

A Provably Starvation-free Distributed Directory Protocol

Hagit Attiya
Vincent Gramoli
Alessia Milani

Context

Distributed Directory Protocol

Context

Distributed Directory Protocol

- Directory protocol: abstraction providing **lookup** and **move** requests

Context

Distributed Directory Protocol

- Directory protocol: abstraction providing **lookup** and **move** requests
- Distributed: clients can issue requests **concurrently** to the same object

Context

Distributed Directory Protocol

- Directory protocol: abstraction providing **lookup** and **move** requests
- Distributed: clients can issue requests **concurrently** to the same object
- Asynchronous: request messages can take an **arbitrary** amount of time

Motivation

- A Distributed Directory Protocol (DDP) implements a request queue used in **Distributed Transactional Memory (DTM)**

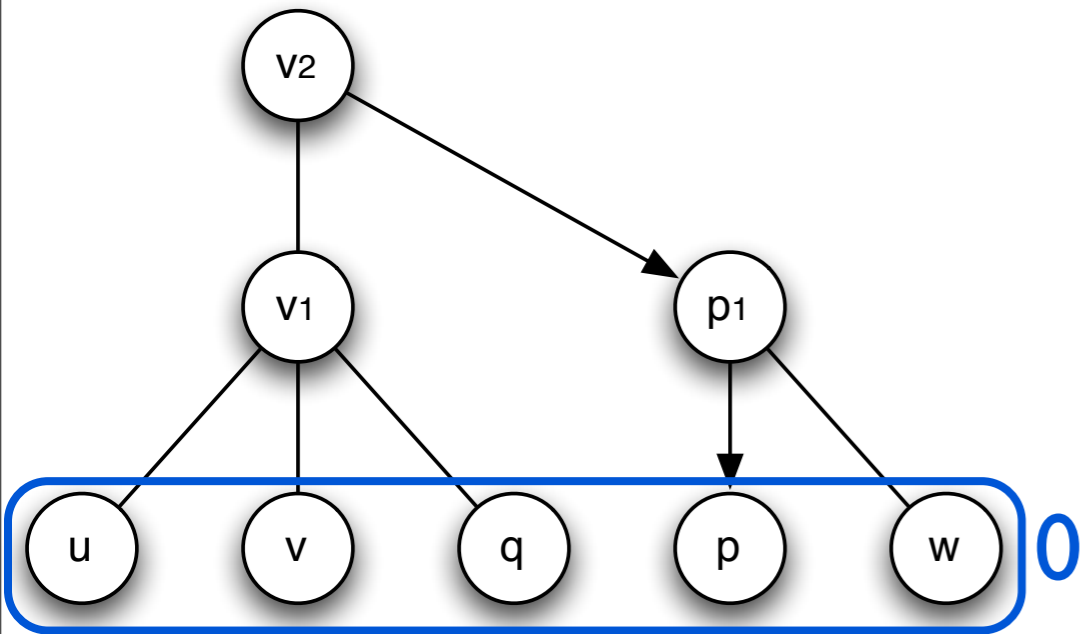
Motivation

- A Distributed Directory Protocol (DDP) implements a request queue used in **Distributed Transactional Memory (DTM)**
- DDP move \Leftrightarrow DTM write
- DDP lookup \Leftrightarrow DTM read

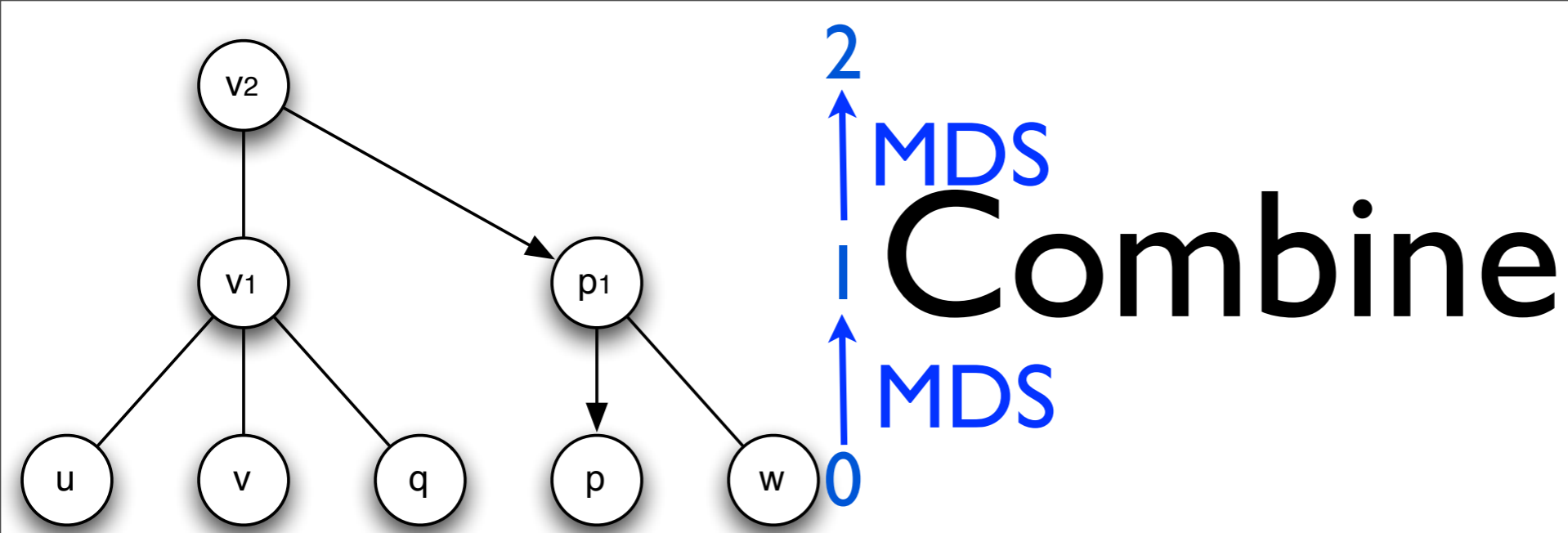
Model

- Tree **overlay** with **reliable** nodes
 - metric distance $d(p,q)$, diameter D
- Nodes receive-compute-send **atomically**
- Nodes are **uniquely** identified
- Node i can **send to j if it knows j**
- **One node** executes **one request** at a time

Combine

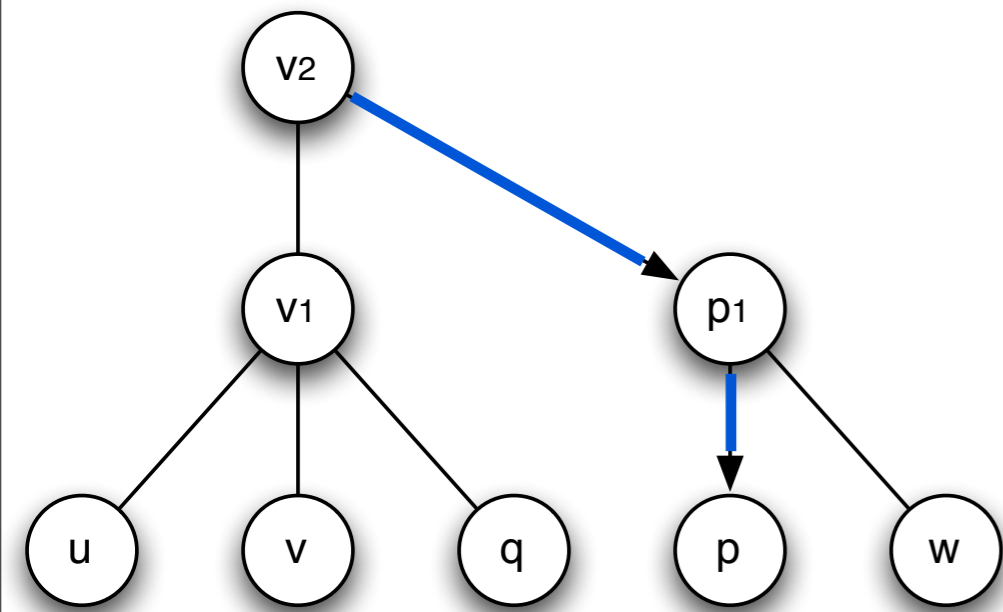


- Overlay tree (consider a single object)
- **leaves** (at level 0) are **physical nodes**



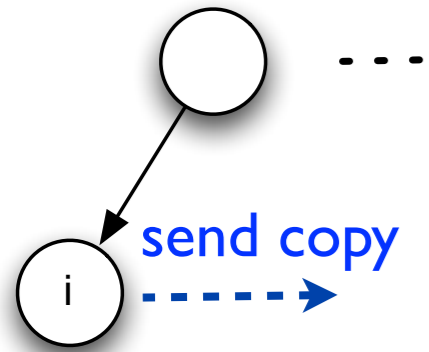
- Overlay tree (consider a single object)
 - **leaves** (at level 0) are **physical nodes**
 - built using recursive **minimal dominated sets (MDS)** or **spanning tree**

Combine



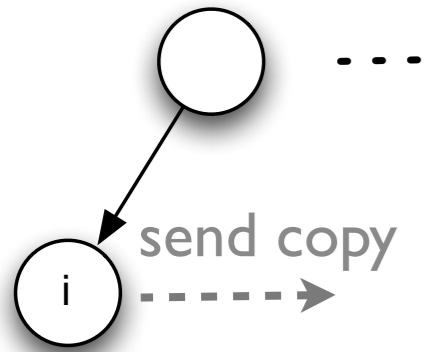
- Overlay tree (consider a single object)
 - leaves (at level 0) are physical nodes
 - built using recursive minimal dominated sets (MDS) or spanning tree
 - Initially, nodes point downward to (next) object owner
 - nodes know their parent in the tree

Lookup of x at i

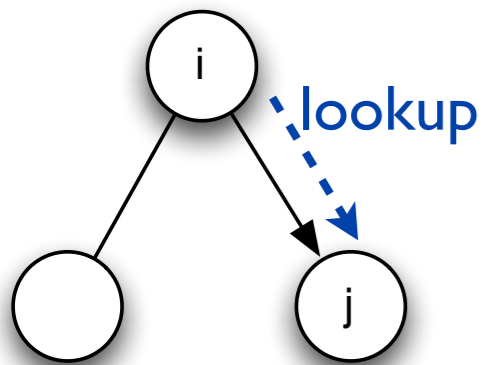


- if i owns x , i sends a copy to requester

Lookup of x at i

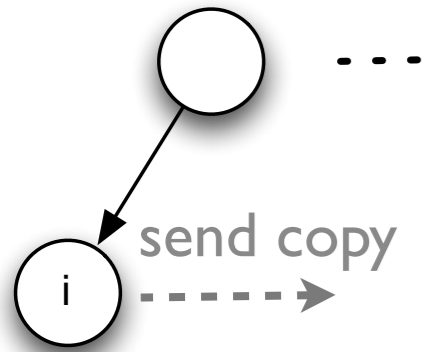


- if i owns x , i sends a copy to requester

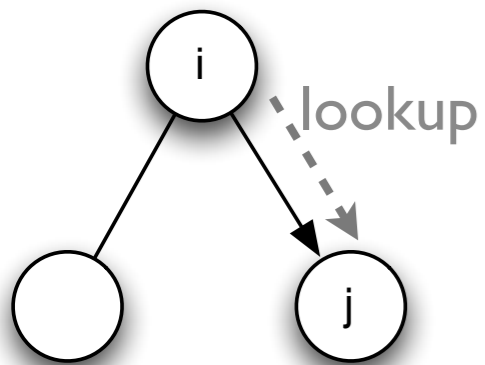


- if $i.pointer$, i calls lookup of x at $i.pointer$

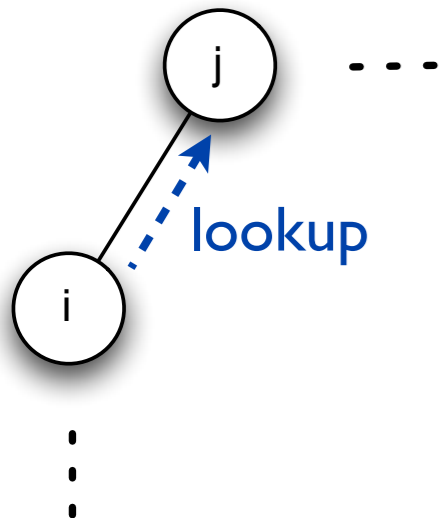
Lookup of x at i



- if i owns x , i sends a copy to requester



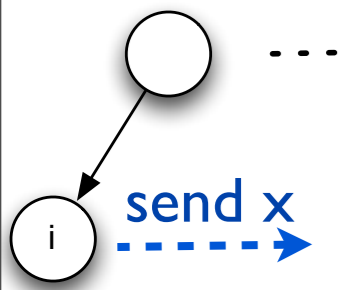
- if $i.pointer$, i calls lookup of x at $i.pointer$



- else, i calls lookup of x at $i.parent$

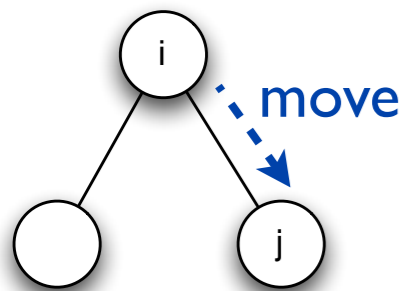
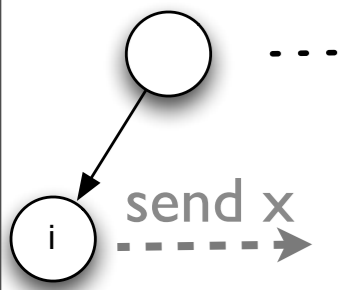
Move of x at i

- similar to lookup x at i
- except that i
- at the end, the object is sent (no copy)



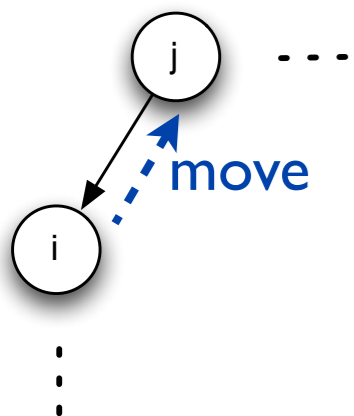
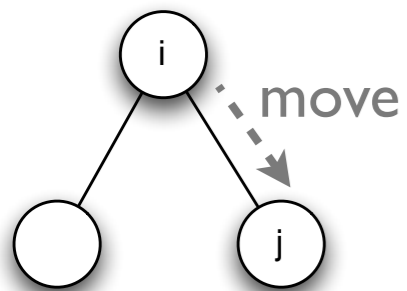
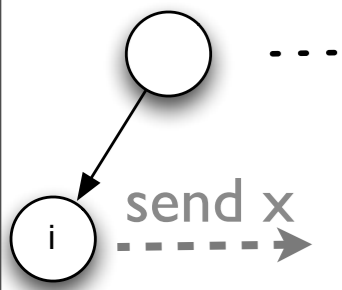
Move of x at i

- similar to lookup x at i
- except that i
- at the end, the object is sent (no copy)
- discards path while going down



Move of x at i

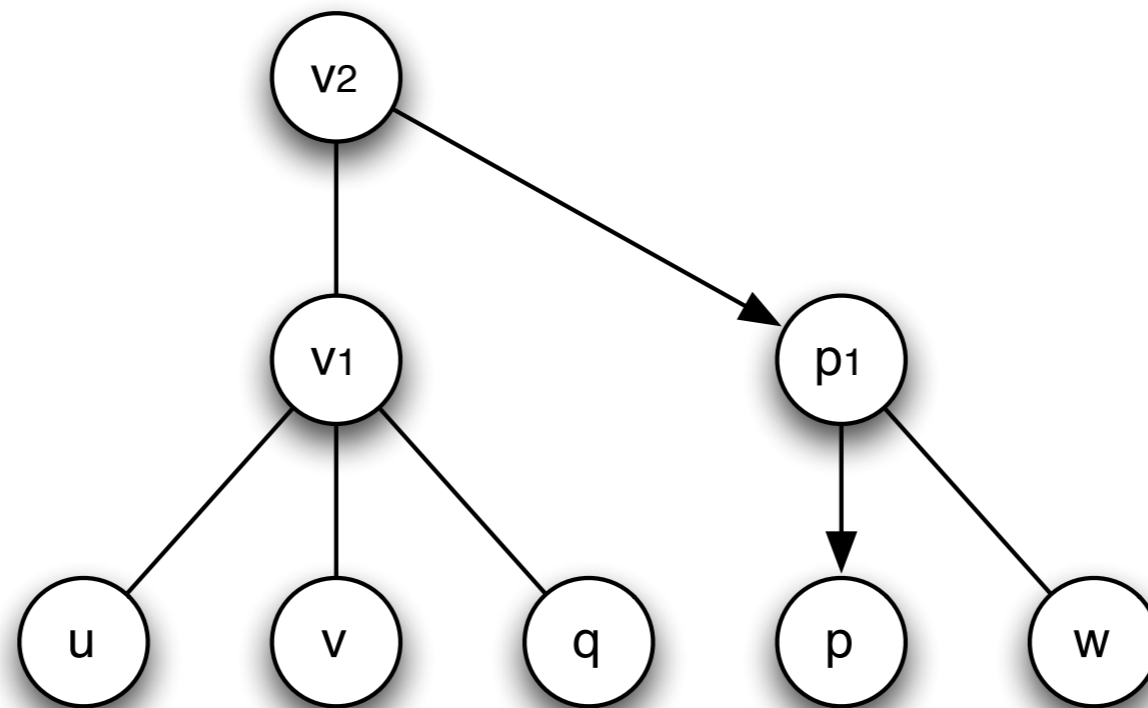
- similar to lookup x at i
- except that i
- at the end, the object is sent (no copy)



- discards path while going down
- sets downward path while going up

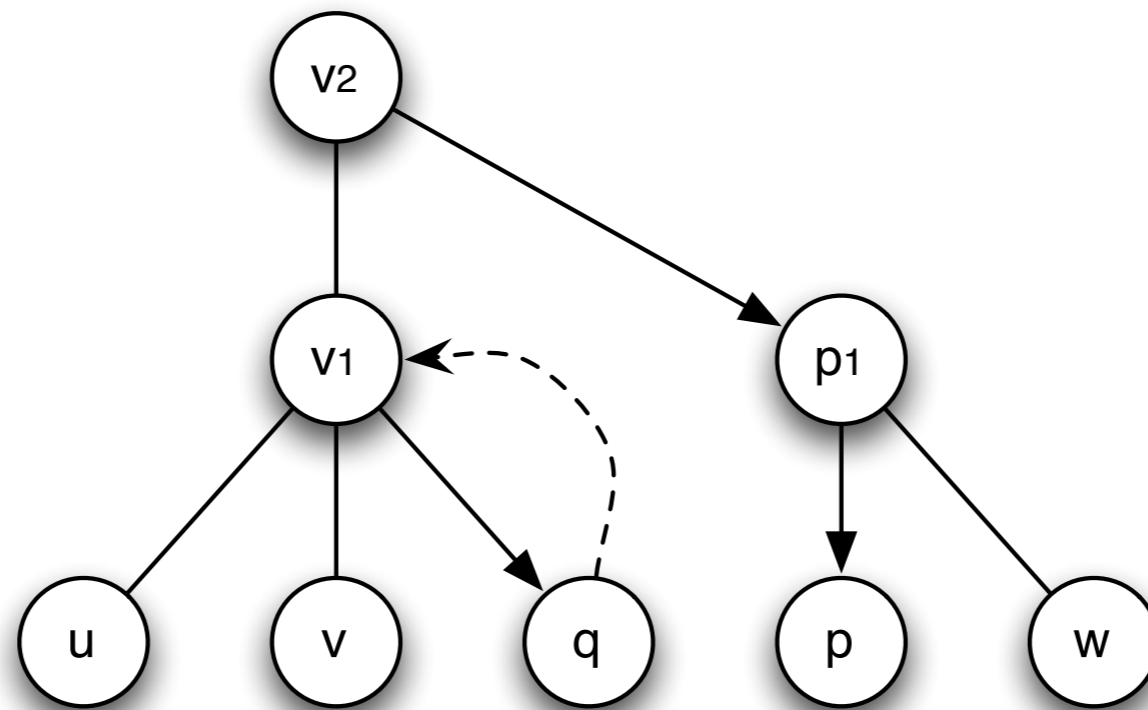
Example

- Initially, p owns the object



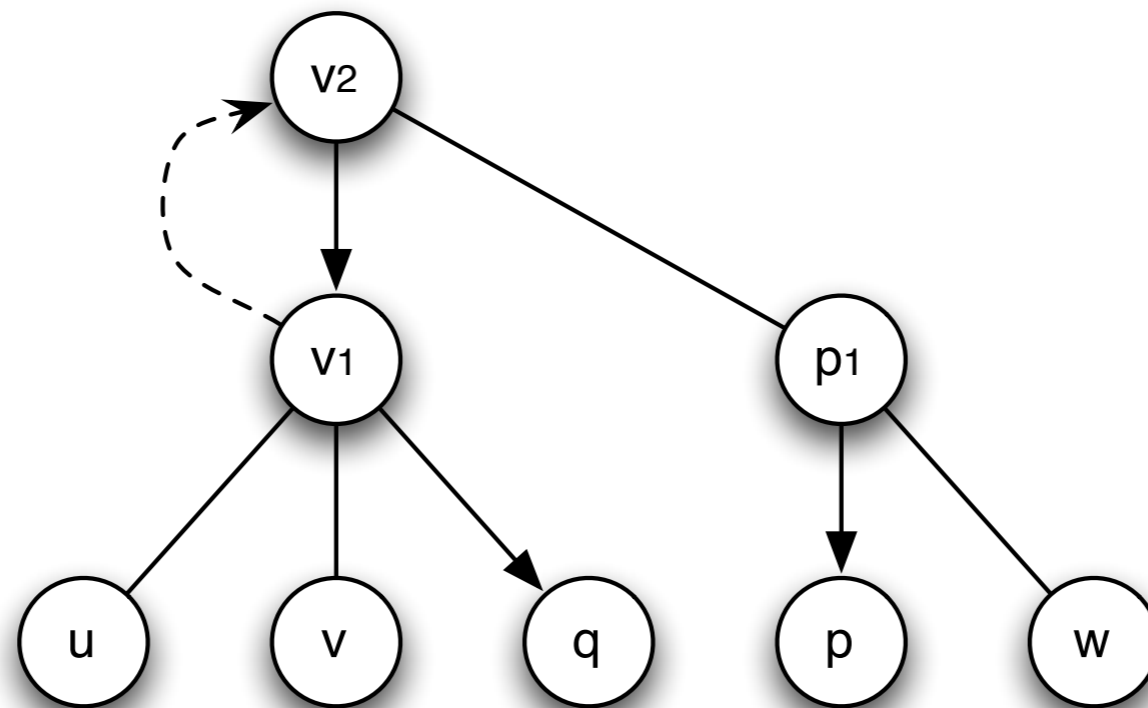
Example

- q initiates a move request by calling `move` at $q.parent = v_1$ that sets its pointer



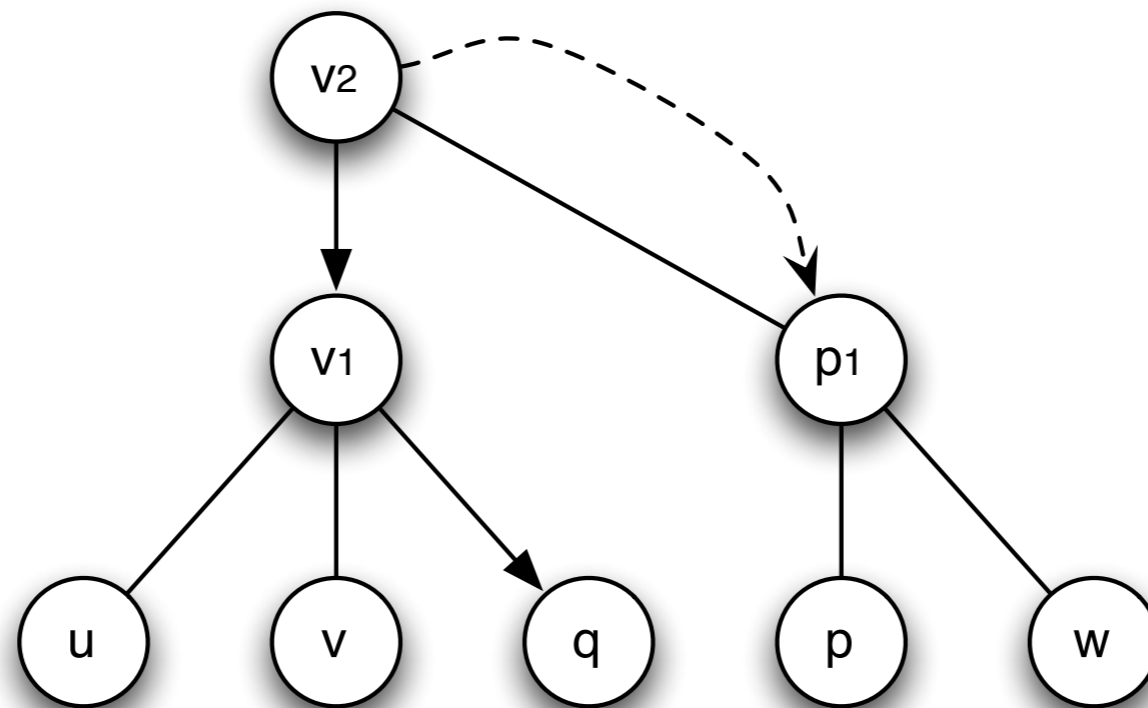
Example

- v_1 calls `move` at $v_1.parent = v_2$ that swaps its pointer



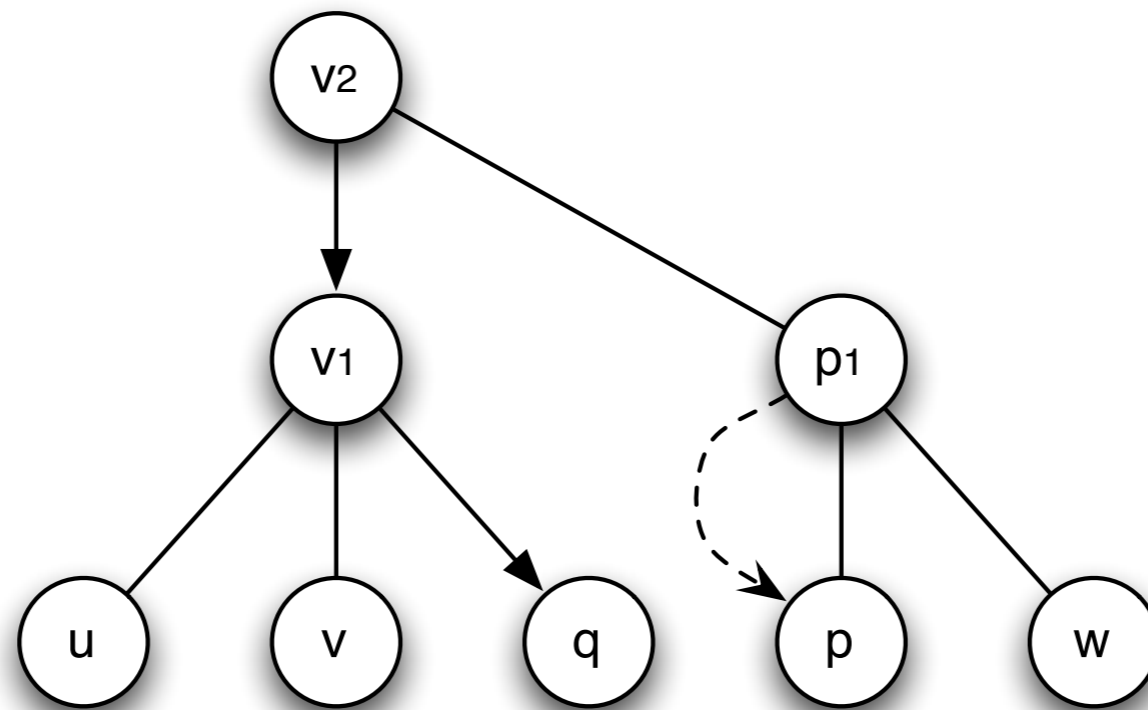
Example

- v_2 calls `move` at $v_2.pointer = p_1$



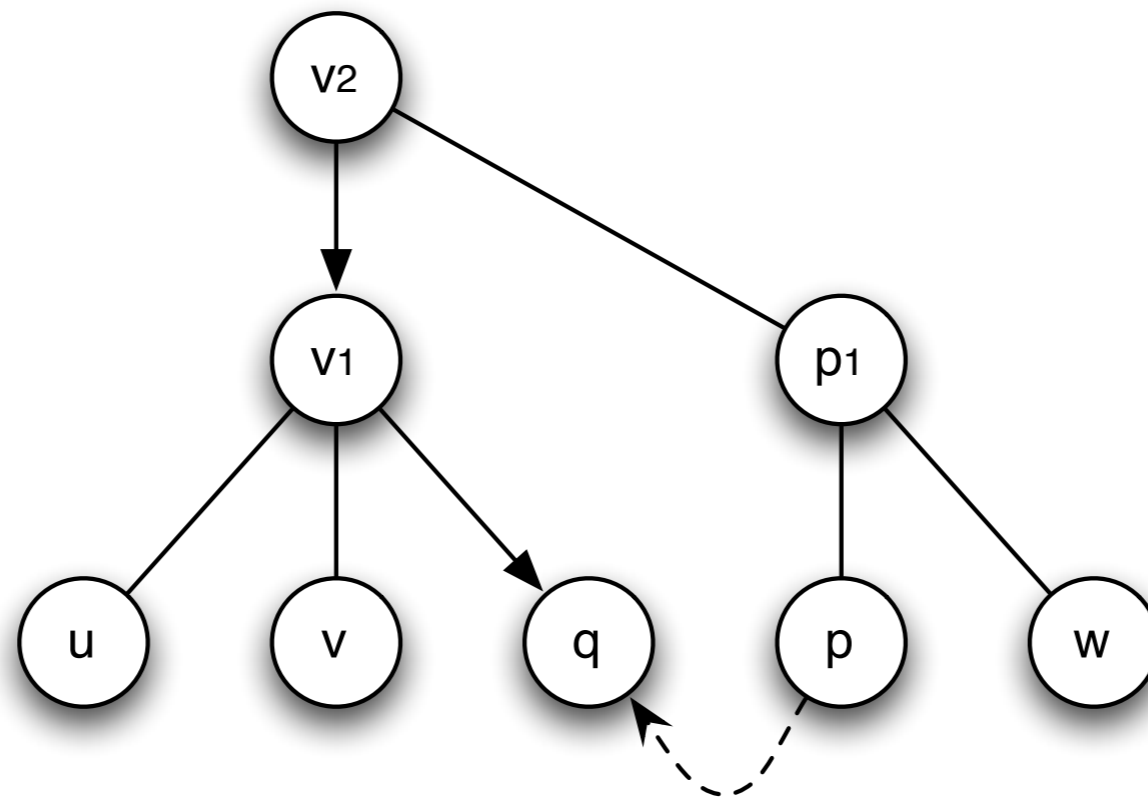
Example

- p_1 calls `move` at $p_1.pointer = p$ and discards its pointer

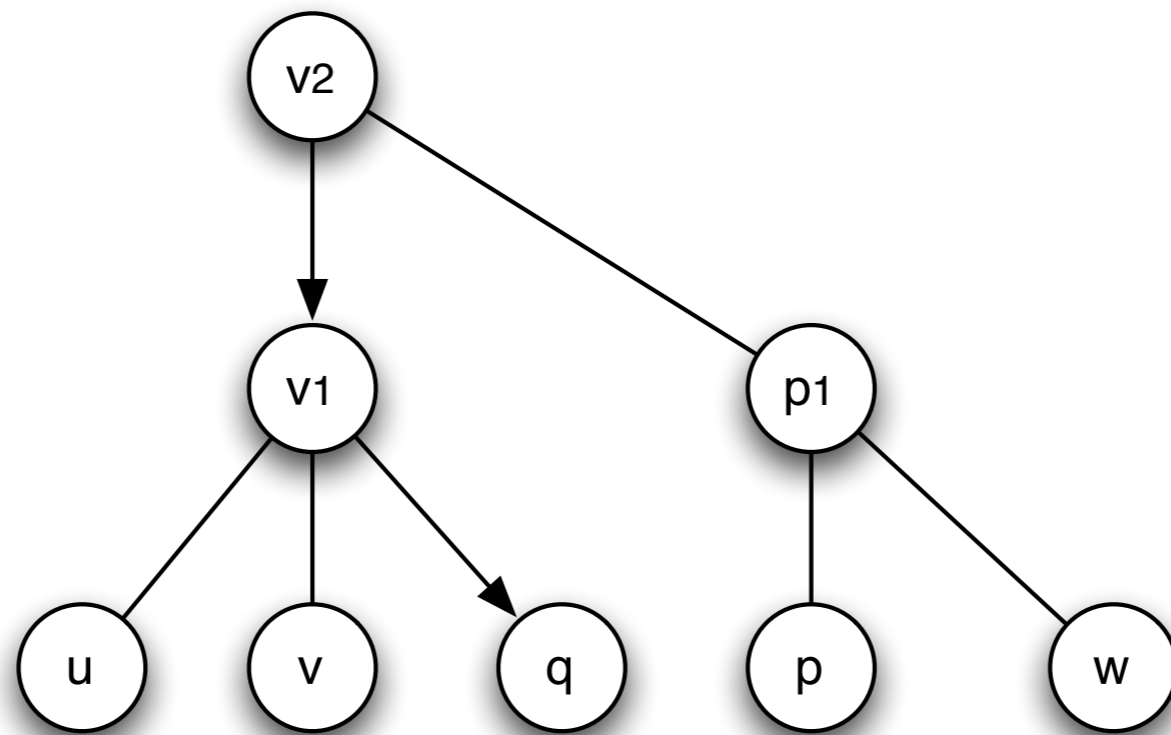
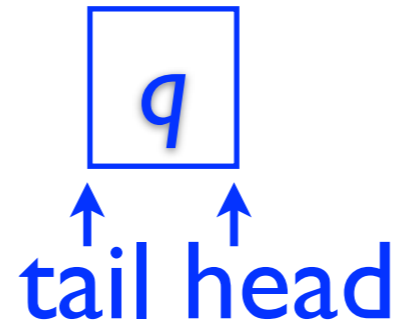


Example

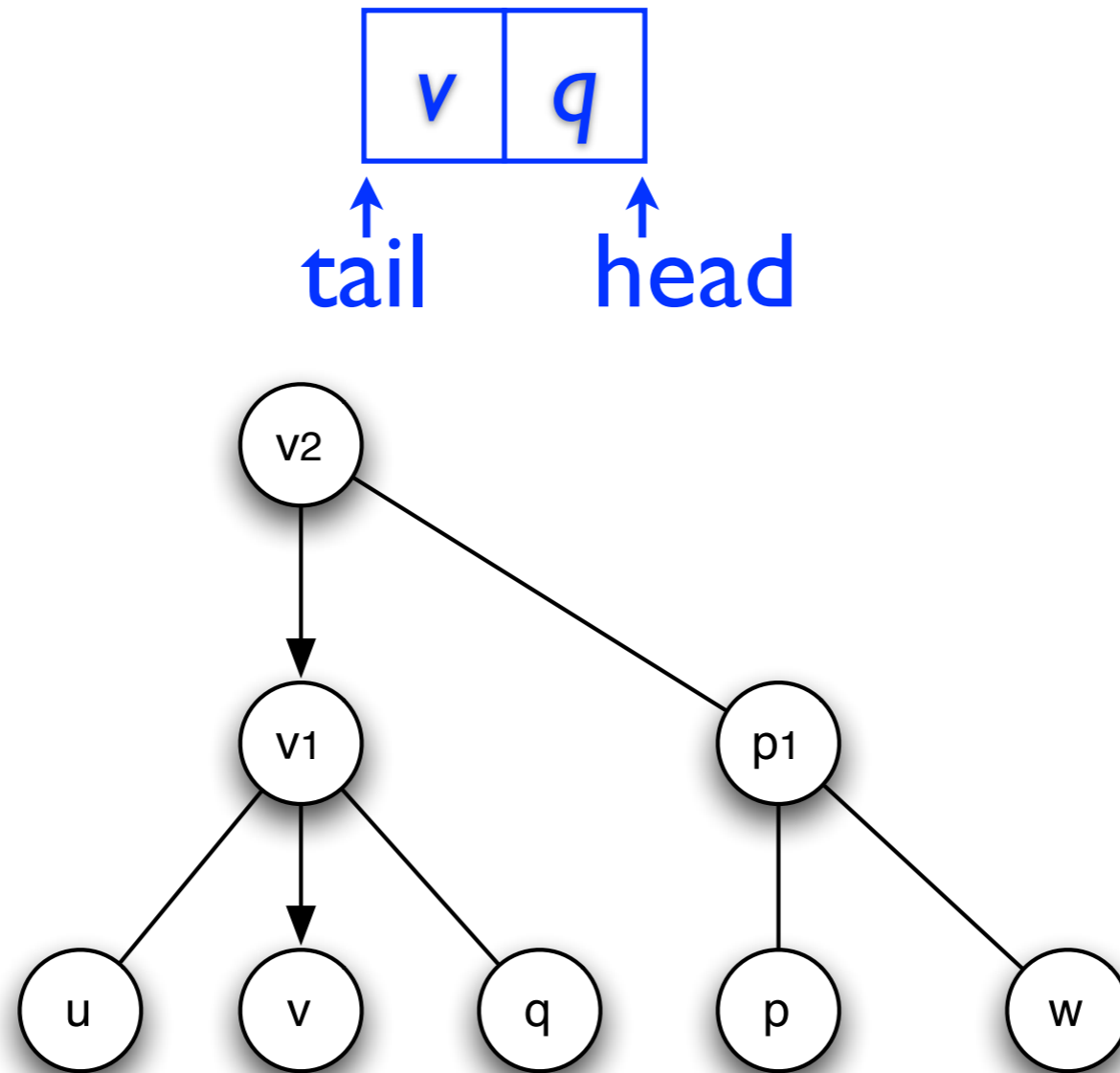
- p sends the object to the initiator q



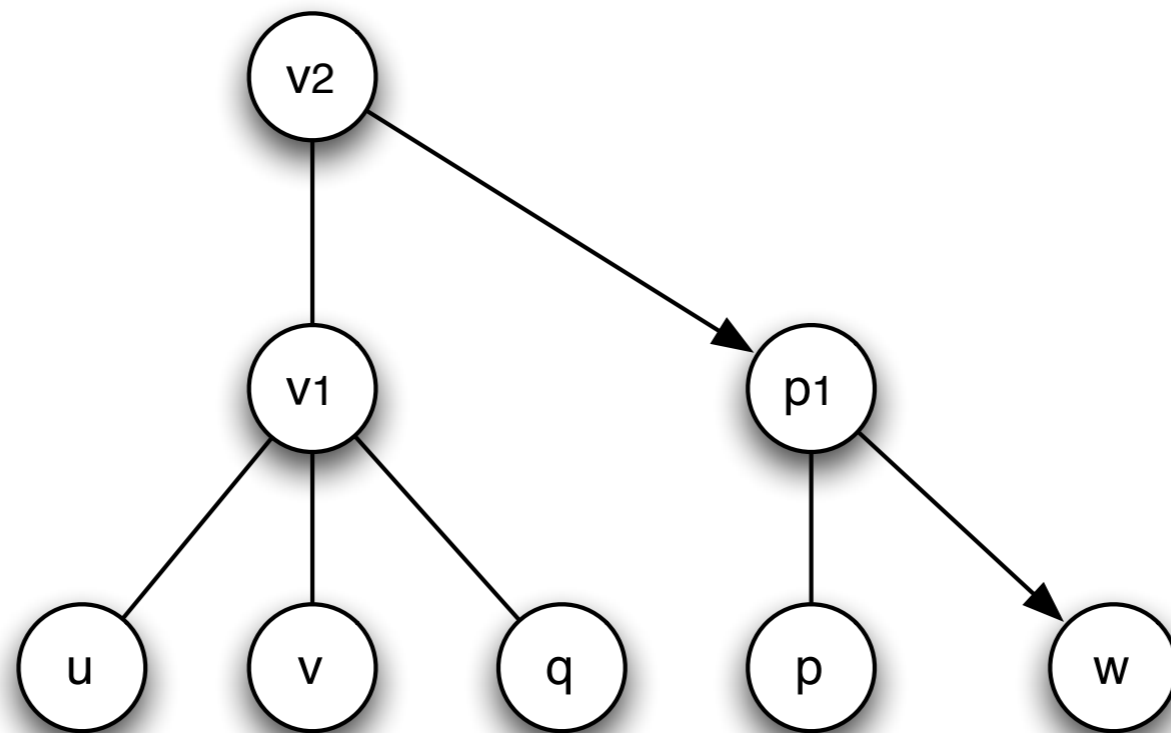
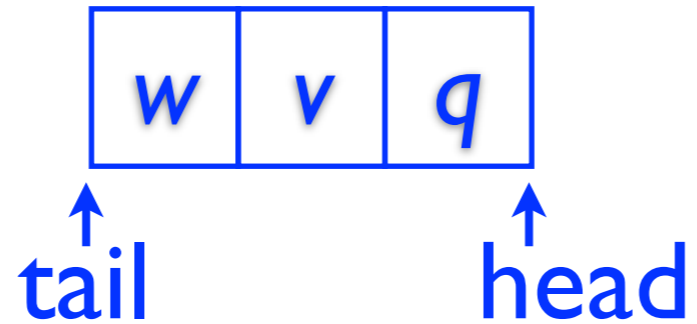
Queuing Analogy



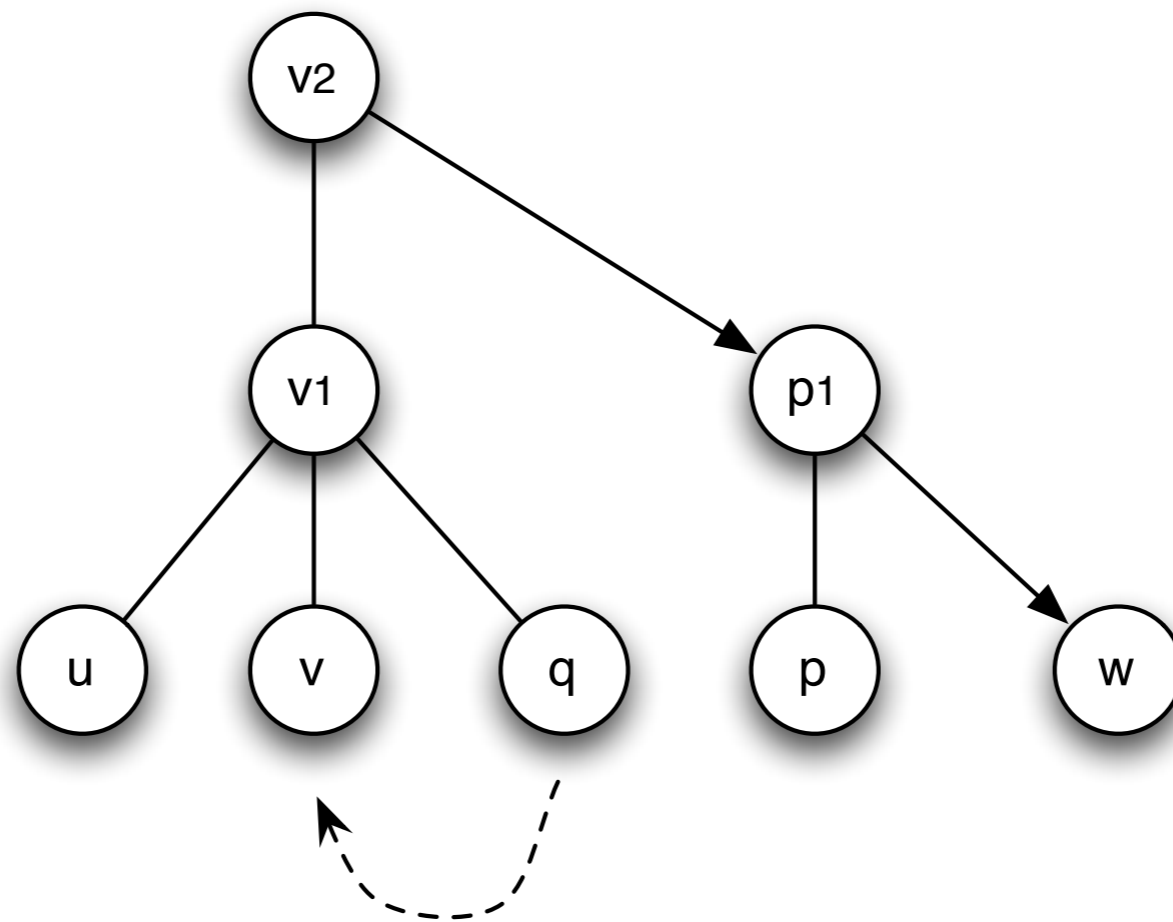
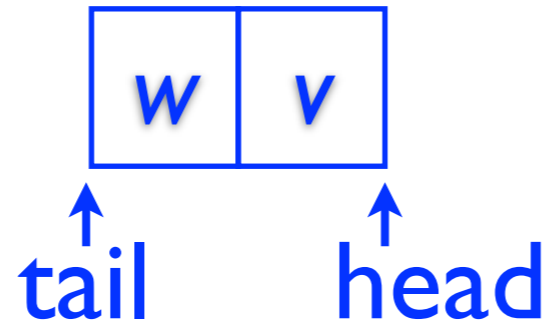
Queuing Analogy



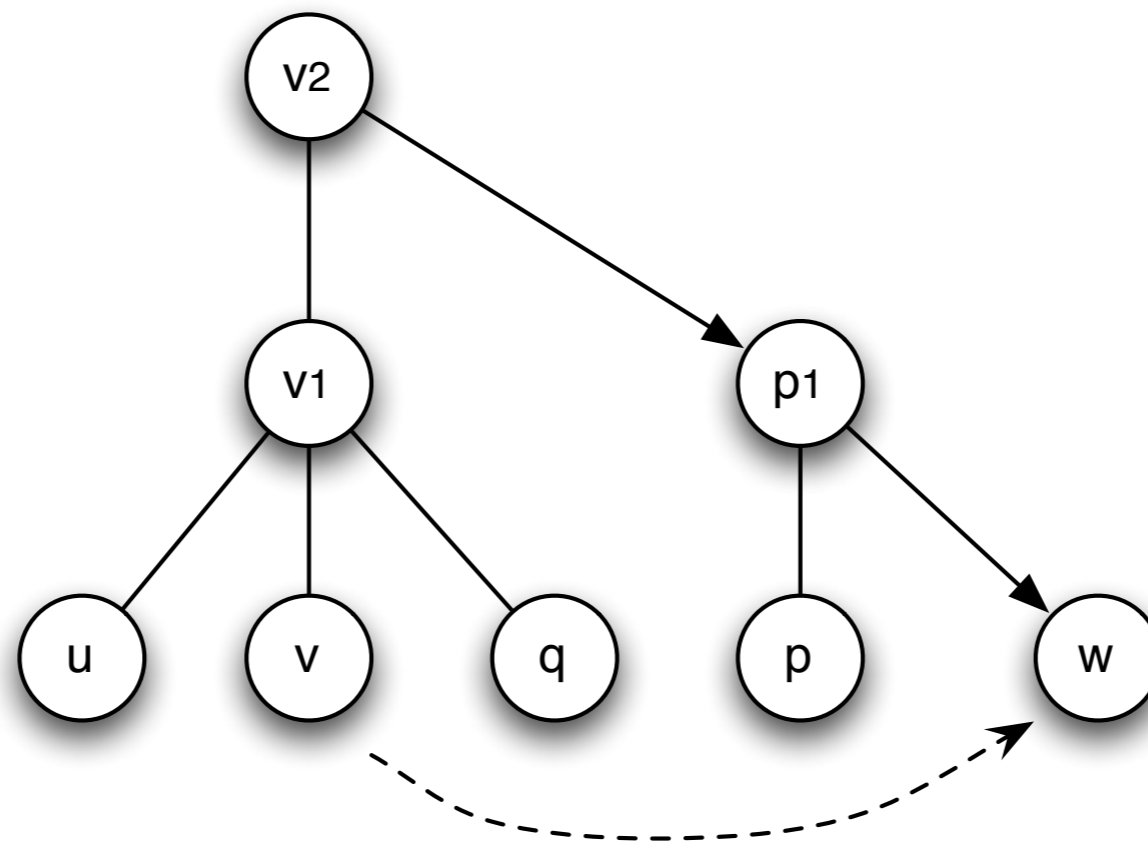
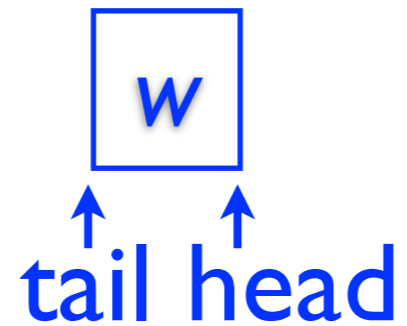
Queuing Analogy



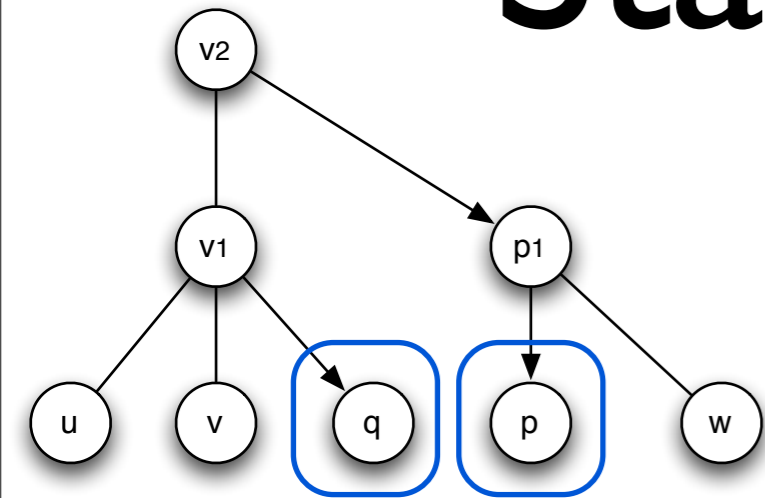
Queuing Analogy



Queuing Analogy



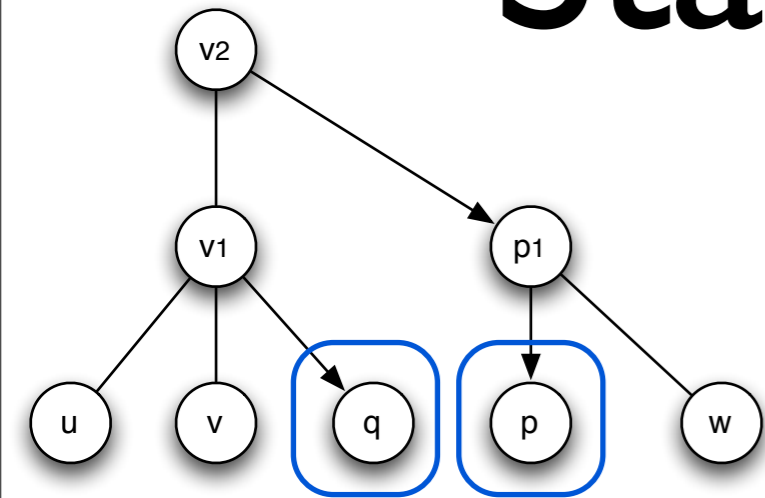
Starvation Problem



- Initially p owns x while q requests it

p

Starvation Problem

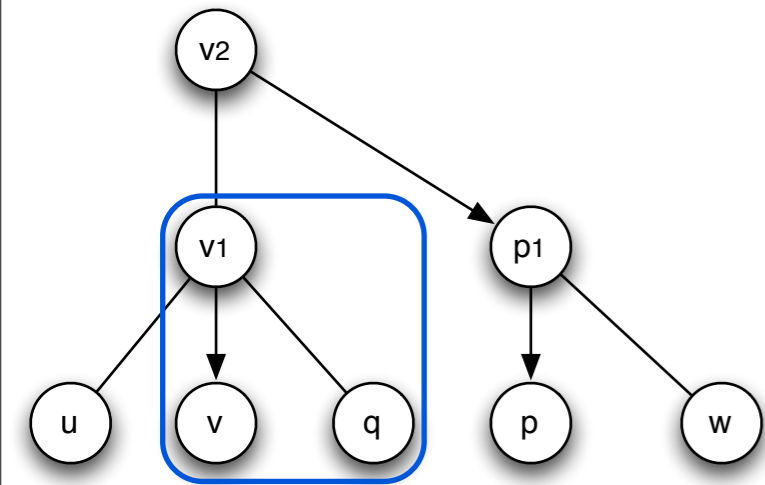


- Initially p owns x while q requests it

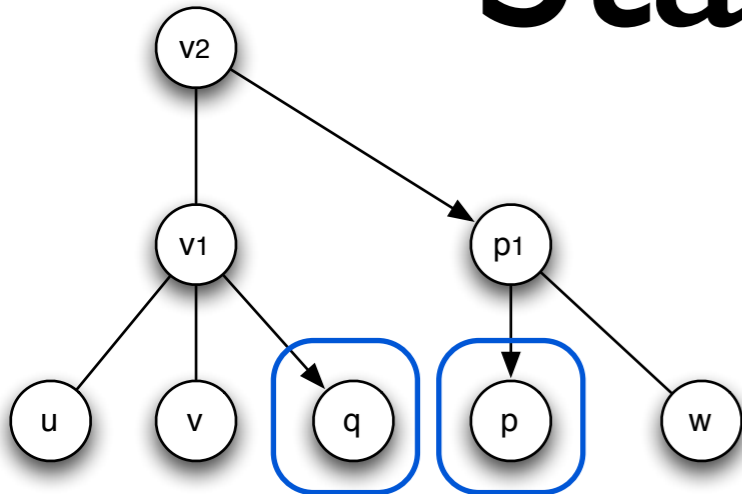


- v requests x and finds node q

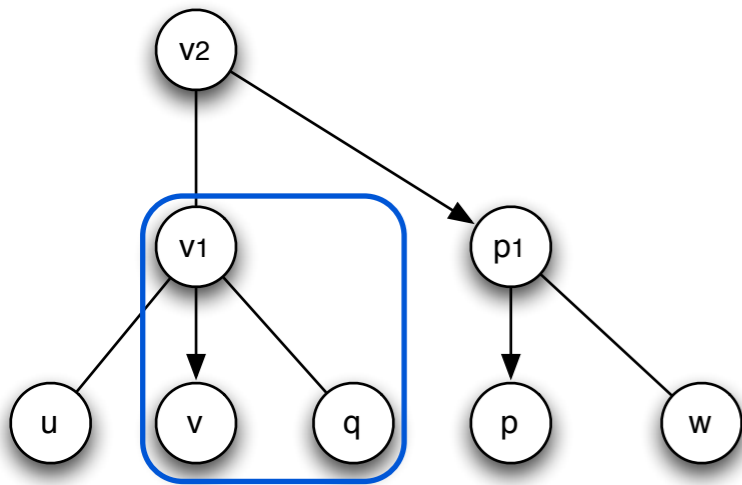
- but q does not own x



Starvation Problem

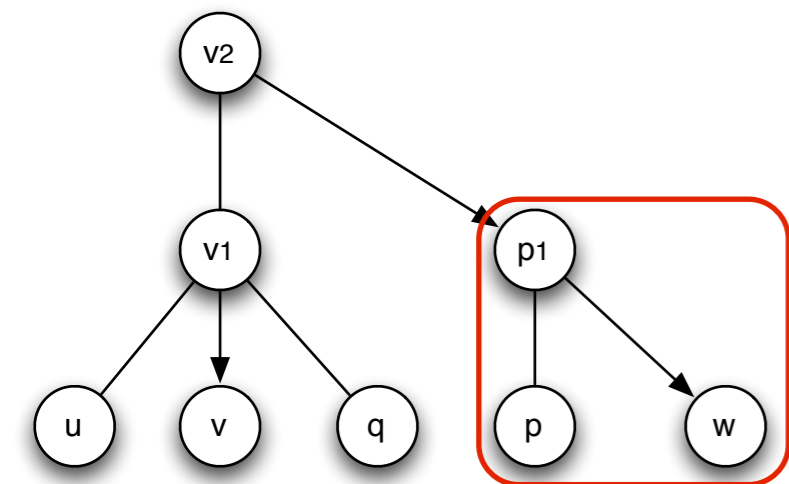


- Initially p owns x while q requests it



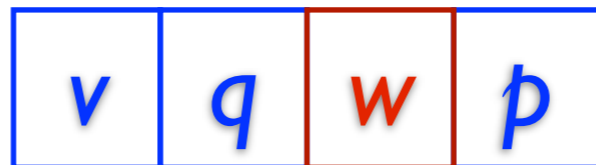
- v requests x and finds node q

- but q does not own x



- then w requests x

- w obtains it prior to v and q



Starvation-freedom

Starvation-freedom: All requests terminate.

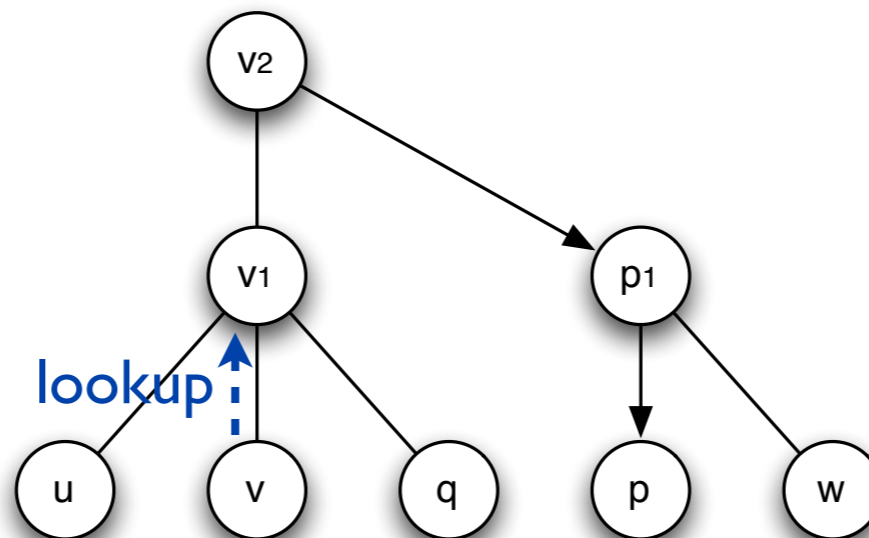
Intuition.

There is always a path of downward pointers from the root to a leaf. (By induction as requests execute.)

A move terminates in a bounded amount of time. (Only a finite number of nodes, n , can split the queue.)

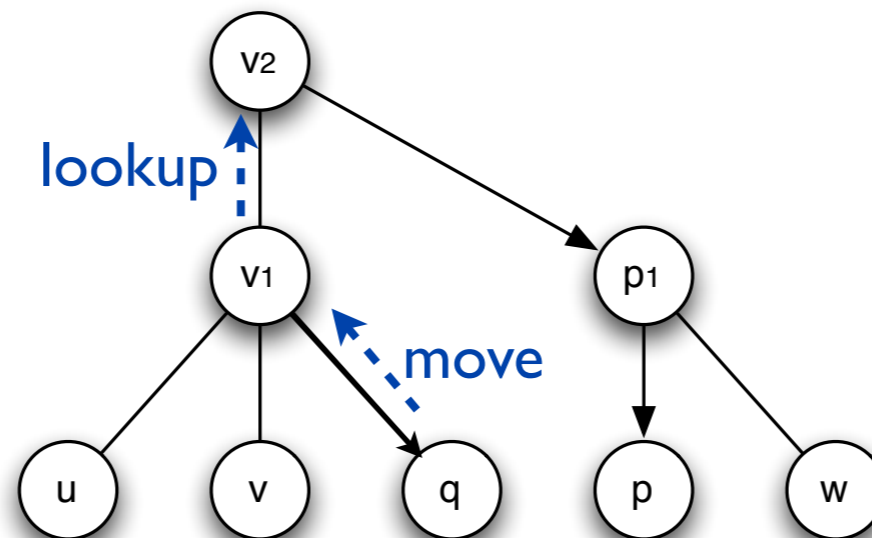
Lost Lookup Problem

- v initiates a lookup then q moves



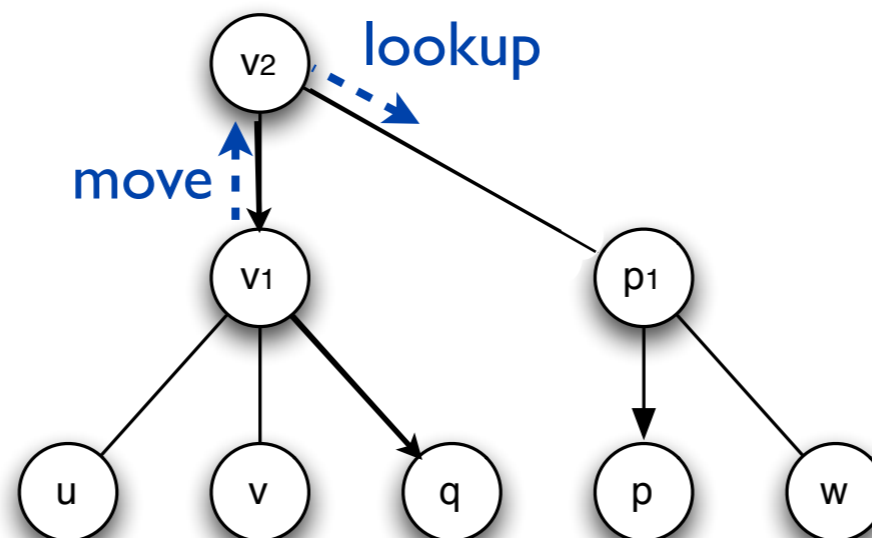
Lost Lookup Problem

- v initiates a lookup then q moves
- v_2 receives the **lookup before the move**



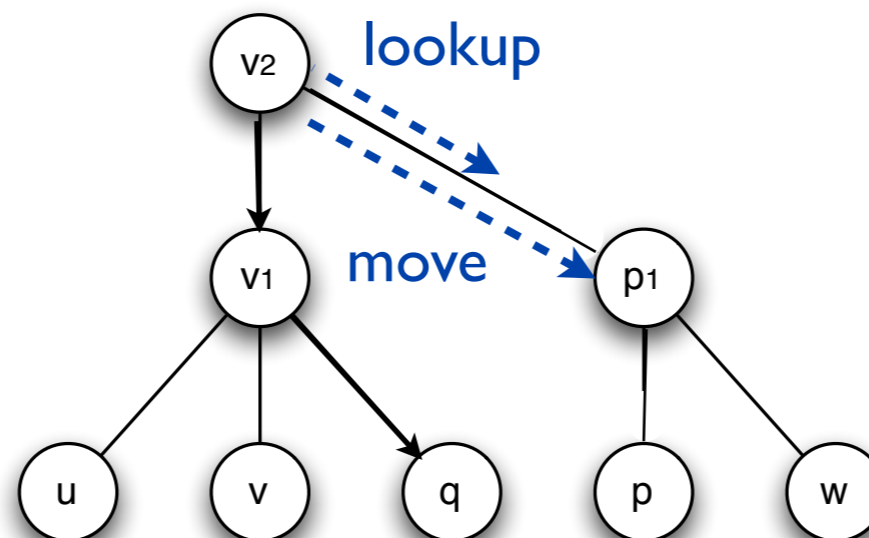
Lost Lookup Problem

- v initiates a lookup then q moves
- v_2 receives the **lookup before the move**
- p_1 receives the **move before the lookup**



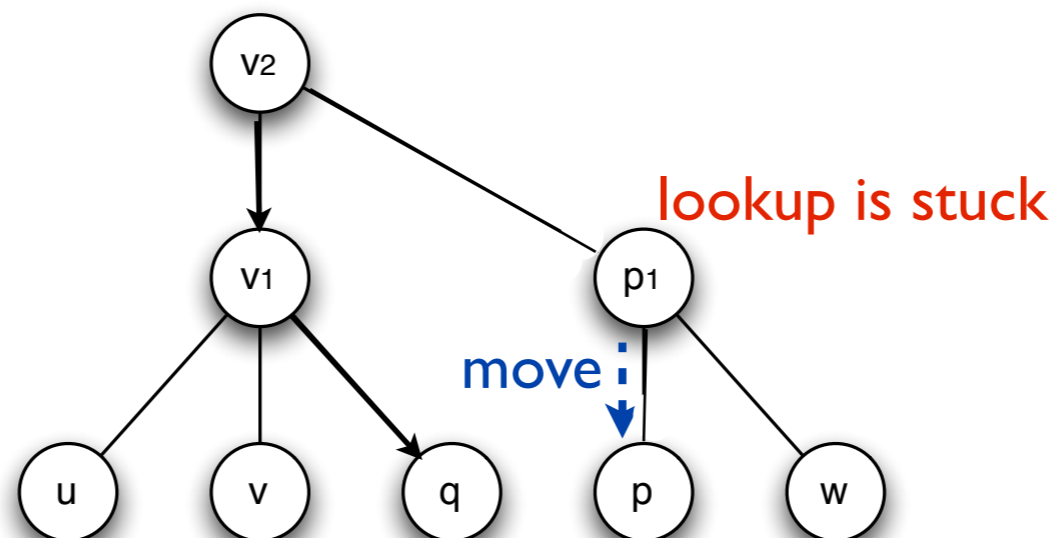
Lost Lookup Problem

- v initiates a lookup then q moves
- v_2 receives the **lookup before the move**
- p_1 receives the **move before the lookup**
- p_1 **resets its $p_1.pointer$** upon reception



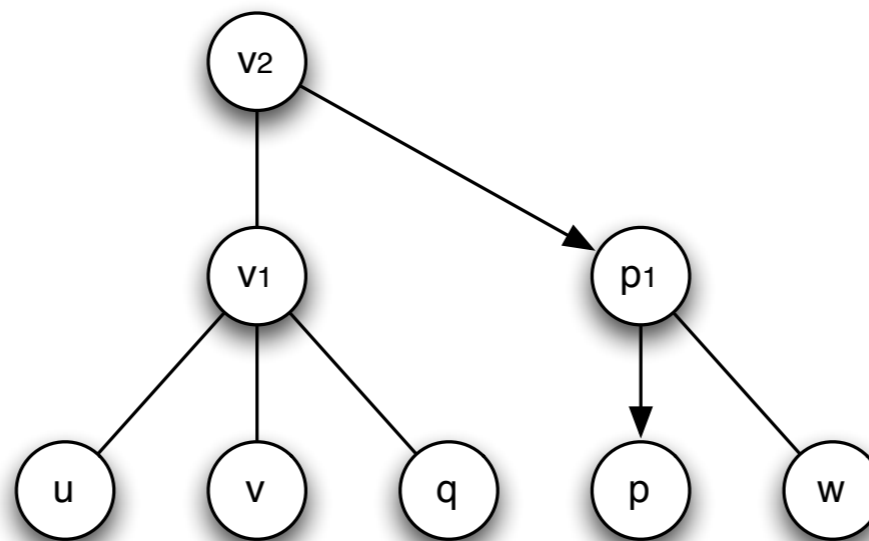
Lost Lookup Problem

- v initiates a lookup then q moves
- v_2 receives the **lookup before the move**
- p_1 receives the **move before the lookup**
- p_1 **resets its $p_1.pointer$** upon reception



How to solve this?

- Combining:
 - Move requests **do not pass over** lookups
 - But **piggybacks** them



Conclusion

- A new directory protocol
Combine
- Does any request complete?
Yes, it is starvation-free!
- Exploiting locality
The requestor gets x from a close node
- Efficiency
 $O(d(p,q))$ message complexity