>THIS IS **THE WAY**

Speeding up the SPB Computation

Jérôme Chiabaut, Nigel Bragg

November, 2009

>THIS IS NORTEL

Motivation

- SPB eliminates signalling and its undesirable attributes from per I-SID shortest path tree installation by :
  - use of computed Group Addresses = ***fn*** (node nickname, I-SID)
  - computation of trees for "**all pairs shortest path**"
- "All pairs shortest path" (APSP) is computationally more intensive than traditional SPF using a single Dijkstra :
  - and despite 7 – 8 generations of Moore's Law since Dijkstra was first deployed in production networks, concern is still expressed over the computational implications of APSP
- This document presents work showing how the computation load of APSP may be reduced :
  - with benefits especially for the most numerous, smallest, and likely most "computationally-challenged" nodes at the network edge

SPB Computation – Classic Method

> For every node in the network do:

- **Compute SPT**: run Dijkstra for the node (the root of the spanning tree)
- **Prune paths**: keep only the shortest paths that go through the node performing the computation
- **I-SID computation**: compute the intersection of the set of I-SIDs for which the root node transmits with the set of I-SIDs for which the paths' endpoints receive

> Pluses:

- Simple, elegant, and quite efficient when most paths are not pruned
- Best possible worst-case performance

> Minuses:

- Most of the paths computed by some nodes, most notably edge nodes, end up being pruned (because they are not offering transit)
- For these nodes, the SPB computation can be very expensive relative to the amount of forwarding state produced

Simple Observations

> If we know that a node **X** in on the shortest path between nodes A & B, then the shortest path between A & B is the concatenation of the shortest paths from **X** to A & **X** to B
  - there is no need to compute APSP if there is another way to know which shortest paths go through the node of interest (**X**) **and which ones don't**

> We use the symmetry of shortest paths to reduce the number of nodes we need to consider
  - First, with a single SPF, we can divide the network into "partitions" whereby connectivity **within** a partition in known to not transit "**X**"
  - So, we need not consider the nodes in the **largest** partition, and only compute SPF for the nodes in the remaining partitions, and still have a complete solution for "**X**"

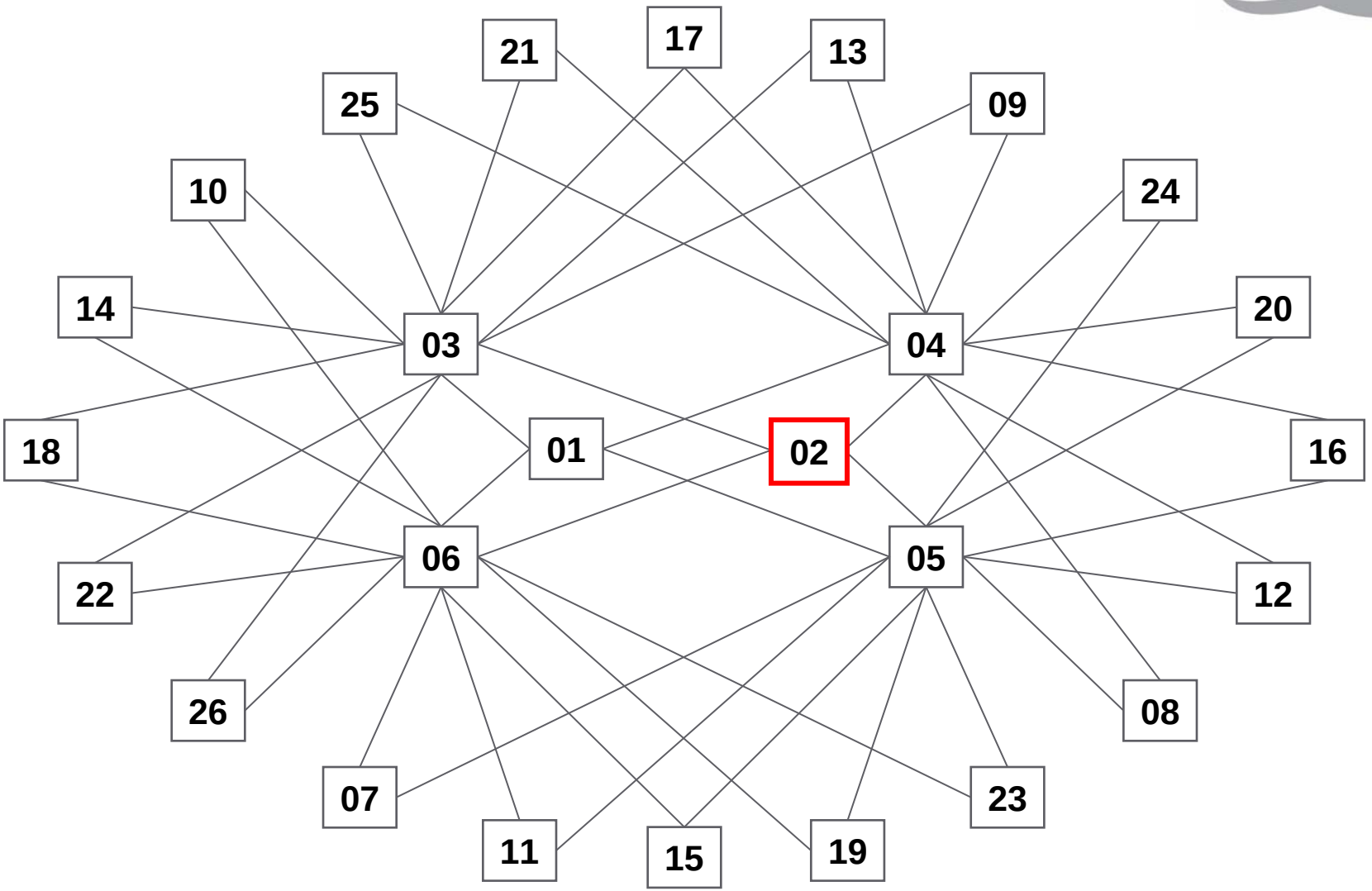
> But we can go further: it can be possible to prove, for some pairs of partitions, that **inter-**partition shortest paths do not transit "**X**"
  - The symmetry and downstream congruency properties of shortest path with tie breaking allows us to infer, from SPF computations rooted on **immediate peers**, that the shortest path between any nodes in a pair of partitions does not transit "**X**" (when the shortest path between the immediate peers does not transit "**X**")
  - This allows us to coalesce partitions together prior to removing the largest from consideration; In some cases the "coalesced" partition will be 100% of the network

Step 1: Partitioning

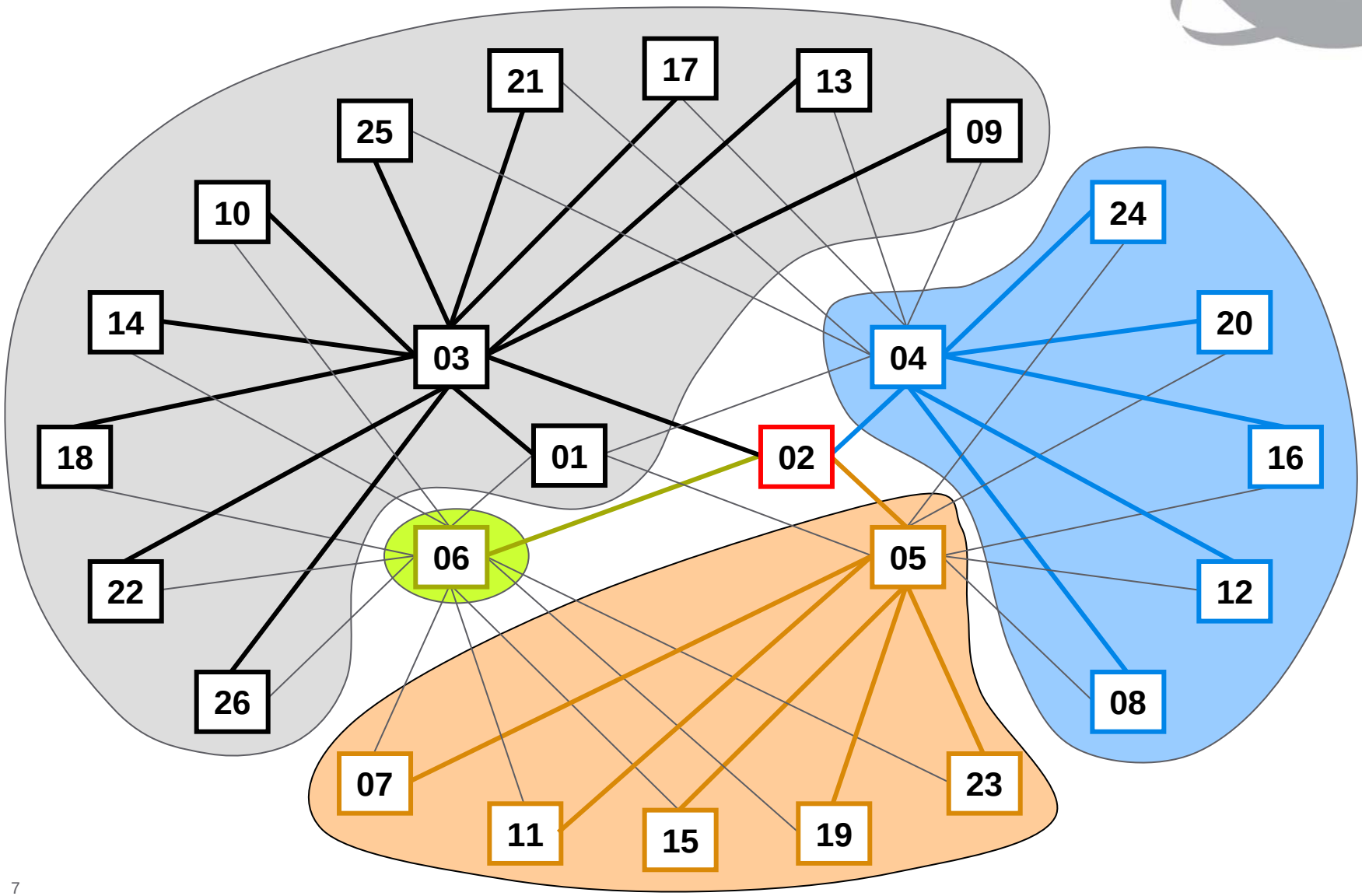
> The node computes its own spanning tree, and for each immediate neighbour computes the set of nodes in the branch of the its spanning tree which transit the neighbour
  - This partitions the set of nodes into the node itself and a set of nodes for each of its neighbours (some of which may be empty)

> The SPB problem can be solved by computing SPF for all the nodes except those in the largest partition
  - All paths of interest have their endpoints in **different** partitions
    - paths within a partition can never get closer to the node than its immediate neighbour, by construction of the “partition”,
  - so paths of interest can all be found by computing SPF for the nodes of all partitions but one, and using the symmetry property to extract the paths from the partition not explicitly computed;
  - clearly, we avoid the **largest** partition to minimise computation

The Algorithm in Pictures – Example 1



Example 1 – Step 1: Partitioning



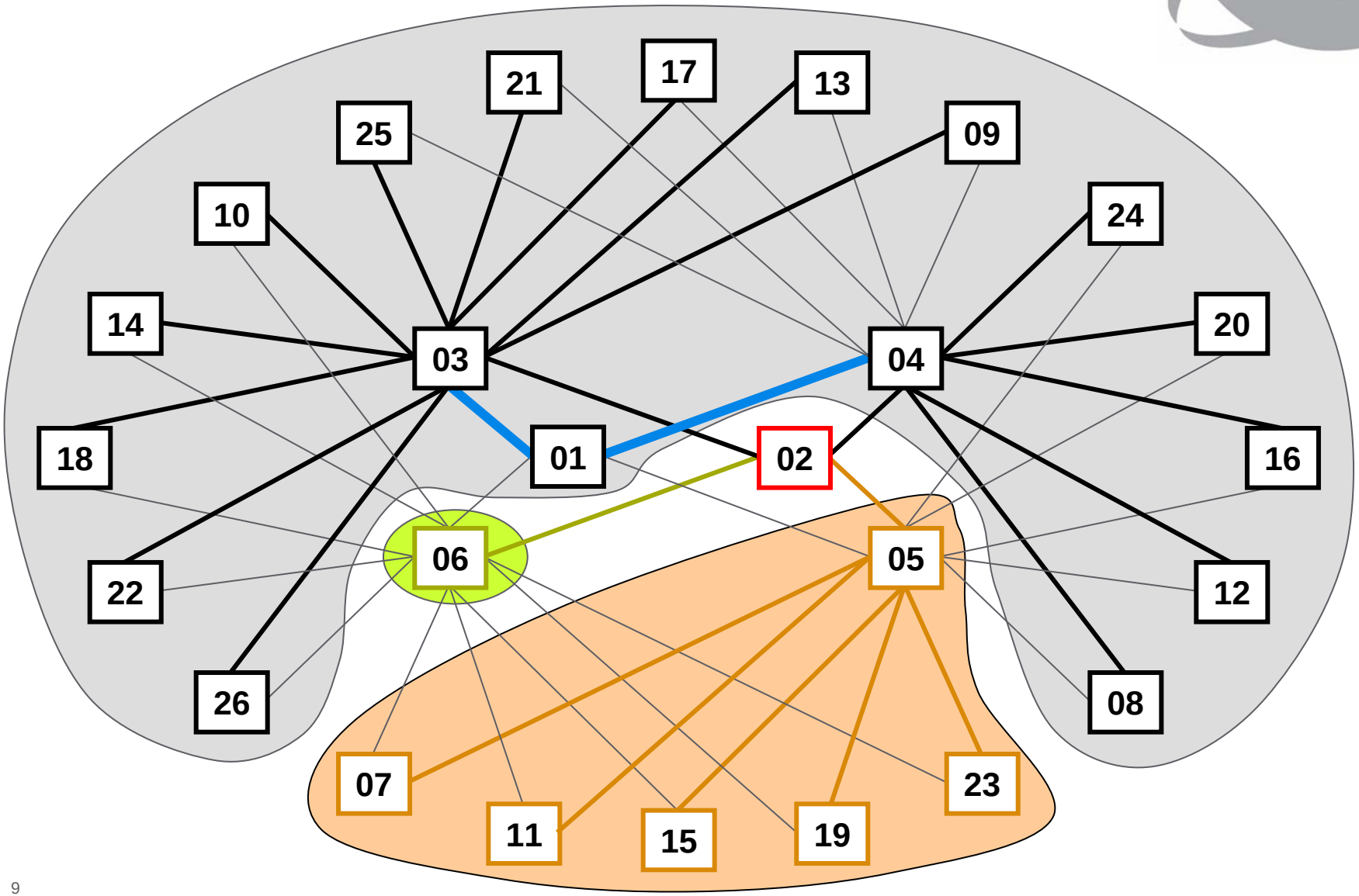
Step 2: Coalescing

> If the shortest path between two of the node's neighbours does not go through the node then all the shortest paths between the two partitions do not go through the node
  - each of the partitions produced by the spanning tree is rooted at one of the node's neighbours

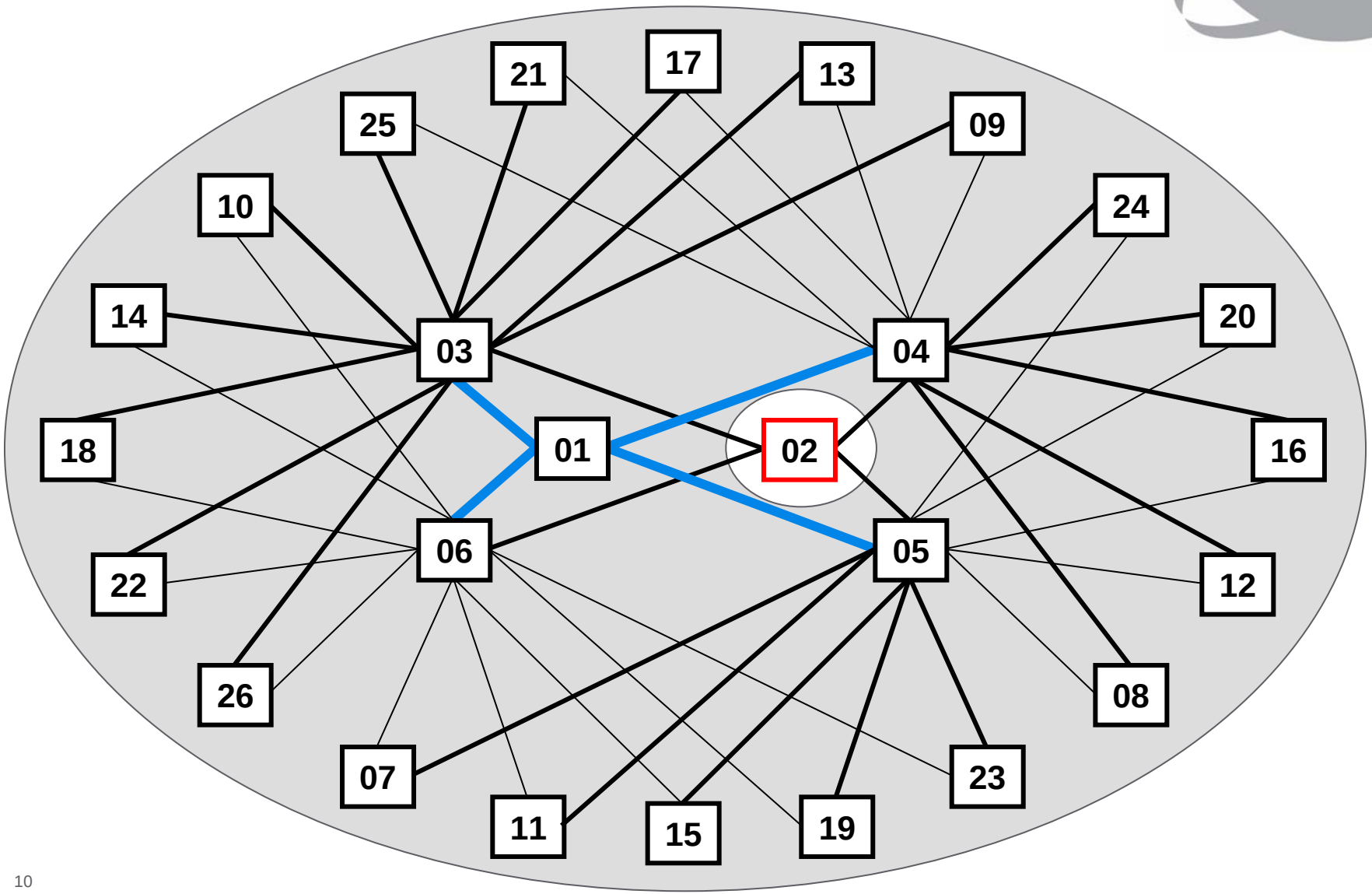
> Two such partitions that are directly connected can be coalesced into a larger partition
  - and three partitions for which the three members are pair-wise directly connected can be coalesced, etc…

> The goal is to coalesce partitions in a way that makes the largest partition as large as possible

Example 1 – Step 2: Coalescing



Example 1 – Step 2: Coalescing (Cont'd)



Step 3: Computing Shortest Paths

- Shortest paths going through the node of interest necessarily have their endpoints in different partitions
  - by construction shortest paths within partitions are necessarily shorter than paths going through the node
- Shortest paths of interest can be found by computing SPF for the nodes of all the partitions but the largest one
  - paths originating in the excluded partition can be found by reversing paths terminating there

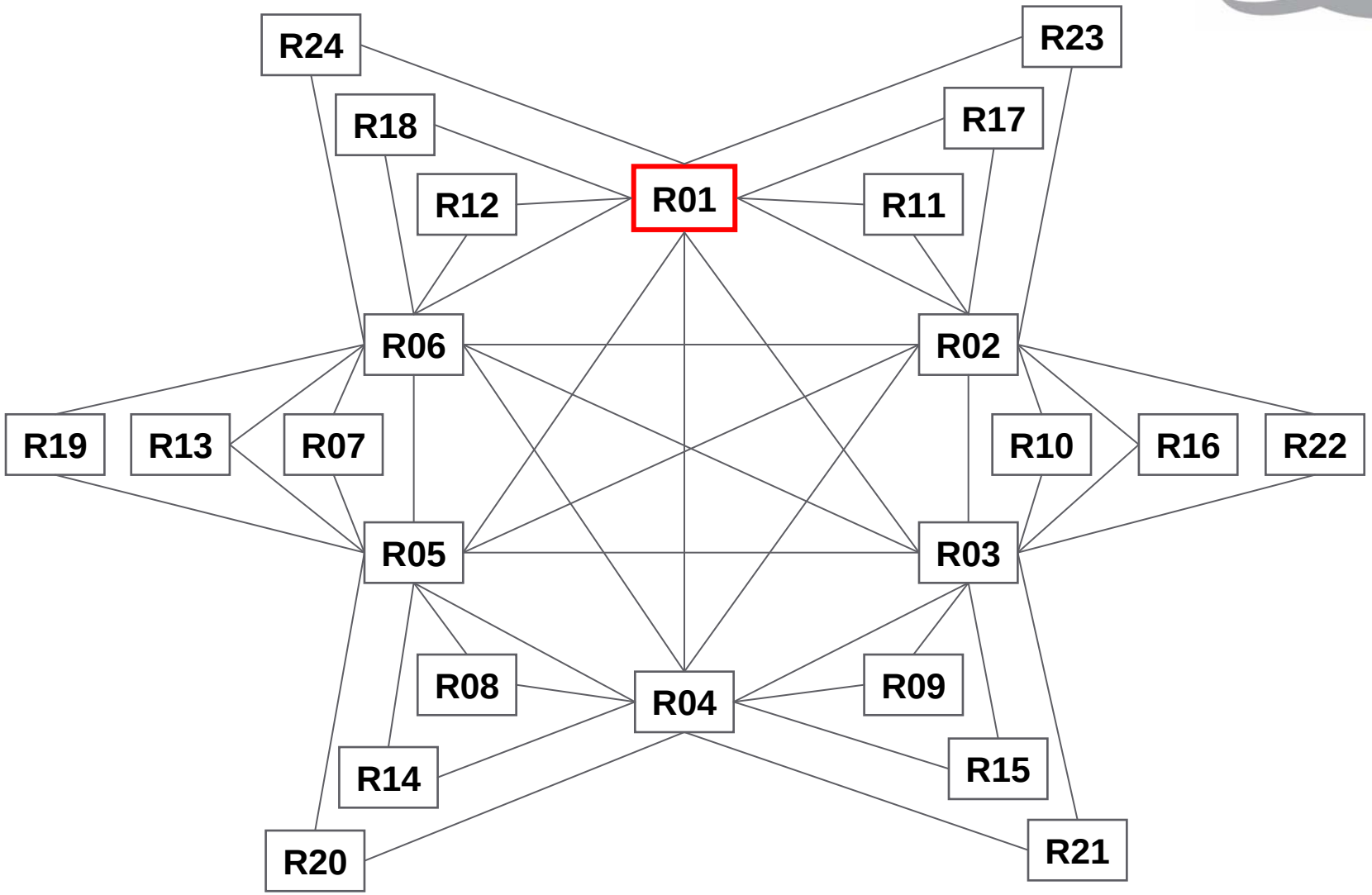
We have called this method "Some Pairs Shortest Path" (SPSP)

Example 1 – Step 3: Computing Shortest Paths

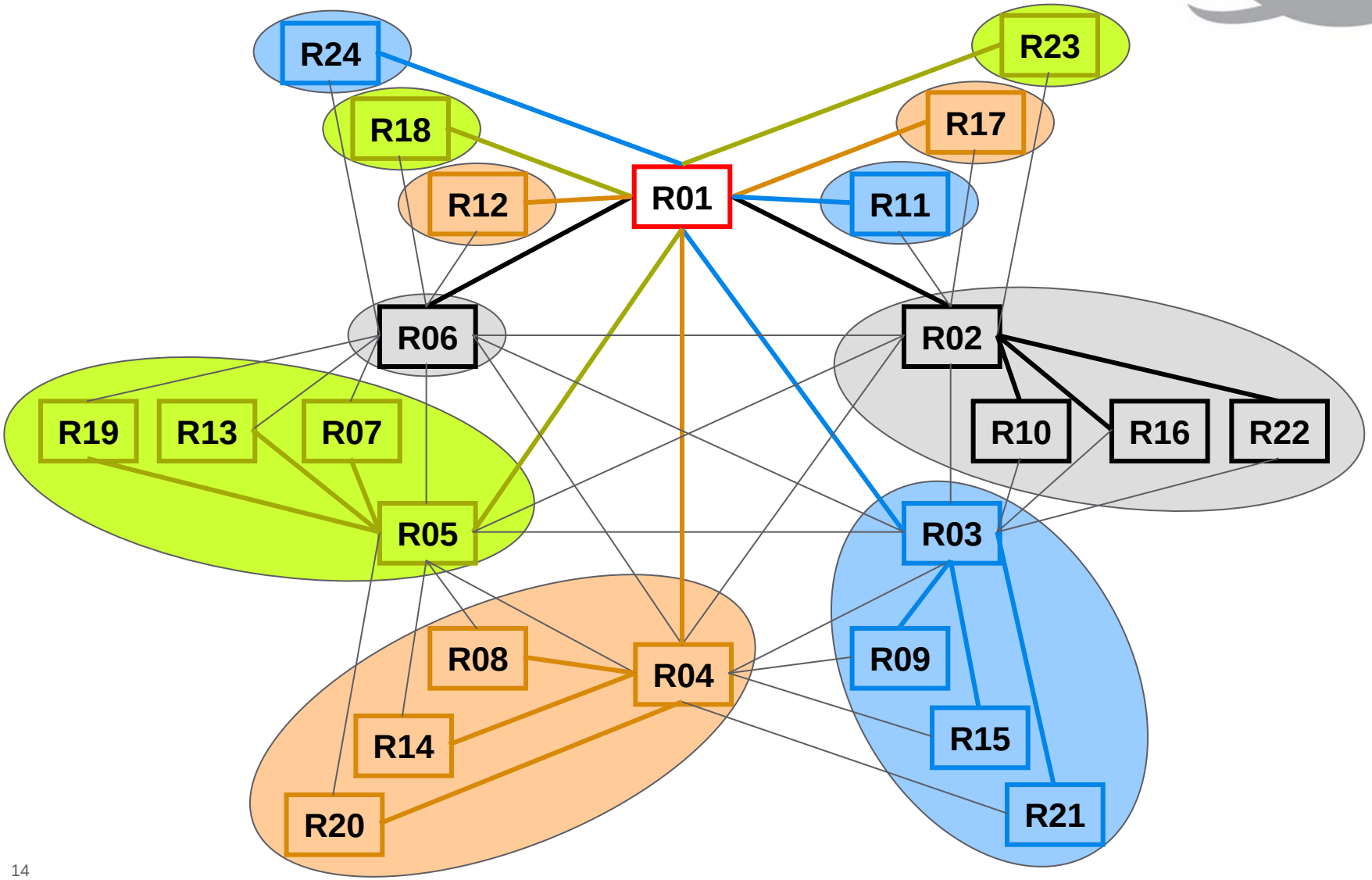
> All the neighbours' partitions have been coalesced into one: not a single shortest path goes through node 2
  - 4 x Dijkstras required by the algorithm (nodes 2, 4, 5, and 6);
  - A shorter proof exists with only 2 x Dijkstras (nodes 1 & 2),
    - but since 1 & 2 are not immediate neighbours the algorithm did not find this

> The same algorithm produces very different results for node 1:
  - The coalescing phase has no effect (because the shortest path connectivity between all partitions transits node 1)
  - The largest partition (node 3) has 11 nodes in it
  - 15 Dijkstras needed instead of 26

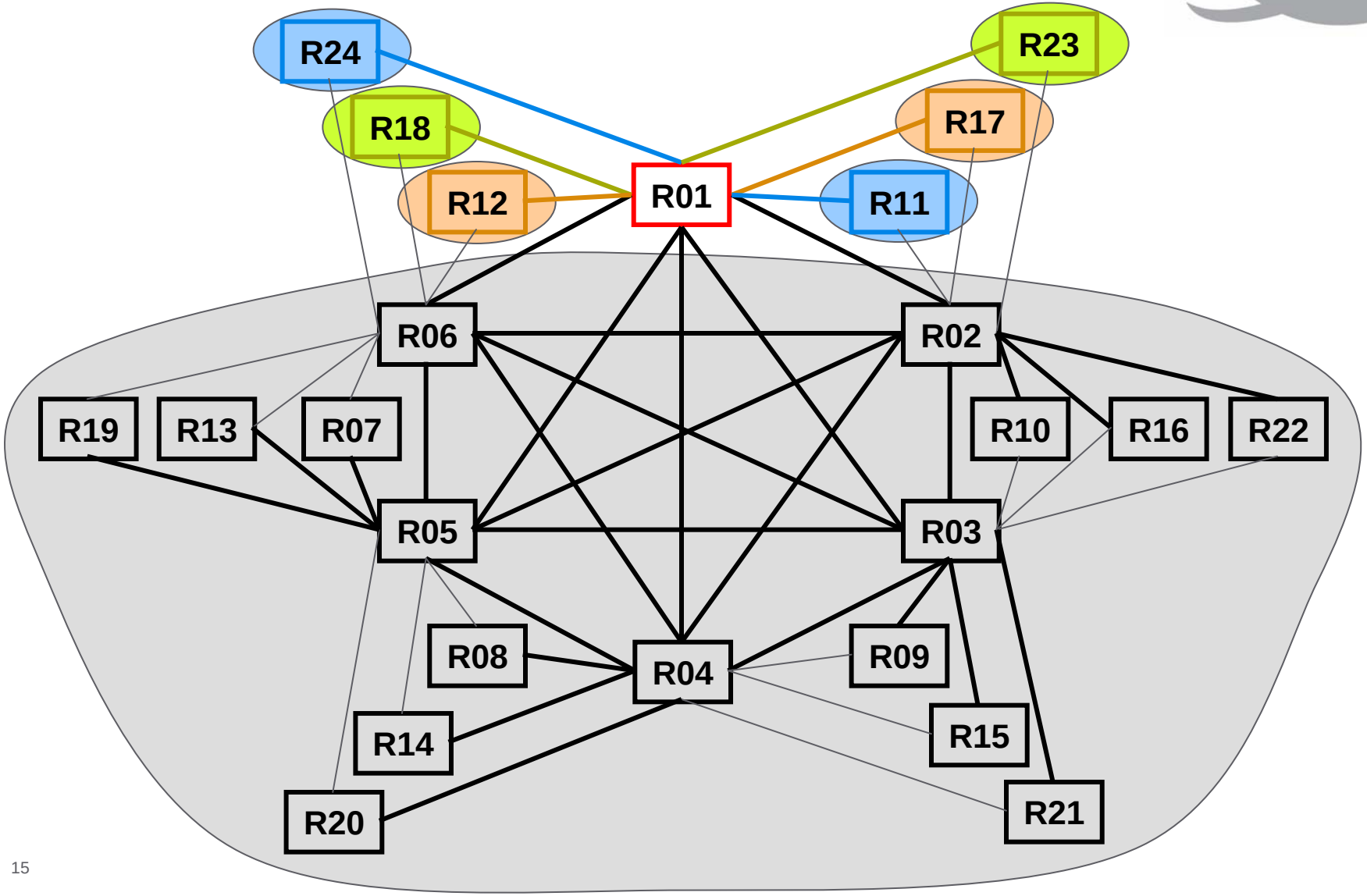
The Algorithm in Pictures – Example 2



Example 2 – Step 1: Partitioning



Example 2 – Step 2: Coalescing



Example 2 – Step 3: Computing Shortest Paths

> All the core nodes' partitions have been coalesced into one
  - 6 x Dijkstras are required by the algorithm (nodes 1 to 6)

> The edge nodes' partitions can't be coalesced because their shortest path to the core does through node 1:
  - Another 6 x Dijkstras are required to complete the computation (nodes 11, 12, 17, 17, 23, and 24)

Performance Results – Core Node

Topology	Nodes	Links	I-SIDs	FDB size	APSP	SPSP
Meshed core	200	550	20,000	8,471	**18.10 ms**	**14.0 ms**
Meshed core	392	1,106	19,600	6,470	**39.80 ms**	**21.80 ms**
Two-Tier	203	413	20,300	15,702	**31.20 ms**	**28.10 ms**
Two-Tier	402	807	20,100	15,432	**74.70 ms**	**63.30 ms**
Two-Tier	803	1,639	20,075	14,196	**224.3 ms**	**135.8 ms**
Light mesh	200	700	20,000	5,808	**21.40 ms**	**16.30 ms**
Light mesh	392	1,568	19,600	4,302	**86.50 ms**	**59.80 ms**
Light mesh	800	3,600	20,000	2,897	**411.2 ms**	**274.6 ms**
Heavy mesh	200	460	20,000	8,296	**19.20 ms**	**19.10 ms**
Heavy mesh	392	924	19,600	6,356	**47.60 ms**	**46.90 ms**
Heavy mesh	800	1,920	20,000	4,861	**163.1 ms**	**162.5 ms**

The new method is never slower than APSP

Performance Results – Edge Node

Topology	Nodes	Links	I-SIDs	FDB size	APSP	SPSP
Meshed core	200	550	20,000	560	**6.30 ms**	**1.0 ms**
Meshed core	392	1,106	19,600	283	**23.10 ms**	**1.70 ms**
Two-Tier	203	413	20,300	609	**8.40 ms**	**1.10 ms**
Two-Tier	402	807	20,100	300	**30.90 ms**	**1.80 ms**
Two-Tier	803	1,639	20,075	152	**139.1 ms**	**4.90 ms**
Light mesh	200	700	20,000	560	**16.0 ms**	**1.10 ms**
Light mesh	392	1,568	19,600	283	**78.8 ms**	**2.10 ms**
Light mesh	800	3,600	20,000	157	**401.0 ms**	**5.60 ms**
Heavy mesh	200	460	20,000	560	**8.20 ms**	**1.10 ms**
Heavy mesh	392	924	19,600	283	**31.70 ms**	**1.70 ms**
Heavy mesh	800	1,920	20,000	157	**137.8 ms**	**4.80 ms**

The new method can be much faster than APSP

Conclusion

> The proposed algorithm reduces the number of Dijkstras that must be performed during the SPB computation
  - Effectiveness depends upon the topology and the position of the node in that topology

> This algorithm is very effective in the following cases:
  - Nodes with very few neighbours (e.g. dual-connected nodes)
  - Nodes that are close to the network periphery (e.g. smaller nodes)
  - Nodes in a lightly meshed core

> It is most beneficial for high volume and cost sensitive edge nodes that offer SPB transit but need compute only a few trees
  - Core nodes' computation could be reduced by about 50% in some cases, i.e. one Moore's Law generation

> This algorithm is a significant implementation optimisation of SPB which enhances network scaling

Backup Slides - topologies

Topologies – Meshed Cores

> Topologies composed of a meshed core surrounded by dual-homed edges
  - Key parameters are the size of the mesh and the size and degree of meshiness of the core

> Lightly meshed large cores
  - Size/2 core routers with 2(log2(Size)-2) trunks and 2 lines each
  - Size/2 edge routers dual homed (2 links)
  - Size*Log2(Size)/2 point-to-point links
  - Size of core routers' neighbourhood of core nodes grows a log(Size)
  - Network diameter increases very slowly with size (4-6 typical)

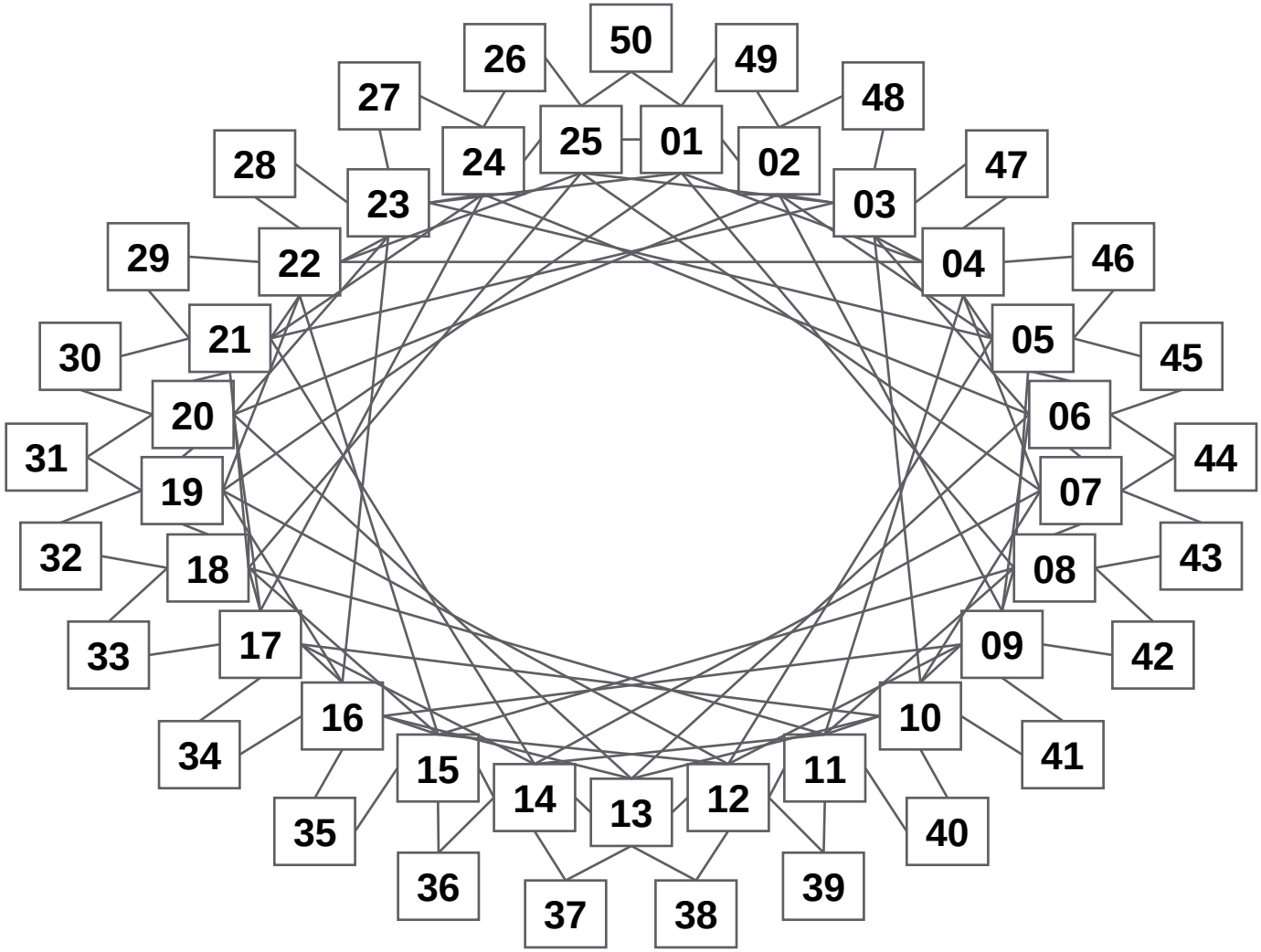
> Fully meshed small cores
  - sqrt(2*Size) core routers with 2*sqrt(Size)-1 trunks and 2*sqrt(Size)-2 lines each
  - Size-sqrt(2*Size) edge routers dual homed (2 links)
  - 3*Size-5/2*sqrt(2*Size) point-to-point links
  - Size of core routers' neighbourhood of core nodes grows a sqrt(Size)
  - Network diameter is always 3

Topologies – Two-Tier Hierarchy

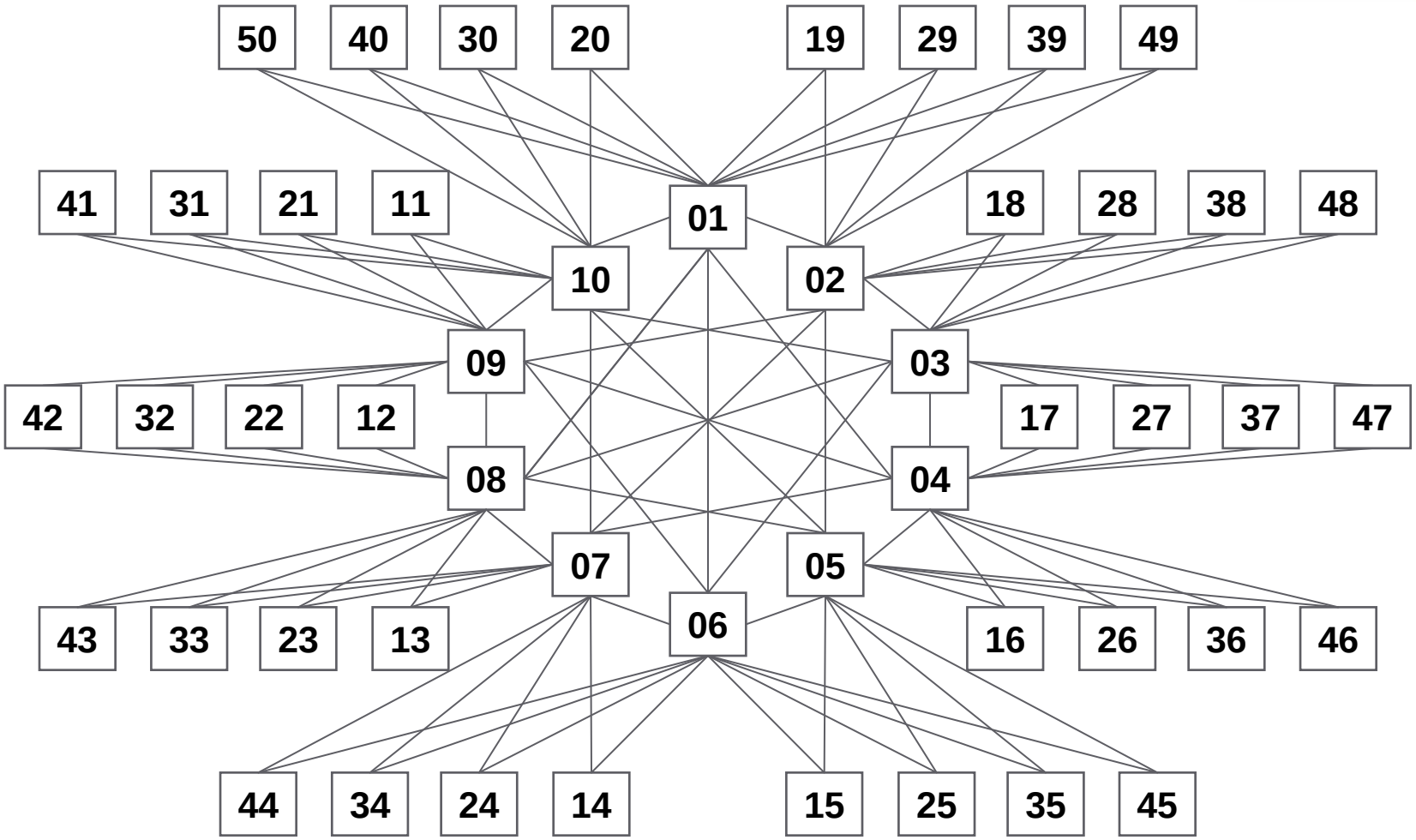
> Topologies composed of a small core surrounded by dual-homed metro nodes surrounded by dual-homed edge nodes
  - Symmetric point-to-point links with metrics between 1 and 9
  - The instance running the algorithm is the core node with the smallest ID
  - Key parameters are the size of the core, metro, and edge

> Two-Tier Hierarchy
  - N core nodes, fully meshed (except for N=2)
  - M metro nodes, dual-homed onto core nodes
  - L edge nodes, dual-homed onto metro nodes
  - L+M+N nodes and 2(L+M)+N(N-1)/2 links in total (2(L+M) for N=2)
  - L/M ~ M/N vary between 4 and 6
  - Small network diameter: 5 (4 for N=2)

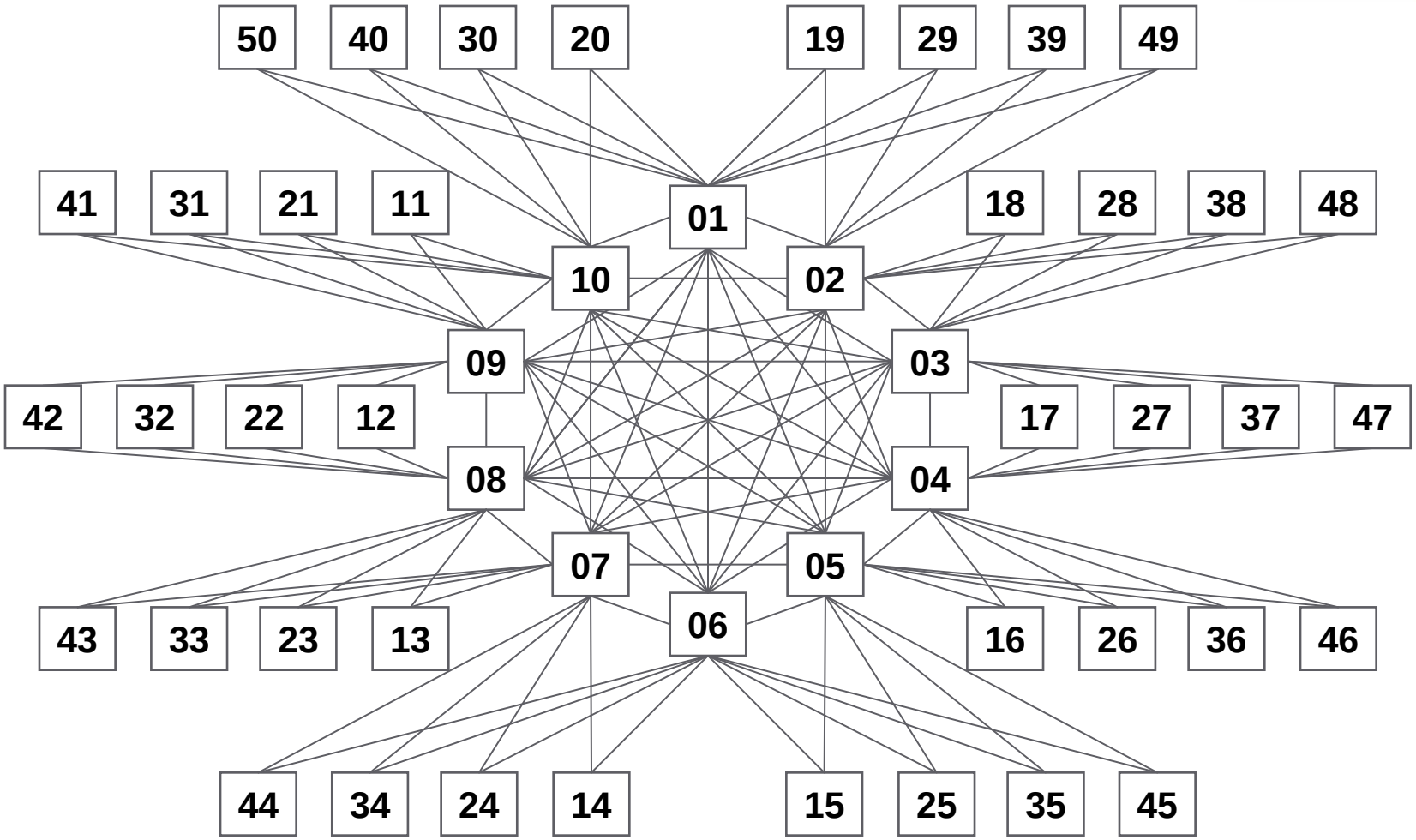
Light Mesh – 50 Switches



Heavy Mesh – 50 Switches



Meshed Core – 50 Switches



Two-Tier – 52 Switches