Agenda for today:

- Using Multi-threaded programming
- Message passing:
  - Parallel primitives
  - Reduce
- Map/Reduce

Upcoming Deadlines

Python Programming 2
Not available until Apr 1 | Due Apr 15 at 10am | -/5 pts

Preparation for Lecture 4/15
Not available until Apr 1 | Due Apr 15 at 10am

Assignment 2
Available until Apr 20 | Due Apr 17 at 10am | -/14 pts

DIT 873 & DAT346
Techniques for Large-Scale Data

Lecture 4

Recap

Monte Carlo: Computing Pi

```
p = multiprocessing.Pool(args.workers)
s = p.map(sample_pi, [n]*args.workers)
```

- List of inputs
- Apply the same function to each list element
- Replace list elements by functions output

Here:
only communication providing argument & returning result!

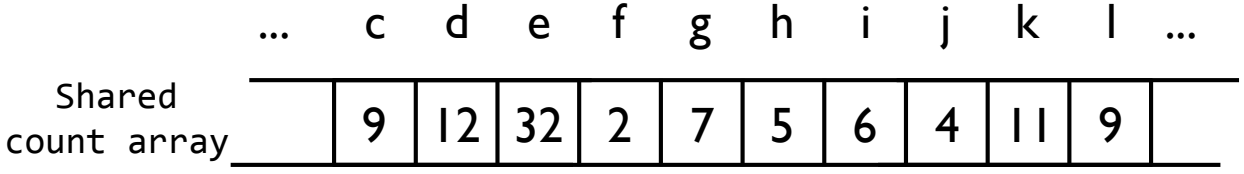
Threads writing to memory

Multi-threaded character counting:

Thread #1

the quick brown fox

```
count['i'] += 1
```




```
count['i'] += 1
```

Thread #2 multi-threading

Possible Execution

1. Thread 1 reads current value 6 from count array into register
2. Thread 1 increments register
3. Thread 2 reads current value 6 from count array
4. Thread 1 writes incremented value 7
5. Thread 2 increments register
6. Thread 2 writes incremented value 7

⇒ count['i'] incorrect

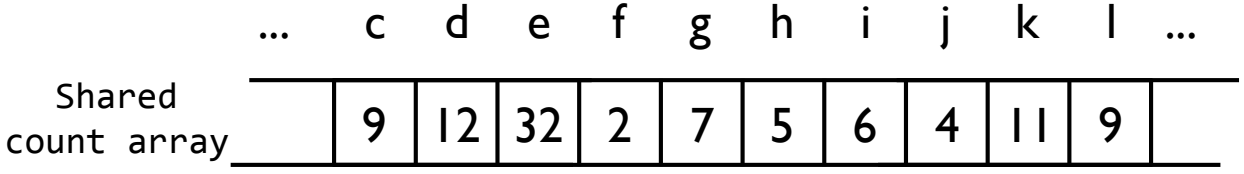
See Sec. 3.8.3.1 Rauber & Rünger (2013)

Threads writing to memory

Multi-threaded character counting:

Thread #1

```
acquire lock count['i']
count['i'] += 1
release lock count['i']
```



Thread #2

```
acquire lock count['i']
count['i'] += 1
release lock count['i']
```

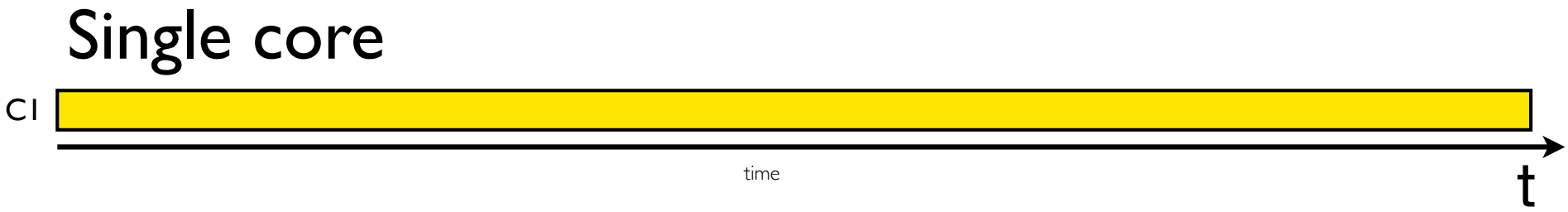
Possible Execution

1. Thread 1 locks count[i]
2. Thread 2 tries to lock count[i] and waits
3. Thread 1 reads current value 6 from count array into register
4. Thread 1 increments register
5. Thread 1 writes incremented value 7
6. Thread 1 releases lock
7. Thread 2 locks count[i]
8. Thread 2 reads current value 7 from count array
9. Thread 2 increments register
10. Thread 2 writes incremented value 8
11. Thread 2 releases lock

⇒ count['i'] correct

See Sec. 3.8.3.1 Rauber & Rünger (2013)

Load Balancing: k-means



Serial and parallel sections on 4 cores

See Sec. 1.3 Rauber & Rünger (2013)

Under the hood: Python arrays

- Numpy, Python array module expose the underlying C-arrays
- This allows sharing them in memory
- See `multiprocessing.RawArray`

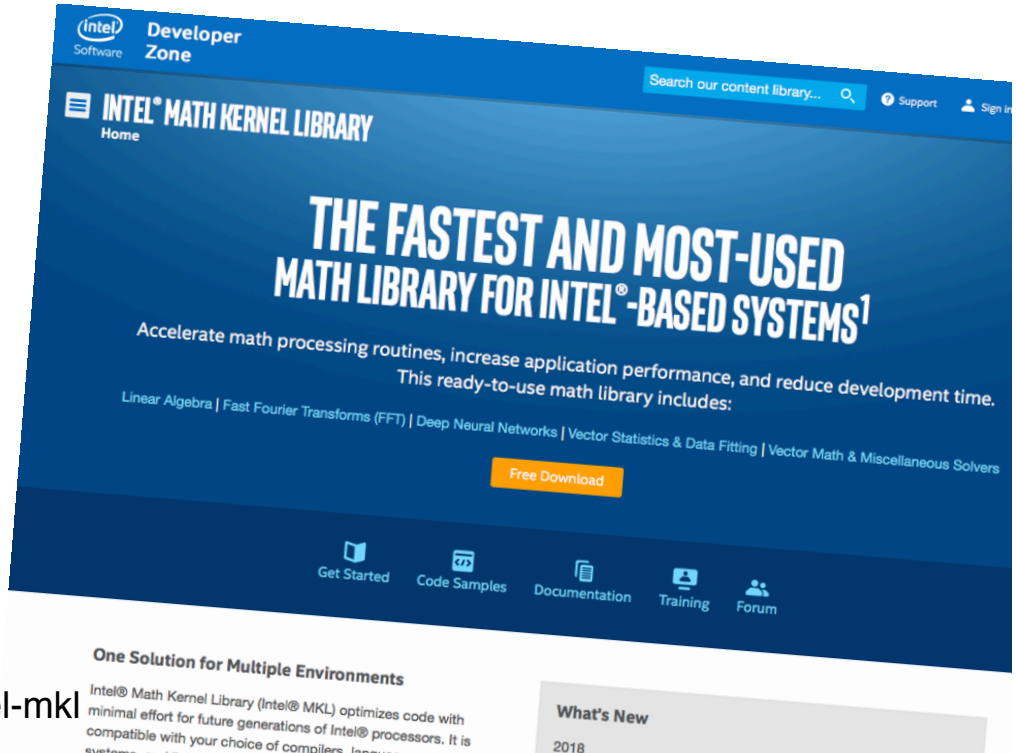
https://docs.python.org/2/library/array.html

Continuing …

Threads in Python for the lazy

- Intel Math Kernel Library (MKL) is multi-threaded
- used by conda install of numpy
- mostly for linalg, some basic array operations including sum

```
import mkl
import numpy as np

mkl.set_num_threads(4)
x = np.ones((N,N))
y = np.ones((N,N))
np.dot(x,y)
```



https://software.intel.com/en-us/mkl

https://software.intel.com/en-us/articles/numpyscipy-with-intel-mkl

Pitfalls

- Multiprocessing + threaded MKL could lead to reduced performance

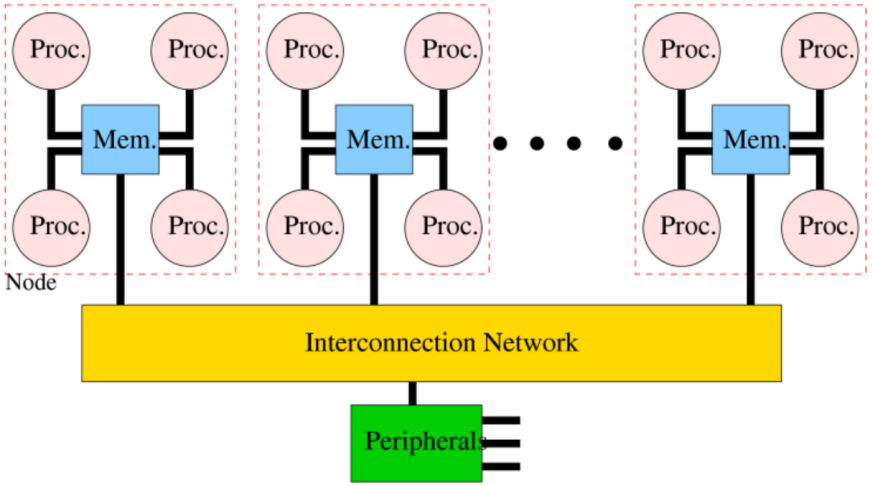
  #MP processes + #ML threads > #cores

Parallel programming models

Message Passing

https://www.scipy.org/topical-software.html#parallel-and-distributed-programming

Parallel programming models: message passing



- Communication incl. exchange of data via interconnect. Infiniband 5 µs (5000 ns + 2 x RAM access)

- Lower throughput for sharing large amount of data vs. RAM
- Explicit parallelization needed
- MPI is library standard

See Sec. 2.3 Nielsen (2016)

Passing a message over Infiniband takes about 5000 ns. This is how many times slower than a memory access?

A	B	C	D	E
1–10	11-50	51-250	251-1000	>1000

Latency

```
          0.5 ns - CPU L1 dCACHE reference
          1   ns - speed-of-light (a photon) travel a 1 ft (30.5cm) distance
          5   ns - CPU L1 iCACHE Branch mispredict
          7   ns - CPU L2  CACHE reference
         71   ns - CPU cross-QPI/NUMA best  case on XEON E5-46*
        100   ns - MUTEX lock/unlock
        100   ns - own DDR MEMORY reference
        135   ns - CPU cross-QPI/NUMA best  case on XEON E7-*
        202   ns - CPU cross-QPI/NUMA worst case on XEON E7-*
        325   ns - CPU cross-QPI/NUMA worst case on XEON E5-46*
     10,000   ns - Compress 1K bytes with Zippy PROCESS
     20,000   ns - Send 2K bytes over 1 Gbps NETWORK
    250,000   ns - Read 1 MB sequentially from MEMORY
    500,000   ns - Round trip within a same DataCenter
 10,000,000   ns - DISK seek
 10,000,000   ns - Read 1 MB sequentially from NETWORK
 30,000,000   ns - Read 1 MB sequentially from DISK
150,000,000   ns - Send a NETWORK packet CA -> Netherlands
|    |    |    |
|    |    | ns|
|    | us|
| ms|
```

Stackoverflow & http://norvig.com/21-days.html#answers

MPI: Message Passing Interface

- See https://www.open-mpi.org/
- Runs on large portion of TOP500 super computers
- Mature code base: OpenMPI v1.0 released Nov 2005. MPI 1.0 standard developed 1992-1994
- Python bindings: http://mpi4py.scipy.org/

Parallel Programming for Multicore and Cluster Systems
by Thomas Rauber and Gudula Rünger, Chap. Chapter "Message-Passing Programming" (Springer, 2017)

See Chap 2: Introduction to HPC
with MPI for Data Science. Nielsen (Springer, 2016)

Open MPI v3.0.1 documentation

Open MPI commands (section 1 man pages)

mpic++ mpicc mpicxx mpiexec mpif77 mpif90 mpifort mpirun ompi-clean ompi-dvm ompi-ps ompi-server ompi-top ompi_info opal_wrapper orte-clean orte-dvm orte-info orte-ps orte-server orte-top orted orterun oshc++ oshcc oshcxx oshfort oshmem_info oshrun shmemc++ shmemcc shmemcxx shmemfort shmemrun

Open MPI general information (section 7 man pages)

ompi_crcp opal_crs orte_filem orte_hosts orte_snapc orte_sstore

MPI API (section 3 man pages)

MPI MPIX_Query_cuda_support MPI_Abort MPI_Accumulate MPI_Add_error_class MPI_Add_error_code MPI_Add_error_string MPI_Address MPI_Aint_add MPI_Aint_diff MPI_Allgather MPI_Allgatherv MPI_Alloc_mem MPI_Allreduce MPI_Alltoall MPI_Alltoallv MPI_Alltoallw MPI_Attr_delete MPI_Attr_get MPI_Attr_put MPI_Barrier MPI_Bcast MPI_Bsend MPI_Bsend_init MPI_Buffer_attach MPI_Buffer_detach MPI_Cancel MPI_Cart_coords MPI_Cart_create MPI_Cart_get MPI_Cart_map MPI_Cart_rank MPI_Cart_shift MPI_Cart_sub MPI_Cartdim_get MPI_Close_port MPI_Comm_accept MPI_Comm_c2f MPI_Comm_call_errhandler MPI_Comm_compare MPI_Comm_connect MPI_Comm_create MPI_Comm_create_errhandler MPI_Comm_create_group MPI_Comm_create_keyval MPI_Get_library_version MPI_Get_processor_name MPI_Get_version MPI_Graph_create MPI_Graph_get MPI_Graph_map MPI_Graph_neighbors MPI_Graph_neighbors_count MPI_Graphdims_get MPI_Grequest_complete MPI_Grequest_start MPI_Group_c2f MPI_Group_compare MPI_Group_difference MPI_Group_excl MPI_Group_f2c MPI_Group_free MPI_Group_incl MPI_Group_intersection MPI_Group_range_excl MPI_Group_range_incl MPI_Group_rank MPI_Group_size MPI_Group_translate_ranks MPI_Group_union MPI_Iallgather MPI_Iallgatherv MPI_Iallreduce MPI_Ialltoall MPI_Ialltoallv MPI_Ialltoallw MPI_Ibarrier MPI_Ibcast MPI_Ibsend MPI_Iexscan MPI_Igather MPI_Igatherv MPI_Improbe MPI_Imrecv MPI_Ineighbor_allgather MPI_Ineighbor_allgatherv MPI_Ineighbor_alltoall MPI_Ineighbor_alltoallv MPI_Ineighbor_alltoallw MPI_Info_c2f MPI_T_pvar_start MPI_T_pvar_stop MPI_T_pvar_write MPI_Test MPI_Test_cancelled MPI_Testall MPI_Testany MPI_Testsome MPI_Topo_test MPI_Type_c2f MPI_Type_commit MPI_Type_contiguous MPI_Type_create_darray MPI_Type_create_f90_complex MPI_Type_create_f90_integer MPI_Type_create_f90_real MPI_Type_create_hindexed MPI_Type_create_hindexed_block MPI_Type_create_hvector MPI_Type_create_indexed_block MPI_Type_create_keyval MPI_Type_create_resized MPI_Type_create_struct MPI_Type_create_subarray MPI_Type_delete_attr MPI_Type_dup MPI_Type_extent MPI_Type_f2c MPI_Type_free MPI_Type_free_keyval MPI_Type_get_attr MPI_Type_get_contents MPI_Type_get_envelope MPI_Type_get_extent MPI_Type_get_extent_x MPI_Type_get_name MPI_Type_get_true_extent MPI_Type_get_true_extent_x MPI_Type_hindexed MPI_Type_hvector MPI_Type_indexed MPI_Type_lb MPI_Type_match_size MPI_Type_set_attr MPI_Type_set_name shmem_float_prod_to shmem_float_put shmem_float_put_nbi shmem_float_set shmem_float_sum_to shmem_float_swap shmem_free shmem_get128 shmem_get128_nbi shmem_get16_nbi shmem_get32 shmem_get32_nbi shmem_get64 shmem_get64_nbi shmem_get8_nbi shmem_getmem shmem_getmem_nbi shmem_global_exit shmem_iget128 shmem_iget32 shmem_iget64 shmem_info_get_nam shmem_info_get_vers shmem_init shmem_int_add shmem_int_and_to_a shmem_int_cswap shmem_int_fadd shmem_int_fetch shmem_int_finc shmem_int_g shmem_int_get shmem_int_get_nbi shmem_int_iget shmem_int_inc shmem_int_iput shmem_int_max_to_a shmem_int_min_to_a shmem_int_or_to_all shmem_int_p shmem_int_prod_to_a shmem_int_put shmem_int_put_nbi shmem_int_set shmem_int_sum_to_a

https://www.open-mpi.org

https://computing.llnl.gov/tutorials/openMP/

Open MPI v3.0.1 documentation https://www.open-mpi.org/doc/current/

MPI_Comm_delete_attr MPI_Comm_disconnect MPI_Comm_dup MPI_Comm_dup_with_info MPI_Comm_f2c MPI_Comm_free MPI_Comm_free_keyval MPI_Comm_get_attr MPI_Comm_get_errhandler MPI_Comm_get_info MPI_Comm_get_name MPI_Comm_get_parent MPI_Comm_group MPI_Comm_idup MPI_Comm_join MPI_Comm_rank MPI_Comm_remote_group MPI_Comm_remote_size MPI_Comm_set_attr MPI_Comm_set_errhandler MPI_Comm_set_info MPI_Comm_set_name MPI_Comm_size MPI_Comm_spawn MPI_Comm_spawn_multiple MPI_Comm_split MPI_Comm_split_type MPI_Comm_test_inter MPI_Compare_and_swap MPI_Dims_create MPI_Dist_graph_create MPI_Dist_graph_create_adjacent MPI_Dist_graph_neighbors MPI_Dist_graph_neighbors_count MPI_Errhandler_create MPI_Errhandler_free MPI_Errhandler_get MPI_Errhandler_set MPI_Error_class MPI_Error_string MPI_Exscan MPI_Fetch_and_op MPI_File_c2f MPI_File_call_errhandler MPI_File_close MPI_File_create_errhandler MPI_File_delete MPI_File_f2c MPI_File_get_amode MPI_File_get_atomicity MPI_File_get_byte_offset MPI_File_get_errhandler MPI_File_get_group MPI_File_get_info MPI_File_get_position MPI_File_get_position_shared MPI_File_get_size MPI_File_get_type_extent MPI_File_get_view MPI_File_iread MPI_File_iread_all MPI_File_iread_at MPI_File_iread_at_all MPI_File_iread_shared MPI_File_iwrite MPI_File_iwrite_all MPI_File_iwrite_at MPI_File_iwrite_at_all MPI_File_iwrite_shared MPI_File_open MPI_File_preallocate MPI_File_read MPI_File_read_all MPI_File_read_all_begin MPI_File_read_all_end MPI_Info_create MPI_Info_delete MPI_Info_dup MPI_Info_env MPI_Info_f2c MPI_Info_free MPI_Info_get MPI_Info_get_nkeys MPI_Info_get_nthkey MPI_Info_get_valuelen MPI_Info_set MPI_Init MPI_Init_thread MPI_Initialized MPI_Intercomm_create MPI_Intercomm_merge MPI_Iprobe MPI_Irecv MPI_Ireduce MPI_Ireduce_scatter MPI_Ireduce_scatter_block MPI_Irsend MPI_Is_thread_main MPI_Iscan MPI_Iscatter MPI_Iscatterv MPI_Isend MPI_Issend MPI_Keyval_create MPI_Keyval_free MPI_Lookup_name MPI_Message_c2f MPI_Message_f2c MPI_Mprobe MPI_Mrecv MPI_Neighbor_allgather MPI_Neighbor_allgatherv MPI_Neighbor_alltoall MPI_Neighbor_alltoallv MPI_Neighbor_alltoallw MPI_Op_c2f MPI_Op_commutative MPI_Op_create MPI_Op_f2c MPI_Op_free MPI_Open_port MPI_Pack MPI_Pack_external MPI_Pack_external_size MPI_Pack_size MPI_Pcontrol MPI_Probe MPI_Publish_name MPI_Put MPI_Query_thread MPI_Raccumulate MPI_Recv MPI_Recv_init MPI_Reduce MPI_Reduce_local MPI_Reduce_scatter MPI_Reduce_scatter_block MPI_Register_datarep MPI_Request_c2f MPI_Request_f2c MPI_Request_free MPI_Request_get_status MPI_Rget MPI_Rget_accumulate MPI_Rput MPI_Rsend MPI_Rsend_init MPI_Scan MPI_Scatter MPI_Scatterv MPI_Type_size MPI_Type_size_x MPI_Type_struct MPI_Type_ub MPI_Type_vector MPI_Unpack MPI_Unpack_external MPI_Unpublish_name MPI_Wait MPI_Waitall MPI_Waitany MPI_Waitsome MPI_Win_allocate MPI_Win_allocate_shared MPI_Win_attach MPI_Win_c2f MPI_Win_call_errhandler MPI_Win_complete MPI_Win_create MPI_Win_create_dynamic MPI_Win_create_errhandler MPI_Win_create_keyval MPI_Win_delete_attr MPI_Win_detach MPI_Win_f2c MPI_Win_fence MPI_Win_flush MPI_Win_flush_all MPI_Win_flush_local MPI_Win_flush_local_all MPI_Win_free MPI_Win_free_keyval MPI_Win_get_attr MPI_Win_get_errhandler MPI_Win_get_group MPI_Win_get_info MPI_Win_get_name MPI_Win_lock MPI_Win_lock_all MPI_Win_post MPI_Win_set_attr MPI_Win_set_errhandler MPI_Win_set_info MPI_Win_set_name MPI_Win_shared_query MPI_Win_start MPI_Win_sync MPI_Win_test MPI_Win_unlock MPI_Win_unlock_all MPI_Win_wait MPI_Wtick MPI_Wtime OMPI_Affinity_str OpenMPI OpenSHMEM _my_pe _num_pes intro_shmem shfree shmalloc shmem_addr_accessible shmem_align shmem_alltoall32 shmem_alltoall64 shmem_alltoalls32 shmem_alltoalls64 shmem_barrier shmem_barrier_all shmem_broadcast32 shmem_broadcast64 shmem_char_g shmem_char_get shmem_char_get_nbi shmem_char_p shmem_int_swap shmem_int_wait shmem_int_wait_until shmem_int_xor_to_all shmem_iput128 shmem_iput32 shmem_iput64 shmem_long_add shmem_long_and_to_all shmem_long_cswap shmem_long_fadd shmem_long_fetch shmem_long_finc shmem_long_g shmem_long_get shmem_long_get_nbi shmem_long_iget shmem_long_inc shmem_long_iput shmem_long_max_to_all shmem_long_min_to_all shmem_long_or_to_all shmem_long_p shmem_long_prod_to_all shmem_long_put shmem_long_put_nbi shmem_long_set shmem_long_sum_to_all shmem_long_swap shmem_long_wait shmem_long_wait_until shmem_long_xor_to_all shmem_longdouble_g shmem_longdouble_get shmem_longdouble_get_nbi shmem_longdouble_iget shmem_longdouble_iput shmem_longdouble_max_to_all shmem_longdouble_min_to_all shmem_longdouble_p shmem_longdouble_prod_to_all shmem_longdouble_put shmem_longdouble_put_nbi shmem_longlong_add shmem_longlong_and_to_all shmem_longlong_cswap shmem_longlong_fadd shmem_longlong_fetch shmem_longlong_finc shmem_longlong_g shmem_longlong_get shmem_longlong_get_nbi shmem_longlong_iget shmem_longlong_inc shmem_longlong_iput shmem_longlong_max_to_all shmem_longlong_min_to_all shmem_longlong_or_to_all shmem_longlong_p shmem_longlong_prod_to_all shmem_longlong_put shmem_longlong_put_nbi shmem_longlong_set shmem_longlong_sum_to_all shmem_longlong_swap shmem_longlong_wait shmem_longlong_wait_until shmem_longlong_xor_to_all shmem_malloc shmem_my_pe shmem_n_pes shmem_pe_accessible shmem_ptr shmem_put128 shmem_put128_nbi

Open MPI v3.0.1 documentation https://www.open-mpi.org

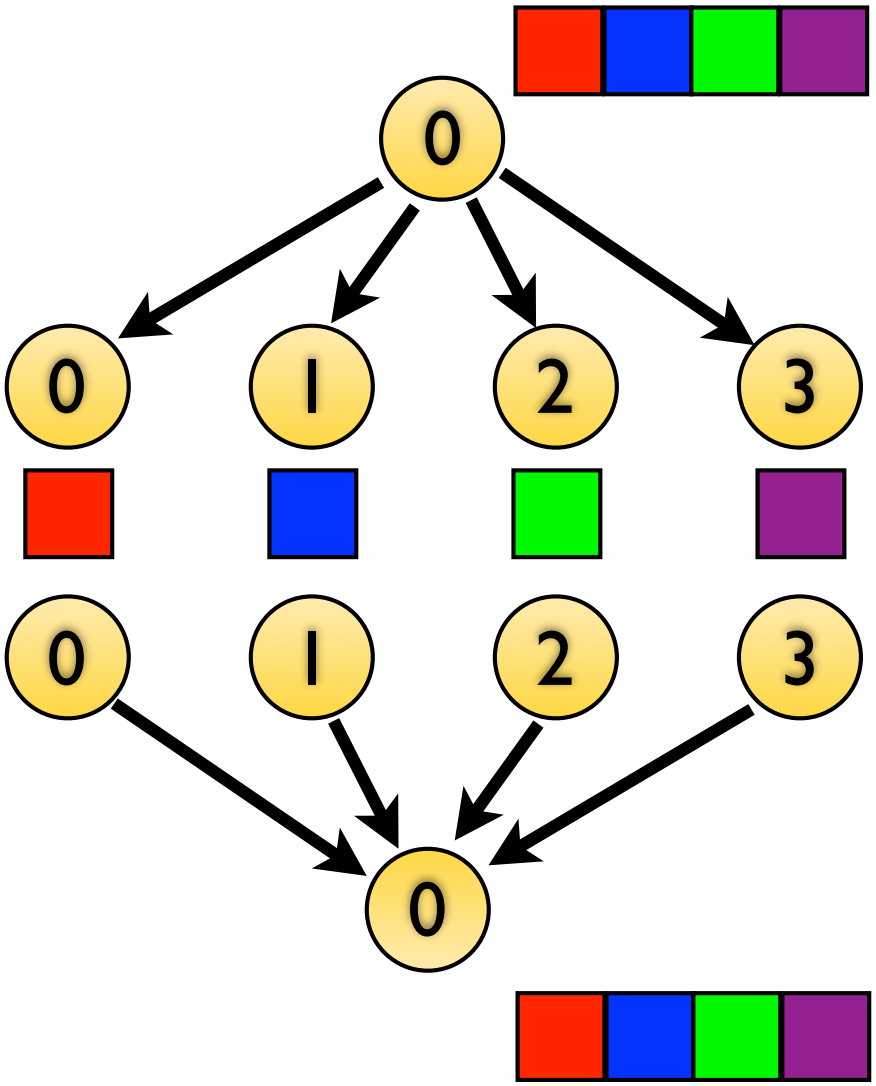
MPI_File_read_at MPI_File_read_at_all MPI_File_read_at_all_begin MPI_File_read_at_all_end MPI_File_read_ordered MPI_File_read_ordered_begin MPI_File_read_ordered_end MPI_File_read_shared MPI_File_seek MPI_File_seek_shared MPI_File_set_atomicity MPI_File_set_errhandler MPI_File_set_info MPI_File_set_size MPI_File_set_view MPI_File_sync MPI_File_write MPI_File_write_all MPI_File_write_all_begin MPI_File_write_all_end MPI_File_write_at MPI_File_write_at_all MPI_File_write_at_all_begin MPI_File_write_at_all_end MPI_File_write_ordered MPI_File_write_ordered_begin MPI_File_write_ordered_end MPI_File_write_shared MPI_Finalize MPI_Finalized MPI_Free_mem MPI_Gather MPI_Gatherv MPI_Get MPI_Get_accumulate MPI_Get_address MPI_Get_count MPI_Get_elements MPI_Get_elements_x MPI_Send MPI_Send_init MPI_Sendrecv MPI_Sendrecv_replace MPI_Sizeof MPI_Ssend MPI_Ssend_init MPI_Start MPI_Startall MPI_Status_c2f MPI_Status_f2c MPI_Status_set_cancelled MPI_Status_set_elements MPI_Status_set_elements_x MPI_T_category_changed MPI_T_category_get_categories MPI_T_category_get_cvars MPI_T_category_get_info MPI_T_category_get_num MPI_T_category_get_pvars MPI_T_cvar_get_info MPI_T_cvar_get_num MPI_T_cvar_handle_alloc MPI_T_cvar_handle_free MPI_T_cvar_read MPI_T_cvar_write MPI_T_enum_get_info MPI_T_enum_get_item MPI_T_finalize MPI_T_init_thread MPI_T_pvar_get_info MPI_T_pvar_get_num MPI_T_pvar_handle_alloc MPI_T_pvar_handle_free MPI_T_pvar_read MPI_T_pvar_readreset MPI_T_pvar_reset MPI_T_pvar_session_create MPI_T_pvar_session_free shmem_char_put shmem_char_put_nbi shmem_clear_cache_inv shmem_clear_cache_line_inv shmem_clear_lock shmem_collect32 shmem_collect64 shmem_complexd_prod_to_all shmem_complexd_sum_to_all shmem_complexf_prod_to_all shmem_complexf_sum_to_all shmem_double_fetch shmem_double_g shmem_double_get shmem_double_get_nbi shmem_double_iget shmem_double_iput shmem_double_max_to_all shmem_double_min_to_all shmem_double_p shmem_double_prod_to_all shmem_double_put shmem_double_put_nbi shmem_double_set shmem_double_sum_to_all shmem_double_swap shmem_fcollect32 shmem_fcollect64 shmem_fence shmem_finalize shmem_float_fetch shmem_float_g shmem_float_get shmem_float_get_nbi shmem_float_iget shmem_float_iput shmem_float_max_to_all shmem_float_min_to_all shmem_float_p shmem_put16_nbi shmem_put32 shmem_put32_nbi shmem_put64 shmem_put64_nbi shmem_put8_nbi shmem_putmem shmem_putmem_nbi shmem_quiet shmem_realloc shmem_set_cache_inv shmem_set_cache_line_inv shmem_set_lock shmem_short_and_to_all shmem_short_g shmem_short_get shmem_short_get_nbi shmem_short_iget shmem_short_iput shmem_short_max_to_all shmem_short_min_to_all shmem_short_or_to_all shmem_short_p shmem_short_prod_to_all shmem_short_put shmem_short_put_nbi shmem_short_sum_to_all shmem_short_wait shmem_short_wait_until shmem_short_xor_to_all shmem_swap shmem_test_lock shmem_udcflush shmem_udcflush_line shmem_wait shmem_wait_until shmemalign shrealloc start_pes

Page last modified: 2 Mar-20

```python
# Run with mpiexec -n 4 python ./mpi-montecarlo.py -s 1000000
from mpi4py import MPI
import argparse
import random
from math import pi

comm = MPI.COMM_WORLD
# This script will be executed once per process
size = comm.Get_size() # Number of processes available
rank = comm.Get_rank() # Rank of the process executing this process

def compute_pi(args):
    n = int(args.steps / size)

    if rank == 0:
        steps = [n] * size
    else:
        steps = None

    steps = comm.scatter(steps, root=0)
    print(f"Rank={rank}, steps={steps}")

    s = sample_pi(steps)

    result = comm.gather(s, root=0)
    if rank == 0:
        n_total = n*size
        s_total = sum(result)
        pi_est = (4.0*s_total)/n_total
        print(" Steps\tSuccess\tPi est.\tError")
        print("%6d\t%7d\t%1.5f\t%1.5f" % (n_total, s_total, pi_est, pi-pi_est))
```

MPI: Scatter/Gather

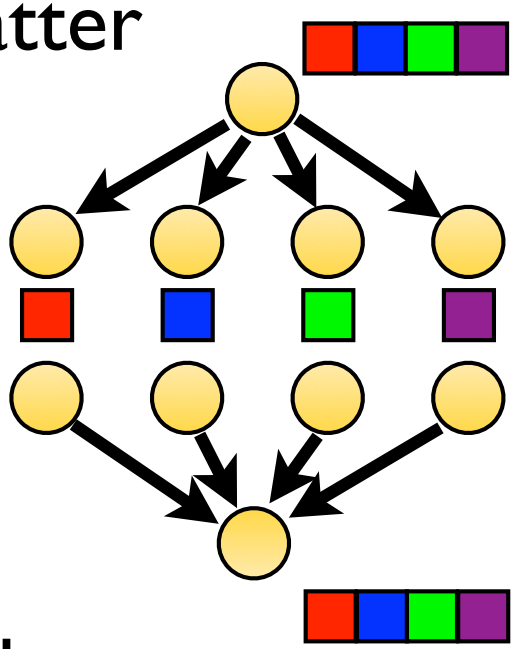
Convention: rank 0 process is root



See Sec. 2.3.1 Nielsen (2016)

MPI scatter/gather vs. mp.map

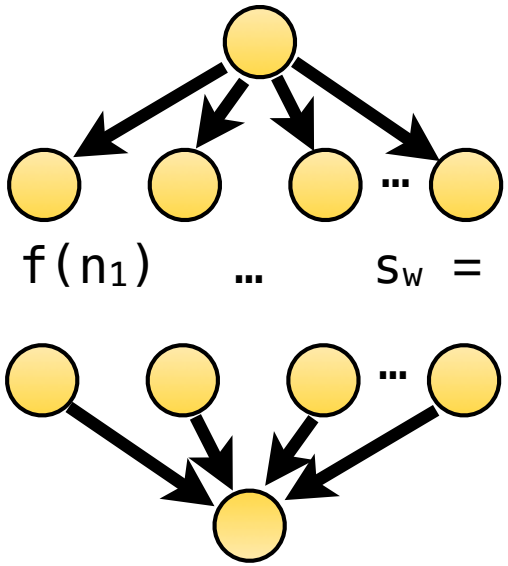
Scatter



Gather

```
s = p.map(f, [n] * w)
```

$[n_1, n_2, n_3, \ldots, n_w]$



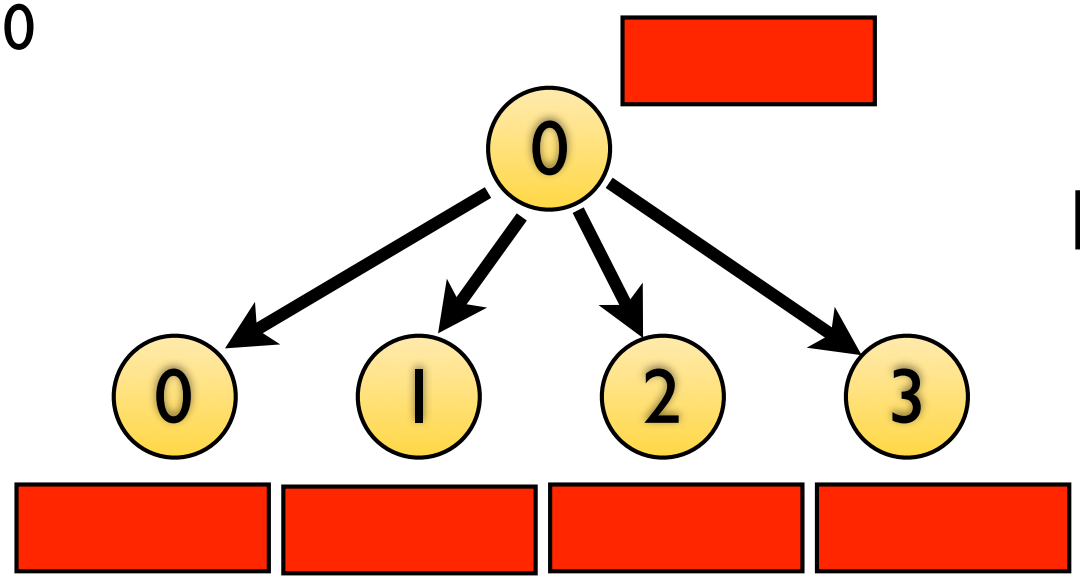
$s_1 = f(n_1)$ … $s_w = f(n_w)$

$[s_1, s_2, s_3, \ldots, s_w]$

See Sec. 2.3.1 Nielsen (2016)

MPI Broadcast

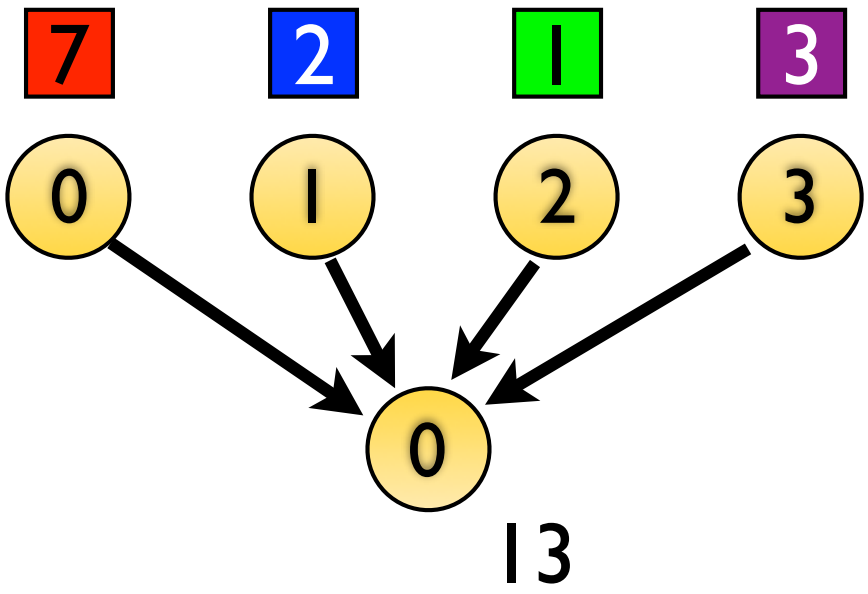
Convention: rank 0 process is root



Broadcast

See Sec. 2.3.1 Nielsen (2016)

MPI Reduce



Sum

Reduce supports binary operations w/ associative law: Sum, Prod, Max, Min, Argmax, Argmin, Logic operations,…

See Sec. 2.3.6 Nielsen (2016)

```
def compute_pi(args):
    n = int(args.steps / size)

    if rank == 0:
        steps = [n] * size
    else:
        steps = None

    steps = comm.scatter(steps, root=0)
    print(f"Rank={rank}, steps={steps}")

    s = sample_pi(steps)

    s_total = 0
    s = np.asarray(s)
    s_total = np.asarray(s_total)

    comm.Reduce(s, s_total, op=MPI.SUM, root=0)

    if rank == 0:
        n_total = n*size
        pi_est = (4.0*s_total)/n_total
        print(" Steps\tSuccess\tPi est.\tError")
        print("%6d\t%7d\t%1.5f\t%1.5f" % (n_total, s_total, pi_est, pi-pi_est))
```

During a **gather** operation, when one message is passed from every one of the n nodes to the root, the total number of message passing operations is …

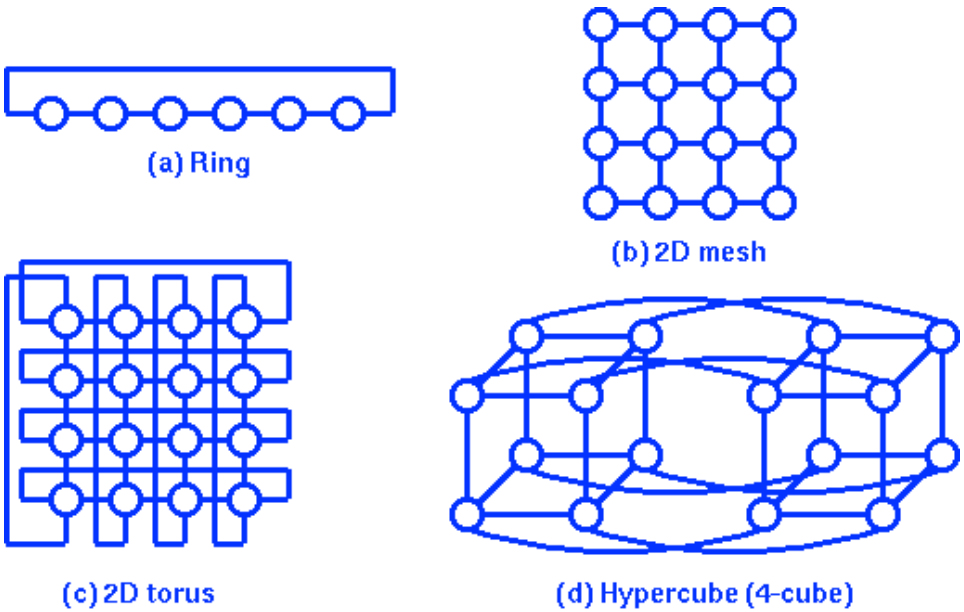
n nodes, n odd

For node farthest from root 1
second-farthest 2, 3rd-farthest 3
+2+3+4+5+…+ (n-1)/2 = 1/8 (n^2-1)
using Gauss

Here a message passing operation is passing the node's own data, or a message received from another node. Assume that no data is aggregated.

A	B	C	D	E
O(log n)	O(n)	O(n log n)	$O(n^2)$	$O(n^3)$

Reduce is *not just* Gather + Function call

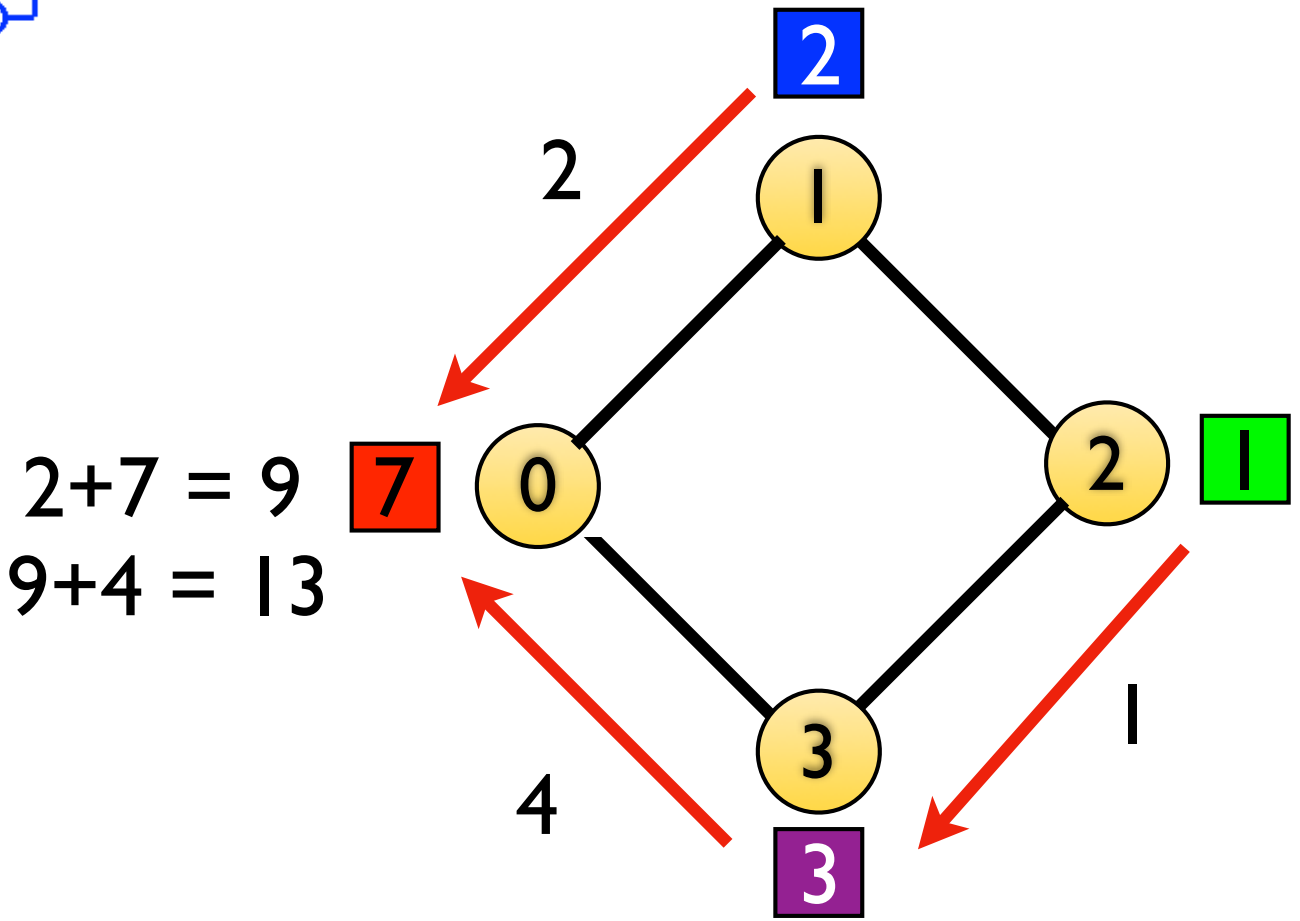


Example Interconnect topologies

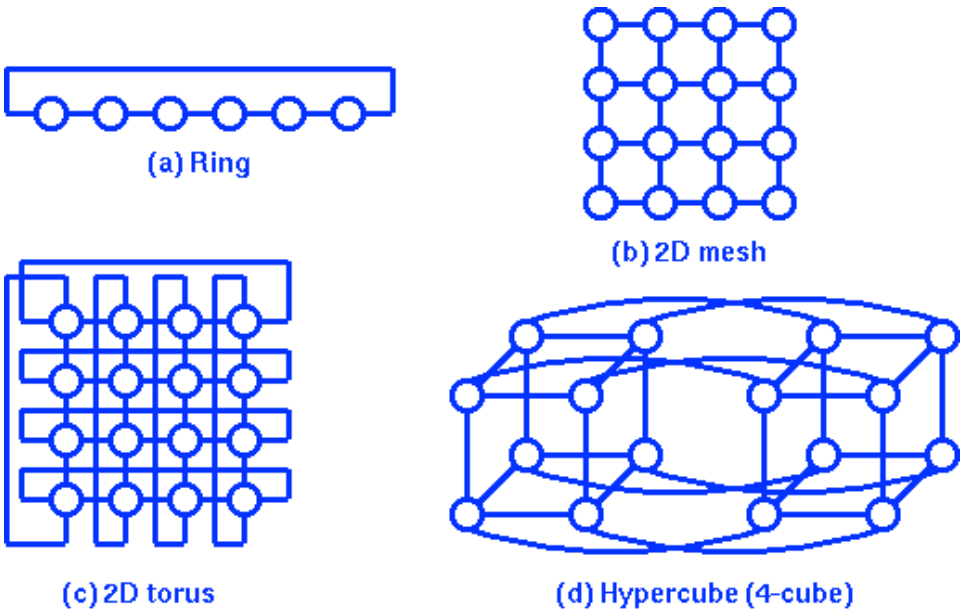
- Communication Bottlenecks close to MPI root
- Substantial Latencies for thousands of cores
- Broadcast, Gather, Scatter similarly adapt to interconnect topology

See Chap 3 of Nielsen (2016)

Reduce is *not* Gather + Function call



Importance of Reduce



Example Interconnect topologies

- Communication Bottlenecks close to root
- Substantial Latencies for thousands of cores
- Reduce operations yield O(nodes) message passing operations instead of $O(nodes^2)$
- Broadcast, Gather, Scatter similarly adapt to interconnect topology

See Chap 3 of Nielsen (2016)

Reflection

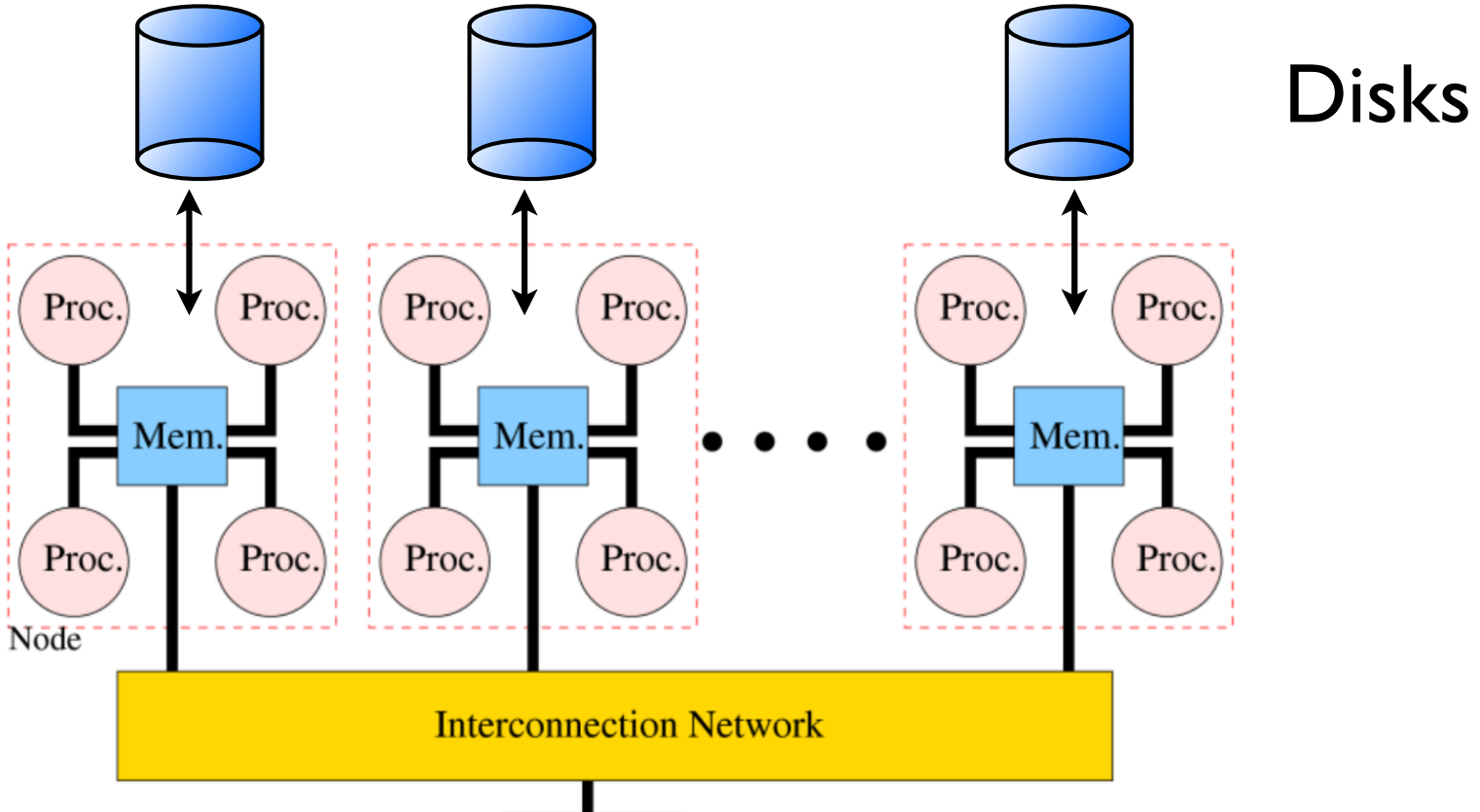
Architectures & Models

- SMP & Multithreading:
  - Shared memory, fast communication between all threads
- Distributed Memory and Message Passing
  - High latency, interconnection topology matters: limited #connections

Workload Comparison

	HPC	Data Science
CPU	Lots pf cycles Floating Point (FP) Little memory	Medium Mixed integer/FP Large memory
Data/File IO	Often very little IO	Massive IO Computations IO-bound
Tasks	**Standard Tools (e.g. matrix solvers)**	Custom tools and **frequently new, changing problems**

System architecture for data-centric computation



Aspects of data-centric computation

- Scale of Data: Parallel IO, Data-compute co-location
- Cost of computation: commodity hardware
- Reliability of systems: Design for failures
- Software is run once: support rapid development:
  - automated parallelization
  - restricted model of computation

Parallel programming models

Map Reduce

See Skiena, Data Science Design Manual, Chap 12.6

http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

Map/reduce assumptions

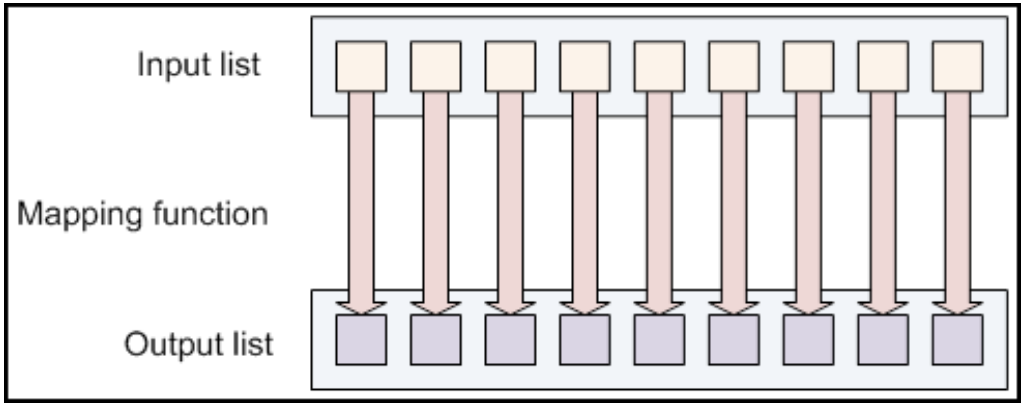
- Input files are distributed over nodes (implicit)
- All data are (key, value) tuples
- Main parallel operations are:
  - Map:
    $(key, value) \rightarrow (key_1, value_1)[,(key_2, value_2),\ldots]$
  - Reduce:
    $(key_1, [value_1, value_2, value_3, ...]) \rightarrow$
    $(key_{new}, value_{new})[,(key_{new2}, value_{new2}),\ldots]$

[ ] optional

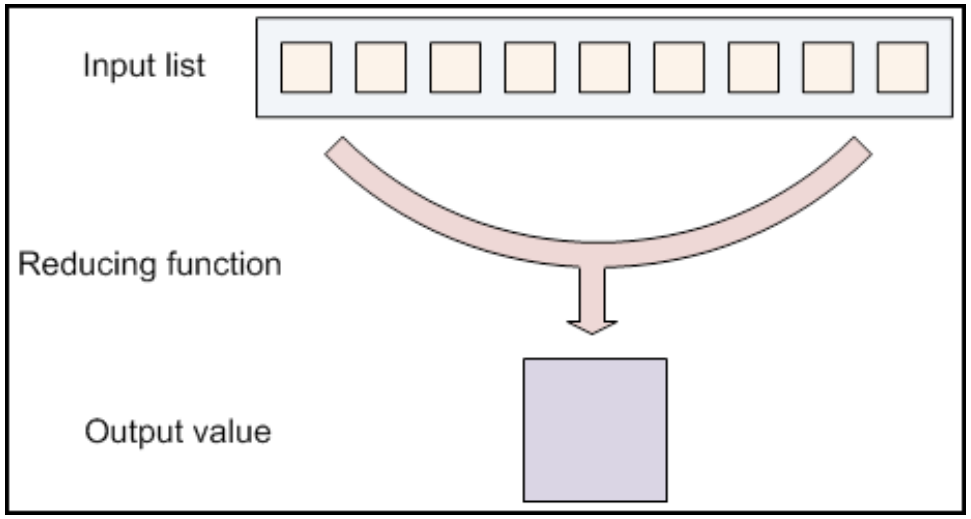
Example: Count number of links to webpages

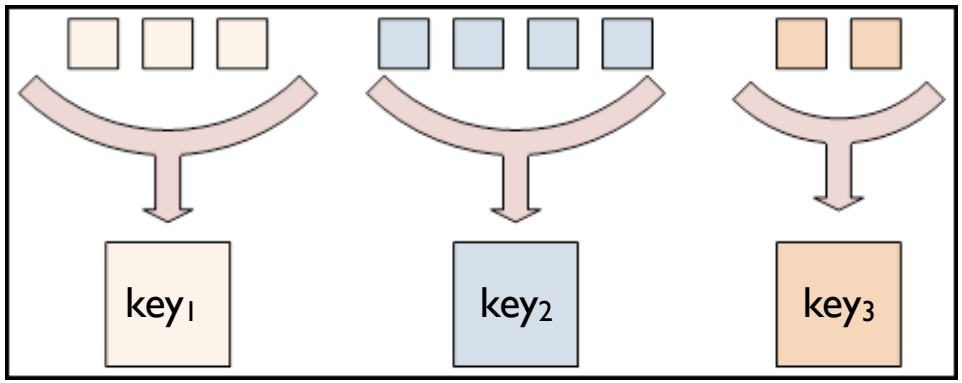
```
Map(DocumentURL,DocumentHTML):
    For each url in DocumentHTML return (url, 1)

Reduce(url, counts) # counts = [count1, count2,...]
    return (url, sum(counts))
```

Map



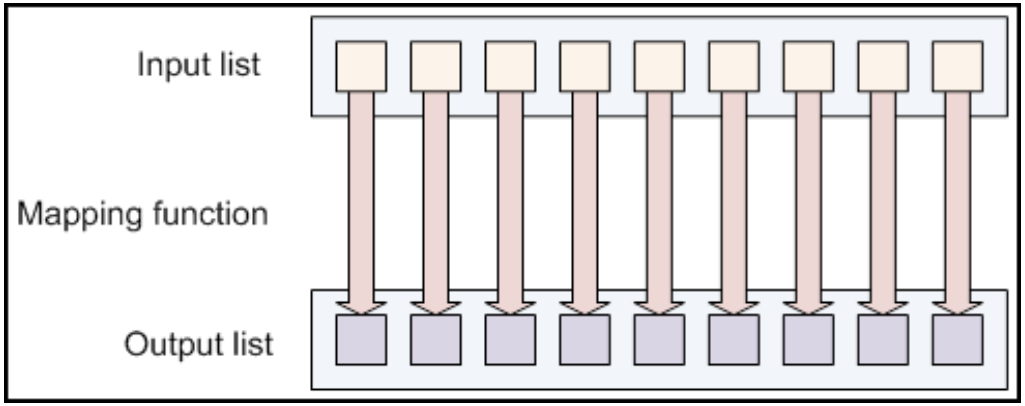
Reduce





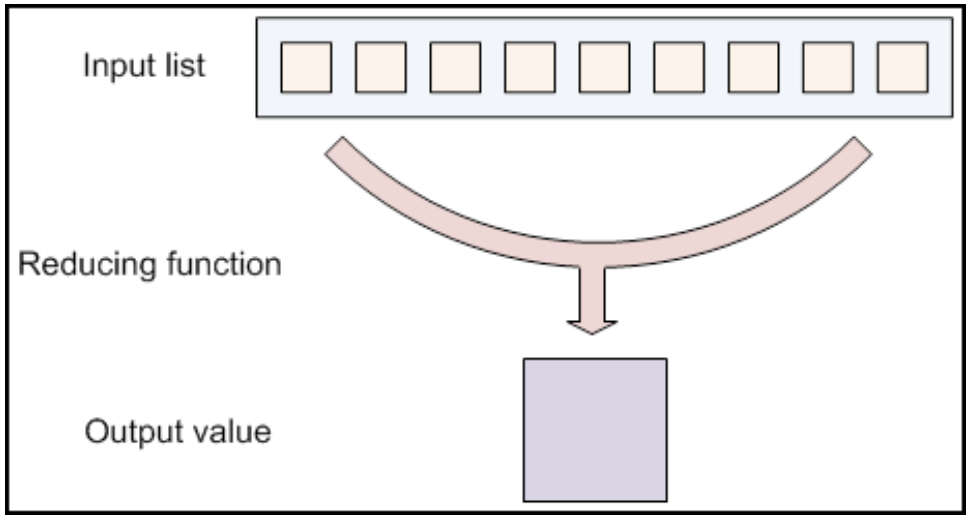
Reduce receives input aggregated by key

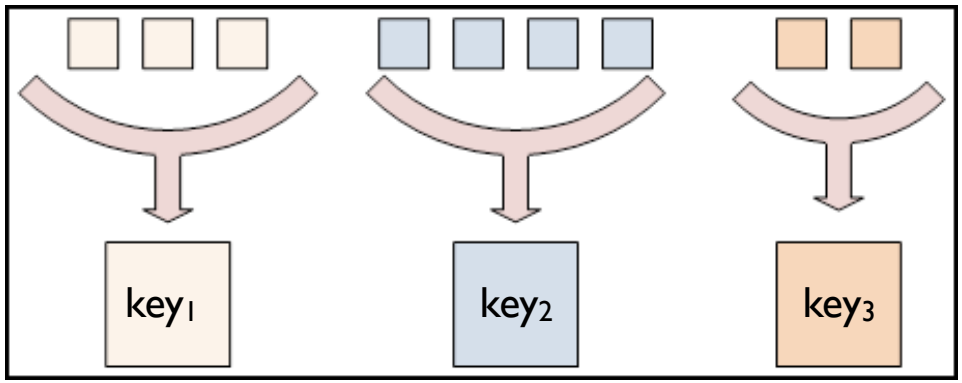
Map



Each map call only needs its input; no assertions about other inputs, execution order, assignment to nodes

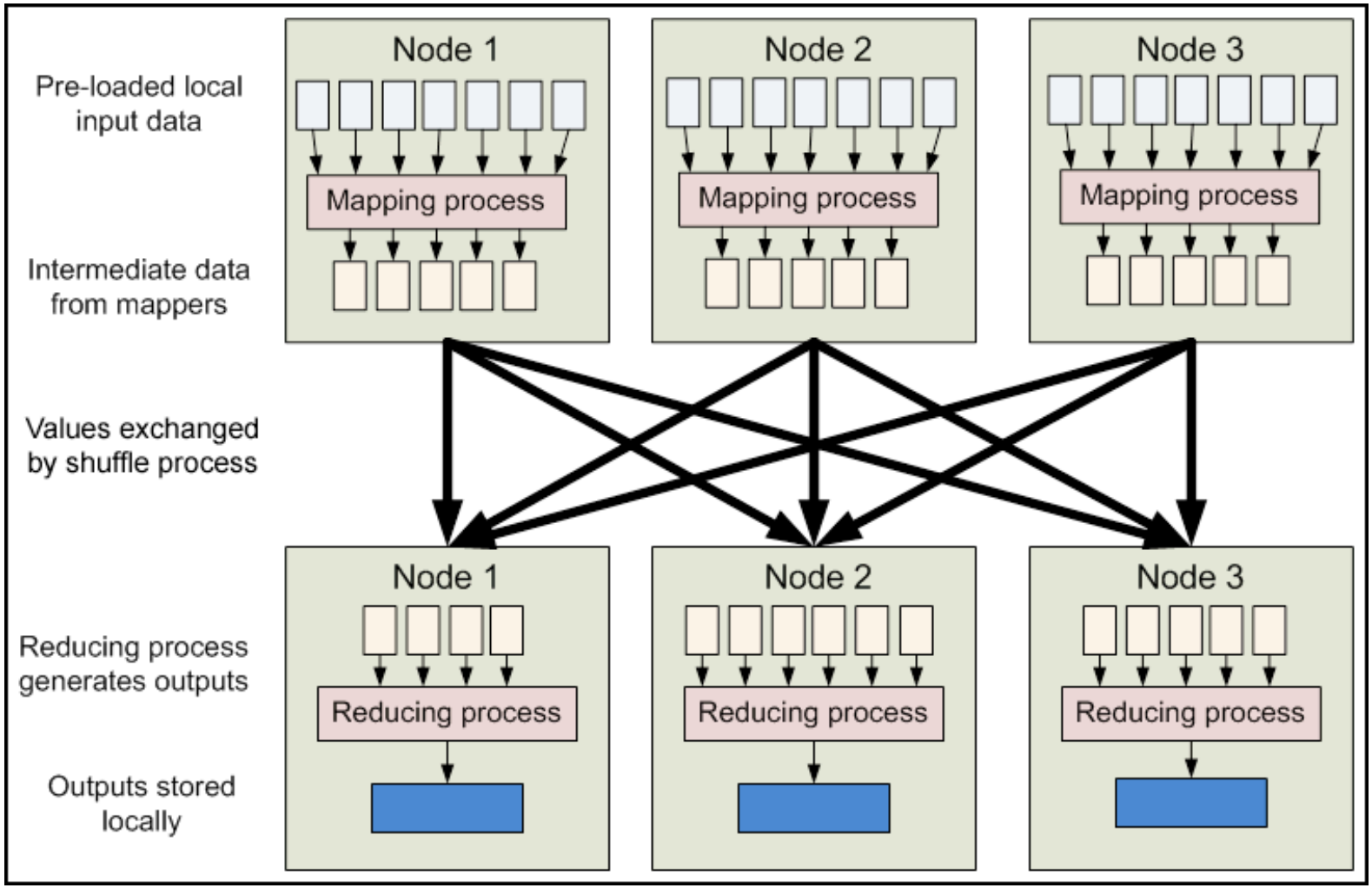
Reduce





Each reduce call receives all values for its key; no assertions about other keys, execution order, assignment to nodes

Map/Reduce Parallelism



MapReduce vs multi-threading and MPI

- No concept of process, thread nor hardware
- No control over data flow
- Reduce obtains all values with the same key
- *Substantial effort behind the scenes*

MrJob: Map/Reduce in Python

- https://github.com/Yelp/mrjob
- https://mrjob.readthedocs.io/en/stable/
- Pure Python
- Runs locally and in the cloud

```
""" Find duplicate keys in a file containing lines consisting of
    key,value
"""
from mrjob.job import MRJob

class FindDuplicates(MRJob):

    def mapper(self, _, line):
        key, value = line.split(',')
        yield (key, 1)

    def reducer(self, key, counts):
        s = sum(counts)
        if s > 1:
            yield (key, s)


if __name__ == '__main__':
    FindDuplicates.run()
```

Implementing wordcount with Map-reduce

```
schliep@MacBookPro13:> wc mpi-summary-statistics.py
      63     197    1918 mpi-summary-statistics.py
```

http://man7.org/linux/man-pages/man1/wc.1.html

```
from mrjob.job import MRJob

class WC(MRJob):

    def mapper(self, _, line):
        nr_words = len(line.split())
        nr_chars = len(line)
        yield ("words", nr_words)
        yield ("chars", nr_chars)

    def reducer(self, key, counts):
        if key == "words":
            yield ("words", sum(counts))
        elif key == "chars":
            yield ("chars", sum(counts))

if __name__ == '__main__':
    WC.run()
```

n = # lines

```
def mapper(self, _, line):
    nr_words = len(line.split())
    nr_chars = len(line)
    yield ("words", nr_words)
    yield ("chars", nr_chars)

def reducer(self, key, counts):
    if key == "words":
        yield ("words", sum(counts))
    elif key == "chars":
        yield ("chars", sum(counts))
```

The maximal speedup achievable by the mapper is …

Theoretical n
real-world k = #nodes

A	B	C	D	E
1	2	4	log(n)	n

n = # lines

```
from mrjob.job import MRJob

class WC(MRJob):

    def mapper(self, _, line):
        nr_words = len(line.split())
        nr_chars = len(line)
        yield ("words", nr_words)
        yield ("chars", nr_chars)

    def reducer(self, key, counts):
        if key == "words":
            yield ("words", sum(counts))
        elif key == "chars":
            yield ("chars", sum(counts))

if __name__ == '__main__':
    WC.run()
```

Creates 2n tuples

Parallel: up to one node per line

n = # lines

```
def mapper(self, _, line):
    nr_words = len(line.split())
    nr_chars = len(line)
    yield ("words", nr_words)
    yield ("chars", nr_chars)

def reducer(self, key, counts):
    if key == "words":
        yield ("words", sum(counts))
    elif key == "chars":
        yield ("chars", sum(counts))
```

The maximal speedup achievable by the reducer is …

A	B	C	D	E
1	2	4	log(n)	n

n = # lines

```
from mrjob.job import MRJob

class WC(MRJob):

    def mapper(self, _, line):
        nr_words = len(line.split())
        nr_chars = len(line)
        yield ("words", nr_words)
        yield ("chars", nr_chars)

    def reducer(self, key, counts):
        if key == "words":
            yield ("words", sum(counts))
        elif key == "chars":
            yield ("chars", sum(counts))

if __name__ == '__main__':
    WC.run()
```

Is called twice.
O(n) input

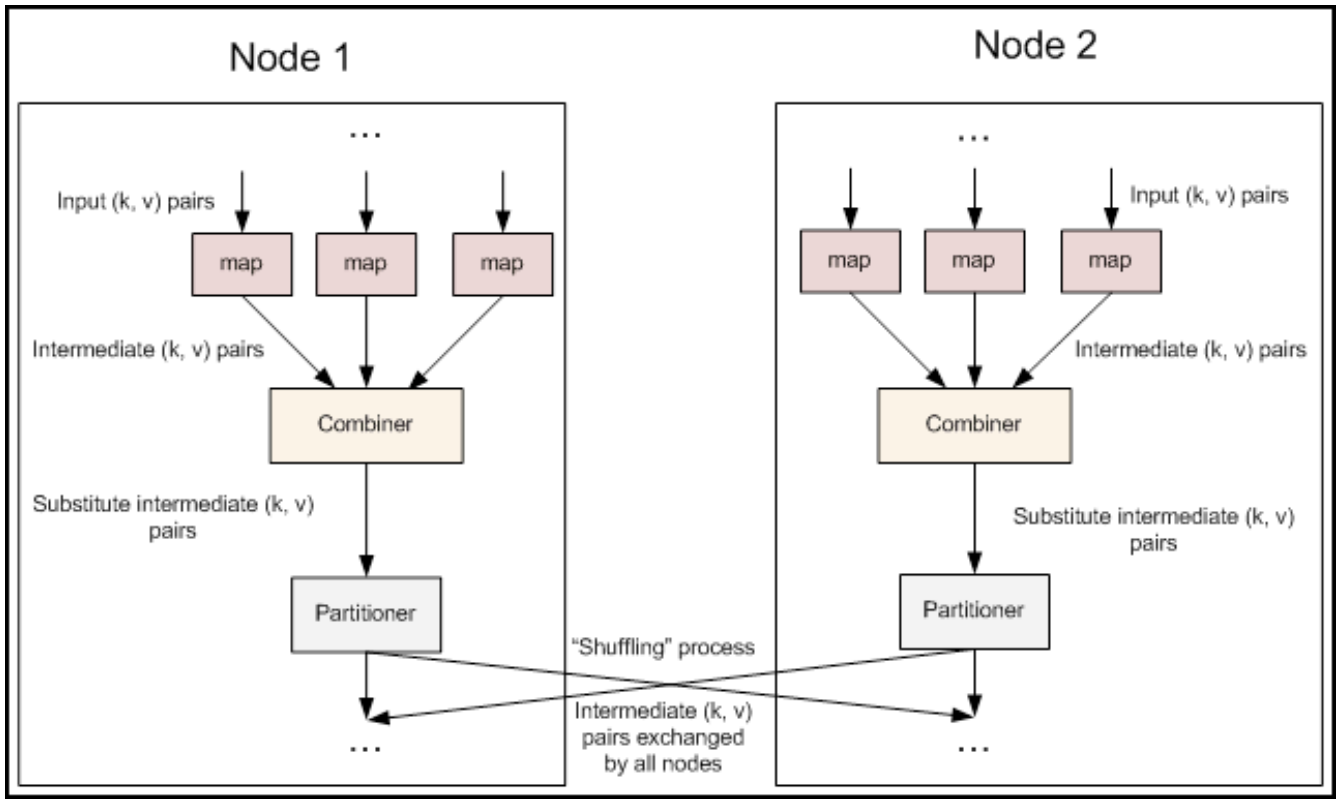
Parallel: up to two nodes

O(n) communication

```
class WC(MRJob):

    def mapper(self, _, line):
        nr_words = len(line.split())
        nr_chars = len(line)
        yield ("words", nr_words)
        yield ("chars", nr_chars)

    def combiner(self, key, counts):
        if key == "words":
            yield ("words", sum(counts))
        elif key == "chars":
            yield ("chars", sum(counts))

    def reducer(self, key, counts):
        if key == "words":
            yield ("words", sum(counts))
        elif key == "chars":
            yield ("chars", sum(counts))

if __name__ == '__main__':
    WC.run()
```

n = # lines
k= #nodes for mapper

Combiners


Node 1
Node 2
Input (k, v) pairs
map
map
map
Intermediate (k, v) pairs
Combiner
Substitute intermediate (k, v) pairs
Partitioner
"Shuffling" process
Intermediate (k, v) pairs exchanged by all nodes

```
class WC(MRJob):

    def mapper(self, _, line):
        nr_words = len(line.split())
        nr_chars = len(line)
        yield ("words", nr_words)
        yield ("chars", nr_chars)

    def combiner(self, key, counts):
        if key == "words":
            yield ("words", sum(counts))
        elif key == "chars":
            yield ("chars", sum(counts))

    def reducer(self, key, counts):
        if key == "words":
            yield ("words", sum(counts))
        elif key == "chars":
            yield ("chars", sum(counts))

if __name__ == '__main__':
    WC.run()
```

n = # lines
k= #nodes for mapper

Creates 2k tuples

As parallel as mapper

Same function as reducer on a single node

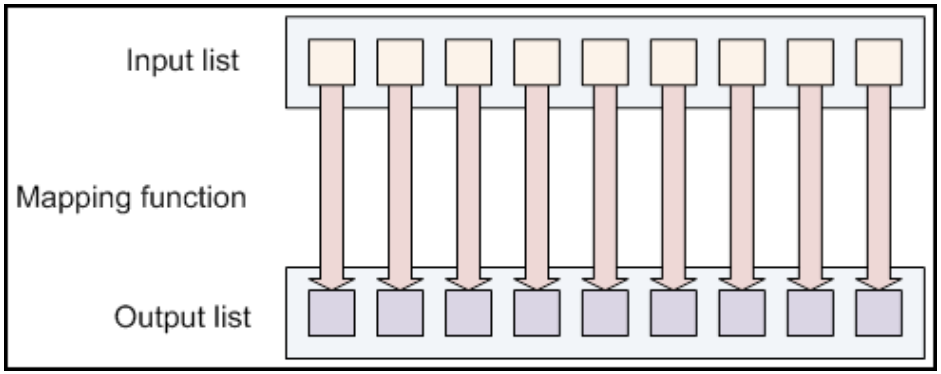
```
class WC(MRJob):

    def mapper(self, _, line):
        nr_words = len(line.split())
        nr_chars = len(line)
        yield ("words", nr_words)
        yield ("chars", nr_chars)

    def combiner(self, key, counts):
        if key == "words":
            yield ("words", sum(counts))
        elif key == "chars":
            yield ("chars", sum(counts))

    def reducer(self, key, counts):
        if key == "words":
            yield ("words", sum(counts))
        elif key == "chars":
            yield ("chars", sum(counts))

if __name__ == '__main__':
    WC.run()
```

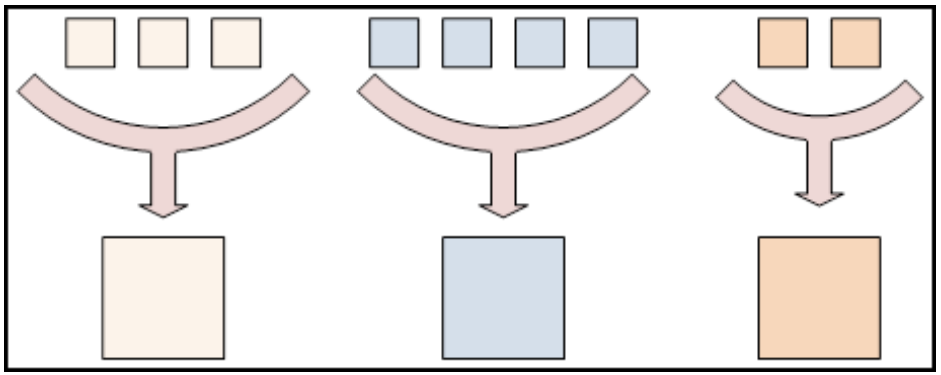
n = # lines
k= #nodes for mapper

Called twice with O(k) input

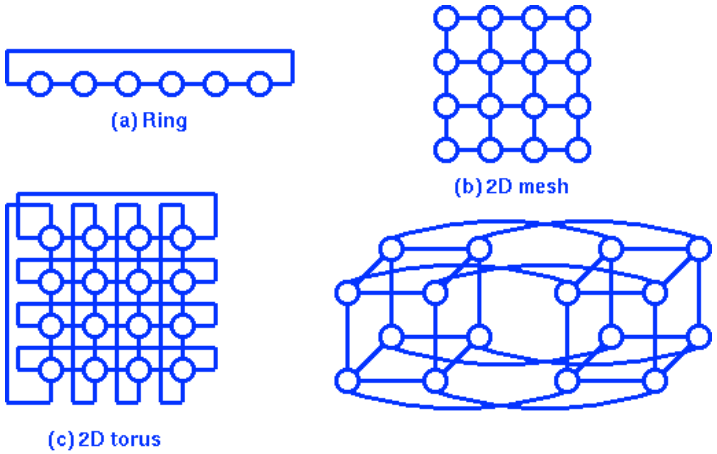
Parallel: two nodes

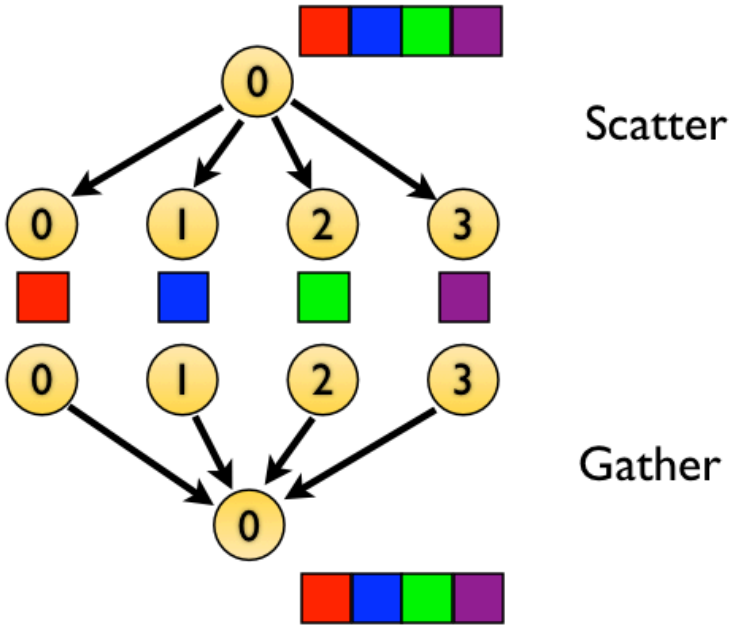
O(k) communication





keys define communication structure and effort implicitly





Explicit parallelization in MPI and threads

Questions & Answers from Chat

Q: Do mapper and combiner always use the same number of nodes? A: A combiner is run on all (key, value) pairs produced by a mapper on one node. So conceptually, yes.

Q: What is the speedup of mapper? A: The theoretical maximal speedup is n = #lines if executed on n nodes. Real-world: the speedup is roughly equal to the number of nodes the mapper is running on

Q: What does reduce do? A: Mapper produces (“words”, 2), (“words”, 4), (“words”, 5) [on some machines, maybe in parallel] then reduce gets called with “words”, [2,4,5] and thus can count the total number of words.

Q: Does that mean that the reducer has to sum n values together? A: If there is no combiner, yes. This is clearly bad, so that is why the combiner has been introduced. So on each of the nodes the combiner has to add up n/k values (this is parallel on k nodes; on each node multiple cores could be used). The reducer only get k values to add up.

Agenda for today:

- Using Multi-threaded programming
- Message passing:
  - Parallel primitives
  - Reduce
- Map/Reduce

Upcoming Deadlines

Python Programming 2
Not available until Apr 1 | Due Apr 15 at 10am | -/5 pts

Preparation for Lecture 4/15
Not available until Apr 1 | Due Apr 15 at 10am

Assignment 2
Available until Apr 20 | Due Apr 17 at 10am | -/14 pts