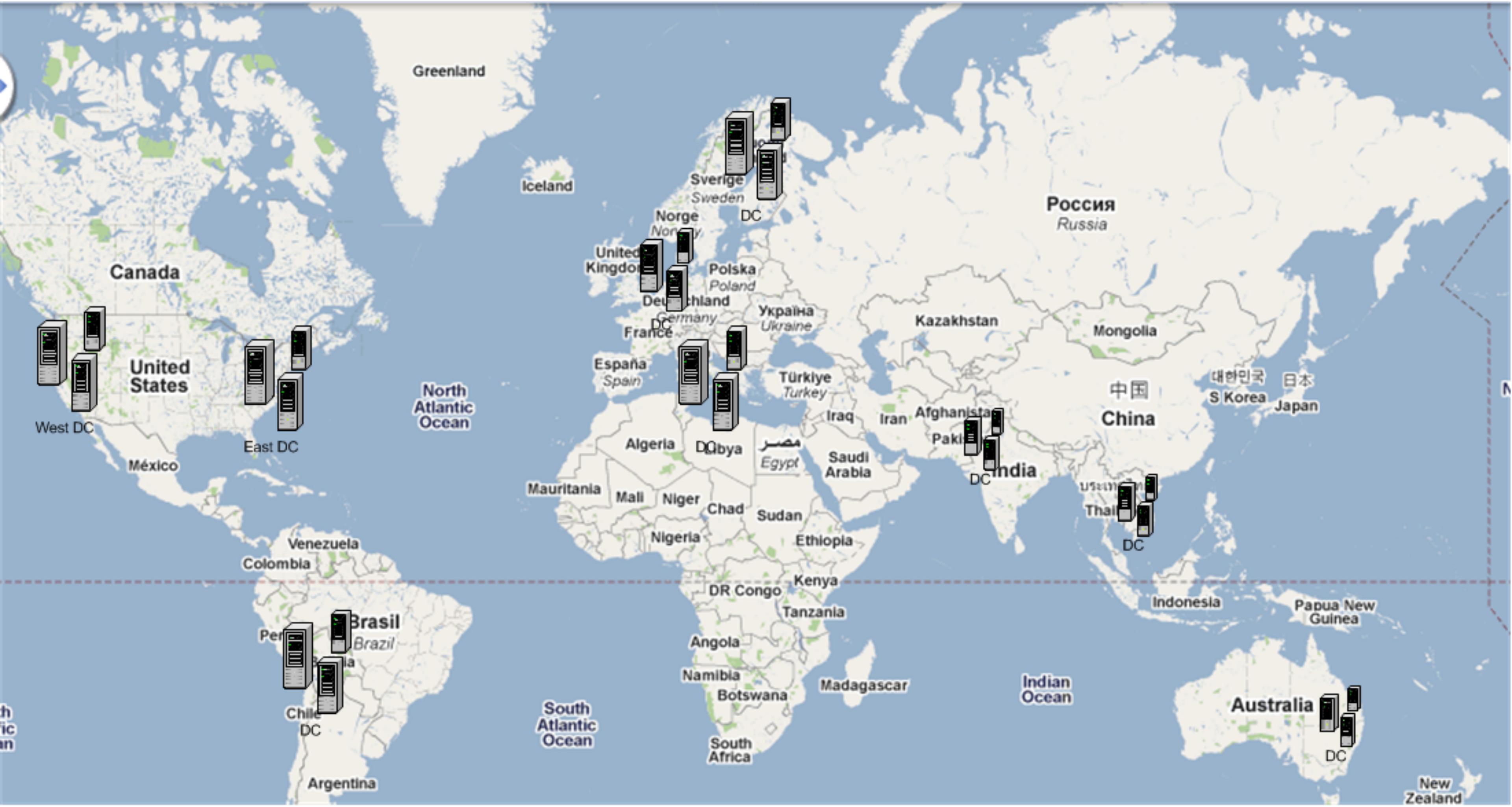
Semantics of Caching with SPOCA - A Stateless, Proportional, Optimally-Consistent Addressing Algorithm

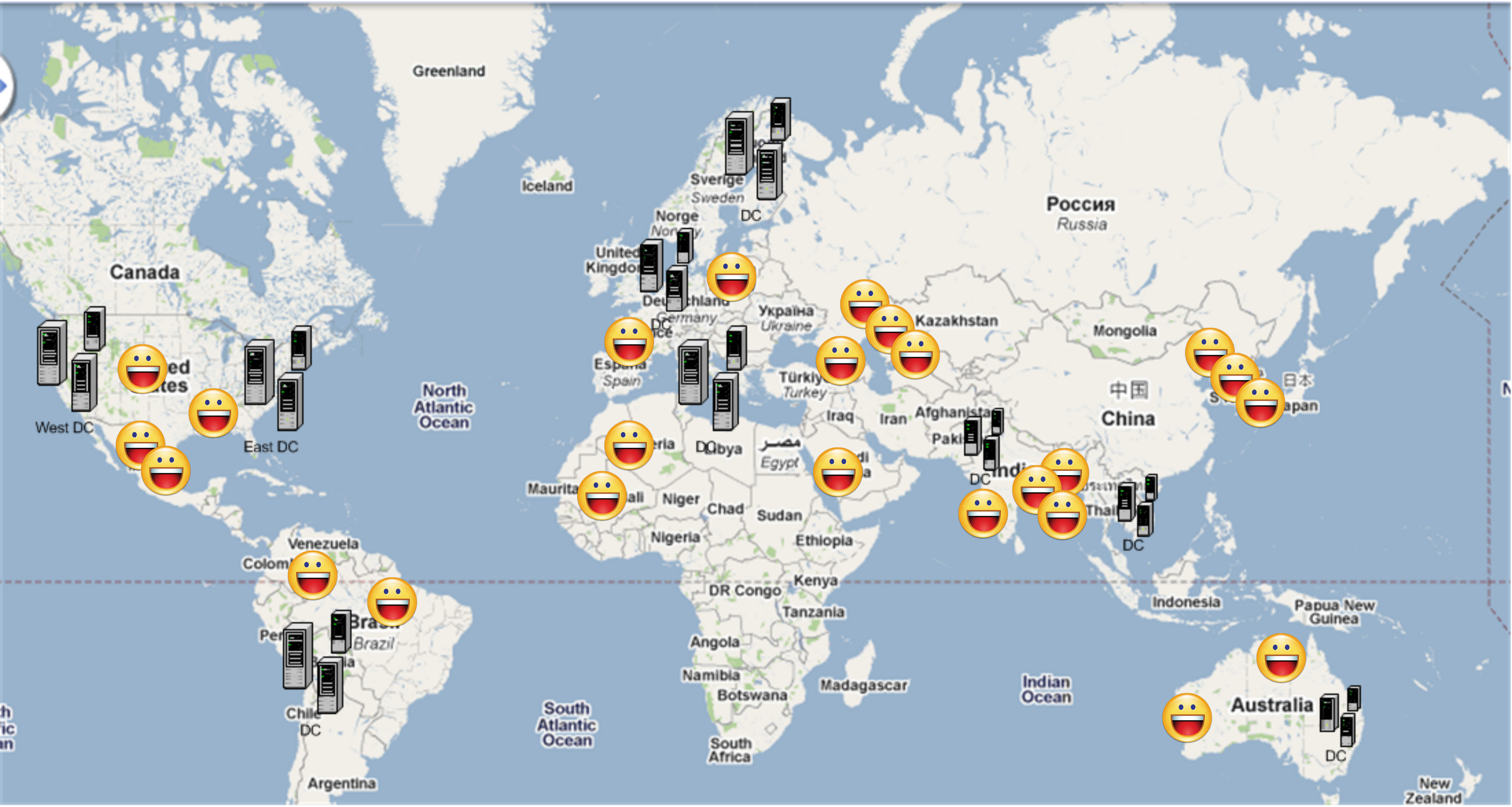
Ashish Chawla, Benjamin Reed, Karl Juhnke, Ghousuddin Syed

Yahoo! Inc

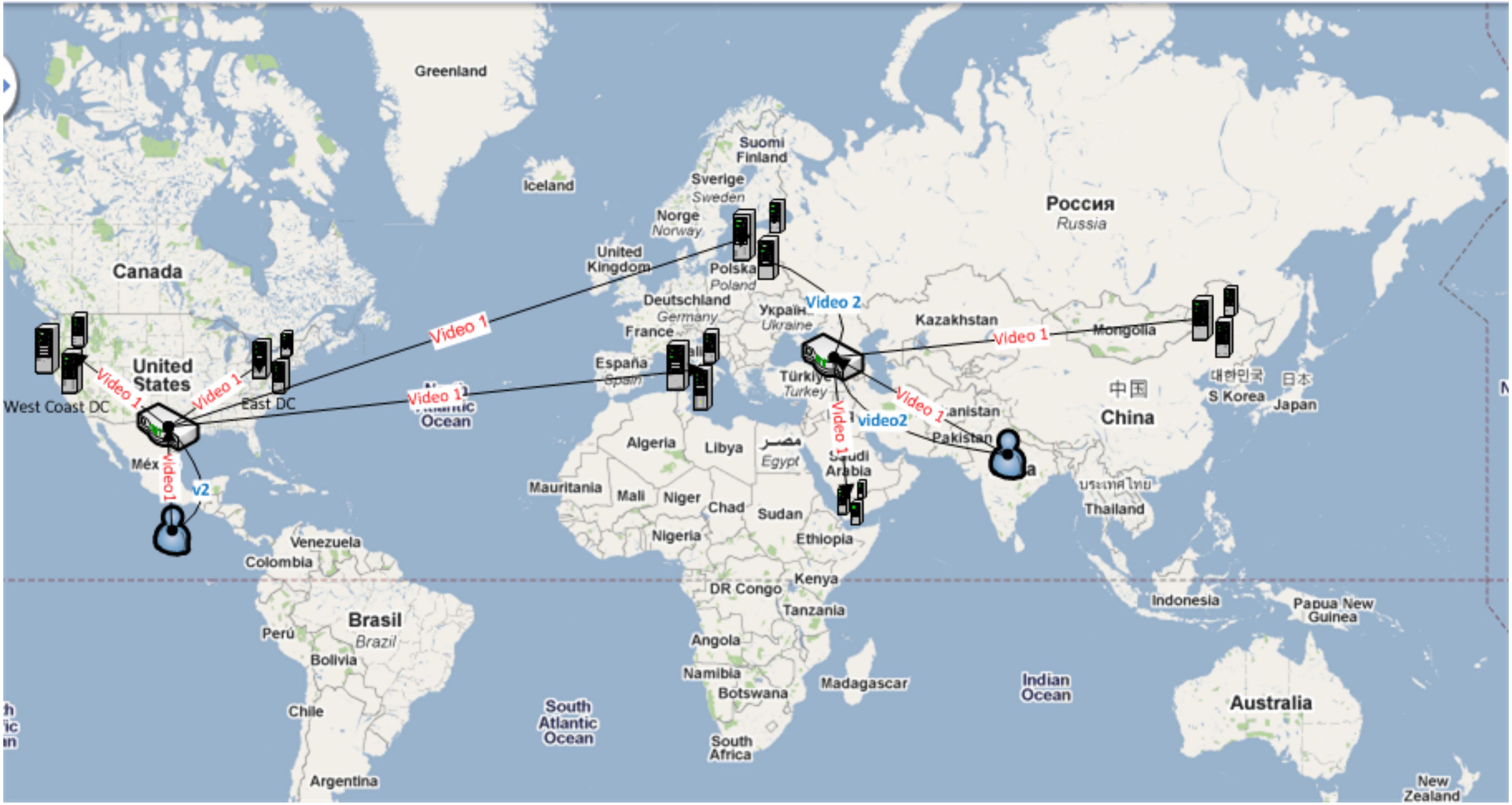
Video Platform



Video Platform



Simple Content Serving Architecture



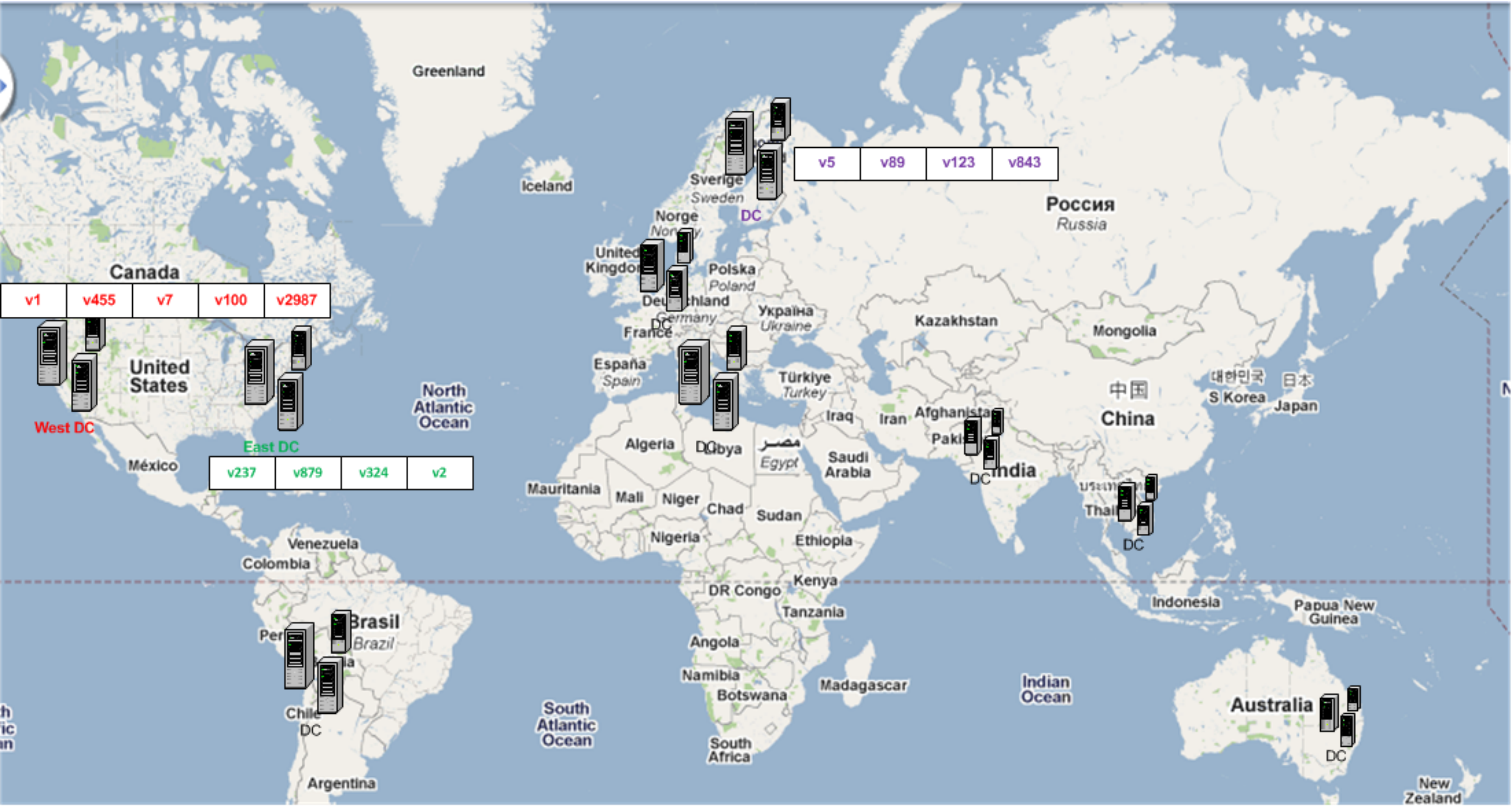
Outline

- Introduction
- **Problem Definition**
- SPOCA and Requirements
- Evaluations
- Conclusion

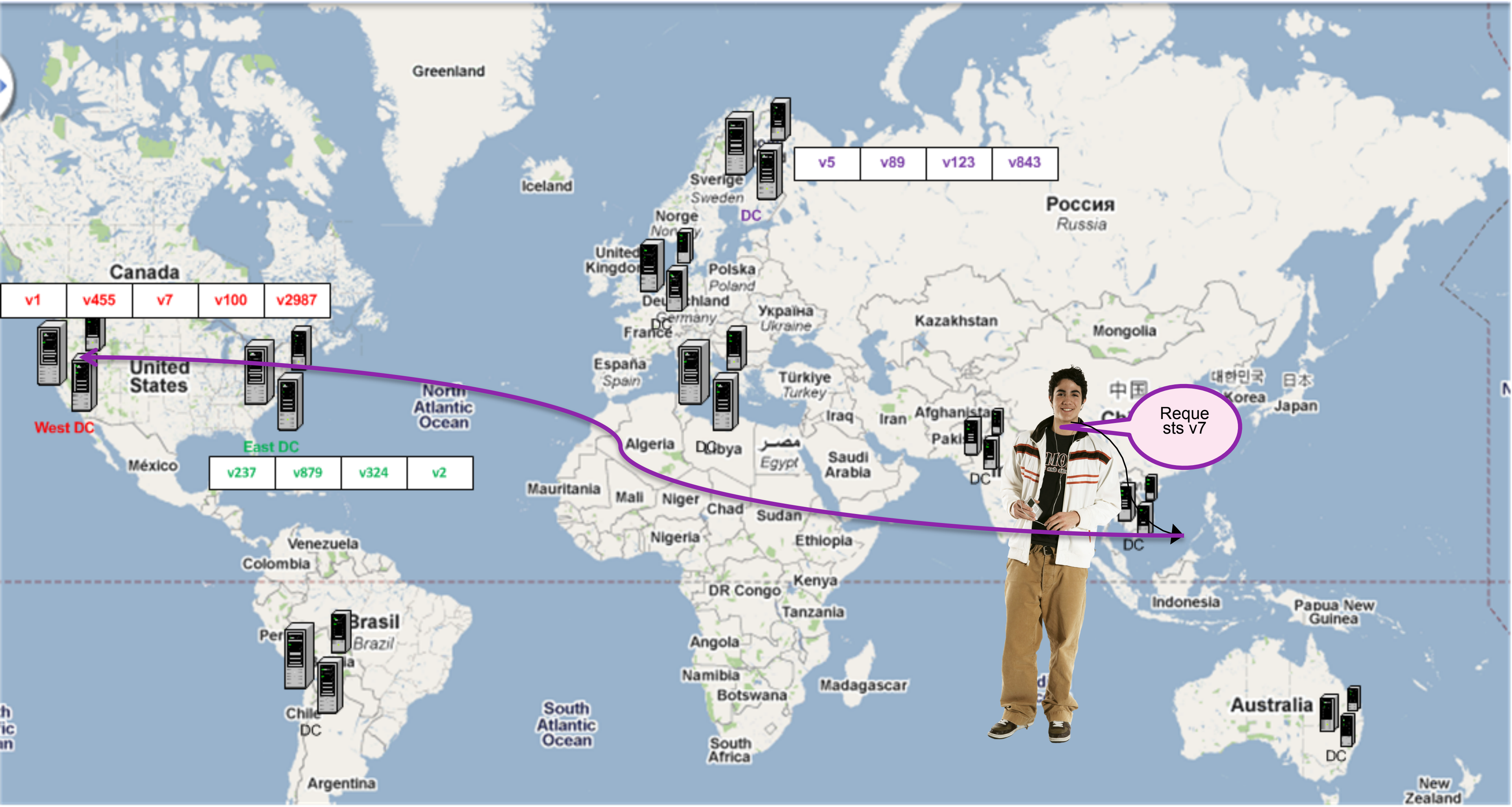
The Problem

- The front-end server disks are a secondary bottleneck.
- Eliminating redundant caching of content also reduces the load on the storage farm.
- An intelligent request-routing policy can produce far more caching efficiency than even a perfect cache promotion policy that must labor under random request routing.
- The cache promotion algorithm not enough.

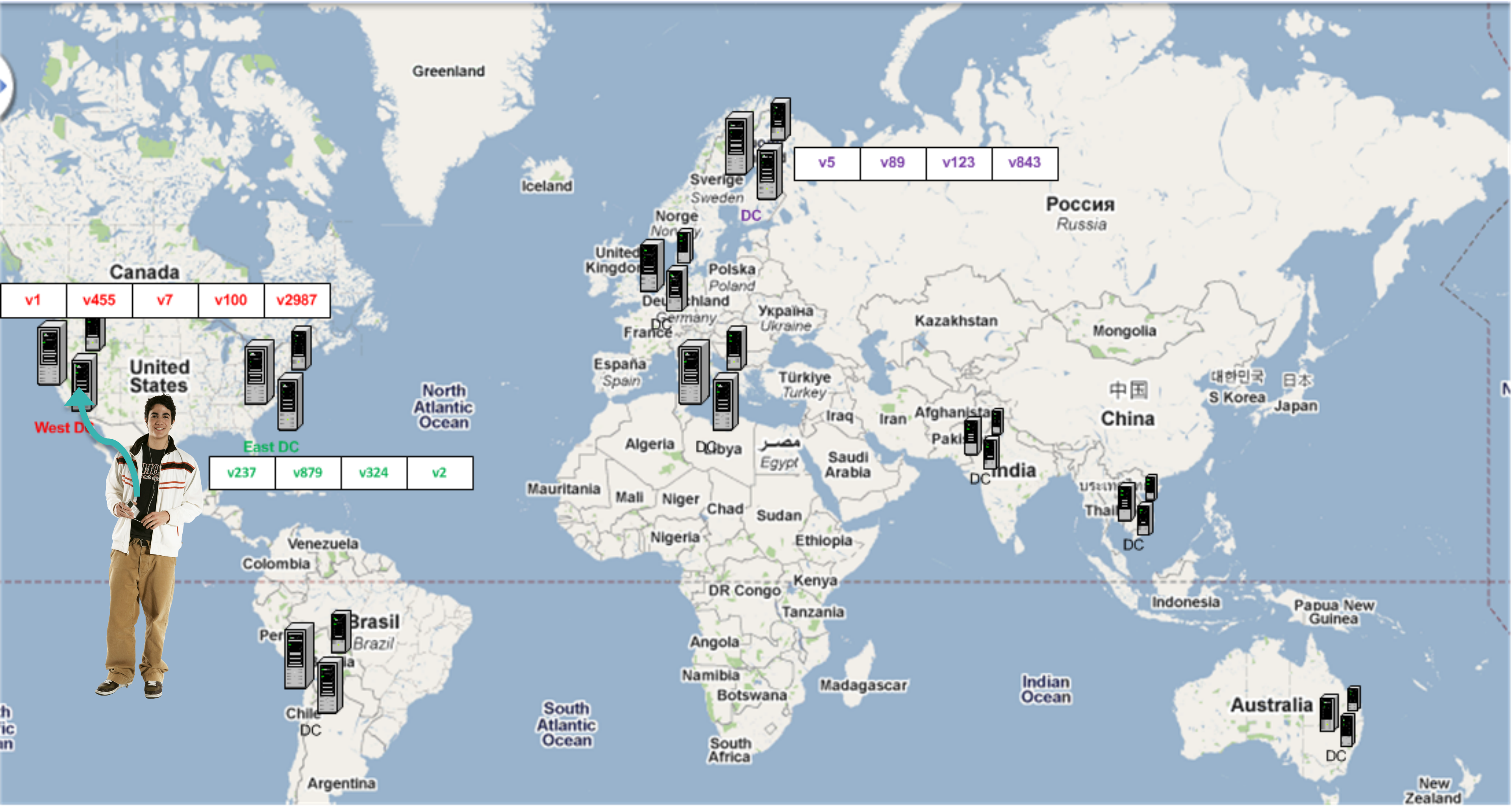
Problems from Geographic Distribution



Problems from Geographic Distribution



Problems from Geographic Distribution



Outline

- Introduction
- Problem Definition
- **SPOCA and Requirements**
- Evaluations
- Conclusion

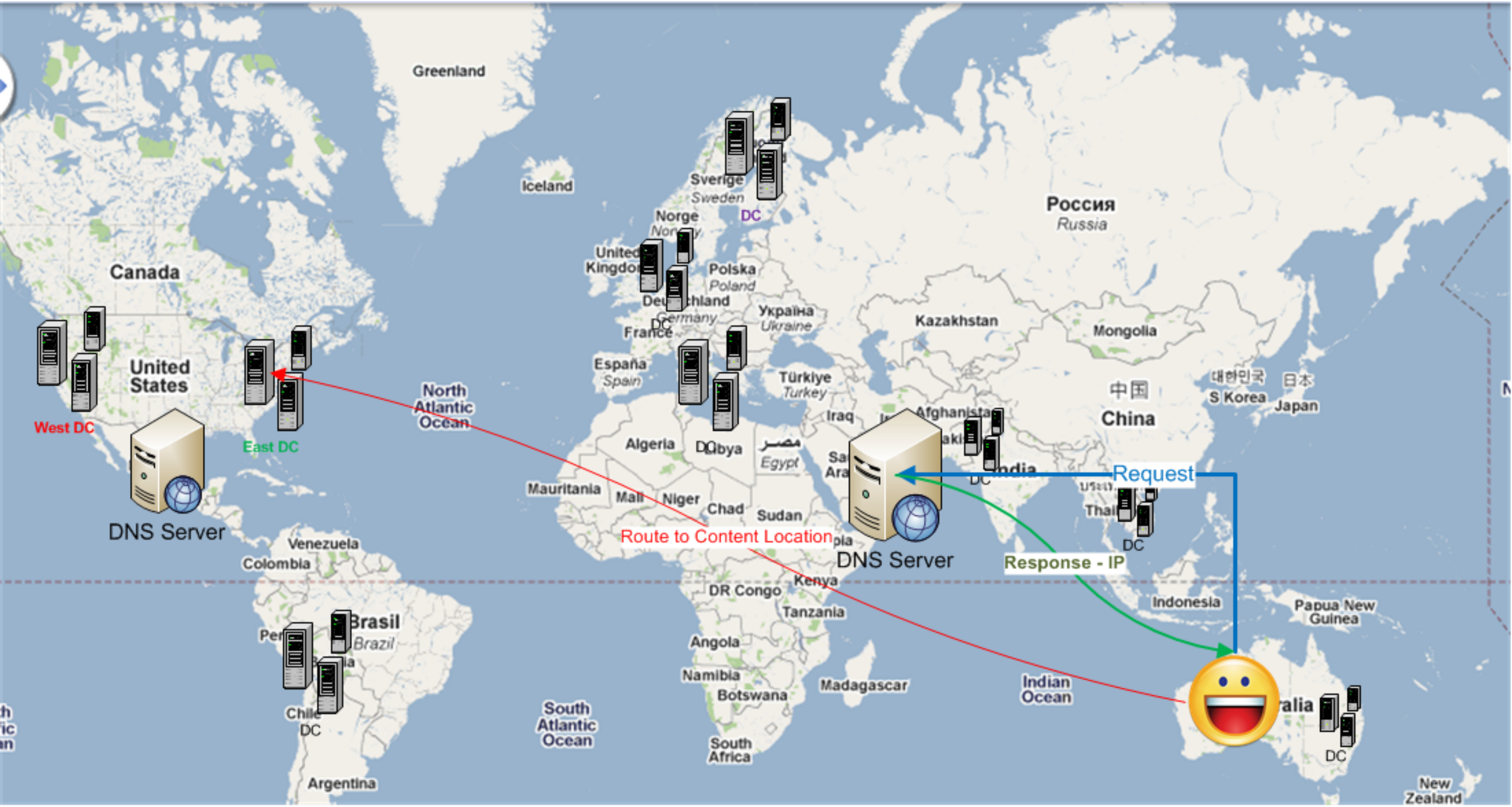
Requirements

- Merge different delivery pools and manage the diverse requirements in an adaptive way.
- Minimize caching disruptions when front-end server leaves or enters the pool - re-address as few files as possible to different servers.
- Proportional distribution of files among servers does not necessarily result in a proportional distribution of requests (Power Law)

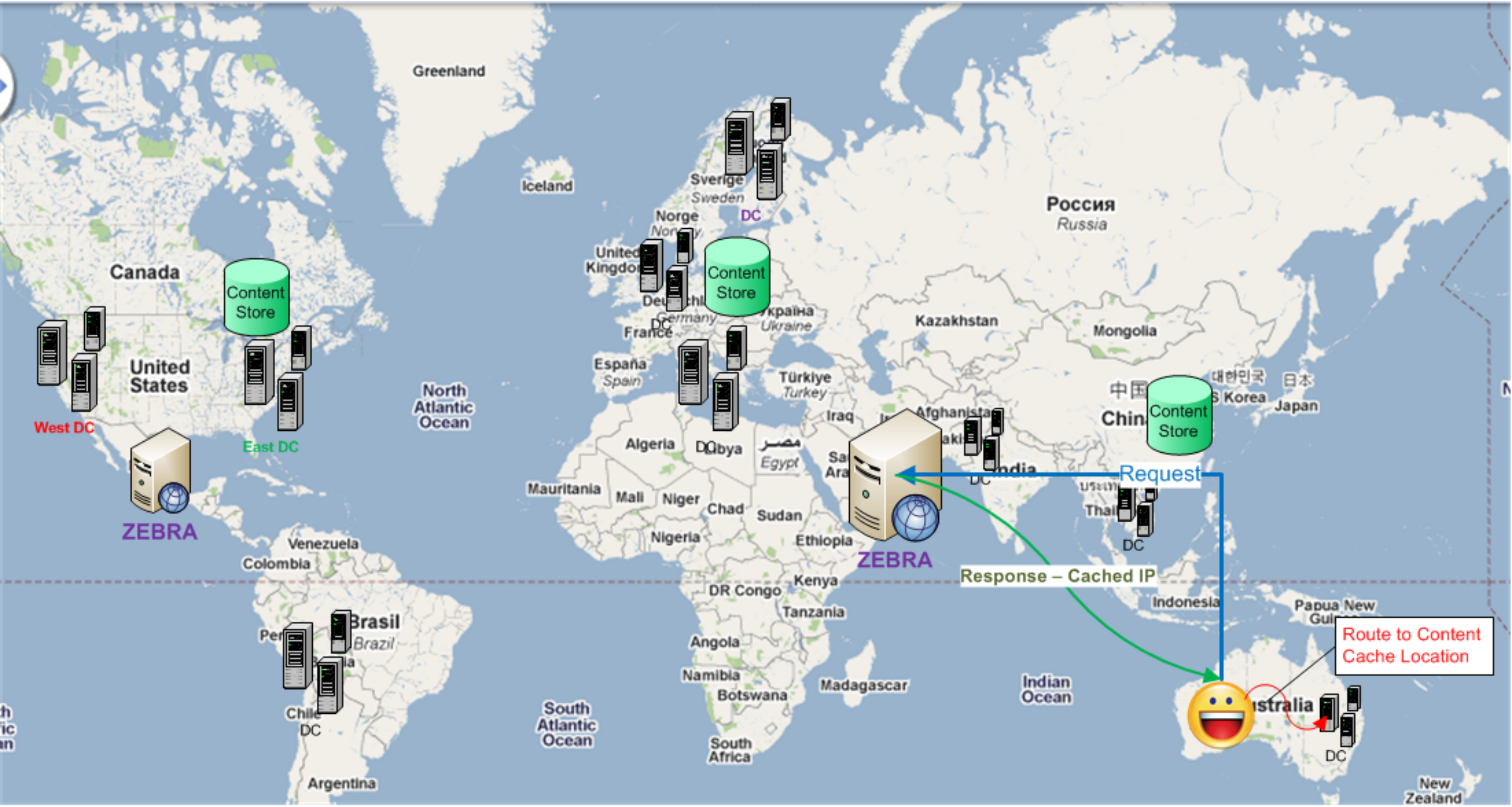
SPOCA and Zebra

- Used in production in a global scenario for web-scale load.
- Shows real world improvements over the simple off-the-shelf solution.
- Implements load balancing, fault tolerance, popular content handling, and efficient cache utilization with a single simple mechanism.

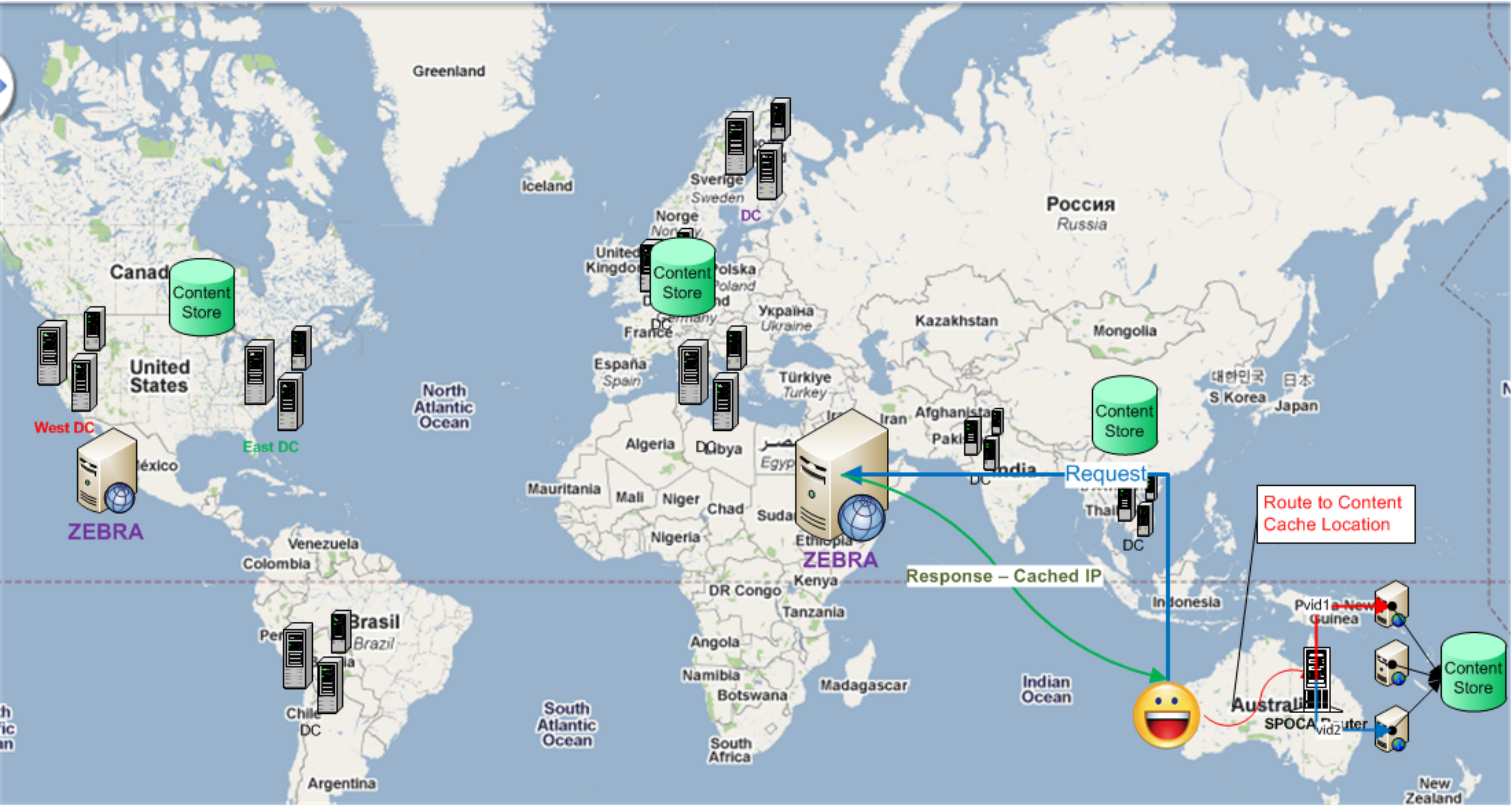
Traditional Approach



Complete Picture



Complete Picture – Inside Data Center

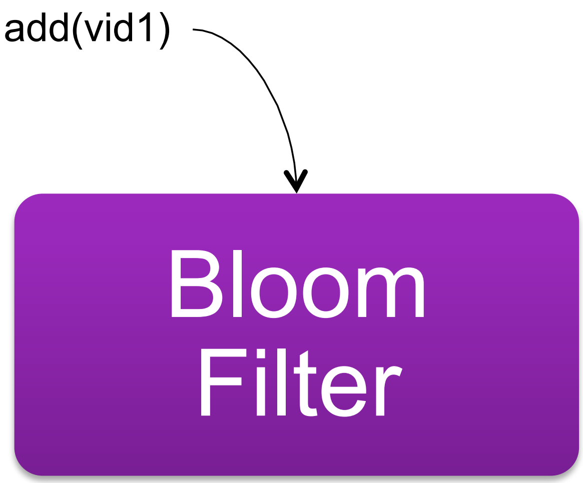


Route to Content Cache Location

Zebra Algorithm

- Handles the geographic component of request routing and content caching
- Based on content popularity, Zebra decides when requests should be routed to content's home locale and when the content should be cached in the nearest locale
- We use bloom filters to determine popularity.

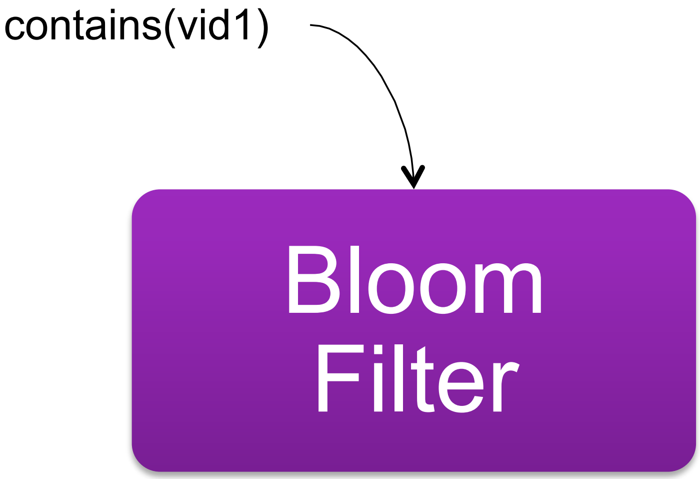
Tracking popularity



add(vid1)

Bloom Filter

Checking Popularity



contains(vid1)

Bloom Filter

What’s the problem here?

- Everything will become popular.
- No way to expire content in bloom filter
- We use a sequence of bloom filters to track popularity.

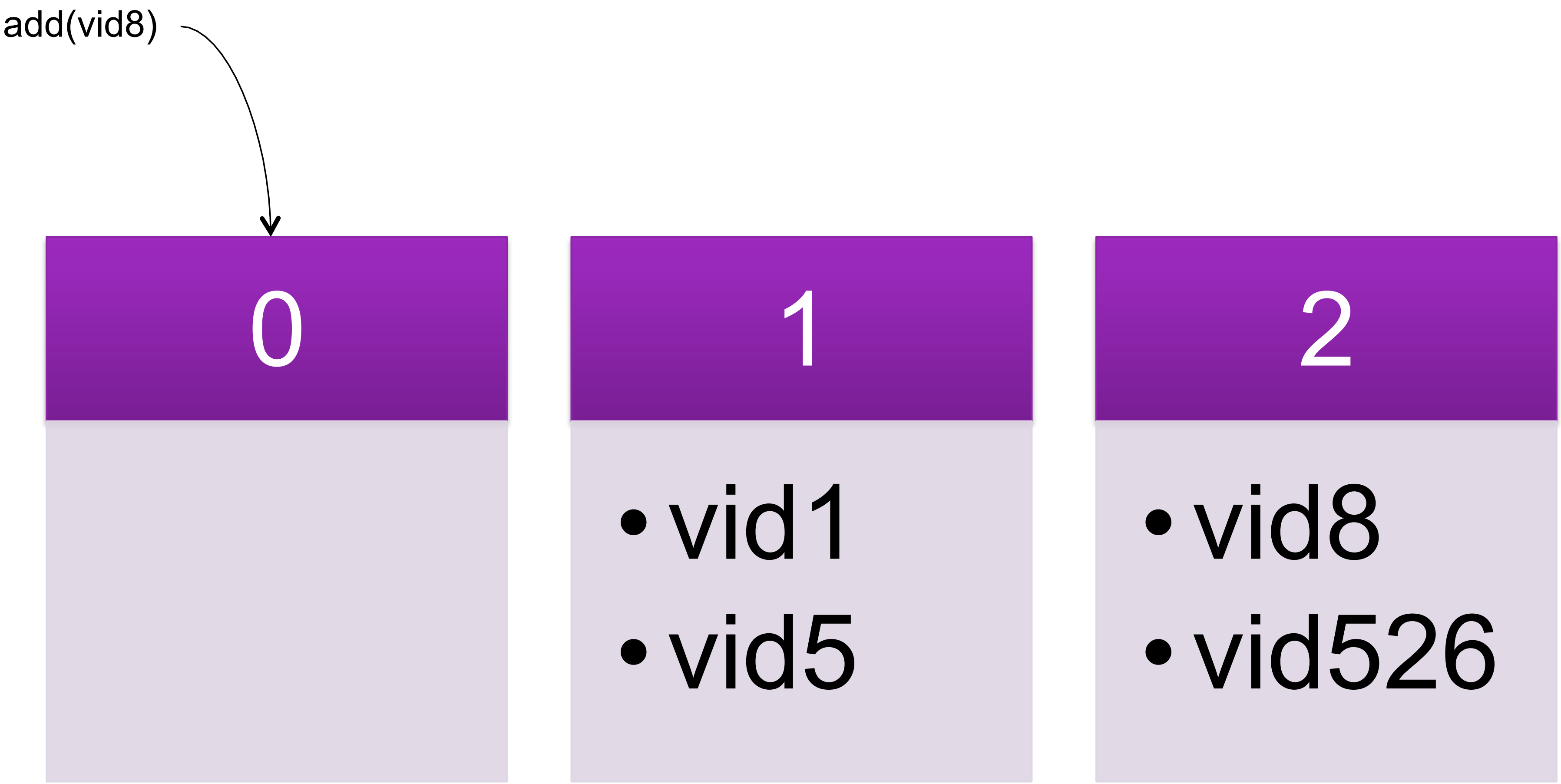
Bloom Filter Representation

0	1	2
• vid1 • vid5	• vid8 • vid526	• vid2 • vid752

Bloom Filter Representation

0	1	2
	• vid1 • vid5	• vid8 • vid526

Bloom Filter Representation



add(vid8)

0

1

- vid1
- vid5

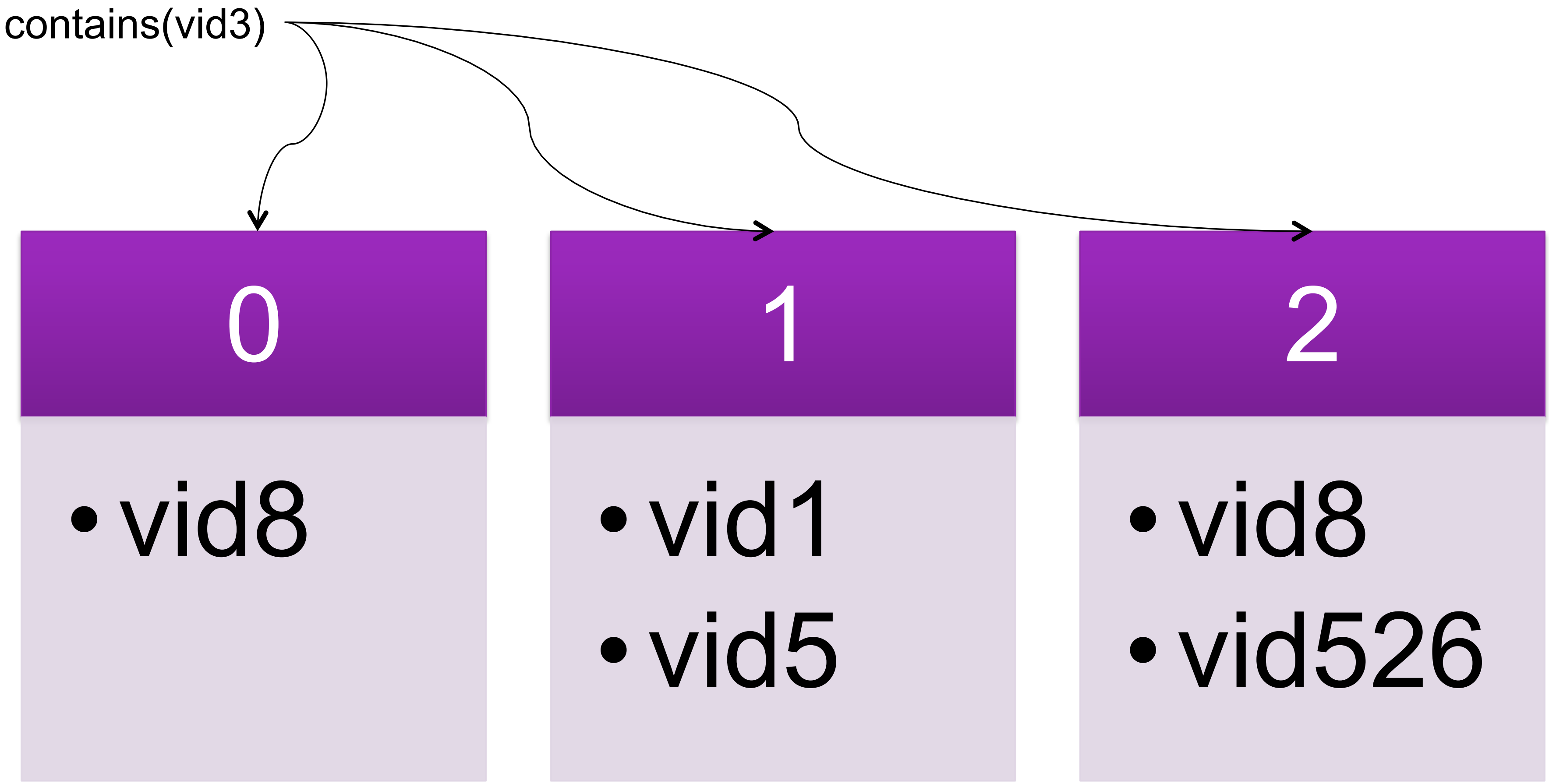
2

- vid8
- vid526

Bloom Filter Representation

0	1	2
• vid8	• vid1 • vid5	• vid8 • vid526

Bloom Filter Representation



contains(vid3)

0

- vid8

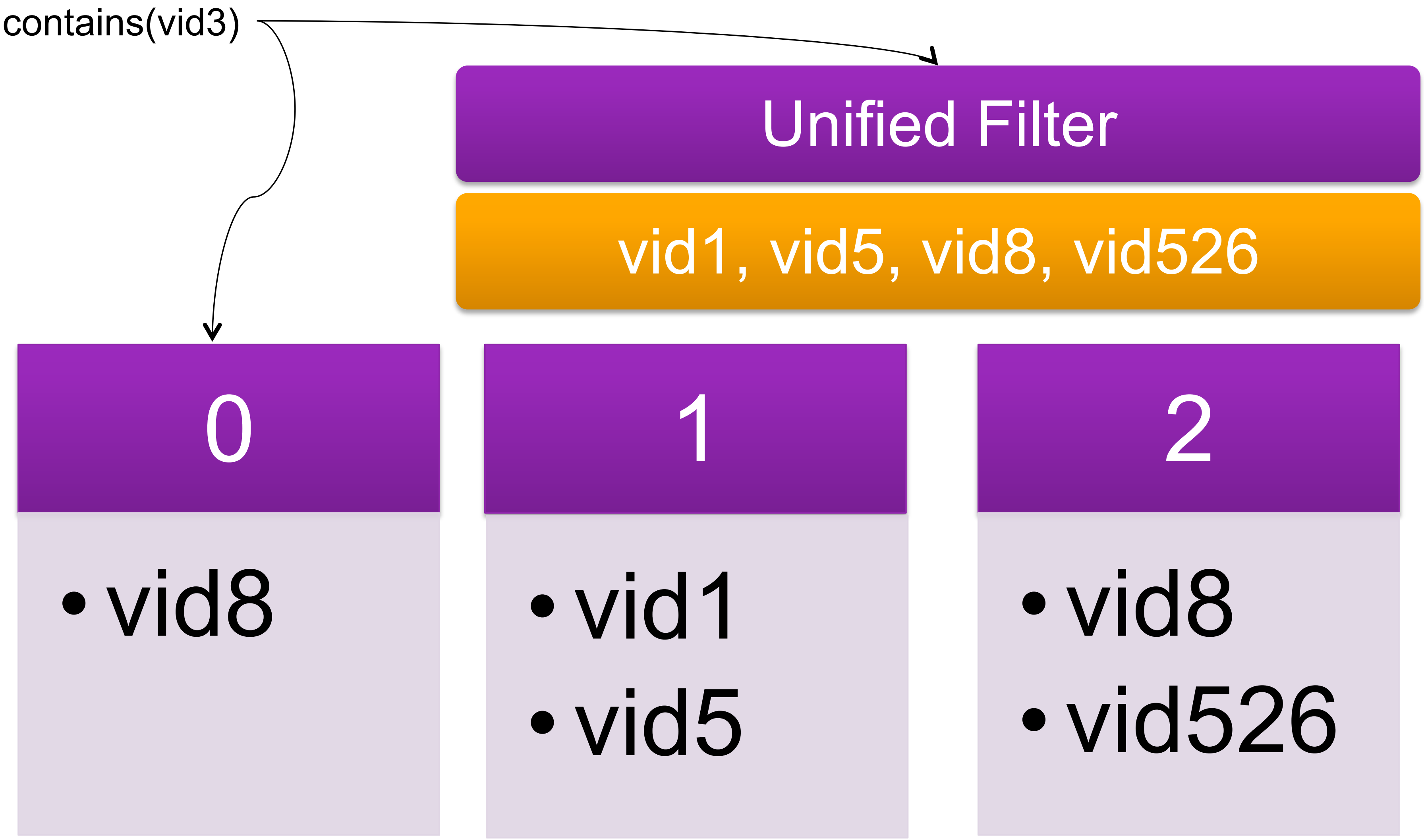
1

- vid1
- vid5

2

- vid8
- vid526

Bloom Filter Representation



contains(vid3)

Unified Filter

vid1, vid5, vid8, vid526

0

- vid8

1

- vid1
- vid5

2

- vid8
- vid526

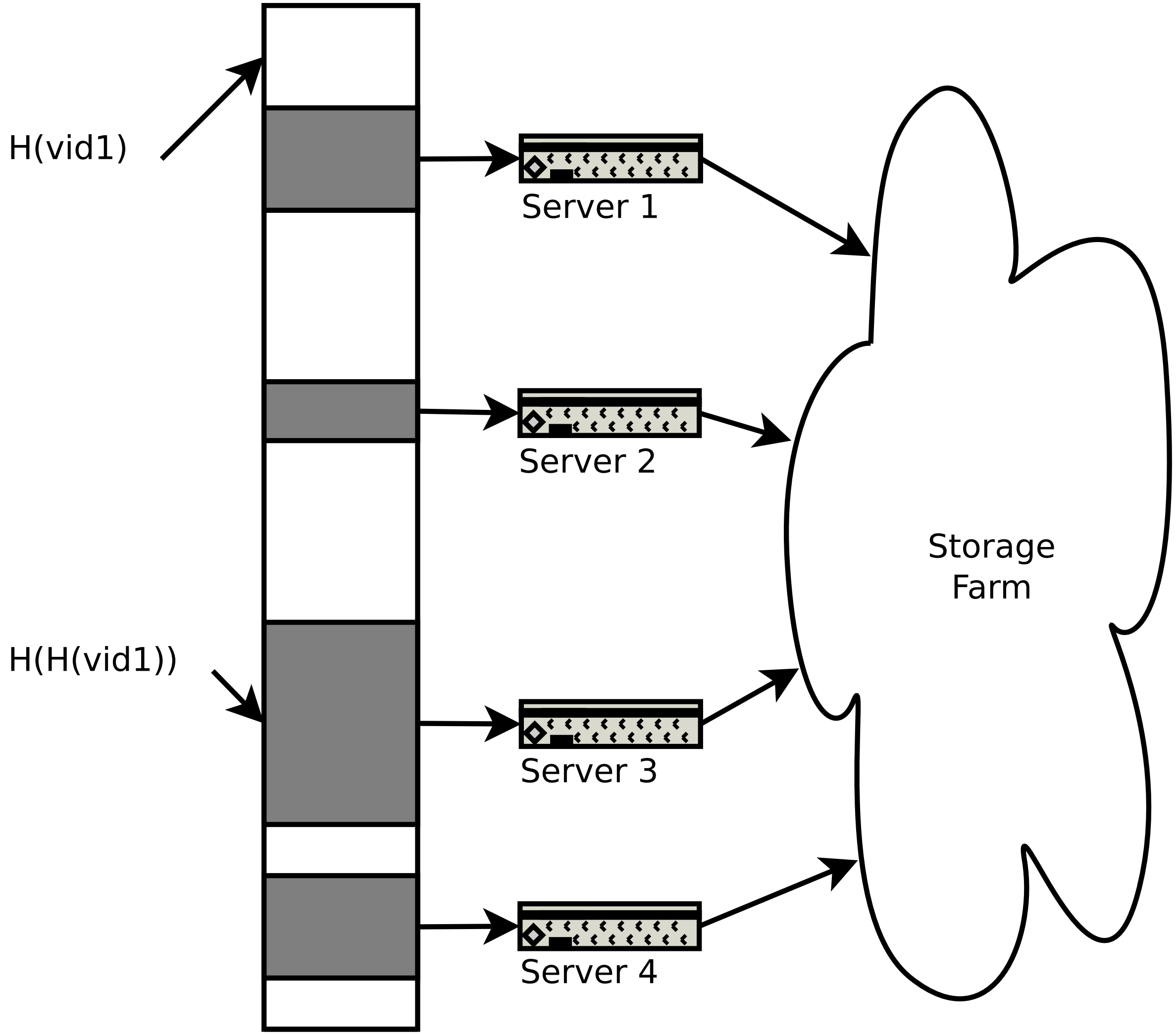
Key Points

- ***Zebra*** determines which serving cluster will handle a given request based on geolocality and popularity.
- ***SPOCA*** determines which front-end server within that cluster will cache and serve the request.

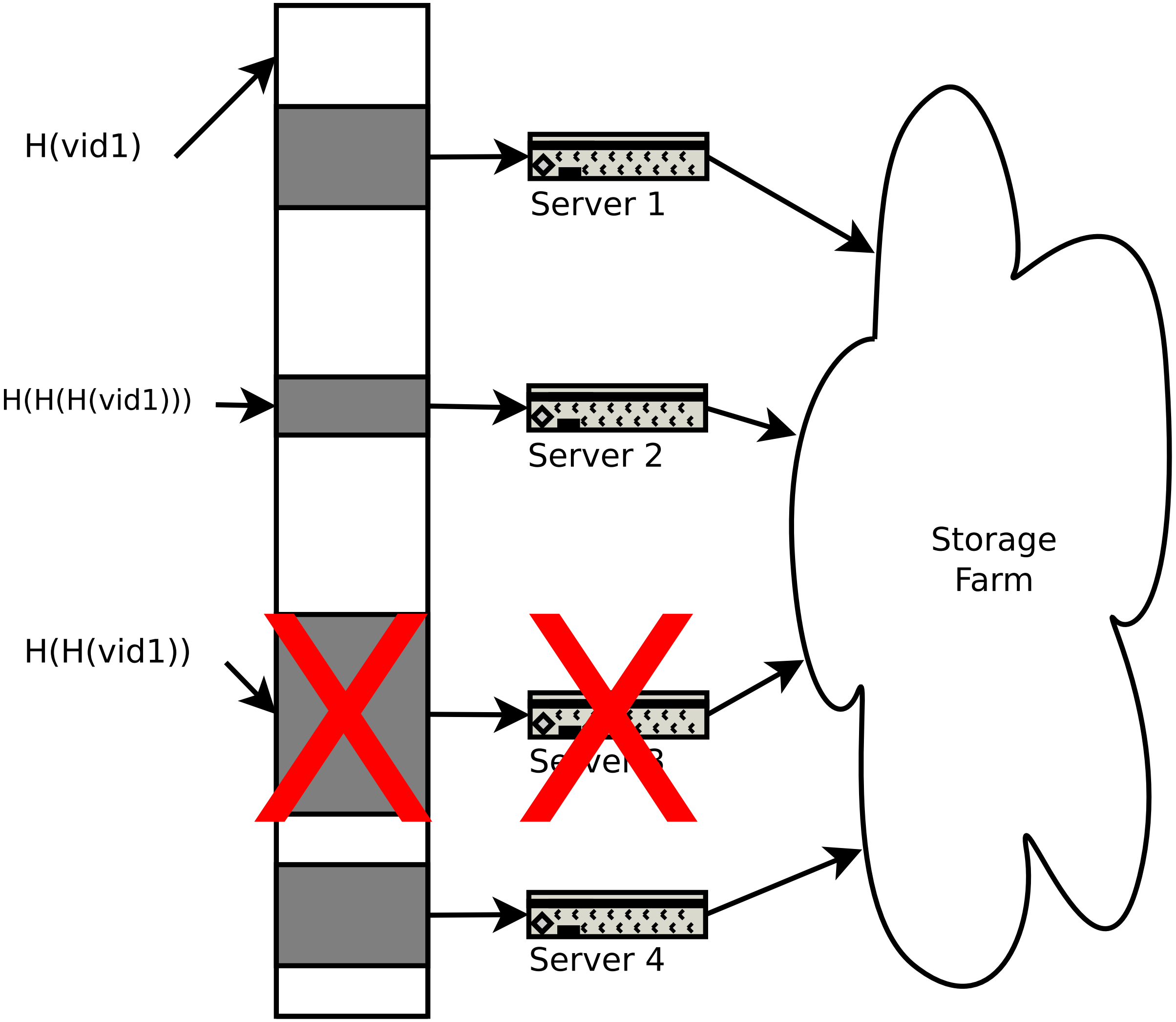
SPOCA Algorithm

- ***Goal***: Maximize cache utilization at the front-end servers.
- Simple content to server assignment function based on a sparse hash space.
- Each front-end server is assigned a portion of the hash space according to its capacity.
- The SPOCA routing function uses a hash function to map names to a point in a hash space.
  - *Input* = the name of the requested content
  - *Output* = the server that will handle the request.
- Re-hashing happens till the result maps to a valid hash space.

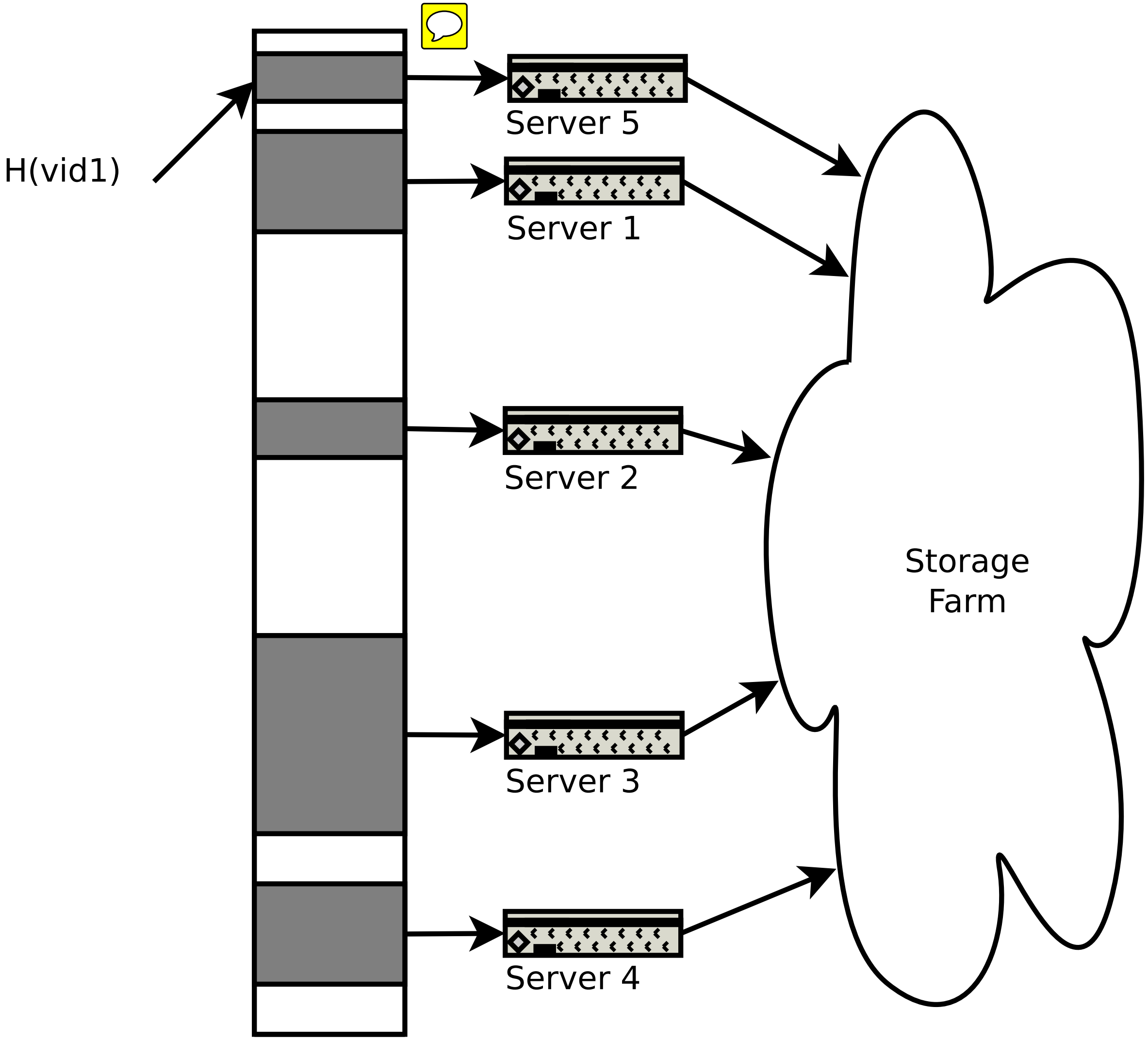
SPOCA hash map example



Failure Handling

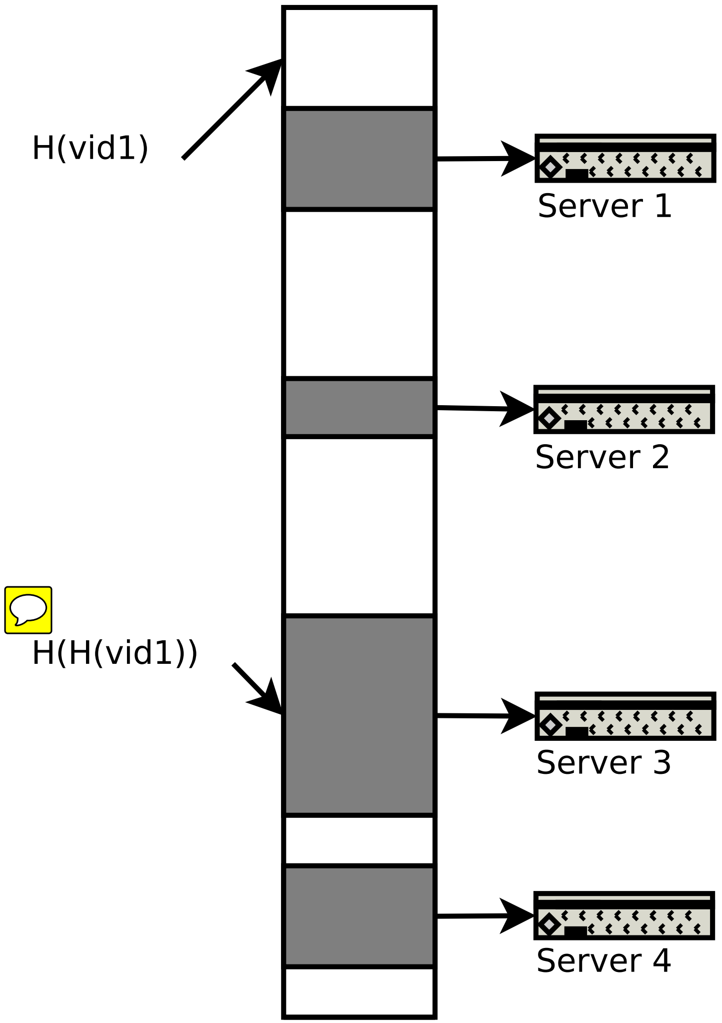


Elasticity


H(vid1)
Server 5
Server 1
Server 2
Server 3
Server 4
Storage Farm

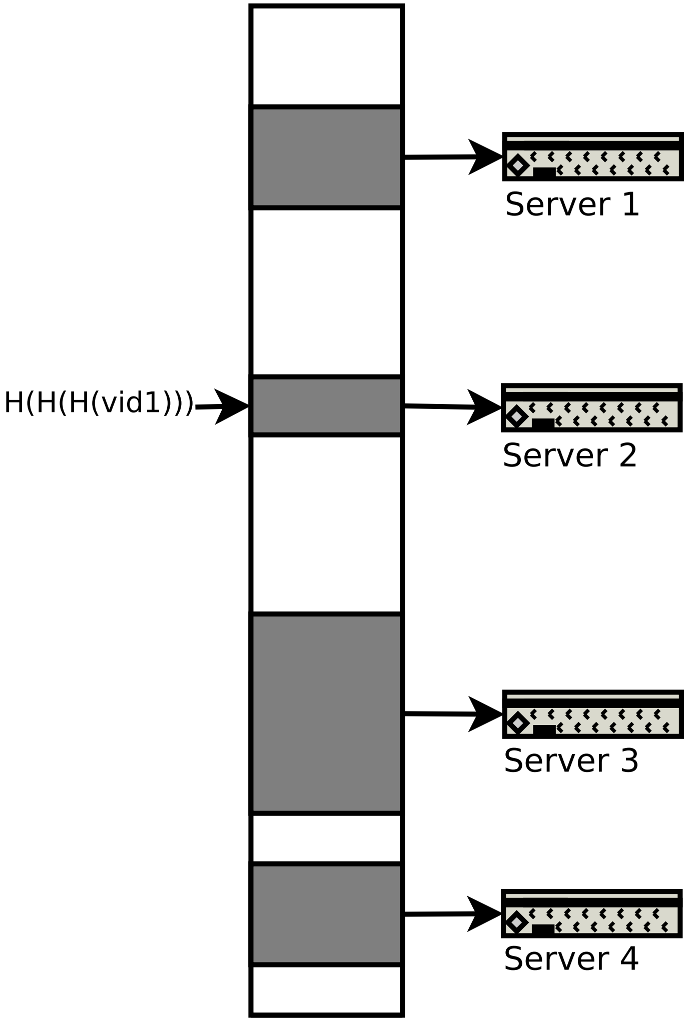
Popular Content

- SPOCA minimizes the number of servers to maximize the aggregate number of cached objects.
- For popular content we need to route requests to multiple front-end servers.
- We store the hashed address of any requested content for a brief popularity window, 150 seconds in our case.
- When the popularity window expires, the stored hash for each object is discarded.



Popularity Window
Before the Request:
{}

Popularity Window
After the Request:
{(vid1, H(H(vid1)))}



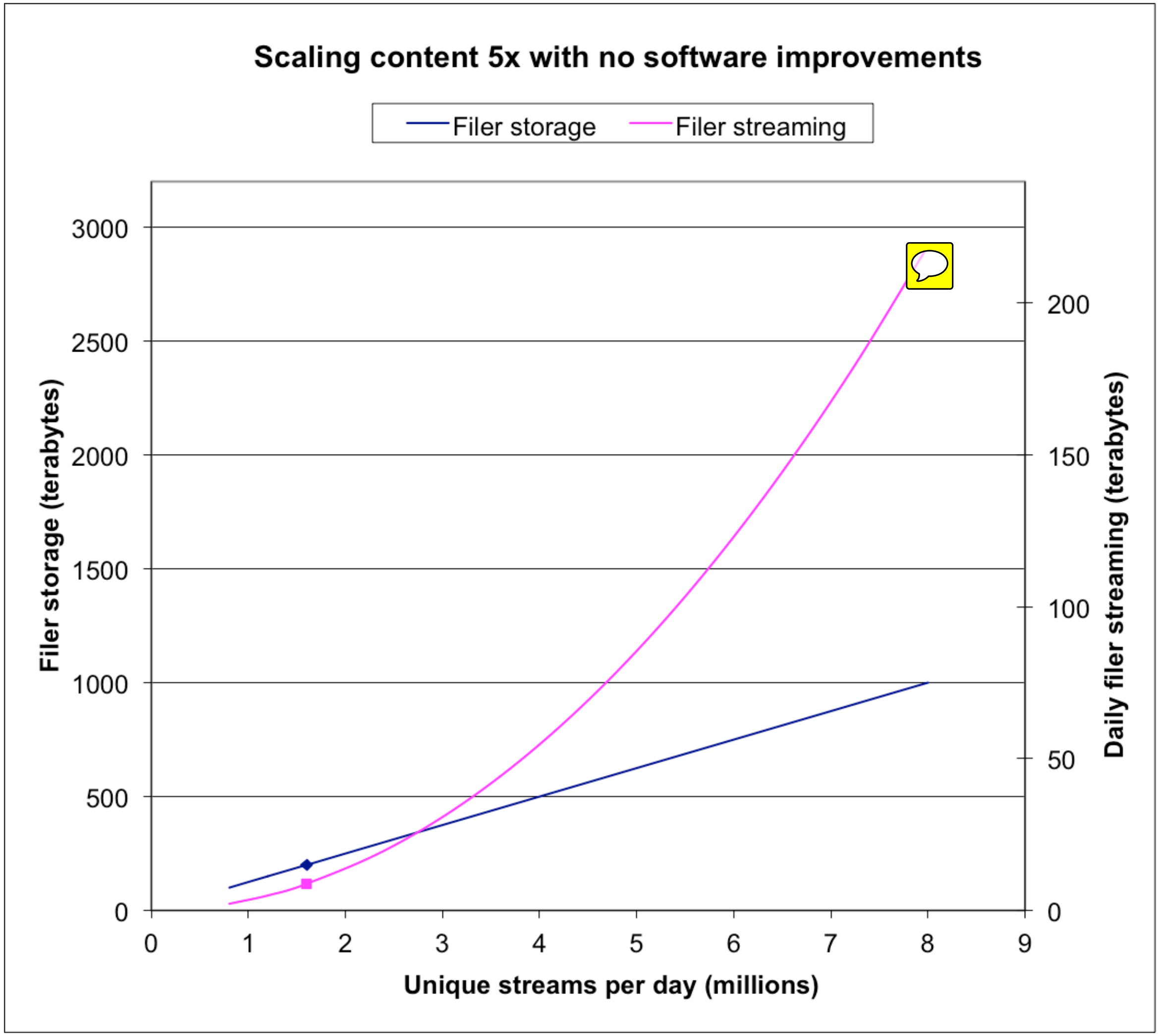
Popularity Window
Before the Request:
{(vid1, H(H(vid1)))}

Popularity Window
After the Request:
{(vid1, H(H(H(vid1))))}

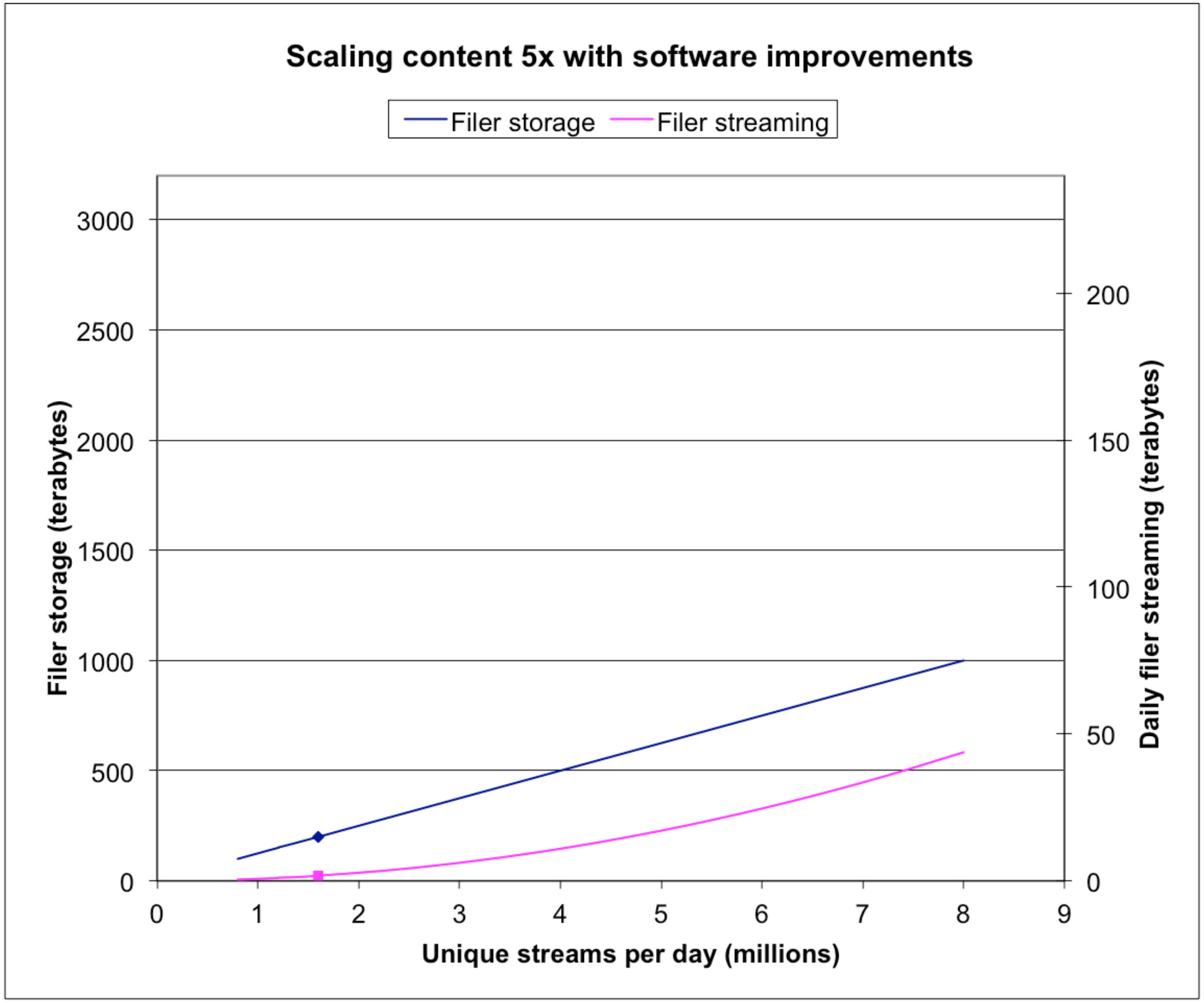
Outline

- Introduction
- Problem Definition
- SPOCA and Requirements
- **Evaluations**
- Conclusion

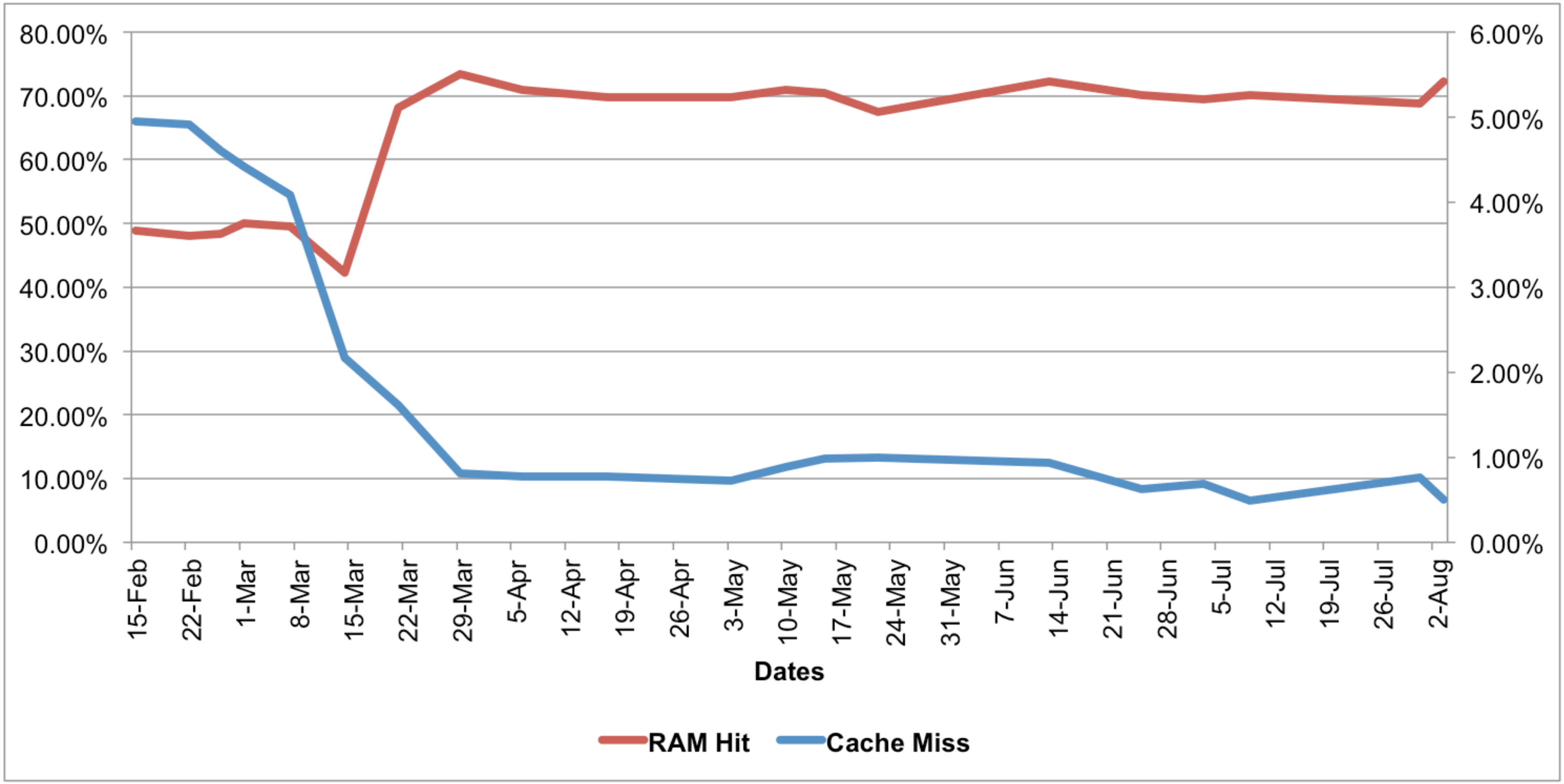
Scaling 5x w/o software improvements



Scaling 5x with software improvements



Memory cache hits



Cache Hit and Misses*

	2/26	3/1	3/5	3/7	3/10	3/14
Download Cache Miss	9.7%	7.2%	4.3%	3.7%	1.8%	0.4%
Download Cache HIT	90.3%	92.8%	95.7%	96.3%	98.2%	99.6%
Flash Cache Miss	21.8%	13.5%	22.0%	14.8%	2.5%	0.7%
Flash RAM hit	57.2%	81.4%	66.1%	71.5%	90.0%	90.1%

* Download and Flash Pools in S1S data center

Conclusion

- Zebra and SPOCA do not have any hard state to maintain or per object meta-data
- Eliminates any per object storage overhead or management, simplifying operations.
- Consolidate content serving into a single pool of servers that can handle files from a variety of different workloads.
- Decouple serving and caching layers.
- Cost savings and end user satisfaction are key success metrics.