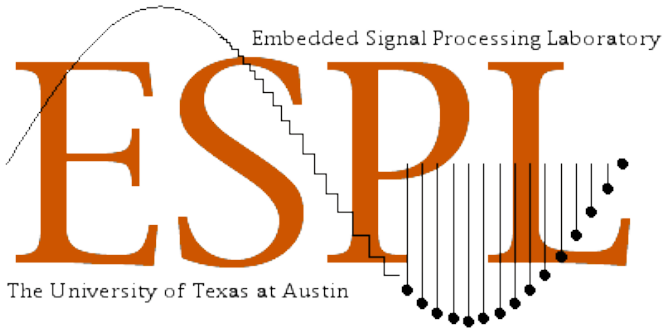
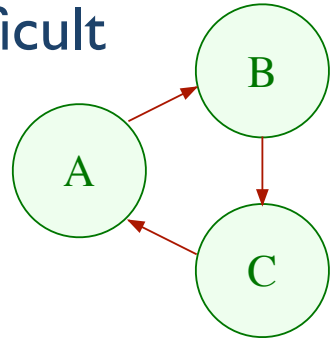
A Distributed Deadlock Detection and Resolution Algorithm for Process Networks

Gregory Allen, Paul Zucknick, Brian Evans
Applied Research Laboratories, and
Dept. of Electrical and Computer Engineering
The University of Texas at Austin
ICASSP 2007



Motivation

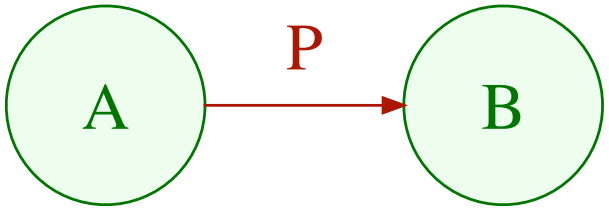
- DSP systems are growing in size and complexity
- Parallel & distributed implementations are necessary
- *Problem:* Effective parallel programming is difficult
  - Non-determinate execution
  - Hard to predict and prevent deadlock
  - Difficult to make scalable software (e.g. rendezvous models)



- Current approaches typically lack formal underpinnings

Process Networks (PN)

- *Solution:* Process Networks, a formal model [Kahn 74]
- Mathematically provable properties
  - Guarantees determinate execution
  - Allows concurrent execution
- A dataflow model
  - Each node represents a computational unit
  - Each edge represents a one-way FIFO queue
- Naturally models parallelism in a DSP system
- Extremely scalable with simple, local scheduling rules



Bounded Scheduling of PN

- Kahn's original PN assumes infinite memory!
- Clever dynamic scheduling of the nodes allows execution in bounded memory, if it is possible [Parks 95]
  - May introduce *artificial deadlocks* due to queue bounds
  - Dynamic deadlock detection & resolution required
  - Lengthen shortest deadlocked full queue to resolve

Author(s)	Parks '95	Geilen & Basten '03
Deadlock detector	Global deadlocks	Local deadlocks
Preserves PN properties	No (counterexamples)	Yes, if an *effective* PN

- Deadlock detection algorithms were not provided

Bounded Scheduling of PN

- Existing distributed algorithm [Mitchell & Merritt 84] can detect presence of deadlocks in a PN [Olson & Evans 05]
- We present an algorithm to detect *and* resolve artificial deadlocks for bounded scheduling of PN
- D4R algorithm: Distributed Dynamic Deadlock Detection and Resolution algorithm
  - Determines whether a deadlock is real or artificial
  - For artificial deadlocks, notifies node which is blocked on culpable queue that must grow for resolution
  - Distributed and scalable (good for distributed PN)

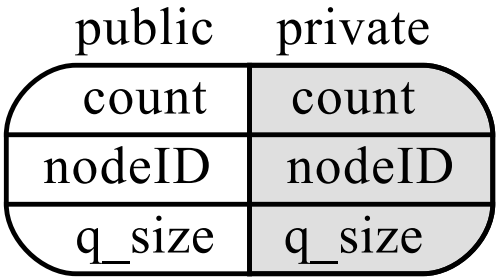
Mitchell & Merritt Algorithm [1984]

- Originally for distributed database applications
- A *single resource* algorithm -- a process waits only on a single other process (also true with PN)
- Each process contains algorithm state variables
- Transactions between interacting (waiting) processes construct a *wait-for* dependency graph
- A dependency cycle indicates a deadlock, which is detected by lowest priority process in the cycle
- Proofs for correctness provided

Our D4R algorithm borrows heavily from M&M

D4R State Variables

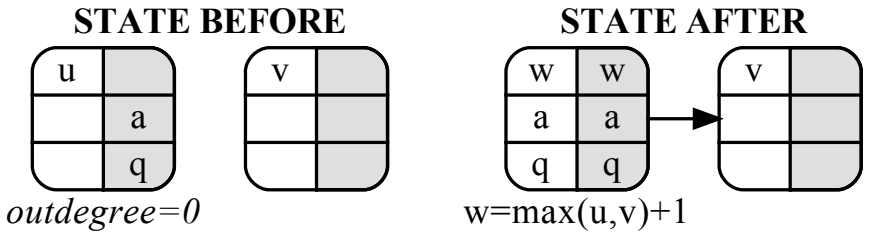
- Each process contains public and private triples of D4R algorithm state information:



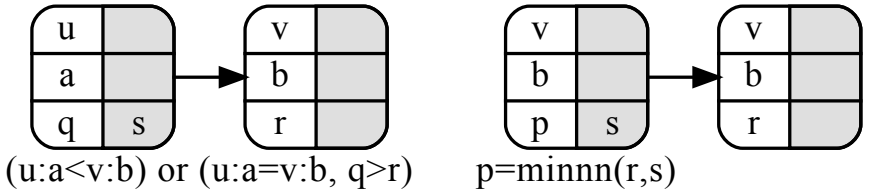
  - *count*, a non-decreasing counter
  - *nodeID*, a unique node identifier
  - *q_size*, size of queue upon which node is blocked
    - Set to -1 when blocking on read of an empty queue
    - Serves same function as M&M's priority variable
    - Will identify the deadlock type and the culpable node
  - *count:nodeID* expresses concatenation (as in M&M)

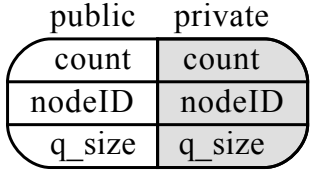
D4R State Transitions

- BLOCK, a node blocks on a single other node
  - *count* is incremented, *q_size* set appropriately



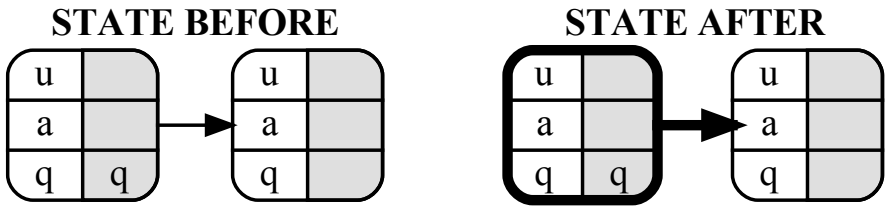
- TRANSMIT, a node's state travels upstream
  - If downstream state changes, it could propagate upward (minnn is minimum non-negative)



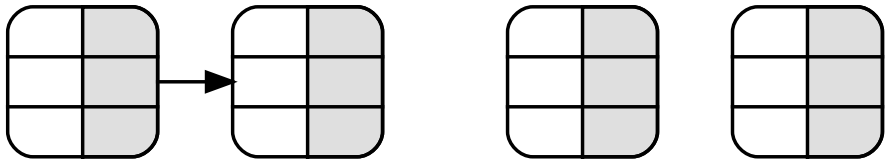


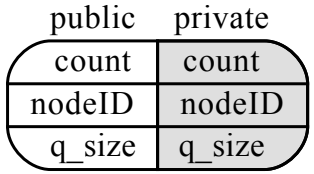
D4R State Transitions (2)

- DETECT, node's state has circuited a *wait-for* cycle
  - If q_size<0 then *real deadlock* -- a cycle of reads
  - Otherwise, blocked on queue that should grow

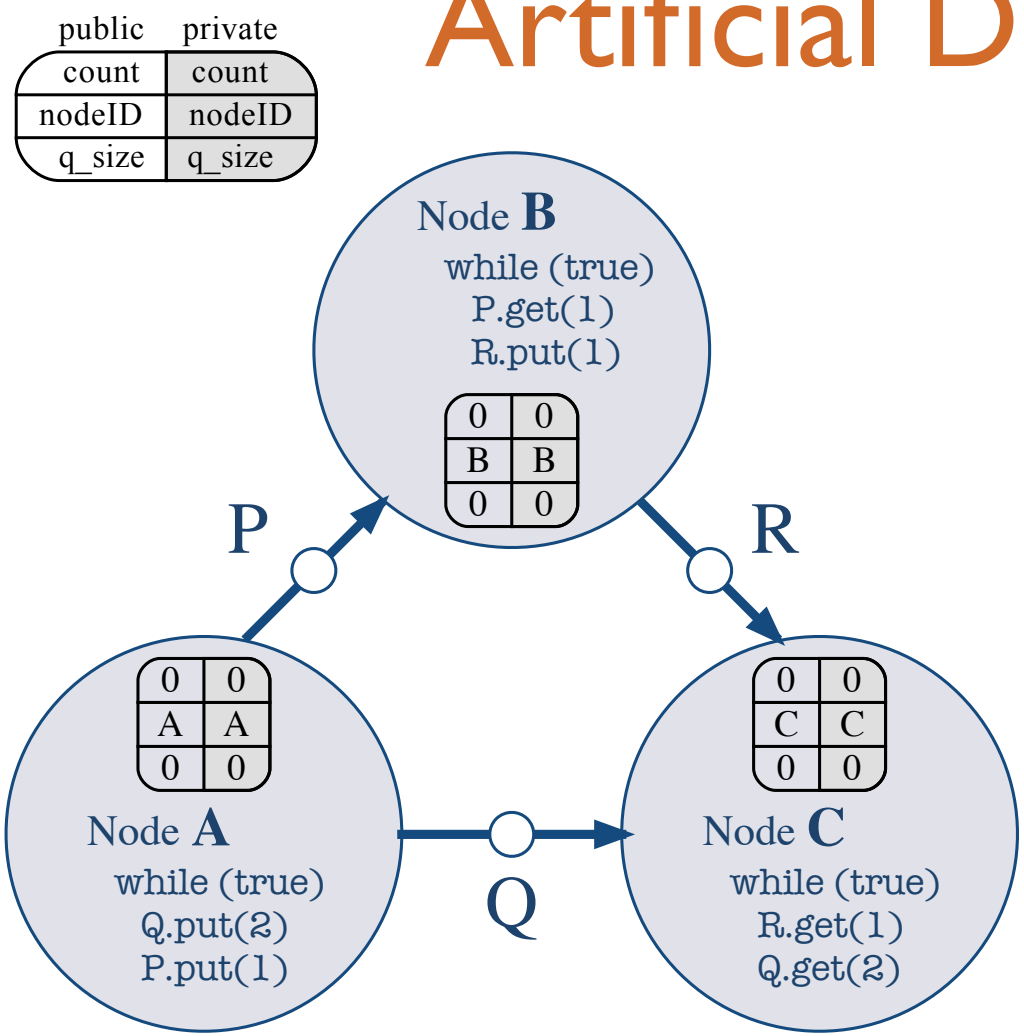


- ACTIVATE, resolve dependency and continue
  - Lengthen the queue to resolve *artificial deadlock*





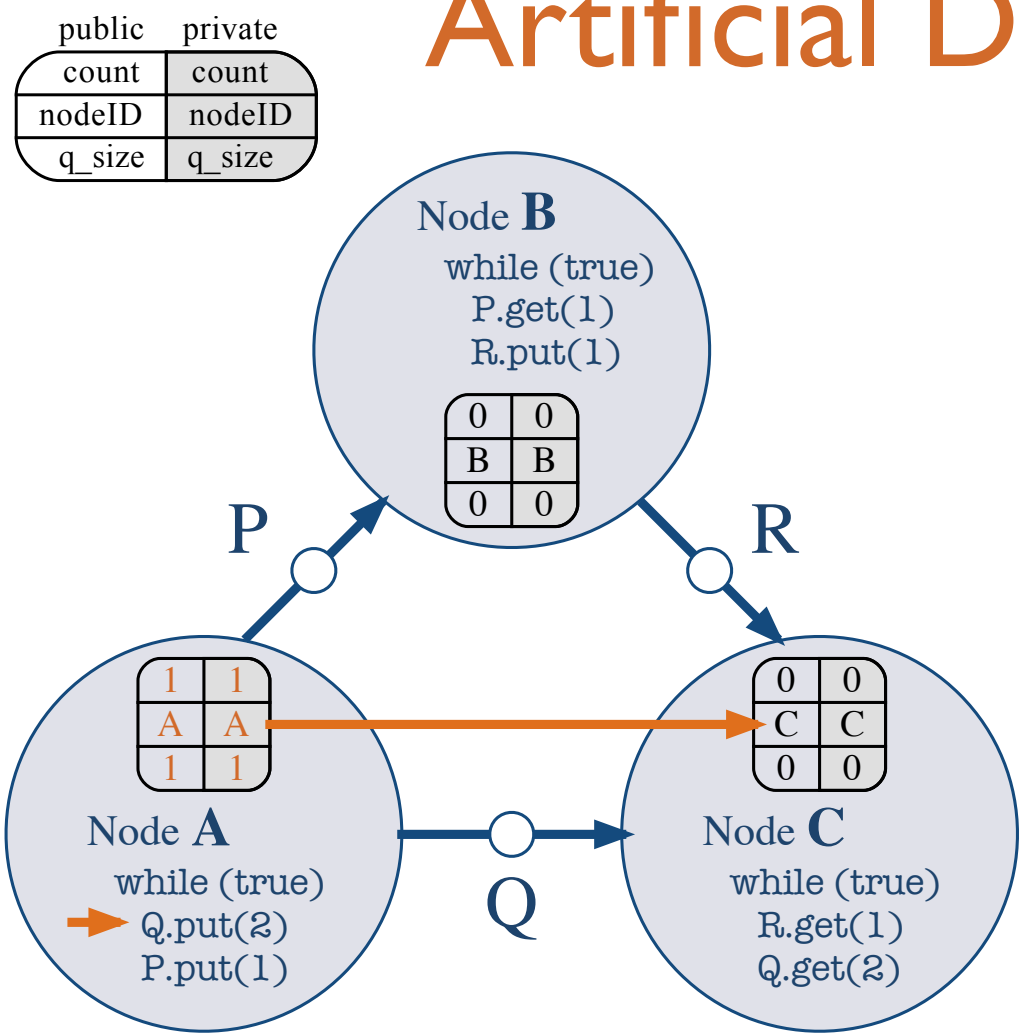
Example: Resolution of an Artificial Deadlock



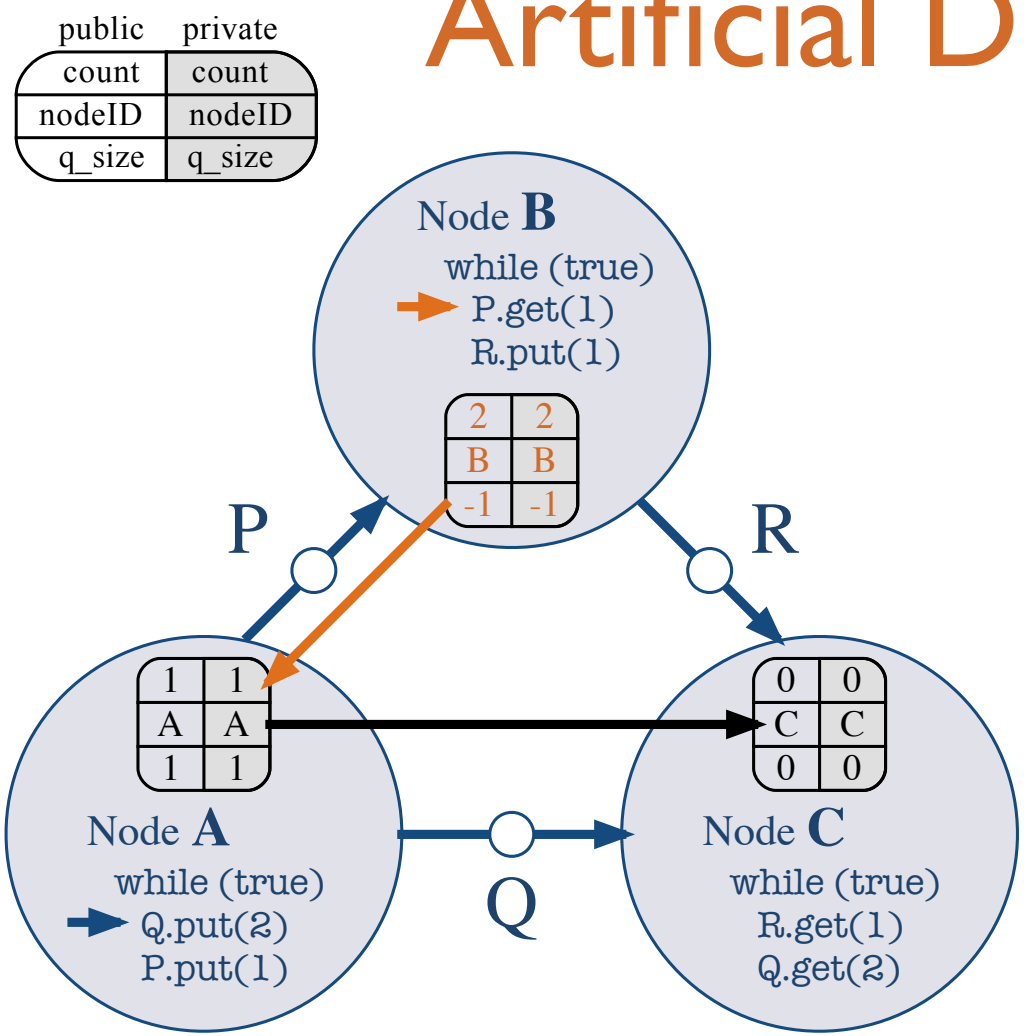
- A Bounded PN
- Initial conditions
  - All queues length 1
  - D4R states initialized
- Each node is an independent thread
- One of several possible orders of execution

Example: Resolution of an Artificial Deadlock

1. A *BLOCKS* on C

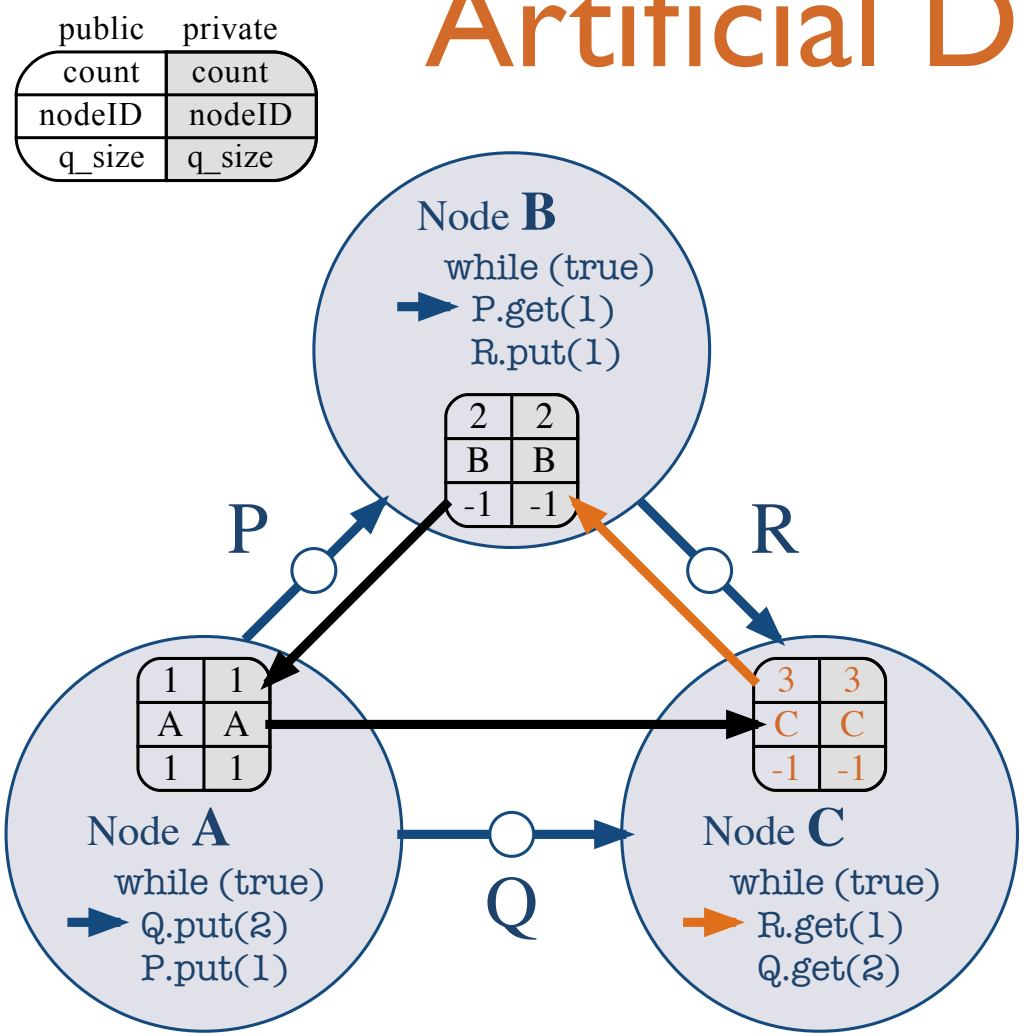


Example: Resolution of an Artificial Deadlock



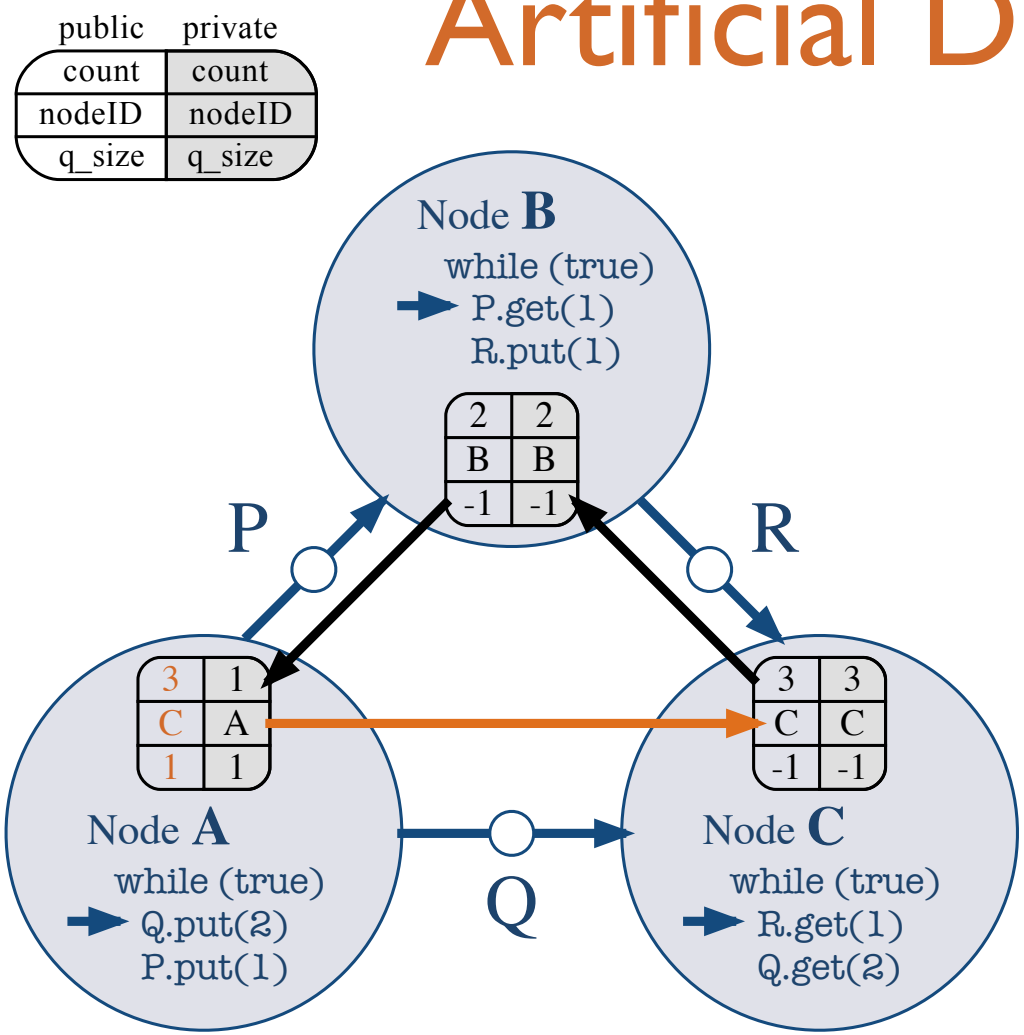
1. A *BLOCKS* on C
2. B *BLOCKS* on A

Example: Resolution of an Artificial Deadlock



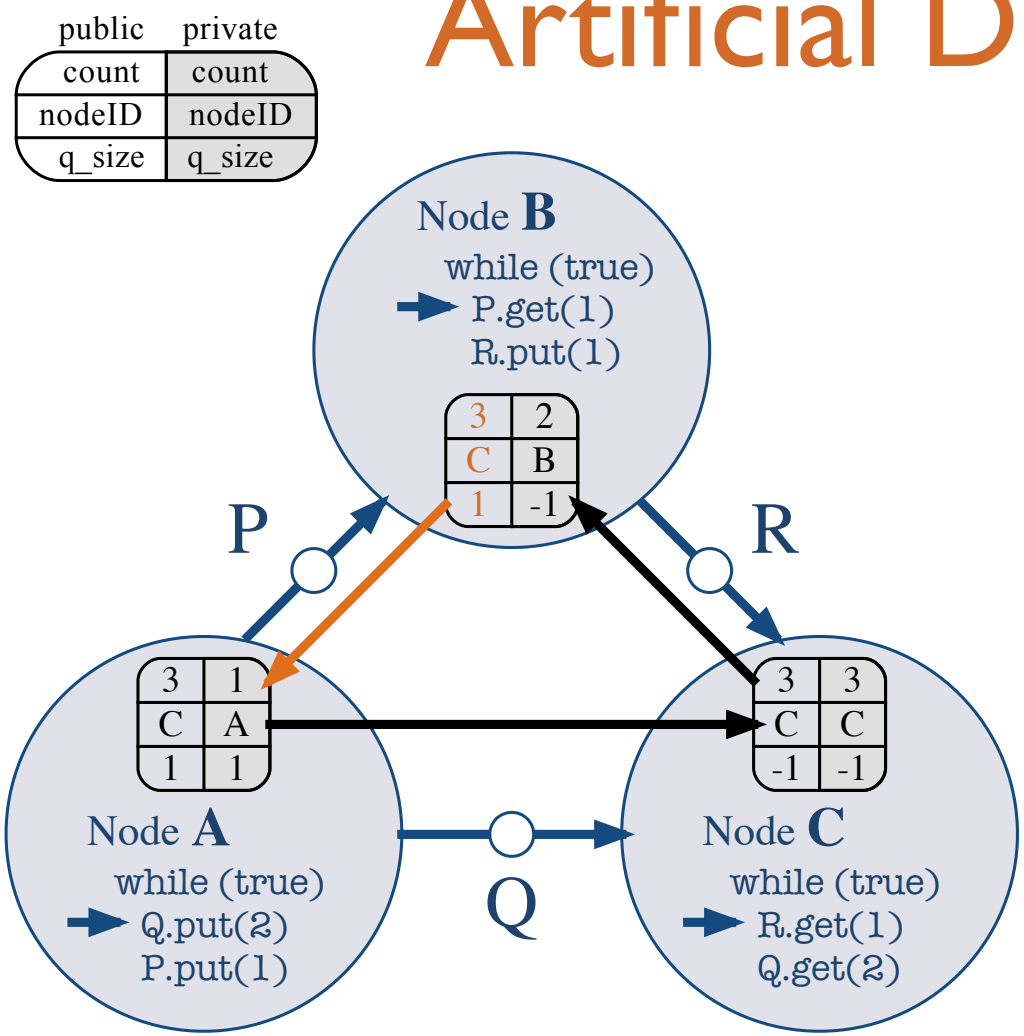
1. A *BLOCKS* on C
2. B *BLOCKS* on A
3. C *BLOCKS* on B

Example: Resolution of an Artificial Deadlock



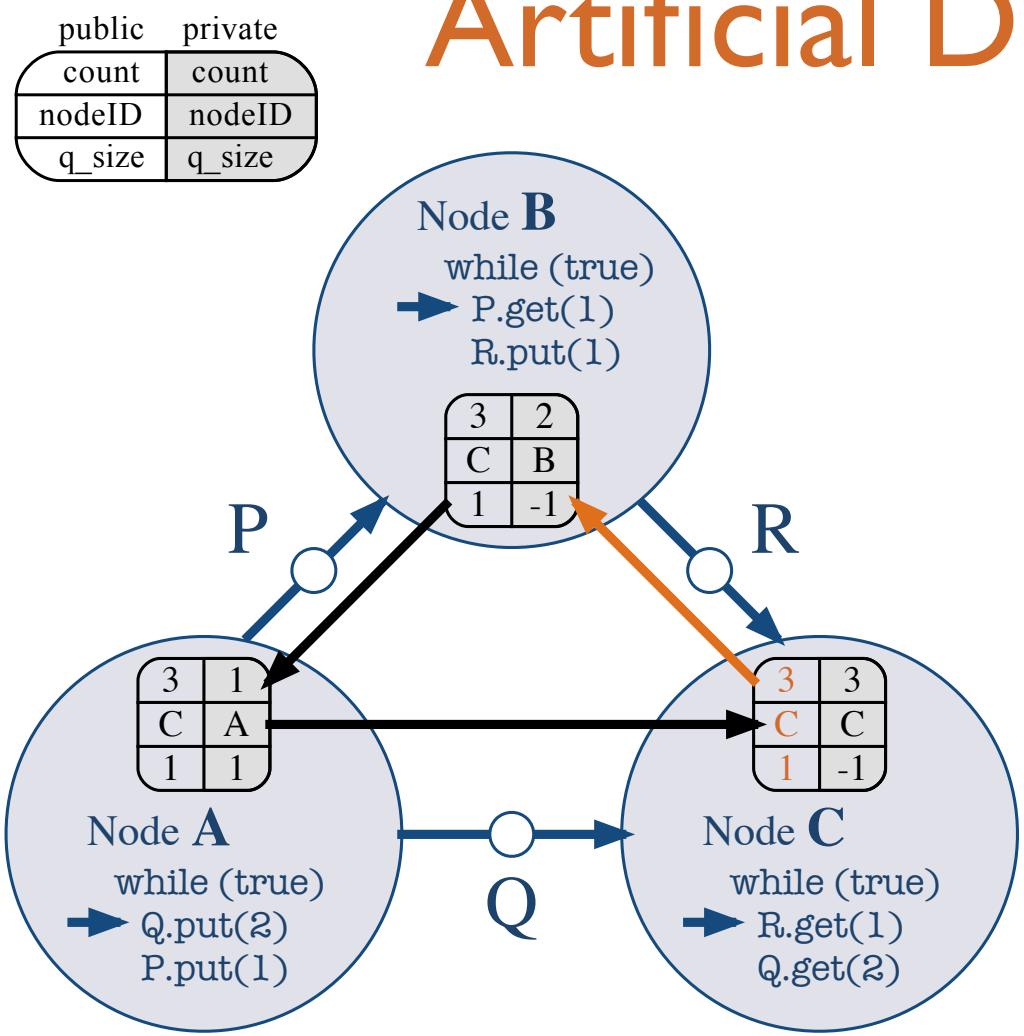
1. A *BLOCKS* on C
2. B *BLOCKS* on A
3. C *BLOCKS* on B
4. A gets *TRANSMIT* from C

Example: Resolution of an Artificial Deadlock



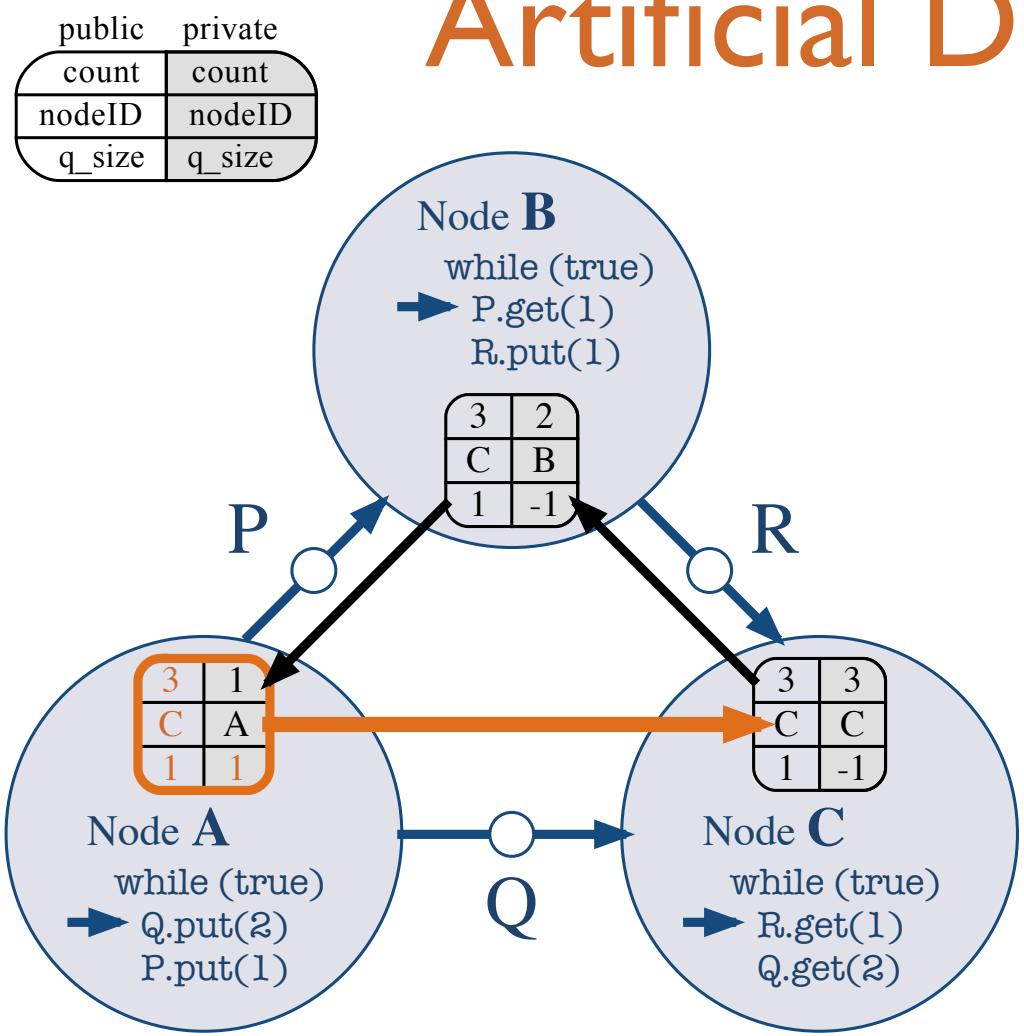
1. A *BLOCKS* on C
2. B *BLOCKS* on A
3. C *BLOCKS* on B
4. A gets *TRANSMIT* from C
5. B gets *TRANSMIT* from A

Example: Resolution of an Artificial Deadlock



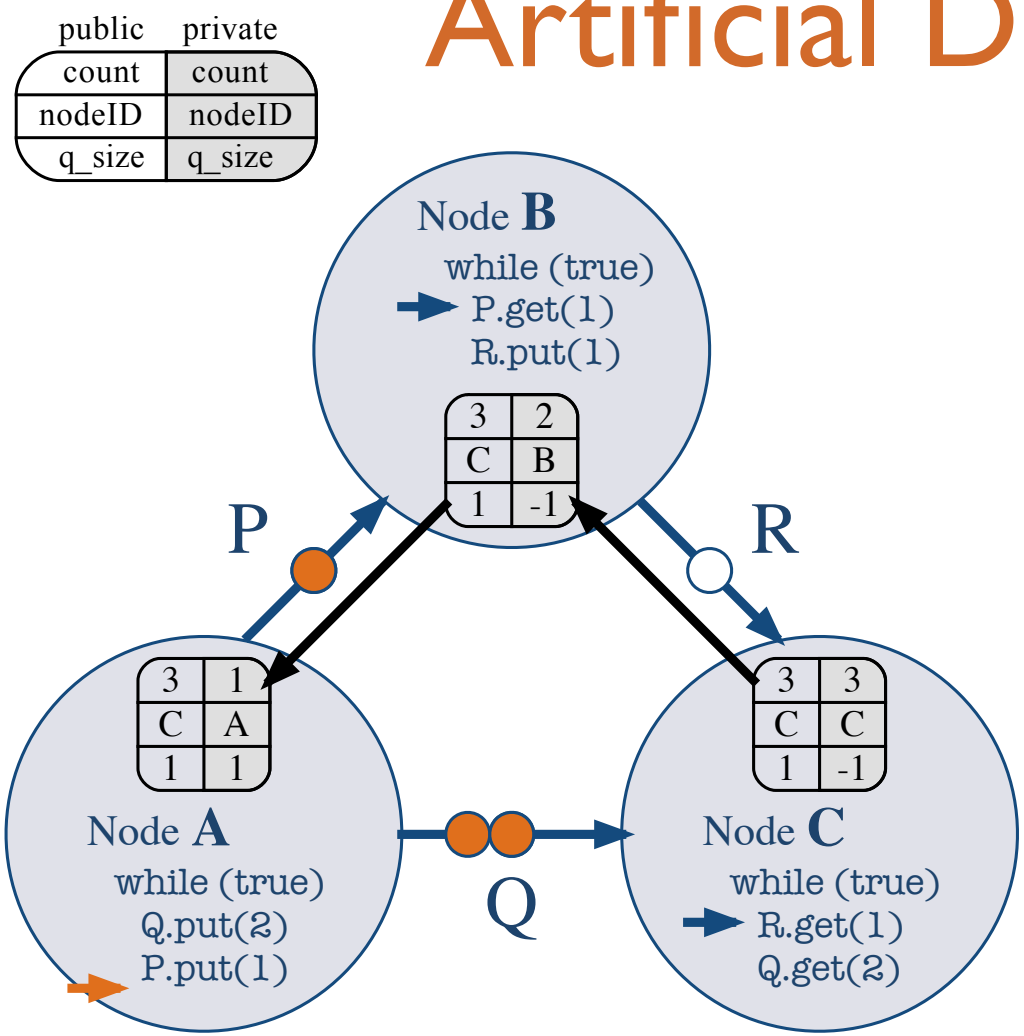
1. A *BLOCKS* on C
2. B *BLOCKS* on A
3. C *BLOCKS* on B
4. A gets *TRANSMIT* from C
5. B gets *TRANSMIT* from A
6. C gets *TRANSMIT* from B

Example: Resolution of an Artificial Deadlock



1. A *BLOCKS* on C
2. B *BLOCKS* on A
3. C *BLOCKS* on B
4. A gets *TRANSMIT* from C
5. B gets *TRANSMIT* from A
6. C gets *TRANSMIT* from B
7. A *DETECTS* deadlock

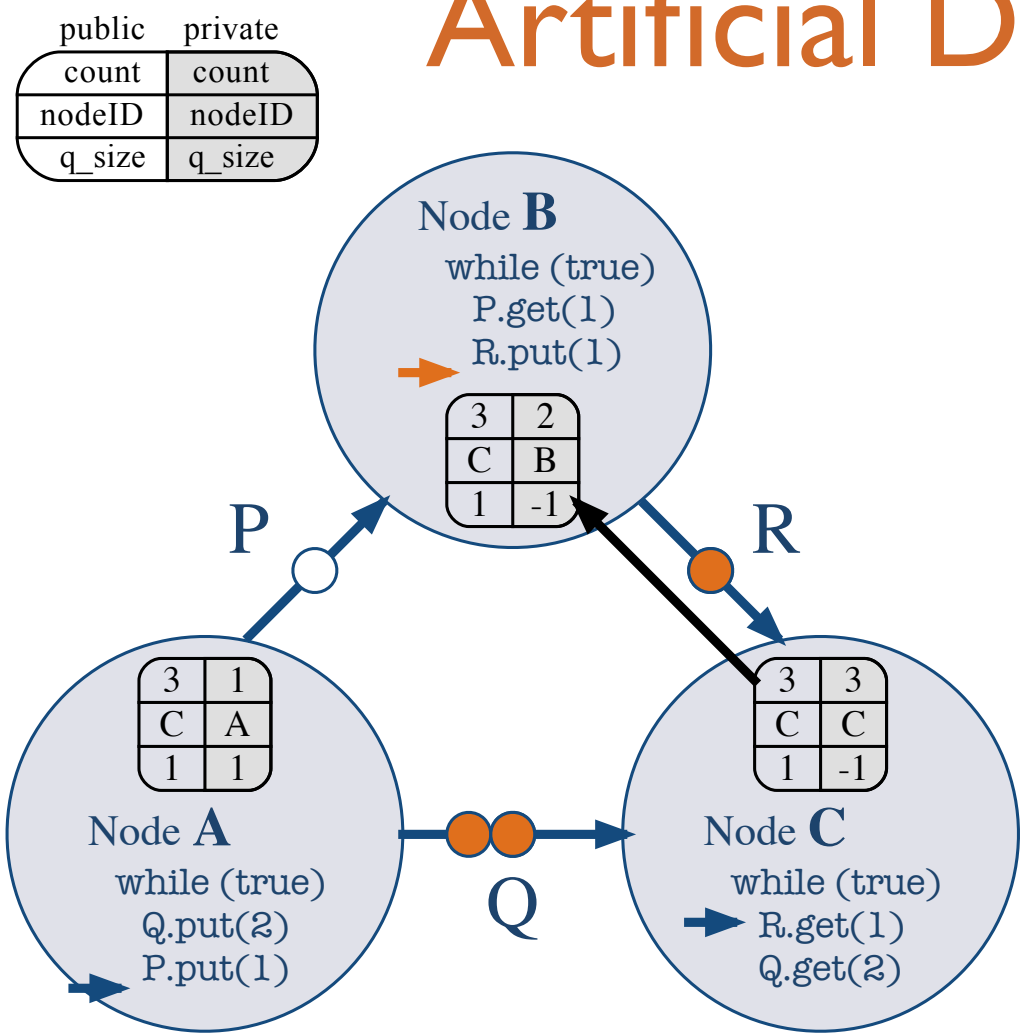
Example: Resolution of an Artificial Deadlock



1. A *BLOCKS* on C
2. B *BLOCKS* on A
3. C *BLOCKS* on B
4. A gets *TRANSMIT* from C
5. B gets *TRANSMIT* from A
6. C gets *TRANSMIT* from B
7. A *DETECTS* deadlock
8. A *ACTIVATES* to continue

   (Grow queue Q to 2)

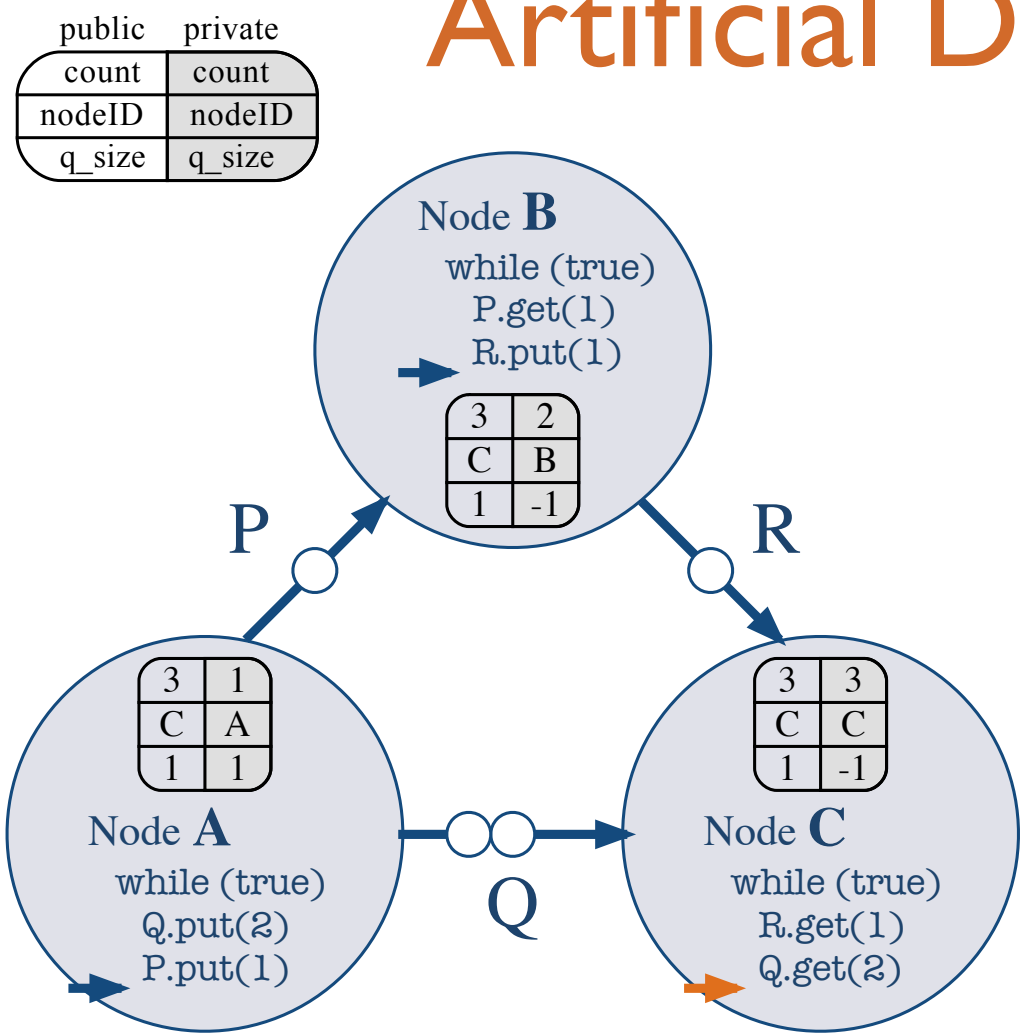
Example: Resolution of an Artificial Deadlock



1. A *BLOCKS* on C
2. B *BLOCKS* on A
3. C *BLOCKS* on B
4. A gets *TRANSMIT* from C
5. B gets *TRANSMIT* from A
6. C gets *TRANSMIT* from B
7. A *DETECTS* deadlock
8. A *ACTIVATES* to continue
9. B *ACTIVATES* to continue

   (dependency resolved)

Example: Resolution of an Artificial Deadlock



1. A *BLOCKS* on C
2. B *BLOCKS* on A
3. C *BLOCKS* on B
4. A gets *TRANSMIT* from C
5. B gets *TRANSMIT* from A
6. C gets *TRANSMIT* from B
7. A *DETECTS* deadlock
8. A *ACTIVATES* to continue
9. B *ACTIVATES*, continues
10. C *ACTIVATES*, continues

Comments

- *Wait-for* arcs coincide with the PN queues
- Larger *counts* and smaller *q_sizes* migrate along the *wait-for* graph in the opposite direction
- Exactly one node *DETECTs* a deadlock in N-1 to 2N-1 *TRANSMIT* steps (where N is number of nodes in cycle)
- Proofs provided in paper, based on [Mitchell & Merritt 84]
- Implementation provided as part of CPN library: http://www.ece.utexas.edu/~allen/CPN
- D4R algorithm performance is not a priority -- *artificial deadlock* is an exceptional condition

Conclusion

- Formal models like Process Networks can simplify development of complex, distributed DSP systems
- Execution in bounded memory requires dynamic deadlock detection and resolution
- Leveraged existing [Mitchell & Merritt 84] distributed algorithm for deadlock detection and resolution
- Provided a Distributed Dynamic Deadlock Detection and Resolution algorithm (D4R) to permit execution of PN in bounded memory
- Permits scalable implementation of bounded PN