Peer-to-Peer and Large-Scale Distributed Systems

Jeff Chase
Duke University

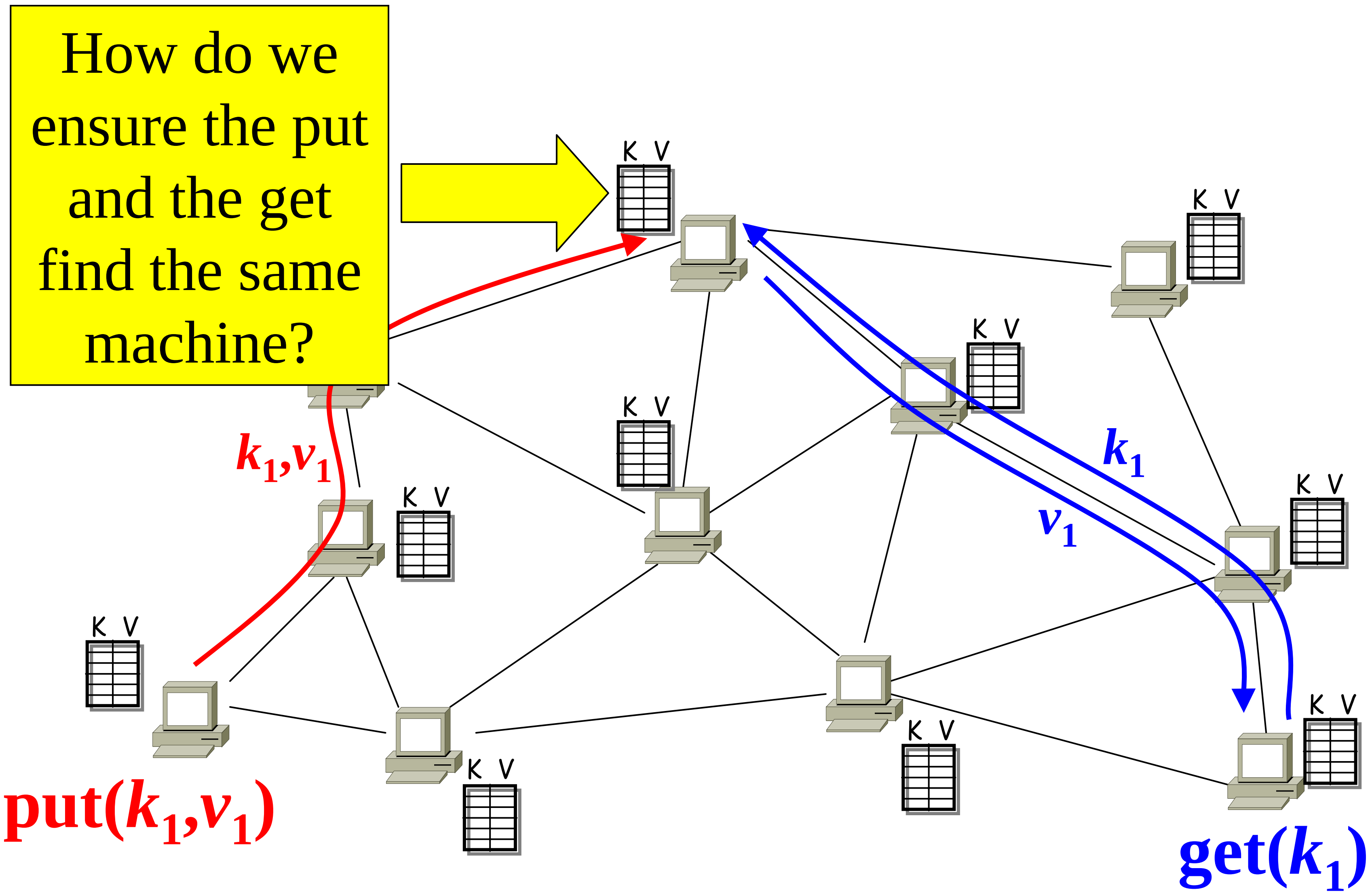
Note

- For CPS 196, Spring 2006, I skimmed a tutorial giving a broad view of the area. It is by Joe Hellerstein at Berkeley and is available at:
  - db.cs.berkeley.edu/jmh/talks/**vldb04-p2ptut-final**.ppt
- I also used some of the following slides on DHTs, all of which are adapted more or less intact from presentations graciously provided by Sean Rhea. They pertain to his Award Paper on Bamboo in Usenix 2005.

What's a DHT?

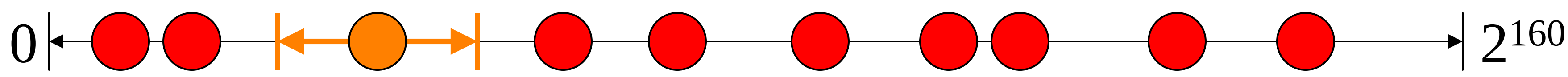
- Distributed Hash Table
  - Peer-to-peer algorithm to offering put/get interface
  - Associative map for peer-to-peer applications
- More generally, provide *lookup* functionality
  - Map application-provided hash values to nodes
  - (Just as local hash tables map hashes to memory locs.)
  - Put/get then constructed above lookup
- Many proposed applications
  - File sharing, end-system multicast, aggregation trees

How DHTs Work



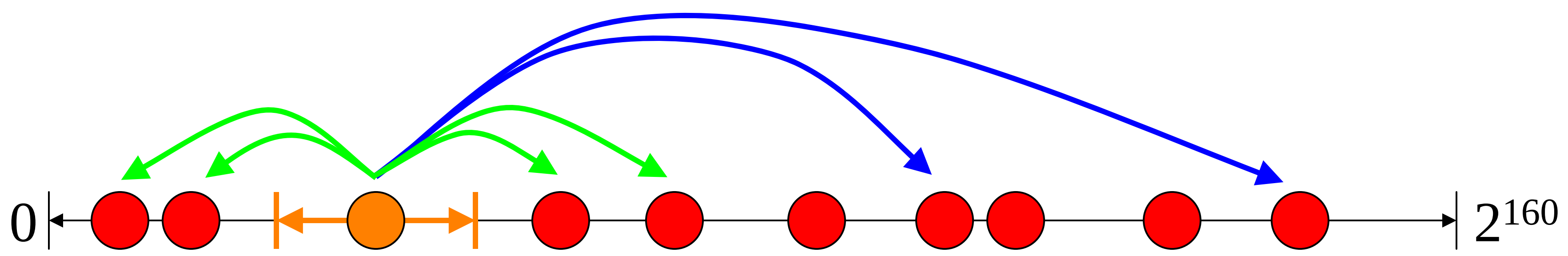
Step 1: Partition Key Space

- Each node in DHT will store some *k,v* pairs
- Given a key space *K*, *e.g.* $[0, 2^{160})$:
  - Choose an identifier for each node, $id_i \in K$, uniformly at random
  - A pair *k,v* is stored at the node whose identifier is closest to *k*



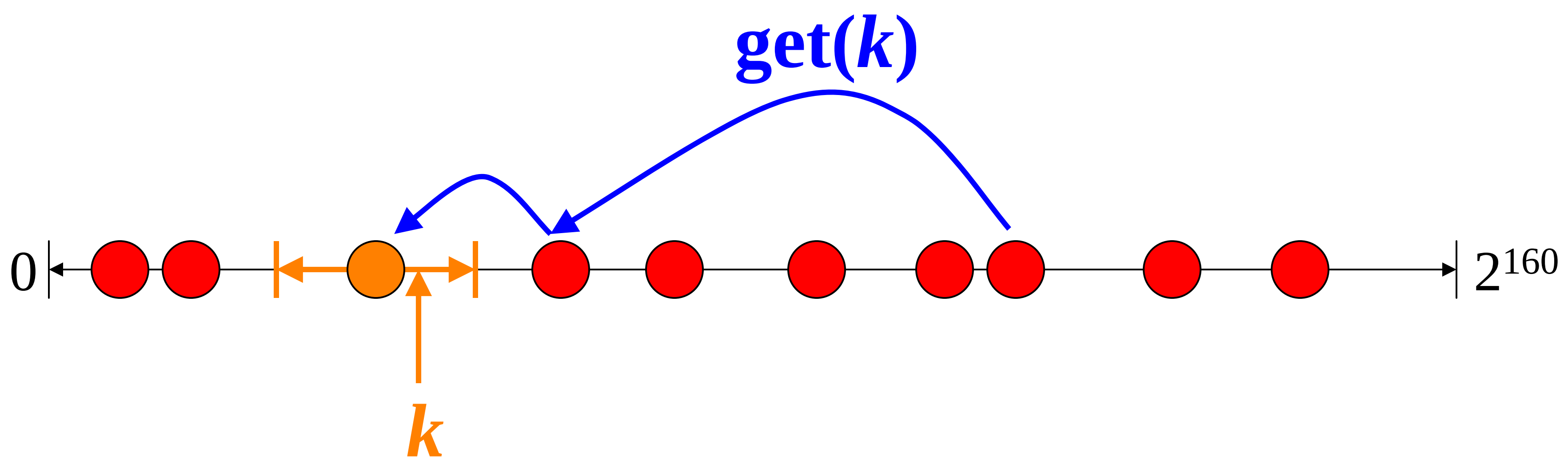
Step 2: Build Overlay Network

- Each node has two sets of neighbors
- **Immediate neighbors in the key space**
  - Important for correctness
- **Long-hop neighbors**
  - Allow puts/gets in O(log *n*) hops



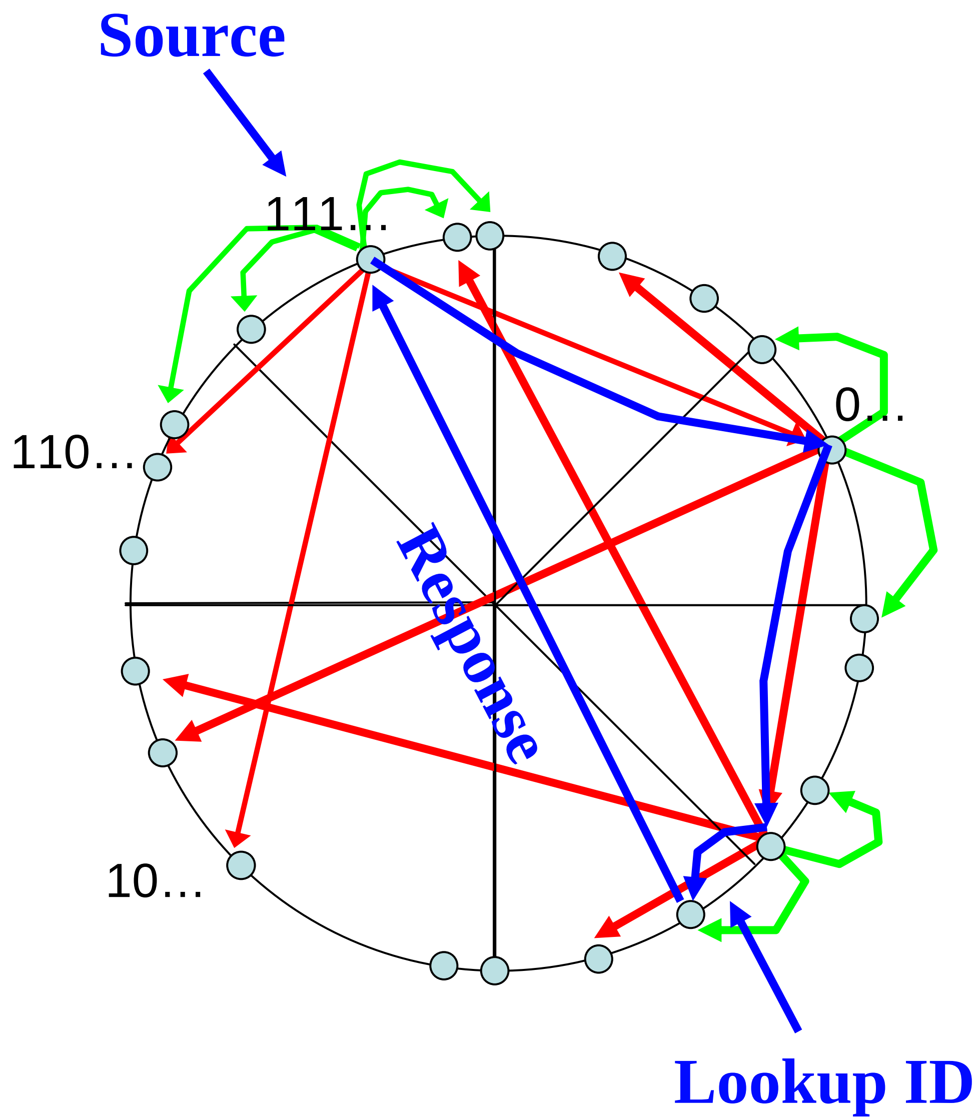
Step 3: Route Puts/Gets Thru Overlay

- Route greedily, always making progress

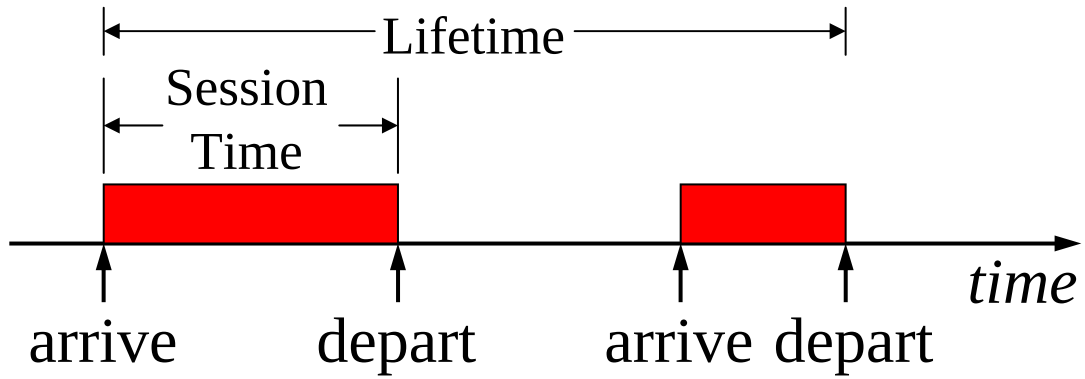


How Does Lookup Work?

- Assign IDs to nodes
  - Map hash values to node with closest ID
- Leaf set is successors and predecessors
  - All that's needed for correctness
- Routing table matches successively longer prefixes
  - Allows efficient lookups



How Bad is Churn in Real Systems?



An hour is an incredibly short MTTF!

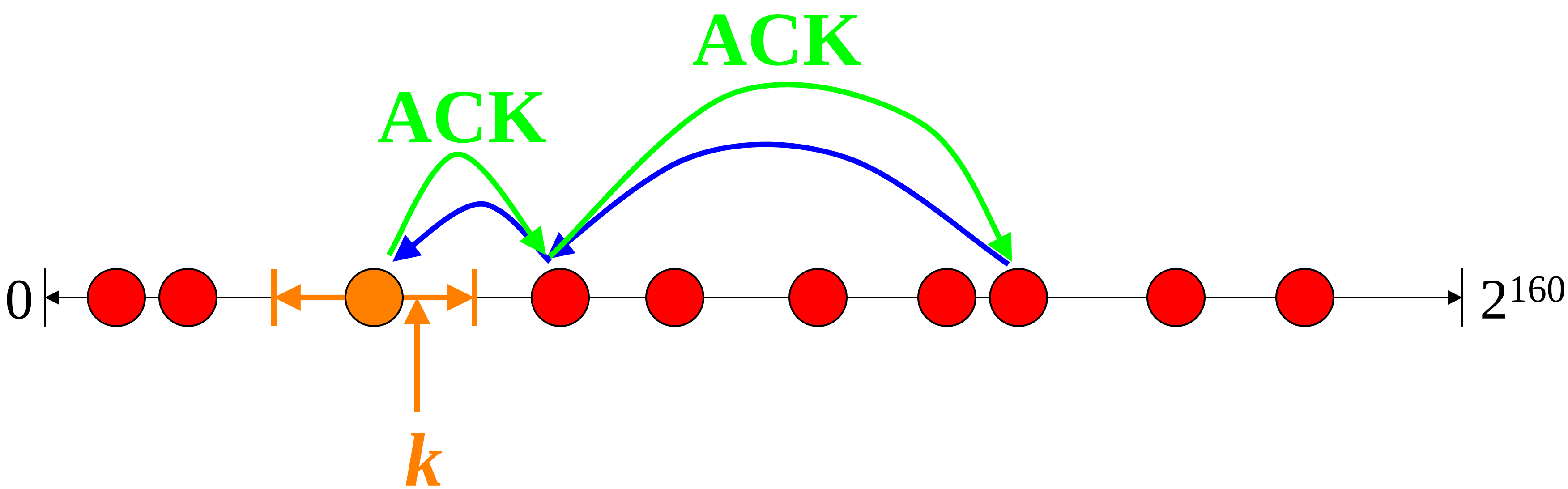
Authors	*Systems Observed*	*Session Time*
SGG02	Gnutella, Napster	50% < 60 minutes
CLL02	Gnutella, Napster	31% < 10 minutes
SW02	FastTrack	50% < 1 minute
BSV03	Overnet	50% < 60 minutes
GDS03	Kazaa	50% < 2.4 minutes

Note on CPS 196, Spring 2006

- We did not cover any of the following material on managing DHT's under churn.

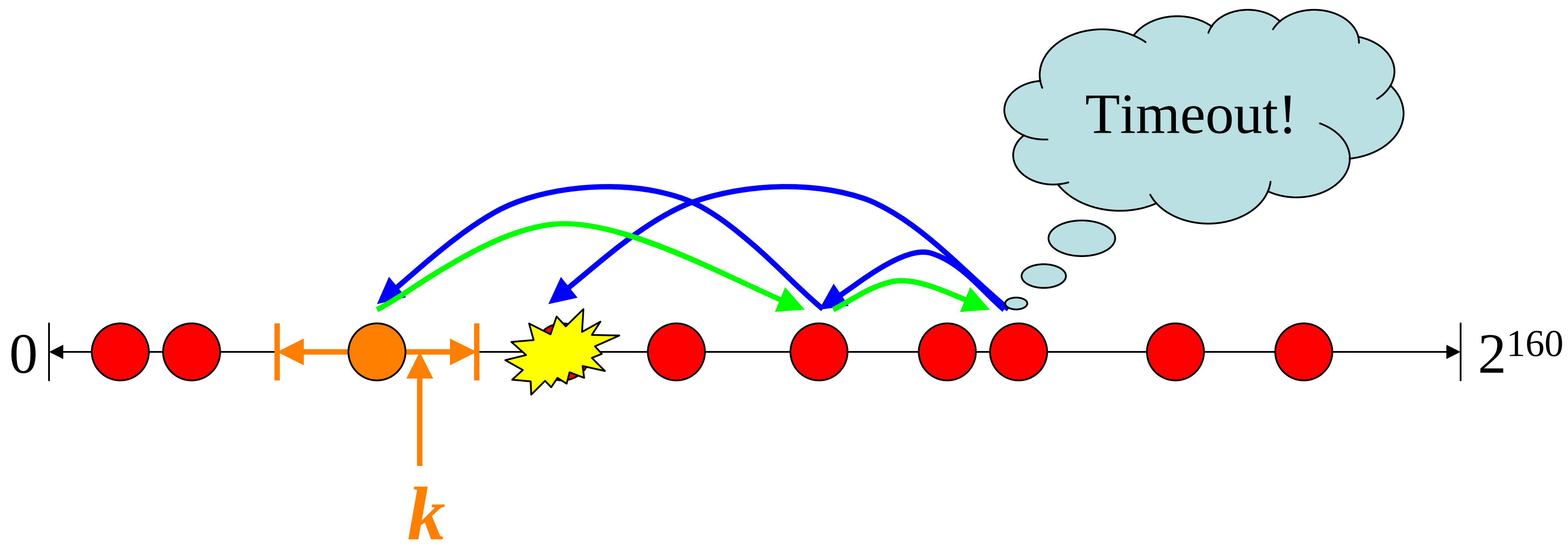
Routing Around Failures

- Under churn, neighbors may have failed
- To detect failures, acknowledge each hop



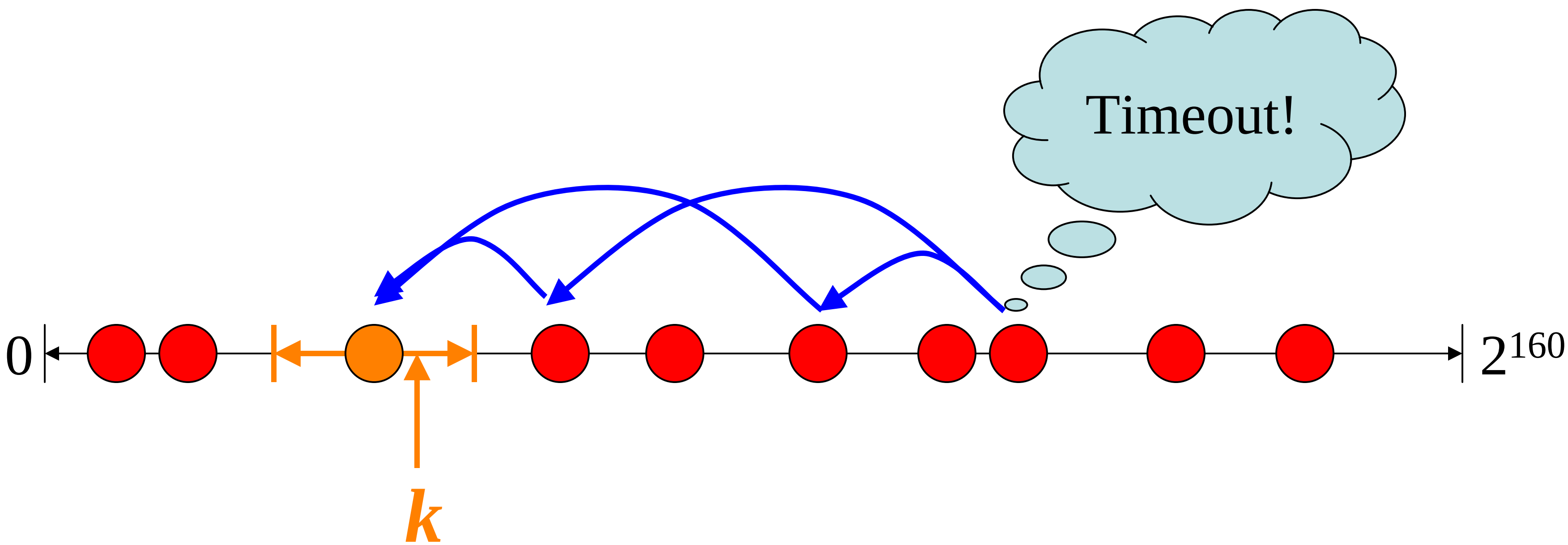
Routing Around Failures

- If we don't receive an ACK, resend through different neighbor



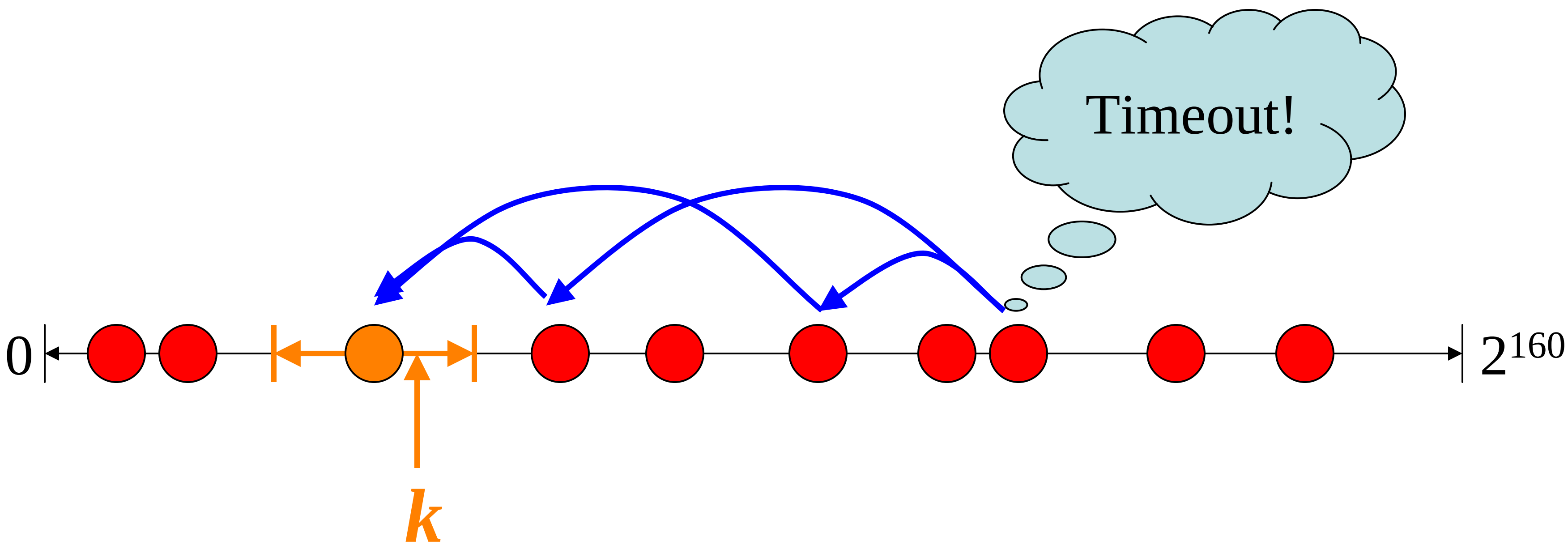
Computing Good Timeouts

- Must compute timeouts carefully
  - If too long, increase put/get latency
  - If too short, get message explosion



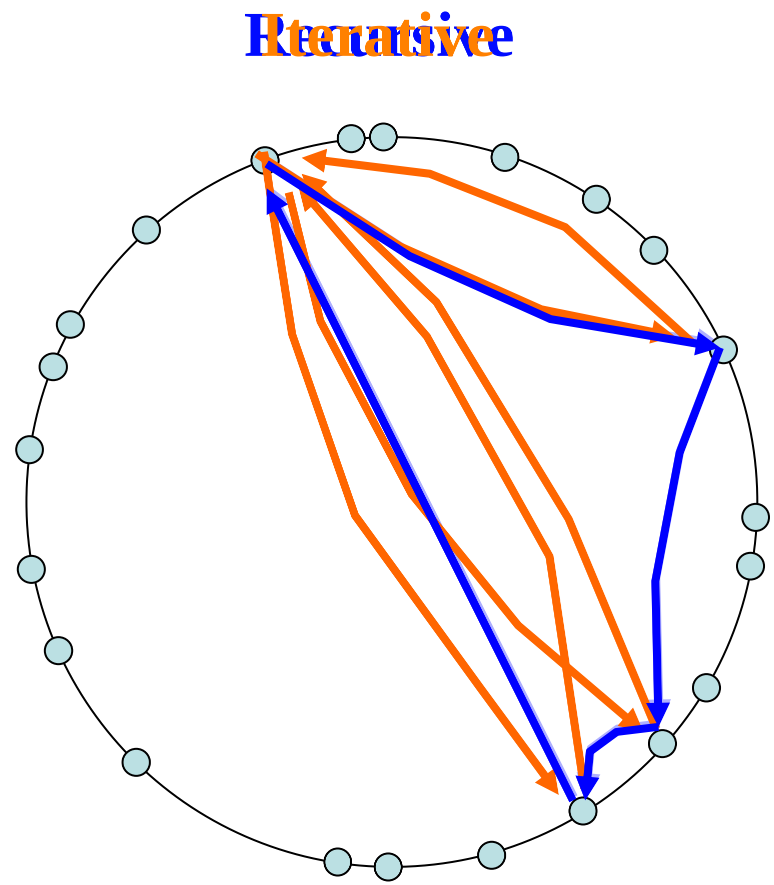
Computing Good Timeouts

- Chord errs on the side of caution
  - Very stable, but gives long lookup latencies



Calculating Good Timeouts

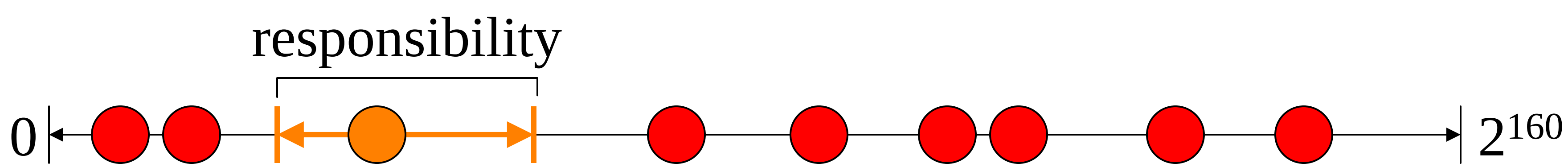
- Use TCP-style timers
  - Keep past history of latencies
  - Use this to compute timeouts for new requests
- Works fine for *recursive* lookups
  - Only talk to neighbors, so history small, current



- In *iterative* lookups, source directs entire lookup
  - Must potentially have good timeout for *any* node

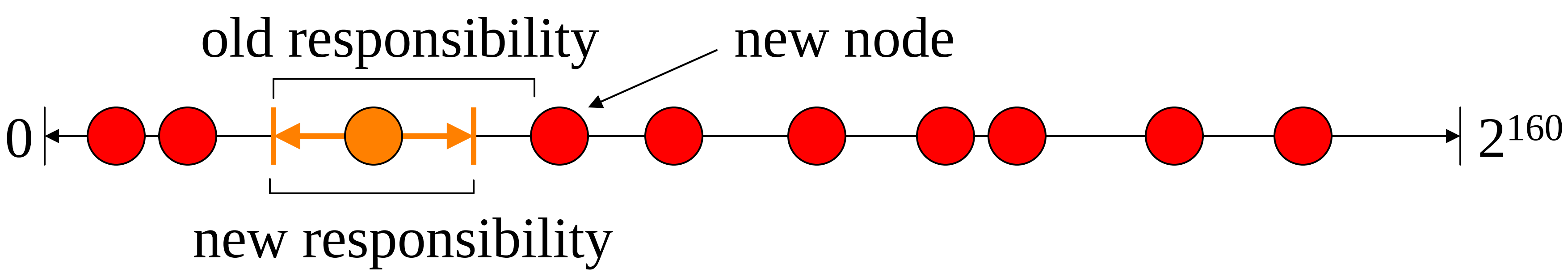
Recovering From Failures

- Can't route around failures forever
  - Will eventually run out of neighbors
- Must also find new nodes as they join
  - Especially important if they're our immediate predecessors or successors:



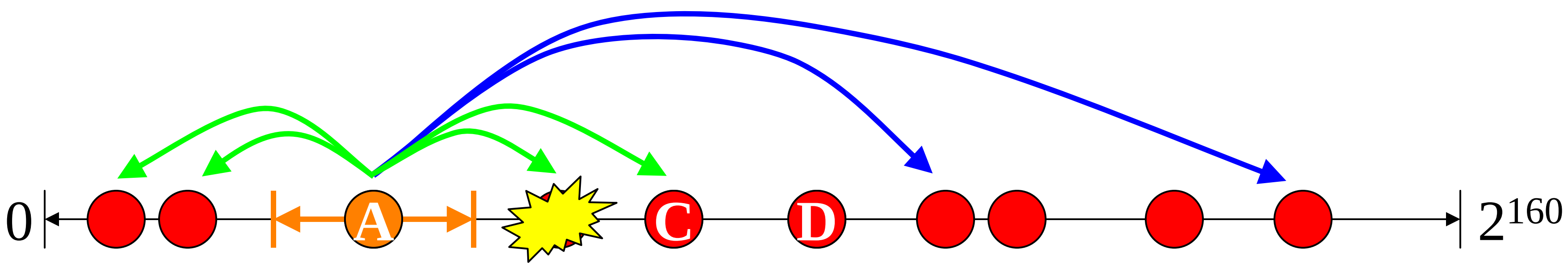
Recovering From Failures

- Can't route around failures forever
  - Will eventually run out of neighbors
- Must also find new nodes as they join
  - Especially important if they're our immediate predecessors or successors:



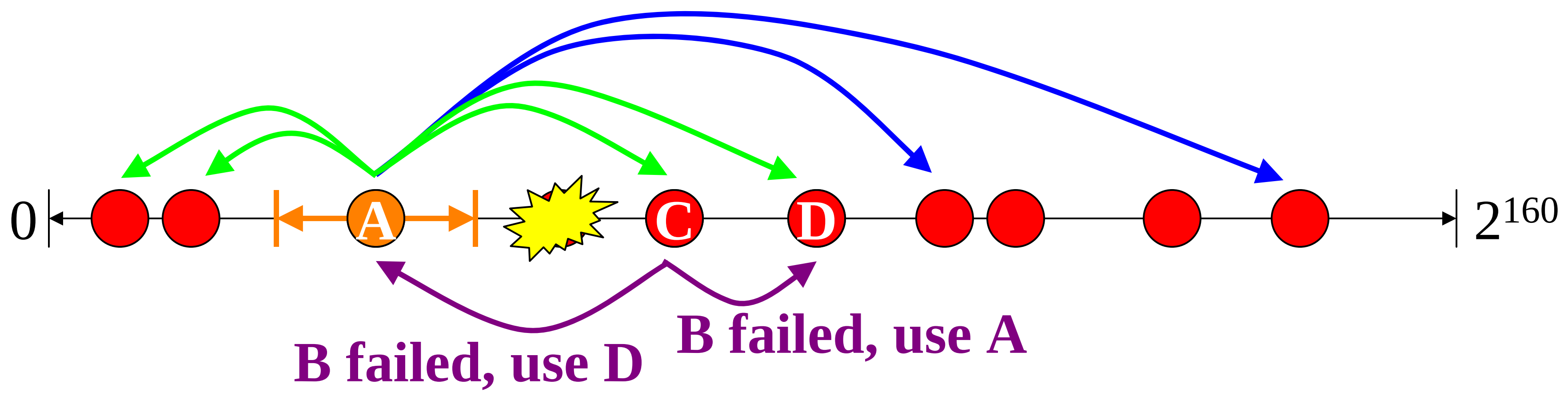
Recovering From Failures

- Obvious algorithm: *reactive* recovery
  - When a node stops sending acknowledgements, notify other neighbors of potential replacements
  - Similar techniques for arrival of new nodes



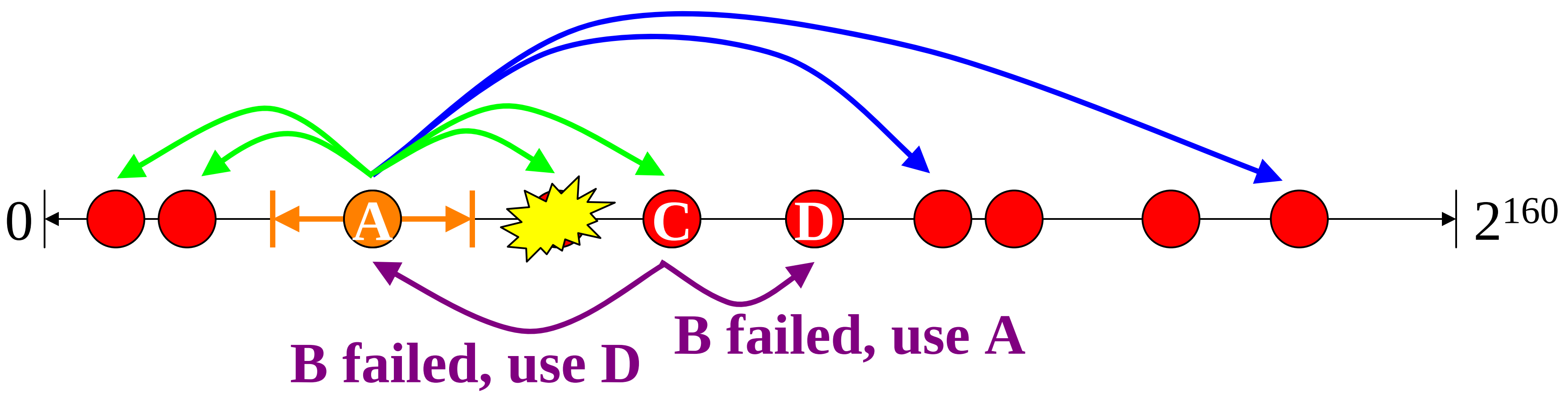
Recovering From Failures

- Obvious algorithm: *reactive* recovery
  - When a node stops sending acknowledgements, notify other neighbors of potential replacements
  - Similar techniques for arrival of new nodes



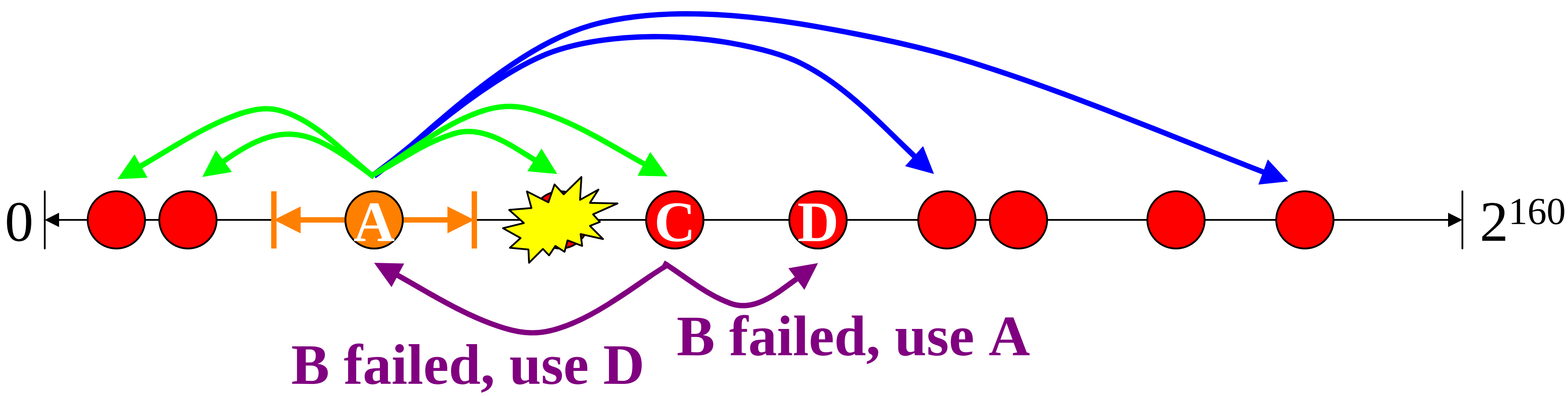
The Problem with Reactive Recovery

- What if B is alive, but network is congested?
  - C still perceives a failure due to dropped ACKs
  - C starts recovery, further congesting network
  - More ACKs likely to be dropped
  - Creates a positive feedback cycle



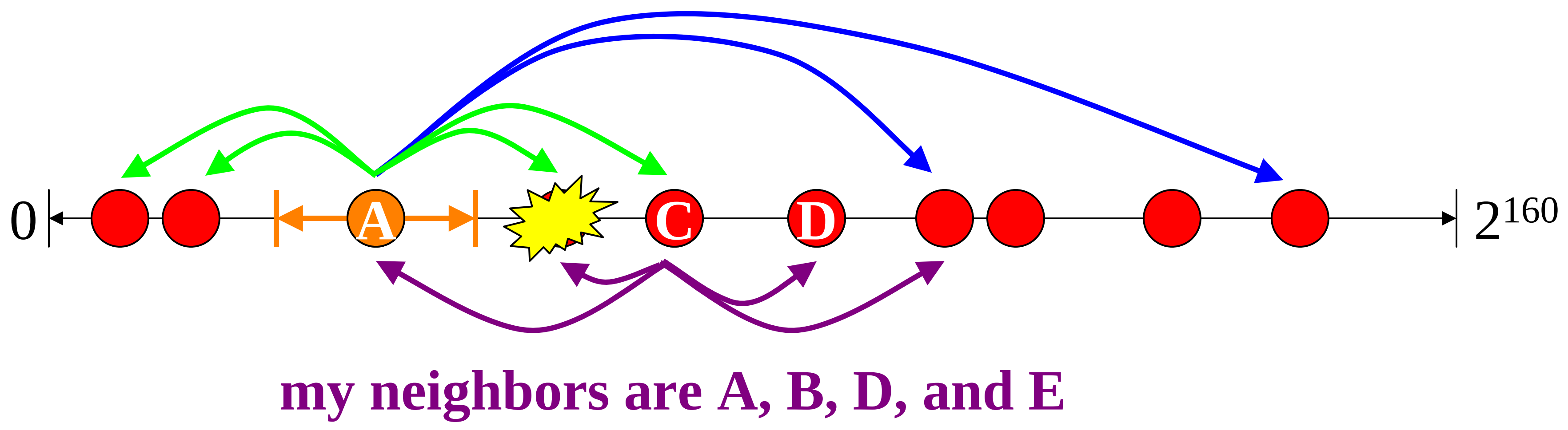
The Problem with Reactive Recovery

- What if B is alive, but network is congested?
- This was the problem with Pastry
  - Combined with poor congestion control, causes network to partition under heavy churn



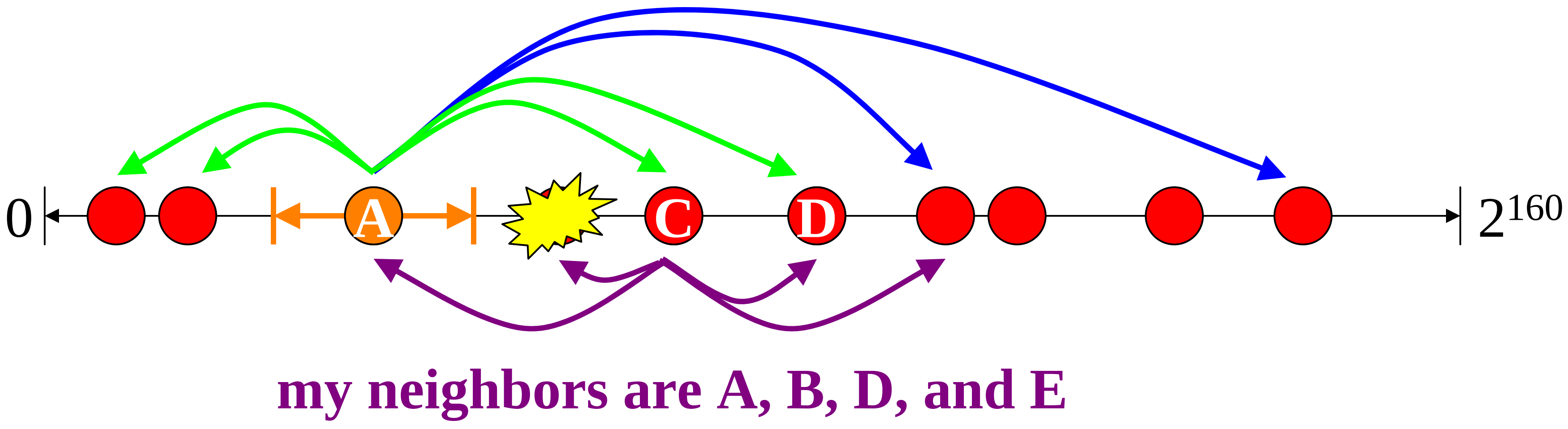
Periodic Recovery

- Every period, each node sends its neighbor list to each of its neighbors



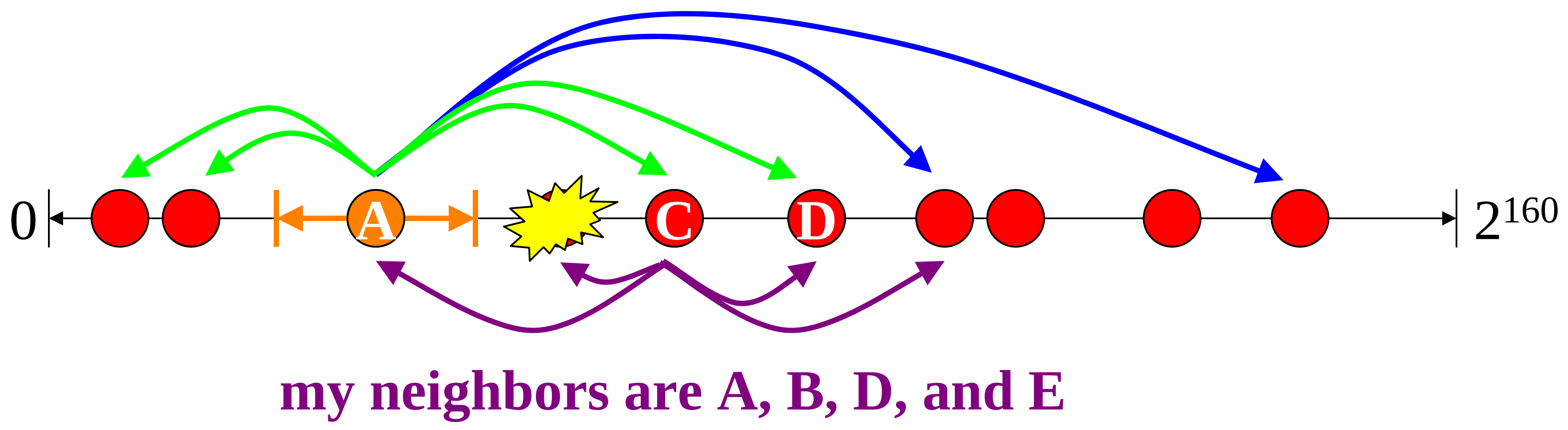
Periodic Recovery

- Every period, each node sends its neighbor list to each of its neighbors



Periodic Recovery

- Every period, each node sends its neighbor list to each of its neighbors
  - Breaks feedback loop



Periodic Recovery

- Every period, each node sends its neighbor list to each of its neighbors
  - Breaks feedback loop
  - Converges in logarithmic number of periods