Parallel SCC and Centrality

Lecture 5

CSCI 4974/6971

15 Sep 2016

Today's Biz

1. Quick Review
2. Reminders
3. Parallel SCC
4. More Centrality
5. Even More MPI
6. More PageRank Tutorial

Today's Biz

1. **Quick Review**
2. Reminders
3. Parallel SCC
4. More Centrality
5. Even More MPI
6. More PageRank Tutorial

Quick Review

- Structure of the Web
  - Directed graph - SCCs and DAGs
  - *Bowtie* - big SCC, in set, out set, tendrils, tubes, disconnected components
- PageRank
  - Centrality measure - which pages hold highest influence
  - Random surfer - PageRank equivalent to relative probability a random surfer visits a given page

More MPI functions

  - `MPI_Allgather(sbuf, scount, MPI_TYPE, rbuf, rcount, MPI_TYPE, MPI_COMM_WORLD)`
  - `MPI_Alltoall(sbuf, scount, MPI_TYPE, rbuf, rcount, MPI_TYPE, MPI_COMM_WORLD)`

Today's Biz

1. Quick Review
2. **Reminders**
3. Parallel SCC
4. More Centrality
5. Even More MPI
6. More PageRank Tutorial

Reminders

- Assignment 1: Monday 19 Sept 16:00
- Assignment 2: Thursday 29 Sept 16:00 (posted soon)
- Project Proposal: Thursday 22 Sept 16:00
- Office hours: Tuesday & Wednesday 14:00-16:00 Lally 317
  - Or email me for other availability
- Class schedule:
  - Social net analysis methods
  - Bio net analysis methods
  - Random networks and usage
- **Today: Leave advisor info for CCI at end of class**

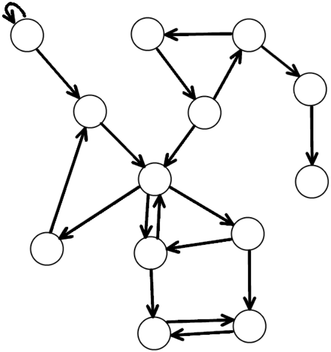
Today's Biz

1. Quick Review
2. Reminders
3. **Parallel SCC**
4. More Centrality
5. Even More MPI
6. More PageRank Tutorial

Parallel Strongly Connected Components Algorithms

Previous Algorithms

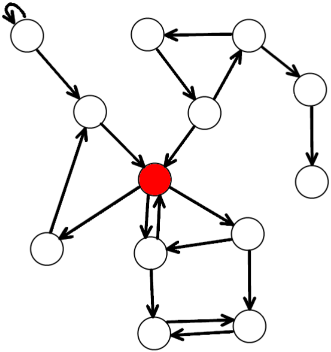
Forward-Backward (FW-BW)



Previous Algorithms

Forward-Backward (FW-BW)

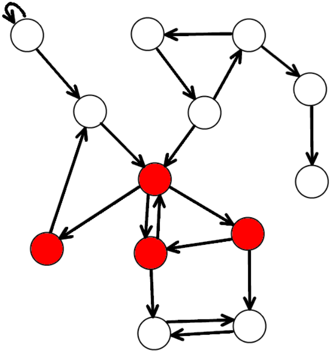
- Select pivot



Previous Algorithms

Forward-Backward (FW-BW)

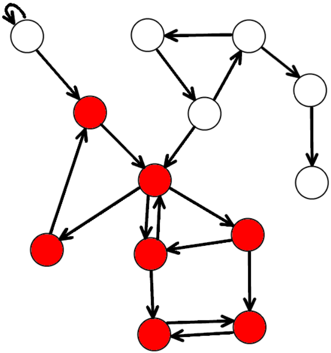
- Select pivot
- Find all vertices that can be reached from the pivot (**descendant** (D))



Previous Algorithms

Forward-Backward (FW-BW)

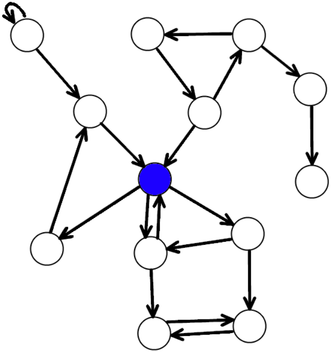
- Select pivot
- Find all vertices that can be reached from the pivot (**descendant** (D))



Previous Algorithms

Forward-Backward (FW-BW)

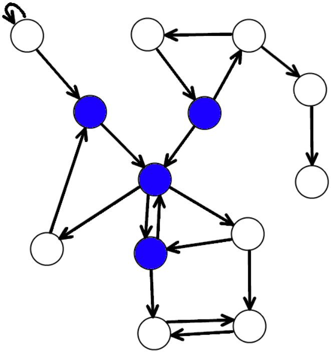
- Select pivot
- Find all vertices that can be reached from the pivot (**descendant** (D))
- Find all vertices that can reach the pivot (**predecessor** (P))



Previous Algorithms

Forward-Backward (FW-BW)

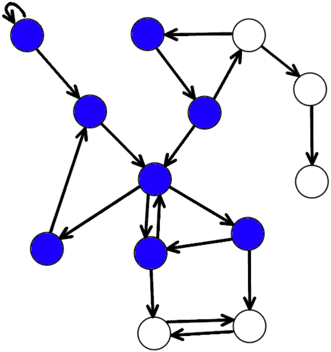
- Select pivot
- Find all vertices that can be reached from the pivot (**descendant** (D))
- Find all vertices that can reach the pivot (**predecessor** (P))



Previous Algorithms

Forward-Backward (FW-BW)

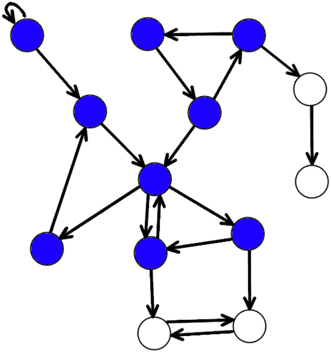
- Select pivot
- Find all vertices that can be reached from the pivot (**descendant** (D))
- Find all vertices that can reach the pivot (**predecessor** (P))



Previous Algorithms

Forward-Backward (FW-BW)

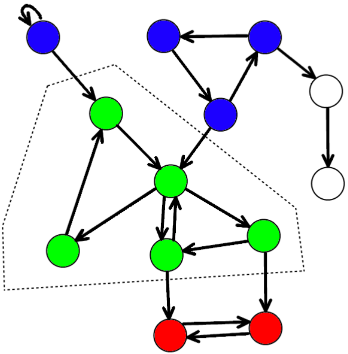
- Select pivot
- Find all vertices that can be reached from the pivot (**descendant** (D))
- Find all vertices that can reach the pivot (**predecessor** (P))



Previous Algorithms

Forward-Backward (FW-BW)

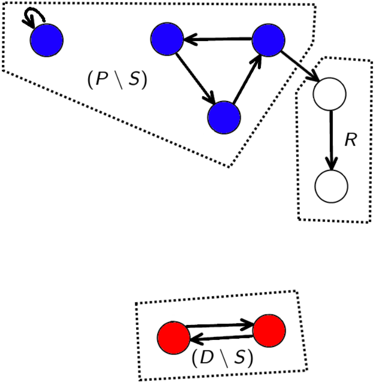
- Select pivot
- Find all vertices that can be reached from the pivot (**descendant** (D))
- Find all vertices that can reach the pivot (**predecessor** (P))
- Intersection of those two sets is an SCC ($S = P \cap D$)



Previous Algorithms

Forward-Backward (FW-BW)

- Select pivot
- Find all vertices that can be reached from the pivot (**descendant** (D))
- Find all vertices that can reach the pivot (**predecessor** (P))
- Intersection of those two sets is an SCC ($S = P \cap D$)
- Now have three distinct sets leftover ($D \setminus S$), ($P \setminus S$), and **remainder** (R)



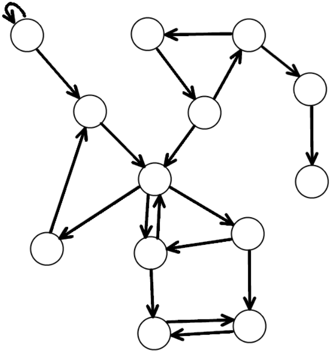
Forward-Backward (FW-BW) Algorithm

```
**procedure** FW-BW($V$)
    **if** $V = \varnothing$ **then**
        **return** $\varnothing$
    Select a *pivot* $u \in V$
    $D \leftarrow \mathrm{BFS}(G(V, E(V)), u)$
    $P \leftarrow \mathrm{BFS}(G(V, E'(V)), u)$
    $R \leftarrow (V \setminus (P \cup D)$
    $S \leftarrow (P \cap D)$
    **new task do** FW-BW($D \setminus S$)
    **new task do** FW-BW($P \setminus S$)
    **new task do** FW-BW($R$)
```

Previous Algorithms

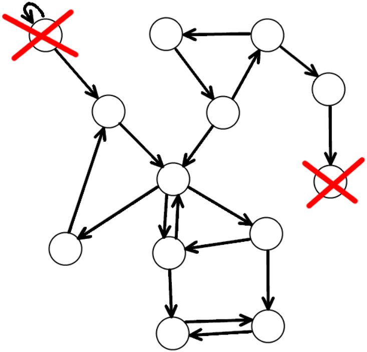
Trimming

- Used to find trivial SCCs



Previous Algorithms
Trimming

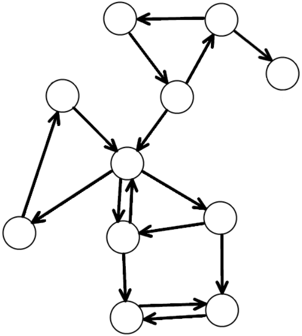
- Used to find trivial SCCs
- Detect and prune all vertices that have an in/out degree of 0 or an in/out degree of 1 with a self loop (simple trimming)



Previous Algorithms

Trimming

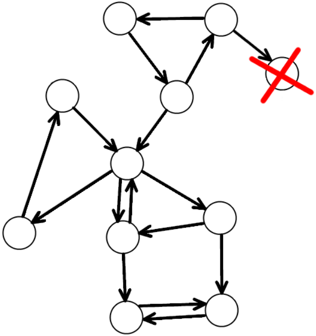
- Used to find trivial SCCs
- Detect and prune all vertices that have an in/out degree of 0 or an in/out degree of 1 with a self loop (simple trimming)



Previous Algorithms

Trimming

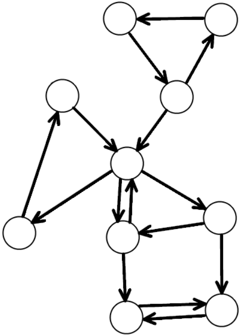
- Used to find trivial SCCs
- Detect and prune all vertices that have an in/out degree of 0 or an in/out degree of 1 with a self loop (simple trimming)
- Repeat iteratively until no more vertices can be removed (complete trimming)



Previous Algorithms

Trimming

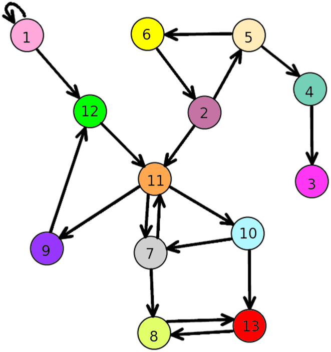
- Used to find trivial SCCs
- Detect and prune all vertices that have an in/out degree of 0 or an in/out degree of 1 with a self loop (simple trimming)
- Repeat iteratively until no more vertices can be removed (complete trimming)



Previous Algorithms

Coloring

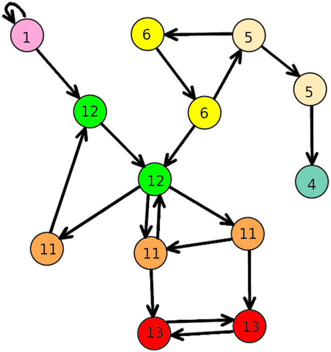
- Consider vertex identifiers as *colors*



Previous Algorithms

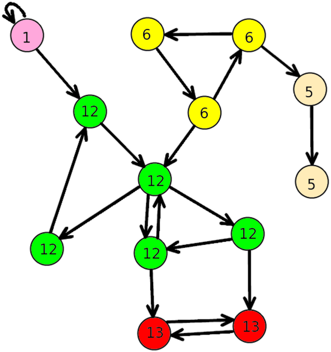
Coloring

- Consider vertex identifiers as *colors*
- Highest colors are propagated **forward** through the network to create sets



Previous Algorithms
Coloring

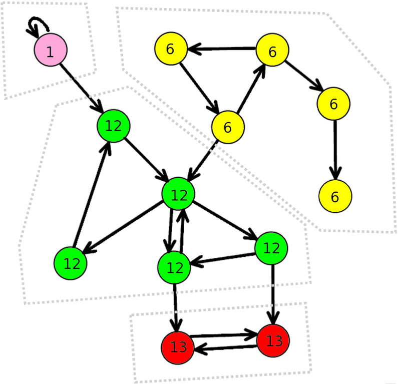
- Consider vertex identifiers as *colors*
- Highest colors are propagated **forward** through the network to create sets



Previous Algorithms

Coloring

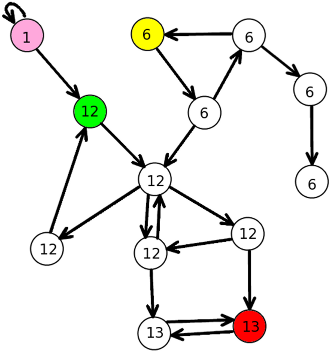
- Consider vertex identifiers as *colors*
- Highest colors are propagated **forward** through the network to create sets



Previous Algorithms

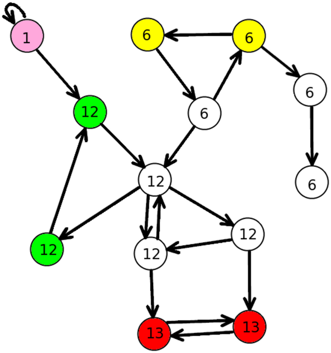
Coloring

- Consider vertex identifiers as *colors*
- Highest colors are propagated **forward** through the network to create sets
- Consider the original vertex of each color to be the *root* of a new SCC



Previous Algorithms
Coloring

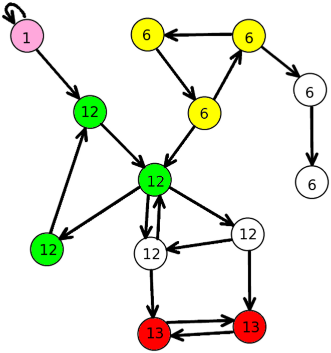
- Consider vertex identifiers as *colors*
- Highest colors are propagated **forward** through the network to create sets
- Consider the original vertex of each color to be the *root* of a new SCC
- Each SCC is all vertices (of the same color as the root) reachable **backward** from each root.



Previous Algorithms

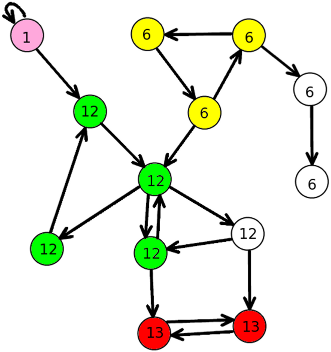
Coloring

- Consider vertex identifiers as *colors*
- Highest colors are propagated **forward** through the network to create sets
- Consider the original vertex of each color to be the *root* of a new SCC
- Each SCC is all vertices (of the same color as the root) reachable **backward** from each root.



Previous Algorithms
Coloring

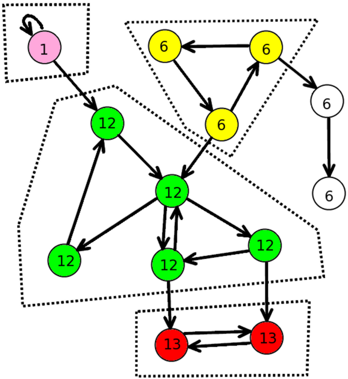
- Consider vertex identifiers as *colors*
- Highest colors are propagated **forward** through the network to create sets
- Consider the original vertex of each color to be the *root* of a new SCC
- Each SCC is all vertices (of the same color as the root) reachable **backward** from each root.



Previous Algorithms

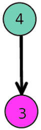
Coloring

- Consider vertex identifiers as *colors*
- Highest colors are propagated **forward** through the network to create sets
- Consider the original vertex of each color to be the *root* of a new SCC
- Each SCC is all vertices (of the same color as the root) reachable **backward** from each root.



Previous Algorithms
Coloring

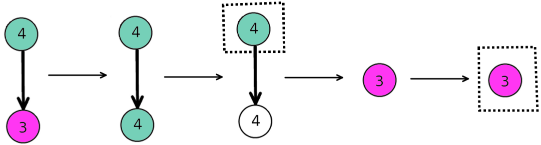
- Consider vertex identifiers as *colors*
- Highest colors are propagated **forward** through the network to create sets
- Consider the original vertex of each color to be the *root* of a new SCC
- Each SCC is all vertices (of the same color as the root) reachable **backward** from each root.
- Remove found SCCs, reset colors, and repeat until no vertices remain



Previous Algorithms

Coloring

- Consider vertex identifiers as *colors*
- Highest colors are propagated **forward** through the network to create sets
- Consider the original vertex of each color to be the *root* of a new SCC
- Each SCC is all vertices (of the same color as the root) reachable **backward** from each root.
- Remove found SCCs, reset colors, and repeat until no vertices remain



Coloring Algorithms

```
procedure ColorSCC($G(V, E)$)
    while $G \neq \varnothing$ do
        for all $u \in V$ do $Colors(u) \leftarrow u$
        while at least one vertex has changed colors do
            for all $u \in V$ in parallel do
                for all $\langle u, v \rangle \in E$ do
                    if $Colors(u) > Colors(v)$ then
                        $Colors(v) \leftarrow Colors(u)$
        for all unique $c \in Colors$ in parallel do
            $V_c \leftarrow \{u \in V : Colors(u) = c\}$
            $SCCV_c \leftarrow BFS(G(V_c, E'(V_c)), u)$
            $V \leftarrow (V \setminus SCCV_c)$
```

Today's Biz

1. Quick Review
2. Reminders
3. Parallel SCC
4. **More Centrality**
5. Even More MPI
6. More PageRank Tutorial

Network Centrality

Slides from Ahmed Louri, University of Arizona

Network Centrality

Based on materials by Lada Adamic, UMichigan

Network Centrality

Which nodes are most ‘central’?

Definition of ‘central’ varies by context/purpose.

Local measure:
degree

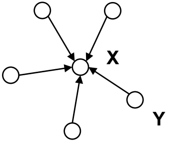
Relative to rest of network:
closeness, betweenness,
eigenvector (Bonacich power centrality)

How evenly is centrality distributed among nodes?
centralization…

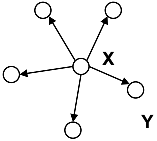
Applications:
Friedkin: Interpersonal Influence in Groups
Baker: The Social Organization of Conspiracy

Centrality: Who’s Important Based On Their Network Position

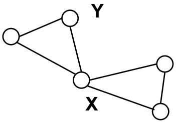
In each of the following networks, X has higher centrality than Y according to a particular measure



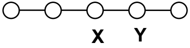
indegree



outdegree



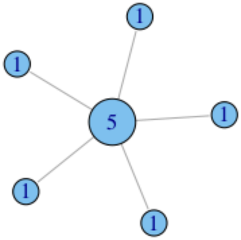
betweenness



closeness

Degree Centrality (Undirected)

He or she who has many friends is most important.

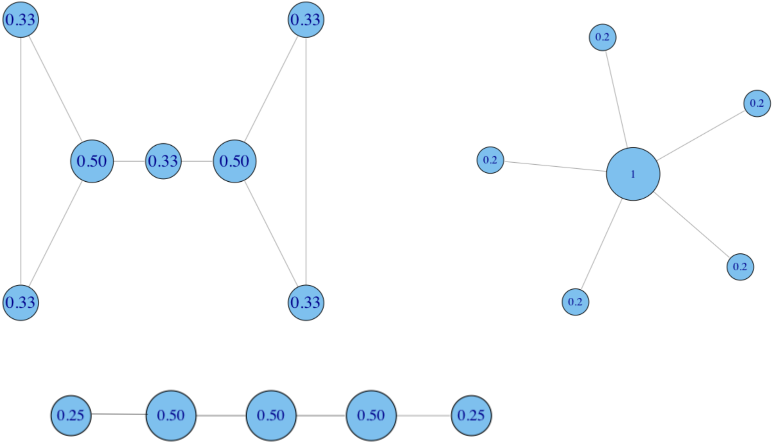


When is the number of connections the best centrality measure?

- ○ people who will do favors for you
- ○ people you can talk to / have coffee with

Degree: Normalized Degree Centrality

divide by the max. possible, i.e. (N-1)



Centralization: How Equal Are The Nodes?

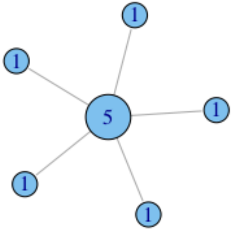
How much variation is there in the centrality scores among the nodes?

Freeman's general formula for centralization (can use other metrics, e.g. gini coefficient or standard deviation):

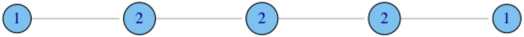
maximum value in the network

$$C_D = \frac{\sum_{i=1}^{g}\left[\boxed{C_D(n^*)} - C_D(i)\right]}{[(N-1)(N-2)]}$$

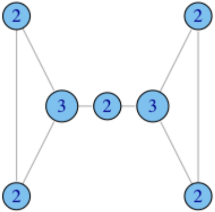
Degree Centralization Examples



$C_D = 1.0$



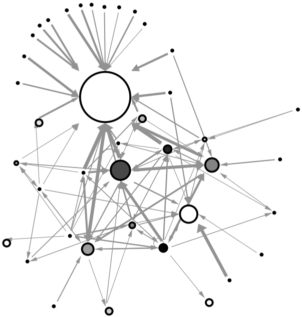
$C_D = 0.167$

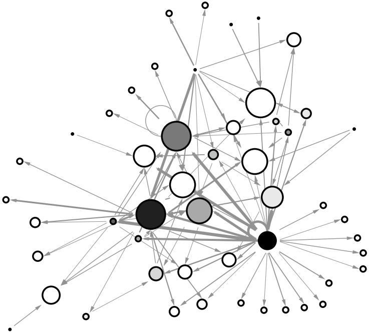


$C_D = 0.167$

Degree Centralization Examples

example financial trading networks



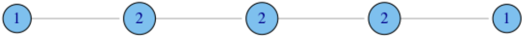


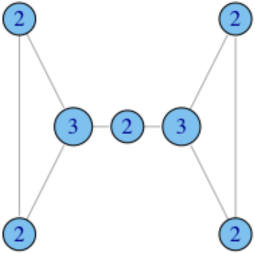
high centralization: one node trading with many others

low centralization: trades are more evenly distributed

When Degree Isn't Everything

In what ways does degree fail to capture centrality in the following graphs?



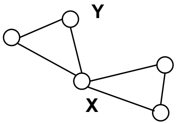


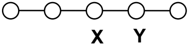
In What Contexts May Degree Be Insufficient To Describe Centrality?

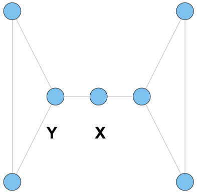
- ability to broker between groups
- likelihood that information originating anywhere in the network reaches you…

Betweenness: Another Centrality Measure

- Intuition: how many pairs of individuals would have to go through you in order to reach one another in the minimum number of hops?
- Who has higher betweenness, X or Y?







Betweenness Centrality: Definition

$$C_B(i) = \sum_{j<k} g_{jk}(i) / g_{jk}$$

Where g_{jk} = the number of geodesics connecting *jk*, and
$g_{jk}(i)$= the number of geodesics that actor *i* is on.

Usually normalized by:

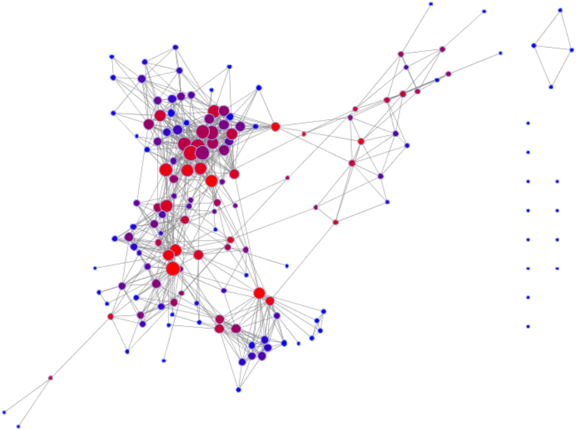
$$C'_B(i) = C_B(i\ ) / \boxed{[(n-1)(n-2)/2]}$$

number of pairs of vertices
excluding the vertex itself

adapted from a slide by James Moody

Example

Example facebook network: nodes are sized by degree, and colored by betweenness.

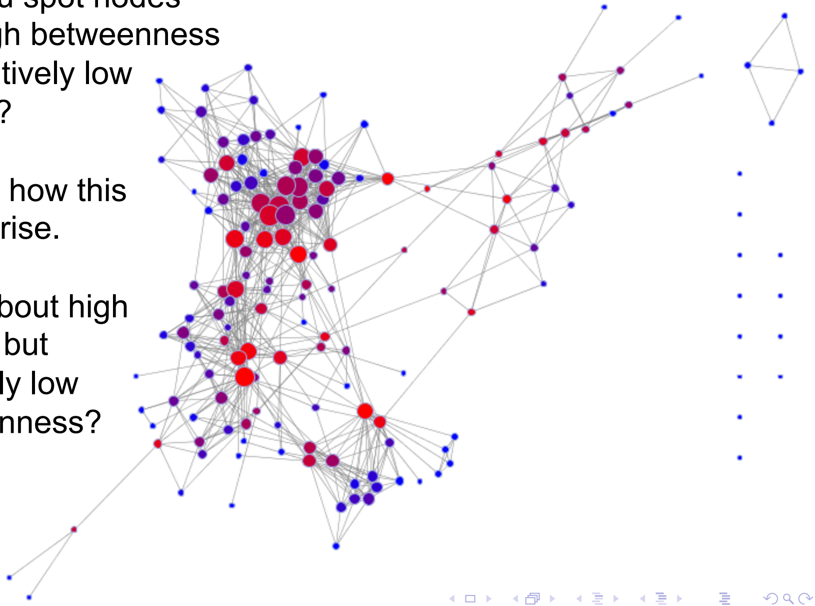


Betweenness Example (Continued)

Can you spot nodes with high betweenness but relatively low degree?

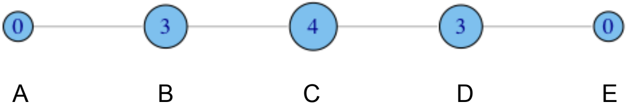
Explain how this might arise.

What about high degree but relatively low betweenness?



Betweenness On Toy Networks

- non-normalized version:

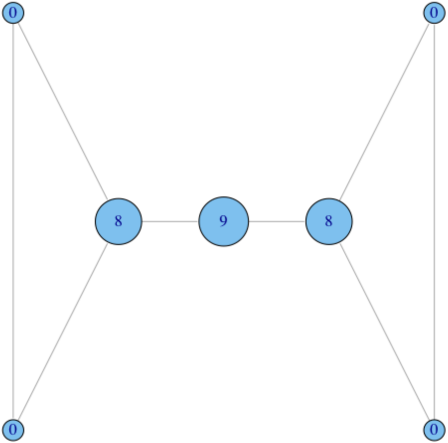


- A lies between no two other vertices
- B lies between A and 3 other vertices: C, D, and E
- C lies between 4 pairs of vertices (A,D),(A,E),(B,D),(B,E)

- note that there are no alternate paths for these pairs to take, so C gets full credit

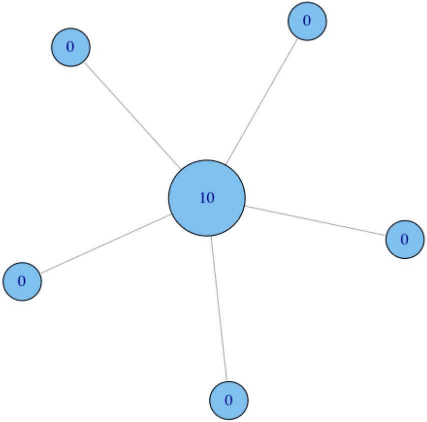
Betweenness On Toy Networks

- non-normalized version:



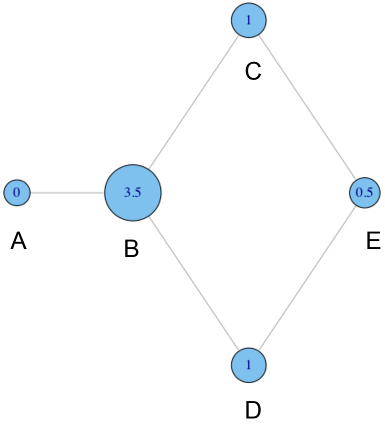
Betweenness On Toy Networks

- non-normalized version:



Betweenness On Toy Networks

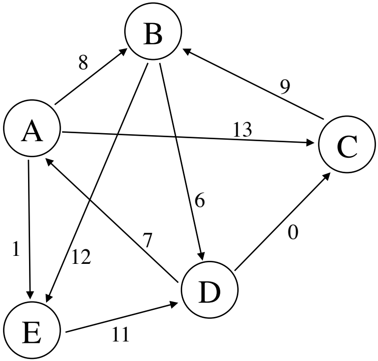
- non-normalized version:



- why do C and D each have betweenness 1?
- They are both on shortest paths for pairs (A,E), and (B,E), and so must share credit:
  - ½+½ = 1
- Can you figure out why B has betweenness 3.5 while E has betweenness 0.5?

All-pairs shortest paths...

“Floyd-Warshall algorithm”



Matrix representation

TO

FROM

$$D^0 \quad \begin{array}{c|ccccc|} & A & B & C & D & E \\ A & 0 & 8 & 13 & - & 1 \\ B & - & 0 & - & 6 & 12 \\ C & - & 9 & 0 & - & - \\ D & 7 & - & 0 & 0 & - \\ E & - & - & - & 11 & 0 \end{array}$$

All-pairs shortest paths...

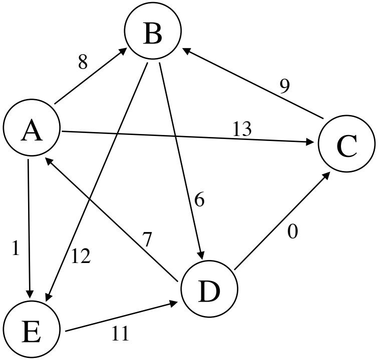
$D^0 = (d^0_{ij})$

$$\begin{array}{c|ccccc|}
A & 0 & 8 & 13 & - & 1 \\
B & - & 0 & - & 6 & 12 \\
C & - & 9 & 0 & - & - \\
D & 7 & - & 0 & 0 & - \\
E & - & - & - & 11 & 0
\end{array}$$

d^k_{ij} = shortest distance from i to j through {1, …, k}

$D^1 = (d^1_{ij})$

$$\begin{array}{c|ccccc|}
A & 0 & 8 & 13 & - & 1 \\
B & - & 0 & - & 6 & 12 \\
C & - & 9 & 0 & - & - \\
D & 7 & \boxed{15} & 0 & 0 & \boxed{8} \\
E & - & - & - & 11 & 0
\end{array}$$



All-pairs shortest paths...

$D^2 = (d_{ij}^2)$

$$\begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} \begin{pmatrix} 0 & 8 & 13 & \boxed{14} & 1 \\ \text{-} & 0 & \text{-} & 6 & 12 \\ \text{-} & 9 & 0 & \boxed{15} & \boxed{21} \\ 7 & 15 & 0 & 0 & 8 \\ \text{-} & \text{-} & \text{-} & 11 & 0 \end{pmatrix}$$

$D^3 = (d_{ij}^3)$

$$\begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} \begin{pmatrix} 0 & 8 & 13 & 14 & 1 \\ \text{-} & 0 & \text{-} & 6 & 12 \\ \text{-} & 9 & 0 & 15 & 21 \\ 7 & \boxed{9} & 0 & 0 & 8 \\ \text{-} & \text{-} & \text{-} & 11 & 0 \end{pmatrix}$$

$D^4 = (d_{ij}^4)$

$$\begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} \begin{pmatrix} 0 & 8 & 13 & 14 & 1 \\ \boxed{13} & 0 & \boxed{6} & 6 & 12 \\ \boxed{22} & 9 & 0 & 15 & 21 \\ 7 & 9 & 0 & 0 & 8 \\ \boxed{18} & \boxed{20} & \boxed{11} & 11 & 0 \end{pmatrix}$$

$D^5 = (d_{ij}^5)$

$$\begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} \begin{pmatrix} 0 & 8 & \boxed{12} & \boxed{12} & 1 \\ 13 & 0 & 6 & 6 & 12 \\ 22 & 9 & 0 & 15 & 21 \\ 7 & 9 & 0 & 0 & 8 \\ 18 & 20 & 11 & 11 & 0 \end{pmatrix}$$

to store the path, another matrix can track the last intermediate vertex

Floyd-Warshall Pseudocode

Input: $D^0 = (d^0_{ij})$ (the initial edge-cost matrix)

Output: $D^n = (d^n_{ij})$ (the final path-cost matrix)

```
for k = 1 to n                // intermediate vertices considered
  for i = 1 to n              // the “from” vertex
    for j = 1 to n            // the “to” vertex
      $d^k_{ij} = \min\{\ d^{k-1}_{ij},\ d^{k-1}_{ik} + d^{k-1}_{kj}\}$
```

best, including vertex k

best, ignoring vertex k

Closeness: Another Centrality Measure

- What if it’s not so important to have many direct friends?
- Or be “between” others
- But one still wants to be in the “middle” of things, not too far from the center

Closeness Centrality: Definition

Closeness is based on the length of the average shortest path between a vertex and all vertices in the graph

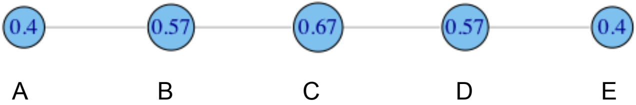
Closeness Centrality:

$$C_c(i) = \left[\sum_{j=1}^{N} d(i,j) \right]^{-1}$$

Normalized Closeness Centrality

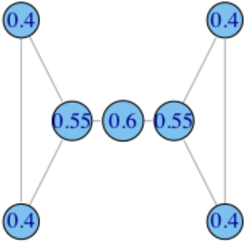
$$C_C^{'}(i) = (C_C(i))/(N-1)$$

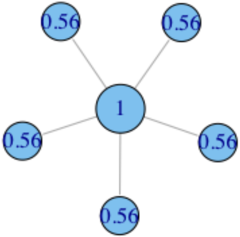
Closeness Centrality: Toy Example



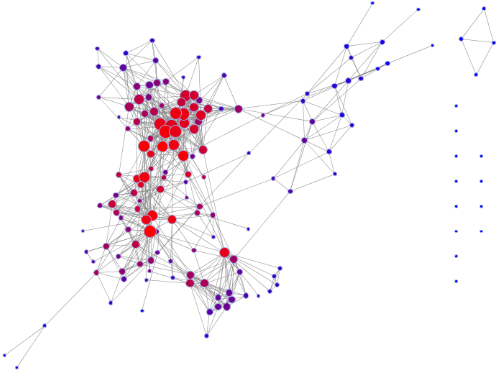
$$C_c^{'}(A) = \left[\frac{\sum_{j=1}^{N} d(A,j)}{N-1}\right]^{-1} = \left[\frac{1+2+3+4}{4}\right]^{-1} = \left[\frac{10}{4}\right]^{-1} = 0.4$$

Closeness Centrality: More Toy Examples





How Closely Do Degree And Betweenness Correspond To Closeness?



- **degree** (number of connections) denoted by size
- **closeness** (length of shortest path to all others) denoted by color

Centrality: Check Your Understanding

- generally different centrality metrics will be positively correlated
- when they are not, there is likely something interesting about the network
- suggest possible topologies and node positions to fit each square

	Low Degree	Low Closeness	Low Betweenness
High Degree			
High Closeness			
High Betweenness			

adapted from a slide by James Moody

Centrality: Check Your Understanding

- generally different centrality metrics will be positively correlated
- when they are not, there is likely something interesting about the network
- suggest possible topologies and node positions to fit each square

	Low Degree	Low Closeness	Low Betweenness
High Degree		Embedded in cluster that is far from the rest of the network	Ego's connections are redundant - communication bypasses him/her
High Closeness	Key player tied to important/active players		Probably multiple paths in the network, ego is near many people, but so are many others
High Betweenness	Ego's few ties are crucial for network flow	Very rare cell. Would mean that ego monopolizes the ties from a small number of people to many others.	

adapted from a slide by James Moody

Extending Betweenness Centrality To Directed Networks

- We now consider the fraction of all directed paths between any two vertices that pass through a node

paths between j and k that pass through i

betweenness of vertex i

$$C_B(i) = \sum_{j,k} g_{jk}(i) / g_{jk}$$

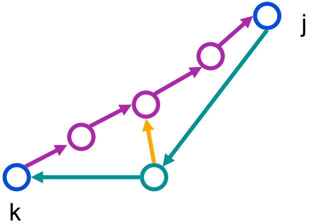
all paths between j and k

- Only modification: when normalizing, we have (N-1)*(N-2) instead of (N-1)*(N-2)/2, because we have twice as many ordered pairs as unordered pairs

$$C'_B(i) = C_B(i) / [(N-1)(N-2)]$$

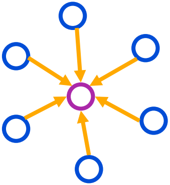
Directed Geodesics

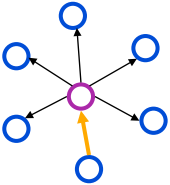
- A node does not necessarily lie on a geodesic from *j* to *k* if it lies on a geodesic from *k* to *j*



Extensions Of Undirected Degree Centrality - Prestige

- degree centrality
  - indegree centrality
    - a paper that is cited by many others has high prestige
    - a person nominated by many others for a reward has high prestige



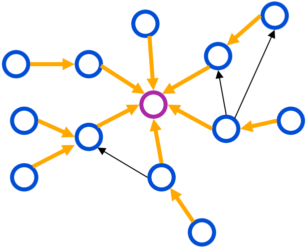


Extensions Of Undirected Closeness Centrality

- closeness centrality usually implies
  - all paths should lead to you

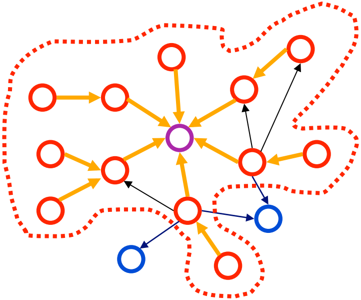
    and unusually not:
  - paths should lead from you to everywhere else

- usually consider only vertices from which the node *i* in question can be reached



Influence Range

- The influence range of *i* is the set of vertices who are reachable from the node *i*



Wrap Up

Centrality

- many measures: degree, betweenness, closeness, ...
- may be unevenly distributed
  - measure via centralization
- extensions to directed networks:
  - prestige
    - influence
  - PageRank

Today's Biz

1. Quick Review
2. Reminders
3. Parallel SCC
4. More Centrality
5. **Even More MPI**
6. More PageRank Tutorial

Even More MPI – Alltoallv

Slides from Lori Pollock, University of Delaware

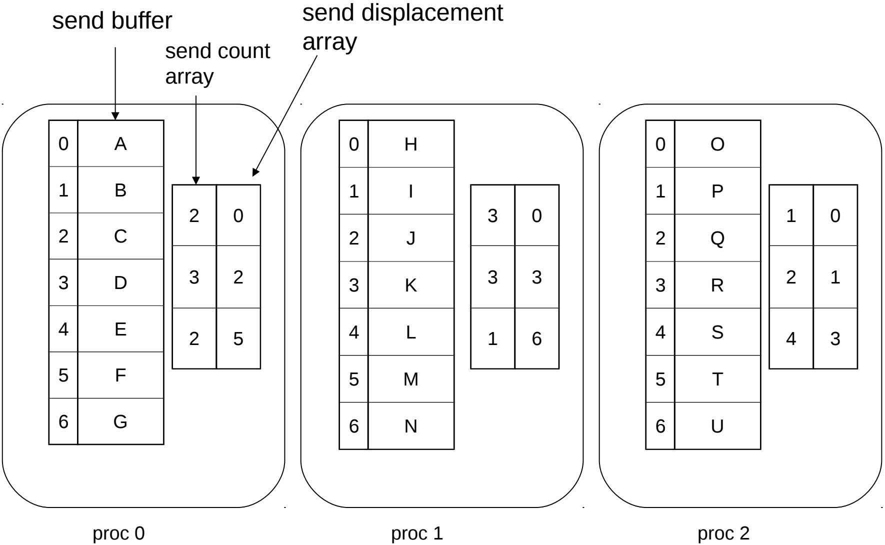
MPI_AlltoAllv Function Outline

```
int MPI_Alltoallv ( void *sendbuf, int *sendcnts, int *sdispls,
MPI_Datatype sendtype,
void *recvbuf, int *recvcnts, int *rdispls,
MPI_Datatype recvtype,
MPI_Comm comm )
```

Input Parameters

sendbuf starting address of send buffer (choice)
sendcounts integer array equal to the group size specifying the number of elements to send to each processor
sdispls integer array (of length group size). Entry j specifies the displacement (relative to sendbuf from which to take the outgoing data destined for process j
recvcounts integer array equal to the group size specifying the maximum number of elements that can be received from each processor
rdispls integer array (of length group size). Entry i specifies the displacement (relative to recvbuf at which to place the incoming data from process i

Each node in parallel community has



send buffer

send count array

send displacement array

proc 0

0	A
1	B
2	C
3	D
4	E
5	F
6	G

2	0
3	2
2	5

proc 1

0	H
1	I
2	J
3	K
4	L
5	M
6	N

3	0
3	3
1	6

proc 2

0	O
1	P
2	Q
3	R
4	S
5	T
6	U

1	0
2	1
4	3

Example of Send for Proc 0

0	A
1	B
2	C
3	D
4	E
5	F
6	G

index

Proc 0 send buffer

0	2
1	3
2	2

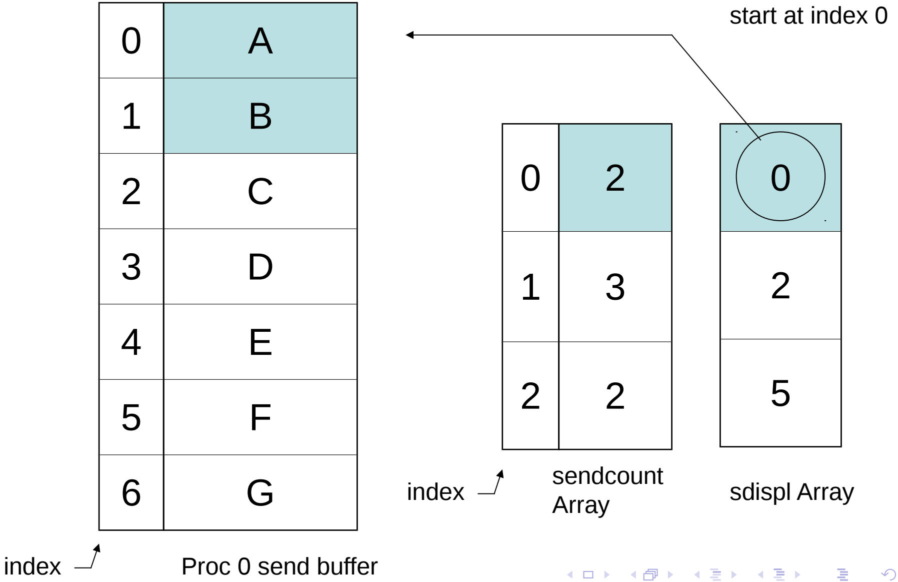
index

sendcount Array

0
2
5

sdispl Array

Example of Send for Proc 0



start at index 0

0	A
1	B
2	C
3	D
4	E
5	F
6	G

index

Proc 0 send buffer

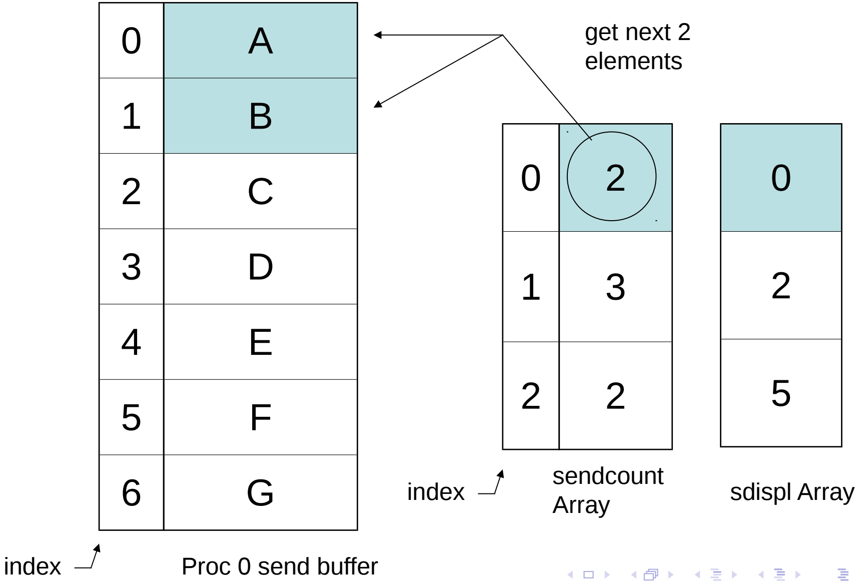
0	2	0
1	3	2
2	2	5

index

sendcount Array

sdispl Array

Example of Send for Proc 0



0	A
1	B
2	C
3	D
4	E
5	F
6	G

index

Proc 0 send buffer

get next 2 elements

0	2
1	3
2	2

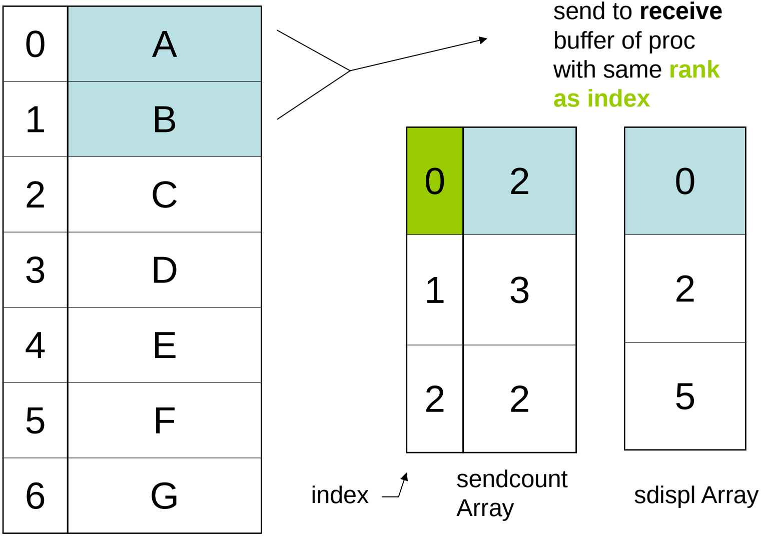
index

sendcount Array

0
2
5

sdispl Array

Example of Send for Proc 0



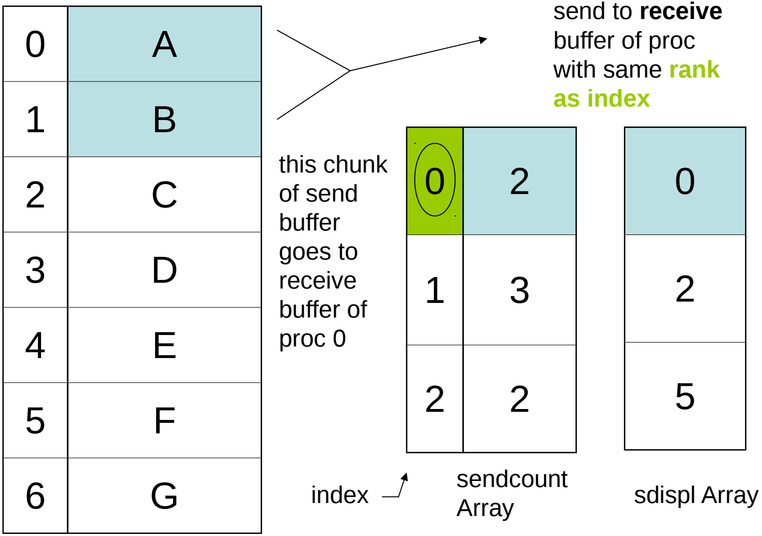
send to **receive** buffer of proc with same **rank as index**

index	Proc 0 send buffer
0	A
1	B
2	C
3	D
4	E
5	F
6	G

index	sendcount Array
0	2
1	3
2	2

sdispl Array
0
2
5

Example of Send for Proc 0



send to **receive** buffer of proc with same **rank as index**

this chunk of send buffer goes to receive buffer of proc 0

0	A
1	B
2	C
3	D
4	E
5	F
6	G

index

Proc 0 send buffer

0	2
1	3
2	2

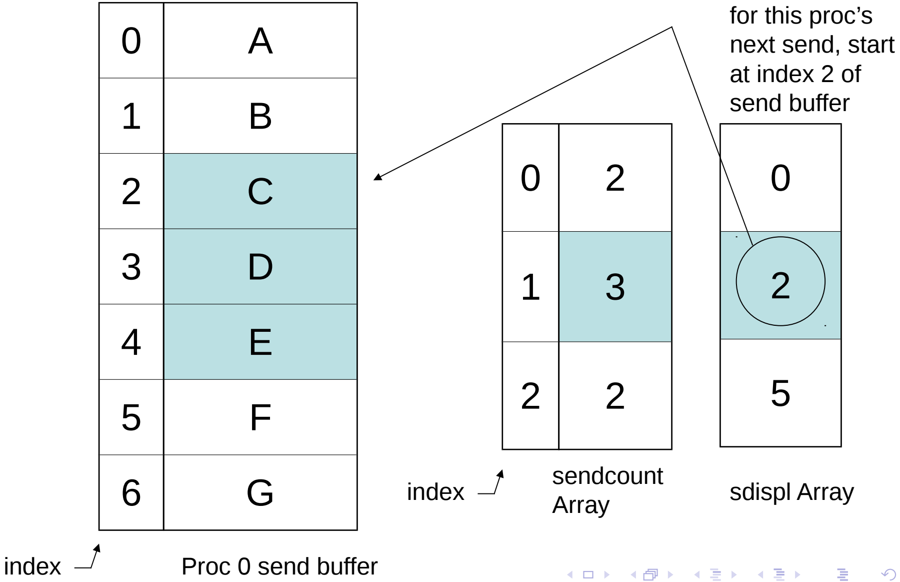
index

sendcount Array

0
2
5

sdispl Array

Example of Send for Proc 0



0	A
1	B
2	C
3	D
4	E
5	F
6	G

index

Proc 0 send buffer

0	2
1	3
2	2

index

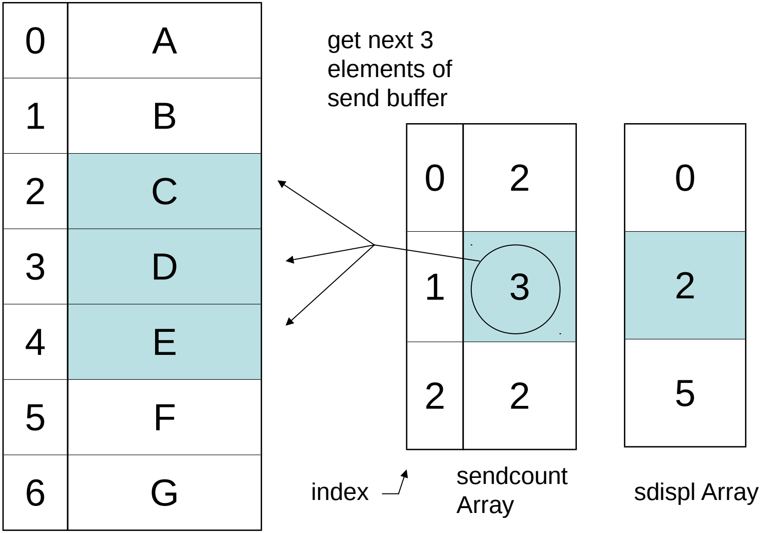
sendcount Array

0
2
5

sdispl Array

for this proc's next send, start at index 2 of send buffer

Example of Send for Proc 0



get next 3 elements of send buffer

index	Proc 0 send buffer
0	A
1	B
2	C
3	D
4	E
5	F
6	G

index	sendcount Array
0	2
1	3
2	2

sdispl Array
0
2
5

index

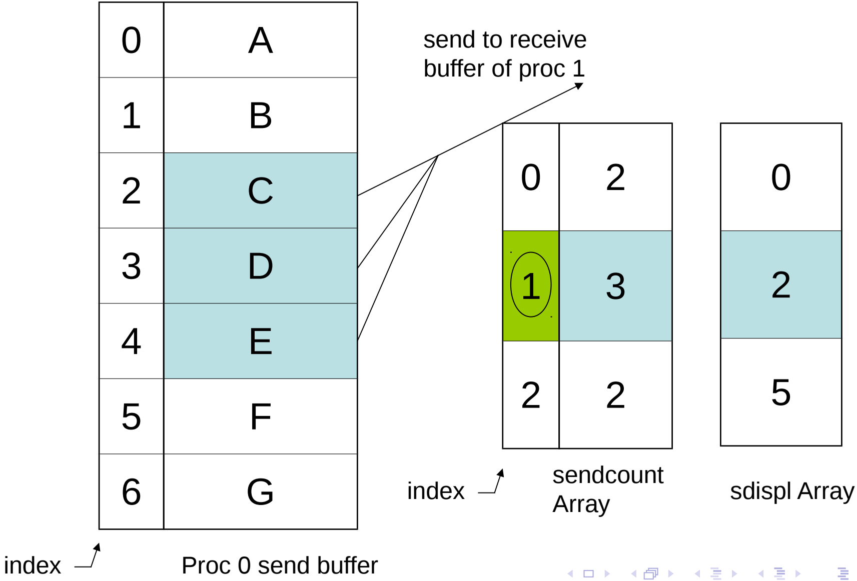
Proc 0 send buffer

index

sendcount Array

sdispl Array

Example of Send for Proc 0



0	A
1	B
2	C
3	D
4	E
5	F
6	G

index

Proc 0 send buffer

send to receive buffer of proc 1

0	2
1	3
2	2

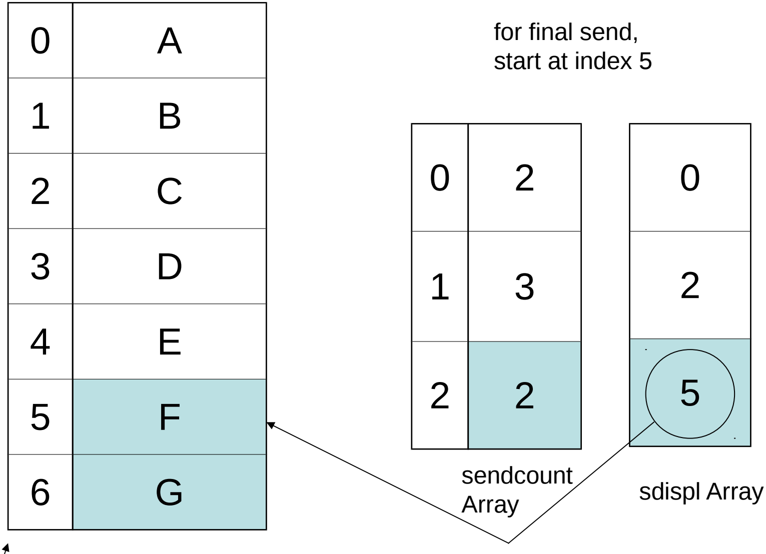
index

sendcount Array

0
2
5

sdispl Array

Example of Send for Proc 0



for final send,
start at index 5

0	A
1	B
2	C
3	D
4	E
5	F
6	G

index

Proc 0 send buffer

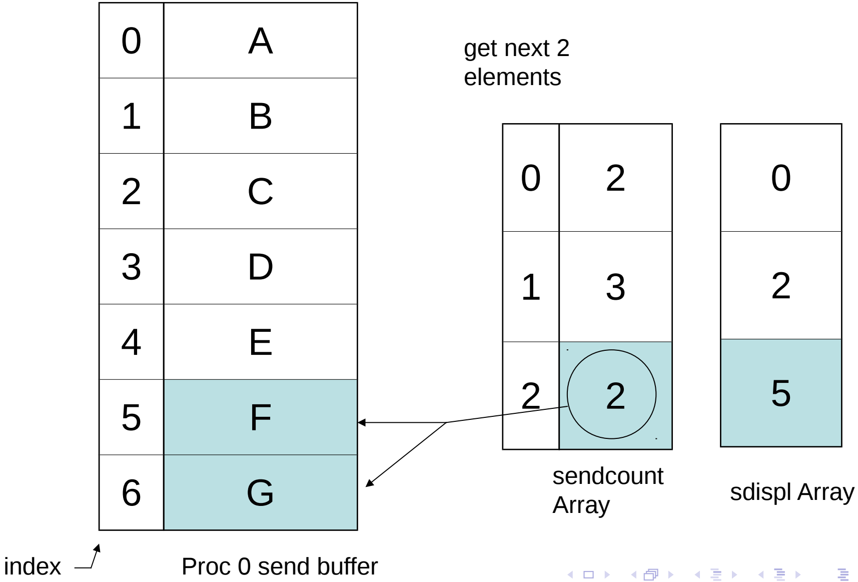
0	2
1	3
2	2

sendcount Array

0
2
5

sdispl Array

Example of Send for Proc 0



0	A
1	B
2	C
3	D
4	E
5	F
6	G

index

Proc 0 send buffer

get next 2 elements

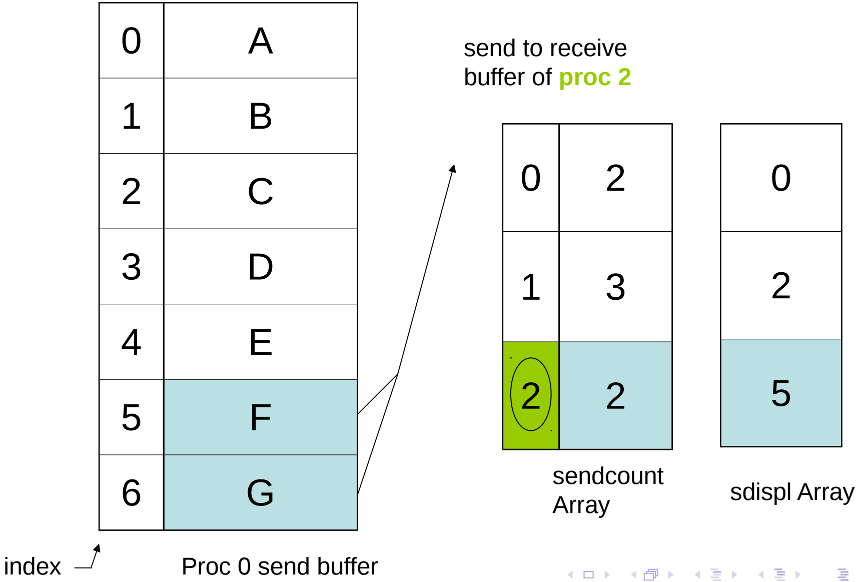
0	2
1	3
2	2

sendcount Array

0
2
5

sdispl Array

Example of Send for Proc 0



0	A
1	B
2	C
3	D
4	E
5	F
6	G

index

Proc 0 send buffer

send to receive buffer of **proc 2**

0	2
1	3
2	2

sendcount Array

0
2
5

sdispl Array

Example of Send for Proc 0

0	A
1	B
2	C
3	D
4	E
5	F
6	G

index

Proc 0 send buffer

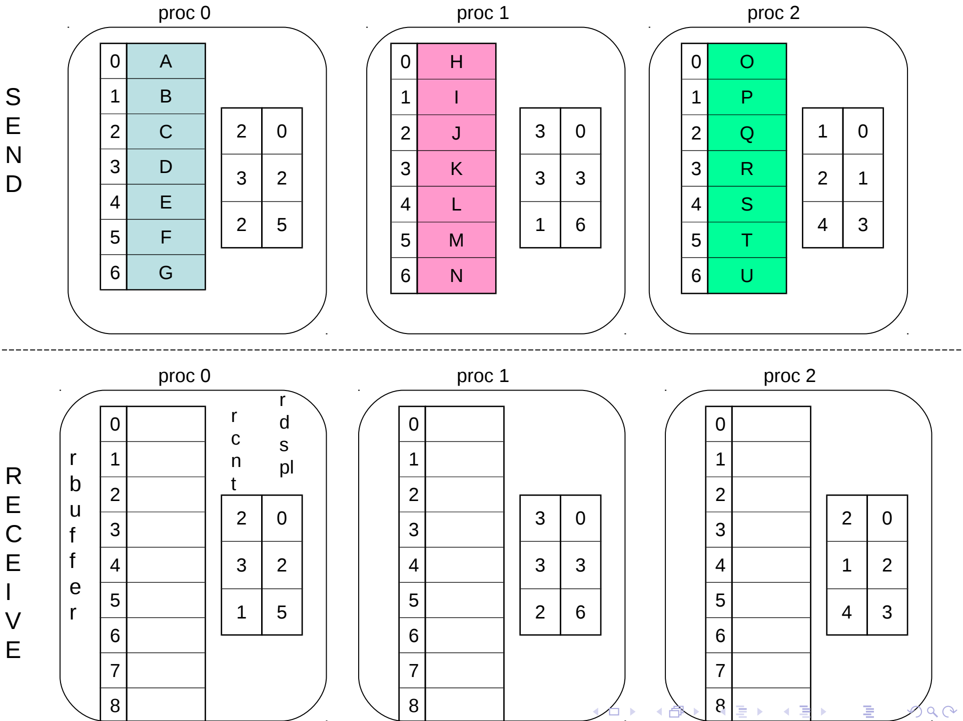
this process occurs for each node in the community

0	2
1	3
2	2

sendcount Array

0
2
5

sdispl Array



SEND

proc 0

0	A
1	B
2	C
3	D
4	E
5	F
6	G

2	0
3	2
2	5

proc 1

0	H
1	I
2	J
3	K
4	L
5	M
6	N

3	0
3	3
1	6

proc 2

0	O
1	P
2	Q
3	R
4	S
5	T
6	U

1	0
2	1
4	3

RECEIVE

proc 0

rbuffer

0	
1	
2	
3	
4	
5	
6	
7	
8	

rcnt	rdspl
2	0
3	2
1	5

proc 1

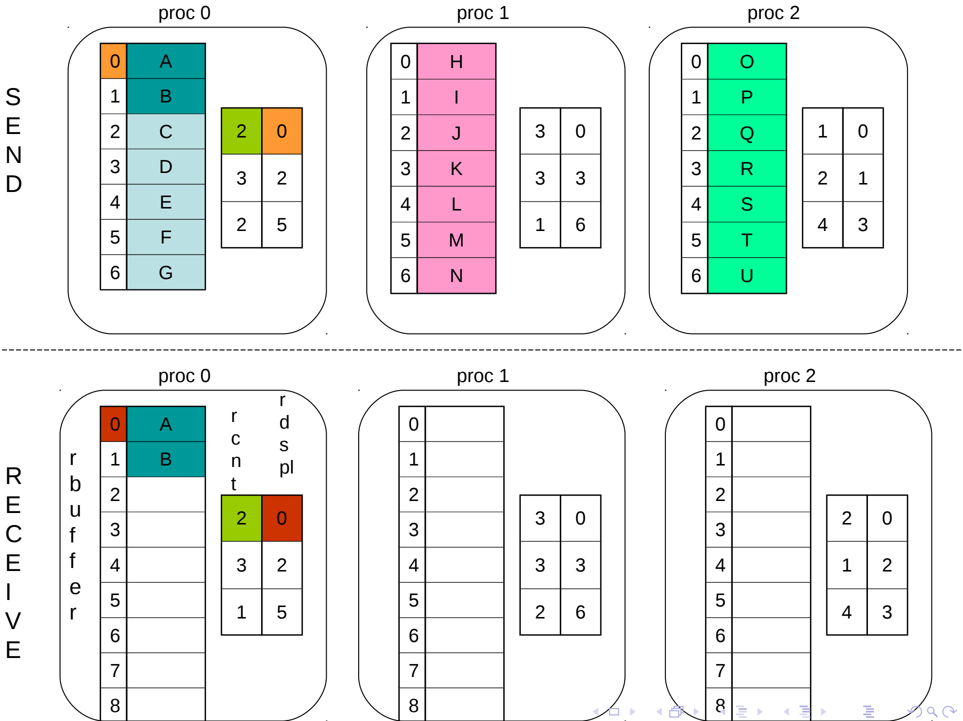
0	
1	
2	
3	
4	
5	
6	
7	
8	

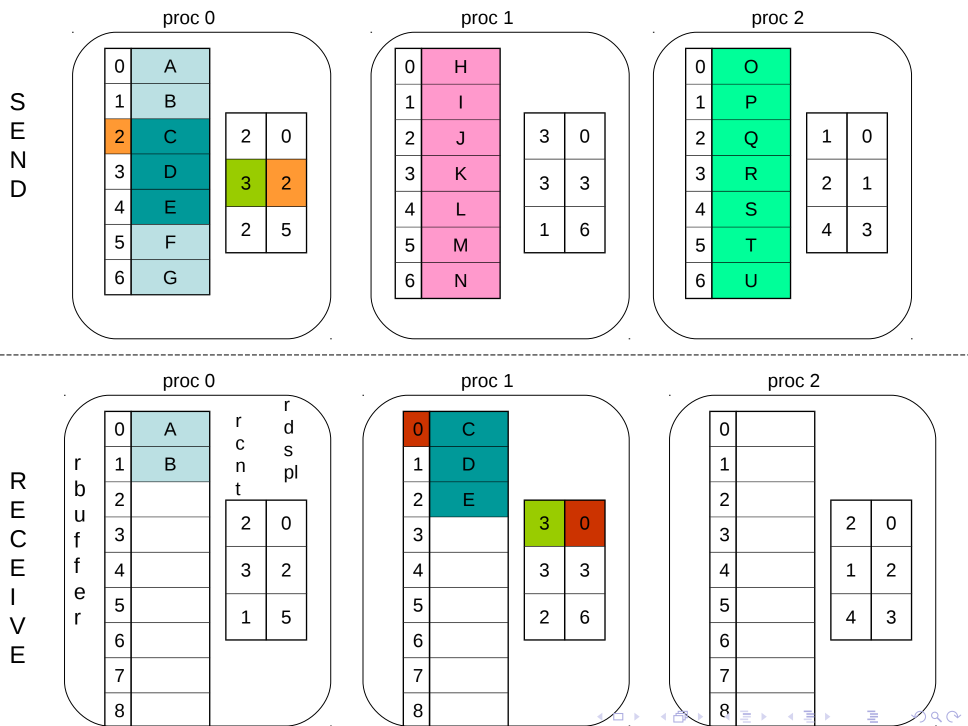
3	0
3	3
2	6

proc 2

0	
1	
2	
3	
4	
5	
6	
7	
8	

2	0
1	2
4	3


SEND
proc 0
0 A
1 B
2 C
3 D
4 E
5 F
6 G
2 0
3 2
2 5
proc 1
0 H
1 I
2 J
3 K
4 L
5 M
6 N
3 0
3 3
1 6
proc 2
0 O
1 P
2 Q
3 R
4 S
5 T
6 U
1 0
2 1
4 3
RECEIVE
proc 0
rbuffer
0 A
1 B
2
3
4
5
6
7
8
rcnt
rdspl
2 0
3 2
1 5
proc 1
0
1
2
3
4
5
6
7
8
3 0
3 3
2 6
proc 2
0
1
2
3
4
5
6
7
8
2 0
1 2
4 3



SEND

proc 0

0	A
1	B
2	C
3	D
4	E
5	F
6	G

2	0
3	2
2	5

proc 1

0	H
1	I
2	J
3	K
4	L
5	M
6	N

3	0
3	3
1	6

proc 2

0	O
1	P
2	Q
3	R
4	S
5	T
6	U

1	0
2	1
4	3

RECEIVE

proc 0

rbuffer

0	A
1	B
2	
3	
4	
5	
6	
7	
8	

rcnt	rdspl
2	0
3	2
1	5

proc 1

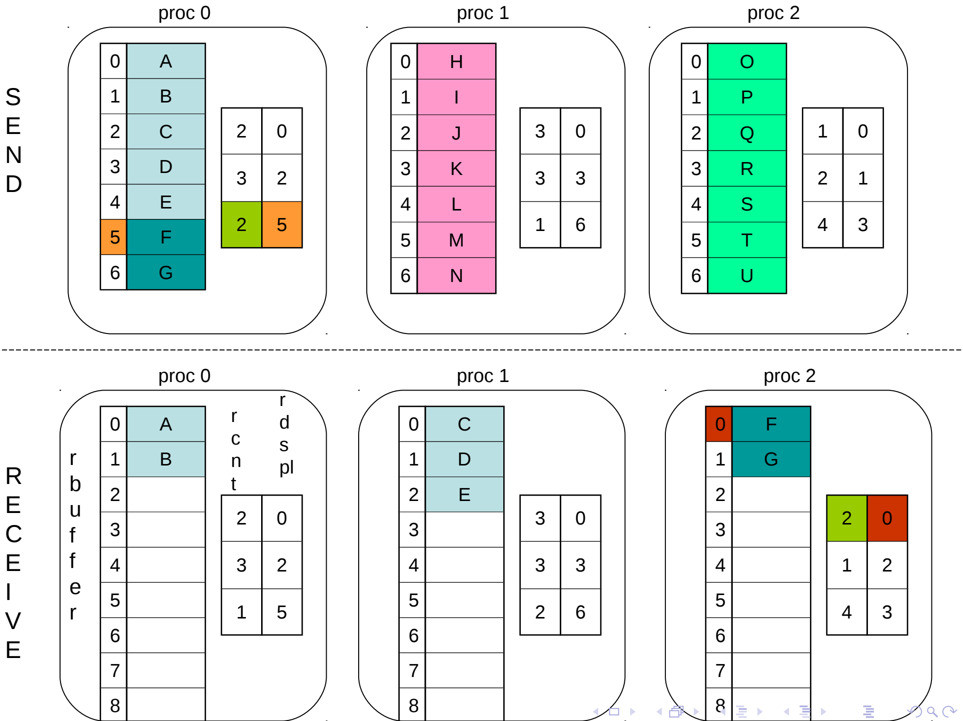
0	C
1	D
2	E
3	
4	
5	
6	
7	
8	

3	0
3	3
2	6

proc 2

0	
1	
2	
3	
4	
5	
6	
7	
8	

2	0
1	2
4	3



SEND

proc 0

0	A
1	B
2	C
3	D
4	E
5	F
6	G

2	0
3	2
2	5

proc 1

0	H
1	I
2	J
3	K
4	L
5	M
6	N

3	0
3	3
1	6

proc 2

0	O
1	P
2	Q
3	R
4	S
5	T
6	U

1	0
2	1
4	3

RECEIVE

proc 0

rbuffer

0	A
1	B
2	
3	
4	
5	
6	
7	
8	

rcnt	rdspl
2	0
3	2
1	5

proc 1

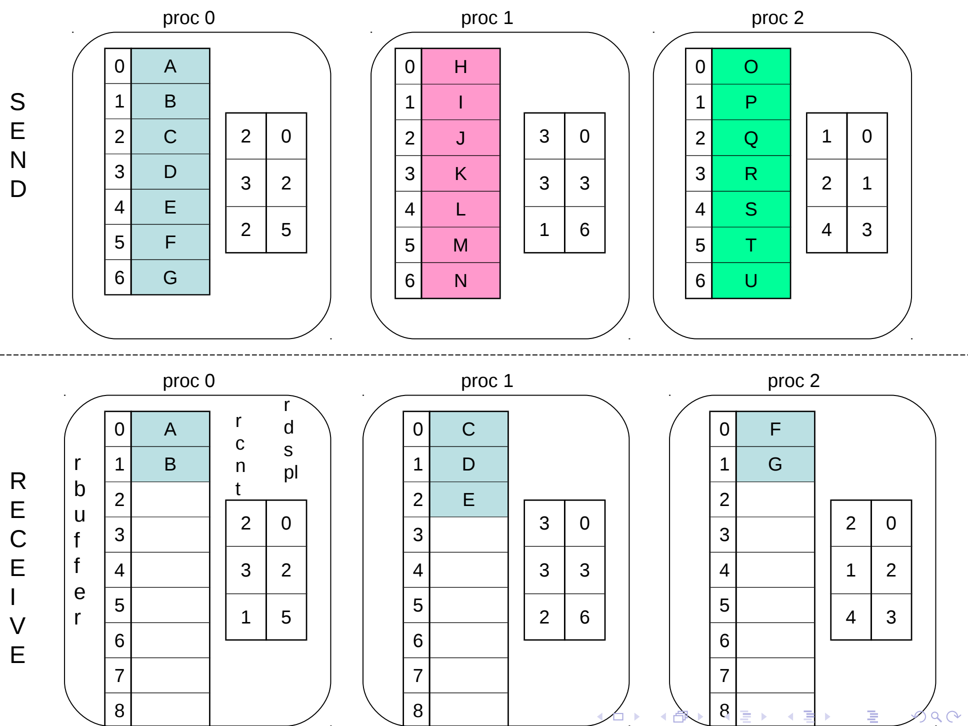
0	C
1	D
2	E
3	
4	
5	
6	
7	
8	

3	0
3	3
2	6

proc 2

0	F
1	G
2	
3	
4	
5	
6	
7	
8	

2	0
1	2
4	3



SEND

proc 0

0	A
1	B
2	C
3	D
4	E
5	F
6	G

2	0
3	2
2	5

proc 1

0	H
1	I
2	J
3	K
4	L
5	M
6	N

3	0
3	3
1	6

proc 2

0	O
1	P
2	Q
3	R
4	S
5	T
6	U

1	0
2	1
4	3

RECEIVE

proc 0

rbuffer

0	A
1	B
2	
3	
4	
5	
6	
7	
8	

rcnt	rdspl
2	0
3	2
1	5

proc 1

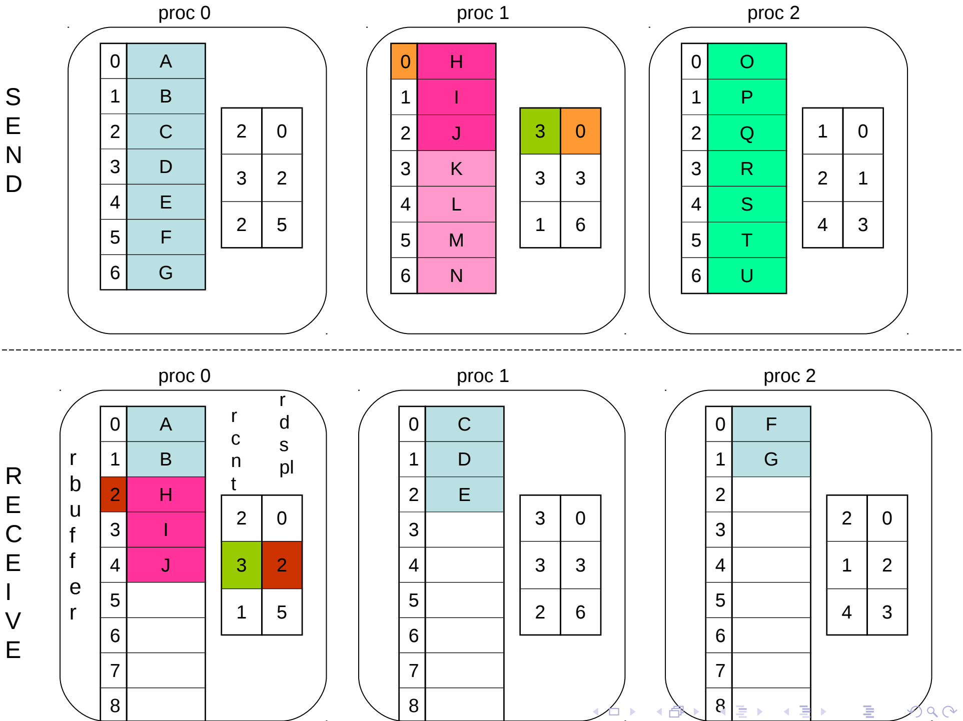
0	C
1	D
2	E
3	
4	
5	
6	
7	
8	

3	0
3	3
2	6

proc 2

0	F
1	G
2	
3	
4	
5	
6	
7	
8	

2	0
1	2
4	3



SEND

proc 0

0	A
1	B
2	C
3	D
4	E
5	F
6	G

2	0
3	2
2	5

proc 1

0	H
1	I
2	J
3	K
4	L
5	M
6	N

3	0
3	3
1	6

proc 2

0	O
1	P
2	Q
3	R
4	S
5	T
6	U

1	0
2	1
4	3

RECEIVE

proc 0

rbuffer

0	A
1	B
2	H
3	I
4	J
5	
6	
7	
8	

rcnt	rdspl
2	0
3	2
1	5

proc 1

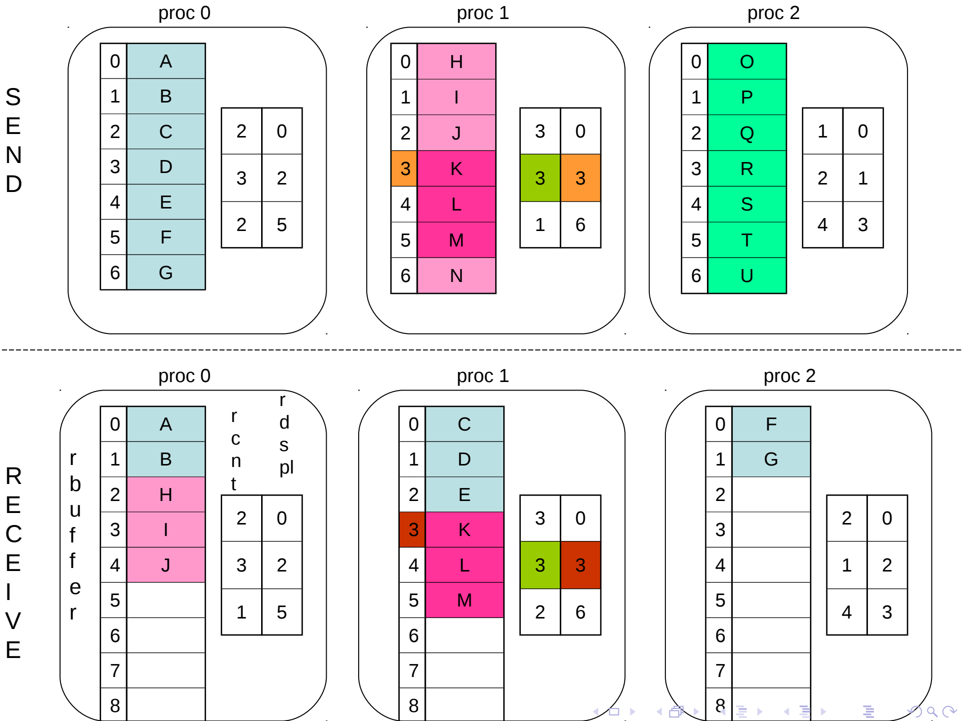
0	C
1	D
2	E
3	
4	
5	
6	
7	
8	

3	0
3	3
2	6

proc 2

0	F
1	G
2	
3	
4	
5	
6	
7	
8	

2	0
1	2
4	3



SEND, proc 0

0	A
1	B
2	C
3	D
4	E
5	F
6	G

2	0
3	2
2	5

SEND, proc 1

0	H
1	I
2	J
3	K
4	L
5	M
6	N

3	0
3	3
1	6

SEND, proc 2

0	O
1	P
2	Q
3	R
4	S
5	T
6	U

1	0
2	1
4	3

RECEIVE, proc 0

0	A
1	B
2	H
3	I
4	J
5	
6	
7	
8	

rcnt	rdspl
2	0
3	2
1	5

RECEIVE, proc 1

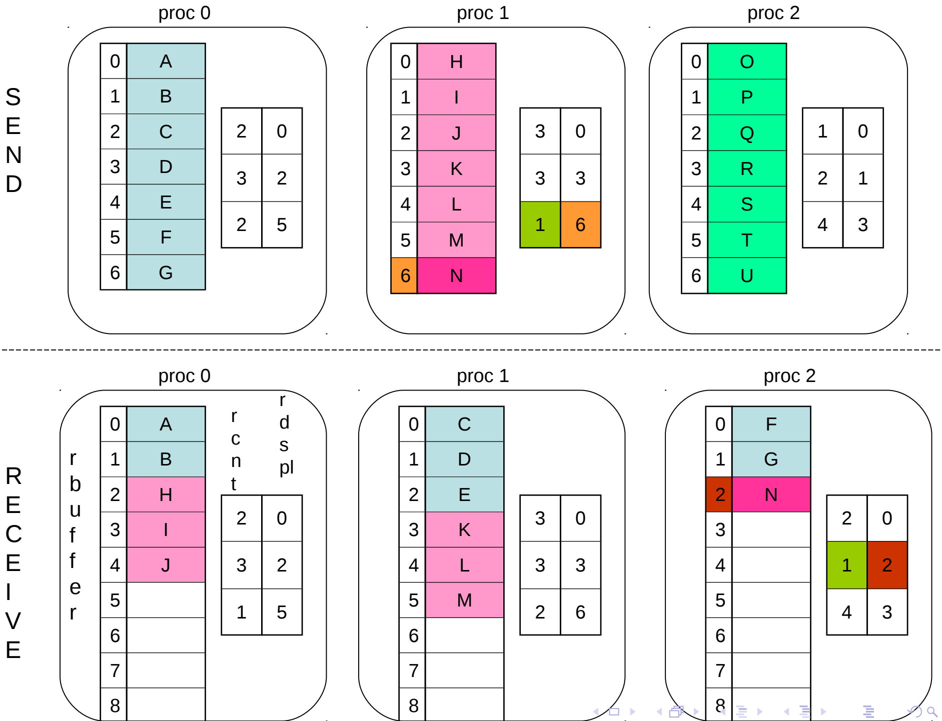
0	C
1	D
2	E
3	K
4	L
5	M
6	
7	
8	

3	0
3	3
2	6

RECEIVE, proc 2

0	F
1	G
2	
3	
4	
5	
6	
7	
8	

2	0
1	2
4	3



SEND

proc 0

0	A
1	B
2	C
3	D
4	E
5	F
6	G

2	0
3	2
2	5

proc 1

0	H
1	I
2	J
3	K
4	L
5	M
6	N

3	0
3	3
1	6

proc 2

0	O
1	P
2	Q
3	R
4	S
5	T
6	U

1	0
2	1
4	3

RECEIVE

proc 0

rbuffer

0	A
1	B
2	H
3	I
4	J
5	
6	
7	
8	

rcnt	rdspl
2	0
3	2
1	5

proc 1

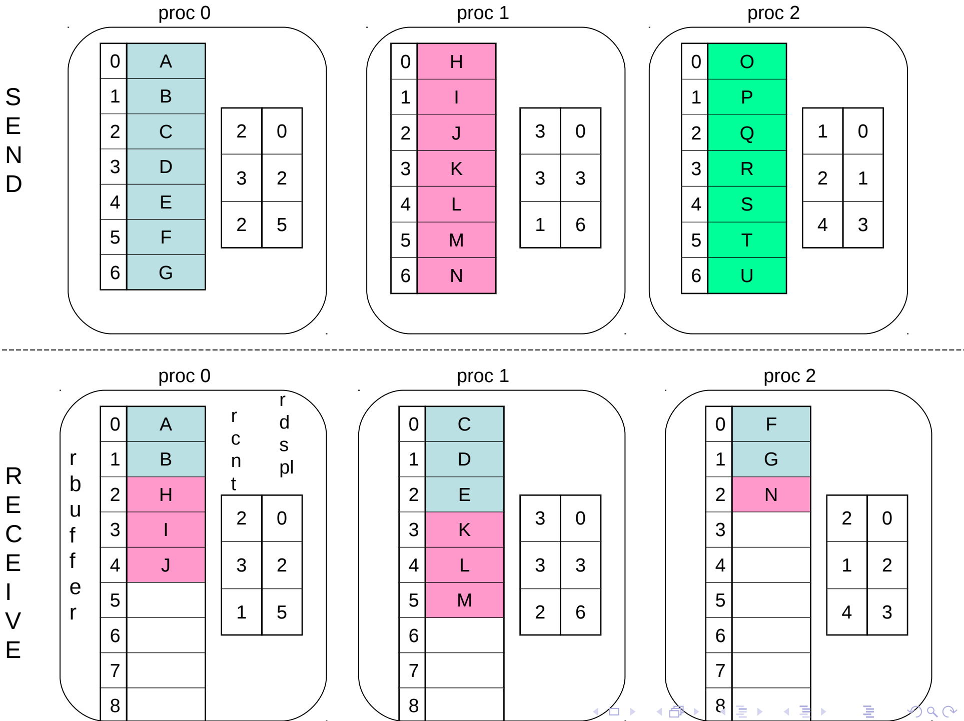
0	C
1	D
2	E
3	K
4	L
5	M
6	
7	
8	

3	0
3	3
2	6

proc 2

0	F
1	G
2	N
3	
4	
5	
6	
7	
8	

2	0
1	2
4	3



SEND

proc 0

0	A
1	B
2	C
3	D
4	E
5	F
6	G

2	0
3	2
2	5

proc 1

0	H
1	I
2	J
3	K
4	L
5	M
6	N

3	0
3	3
1	6

proc 2

0	O
1	P
2	Q
3	R
4	S
5	T
6	U

1	0
2	1
4	3

RECEIVE

proc 0

rbuffer

0	A
1	B
2	H
3	I
4	J
5	
6	
7	
8	

rcnt	rdspl
2	0
3	2
1	5

proc 1

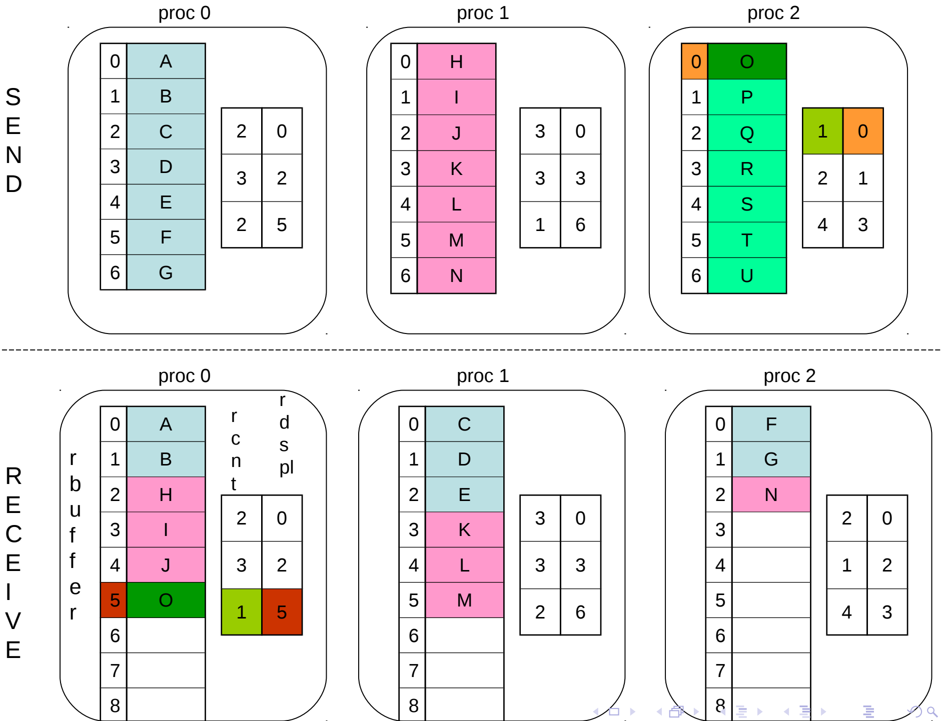
0	C
1	D
2	E
3	K
4	L
5	M
6	
7	
8	

3	0
3	3
2	6

proc 2

0	F
1	G
2	N
3	
4	
5	
6	
7	
8	

2	0
1	2
4	3



SEND

proc 0

0	A
1	B
2	C
3	D
4	E
5	F
6	G

2	0
3	2
2	5

proc 1

0	H
1	I
2	J
3	K
4	L
5	M
6	N

3	0
3	3
1	6

proc 2

0	O
1	P
2	Q
3	R
4	S
5	T
6	U

1	0
2	1
4	3

RECEIVE

proc 0

rbuffer

0	A
1	B
2	H
3	I
4	J
5	O
6	
7	
8	

rcnt	rdspl
2	0
3	2
1	5

proc 1

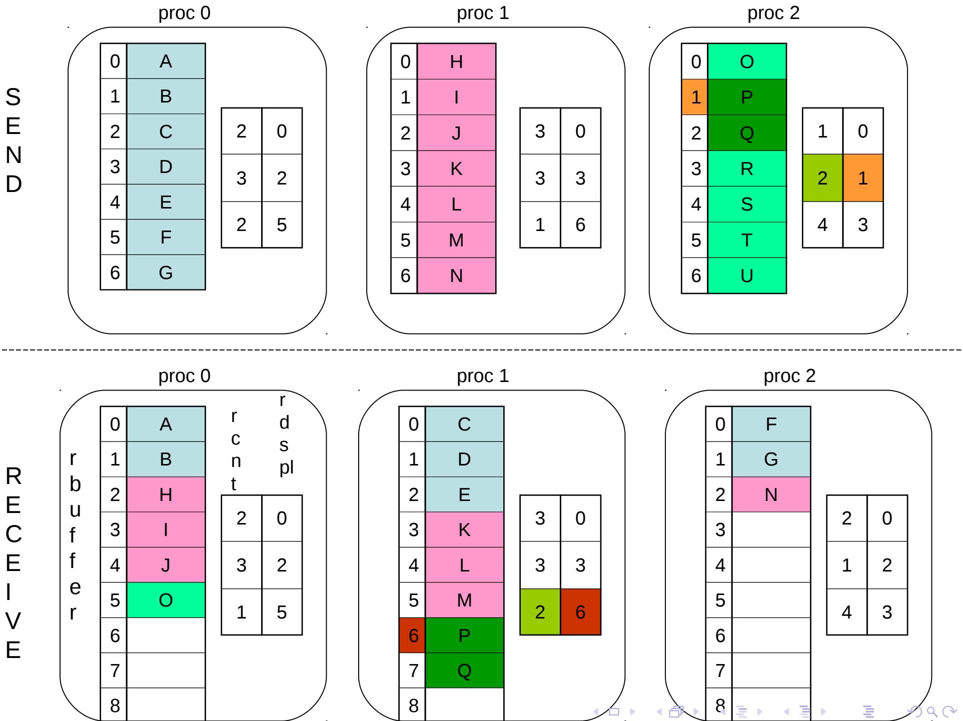
0	C
1	D
2	E
3	K
4	L
5	M
6	
7	
8	

3	0
3	3
2	6

proc 2

0	F
1	G
2	N
3	
4	
5	
6	
7	
8	

2	0
1	2
4	3



SEND

proc 0

0	A
1	B
2	C
3	D
4	E
5	F
6	G

2	0
3	2
2	5

proc 1

0	H
1	I
2	J
3	K
4	L
5	M
6	N

3	0
3	3
1	6

proc 2

0	O
1	P
2	Q
3	R
4	S
5	T
6	U

1	0
2	1
4	3

RECEIVE

proc 0

rbuffer

0	A
1	B
2	H
3	I
4	J
5	O
6	
7	
8	

rcnt	rdspl
2	0
3	2
1	5

proc 1

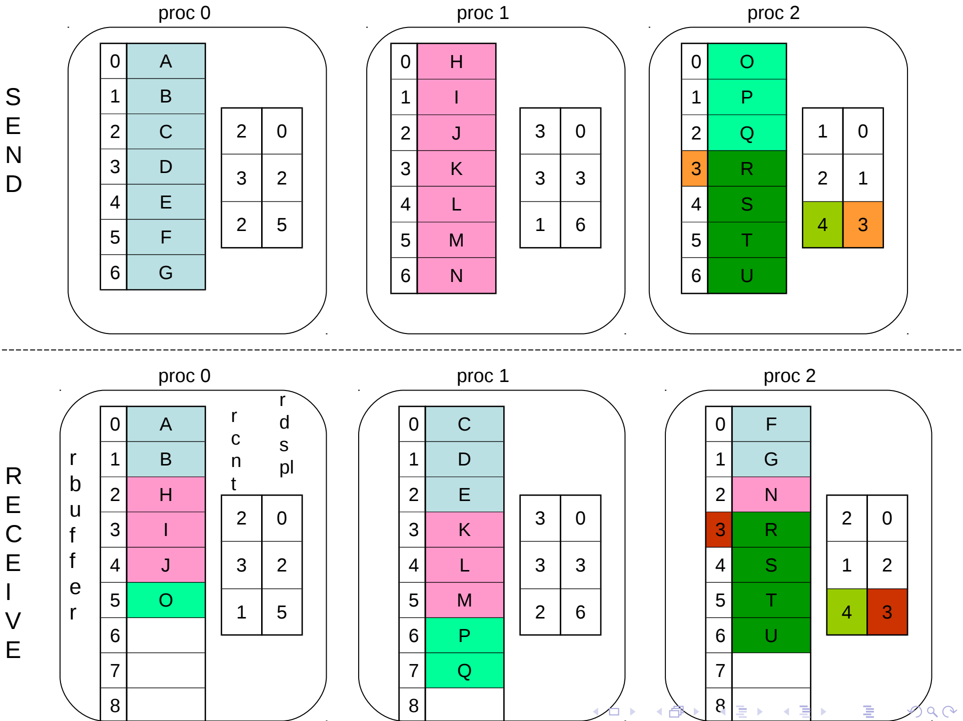
0	C
1	D
2	E
3	K
4	L
5	M
6	P
7	Q
8	

3	0
3	3
2	6

proc 2

0	F
1	G
2	N
3	
4	
5	
6	
7	
8	

2	0
1	2
4	3



SEND

proc 0

0	A
1	B
2	C
3	D
4	E
5	F
6	G

2	0
3	2
2	5

proc 1

0	H
1	I
2	J
3	K
4	L
5	M
6	N

3	0
3	3
1	6

proc 2

0	O
1	P
2	Q
3	R
4	S
5	T
6	U

1	0
2	1
4	3

RECEIVE

proc 0

rbuffer

0	A
1	B
2	H
3	I
4	J
5	O
6	
7	
8	

rcnt	rdspl
2	0
3	2
1	5

proc 1

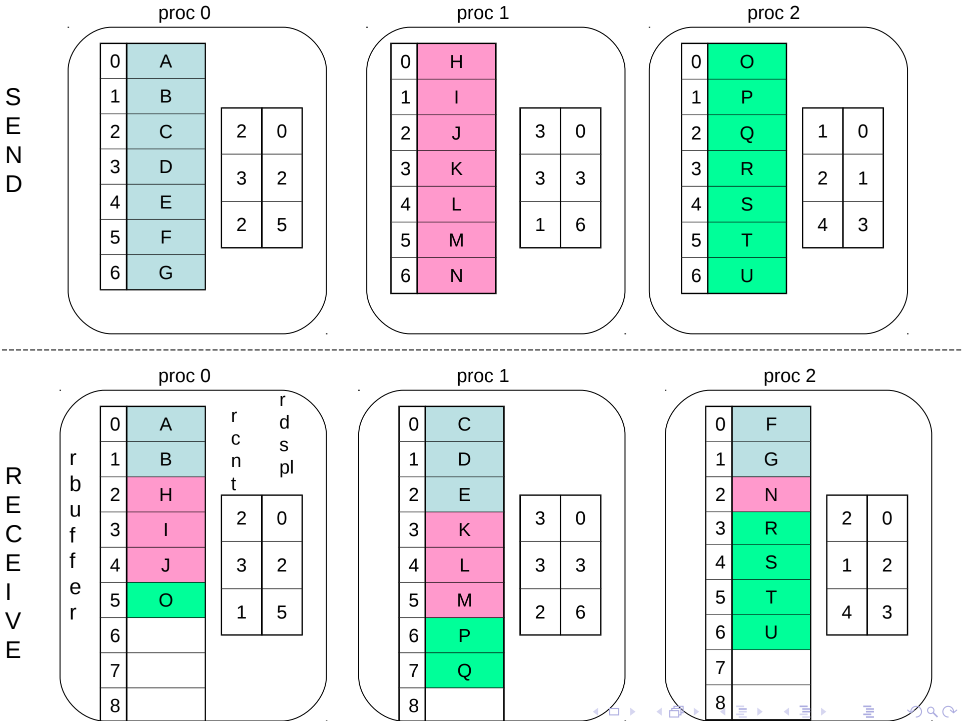
0	C
1	D
2	E
3	K
4	L
5	M
6	P
7	Q
8	

3	0
3	3
2	6

proc 2

0	F
1	G
2	N
3	R
4	S
5	T
6	U
7	
8	

2	0
1	2
4	3



SEND

proc 0

0	A
1	B
2	C
3	D
4	E
5	F
6	G

2	0
3	2
2	5

proc 1

0	H
1	I
2	J
3	K
4	L
5	M
6	N

3	0
3	3
1	6

proc 2

0	O
1	P
2	Q
3	R
4	S
5	T
6	U

1	0
2	1
4	3

RECEIVE

proc 0

rbuffer

0	A
1	B
2	H
3	I
4	J
5	O
6	
7	
8	

rcnt	rdspl
2	0
3	2
1	5

proc 1

0	C
1	D
2	E
3	K
4	L
5	M
6	P
7	Q
8	

3	0
3	3
2	6

proc 2

0	F
1	G
2	N
3	R
4	S
5	T
6	U
7	
8	

2	0
1	2
4	3

Notes on AlltoAllv

- A receive buffer could potentially be as large as the sum of all send buffer sizes
- Care must be taken to coincide send counts with receive counts and displacements so data is not overwritten

Today's Biz

1. Quick Review
2. Reminders
3. Parallel SCC
4. More Centrality
5. Even More MPI
6. **More PageRank Tutorial**

More PageRank Tutorial

1. OpenMP - Work Queueing
2. MPI - Alltoallv Communication

More PageRank Tutorial
Blank code and data available on website
(Lecture 5)

`www.cs.rpi.edu/~slotag/classes/FA16/index.html`