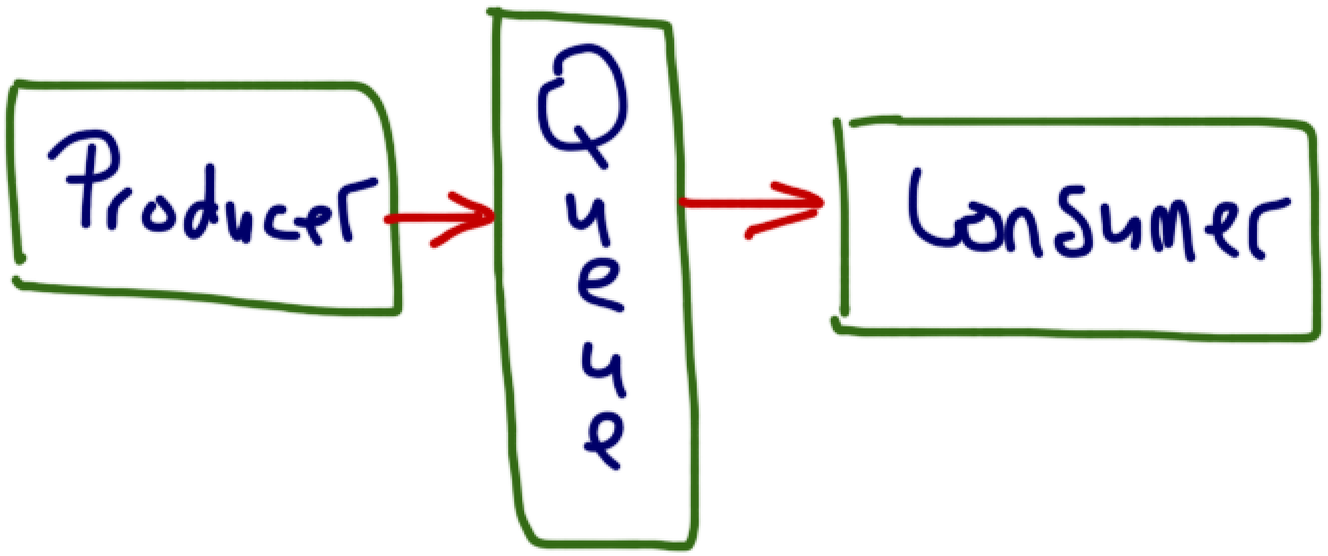
One Sided Notification for the Producer/Consumer Pattern

Torsten Hoefler,
ETH Zurich, Switzerland
Data by Roberto Belli

Producer/Consumer Synchronization

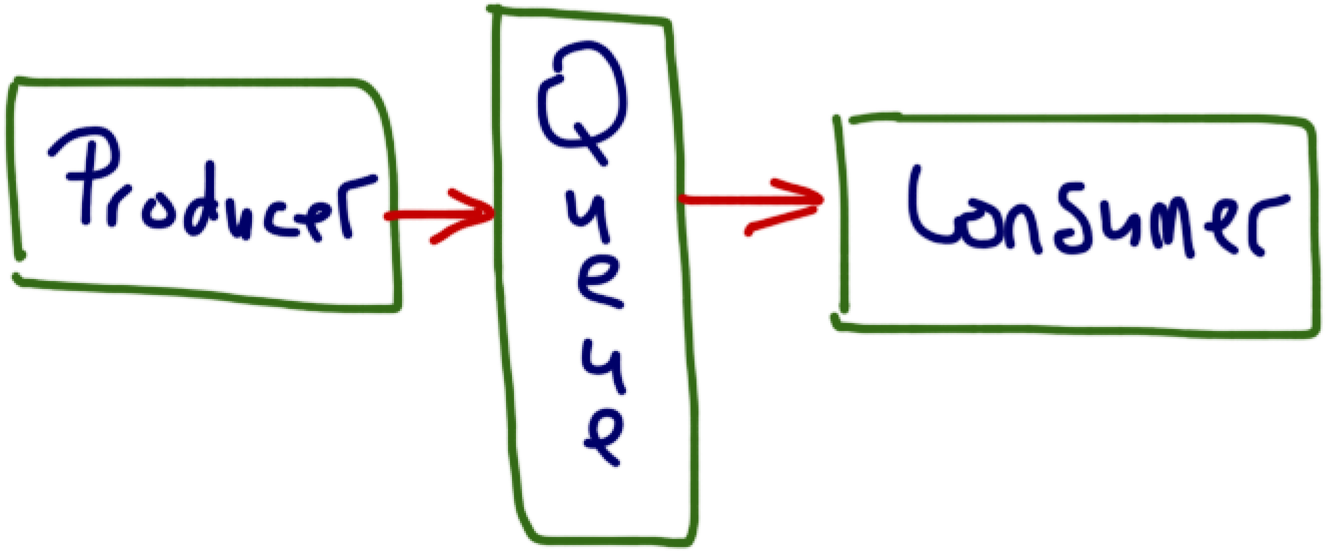


```
while(true) {
  produce()
  MPI_Send(prod, 1, INT, 1, 99, comm)
}
```

```
while(true) {
  MPI_Recv(prod, 1, INT, 1, 99, comm, stat)
  consume()
}
```

Producer/Consumer Synchronization in RMA

- **Active synchronization: fence**

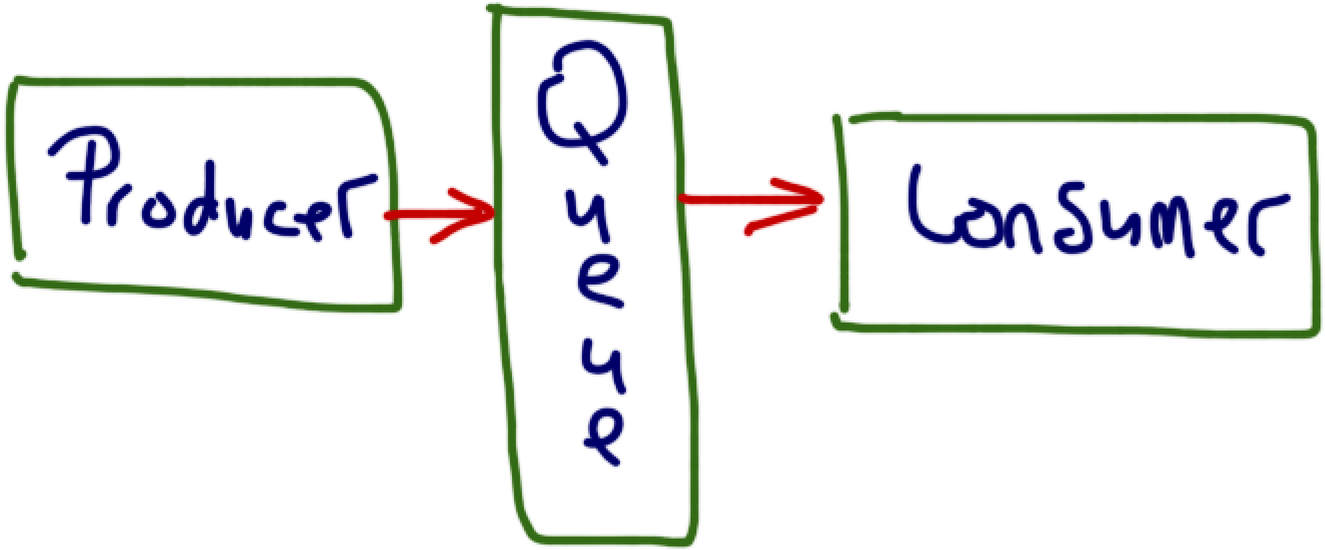


```
while(true) {
  produce()
  MPI_Put(prod, 1, INT, 1, 0, 1, INT, win)
  MPI_Win_fence(win);
}
```

```
while(true) {
  MPI_Win_fence(win)
  consume()
}
```

Producer/Consumer Synchronization in RMA

- **Active synchronization: general active target**

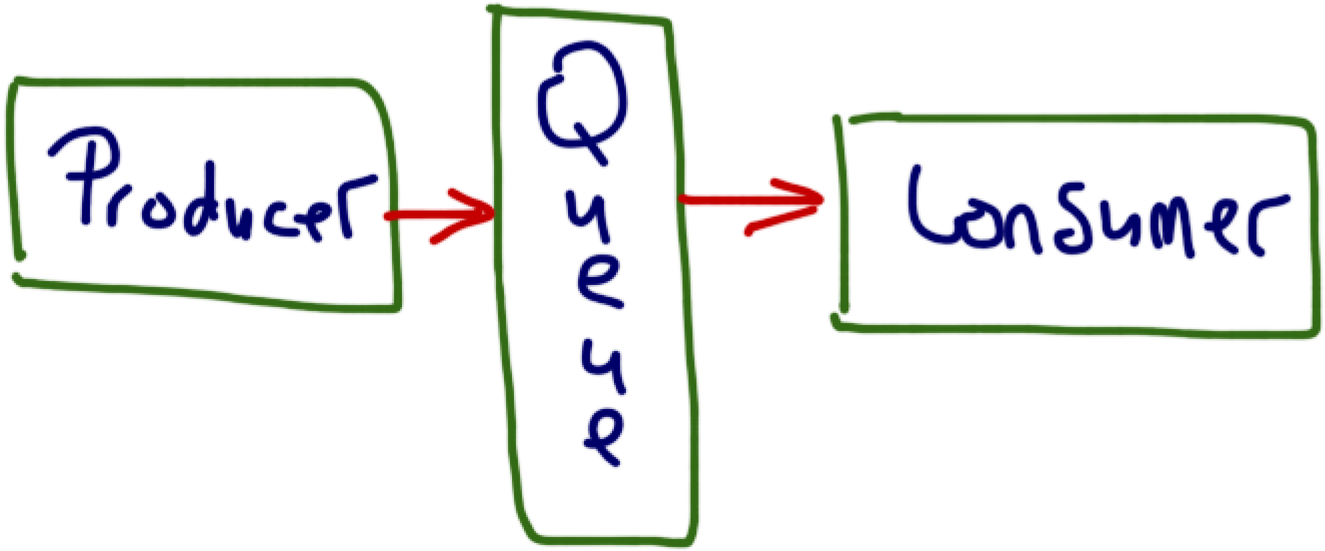


```
while(true) {
  produce()
  MPI_Win_start(grp, 0, win)
  MPI_Put(prod, 1, INT, 1, 0, 1, INT, win)
  MPI_Win_complete(win)
}
```

```
while(true) {
 MPI_Win_post(grp, 0, win)   consume()
 MPI_Win_wait(win)
 consume()
}
```

Producer/Consumer Synchronization in RMA

- **Passive synchronization: lock/unlock with barrier**

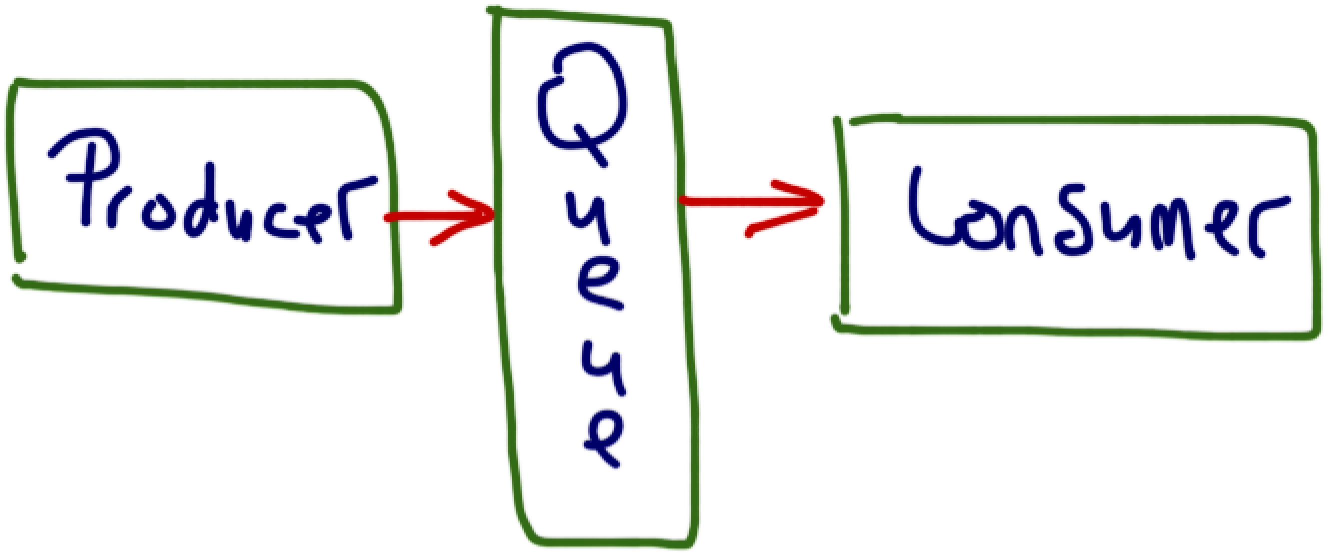


```
while(true) {
  produce()
  MPI_Win_lock(excl, 0, 0, win)
  MPI_Put(prod, 1, INT, 1, 0, 1, INT, win)
  MPI_Win_unlock(0,win)
  MPI_Barrier()
}
```

```
while(true) {
  MPI_Barrier()
  MPI_Win_lock(excl, 0, 0, win)
  consume()
  MPI_Win_unlock(0,win)
}
```

Producer/Consumer Synchronization in RMA

- **Passive synchronization: lock/unlock with send/recv**

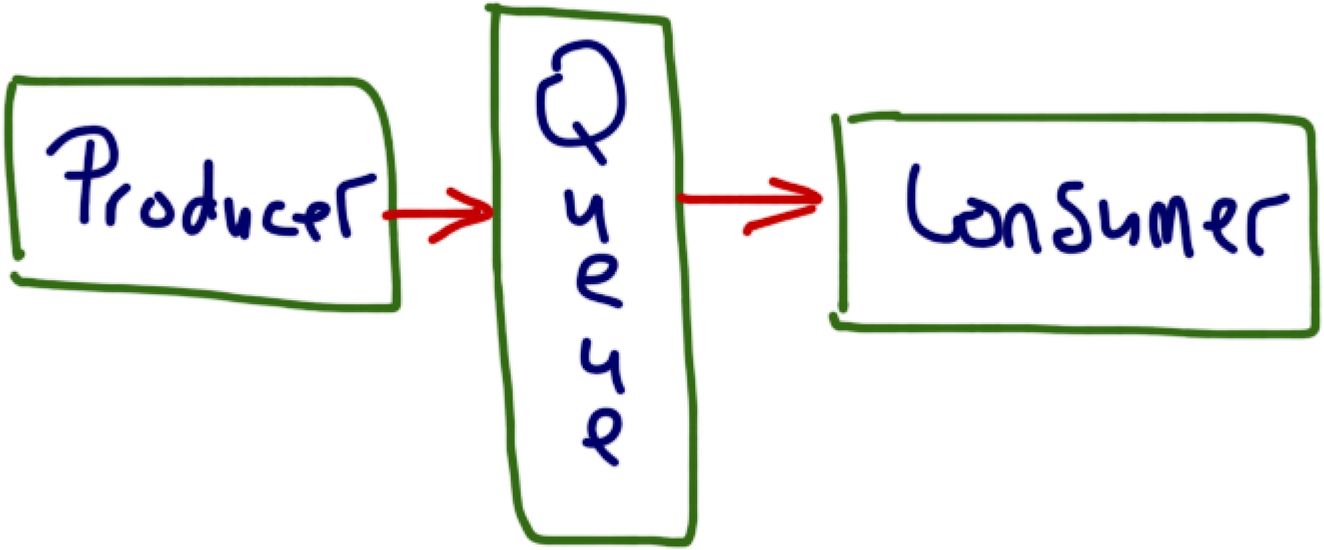


```
while(true) {
  produce()
  MPI_Win_lock(excl, 0, 0, win)
  MPI_Put(prod, 1, INT, 1, 0, 1, INT, win)
  MPI_Win_unlock(0,win)
  MPI_Send(flag, 1, INT, 1, 99, comm)
}
```

```
while(true) {
  MPI_Recv(flag, 1, INT, 1, 99, comm, stat)
  MPI_Win_lock(excl, 0, 0, win)
  consume()
  MPI_Win_unlock(0,win)}
```

Producer/Consumer Synchronization in RMA

- **Passive synchronization: lock_all with send/recv**



```
MPI_Win_lock_all(0, win)
while(true) {
  produce()
  MPI_Put(prod, 1, INT, 1, 0, 1, INT, win)
  MPI_Win_flush(0, win)
  MPI_Send(flag, 1, INT, 1, 99, comm)
}
MPI_Win_unlock_all(win)
```

```
while(true) {
  MPI_Recv(flag, 1, INT, 1, 99, comm, stat)
  MPI_Win_lock(excl, 0, 0, win)
  consume()
  MPI_Win_unlock(0,win)
}
```

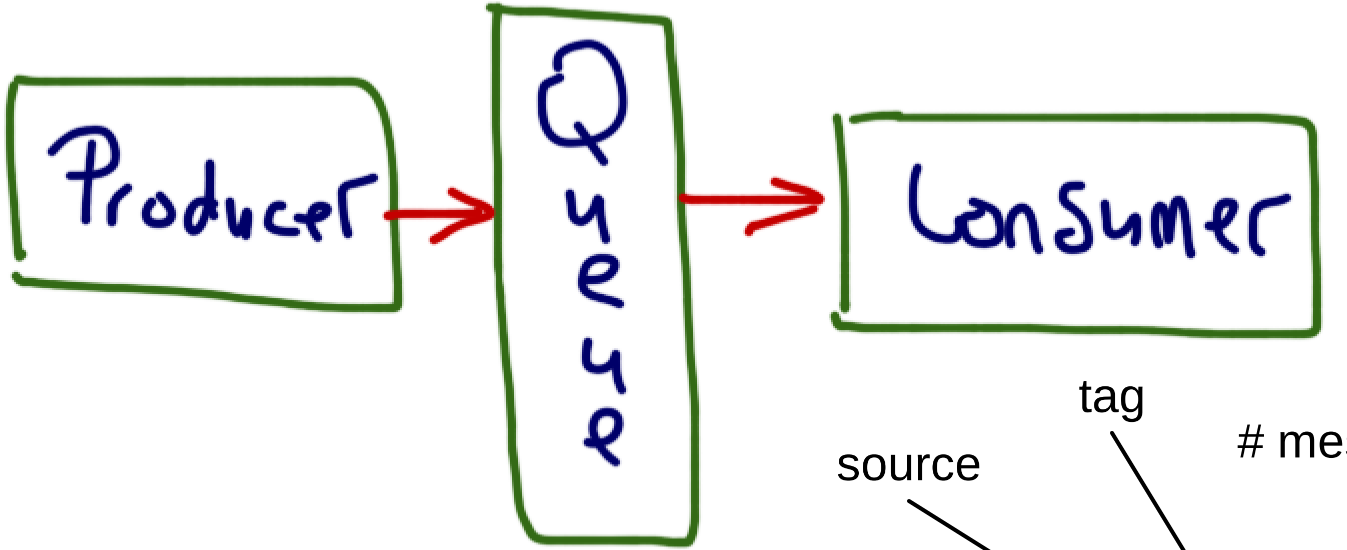
What do we want for a Prod/Cons Pattern

- **Something in between active and passive synchronization**
  - Not the full exposure epoch concept for active (too heavy)
  - Yet, have remote involvement (notification) at the destination
  - Destination is **not** needed to make progress

- **Half-passive synchronization: remote notifications**
  - Re-using MPI requests – similar to persistent communication
    *MPI_Start(MPI_Request *request)*
  - One-sided use-case often involves multiple transfers for a logical message
    We need a counter (well, less MPI-ish)
    What do we count??
  - Counting messages!
    Could also count elements, but that would involve datatypes? Possible!
  - Matching like MPI messages (src, tag)-tuple
    Re-using existing concepts

Producer/Consumer Synchronization in RMA

- **Passive synchronization: lock_all with send/recv**



source
tag
# messages

```
while(true) {
  produce()
  MPI_Put_notify(prod, 1, INT, 1, 0, 1,
                            INT, win, 99)
  MPI_Win_flush(0, win)
}
```

```
while(true) {
  MPI_Notification_init(win, 0, 99, 1, &req)
  MPI_Wait(&req, stat)
  consume()
}
```

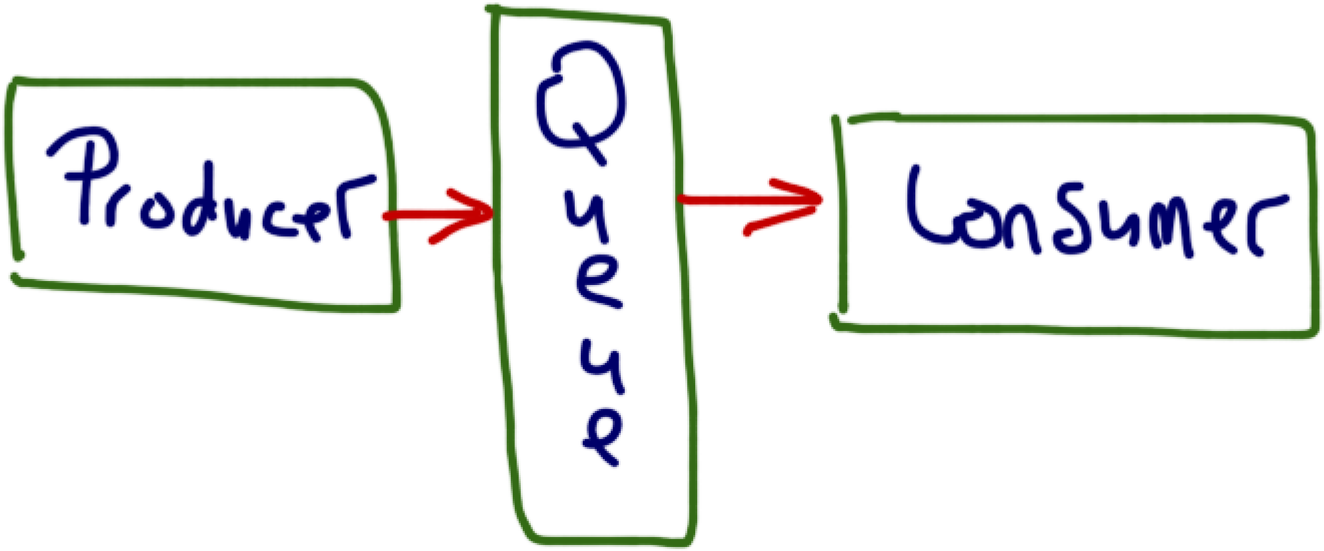
Producer/Consumer Synchronization in RMA

```
while(true) {
  produce()
  MPI_Put_notify(prod, 1, INT, 1, 0, 1,
                      INT, win, 99)
  MPI_Win_flush(0, win)
}
```

```
while(true) {
  MPI_Notification_init(win, 0, 99, 1, &req)
  MPI_Wait(&req, stat)
  consume()
}
```

- **Somewhat odd with the epoch concept**
- **Option 1:**
  - Put_notify() would close and open an access epoch
  - Would be a communication **and** synchronization operation
  - Has to flush all previous messages as well
- **Option 1:**
  - Put_notify() is outside the epoch concept
  - Somewhat ugly in the details and for tools

Alternative? Maybe hard to implement? Cf. #439



```
while(true) {
  produce()
  MPI_Put(prod, 1, INT, 1, 0, 1,
                         INT, win)
  MPI_Win_flush_notify(0, win)
}
```

```
while(true) {
  MPI_Notification_init(win, 0, 99, 1, &req)
  MPI_Wait(&req, stat)
  consume()
}
```

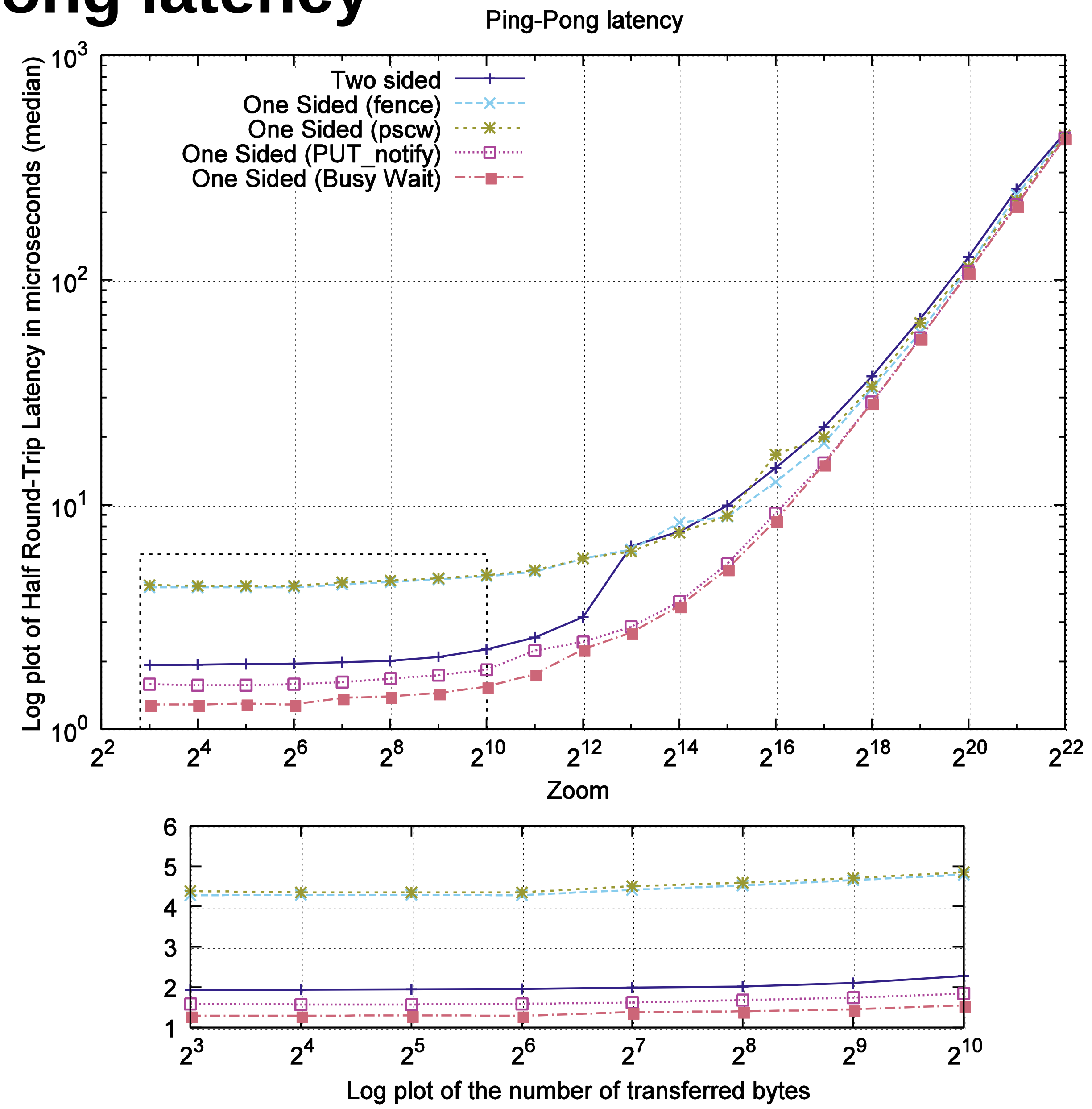
Alternative? Maybe hard to implement? Cf. #439

```
while(true) {
  produce()
  MPI_Put(prod, 1, INT, 1, 0, 1,
                          INT, win)
  MPI_Win_flush_notify(0, win)
}
```

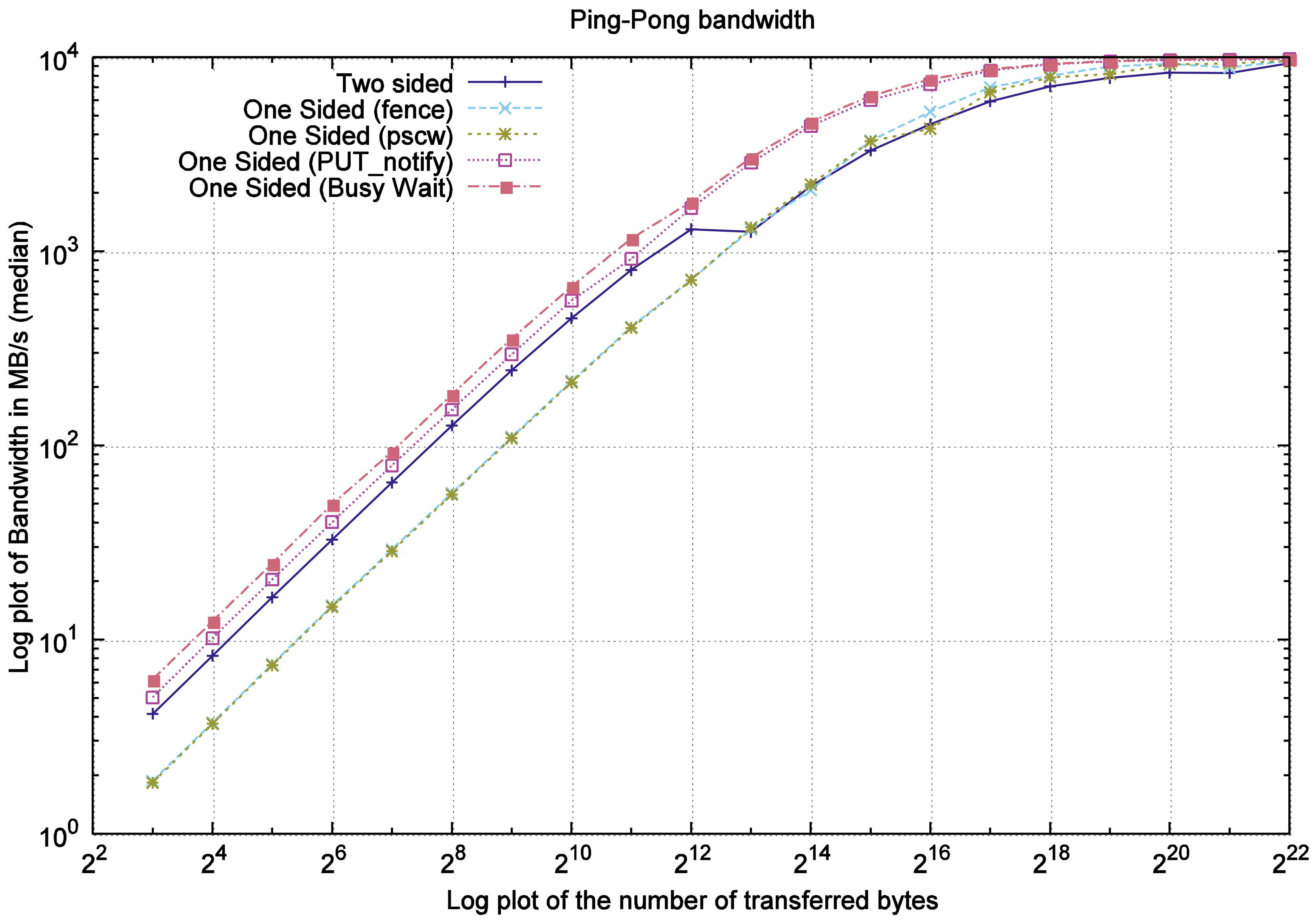
```
while(true) {
  MPI_Notification_init(win, 0, 99, 1, &req)
  MPI_Wait(&req, stat)
  consume()
}
```

- **Slightly better!**
- **Yet, hard to implement on today’s hardware**
  - InfiniBand: rely on in-order and put with immediate
  - Cray DMAPP: impossible, uGNI: not 100% sure (maybe)
  - Shared memory: mfence + store

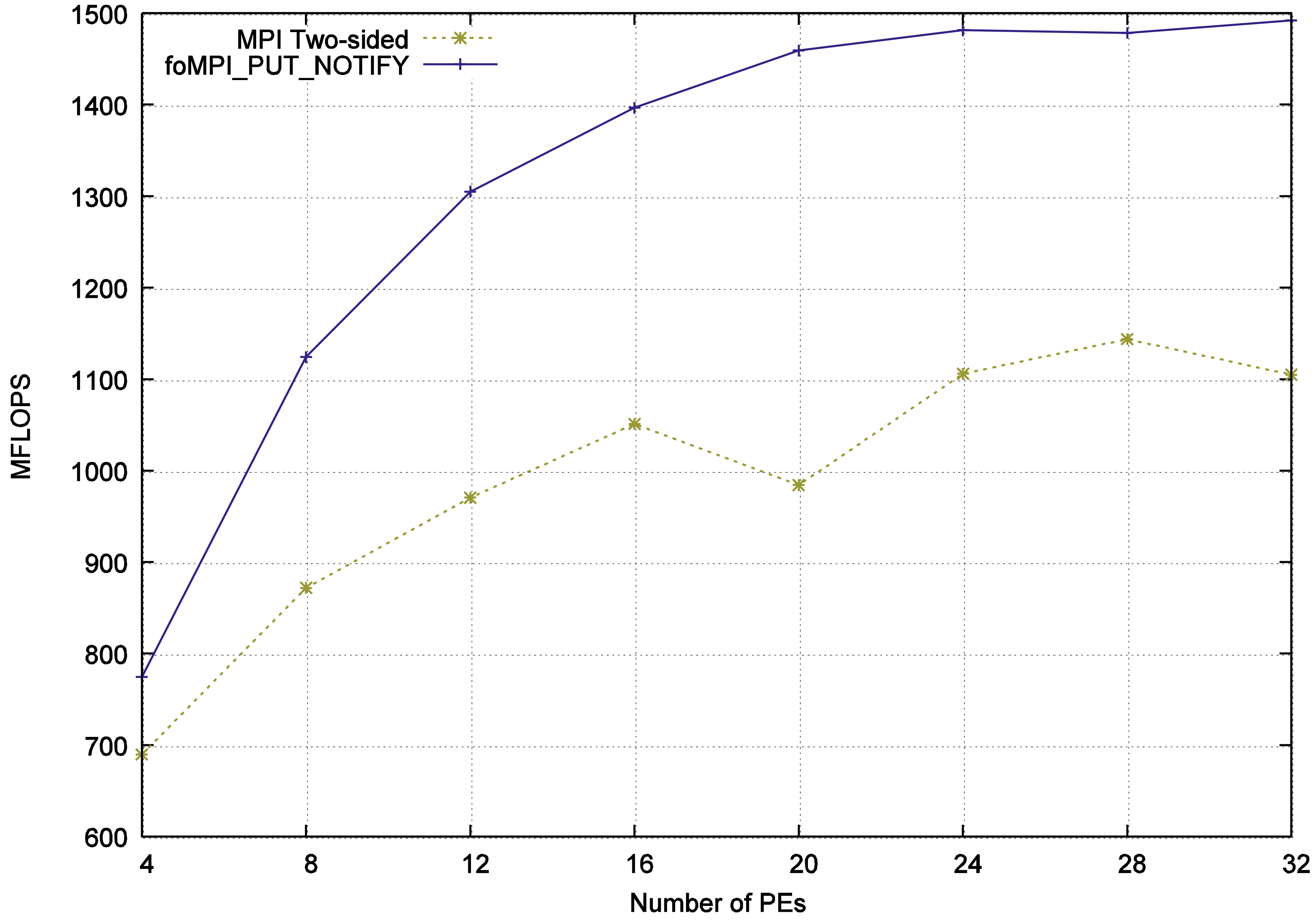
Ping-pong latency



Ping-pong bandwidth



Pipelined Stencil (Intel Microbenchmarks)



Discussion

- **Our foMPI [1] implementation will be released soon**
  - on top of uGNI
  - using Op_notify
  - will be 100ns slower than polling (but safe without seq. consistency)
  - 100 ns still missing (fighting for it)
- **Op_notify or Sync_notify?**
  - Implementation tradeoffs (should they hold us back)?
- **This is all single-producer-single-consumer**
  - What about multiple-producer-multiple-consumer?
  - Needs advanced HW support to be fast (remote memory management)
- **Do we want this at all?**

[1]: R. Gerstenberger, M. Besta, TH: Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided, SC13