Scalable Data Management and Storage on the Cloud:

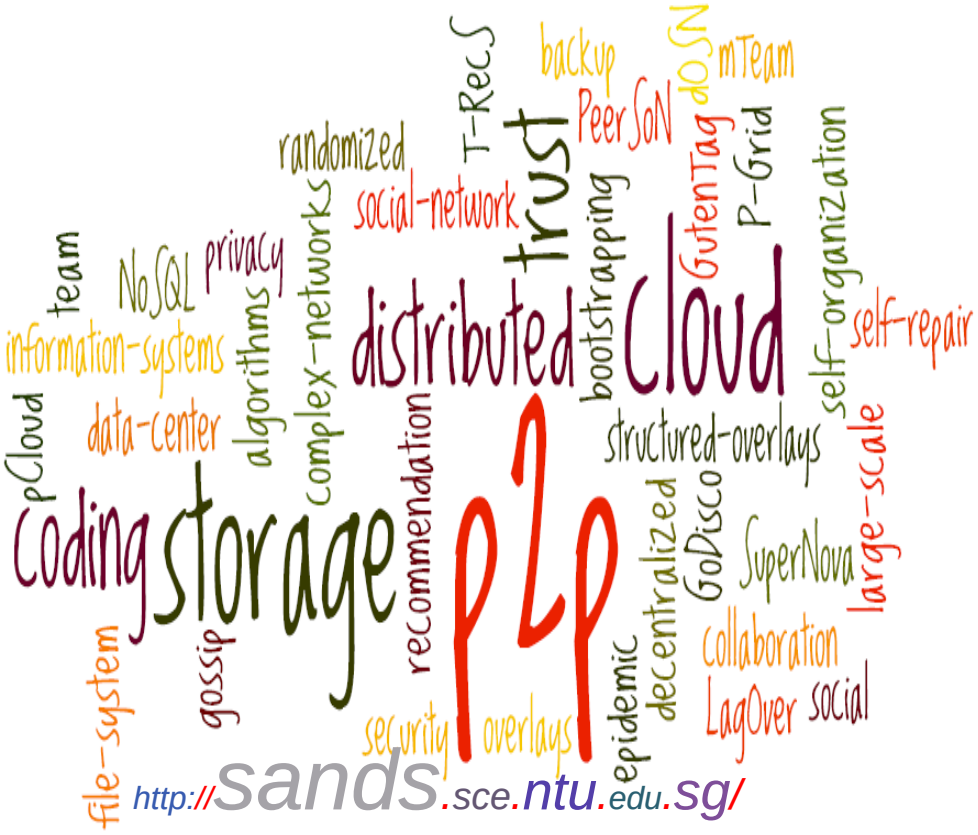
- State of the Art and Emerging Trends

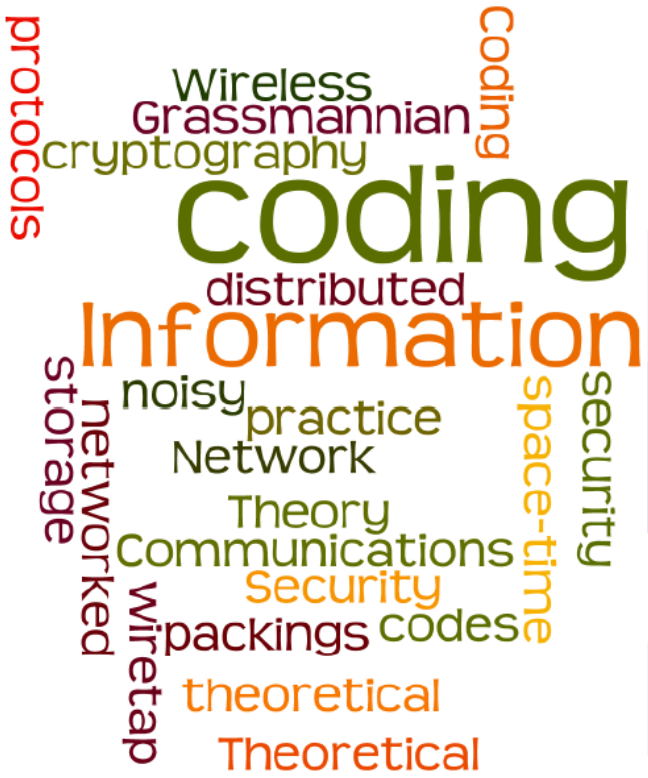
Presented by

Anwitaman Datta & Frédérique Oggier
NTU Singapore

ICDCN 2012, Hong Kong

Who are we?





Anwitaman Datta
School of Computer Engineering
NTU Singapore

Frédérique Oggier
School of Physical and Mathematical Sciences
NTU Singapore

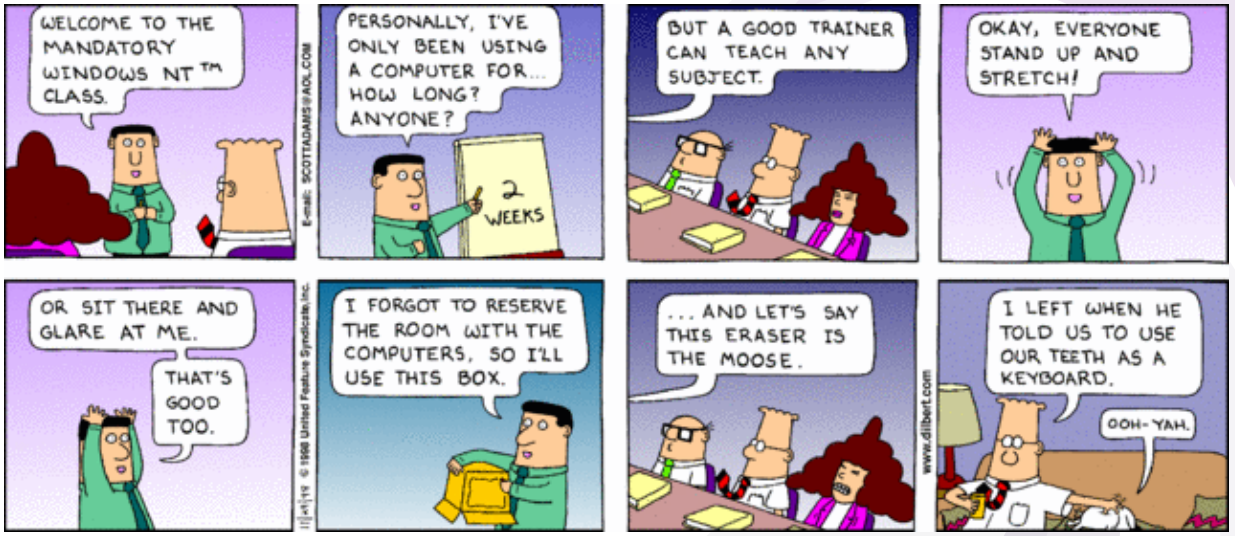
© 2012 A. Datta & F. Oggier, NTU Singapore

Who are we? Disclaimer

A note from the trenches: "You know you have a large storage system when you get paged at 1 AM because you only have a few petabytes of storage left." – from Andrew Fikes' (Principal Engineer, Google) faculty summit talk ` Storage Architecture and Challenges `, 2010.

… and some ask/say: why do you care about efficient storage space utilization, it is so cheap ...

We two never get such calls!!



© 2012 A. Datta & F. Oggier, NTU Singapore

Outline

- Distributed storage systems
  - Scale how much? Scale how?
  - From P2P to Data centers
- Data storage & management
  - NoSQL systems
    - Detailed examples: Dynamo, GFS, …
- Erasure coding based efficient fault-tolerant storage
  - Background: RAID, (traditional) erasure codes
  - Codes tailor-made for storage
    - Detailed examples: Regenerating codes, Self-repairing codes, …
- Appendix: MapReduce

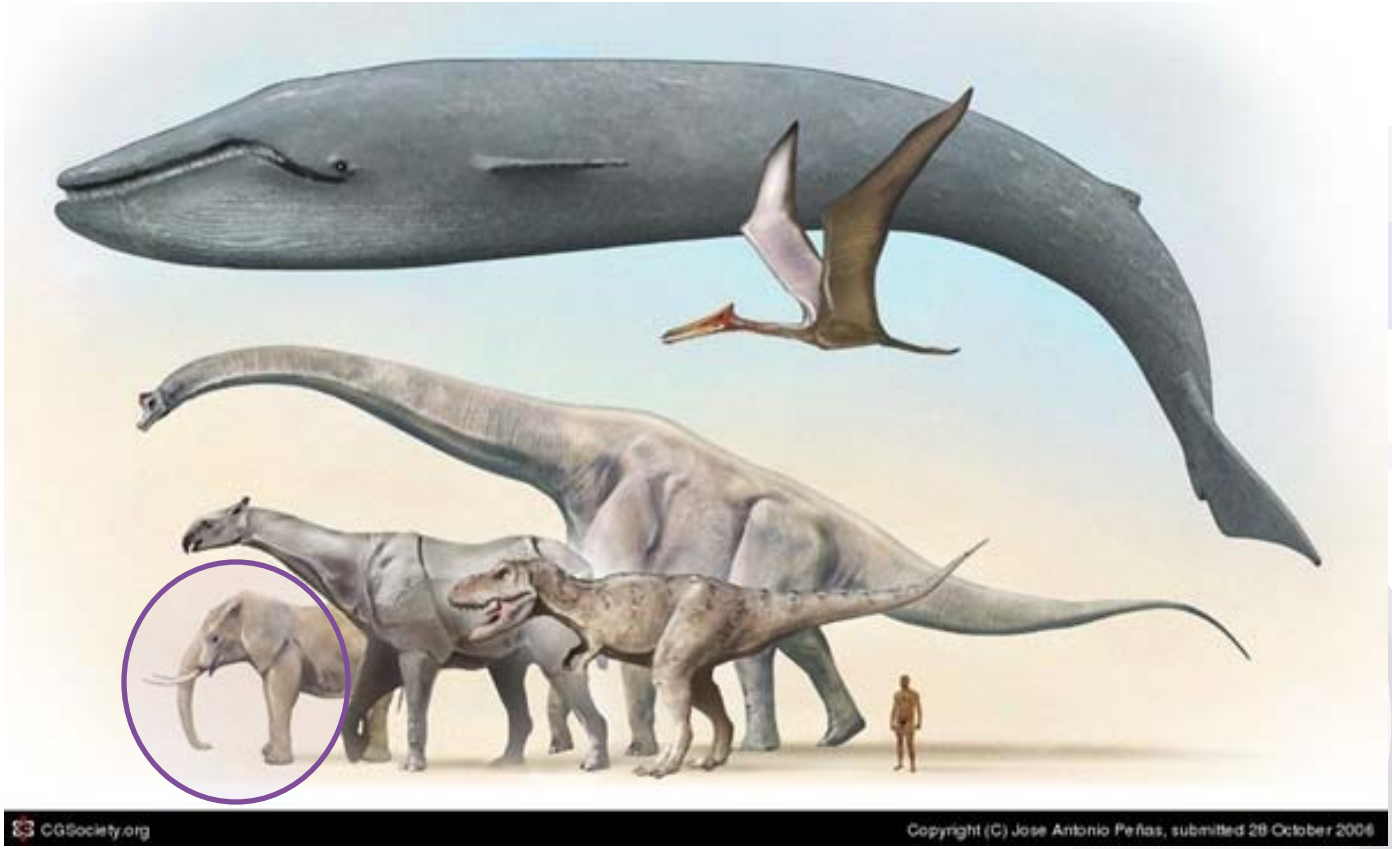
The "aging" state-of-the-art

An emerging trend …

© 2012 A. Datta & F. Oggier, NTU Singapore

What scale are we talking about?

- Rather big …



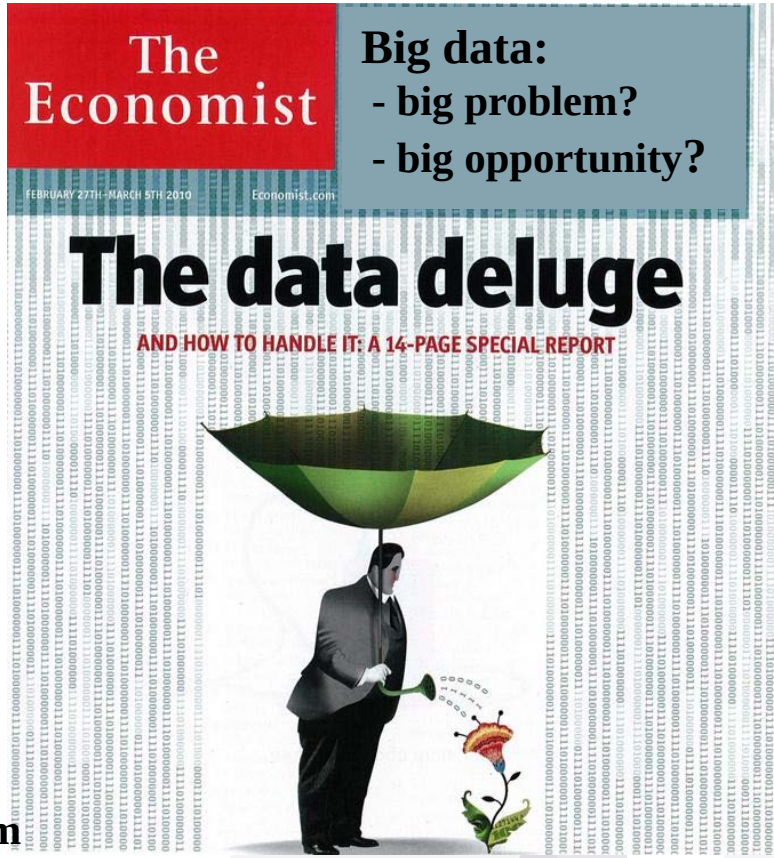
© 2012 A. Datta & F. Oggier, NTU Singapore

Big data

- June 2011 **EMC²** study
  - world's data is ***more than doubling*** every 2 years
    - faster than Moore's Law
  - 1.8 ***zettabytes*** of data to be created in 2011

Zetta: 10^{21}

Zettabyte: If you stored all of this data on DVDs, the stack would reach from the Earth to the moon and back.



*** http://www.emc.com/about/news/press/2011/20110628-01.htm**

© 2012 A. Datta & F. Oggier, NTU Singapore

The data deluge: Some numbers

- **Facebook** "currently" (in 2010) stores over **260 billion images**, which translates to over 20 petabytes of data. Users upload *one billion new photos (60 terabytes) each week* and Facebook *serves over one million images per second at peak*. [Beaver et al. in "Haystack" paper]
- On "Saturday", **photo number four billion** was uploaded to photo sharing site **Flickr**. This comes *just five and a half months after the 3 billionth* and nearly 18 months after photo number two billion. – Mashable (13th October 2009)

  [http://mashable.com/2009/10/12/flickr-4-billion/]

© 2012 A. Datta & F. Oggier, NTU Singapore

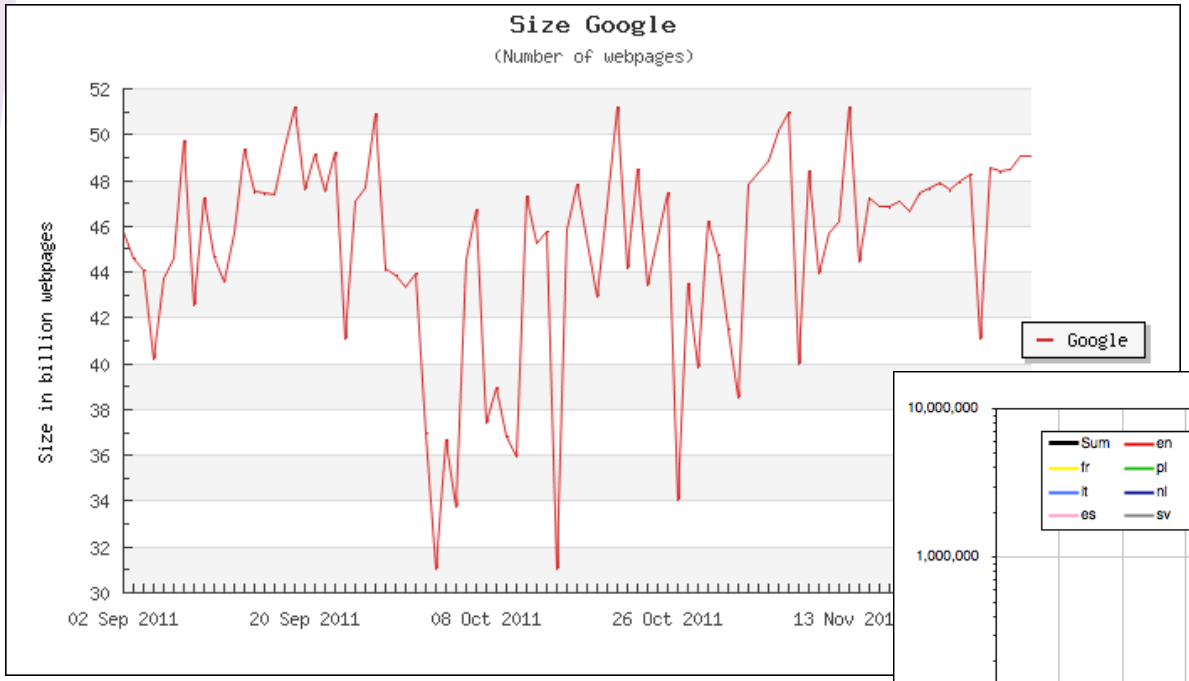
The data deluge: Some numbers

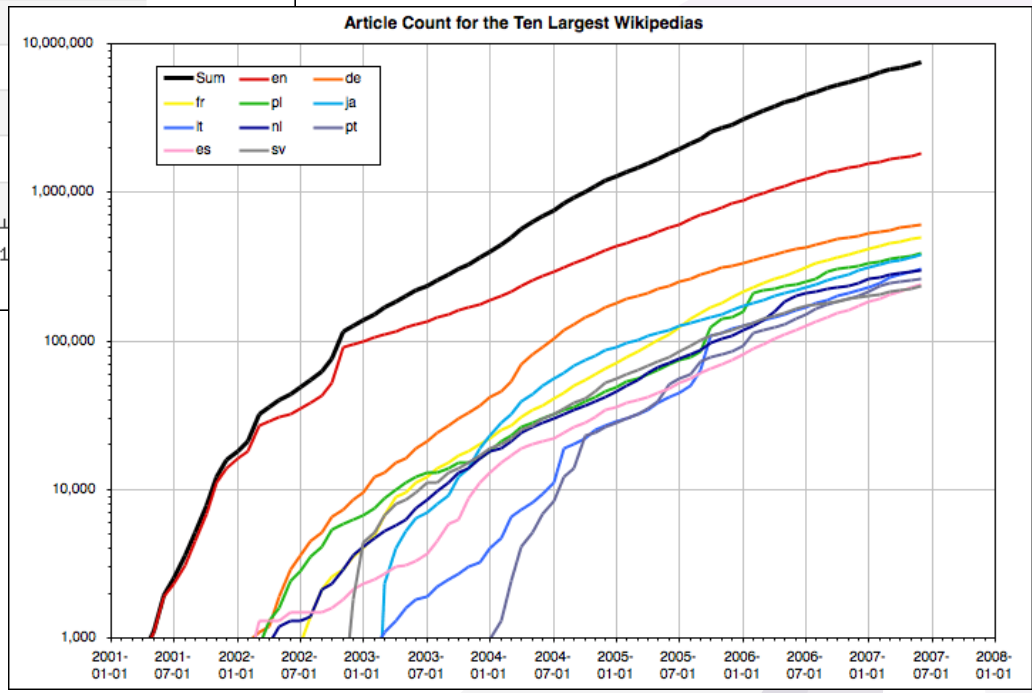
- Caffeine lets us index web pages on an enormous scale. In fact, every second Caffeine *processes hundreds of thousands of pages in parallel*. If this were a pile of paper it would *grow three miles taller every second*. Caffeine takes up nearly *100 million gigabytes of storage in one database* and adds new information at a rate of **hundreds of thousands of gigabytes per day**. You would need 625,000 of the largest iPods to store that much information; if these were stacked end-to-end they would go for more than 40 miles.

http://**google**blog.blogspot.com/2010/06/our-new-search-index-caffeine.html

© 2012 A. Datta & F. Oggier, NTU Singapore

The data deluge: Some numbers





© 2012 A. Datta & F. Oggier, NTU Singapore

Scale how?

To scale vertically (or **scale up**) means to add resources to a single node in a system*

Scale up

To scale horizontally (or **scale out**) means to add more nodes to a system, such as adding a new computer to a distributed software application*

Scale out

© 2012 A. Datta & F. Oggier, NTU Singapore

* Definitions from Wikipedia

Distribution is essential

- Scaling up
  - May just not even be feasible
  - Even if feasible, it will be very expensive
  - What happens when “the” machine fails?
- Scaling out => distributed storage
  - Distribution => added complexity and vulnerabilities
    - latency, consistency, faults, …
    - CAP theorem
      - Consistency, Availability, Partition tolerance – choose any two
    - but, not distributing is not a choice!

© 2012 A. Datta & F. Oggier, NTU Singapore

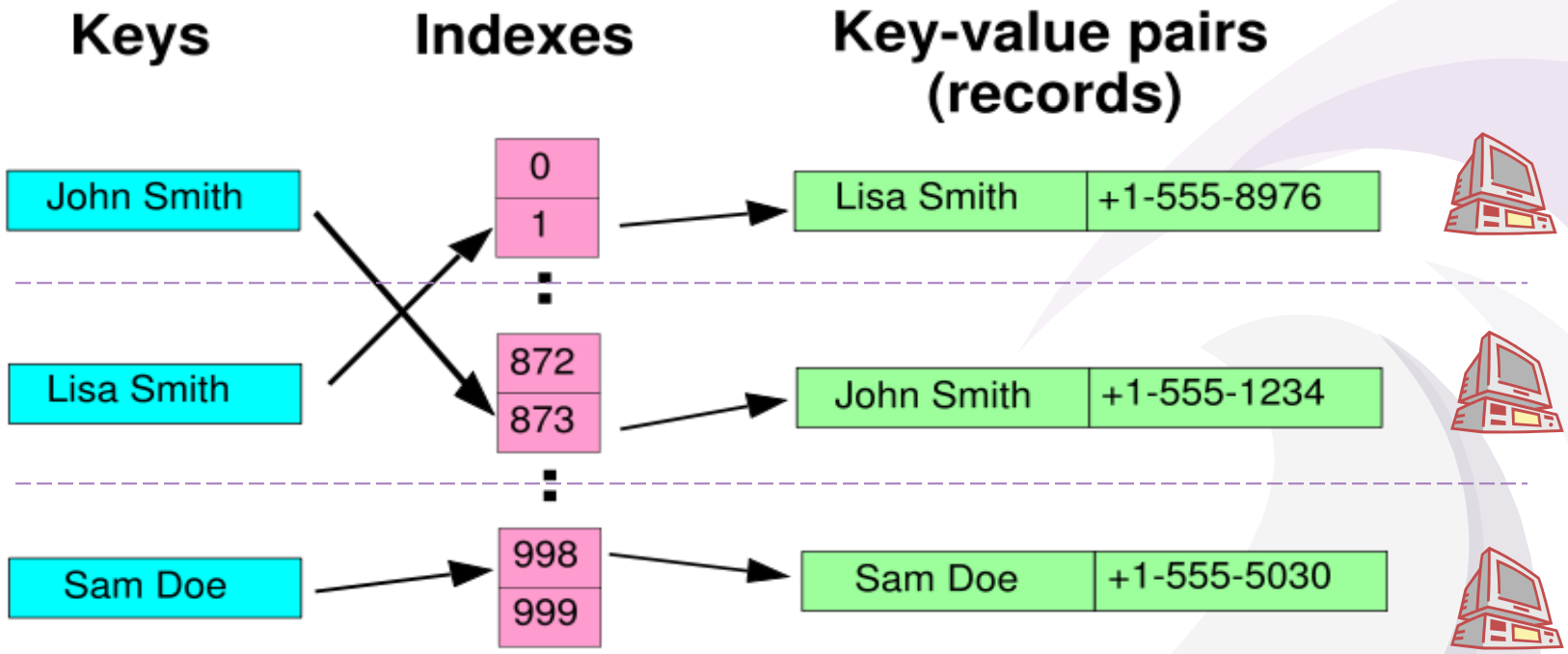
Distributed Storage Systems

- Different flavors of functionality
  - File level data storage: Network-attached storage
  - Block level data storage: Storage area network
  - Distributed databases
  - Caching/CDNs
  - …
- Different flavors of architecture
  - Centralized, decentralized/peer-to-peer, hybrid …
- Different flavors of interfaces
  - depends on/constrains functionality

© 2012 A. Datta & F. Oggier, NTU Singapore

Key-Value Stores

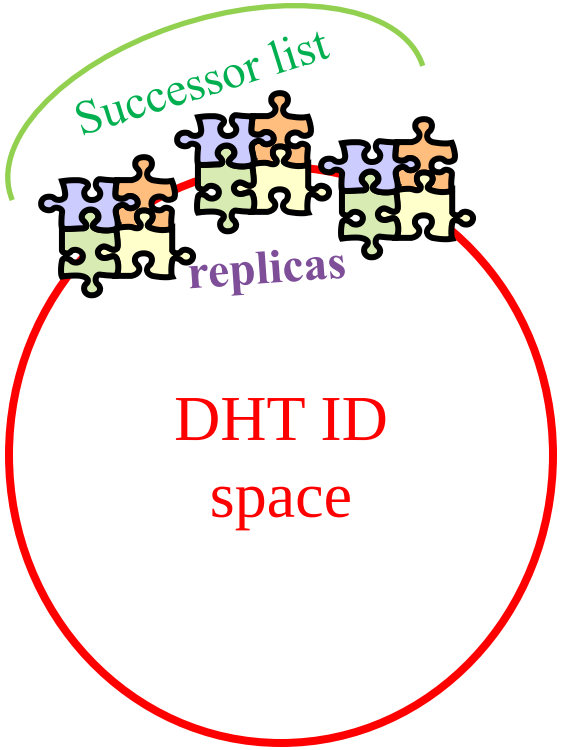
- Popularized by distributed hash tables
  - Basic operations: Get, Put, …
  - Easy to distribute (partition the key space)



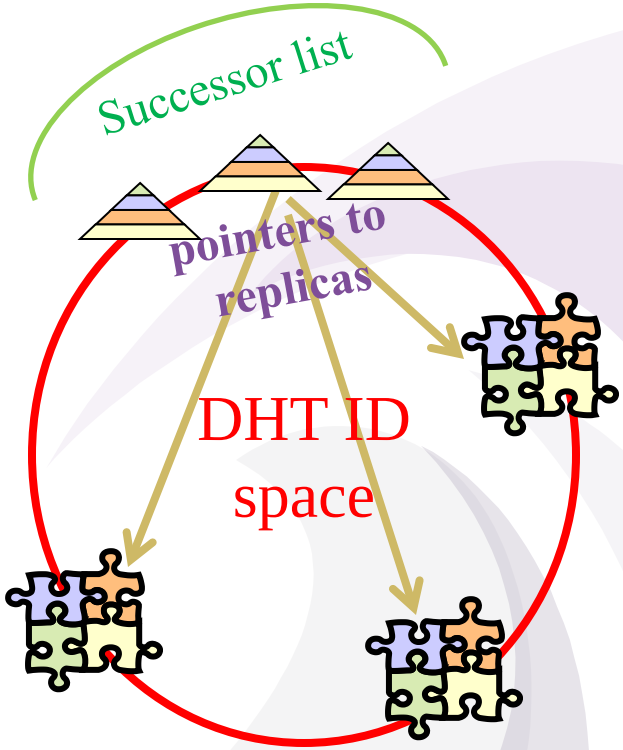
© 2012 A. Datta & F. Oggier, NTU Singapore

Distributed key-value store based storage

- DHT itself as the storage layer



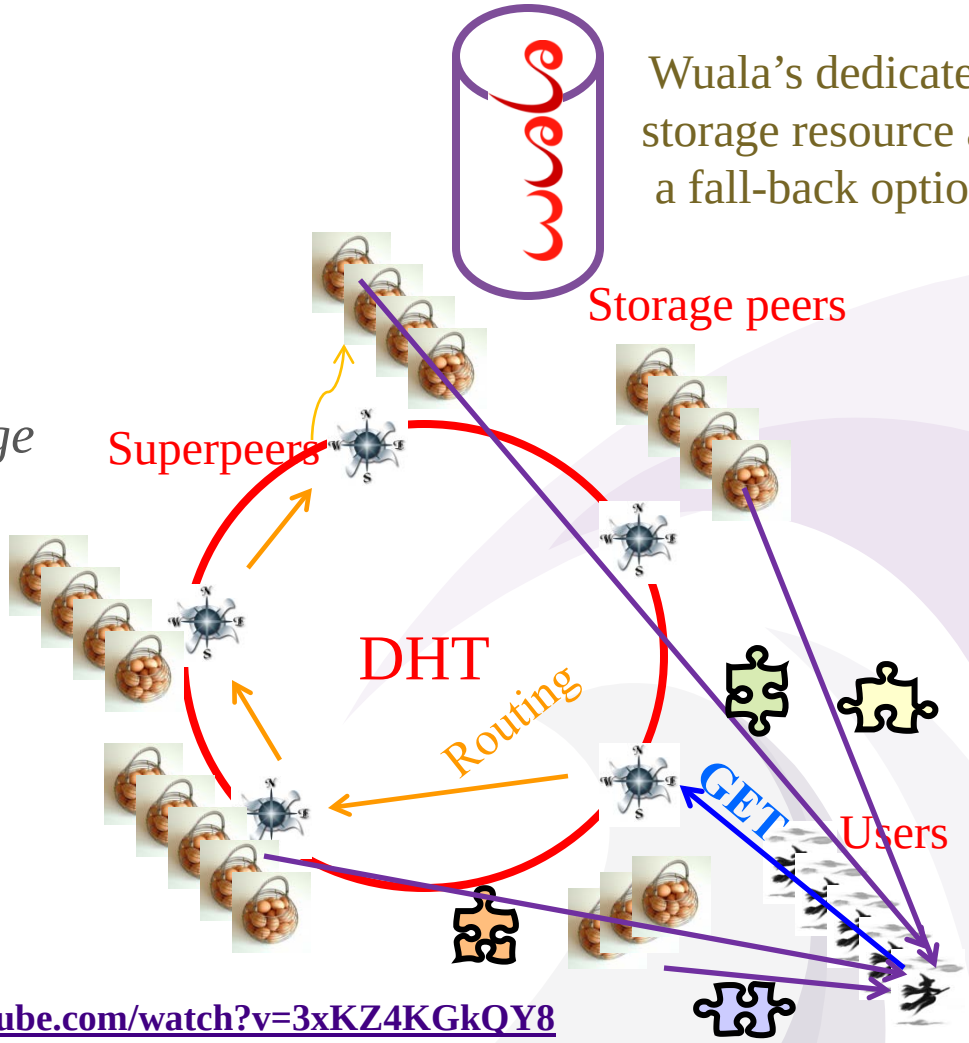
- DHT as a directory service



© 2012 A. Datta & F. Oggier, NTU Singapore

Wuala's 3-tier architecture

- Complete disentanglement of indexing and storage
- Many (encoded) fragments per object
  - Suitable for sharing very *large* but *static* files
  - Parallel download
- Piggy-backed, large DHT routing states
  - So very few hops needed, gives high through-put

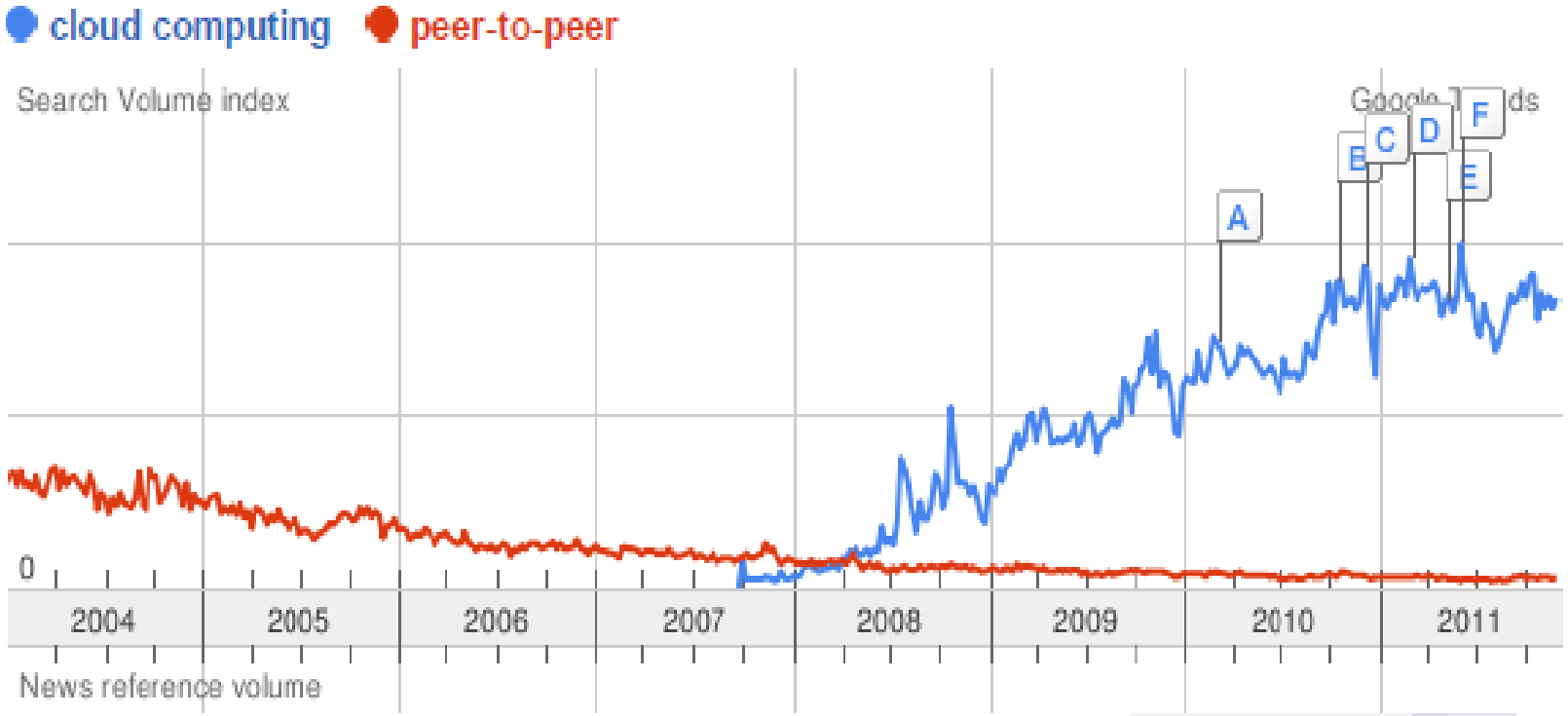


Source: Google tech talk on Wuala: http://www.youtube.com/watch?v=3xKZ4KGkQY8

© 2012 A. Datta & F. Oggier, NTU Singapore

Wait a moment!

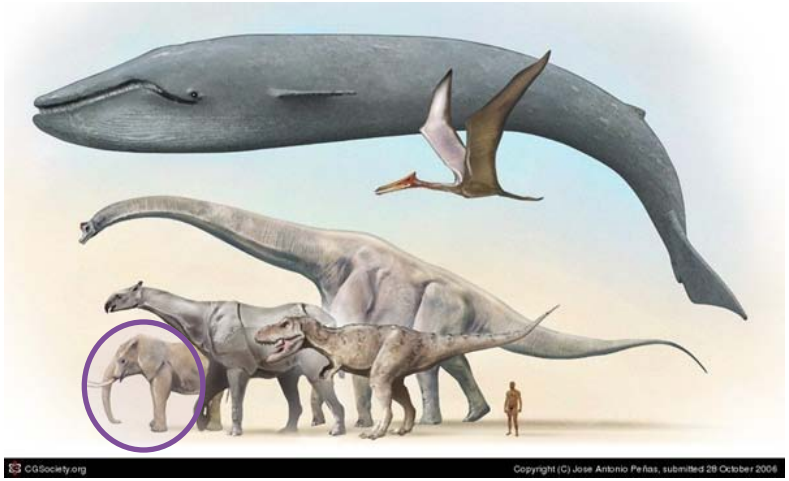
- We are not here to talk about P2P systems, the title mentioned "on the cloud"!
  - **Don't panic**! The fundamental things apply (even as time goes by …)



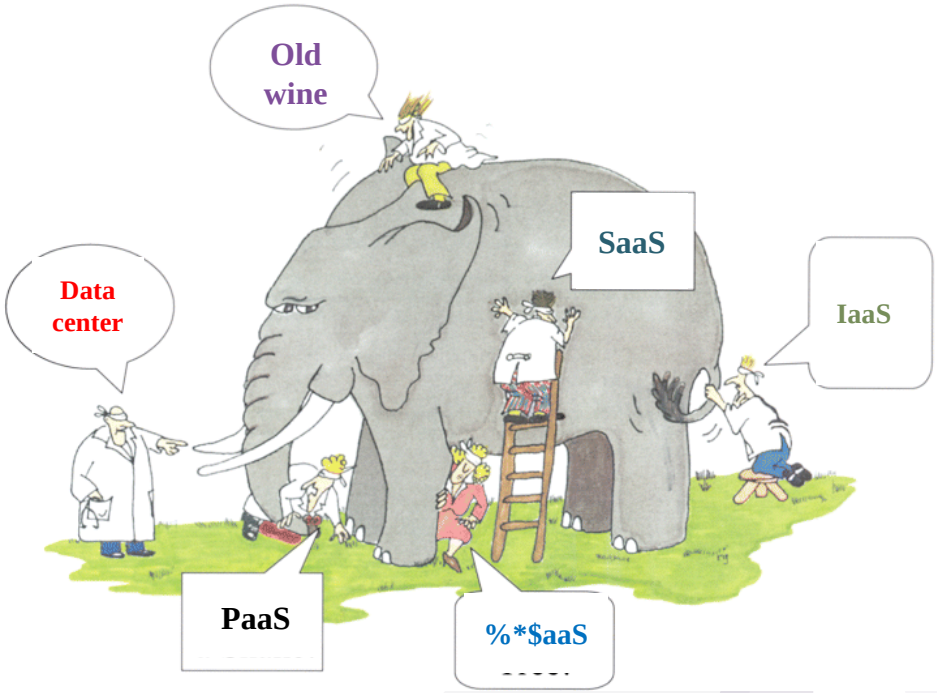
© 2012 A. Datta & F. Oggier, NTU Singapore

But, what is the cloud?

- At least, we can all agree
  - Cloud is something “big” and happening!
  - It’s all of these …
    - … and some more!



Systems of elephantine proportion needed



© 2012 A. Datta & F. Oggier, NTU Singapore

NIST definition for cloud computing

- Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction

© 2012 A. Datta & F. Oggier, NTU Singapore

Two sides of the cloud coin

- Outside view
  - A “single/exclusive” entity
    - Access through a “demilitarized zone” …
    - API based
    - Agnostic to multi-tenancy
  - Infinite/elastic resources
    - Pay per use, on-demand, …
  - Browser based access (often)
    - Anytime, anywhere, any device …

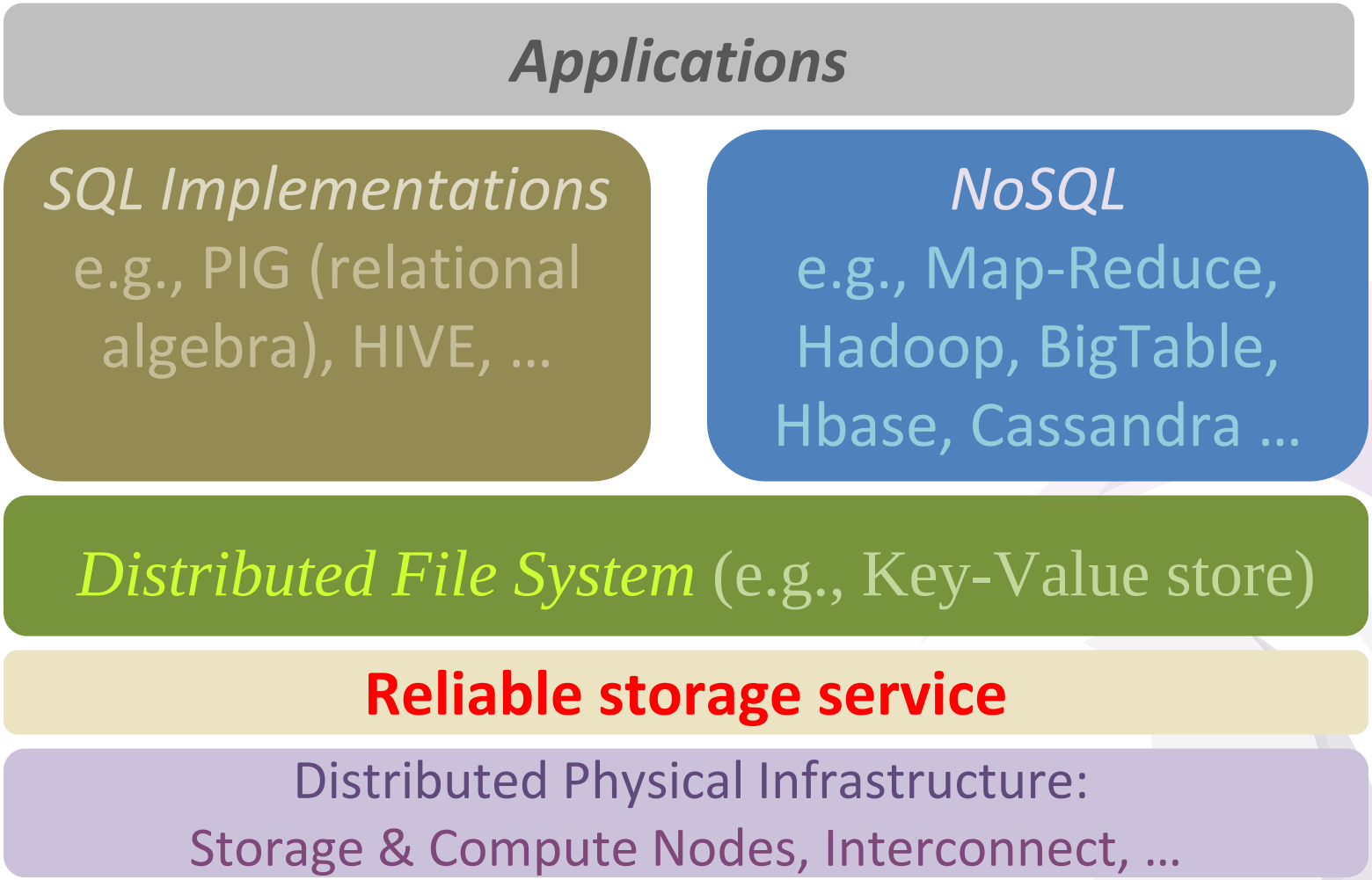
© 2012 A. Datta & F. Oggier, NTU Singapore

Two sides of the cloud coin

- Inside view
  - Pool of resources
    - In flux: New compute units joining, old ones retiring
    - Self-*: Load-balancing, fault-tolerance, auto-configuration, …
  - Multi-tenancy
    - Virtualization, transparent migration, …
  - Distributed file system, data-management, data processing
    - Google's GFS, Amazon's Dynamo, Facebook's Cassandra,Yahoo!'s Pnuts
    - Map Reduce/Hadoop, Pig, Chubby, …

© 2012 A. Datta & F. Oggier, NTU Singapore

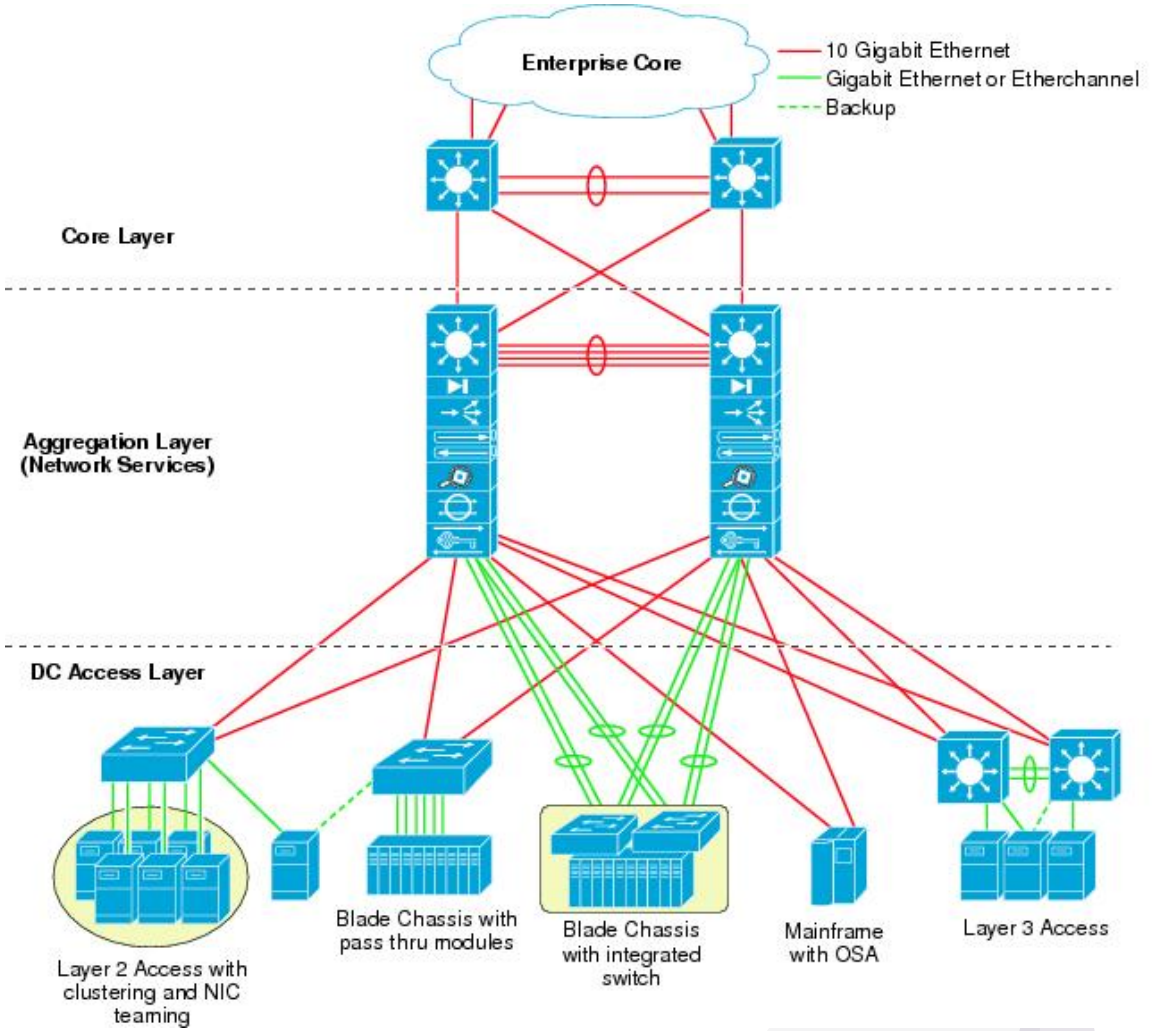
The new stack



Disclaimer: This "stack" is a personal "view", and is not universal

© 2012 A. Datta & F. Oggier, NTU Singapore

Data center



Source of topology: http://www.cisco.com/en/US/docs/solutions/Enterprise/Data_Center/vmware/VMware.html

Note: More recently, the trend is to deploy "Fat Tree" interconnects

© 2012 A. Datta & F. Oggier, NTU Singapore

Data center design evolution

Gen 1 DC
Collocation

Gen 2

Gen 3

Gen 4 (future)
Modular Data Center

Deployment Scale Unit

Server

Capacity

Rack

Density and Sustainability

Containers

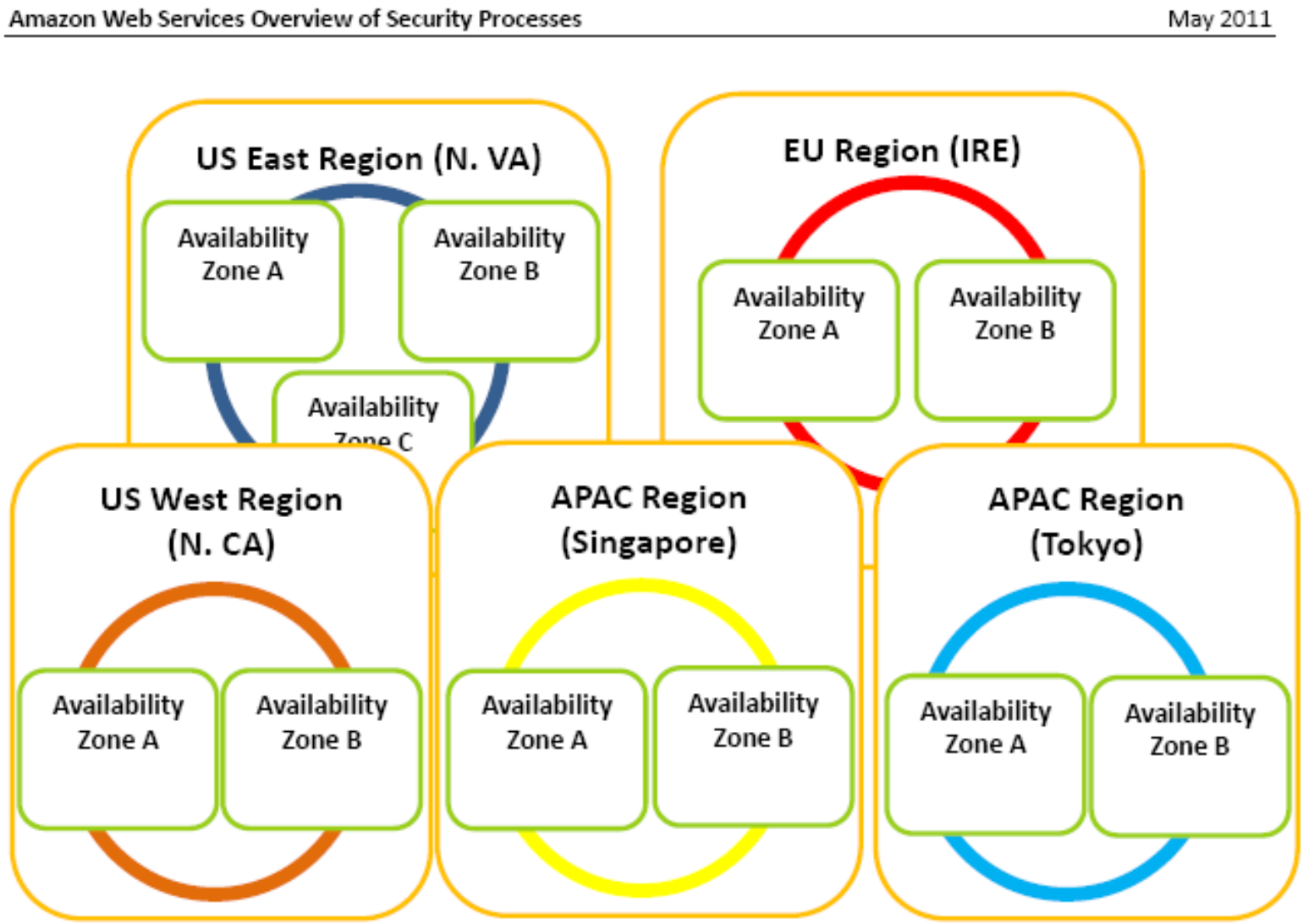
Scalability Thousands of Servers

Pre-Assembled Components

Right Time to Market, Lower TCO (PUE) Scalable Data Centers

Slide courtesy **Roger Barga** (Microsoft) from his P2P 2009 Keynote talk

Amazon's AWS: Availability zones



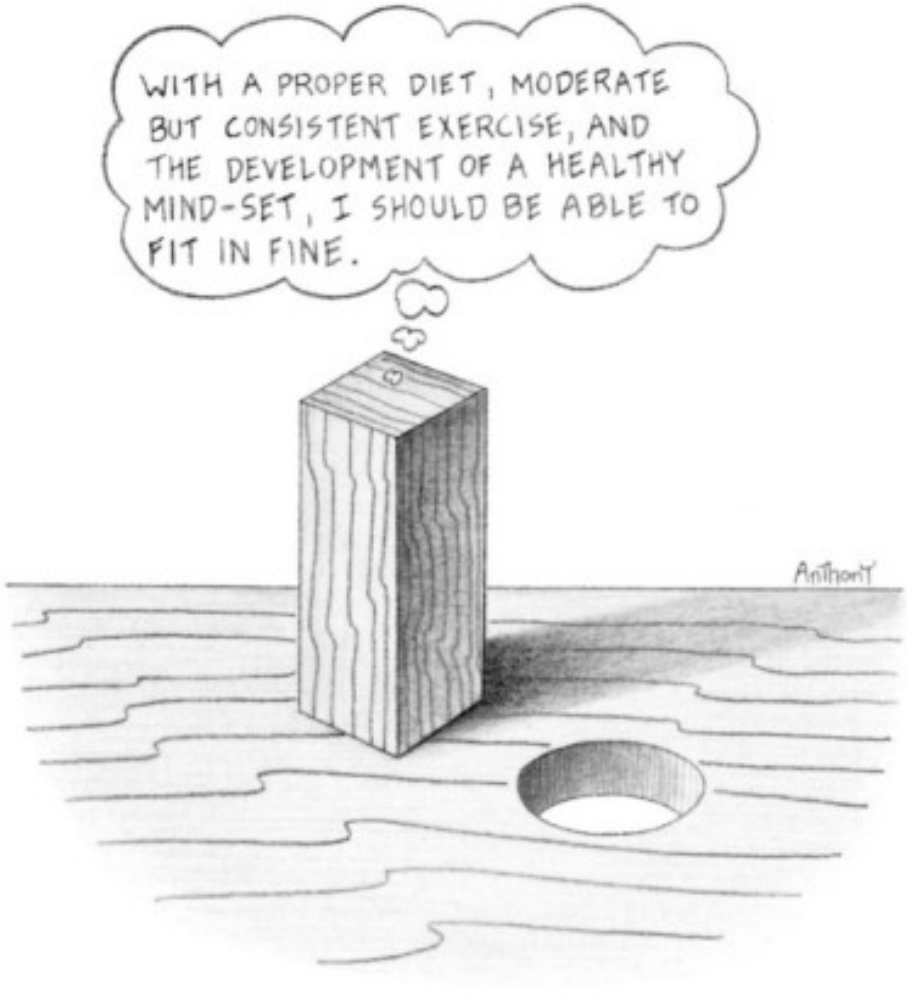
Build physical infrastructure to store immense data …

Still, how to manage so much data and distribution???

© 2012 A. Datta & F. Oggier, NTU Singapore

The “why” decides the “what” & the “how”

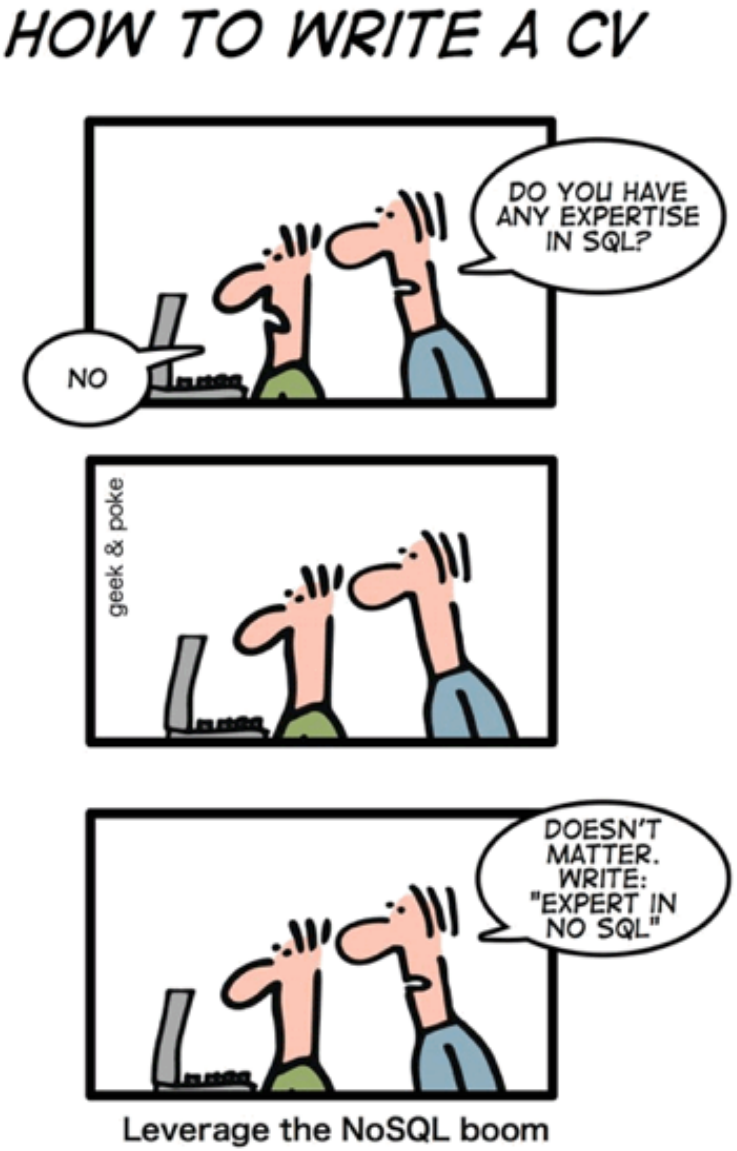
- Workload based design
  - One size does NOT fit all …



© 2012 A. Datta & F. Oggier, NTU Singapore

Not Only SQL

- Time tested RDBMS is
  - Neither always needed
  - Nor scales (out)
- Various workload specific custom storage & data management solutions
  - NoSQL



© 2012 A. Datta & F. Oggier, NTU Singapore

amazon.com

Dynamo

GFS, BigTable, MapReduce, …

HDFS, PNUTS, …

YAHOO!

Microsoft

Dryad, Azure, …

Cassandra, Haystack, …

facebook

© 2012 A. Datta & F. Oggier, NTU Singapore

In fact: Many varieties of NoSQL systems …

- Document store
- Graph
- Key-value store
- Multivalue databases
- Object database
- Tabular
- Tuple store
- …

© 2012 A. Datta & F. Oggier, NTU Singapore

Amazon's Key-Value Store: Dynamo

Dynamo: Amazon's Highly Available Key-Value Store

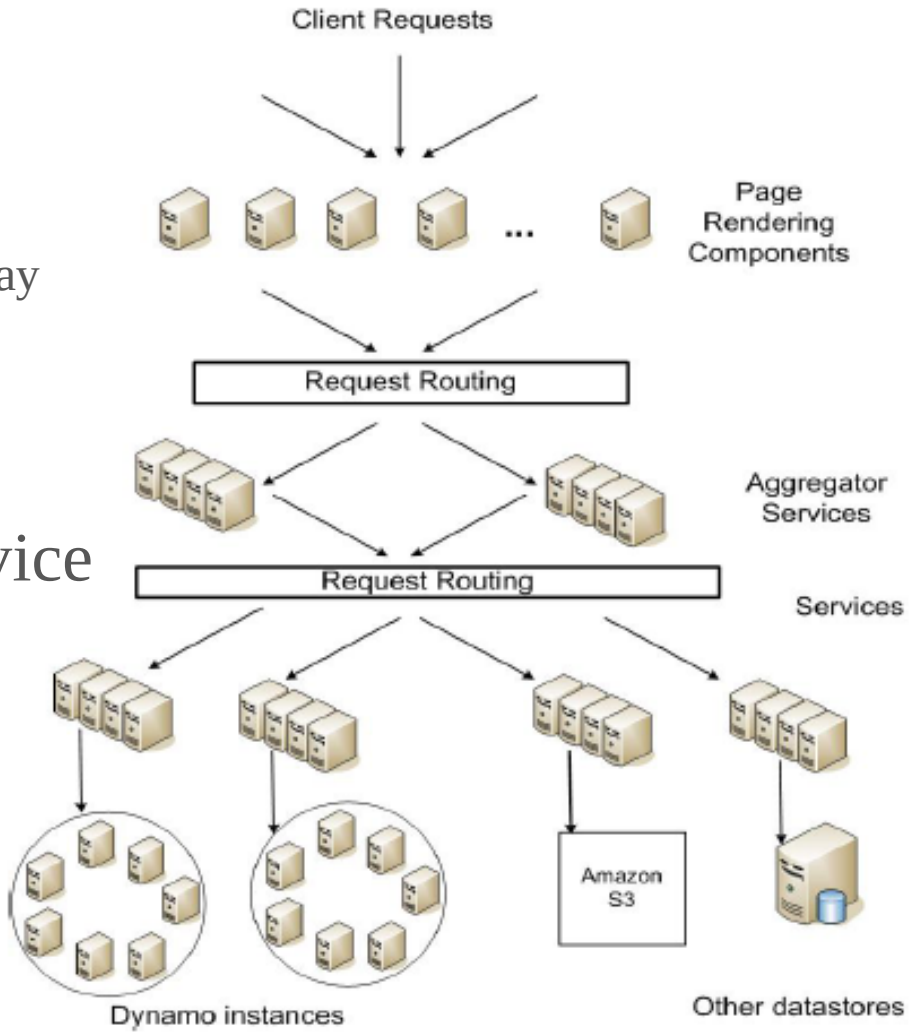
Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels

SOSP 2007

© 2012 A. Datta & F. Oggier, NTU Singapore

Fast and furious

- Latency sensitivity
  - Shopping Cart Service
    - 10s of millions of requests per day
    - Millions of checkouts each day
    - Hundreds of thousands of concurrent activities
- Stringent SLAs for each service
  - 99.9th percentile < 300ms
  - Mean/std. dev. inadequate
- One page request
  - 100s of services
    - Multiple service dependencies



© 2012 A. Datta & F. Oggier, NTU Singapore

The show must go on

- Extremely high availability
  - Down time => lost business
- “customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados”
- an infrastructure comprised of millions of components
  - tens of thousands of servers located across many data centers world-wide
  - a small but significant number of server and network components that are failing at any given time

The dreaded earthquake - enhanced tornado being hit by a radioactive meteor event.

© 2012 A. Datta & F. Oggier, NTU Singapore

The show must go on

- Infrastructure comprised of millions of components
  - tens of thousands of servers located across many data centers world-wide
  - a small but significant number of server and network components that are failing at any given time
- Redundancy needed for fault tolerance
  - Trade-off: Availability is more important than consistency
- An “always writeable” data store
  - Conflict resolution complexity at “reads”
    - Unlike most traditional data stores
    - Can be handled at clients (using application logic)
      - Data store provides some default fall-back option “last write wins”

© 2012 A. Datta & F. Oggier, NTU Singapore

KISS: Keep it simple, Stupid!

- Both stateless and stateful (needing persistent storage) services
- Most services access/store data using "primary keys"
  - No need for complex queries
  - No operation span multiple data items
  - Relatively small objects (<1MB)
- RDBMS is an overkill!
  - Also, difficult/impossible to scale-out
  - Needs much more expensive hardware/administration

© 2012 A. Datta & F. Oggier, NTU Singapore

Other embraced design principles

- Incremental scalability
- Symmetric
  - Simpler system provisioning & maintenance
- Decentralized
  - Self-*, no single point of failure, …
- Heterogeneity friendly

© 2012 A. Datta & F. Oggier, NTU Singapore

Dynamo architecture: Many considerations

Changes need to be well thought out!

© 2012 A. Datta & F. Oggier, NTU Singapore

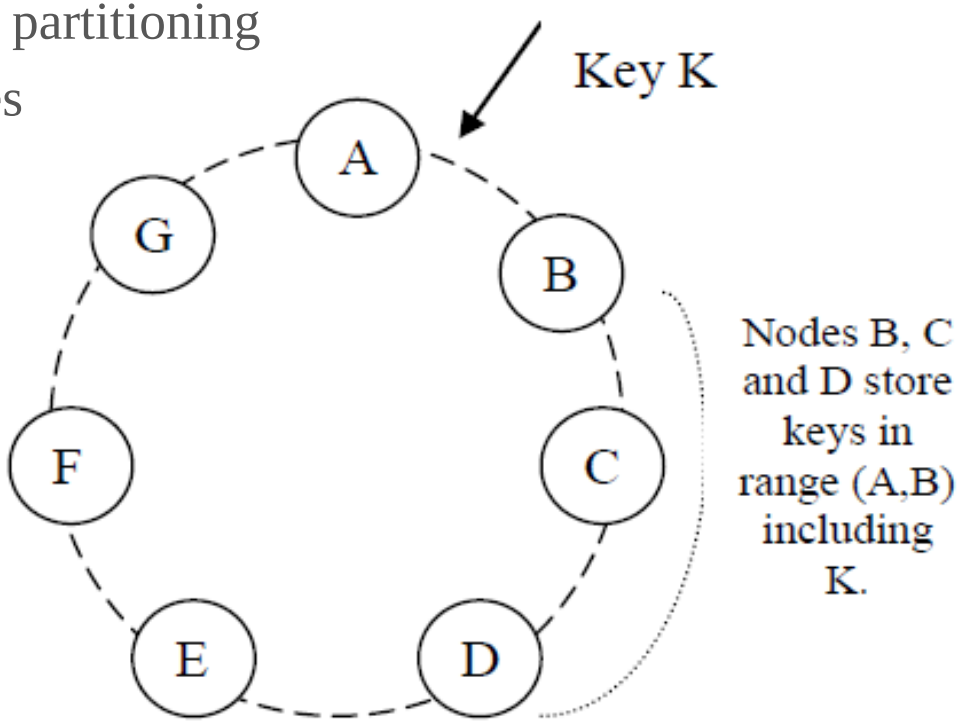
Dynamo architecture: Interface

- **get**(key)
  - Locate object replicas associated with the key
  - Return object/list of objects, with context
- **put**(key, context, object)
- Context encodes system metadata
  - E.g., version
- MD5[Caller key] → 128 bit identifier

© 2012 A. Datta & F. Oggier, NTU Singapore

Dynamo architecture: Data placement

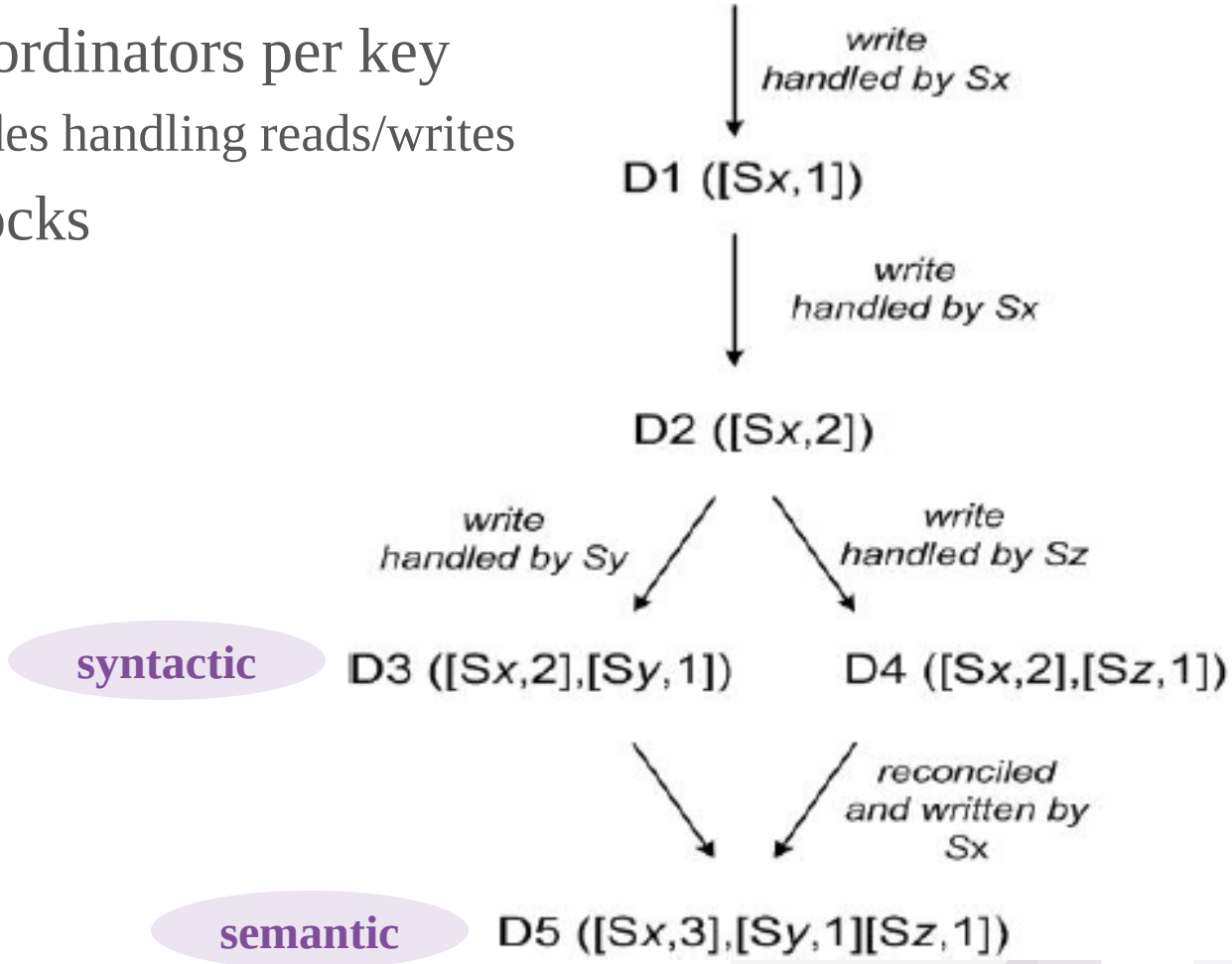
- Zero-hop DHT
  - Consistent hashing based data partitioning
  - All nodes know all other nodes
- Multiple "tokens" per node
  - Virtual node instances
  - Easy to handle heterogeneity
  - Node departure/arrival
    - Load is distributed
- Replication (configurable)
  - For any key: ***preference list***
    - Ensure distinct physical nodes
    - Across multiple data centers



© 2012 A. Datta & F. Oggier, NTU Singapore

Dynamo architecture: Data versioning

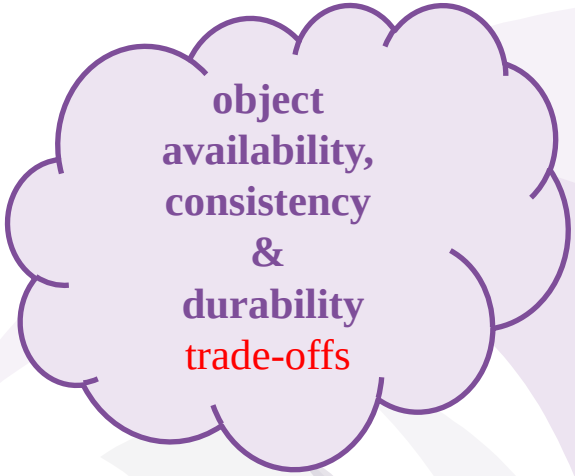
- Many potential coordinators per key
  - Coordinators: Nodes handling reads/writes
- Version: vector clocks
- Reconciliation
  - Syntactic
  - Semantic



© 2012 A. Datta & F. Oggier, NTU Singapore

Dynamo architecture: Executing get()/put()

- Symmetry
  - Client can send get/put requests for any key to any Dynamo node
- Sloppy-quorum
  - First N healthy nodes in the preference list
  - R+W > N quorum
- Upon receiving put() request
  - coordinator generates vector clock, writes locally
  - sends to N highest-ranked reachable nodes
  - W-1 acks implies a successful write
- Upon receiving get() request
  - coordinator requests for all existing versions to N highest-ranked nodes
  - waits for R responses, gathers all versions, and sends all causally unrelated versions



© 2012 A. Datta & F. Oggier, NTU Singapore

Dynamo architecture: Other aspects

- Hinted handoff: Always writeable
- Gossip based
  - failure detection
  - membership information propagation
- Buffered writes:
  - Writes stored in main memory buffer, periodically written to storage
  - Improves latency, risks durability

© 2012 A. Datta & F. Oggier, NTU Singapore

Dynamo in December 2006

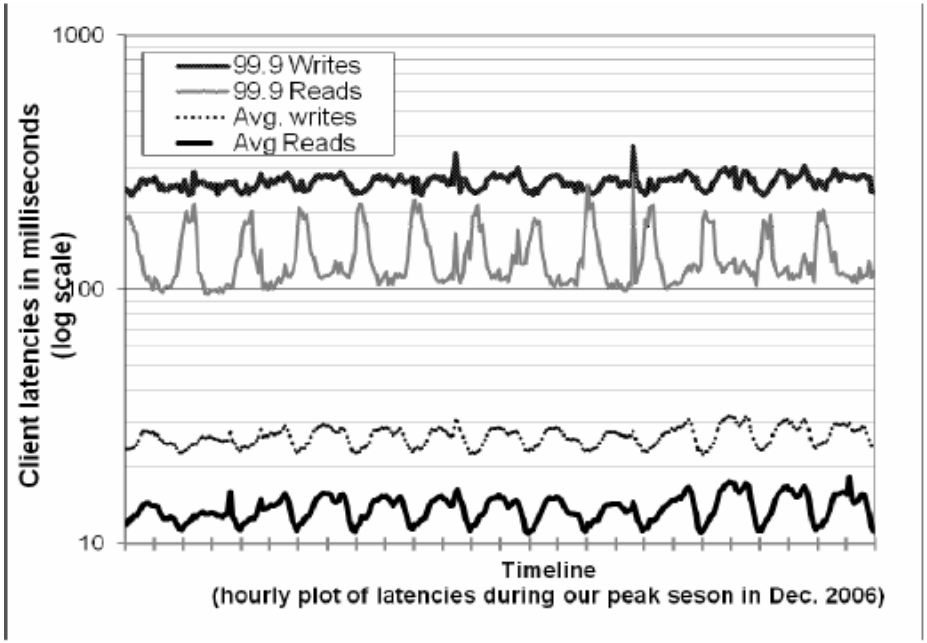


Figure 4: Average and 99.9 percentiles of latencies for read and write requests during our peak request season of December 2006. The intervals between consecutive ticks in the x-axis correspond to 12 hours. Latencies follow a diurnal pattern similar to the request rate and 99.9 percentile latencies are an order of magnitude higher than averages

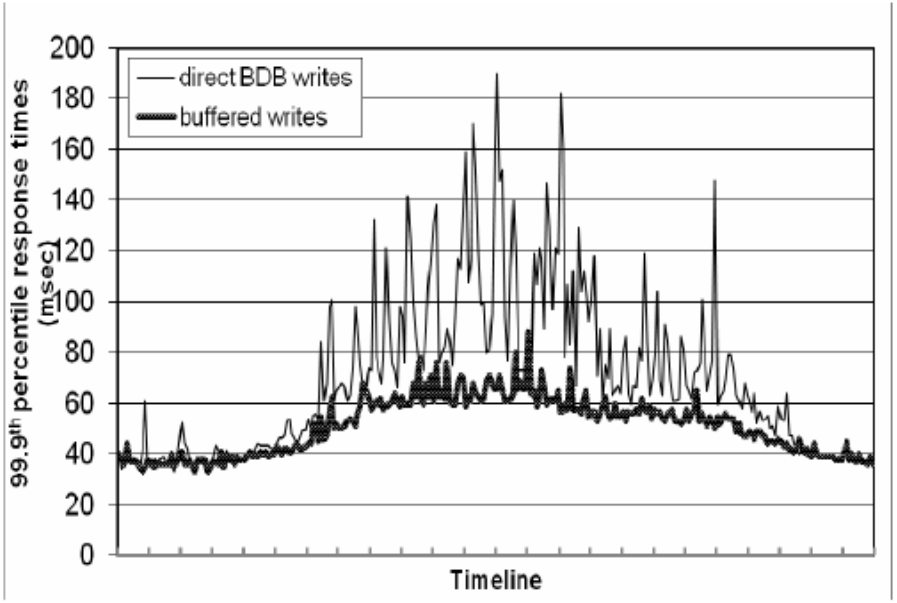


Figure 5: Comparison of performance of 99.9th percentile latencies for buffered vs. non-buffered writes over a period of 24 hours. The intervals between consecutive ticks in the x-axis correspond to one hour.

© 2012 A. Datta & F. Oggier, NTU Singapore

Dynamo: Summary

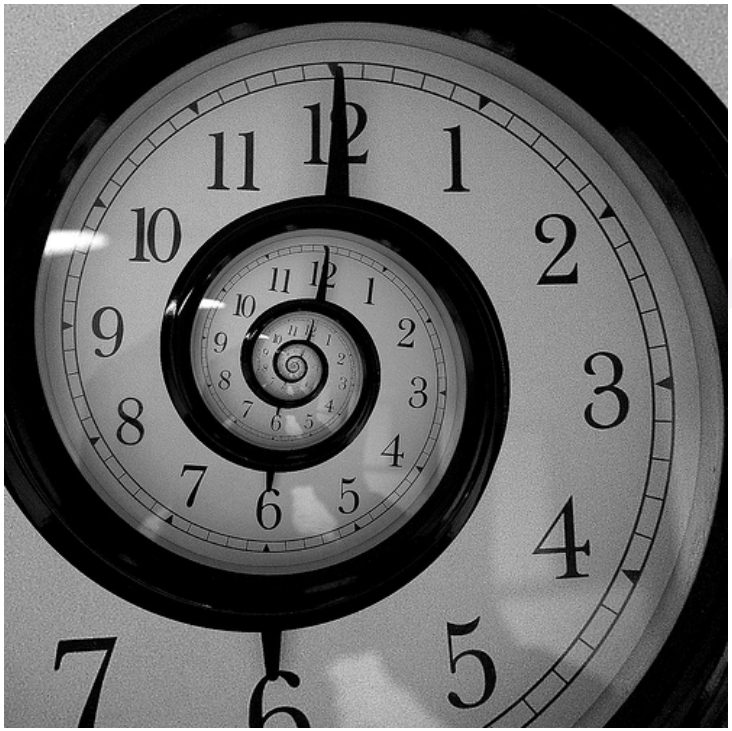
Table 1: Summary of techniques used in *Dynamo* and their advantages.

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.

Distributed File System

- Dynamo provides a bare-bone storage service
  - using the key-value data structure
  - how about something more sophisticated, i.e., a file system?

Dynamo@ SOSP'07



GFS@ SOSP'03

© 2012 A. Datta & F. Oggier, NTU Singapore

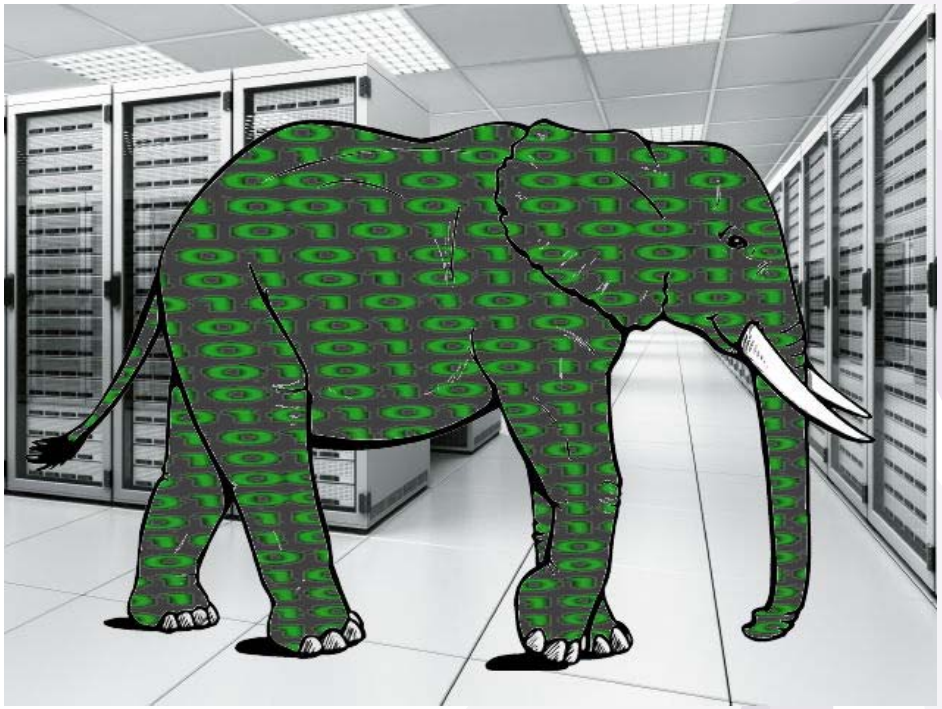
GFS

The Google File System
Sanjay Ghemawat, Howard Gobioff and Shun-Tak Leung
SOSP 2003

© 2012 A. Datta & F. Oggier, NTU Singapore

The big fat Google computations

- Throughput for bulk data processing in batches
  - High sustained bandwidth is more important
    - Than low latency
- Few million files
  - Mostly > 100 MB
    - Multi-GB files very common
  - Small files must be supported
    - But no need for optimization



© 2012 A. Datta & F. Oggier, NTU Singapore

Workload is primarily two kinds of reads

- Large streaming reads
  - 100s of KBs, > 1MB …
- Small random reads

- Successive operations from a client *read contiguous region* of a file
- Applications can sort small reads to advance steadily through a file
  - To avoid going back & forth

© 2012 A. Datta & F. Oggier, NTU Singapore

Many large sequential writes/appends …

- Mainly when a file is being created
  - Once written, it is seldom modified
- Small writes at arbitrary position needs to be supported
  - But do not have to be efficient!

- Multiple clients that concurrently append
  - Producer-consumer queues, many-way merging
    - e.g., MapReduce operations
  - Needs atomicity

© 2012 A. Datta & F. Oggier, NTU Singapore

KISS: Keep it simple, Stupid!

- Basic operations
  - E.g., Not standard (POSIX) compliant
    - create, delete, open, close, read, write files
    - snapshot, record append

© 2012 A. Datta & F. Oggier, NTU Singapore

GFS architecture

- Single Master
  - Simplifies design
    - Can carry out sophisticated data placement and replication decisions using global knowledge
  - What about fault-tolerance, bottleneck, …?
- Multiple chunk servers & multiple clients
  - Could be run on same machines

- Note: HDFS (Hadoop™ Distributed File System) follows a similar architecture

© 2012 A. Datta & F. Oggier, NTU Singapore

Files → Fixed-sized chunks

- Global 64 bit uid for each chunk
  - Assigned by master at chunk creation
- Chunkservers store chunks on local disks
  - As Linux files
  - Chunks replicated for reliability
    - Default: 3 replicas
      - Collosus (GFS v 2) uses erasure codes: 1.5x
- Clients interact with master for metadata
  - Interacts with chunkservers for actual data manipulation
    - Caching: Client "cache" metadata
    - Chunkservers have automatic Linux caching
    - No other caching at either clients/chunkservers:
      - Neither meaningful nor feasible … (for the involved workloads)

Maximize:
- Data reliability
  - e.g., not all replicas on the same rack
- Network utilization
  - aggregate BW of racks
    - * during reads
    - * but multi-rack write traffic

© 2012 A. Datta & F. Oggier, NTU Singapore

The Single Master

- Maintains all system metadata (held in *main memory*)
  - Namespace
  - Mapping from files to chunks
  - Current locations of chunks

Advantages:
- Performance
- Easy and efficient scan for
  - garbage collection
  - re-replication
  - migration to rebalance

Periodic ***HeartBeat*** with chunkservers to keep track of status, chunk locations, …

Stored persistently ***logging mutations in an operations log*** **storing in** ***local disk + master replicas***

If this information is lost, then even if the chunks "survive", the file system is useless.

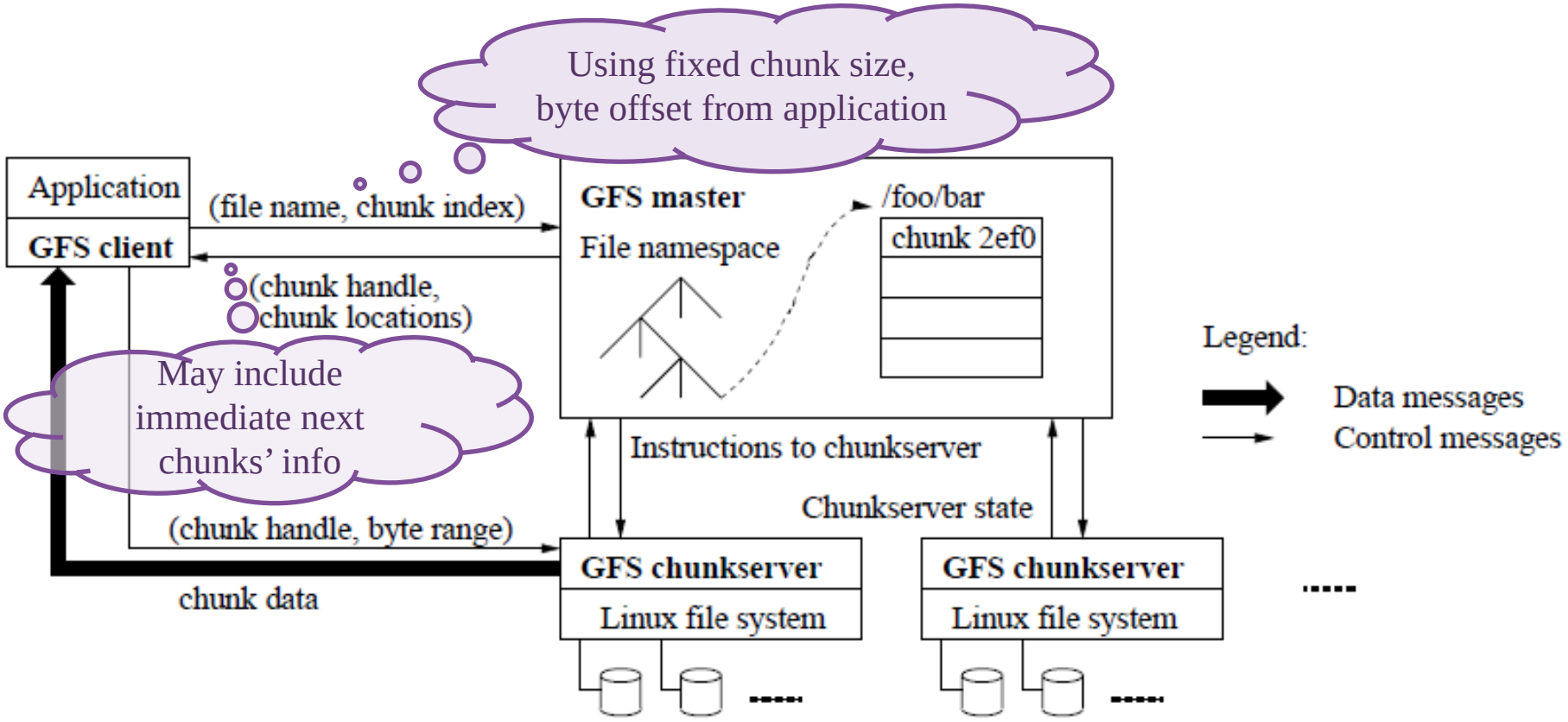
© 2012 A. Datta & F. Oggier, NTU Singapore

Chunk size and system scalability

- 64MB
  - + Reduces clients' need to interact with master
    - Many operations are contiguous/sequential on a file
    - involves same chunkserver
    - Client can cache chunk locations for even a multi-TB working set
  - + Client likely to carry out more operations on same chunk
    - Since chunks are large
    - Amortizes network connection costs (persistent TCP connection)
  - + Reduces the size of metadata stored on the master
    - So that it ***fits in memory*** (<64 bytes metadata per chunk at Master)
      - Significant performance boost!
  - – Possibility of hot spot
  - – Fragmentation/poor space utilization
    - Most files are large, so only last chunk partially filled
    - Further mitigated using lazy space allocation

© 2012 A. Datta & F. Oggier, NTU Singapore

Master, client, chunkserver interactions



Design choices: *Reduce the master's involvement* as much as possible

- Cache chunkserver meta-info at clients
- Obtain/send extra meta-info in advance (amortizing cost of communication)
- "Heavy" tasks directly between clients/chunkservers

© 2012 A. Datta & F. Oggier, NTU Singapore

Namespace & locks

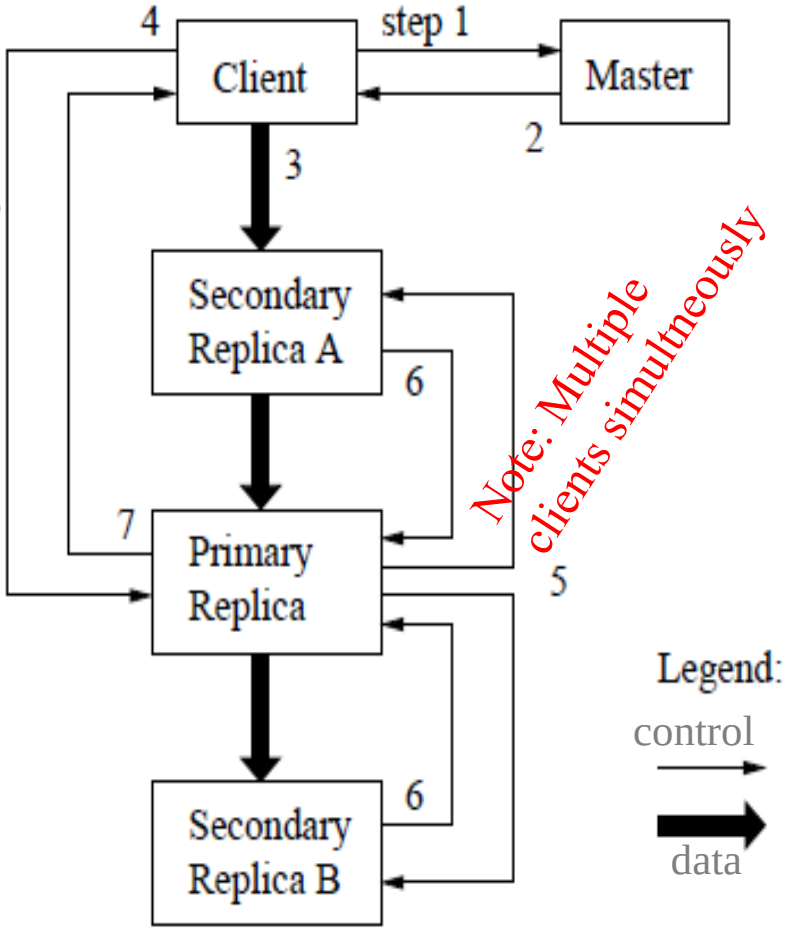
- GFS namespace
  - No per directory data structures
  - Lookup table: Mapping full pathnames to metadata
    - Use prefix compression for storage in memory
- Each node in namespace tree has associated read/write lock
  - To manipulate /d1/d2/…/dn/leaf
    - Obtain read locks for /d1, /d1/d2, … /d1/d2/…/dn
    - Obtain read or write lock for /d1/d2/…/dn/leaf

- Notes:
  - File creation does not require a write lock on the parent directory because there is no “directory”, or inode-like, data structure to be protected from modification, thus allowing multiple simultaneous mutations in a directory
  - In that order (consistent total order) to avoid deadlocks
  - Allocated lazily, removed once not in use

© 2012 A. Datta & F. Oggier, NTU Singapore

Leases for mutations

- One chunk replica is given a lease to act as the "primary" replica
  - Typically for 60 seconds
    - Renewable repeatedly if mutation continues
      - over HeartBeat messages
    - Master can revoke lease (e.g., for snapshot)
  - Primary determines a serial order for all mutations in a chunk
    - Other replicas follow this order
- Global mutation order is defined by
  - The lease grant order
  - Serial order inside a lease



Data is pipelined linearly:
- disentangled from control messages
- optimized for network topology
- leveraging on full-duplex links
- providing1MB Tx in ~80ms+

© 2012 A. Datta & F. Oggier, I

Several other things ...

- Snapshot
  - Using copy on write
- Atomic record appends
  - Special/different from writes, since Google had append heavy workload
- Consistency model
  - Complicated semantics
    - both for defining consistency as well as replicas
- Stale replica detection, data integrity (using check sums)
- Garbage collection
- replica creation, re-creation, rebalancing

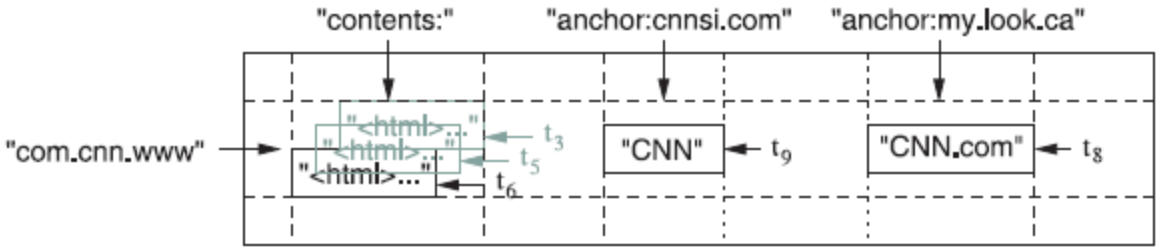
© 2012 A. Datta & F. Oggier, NTU Singapore

In the meanwhile … Google grew

- Mix of applications …
  - Gmail, docs, buzz, …
  - Google earth
  - Youtube …
- Latency sensitivity
- Diverse “sizes” of data
- Both read/write intensive

BigTable: Support for structured data

- Data organized into three dimensions
  - rows, columns, timestamps
    - Not a full fledged RDMBS
- It is an “application” on top of GFS
  - but, is part of the “infrastrcuture”
    - For other applications
  - relies on Paxos algo. based Chubby
    - A highly available and persistent lock service



© 2012 A. Datta & F. Oggier, NTU Singapore

Google's next generation FS: Collossus

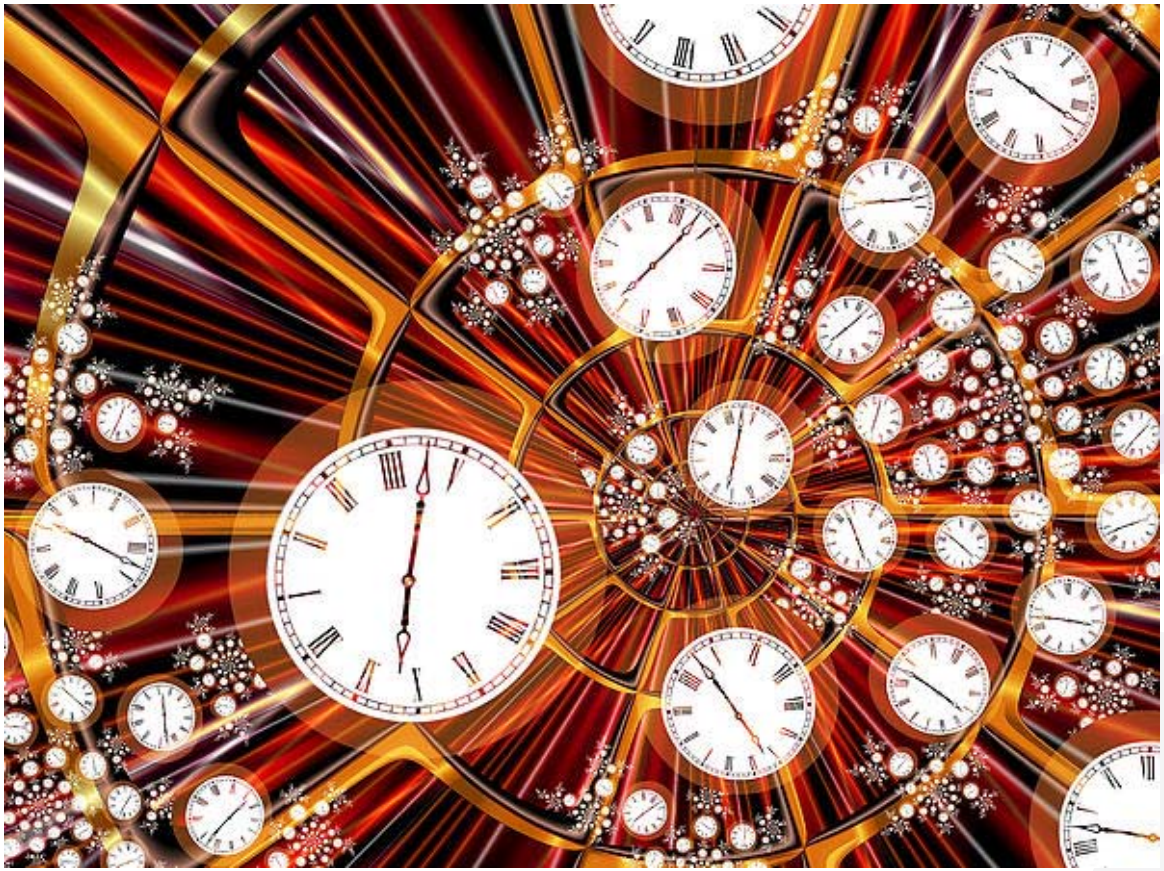
Details are "sketchy"

- Infrastructure for diverse mix of workload …
- Cluster-level file system
- Data typically written using Reed-Solomon (1.5x) erasure codes (ECs)*
- Support for smaller "chunks"
  - More flexibility for diverse application
- Distribution of Master functionality
- Why ECs? Can we do better than the state of the art?
  - Second half!

* With Microsoft's Azure also using ECs, they are now pretty much a part of mainstream deployment

© 2012 A. Datta & F. Oggier, NTU Singapore

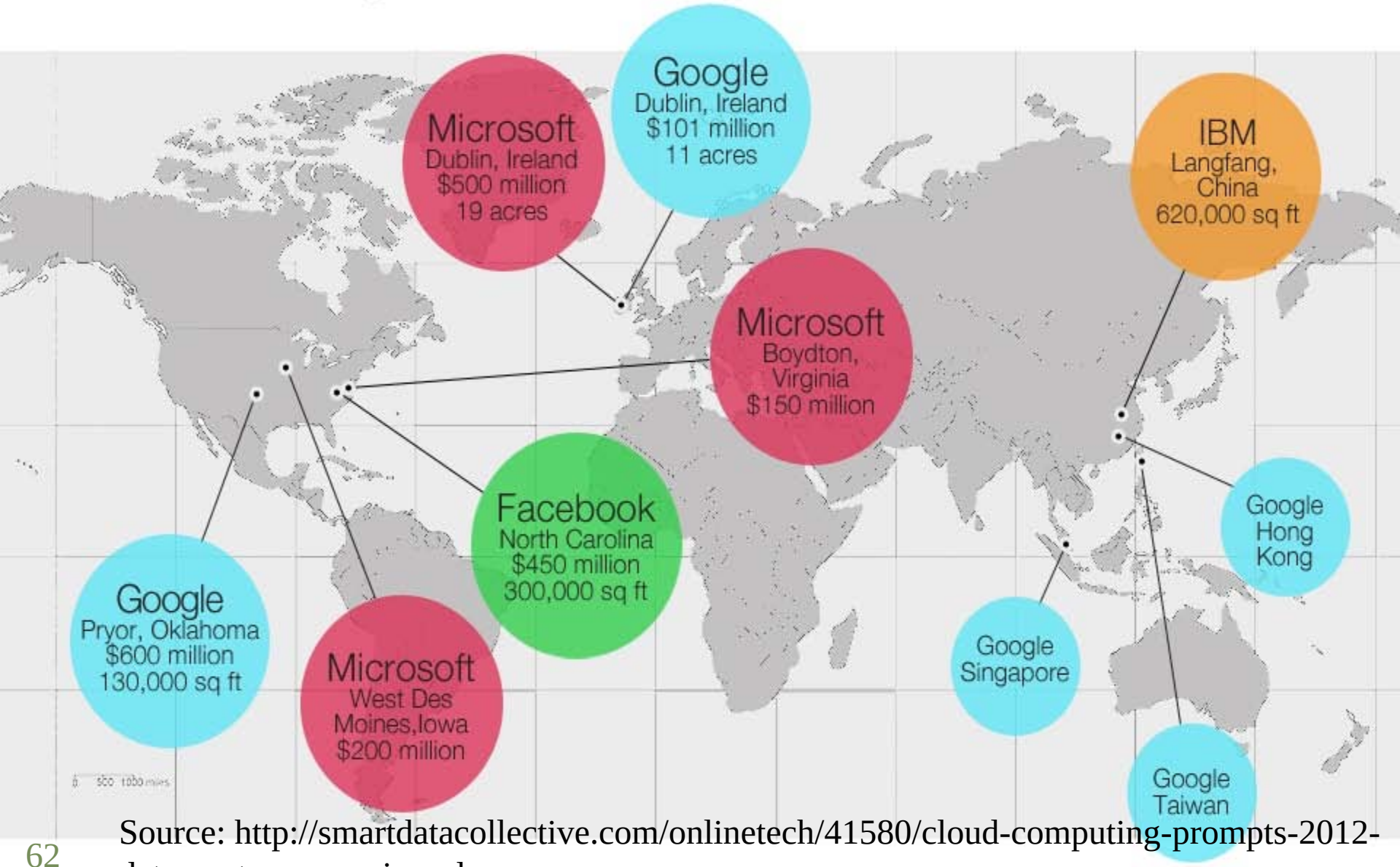
Time for a break!



© 2012 A. Datta & F. Oggier, NTU Singapore

2012 Data Center Expansion Plans

Cloud Computing Prompts Worldwide Expansion



Source: http://smartdatacollective.com/onlinetech/41580/cloud-computing-prompts-2012-data-center-expansion-plans

Failure Is Inevitable

- But, failure of the system is not an option!
  - Failure is the pillar of rivals' success …
- Solution: Redundancy & Distribution

© 2012 A. Datta & F. Oggier, NTU Singapore

Is the Danger Real? *Yes*

Online Backup Company Carbonite Loses Customers' Data, Blames And Sues Suppliers (Updated)

Robin Wauters
Mar 23, 2009

TechCrunch

Hotmail Data Loss Reveals Cloud Trust Issues

PCWorld

By Keir Thomas, PCWorld

Hot on the heels of what was possibly the first major cloud data leak a few weeks ago, as the new year got underway Microsoft followed up by appearing to wipe the e-mails of a significant number of Hotmail users.

Gmail outage passes 24 hours for some (updated)

CNNMoney
A Service of CNN, Fortune & Money

By Seth Weintraub February 28, 2011: 1:01 PM ET

Amazon's Cloud Crash Disaster Permanently Destroyed Many Customers' Data

Henry Blodget | Apr. 28, 2011, 7:10 AM

BUSINESS INSIDER

© 2012 A. Datta & F. Oggier, NTU Singapore

Data Center Fault-Tolerance

- Faults are *omnipresent*
  - *Hardware, network, software, human, misconfiguration, …*
- *Cascade of failures* in interdependent networks
  - Power failure => Network switches stop working
  - Network failure => Control system for power system ineffective

© 2012 A. Datta & F. Oggier, NTU Singapore

Some Numbers

- Data from Los Alamos National Laboratory (DSN 2006), gathered over 9 years, 4750 machines, 24101 CPUs.
- Distribution of failures:
  - Hardware 60%
  - Software 20%
  - Network/Environment/Humans 5%
- Failures occurred between once a day to once a month.

© 2012 A. Datta & F. Oggier, NTU Singapore

Redundancy Based Fault Tolerance

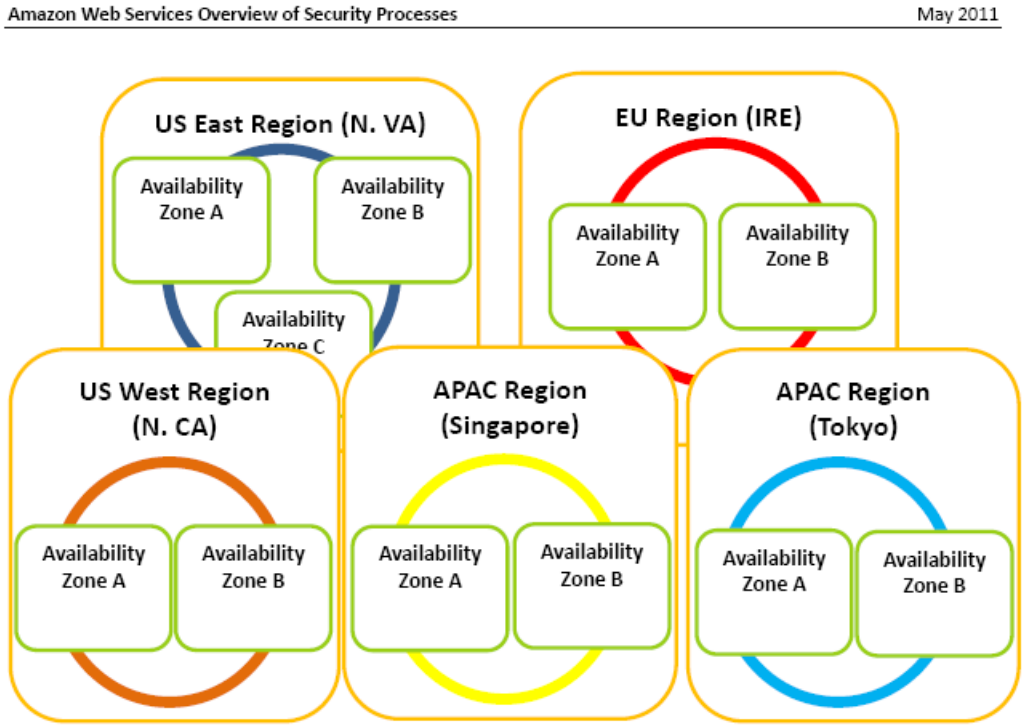
- Replicate data
  - e.g., 3 or more copies
  - In nodes on different racks
    - Can deal with switch failures

- Power back-up using battery between racks (Google)

© 2012 A. Datta & F. Oggier, NTU Singapore

Redundancy Based Fault Tolerance

- Using "independent" physical infrastructure
  - Over different availability zones (Amazon AZ)
    - How independent are components in a complex network?
  - Over multiple geographical regions



Note: The recent (April 2011) AWS outage was the first *region-wide failure*

© 2012 A. Datta & F. Oggier, NTU Singapore

Five Levels of Redundancy

- Physical
- Virtual resource
- Availability zone
- Region
- Cloud

From: http://broadcast.**oreilly.com**/2011/04/the-aws-outage-the-clouds-shining-moment.html

© 2012 A. Datta & F. Oggier, NTU Singapore

At What Cost?

- Failure is not an option, but …
  - … are the overheads acceptable?

Reducing the Overheads of Redundancy

- Erasure codes
  - Much lower storage overhead
  - High level of fault-tolerance
    - In contrast to replication or RAID based systems
- Has the potential to significantly improve the "bottomline"
  - Can it however match the performance needs?
    - An open question*

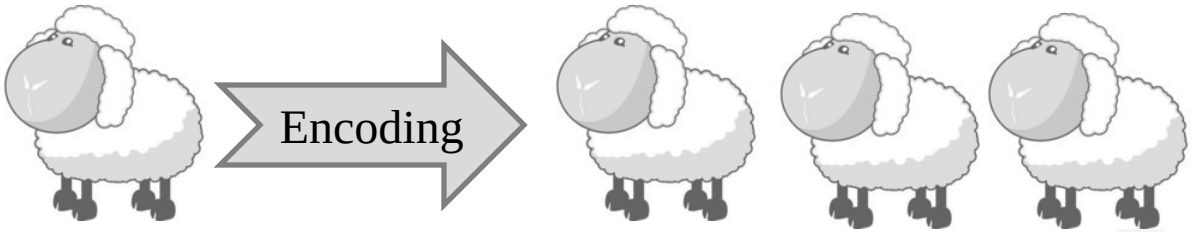
Does erasure coding have a role to play in my data center?
Zhe Zhang, Amey Deshpande, Xiaosong Ma, Eno Thereska, Dushyanth Narayanan
MSR TR 2010

* Note: Both Google's new DFS Collossus, as well as Microsoft's Azure now use ECs

© 2012 A. Datta & F. Oggier, NTU Singapore

Erasure Codes (ECs)

- An (n,k) erasure code = a map that takes as input k blocks and outputs n blocks, thus introducing n-k blocks of redundancy.
- 3 way replication is a (3,1) erasure code!



k=1 block *n=3 encoded blocks*

- An erasure code such that the k original blocks can be recreated out of any k encoded blocks is called MDS (maximum distance separable).

© 2012 A. Datta & F. Oggier, NTU Singapore

Reed-Solomon Codes

(named after Irving S. Reed and Gustave Solomon)

- Reed-Solomon Codes are well-known erasure codes.
- Encoding of $(o_1,\ldots,o_k)$ is done by polynomial evaluation:

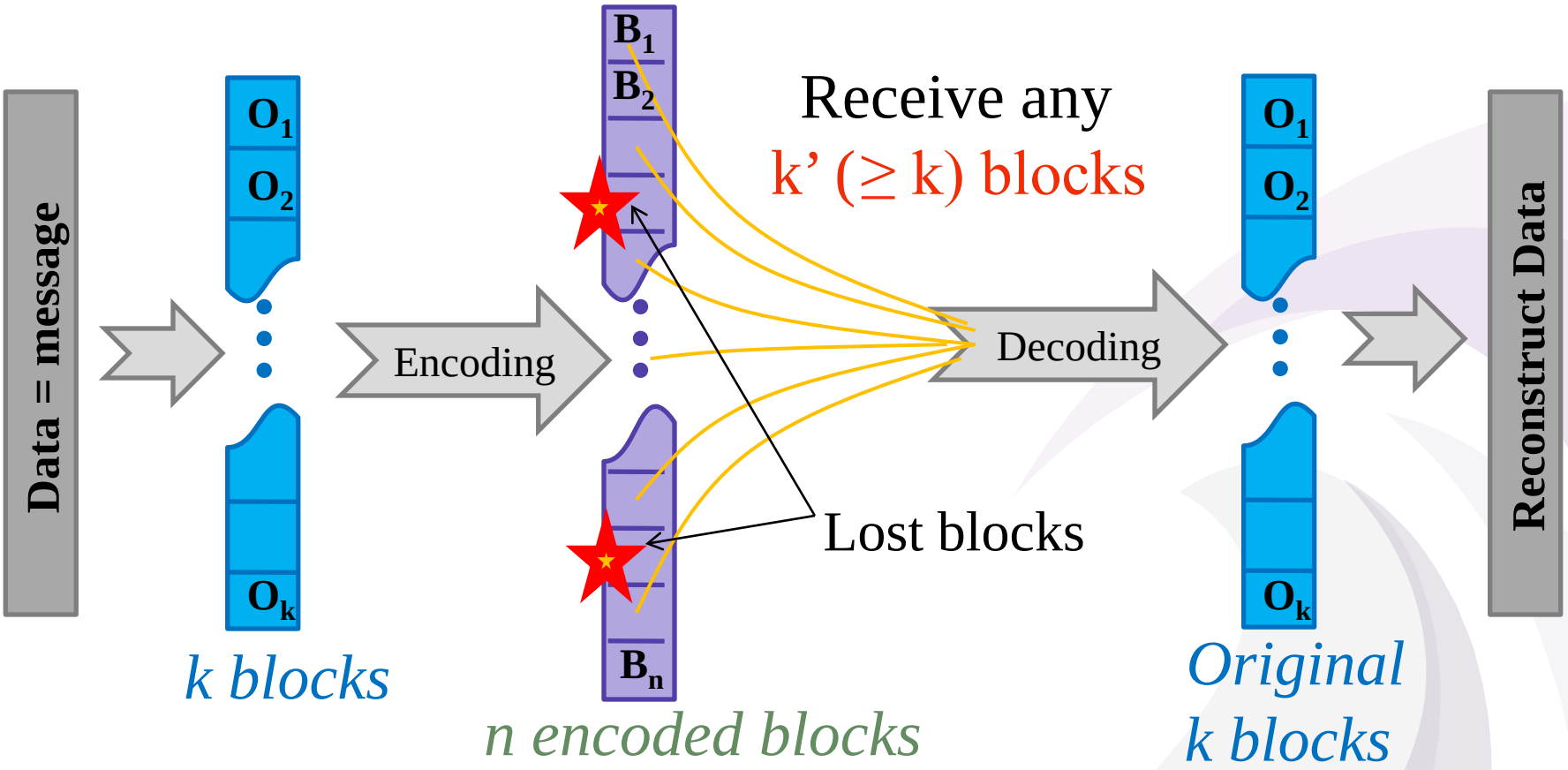
$$p(X) = \sum_{i=0}^{k-1} p_i X^i, \quad p_i = o_{i+1}$$

- The encoding blocks are then $p(\alpha_1),\ldots,p(\alpha_n)$.

© 2012 A. Datta & F. Oggier, NTU Singapore

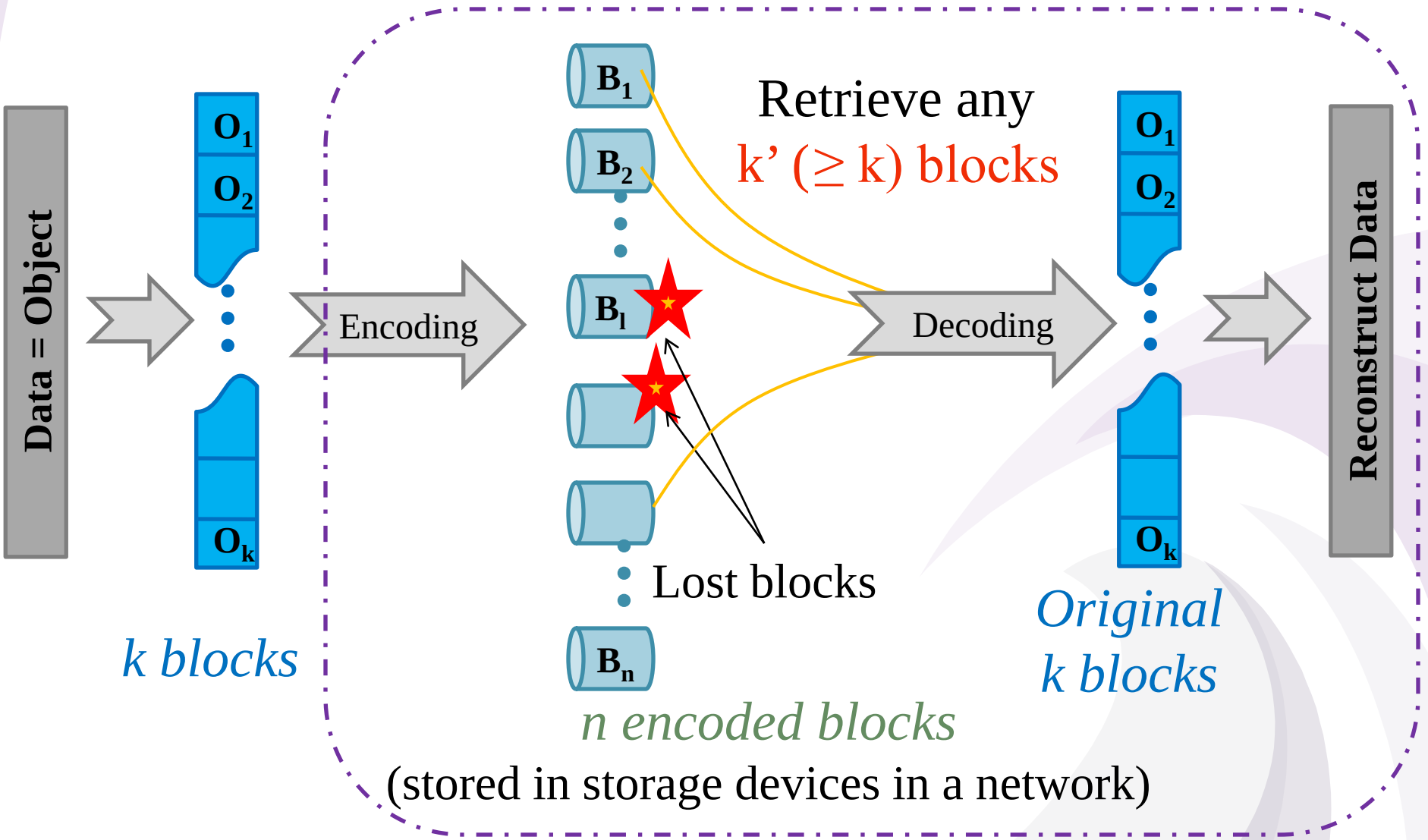
Erasure Codes (ECs)

- Originally designed for communication
  - EC(***n***,***k***)



© 2012 A. Datta & F. Oggier, NTU Singapore

Erasure Codes for Networked Storage



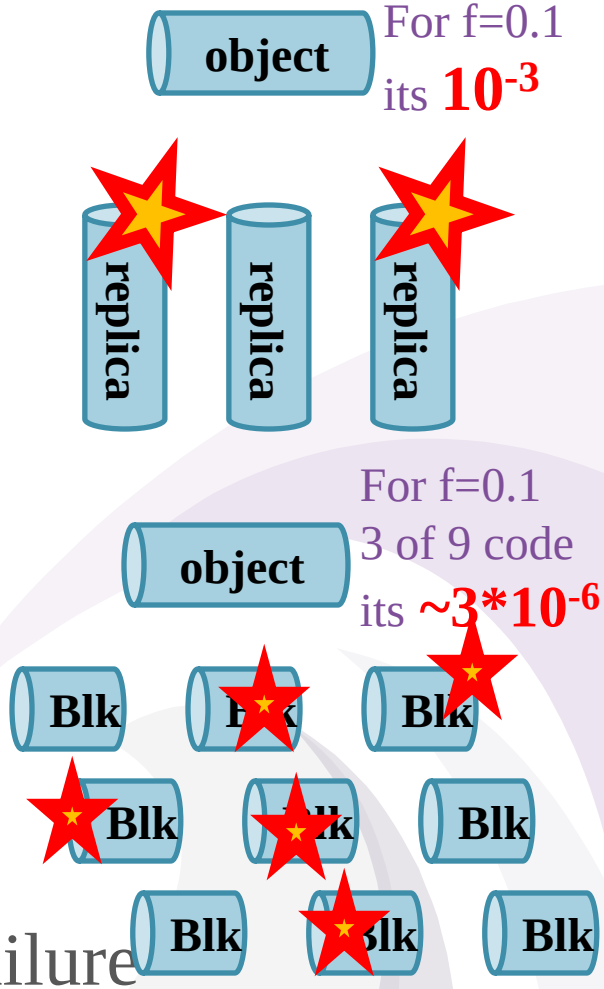
© 2012 A. Datta & F. Oggier, NTU Singapore

Static Resilience

- Replicated *r* times
  - Faults that can be tolerated: *r-1*
  - Probability of failure: f^r
  - Storage efficiency: 1/*r*
  - Access: Find any one good replica
- Erasure coded (*k of n*)
  - Faults that can be tolerated: *n-k*
  - Probability of failure:

$$\sum_{j=1}^{k} \binom{n}{n-k+j} f^{n-k+j} (1-f)^{k-j}$$

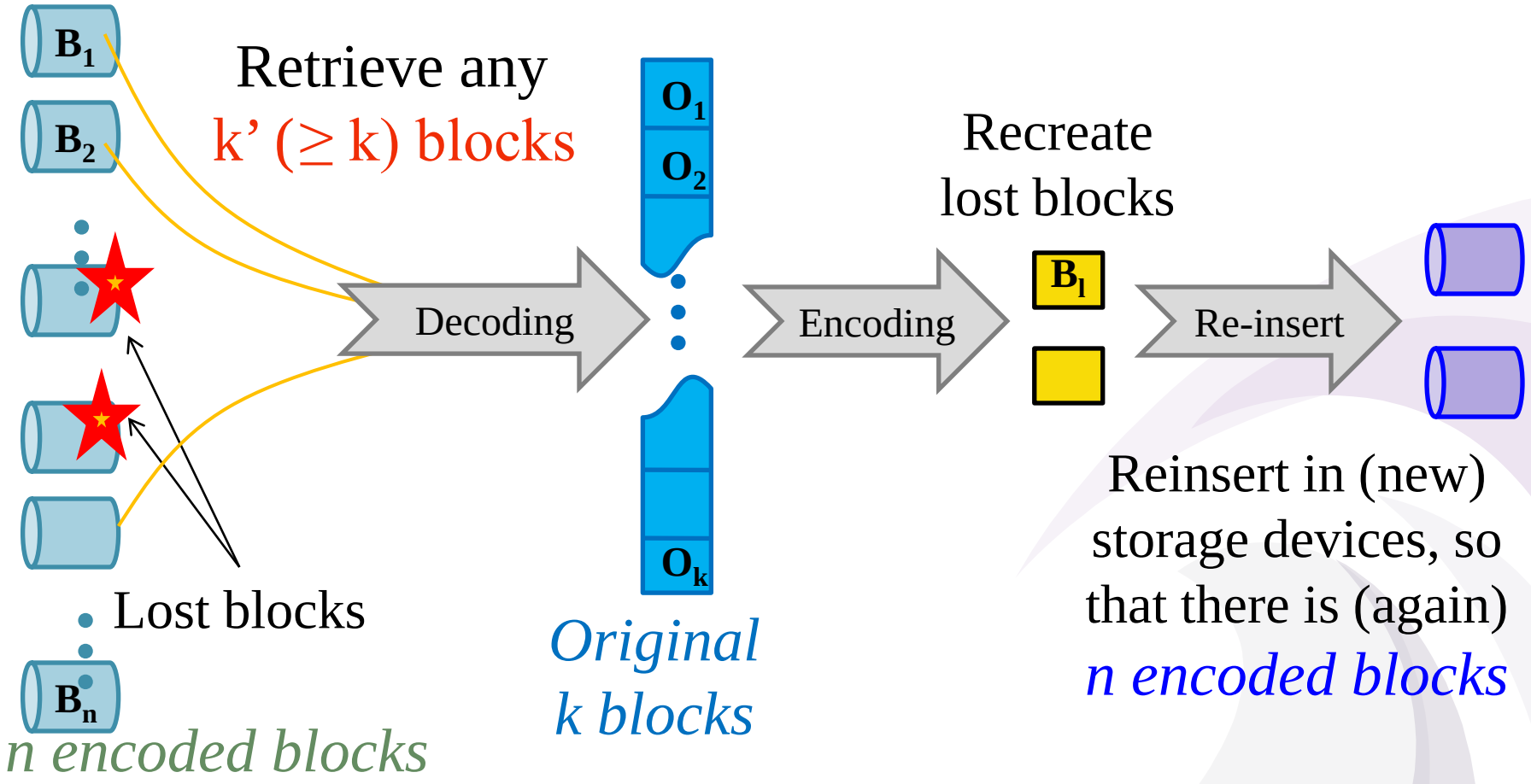
  - Storage efficiency: k/*n*
  - Access: Find *k* good blocks
- Assumption: Peer failure is i.i.d. with failure probability ***f***



© 2012 A. Datta & F. Oggier, NTU Singapore

Replenishing Lost Redundancy for ECs

- Repair needed for long term resilience.



- Repairs are expensive!

© 2012 A. Datta & F. Oggier, NTU Singapore

Can We Do Better?

- Erasure codes tailor-made for distributed networked storage.

Pyramid codes

Regenerating codes

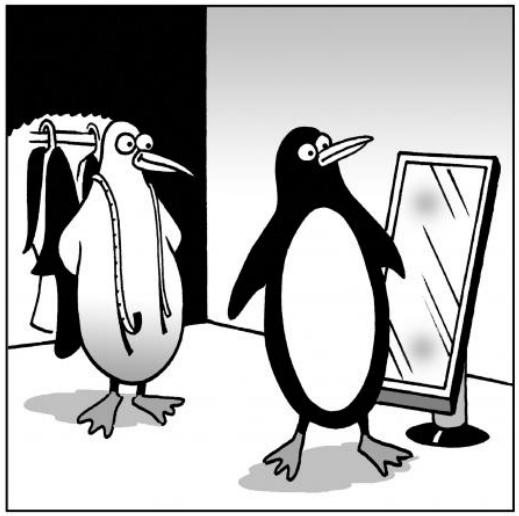
Self-repairing codes

© 2012 A. Datta & F. Oggier, NTU Singapore

What are Tailored-Made Codes?

Desired code properties include:

- Low repair bandwidth
- Low storage overhead
- Good fault tolerance



But also:

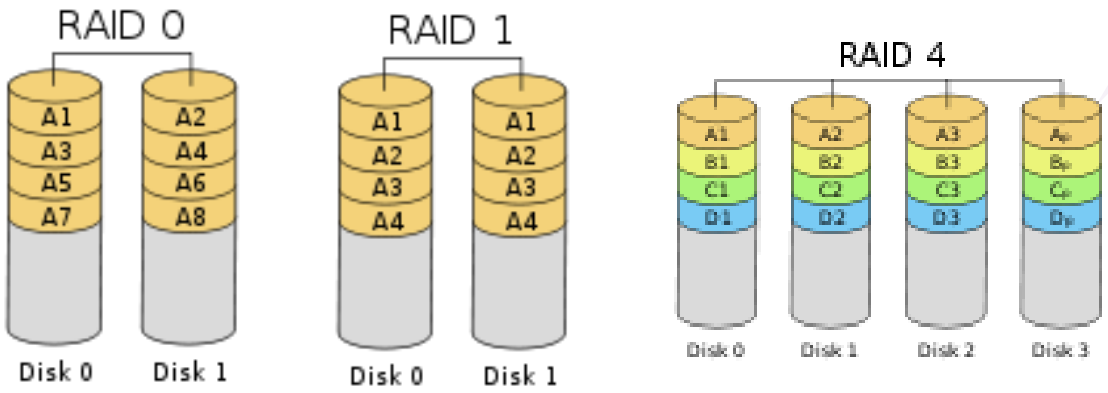
- Repair time
- I/O
- …

© 2012 A. Datta & F. Oggier, NTU Singapore

RAID

(**redundant array of independent disks**; originally **redundant array of inexpensive disks**)

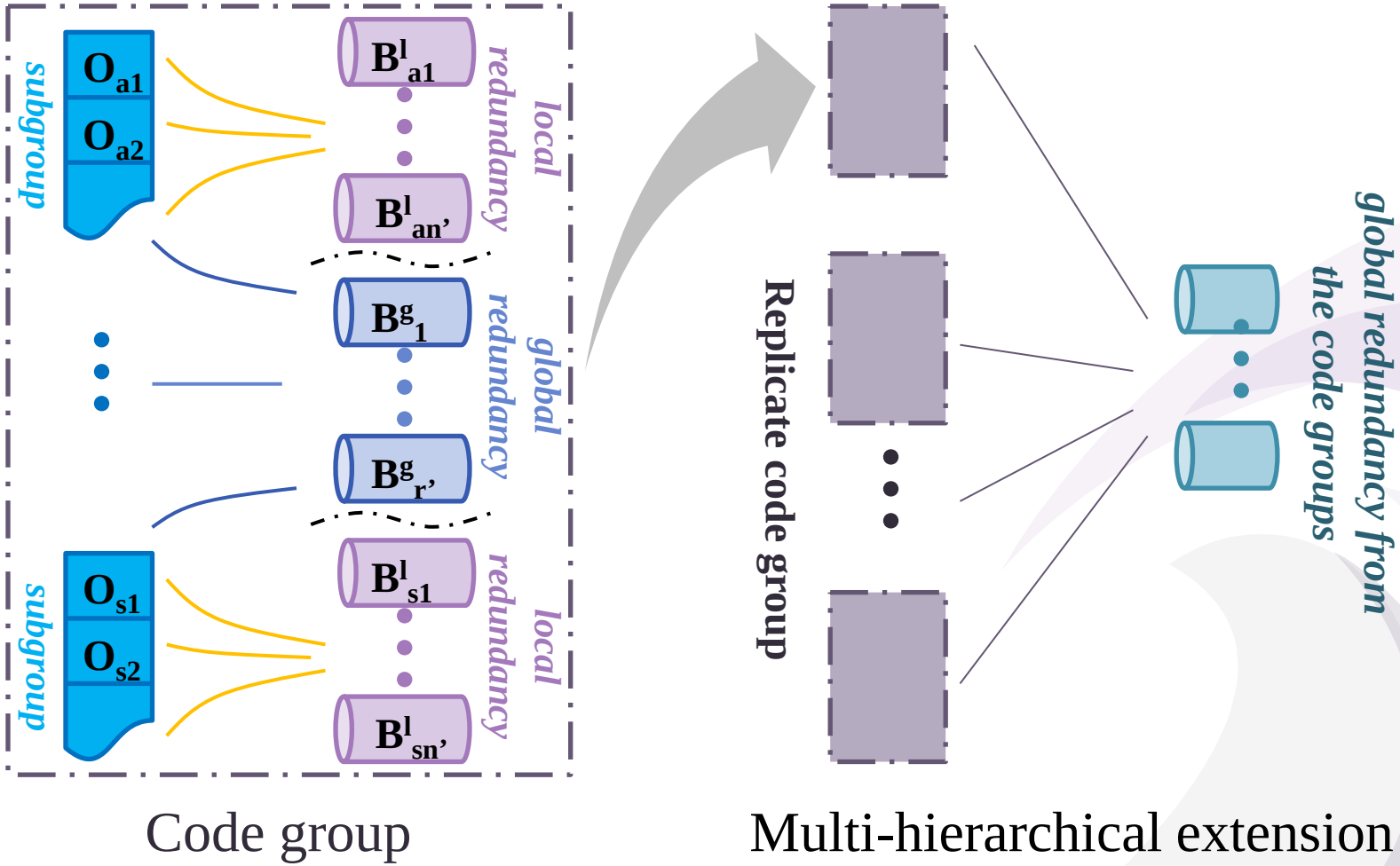
- RAID 0 = store data across multiple drives so that the disk head can read more data in a single move (striping)
- RAID 1 = replication is introduced (mirroring)
- RAID 2,3,4 ,5 = parity bit , sum of all bits across one drive (Hamming parity code)



© 2012 A. Datta & F. Oggier, NTU Singapore

Pyramid & Hierarchical Codes

- Essentially "nested" use of erasure codes.



Code group

Multi-hierarchical extension

© 2012 A. Datta & F. Oggier, NTU Singapore

Pyramid & Hierarchical Codes

- If `small' number of faults
  - Communication restricted within the `hierarchy' suffice
  - Progressively go higher-up for larger number of faults
  - Isolated faults can be repaired independently
- Naturally maps to hierarchical data-center design?

Pros

- Asymmetry
  - Different encoded blocks have different importance
  - Difficult to analyze
  - Complex algorithm (for decoding/repair) and system design

Cons

Pyramid Codes: Flexible Schemes to Trade Space for Access Efficiency in Reliable Data Storage Systems

Cheng Huang, Minghua Chen, and Jin Li

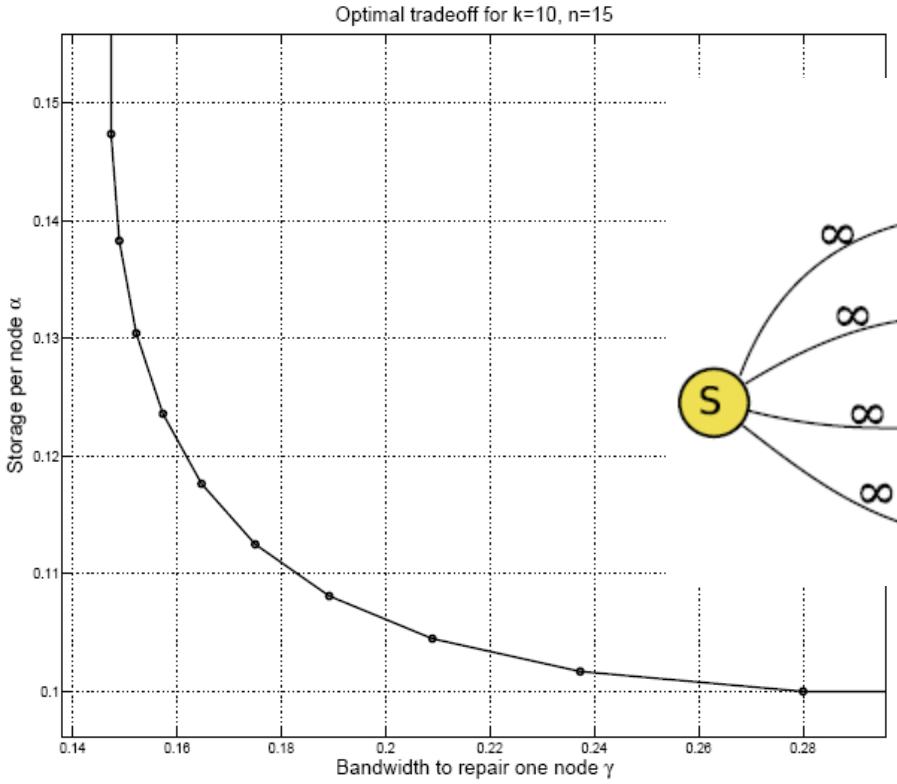
NCA 2007

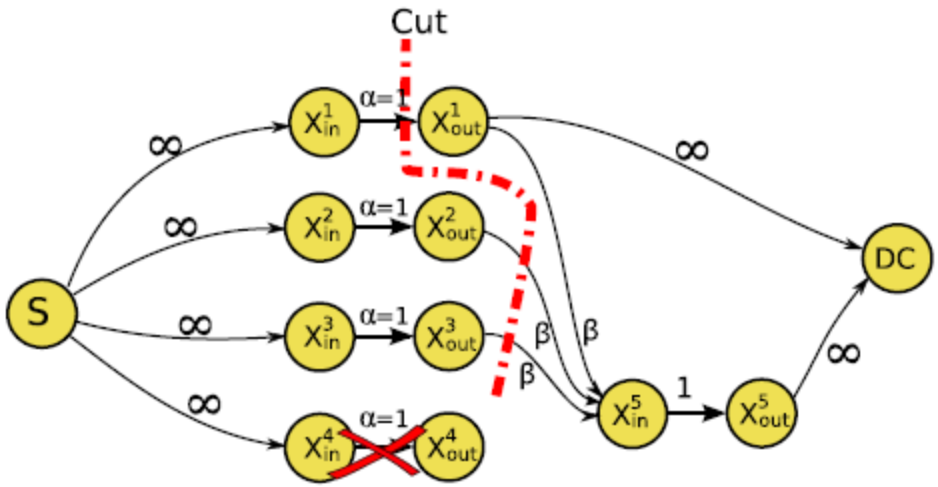
Another (essentially identical, but independent) proposal**: Hierarchical Codes**: How to Make Erasure Codes Attractive for Peer-to-Peer Storage Systems *from Eurecom*

© 2012 A. Datta & F. Oggier, NTU Singapore

Regenerating Codes

- Network information flow based arguments to determine "optimal" trade-off of storage/repair-bandwidth

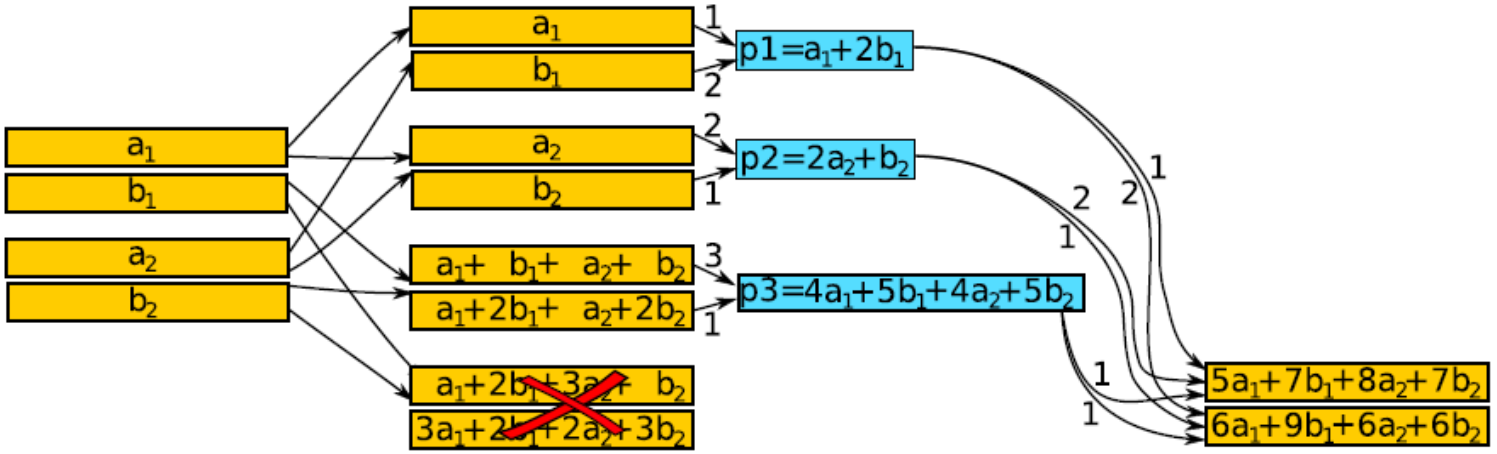




© 2012 A. Datta & F. Oggier, NTU Singapore

Regenerating Codes

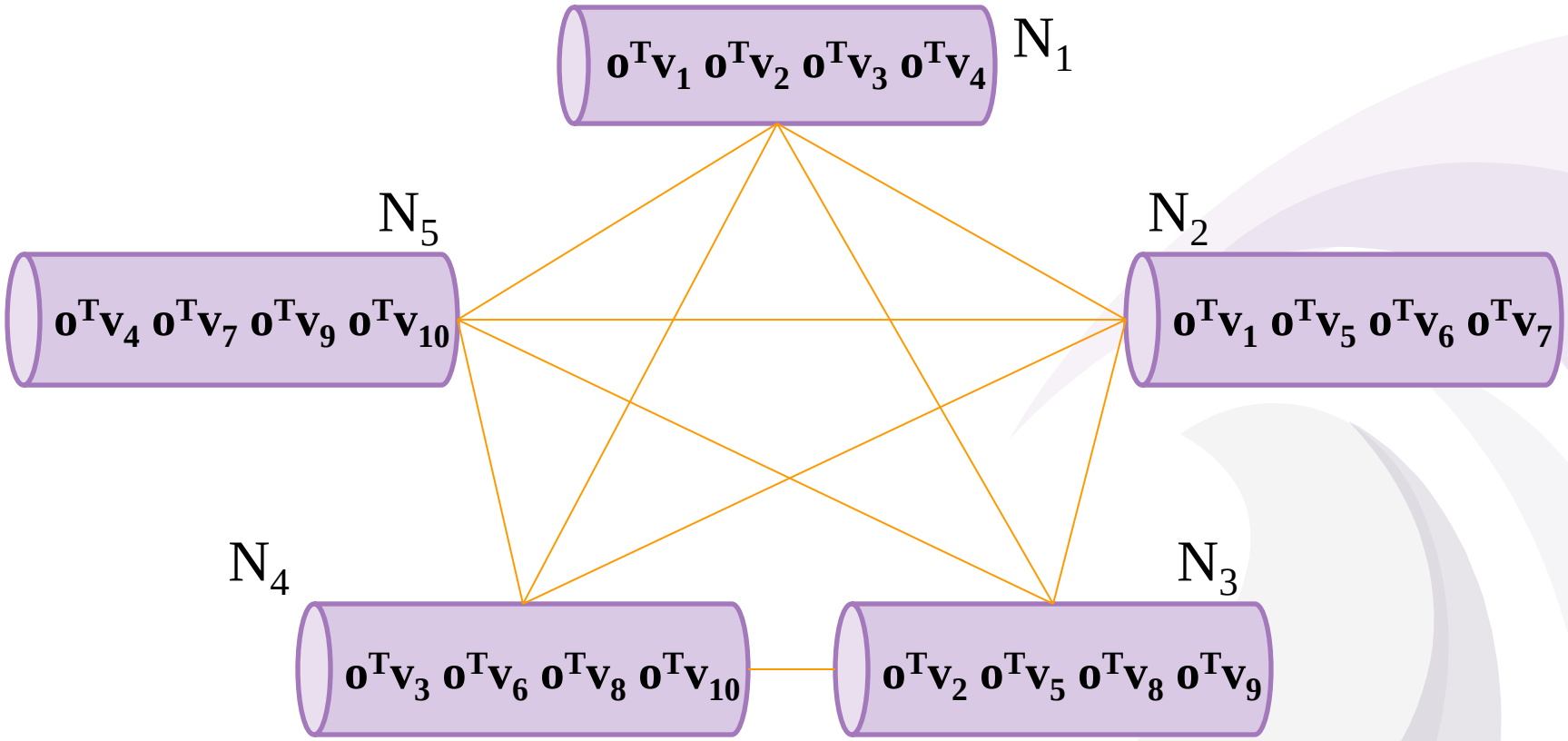
- Example code (w/ Functional Repair)
  - Based on random linear network coding



© 2012 A. Datta & F. Oggier, NTU Singapore

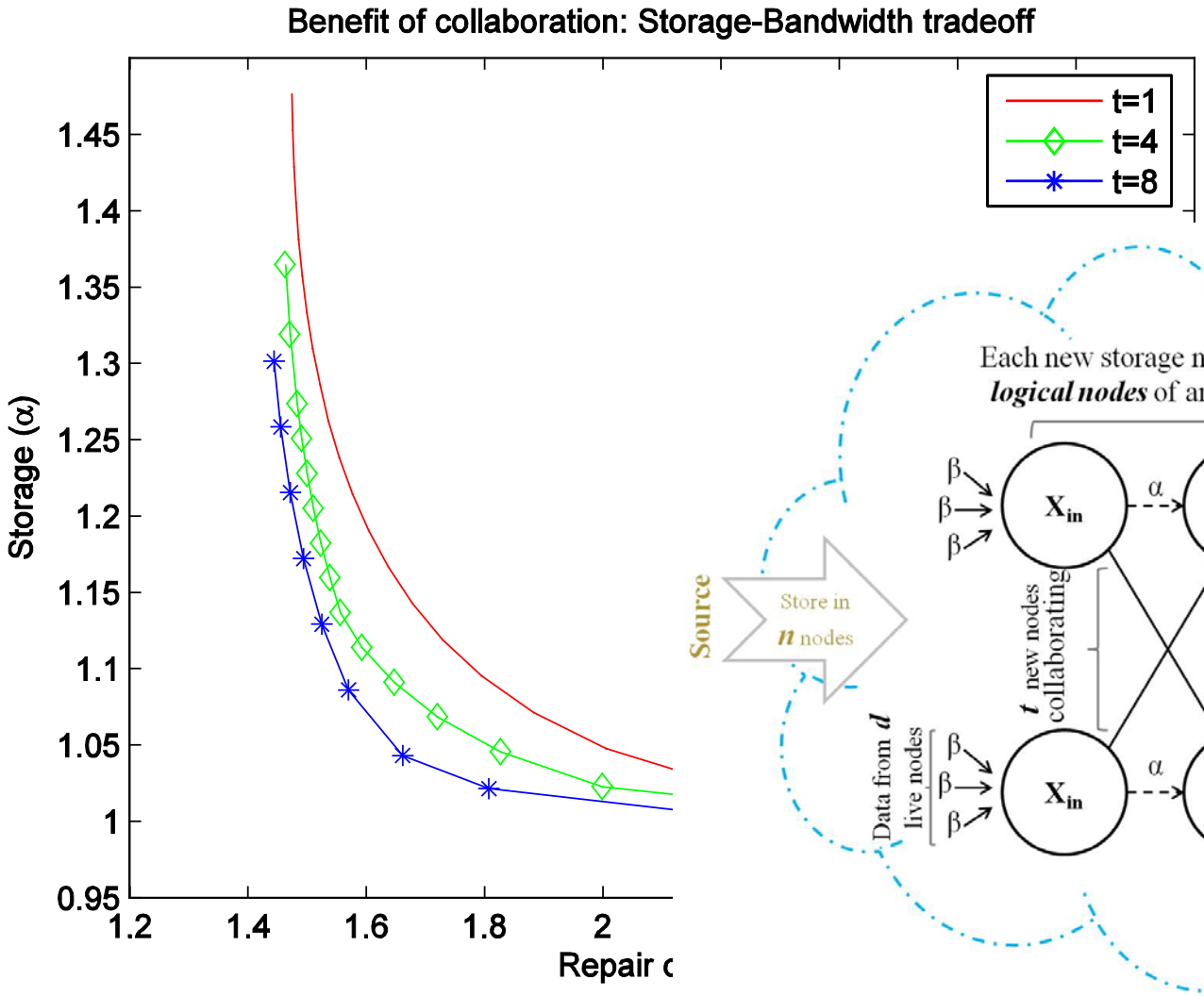
Regenerating Codes

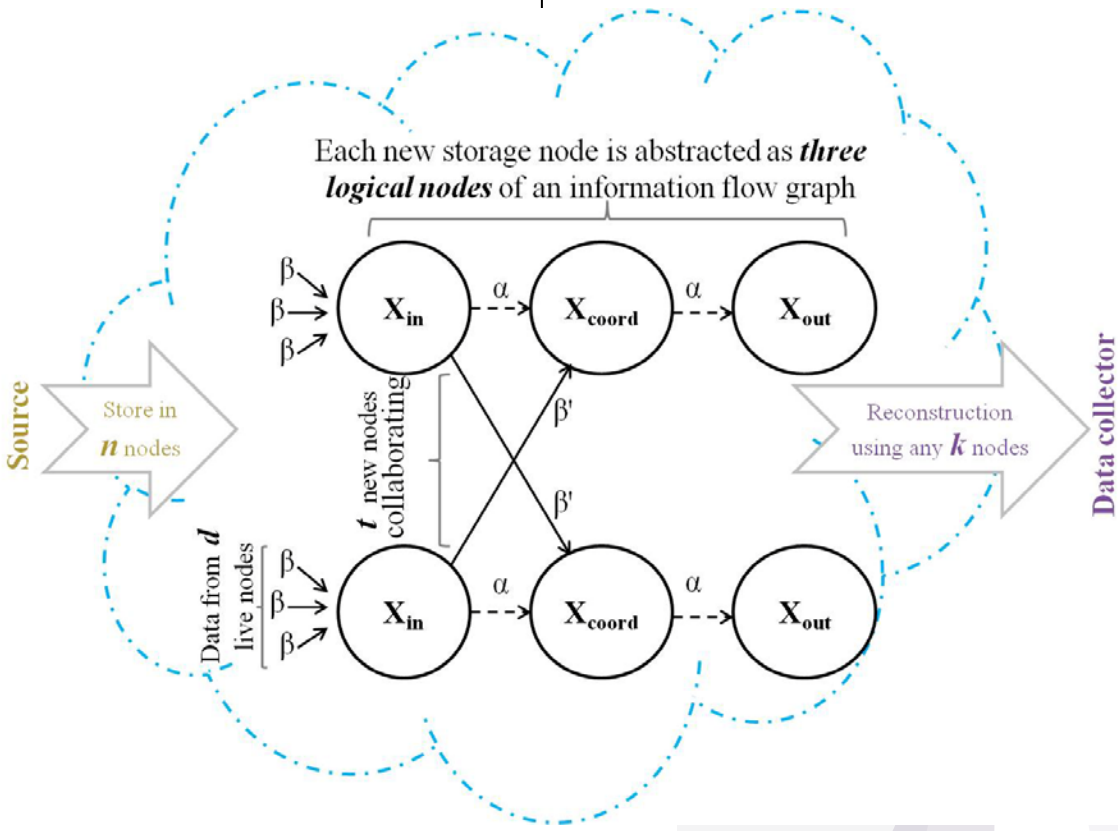
- Example code (w/ Exact Repair)
  - Construction for one failure, contacting all live nodes for repair



© 2012 A. Datta & F. Oggier, NTU Singapore

Collaborative Regenerating Codes





© 2012 A. Datta & F. Oggier, NTU Singapore

(Collaborative) Regenerating Codes

- Pros
  - Network information flow analysis determines optimal (w.r.to repair bandwidth)
    - Catch: Subject to MDS property of code (more on this, later)
  - Some proposed codes apply network coding on top of ECs
    - Inherit the properties for EC for de/coding
- Cons
  - Codes for only specific points on trade-off curve
    - Information flow analysis itself does not suggest any code
  - Restrictive
    - Some proposed codes can carry out repair only for one fault
    - Needs to contact all live nodes for repair to be optimal
  - Not simple: Algorithmic as well as system design

© 2012 A. Datta & F. Oggier, NTU Singapore

Design Space For Cheaper Repairs

- What is the best one can do (w.r.to repairs)?
  - Minimize bandwidth usage per repair
    - Regenerating codes
  - Minimize number of live nodes used per repair
    - Self-repairing codes

© 2012 A. Datta & F. Oggier, NTU Singapore

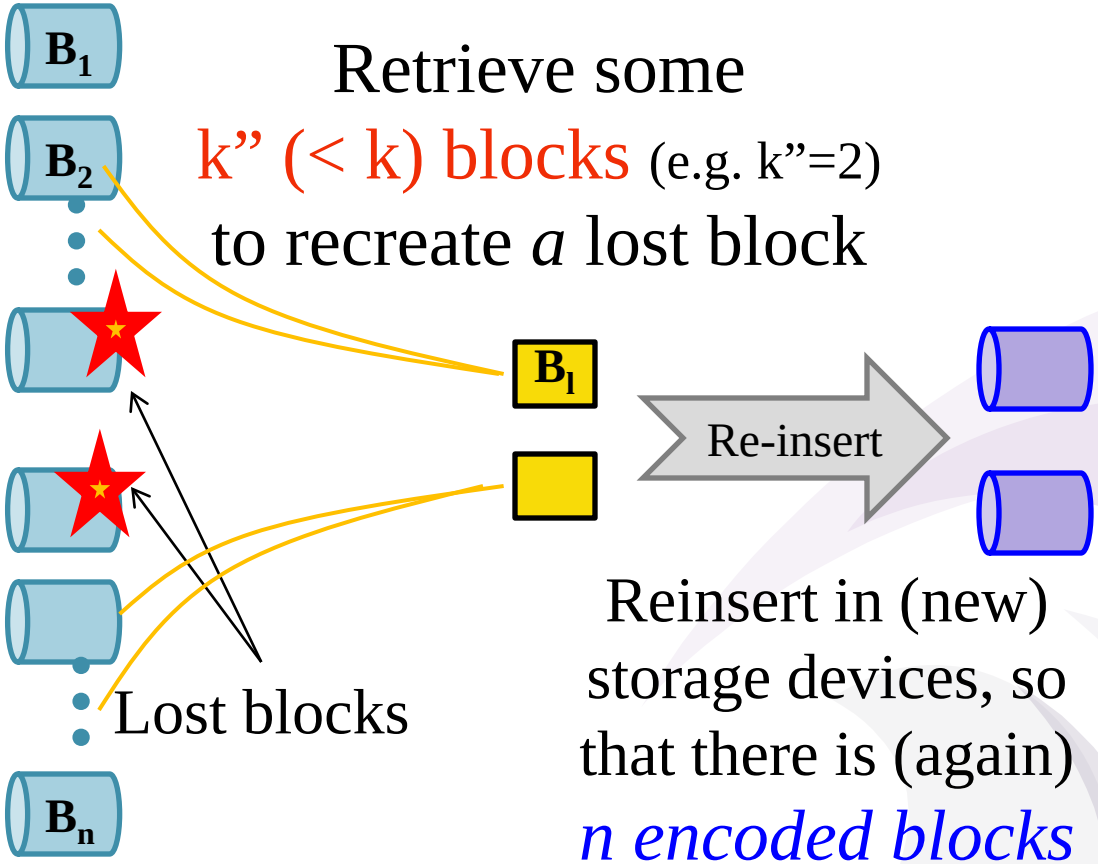
Self-repairing Codes (SRC)

- Self-repairing codes are (n, k) codes s.t.
  - encoded fragments can be repaired directly from other subsets of encoded fragments.
  - a fragment can be repaired from a fixed number of encoded fragments, independently of which specific blocks are missing
    - Analogous to erasure codes supporting reconstruction using any n - k losses, independently of which
  - number of live nodes contacted for repair is minimized.

Self-repairing Homomorphic Codes for Distributed Storage Systems
Frédérique Oggier and Anwitaman Datta
Infocom 2011

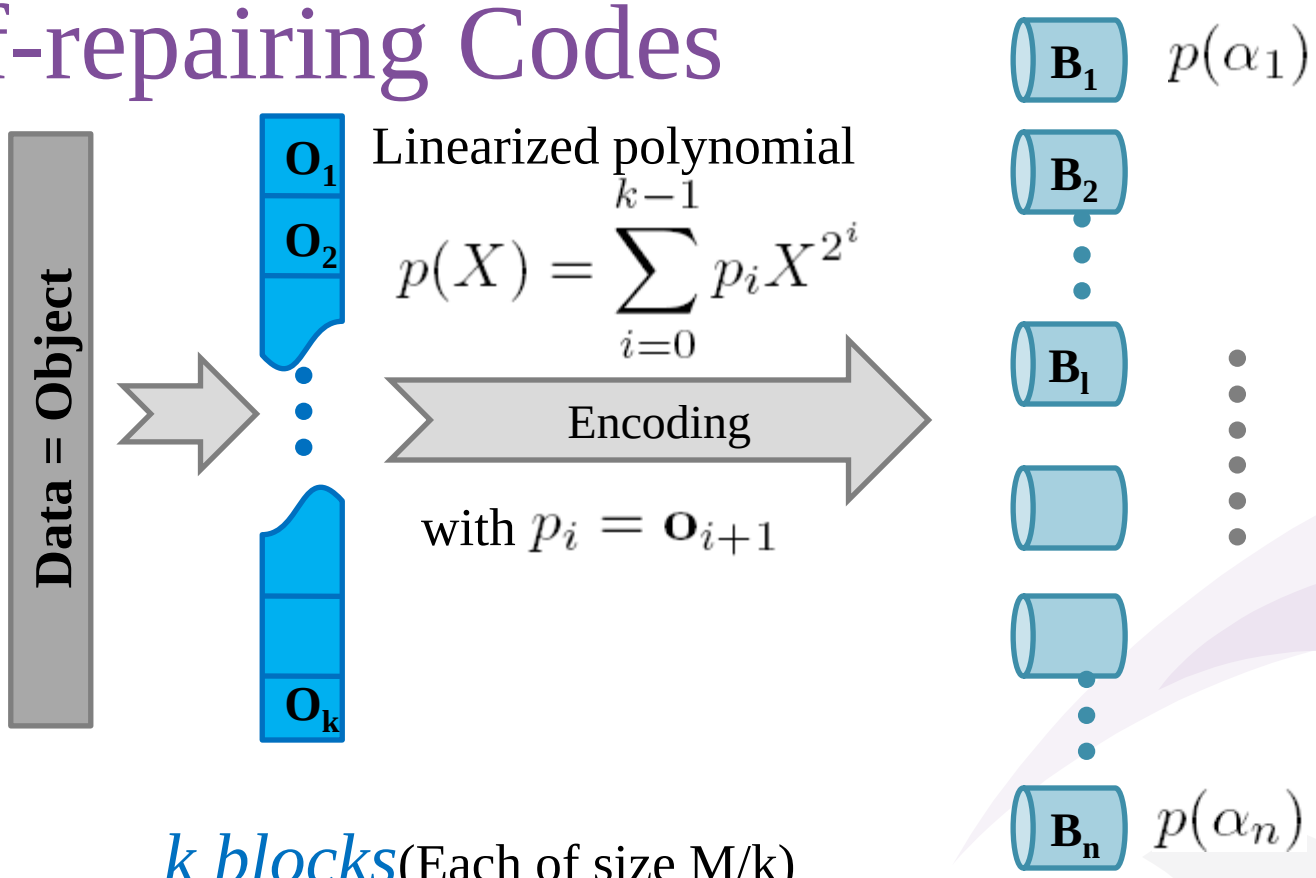
© 2012 A. Datta & F. Oggier, NTU Singapore

Self-repairing Codes: Blackbox View



© 2012 A. Datta & F. Oggier, NTU Singapore

Self-repairing Codes



k blocks (Each of size M/k)

n encoded blocks

- There is at least one pair to repair a node, for up to (n-1)/2 simultaneous failures (Parallel & fast repair of multiple faults)

© 2012 A. Datta & F. Oggier, NTU Singapore

One Example

Self-repairing Codes for Distributed Storage Systems – A Projective Geometric Construction
Frédérique Oggier and Anwitaman Datta **ITW 2011**

node	basis vectors	data stored
N_1	$v_1 = (1000),\ v_2 = (0110)$	$\{o_1, o_2 + o_3\}$
N_2	$v_3 = (0100),\ v_4 = (0011)$	$\{o_2, o_3 + o_4\}$
N_3	$v_5 = (0010),\ v_6 = (1101)$	$\{o_3, o_1 + o_2 + o_4\}$
N_4	$v_7 = (0001),\ v_8 = (1010)$	$\{o_4, o_1 + o_3\}$
N_5	$v_9 = (1100),\ v_{10} = (0101)$	$\{o_1 + o_2, o_2 + o_4\}$

Repair using *two* nodes

Say N_1 and N_3

$(o_1+o_2+o_4) + (o_1) \Rightarrow \mathbf{o_2+o_4}$

$(o_3) + (o_2+o_3) \Rightarrow \mathbf{o_2}$

$(o_1) + (\mathbf{o_2}) \Rightarrow \mathbf{o_1+ o_2}$

Four pieces needed to regenerate two pieces

Repair using *three* nodes

Say N_2, N_3 and N_4

$(o_2) + (o_4) \Rightarrow \mathbf{o_2+ o_4}$

$(o_1+o_2+o_4) + (o_4) \Rightarrow \mathbf{o_1+o_2}$

Three pieces needed to regenerate two pieces

One Example: Reconstruction

node	basis vectors	data stored
N_1	$v_1 = (1000),\ v_2 = (0110)$	$\{o_1, o_2 + o_3\}$
N_2	$v_3 = (0100),\ v_4 = (0011)$	$\{o_2, o_3 + o_4\}$
N_3	$v_5 = (0010),\ v_6 = (1101)$	$\{o_3, o_1 + o_2 + o_4\}$
N_4	$v_7 = (0001),\ v_8 = (1010)$	$\{o_4, o_1 + o_3\}$
N_5	$v_9 = (1100),\ v_{10} = (0101)$	$\{o_1 + o_2, o_2 + o_4\}$

$\mathbf{o_3}$

$\mathbf{o_4}$

$(o_3) + (o_1+o_3) \Rightarrow \mathbf{o_1}$

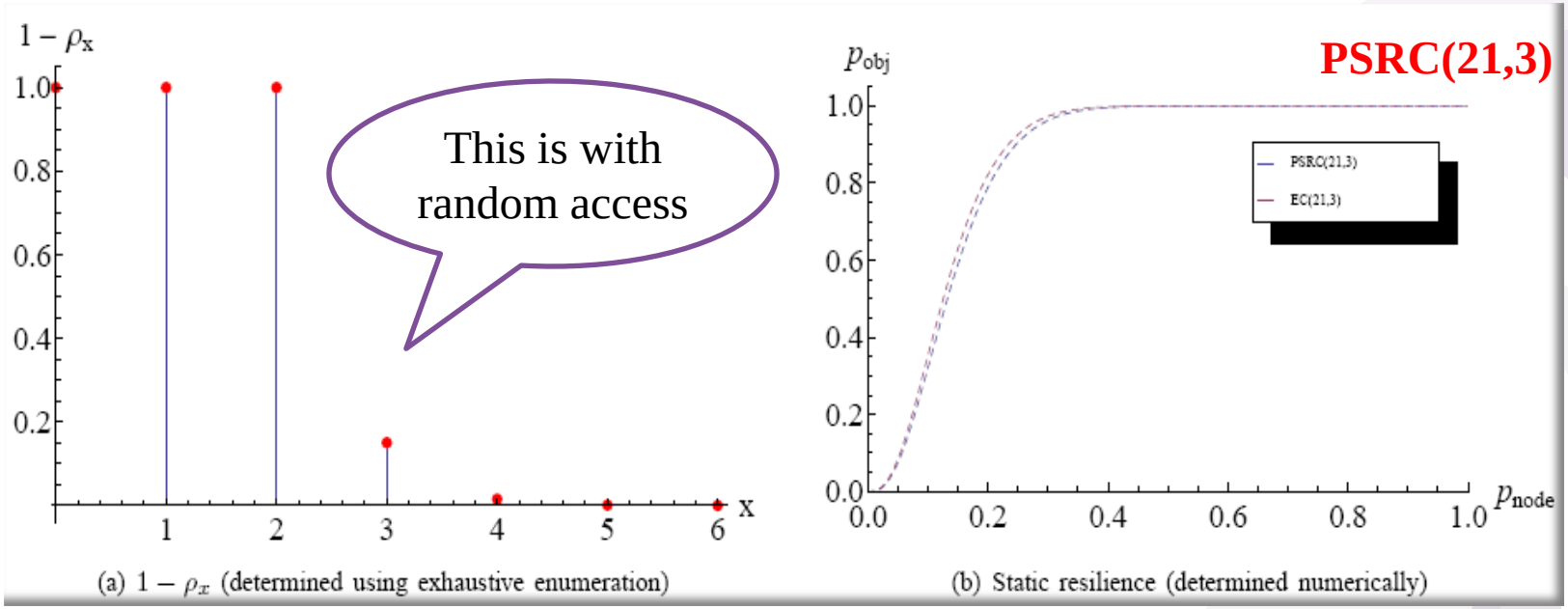
$(\mathbf{o_1}) + (o_4) + (o_1+o_2+o_4) \Rightarrow \mathbf{o_2}$

Reconstruction, say using N_3, N_4 and N_5

© 2012 A. Datta & F. Oggier, NTU Singapore

Maximum Distance Separable (MDS)?

- SRC is not MDS (and can not be!)
  - Does it matter?
    - **No**t much
    - In practice, access will be "planned" …
  - PSRC needs less bandwidth than `optimal' RGC!



© 2012 A. Datta & F. Oggier, NTU Singapore

Practical properties

- (Current) SRCs are not systematic
  - PSRC is "like systematic"
  - Need to contact more nodes (than k)
    - To obtain systematic `pieces'
    - Same total bandwidth usage
      - Parallel download for access can even be an `advantage'
    - `mixed' strategies for access, i.e. get some systematic pieces, and some others …
      - **Power saving** (by switching off nodes) strategies possible
- Coding/decoding in PSRC are both using
  - XOR operations only

node	basis vectors	data stored
N_1	$v_1 = (1000),\ v_2 = (0110)$	$\{o_1, o_2 + o_3\}$
N_2	$v_3 = (0100),\ v_4 = (0011)$	$\{o_2, o_3 + o_4\}$
N_3	$v_5 = (0010),\ v_6 = (1101)$	$\{o_3, o_1 + o_2 + o_4\}$
N_4	$v_7 = (0001),\ v_8 = (1010)$	$\{o_4, o_1 + o_3\}$
N_5	$v_9 = (1100),\ v_{10} = (0101)$	$\{o_1 + o_2, o_2 + o_4\}$

© 2012 A. Datta & F. Oggier, NTU Singapore

Outlook

- Interested to
  - Follow: **http://sands.sce.ntu.edu.sg/CodingForNetworkedStorage/**
  - Get involved: {anwitaman,frederique}@ntu.edu.sg

© 2012 A. Datta & F. Oggier, NTU Singapore

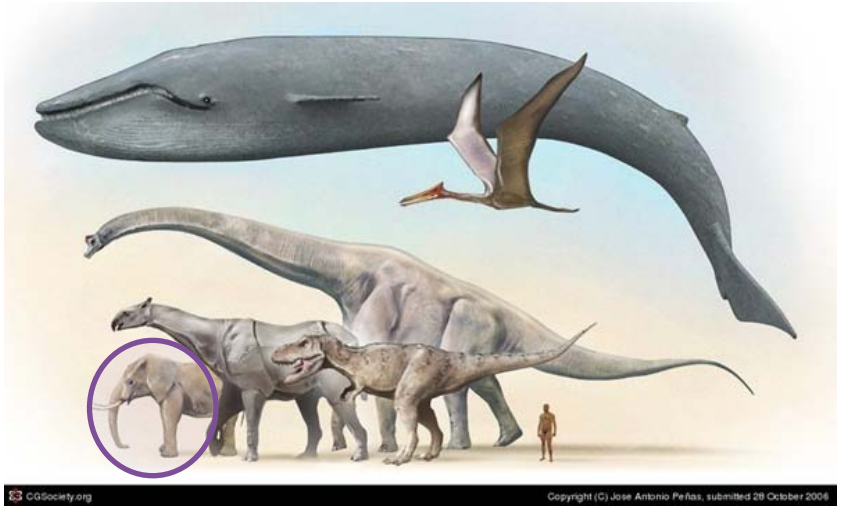
Appendix

MapReduce: Simplified Data Processing on Large Clusters
Jeffrey Dean and Sanjay Ghemawat
OSDI 2004

© 2012 A. Datta & F. Oggier, NTU Singapore

Map-Reduce

- Introduced by Google in 2004
  - Also available as open source Hadoop (w/ HDFS, etc)



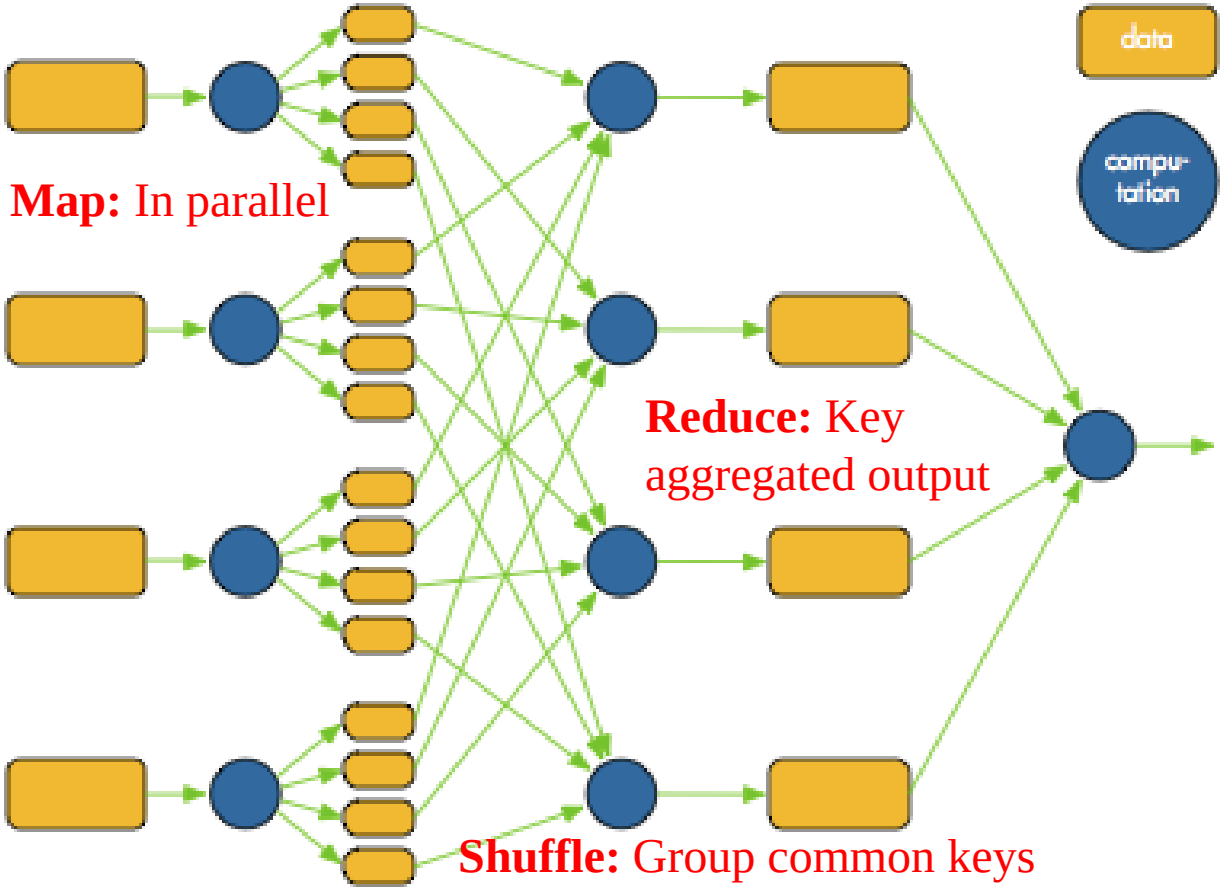
Systems of elephantine proportion needed

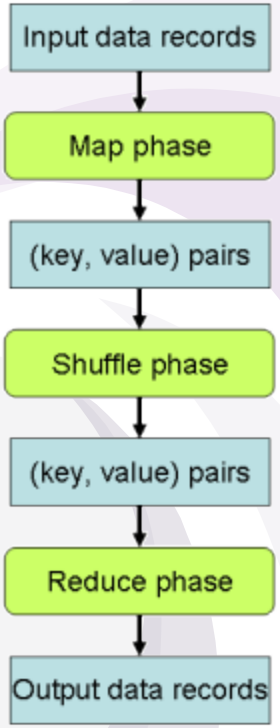
Distributed Processing of Big Data

© 2012 A. Datta & F. Oggier, NTU Singapore

The big picture

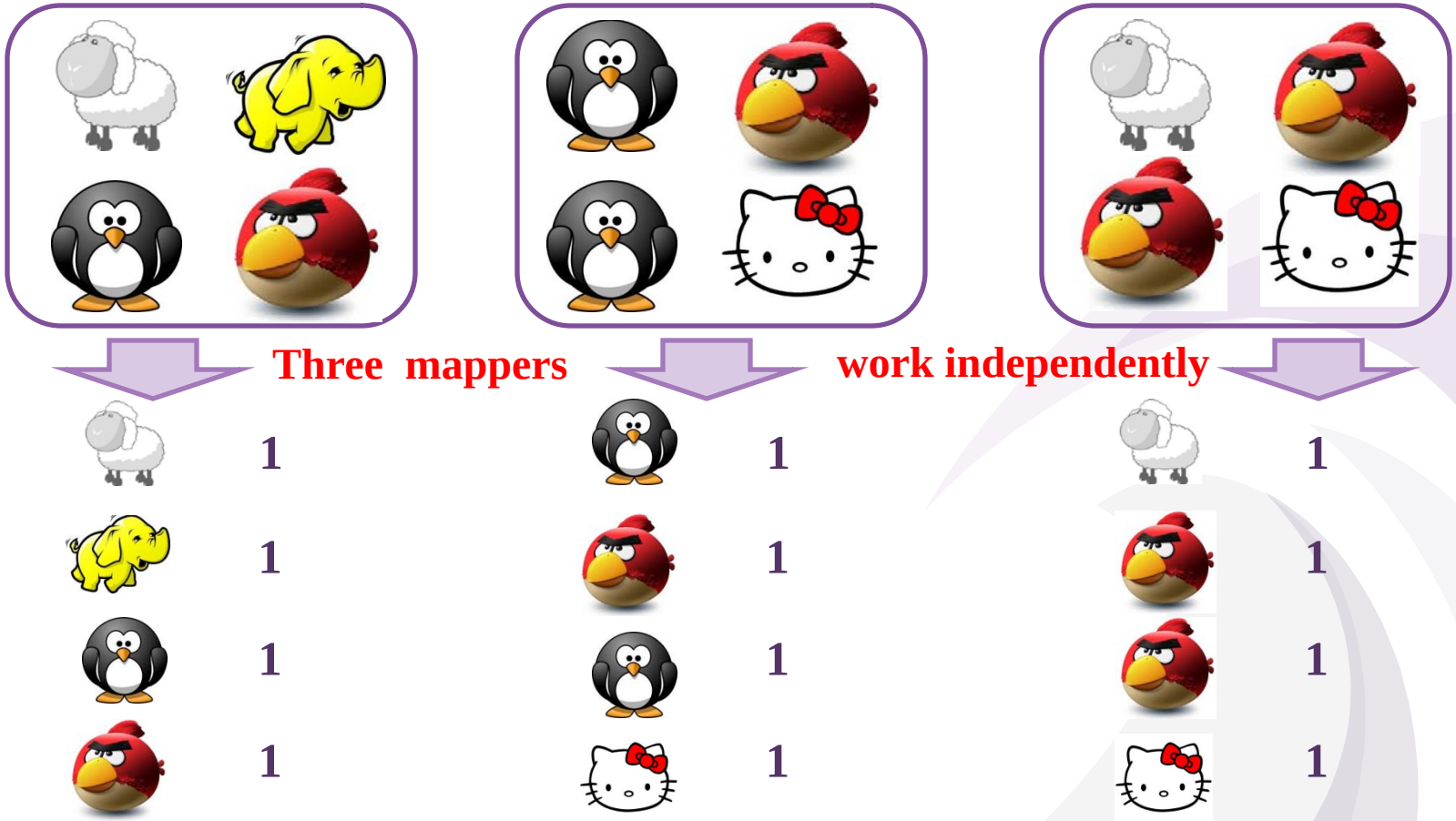
- Map: Transforms input data to intermediate (key, value) pair
- Reduce: Transforms all values for given key to final output





"Toy" example: Map

- Canonical example: Count



© 2012 A. Datta & F. Oggier, NTU Singapore

Shuffle

Key based grouping/sorting

© 2012 A. Datta & F. Oggier, NTU Singapore

Reduce

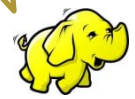
A real computation typically involves multiple rounds of MapReduce


1, 1, 1, 1


1, 1


1, 1


1, 1, 1


1

two reducers

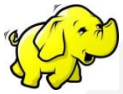

4


2

Final Output


2


3


1

© 2012 A. Datta & F. Oggier, NTU Singapore