INTER-PROCESS COMMUNICATION ON STEROIDS

ATLAS SOFTWARE & COMPUTING WORKSHOP

OCTOBER 2012

Roberto A. Vitillo (LBNL)

CONTEXT

- AthenaMP communications
  - reader process sends events to workers
- Coprocessor communications
  - Athena[MP] jobs interacting with a GPU server process
- Available IPC mechanisms
  - shared memory with explicit synchronization
  - message passing with implicit synchronization

MESSAGE PASSING MODEL

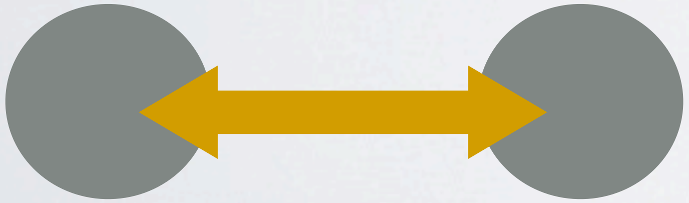
- One of the most successful models for providing concurrency
  - data and synchronization in a single unit
- Actor Model
  - processes have an identity
  - communicate by sending messages to mailing addresses
  - Erlang, Scala
- Process calculi
  - processes are anonymous
  - communicate by sending messages through named channels
  - Go Programming Language

PATTERNS

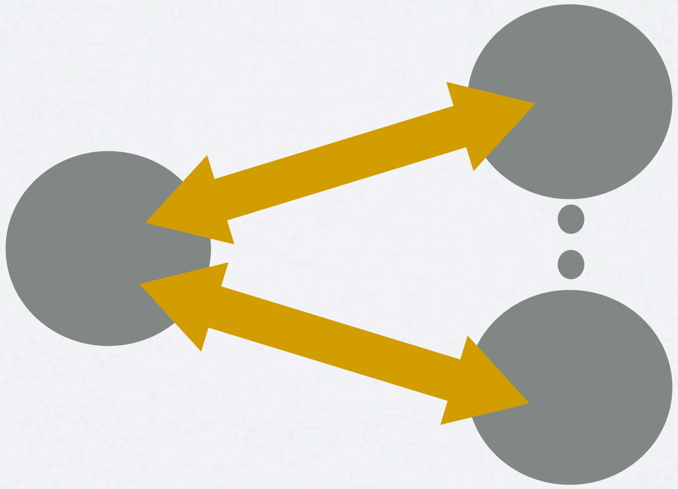
- Producer & Consumer
  - producer pushes messages
  - consumer pulls messages
- Client & Server
  - client makes a request
  - server replies to a request

CHANNELS

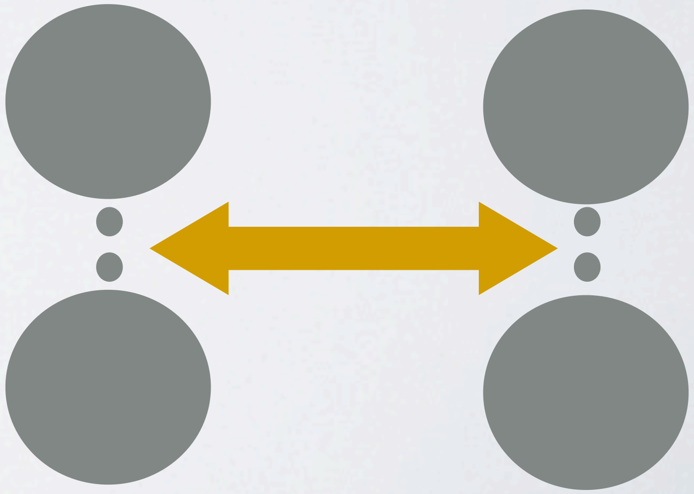
- Properties of a channels:
  - name
  - context (thread, local-process, distributed-process)
  - asynchronous(k)
  - topology



one-to-one



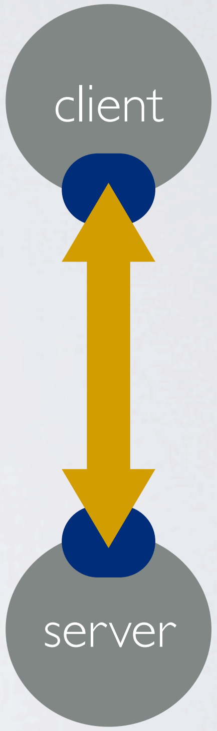
one-to-many



many-to-many

SOCKETS

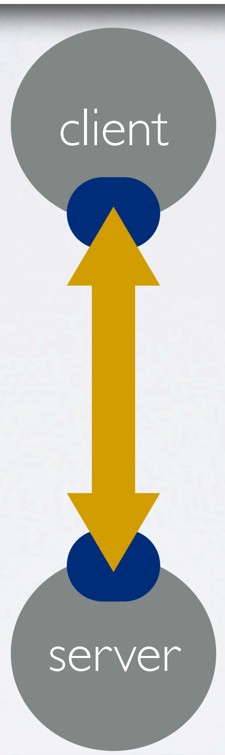
- Each end of a channel is attached to a Socket
- Different patterns have different Sockets,
  - e.g. ProducerSocket, ConsumerSocket
- A Socket allows to:
  - send() buffers of data to its peers (buffer-blocking)
  - receive() buffers of data from its peers (blocking)



SOCKETS

```
Channel channel("service", ONE_TO_ONE)
ISocket *socket = factory->createClientSocket(channel);

socket->send("ping", 5);
socket->receive(&buffer);
```



```
Channel channel("service", ONE_TO_ONE);
ISocket *socket = factory->createServerSocket(channel);

while(true){
    socket->receive(&buffer);
    socket->send("pong");
}
```

DATA TRANSFER TECHNIQUES

- Zero-Copy
  - page table entries are transferred from the source to the destination process
  - requires the buffers to have the same alignment
- Double copy in shared memory
  - double buffering allows the sender and receiver to transfer data in parallel
- Delegate the copy to a NIC
  - avoids cache pollution

DATA TRANSFER TECHNIQUES

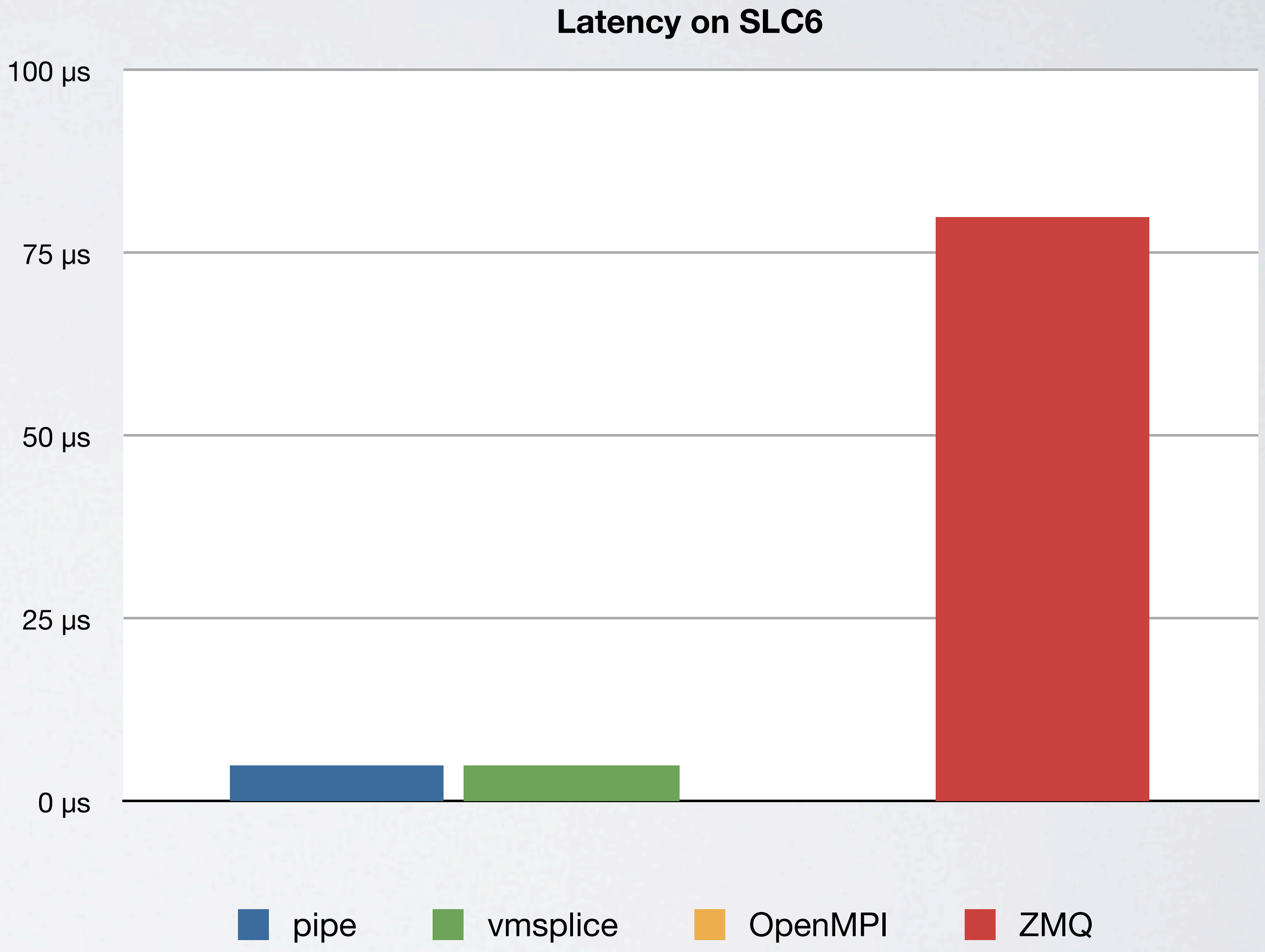
- Single Copy with ptrace()
  - debugging facility, sender processes is stopped
  - receiver process accesses the sender’s process address space and performs a copy
- Single Copy through a Kernel facility
  - vmsplice (Kernel 2.6.17)
  - KNEM (Kernel 2.6.15, requires module)
  - Cross Memory Attach (Kernel 3.2)

IMPLEMENTATION

- The API is currently implemented with ZeroMQ
  - provides a default fall back implementation
  - lock-free queues for threads
  - AF_UNIX sockets for local processes
  - TCP sockets for distributed processes
- The implementation switches according to the channel configuration
  - many-to-many channel triggers the creation of a broker process that handles all communications
  - one-to-one, producer-consumer uses a UNIX pipe with vmsplice()

BENCHMARK: LATENCY

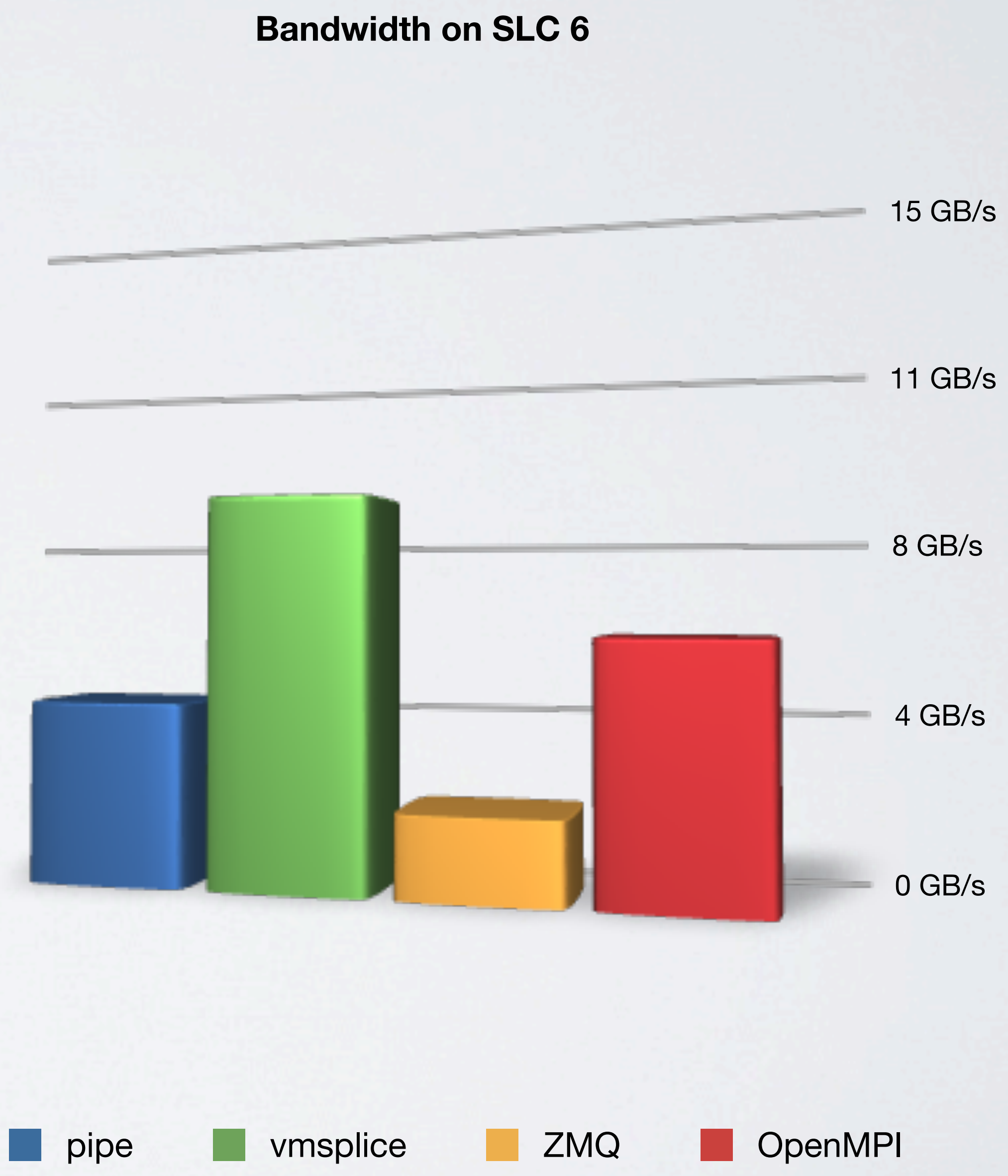
- Test: 1 Byte Ping Pong
- MPI is using shared memory
  - double copy
  - avoids expensive system call which dominates for small messages
- The pipe implementations are dominated by the system call overhead



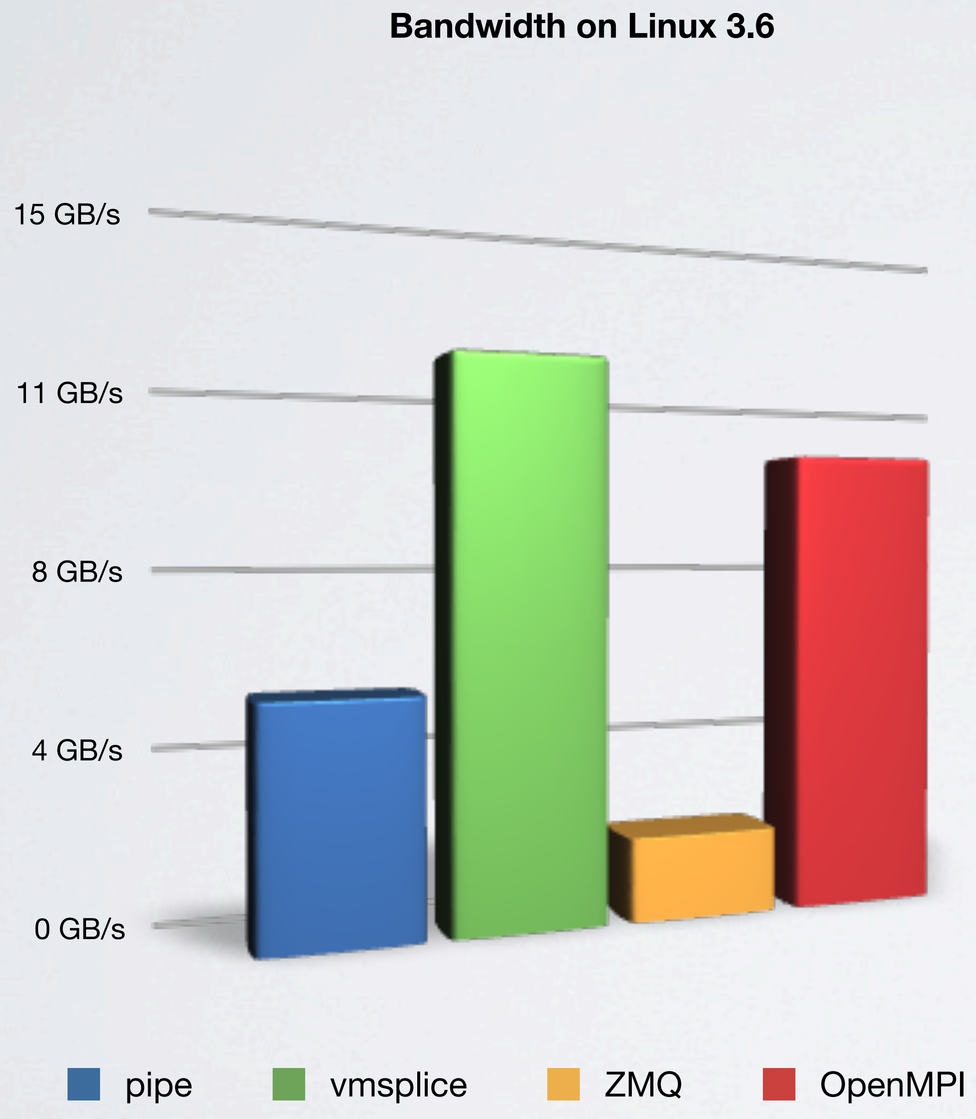
Ivy Bridge, ZMQ 2.2, OpenMPI 1.6.2 without KNEM

BENCHMARK: BANDWIDTH

- Benchmark: 1 Megabyte Ping Pong
- Pipe performs 2 copies (U-K-U)
- Spliced pipe performs a single copy (K-U)
- OpenMPI is using shared memory as buffer and performs 2 copies
  - copies are performed in parallel on both ends
  - consumes more CPU
  - pollutes the caches
- ZMQ uses AF_UNIX sockets, i.e. two copies are performed
  (U-K-U)
  - all sorts of buffers in between
  - optimized for distributed environments



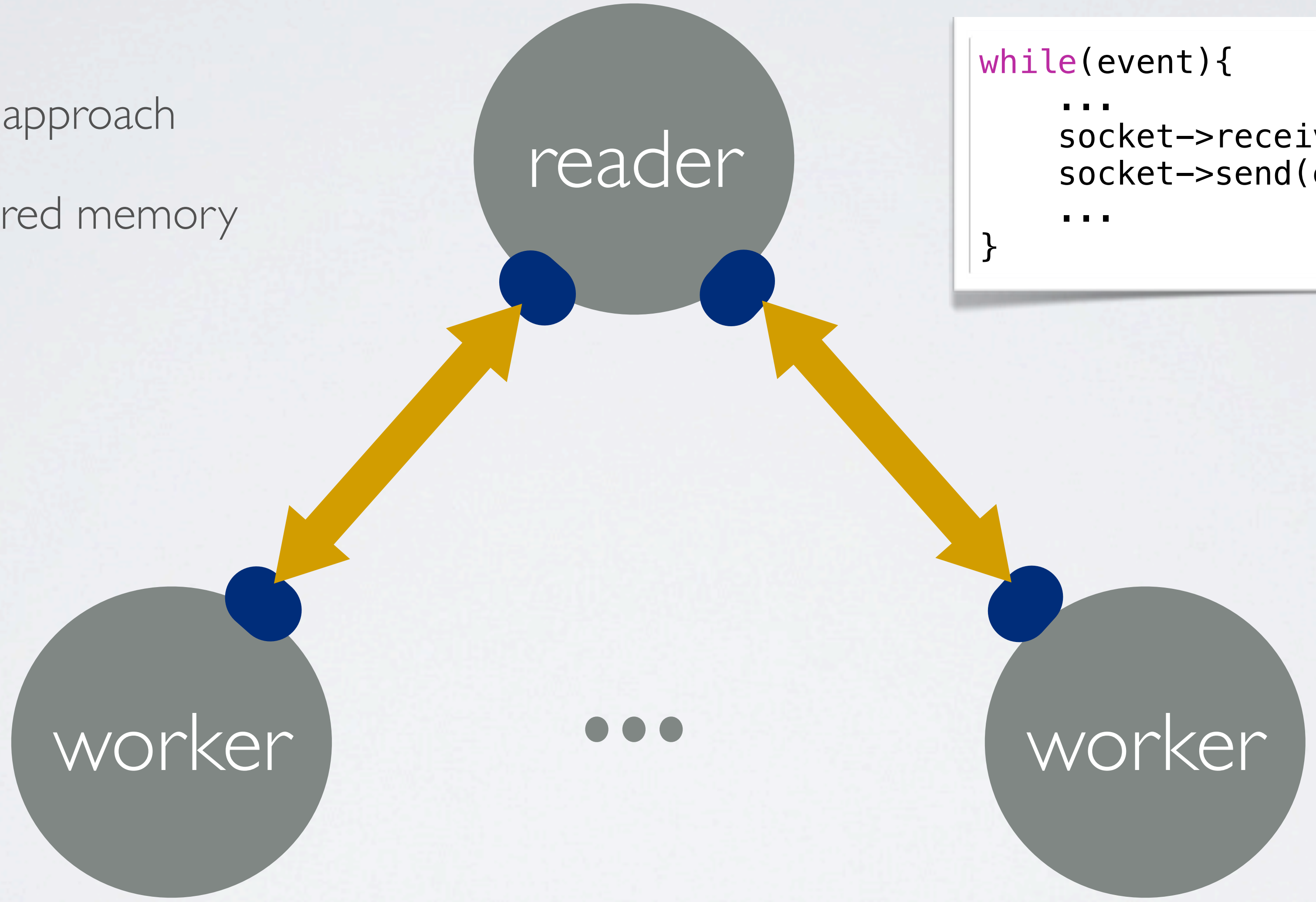
BENCHMARK: BANDWIDTH



- Benchmark: 1 Megabyte Ping Pong
- Pipe performs 2 copies (U-K-U)
- Spliced pipe performs a single copy (K-U)
- OpenMPI is using shared memory as buffer and performs 2 copies
  - copies are performed in parallel on both ends
  - consumes more CPU
  - pollutes the caches
- ZMQ uses AF_UNIX sockets, i.e. two copies are performed
  (U-K-U)
  - all sorts of buffers in between
  - optimized for distributed environments

IN ACTION: ATHENAMP

- See Vakho's talk
- Temporary hybrid approach
  - metadata in shared memory
- Benefits:
  - load balancing
  - synchronization



```
while(event){
    ...
    socket->receive(&ready_notification);
    socket->send(event, event_size);
    ...
}
```

```
while(true){
    ...
    socket->send(ready_notification, size);
    socket->receive(&event);
    ...
}
```

```
while(true){
    ...
    socket->send(ready_notification, size);
    socket->receive(&event);
    ...
}
```

CONCLUSION

- Library provides an uniform message-passing abstraction for inter-process communication
- Data and synchronization in a single unit
- Communication patterns and topologies allow to
  - reduce latency
  - increase bandwidth
  - express parallelism
- More implementations will be considered
  - shared memory segment for small messages (<10K, low latency)
  - MPI? Doesn't behave well with forking processes...

QUESTIONS?