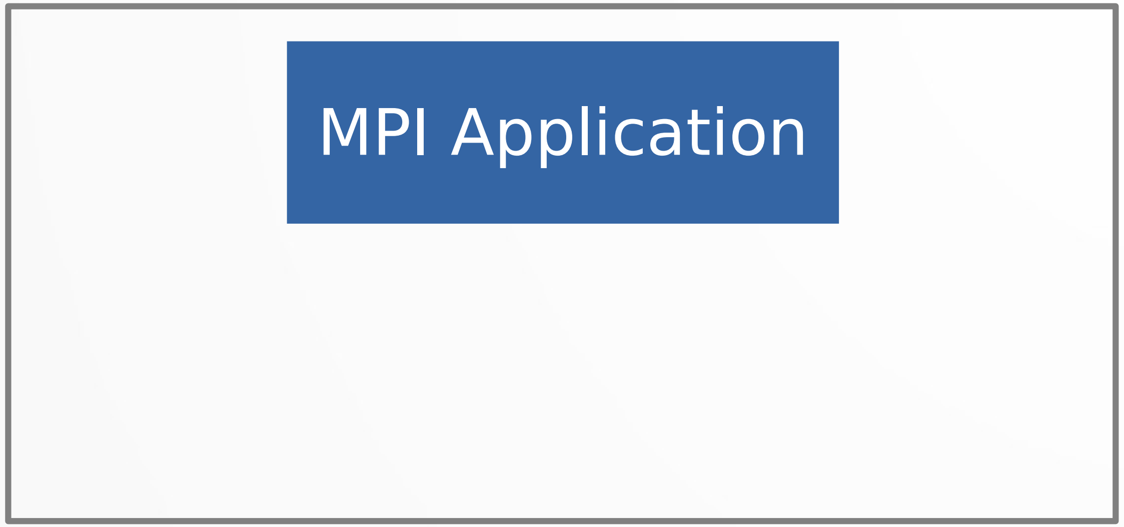
# MPI Design and Implementation

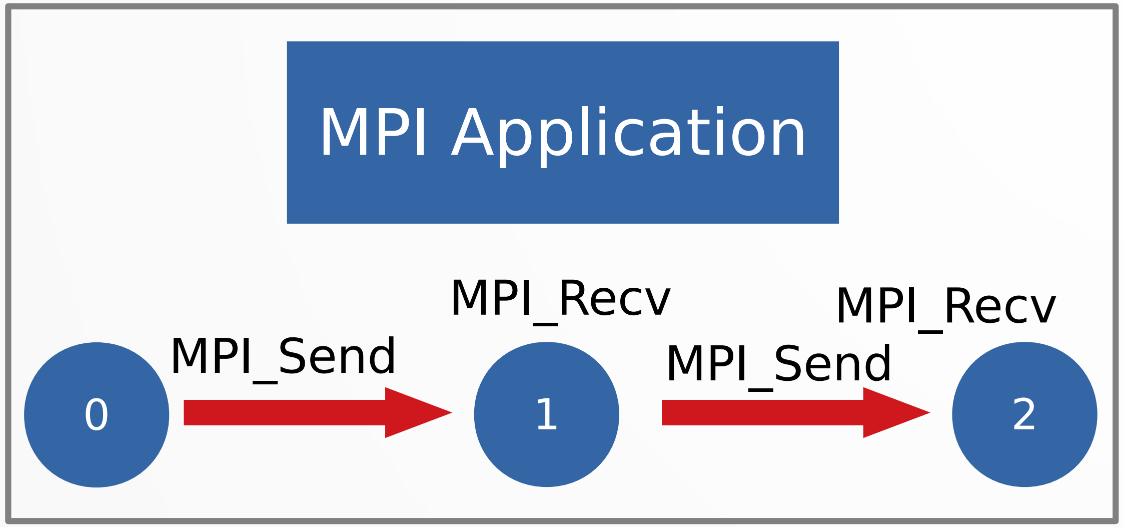
## Programming Models for Emerging Platforms

# MPI *Standard*

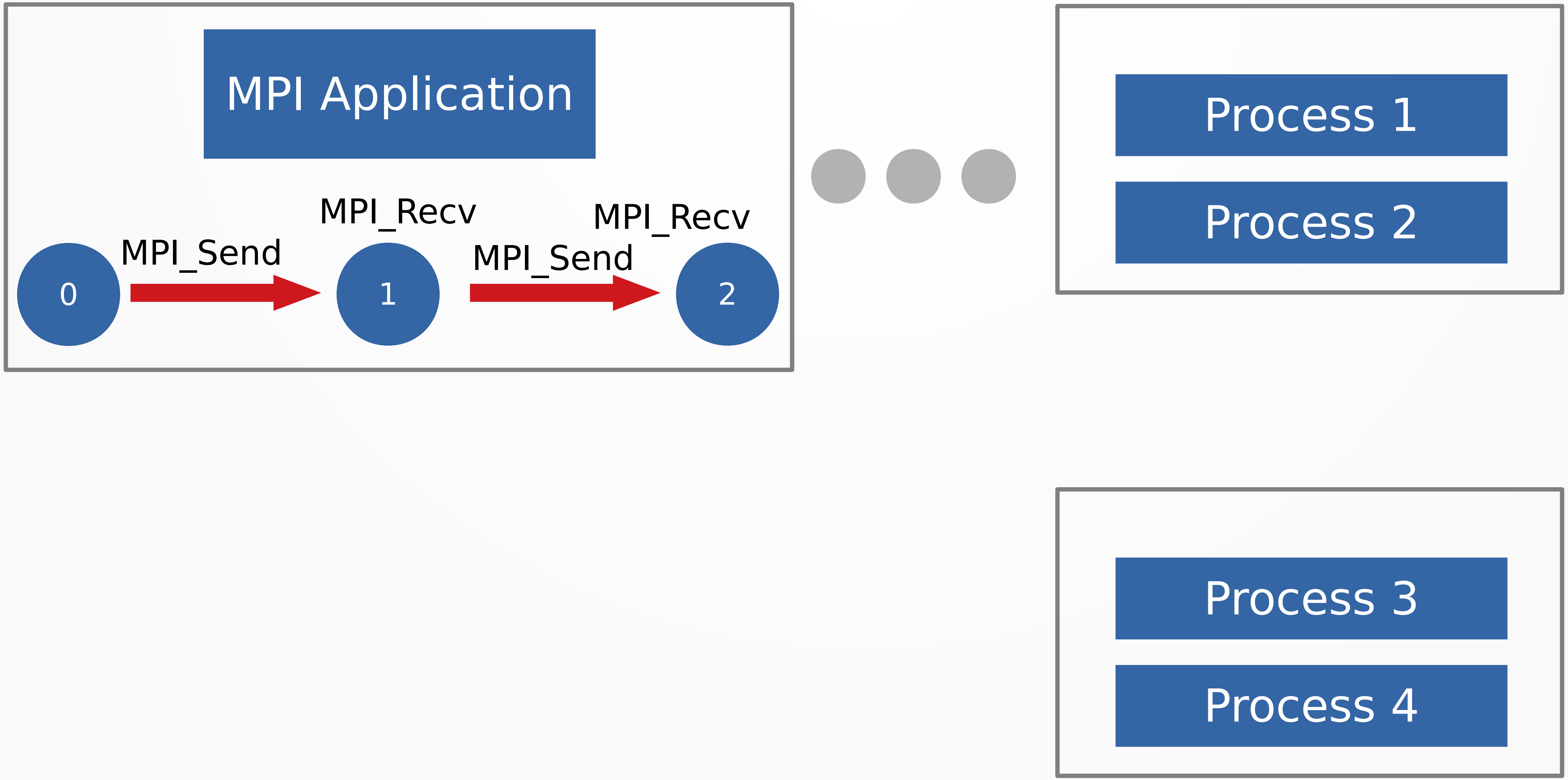


MPI Application

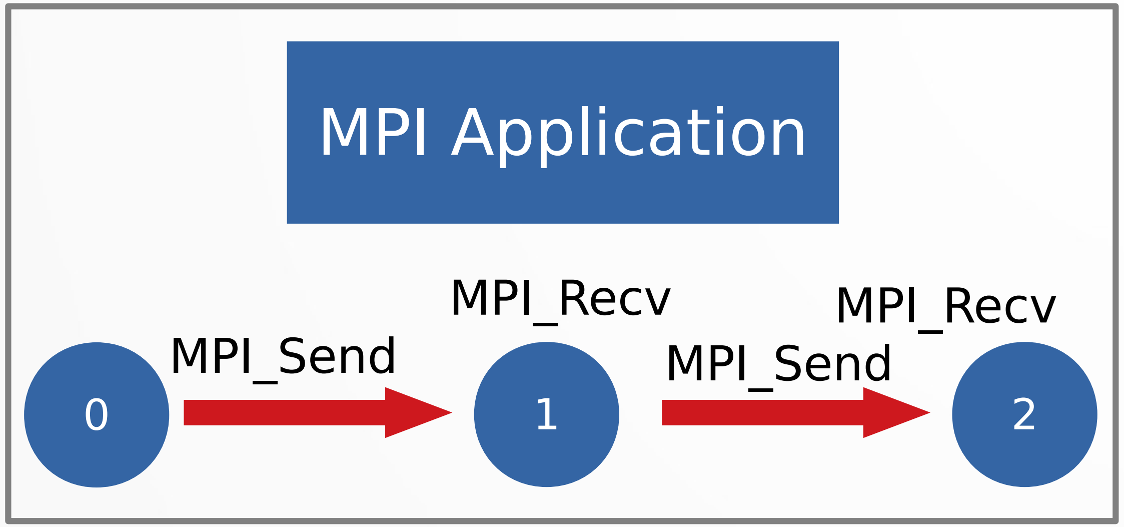
# MPI *Standard*

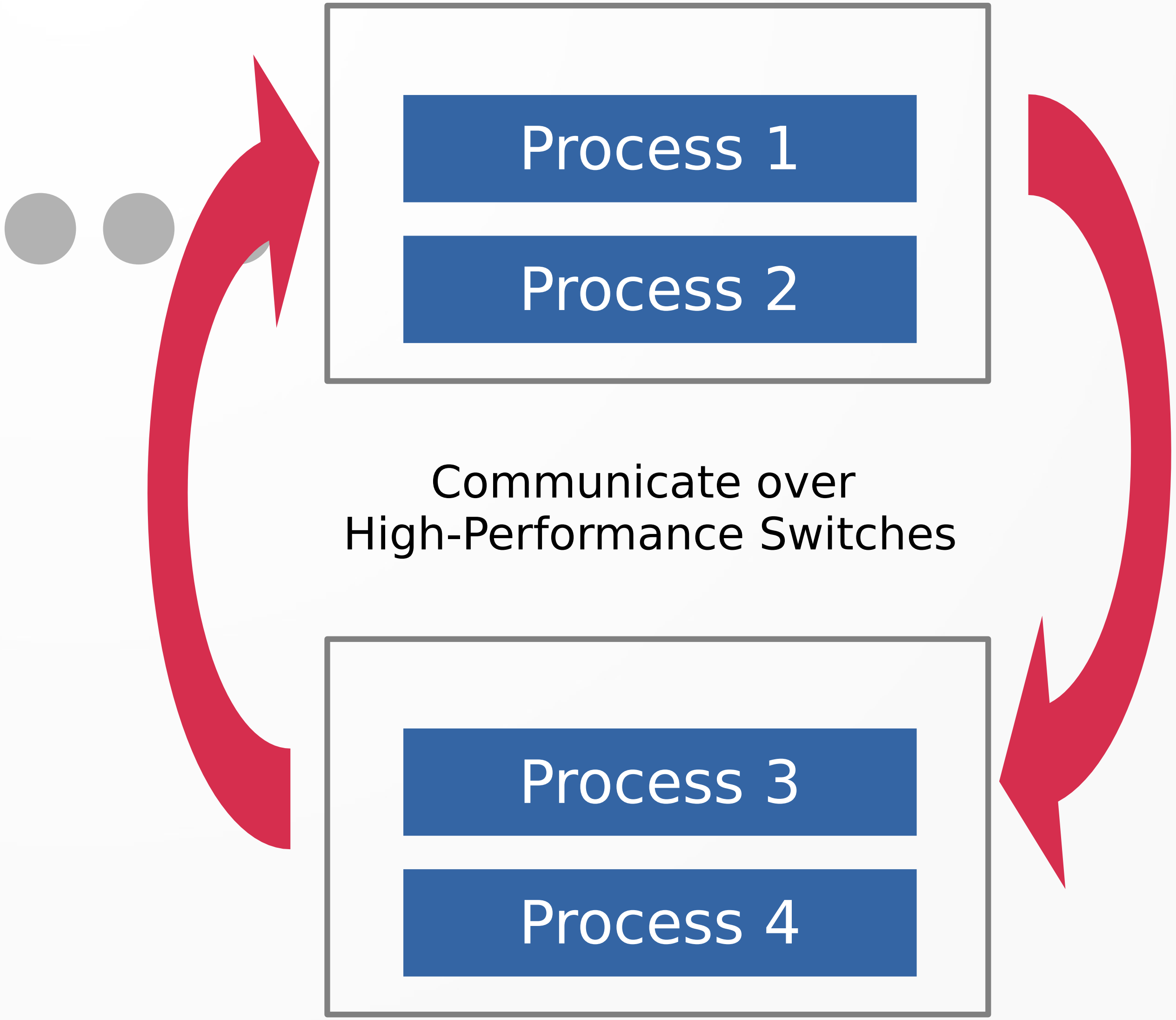


# MPI *Standard*



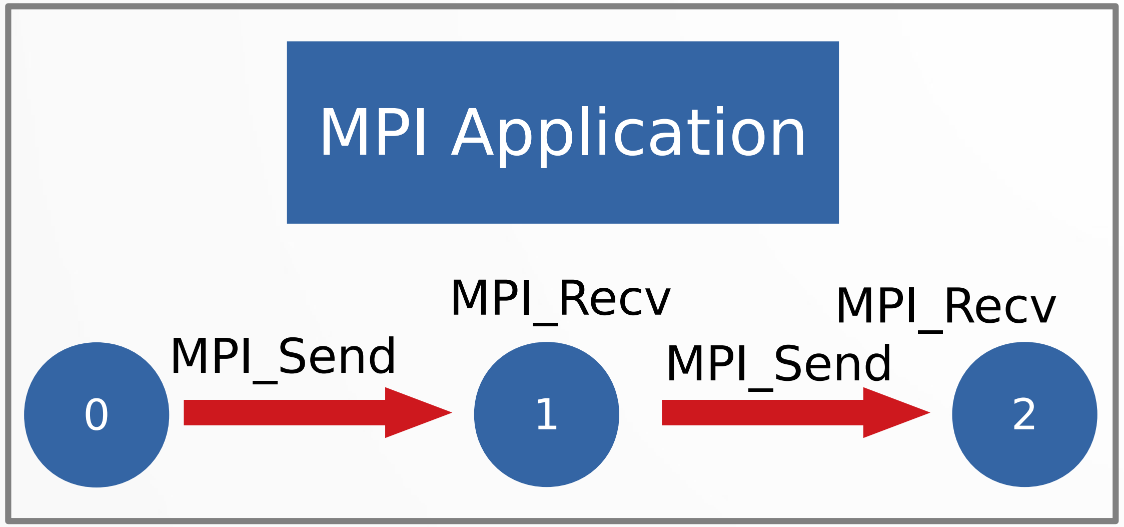
# MPI *Standard*





**Distributed Memory Parallel Supercomputer**

# MPI *Standard*



MPI Application

0

MPI_Send

1

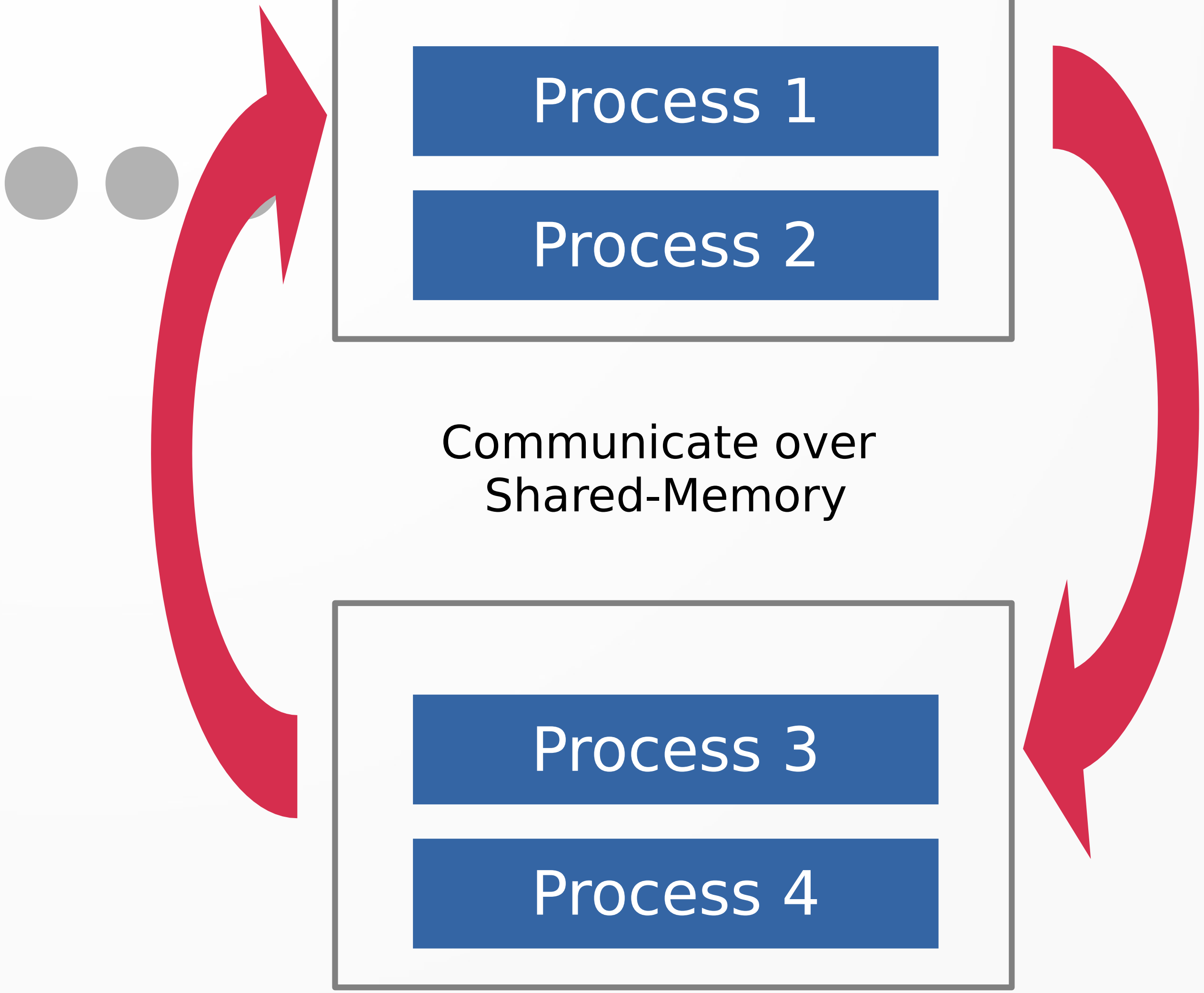
MPI_Recv

MPI_Send

2

MPI_Recv

**Shared-Memory Architecture**



Process 1

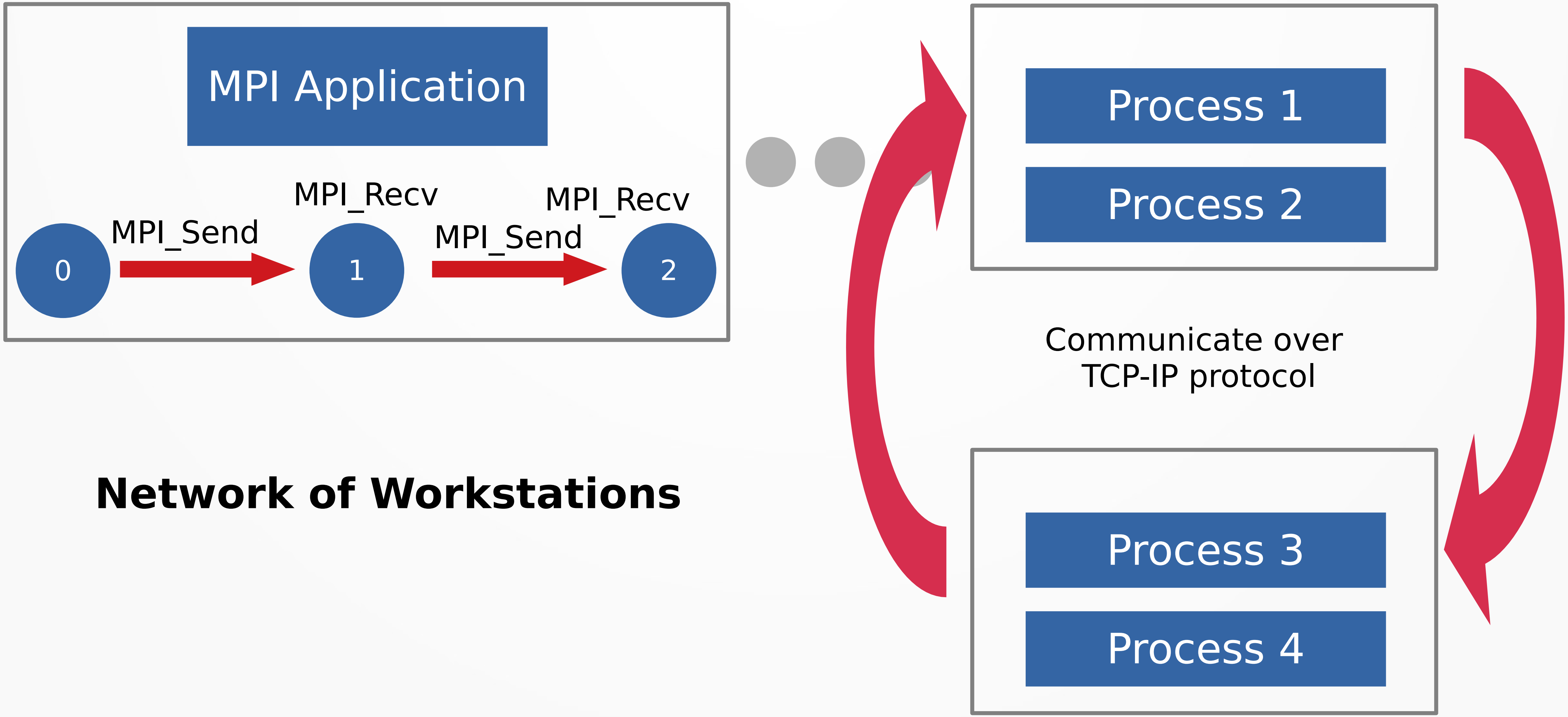
Process 2

Communicate over
Shared-Memory

Process 3

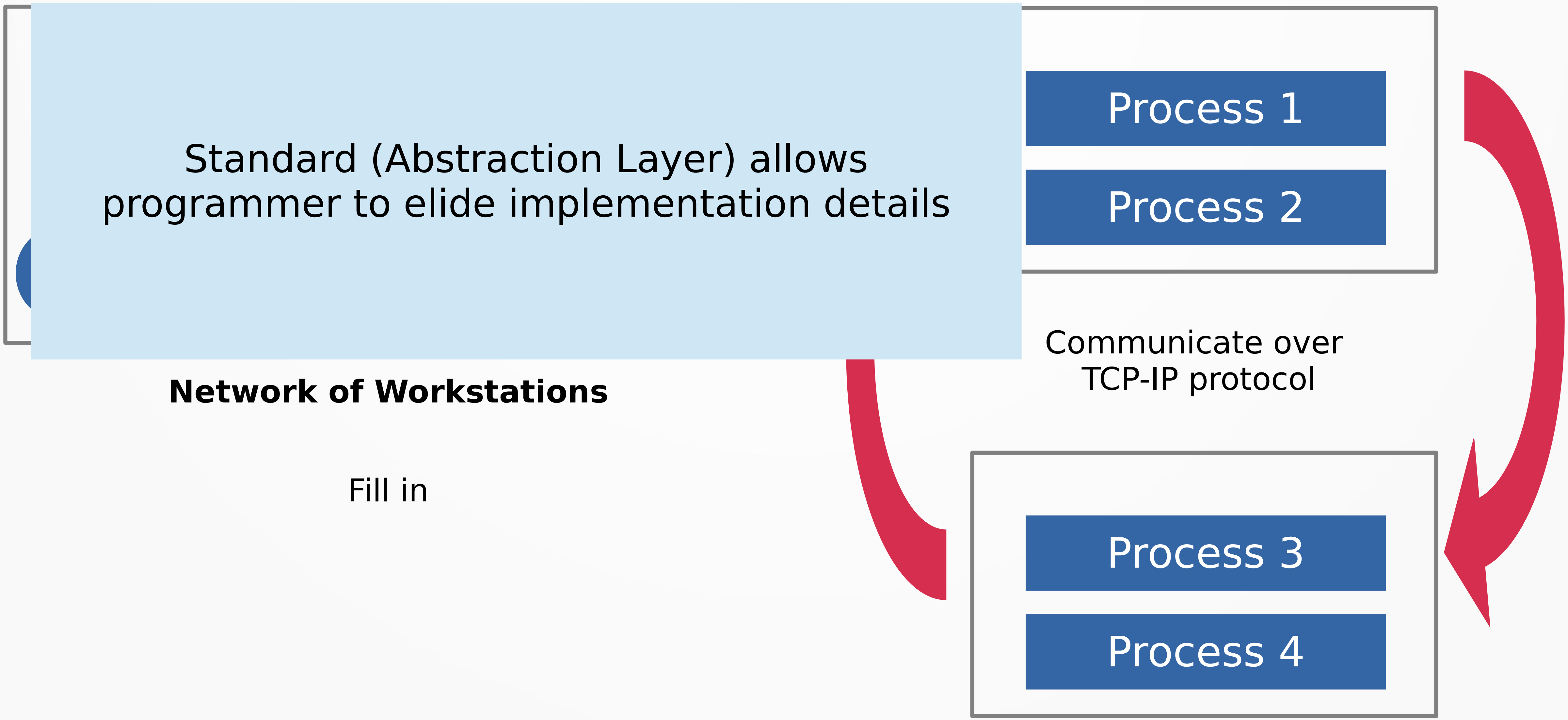
Process 4

# MPI *Standard*



**Network of Workstations**

# MPI *Standard*



Standard (Abstraction Layer) allows programmer to elide implementation details

**Network of Workstations**

Fill in

Process 1

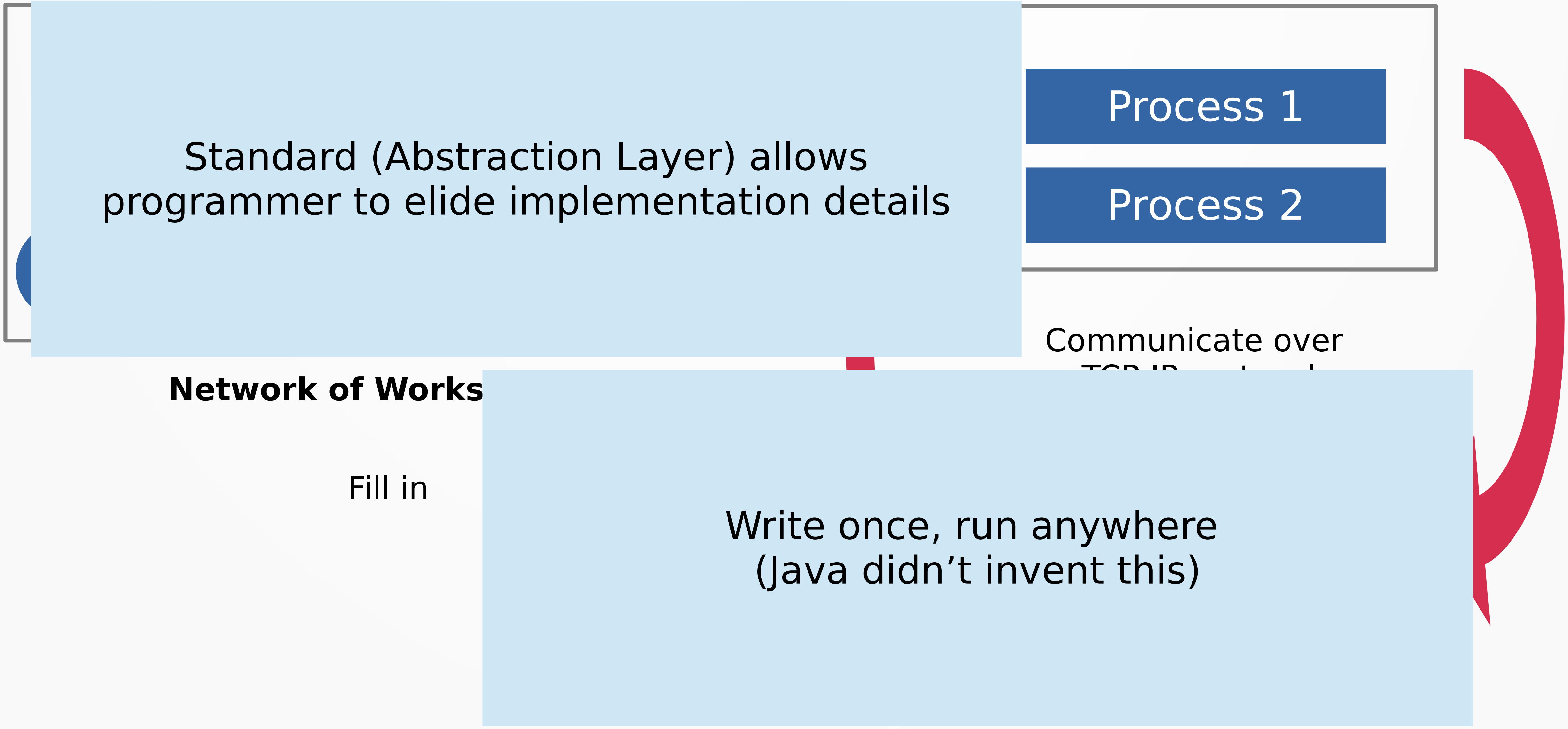
Process 2

Communicate over TCP-IP protocol

Process 3

Process 4

# MPI *Standard*



Standard (Abstraction Layer) allows programmer to elide implementation details

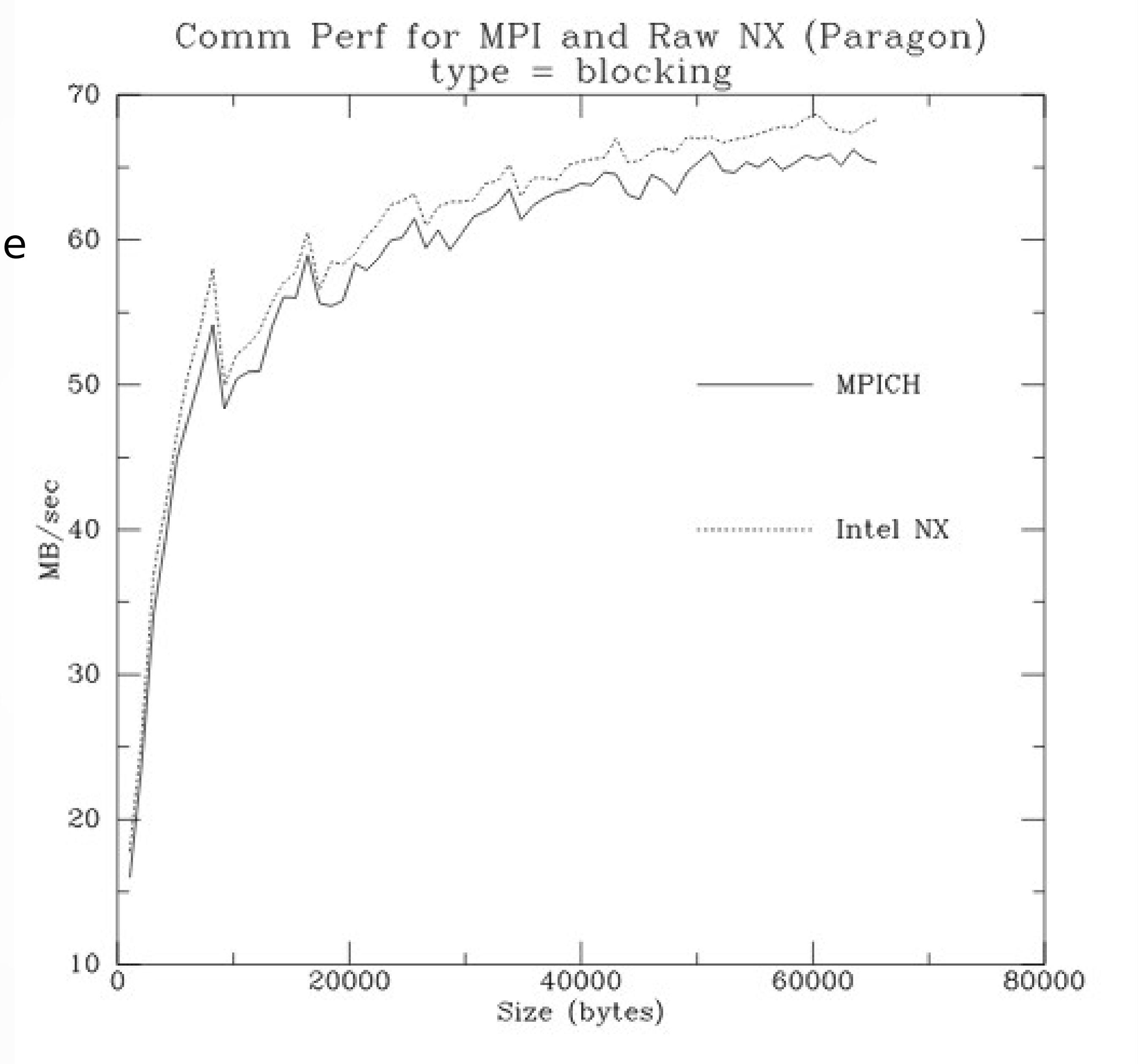
**Network of Works**

Fill in

Write once, run anywhere
(Java didn’t invent this)

Compares MPICH with native machine communication interface (NX)

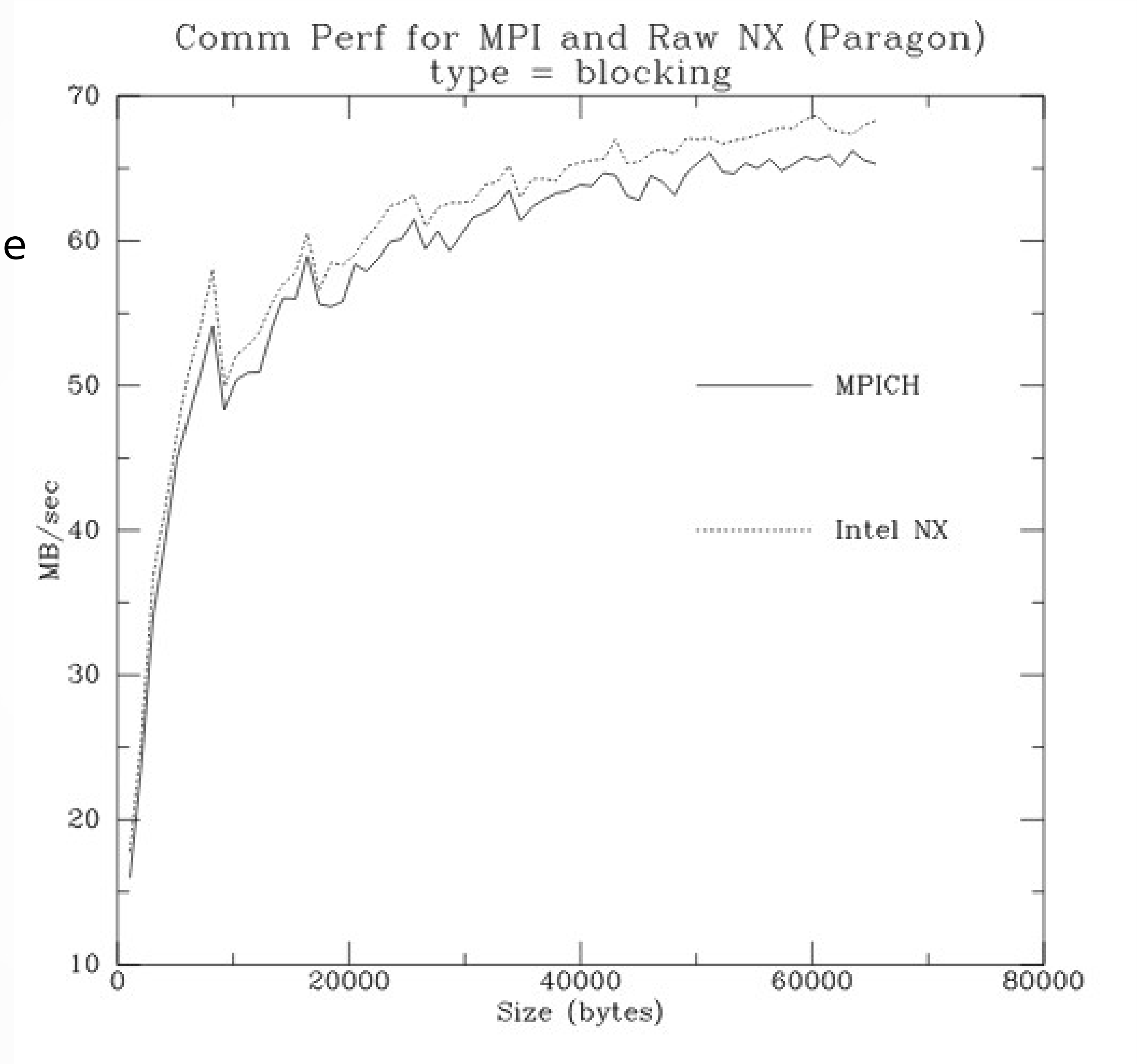
MPICH uses NX on
the Paragron machine

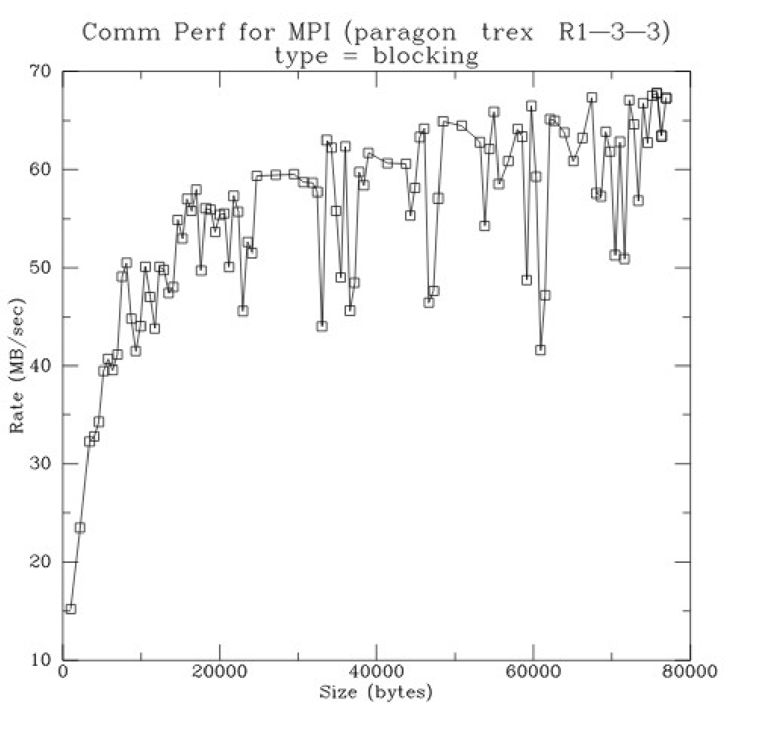


Compares MPICH with native machine communication interface (NX)

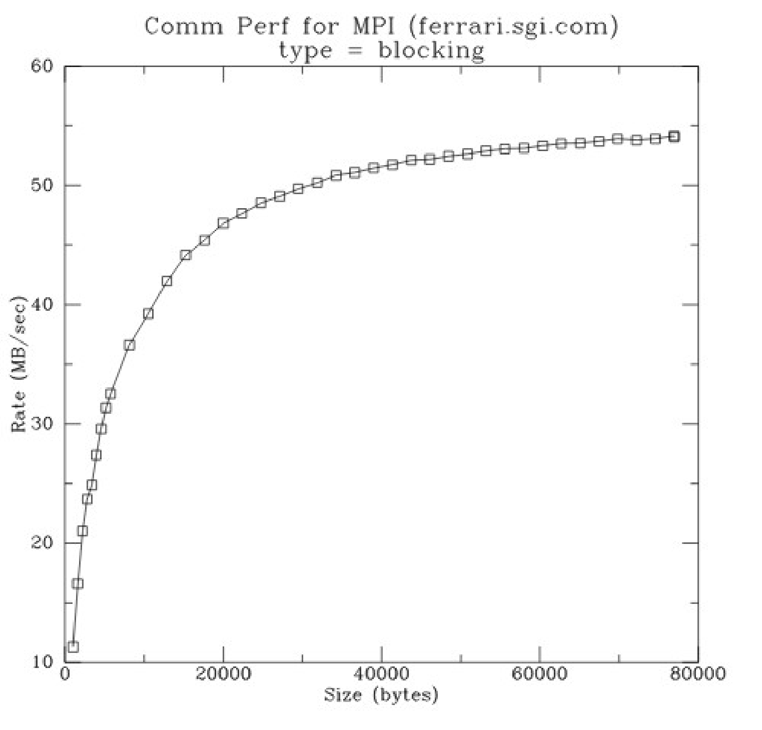
MPICH uses NX on the Paragron machine

There is almost always a *cost* of abstraction

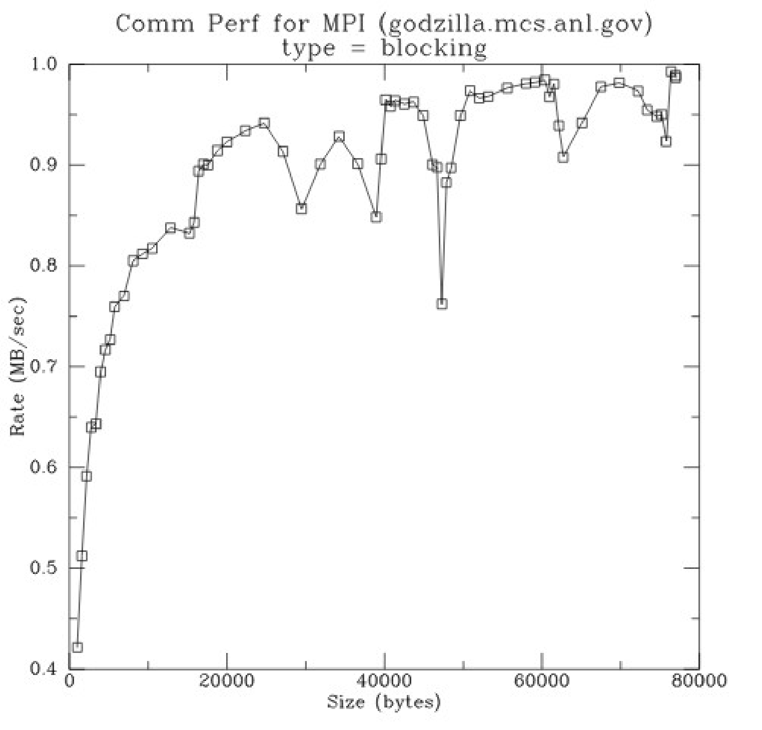




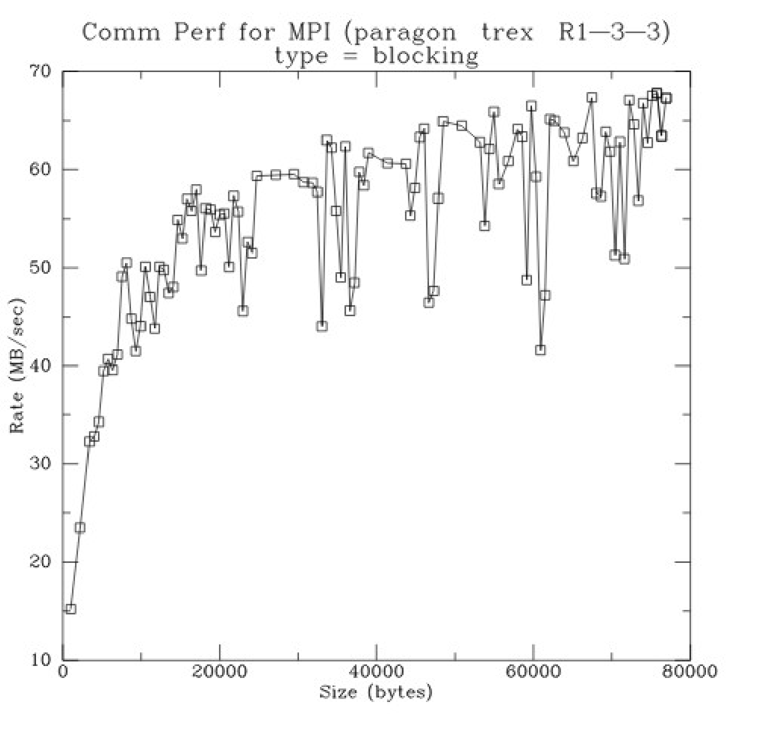
High-Performance Switch



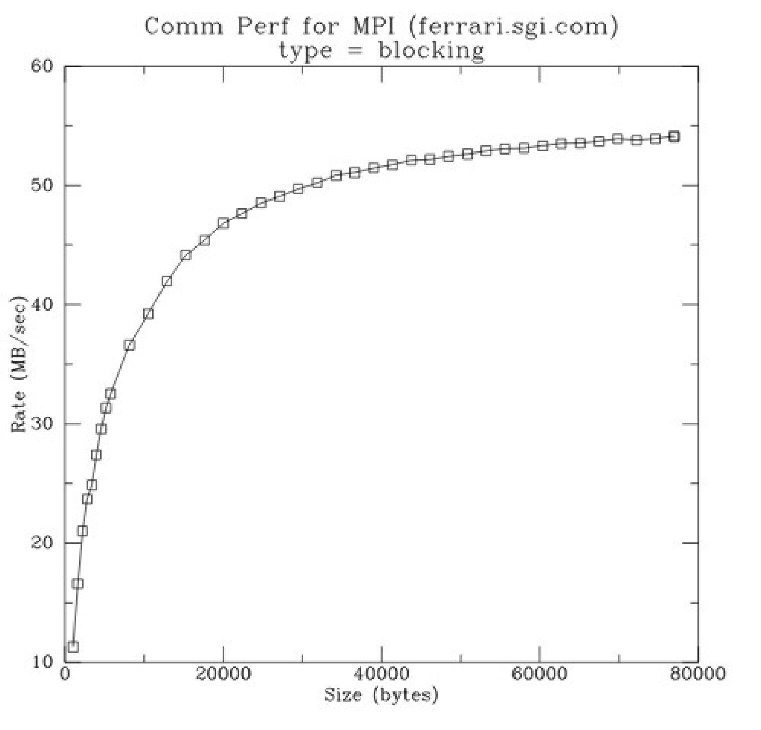
Shared-Memory



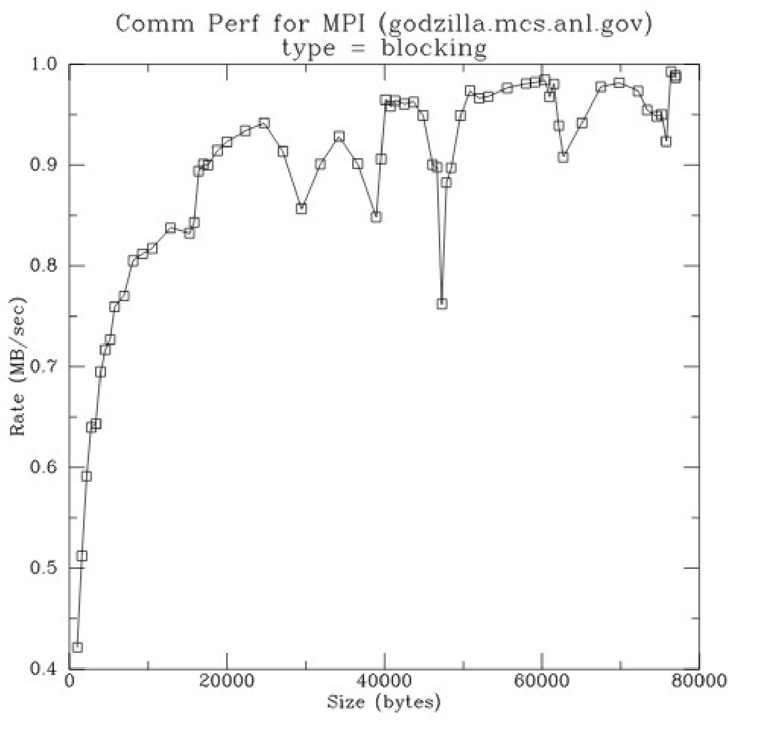
Workstation Network



High-Performance Switch



Shared-Memory



Workstation Network

View this as increasing the range of possible programming environments, not which machine / environment is best

# How to *implement* the standard to be portable and extensible?

# Interface Design

- We often say “provide an interface” as if it is a simple, straightforward part of API design

# Interface Design

- We often say “provide an interface” as if it is a simple, straightforward part of API design
- It’s not! Particularly when lower-layers are involved

# Interface Design

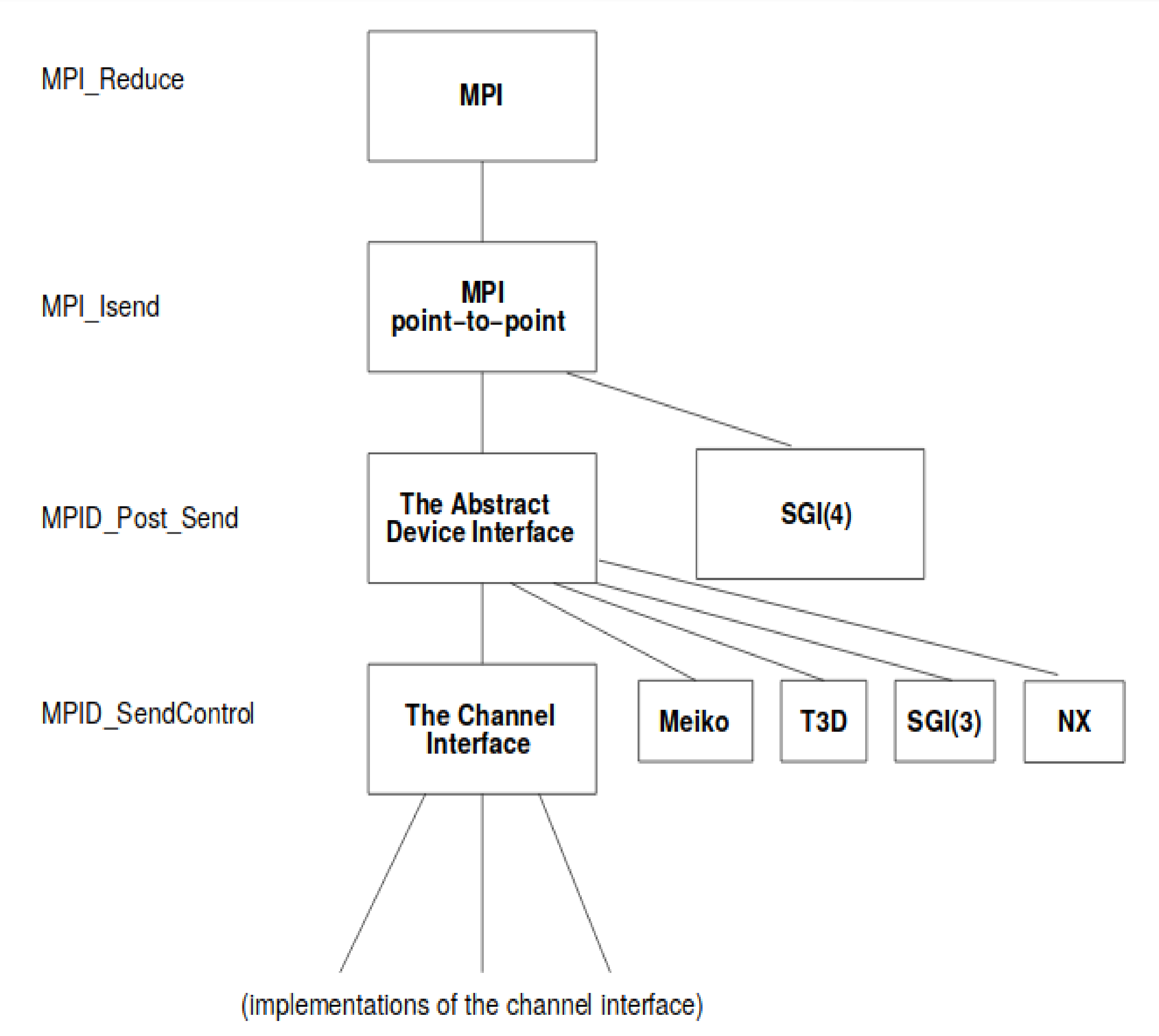
- We often say “provide an interface” as if it is a simple, straightforward part of API design
- It’s not! Particularly when lower-layers are involved
- If you expose too little to the interface, can’t do anything useful. If you expose too much, might as well not have an interface because you must re-implement everything

# Interface Design

- Consider process manipulation (MPI needs processes)
- What even is a “process”? Is a “process” the same on all machines? Specialized hardware? Across network?
- All this needs abstraction

# Abstract Device Interface

- As far as MPI goes, a device needs to provide functions for:
  - 1. Specifying a message to be sent or received
  - 2. Moving data between API and message-passing hardware (User space, device space)
  - 3. Manage pending messages (send and receive)
  - 4. Provide information about execution environment



MPI_Reduce

MPI

MPI_Isend

MPI point-to-point

MPID_Post_Send

The Abstract Device Interface

SGI(4)

MPID_SendControl

The Channel Interface

Meiko

T3D

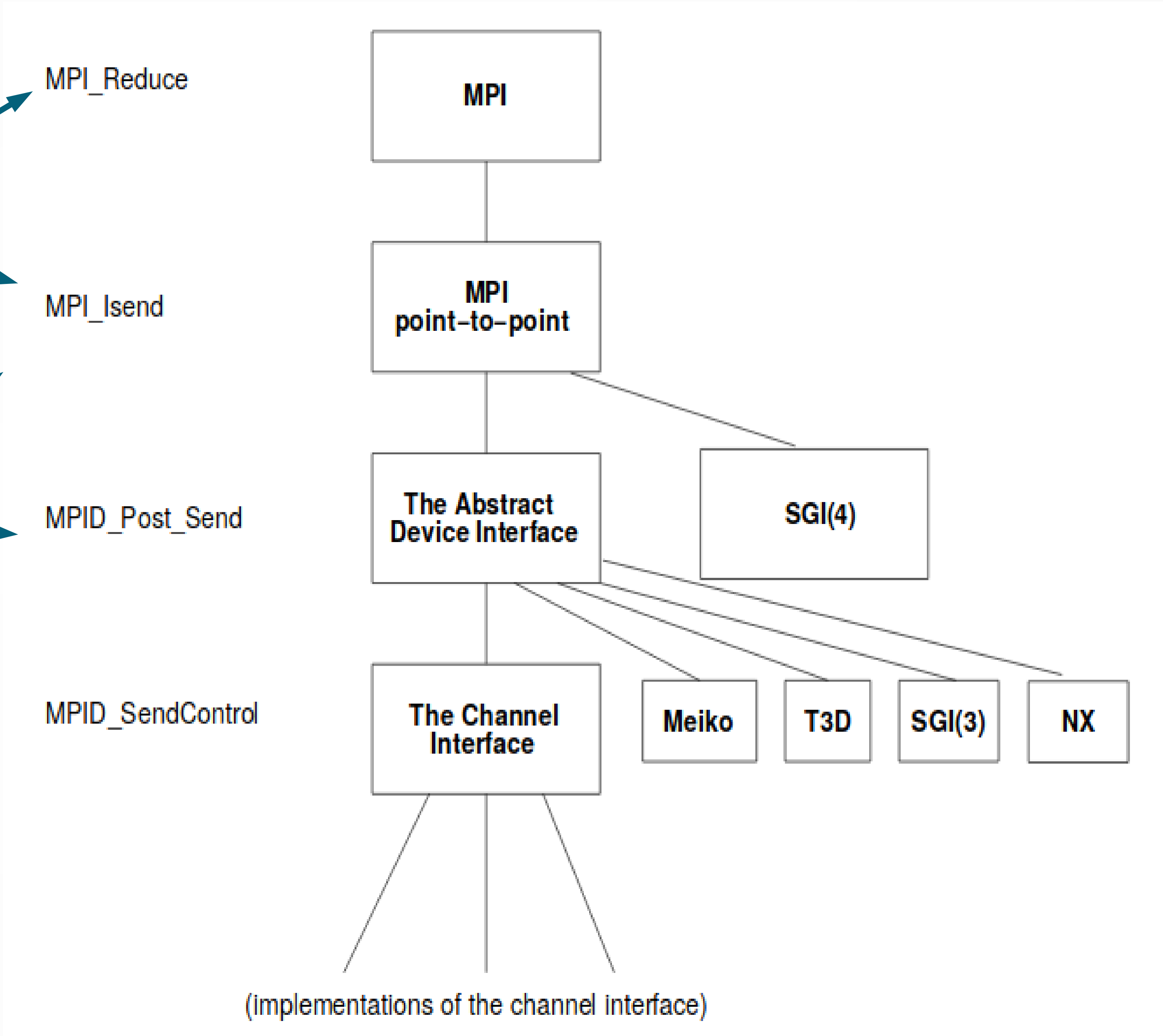
SGI(3)

NX

(implementations of the channel interface)

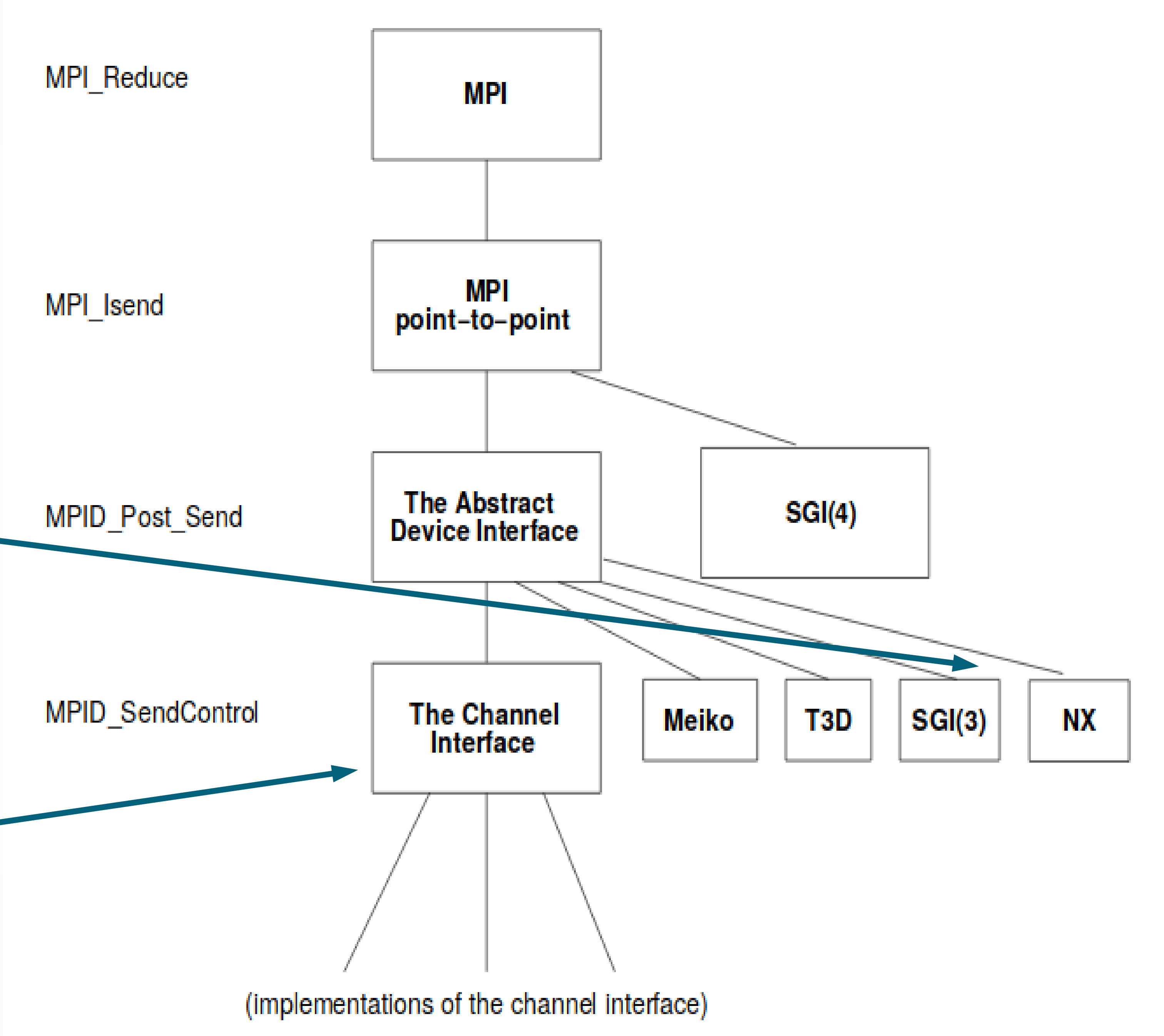
Collective communication implemented over point-to-point

Send moves from API to (ADI) referred to as MPID in code



Specialized hardware can provide implementation

Or use a specialized implementation on general hardware (channel interface)



# Channel Interface

- Low-level functions to physically transfer data from one process space to another process space
- Remember, MPI abstracted lots of details. Even sockets abstract details, but its not that simple...
- Data exchange mechanisms
- Eager
- Rendezvous
- Get

# Channel Interface

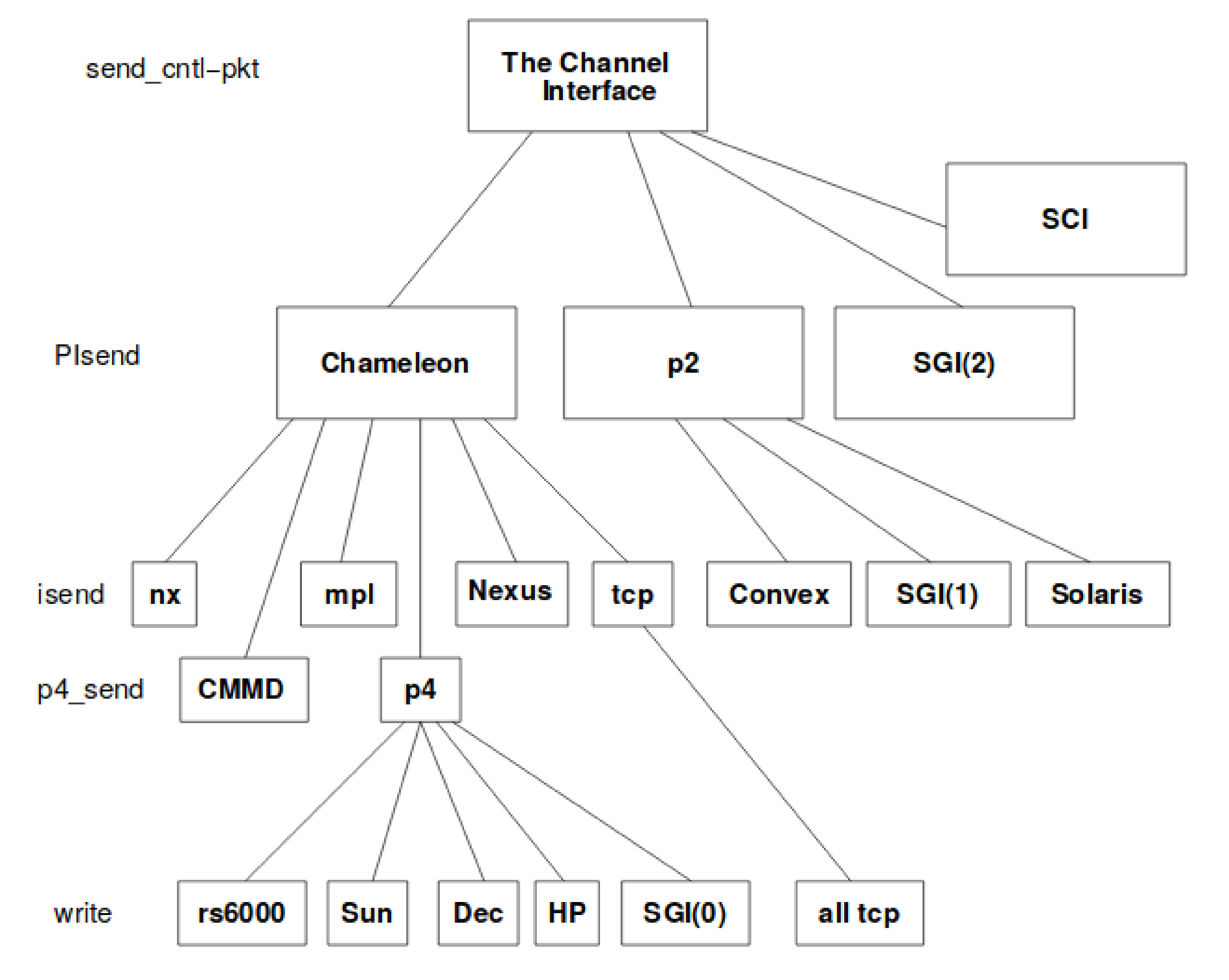
- Data exchange mechanisms
  - Eager
    - Data immediately sent to receiver
    - If receiver is not available to accept data, a buffer is allocated on receiving process to temporarily hold data

# Channel Interface

- Data exchange mechanisms
  - Rendezvous
    - Data sent only when requested from receiver
    - Receiver sends a request for data and a way for the data to be transferred
    - Robust, but less efficient

# Channel Interface

- Data exchange mechanisms
  - Get
    - Data read directly from receiver
    - Similar to rendezvous, but likely uses a form of **memcpy** to directly grab data
    - Requires specialized hardware (shared memory)


send_cntl–pkt
The Channel Interface
SCI
PIsend
Chameleon
p2
SGI(2)
isend
nx
mpl
Nexus
tcp
Convex
SGI(1)
Solaris
p4_send
CMMD
p4
write
rs6000
Sun
Dec
HP
SGI(0)
all tcp

# Exercise

- Revisit (circular-solved.c) using collective communication
  - Try a Scatter / Gather implementation
  - Try a Reduce implementation

# Acknowledgements

- A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard
  - Gropp, Lusk, Doss, Skjellum
  - http://web.cse.ohio-state.edu/~panda.2/788/papers/3a_P567.pdf