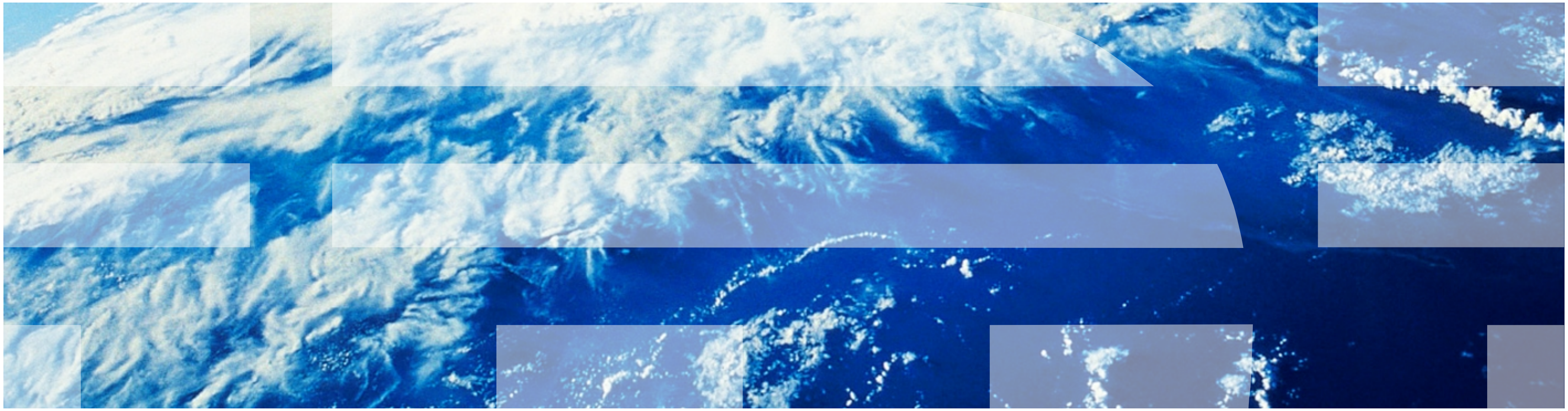
Jennifer Parker

January, 10, 2012

Discovery Stitchers

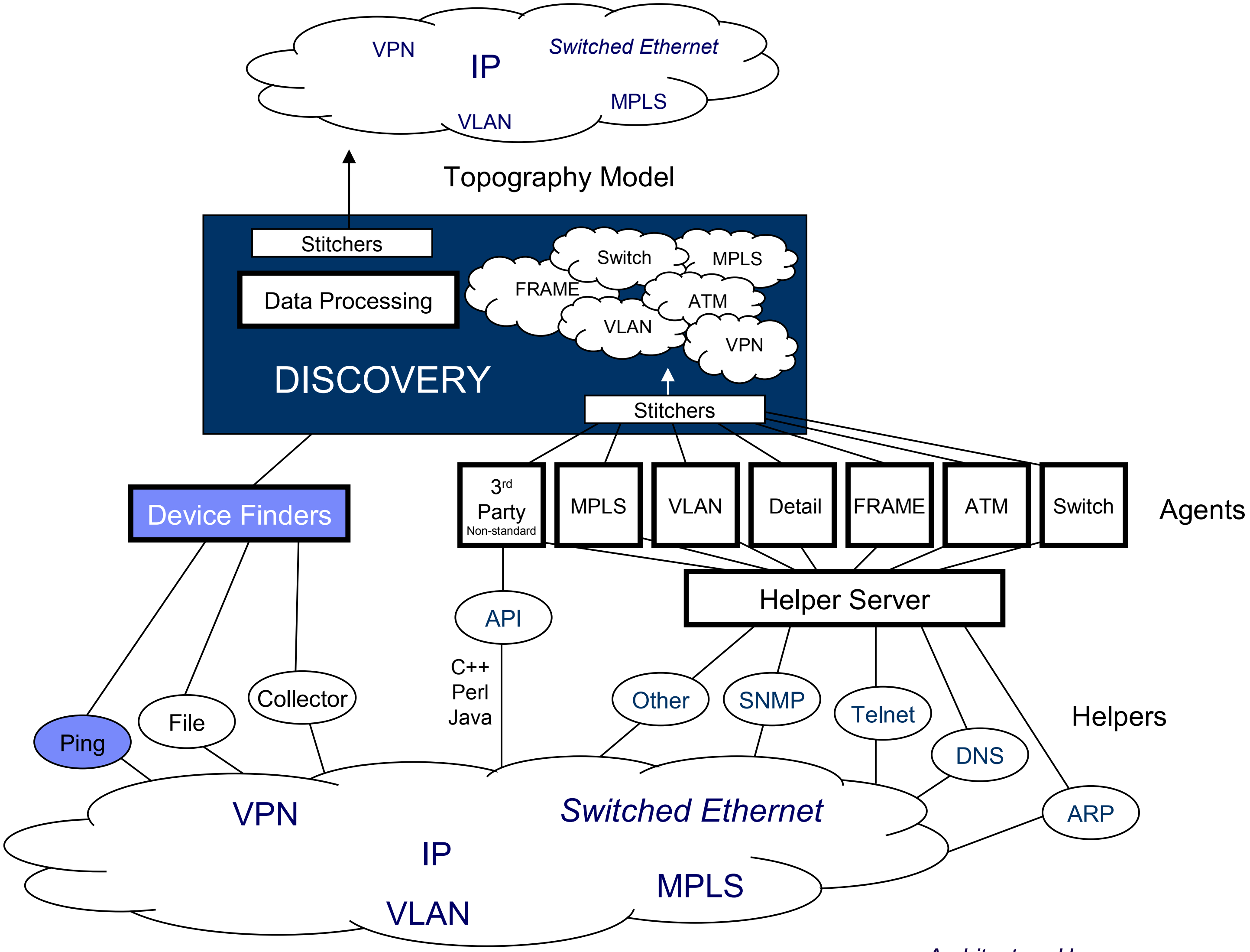


What are stitchers?

- In the context of discovery, stitchers “stitch” the data retrieved during discovery to form a coherent topology

- In the context of the event gateway, stitchers there are four subtypes:
  - Topology lookup stitchers
  - Data extraction stitchers
  - Entity retrieval stitchers
  - Event enrichment stitchers

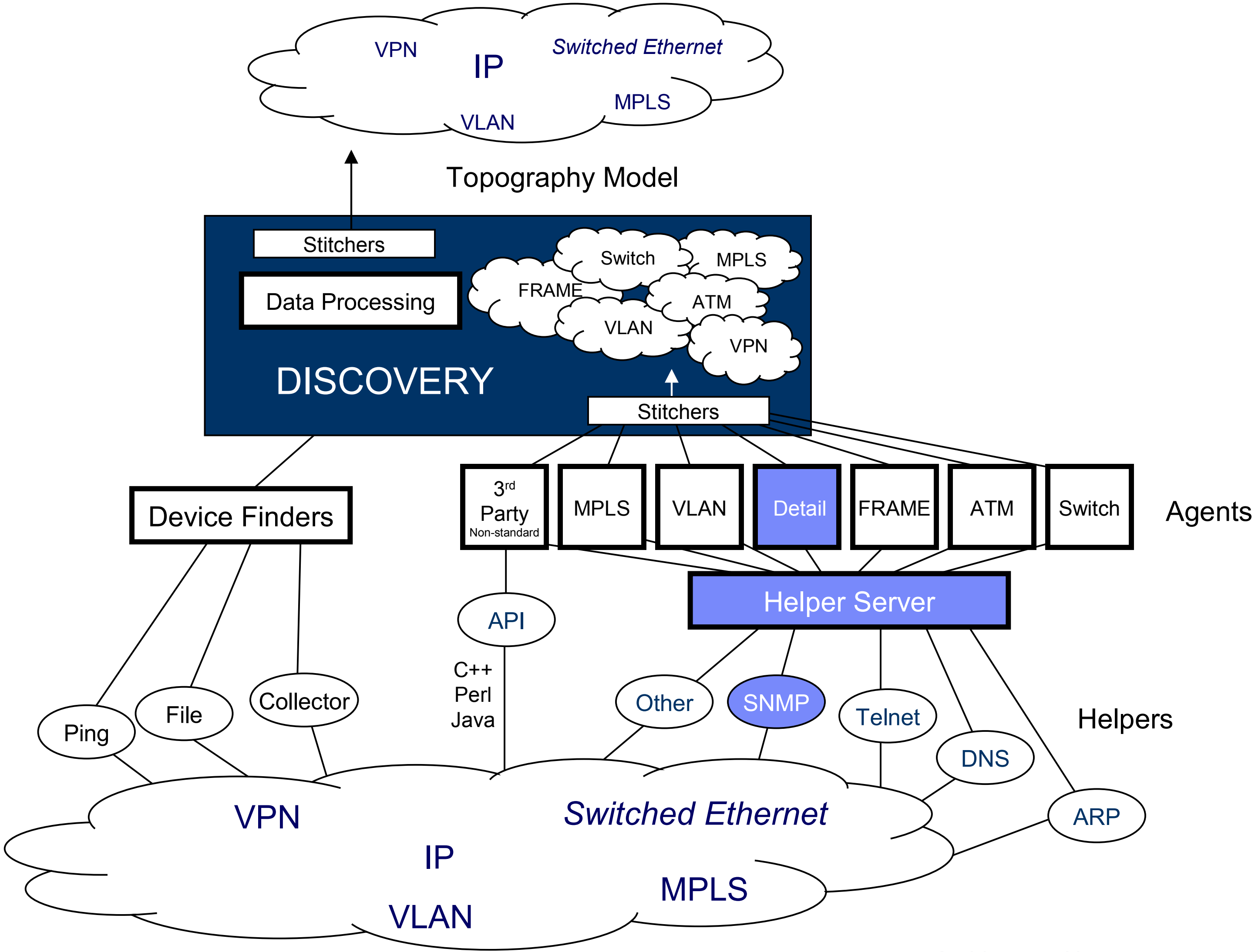
Discovery Process -- Finders



🟦 IP Address

Architecture Home

Discovery Process

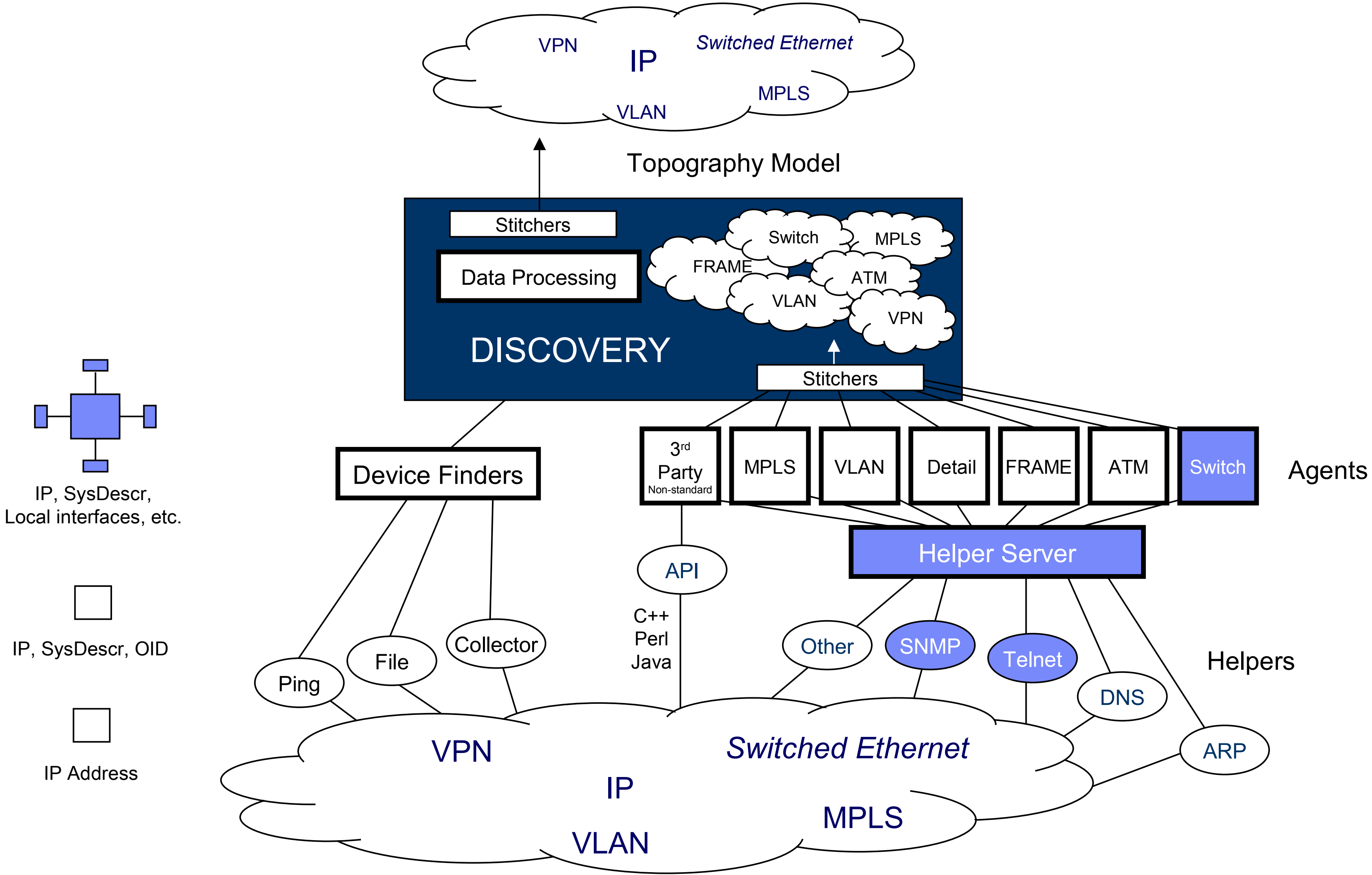


🟦 IP, SysDescr, OID

□ IP Address

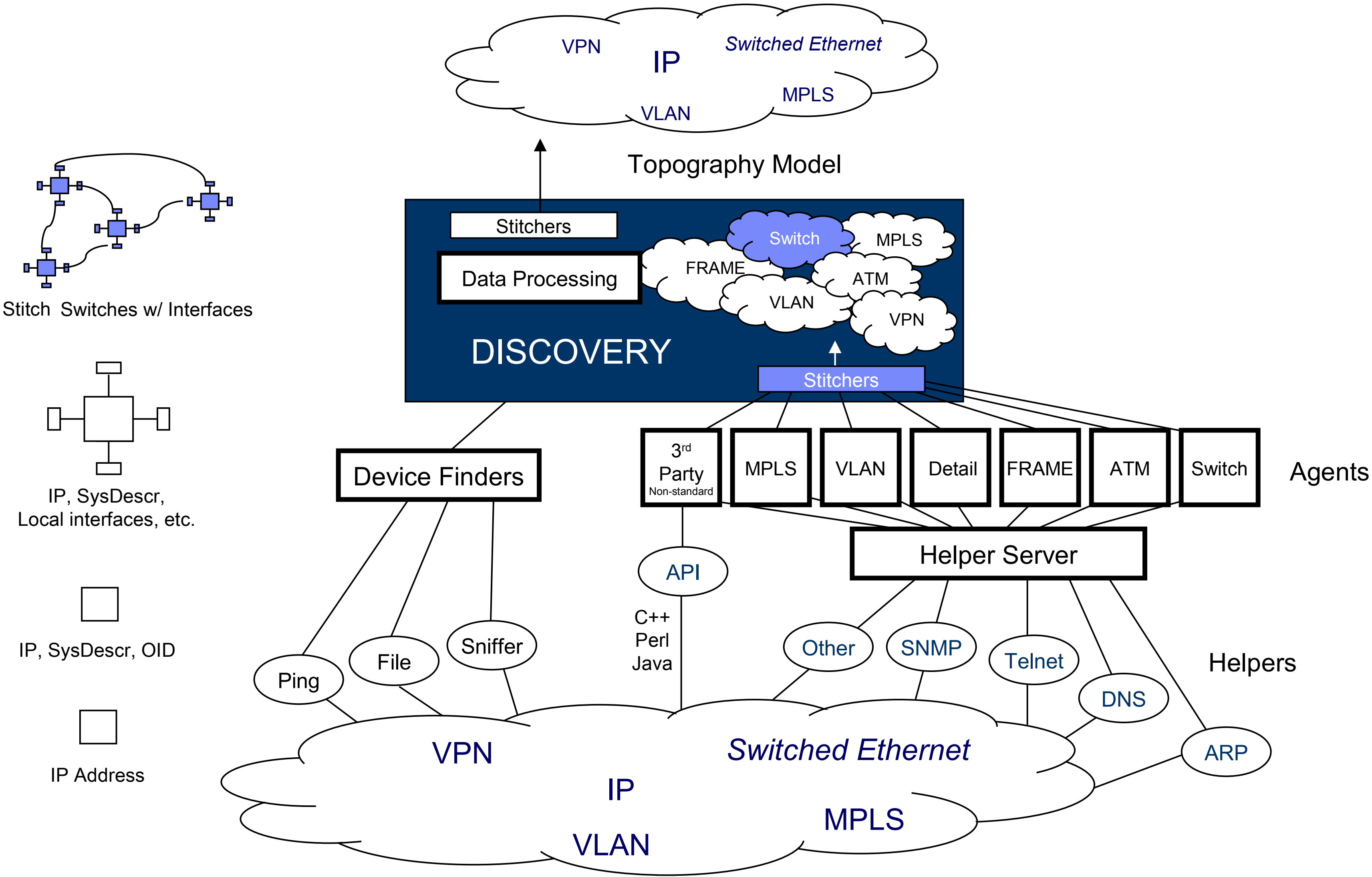
Architecture Home

Discovery Process



Architecture Home

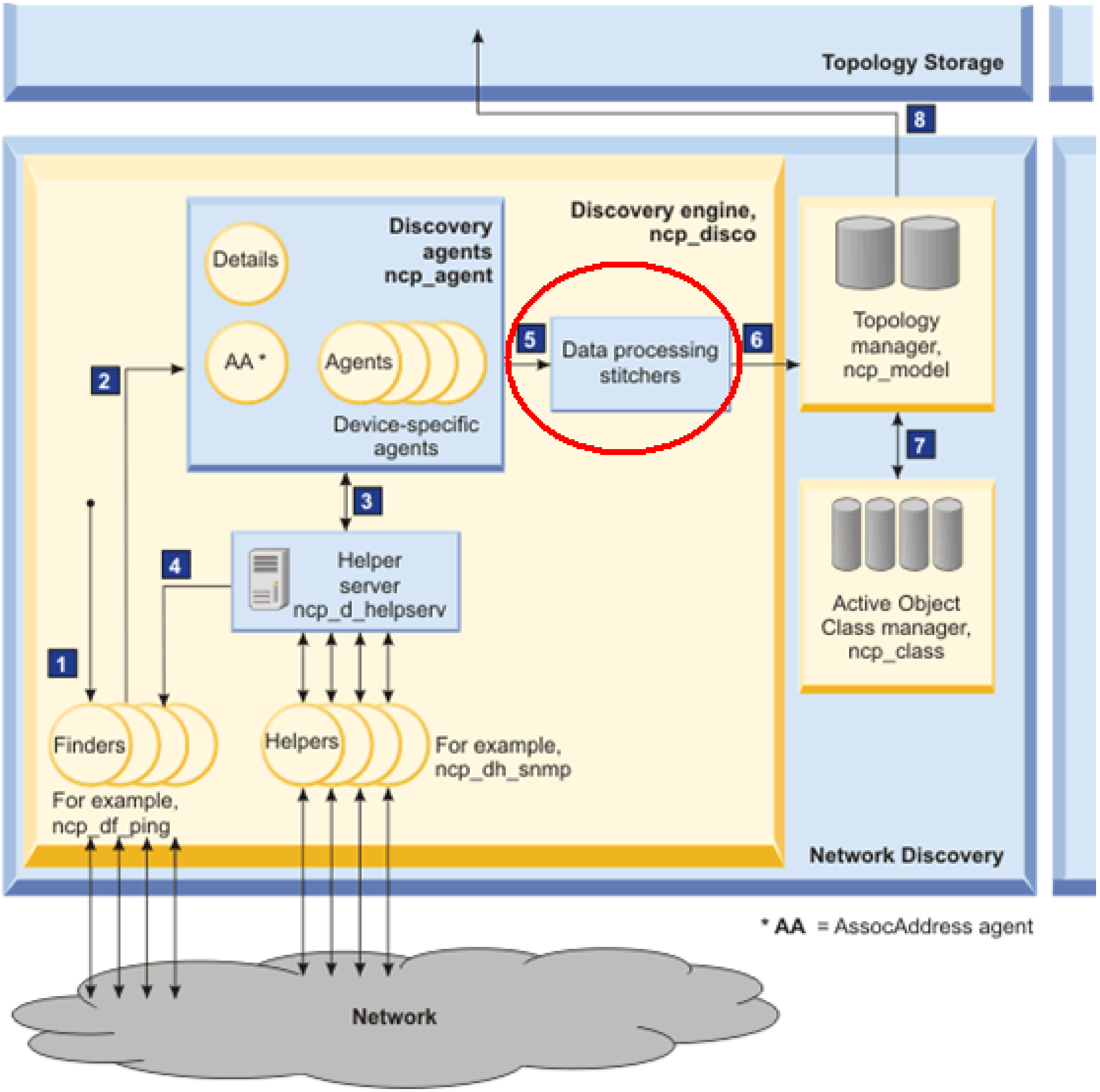
Discovery Process



Architecture Home

Discovery Stitchers – where do they fit in?

The stitchers take the data retrieved by the agents and stitch the topology together, and pass it to the model process.


Topology Storage
8
Discovery agents ncp_agent
Discovery engine, ncp_disco
Details
AA *
Agents
Device-specific agents
5
Data processing stitchers
6
Topology manager, ncp_model
7
Active Object Class manager, ncp_class
2
3
4
Helper server ncp_d_helpserv
1
Finders
For example, ncp_df_ping
Helpers
For example, ncp_dh_snmp
Network Discovery
* AA = AssocAddress agent
Network

What are the stitchers doing when?

- Phase 1 (Data Collection Phase) stitchers:
  - Process the finder and agent returns
  - Decide which device should be handled by which agent
- Phase -1 (Data Processing Phase) stitchers:
  - Move data to working entities
  - Build interface names, labels
  - Create correct containment and connections
  - Merge those pieces of info into a single representation of the topology
  - Unmanage devices and interfaces
  - Sends the topology to model

Why modify stitchers or write a new one?

- Add devices that aren't automatically discoverable
- Change the order of precedence when merging the discovered layers
- Manage or unmanage devices during discovery
- Add data into ExtraInfo to be used later for partitioning data in the gui

Stitcher structure

- StitcherTrigger
  - Conditions which cause the stitcher to run; commonly seen triggers:
    - ActOnDemand – it runs when called, usually in another stitcher, though possibly by a user via oql.
    - ActOnTableInsert – this stitcher will run when an insert is done on a particular table; most often seen with the agent returns tables.
    - ActOnTimedTrigger – the stitcher will be called at a particular time; this is how we schedule a recurring full discovery. Note: You can only have a single timed trigger per stitcher.

- StitcherRules
  - This is the actual code of the stitcher
    - It's a perl-like language.
    - The language and usage is documented in the InfoCenter.

Real World Example: I need to run my discovery twice a week

If I wanted to run my full discovery on Wednesday and Saturday at 11PM, it seems like I should be able to do something like this:

```
/////////////////////////////////////////////////////////////////////
//
//      Full Discovery Stitcher
//
//////////////////////////////////////////////////////////////////////

UserDefinedStitcher
{
        StitcherTrigger
        {
ActOnTimedTrigger(( m_DayOfWeek , m_TimeOfDay )
      values ( 3 , 2300 ) ; ) ;
ActOnTimedTrigger(( m_DayOfWeek , m_TimeOfDay )
      values ( 6 , 2300 ) ; ) ;
}


        StitcherRules
        {
…
```

However, you can only use a single ActOnTimedTrigger per stitcher, thus this will fail.

Solution: Create two stitchers

One is for Wednesday:

```
UserDefinedStitcher
{
          StitcherTrigger
          {
                    ActOnTimedTrigger(( m_DayOfWeek , m_TimeOfDay ) values ( 3 , 2300 ) ; ) ;
          }

          StitcherRules
          {
                    ExecuteStitcher('FullDiscovery');
          }
}
```

And the other is for Saturday:

```
UserDefinedStitcher
{
          StitcherTrigger
          {
                    ActOnTimedTrigger(( m_DayOfWeek , m_TimeOfDay ) values ( 6 , 2300 ) ; ) ;
          }

          StitcherRules
          {
                    ExecuteStitcher('FullDiscovery');
          }
}
```

Real World Example: I want to unmanage all of my loopback interfaces

This customer never wanted to see events from his loopback interfaces

- Can create a filter so loopback ifs wouldn’t be polled
- Used a filter in the AEL to show only NmosManagedStatus = 0
- What if a new poll policy is created, and the operator didn’t know to put in the interface filter?

Solution: Unmanage the loopbacks in TagManagedEntities

```
//----------------------------------------------------------------
// Modify this statement to unmanage specific types of interface
//----------------------------------------------------------------
oqlUpdate = oqlUpdate + interfaceFilter + rediscoveryFilter +
            " AND
                 (
                     ExtraInfo->m_IfDescr like 'Dialer'
                     OR
                     ExtraInfo->m_IfDescr like 'Async'
                     OR
                     ExtraInfo->m_IfDescr like 'Virtual'
                     OR
                     ExtraInfo->m_IfDescr like 'Null'
                     OR
                     ExtraInfo->m_IfDescr like 'NULL'
                     OR
                     ExtraInfo->m_IfDescr like 'Vlan'
                     OR
                     ExtraInfo->m_IfDescr like 'VLAN'
                     OR
                     ExtraInfo->m_IfAlias like 'NoMon'
                     OR
                     ExtraInfo->m_IfType = 24

                 )
            );";

ExecuteOQL(oqlUpdate);
```

Questions?