The Power of Algorithms
(solving scalability of video streaming)

Mikkel Thorup

Center for **B**asic **A**lgorithms **R**esearch **C**openhagen

Algorithms to handle BIG data

Powered by Mathematics

The amount of data grows much faster than computer speeds, so need for efficient algorithms to process data becomes more and more urgent.

Randomized Algorithms

I am particularly fascinated by the use of randomness in computation.

Almost everything is simpler and faster with randomized algorithms. Big Data cannot be handled without randomness.

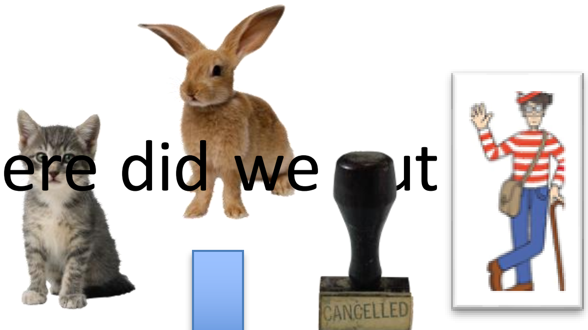
Distribute objects in storage boxes.

What happens on a farm?

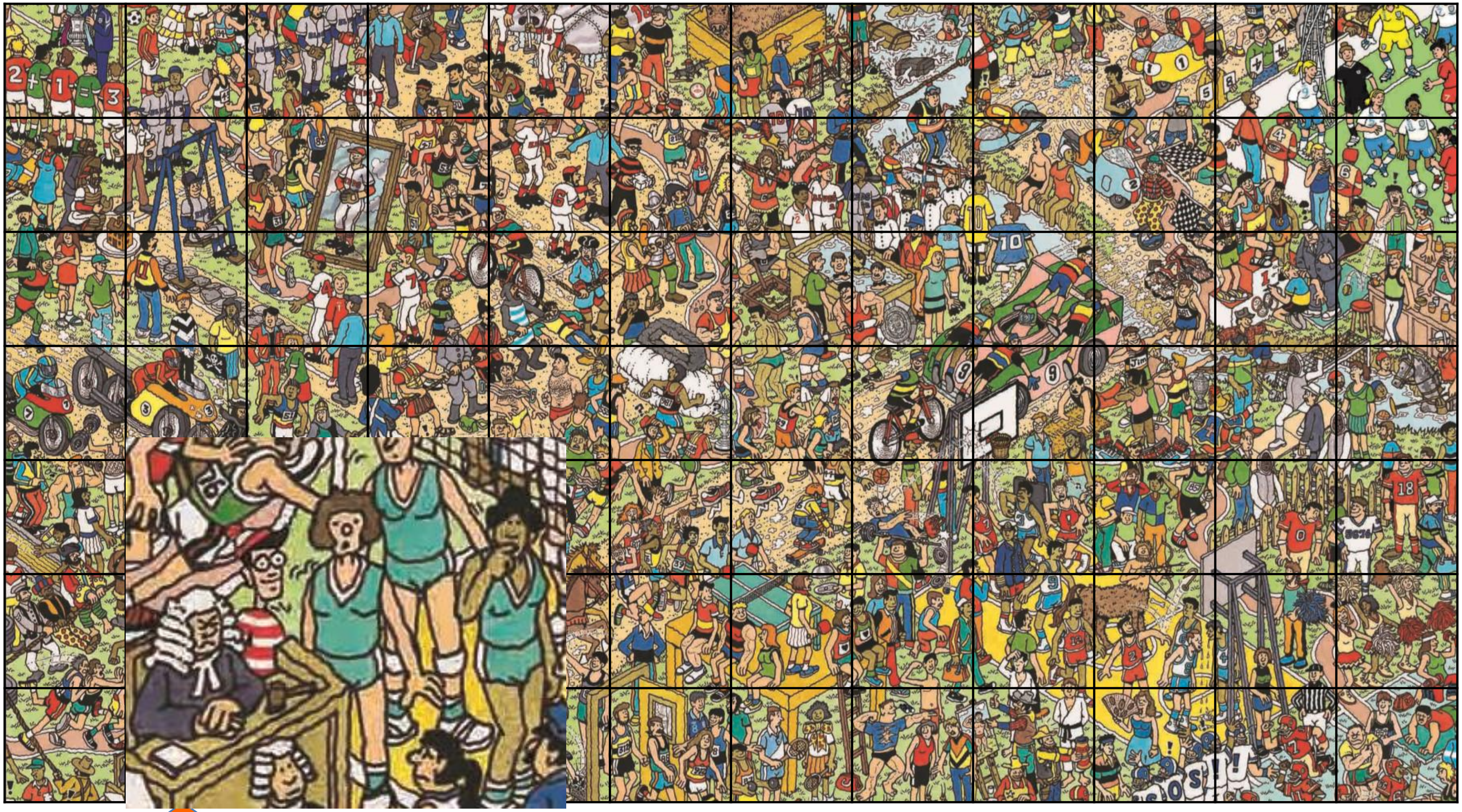
1 Animals 	5	9
2 Office	6	10
3	7	11
4	8	12

Distribute objects in storage boxes.

Where did we put ?

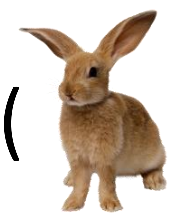


Where is  ?

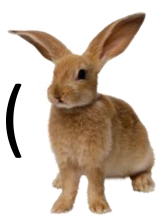
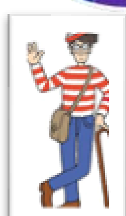


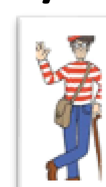
Fully-Random Hash Functions

What we want is a re-computable fully-random hash function h assigning independent random box number 1,...,12 to every possible object:

h() = 

h() = 

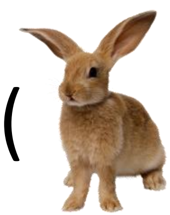
h() = h() with probability 1/12.

With 18 other objects, on average  expected to share box with 18/12=1.5 objects.

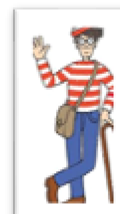
Fully-Random Hash Functions

What we want is a re-computable fully-random hash function h assigning independent random box number 1,…,12 to every possible object:

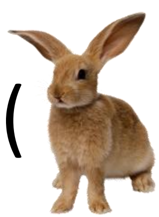
IMPOSSIBLE

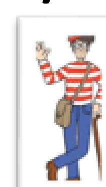
h() = 

10

h() = 

4

h() = h() with probability 1/12.

With 18 other objects, on average  expected to share box with 18/12=1.5 objects.

Random Hash Functions

Re-computable random hash function h assigning random box 1,…,12 to every object.

On computer objects have numbers: **385**, **936**

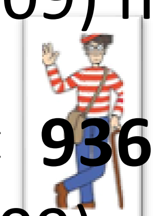
 

Pick two random $\boxed{945}$, $\boxed{749}$ < 1009 (prime)

≠ 0

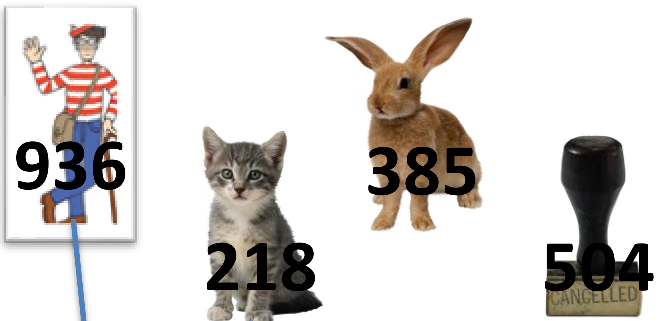
h(**385**) = ((($\boxed{945}$ × **385** + $\boxed{749}$) mod 1009) mod 12) + 1 = 2

h(**936**) = ((($\boxed{945}$ × **936** + $\boxed{749}$) mod 1009) mod 12) + 1 = 5

= prob < 1/12

Distribute objects in storage boxes.



(((936 218 × 679 + 495)

mod 1009) mod 12) +1

= 10 13

Used to store and find things in computers since 1956.

1	5	9
2	6	10
3	7	11
4	8	12

Example using my own research

Company Vimeo

Main competitor of YouTube – 170 million users/month.

Serves about 1 billion requests for video clips per day.

Key technology: Consistent hashing

Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications

Ion Stoica*, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan†
MIT Laboratory for Computer Science
chord@lcs.mit.edu
http://pdos.lcs.mit.edu/chord/

Abstract

A fundamental problem that confronts peer-to-peer applications is

and the state maintained by each node scaling logarithmically with the number of Chord nodes.

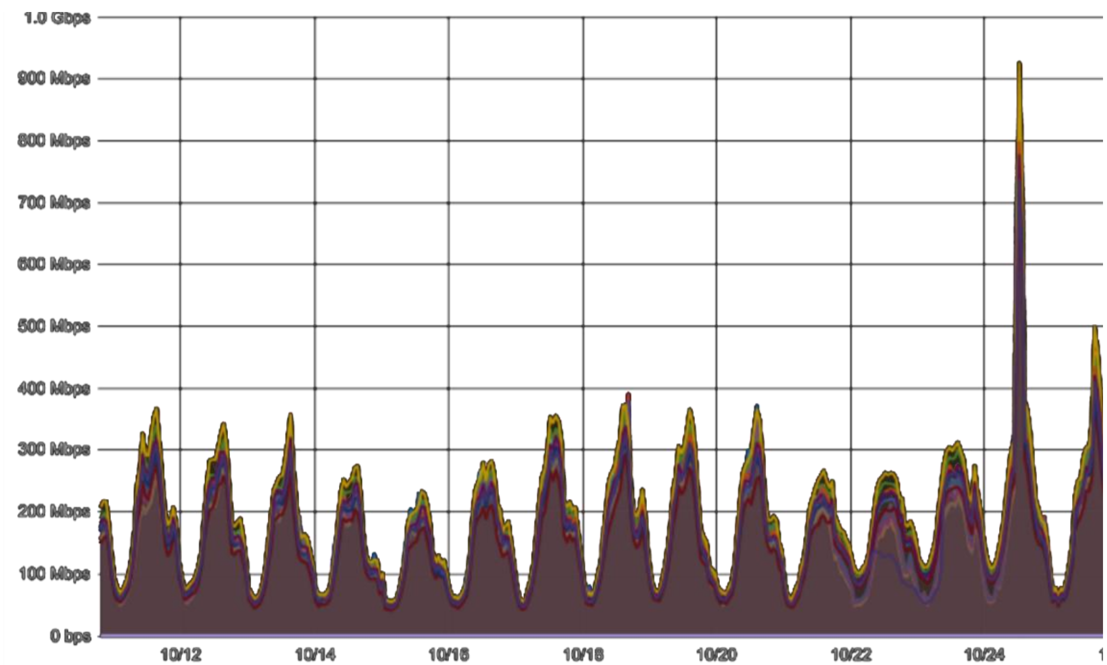
and involves relatively little movement of keys when nodes join and leave the system.

event results in no more than $O(\log^2 N)$ messages.

Three features that distinguish Chord from many other peer-to-peer lookup protocols are its simplicity, provable correctness, and

Title 1–20	Cited by	Year
Chord: A scalable peer-to-peer lookup service for internet applications I Stoica, R Morris, D Karger, MF Kaashoek, H Balakrishnan ACM SIGCOMM Computer Communication Review 31 (4), 149-160	12552	2001

Vimeo’s bandwidth bottleneck



Issue: High bandwidth requirement…

From algorithm theory to industrial reality

Vimeo Engineering Blog Follow

Improving load balancing with a new consistent-hashing algorithm

We run Vimeo's dynamic video packager, Skyfire, in the cloud, serving almost a billion DASH and HLS requests per day. That's a lot! We're very happy with the way that it performs, but scaling it up to today's traffic and beyond has been an interesting challenge. Today I'd like to talk about a new algorithmic development, *bounded-load consistent hashing*, and how it eliminates a bottleneck in our video delivery.

Cornell Uni
Library

arXiv.org > cs > arXiv:

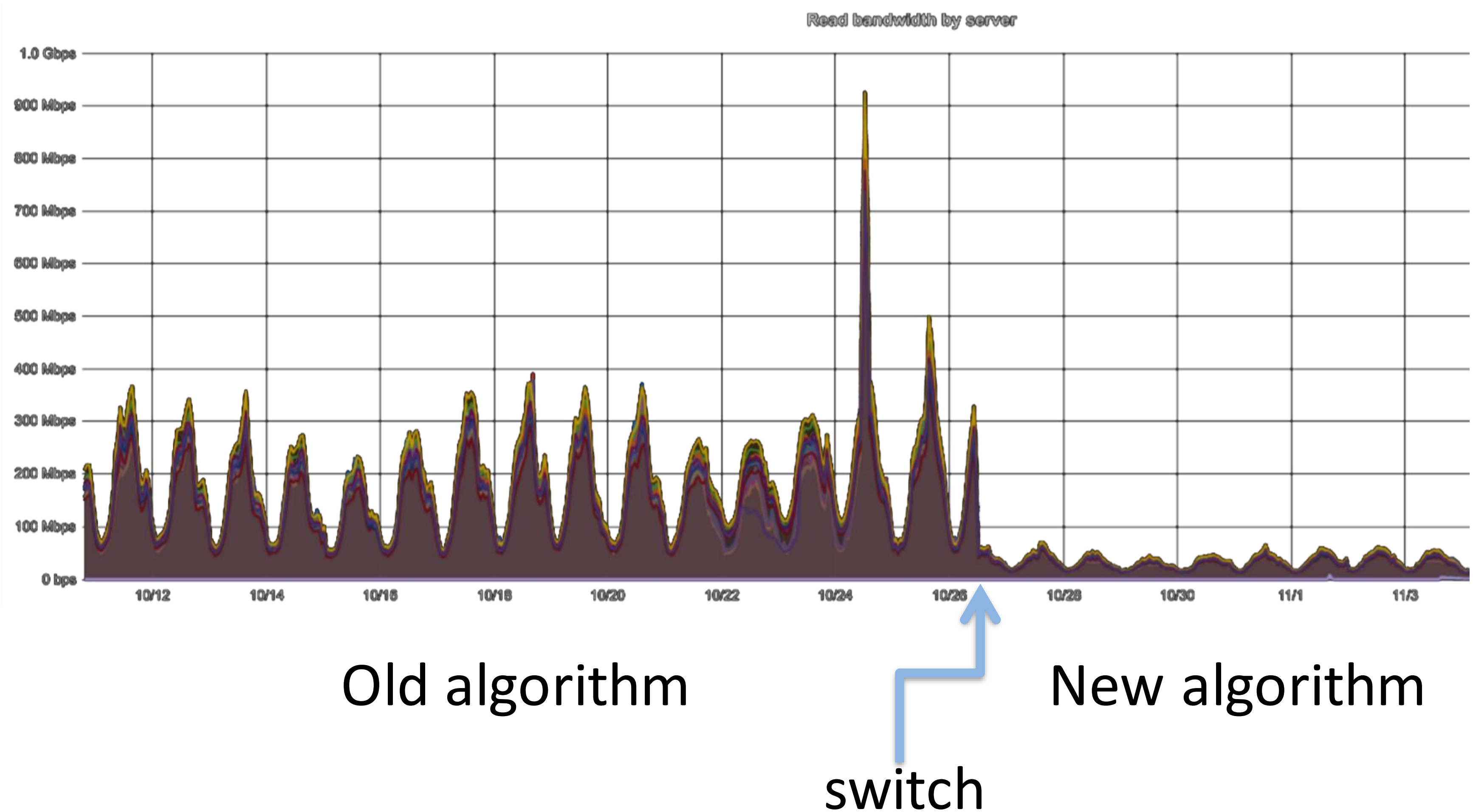
Computer Science > Data Structures and Algorithms

Consistent Hashing with Bounded Loads

Vahab Mirrokni, Mikkel Thorup, Morteza Zadimoghaddam

(Submitted on 3 Aug 2016)

Eliminating the bandwidth bottleneck



Classic Consistent Hashing (unbounded loads)

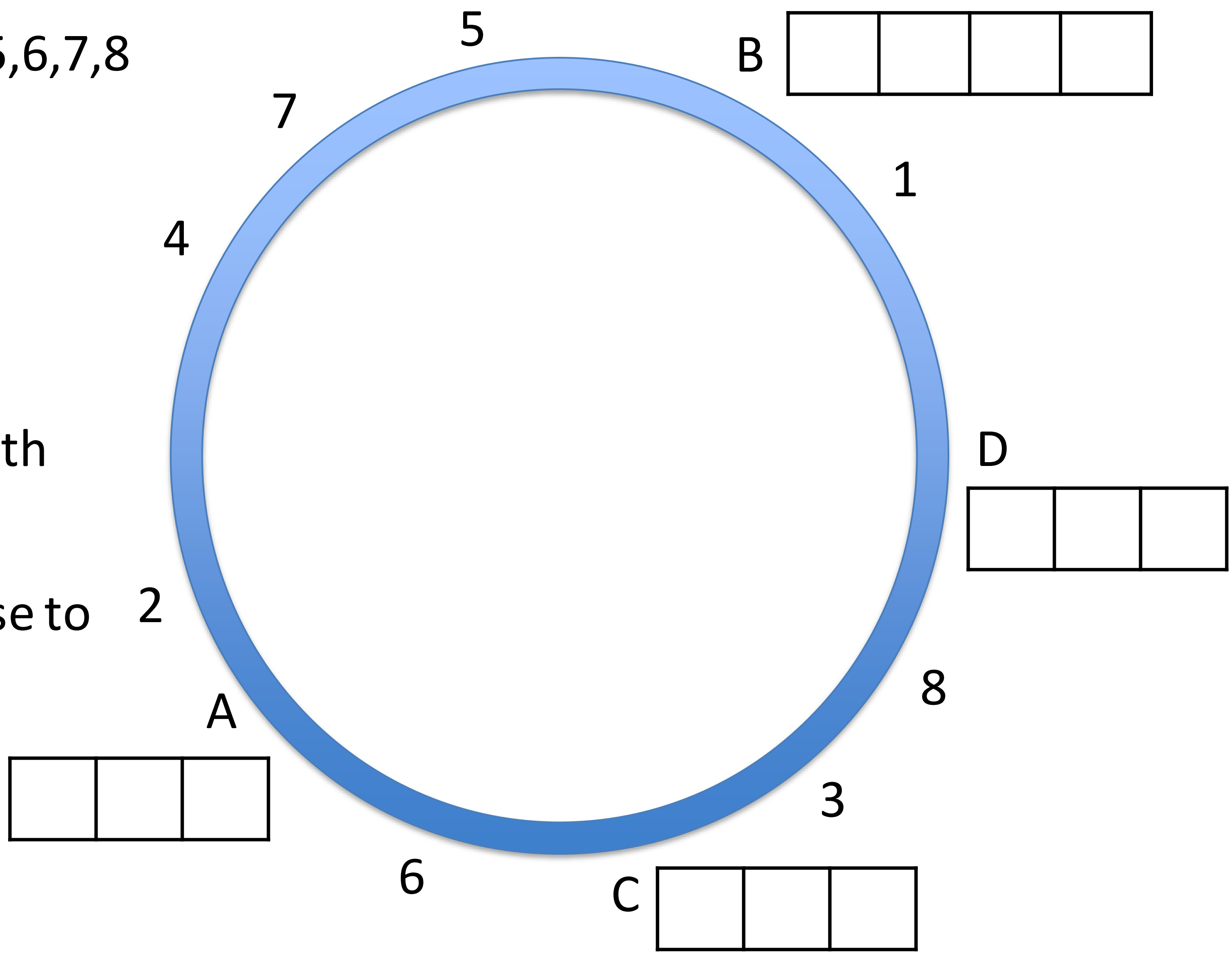
- Problem:
  - Assign clients to servers so server of client easy to find.
  - Dynamic system where both clients and servers can join and leave.
  - Reassign as few clients as possible.
- Algorithmic Solution:
  - Map clients and servers to cycle using random hash function.
  - Client goes clockwise to first server.

Consistent hashing (unbounded loads)

Clients 1,2,3,4,5,6,7,8
Servers A,B,C,D

Map to cycle with
hash function.

Client clock-wise to
first server.

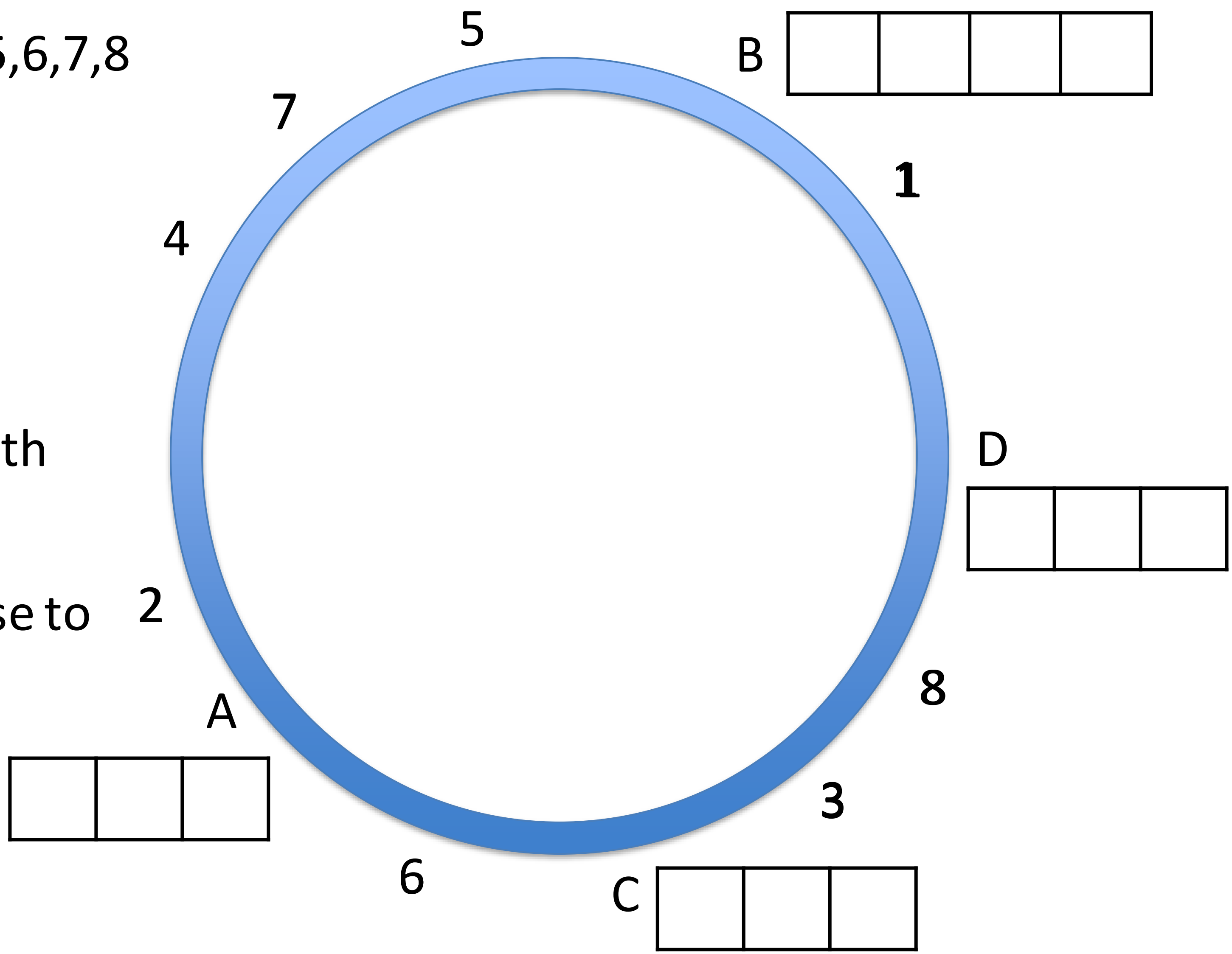


Consistent hashing (unbounded loads)

Clients 1,2,3,4,5,6,7,8
Servers A,B,C,D

Map to cycle with hash function.

Client clock-wise to first server.



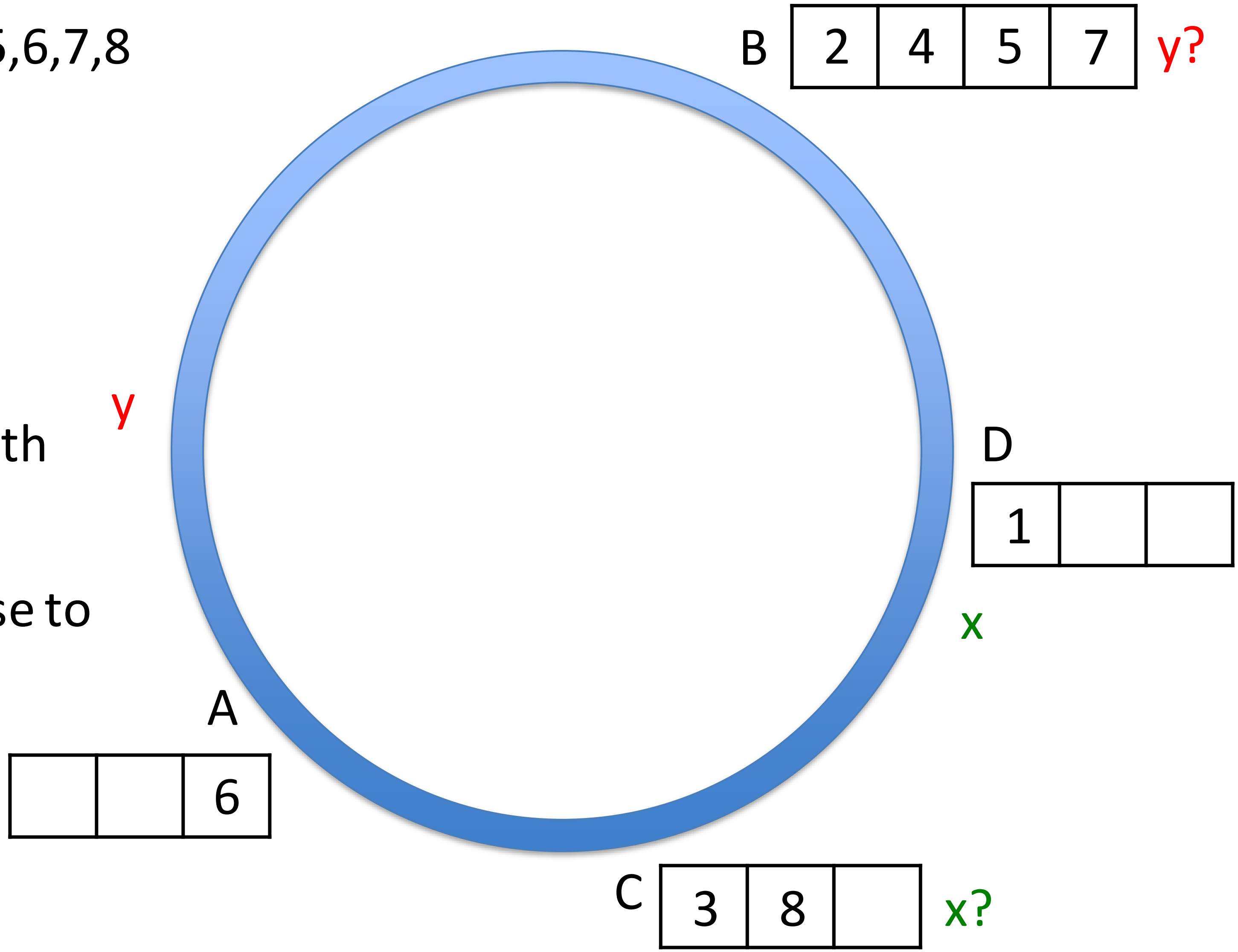
Consistent hashing (unbounded loads)

Who serves client x? y?

Clients 1,2,3,4,5,6,7,8
Servers A,B,C,D

Map to cycle with hash function.

Client clock-wise to first server.



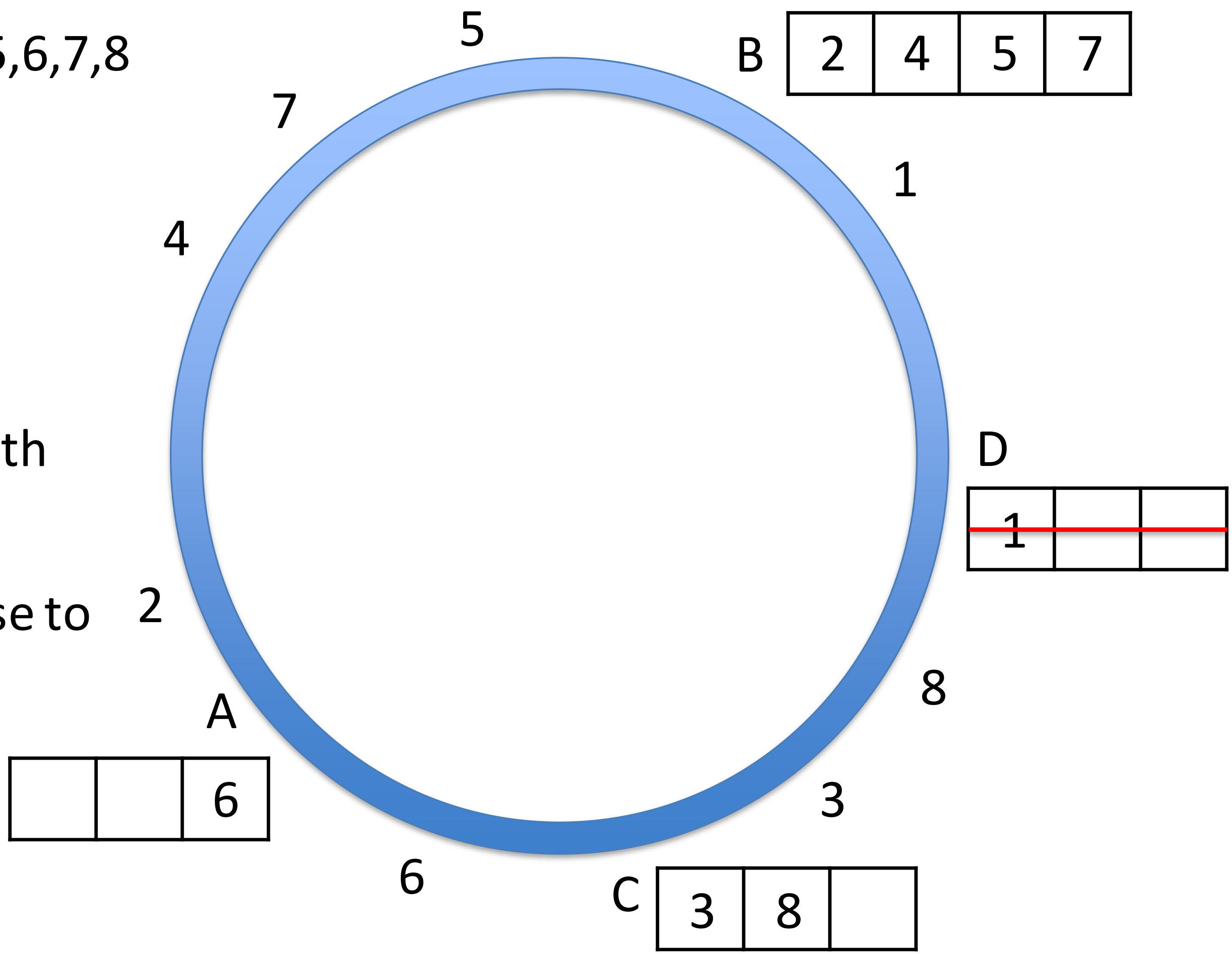
Consistent hashing (unbounded loads)

Server D leaves

If server D arrives, we do the opposite

Clients 1,2,3,4,5,6,7,8

Servers A,B,C,~~D~~



Map to cycle with hash function.

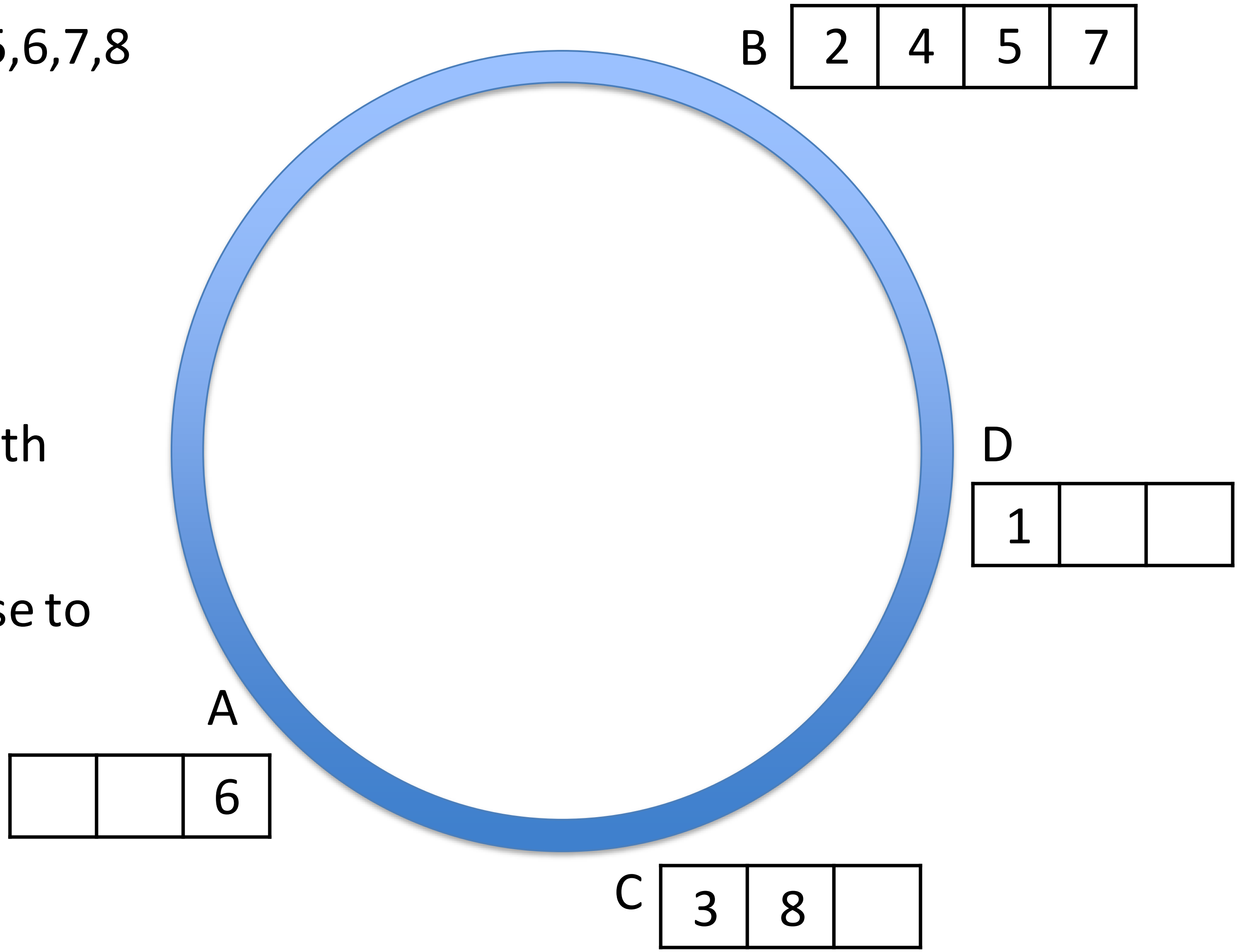
Client clock-wise to first server.

Consistent hashing (unbounded loads)

Clients 1,2,3,4,5,6,7,8
Servers A,B,C,D

Map to cycle with hash function.

Client clock-wise to first server.



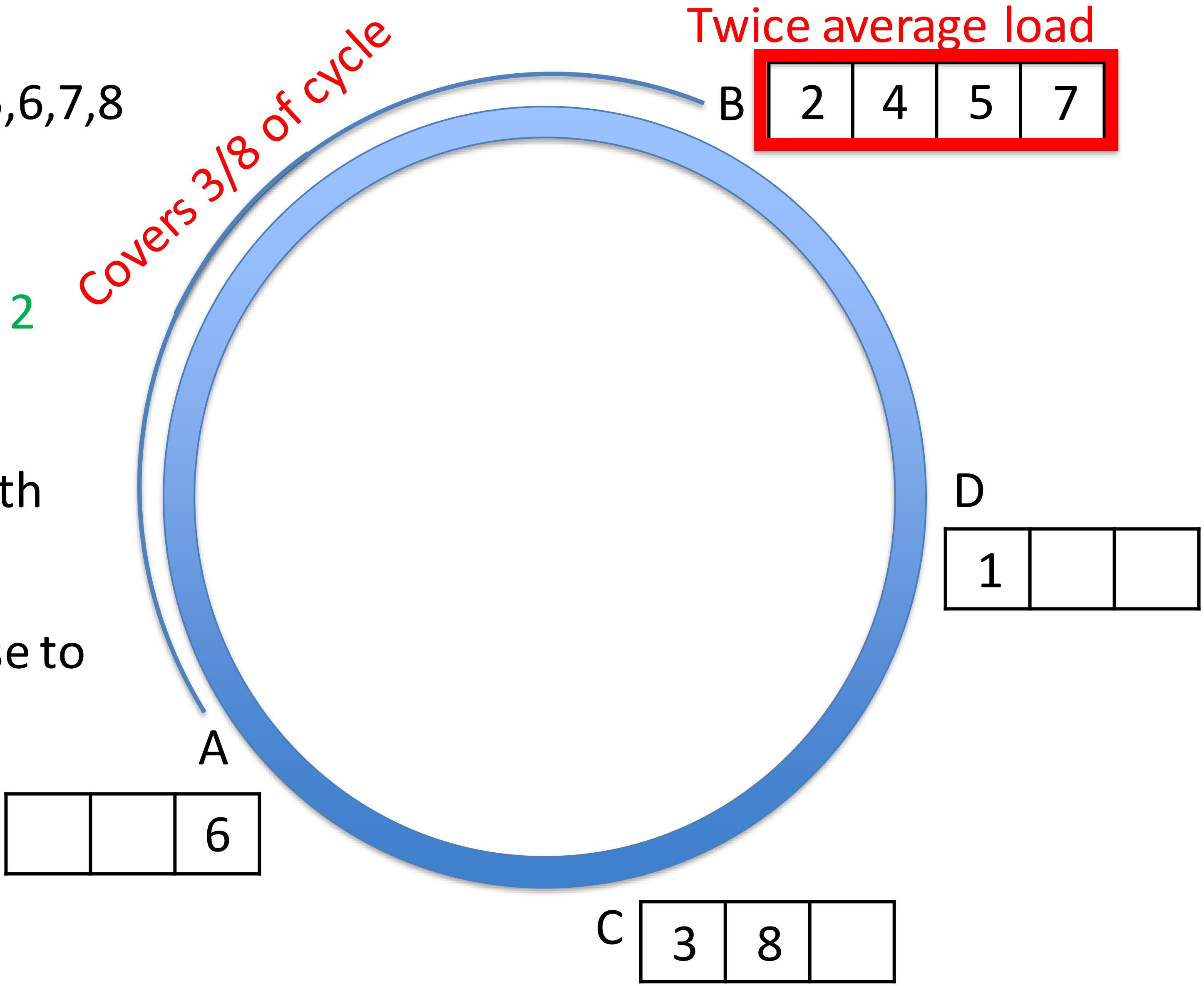
Consistent hashing (unbounded loads)

Clients 1,2,3,4,5,6,7,8
Servers A,B,C,D

Aver. load 8/4= 2

Map to cycle with hash function.

Client clock-wise to first server.



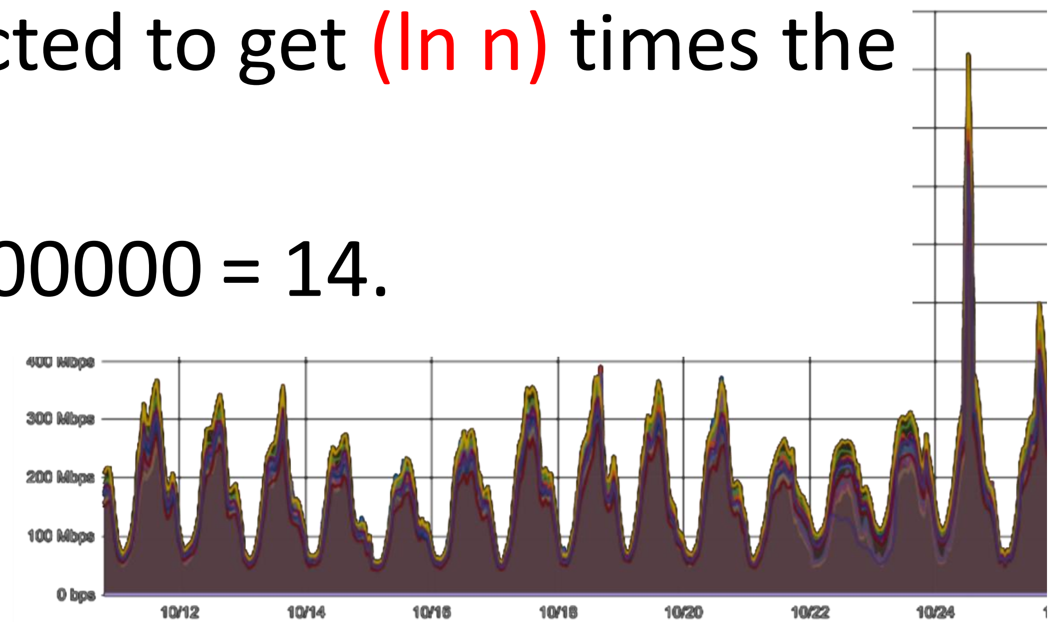
Consisten Hashing (Unbounded Loads)

If we randomly place n servers on cycle, and each covers segment from preceeding server, then expect some server to cover fraction

$$(\ln n)/n$$

Such server expected to get $(\ln n)$ times the average load.

$\ln 1000 = 7$, $\ln 1000000 = 14$.



Consistent hashing with bounded loads

- Problem:
  - Assign clients to servers: server of client easy to find.
  - Dynamic system where both clients and servers can join and leave. Reassign as few clients as possible.
  - No server has more than $1.5 \times$ average number of clients (the load bound).
- Our Algorithmic Solution:
  - Map clients and servers to cycle using random hash function.
  - Client goes clockwise to first non-full server.

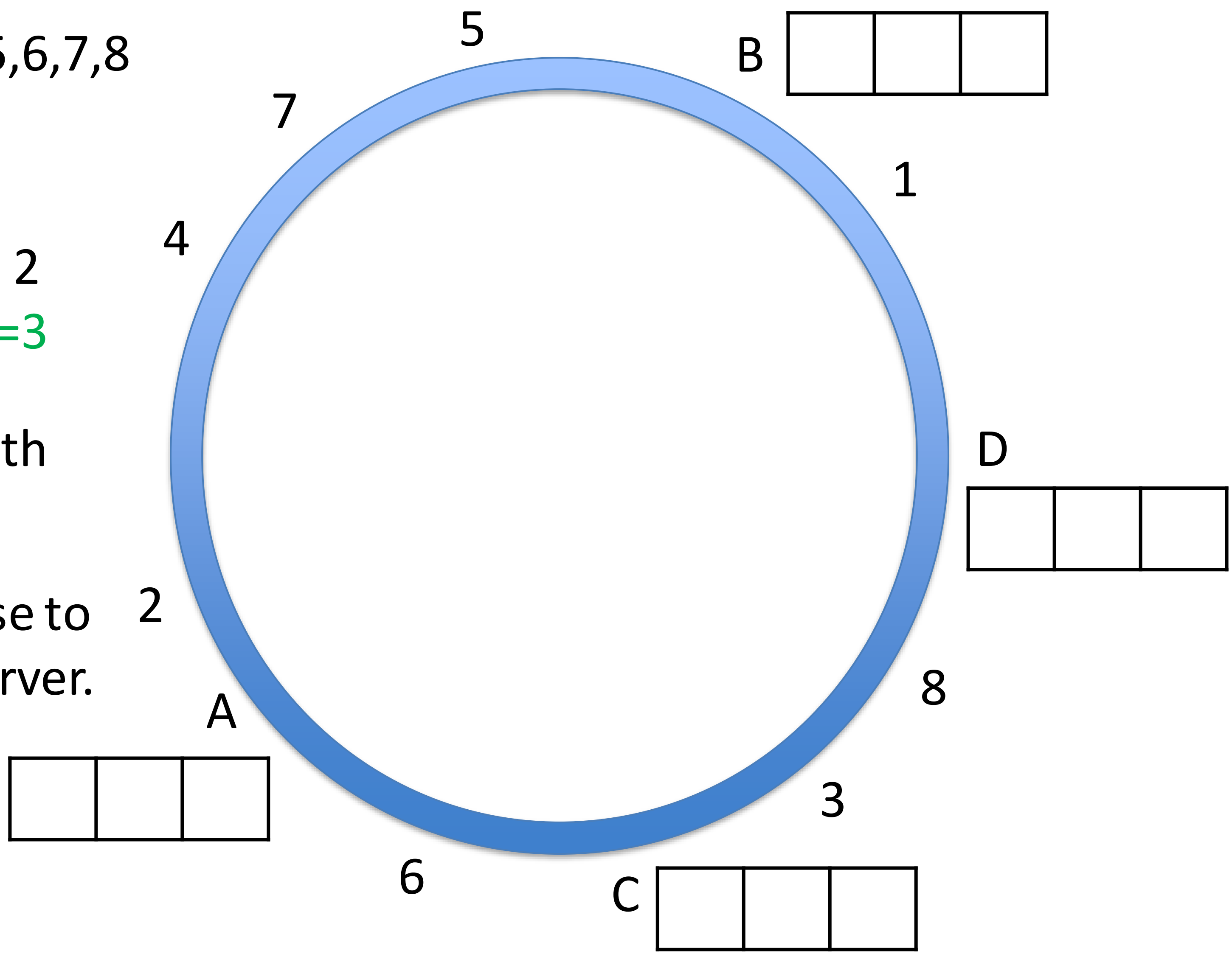
Consistent hashing with bounded loads

Clients 1,2,3,4,5,6,7,8
Servers A,B,C,D

Aver. load 8/4= 2
Max load 1.5×2=3

Map to cycle with hash function.

Client clock-wise to first non-full server.



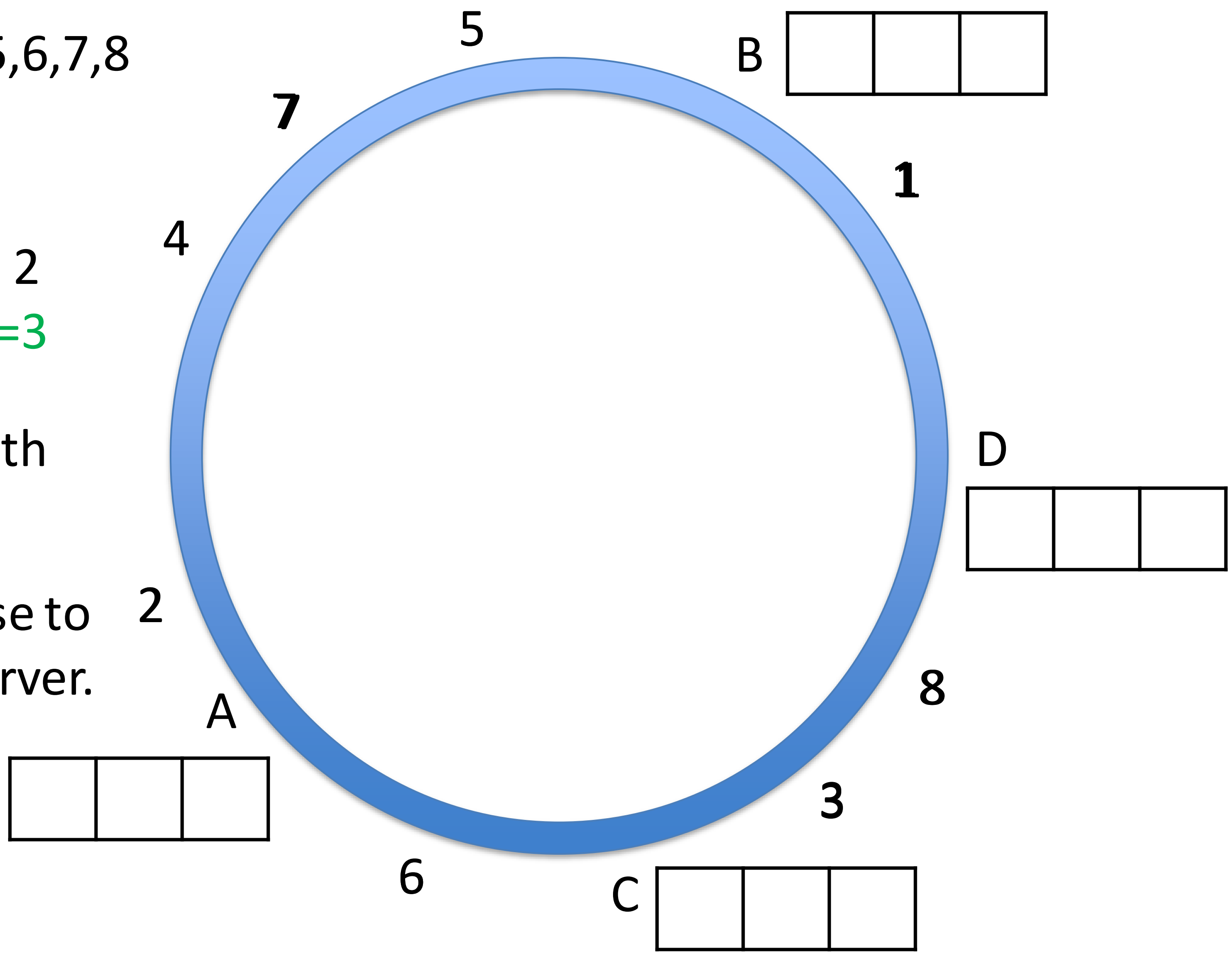
Consistent hashing with bounded loads

Clients 1,2,3,4,5,6,7,8
Servers A,B,C,D

Aver. load 8/4= 2
Max load $1.5 \times 2 = 3$

Map to cycle with hash function.

Client clock-wise to first non-full server.



Consistent hashing with bounded loads

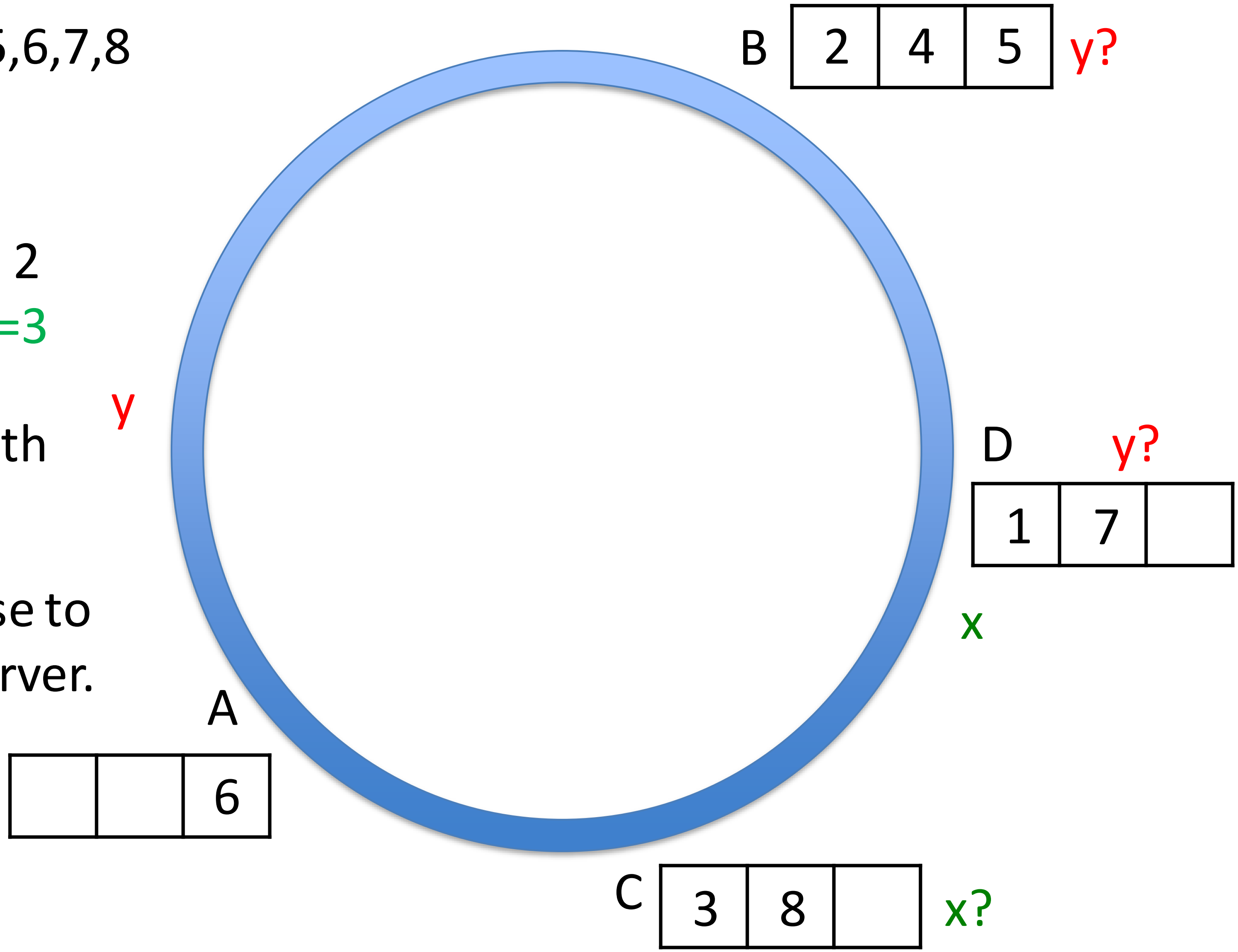
Who serves client x? y?

Clients 1,2,3,4,5,6,7,8
Servers A,B,C,D

Aver. load 8/4= 2
Max load 1.5×2=3

Map to cycle with hash function.

Client clock-wise to first non-full server.



Consistent hashing with bounded loads

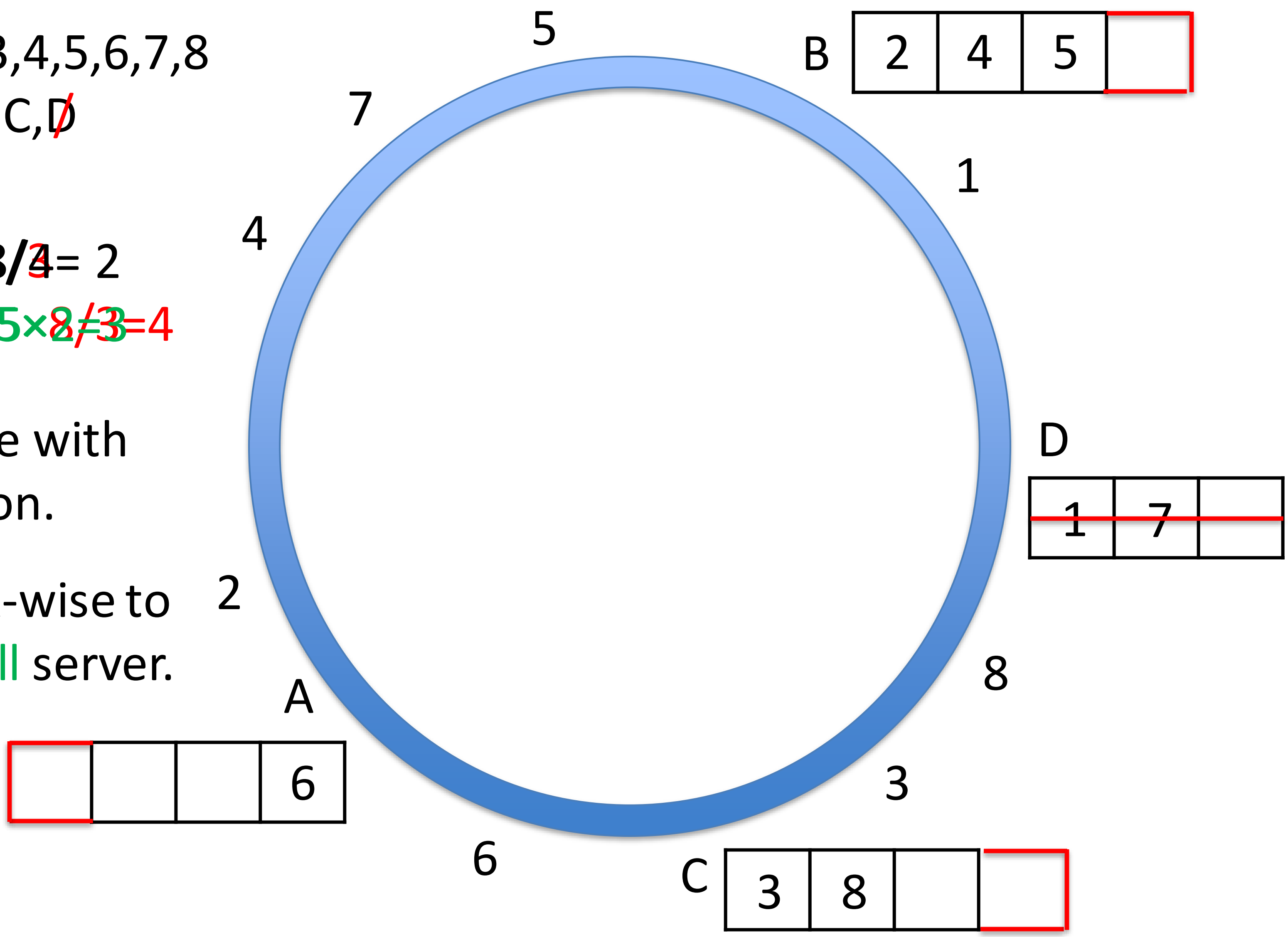
Server D leaves - more complicated ..

Clients 1,2,3,4,5,6,7,8
Servers A,B,C,~~D~~

Aver. load 8/~~4~~3= 2
Max load 1.5×~~2=3~~8/3=4

Map to cycle with hash function.

Client clock-wise to first non-full server.



Cost of Consistent hashing with bounded loads

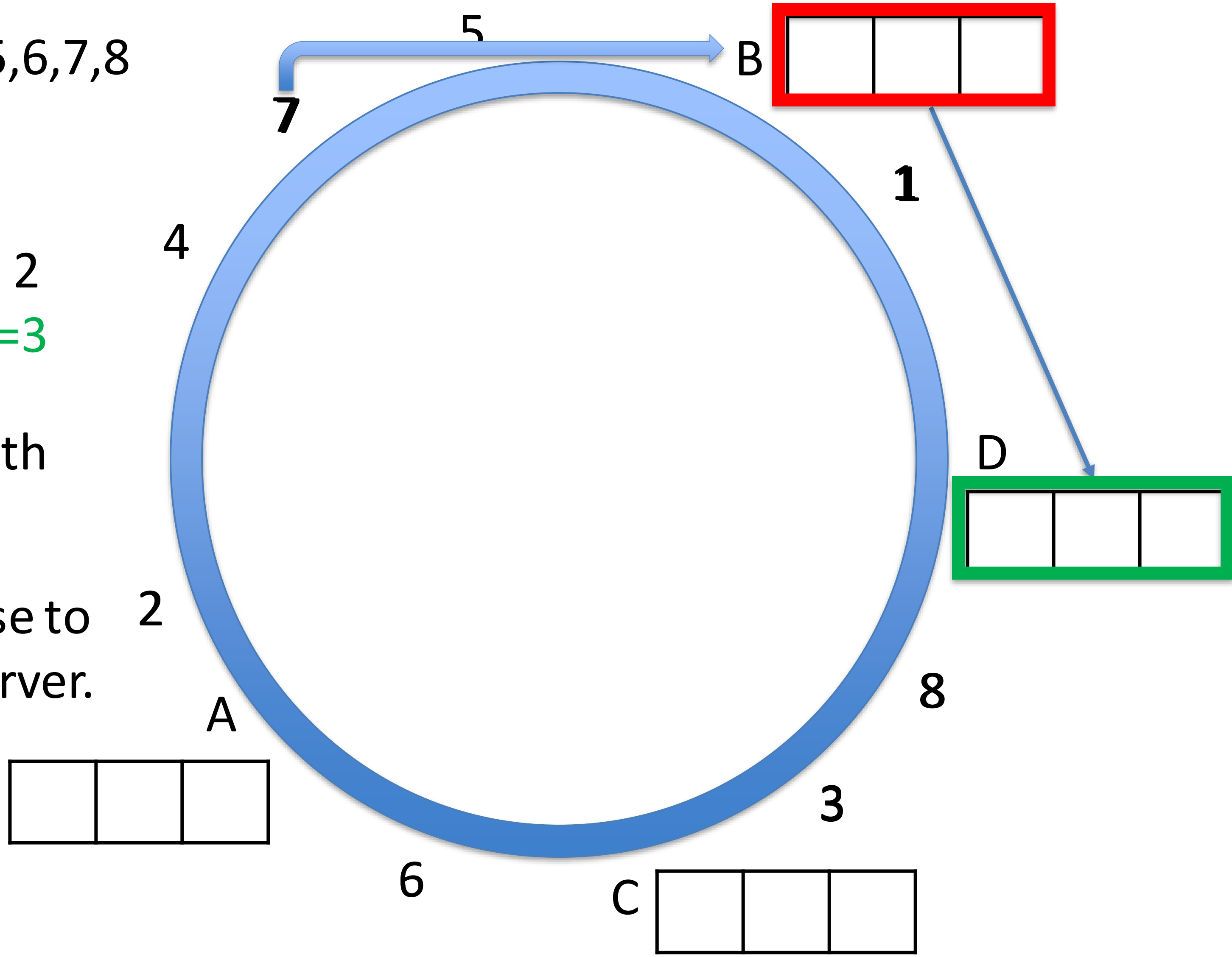
How many full passed on way to non-full ? 1

Clients 1,2,3,4,5,6,7,8
Servers A,B,C,D

Aver. load 8/4= 2
Max load 1.5×2=3

Map to cycle with hash function.

Client clock-wise to first non-full server.



Consistent hashing with bounded loads

Theorem With load-bound = $(1 + \varepsilon) \times$ aver-load, the expected number of full servers passed to non-full is proportional to $1/\varepsilon^2$.

For example, with $\varepsilon = 0.1 = 10\%$, $1/\varepsilon^2$=100

The bound holds no matter the number of clients and servers which for Vimeo approaches billions.

Basic algorithmic research with many applications

- Our algorithm has no details specific to video streaming. Works for *any* dynamic allocation system in the world – now used also in Google's cloud and other companies.
- Mathematical analysis based on properties of degree-4 polynomials with random coefficients – the theory of which was originally developed with other applications in mind.

Lemma 10. *The expected number of balls hashing directly to any*
expected number of balls forwarded into q from its predecessor q^- is
not active, and its active successor q^+ is given an extra capacity of o
bins starting from q^+ is $O((\log c)/c^2)$.

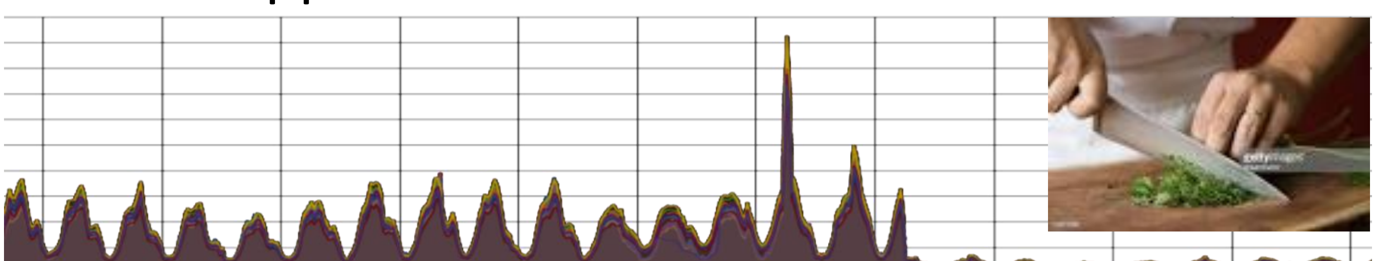
Proof. For the first statement, we note that the expected number of ba
n/r for any $0 \leq i \leq r$. These are not added to q if some bin hash to $[h$
event because balls and bins hash independently. The expected numb
is $\mu = i(n-1)/r$. For $i \geq r/(n-1)$, we have $\mu \geq 1$, and then, by
in $[h(q) - i, h(q))$ is $O((\mu + \mu^2)/(\mu - 0)^4) = O(1/\mu^2) = O((r/($
hashing directly to q is thus bounded by

$$n/r \cdot \left(\lfloor r/(n-1) \rfloor + \sum_{i=\lfloor r/(n-1) \rfloor + 1}^{\infty} (r/(ni))^2 \right.$$

We also have to consider the probability that the preceding bin q^- fo
we would need q^- to be filled even if we increased its capacity by 1
least 2. This is bounded by the probability of having an interval $I \ni$
bins including one with capacity at least 2. This is what we analyze
$\Pr[d \geq 1] \leq \mathbf{E}[d] = O((\log c/c^2)$. By the capacity constraint, the
forwarded to and end in q is $2cm/n$, so the expected number is

$$O((\log c/c^2) 2cm/n = O((m/n)(\log$$

Next we ask for the expected number d of full bins starting from the
bin q, when q^+ is given an extra capacity of one. Again this implies th
the analysis from the proof of Lemma 9 implies that $\mathbf{E}[d] = O((\log c$



Consistent hashing with bounded loads

Theorem With load-bound = $(1+\varepsilon)$ $\times$ aver-load, the expected number of full servers passed to non-full is proportional to $1/\varepsilon^2$.

Recent improvement with new algorithm to:

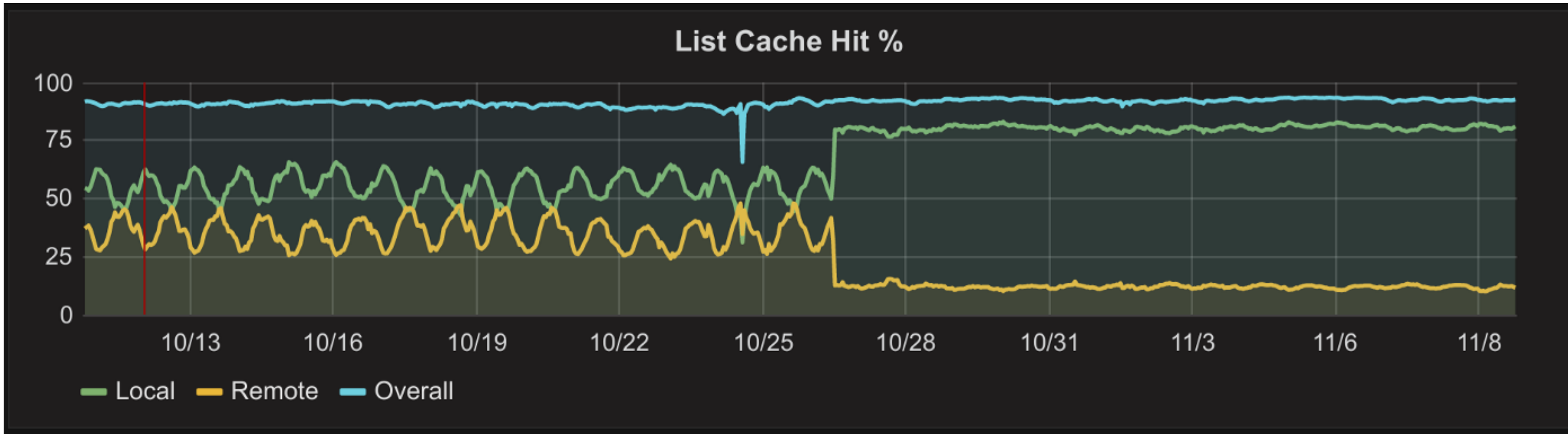
Theorem With load-bound = $(1+\varepsilon)$ $\times$ aver-load, the expected number of full servers passed to non-full is proportional to $1/\varepsilon$

For example, with $\varepsilon = 0.01 = 1\%$,

$1/\varepsilon^2$ = 10000 improved to $1/\varepsilon$ = 100

The new bound is the best possible. Nothing better can ever be done.

Energy saving in servers



- Green line is when clients served locally. Yellow is remote.
- Server farms emit more CO_2 than all air traffic.