OPEN DHT
A Public DHT Service

Sean Rhea, Brighten Godfrey, Brad Karp,
John Kubiatowicz, Sylvia Ratnasamy,
Scott Shenker, Ion Stoica, and Harlan Yu

UC Berkeley and Intel Research

August 23, 2005

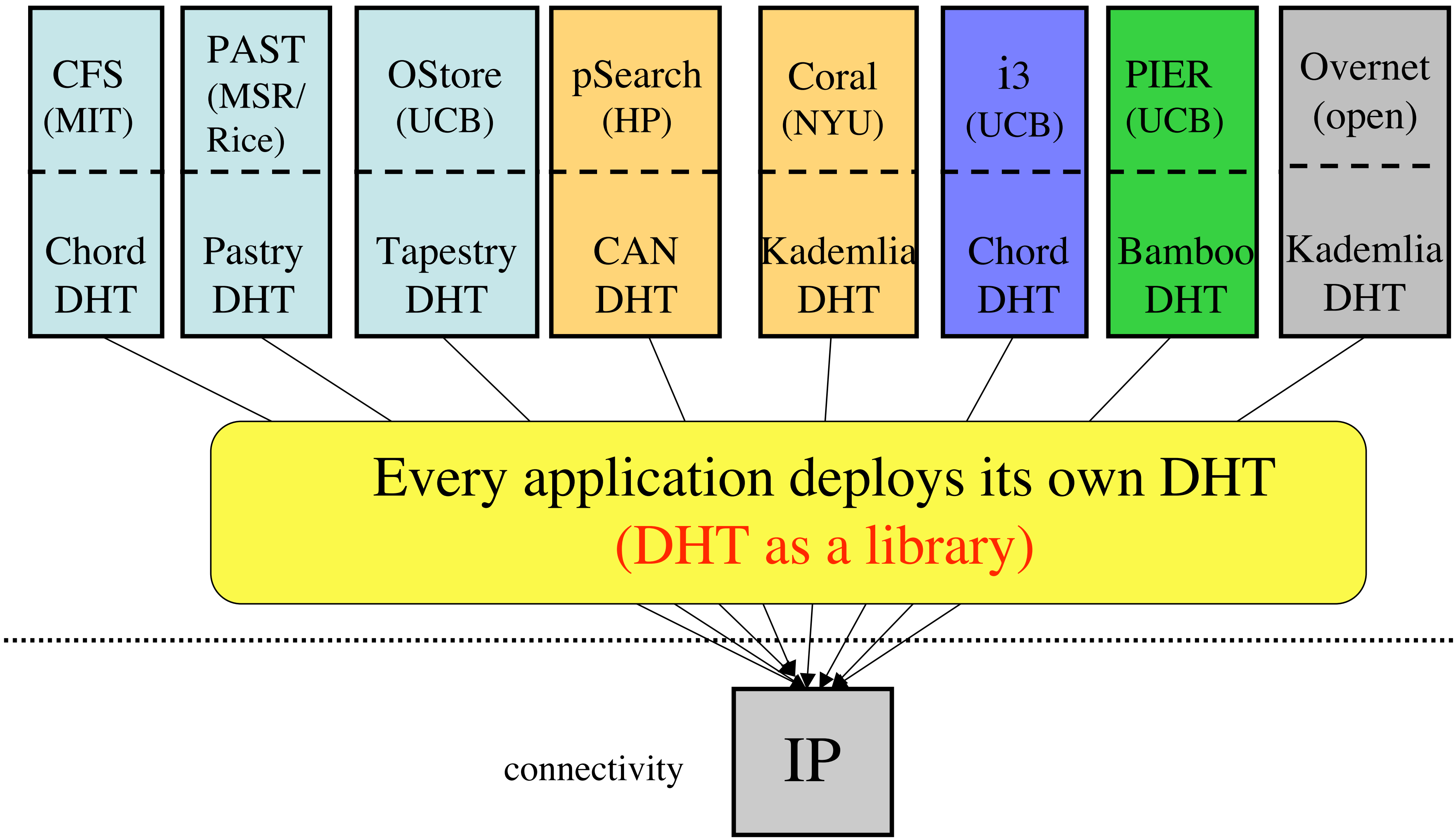
Two Assumptions

1. Most of you have a pretty good idea how to build a DHT
2. Many of you would like to forget

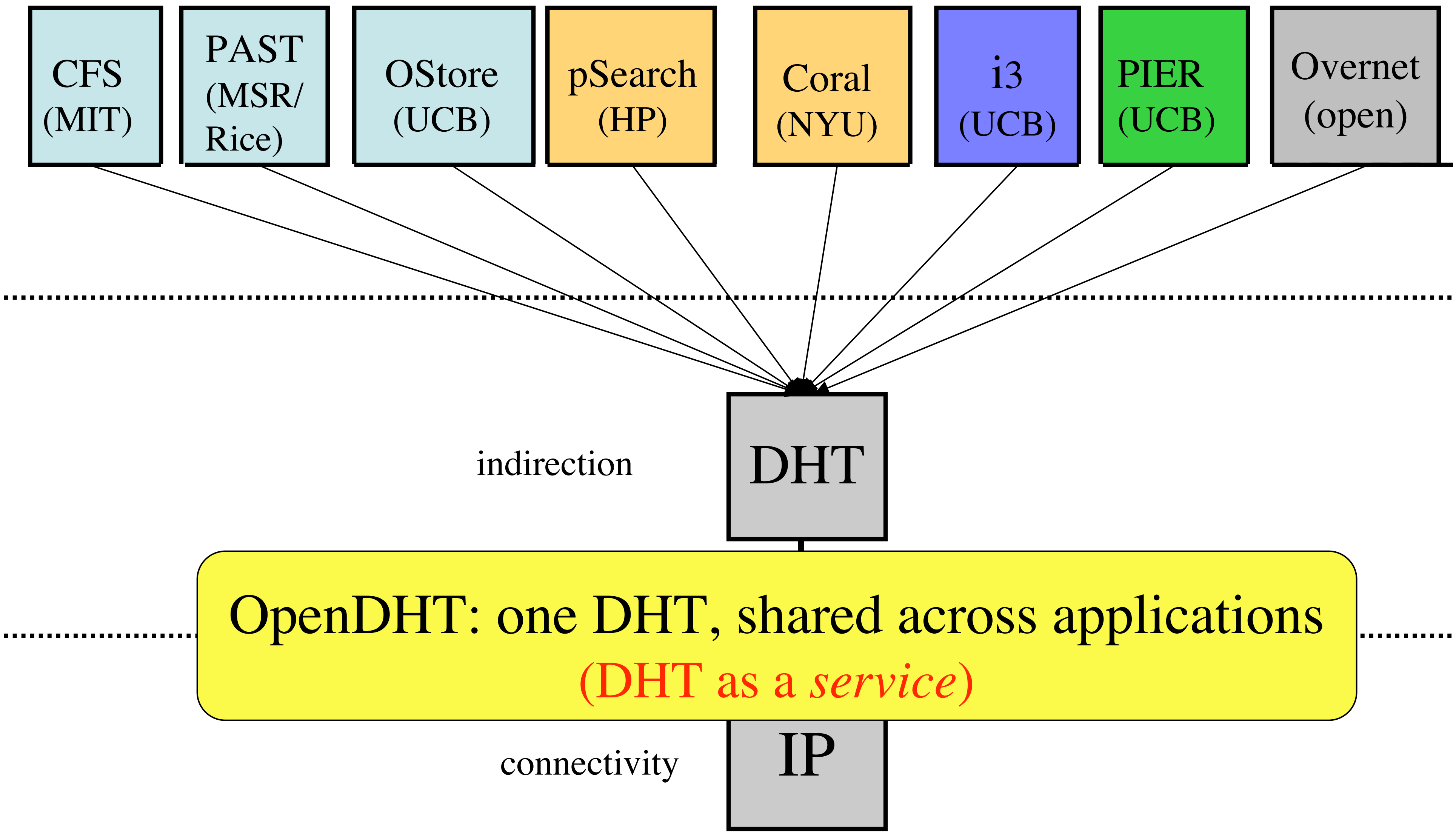
My talk today:

How to *avoid* building one

DHT Deployment Today



DHT Deployment Tomorrow?



Two Ways To Use a DHT

1. The Library Model
   – DHT code is linked into application binary
   – Pros: flexibility, high performance
2. The Service Model
   – DHT accessed as a service over RPC
   – Pros: easier deployment, less maintenance

The OpenDHT Service

- 200-300 Bamboo [USENIX'04] nodes on PlanetLab
  - All in one slice, all managed by us
- Clients can be arbitrary Internet hosts
  - Access DHT using RPC over TCP
- Interface is simple put/get:
  - put(*key*, *value*) — stores *value* under *key*
  - get(*key*) — returns all the values stored under *key*
- Running on PlanetLab since April 2004
  - Building a community of users

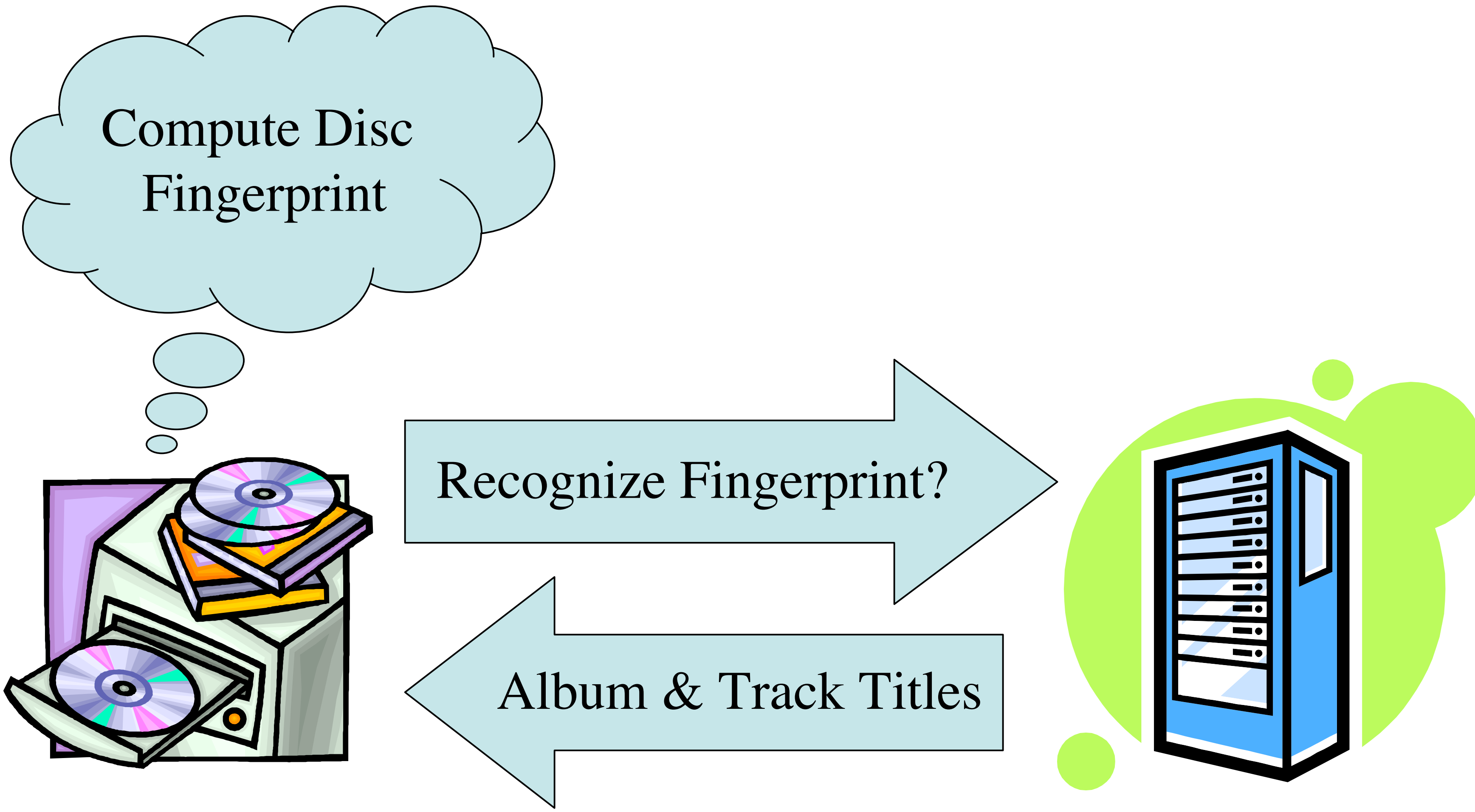
OpenDHT Applications

Application	*Uses OpenDHT for*
Croquet Media Manager	replica location
DOA	indexing
HIP	name resolution
DTN Tetherless Computing Architecture	host mobility
Place Lab	range queries
QStream	multicast tree construction
VPN Index	indexing
DHT-Augmented Gnutella Client	rare object search
FreeDB	storage
Instant Messaging	rendezvous
CFS	storage
*i*3	redirection

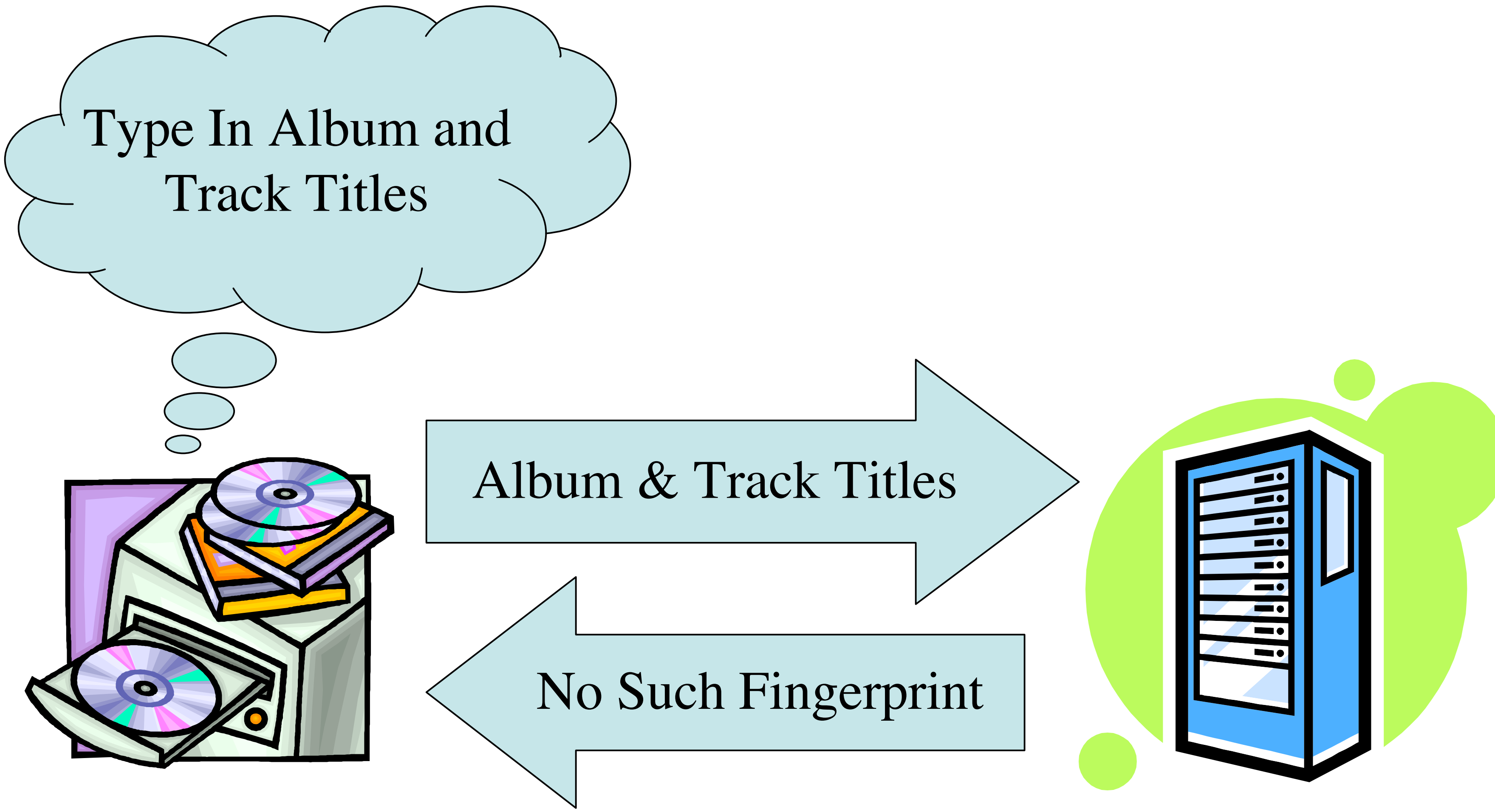
OpenDHT Benefits

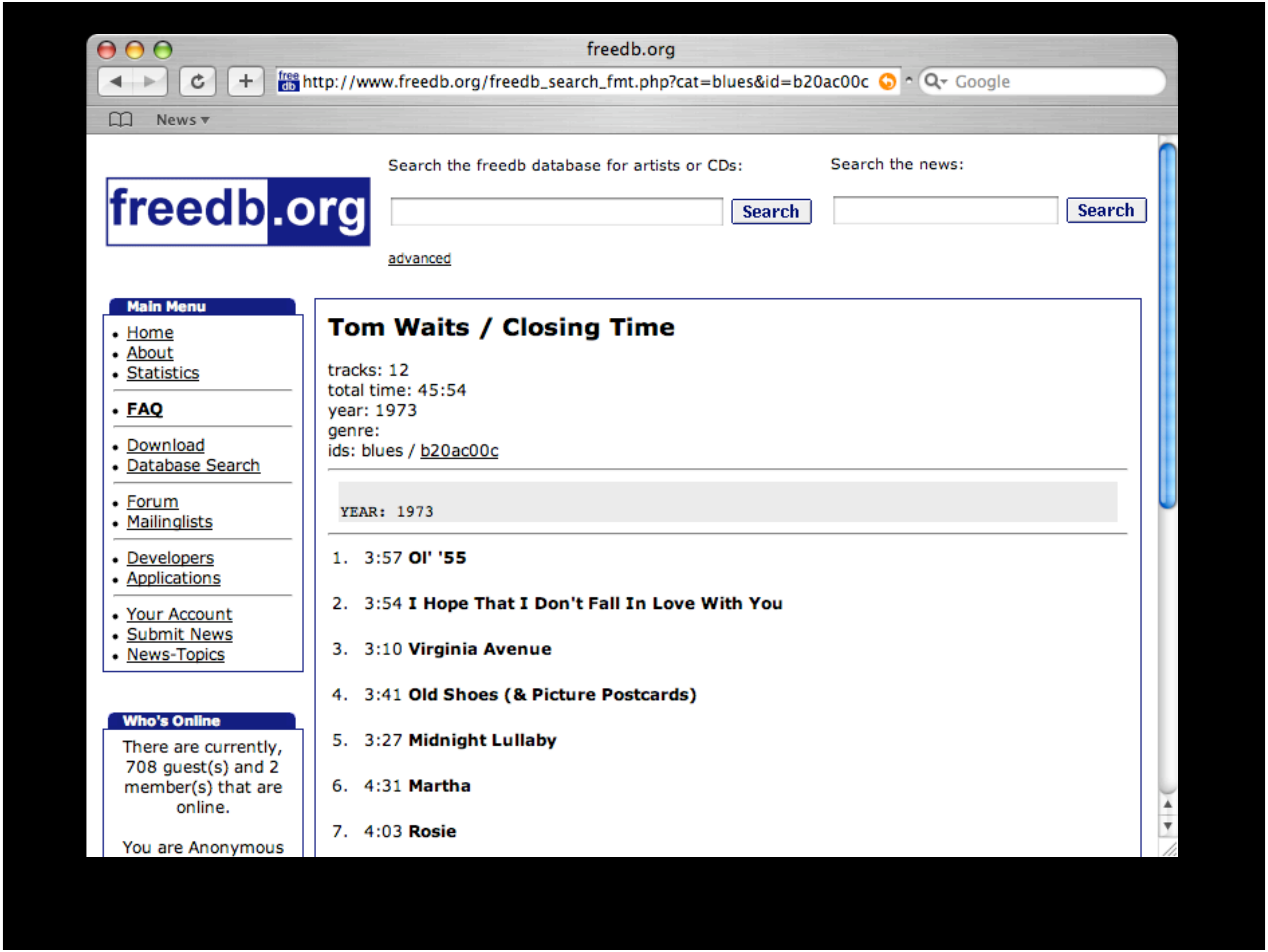
- OpenDHT makes applications
  - Easy to build
    - Quickly bootstrap onto existing system
  - Easy to maintain
    - Don't have to fix broken nodes, deploy patches, etc.
- Best illustrated through example

An Example Application: The CD Database



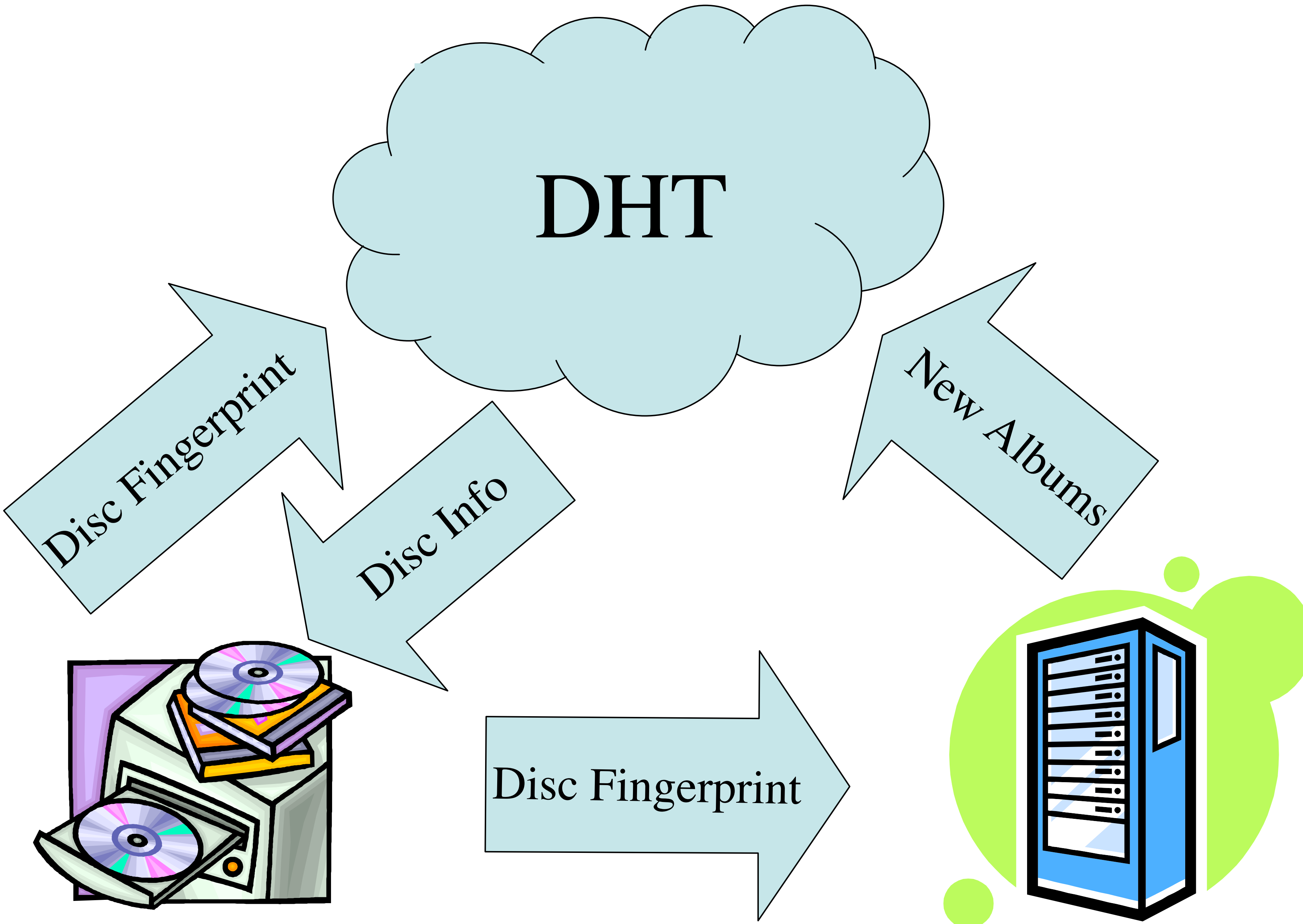
An Example Application: The CD Database




freedb.org
http://www.freedb.org/freedb_search_fmt.php?cat=blues&id=b20ac00c
Google
News
freedb.org
Search the freedb database for artists or CDs:
Search
advanced
Search the news:
Search
Main Menu
Home
About
Statistics
FAQ
Download
Database Search
Forum
Mailinglists
Developers
Applications
Your Account
Submit News
News-Topics
Who's Online
There are currently, 708 guest(s) and 2 member(s) that are online.
You are Anonymous
Tom Waits / Closing Time
tracks: 12
total time: 45:54
year: 1973
genre:
ids: blues / b20ac00c
YEAR: 1973
1. 3:57 Ol' '55
2. 3:54 I Hope That I Don't Fall In Love With You
3. 3:10 Virginia Avenue
4. 3:41 Old Shoes (& Picture Postcards)
5. 3:27 Midnight Lullaby
6. 4:31 Martha
7. 4:03 Rosie

A DHT-Based FreeDB Cache

- FreeDB is a volunteer service
  - Has suffered outages as long as 48 hours
  - Service costs born largely by volunteer mirrors
- Idea: Build a cache of FreeDB with a DHT
  - Add to availability of main service
  - Goal: explore how easy this is to do


DHT
Disc Fingerprint
Disc Info
New Albums
Disc Fingerprint

Building a FreeDB Cache Using the Library Approach

1. Download Bamboo/Chord/FreePastry
2. Configure it
3. Register a PlanetLab slice
4. Deploy code using Stork
5. Configure AppManager to keep it running
6. Register some gateway nodes under DNS
7. Dump database into DHT
8. Write a proxy for legacy FreeDB clients

Building a FreeDB Cache Using the Service Approach

1. Dump database into DHT
2. Write a proxy for legacy FreeDB clients

- We built it
  - Called FreeDB on OpenDHT (FOOD)

food.pl (~/Desktop) - VIM

File | Edit | Tools | Syntax | Buffers | Window | Help

```perl
#!/usr/bin/perl
use HTTP::Daemon; use HTTP::Status; use HTTP::Request; use Digest::SHA1;
use Compress::Bzip2; use MIME::Base64; require Frontier::Client;
$cl=Frontier::Client->new( 'url' => $ARGS[0], 'encoding' => 'ISO-8859-1' );
$d=HTTP::Daemon->new(LocalAddr => 'localhost', LocalPort => 4444, Reuse => 1,);
@cats=("blues","classical","country","data","folk","jazz","misc","newage",
"reggae","rock","soundtrack"); sub fm{($a,$b)=@_; if ($##$a==$##$b){
($i,$avg,$o,$aprev,$bprev)= (0,0,0,0,0);for($i=0;$i<=$##$a;$i++){
$o=abs(($$a[$i]-$aprev)-($$b[$i]-$bprev)); $avg+=$o; if($o>900){return 0;}
$aprev=$$a[$i]; $bprev=$$b[$i];} $avg/=($##$a+1); return ($avg<=(225))?1:0;}}
sub dp {($dd)=@_; @dlines=split /\n/,$dd; $inoffsets=0; $dlen=0; my @offsets;
$dd=~/DTITLE=(.*?)\n/; $dt=$1; foreach(@dlines){if(/Track\s+frame\s+offsets:/){
$inoffsets=1; next;} elsif($inoffsets){if (/Disc\s+length:\s+([0-9]+)/) {
$dlen=$1;last;} elsif(/([0-9]+)/){push @offsets, $1;}}} if($dlen==0){die($fn);}
return ($dt,$dlen,@offsets);} sub uri_parse{($uri)=@_; $r=HTTP::Response->new();
if($uri=~/cddb\+query\+([0-9a-f]+)\+([0-9]+?)\+/){@q=split /\+/,$',$2+1;
$#q=$2-1;$sha1=Digest::SHA1->new;$sha1->add($1);$hexkey=$sha1->clone->hexdigest;
$key=$sha1->b64digest.'='; $pmark=''; $found=0; do{$rpcresp=$cl->call('get',
$cl->base64($key),$cl->int(1024),$cl->base64($pmark),$cl->string('FOOD'));
foreach $disc (@{$$rpcresp[0]}){$foo=decode_base64($disc->value());
$p=Compress::Bzip2::decompress_init();$bar=substr($foo, 1);$dd=$p->finish($bar);
($dt,$dlen,@idx)=dp($dd);$cat=$cats[ord(substr($foo,0,1))];if(fm(\@q,\@idx)) {
$cache{$cat.$1}=$dd; $r->content("200 $cat $1 $dt\n"); return $r;}}
$pmark = $$rpcresp[1]->value(); last if ($found == 1);} while(length($pmark)>0);
$r->content("202 No match found\n");}
elsif ($uri =~ /cddb\+read\+([a-z]+)\+([0-9a-f]+)\&/) {$r->add_content(
"210 $1 $2 CD database entry follows (until terminating `.')\n");
$r->add_content($cache{$1.$2}); $r->add_content('.\n');} else{$r=0;}return $r;}
while ($c=$d->accept){while($r=$c->get_request){if ($r->method eq 'GET') {
$repl = uri_parse ($r->uri); if ($repl) {$c->send_response($repl);}
else {$c->send_error(RC_FORBIDDEN);}}} $c->close; undef($c);}
~
~
```

31,61 All

Building a FreeDB Cache Using the Service Approach

1. Dump database into DHT
2. Write a proxy for legacy FreeDB clients

- We built it
  - Called FreeDB on OpenDHT (FOOD)
  - Cache has ↓ latency, ↑ availability than FreeDB

Talk Outline

- Introduction and Motivation
- Challenges in building a shared DHT
  - Sharing between applications
  - Sharing between clients
- Current Work
- Conclusion

Is Providing DHT Service Hard?

- Is it any different than just running Bamboo?
  - Yes, sharing makes the problem harder
- OpenDHT is shared in two senses
  - Across applications → need a flexible interface
  - Across clients → need resource allocation

Sharing Between Applications

- Must balance generality and ease-of-use
  - Many apps (FOOD) want only simple put/get
  - Others want lookup, anycast, multicast, etc.
- OpenDHT allows only put/get
  - But use client-side library, ReDiR, to build others
  - Supports lookup, anycast, multicast, range search
  - Only constant latency increase on average
  - (Different approach used by DimChord [KR04])

Sharing Between Clients

- Must authenticate puts/gets/removes
  - If two clients put with same key, who wins?
  - Who can remove an existing put?
- Must protect system's resources
  - Or malicious clients can deny service to others
  - The remainder of this talk

Protecting Storage Resources

- Resources include network, CPU, and disk
  - Existing work on network and CPU
  - Disk less well addressed
- As with network and CPU:
  - Hard to distinguish malice from eager usage
  - Don’t want to hurt eager users if utilization low
- Unlike network and CPU:
  - Disk usage persists long after requests are complete
- Standard solution: quotas
  - But our set of active users changes over time

Fair Storage Allocation

- Our solution: give each client a fair share
  - Will define “fairness” in a few slides
- Limits strength of malicious clients
  - Only as powerful as they are numerous
- Protect storage on each DHT node separately
  - Global fairness is hard
  - Key choice imbalance is a burden on DHT
  - Reward clients that balance their key choices

Two Main Challenges

1. Making sure disk is available for new puts
   – As load changes over time, need to adapt
   – Without some free disk, our hands are tied

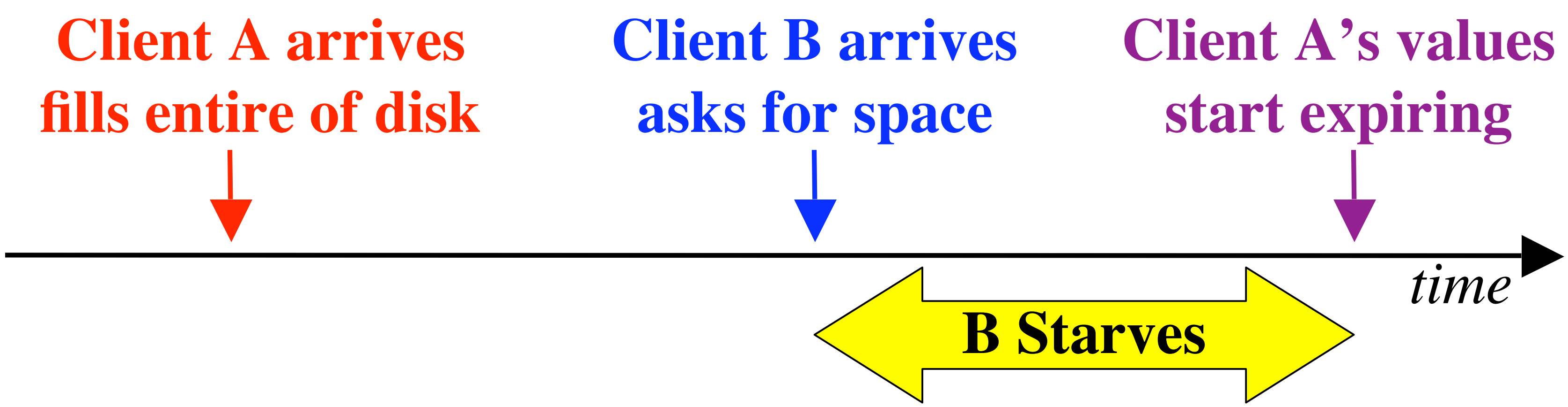
2. Allocating free disk fairly across clients
   – Adapt techniques from fair queuing

Making Sure Disk is Available

- Can't store values indefinitely
  - Otherwise all storage will eventually fill
- Add time-to-live (TTL) to puts
  - put (key, value) → put (key, value, ttl)
  - (Different approach used by Palimpsest [RH03])

Making Sure Disk is Available

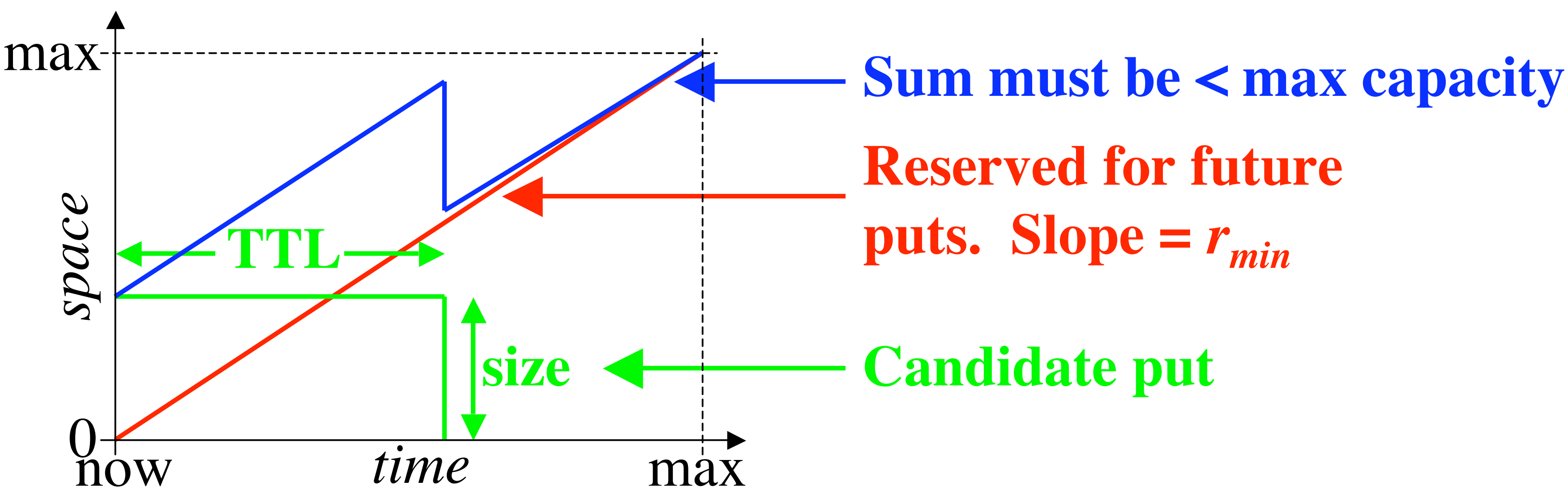
- TTLs prevent long-term starvation
  - Eventually all puts will expire
- Can still get short term starvation:



Making Sure Disk is Available

- Stronger condition:

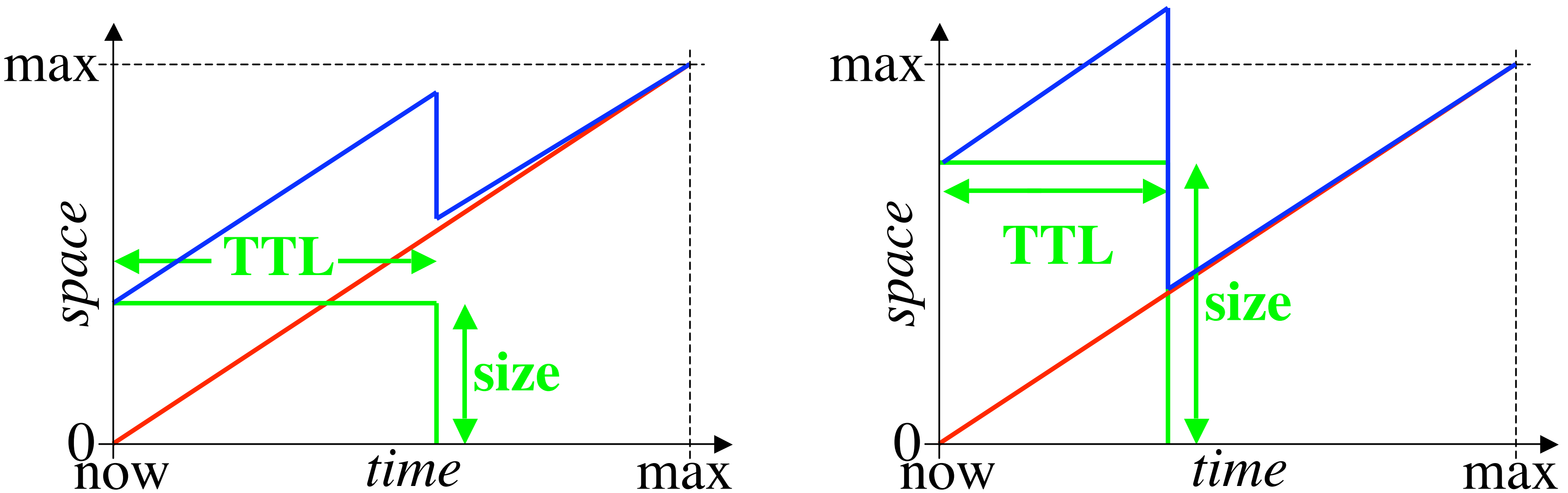
  Be able to accept r_{min} bytes/sec new data at all times



Making Sure Disk is Available

- Stronger condition:

Be able to accept r_{min} bytes/sec new data at all times



Making Sure Disk is Available

- Formalize graphical intuition:

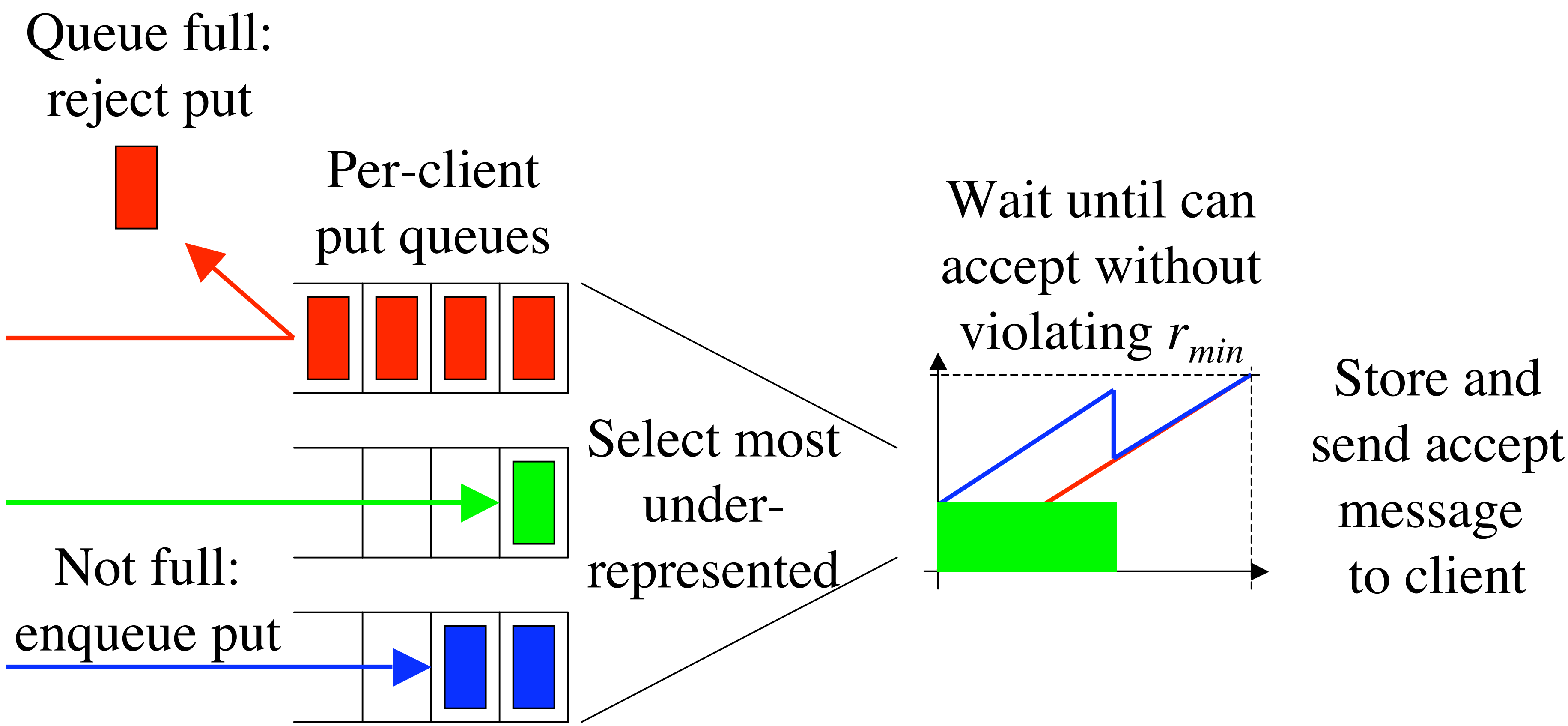
  $$f(\tau) = B(t_{now}) - D(t_{now}, t_{now} + \tau) + r_{min} \times \tau$$

- To accept put of size x and TTL l:

  $$f(\tau) + x < C \text{ for all } 0 \leq \tau < l$$

- This is non-trivial to arrange
  - Have to track $f(\tau)$ at all times between now and max TTL?
- Can track the value of f efficiently with a tree
  - Leaves represent inflection points of f
  - Add put, shift time are $O(\log n)$, n = # of puts

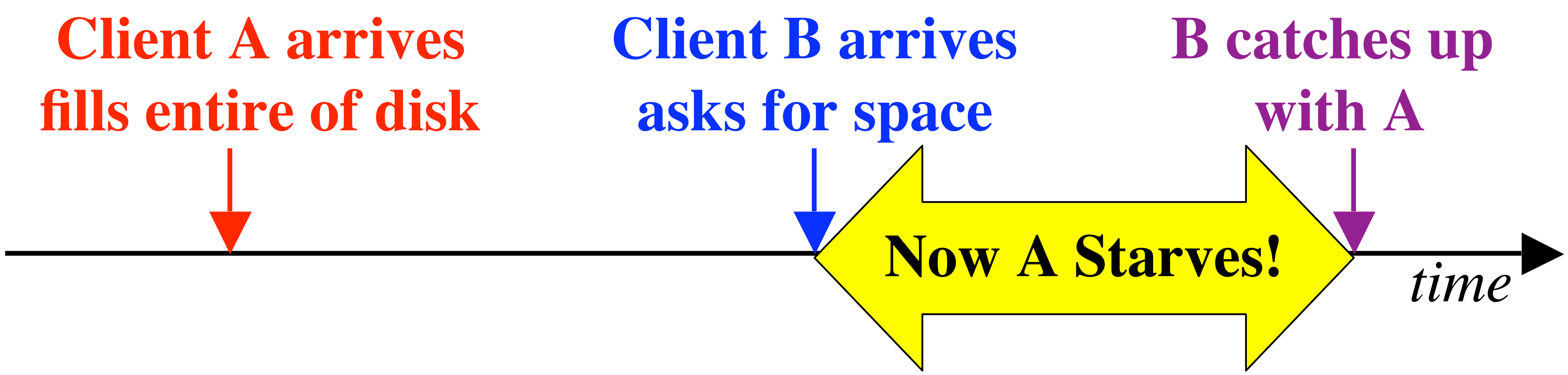
Fair Storage Allocation



The Big Decision: Definition of “most under-represented”

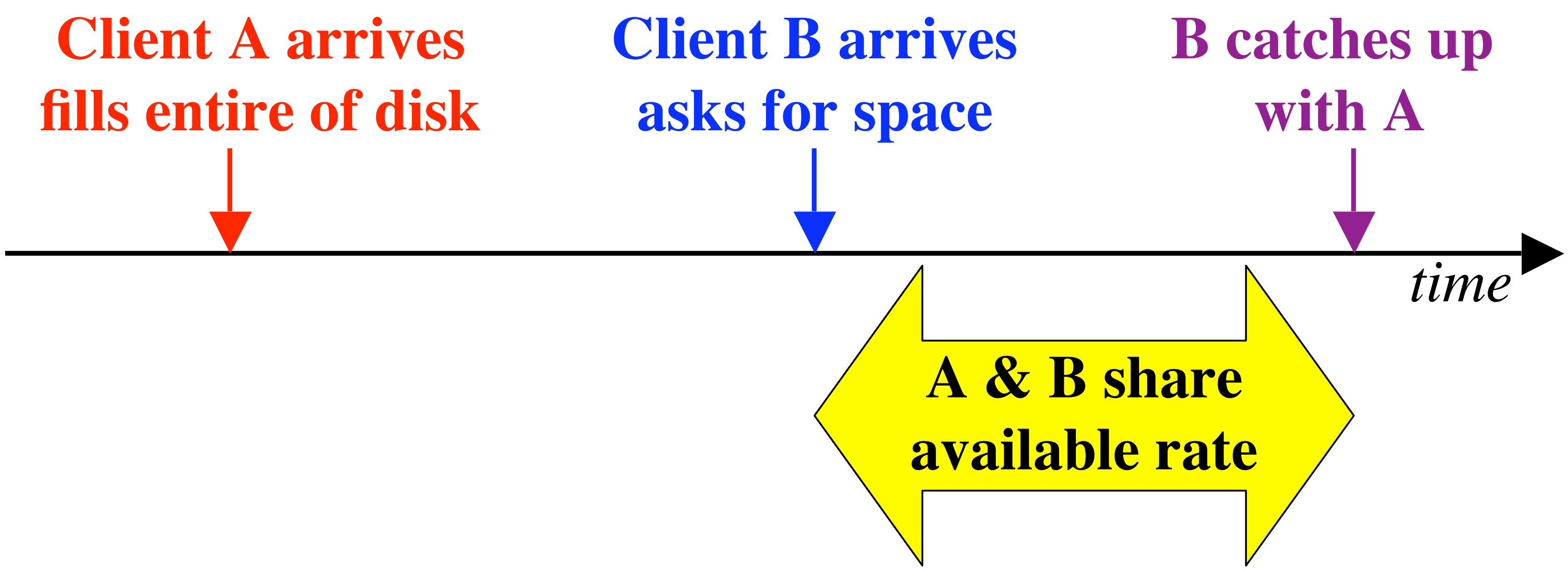
Defining "Most Under-Represented"

- Not just sharing disk, but disk over time
  - 1-byte put for 100s same as 100-byte put for 1s
  - So units are bytes × seconds, call them *commitments*
- Equalize total commitments granted?
  - No: leads to starvation
  - A fills disk, B starts putting, A starves up to max TTL



Defining "Most Under-Represented"

- Instead, equalize *rate* of commitments granted
  - Service granted to one client depends only on others putting "at same time"



Defining "Most Under-Represented"

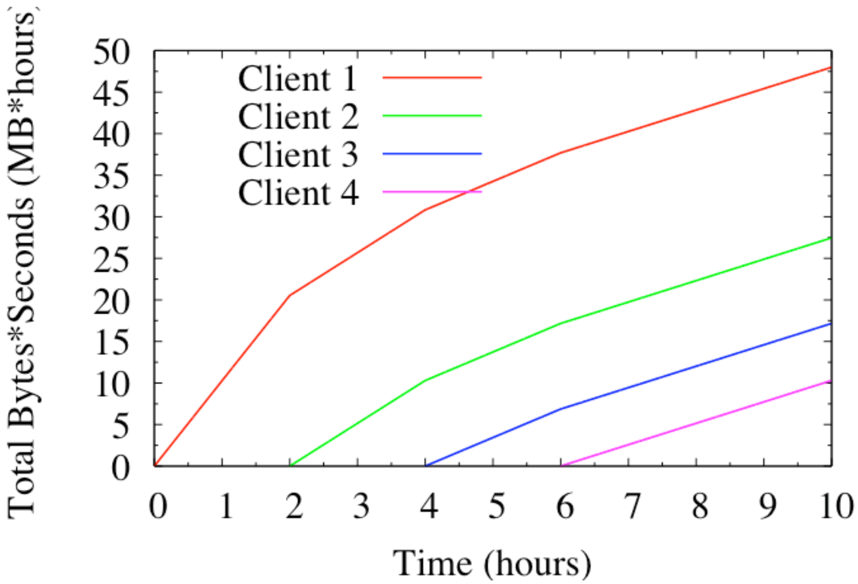
- Instead, equalize *rate* of commitments granted
  - Service granted to one client depends only on others putting "at same time"
- Mechanism inspired by Start-time Fair Queuing
  - Have virtual time, v(t)
  - Each put gets a start time $S(p_c^i)$ and finish time $F(p_c^i)$

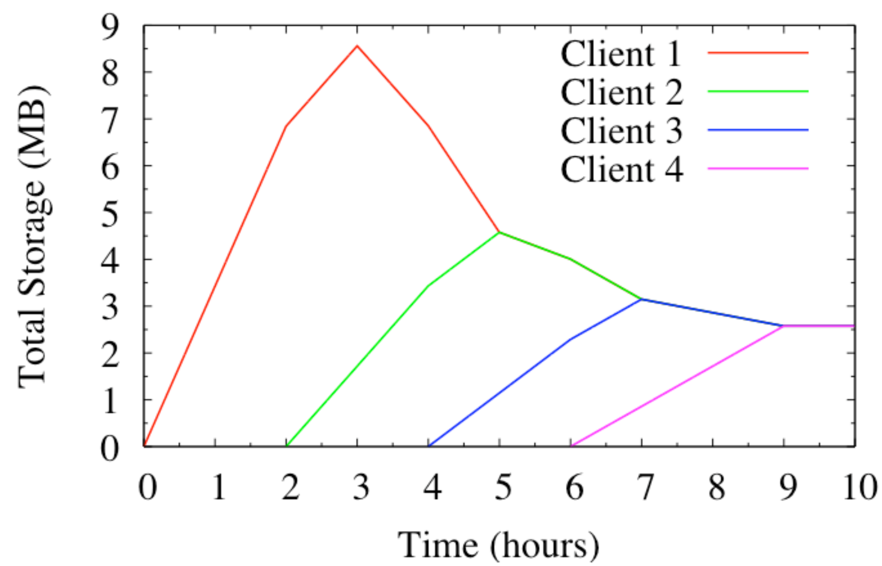
$$F(p_c^i) = S(p_c^i) + size(p_c^i) \times ttl(p_c^i)$$

$$S(p_c^i) = \max(v(A(p_c^i)) - \varepsilon, F(p_c^{i-1}))$$

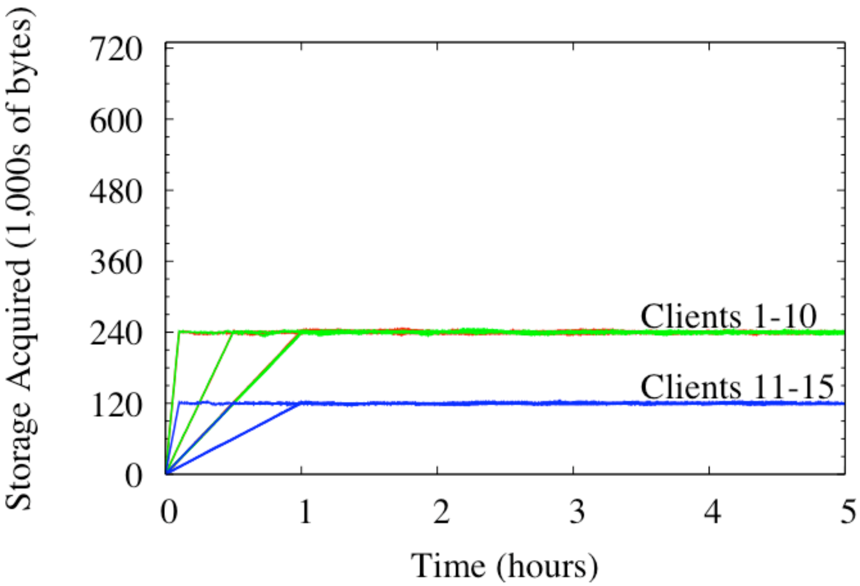
v(t) = maximum start time of all accepted puts

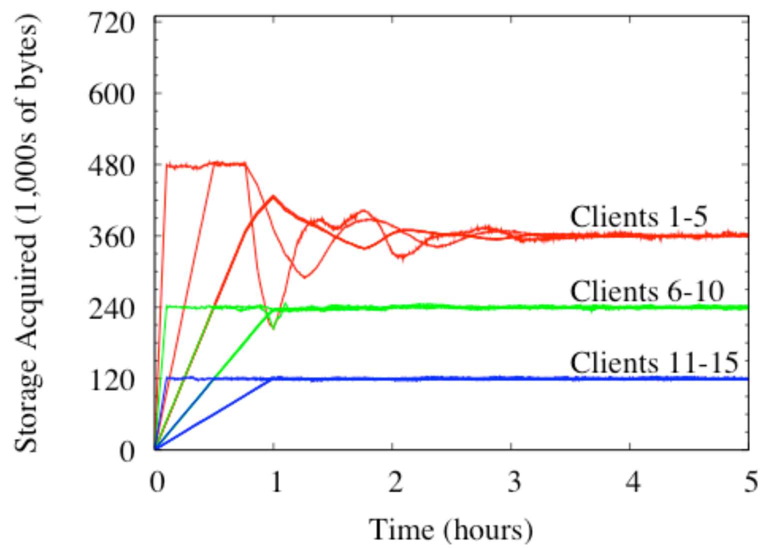
Fairness with Different Arrival Times





Fairness With Different Sizes and TTLs

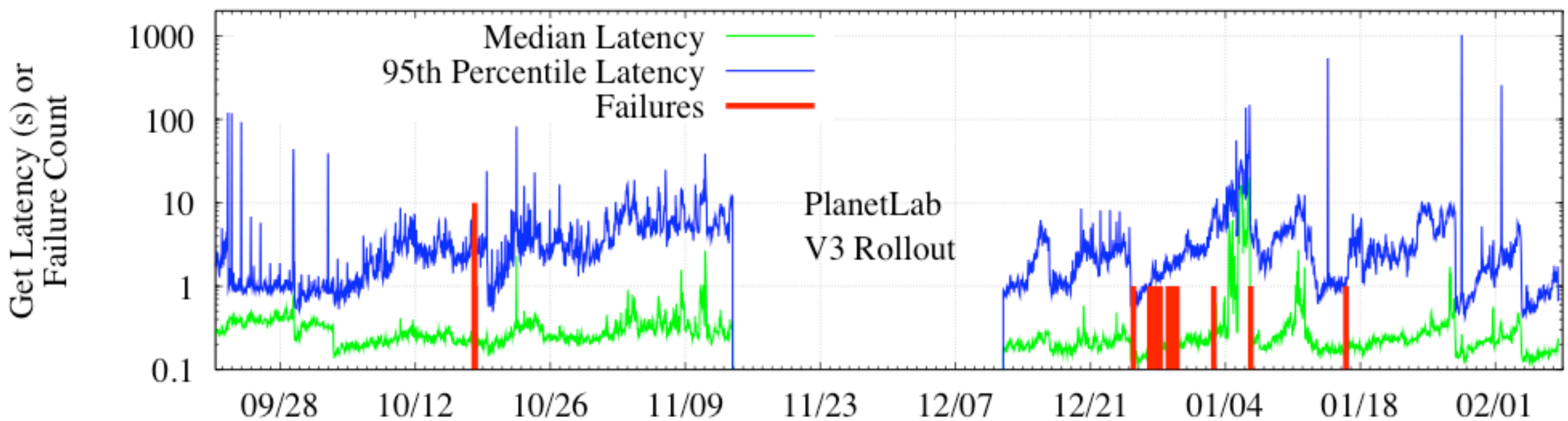




Talk Outline

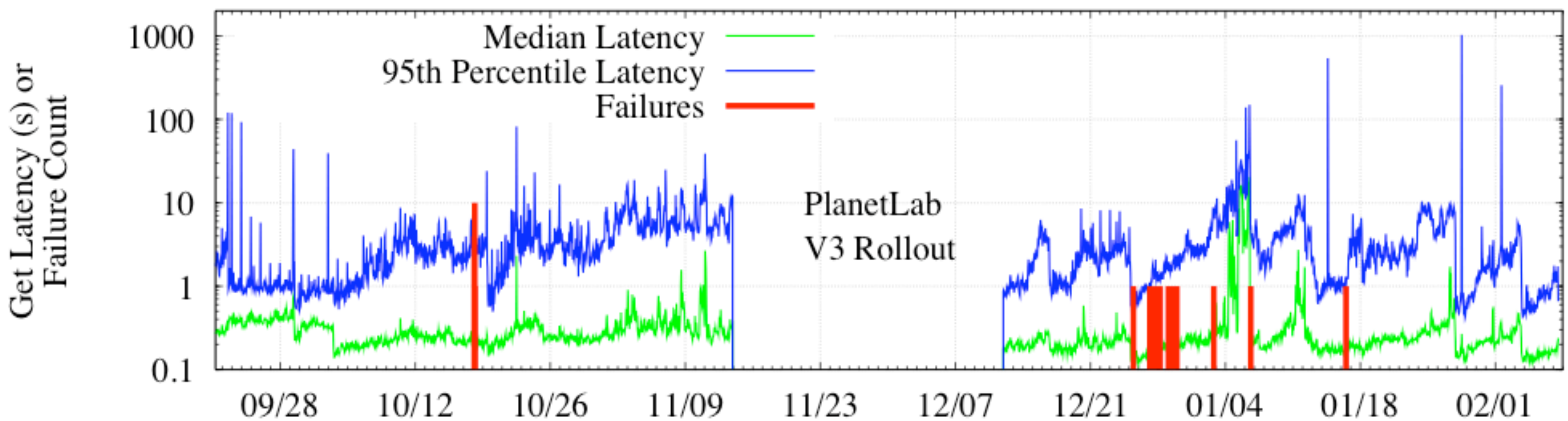
- Introduction and Motivation
- Challenges in building a shared DHT
  - Sharing between applications
  - Sharing between clients
- Current Work
- Conclusion

Current Work: Performance



- Only 28 of 7 million values lost in 3 months
  - Where "lost" means unavailable for a full hour
- On Feb. 7, 2005, lost 60/190 nodes in 15 minutes to PL kernel bug, only lost one value

Current Work: Performance



- Median get latency ~250 ms
  - Median RTT between hosts ~ 140 ms
- But 95th percentile get latency is atrocious
  - And even median spikes up from time to time

The Problem: Slow Nodes

- Some PlanetLab nodes are just really slow
  - But set of slow nodes changes over time
  - Can’t “cherry pick” a set of fast nodes
  - Seems to be the case on RON as well
  - May even be true for managed clusters (MapReduce)
- Modified OpenDHT to be robust to such slowness
  - Combination of delay-aware routing and redundancy
  - Median now 66 ms, 99th percentile is 320 ms (using 2X redundancy)

Conclusion

- Focusing on how to *use* a DHT
  - Library model: flexible, powerful, often overkill
  - Service model: easy to use, shares costs
  - Both have their place, we're focusing on the latter
- Challenge: Providing for sharing
  - Across applications → flexible interface
  - Across clients → fair resource sharing
- Up and running today

To try it out:

(code at http://opendht.org/users-guide.html)

```
$ ./find-gateway.py | head -1
planetlab5.csail.mit.edu

$ ./put.py http://planetlab5.csail.mit.edu:5851/ Hello World 3600
Success

$ ./get.py http://planetlab5.csail.mit.edu:5851/ Hello
World
```

Identifying Clients

- For fair sharing purposes, a client is its IP addr
  - Spoofing prevented by TCP's 3-way handshake
- Pros:
  - Works today, no registration necessary
- Cons:
  - All clients behind NAT get only one share
  - DHCP clients get more than one share
- Future work: authentication at gateways