MPI.jl

Lauren Milechin

October 2020

Massachusetts Institute of Technology

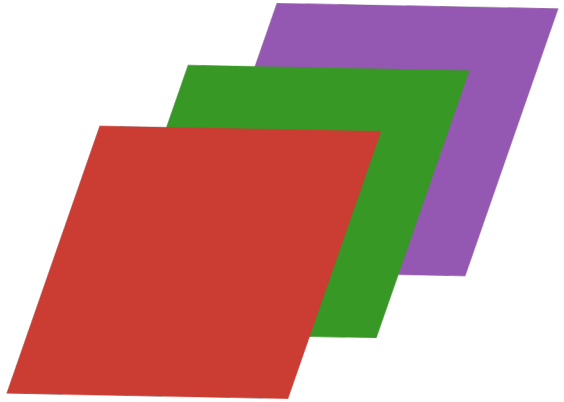
What is MPI?
Message Passing Interface

- MPI is a message-passing library interface standard
  - Specification, not an implementation
  - Library, not a language
  - Message-passing programming model (distributed memory model)

- Free, portable implementations available (e.g. MPICH, OpenMPI)

- MPI introduced in 1993 at SC Conference (SC'93)
  - Implementations < 1 year later
    - Vendors now provide optimized implementations (e.g. Intel: IMPI, Microsoft: MS-MPI, IBM, …)

- MPI.jl is a Julia wrapper for a C MPI implementations
  - Works with OpenMPI, MPICH, Intel MPI, Microsoft MPI, …



MPI.jl

MPI.jl

mpi_hello_world.c

```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char **argv){
    int rank, nproc;
    int name_len;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    /* Initialize MPI environment */
    MPI_Init(&argc, &argv);

    /* Get MPI process rank id */
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    /* Get number of MPI processes in this communicator */
    MPI_Comm_size(MPI_COMM_WORLD, &nproc);

    /* Print hello world message */
    printf("Hello world, I am rank %d out of %d processors\n",processor_name, rank, nproc);

    /* Finalize MPI environment */
    MPI_Finalize();
    return 0;
}
```

helloworld.jl

```julia
using MPI




# Initialize MPI environment
MPI.Init()

# Get MPI process rank id
rank = MPI.Comm_rank(MPI.COMM_WORLD)

# Get number of MPI processes in this communicator
nproc = MPI.Comm_size(MPI.COMM_WORLD)

# Print hello world message
print("Hello world, I am rank $(rank) of $(nproc) processors\n")
```

MPI.jl

mpi_hello_world.c

```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char **argv){
    int rank, nproc;
    int name_len;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    /* Initialize MPI environment */
    MPI_Init(&argc, &argv);

    /* Get MPI process rank id */
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    /* Get number of MPI processes in this communicator */
    MPI_Comm_size(MPI_COMM_WORLD, &nproc);

    /* Print hello world message */
    printf("Hello world, I am rank %d out of %d processors\n",processor_name, rank, nproc);

    /* Finalize MPI environment */
    MPI_Finalize();
    return 0;
}
```

```
lmilechin@d-14-12-2:~$ mpirun -np 4 helloworld
Hello world, I am rank 0 out of 4 processors
Hello world, I am rank 1 out of 4 processors
Hello world, I am rank 3 out of 4 processors
Hello world, I am rank 2 out of 4 processors
```

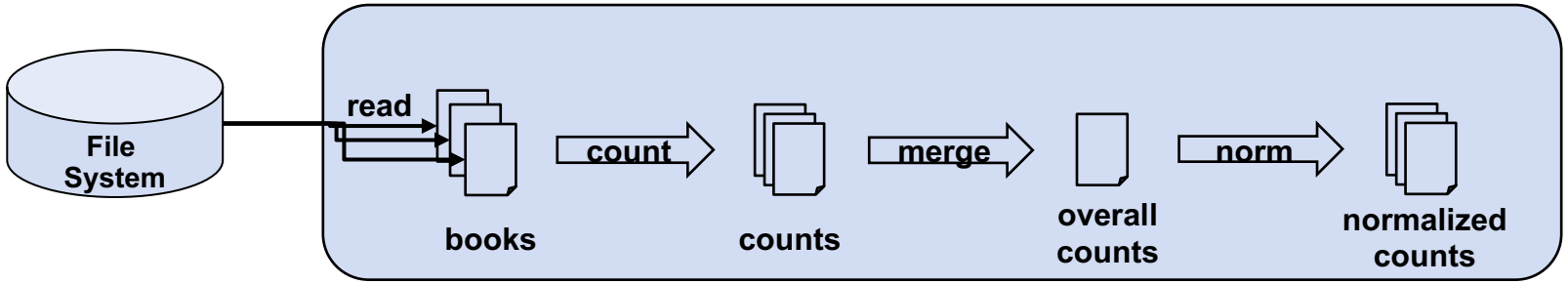
helloworld.jl

```julia
using MPI




# Initialize MPI environment
MPI.Init()

# Get MPI process rank id
rank = MPI.Comm_rank(MPI.COMM_WORLD)

# Get number of MPI processes in this communicator
nproc = MPI.Comm_size(MPI.COMM_WORLD)

# Print hello world message
print("Hello world, I am rank $(rank) of $(nproc) processors\n")
```

```
lmilechin@d-14-12-2:~$ mpirun -np 4 julia helloworld.jl
Hello world, I am rank 0 out of 4 processors
Hello world, I am rank 2 out of 4 processors
Hello world, I am rank 3 out of 4 processors
Hello world, I am rank 1 out of 4 processors
```

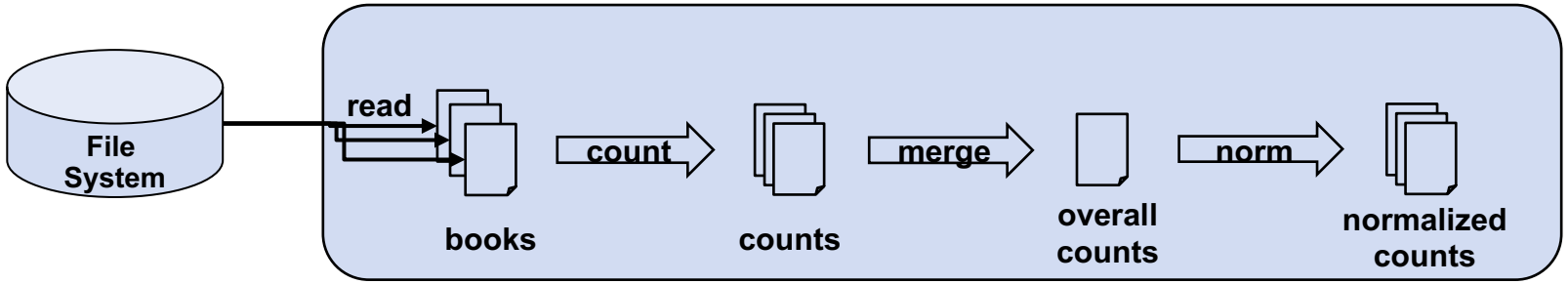
“Normalized” Word Count

- Several files of documents
- Want a summary of what they contain
  - What are the normalized word counts for each document?



"Normalized" Word Count

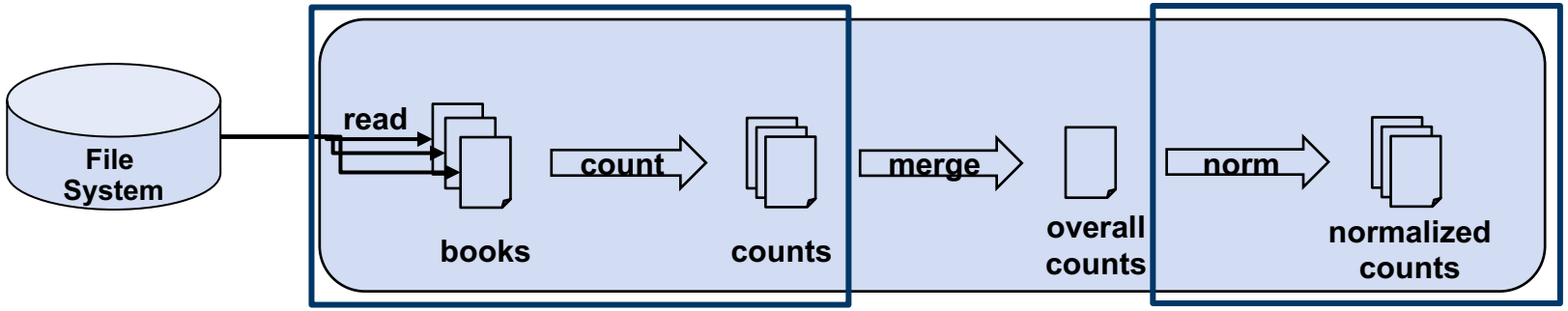
- Several files of documents
- Want a summary of what they contain
  - What are the normalized word counts for each document?



- Where is the independence?
- What are the data access patterns?
  - Where is the data and where does it need to go?

"Normalized" Word Count

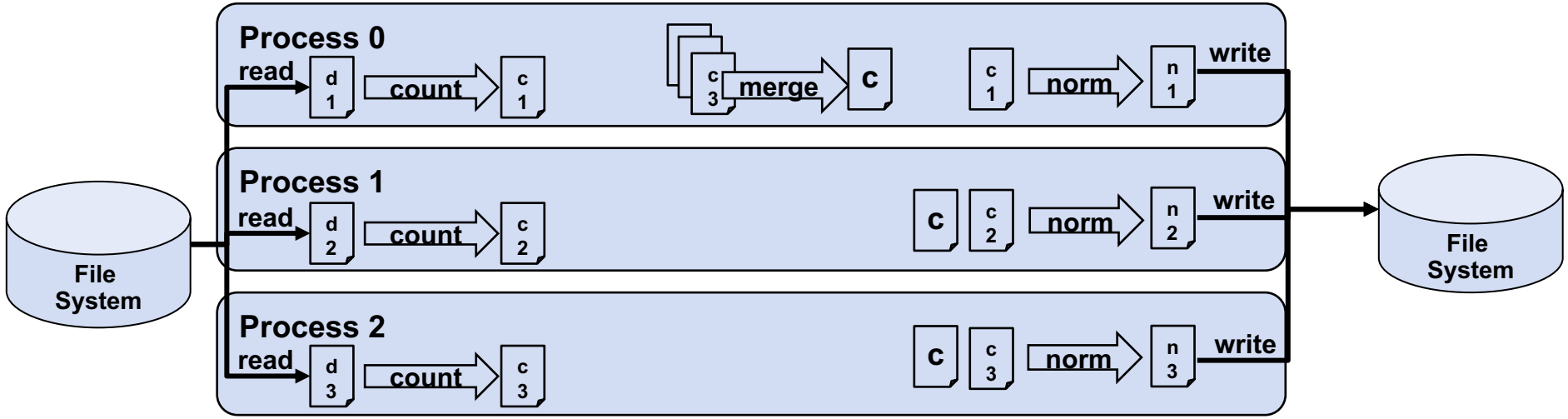
- Several files of documents
- Want a summary of what they contain
  - What are the normalized word counts for each document?



- Where is the independence?
- What are the data access patterns?
  - Where is the data and where does it need to go?

"Normalized" Word Count

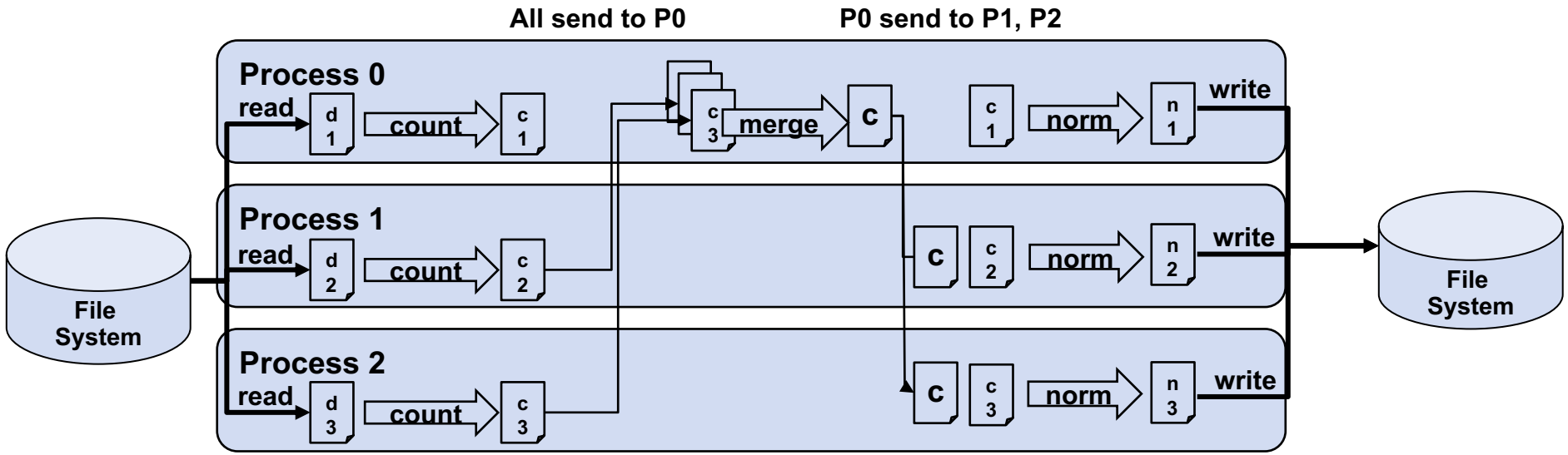
- Several files of documents
- Want a summary of what they contain
  - What are the normalized word counts for each document?



- What are the data access patterns?
  - Where is the data and where does it need to go?

"Normalized" Word Count

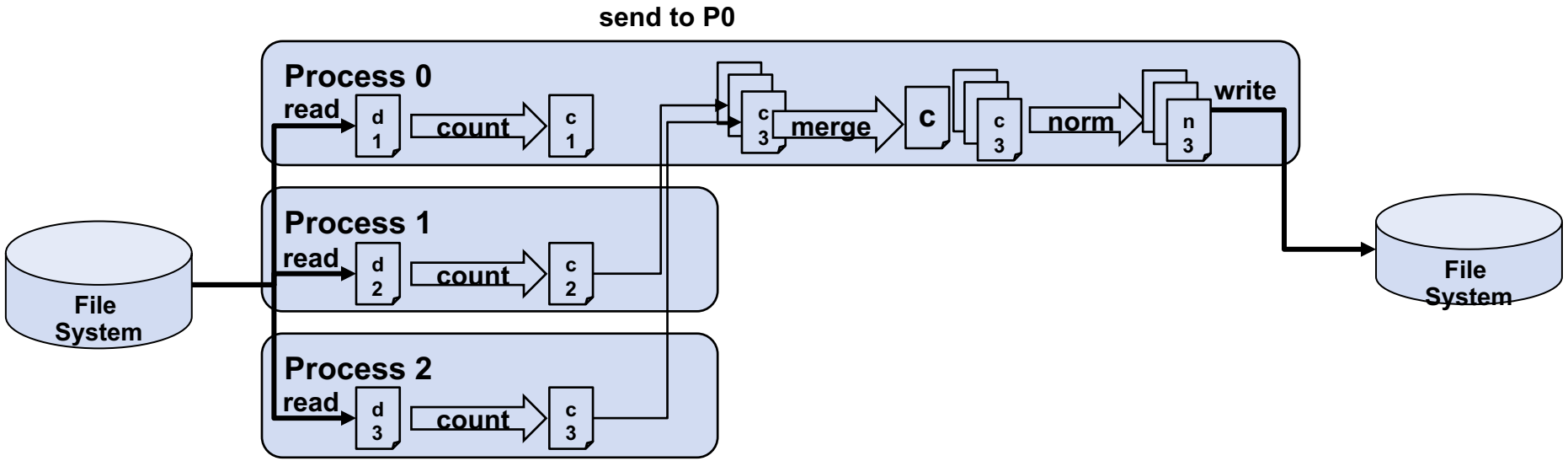
- Several files of documents
- Want a summary of what they contain
  - What are the normalized word counts for each document?



- What are the data access patterns?
  - Where is the data and where does it need to go?

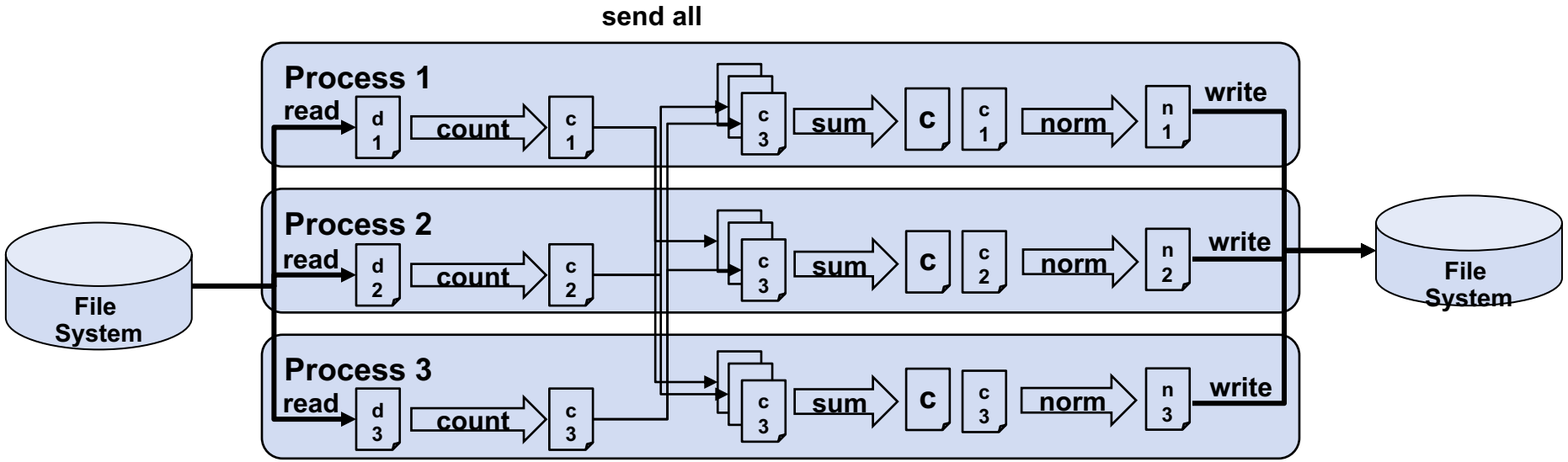
“Normalized” Word Count

- Several files of documents
- Want a summary of what they contain
  - What are the normalized word counts for each document?

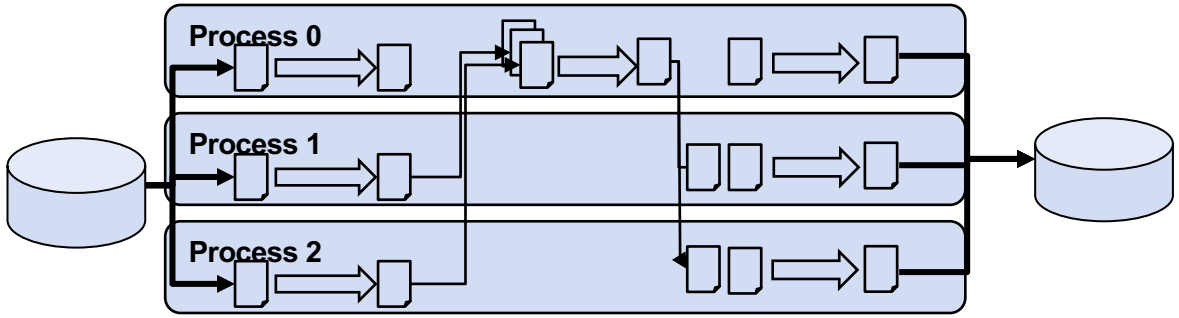


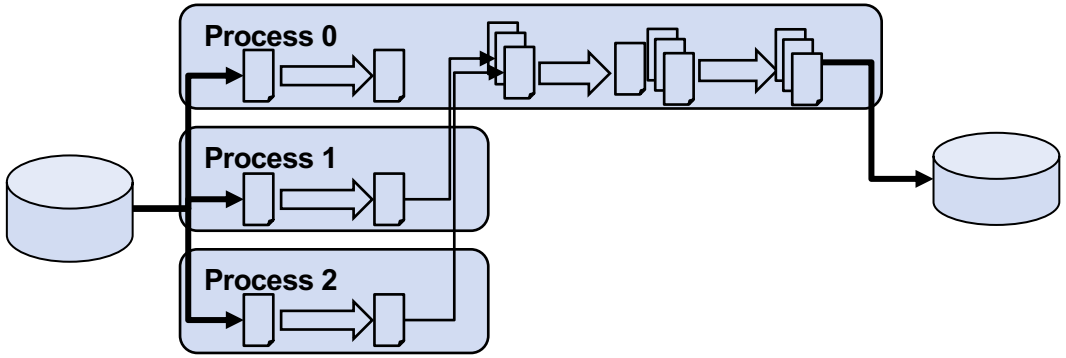
“Normalized” Word Count

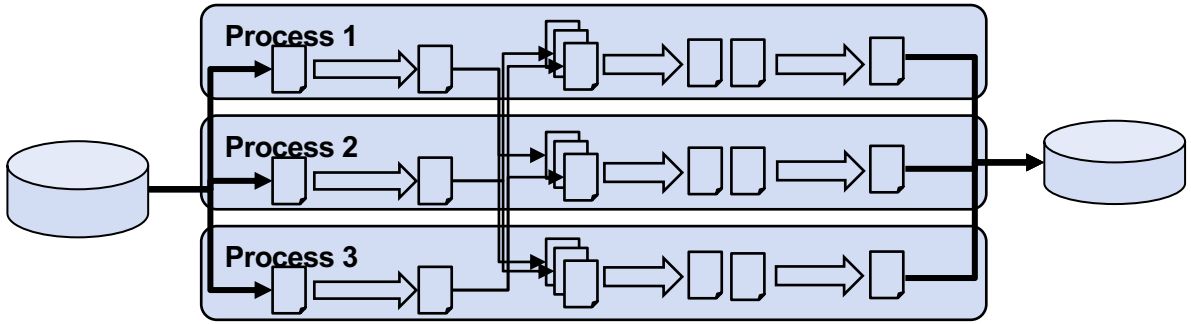
- Several files of documents
- Want a summary of what they contain
  - What are the normalized word counts for each document?



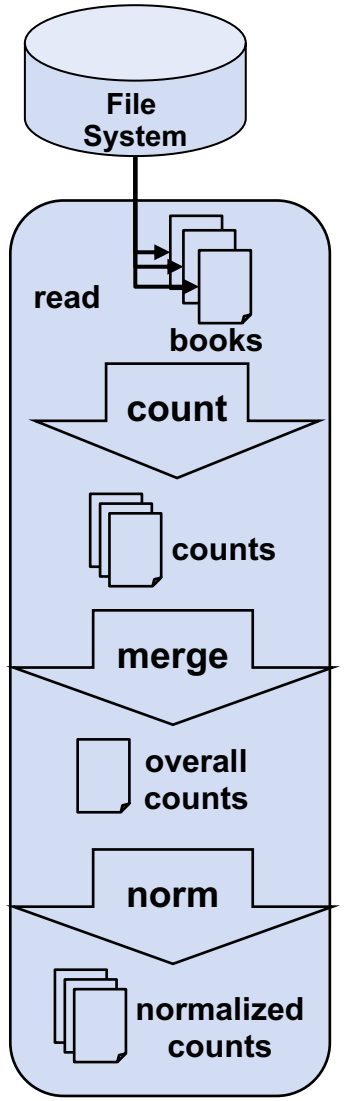
“Normalized” Word Count





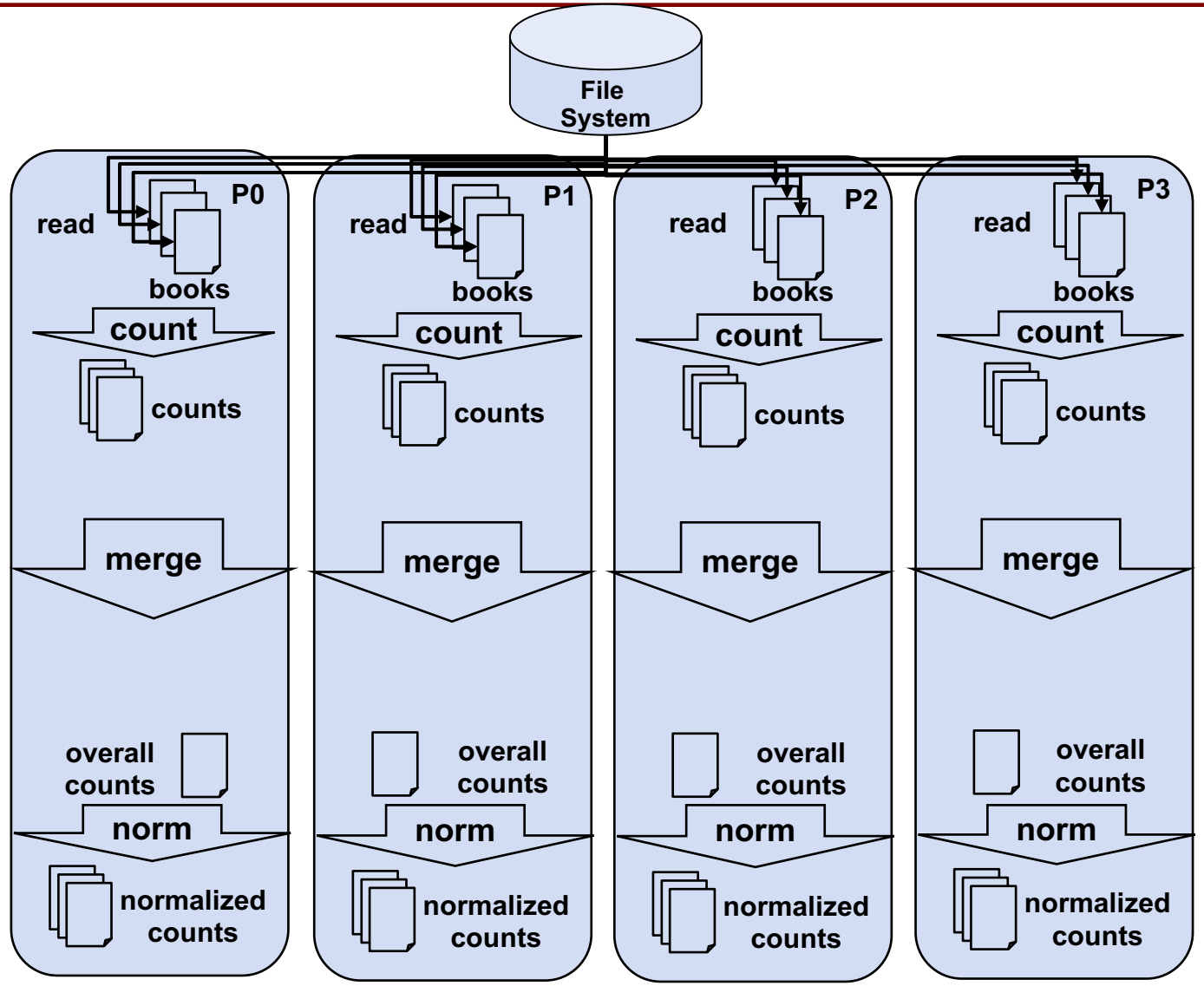


“Normalized” Word Count Serial




```
counts = countwords.(fnames)

overallcounts = merge(+,counts…)

for count in counts
        normcount = merge(/,count,overallcounts)
        # print top 5
end
```

“Normalized” Word Count- MPI

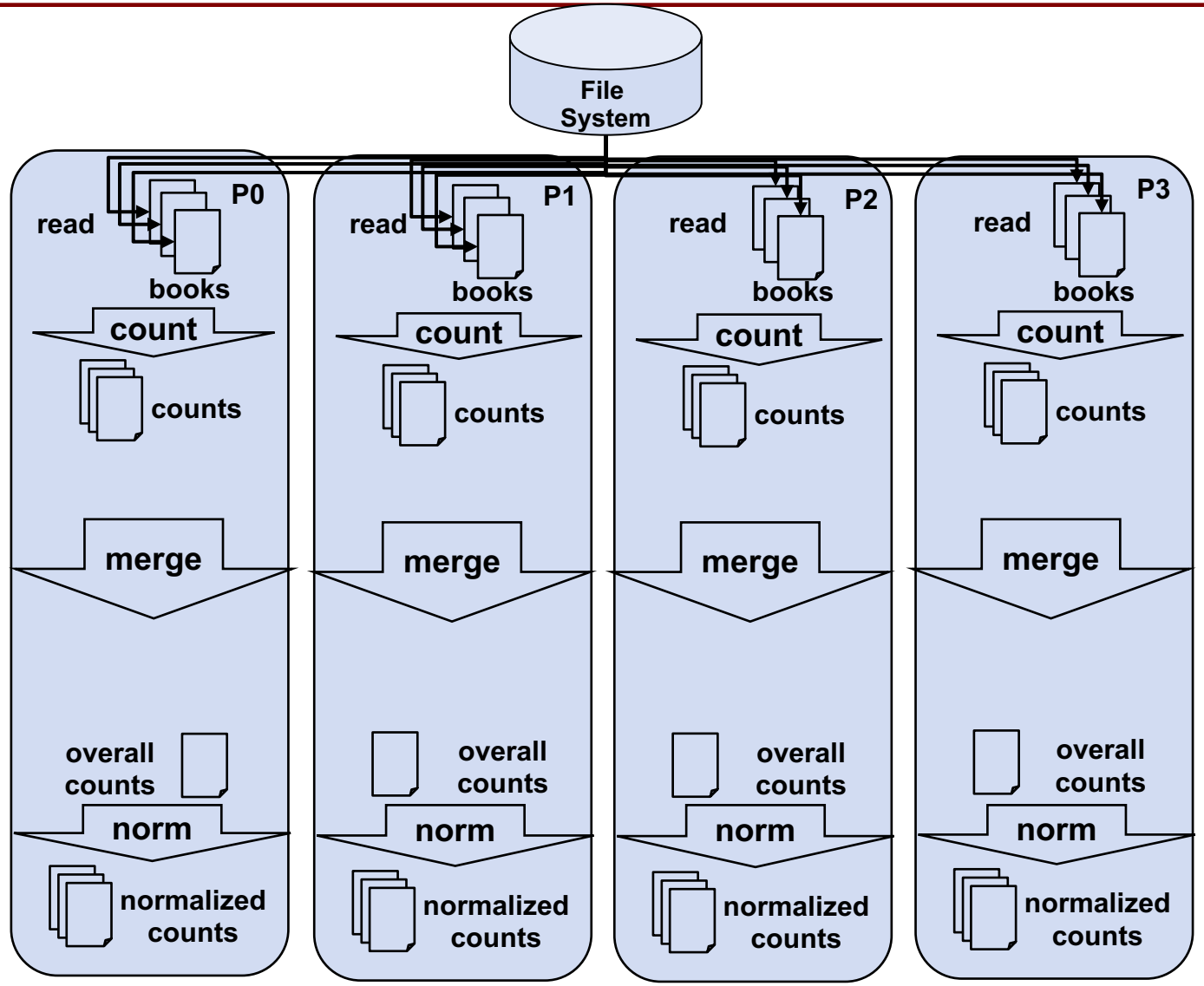



```
mycounts = countwords.(fnames)
```

```
overallcounts = merge(+,allcounts…)
```

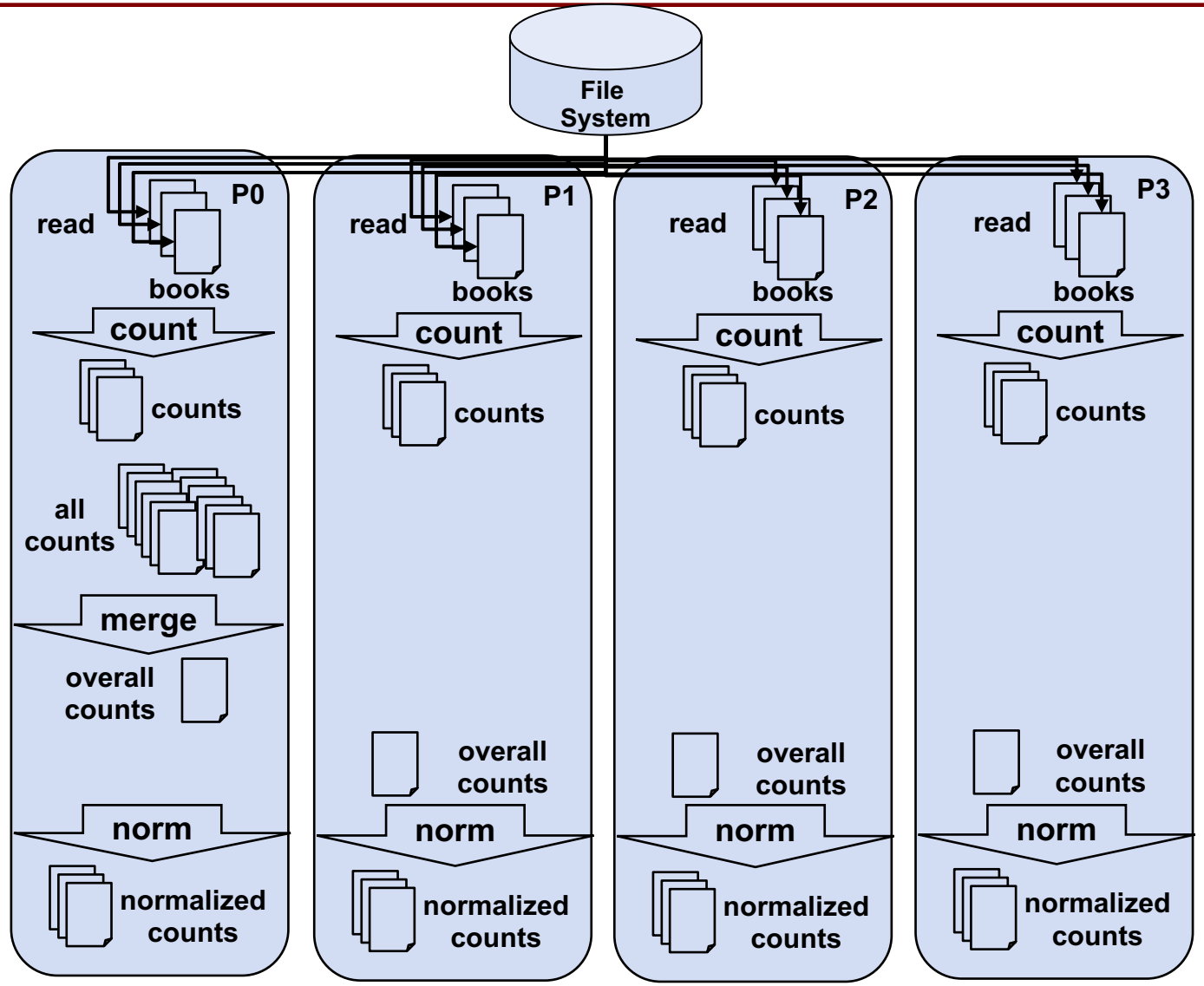
```
for count in mycounts
  normcount = merge(/,mycount,overallcounts)
  # print top 5
end
```

“Normalized” Word Count- MPI




```
myfnames = distributefiles(fnames)
mycounts = countwords.(myfnames)

overallcounts = merge(+,allcounts…)

for count in mycounts
  normcount = merge(/,mycount,overallcounts)
  # print top 5
end
```

“Normalized” Word Count- MPI

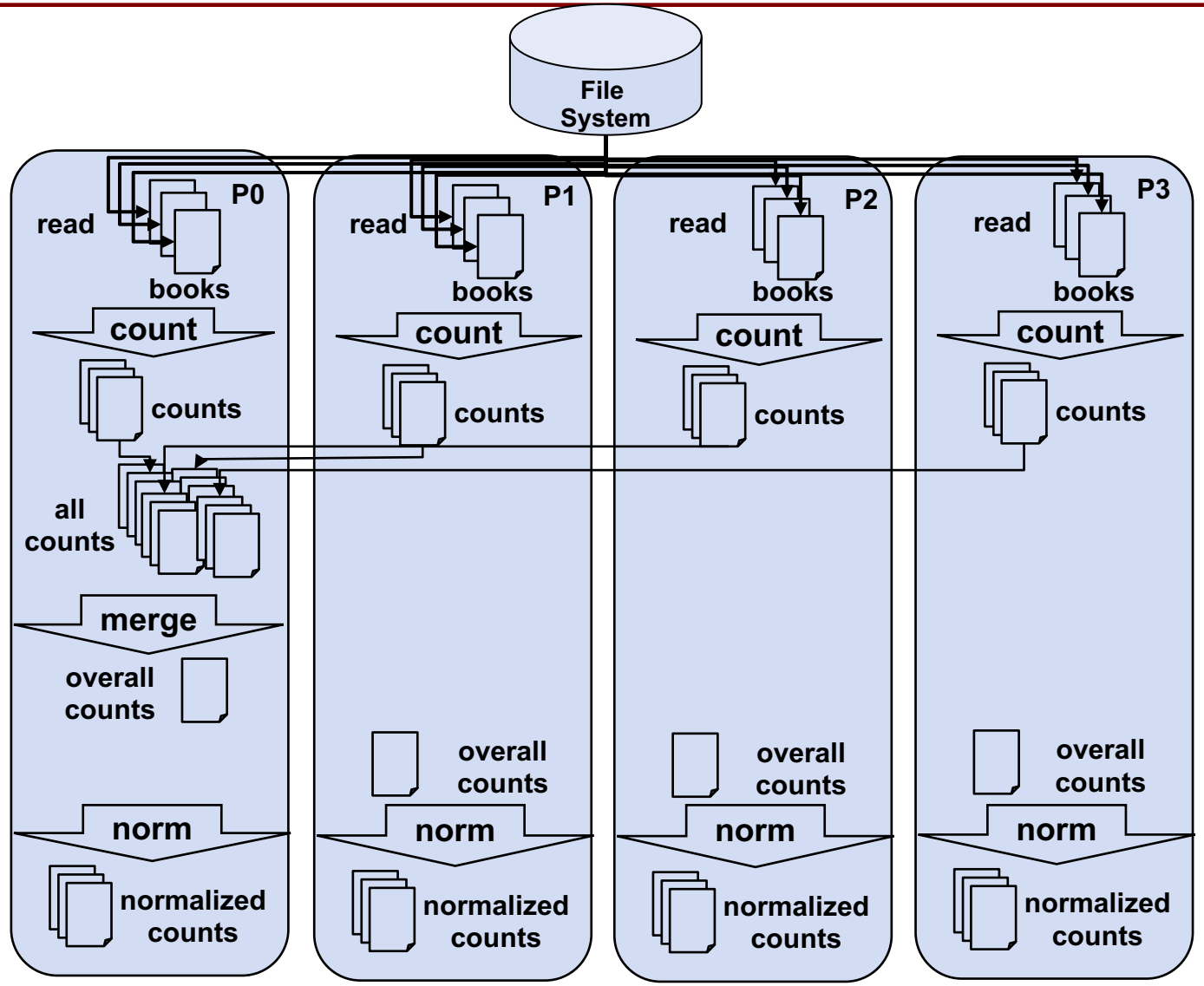



```
myfnames = distributefiles(fnames)

mycounts = countwords.(myfnames)
```

```
P0: overallcounts = merge(+,allcounts…)
```

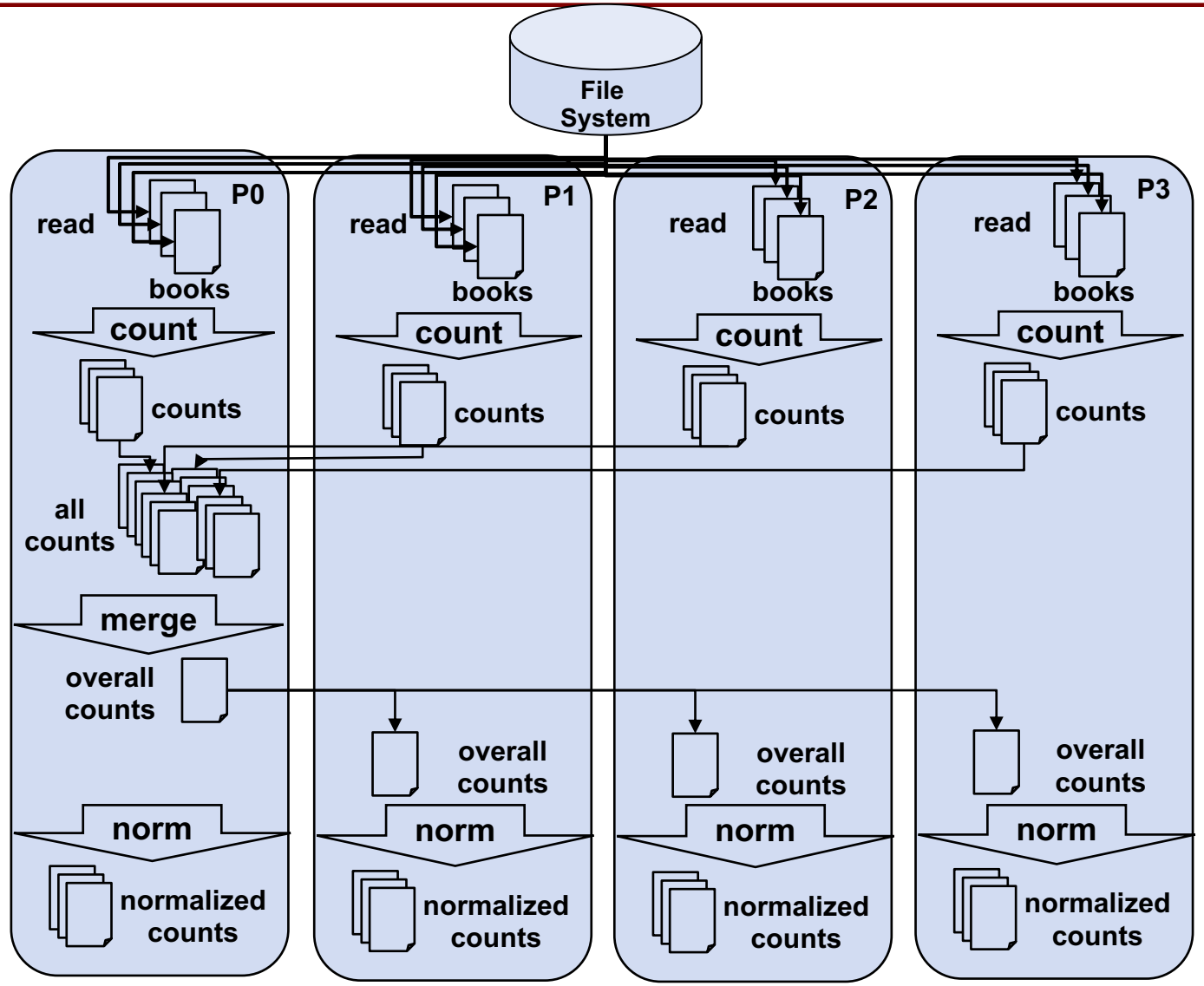
```
for count in mycounts
  normcount = merge(/,mycount,overallcounts)
  # print top 5
end
```

“Normalized” Word Count- MPI




```
myfnames = distributefiles(fnames)

mycounts = countwords.(myfnames)

P1,P2,P3: Send to P0
P0: Receive from P1,P2,P3

P0: overallcounts = merge(+,allcounts…)

for count in mycounts
  normcount = merge(/,mycount,overallcounts)
  # print top 5
end
```

“Normalized” Word Count- MPI




```
myfnames = distributefiles(fnames)

mycounts = countwords.(myfnames)

P1,P2,P3: Send to P0
P0: Receive from P1,P2,P3


P0: overallcounts = merge(+,allcounts…)

P0: Send to P1,P2,P3
P1,P2,P3: Receive from P0

for count in mycounts
  normcount = merge(/,mycount,overallcounts)
  # print top 5
end
```

MPI Terminology/Standard Commands

helloworld.jl

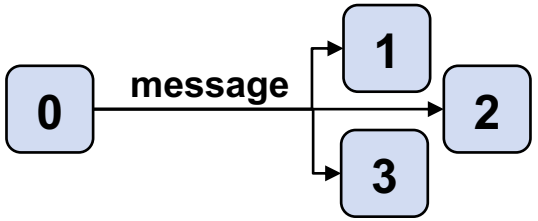
```julia
using MPI

...
# Initialize MPI environment
MPI.Init()

# Get MPI process rank id
rank = MPI.Comm_rank(MPI.COMM_WORLD)

# Get number of MPI processes in this communicator
nproc = MPI.Comm_size(MPI.COMM_WORLD)

# Print hello world message
print("Hello world, I am rank $(rank) of $(nproc) processors\n")
```

- **MPI: Message Passing Interface**
- **MPI.init(): Initializes the MPI environment**
- **MPI.COMM_WORLD: the MPI communicator- everything ranks need to communicate**
- **MPI.Comm_size: the number of MPI processes (N)**
- **MPI.Comm_rank: an integer 0:N-1; each MPI process has a different rank, you can think of it as a process ID**

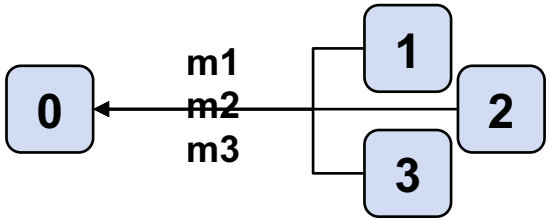
Some MPI Terminology/Standard Commands

- **Send/Receive: Point-to-point communication: rank i sends a message to rank j and rank j receives a message from rank i**
  - **Blocking/Unblocking: Blocked send/receives wait until the message has been sent or received before proceeding; unblocking send/receives and then continues**

- **Broadcast: one-to-all communication: rank i sends message to rank 0:i-1,i+1:N**

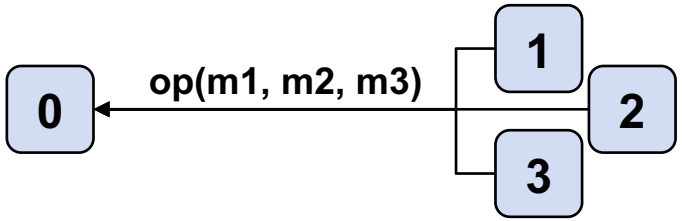


- **Gather: all-to-one communication: rank 0:i-1,i+1:N sends message to rank i**

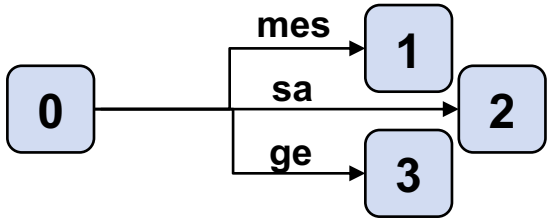


Some MPI Terminology/Standard Commands

- **Reduce: all-to-one communication: gather messages onto rank i and perform a reduce operation**



- **Scatter: one-to-all send one piece to each**



MPI.jl Basic Send and Recieve

```
MPI.Send(buf, dest::Integer, tag::Integer, comm::Comm)
MPI.Send(obj::T, dest::Integer, tag::Integer, comm::Comm) where T
MPI.send(obj, dest::Integer, tag::Integer, comm::Comm)
```

- obj/buf: The ”message”, what you are sending
- dest: Where you are sending the message to
- tag: Unique identifier for this message
- comm: The MPI communicator

```
MPI.Recv!(data, src::Integer, tag::Integer, comm::Comm)
MPI.Recv(::Type{T}, src::Integer, tag::Integer, comm::Comm)
MPI.recv(src::Integer, tag::Integer, comm::Comm)
```

- src: The message source, who is sending the message
- tag: The matching unique identifier from the send
- comm: The MPI communicator

See https://juliaparallel.github.io/MPI.jl/stable/pointtopoint/ for more functions and full descriptions

MPI.jl Basic Send and Recieve

```
MPI.Send(buf, dest::Integer, tag::Integer, comm::Comm)
MPI.Send(obj::T, dest::Integer, tag::Integer, comm::Comm) where T
MPI.send(obj, dest::Integer, tag::Integer, comm::Comm)
```

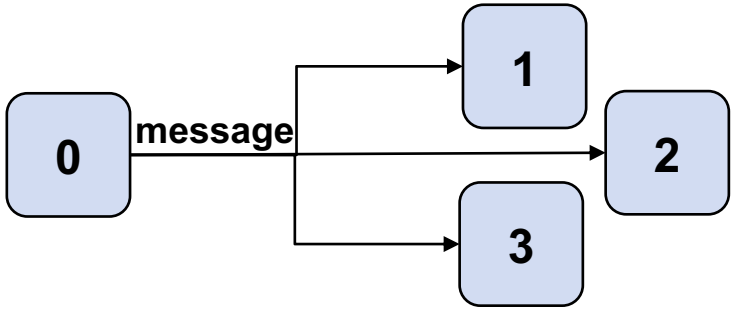
- obj/buf: The ”message”, what you are sending
- dest: Where you are sending the message to
- tag: Unique identifier for this message
- comm: The MPI communicator

```
MPI.Recv!(data, src::Integer, tag::Integer, comm::Comm)
MPI.Recv(::Type{T}, src::Integer, tag::Integer, comm::Comm)
MPI.recv(src::Integer, tag::Integer, comm::Comm)
```

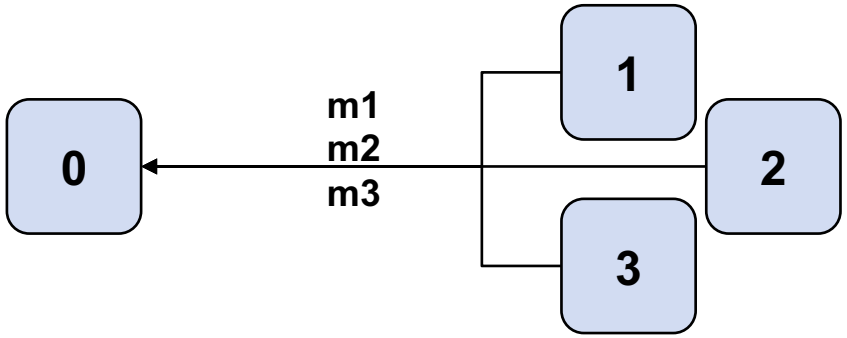
- src: The message source, who is sending the message
- tag: The matching unique identifier from the send
- comm: The MPI communicator

See https://juliaparallel.github.io/MPI.jl/stable/pointtopoint/ for more functions and full descriptions

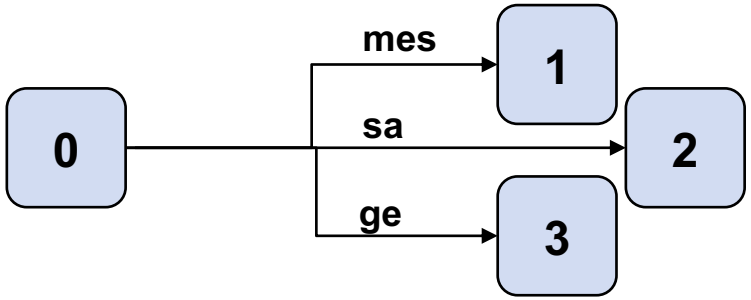
Collective Communication Calls



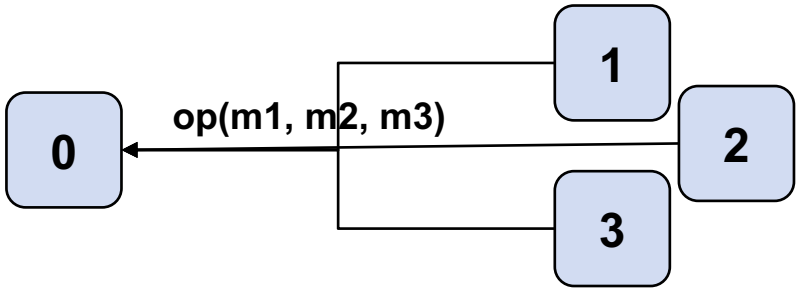
Broadcast



Gather



Scatter



Reduce

MPI.jl On Supercloud

- **On the Login Node, load the MPI and Julia modules**

```
module load mpi
module load julia
```

- **Tell MPI.jl to use the system MPI**

```
export JULIA_MPI_BINARY=system
```

- **Start Julia and add/build the MPI.jl package (press] to get to pkg mode)**