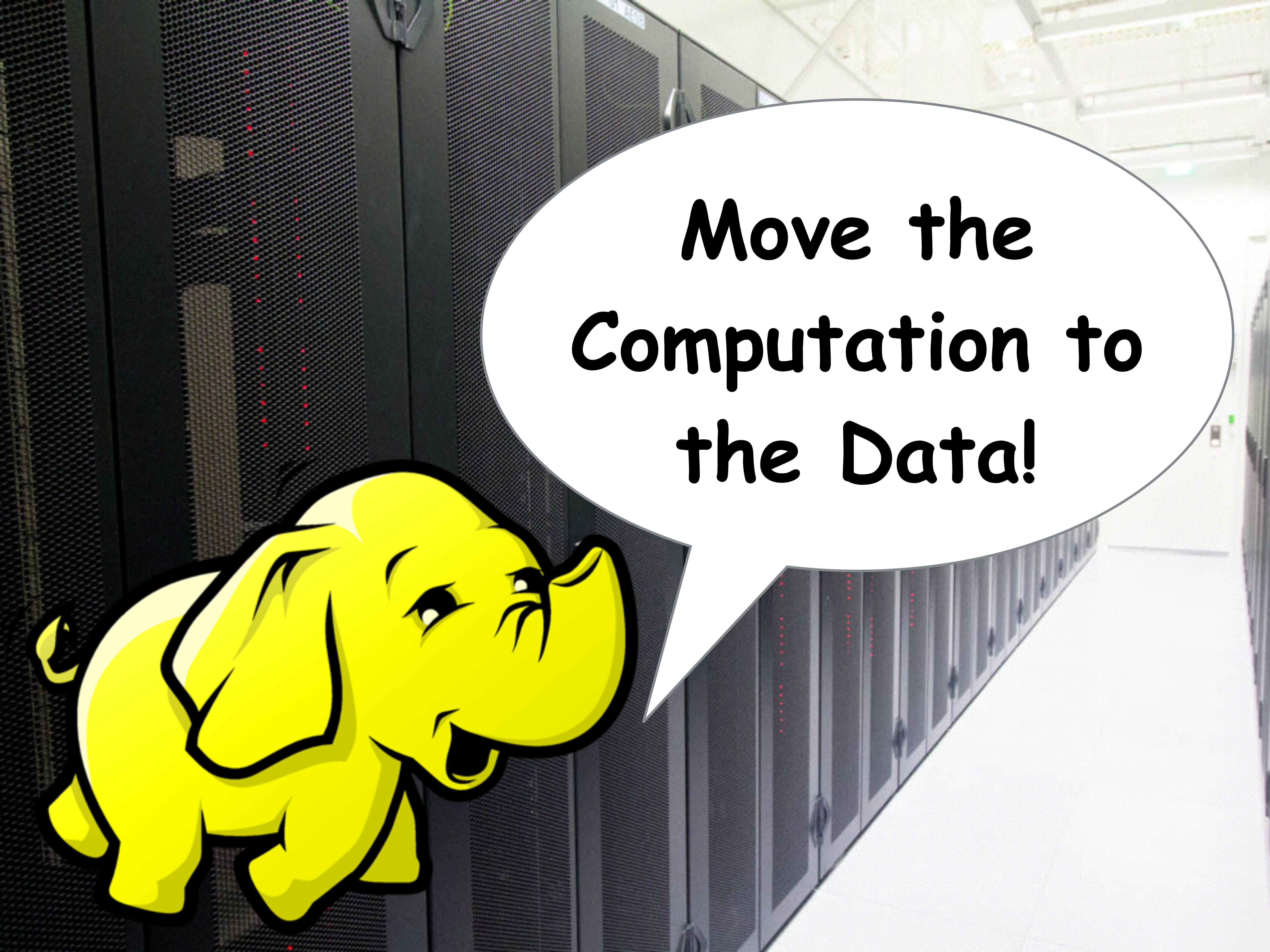
Flat Datacenter Storage

Edmund B. Nightingale, Jeremy Elson, Jinliang Fan,
Owen Hofmann, Jon Howell, Yutaka Suzue

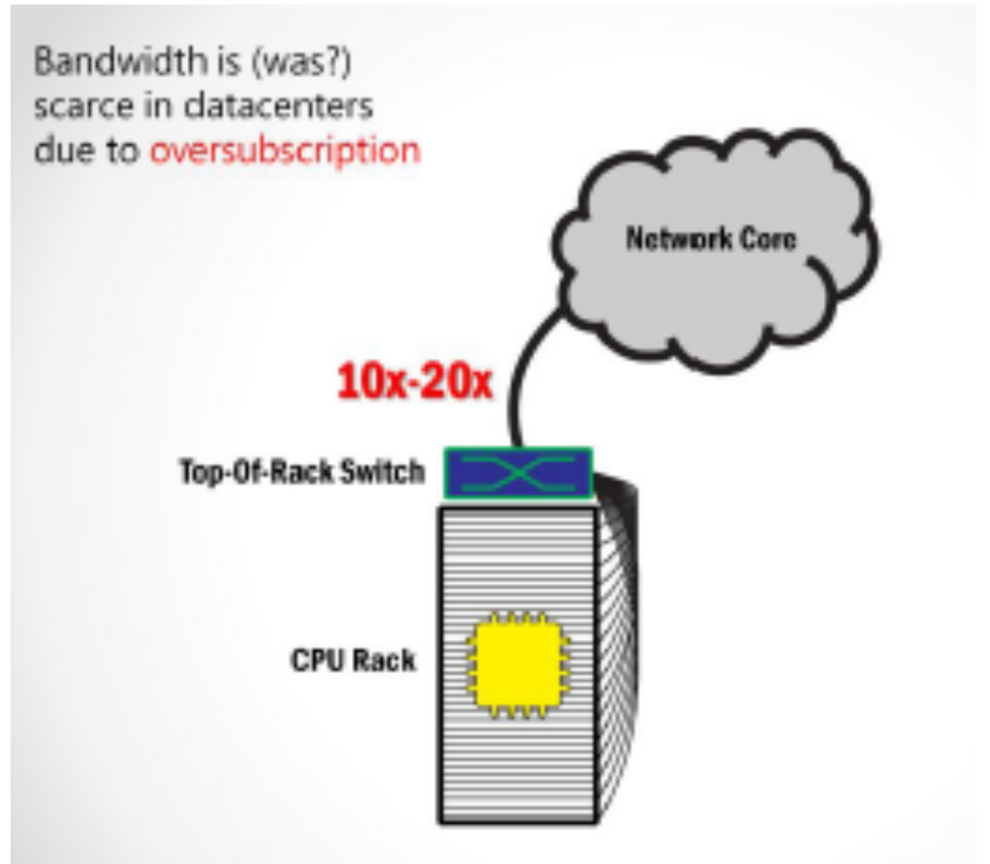
Presented by Rashmi Vinayak
9/21/2015

(Slides sourced from Jeremy Elson's presentation at OSDI 2012 and Alex Rasmussen's presentation at Papers We Love SF #11 with some modifications)


Move the Computation to the Data!

Why move computation close to data?

Because remote access is slow due to oversubscription



Locality adds complexity

- Need to be aware of where the data is
  - Non-trivial scheduling algorithm
  - Moving computations around is not easy
- Need a data-parallel programming model
  - cannot express all desired computations efficiently

What if the network
is *not oversubscribed?*

Consequences

- No local vs. remote disk distinction
- Simpler work schedulers
- Simpler programming models

FDS

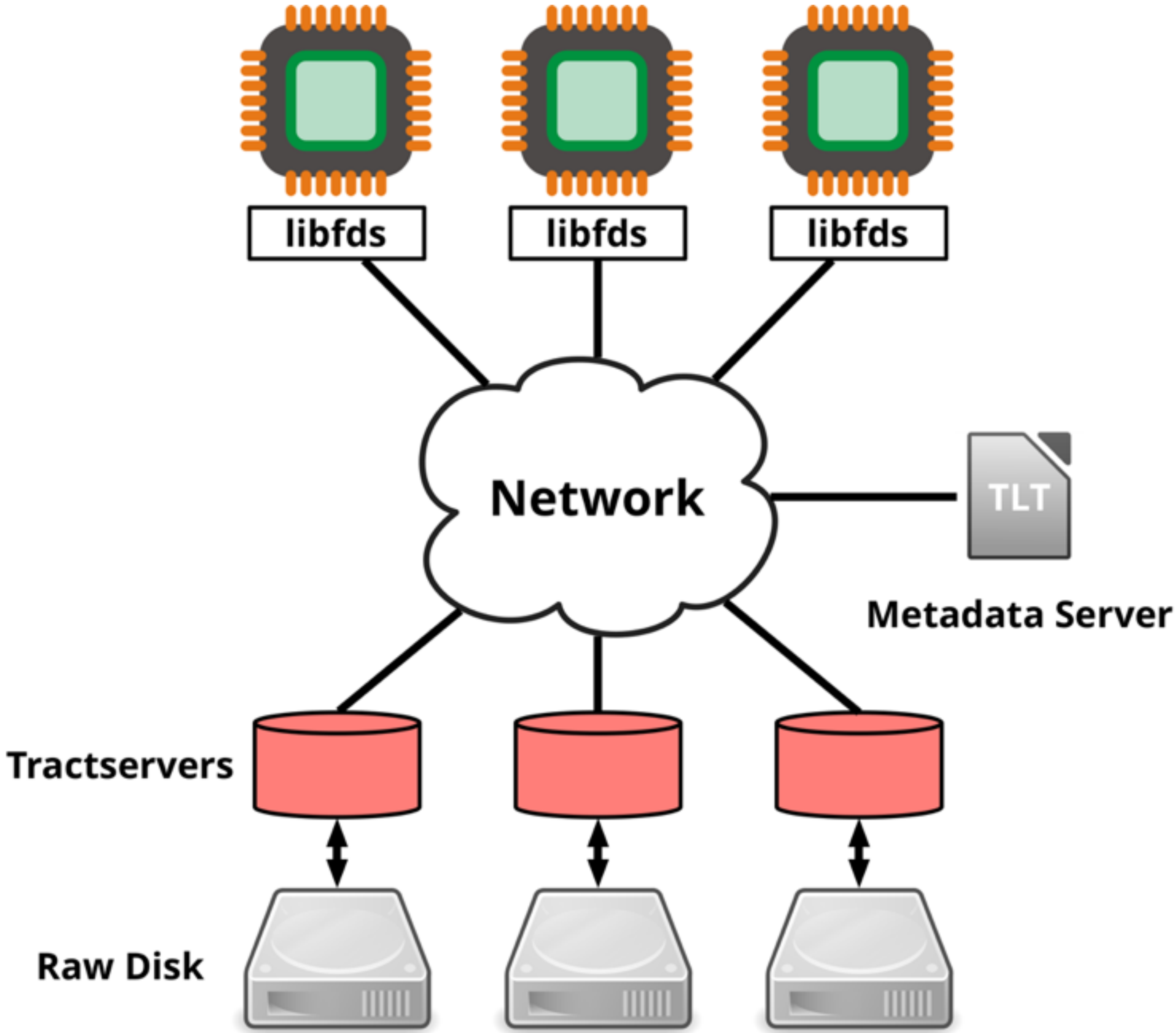
Object Storage

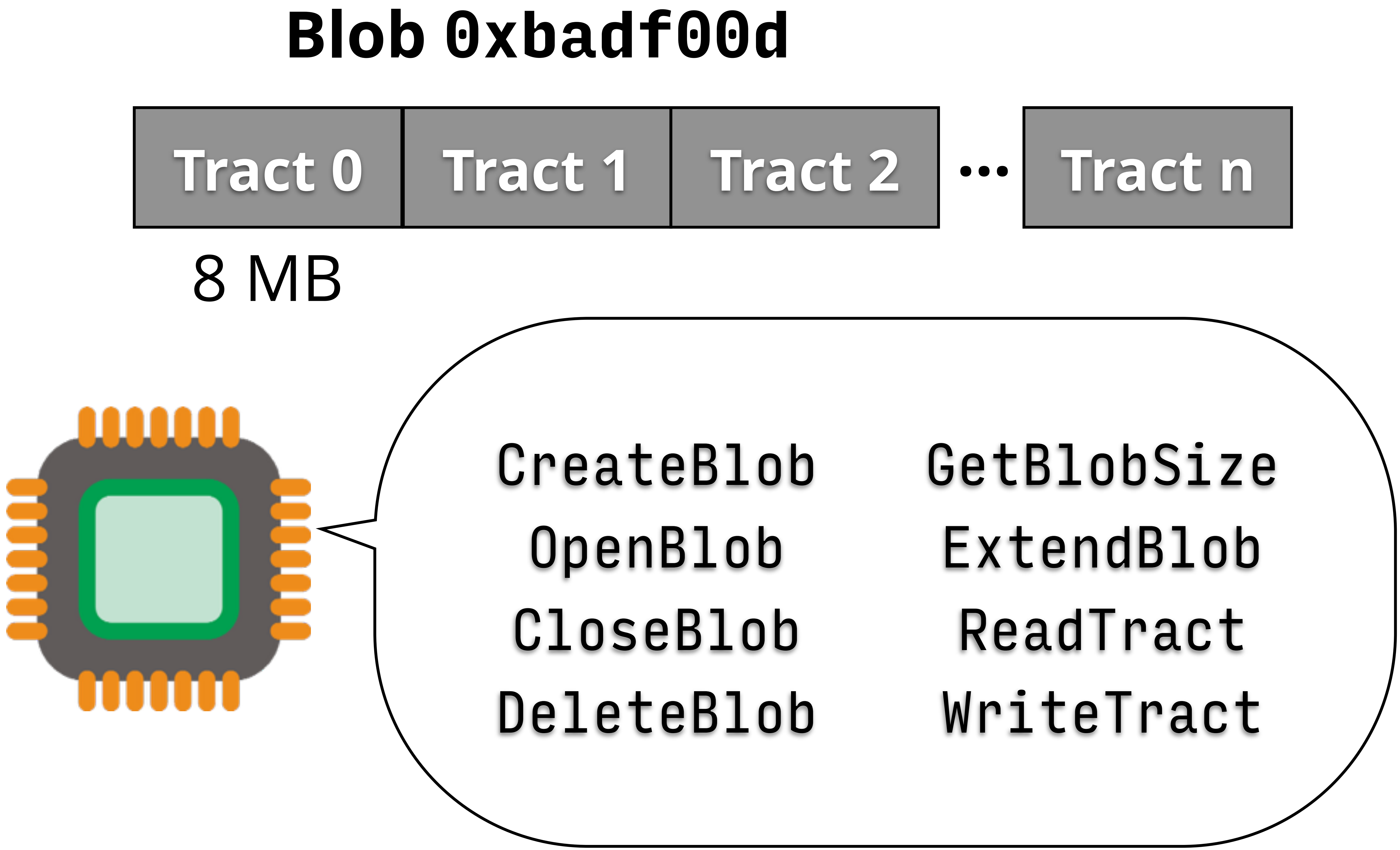
Assuming

No Oversubscription

Outline

- Introduction
- Architecture and API
- Metadata management
- Replication and Recovery
- Network
- Evaluation
- Discussion
- One-minute plug


libfds
libfds
libfds
Network
TLT
Metadata Server
Tractservers
Raw Disk


Blob 0xbadf00d
Tract 0
Tract 1
Tract 2
...
Tract n
8 MB
CreateBlob
OpenBlob
CloseBlob
DeleteBlob
GetBlobSize
ExtendBlob
ReadTract
WriteTract

API Guarantees

- Tractserver writes are **atomic**
- Calls are **asynchronous**
  - Allows deep pipelining
- **Weak consistency** to clients

Outline

- Introduction
- Architecture and API
- Metadata management
- Replication and Recovery
- Network
- Evaluation
- Discussion
- One-minute plug

Tract Locator Table

Tract Locator	Version	TS
1	0	A
2	0	B
3	2	D
4	0	A
5	3	C
6	0	F
...	...	...

```
Tract_Locator =
TLT[(Hash(GUID) + Tract) % len(TLT)]
```

```
Tract_Locator =
TLT[(Hash(GUID) + Tract) % len(TLT)]
```

Randomize blob’s tractserver,
even if GUIDs aren’t random
(uses SHA-1)

```
Tract_Locator =
TLT[(Hash(GUID) + Tract) % len(TLT)]
```

Large blobs use all TLT entries uniformly

```
Tract_Locator =
TLT[(Hash(GUID) - 1) % len(TLT)]
```

Blob Metadata is Distributed

Cluster Growth

Tract Locator	Version	TS
1	0	A
2	0	B
3	2	D
4	0	A
5	3	C
6	0	F
...	...	...

Cluster Growth

Tract Locator	Version	TS
1	1	NEW / A
2	0	B
3	2	D
4	1	NEW / A
5	4	NEW / C
6	0	F
...	...	...

Cluster Growth

Tract Locator	Version	TS
1	2	NEW
2	0	A
3	2	A
4	2	NEW
5	5	NEW
6	0	A
...	...	...

Outline

- Introduction
- Architecture and API
- Metadata management
- Replication and Recovery
- Network
- Evaluation
- Discussion
- One-minute plug

Replication

- For both *fault-tolerance* and *availability*
- Supports variable replication factors for different blobs
  - 1-replica for intermediate computations, 3 replicas for archival data and over-replicate popular blobs
  - replication factor stored in the blob meta data

Replication

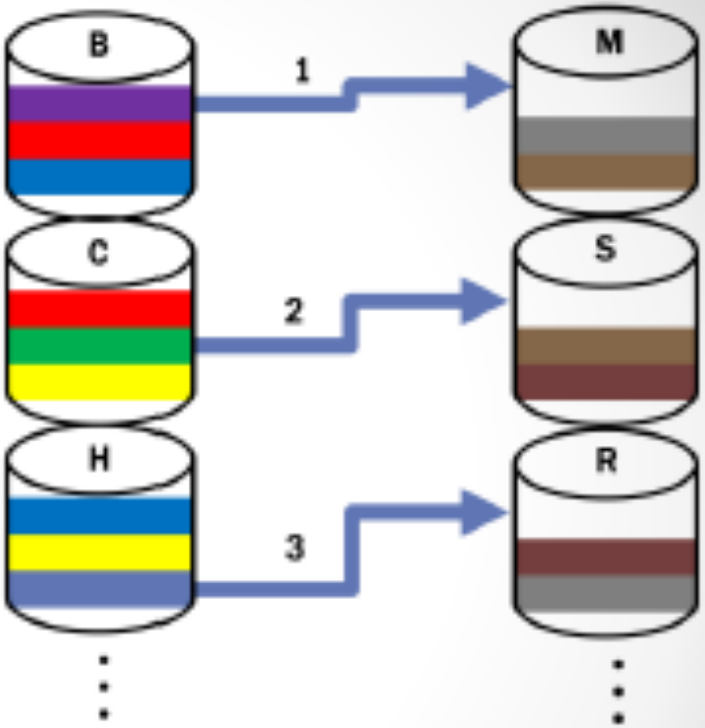
Tract Locator	Version	Replica 1	Replica 2	Replica 3
1	0	A	B	C
2	0	A	C	Z
3	0	A	D	H
4	0	A	E	M
5	0	A	F	G
6	0	A	G	P
...	...	...	...	...

Replication

- `Create`, `Delete`, `Extend`:
  - client writes to primary
  - primary 2PC to replicas
- `Write` to all replicas
- `Read` from random replica

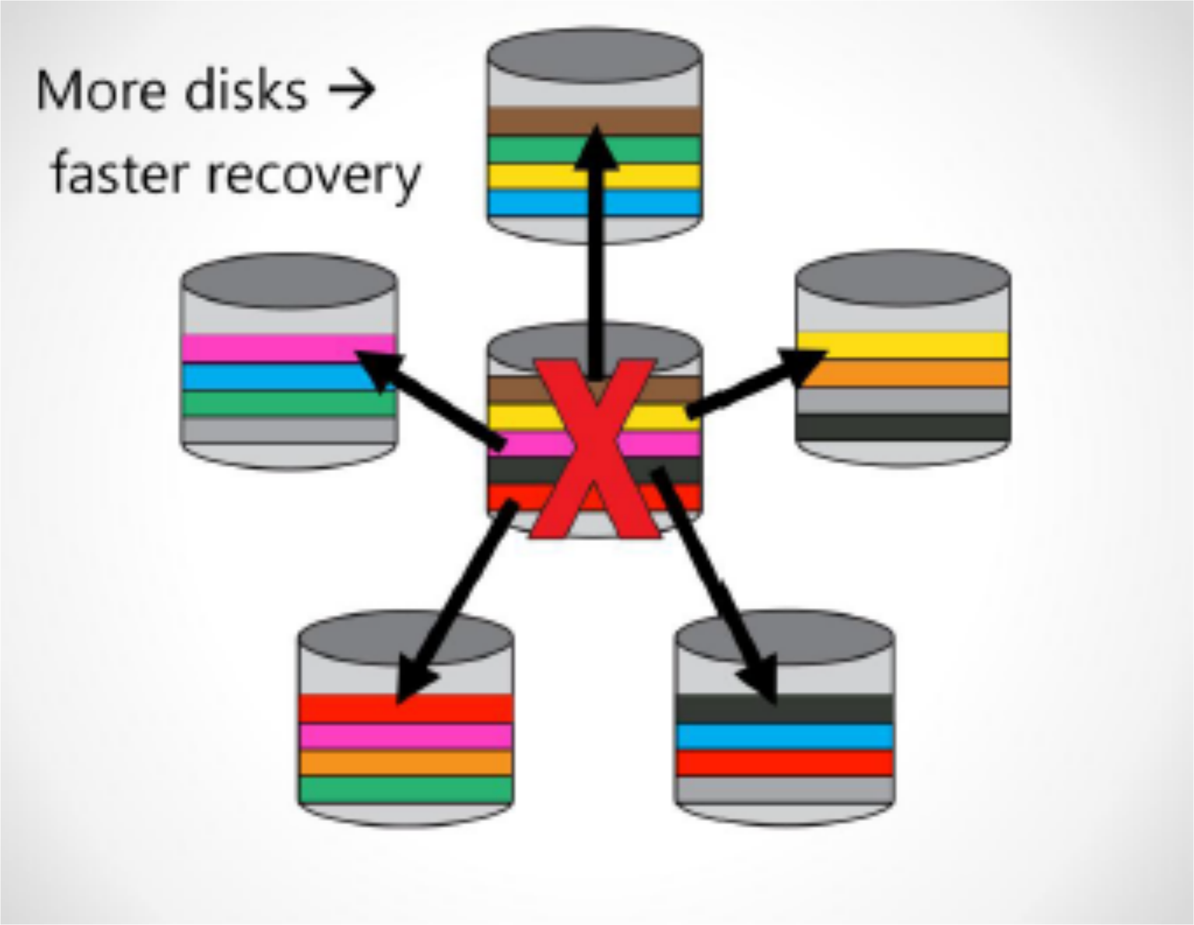
Recovery

Locator	Disk 1	Disk 2	Disk 3
1	M	B	C
2	S	C	Z
3	R	D	H
4	D	E	M
5	S	F	G
6	N	G	P
...	...	...	...
648	Z	W	H
649	Z	X	L
650	Z	Y	C



- All **disk pairs** appear in the table
- *n* disks each recover 1/*n*th of the lost data in parallel

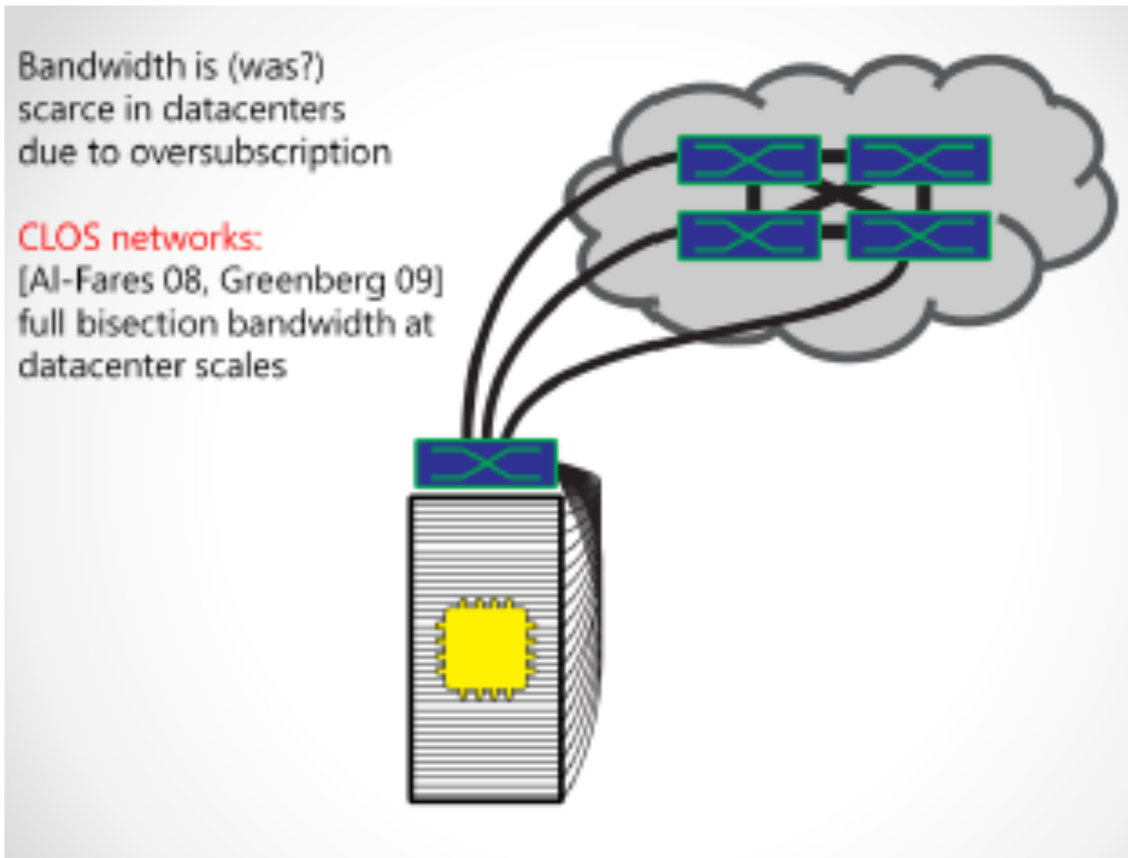
Recovery



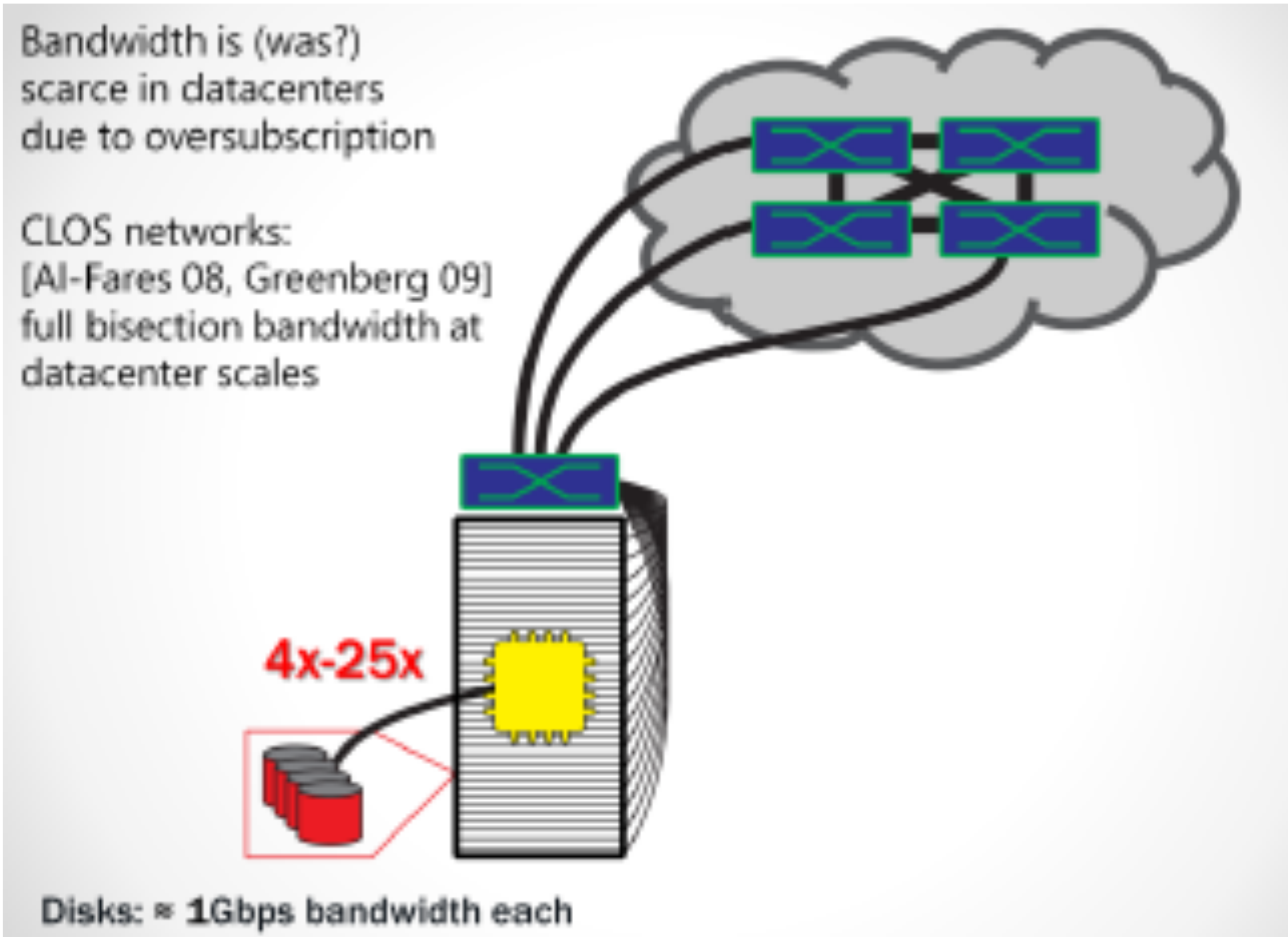
Outline

- Introduction
- Architecture and API
- Metadata management
- Replication and Recovery
- Network
- Evaluation
- Discussion
- One-minute plug

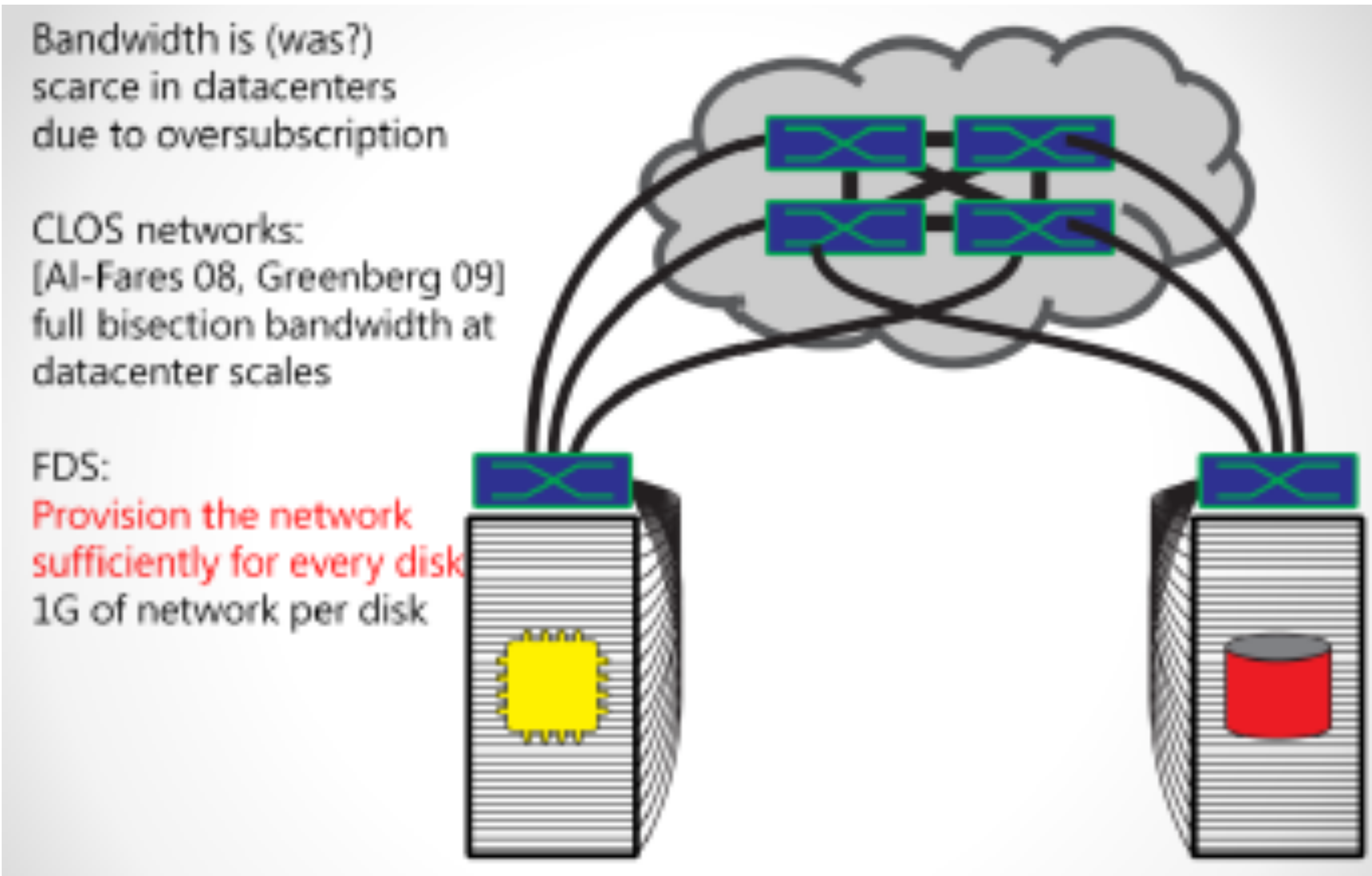
How to make network not a bottleneck?



How to make network not a bottleneck?



How to make network not a bottleneck?

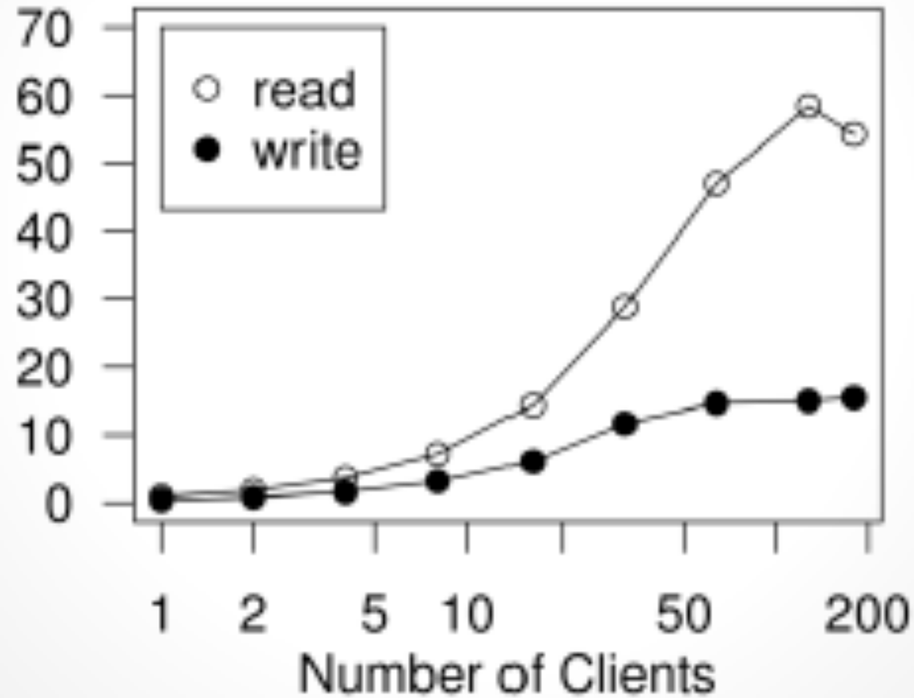


Outline

- Introduction
- Architecture and API
- Metadata management
- Replication and Recovery
- Network
- Evaluation
- Discussion
- One-minute plug

Read/Write Performance

Triple-Replicated Tractservers, 10G Clients



Failure Recovery Results

Disks in Cluster	Disks Failed	Data Recovered	Time
100	1	47 GB	19.2 ± 0.7s
1,000	1	47 GB	3.3 ± 0.6s
1,000	1	92 GB	6.2 ± 6.2s
1,000	7	655 GB	33.7 ± 1.5s

- We recover at about 40 MB/s/disk + detection time
- 1 TB failure in a 3,000 disk cluster: ~17s

High Application Performance: Minute Sort

MinuteSort—Daytona class (general purpose)					
FDS, 2012	256	1,033	1,401 GB	59 s	46 MB/s
Yahoo!, Hadoop, 2009 [25]	1,408	5,632	500 GB	59 s	3 MB/s

15x efficiency improvement!

Outline

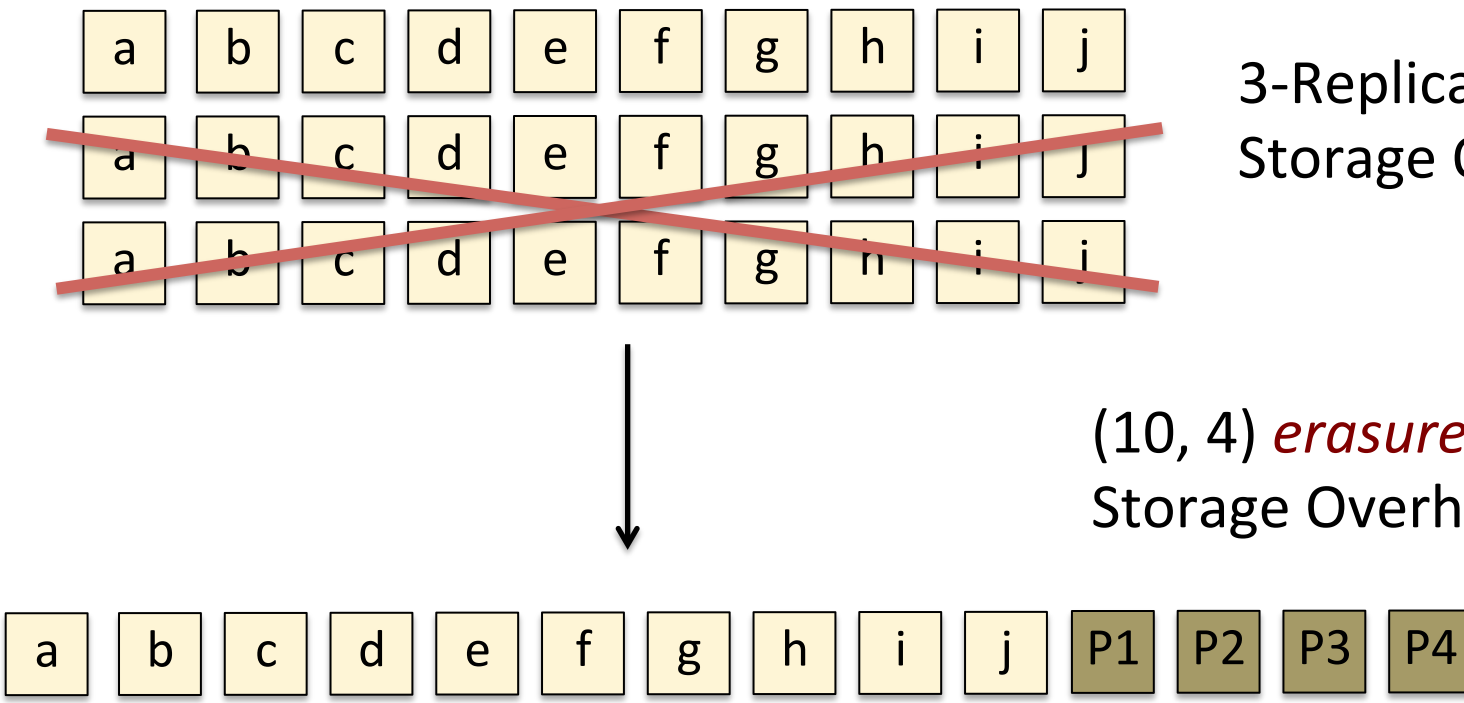
- Introduction
- Architecture and API
- Metadata management
- Replication and Recovery
- Network
- Evaluation
- Discussion
- One-minute plug

Discussion

- Is the problem real? Why different?
  - Yes (a clean slate design when BW not a bottleneck)
  - A new combination of system assumptions (full bisection BW) + workload (blob storage)

- Influential in 10 years? Yes
  - Increasing popularity of object/blob stores and feasibility of full bisection bandwidth networks
  - SSDs will allow much finer striping

Project: *Erasure coding for better performance*

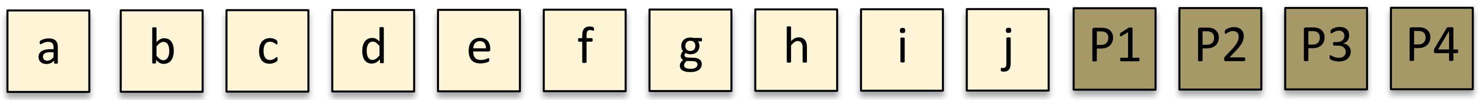


3-Replication
Storage Overhead: 3x

(10, 4) *erasure code*
Storage Overhead: 1.4x

- Any 10 units sufficient
- Can tolerate any 4-failures

Many properties: useful beyond fault tolerance



- *Load balance* by randomly choosing 10 units
- *Straggler mitigation* by connecting to > 10 and using the first 10 to respond

Help reining in *tail latencies* or in *increasing throughput* for skewed workloads

Talk to me or send me an email if you are interested in this research project (rashmikv@eecs)

Thanks!