Design and Analysis of the Network Software Stack of an Asynchronous Many-Task System -- The LCI Parcelport of HPX

Jiakun Yan (UIUC), Hartmut Kaiser (LSU), Marc Snir (UIUC)

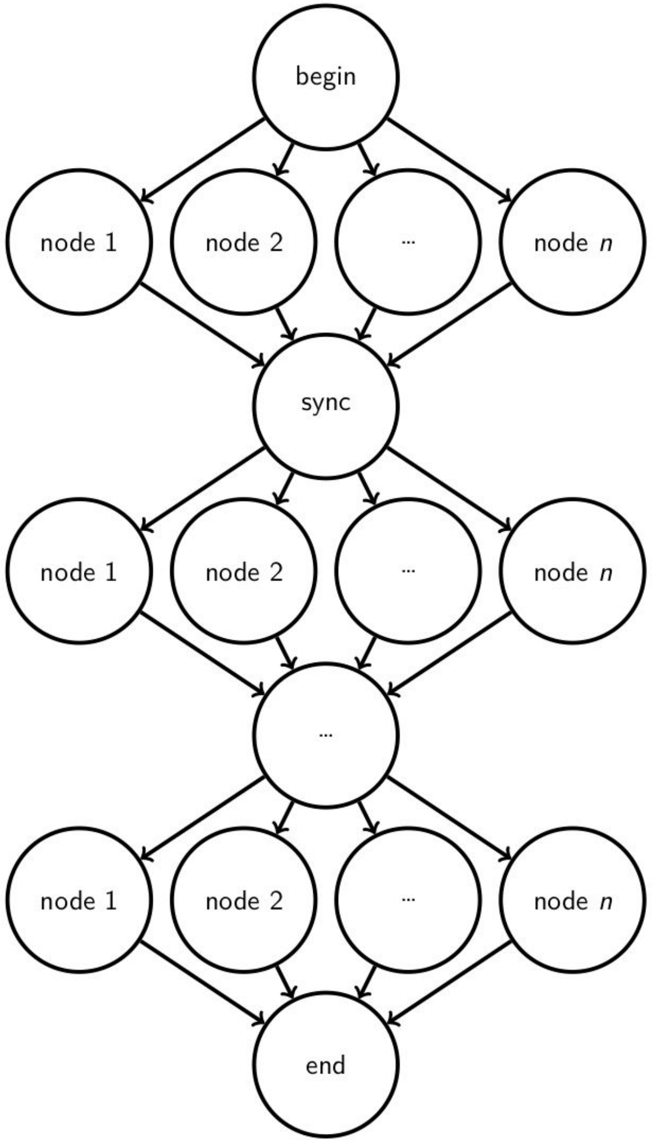
Parallel Applications Workshop, Alternatives To MPI+X
November 13, 2023

Traditional Programming Model

Bulk-Synchronous Programming (BSP)

Challenges:

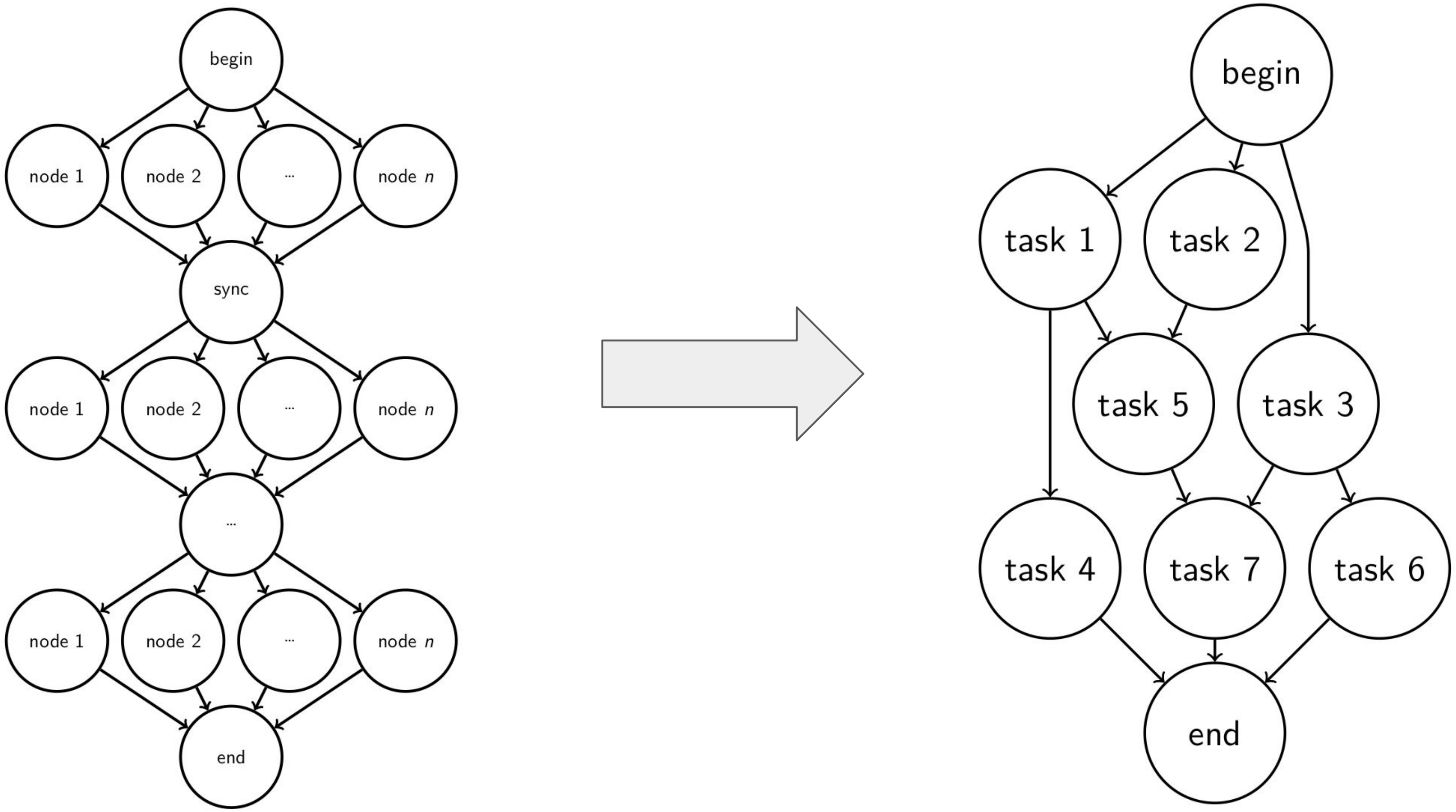
- Heterogeneous Architectures
  - Multicore CPUs + GPUs + other accelerators…
- Irregular applications
  - Load imbalance, data-driven control flow…



Asynchronous Many-Task Systems (AMTs)

Users: Tasks + Dependencies

Runtime: scheduling, data movement (communication), etc.



Changes in Communication Characteristics

Larger number of independent messages

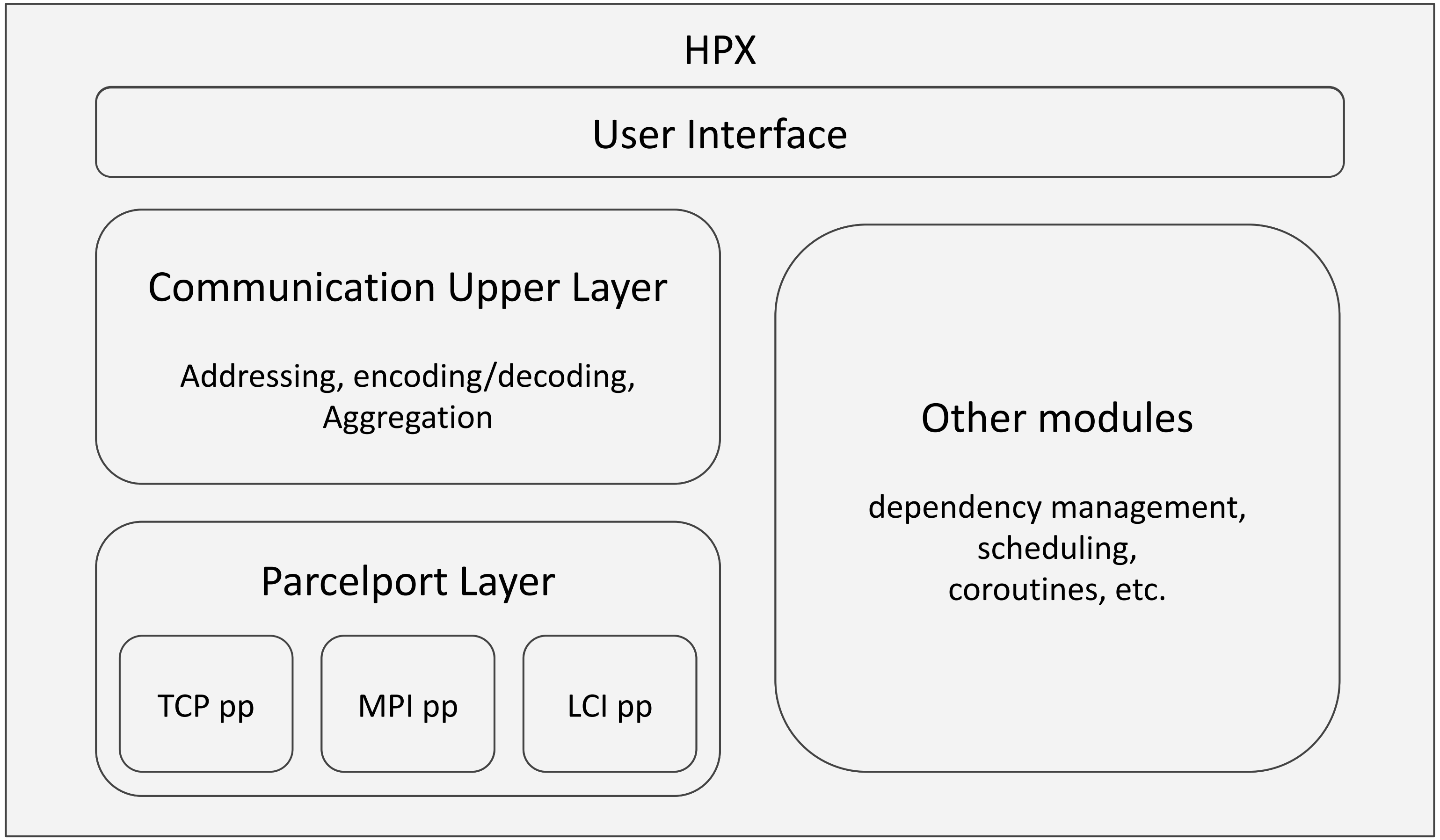
Smaller messages

Dynamic communication patterns

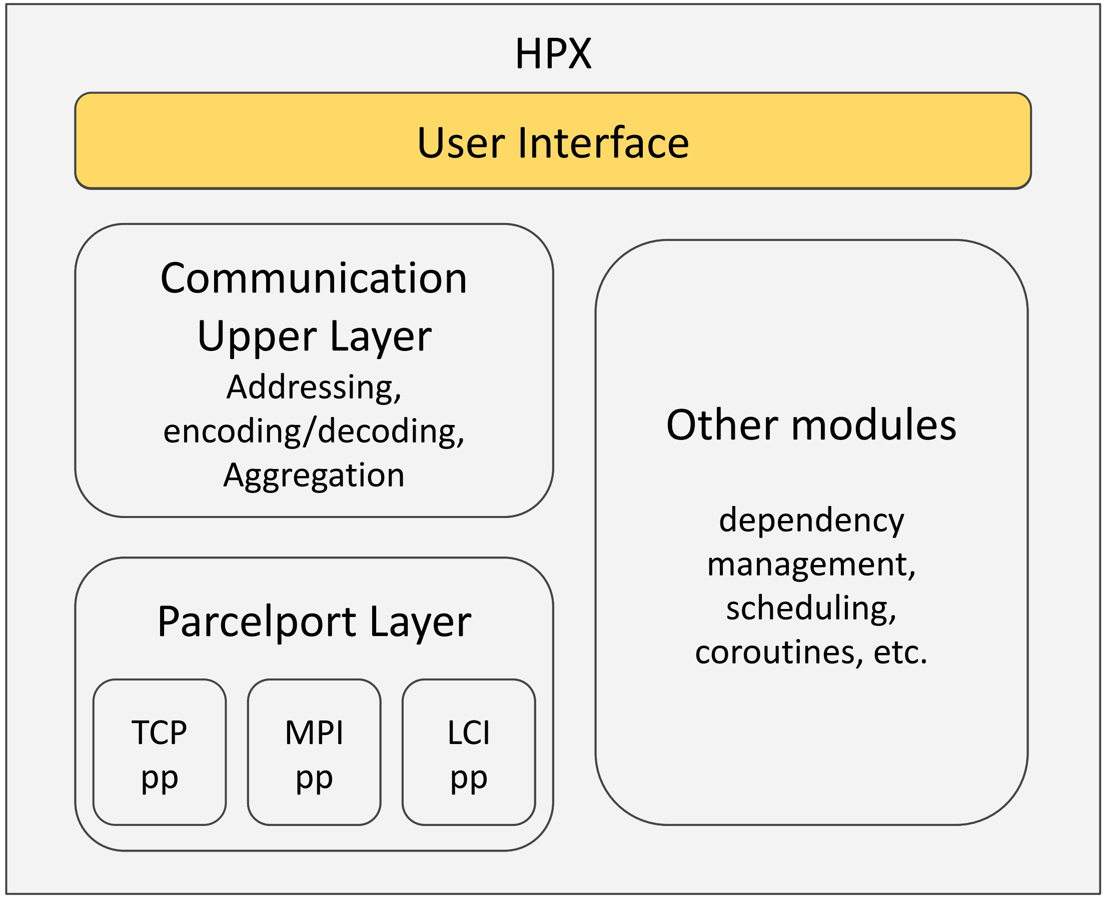
Multithreaded

HPX and Its Communication Stack

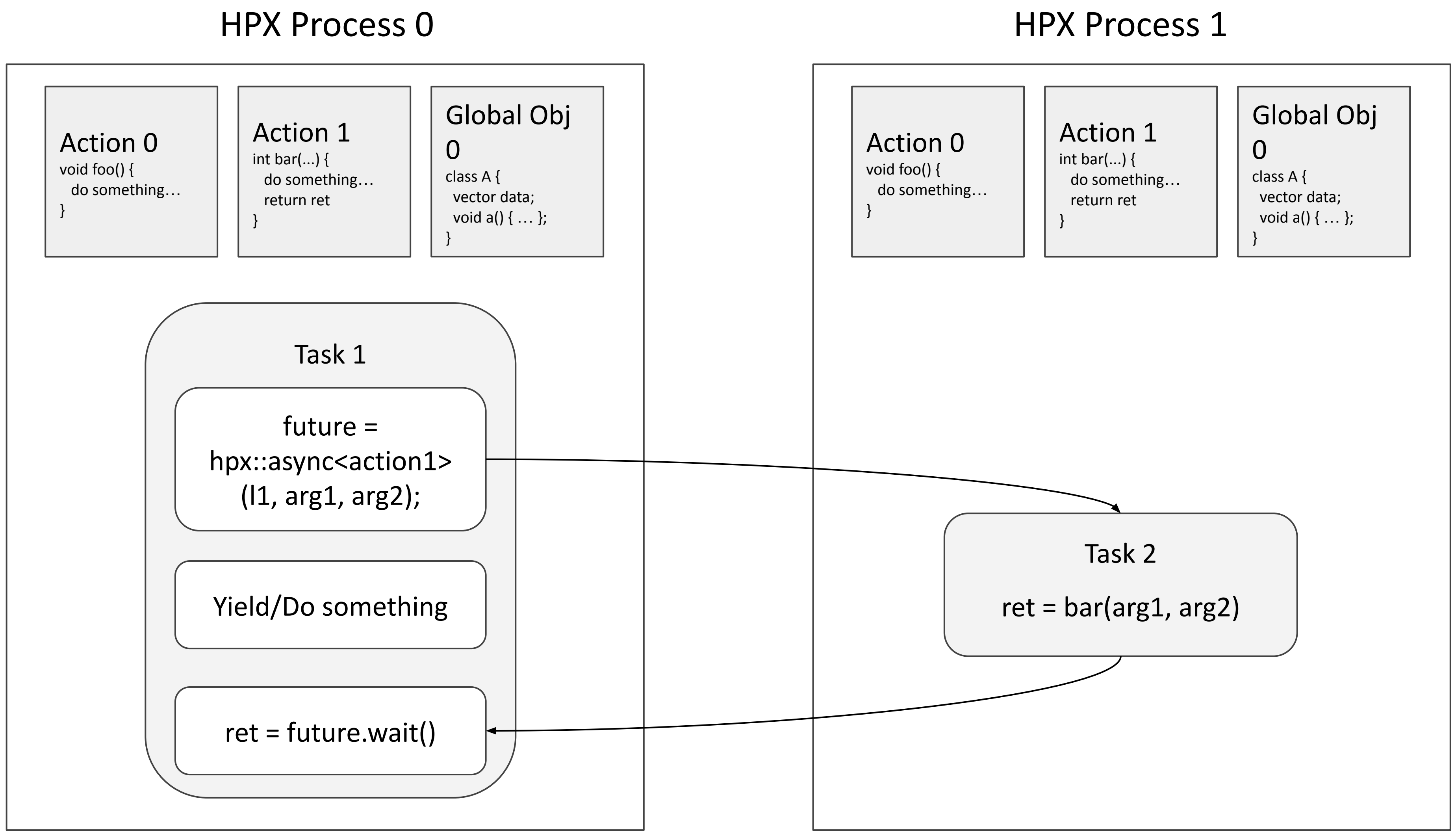
HPX Communication Software Stack



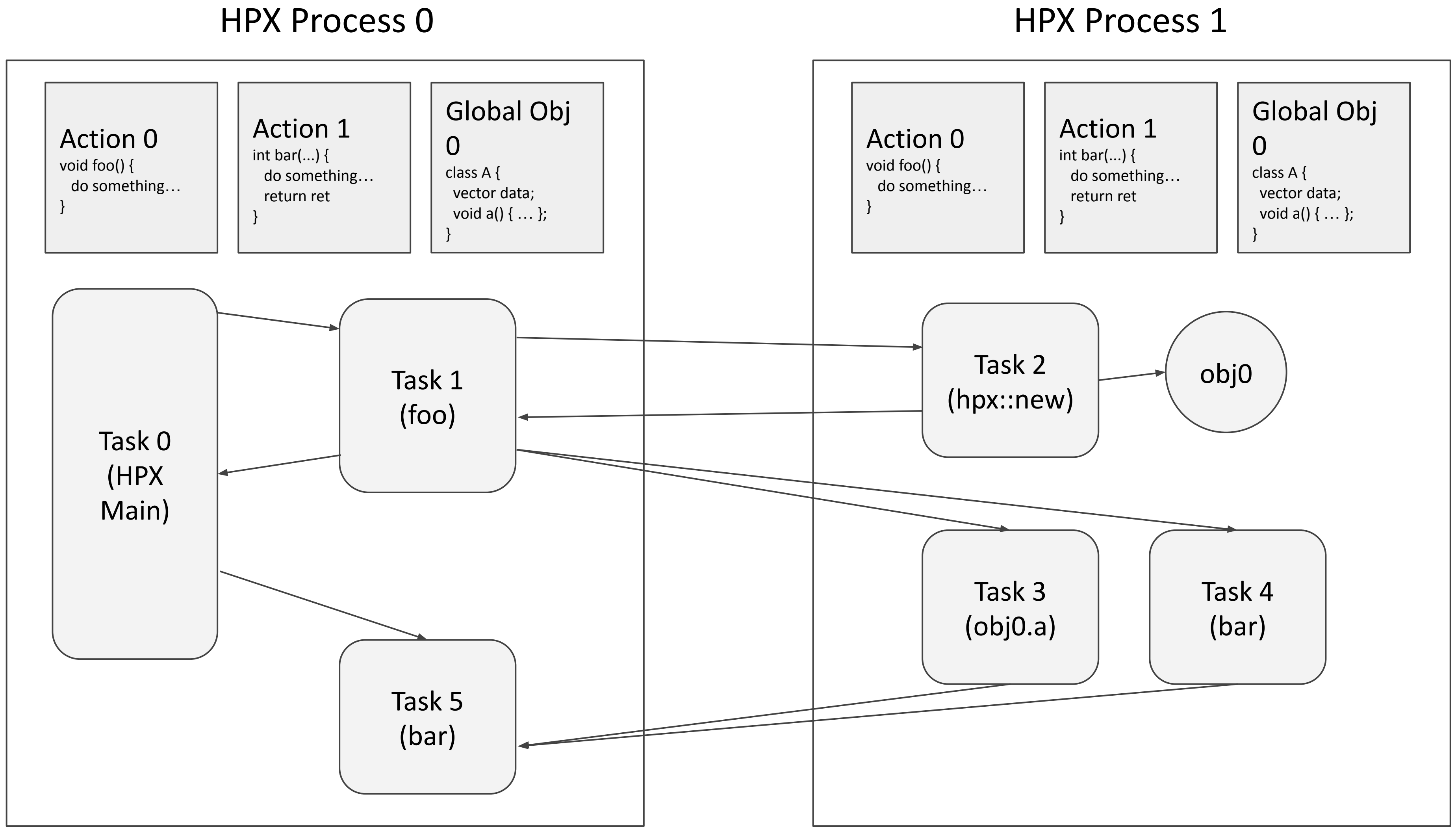
User Interface



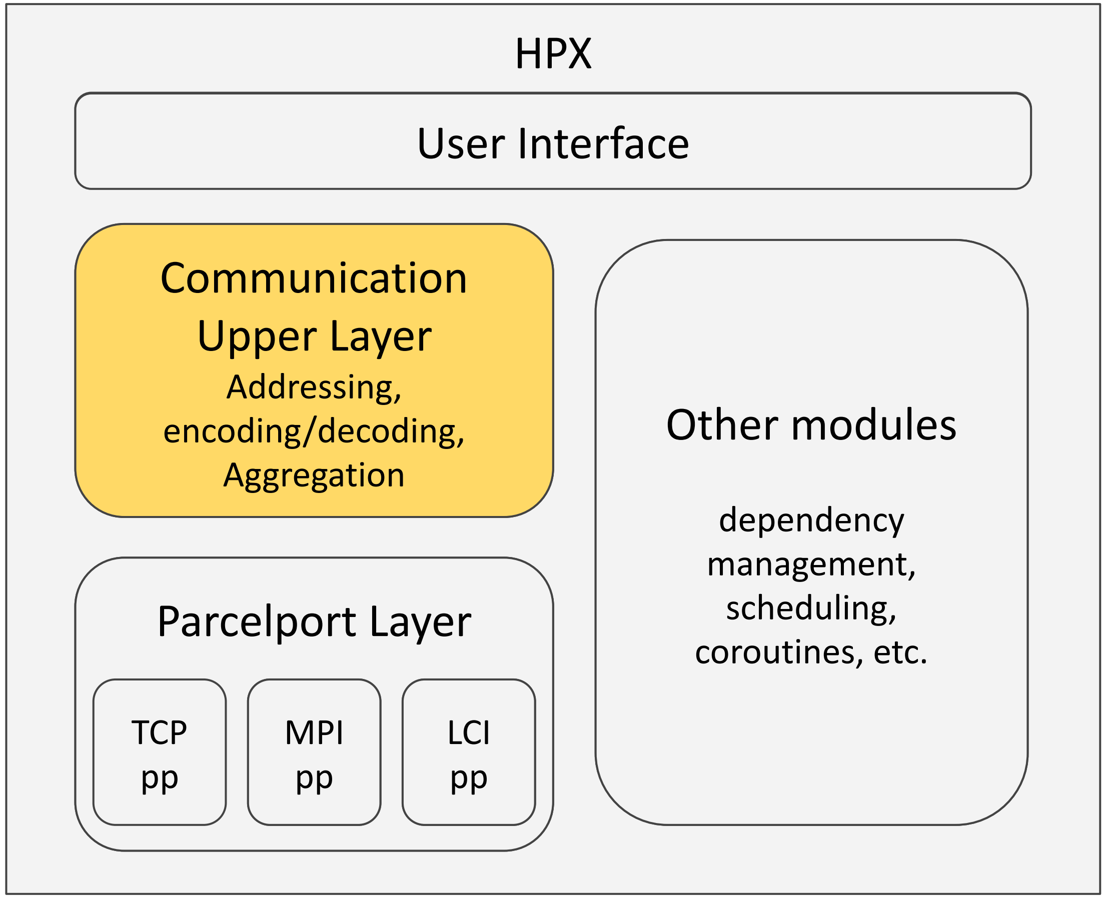
Remote Task Invocation



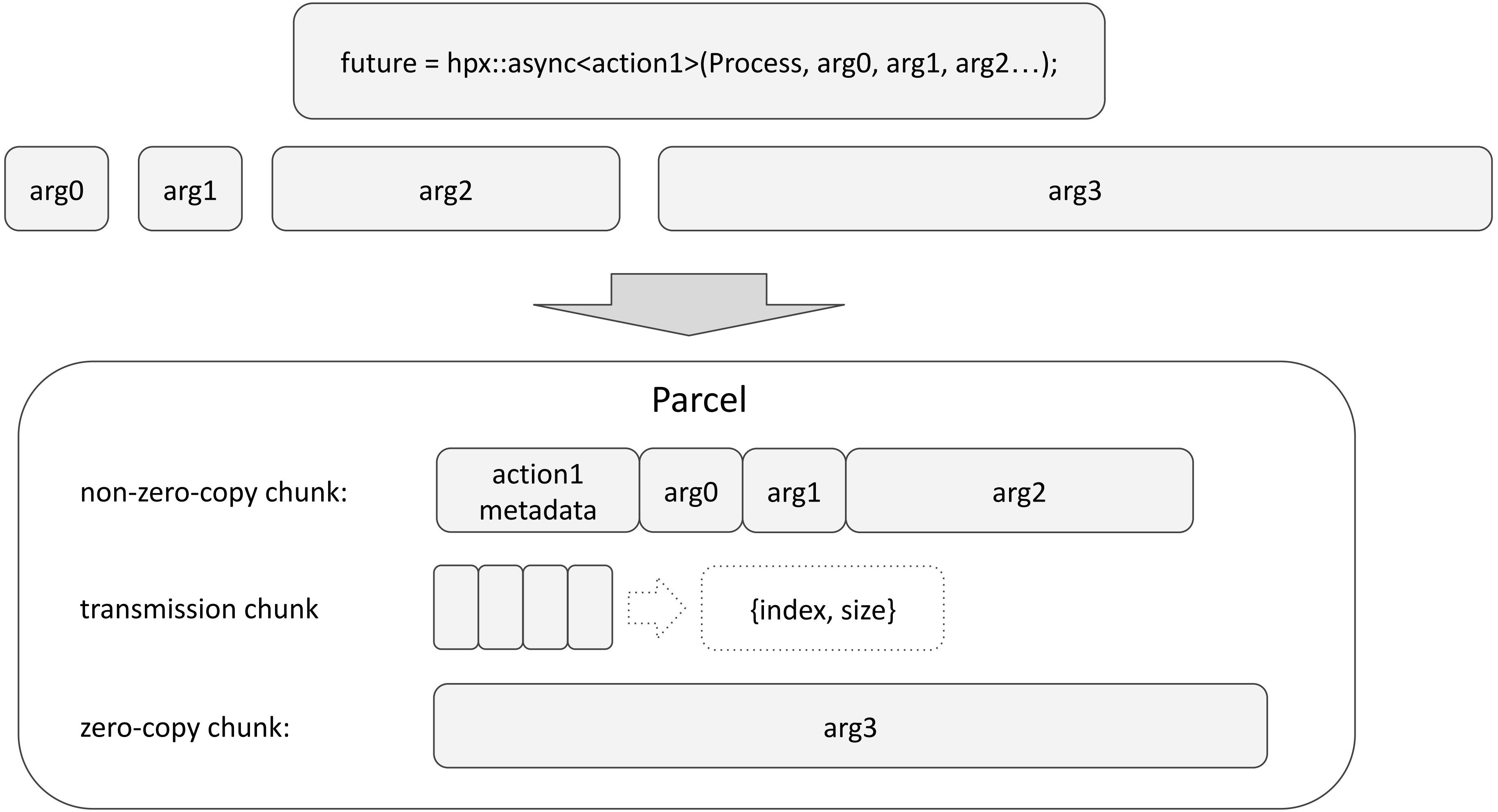
Composing Futures -> Task Graph



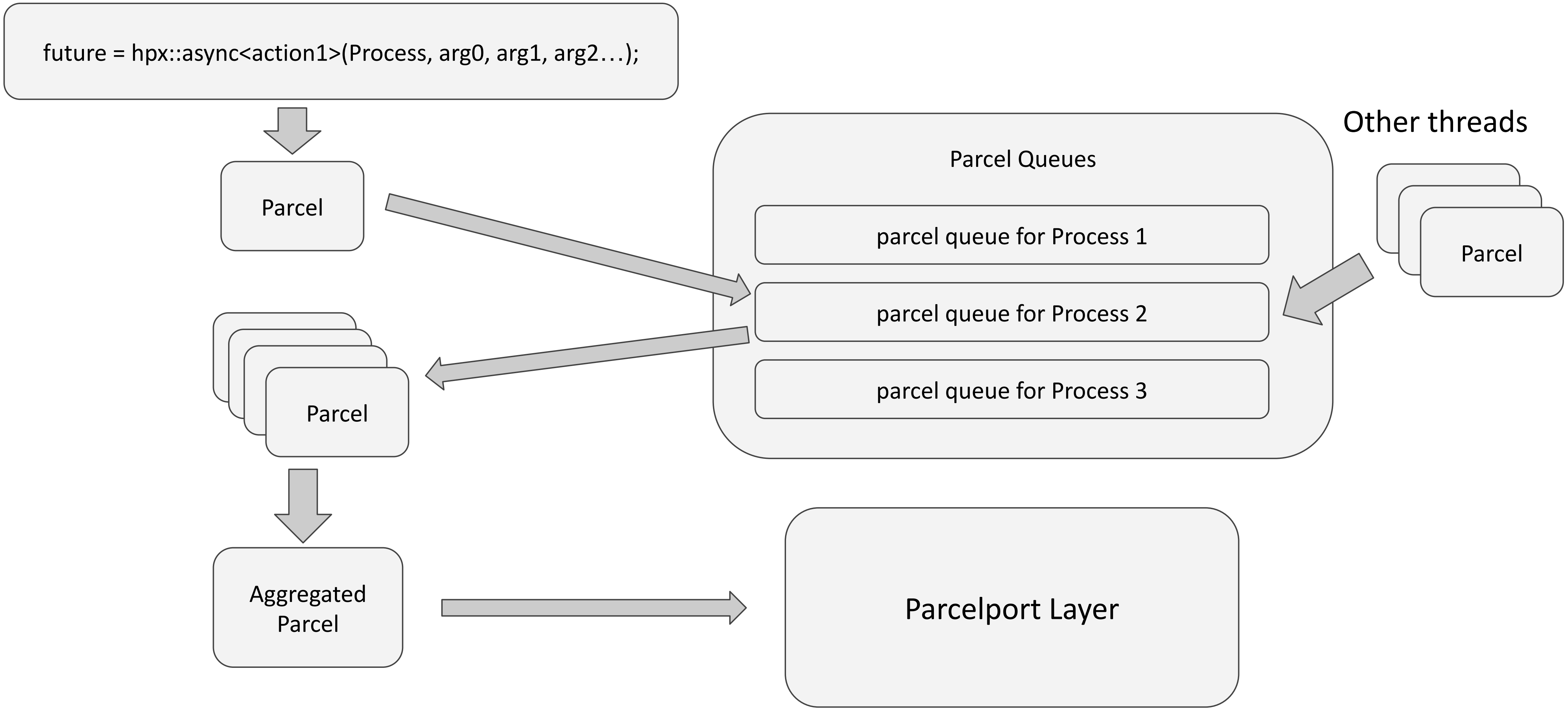
Communication Upper Layer



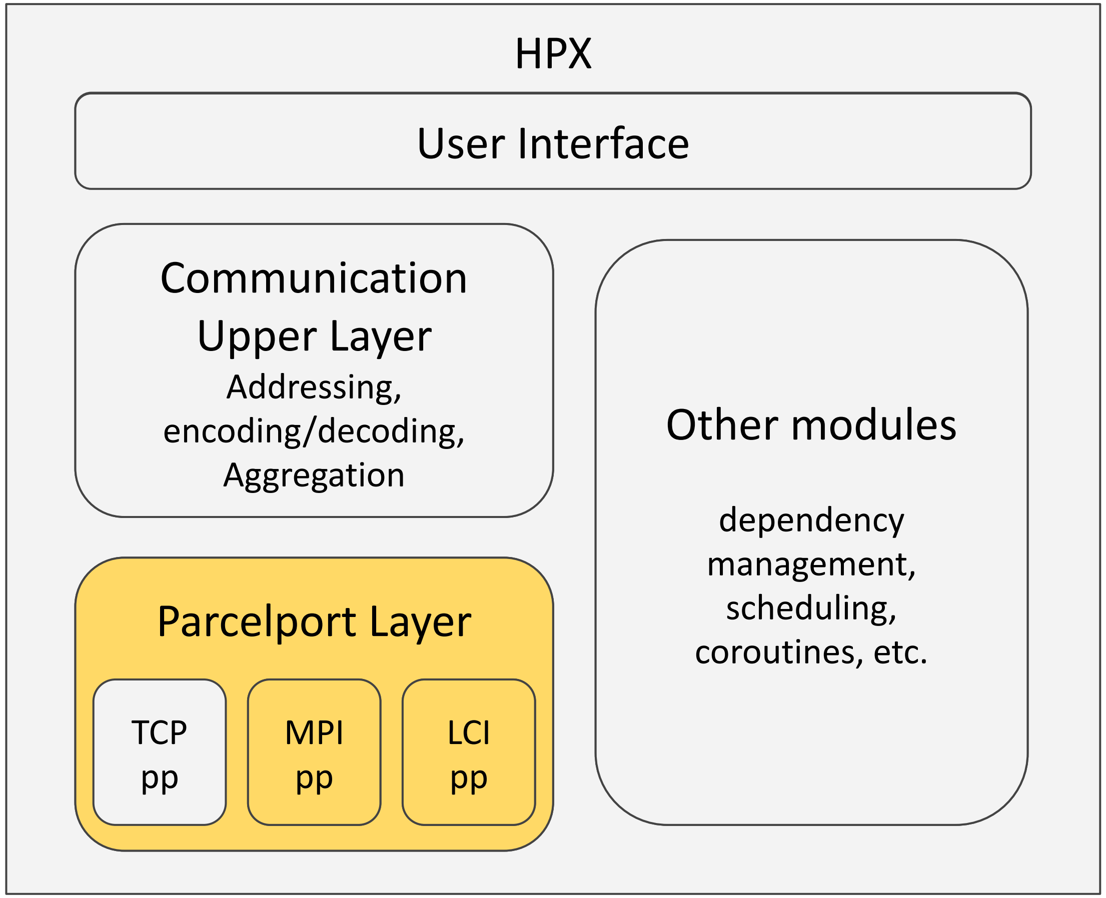
Communication Upper Layer: Parcel encoding



Communication Upper Layer: Aggregation



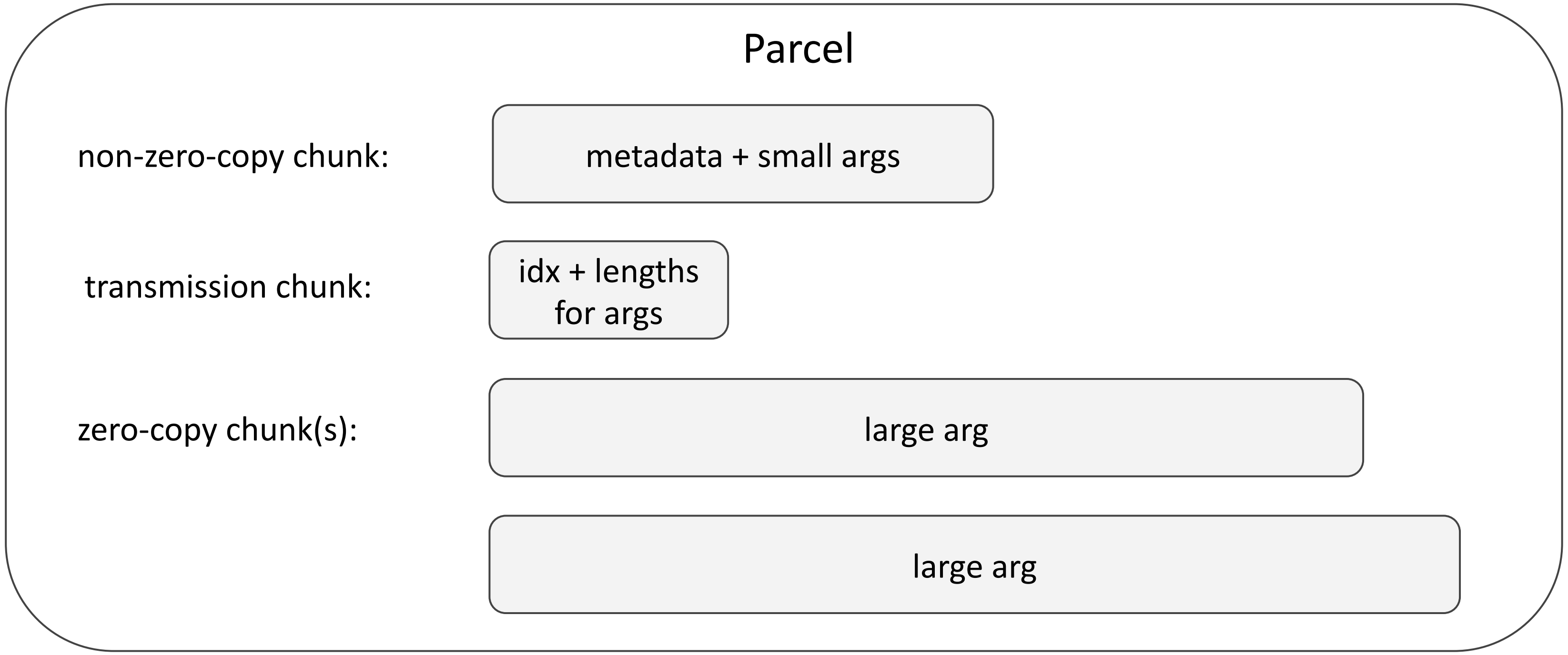
Parcelport Layer



Parcelport Layer

The job of parcelport layer is to transfer a parcel to the specified remote Process.

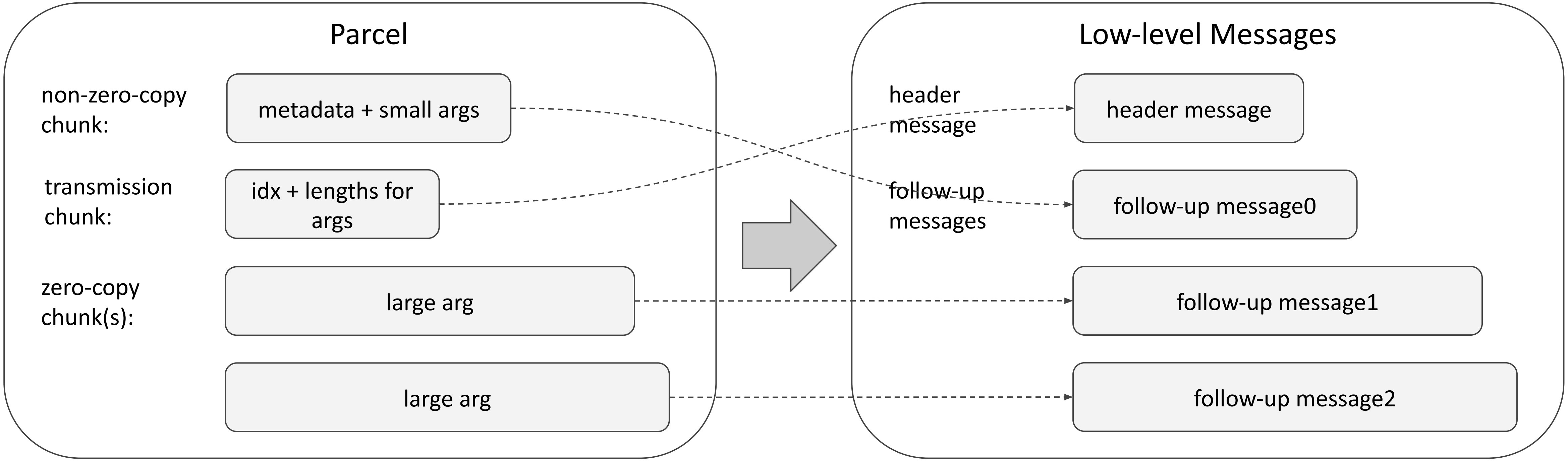
(transmission chunk is only needed when there is at least one zero-copy chunk)



MPI/LCI Parcelport: Header

Generate a “header” of fixed length (tag, num of zc chunk, size of nzc/t chunk…)

Piggybacking non-zero-copy chunk or transmission chunk if possible.

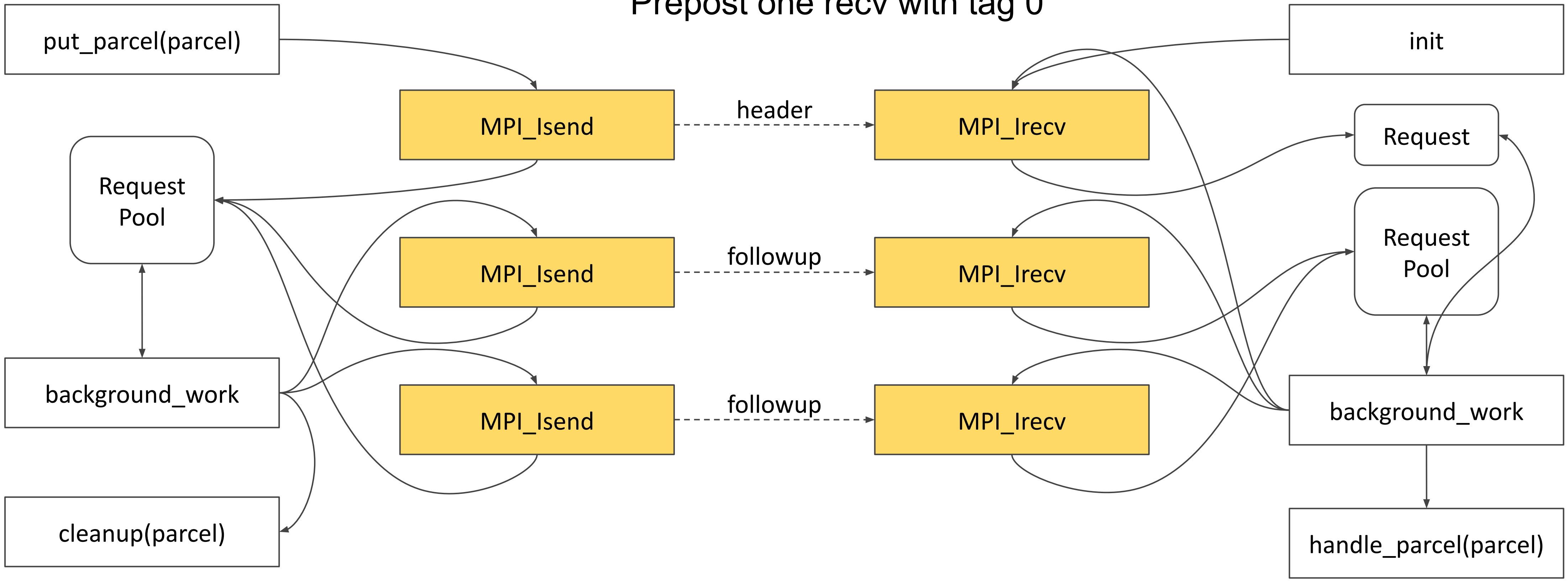


MPI Parcelport: Communication Primitives

All messages are transferred with MPI_Isend/Irecv

header: tag 0; followup: a distinct tag

Prepost one recv with tag 0

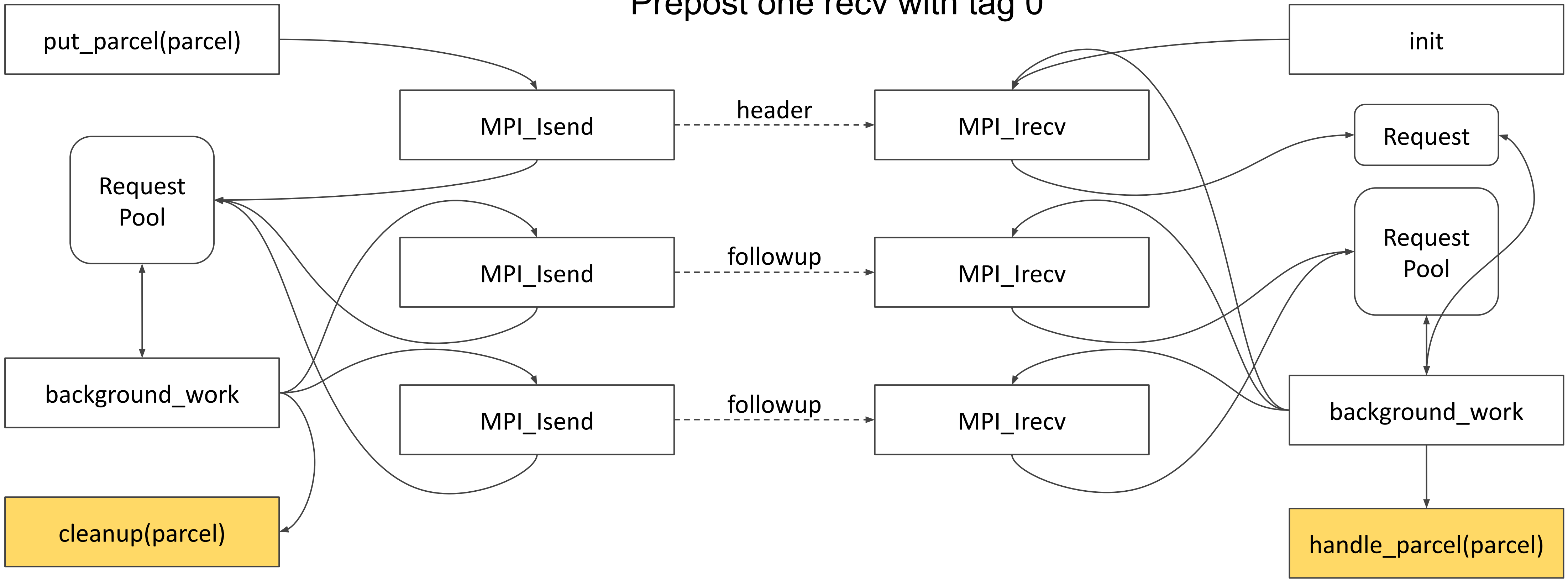


MPI Parcelport: Communication Primitives

All messages are transferred with MPI_Isend/Irecv

header: tag 0; followup: a distinct tag

Prepost one recv with tag 0

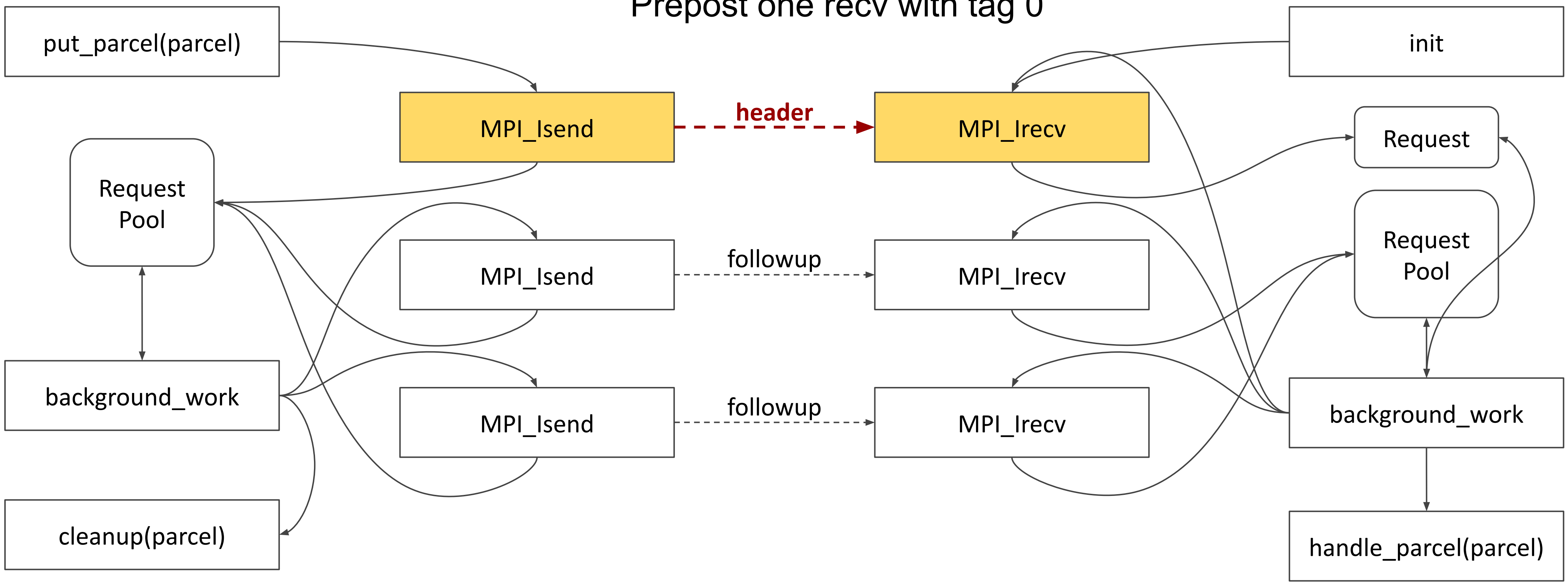


MPI Parcelport: Communication Primitives

All messages are transferred with MPI_Isend/Irecv

header: tag 0; followup: a distinct tag

Prepost one recv with tag 0

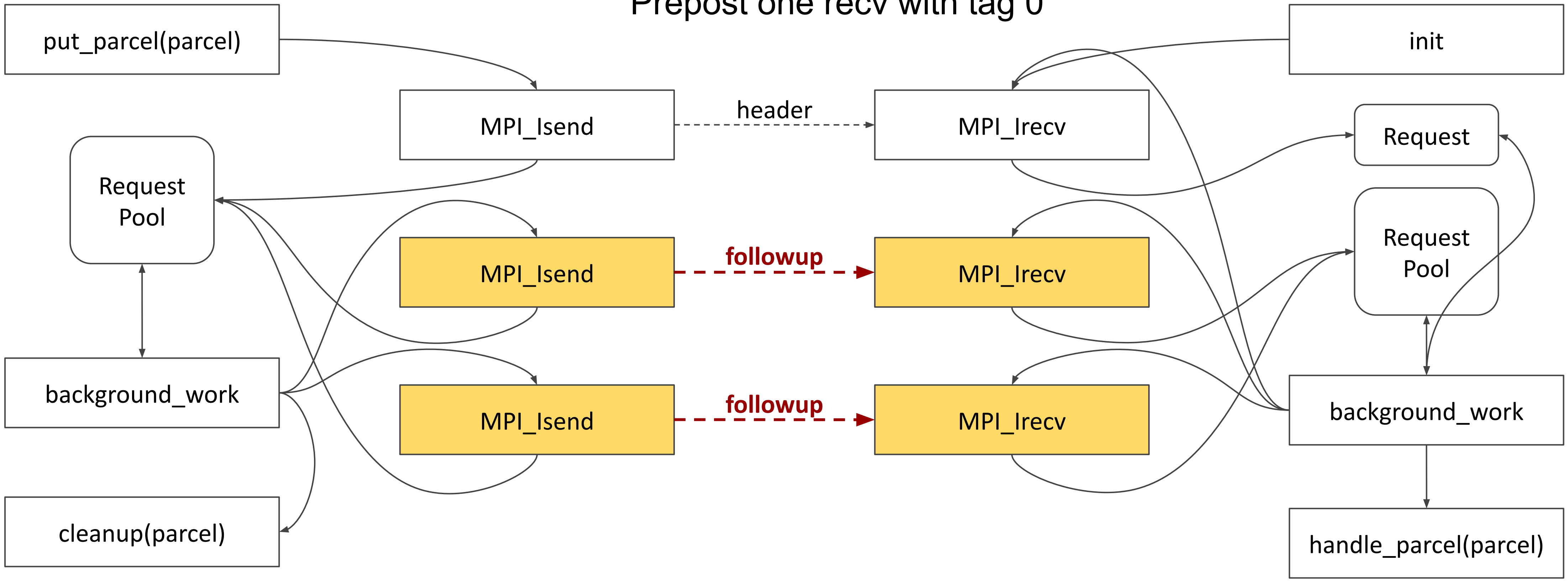


MPI Parcelport: Communication Primitives

All messages are transferred with MPI_Isend/Irecv

header: tag 0; **followup: a distinct tag**

Prepost one recv with tag 0

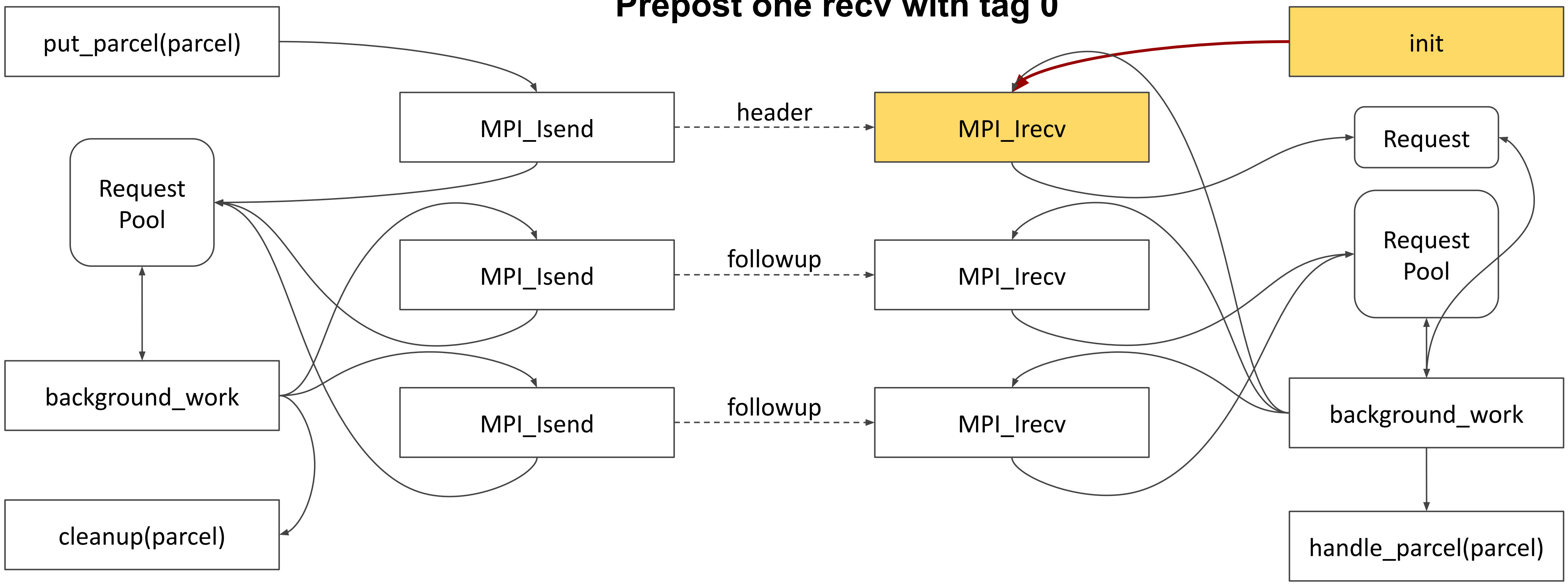


MPI Parcelport: Communication Primitives

All messages are transferred with MPI_Isend/Irecv

header: tag 0; followup: a distinct tag

Prepost one recv with tag 0

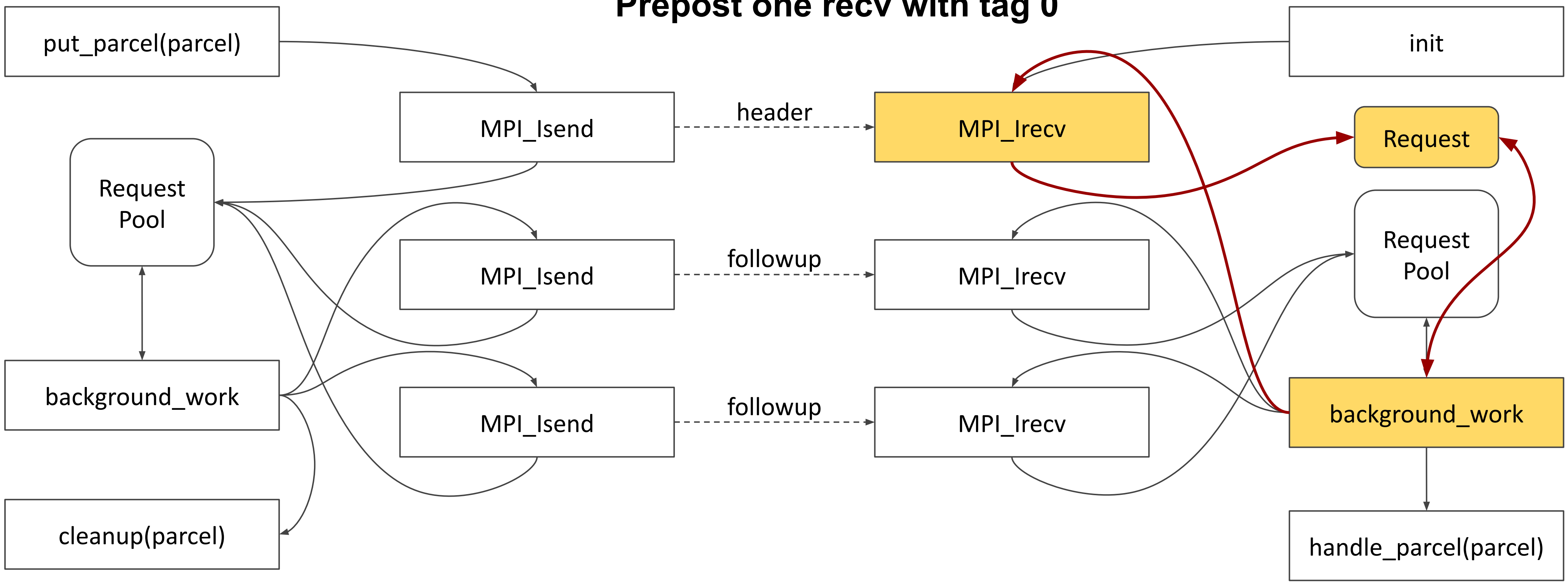


MPI Parcelport: Communication Primitives

All messages are transferred with MPI_Isend/Irecv

header: tag 0; followup: a distinct tag

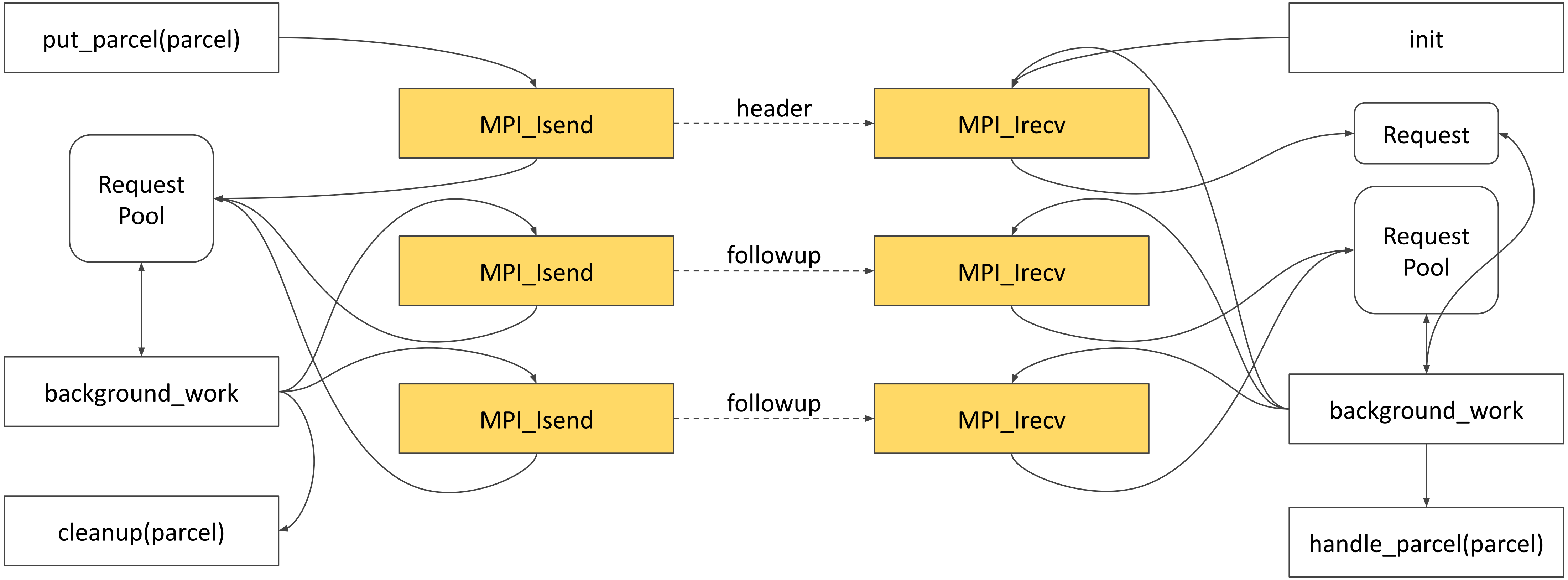
Prepost one recv with tag 0



MPI Parcelport: Synchronization

Isends/Irecvs are posted one by one

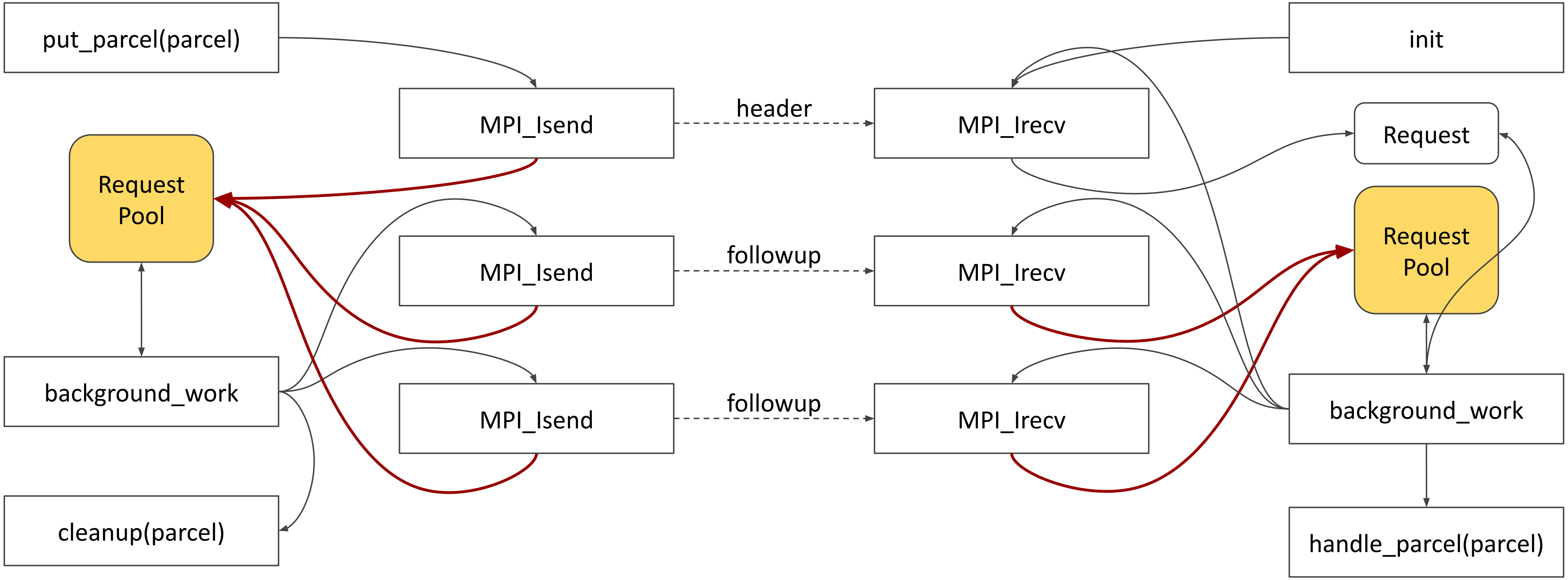
Request pools are checked in a round-robin manner.



MPI Parcelport: Synchronization

Isends/Irecvs are posted one by one

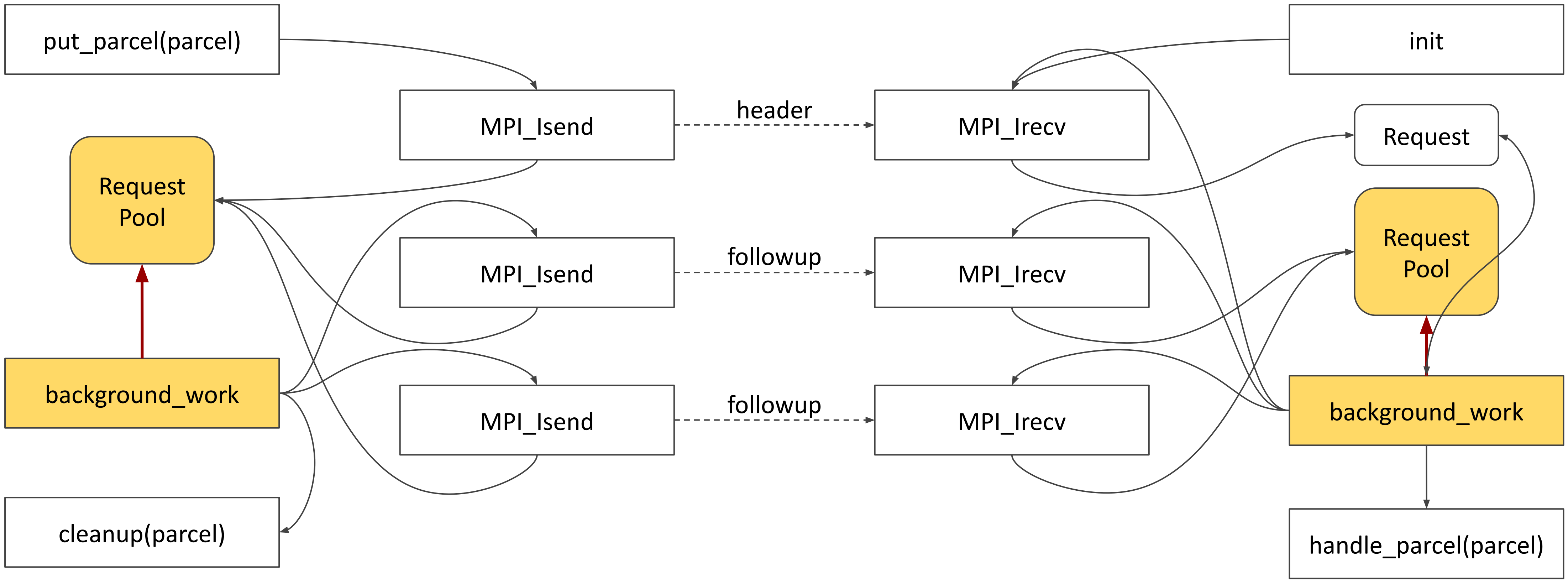
Request pools are checked in a round-robin manner.



MPI Parcelport: Synchronization

Isends/Irecvs are posted one by one

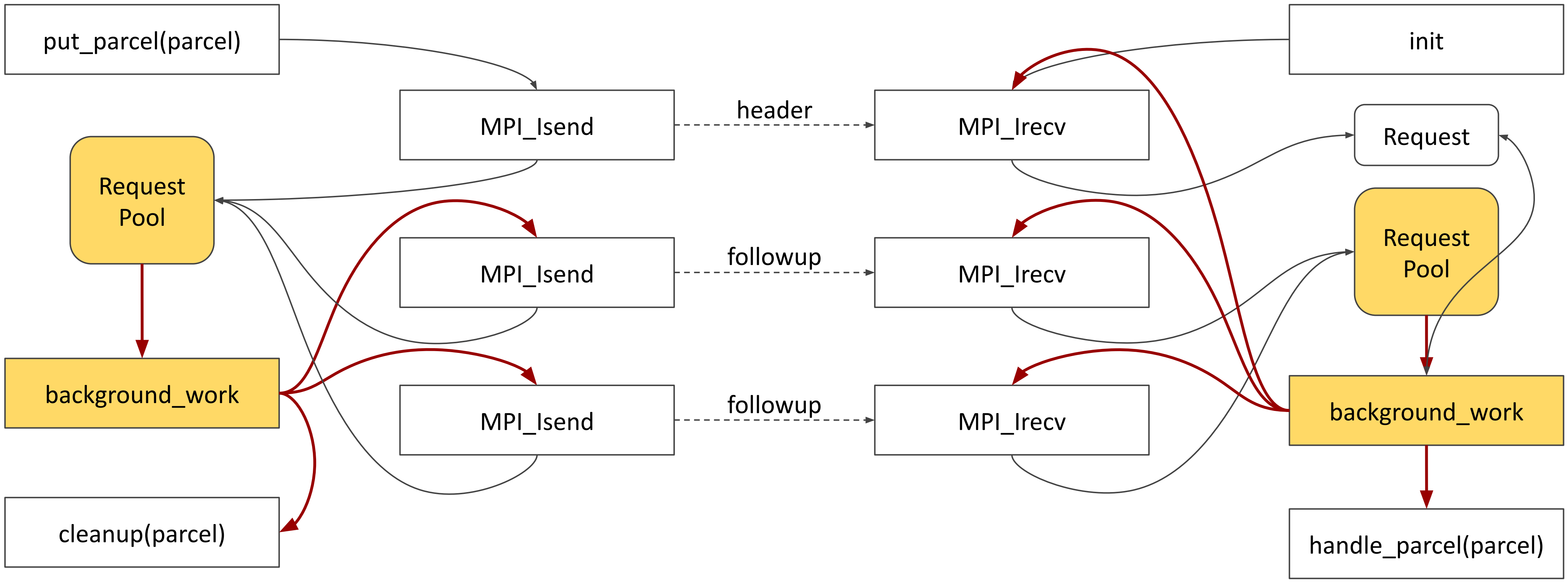
Request pools are checked in a round-robin manner.



MPI Parcelport: Synchronization

Isends/Irecvs are posted one by one

Request pools are checked in a round-robin manner.

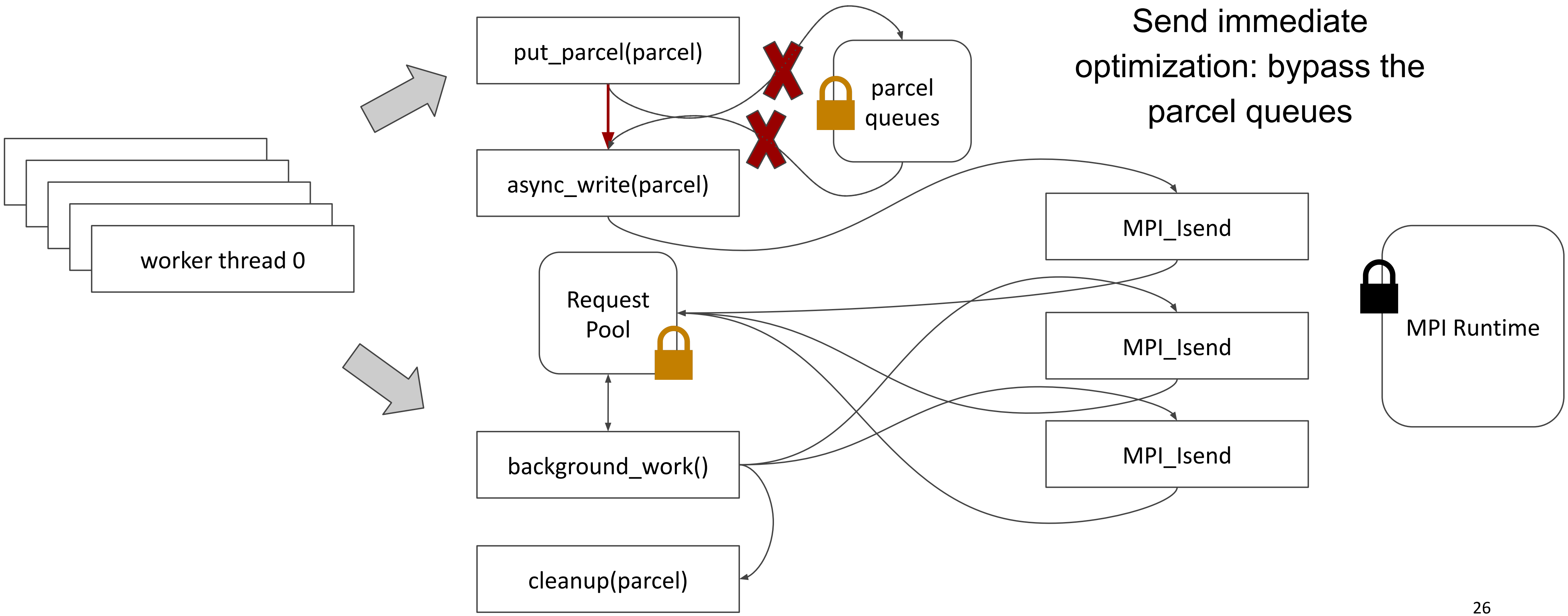


MPI Parcelport: Threads

All worker threads can call `put_parcel`

All worker threads can call `background_work`

Send immediate optimization: bypass the parcel queues

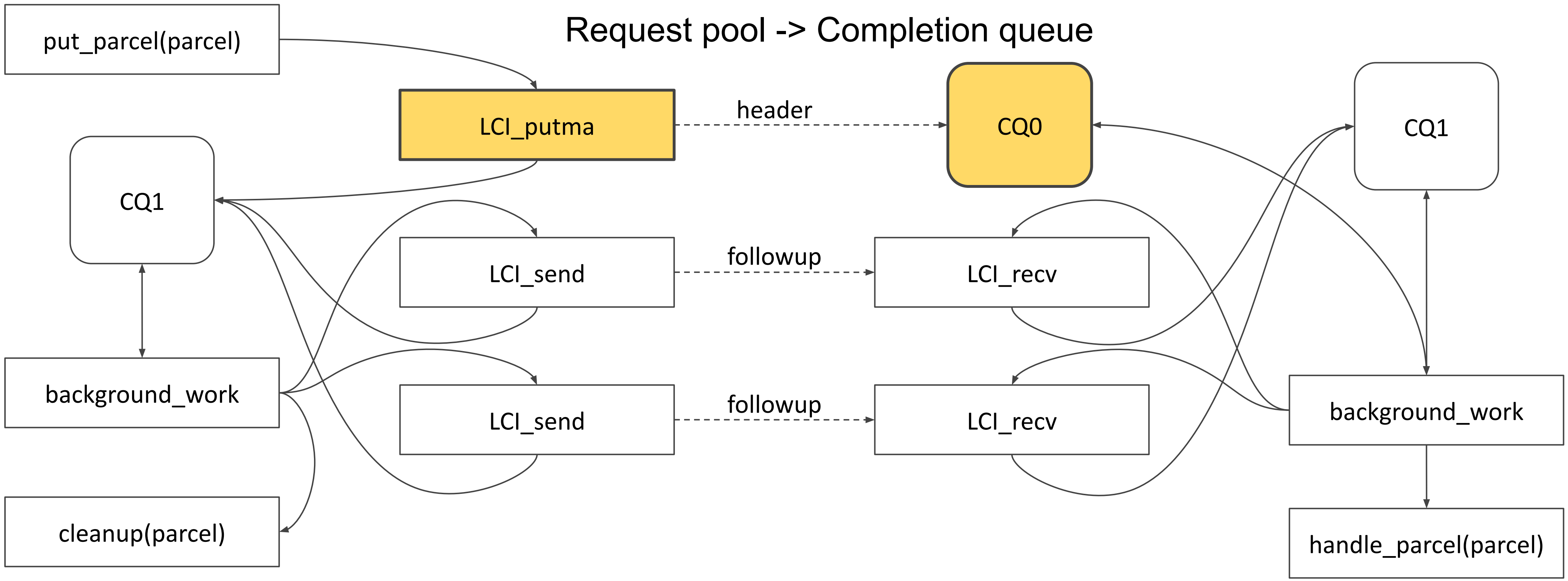


LCI Parcelport

Use LCI_putma (eager put w. remote signal & target buffer allocation) for header message.

Use LCI_send/recv for follow-up messages

Request pool -> Completion queue

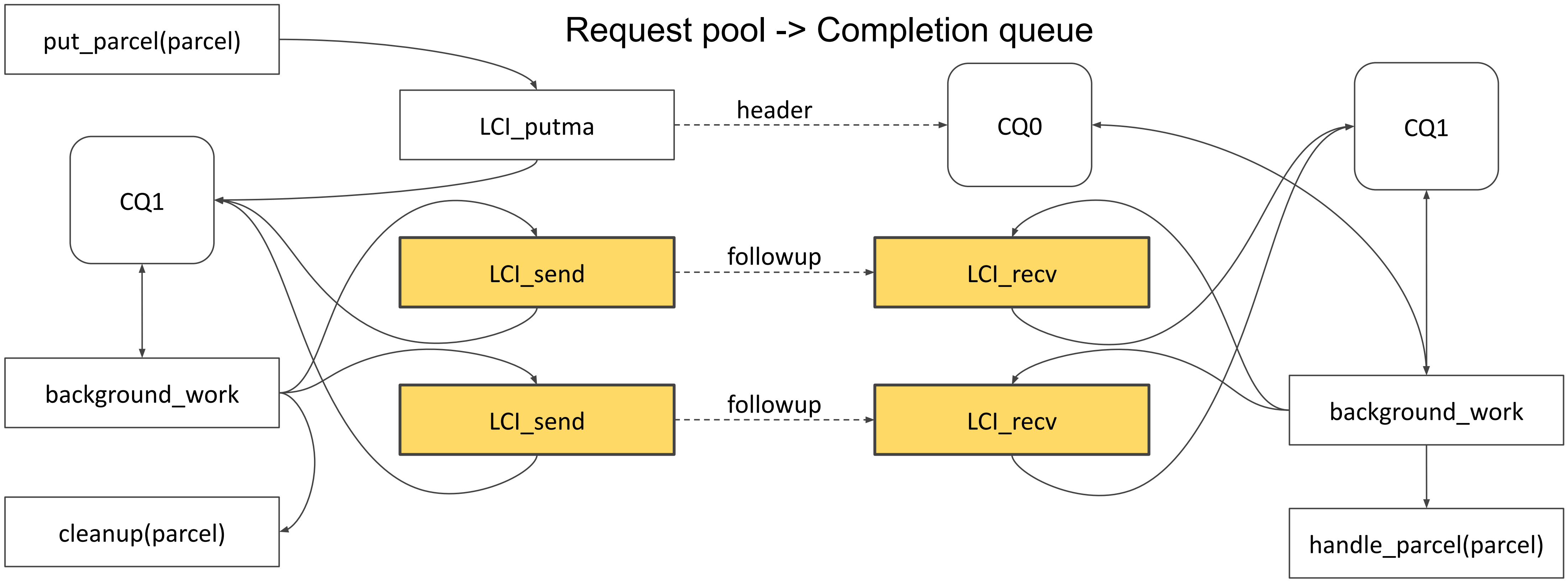


LCI Parcelport

Use LCI_putma (eager put w. remote signal & target buffer allocation) for header message.

Use LCI_send/recv for follow-up messages

Request pool -> Completion queue

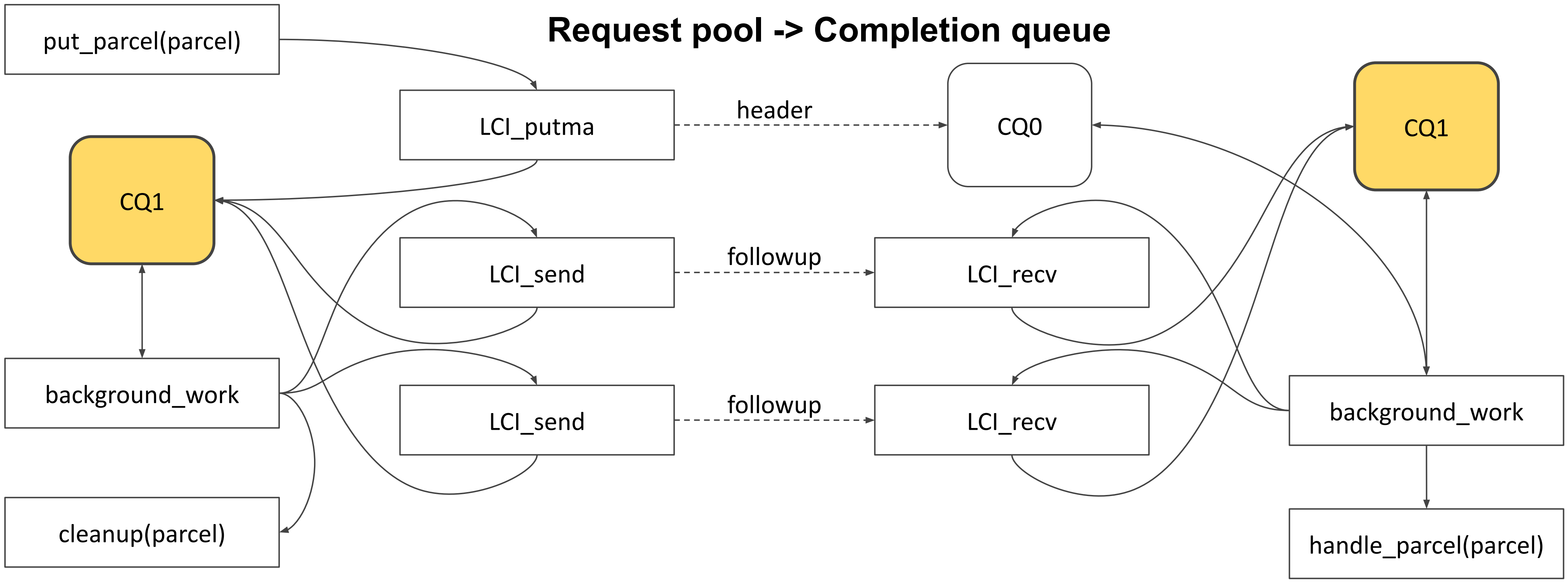


LCI Parcelport

Use LCI_putma (eager put w. remote signal & target buffer allocation) for header message.

Use LCI_send/recv for follow-up messages

Request pool -> Completion queue

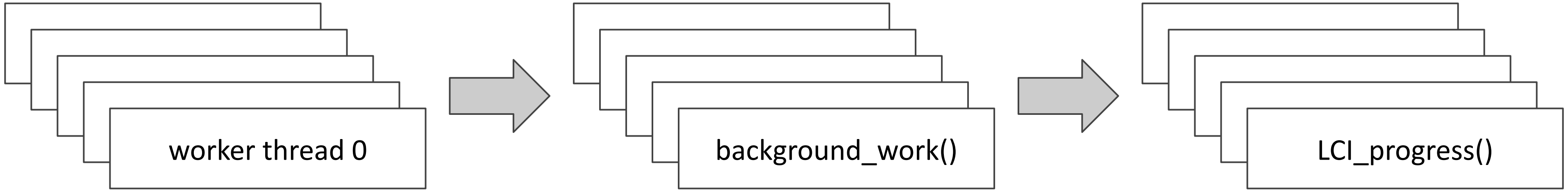


LCI Parcelport: LCI_progress

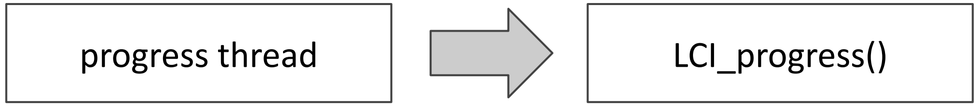
Explicit LCI_progress function.

Two options

All workers call LCI_progress()



Dedicated progress thread



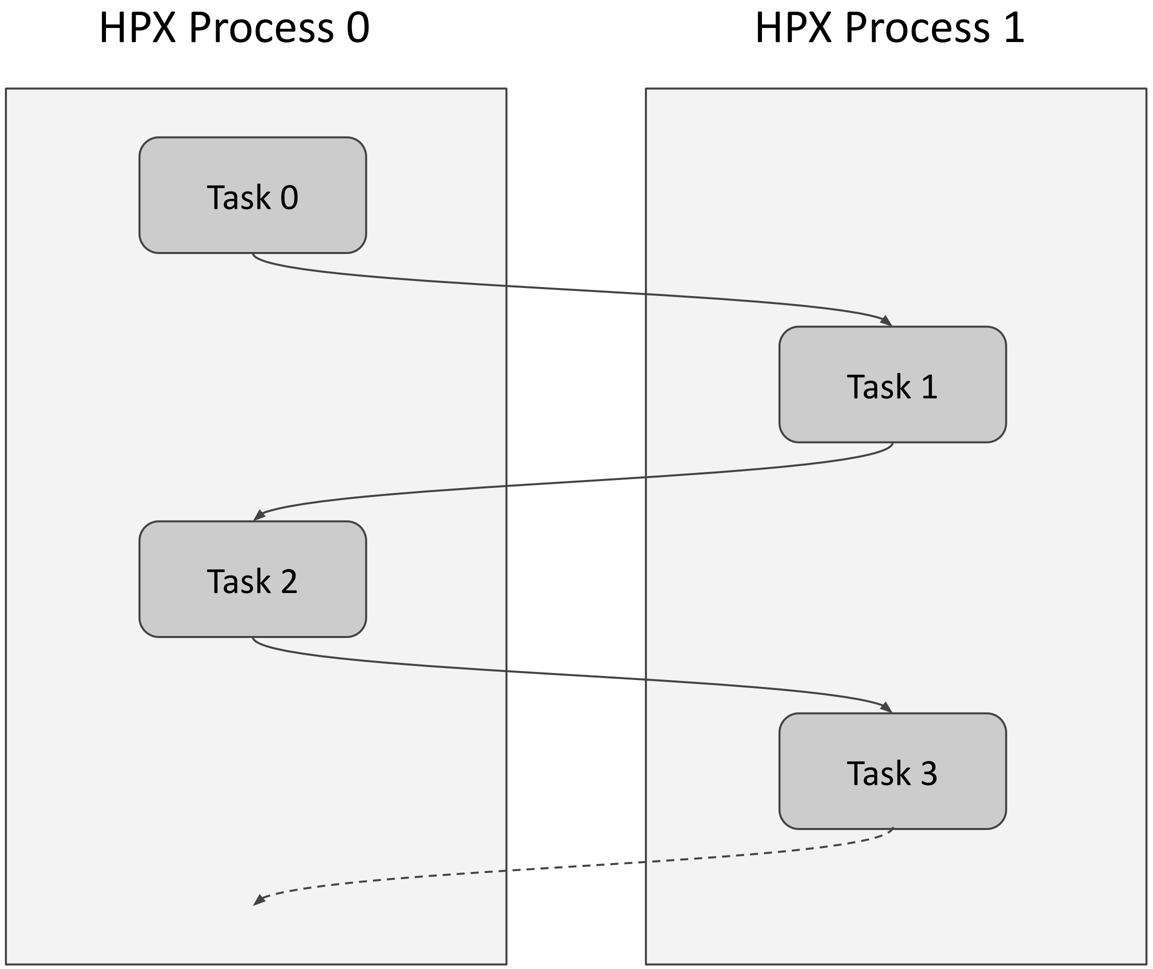
Evaluation

SDSC Expanse

- CPU
  - AMD EPYC 7742 64-Core Processor
  - 2 sockets, 128 cores per node
- Memory
  - 256 GB, DDR4
- NIC
  - Mellanox ConnectX-6
- Interconnect
  - HDR InfiniBand
  - 2x50 Gbps
- Software:
  - OpenMPI 4.1.5
  - UCX 1.14.0

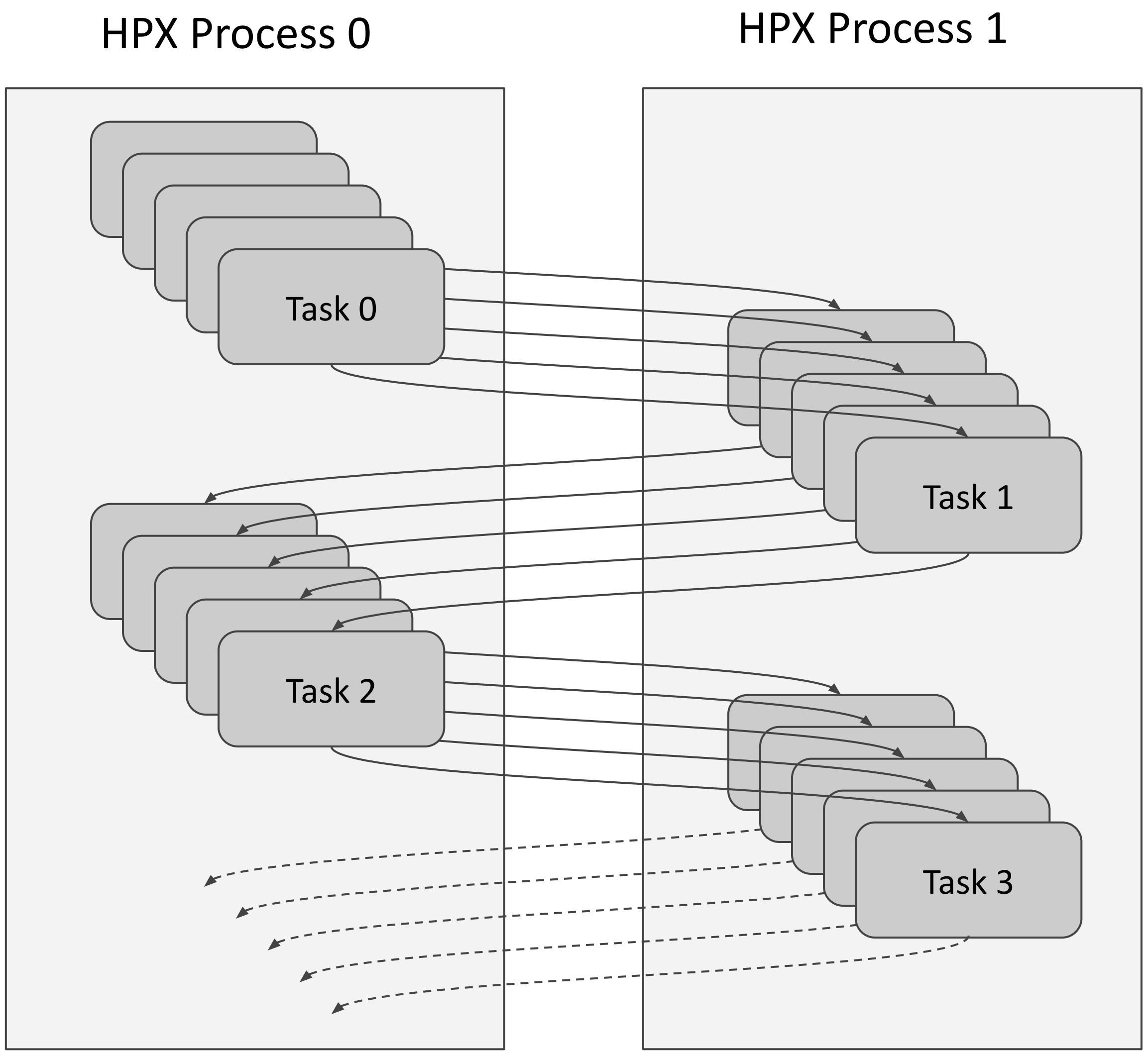
HPX Microbenchmarks

Task Chain



HPX Microbenchmarks

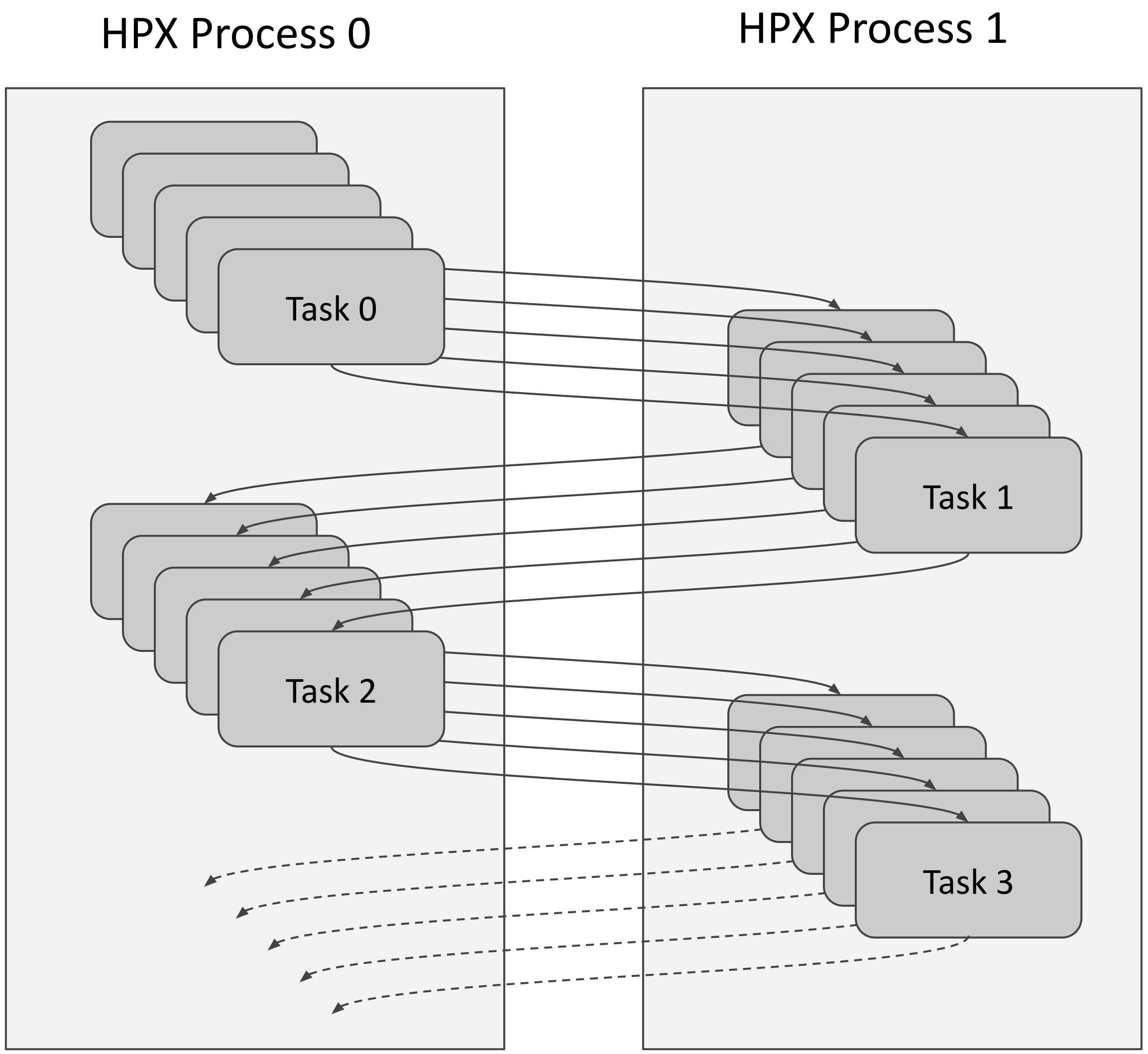
Multiple Chains of Tasks



HPX Microbenchmarks

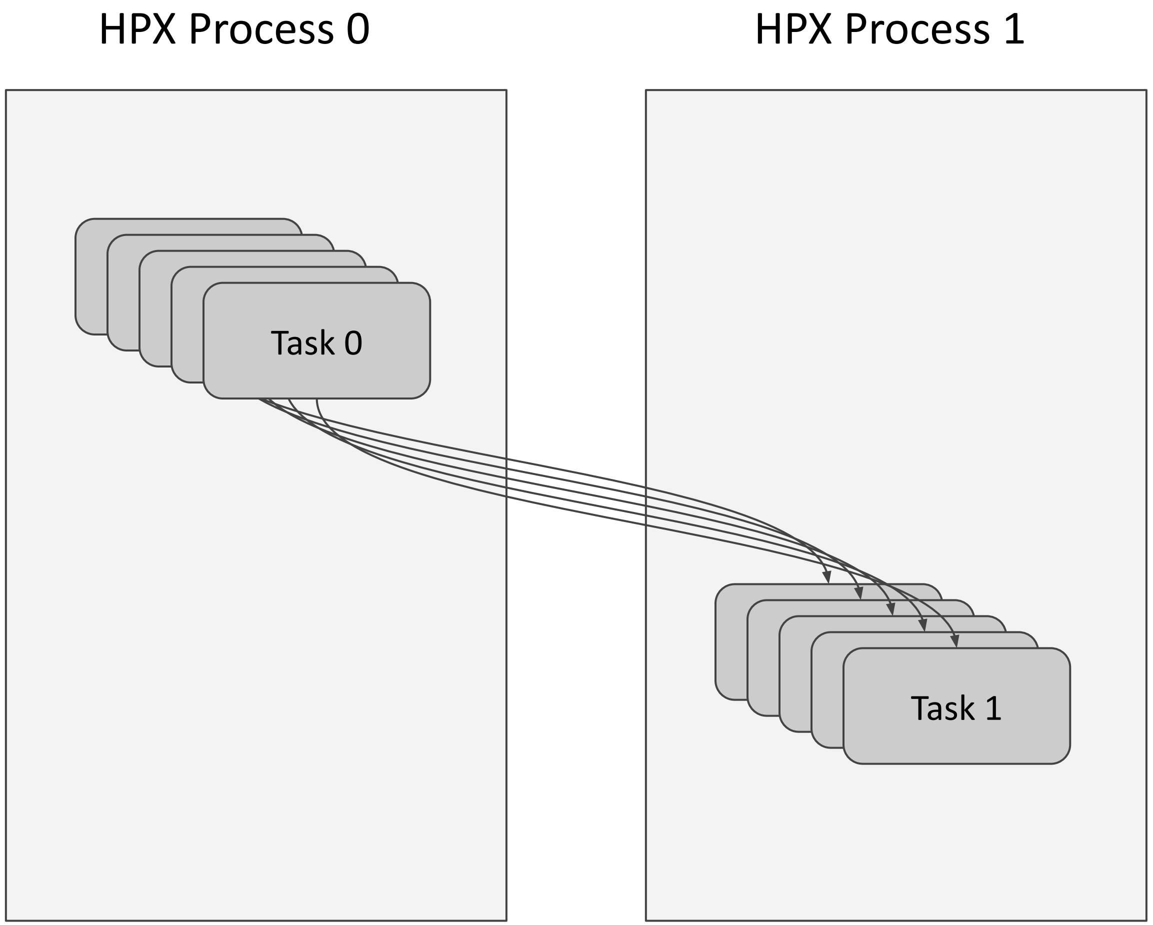
Multiple Chains of Tasks

- number of chains
- length of chains
- message size
- injection rate

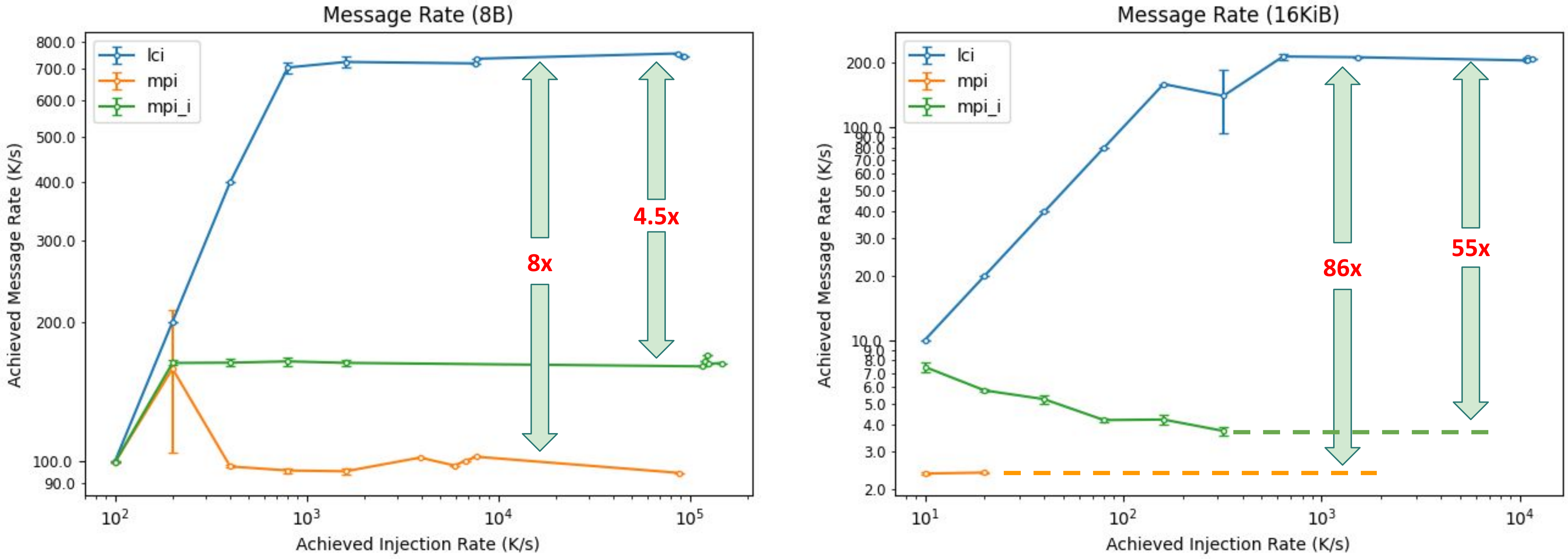


HPX Microbenchmark: Message Rate

- a very large number of chains
- chain length = 2
- varying message size
- varying injection rate



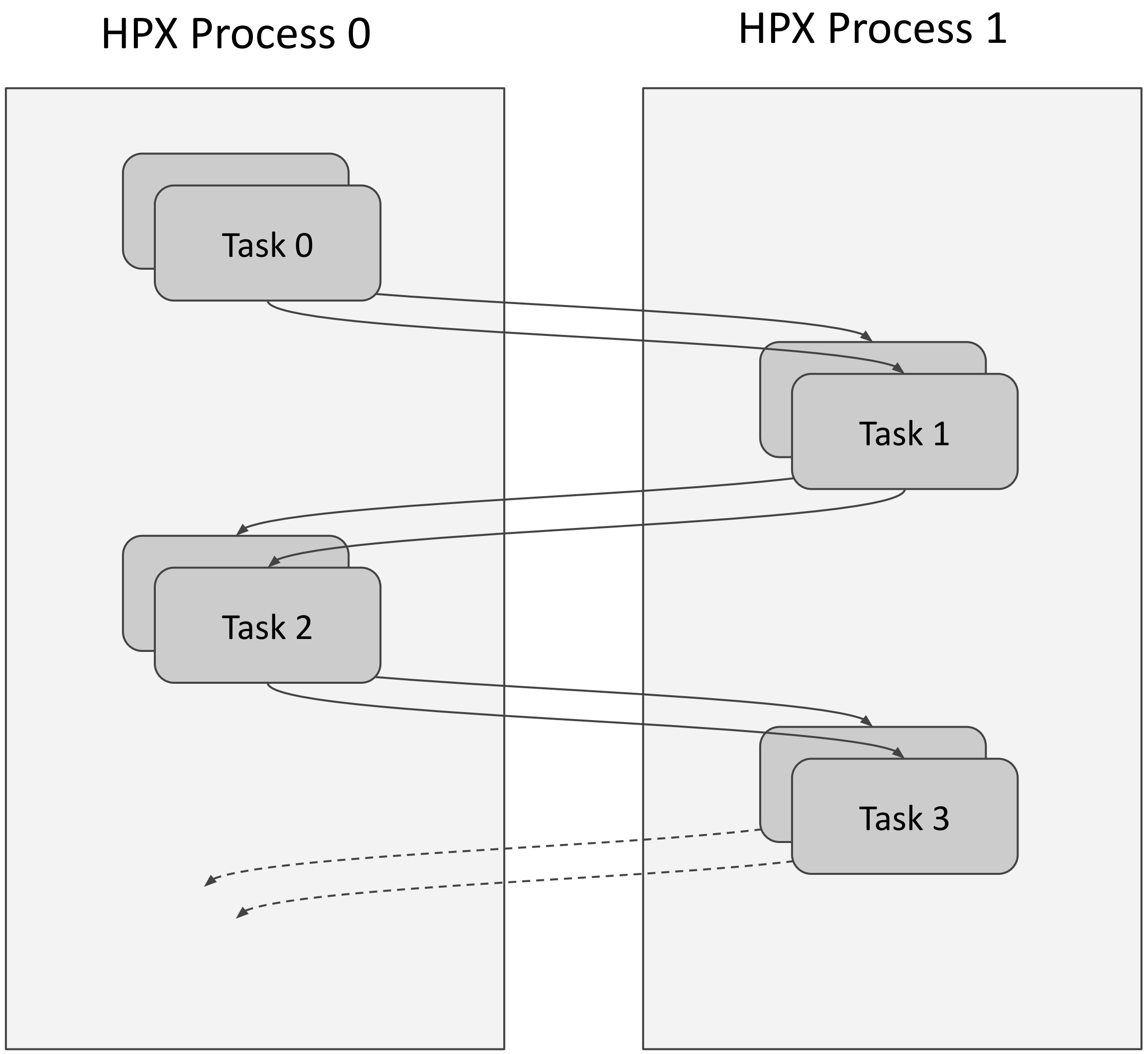
LCI parcelport v.s. MPI parcelport



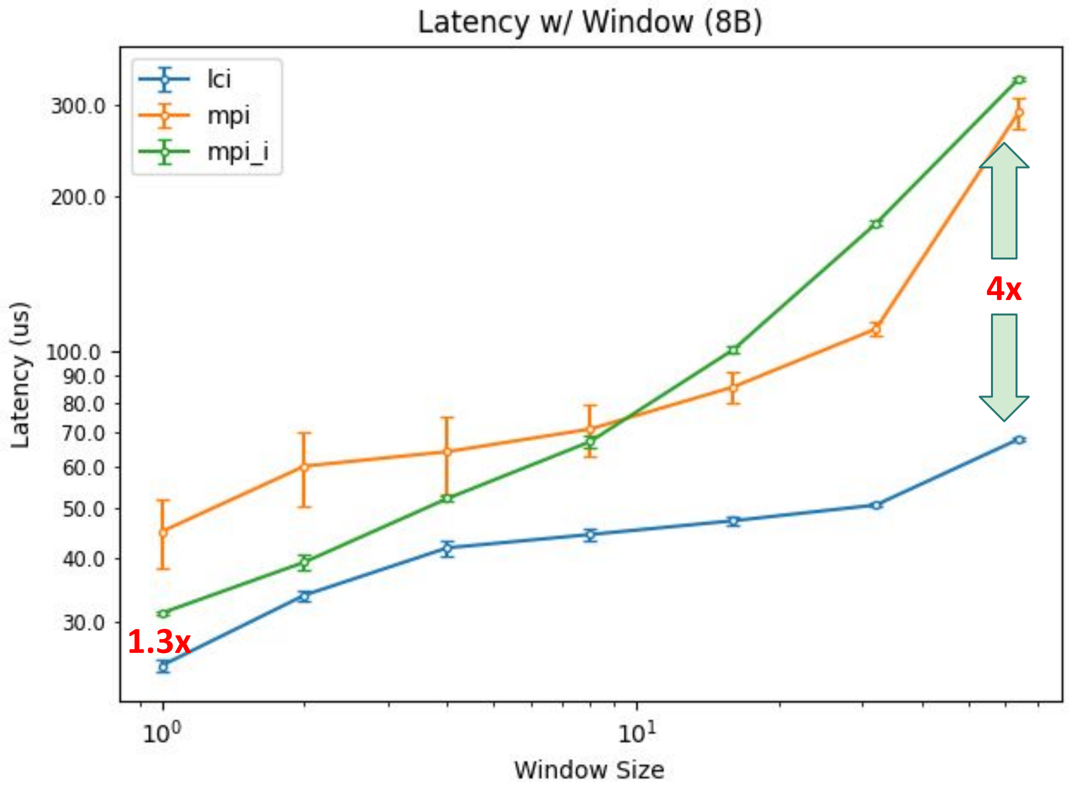
lci	The LCI parcelport with the send immediate optimization (bypass the parcel queues)
mpi	The default MPI parcelport (with the parcel queues)
mpi_i	The MPI parcelport with the send immediate optimization (bypass the parcel queues)

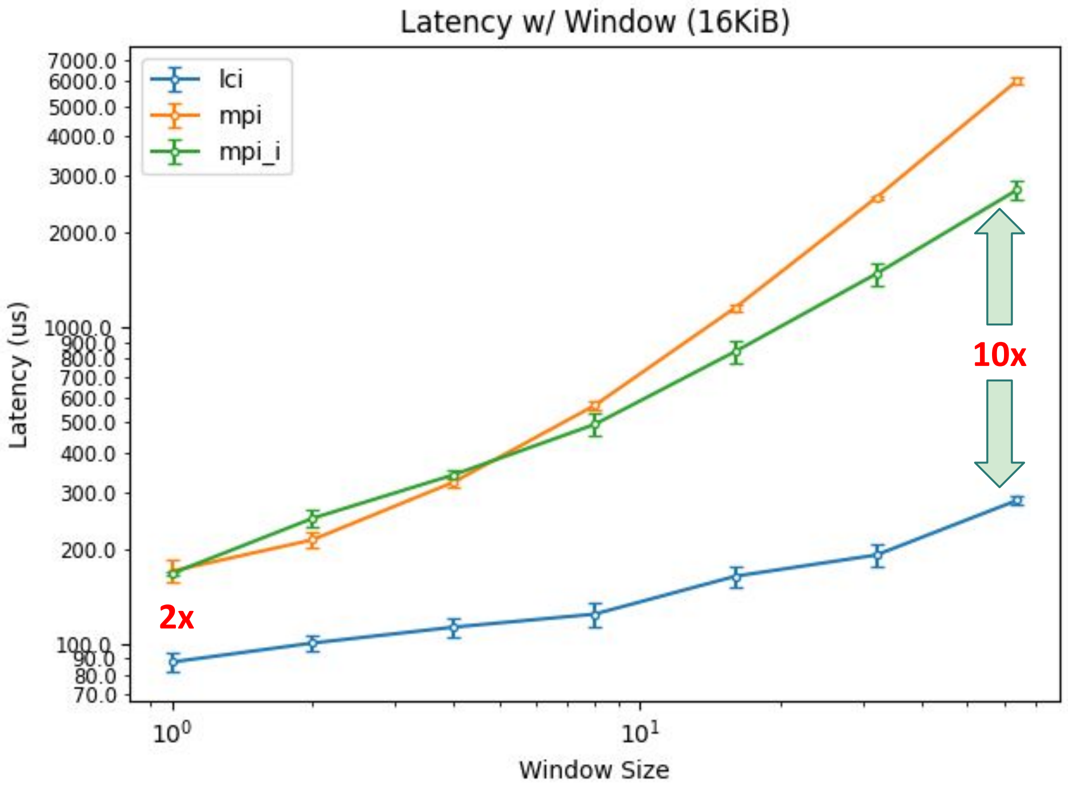
HPX Microbenchmark: Latency

- a small number of chains
- very large chain length
- varying message size



LCI parcelport v.s. MPI parcelport





lci	The LCI parcelport with the send immediate optimization (bypass the parcel queues)
mpi	The default MPI parcelport (with the parcel queues)
mpi_i	The MPI parcelport with the send immediate optimization (bypass the parcel queues)

Application-level benchmark: Octo-Tiger

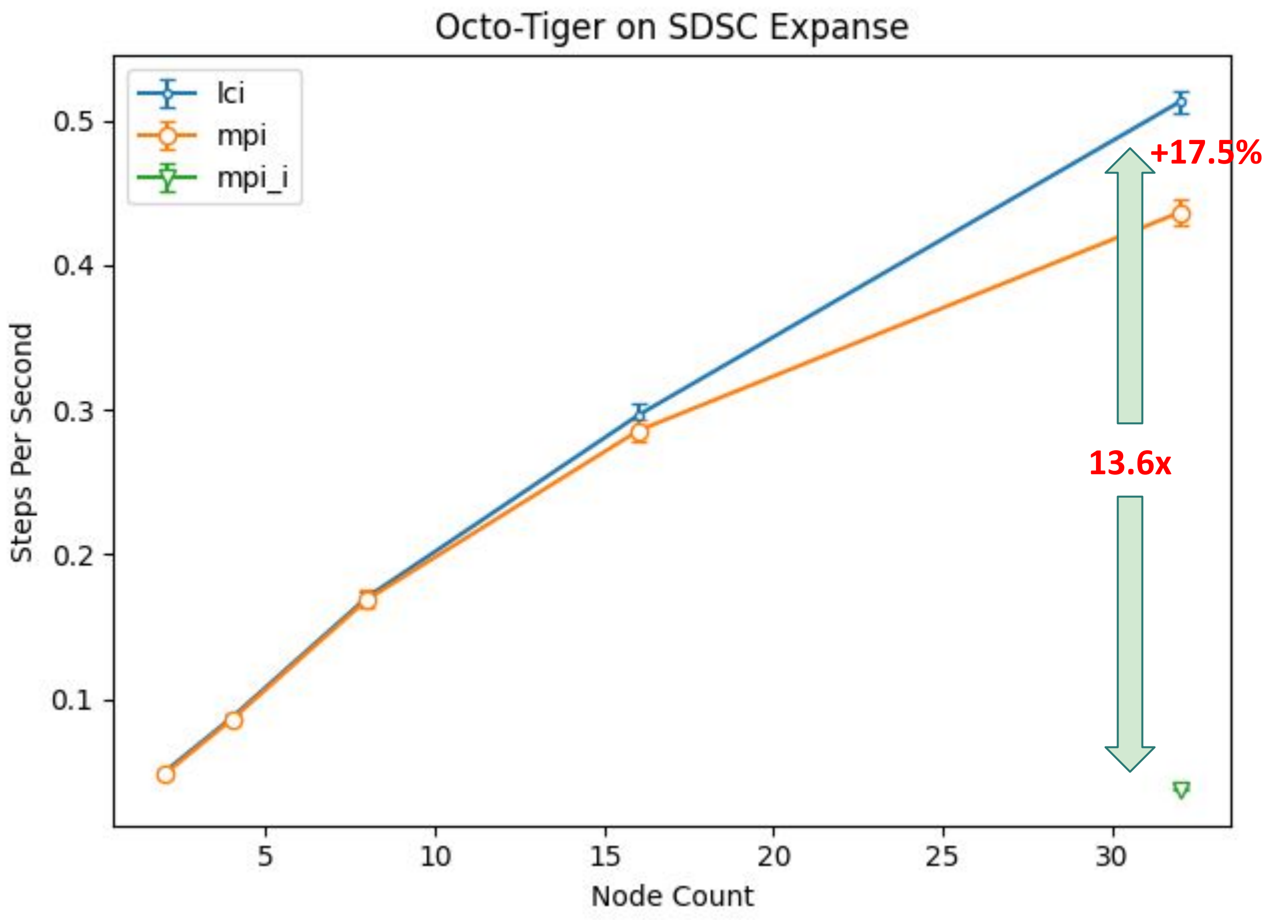
- Astrophysics program simulating the evolution of star systems
- Based on fast multipole method on adaptive Octrees
- Implemented on top of HPX.

[1] Octotiger GitHub repo: https://github.com/STEllAR-GROUP/octotiger
[2] Dominic C. Marcello, Sagiv Shiber, Orsola De Marco, Juhan Frank, Geoffrey C. Clayton, Patrick M. Motl, Patrick Diehl, Hartmut Kaiser, "Octo-Tiger: A New, 3D Hydrodynamic Code for Stellar Mergers that uses HPX Parallelisation", accepted for publication in the Monthly Notices of the Royal Astronomical Society, 2021

Strong Scaling

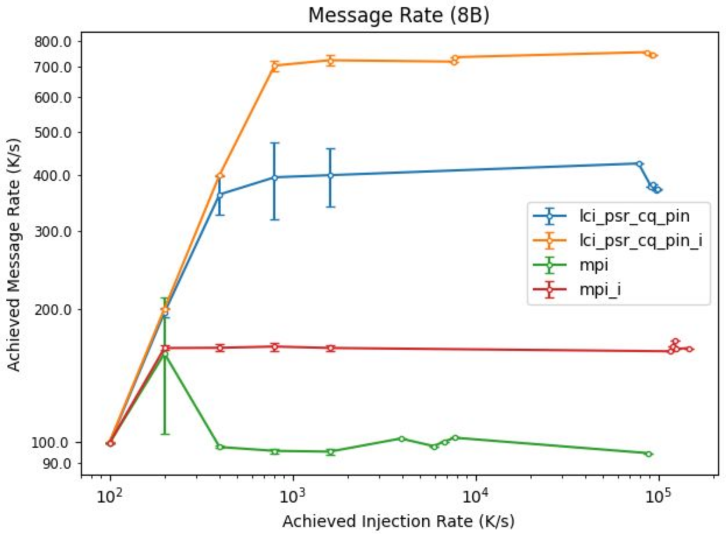
Problem Size:

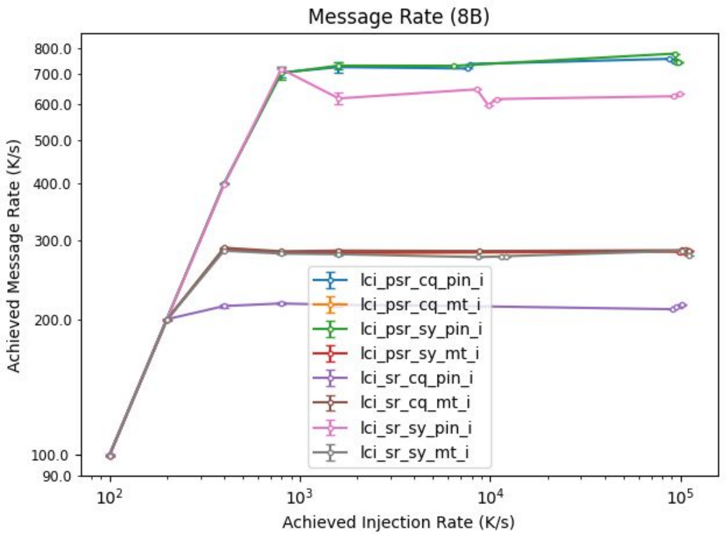
- Max tree level: 6.
- Initial tree size: 31817 nodes with 27840 leaves.
- Subgrid size per tree node: 8x8x8
- Iteration: 5.



lci	The LCI parcelport with the send immediate optimization (bypass the parcel queues)
mpi	The default MPI parcelport (with the parcel queues)
mpi_i	The MPI parcelport with the send immediate optimization (bypass the parcel queues)

More in the paper…





lci_psr_cq_pin
lci_psr_cq_pin_i
lci_psr_cq_mt_i
lci_psr_sy_pin_i
lci_psr_sy_mt_i
lci_sr_cq_pin_i
lci_sr_cq_mt_i
lci_sr_sy_pin_i
lci_sr_sy_mt_i
mpi
mpi_i

The send immediate optimizations
Communication Primitives
Completion Mechanisms
Progress Engine

sr	send+recv
psr	put+send+recv
sy	synchronizer pools
cq	completion queue
pin	dedicated progress thread
mt	all worker threads making progress
i	enable send immediate optimization

Lessons Learned

- The LCI parcelport outperforms the MPI parcelport, by a large margin.
- **Message Aggregation** (the parcel queues) yields mixed results.
- Using a **dedicated progress thread** is almost always helpful.
- Polling one **completion queue** is preferable to polling multiple requests or synchronizers.
- A **put with a remote completion** (queue) signal achieves better performance than send-recv at high small-message rates

Lessons Learned

- The LCI parcelport outperforms the MPI parcelport, by a large margin.
- **Message Aggregation** (the parcel queues) yields mixed results.
- Using a **dedicated progress thread** is almost always helpful.
- Polling one **completion queue** is preferable to polling multiple requests or synchronizers.
- A **put with a remote completion** (queue) signal achieves better performance than send-recv at high small-message rates

Design and Analysis of the Network Software Stack of an Asynchronous Many-Task System
-- The LCI Parcelport of HPX

Jiakun Yan, Hartmut Kaiser, Marc Snir

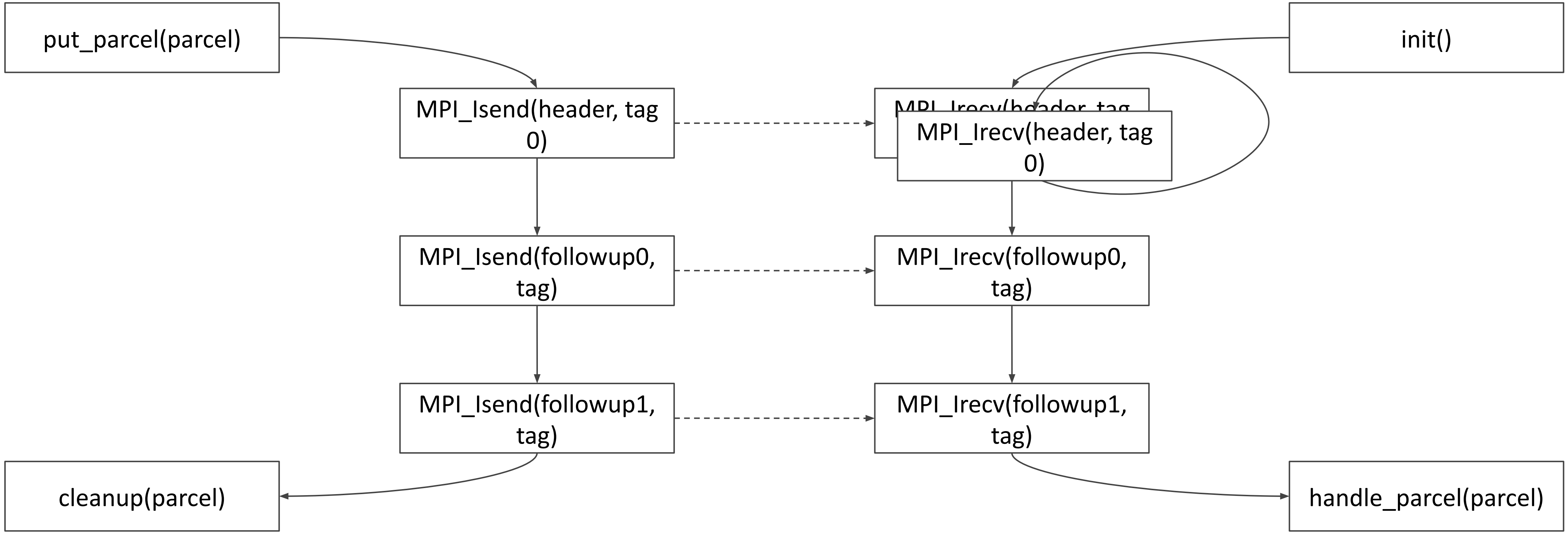
Questions: jiakuny3@illinois.edu

DOI: 10.1145/3624062.3624598
HPX GitHub: https://github.com/STEllAR-GROUP/hpx
LCI GitHub: https://github.com/uiuc-hpc/LC

MPI Parcelport: Communication Primitives

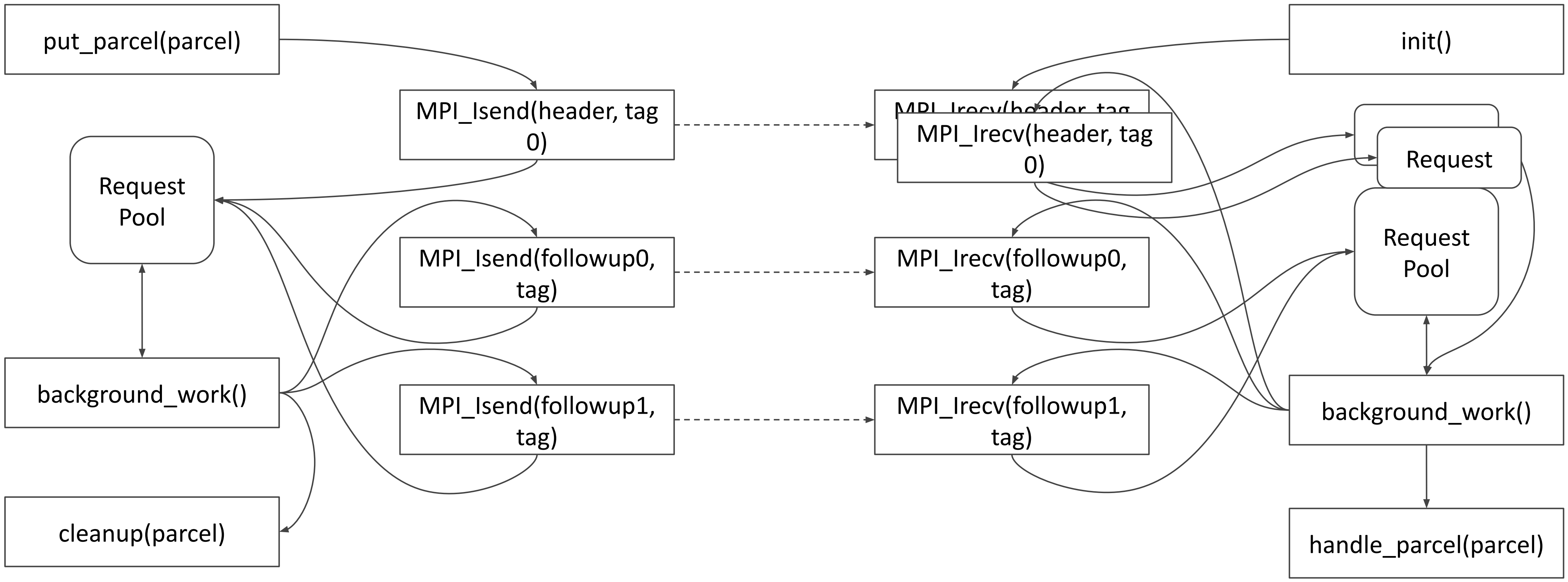
Prepost one recv with tag 0

All messages are transferred with MPI_Isend/Irecv



MPI Parcelport: Synchronization

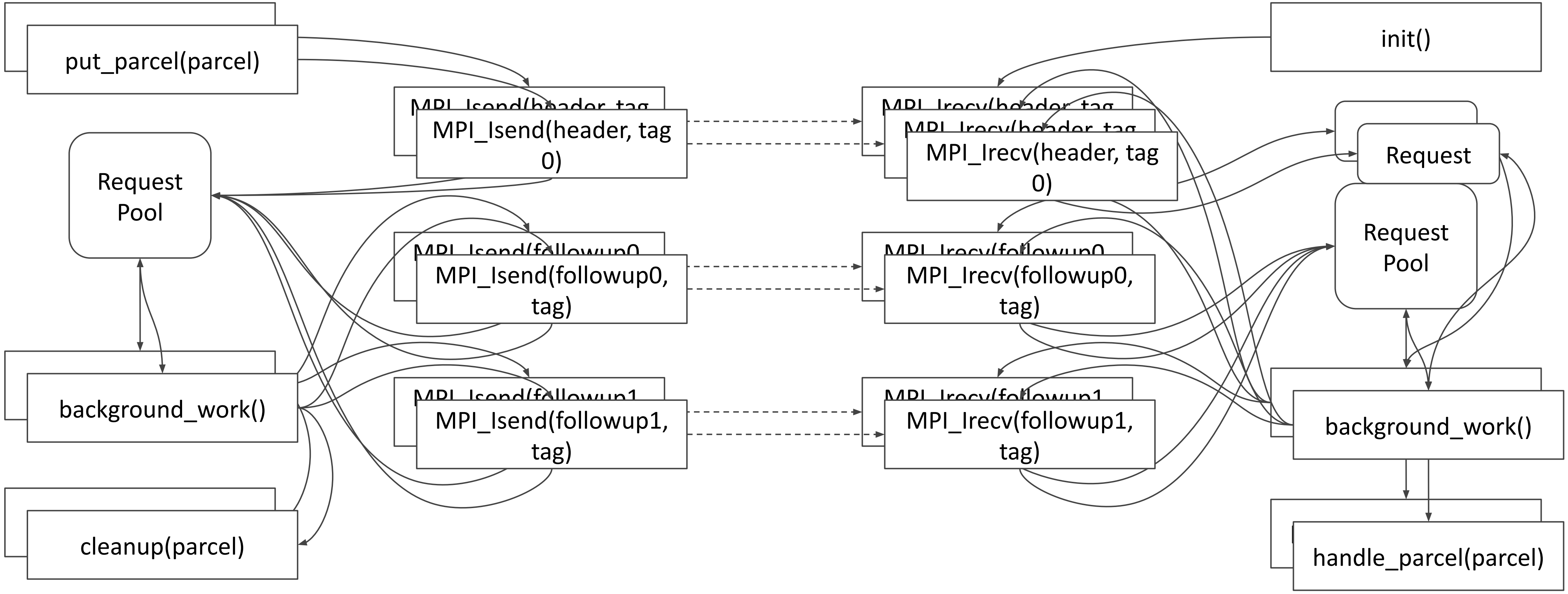
Request pools are checked in a round-robin manner.



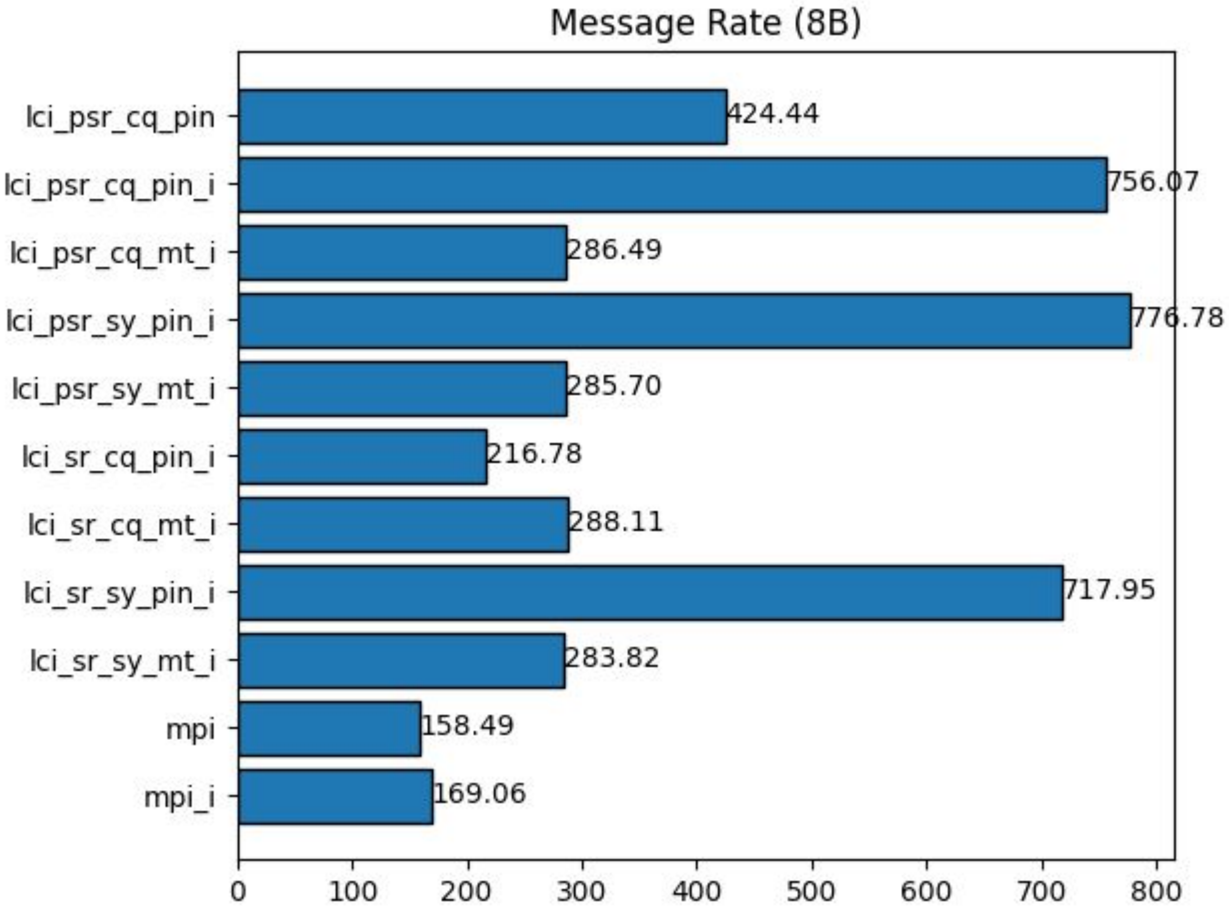
MPI Parcelport: Concurrency

Only one pending low-level message send/recv is allowed per HPX message.

Multiple HPX messages can be sent simultaneously.



E.g. Message Rate (8-byte)



sr	send+recv
psr	put+send+recv
sy	synchronizer pools
cq	completion queue
pin	dedicated progress thread
mt	all worker threads making progress
i	enable send immediate optimization

Conclusion

- Changes in architectures and problems call for new programming model: Asynchronous Many-Task Systems.
- The performance of task systems heavily relies on the performance of the communication subsystem.
- This paper focus on
  - Detailed design and implementation of the communication stack of an AMT: HPX.
  - Various design decisions and optimizations of HPX’s new communication backend: the LCI parcelport.
  - The performance evaluation of
    - the LCI parcelport v.s. the MPI parcelport.
    - the design decisions of the LCI parcelport.

Conclusion

Design and Analysis of the Network Software Stack of an Asynchronous Many-Task System -- The LCI Parcelport of HPX

Jiakun Yan, Hartmut Kaiser, Marc Snir

Questions: jiakuny3@illinois.edu

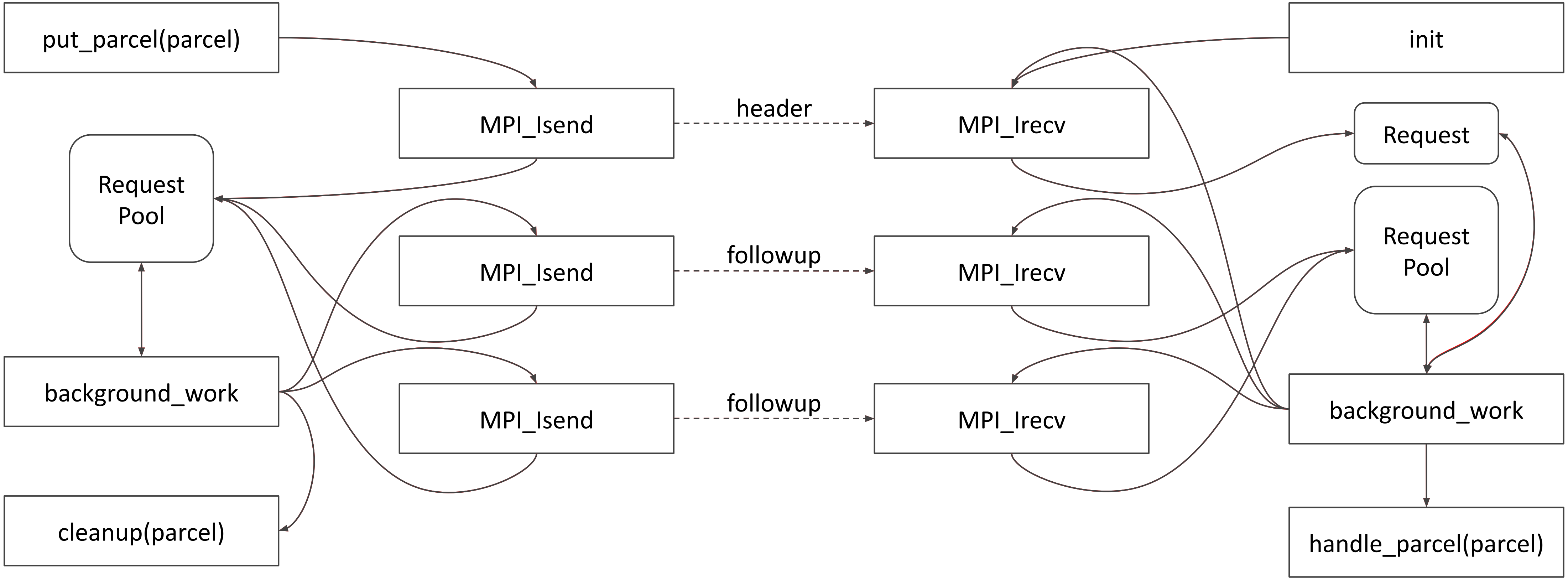
DOI: 10.1145/3624062.3624598

HPX GitHub: https://github.com/STEllAR-GROUP/hpx
LCI GitHub: https://github.com/uiuc-hpc/LC

MPI Parcelport: Communication Primitives

Prepost one recv with tag 0

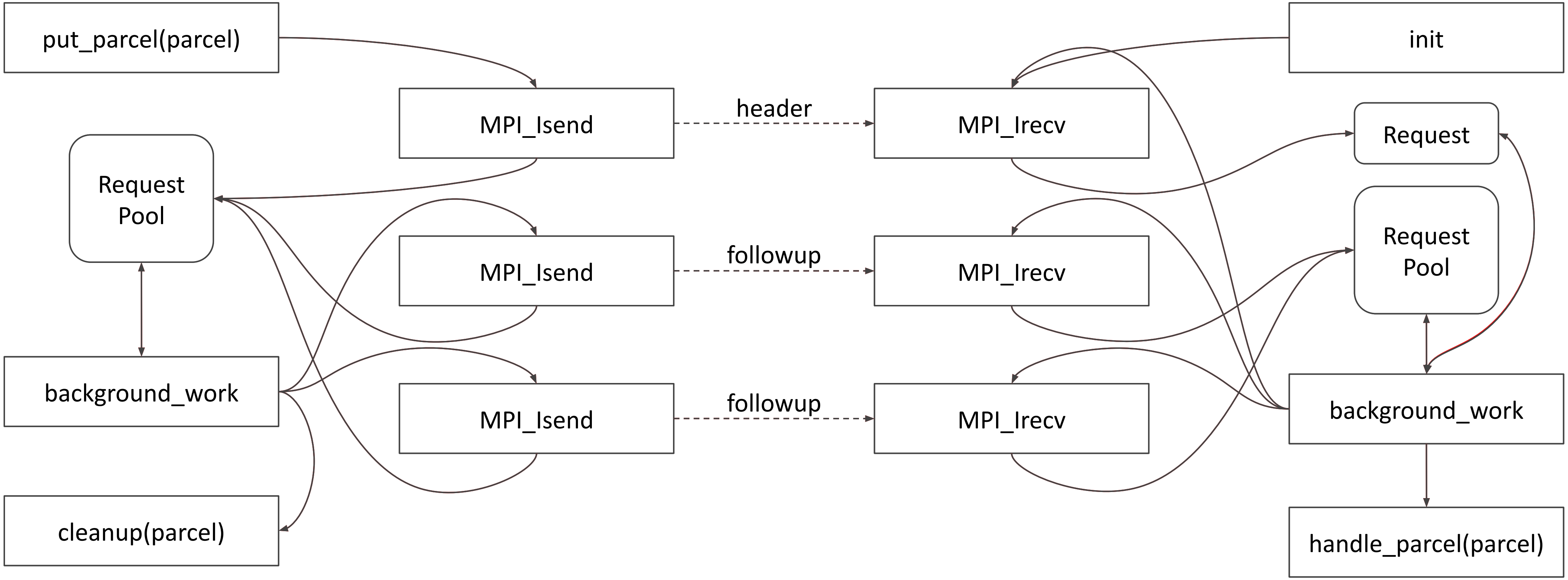
All messages are transferred with MPI_Isend/Irecv



MPI Parcelport: Synchronization

Isends/Irecvs are posted one by one

Request pools are checked in a round-robin manner.



MPI Parcelport: Concurrency

Multiple HPX messages can be sent simultaneously.