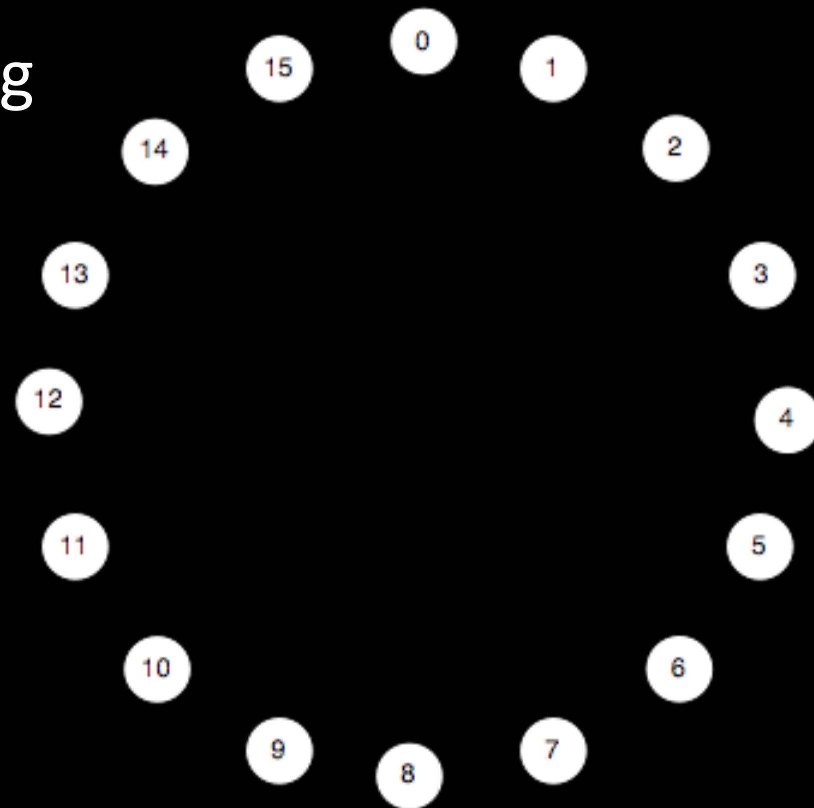
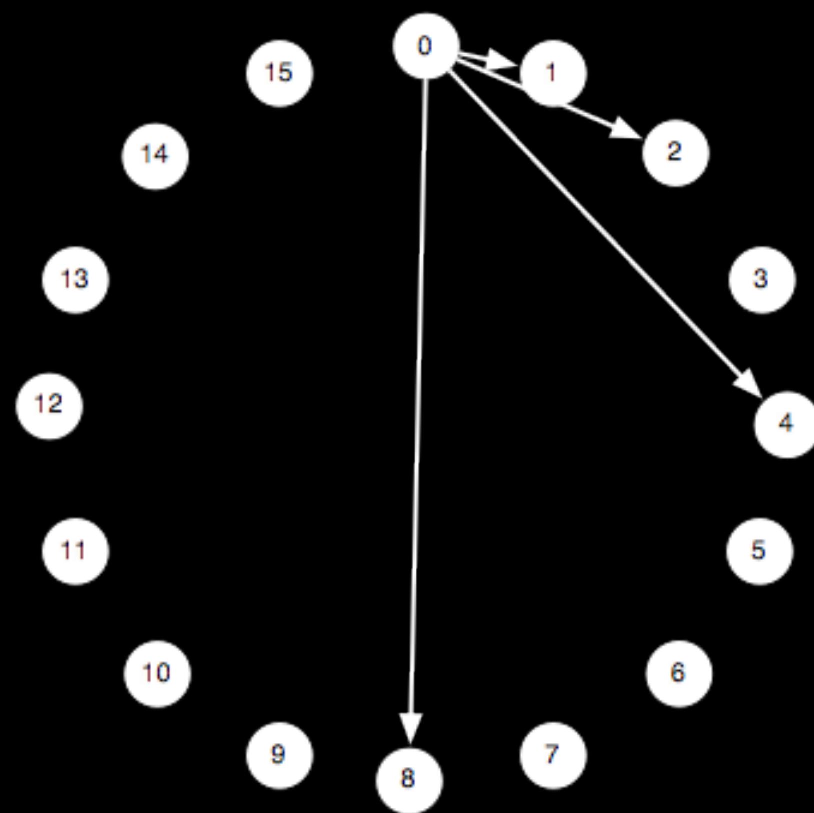
An Example DHT: Chord

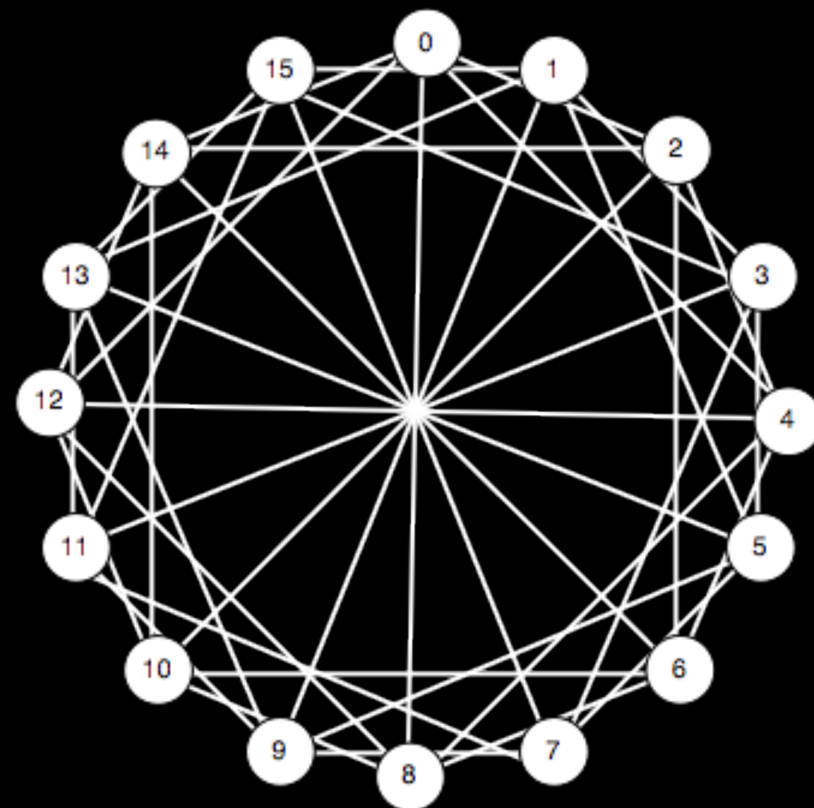
- Assume $n = 2^m$ nodes for a moment
  - A "complete" Chord ring



An Example DHT: Chord

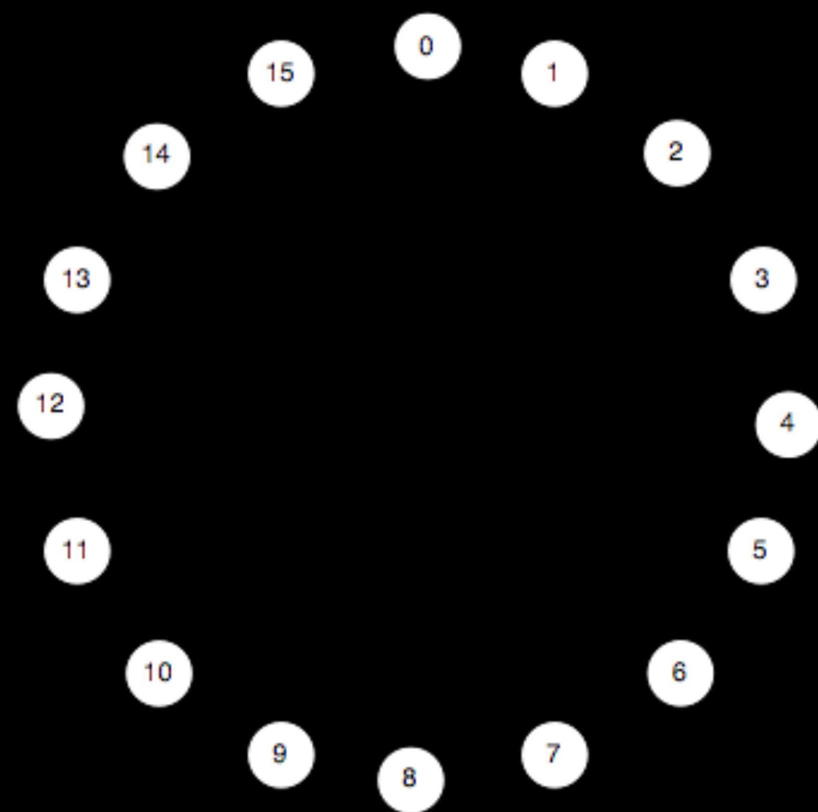


An Example DHT: Chord



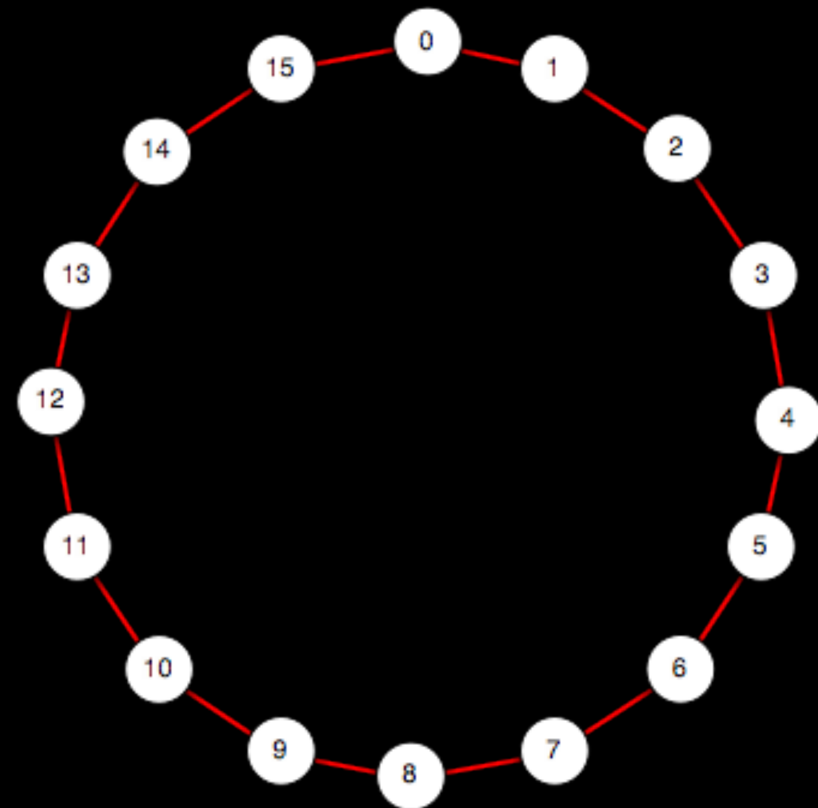
An Example DHT: Chord

- Overlayed 2^k-Gons



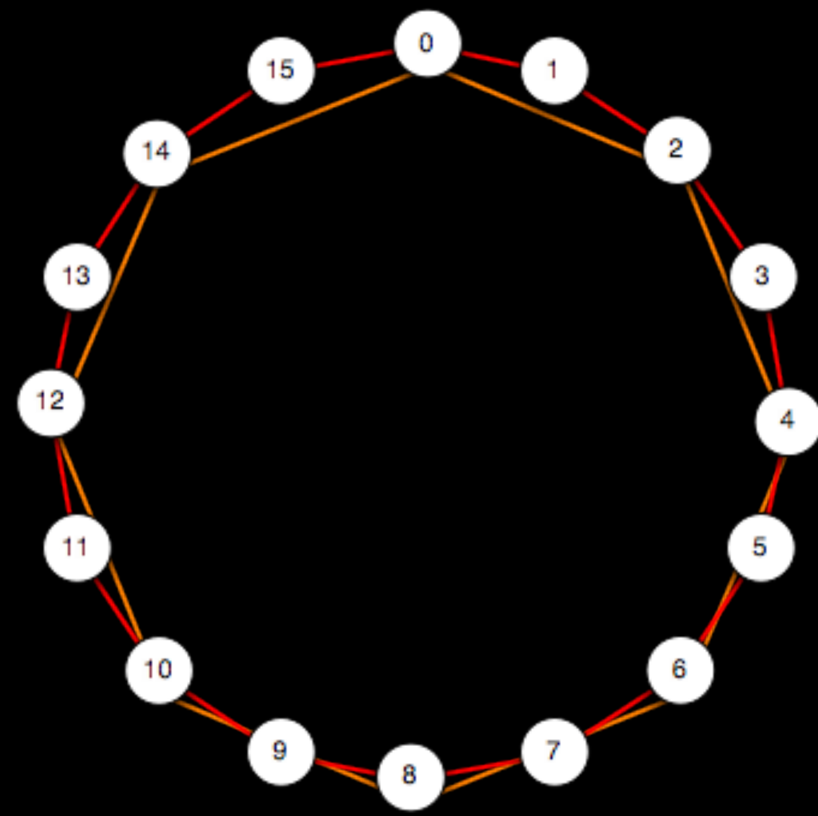
An Example DHT: Chord

- Overlayed 2^k-Gons



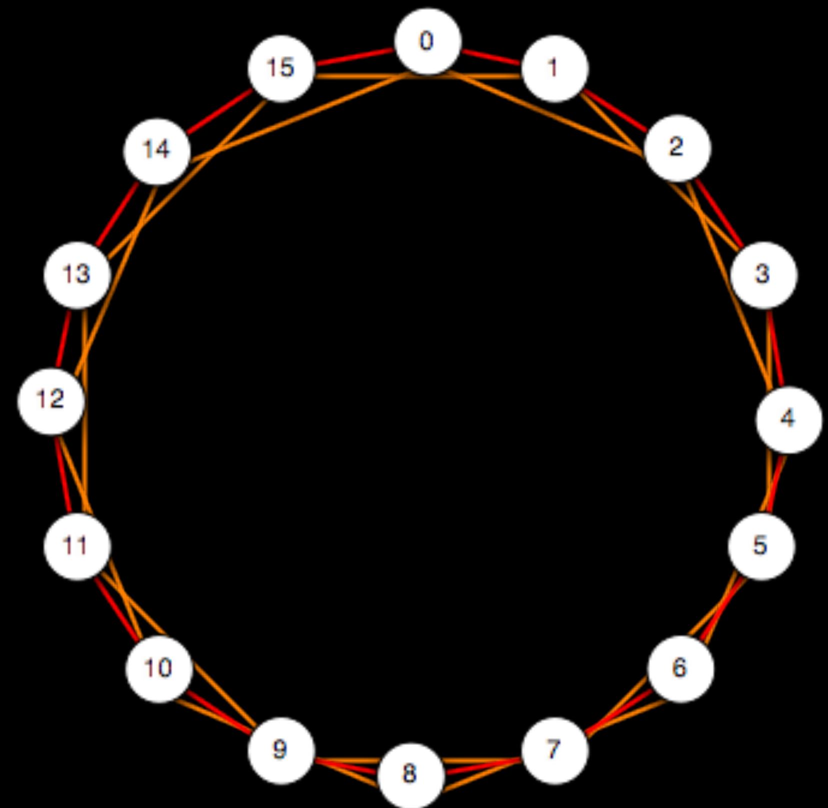
An Example DHT: Chord

- Overlayed 2^k-Gons



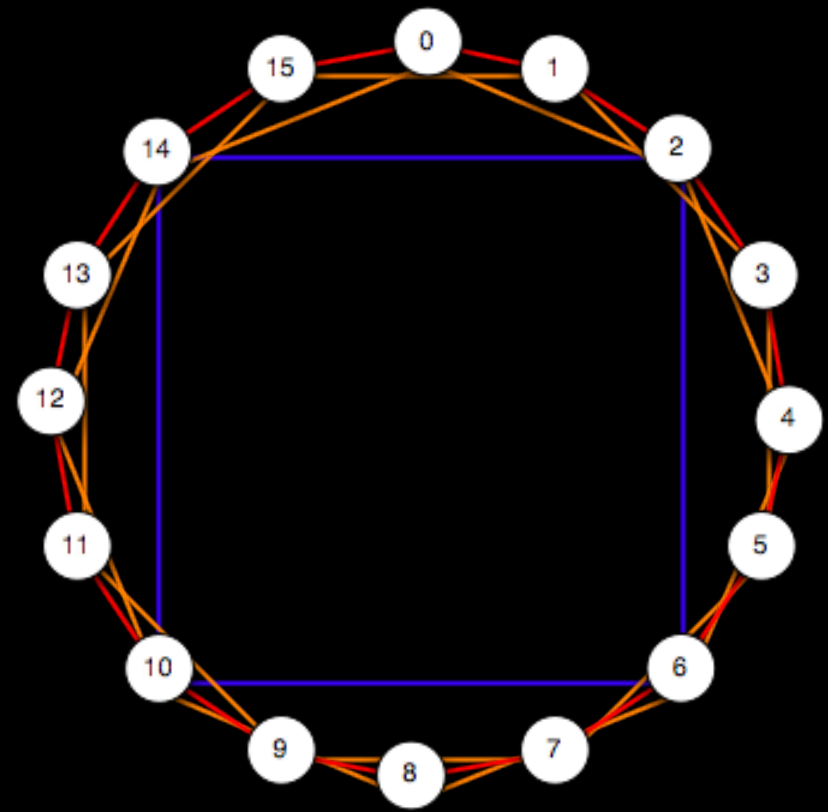
An Example DHT: Chord

- Overlayed 2^k-Gons



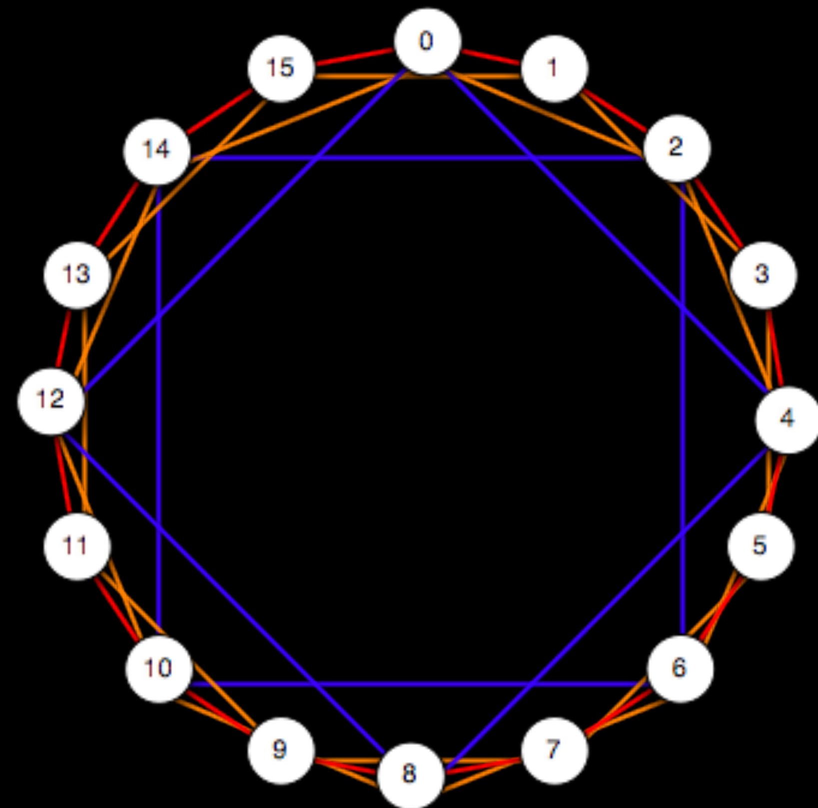
An Example DHT: Chord

- Overlayed 2^k-Gons



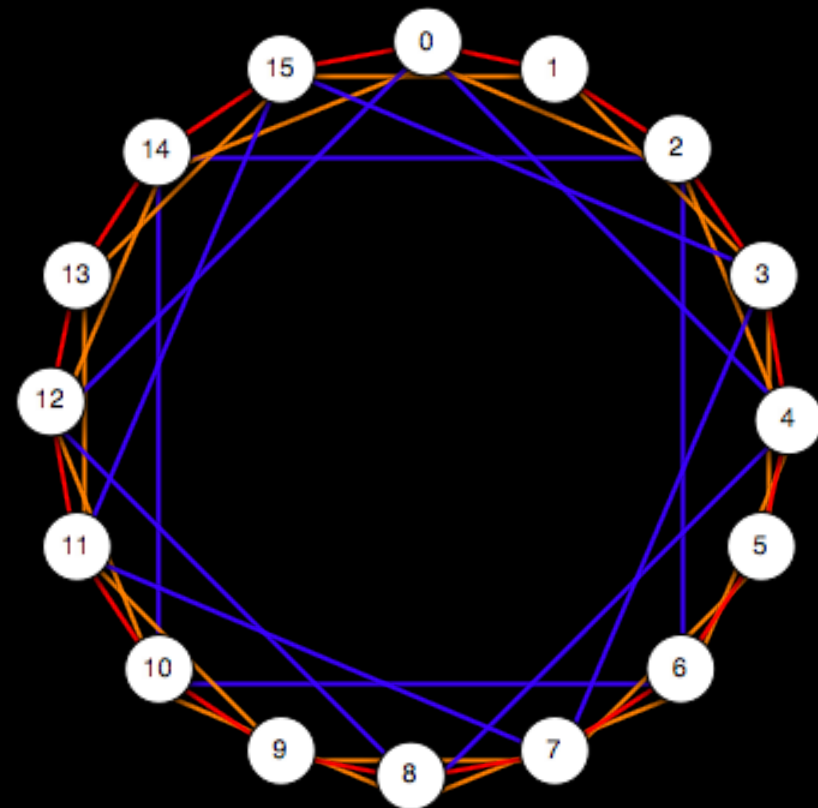
An Example DHT: Chord

- Overlayed 2^k-Gons



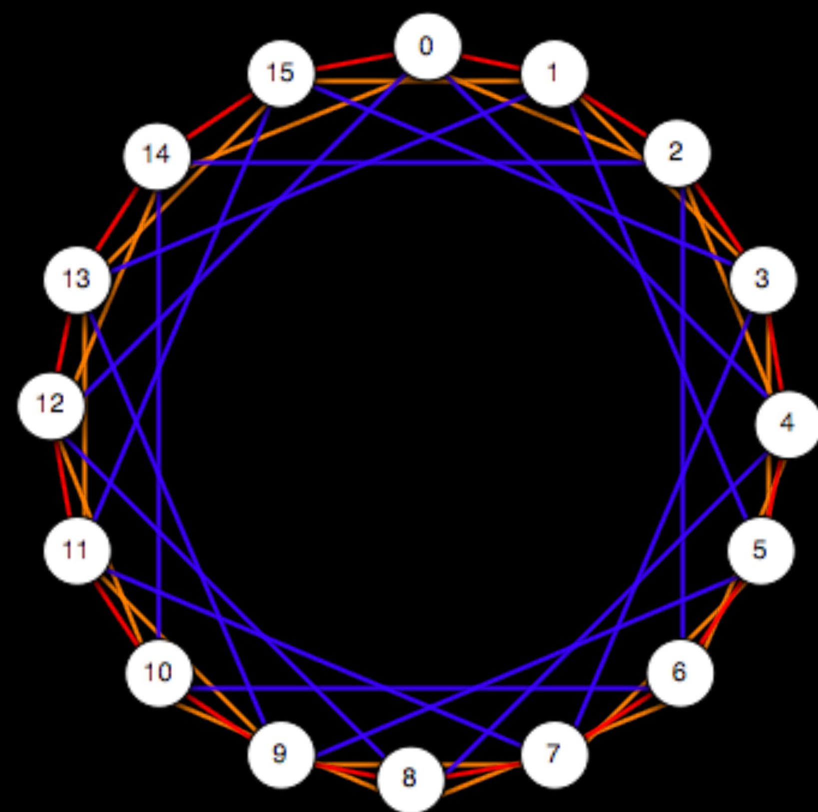
An Example DHT: Chord

- Overlayed 2^k-Gons



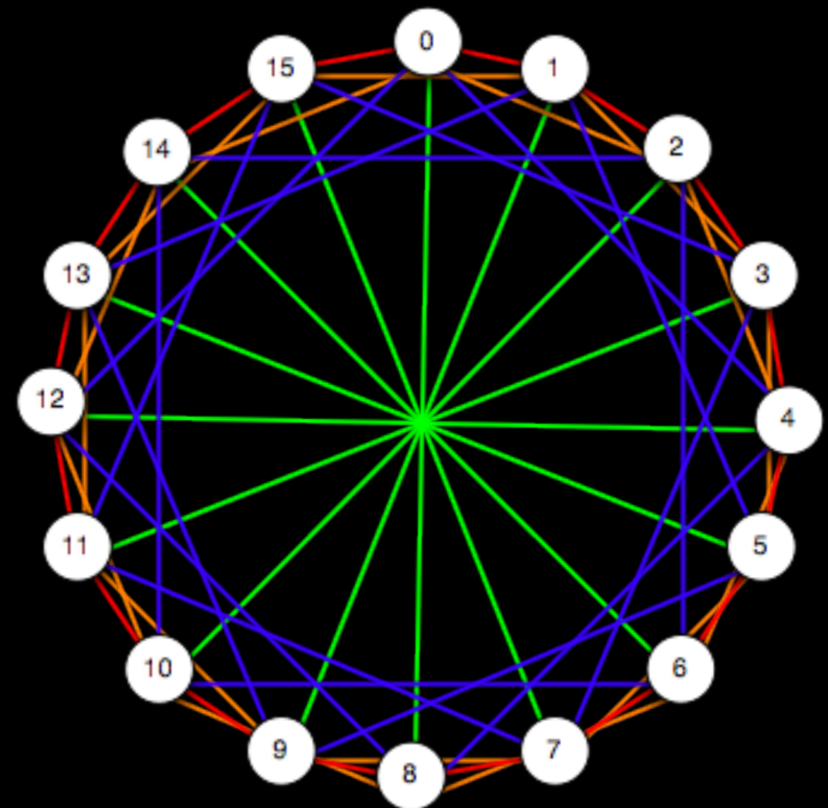
An Example DHT: Chord

- Overlayed 2^k-Gons



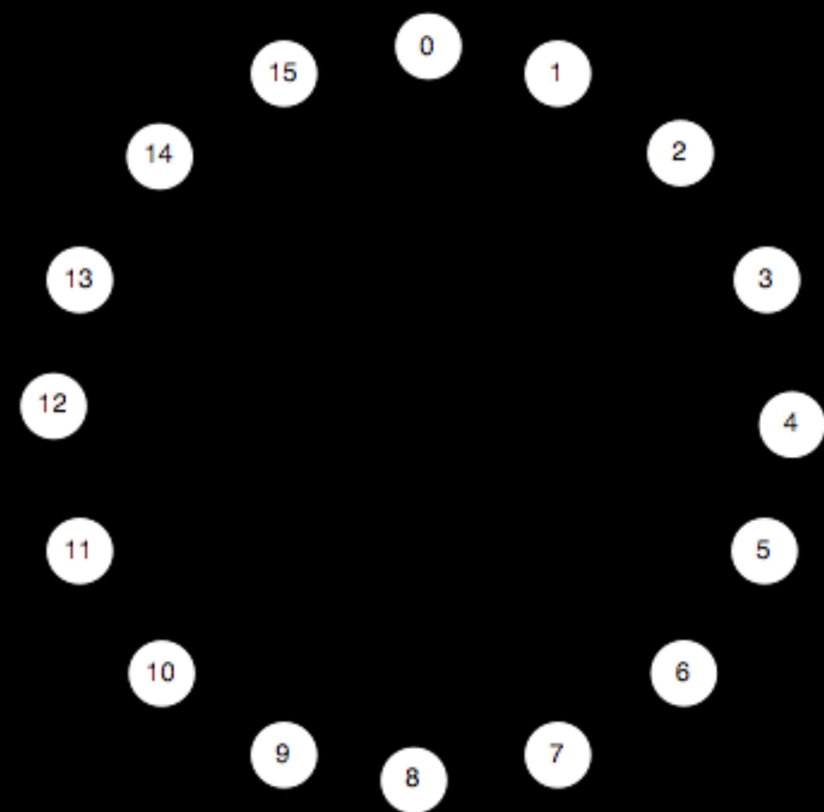
An Example DHT: Chord

- Overlayed 2^k-Gons



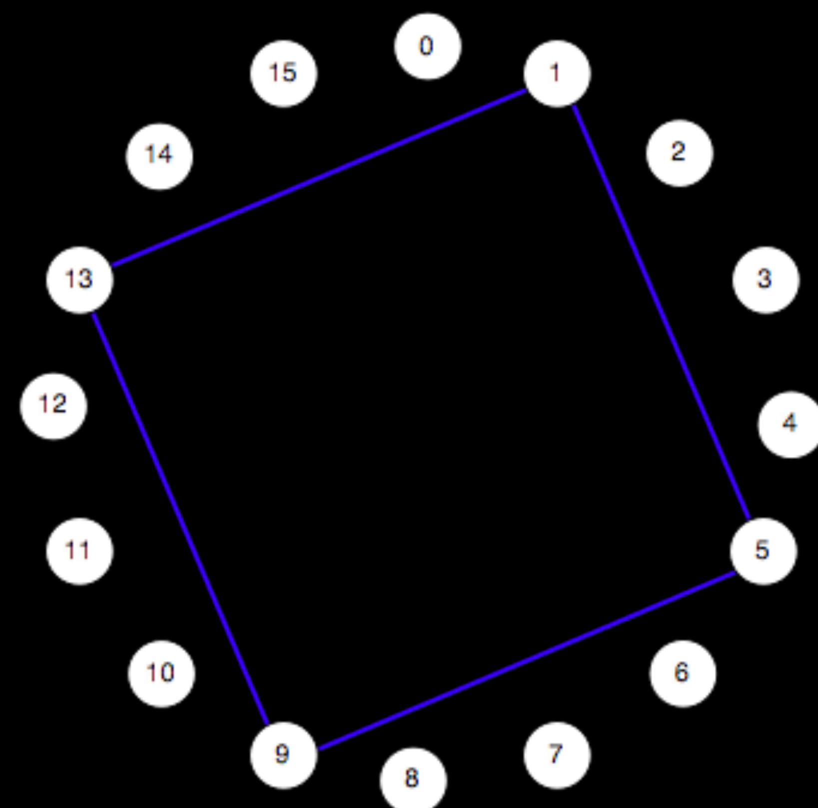
Routing in Chord

- At most one of each Gon
- E.g. 1-to-0



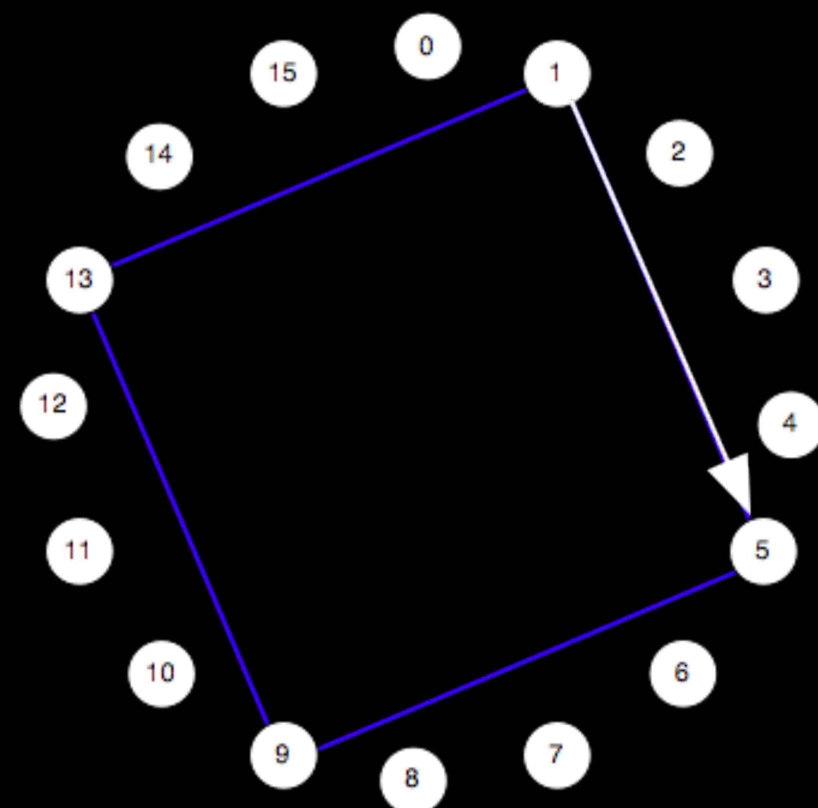
Routing in Chord

- At most one of each Gon
- E.g. 1-to-0



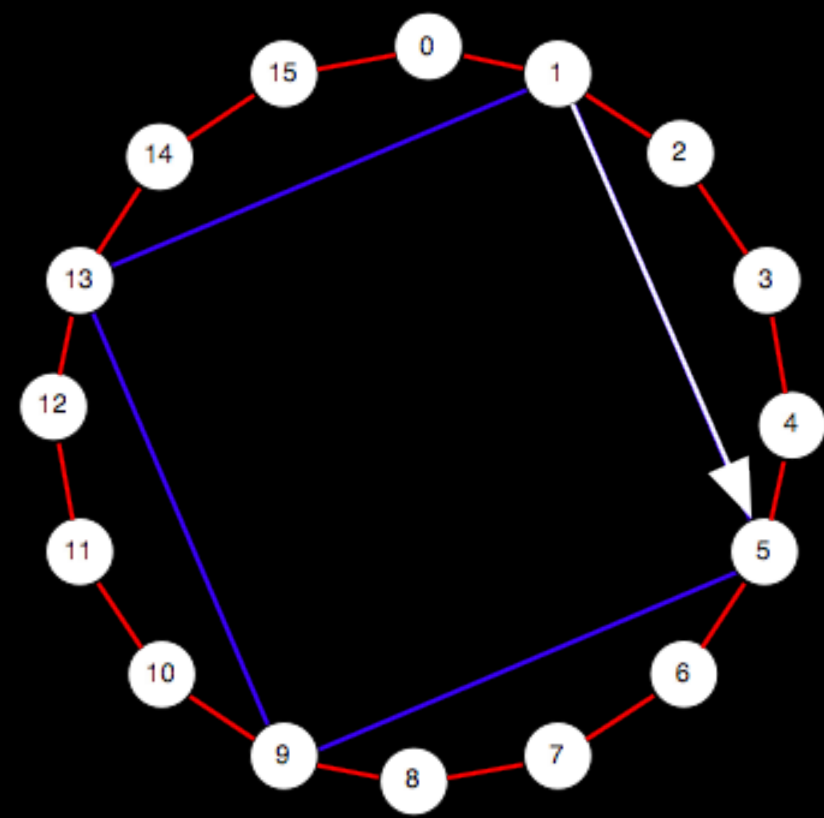
Routing in Chord

- At most one of each Gon
- E.g. 1-to-0



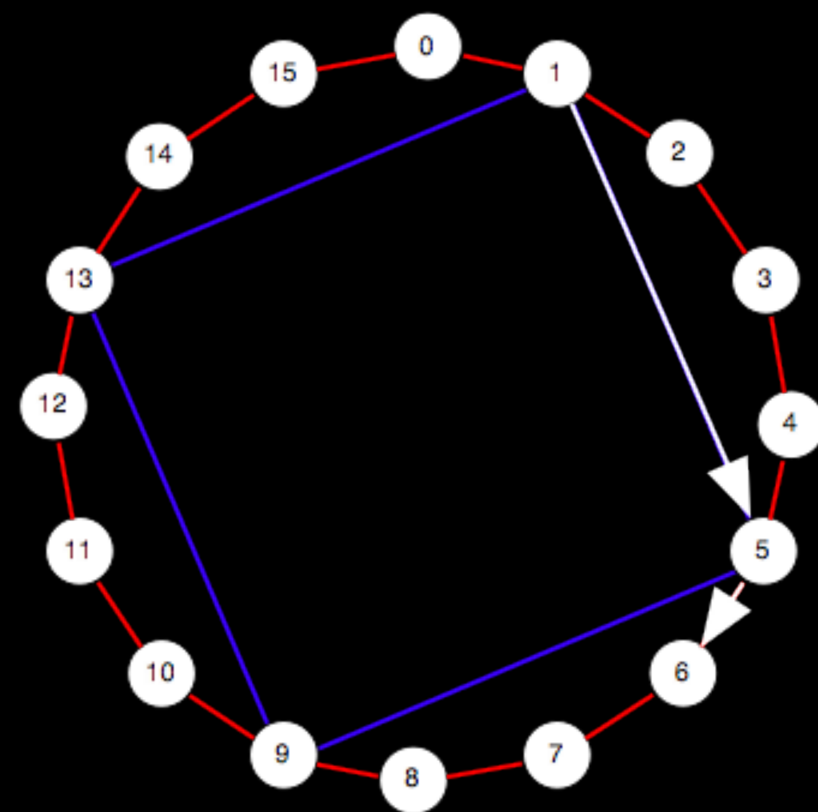
Routing in Chord

- At most one of each Gon
- E.g. 1-to-0



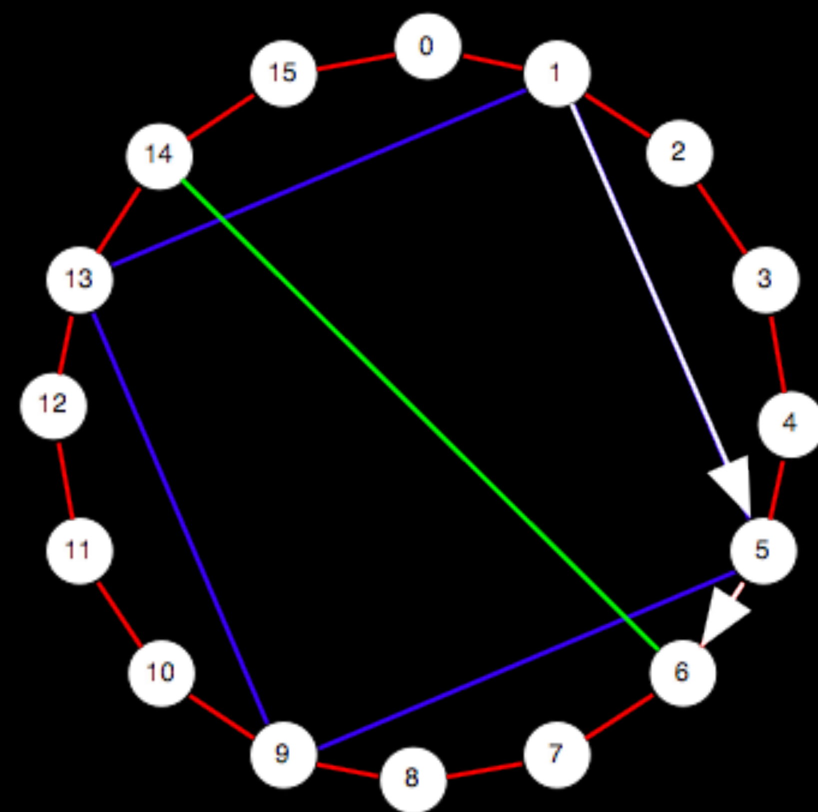
Routing in Chord

- At most one of each Gon
- E.g. 1-to-0



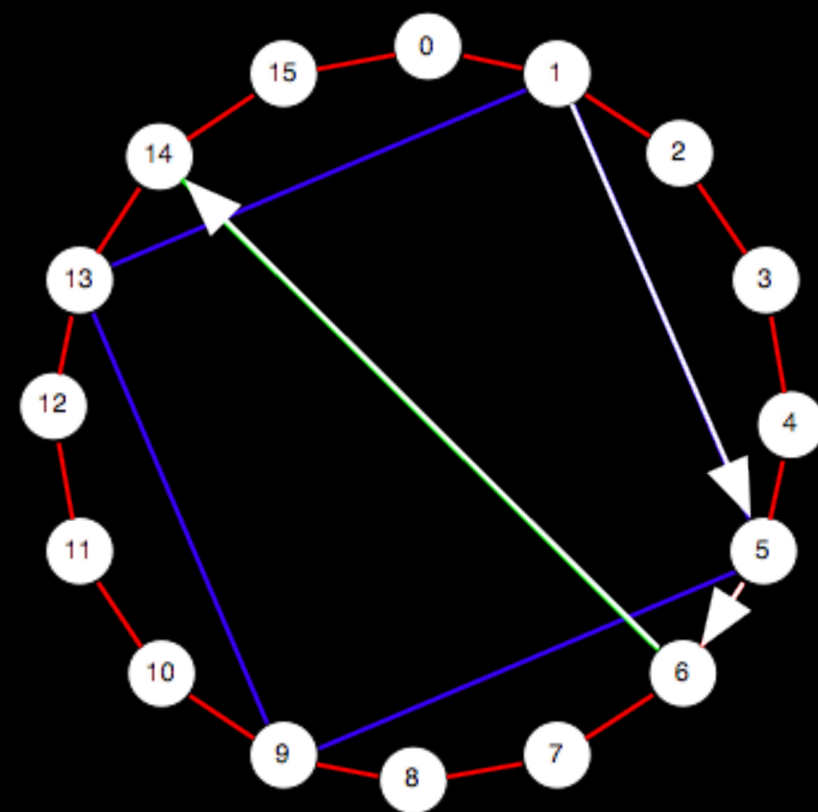
Routing in Chord

- At most one of each Gon
- E.g. 1-to-0



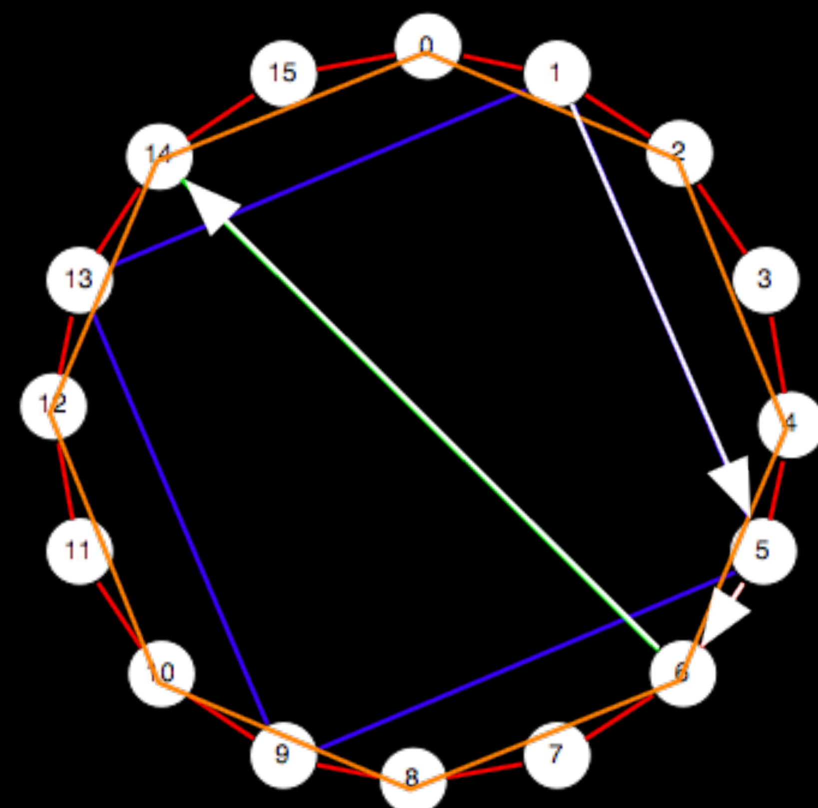
Routing in Chord

- At most one of each Gon
- E.g. 1-to-0



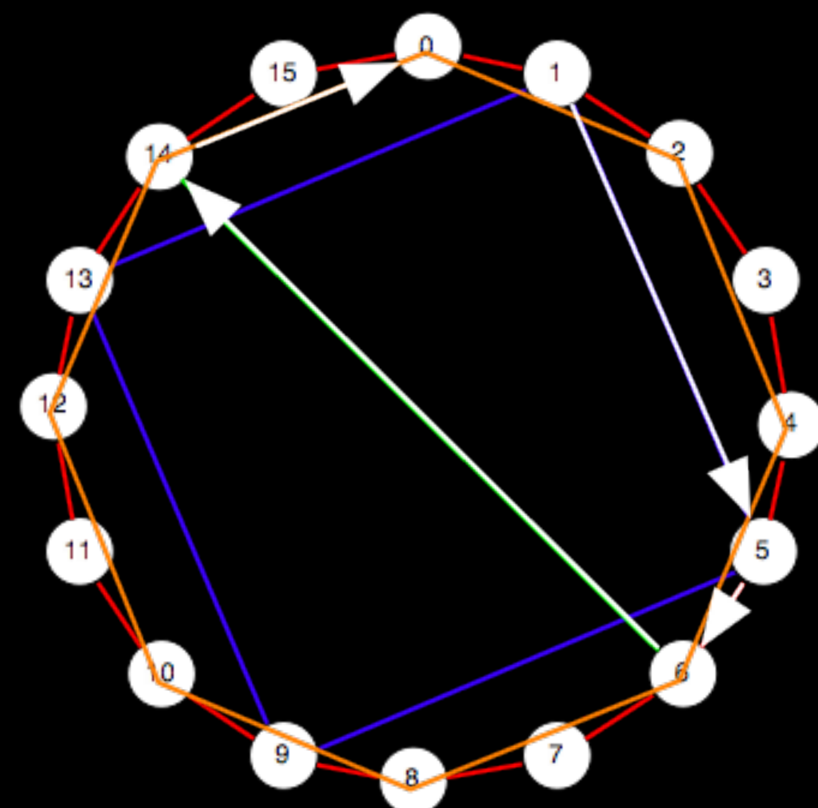
Routing in Chord

- At most one of each Gon
- E.g. 1-to-0



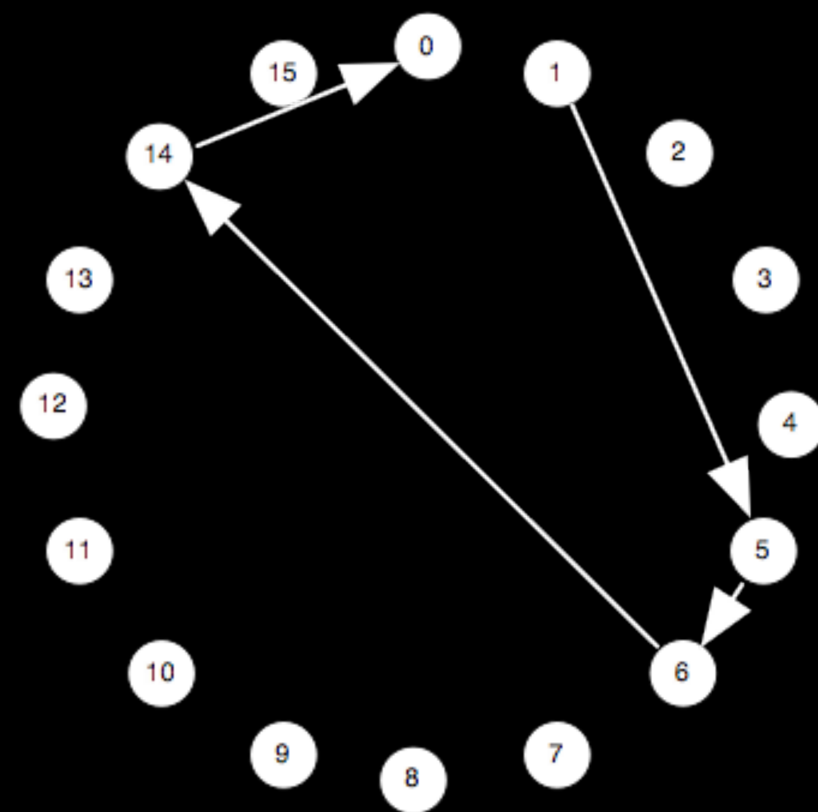
Routing in Chord

- At most one of each Gon
- E.g. 1-to-0



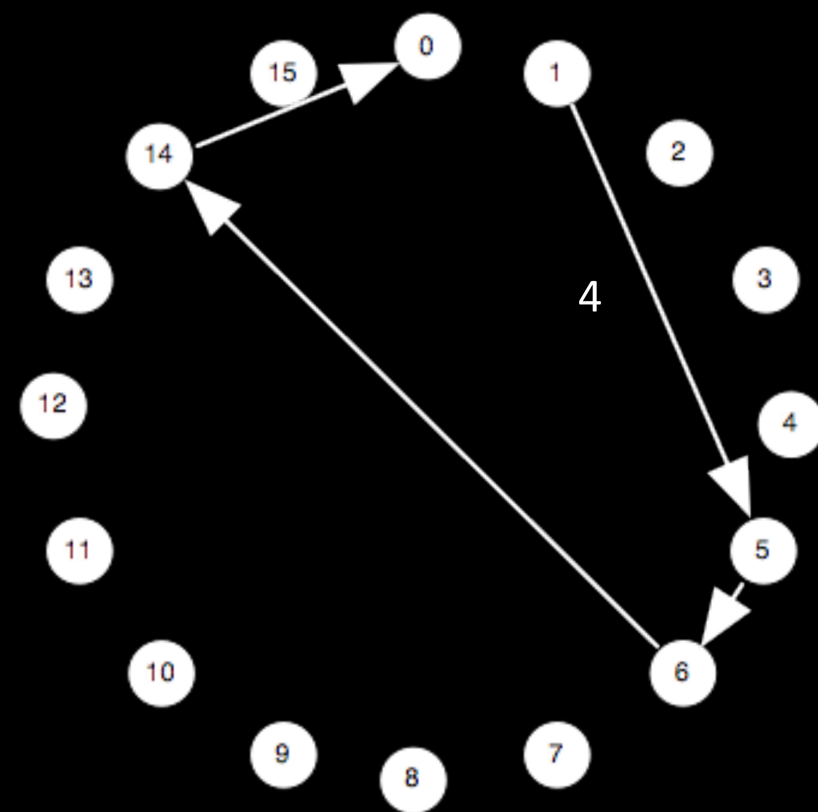
Routing in Chord

- At most one of each Gon
- E.g. 1-to-0
- What happened?
  - We constructed the binary number 15!
  - Routing from *x* to *y* is like computing *y* - *x* mod *n* by summing powers of 2



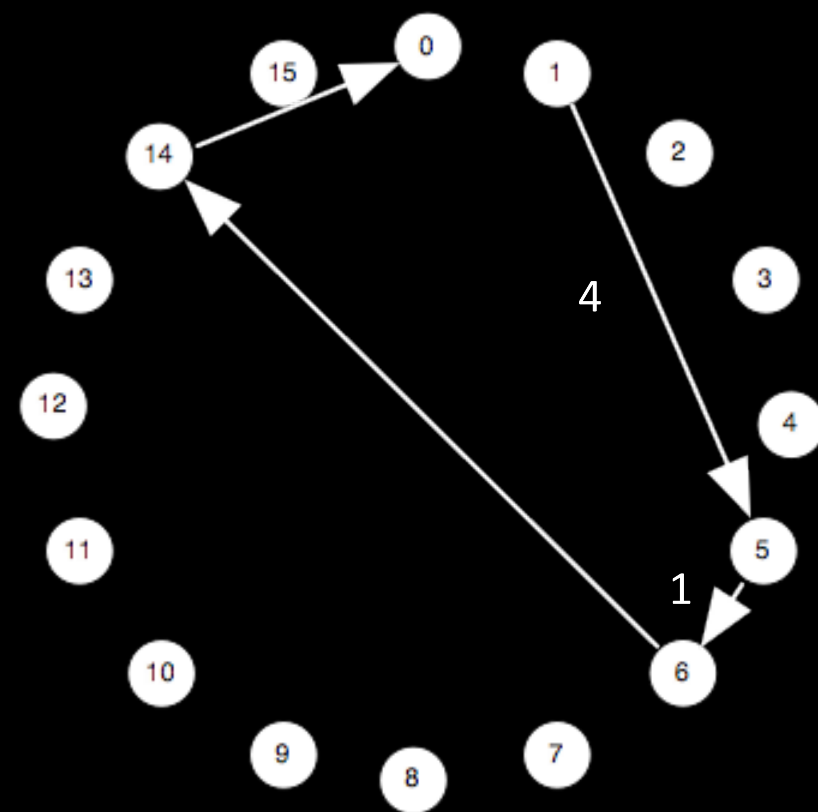
Routing in Chord

- At most one of each Gon
- E.g. 1-to-0
- What happened?
  - We constructed the binary number 15!
  - Routing from *x* to *y* is like computing *y* - *x* mod *n* by summing powers of 2



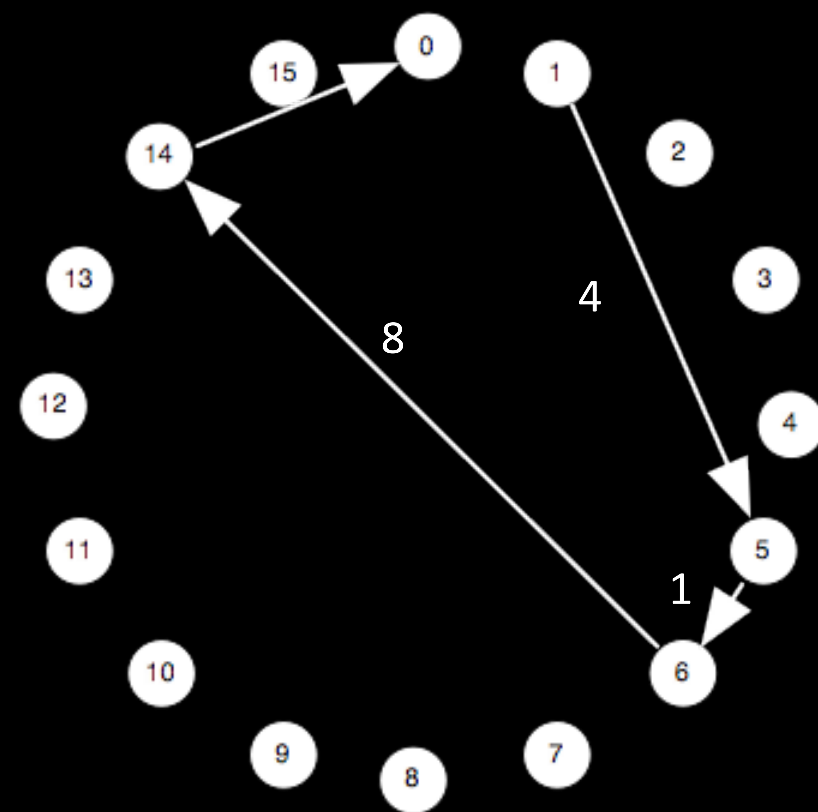
Routing in Chord

- At most one of each Gon
- E.g. 1-to-0
- What happened?
  - We constructed the binary number 15!
  - Routing from *x* to *y* is like computing *y* - *x* mod *n* by summing powers of 2



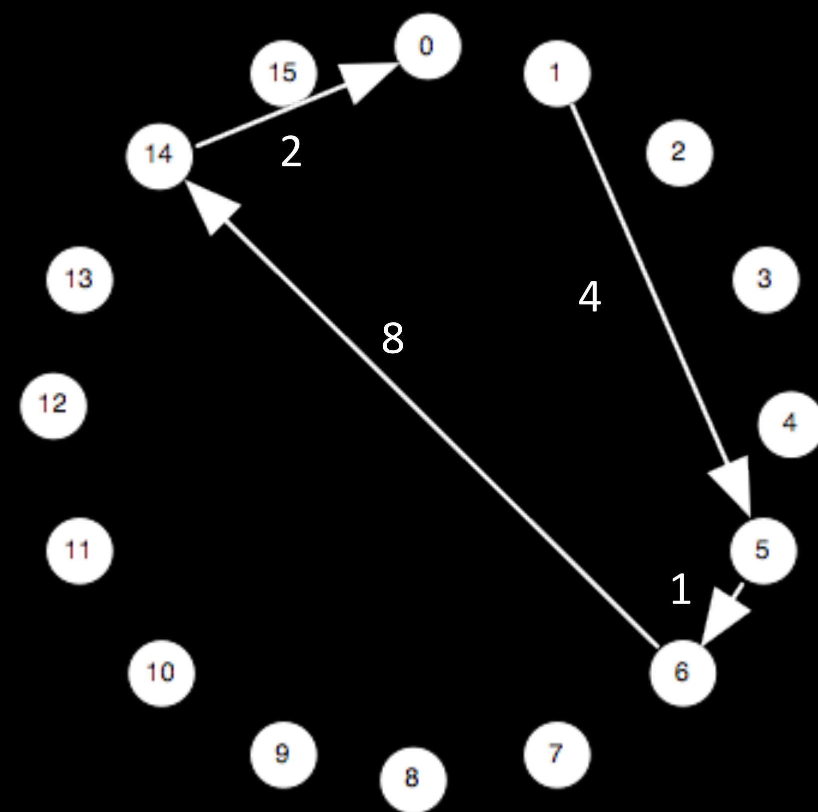
Routing in Chord

- At most one of each Gon
- E.g. 1-to-0
- What happened?
  - We constructed the binary number 15!
  - Routing from *x* to *y* is like computing *y* - *x* mod *n* by summing powers of 2



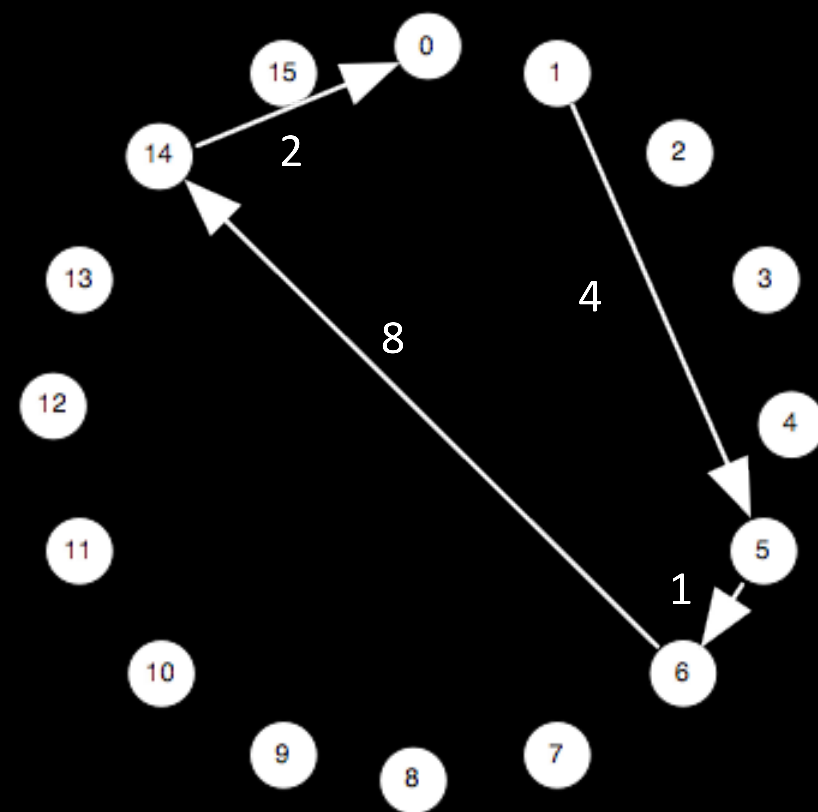
Routing in Chord

- At most one of each Gon
- E.g. 1-to-0
- What happened?
  - We constructed the binary number 15!
  - Routing from *x* to *y* is like computing *y* - *x* mod *n* by summing powers of 2



Routing in Chord

- At most one of each Gon
- E.g. 1-to-0
- What happened?
  - We constructed the binary number 15!
  - Routing from *x* to *y* is like computing *y* - *x* mod *n* by summing powers of 2



Diameter: log *n* (1 hop per gon type)
Degree: log *n* (one outlink per gon type)

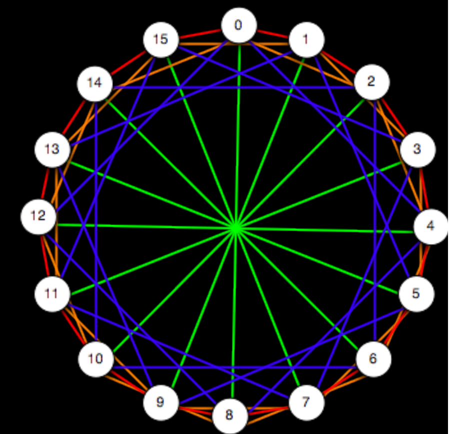
What is happening here? Algebra!

What is happening here? Algebra!

- Underlying group-theoretic structure
  - Recall a *group* is a set S and an operator $\bullet$ such that:
    - S is closed under $\bullet$
    - Associativity: $(AB)C = A(BC)$
    - There is an *identity* element $I \in S$ *s.t.* $IX = XI = X$ for all $X \in S$
    - There is an inverse $X^{-1} \in S$ for each element $X \in S$ *s.t.* $XX^{-1} = X^{-1}X = I$
- The *generators* of a group
  - Elements $\{g_1, \ldots, g_n\}$ s.t. application of the operator on the generators produces all the members of the group.
- Canonical example: $(Z_n, +)$
  - Identity is 0
  - A set of generators: {1}
  - A different set of generators: {2, 3}

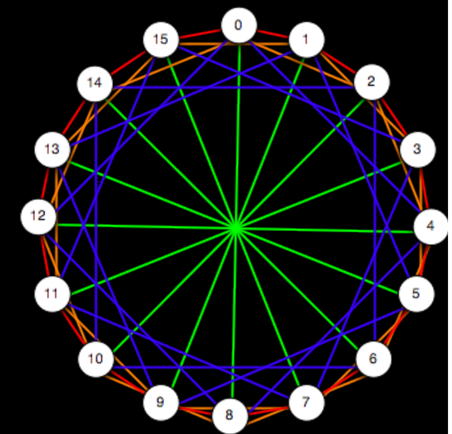
Cayley Graphs

- The *Cayley Graph (S, E)* of a group:
  - Vertices corresponding to the underlying set *S*
  - Edges corresponding to the *actions of the generators*
- (Complete) Chord is a Cayley graph for $(Z_n,+)$
  - $S = Z \bmod n$ ($n = 2^k$).
  - Generators $\{1, 2, 4, \ldots, 2^{k-1}\}$
  - That's what the gons are all about!
- Fact: Most (complete) DHTs are Cayley graphs
  - And they didn't even know it!
  - Follows from parallel InterConnect Networks (ICNs)
    - Shown to be group-theoretic [Akers/Krishnamurthy '89]



Cayley Graphs

- The *Cayley Graph (S, E)* of a group:
  - Vertices corresponding to the underlying set *S*
  - Edges corresponding to the *actions of the generators*
- (Complete) Chord is a Cayley graph for $(Z_n,+)$
  - $S = Z \bmod n$ ($n = 2^k$).
  - Generators $\{1, 2, 4, \ldots, 2^{k-1}\}$
  - That's what the gons are all about!
- Fact: Most (complete) DHTs are Cayley graphs
  - And they didn't even know it!
  - Follows from parallel InterConnect Networks (ICNs)
    - Shown to be group-theoretic [Akers/Krishnamurthy '89]

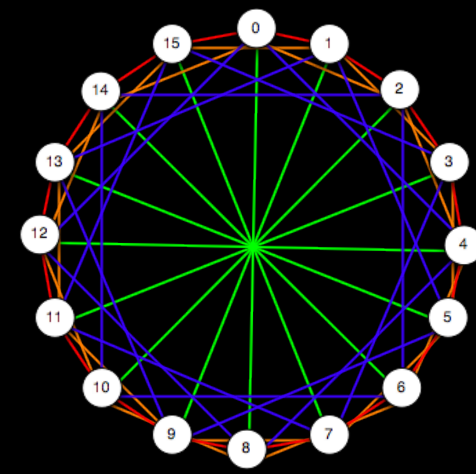


Note: the ones that aren't Cayley Graphs are *coset graphs*,
a related group-theoretic structure

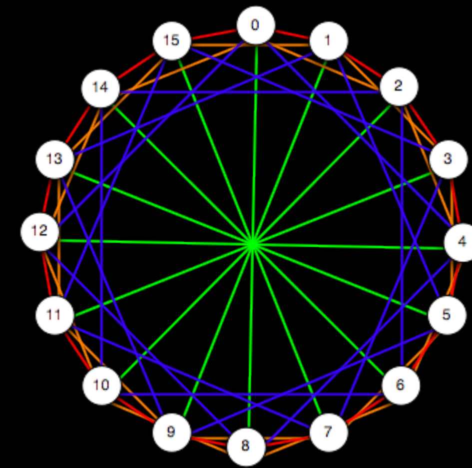
So…?

- Two questions:
  - How did this happen?
  - Why should you care?

How Hairy met Cayley

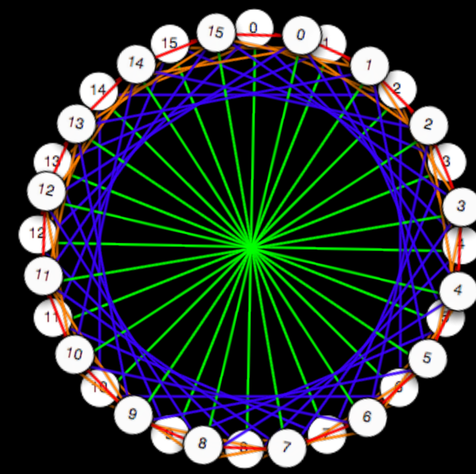


How Hairy met Cayley



- What do you want in a structured network?
  - Uniformity of routing logic
  - Efficiency/load-balance of routing and maintenance
  - Generality at different scales
- Theorem: All Cayley graphs are *vertex symmetric.*
  - I.e. isomorphic under swaps of nodes
  - So routing from *y* to *x* looks just like routing from (*y*-*x*) to 0
    - The routing code at each node is the same! Simple software.
    - Moreover, under a random workload the routing responsibilities (congestion) at each node are the same!
- Cayley graphs tend to have good degree/diameter tradeoffs
  - Efficient routing with few neighbors to maintain
- Many Cayley graphs are *hierarchical*
  - Made of smaller Cayley graphs connected by a new generator
    - E.g. a Chord graph on 2^{m+1} nodes looks like 2 interleaved (half-notch rotated) Chord graphs of 2^m nodes with half-notch edges
    - Again, code is nice and simple

How Hairy met Cayley



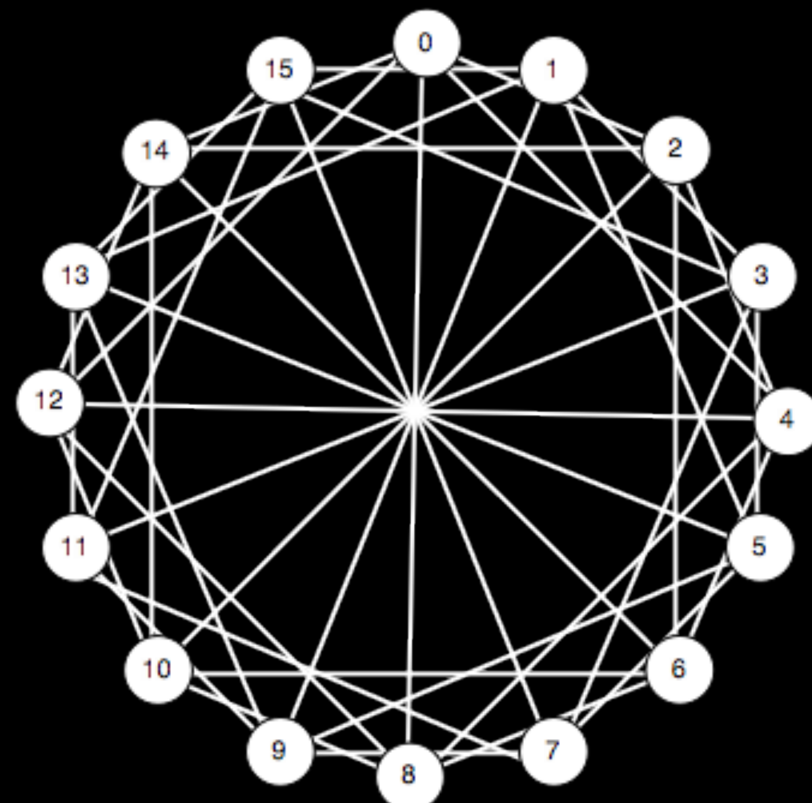
- What do you want in a structured network?
  - Uniformity of routing logic
  - Efficiency/load-balance of routing and maintenance
  - Generality at different scales
- Theorem: All Cayley graphs are *vertex symmetric.*
  - I.e. isomorphic under swaps of nodes
  - So routing from *y* to *x* looks just like routing from (*y*-*x*) to 0
    - The routing code at each node is the same! Simple software.
    - Moreover, under a random workload the routing responsibilities (congestion) at each node are the same!
- Cayley graphs tend to have good degree/diameter tradeoffs
  - Efficient routing with few neighbors to maintain
- Many Cayley graphs are *hierarchical*
  - Made of smaller Cayley graphs connected by a new generator
    - E.g. a Chord graph on 2^{m+1} nodes looks like 2 interleaved (half-notch rotated) Chord graphs of 2^m nodes with half-notch edges
    - Again, code is nice and simple

Upshot

- Good DHT topologies will be Cayley/Coset graphs
  - A replay of ICN Design
  - But DHTs can use funky “wiring” that was infeasible in ICNs
  - All the group-theoretic analysis becomes suggestive
- Clean math describing the topology helps crisply analyze efficiency
  - E.g. degree/diameter tradeoffs
  - E.g. shapes of distribution/aggregation trees
- Really no excuse to be “sloppy”

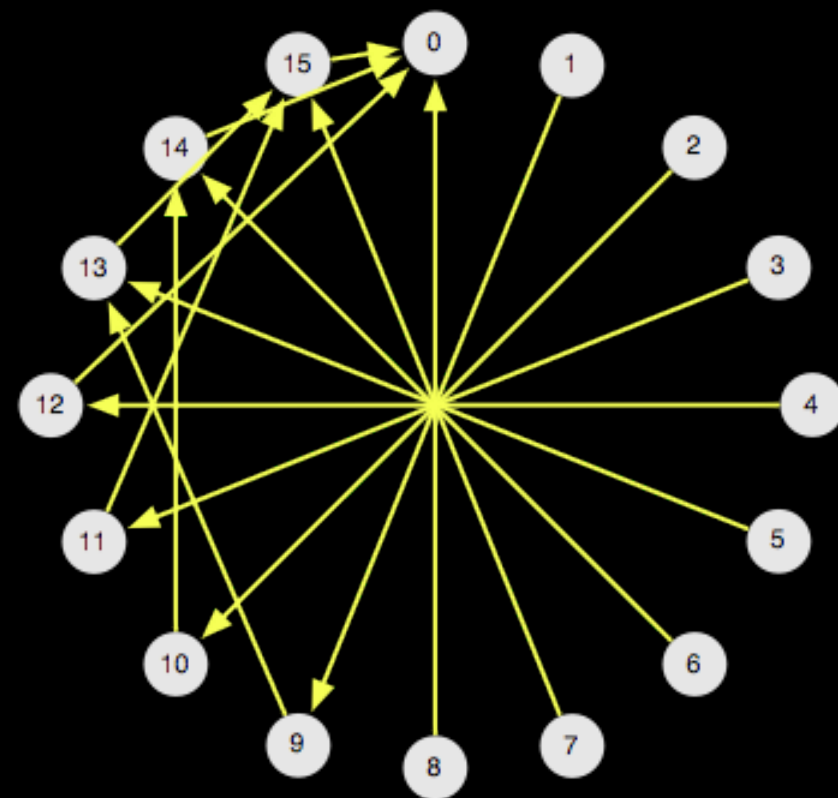
Consider Aggregation in Chord

- Everybody sends their message to node 0
- Assume greedy jumps (increasing Gon-order)
- Intercept messages and aggregate along the way



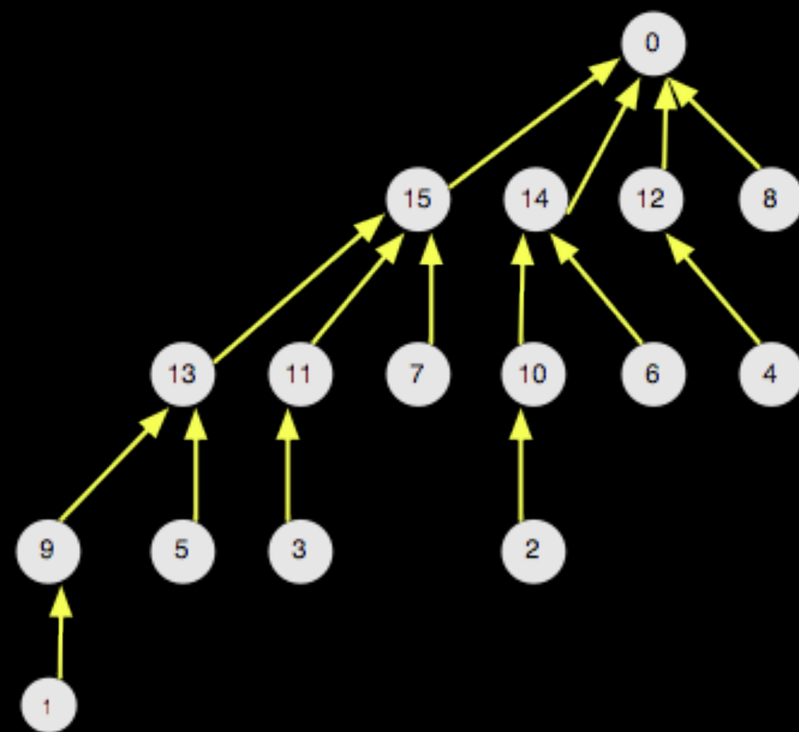
Consider Aggregation in Chord

- Everybody sends their message to node 0
- Assume greedy jumps (increasing Gon-order)
- Intercept messages and aggregate along the way



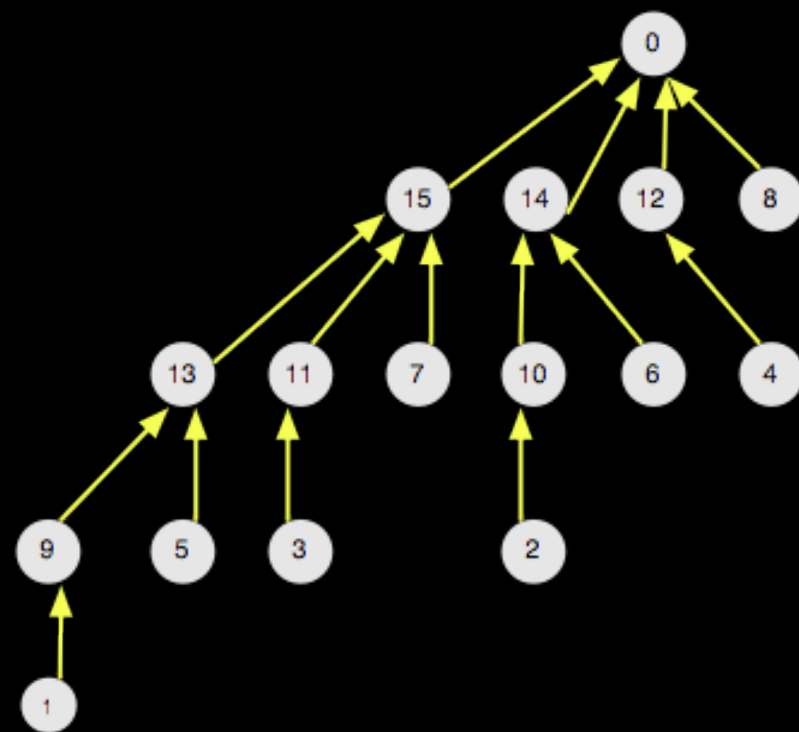
Consider Aggregation in Chord

- Everybody sends their message to node 0
- Assume greedy jumps (increasing Gon-order)
- Intercept messages and aggregate along the way



Consider Aggregation in Chord

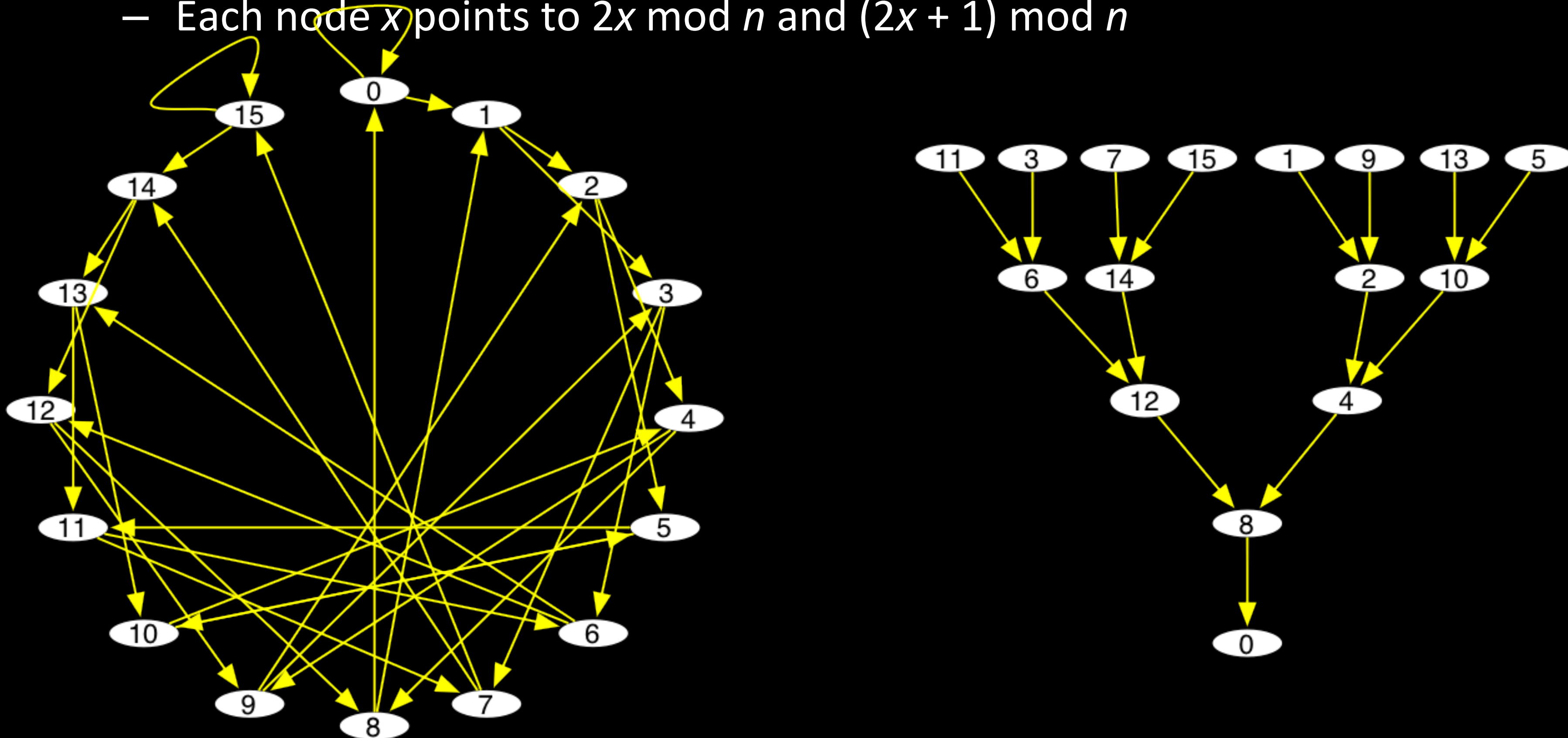
- Everybody sends their message to node 0
- Assume greedy jumps (increasing Gon-order)
- Intercept messages and aggregate along the way



Binomial Tree!!

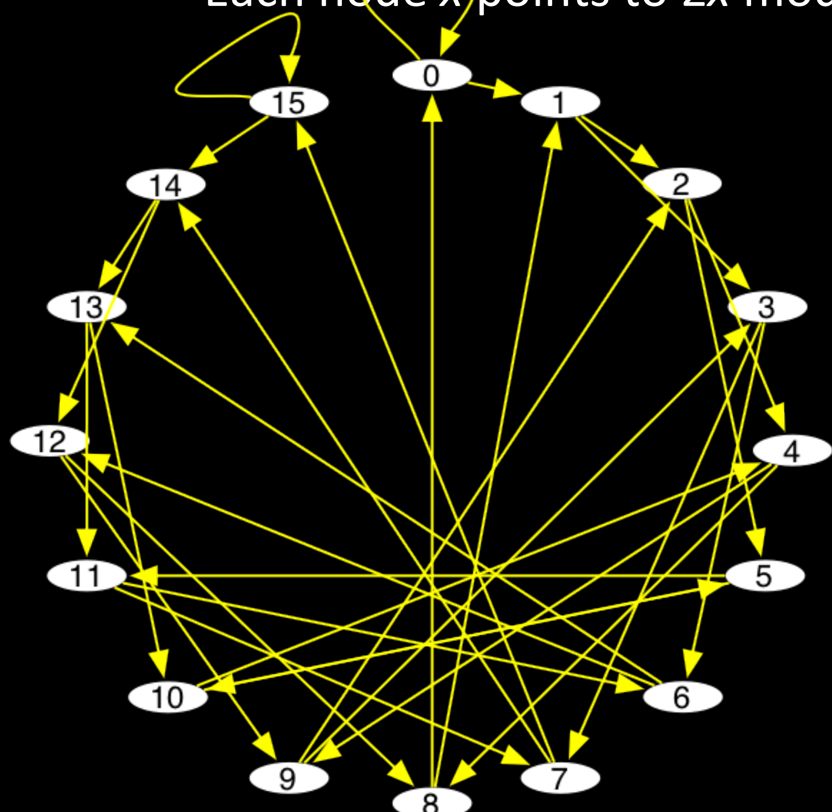
Aggregation in Koorde

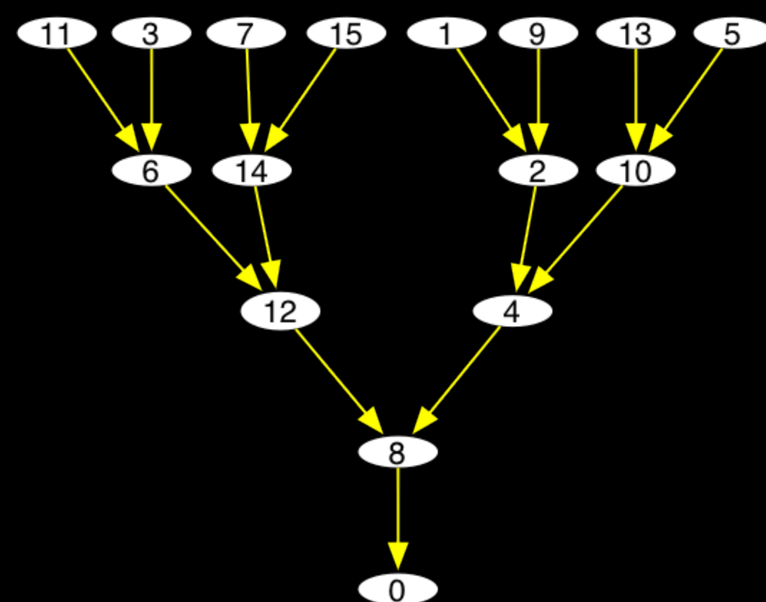
- Recall the DeBruijn graph:
  - Each node x points to $2x \bmod n$ and $(2x + 1) \bmod n$



Aggregation in Koorde

- Recall the DeBruijn graph:
  - Each node x points to $2x$ mod n and $(2x + 1)$ mod n





(But note: not node-symmetric)