

Lessons Learned in

# Multilayer Network Modelling

---

Freek Dijkstra

Universiteit van Amsterdam

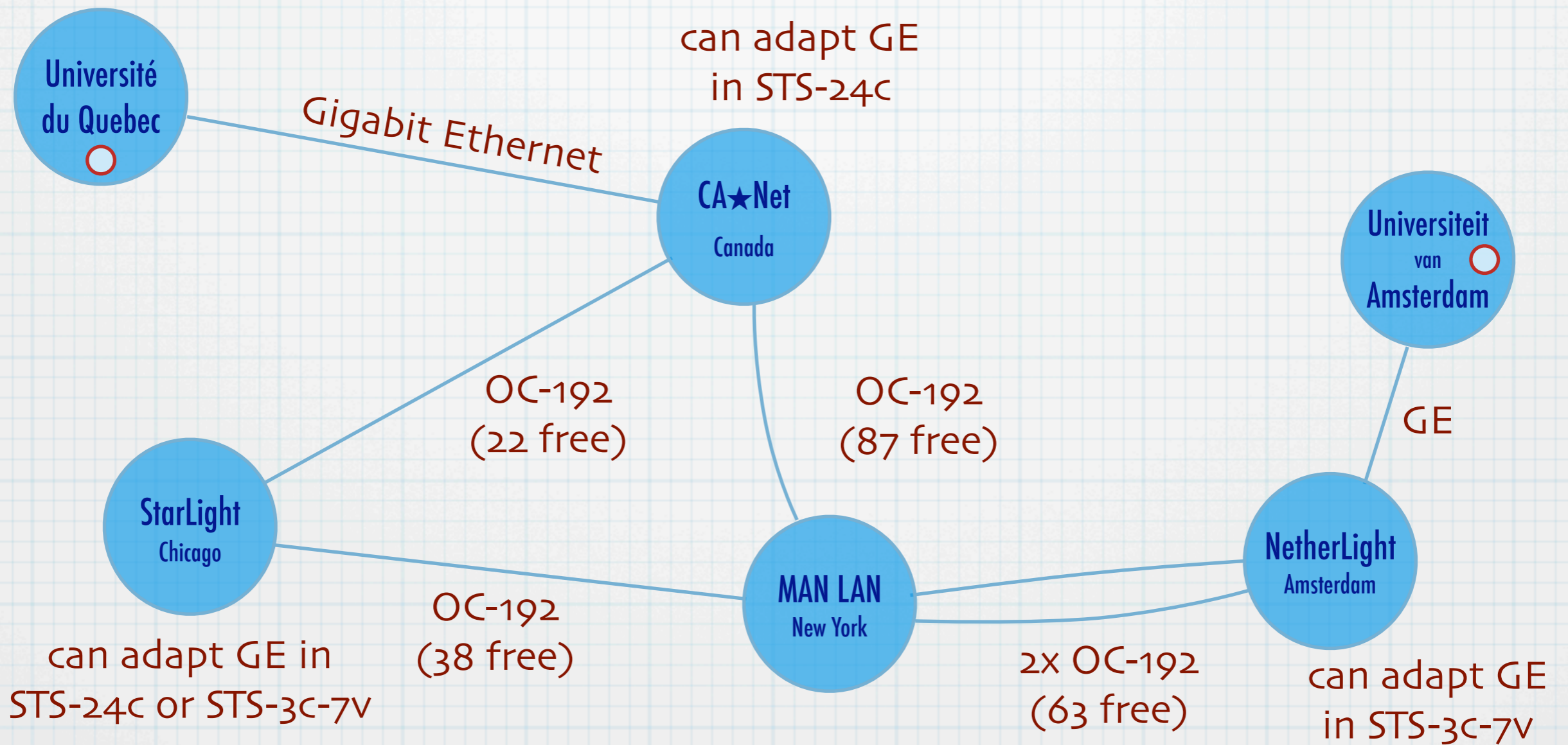
with help of:

Bert Andree, Paola Grosso, Jeroen van der Ham,  
Karst Koymans, Cees de Laat



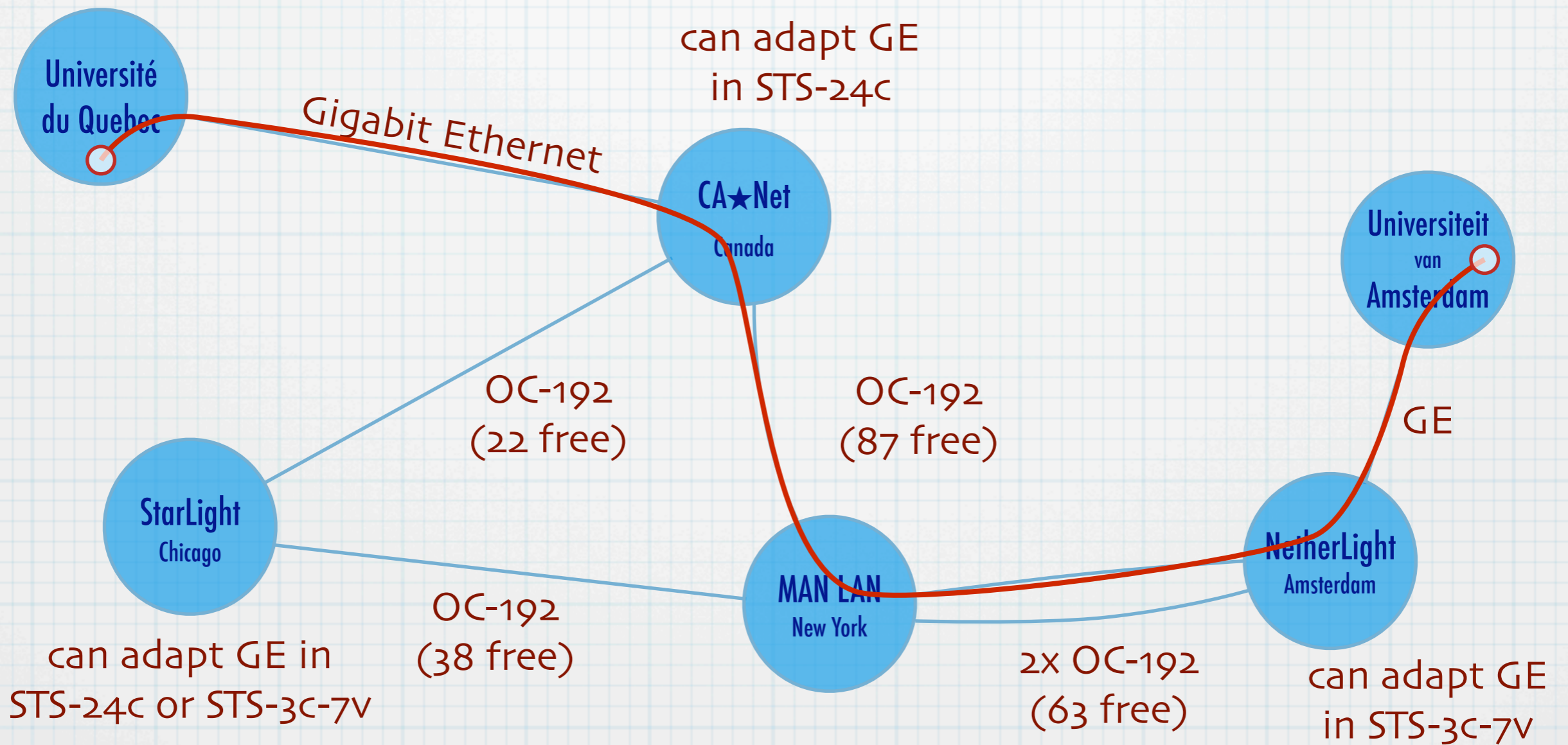
Lessons we learned while developing multi-layer NDL

# Historic Example (modified)



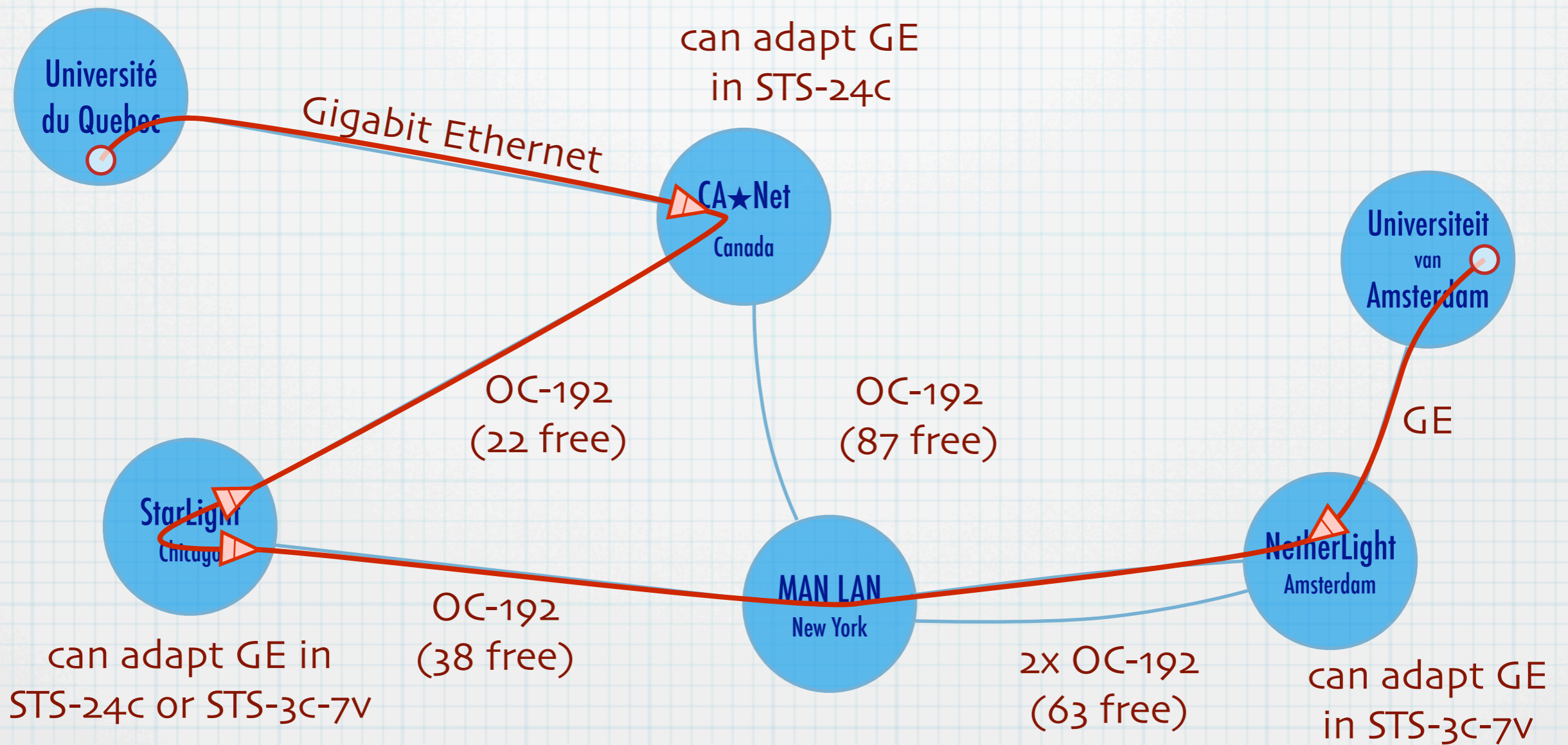
Path finding through multi-layer networks is hard. In fact, I claim that link-constrained algorithms are not sufficient. This is a counter example that proves this claim.

# Historic Example (modified)



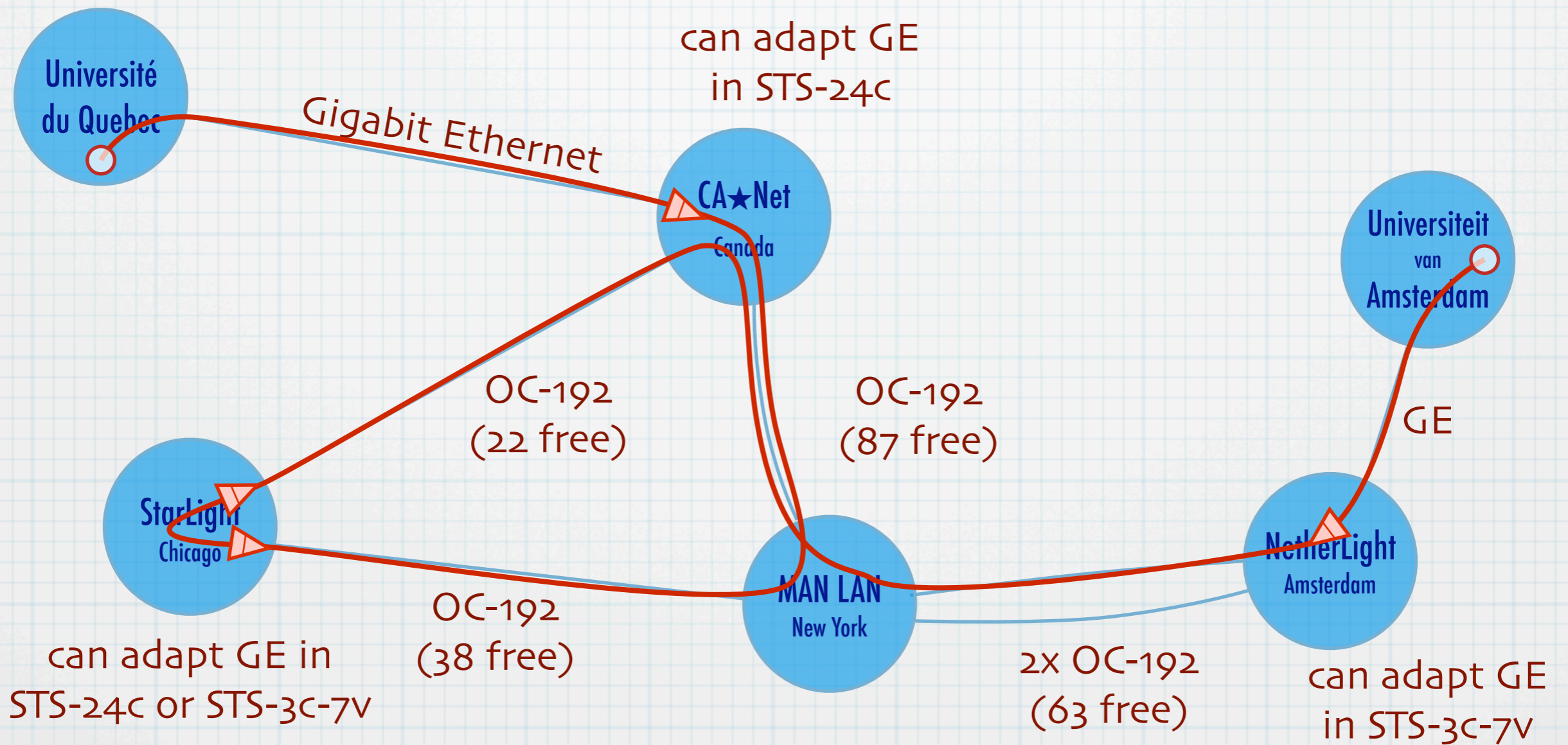
First attempt: invalid path, since adaptation of GE in STS-24c is not compatible with adaptation of GE in STS-3c-7v.

# Historic Example (modified)



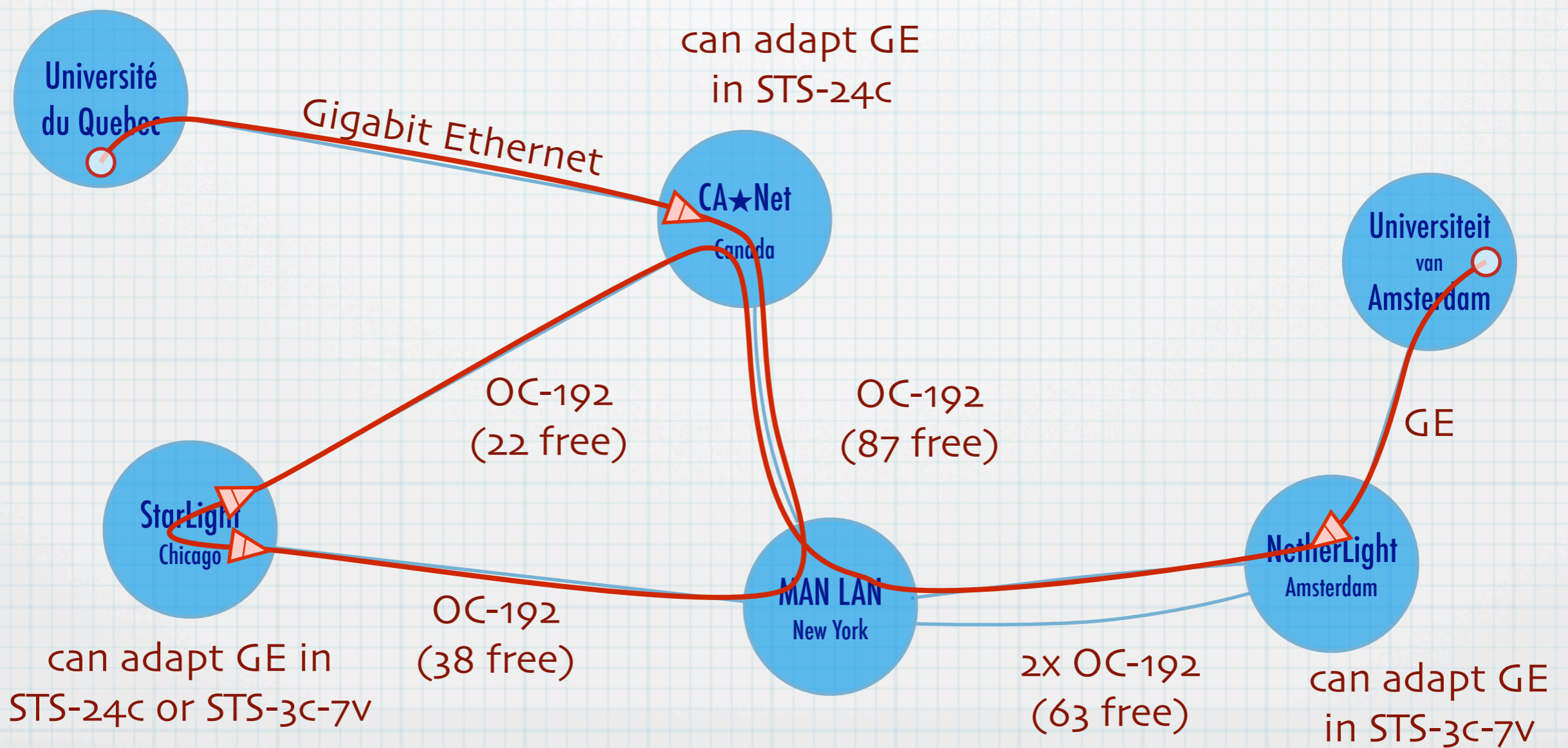
Second attempt: this is an invalid path because there are only 22 free STS channels between CA\*net and StarLight, but 24 are required.

# Historic Example (modified)



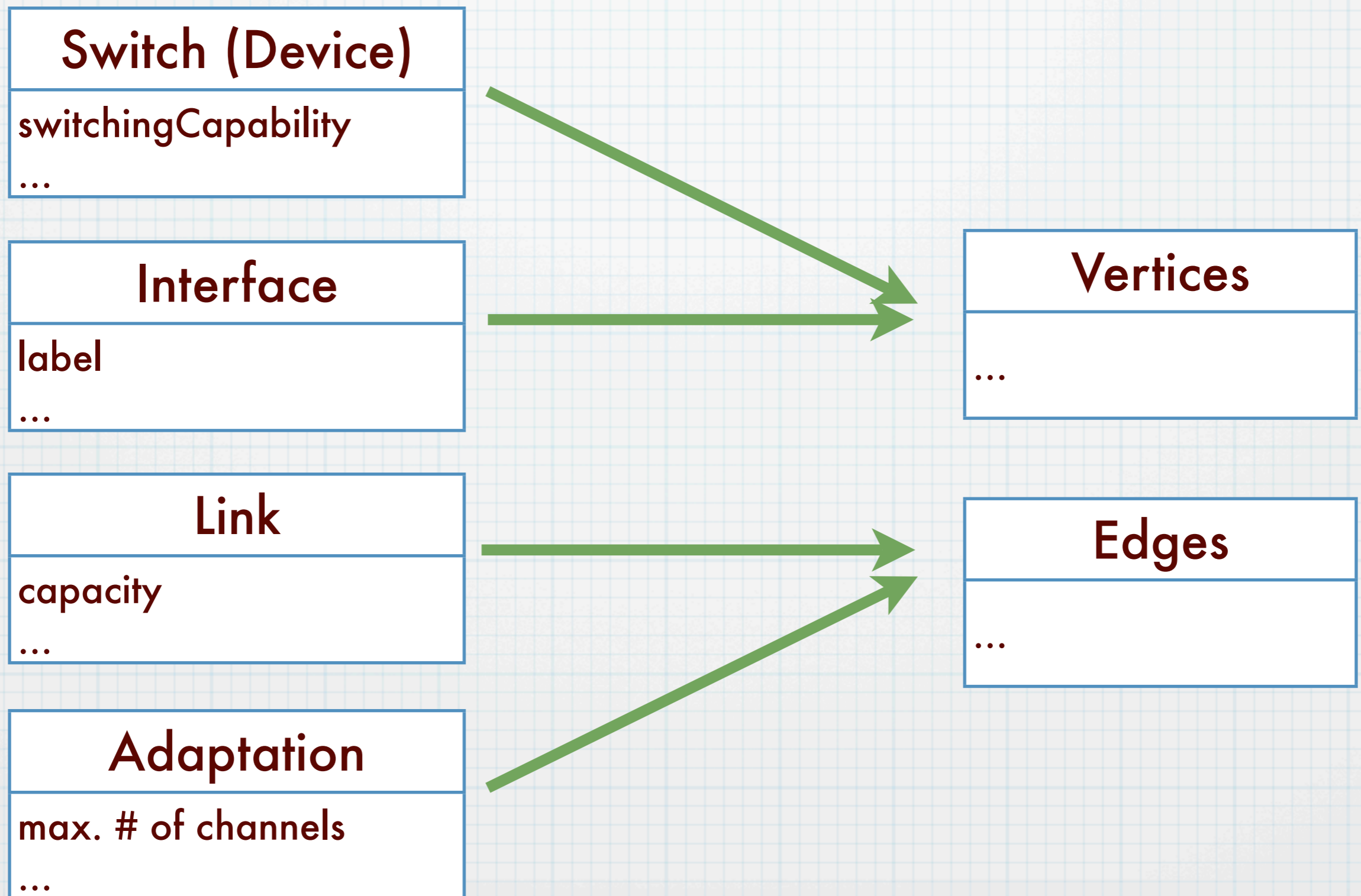
This is the shortest path through this network. You can not just consider one layer in this example: Quebec and Amsterdam do not even know about SDH. MAN LAN does not understand Ethernet. Adaptations are important. We need a new path finding algorithm. "Link-constrained path finding algorithms are not sufficient for multi layer networks if links are 1:1 mapped to edges"

☑ Multi-layer path finding can **not** use **link-constrained** path finding algorithms. **Path-constrained** algorithms are required.



Link-constrained path finding algorithms are not sufficient for multi layer networks if links are 1:1 mapped to edges.

# ☑ Multi-layer networks can not be represented as a simple graph.



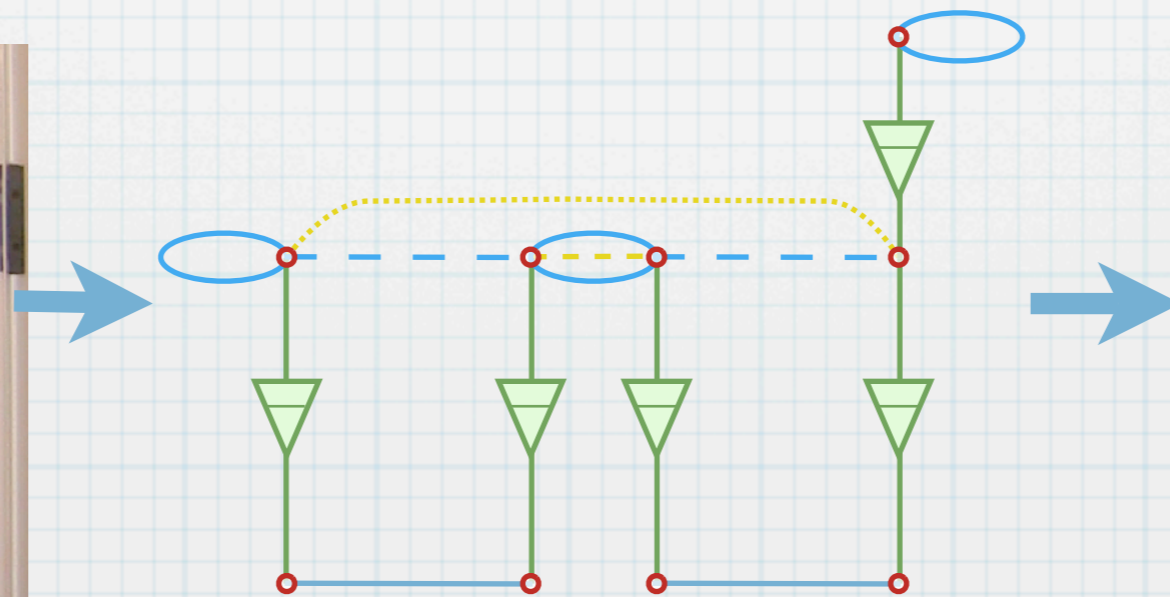
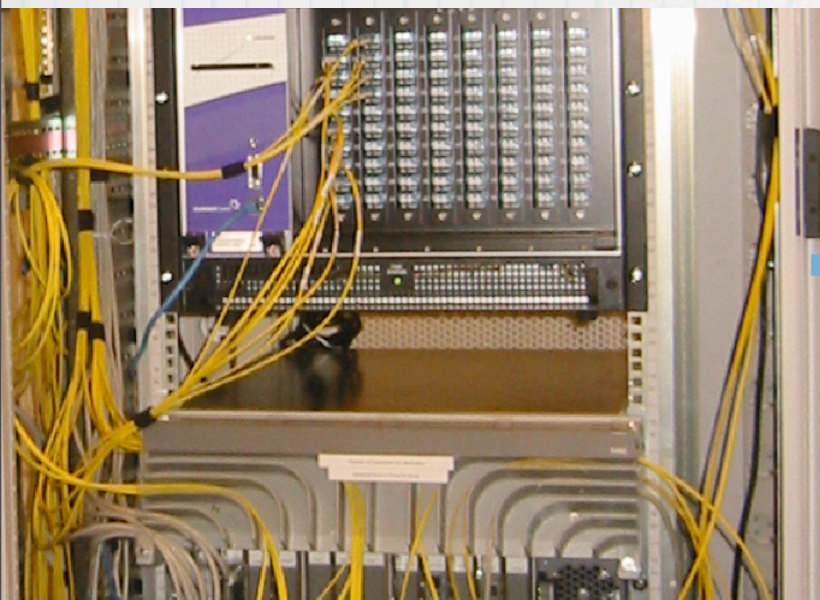
Multi-layer networks have 3 to 5 fundamental “building blocks”. Graphs only have 2 building blocks. Mapping is not possible without loss of information, or without extending graph with more building blocks (e.g. labels attached to edges or vertices)

# The Modelling Process

Network Elements

Functional Elements

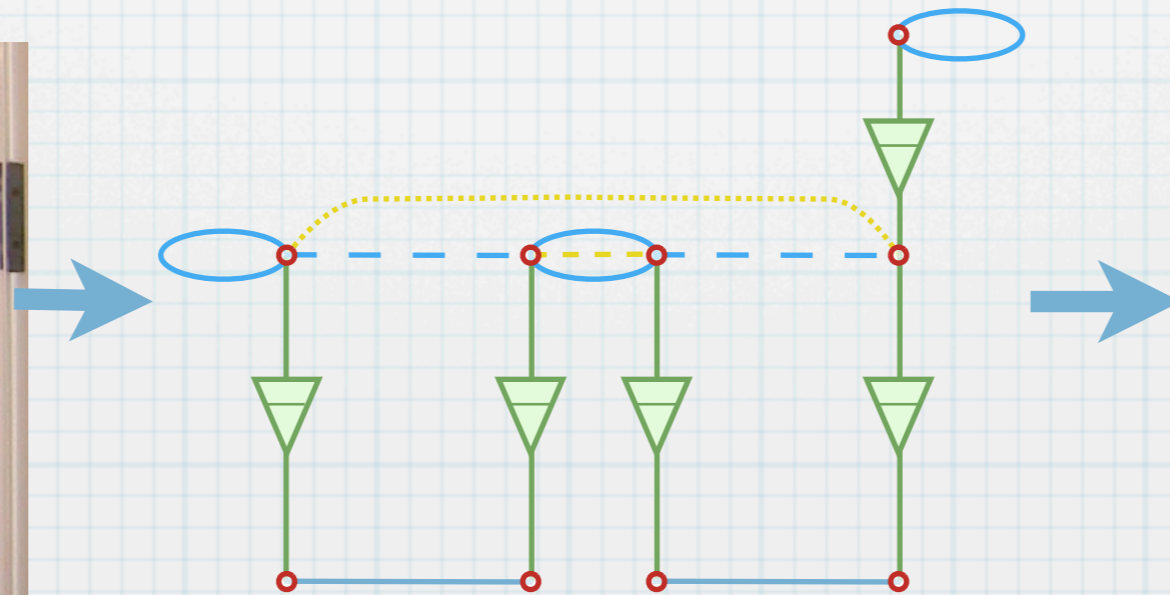
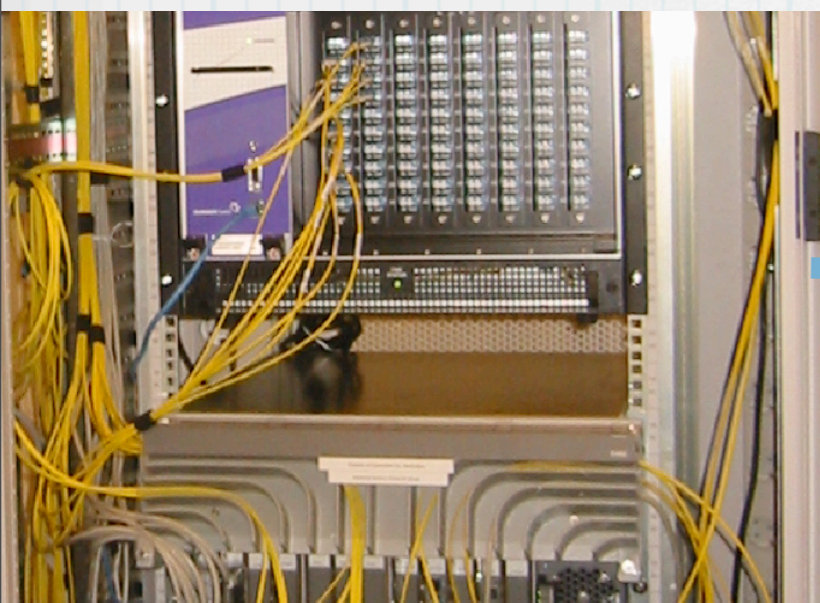
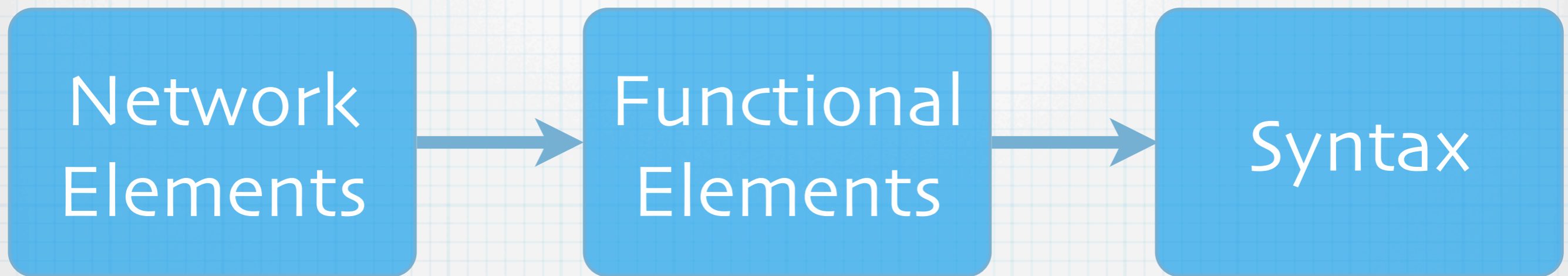
Syntax



```
<ndl:Device rdf:about="#Force10">
  <ndl:hasInterface rdf:resource=
    "#Force10:te6/0"/>
</ndl:Device>
<ndl:Interface rdf:about="#Force10:te6/0">
  <rdfs:label>te6/0</rdfs:label>
  <ndl:capacity>1.25E6</ndl:capacity>
  <ndlconf:multiplex>
    <ndlcap:adaptation rdf:resource=
      "#Tagged-Ethernet-in-Ethernet"/>
    <ndlconf:serverPropertyValue
      rdf:resource="#MTU-1500byte"/>
  </ndlconf:multiplex>
  <ndlconf:hasChannel>
    <ndlconf:Channel rdf:about=
      "#Force10:te6/0:vlan4">
      <ndleth:hasVlan>4</ndleth:hasVlan>
      <ndlconf:switchedTo rdf:resource=
        "#Force10:gi5/1:vlan7"/>
    </ndlconf:Channel>
  </ndlconf:hasChannel>
</ndl:Interface>
```

- Network elements: physical devices
- Map to a model (G.805 functional elements, graphs, etc.)
- Describe model in a concise, but compact syntax (e.g. NDL, OSPF-TE LSA's, etc.)

- ☑ It is easy to translate between two different syntaxes with the same model.
- ☑ It is hard to convert between two models.



```
<ndl:Device rdf:about="#Force10">
  <ndl:hasInterface rdf:resource=
    "#Force10:te6/0"/>
</ndl:Device>
<ndl:Interface rdf:about="#Force10:te6/0">
  <rdfs:label>te6/0</rdfs:label>
  <ndl:capacity>1.25E6</ndl:capacity>
  <ndlconf:multiplex>
    <ndlcap:adaptation rdf:resource=
      "#Tagged-Ethernet-in-Ethernet"/>
    <ndlconf:serverPropertyValue
      rdf:resource="#MTU-1500byte"/>
  </ndlconf:multiplex>
  <ndlconf:hasChannel>
    <ndlconf:Channel rdf:about=
      "#Force10:te6/0:vlan4">
      <ndleth:hasVlan>4</ndleth:hasVlan>
      <ndlconf:switchedTo rdf:resource=
        "#Force10:gi5/1:vlan7"/>
    </ndlconf:Channel>
  </ndlconf:hasChannel>
</ndl:Interface>
```

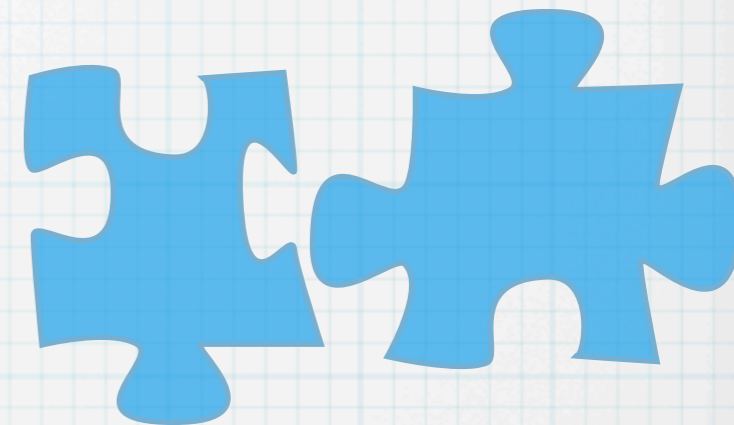
Having a common model very much helps interoperability between different control planes.

- ☑ There is a loose relation between a device and capabilities (e.g. switching capability). A domain also has capabilities.
- ☑ A network is a collection of network elements; A domain is an institute (collection of people) enforcing the policy of a set of resources (network- and other resources)

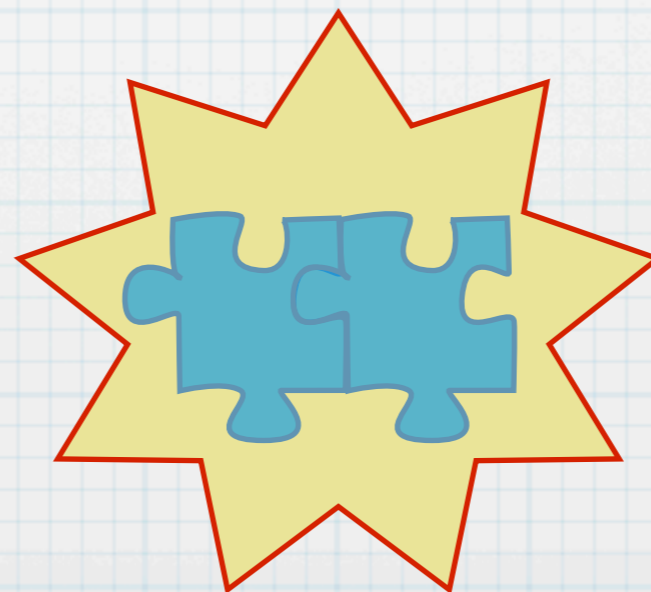
☑ Technologies evolve over time



☑ Multiple technologies causes incompatibilities



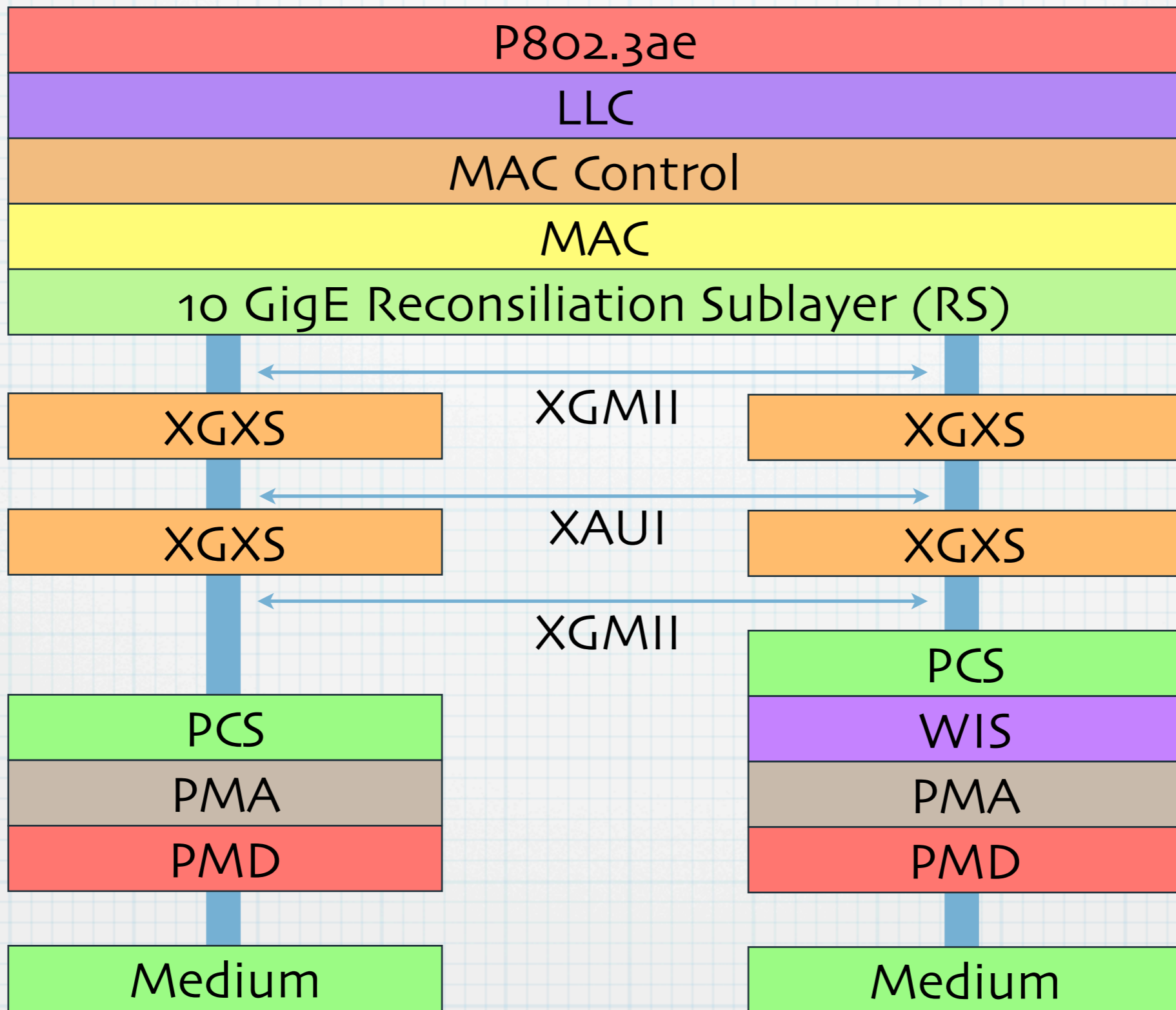
☑ There will always be incompatibilities (until one standard technology is chosen, if ever)



Incompatibilities will always exist, and thus always must be described.

# Path setup is considered a multi-stage process:

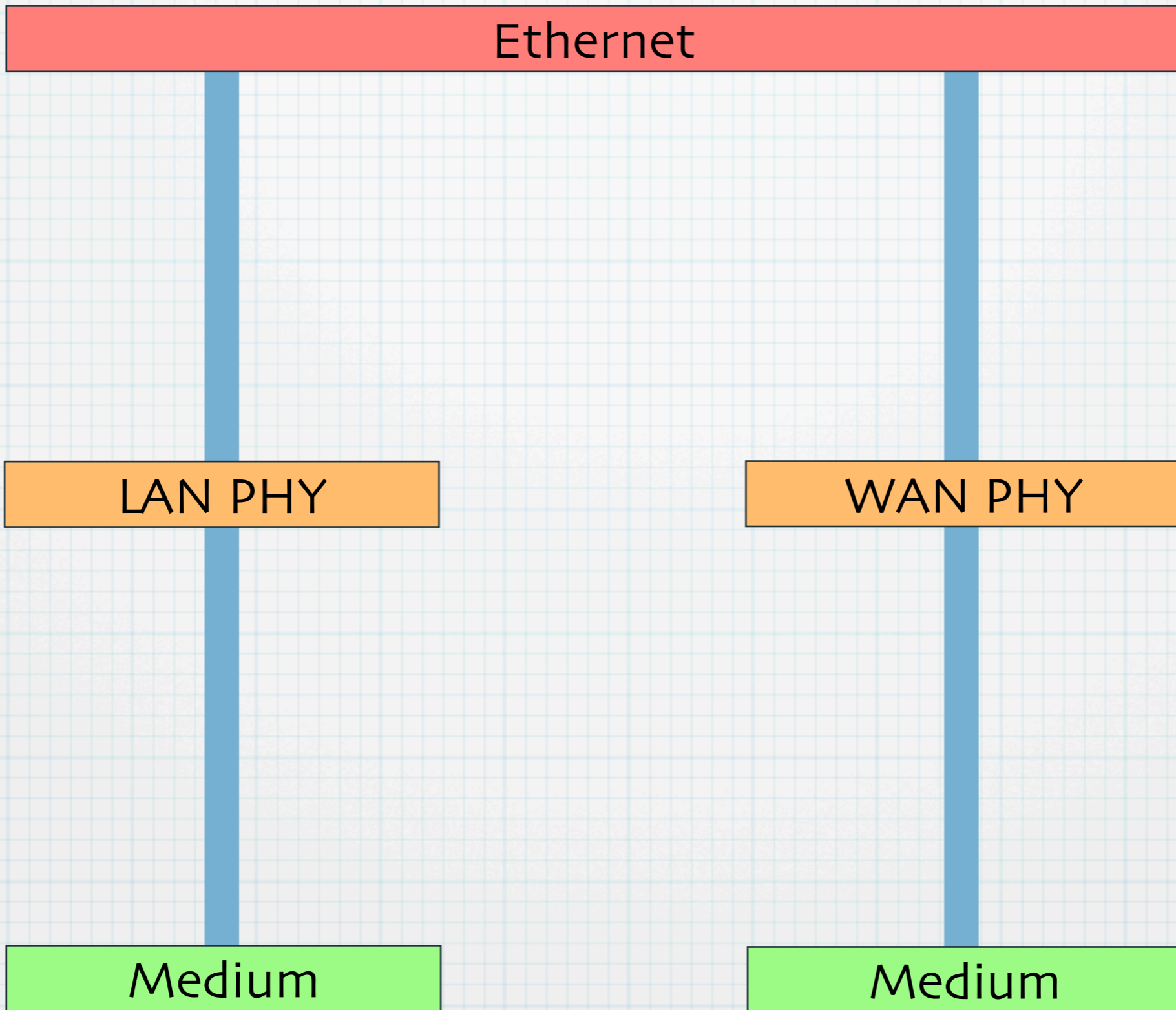
- Routing (*distribution of resource information and state to neighbours*)
- Path computation
- Signalling (*optional choice of parameters*)
- Provisioning (*configuration of resources*)
- ☑ A perfect path finding algorithm needs **all** available information
- ☑ There is a trade-off between a perfect algorithm and information abstraction.



Data link layer only

Reason to separate technology schemas from other schemas: we want to allow changes to the schema.

It is not clear how much simplification we allow in the schema. Here is an example of a model with lots of details. Left: LAN PHY, Right: WAN PHY Ethernet



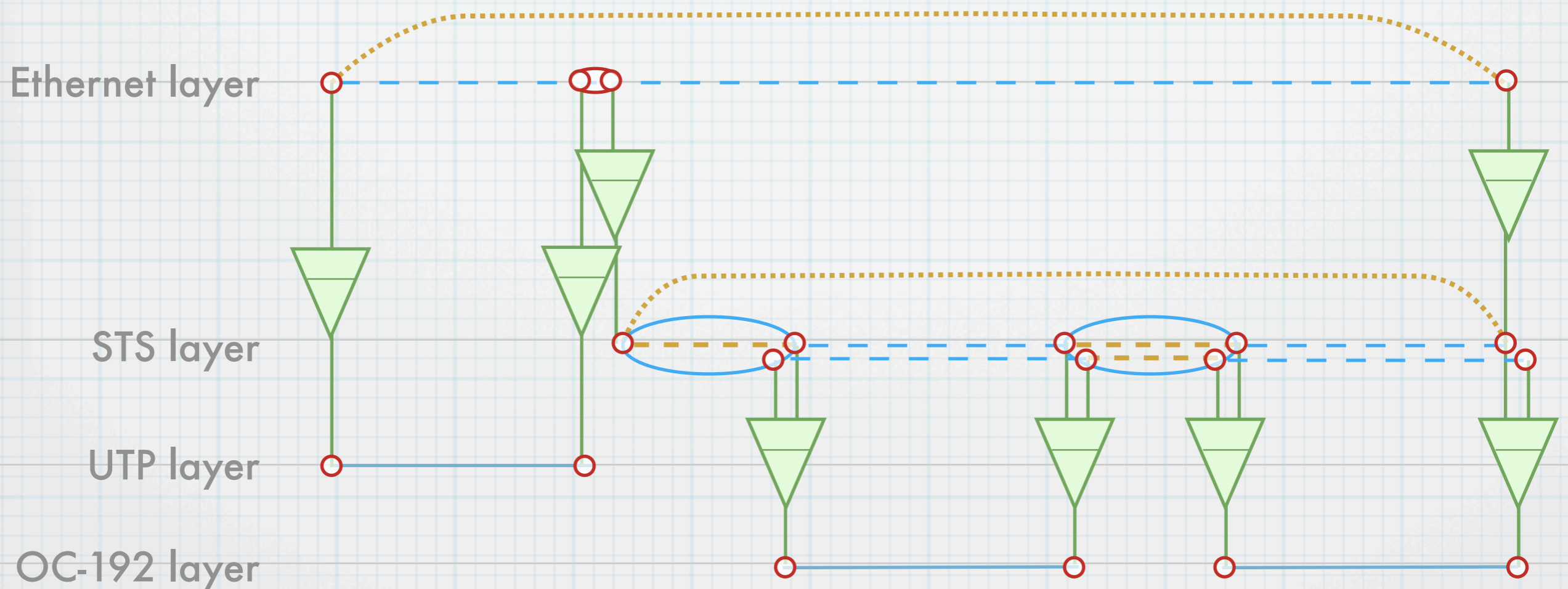
Data link layer only

More simple description of same layers.  
Reason to for me describe networks: describe incompatibilities for path finding, so I don't need more info. Perhaps someone else may.

- ☑ It is only relevant to describe incompatibilities, not other details
- ☑ Technologies evolve over time
- ☑ Multiple technologies causes incompatibilities
- ☑ Incompatibilities changes over time
- ☑ What to describe changes over time.

For example, when there is only one wavelength in fiber, there is no need to describe it. With WDM it is necessary. If all lasers are tunable and all device can do wavelength conversions, there are no compatibilities anymore, and it does not need to be described anymore.

- ☑ G.805 and G.800 allow descriptions of the **state** of a network
- ☑ No model exist to describe how to **change that state**, and who may do so



There are no models to describe state changes (= capabilities).  
GMPLS can describe capabilities, but does not have a formal model.

## At least four possible constraints:

- Topology (*is there a link?*)
  - Technology (*is it compatible?*)
  - Scheduling (*is it available?*)
  - Policy (*may I use it?*)
- state**
- state changes**

- ☑ A path finding algorithm should take all above constraints into account

So far we looked to topology and technology. Not to scheduling and policy.  
Technology covers both state and state changes (capabilities)

## At least four possible constraints:

- Topology (*is there a link?*)
  - Technology (*is it compatible?*)
  - Scheduling (*is it available?*)
  - Policy (*may I use it?*)
- state**
- state changes**

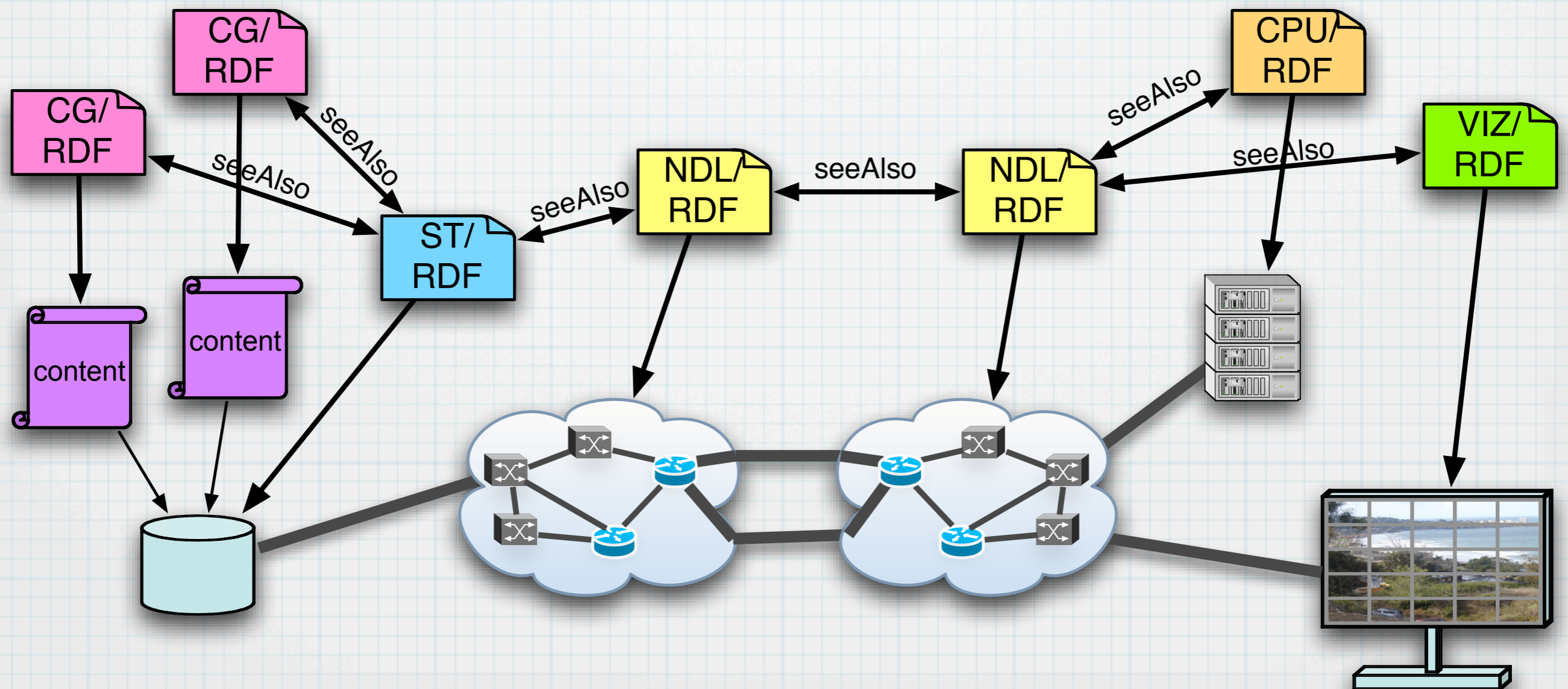
A good algorithm first eliminates most impossible alternatives

This depends on future use and choices in the technology and policy

If there are lots of incompatibilities, it makes sense to first look at technology constraints. If policies are strict, those should first be examined. If the usage is very high, that is the primary constraint.

# A metascheduler may take into account:

- Network resources
- Computing resources
- Storage resources



RDF allows the coupling between different information sources.

# A metascheduler may take into account:

- Network resources
- Computing resources
- Storage resources
- ☑ Regardless of the chosen algorithm, this information needs to be distributed
- ☑ Different information sources should be linked (e.g. host and network)
- ☑ A syntax is required that can refer to other information types and sources

RDF allows the coupling between different information sources.

Questions