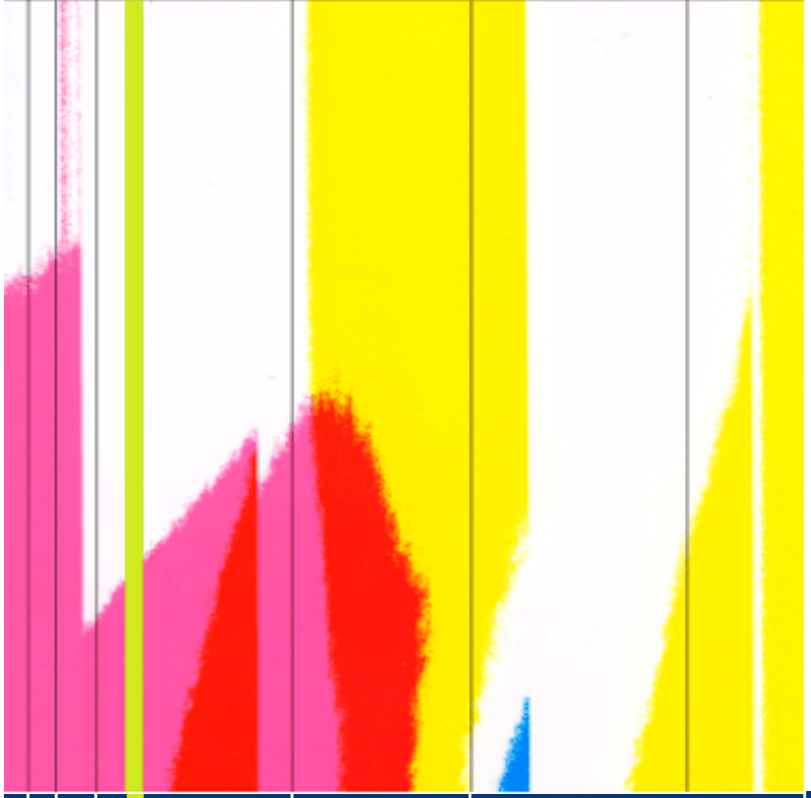


Pastis: peer-to-peer file system for persistant large-scale storage

Journée Iliatech - 14 Mai 2009

Jean-Michel Busca, Fabio Picconi, Pierre Sens, Véronique Simon

Equipe REGAL (LIP6/INRIA)

Outline

1. **P2P File Systems**

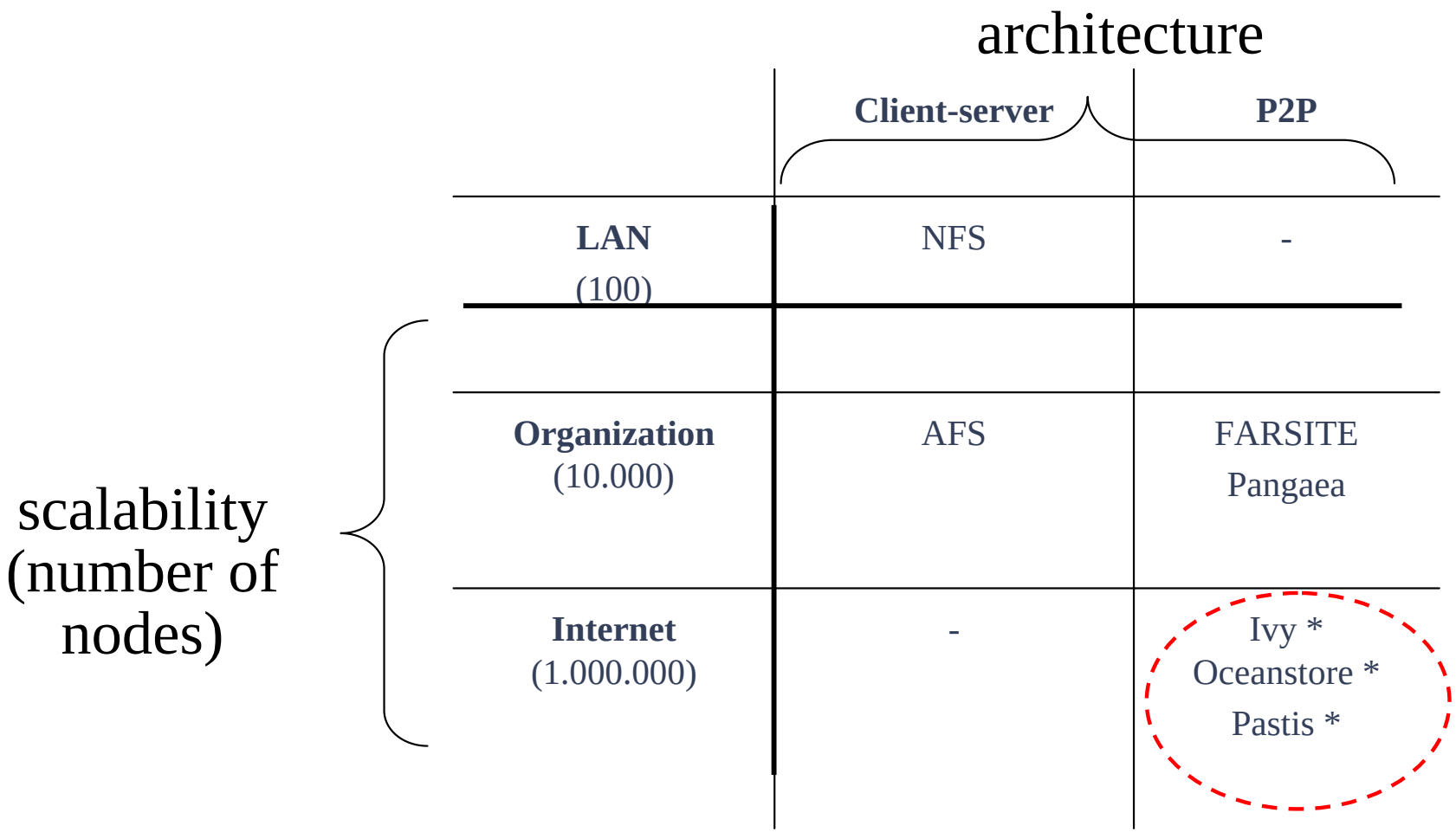
2. **Pastis**

3. **Performance evaluation**

Motivations for P2P storage

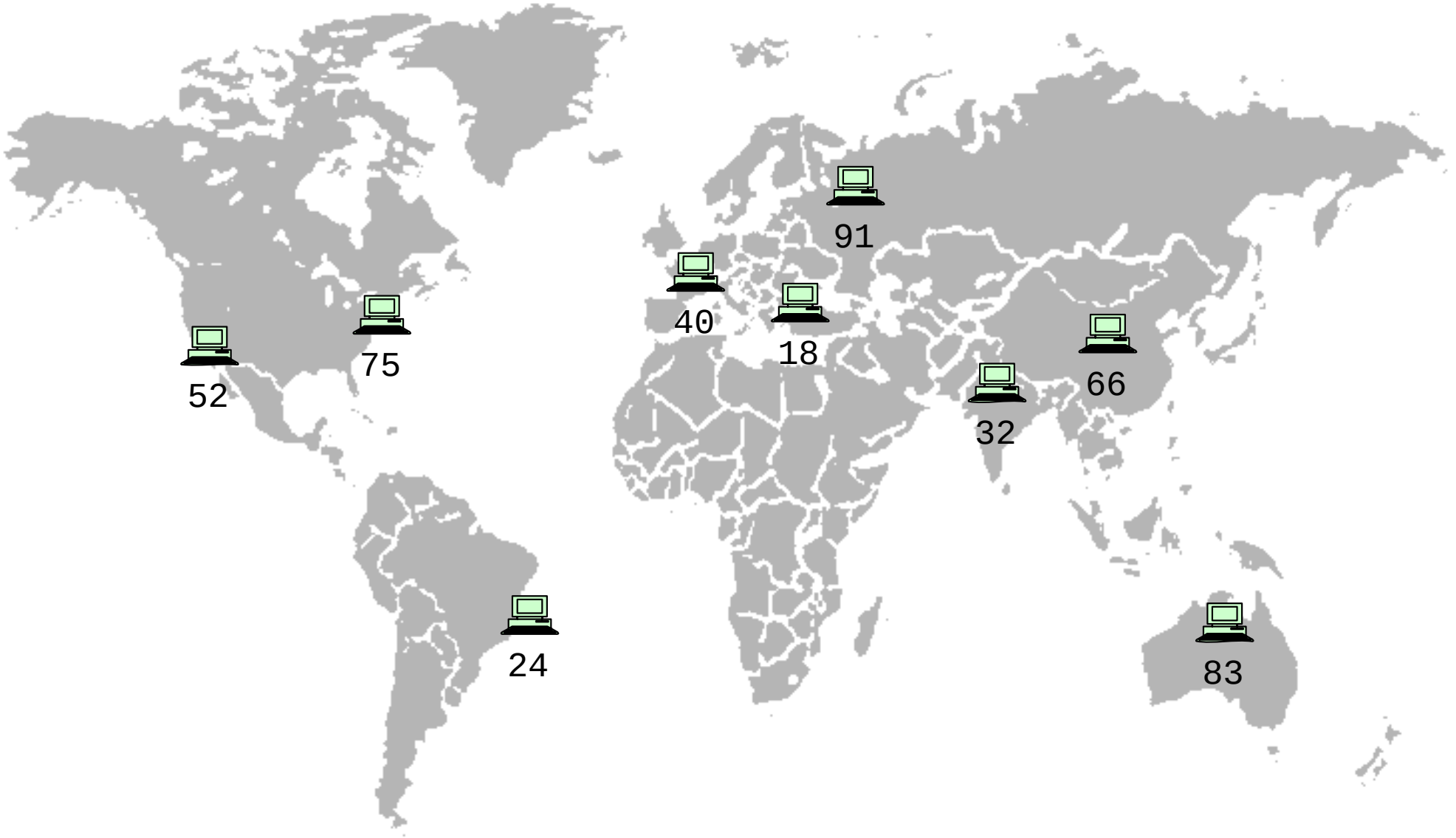
- Huge storage capacity
  - Transparently use millions of disks (set top boxes, nano-data center)
- Increase the reliability
  - Highly distribution of data
- Applications
  - Backup
  - Data sharing
  - Collaborative applications
  - Storage of large data

Distributed file systems

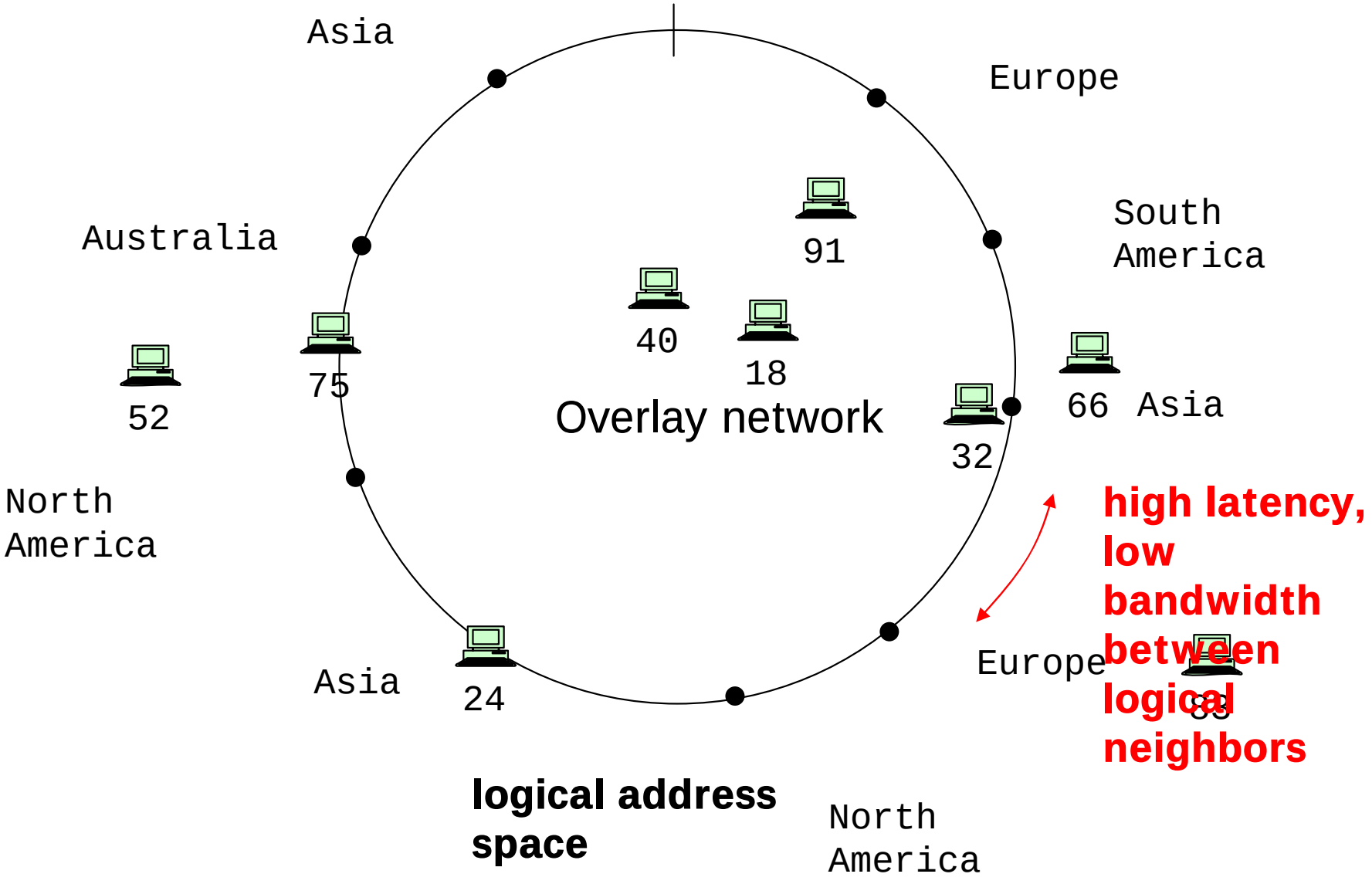


*** uses a Distributed Hash Table (DHT) to store data**

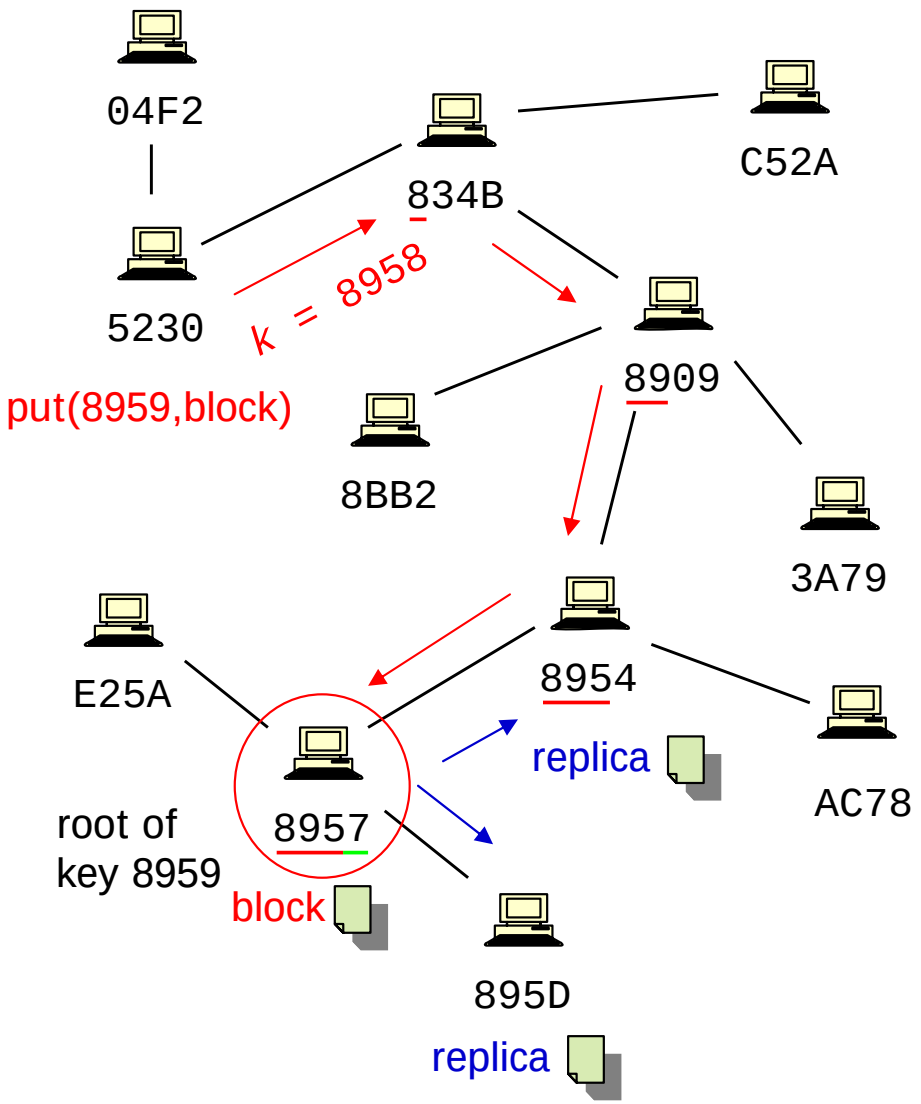
Distributed Hash Tables

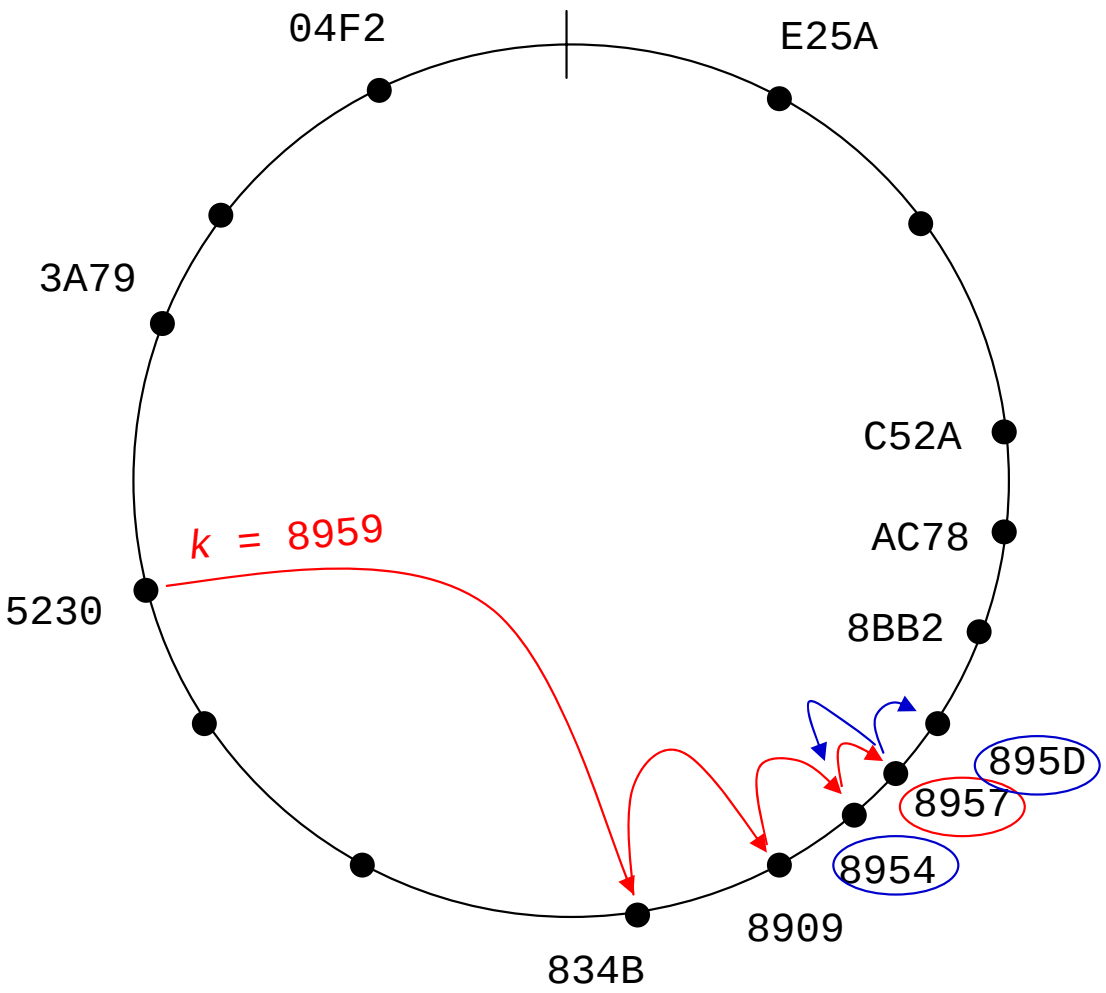


DHTs

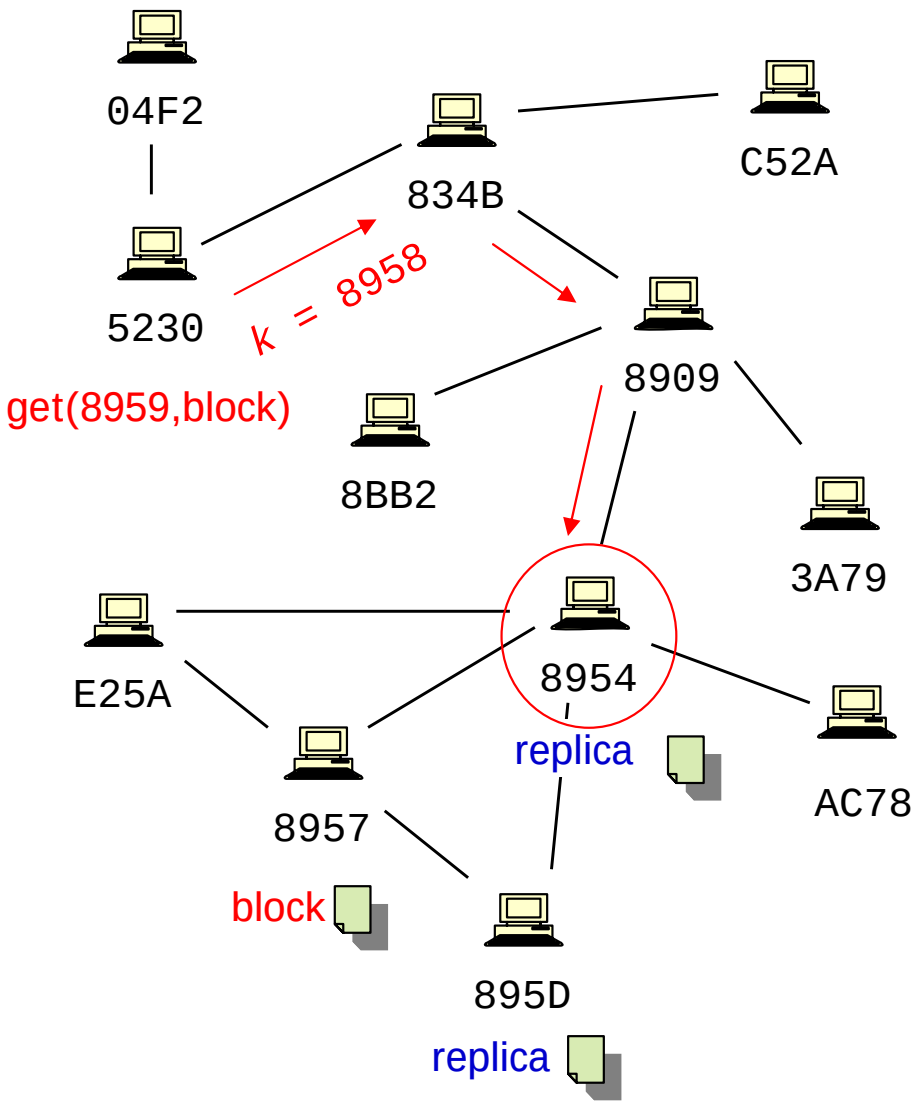


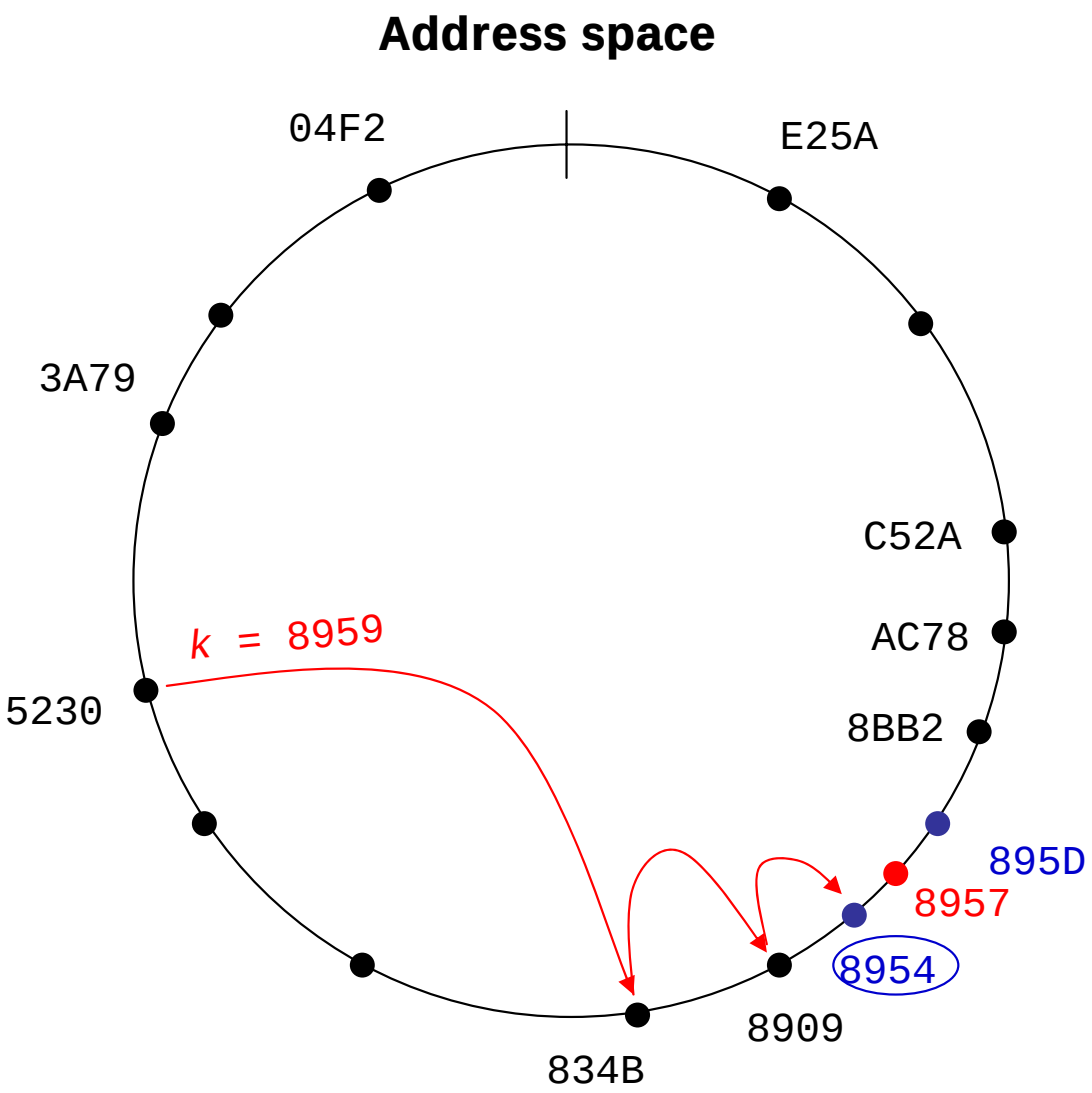
Insertion of blocks in DHT



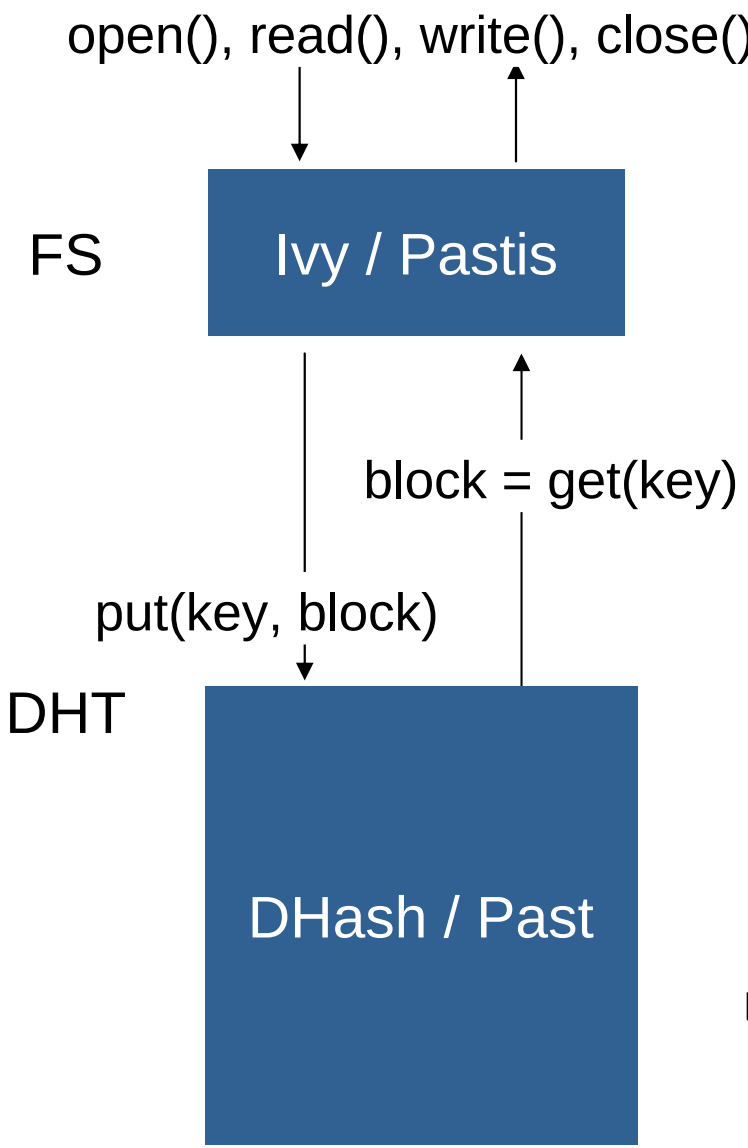


Insertion of blocks in DHT





P2P File systems architecture



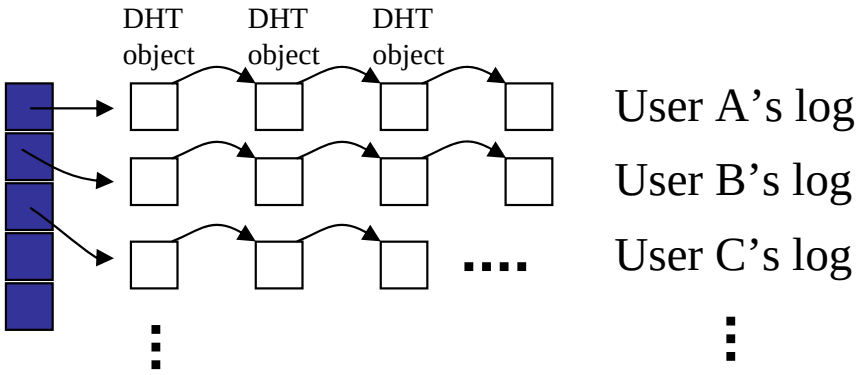
- files and directories
- read-write access semantics
- security and access control

- block store (DHT)
- message routing

- scalability
- fault-tolerance
- self-organization

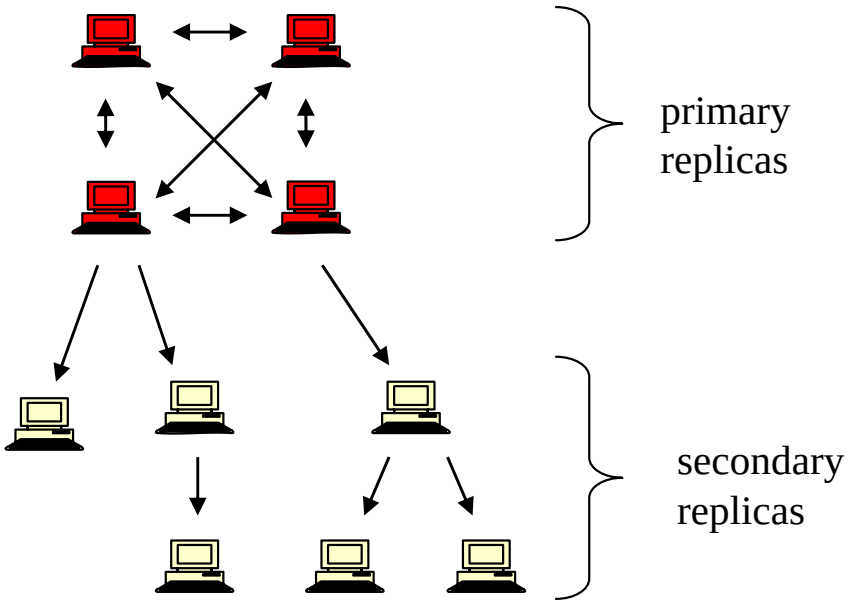
Ivy [OSDI'02]

- log-based, one log per user
- fast writes, slow reads
- limited to small number of users



Oceanstore [FAST'03]

- updates serialized by primary replicas
- partially centralized system
- BFT agreement protocol requires well-connected primary replicas

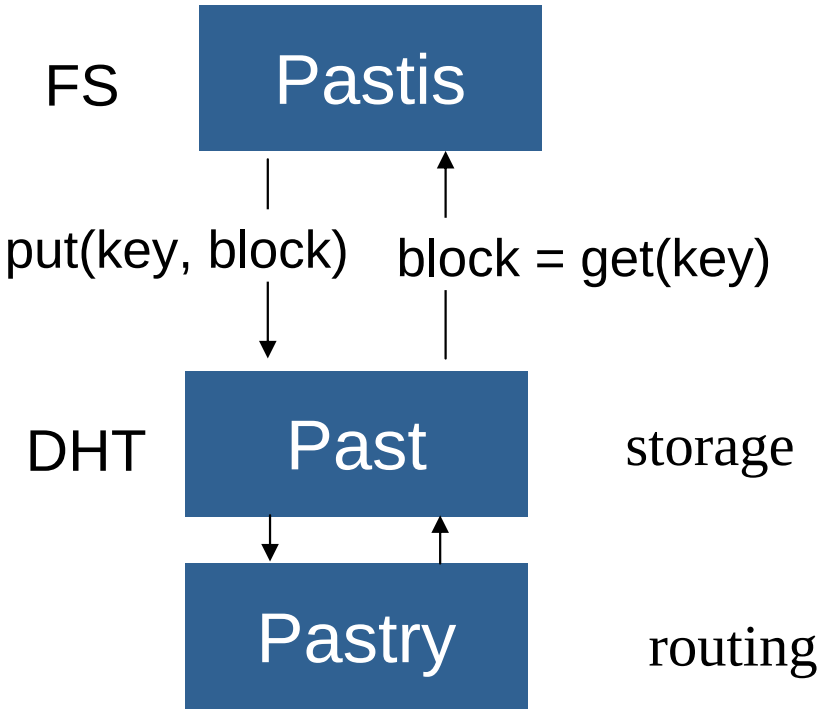


Pastis

Pastis design

Design goals

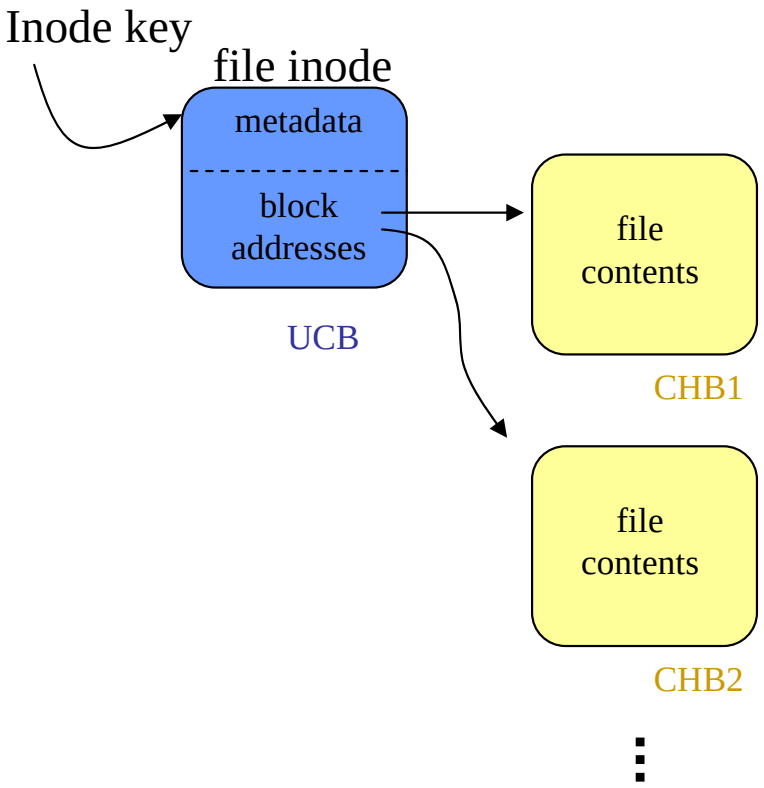
- simple
- completely decentralized
- scalable (network size and number of users)

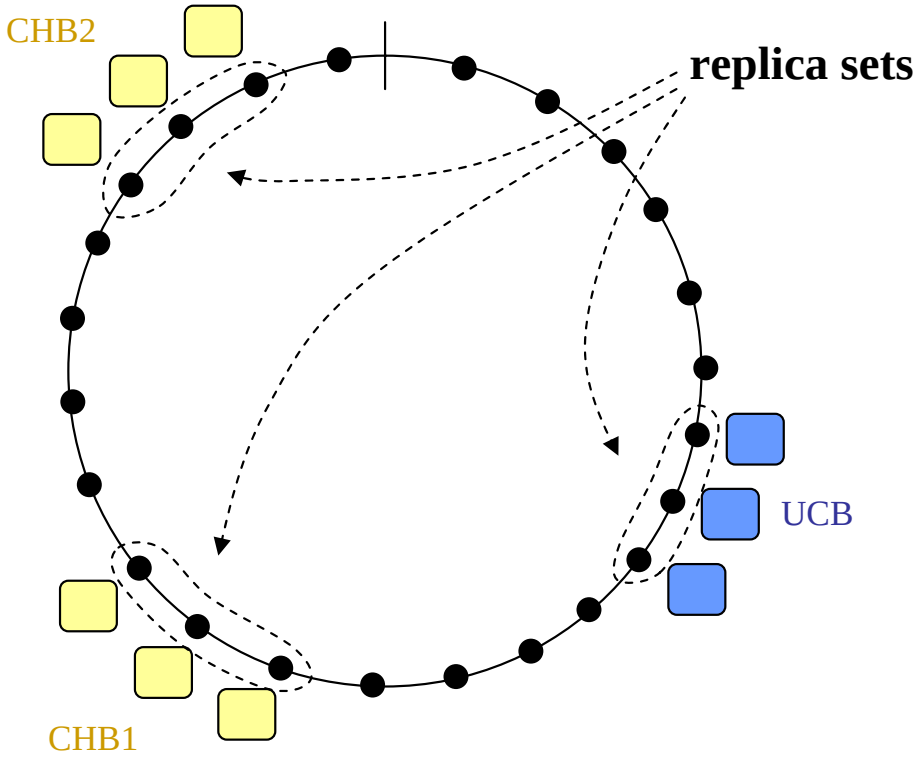


Pastis data structures

Data structures similar to the Unix file system

- *inodes* are stored in modifiable DHT blocks (UCBs)
- file contents are stored in immutable DHT blocks (CHBs)

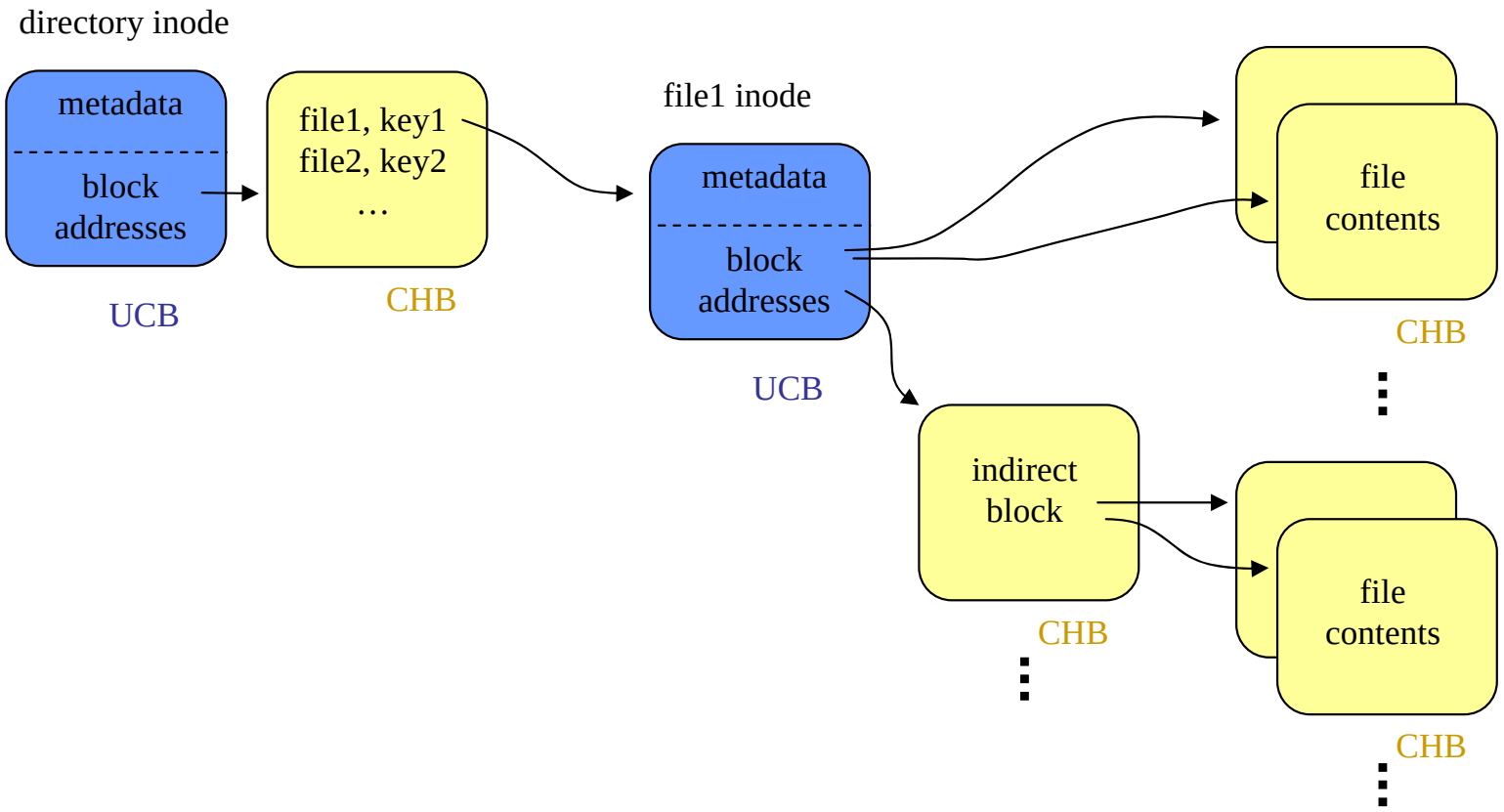




DHT address space

Pastis data structures (cont.)

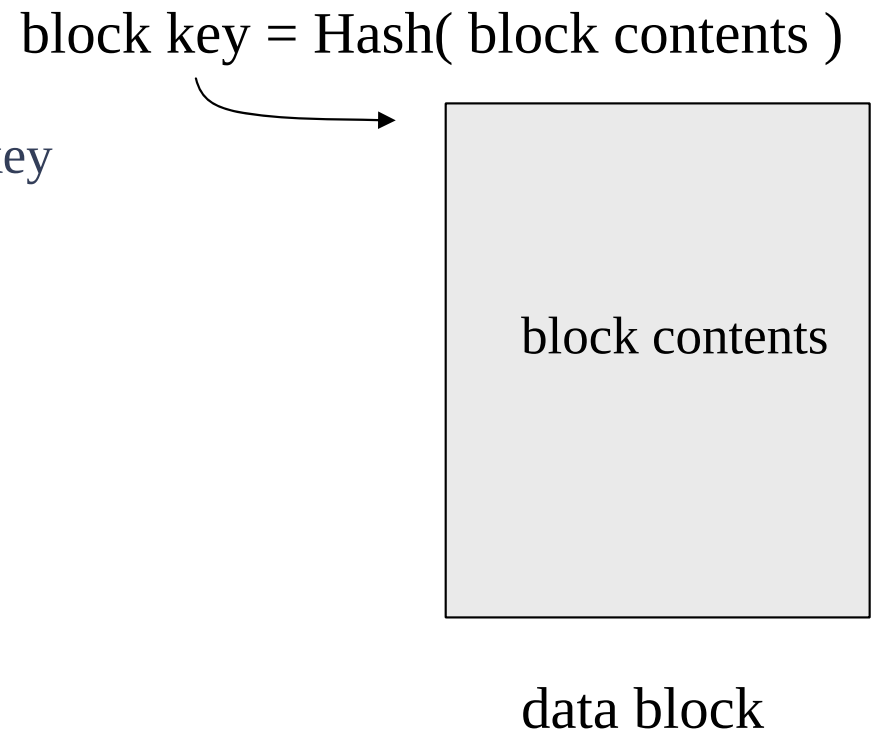
- directories contain <file name, inode key> entries
- use indirect blocks for large files



Content Hash Block

Content Hash Block

- block contents determine block key
- can detect if block is modified
- block is **immutable**



User Certificate Blocks

(KB_{pub}, KB_{priv}) associated to each block

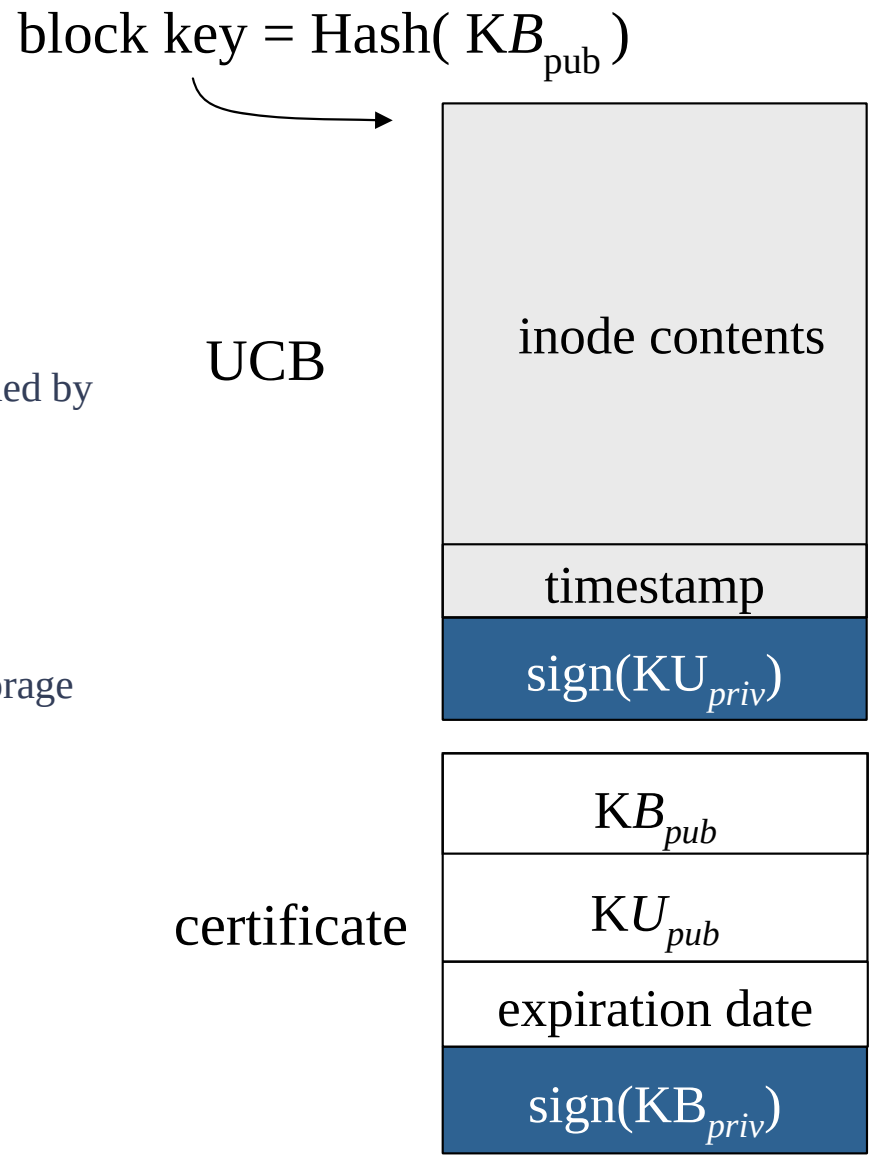
(KU_{pub}, KU_{priv}) associated to each user

Certificate

- grants *write* access to a given user (identified by KUpub)
- issued by the file owner
- expiration date allows access revocation

Authentication

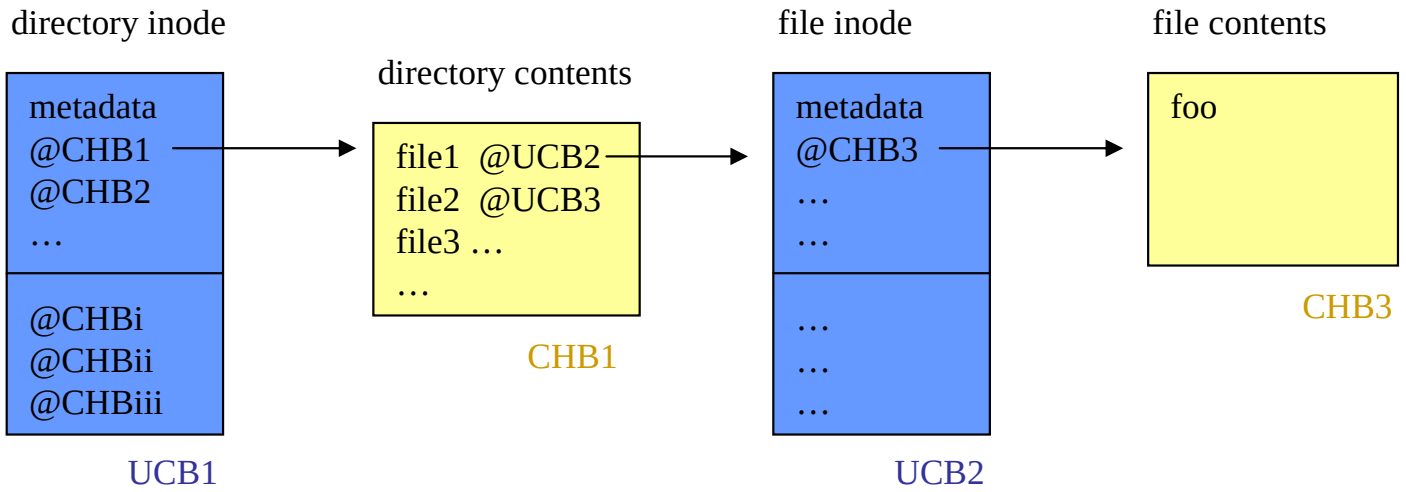
- Verify signature of certificate using the storage key (KB_{pub})
- Verify signature of UCB using the KU_{pub}



Pastis – Update handling

File update

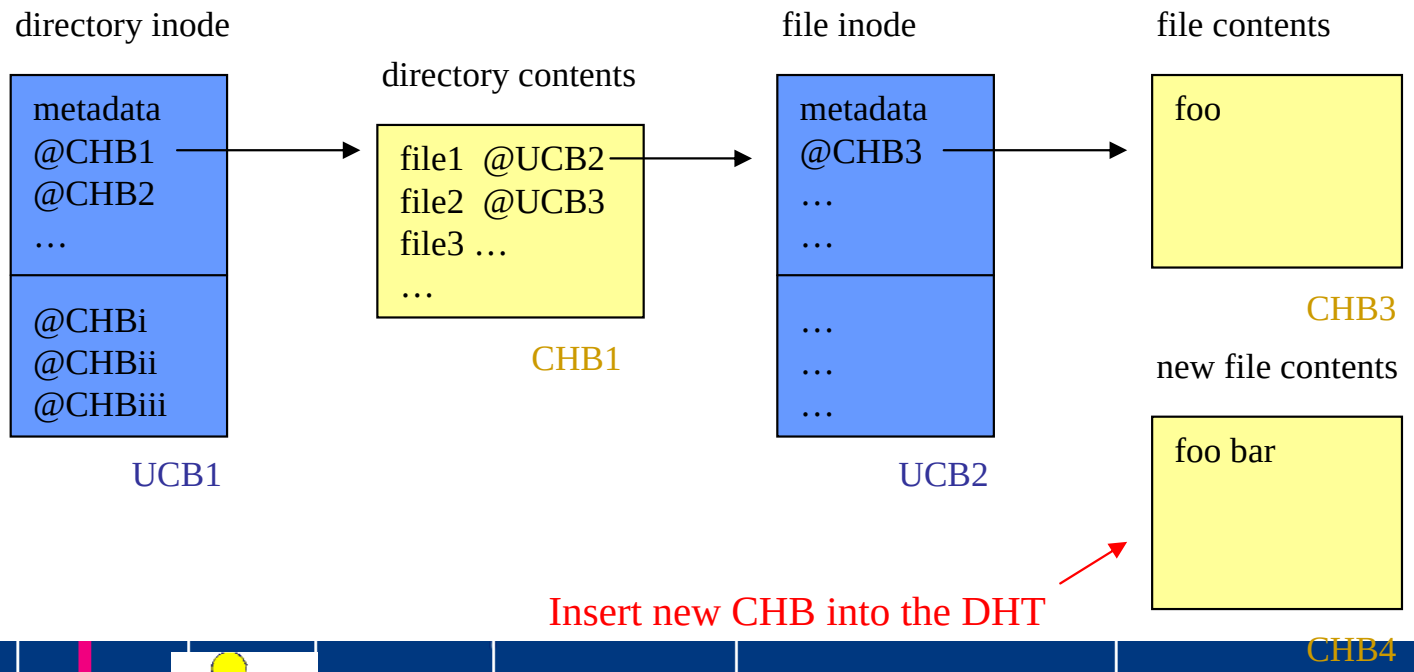
- insert the new file contents (CHBs)
- reinsert the file inode (UCB)
- replace data blocks (CHBs)



Pastis – Update handling

File update

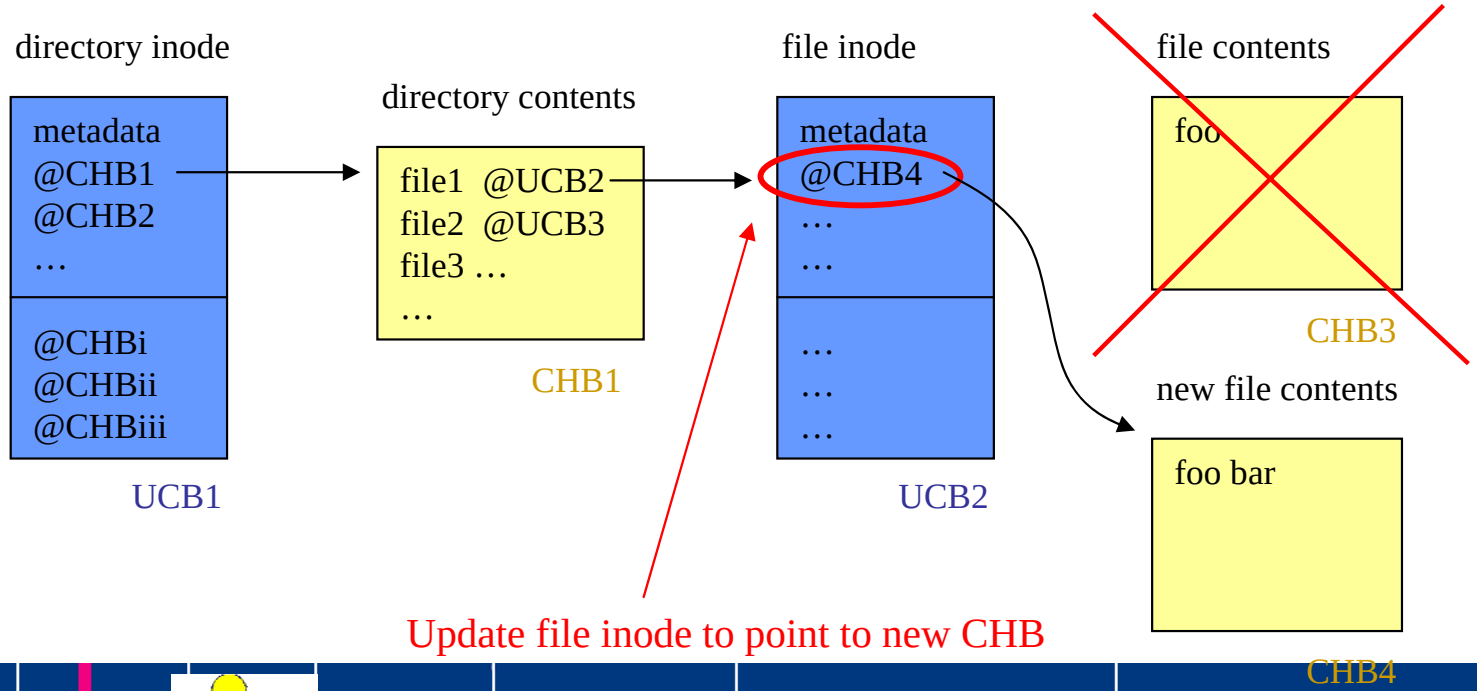
- insert the new file contents (CHBs)
- reinsert the file inode (UCB)
- replace data blocks (CHBs)



Pastis – Update handling

File update

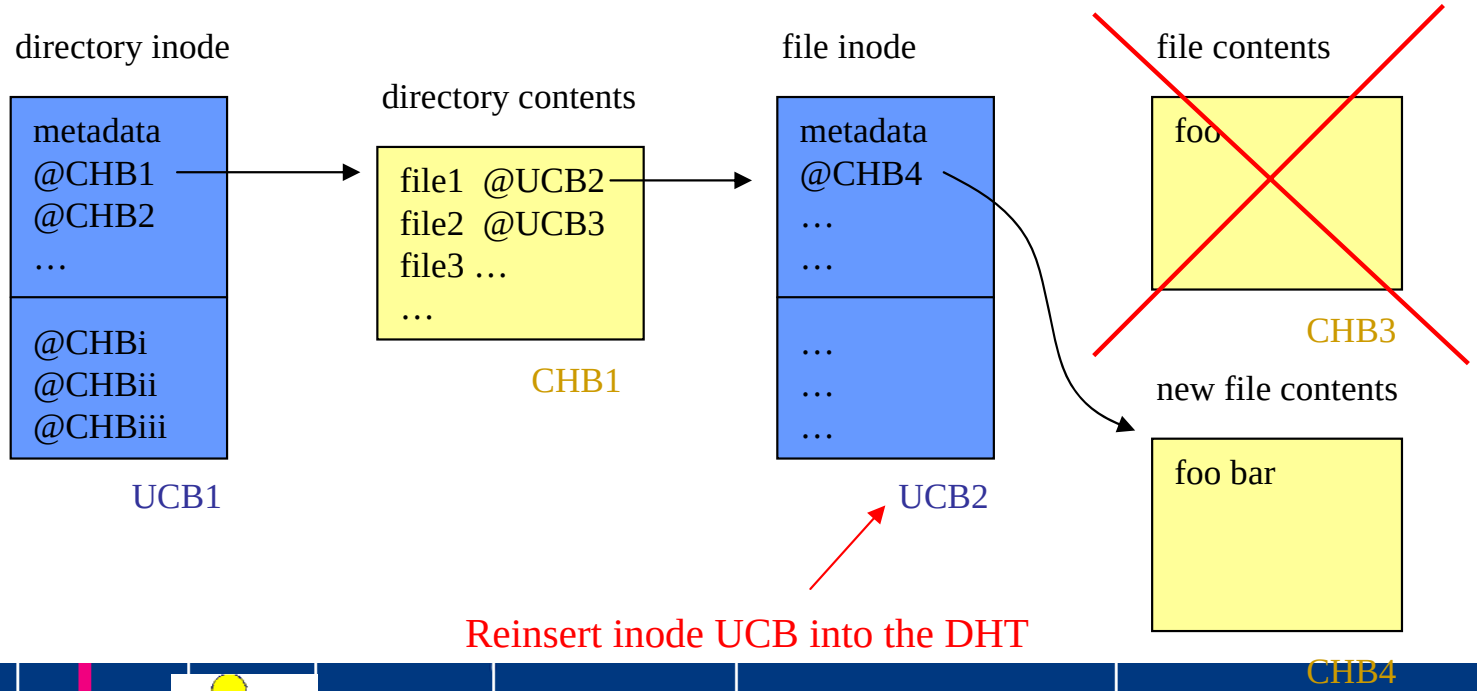
- insert the new file contents (CHBs)
- reinsert the file inode (UCB)
- replace data blocks (CHBs)



Pastis – Update handling

File update

- insert the new file contents (CHBs)
- reinsert the file inode (UCB)
- replace data blocks (CHBs)

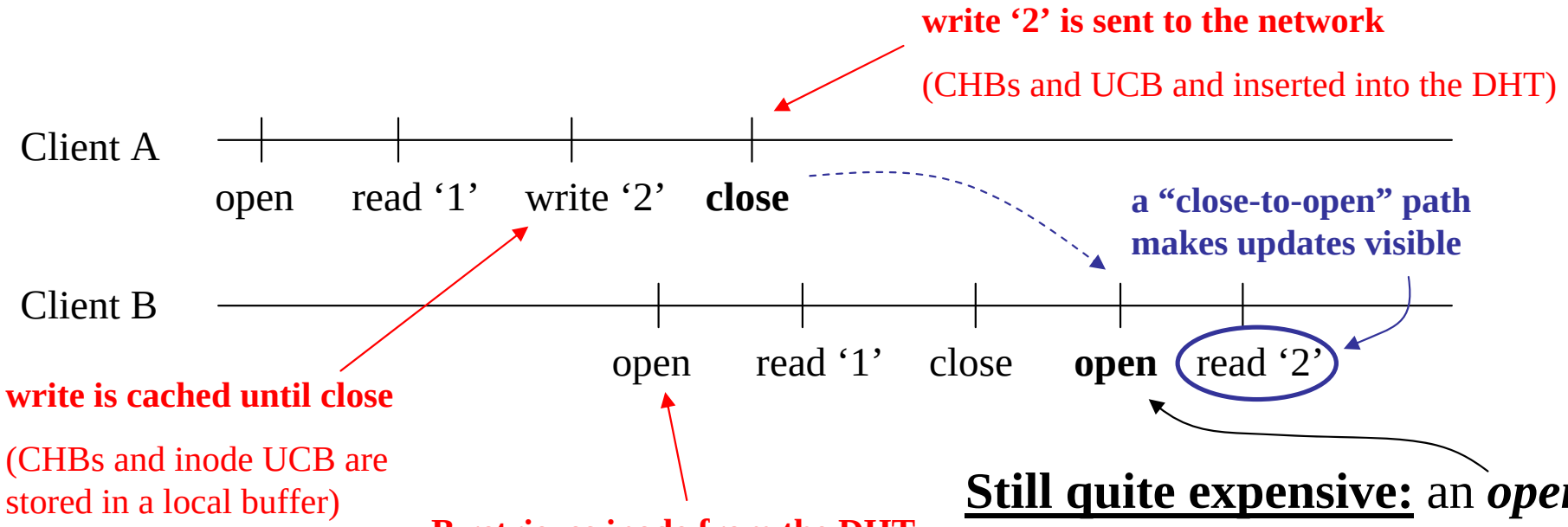


Pastis – Consistency

Strict consistency → too expensive, requires too many network accesses

***Close-to-open* consistency**

- open(): returns the latest version of the file commited by close()
- between open() and close(): user only sees his own updates
- defer writes until file is closed

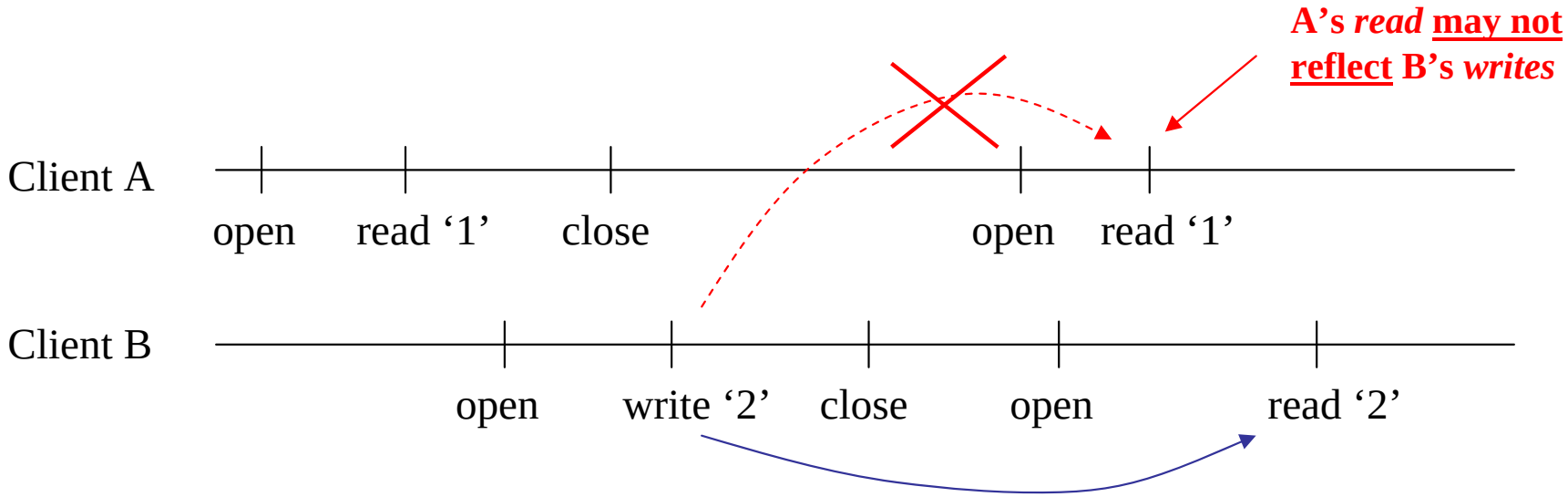


Still quite expensive: an ***open*** requires retrieving the most up-to-date inode replica

Pastis – Consistency

***Read-your-writes* consistency**

- relaxation of the *close-to-open* model
- read() must reflect previous **local** writes only
- writes from other clients **may or may not** be visible



An ***open*** does not require retrieving the most up-to-date inode replica, just fetch **one** inode replica **not older** than those accessed previously

Evaluation

Evaluation

Prototype

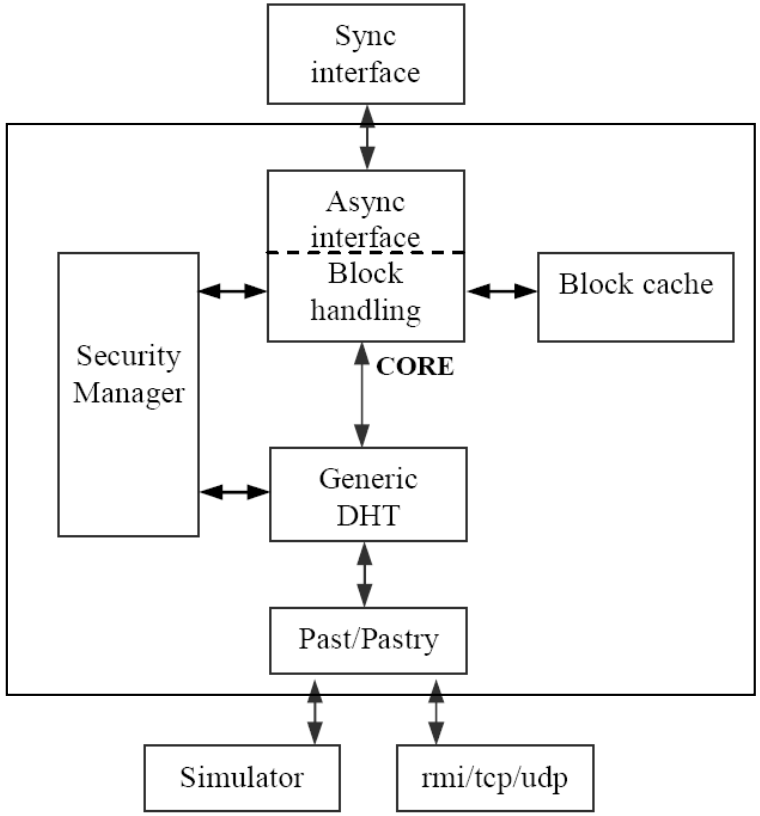
- programmed in Java
- Client interface : NFS, Fuse
- Test program: Andrew Benchmark
  - Phase 1: create subdirectories
  - Phase 2: copy files
  - Phase 3: read file attributes
  - Phase 4: read file contents
  - Phase 5: make command

Emulation

- LAN with one DHT node per machine
- DummyNet router emulates WAN latencies

Simulation

- discrete event simulator - LS^3
- simulates overlay network latency



Pastis performance with concurrent clients

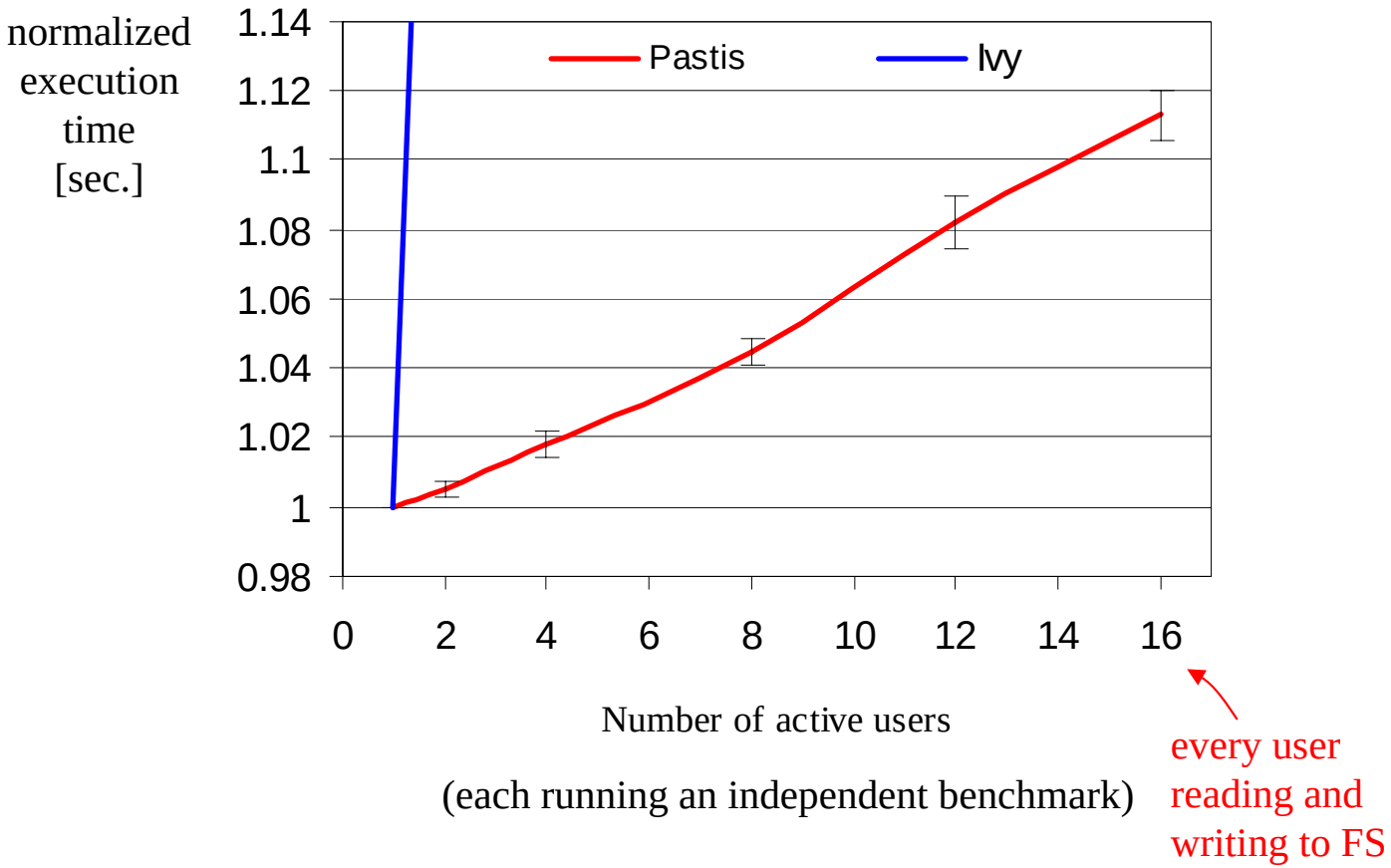
Configuration

16 DHT nodes

100 ms constant inter-node latency (Dummynet)

4 replicas per object

close-to-open consistency



Ivy's read overhead increases rapidly with the number of users

(the client must retrieve the records of more logs)

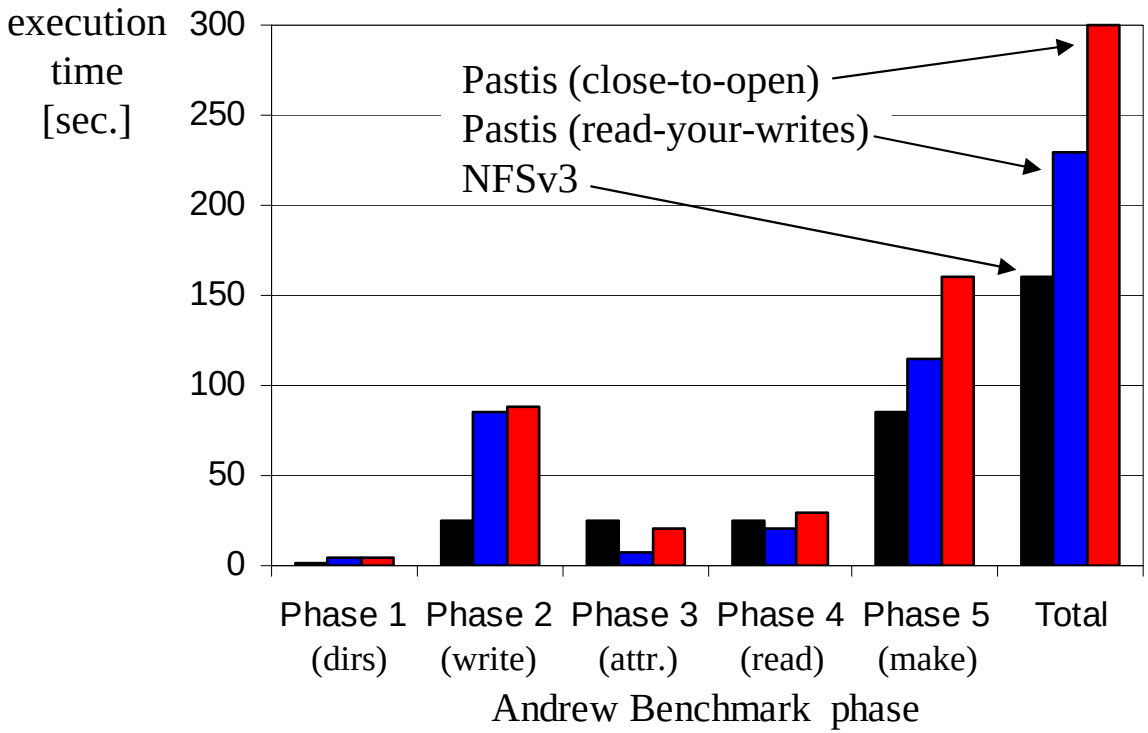
Pastis consistency models

Configuration

16 DHT nodes

100 ms constant inter-node latency (Dummynet)

4 replicas per object

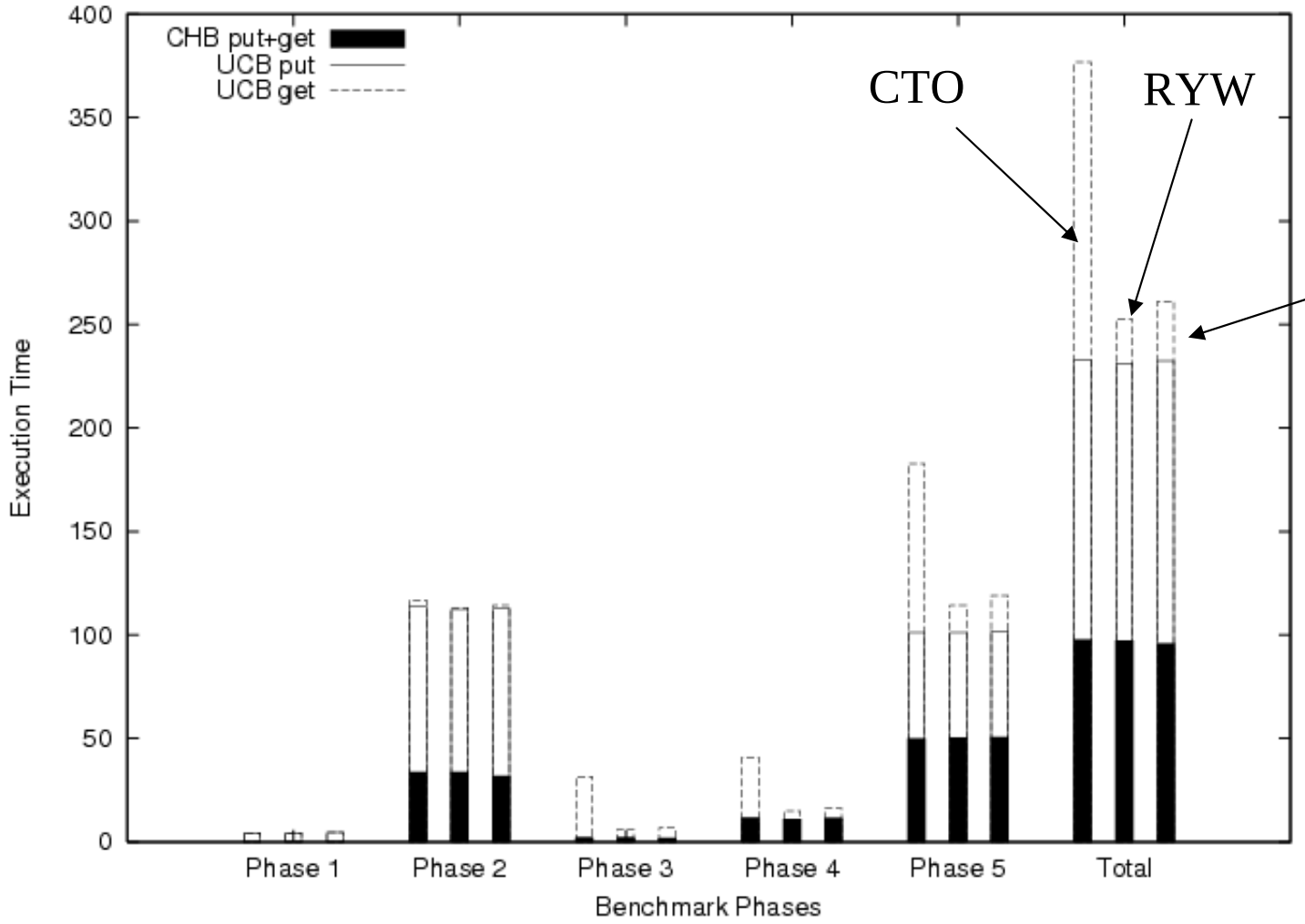


performance penalty compared to NFS (close-to-open)

Pastis	1.9
Ivy [OSDI'02]	2.0 – 3.0
Oceanstore [FAST'03]	2.55

Evaluation: consistency models

N = 32768, sphere topology, max. latency: 300 ms, k = 16



Conclusion

- Pastis
  - simple
  - completely decentralized (cf. Oceanstore)
  - scalable number of users (cf. Ivy)
  - good performance thanks to:
    - *PAST-Pastry*'s locality properties
    - relaxed consistency models (*close-to-open*, *read-your-writes*)
- Future work
  - explore new consistency models
  - flexible replica location
  - evaluation in a wide-area testbed (Planetlab)

Links

Pastis : http://gforge.inria.fr/projects/pastis/
http://regal.lip6.fr/projects/pastis

Pastry, Past : http://freepastry.rice.edu