Not Another Boring Vendor Talk

This is what they get for sending an engineer, not a marketing guy

Jeffrey M. Squyres
Cisco Systems

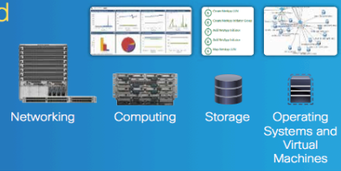
23 September 2015

Cisco® Unified Computing System

With Intelligent Intel® Xeon® Processors

Manage Cisco UCS Integrated Infrastructure Solutions

Cisco UCS Director Software
Automate integrated infrastructure orchestration and management



Manage Multiple Domains Worldwide

Cisco UCS Central Software
Manage multiple domains on the same campus or worldwide

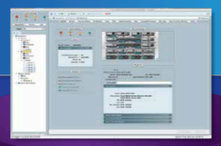
Manage a Single Domain

XML API

Command-Line Interface (CLI)

Cisco UCS® Manager
Scale up to 160 blade or rack servers in a single management domain

GUI



Cisco UCS Integrated Infrastructure Solutions

Accelerate and simplify application deployment

FlexPod

VCE Vblock™ System

Cisco Solutions for EMC VSPEX

Nimble Storage SmartStack

Cisco Solutions for Hitachi UCP Select

VersaStack Solution by Cisco and IBM

Cisco UCS 6324 Fabric Interconnect
Creates an all-in-one Cisco UCS Mini solution for remote offices and branch offices

Cisco UCS 5108 Blade Server Chassis

Cisco UCS 2200 Series Fabric Extenders
Scale without complexity

Cisco UCS 6200 Series Fabric Interconnects
Single point of connectivity and management

Optional Cisco Nexus® 2232PP 10GE Fabric Extenders

Cisco R-Series Racks

Cisco SingleConnect Technology
Connect LAN, SAN, and management networks and physical and virtual servers with one physical connection

Cisco UCS B-Series Blade Servers

Cisco UCS B200 M3 and M4

Cisco UCS B420 M3 and M4

Enterprise Class

Cisco UCS C-Series Rack Servers

Cisco UCS Virtual Interface Card (VIC) 1340

Cisco UCS B460 M4

Cisco UCS B260 M4

Cisco UCS C460 M4

Mission Critical

Cisco UCS Virtual Interface Card (VIC) 1225

Cisco UCS C240 M3 and M4

Cisco UCS C220 M3 and M4

Cisco UCS C3160

Enterprise Class

320 servers in a single rack

Cisco UCS M-Series Modular Servers

Cisco UCS M4308 Modular Chassis

Cisco UCS M142 Compute Cartridge

Cisco UCS Invicta™ Series

Cisco UCS Invicta C710SR Routers

Cisco UCS Invicta C3124SN Nodes

Cisco UCS Invicta Scaling System

Cisco UCS Invicta C3124SA Appliance

Solid-State Application Acceleration

Cisco® Unified Computing System

With Intelligent Intel® Xeon® Processors

Cisco has fantastic server products

Please buy some

Manage Cisco UCS Integrated Infrastructure Solutions

Cisco UCS Director Software
Automate integrated infrastructure orchestration and management

Networking

Manage Multiple Domains

Cisco UCS Central Software
Manage multiple domains across multiple campus or

Manage a Single Domain

Cisco UCS Integrated Infrastructure Solutions

Accelerate and simplify application deployment

FlexPod

VCE Vblock™ System

Solutions
C VSPEX

Nimble Storage SmartStack

Cisco Solutions for Hitachi UCP Select

VersaStack Solution by Cisco and IBM

Cisco UC
Fabric Inte
Creates an a
Cisco UCS Mi
for remote offi
branch offices

nect
chnology
Connect LAN, SAN, and management networks and physical and virtual servers with one physical connection

Cisco UCS B-Series Blade Servers

Cisco UCS B200 M3 and M4

Cisco UCS B420 M3 and M4

Enterprise Class

Servers

Cisco UCS B260 M4

Cisco UCS C460 M4

Mission Critical

Cisco UCS Virtual Interface Card (VIC) 1225

Cisco UCS C240 M3 and M4

Cisco UCS C220 M3 and M4

Cisco UCS C3160

Enterprise Class

320 servers in a single rack

Cisco UCS M-Series Modular Servers

Cisco UCS M4308 Modular Chassis

Cisco UCS M142 Compute Cartridge

Cisco UCS Invicta™ Series

Cisco UCS Invicta C710SR Routers

Cisco UCS Invicta C3124SN Nodes

Cisco UCS Invicta Scaling System

Cisco UCS Invicta C3124SA Appliance

Solid-State Application Acceleration

I was planning on a ~20 minute talk

Then Guillaume told me I had to take an hour (!)

So you get two talks – for the price of one!

Talk 1

Thoughts on fixing problems with MPI_INIT and MPI_FINALIZE

A glimpse into the MPI Forum

Libfabric

Talk 2

Cisco’s journey from the legacy Verbs API to Libfabric

MPI_INIT and MPI_FINALIZE

Tales of Woe

Before MPI-3.1, this could be erroneous

```
int main(int argc, char **argv) {
    MPI_Init_thread(…, MPI_THREAD_FUNNELED, …);
    pthread_create(…, my_thread1_main, NULL);
    pthread_create(…, my_thread2_main, NULL);
    // …
}
```

```
int my_thread1_main(void *context) {
    MPI_Initialized(&flag);
    // …
}
```

```
int my_thread2_main(void *context) {
    MPI_Initialized(&flag);
    // …
}
```

Before MPI-3.1, this could be erroneous

```
int main(int argc, char **argv) {
    MPI_Init_thread(…, MPI_THREAD_FUNNELED, …);
    pthread_create(…, my_thread1_main, NULL);
    pthread_create(…, my_thread2_main, NULL);
    // …
}
```

```
int my_thread1_main(void *context) {
    MPI_Initialized(&flag);
    // …
}
```

These might run at the same time (!)

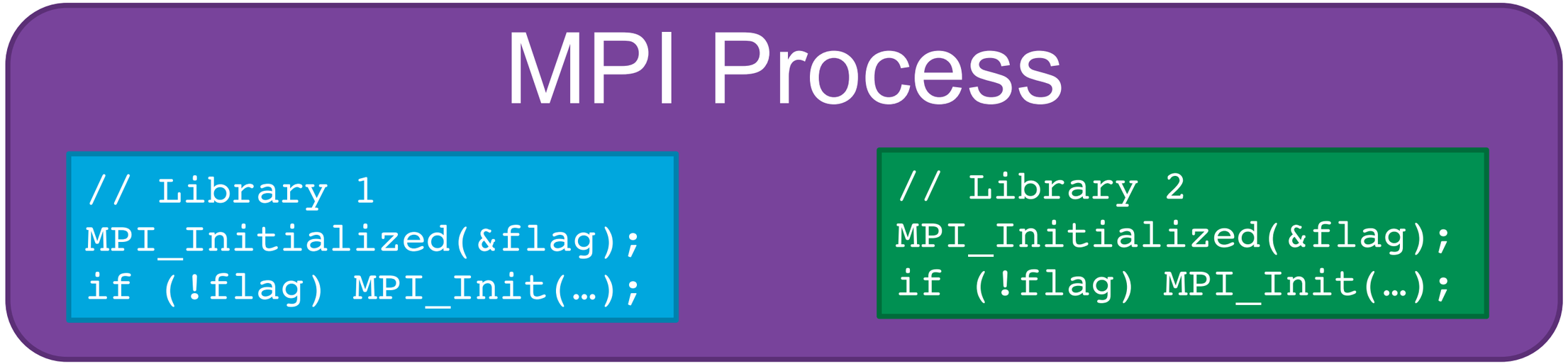
```
int my_thread2_main(void *context) {
    MPI_Initialized(&flag);
    // …
}
```

The MPI-3.1 solution

- MPI_INITIALIZED (and friends) are allowed to be called at any time
  - …even by multiple threads
  - …regardless of MPI_THREAD_* level
- This is a simple, easy-to-explain solution
  - And probably what most applications do, anyway ☺
- But many other paths were investigated

MPI_INIT / FINALIZE limitations

- Cannot call MPI_INIT more than once
- Cannot set error behavior of MPI_INIT
- Cannot re-initialize MPI after it has been finalized
- Cannot init MPI from different entities within a process without a priori knowledge / coordination



MPI_INIT / FINALIZE limitations

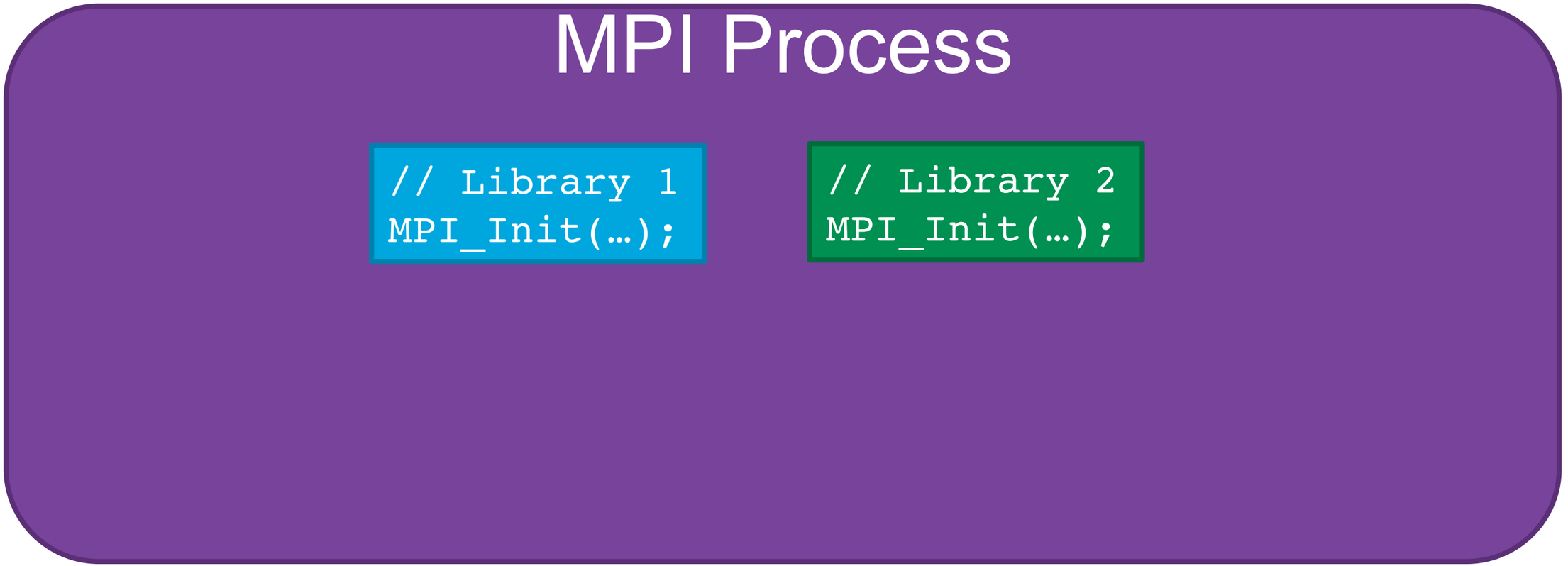
- Cannot call MPI_INIT more than once
- Cannot set error behavior of MPI_INIT
- Cannot re-initialize MPI after it has been finalized
- Cannot init MPI from different entities within a process without a priori knowledge / coordination

1994 called.

They want their API design back.

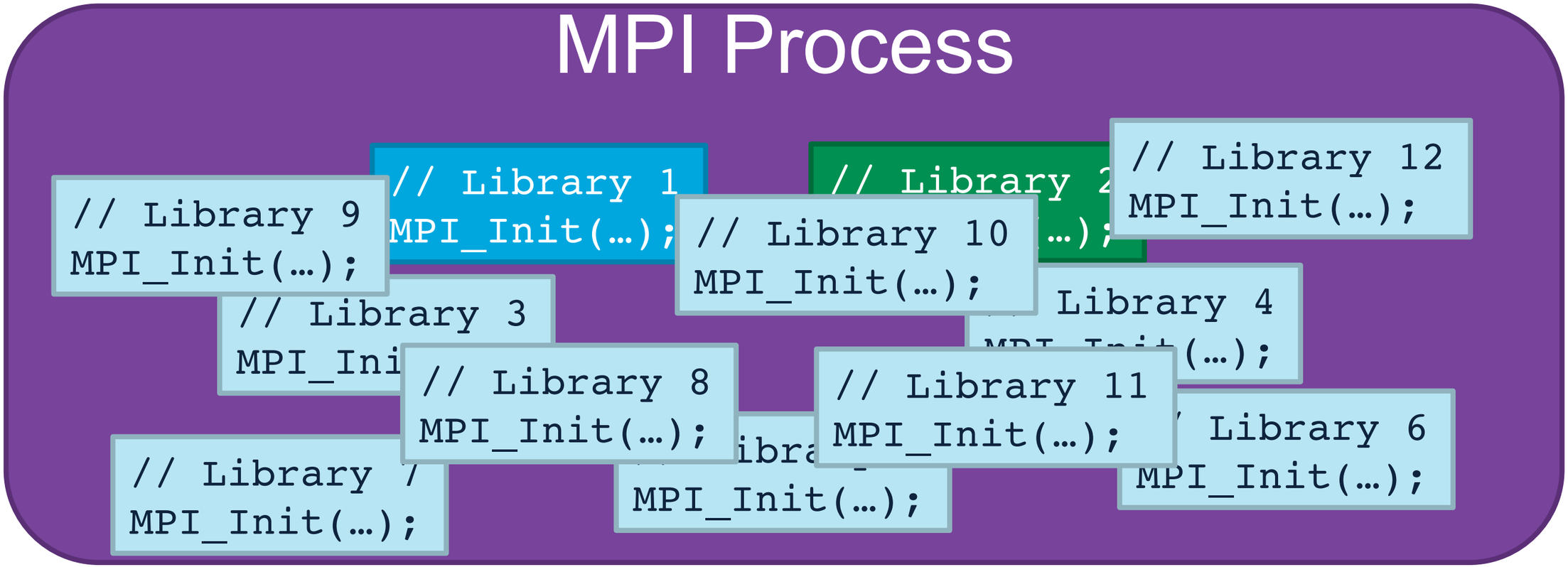
What we should have

- Call MPI_INIT as many times as you like
- By whomever wants to call it



What we should have

- Call MPI_INIT as many times as you like
- By whomever wants to call it



…but that has its own complications

Do you have to call MPI_FINALIZE exactly that many times?

Do you allow MPI_INIT after MPI_FINALIZE?

Or perhaps you only allow MPI_INIT before MPI has been finalized?

How can you tell if it's safe to call MPI_INIT? Atomic "test-and-init"?

We need something new

WARNING!

The following are just (incomplete) crazy ideas

New MPI concept: a session

```
int main(int argc, char **argv) {
    pthread_create(…, my_thread1_main, NULL);
    pthread_create(…, my_thread2_main, NULL);
    …
}
```

```
int my_thread1_main(void *context) {
    MPI_Session session;
    MPI_Session_create(…, &se

    // Do MPI things

    MPI_Session_free(&session
}
```

```
int my_thread2_main(void *context) {
    MPI_Session session;
    MPI_Session_create(…, &session);

    // Do MPI things

    MPI_Session_free(&session);
}
```

New MPI concept: a session

Create communicators from sessions

```
int my_thread1_main(void *context) {
    MPI_Session session;
    MPI_Session_create(&session);
    MPI_Comm_create_from_session(session, &comm)

    // Do MPI thin

    MPI_Comm_free(
    MPI_Session_fr
}
```

```
int my_thread1_main(void *context) {
    MPI_Session session;
    MPI_Session_create(&session);
    MPI_Comm_create_from_session(session, &comm)

    // Do MPI things with comm

    MPI_Comm_free(&comm);
    MPI_Session_free(&session);
}
```

Problems that sessions solve

Each entity (library?) in an OS process can have its own session

Any session-local state can be encapsulated in the handle

Entities can create / destroy sessions at any time
…in any thread

…but what about MPI_COMM_WORLD?

MPI_COMM_WORLD. Sigh.

- When is MPI_COMM_WORLD created (and/or initialized)?
- When is MPI_COMM_WORLD destroyed?
- Can you use MPI_COMM_WORLD with any session?

→ There doesn't seem to be an obvious relation between MCW and individual sessions (ditto for MPI_COMM_SELF)

What if we get rid of MPI_COMM_WORLD?

Problems that solves

- Addresses logical inconsistency with session concept
- Clean separation of communicators between sub-entities

  …maybe *slightly* better than we have it today (sub-entities dup'ing COMM_WORLD)

- Side effects:

  Fault tolerance issues become easier

  Opens some possibilities for scalability improvements

Problems that creates

- Users will riot

…but what if they don’t?

Open questions

- What would be the forward / backward compatibility strategy?

  E.g., deprecate INIT, FINALIZE, INITIALIZED, FINALIZED…?

- What are the other arguments to MPI_SESSION_CREATE?
- Can you call both MPI_INIT and MPI_SESSION_CREATE in the same process?
- Can you do anything else with a session?

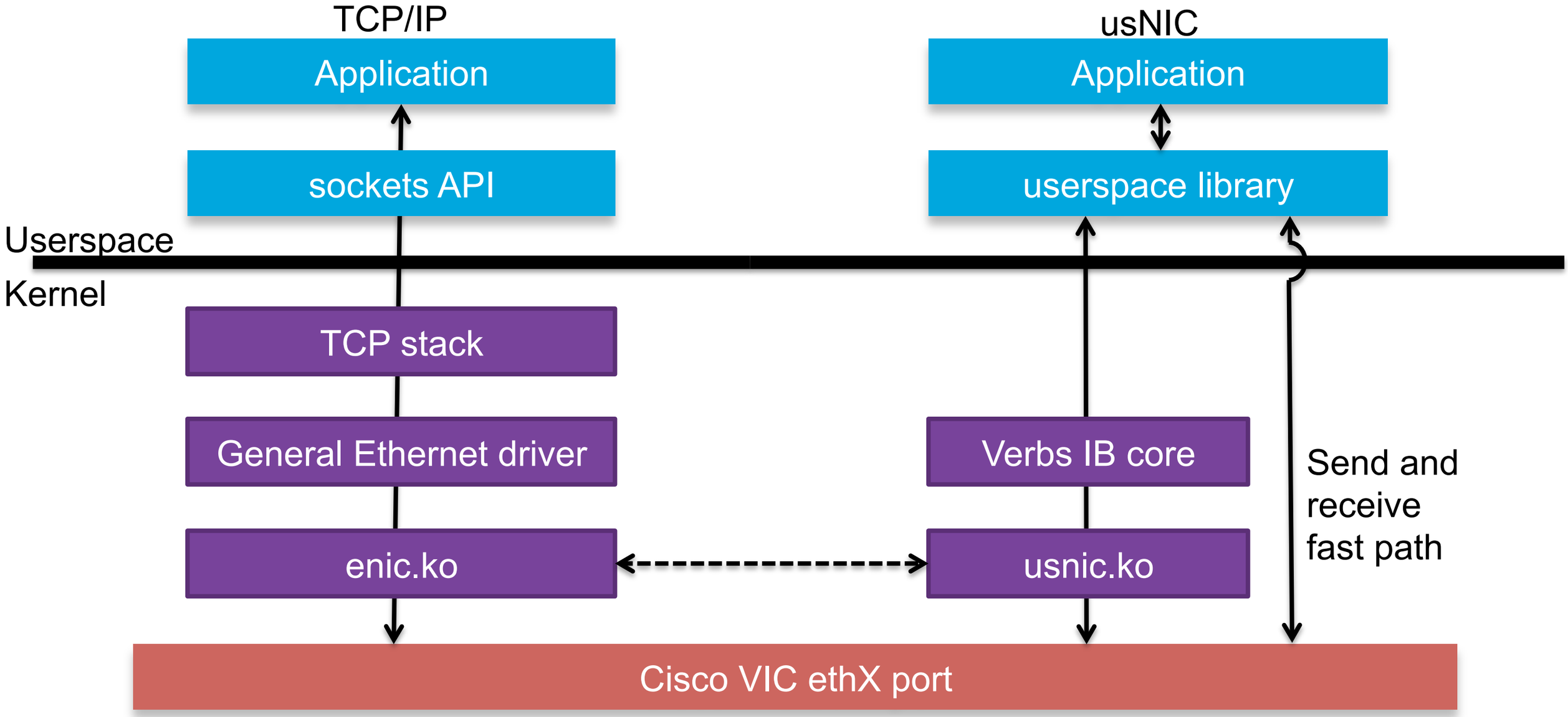
Sooo… what happens next?

Come to the MPI Forum meeting

Discuss this and other scintillating MPI topics

Cisco’s journey from Verbs to Libfabric

Cisco usNIC: OS bypass to the same ethX interface



Verbs is a fine API.

. . . if you make InfiniBand hardware.

libfabric thing

(see libfabric.org community for details)

Which API should be our way forward for kernel bypass?

Keep in mind, Cisco already supports UD Verbs

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

IBV_MTU_256
IBV_MTU_512
IBV_MTU_1024 ← 1500
IBV_MTU_2048
IBV_MTU_4096 ← 9000

Comparison: MTU

Libfabric

- Integer (not enum) endpoint attribute

Comparison: MTU

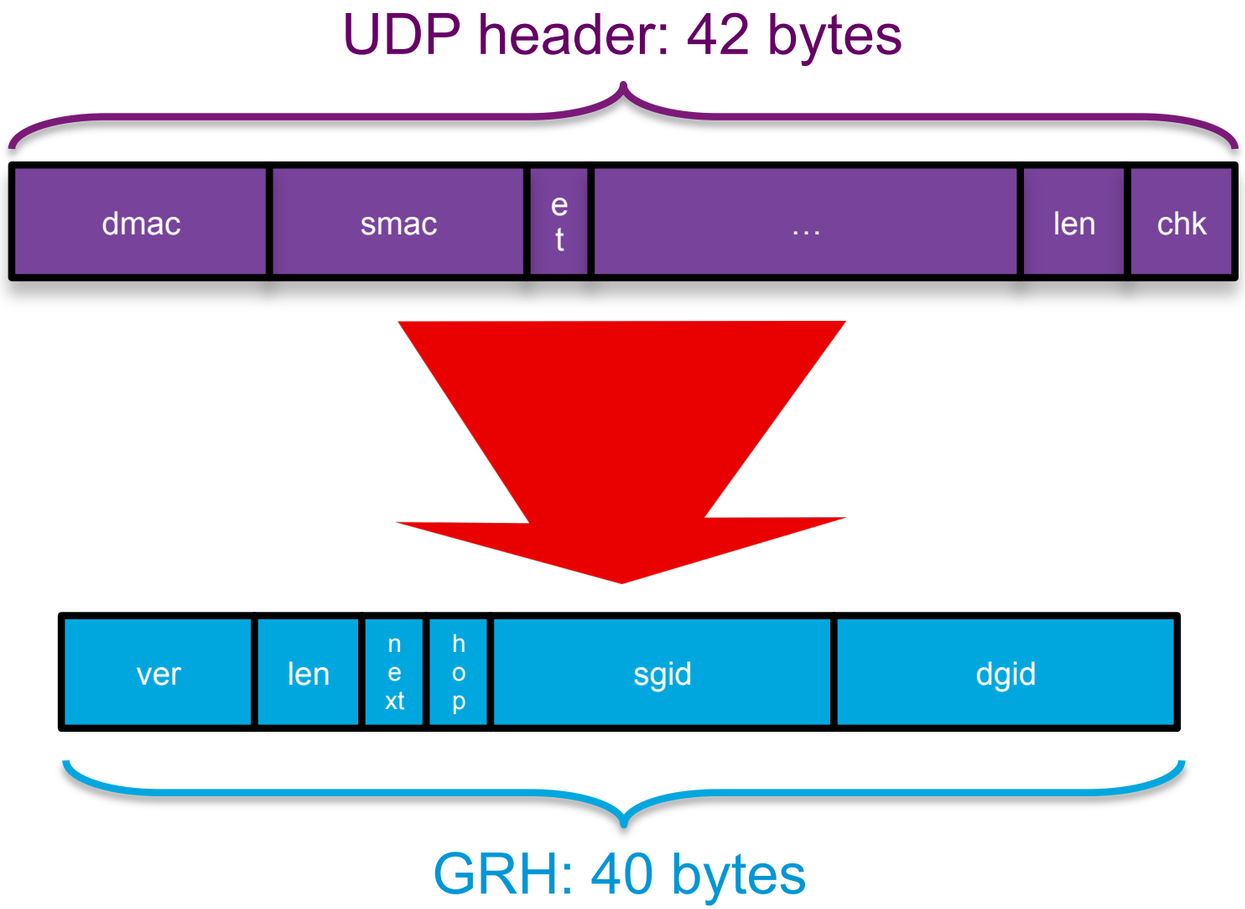
Libfabric

- Integer (not enum) endpoint attribute

Comparison: Unreliable datagram

Verbs

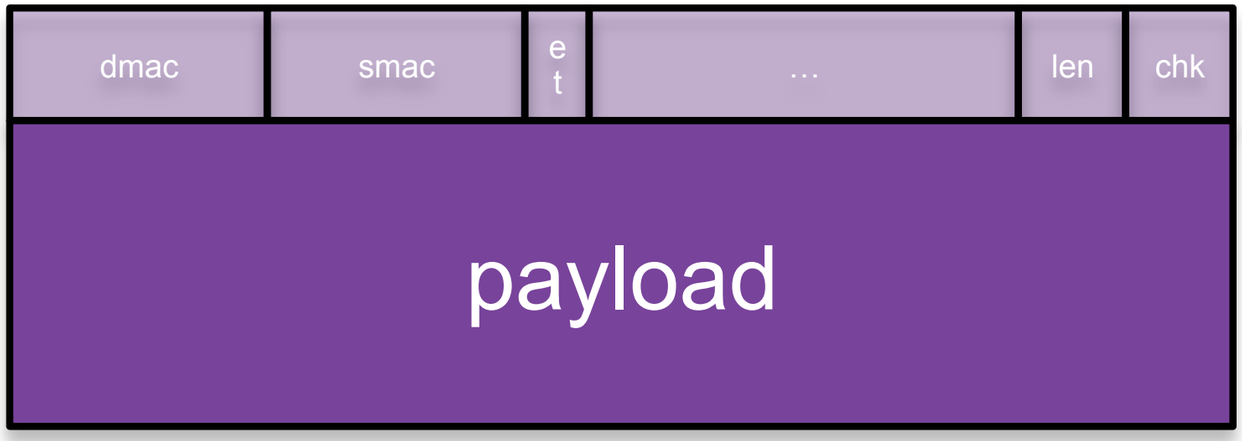
- Mandatory GRH structure
  - InfiniBand-specific header
- 40 bytes
  - UDP header is 42 bytes
  - …and a different format
- Breaks ib_ud_pingpong
- usnic verbs provider used “magic” ibv_port_query() to return extensions pointers
  - E.g., enable 42-byte UDP mode



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

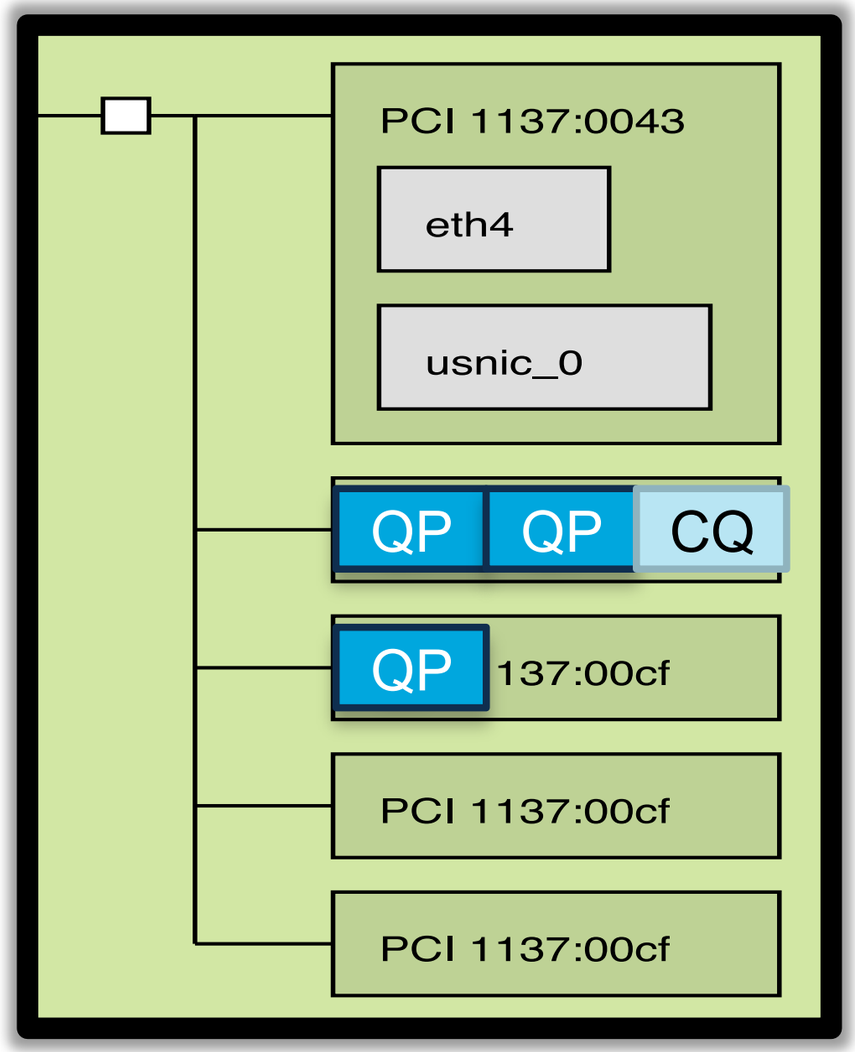
DONE

Comparison: 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

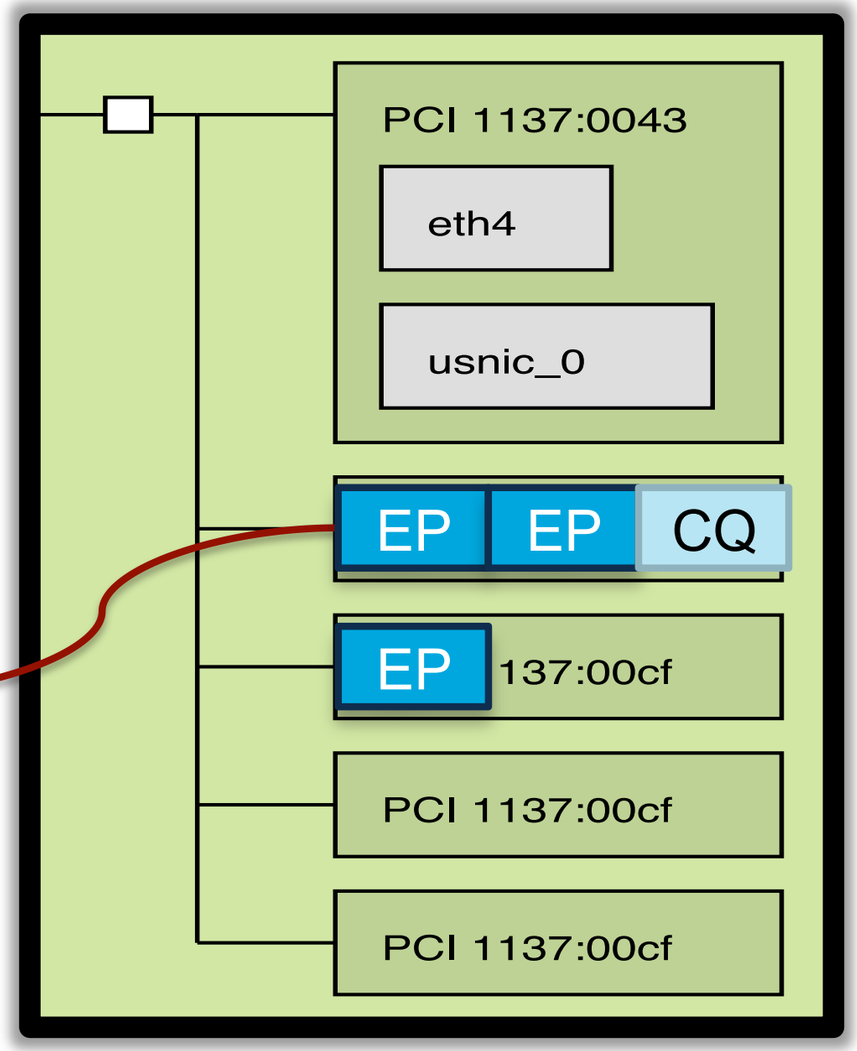
Libfabric

- Maps nicely to SR-IOV
- Fabric → PCI physical function (PF)
- Domain → PCI 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

DOOM

Everything is awesome

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

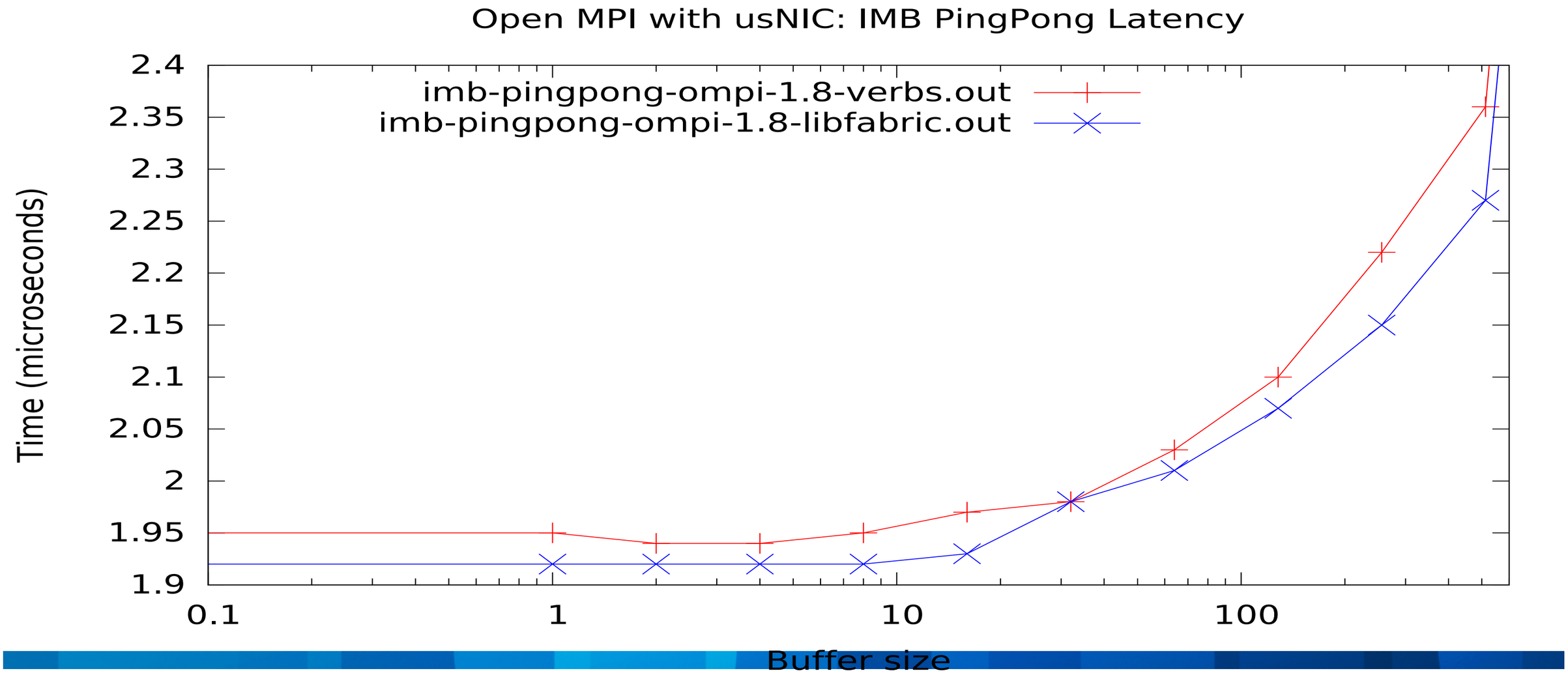
  fi_send(buffer, …)

- Variable-length prefix receive

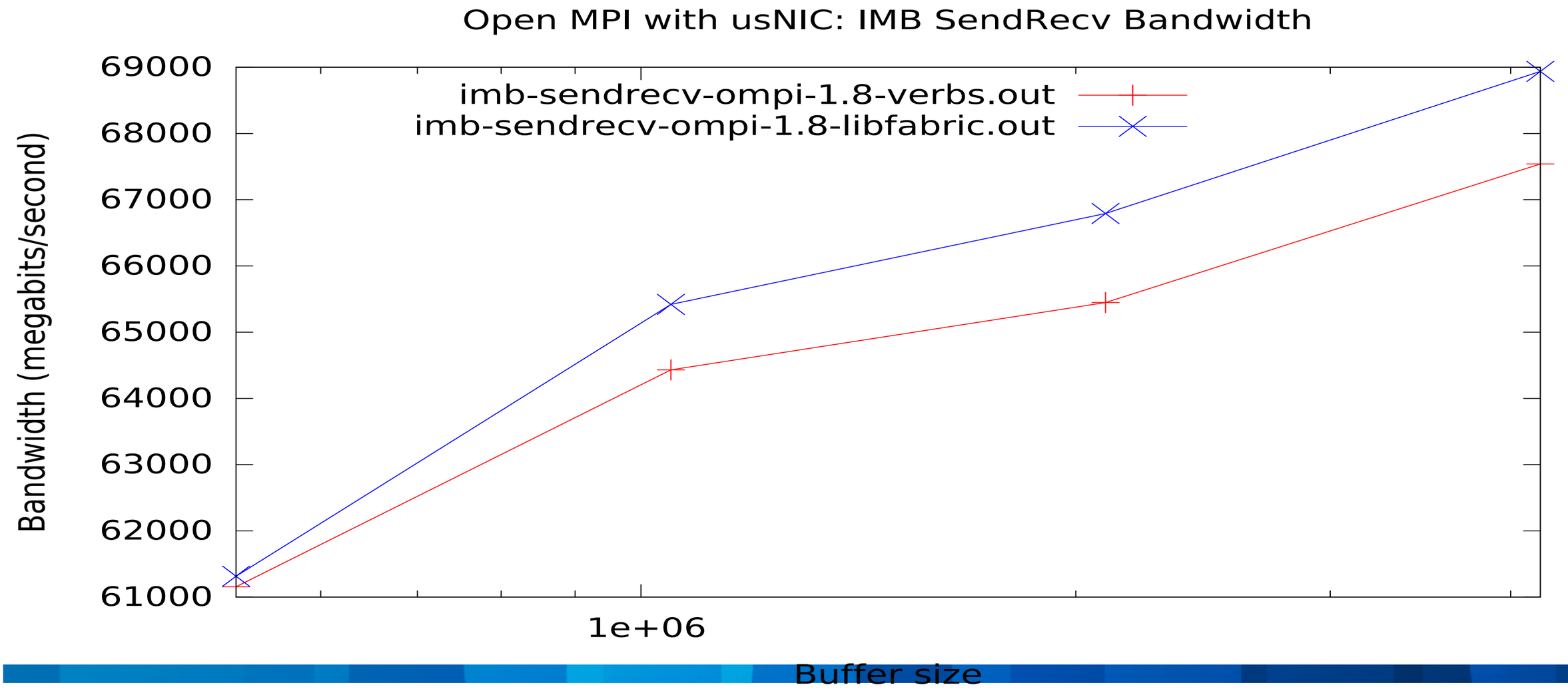
  Provider-specific

- Fewer branches in completions

libfabric performance vs. Linux verbs



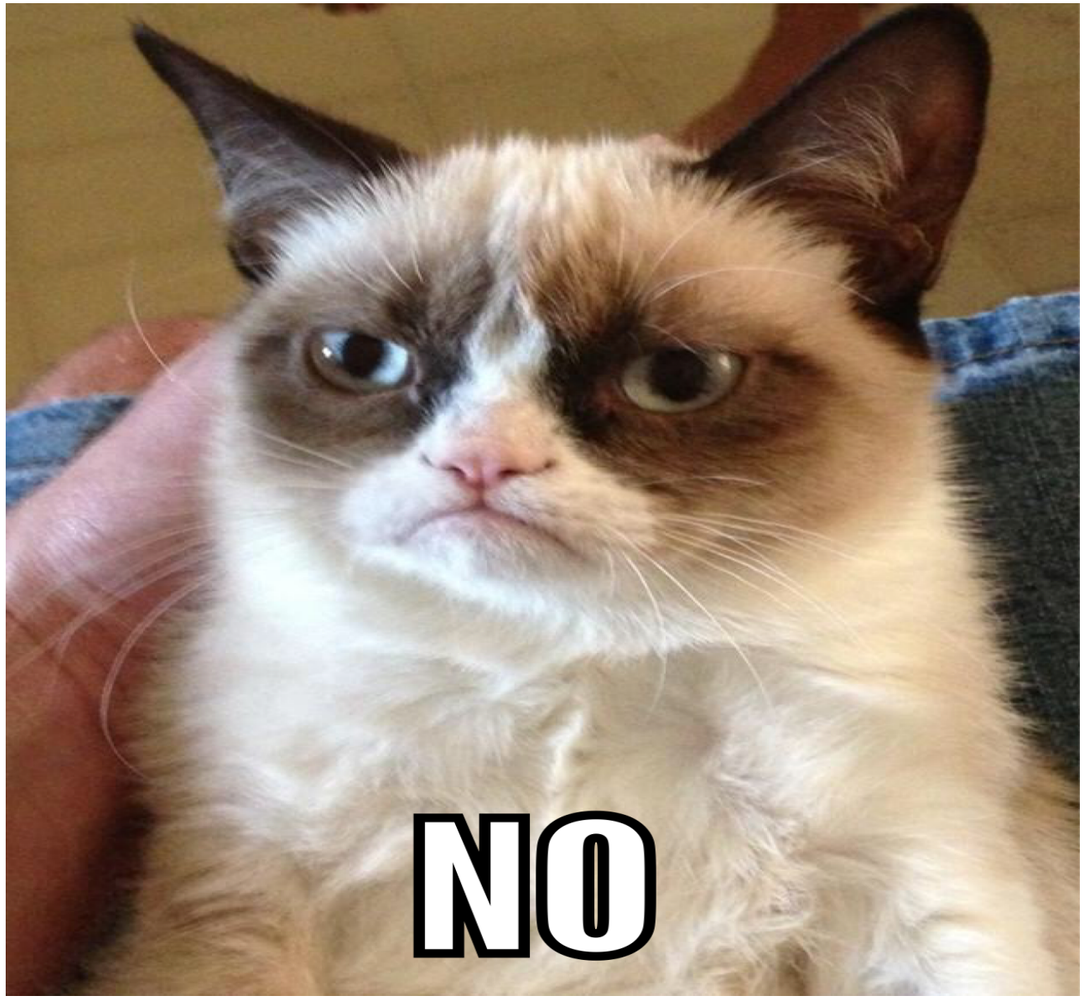
libfabric performance vs. Linux verbs



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
- Conclusion: possible (obviously), but not preferable

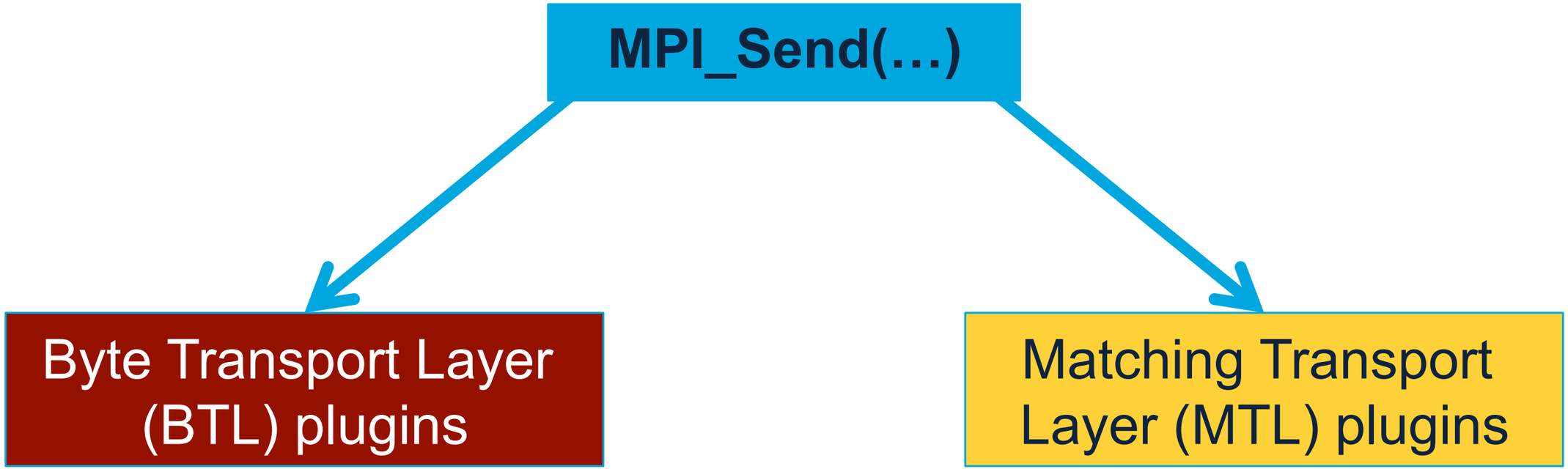
Libfabric

- Whole API designed with multiple vendor hardware models in mind
- Much easier to match our hardware to core Libfabric concepts
- Conclusion: much more preferable than verbs

Ok, so let’s do libfabric!

Does it play well with MPI?

Open MPI has two major types of transports



BTL

- Inherently multi-device
  - Round-robin for small messages
  - Striping for large messages
- Major protocol decisions and MPI message matching driven by an Open MPI engine

Byte Transport Layer (BTL) plugins

MTL

- Most details hidden by network API
  - MXM
  - Portals
  - PSM
- As a side effect, must handle:
  - Process loopback
  - Server loopback (usually via shared memory)

Matching Transport Layer (MTL) plugins

BTL and MTL plugins

Byte Transport Layer (BTL) plugins

- IB / iWarp (verbs)
- Portals
- SCIF
- Shared memory
- TCP
- uGNI
- usNIC (verbs)

Matching Transport Layer (MTL) plugins

- MXM
- Portals
- PSM
- PSM2

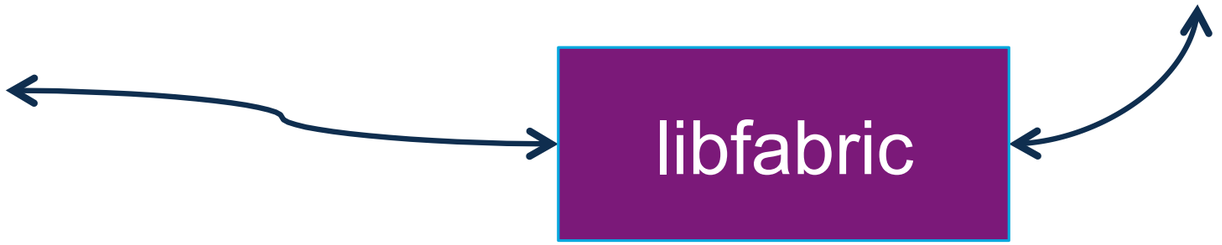
Now featuring 200% more libfabric

Byte Transport Layer (BTL) plugins

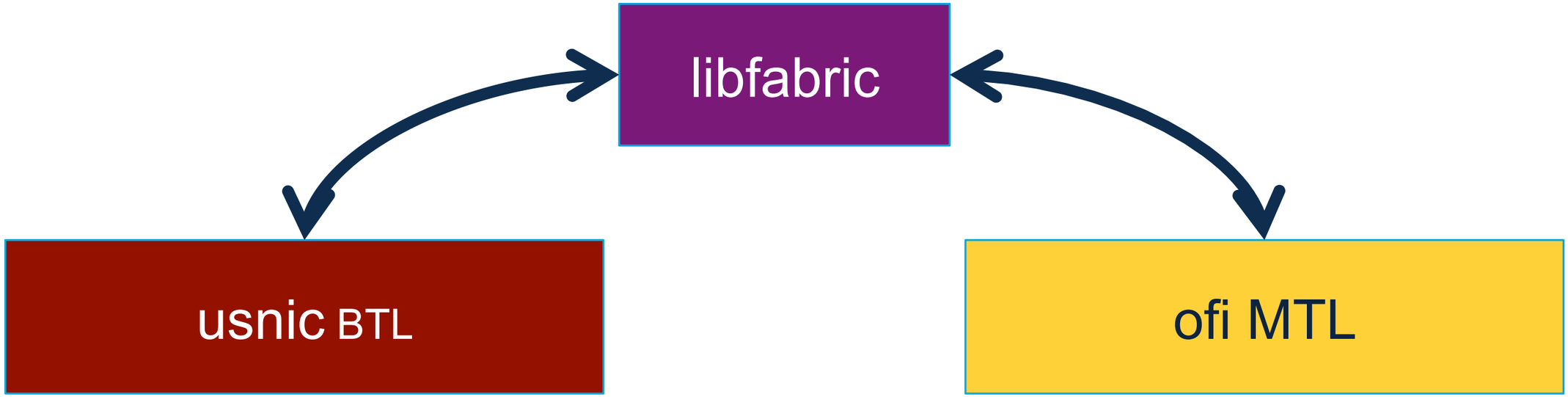
- IB / iWarp (verbs)
- Portals
- SCIF
- Shared memory
- TCP
- uGNI
- usNIC

Matching Transport Layer (MTL) plugins

- MXM
- Portals
- PSM
- PSM2
- ofi



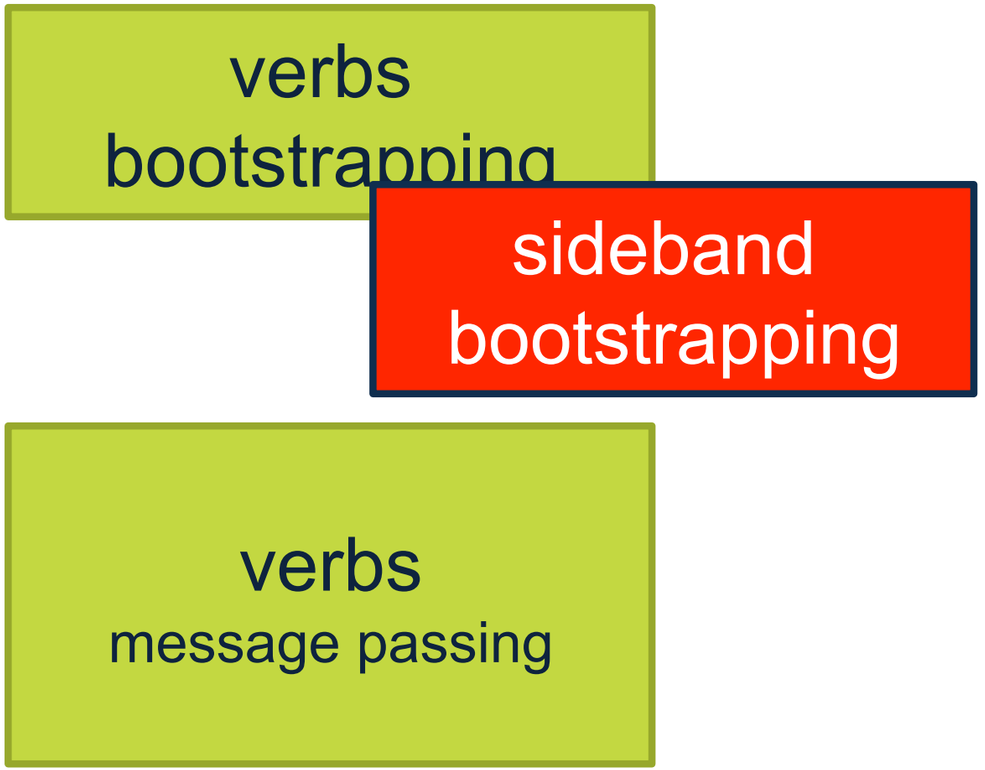
Libfabric-based plugins



- Cisco developed
- usNIC-specific
- OFI point-to-point / UD
- Tested with usNIC

- Intel developed
- Provider neutral
- OFI tag matching
- Tested with PSM / PSM2

First experiment
usnic BTL: verbs → libfabric



1. Find the corresponding ethX device
2. Obtain MTU
3. Open usNIC-specific configuration options

First experiment
usnic BTL: verbs → libfabric

Bootstrapping sequence totally different

No sideband bootstrapping

~1:1 swap of verbs → libfabric calls

Second experiment
Two different libfabric usage models

usnic BTL

- For a specific provider
  - Ask fi_getinfo() for prov_name="usnic"
- Use usNIC extensions
  - Netmask, link speed, IP device name, etc.
- usNIC-specific error messages

ofi MTL

- For any tag-matching provider
- No extension use
  - 100% portable
- Generic error messages

Second experiment
Two different libfabric usage models

usnic BTL

- For a specific provider
  - Ask fi_g… for prov_n…ic”
- Use us…nsio…
  - Netmas… name, …
- usNIC-s… messag…

ofi MTL

- Any tag-matching …rovider
- No extension use
  - 100% portable
- Generic error messages

100% libfabric goodness

Summary

- Libfabric is the Way Forward for Cisco
  - Open community
  - Matches our hardware
  - Performance benefits
  - Features benefits
- Libfabric matches MPI
  - Has features MPI has been asking for… for years
  - Optimistic about its future
  - (come join us!)

Thank you.