Recursive Orchestration of Allotted Resources with Dynamic Instantiation

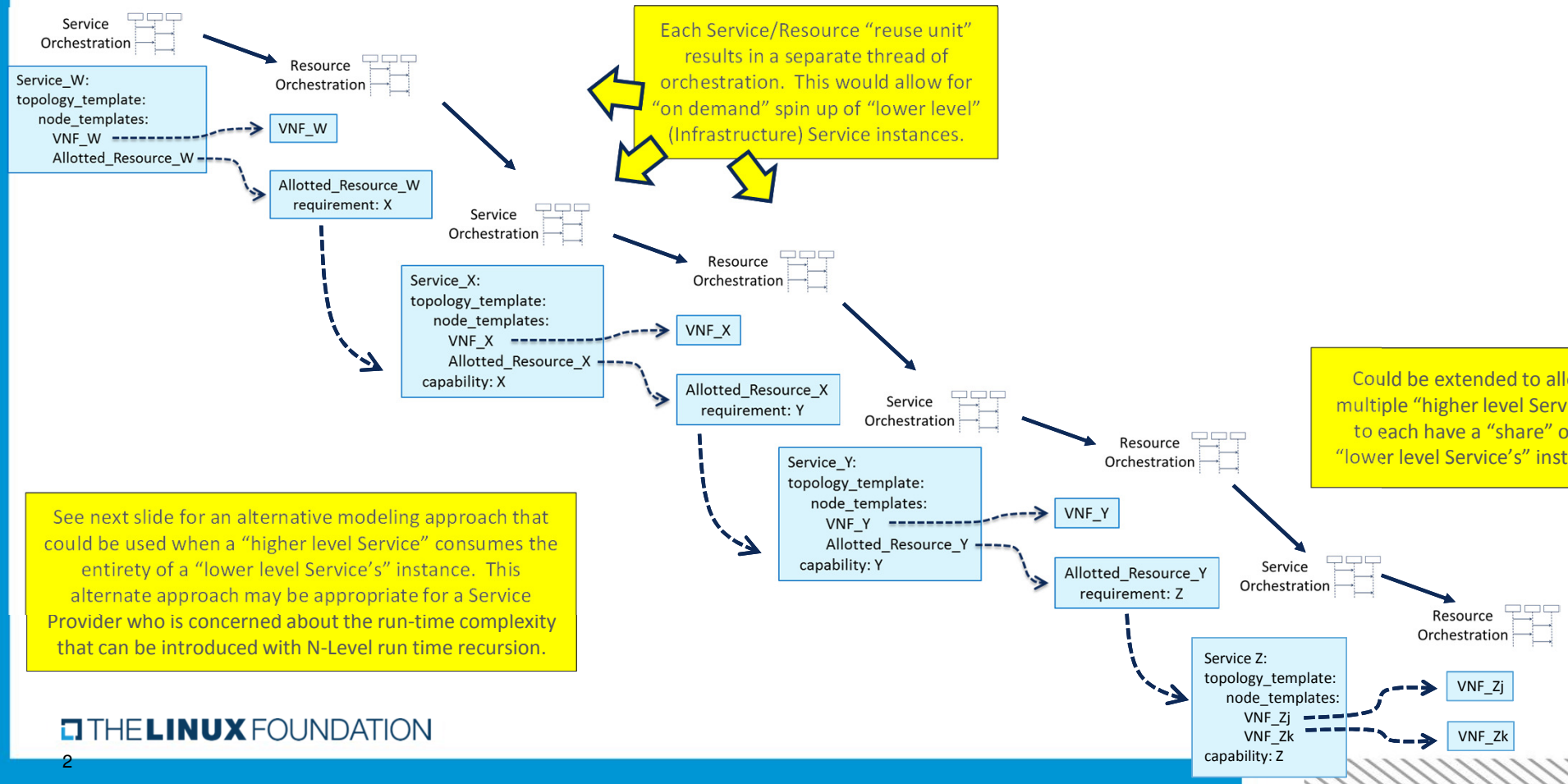
Gil Bullard, AT&T

February 5, 2018

DRAFT FOR DISCUSSION

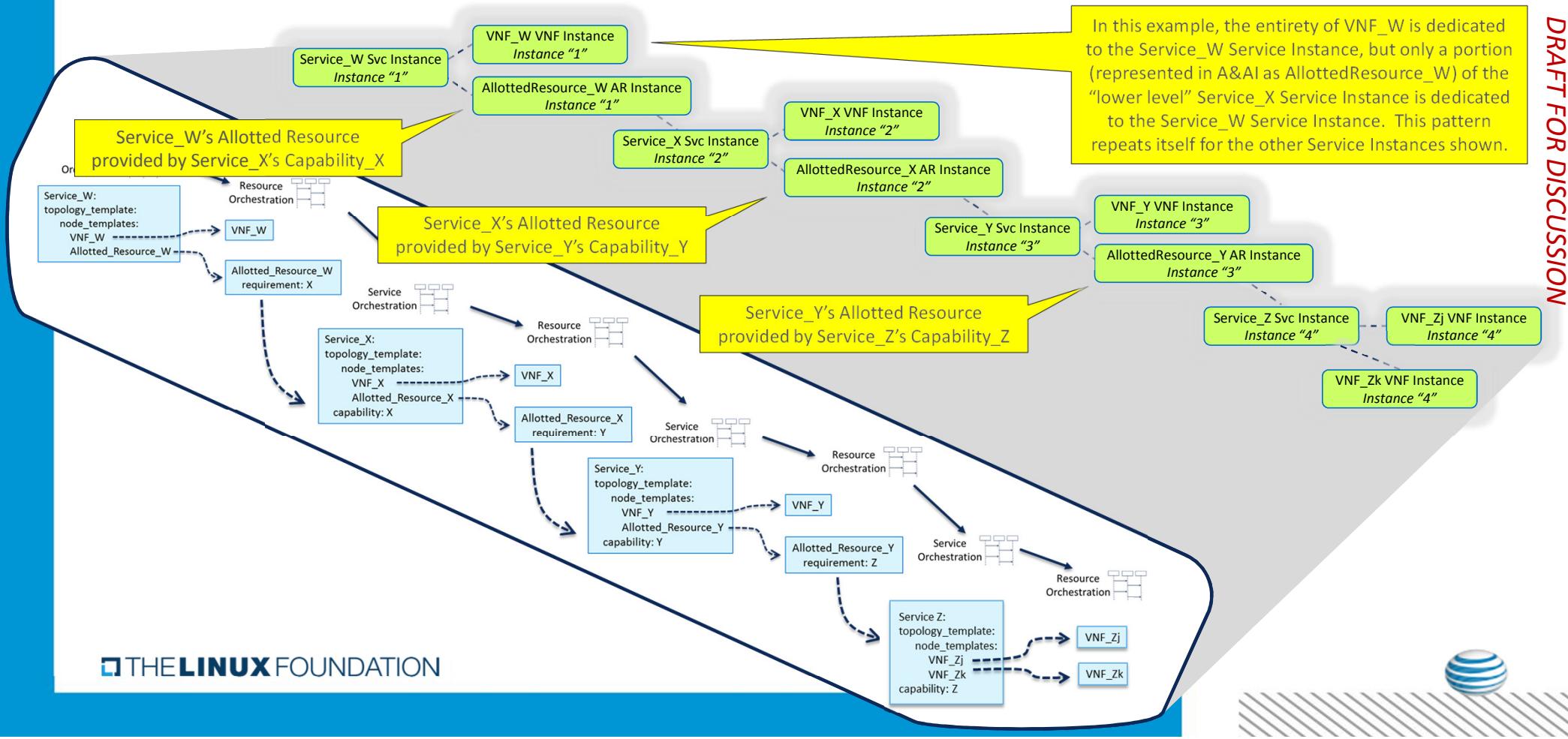
N-Level Run Time Nesting? Let The Service Providers Decide

Service_W Modeling Example 1



DRAFT FOR DISCUSSION

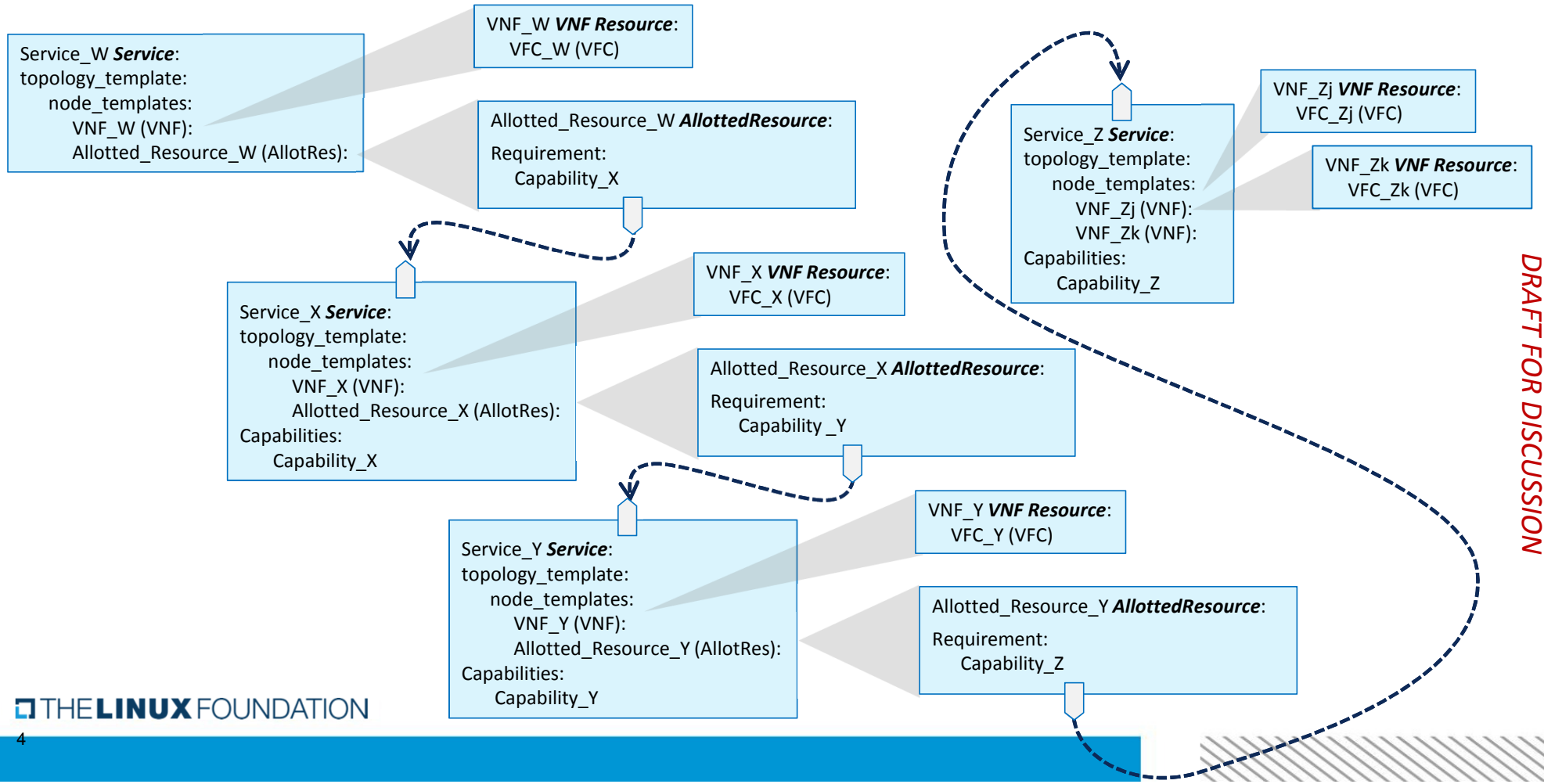
A&AI Instance Representation of Service_W Example 1



DRAFT FOR DISCUSSION

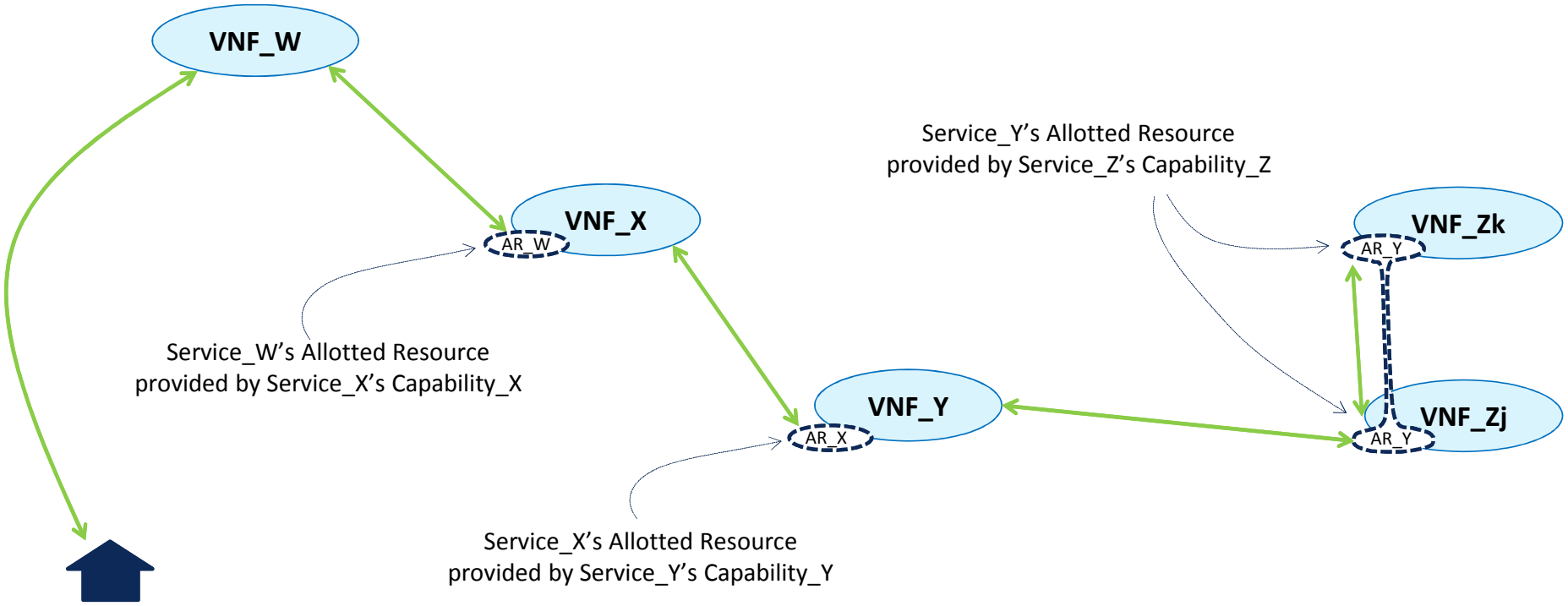
THE LINUX FOUNDATION

Topological Model for Service_W Example 1



DRAFT FOR DISCUSSION

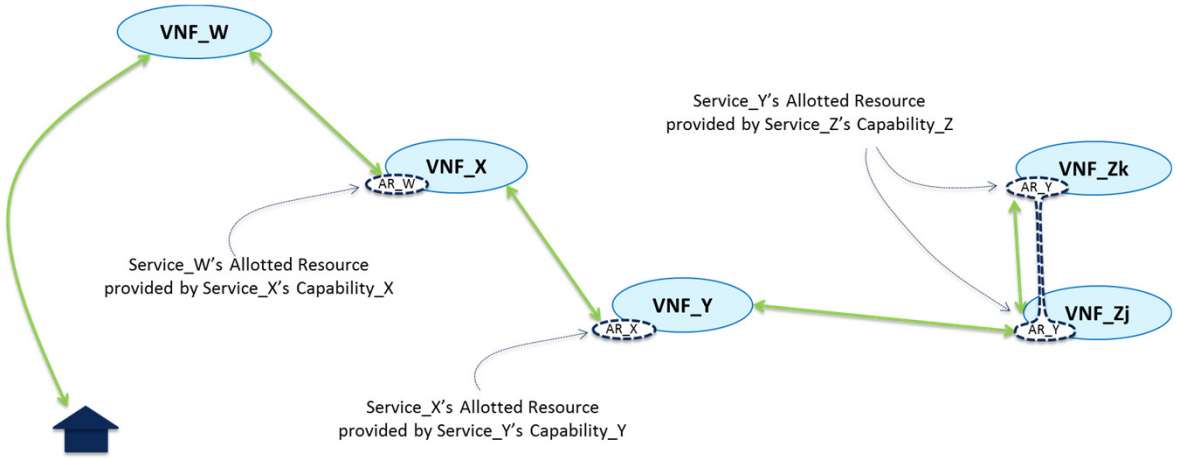
“VNF Chaining” Data Flow for Service_W Example 1



DRAFT FOR DISCUSSION

Modeling Network Latency Homing Constraints for Allotted Resources

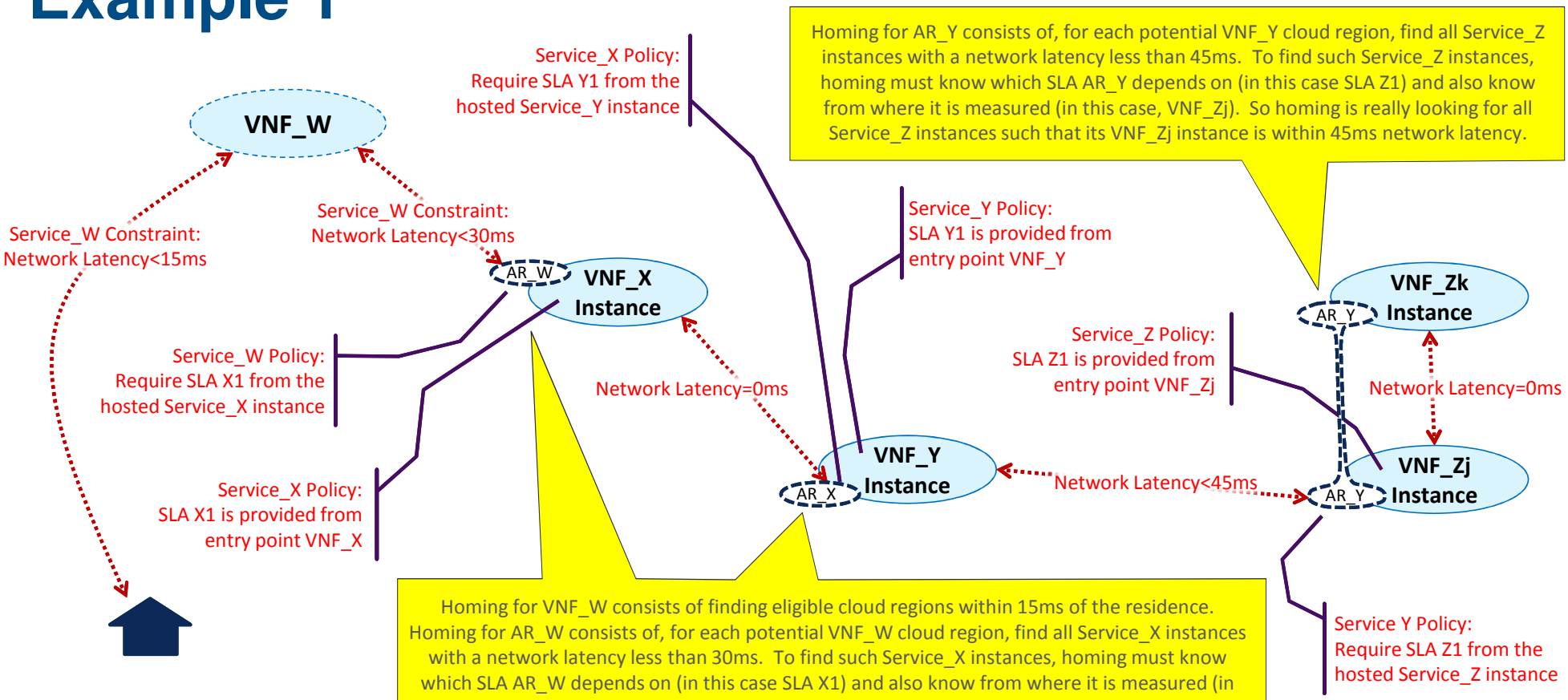
If Service_W is sensitive to network latency beween VNF_W and the VNF_X that hosts AR_W, then the homing algorithm will need to select only VNF_X instances that meet the Service_W constraint. However, we don't want to write any homing (or any other) policies for Service_W in terms of the internal structure of the underlying "lower order" Service type.



We can instead write the network latency constraint in terms of two policies, one a Service_W policy and one a Service_X policy.

Specifically, we will define the concept of an "SLA" that the lower order service will advertise. We will give the "higher order" Service a policy as to which SLA it requires from the "lower order" Service type. We will have the "lower order" Service type have a policy which indicates from which VNF the SLA is measured (mirroring the data path)

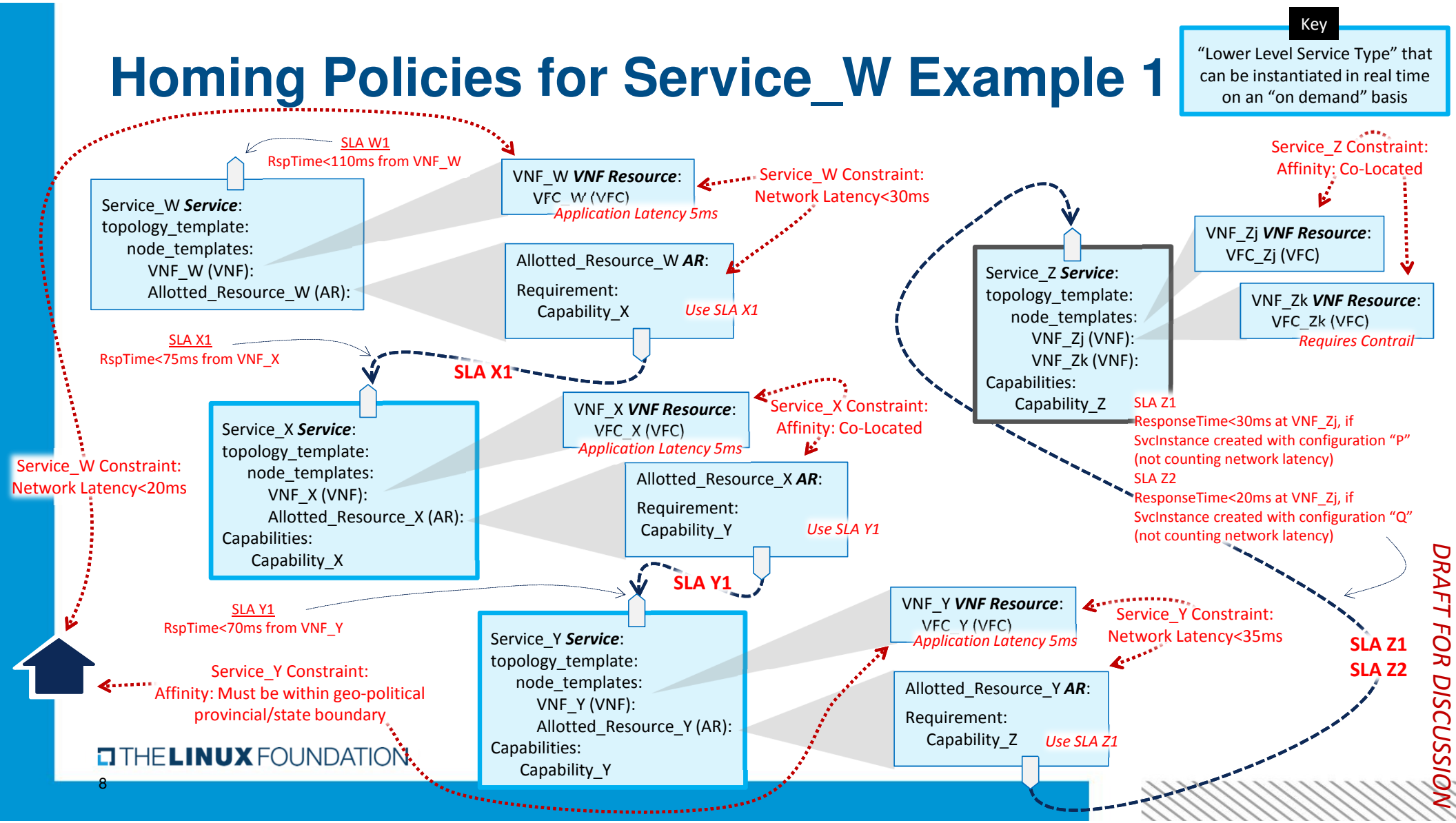
Modeling Network Latency for Service_W Example 1



Homing Policies for Service_W Example 1

Key

"Lower Level Service Type" that can be instantiated in real time on an "on demand" basis



DRAFT FOR DISCUSSION

Decomposition Structure for Service_W Example 1

Svc Type	Rsc Type	AR Capab Svc	SLA Policies	Homing Constraints	Capab Svc Struct
Service_W			W1: RspTime<80ms from VNF_W	Ntw Latency: VNF_W <-> AR_W < 30ms	
Service_W	VNF_W			Ntw Latency:Residence <-> VNF_X < 15ms	
Service_W	AR_W	Service_X		Require SLA X1 from Service_X instance	

AR_W Capab Svc Struct

Svc Type	Rsc Type	AR Capab Svc	SLA Policies	Homing Constraints	Capab Svc Struct
Service_X			X1: RspTime<45ms from VNF_Y	Affinity: VNF_X, AR_X Co-Located	
Service_X	VNF_X				
Service_X	AR_X	Service_Y	Require SLA Y1 from Service_Y instance		

AR_X Capab Svc Struct

Svc Type	Rsc Type	AR Capab Svc	SLA Policies	Homing Constraints	Capab Svc Struct
Service_Y			Y1: RspTime<40ms from VNF_Y	Ntw Latency: VNF_Y <-> AR_Y < 45ms	
Service_Y	VNF_Y			Affinity: Residence, VNF_Y within state boundary {CA, OR, MA, RI, NH}	
Service_Y	AR_Y	Service_Z	Require SLA Z1 from Service_Z instance		

AR_Y Capab Svc Struct

Svc Type	Rsc Type	AR Capab Svc	SLA Policies	Homing Constraints	Capab Svc Struct
Service_Z			Z1: <30ms with config "Q" Z2: <20ms with config "P"	Affinity: VNF_Zj, VNF_Zk Co-Located	
Service_Z	VNF_Zj				
Service_Z	VNF_Zk				

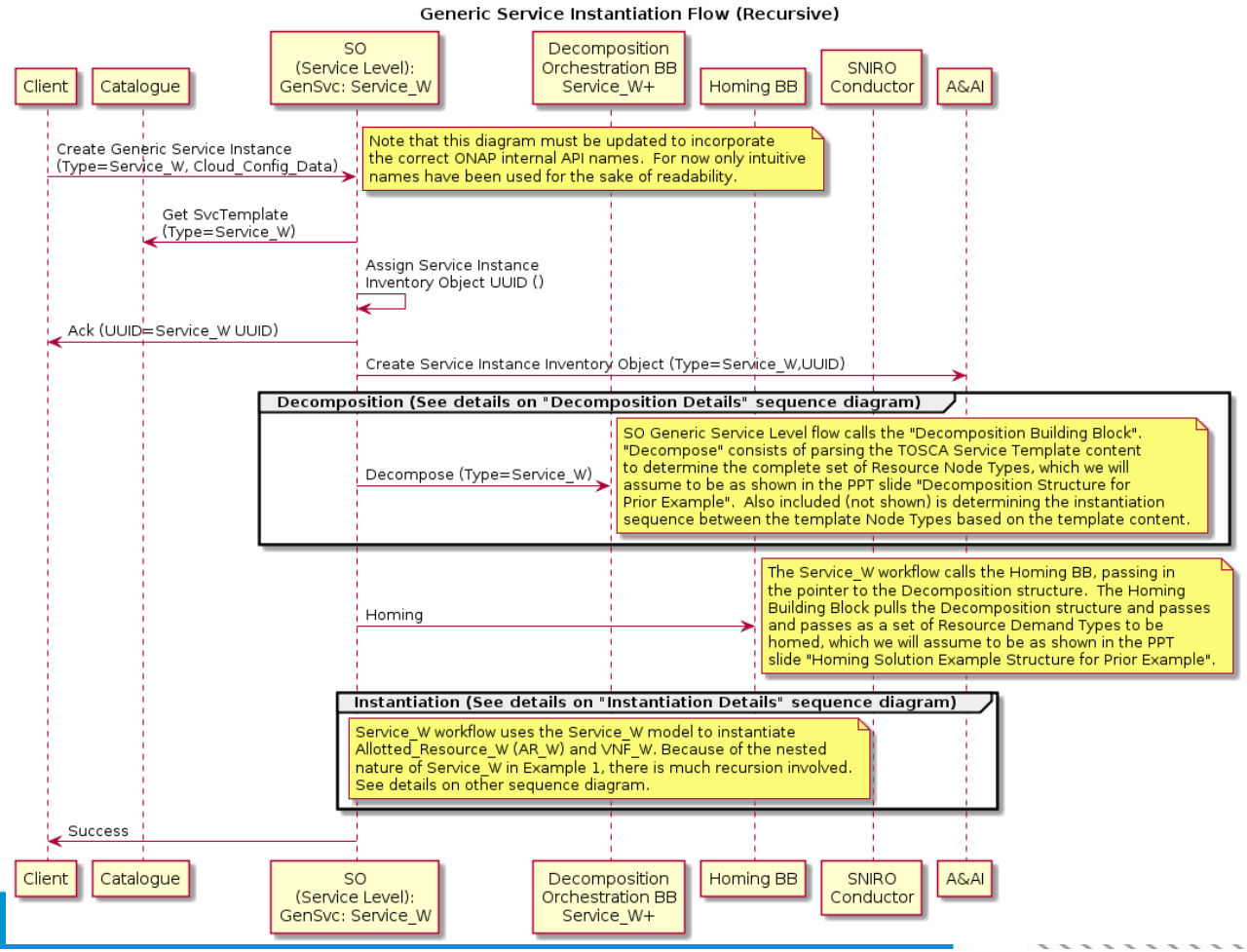
Decomposition and Homing Approach

Note that, from a Service_W perspective, the goal of homing is to find a Service_X instance which meets the Service_W "Ntw Latency" constraint of "VNF_W <-> AR_W < 30ms". This would require decomposition to create the Service_W rows in the decomposition example. If such service instance is found, then homing is complete. However, if no such Service_X instance exists, homing can determine that a new one should be created "on demand."

Creation of a new Service_X instance would require decomposition of Service_X (i.e., the Service_X rows only) for a second homing attempt. From the Service_X perspective, the goal of homing is to find a Service_Y instance which meets the Service_X "Affinity" constraint that "VNF_X, AR_X Co-Located" and such that the "Ntw Latency" constraint of "VNF_W <-> AR_W < 30ms" is also met. (Note that the network latency of AR_W is measured from the Capability_X SLA, which is in turn measured from VNF_X.) Thus, in order to solve the Service_X homing problem, consideration must be given to the Service_W constraints. If homing finds no such Service_Y instance, it can determine that a new one should be created "on demand."

From this point the recursion pattern is set: for nested Services such that the "lower level" Services can be instantiated "on demand", it is necessary to solve the homing problem holistically. Thus, we will opt in the subsequent slides for SO to do a full decomposition prior to a single homing attempt.

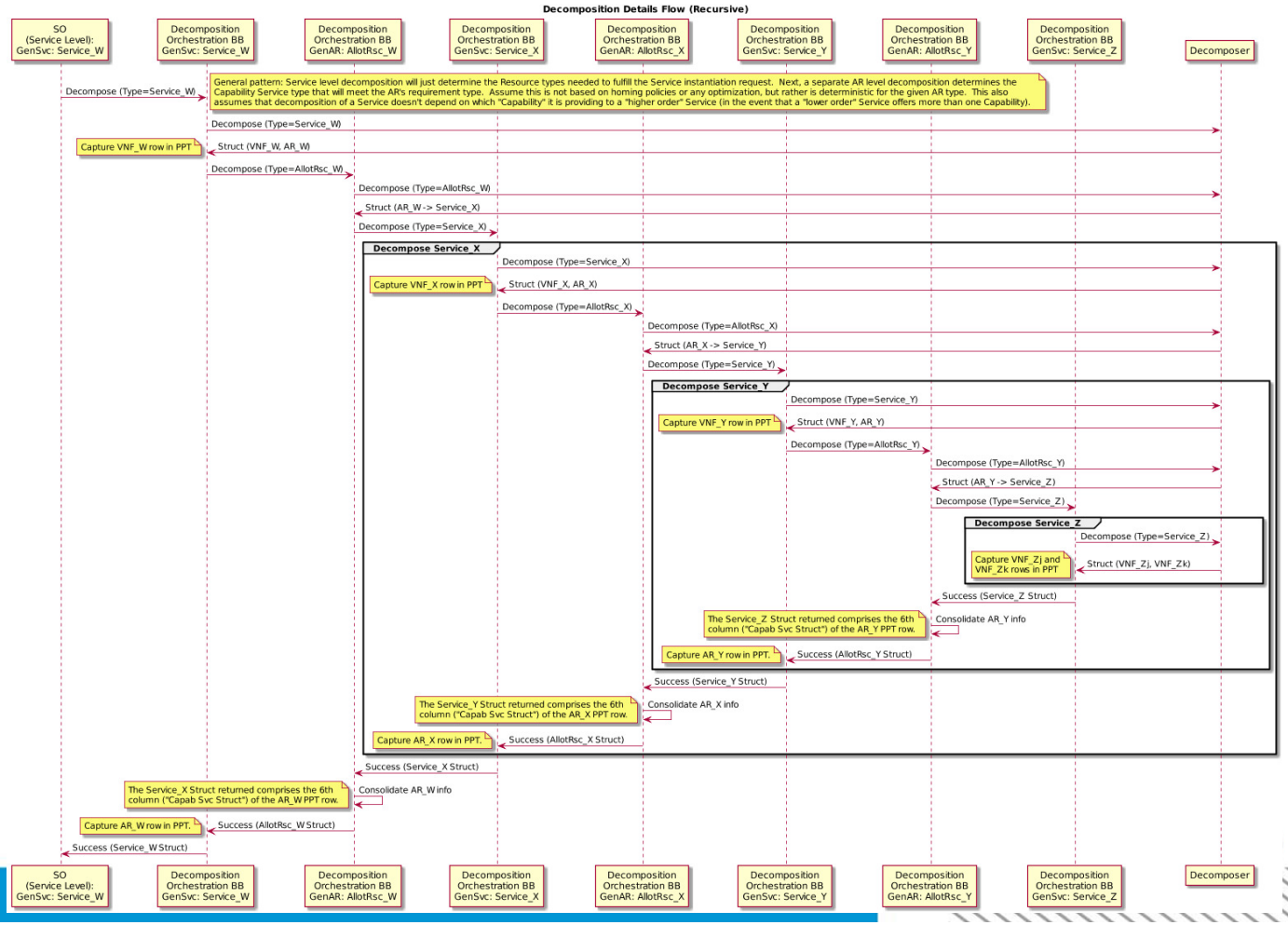
Generic Service Level Flow for Service_W Example 1



onap_uc_Generic_Service_Recursive_p1.html

DRAFT FOR DISCUSSION

Decomposition Detail Flow for Prior Example



onap_uc_Generic_Service_Decomp_p1.html

DRAFT FOR DISCUSSION

Homing Solution Example for Service_W Example 1

Service Type	Resource Type	Allotted Resource Capability Service	Capability Service Struct	Homing Solution
Service_W	VNF_W			Cloud_Region_1
Service_W	Allotted_Resource_W	Service_X		Instantiation_Needed

AR_W Homing Structure

Service Type	Resource Type	Allotted Resource Capability Service	Capability Service Struct	Homing Solution
Service_X	VNF_X			Cloud_Region_2
Service_X	Allotted_Resource_X	Service_Y		Instantiation_Needed

AR_X Homing Structure

Service Type	Resource Type	Allotted Resource Capability Service	Capability Service Struct	Homing Solution
Service_Y	VNF_Y			Cloud_Region_2
Service_Y	Allotted_Resource_Y	Service_Z		Service_Z_Instance_327

AR_Y Homing Structure

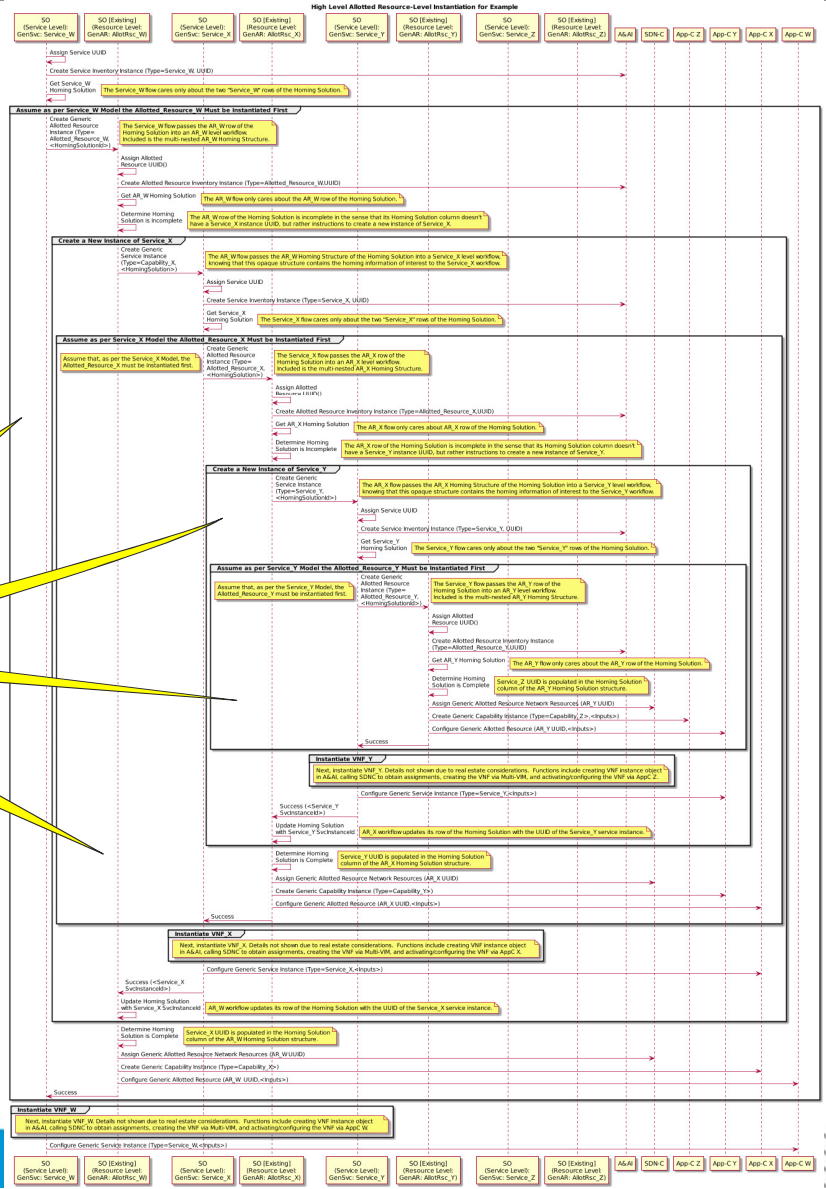
Service Type	Resource Type	Allotted Resource Capability Service	Homing Solution
Service_Z	VNF_Zj		As Exists
Service_Z	VNF_Zk		As Exists

Instantiation Detail Flow for Service_W Example 1

Scale PPT to 300% to view detail. ☺

Note recursion in the process

onap_uc_Generic_Resource_VNF_Recursive.html

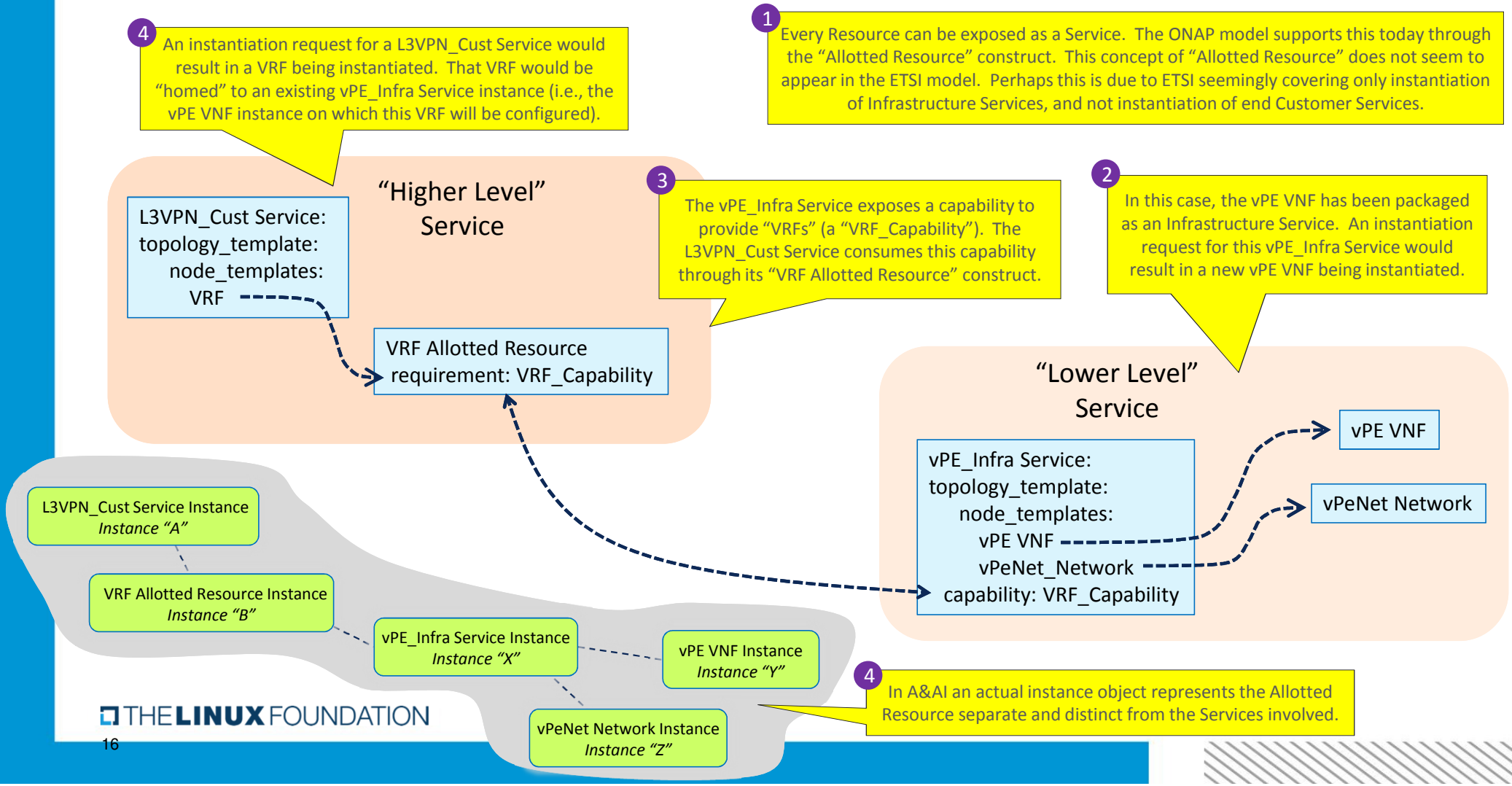


THE LINUX FOUNDATION

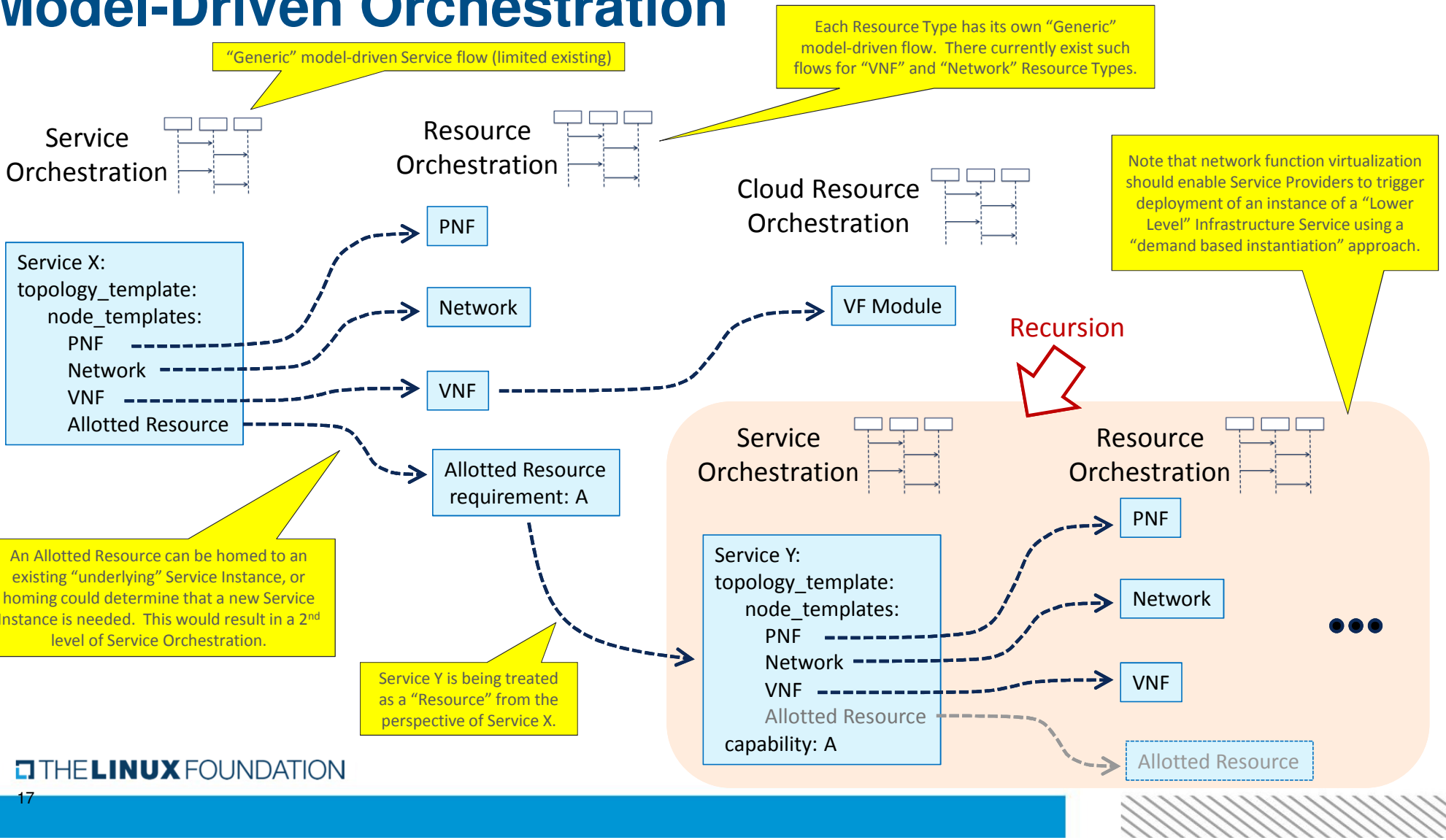
DRAFT FOR DISCUSSION

Backup Slides

Allotted Resources – vPE/VRF Example



Model-Driven Orchestration



N-Level Run Time Nesting? Let The Service Providers Decide

Service_W Modeling Example 2

For the case whereby a "higher level Service" consumes the entirety of a "lower level Service's" instance, SDC should support the Design Time ability to construct an "upper" Service Definition from other Services definitions via substitution mapping (a.k.a., "Compile Time Nesting")

Service W:
topology_template:
node_templates:
VNF_W
Service_X

Service X:
topology_template:
node_templates:
VNF_X
Service_Y

Service Y:
topology_template:
node_templates:
VNF_Y
Service_Z

Service Z:
topology_template:
node_templates:
VNF_Zj
VNF_Zk

Substitution Mapping

Design Time

Distribution Time & Run Time

The "lower level Services" would not be visible at Distribution Time. Hence a "flattening" of the run-time orchestration would result.

Service_W:
topology_template:
node_template:
VNF_W
VNF_X
VNF_Y
VNF_Zj
VNF_Zk

VNF_W
VNF_X
VNF_Y
VNF_Zj
VNF_Zk

A&AI Instance Representation of Service_W Modeling Example 2

Service_W Svc Instance *Instance "1"*

VNF_W VNF Instance *Instance "1"*
VNF_X VNF Instance *Instance "2"*
VNF_Y VNF Instance *Instance "3"*
VNF_Zj VNF Instance *Instance "4"*
VNF_Zk VNF Instance *Instance "4"*

The entirety of these VNFs, not just a portion thereof, are dedicated to the Service_W Service Instance.