Data Structures from the Future: Bloom Filters, Distributed Hash Tables, and More!

Tom Limoncelli,
Google NYC
tlim@google.com

Why am I here?

I have no idea.

Why are you here?

I have 3 theories...

Why are you here?

1. You thought this was the Dreamworks talk.

Why are you here?

2. You’re still drunk from last night.

Why are you here?

3. You can’t manage what you don’t understand.

Overview

1. Hashes & Caches
2. Bloom Filters
3. Distributed Hash Tables (DHTs)
4. Key/Value Stores (NoSQL)
5. Google Bigtable

Disclaimer #1

There will be hand-waving.

The Presence of Slides

!=

"Being Prepared"

Disclaimer #2

You could learn most of this from Wikipedia. Really. Did I mention they’re talking about Shrek in the other room?

Disclaimer #3

My LISA 2008 talk also conflicted with a talk from Dreamworks.

To understand this talk, you must understand:

- Hashes
- Caches

Hashes

What is a Hash?

A fixed-size summary of a large amount of data.

Checksum

- Simple checksum:
  - Sum the byte values. Take the last digit of the total.
  - Pros: Easy. Cons: Change order, same checksum.
- Improvement: Cyclic Redundancy Check
  - Detects change in order.

Hash

- “Cryptographically Unique”
  - Difficult to generate 2 files with the same MD5 hash
  - Even more difficult to make a “valid second file”:
    - The second file is a valid example of the same format. (i.e. both are HTML files)

How do crypto hashes work?

"It works because of math."

Matt Blaze, Ph.D

Reversible/Irreversible Functions

[ ] + 105 = 205

[ ] mod 10 = 4

Some common hashes

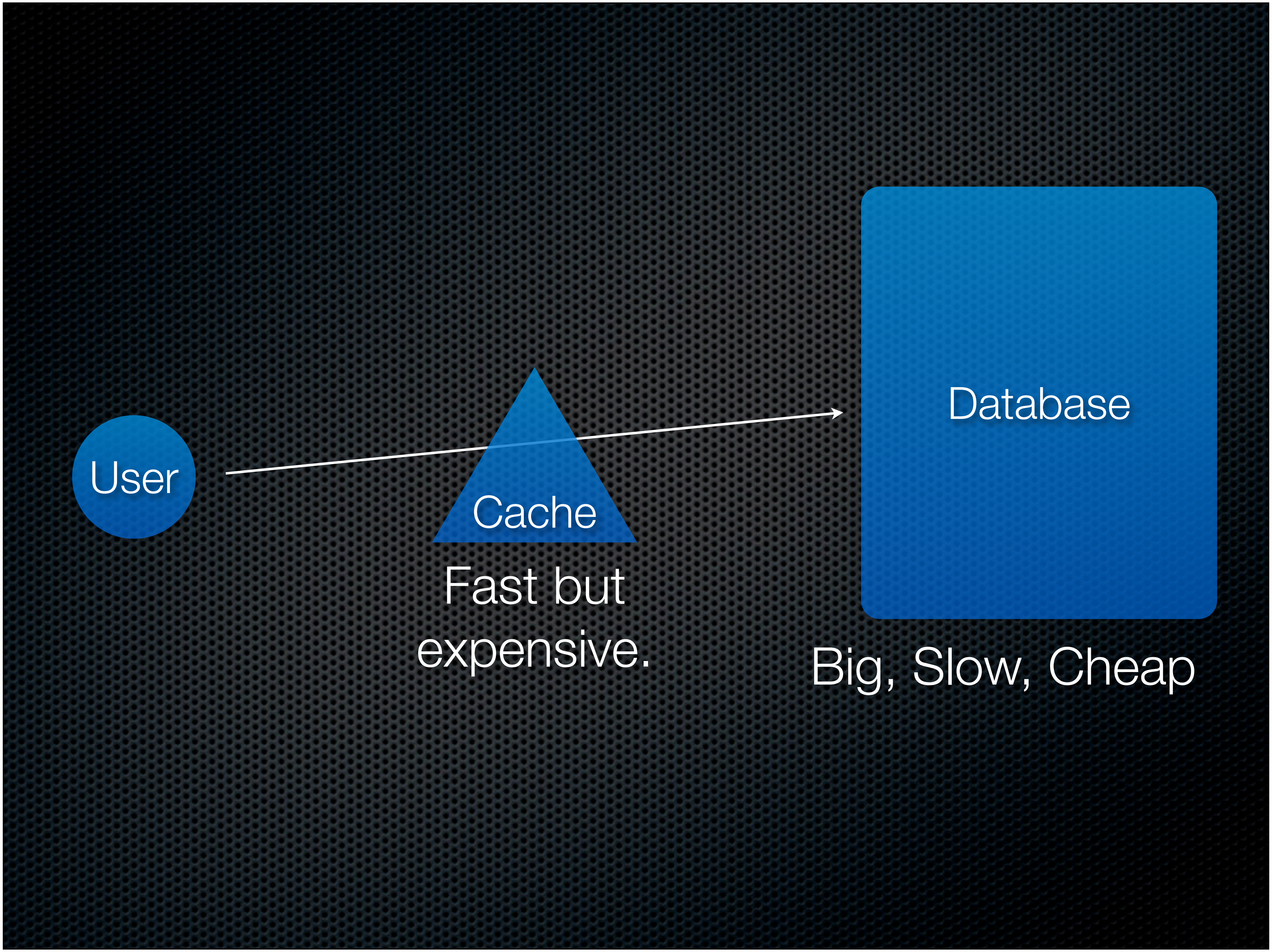
MD4	
MD5	
SHA1	
SHA2	
AES-Hash	

Hashes

Caches

What is a Cache?

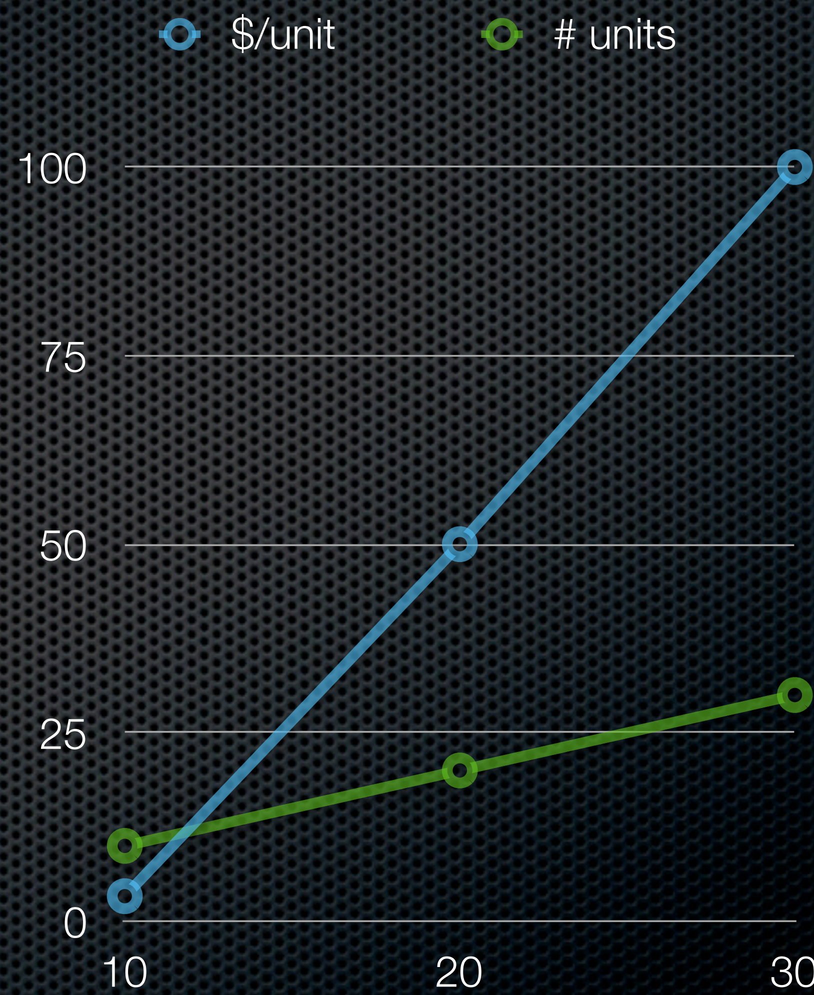
- Using a small/expensive/fast thing to make a big/cheap/slow thing faster.

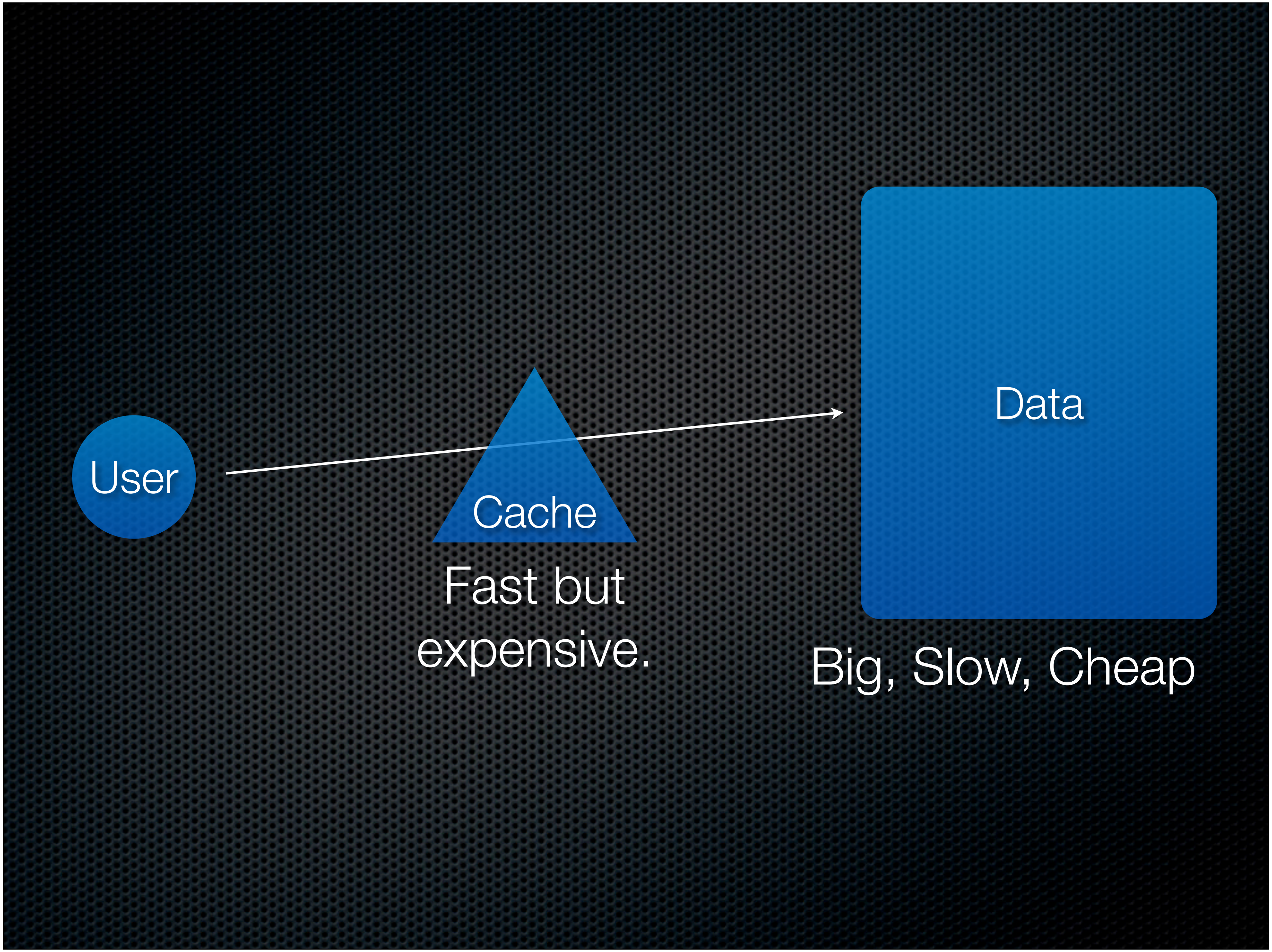

User
Cache
Fast but expensive.
Database
Big, Slow, Cheap

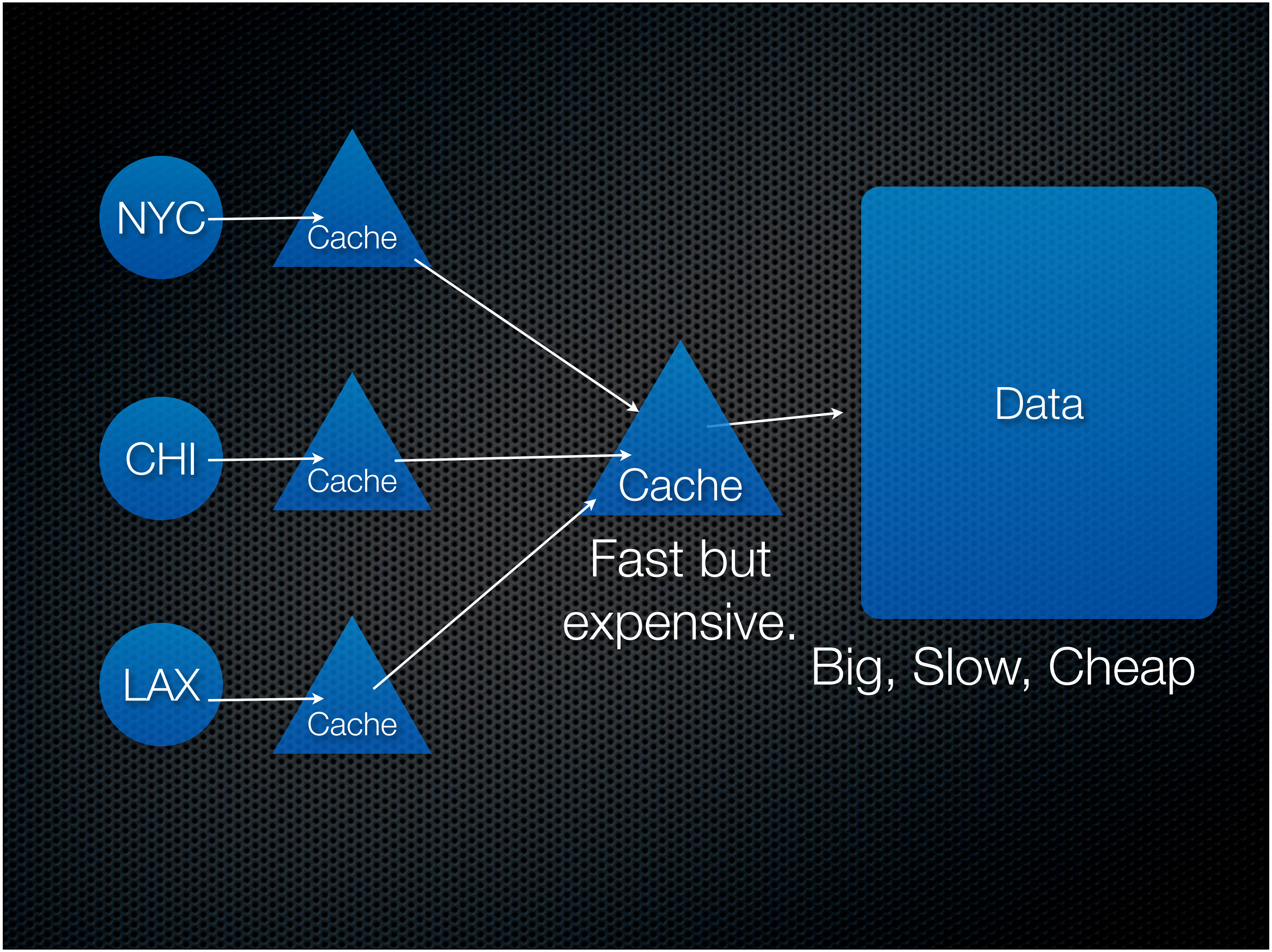
- Metric used to grade?
  - The “hit rate”: hits / total queries
- How to tune?
  - Add additional storage
  - Smallest increment: Result size.

- Suppose cache is X times faster
  - ...but Y times more expensive
- Balance cost of cache vs. savings you can get:
  - Web cache achieves 30% hit rate, costs $/MB
  - 33% of cachable traffic costs $/MB from ISP.
    - What about non-cachable traffic?
    - What about query size?

- Value of next increment is less than the previous:
  - 10 units of cache achieves 30% hit rate
  - +10 units, hit rate goes to 32%
  - +10 more units, hit rate goes to 33%




User
Cache
Fast but expensive.
Data
Big, Slow, Cheap


NYC
CHI
LAX
Cache
Cache
Cache
Cache
Fast but expensive.
Data
Big, Slow, Cheap

	Simple Cache	NCACHE	Intelligent
Add new data?	Ok	Not found	Ok
Delete data?	Stale	Stale	Ok
Modify data?	Stale	Stale	Ok

Caches

Bloom Filters

What is a Bloom Filter?

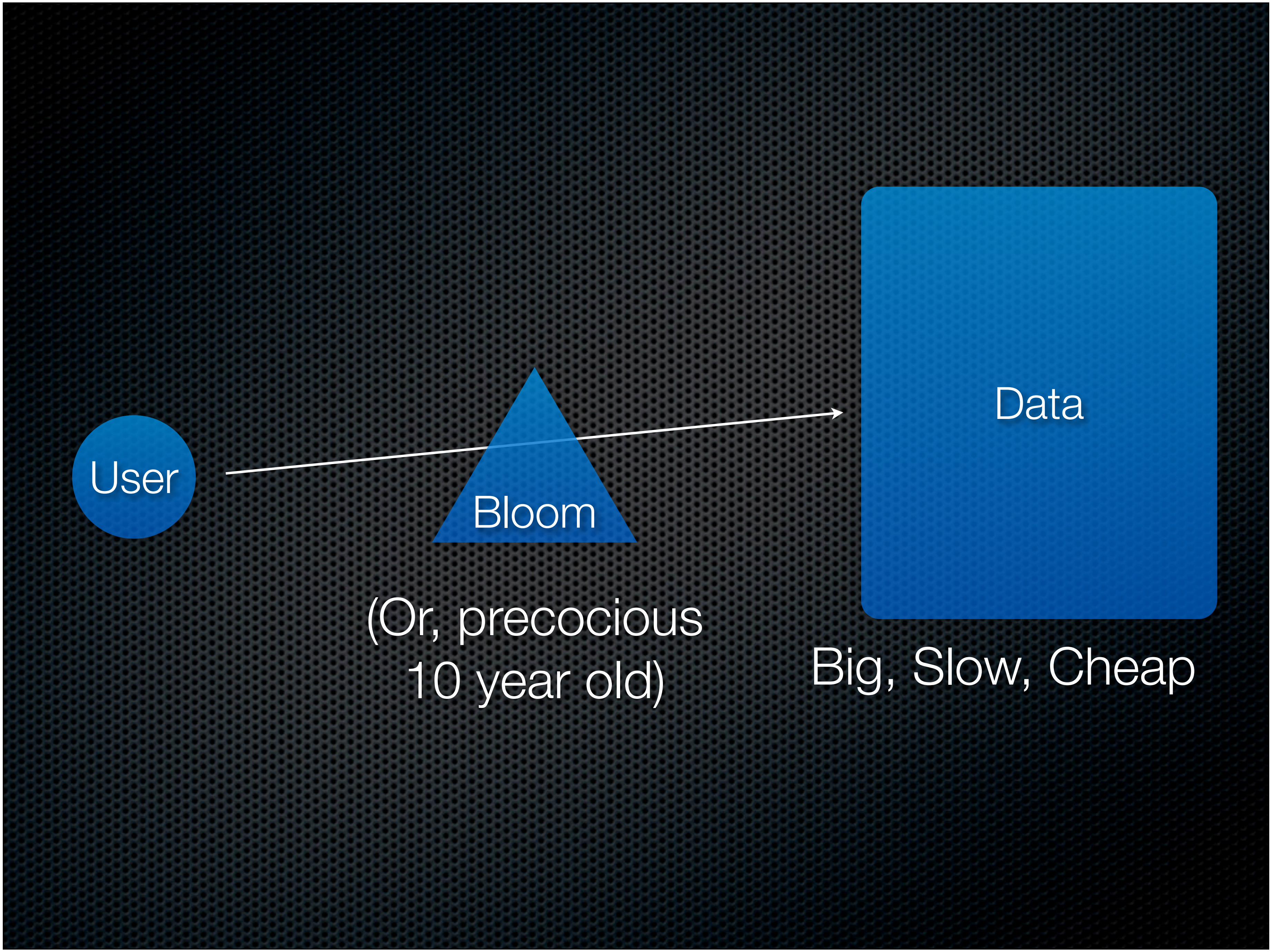
- Knowing when NOT to waste time seeking out data.
- Invented in Burton Howard Bloom in 2070

What is a Bloom Filter?

- Knowing when NOT to waste time seeking out data.
- Invented in Burton Howard Bloom in 1970

I invented Bloom Filters when I was 10 years old.


C
D
B
ZX AAA Road 695-8311
Travel 594-07oo
ROLODEX


User
Bloom
Data
(Or, precocious
10 year old)
Big, Slow, Cheap

Using the last 3 bits of hash:

```
Olson 000100001111
Polk  000000000011
Smith 001011101110
Singh 001000011110
```

```
000
001
010
011
100
101
110
111
```

Using the last 3 bits of hash:

```
Olson 000100001111
Polk  000000000011
Smith 001011101110
Singh 001000011110

Lakey 111110000000
Baird 001011011111
Camp  001101001010
Johns 010100010100
Burd  111000001101
Bloom 110111000011
```

```
000 ✓
001
010 ✓
011 ✓
100 ✓
101 ✓
110 ✓
111 ✓
```

Using the last 4 bits of hash:

```
Olson 000100001111
Polk  000000000011
Smith 001011101110
Singh 001000011110

Lakey 111110000000
Baird 001011011111
Camp  001101001010
Johns 010100010100
Burd  111000001101
Bloom 110111000011
```

0000 ✓	1000
0001	1001
0010	1010 ✓
0011 ✓	1011
0100 ✓	1100
0101	1101 ✓
0110	1110 ✓
0111	1111 ✓

7/16 = 44%

bits of hash		# Entries	Bytes	<25% 1’s
3	2^3	8	1	2
4	2^4	16	2	4
5	2^5	32	4	8
6	2^6	64	8	16
7	2^7	128	16	32
8	2^8	256	32	64
20	2^8	1048576	131072	262144
24	2^32	16777216	2M	4.1 Million
32	2^64	4294967296	512M	1 Billion

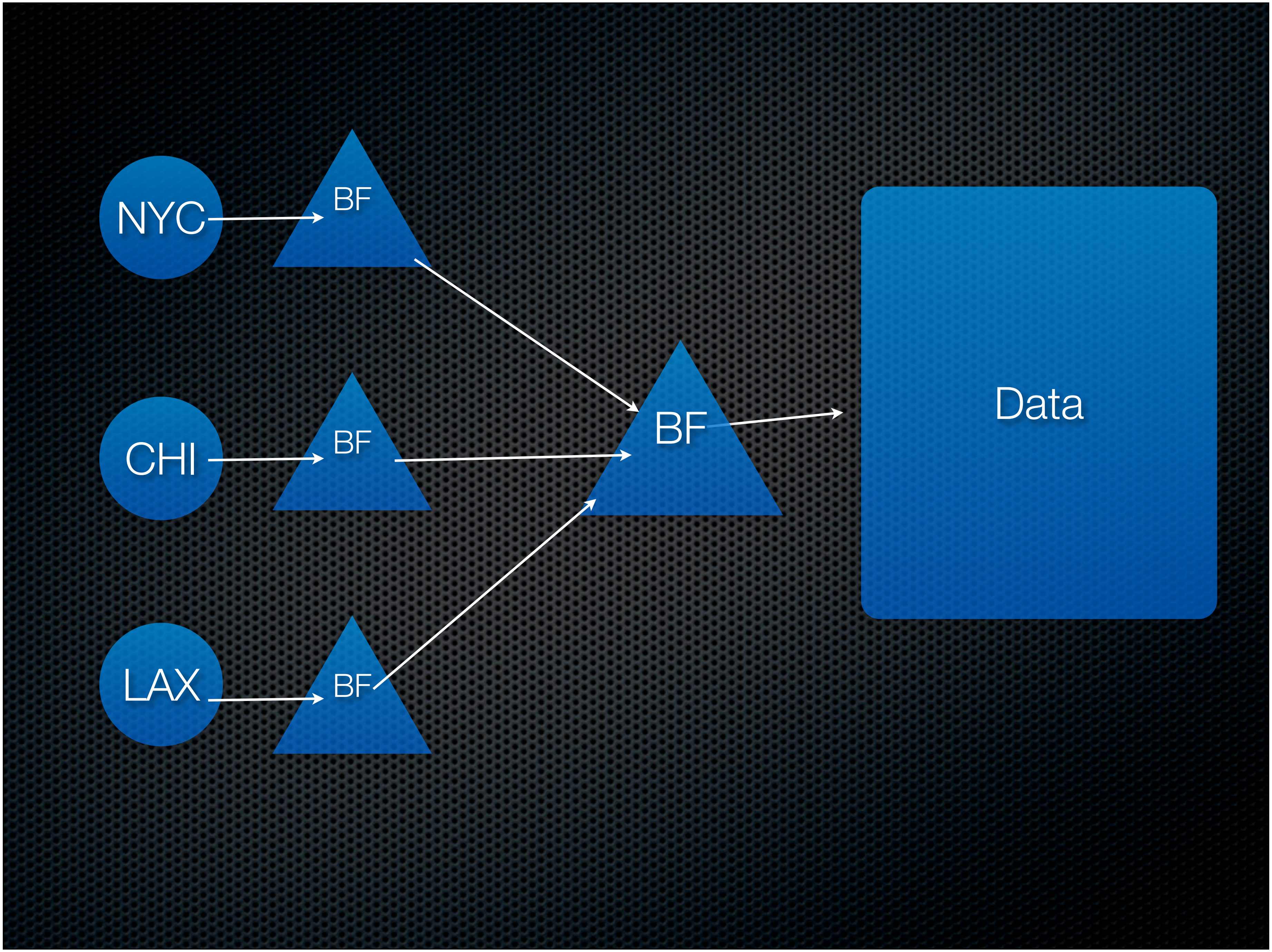
- When to use? Sparse Data
- When to tune: When more than x% are “1”
- Pitfall: To resize, must rescan all keys.

- Minimum Increment doubles memory usage:
  - Each increment is MORE USEFUL than the previous.
  - But exponentially MORE EXPENSIVE!

Bloom Filter sample uses

- Databases: Accelerate lookups of indices.
- Simulations: Often having, big, sparse databases.
- Routers: Speeds up route table lookups.

Distributed Bloom Filters?


NYC
BF
CHI
BF
LAX
BF
BF
Data

What if your Bloom Filter is out of date?

- New data added: BAD. Clients may not see it.
- Data changed: Ok
- Data deleted: Ok, but not as efficient.

How to perform updates?

- Master calculates bitmap once.
- Sends it to all clients
- For a 20-bit table, that’s 130K. Smaller than most GIFs!

- Reasonable for daily, hourly, updates.

Terminal — bash — 80×24

```
$
$
$
$
$
$
$ cd ~/Library/Application\ Support/Google/Chrome
$ ls -lh *Bloom*
-rw-r--r--@ 1 tlim  5000   6.2M Nov 10 15:05 Safe Browsing Bloom
-rw-------@ 1 tlim  5000   1.8M Nov 10 15:05 Safe Browsing Bloom Filter 2
-rw-r--r--@ 1 tlim  5000     0B Nov 10 17:02 Safe Browsing Bloom_new
$ 
```

Big Bloom Filters often use 96, 120 or 160 bits!

Bloom Filters

Hash Tables

What is a Hash Table?

- It's like an array.
- But the index can be anything "hashable".

Hash tables

- Perl hash:
  - `$thing{’b’} = 123;`
  - `$thing{‘key2’} = ”value2”;`
  - `print $thing{‘key2’};`
- Python Dictionary or “dict”:
  - `thing = {}`
  - `thing[‘b’]  = 123`
  - `thing[‘key2’] = “value2”`
  - `print thing[‘key2’]`

hash('cow') = 78f825

hash('bee') = 92eb5f hash('sheep') = 92eb5f

Bucket	Data
78f825	("cow", "moo")
92eb5f	("bee", "buzz") , ('sheep', 'baah')

Hash Tables

Distributed Hash Tables (DHTs)

What is a DHT?

A hash table so big you have to spread it over multiple of machines.

Wouldn't an infinitely large hash table be awesome?

Web server

- lookup(url) -> page contents
  - ‘index.html’ -> ‘<html><head>...’
  - ‘/images/smile.png’ -> 0x4d4d2a...

Virtual Web server

- lookup(vhost/url) -> page contents
  - ‘cnn.com/index.html’ -> ‘<html><he...’
  - ‘time.com/images/smile.png’ -> 0x4d...

Virtual FTP server

- lookup(host:path/file) -> file contents
  - ‘ftp.gnu.org:public/gcc.tgz’
  - ‘ftp.usenix.org:public/usenix.bib’

NFS server

- lookup(host:path/file) -> file contents
  - ‘srv1:home/tlim/Documents/foo.txt’ -> file contents
  - ‘srv2:home/tlim/TODO.txt‘ -> file contents

Usenet (remember usenet?)

- lookup(group:groupname:artnumber) -> article
  - lookup('group:comp.sci.math:987765')
- lookup(id:message-id) -> pointer
  - lookup('id:foo-12345@uunet') -> 'group:comp.sci.math:987765'

IMAP

- lookup(‘server:user:folder:NNNN’)

  -> email message

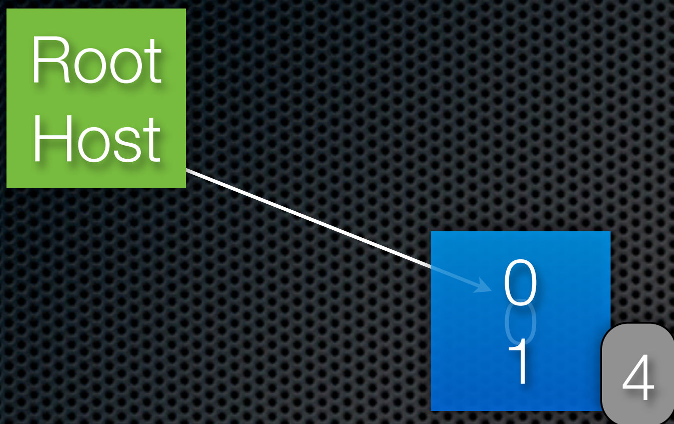
Our DVD Collection

- hash(disc image) -> disc image
- How do I find a particular disk?
  - Keep a lookup table of name -> hash
  - Benefit: Two people with the same DVD? It only gets stored once.

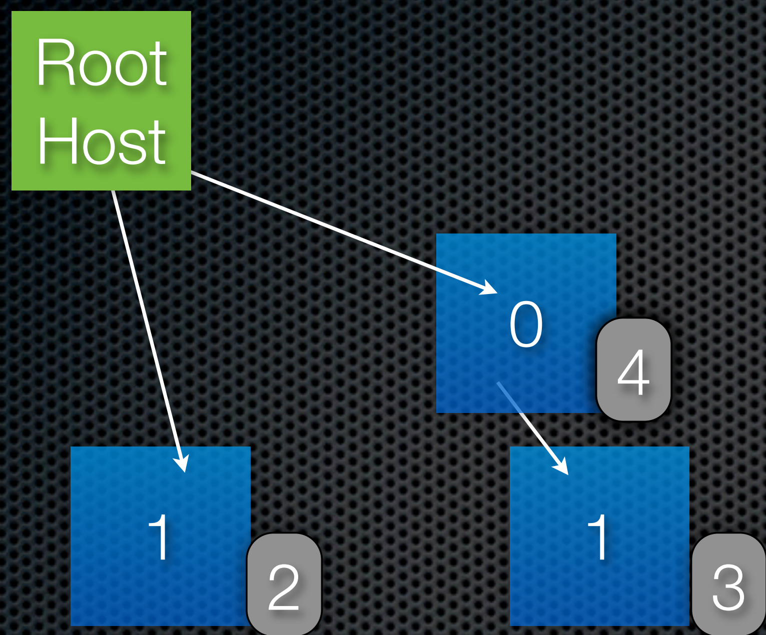
How would this work?

Load it up!

0100100111011001
0001000101100011
1001110100110111
1110001010010110
0011000000000100

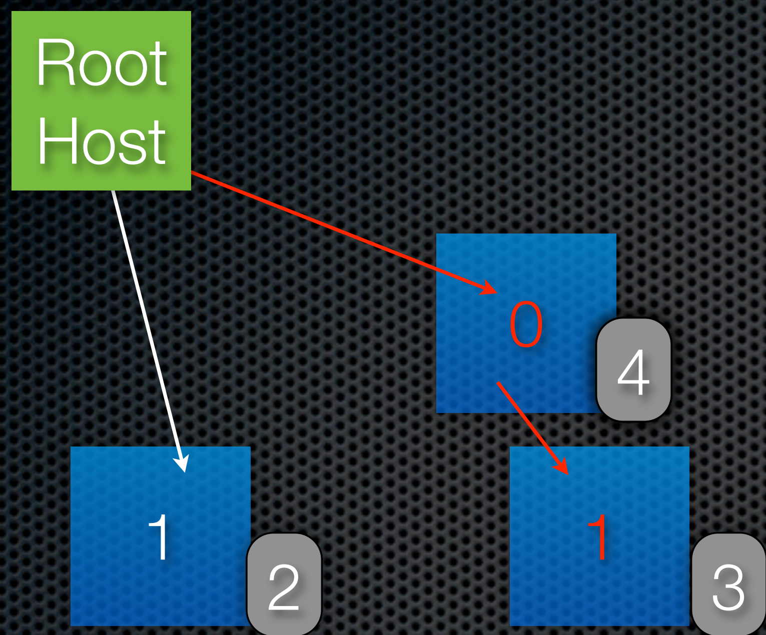


Split



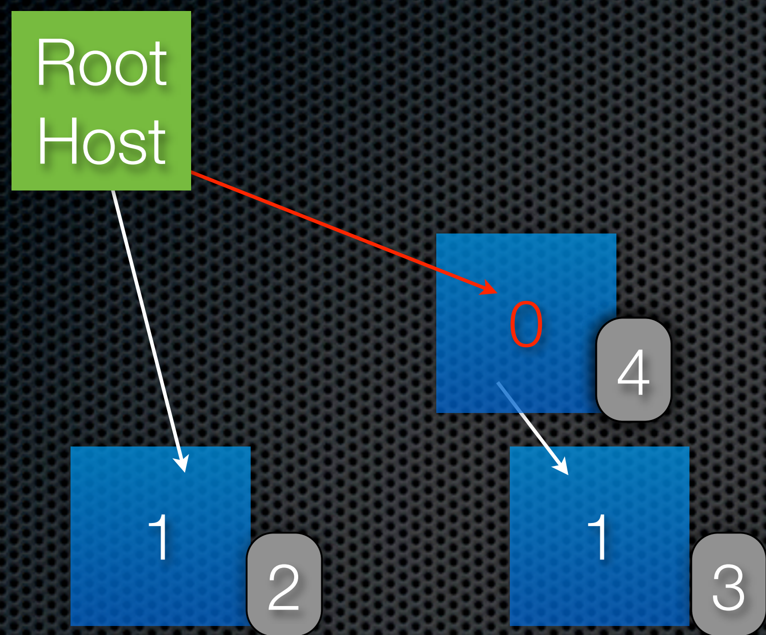
0100100111011001
0001000101100011
1001110100110111
1110001010010110
0011000000000100
0110000111101100
0100000001101011
0010111000000001
0011000101111000

’01....’



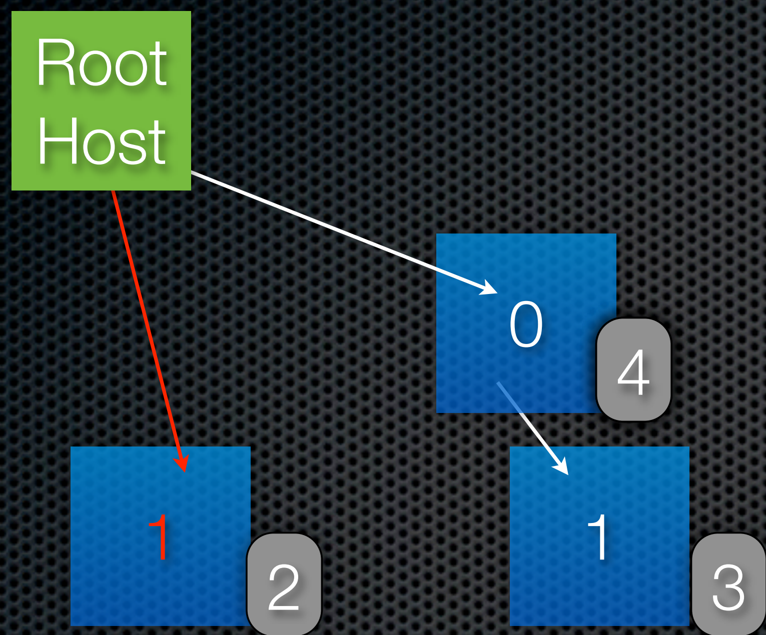
0100100111011001
0001000101100011
1001110100110111
1110001010010110
0011000000000100
0110000111101100
0100000001101011
0010111000000001
0011000101111000

‘0...’



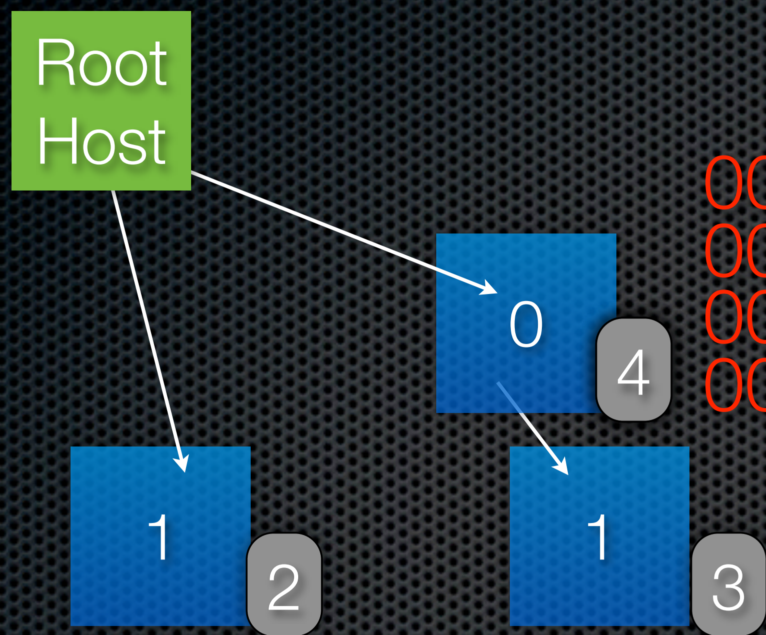
0100100111011001
0001000101100011
1001110100110111
1110001010010110
0011000000000100
0110000111101100
0100000001101011
0010111000000001
0011000101111000

‘1...’



0100100111011001
0001000101100011
1001110100110111
1110001010010110
0011000000000100
0110000111101100
0100000001101011
0010111000000001
0011000101111000

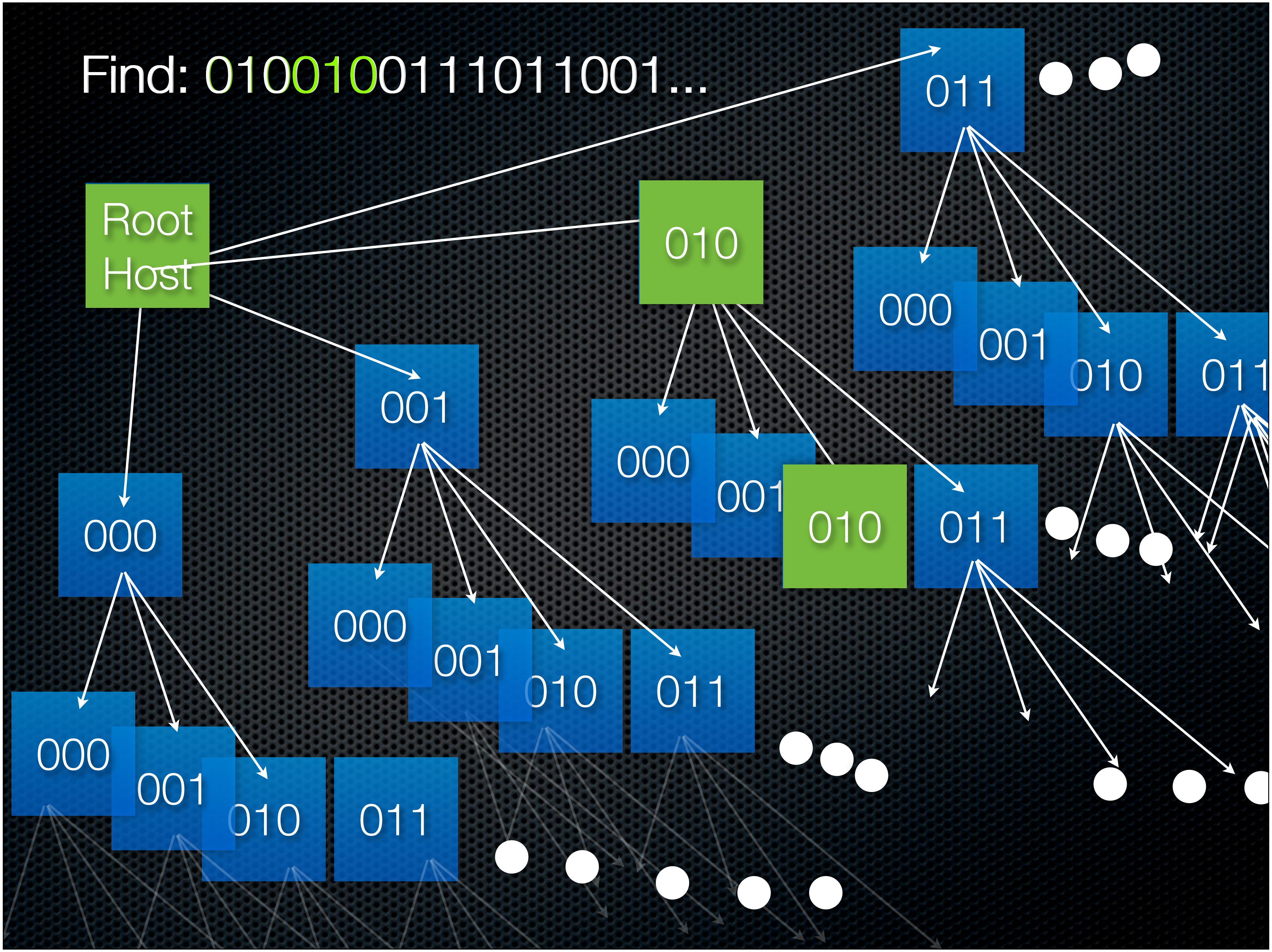
Split



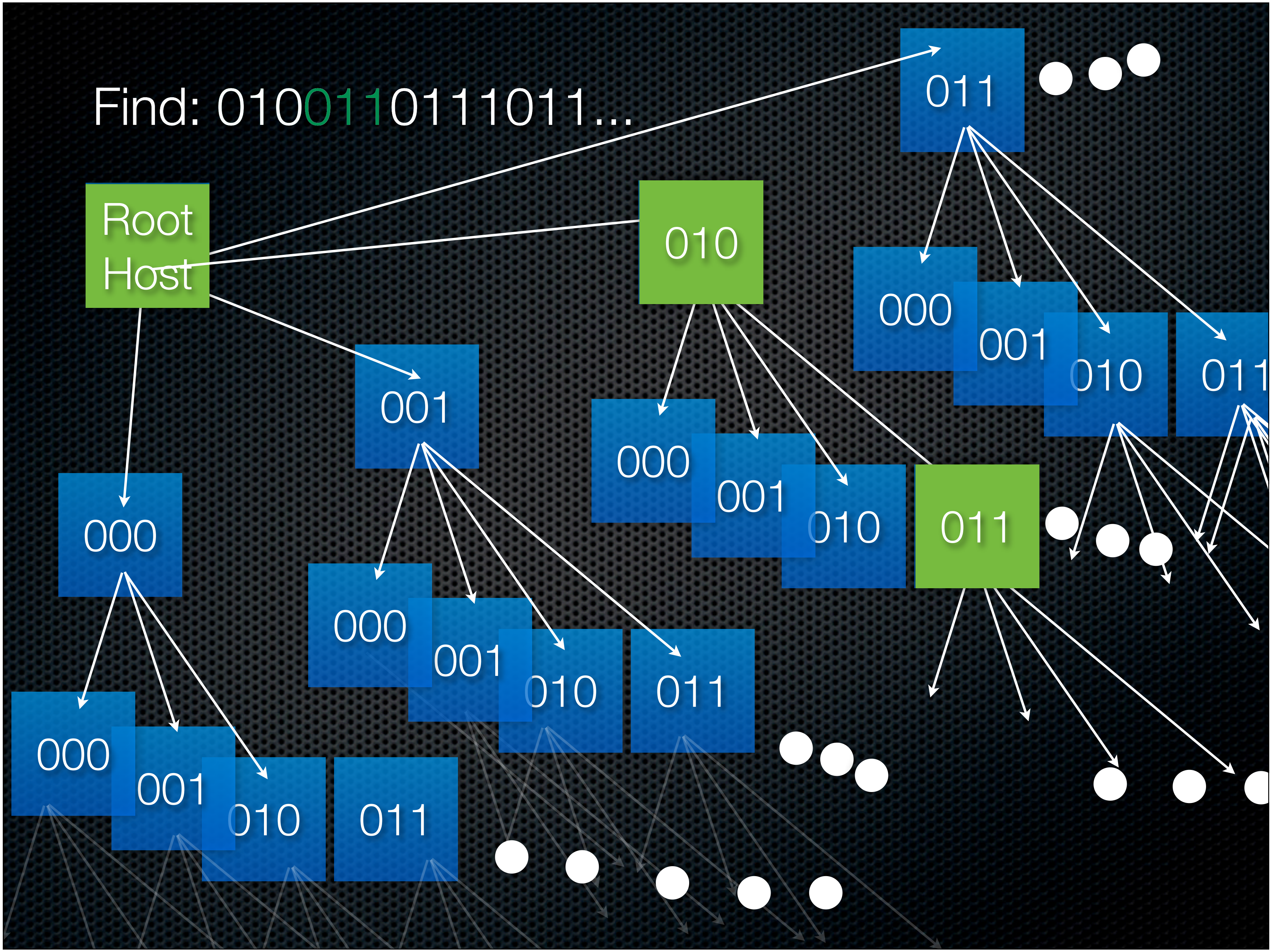
0001000101100011
0011000000000100
0010111000000001
0011000101111000

0100100111011001
0110000111101100
0100000001101011

1001110100110111
1111000101001011 0


Find: 010010011101101001...
Root Host
010
011
001
000
000
001
010
011
000
001
010
011
000
001
010
011
000
001
010
011
000
001
010
011

Find: 0100110111011...

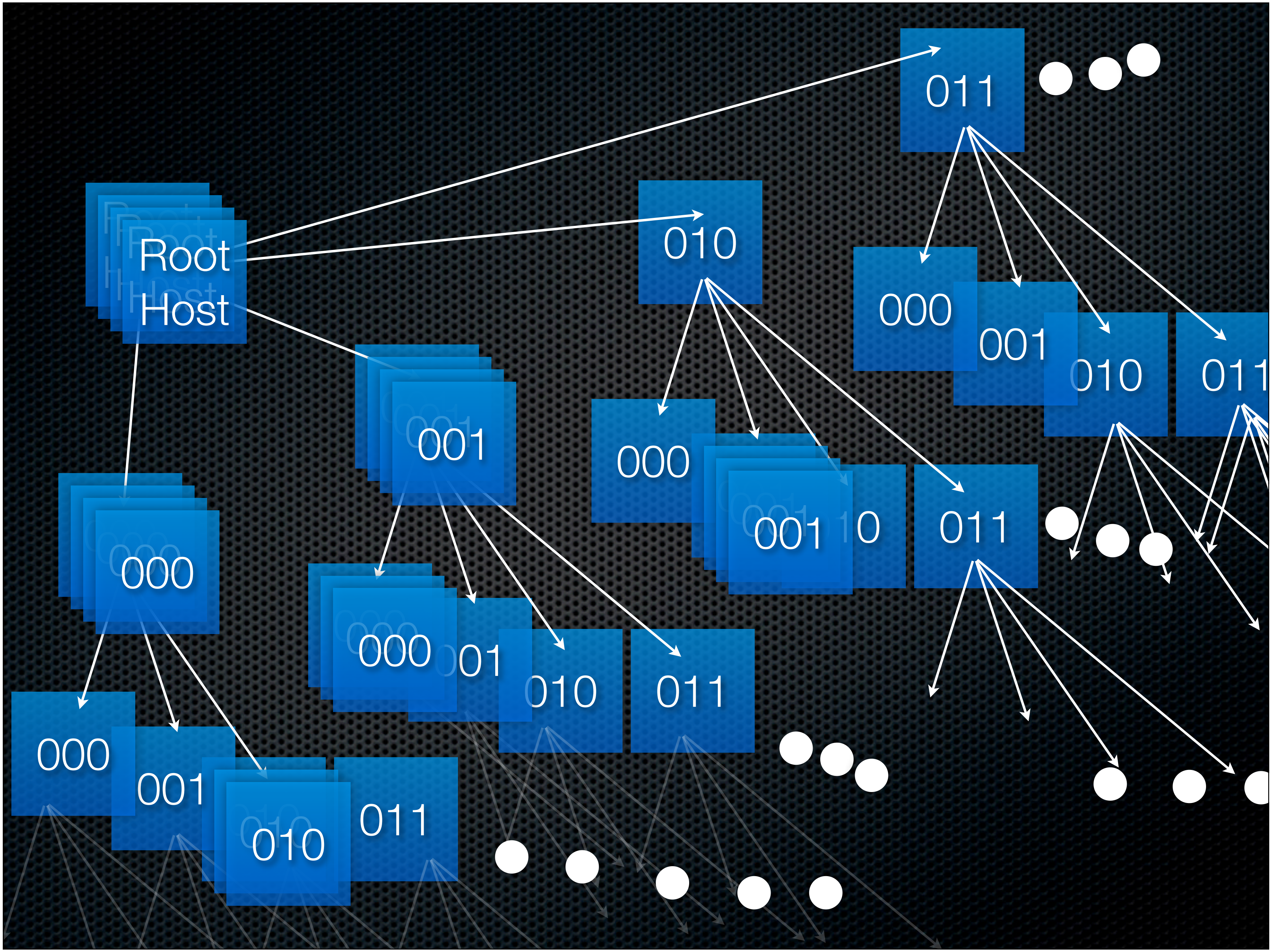

Find: 010011011101 1...
Root Host
010
011
001
000
000
001
010
011
000
001
010
011
000
001
010
011
000
001
010
011
000
001
010
011

Each host stores:

- All the data that “leaf” there.
- The list of parent nodes talking to it.
- The list of children it knows about.

Dynamically Adjusting:

- Data hashes in “clumps” making some hosts under-full and some hosts over-full.
- Host running out of storage?
  - Split in two. Give half the data to another node.
- Host running out of bandwidth?
  - Clone data and load-balance.


Root
Host
011
010
001
000
000
001
010
011
000
001
010
011
000
001
010
011
000
001
010
011

Real DHTs in action

- Peer 2 Peer file-sharing networks.
- Content Delivery Networks (CDNs like Akamai)
- Cooperative Caches

Distributed Hash Tables (DHTs)

Key/Value Stores

Some common Key/Value Stores

- “NoSQL”
  - CouchDB
  - MongoDB
  - Apache Cassandra
  - Terrastore
  - Google Bigtable

Name	Email	Address
Tom Limoncelli	tlim@google.com	1515 Main Street
Mary Smith	mary@example.com	111 One Street
Joe Bond	joe@007.com	7 Seventh St

Name	Email	Address
Tom Limoncelli	tlim@google.com	1515 Main Street
Mary Smith		
Joe Bond		

User	Transaction	Amount
Tom Limoncelli	Deposit	100
Mary Smith	Deposit	200
Tom Limoncelli	Withdraw	50

Id	Name	Email	Address
1	Tom Limoncelli	tlim@google.com	1515 Main Street
2	Mary Sr		
3	Joe Bo		

User Id	Transaction	Amount
1	Deposit	100
2	Deposit	200
1	Withdraw	50

Id	Name	Email	Address
1	Tom Limoncelli	tlim@google.com	1515 Main Street
2	Mary B		
3	Joe Bo		

User Id	Transaction	Amount
1	Deposit	100
2	Deposit	200
3	Withdraw	50

Relational Databases

- 1st Normal Form
- 2nd Normal Form
- 3rd Normal Form

- ACID: Atomicity, Consistency, Isolation, Durability

Key/Value Stores

- Keys
- Values
- BASE: Basically Available, Soft-state, Eventually consistent

Eventually?

- Who cares! This is the web, not payroll!

- Change the address listed in your profile.
- Might not propagate to Europe for 15 minutes.
- Can you fly to Europe in less than 15 minutes?
  - And if you could, would you care?

Key/Value example:

Key	Value
tlim@google.com	BLOB OF DATA
mary@example.com	BLOB OF DATA
joe@007.com	BLOB OF DATA

Key/Value example:

Key	Value
tlim@google.com	{ 'name': 'Tom Limoncelli', 'address': '1515 Main Street' }
mary@example.com	{ 'name': 'Mary Smith', 'address': '111 One Street' }
joe@007.com	{ 'name': 'Joe Bond', 'address': '7 Seventh St' }

Google Protobuf:

http://code.google.com/p/protobuf/

Key	Value

tlim@google.com	message Person {
	  required string name = 1;
	  optional string address = 2;
	  repeated string phone = 3;
	}

mary@example.com	{
	  ‘name’: ‘Mary Smith’,
	  ‘address’: ‘111 One Street’,
	  ‘phone’: [‘201-555-3456’, ‘908-444-1111’]
	}

joe@007.com	{
	  ‘name’: ‘Joe Bond’,
	  ‘phone’: [‘862-555-9876’]
	}

Key/Value Stores

Bigtable

Bigtable

- Google’s very very large database.
  - OSDI'06
  - http://labs.google.com/papers/bigtable.html
- Petabytes of data across thousands of commodity servers.
- Web indexing, Google Earth, and Google Finance

Bigtable Keys

- Can be very huge.
- Don’t have to have a value! (i.e the value is “null”)
- Query by
  - Key
  - Key start/stop range (lexigraphical order)

Long keys are cool.

Key	Value
Main St/123/Apt1	
Main St/123/Apt2	
Main St/200	Olson

Query range:
Start: “Main St/123”
End: infinity

Bigtable Values

- Values can be huge. Gigabytes.
- Multiple values per key, grouped in “families”:
  - “key:family:family:family:...”

Families

- Within a family:
  - Sub-keys that link to data.
  - Sub-keys are dynamic: no need to pre-define.
  - Sub-keys can be repeated.

Example: Crawl the web

- For every URL:
  - Store the HTML at that location.
  - Store a list of which URLs link to that URL.
  - Store the “anchor text” those sites used.

  - ANCHOR TEXT

- http://www.cnn.com
  - <html>.........</html>

- http://tomontime.com
  - <html>
    - <p>As you may have read on my favorite news site there is...

Family | Another family

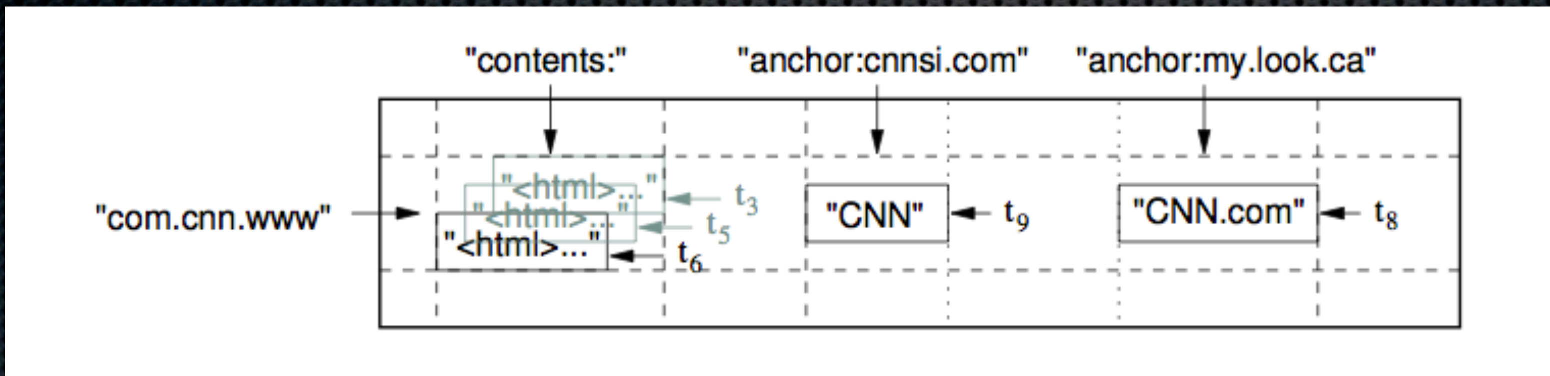
Key	contents:	anchor:tomontime.com	anchor:cnnsi.com
com.cnn.www	<html>...	my favorite news site	CNN

Key	contents:	anchor:everythingsysadmin.com
com.tomontime	<html>...	videos

Each Family has its own...

- Permissions (who can read/write/admin)
- QoS (optimize for speed, storage diversity, etc.)

Plus “time”



- All updates are timestamped.
- Retains at least *n* recent updates or “never”.
- Expired updates are garbage collected “eventually”.

Bigtable

Further Reading:

- Bigtable:
  - http://research.google.com
- A visual guide to NoSQL:
  - http://blog.nahurst.com/visual-guide-to-nosql-systems
- HashTables, DHTs, everything else
  - Wikipedia

Other futuristic topics:

- Stop using “locks”, eliminate all deadlocks:
  - STM: Software Transactional Memory
- Centralized routing: (you’d be surprised)
  - 2 minute overview: www.openflowswitch.org
  - (the 4 minute demo video is MUCH BETTER)
- “Network Coding”: n^2 more bandwidth?
  - SciAm.com: “Breaking Network Logjams”

Q&A

How to do a query?

KEY	VALUE
bird	“{ legs=2, horns=0, covering=‘feathers’ }”
cat	“{ legs=4, horns=0, covering=‘fur’ }”
dog	“{ legs=4, horns=0, covering=‘fur’ }”
spider	“{ legs=8, horns=0, covering=‘hair’ }”
unicorn	“{ legs=4, horns=1, covering=‘hair’ }”

“Which animals have 4 legs?”

- Iterate over entire list
  - Open up each blob
  - Parse data
  - Accumulate list

SLOW!

KEY	VALUE
animal:bird	“{ legs=2, horns=0, covering=‘feathers’ }”
animal:cat	“{ legs=4, horns=0, covering=‘fur’ }”
animal:dog	“{ legs=4, horns=0, covering=‘fur’ }”
animal:spider	“{ legs=8, horns=0, covering=‘hair’ }”
animal:unicorn	“{ legs=4, horns=1, covering=‘hair’ }”
legs:2:bird	
legs:4:cat	
legs:4:dog	
legs:4:unicorn	
legs:8:spider	

Iterate:
Start: “legs:4”
End: “legs:5”

Up to, but not including “end”

legs=4 AND covering=fur

- More indexes + the “zig zag” algorithm.
- More indexed attributes = the slower insertions
- Automatic if you use AppEngine’s storage system