



# CVP Advanced Topics (Part 1)

Paul Tindall  
EMEAR Customer Collaboration Technical Consulting

March 2014

# Topics We'll Examine

All of the topics covered result from actual solution requirements and work on real-world use cases where CVP is the chosen call handling platform and deployed in either Standalone or Comprehensive model



# Agenda

- REFER transfers and context data passing
- Correct codec selection in multi-region scenarios
- Intelligent call rejection

# REFER Transfers With Data



# CVP REFERs With Data Passing

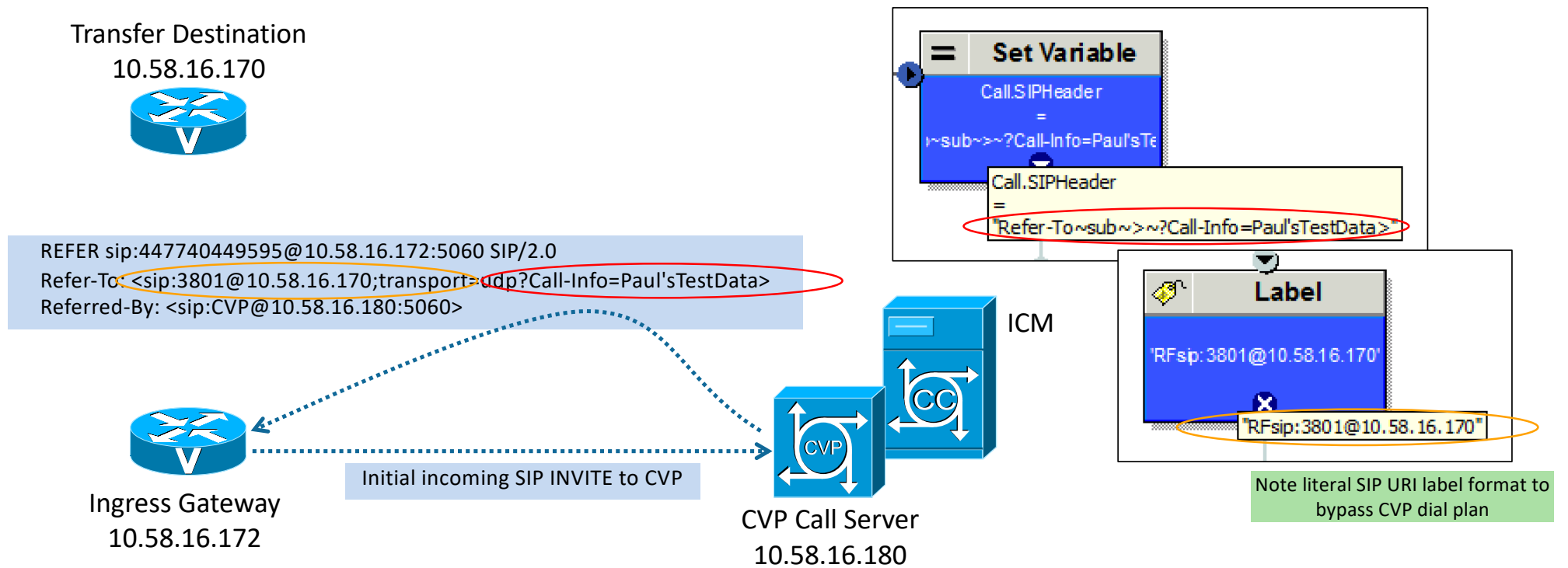
## Use Case / Challenge

- SIP REFERs are used to instruct the calling (or called) user agent to transfer the call
- Useful for dropping CVP/ICM out of the call signalling path
  - Perform the transfer at the ingress gateway or even further back along SIP trunk
  - Typical use case is for transfers to third-party ACD/PBXs (non-CCE integrated)
  - Avoids tromboning call signalling through CVP Call Server B2BUA
  - Can also avoid tromboning media through ingress gateway depending on where the REFER is consumed

**Problem: How do we pass context data with the transfer using REFER?**

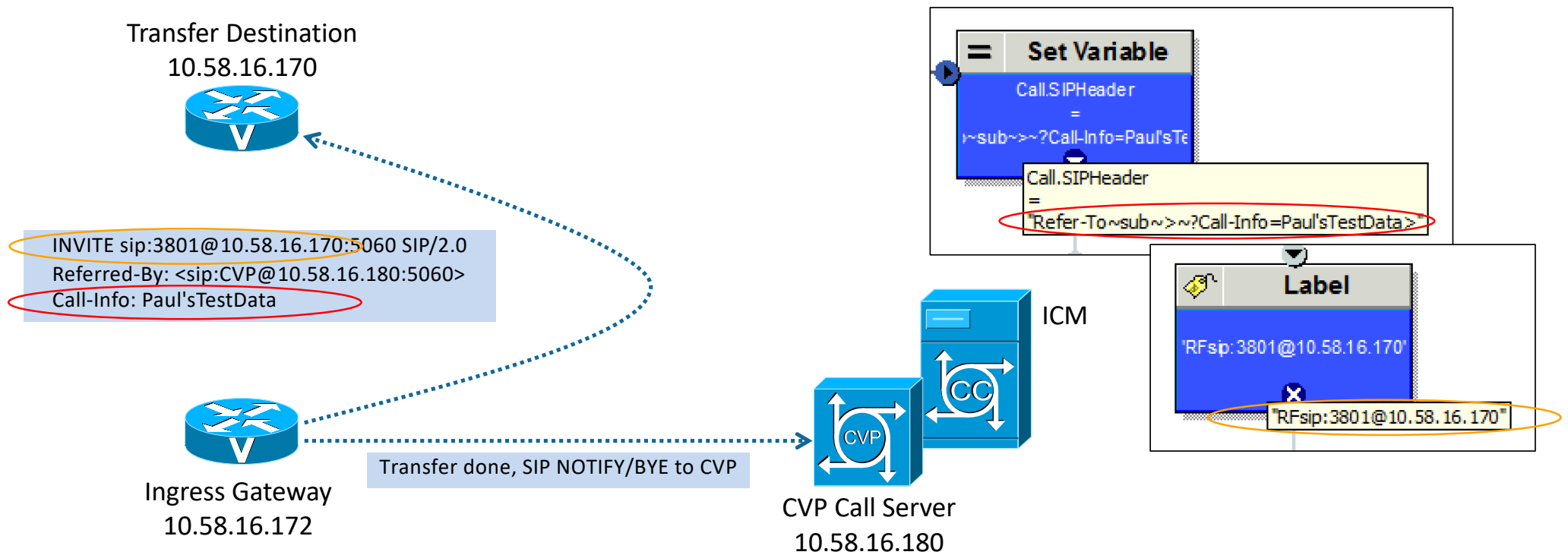
# CVP REFERs With Data Passing

## ICM Invokes Refer Mode Transfer



# CVP REFERs With Data Passing

## Ingress Gateway Consumes REFER



# CVP REFERs With Data Passing

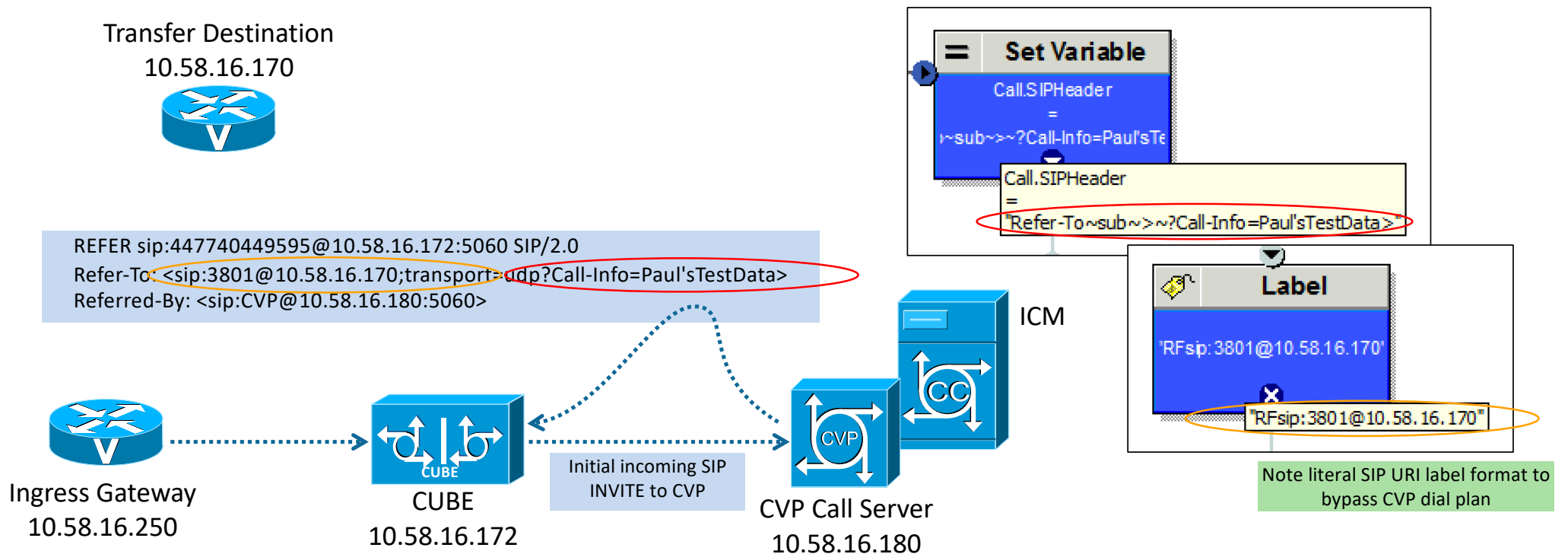
## Not Quite Finished

- Unfortunately not quite that straightforward to consume the REFER
- CVP Survivability intercepts the **ev\_transfer\_request** event when the REFER is received
- Survivability manually performs the transfer but ...
- It doesn't populate the SIP header using the Refer-To Call-Info URI parameter
  - Either don't use CVP Survivability, or ...
  - Modify it to let the gateway default handling perform the REFER
  - Comment-out the **ev\_transfer\_request** entry in the state machine so it's ignored



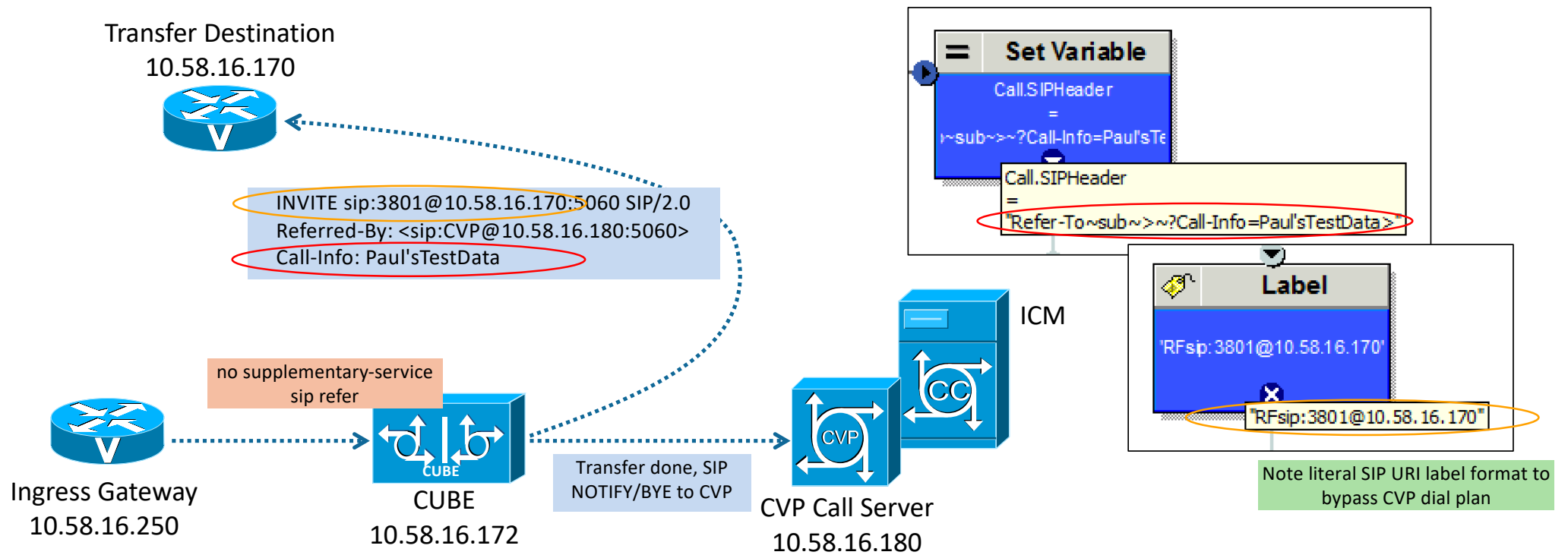
# CVP REFERs With Data Passing

What if CUBE is Present?



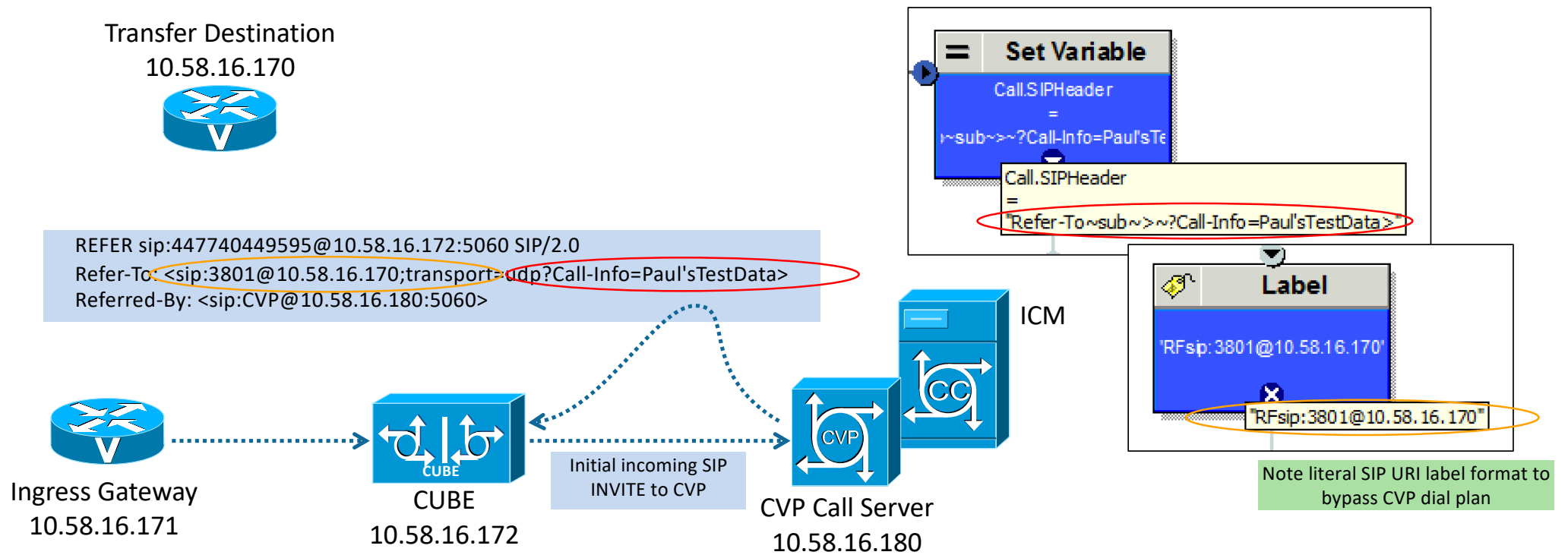
# CVP REFERs With Data Passing

## CUBE Consumes REFER



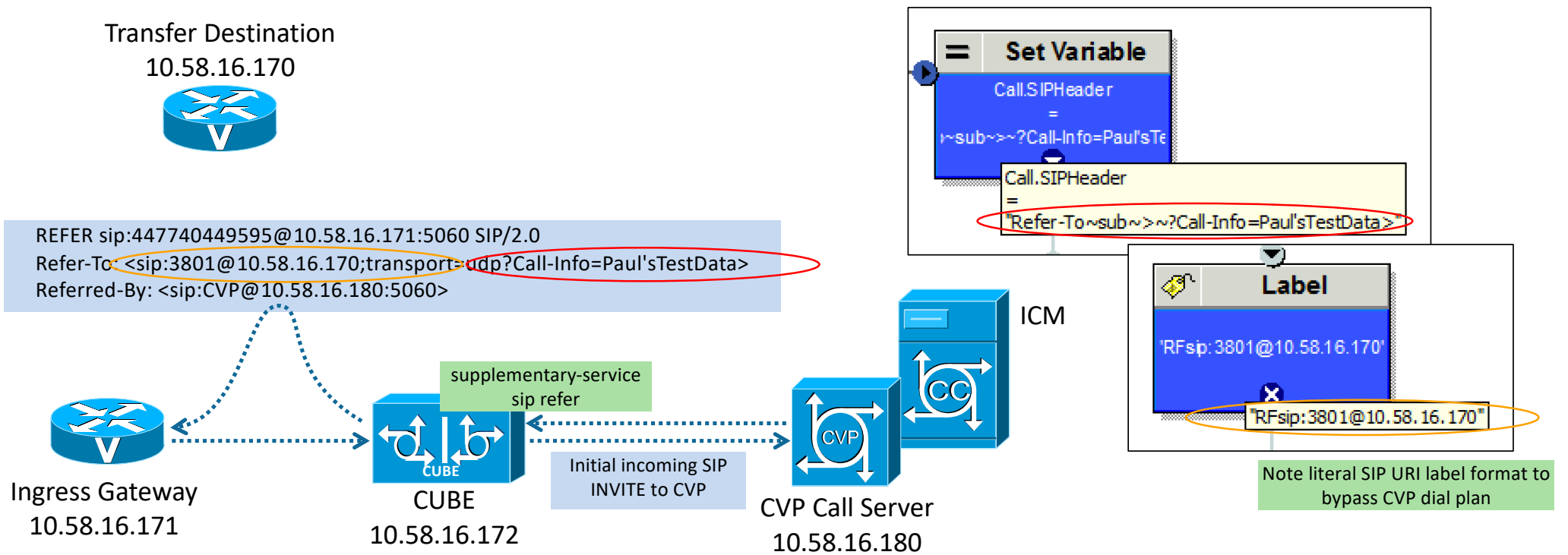
# CVP REFERs With Data Passing

## CUBE REFER Passthru



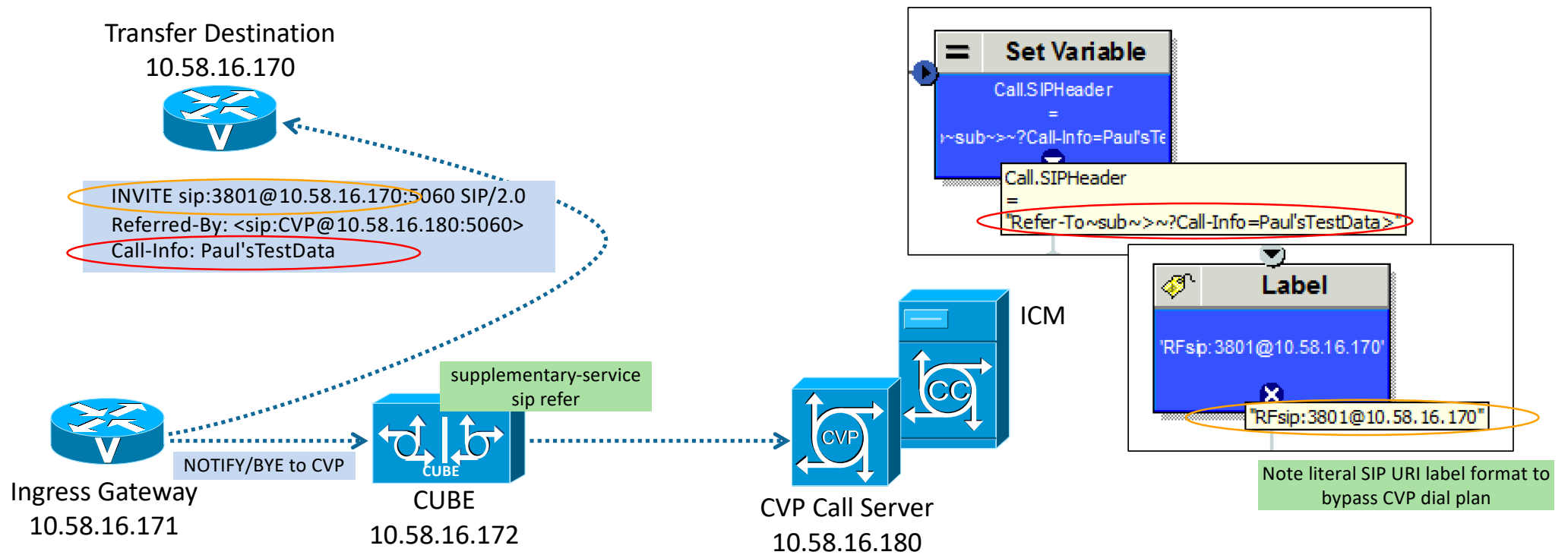
# CVP REFERs With Data Passing

## CUBE REFER Passthru



# CVP REFERs With Data Passing

## Ingress Gateway Consumes REFER



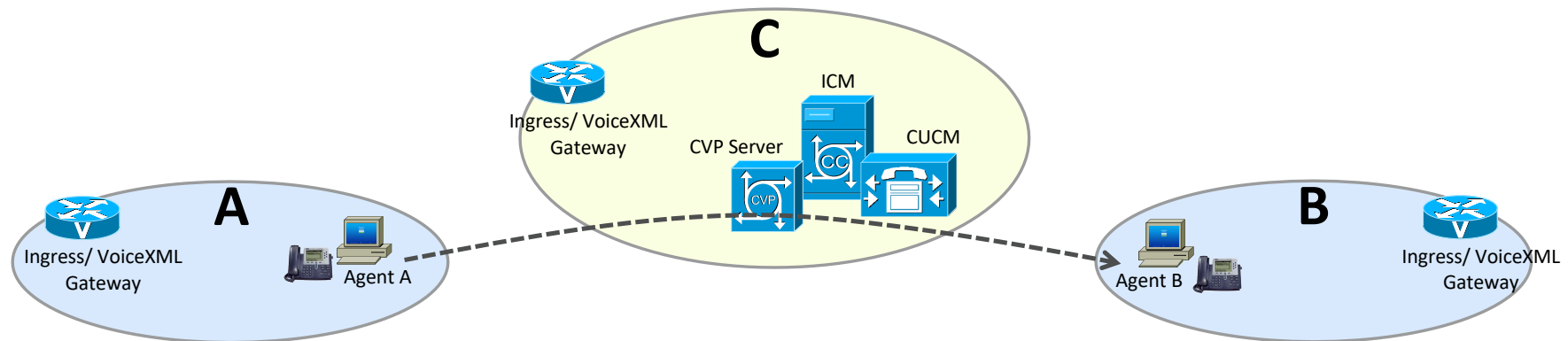
# Correct Codec Selection In Multi-Region Scenarios



# Codec Selection Across Regions

## Use Case / Challenge

- Consider the architecture



- Calls originating at Region A transferred by CVP to Region B
- G.711 required for calls within the region
- G.729 required for calls between regions

# So, What's The Problem?

## Didn't CVP 8.x Locations Awareness Fix This?

- CVP sits in the signalling path between calling and called user agents
- CVP receives **originating endpoint location** from UCM

```
INVITE sip:8808@10.58.16.180:5060 SIP/2.0
From: <sip:4708@10.58.16.175>;tag=407506~0ae7cfdd-e991-41e8-80de-0f6cf4701aa2-20499468
To: <sip:8808@10.58.16.180>
User-Agent: Cisco-CUCM9.0
Call-Info: <urn:x-cisco-remotecc:callinfo>;x-cisco-loc-id=0b8c38fb-d63c-646b-7b79-b71c62ec64da;x-cisco-loc-name=London; ...
```

- CVP sends **originating endpoint location** and **user agent IP address** to UCM

```
INVITE sip:4441@10.58.16.175;transport=udp SIP/2.0
From: 4708 <sip:4708@10.58.16.180:5060>;tag=dse9c1f888
To: <sip:4441@10.58.16.175;transport=udp>
User-Agent: CVP 9.0 (1) Build-670
Call-Info: <sip:10.58.16.175:5060>;purpose=x-cisco-origIP
Call-Info: <urn:x-cisco-remotecc:callinfo>;x-cisco-loc-id=0b8c38fb-d63c-646b-7b79-b71c62ec64da;x-cisco-loc-name=London; ...
```



# So, What's The Problem?

Didn't CVP 8.x Locations Awareness Fix This?

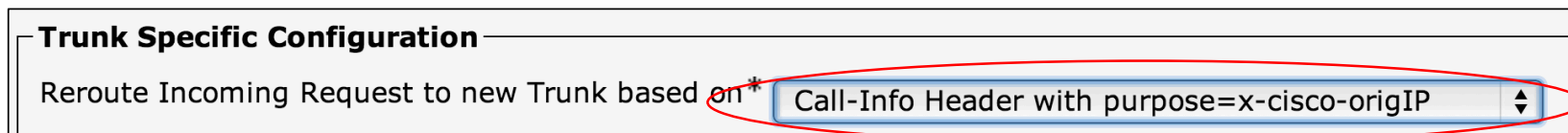
- UCM uses the Call-Info field location information
  - Knows calling and called endpoint locations
  - Performs correct bandwidth pegging
- Successfully addresses locations-based CAC when CVP is present
  - But codec selection is still based on CVP to UCM SIP trunk region

**Problem: UCM needs to know/use the calling endpoint region for calls received from CVP**

# No Problem With Gateways

Ingress Gateway → CVP → Agent Phone

- Originating IP address enables a fix-up if the call is from a gateway
  - Use this option on SIP trunk profile



- Selects an alternative SIP trunk based on x-cisco-origIP
  - Call is processed as though it came directly from the gateway
- Both CAC and codec selection are handled correctly

# Problem Happens When Agents Call

Agent A → CVP → Agent B

- If call is from an IP phone, Call-Info x-cisco-origIP is the UCM address
  - Unlike calls originating from gateways, it doesn't help us at all
- Call-Info x-cisco-loc-name contains the IP phone's location
  - UCM can't use that to determine the codec, it needs the region
  - UCM has no mechanism currently to derive region from the Call-Info header even if it was present

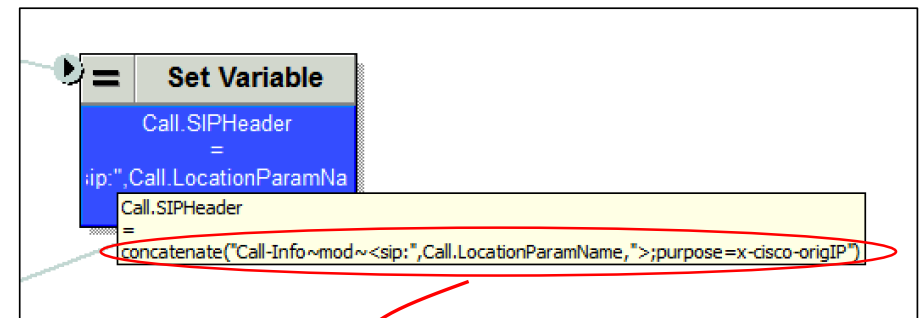
# Forcing The Calling Party Region

## Workaround

### 1. CVP/ICM knows the calling party location

- Either receives it on incoming INVITE from UCM
- Or, mapped from originating IP by CVP using its own location table

### 2. ICM script sets Call-Info SIP header x-cisco-origIP to that caller **location name**



```
Request-Line: INVITE sip:4441@10.58.16.175;transport=udp SIP/2.0
From: 4708 <sip:4708@10.58.16.180:5060>;tag=dsd5048ee4
To: <sip:4441@10.58.16.175;transport=udp>
User-Agent: CVP 9.0 (1) Build-670
Call-Info: <sip:London>;purpose=x-cisco-origIP
```

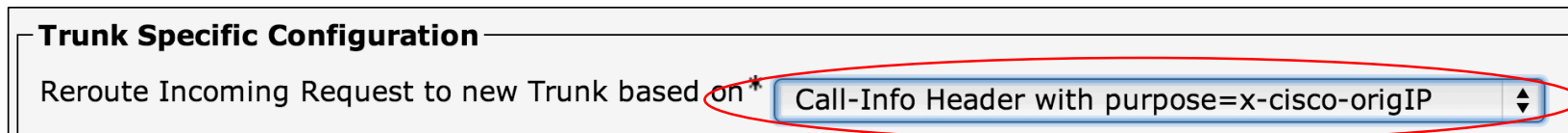
# Forcing The Calling Party Region

## Workaround

3. UCM does reroute to new trunk based on originating IP

**Trunk Specific Configuration**

Reroute Incoming Request to new Trunk based on \* **Call-Info Header with purpose=x-cisco-origIP**



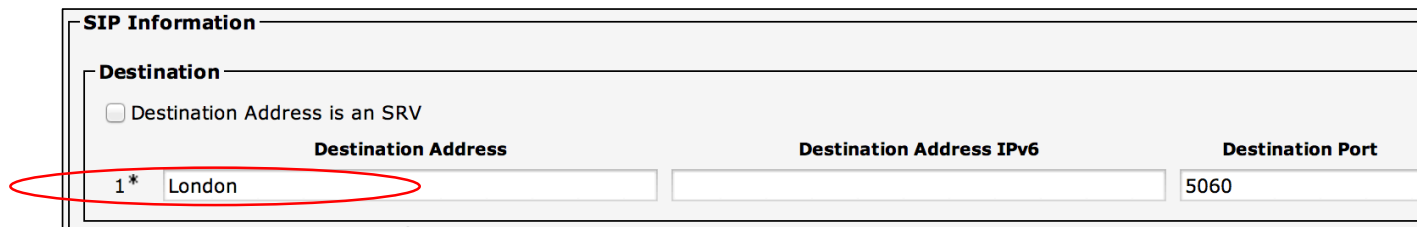
3. Locates a dummy trunk with its IP address matching the **location name** (contents of the Call-Info header)

**SIP Information**

**Destination**

Destination Address is an SRV




	Destination Address	Destination Address IPv6	Destination Port
1 *	London		5060



# Forcing The Calling Party Region

## Workaround

5. UCM uses the trunk device pool / region as normal to select the correct codec for the call
6. Needs a dummy trunk to be added for each calling party location

<input type="checkbox"/>		Name ^	Description	Calling Search Space	Device Pool	Route Pattern	Partition	Route Group	Priority	Trunk Type
<input type="checkbox"/>		<a href="#">Backwell_Dummy_Trunk</a>			<a href="#">Paul_Lab</a>					SIP Trunk
<input type="checkbox"/>		<a href="#">London_Dummy_Trunk</a>			<a href="#">London_Lab</a>					SIP Trunk
<input type="checkbox"/>		<a href="#">Rome_Dummy_Trunk</a>			<a href="#">Rome_Lab</a>					SIP Trunk

5. Same approach is compatible with calls originating at gateways
  - Common dummy trunk per location rather than one per gateway IP address

# Intelligent Call Rejection



# Intelligent Call Rejection

## Use Case / Challenge

- Not always desirable to answer all calls
- Sometimes want to reject calls without answering them, based on:
  - Particular call signaling content
  - Caller blacklist
  - System load
  - Number of calls active already on particular services
  - Reduce call costs for toll-free numbers
- Reject calls with specific cause code:
  - Most commonly require Busy or Unassigned Number

**Problem: (a) Calls sent to IVR are answered immediately  
(b) Need a way to disconnect with the required cause code**



# Intelligent Call Rejection

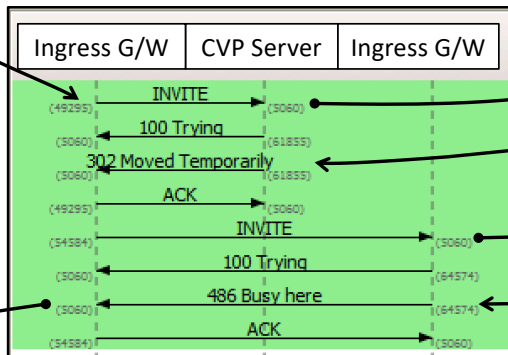
## ICM Script / CVP Comprehensive Model

- Use REFER mode transfer with CVP immediately, before performing Send To VRU
  - REFER is only performed when CVP has already transferred/answered the call
  - Does a redirect when the call hasn't been answered
  - 302 Temporarily Moved response is sent to the calling user agent
  - Includes redirect destination in the Contact header
  - Gateway sends new SIP INVITE to the redirect destination (typically on the same gateway)
  - That destination is configured in IOS to reject the call
  - Or, could invoke a TCL script that simply does a leg disconnect with the required cause code

# Intelligent Call Rejection

## ICM Script / CVP Comprehensive Model

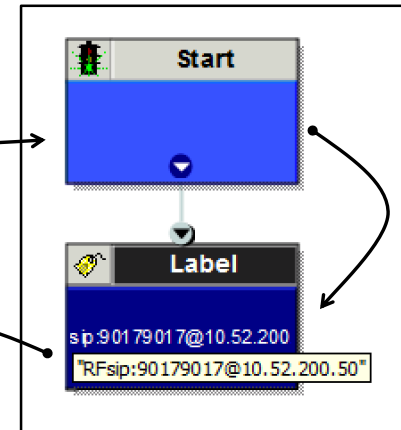
```
Jan 3 15:38:37.489: ISDN BR0/1/0 Q931: RX <- SETUP pd = 8 callref = 0x44
Jan 3 15:38:37.525: ISDN BR0/1/0 Q931: TX -> CALL_PROC pd = 8 callref = 0xC4
```



```
SIP/2.0 302 Moved Temporarily
Contact: <sip:90179017@10.52.200.50>
```

```
INVITE sip:90179017@10.52.200.50:5060 SIP/2.0
```

```
SIP/2.0 486 Busy here
Reason: Q.850;cause=17
```



```
Jan 3 15:38:37.597: ISDN BR0/1/0 Q931: TX -> DISCONNECT pd = 8 callref = 0xC4
Cause i = 0x8091 - User busy
Jan 3 15:38:37.689: ISDN BR0/1/0 Q931: RX <- RELEASE pd = 8 callref = 0x44
Jan 3 15:38:37.689: ISDN BR0/1/0 Q931: TX -> RELEASE_COMP pd = 8 callref = 0xC4
```

```
dial-peer voice 17 voip
call-block translation-profile incoming busy
call-block disconnect-cause incoming user-busy
incoming called-number 90179017
dtmf-relay rtp-nte
codec g711ulaw
!
voice translation-profile busy
translate called 17
!
voice translation-rule 17
rule 1 reject /90179017/
```

# Intelligent Call Rejection

## ICM Script / CVP Comprehensive Model

- REFER label can be a numeric label or SIP URI
  - SIP URI is sent as-is
  - Numeric label is resolved using CVP static dial plan
  - Use RF prefix or set ECC user.sip.refertransfer to “y”
- Remember that survivability.tcl traps abnormal disconnects (cause value not 16)
  - Either don't use it, or ...
  - Customise TCL to suppress call recovery on required failure causes, and ...
  - Comment out “leg setupack leg\_incoming” to prevent spurious immediate answer

# Intelligent Call Rejection

## ICM Script / CVP Comprehensive Model

- Survivability.tcl must disconnect rather than perform recovery on initial call setup failures as required
- Necessitates modifying **CVPTransferDone** procedure

```
"ls_004" {
  set msg "***** Call setup failed: CVP number is invalid $displayStr *****"
  LogMsg "DEBUG" $msg
  set recovery 1
  RecoveryActivities
}

"ls_007" {
  set msg "***** Call setup failed: CVP number is busy – disconnecting caller as custom action *****"
  LogMsg "DEBUG" $msg
  fsm setstate CALLER_DISCONNECTED
  leg disconnect $incoming -c 17
}
```

# Intelligent Call Rejection

## CVP Standalone Model

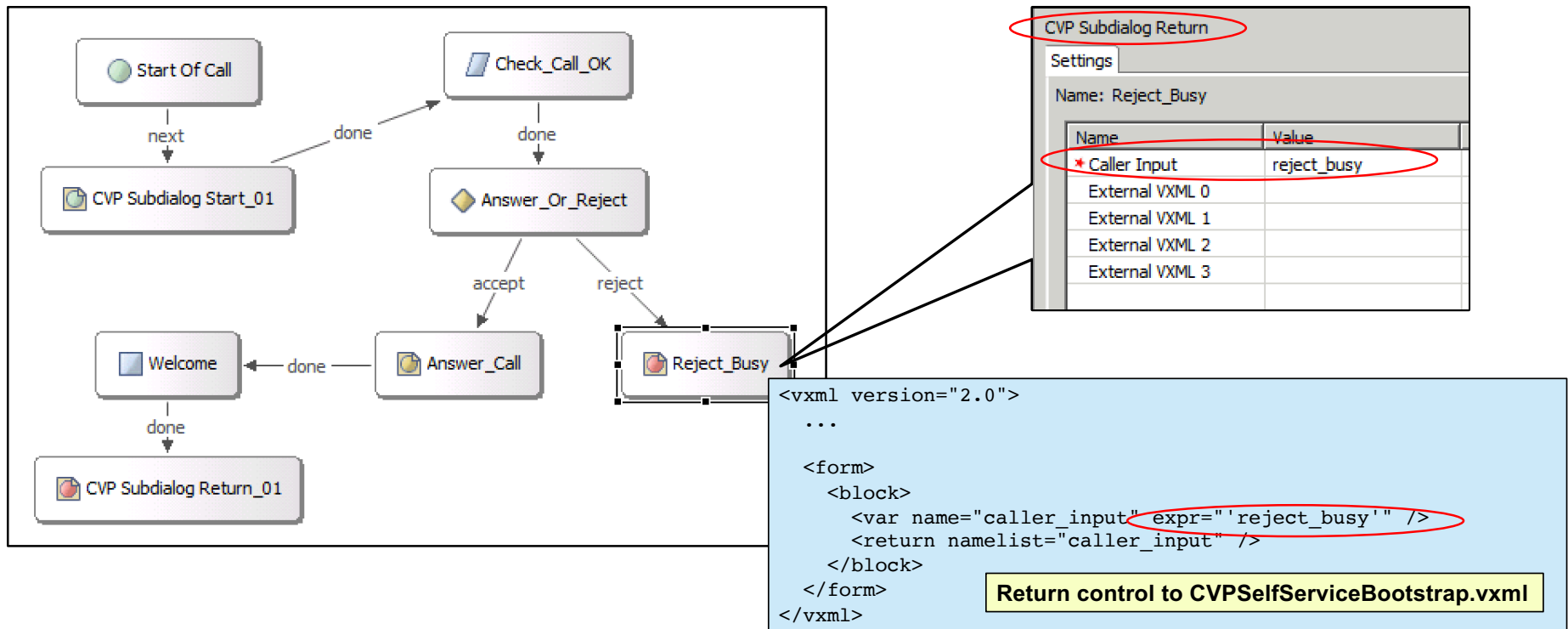
- Should be as simple as disconnecting via VoiceXML

```
<disconnect cisco-disc_cause="17"/>
```
- But this will leave the CVP session hanging until its timeout expires
- Alternative is to let CVPSelfServiceBootstrap.vxml do the disconnect
- Normal call flow:
  - Call Studio application ends via CVP Subdialog Return
  - CVPSelfServiceBootstrap.vxml processes “Caller Input” return parameter
  - By default, will result in Normal Clearing (16)
  - Certain predefined values returned to indicate failures or invoke non-VoiceXML transfers

# Intelligent Call Rejection

## CVP Standalone Model

- Return additional custom defined values on the CVP Subdialog Return



# Intelligent Call Rejection

## CVP Standalone Model

- Modify CVPSelfServiceBootstrap.vxml to process those values
  - reject\_busy
  - reject\_unassigned

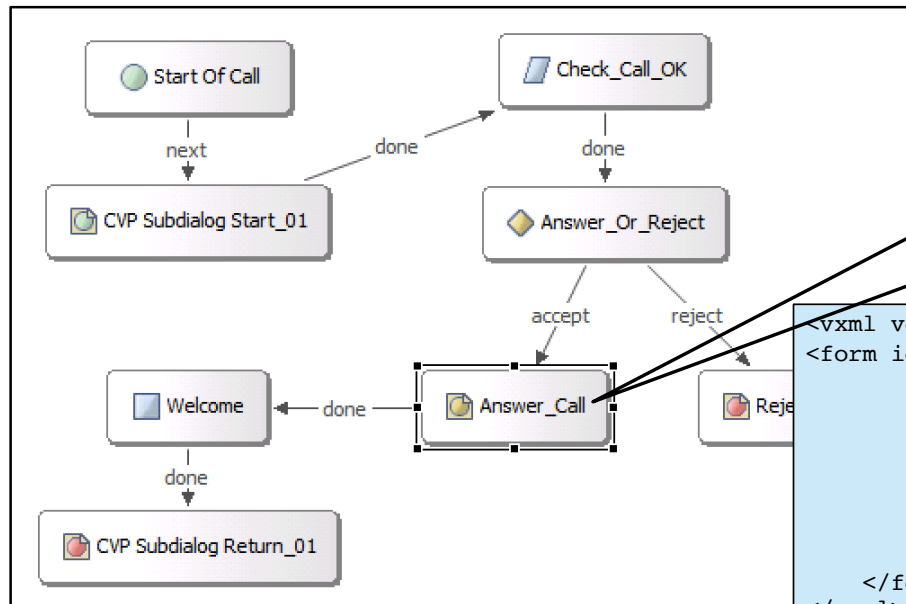
```
<subdialog name="RunCVPServer" ... >
  <filled>
    ...
    <elseif cond="RunCVPServer.caller_input == 'reject_busy'"/>
      <disconnect cisco-disc_cause="17"/>
    <elseif cond="RunCVPServer.caller_input == 'reject_unassigned'"/>
      <disconnect cisco-disc_cause="1"/>
    <else/>
      <disconnect cisco-disc_cause="16"/>
    </if>
  </filled>
</subdialog>
```

```
Jan 3 15:38:37.489: ISDN BR0/1/0 Q931: RX <- SETUP pd = 8 callref = 0x44
Jan 3 15:38:37.525: ISDN BR0/1/0 Q931: TX -> CALL_PROC pd = 8 callref = 0xC4
Jan 3 15:38:37.597: ISDN BR0/1/0 Q931: TX -> DISCONNECT pd = 8 callref = 0xC4
Cause i = 0x8091 - User busy
Jan 3 15:38:37.689: ISDN BR0/1/0 Q931: RX <- RELEASE pd = 8 callref = 0x44
Jan 3 15:38:37.689: ISDN BR0/1/0 Q931: TX -> RELEASE_COMP pd = 8 callref = 0xC4
```

# Intelligent Call Rejection

## CVP Standalone Model

- Not quite finished yet – still two things remaining
  1. Not answering the call automatically when it hits the gateway
  2. Explicitly answering it from the application



Subdialog Invoke	
Settings Data	
Name: Answer_Call	
Name	Value
* Subdialog URI	invoke_answer.vxml
* Local Application	false
Parameter	
Return Value	

```
<vxml version="2.0">
<form id="answer_call">
  <object name="handoff" classid="builtin://com.cisco.callhandoff">
    <param name="return" expr="true"/>
    <param name="app-uri" expr="'builtin://answer'"/>
  </object>
  <block>
    <return/>
  </block>
</form>
</vxml>
```

**invoke\_answer.vxml**

**VoiceXML cannot directly answer the call so use an <object> element to temporarily handoff to a TCL application via service 'answer' to perform 'leg connect'**



# Intelligent Call Rejection

## CVP Standalone Model – Explicitly Answering Call

```
# Answer call and return to calling application -----  
proc handoff_event { } {  
    set leg1 [infotag get evt_legs]  
    puts ">>> ANSWER.TCL assumed control of call leg $leg1"  
    leg connect $leg1  
    handoff return $leg1  
}  
  
# State machine -----  
set FSM(CALL_INIT, ev_handoff)  
set FSM(any_state, ev_disconnect_done)  
set FSM(any_state, ev_disconnected)  
  
fsm define FSM CALL_INIT
```

2. Connect the call  
3. Return control to VoiceXML

1. Handoff from VoiceXML triggers handoff\_event procedure

"handoff\_event, same\_state"  
"cleanup, same\_state"  
"disconnect, same\_state"

Note. Disconnect handling omitted for clarity

- Handoff event handler is invoked via the state machine
- It retrieves the ID of the call leg that it now has control of
- Performs a connect on the call leg
- Returns control back to VoiceXML

# Intelligent Call Rejection

## CVP Standalone Model – Preventing Immediate Answer

- CVPSelfSevice.tcl script has to be modified to not answer the call
- This does mean a minor change to a released/supported CVP TCL script but unfortunately no alternative approach
- Comment-out signalling so that only leg proceeding is executed

```
#-----  
# Procedure act_Start  
#  
# This procedure is executed when it receives an ev_setup indication event.  
# A setup acknowledgement, call proceeding, and connect message are sent to  
# the incoming call leg and finally the call is handed off to the CVPSelfService app.  
#-----  
proc act_Start { } {  
  
    ...  
#     leg setupack leg_incoming  
#     leg proceeding leg_incoming  
#     leg alert leg_incoming  
#     leg connect leg_incoming  
    ...  
}
```

Thank you

