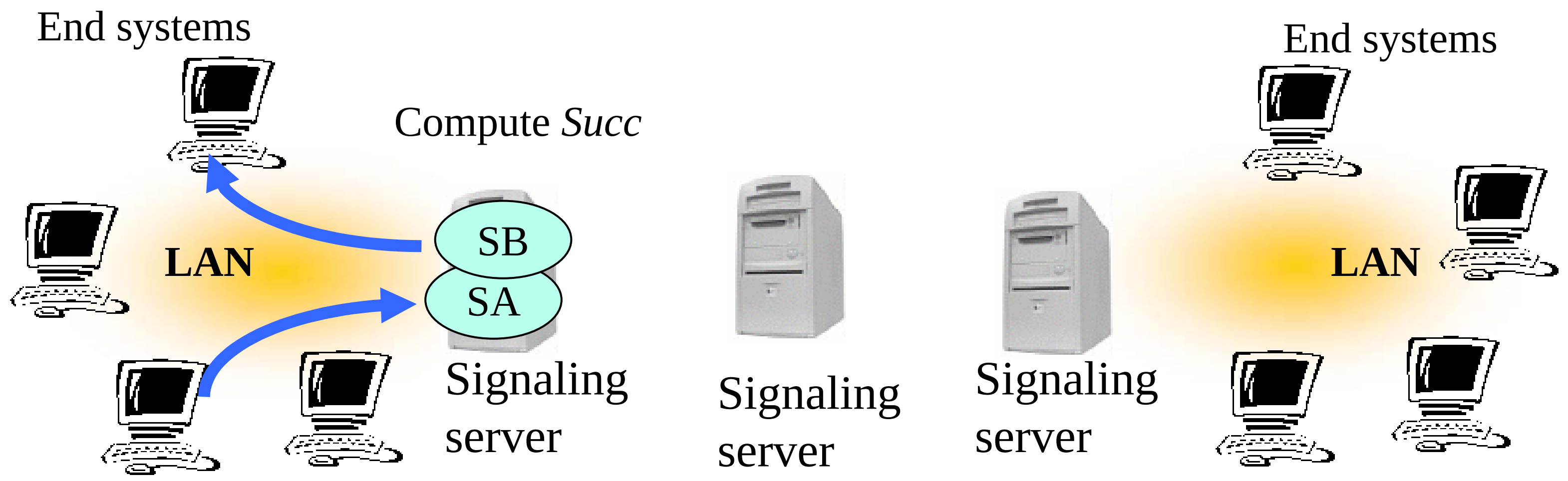
http://www-kiku.ics.es.osaka-u.ac.jp/~pattara/CPL/

## Conclusion and Future Work

- New eight semantic warnings.
- Definition of FI in CPL programmable environment.
- Algorithm *Succ* to detect FIs involved in a call.

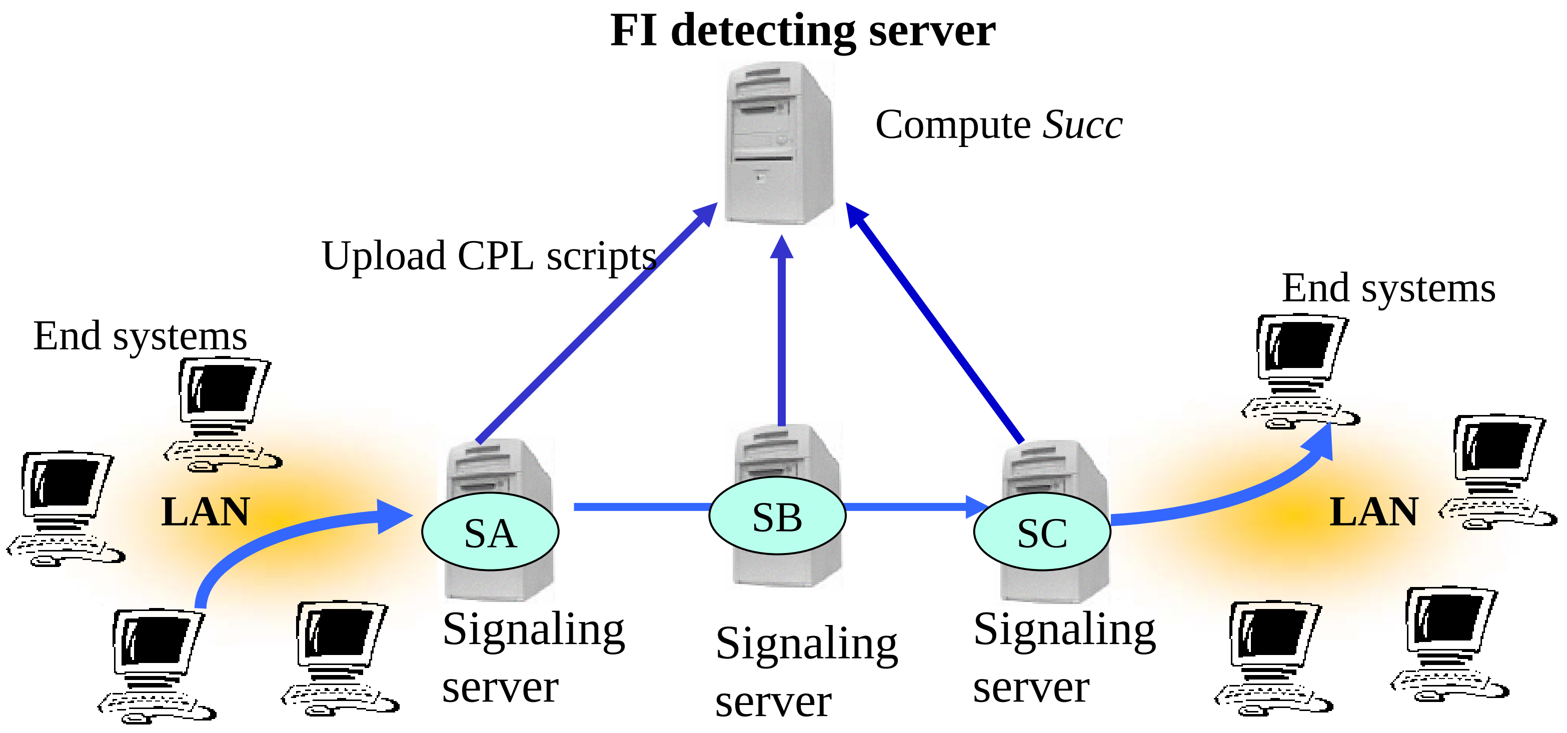
- Future work
  - Run-time FI detection mechanism.
  - Evaluation of how many FIs can be covered
  - FI between programmable services and ready-made services.

## Intra-Server Call



- Relatively easy to detect FI.
  - FI detector in VOCAL front-end.

# Global FI Detecting Server

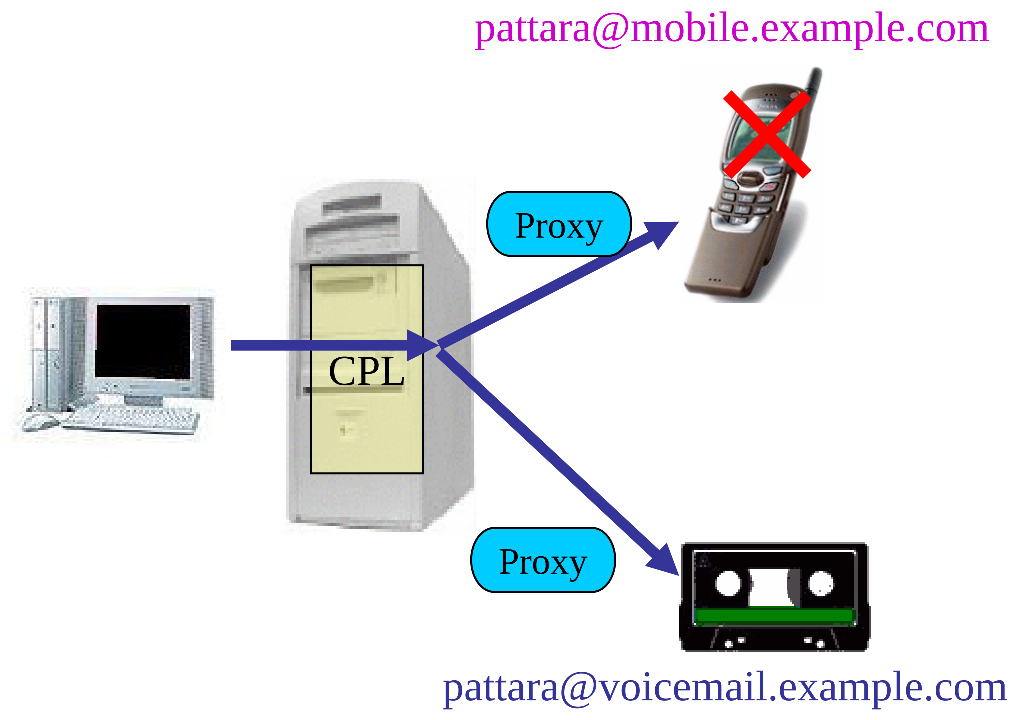


- For public Internet
  - Quite difficult to realize due to privacy/authentication.
  - Resolution - ABSOLUTELY NO WARRANTY policy?
- For dedicated service
  - Possibility to use dedicated servers and channels.

## Multiple forwarding addresses (MF)

***Definition:*** After multiple addresses are set by <location> tags, <proxy> or <redirect> comes.

Unreachable Terminal



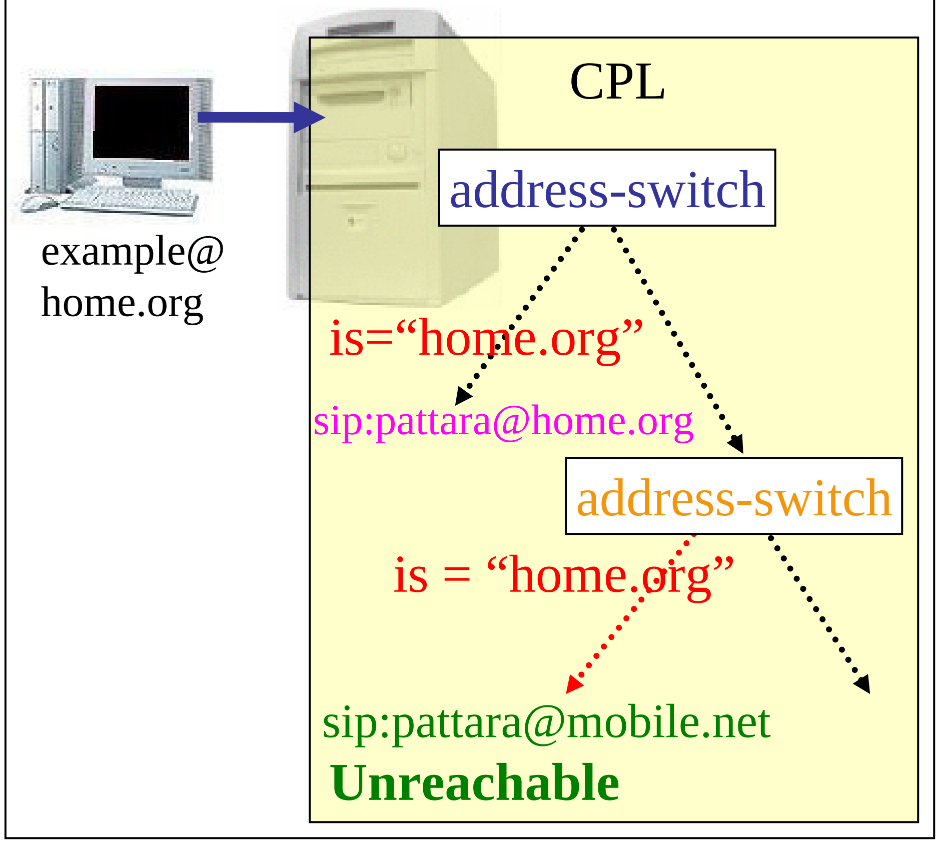
**Immediately answer**

```
<cpl>
  <incoming>
    <location url=
          "sip:pattara@mobile.example.com">
    <location url=
          "sip:pattara@voicemail.example.com">
    <proxy />
    </location>
  </incoming>
</cpl>
```

## Identical switches with the same parameters (IS)

***Definition:*** After a switch tag with a parameter, the same switch with the same parameter comes.

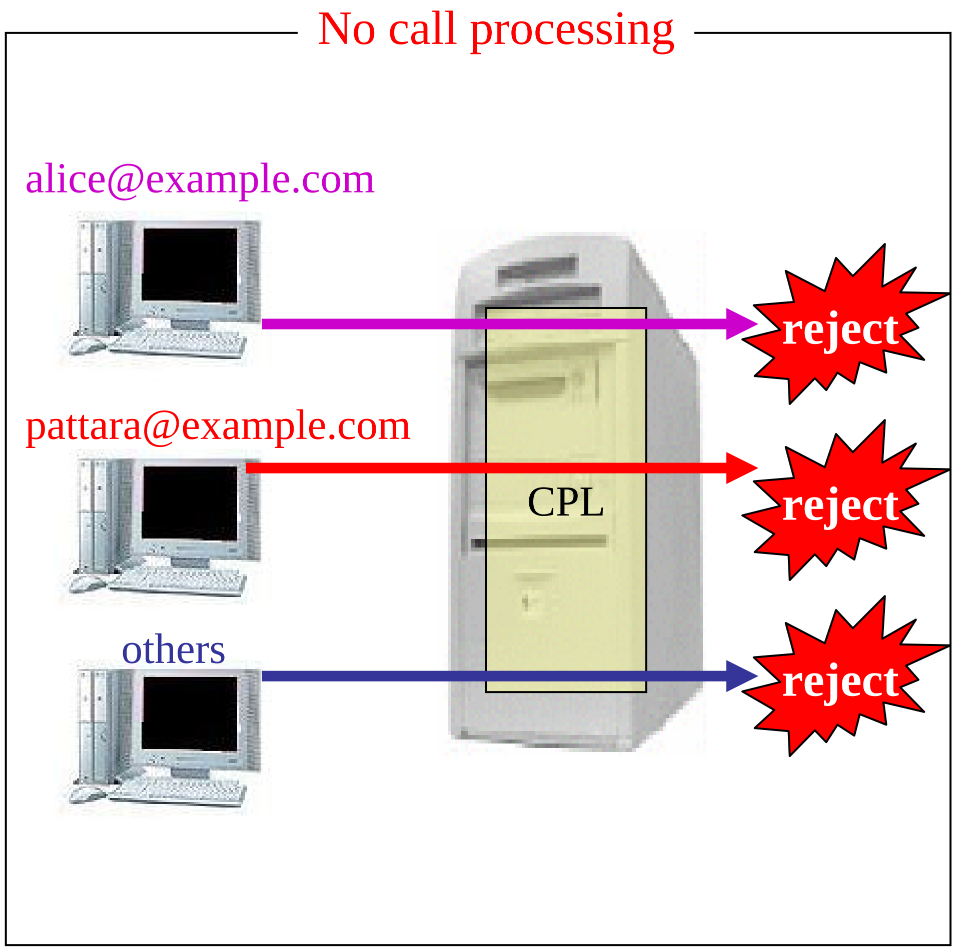
Unreachable and redundant script



```
<cpl>
 <incoming>
   <address-switch field="origin" subfield="host">
     <address subdomain-of="home.org">
       <location url="sip:pattara@home.org">
         <proxy />
       </location>
     </address>
     <otherwise>
       <address-switch field="origin" subfield="host">
         <address subdomain-of="home.org">
           <location url="sip:pattara@mobile.net">
             <proxy />
           </location>
         </address>
       </address-switch>
     </otherwise>
   </address-switch>
 </incoming>
</cpl>
```

## Call rejection in all paths (CR)

***Definition:*** All execution paths terminate at <reject>.



```
<cpl>
 <incoming>
  <address-switch field="origin">
   <address is=“sip:alice@example.com">
    <reject status="reject"
        reason="I don’t accept call from alice" />
   </address>
   <address is="sip:pattara@example.com">
    <reject status="reject"
        reason="I don’t accept call from Pattara" />
   </address>
   <otherwise>
    <reject status="reject"
        reason="I don’t accept call from anyone" />
   </otherwise>
  </address-switch>
 </incoming>
</cpl>
```

## Unused Subactions (US)

***Definition:*** Subaction <subaction id= "foo" > exists, but <subaction ref= "foo" > does not.

Redundant script

```
<cpl>
  <subaction id="mobile">
    <location url="sip:jones@mobile.example.com" >
      <proxy />
    </location>
  </subaction>

  <incoming>
    <location url="sip:jones@example.com">
      <proxy />
    </location>
  </incoming>
</cpl>
```

## Successive Algorithm

A call scenario could involve more than two scripts, because of successive *redirect* and *proxy*

Compute *a set of scripts* to be combined by proposed algorithm *Successive*

- Input and output
  - *Input*: call originator, call scenario
  - *Output*: a set of scripts to be combined
- Identify *processing type* and *next address* in scripts
  - *Processing type*: how is the call processed (proxy, redirect, reject, or connected to end system)
  - *Next address*: where the call is directed next
- Create *a set of script*, according to processing type