

Cooperative Storage Systems

Ken Birman

Cornell University. CS5410 Fall 2008.



Cooperative Storage

- Early uses of P2P systems were mostly for downloads
- But idea of cooperating to store documents soon emerged as an interesting problem in its own right
 - For backup
 - As a cooperative way to cache downloaded material from systems that are sometimes offline or slow to reach
 - In the extreme case, for anonymous sharing that can resist censorship and attack
- Much work in this community... we'll focus on some representative systems



Storage Management and Caching in PAST

- System Overview
- Routing Substrate
- Security
- Storage Management
- Cache Management



PAST System Overview

- PAST (Rice and Microsoft Research)
 - Internet-based, self-organizing, P2P global storage utility
 - Goals
 - Strong persistence
 - High availability
 - Scalability
 - Security
 - Pastry
 - Peer-to-Peer routing scheme



PAST System Overview

- API provided to clients
 - `fileId = Insert(name, owner-credentials, k, file)`
 - Stores a file at a user-specified number of k of diverse nodes
 - `fileId` is computed as the secure hash (SHA-1) of the file's name, the owner's public key and a seed
 - `file = Lookup(fileId)`
 - Reliably retrieves a copy of the file identified by `fileId` from a “near” node
 - `Reclaim(fileId, owner-credentials)`
 - Reclaims the storage occupied by the k copies of the file identified by `fileId`
 - `fileId` – 160 bits identifier among which 128 bits form the most significant bits (msb)
 - `nodeId` – 128-bit node identifier



Storage Management Goals

- Goals
 - High global storage utilization
 - Graceful degradation as the system approaches its maximal utilization
- Design Goals
 - Local coordination
 - Fully integrate storage management with file insertion
 - Reasonable performance overhead



Routing Substrate: Pastry

- PAST is layered on top of Pastry
 - As we saw last week, an efficient peer-to-peer routing scheme in which each node maintains a routing table
- Terms we'll use from the Pastry literature:
 - Leaf Set
 - 1/2 numerically closest nodes with larger nodeIds
 - 1/2 numerically closest nodes with smaller nodeIds
 - Neighborhood Set
 - L closest nodes based on network proximity metric
 - Not used for routing
 - Used during node addition/recovery



Storage Management in PAST

- Responsibilities of the storage management
 - Balance the remaining free storage space
 - Maintain copies of each file in k nodes with nodeIds closest to the fileId
 - Conflict?
- Storage load imbalance
 - Reason
 - Statistical variation in the assignment of nodeIds and fileIds
 - Size distribution of inserted files varies
 - The storage capacity of individual PAST nodes differs
 - How to overcome?



Storage Management in PAST

- Solutions for load imbalance
 - Per-node storage
 - Assume storage capacities of individual nodes differ by no more than two orders of magnitude
 - Newly joining nodes have too large advertised storage capacity
 - Split and join under multiple nodeIds
 - Too small advertised storage capacity
 - Reject



Storage Management in PAST

- Solutions for load imbalance
 - Replica diversion
 - Purpose
 - Balance free storage space among the nodes in a leaf set
 - When to apply
 - Node A, one of the k closest nodes, cannot accommodate a copy locally
 - How?
 - Node A chooses a node B in its leaf set such that
 - B is not one of the k -closest nodes
 - B doesn't hold a diverted replica of the file



Storage Management in PAST

- Solutions for load imbalance
 - Replica diversion
 - Policies to avoid performance penalty of unnecessary replica diversion
 - Unnecessary to balance storage space when utilization of all nodes is low
 - Preferable to divert a large file
 - Always divert a replica from a node with free space significantly below average to a node significantly above average



Storage Management in PAST

- Solutions for load imbalance
 - File diversion
 - Purpose
 - Balance the free storage space among different portions of the nodeId space in PAST
 - Client generates a new fileId using a different seed and retries for up to three times
 - Still cannot insert the file?
 - Retry the operations with a smaller file size
 - Smaller number of replicas (k)



Caching in PAST

- Caching
 - Goal
 - Minimize client access latencies
 - Maximize the query throughput
 - Balance the query load in the system
 - A file has k replicas. Why caching is needed?
 - A highly popular file may demand many more than k replicas
 - A file is popular among one or more local clusters of clients



Caching in PAST

- Caching Policies
 - Insertion policy
 - A file routed through a node as part of lookup or insert operation is inserted into local disk cache
 - If current available cache size * c is greater than file size
 - c is fraction
 - Replacement policy
 - GreedyDual-Size (GD-S) policy
 - Weight H_d associated with a file d , which inversely proportional to file size d
 - When replacement happens, remove file v whose H_v is the smallest among all cached files

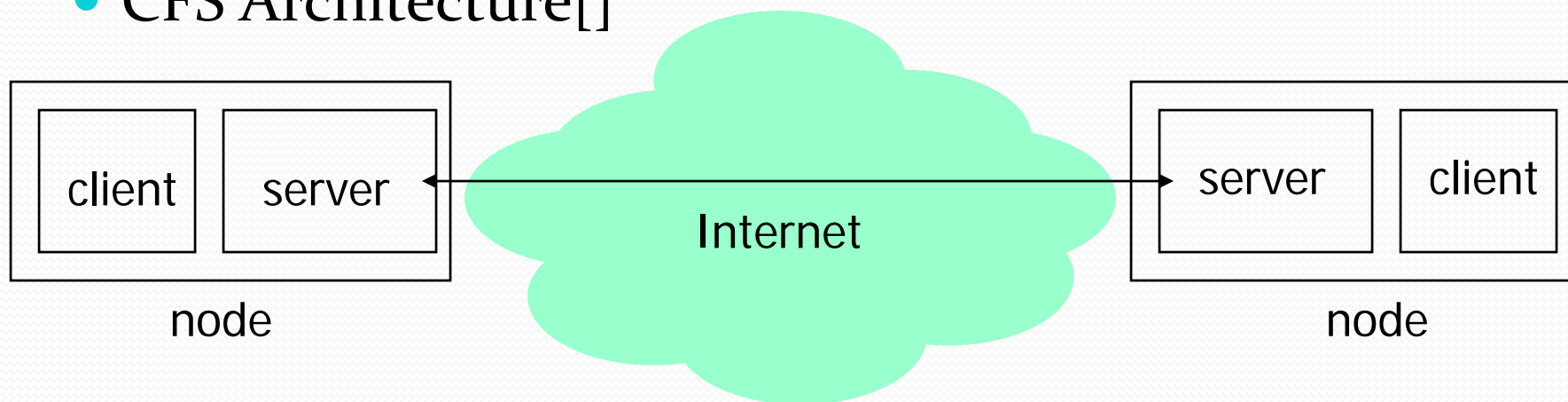


Wide-area cooperative storage with CFS

- System Overview
- Routing Substrate
- Storage Management
- Cache Management

CFS System Overview

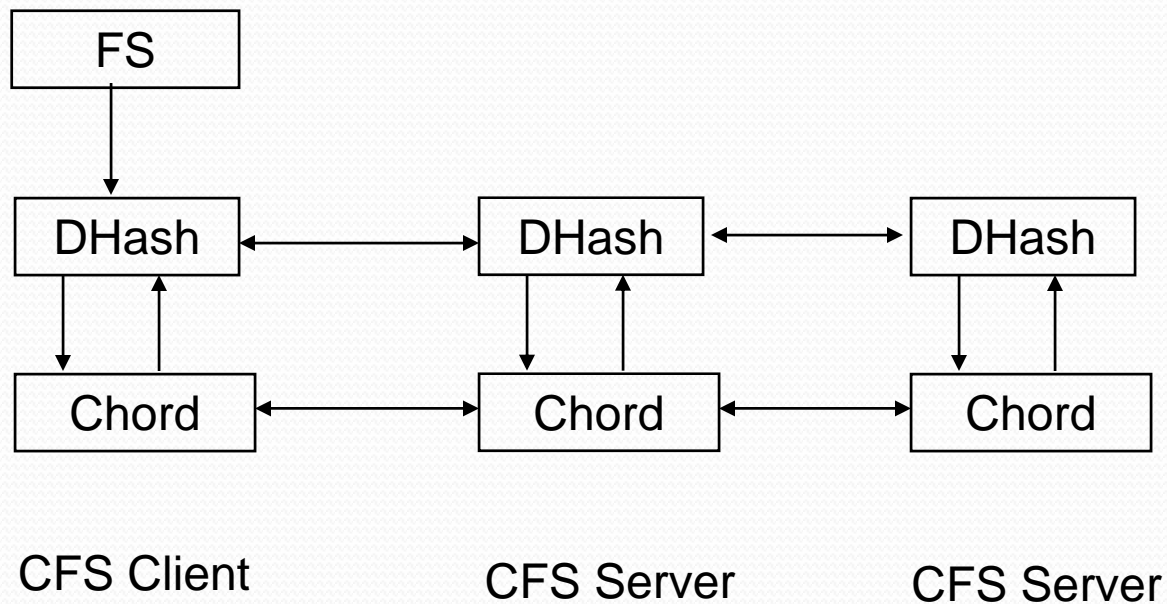
- CFS (Cooperative File System) is a P2P read-only storage system
- CFS Architecture[]



- Each node may consist of a client and a server

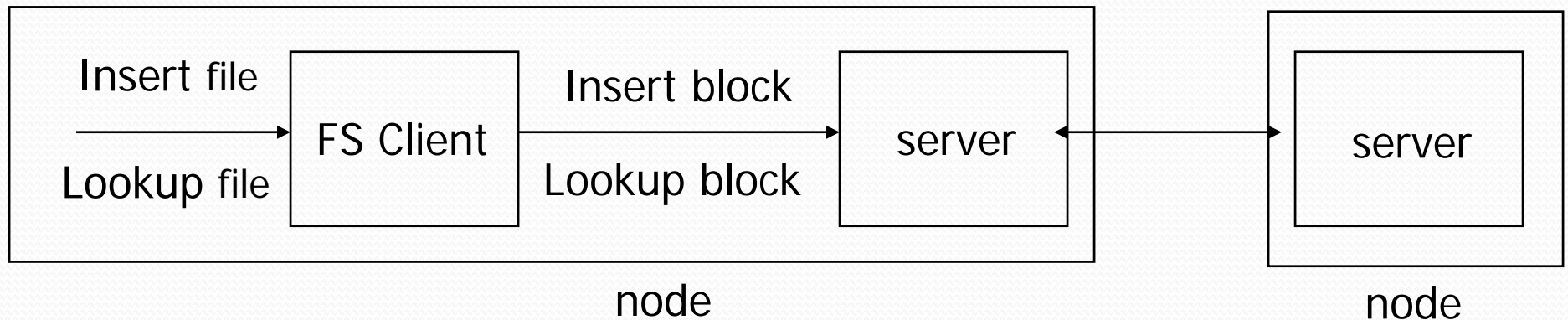
CFS System Overview

- CFS software structure



CFS System Overview

- Client-Server Interface []



- Files have unique name
- Uses the DHash layer to retrieve blocks
- Client DHash layer uses the client Chord layer to locate the servers holding desired blocks



CFS System Overview

- Publishers split files into blocks
- Blocks are distributed over many servers
- Clients is responsible for checking files' authenticity
- DHash is responsible for storing, replicating, caching and balancing blocks
- Files are read-only in the sense that only publisher can update them



CFS System Overview

- Why use blocks? []
 - Load balance is easy
 - Well-suited to serving large, popular files
 - Storage cost of large files is spread out
 - Popular files are served in parallel
- Disadvantages?
 - Cost increases in terms of one lookup per block



Routing Substrate in CFS

- CFS uses the Chord scheme to locate blocks
- Consistent hashing
- Two data structures to facilitate lookups
 - Successor list
 - Finger table



Storage Management in CFS

- Replication

- Replicate each block on k CFS servers to increase availability
- The k servers are among Chord's r -entry successor list ($r > k$)
- The block's successor manages replication of the block
- DHash can easily find the identities of these servers from Chord's r -entry successor list
- Maintain the k replicas automatically as servers come and go

Caching in CFS

- Caching
 - Purpose
 - Avoid overloading servers that hold popular data
 - Each DHash layer sets aside a fixed amount of disk storage for its cache



- Long-term blocks are stored for an agree-upon interval
 - Publishers need to refresh periodically



Caching in CFS

- Caching
 - Block copies are cached along the lookup path
 - DHash replaces cached blocks in LRU order
 - LRU makes cached copies close to the successor
 - Meanwhile expands and contracts the degree of caching according to the popularity



Storage Management vs Caching in CFS

- Comparison of replication and caching
 - Conceptually similar
 - Replicas are stored in predictable places
 - DHash can ensure enough replicas always exist
 - Blocks are stored for an agreed-upon finite interval
 - Number of cached copies are not easily counted
 - Cache uses LRU



Storage Management in CFS

- Load balance
 - Different servers have different storage and network capacities
 - To handle heterogeneity, the notion of virtual server is introduced
 - A real server can act as multiple virtual servers
 - Virtual NodeId is computed as
 - $\text{SHA-1}(\text{IP Address, index})[]$



Storage Management in CFS

- Load balance
 - Number of virtual servers is proportional to the server's storage and network capacity
 - Disadvantages of using virtual server
 - The number of hops during lookup may increase
 - How to overcome?
 - Allow virtual servers on the same physical server to examine each others' routing tables



Storage Management in CFS

- Quotas
 - Goal
 - Avoid malicious injection of large quantities of data
 - Per-publisher quotas
 - CFS bases quotas on the IP address of the publisher to avoid centralized authentication
- Updates and Deletion
 - Only the publishers are allowed to update CFS



Storage Management in CFS

- Updates and Deletion
 - CFS doesn't support explicit delete operation
 - Blocks are stored for an agreed-upon finite interval
 - Publishers must periodically refresh their blocks
 - CFS server may delete blocks that have not been refreshed recently
 - Benefit?
 - Automatically recover from malicious insertions



Comparisons of the two systems

- File storage
 - PAST stores whole files
 - CFS stores blocks
- Load balance
 - PAST: Replication Diversion, File Diversion
 - CFS: Virtual Server
- Caching
 - Both cache copies along lookup path



But could they thrash?

- Intended behavior assumes this copying is pretty fast
 - We fix the edge of the ring... fix up the replicas... done
- Actual behavior: could be so slow that on expectation, more churn will already have happened before the copying terminates
 - In this case further rounds of copying and rebalancing need to happen
 - Vision: a form of “thrashing”, like when a VM system gets overloaded because programs have poor hit rates
 - Nobody knows if this happens in the wild...



Censor-Resistant storage

- Work in this area assumes that the documents stored in a P2P storage system aren't just random stuff
 - Why use P2P in the first place?
 - Mazieres and his colleagues suspect that it is to ensure freedom of speech even in climates with censorship
- Their goal?
 - A collaborative storage system that maintains document availability in the presence of adversaries who wish to suppress the document.
 - Also makes it possible to deny that you were the author of the document

Why Censorship-Resistant Publishing?

- Political Dissent



- “Whistleblowing”




- Human Rights Reports





Possible Solutions

- Collection of WWW servers
 - CGI scripts to accept files
 - each file replicated on other participating servers
- Usenet
 - Send file to Usenet server
 - Automatically replicated via NNTP
- Tangler
 - Uses a P2P overlay to solve the problem



The Tangler Censorship-Resistant Publishing System

- Designed to be a practical and implementable censorship-resistant publishing system.
- Addresses some deficiencies of previous work
- Contributions include –
 - A unique publication mechanism called *entanglement*
 - The design of a self-policing storage network that ejects faulty nodes



Tangler Design

- Small group (<100) of volunteer servers
- Each server has public/private key pair
- Each server donates disk space to system (publishing limit)
- Agreement on volunteer servers, public keys and donated disk space
- Published documents are divided into equal sized blocks, and combined with blocks of previously published documents (*entanglement*)
- Entangled blocks are stored on servers
- Each server verifies other servers compliance with Tangler protocols

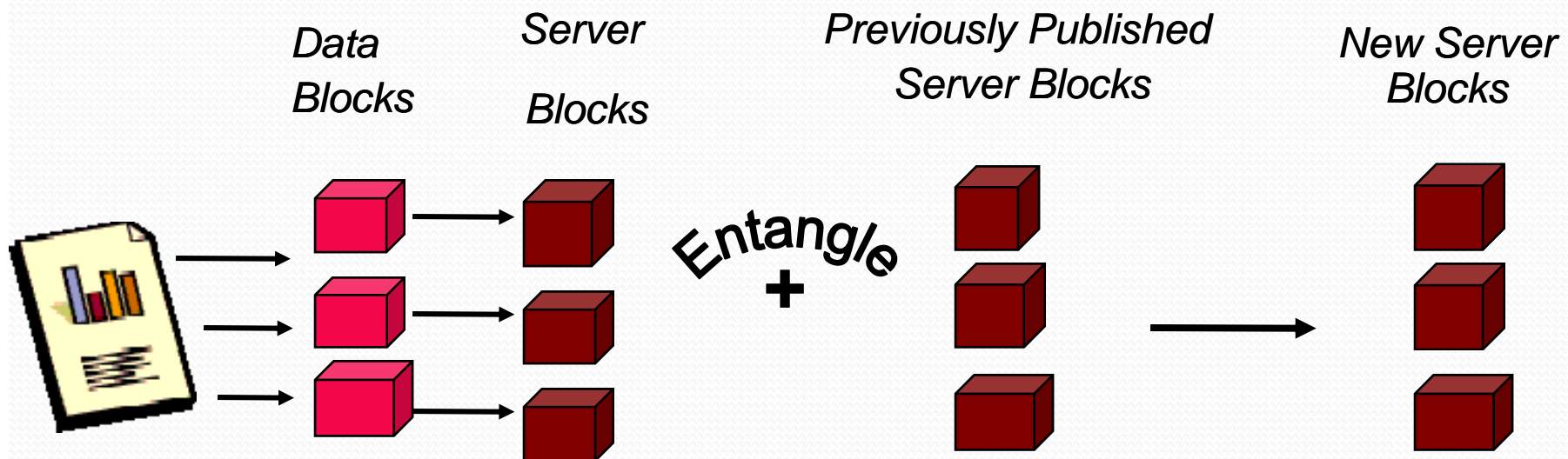


Tangler Goals

- Anonymity – Users can publish and read documents anonymously
- Document availability through replication
- Integrity guarantees on data (tamper & update)
- No server is storing objectionable documents
 - Decoupling between document and blocks
 - Blocks not permanently tied to specific servers
 - Server cannot chose which blocks to store or serve
- Misbehaving servers should be ejected from system

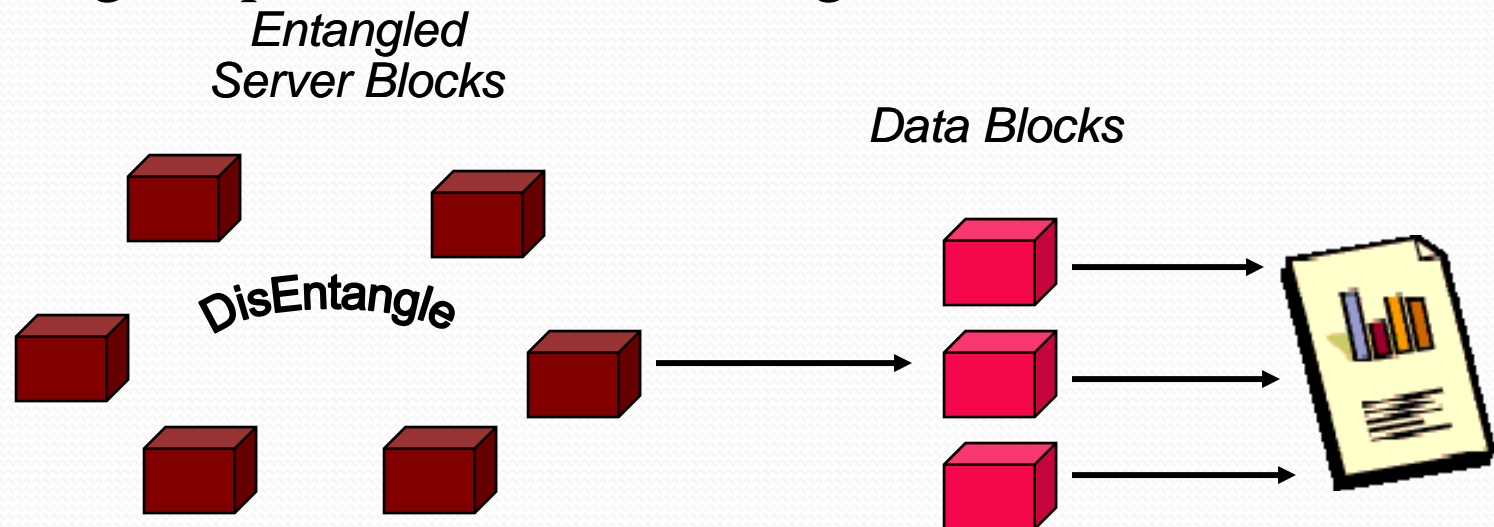
Publish Operation

- Document broken into *data blocks*
- Data blocks transformed into *server blocks*
- Server blocks combined with those of previously published server blocks (*entanglement*)
- Entangled server blocks are stored on servers



Document Retrieval Operation

- Retrieve entangled server blocks from servers
- Entanglement is fault tolerant – don't need all entangled blocks to re-form data blocks
- DisEntangle Operation re-forms original data blocks



Block Entanglement Algorithm

- Utilizes Shamir's Secret Sharing Algorithm
 - Given a secret S can form n shares
 - Any k of them can re-form S
 - Less than k shares provide no information about S
- Entanglement is a secret sharing scheme with $n=4$ and $k=3$
- Two shares are previously published server blocks
- Two additional shares are created



Benefits Of Entanglement

- Dissociates blocks served from documents published
 - Single block belongs to multiple documents
 - Servers just hosting blocks
- Incentive
 - Cache server blocks of entangled documents
 - Monitor availability of other server blocks
 - Re-inject blocks that have been deleted



Tangler Servers (Tangle-Net)

- All servers fall into one of two categories –
non-faulty = follow Tangle protocols
faulty = servers that exhibit Byzantine failures
- All **non-faulty** servers are synchronized to within 10 minutes of correct time.
- Time is divided into *rounds* (24 hour period)
 - Round 0 = Jan 1, 2002 (12:00AM)
- Fourteen consecutive rounds form an *epoch*



Tangler Round

- Round Activity (concurrent actions)
 - Request storage tokens from other servers
 - Grant storage tokens to other servers
 - Send and receive blocks
 - Monitor protocol compliance of other servers
 - Process join requests
 - Entangle new collections and retrieve old collections
- End of round
 - Commit to blocks received from servers (Merkle Tree)
 - Generate public/private key pair for the round
 - Broadcast next round commitment and public key

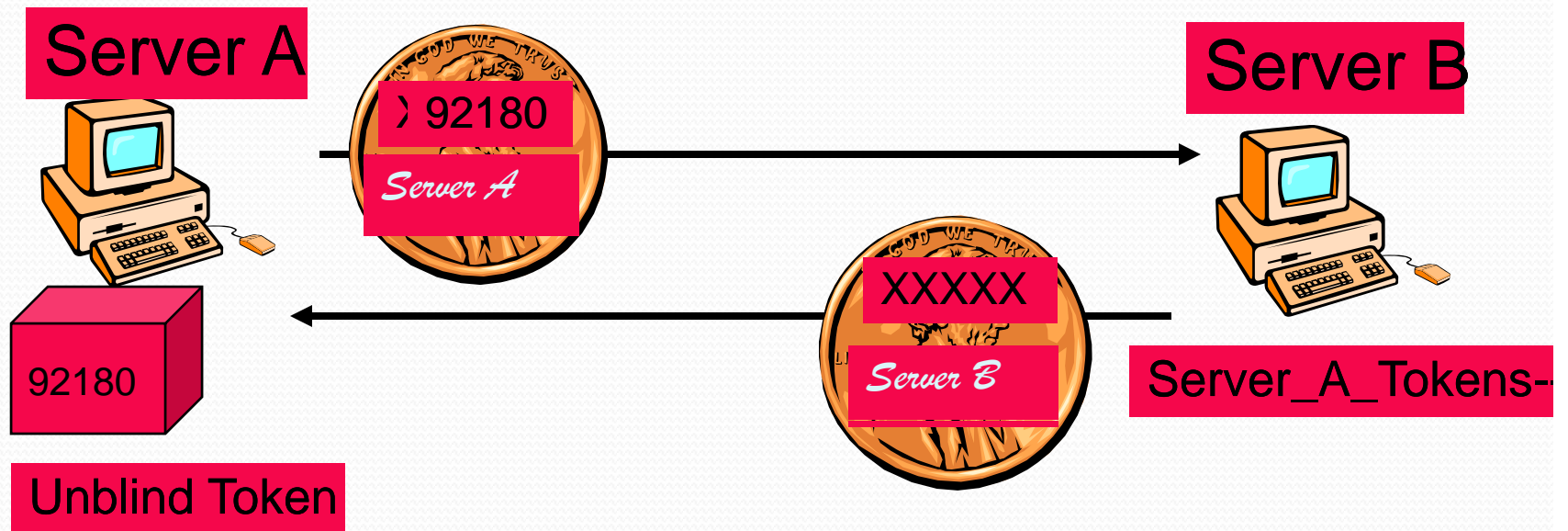


Storage Tokens

- Two step protocol to store blocks
- First Step - Acquire storage tokens
 - Every server entitled to number of storage tokens from every other server
 - Tokens acquired **non-anonymously**, requests are signed by requestor
- Second Step – Redeem Token
 - Send block & token anonymously to storing server
 - Anonymous communication supported by Mix-Net

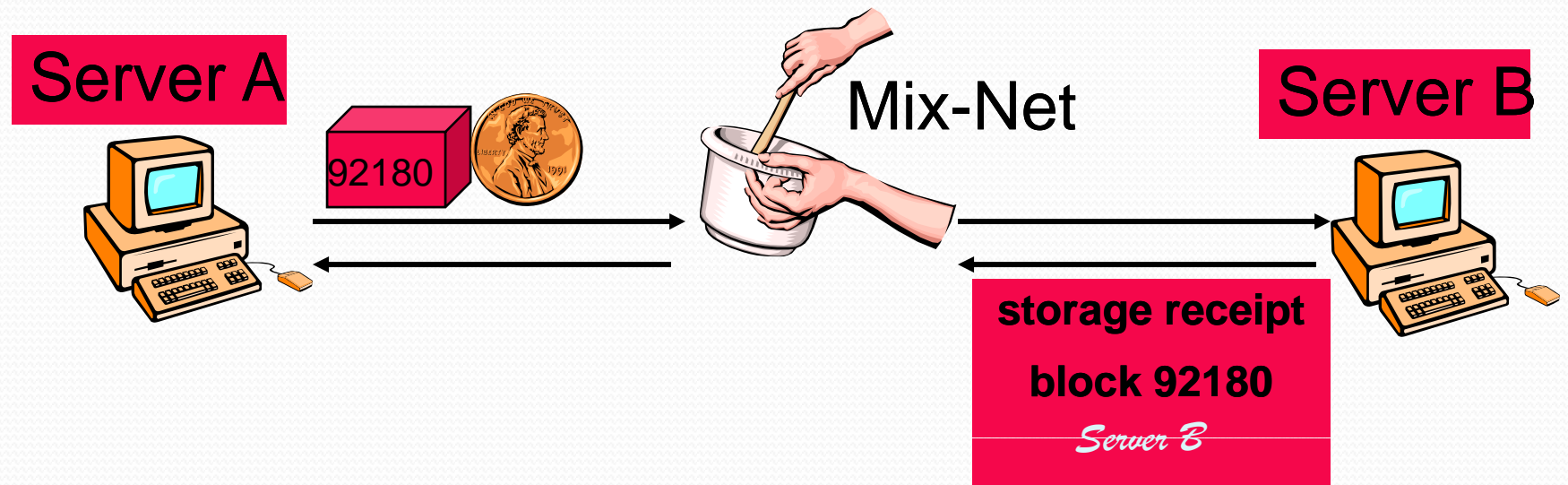
Storage Token Request

- Server A wants to store block 92180 on Server B
- Server A creates a blinded request for a token
- The blinded request is sent to server B
- Server B signs the request and returns it to A
- Server A unblinds request obtaining the token



Redeeming A Token

- Server A sends token & block through Mix-Net to B
- Server B checks token signature, stores block, and returns signed receipt over Mix-Net
- Server B commits to hash tree of all blocks





Membership Changes

- At end of epoch all non-faulty servers perform Byzantine Consensus algorithm
- Each server can vote out any other members
- New servers can join at any time but must serve as a storage-only server for a probationary period of two complete epochs
- A probationary server is admissible if it was not ejectable for at least two consecutive epochs.
- Majority vote wins



Threats

- Majority of servers are adversarial
 - Adversarial servers join
 - Force non-faulty servers off
- Publishing server discovery
 - Force suspected server off network
 - Should be able to republish on another server but may not have same credit limit
- Probabilistic failure (difficult to remove)



Summary

- P2P cooperative storage has been a major research area for the community looking at network overlays
 - Basically, they build an overlay somehow
 - Then store files in it
 - Much thought has gone into robustness
- Tangler is the “iron clad tank” of P2P cooperative storage; PAST and CFS are relatively light weight
- But one worry is that all of these systems may suffer from forms of thrashing driven by churn