OPENFABRICS ALLIANCE

2022 OFA Virtual Workshop

IN-NETWORK COLLECTIVE COMMUNICATION ACCELERATIONS

OFI COLLECTIVES

Sean Hefty

Intel Corporation

April 2022

OVERVIEW

Introduction:

How collectives differ
Collective operations

Software abstraction:

Identify collective membership
Setup communication groups
Invoke collective

Other thoughts:

Ensure efficient mappings

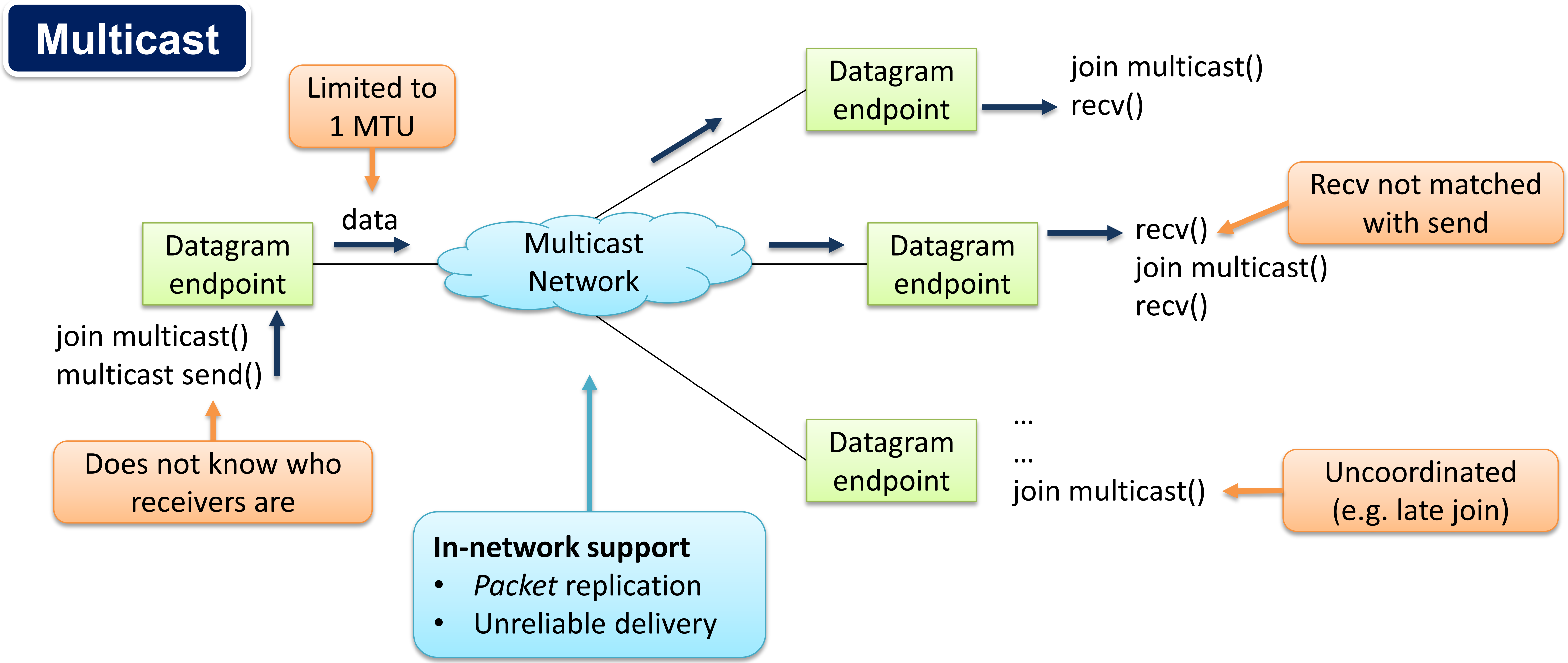
Focus on enabling the technology

Accelerations in switches, NICs, platforms, FPGAs

No discussion on effectiveness

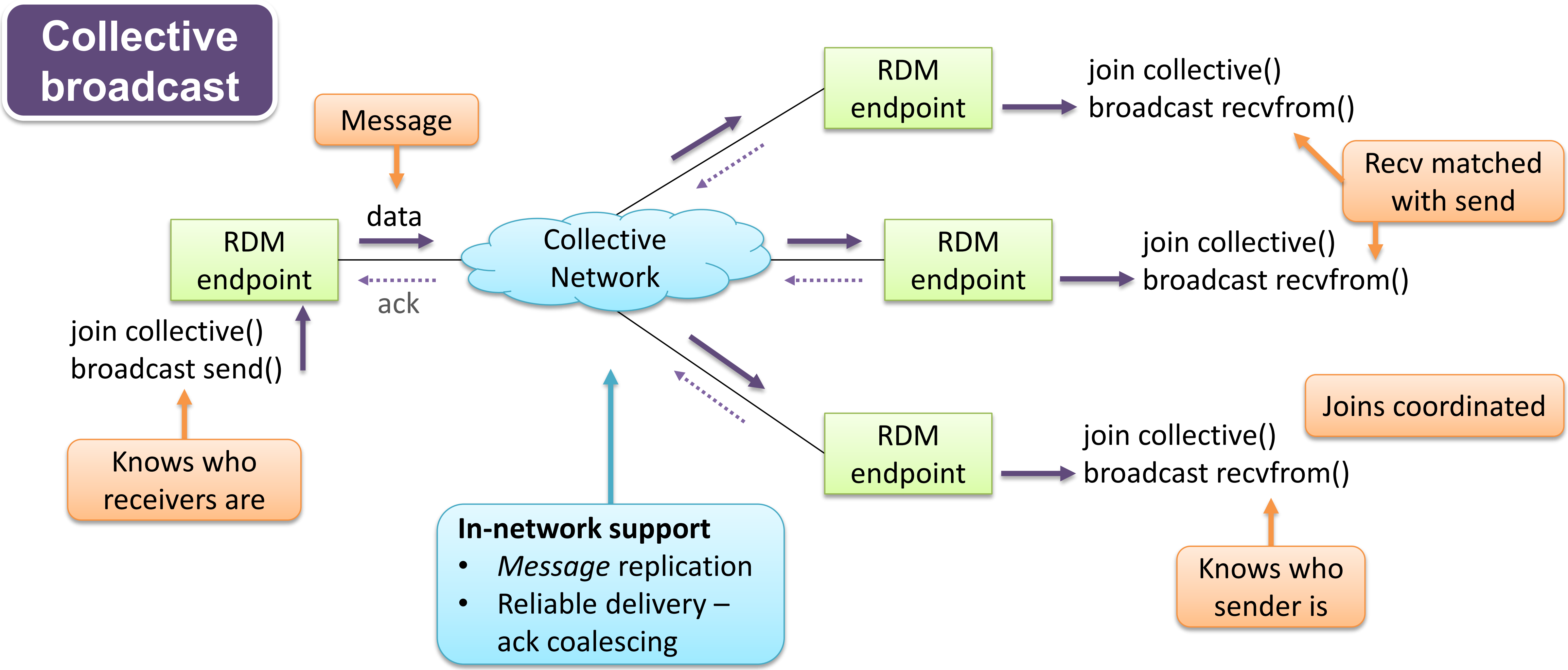
HOW COLLECTIVES DIFFER

MULTICAST VS COLLECTIVE BROADCAST



HOW COLLECTIVES DIFFER

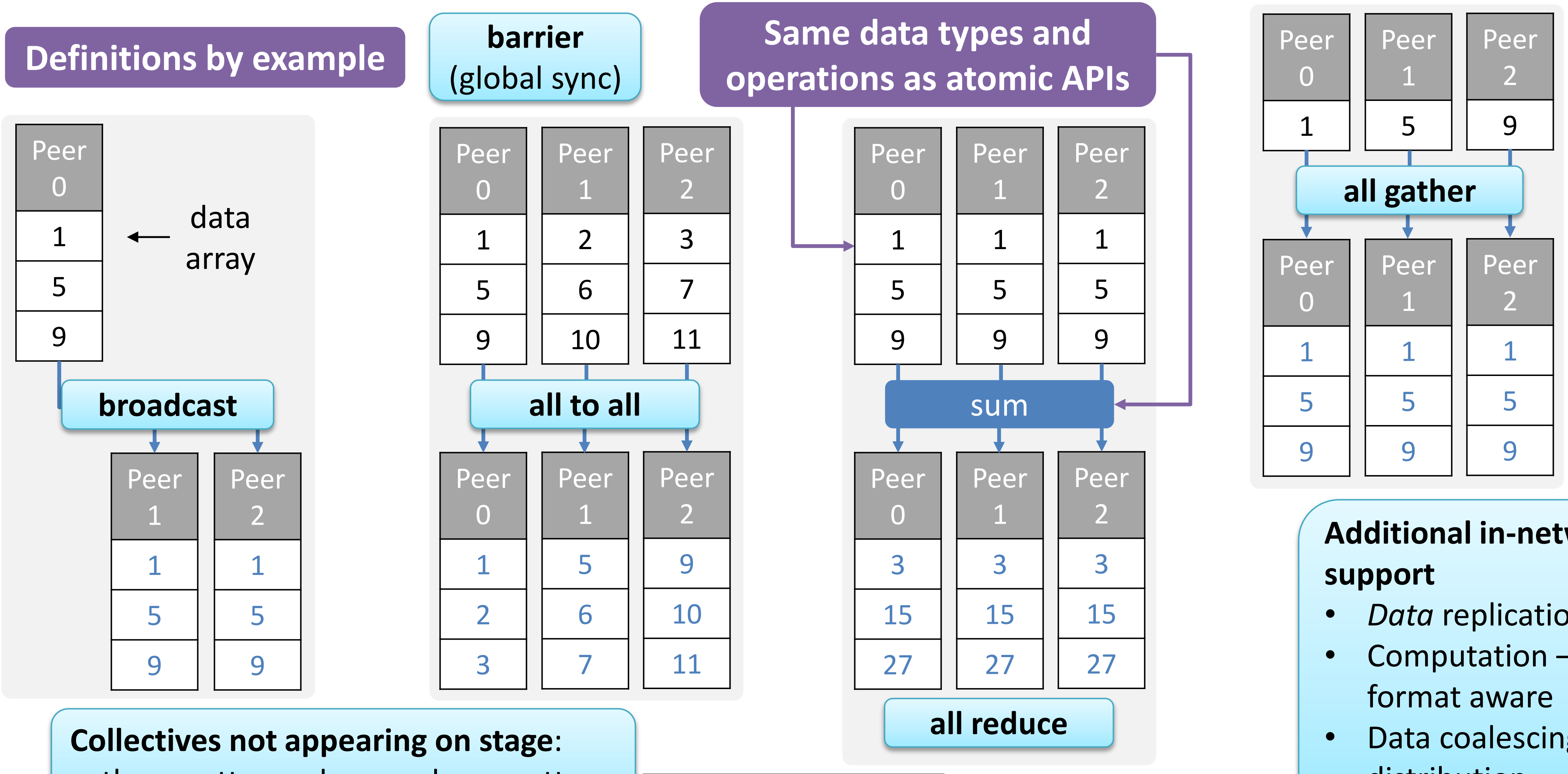
MULTICAST VS COLLECTIVE BROADCAST



RDM – Reliable Datagram

COLLECTIVE OPERATIONS

CONCEPTUAL: “MULTICAST ATOMICS”



Collectives not appearing on stage: gather, scatter, reduce, reduce-scatter

Additional in-network support

- *Data* replication
- Computation – data format aware
- Data coalescing and distribution

SOFTWARE ABSTRACTION
LIBFABRIC COLLECTIVE API'S

1. **Identify collective membership**
   - Select participating peers
   - Local operation – address vector sets
2. **Setup communication groups**
   - Coordinated join among members
   - Network operation (maybe) – 2 supported models
3. **Invoke collective**
   - Collective data transfer operation

IDENTIFY COLLECTIVE MEMBERSHIP
ADDRESS VECTOR SETS

Existing API

AV
Represents the peer universe

Application visible addresses of peer endpoints

Translated addresses

Unicast address used by API

e.g.: IP : Port
10.0.0.1 : 7000
10.0.0.1 : 7001
10.0.0.2 : 7000
10.0.0.3 : 7003
…

map

Address Vector	
fi_addr_t	**Fabric Address**
0	100:3:50
1	100:3:51
2	101:3:83
3	102:3:64
…	…

Collective Extension

Local setup to identify members for new group

AV Set
Set of peers in a collective group

Associated collective address

select

Address Vector Set
fi_addr_t
0
2
4
6
…

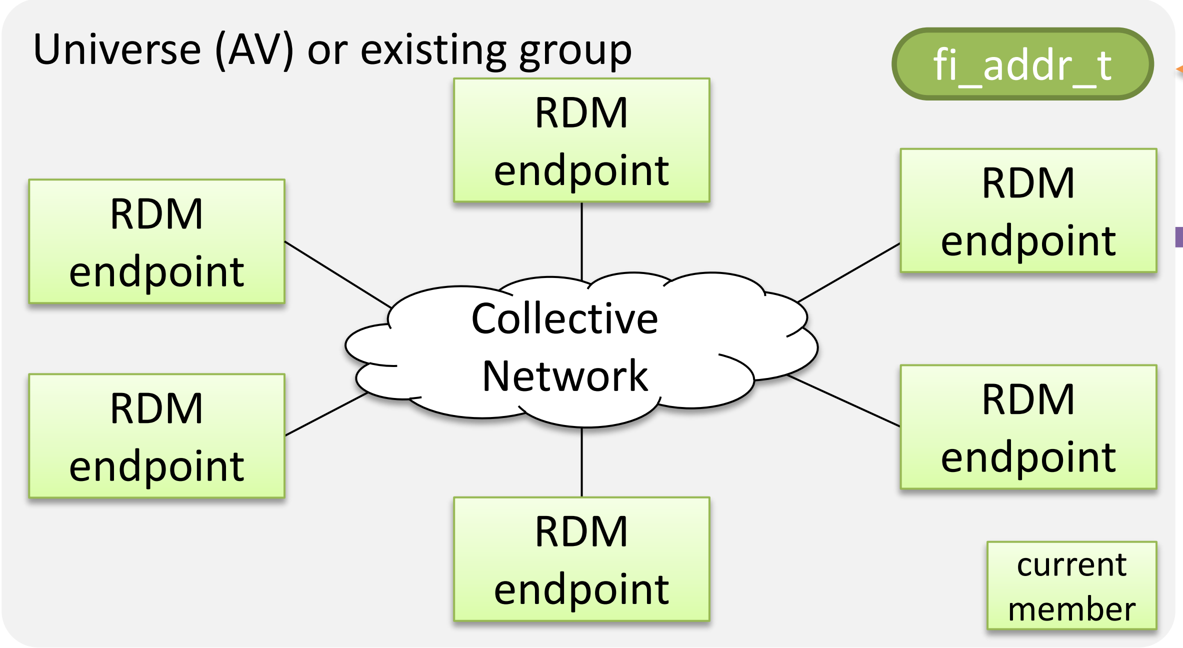
fi_addr_t

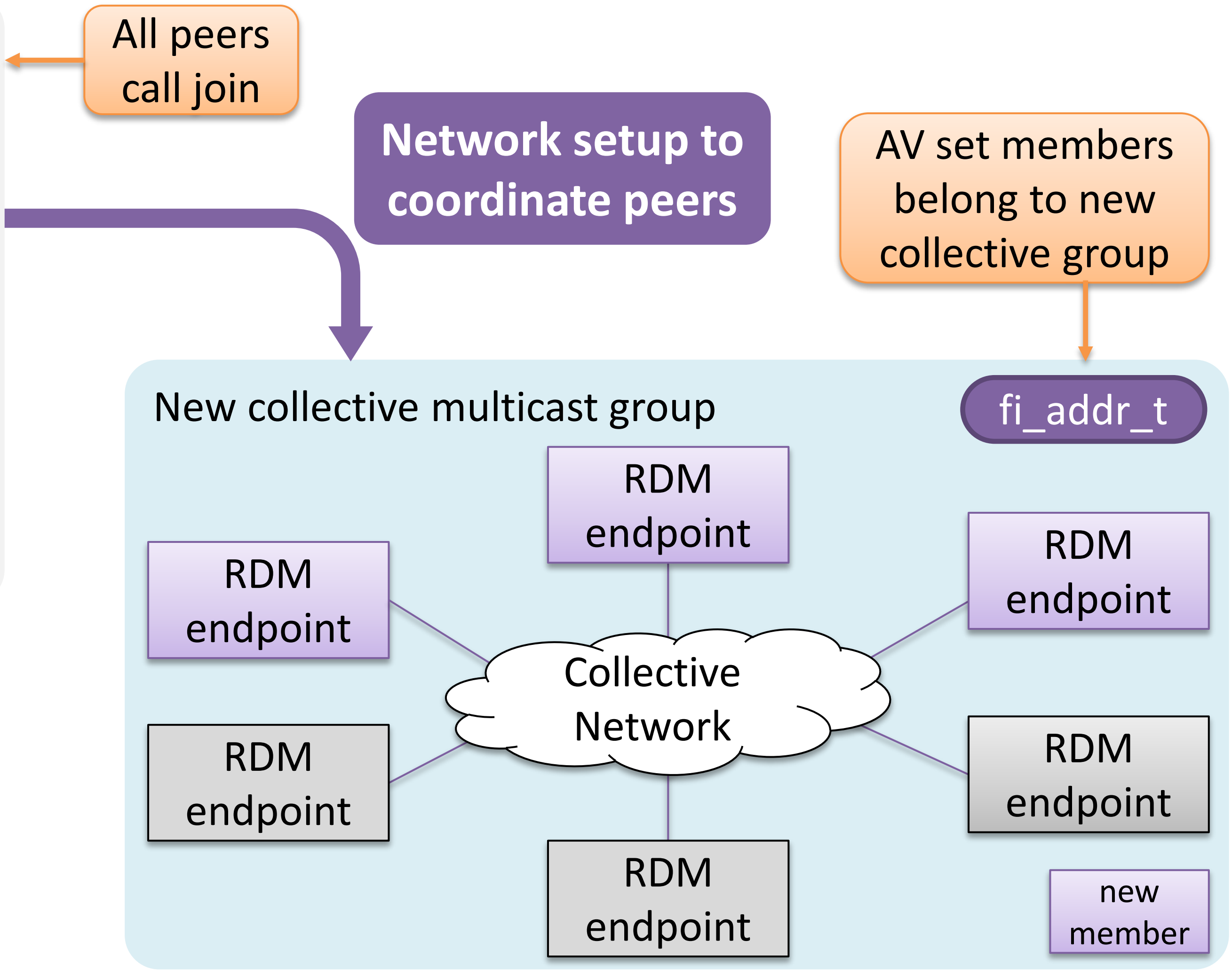
AV Set operations
- Insert
- Remove
- Union
- Intersect
- Difference

AV – Address Vector

SETUP COMMUNICATION GROUPS

JOIN COLLECTIVE





Join Models

- ***OFI-managed***
  - Provider implements coordination in-band
  - Provider reserves resources
- ***Application-managed***
  - Group was established out-of-band
  - Resources already reserved

INVOKE COLLECTIVE

SAMPLE API FLOW

```
struct fi_info *hints, *info;

hints fi_allocinfo();
<format hints>
hints->caps |= FI_COLLECTIVE;

fi_getinfo(FI_VERSION(1,14), hostname, NULL, FI_SOURCE, hints, &info);

<allocate fabric resources>

struct fi_collective_attr attr = {0};
attr.op = FI_SUM;
attr.datatype = FI_FLOAT;

fi_query_collective(domain, FI_ALLREDUCE, &attr, 0);
assert(attr.datatype_attr.count >= 100 && attr.max_members >= 50)
```

Request support for in-network collectives

Verify support for collective that we need

INVOKE COLLECTIVE

SAMPLE API FLOW

```
struct fid_av_set *av_set;
struct fi_av_set_attr attr = {0};
attr.start_addr = 2;
attr.end_addr = 100;
attr.stride = 2;
fi_av_set(av, &attr, &av_set, NULL);
```

Create AV set and identify group members

First join involves all peers

```
struct fid_mc *mc;
fi_join_collective(ep, FI_ADDR_UNAVAIL, av_set, 0, &mc, NULL);
```

Creates collective multicast group

```
struct fi_eq_entry entry;
uint32_t event;
fi_eq_sread(eq, &event, &entry, sizeof(entry), -1, 0);
assert(event == FI_JOIN_COMPLETE);
```

Join completes asynchronously

```
fi_allreduce(ep, input_array, 100, NULL, result_array, NULL,
             fi_mc_addr(mc), FI_FLOAT, FI_SUM, 0, my_context);
```

Asynchronous all-reduce operation

OTHER THOUGHTS

ENSURE EFFICIENT MAPPINGS

Managing in-network resources

- Guarantee resources are available
- App may want to prioritize which collectives to accelerate
- **API object: collective resource tokens?**

Priority

- Define impact on active collectives
- Preempt possible? Pause-resume or abort/cancel?
- libfabric defines priority at the endpoint level
- **Do resource tokens act as a proxy?**

OTHER THOUGHTS
ENSURE EFFICIENT MAPPINGS

Reproducibility of results

- Order that data is fed into operations can produce different results
- Relaxed reproducibility can reduce in-network memory
- **Setting: per-operation, group (AV set), resource token?**

Sparse data

- Avoid sending / storing null data
- **Define a compact, data aware SGL?**

OTHER THOUGHTS
ENSURE EFFICIENT MAPPINGS

Network topology

- Query collective support - local vs global?
- Peer endpoints relative to switches and accelerators
- **Scope of the job or resource manager?**

Programmable in-network accelerations

- Non-collective operations
- How does app specify operation and parameters?
- Entity responsible for programming switch/FPGA?

THANK YOU