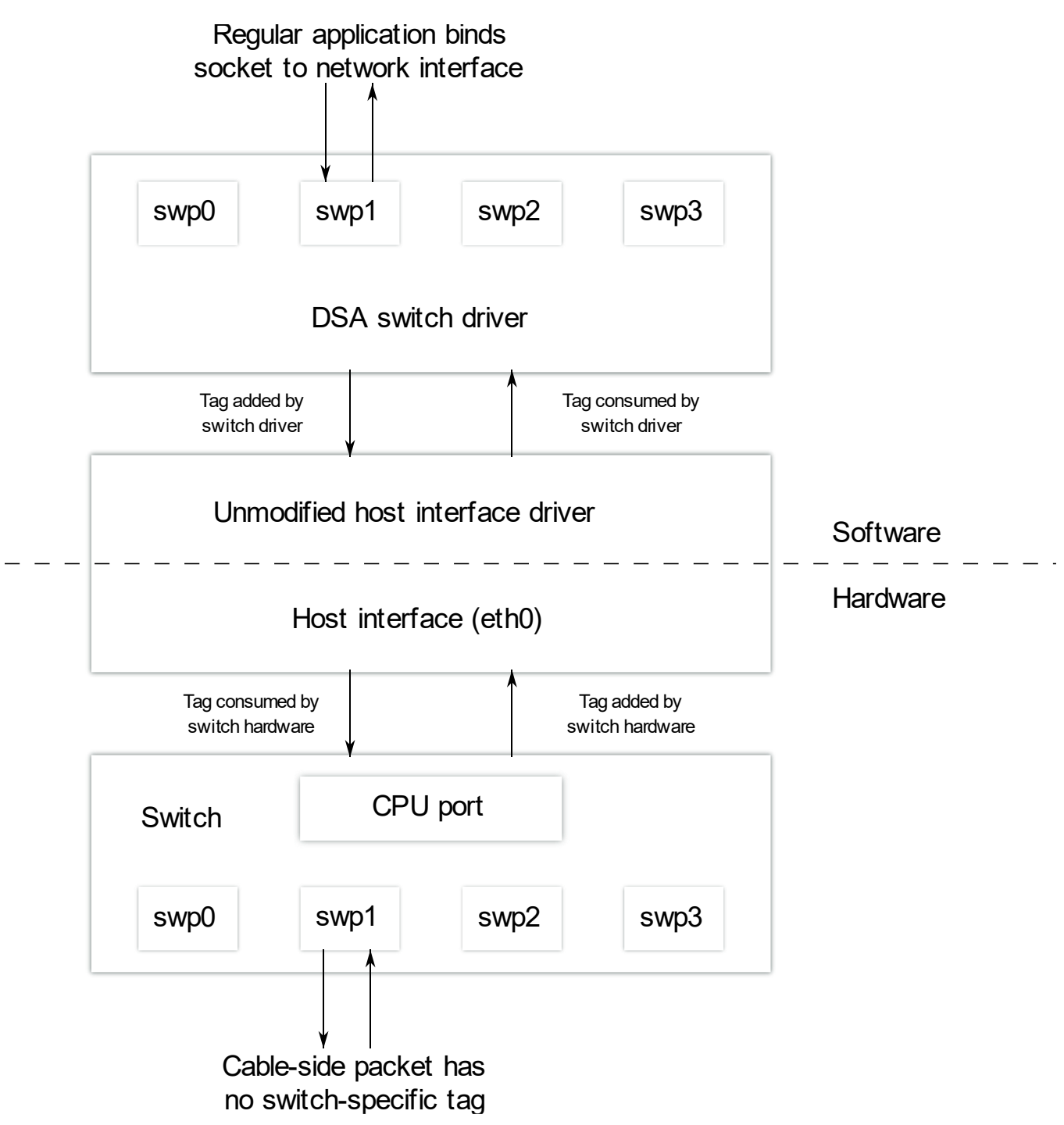
LINUX PLUMBERS CONFERENCE September 20-24, 2021

DSA switches: domesticating a savage beast

Vladimir Oltean

Distributed Switch Architecture: an overview

- Framework for managing Ethernet switches
- Driver writer FAQ: *should I write a plain switchdev or a DSA driver for my hardware?*
  - switch supports direct packet I/O => plain switchdev
  - switch supports indirect packet I/O through an Ethernet port of the host => DSA

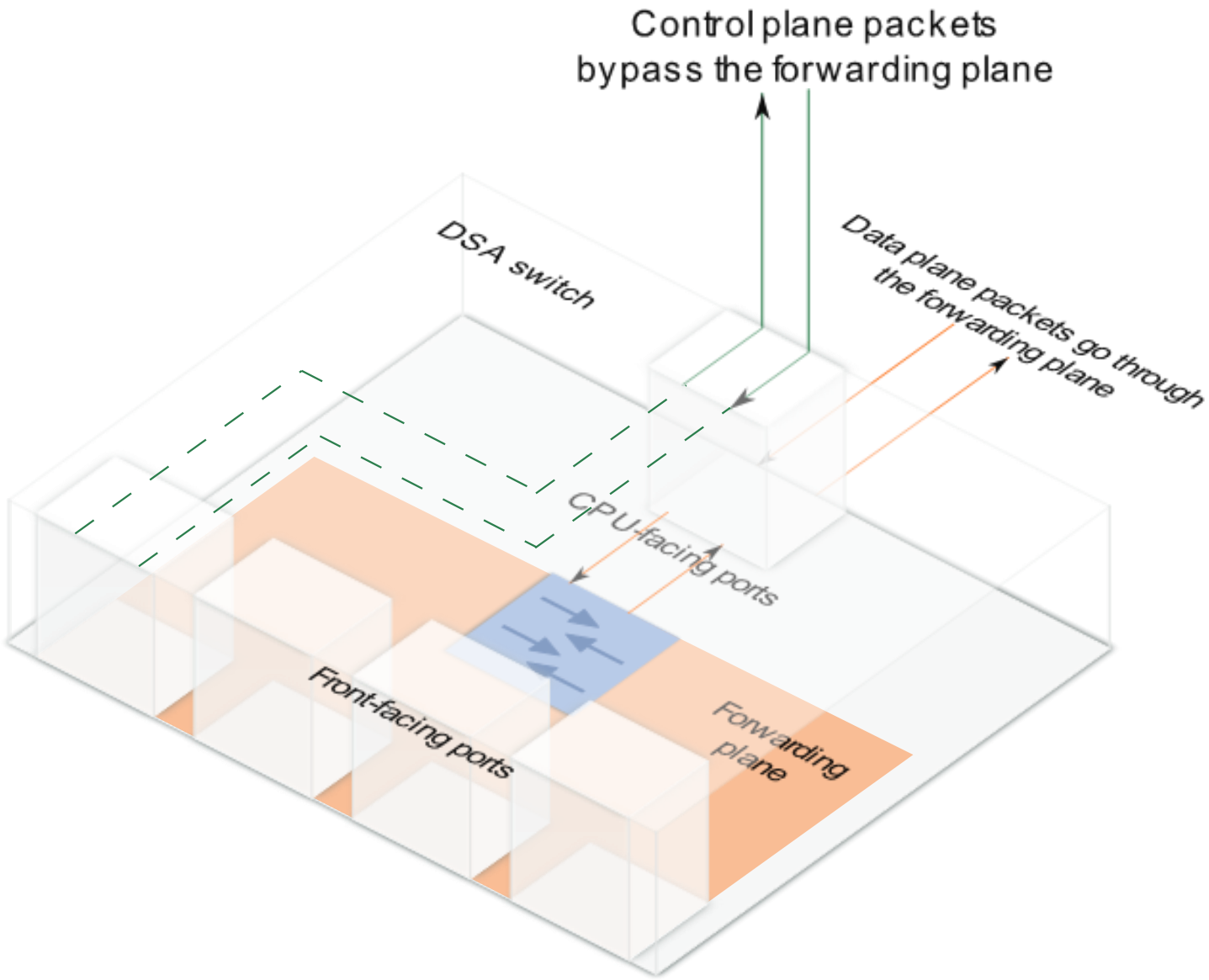


Summary of changes to DSA

- Separation between control plane and data plane packets
- Support for unoffloaded upper interfaces
- RX filtering
- Support for cross-chip bridging in more varied topologies
  - Disjoint trees
  - “H” trees

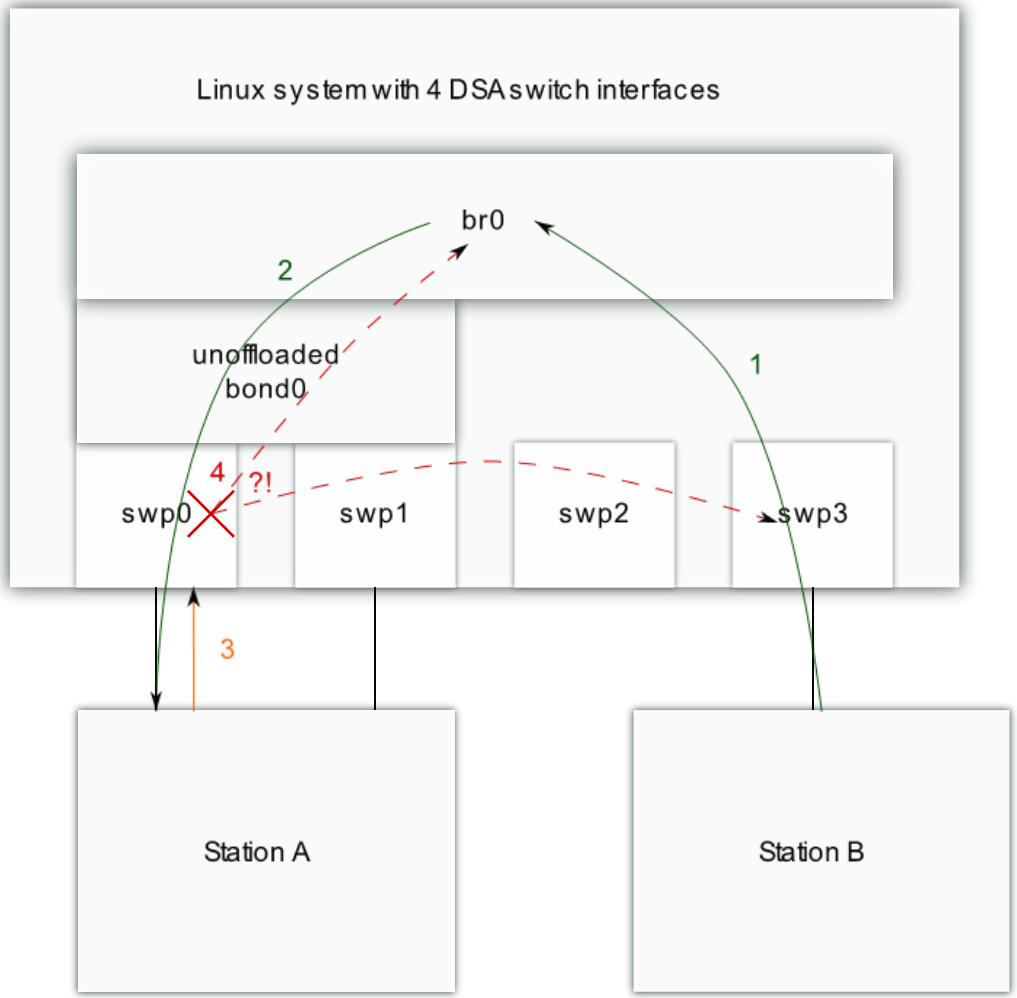
The data plane and the control plane

- Different classes of switches
  - Fully managed
  - Unmanaged
  - Lightly managed
- tag_8021q can help with the unmanaged and lightly managed switches, but only until the bridge claims the VLAN table
  - must teach the bridge about data plane packets
  - TX forwarding offload: extend skb->offload_fwd_mark for TX



Offloading software upper interfaces

- Offloading support added for LAG and HSR/PRP
- Repaired the software fallback which got broken when DSA was integrated with switchdev
  - API added to switchdev for drivers to explicitly declare that they offload a bridge port
- FDB isolation is still an issue



1. Station B sends an ICMP Echo request to Station A. Station A is behind an unoffloaded bridge port (bond0), so swp3 must forward it towards the CPU.
swp3 is an offloaded bridge port, so Station B's MAC address is learned on this source port.

2. The bridge reinject it towards bond0 which will send towards either swp0 or swp1 according to the xmit hash policy

3. Station A generates an ICMP echo reply towards Station B's MAC address.

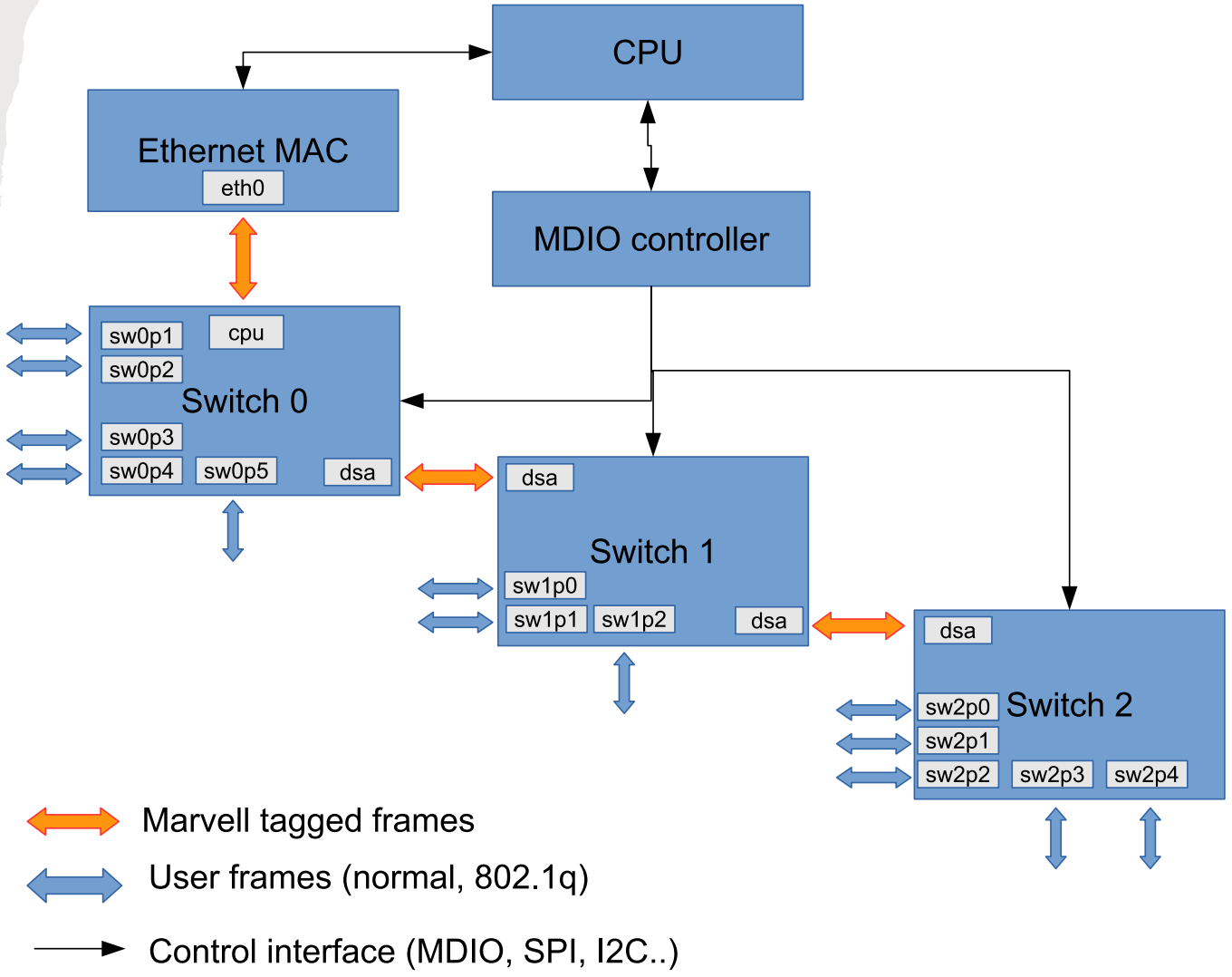
4. In lack of FDB isolation, swp0 will look up Station B's destination MAC address and will find the entry pointing towards swp3, so it will attempt to shortcircuit the CPU and forward towards that port directly. But forwarding is blocked from swp0 towards that port => packet is dropped.

RX filtering

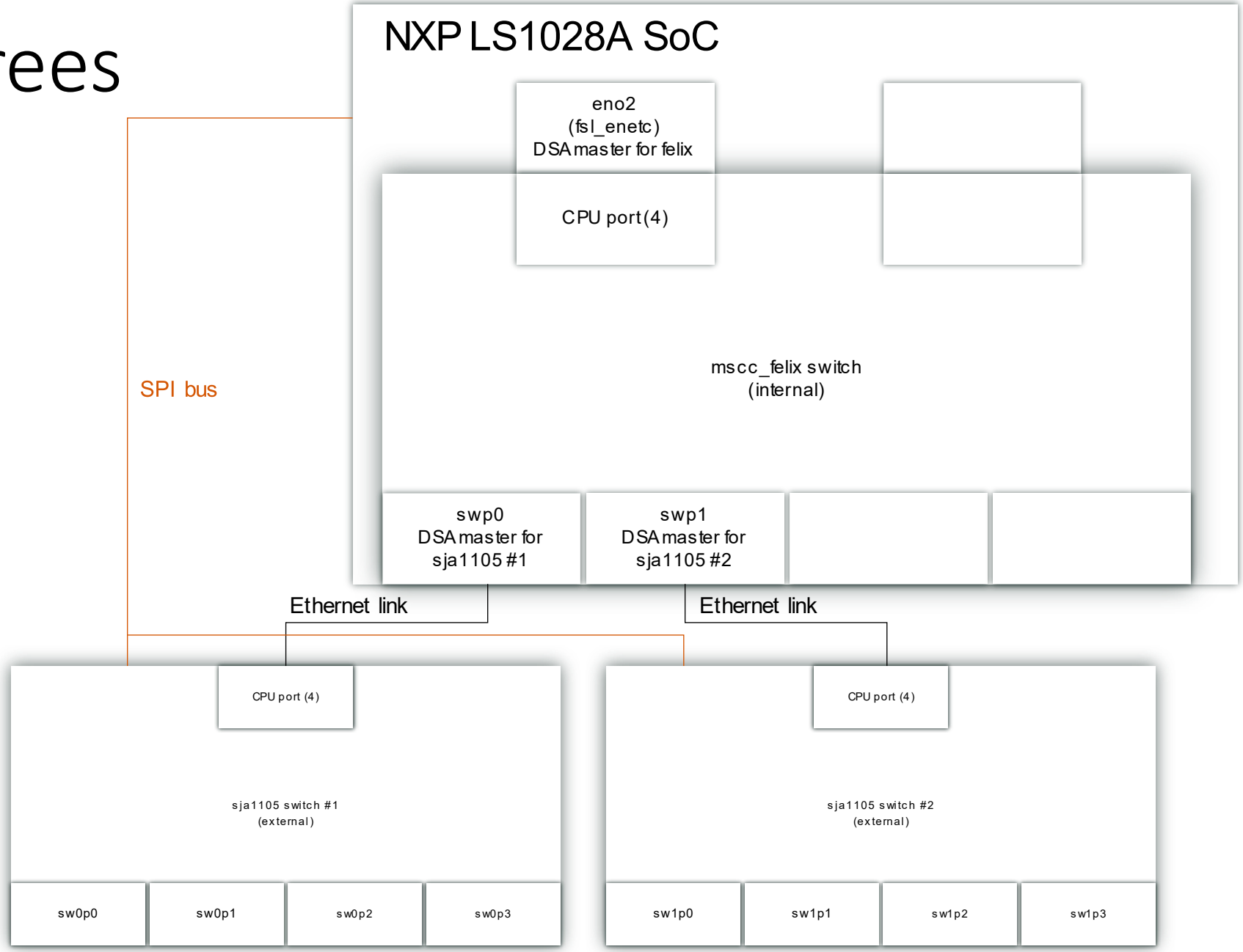
- No RX filtering for standalone ports (IFF_UNICAST_FLT), just bridged
- Tell the hardware which addresses must be filtered towards the host
  - Assisted learning on the CPU port replaces hardware source address learning, sniffs switchdev FDB events on foreign interfaces
  - Port MAC addresses, the bridge device MAC address are offloaded by switchdev as local/permanent FDB entries
- Still cannot remove CPU port from the flooding domain of user ports
  - DSA interfaces might be bridged with foreign interfaces
  - Bridge with upper interfaces might become promiscuous (no IFF_UNICAST_FLT)

Switch topology changes

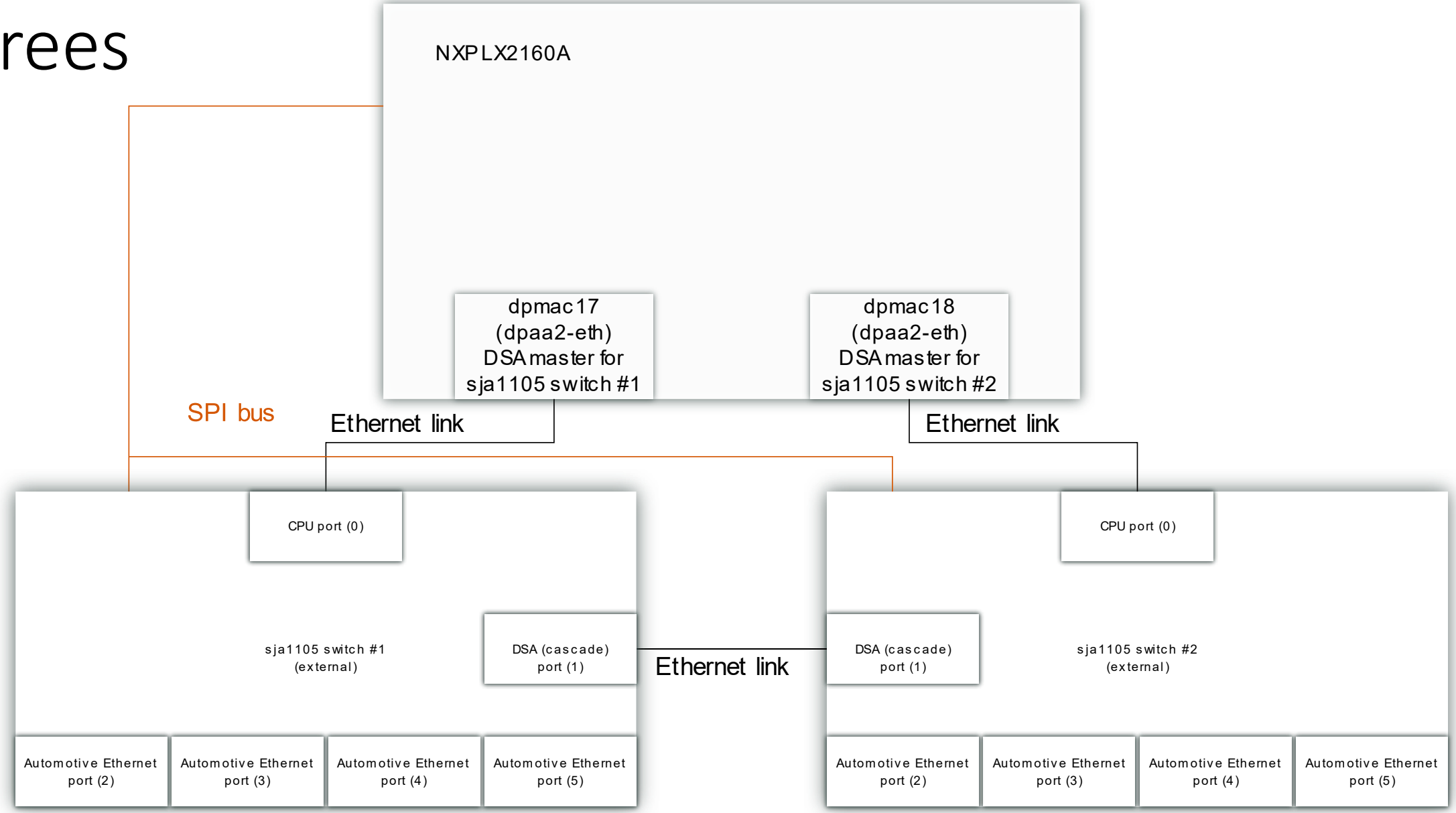
- Daisy chains have been the norm, but crazy people always like to “innovate”
- Changes at the cross-chip notifier level to support bridging between DSA user ports in other circumstances



Disjoint trees



H trees


NXP LX2160A
dpmac17
(dpaa2-eth)
DSA master for
sja1105 switch #1
dpmac18
(dpaa2-eth)
DSA master for
sja1105 switch #2
SPI bus
Ethernet link
Ethernet link
CPU port (0)
CPU port (0)
sja1105 switch #1
(external)
DSA (cascade)
port (1)
Ethernet link
DSA (cascade)
port (1)
sja1105 switch #2
(external)
Automotive Ethernet
port (2)
Automotive Ethernet
port (3)
Automotive Ethernet
port (4)
Automotive Ethernet
port (5)
Automotive Ethernet
port (2)
Automotive Ethernet
port (3)
Automotive Ethernet
port (4)
Automotive Ethernet
port (5)

Conclusion

- Transforming the wide variety of DSA switches into something that is compatible with the network stack's expectations requires a good amount of creativity
- The risk is that we might shoehorn them into something that departs from the use case they were intended for
- For the finer points, be sure to read the full paper with the same name!
  - https://linuxplumbersconf.org/event/11/contributions/949/