

COS3043

System Fundamentals

Lecture 10

Topics

1.	Abstractions 1.1 Hardware Resources 1.2 OS Functionality 1.3 Managing the CPU and Memory
2.	OS Structure 2.1 SPIN Approach 2.2 Exokernel Approach 2.3 L3/L4 Micro-Kernel Approach
3.	Virtualization 3.1 Intro to Virtualization 3.2 Memory Virtualization 3.3 CPU and Device Virtualization
4.	Parallelism 4.1 Shared Memory Machines 4.2 Synchronization 4.3 Communication 4.4 Scheduling
5.	Distributed Systems 5.1 Definitions 5.2 Lamport Clocks 5.3 Latency Limit

6.	Distributed Object Technology 6.1 Spring Operating System 6.2 Java RMI 6.3 Enterprise Java Beans
7.	Design and Implementation of Distributed Services 7.1 Global Memory System 7.2 Distributed Shared Memory 7.3 Distributed File System
8.	System Recovery 8.1 Lightweight Recoverable Virtual Memory 8.2 Rio Vista 8.3 Quicksilver
9.	Internet Scale Computing 9.1 GiantScale Services 9.2 Content Delivery Networks 9.3 MapReduce
10.	Real-Time and Multimedia 10.1 Persistent Temporal Streams

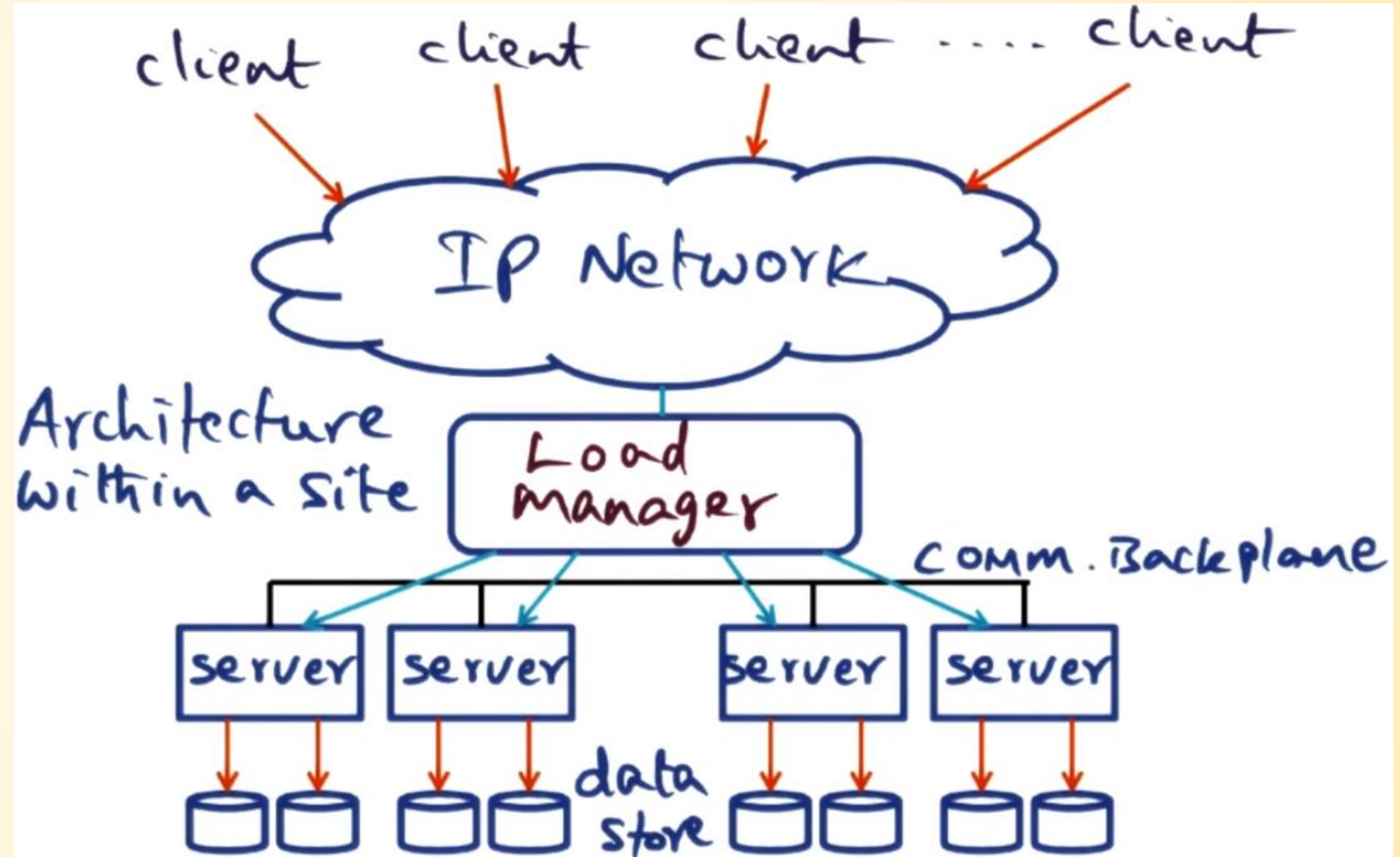
List of Discussion

- Giant Scale Services
- Content Delivery Network
- Map-Reduce

Giant Scale Services (GSS)



Genetic Service Model - GSS



Cluster as Workhorses

Circa 2000

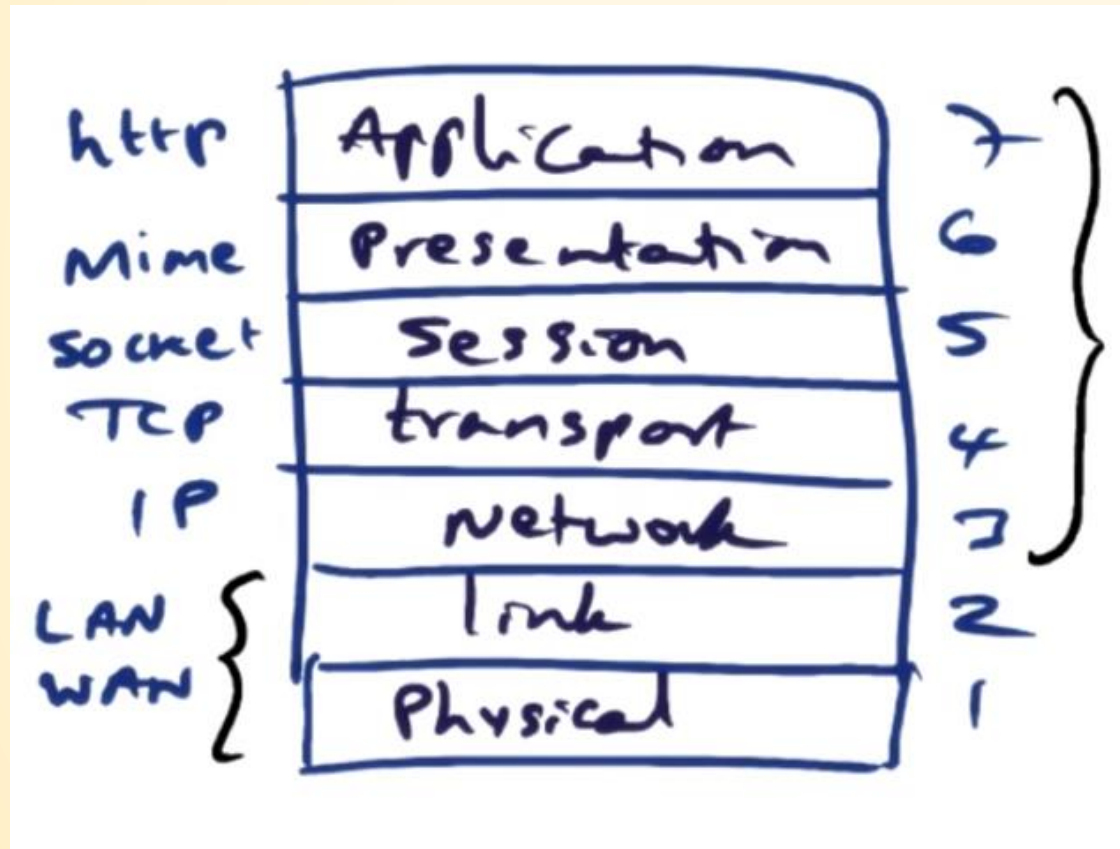
Table A. Example clusters for giant-scale services.

Service	Nodes	Queries	Nodes
AOL Web cache	>1,000	10B/day	4-CPU DEC 4100s
Inktomi search engine	>1,000	>80M/day	2-CPU Sun Workstations
Geocities	>300	>25M/day	PC Based
Anonymous Web-based e-mail	>5,000	>1B/day	FreeBSD PCs

10x to 100x
today compared
to circa 2000

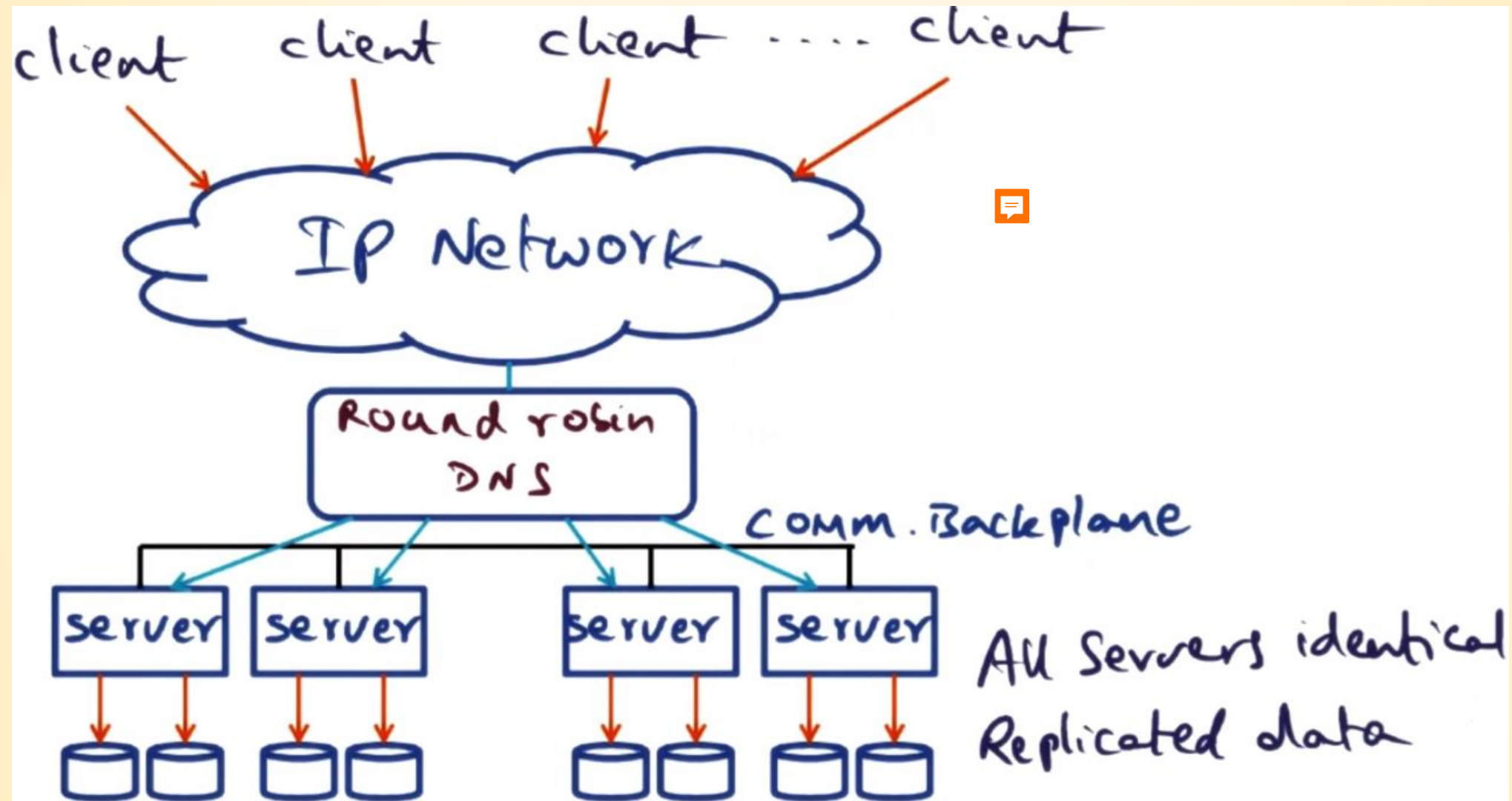
Load Management Choices

OSI Reference Model

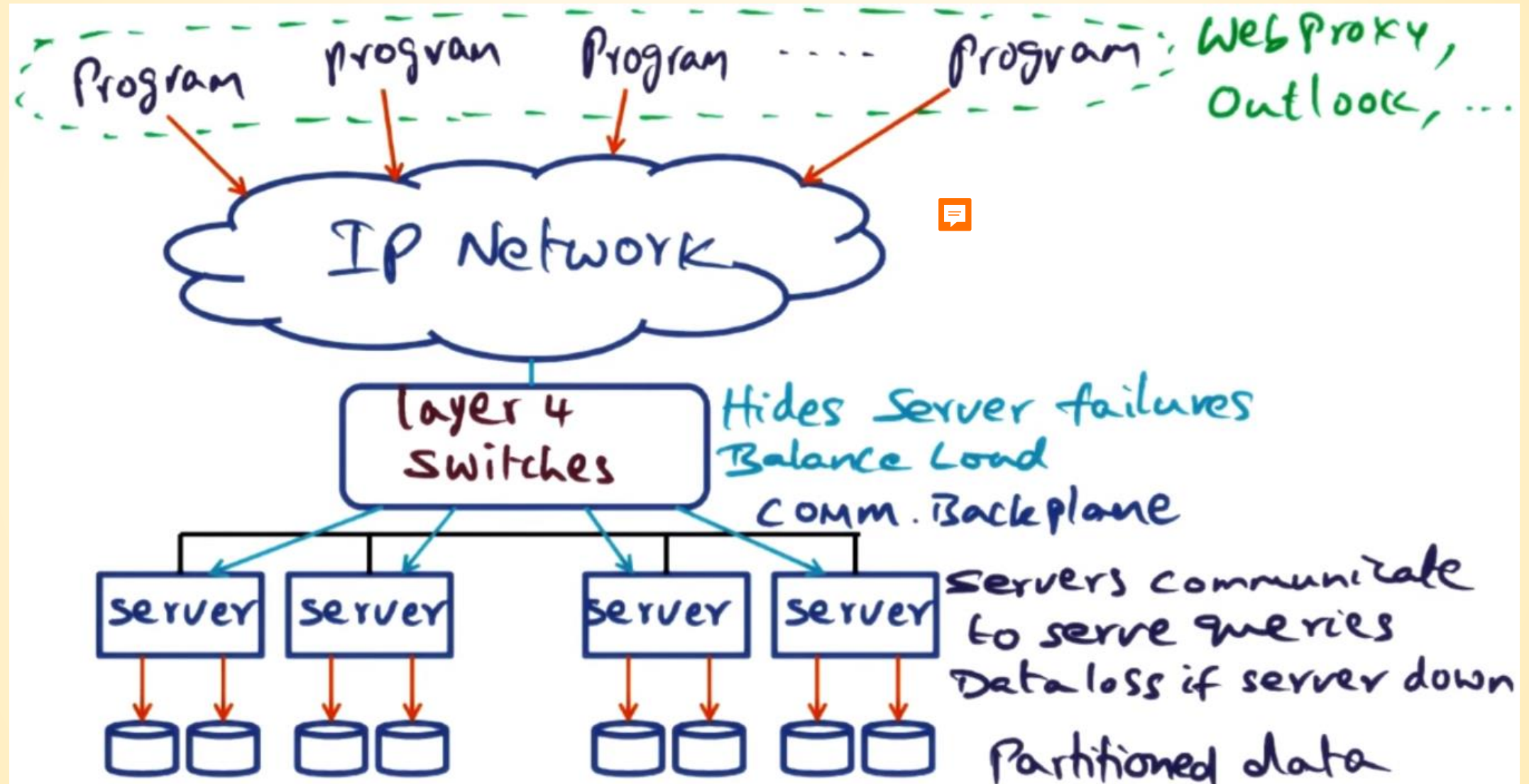


- Increasing functionality of load manager

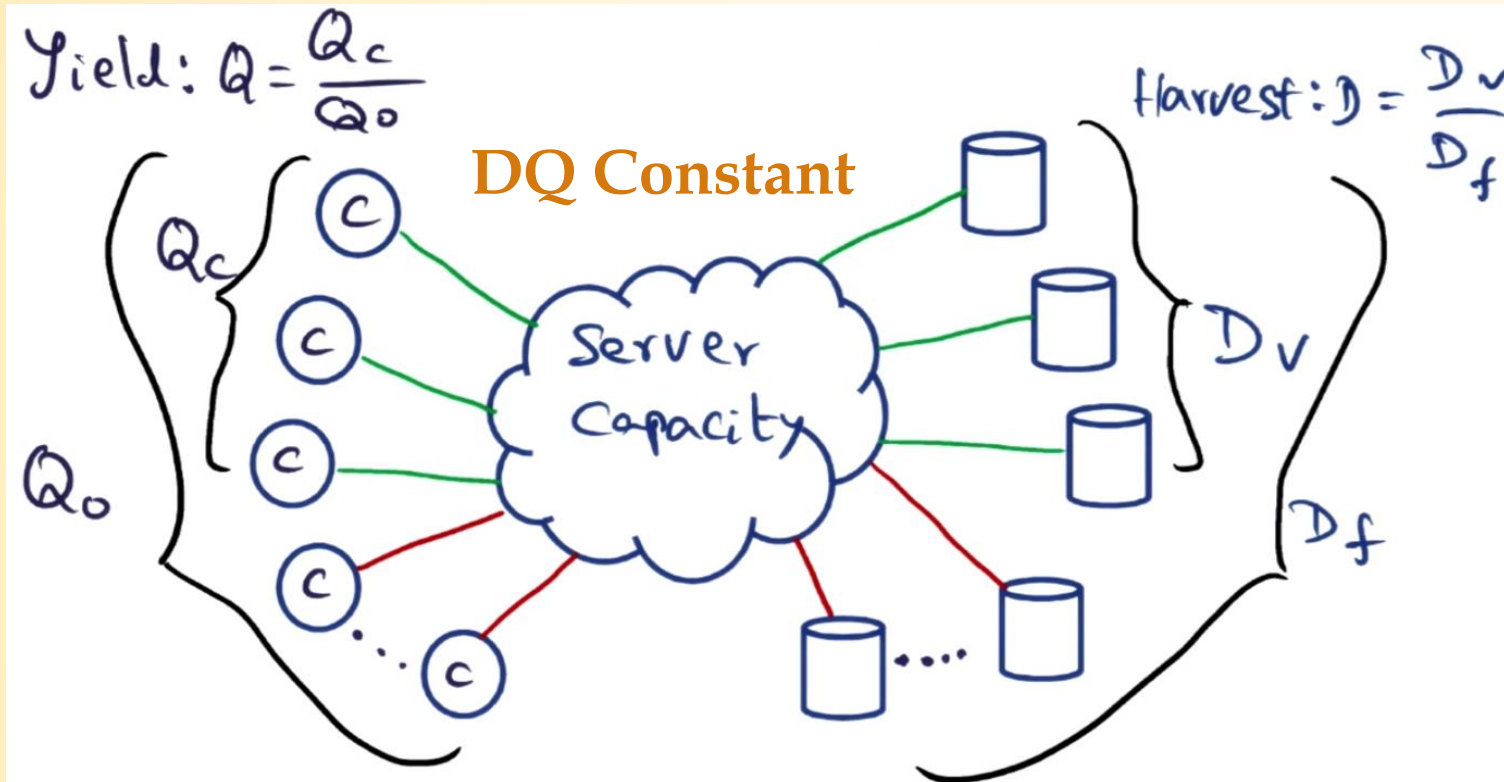
Load Manager at Network Level



Load Manager at Transport Level or Higher



DQ Principle



Definitions:

Q_c = completed requests

Q_o = offered load

Yield Q (ratio between 0 and 1)

D_v = available data

D_f = full corpus of data

Harvest D (ratio between 0 and 1)

Product of DQ is constant

– Defines server capacity for system admin

Replication Vs. Partition



D unchanged

$Q \downarrow$



$D \downarrow$

Q unchanged

Graceful Degradation

How do we deal with server **saturation**?

Note that DQ defines **capacity**.

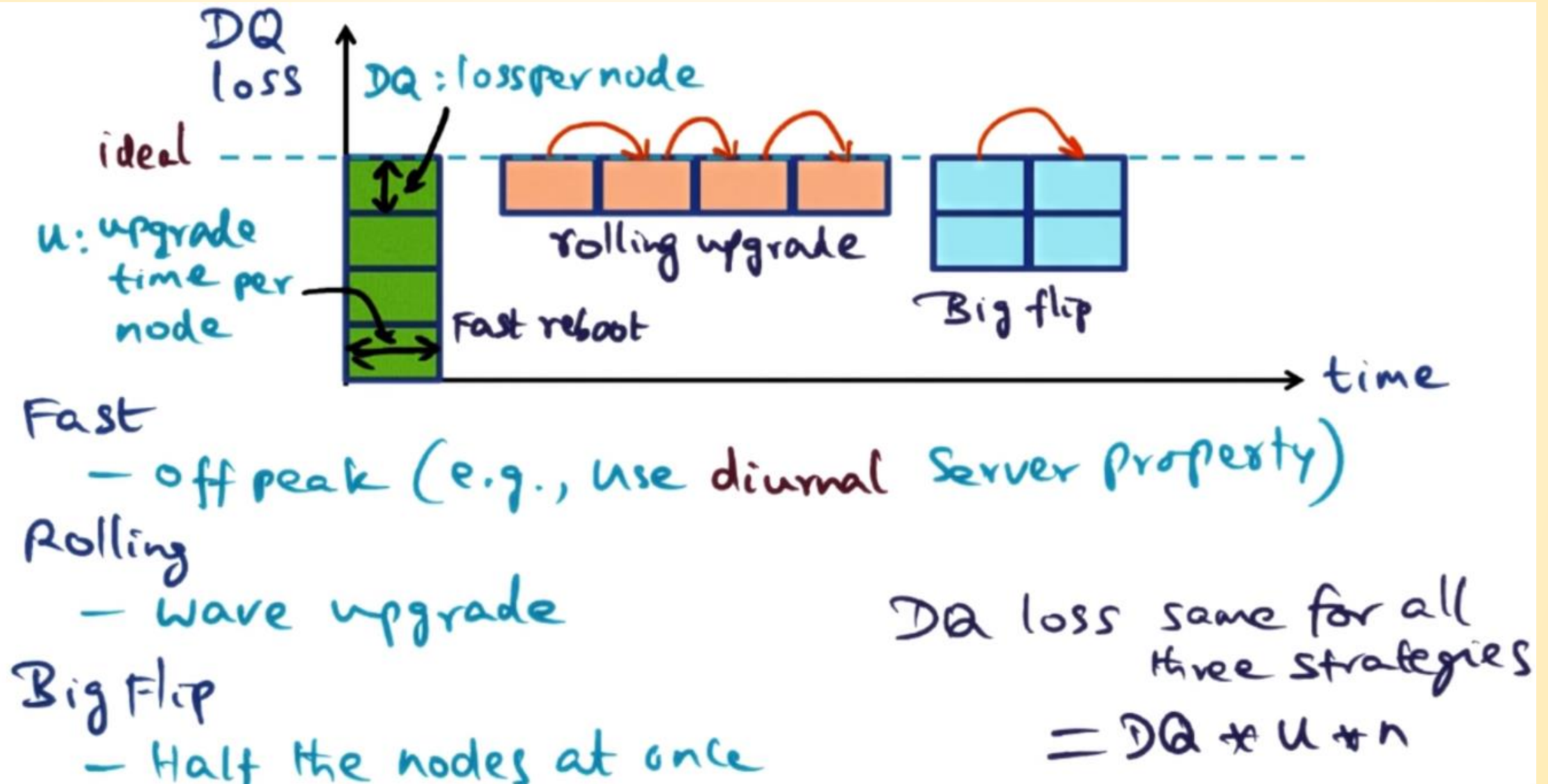
Decision of which to sacrifice and which to keep constant,
i.e.:

- D fixed; $Q \downarrow$
- $D \downarrow$; Q fixed

→ System designer decides to optimize either for yield or for harvest.

Discuss the scenario of a video server.

Online Evolution and Growth



Further Own Reading

Lessons from Giant-Scale Services

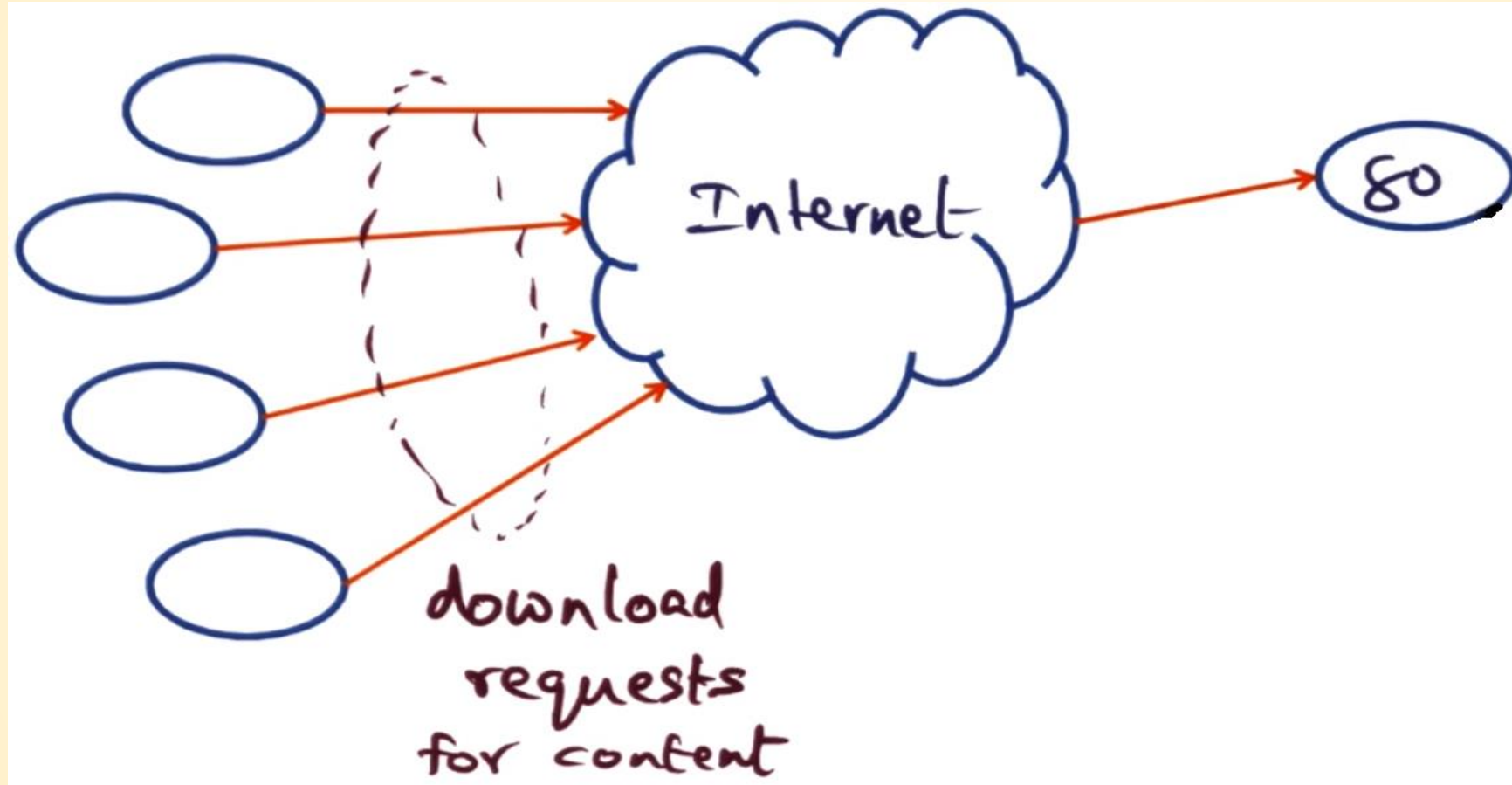
PDF: <https://people.eecs.berkeley.edu/~brewer/papers/GiantScale.pdf>

Reading checklists for Giant-Scale Services:

- Reasons for the success of these giant-scale services.
- Load Manager
- DQ Principle
- Replication vs Partition
- Graceful Degradation
- Online Evolution and Growth

Content Delivery Network (CDN)

HTTP(S) Request – Potential Problem



→ Leads to Origin Server Overload

Solutions to Origin Server Overload

Two solutions



1. Web Proxy

- not good enough for live video content

2. Content Delivery Network (CDN)

- content mirrored stored geographically.
- user request dynamically re-routed to geo-local mirror

Content Delivery Network (CDN)

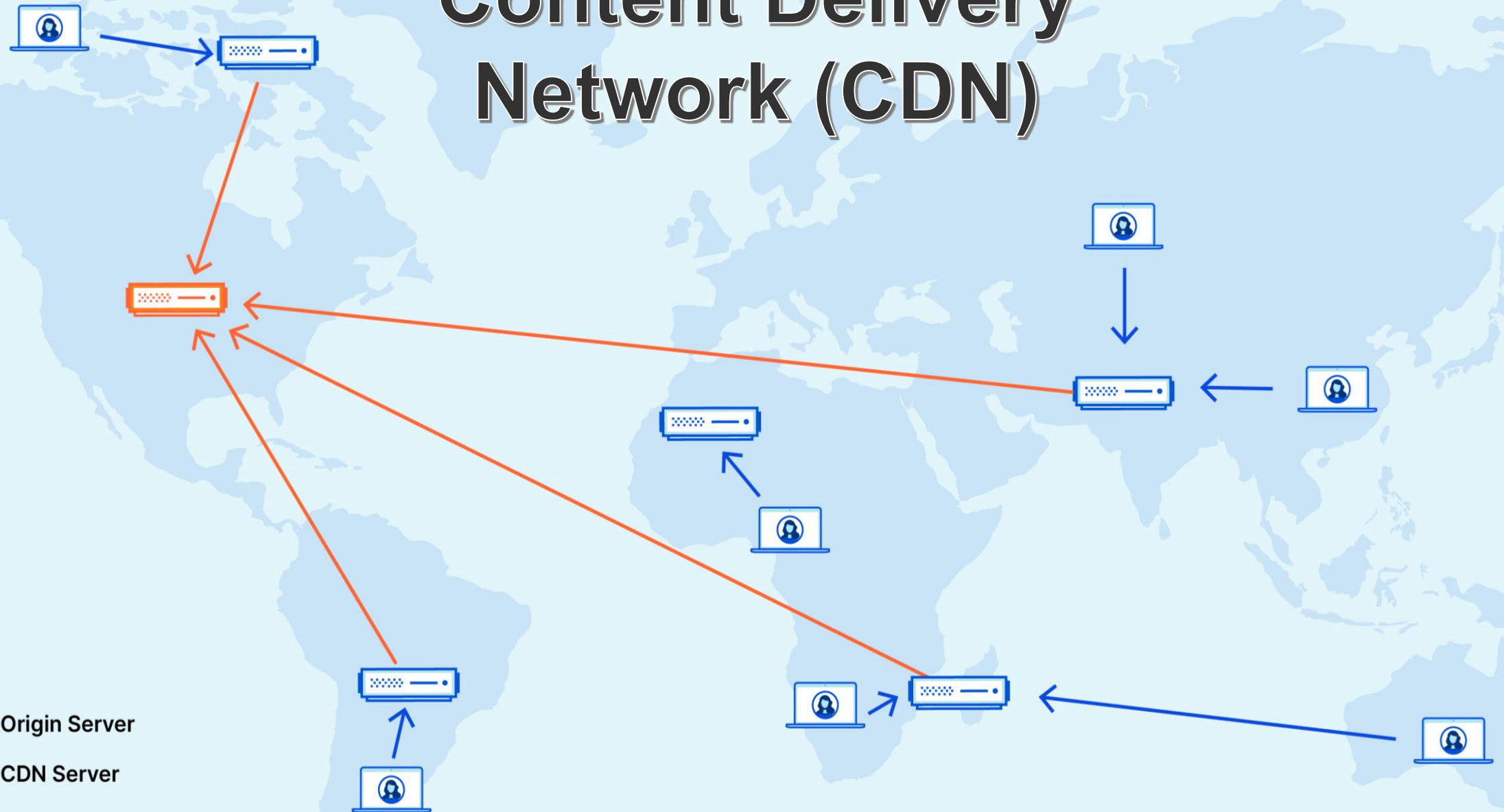


Image credit : Cloudflare (www.cloudflare.com)

Short Exercise (10 minutes)

- List out some of the CDN provider examples.
- Describe the benefits of CDN.

CDN

Watch a short video that explains CDN, and how it works.

What Is A CDN? How Does It Work? - ByteByteGo

- <https://www.youtube.com/watch?v=RI9np1LWzqw>

Some examples of CDN providers:

→ Amazon CloudFront, Google Cloud CDN, Cloudflare, Akamai

CDN – Common routing technologies

- DNS-based routing
 - Each PoP (Point-Of-Presence) has its own IP address
 - DNS returns the IP address of the PoP closest to requestor.
- Anycast
 - All PoPs share the same IP address
 - When a request comes into the Anycast network, the network sends the request to the PoP closest to the requestor.

CDN - Content caching

- Static content
 - PoP will cache the content from origin server.
 - PoP will also optimize the static content.
- Dynamic content
 - This is usually uncacheable content.
 - Bad performance due to expensive TLS handshake.
 - By terminating TLS connection at PoP, latency is reduced, which leads to performance improvement.

Benefits of CDNs

Allows organizations to offload content delivery to third-party vendors.

Key benefits of CDNs:

- **Performance**
 - Reduce page load time for users
 - Reduce bandwidth costs for origin server (which includes infrastructure and connectivity costs)
- **Security**
 - Reduce impact from DDoS attacks
- **Availability**
 - Increase content availability even with increased web traffic to ensure uninterrupted service.
 - Increase availability of origin server.

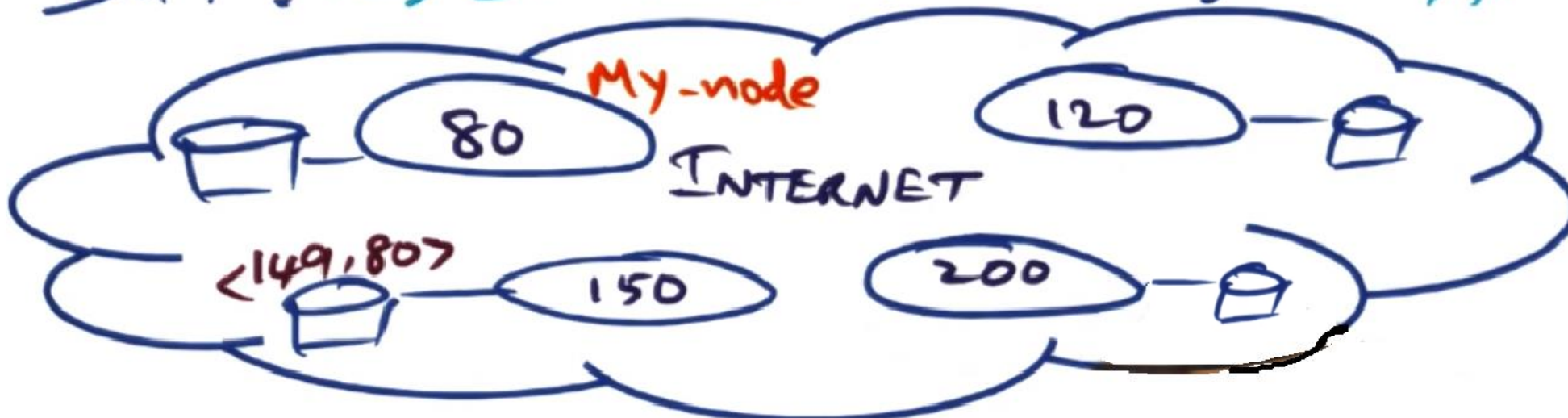
Distributed Hash Table (DHT)

$\langle \text{Key}, \text{value} \rangle$

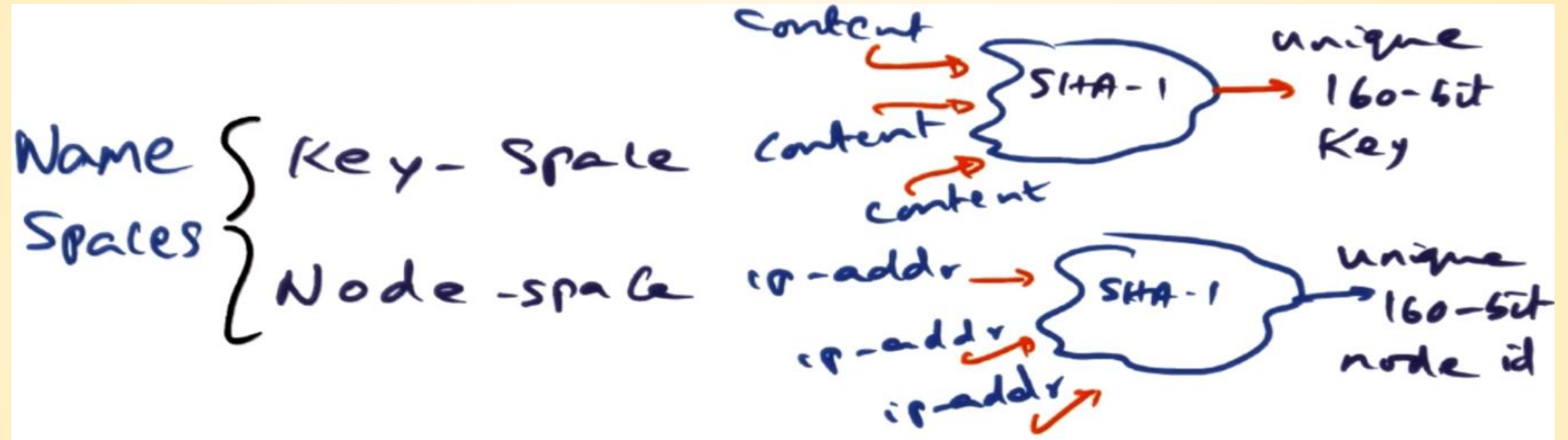
Content hash node-id where content stored

content-hash = 149 $\Rightarrow \langle 149, 80 \rangle \Rightarrow$ Where to store?

DHT: Key \approx node-id for storing $\langle \text{Key}, \text{value} \rangle$



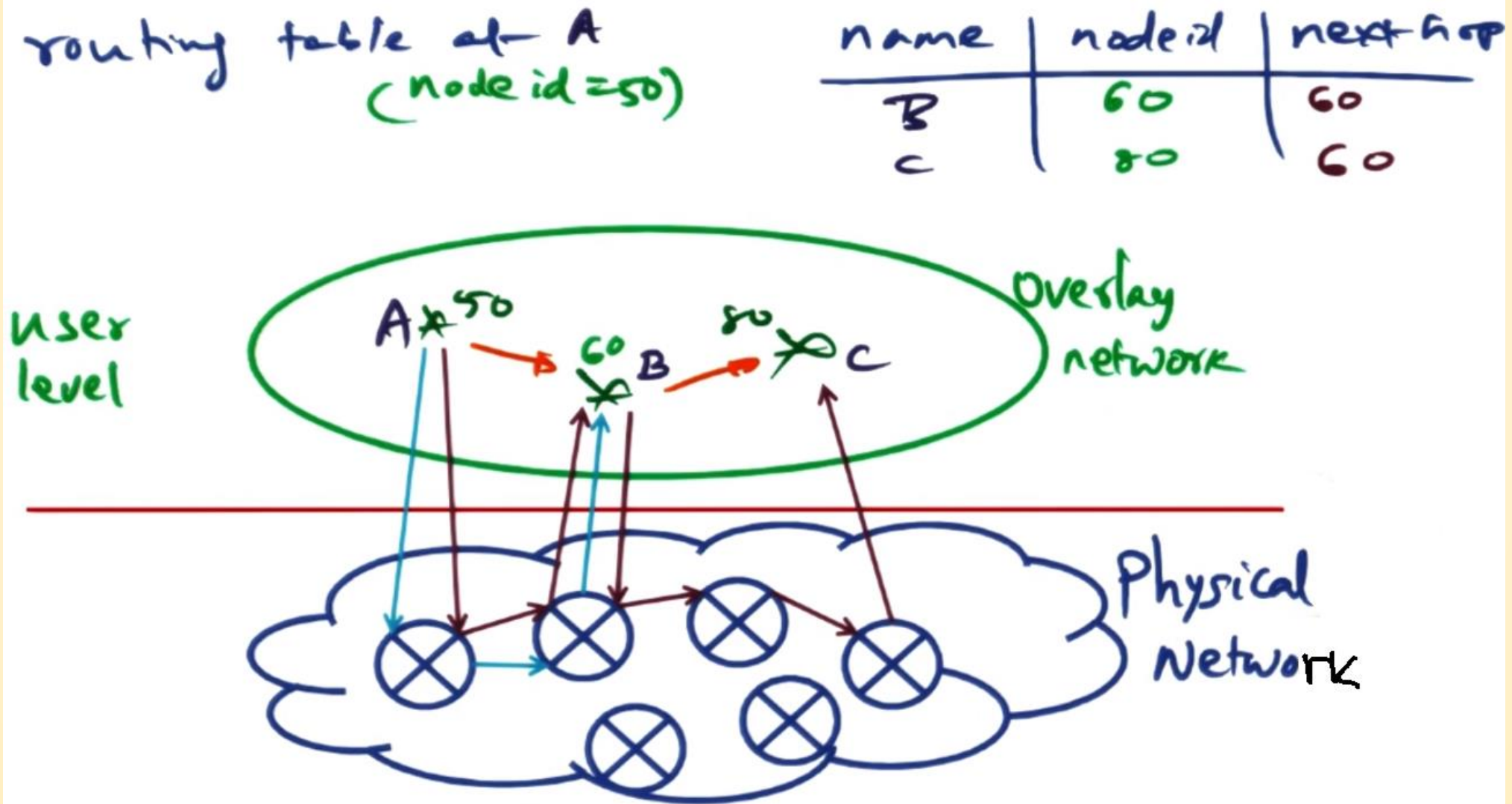
DHT Details



Objective :

$\langle \text{Key} \rangle \rightarrow \text{nodeid} \langle N \rangle$
such that $\langle \text{Key} \rangle \underline{=} \langle N \rangle$

CDN - An Overlay Network



Overlay Network in General

Examples of overlay

OS Level

— IP network is an overlay
on LAN

IP Addr	Mac Addr

App Level

— CDN is an overlay
on TCP/IP

Node ID	IP Addr

DHT and CDN

Placement

— put < key, value >

Content hash node-id where content stored

Retrieval of value given key

— get < key >

— get back < value >

Traditional Approach (Greedy)

place $\langle \text{key}, \text{value} \rangle$ in
node N , where $N \approx \text{key}$

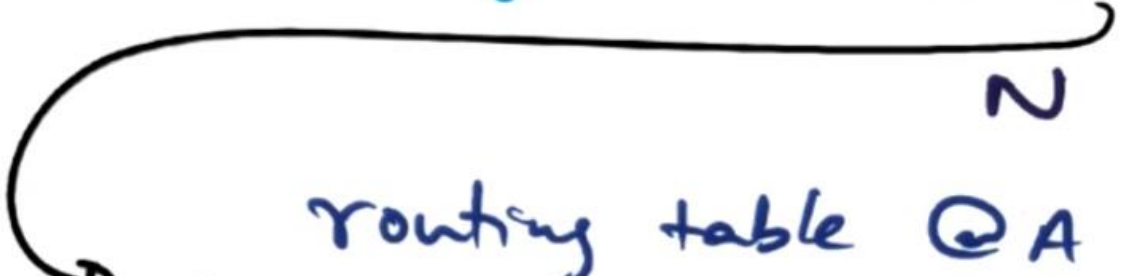
retrieve :

given key K

goto node
(closest to key K)

routing table @ A

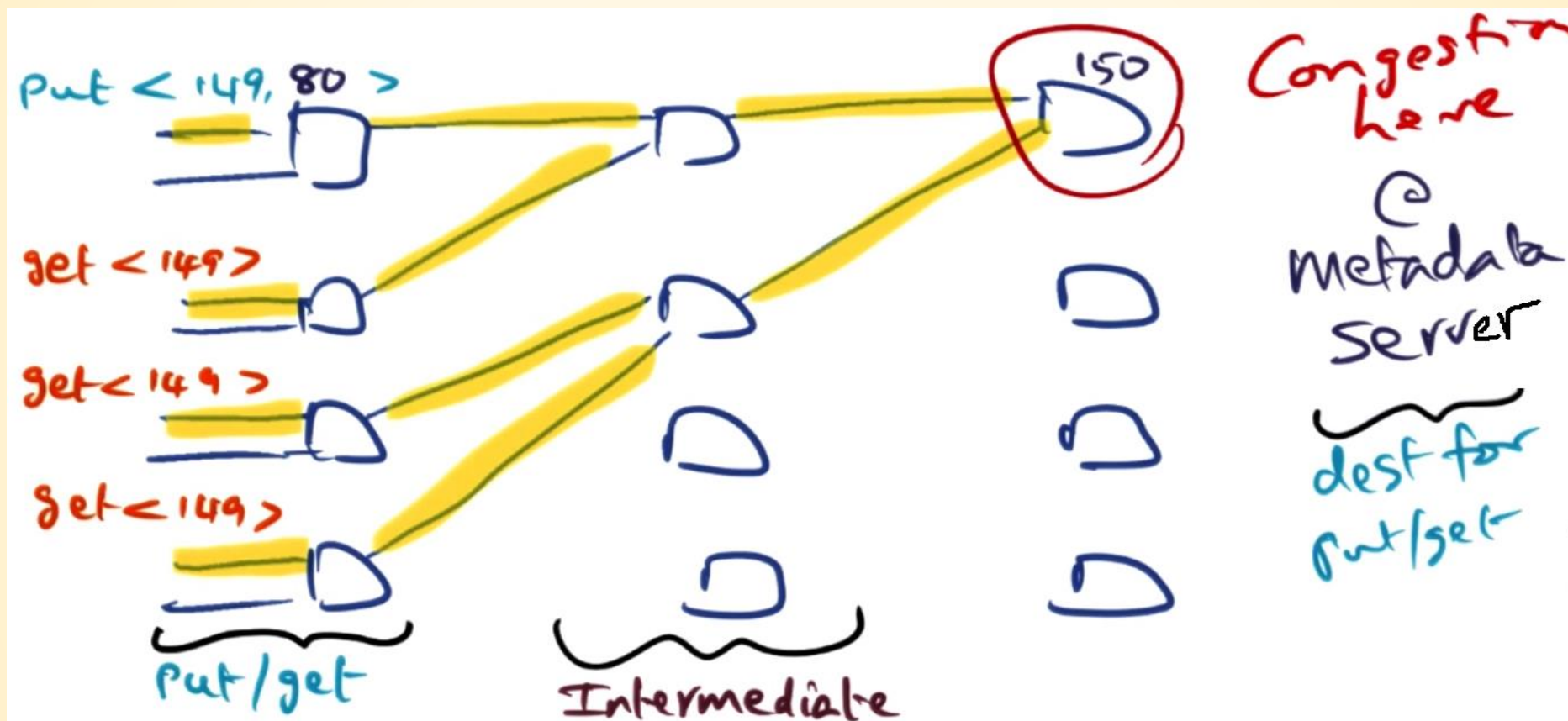
routing table @ B



Known Peers (N)	IP-addr
60	...
79	...

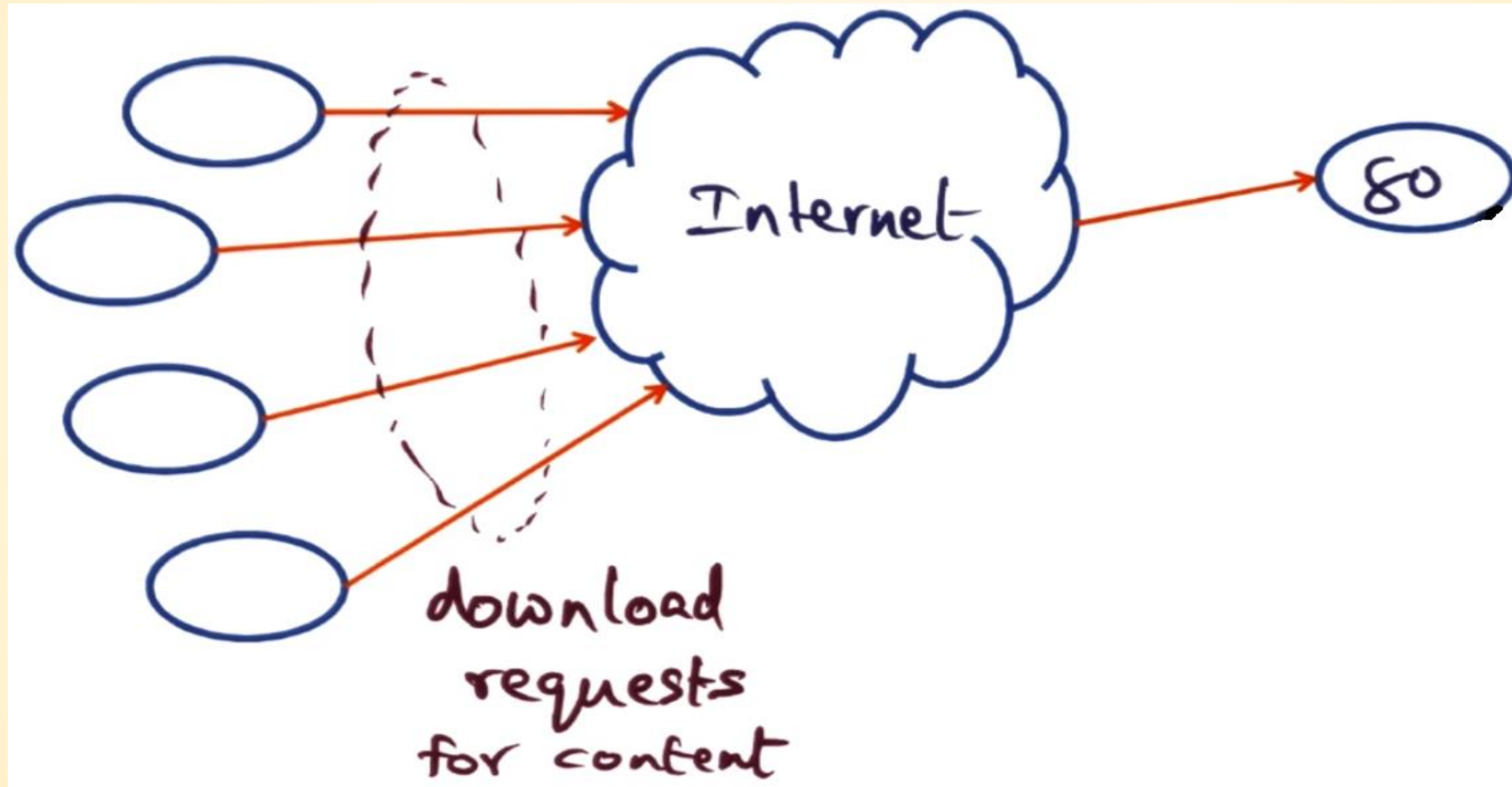
Known Peers (N)	IP-addr
60	...
109	...

Problem 1: Traditional Approach (Greedy)



Leads to Metadata Server Overload

Problem 2: Traditional Approach (Greedy)



Leads to Origin Server Overload

Map-Reduce

Map-Reduce

Watch a short video that explains Map-Reduce function in Big Data analysis.

MapReduce – Computerphile

- <https://www.youtube.com/watch?v=cvhKoniK5Uo>

What is Map-Reduce?

Input & output to each of map & reduce

- $\langle \text{Key, Value} \rangle$ Pairs

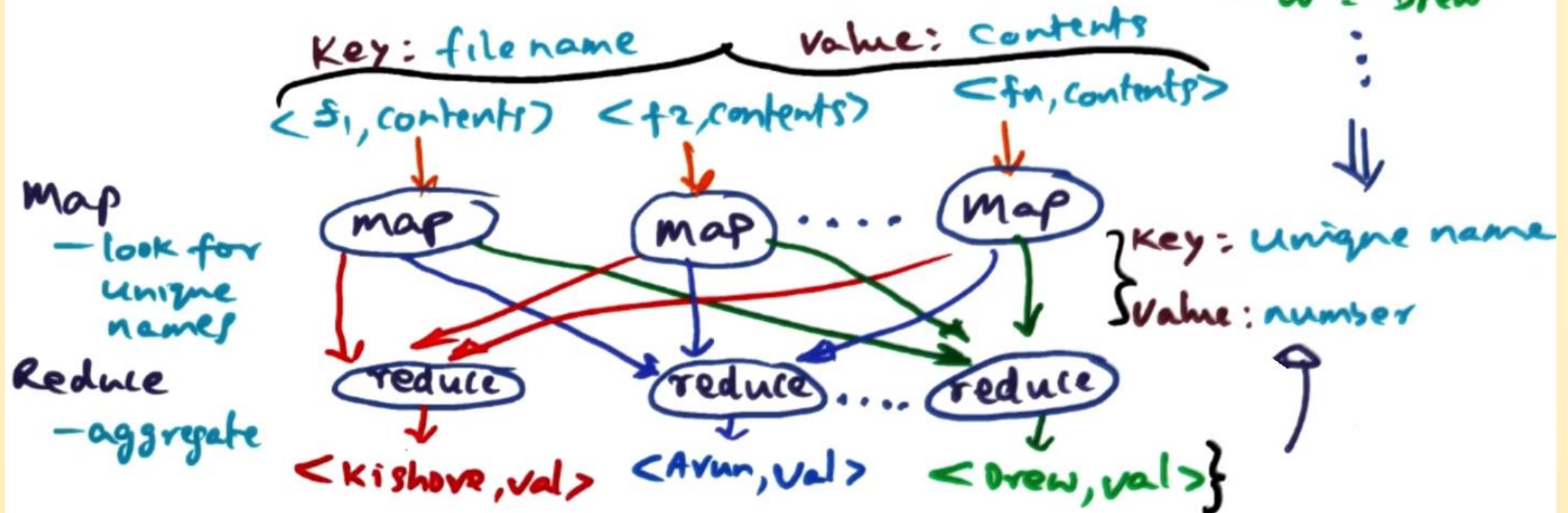
Example

- Emit # of occurrence of names in docs

- W = "Kishore"

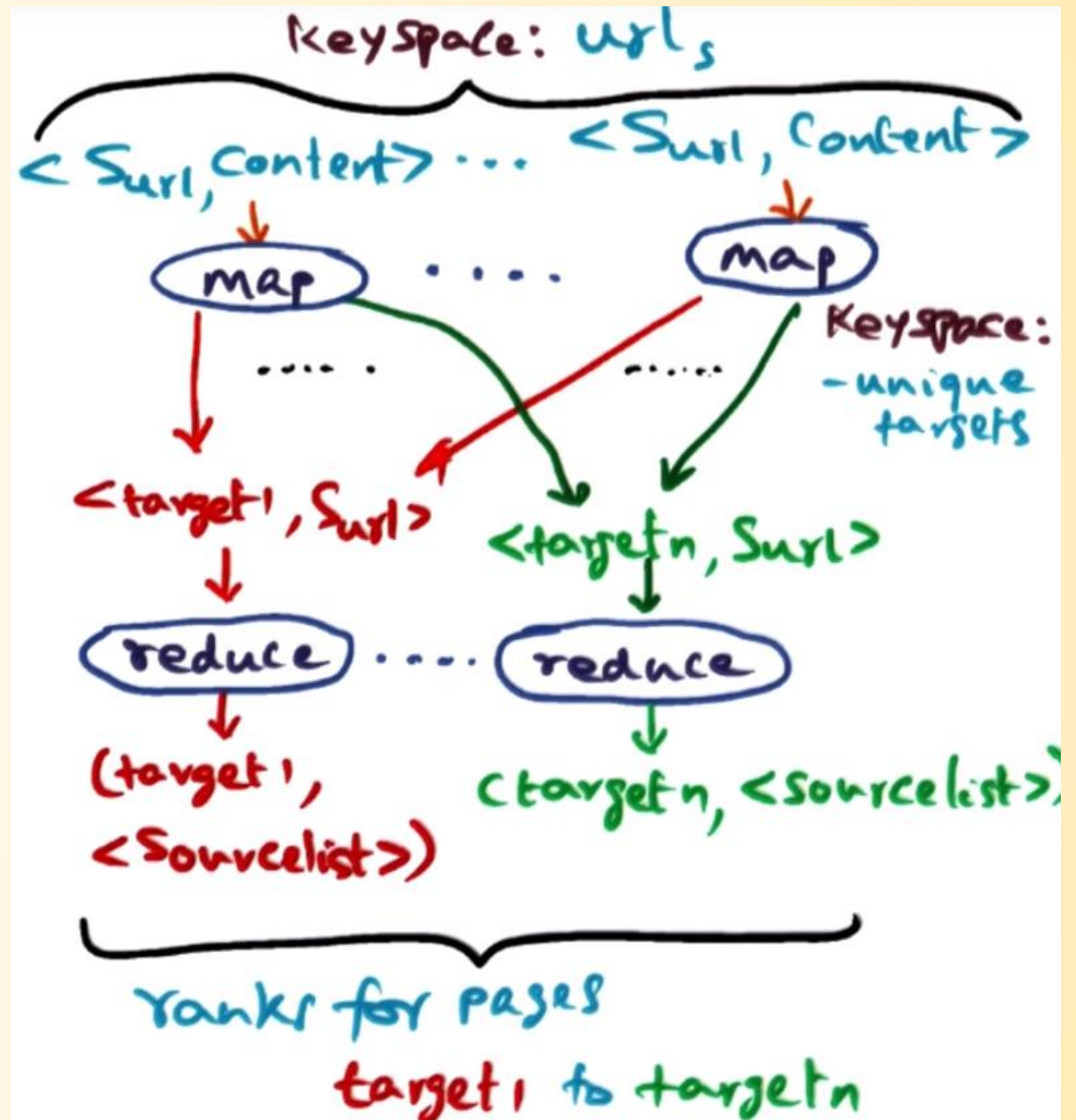
- W = "Arun"

- W = "Drew"

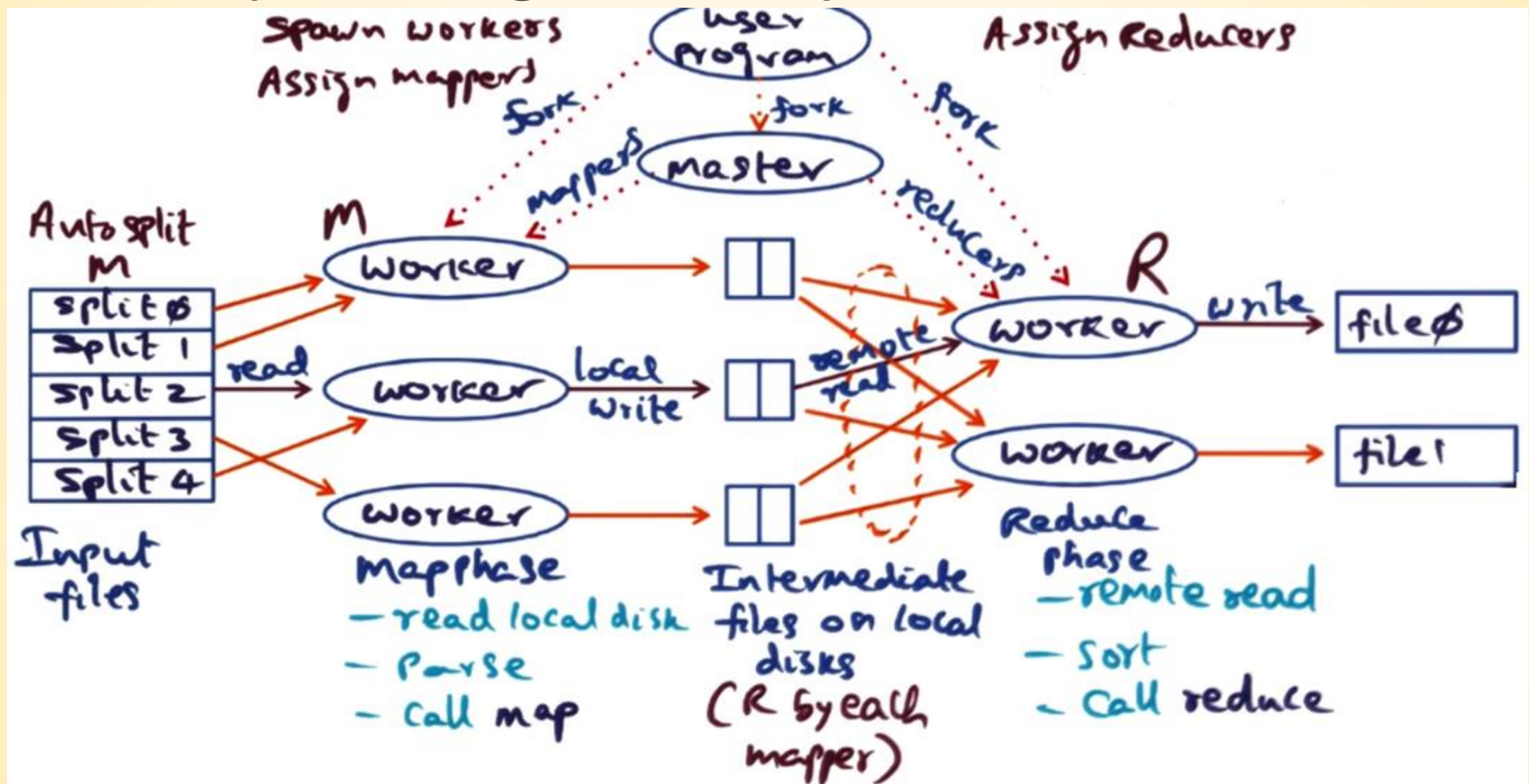


Why Map-Reduce?

- Several steps in giant scale service expressible in “map-reduce”.
- Map-Reduce functions are embarrassingly parallel and independent process
- Big data set.



Heavy Lifting Done By Runtime



Issues to be Dealt by Runtime

Master data structures

- Location of files created by completed mappers
- Scoreboard of mapper & reducer assignments
- Fault tolerance
 - start new instances if no timely response
 - completion message from redundant stragglers
- Locality management – e.g. making sure the working set of computations fit in the closest level of memory hierarchy of a process.
- Task granularity - for good load balance of computational resources
- Backup tasks