

Computer Science 226, Advanced Database Systems (3 units)



Class Instructor:

David Ruby

Class Hours:

TTh 5:30 – 6:45

Office:

Science II – 273

Email:

druby@csufresno.edu

Database Systems: The Complete Book(3rd)
by Hector Garcia-Molina,
Jeffrey D. Ullman & Jennifer Widom

DATABASE
SYSTEMS
THE
COMPLETE
BOOK



Parallel & Distributed Databases

Chapter 20

- PART IV: Database System Implementation
 - 13 Secondary Storage Management
 - 14 Index Structures
 - 15 Query Execution
 - 16 The Query Compiler
 - 17 Coping w/ System Failures
 - 18 Concurrency Control
 - 19 More About Transaction Management
 - 20 Parallel & Distributed Databases

Parallel & Distributed Databases

Chapter 20

- 20.1 Parallel Algorithms on Relations
- 20.2 Map/Reduce Parallelism
- 20.3 Distributed Databases
- 20.4 Distributed Query Processing
- 20.5 Distributed Commit
- 20.6 Distributed Locking
- 20.7 Peer-to-Peer Distributed Search

Parallel & Distributed Databases

Chapter 20

- **20.1 Parallel Algorithms on Relations**
- **20.2 Map/Reduce Parallelism**
- 20.3 Distributed Databases
- 20.4 Distributed Query Processing
- 20.5 Distributed Commit
- 20.6 Distributed Locking
- **20.7 Peer-to-Peer Distributed Search**

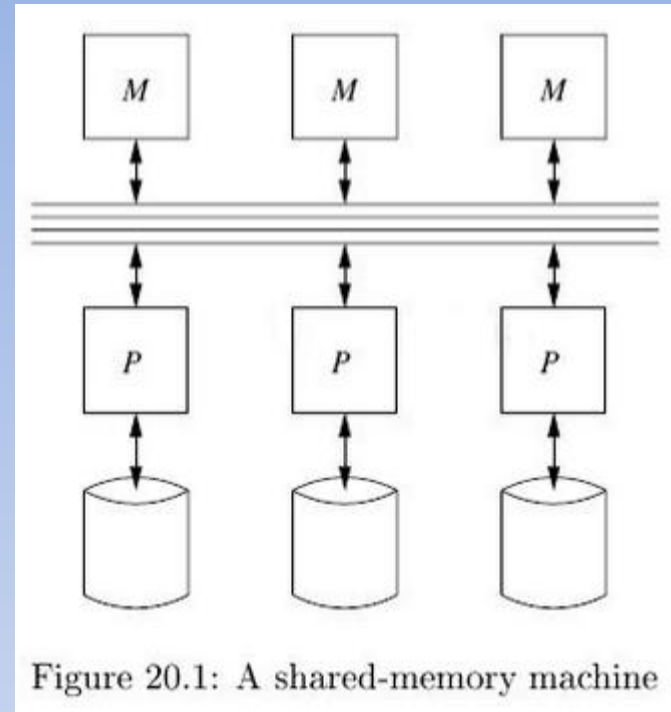
Parallel Computing

- Not surprising Parallel Computing for Databases is Important
 - Big Data
 - NoSQL
- Explore map-reduce paradigm
 - Hadoop
 - Spark
- Explore different parallel architectures
- Peer-to-Peer Networks

Good News

- Good news is most Database operations respond well to parallelism
- Simple modifications to most standard algorithms enable exploiting parallelism:
 - Time to complete operation on a p-processor machine is about $1/p$ of the time required on single processor machine!!

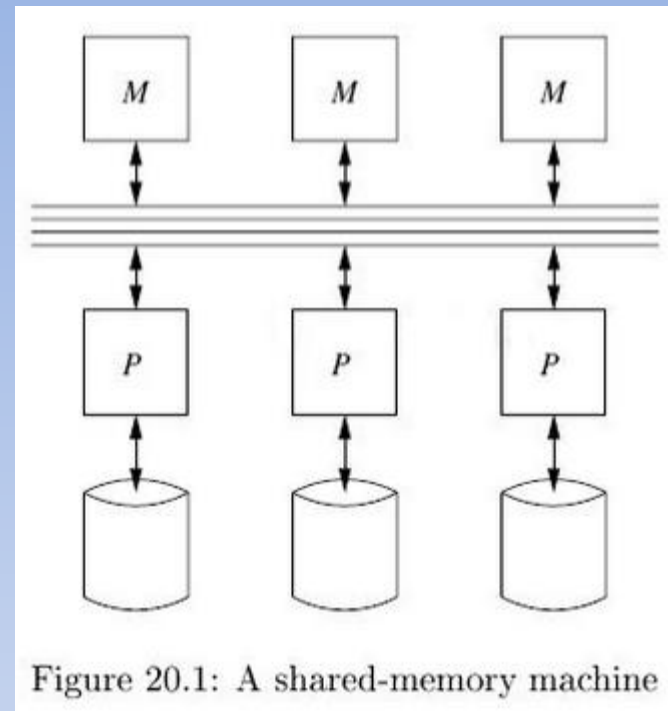
Parallel Computing w/ Shared-Memory Machine



- Parallel Architectures classified by how tightly coupled the system resources
- Most tightly coupled system w/ Shared-Memory Machine

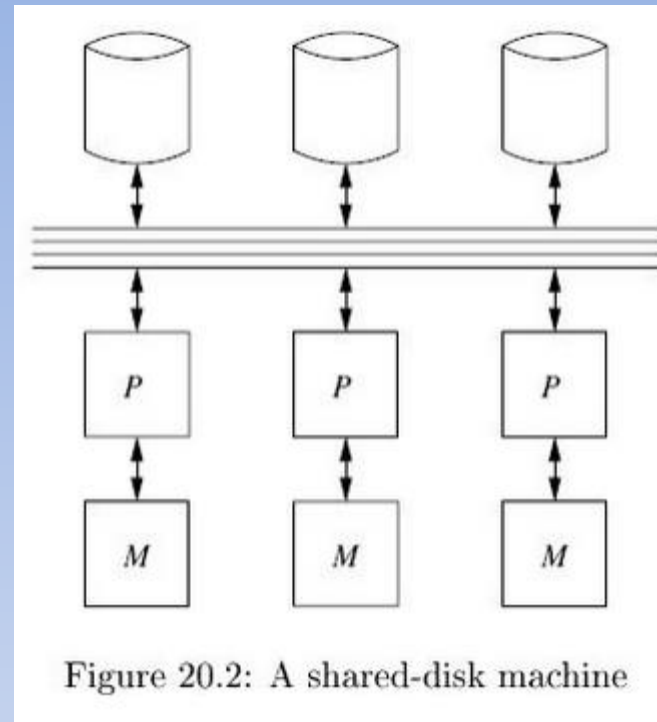
Parallel Computing w/ Shared-Memory Machine

- Each processor P has access to all memory M .
 - Single address space
- Memory access time w/ vary
 - Each processor will have some private memories.
 - Accessing memories of other processors require slightly more time.
 - NUMA (nonuniform memory access)

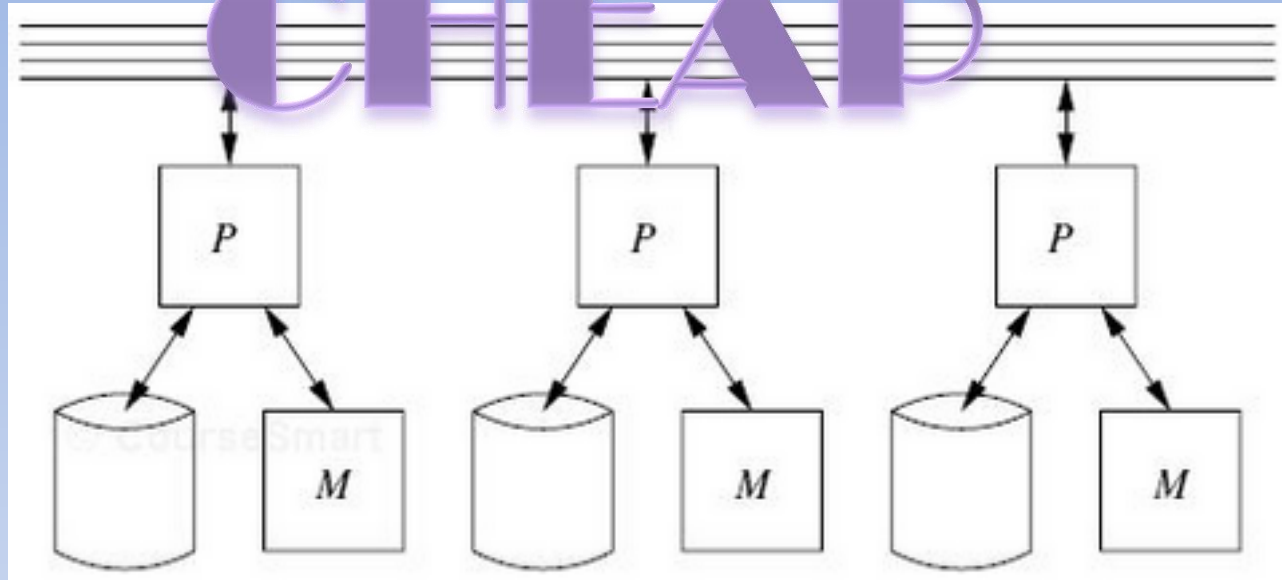


Parallel Computing w/ Shared-Disk Machine

- Slightly less coupled:
- Systems share secondary storage
- BUT keep memories private
- Disk Farms:
 - Network attached storage (NAS)
 - Store and transfer files
- Storage Area Networks
 - Store and transfer disk blocks to and from processors!

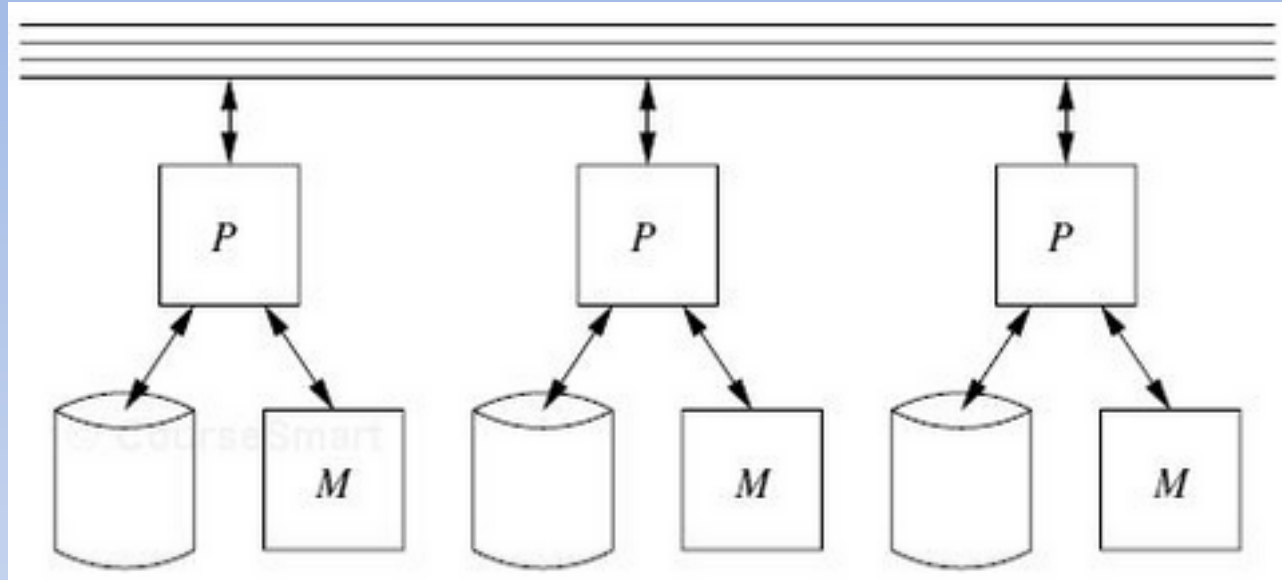


Parallel Computing w/ Shared-Nothing



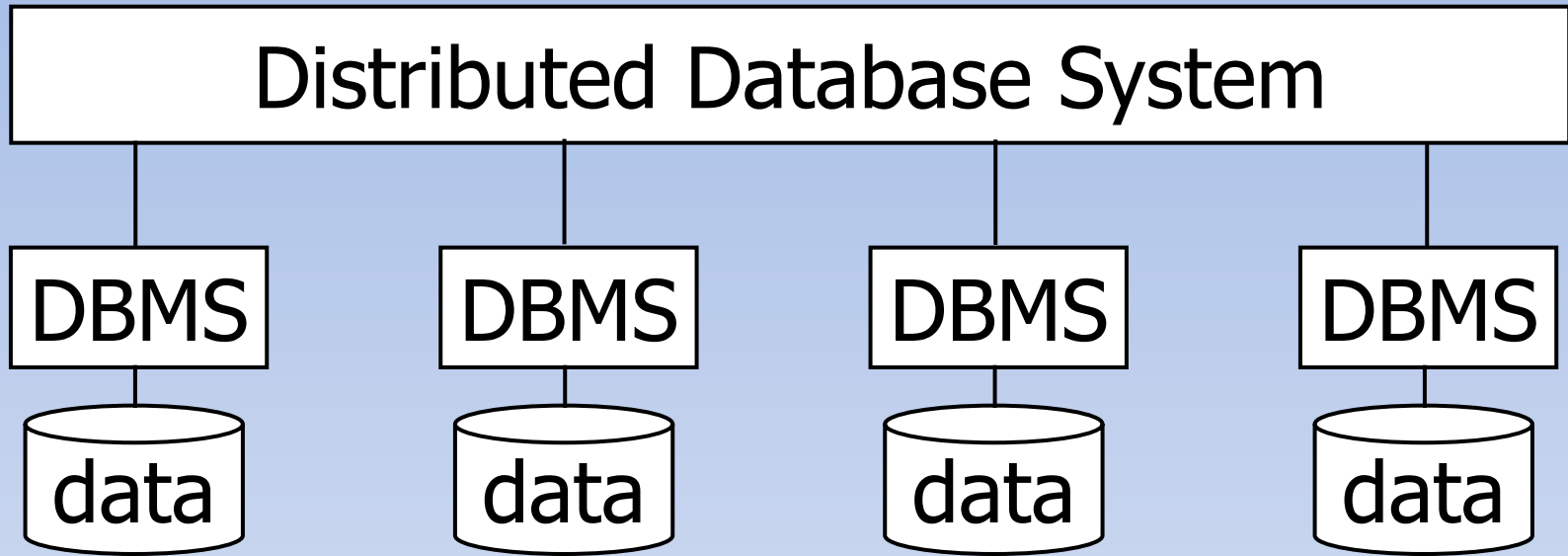
- Most commonly used for DB Systems!
- COMMODITY Machines mounted in Racks
Connected w/ Internal Network
 - Multiple racks connected w/ external network

Parallel Computing w/ Shared-Nothing



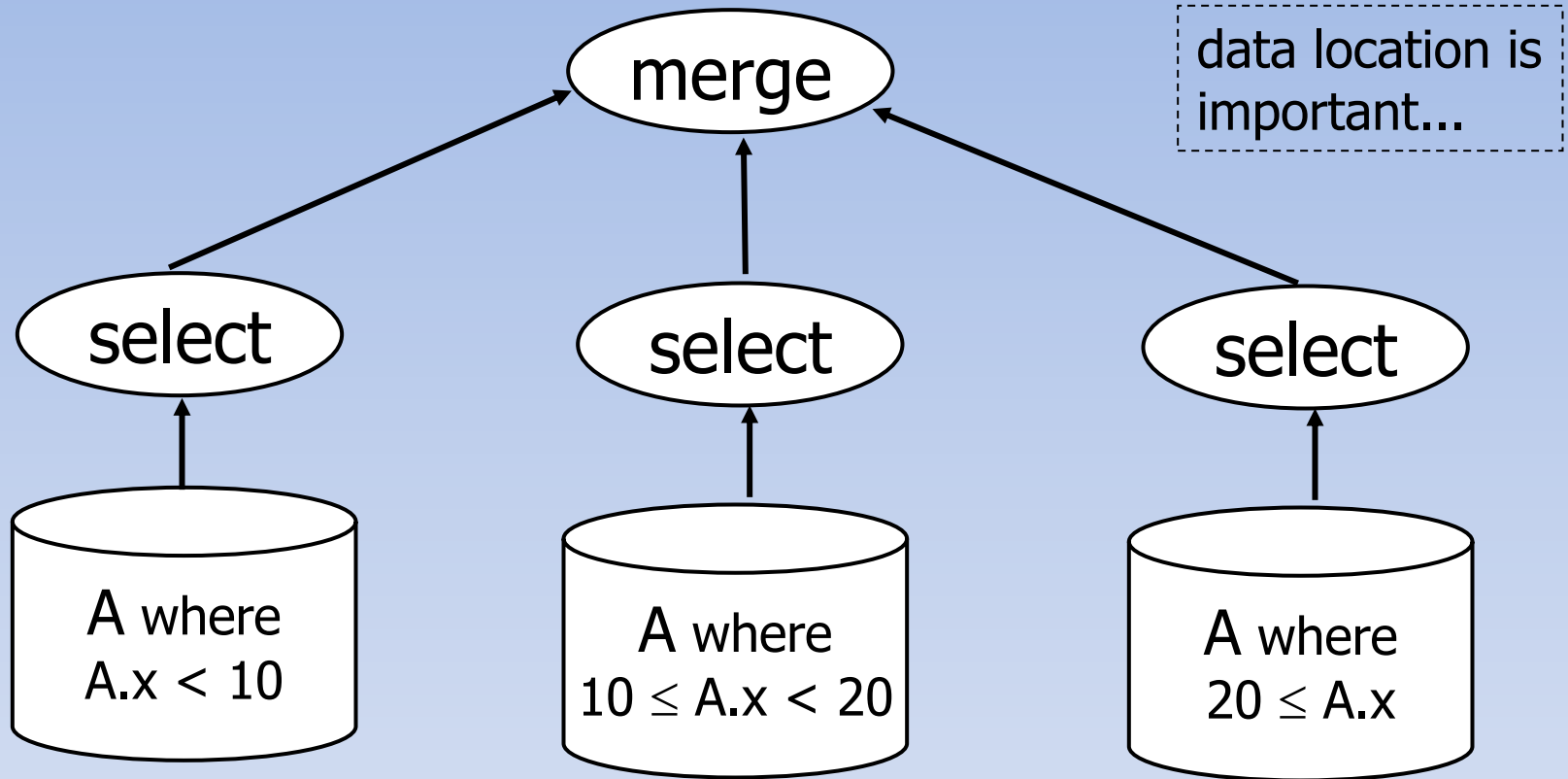
- Processors exchange messages w/ other processors.
 - i.e., asking for tuples from disk.
- Algorithm design must account for communication delay!
 - Algorithms frequently take advantage of long messages!

DBMS w/ Shared-Nothing



- Data distributed across multiple machines

Select * from A where cond;



- Data is distributed across multiple machines

select * from A where a=10

- If all values for a=10 are on same machine, THEN parallelism reduced.
- HASH function needs to incorporate all attribute values from tuple!
 - Changing one component of a tuple t can change $h(t)$ to be any possible bucket number
- Possible Hash:
 - Convert each component to int in $0..b-1$.
 - Add ints and divide by B
 - Use remainder as Bucket Number!

Operations on Multiple Relations

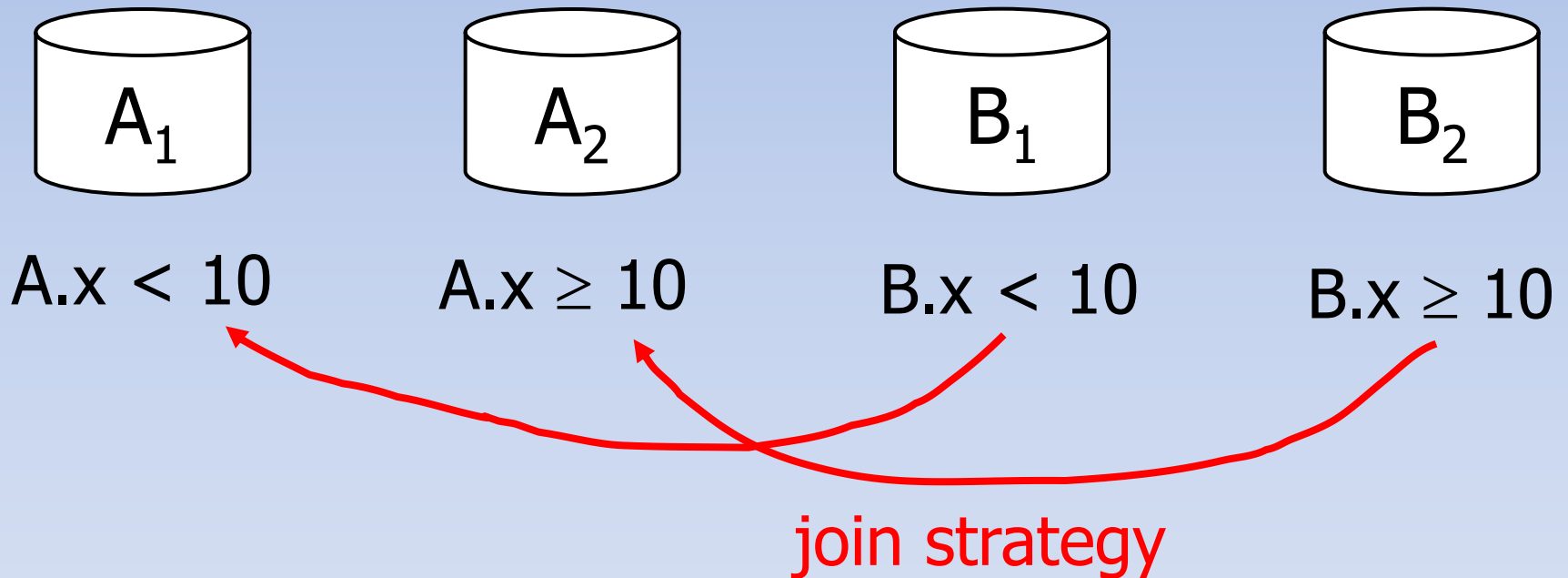
- Operations like union, intersection, difference of two relations R & S .
 - $R(X, Y), S(X, Y)$
- Key idea is to HASH them using the exact same HASH function.
- Perform ops in parallel, THEN merge result!
- If relations are not hashed w/ same function:
 - Re hash both in parallel using same function.

Joins

- Idea: Hash tuples using only the join attributes.
- CONSIDER:
 - $A(x, z), B(x, z)$
 - Select * from A natural join B;

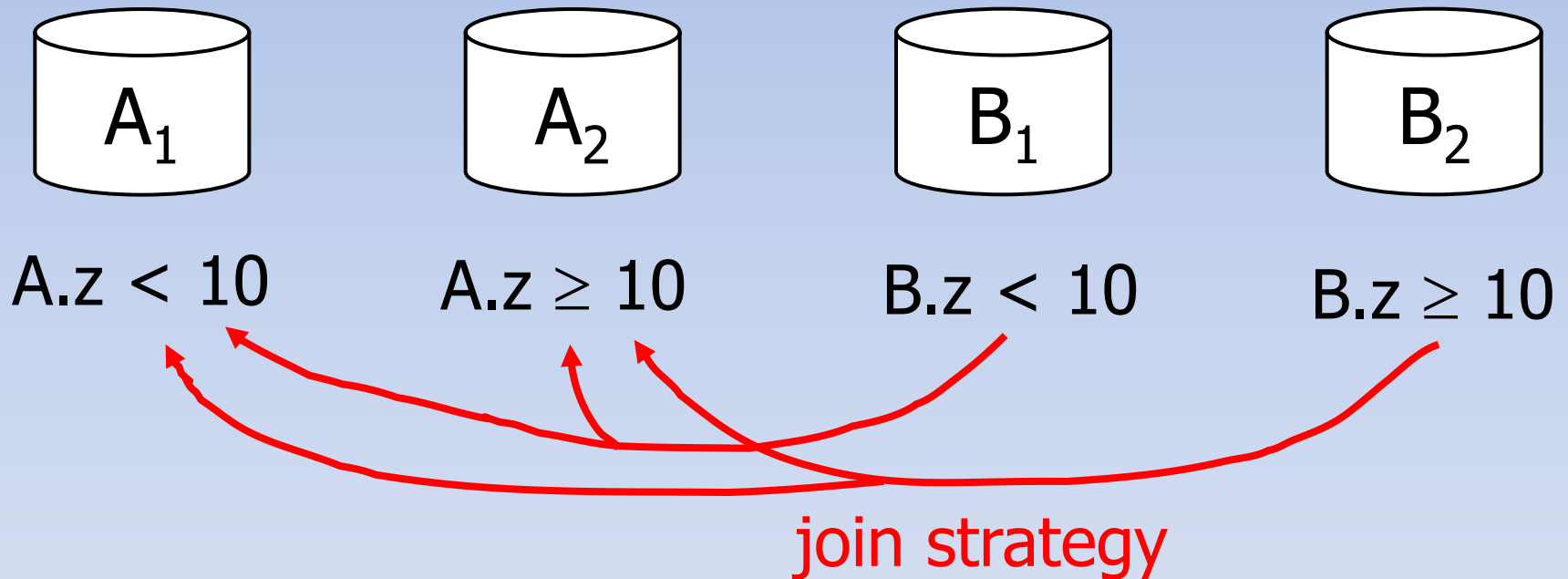
Join Processing

- Example: JOIN A, B over attribute X



Join Processing

- Example: JOIN A, B over attribute X



Grouping & Aggregation

- Similar to Joins:
 - Distribute tuples using hash function h that depends only on grouping attributes.
 - If each processor has all tuples for one bucket, it can process locally using standard uniprocessor algorithm!

Performance w/ Parallelism

Unary Ops

- Total work w/ Parallelism cannot be less than uniprocessor.
 - Elapsed time reduced w/ p processors
- Unary Operations like `SELECT` can be completed in $1/p$ th the time of uniprocessor

Performance w/ Parallelism

Joins

- Joins are expensive, but still benefit from parallelism!
- $R(X, Y)$ natural join $S(Y, Z)$
- We'll start by hashing all of the data from R and S using just shared attributes Y !
- Requires reading all of the tuples of R & S from disk:
 - Requires $B(R) + B(S)$ disk I/O's

Performance w/ Parallelism

Joins

- Read all tuples from R & S from disk.
 - Requires $B(R) + B(S)$ disk I/O's
- Ship tuples to hash bucket.
 - $(\frac{p-1}{p})(B(R) + B(S))$
 - $(\frac{1}{p})$ of data remains in place!
- Each processor may need to store, then re-read data to complete Join.

Performance w/ Parallelism

Joins

- Ship tuples to hash bucket.
 - $(\frac{p-1}{p})(B(R) + B(S))$
 - $(\frac{1}{p})$ of data remains in place!
 - Dominated by disk I/O!
- $3(B(R) + B(S))/p$
 - Initial read: $(B(R) + B(S))/p$
 - Assuming data is distributed over processors
 - Each processor stores and rereads: $2(B(R) + B(S))/p$
- $2(B(R) + B(S))/p$ to do the actual joins
- TOTAL: $5(B(R) + B(S))/p$
 - Uniprocessor requires: $3(B(R) + B(S))$

Performance w/ Parallelism

Joins

- $5(B(R) + B(S))/p$ can be further improved!
- In some cases the algorithm used at each processor can be improved since it is operating on less data than the full relation!
- Algorithm can be smart about insuring processor's ready to process data when received :
 - rather than writing data to disk then rereading it!

Exercise 20.1.1

- Suppose that a disk I/O takes 100 milliseconds. Let $B(R) = 100$, so the disk I/O's for computer $\sigma_c(R)$ on a uniprocessor machine will take 10 seconds. What is the speedup if this selection is executed on a parallel machine with p processors where:
 - a. $p=8$
 - b. $p=100$
 - c. $p=1000$

Exercise 20.1.1

- Suppose that a disk I/O takes 100 milliseconds. Let $B(R) = 100$, so the disk I/O's for computer $\sigma_c(R)$ on a uniprocessor machine will take 10 seconds. What is the speedup if this selection is executed on a parallel machine with p processors where:
- Assuming our data gets hashed evenly across processors!
 - a. $p=8 : 10\text{seconds}/8 = 1.25\text{seconds}$
 - b. $p=100 : 10/100 = 100\text{ms}$
 - c. $p=1000 : 10\text{ms}$

Exercise 20.1.2

w/ Example 20.2

- $R(X, Y) \bowtie S(Y, Z)$
 - $B(R) = 1000$
 - $B(S) = 500$
- 101 buffers at each of 10 Processors
 1. Hash tuples of R&S to 10 Processors w/ using only attributes Y.
 - $(1500)/10=150$ per processor
 - 15 stays, 135 goes, for total transport of 1350.
 2. 50 blocks from S stored in memory (101 buffers capacity) then as 100 blocks of R processed as arrives.
- Total cost is only 1500 disk I/O's

Exercise 20.1.2

w/ Example 20.2

- $R(X, Y) \bowtie S(Y, Z)$
 - $B(R) = 1000$
 - $B(S) = 500$
- 101 buffers at each of 10 Processors
 1. Hash tuples of R&S to 10 Processors w/ using only attributes Y.
 - $(1500)/10=150$ per processor
 - 15 stays, 135 goes, for total transport of 1350.
 2. 50 blocks from S stored in memory (101 buffers capacity) then as 100 blocks of R processed as arrives.
- Total cost is only 1500 disk I/O's
- Time required is 150 disk I/O's
 - 1500 Spread over the 10 processors
 - Plus time to ship tuples
- Improvements from Processors and Memory!
 - 10 Processors
 - 1010 buffers

Exercise 20.1.2

w/ Example 20.2

- $R(X, Y) \bowtie S(Y, Z)$
 - $B(R) = 1000$
 - $B(S) = 500$
- 101 buffers at each of 10 Processors
 1. Hash tuples of R&S to 10 Processors w/ using only attributes Y.
 - $(1500)/10=150$ per processor
 - 15 stays, 135 goes, for total transport of 1350.
 2. 50 blocks from S stored in memory (101 buffers capacity) then as 100 blocks of R processed as arrives.
- Total cost is only 1500 disk I/O's
- Time required is 150 disk I/O's
 - 1500 Spread over the 10 processors
 - Plus time to ship tuples
- Improvements from Processors and Memory!
 - 10 Processors
 - 1010 buffers

Exercise 20.1.2 w/ Example 20.2

- Given $B(R)$, $B(S)$, and M (number of blocks of memory for each processor). What are the conditions for the algorithm described!?

Exercise 20.1.2

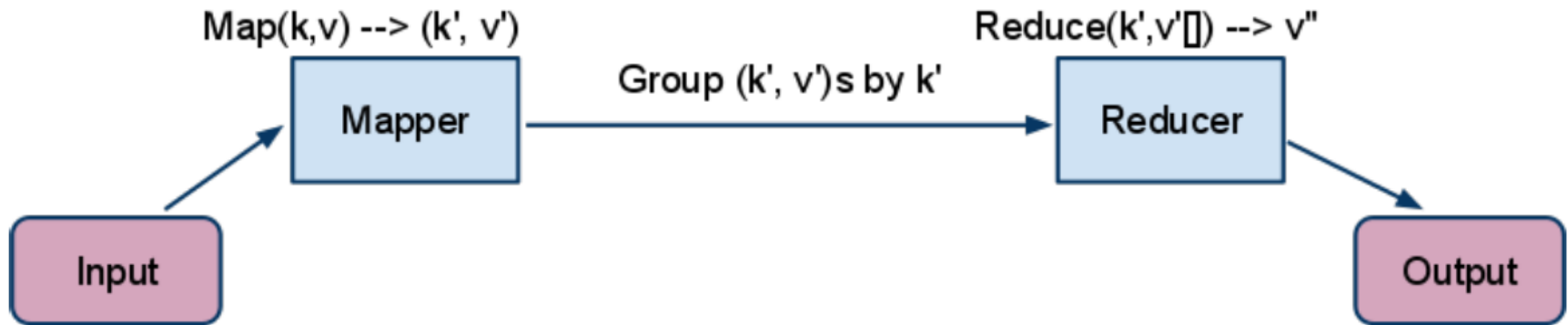
w/ Example 20.2

- Given $B(R)$, $B(S)$, and M (number of blocks of memory for each processor). What are the conditions for the algorithm described!?
- $(M-1) > B(S)/p$, given $B(S) < B(R)$
 - All of S must at each processor fit in memory
 - $B(S)/p$
 - 1 block of R must also be in memory to process Join.
 - $M > (B(S)/p) + 1 \Rightarrow M-1 > B(S)/p$

Parallel Processing w/ Map-Reduce

Map-Reduce Model

- Inspired by Map/Reduce in functional programming languages, such as LISP from 1960's



Map-Reduce Parallelism Framework

- High-Level programming system that works well with DB processes
- User writes two functions:
 - Map: process each item
 - Reduce: Combines results from Map function
- Master controller divides data into chunks and assigned processors.
- Map is run on processors w/ chunks
- Processor then assigned to run Reduce w/ Map Results.

Map-Reduce Word Count Example

- Input: Large number of text documents
- Task: Compute word count across all the document
- Solution
 - Mapper:
 - For every word in a document output (word, "1")
 - Reducer:
 - Sum all occurrences of words and output (word, total_count)

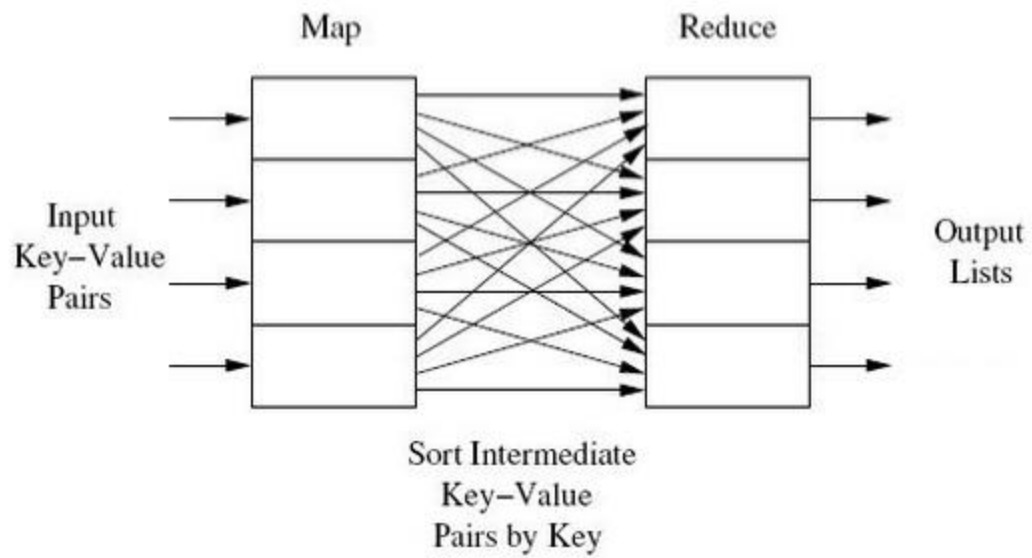


Figure 20.4: Execution of map and reduce functions

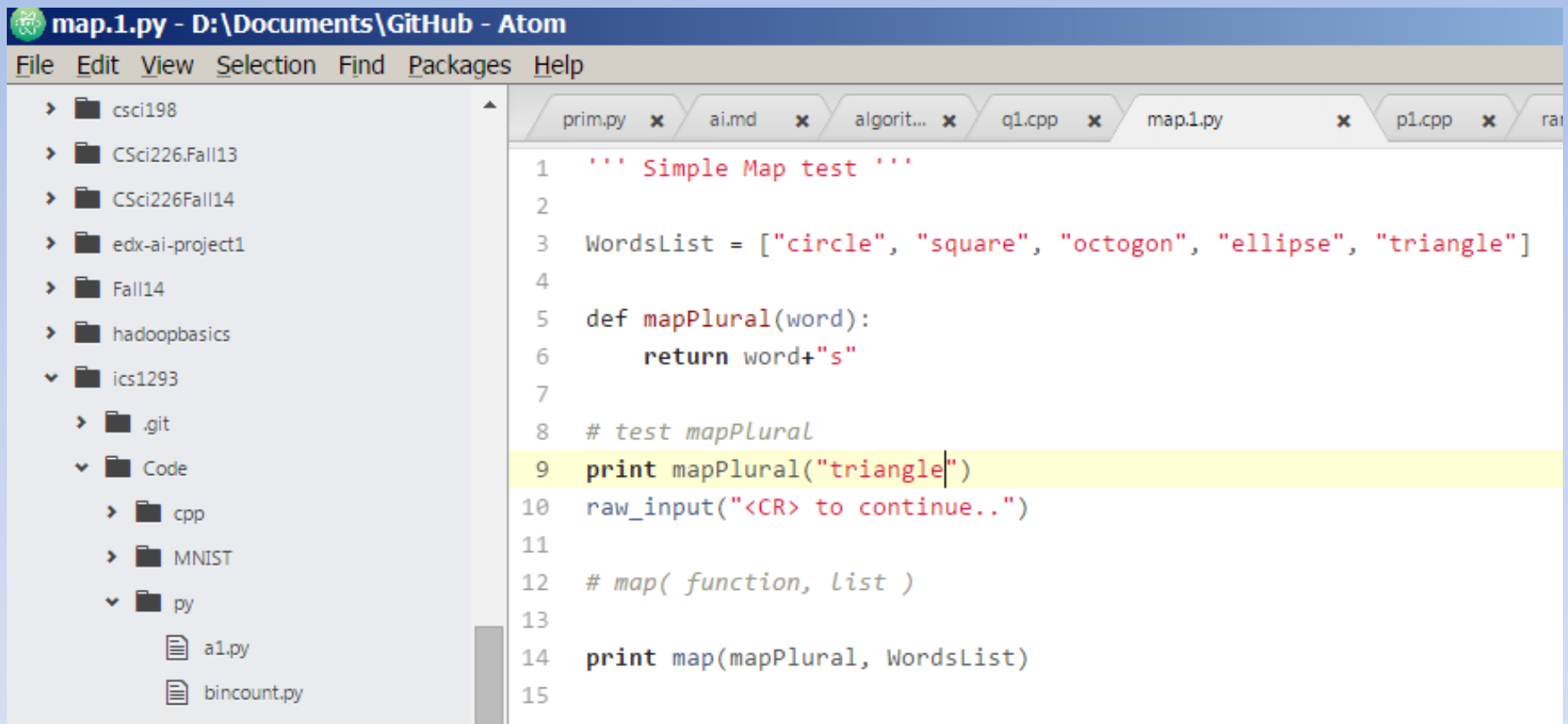
Map Function

- INPUT: set of key - value records
- Map is executed by one or more processes, located at any number of processors .
 - Each map process is given a chunk of the entire input data on which to work .
- The map function is designed to take one key - value pair as input and to produce a list of key - value pairs as output .
 - The types of keys and values for the output of the map function need not be the same as the types of input keys and values .
 - The keys that are output from the map function are not true keys in the database sense.
 - there can be many pairs with the same key value .

Map Function

- OUTPUT: from executing all the map processes is a collection of key - value pairs called the intermediate result.
- These key - value pairs are the outputs of the map function applied to every input pair . Each pair appears at the processor that generated it .
 - Remember that there may be many map processes executing the same function on different parts of input file at different processors.
- The key field of output pairs plays a special role in the REDUCE process.

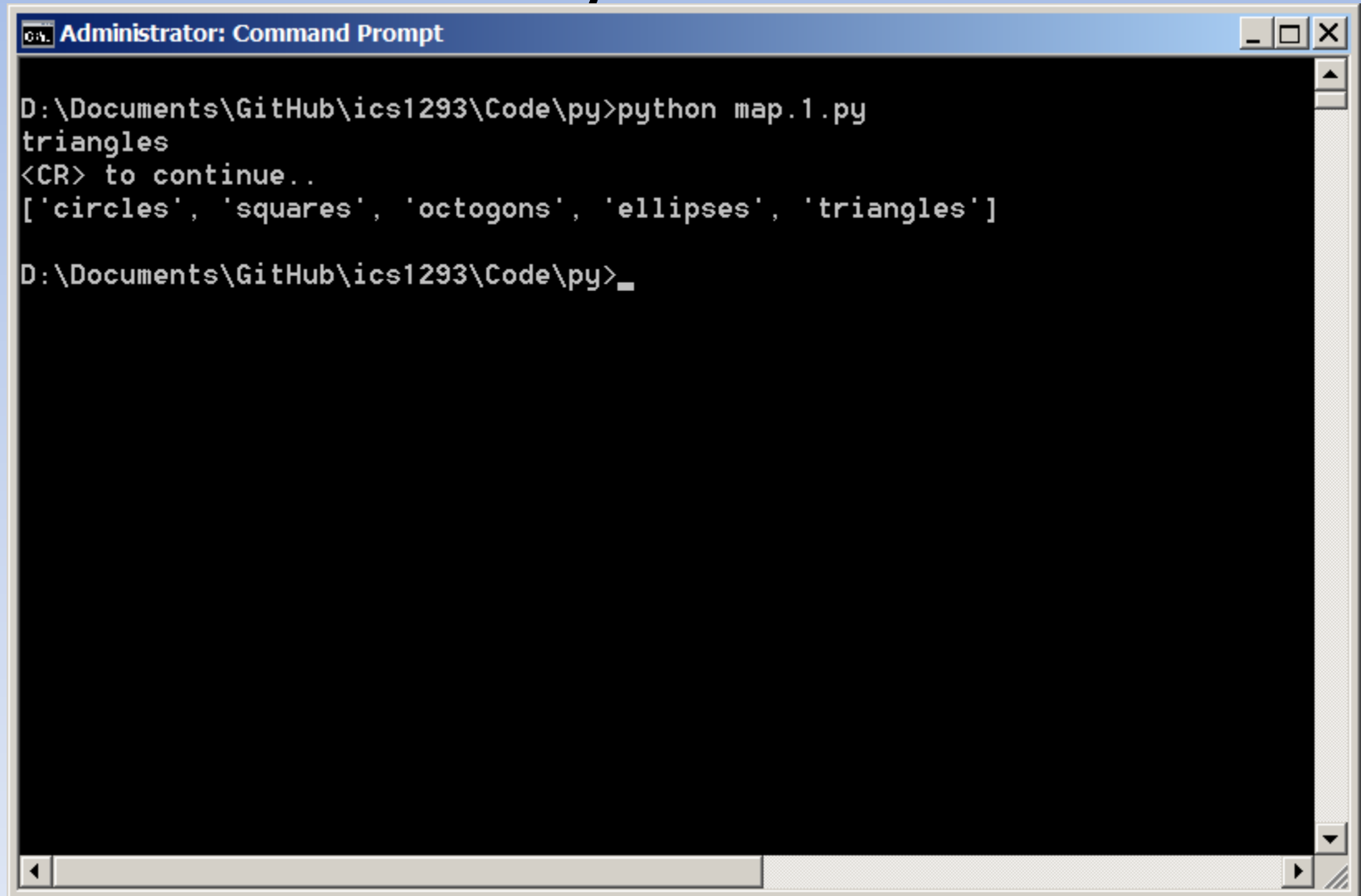
Map w/ Python



The screenshot shows the Atom text editor interface. The title bar indicates the file is `map.1.py` located at `D:\Documents\GitHub - Atom`. The menu bar includes `File`, `Edit`, `View`, `Selection`, `Find`, `Packages`, and `Help`. The file explorer on the left displays a directory tree with folders like `csci198`, `CSci226.Fall13`, `CSci226Fall14`, `edx-ai-project1`, `Fall14`, `hadoopbasics`, `ics1293`, `.git`, `Code`, `cpp`, `MNIST`, and `py`. The `py` folder is expanded, showing files `a1.py` and `bincount.py`. The main editor area shows the code for `map.1.py` with line numbers 1 through 15. The code defines a `mapPlural` function and uses `map` to apply it to a list of words. Line 9, `print mapPlural("triangle")`, is highlighted in yellow.

```
1  ''' Simple Map test '''
2
3  WordsList = ["circle", "square", "octagon", "ellipse", "triangle"]
4
5  def mapPlural(word):
6      return word+"s"
7
8  # test mapPlural
9  print mapPlural("triangle")
10 raw_input("<CR> to continue..")
11
12 # map( function, list )
13
14 print map(mapPlural, WordsList)
15
```


Map w/ Python



```
Administrator: Command Prompt
D:\Documents\GitHub\ics1293\Code\py>python map.1.py
triangles
<CR> to continue..
['circles', 'squares', 'octogons', 'ellipses', 'triangles']
D:\Documents\GitHub\ics1293\Code\py>_
```

Python w/ Lambda Function

```
>>>map(lambda x: x+"s", ["circle", "square", "triangle"])  
  
['circles', 'squares', 'triangles']
```

Python w/ Lambda Function

```
>>>map(lambda x: x+"s", ["circle", "square", "triangle"])\n['circles', 'squares', 'triangles']
```

[Download](#)[Libraries ▾](#)[Documentation ▾](#)[Examples](#)[Community ▾](#)[FAQ](#)

MLlib is Apache Spark's scalable machine learning library.

Easy to Deploy

Runs on existing Hadoop clusters and data.

If you have a Hadoop 2 cluster, you can run Spark and MLlib without any pre-installation. Otherwise, Spark is easy to run [standalone](#) or on [EC2](#) or [Mesos](#). You can read from [HDFS](#), [HBase](#), or any Hadoop data source.



Example 20.3

- In many case initial input might be a set of documents w/
 - keys being document ids
 - values being document
- Example Map Function:
 - INPUT: single pair (i, d) where i is the document id and d is document
 - Algorithm: Scan d and find all word w
 - OUTPUT: Output a list of pairs (i, w) where i is document id and w is a word in document.

Reduce Function

- The reduce function is also executed by one or more processes
- INPUT: Single key value together with a list of values that appear with it in the intermediate result.
 - Duplicates are not eliminated
- The reduce function is applied to pairs of the values from the list until a single result is produced.

Reduce w/ Python

- Python reduce is applied to a single list.
 - Difference with MapReduce is that there will be a separate list for each key value.
- Python reduce basically run on each list associated with each key.

Reduce w/ Python

WordCounts =

```
[ ("circle", [0, 1, 1, 3, 5]), ("ellipse", [0, 0, 0, 2, 1])]
```

```
reduce( lambda x, y: max(x,y), List )
```

- What is produced by reduce function when applied to “circle” list, and “ellipse” list?
- 5 for Circle
- 2 for Ellipse

Reduce w/ Python

WordCounts =

[(“circle”, [0, 1, 1, 3, 5]), (“ellipse”, [0, 0, 0, 2, 1])]

reduce(lambda x, y: x+y, List)

- What is produced by reduce function when applied to “circle” list, and “ellipse” list?
- Circle=10
- Ellipse=3

Reduce Function w/ Spark

[Overview](#)[Programming Guides ▾](#)[API Docs ▾](#)[Deploying ▾](#)[More ▾](#)`distinct([numTasks])`

Return a new dataset that contains the distinct elements of the source dataset.

`groupByKey([numTasks])`

When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs.

Note: If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using `reduceByKey` or `aggregateByKey` will yield much better performance.

Note: By default, the level of parallelism in the output depends on the number of partitions of the parent RDD. You can pass an optional `numTasks` argument to set a different number of tasks.

`reduceByKey(func, [numTasks])`

When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function *func*, which must be of type (V,V) => V. Like in `groupByKey`, the number of reduce tasks is configurable through an optional second argument.

`aggregateByKey(zeroValue)(seqOp, combOp,
[numTasks])`

When called on a dataset of (K, V) pairs, returns a dataset of (K, U) pairs where the values for each key are aggregated using the given combine functions and a neutral "zero" value. Allows an aggregated value type that is different than the input value type, while avoiding unnecessary allocations. Like in `groupByKey`, the number of reduce tasks is configurable through an optional second argument.

`sortByKey([ascending], [numTasks])`

When called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the boolean `ascending` argument.

Reduce by Key (kinda) w/ Python

```
[("circle", [0, 1, 1, 3, 5]), ("ellipse", [0, 0, 0, 2, 1])]
```

```
map(  
    lambda L:
```

```
        ( L[0], reduce(lambda x,y: x+y, L[1]) ),
```

```
        WordCounts)
```

- What do you get?

Reduce by Key (kinda) w/ Python

```
[("circle", [0, 1, 1, 3, 5]), ("ellipse", [0, 0, 0, 2, 1])]
map(
    lambda L:
        ( L[0], reduce(lambda x,y: x+y, L[1]) ),
    WordCounts)
```

- What do you get?
 - [('circle', 10), ('ellipse', 3)]



Scalable Machine Learning

Learn the underlying principles required to develop scalable machine learning pipelines and gain hands-on experience using Apache Spark.

Berkeley
UNIVERSITY OF CALIFORNIA

Peer-To-Peer Distributed Search

- With Peer-To-Peer networks, data is distributed over many nodes
- BUT NO : Centralized Index!!
- Searching Surprisingly Hard!
- Distributed Hashing allows networks to grow and shrink w/ efficient searching!

Peer-To-Peer Networks

- A peer-to-peer network is a collection of nodes/peers (collaborating machines) that:
 - Are Autonomous:
 - Participants do not respect any central control and leave and join at will!
 - Are Loosely Coupled:
 - Using Internet Communication
 - Not Hard Wired
 - Are EQUAL in Functionality
 - Share Resources

Central Control Index Issues

- Early systems maintained centralized control index.
- Central control index could identify peer/node containing items!
- Later systems eliminate this Central Control Index!!
- Locating elements functionality is distributed (like data) throughout network
 - ELIMINATING need for Central Control Index!!
 - NO CHOICE when network is extremely large!

Distributed-Hashing Problem

- Our problem is lookup records
- Records consist for key value pairs:
 - key K
 - value V
- When key-value table is small, no problems.
- Interesting case w/ key-value table is too large for single node.

Distributed-Hashing Problem

- Abstract the problem as follows:
 1. At any time, only one node among peers knows the value for a given key K.
 2. Key-Value pairs are distributed equally among peers.
 3. Any nodes can ask a peer for a value V associated with key K.
 - V should be obtained using number of messages much smaller than number of nodes (as number of nodes grows large).
 4. Routing information must also grow more slowly than the number of nodes!

Centralized Solutions

- When participants are fixed (or change slowly) then problem has simple solutions.
 - For example: hash function used to hash keys to nodes.
 - Value is placed at node `hashFunction(key)`
- Google approach for indexing entire web.
 - Acts like central control
 - Actually distributed over many nodes

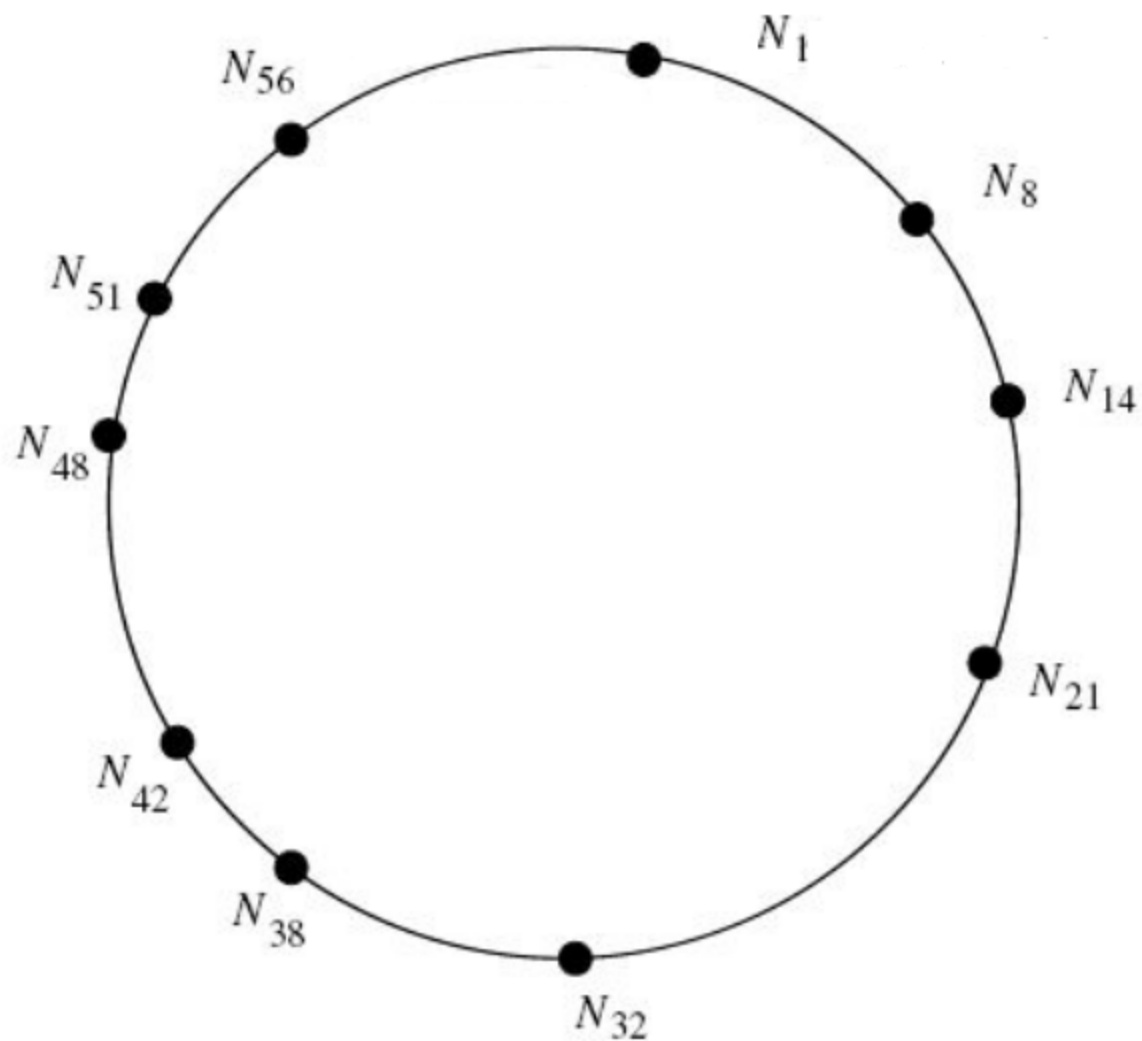


Figure 20.14: A chord circle

Chord Circle

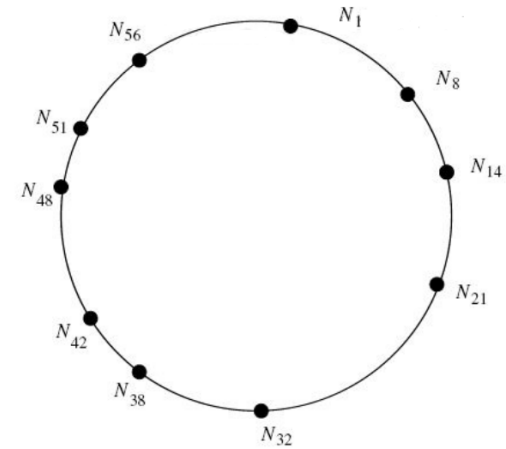


Figure 20.14: A chord circle

- Peers are arranged around a circle!
- Each nodes knows it predecessor and successor.
- Nodes have list of links of nodes located at exponentially growing set of distances:
 - Links are chords
 - Finger Table stores chords

Adding a Node to the Circle

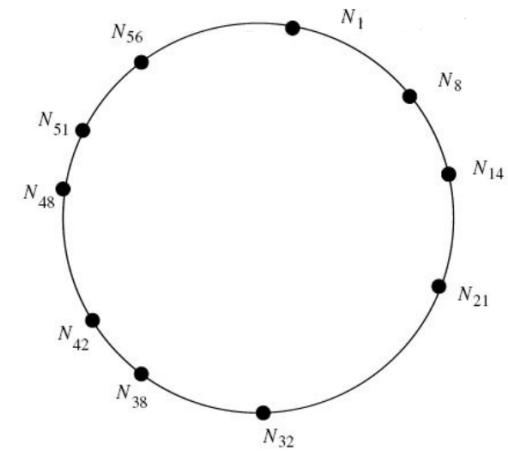


Figure 20.14: A chord circle

- Hash its ID i as $h(i)$
- Place new node with ID i at position $h(i)$
 - $N_{h(i)}$
- Hash function maps IP addresses and keys to m -bit numbers

Adding a Key-Value to the Circle

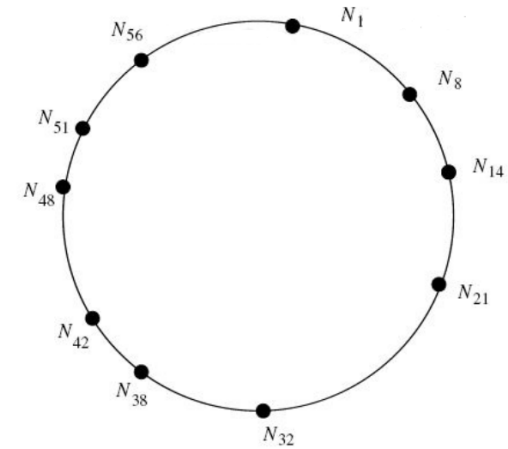


Figure 20.14: A chord circle

- Hash its key K as $h(K)$
- Place Key-Value pair (K, V) at node with N_j where $h(K) \leq j$
 - If $h(K)$ is larger than largest Node id, it is placed at first node
 - The node clockwise of largest Node.

Sending Messages

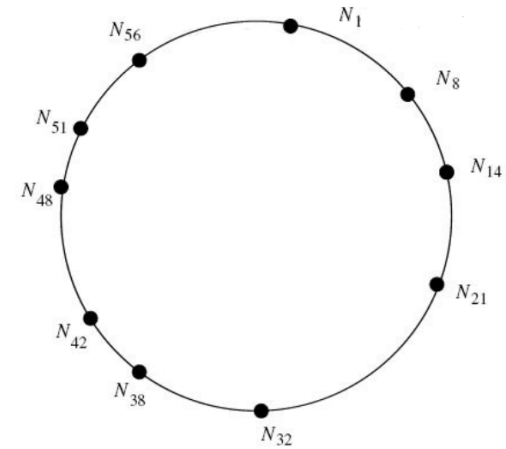


Figure 20.14: A chord circle

- Each node links to its predecessor and successor.
 - These links can be used to find any node (to send messages).
- Can we do better??

Finger Table

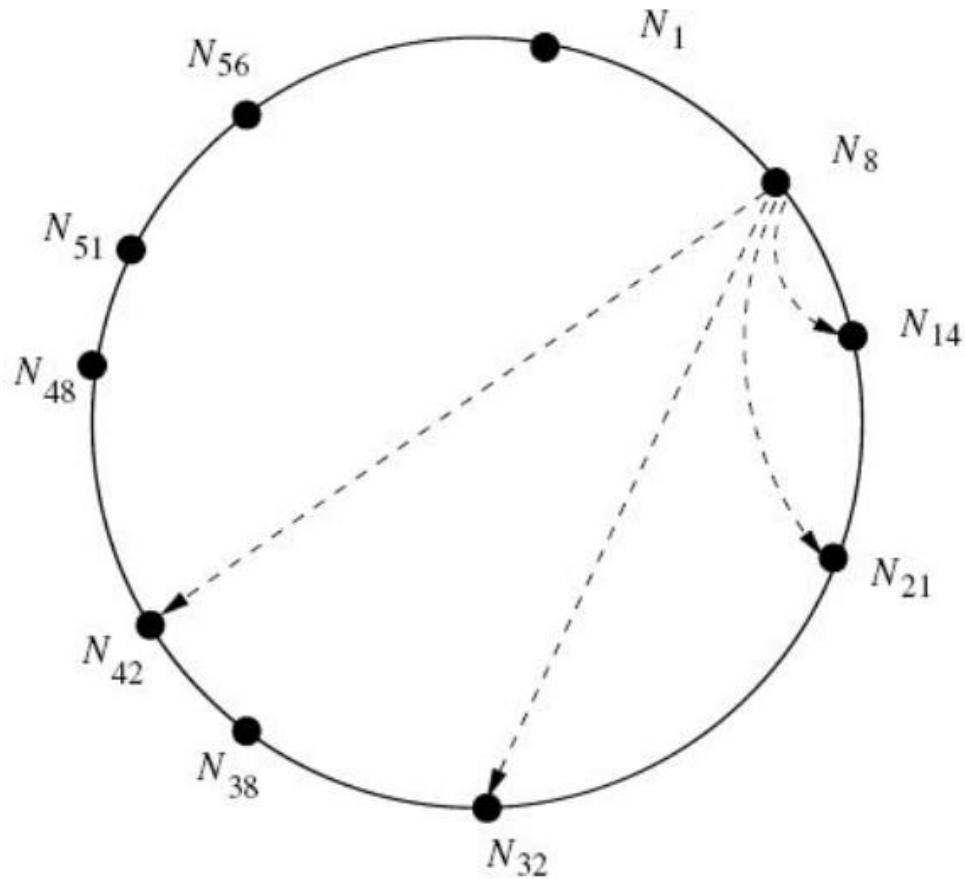
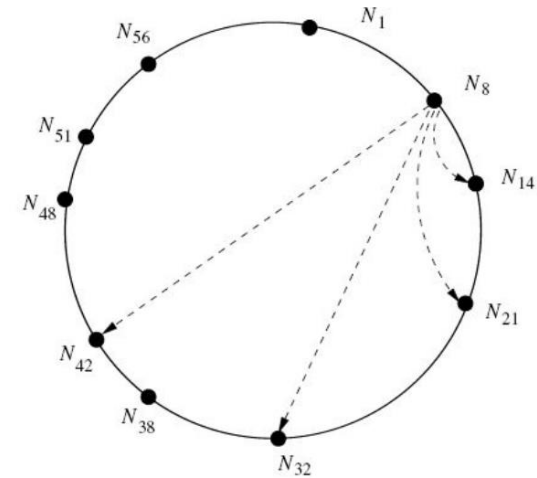


Figure 20.16: Links in the finger table for N_8

Finger Table

Figure 20.16: Links in the finger table for N_8

- Finger table stores first node found at distances around the circle that are a power of 2.

Distance	1	2	4	8	16	32
Node	N_{14}	N_{14}	N_{14}	N_{21}	N_{32}	N_{42}

Figure 20.15: Finger table for N_8

Example 20.20:

N_8 Finger Table

Distance	1	2	4	8	16	32
Node	N_{14}	N_{14}	N_{14}	N_{21}	N_{32}	N_{42}

Figure 20.15: Finger table for N_8

- Distance 1:
 - $8 + 1 = 9$
 - N_{14} first node (smallest node) greater than 9.
- Distance 2:
 - $8 + 2 = 10$
 - N_{14} first node (smallest node) greater than 10.
- Distance 4:
 - $8 + 4 = 12$
 - N_{14} first node (smallest node) greater than 12.
- Distance 8:
 - $8 + 8 = 16$
 - N_{21} first node (smallest node) greater than 16.
- Distance 16:
 - $8 + 16 = 24$
 - N_{32} first node (smallest node) greater than 24.
- Distance 32:
 - $8 + 32 = 40$
 - N_{42} first node (smallest node) greater than 40.

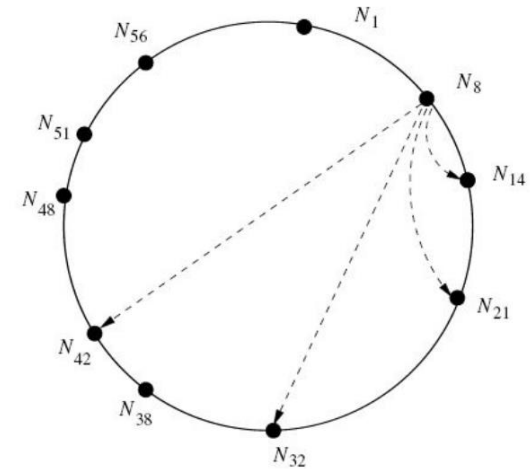


Figure 20.16: Links in the finger table for N_8

Search w/ Algorithm 20.21

- Key Algorithm for Searching
- N_i initiates request for the value associated with key K .
- $h(K) = j$
- OUTPUT:
 - A sequence of messages
 - Messages are sent between nodes
 - Messages result V (the value paired with key K) being send to N_i
 - OR: Statement that pair does not exist!

Search w/ Algorithm 20.21

Step 1

- Search ends if the current node N_c has successor node N_s and:
$$c < j \leq x$$
- N_c sends a message to N_s asking for (K, V)
- N_c tells N_s that N_i is the original requestor
- N_s either sends the value V , or notifies j that k does not exist.

Search w/ Algorithm 20.21

Step 2

- The current node N_c consults its finger table.
- N_c finds the highest-numbered node N_h that is less than j !
- N_c sends a message to N_h asking for (K, V) on behalf of N_i .
 - N_h then becomes the current node N_c

Example 20.22: Searching using Finger Table

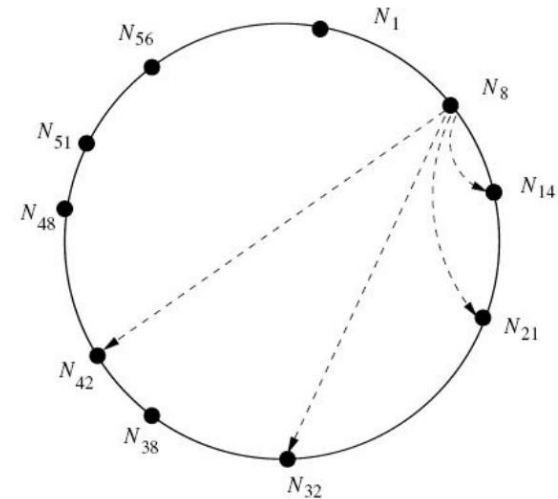


Figure 20.16: Links in the finger table for N_8

- Assume N_8 wants to find the value V for key K .
 - $h(K) = 54$
- N_8 successor is N_{14}
 - 54 is not between 8 and 14.
- N_8 searches finger table for largest node less than 54.
 - Finds N_{42}

Example 20.22: Searching using Finger Table

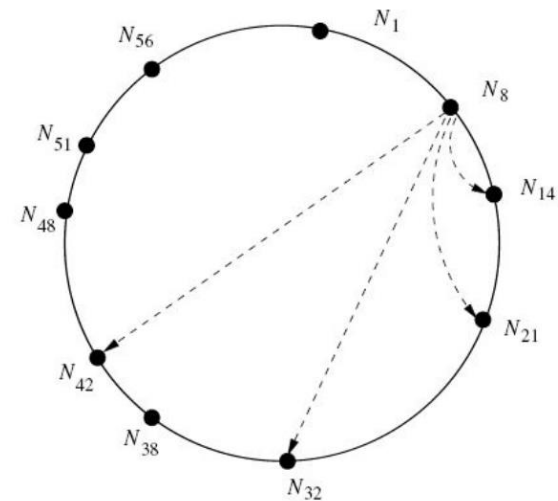


Figure 20.16: Links in the finger table for N_8

- N_8 asks N_{42} to return value for key 54!
- N_{42} has successor N_{48}
 - 54 is not between 42 and 48
- N_{42} examines its finger table:

Example 20.22: Searching using Finger Table

- N_{42} examines its finger table:
- The last node less than 54 is N_{51}
 - In circular sense
- N_{51} successor is N_{56}
 - 54 is between 51 and 56
 - K 's value if it exists must be at N_{56}
- N_{51} sends request to N_{56} , who then responds to N_8

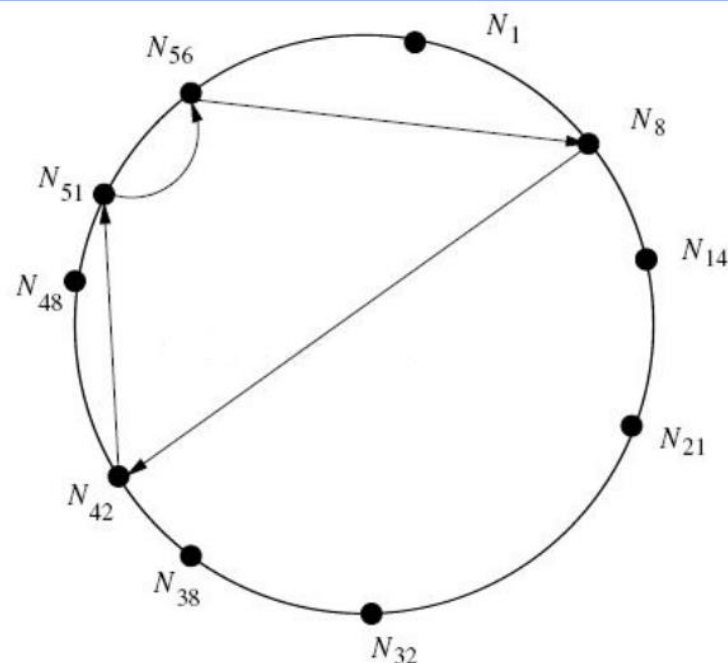


Figure 20.17: Message sequence in the search for (K, V)

Entering the Circle

- A new node N_i must know at least one member to enter the circle.
- The node asked would search for the index value for $h(N_i)$.
 - This node is the successor for N_i .
- For N_i to become part of the circle:
 - Change pred and succ links, so N_i is properly linked into circle.
 - Rearrange data so N_i gets all the data at N_j that belongs to N_i .
 - key-value pairs N_i has where the key hashes to less than i .
- For concurrency issues, we proceed in two steps.
 - FIRST: set succ of N_i to N_j and pred of N_i to nil.

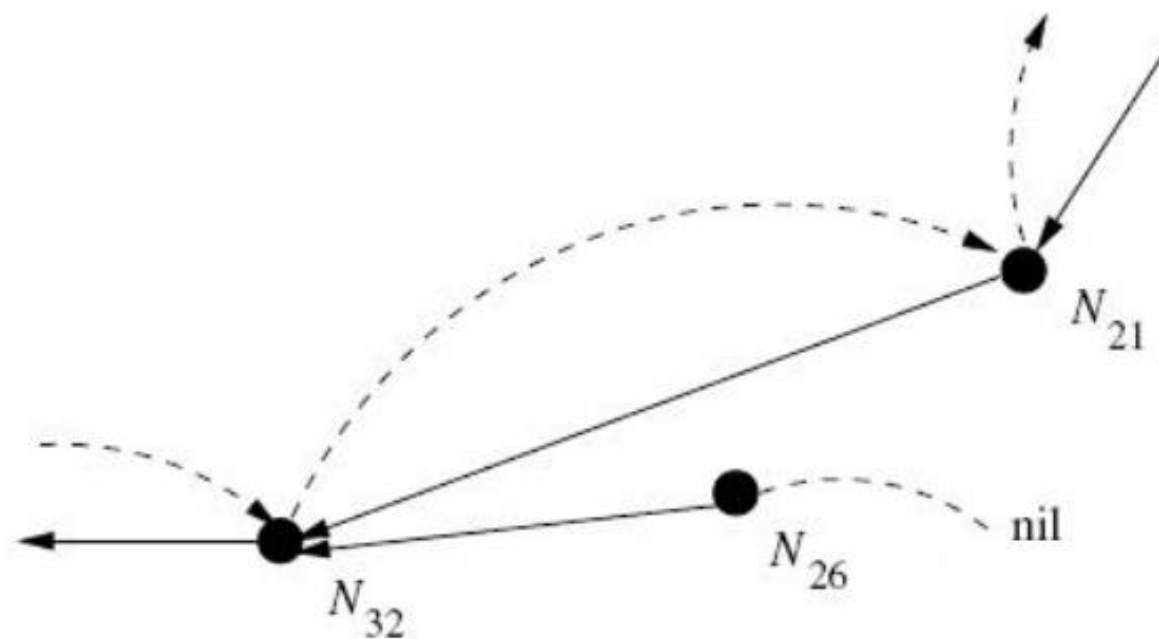


Figure 20.18: Adding node N_{26} to the network of peers

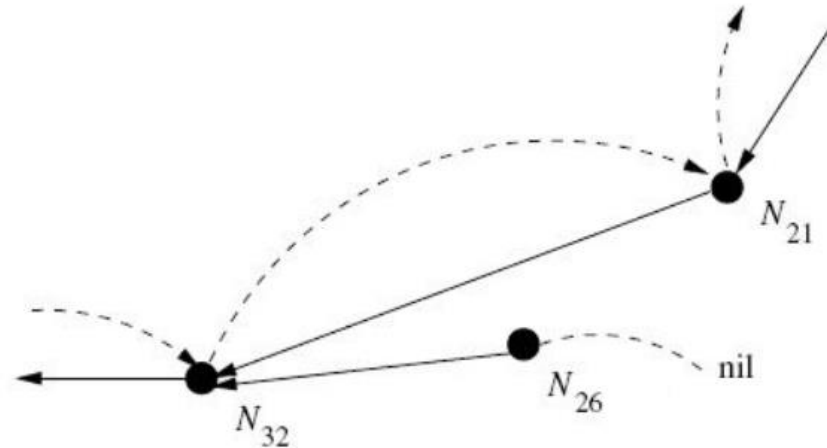


Figure 20.18: Adding node N_{26} to the network of peers

- N_{32} and N_{21} are not yet aware of the new node N_{26}
- Step two is a stabilization Step done automatically periodically by all nodes.
- During the stabilization step :
 - predecessors and successors are updated.
 - data is shared as needed with

Stabilization Process: Performed at node N_{26}

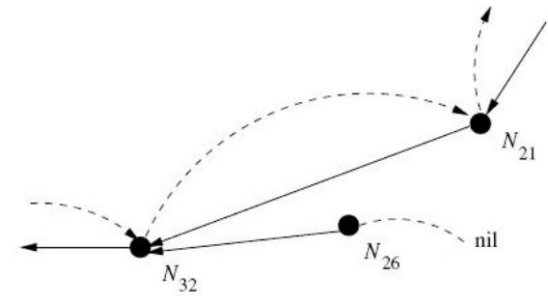


Figure 20.18: Adding node N_{26} to the network of peers

1. N sends a message to its successor S asking for S 's predecessor.
 - If S 's predecessor equals N , skip to step 4.
2. If P lies strictly between N and S :
 - N records P as its successor.
 - NOT True in example above
3. S' is the current successor of N (either S or P)
 - If the pred of S' is nil OR N is between S' and its Pred.
 - N tells S' to set N as its pred.
4. S' shares its data with N .
 - All (K,V) pairs at S' such that $h(K) \leq N$ are moved to N .

Stabilization Process: Performed at node N_{26}

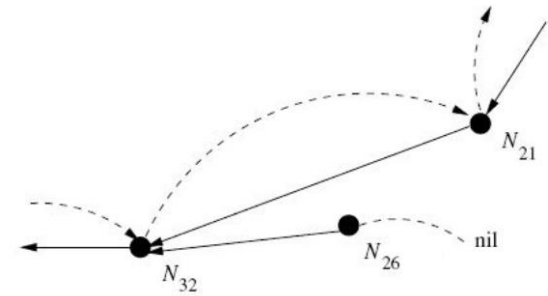


Figure 20.18: Adding node N_{26} to the network of peers

- Stabilization Step:
 - $N = N_{26}$
 - $S = N_{32}$
 - $P = N_{21}$
- P does not lie between N and S , so step 2 makes no changes.
 - $S' = S = N_{32}$
- N does lie between S' and its predecessor N_{21}
 - N_{26} is made predecessor of N_{32}

Stabilization Process: Performed at node N_{26}

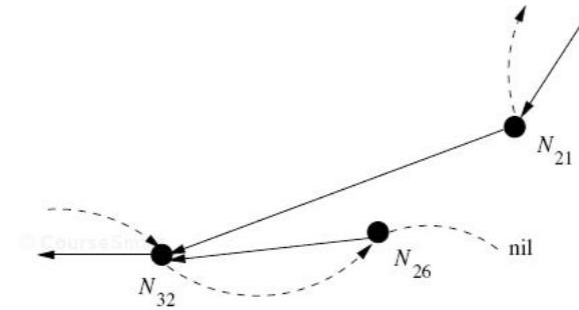


Figure 20.19: After making N_{26} the predecessor of N_{32}

- Stabilization Step:
 - $N = N_{26}$
 - $S = N_{32}$
 - $P = N_{21}$
- P does not lie between N and S , so step 2 makes no changes.
 - $S' = S = N_{32}$
- N does lie between S' and its predecessor N_{21}
 - N_{26} is made predecessor of N_{32}

Stabilization Process: Performed at node N_{21}

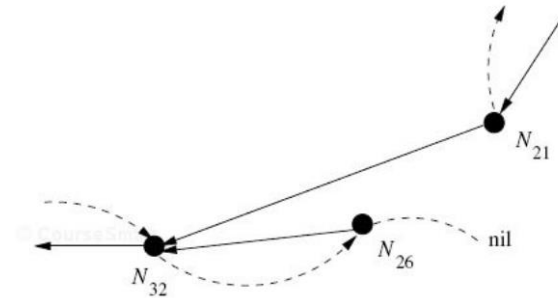


Figure 20.19: After making N_{26} the predecessor of N_{32}

- Stabilization Step:
 - $N = N_{21}$
 - $S = N_{32}$
 - $P = N_{26}$
- STEP 2: P does lie between N and S :
 - N_{26} becomes successor of N_{21}
- STEP 3: $S' = N_{26}$: since pred is nil, it is set to $N = N_{21}$
 - N_{26} is made predecessor of N_{32}

Stabilization

Process:

Performed at
node N_{21}

- Stabilization Step:
 - $N = N_{21}$
 - $S = N_{32}$
 - $P = N_{26}$
- STEP 2: P does lie between N and S:
 - N_{26} becomes successor of N_{21}
- STEP 3: $S' = N_{26}$: since pred is nil, it is set to $N = N_{21}$
 - N_{26} is made predecessor of N_{32}

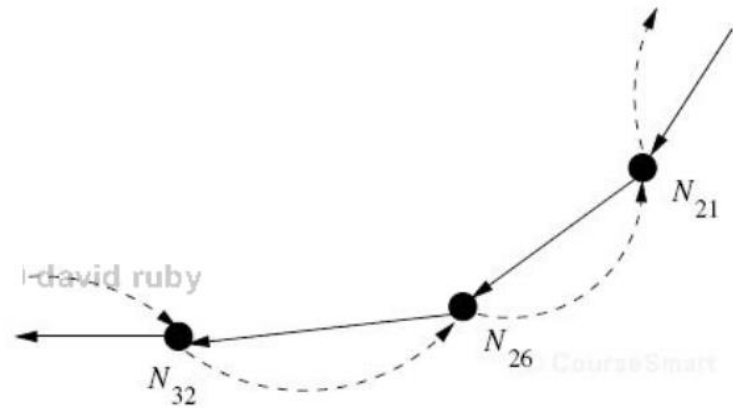


Figure 20.20: After N_{21} runs the stabilization algorithm

Periodic Finger Table Updates

- After the previous adjustments further work remains.
- Finger tables may still need updates.
 - Some mistakenly link to N_{32} instead of the new N_{26}

Periodic Finger Table Updates

- In order to update (or create) the finger table, we use our Lookup Algorithm (Algorithm 20.21).
- For each $i=1, 2, 3, 8, \dots$ a node N must lookup node $(N + i) \bmod 2^m$
 - hash function h produces m -bit numbers
- When it gets back the node where the network thinks such a key would be located:
 - N sets its finger-table entry for distance i to that node value!

Initial Finger Table

- The lookup algorithm can also be used to build initial finger table.
- First Finger Table entry is successor node:
 - at distance 1.
- Finding Node for the entry at distance 2 can be found by calling upon the node successor node now at position 1 of finger table.
- In general the Node at position $2*i$ is either:
 - The same as at position i .
 - Found by calling upon node at position i !

Leaving the Circle

- In peer-to-peer networks, nodes can leave the circle at any time.
- There are two cases to consider.
 - Case 1: The node leaves the circle “gracefully”!
 - Case 2: A node “fails” !

Leaving the Circle -- Gracefully

- Two leave the circle “Gracefully” a node must:
 1. Notify its predecessor and successor that it is leaving.
 - They can become each other’s predecessor and successor.
 2. Transfer all its data to its successor!!
- Network will still have errors, but these will be corrected :
 - When Nodes encounters a broken link in finger table.
 - Periodic Updates.

When Nodes Fail!

- In the worst case, when a node fails, its data could be lost!
- The only way to avoid lost data is through data replication!

Managing Failures w/ Replication

- In order to avoid data loss in the case of Node going down...!
- Replication is Required!!!
- One Approach:
- Each nodes replicates its data at its successor and predecessor.
- Each nodes records its predecessor & successor, along with its predecessor's predecessor & successor's successor.

Managing Replication w/ Clustering

- Another approach to node failure is :
 - clustering nodes
 - replicating data within clusters
- Clusters can split and merge in order to manager their size!

Data Technologies

- Semi-structure Data w/ XML
 - Xpath/Xquery
 - XSLT
- RDBMS
 - MySQL
- Distributed Data
 - MongoDB
 - Map-Reduce, Hadoop, Spark
- Peer-To-Peer Protocols

Peer-To-Peer Protocols

- What type of project might we want with Peer-To-Peer Protocols!



Chat



Library



Search



Hot List

Search Fields:

Artist: public enemy

Song Title:

Max Results: 100

Clear Fields

Find It

Advanced Fields (Optional):

Bitrate must be:

Frequency must be:

Ping Time must be:

Line Speed must be:



Filename	Filesize	Bitrate	Freq	Length	User	Line Speed	Ping	
Hard Rock\StainD with Fred Durst and DJ ...	3,665,975	128	44100	3:49	Emub7	DSL	N/A	
Public enemy & Anthrax-Bring the noise.mp3	3,343,410	128	44100	3:30	Tish69	Unknown	N/A	
Puff Daddy - Public Enemy Num one.mp3	3,925,055	128	44100	4:05	CSUF-A...	T1	N/A	
Music\Public Enemy (He Got Game).mp3	3,787,920	128	44100	3:57	Joe-dog...	Cable	N/A	
Music\Public Enemy - Don't Believe the Hy...	6,378,628	160	44100	5:17	gonztshi...	56K Modem	N/A	
Music\Public Enemy - Do You Wanna Go ...	3,766,271	128	44100	3:55	matbar	Cable	N/A	
Music\Public Enemy - Fight the power.mp3	3,826,536	128	44100	3:59	gonztshi...	56K Modem	N/A	
Old School\Anthrax_n_Public_Enemy-Bring...	3,334,814	128	44100	3:29	Shock...	T1	N/A	
another cd\StainD with Fred Durst and DJ ...	3,657,728	128	44100	3:49	Wingma...	Cable	N/A	
Public Enemy - He Got Game.mp3	4,564,610	128	44100	4:44	bmthis27	T1	N/A	

Returned 100 results.

Get Selected Song(s)

Add Selected User to Hot List

Online (User): Sharing 0 Songs.

Currently 709,082 songs (2,845 gigabytes) available in 4,708 libraries.



BitTorrent, Inc.

Internet

51-200 employees

Home

Careers

BitTorrent, Inc. is one of the world's leading peer-based technology companies. We maintain a globally recognized ecosystem of technology protocols, consumer software, and consumer electronics devices that help people find, share and move digital media. We are the creators of the BitTorrent protocol and proponents of an open Internet. Our technologies are used by hundreds of millions of people around the world and currently drive between 20% and 40% of global Internet traffic.

Want to work with us? Visit our careers page for current openings and help us shape the future of the web.

<http://www.bittorrent.com/company/about/jobs>

Specialties

BitTorrent, Social Sharing, Consumer Software, uTorrent, Apps, BTML, Live Streaming, p2p, peer to peer, networking

Website

<http://www.bittorrent.com>

Industry

Internet

Type

Privately Held

Headquarters

303 2nd Street Suite S600 San
Francisco, CA 94107 United States

Company Size

51-200 employees

Founded

2004

BitTorrent 7.2.1

File Options Help

Find Content

Torrents (1)

- Downloading (1)
- Completed (0)
- Active (1)
- Inactive (0)
- Labels (1)

Feeds (0)

Apps (32 new)

Name	#	Size	Done	Status	S	Seeds	Peers	Down Speed	Up Speed	ETA
Super OS 11.04 64 bits Ubuntu-...	1	1.19 GB	0.0%	Downloading		4 (23)	1 (10)	2.9 kB/s		3w 1d

General Trackers Peers Pieces Files Speed Logger

Downloaded: 0.0 %

Availability: 4.031

Transfer

Time Elapsed:	10s	Remaining:	3w 1d	Wasted:	0 B (0 hashfails)
Downloaded:	16.0 kB	Uploaded:	0 B	Seeds:	4 of 5 connected (23 in swarm)
Download Speed:	2.9 kB/s (avg. 1.5 kB/s)	Upload Speed:	0.0 kB/s (avg. 0 B/s)	Peers:	1 of 28 connected (4 in swarm)
Down Limit:	∞	Up Limit:	∞	Share Ratio:	0.000
Status:	Downloading				

General

Save As: C:\Users\Softonic ES\Downloads\Super OS 11.04 64 bits Ubuntu-Linux based

Total Size: 1.19 GB (16.0 kB done) Pieces: 1220 x 1.00 MB (have 0)

Created On: 31/05/2011 0:10:47

Hash: 95A2B4E4 517A6B55 177FE6E2 71985243 70F27522

Comment: Auto-generated torrent by Mininova.org CD

DHT: 258 nodes (Updating) D: 3.2 kB/s T: 414.7 kB U: 0.9 kB/s T: 75.8 kB



Anonymity Online

Protect your privacy. Defend yourself against network surveillance and traffic analysis.



[Download Tor](#) 

- ➔ Tor prevents people from learning your location or browsing habits.
- ➔ Tor is for web browsers, instant messaging clients, and more.
- ➔ Tor is free and open source for Windows, Mac, Linux/Unix, and Android

What is Tor?

Tor is free software and an open network that helps you defend against traffic analysis, a form of network surveillance that threatens personal freedom and privacy, confidential business activities and relationships, and state security.

[Learn more about Tor »](#)

Why Anonymity Matters

Tor protects you by bouncing your communications around a distributed network of relays run by volunteers all around the world: it prevents somebody watching your Internet connection from learning what sites you visit, and it prevents the sites you visit from learning your physical location.

[Get involved with Tor »](#)

What else might we build?

- Distributed Anonymous Database System
- Private Sharing Network
 - People don't like that your central server is recording too much!
- Incorporate Public-Key Cryptosystem

Peer-To-Peer Technologies

- Clustering Nodes into Trust Groups
- Tracking membership activity.
 - Taking down nodes allowing easy access.