# Fast and Secure Distributed Read-Only File System

Kevin Fu, M. Frans Kaashoek
MIT Laboratory for Computer Science

David Mazières
NYU Computer Science Department

`http://www.fs.net/`

# Much read-only data improperly trusted

- **People install/upgrade software over the Internet**
  - No guarantee you are talking to the right host
  - No guarantee server has not been compromised
  - No guarantee you can trust a mirror site's owner
- **Central servers configure/upgrade machines**
  - *sup*, anonymous *rsync*, AFS read-only—all insecure
- **People base financial decisions on public data**
  - Stock quotes, financial news

# Why people avoid security

- **Performance**
  - Public-key cryptography can cripple throughput (e.g. SSL)
- **Scalability and reliability**
  - Widespread replication essential for popular data
  - The more replicas, the less they can be trusted
- **Convenience**
  - Most users will skip optional verification steps
  - Often hard to understand precise security guarantees

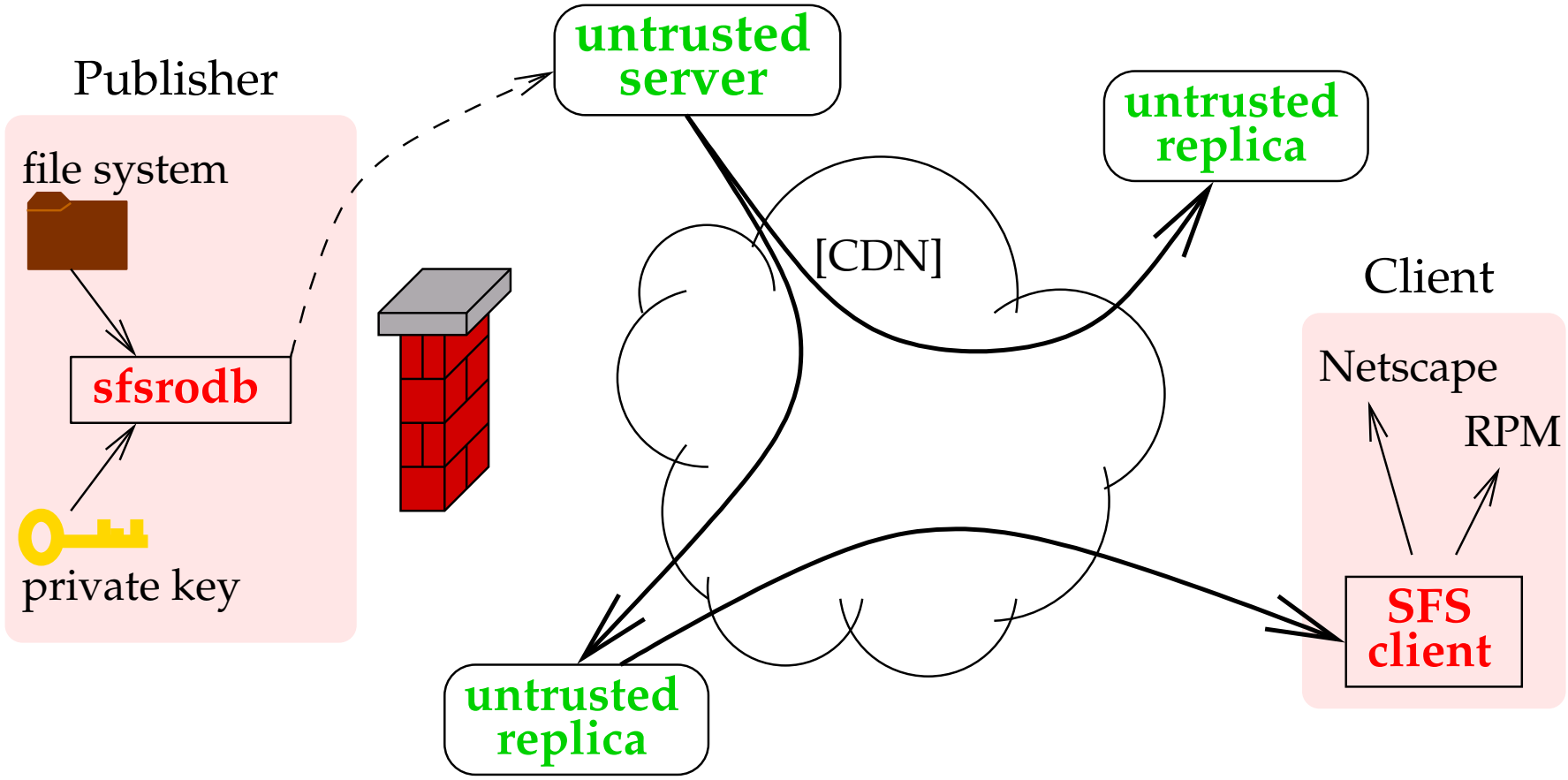
# Example: PGP-sign data off-line

- **Advantages:**
  - Compromising server does not circumvent PGP security
  - Data can be replicated on untrusted servers
- **Not general purpose**
- **Most users will ignore signatures**
- **Requires continued attention of user**
  - Was file signed by authoritative key?
  - Is a signed file the latest version?
  - Does signed contents of file match file name?
  - Were two separately signed files published together?

# Solution: SFS read-only file system

- **Convenience: Use the file system interface**
  - Publish any data
  - Access it from any application
- **Scalability: Separate publishing from distribution**
  - Off-line publisher produces signed database
  - On-line servers/replicas completely untrusted
- **Intrinsic security: Nothing for user to do**
  - Every file system has a public key (specified in name)
  - Client automatically verifies integrity of files

# SFSRO Architecture

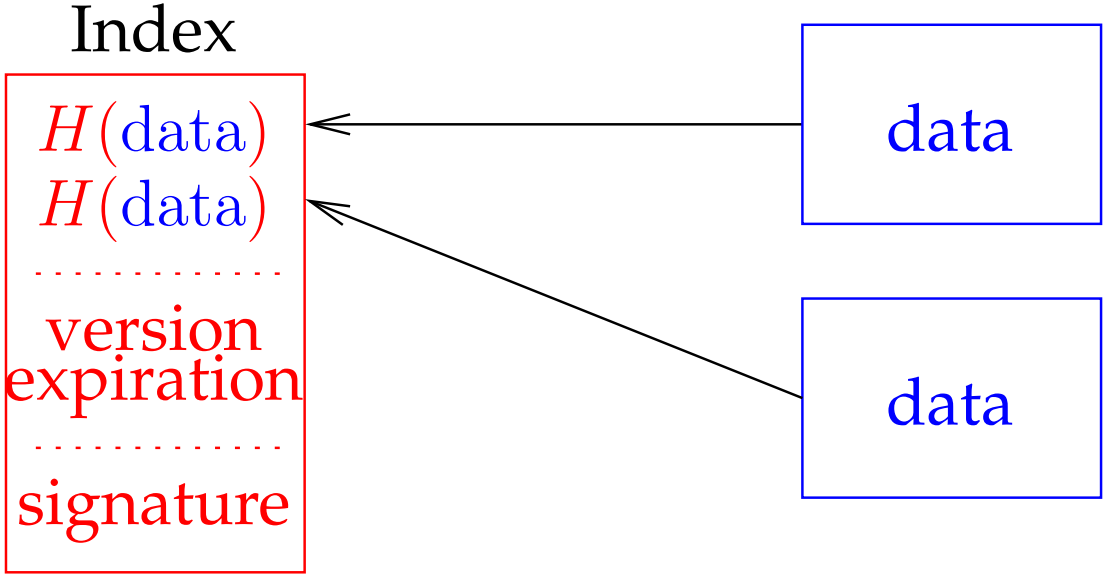



- **Publisher stores files in replicated database**
- **Clients verify files without trusting servers**

# Cryptographic primitives

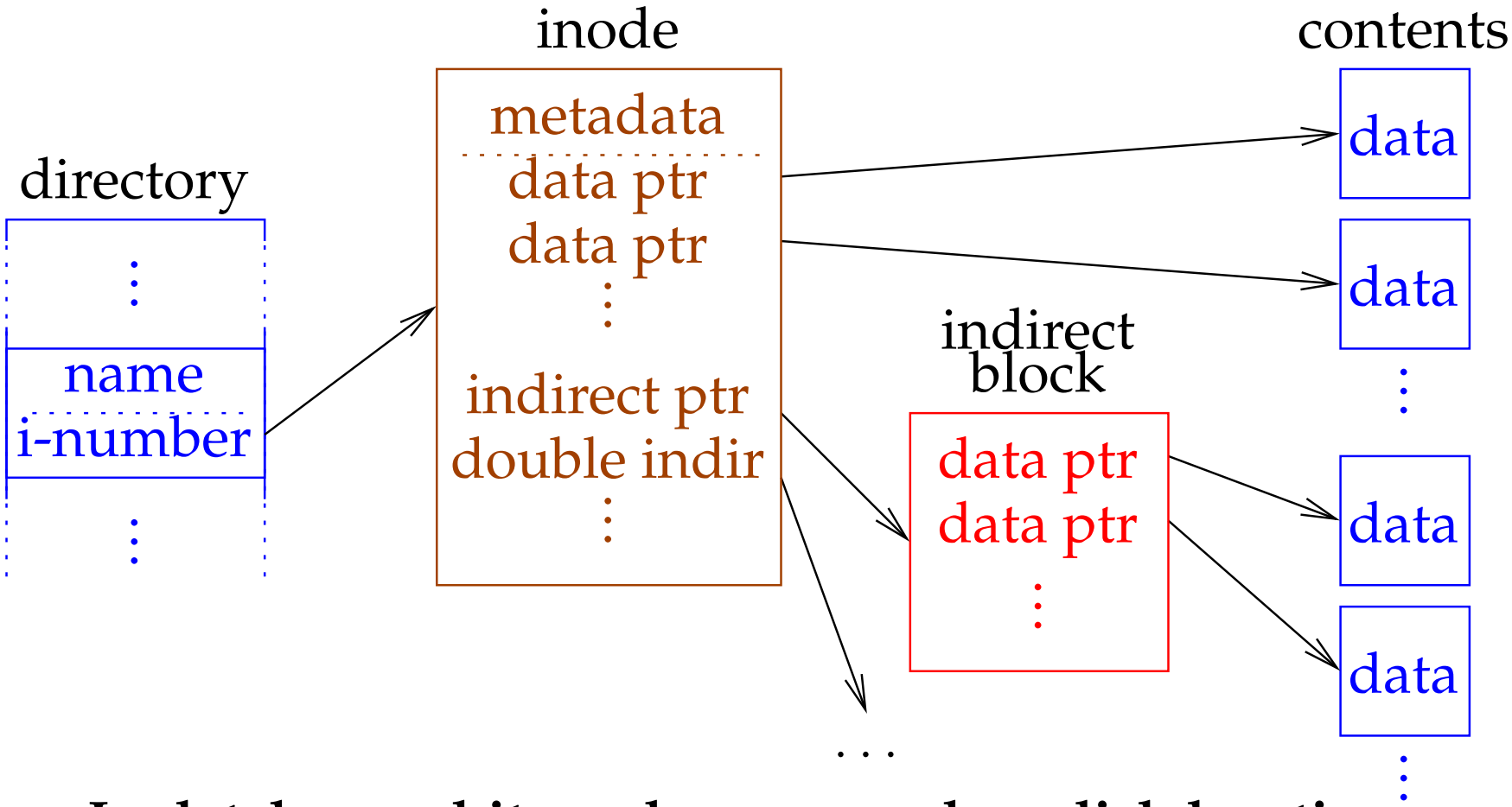
- **Digital signatures**
  - Client knows server public key in advance
  - When server signs data, client can verify integrity
  - Cost: $\sim$24 msec to sign, $\sim$80 $\mu$sec to verify
  - If server signs multiple versions, must ensure freshness
- **Collision-resistant hashes (Computationally infeasible to find** $x \neq y,\ H(x) = H(y)$**)**
  - Server hashes data securely, transmits hash to client
  - Client hashes untrusted copy, compares to trusted hash
  - Cost: 15+ MBytes/sec to hash

# Example: Publishing 2 blocks of data



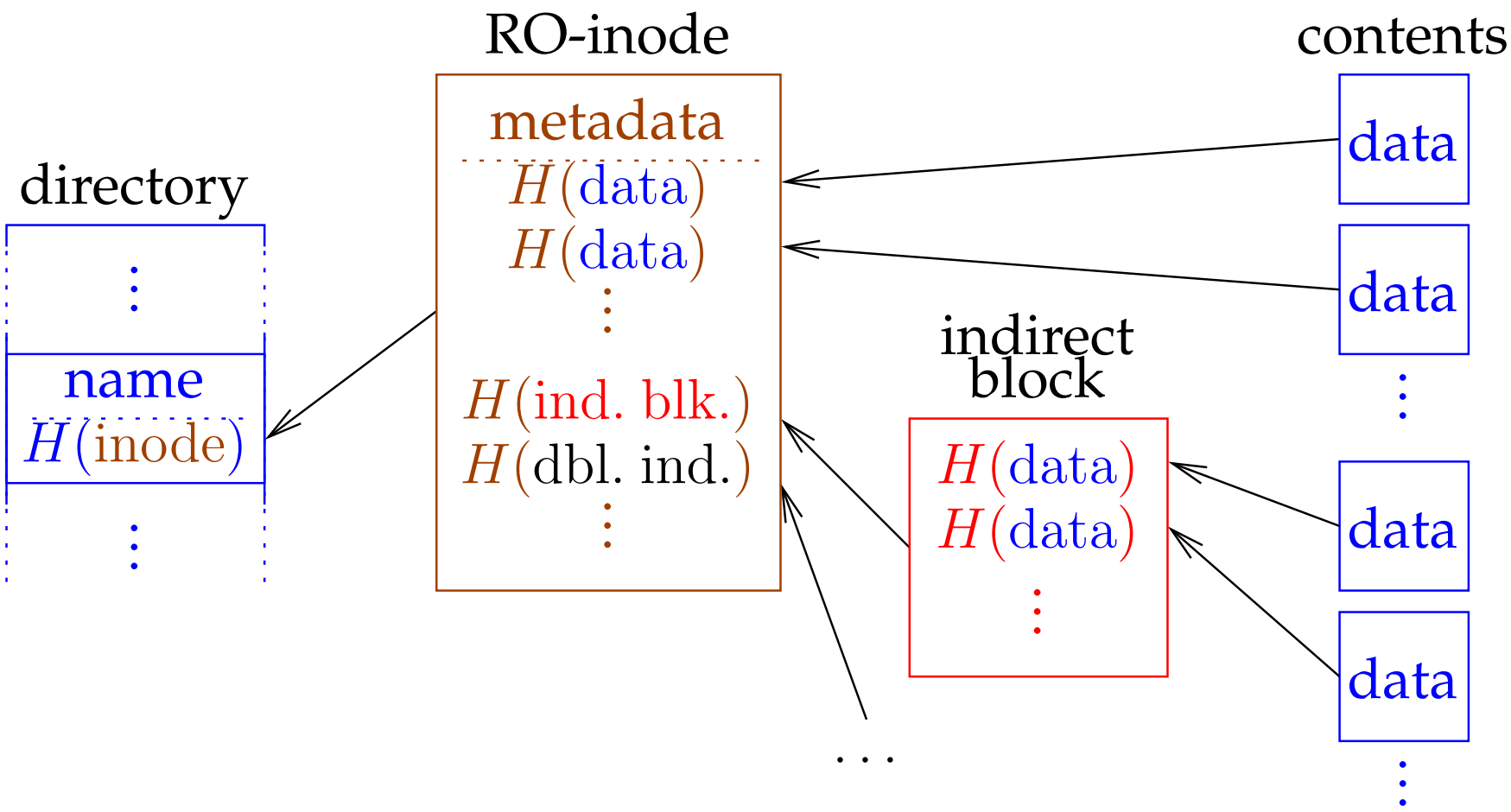

- **Digitally sign version & hashes of blocks**
  - Can verify one block without having the other
  - Two blocks must come from same version of file
- **Generalize technique to an entire file system**

# Traditional FS data structures




- **In database arbitrary key can replace disk location**

# Read-only data structures



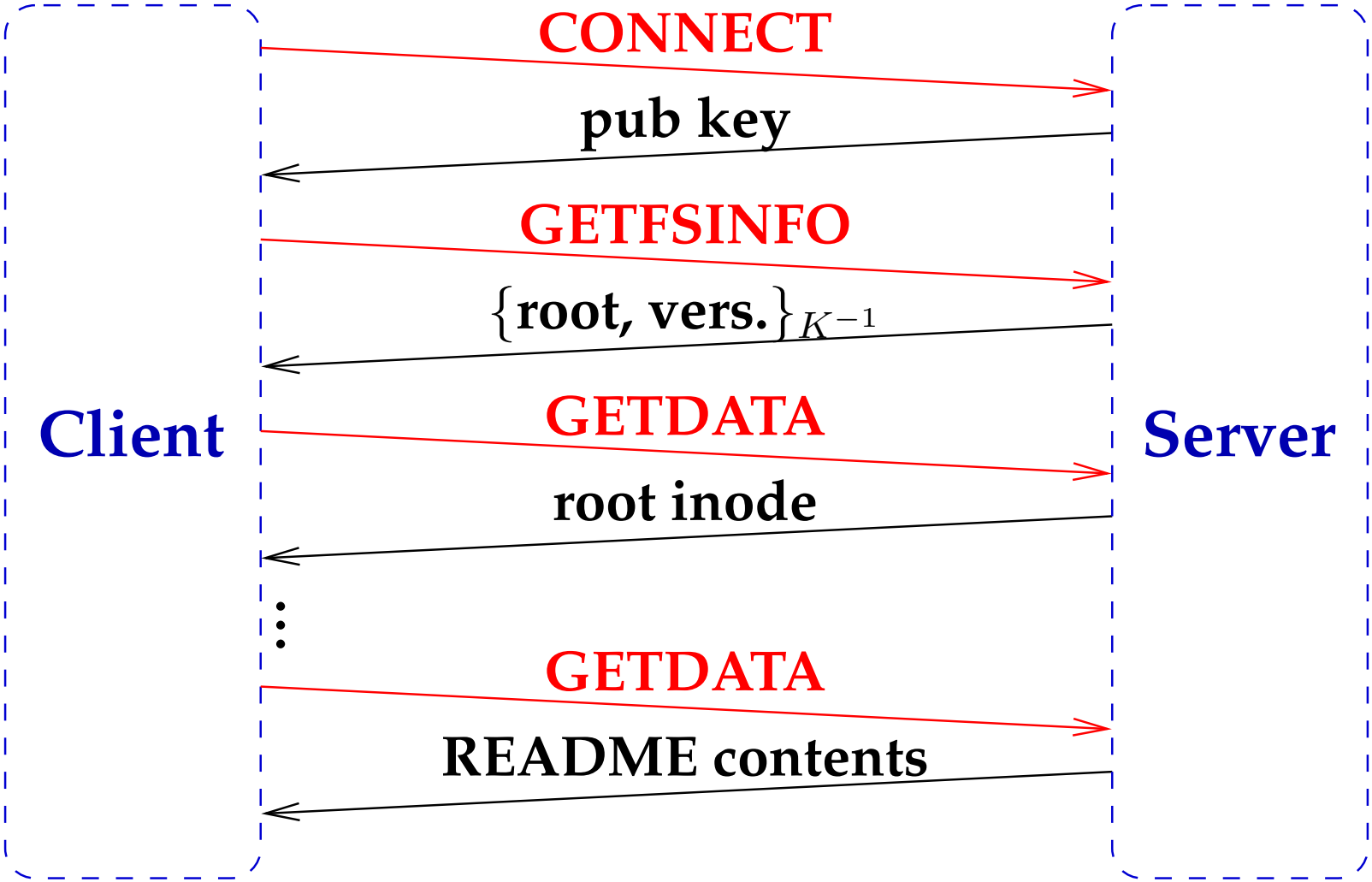

- **Index all data & metadata with cryptographic hash**

# The SFSRO protocol

- **CONNECT () – Initiate SFSRO protocol**
- **GETFSINFO () – Get signed hash of root directory**
- **GETDATA (*hash*) – Get block with *hash* value**
- **All data interpreted entirely by client**
  - Server need know nothing about file system structure
  - Makes server fast and simple ($< 400$ lines of code)

# Example: File Read

`/sfs/sfs.nyu.edu:bzcc5hder7cuc86kf6qswyx6yuemnw69/README`



# Incremental replication

- **Replicas need transfer only modified data**
- ***pulldb* utility incrementally updates a replica**
  - Uses SFSRO protocol to traverse file system
  - Stores all hashes/values encountered in new database
  - Avoids transferring any hashes already in old database
  - Unchanged directories automatically pruned from transfer
- **Makes short signature durations practical**

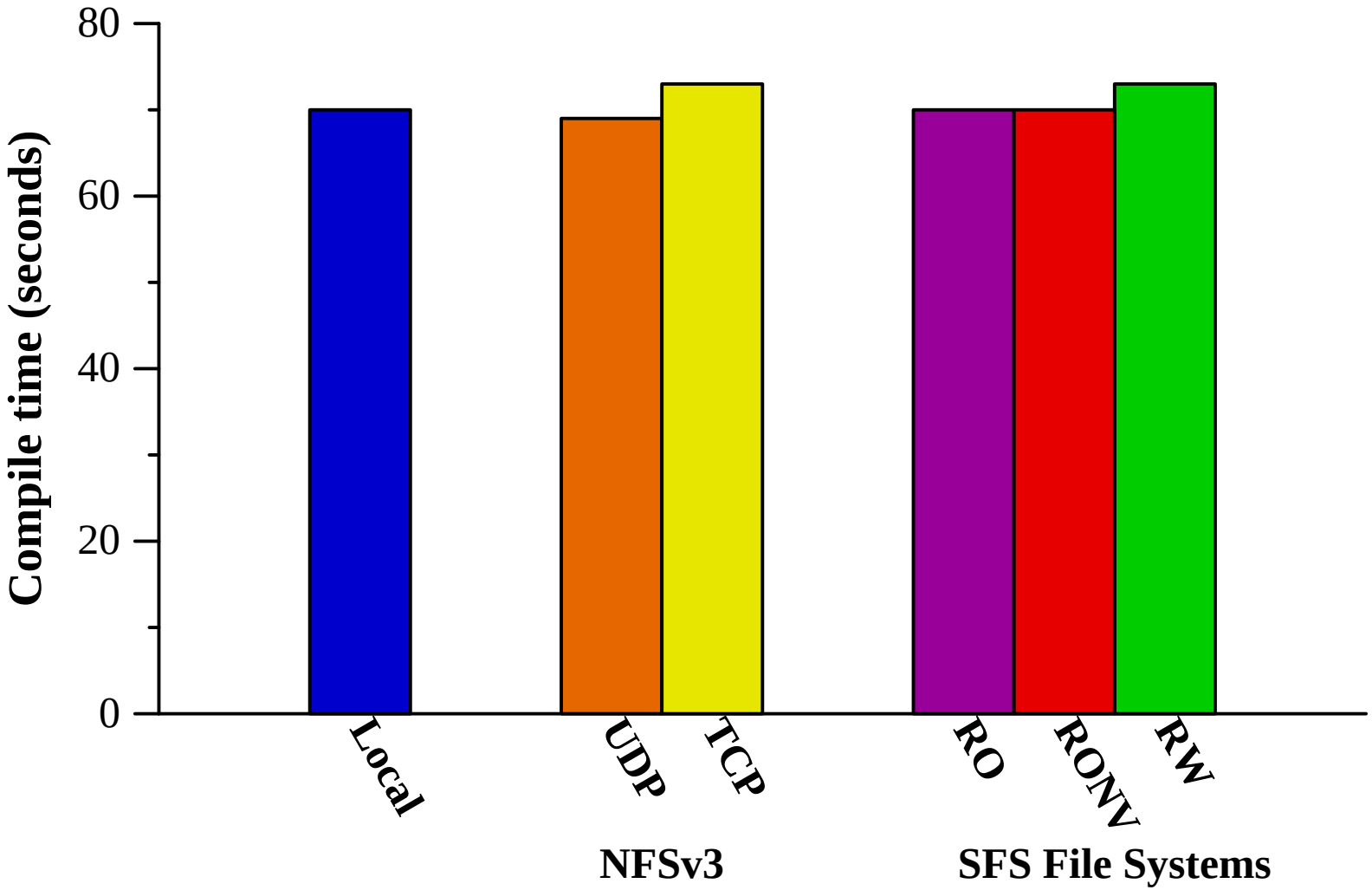
# Application: RedHat distribution

- **Publish `ftp.redhat.com` via SFSRO**
  - Push out new signature every 24 hours
- **Advantages:**
  - Volunteer mirror sites need no longer be trusted
  - Install from file system, not URL (easier to browse)
  - Secure automatic upgrade becomes a simple script
  - Can revoke/update flawed packages quickly
  - File names securely bound to contents
  - Easy for users to understand security properties

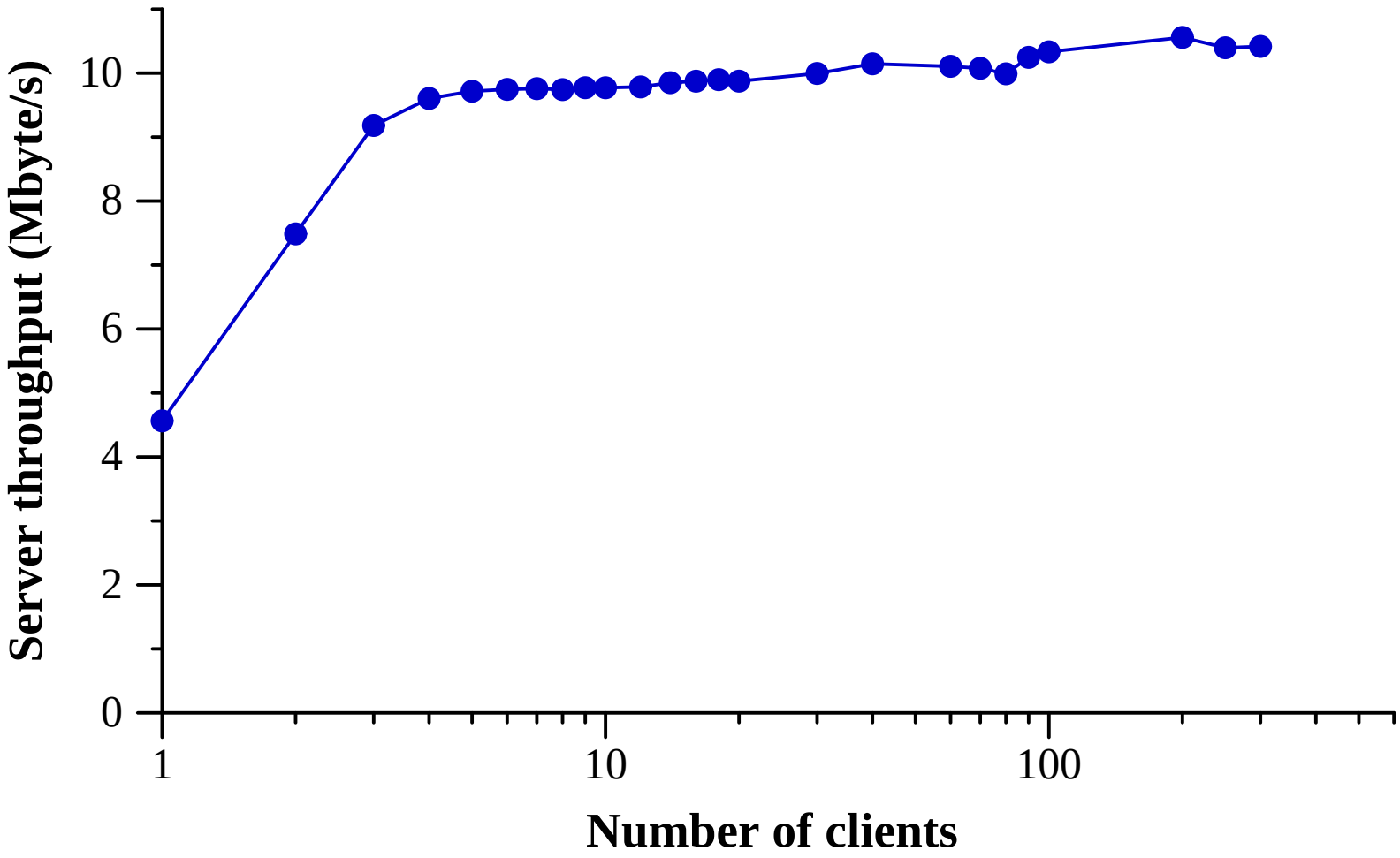
# Application: Software distribution

- **Distribute open-source software via SFSRO**
  - Users can compile directly from the distribution
- **Benchmark: Compile emacs-20.6 from source code**
  - 550 MHz Pentium IIIs, 256 MBytes RAM, FreeBSD 3.3
  - Warm server cache, cold client cache

# Performance: Emacs compile



# Scalability: Emacs compile



# Application: Certificate authorities

- **SFS specifies public keys of servers in file names:**

  `/sfs/sfs.nyu.edu:bzcc5hder7cuc86kf6qswyx6yuemnw69`

- **Symbolic links hide public keys from users:**

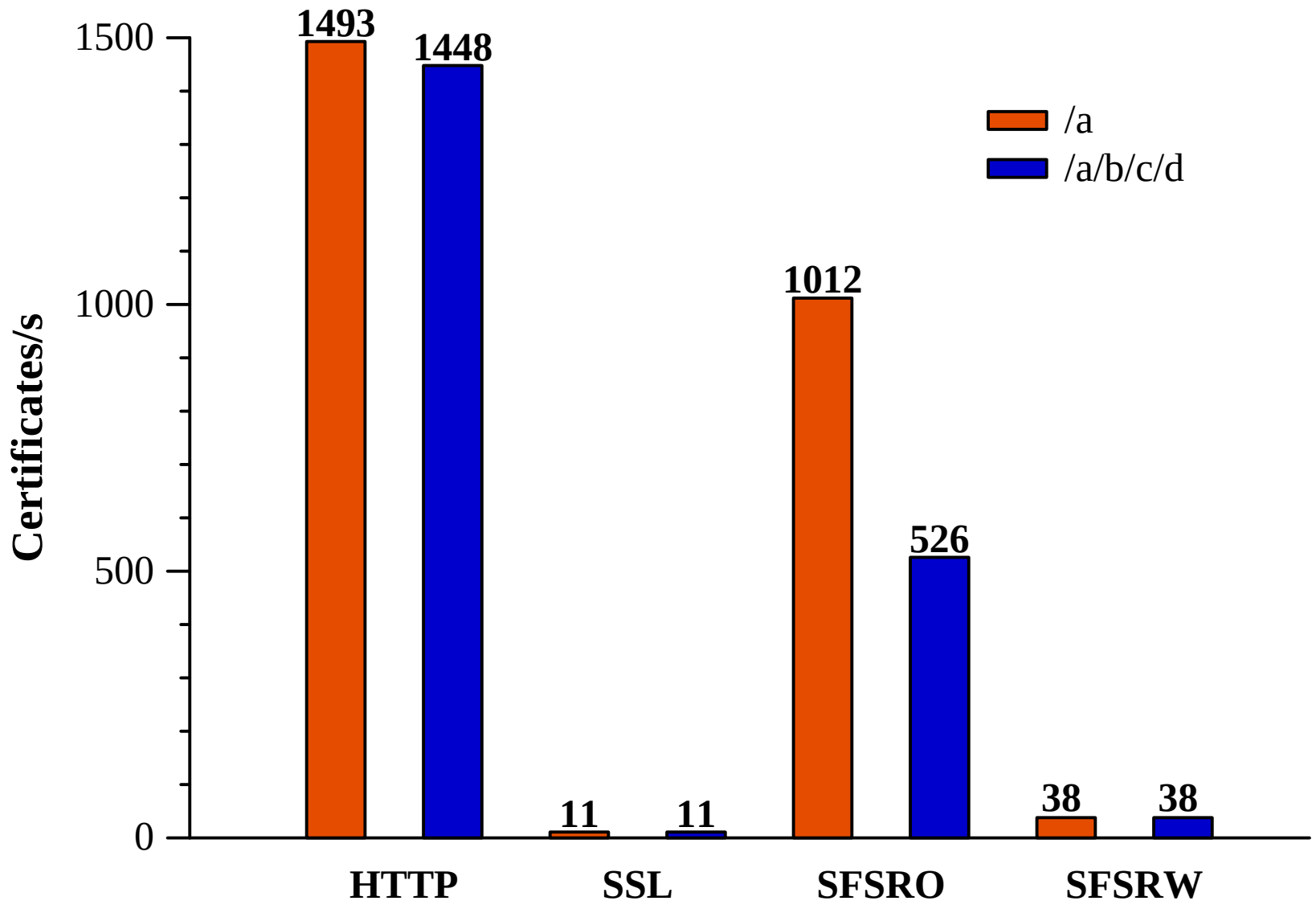
  `/verisign` $\rightarrow$ `/sfs/sfs.verisign.com:r6ui9gw···pfbz4pe`

- **SFSRO can serve name-to-key bindings:**

  `/verisign/nyu` $\rightarrow$ `/sfs/sfs.nyu.edu:bzcc5hd···uemnw69`

- **Better revocation than traditional CAs**
  - Signature can realistically expire in hours, not months
  - Cannot revert one certificate without reverting them all

# Scalability: Certificate downloads



# Conclusions

- **Public read-only data needs integrity guarantees.**
- **Cannot realistically sacrifice performance, scalability, or convenience to get those guarantees.**
- **SFSRO achieves integrity without sacrifice**
  - Off-line publishing has cost independent of server load
  - Dirt-simple server offloads cryptographic costs to clients
  - File system is the most convenient/universal interface