An OFI Communication Layer for the Chapel Runtime

Sung-Eun Choi
Principal Engineer
Cray R&D

Chapel Implementers and Users Workshop
2 June 2017

Portability

- **End users: Problem solved.**
  - Chapel is one of many options for portable parallel programming
    - Some are better than others ☺
- **Middleware now bears the responsibility**
  - Chapel (and other languages and libraries) use internal APIs to manage portability issues
  - Fewer people need to be experts
  - But still need to be an expert in a number of vendor options

OFI libfabric

- **OpenFabrics Interfaces Working Group (OFIWG) was formed in August 2013, chaired by Intel and Cray**
  - Open working group and open source development
    - Diverse set of experts from industry, government and academia
  - Input collected from HPC middleware developers
  - Enable best performance on any vendor hardware

Charter: *Develop an extensible, **open source** framework and interface aligned with **upper-layer protocols and applications** needs for **high-performance fabric** services.*

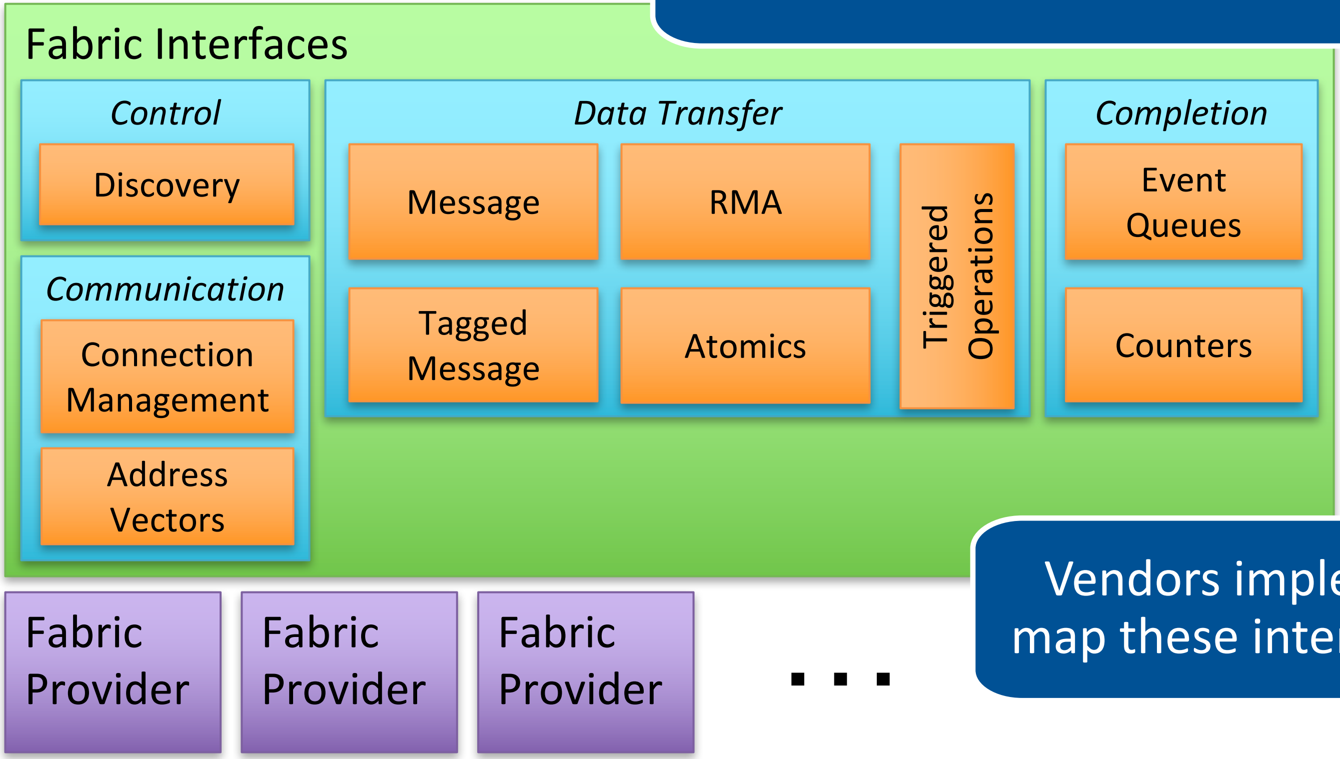
Result: libfabric

Outline

- Brief overview of libfabric
- Chapel ofi communication layer
- Lessons learned
- Status and conclusions

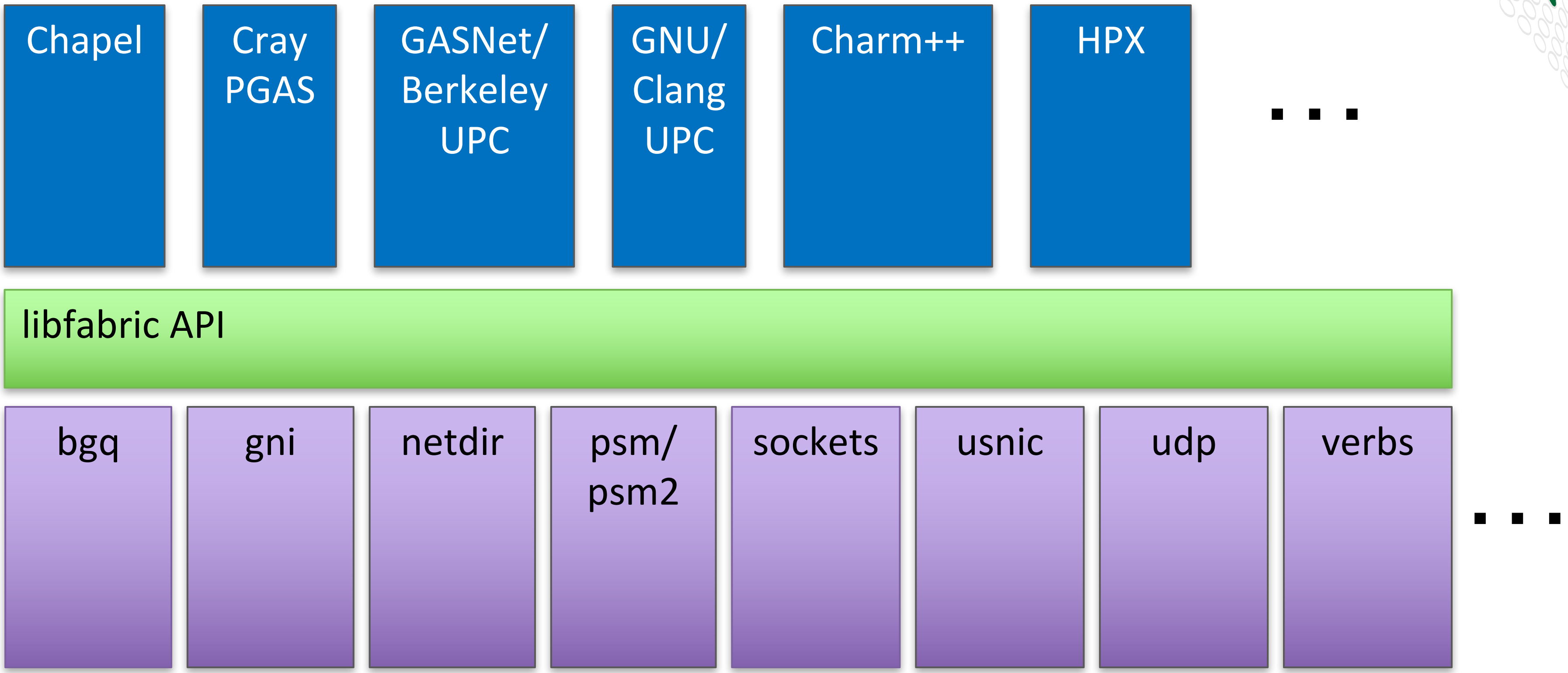
libfabric in a nutshell

Framework defines portable interfaces for HPC middleware



Vendors implement *providers* to map these interfaces to their fabric

libfabric in real life



The libfabric API

- **Control services**
  - Discovery of available providers and services
- **Communication services**
  - Connection and address management including *address vectors*
- **Data transfer services**
  - One-sided (RMA)
  - Two-sided (send/recv and tagged send/recv)
  - Atomic memory operations
  - Triggered operations

The libfabric API (cont.)

- **Completion services**
  - Completion queues (CQs) and counters for requested operations
  - Upon success…
    - indicates that source buffer can be reused (transmit)
    - returns result of data transfer operations (receive)
  - Upon failure…
    - returns error code

Other libfabric features

- Connected and unconnected endpoint types
- Thread safety options
- Data and control progress models
- Memory registration
- Extensible interface
- …

Unique features of libfabric

- **Dynamic provider selection**
  - Can use more than one provider in a single program
- **Providers are not required to implement the entire API**
  - May choose to omit functionality not available in hardware
  - Client and provider negotiate
- **API is portable, but may still want provider-specific code**
  - Provider-specific extensions
- **All data transfer calls are non-blocking**
  - Must use completion queues or counters (in most cases)

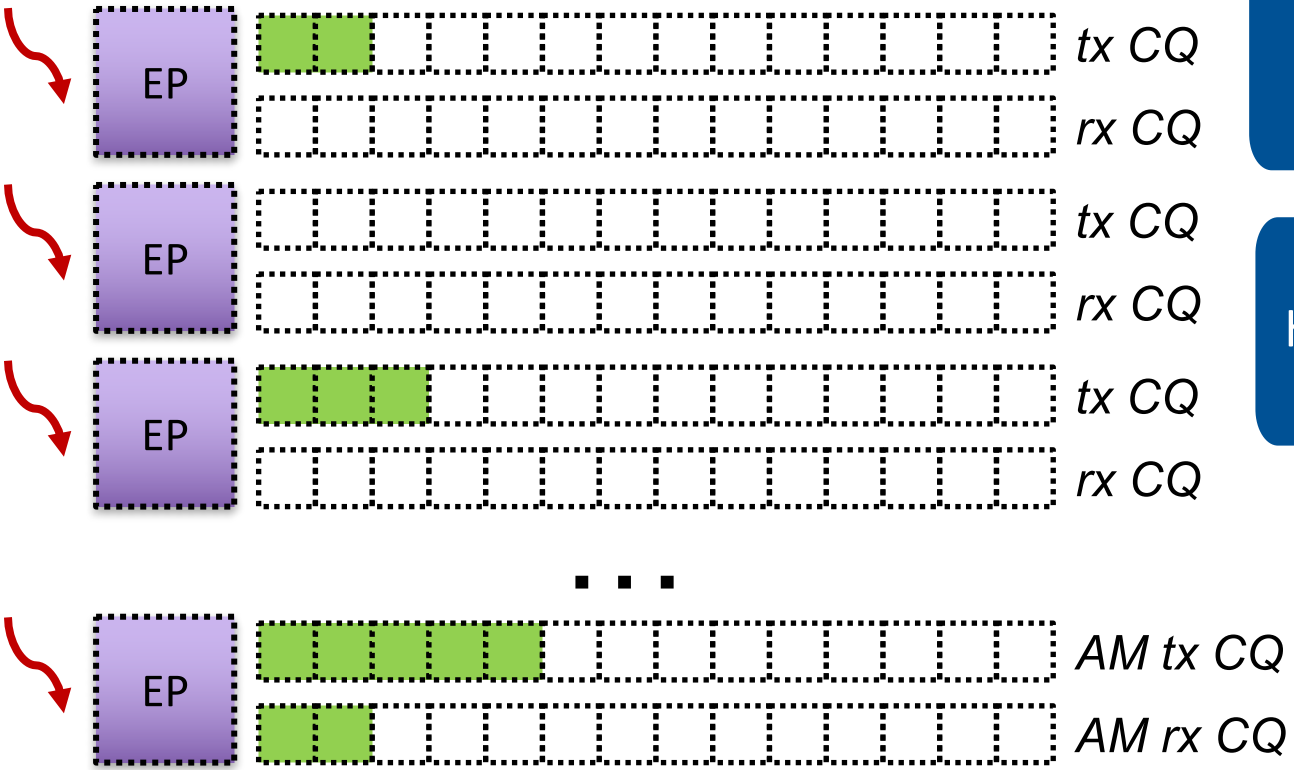
Outline

- Brief overview of libfabric
- Chapel ofi communication layer
- Lessons learned
- Status and conclusions

Chapel's communication layer

- **Compiler's interface to low level data transfer**
  - Initialization, global coordination and tear down
  - Data transfer operations (put and gets)
  - Active message interface (remote on statements)
- **Other stuff**
  - Progression
  - Interactions with the rest of the runtime
  - Comm layer diagnostics
  - Comm layer callbacks (e.g., for chplvis)

Comm layer design (sort of)



Each pthread has its own endpoint

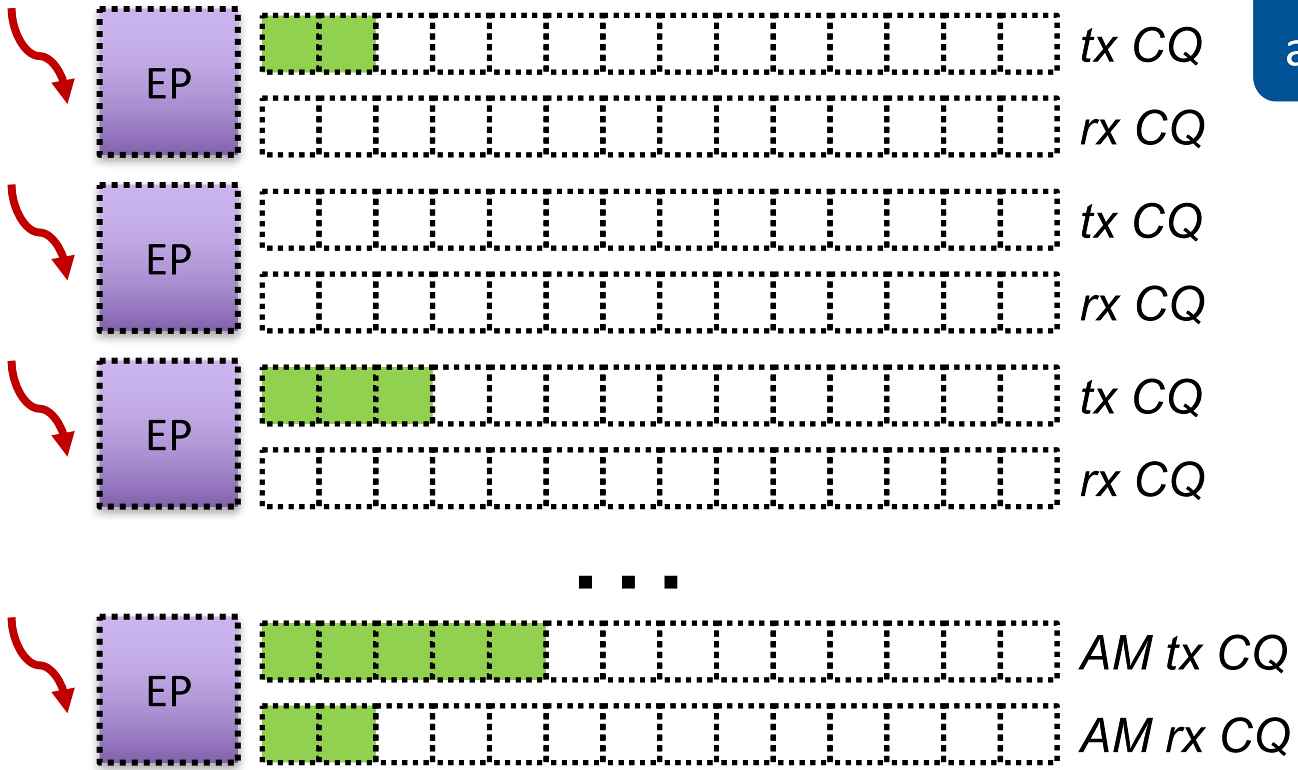
Each endpoint has transmit and receive CQs

The progress thread manages active messages

Progression

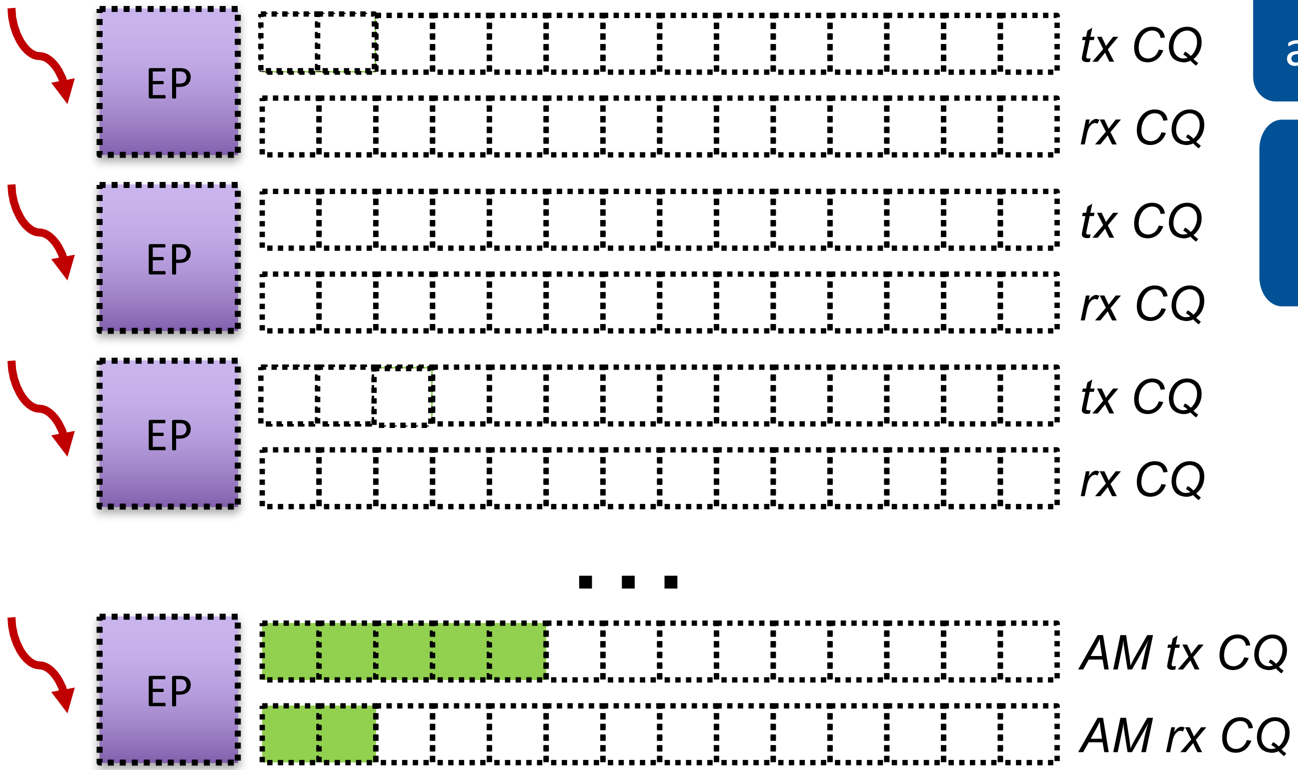
- **Chapel progression is about servicing active messages**
  - Execute on statement
- **Network progression is about resource management**
  - Must free up hardware resources consumed by in flight messages
- **Comm layer must do both**
  - Does not use libfabric auto-progress
  - All about checking CQs

Progress loop (sort of)



Query tx CQs and restart tasks

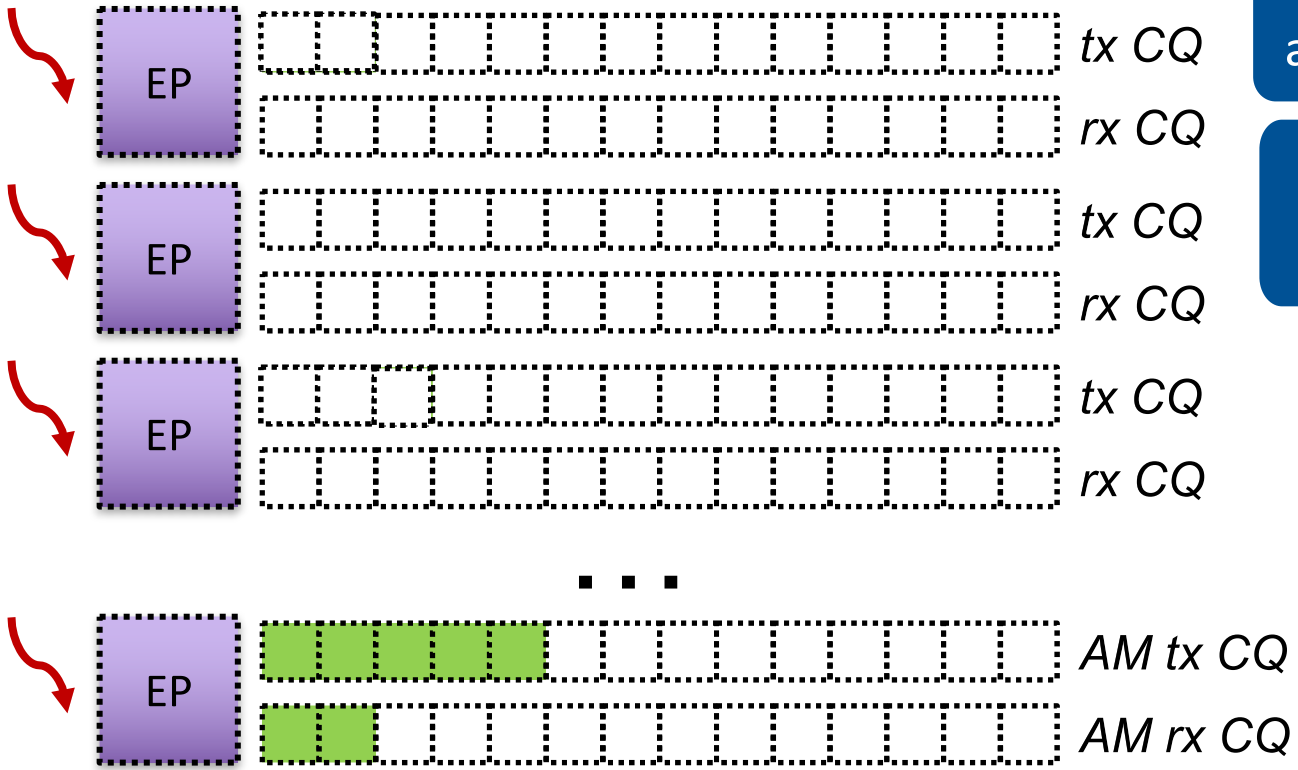
Progress loop (sort of)



Query tx CQs and restart tasks

Query rx CQs (for remote progress)

Progress loop (sort of)

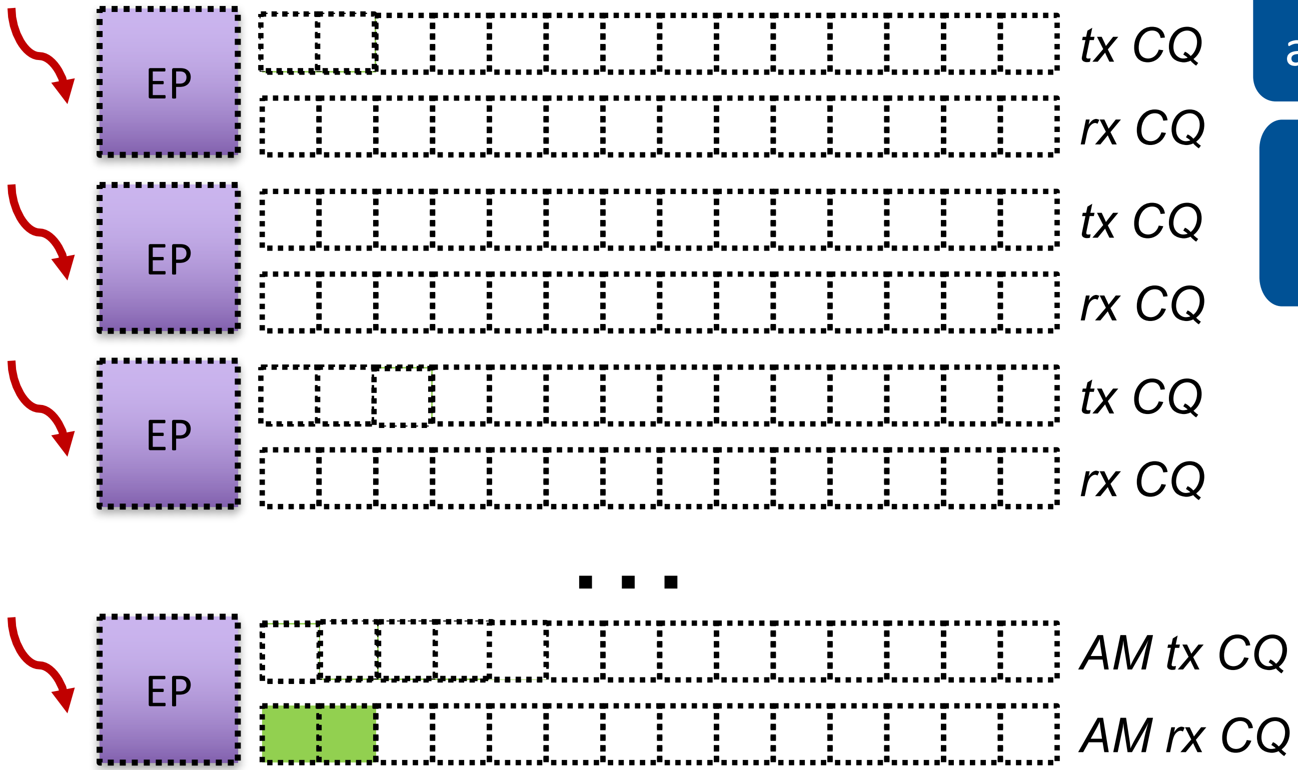


Query tx CQs and restart tasks

Query rx CQs (for remote progress)

Query AM tx CQ

Progress loop (sort of)



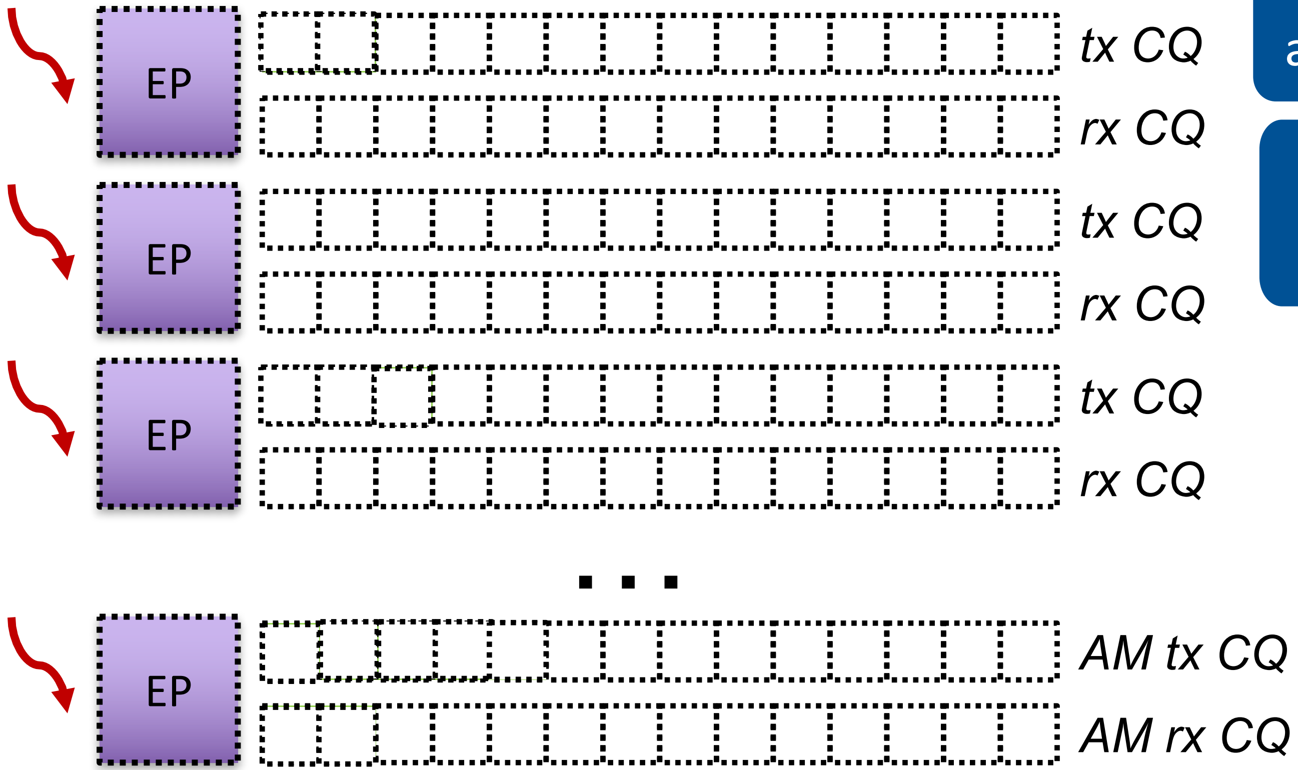
Query tx CQs and restart tasks

Query rx CQs (for remote progress)

Query AM tx CQ

Query AM rx CQ, launch AMs, send acks

Progress loop (sort of)



Query tx CQs and restart tasks

Query rx CQs (for remote progress)

Query AM tx CQ

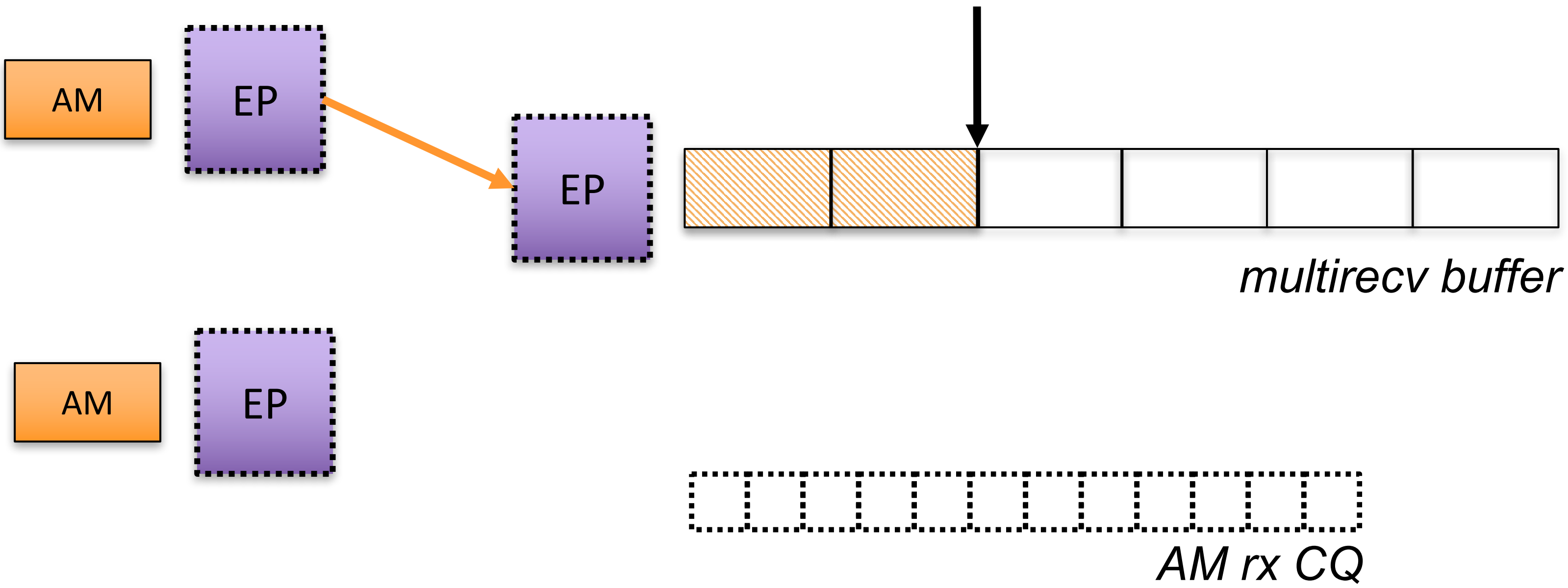
Query AM rx CQ, launch AMs, send acks

Active message implementation

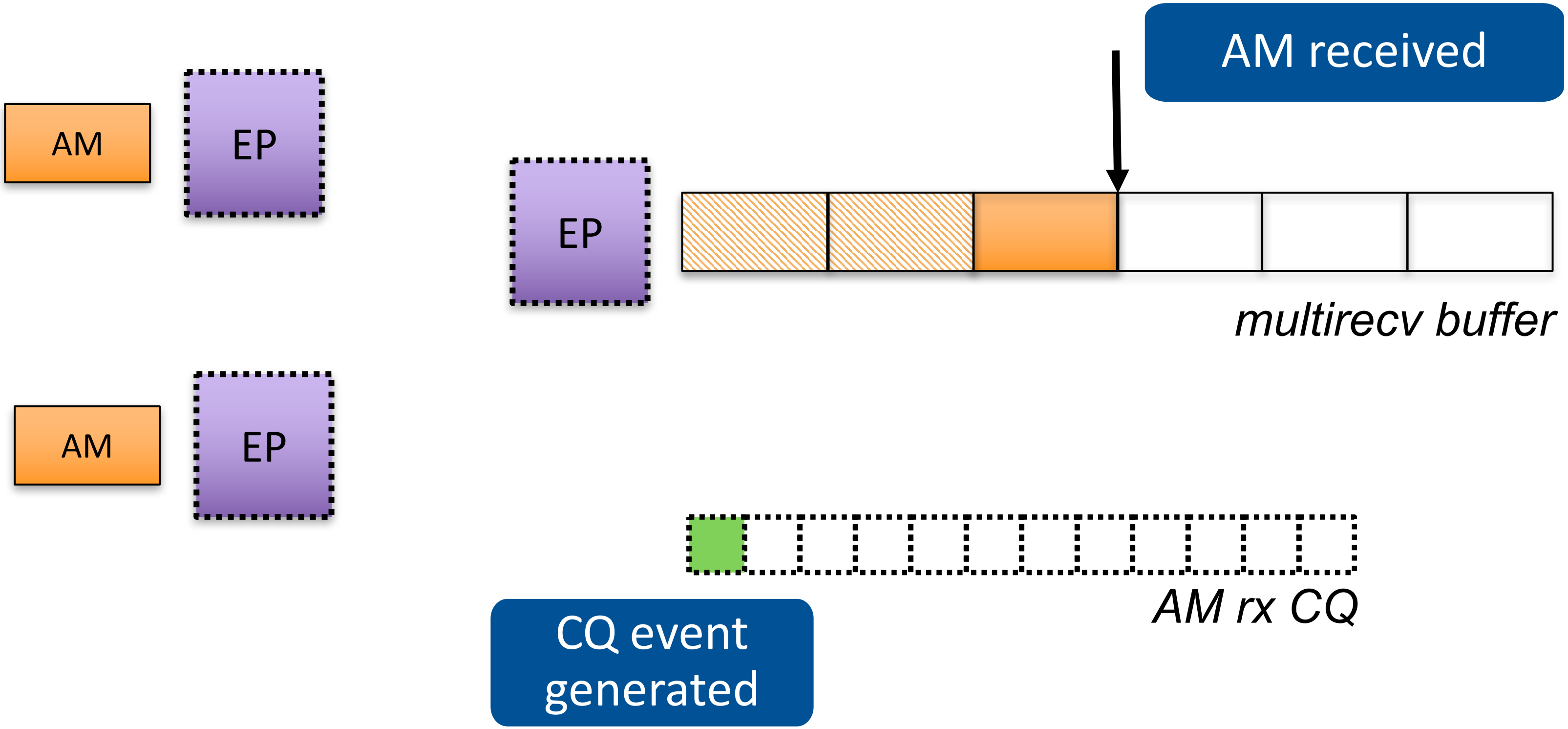
- **Two-sided operation (send/recv)**
  - Initiating locale sends an message to the remote endpoint
  - Remote locale posts one or more *multi-recv* buffers on the endpoint

- **Active message processing**
  - Query AM rx CQ
  - Run or launch on statement body
  - Ack using address in the active message (put)
  - Provider returns a special CQ event when the buffer is consumed

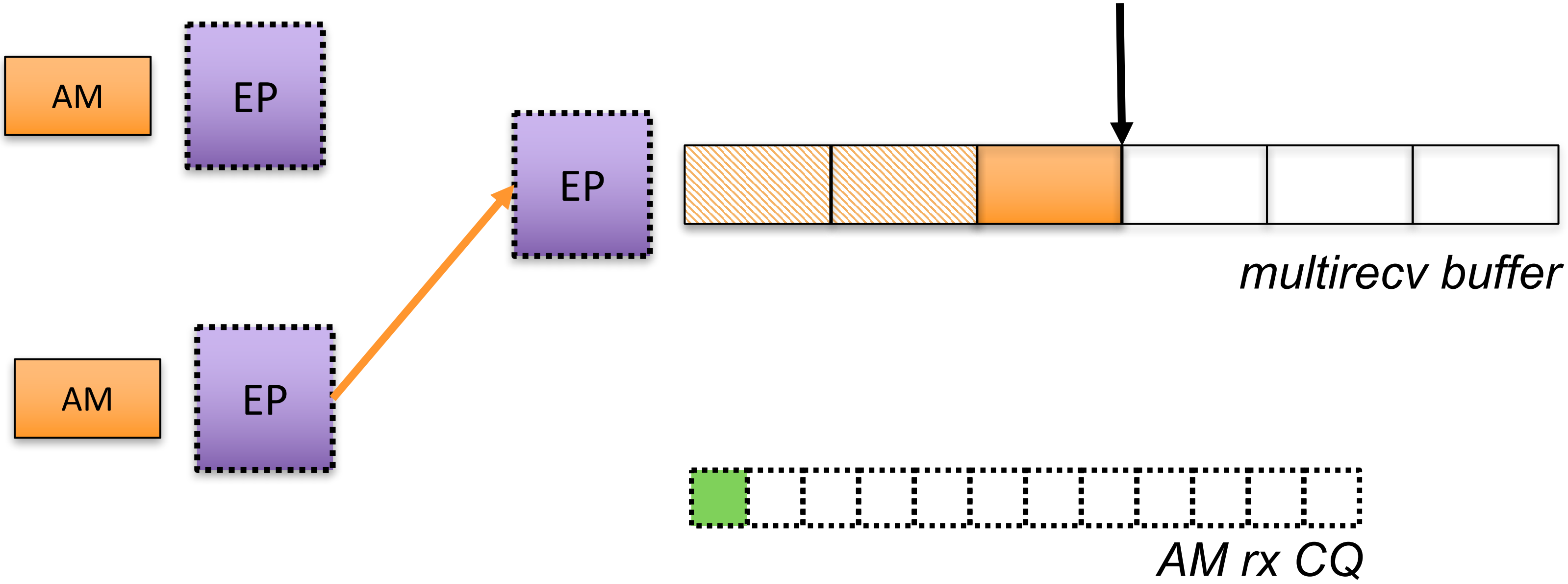
Active message example



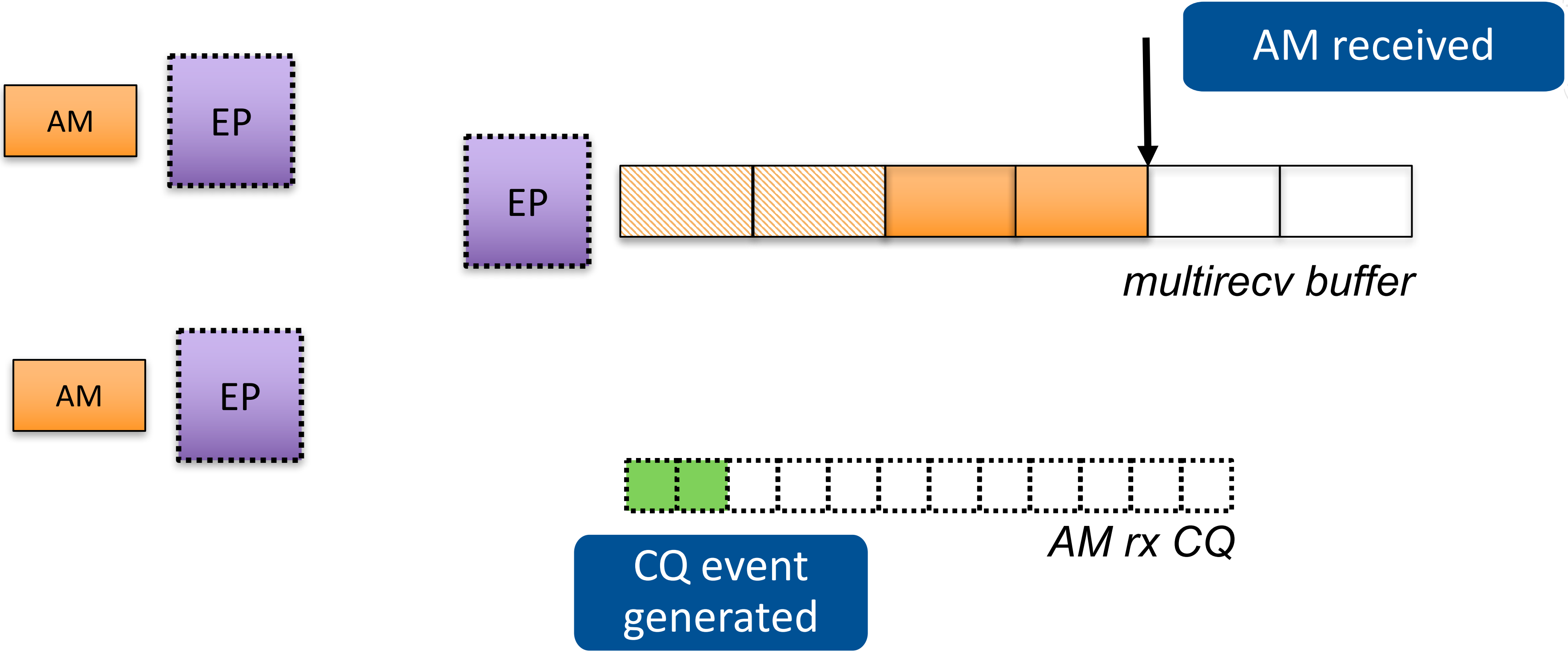
Active message example



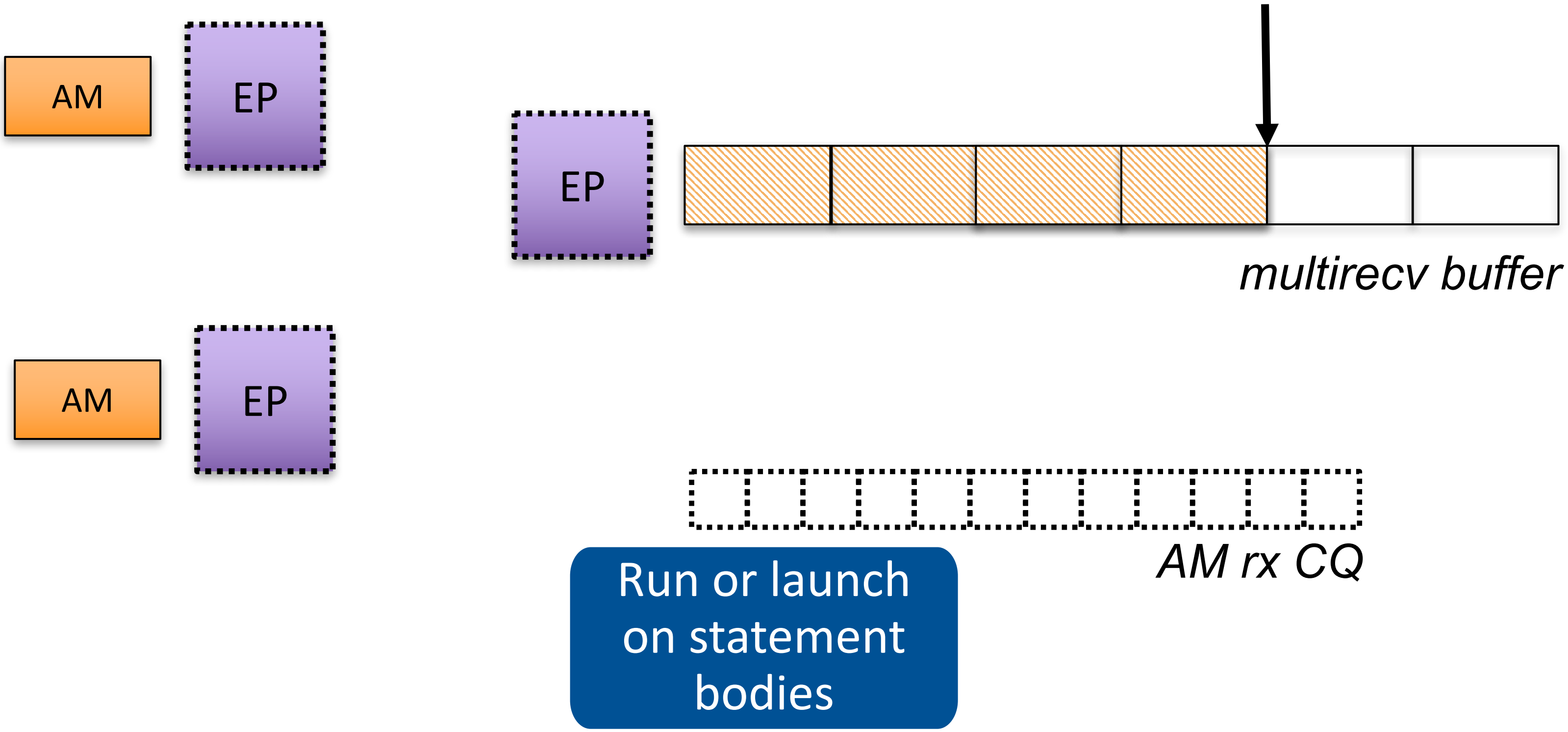
Active message example



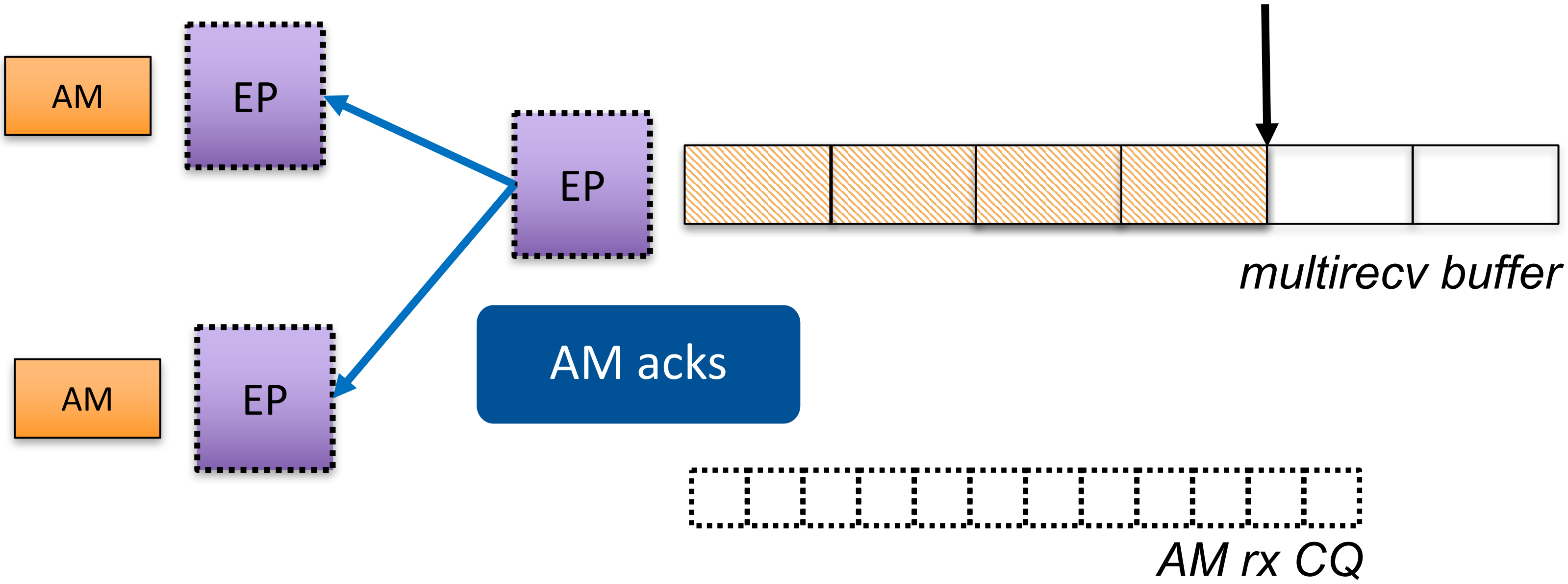
Active message example



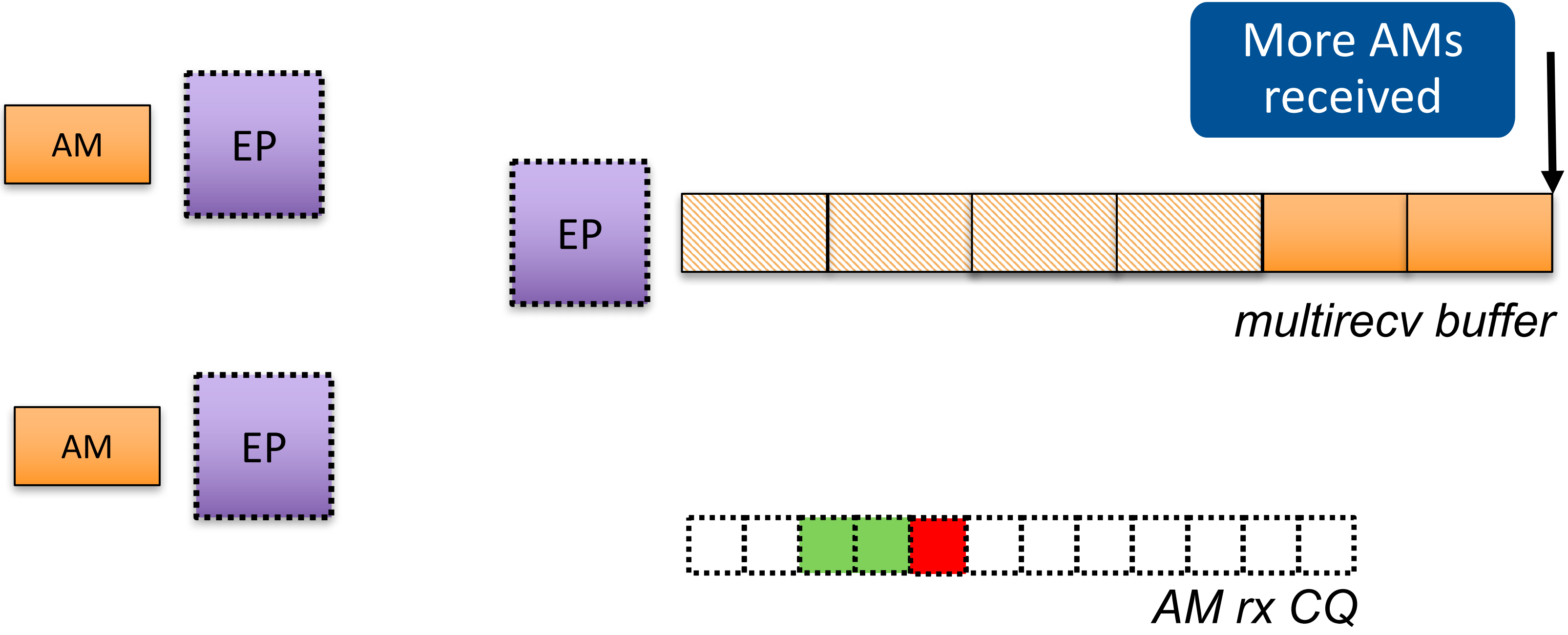
Active message example



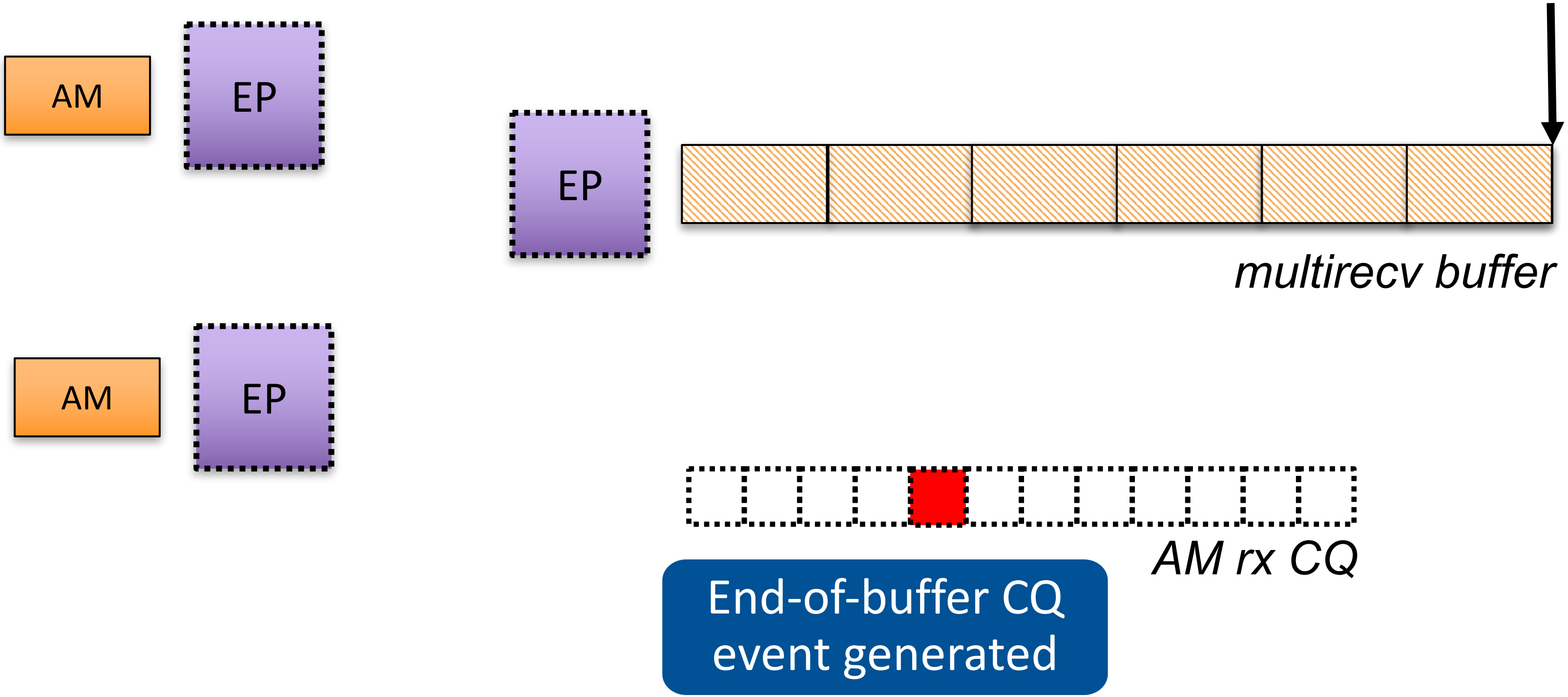
Active message example



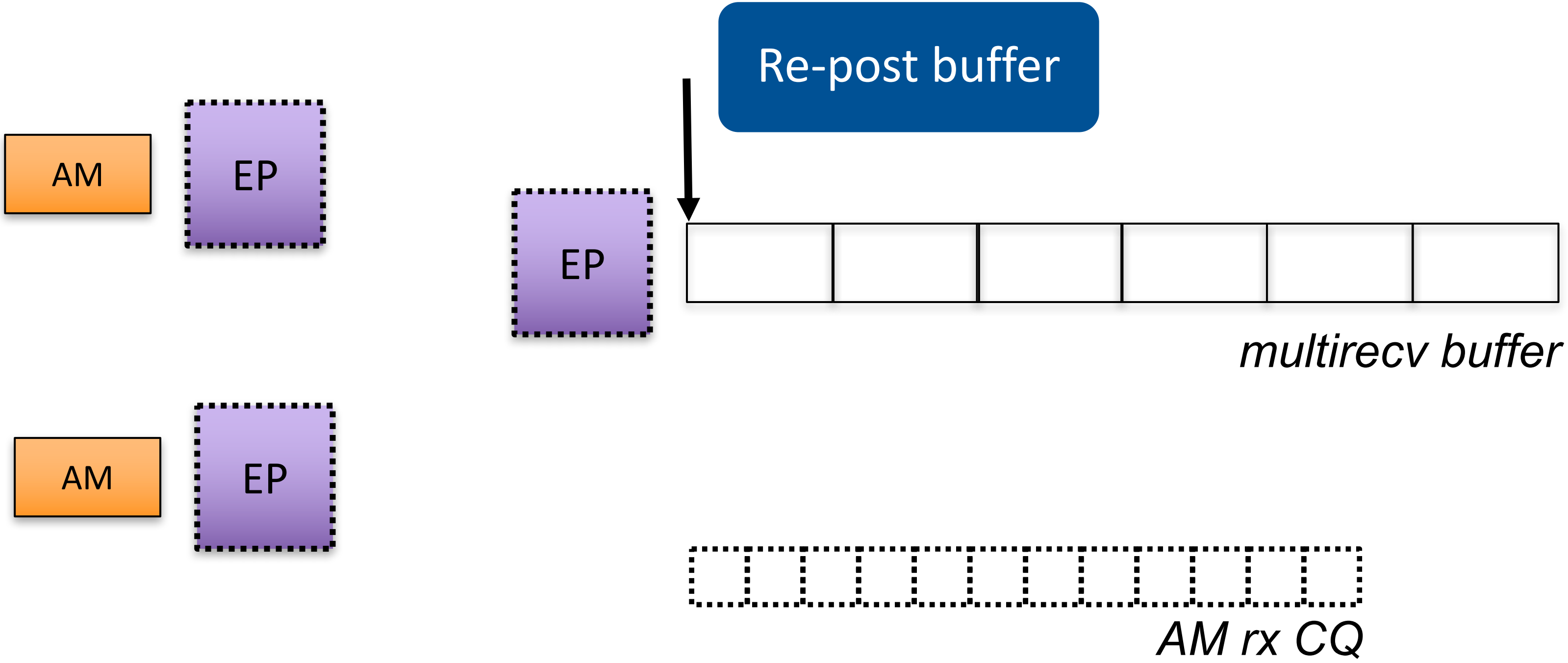
Active message example



Active message example



Active message example



Outline

- Brief overview of libfabric
- Chapel ofi communication layer
- Lessons learned
- Status and conclusions

Lessons learned: libfabric

- **Writing portable code is not always easy**
  - If feature X is not available, must implement using feature Y
- **Start up is still a pain**
  - I cheated and used PMI (for now)
- **Want to know if an operation is supported in hardware**
  - e.g., I'd rather do the atomic with the module code if it's not supported
- **Manual progress as defined is cumbersome**
  - Must progress individual completion structures
- **Can we utilize the auto progress thread?**
  - e.g., small function to be invoked by the internal progress thread

Lessons learned: Chapel

- **Make comm layer a dynamic decision**
  - One (or more) fewer compile time constraints
- **Refactor strided operations so as to reuse logic**
  - Currently logic is replicated in every comm layer
- **Make network atomics part of comm layer interface**
  - Unsupported atomics should be implemented by the module
- **Enable use of hardware support for collectives**
  - No way to use triggered operations or other hardware support

Outline

- Brief overview of libfabric
- Chapel ofi communication layer
- Lessons learned
- Status and conclusions

Status

- **Basic initialization and teardown in place**
- **Comm diagnostics and callbacks in place**
- **External prototypes**
  - Put/get
  - Progress loop (partially in place)
  - AM infrastructure (partially in place)

Conclusions

- **OFI libfabric promises portability and performance**
  - Still might need per-platform tuning (provider constraints, last 10%)
  - Vendors must adopt it (outlook good)

- **Chapel comm layer should use it ☺**
  - More complicated in some ways (start up, multiple implementations)
  - Less complicated in other ways (API designed for middleware)

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, and URIKA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.

For more info

- **OFIWG libfabric: https://ofiwg.github.io/libfabric/**
  - General overview, man pages and other documentation
- **ofiwg repo: https://github.com/ofiwg/libfabric**
  - Main upstream project (releases cut from here)
- **ofi-cray repo: https://github.com/ofi-cray/libfabric-cray**
  - Cray XC development and GNI provider-specific wikis