SiRiUS: Securing Remote Untrusted Storage

NDSS 2003

Eu-Jin Goh, Hovav Shacham,
Nagendra Modadugu,
and Dan Boneh
Stanford University

Introduction

Secure network file systems not widespread. Why?

1. Hard to deploy
   - No backwards compatibility
2. File sharing not well supported
   - No file sharing ability, or
   - Fully trusted server handles file sharing

Insecure Network File Systems

Legacy network file systems

- widely used: NFS, CIFS, Yahoo!
- insecure: NFS v2
  - Weak authentication: UID/GID
  - Fully trusted server

SiRiUS Goals

1. *<u>No changes</u>* to remote file server
   - Implies crypto techniques
2. Easy for end users to deploy
   - Minimal client software, no kernel changes
3. File Sharing with fine grained access control
   - Read-write separation
4. Minimize trust in file server

Existing Secure File Systems

1. CFS – Blaze
    - Single user: no file sharing
2. SFS – Mazières et al.
    - File sharing but uses trusted server
3. SUNDR – Mazières et al.
    - File sharing by untrusted server
    - Not easy to deploy: requires block servers

⋮

"Although NFS version 2 has been superseded in recent years by NFS version 3, system administrators are slow to upgrade ... so NFS version 2 is not only widespread, it is still by far the most popular version of NFS."

NFS v3 Designers

4½ years after NFS v3 introduced

Design Limitations

Cannot defend against DOS attacks:

- attacker breaking into file server can delete all files
- Solution:
  1. Keep good backups: SiRiUS easy to backup
  2. Replicate files using quorum systems
     - e.g. Reiter-Mahlki (1997)

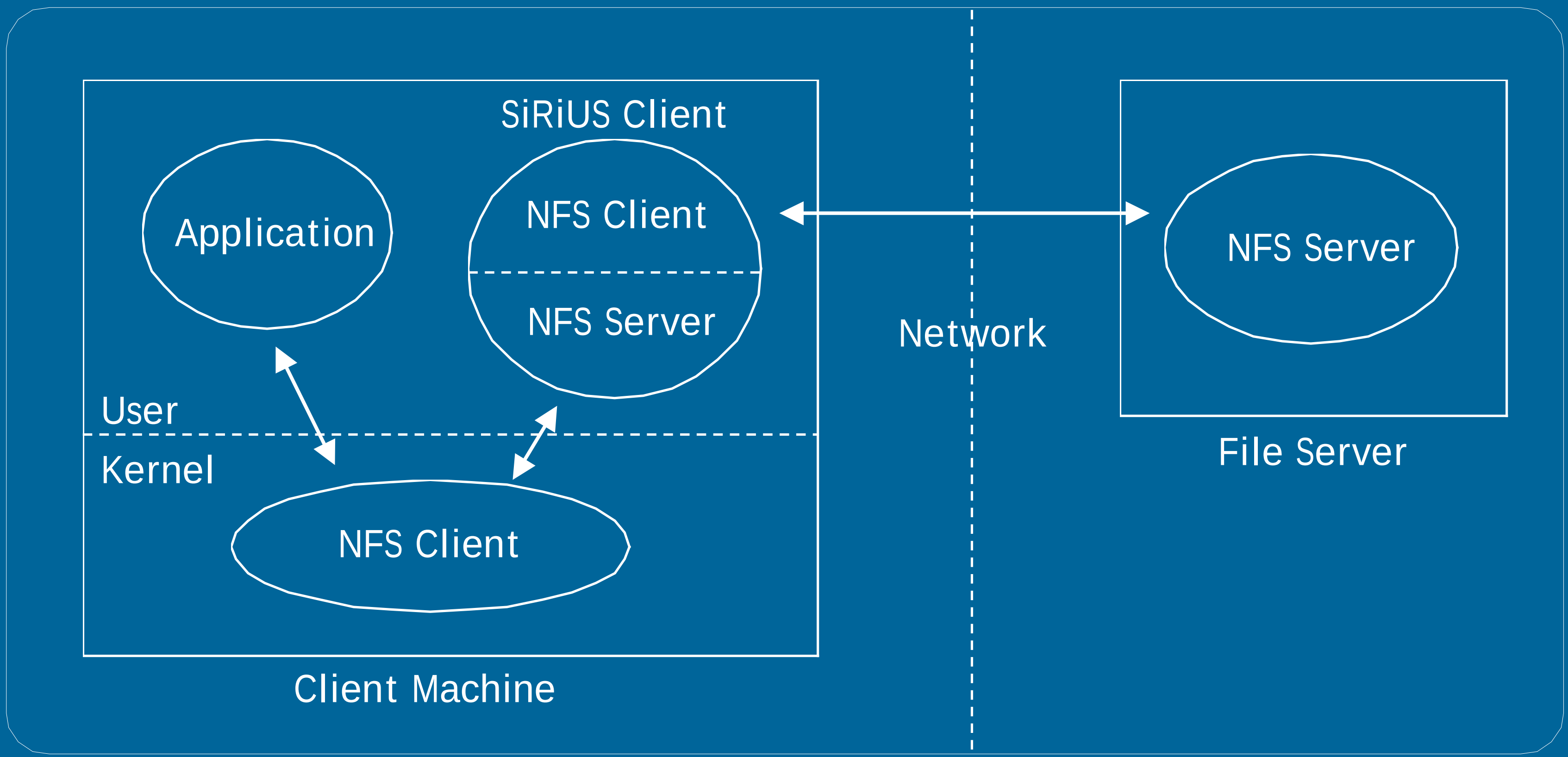
SiRiUS Usage Model

- SiRiUS is a file system layered over existing network file systems
- Stop-gap measure until full upgrade of legacy systems

Security Design

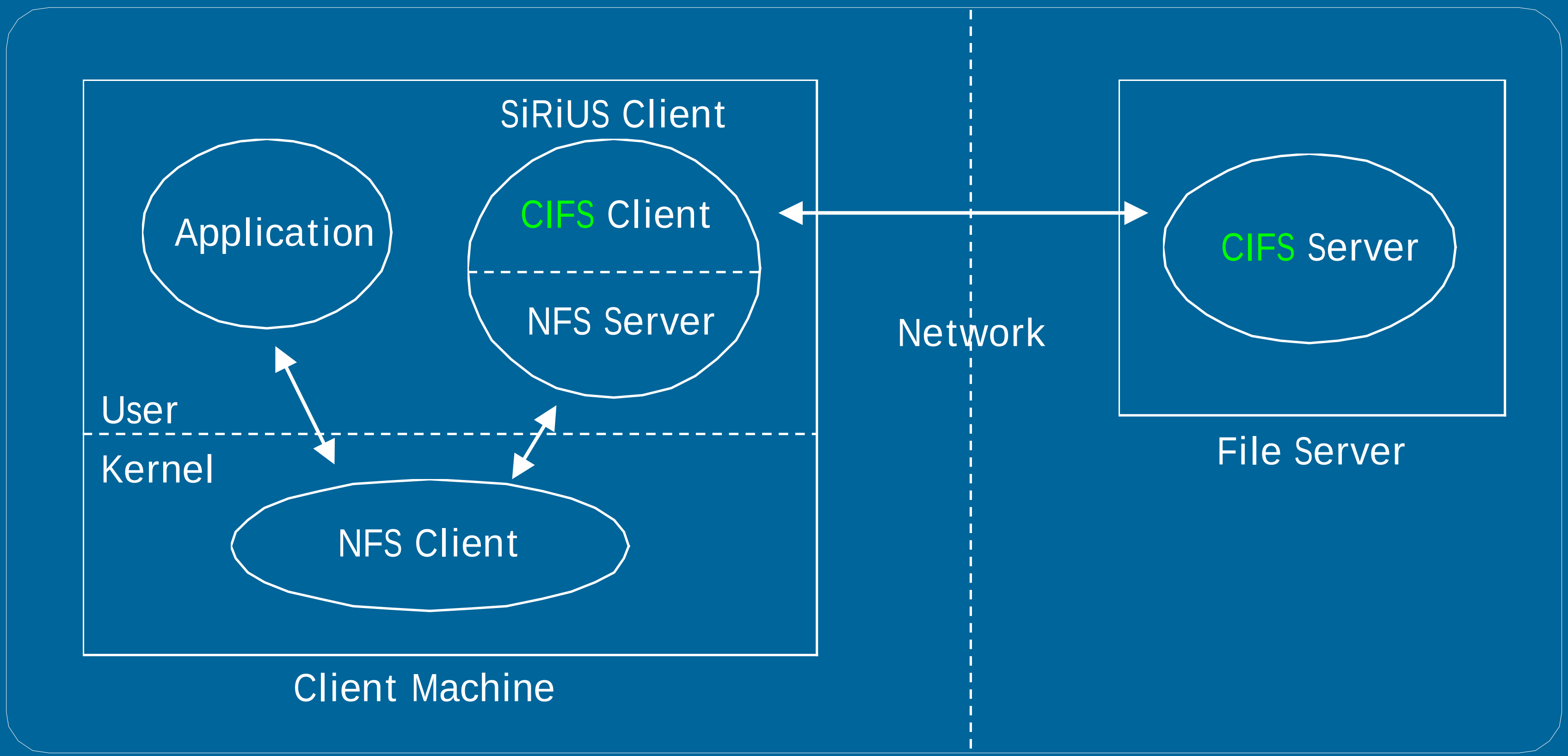
1. Confidentiality and integrity
2. Cryptographic file level read-write access controls
3. Simple key management
4. Simple access control revocation
5. Freshness guarantees for access control meta data

Architecture



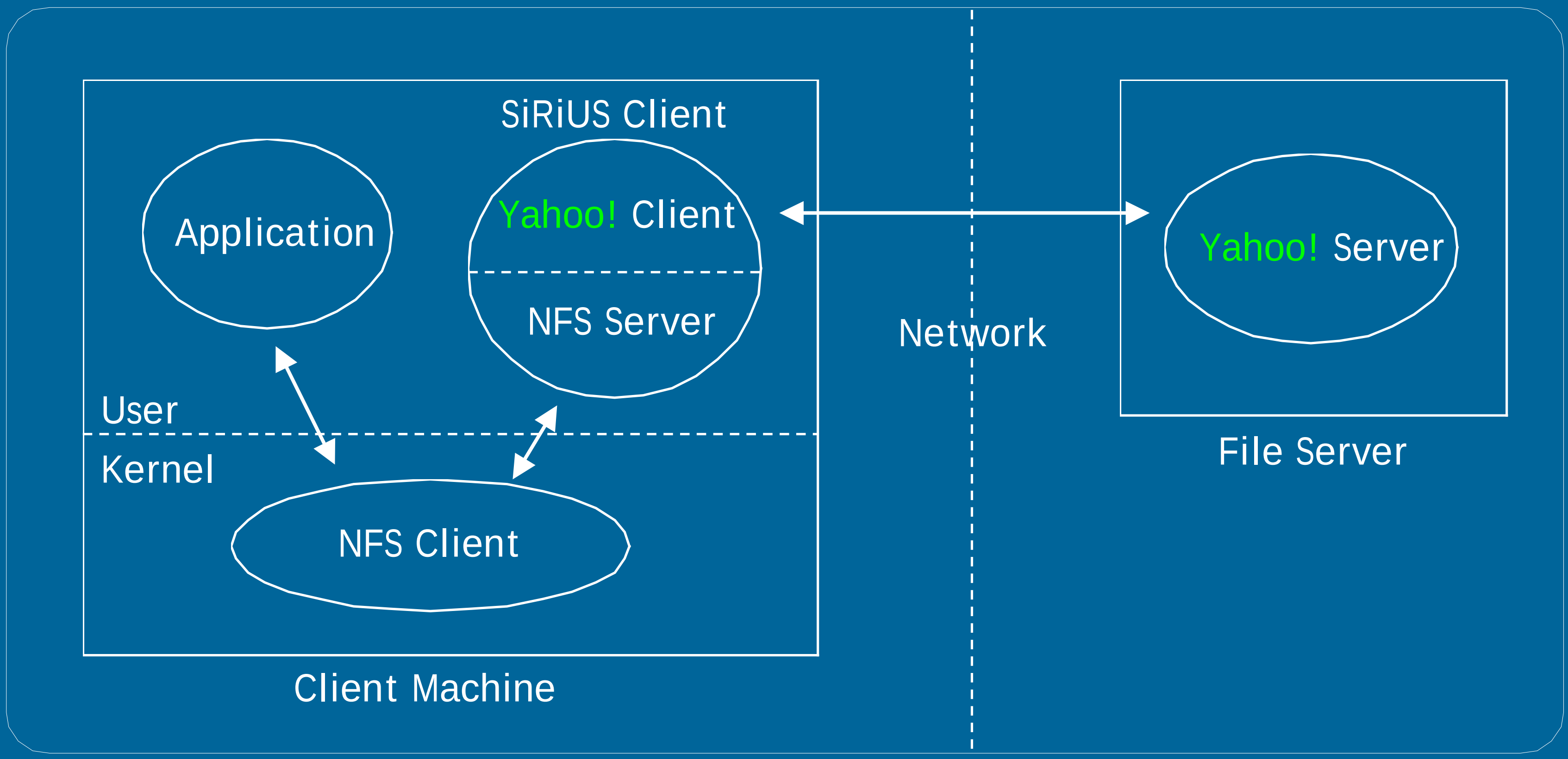
SiRiUS layered over NFS

Architecture



SiRiUS layered over CIFS

Architecture



SiRiUS layered over Yahoo!

File Data Security

- Each file has *unique*:
  1. File Encryption Key (FEK)
  2. File Signing Key (FSK)
- FEK, FSK control file read-write access
- Users keep only *2 keys for all files*:
  1. Master Signing Key (MSK)
  2. Master Encryption Key (MEK)
- MSK, MEK control all file FEK and FSK access

File Structures

Files on remote server split in 2 parts

1. md-file contains the file meta data. e.g. access control information
2. d-file contains the file data

File Structures

ENC_{FEK}[File Data]	SIG_{FSK} [File Data Hash]

d-file

Enc. Key Block (Owner)	Enc. Key Block (User 1)	File Sig. Pub. Key (FSK)	Time Stamp	File name	SIG_{MSK} [Meta Data Hash]

md-file

Encrypted Key Blocks

Read-write
Username (KeyID)
File Enc. Key (FEK)
File Sig. Private Key (FSK)

Encrypted with username’s MEK public key

Read only
Username (KeyID)
File Enc. Key (FEK)

Meta Data File Creation

Meta Data File Creation

1) Generate file keys (FSK and FEK)

File Enc. Key (FEK)
File Sig. Private Key (FSK)

Meta Data File Creation

Username (KeyID)

File Enc. Key (FEK)

File Sig. Private Key (FSK)

1) Generate file keys (FSK and FEK)

2) Create encrypted key block.

Encrypted with owner's MEK public key

Meta Data File Creation

Enc. Key Block (Owner)	File Sig. Pub. Key (FSK)	Time Stamp	File name

3) Append Pub FSK, time stamp, and file name to enc. key block

Meta Data File Creation

Enc. Key Block (Owner)	File Sig. Pub. Key (FSK)	Time Stamp	File name	SIG_{MSK} [Meta Data Hash]

3) Append Pub FSK, time stamp, and file name to enc. key block

4) Hash and sign using owner's master signing key

Meta Data File Creation

Enc. Key Block (Owner)	File Sig. Pub. Key (FSK)	Time Stamp	File name	SIG_{MSK} [Meta Data Hash]

3) Append Pub FSK, time stamp, and file name to enc. key block

4) Hash and sign using owner's master signing key

5) Update md-file freshness tree

Why are freshness guarantees needed?

- Can verify latest version of info is read
- md-file freshness prevents rollback of revoked privileges

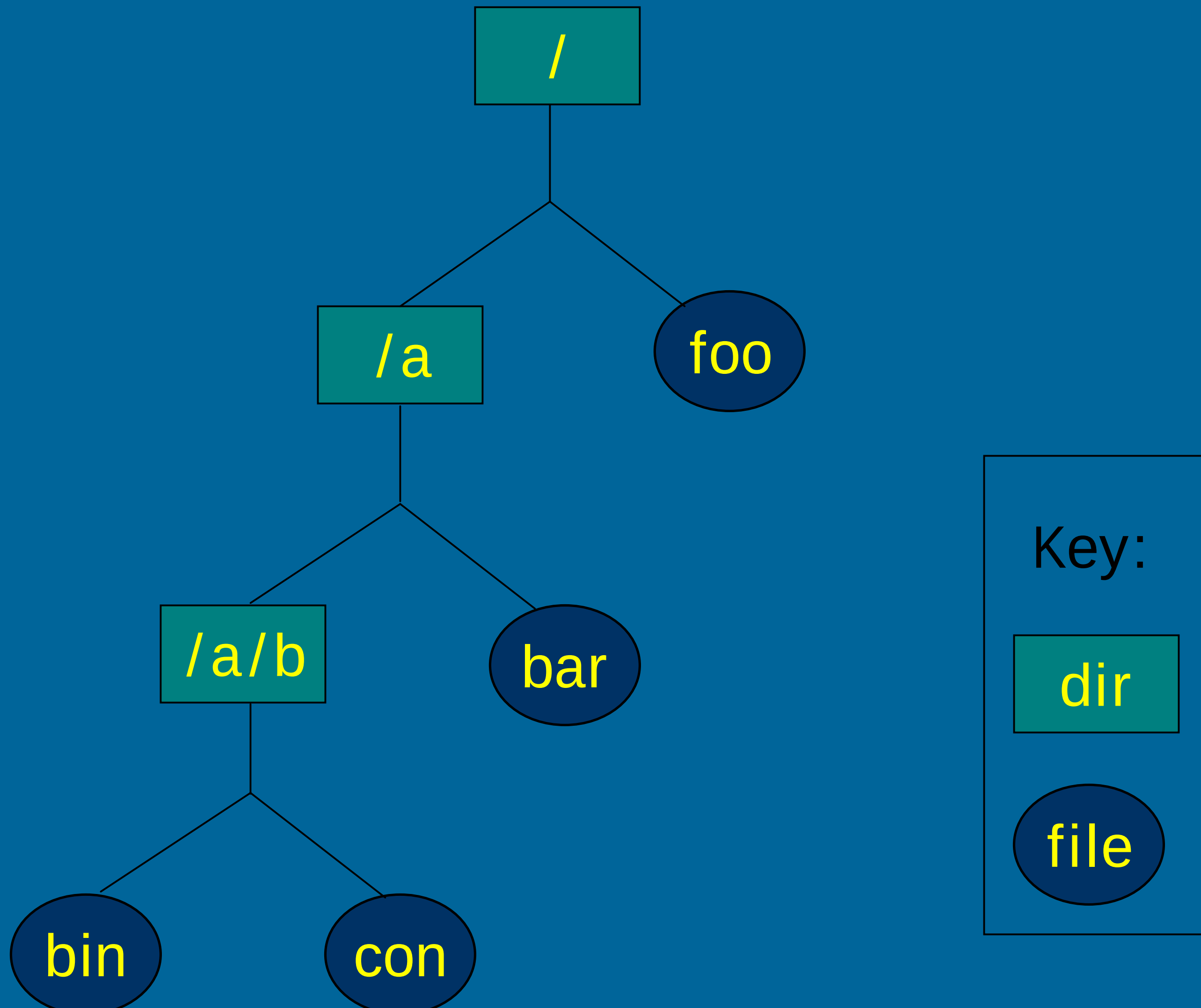
Rollback Revoked Privileges

1. Bob revokes write access from Alice
2. Alice replaces new md-file with saved (older) copy
3. Replacement restores write privileges
4. Alice can undetectably write to d-file

Freshness Overview

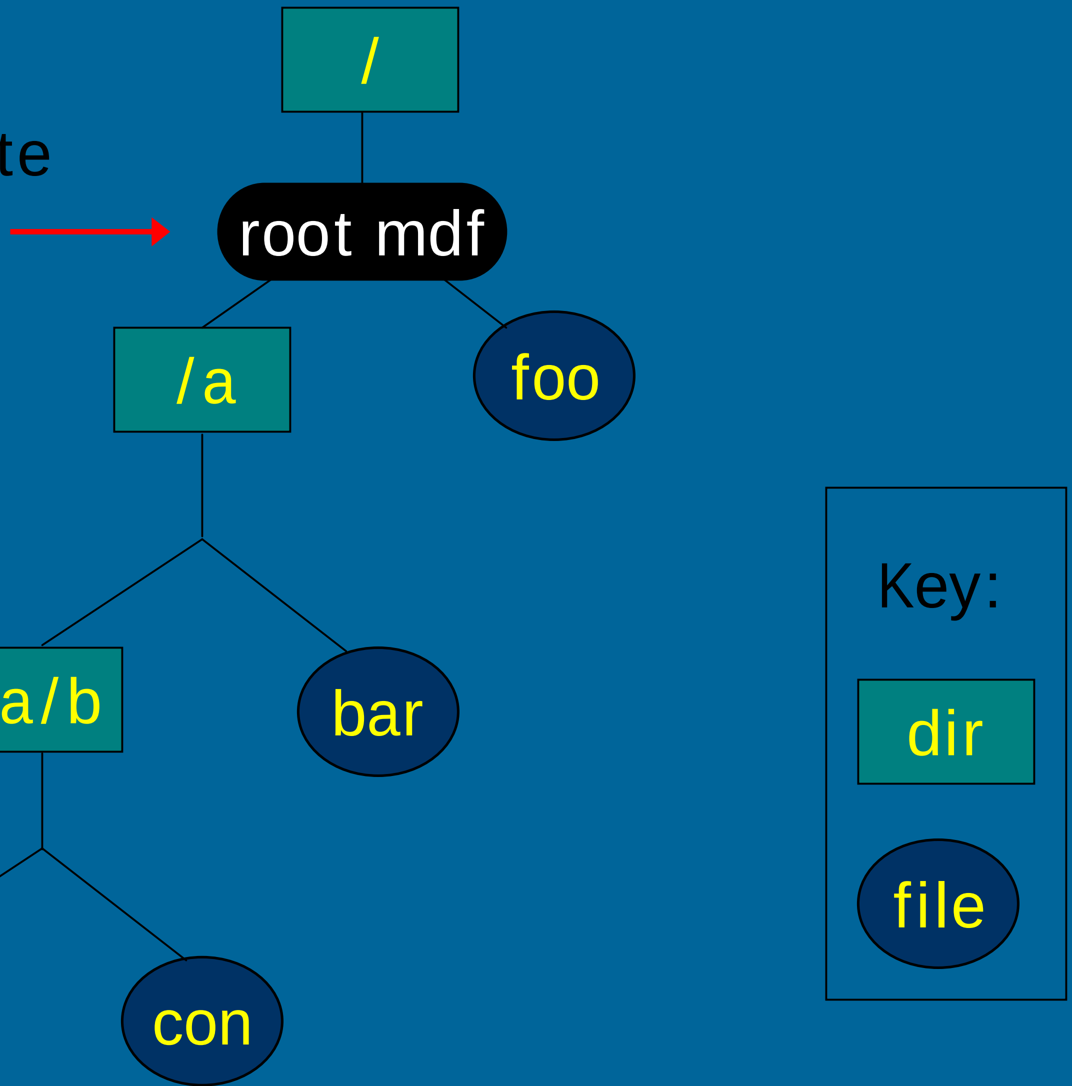
- SiRiUS client generates hash tree of all md-files owned by user
- Hash tree root: hash of all the md-files
- Every directory has mdf-file made of the hash of:
  1. md-files in that directory
  2. mdf-files of sub directories

Hash Tree Generation



Hash Tree Generation

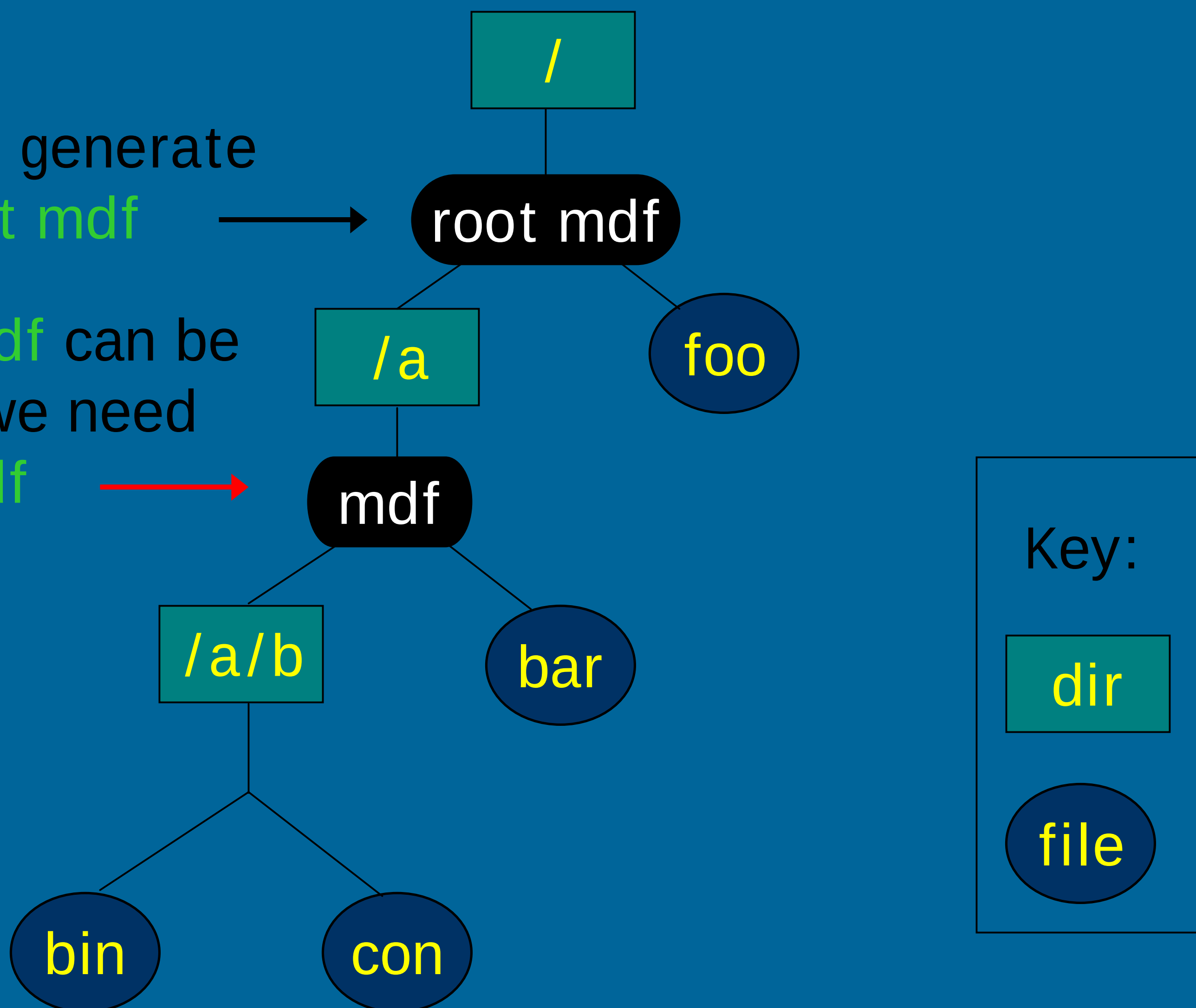
Want to generate root mdf



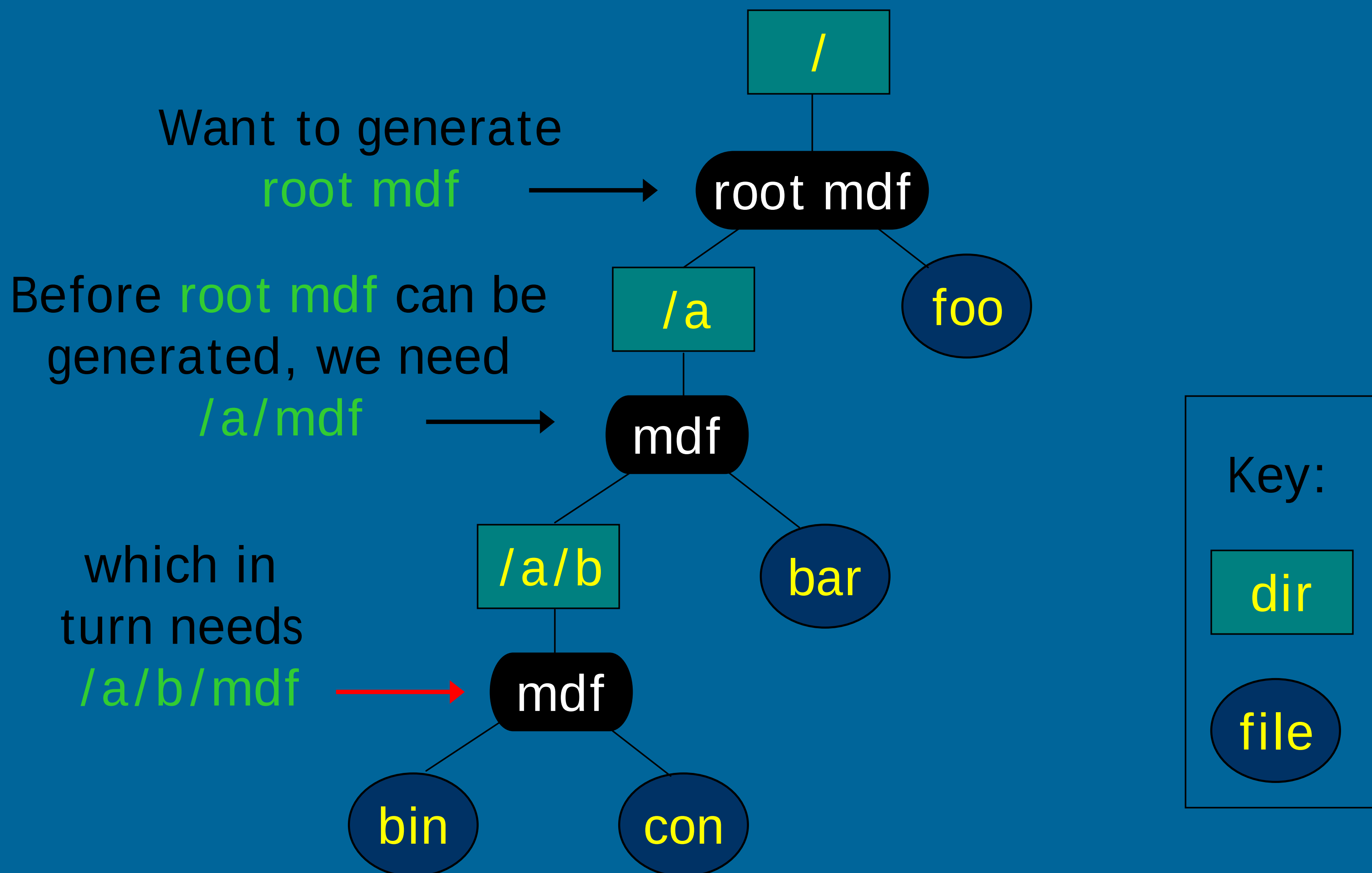
Hash Tree Generation

Want to generate
root mdf

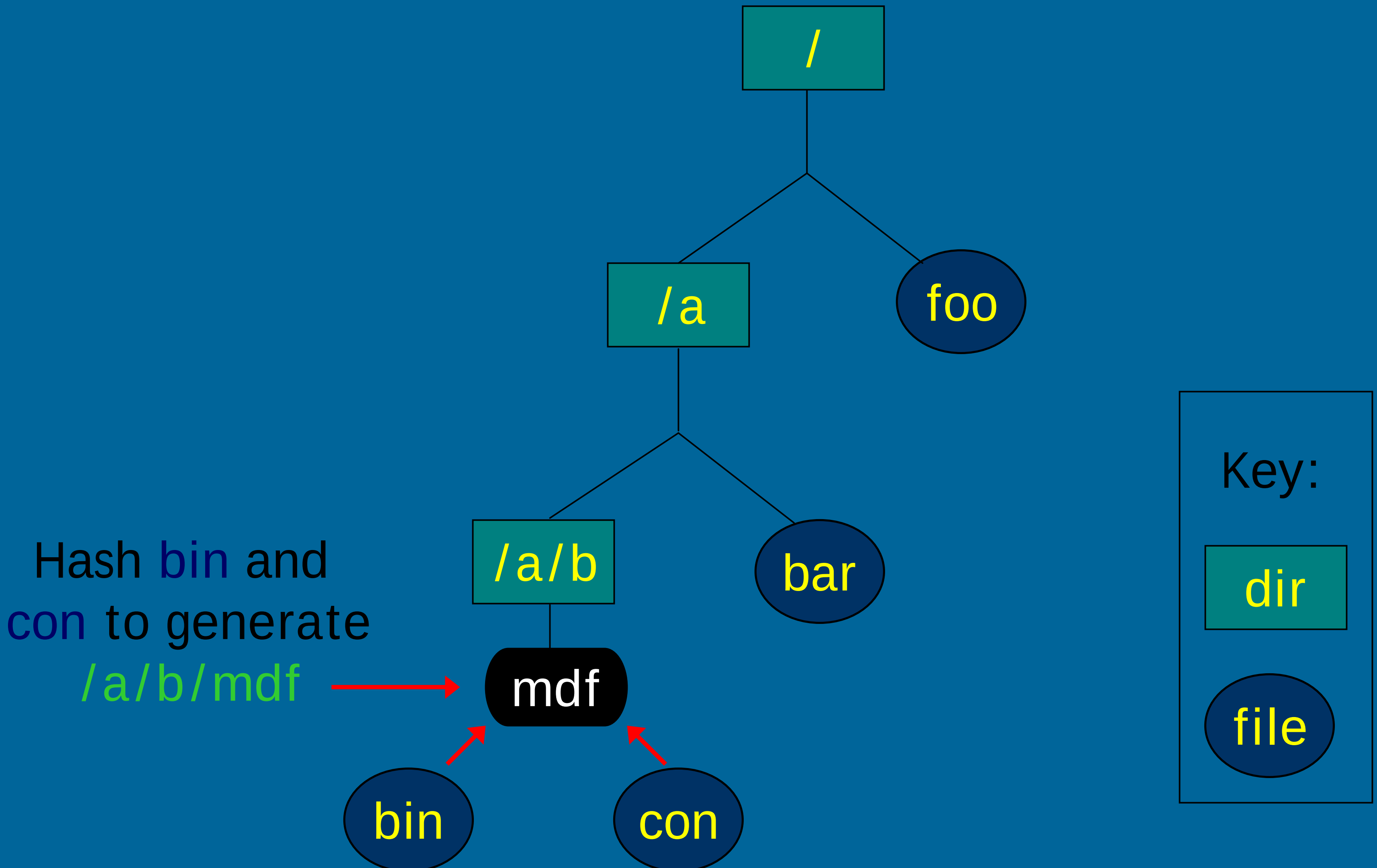
Before root mdf can be generated, we need /a/mdf



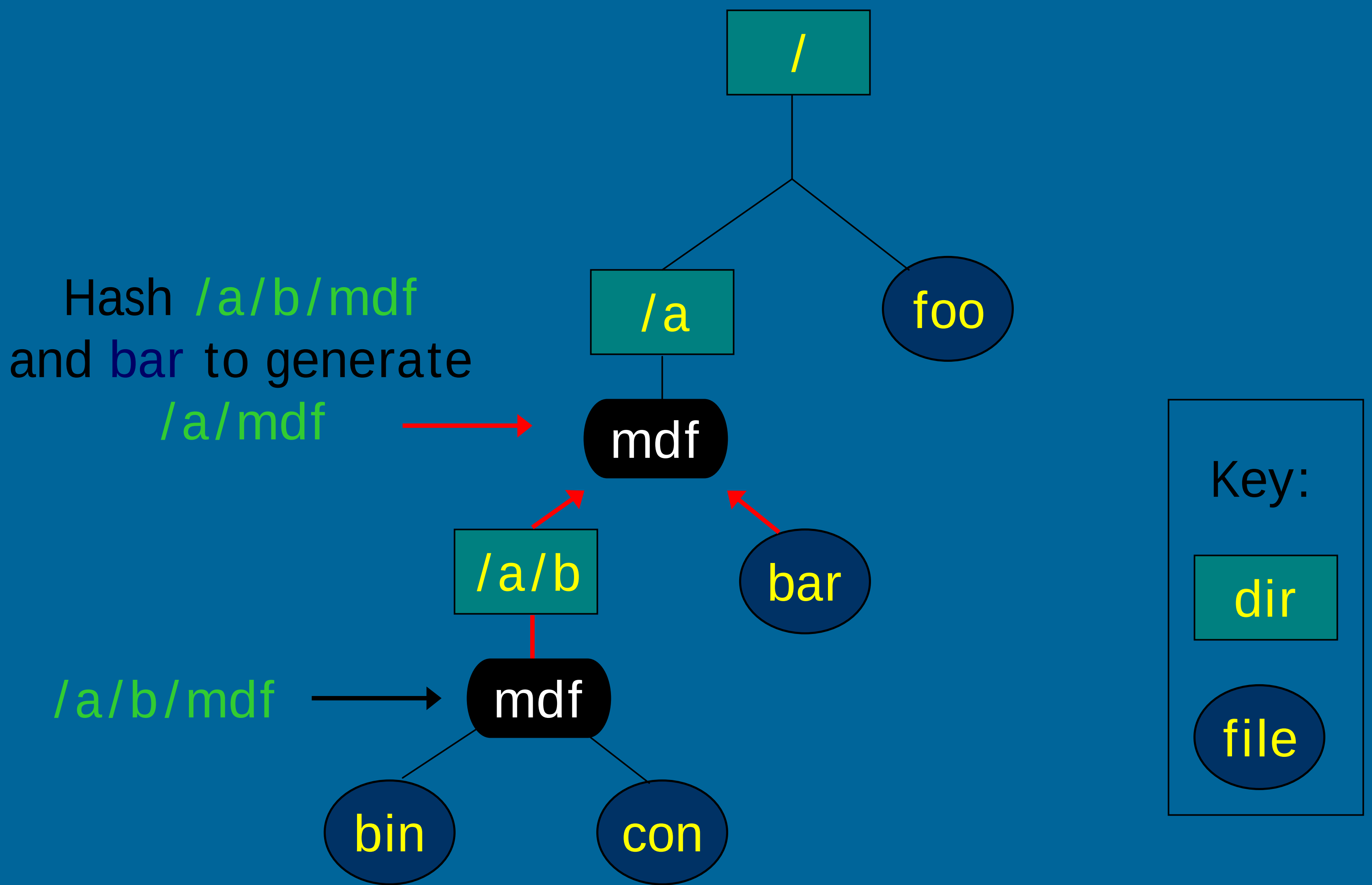
Hash Tree Generation



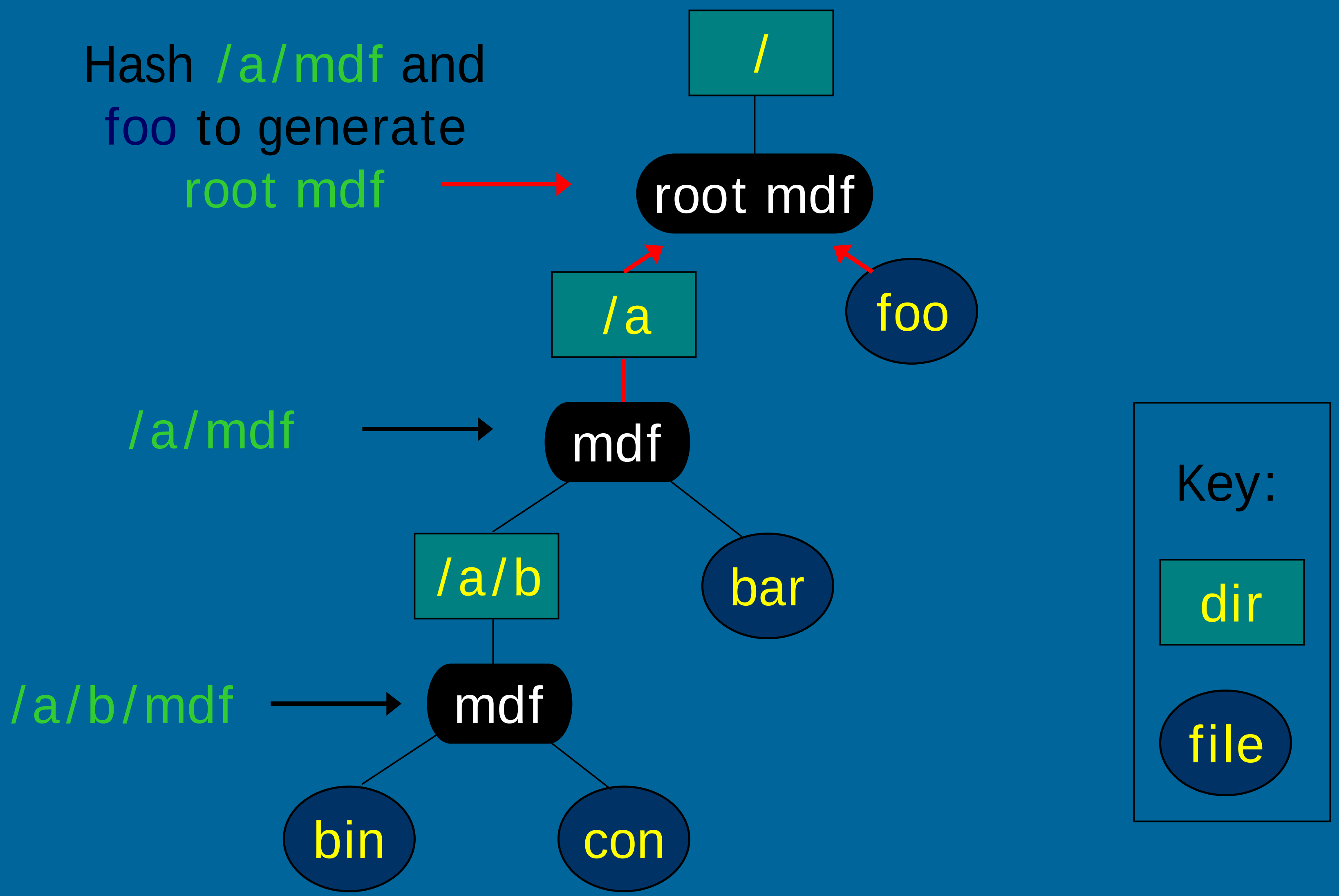
Hash Tree Generation



Hash Tree Generation



Hash Tree Generation



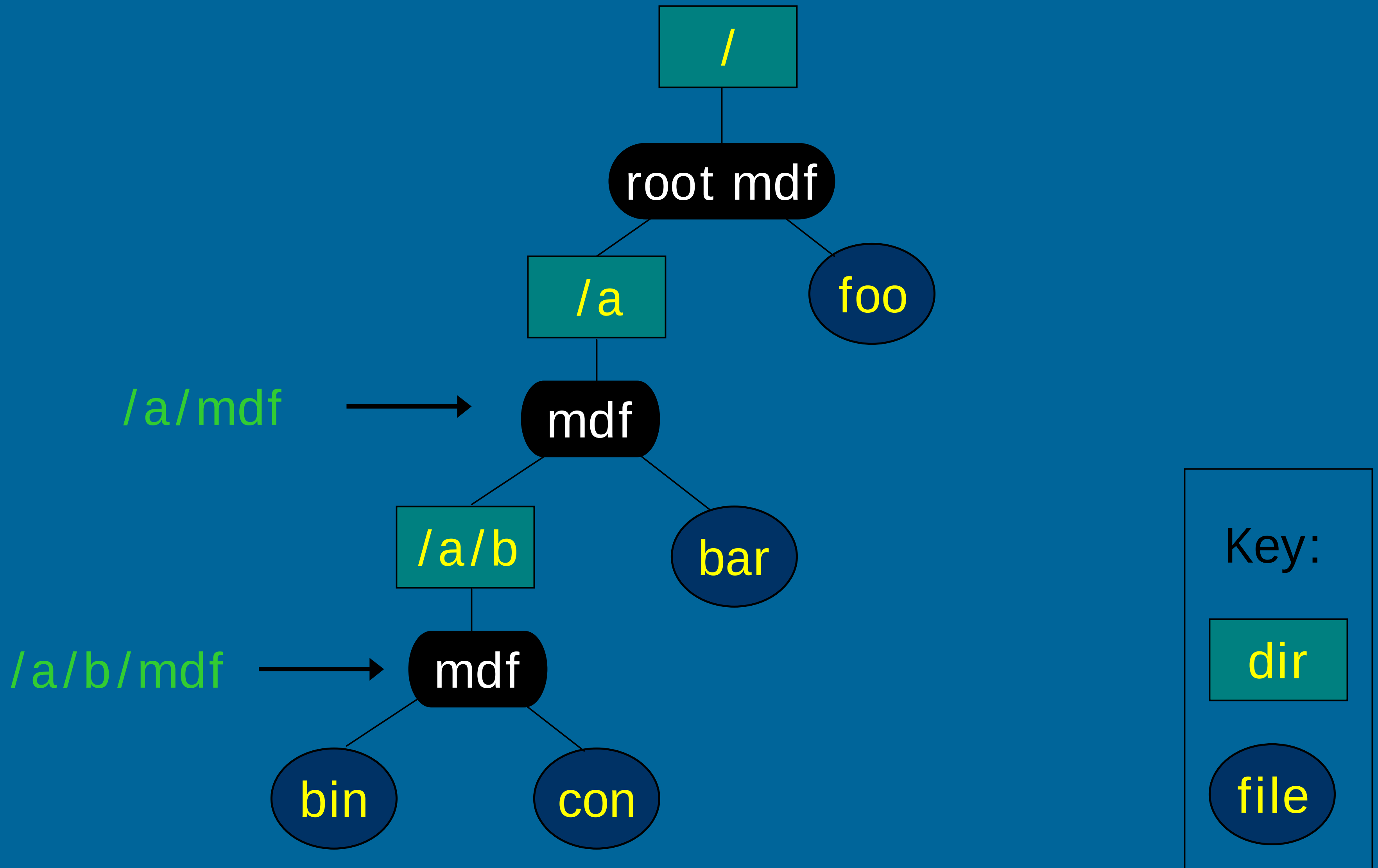
Root mdf-file

- Contains a time stamp
- Time stamp updated by client at specified time intervals
- Signed by owner of the md-files

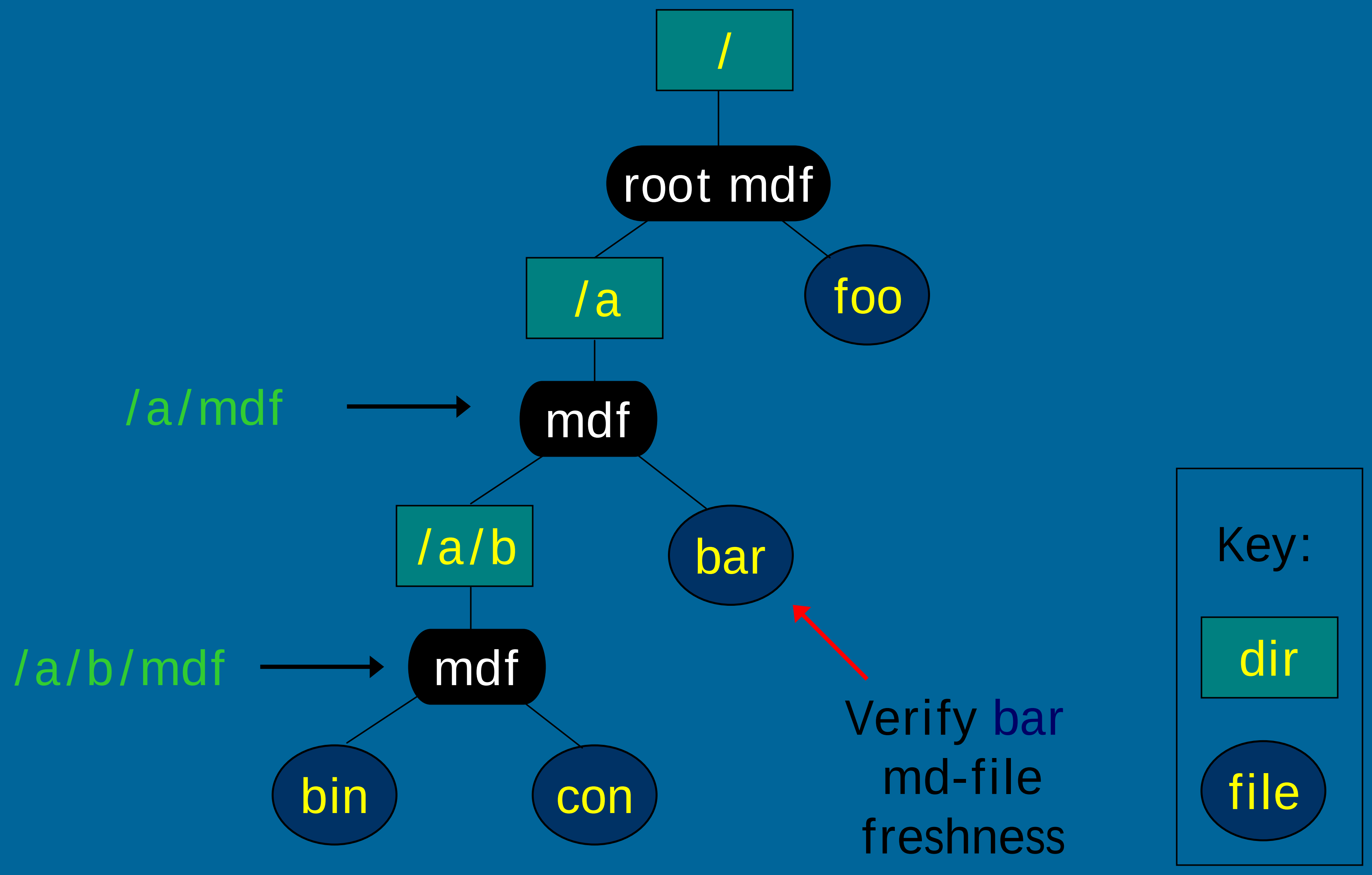
Hash Tree Generation

1. Generated only once
2. Generated by owner of md-files
3. Hash tree cacheable
4. Updated only on md-file changes

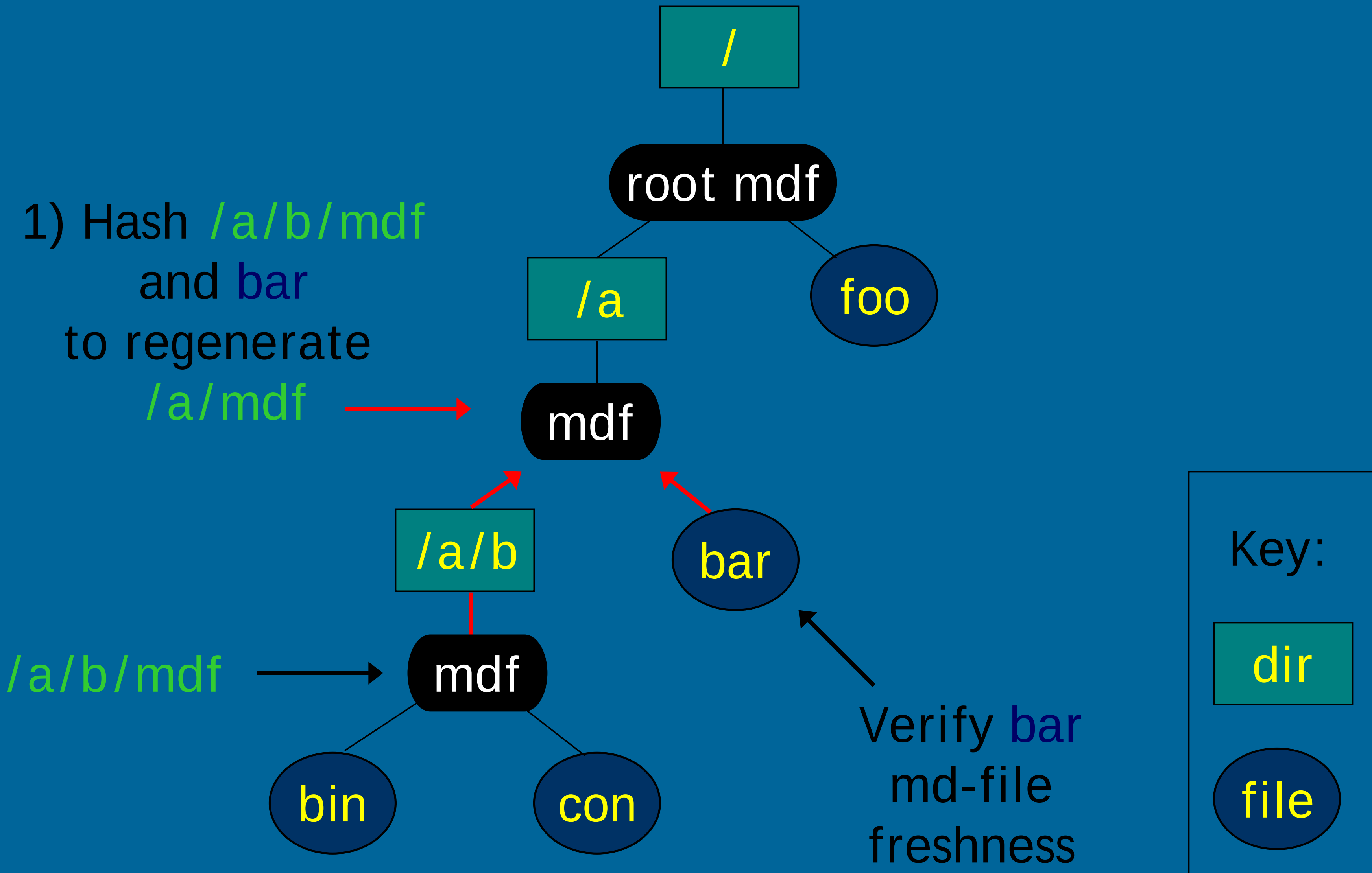
Verify md-file Freshness



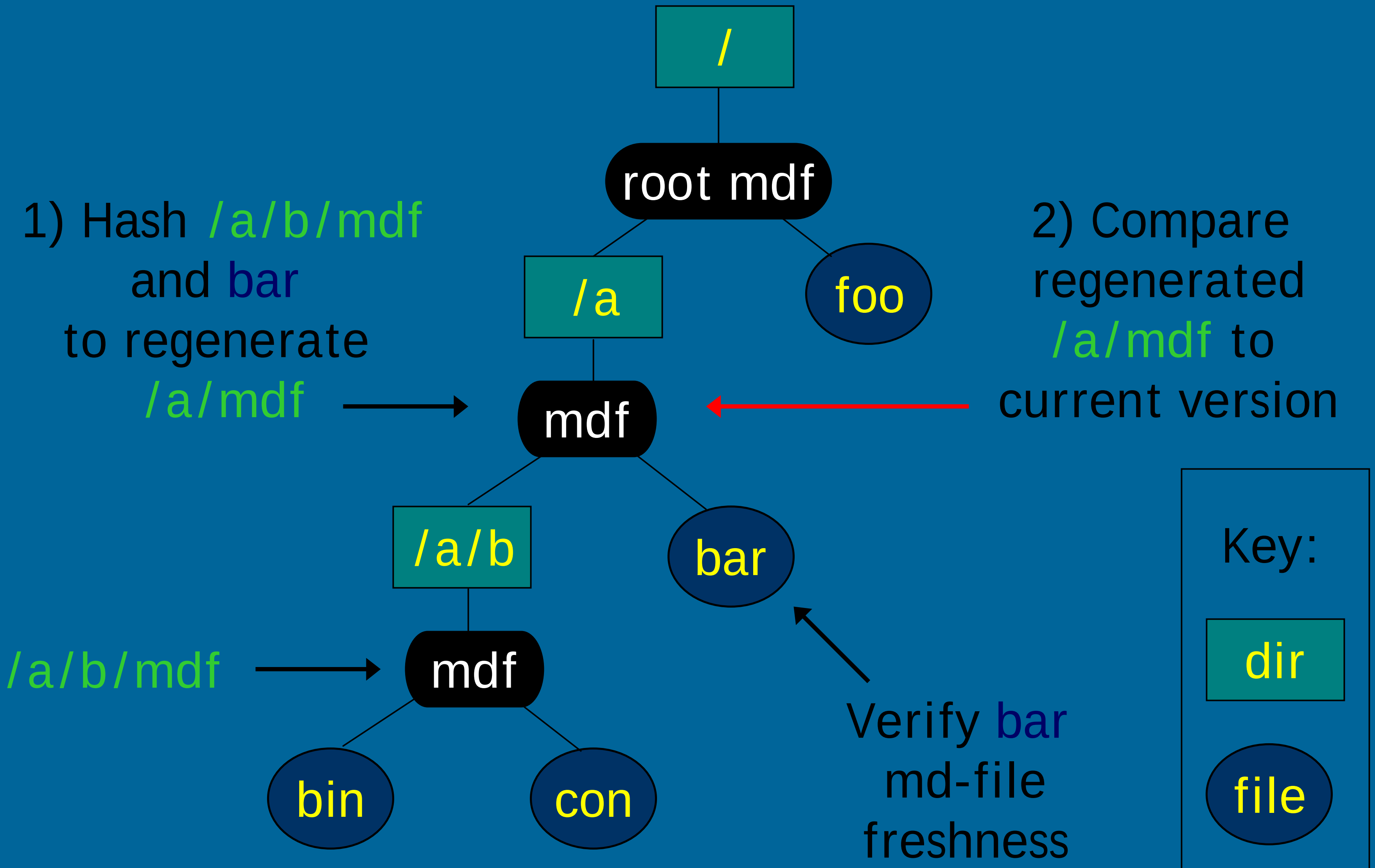
Verify md-file Freshness



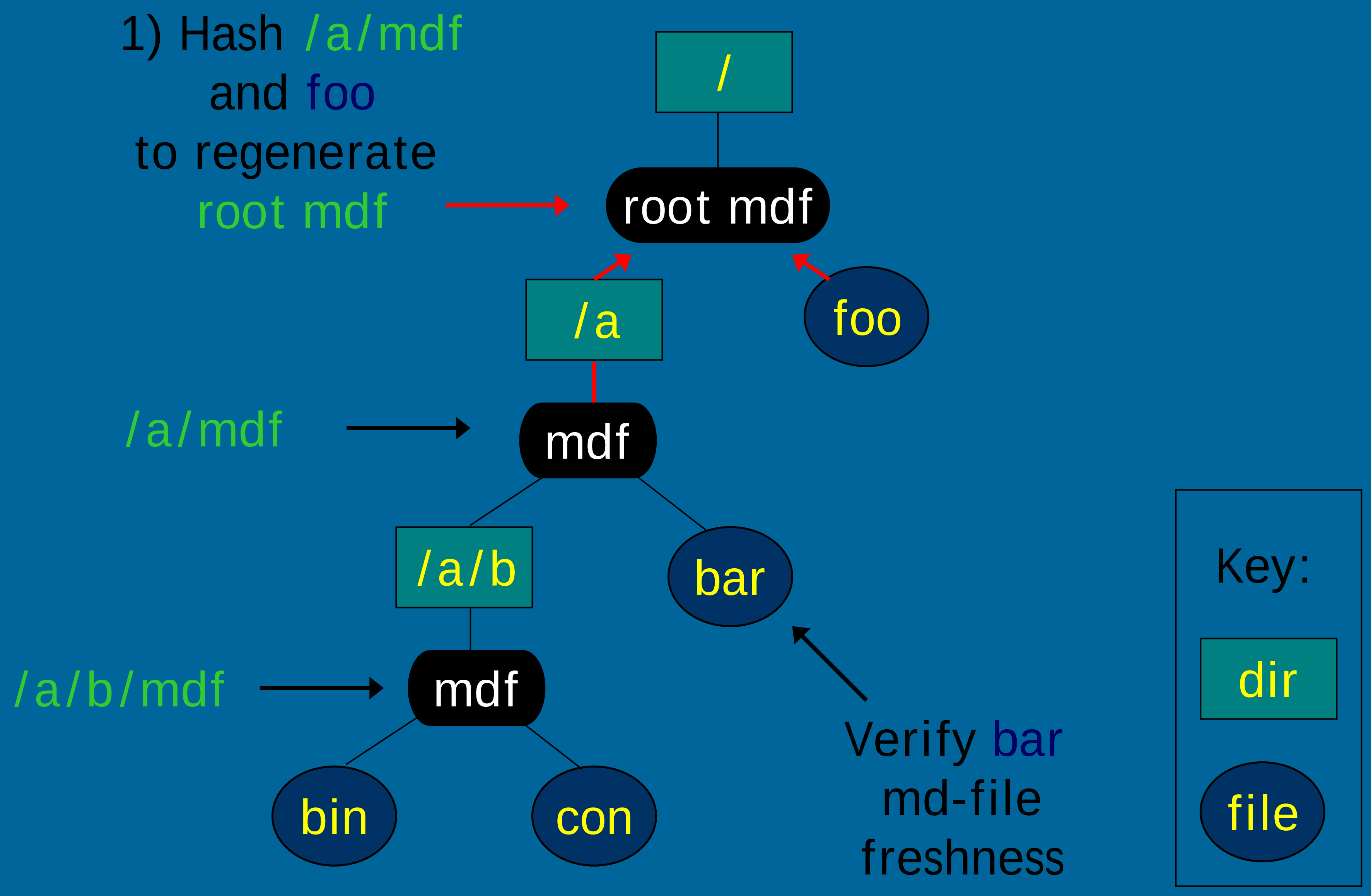
Verify md-file Freshness



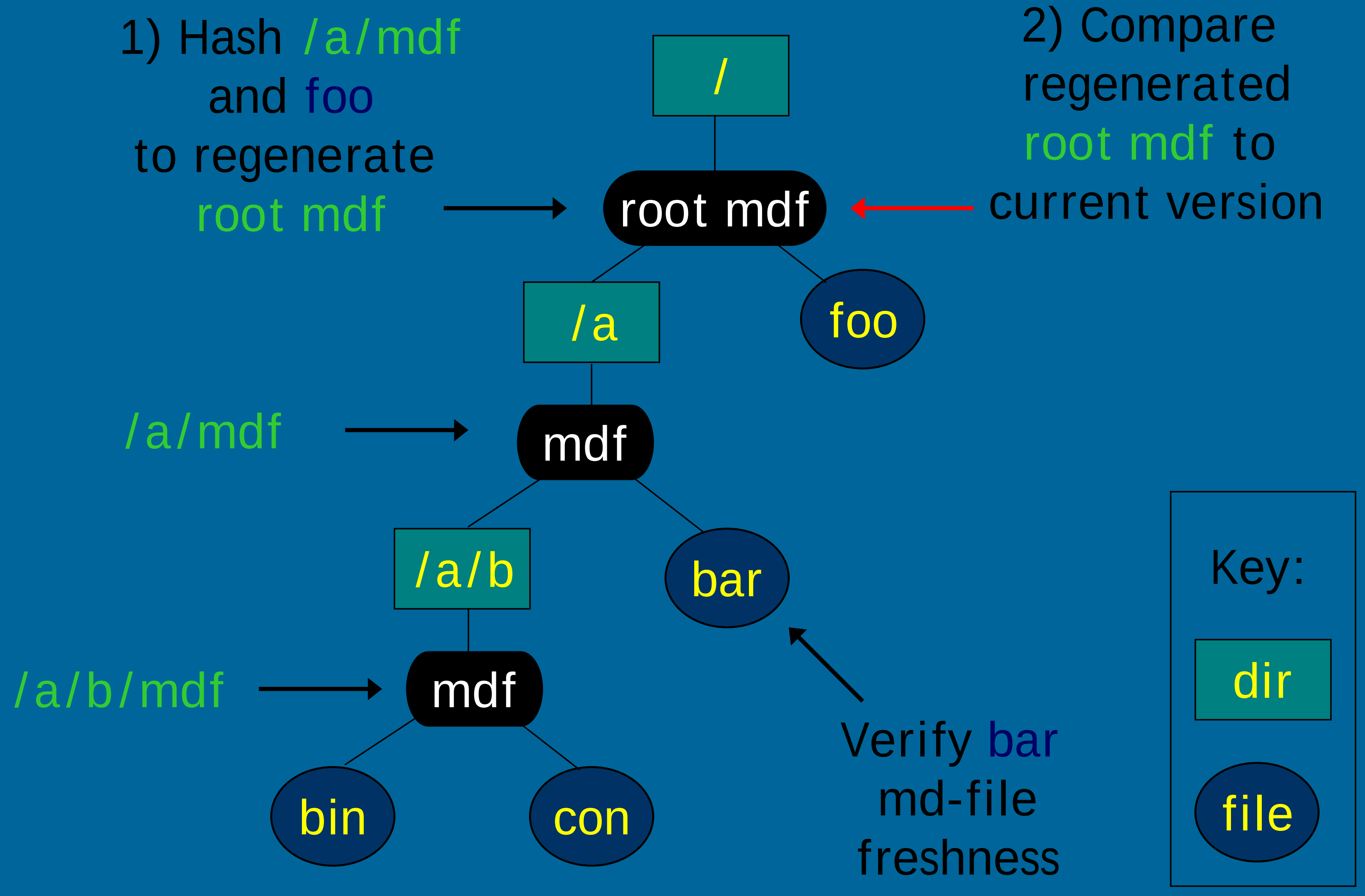
Verify md-file Freshness



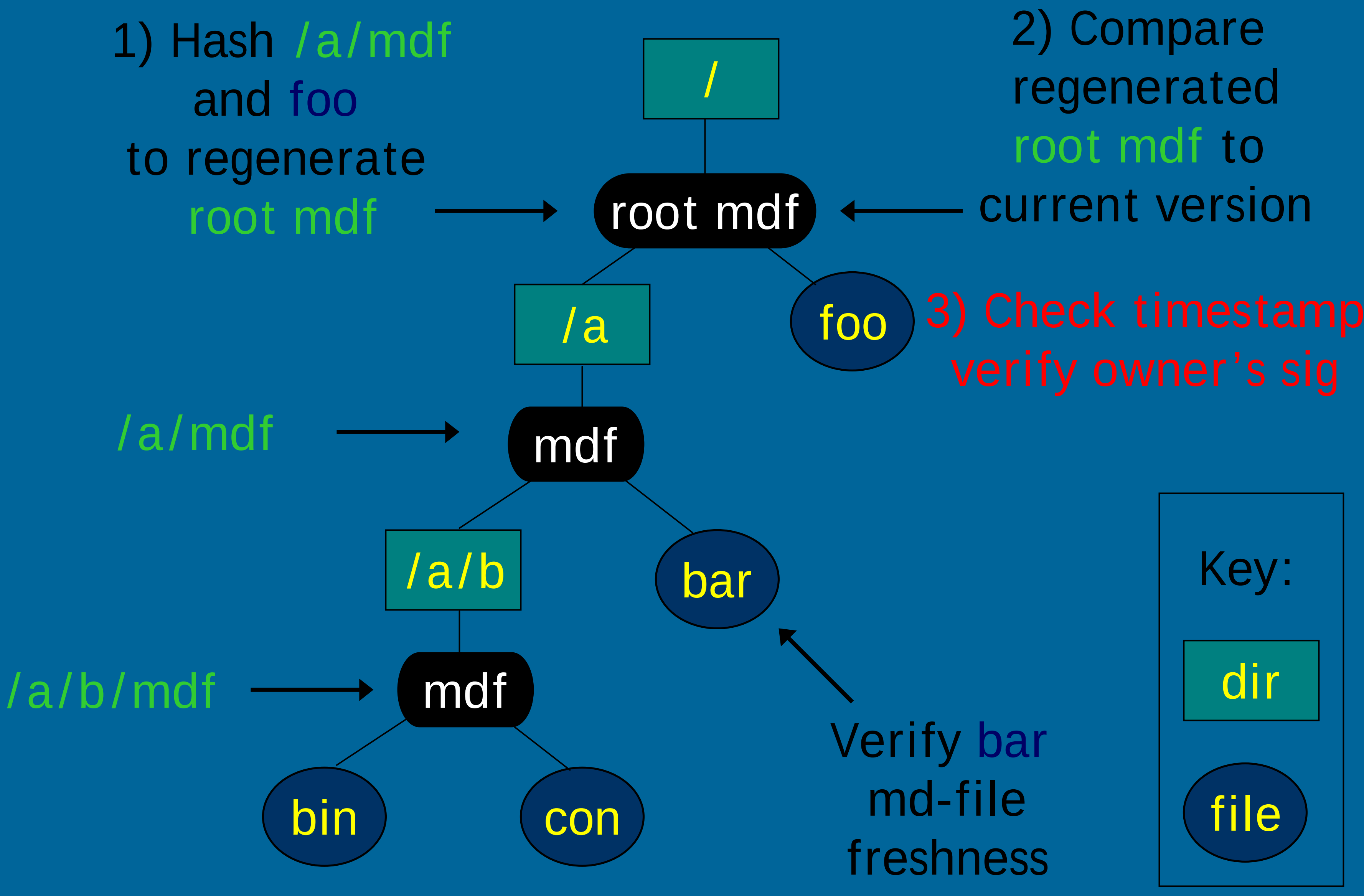
Verify md-file Freshness



Verify md-file Freshness



Verify md-file Freshness



File System Operations

1. Create, read, write, rename, unlink, share files
2. Symbolic links but no hard links
3. User access revocation

User 1 Access Revocation

Enc. Key Block (Owner)	Enc. Key Block (User 1)	File Sig. Pub. Key (FSK)	Time Stamp	File name	SIG_{MSK} [Meta Data Hash]

User 1 Access Revocation

Enc. Key Block (Owner)	Enc. Key Block (User 1)	File Sig. Pub. Key (FSK)	Time Stamp	File name	SIG_{MSK} [Meta Data Hash]

1) Regenerate new file keys

User 1 Access Revocation

Enc. Key Block (Owner)	Enc. Key Block (User 1)	File Sig. Pub. Key (FSK)	Time Stamp	File name	SIG_{MSK} [Meta Data Hash]

1) Regenerate new file keys
2) Remove user 1 key block

User 1 Access Revocation

Enc. Key Block (Owner)	File Sig. Pub. Key (FSK)	Time Stamp	File name	SIG_{MSK} [Meta Data Hash]

3) Update file sig. key and enc. key blocks

1) Regenerate new file keys

2) Remove user 1 key block

User 1 Access Revocation

Enc. Key Block (Owner)	File Sig. Pub. Key (FSK)	Time Stamp	File name	SIG_{MSK} [Meta Data Hash]

3) Update file sig. key and enc. key blocks

4) Update time stamp

1) Regenerate new file keys

2) Remove user 1 key block

User 1 Access Revocation

Enc. Key Block (Owner)	File Sig. Pub. Key (FSK)	Time Stamp	File name	SIG_{MSK} [Meta Data Hash]

3) Update file sig. key and enc. key blocks

4) Update time stamp

5) Update hash and sig

1) Regenerate new file keys

2) Remove user 1 key block

User 1 Access Revocation

Enc. Key Block (Owner)	File Sig. Pub. Key (FSK)	Time Stamp	File name	SIG_{MSK} [Meta Data Hash]

3) Update file sig. key and enc. key blocks

4) Update time stamp

5) Update hash and sig

1) Regenerate new file keys

2) Remove user 1 key block

6) Update freshness hash tree

User 1 Access Revocation

Enc. Key Block (Owner)	File Sig. Pub. Key (FSK)	Time Stamp	File name	SIG_{MSK} [Meta Data Hash]

3) Update file sig. key and enc. key blocks

4) Update time stamp

5) Update hash and sig

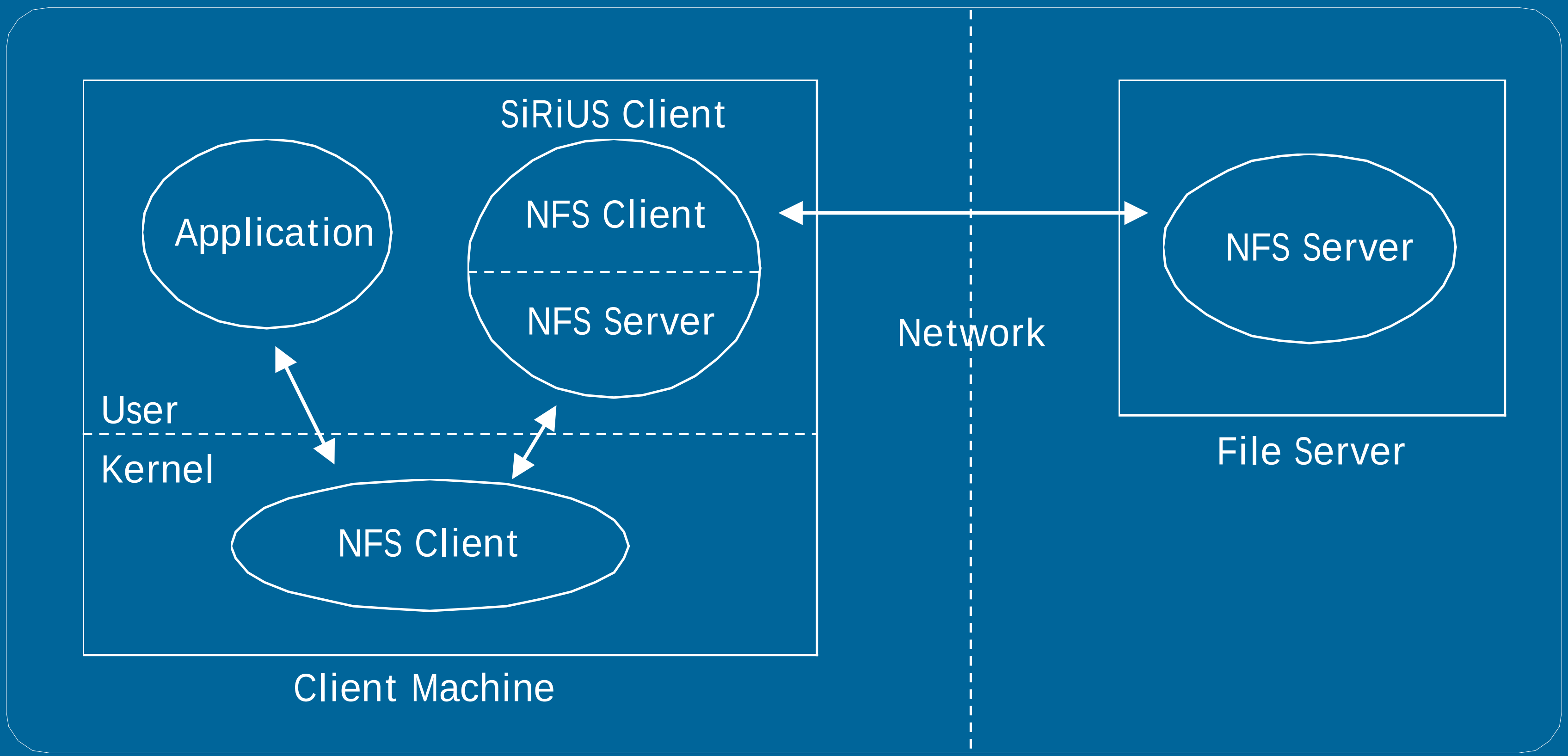
1) Regenerate new file keys

2) Remove user 1 key block

6) Update freshness hash tree

7) reencrypt file data

Architecture



SiRiUS layered over NFS using SFS toolkit

Implementation Details

- Multiplex incoming NFS requests into multiple outgoing NFS requests
- NFS file handle cache
- Changing file access controls
- Random access
  - Essential for good performance in partial file reads/writes

Random Access

Existing crypto file systems that support random access either

1. use block storage servers (SUNDR)
2. don't encrypt data on server (SFS)

Method:

1. View file as a series of blocks
2. Hash tree for file integrity

Similar construction used for authenticating digital streams – Wong and Lam (1998).

Performance

1. Public key encryption - RSA-1024
2. Signatures - DSA-512
3. Data file encryption - AES-128
4. Linux 2.4

   NFS server - 1.13 GHz P3

   NFS client - 866 MHz P3-M

   100 Mbps link

Performance

Test	File Size	Kernel NFS	DumbFS	SiRiUS
Create File	0	0.4	3.4	14.5
Delete File	0	0.3	0.4	1.1
Seq. Read	8 KB	0.9	1.4	18.0
Seq. Write	8 KB	1.1	2.0	21.9
Seq. Read	1 MB	96.7	97.8	223.8
Seq. Write	1 MB	100.0	102.7	632.9

Times are in milliseconds.

Other Extensions

- Encrypted path names
- Large scale group sharing using NNL broadcast encryption
- Maintaining traditional file system semantics using union mounts
- Union mounts also solve d-file rollback