Lecture 3: Multiprocessor Scheduling

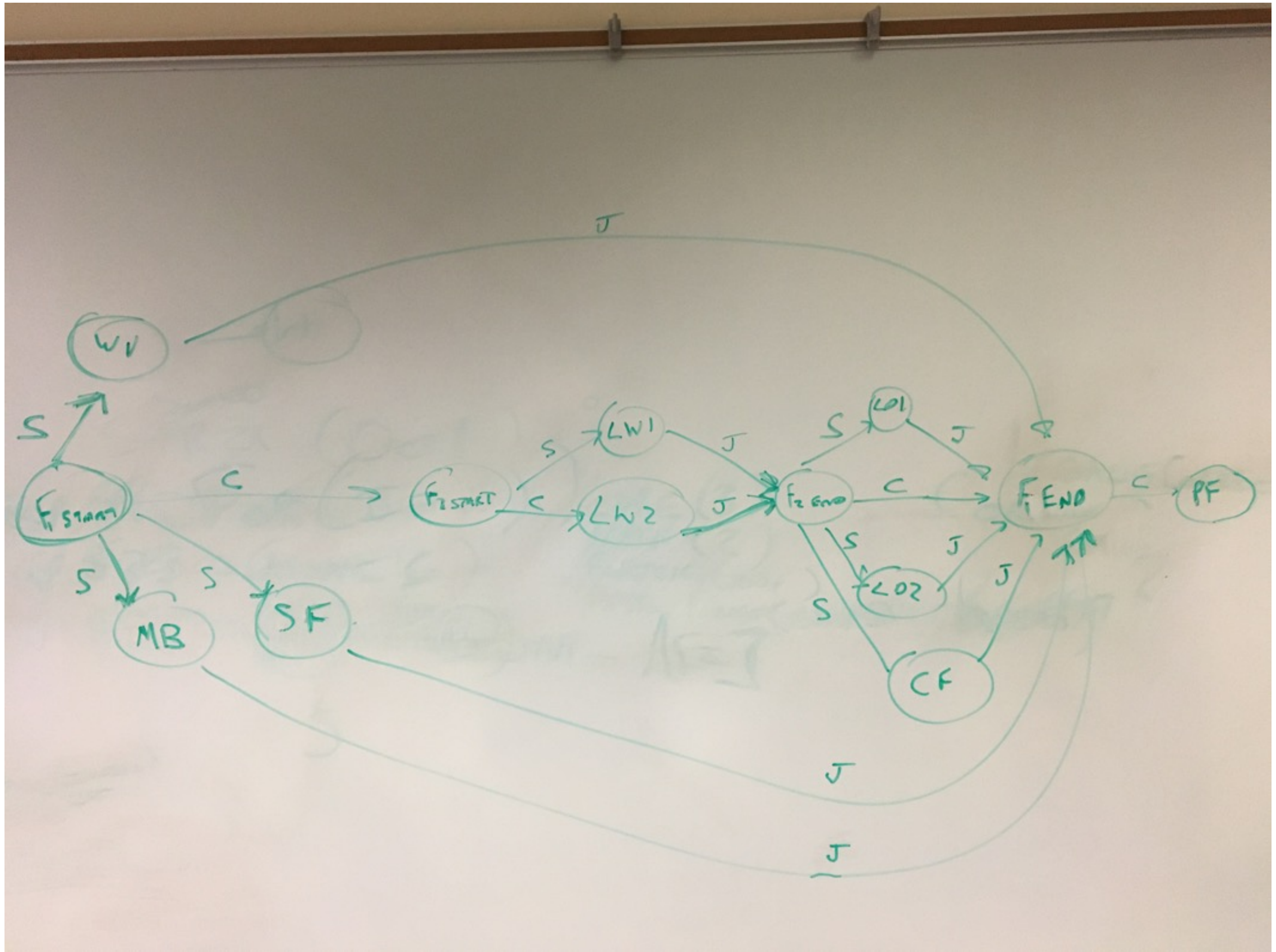
Mack Joyner
mjoyner@rice.edu

http://comp322.rice.edu

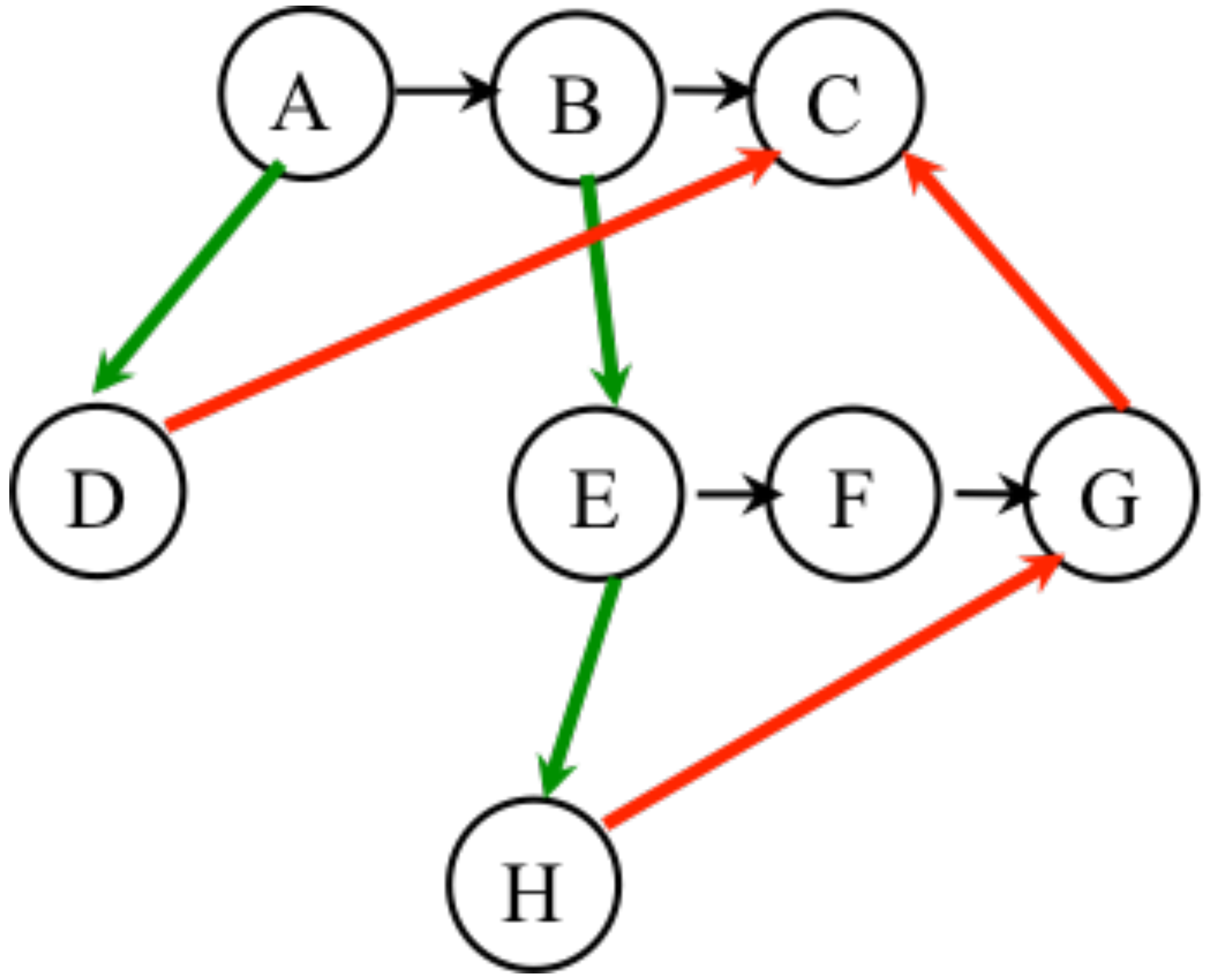
Computation Graph Exercise

```
finish { (F1)
   async  (WV) { Watch COMP 322 video for topic 1.2 by 1pm on Wednesday
                 Watch COMP 322 video for topic 1.3 by 1pm on Wednesday
                }
   async (MB)   Make your bed
   async (SF) { Clean out your fridge
                Buy food supplies and store them in fridge }
   finish (F2) { async  Run load 1 in washer (LW1)
                 Run load 2 in washer (LW2) }
    async Run load 1 in dryer (LD1)
    async Run load 2 in dryer (LD2)
    async Call your family (CF)
}
Post on Facebook that you're done with all your tasks! (PF)
```

Computation Graph Exercise



One Possible Solution to Worksheet 2 (Reverse Engineering a Computation Graph)




```
A();
finish { // F1
  async D();
  B();
  E();
  finish { // F2
    async H();
    F();
  } // F2
  G();
} // F1
C();
```

Observations:

- Any node with out-degree > 1 must be an async (must have an outgoing spawn edge)
- Any node with in-degree > 1 must be an end-finish (must have an incoming join edge
- Adding or removing transitive edges does not impact ordering constraints

Ordering Constraints and Transitive Edges in a Computation Graph

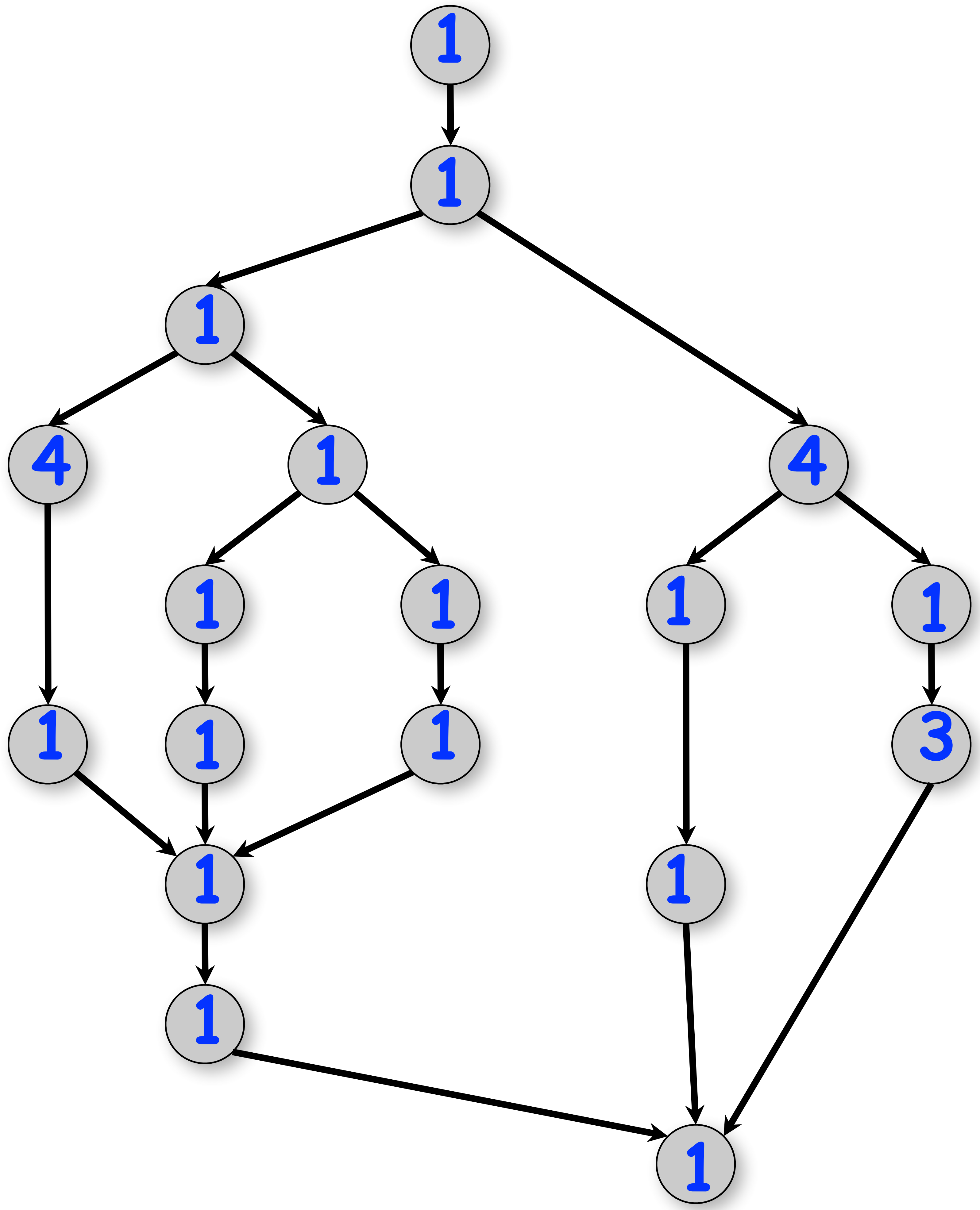
- The primary purpose of a computation graph is to determine if an ordering constraint exists between two steps (nodes)
  - —Observation: Node A must be performed before node B if there is a path of directed edges from A and B

- An edge, X →Y, in a computation graph is said to be *transitive* if there exists a path of directed edges from X to Y that does not include the X →Y edge
  - —Observation: Adding or removing a transitive edge does not change the ordering constraints in a computation graph

Ideal Parallelism (Recap)

- Define ideal parallelism of Computation G Graph as the ratio, WORK(G)/CPL(G)

- Ideal Parallelism only depends on the computation graph, and is the speedup that you can obtain with an unbounded number of processors



Example:

WORK(G) = 26

CPL(G) = 11

Ideal Parallelism = WORK(G)/CPL(G) = 26/11 ~ 2.36

What is the critical path length of this parallel computation?

```
finish { // F1
  async A; // Boil water & pasta (10)
  finish { // F2
    async B1; // Chop veggies (5)
    async B2; // Brown meat (10)
  } // F2
  B3; // Make pasta sauce (5)
} // F1
```

Step B1

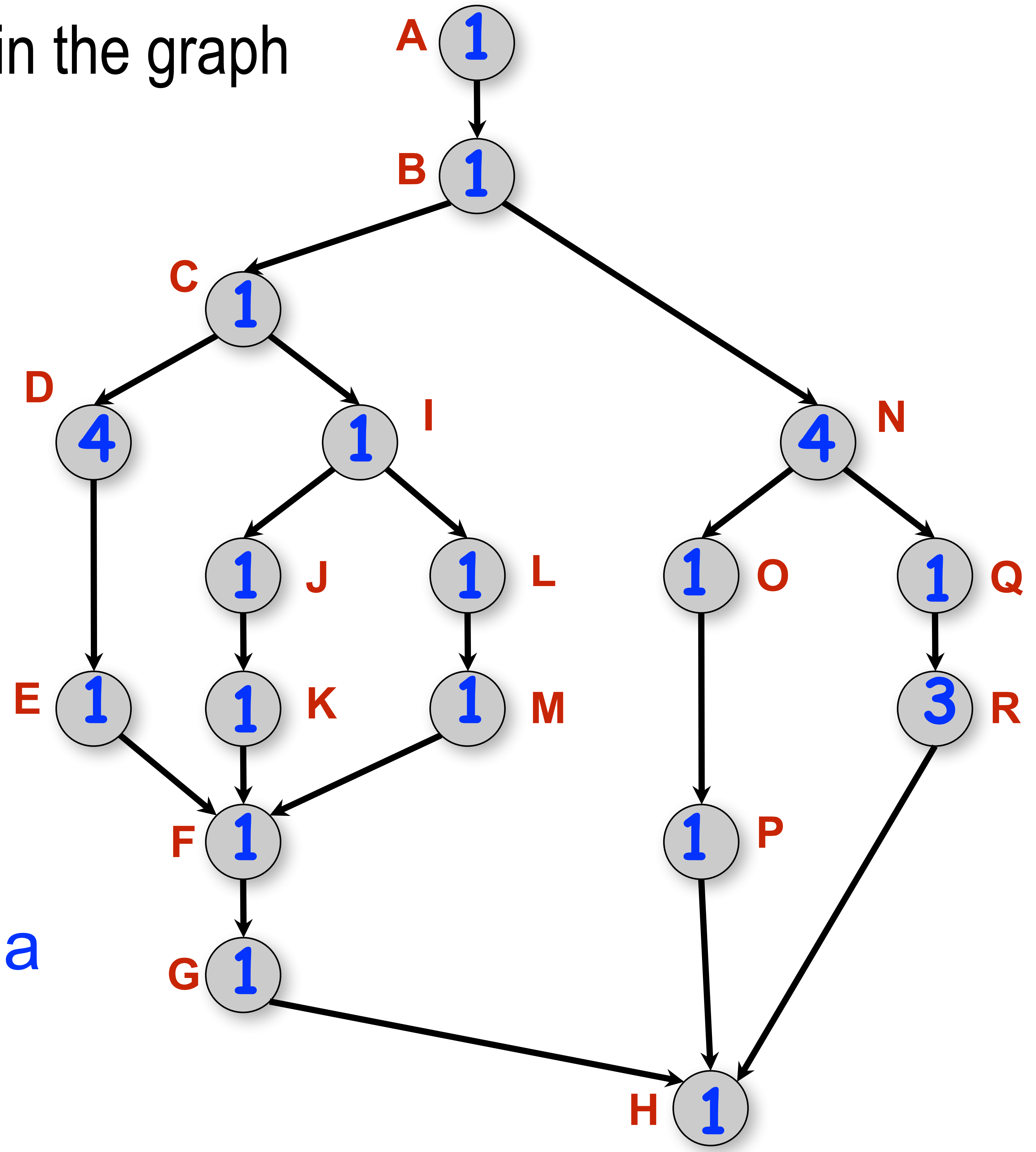
Step B2

Step A

Step B3

Scheduling of a Computation Graph on a fixed number of processors

Node label = time(N), for all nodes N in the graph



Start time	Proc 1	Proc 2	Proc 3
0	A		
1	B		
2	C	N	
3	D	N	I
4	D	N	J
5	D	N	K
6	D	Q	L
7	E	R	M
8	F	R	O
9	G	R	P
10	H		
11	Completion time = 11		

NOTE: this schedule achieved a completion time of 11. Can we do better?

Scheduling of a Computation Graph on a fixed number of processors

- Assume that node N takes TIME(N) regardless of which processor it executes on, and that there is no overhead for creating parallel tasks

- A schedule specifies the following for each node
  - —START(N) = start time
  - —PROC(N) = index of processor in range 1...P

  such that
  - —START(i) + TIME(i) <= START(j), for all CG edges from i to j (Precedence constraint)
  - —A node occupies consecutive time slots in a processor (Non-preemption constraint)
  - —All nodes assigned to the same processor occupy distinct time slots (Resource constraint)

Greedy Schedule

- A greedy schedule is one that never forces a processor to be idle when one or more nodes are ready for execution

- A node is ready for execution if all its predecessors have been executed
- Observations
  - T_1 = WORK(G), for all greedy schedules
  - T_∞ = CPL(G), for all greedy schedules

- $T_P(S)$ = execution time of schedule S for computation graph G on P processors

Lower Bounds on Execution Time of Schedules

- Let T_P = execution time of a schedule for computation graph G on P processors
  - T_P can be different for different schedules, for same values of G and P

- Lower bounds for all greedy schedules
  - Capacity bound: $T_P \geq WORK(G)/P$
  - Critical path bound: $T_P \geq CPL(G)$

- Putting them together
  - $T_P \geq \max(WORK(G)/P, CPL(G))$

Upper Bound on Execution Time of Greedy Schedules

Theorem [Graham '66].
Any greedy scheduler achieves

$$T_P \le WORK(G)/P + CPL(G)$$

Start time	Proc 1	Proc 2	Proc 3
0	A		
1	B		
2	C	N	
3	D	N	I
4	D	N	J
5	D	N	K
6	D	Q	L
7	E	R	M
8	F	R	O
9	G	R	P
10	H		
11			

Proof sketch:
Define a time step to be complete if P processors are scheduled at that time, or incomplete otherwise

complete time steps ≤ WORK(G)/P

incomplete time steps ≤ CPL(G)

Bounding the Performance of Greedy Schedulers

Combine lower and upper bounds to get

$\max(\text{WORK}(G)/P, \text{CPL}(G)) \leq T_P \leq \text{WORK}(G)/P + \text{CPL}(G)$

Corollary: Any greedy scheduler achieves execution time T_P that is within a factor of 2 of the optimal time (since max(a,b) and (a+b) are within a factor of 2 of each other, for any $a \geq 0, b \geq 0$).