Horus: Fine-Grained Encryption-Based Security for High Performance Petascale Storage

Ranjana Rajendran • Ethan L. Miller • Darrell D. E. Long

Storage Systems Research Center
University of California, Santa Cruz

CRIS
NSF Center for Research in Intelligent Storage

ssrc

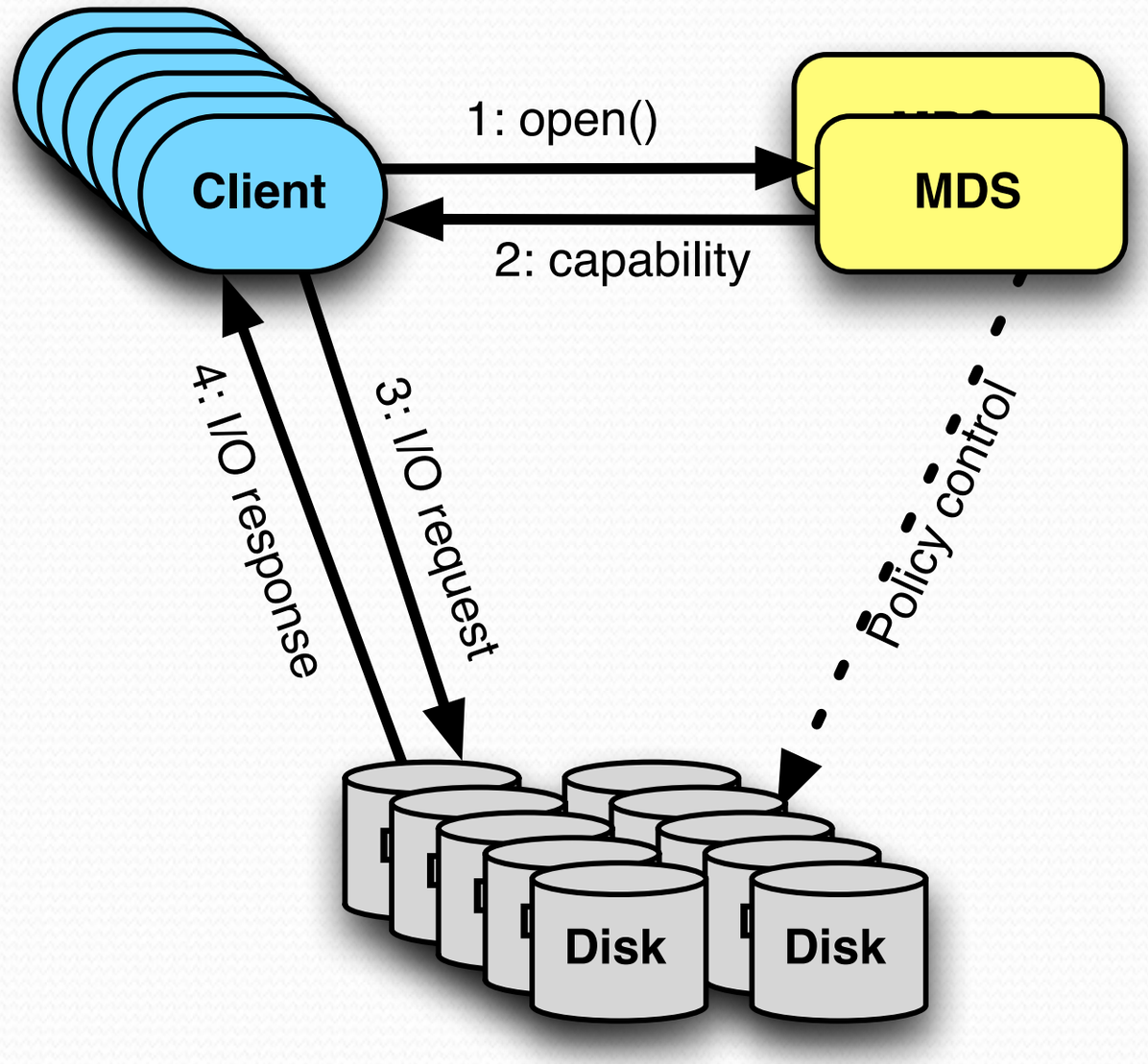
Baskin Engineering
UC SANTA CRUZ

What's the problem?

- Ever-increasing volume of data
  - More files
  - **Larger** files
- Ever-increasing **threat**
  - Intrusions
  - Insider attacks
  - Accidental data leakage
- HPC systems have a lot of vulnerabilities
  - Storage nodes
  - Metadata servers
  - Thousands of clients
- Goal: limit the risk of data leakage in an HPC system
- Goal: allow protection of *some* parts of a file

Typical HPC storage environment

- Clients interact with MDS to open files
- Clients interact directly with storage to read/write data



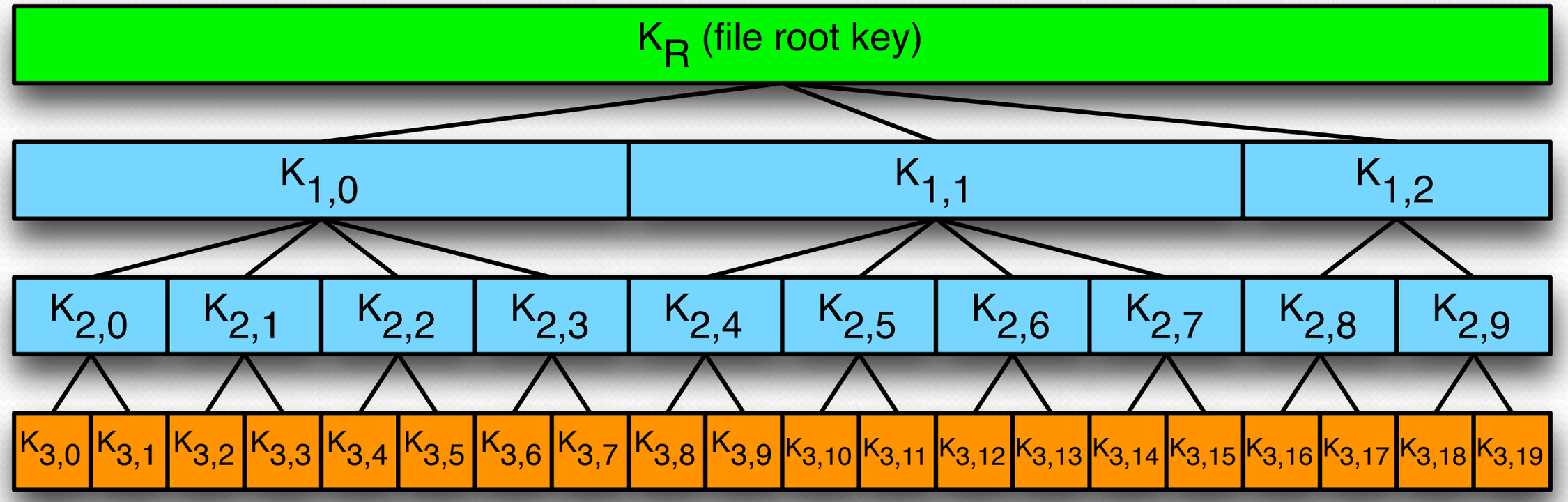
- Maat [SC07] can provide authorization
  - No encryption...

Threat model: leakage of confidential HPC data

- Traditional encryption: one key per file
  - Data can be encrypted at the client
  - Still vulnerable to leaks
- Compromised storage devices / nodes
  - Little risk if data is encrypted
  - High risk if done with other compromises
- Compromised metadata servers
  - Potential for leaking keys
  - Difficult to secure given complexity
- Compromised client (compute) nodes
  - Keys from a single client can leak the whole file!
  - There are *thousands* of clients...

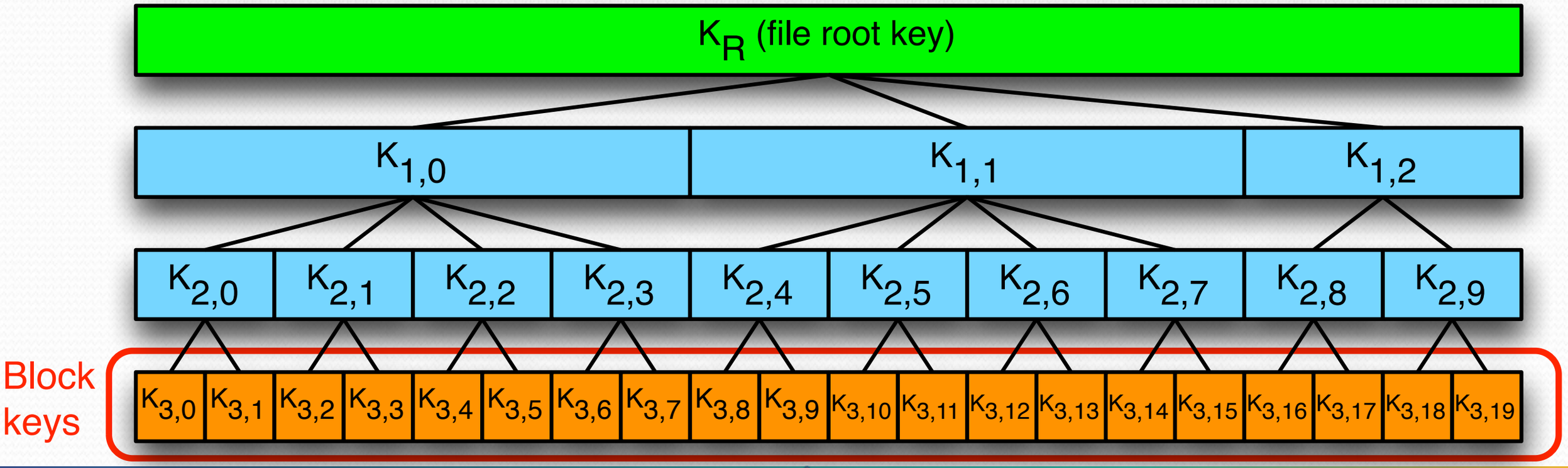
Keyed hash trees

- Solution: use keyed hash trees to generate block keys from the file key
  - Clients only get the block keys they need
  - Clients can't encrypt / decrypt data for which they don't have keys
- Nodes at any level of the tree can be given out
  - Value of a key depends on parent and key's position
  - Simple to derive block key from any key above it



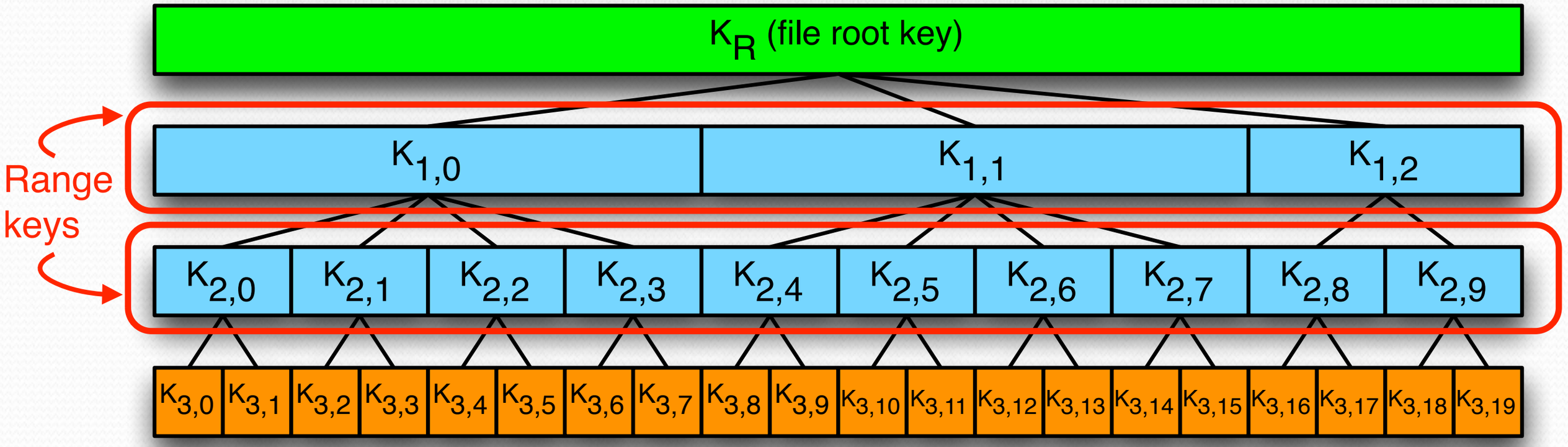
Keyed hash trees

- Solution: use keyed hash trees to generate block keys from the file key
  - Clients only get the block keys they need
  - Clients can't encrypt / decrypt data for which they don't have keys
- Nodes at any level of the tree can be given out
  - Value of a key depends on parent and key's position
  - Simple to derive block key from any key above it

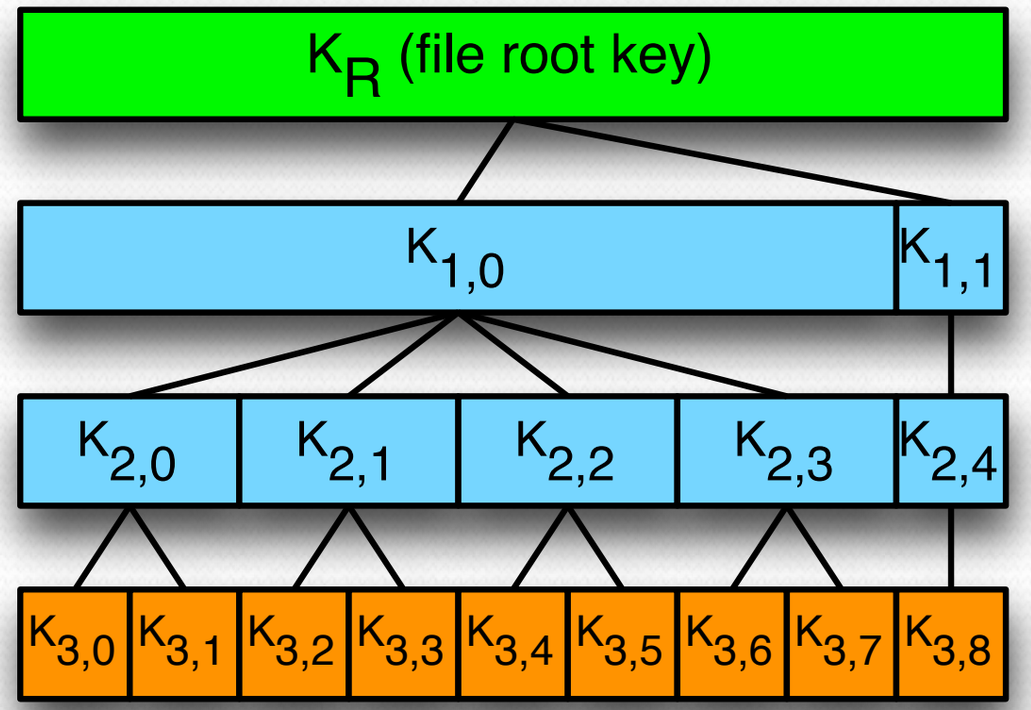


Keyed hash trees

- Solution: use keyed hash trees to generate block keys from the file key
  - Clients only get the block keys they need
  - Clients can't encrypt / decrypt data for which they don't have keys
- Nodes at any level of the tree can be given out
  - Value of a key depends on parent and key's position
  - Simple to derive block key from any key above it



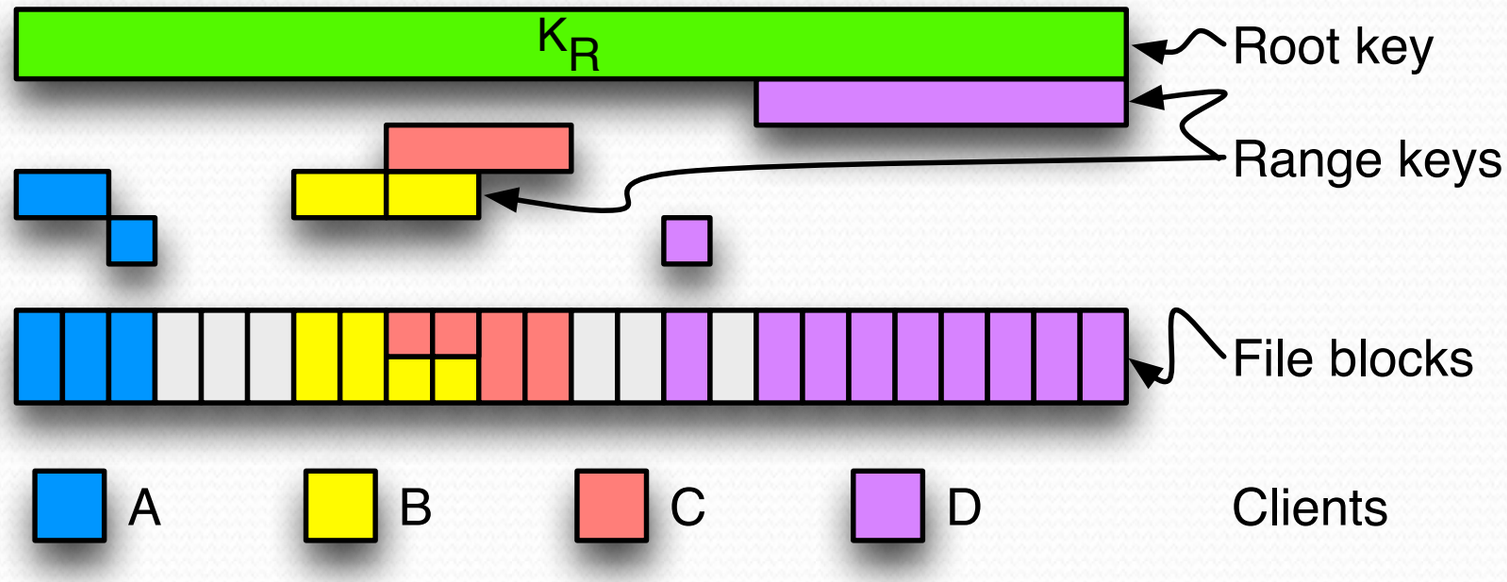
Generating block keys



for x = start + 1 **to** end **do**
  $k \leftarrow keyed\_hash(k, x \,||\, \lfloor b/B_x \rfloor)$
end for
return k

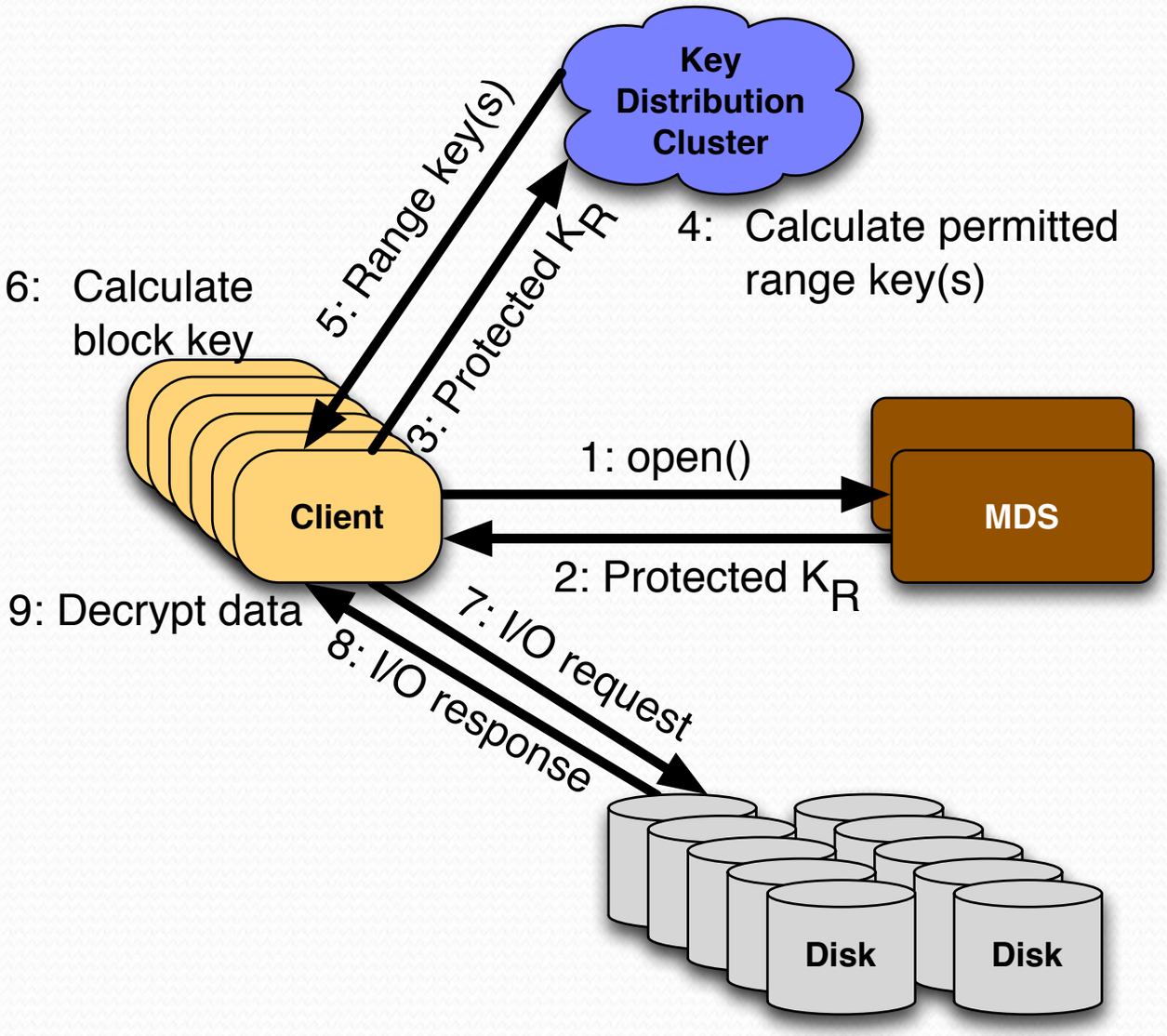
- Start at root key
- At each level, generate new key from
  - Parent key
  - Level number
  - "Offset" in the level
- Process can be split
- Simple to go down the tree
- "Difficult" to go up the tree (or sideways)

Handing out range keys



- Provide only needed range keys to each client
  - Ranges cover any number of blocks
  - Ranges must be aligned to key
    - Hand out multiple range keys to a client if needed
- Range key usage is flexible
  - Multiple clients can get key for a single block
  - Any range key that “covers” a block can be used to generate its key

Using Horus



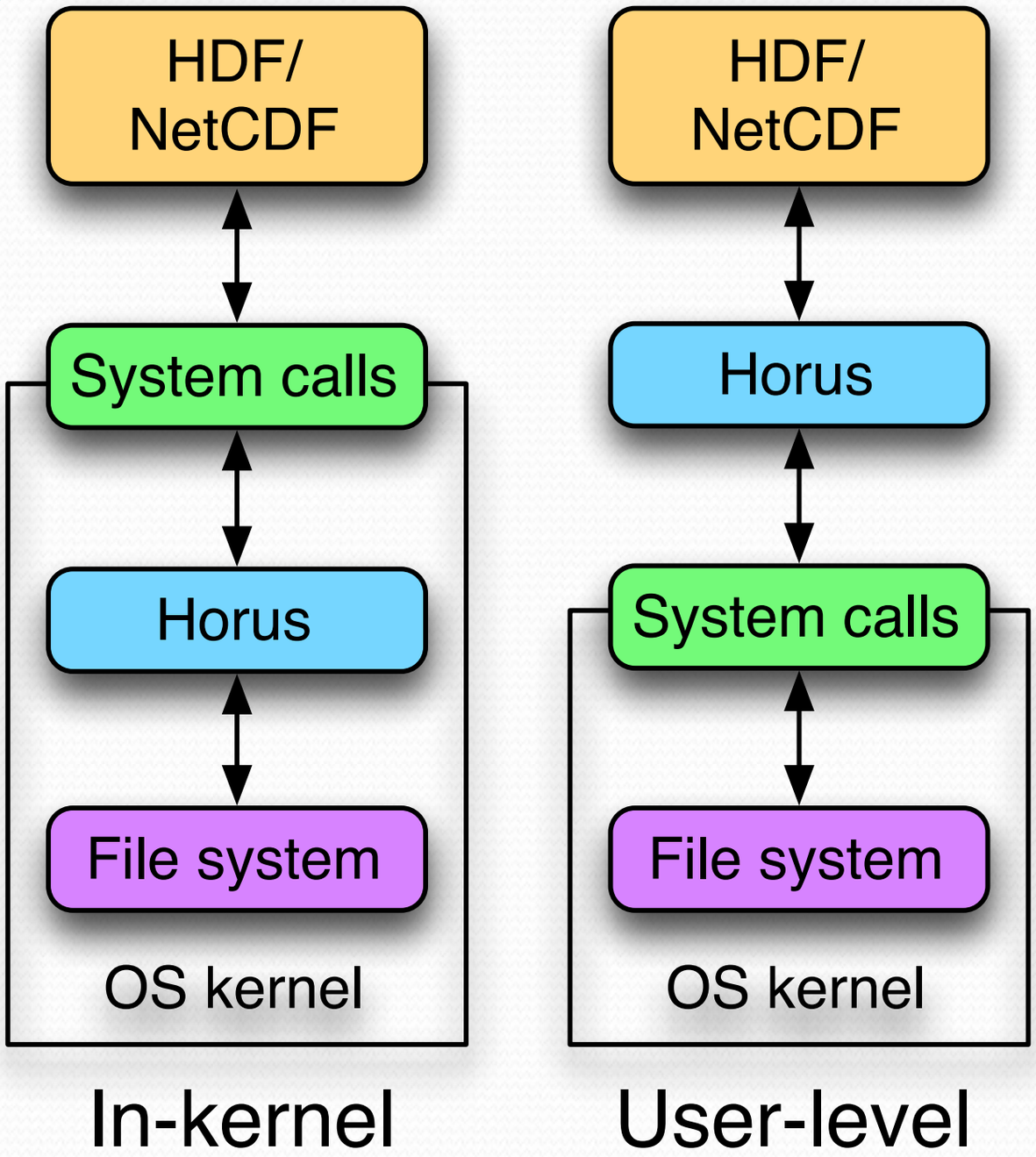
- Key Distribution Cluster can run
  - Separately
    - Stateless: easier to reset between computations
  - On MDS
  - On nodes doling out work units for computation
- Keys stored using public-key encryption
  - Client forwards key to KDC
  - KDC could request key from MDS directly

Storing file root keys

- Encrypt file root keys with users' public keys
  - Lockbox structure similar to those used in many secure file systems
- Store file root keys in the file system
  - In a separate file
  - In extended attributes attached to the file

- Alternative approach: supply file keys as part of the setup for the computation
  - More secure?
  - May require additional infrastructure

Using Horus as an encryption layer



- In-kernel implementation
  - May be a bit faster
  - Requires OS changes
- User-level implementation
  - No OS changes
  - Could leverage data layout knowledge
    - Divide file by content rather than by block offset

Horus security

- Data is only in the clear on clients
  - Storage nodes can't leak data
  - MDS can't leak data (or keys)
- Only a client can leak data
  - Client can only leak data for which it has a key
- Requires large-scale client compromise to leak the entire file
- Can't leak "idle" files without obtaining user's private key
- Revocation is an issue (as with other encrypted file systems)

Ongoing work

- User-level implementation of Horus
  - Layered just above system calls
  - Uses extended attributes for key storage
  - Includes protocol to communicate with KDC

- Explore tradeoffs between deeper tree and wider range keys

- Eventually, integrate into HPC file system such as Ceph or PVFS

Conclusions

- Security is becoming increasingly important for HPC
  - Leaving data in the clear may no longer be acceptable
- Horus prevents many attacks
  - Compromise of disks or MDS
  - Small-scale compromise of compute nodes & clients
- Horus allows sharing differential security for portions of large files
- Horus can run in the kernel or at user level

➡Provide greater confidentiality for HPC data

Questions?

Supported by

NSF

CRIS
NSF Center for Research in Intelligent Storage

U.S. DEPARTMENT OF ENERGY | Office of Science

iTRIS
Center for Information Technology Research in the Interest of Society