13th ANNUAL WORKSHOP 2017

OMNI-PATH FABRIC TOPOLOGIES AND ROUTING

Renae Weber, Software Architect

Intel Corporation

March 30, 2017

LEGAL DISCLAIMERS

AGENDA

- **Omni-Path Routing Algorithms & Constructs**
- **Omni-Path Fabric Topologies**
- **Omni-Path Routing Algorithm Extensions**

OMNI-PATH ROUTING ALGORITHMS

- **Fabric Topologies and Routing**
  - Routing Algorithms supported by Intel® Omni-Path FM
    - Shortest Path
    - Device Group Routing
    - Up/down Fat Tree
    - DOR Mesh/Torus
    - Enhanced Hypercube

ROUTING ALGORITHM CONFIGURATION

```
<!-- ********************* Fabric Routing ***************************** -->
<!-- The following Routing Algorithms are supported                    -->
<!-- shortestpath - pick shortest path and balance lids on ISLs        -->
<!-- dgshortestpath - A variation of shortestpath that uses the        -->
<!--             RoutingOrder parameter to control the order in which  -->
<!--             switch egress ports are assigned to LIDs being routed -->
<!--             through the fabric. This can provide a better balance -->
<!--             of traffic through fabrics with multiple types of end -->
<!--             nodes.                                                -->
<!--             See the <DGShortestPathTopology> section, below, for  -->
<!--             more information.                                     -->
<!-- fattree -   A variation of shortestpath with better balancing     -->
<!--             and improved SM performance on fat tree-like fabrics. -->
<!-- dor -       Dimension ordered routing for use with mesh and torus -->
<!--             topologies cabled in port-to-dimension order.         -->
<!--             See the <MeshTorusTopology> section below for more    -->
<!--             information. (Intel Technical Preview)                -->
<!-- hypercube - A variation of dg shortest path for enhanced          -->
<!--             hypercube and mesh. (Supported via 3rd parties)       -->
<RoutingAlgorithm>shortestpath</RoutingAlgorithm>
```

FABRIC TOPOLOGIES AND ROUTING

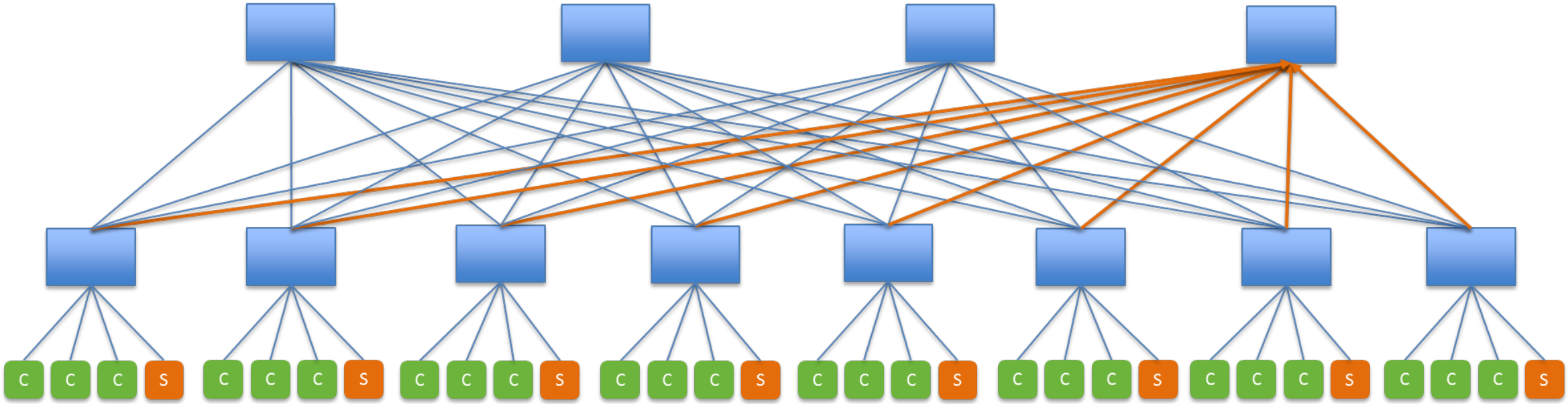
- **Fabric Topologies and Routing**
  - Balance between the following
    - Performance
    - Avoidance of deadlocks due to credit loops
    - Resiliency to fabric disruptions
      - Minimal disruption
    - Fabric bring up time

OMNI-PATH DEVICE GROUP ROUTING

OBLIVIOUS ROUTING

Patterns in physical fabric layout → patterns in traffic routing → congestion

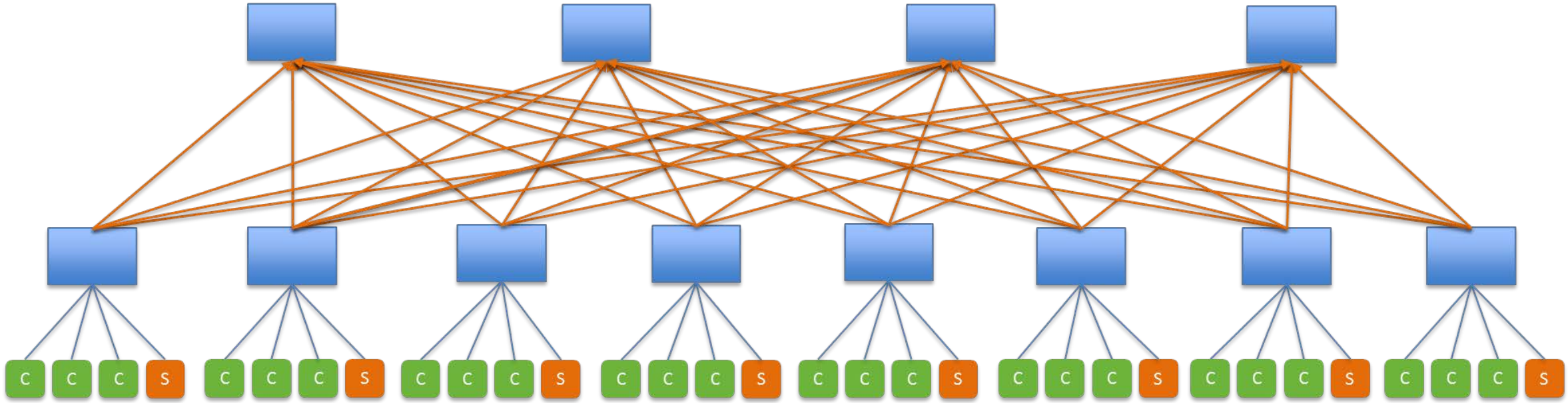
- Routing all Storage traffic through single switch
  - Single point of failure
  - Compromises QoS
    - VL resources per device type
    - All storage using single ISL



DEVICE GROUP ROUTING

Device Group Routing

- Define device groups used to specify routing order (up to 8 groups)
  - Example: compute nodes, storage nodes
- Balance traffic across these groups
- Provides better static load balancing
  - Leverages QoS



DEVICE GROUP ROUTING CONFIGURATION

- Configuring Device Group Routing

```
<DGShortestPathTopology>
    <!-- RoutingOrder lists the device groups in the order they should -->
    <!-- be handled. Each device group must have been declared in the  -->
    <!-- DeviceGroups section.                                         -->
    <RoutingOrder>
      <DeviceGroup>Compute</DeviceGroup>
      <DeviceGroup>Storage</DeviceGroup>
    </RoutingOrder>
</DGShortestPathTopology>
```

DEVICE GROUP CONFIGURATION

- Sample device group constructs

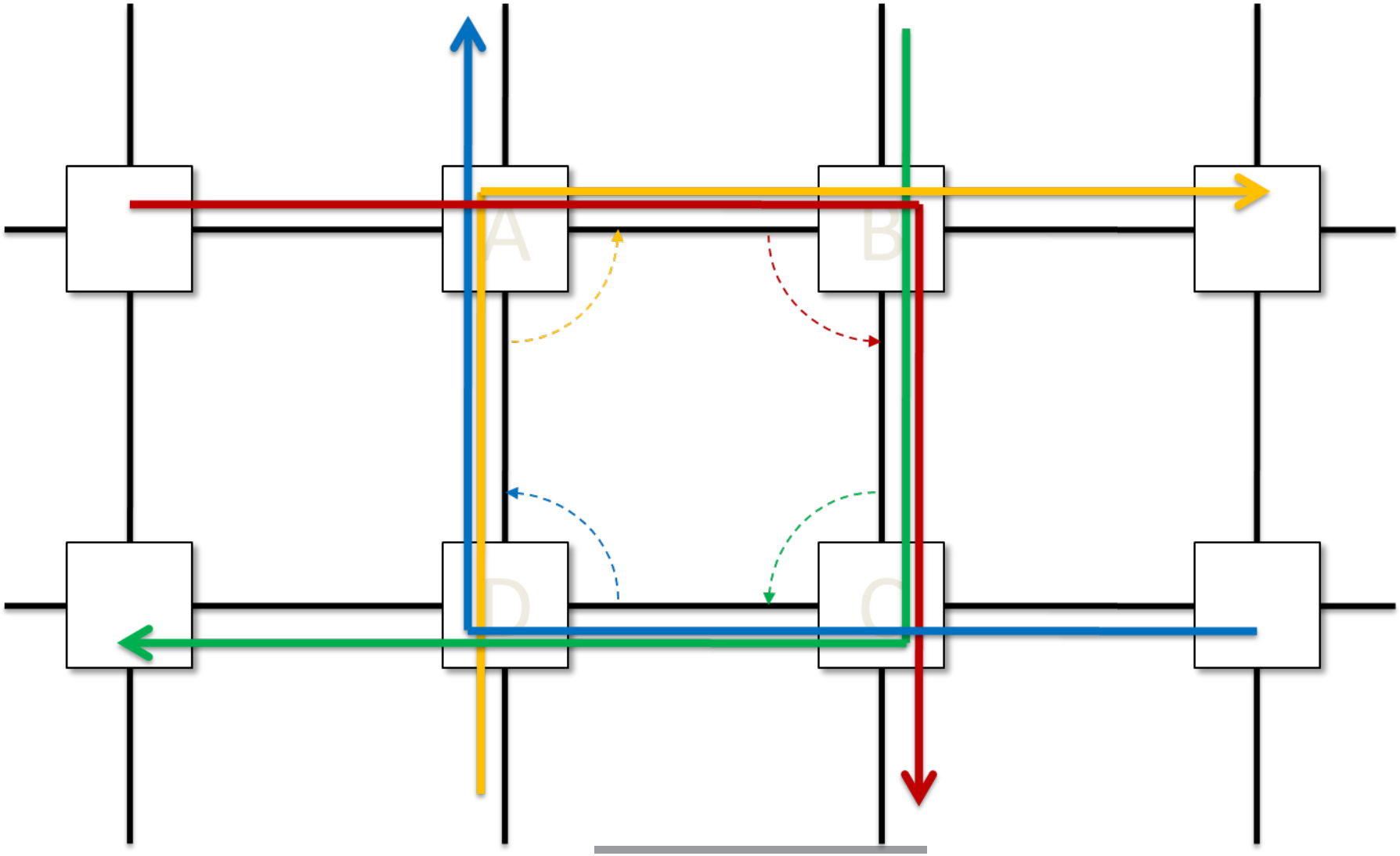
```
<DeviceGroup>
   <Name>SampleDeviceGroup</Name>
   <SystemImageGUID>0x123567812345678</SystemImageGUID>
   <NodeGUID>0x123567812345678</NodeGUID>
   <PortGUID>0x123567812345678</PortGUID>
   <NodeDesc>Some Name</NodeDesc>
   <IncludeGroup>AllEndNodes</IncludeGroup>
</DeviceGroup>
```

- Wildcard on Node Description

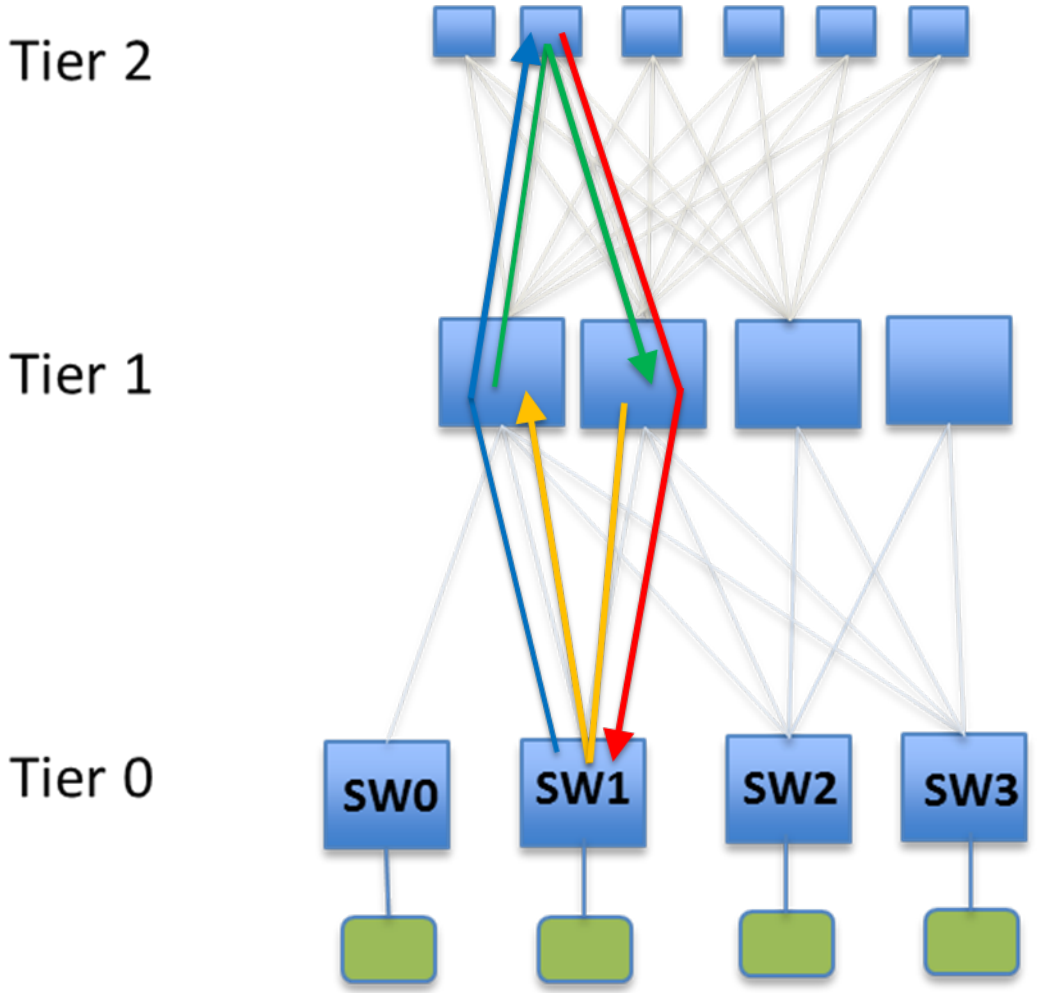
```
<DeviceGroup>
   <Name>Rack1DG</Name>
   <NodeDesc>*Rack1*</NodeDesc>
</DeviceGroup>
```

DEADLOCK AVOIDANCE

ANATOMY OF A CREDIT LOOP

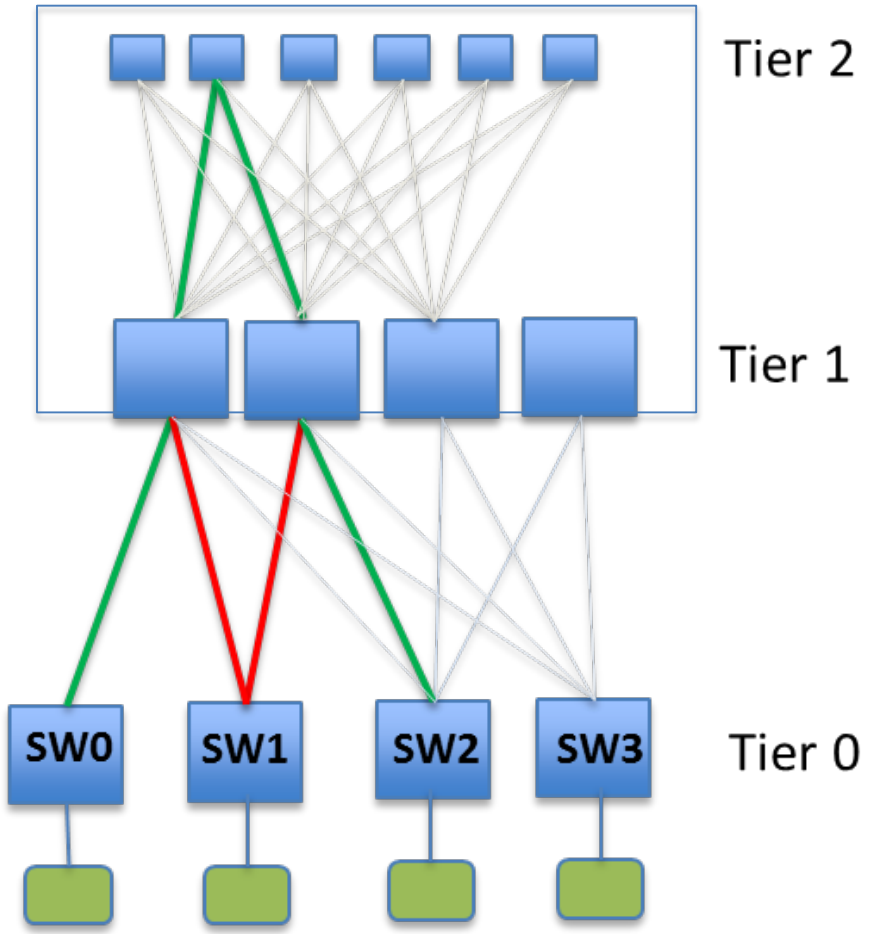


FAT TREE CREDIT LOOP



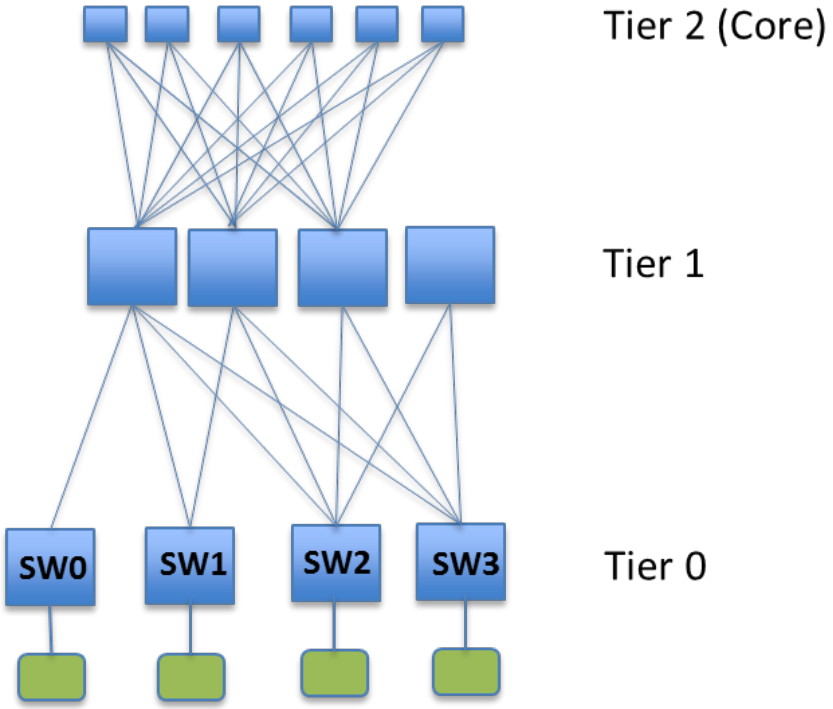
SPINE FIRST ROUTING

- **Credit loop**
  - Due to use of path from SW0 to SW2 using intermediate SW1
- **Spine First Routing**
  - If both up and down routes exist, always route up
    - Deadlock avoidance
  - Director Class Switch
    - System Image GUID is used to determine up link
    - If two equal cost paths exist, select next hop with the same system image GUID



FAT TREE ROUTING

- **Omni-Path Fat Tree Routing Algorithm**
  - Provides up/down routing for deadlock avoidance
    - Extends spine first routing
    - Identifies direction of links in fat tree topologies
      - If both up & down routes exist, chooses up links
  - Discovery walks fat tree to determine
    - Uplink ports
    - Downlink ports
    - Trunk groups
  - Uses topology information calculated during discovery
    - Balances routes across fabric and between each switch in sub-cluster

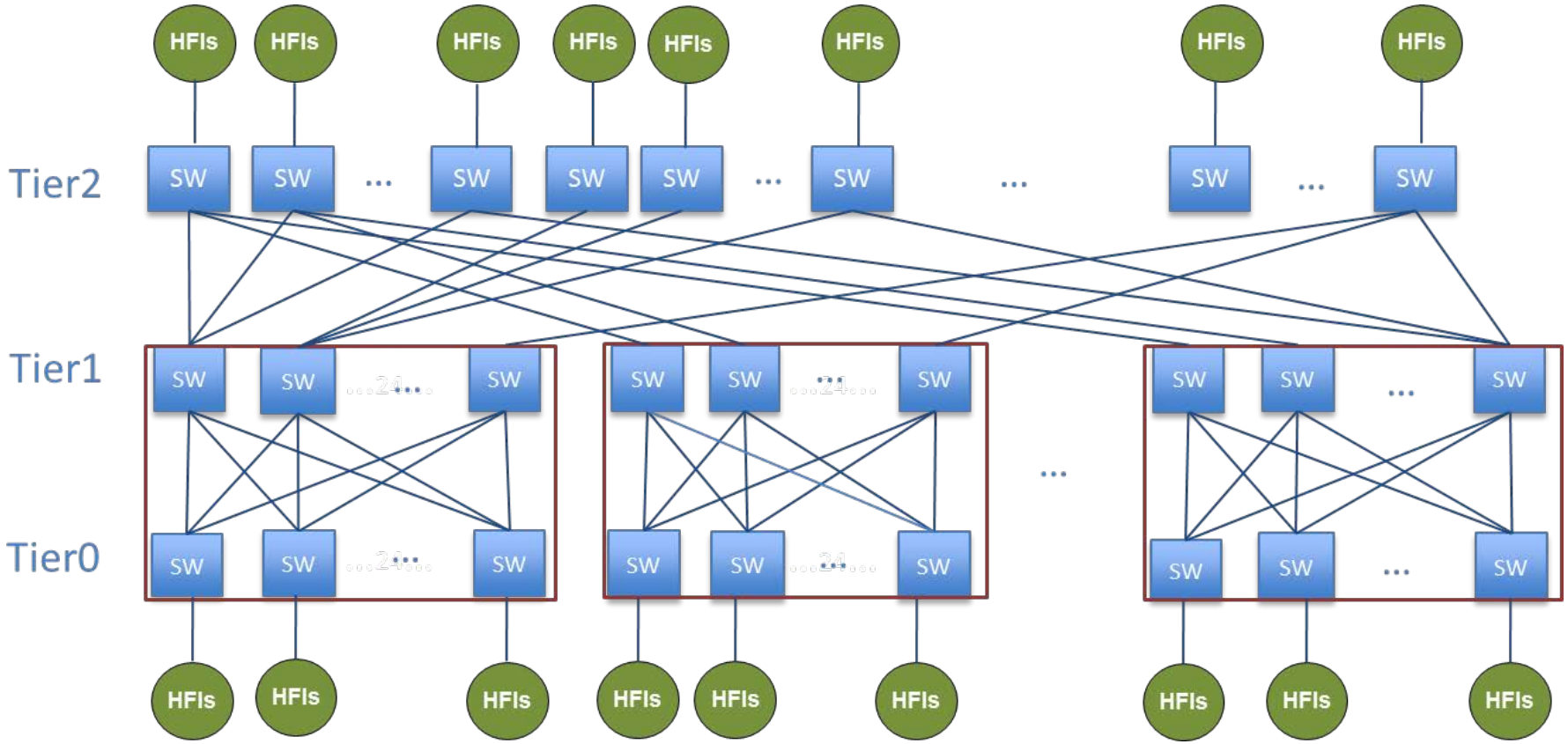


FAT TREE ROUTING CONFIGURATION

Well-formed fat tree, all hosts at lowest tier.

```
<FatTreeTopology>
    <!-- The number of tiers in the fat tree                          -->
    <TierCount>3</TierCount>
    <!-- Indicates whether or not the HFIs are on the same tier in the  -->
    <!-- fat tree.                                                     -->
    <FIsOnSameTier>1</FIsOnSameTier>
    <!-- The following must be setup if the HFIs are not on the same tier. -->
    <!-- It identifies the root/core switches in the fat tree.         -->
    <!-- <CoreSwitches>CoreDeviceGroup</CoreSwitches>                  -->
    <!-- Nodes may be specified for exclusion from initial round-robin  -->
    <!-- to give better route balancing of remaining nodes.            -->
    <!-- This may be useful in asymmetric fat trees or to initially    -->
    <!-- balance across compute nodes in the tree.                     -->
    <!-- <RouteLast>hfi_device_group</RouteLast>                       -->
</FatTreeTopology>
```

NODES AT CORE



FAT TREE ROUTING CONFIGURATION

Asymmetric fat tree, hosts at core or on different tiers

```
<FatTreeTopology>
    <!-- The number of tiers in the fat tree                          -->
    <TierCount>3</TierCount>
    <!-- Indicates whether or not the HFIs are on the same tier in the -->
    <!-- fat tree.                                                    -->
    <FIsOnSameTier>0</FIsOnSameTier>
    <!-- The following must be setup if the HFIs are not on the same tier. -->
    <!-- It identifies the root/core switches in the fat tree.        -->
    <CoreSwitches>CoreDeviceGroup</CoreSwitches>
    <!-- Nodes may be specified for exclusion from initial round-robin -->
    <!-- to give better route balancing of remaining nodes.           -->
    <!-- This may be useful in asymmetric fat trees or to initially   -->
    <!-- balance across compute nodes in the tree.                    -->
    <!-- <RouteLast>hfi_device_group</RouteLast>                      -->
</FatTreeTopology>
```

CORE GROUP DEVICE GROUP CONFIGURATION

Specify core switch group:

- Wildcard on Node Description

```
<DeviceGroup>
  <Name>CoreSwitchGroup</Name>
  <NodeDesc>*Core*</NodeDesc>
</DeviceGroup>
```

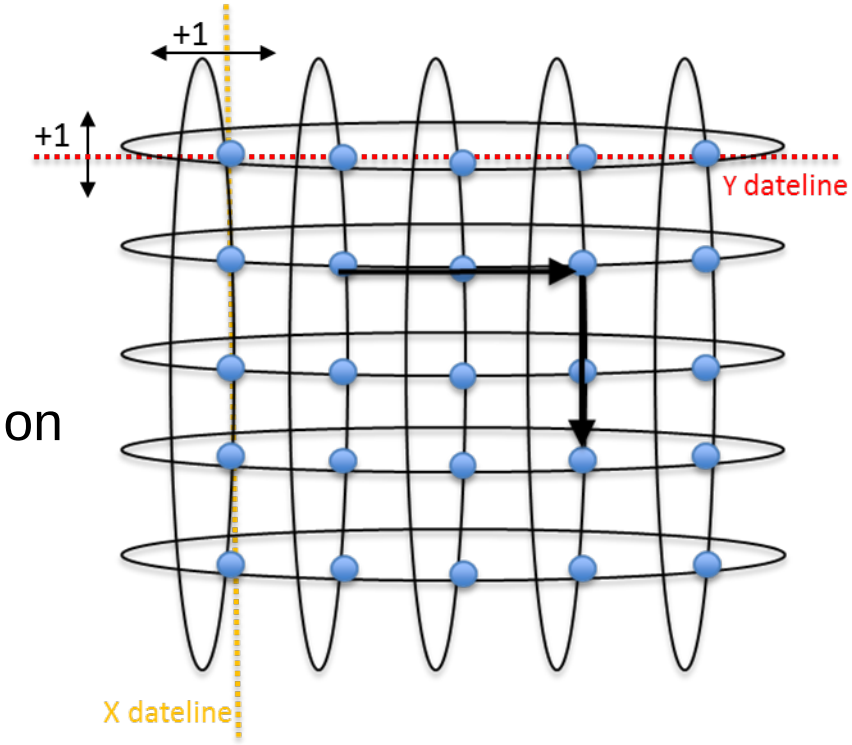
- Explicit NodeGUID

```
<DeviceGroup>
  <Name>CoreSwitchGroup</Name>
  <NodeGUID>0x123567812345678</NodeGUID>
        …
  <NodeGUID>0x123567887654321</NodeGUID>
</DeviceGroup>
```

TORUS TOPOLOGIES

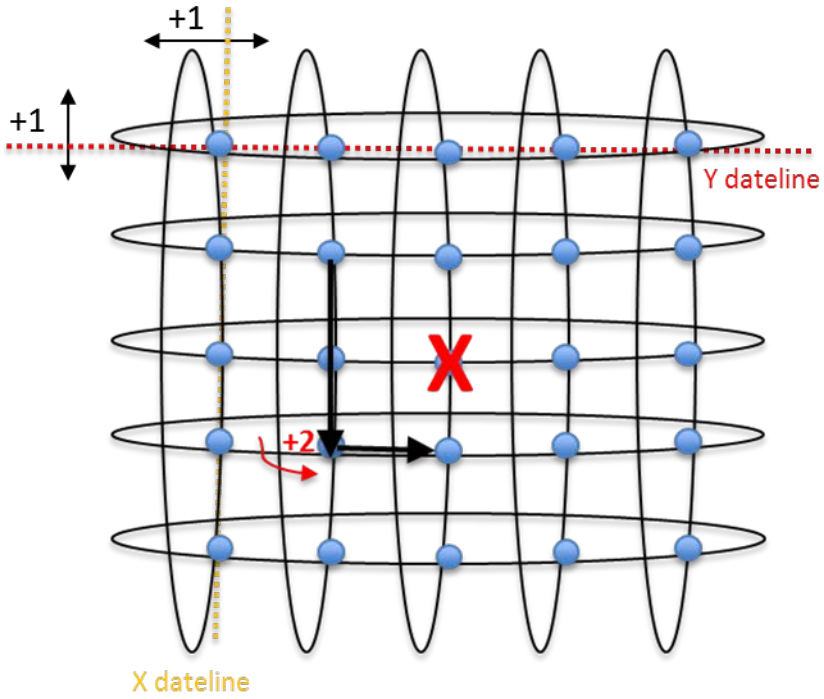
DOR ROUTING FOR TORUS TOPOLOGIES

- **Dimension Ordered Routing with Dateline**
  - Routing order is by dimension
  - Dateline
    - Used for torus deadlock freedom
      - Credit loop free
    - Increment SC/VL when crossing dateline in given dimension
    - Requires 2 VLs
- **Balanced routing**
  - Balanced by locality (switch to switch) and across fabric



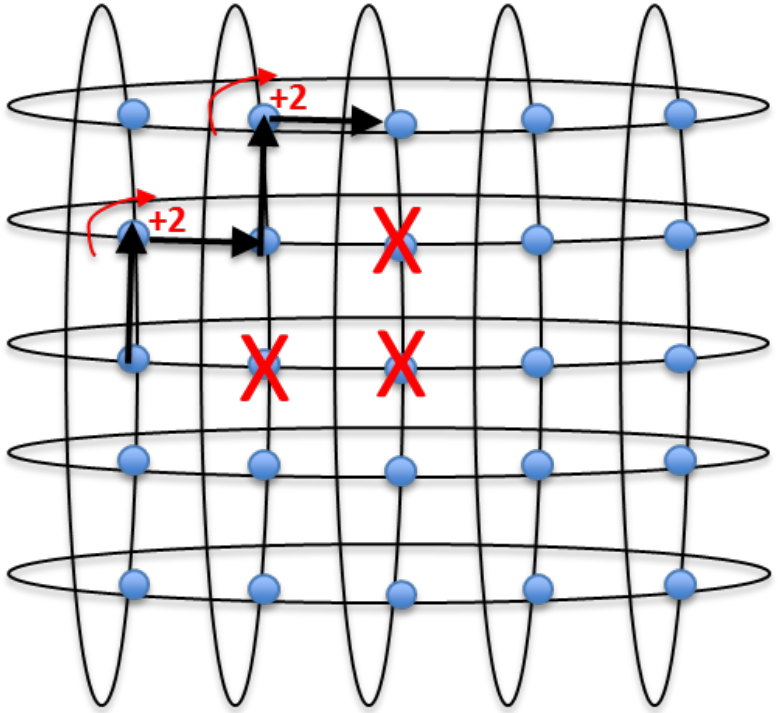
DOR ROUTING DISRUPTION HANDLING

- **Disruption Handling**
  - Additional set of VLs used for disruption handling
    - Escape VLs
      - Jump to new set on illegal turn to lower dimension
      - Stays on new set for remainder of path
  - Accomplished via SC2SC mapping
    - On an illegal turn: SC' -> SC_{i+2}
    - *No SL or path record changes required*



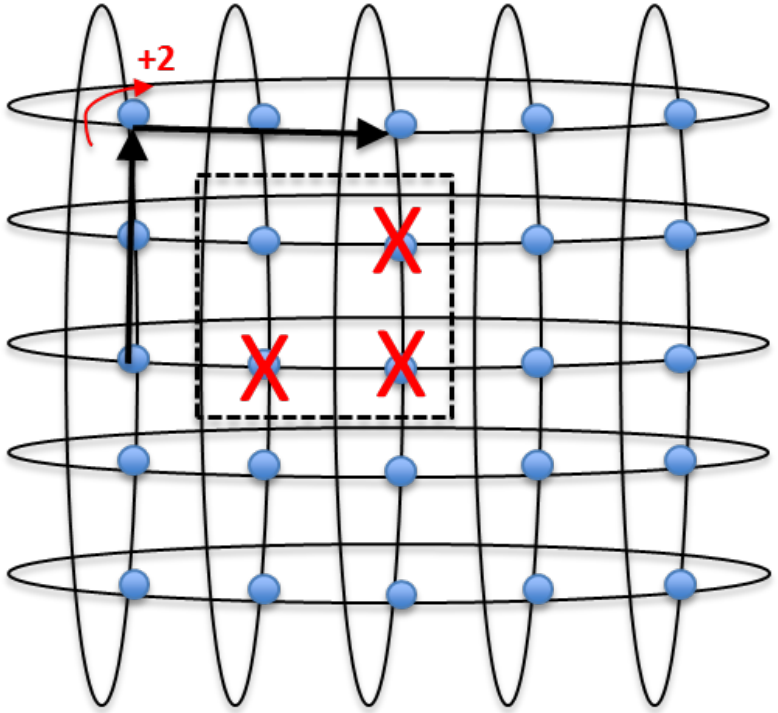
DOR ROUTING HANDLING MULTIPLE FAILURES

- **Disruption Handling**
  - Routing close to failure causes multiple bad turns
    - Would require another set of VLs
      - Credit loop avoidance



FAULT REGION

- **Fault Region**
  - Avoids multiple bad turns
    - Credit loop avoidance
  - Switch within a fault region
    - Route to them
    - Route legal DOR paths through them
    - Don’t route through on broken path
      - Avoid multiple illegal turns



OMNI-PATH FAST FABRIC TOOLS

- **opareport –o validatevlcreditloops**

```
Getting All Node Records...
Done Getting All Node Records
Done Getting All Link Records
Done Getting All Cable Info Records
Done Getting All SM Info Records
Done Getting vFabric Records
Getting All FDB Tables...
Done Getting All FDB Tables
Getting All Port VL Tables...
Done Getting All Port VL Tables
Validate Credit Loop Routes
Done Building All Routes
Fabric summary: 459 devices, 408 HFIs, 51 switches,
        14128 connections, 186864 routing decisions,
        166056 analyzed routes, 0 incomplete routes
Done Building Graphical Layout of All Routes
Routes are deadlock free (No credit loops detected)
Done Deallocating All Vertices
```

ENHANCED HYPERCUBE

- **Enhanced Hypercube**
  - Hypercube and mesh routing
  - Configure port order and cost to control dimension order routing
  - Balanced by locality (switch to switch) and across fabric

```
<HypercubeTopology>
  <EnhancedRoutingCtrl>
    <Switches>SwitchDeviceGroup1</Switches
    <PortData>
      <pPort>25</pPort>
      <vPort>1</vPort>
      <Cost>10</Cost>
    </PortData>
    <PortData>
      <pPort>28</pPort>
      <vPort>2</vPort>
      <Cost>11</Cost>
    </PortData>
  </EnhancedRoutingCtrl>
</HypercubeTopology>
```

ADAPTIVE ROUTING

- **Adaptive Routing**
  - Supported by all Omni-Path Routing Algorithms
  - Medium grained
    - Fabric Manager provides alternate paths to switch FW
    - Switch FW makes changes in event of congestion
    - Uses alternate routes to quickly respond to failures
      - Redundant ISL failover at switch

ROUTING MODULES

- **Routing Module Extensions**
  - Used to add new routing algorithms to the Omni-Path FM
  - Function pointers, wrappers, and structures that store behaviors for a specific topology
  - Decoupling of discovery processing, routing, and QoS logic for specific topologies from main line code

topop->routingModule->funcs.<function>

13th ANNUAL WORKSHOP 2017

THANK YOU

Renae Weber

Intel Corporation