Week 13 – Monday

COMP 3400

Last time

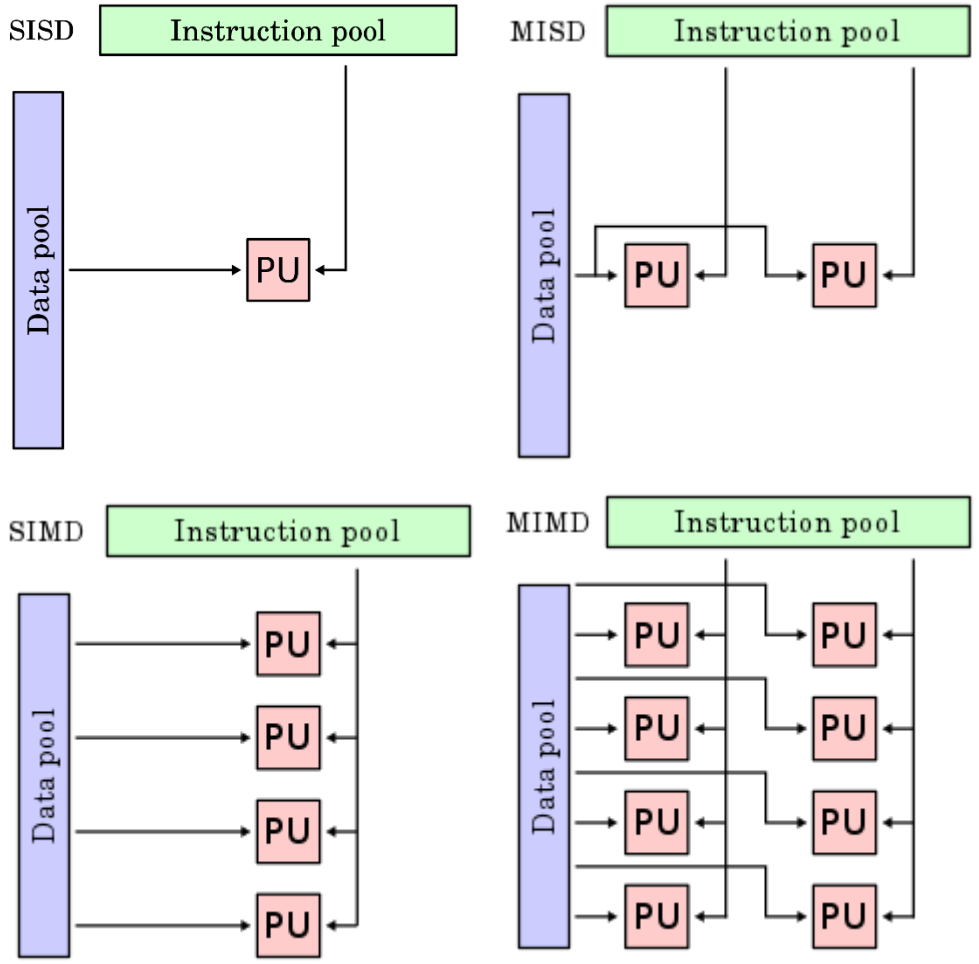
- What did we talk about last time?
- Parallelism vs. concurrency
  - Task parallelism
  - Data parallelism
- Parallel algorithmic strategies
  - Embarrassingly parallel
  - Divide and conquer
  - Pipelines
- Parallel implementation strategies
  - Fork/join
  - Map/reduce
  - Manager/worker

Questions?

Assignment 7

Flynn's taxonomy

- Flynn's taxonomy divides hardware into how they can deal with multiple instructions and multiple pieces of data
  - **Single Instruction Single Data (SISD)** is sequential processing of one piece of data with one instruction
  - **Single Instruction Multiple Data (SIMD)** is processing several pieces of data with the same instruction, like the vector processing done in graphics cards
  - **Multiple Instruction Single Data (MISD)** isn't used commonly, but it can allow for fault-tolerance because different instructions are executed in parallel on the same data
  - **Multiple Instruction Multiple Data (MIMD)** is processing different instructions on different data at the same time



Images from Wikipedia

Limits of Parallelism

Speedup

- **Speedup** is how much faster a parallel solution is compared to a sequential one
- The formula is $\frac{T_{sequential}}{T_{parallel}}$
  - $T_{sequential}$ is the amount of time the sequential solution takes
  - $T_{parallel}$ is the amount of time the parallel solution takes
- Thus, if a sequential solution to a problem takes 100 seconds, and the parallel solution takes 50 seconds, the speedup is 2

Limits of parallelism

- The study of parallel processing is, unfortunately, filled with bad news
- A parallel program running on ***n*** processors can never run more than ***n*** times faster than a well-written program for 1 processor
- Usually, running a parallel program on ***n*** processors is nowhere close to running ***n*** times faster
  - What is called **linear** speed-up

Amdahl's Law and strong scaling

- What if you had 16 cores? Or 1,000 cores? Or a million?
- How much speedup can you get?
- Some part of the program has to be executed sequentially
  - Reading input
  - Starting threads
  - Combining results
- Amdahl's law says that the maximum speedup possible is $\frac{1}{(1-p)+\frac{p}{N}}$
  - p is the fraction of a program that can be parallel
  - N is the number of processors

Consequences of Amdahl's law

- What if we had unlimited cores?
- We can take the equation $\frac{1}{(1-p)+\frac{p}{N}}$ and plug in ∞ for N
- Doing so would mean, even with infinite cores, we could never have better speedup than $\frac{1}{(1-p)}$
- Let's say that 90% of a program can be parallelized
- What's the maximum possible speedup you can get?
- $S = \frac{1}{(1-p)} = \frac{1}{(1-.9)} = \frac{1}{.1} = 10$

Gustafson's Law and weak scaling

- Unfortunately, Amdahl's Law makes the unrealistic assumption that there's no extra overhead for creating more threads
  - This assumption is called **strong scaling**
- Gustafson's Law tries to take a more realistic approach by letting speedup be $S = 1 - p + sp$
  - p is the percentage of work that can benefit from some improvement in execution (not just parallelism)
  - s is the amount of improvement
- In Gustafson's Law, speedup means how much more data can be processed in the same amount of time
  - This approach is called **weak scaling**

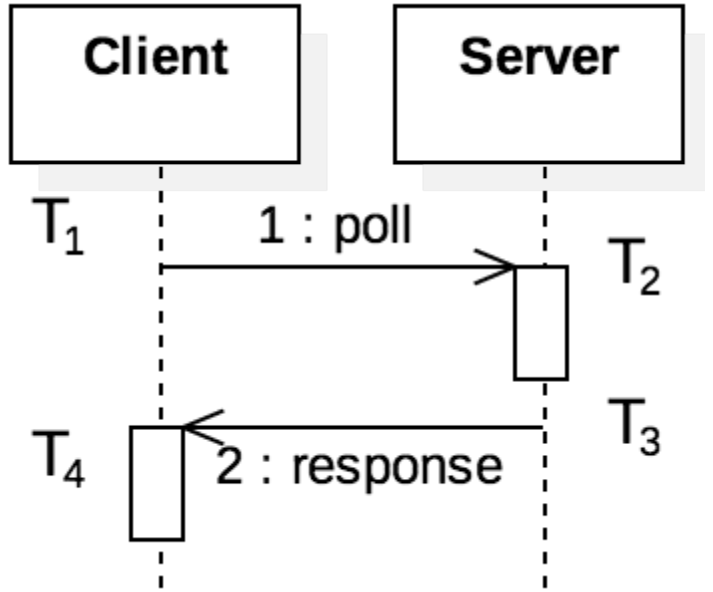
Timing in Distributed Environments

Timing in distributed environments

- When working on a single computer, there's only one clock
- Thus, multiple threads can use this clock to record events in a mutually consistent way
  - Like adding timestamps to log files
- Distributed systems don't have a single, reliable clock
  - Each computer might have a slightly (or completely) different time
  - Clocks on each computer drift with respect to each other
  - These problems get worse as distance (and network delays) increase

Clock synchronization

- We can synchronize clocks based on a centralized server
- A problem is that the time a message takes in the network is unpredictable
- Network Time Protocol (NTP) is a protocol to do this:
  - Client sends a message at T_1
  - Server receives the message at T_2
  - Server replies at T_3
  - Client receives the message at T_4
- Offset = $\frac{(T_2 - T_1) + (T_3 - T_4)}{2}$
  - The offset is a measurement of the difference in times between the client and server
- Delay = $(T_4 - T_1) - (T_3 - T_2)$
  - The delay is a measurement of how long it takes for the messages to make a round trip
- Algorithms process a number of offset and delay values to try to find the most accurate offset



Logical clocks

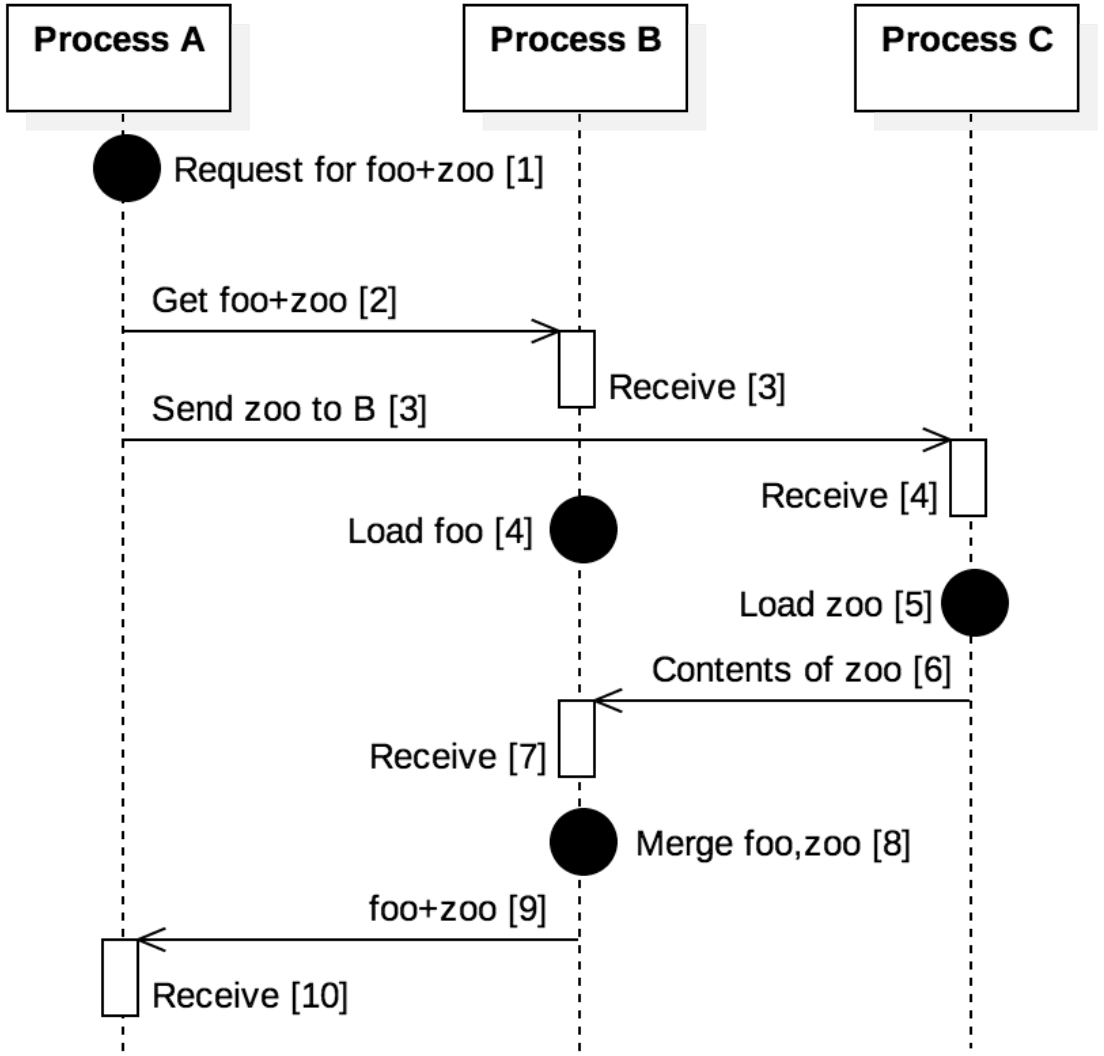
- Large systems can't effectively use protocols like NTP
  - There are too many nodes to synchronize
  - The number of messages needed to synchronize becomes large
- Logical clocks are an alternative system using messages to track the order of events
- We're only trying to know the sequence of events, not their exact times

Lamport timestamps

- Lamport timestamps are one way to implement logical clocks
  - Named after Leslie Lamport, of LaTeX fame
- Each process keeps an internal counter of events that it sees
  - When a local event occurs, the counter is incremented
  - When a process sends or receives a message, it increments its counter
- Messages have timestamps
  - When a process receives a message, it updates its internal counter to the message's timestamp if that timestamp is larger

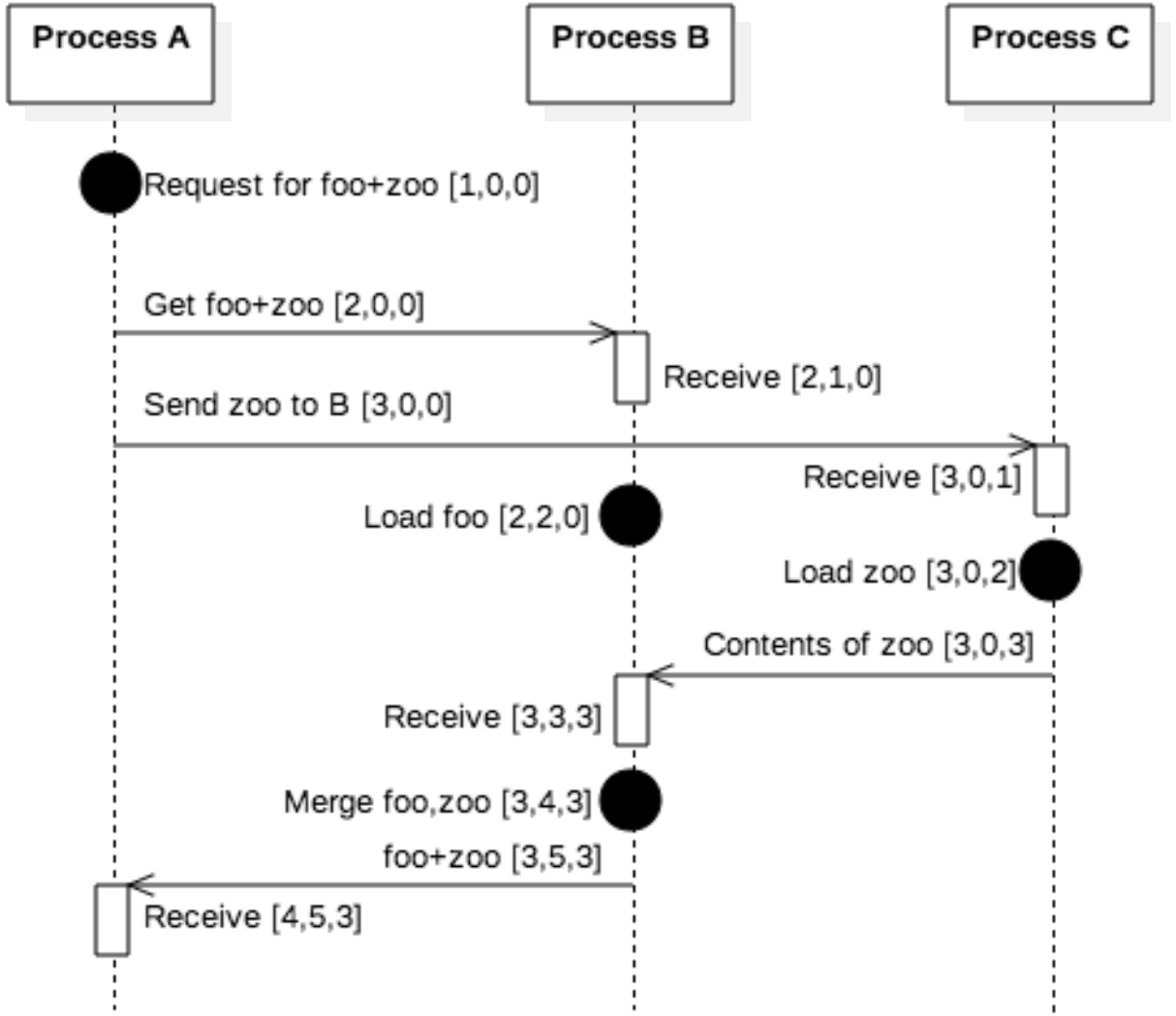
Example of Lamport timestamps

- Consider a cloud system where requests can be made for files
- Process A gets a request for the files "foo" merged with "zoo"
- Timestamps are updated as messages flow through the system
- Timestamps are purely relative and have no meaning to processes not involved in the exchanges



Vector clocks

- Lamport timestamps only give indirect information about the state of other processes
- **Vector clocks** extend the idea of Lamport timestamps by making every process keep a counter for every process
- When a message from one process arrives, the receiving process can update all of its counters based on whatever is larger
- Vector clocks give much more information about how many events have been experienced by other processes



Reliable Storage and Location

Reliable data storage

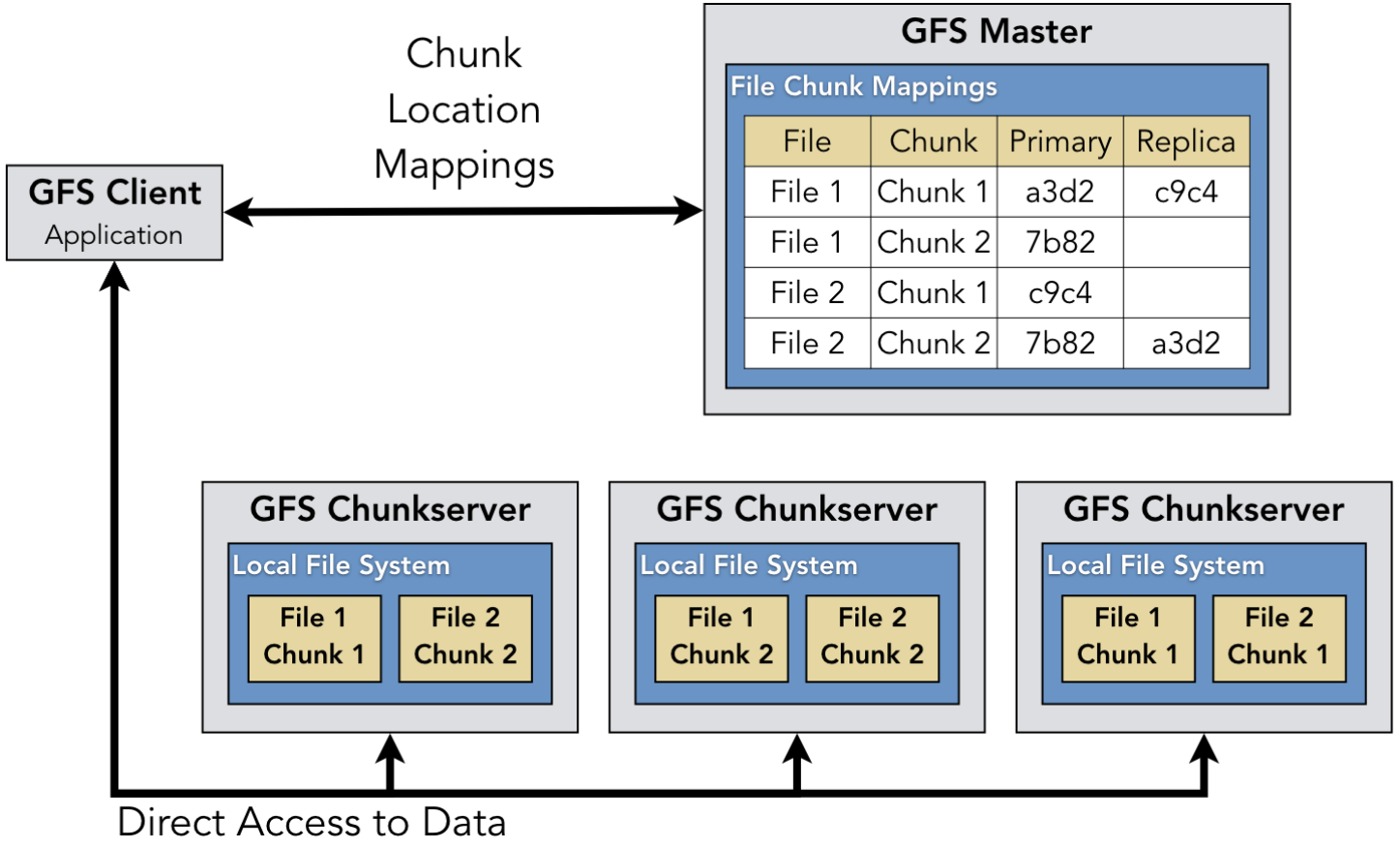
- If you want to get a file from a web server, you can go to a URL and make an HTTP request
- Unfortunately, if that server is down or unreachable, you can't get the file
- For this reason, distributed systems are often used to store data
- A key feature of distributed data storage is **replication**, keeping multiple copies of the same data
  - Replication avoids a single point of failure
  - If done correctly, replication can also do load balancing, improving performance by providing multiple sources for data

Google File System

- The Google File System (GFS) is a distributed storage system
- GFS was designed to store Google's internal data, like the data structures used for PageRank
- Files are often large, so they're broken into chunks
- Chunks are stored on chunkservers as regular files
- A master server stores a table mapping file chunks to their locations

Illustration of GFS

- Each chunk has a primary chunkserver as well as replicas
- The chunks are identical, but the primary chunkserver is the only place where the chunk can be modified
  - It propagates changes to the other chunkservers
  - This redundancy makes writing to GFS slower, even though reading is relatively fast
- The master server periodically sends messages to the chunkservers to get their current status

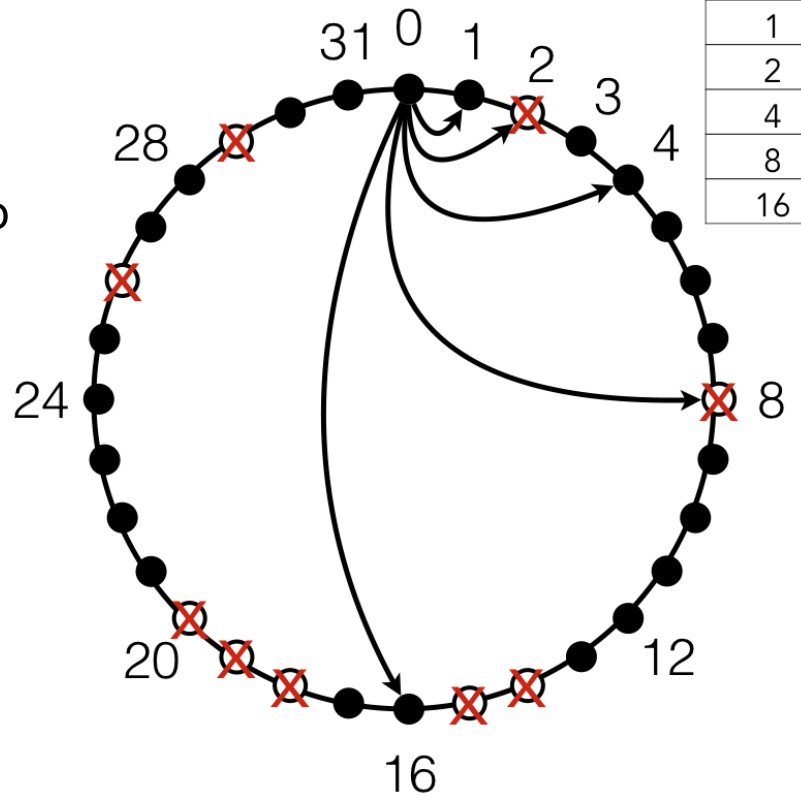


Distributed hash tables

- GFS was designed by Google for its own purposes
  - It uses a central server
  - Servers keep information about each other
- What if we have no idea what servers are going to be in the network?
- **Distributed hash tables (DHT)** are an approach for mapping arbitrary objects to arbitrary servers
- DHTs are a way to organize a peer-to-peer network to avoid query flooding

Chord DHT

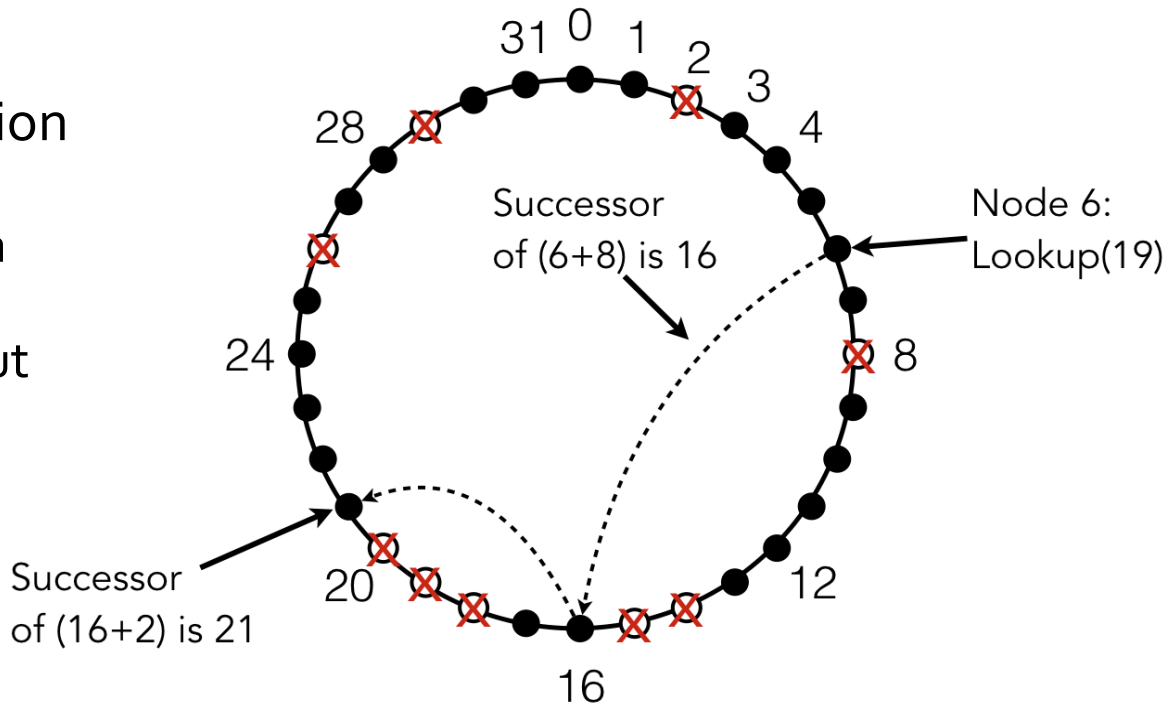
- Chord was one of the first algorithms for a DHT, introduced in 2001
- Each node has a unique identifier (often its IP address) that's hashed to provide a location in a circle
  - If the hash is ***n*** bits long, the DHT can support up to 2^n nodes
- Most locations in the circle are empty
- Each node has a "finger table," tracking successor elements in increasing powers of 2 away on the circle
  - If the power of 2 node is missing, it tracks the next non-missing node
- The example on the right is only for $2^5 = 32$ nodes



Hop	Successor(0+hop)
1	1
2	3
4	5
8	9
16	16

Files in Chord DHT

- When a file is added, it's hashed
- Whichever node has that hash value (or is its successor) is the location of that file
- On the right, node 6 is looking for a file at location 19 (the successor of 18)
  - It looks at 6 + 8 = 14, which doesn't exist but has a successor of 16
  - Then it looks at 16 + 2 = 18, which doesn't exist but has a successor of 21
  - Node 21 is where the file is supposed to be
- The details get a little more complex, but the practical result is that a file can be found with O(log ***n***) requests, where ***n*** is the size of the network
- Replication is done by caching files at nodes that were part of the lookup to find the file



Upcoming

Next time…

- Finish reliable storage
- Consensus in distributed systems
- Blockchain

Reminders

- Work on Assignment 7
  - **Due Thursday**
- Read sections 9.6, 9.7, and 9.8