Overlay (and P2P) Networks

Part II

- Structured Overlay Networks
  - Content Delivery Networks
    - Akamai
    - Coral
  - Amazon Dynamo

Samu Varjonen

Ashwin Rao

Content Delivery Networks

Limitations of Web Proxies
(Caching)

- Inability of cache all objects/content
  - Dynamic Data
  - Encrypted data
- Server Side Analytics
  - Hit Metering, User Demographics, etc.
- Scalability
  - Inability to support **flash crowds**
- ...

Content Delivery Networks

- Role
  - Redirect content requests to an 'optimal site'
  - Cache and Serve content from 'optimal site'
  - Export logs and other information to origin servers
- Redirection mechanism
  - DNS redirection
  - URL rewriting

Critical Issues in Deploying CDNs

- Servers Placement
  - Where to place the servers?
  - How many in each location?
- Content Selection
  - Which content to distribute in CDNs?
- Content Replication
  - Proactive push from origin server
  - Cooperative vs Uncooperative Pulls
- Pricing

George Pallis et al. **"Insight and perspectives for content delivery networks."** In Communicatoins of ACM 49, 1 (January 2006),

Server Placement Problem

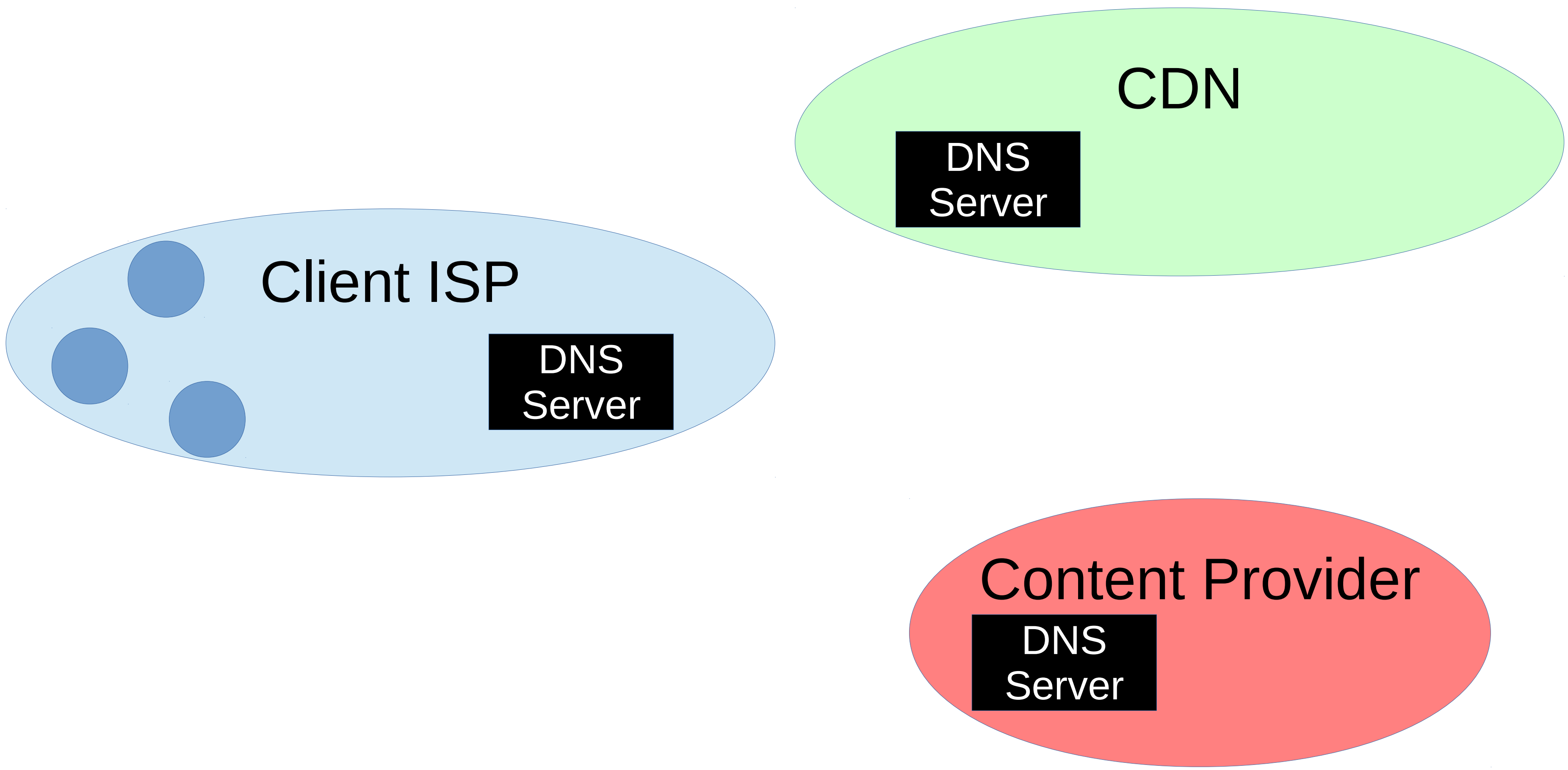
Given N possible locations at edge of the Internet, we are able to place K (K<N) surrogate servers, how to place them to minimize the total cost?

- Minimum K-median problem
  - Given N points we need to select K centers
  - Assign each input point j to a center 'closest' to it
  - Minimize the sum of distances between each j and its center
    - NP-Hard

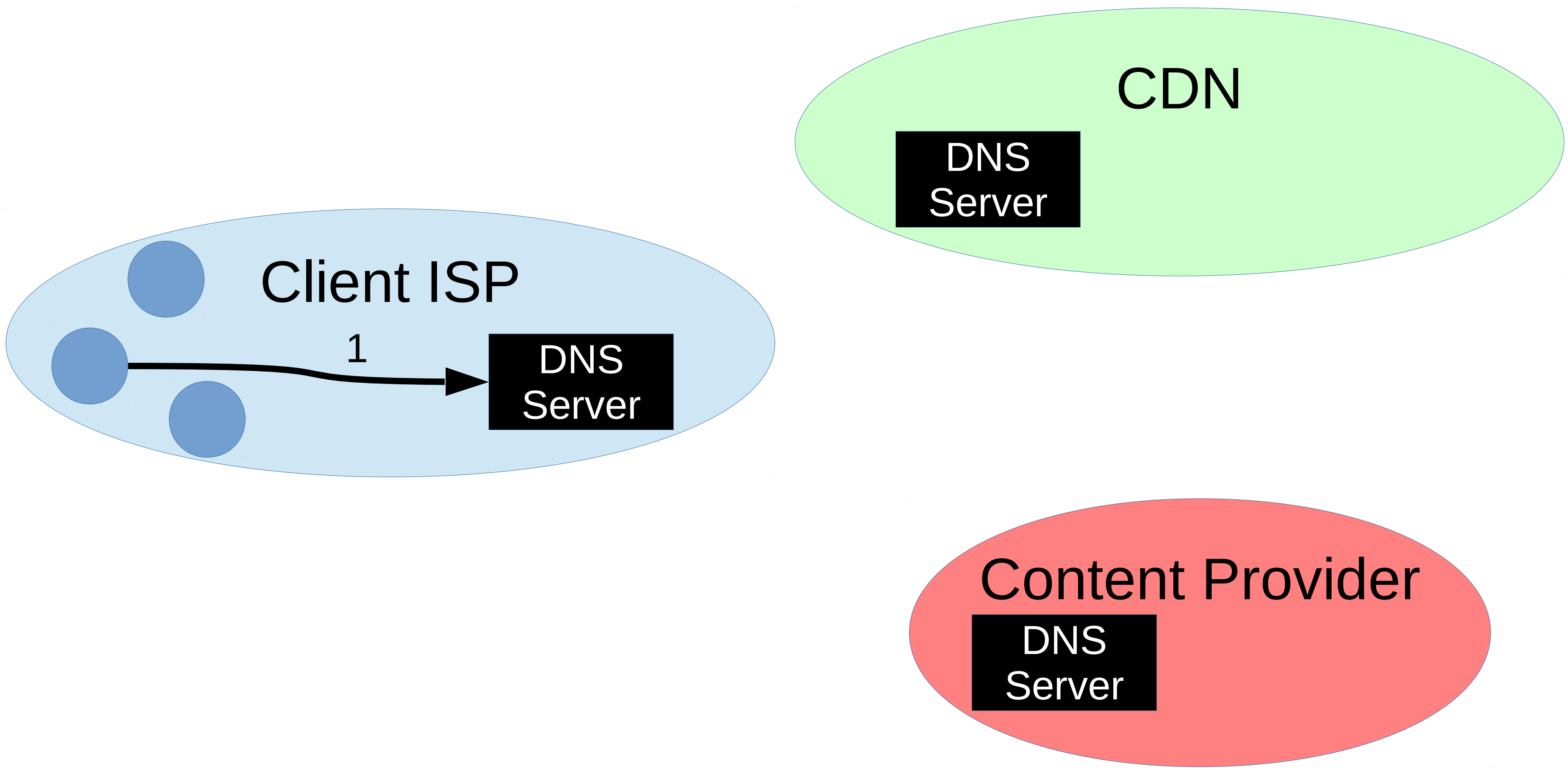
Redirection Techniques

- Routing Strategy
  - Anycast
  - Load Balancing
- Application specific selection
  - HTTP redirection
- Naming based redirection
  - DNS

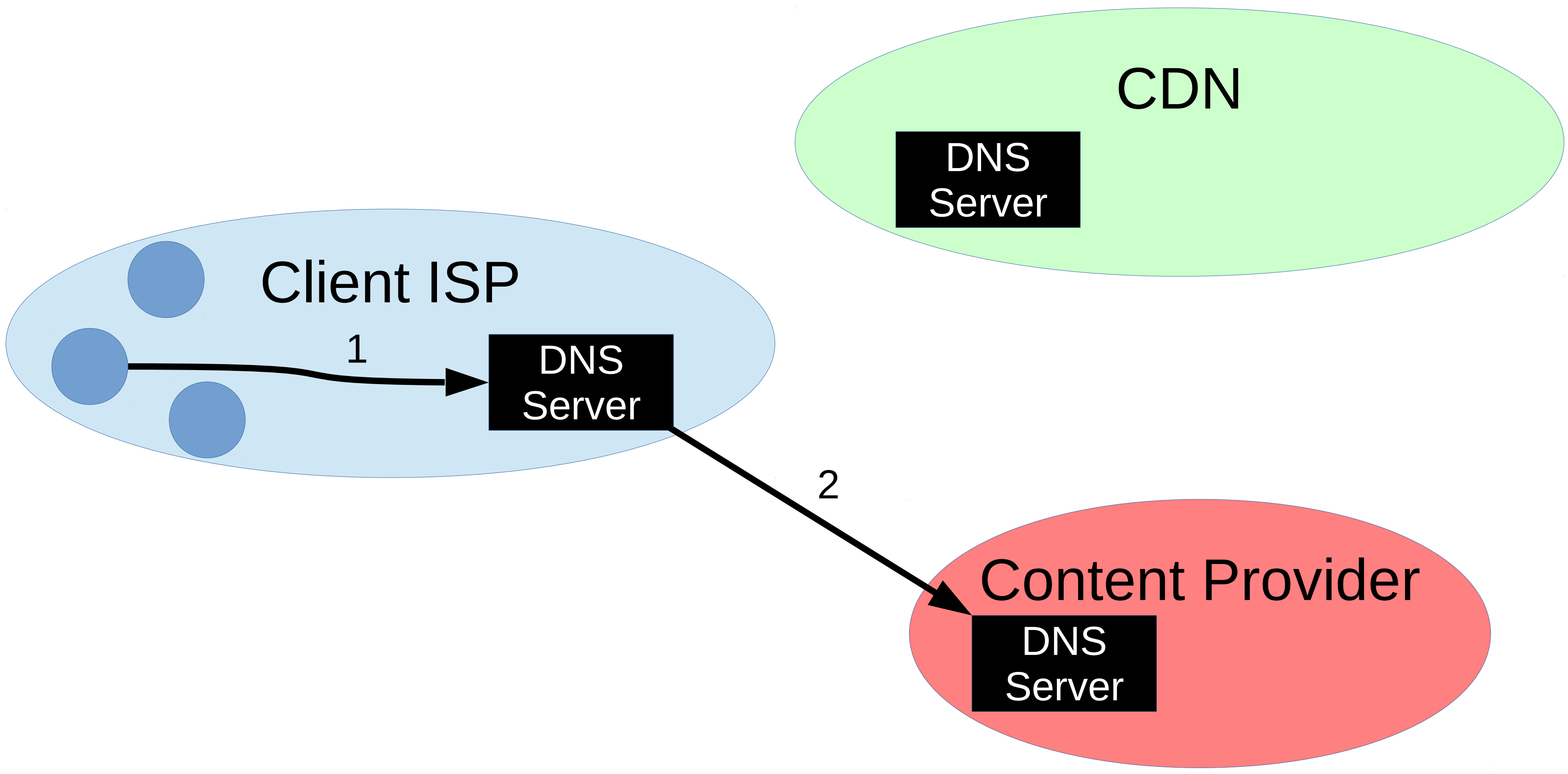
DNS Based Redirection



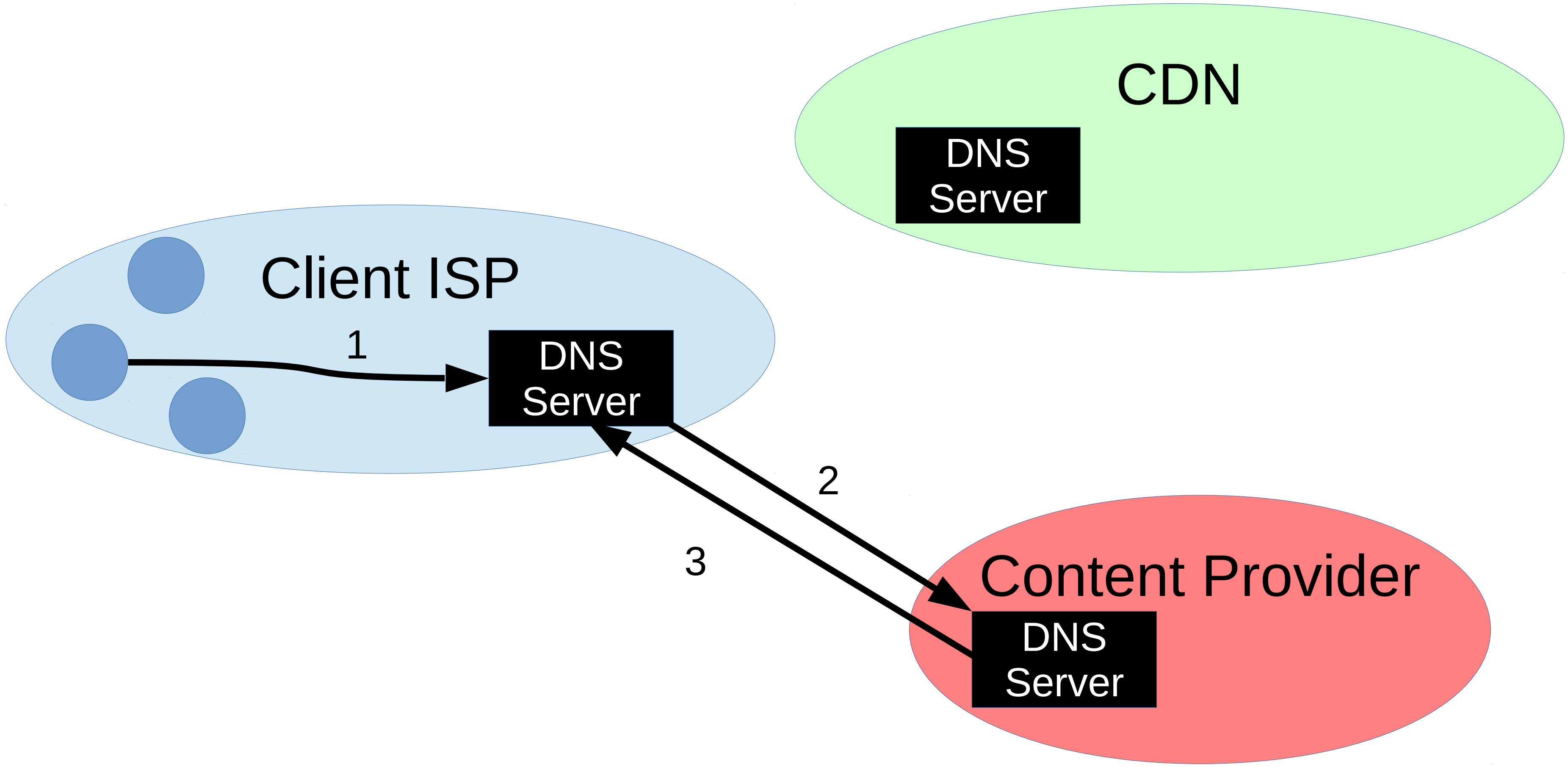
DNS Based Redirection



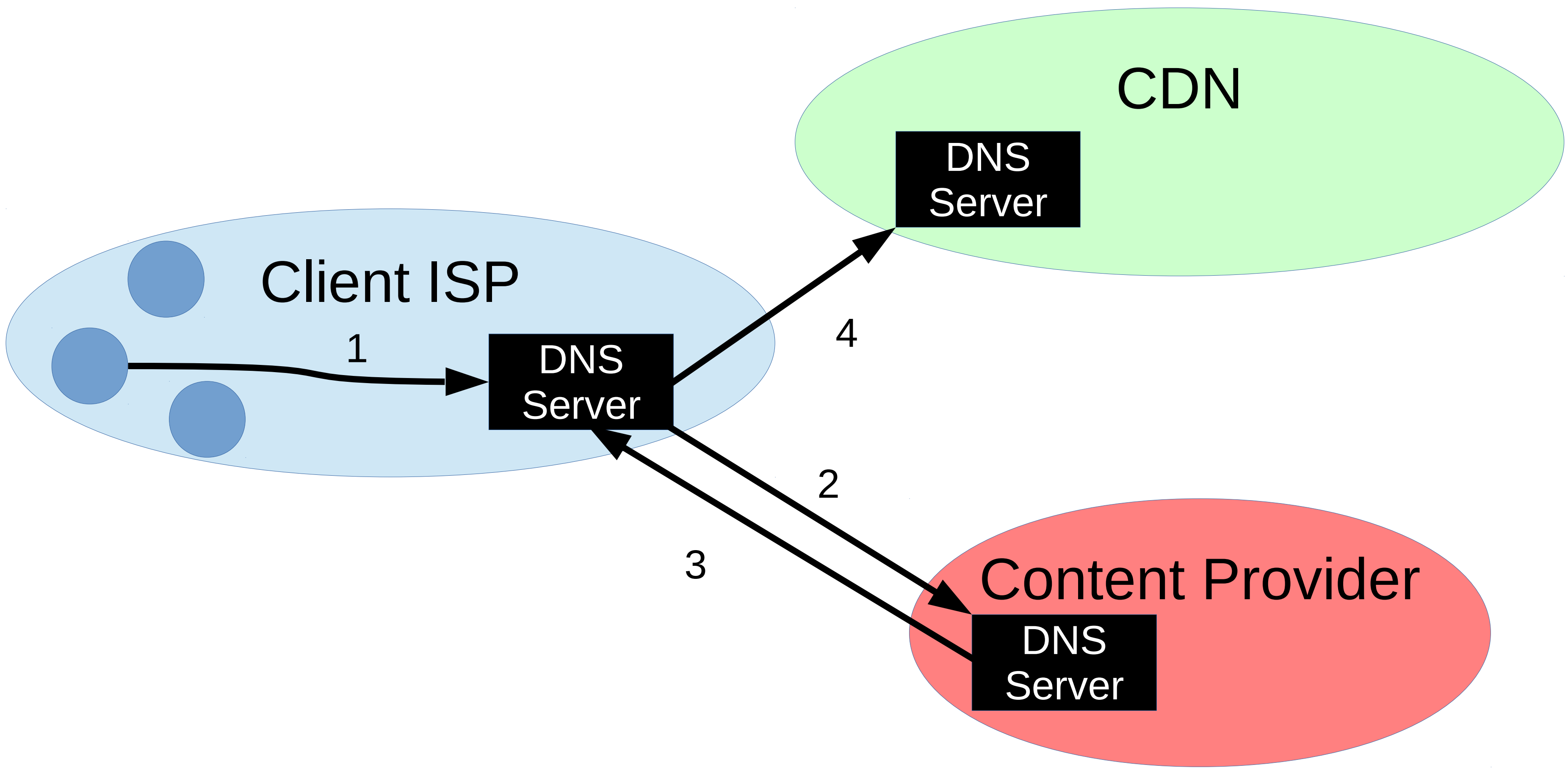
DNS Based Redirection



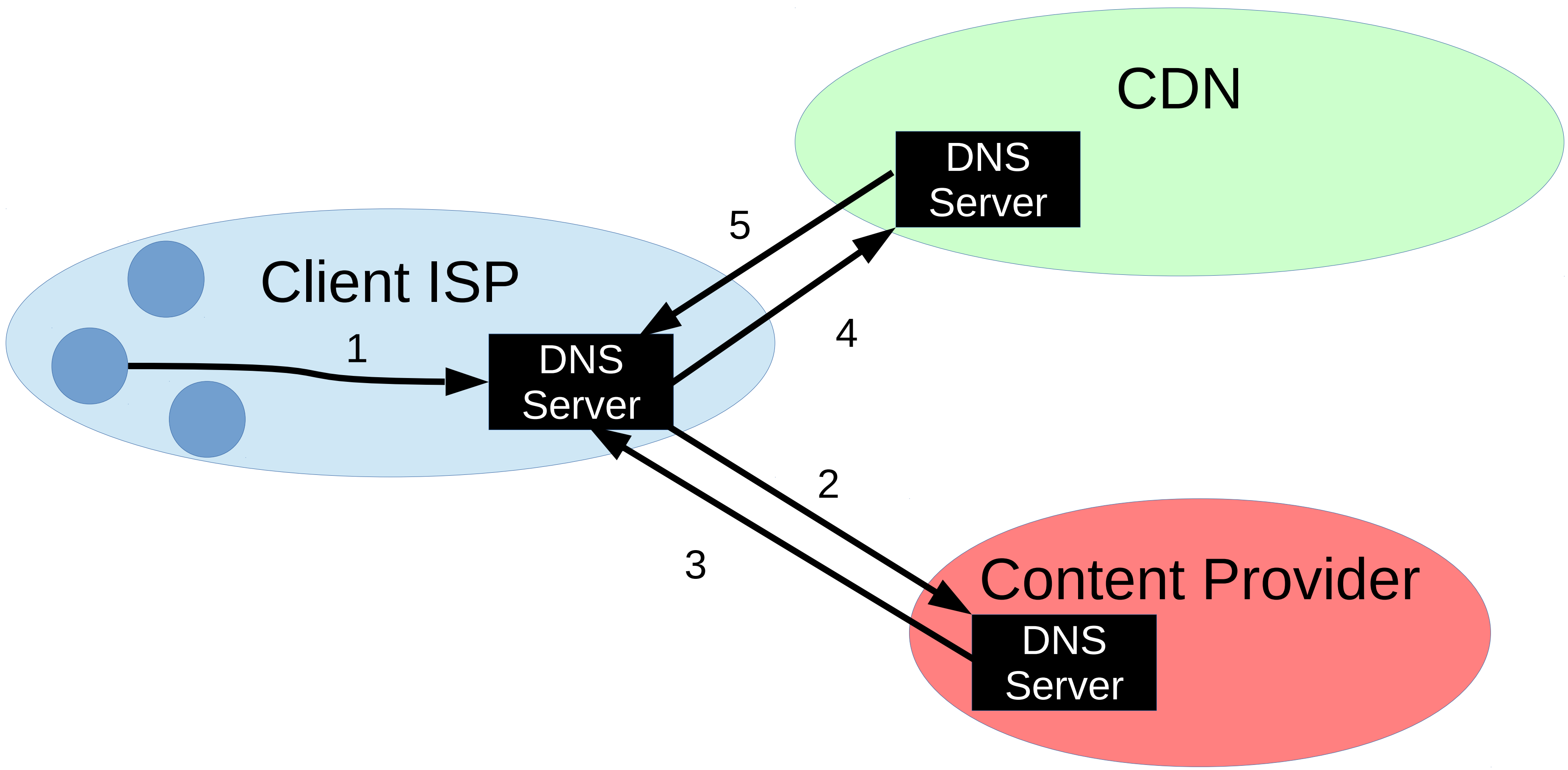
DNS Based Redirection



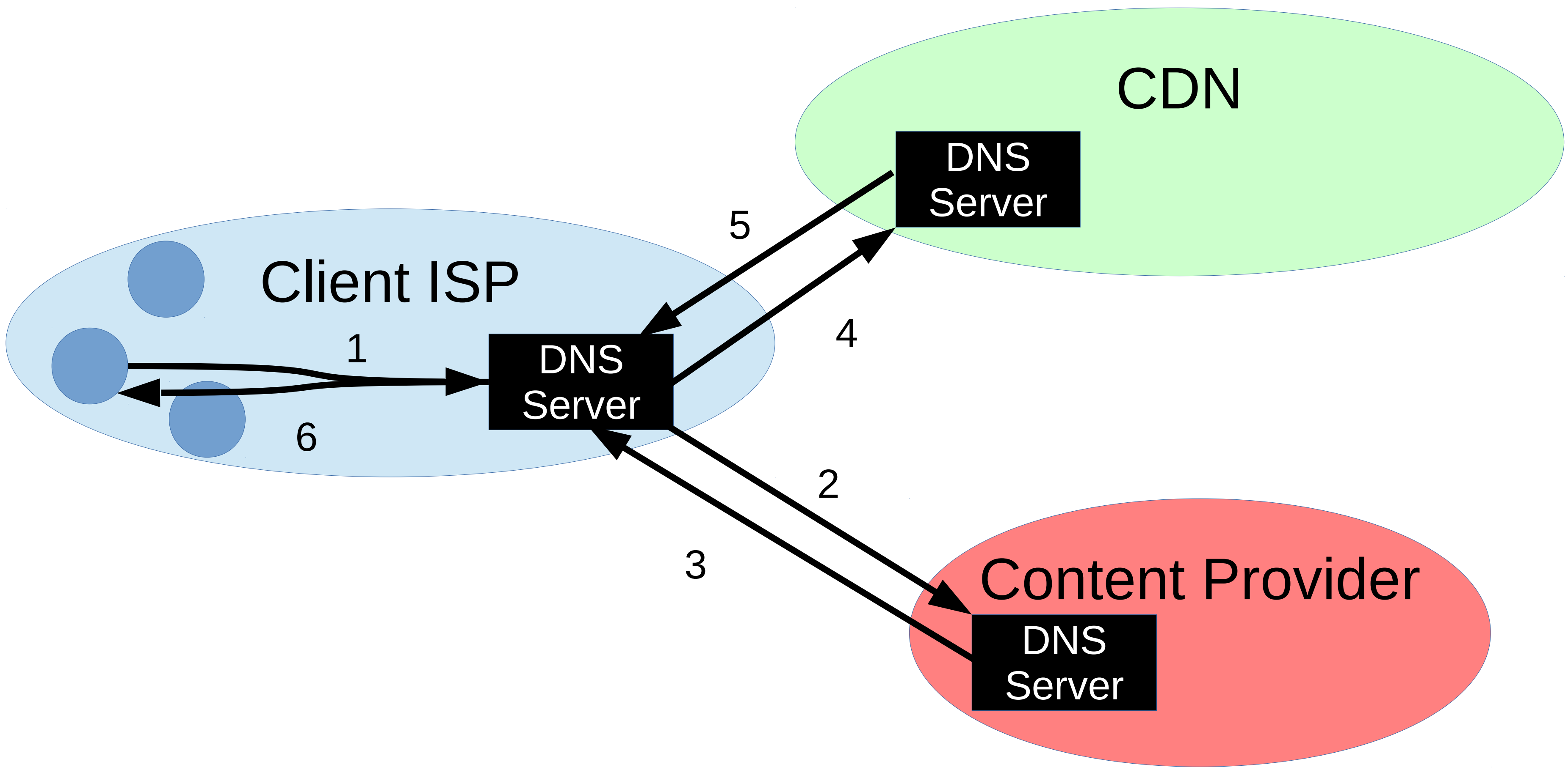
DNS Based Redirection



DNS Based Redirection



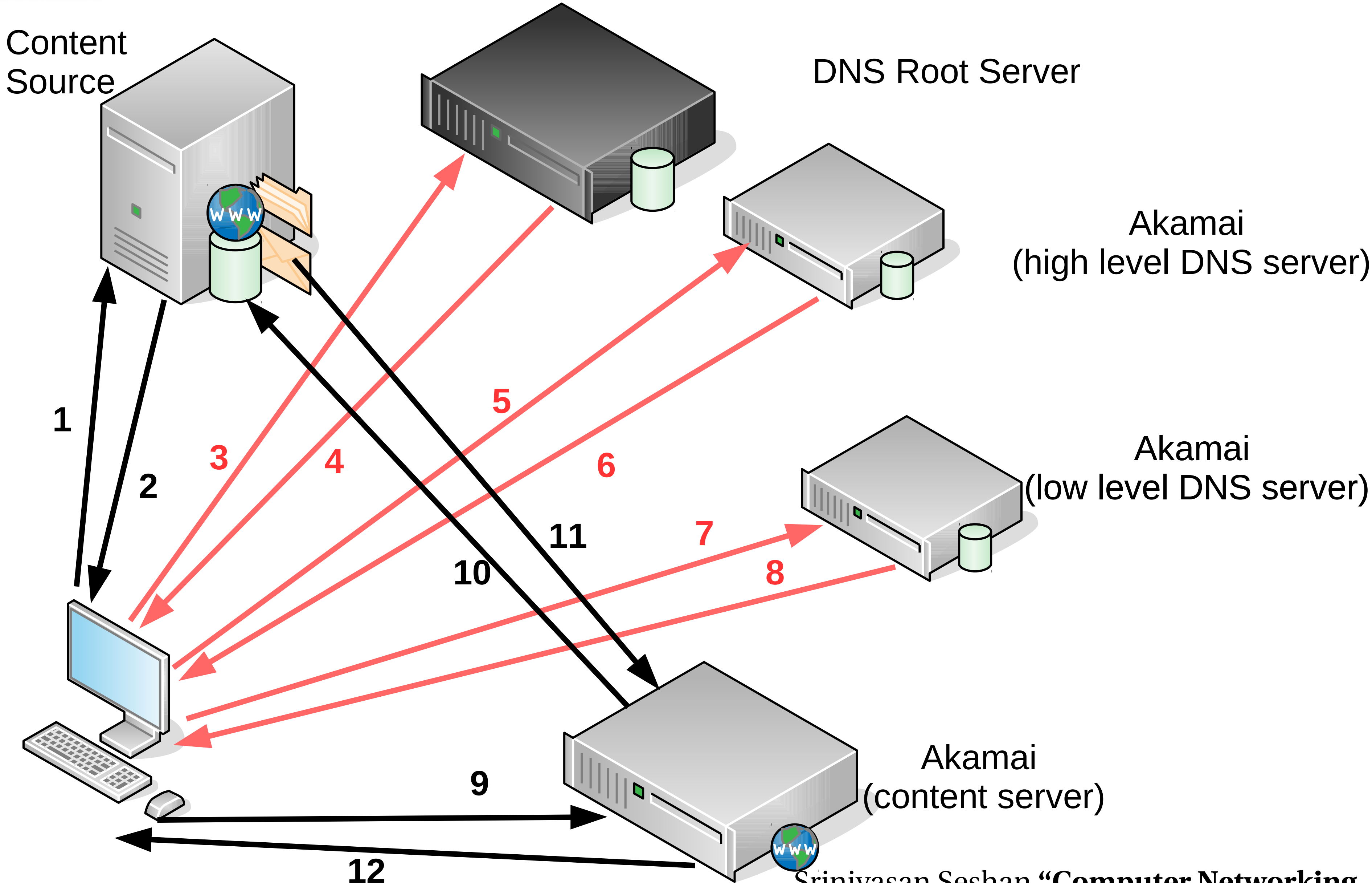
DNS Based Redirection



Akamai CDN
(overview)

- Client requests content from Original Server
  - URLs for content in CDN modified in the original response
- Client resolves <content>.<akamai host> name
- Server from the region (best server) chosen
- Client fetches content from akamai server

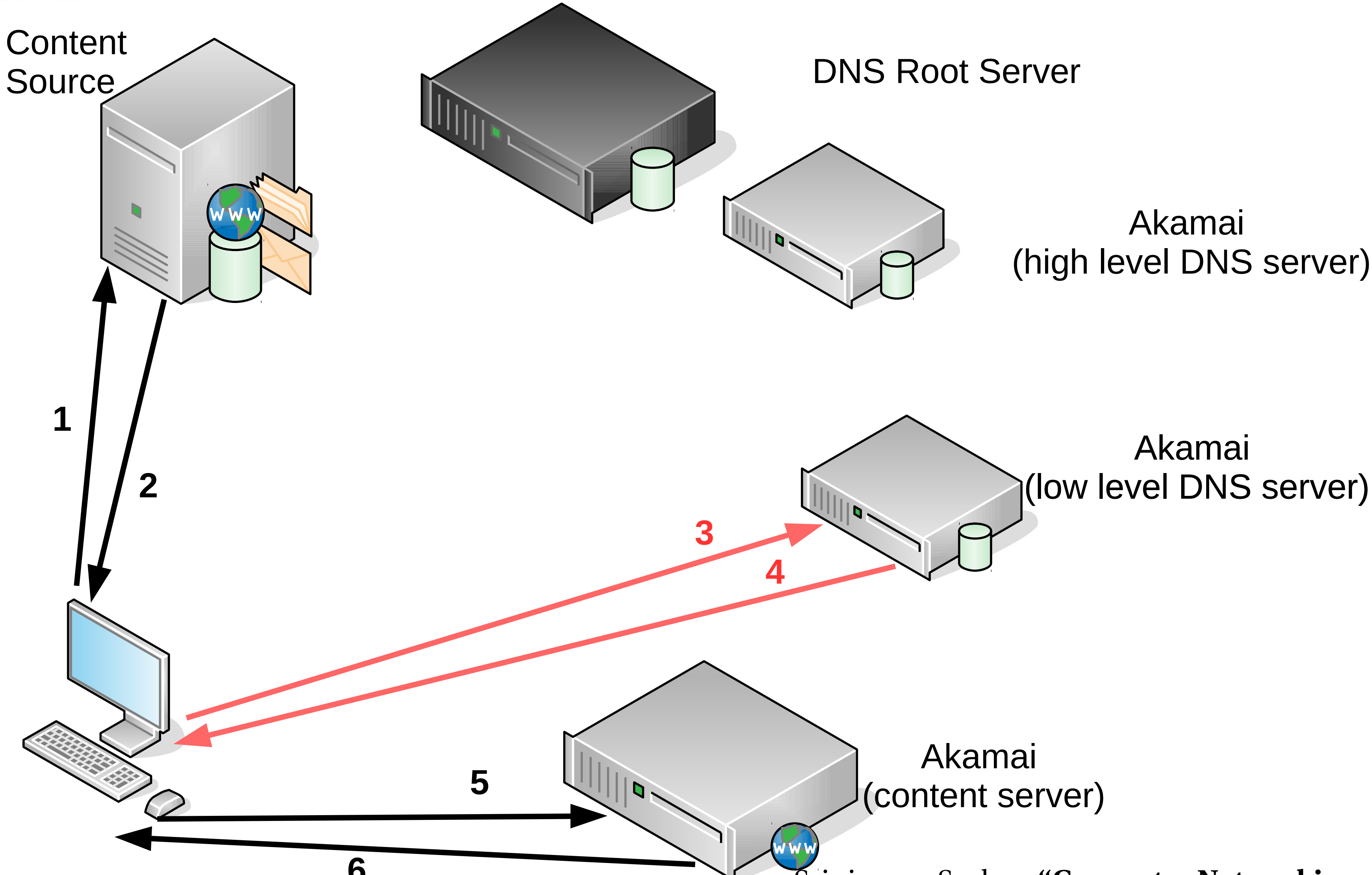
Akamai
(initial request)



Srinivasan Seshan **"Computer Networking Caching, CDN, Consistent Hashing, P2P"**

Akamai

(subsequent request)



Srinivasan Seshan **“Computer Networking Caching, CDN, Consistent Hashing, P2P”**

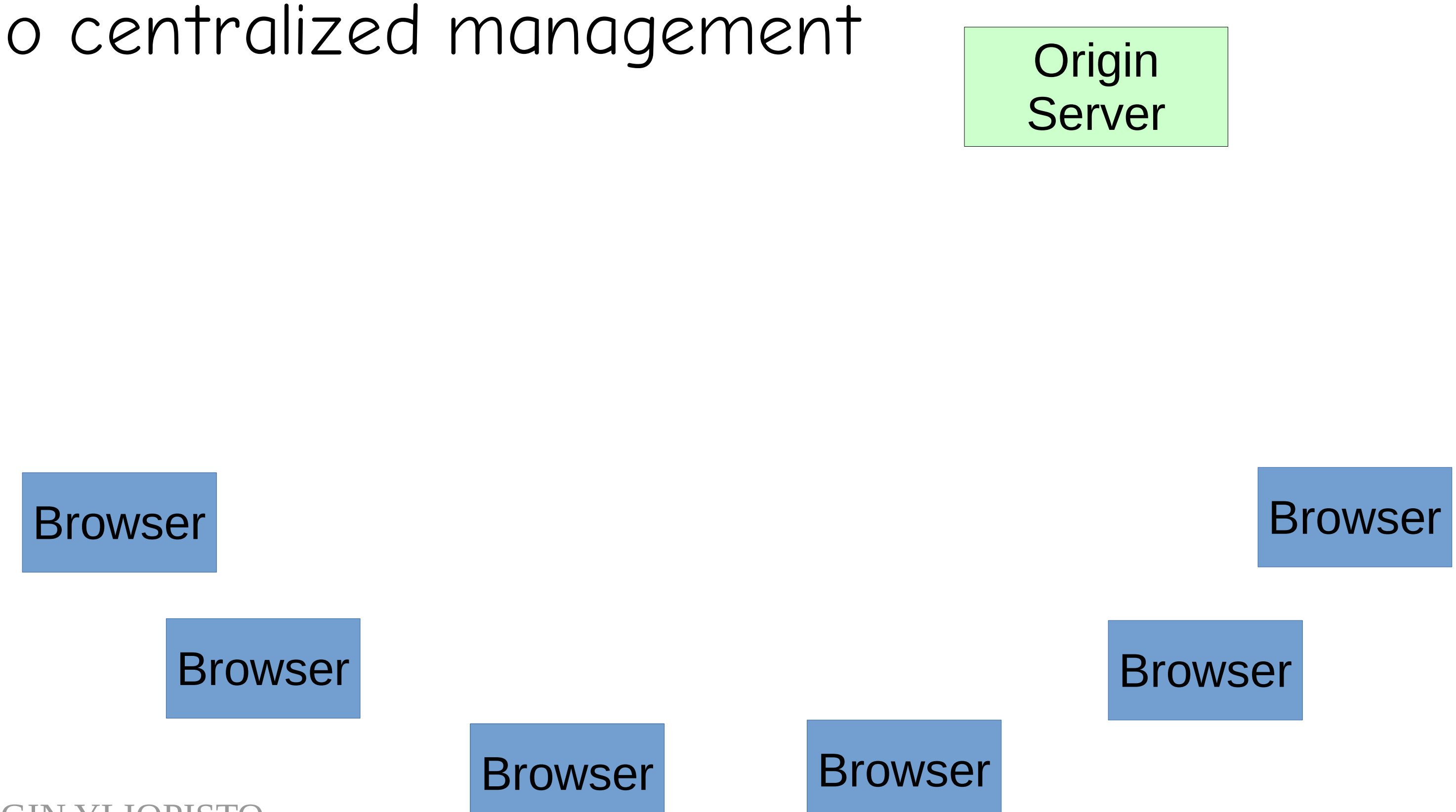
Democratizing Content Publication with Coral (Coral CDN)

Coral Objectives

- Pool resources to dissipate Flash Crowds
- Work with unmodified clients
- Fetch content only once from Origin
- No centralized management

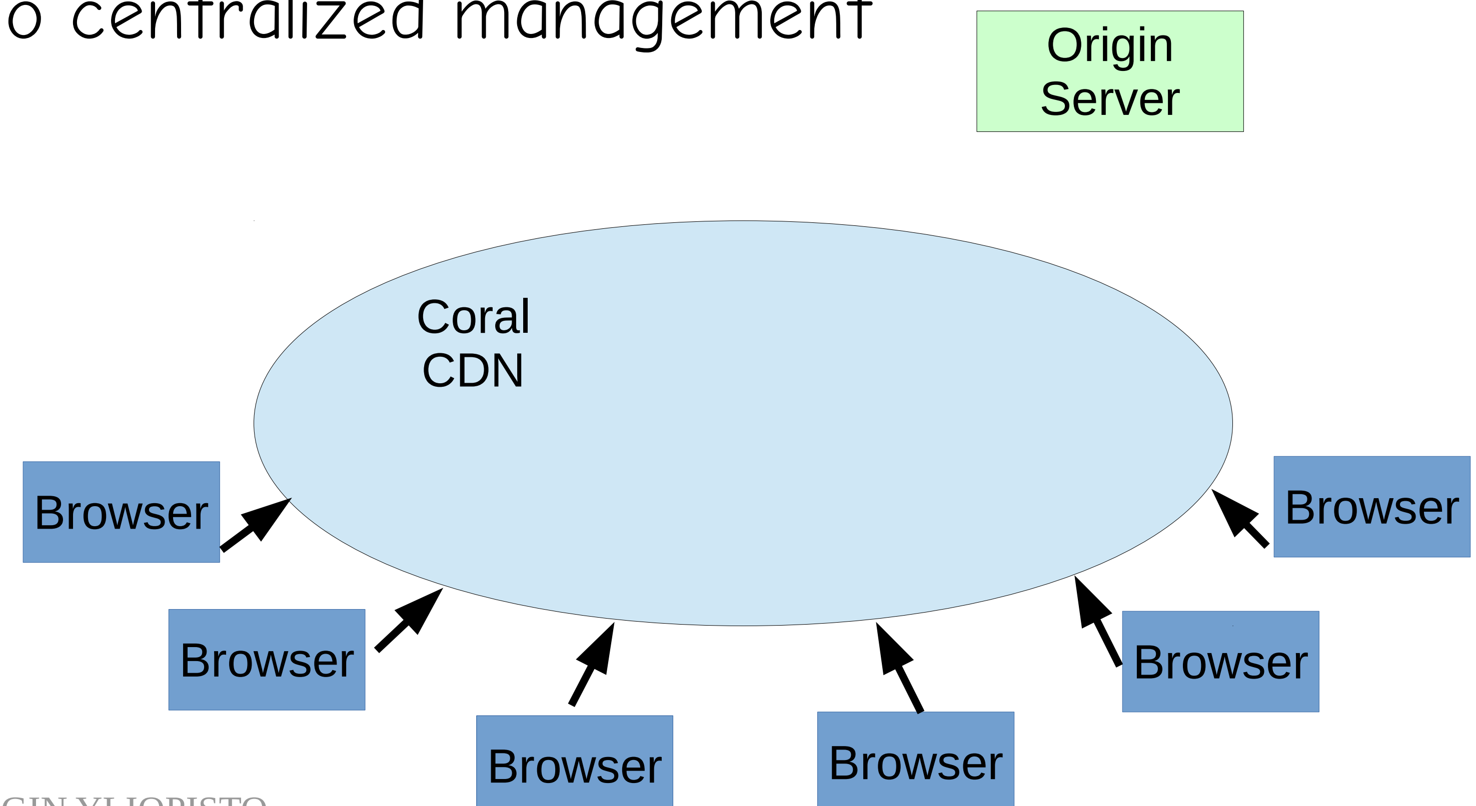
Coral Objectives

- Pool resources to dissipate Flash Crowds
- Work with unmodified clients
- Fetch content only once from Origin
- No centralized management



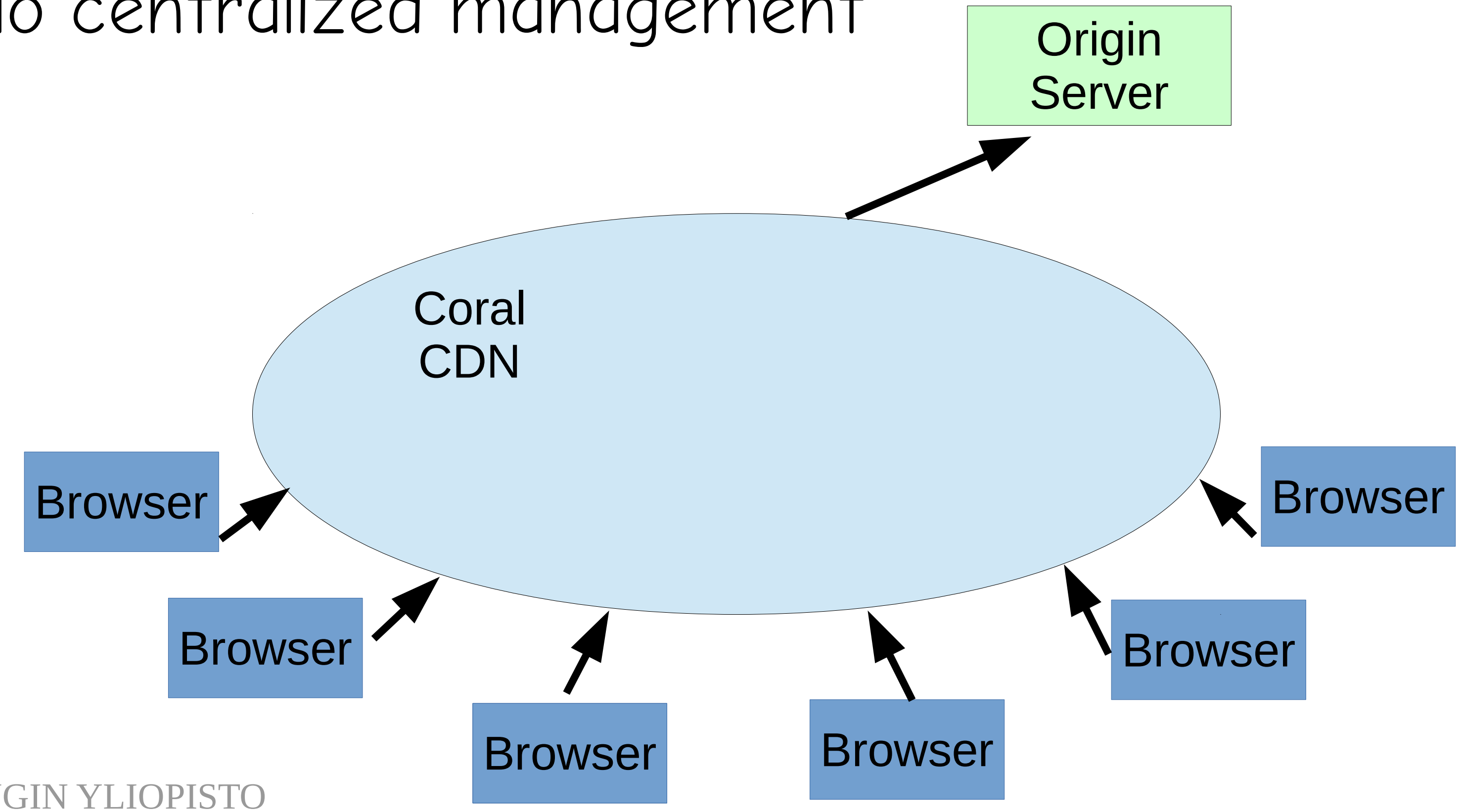
Coral Objectives

- Pool resources to dissipate Flash Crowds
- Work with unmodified clients
- Fetch content only once from Origin
- No centralized management



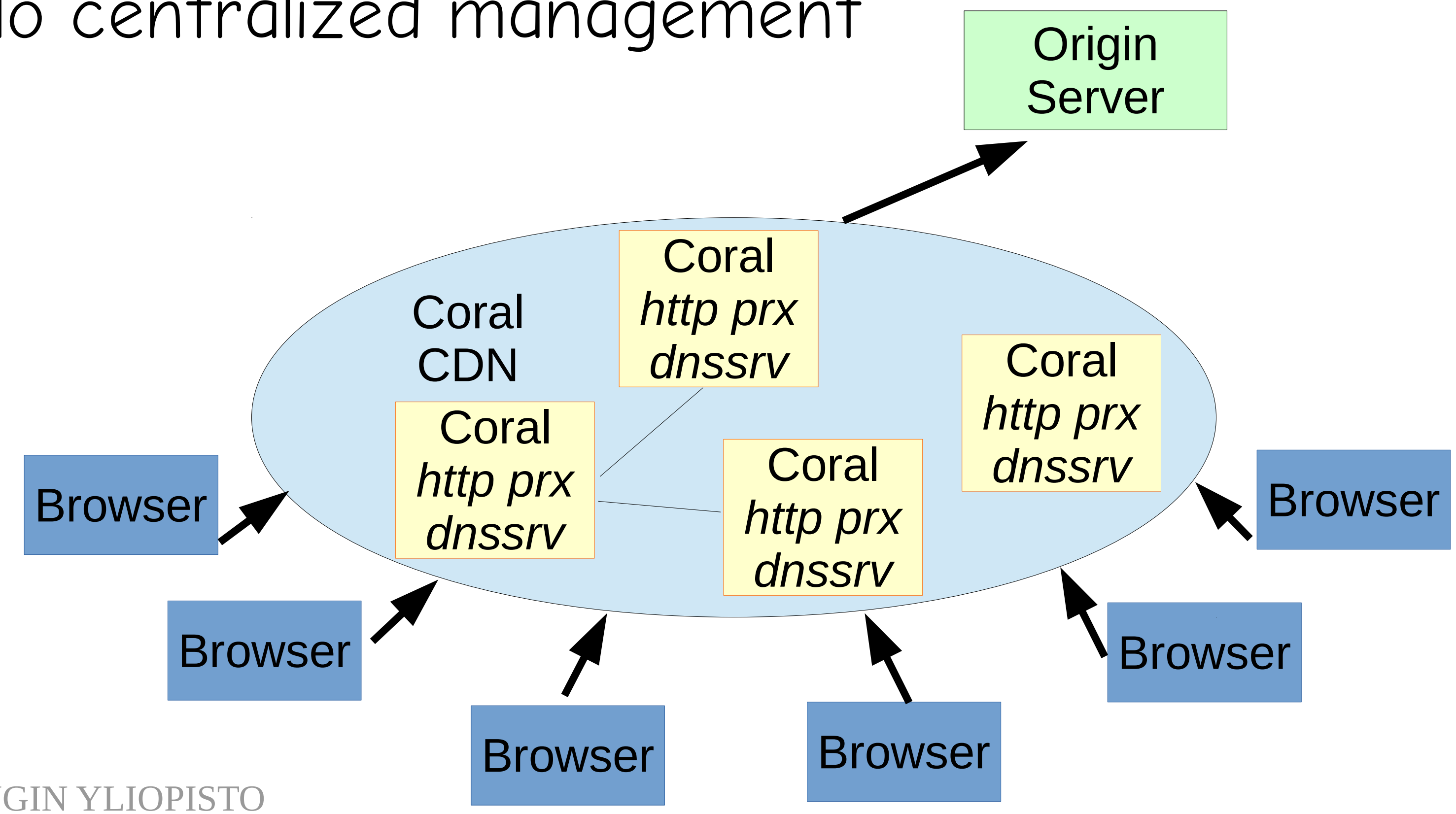
Coral Objectives

- Pool resources to dissipate Flash Crowds
- Work with unmodified clients
- Fetch content only once from Origin
- No centralized management



Coral Objectives

- Pool resources to dissipate Flash Crowds
- Work with unmodified clients
- Fetch content only once from Origin
- No centralized management



Using Coral

Using Coral

- Origin Server rewrites URLs
  - abc.com → abc.com.coralhost:coralport
  - Redirect clients to Coral server

Using Coral

- Origin Server rewrites URLs
  - abc.com → abc.com.coralhost:coralport
  - Redirect clients to Coral server
- Coral CDN Components
  - DNS server
    - Given address of resolver used by the client, return the address of proxy near the client

Using Coral

- Origin Server rewrites URLs
  - abc.com → abc.com.coralhost:coralport
  - Redirect clients to Coral server
- Coral CDN Components
  - DNS server
    - Given address of resolver used by the client, return the address of proxy near the client
  - HTTP proxy
    - Given the URL find nearest proxy that has content
    - Cache the content (DHT)

Using Coral

- Origin Server rewrites URLs
  - abc.com → abc.com.coralhost:coralport
  - Redirect clients to Coral server
- Coral CDN Components
  - DNS server
    - Given address of resolver used by the client, return the address of proxy near the client
  - HTTP proxy
    - Given the URL find nearest proxy that has content
    - Cache the content (DHT)
  - Distributed Sloppy Hash Table (DSHT)

Using Coral

- Origin Server rewrites URLs
  - abc.com → abc.com.coralhost:coralport
  - Redirect clients to Coral server
- Coral CDN Components
  - DNS server
    - Given address of resolver used by the client, return the address of proxy near the client
  - HTTP proxy
    - Given the URL find nearest proxy that has content
    - Cache the content (DHT)
  - Distributed Sloppy Hash Table (DSHT)
    - No load-balancing & content locality support in Basic DHTs (Chord)

Coral System Overview

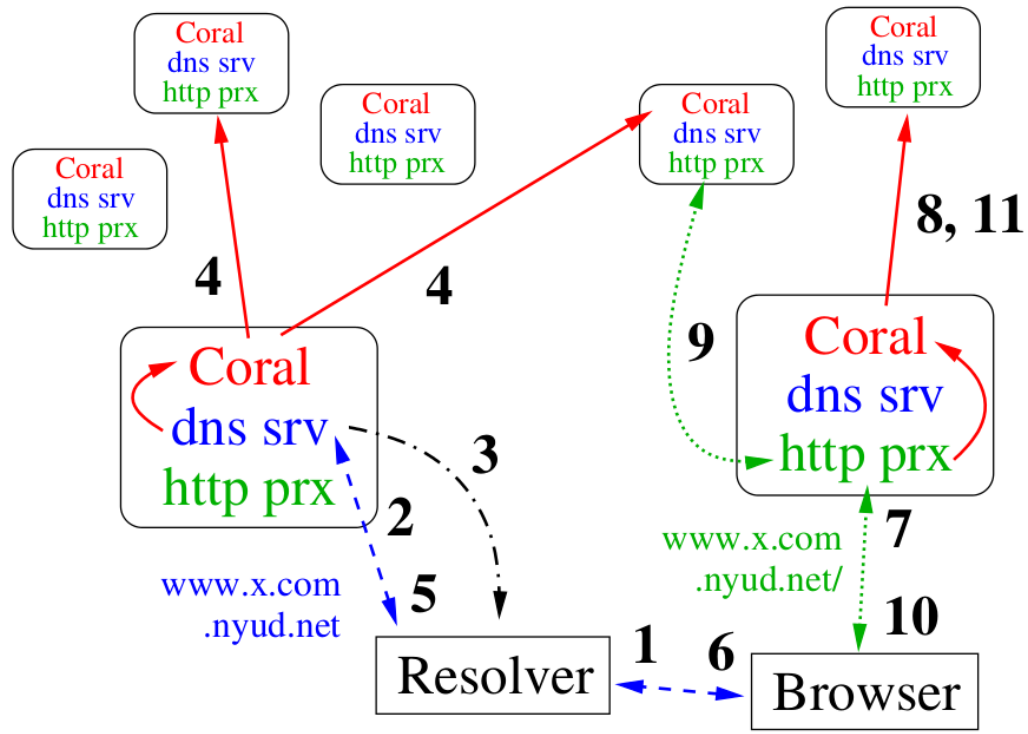


Figure 1: Using CoralCDN, the steps involved in resolving a Coralized URL and returning the corresponding fi le, per Section 2.2. Rounded boxes represent CoralCDN nodes running Coral, DNS, and HTTP servers. Solid arrows correspond to Coral RPCs, dashed arrows to DNS traffi c, dotted-dashed arrows to network probes, and dotted arrows to HTTP traffi c.

Freedman, Michael et al. **"Democratizing Content Publication with Coral."** In NSDI 2004.

Hierarchical Indexing

Hierarchical Indexing

- Diameter, Clusters, Levels
  - Each Coral Node part of several DSHTs called clusters
  - Each cluster characterized by max RTT (diameter)
  - Fixed hierarchy of diameters called levels
  - Group of nodes can form a level-i cluster if the pair-wise RTT less than threshold for level-i
    - Paper uses 3 (levels): 20ms (2), 60ms (1), ∞ (0)

Hierarchical Indexing

- Diameter, Clusters, Levels
  - Each Coral Node part of several DSHTs called clusters
  - Each cluster characterized by max RTT (diameter)
  - Fixed hierarchy of diameters called levels
  - Group of nodes can form a level-i cluster if the pair-wise RTT less than threshold for level-i
    - Paper uses 3 (levels): 20ms (2), 60ms (1), ∞ (0)
- SHA-1 for Coral Keys and Node-Ids

Hierarchical Indexing

- Diameter, Clusters, Levels
  - Each Coral Node part of several DSHTs called clusters
  - Each cluster characterized by max RTT (diameter)
  - Fixed hierarchy of diameters called levels
  - Group of nodes can form a level-i cluster if the pair-wise RTT less than threshold for level-i
    - Paper uses 3 (levels): 20ms (2), 60ms (1), ∞ (0)
- SHA-1 for Coral Keys and Node-Ids
- Bitwise XOR is distance (Kademlia)
  - Longer matching prefix numerically closer
  - Key stored at node having ID "close" to key

Routing and Sloppy Storage

Freedman, Michael et al. **"Democratizing Content Publication with Coral."** In NSDI 2004.

Routing and Sloppy Storage

- Routing
  - Routing table size logarithmic in total number of nodes

Freedman, Michael et al. **"Democratizing Content Publication with Coral."** In NSDI 2004.

Routing and Sloppy Storage

- Routing
  - Routing table size logarithmic in total number of nodes
- Sloppy Storage
  - Cache key/value pairs at nodes whose IDs are close to the key being referenced
  - Reduces hot-spot congestion for popular content

Freedman, Michael et al. **"Democratizing Content Publication with Coral."** In NSDI 2004.

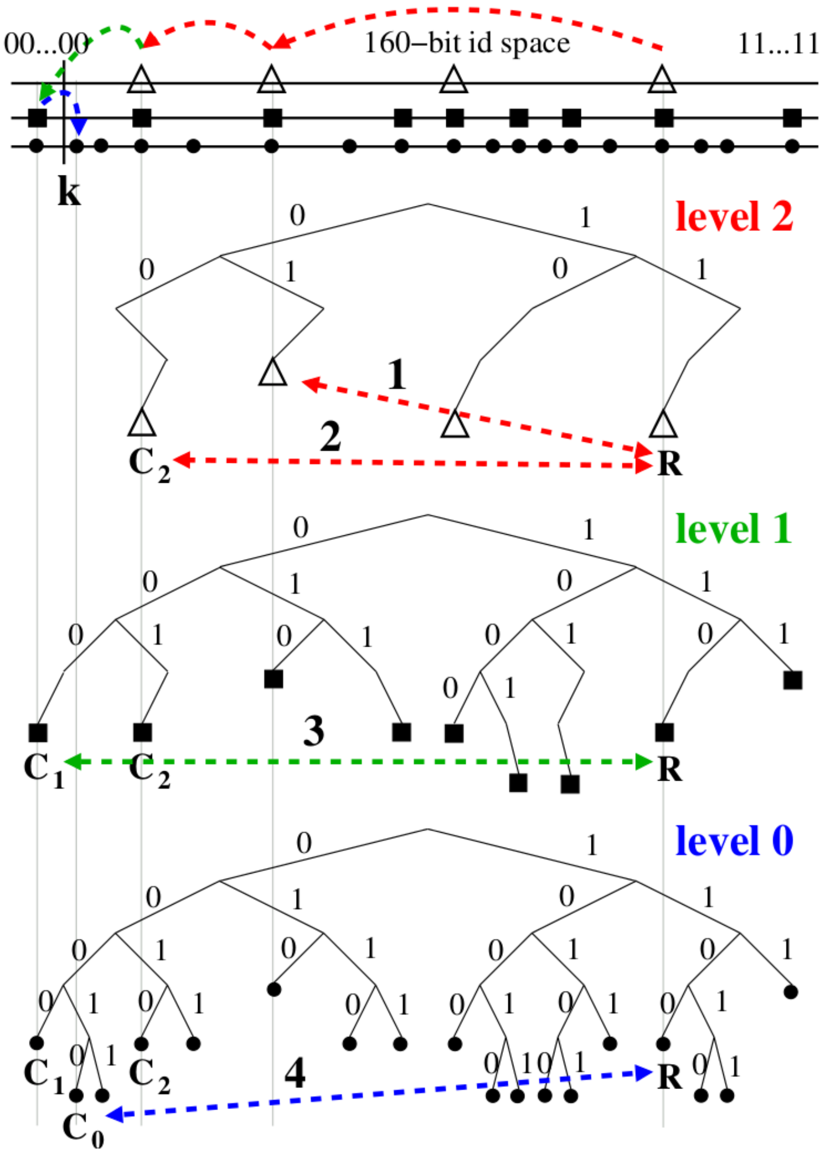
Routing and Sloppy Storage

- Routing
  - Routing table size logarithmic in total number of nodes
- Sloppy Storage
  - Cache key/value pairs at nodes whose IDs are close to the key being referenced
  - Reduces hot-spot congestion for popular content

Freedman, Michael et al. **"Democratizing Content Publication with Coral."** In NSDI 2004.

Routing and Sloppy Storage

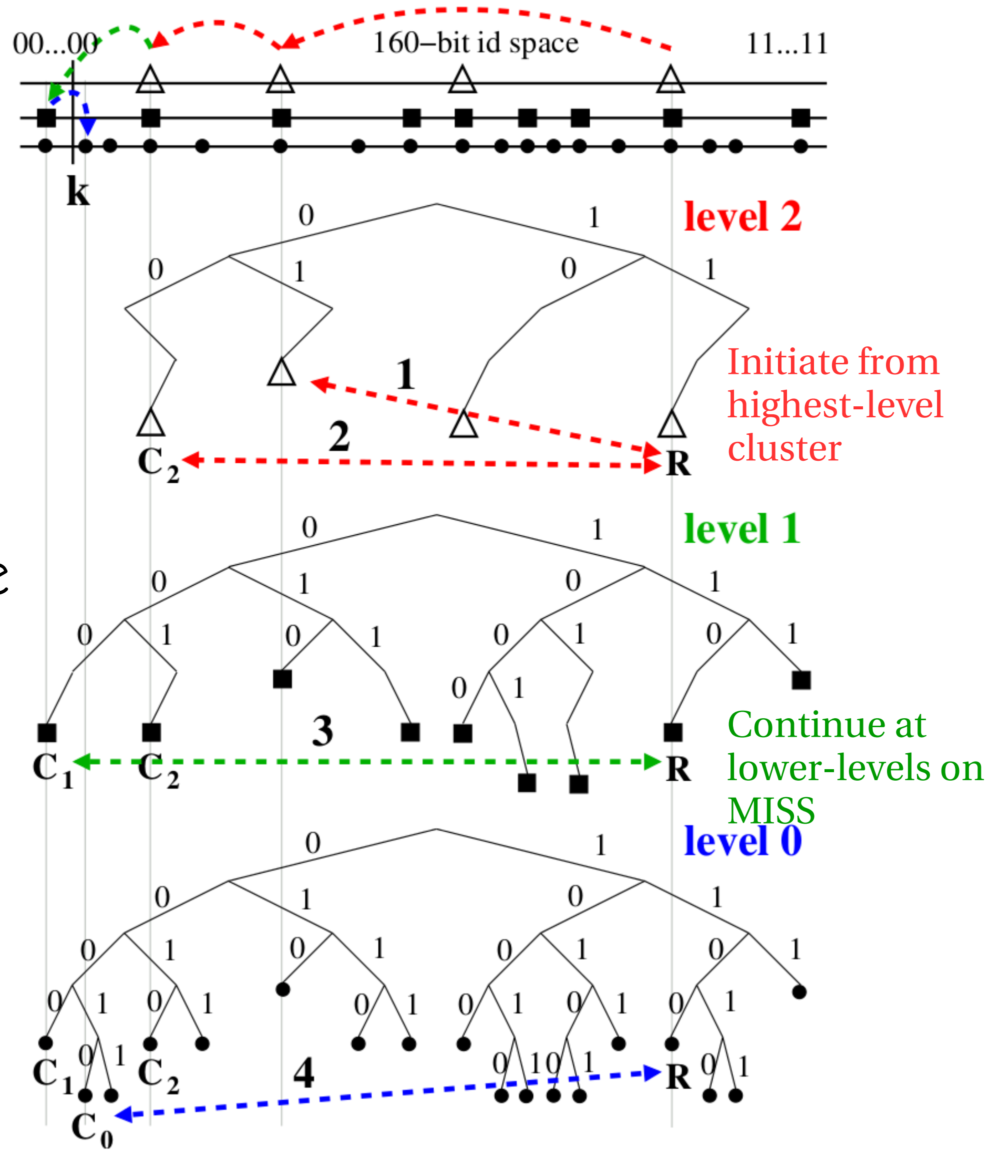
- Routing
  - Routing table size logarithmic in total number of nodes
- Sloppy Storage
  - Cache key/value pairs at nodes whose IDs are close to the key being referenced
  - Reduces hot-spot congestion for popular content



Freedman, Michael et al. **"Democratizing Content Publication with Coral."** In NSDI 2004.

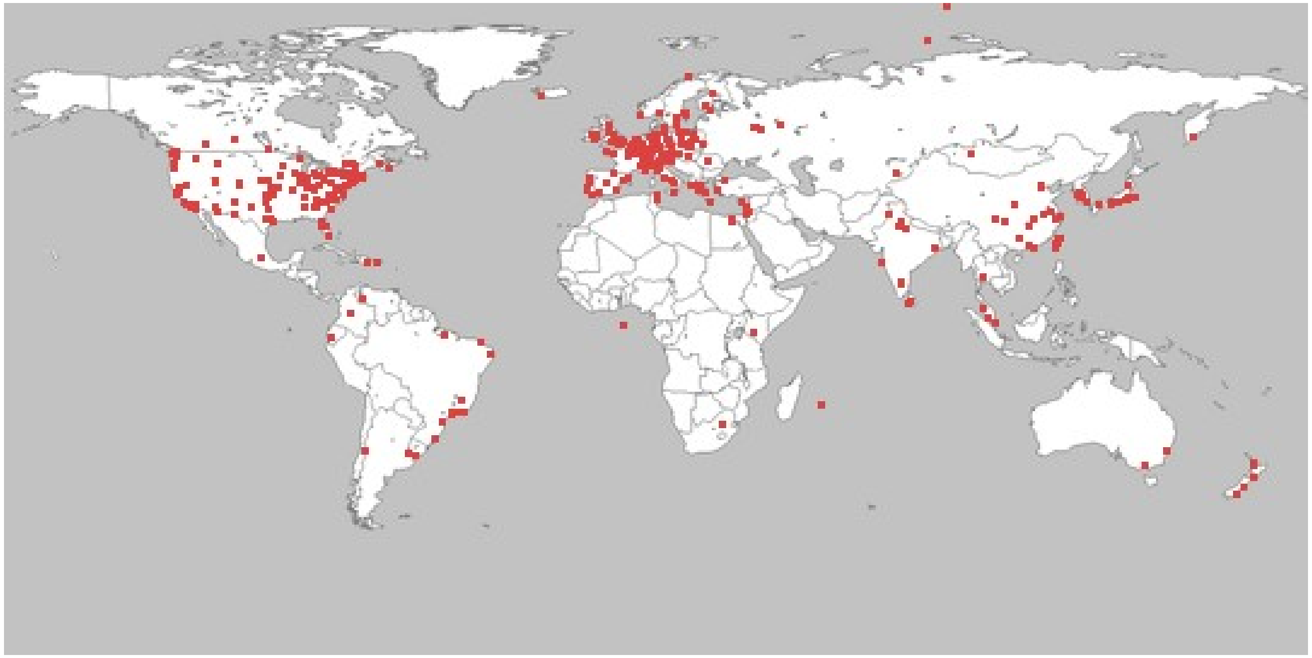
Routing and Sloppy Storage

- Routing
  - Routing table size logarithmic in total number of nodes
- Sloppy Storage
  - Cache key/value pairs at nodes whose IDs are close to the key being referenced
  - Reduces hot-spot congestion for popular content



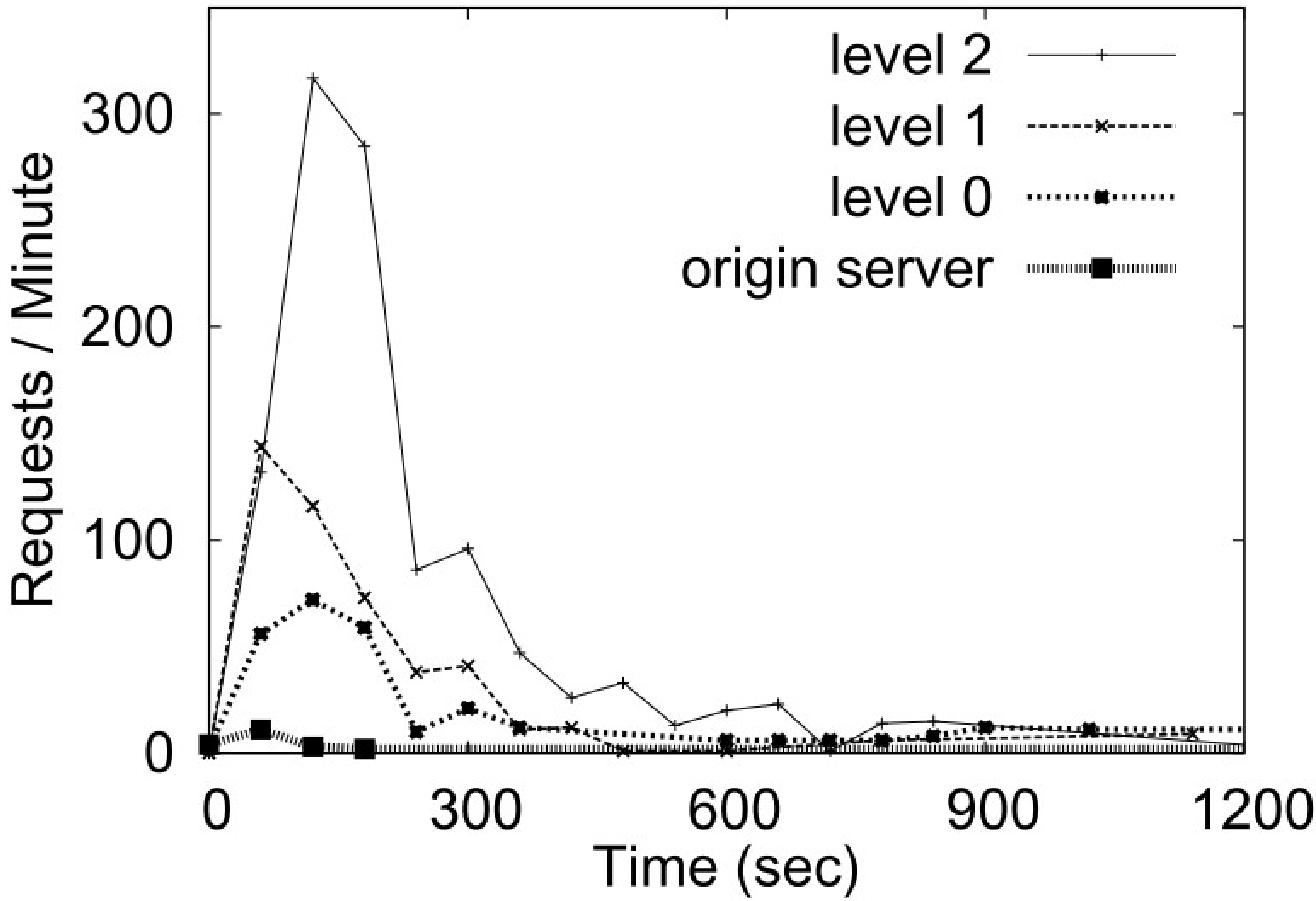
Freedman, Michael et al. **"Democratizing Content Publication with Coral."** In NSDI 2004.

Coral Implemented on PlanetLab



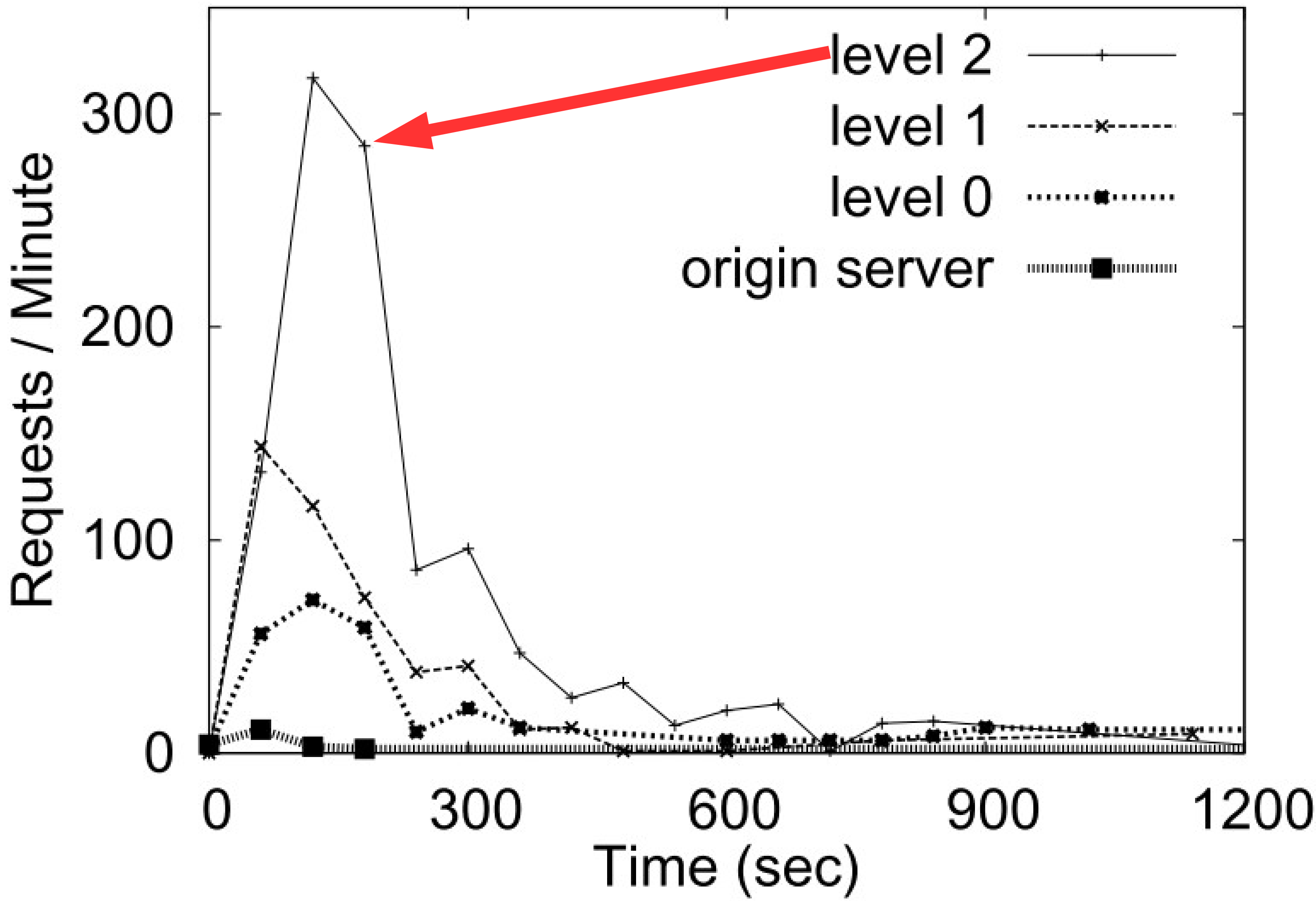
Global Research Network
As of Feb 2014, PlanetLab has 1181 nodes at 567 sites

Reduction in Server Load



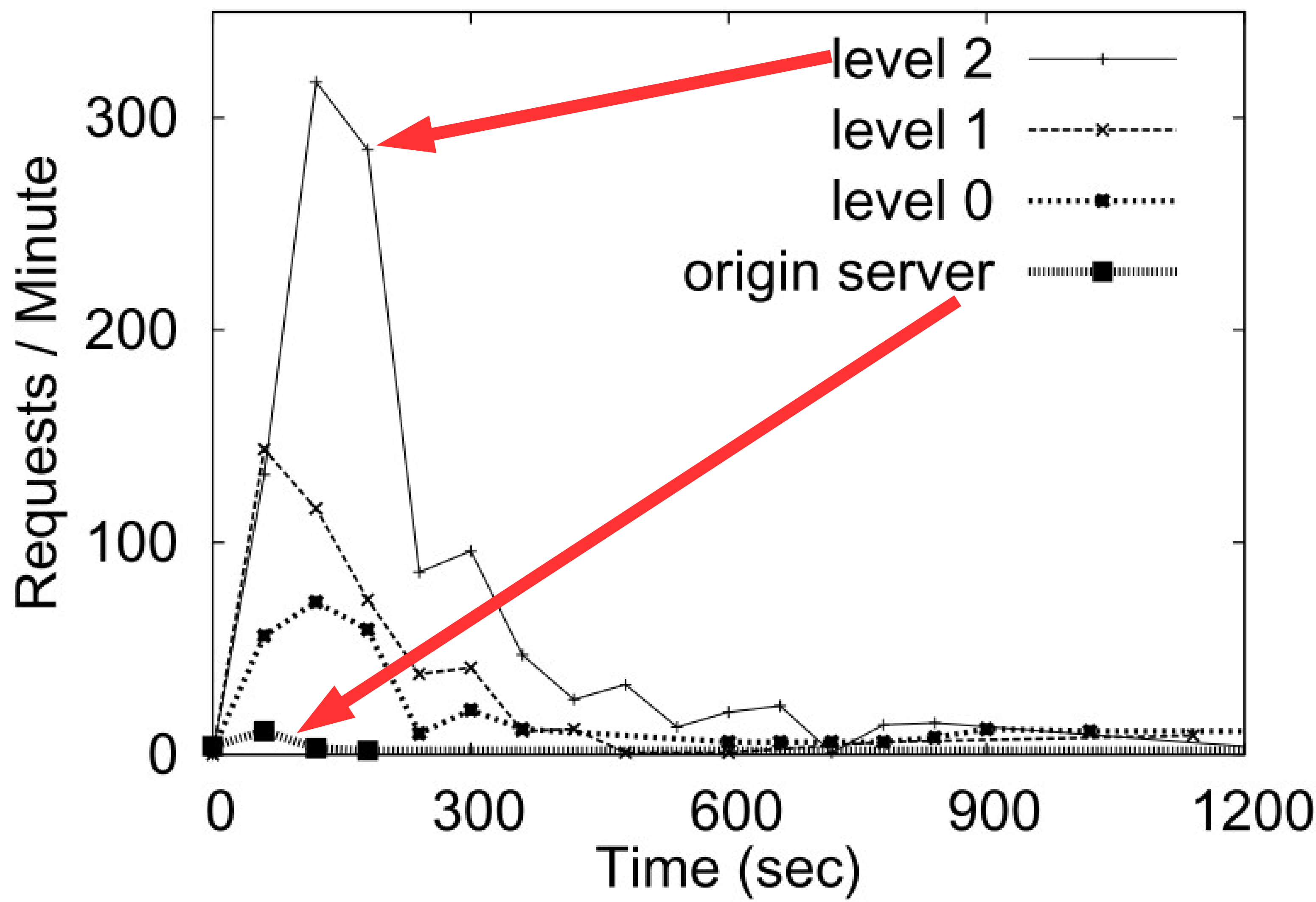
Freedman, Michael et al. **"Democratizing Content Publication with Coral."** In NSDI 2004.

Reduction in Server Load



Freedman, Michael et al. **"Democratizing Content Publication with Coral."** In NSDI 2004.

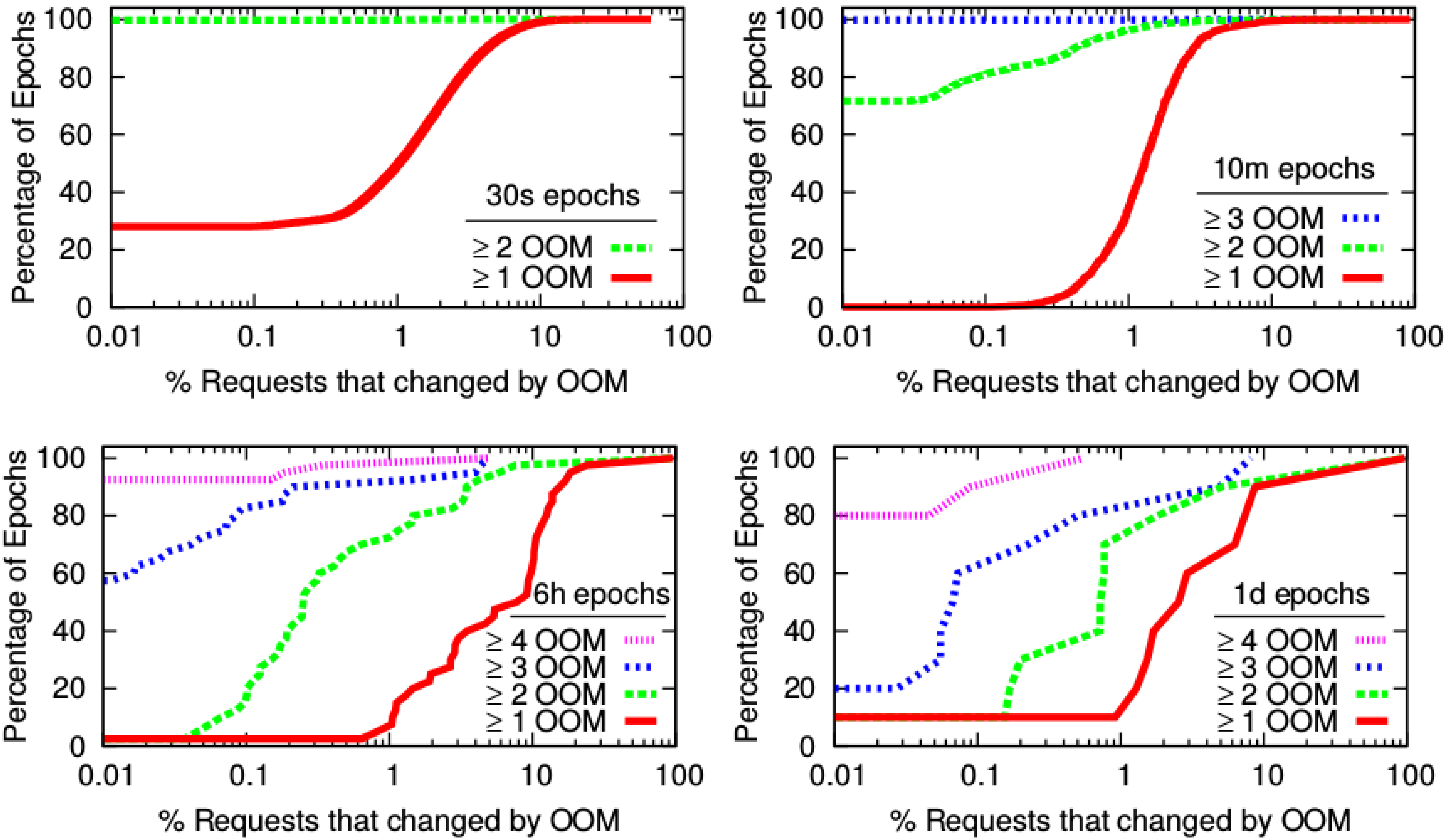
Reduction in Server Load



Freedman, Michael et al. **"Democratizing Content Publication with Coral."** In NSDI 2004.

Dynamics of Flash Crowds

28% of 30s epochs have no domains with a ≥ 1 OOM rate increase



Freedman, Michael. **"Experiences with CoralCDN: A Five-Year Operational View."** In NSDI 2010.

Insights from 5 year Deployment

- A large majority of its traffic does not require any cooperative caching
  - Handling of flash crowds relies on cooperative caching
- Flash Crowds
  - Small fraction of CoralCDN's domains experience large rate increases within short time periods
  - Flash crowd domains' traffic accounts for a small fraction of the total requests
  - Request rate increases very rarely occur on the order of seconds
- Content delivery via untrusted nodes requires the HTTP protocol to support end-to-end signatures for content integrity

Freedman, Michael. **"Experiences with CoralCDN: A Five-Year Operational View."** In NSDI 2010.

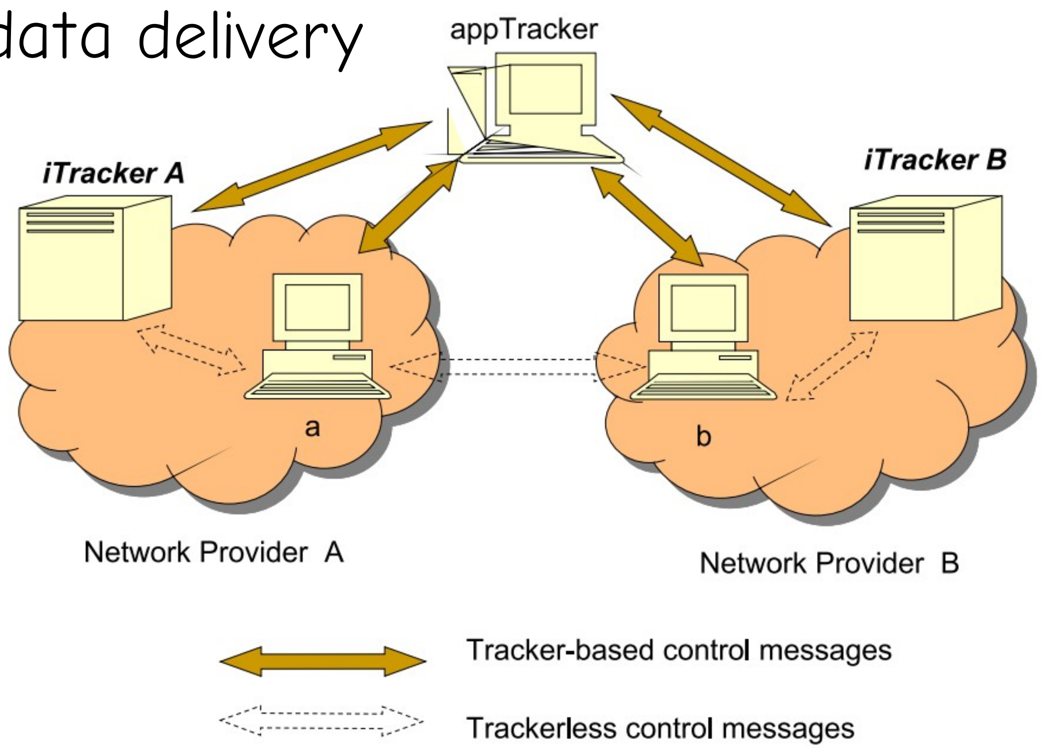
Other CDNs

P4P (Provider Portals for Applications)

- P2P applications may be oblivious to underlying network
  - Lot of inter-domain traffic (Karagiannis et al. 2005)
- Approaches to address this problem
  - ISP approaches
    - Block P2P, Rate-limit P2P, Cache content, etc.
  - P2P approaches
    - Locality (Ono Project)

itracker of P4P

- Network provider runs an iTracker
- itracker used by ISPs to provides additional information regarding network topology
  - P2P networks may choose to utilize to optimize network data delivery



Haiyong Xie et al. ”P4p: provider portal for applications.” In SIGCOMM 2008

Maygh P2P CDN

- P2P CDN on Browser

Liang Zhang et al. **"Maygh: building a CDN from client web browsers."** In EuroSys '13.

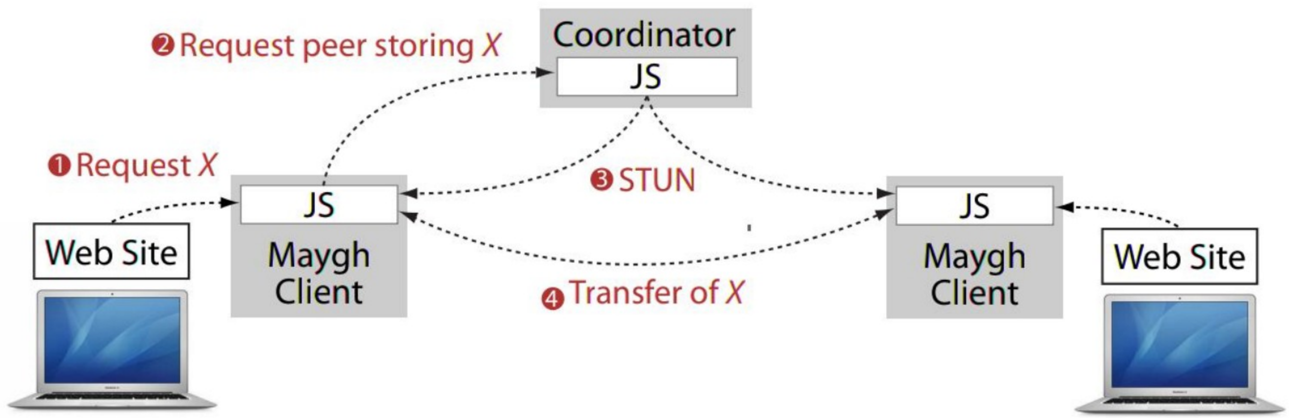
Maygh P2P CDN

- P2P CDN on Browser
  - Leverage on WebSockets, WebRTC, WebStorage API

Liang Zhang et al. **"Maygh: building a CDN from client web browsers."** In EuroSys '13.

Maygh P2P CDN

- P2P CDN on Browser
  - Leverage on WebSockets, WebRTC, WebStorage API



Liang Zhang et al. **"Maygh: building a CDN from client web browsers."** In EuroSys '13.

CDNI
(CDN Interconnection)

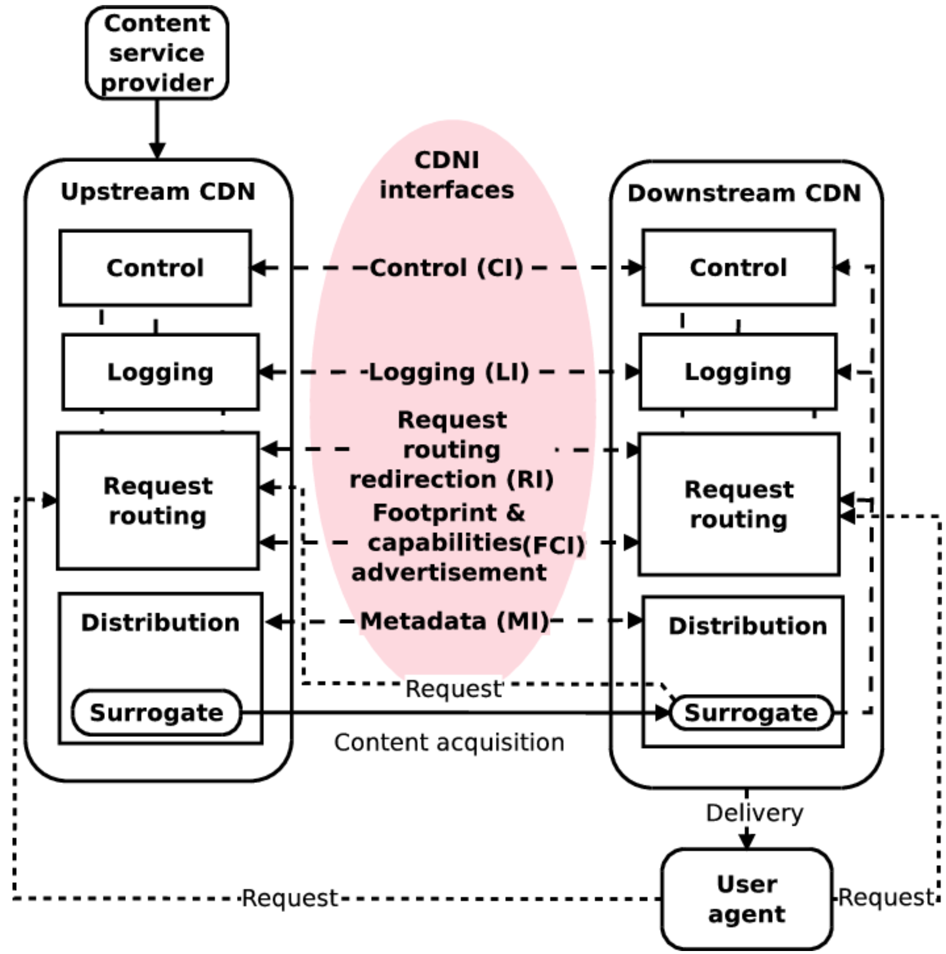
- Leverage collective CDN footprint
  - One CDN to reuse resources of another CDN provider
  - ISPs can deploy their own CDNs

Niven-Jenkins etal. **"Content distribution network interconnection (CDNI) problem statement."** RFC 6707 (2012).

CDNI

(CDN Interconnection)

- Leverage collective CDN footprint
  - One CDN to reuse resources of another CDN provider
  - ISPs can deploy their own CDNs



Niven-Jenkins etal. **"Content distribution network interconnection (CDNI) problem statement."** RFC 6707 (2012).

Amazon Dynamo

ACID
(Recap)

Theo Haerder et al. **"Principles of transaction-oriented database recovery."** ACM Computing Surveys. 1983.

ACID
(Recap)

- Atomicity

Theo Haerder et al. **"Principles of transaction-oriented database recovery."** ACM Computing Surveys. 1983.

ACID
(Recap)

- Atomicity
  - All or nothing

Theo Haerder et al. **"Principles of transaction-oriented database recovery."** ACM Computing Surveys. 1983.

ACID
(Recap)

- Atomicity
  - All or nothing
- Consistency

Theo Haerder et al. **"Principles of transaction-oriented database recovery."** ACM Computing Surveys. 1983.

ACID
(Recap)

- Atomicity
  - All or nothing
- Consistency
  - Successful transaction commits only legal results

Theo Haerder et al. **"Principles of transaction-oriented database recovery."** ACM Computing Surveys. 1983.

ACID
(Recap)

- Atomicity
  - All or nothing
- Consistency
  - Successful transaction commits only legal results
- Isolation

Theo Haerder et al. **"Principles of transaction-oriented database recovery."** ACM Computing Surveys. 1983.

ACID
(Recap)

- Atomicity
  - All or nothing
- Consistency
  - Successful transaction commits only legal results
- Isolation
  - Events within a transaction must be hidden from other transactions running concurrently

Theo Haerder et al. **"Principles of transaction-oriented database recovery."** ACM Computing Surveys. 1983.

ACID
(Recap)

- Atomicity
  - All or nothing
- Consistency
  - Successful transaction commits only legal results
- Isolation
  - Events within a transaction must be hidden from other transactions running concurrently
- Durability

Theo Haerder et al. **"Principles of transaction-oriented database recovery."** ACM Computing Surveys. 1983.

ACID
(Recap)

- Atomicity
  - All or nothing
- Consistency
  - Successful transaction commits only legal results
- Isolation
  - Events within a transaction must be hidden from other transactions running concurrently
- Durability
  - Once a transaction has been committed its results, the system must guarantee the results survive subsequent malfunctions

Theo Haerder et al. **"Principles of transaction-oriented database recovery."** ACM Computing Surveys. 1983.

CAP Theorem

A. Fox et al. **“Harvest, yield, and scalable tolerant systems.”** HotOS. 1999.

CAP Theorem

- **C**: Strong Consistency (single-copy ACID consistency)
- **A**: High Availability (available at all times)
- **P**: Partition Resilience (survive partition between replicas)

A. Fox et al. **“Harvest, yield, and scalable tolerant systems.”** HotOS. 1999.

CAP Theorem

- **C**: Strong Consistency (single-copy ACID consistency)
- **A**: High Availability (available at all times)
- **P**: Partition Resilience (survive partition between replicas)

PICK ANY TWO

A. Fox et al. **“Harvest, yield, and scalable tolerant systems.”** HotOS. 1999.

CAP Theorem

- **C**: Strong Consistency (single-copy ACID consistency)
- **A**: High Availability (available at all times)
- **P**: Partition Resilience (survive partition between replicas)

PICK ANY TWO

- C A without P

A. Fox et al. **"Harvest, yield, and scalable tolerant systems."** HotOS. 1999.

CAP Theorem

- **C**: Strong Consistency (single-copy ACID consistency)
- **A**: High Availability (available at all times)
- **P**: Partition Resilience (survive partition between replicas)

PICK ANY TWO

- C A without P
- C P without A

A. Fox et al. **"Harvest, yield, and scalable tolerant systems."** HotOS. 1999.

CAP Theorem

- **C**: Strong Consistency (single-copy ACID consistency)
- **A**: High Availability (available at all times)
- **P**: Partition Resilience (survive partition between replicas)

PICK ANY TWO

- C A without P
- C P without A
- A P without C

A. Fox et al. **“Harvest, yield, and scalable tolerant systems.”** HotOS. 1999.

CAP Theorem

- **C**: Strong Consistency (single-copy ACID consistency)
- **A**: High Availability (available at all times)
- **P**: Partition Resilience (survive partition between replicas)

PICK ANY TWO

- C A without P
- C P without A
- A P without C

Popular work-around – reduced consistency (eventual consistency) or reduced availability

A. Fox et al. **"Harvest, yield, and scalable tolerant systems."** HotOS. 1999.

Two Phase Perspective of CAP

- Two phase commit
  - P1: Coordinator asks databases to perform a pre-commit and asks them if commit is possible. If all DBs agree then proceed to P2
  - P2: Coordinator asks DBs to commit
- Two phase commit supports consistency and partitioning. How is availability violated?
  - ***Availability of any system is the product of the availability of the components required for the operation***
- ACID provides Consitency. Partion Tolerance is essential. How do you achieve Availability?
  - BASE

Dan Pritchett. **"BASE: An Acid Alternative."** ACM Queue. 2008

BASE

- Basically available, Soft state, Eventually consistent
- Strong vs Eventual (informal comparison)
  - Strong: Every replica sees every update in the same order (atomic updates)
  - Eventual: every replica will eventually see updates and eventually agree on all values (non-atomic updates)
- Eventual Consistency
  - Database consistency will be in a state of flux but eventually it will be consistent
  - Reads might not return the results of the latest update

Dan Pritchett. **“BASE: An Acid Alternative.”** ACM Queue. 2008

Requirements from Dynamo

G. DeCandia et al. **"Dynamo: Amazon's Highly Available Key-value Store,"** In SOSP 2007.

Requirements from Dynamo

- Key-value store
  - shopping carts, seller lists, preferences, product catalog

G. DeCandia et al. **"Dynamo: Amazon's Highly Available Key-value Store,"** In SOSP 2007.

Requirements from Dynamo

- Key-value store
  - shopping carts, seller lists, preferences, product catalog
- System built using off-the-shelf hardware.
- Platform must scale to support continuous growth
- Address tradeoff of high-availability, guaranteed performance, cost-effectiveness, and performance

G. DeCandia et al. **"Dynamo: Amazon's Highly Available Key-value Store,"** In SOSP 2007.

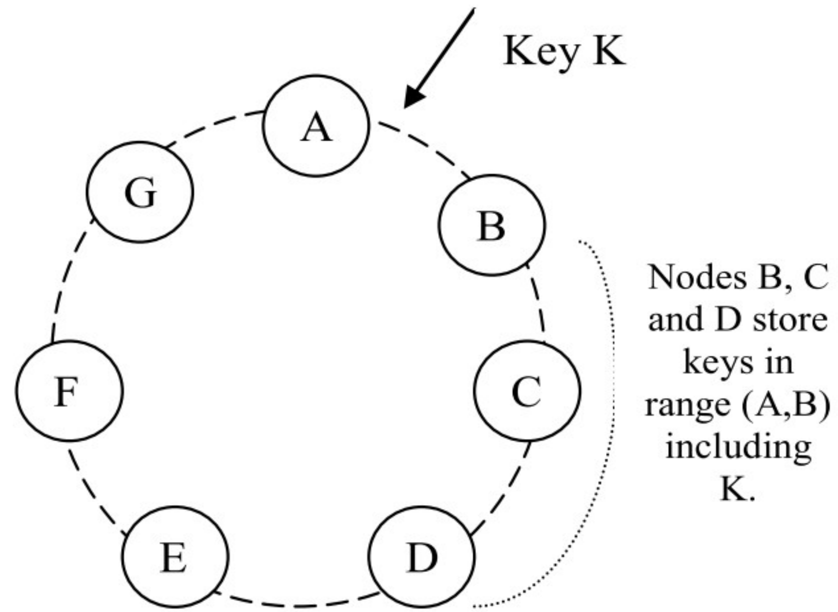
Requirements from Dynamo

- Key-value store
  - shopping carts, seller lists, preferences, product catalog
- System built using off-the-shelf hardware.
- Platform must scale to support continuous growth
- Address tradeoff of high-availability, guaranteed performance, cost-effectiveness, and performance
  - ”The system needs to have scalable and robust solutions for load balancing, membership and failure detection, failure recovery, replica synchronization, overload handling, state transfer, concurrency and job scheduling, request marshalling, request routing, system monitoring and alarming, and configuration management”

G. DeCandia et al. **“Dynamo: Amazon’s Highly Available Key-value Store,”** In SOSP 2007.

Partitioning and Replication in Dynamo

- Consistent Hashing DHT
  - Virtual nodes in DHT
  - Each physical node added as multiple virtual nodes
- Each data-item replicated in N nodes
  - Each virtual node responsible for the region between it and its Nth predecessor
  - Preference List: list of nodes (in (multiple datacenters) storing a key



G. DeCandia et al. **"Dynamo: Amazon's Highly Available Key-value Store,"** In SOSP 2007.

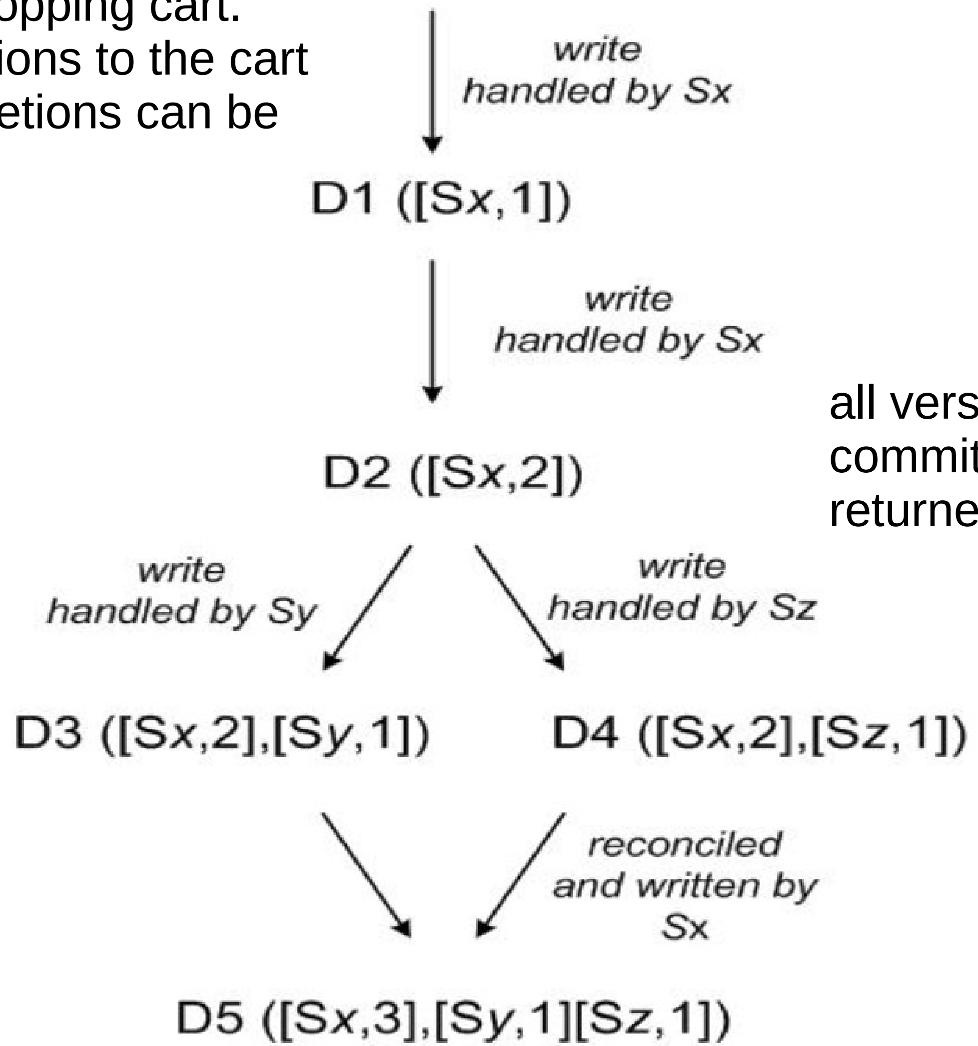
API

- get (key)
  - may return many versions of the same object
- put(key, context, object)
  - Context: encodes system metadata and includes information such as the version of the object
  - may return to its caller before the update has been applied at all the replicas
  - An object may have different version sub-histories
- Vector clock based versioning
  - One vector clock associated with every version of objects

Data Versioning

Objects versions: D1, D2, D3, ...

Assume object is shopping cart. Requirements: additions to the cart don't get lost but deletions can be lost



all versions of the object committed to the system are returned when read

G. DeCandia et al. **"Dynamo: Amazon's Highly Available Key-value Store,"** In SOSP 2007.

Sloppy Quorum

Sloppy Quorum

- Read + Write involves N nodes from the preference list
  - R: minimum number of nodes for Read
  - W: minimum number of nodes for Write

Sloppy Quorum

- Read + Write involves N nodes from the preference list
  - R: minimum number of nodes for Read
  - W: minimum number of nodes for Write
- R + W > N
  - R = W = 5 → high consistency but system is vulnerable to network partitions
  - R = W = 1 → weak consistency with failure
  - Typical values of (N, R, W) = (3,2,2) → balance between performance and consistency

Read and Write Operations

Read and Write Operations

- Coordinator
  - Node responsible for read/writes
  - First node in the preference list

Read and Write Operations

- Coordinator
  - Node responsible for read/writes
  - First node in the preference list
- Write Operation

Read and Write Operations

- Coordinator
  - Node responsible for read/writes
  - First node in the preference list
- Write Operation
  - New vector clock from coordinator
  - Write locally and forward to N-1 nodes, if W-1 nodes respond then write was successful

Read and Write Operations

- Coordinator
  - Node responsible for read/writes
  - First node in the preference list
- Write Operation
  - New vector clock from coordinator
  - Write locally and forward to N-1 nodes, if W-1 nodes respond then write was successful
- Read Operation

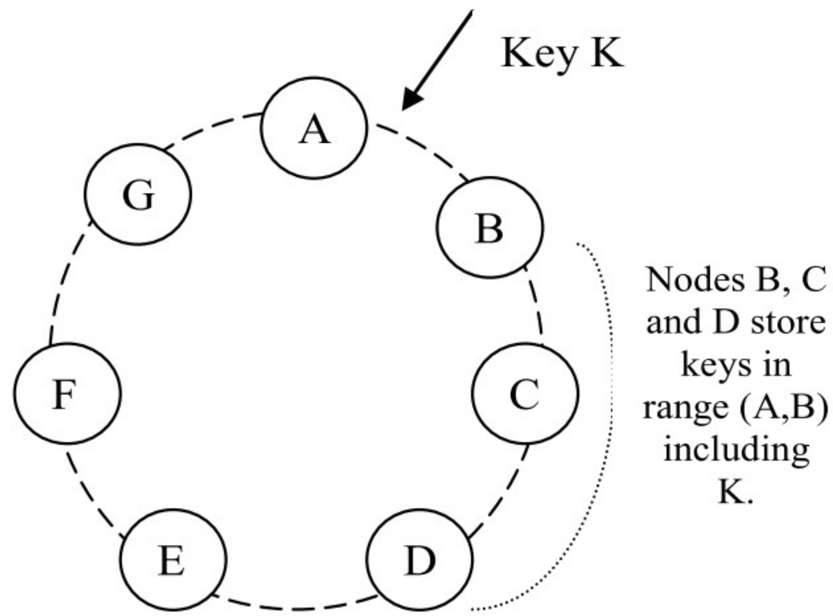
Read and Write Operations

- Coordinator
  - Node responsible for read/writes
  - First node in the preference list
- Write Operation
  - New vector clock from coordinator
  - Write locally and forward to N-1 nodes, if W-1 nodes respond then write was successful
- Read Operation
  - Forward request to N-1 nodes, if R-1 nodes respond then forward to user
  - User resolves conflicts and writes back result

Membership Changes

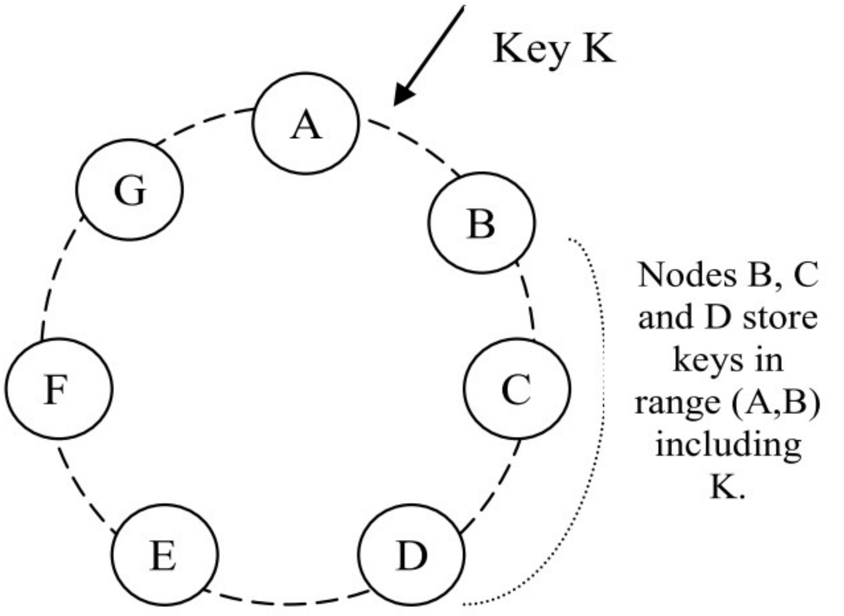
- Gossip-based Protocol to propagate membership changes
  - Each node contacts a peer chosen at random every second and the two nodes efficiently reconcile their persisted membership change histories
- Each node is aware of the key ranges handled by its peers

Handling Failures: Hinted Handoff



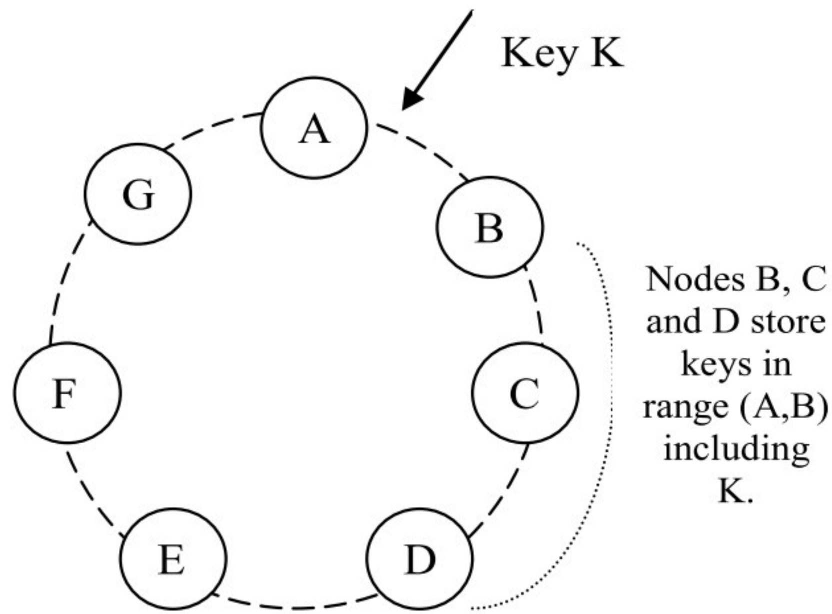
Handling Failures: Hinted Handoff

- Imagine A goes down and N=3



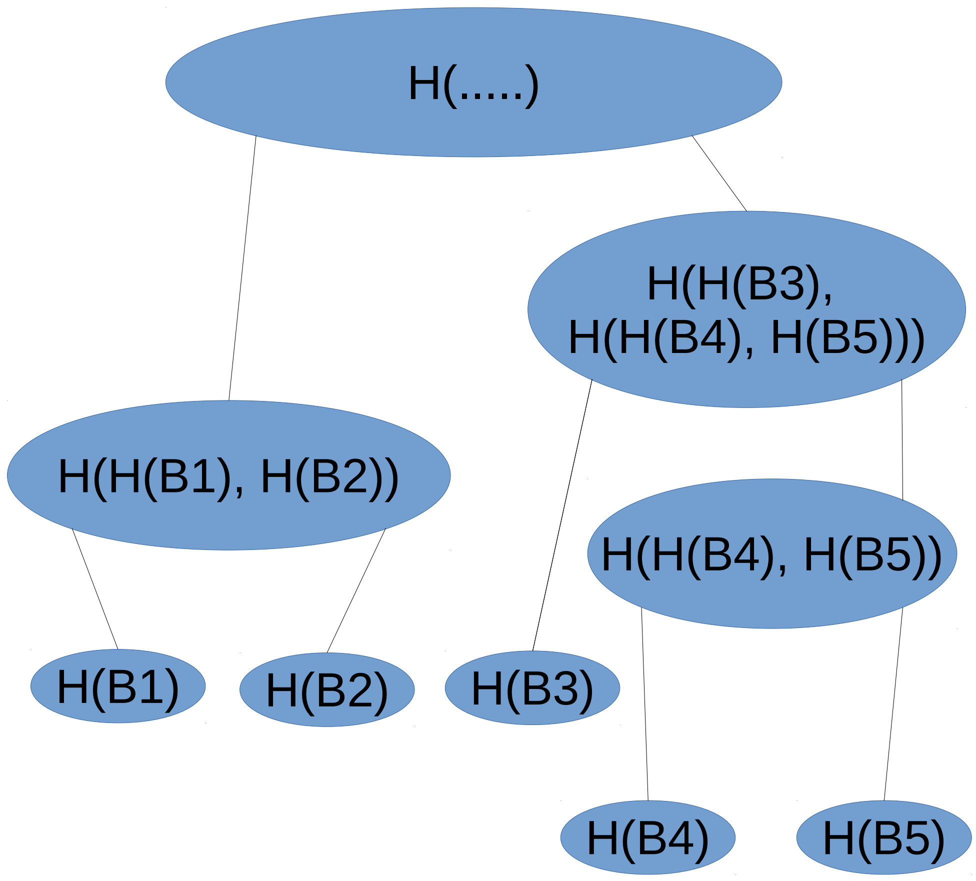
Handling Failures: Hinted Handoff

- Imagine A goes down and N=3
- Keys stored by A will now be stored by D
- D is hinted in the metadata that it is storing keys meant for A
- When A recovers, the keys at D are now copied to A



Handling Failures: Merkle Trees

- Minimize the amount of transferred data
- Merkle Tree:
  - Leaves are hashes of keys
  - Parents are hashes of children
- Each node maintains seperate Merkle tree for each key-range



Summary

Problem	Technique	Advantage
Partitioning	Consistent Hashing	Incremental Scalability
High Availability for writes	Vector clocks with reconciliation during reads	Version size is decoupled from update rates.
Handling temporary failures	Sloppy Quorum and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available.
Recovering from permanent failures	Anti-entropy using Merkle trees	Synchronizes divergent replicas in the background.
Membership and failure detection	Gossip-based membership protocol and failure detection.	Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information.

G. DeCandia et al. **"Dynamo: Amazon's Highly Available Key-value Store,"** In SOSP 2007.

References

Important References

- A. Fox et al. **“Harvest, yield, and scalable tolerant systems.”** HotOS. 1999.
- Theo Haerder et al. **"Principles of transaction-oriented database recovery."** ACM Computing Surveys. 1983.
- Dan Pritchett. **“BASE: An Acid Alternative.”** ACM Queue. 2008.
- G. DeCandia et al. **“Dynamo: Amazon’s Highly Available Key-value Store,”** In SOSP 2007.