Experiences Building an OFI Provider for usNIC

"Why we loves the libfabric"

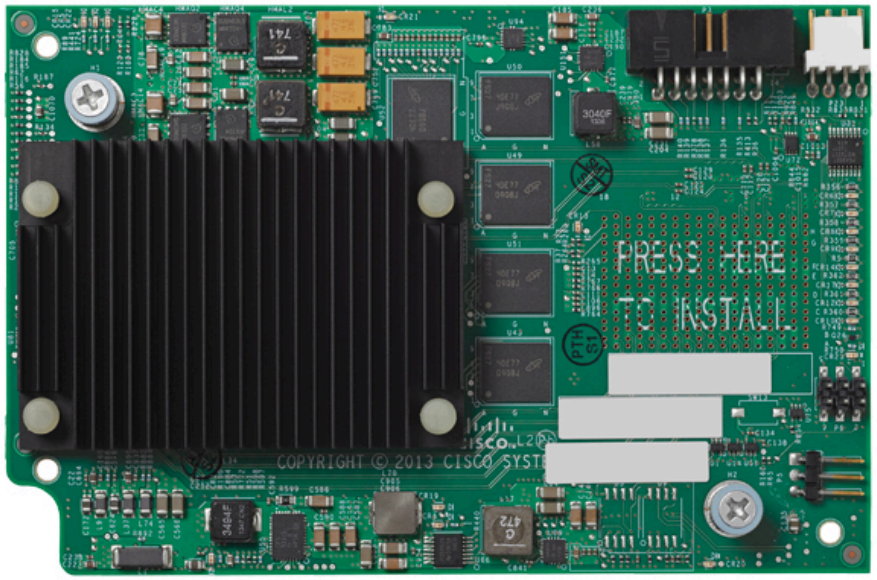
Jeffrey M. Squyres

16 March 2015

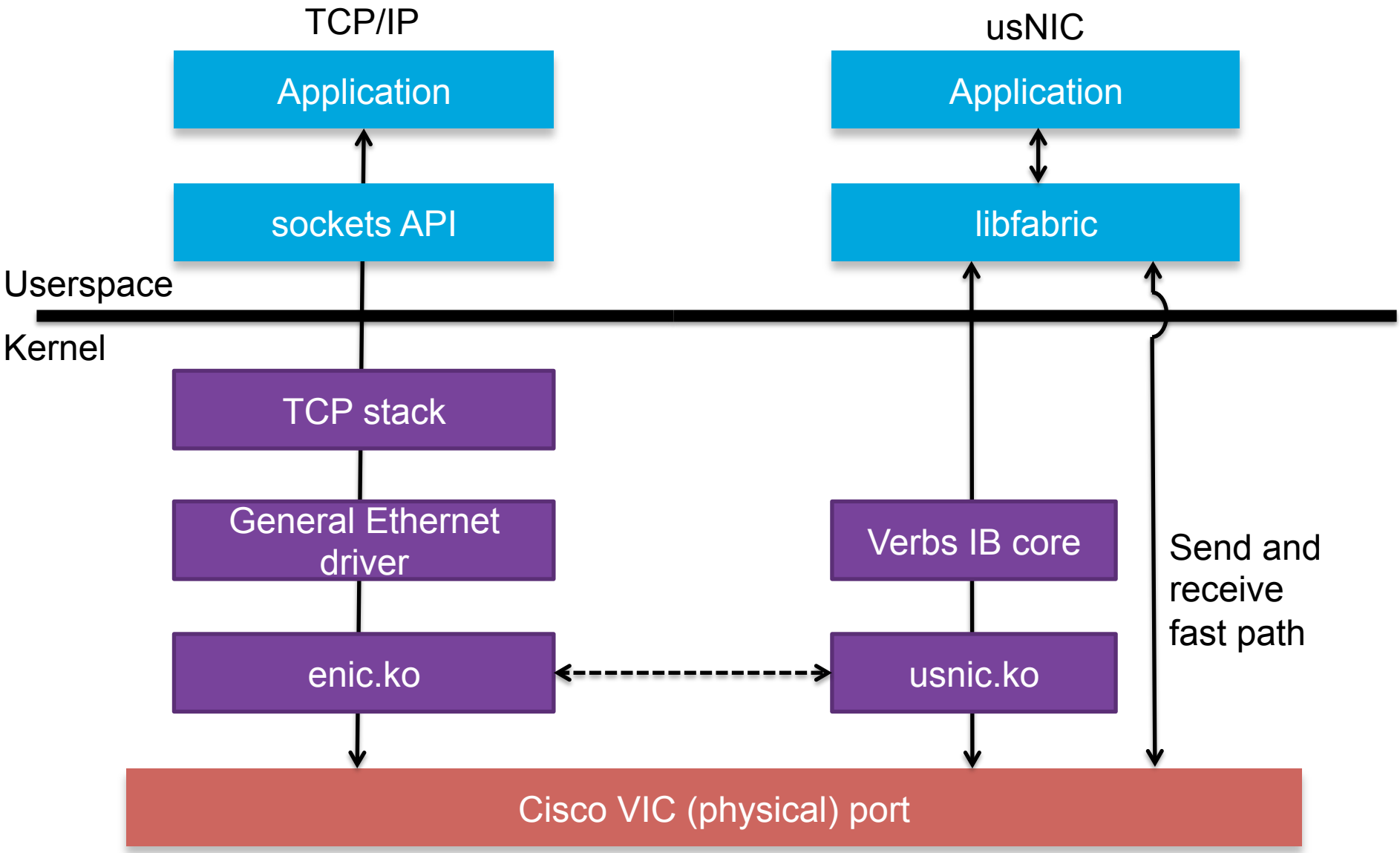
Cisco VIC overview

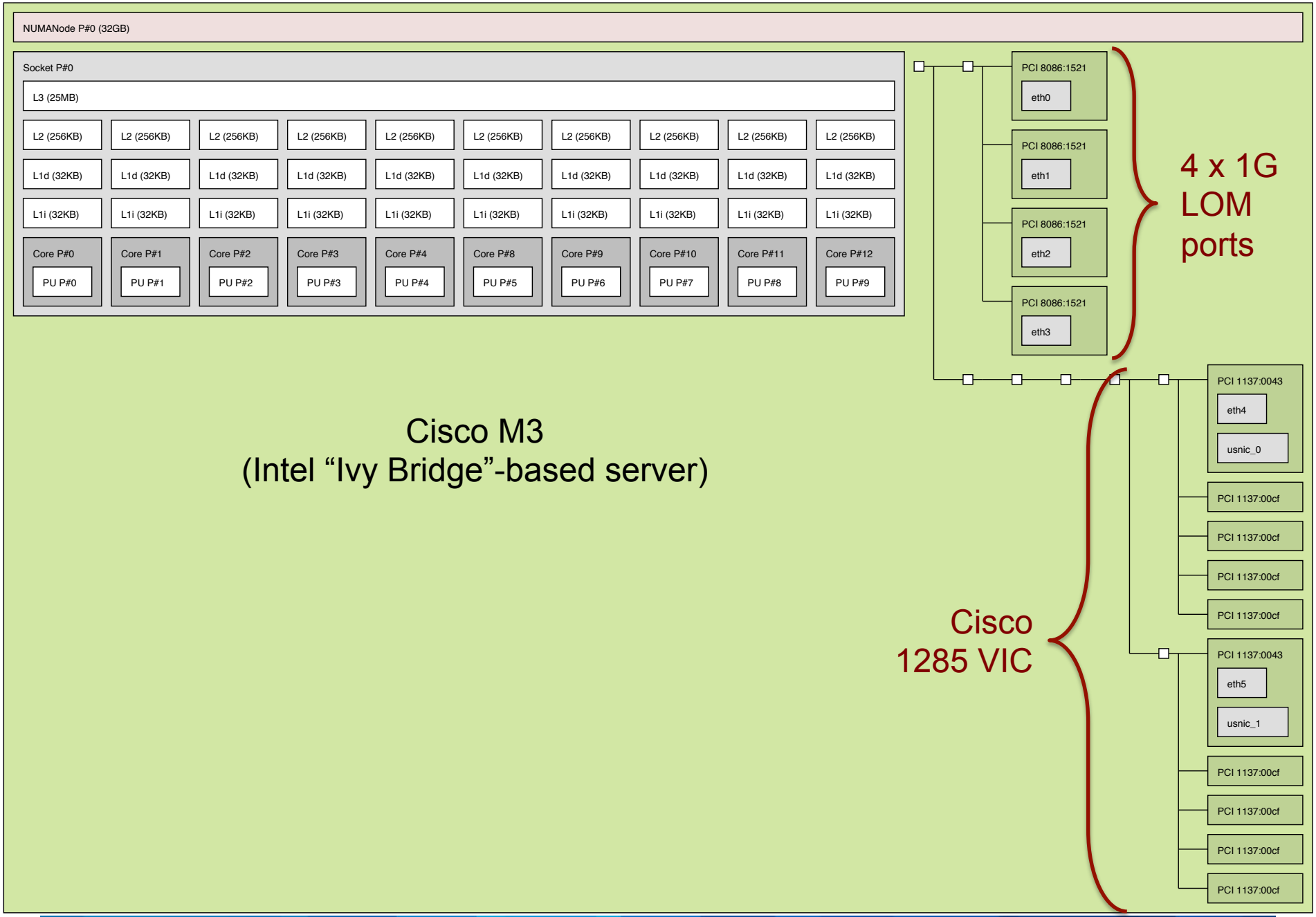
- Cisco Virtual Interface Card (VIC)
- Converged, virtualized NIC
  - Ethernet, FCoE
  - SR-IOV (PCI PF, VF)
- 3rd generation **80Gbps** Cisco ASIC
  - 2 x 40Gbps Ethernet ports
  - Mezzanine form factor: shipping now
  - PCI form factor: shipping soon

Cisco VIC 1380 (3g Mezz, dual 40G)

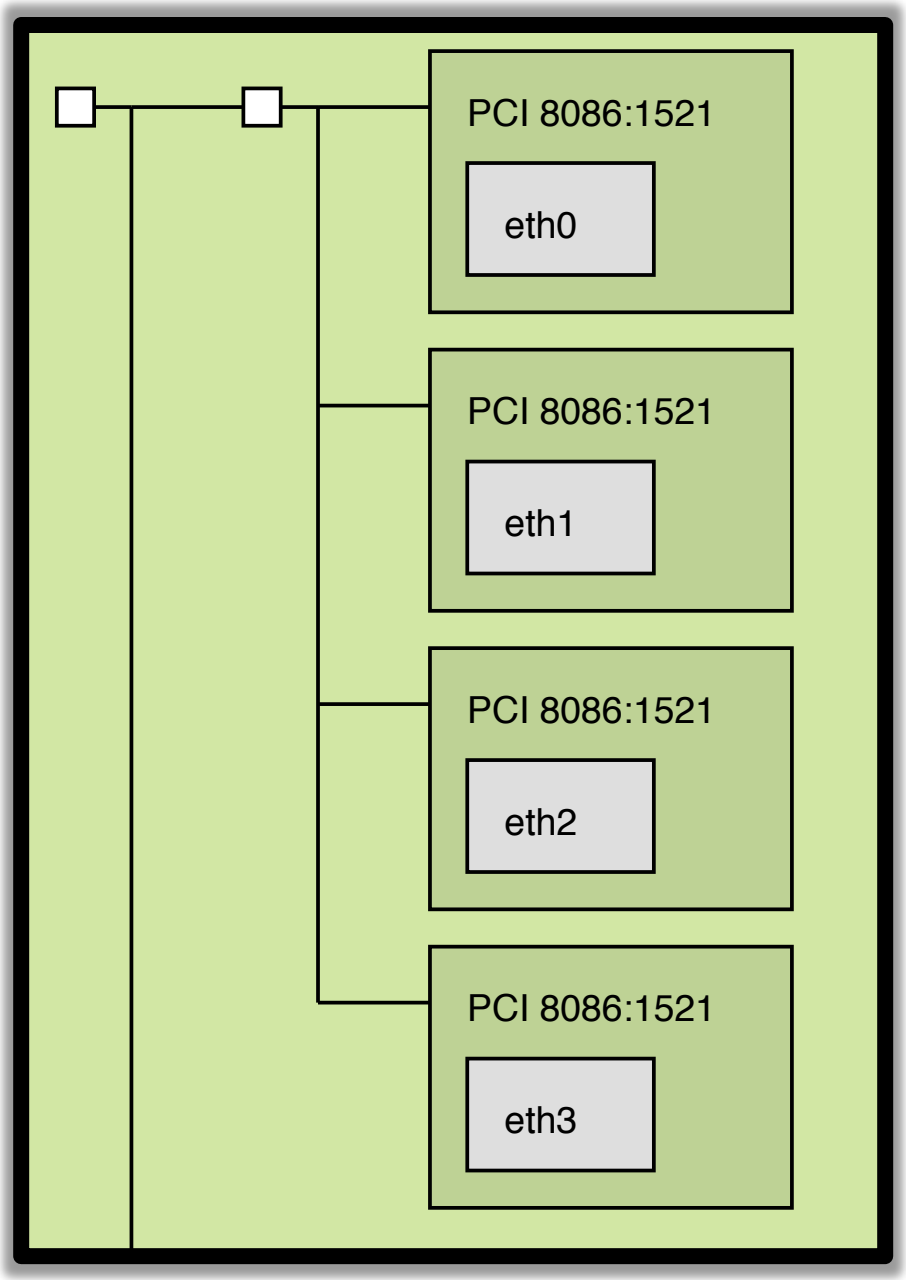


usNIC: OS bypass to the same ethX interface

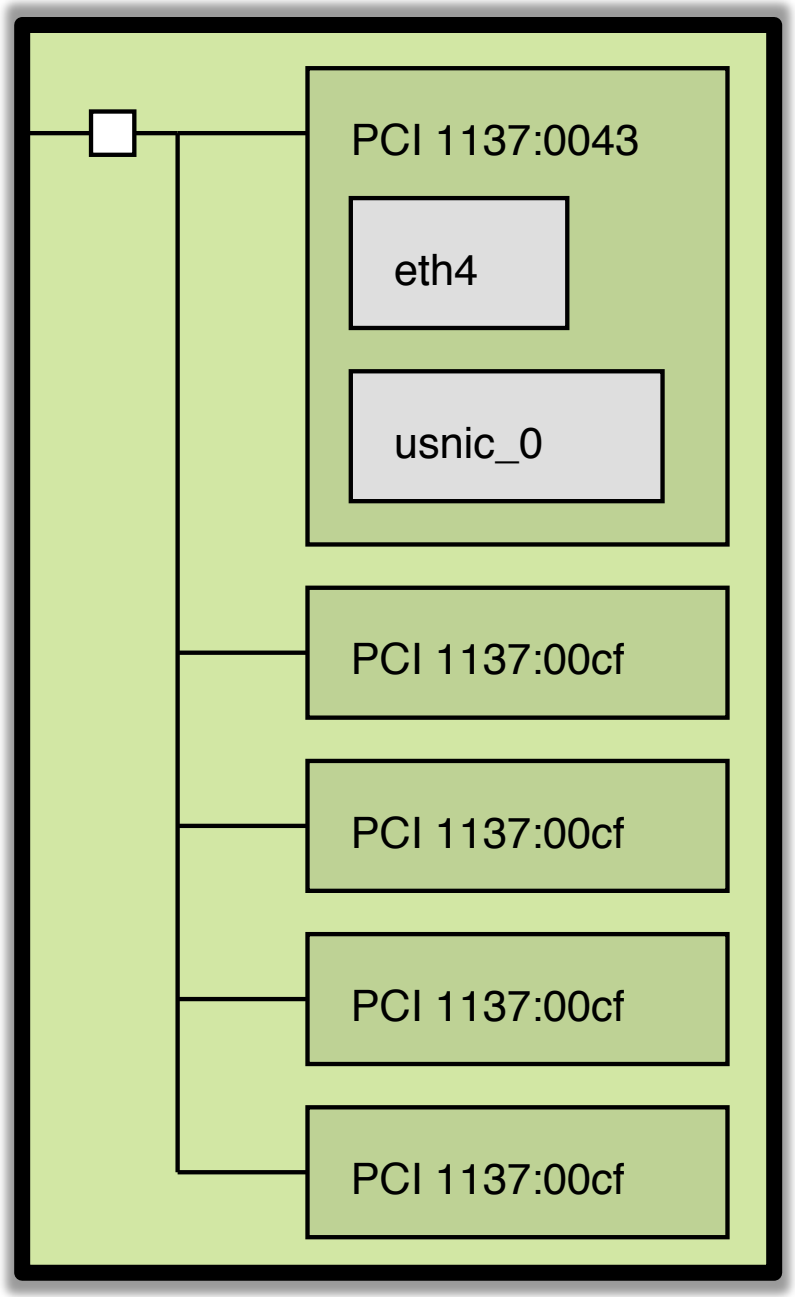




4G x 1G
LOM
ports
(ignore these)



Cisco
1285 VIC
(one of the dual
ports)



Verbs is a fine API.

…if you make InfiniBand hardware.

…but now there's this libfabric thing

Which API should be our way forward for kernel bypass?

Keep in mind, Cisco already has a UD verbs provider

Verbs

Pros

- Mature, stable
- Only way to get kernel provider upstream
- Brand-name recognition
- Already shipping a Cisco UD verbs provider

Cons

- Highly InfiniBand-specific
- Dominated by a single vendor
  - Common usage full of that vendor's extensions
- Upstream maintainer is disinterested, not part of the community

Libfabric

Pros

- New
  - Design for modern hardware, software
- Much more general hardware model
- No legacy / backwards compatibility issues (yet)
- Co-design with MPI community
- Active community

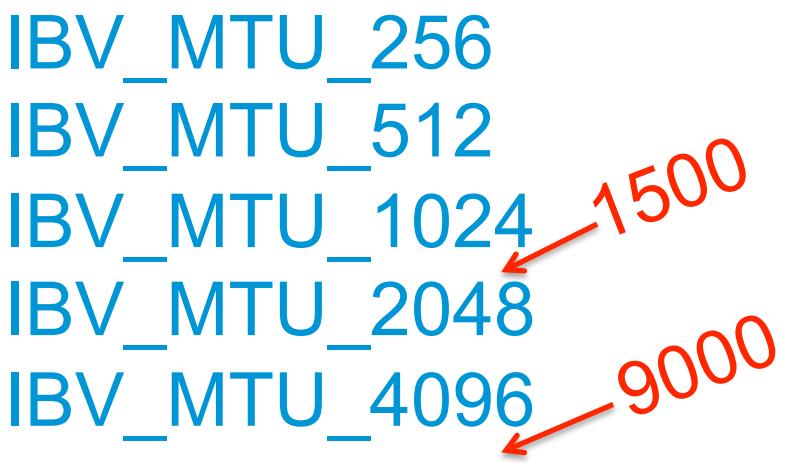
Cons

- New
  - Must educate partners / customers
- Does not exactly match IB verbs kernel interface

Comparison: MTU

Verbs

- Monotonic enum
- Could not add popular Ethernet values
  - 1500
  - 9000
- usNIC verbs provider had to lie (!)
  - …just like iWARP providers
- MPI had to match verbs device with IP interface to find real MTU



Comparison: MTU

Libfabric

- Integer (not enum) endpoint attribute

Comparison: MTU

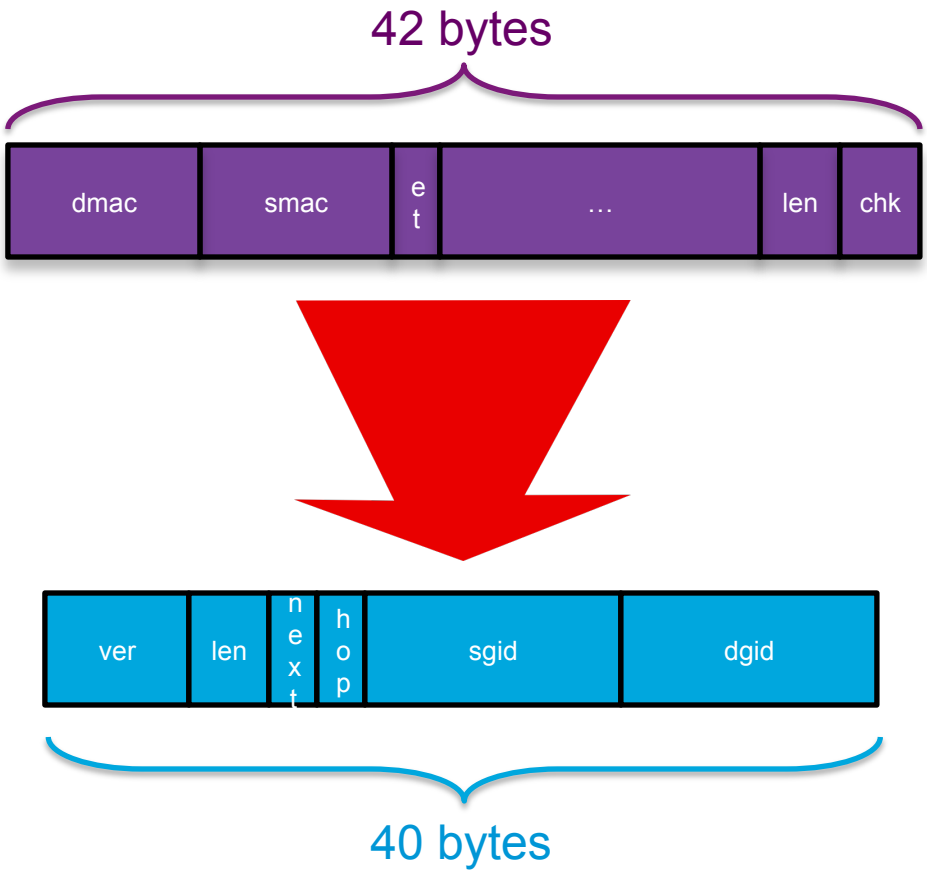
Libfabric

- Integer (not enum) endpoint attribute

Comparison: Unreliable datagram

Verbs

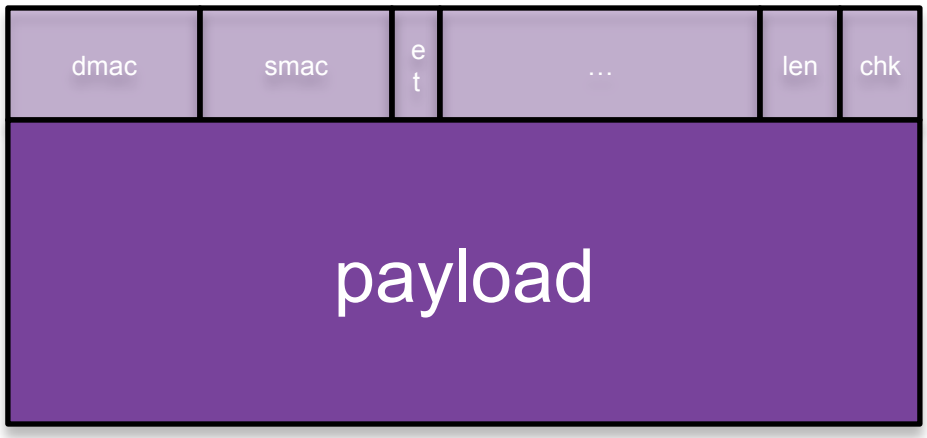
- Mandatory GRH structure
  - InfiniBand-specific header
- 40 bytes
  - UDP header is 42 bytes
  - …and a different format
- Breaks ib_ud_pingpong
- usnic verbs provider used “magic” ibv_port_query() to return extensions pointers
  - E.g., enable 42-byte UDP mode



Comparison: Unreliable datagram

Libfabric

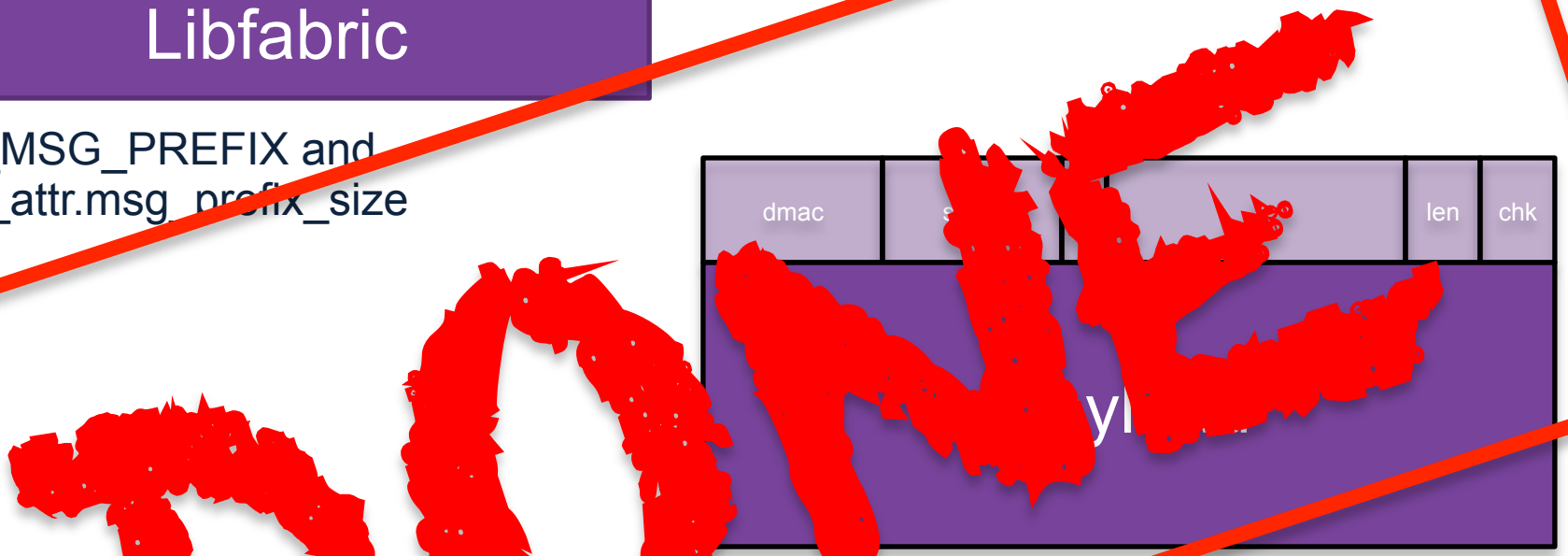
- FI_MSG_PREFIX and ep_attr.msg_prefix_size



Comparison: Unreliable datagram

Libfabric

- FI_MSG_PREFIX and ep_attr.msg_prefix_size



DONE

Comparison: Reliable datagram

Verbs

- Not implemented
- (Assumed to be) Too much work to get upstream

Sad panda needs a hug

Comparison: Reliable datagram

Libfabric

- FI_EP_RDM

Comparison: Reliable datagram

Libfabric

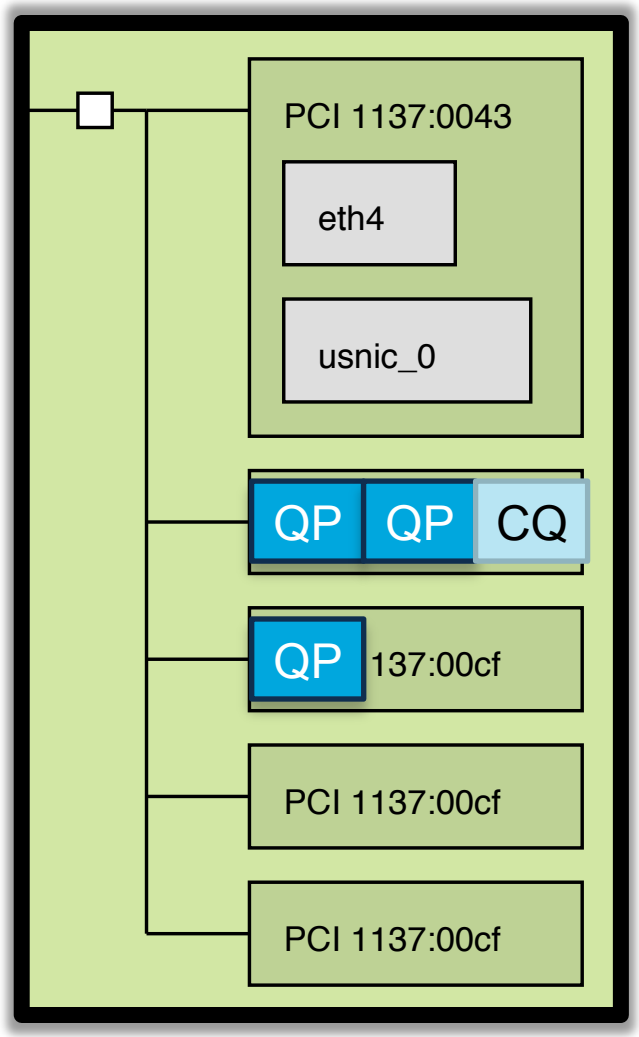
- FI_EP_RDM

Comparison: Hardware model

Verbs

- Tuple: (device, port)
  - Usually a physical device and port
  - Does not match virtualized VIC hardware
- Queue pair
- Completion queue

ibv_device
ibv_port



Comparison: Hardware model

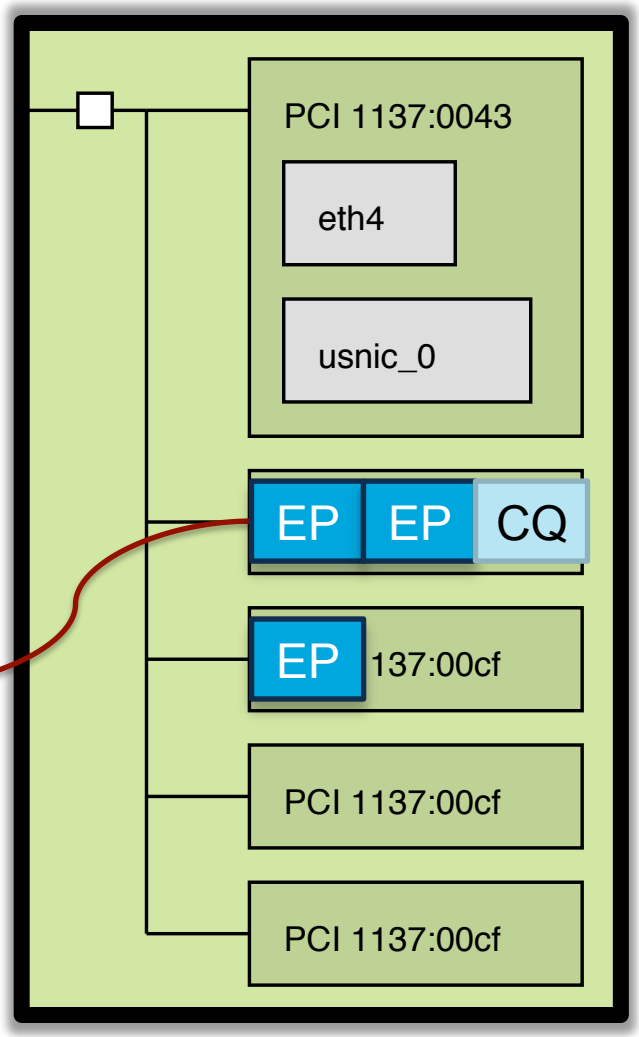
Libfabric

- Maps nicely to SR-IOV
- Fabric → Physical function (PF)
- Domain → Virtual function (VF)
- Endpoint → Resources in VF

fi_fabric

fi_domain

fi_endpoint
(resources in domain)



Comparison: Addressing

Verbs

- GID and GUID
  - No easy mapping back to IP interface
- usnic verbs provider encoded MAC in GID
  - Still cumbersome to map back to IP interface
- Could use RDMA CM
  - …but that would be a ton more code

```
mac[0] = gid->raw[8] ^ 2;
mac[1] = gid->raw[9];
mac[2] = gid->raw[10];
mac[3] = gid->raw[13];
mac[4] = gid->raw[14];
mac[5] = gid->raw[15];
```

Comparison: Addressing

Libfabric

- Can use IP addressing directly

Everything is awesome

Comparison: Addressing

Libfabric

- Can use IP addressing directly

Everything is awesome

Comparison: Netmask

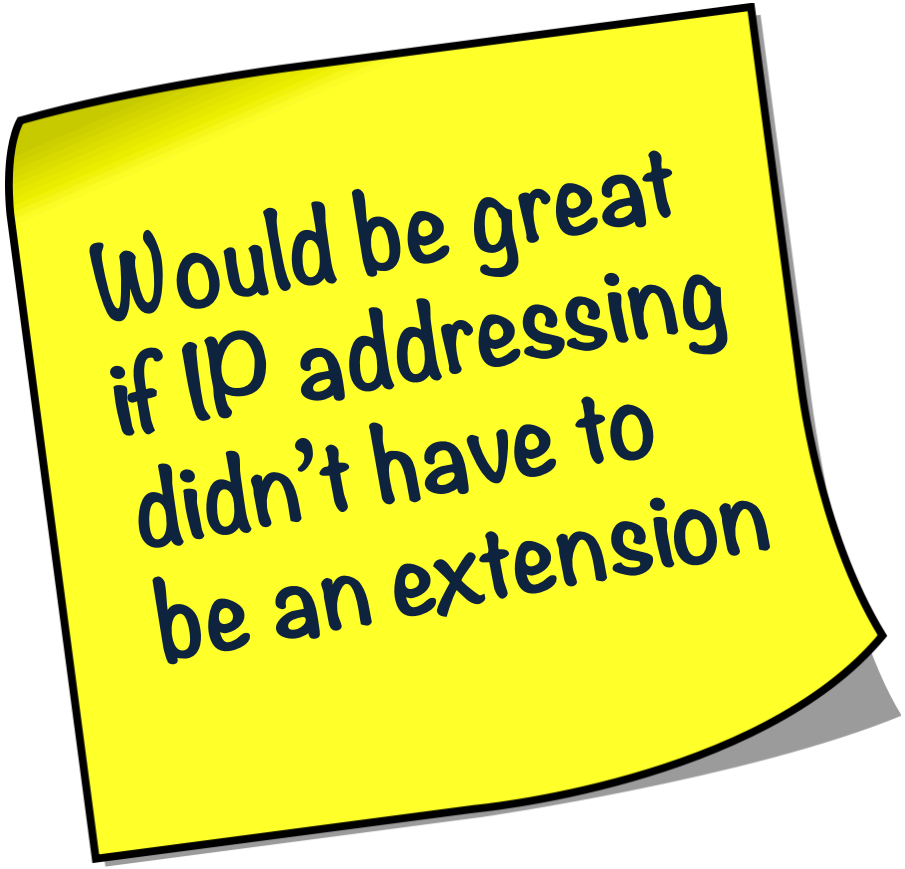
Verbs

- Fuggedaboutit

Comparison: Netmask

Libfabric

- usnic provider extension
  - Included in the upstream API
- Directly obtain:
  - IP: Netmask
  - IP: Linux interface name
  - Physical: Link speed
  - SR-IOV: Number of VFs
  - SR-IOV: QPs per VF
  - SR-IOV: CQs per VF



Comparison: Performance

Verbs

- Generic send call
  - ibv_post_send(…SG list…)
  - Lots of branches
- Wasteful allocations
- No prefixed receive
- Branching in completions

Comparison: Performance

Libfabric

- Multiple types of send calls

  fi_send(buffer, …)

- Variable-length prefix receive

  Provider-specific

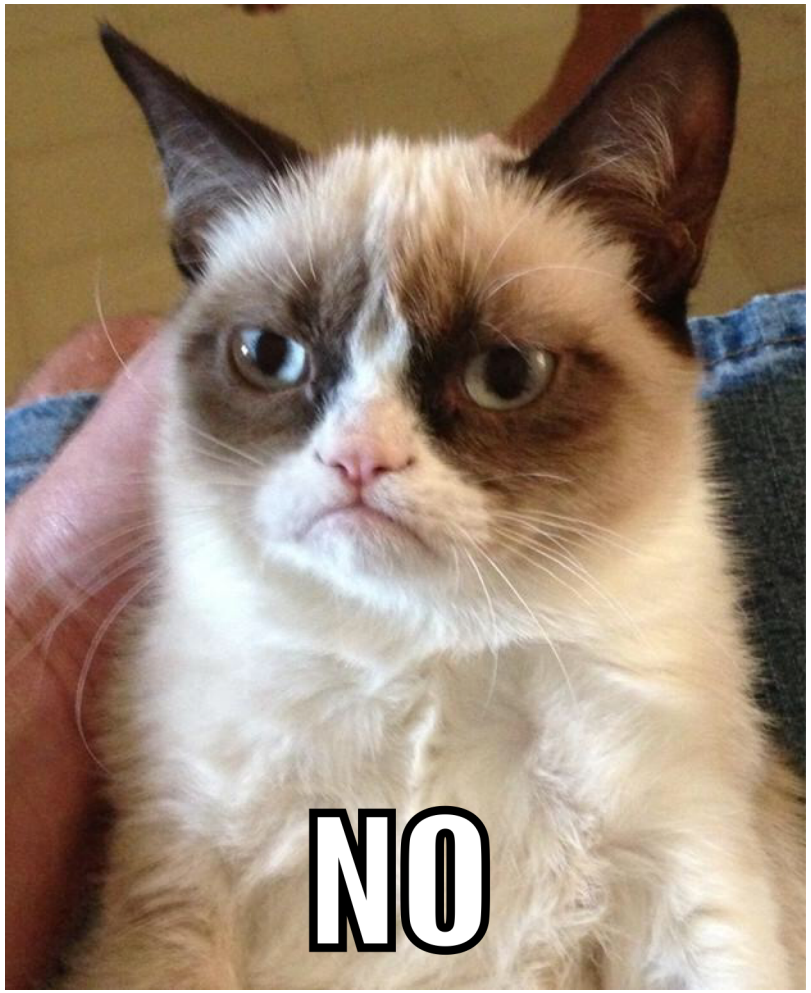
- Fewer branches in completions

(see Open MPI presentation later today)

Comparison: Application centricity

Verbs

- Performance issues
- Memory registration still a problem
- No MPI-style tag matching
- One-sided capabilities do not match MPI
- Network topology is a separate API



Comparison: Application centricity

Libfabric

- Performance happiness
- Many MPI-helpful features:
  - Tag matching
  - One-sided operations
  - Triggered operations
- Inherently designed to be more than just point-to-point
- More work to be done… but promising
  - MMU notify
  - Network topology

Conclusions

Verbs

- Long design discussions about how to expose Ethernet / VIC concepts in the verbs API

  …usually with few good answers

  Especially problematic with new VIC features over time

- Eventually resulted in horrible "magic" port query hack

- Conclusion: possible (obviously), but not preferable

Libfabric

- Whole API designed with multiple vendor hardware models in mind
- Still "new" enough to be able to change APIs when corner cases are found
- Much easier to match our hardware to core Libfabric concepts

- Conclusion: much more preferable than verbs

Thank you.