ALTO Topology Extension

draft-yang-alto-topology-05

G. Bernstein (gregb@grotto-networking.com)

M. Scharf (michael.scharf@alcatel-lucent.com)

Young Lee (leeyoung@huawei.com)

W. Roome (w.roome@alcatel-lucent.com)

Xiao Shi (xiao.shi@yale.edu)

Y. Richard Yang (yry@cs.yale.edu)

November 13, 2014 @ IETF 91

Document Status

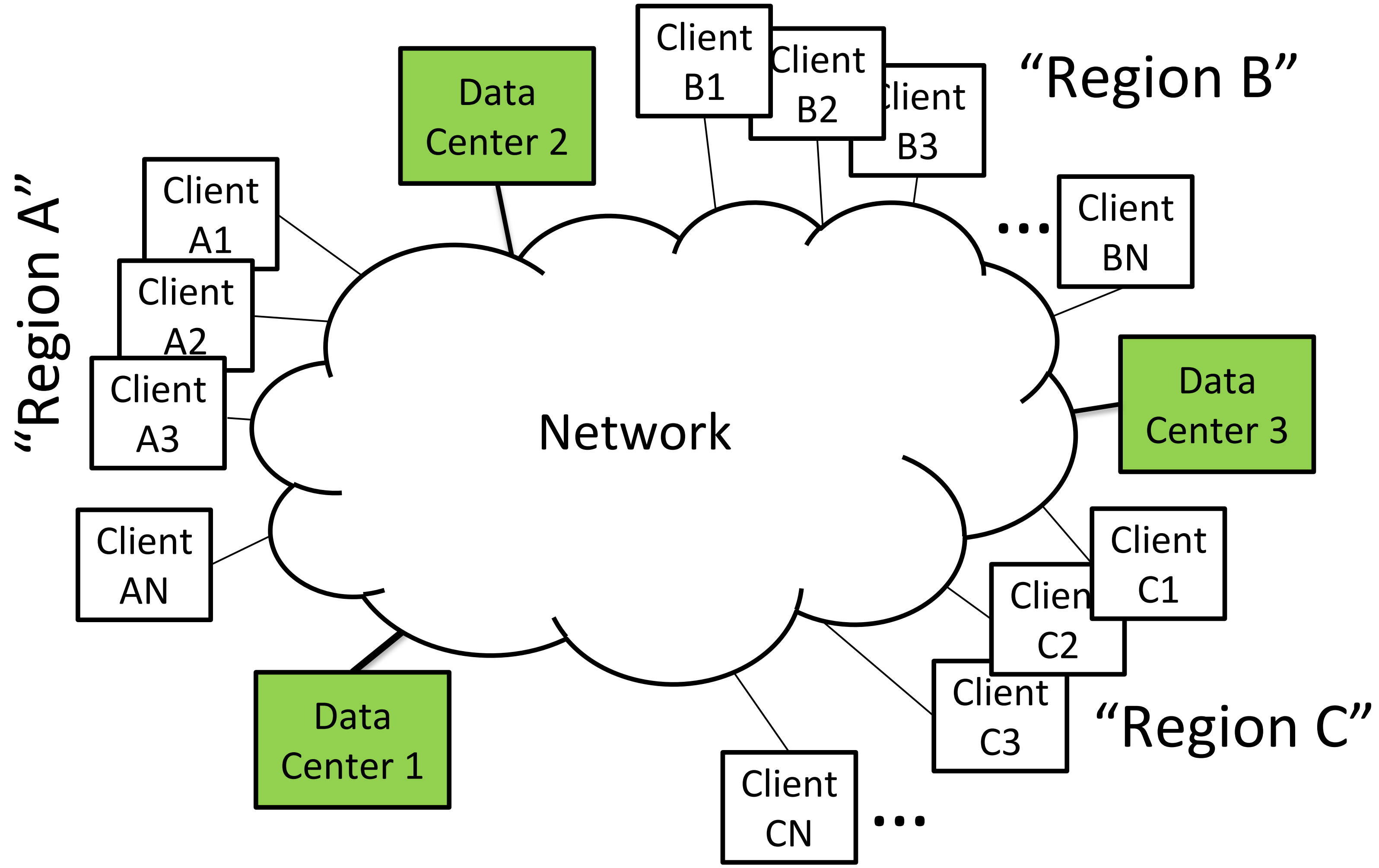
- Merged (`draft-scharf-alto-topology-00` and `draft-yang-alto-topology-03`).

- The goal is to address charter milestone (Jul 2015 - Submit network graph format document).

Changes from –03 to –05

- Revise the multi-flow use case example (Section 3) to better illustrate the need of revealing individual network elements.
- Extend from path vector to flow to allow multipath routing (to be added in -06)
- Revise the use cases of providing a graph (node-link) representation (Section 5) in ALTO
  - Compact representation
  - Application path selection
- Add discussion on using property graph (Section 6.4)
- Add discussion on issues of pure hierarchical design
  - Motivated by discussions at IETF90 on next steps
    - "Evaluate current design on extensibility, e.g., supporting additional requirements such as Hierarchy, hyperedges, port, dynamics, selector (filtering) API"
- Provide a YANG model

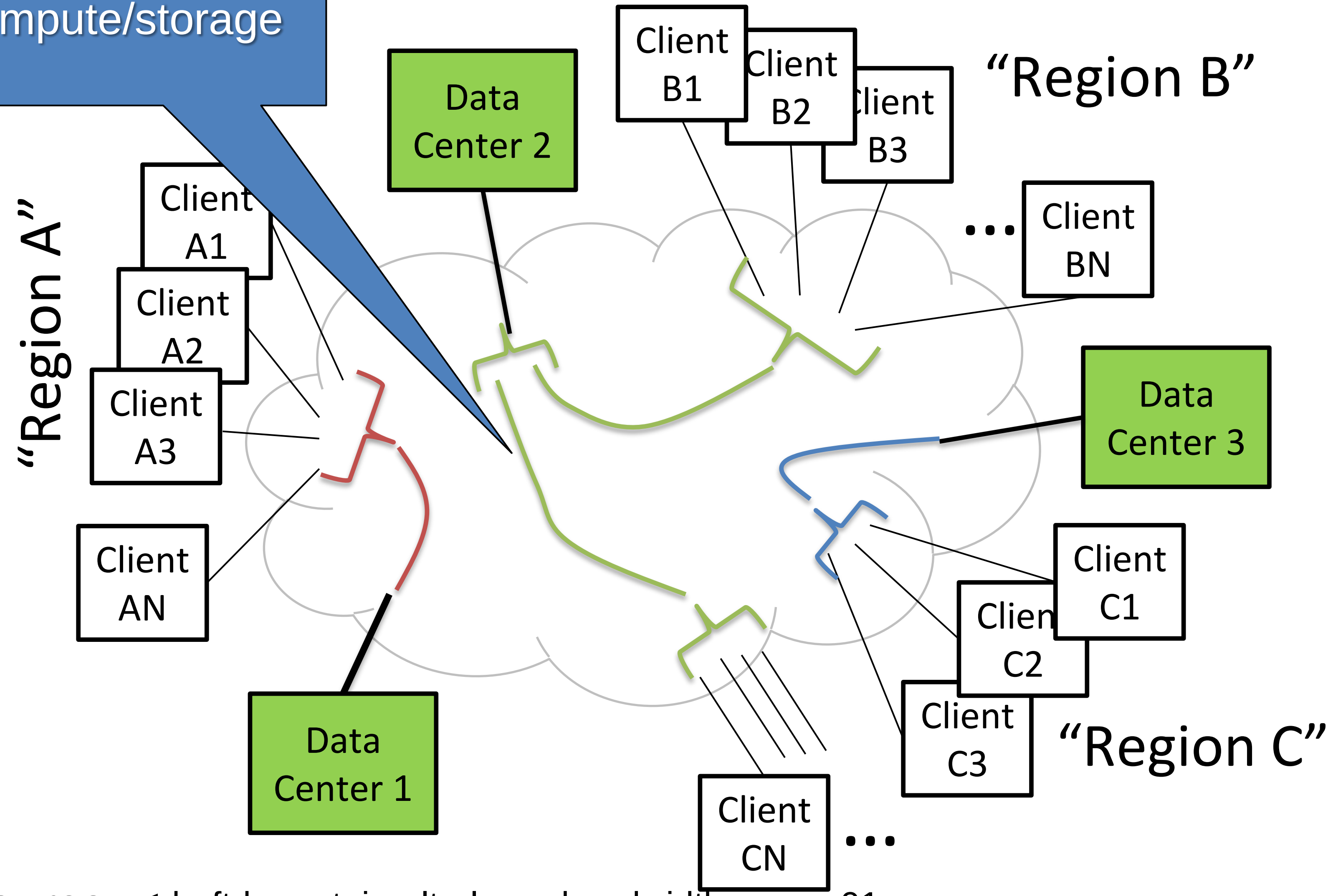
Multi-flow Scheduling: More General Setting



Source: <draft-bernstein-alto-large-bandwidth-cases-01>

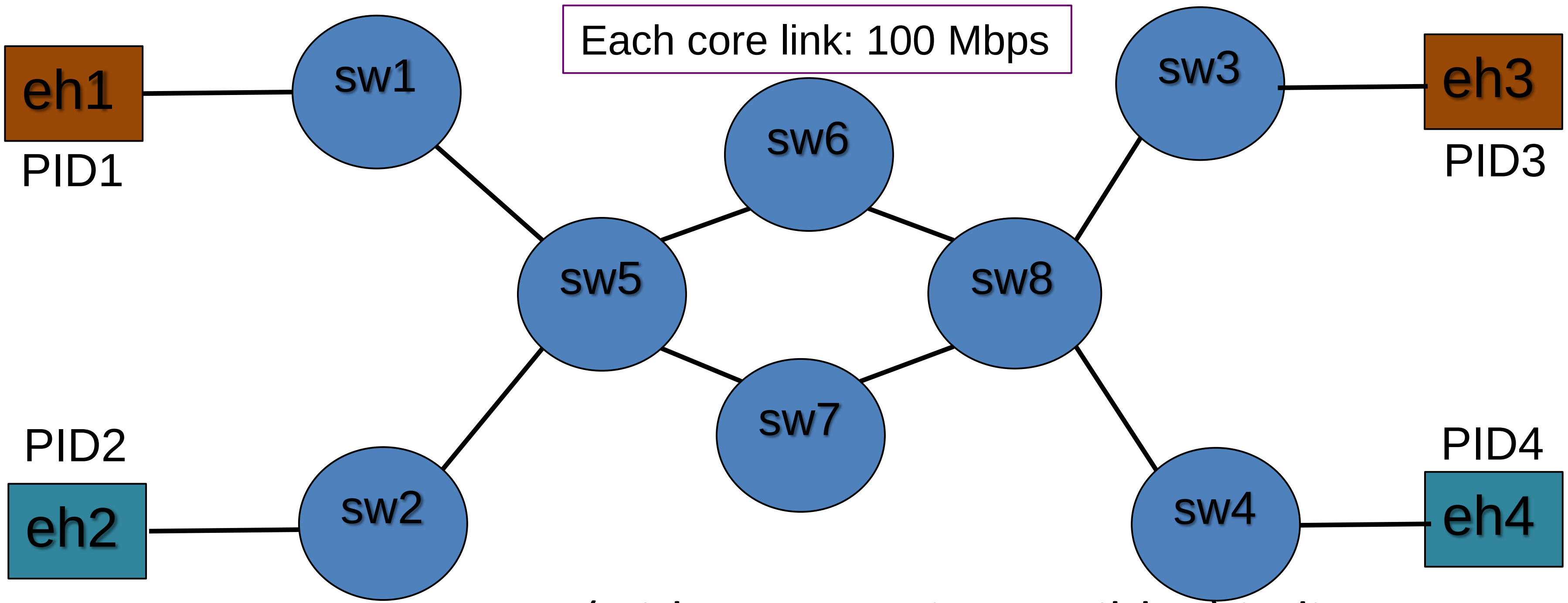
Multi-flow Scheduling

Chosen paths considered both net and compute/storage



Source: <draft-bernstein-alto-large-bandwidth-cases-01>

Multi-flow Scheduling



- Assume ALTO cost map (with cost metric = avail-bw) indicates:
  - PID1 (eh1) -> PID3 (eh3): 100 Mbps; PID2 (eh2) -> PID4 (eh4): 100 Mbps
- Ambiguity:
  - Two disjoint paths (200 Mbps when concurrent)
    - PID1 -> PID3: sw5 -> sw6 -> sw8;
    - PID2 -> PID4: sw5 -> sw7 -> sw8
  - Shared bottleneck (still 100 Mbps when concurrent)

General App-Net Joint Opt. Setting

- App
  - Computes a1 and a2 amounts of resources on path PID1->PID3, and path PID2->PID4 respectively, according to app goal and constraints
  - Two types app constraints
    - Non-network constraints
    - Network constraints
      - a1 <= avail-bw1
      - a2 <= avail-bw2

- Missing network constraints:
  - The amount a1 will map to P1 = {a11, a12, a13…} amount of resource on each element along the path of PID1->PID3; same for PID2->PID4.
  - Topology abstraction does not reveal P1 or P2, and hence, the client does not know if a1 -> P1; a2 -> P2 in parallel will violate constraints (e.g., when P1 ^ P2 is not empty).

Design Choice I: Path Vector

- Providing path vector as a new cost mode for ALTO cost map so that application can be informed of individual network elements along network chosen path, through which it can infer shared bottlenecks, shared reliability risk, etc.

```
object {
        cost-map.DstCosts.JSONValue -> JSONString<0,*>;
        meta.cost-mode = "path-vector”;
} InfoResourcePVCostMap : InfoResourceCostMap;
```

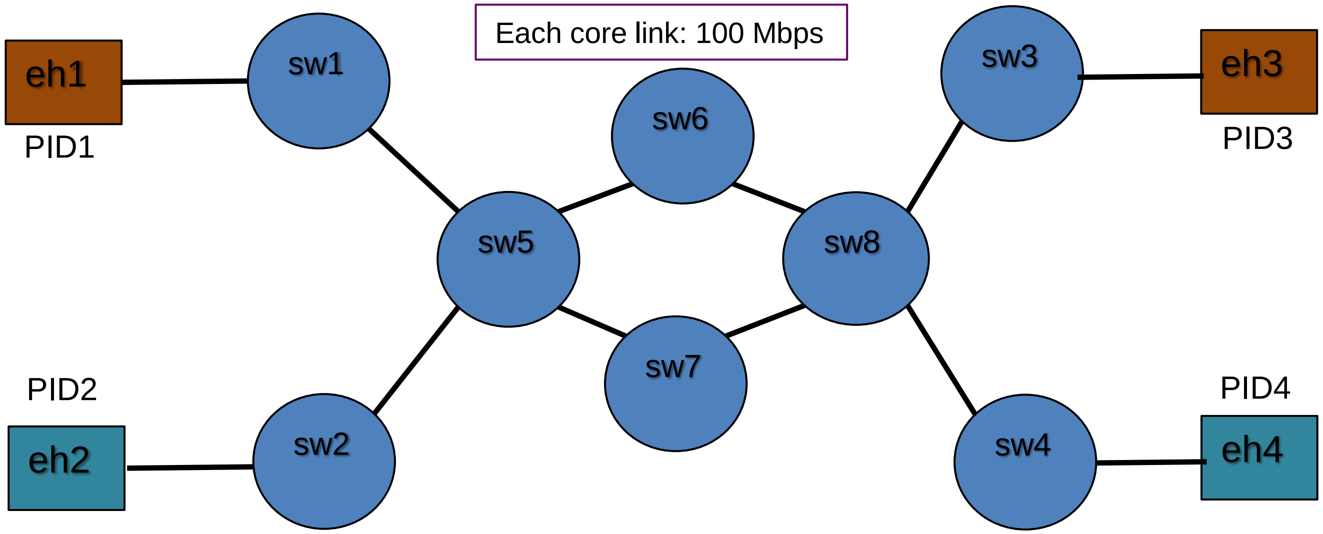
Path Vector: Example

```
HTTP/1.1 200 OK
Content-Length: TDB
Content-Type: application/alto-costmap+json

{  "meta" : {
     "dependent-vtags" : [
        { "resource-id": "my-default-network-map”,
           "tag": "3ee2cb7e8d63d9fab71b9b34cbf764436315542e” },
        {"resource-id": "my-topology-map",
           "tag": "4xee2cb7e8d63d9fab71b9b34cbf76443631554de"
        }
      ],
      "cost-type" : {"cost-mode"  : "path-vector” } },

      "cost-map" : {
         "PID1": { "PID1":[], "PID2":["ne56", "ne67"], "PID3":[], "PID4":["ne57"]
          },
          "PID2": { "PID1":["ne75"], "PID2":[], "PID3":["ne75"], "PID4":[]
          }, …
     }
}
```

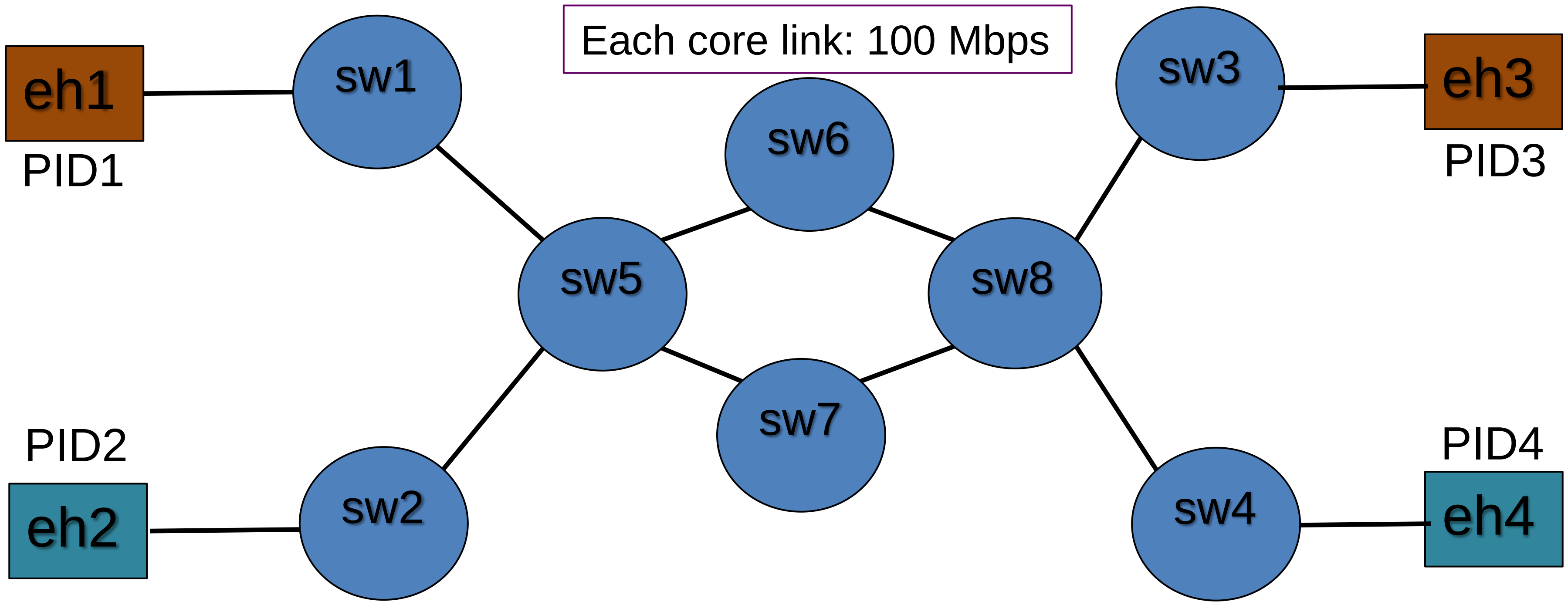
Design Choice 2: General Flow

- Handle the case of multi-path routing such as ECMP

```
object {
   JSONNumber w;      // flow weight
   JSONString    ne;      // network element
} FlowElement;

object {
       cost-map.DstCosts.JSONValue -> FlowElement<0,*>;
       meta.cost-mode = ”flow”;
} InfoResourcePVCostMap : InfoResourceCostMap;
```



Multi-flow Scheduling



- ECMP for eh1 -> eh3, single path through sw6 for eh2 -> eh4
  - PID1 -> PID3: [{“ne”: “sw5-6”, “w”: 0.5},
    {“ne”: “sw6-8”, “w”: 0.5},
    {“ne”: “sw5-7”, “w”: 0.5},
    {“ne”: “sw7-8”, “w”: 0.5}]
  - PID2 -> PID4: : [{“ne”: “sw5-6”, “w”: 1},
    {“ne”: “sw6-8”, “w”: 1}]

Discussion: Design Choices

1. Both path vector and flow
2. Extend the path vector mode (no flow mode):
   PIDi -> PIDj: an object of two arrays

```
"cost-map" : {
            "PID1": { "PID1": { “pv”: [], “w”: [] },
                          "PID2": { “pv”: ["ne56", "ne67"], “w”: [1, 1] },
             },
…
```

3. Extend the flow mode (no path vector mode):
   PIDi -> PIDj: an array of objects each with two elements

```
"cost-map" : {
            "PID1": { "PID1": [ {“pv”: [], “w”: [] },
                          "PID2": { “pv”: [{“ne”: "ne56", “w”: 1},
                                                    {"ne”: “ne67”, “w”: 1 }
             },
…
```

Summary

- Path-vector, flow, and NetElementPropertyMap together form a relatively simple, complete routing abstraction specification for networks with **given** routing
  - **With them, app can derive network constraints properly**

- NetElementPropertyMap does not need to distinguish between links and nodes

- A main issue:
  - Path vector does not scale well: size is O(N^2 P), where N is # PIDs, P is avg # of hop counts.

Improving Scalability: Graph + PV

- Approach
  - Node-link graph provides default shortest path (or ECMP) routing
  - Path vector/flow encodes only exceptions to a well-known routing

```
HTTP/1.1 200 OK
Content-Length: TDB
Content-Type: application/alto-costmap+json

{  "meta" : {
     "dependent-vtags" : [
        { "resource-id": "my-default-network-map”,
          "tag": "3ee2cb7e8d63d9fab71b9b34cbf764436315542e” },
        { "resource-id": "my-topology-map",
          "tag": "4xee2cb7e8d63d9fab71b9b34cbf76443631554de"
        }
      ],
      "cost-type" : {"cost-mode"  : "path-vector” } }, // need to indicate routing alg
      "cost-map" : {
         "PID1": {"PID2": ["ne56", "ne67”]},
       }
    }
}
```

Discussion: Graph Selector API

- Only example: select the union of nodes and links that are on the path vectors of a given set of src/dst PIDs
  - Did not consider joint app/net traffic engineering; assume fixed routing
  - Two types of app constraints
    - Non-network constraints [unknown to network]
    - Network constraints [assumed to be known to network]
      - a1 <= avail-bw1
      - a2 <= avail-bw2
    - Network computes minimal abstract topology to make sure app can compute the correct network constraints

- Other selectors?

Remaining Issue: Node-Link Graph Detailed Design

Design Choice I: Unified PID and Network Node

- Define all nodes in Network Map; PID properties provide details

```
HTTP/1.1 200 OK
Content-Length: TBD
Content-Type: application/alto-networkmap+json

{
  "meta": {
    ...
  },
  "network-map": {
    "H1": { "ipv4": [ "10.0.1.0/24" ] },
    "H2": { "ipv4": [ "10.0.2.0/24" ] },
    "H3": { "ipv4": [ "10.0.3.0/24" ] },
    "H4": { "ipv4": [ "10.0.4.0/24" ] },
    ”sw1": { }, ”sw2": { }, ”sw3": { }, ”sw4": { }, ”sw5": { }, ”sw6": { }, ”sw7": { },
    "Default": {
      "ipv4": [ "0.0.0.0/0" ], "ipv6": [ "::/0" ]
    }
  }
```

Design Choice II: Separate PID and Network Node

```
{
 "nodes": [
   "n0" : {
    },
   "n1" : {
   },
   ...
 ],
 "edges": [
  { "src": "node:n0",
    "dst": "node:n1",
    "type": "directed",
    "cost" : [ {
       "cost-metric" : "delay",
       "value" : "3"
     }, {
       "cost-metric" : "availbw",
       "value" : "50"
     }, {
       "cost-metric" : "risk-group",
       "value" : ["SLRG3"]
    } ]
  },
  ...
 ]
}
```

Abstract network nodes in graph

Abstract network edges in graph

Node/edge properties, such as multiple TE metrics from draft-wu-alto-te-metrics

Node-edge

```
GET /costmap/pathvec HTTP/1.1
Host: alto.example.com
Accept: application/alto-costmap
    +json,application/alto-error+json
```

```
HTTP/1.1 200 OK
Content-Length: TBA
Content-Type: application/alto-costmap+json

{
  "meta" : {
   ...
    "cost-type" : {"cost-mode"  : "path-vector"}
  },
  "cost-map" : {
   "PID1": { "PID1": [ ],
             "PID2": ["n0",  "n1"]
             "PID3":  ["n0"] },
   "PID2": { ...},
   "PID3": { ...}
 }
```

Vector of network nodes traversed.

Path-vector Cost Map

Discussion: Which Design?

- Unified PID and network node
- Separate network node from PID
- Currently favors a separation design
  - Link: transmission without switching
  - Node: ability to switch from incoming link to outgoing link(s)
  - PID is a set of endpoint groups

Implication of Choice II: Different Types of Links

- There are two types of links
  - Between two network nodes vs
  - Between a network node and a PID
- Indicate by link type

```
"links" : {
  "e1" : {"type": "pid-attach",
            "src" : "PID1", "dst": "sw1",
            "costs": ..
  },
  "e2" : {"type": "transit",
           "src" : "sw1", "dst": "sw2",
           "costs": ..
  },
  …
}
```

Bigger Context: Link Type as Property Graph

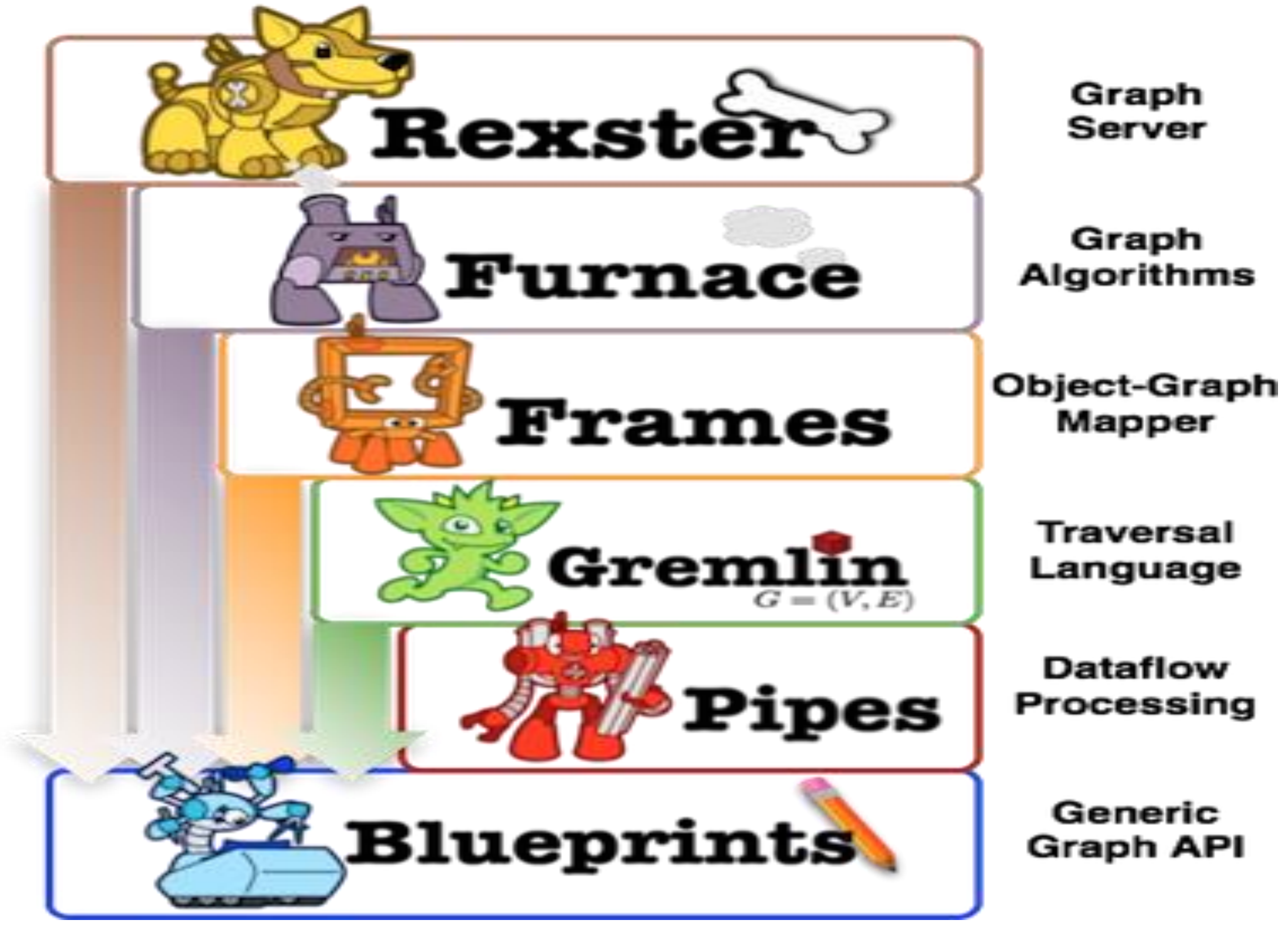
- The "separation design" is a type of property graph
  - A graph consists of a set of element objects, where each object can have any number of key/value pairs associated with it (i.e., properties)
    - Vertex: an object with a unique identifier.
      - each vertex has a set of outgoing edges.
      - each vertex has a set of incoming edges.
      - [each vertex has a collection of properties defined by a map from key to value]
    - Edge: also an object with a unique identifier
      - each edge has an outgoing tail vertex.
      - each edge has an incoming head vertex.
      - each edge has a label that denotes the type of relationship between its two vertices.
      - [each edge has a collection of properties defined by a map from key to value.]

YANG Model

```
module: alto-service-topology
  +--ro resources
     +--ro topology-maps*
        +--ro topology-map [resource-id]
           +--ro resource-id    alto:resource-id
           +--ro tag            alto:tag-string
           +--ro nodes* [node-id]
           |  +--ro node-id                node-id
           |  +--ro node-properties* [property-name]
           |     +--ro property-name     string
           |     +--ro property-value?   string
           +--ro links* [link-id]
              +--ro link-id    link-id
              +--ro src        union
              +--ro dst        union
              +--ro type       string
              +--ro cost* [cost-metric]
                 +--ro cost-metric    alto:cost-metric
                 +--ro value
```

Support for Graph Traversal and Selection

- Benefit: a format that would facilitate easy, advanced graph computation by applications, using existing software tools



Hierarchy/Abstraction

- Q: Do we extend the basic node-link informational resource to convey hierarchy (i.e., make hierarchy/abstraction client aware)?

Extensibility

- Selector (filtering) API
- Dynamics (calendaring)
- Hierarchy
- Hyperedges
- Port

Data-Driven ALTO Topology Model: A YANG Model

YANG Model XML vs Gremlin Formats

```
<topology-maps>
  <topology-map>
    <resource-id>my-default-topology-map</resource-id>
    <tag>abcd1234</tag>
    <nodes>
      <node-id>sw1</node-id>
      <node-properties>
        <property-name>color</property-name>
        <property-value>green</property-value>
      </node-properties>
    </nodes>
    <nodes>
      <node-id>sw2</node-id>
      <node-properties>
        <property-name>color</property-name>
        <property-value>blue</property-value>
      </node-properties>
    </nodes>
    <nodes>
      <node-id>sw3</node-id>
      <node-properties>
        <property-name>color</property-name>
        <property-value>blue</property-value>
      </node-properties>
    </nodes>
```

YANG Model XML vs Gremlin Formats

```
 <links>
  <link-id>e1</link-id>
  <src>sw1</src>
  <dst>sw2</dst>
  <type>transit</type>
  <cost>
    <cost-metric>avail-bw</cost-metric>
    <value>100</value>
  </cost>
  <cost>
    <cost-metric>delay</cost-metric>
    <value>20</value>
  </cost>
<links>
<links>
  <link-id>e2</link-id>
  <src>sw2</src>
  <dst>sw3</dst>
  <type>transit</type>
  <cost>
    <cost-metric>avail-bw</cost-metric>
    <value>200</value>
  </cost>
  <cost>
    <cost-metric>delay</cost-metric>
    <value>50</value>
  </cost>
<links>
<links>
  <link-id>e3</link-id>
  <src>PID1</src>
  <dst>sw3</dst>
  <type>pid-attach</type>
  <cost>
    <cost-metric>avail-bw</cost-metric>
    <value>300</value>
  </cost>
  <cost>
    <cost-metric>delay</cost-metric>
    <value>10</value>
  </cost>
<links>
</topology-maps>
```

Gremlin Formats (GraphML)

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
        http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">

  <!-- definition for properties -->
  <key id="color" for="node" attr.name="color" attr.type="string" />
  <key id="avail-bw" for="edge" attr.name="avail-bw" attr.type="int" />
  <key id="delay" for="edge" attr.name="delay" attr.type="int" />

  <graph id="my-default-topology-map" edgedefault="directed">
    SEE NEXT SLIDES
  </graph>

</graphml>
```

Gremlin Formats (GraphML)

```
<node id="PID1"/>            <!-- PIDs from network map -->
<node id="sw1">              <!-- nodes from topology map -->
  <data key="color">green</data>
</node>
<node id="sw2">
  <data key="color">blue</data>
</node>
<node id="sw3">
  <data key="color">blue</data>
</node>

<edge id="e1" source="sw1" target="sw2" label="transit">
  <data key="avail-bw">100</data>
  <data key="delay">20</data>
</edge>
<edge id="e2" source="sw2" target="sw3" label="transit">
  <data key="avail-bw">200</data>
  <data key="delay">50</data>
</edge>
<edge id="e3" source="PID1" target="sw3" label="pid-attach">
  <data key="avail-bw">300</data>
  <data key="delay">10</data>
</edge>
```

YANG Model XML to Gremlin Format (GraphML)

- YANG network map PIDs -> GraphML nodes
- YANG topology map: nodes and edges -> GraphML nodes and edges
  - Node property name, value -> GraphML key for nodes
    - E.g. `<key id=“color" for="node" attr.name="color" attr.type="string“ />`
    - `<node id="sw2"> <data key="color">blue</data> </node>`
  - Edge type -> GraphML edge label
  - Edge cost -> GraphML key for edges
    - E.g. `<key id="delay" for="edge" attr.name=“delay” attr.type="int" />`

```
<edge id="e2" source="sw2" target="sw3" label="transit">
  <data key="delay">50</data>
</edge>
```

Backup Slides

Initial ALTO Node-Edge Graph Design: Requirements

- Be compatible with and reuse existing ALTO information resources such as Network Maps
- Be simple, yet extensible/general to encode diverse network graphs to applications
- Be modular

Initial ALTO Node-Edge Graph Design: Nodes

- Defined in Network Map; PID properties provide details

```
HTTP/1.1 200 OK
Content-Length: TBD
Content-Type: application/alto-networkmap+json

{
  "meta": {
    ...
  },
  "network-map": {
    "H1": { "ipv4": [ "10.0.1.0/24" ] },
    "H2": { "ipv4": [ "10.0.2.0/24" ] },
    "H3": { "ipv4": [ "10.0.3.0/24" ] },
    "H4": { "ipv4": [ "10.0.4.0/24" ] },
    ”sw1": { }, ”sw2": { }, ”sw3": { }, ”sw4": { }, ”sw5": { }, ”sw6": { }, ”sw7": { },
    "Default": {
      "ipv4": [ "0.0.0.0/0" ], "ipv6": [ "::/0" ]
    }
  }
```

Initial ALTO Node-Edge Graph Design: Edges

- Defined in a new InfoResource named Edge Map

```
HTTP/1.1 200 OK
Content-Length: TDB
Content-Type: application/alto-edgemap+json

{  "meta" : {
      "dependent-vtags" : [
       { "resource-id": "my-default-network-map”,
          "tag": "3ee2cb7e8d63d9fab71b9b34cbf764436315542e” },
       {"resource-id": "my-default-edge-map",
        "tag": "4xee2cb7e8d63d9fab71b9b34cbf76443631554de"
       }],
       “edge-default” : “undirected”, “edge-label-default” : “connect”,
     }
      ”edge-map" : {
         “e1” : {“source” : “PID1”, “target”: “sw1”, “cost”: {}}
         …
      }
}
```

single-relational graph as default, allows multi-relational in future

Next Step

- Evaluate current design on extensibility, e.g., supporting additional requirements:
  - Hierarchy, hyperedges, port, dynamics, selector (filtering) API

- Target: a complete version by IETF 91 for WG review.

Backup Slides: Survey of Graph Tools/Formats

GraphViz

- A format used in graph visualization
- Example

```
graph hello2 {

  // Hello World with nice colors and big fonts

  Node1 [label="Hello, World!", color=Blue, fontcolor=Red,
      fontsize=24, shape=box]

 B [label="The boss"]      // node B
 E [label="The employee"]  // node E

 B->E [label="commands", dir=back, fontcolor=red]
 // revert arrow direction
}
```

NetworX

- NetworkX uses a “dictionary of dictionaries of dictionaries” as the basic network graph data structure
  - The keys are nodes so G[u] returns an adjacency dictionary keyed by neighbor to the edge attribute dictionary.
- **Example: an undirected graph with the edges (‘A’,’B’), (‘B’,’C’)**
  - **>>> G=nx.Graph()**
  - **>>> G.add_edge(’A’,’B’)**
  - **>>> G.add_edge(’B’,’C’)**
  - **>>> print(G.adj)**
  - **{’A’: {’B’: {}}, ’C’: {’B’: {}}, ’B’: {’A’: {}, ’C’: {}}}**

- The format allows fast lookup with reasonable storage for large sparse networks.
- Problem if use in ALTO: unrestricted edge properties; no node properties.

GEXF

- Nodes and edges can have attributes, but must be declared
- Example

```
<gexf xmlns="http://www.gexf.net/1.2draft"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.gexf.net/1.2draft
http://www.gexf.net/1.2draft/gexf.xsd" version="1.2">
  <meta lastmodifieddate="2009-03-20">
    <creator>Gephi.org</creator>
    <description>A Web network</description>
  </meta>
  <graph defaultedgetype="directed">
    <attributes class="node">
      <attribute id="0" title="url" type="string"/>
      <attribute id="1" title="indegree" type="float"/>
      <attribute id="2" title="frog" type="boolean">
        <default>true</default>
      </attribute>
    </attributes>
    SEE NEXT SLIDES
  </graph>
</gexf>
```

GEXF: Example

```
<nodes>
        <node id="0" label="Gephi">
           <attvalues>
              <attvalue for="0" value="http://gephi.org"/> <attvalue for="1" value="1"/>
           </attvalues>
        </node>
        <node id="1" label="Webatlas">
           <attvalues>
              <attvalue for="0" value="http://webatlas.fr"/> <attvalue for="1" value="2"/>
           </attvalues>
        </node>
        <node id="2" label="RTGI">
           <attvalues>
              <attvalue for="0" value="http://rtgi.fr"/> <attvalue for="1" value="1"/>
           </attvalues>
        </node>
        <node id="3" label="BarabasiLab">
           <attvalues>
              <attvalue for="0" value="http://barabasilab.com"/> <attvalue for="1" value="1"/>
              <attvalue for="2" value="false"/>
           </attvalues>
        </node>
</nodes>
```

GEXF: Example

```
<edges>
        <edge id="0" source="0" target="1"/>
        <edge id="1" source="0" target="2"/>
        <edge id="2" source="1" target="0"/>
        <edge id="3" source="2" target="1"/>
        <edge id="4" source="0" target="3"/>
 </edges>
```

GEXF: Advanced Features

- Dynamics
  - Each node/edge/data can have a lifetime, e.g.,
  - <edge id="1" source="0" target="2" start="2009-03-01" end="2009-03-10"/>

- Hierarchy
  - Allow nested definition of nodes
  - Or specify parent id (can have multiple parent ids forming polygeny)

GraphML

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
    http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
 <key id="d0" for="node" attr.name="color" attr.type="string">
  <default>yellow</default>
 </key>
 <key id="d1" for="edge" attr.name="weight”
                            attr.type="double"/>
 <graph id="G" edgedefault="undirected”>
  SEE NEXT SLIDES
 </graph>
</graphml>
```

GraphML: Example

```
<node id="n0">
    <data key="d0">green</data>
</node>
<node id="n1"/>
<node id="n2">
    <data key="d0">blue</data>
</node>
<node id="n3">
    <data key="d0">red</data>
</node>
<node id="n4"/>
<node id="n5">
    <data key="d0">turquoise</data>
</node>
```

GraphML: Example

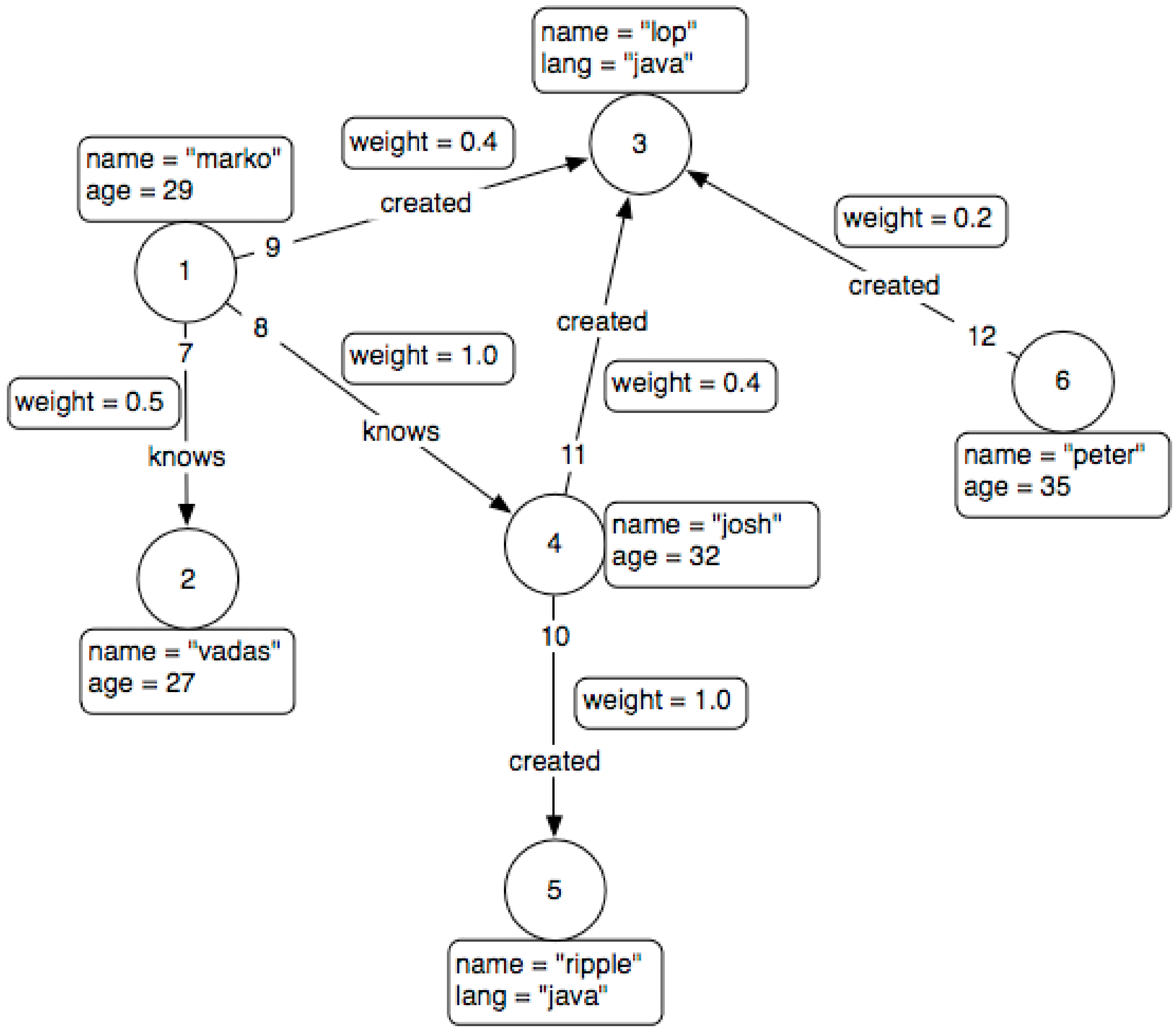
```
<edge id="e0" source="n0" target="n2">
    <data key="d1">1.0</data>
</edge>
<edge id="e1" source="n0" target="n1">
    <data key="d1">1.0</data>
</edge>
<edge id="e2" source="n1" target="n3">
    <data key="d1">2.0</data>
</edge>
<edge id="e3" source="n3" target="n2"/>
<edge id="e4" source="n2" target="n4"/>
<edge id="e5" source="n3" target="n5"/>
<edge id="e6" source="n5" target="n4">
    <data key="d1">1.1</data>
</edge>
```

GraphML: Advanced features

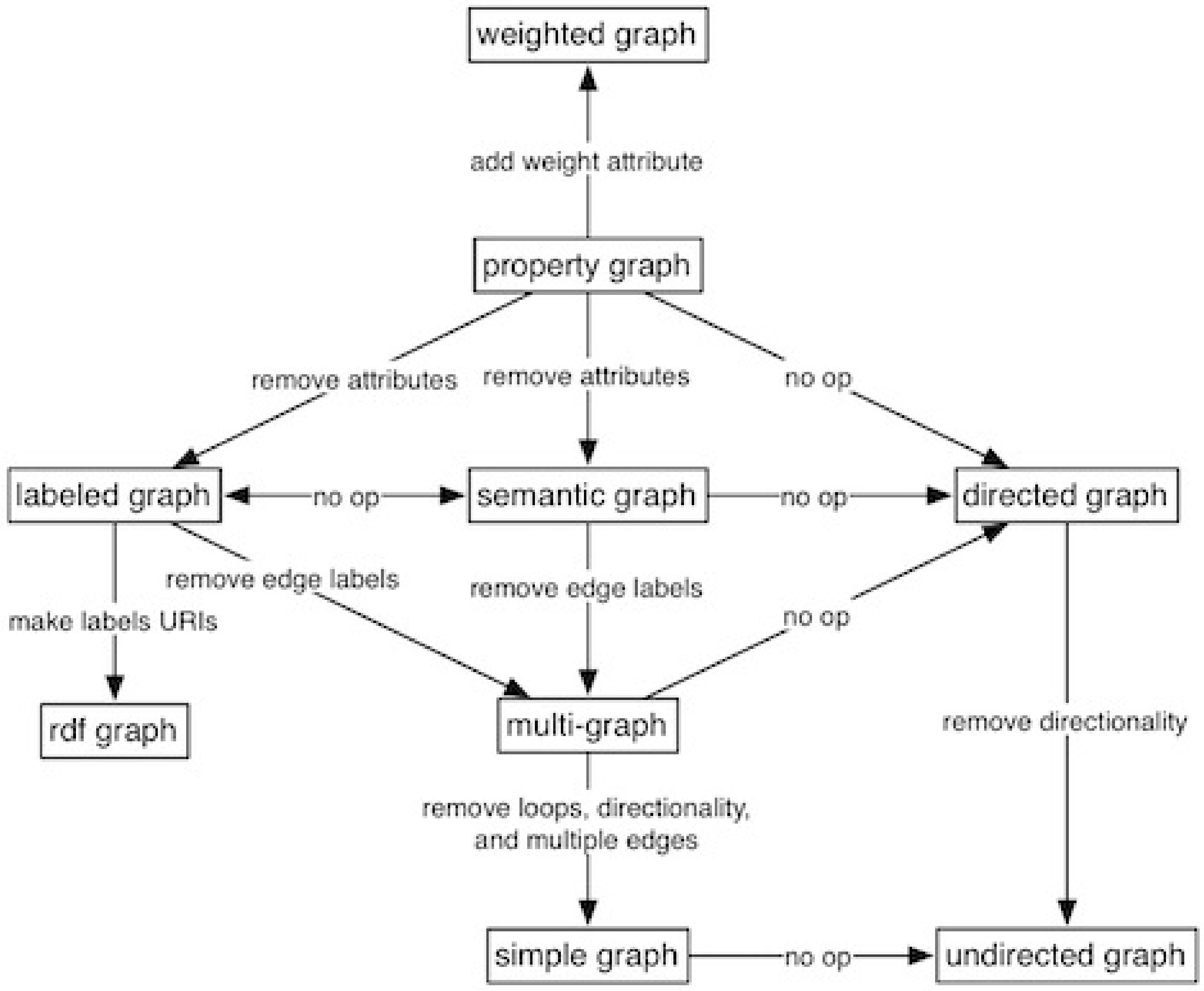
- Nested graphs
- Hyperedges
- Port

Gremlin

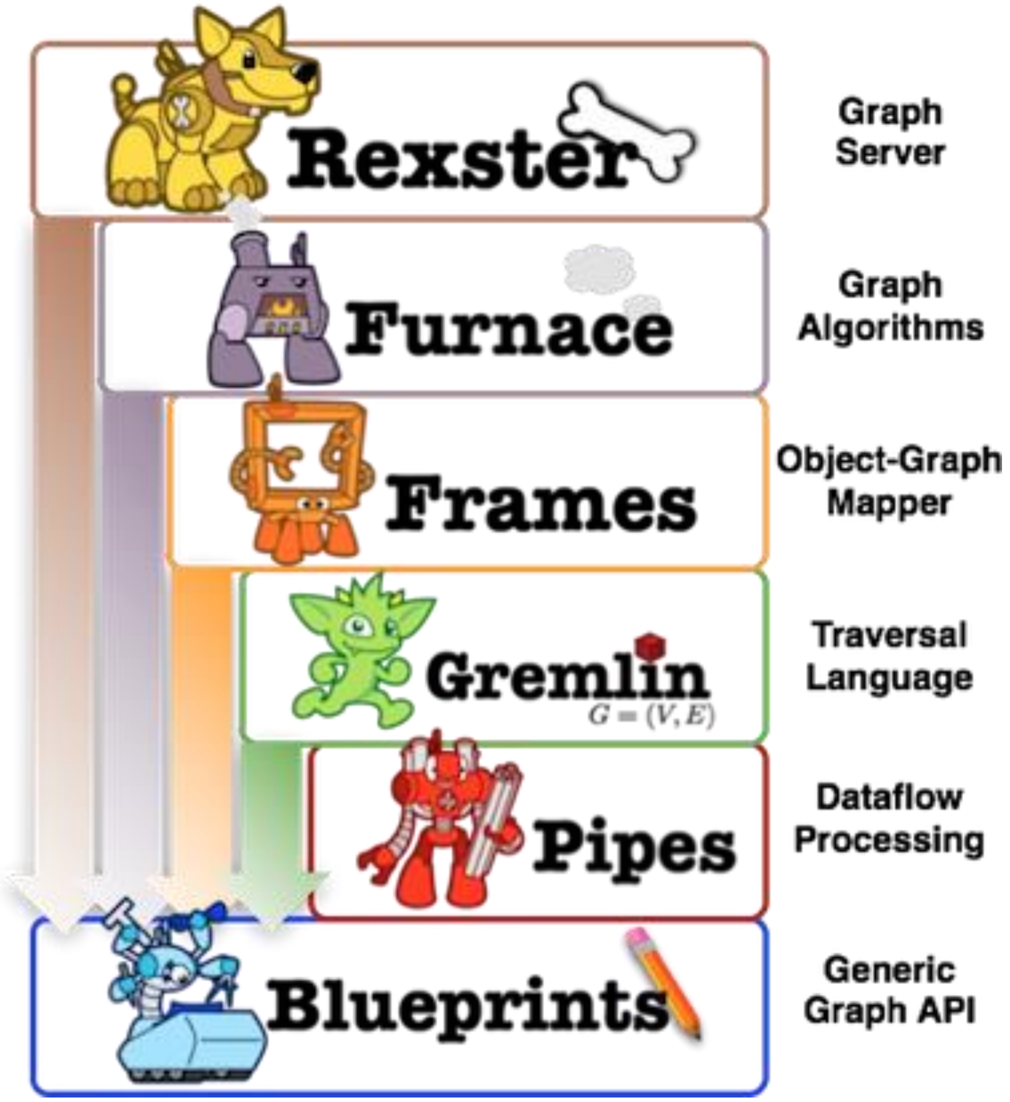
- port



Blueprint


weighted graph
add weight attribute
property graph
remove attributes
remove attributes
no op
labeled graph
no op
semantic graph
no op
directed graph
remove edge labels
make labels URIs
remove edge labels
no op
rdf graph
multi-graph
remove directionality
remove loops, directionality, and multiple edges
simple graph
no op
undirected graph

Gremlin Framework



Related i2rs Efforts

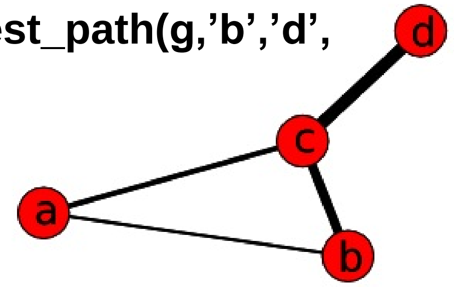
- http://tools.ietf.org/id/draft-medved-i2rs-topology-requirements-00.txt
    - Abstraction, hierarchy, tracable, filtering, push/subscription
- http://tools.ietf.org/id/draft-medved-i2rs-topology-im-01.txt

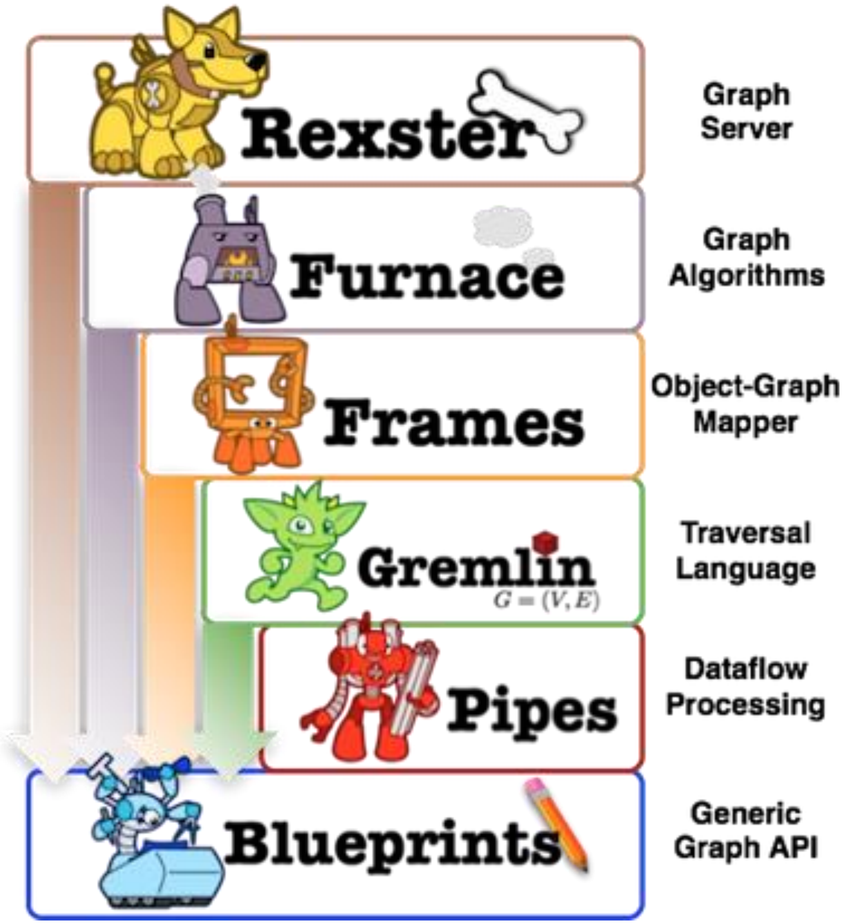
Support for Graph Traversal

- Benefit: a format that would facilitate easy, advanced graph computation by applications, using existing software tools

Example: using NetworkX

```
>>> import networkx as nx
>>> g=nx.Graph()
>>> g.add_edge('a','b',routingcost=1 )
>>> g.add_edge('b','c', routingcost=15)
>>> print(g.adj)
{'a': {'b': {'routingcost':1}}, 'c': {'b': {…}},
'b': {'a': {…}, 'c': {…}}}
>>> g.add_edge('a','c', routingcost=10)
>>> g.add_edge('c','d', routingcost=22)
>>> print(nx.shortest_path(g,'b','d')
['b', 'c', 'd']
>>> print(nx.shortest_path(g,'b','d',
'routingcost')
['b', 'a', 'c', 'd']
```



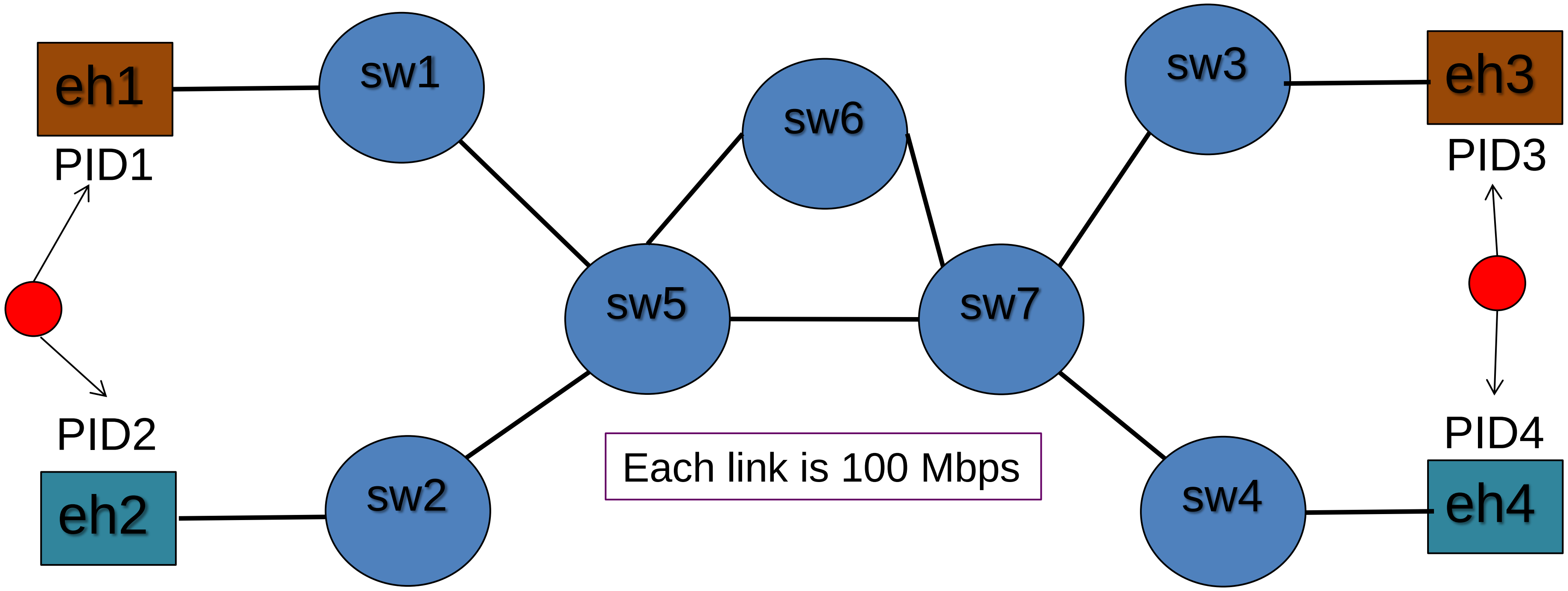


Use Case: Mapping GraphViz

- A format used in graph visualization
- Example

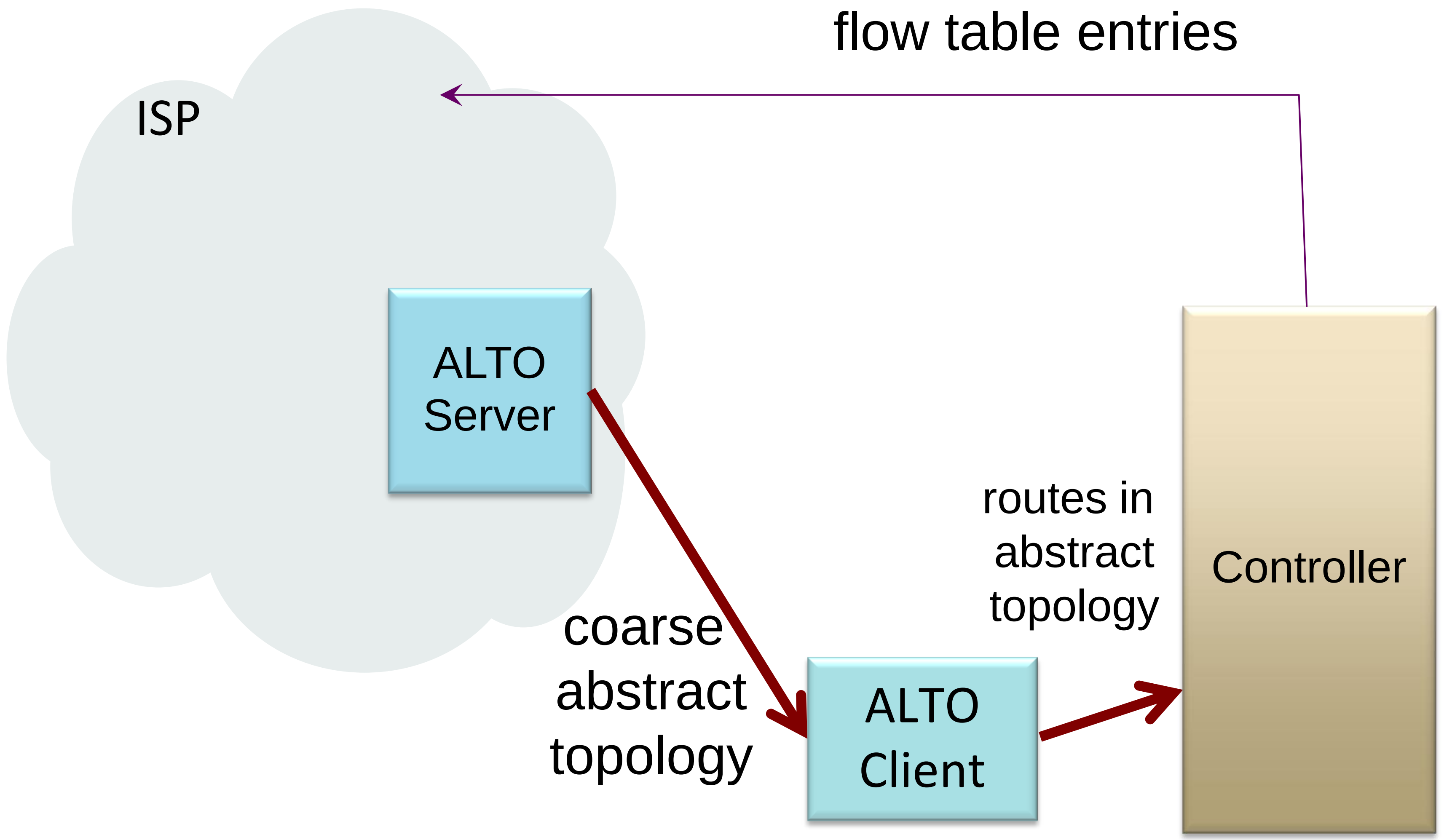
```
graph hello2 {

  // Hello World with nice colors and big fonts

  Node1 [label="Hello, World!", color=Blue, fontcolor=Red,
      fontsize=24, shape=box]

 B [label="The boss"]      // node B
 E [label="The employee"]  // node E

 B->E [label="commands", dir=back, fontcolor=red]
 // revert arrow direction
}
```

Multi-flow Scheduling (App-Path-Selection Use Case)

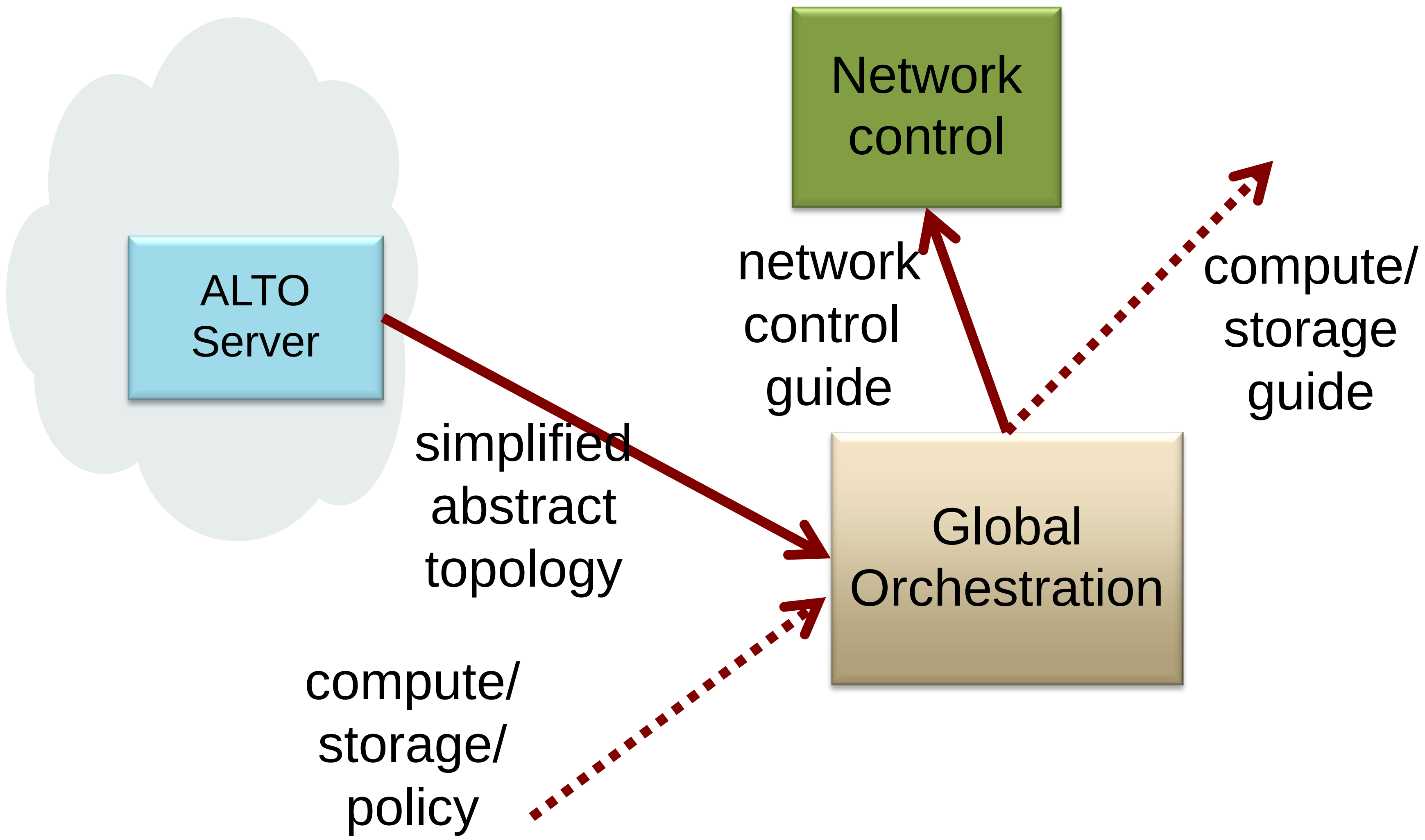


- Application computes maximum flow
  - PID1 -> PID3: sw5 -> sw6 -> sw7;
  - PID2 -> PID4: sw5 -> sw7

App-Path-Selection Work Flow



App-Path-Selection Work Flow


ALTO Server
Network control
network control guide
compute/ storage guide
simplified abstract topology
Global Orchestration
compute/ storage/ policy