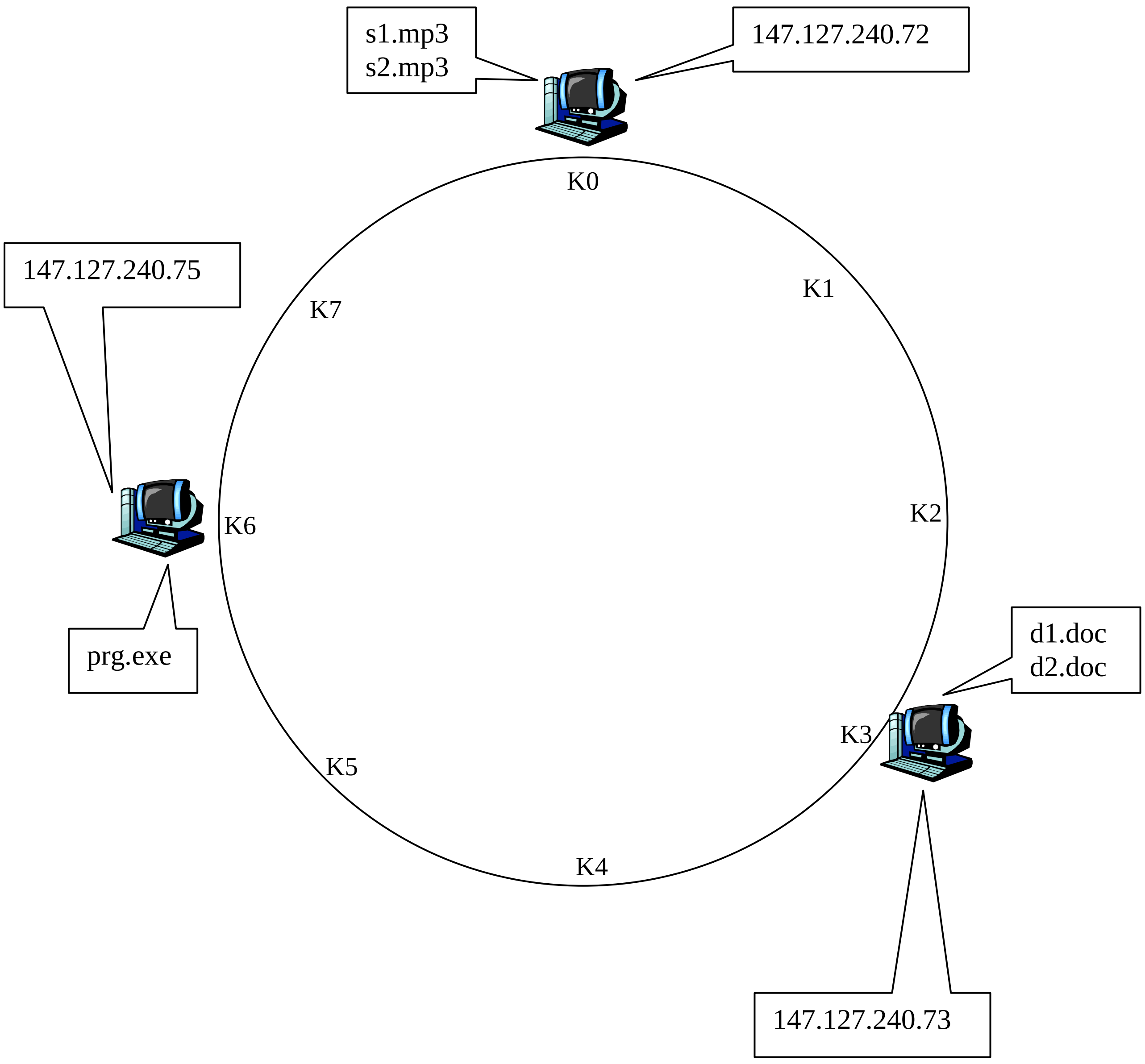
# Chord Protocol

Chord is a lookup protocol for searching data by using consistent hashing. The main concept of the chord protocol is the fragment of file name which indexing into sorted order and distributed its fragment to other nodes. Since the indexing (keyword) is already sorted and there is the finger table which created by using $2^m$ degree, then the lookup process can go like dividing into half for each time.

Imagine of the small network, we have only 3 computers in the networks and each computer share files following the picture below:



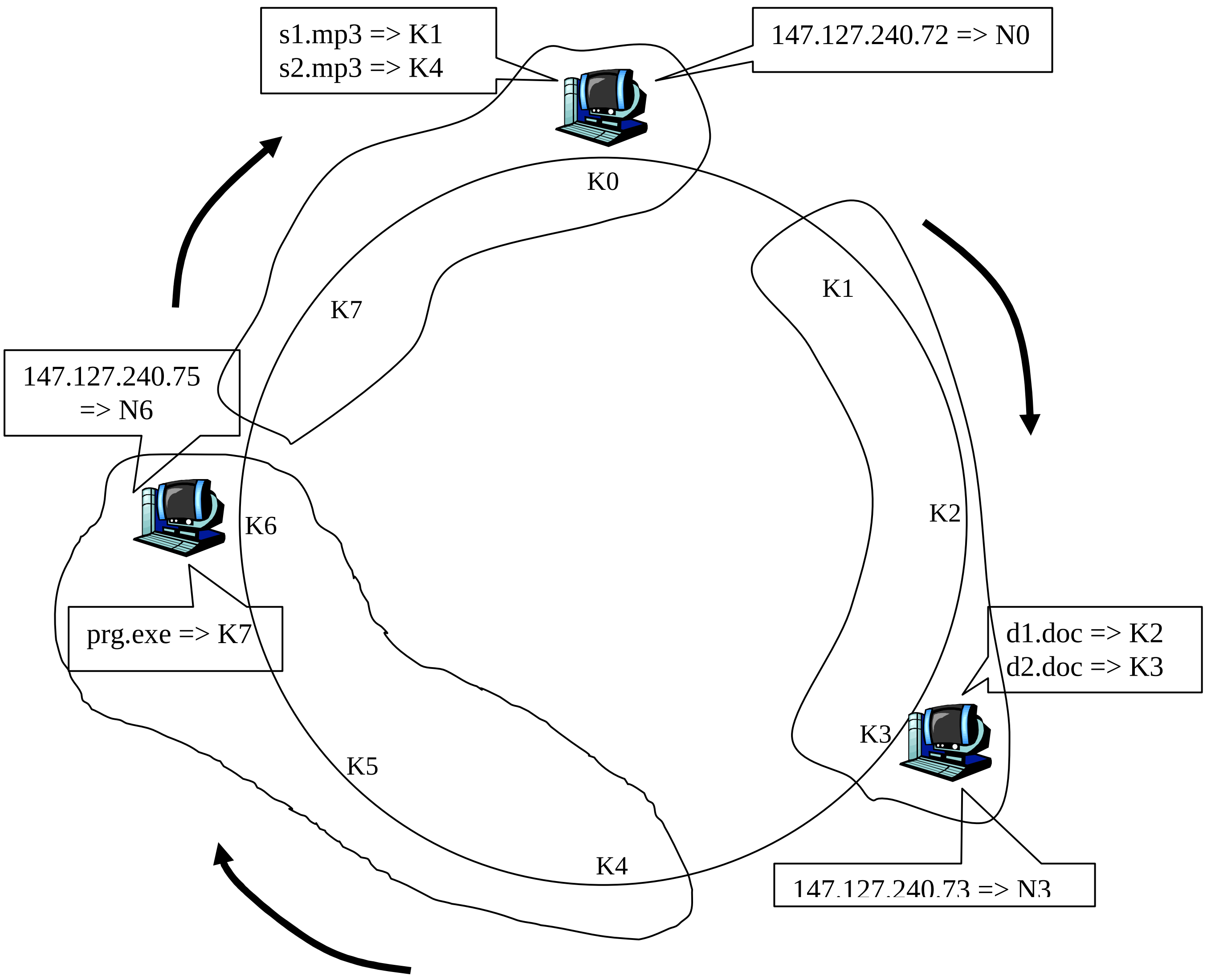
Use SHA-1 as the hashing function.

In practical, we use hashing function with 160-bit, which mean we will have $2^{160}$ maximum number of sharing files (and number of computers) in the network. That's large enough but for easy to understand, we will use 3 bit hashing function since the maximum number of sharing files is 8 (because of $2^3$)

By: Warodom Werapun

| **<u>Hashing function</u>** | **<u>Assume Hashing</u>** |
|---|---|
| **SHA-1 (3 bit hashing)** | SHA-1("147.127.240.72") => 000 => N0 |
| 000 -> K0 | SHA-1("147.127.240.73") => 011 => N3 |
| 001 -> K1 | SHA-1("147.127.240.75") => 110 => N6 |
| 010 -> K2 | |
| 011 -> K3 | SHA-1("s1.mp3") => 001 => K1 |
| 100 -> K4 | SHA-1("s2.mp3") => 100 => K4 |
| 101 -> K5 | SHA-1("d1.doc") => 010 => K2 |
| 110 -> K6 | SHA-1("d2.doc") => 011 => K3 |
| 111 -> K7 | SHA-1("prg.exe") => 111 => K7 |

Next, mapping the Node IP address and sharing files with the chord ring. Chord works in the clockwise directional. Then, node will handle only all previous keys before previous node following the picture below:



**Create Finger Table**

| N0’s finger table | | | | N3’s finger table | | | | N6’s finger table | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $0+2^1$ | 1…1 | N3 | | $3+2^1$ | 4…4 | N6 | | $6+2^1$ | 7…7 | N0 |
| $0+2^2$ | 2…3 | N3 | | $3+2^2$ | 5…6 | N6 | | $6+2^2$ | 0…1 | N0 |
| $0+2^3$ | 4…0 | N6 | | $3+2^3$ | 7…3 | N0 | | $6+2^3$ | 2…6 | N3 |

**Searching Example**

**1. Search with the keyword “s1.mp3” at Node “N0”.**

1. “s1.mp3” assumes that translate to keyword K1.
2. N0 handle only K7 & K0 (that mean N0 know exactly data location of K7 & K0”), then, go to ask other nodes by using finger table.
3. K1 is matched with the first row of N0’s finger table. That’s N3.
4. N3 handle K1, so N3 know exactly data location of K1, then it return address of N0 to user.

This example above is looked confusing since N0 already have K1, but it still forward to ask N3. We will give you another example.

**2. Search with the keyword “prg.exe” at Node “N3”.**

1. “prg.exe” assumes that translate to keyword K7.
2. N3 handle only K1 & K2 (that mean N3 know exactly data location of K1 & K2”), then, go to ask other nodes by using finger table.
3. K7 is matched with the third row of N3’s finger table. That’s N0.
4. N0 handle K7, so N0 know exactly data location of K7, then it return address of N6 to user.

From the example 2, you can see that we do not need to search at node N6, we cross it to N0. If there are so many nodes in the network, how fast it will improve?

Because of routing with the finger table ($2^m$), then the order of searching is O ($\log_2 N$).

**Chord Disadvantage:**

1. Search Matching

There are some disadvantages that is the searching keyword must be exactly matching with the file name because of the SHA-1. Changing only 1 alphabet makes the result of the SHA-1 totally different. Then, there are other stem words (synonym) for other hash values, to improve the searching which may do in the application layer.

In contrast, the different keyword may be matching with the same hash value but this problem had less possible occurs since we use 160-bit ($2^{160}$) for SHA-1.

2. High probability

Chord protocol must be work with high probability. This mean, the generation of the keyword and node-id must be good distribution, otherwise, we cannot gain the benefit of $2^m$ from finger table which makes the chord implementation more difficult.