

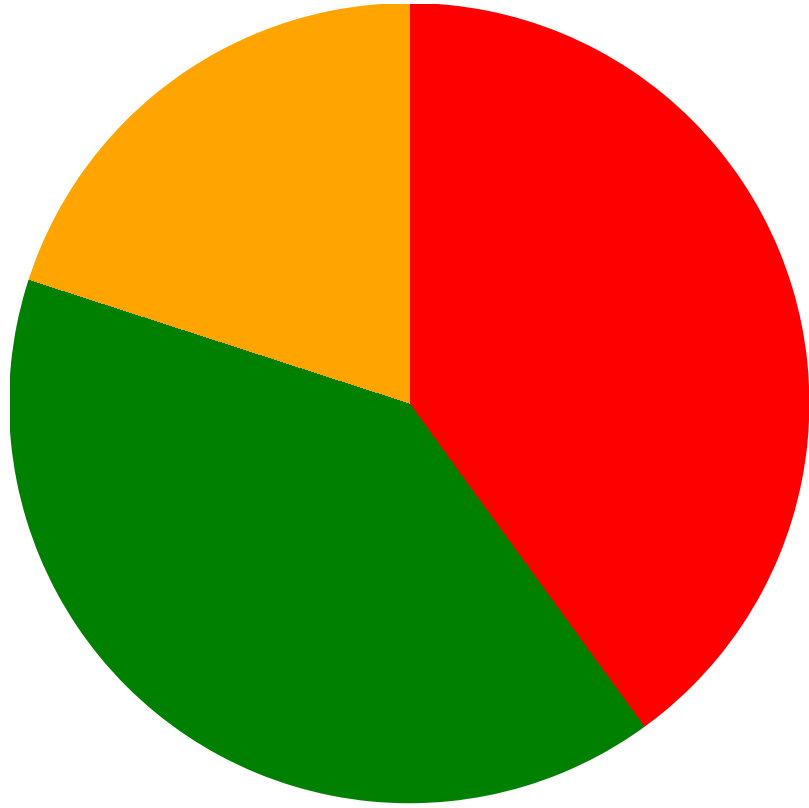
# The 16-bit Datacenter

Brandon Ewing

CHI-NOG 10, Oct 2022

[www.github.com/bewing/chinog-10](https://www.github.com/bewing/chinog-10)

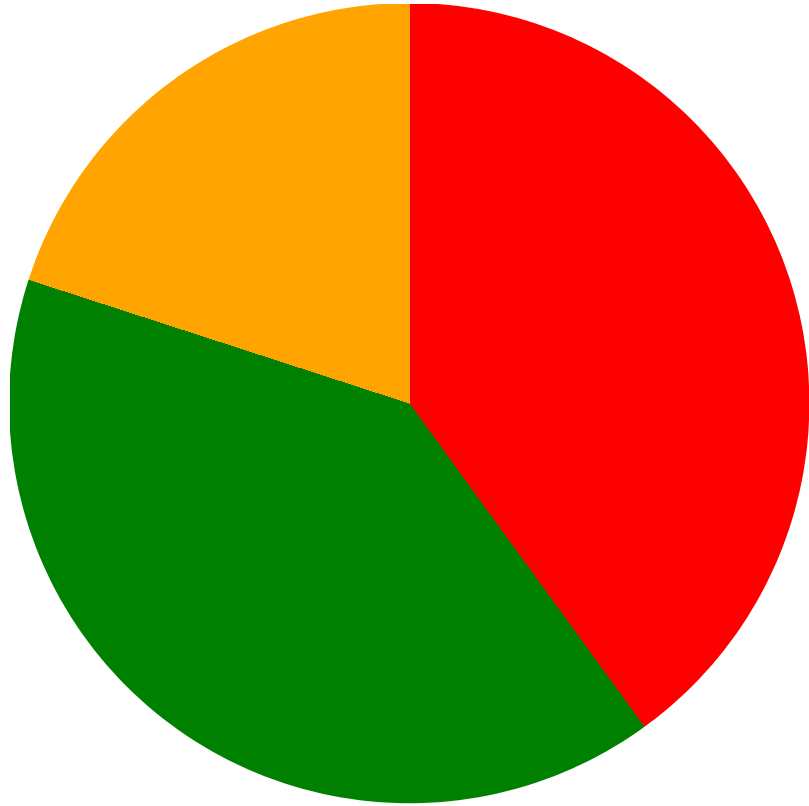
# Talk Contents



40% Knowledge Transfer

40% Stuff to think about

# Talk Contents

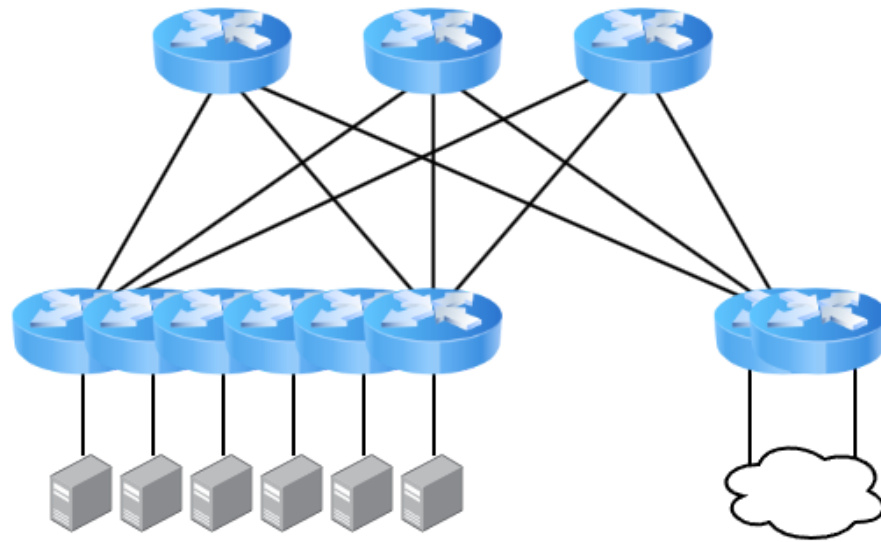



40% Knowledge Transfer

40% Stuff to think about

20% Please tell me he's not  
doing that in prod

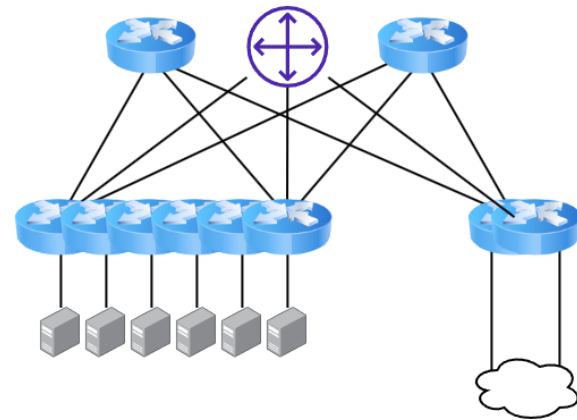
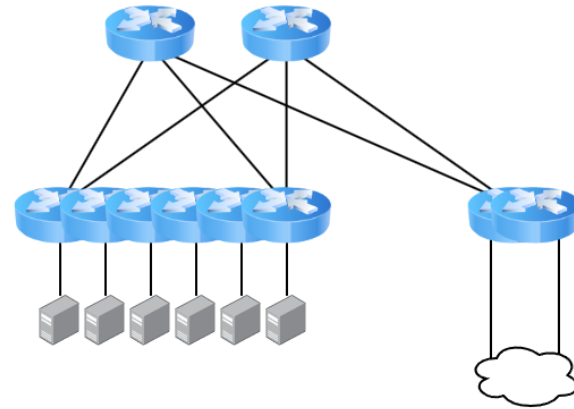
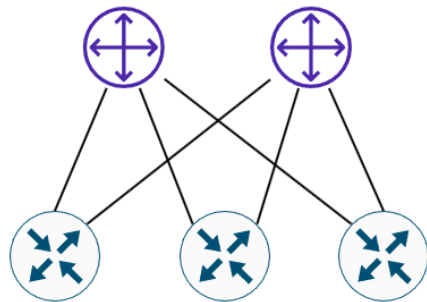
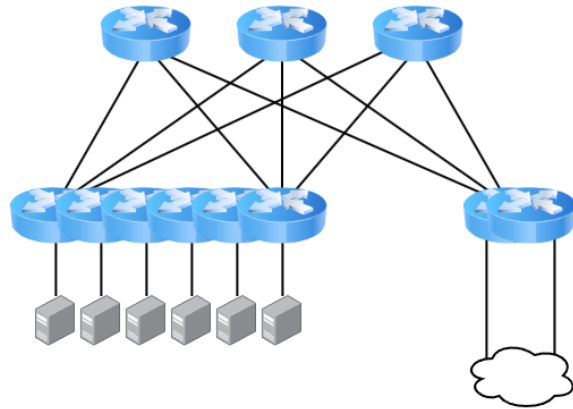
# Simplicity



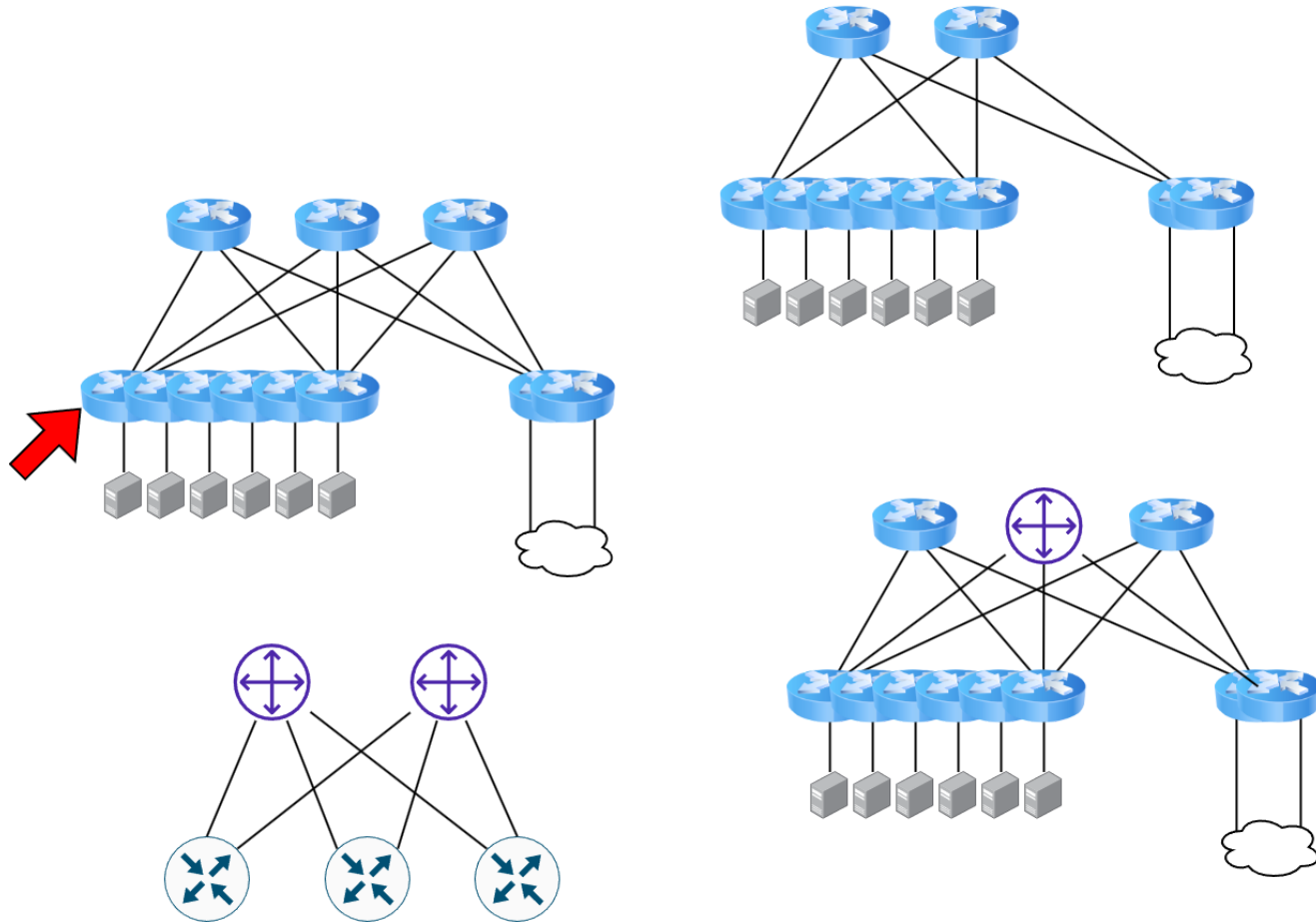
The background is a vibrant green with a pattern of white, irregular lines that resemble a grid or a map. Overlaid on this are several palm trees with dark green fronds and yellow-green trunks. Some of the palm trees have clusters of coconuts hanging from them. The overall style is that of a tropical-themed graphic or poster.

**MANU  
MONTHS LATER**

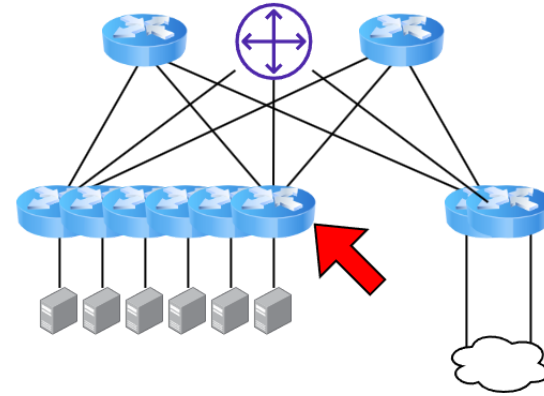
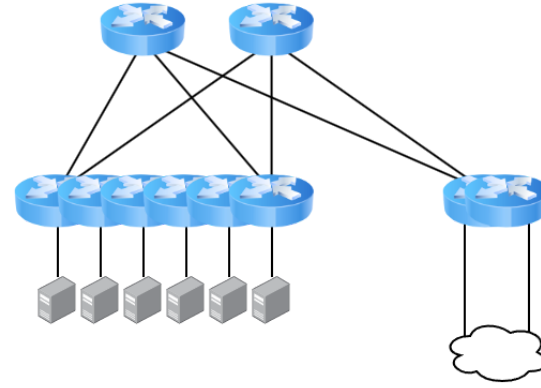
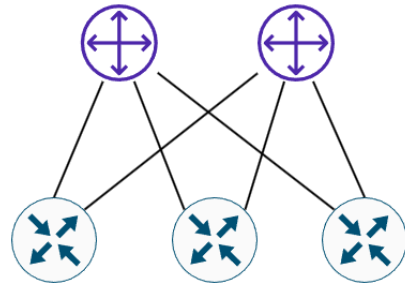
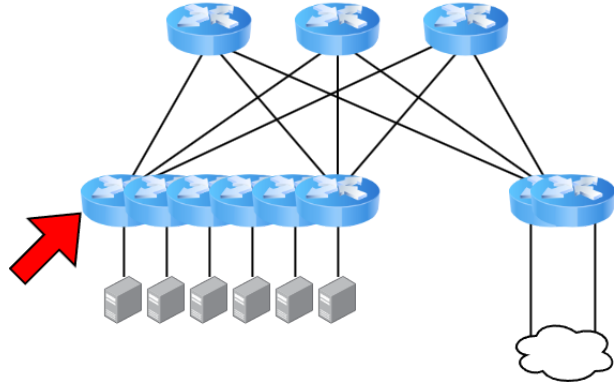
# Complexity



# Complexity

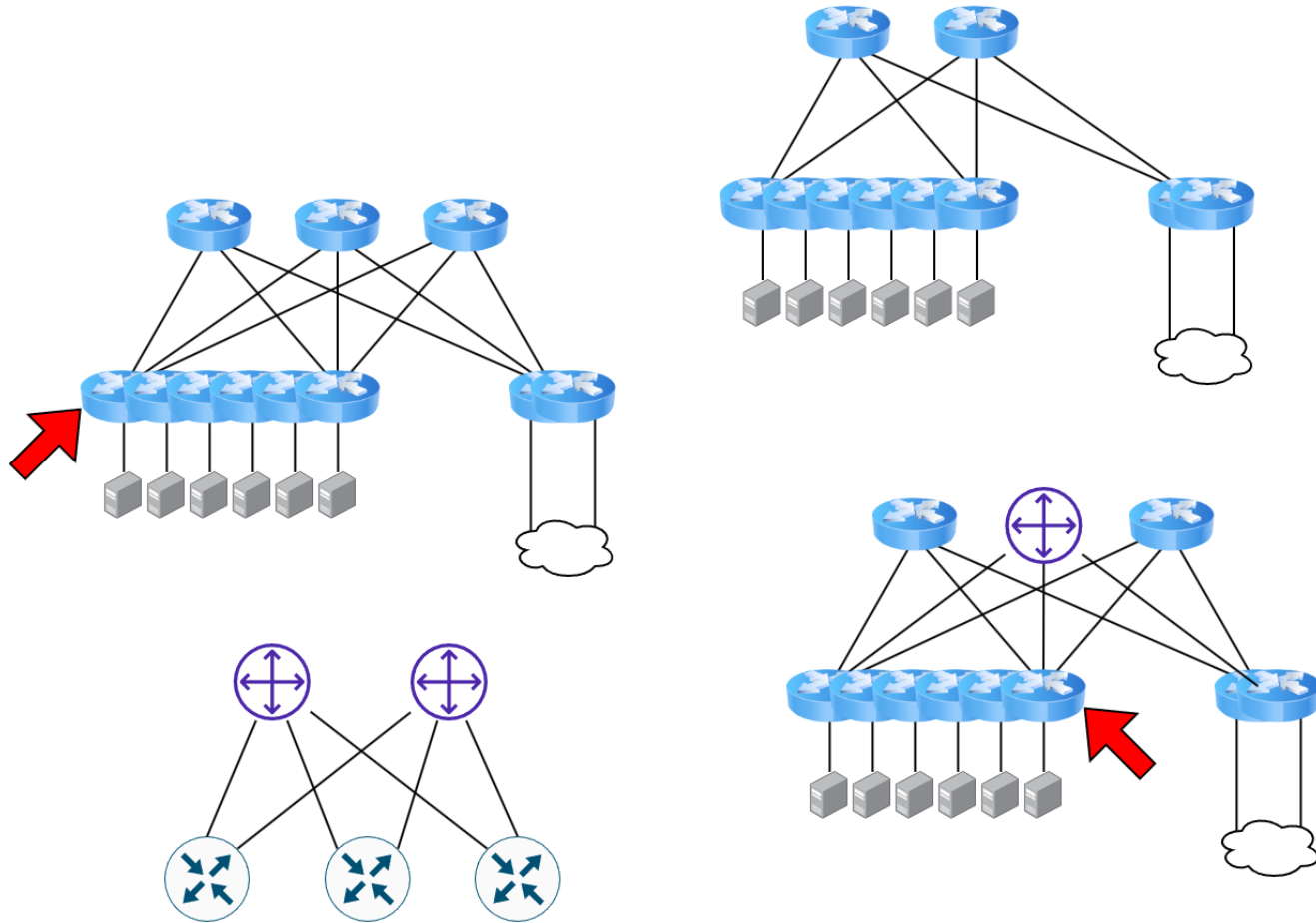


# Complexity





# Complexity



```
$ wc -l configs/DC1-LEAF1
2130 configs/DC1-LEAF1
$ git diff --color --stat --no-index configs/DC1-LEAF1 configs/DC4-LEAF9
configs/{DC1-LEAF1 => DC4-LEAF9} | 881 ++++++++ - - - - -
1 file changed, 362 insertions(+), 519 deletions(-)
```

# Baseline

- Use templates
- Same templates for provisioning and configuration
- If you aren't pushing configs, at least run audits
- Clean up cruft!

# Context

```
nodes:
- hostname: dc1-leaf1
  region: dc1
  role: leaf
  aaa:
  - 192.168.3.2
  - 192.168.10.1
  - 192.168.14.10
  addresses:
    Loopback0:
      ipv4:
      - 172.18.4.10/32
      ipv6:
      - "2001:db8::1/128"
    Ethernet1/1:
      ipv4:
      - 10.0.0.0/31
      ipv6:
      - "2001:2b8:1::1/64"
  bgp:
    router-id: 172.18.4.10
    asn: "65534"
    ipv4:
      peer-groups:
      - name: SPINE-LEAF
        outbound-policy: SPINE-TO-LEAF
        inbound-policy: LEAF-TO-SPINE
      neighbors:
      - address: 10.0.0.1
        asn: "65234"
  dns:
    search: warningg.com
```

```
$ stat --format "%s" host.yaml  
608
```

```
$ stat --format "%s" host.yaml  
608
```

608 bytes > 16 bits

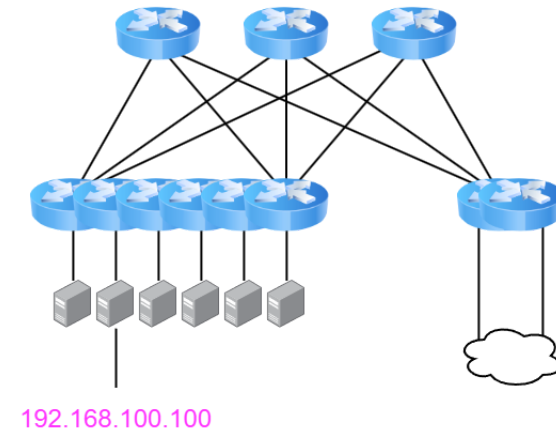
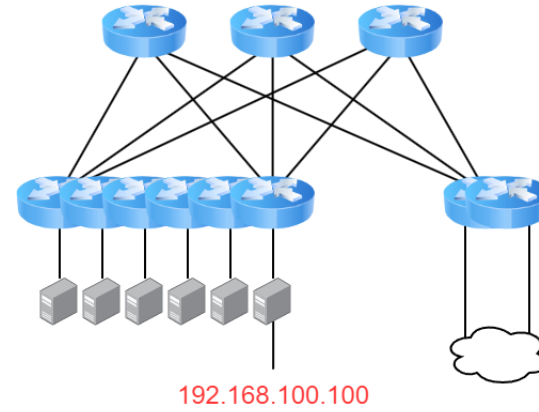
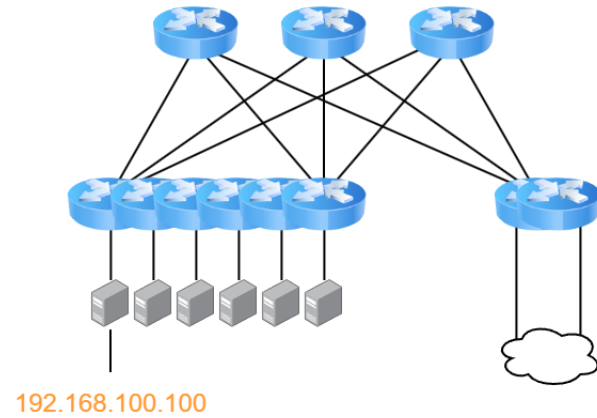
## DC1-LEAF4

## DC4-LEAF9

```
-ip nameserver 192.168.0.0  
-ip nameserver 192.168.100.100  
!  
-tacacs-server 192.168.3.2  
tacacs-server 192.168.10.1  
-tacacs-server 192.168.14.10
```

```
+ip nameserver 10.240.0.0  
+ip nameserver 10.244.100.100  
!  
+tacacs-server 192.168.14.10  
tacacs-server 192.168.10.1  
+tacacs-server 192.168.3.2
```

# Boilerplate



## DC1-LEAF4

## DC4-LEAF9

```
ip nameserver 192.168.100.100
!  
tacacs-server 192.168.3.2  
tacacs-server 192.168.10.1  
tacacs-server 192.168.14.10
```

```
ip nameserver 192.168.100.100
!  
tacacs-server 192.168.3.2  
tacacs-server 192.168.10.1  
tacacs-server 192.168.14.10
```

- Anycast when possible
  - Make sure your application withdraws itself if not healthy
  - Avoid ECMP through policy
- Global services with fallbacks if not
  - If RTT is important, programmatically order them with template logic



# Context

```
nodes:
- hostname: dc1-leaf1
  region: dc1
  role: leaf
  addresses:
    Loopback0:
      ipv4:
        - 172.18.4.10/32
      ipv6:
        - "2001:db8::1/128"
    Ethernet1/1:
      ipv4:
        - 10.0.0.0/31
      ipv6:
        - "2001:2b8:1::1/64"
bgp:
  router-id: 172.18.4.10
  asn: "65534"
  ipv4:
    peer-groups:
      - name: SPINE-LEAF
        outbound-policy: SPINE-TO-LEAF
        inbound-policy: LEAF-TO-SPINE
    neighbors:
      - address: 10.0.0.1
        asn: "65234"
```

# Don't Repeat Yourself

- Don't have the same info twice (IPv4 Lo0, router-id)
- Generate the IPv6 loopback

2001:db8::/64 + 172.18.4.10/32

2001:db8::ac12:40ac

2001:db8::172:18:4:10

2001:db8::172.18.4.10

# Context

```
nodes:
- hostname: dc1-leaf1
  region: dc1
  role: leaf
  router-id: 172.18.4.10
  addresses:
    Ethernet1/1:
      ipv4:
        - 10.0.0.0/31
      ipv6:
        - "2001:2b8:1::1/64"
  bgp:
    asn: "65534"
    ipv4:
      peer-groups:
        - name: SPINE-LEAF
          outbound-policy: SPINE-TO-LEAF
          inbound-policy: LEAF-TO-SPINE
      neighbors:
        - address: 10.0.0.1
          asn: "65234"
```

# Don't Repeat Yourself

This is the 20% part

- Don't have the same info twice (IPv4 Lo0, router-id)
- Generate the IPv6 loopback

172

18

4

10

10101100

00010010

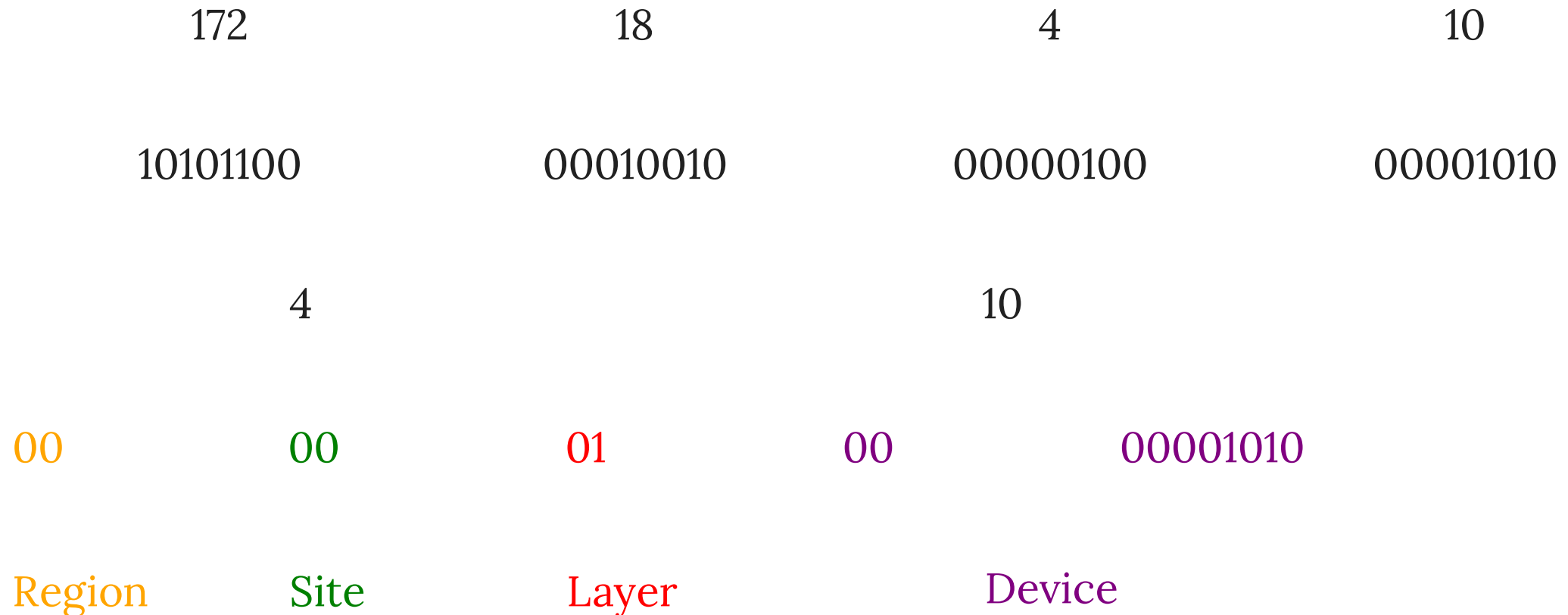
00000100

00001010

# Don't Repeat Yourself

This is the 20% part

- Don't have the same info twice (IPv4 Lo0, router-id)
- Generate the IPv6 loopback



# Don't Repeat Yourself

This is the 20% part

- Don't have the same info twice (IPv4 Lo0, router-id)
- Generate the IPv6 loopback
- Encode information into bit fields
- BGP ASN? Sure!

Region

Site

Layer

Device

65 X X X . X X X XX

<16 bits>.<16 bits>

This is the 20% part

REGION 0 LEAF

172.18.4.10

Region

Site

Layer

Device

00

00

01

00

00001010

65 0 0 1 . (4 \* 256 + 10)

65001.1034

This is the 20% part

REGION 3 SUPERSPINE

172.18.72.75

Region

Site

Layer

Device

11

00

11

00

01001011

65 3 0 3 .(72 \* 256 + 75)

65303.18507



- Why 16 bits?
  - Have to title the talk somehow
  - IPv4 Reachability
  - BGP 4-byte Private ASN space ~25 bits

Please do not do this in prod

# Context

```
nodes:
- router-id: 172.18.4.10
  addresses:
    Ethernet1/1:
      ipv4:
        - 10.0.0.0/31
      ipv6:
        - "2001:2b8:1::1/64"
  bgp:
    ipv4:
      peer-groups:
        - name: SPINE-LEAF
          outbound-policy: SPINE-TO-LEAF
          inbound-policy: LEAF-TO-SPINE
      neighbors:
        - address: 10.0.0.1
          asn: "65234"
```

# Interfaces

## DC1-LEAF4

## DC4-LEAF9

```
interface Ethernet49/1
-  description DC1-SPINE1:Et5/1
  no switchport
-  ip address 10.0.0.0/31
-  ipv6 address 2001:db8::ffff:0a00:0/127
  pim ipv4 sparse-mode
  pim ipv6 sparse-mode
```

```
interface Ethernet 49/1
+  description DC4-SPINE3:Et5/1
  no switchport
+  ip address 10.5.49.22/31
+  ipv6 address 2001:db8::ffff:a05:3116/127
  pim ipv4 sparse-mode
  pim ipv6 sparse-mode
```

```
router bgp 65001.1034
  neighbor SPINES-v4 peer-group
  neighbor SPINES-v6 peer-group
-  neighbor 10.0.0.1 peer-group SPINES-v4
-  neighbor 10.0.0.1 remote-as 65002.3080
-  neighbor 2001:db8:ffff:0a00:1 peer-group SPINES-v6
-  neighbor 2001:db8:ffff:0a00:1 remote-as 65002.3080
  address-family ipv4 unicast
    peer-group SPINES-v4 activate
    no peer-group SPINES-v6 activate
  !
  address-family ipv6 unicast
    no peer-group SPINES-v4 activate
    peer-group SPINES-v6 activate
  !
  !
```

```
router bgp 65031.1077
  neighbor SPINES-v4 peer-group
  neighbor SPINES-v6 peer-group
+  neighbor 10.5.49.23 peer-group SPINES-v4
+  neighbor 10.5.49.23 remote-as 65032.3088
+  neighbor 2001:db8::ffff:a05:3117 peer-group SPINES-v6
+  neighbor 2001:db8::ffff:a05:3117 remote-as 65032.3088
  address-family ipv4 unicast
    peer-group SPINES-v4 activate
    no peer-group SPINES-v6 activate
  !
  address-family ipv6 unicast
    no peer-group SPINES-v4 activate
    peer-group SPINES-v6 activate
  !
  !
```

# Interfaces

- RFC5549 - IPv4 NLRI in IPv6 Peering
- Allows IPv4 reachability over just IPv6 peerings
- No longer need IPv4 BGP peerings
- No longer need IPv4 point to point interfaces!

```
interface Ethernet49/1
  ipv6 address 2001:db8::ffff:0a00:0/127
  pim ipv4 sparse-mode
  pim ipv6 sparse-mode
!
router bgp 65001.1034
  neighbor SPINES peer-group
  neighbor 2001:db8::ffff:0a00:1/127 peer-group SPINES
  neighbor 2001:db8::ffff:0a00:1/127 remote-as 65002.3080
  address-family ipv4 unicast
    bgp next-hop address-family ipv6
    neighbor SPINES activate
    neighbor SPINES next-hop address-family ipv6 originate
  !
  address-family ipv6 unicast
    neighbor SPINES activate
  !
!
```

# But wait, there's more!

- draft-white-linklocal-capability
- Widely supported across vendors
- Standardizes existing practice of BGP peering via link-local IPv6 addresses
- Now we don't need any globally unique addressing!

```
interface Ethernet49/1
  ipv6 address fe80::0/64
  pim ipv4 sparse-mode
  pim ipv6 sparse-mode
!
router bgp 65001.1034
  neighbor SPINES peer-group
  neighbor fe80::1%Ethernet49/1 peer-group SPINES
  neighbor fe80::1%Ethernet49/1 remote-as 65002.3080
  address-family ipv4 unicast
    bgp next-hop address-family ipv6
    neighbor SPINES activate
    neighbor SPINES next-hop address-family ipv6 originate
  !
  address-family ipv6 unicast
    neighbor SPINES activate
  !
!
```

# BGP Peerings

- BGP Peer autodetection
- Multiple IETF IDR WG proposals (see [draft-ietf-idr-bgp-autoconf-considerations](#))
- Some Layer2 (LLDP), some Layer3
- Some are secured, some aren't
- Some are stateful, some are stateless
- No real consensus yet
- Trying to support all DC use cases

# BGP Peerings

- More than one vendor supports IPv6 link-local peer autodetection
- Uses IPv6 RAs to identify routers on an interface
- No current RFC or I-D for this behavior
- There may be interoperability issues

```
interface Ethernet49/1
  ipv6 enable
  pim ipv4 sparse-mode
  pim ipv6 sparse-mode
!
peer-filter SPINES
  10 match 42000000000-4294967294 result accept
!
router bgp 65001.1034
  neighbor SPINES peer-group
  neighbor interface Ethernet49/1 peer-group SPINES peer-filter SPINES
  address-family ipv4 unicast
    bgp next-hop address-family ipv6
    neighbor SPINES activate
    neighbor SPINES next-hop address-family ipv6 originate
  !
  address-family ipv6 unicast
    neighbor SPINES activate
  !
!
```

- Neighborships can specify AS ranges
- Still have to statically map a policy to an interface

## DC1-SPINE1

```
neighbor interface Ethernet1-48 peer-group LEAVES peer-filter LEAVES  
neighbor interface Ethernet49/1-52/1 peer-group SUPERSPINES peer-filter SUPERSPINES
```

- While we can predict this in our lab, production isn't as nice

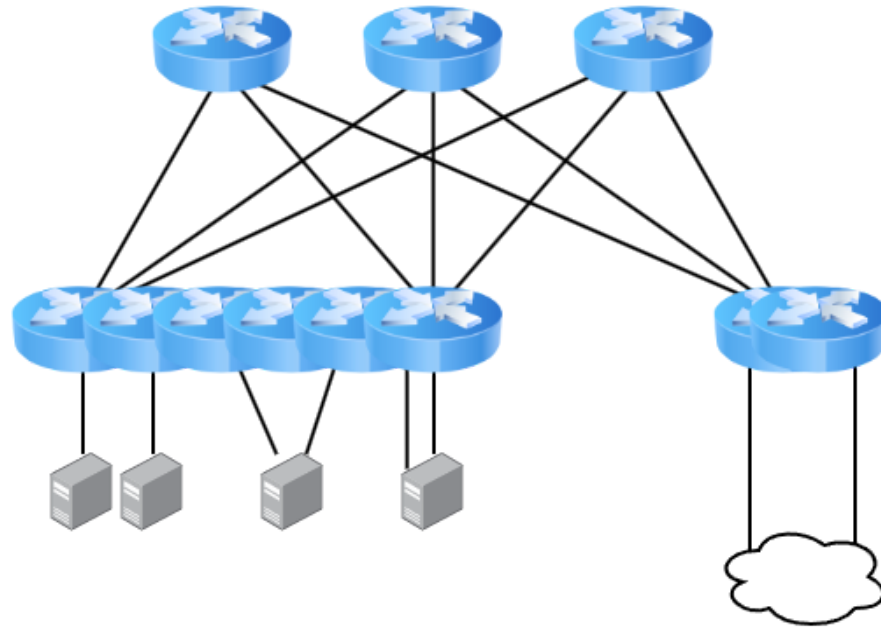
```
neighbor interface Ethernet52/4 peer-group LEAVES peer-filter TEMP-LEAF
```



# Hosts

- If your virtualization and tenancy model supports it, go deeper!
- Assign hosts router-ids
  - Deployment
  - Custom DHCP
- Advertise reachability via host-based BGP
  - GoBGP
  - FRR
- IPv6 LL Autodetection Supported!

# Hosts



- Connect servers to one or more leaves
- No more Layer 2 problems!
  - LACP/MLAG
  - Spanning Tree

# Hosts

```
router bgp 65001.1034
  neighbor SPINES peer-group
  neighbor SERVERS peer-group
  neighbor interface Ethernet1-48 peer-group SERVERS peer-filter SERVERS
  neighbor interface Ethernet49/1-52/4 peer-group SPINES peer-filter SPINES
  address-family ipv4 unicast
    bgp next-hop address-family ipv6
    neighbor SERVERS activate
    neighbor SERVERS next-hop address-family ipv6 originate
    neighbor SPINES activate
    neighbor SPINES next-hop address-family ipv6 originate
  !
  address-family ipv6 unicast
    neighbor SERVERS activate
    neighbor SPINES activate
  !
!
```

```
router bgp 65001.1034
  neighbor SPINES peer-group
  neighbor SERVERS peer-group
  neighbor interface Ethernet1-48 peer-group SERVERS peer-filter SERVERS
  neighbor interface Ethernet49/1-52/4 peer-group SPINES peer-filter SPINES
  address-family ipv4 unicast
    bgp next-hop address-family ipv6
    neighbor SERVERS activate
    neighbor SERVERS next-hop address-family ipv6 originate
    neighbor SPINES activate
    neighbor SPINES next-hop address-family ipv6 originate
  !
  address-family ipv6 unicast
    neighbor SERVERS activate
    neighbor SPINES activate
  !
!
```

- What would be great is if remote AS determined policy
- Ask your vendor if this is something they can support
- May require stronger security (RFC 5925 TCP-AO)

# Recap

- Eliminated boilerplate
  - templates
  - anycast
- Derived as much as possible from the router-id
- Removed non-loopback addressing
- BGP to everything

# Context

```
nodes:
- hostname: dc1-leaf1
  region: dc1
  role: leaf
  aaa:
  - 192.168.3.2
  - 192.168.10.1
  - 192.168.14.10
  addresses:
    Loopback0:
      ipv4:
      - 172.18.4.10/32
      ipv6:
      - "2001:db8::1/128"
    Ethernet1/1:
      ipv4:
      - 10.0.0.0/31
      ipv6:
      - "2001:2b8:1::1/64"
  bgp:
    router-id: 172.18.4.10
    asn: "65534"
    ipv4:
      peer-groups:
      - name: SPINE-LEAF
        outbound-policy: SPINE-TO-LEAF
        inbound-policy: LEAF-TO-SPINE
      neighbors:
      - address: 10.0.0.1
        as: "65534"
```

```
nodes:
- router-id: 172.18.8.0      # site1-spine
- router-id: 172.18.8.1    # site1-spine
- router-id: 172.18.4.2    # site1-leaf
- router-id: 172.18.4.3    # site1-leaf
- router-id: 172.18.4.4    # site1-leaf
- router-id: 172.18.0.5    # site1-server
- router-id: 172.18.0.6    # site1-server
- router-id: 172.18.0.7    # site1-server
- router-id: 172.18.0.8    # site1-server
- router-id: 172.18.28.9   # site2-superspine
- router-id: 172.18.24.10  # site2-spine
- router-id: 172.18.20.11  # site2-leaf
- router-id: 172.18.16.12  # site2-server
```







**Thank you**  
**Questions?**

[www.github.com/bewing/chinog-10](http://www.github.com/bewing/chinog-10)

# Bitshifting

```
func loadNodeData(routerId string) (NodeData, error) {
    nd := NodeData{}
    ip, err := netip.ParseAddr(routerId)
    if err != nil {
        return nd, err
    }
    data := ip.AsSlice()[2]
    typeByte := data & 12 >> 2
    if typeByte^1 == 0 {
        nd.Type = "leaf"
        nd.Layer = 1
    } else if typeByte^2 == 0 {
        nd.Type = "spine"
        nd.Layer = 2
    } else if typeByte^3 == 0 {
        nd.Type = "superspine"
        nd.Layer = 3
    } else {
        nd.Type = "server"
        nd.Layer = 0
    }
    nd.Region = int(data & 192 >> 6)
    nd.Site = int(data & 48 >> 4)

    nd.ASN = fmt.Sprintf("65%d%d%d.%d", nd.Region, nd.Site, nd.Layer, int(ip.AsSlice()[2])*256+int(ip.AsSlice()[3]))
    return nd, nil
}
```