# A Declarative and Expressive Approach to Control Forwarding in Carrier-Grade Networks

Stefano Vissicchio

UCLouvain

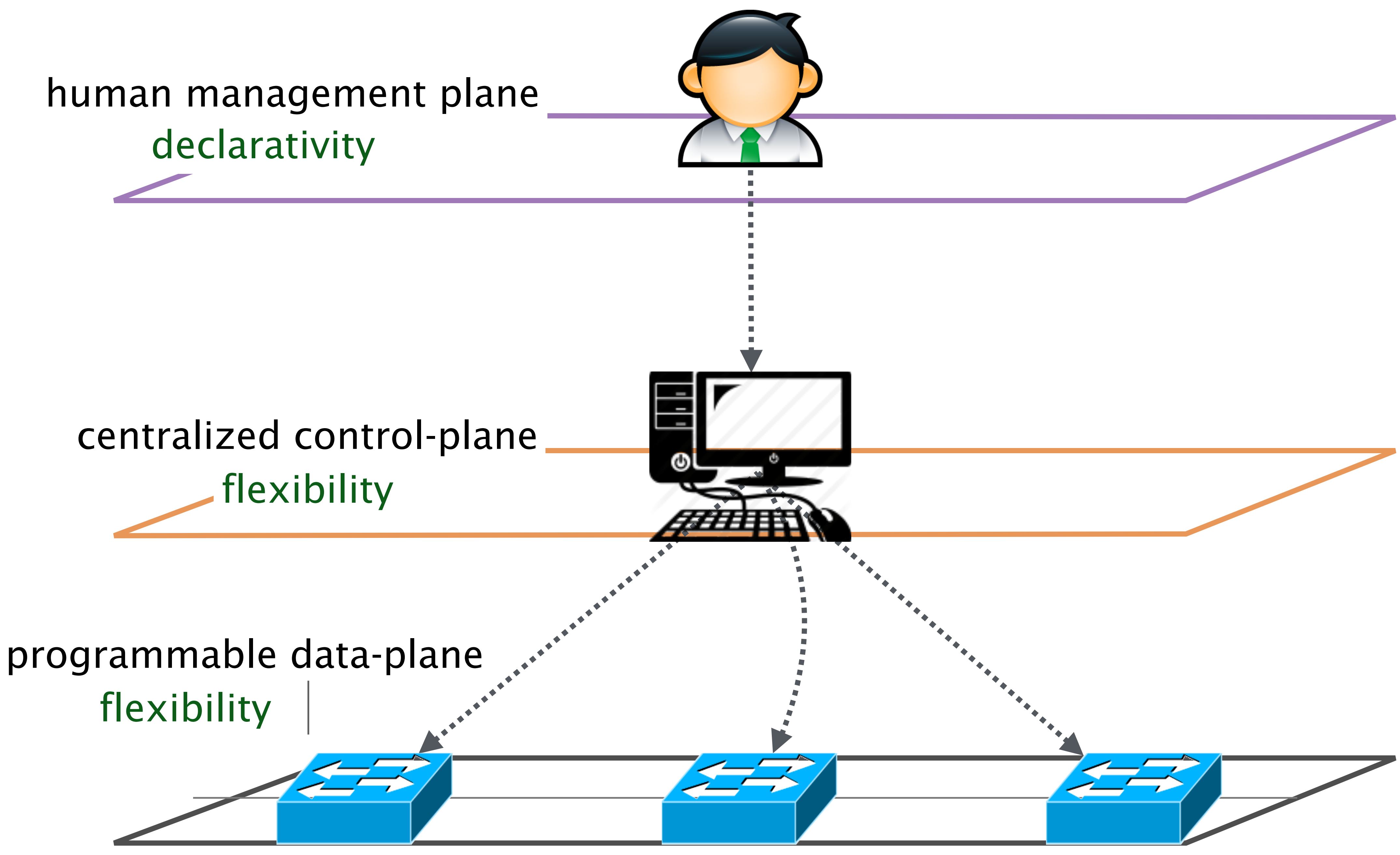
SIGCOMM

18th August 2015

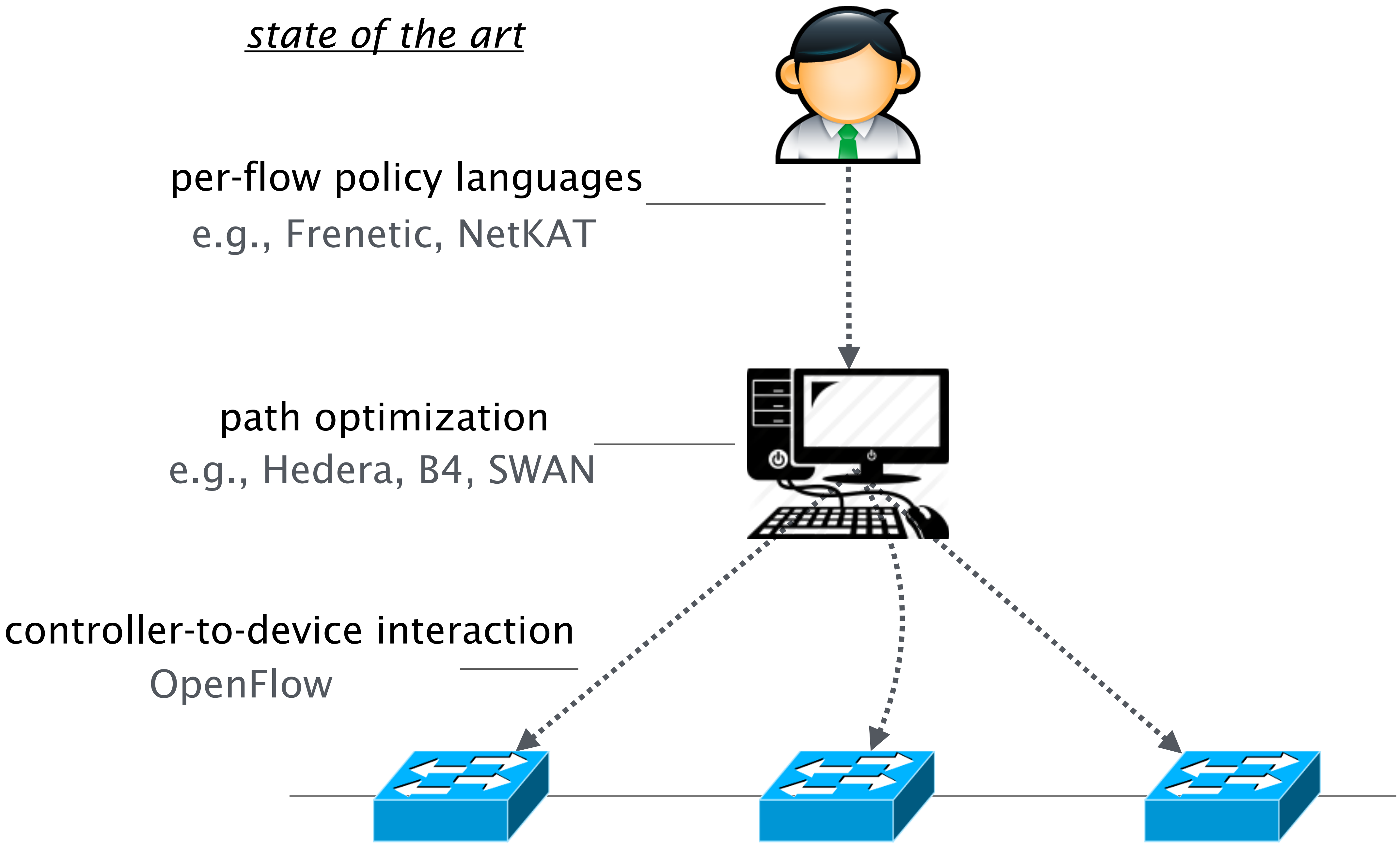
Joint work with

R. Hartert, P. Schaus, O. Bonaventure (UCLouvain),
C. Filsfils, T. Thelkamp (Cisco) and P. Francois (IMDEA)

# Two key features for SDN success are declarativity and flexibility



# SDN has been proven *advantageous* in several settings, from data centers to WANs



We study how to implement SDN
in carrier-grade networks

We study how to implement SDN

in carrier-grade networks

extreme scalability

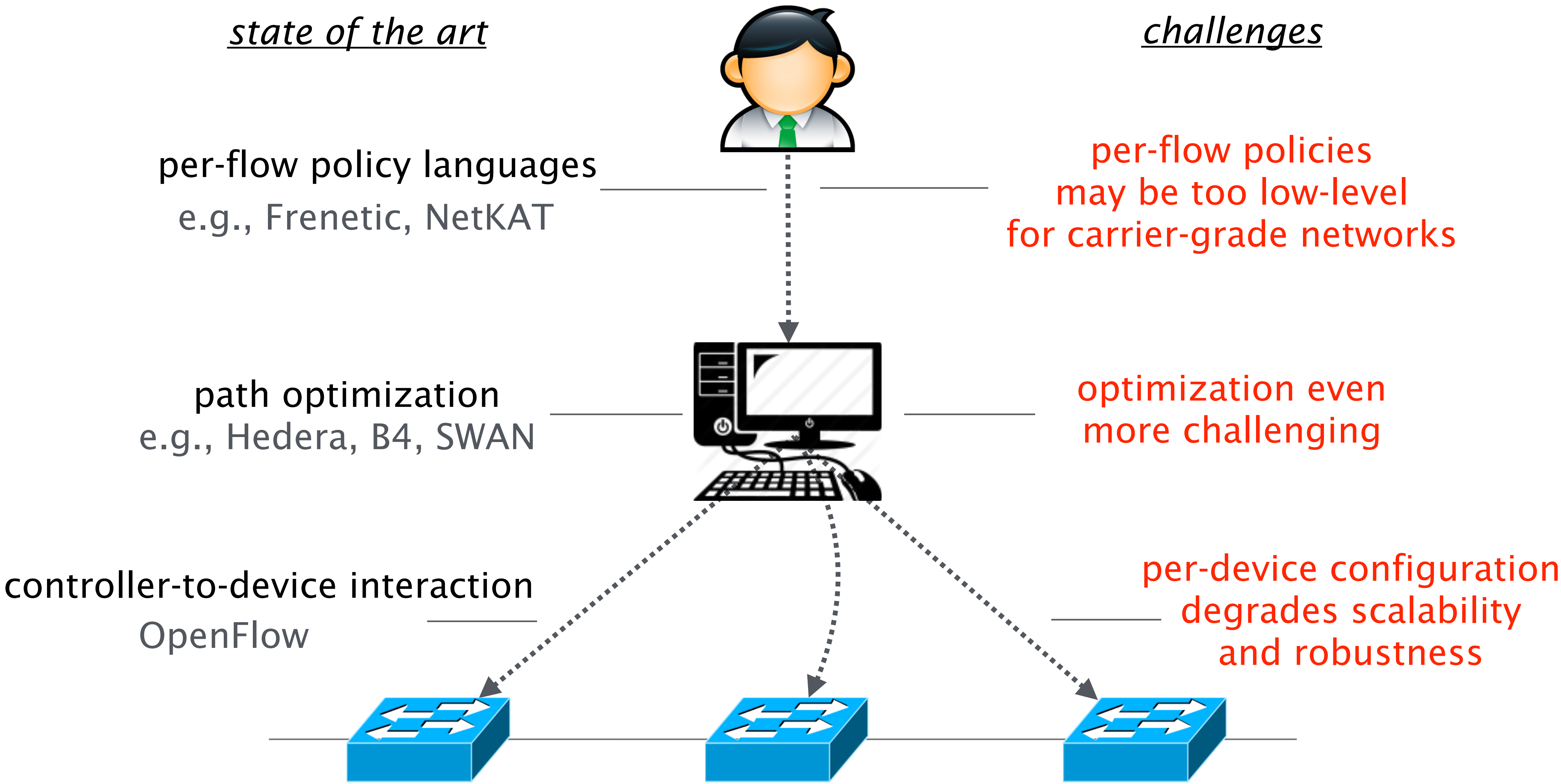
- order of million destinations
- hundreds of geographically-distributed devices

We study how to implement SDN
in carrier-grade networks

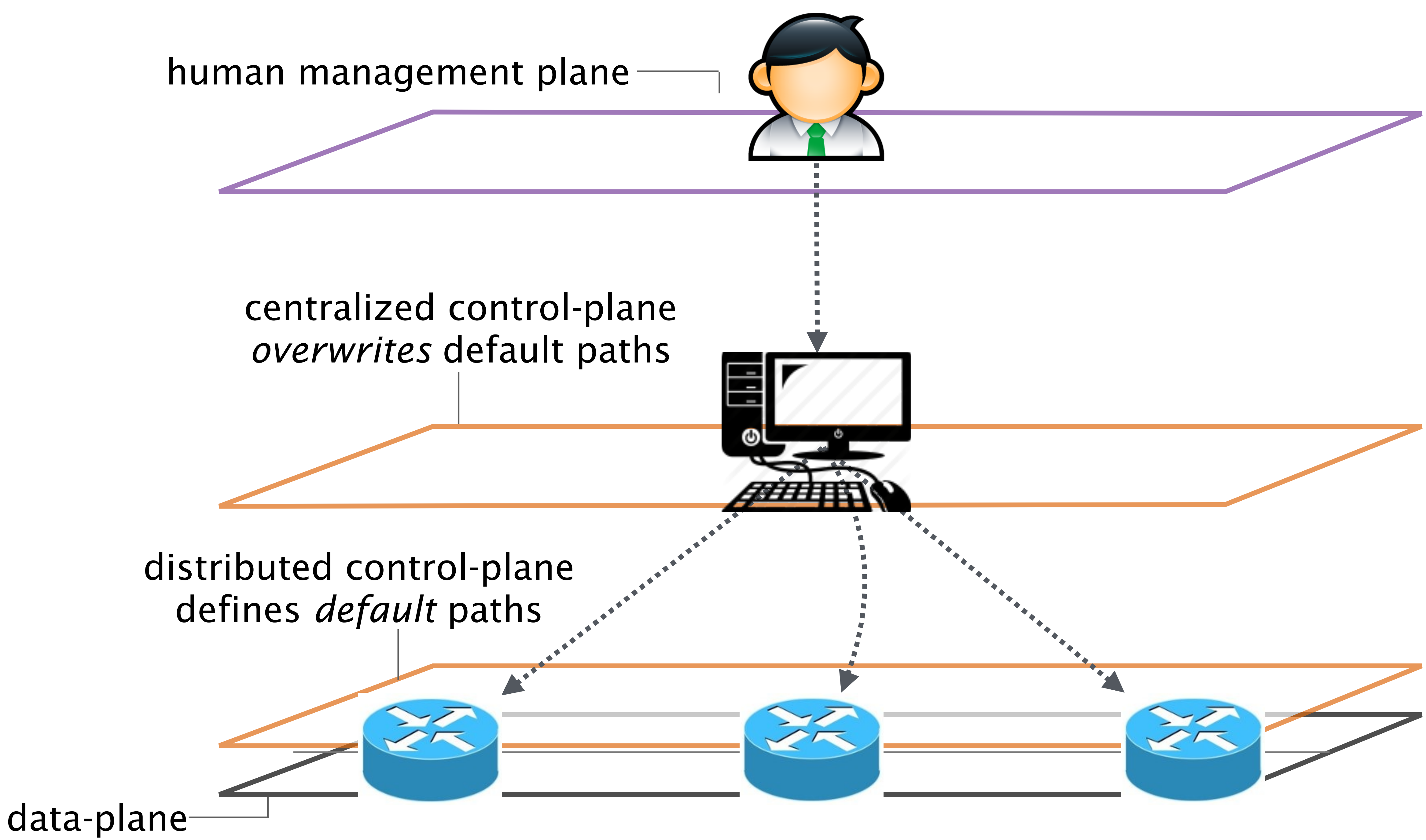
strict robustness requirements

- fast failure recovery to comply with SLAs

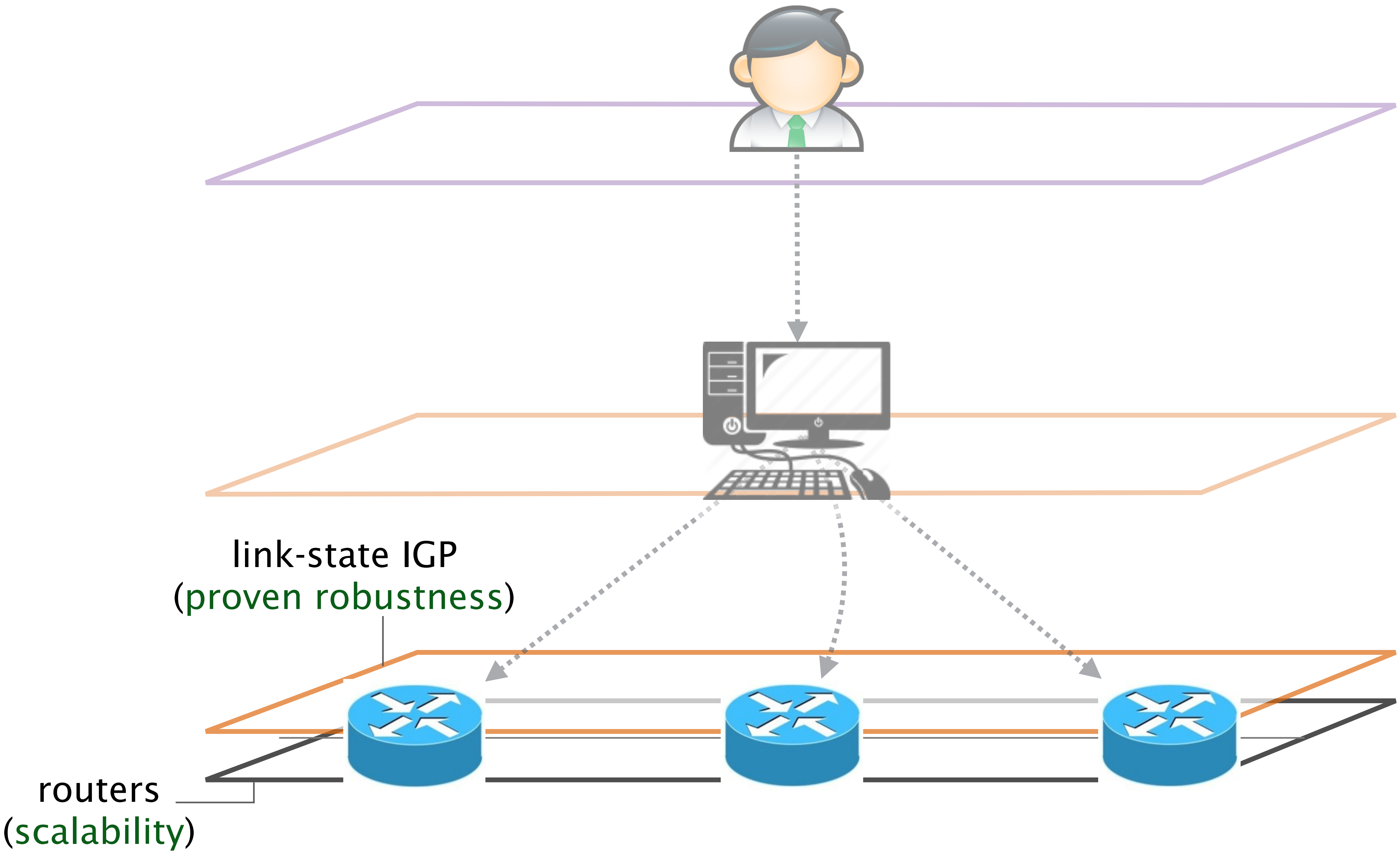
# Extreme robustness and scalability comes with new *challenges* for SDN



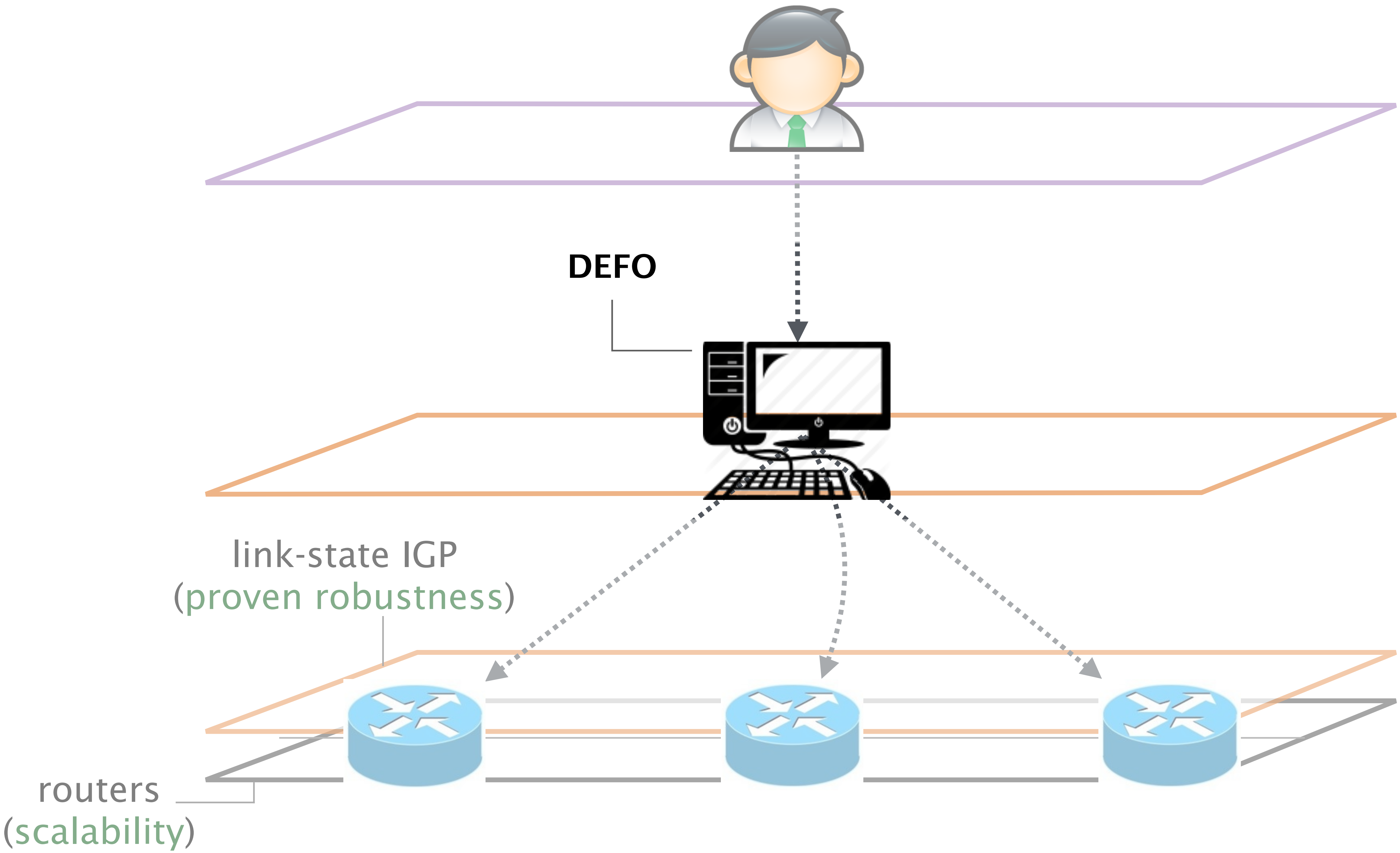
# We study a network architecture including two control-planes



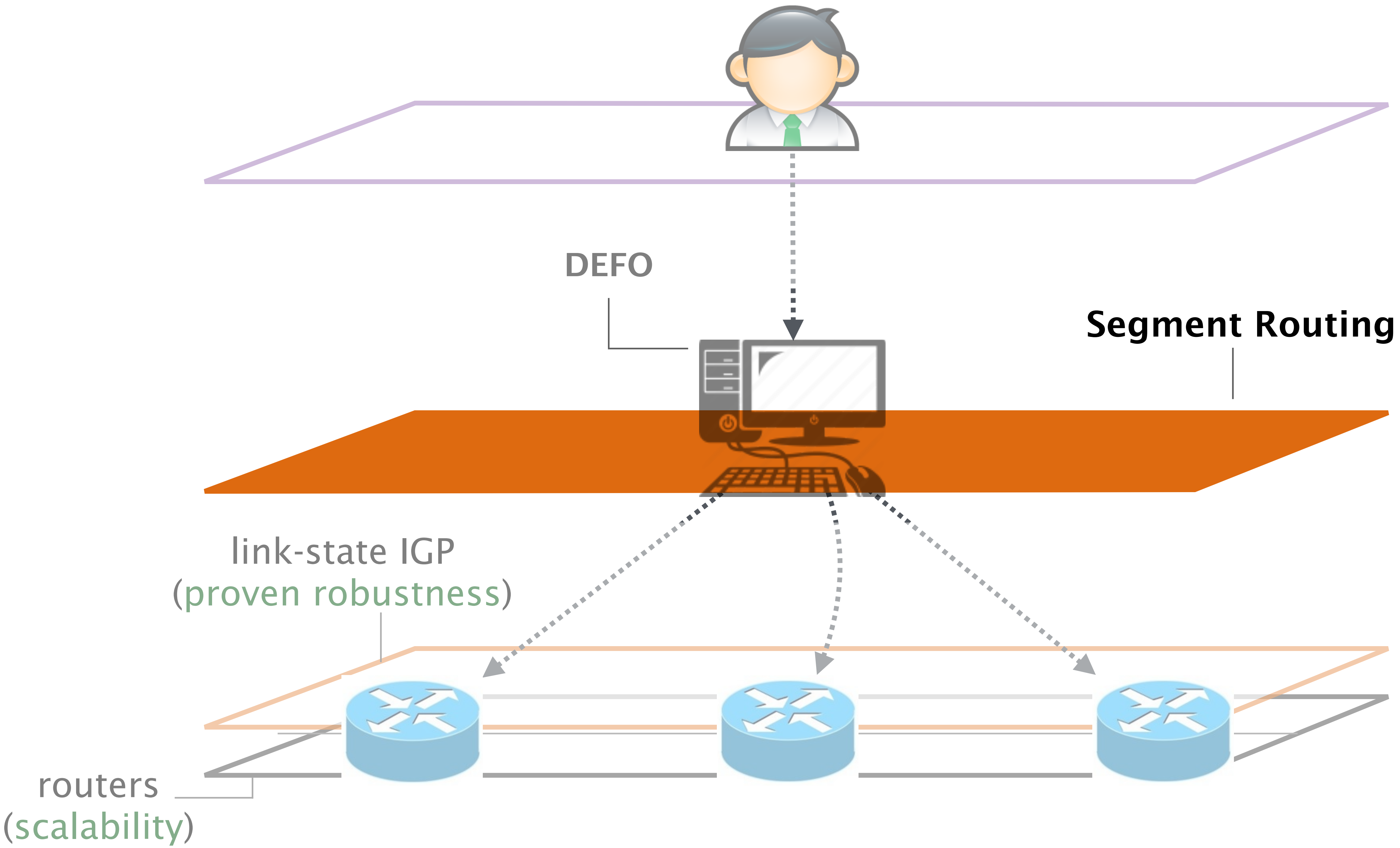
# We use the distributed control-plane to ensure network-wide connectivity



We design and implement DEFO,
that translates high-level *goals* into *optimized paths*



# We evaluate Segment Routing and commercial alternatives to realize optimized paths on routers



DEFO interface is based on goals
expressing desired forwarding at high-level

DEFO interface is based on goals expressing desired forwarding at high-level

flow aggregates called **demands**

constraints and objectives on demands,
expressed by **forwarding functions**

DEFO interface is based on goals
expressing desired forwarding at high-level

# Forwarding functions map demands to parameters associated to its forwarding paths

| forwarding function | DEFO DSL constructs |
|---|---|
| max link load | demand.load |
| max path delay | demand.latency |
| optimization overhead | demand.deviations |
| node traversal | demand passThrough {sw1,sw2} |
| sequencing | demand passThrough {sw1,sw2} then {fw} |
| node avoidance | demand avoid {sw1,sw2} |

# DEFO interface can intuitively express *classic* traffic engineering goals

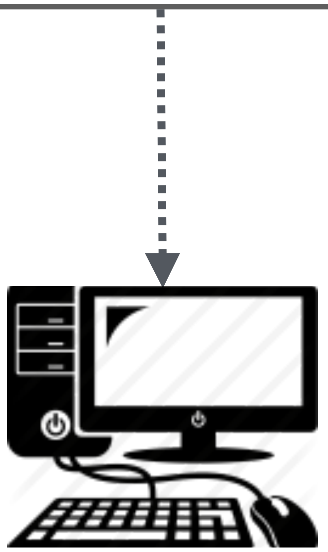
```
var maxLoad = max(load,topology.links)
val goal = new Goal(topology){
      minimize(maxLoad)}
```

# DEFO interface can intuitively express *refined* traffic engineering goals

```
var maxLoad = max(load,topology.links)
val goal = new Goal(topology){
      for(d <- LowDelayDemands)
            add(d.latency <= 10.ms)
      minimize(maxLoad)}
```

# DEFO interface can intuitively express *service chaining* constraints

```
var maxLoad = max(load,topology.links)
val goal = new Goal(topology){
      for(d <- LowDelayDemands)
            add(d.latency <= 10.ms)
      for(d <- ServiceDemands)
            add(d passThrough Set1 then Set2)
      minimize(maxLoad)}
```



# DEFO returns the best solution that it finds within a configurable amount of time

```
var maxLoad = max(load,topology.links)
val goal = new Goal(topology){
    for(d <- LowDelayDemands)
        add(d.latency <= 10.ms)
    for(d <- ServiceDemands)
        add(d passThrough Set1 then Set2)
    minimize(maxLoad)}
DEFO(goal).solve(30.sec)
```

Given an input goal, DEFO computes optimized paths accommodating it

# The computation of optimized paths from high-level goals is challenging

already hard in practice

```
var maxLoad = max(load,topology.links)
val goal = new Goal(topology){
      minimize(maxLoad)}
```

# DEFO implements a heuristic approach

- assumes even load balancing
  supported by all routers for equal-cost multi-path

- limits the number of variables
  to represent optimized paths

- adopts tailored heuristics
  to compute optimized paths

# DEFO implements a heuristic approach

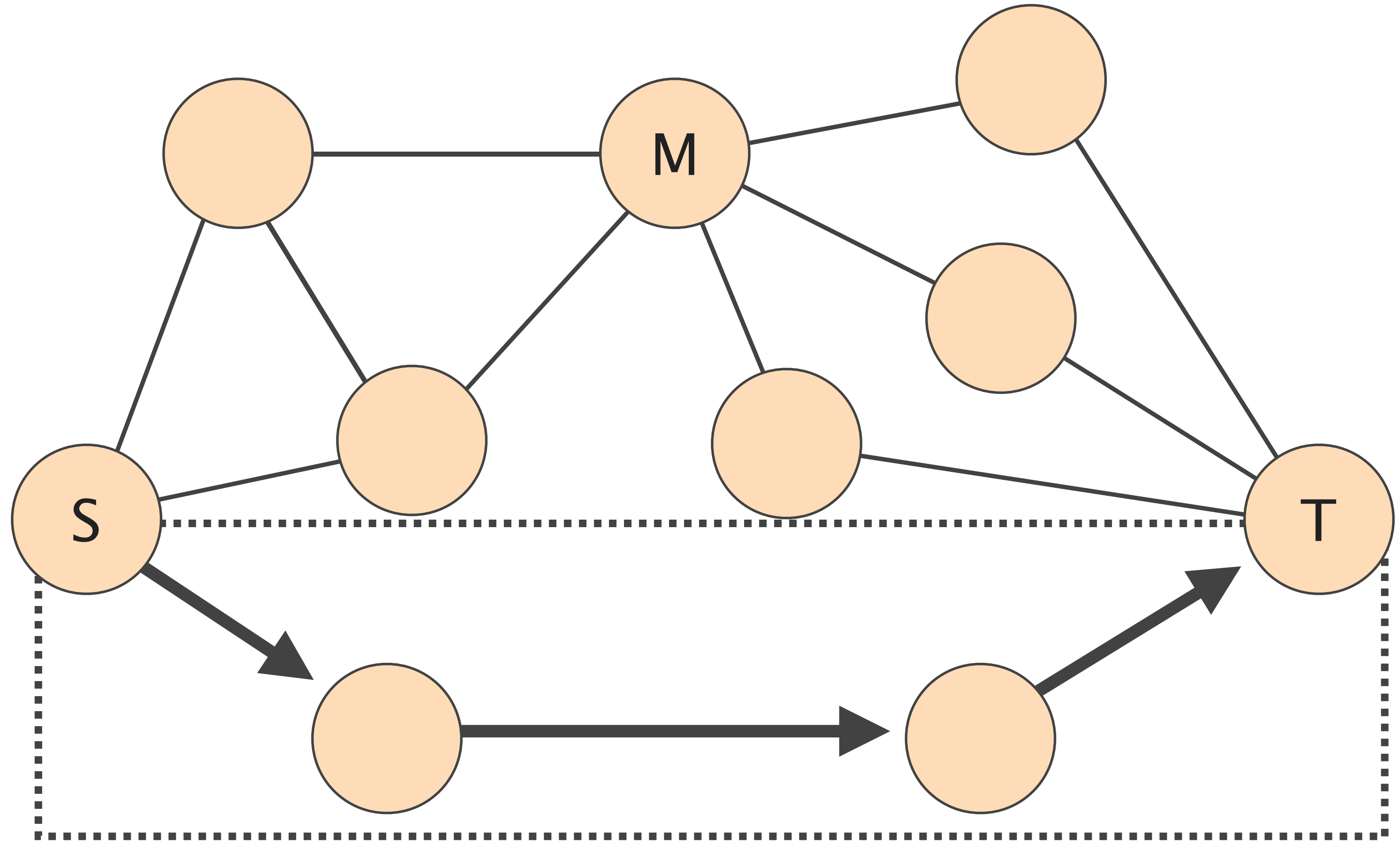
- assumes even load balancing
  supported by all routers for equal-cost multi-path

- limits the number of variables
  to represent optimized paths

- adopts tailored heuristics
  to compute optimized paths

# DEFO builds optimized paths as concatenations of default paths

DEFO representation: []

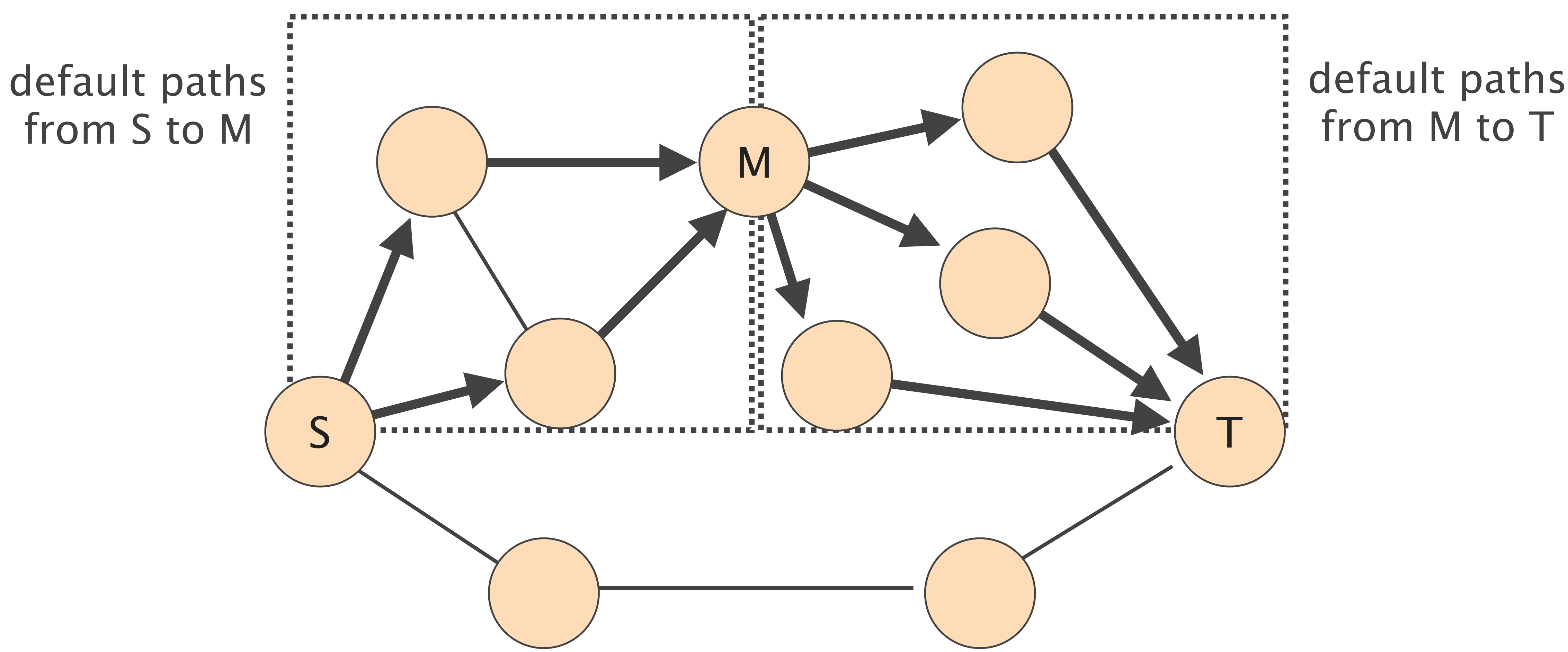



default path from S to T

# DEFO builds optimized paths as concatenations of default paths

list of middlepoints
(DEFO variable)

DEFO representation: [M]



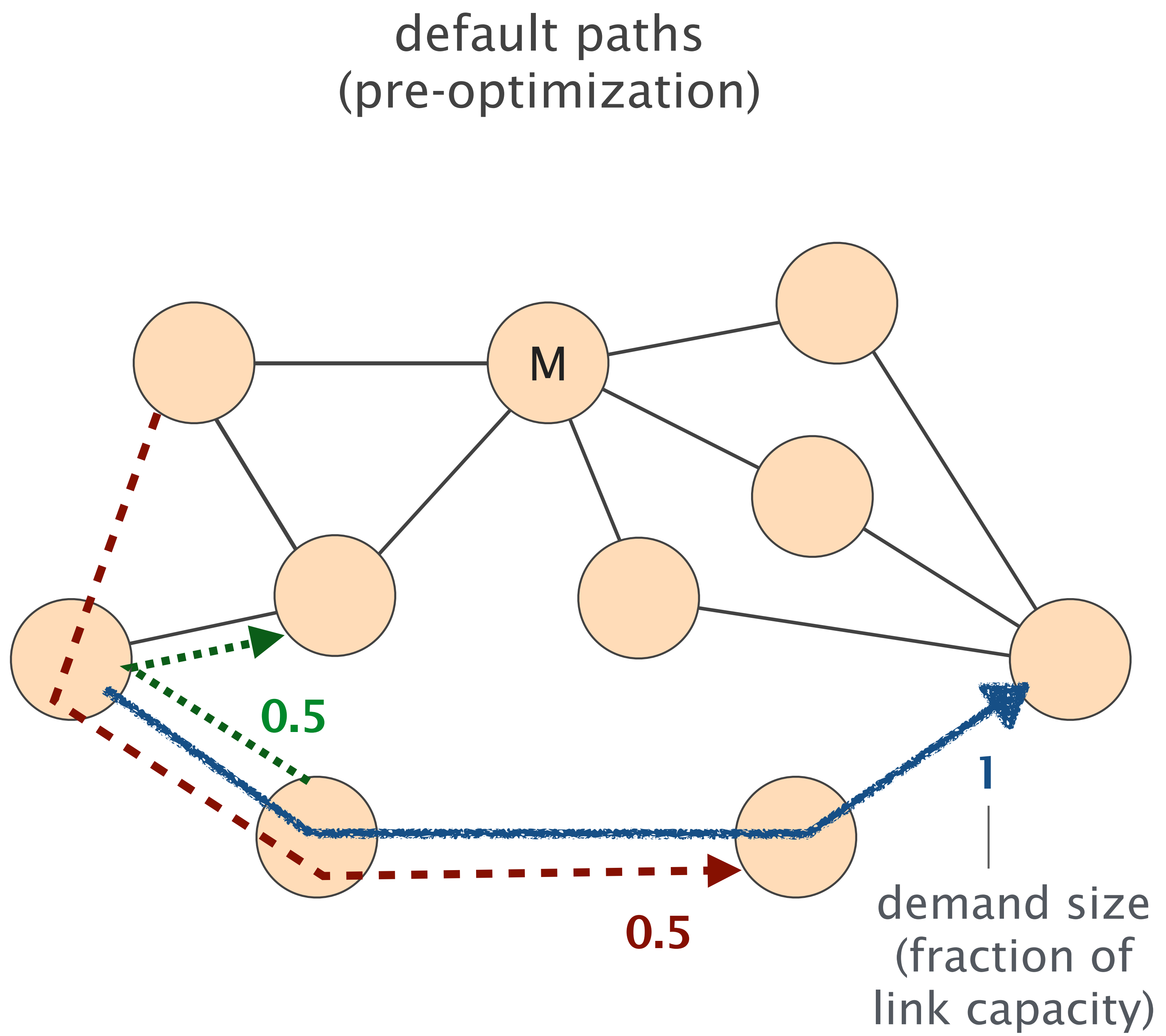
# DEFO implements a heuristic approach

- assumes even load balancing
  supported by all routers for equal-cost multi-path

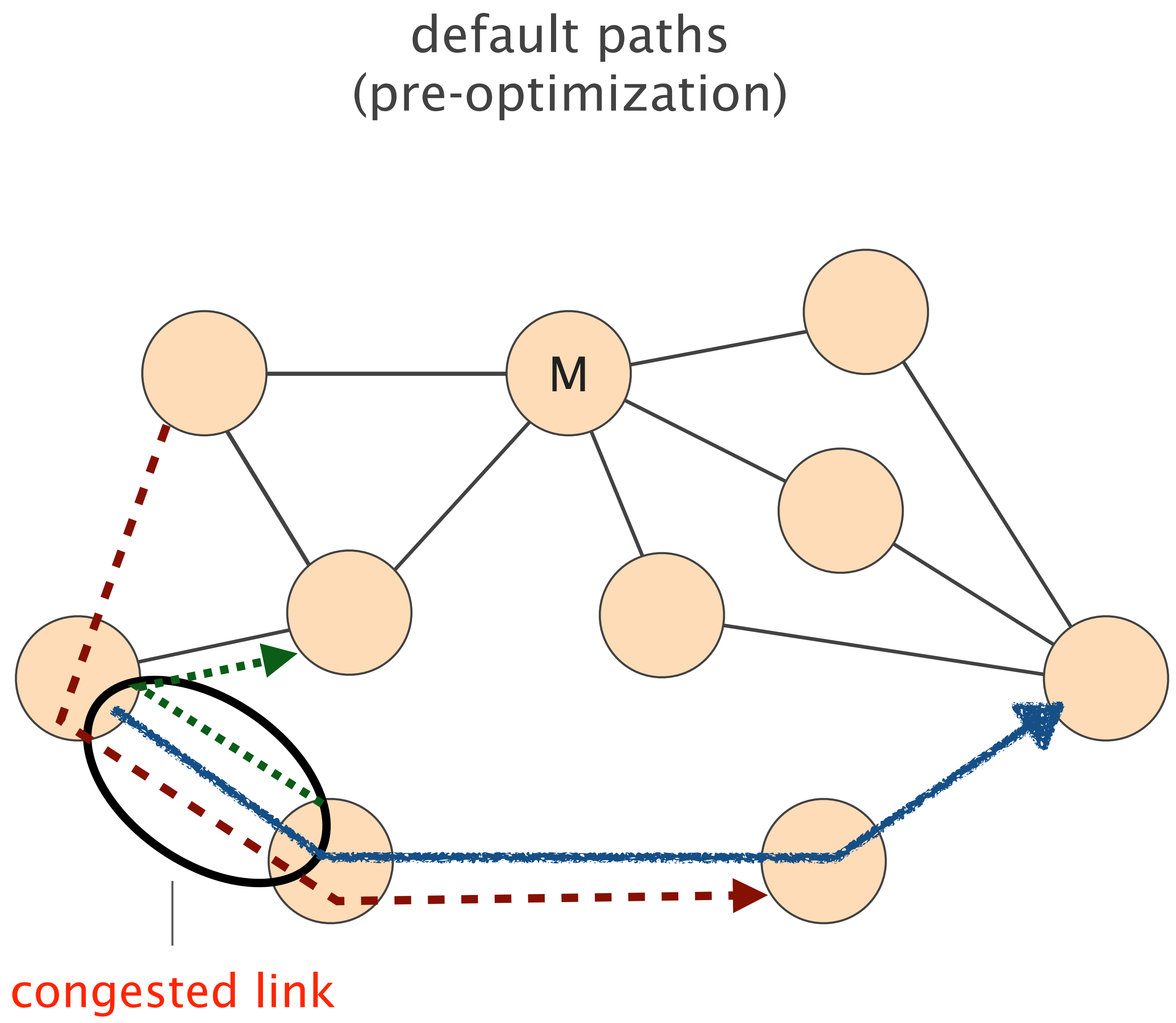
- limits the number of variables
  to represent optimized paths

- adopts tailored heuristics
  to compute optimized paths

# Consider an input network when only default (IGP) paths are configured

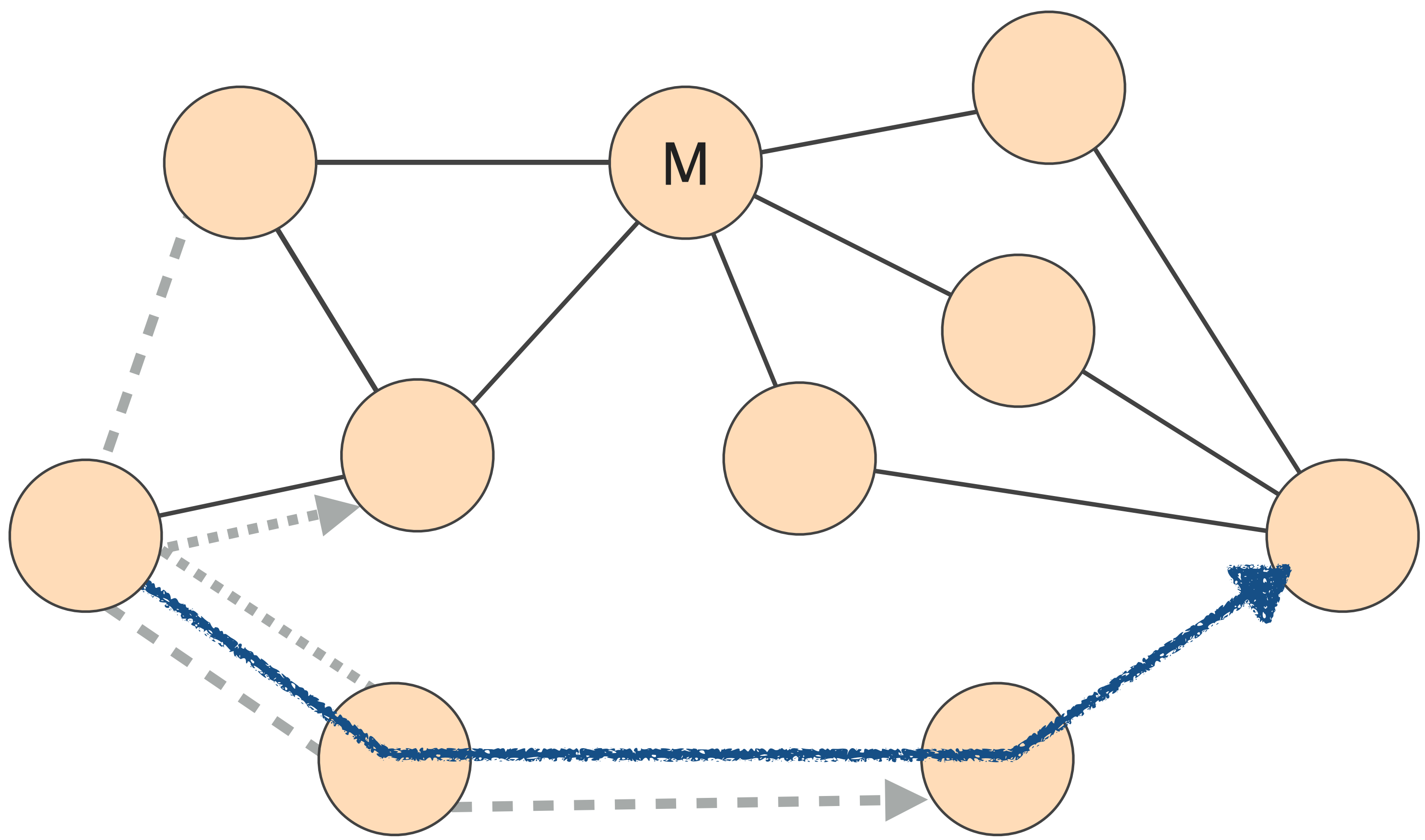


# In the example, a link is overloaded (all demands pass through it)



# DEFO heuristically optimizes forwarding, taking one demand at the time

1. select the worst demand for the objective function

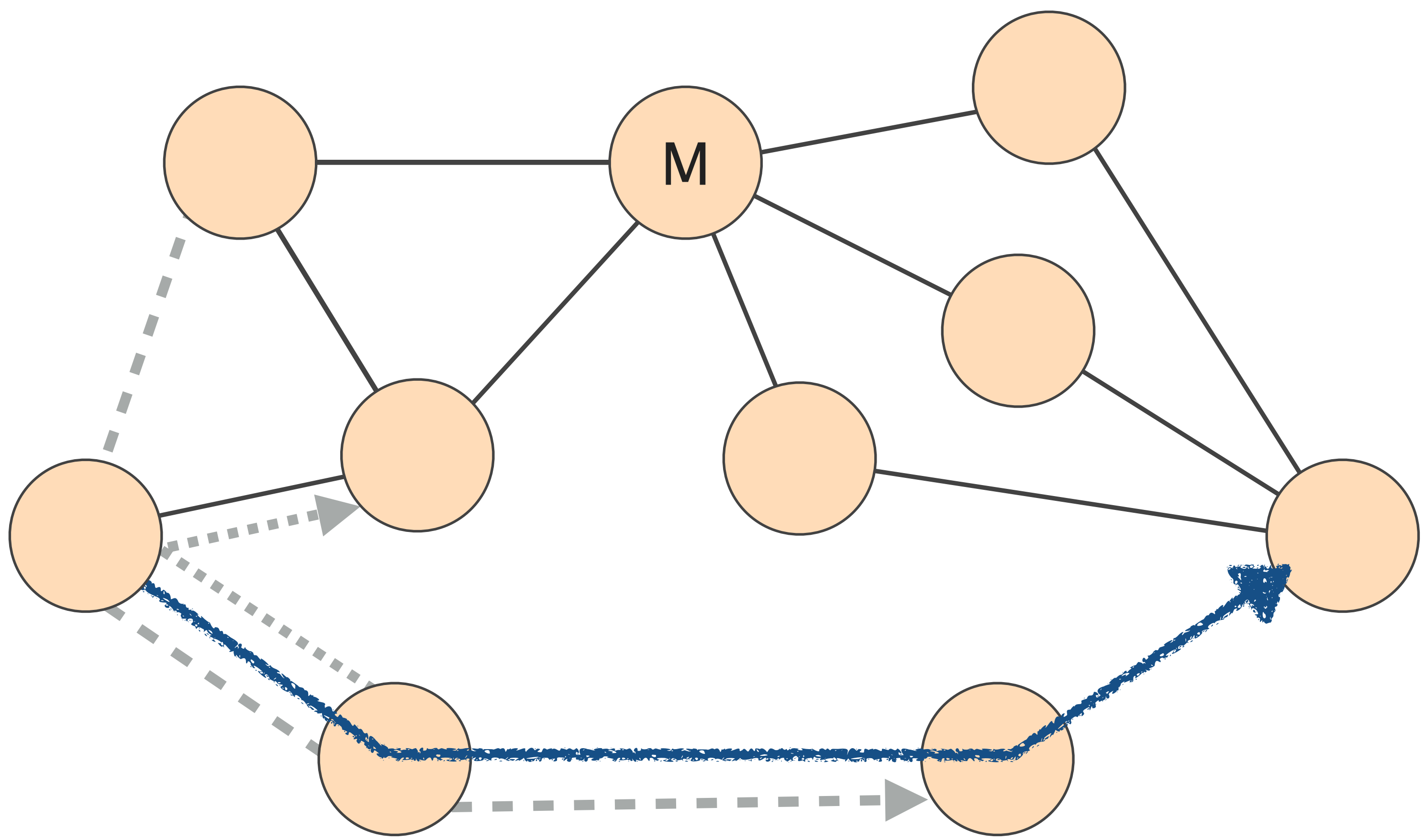


# DEFO locally optimizes every demand, first trying to use default paths

1. select the worst demand for the objective function

2. redirect the demand by iteratively

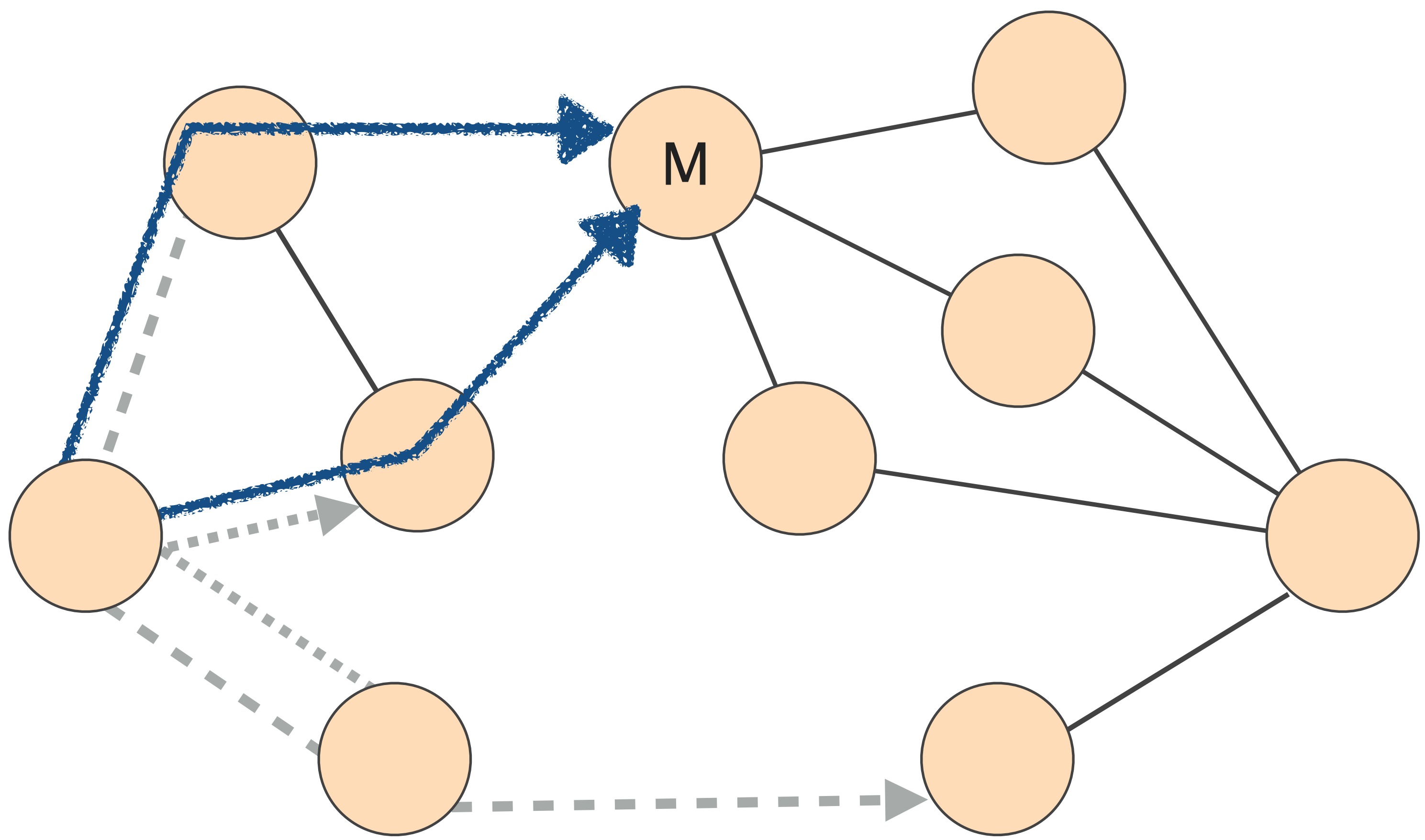
**A. try the destination**



# DEFO locally optimizes every demand, greedily selecting middlepoints

1. select the worst demand for the objective function

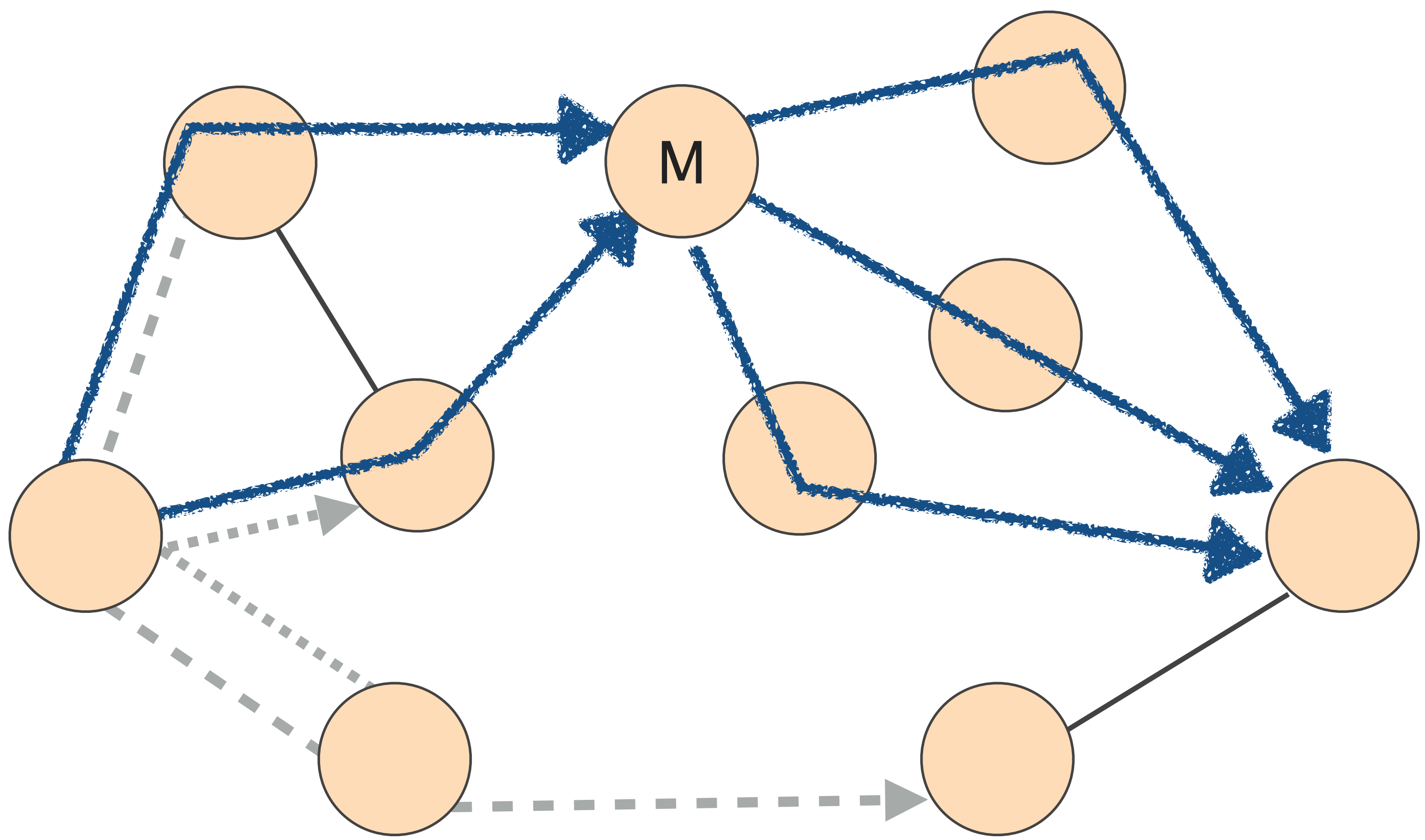
2. redirect the demand by iteratively
    A. try the destination
    **B. select the locally optimal middlepoint**



# DEFO locally optimizes every demand, using default paths as much as possible

1. select the worst demand for the objective function

2. redirect the demand by iteratively
   - **A. try the destination**
   - B. select the locally optimal middlepoint

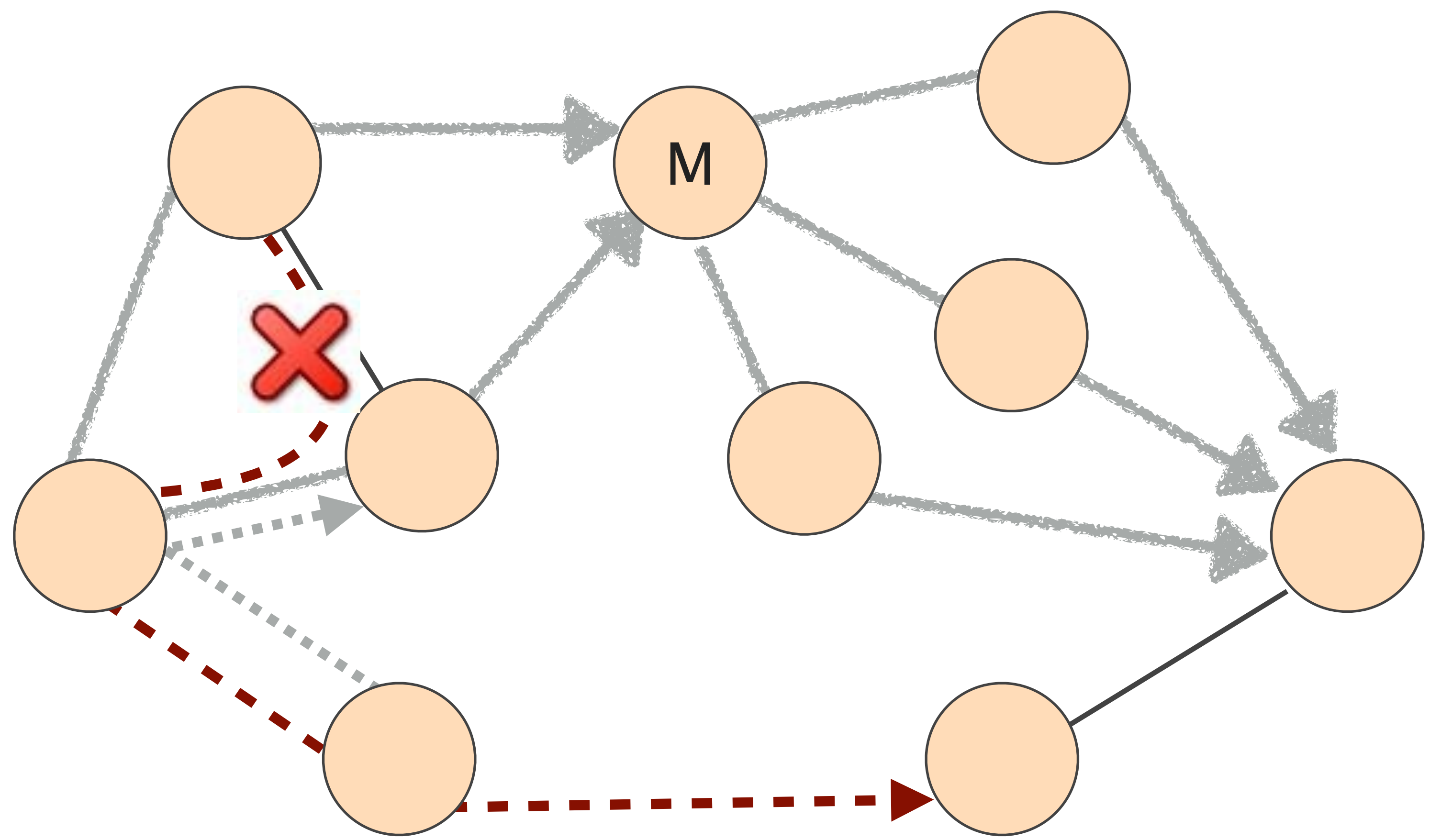


# DEFO prunes search space during computation, progressively removing unfeasible options

1. select the worst demand for the objective function

2. redirect the demand by iteratively
   A. try the destination
   B. select the locally optimal middlepoint

3. update the domain of all variables



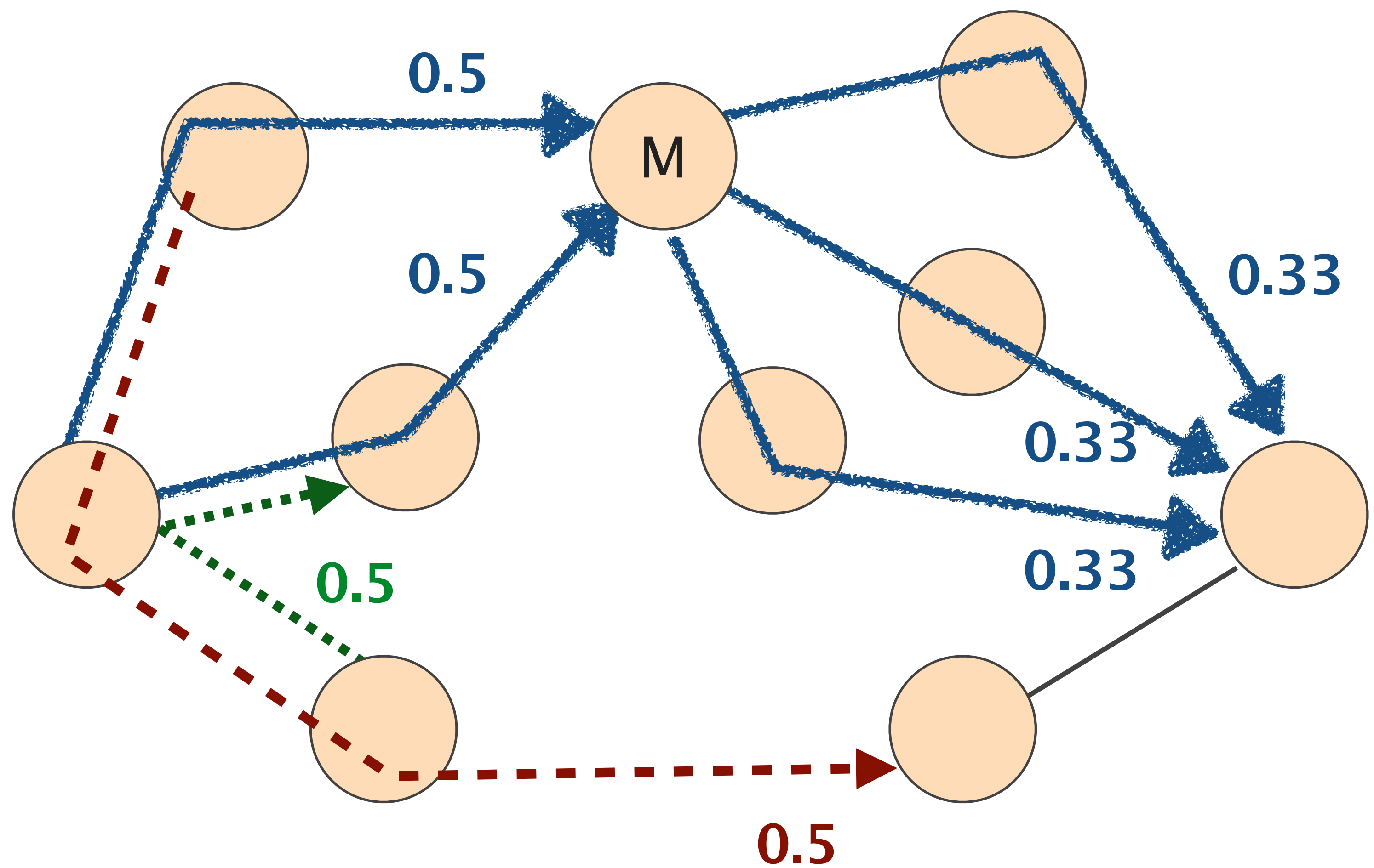
# Iterating on all demands leads to a solution (paths for all demands)

Until all demands are optimized

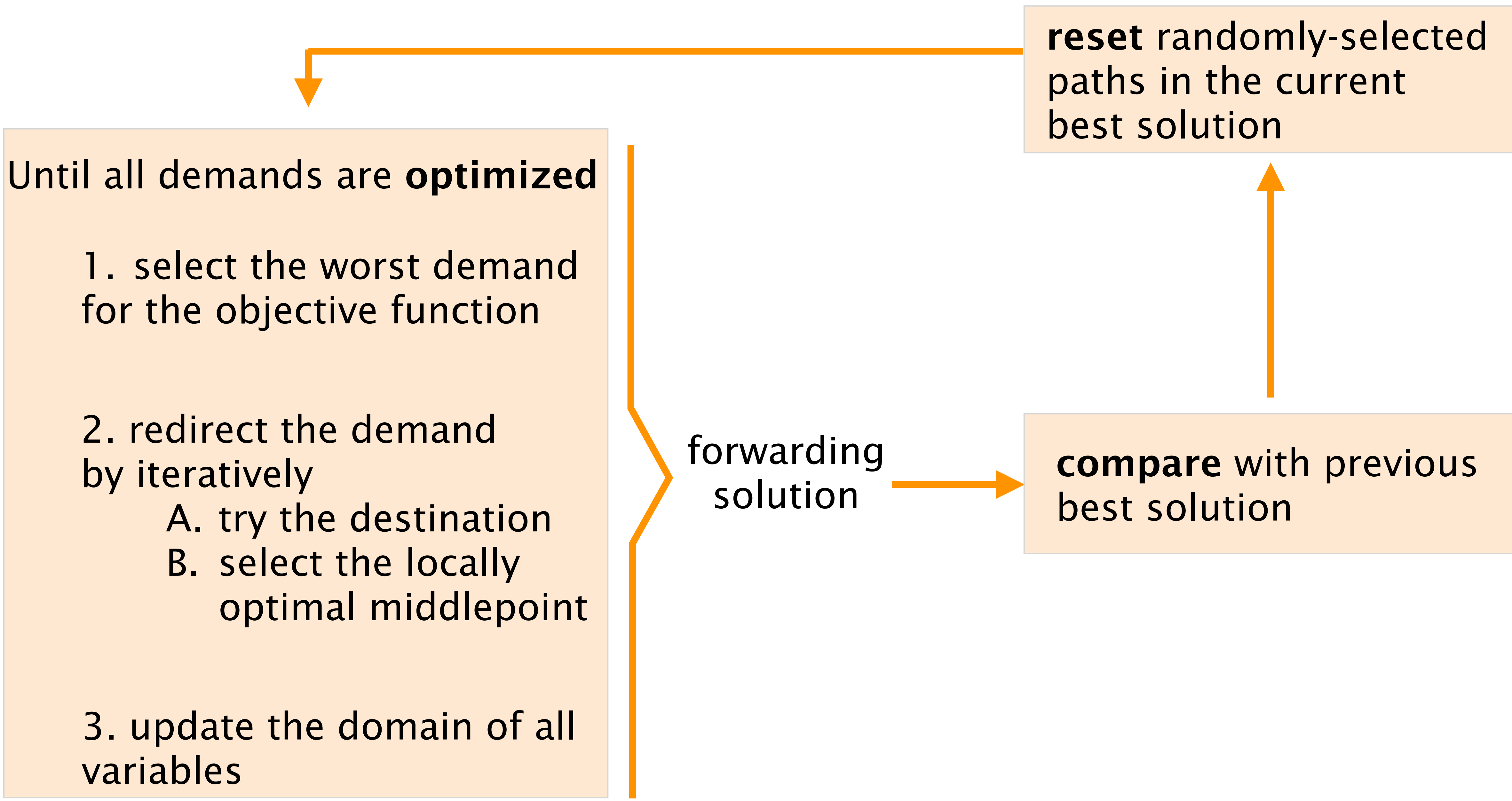
1. select the worst demand for the objective function

2. redirect the demand by iteratively
    A. try the destination
    B. select the locally optimal middlepoint

3. update the domain of all variables



# To avoid local minima, DEFO partially resets the best solution



# We implemented this approach in Constraint Programming (CP)

- ad-hoc data structures
  to store and modify middlepoints in polynomial time
- inference algorithms for each input constraint
  to update variables' domain
- customized CP search
  to implement our heuristics

# Consider bandwidth optimization on *real* networks and traffic matrices

proportionally-inflated real traffic matrices

| topology | initial load |
|---|---|
| ISP1 | 121% |
| ISP2 | 120% |
| ISP3 | 120% |
| ISP4 | 121% |

150-600 nodes
700-2,000 links
10k-115k demands

# We computed the theoretical optimum with the multi-commodity flow Linear Program (LP)

LP on a powerful server
(32-core, 96GB RAM)

| topology | initial load | theoretical optimum |
|---|---|---|
| ISP1 | 121% | 81% |
| ISP2 | 120% | 89% |
| ISP3 | 120% | NA |
| ISP4 | 121% | 86% |

theoretical optimum: fractional traffic splitting (hardly supported)

NA: ran out of memory (too many variables)

# DEFO computes *excellent* paths
# for classic goals, like traffic engineering

DEFO with max
2 middlepoints
per demand

| topology | initial load | theoretical optimum | DEFO |
|---|---|---|---|
| ISP1 | 121% | 81% | 90% |
| ISP2 | 120% | 89% | 94% |
| ISP3 | 120% | NA | 94% |
| ISP4 | 121% | 86% | 89% |

# DEFO *quickly* computes excellent paths for classic goals, like traffic engineering

| topology | initial load | theoretical optimum<br>(several hours on a powerful server (32-core, 96GB RAM)) | DEFO<br>(3 minutes on <u>this</u> laptop) |
|---|---|---|---|
| ISP1 | 121% | 81% | 90% |
| ISP2 | 120% | 89% | 94% |
| ISP3 | 120% | NA | 94% |
| ISP4 | 121% | 86% | 89% |

# DEFO *quickly* computes excellent paths for classic goals, like traffic engineering

| topology | initial load | theoretical optimum (several hours on a powerful server (32-core, 96GB RAM)) | DEFO (3 minutes on this laptop) |
|---|---|---|---|
| ISP1 | 121% | 81% | 90% |
| ISP2 | 120% | 89% | 94% |
| ISP3 | 120% | NA | 94% |
| ISP4 | 121% | 86% | 89% |

We obtained consistent results on inferred and synthetic topologies (released at http://sites.uclouvain.be/defo/)

Our evaluation shows that DEFO *outperforms* state-of-the-art traffic engineering tools (Cisco MATE)

- optimizes more than shortest-path routing
  avoiding congestion when IGP-WO cannot

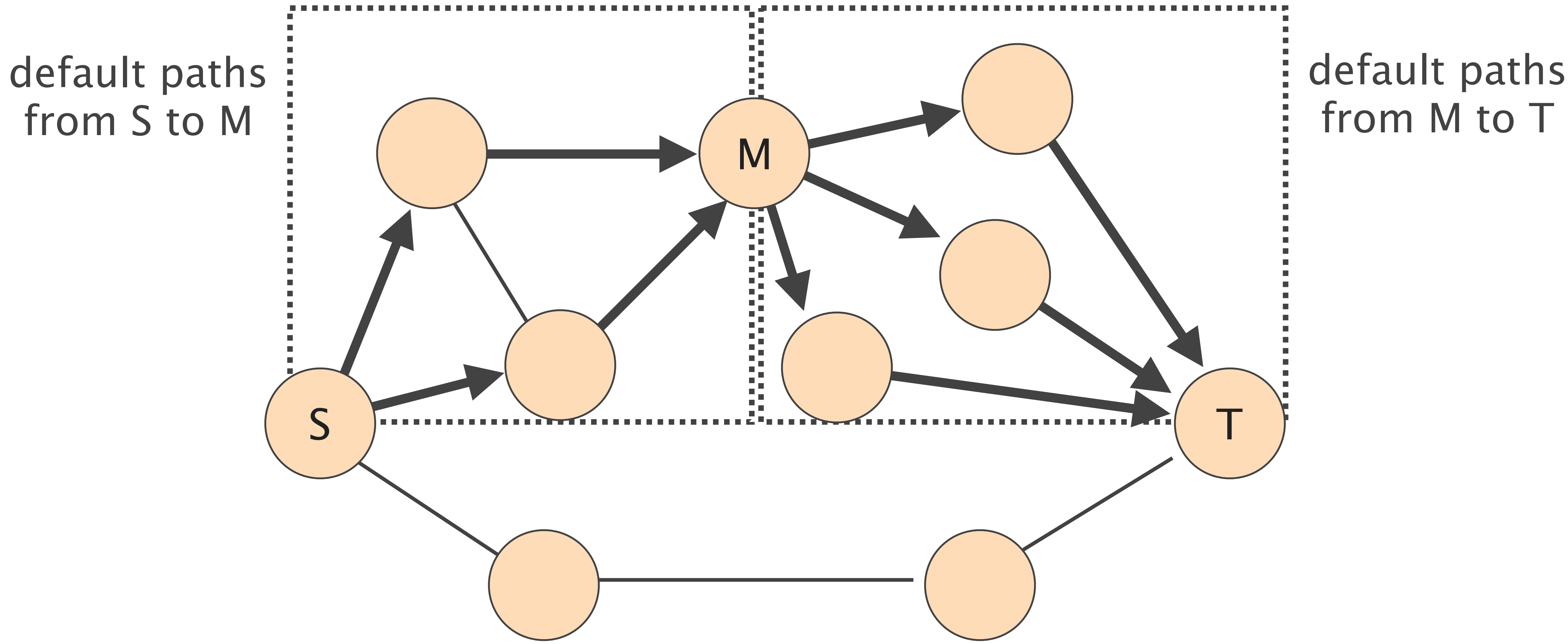
- eases operation with respect to tunneling
  requiring much less demands to be optimized

- supports a larger set of use cases
  from delay-respectful goals to service chaining

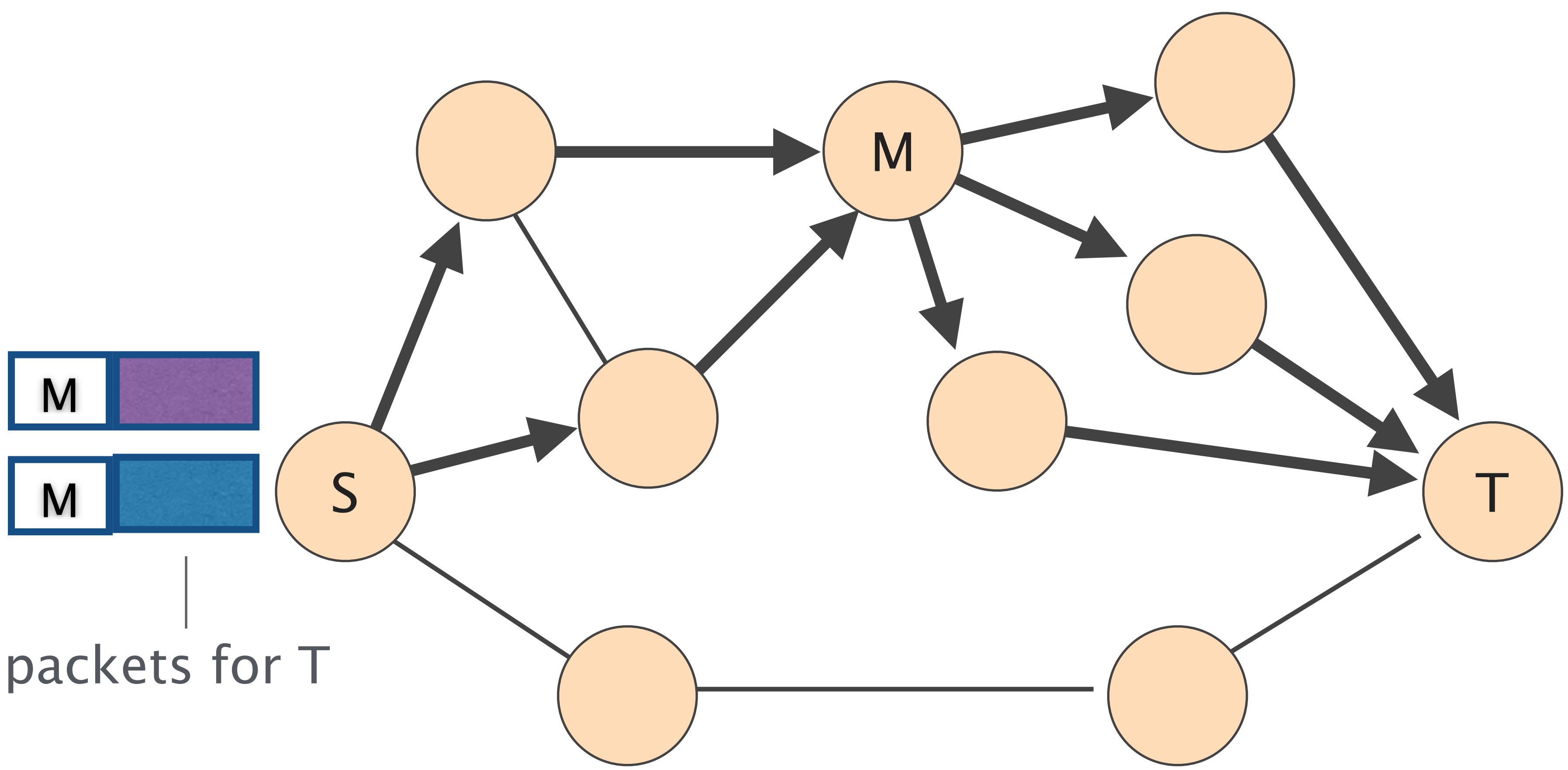
We evaluated commercial solutions
to implement DEFO paths

# Consider again an optimized path with one or more middlepoints

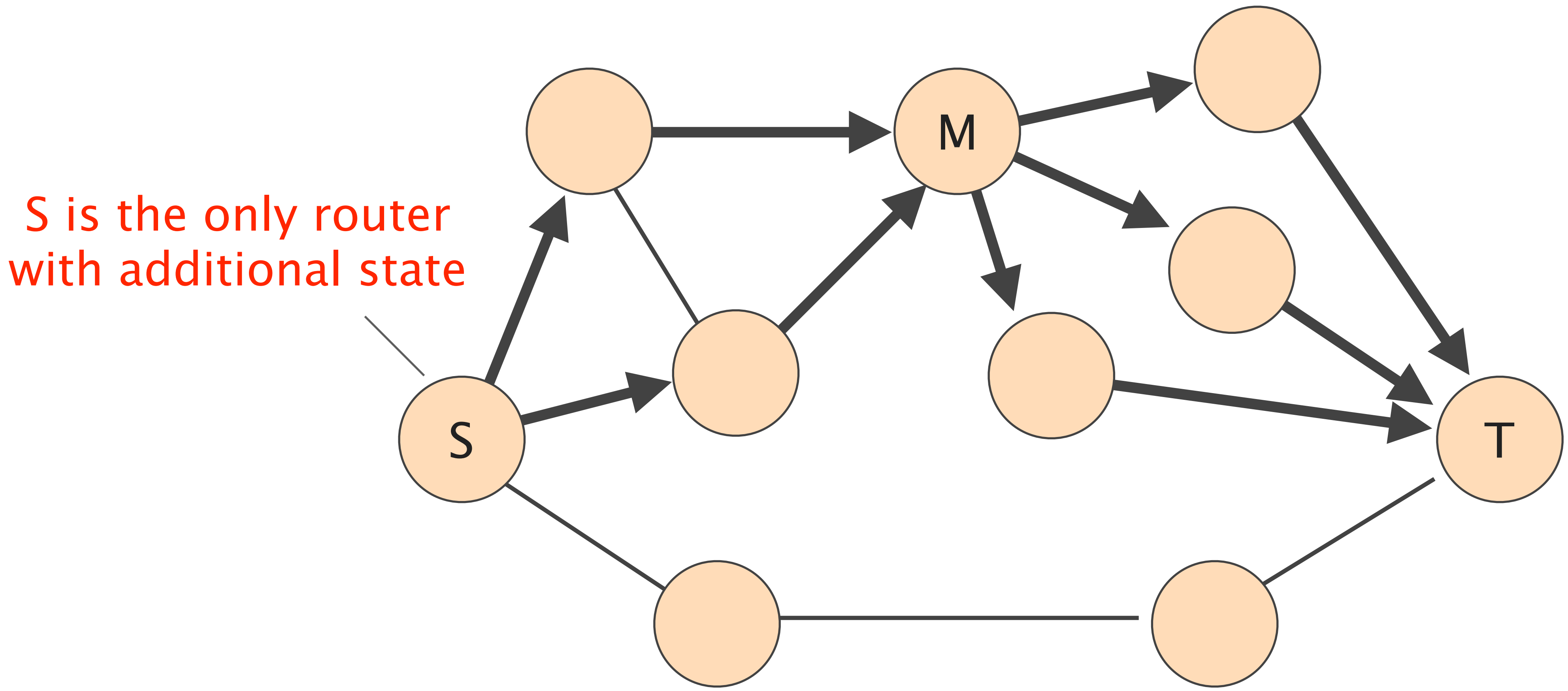
DEFO representation: [M]



# Segment Routing devices enrich packets with instructions on nodes to be traversed



# Segment Routing improves scalability in terms of state to be kept on devices



# We evaluated the scalability gain of Segment Routing, in terms of forwarding entries

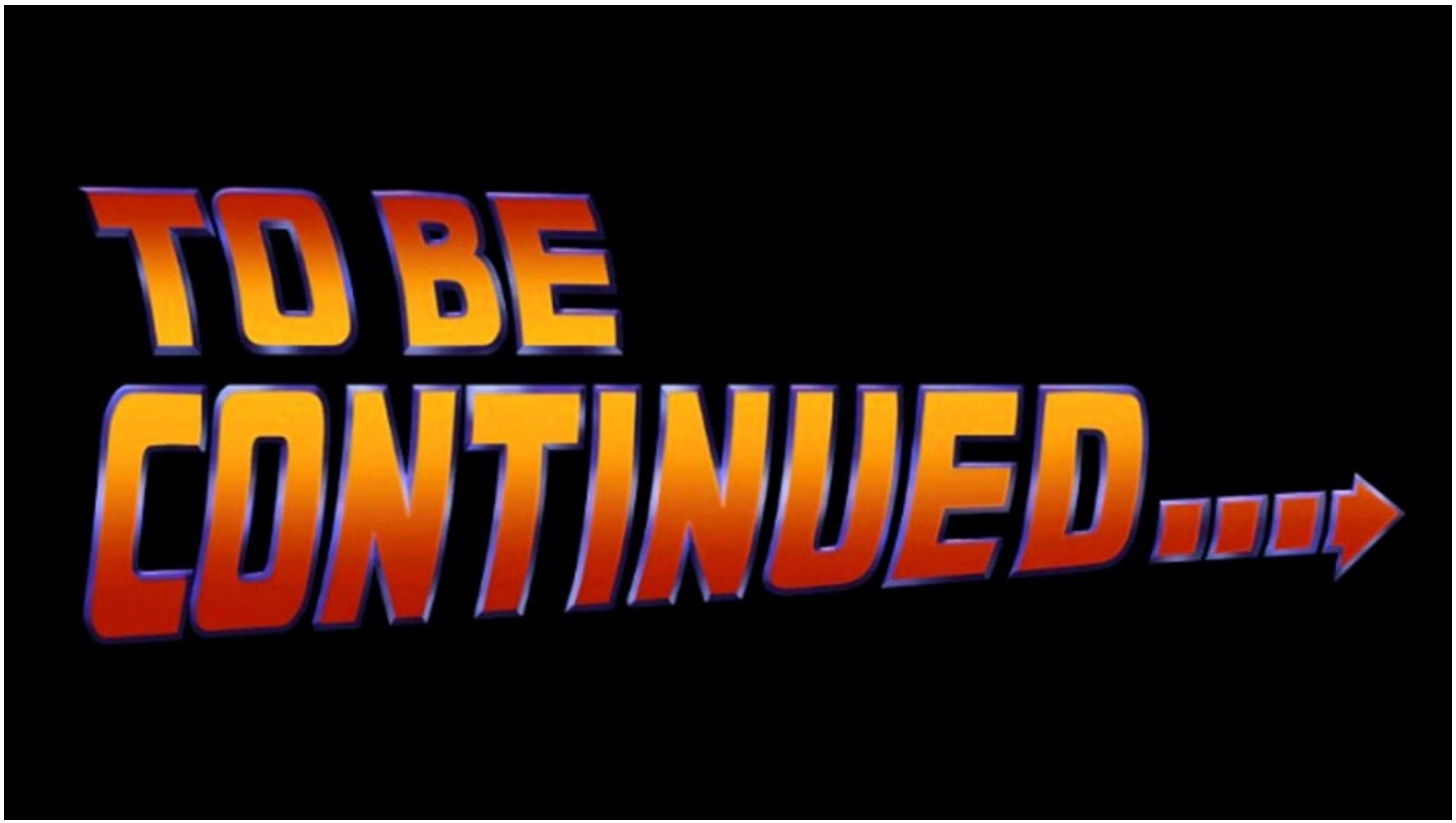
- 2-3 order of magnitude vs. hop by hop
  one entry per source-destination path per device

- 1.5-10x vs. end to end tunnelling
  one tunnel per source-destination path

- 1.5-5x vs. middlepoint to middlepoint tunnelling
  one tunnel per path between middlepoints

We leave a question open:

Is an ad-hoc protocol (Segment Routing) strictly needed?



# A Declarative and Expressive Approach to Control Forwarding in Carrier-Grade Networks